PROGRAMMAZIONE IN ACTIONSCRIPT™ 3.0

Adobe

Programmazione in ActionScript™ 3.0

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA

Indice

Informazioni sul manuale

Questo manuale fornisce le basi per sviluppare applicazioni in ActionScript™ 3.0.
Per comprendere al meglio gli argomenti e le tecniche descritte, il lettore dovrebbe già conoscere determinati concetti generali di programmazione quali i tipi di dati, le variabili, i cicli e le funzioni, così come alcuni aspetti fondamentali della programmazione a oggetti quali le classi e l'ereditarietà. La conoscenza di ActionScript 1.0 o ActionScript 2.0 è utile ma non indispensabile.

Sommario

Uso del manuale

I capitoli del manuale sono organizzati nei seguenti gruppi logici per aiutare il lettore a individuare più facilmente le aree correlate della documentazione di ActionScript:

Capitoli	Descrizione
Capitoli da 1 a 4: panoramica della programmazione in ActionScript	Descrivono i concetti fondamentali di ActionScript 3.0: la sintassi del linguaggio, le istruzioni e gli operatori, la bozza della specifica del linguaggio ECMAScript Edizione 4, la programmazione a oggetti in ActionScript e il nuovo approccio alla gestione degli oggetti di visualizzazione nell'elenco di visualizzazione di Adobe® Flash® Player 9.
Capitoli da 5 a 10: tipi di dati e classi di base di ActionScript 3.0	Descrivono i tipi di dati di primo livello di ActionScript 3.0 che fanno anche parte della specifica del linguaggio ECMAScript.
Capitoli da 11 a 26: API di Flash Player	Descrivono varie funzioni importanti che sono implementate nei pacchetti e nelle classi specifiche di Adobe Flash Player 9: gestione degli eventi, connettività di rete e comunicazioni, input e output dei file, interfaccia esterna, modello di sicurezza dell'applicazione e altro ancora.

Questo manuale contiene anche numerosi file di esempio, che dimostrano i concetti della programmazione di applicazioni per le classi più importanti e utilizzate più comunemente. I file di esempio sono inseriti in un pacchetto, in modo che sia facile scaricarli e utilizzarli con Adobe® Flash® CS3 Professional, e possono includere file wrapper. Tuttavia, il codice di esempio principale è semplice ActionScript 3.0 che può essere utilizzato in qualsiasi ambiente di sviluppo.

ActionScript 3.0 può essere scritto e compilato in vari modi:

- Utilizzando l'ambiente di sviluppo Adobe Flex Builder 2
- Utilizzando qualunque editor di testo e compilatore della riga di comando, come quello fornito con Flex Builder 2
- Utilizzando lo strumento di creazione Adobe® Flash® CS3 Professional

Per ulteriori informazioni sugli ambienti di sviluppo di ActionScript, vedere Capitolo 1, "Introduzione ad ActionScript 3.0"

Per comprendere gli esempi di codice inclusi nel manuale non è necessario avere già esperienza nell'uso di ambienti di sviluppo integrati per ActionScript, quali Flex Builder o lo strumento di creazione di Flash. Tuttavia, è utile fare riferimento alla documentazione di questi programmi per imparare a utilizzarli per scrivere e compilare codice ActionScript 3.0. Per ulteriori informazioni, vedere "Accesso alla documentazione di ActionScript" a pagina 17.

Accesso alla documentazione di ActionScript

Poiché il manuale si concentra sulla descrizione di ActionScript 3.0, che è un linguaggio di programmazione a oggetti articolato e potente, non tratta in modo dettagliato il processo di sviluppo delle applicazioni o il flusso di lavoro all'interno di un programma o di un'architettura server particolare. Pertanto, oltre al manuale *Programmazione in ActionScript 3.0*, è opportuno consultare altre fonti di documentazione nel corso del lavoro di progettazione, sviluppo, prova e distribuzione delle applicazioni ActionScript 3.0.

Documentazione di ActionScript 3.0

Questo manuale consente di acquisire familiarità con gli aspetti principali del linguaggio di programmazione ActionScript 3.0 e fornisce dettagli ed esempi di implementazione, che illustrano le funzioni più importanti del linguaggio. Tuttavia, questo manuale non è un dizionario completo di tutti gli elementi del linguaggio. Per informazioni esaustive, consultare la *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*, che descrive ogni classe, metodo, proprietà ed evento del linguaggio. La *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0* contiene informazioni di riferimento dettagliate relative al linguaggio di base, ai componenti Flash (nei pacchetti fl) e alle API di Flash Player (nei pacchetti flash).

Documentazione di Flash

Se si utilizza l'ambiente di sviluppo Flash, è utile consultare i manuali seguenti:

Manuale	Descrizione
Uso di Flash	Descrive come sviluppare applicazioni Web dinamiche nell'ambiente di creazione di Flash
Programmazione in ActionScript 3.0	Descrive l'uso specifico del linguaggio ActionScript 3.0 e dell'API principale di Flash Player

Manuale	Descrizione
Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0	Fornisce esempi di sintassi, uso e codice dei componenti Flash e dell'API di ActionScript 3.0
Uso dei componenti ActionScript 3.0	Illustra in dettaglio l'uso dei componenti per sviluppare le applicazioni Flash
Apprendimento di ActionScript 2.0 in Adobe Flash	Contiene una panoramica della sintassi di ActionScript 2.0 e informazioni sull'uso di ActionScript 2.0 a seconda del tipo di oggetti con cui si lavora
Guida di riferimento di ActionScript 2.0	Fornisce esempi di sintassi, uso e codice dei componenti Flash e dell'API di ActionScript 2.0
Uso dei componenti ActionScript 2.0	Illustra in dettaglio come utilizzare i componenti di ActionScript 2.0 per sviluppare applicazioni Flash
Guida di riferimento dei componenti ActionScript 2.0	Descrive ogni componente disponibile nella versione 2 di Adobe Component Architecture e la relativa API
Estensione di Flash	Descrive gli oggetti, i metodi e le proprietà disponibili nell'API JavaScript
Guida introduttiva di Flash Lite 2.x	Illustra come utilizzare Adobe® Flash® Lite™ 2.x per sviluppare applicazioni e fornisce esempi di sintassi, uso e codice per le funzioni ActionScript disponibili con Flash Lite 2.x
Sviluppo di applicazioni Flash Lite 2.x	Spiega come sviluppare applicazioni Flash Lite 2.x
Introduzione ad ActionScript per Flash Lite 2.x	Introduce allo sviluppo di applicazioni con Flash Lite 2.x e descrive tutte le funzioni ActionScript disponibili per gli sviluppatori Flash Lite 2.x
Guida di riferimento di ActionScript per Flash Lite 2.x	Fornisce esempi di sintassi, uso e codice per l'API ActionScript 2.0 disponibile in Flash Lite 2.x
Guida introduttiva di Flash Lite 1.x	Fornisce un'introduzione a Flash Lite 1.x e descrive come provare il contenuto utilizzando l'emulatore Adobe® Device Central CS3
Sviluppo di applicazioni Flash Lite 1.x	Descrive come sviluppare applicazioni per i dispositivi mobili che utilizzano Flash Lite 1.x

Manuale	Descrizione
Apprendimento di ActionScript per Flash Lite 1.x	Descrive come utilizzare ActionScript nelle applicazioni Flash Lite 1.x e tutte le funzioni ActionScript disponibili in Flash Lite 1.x
Guida di riferimento di ActionScript per Flash Lite 1.x	Fornisce informazioni sulla sintassi e l'uso degli elementi ActionScript disponibili in Flash Lite 1.x

Risorse di apprendimento per ActionScript

Oltre al contenuto di questi manuali, Adobe offre articoli aggiornati periodicamente, idee per la progettazione ed esempi nel Centro per sviluppatori Adobe e nell'Adobe Design Center.

Centro per sviluppatori Adobe

Nel Centro per sviluppatori Adobe sono disponibili informazioni aggiornate su ActionScript, articoli relativi allo sviluppo di applicazioni reali e informazioni su importanti problemi emergenti. Visitare il Centro per sviluppatori all'indirizzo www.adobe.com/go/developer_it.

Adobe Design Center

Consente di ottenere le informazioni più aggiornate sulla progettazione digitale e di immagini in movimento. È possibile accedere alla produzione dei più noti artisti, scoprire le nuove tendenze di progettazione e affinare le proprie competenze con esercitazioni, flussi di lavoro di base e tecniche avanzate. Si consiglia di controllare un paio di volte al mese se sono disponibili esercitazioni e articoli aggiornati, nonché elementi di gallerie fotografiche da cui trarre ispirazione. Visitare il Design Center all'indirizzo www.adobe.com/go/designcenter_it.

CAPITOLO 1

Introduzione ad ActionScript 3.0

1

Questo capitolo fornisce una panoramica di ActionScript 3.0, la nuova e rivoluzionaria versione del linguaggio ActionScript.

Sommario

Informazioni su ActionScript

ActionScript è il linguaggio di programmazione dell'ambiente runtime di Adobe Flash Player. Rende possibili l'interattività, la gestione dei dati e molte altre operazioni relative al contenuto e alle applicazioni Flash.

ActionScript viene eseguito dall'AVM (ActionScript Virtual Machine), che fa parte di Flash Player. Il codice ActionScript viene solitamente compilato in *formato codice byte* (una sorta di linguaggio di programmazione generato e interpretato dai computer) mediante un compilatore, come quello incorporato in Adobe Flash CS3 Professional o in Adobe® Flex™ Builder™ oppure disponibile nell'SDK Adobe® Flex™ e in Flex™ Data Services. Il codice byte viene incorporato in file SWF, che vengono quindi eseguiti in Flash Player (l'ambiente runtime).

ActionScript 3.0 offre un solido modello di programmazione che risulta familiare agli sviluppatori che hanno un'esperienza di base nella programmazione a oggetti. Di seguito sono elencate alcune delle caratteristiche principali di ActionScript 3.0:

- Una nuova macchina virtuale ActionScript, chiamata AVM2, che utilizza un nuovo set di istruzioni per il codice byte e garantisce un notevole miglioramento delle prestazioni.

- Una più moderna base di codice del compilatore, più aderente allo standard ECMAScript (ECMA 262) e in grado di fornire ottimizzazioni più estese rispetto alle versioni precedenti del compilatore.
- Un'API (interfaccia di programmazione applicazioni) più estesa e migliorata, con controllo di basso livello degli oggetti e modello a oggetti effettivo.
- Un linguaggio di base fondato sulla bozza di specifica del nuovo linguaggio ECMAScript (ECMA-262) edizione 4.
- Un'API XML basata su ECMAScript for XML (E4X), come indicato nella specifica ECMA-357 edizione 2. E4X è un'estensione di ECMAScript che aggiunge XML al linguaggio come tipo di dati nativo.
- Un modello di eventi basato sulla specifica degli eventi DOM (Document Object Model) Livello 3.

Vantaggi di ActionScript 3.0

ActionScript 3.0 estende le funzionalità delle versioni precedenti del linguaggio di script ActionScript. È concepito per facilitare la creazione di applicazioni sofisticate con set di dati di grandi dimensioni e con basi di codice a oggetti e riutilizzabili. Sebbene ActionScript 3.0 non sia indispensabile per creare contenuto da eseguire in Adobe Flash Player 9, la nuova versione apre le porte a miglioramenti delle prestazioni che sono possibili solo con la nuova macchina virtuale, AVM2. Il codice ActionScript 3.0 può essere eseguito a una velocità fino a otto volte maggiore rispetto al codice ActionScript delle versioni precedenti.

La versione precedente della macchina virtuale ActionScript, AVM1, è in grado di eseguire codice ActionScript 1.0 e 2.0. AVM1 è supportata da Flash Player 9 per garantire la compatibilità retroattiva con il contenuto creato nelle versioni precedenti di ActionScript. Per ulteriori informazioni, vedere "Compatibilità con le versioni precedenti" a pagina 27.

Novità di ActionScript 3.0

Sebbene ActionScript 3.0 contenga molte classi e caratteristiche che risulteranno familiari ai programmatori ActionScript, la nuove versione del linguaggio è concettualmente e "architettonicamente" diversa da quelle precedenti. I miglioramenti introdotti in ActionScript 3.0 riguardano varie caratteristiche aggiunte al linguaggio di base e un'API di Flash Player migliorata, che offre un maggiore controllo sugli oggetti di basso livello.

Caratteristiche del linguaggio di base

Il linguaggio di base (core language) definisce i blocchi costitutivi del linguaggio di programmazione, quali le istruzioni, le espressioni, le condizioni, i cicli e i tipi. ActionScript 3.0 include molte nuove caratteristiche che consentono di velocizzare il processo di sviluppo.

Eccezioni runtime

ActionScript 3.0 segnala più condizioni di errore rispetto alle versioni precedenti del linguaggio. Le eccezioni runtime sono utilizzate per le condizioni di errore più comuni e consentono di migliorare il lavoro di debug e di sviluppare applicazioni più efficienti nella gestione degli errori. Per gli errori runtime sono disponibili tracce di stack annotate con l'indicazione del file di origine e del numero di riga, che consentono una rapida individuazione degli errori.

Tipi runtime

In ActionScript 2.0, le annotazioni di tipo fungevano principalmente da aiuto per lo sviluppatore; in fase di runtime, a tutti i valori veniva assegnato un tipo in modo dinamico. In ActionScript 3.0, le informazioni sul tipo vengono mantenute in fase di runtime e utilizzate per vari scopi. Flash Player 9 esegue la verifica runtime del tipo, aumentandone così la sicurezza per il sistema. Le informazioni sul tipo vengono utilizzate anche per indicare le variabili nelle rappresentazioni native dei sistemi, ottenendo così migliori prestazioni e una riduzione dell'uso di memoria.

Classi chiuse

ActionScript 3.0 introduce il concetto di classe chiusa (sealed). Una classe chiusa possiede unicamente le proprietà e i metodi definiti in fase di compilazione e non consente l'aggiunta di ulteriori proprietà o metodi. Questa caratteristica consente una verifica più rigorosa in fase di compilazione e permette di sviluppare programmi più solidi. Inoltre, migliora l'uso della memoria perché non è necessaria una tabella di hash interna per ogni istanza di oggetto. Le classi dinamiche possono essere specificate mediante la parola chiave `dynamic`. Tutte le classi di ActionScript 3.0 sono chiuse ("sealed") per impostazione predefinita, ma possono essere dichiarate dinamiche mediante la parola chiave `dynamic`.

Chiusure di metodo

In ActionScript 3.0 una chiusura di metodo è in grado di memorizzare la propria istanza di oggetto originale. Questa caratteristica è utile per la gestione degli eventi. In ActionScript 2.0, le chiusure di metodo non memorizzavano l'istanza di oggetto dalla quale venivano estratte e si verificava pertanto un comportamento inatteso quando una chiusura di metodo veniva richiamata. L'inconveniente veniva spesso risolto utilizzando la classe mx.utils.Delegate, che ora non è più necessaria.

ECMAScript for XML (E4X)

ActionScript 3.0 implementa ECMAScript for XML (E4X), recentemente standardizzato come ECMA-357. E4X offre una serie funzionale di costrutti di linguaggio per la gestione del codice XML. A differenza delle tradizionali API di analisi XML, in E4X XML viene elaborato come tipo di dati nativo del linguaggio. E4X sveltisce lo sviluppo di applicazioni che utilizzano il codice XML riducendo drasticamente la quantità di codice richiesta. Per ulteriori informazioni sull'implementazione di E4X in ActionScript 3.0, vedere Capitolo 11, "Operazioni con XML" a pagina 367.

Per visualizzare la specifica ECMA del linguaggio E4X, visitare il sito Web www.ecma-international.org.

Espressioni regolari

ActionScript 3.0 include il supporto nativo per le espressioni regolari e permette quindi di trovare e modificare rapidamente stringhe di codice. ActionScript 3.0 implementa il supporto delle espressioni regolari così come sono definite nella specifica del linguaggio ECMAScript edizione 3 (ECMA-262).

Spazi dei nomi

Gli spazi dei nomi (namespace) sono simili ai tradizionali specificatori di accesso utilizzati per controllare la visibilità delle dichiarazioni (`public`, `private`, `protected`). Funzionano come specificatori di accesso personalizzati, ai quali è possibile assegnare un nome a piacere. Gli spazi dei nomi includono un URI (Universal Resource Identifier) per evitare possibili conflitti e vengono utilizzati anche per rappresentare gli spazi dei nomi XML quando si lavora con il linguaggio E4X.

Nuovi tipi di base

ActionScript 2.0 prevede un unico tipo numerico, Number, che corrisponde a un numero a virgola mobile e precisione doppia. ActionScript 3.0 include i tipi int e uint. Il tipo int è un numero intero a 32 bit con segno, che consente al codice ActionScript di sfruttare la velocità di elaborazione matematica dei numeri interi della CPU. È utile nei contatori di ciclo e nelle variabili in cui sono utilizzati i numeri interi. Il tipo uint è un numero intero a 32 bit senza segno, utile per i valori di colore RGB, i conteggi di byte e altre situazioni.

Caratteristiche dell'API di Flash Player

L'API di Flash Player in ActionScript 3.0 contiene molte nuove classi che consentono di controllare gli oggetti a basso livello. L'architettura del linguaggio è completamente nuova e più intuitiva. Le classi sono troppo numerose per essere trattate in dettaglio in questa sede, pertanto le sezioni che seguono evidenziano le novità più significative.

Modello di eventi DOM3

Il modello di eventi DOM 3 (Document Object Model Level 3) fornisce una modalità standardizzata per la generazione e la gestione dei messaggi di evento, che consentono agli oggetti delle applicazioni di interagire e comunicare, mantenendo il proprio stato e reagendo ai cambiamenti. Realizzato in base alla Specifica degli eventi DOM Level 3 del World Wide Web Consortium, questo modello offre un meccanismo più semplice ed efficiente rispetto agli eventi di sistema disponibili nelle versioni precedenti di ActionScript.

Gli eventi e gli errori si trovano nel pacchetto flash.events. L'architettura dei componenti Flash utilizza lo stesso modello di eventi dell'API di Flash Player, quindi il sistema di eventi è uniforme in tutta la piattaforma Flash.

API dell'elenco di visualizzazione

L'API per l'accesso all'elenco di visualizzazione di Flash Player (la struttura che contiene tutti gli elementi visivi di un'applicazione Flash) consiste di classi che permettono di lavorare con gli elementi visivi di base in Flash.

La nuova classe Sprite è un blocco strutturale leggero, simile alla classe MovieClip ma più appropriato come classe base per i componenti dell'interfaccia utente. La nuova classe Shape rappresenta le forme vettoriali raw. Per queste classi è possibile creare "naturalmente" le istanze con l'operatore `new`, nonché riassegnare dinamicamente l'elemento principale in qualunque momento.

La gestione della profondità è ora automatica e incorporata in Flash Player, di conseguenza non è più necessaria l'assegnazione dei numeri di profondità. Sono disponibili nuovi metodi per specificare e gestire lo z-order (profondità) degli oggetti.

Gestione di dati e contenuti dinamici

ActionScript 3.0 include dei meccanismi intuitivi e uniformi nell'intera API per il caricamento e la gestione di risorse e dati nell'applicazione Flash. La nuova classe Loader fornisce un metodo unico per il caricamento di file SWF e immagini e permette di accedere a informazioni dettagliate sul contenuto caricato. La classe `URLLoader` offre un meccanismo distinto per il caricamento di testo e dati binari nelle applicazioni basate su dati, mentre la classe Socket permette di leggere e scrivere dati binari nei socket server in qualunque formato.

Accesso di basso livello ai dati

Varie API forniscono un accesso di basso livello ai dati che non era disponibile nelle versioni precedenti di ActionScript. Per i dati in corso di scaricamento, la classe URLStream (implementata da URLLoader) permette di accedere ai dati in formato raw binario mentre vengono scaricati. La classe ByteArray consente di ottimizzare la lettura, la scrittura e la gestione dei dati binari. La nuova API Sound fornisce un controllo dettagliato del suono mediante le classi SoundChannel e SoundMixer. Nuove API relative alla sicurezza forniscono informazioni sui privilegi di sicurezza di un file SWF o di un contenuto caricato, rendendo possibile una migliore gestione degli errori di sicurezza.

Operazioni con il testo

Il pacchetto flash.text di ActionScript 3.0 contiene tutte le API relative al testo. La classe TextLineMetrics fornisce dati metrici dettagliati per una riga di testo all'interno di un campo di testo; sostituisce il metodo `TextField.getLineMetrics()` di ActionScript 2.0. La classe TextField presenta nuovi metodi di basso livello interessanti, che forniscono informazioni specifiche su una riga di testo o un singolo carattere di un campo di testo. Tali metodi comprendono `getCharBoundaries()`, che restituisce un rettangolo che rappresenta il riquadro di delimitazione di un carattere, `getCharIndexAtPoint()`, che restituisce l'indice del carattere in un punto specifico, e `getFirstCharInParagraph()`, che restituisce l'indice del primo carattere di un paragrafo. I metodi a livello di riga includono `getLineLength()`, che restituisce il numero di caratteri di una riga di testo specifica, e `getLineText()`, che restituisce il testo della riga specificata. La nuova classe Font consente di gestire i caratteri incorporati nei file SWF.

Compatibilità con le versioni precedenti

Come sempre, Flash Player garantisce la totale compatibilità retroattiva con il contenuto pubblicato esistente. Qualunque contenuto che veniva eseguito nelle versioni precedenti di Flash Player può essere eseguito anche in Flash Player 9. L'introduzione di ActionScript 3.0 in Flash Player 9, tuttavia, presenta alcune problematiche relativamente all'interoperatività tra contenuto vecchio e nuovo in Flash Player 9. Di seguito sono elencati i principali problemi di compatibilità:

- Un unico file SWF non può combinare codice ActionScript 1.0 o 2.0 con codice ActionScript 3.0.
- Il codice ActionScript 3.0 può caricare un file SWF scritto in ActionScript 1.0 o 2.0, ma non può accedere alle variabili e alle funzioni del file SWF.
- I file SWF scritti in ActionScript 1.0 o 2.0 non possono caricare file SWF scritti in ActionScript 3.0. Questo significa che i file SWF creati in Flash 8 o Flex Builder 1.5 (o in versioni precedenti) non possono caricare file SWF di ActionScript 3.0.

  L'unica eccezione a questa regola è rappresentata dalla possibilità di sostituire un file SWF di ActionScript 2.0 con un file SWF di ActionScript 3.0, a condizione che non sia stato ancora caricato nulla in nessuno dei livelli del file SWF di ActionScript 2.0. Questa operazione è possibile includendo nel file SWF di ActionScript 2.0 una chiamata a `loadMovieNum()` e passando il valore 0 al parametro `level`.
- In generale, è necessario eseguire la migrazione dei file SWF scritti in ActionScript 1.0 o 2.0 se questi devono interagire con file SWF scritti in ActionScript 3.0. Ad esempio, si supponga di aver creato un lettore multimediale con ActionScript 2.0. Il lettore carica vari tipi di contenuto, a loro volta creati in ActionScript 2.0. Non è invece possibile creare nuovo contenuto in ActionScript 3.0 e caricarlo nel lettore multimediale. Per poter eseguire questo tipo di operazione, occorre effettuare la migrazione del lettore ad ActionScript 3.0.

  Al contrario, un lettore multimediale creato in ActionScript 3.0 è in grado di caricare contenuto ActionScript 2.0.

La tabella seguente riepiloga le limitazioni relative al caricamento di nuovo contenuto e all'esecuzione del codice nelle versioni precedenti Flash Player, nonché quelle relative allo scambio di script (cross-scripting) tra file SWF scritti in versioni diverse di ActionScript.

Funzionalità supportata	**Ambiente runtime**		
	Flash Player 7	Flash Player 8	Flash Player 9
Può caricare file SWF pubblicati per	7 e precedenti	8 e precedenti	9 e precedenti
Contiene questa AVM	AVM1	AVM1	AVM1 e AVM2
Esegue i file SWF scritti in ActionScript	1.0 e 2.0	1.0 e 2.0	1.0, 2.0 e 3.0

Funzionalità supportata*	**Contenuto creato in**	
	ActionScript 1.0 e 2.0	ActionScript 3.0
Può caricare contenuto ed eseguire codice nel contenuto creato in	Solo ActionScript 1.0 e 2.0	ActionScript 1.0 e 2.0, e ActionScript 3.0
Può scambiare script con contenuto creato in	Solo ActionScript 1.0 e 2.0†	ActionScript 3.0‡

* Contenuto eseguito in Flash Player 9 o versioni successive. Il contenuto eseguito in Flash Player 8 o versioni precedenti può caricare, visualizzare, eseguire e scambiare script solo in ActionScript 1.0 e 2.0.

† ActionScript 3.0 tramite Local Connection.

‡ ActionScript 1.0 e 2.0 tramite LocalConnection.

CAPITOLO 2

Guida introduttiva ad ActionScript

2

Questo capitolo offre una panoramica del linguaggio di programmazione ActionScript e pone le basi di cui l'utente ha bisogno per capire i concetti e gli esempi presentati negli altri capitoli del manuale. Il contenuto del capitolo si articola in due blocchi: la descrizione delle nozioni fondamentali sulla programmazione, con relativa applicazione nell'ambiente ActionScript, e le operazioni essenziali per organizzare e creare un'applicazione tramite ActionScript.

Sommario

Nozioni fondamentali sulla programmazione

Dal momento che ActionScript è un linguaggio di programmazione, è utile avere presenti alcuni concetti fondamentali riguardanti la programmazione.

Che cosa sono i programmi

Per iniziare, è importante avere presente, da un punto di vista concettuale, che cosa si intende per programma di un computer e che cosa fa un programma. I concetti fondamentali relativi ai programmi sono due:

- Un programma è una serie di istruzioni o procedure che il computer esegue.
- Ogni procedura consta, sostanzialmente, nella manipolazione di informazioni o dati.

Un programma per computer è un elenco di comandi dettagliati che l'utente impartisce al computer e che il computer esegue in sequenza. Ogni singolo comando si chiama *istruzione* e come vedremo più avanti, tutte le istruzioni di ActionScript terminano con un punto e virgola.

In sostanza, l'istruzione di un programma manipola dei dati che si trovano nella memoria del computer. In un caso semplice, l'utente può impartire un'istruzione che richiede al computer di sommare due numeri e di salvare il risultato in memoria. In un caso più complesso, l'utente scrive un programma che sposta un rettangolo disegnato sullo schermo in una posizione diversa. Il computer memorizza alcune informazioni sul rettangolo, come le coordinate x e y che ne descrivono la posizione attuale, le misure della base e dell'altezza, il colore e così via. Tutti questi frammenti di informazioni vengono trattenuti nella memoria del computer.
Un programma in grado di spostare il rettangolo in un punto diverso dello schermo conterrà comandi del tipo "imposta la coordinata x su 200; imposta la coordinata y su 150" (si tratta, in altre parole, di specificare quali nuovi valori il computer deve utilizzare come coordinate x e y). Ovviamente, per poter convertire questi valori nell'immagine visualizzata sullo schermo, il computer interpreta e manipola i dati forniti. Tuttavia, per il livello di approfondimento che ci siamo prefissati di raggiungere, è sufficiente sapere che l'operazione "sposta il rettangolo in un punto diverso dello schermo" consiste nel cambiare alcuni dati presenti nella memoria del computer.

Variabili e costanti

Dal momento che la programmazione consiste principalmente nella manipolazione di dati presenti nella memoria del computer, è stato escogitato un metodo per rappresentare un singolo valore all'interno di un programma. L'entità che rappresenta un valore presente nella memoria del computer si chiama *variabile*. Nel momento in cui scrive le istruzioni del programma, il programmatore inserisce il nome della variabile in luogo del suo valore; è infatti compito del computer prelevare il valore che ha in memoria ogni volta che incontra quella determinata variabile. Ad esempio, supponiamo di avere due variabili chiamate `value1` e `value2` e che entrambe contengano un numero. Per sommare i due valori delle variabili si potrebbe scrivere la seguente istruzione:

```
value1 + value2
```

Quando esegue l'istruzione, il computer individua il valore di ciascuna variabile e li somma.

In ActionScript 3.0, le variabili sono caratterizzate da tre aspetti:

- Il nome della variabile
- Il tipo di dati che la variabile può rappresentare
- L'effettivo valore memorizzato nel computer

Abbiamo già chiarito il concetto che il computer usa il nome della variabile come segnaposto del valore. Ma il tipo di dati è un aspetto altrettanto importante. Durante la creazione di una variabile in ActionScript, è necessario specificare il tipo di dati che la variabile potrà contenere. Da questo momento in avanti, le istruzioni del programma potranno memorizzare solo quel tipo di dati nella variabile e qualsiasi operazione eseguita sul valore dovrà rispettare le caratteristiche associate al tipo di dati assegnato. Per creare una variabile, operazione comunemente definita come *dichiarare* una variabile, in ActionScript si usa l'istruzione `var`:

```
var value1:Number;
```

In questo caso, abbiamo istruito il computer di creare la variabile `value1`, che potrà contenere solo dati di tipo Number ("Number" è un tipo di dati specifico di ActionScript che corrisponde a "valore numerico"). In alternativa, è possibile associare immediatamente un valore alla variabile:

```
var value2:Number = 17;
```

In Adobe Flash CS3 Professional esiste un altro modo per dichiarare una variabile. Quando si inserisce un simbolo di clip filmato, un simbolo di pulsante o un campo di testo sullo stage, è possibile associarlo a un nome di istanza nella finestra di ispezione Proprietà. In modo completamente trasparente per l'utente, Flash crea una variabile con lo stesso nome dell'istanza che il programmatore può utilizzare nel codice ActionScript per richiamare quell'elemento dello stage. Per esempio, se si inserisce sullo stage un simbolo di clip filmato a cui si assegna il nome di istanza `rocketShip`, ogni volta che si utilizza la variabile `rocketShip` nel codice ActionScript si richiama quel clip filmato.

Tipi di dati

ActionScript mette a disposizione vari tipi di dati da utilizzare per le variabili. Alcuni di essi possono essere considerati tipi di dati "semplici" o "fondamentali":

- String: unità di testo, come un nome o il capitolo di un libro
- Dati numerici: ActionScript 3.0 comprende tre sottogruppi specifici per i dati numerici:
  - Number: qualsiasi valore numerico, con o senza decimali
  - int: numero intero senza decimali
  - uint: numero intero senza segno, vale a dire un numero intero non negativo
- Boolean: tipo di dati i cui unici valori possibili sono vero o falso, che descrive, ad esempio, se uno switch è attivo o se due valori sono uguali

I tipi di dati semplici rappresentano una porzione di dati singola: ad esempio, un unico numero o un'unica sequenza di testo. Tuttavia, la maggior parte dei tipi di dati definiti tramite ActionScript sono dati complessi in quanto rappresentano un gruppo di valori combinati tra loro. Per esempio, una variabile del tipo Date rappresenta un valore unico, cioè una data precisa. Tuttavia, il valore della data è rappresentato da una serie di valori: il giorno, il mese, l'anno, le ore, i minuti, i secondi e così via, vale a dire da una serie di numeri distinti.
Di conseguenza, mentre noi concepiamo una data come un valore singolo (e lo possiamo trattare come tale impostando una variabile Date), il computer considera la data come una serie di valori che, combinati, definiscono un momento preciso.

La più parte dei tipi di dati incorporati, e dei tipi di dati definiti dai programmatori, sono in realtà dati complessi. Alcuni dei tipi di dati complessi più conosciuti sono:

- MovieClip: simbolo di clip filmato
- TextField: campo di testo dinamico o di input
- SimpleButton: simbolo di pulsante
- Date: indicazione temporale precisa (data e ora)

Due termini che vengono spesso utilizzati come sinonimo di tipo di dati sono classe e oggetto. Una *classe* è semplicemente la definizione di un tipo di dati, il modello di riferimento per tutti gli oggetti appartenenti a quel tipo di dati. Come dire, "tutte le variabili del tipo di dati Example hanno queste caratteristiche: A, B e C". Un *oggetto*, d'altro canto, è una istanza di una classe; una variabile di tipo MovieClip può essere descritta come un oggetto MovieClip. Gli esempi seguenti sono modi diversi per esprimere lo stesso concetto:

- Il tipo di dati della variabile `myVariable` è Number.
- La variabile `myVariable` è un'istanza Number.
- La variabile `myVariable` è un oggetto Number.
- La variabile `myVariable` è un'istanza della classe Number.

Operazioni con gli oggetti

ActionScript è un cosiddetto linguaggio di programmazione orientato agli oggetti.
La programmazione orientata agli oggetti è semplicemente un modello di programmazione, un modo di organizzare il codice per definire un programma che utilizza gli oggetti.

In precedenza, abbiamo descritto un programma per computer come una serie di istruzioni o procedure che il computer esegue. Concettualmente, quindi, potremmo immaginare che un programma è solo un lunghissimo elenco di istruzioni. In realtà, nella programmazione orientata agli oggetti, le istruzioni sono raggruppate in base all'oggetto a cui si riferiscono; di conseguenza, il codice è organizzato in moduli funzionali dove le procedure correlate o i dati correlati sono raggruppati in un contenitore.

Chi sa utilizzare i simboli in Flash è abituato a utilizzare gli oggetti. Immaginiamo che sia stato definito un simbolo di filmato clip, ad esempio il disegno di un rettangolo, e che una copia di questo sia stata inserita sullo stage. Il simbolo del filmato clip è anche, letteralmente, un oggetto di ActionScript; oltre a essere una istanza della classe MovieClip.

Il programmatore può modificare varie caratteristiche del filmato clip. Dopo averlo selezionato, può cambiarne i valori nella finestra di ispezione Proprietà, cioè modificare la coordinata x, la larghezza, regolarne i colori, modificando il valore alfa (trasparenza), o applicare un filtro ombra esterna. Con altri strumenti di Flash può apportare ulteriori modifiche, ad esempio può applicare una rotazione usando lo strumento Trasformazione libera. Le operazioni eseguite per modificare il filmato clip nell'ambiente di creazione di Flash possono essere impostate anche in ActionScript cambiando i frammenti di dati che vengono combinati nel pacchetto definito oggetto MovieClip.

Nella programmazione orientata agli oggetti di ActionScript, una classe può comprendere tre tipi di caratteristiche:

- Proprietà
- Metodi
- Eventi

Combinati insieme, questi elementi permettono di manipolare i dati utilizzati dal programma e di definire le azioni da eseguire e la relativa sequenza.

Proprietà

Una proprietà rappresenta una porzione di dato che, combinata con altre, fa parte dell'oggetto. Un oggetto "song" potrebbe avere le proprietà `artist` e `title`; la classe MovieClip ha proprietà del tipo `rotation`, `x`, `width` e `alpha`. Le proprietà vengono utilizzate come se fossero variabili singole, in realtà, si può addirittura considerare le proprietà come variabili secondarie di un oggetto.

Ecco alcuni esempi dell'impiego di proprietà nel codice ActionScript. Questa riga di codice sposta il filmato clip denominato `square` nella coordinata x di 100 pixel:

```
square.x = 100;
```

Questa riga di codice utilizza la proprietà rotation per imprimere al filmato clip `square` la stessa rotazione del filmato clip `triangle`:

```
square.rotation = triangle.rotation;
```

Questa riga di codice modifica la scala orizzontale del filmato clip `square` su un valore di una volta e mezzo superiore rispetto all'originale:

```
square.scaleX = 1.5;
```

Notare la struttura che accomuna queste righe di codice: si scrive la variabile (`square`, `triangle`) come nome dell'oggetto seguita da un punto (`.`) a sua volta seguito dal nome della proprietà (`x`, `rotation`, `scaleX`). Il punto, conosciuto come *operatore punto*, introduce un elemento secondario di un oggetto. L'intera struttura, "nome variabile-punto-nome proprietà," viene considerata un'unica variabile che dà il nome a un singolo valore della memoria del computer.

Metodi

Un *metodo* è un'azione eseguita da un oggetto. Per esempio, se usando Flash si è creato un simbolo di clip filmato contenente vari fotogrammi e animazioni nella linea temporale, è possibile riprodurre o interrompere il filmato clip oppure spostare l'indicatore di riproduzione su un particolare fotogramma.

Questa riga di codice indica di avviare la riproduzione del filmato clip denominato `shortFilm`:

```
shortFilm.play();
```

Questa riga di codice indica di interrompere la riproduzione del filmato clip denominato `shortFilm` (l'indicatore di riproduzione si ferma nel punto in cui si trova, come succede quando si preme il tasto pausa di un video):

```
shortFilm.stop();
```

Questa riga di codice indica di spostare l'indicatore di riproduzione al fotogramma 1 e di interrompere la riproduzione del filmato clip denominato `shortFilm` (come succede quando si riavvolge un video):

```
shortFilm.gotoAndStop(1);
```

Risulta chiaro che si invocano i metodi proprio come le proprietà: si scrive il nome dell'oggetto (una variabile) seguito da un punto, poi si aggiunge il nome del metodo seguito da due parentesi. Le parentesi indicano la *chiamata* al metodo, vale a dire che si sta richiedendo all'oggetto di eseguire quella determinata azione. A volte le parentesi possono contenere dei valori o delle variabili che forniscono informazioni aggiuntive da utilizzare per eseguire l'azione. Questi valori aggiuntivi vengono definiti *parametri* del metodo. Per esempio, per il metodo `gotoAndStop()` è necessario specificare il fotogramma dove l'indicatore di riproduzione si deve fermare, pertanto richiede la specifica di un parametro tra parentesi. Altri metodi, come `play()` e `stop()`, sono sufficientemente chiari e non richiedono informazioni supplementari, anche se le due parentesi del nome vengono mantenute.

Contrariamente alle proprietà e alle variabili, i metodi non sono utilizzati come segnaposto di valori, nonostante alcuni di essi possano eseguire dei calcoli e restituire un risultato che può essere utilizzato come una variabile. Per esempio, il metodo `toString()` della classe Number converte il valore numerico in una stringa di testo:

```
var numericData:Number = 9;
var textData:String = numericData.toString();
```

Si può utilizzare il metodo `toString()` per visualizzare il valore della variabile Number in un campo di testo dello schermo. La proprietà `text`, che rappresenta il testo visualizzato sullo schermo, della classe TextField è definita come String, di conseguenza può contenere solo valori di testo. Questa riga di codice converte il valore numerico della variabile `numericData` in testo e lo visualizza sullo schermo nell'oggetto TextField denominato `calculatorDisplay`:

```
calculatorDisplay.text = numericData.toString();
```

Eventi

Abbiamo descritto un programma per computer come una serie di istruzioni che il computer esegue in sequenza. I programmi più semplici non sono niente di più: una serie di procedure che il computer esegue e terminate le quali il programma finisce. I programmi creati con ActionScript, tuttavia, sono progettati per non interrompersi e attendere l'input da parte dell'utente o il verificarsi di altre condizioni. Gli eventi sono il meccanismo che definisce le istruzioni che il computer esegue e la relativa sequenza.

In sostanza, gli *eventi* sono condizioni che si verificano di cui ActionScript è consapevole e a cui è in grado di reagire. Molti eventi solo collegati all'interazione dell'utente, la selezione di un pulsante o la pressione di un tasto della tastiera, ma ne esistono anche di altri tipi. Per esempio, se si usa ActionScript per caricare un'immagine esterna, esiste un evento che può avvertire quando l'immagine è stata caricata completamente. In pratica, durante l'esecuzione di un programma ActionScript, Adobe Flash Player "rimane in ascolto" e attende il verificarsi di determinati eventi; quando questi si realizzano, Flash esegue il codice ActionScript specificato per quei particolari eventi.

Principi di gestione degli eventi

Per *gestione degli eventi* si intende la definizione delle azioni da eseguire in risposta a particolari eventi. La scrittura di codice ActionScript per la gestione degli eventi presuppone l'identificazione di tre elementi importanti:

- L'origine dell'evento: Su che oggetto si verificherà l'evento? Ad esempio, quale pulsante verrà selezionato o quale oggetto Loader carica l'immagine? L'origine dell'evento corrisponde anche al *target dell'evento*, in quanto si tratta dell'oggetto in cui Flash Player esegue l'azione (in cui l'evento si verifica).
- L'evento: qual è l'evento che si attende, l'evento a cui bisogna rispondere? È particolarmente importante identificare correttamente l'evento, perché molti oggetti attivano eventi diversi.
- La risposta: Che cosa deve succedere quando l'evento si realizza?

Il codice ActionScript per la gestione degli eventi deve sempre comprendere questi tre elementi e riprodurre questa struttura di base (gli elementi in grassetto sono segnaposti modificabili a seconda del caso specifico):

```
function eventResponse(eventObject:EventType):void
{
  // Le azioni di risposta agli eventi vanno definite qui.
}

eventSource.addEventListener(EventType.EVENT_NAME, eventResponse);
```

Questo esempio di codice svolge due compiti. Definisce una funzione, vale a dire che specifica le azioni che devono avvenire in risposta all'evento. In secondo luogo, chiama il metodo `addEventListener()` dell'oggetto di origine, in pratica associando la funzione all'evento, in modo che quando si verifica l'evento vengono attivate le azioni descritte nella funzione. Ora affronteremo ogni compito in maggiore dettaglio.

Una *funzione* rappresenta un modo per raggruppare insieme varie azioni creando un'unica entità dotata di un nome proprio. Le funzioni sono quasi identiche ai metodi, con l'unica differenza che non sono necessariamente associate a una classe specifica; infatti, si potrebbe definire i metodi come funzioni associate a una classe specifica. Una funzione per la gestione di eventi deve avere obbligatoriamente un nome (in questo caso `eventResponse`) e un parametro (nel nostro esempio: `eventObject`). La specifica di un parametro per una funzione equivale alla dichiarazione di una variabile, di conseguenza, per il parametro è necessario definire il tipo di dati. Per ogni evento viene definita una classe di ActionScript e il tipo di dati specificato per il parametro della funzione è sempre la classe associata all'evento a cui si intende rispondere. Infine, tra le parentesi graffe (`{ ... }`), scrivere le procedure che il computer deve eseguire quando l'evento si verifica.

Una volta scritta la funzione di gestione degli eventi, è necessario comunicare all'oggetto di origine dell'evento (l'oggetto su cui si verifica l'evento, ad esempio, il pulsante) che la funzione deve essere richiamata nel momento in cui l'evento si verifica. A questo scopo, si chiama il metodo `addEventListener()` dell'oggetto; infatti, tutti gli oggetti associati ad eventi sono associati anche al metodo `addEventListener()`. Il metodo `addEventListener()` accetta due parametri:

- Prima di tutto, il nome dell'evento specifico a cui rispondere. Come già spiegato in precedenza, ogni evento è collegato a una classe specifica e nella classe ogni evento ha un valore predefinito speciale, o nome, che l'utente deve usare come primo parametro del metodo.
- Secondariamente, il nome della funzione di risposta all'evento. Si noti che i nomi di funzione passati come parametri vanno scritti senza parentesi.

Analisi del processo di gestione degli eventi

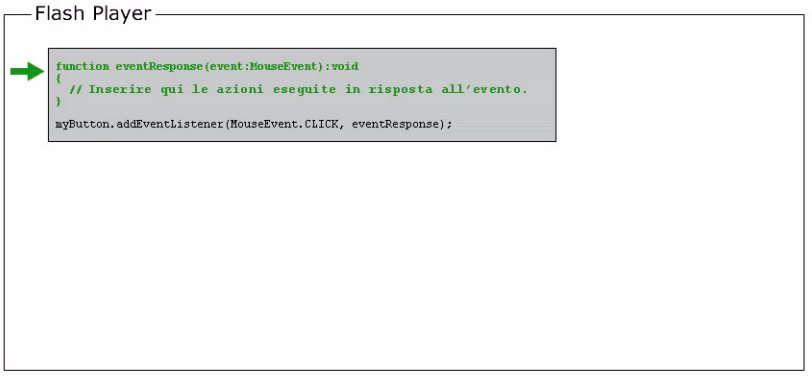
Segue una descrizione dettagliata del processo che si mette in atto quando si crea un listener di eventi. L'esempio illustra la creazione di una funzione di listener da chiamare quando l'utente fa clic sull'oggetto `myButton`.

Il codice scritto dal programmatore è il seguente:

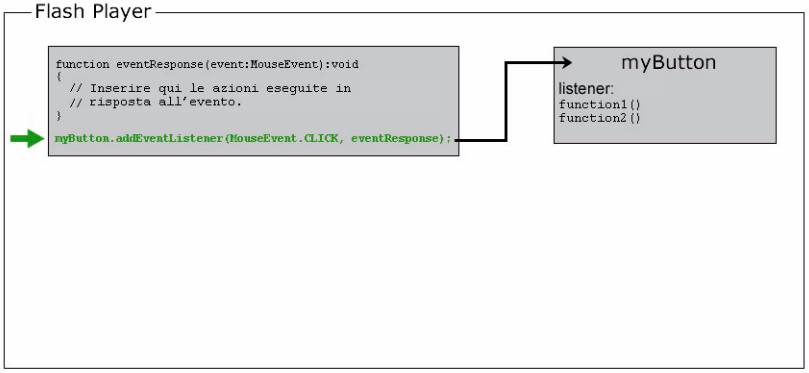
```
function eventResponse(event:MouseEvent):void
{
  // Actions performed in response to the event go here.
}

myButton.addEventListener(MouseEvent.CLICK, eventResponse);
```

Ecco cosa succede quando il codice viene eseguito da Flash Player:

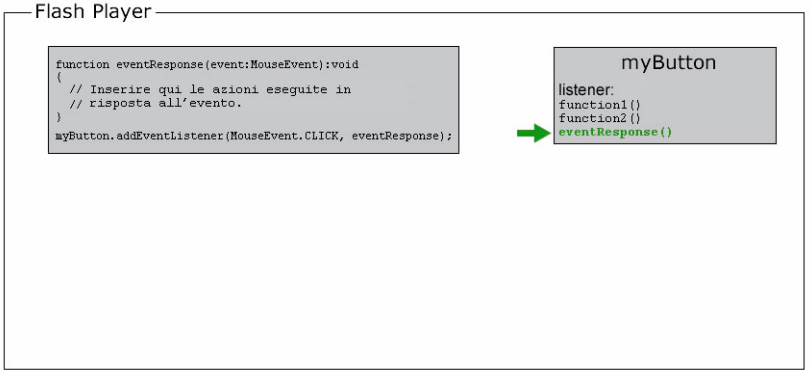
1. Nel momento in cui carica il file SWF, Flash Player registra l'esistenza della funzione `eventResponse()`.



2. Flash Player esegue il codice (precisamente le righe di codice che non fanno parte di una funzione). In questo caso si tratta di una sola riga di codice: la chiamata al metodo `addEventListener()` sull'oggetto di origine dell'evento (denominato `myButton`) e il passaggio della funzione `eventResponse` come parametro.



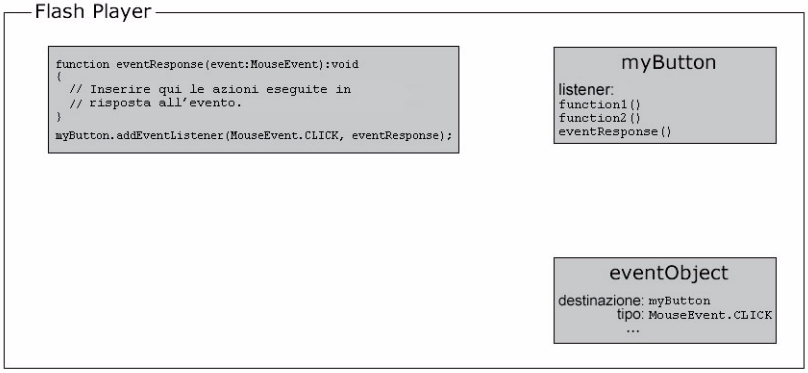
   a. Internamente, `myButton` comprende una serie di funzioni che intercettano ciascuno di questi eventi, quindi, quando viene chiamato il metodo `addEventListener()`, `myButton` registra la funzione `eventResponse()` tra i suoi listener di eventi.



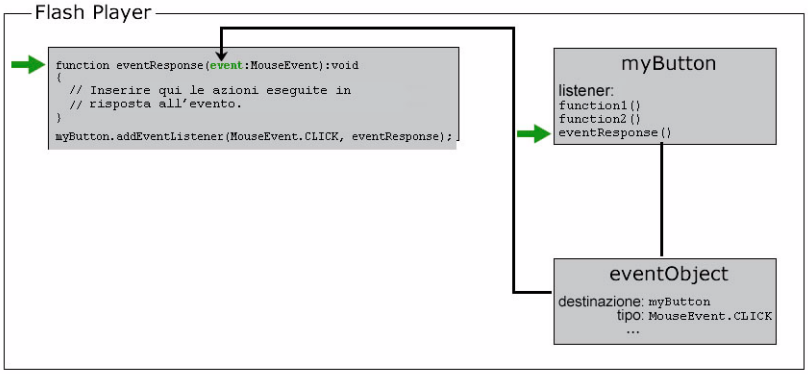
3. A un certo punto, quando l'utente fa clic sull'oggetto `myButton`, viene attivato l'evento `click` (definito nel codice come `MouseEvent.CLICK`).

A questo punto, si verifica quanto segue:

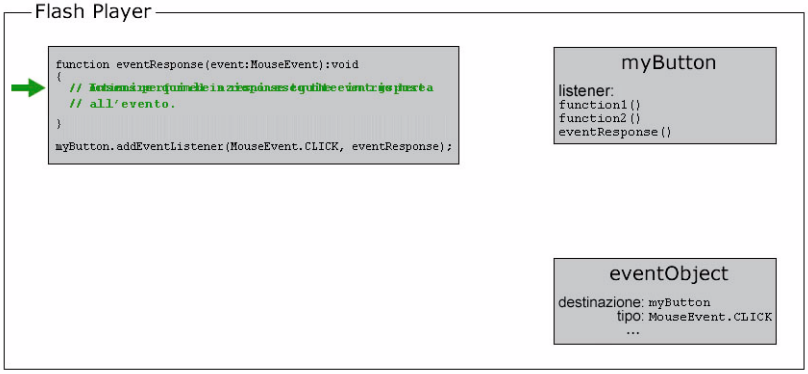
a. Flash Player crea in oggetto, un'istanza della classe associata all'evento in questione (nel nostro esempio: MouseEvent). Per molti eventi si tratterà di un'istanza della classe Event; per gli eventi del mouse si tratterà di un'istanza di MouseEvent e per tutti gli altri eventi sarà un'istanza della classe associata all'evento. L'oggetto creato viene definito *oggetto evento* e contiene informazioni specifiche sull'evento che si è verificato: di che tipo di evento si tratta, dove si è verificato e altre eventuali informazioni.



b. Flash Player esamina l'elenco di listener di eventi registrati in `myButton` e chiama ogni singola funzione passando l'oggetto evento come parametro. Dal momento che la funzione `eventResponse()` è uno dei listener di `myButton`, come parte del processo Flash Player chiama la funzione `eventResponse()`.



c. Nel momento in cui viene chiamata la funzione `eventResponse()`, il suo codice viene eseguito e le azioni specificate vengono attivate.



Esempi di gestione degli eventi

Seguono alcuni esempi concreti di eventi per dare un'idea degli elementi evento più comuni e delle possibili variazioni disponibili per scrivere codice per la gestione di eventi:

- Selezionare un pulsante per avviare la riproduzione del filmato clip corrente. Nell'esempio che segue, `playButton` è il nome istanza del pulsante e `this` è un nome speciale che significa "l'oggetto corrente":

```
this.stop();

function playMovie(event:MouseEvent):void
{
  this.play();
}

playButton.addEventListener(MouseEvent.CLICK, playMovie);
```

- Rilevare l'inserimento di testo in un campo di testo. Nell'esempio che segue, `entryText` è un campo di testo di input e `outputText` è un campo di testo dinamico:

```
function updateOutput(event:TextEvent):void
{
  var pressedKey:String = event.text;
  outputText.text = "You typed: " + pressedKey;
}

entryText.addEventListener(TextEvent.TEXT_INPUT, updateOutput);
```

- Selezionare un pulsante per accedere a un URL. Nell'esempio che segue, `linkButton` è il nome di istanza del pulsante:

```
function gotoAdobeSite(event:MouseEvent):void
{
  var adobeURL:URLRequest = new URLRequest("http://www.adobe.com/");
  navigateToURL(adobeURL);
}

linkButton.addEventListener(MouseEvent.CLICK, gotoAdobeSite);
```

Creazione di istanze di oggetti

Per poter utilizzare un oggetto in ActionScript, l'oggetto deve già essere stato creato. Il primo passo nella creazione di un oggetto è la dichiarazione di una variabile, vale a dire, l'identificazione di una porzione di memoria del computer destinata a contenere dei dati. In seguito, questo spazio vuoto va riempito assegnando un valore alla variabile, in pratica creando un oggetto e memorizzandolo nella variabile in modo che possa diventare utilizzabile. Il processo di creazione di un oggetto corrisponde alla creazione di *un'istanza* di una particolare classe.

Esiste un modo semplice per creare un'istanza di un oggetto che non prevede l'utilizzo di ActionScript. In Flash, quando si inserisce un simbolo di clip filmato, un simbolo di pulsante o un campo di testo sullo stage e ad esso si assegna un nome di istanza nella finestra di ispezione Proprietà, l'applicazione dichiara automaticamente una variabile usando quel nome di istanza, crea un'istanza dell'oggetto e memorizza l'oggetto nella variabile. Analogamente, quando si usa Adobe Flex Builder per creare un componente in Macromedia® MXML™ di Adobe (codificando un tag MXML o inserendo il componente nell'editor in modalità Design) e al componente si assegna un ID (nel codice MXML o nella vista Proprietà Flex), l'ID diventa il nome di una variabile di ActionScript all'interno della quale viene memorizzata l'istanza del componente creata.

Tuttavia, non sempre l'utente desidera creare un oggetto graficamente. Esistono infatti altri metodi per creare istanze di oggetti usando solo ActionScript. Innanzi tutto, molti tipi di dati di ActionScript consentono di creare istanze usando un'*espressione letterale*, vale a dire, un valore scritto direttamente in linguaggio ActionScript. Di seguito sono forniti alcuni esempi.

- Valore numerico letterale (inserire il numero direttamente):

```
var someNumber:Number = 17.239;
var someNegativeInteger:int = -53;
var someUint:uint = 22;
```

- Valore String letterale (racchiudere il testo tra virgolette):

```
var firstName:String = "George";
var soliloquy:String = "To be or not to be, that is the question...";
```

- Valore Boolean letterale (usare i valori letterali `true` o `false`):

```
var niceWeather:Boolean = true;
var playingOutside:Boolean = false;
```

- Valore XML letterale (inserire XML direttamente):

```
var employee:XML = <employee>
    <firstName>Harold</firstName>
    <lastName>Webster</lastName>
  </employee>;
```

ActionScript definisce espressioni letterali anche per i tipi di dati Array, RegExp, Object e Function. Per informazioni dettagliate sulle classi, vedere "Operazioni con gli array" a pagina 239, "Uso delle espressioni regolari" a pagina 305 e "Tipo di dati Object" a pagina 103.

Per qualsiasi altro tipo di dati si crea un'istanza dell'oggetto usando l'operatore `new` con il nome della classe in questo modo:

```
var raceCar:MovieClip = new MovieClip();
var birthday:Date = new Date(2006, 7, 9);
```

La creazione di un oggetto mediante l'operatore `new` è comunemente nota come "chiamare la funzione di costruzione della classe". La *funzione di costruzione* è uno speciale metodo chiamato come parte del processo di creazione di un'istanza di una classe. Tenere presente che per creare un'istanza in questo modo, il nome della classe deve essere seguito dalle parentesi e può essere necessario specificare dei parametri, proprio come accade quando si chiama un metodo.

NOTA

È possibile usare l'operatore `new` per creare un'istanza dell'oggetto anche per i tipi di dati che consentono di creare istanze tramite espressioni letterali. Queste due righe di codice, ad esempio, generano lo stesso risultato:

```
var someNumber:Number = 6.33;
var someNumber:Number = new Number(6.33);
```

È importante avere dimestichezza con il metodo `new ClassName()` per la creazione degli oggetti perché se si presenta la necessità di creare un'istanza di un tipo di dati ActionScript per cui non è prevista una rappresentazione visiva (per cui non si può inserire un elemento sullo stage di Flash né usare la modalità Design dell'editor MXML di Flex Builder), l'unica opzione di cui si dispone è creare l'oggetto direttamente in ActionScript usando l'operatore `new`.

In particolare, Flash permette di usare l'operatore `new` per creare un'istanza di un simbolo di filmato clip definito nella Libreria ma non inserito sullo stage. Per ulteriori informazioni a riguardo, vedere "Creazione di oggetti MovieClip mediante ActionScript" a pagina 536.

Elementi comuni dei programmi

Oltre alla dichiarazione di variabili, alla creazione di istanze di oggetti e alla manipolazione degli oggetti tramite proprietà e metodi, esistono altri elementi fondamentali per la creazione dei programmi ActionScript.

Operatori

Gli *operatori* sono simboli speciali (o, più raramente, parole) che permettono di eseguire dei calcoli. Vengono utilizzati prevalentemente per operazioni aritmetiche e per comparare fra loro dei valori. Come regola generale, un operatore utilizza uno o più valori per estrapolare un risultato. Ad esempio:

- L'operatore di somma (+) somma tra loro più valori e restituisce un numero come risultato:

  ```
  var sum:Number = 23 + 32;
  ```

- L'operatore di moltiplicazione (*) moltiplica un valore per un altro e restituisce un numero come risultato:

  ```
  var energy:Number = mass * speedOfLight * speedOfLight;
  ```

- L'operatore di uguaglianza (==) confronta due valori per stabilire se sono uguali e restituisce un valore booleano (true o false):

  ```
  if (dayOfWeek == "Wednesday")
  {
    takeOutTrash();
  }
  ```

  Come si può evincere da questi esempi, l'operatore di uguaglianza e in generale gli operatori di "confronto", vengono utilizzati prevalentemente con l'istruzione `if` per determinare se le procedure che seguono devono essere eseguite o meno.

  Per maggiori dettagli ed esempi sull'uso degli operatori, vedere "Operatori" a pagina 116.

Commenti

Durante la scrittura del codice ActionScript può essere utile aggiungere delle note per spiegare il funzionamento di determinate righe o annotare il motivo di una particolare scelta.
I *commenti del codice* sono uno strumento che permette di aggiungere testo che viene ignorato dal computer. Nel codice ActionScript si possono aggiungere due tipi di commenti:

- Commento a riga singola: Per indicare che una riga è un commento, inserire due barre in qualsiasi punto della riga. Quanto segue le due barre viene ignorato dal computer:

```
// Questo è un commento e viene ignorato dal computer.
var age:Number = 10; // Imposta l'età su 10 per impostazione predefinita.
```

- Commenti su più righe: Un commento che occupa più righe comprende un indicatore di inizio (`/*`), il commento vero e proprio seguito da un indicatore di fine (`*/`). Tutto il testo scritto tra i due indicatori viene ignorato dal computer, indipendentemente dal numero di righe occupate:

```
/*
Si può inserire una descrizione lunghissima, che spieghi
l'uso di una particolare funzione o il significato di un'intera
  sezione di codice.

In ogni caso, tutte queste righe vengono ignorate dal computer.
*/
```

I commenti vengono spesso utilizzati anche per un altro scopo: disabilitare temporaneamente una o più righe di codice, ad esempio, per testare soluzioni alternative o individuare il motivo per cui il codice ActionScript non funziona come previsto.

Controllo del flusso

Spesso, all'interno di un programma, si presenta la necessità di ripetere determinate azioni, di eseguire solo certe azioni e non altre o di eseguire azioni alternative a seconda delle condizioni.
Il *controllo del flusso* è il meccanismo che gestisce quali azioni devono essere eseguite.
In ActionScript sono disponibili vari tipi di elementi di controllo del flusso.

- Funzioni: le funzioni agiscono come scorciatoie: raggruppano una serie di azioni in un'unica entità e consentono di eseguire dei calcoli. Sono particolarmente importanti per la gestione di eventi ma anche come strumento per raggruppare serie di istruzioni.
  Per ulteriori informazioni sulle funzioni, vedere "Funzioni" a pagina 130.
- Cicli: i cicli sono costrutti che permettono di eseguire ciclicamente un gruppo di istruzioni per un determinato numero di volte o fino al verificarsi di una determinata condizione. I cicli consentono di manipolare vari elementi collegati tra loro tramite l'uso di una variabile il cui valore cambia a ogni iterazione. Per ulteriori informazioni sui cicli, vedere "Ripetizione ciclica" a pagina 127.

- Istruzioni condizionali: le istruzioni condizionali consentono di definire procedure da eseguire solo in alcune circostanze o di presentare pezzi di codice alternativi da eseguire a seconda delle condizioni che si verificano. L'istruzione condizionale più usata è la struttura `if`. `if` permette di verificare un valore o un'espressione tra parentesi. Se l'espressione risulta vera (`true`) viene eseguito il codice riportato tra le parentesi graffe; in caso contrario, il codice tra parentesi graffe viene ignorato. Ad esempio:

```
if (age < 20)
{
  // mostra contenuto speciale per teenager
}
```

L'istruzione generalmente associata a `if` è `else`, la quale indica le procedure alternative da eseguire se la condizione non è `true`:

```
if (username == "admin")
{
  // esegue operazioni riservate agli amministratori, come mostrare
  // opzioni supplementari
}
else
{
  // esegue operazioni non riservate agli amministratori
}
```

Per ulteriori informazioni sulle istruzioni condizionali, vedere "Istruzioni condizionali" a pagina 124.

Esempio: Porzione del portfolio animazioni

Questo esempio ha lo scopo di offrire all'utente una prima opportunità di scoprire come è possibile unire parti di ActionScript in un'applicazione ActionScript completa, anche se densa di codice. La porzione del portfolio animazioni è un esempio di come sia possibile modificare un'animazione lineare esistente (ad esempio, una porzione creata per un cliente), aggiungendovi alcuni elementi interattivi minori appropriati, e incorporarla in un portfolio in linea. Il comportamento interattivo che verrà aggiunto all'animazione in questo esempio include due pulsanti su cui l'utente può fare clic: uno per avviare l'animazione e l'altro per accedere a un URL separato (ad esempio il menu del portfolio o la home page dell'autore).

Il processo di creazione di questa porzione può essere suddiviso nelle seguenti fasi principali:

1. Preparazione del file FLA per l'aggiunta di elementi ActionScript e interattivi
2. Creazione e aggiunta di pulsanti
3. Scrittura del codice ActionScript
4. Prova dell'applicazione

Preparazione per l'aggiunta di elementi interattivi

Prima di aggiungere elementi interattivi all'animazione, è utile configurare il file FLA creando alcuni spazi in cui aggiungere il nuovo contenuto. Questo comprende la creazione dello spazio effettivo sullo stage in cui i pulsanti verranno posizionati e anche la creazione di "spazio" nel file FLA per mantenere separati gli elementi diversi.

Per configurare il file FLA per l'aggiunta di elementi interattivi:

1. Se non si dispone di un'animazione lineare a cui aggiungere elementi interattivi, creare un nuovo file FLA con un'animazione semplice, ad esempio un'unica interpolazione di movimento o di forma. In alternativa, aprire il file FLA contenente l'animazione da utilizzare nel progetto e salvarlo con un nuovo nome per creare un nuovo file di lavoro.
2. Decidere la posizione in cui devono essere visualizzati i due pulsanti (uno per avviare l'animazione e l'altro per collegare al portfolio o alla home page dell'autore). Se necessario, liberare o aggiungere spazio sullo stage per il nuovo contenuto. Se l'animazione non è dotata di una finestra di avvio, è possibile crearne una nel primo fotogramma (è possibile anche spostare l'animazione in modo che venga avviata nel fotogramma 2 o in uno successivo).
3. Aggiungere un nuovo livello sopra gli altri livelli nella linea temporale e denominarlo **buttons**. Questo è il livello in cui verranno aggiunti i pulsanti.
4. Aggiungere un nuovo livello sopra il livello buttons e denominarlo **actions**. Questo è il livello in cui verrà aggiunto all'applicazione il codice ActionScript.

Creazione e aggiunta dei pulsanti

A questo punto, è necessario creare e posizionare i pulsanti che costituiscono il fulcro dell'applicazione interattiva.

Per creare e aggiungere pulsanti al file FLA:

1. Mediante gli strumenti di disegno, creare l'aspetto visivo del primo pulsante (il pulsante "play") nel livello buttons. Ad esempio, è possibile disegnare un'ovale orizzontale contenente del testo.
2. Mediante lo strumento Selezione, selezionare tutte le parti grafiche del pulsante.
3. Dal menu principale, scegliere Elabora > Converti in simbolo.
4. Nella finestra di dialogo, scegliere Pulsante come tipo di simbolo, assegnare un nome al simbolo e fare clic su OK.

5. Con il pulsante selezionato, nella finestra di ispezione Proprietà assegnare al pulsante il nome di istanza `playButton`.
6. Ripetere i punti da 1 a 5 per creare il pulsante che collega alla home page dell'autore. Denominare questo pulsante `homeButton`.

Scrittura del codice

Il codice ActionScript per questa applicazione può essere suddiviso in tre set di funzionalità, sebbene vengano inserite tutte nella stessa posizione. Le tre azioni che il codice deve eseguire sono le seguenti:

- Interrompere l'indicatore di riproduzione non appena il file SWF viene caricato (quando l'indicatore di riproduzione raggiunge il fotogramma 1).
- Attendere un evento per avviare la riproduzione del file SWF quando l'utente fa clic sul pulsante play.
- Attendere un evento per indirizzare il browser all'URL appropriato quando l'utente fa clic sul pulsante della home page dell'autore.

Per creare il codice che interrompe l'indicatore di riproduzione quando raggiunge il fotogramma 1:

1. Selezionare il fotogramma chiave nel fotogramma 1 del livello actions.
2. Per aprire il pannello Azioni, dal menu principale scegliere Finestra > Azioni.
3. Nel riquadro dello script, immettere il codice seguente:

```
stop();
```

Per scrivere il codice per avviare l'animazione quando si fa clic sul pulsante play:

1. Al termine del codice inserito nei punti precedenti, aggiungere due righe vuote.
2. Inserire il codice seguente alla fine dello script:

```
function startMovie(event:MouseEvent):void
{
  this.play();
}
```

Questo codice definisce una funzione di nome `startMovie()`. Quando la funzione `startMovie()` viene chiamata, viene avviata la riproduzione della linea temporale principale.

3. Nella riga successiva al codice aggiunto al punto precedente, inserire la riga di codice seguente:

```
playButton.addEventListener(MouseEvent.CLICK, startMovie);
```

Questa riga di codice registra la funzione `startMovie()` come listener per l'evento `click` di `playButton`. In altre parole, fa in modo che quando si fa clic sul pulsante di nome `playButton`, venga chiamata la funzione `startMovie()`.

Per scrivere codice per indirizzare il browser a un URL quando si fa clic sul pulsante home page:

1. Al termine del codice inserito nei punti precedenti, aggiungere due righe vuote.
2. Inserire il codice seguente alla fine dello script:

```
function gotoAuthorPage(event:MouseEvent):void
{
  var targetURL:URLRequest = new URLRequest("http://example.com/");
  navigateToURL(targetURL);
}
```

Questo codice definisce una funzione di nome `gotoAuthorPage()`. Questa funzione prima crea un'istanza URLRequest che rappresenta l'URL http://example.com/, quindi passa questo URL alla funzione `navigateToURL()`, che determina l'apertura dell'URL nel browser.

3. Nella riga successiva al codice aggiunto al punto precedente, inserire la riga di codice seguente:

```
homeButton.addEventListener(MouseEvent.CLICK, gotoAuthorPage);
```

Questa riga di codice registra la funzione `gotoAuthorPage()` come listener per l'evento `click` di `homeButton`. In altre parole, fa in modo che quando si fa clic sul pulsante di nome `homeButton`, venga chiamata la funzione `gotoAuthorPage()`.

Prova dell'applicazione

A questo punto, l'applicazione dovrebbe essere completamente funzionante. Per verificarlo, effettuare una prova.

Per provare l'applicazione:

1. Dal menu principale, scegliere Controllo > Prova filmato. Flash crea il file SWF e lo apre in una finestra di Flash Player.
2. Provare entrambi i pulsanti per verificare che eseguano le funzioni impostate.

3. Se i pulsanti non funzionano, controllare le seguenti condizioni:
   - I pulsanti hanno nomi di istanze distinti?
   - Per le chiamate al metodo `addEventListener()` vengono utilizzati gli stessi nomi delle istanze dei pulsanti?
   - I nomi evento utilizzati nelle chiamate al metodo `addEventListener()` sono corretti?
   - Il parametro specificato per ciascuna funzione è corretto? (Entrambe dovrebbero avere un singolo parametro con tipo di dati MouseEvent.)

   Tutti questi e la maggior parte degli altri potenziali errori generano un messaggio di errore quando si sceglie il comando Prova filmato oppure quando si fa clic sul pulsante. Cercare nel pannello Errori del compilatore eventuali errori del compilatore (quelli che si verificano quando si sceglie Prova filmato per la prima volta) e nel pannello Output eventuali errori di runtime (errori che si verificano mentre la riproduzione del file SWF è in corso, ad esempio quando si fa clic su un pulsante).

Creazione di applicazioni con ActionScript

Per il processo di creazione di applicazioni tramite il codice ActionScript non è sufficiente conoscere la sintassi e i nomi delle classi. Nonostante la maggior parte delle informazioni contenute in questo manuale si concentri su questi due aspetti (sintassi e impiego delle classi ActionScript), è opportuno che l'utente sappia quali programmi può usare per scrivere codice ActionScript, come organizzare il codice ActionScript e inserirlo in un'applicazione e di quali fasi si compone lo sviluppo di un'applicazione ActionScript.

Opzioni per l'organizzazione del codice

Il codice ActionScript 3.0 può essere utilizzato come "motore" per ogni tipo di applicazione, da semplici animazioni grafiche a sofisticati sistemi di elaborazione delle transazioni client/server. A seconda del tipo di applicazione che si intende creare, si può scegliere di adottare uno o più metodi per inserire il codice ActionScript nel proprio progetto.

Memorizzazione del codice nei fotogrammi di una linea temporale di Flash

Nell'ambiente di creazione di Flash, è possibile aggiungere codice ActionScript a qualunque fotogramma della linea temporale. Il codice viene eseguito durante la riproduzione del filmato, nel momento in cui l'indicatore di riproduzione arriva al fotogramma.

L'inserimento di codice ActionScript nei fotogrammi rappresenta un metodo semplice per aggiungere comportamenti alle applicazioni realizzate in Flash. È possibile aggiungere codice a qualunque fotogramma della linea temporale principale o della linea temporale di qualsiasi simbolo MovieClip. Tuttavia, tutta questa flessibilità ha un prezzo. Quando si creano applicazioni di grandi dimensioni, diventa difficile ricordare quali script sono stati inseriti in quali fotogrammi, di conseguenza l'applicazione può diventare complicata da gestire con il passare del tempo.

Molti sviluppatori scelgono di semplificare l'organizzazione del codice ActionScript usato in Flash inserendolo solo nel primo fotogramma di una linea temporale oppure su un livello specifico del documento Flash. Il risultato è una maggiore facilità nell'individuazione e nella gestione del codice contenuto nei file FLA di Flash. Tuttavia, per utilizzare lo stesso codice in un altro progetto Flash, è necessario copiarlo e incollarlo nel nuovo file.

Affinché il codice ActionScript possa essere riutilizzato in altri progetti Flash successivi, è necessario salvarlo in file ActionScript esterni (file di testo con l'estensione .as).

Memorizzazione del codice in file ActionScript

Se un progetto prevede un uso intensivo di codice ActionScript, il modo migliore per organizzarlo è inserirlo in file di origine distinti con l'estensione .as. I file ActionScript possono essere strutturati in due modi diversi, a seconda di come si intenda utilizzarli all'interno dell'applicazione.

- Codice ActionScript non strutturato: righe di codice ActionScript, istruzioni e definizioni di funzioni comprese, scritte come se fossero immesse direttamente nella linea temporale, nel file MXML e così via.

  Si può accedere a questo tipo di codice usando l'istruzione `include` di ActionScript o il tag `<mx:Script>` in un file MXML di Adobe Flex. L'istruzione `include` inserisce il contenuto di un file ActionScript esterno in una posizione specifica e con un'area di validità specifica all'interno di uno script, come se fosse stato incluso direttamente in tale posizione. Nel linguaggio MXML di Flex, il tag `<mx:Script>` permette di specificare un attributo di origine che identifica un file ActionScript esterno da caricare in quel punto preciso dell'applicazione. Per esempio, il tag seguente carica il file ActionScript esterno Box.as:

  ```
  <mx:Script source="Box.as" />
  ```

- Definizione della classe ActionScript: definizione di una classe ActionScript, che comprende le definizioni dei metodi e delle proprietà.

  Quando si definisce una classe, è possibile accedere al codice ActionScript al suo interno creando un'istanza della classe e usandone le proprietà, i metodi e gli eventi come se si trattasse di una classe ActionScript incorporata. Questa procedura si compone di due fasi:

  - Usare l'istruzione `import` per specificare il nome completo della classe affinché il compilatore ActionScript sappia dove trovarla. Ad esempio, per utilizzare la classe MovieClip in ActionScript, è necessario prima importarla specificandone il nome completo, costituito da pacchetto e classe:

    ```
    import flash.display.MovieClip;
    ```

    In alternativa, si può importare il pacchetto che contiene la classe MovieClip, il che equivale a scrivere un'istruzione `import` separata per ogni classe del pacchetto:

    ```
    import flash.display.*;
    ```

    Si sottraggono a questa regola le classi di primo livello, che non sono definite all'interno del pacchetto.

    NOTA In Flash, se gli script sono associati a fotogrammi della linea temporale, la maggior parte delle classi incorporate (nei pacchetti flash.*) vengono importate automaticamente. Tuttavia, se si utilizzano classi personalizzate o si lavora con i componenti di creazione di Flash (i pacchetti fl.*), è necessario importare esplicitamente ogni classe per poter scrivere del codice che crei istanze di tali classi.

  - Scrivere del codice che faccia esplicitamente riferimento al nome della classe (generalmente si procede dichiarando una variabile con la classe come tipo di dati e si memorizza nella variabile un'istanza della classe). Inserendo un riferimento a un nome di classe diverso nel codice ActionScript, si indica al compilatore di caricare la definizione di tale classe. Ad esempio, data una classe esterna chiamata Box, l'istruzione seguente crea una nuova istanza di tale classe:

    ```
    var smallBox:Box = new Box(10,20);
    ```

    Quando incontra per la prima volta il riferimento alla classe Box, il compilatore cerca nel codice di origine caricato la definizione della classe Box.

Scelta dello strumento adeguato

Lo strumento più adatto, o gli strumenti da usare congiuntamente, per scrivere e modificare il codice ActionScript dipende dalle esigenze del progetto e dalle risorse di cui si dispone.

Strumento di creazione di Flash

Oltre alle funzionalità grafiche e di animazione, Adobe Flash CS3 Professional offre strumenti per utilizzare il codice ActionScript, sia per allegarlo a elementi di file FLA che per creare file esterni ActionScript. Lo strumento di creazione di Flash è ideale per i progetti che prevedono un uso intensivo di animazioni e video o quelli in cui l'utente vuole creare personalmente la maggior parte delle risorse grafiche, specialmente se si tratta di progetti che richiedono interazione minima da parte dell'utente o funzionalità che richiedono ActionScript.
Lo strumento di creazione di Flash è la scelta vincente anche per gli utenti che preferiscono creare elementi visivi e scrivere il codice usando la medesima applicazione. Un ulteriore vantaggio offerto da Flash consiste nella possibilità di utilizzare componenti dell'interfaccia utente prestabiliti, creare file SWF più piccoli e associare facilmente skin al progetto.

Adobe Flash CS3 Professional offre due strumenti per la scrittura del codice ActionScript:

- Pannello Azioni: pannello disponibile per la gestione dei file FLA e che consente di scrivere codice ActionScript e di associarlo a fotogrammi della linea temporale.
- Finestra script: editor di testo dedicato ai file di codice ActionScript (.as).

Flex Builder

Adobe Flex Builder è lo strumento privilegiato per la creazione di progetti con struttura Flex. Oltre al layout visivo e agli strumenti di modifica MXML, Flex Builder offre un editor di ActionScript completamente funzionale che permette di creare progetti Flex o contenenti esclusivamente codice ActionScript. Le applicazioni Flex offrono svariati vantaggi tra cui: una ricca gamma di componenti dell'interfaccia utente prestabiliti, controlli flessibili per il layout dinamico e meccanismi incorporati per l'interazione con origini dati esterne e il collegamento di dati esterni agli elementi dell'interfaccia utente. Tuttavia, in ragione del codice supplementare necessario per attivare queste funzioni, le applicazioni Flex generano file SWF di dimensioni superiori e non permettono una facile riassegnazione totale degli skin come le controparti di Flash.

Usare Flex Builder per creare applicazioni Internet complete e ricche di dati con Flex, per modificare il codice ActionScript e MXML e per creare un'anteprima visiva dell'applicazione, tutto usando un unico strumento.

Editor di ActionScript esterno

Dal momento che i file ActionScript (.as) sono semplici file di testo, è possibile scrivere file ActionScript usando un qualsiasi editor di testo. Oltre ai prodotti specifici per ActionScript di Adobe, sono disponibili vari editor di testo esterni dotati di funzionalità specifiche per ActionScript. Si possono scrivere file MXML e classi ActionScript usando un qualsiasi editor di testo. A partire da questi file, si può creare un'applicazione SWF (Flex o ActionScript pura) usando Flex SDK, che comprende le classi con struttura Flex e il compilatore Flex.
In alternativa, molti sviluppatori scrivono le classi ActionScript con un editor di ActionScript esterno e usano lo strumento di creazione di Flash per creare il contenuto grafico.

Si può scegliere di usare un editor di ActionScript esterno se:

- Si preferisce scrivere il codice ActionScript con un programma diverso mentre si sviluppano gli elementi visivi in Flash.
- Si usa un'applicazione per la programmazione di codice non ActionScript, come la creazione di pagine HTML o lo sviluppo di applicazioni con un linguaggio di programmazione diverso, e si desidera usare quella stessa applicazione anche per scrivere codice ActionScript.
- Si vogliono creare dei progetti Flex o ActionScript puri tramite Flex SDK senza ricorrere a Flash o Flex Builder.

Alcuni dei migliori editor di testo con supporto specifico per ActionScript sono:

- Adobe Dreamweaver® CS3
- ASDT
- FDT
- FlashDevelop
- PrimalScript
- SE|PY
- XCode (con modello e file con suggerimenti per il codice di ActionScript)

Processo di sviluppo ActionScript

Indipendentemente dalla dimensione del progetto da realizzare, definire un processo di progettazione e sviluppo dell'applicazione aiuta a lavorare in modo più produttivo ed efficace. Ecco le fasi fondamentali che deve comprendere un processo di sviluppo di un'applicazione tramite ActionScript 3.0:

1. Progettare l'applicazione.

   È importante descrivere l'applicazione prima di iniziare a crearla.

2. Comporre il codice ActionScript 3.0.

   È possibile creare il codice ActionScript utilizzando Flash, Flex Builder, Dreamweaver o un editor di testo.

3. Creare un file di applicazione Flash o Flex per l'esecuzione del codice.

   Nello strumento di creazione di Flash, questa fase richiede la creazione di un nuovo file FLA, la definizione delle impostazioni di pubblicazione, l'aggiunta dei componenti dell'interfaccia utente all'applicazione e l'inserimento di riferimenti al codice ActionScript. Nell'ambiente di sviluppo Flex, per creare un nuovo file di applicazione è necessario definire l'applicazione e aggiungere i componenti dell'interfaccia utente mediante MXML, quindi creare i riferimenti al codice ActionScript.

4. Pubblicare e provare l'applicazione ActionScript.

   In questa fase occorre eseguire l'applicazione in Flash o in Flex per verificare che tutto funzioni nel modo previsto.

Tenere presente che non è indispensabile eseguire questi passaggi nell'ordine in cui sono presentati né completare una fase prima di passare alla successiva. Ad esempio, si può cominciare progettando una schermata dell'applicazione (punto 1), quindi creare la grafica, i pulsanti eccetera (punto 3), poi scrivere il codice ActionScript (punto 2) e infine testare l'applicazione (punto 4). In alternativa, si può progettare una parte dell'applicazione, aggiungere un pulsante o un elemento dell'interfaccia per volta, scrivere il codice ActionScript per ogni elemento e testare il risultato dopo averlo generato. Nonostante sia importante avere ben chiare queste quattro fasi del processo di sviluppo, nel concreto è spesso più utile spostarsi da una all'altra in base alla necessità del momento.

Creazione di classi personalizzate

Il processo di creazione delle classi da utilizzare nei progetti può incutere un certo timore. Tuttavia, la parte più impegnativa del processo è la progettazione della classe, vale a dire individuare i metodi, le proprietà e gli eventi che dovrà comprendere.

Strategie per la progettazione di una classe

La progettazione orientata agli oggetti è una disciplina molto complessa, a cui studiosi hanno consacrato intere carriere accademiche e professionali. Tuttavia, è possibile suggerire alcuni approcci per aiutare l'utente a orientarsi.

1. Considerare il ruolo che le istanze di questa classe dovranno svolgere all'interno dell'applicazione. Generalmente, gli oggetti espletano una di questi tre funzioni:
   - Oggetto contenitore di valori: si tratta di oggetti destinati a contenere dei valori e sono caratterizzati da varie proprietà e pochi metodi (a volte nessun metodo). Sono spesso rappresentazioni in codice di elementi chiaramente definiti, come ad esempio una canzone (classe Song) o un gruppo di canzoni (classe Playlist) all'interno di un'applicazione di riproduzione musicale.
   - Oggetto di visualizzazione: oggetti che vengono visualizzati sullo schermo, come elementi dell'interfaccia utente quali elenchi a discesa o comunicazioni delle stato, elementi grafici quali creature di un video game eccetera.
   - Struttura dell'applicazione: oggetti che espletano vari ruoli di supporto nell'ambito della logica e dell'elaborazione eseguita dalle applicazioni. Alcuni esempi: un oggetto che esegue determinati calcoli in una simulazione di biologia; un oggetto responsabile della sincronizzazione di valori tra il comando del quadrante e il valore del volume in un'applicazione di riproduzione musicale; un oggetto che gestisce le regole di un video game; un oggetto che carica un'immagine salvata in un'applicazione di disegno.
2. Scegliere la funzionalità che dovrà svolgere la classe. I vari tipi di funzionalità spesso diventano i metodi della classe.
3. Se la classe fungerà da oggetto contenitore di valori, scegliere i dati che le istanze conterranno. Queste voci sono buoni candidati da convertire in proprietà.

4. Poiché la classe viene progettata espressamente per un progetto, quello che più conta è assegnarle la funzionalità che l'applicazione richiede. Può essere utile porsi queste domande:
   - Che tipo di informazioni verranno memorizzate, monitorate e manipolate dall'applicazione? La risposta a questa domanda consente di individuare gli oggetti contenitore e le proprietà necessari.
   - Che azioni verranno eseguite, ad esempio, quando si carica l'applicazione, quando si seleziona un pulsante specifico, quando si interrompe la riproduzione di un filmato e così via? Queste azioni sono candidati eccellenti per i metodi (o per le proprietà, se le "azioni" prevedono solo la modifica di valori singoli).
   - Per eseguire ogni singola azione, di che informazioni ha bisogno la classe? Questi dati diventeranno i parametri del metodo.
   - Durante l'esecuzione dell'applicazione, quali cambiamenti che si verificano nella classe devono essere comunicati alle altre parti dell'applicazione? Questi sono ottimi candidati per gli eventi.
5. Se esiste un oggetto simile a quello che si vorrebbe creare ma a cui mancano alcune funzionalità, valutare la possibilità di creare una sottoclasse, cioè una classe che amplia la funzionalità di una classe esistente, piuttosto che creare una classe completamente nuova. Ad esempio, per creare una classe che funga da oggetto visivo dello schermo, si può usare come base il comportamento di un oggetto di visualizzazione esistente (come Sprite o MovieClip). In questo caso, MovieClip (o Sprite) sarebbe la *classe base* la cui funzionalità verrebbe ampliata nella nuova classe. Per ulteriori informazioni sulla creazione delle sottoclassi, vedere "Ereditarietà" a pagina 174.

Scrittura del codice per una classe

Una volta progettata la classe, o perlomeno, una volta stabilite le informazioni di cui tenere traccia e le azioni da eseguire, la sintassi da usare per scrivere il codice della classe è abbastanza semplice.

Ecco i passaggi fondamentali da seguire per creare una classe ActionScript:

1. Aprire un nuovo file di testo usando un programma che supporta ActionScript, come Flex Builder o Flash, in un'applicazione di programmazione generica tipo Dreamweaver o in un'applicazione che supporta i documenti di solo testo.

2. Immettere un'istruzione `class` per definire il nome della classe digitando `public class` seguito dal nome della classe, seguito, a sua volta, da due parentesi graffe per racchiudere il contenuto della classe (il metodo e le proprietà). Ad esempio:

```
public class MyClass
{
}
```

   Il termine `public` indica che la classe è accessibile da qualsiasi altra riga di codice. Per alternative, vedere "Attributi dello spazio dei nomi per il controllo dell'accesso" a pagina 154.

3. Digitare un'istruzione che indichi il nome del pacchetto che ospiterà il proprio pacchetto. La sintassi è composta dalla parola `package`, seguita dal nome assegnato al pacchetto seguito, a sua volta, da due parentesi graffe (che racchiudono il blocco di istruzioni `class`). Ad esempio, intervenendo sul codice dell'esempio precedente si ottiene:

```
package mypackage
{
  public class MyClass
  {
  }
}
```

4. Definire tutte le proprietà della classe usando l'istruzione `var` all'interno del corpo della classe; la sintassi corrisponde a quella utilizzata per dichiarare una variabile (con l'aggiunta del modificatore `public`). Ad esempio, l'aggiunta di queste righe tra le parentesi graffe della definizione della classe crea le proprietà `textVariable`, `numericVariable` e `dateVariable`:

```
public var textVariable:String = "some default value";
public var numericVariable:Number = 17;
public var dateVariable:Date;
```

5. Definire tutti i metodi della classe replicando la sintassi utilizzata per definire una funzione. Ad esempio:

   - Per creare il metodo `myMethod()`, digitare:

```
public function myMethod(param1:String, param2:Number):void
{
  // esegue un'operazione con i parametri
}
```

- Per creare il metodo costruttore, cioè lo speciale metodo invocato come parte del processo di creazione di un'istanza di una classe, creare un metodo a cui assegnare l'esatto nome della classe:

```
public function MyClass()
{
  // imposta il valore iniziale delle proprietà
  // oppure imposta l'oggetto
  textVariable = "Hello there!";
  dateVariable = new Date(2001, 5, 11);
}
```

  Se non si definisce un metodo costruttore per la classe, il compilatore ne crea automaticamente uno vuoto, cioè uno sprovvisto di parametri e istruzioni.

Esistono alcuni altri elementi che si possono definire per la classe. Si tratta di elementi più complessi.

- Gli *accessor* sono vie di mezzo tra i metodi e le proprietà. Nel codice che definisce una classe, l'accessor viene scritto come un metodo per eseguire varie azioni, al contrario di una proprietà che si limita a leggere o ad assegnare un valore. Tuttavia, quando si crea un'istanza della classe, l'accessor viene trattato come una proprietà e se ne usa il nome per leggere o assegnare il valore. Per ulteriori informazioni, vedere "Metodi supplementari get e set" a pagina 164.
- In ActionScript, la definizione degli eventi non richiede una sintassi particolare. Infatti, è sufficiente utilizzare la funzionalità della classe EventDispatcher per tenere traccia dei listener di eventi e per comunicare loro gli eventi. Per ulteriori informazioni sulla creazione di eventi all'interno delle classi, vedere Capitolo 10, "Gestione degli eventi" a pagina 335.

Suggerimenti per l'organizzazione delle classi

Contrariamente a quanto avveniva nelle versioni precedenti di ActionScript, ActionScript 3.0 non impone più il limite di una classe per file. Con ActionScript 3.0 è possibile salvare il codice di origine per più classi in un unico file .as. In alcuni casi può sembrare utile impacchettare più classi in un unico file di origine, ma in genere questa pratica di programmazione è sconsigliata per un paio di motivi:

- È difficile riutilizzare singole classi se sono impacchettate insieme in un unico file di grandi dimensioni.
- È difficile individuare il codice di origine di una singola classe se il nome del file non corrisponde a quello della classe.

Per questi motivi Adobe consiglia di salvare sempre il codice di origine di ciascuna classe in un file distinto con lo stesso nome della classe.

Esempio: Creazione di un'applicazione di base

È possibile creare file di origine ActionScript esterni con estensione .as utilizzando Flash, Flex Builder, Dreamweaver o qualsiasi editor di testo.

ActionScript 3.0 può essere utilizzato in vari ambienti di sviluppo diversi, tra cui Flash e Flex Builder.

Questa sezione guida il lettore attraverso le varie fasi necessarie per creare e perfezionare una semplice applicazione ActionScript 3.0 utilizzando lo strumento di creazione di Flash o Flex Builder 2. L'applicazione da realizzare presenta un metodo semplice per utilizzare file di classe ActionScript 3.0 esterni nelle applicazioni Flash e Flex. Questo metodo è valido anche per tutte le altre applicazioni di esempio illustrate nel manuale.

Progettazione dell'applicazione ActionScript

È importante ideare l'applicazione da realizzare prima di iniziare a svilupparla.

La rappresentazione del progetto può essere estremamente semplice, ad esempio solo il nome dell'applicazione e una breve descrizione del suo scopo, oppure complessa, ad esempio una serie di documenti di requisiti contenenti vari diagrammi UML (Unified Modeling Language). Questo manuale non entra nei dettagli della progettazione software e delle relative procedure, ma è importante sapere che la fase progettuale è essenziale nello sviluppo di applicazioni ActionScript.

Il primo esempio di applicazione ActionScript è un'applicazione "Hello World" standard, al cui progettazione è quindi estremamente semplice:

- L'applicazione si chiamerà HelloWorld.
- Visualizzerà un unico campo di testo contenente le parole "Hello World!".
- Per facilitarne il riutilizzo, farà uso di una singola classe a oggetti, di nome Greeter, utilizzabile in un documento Flash o in un'applicazione Flex.
- Dopo aver creato una versione di base dell'applicazione, si procederà all'aggiunta di nuove funzionalità per consentire all'utente di immettere un nome utente e all'applicazione di verificare la presenza del nome specificato in un elenco di utenti conosciuti.

Con questa semplice definizione del progetto, è possibile iniziare a realizzare l'applicazione vera e propria.

Creazione del progetto HelloWorld e della classe Greeter

La definizione del progetto per l'applicazione Hello World afferma che il codice deve essere facile da riutilizzare. Sulla base di questo obiettivo, l'applicazione impiega una singola classe a oggetti, chiamata Greeter, che viene utilizzata all'interno di un'applicazione creata in Flex Builder o nello strumento di creazione di Flash.

Per creare la classe Greeter nello strumento di creazione di Flash:

1. In Flash, selezionare File > Nuovo.
2. Nella finestra di dialogo Nuovo documento, selezionare File ActionScript e fare clic su OK.

   Viene visualizzata una nuova finestra di modifica ActionScript.
3. Selezionare File > Salva. Selezionare una cartella di destinazione per l'applicazione, chiamare il file **Greeter.as** e fare clic su OK.

   Procedere con la sezione "Aggiunta di codice alla classe Greeter" a pagina 62.

Aggiunta di codice alla classe Greeter

La classe Greeter definisce un oggetto, `Greeter`, che potrà essere utilizzato nell'applicazione HelloWorld.

Per aggiungere codice alla classe Greeter:

1. Immettere il codice seguente in un nuovo file:

```
package
{
  public class Greeter
  {
    public function sayHello():String
    {
      var greeting:String;
      greeting = "Hello World!";
      return greeting;
    }
  }
}
```

   La classe Greeter include un unico metodo `sayHello()`, che restituisce una stringa con il testo "Hello" al nome utente che viene specificato.
2. Selezionare File > Salva per salvare il file ActionScript.

La classe Greeter è ora pronta per essere utilizzata in un'applicazione Flash o Flex.

Creazione di un'applicazione che utilizza il codice ActionScript

La classe Greeter appena creata definisce un set autonomo di funzioni software, ma non rappresenta un'applicazione completa. Per utilizzarla, è necessario creare un documento Flash o un'applicazione Flex.

L'applicazione HelloWorld crea una nuova istanza della classe Greeter. Di seguito viene spiegato come associare la classe Greeter alla propria applicazione.

Per creare un'applicazione ActionScript con lo strumento di creazione di Flash:

1. Selezionare File > Nuovo.
2. Nella finestra di dialogo Nuovo documento, selezionare Documento Flash e fare clic su OK.

   Viene visualizzata una nuova finestra di Flash.
3. Selezionare File > Salva. Selezionare la stessa cartella in cui si trova il file di classe Greeter.as, denominare il file **HelloWorld.fla** e fare clic su OK.
4. Nel pannello Strumenti di Flash, selezionare lo strumento Testo e trascinare il puntatore sullo stage in modo da definire un nuovo campo di testo largo circa 300 pixel e alto circa 100 pixel.
5. Nella finestra Proprietà, con il campo di testo selezionato sullo stage, digitare `mainText` come nome di istanza del campo di testo.
6. Fare clic sul primo fotogramma della linea temporale.
7. Nel pannello Azioni, digitare il seguente script:

   ```
   var myGreeter:Greeter = new Greeter();
   mainText.text = myGreeter.sayHello("Bob");
   ```
8. Salvare il file.

Procedere con la sezione "Pubblicazione e prova dell'applicazione ActionScript" a pagina 64.

Pubblicazione e prova dell'applicazione ActionScript

Lo sviluppo di software è un processo iterativo. Il codice viene scritto, compilato e corretto finché la compilazione non risulta priva di errori. L'applicazione compilata viene quindi eseguita e provata per verificare che soddisfi l'obiettivo per cui è stata progettata; in caso contrario, si modifica il codice fino a ottenere il risultato previsto. Gli ambienti di sviluppo Flash e Flex Builder offrono vari modi per pubblicare e provare le applicazioni ed eseguirne il debug.

Di seguito è indicata la procedura di base da seguire per provare l'applicazione HelloWorld in ciascun ambiente.

Per pubblicare e provare un'applicazione ActionScript con lo strumento di creazione di Flash:

1. Pubblicare l'applicazione e verificare se vengono generati errori di compilazione. Nello strumento di creazione di Flash, selezionare Controllo > Prova filmato per compilare il codice ActionScript ed eseguire l'applicazione HelloWorld.
2. Se nel pannello Output vengono visualizzati errori o avvisi durante la prova dell'applicazione, correggere le cause di tali problemi nel file HelloWorld.fla o HelloWorld.as ed eseguire una nuova prova dell'applicazione.
3. Se non vi sono errori di compilazione, appare una finestra di Flash Player con l'applicazione Hello World. Nella finestra è visualizzato il testo "Hello, Bob".

A questo punto è stata creata una semplice ma completa applicazione orientata agli oggetti, che utilizza ActionScript 3.0. Procedere con la sezione "Perfezionamento dell'applicazione HelloWorld" a pagina 64.

Perfezionamento dell'applicazione HelloWorld

Per rendere un po' più interessante l'applicazione, si può fare in modo che chieda un nome utente e convalidi il nome specificato confrontandolo con un elenco di nomi predefinito.

Innanzi tutto occorre aggiornare la classe Greeter aggiungendo una nuova funzionalità, per poi aggiornare l'applicazione Flash o Flex in modo che utilizzi tale funzionalità.

Per aggiornare il file Greeter.as:

1. Aprire il file Greeter.as.

2. Modificare il contenuto del file in base all'esempio seguente (le righe nuove o modificate sono evidenziate in grassetto):

```
package
{
  public class Greeter
  {
    /**
     * Defines the names that should receive a proper greeting.
     */
    public static var validNames:Array = ["Sammy", "Frank", "Dean"];

    /**
     * Builds a greeting string using the given name.
     */
    public function sayHello(userName:String = ""):String
    {
      var greeting:String;
      if (userName == "")
      {
        greeting = "Hello. Please type your user name, and then press
the Enter key.";
      }
      else if (validName(userName))
      {
        greeting = "Hello, " + userName + ".";
      }
      else
      {
        greeting = "Sorry, " + userName + ", you are not on the list.";
      }
      return greeting;
    }

    /**
     * Checks whether a name is in the validNames list.
     */
    public static function validName(inputName:String = ""):Boolean
    {
      if (validNames.indexOf(inputName) > -1)
      {
        return true;
      }
      else
      {
        return false;
      }
    }
  }
}
```

A questo punto la classe Greeter presenta alcune nuove caratteristiche:

- L'array `validNames` elenca i nomi utente validi. L'array viene inizializzato su un elenco di tre nomi nel momento in cui la classe Greeter viene caricata.
- Il metodo `sayHello()` ora accetta un nome utente e modifica il saluto in base ad alcune condizioni. Se `userName` è una stringa vuota (`""`), la proprietà `greeting` viene impostata in modo da richiedere un nome all'utente. Se il nome utente è valido, il saluto diventa `"Hello, nomeUtente."` Infine, se nessuna di queste due condizioni è soddisfatta, la variabile `greeting` viene impostata su `"Sorry, nomeUtente, you are not on the list."`
- Il metodo `validName()` restituisce `true` se `inputName` viene trovato nell'elenco `validNames` oppure `false` se non viene trovato. L'istruzione `validNames.indexOf(inputName)` confronta ciascuna delle stringhe dell'array `validNames` con la stringa `inputName`. Il metodo `Array.indexOf()` restituisce la posizione di indice della prima istanza di un oggetto di un array, oppure il valore -1 se l'oggetto non viene trovato nell'array.

A questo punto si passa a modificare il file Flash o Flex che fa riferimento a questa classe ActionScript.

Per modificare l'applicazione utilizzando lo strumento di creazione di Flash:

1. Aprire il file HelloWorld.fla.
2. Modificare lo script nel fotogramma 1 in modo che venga passata una stringa vuota (`""`) al metodo `sayHello()` della classe Greeter:

```
var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello("");
```

3. Selezionare lo strumento Testo nel pannello Strumenti, quindi creare due nuovi campi di testo sullo stage, uno accanto all'altro e direttamente sotto il campo `mainText` già presente.
4. Nel primo campo di testo, digitare il testo **User Name:** come etichetta.
5. Selezionare l'altro campo di testo aggiunto e, nella finestra di ispezione Proprietà, selezionare Testo di input come tipo di campo. Digitare `textIn` come nome di istanza.
6. Fare clic sul primo fotogramma della linea temporale principale.

7. Nel pannello Azioni, aggiungere le seguenti righe alla fine dello script esistente:

```
mainText.border = true;
textIn.border = true;

textIn.addEventListener(KeyboardEvent.KEY_UP, keyPressed);

function keyPressed(event:Event):void
{
  if (event.keyCode == Keyboard.ENTER)
  {
    mainText.text = myGreeter.sayHello(textIn.text);
  }
}
```

Il nuovo codice aggiunge la seguente funzionalità:

- Le prime due righe definiscono semplicemente i bordi dei due campi di testo.
- Un campo di testo di input, come `textIn`, può inviare una serie di eventi. Il metodo `addEventListener()` permette di definire una funzione che viene eseguita quando si verifica un determinato tipo di evento. In questo caso, l'evento è la pressione del tasto Invio della tastiera.
- La funzione personalizzata `keyPressed()` chiama il metodo `sayHello()` dell'oggetto `myGreeter`, passando il testo del campo di testo `textIn` come parametro. Tale metodo restituisce la stringa greeting basata sul valore passato. La stringa restituita viene quindi assegnata alla proprietà `text` del campo di testo `mainText`.

Lo script completo del fotogramma 1 è il seguente:

```
mainText.border = true;
textIn.border = true;

var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello("");

textIn.addEventListener(KeyboardEvent.KEY_UP, keyPressed);

function keyPressed(event:Event):void
{
  if (event.keyCode == Keyboard.ENTER)
  {
    mainText.text = myGreeter.sayHello(textIn.text);
  }
}
```

8. Salvare il file.
9. Selezionare Controllo > Prova filmato per eseguire l'applicazione.

   Viene richiesto di immettere un nome utente. Se il nome immesso è valido (Sammy, Frank o Dean), l'applicazione visualizza il messaggio di conferma "hello".

Esempi successivi

Dopo aver sviluppato ed eseguito l'applicazione ActionScript 3.0 "Hello World", si posseggono già le conoscenze di base necessarie per eseguire gli altri esempi di codice presentati nel manuale.

Prova degli esempi di codice contenuti nei capitoli

Mentre si consulta il manuale, è possibile provare gli esempi di codice utilizzati per illustrare i vari argomenti. Questa prova può consistere nel visualizzare il valore delle variabili in punti specifici del programma oppure nel visualizzare o nell'interagire con il contenuto visualizzato sullo schermo. Per provare l'interazione o il contenuto visivo, gli elementi richiesti vengono indicati prima o all'interno del codice: è sufficiente creare un documento con gli elementi descritti. Per visualizzare il valore di una variabile in un punto specifico nel programma, è possibile procedere in vari modi. Ad esempio, utilizzare una funzione di debug, come quella incorporata in Flex Builder e Flash. Tuttavia, il modo più semplice consiste nello stampare i valori delle variabili in una determinata posizione in cui siano visibili.

La procedura seguente spiega come creare un documento Flash da utilizzare per la prova di un codice e la visualizzazione di valori di variabili:

Per creare un documento Flash per provare gli esempi contenuti nei capitoli:

1. Creare un documento Flash e salvarlo sul disco rigido.
2. Per visualizzare i valori di prova in un campo di testo sullo stage, attivare lo strumento Testo e creare un nuovo campo di testo dinamico sullo stage. L'ideale è un campo di testo largo e alto con il tipo di riga Multiriga e il bordo attivato. Nella finestra di ispezione Proprietà, assegnare al campo di testo un nome di istanza (ad esempio "outputText"). Per scrivere i valori nel campo di testo, aggiungere al codice di esempio un codice che chiama il metodo `appendText()` (come descritto sotto).
3. In alternativa, per visualizzare i risultati dell'esempio, è possibile aggiungere al codice di esempio una chiamata alla funzione `trace()` (come descritto sotto).
4. Per provare un determinato esempio, copiare il codice nel pannello Azioni; se necessario, aggiungere una chiamata alla funzione `trace()` oppure un valore al campo di testo utilizzando il relativo metodo `appendText()`.
5. Dal menu principale, selezionare Controllo > Prova filmato per creare un file SWF e visualizzare i risultati.

Esistono due modi per visualizzare i valori delle variabili mentre si effettua la prova degli esempi: scrivere i valori in un'istanza di campo di testo sullo stage oppure utilizzare la funzione `trace()` per stampare i valori sul pannello Output.

- La funzione `trace()`: la funzione `trace()` di ActionScript scrive il valore di qualsiasi parametro che le viene inviato (variabile o espressione letterale) nel pannello Output. Molti esempi di codice contenuti nel manuale includono già una chiamata alla funzione `trace()`, pertanto è sufficiente copiare tali codici nel documento e provare il progetto. Per utilizzare la funzione `trace()` per provare il valore di una variabile contenuta in un codice che non la include, aggiungere al codice una chiamata a `trace()`, che passa la variabile come parametro. Ad esempio, nel caso di un codice come quello riportato in questo capitolo,

  ```
  var albumName:String = "Three for the money";
  ```

  è possibile copiare il codice nel pannello Azioni, quindi aggiungere una chiamata alla funzione `trace()`, come mostrato di seguito, per provare il risultato del codice:

  ```
  var albumName:String = "Three for the money";
  trace("albumName =", albumName);
  ```

  Quando si esegue il programma, viene stampata la riga seguente:

  ```
  albumName = Three for the money
  ```

  Ciascuna chiamata alla funzione `trace()` può richiedere più parametri che vengono legati tra loro per formare una singola riga stampata. Al termine di ciascuna chiamata viene aggiunta un'interruzione di riga, in modo che le chiamate separate vengano stampate su righe separate.
- Un campo di testo sullo stage: se si preferisce non utilizzare la funzione `trace()`, è possibile creare un campo di testo dinamico sullo stage mediante lo strumento Testo e scrivere i valori in questo campo per visualizzare i risultati di un codice. Il metodo `appendText()` della classe TextField può essere utilizzato per aggiungere un valore stringa al termine del contenuto del campo di testo. Per accedere a un campo di testo utilizzando ActionScript, assegnarvi un nome di istanza nella finestra di ispezione Proprietà.
  Ad esempio, se il campo di testo presenta il nome di istanza `outputText`, il codice seguente può essere utilizzato per controllare il valore della variabile `albumName`:

  ```
  var albumName:String = "Three for the money";
  outputText.appendText("albumName = ");
  outputText.appendText(albumName);
  ```

  Questo codice scrive il testo seguente nel campo di testo `outputText`:

  ```
  albumName = Three for the money
  ```

Come indica l'esempio, il metodo `appendText()` aggiunge il testo nella stessa riga del contenuto precedente, pertanto è possibile aggiungere più valori alla stessa riga di testo utilizzando più chiamate al metodo `appendText()`. Per fare in modo che il testo venga scritto nella riga successiva, è possibile aggiungere un carattere nuova riga (`"\n"`):

```
outputText.appendText("\n"); // adds a line break to the text field
```

A differenza della funzione `trace()`, il metodo `appendText()` accetta esclusivamente un valore come parametro. Questo valore deve essere una stringa (un'istanza String oppure un valore letterale di stringa). Per stampare il valore di una variabile non stringa, è necessario prima convertire il valore in una stringa. A tal scopo, il modo più semplice è chiamare il metodo `toString()` dell'oggetto:

```
var albumYear:int = 1999;
outputText.appendText("albumYear = ");
outputText.appendText(albumYear.toString());
```

Operazioni con gli esempi di fine capitolo

La maggior parte dei capitoli, come il presente, si concludono con un esempio significativo che combina molti dei concetti trattati nel capitolo. A differenza dell'esempio "Hello World" di questo capitolo, tuttavia, tali esempi non vengono presentati sotto forma di esercitazione dettagliata. Per ciascun esempio viene evidenziato e discusso il codice ActionScript 3.0 da utilizzare, ma non vengono fornite istruzioni per l'esecuzione degli esempi in un ambiente di sviluppo specifico. Tuttavia, con il manuale vengono distribuiti tutti i file necessari per compilare ed eseguire gli esempi con facilità nell'ambiente di sviluppo in uso.

CAPITOLO 3

Linguaggio e sintassi ActionScript

3

ActionScript 3.0 comprende sia il linguaggio ActionScript di base che l'API di Adobe Flash Player. Il linguaggio di base è quella parte di ActionScript che implementa la bozza di specifica del nuovo linguaggio ECMAScript (ECMA-262) edizione 4. L'API di Flash Player fornisce l'accesso programmatico a Flash Player.

Questo capitolo offre una breve introduzione al linguaggio ActionScript e alla sua sintassi. Dopo averlo letto si avrà una conoscenza di base di come lavorare con i tipi di dati e le variabili, utilizzare la sintassi corretta e controllare il flusso di dati nel programma.

Sommario

Panoramica del linguaggio

Gli oggetti costituiscono il cuore del linguaggio ActionScript 3.0, i suoi blocchi costitutivi fondamentali. Ogni variabile che si dichiara, ogni funzione che si scrive e ogni istanza di classe che si crea è un oggetto. Si può pensare a un programma ActionScript 3.0 come se fosse un gruppo di oggetti che svolgono certe attività, rispondono agli eventi e comunicano gli uni con gli altri.

I programmatori che utilizzano la programmazione a oggetti (OOP) in Java o C++ possono considerare gli oggetti come dei moduli che contengono due tipi di membri: i dati, memorizzati in variabili membro o proprietà, e i comportamenti, accessibili mediante metodi. La bozza ECMAScript edizione 4, lo standard sul quale è basato ActionScript 3.0, definisce gli oggetti in un modo simile ma leggermente diverso. Nella bozza ECMAScript, infatti, gli oggetti sono semplicemente raccolte di proprietà, le quali sono contenitori non solo di dati, ma anche di funzioni e altri oggetti. Se una funzione è associata a un oggetto in questo modo, prende il nome di metodo.

Sebbene la definizione della bozza ECMAScript possa sembrare un po' strana ai programmatori Java o C++, in pratica la definizione dei tipi di oggetto con le classi ActionScript 3.0 è molto simile al modo in cui le classi vengono definite in Java o C++. La distinzione tra le due definizioni di oggetto è importante ai fini della discussione sul modello di oggetti di ActionScript e per altri argomenti avanzati, ma nella maggior parte delle situazioni il termine *proprietà* equivale a variabile membro di classe e si contrappone a metodo. La *Guida di riferimento del linguaggio e dei componenti ActionScript 3.0*, ad esempio, usa il termine *proprietà* per indicare le variabili o le proprietà getter-setter, e il termine *metodi* per indicare le funzioni che fanno parte di una classe.

Una sottile differenza tra le classi di ActionScript e quelle di Java o C++ è rappresentata dal fatto che in ActionScript le classi non sono semplicemente entità astratte, bensì sono rappresentate da *oggetti di classe* in cui vengono memorizzate le proprietà e i metodi della classe. Ciò rende possibile l'uso di tecniche che possono sembrare inusuali ai programmatori Java e C++, come l'inclusione di istruzioni o di codice eseguibile al livello principale di una classe o di un pacchetto.

Un'altra differenza tra le classi ActionScript e le classi Java o C++ è costituita dal fatto che ogni classe ActionScript possiede un elemento chiamato *oggetto prototipo*. Nelle versioni precedenti di ActionScript, gli oggetti prototipo, collegati tra loro nelle *catene di prototipi*, costituivano insieme la base dell'intera gerarchia di ereditarietà delle classi. In ActionScript 3.0, invece, gli oggetti prototipo svolgono una funzione secondaria nel sistema dell'ereditarietà. L'oggetto prototipo può tuttavia essere ancora utile come alternativa a proprietà e metodi statici quando si desidera condividere una proprietà e il relativo valore tra tutte le istanze di una classe.

In passato, i programmatori ActionScript esperti potevano modificare direttamente la catena di prototipi con elementi di linguaggio incorporati speciali. Ora che il linguaggio fornisce un'implementazione più avanzata di un'interfaccia di programmazione basata su classi, molti di questi elementi speciali, ad esempio `__proto__` e `__resolve`, non fanno più parte del linguaggio. Inoltre, le ottimizzazioni apportate al meccanismo di ereditarietà interno, che migliorano le prestazioni di Flash Player in modo significativo, precludono l'acceso diretto a tale meccanismo.

Oggetti e classi

In ActionScript 3.0 ogni oggetto è definito da una classe. Una classe può essere considerata come un modello di base per un tipo di oggetto. Le definizioni di classe possono riguardare variabili e costanti, che memorizzano valori di dati, oppure metodi, ovvero funzioni che incorporano un comportamento associato alla classe. I valori memorizzati nelle proprietà possono essere *valori di base* (primitive values) o altri oggetti. I valori di base sono numeri, stringhe o valori booleani.

ActionScript contiene una serie di classi incorporate che fanno parte del linguaggio di base. Alcune di tali classi, quali Number, Boolean e String, rappresentano i valori di base disponibili in ActionScript. Altre, come Array, Math e XML, definiscono oggetti più complessi che fanno parte dello standard ECMAScript.

Tutte le classi, incorporate o definite dall'utente, derivano dalla classe Object. Per i programmatori che hanno già utilizzato ActionScript in precedenza è importante notare che il tipo di dati Object non è più quello predefinito, anche se tutte le altre classi derivano ancora da esso. In ActionScript 2.0 le due righe di codice seguenti erano equivalenti perché l'assenza di un'annotazione di tipo comportava che una variabile fosse del tipo Object:

```
var someObj:Object;
var someObj;
```

ActionScript 3.0 invece introduce il concetto di "variabili senza tipo", che possono essere indicate nei due modi seguenti:

```
var someObj:*;
var someObj;
```

Una variabile senza tipo non è la stessa cosa di una variabile del tipo Object. La differenza fondamentale consiste nel fatto che le variabili senza tipo possono contenere il valore speciale `undefined`, mentre una variabile del tipo Object non può.

Si possono definire classi personalizzate utilizzando la parola chiave `class`. È possibile dichiarare le proprietà delle classi in tre modi: le costanti vengono definite con la parola chiave `const`, le variabili con la parola chiave `var` e le proprietà getter/setter utilizzando gli attributi `get` e `set` in una dichiarazione di metodo. I metodi vengono dichiarati con la parola chiave `function`.

Un'istanza di una classe viene creata utilizzando l'operatore `new`. Nell'esempio seguente viene creata un'istanza della classe Data chiamata `myBirthday`.

```
var myBirthday:Date = new Date();
```

Pacchetti e spazi dei nomi

Pacchetti e spazi dei nomi sono concetti correlati. I pacchetti consentono di "impacchettare" insieme le definizioni di classe in modo da facilitare la condivisione del codice e ridurre al minimo i conflitti tra nomi. Gli spazi dei nomi permettono di controllare la visibilità degli identificatori, quali i nomi di proprietà e di metodi, e possono essere applicati al codice sia all'interno che all'esterno di un pacchetto. I pacchetti consentono di organizzare i file di classe e gli spazi dei nomi di gestire la visibilità di singole proprietà e metodi.

Pacchetti

In ActionScript 3.0, i pacchetti sono implementati insieme agli spazi dei nomi, ma non sono la stessa cosa. Quando si dichiara un pacchetto, si sta implicitamente creando un tipo speciale di spazio dei nomi con la garanzia che sarà noto in fase di compilazione. Al contrario, uno spazio dei nomi creato esplicitamente non è necessariamente noto in fase di compilazione.

Nell'esempio seguente viene utilizzata la direttiva `package` per creare un pacchetto semplice che contiene un'unica classe:

```
package samples
{
  public class SampleCode
  {
    public var sampleGreeting:String;
    public function sampleFunction()
    {
      trace(sampleGreeting + " from sampleFunction()");
    }
  }
}
```

Il nome della classe di questo esempio è SampleCode. Poiché la classe è all'interno del pacchetto samples, il compilatore qualifica automaticamente il nome della classe in fase di compilazione con il rispettivo nome completo: samples.SampleCode. Inoltre il compilatore qualifica i nomi di eventuali proprietà o metodi; di conseguenza, `sampleGreeting` e `sampleFunction()` diventano rispettivamente `samples.SampleCode.sampleGreeting` e `samples.SampleCode.sampleFunction()`.

Molti sviluppatori, specialmente quelli con esperienza Java, potrebbero scegliere di collocare solo le classi al livello principale di un pacchetto. Tuttavia, ActionScript 3.0 supporta non solo le classi in tale posizione, ma anche le variabili, le funzioni e persino le istruzioni. Uno degli usi avanzati di questa caratteristica consiste nel definire uno spazio dei nomi al livello principale di un pacchetto, in modo che sia disponibile a tutte le classi del pacchetto. Si noti, tuttavia, che soltanto due specificatori di accesso (`public` e `internal`) sono consentiti al livello principale di un pacchetto. A differenza di Java, che consente di dichiarare come private le classi nidificate, ActionScript 3.0 non supporta né le classi nidificate né quelle private.

Per molti altri aspetti, tuttavia, i pacchetti di ActionScript 3.0 sono simili a quelli del linguaggio di programmazione Java. Come si può notare nell'esempio precedente, i riferimenti ai nomi completi dei pacchetti vengono espressi utilizzando l'operatore punto (`.`), proprio come in Java. È possibile sfruttare i pacchetti per organizzare il codice in una struttura gerarchica intuitiva che possa essere utilizzata da altri programmatori. In questo modo si facilita la condivisione del codice, poiché si possono creare pacchetti personalizzati da mettere a disposizione degli altri e utilizzare nel proprio codice pacchetti creati da altri programmatori.

Inoltre, l'uso dei pacchetti fa sì che i nomi di identificazione utilizzati siano univoci e non in conflitto con quelli di altri pacchetti. Si potrebbe addirittura sostenere che questo sia il vantaggio principale dei pacchetti. Ad esempio, si supponga che due programmatori abbiano deciso di condividere il loro codice e che ciascuno dei due abbia creato una classe chiamata SampleCode. Senza i pacchetti si creerebbe un conflitto tra i nomi e l'unica soluzione sarebbe quella di rinominare una delle due classi. Grazie ai pacchetti, invece, il conflitto viene evitato facilmente collocando una o (preferibilmente) entrambe le classi in pacchetti con nomi univoci.

È anche possibile includere dei punti (.) incorporati in un pacchetto per creare pacchetti nidificati, allo scopo di realizzare un'organizzazione gerarchica dei pacchetti. Un buon esempio di questo caso è il pacchetto flash.xml dell'API di Flash Player, che è nidificato all'interno del pacchetto flash.

Il pacchetto flash.xml contiene il parser XML (utilità di analisi sintattica) convenzionale utilizzato nelle versioni precedenti di ActionScript. Uno dei motivi per cui è stato ora incluso nel pacchetto flash.xml è il conflitto tra il nome della vecchia classe XML e il nome della nuova classe XML che implementa la funzionalità della specifica XML per ECMAScript (E4X) disponibile in ActionScript 3.0.

Sebbene lo spostamento della vecchia classe XML in un pacchetto sia un buon accorgimento, la maggior parte degli utenti delle classi XML tradizionali importerà il pacchetto flash.xml, generando così lo stesso conflitto di nomi, a meno che non si ricordi di utilizzare sempre il nome completo della classe XML tradizionale (flash.xml.XML). Per evitare questa situazione, la vecchia classe XML è stata rinominata in XMLDocument, come illustrato nell'esempio che segue:

```
package flash.xml
{
  class XMLDocument {}
  class XMLNode {}
  class XMLSocket {}
}
```

L'API di Flash Player è organizzata quasi interamente all'interno del pacchetto flash. Ad esempio, il pacchetto flash.display contiene l'API dell'elenco di visualizzazione, e il pacchetto flash.events contiene il nuovo modello di eventi.

Creazione di pacchetti

ActionScript 3.0 offre una notevole flessibilità nell'organizzazione di pacchetti, classi e file di origine. Le versioni precedenti di ActionScript consentivano l'uso di un'unica classe per ogni file di origine e imponevano che il nome del file di origine corrispondesse a quello della classe. ActionScript 3.0 invece permette di includere più classi nello stesso file di origine, ma una sola classe di ogni file può essere resa disponibile al codice esterno al file stesso. In altre parole, solo una classe di ogni file può essere dichiarata all'interno di una dichiarazione di pacchetto. Le eventuali altre classi devono essere dichiarate all'esterno, pertanto risultano invisibili al codice esterno al file di origine. Il nome della classe dichiarata nella definizione del pacchetto deve corrispondere al nome del file di origine.

La maggiore flessibilità di ActionScript 3.0 si può notare anche nel modo in cui vengono definiti i pacchetti. Nelle versioni precedenti di ActionScript, i pacchetti rappresentavano semplicemente delle directory in cui venivano inseriti i file di origine e non venivano dichiarati con l'istruzione `package`, bensì si includeva il nome del pacchetto nel nome di classe completo all'interno della dichiarazione di classe. In ActionScript 3.0 i pacchetti rappresentano ancora delle directory, ma il loro contenuto non è limitato alle classi. L'istruzione `package` di ActionScript 3.0 consente di dichiarare non solo un pacchetto ma anche variabili, funzioni e spazi dei nomi al livello principale di un pacchetto. È anche possibile includere istruzioni eseguibili, sempre al livello principale del pacchetto. Se si sceglie di dichiarare variabili, funzioni o spazi dei nomi al livello principale di un pacchetto, gli unici attributi disponibili a quel livello sono `public` e `internal`, e soltanto una dichiarazione a livello di pacchetto per ciascun file può utilizzare l'attributo `public`, a prescindere che venga dichiarata una classe, una variabile, una funzione o uno spazio dei nomi.

I pacchetti sono utili per organizzare il codice e per evitare conflitti tra i nomi. Il concetto di pacchetto non va confuso con quello di ereditarietà delle classi, al quale non è correlato. Due classi che risiedono nello stesso pacchetto hanno uno spazio dei nomi in comune, ma non sono necessariamente correlate tra loro in alcun altro modo. Analogamente, un pacchetto nidificato potrebbe non avere alcune relazione semantica con il pacchetto di livello superiore.

Importazione dei pacchetti

Per utilizzare una classe che si trova all'interno di un pacchetto, occorre importare il pacchetto stesso o la classe specifica. In ActionScript 2.0, invece, l'importazione delle classi era opzionale.

Ad esempio, si consideri la classe SampleCode illustrata in una parte precedente del capitolo. Se la classe risiede in un pacchetto chiamato samples, è necessario utilizzare una delle seguenti istruzioni import per utilizzare la classe SampleCode:

```
import samples.*;
```

oppure

```
import samples.SampleCode;
```

In generale, le istruzioni `import` devono essere scritte nel modo più specifico possibile. Se si prevede di utilizzare solo la classe SampleCode del pacchetto `samples`, si deve importare solo la classe SampleCode anziché l'intero pacchetto al quale appartiene. L'importazione di interi pacchetti può determinare conflitti imprevisti tra i nomi.

Inoltre, è necessario collocare il codice che definisce il pacchetto o la classe all'interno del *percorso di classe*. Il percorso di classe (classpath) è un elenco definito dall'utente dei percorsi di directory locali, che determina dove il compilatore cercherà i pacchetti e le classi importate. Talvolta il percorso di classe viene anche definito *percorso di compilazione (build path)* o *percorso di origine (source path)*.

Dopo aver importato correttamente la classe o il pacchetto, è possibile utilizzare il nome completo della classe (samples.SampleCode) oppure semplicemente il nome della classe (SampleCode).

I nomi completi sono utili quando classi, metodi o proprietà con nomi identici possono creare ambiguità nel codice, ma possono risultare più difficili da gestire se utilizzati per tutti gli identificatori. Ad esempio, l'uso del nome completo determina un codice troppo verboso quando si crea un'istanza della classe SampleCode:

```
var mySample:samples.SampleCode = new samples.SampleCode();
```

Man mano che aumentano i livelli di nidificazione dei pacchetti, diminuisce la leggibilità del codice. Nei casi in cui si è sicuri che la presenza di identificatori ambigui non rappresenta un problema, l'uso degli identificatori semplici consente di ottenere codice più leggibile.
Ad esempio, il codice che crea un'istanza della classe SampleCode è molto più semplice se si include solo l'identificatore di classe:

```
var mySample:SampleCode = new SampleCode();
```

Se si tenta di utilizzare nomi di identificatori senza aver prima importato il pacchetto o la classe corrispondente, il compilatore non è in grado di trovare le definizioni di classe. D'altro canto, tuttavia, se si importa un pacchetto o una classe e si tenta di definire un nome che è in conflitto con un nome importato, viene generato un errore.

Quando si crea un pacchetto, lo specificatore di accesso predefinito per tutti i membri del pacchetto è `internal`, vale a dire che, per impostazione predefinita, tutti i membri del pacchetto sono visibili soltanto agli altri membri dello stesso pacchetto. Se si desidera che una classe sia disponibile al codice esterno al pacchetto, deve essere dichiarata come `public`.
Ad esempio, il pacchetto seguente contiene due classi, SampleCode e CodeFormatter:

```
// file SampleCode.as
package samples
{
  public class SampleCode {}
}

// file CodeFormatter.as
package samples
{
  class CodeFormatter {}
}
```

La classe SampleCode è visibile al di fuori del pacchetto perché è stata dichiarata con l'attributo `public`. La classe CodeFormatter class, al contrario, è visibile solo nel pacchetto samples. Se si tenta di accedere alla classe CodeFormatter esternamente al pacchetto samples, viene generato un errore, come nell'esempio seguente:

```
import samples.SampleCode;
import samples.CodeFormatter;
var mySample:SampleCode = new SampleCode(); // ok, classe pubblica
var myFormatter:CodeFormatter = new CodeFormatter(); // Errore
```

Se si desidera che ambedue le classi siano disponibili al codice esterno al pacchetto, occorre dichiarare entrambe come `public`. Non è possibile applicare l'attributo `public` alla dichiarazione del pacchetto.

I nomi completi sono utili per risolvere conflitti tra nomi che potrebbero verificarsi quando si usano i pacchetti. Uno scenario di questo tipo si potrebbe presentare se si importano due pacchetti che definiscono classi con lo stesso identificatore. Ad esempio, esaminare il pacchetto seguente, che contiene a sua volta una classe chiamata SampleCode:

```
package langref.samples
{
  public class SampleCode {}
}
```

Se si importano entrambe le classi, come nell'esempio seguente, si determina un conflitto di nomi quando si fa riferimento alla classe SampleCode:

```
import samples.SampleCode;
import langref.samples.SampleCode;
var mySample:SampleCode = new SampleCode(); // conflitto tra nomi
```

Il compilatore non ha modo di sapere quale classe SampleCode deve essere utilizzata. Per risolvere il conflitto, è necessario utilizzare il nome completo di ciascuna classe, come indicato di seguito:

```
var sample1:samples.SampleCode = new samples.SampleCode();
var sample2:langref.samples.SampleCode = new langref.samples.SampleCode();
```

NOTA

I programmatori con esperienza in C++ spesso confondono l'istruzione `import` con `#include`. La direttiva `#include` è necessaria in C++ perché i compilatori C++ elaborano un file alla volta e non cercano le definizioni di classe in altri file a meno che non venga incluso esplicitamente un file di intestazione. Anche in ActionScript 3.0 è presente una direttiva `include`, che tuttavia non è progettata per l'importazione di classi e pacchetti. Per importare classi o pacchetti in ActionScript 3.0, è necessario utilizzare l'istruzione `import` e inserire nel percorso di classe il file di origine che contiene il pacchetto.

Spazi dei nomi

Gli spazi dei nomi permettono di controllare la visibilità delle proprietà e dei metodi creati dal programmatore. Gli specificatori di controllo dell'accesso `public`, `private`, `protected` e `internal` possono essere considerati come spazi dei nomi incorporati. Se si riscontra che questi specificatori predefiniti non soddisfano tutte le proprie esigenze, è possibile creare spazi dei nomi personalizzati.

Se si ha esperienza con gli spazi dei nomi XML, questo argomento risulterà già noto, benché la sintassi e i dettagli dell'implementazione ActionScript siano leggermente diversi rispetto al linguaggio XML. Se è la prima volta che si utilizzano gli spazi dei nomi, il concetto in sé è di facile comprensione ma l'implementazione prevede una terminologia specifica che occorrerà apprendere.

Per capire come funzionano gli spazi dei nomi, è utile sapere che il nome di una proprietà o di un metodo contiene sempre due parti: un identificatore e uno spazio dei nomi. L'identificatore può essere considerato equivalente a un nome. Ad esempio, gli identificatori nella seguente definizione di classe sono `sampleGreeting` e `sampleFunction()`:

```
class SampleCode
{
  var sampleGreeting:String;
  function sampleFunction () {
    trace(sampleGreeting + " from sampleFunction()");
  }
}
```

Quando una definizione non è preceduta da un attributo namespace, il suo nome viene qualificato dallo spazio dei nomi predefinito `internal`, in base al quale la definizione è visibile unicamente ai chiamanti interni allo stesso pacchetto. Se è impostata la modalità rigorosa, il compilatore genera un'avvertenza per segnalare che lo spazio dei nomi `internal` viene applicato a tutti gli identificatori privi di attributo namespace. Per far sì che un identificatore sia disponibile senza limiti, è necessario anteporre esplicitamente al suo nome l'attributo `public`. Nell'esempio di codice precedente, `sampleGreeting` e `sampleFunction()` hanno entrambi il valore `internal` per lo spazio dei nomi.

L'utilizzo degli spazi dei nomi prevede tre passaggi fondamentali. Innanzi tutto, occorre definire lo spazio dei nomi mediante la parola chiave `namespace`. Ad esempio, il codice seguente definisce lo spazio dei nomi `version1`:

```
namespace version1;
```

In secondo luogo, si applica lo spazio dei nomi utilizzandolo al posto di uno specificatore di accesso in una proprietà o nella dichiarazione di un metodo. L'esempio seguente inserisce una funzione denominata `myFunction()` nello spazio dei nomi `version1`:

```
version1 function myFunction() {}
```

Infine, una volta applicato lo spazio dei nomi, è possibile farvi riferimento mediante la direttiva `use` oppure qualificando il nome di un identificatore con lo spazio dei nomi. L'esempio seguente fa riferimento alla funzione `myFunction()` mediante la direttiva `use`:

```
use namespace version1;
myFunction();
```

È possibile inoltre utilizzare un nome qualificato per fare riferimento alla funzione `myFunction()`, come nell'esempio seguente:

```
version1::myFunction();
```

Definizione degli spazi dei nomi

Gli spazi dei nomi contengono un solo valore, l'URI (Uniform Resource Identifier), che talvolta viene definito *nome dello spazio dei nomi*. Un URI consente di rendere univoca la definizione di uno spazio dei nomi.

Per creare uno spazio dei nomi, occorre dichiarare una definizione namespace. Quest'ultima può contenere un URI esplicito (come per gli spazi dei nomi XML) oppure esserne priva. L'esempio seguente mostra uno spazio dei nomi definito con un URI:

```
namespace flash_proxy = "http://www.adobe.com/flash/proxy";
```

L'URI funge da stringa di identificazione univoca per lo spazio dei nomi. Se viene omesso, come nell'esempio seguente, il compilatore crea al suo posto una stringa di identificazione univoca interna, alla quale non è possibile accedere.

```
namespace flash_proxy;
```

Una volta definito uno spazio dei nomi, con o senza URI, non è possibile ridefinirlo all'interno della stessa area di validità. Un eventuale tentativo in questo senso produce un errore del compilatore.

Se uno spazio dei nomi viene definito all'interno di un pacchetto o di una classe, potrebbe non essere visibile per il codice esterno a tale pacchetto o classe, a meno che non venga utilizzato lo specificatore di controllo accesso appropriato. Ad esempio, nel codice seguente lo spazio dei nomi `flash_proxy` è definito all'interno del pacchetto flash.utils. Nell'esempio, l'assenza di uno specificatore di controllo accesso fa sì che lo spazio dei nomi `flash_proxy` sia visibile solo al codice che si trova nel pacchetto flash.utils e invisibile a tutto il codice esterno a tale pacchetto:

```
package flash.utils
{
  namespace flash_proxy;
}
```

Il codice seguente utilizza l'attributo `public` per rendere lo spazio dei nomi `flash_proxy` visibile al codice esterno al pacchetto:

```
package flash.utils
{
  public namespace flash_proxy;
}
```

Applicazione degli spazi dei nomi

Applicare uno spazio dei nomi significa inserire una definizione all'interno di uno spazio dei nomi. Le definizioni che possono essere inserite negli spazi dei nomi sono le funzioni, le variabili e le costanti (non è possibile inserire una classe in uno spazio dei nomi personalizzato).

Si consideri, ad esempio, una funzione dichiarata mediante lo specificatore di controllo accesso `public`. L'uso dell'attributo `public` in una definizione di funzione determina l'inserimento della funzione nello spazio dei nomi pubblico e la conseguente disponibilità della funzione stessa per tutto il codice. Una volta definito uno spazio dei nomi personalizzato, è possibile utilizzarlo come l'attributo `public`, rendendo disponibile la definizione al codice che può fare riferimento allo spazio dei nomi. Ad esempio, se si definisce lo spazio dei nomi `example1`, è possibile aggiungere un metodo chiamato `myFunction()` usando `example1` come attributo, come nell'esempio seguente:

```
namespace example1;
class someClass
{
  example1 myFunction() {}
}
```

La dichiarazione del metodo `myFunction()` mediante l'attributo namespace `example1` indica che il metodo appartiene allo spazio dei nomi `example1`.

Quando si applica uno spazio dei nomi, tenere presente quanto segue:

- È possibile applicare un solo spazio dei nomi a ogni dichiarazione.
- Non è possibile applicare un attributo namespace a più di una definizione alla volta. In altre parole, se si vuole applicare uno spazio dei nomi a dieci diverse funzioni, è necessario aggiungere l'attributo namespace a ciascuna delle dieci definizioni di funzione.
- Se si applica uno spazio dei nomi, non è possibile specificare anche uno specificatore di controllo accesso, poiché i due elementi si escludono a vicenda. In altre parole, non è possibile dichiarare una funzione o una proprietà con l'attributo `public`, `private`, `protected` o `internal` dopo aver già applicato lo spazio dei nomi corrispondente.

Riferimenti agli spazi dei nomi

Non occorre fare riferimento in modo esplicito a uno spazio dei nomi quando si utilizza un metodo o una proprietà dichiarata con uno degli specificatori di controllo accesso, quali `public`, `private`, `protected` e `internal`, dal momento che l'accesso a questi spazi dei nomi è controllato dal contesto. Ad esempio, le definizioni inserite nello spazio dei nomi `private` sono automaticamente disponibili al codice che si trova all'interno della stessa classe. Nel caso degli spazi dei nomi personalizzati, tuttavia, questa "sensibilità al contesto" non esiste. Per utilizzare un metodo o una proprietà inserita in uno spazio dei nomi personalizzato, è indispensabile fare riferimento in modo esplicito allo spazio dei nomi.

È possibile fare riferimento agli spazi dei nomi mediante la direttiva `use namespace` oppure qualificando il nome con lo spazio dei nomi mediante la sintassi `::`. La direttiva `use namespace` permette di "aprire" lo spazio dei nomi, in modo che possa applicare eventuali identificatori non qualificati. Ad esempio, se è stato definito lo spazio dei nomi `example1`, è possibile accedere ai nomi al suo interno utilizzando la sintassi `use namespace example1`:

```
use namespace example1;
myFunction();
```

È possibile aprire più spazi dei nomi nello stesso momento. Quando si utilizza la direttiva `use namespace`, lo spazio dei nomi rimane aperto nell'intero blocco di codice in cui è stato aperto. Non esiste un modo per chiudere esplicitamente uno spazio dei nomi.

La presenza contemporanea di due o più spazi dei nomi aperti, tuttavia, aumenta la probabilità di conflitti tra nomi. Se si preferisce non aprire uno spazio dei nomi, è possibile evitare di usare la direttiva `use namespace` oppure qualificando il nome del metodo o della proprietà con lo spazio dei nomi e il segno ::. Ad esempio, nel codice seguente il nome `myFunction()` viene qualificato con lo spazio dei nomi `example1`:

```
example1::myFunction();
```

Uso degli spazi dei nomi

Nella classe flash.utils.Proxy dell'API di Flash Player è possibile trovare un esempio di uno spazio dei nomi reale che viene utilizzato per evitare conflitti tra nomi. La classe Proxy, che sostituisce la proprietà `Object.__resolve` di ActionScript 2.0, permette di intercettare i riferimenti a proprietà o metodi non definiti prima che si verifichi un errore. Tutti i metodi della classe Proxy si trovano all'interno dello spazio dei nomi `flash_proxy`, al fine di evitare conflitti tra i nomi.

Per comprendere meglio come viene utilizzato lo spazio dei nomi `flash_proxy`, è necessario capire come funziona la classe Proxy. La funzionalità di questa classe è disponibile unicamente alle classi che ereditano da essa. In altri termini, se si desidera utilizzare i metodi della classe Proxy su un oggetto, la definizione di classe dell'oggetto deve estendere la classe Proxy.
Ad esempio, se si vuole intercettare i tentativi di chiamare un metodo non definito, occorre estendere la classe Proxy ed eseguire l'override del metodo `callProperty()` della stessa classe.

Si è detto in precedenza che l'implementazione degli spazi dei nomi è solitamente un processo che prevede tre passaggi: definire uno spazio dei nomi, applicarlo e farvi riferimento. Tuttavia, poiché non viene mai fatto riferimento in modo esplicito a nessuno dei metodi della classe Proxy, lo spazio dei nomi `flash_proxy` viene solo definito e applicato, e non specificato in un riferimento. L'API di Flash Player definisce lo spazio dei nomi `flash_proxy` e lo applica nella classe Proxy. Il codice scritto dal programmatore deve soltanto applicare lo spazio dei nomi `flash_proxy` alle classi che estendono la classe Proxy.

Lo spazio dei nomi `flash_proxy` è definito nel pacchetto flash.utils in un modo simile al seguente:

```
package flash.utils
{
  public namespace flash_proxy;
}
```

Lo spazio dei nomi viene applicato ai metodi della classe Proxy come indicato nel seguente codice estratto da tale classe:

```
public class Proxy
{
  flash_proxy function callProperty(name:*, ... rest):*
  flash_proxy function deleteProperty(name:*):Boolean
  ...
}
```

Come illustra l'esempio di codice che segue, è necessario prima importare sia la classe Proxy che lo spazio dei nomi `flash_proxy`, quindi occorre dichiarare la classe in modo che estenda la classe Proxy (nonché aggiungere l'attributo `dynamic`, se il compilatore è in modalità rigorosa). Se si sostituisce il metodo `callProperty()`, si deve utilizzare lo spazio dei nomi `flash_proxy`.

```
package
{
  import flash.utils.Proxy;
  import flash.utils.flash_proxy;

  dynamic class MyProxy extends Proxy
  {
```

```
    flash_proxy override function callProperty(name:*, ...rest):*
    {
      trace("method call intercepted: " + name);
    }
  }
}
```

Se si crea un'istanza della classe MyProxy e si chiama un metodo non definito (ad esempio il metodo `testing()` chiamato nell'esempio che segue), l'oggetto Proxy intercetta la chiamata al metodo ed esegue le istruzioni contenute nel metodo `callProperty()` sostituito (in questo caso, una semplice istruzione `trace()`).

```
var mySample:MyProxy = new MyProxy();
mySample.testing(); // chiamata di metodo intercettata: testing
```

L'inclusione dei metodi della classe Proxy all'interno dello spazio dei nomi `flash_proxy` presenta due vantaggi. Innanzi tutto, la presenza di uno spazio dei nomi separato riduce la quantità di elementi nell'interfaccia pubblica di qualunque classe che estende la classe Proxy. (Vi sono circa una dozzina di metodi nella classe Proxy che possono essere sostituiti, nessuno dei quali è progettato per essere chiamato direttamente. Includerli tutti nello spazio dei nomi pubblico potrebbe creare confusione.) In secondo luogo, l'uso dello spazio dei nomi `flash_proxy` permette di evitare conflitti tra i nomi qualora la sottoclasse Proxy contenga nomi di metodi di istanze che corrispondono a nomi di metodi della classe Proxy. Ad esempio, si supponga che uno dei metodi sia denominato `callProperty()`. Il codice seguente è valido perché la versione personalizzata del metodo `callProperty()` si trova in uno spazio dei nomi diverso:

```
dynamic class MyProxy extends Proxy
{
  public function callProperty() {}
  flash_proxy override function callProperty(name:*, ...rest):*
  {
    trace("method call intercepted: " + name);
  }
}
```

Gli spazi dei nomi sono utili anche quando si desidera fornire accesso a metodi e proprietà in un modo che non sarebbe possibile con i quattro specificatori di controllo accesso (`public`, `private`, `internal` e `protected`). Ad esempio, se si hanno vari metodi di utilità sparsi in diversi pacchetti e si desidera renderli disponibili a tutti i pacchetti senza tuttavia renderli pubblici, è possibile creare un nuovo spazio dei nomi e utilizzarlo come specificatore di controllo accesso personalizzato.

L'esempio seguente utilizza uno spazio dei nomi definito dall'utente per raggruppare due funzioni che si trovano in pacchetti differenti. Grazie a questa operazione, è possibile rendere visibili entrambe le funzioni a una classe o a un pacchetto tramite una singola istruzione `use namespace`.

L'esempio che segue esempio usa quattro file per dimostrare questa tecnica. Tutti i file devono trovarsi all'interno del percorso di classe. Il primo, myInternal.as, viene utilizzato per definire lo spazio dei nomi `myInternal`. Poiché si trova in un pacchetto denominato example, è necessario inserirlo in una cartella con lo stesso nome. Lo spazio dei nomi viene contrassegnato come `public` in modo che possa essere importato in altri pacchetti.

```
// myInternal.as nella cartella example
package example
{
  public namespace myInternal = "http://www.adobe.com/2006/actionscript/
  examples";
}
```

Il secondo e il terzo file, Utility.as e Helper.as, definiscono le classi che contengono i metodi da rendere disponibili agli altri pacchetti. Poiché la classe Utility è nel pacchetto example.alpha, il file deve essere inserito in una cartella denominata alpha a sua volta contenuta nella cartella example. Allo stesso modo, la classe Helper si trova nel pacchetto example.alpha e il file corrispondente deve essere inserito nella sottocartella beta della cartella example. Entrambi i pacchetti, example.alpha e example.beta, devono importare lo spazio dei nomi per poterlo utilizzare.

```
// Utility.as nella cartella example/alpha
package example.alpha
{
  import example.myInternal;

  public class Utility
  {
    private static var _taskCounter:int = 0;

    public static function someTask()
    {
      _taskCounter++;
    }

    myInternal static function get taskCounter():int
    {
      return _taskCounter;
    }
  }
}
```

```
// Helper.as nella cartella example/beta
package example.beta
{
  import example.myInternal;

  public class Helper
  {
    private static var _timeStamp:Date;

    public static function someTask()
    {
      _timeStamp = new Date();
    }

    myInternal static function get lastCalled():Date
    {
      return _timeStamp;
    }
  }
}
```

Il quarto file, NamespaceUseCase.as, è la classe principale dell'applicazione e deve essere allo stesso livello della cartella example. In Adobe Flash CS3 Professional, questa classe verrebbe utilizzata come la classe documento per il file FLA. Anche la classe NamespaceUseCase importa lo spazio dei nomi `myInternal` e lo utilizza per chiamare i due metodi statici contenuti negli altri pacchetti. L'esempio utilizza i metodi statici solo per semplificare il codice. Nello spazio dei nomi `myInternal` è possibile inserire sia metodi statici che di istanza.

```
// NamespaceUseCase.as
package
{
  import flash.display.MovieClip;
  import example.myInternal;       // importa lo spazio dei nomi
  import example.alpha.Utility;    // importa la classe Utility
  import example.beta.Helper;      // importa la classe Helper

  public class NamespaceUseCase extends MovieClip
  {
    public function NamespaceUseCase()
    {
      use namespace myInternal;

      Utility.someTask();
      Utility.someTask();
      trace(Utility.taskCounter); // 2

      Helper.someTask();
      trace(Helper.lastCalled);   // [ultima chiamata di someTask()]
    }
  }
}
```

Variabili

Le variabili consentono di memorizzare valori da utilizzare in un programma. Per dichiarare una variabile, è necessario utilizzare l'istruzione `var` nel suo nome. In ActionScript 2.0, l'uso dell'istruzione `var` è previsto solo se si utilizzano le annotazioni di tipo, mentre è obbligatorio in tutti i casi in ActionScript 3.0. Ad esempio, la seguente riga di codice ActionScript dichiara una variabile denominata `i`:

```
var i;
```

Se si omette l'istruzione `var` quando si dichiara una variabile, viene generato un errore del compilatore in modalità rigorosa e un errore runtime in modalità standard. Ad esempio, la riga di codice seguente genera un errore se la variabile `i` non è stata definita in precedenza:

```
i; // errore se i non è stata definita in precedenza
```

L'associazione di una variabile a un tipo di dati deve essere effettuata al momento di dichiarare la variabile. La dichiarazione di una variabile senza indicazione del tipo è consentita ma genera un'avvertenza del compilatore in modalità rigorosa. Per designare il tipo di una variabile, si aggiunge al nome un carattere di due punti (:) seguito dal tipo. Ad esempio, il codice seguente dichiara una variabile denominata `i` del tipo int:

```
var i:int;
```

È possibile assegnare un valore a una variabile utilizzando l'operatore di assegnazione (=). Ad esempio, il codice seguente dichiara la variabile `i` e vi assegna il valore 20:

```
var i:int;
i = 20;
```

Può risultare più pratico assegnare un valore a una variabile nel momento in cui viene dichiarata, come nell'esempio seguente:

```
var i:int = 20;
```

Questa tecnica viene comunemente utilizzata non solo quando si assegnano valori di base come numeri interi e stringhe, ma anche quando si un crea un array o un'istanza di una classe. L'esempio seguente mostra un array che viene dichiarato e al quale viene assegnato un valore nella stessa riga di codice.

```
var numArray:Array = ["zero", "one", "two"];
```

Un'istanza di una classe può essere creata utilizzando l'operatore `new`. L'esempio seguente crea un'istanza della classe `CustomClass` e assegna alla variabile `customItem` un riferimento alla nuova istanza creata:

```
var customItem:CustomClass = new CustomClass();
```

È possibile dichiarare più variabili nella stessa riga di codice utilizzando l'operatore virgola (,) per separarle. Ad esempio, il codice seguente dichiara tre variabili nella stessa riga:

```
var a:int, b:int, c:int;
```

È anche possibile assegnare valori a ciascuna delle variabili nella stessa riga di codice. Ad esempio, il codice seguente dichiara tre variabili (`a`, `b` e `c`) e assegna un valore a ciascuna:

```
var a:int = 10, b:int = 20, c:int = 30;
```

L'uso dell'operatore virgola per raggruppare le dichiarazioni di variabili nella stessa istruzione può tuttavia ridurre la leggibilità del codice.

L'area di validità delle variabili

L'*area di validità* (scope) di una variabile è la parte di codice all'interno della quale è possibile accedere alla variabile mediante un riferimento lessicale. Una variabile *globale* è definita in tutte le aree del codice, mentre una variabile *locale* è definita in una sola parte. In ActionScript 3.0, alle variabili viene sempre assegnata l'area di validità della funzione o della classe nella quale sono dichiarate. Una variabile globale è una variabile definita all'esterno di qualunque funzione o definizione di classe. Ad esempio, il codice seguente crea una variabile globale `strGlobal` dichiarandola all'esterno di qualunque funzione. L'esempio mostra che una variabile globale è disponibile sia all'interno che all'esterno della definizione di funzione.

```
var strGlobal:String = "Global";
function scopeTest()
{
  trace(strGlobal); // Variabile globale
}
scopeTest();
trace(strGlobal); // Variabile globale
```

Una variabile locale viene dichiarata all'interno di una definizione di funzione. L'area di codice più piccola per la quale è possibile definire una variabile locale è una definizione di funzione. Una variabile locale dichiarata all'interno di una funzione esiste solo in tale funzione. Se, ad esempio, si dichiara una variabile denominata `str2` all'interno di una funzione denominata `localScope()`, tale variabile non è disponibile all'esterno della funzione.

```
function localScope()
{
  var strLocal:String = "local";
}
localScope();
trace(strLocal); // Errore perché strLocal non è definita a livello globale
```

Se la variabile locale è già stata dichiarata con lo stesso nome come variabile globale, la definizione locale ha la precedenza sulla definizione globale all'interno dell'area di validità della variabile locale. La variabile globale rimane valida all'esterno della funzione. Il codice seguente, ad esempio, crea una variabile globale di tipo String denominata `str1`, quindi crea una variabile locale con lo stesso nome all'interno della funzione `scopeTest()`. L'istruzione `trace` all'interno della funzione genera il valore locale della variabile, ma all'esterno della funzione genera il valore globale della variabile.

```
var str1:String = "Global";
function scopeTest ()
{
  var str1:String = "Local";
  trace(str1); // Variabile locale
}
scopeTest();
trace(str1); // Variabile globale
```

La variabili ActionScript, a differenza delle variabili C++ e Java, non possono avere un'area di validità a livello di blocco. Un blocco di codice è qualunque gruppo di istruzioni compreso tra una parentesi graffa di apertura (`{`) e una di chiusura (`}`). In alcuni linguaggi di programmazione, come C++ e Java, le variabili dichiarate in un blocco di codice non sono disponibili all'esterno del blocco. Questa restrizione viene definita "area di validità a livello di blocco" e non esiste in ActionScript. Una variabile dichiarata all'interno di un blocco di codice è disponibile non solo in tale blocco ma anche nelle altre parti della funzione alla quale esso appartiene. Ad esempio, la funzione seguente contiene variabili definite in vari blocchi di codice, ciascuna delle quali è disponibile nell'intera funzione.

```
function blockTest (testArray:Array)
{
  var numElements:int = testArray.length;
  if (numElements > 0)
  {
    var elemStr:String = "Element #";
    for (var i:int = 0; i < numElements; i++)
    {
      var valueStr:String = i + ": " + testArray[i];
      trace(elemStr + valueStr);
    }
    trace(elemStr, valueStr, i);   // Tutte ancora definite
  }
  trace(elemStr, valueStr, i); // Tutte definite se numElements > 0
}

blockTest(["Earth", "Moon", "Sun"]);
```

Un'implicazione interessante dell'assenza dell'area di validità a livello di blocco è la possibilità di leggere o scrivere una variabile prima che venga dichiarata, a condizione che la dichiarazione sia inclusa prima della fine della funzione. Ciò è possibile grazie a una tecnica chiamata *hoisting*, in base alla quale il compilatore sposta tutte le dichiarazioni di variabili all'inizio della funzione. Ad esempio, il codice seguente viene compilato correttamente anche se la funzione iniziale `trace()` della variabile `num` ha luogo prima che tale variabile sia dichiarata:

```
trace(num); // NaN
var num:Number = 10;
trace(num); // 10
```

Il compilatore tuttavia non esegue l'hoisting per le istruzioni di assegnazione, motivo per cui l'istruzione iniziale `trace()` di `num` produce `NaN` (not a number), che è il valore predefinito per le variabili del tipo di dati Number. Pertanto, è possibile assegnare valori alle variabili anche prima che siano dichiarate, come nell'esempio seguente:

```
num = 5;
trace(num); // 5
var num:Number = 10;
trace(num); // 10
```

Valori predefiniti

Per *valore predefinito* si intende il valore che una variabile contiene prima che ne venga impostato il valore. Una variabile viene *inizializzata* quando se ne imposta il valore per la prima volta. Se si dichiara una variabile ma non se ne imposta il valore, tale variabile rimane *non inizializzata*. Il valore di una variabile non inizializzata dipende dal suo tipo di dati. La tabella seguente descrive i valori predefiniti delle variabili, organizzati per tipo di dati:

Tipo di dati	Valore predefinito
Boolean	`false`
int	0
Number	`NaN`
Object	`null`
String	`null`
uint	0
Non dichiarato (equivalente all'annotazione di tipo `*`)	`undefined`
Tutte le altre classi, comprese quelle definite dall'utente.	`null`

Per le variabili di tipo Number, il valore predefinito è `NaN` (not a number), che è un valore speciale definito dallo standard IEEE-754 per indicare un valore che non rappresenta un numero.

Se si dichiara una variabile senza dichiararne il tipo di dati, viene applicato il tipo predefinito `*`, che indica che la variabile è priva di tipo. Inoltre, se una variabile senza tipo non viene inizializzata con un valore, il valore predefinito è `undefined`.

Per i tipi di dati diversi da Boolean, Number, int e uint, il valore predefinito di qualunque variabile non inizializzata è `null`, sia per le classi definite dall'API di Flash Player che per quelle personalizzate definite dall'utente.

Il valore `null` non è valido per le variabili di tipo Boolean, Number, int o uint. Se si tenta di assegnarlo a una di queste variabili, viene convertito nel valore predefinito per il tipo di dati corrispondente. È invece possibile assegnare il valore `null` alle variabili di tipo Object. Se si tenta di assegnare il valore `undefined` a una variabile Object, il valore viene convertito in `null`.

Per le variabili di tipo Number esiste una funzione speciale di primo livello denominata `isNaN()`, che restituisce il valore booleano `true` se la variabile non è un numero oppure `false` in caso contrario.

Tipi di dati

Un *tipo di dati* definisce un set di valori. Ad esempio, il tipo di dati Boolean comprende esattamente due valori: `true` e `false`. Oltre a Boolean, ActionScript 3.0 definisce vari altri tipi di dati di uso comune, come String, Number e Array. È inoltre possibile creare tipi di dati personalizzati utilizzando le classi o le interfacce per definire un set di valori specifici. In ActionScript 3.0 tutti i valori, di base o complessi, sono oggetti.

Un *valore di base* appartiene a uno dei seguenti tipi di dati: Boolean, int, Number, String e uint. I valori di base solitamente consentono di lavorare con maggiore rapidità rispetto a quando si utilizzano valori complessi, perché ActionScript memorizza i valori di base con un metodo speciale che permette di ottimizzare la memoria e la velocità di elaborazione.

NOTA

Per i lettori interessati ai dettagli tecnici: ActionScript memorizza i valori di base internamente come oggetti immutabili. Questo tipo di memorizzazione permette di passare un riferimento anziché un valore, ottenendo lo stesso effetto. In questo modo è possibile ridurre l'uso della memoria e aumentare la velocità, dal momento che i riferimenti hanno dimensioni notevolmente inferiori ai valori veri e propri.

Un *valore complesso* è qualunque valore diverso da un valore di base. I tipi di dati che definiscono set di valori complessi includono Array, Date, Error, Function RegExp, XML e XMLList.

Molti linguaggi di programmazione fanno distinzione tra i valori di base e i relativi oggetti wrapper. Java, ad esempio, prevede un valore di base int e la classe wrapper java.lang.Integer che lo contiene. I valori di base Java non sono oggetti, ma i relativi wrapper lo sono, il che rende i valori di base utili per determinate operazioni e gli oggetti wrapper più indicati per altre. In ActionScript 3.0, i valori di base e i relativi oggetti sono, ai fini pratici, indistinguibili. Tutti i valori, compresi quelli di base, sono oggetti. Flash Player tratta questi tipi di base come casi speciali che si comportano come oggetti ma non richiedono il lavoro di elaborazione solitamente associato alla creazione di oggetti. Ciò significa che le due righe di codice seguenti sono equivalenti:

```
var someInt:int = 3;
var someInt:int = new int(3);
```

Tutti i tipi di dati di base e complessi elencati sopra sono definiti dalle classi di base di ActionScript 3.0. Tali classi consentono di creare oggetti utilizzando valori letterali anziché l'operatore `new`. Ad esempio, è possibile creare un array specificando un valore letterale oppure la funzione di costruzione della classe Array, come nell'esempio seguente:

```
var someArray:Array = [1, 2, 3]; // Valore letterale
var someArray:Array = new Array(1,2,3); // Funzione di costruzione Array
```

Verifica del tipo

La verifica del tipo può avvenire sia in fase di compilazione che di runtime. I linguaggi che prevedono tipi statici, come C++ e Java, eseguono la verifica del tipo in fase di compilazione, mentre quelli che prevedono tipi dinamici, come Smalltalk e Python, gestiscono la verifica del tipo in fase di runtime. ActionScript 3.0 appartiene alla seconda categoria ed esegue la verifica del tipo in fase di runtime, ma supporta anche la verifica in fase di compilazione tramite una modalità speciale del compilatore definita *modalità rigorosa*. Nella modalità rigorosa la verifica del tipo avviene sia in fase di compilazione che di runtime, mentre in modalità standard viene eseguita solo in fase di runtime.

I linguaggi con assegnazione dinamica del tipo offrono una grande flessibilità in fase di strutturazione del codice, tuttavia comportano il rischio che gli errori di tipo si manifestino in fase di runtime. Al contrario, i linguaggi con tipi statici segnalano tutti gli errori di tipo in fase di compilazione ma richiedono che le informazioni sui tipi siano già note in tale fase.

Verifica del tipo in fase di compilazione

La verifica del tipo in fase di compilazione è spesso preferita nei progetti di grandi dimensioni perché, con il crescere delle dimensioni del progetto, la flessibilità dei tipi di dati perde generalmente importanza a favore della possibilità di rilevare tutti gli errori di tipo con la massima tempestività. È per questo motivo che, per impostazione predefinita, il compilatore ActionScript in Adobe Flash CS3 Professional e in Adobe Flex Builder 2 è configurato per funzionare in modalità rigorosa.

Per eseguire la verifica del tipo in fase di compilazione, il compilatore deve conoscere il tipo di dati delle variabili o delle espressioni contenute nel codice. Per dichiarare esplicitamente il tipo di dati di una variabile, aggiungere al nome della variabile l'operatore due punti (:) seguito dal tipo di dati come suffisso. Per associare un tipo di dati a un parametro, utilizzare l'operatore due punti seguito dal tipo di dati. Ad esempio, il codice seguente aggiunge il tipo di dati al parametro `xParam` e dichiara una variabile `myParam` con un tipo di dati esplicito:

```
function runtimeTest(xParam:String)
{
  trace(xParam);
}
var myParam:String = “hello”;
runtimeTest(myParam);
```

In modalità rigorosa, il compilatore ActionScript segnala i casi di mancata corrispondenza del tipo come errori di compilazione. Ad esempio, il codice seguente dichiara un parametro di funzione `xParam`, del tipo Object, ma successivamente tenta di assegnare valori del tipo String e Number allo stesso parametro. In modalità rigorosa, questo codice genera un errore del compilatore.

```
function dynamicTest(xParam:Object)
{
  if (xParam is String)
  {
    var myStr:String = xParam; // Errore del compilatore in modalità
                               // rigorosa
    trace("String: " + myStr);
  }
  else if (xParam is Number)
  {
    var myNum:Number = xParam; // Errore del compilatore in modalità
                               // rigorosa
    trace("Number: " + myNum);
  }
}
```

Anche in modalità rigorosa, tuttavia, è possibile scegliere di non eseguire la verifica del tipo in fase di compilazione non specificando il tipo sul lato destro di un'istruzione di assegnazione. Per contrassegnare una variabile o un'espressione come priva di tipo, è possibile omettere l'annotazione di tipo o utilizzare l'annotazione speciale dell'asterisco (*). Ad esempio, se il parametro `xParam` contenuto nel codice precedente viene modificato omettendo l'annotazione di tipo, il codice verrà compilato anche in modalità rigorosa:

```
function dynamicTest(xParam)
{
  if (xParam is String)
  {
    var myStr:String = xParam;
    trace("String: " + myStr);
  }
  else if (xParam is Number)
  {
    var myNum:Number = xParam;
    trace("Number: " + myNum);
  }
}
dynamicTest(100)
dynamicTest("one hundred");
```

Verifica del tipo in fase di runtime

Il controllo del tipo in runtime viene eseguito in ActionScript 3.0 sia nella modalità di compilazione rigorosa che in modalità standard. Si consideri una situazione in cui il valore 3 viene passato come argomento a una funzione per la quale è previsto array. In modalità rigorosa, il compilatore genera un errore perché il valore 3 non è compatibile con il tipo di dati Array. Se si disattiva la modalità rigorosa e si attiva la modalità standard, il compilatore non rileva la mancata corrispondenza del tipo, ma la verifica in fase di runtime eseguita da Flash Player genera un errore runtime.

L'esempio seguente mostra una funzione denominata `typeTest()` per la quale è previsto un argomento Array ma alla quale viene invece passato il valore 3. Questa situazione genera un errore runtime in modalità standard perché il valore 3 non è un membro del tipo di dati dichiarato del parametro (Array).

```
function typeTest(xParam:Array)
{
  trace(xParam);
}
var myNum:Number = 3;
typeTest(myNum);
// Errore runtime nella modalità standard di ActionScript 3.0
```

In determinati casi può essere generato un errore runtime di tipo anche in modalità rigorosa. Questa situazione può verificarsi quando, pur essendo attiva la modalità rigorosa, si sceglie di non eseguire la verifica del tipo in fase di compilazione specificando una variabile senza tipo. L'uso di una variabile senza tipo non elimina la verifica del tipo, bensì la rimanda alla fase di runtime. Ad esempio, se la variabile `myNum` di cui all'esempio precedente non ha un tipo di dati dichiarato, il compilatore non è in grado di rilevare la mancata corrispondenza del tipo ma Flash Player genera un errore runtime perché esegue un confronto tra il valore runtime di `myNum`, che è impostato su 3 dall'istruzione di assegnazione, e il tipo di dati di `xParam`, che è Array.

```
function typeTest(xParam:Array)
{
  trace(xParam);
}
var myNum = 3;
typeTest(myNum);
// Errore runtime in ActionScript 3.0
```

La verifica in fase di runtime consente anche un uso più flessibile dell'ereditarietà rispetto a quella in fase di compilazione. Rimandando la verifica del tipo alla fase di runtime, la modalità standard permette di fare riferimento alle proprietà di una sottoclasse anche se si esegue un *upcast*, ovvero si utilizza una classe di base per dichiarare il tipo di un'istanza di classe ma una sottoclasse per creare l'istanza vera e propria. Ad esempio, è possibile creare una classe denominata ClassBase che può essere estesa (non è possibile estendere le classi con attributo `final`):

```
class ClassBase
{
}
```

Successivamente è possibile creare una sottoclasse di ClassBase denominata ClassExtender, dotata di una proprietà di nome `someString`, come nel codice seguente:

```
class ClassExtender extends ClassBase
{
  var someString:String;
}
```

Utilizzando entrambe le classi, è possibile definire un'istanza di classe dichiarata mediante il tipo di dati ClassBase ma creata utilizzando la funzione di costruzione ClassExtender.
Un upcast è considerato un'operazione sicura perché la classe di base non contiene proprietà o metodi che non sono disponibili nella sottoclasse.

```
var myClass:ClassBase = new ClassExtender();
```

Una sottoclasse, al contrario, può contenere proprietà o metodi che non sono presenti nella rispettiva classe di base. Ad esempio, la classe ClassExtender contiene la proprietà `someString`, che non esiste nella classe ClassBase. Nella modalità standard di ActionScript 3.0, è possibile fare riferimento a questa proprietà usando l'istanza `myClass` senza generare un errore in fase di compilazione, come mostra l'esempio seguente:

```
var myClass:ClassBase = new ClassExtender();
myClass.someString = "hello";
// Nessun errore nella modalità standard di ActionScript 3.0
```

L'operatore is

L'operatore `is`, introdotto in ActionScript 3.0, consente di verificare se una variabile o un'espressione è un membro di un determinato tipo di dati. Nelle versioni precedenti di ActionScript, la stessa funzionalità era fornita dall'operatore `instanceof`, che tuttavia in ActionScript 3.0 non va utilizzato per verificare l'appartenenza a un tipo di dati. È necessario utilizzare l'operatore `is` anziché `instanceof` per eseguire la verifica manuale del tipo, perché l'espressione `x instanceof y` si limita a controllare se nella catena di prototipi di `x` è presente `y` (in ActionScript 3.0, la catena di prototipi non fornisce un quadro completo della gerarchia di ereditarietà).

L'operatore `is` invece esamina la gerarchia di ereditarietà effettiva e consente di verificare non solo se un oggetto è un'istanza di una particolare classe, ma anche se è un'istanza di una classe che implementa una particolare interfaccia. L'esempio seguente crea un'istanza della classe Sprite denominata `mySprite` e utilizza l'operatore `is` per verificare se `mySprite` è un'istanza delle classi Sprite e DisplayObject e se implementa l'interfaccia IEventDispatcher:

```
var mySprite:Sprite = new Sprite();
trace(mySprite is Sprite);                  // true
trace(mySprite is DisplayObject);           // true
trace(mySprite is IEventDispatcher);        // true
```

L'operatore `is` controlla la gerarchia di ereditarietà e segnala correttamente che `mySprite` è compatibile con le classi Sprite e DisplayObject (Sprite è una sottoclasse della classe DisplayObject). Inoltre, verifica inoltre se `mySprite` eredita da qualunque classe che implementa l'interfaccia IEventDispatcher. Poiché la classe Sprite eredita dalla classe EventDispatcher, che implementa l'interfaccia IEventDispatcher, l'operatore `is` segnala correttamente che `mySprite` implementa la stessa interfaccia.

Il codice dell'esempio seguente esegue gli stessi controlli di quello precedente, ma con l'operatore `instanceof` al posto di `is`. L'operatore `instanceof` identifica correttamente `mySprite` come istanza di Sprite o DisplayObject, ma restituisce `false` quando viene utilizzato per verificare se `mySprite` implementa l'interfaccia IEventDispatcher.

```
trace(mySprite instanceof Sprite);           // true
trace(mySprite instanceof DisplayObject);    // true
trace(mySprite instanceof IEventDispatcher); // false
```

L'operatore as

Anche l'operatore `as`, introdotto in ActionScript 3.0, consente di verificare se un'espressione è un membro di un determinato tipo di dati. A differenza di `is`, tuttavia, `as` non restituisce un valore booleano, bensì il valore dell'espressione (invece di `true`) oppure `null` (invece di `false`). L'esempio seguente mostra il risultato che si ottiene utilizzando l'operatore `as` anziché `is` per controllare semplicemente se un'istanza Sprite è membro dei tipi di dati DisplayObject, IEventDispatcher e Number.

```
var mySprite:Sprite = new Sprite();
trace(mySprite as Sprite); // [oggetto Sprite]
trace(mySprite as DisplayObject); // [oggetto Sprite]
trace(mySprite as IEventDispatcher); // [oggetto Sprite]
trace(mySprite as Number);   // null
```

Quando si usa l'operatore `as`, l'operando sulla destra deve essere un tipo di dati. Un eventuale tentativo di utilizzare un'espressione diversa da un tipo di dati come operando sulla destra produrrebbe un errore.

Classi dinamiche

Una classe *dinamica* definisce un oggetto che può essere alterato in fase di runtime aggiungendo o modificandone le proprietà e i metodi. Un classe non dinamica, ad esempio la classe String, si definisce *chiusa* (sealed in inglese). Non è possibile aggiungere proprietà o metodi a una classe chiusa in runtime.

Per creare una classe dinamica si utilizza l'attributo `dynamic` nella relativa dichiarazione. Ad esempio, il codice seguente crea una classe dinamica denominata `Protean`:

```
dynamic class Protean
{
  private var privateGreeting:String = "hi";
  public var publicGreeting:String = "hello";
  function Protean()
  {
    trace("Protean instance created");
  }
}
```

Se successivamente si crea un'istanza della classe `Protean`, è possibile aggiungervi proprietà o metodi esternamente alla definizione della classe. Ad esempio, il codice seguente crea un'istanza della classe `Protean` e vi aggiunge due proprietà denominate rispettivamente `aString` e `aNumber`:

```
var myProtean:Protean = new Protean();
myProtean.aString = "testing";
myProtean.aNumber = 3;
trace(myProtean.aString, myProtean.aNumber); // testing 3
```

Le proprietà aggiunte a un'istanza di una classe dinamica sono entità runtime, quindi le verifiche di qualsiasi tipo vengono eseguite in fase di runtime. Non è possibile aggiungere un'annotazione di tipo a una proprietà creata in questo modo.

È anche possibile aggiungere un metodo all'istanza `myProtean` definendo una funzione e associandola a una proprietà dell'istanza `myProtean`. Il codice seguente sposta l'istruzione trace in un metodo denominato `traceProtean()`:

```
var myProtean:Protean = new Protean();
myProtean.aString = "testing";
myProtean.aNumber = 3;
myProtean.traceProtean = function ()
{
  trace(this.aString, this.aNumber);
};
myProtean.traceProtean(); // testing 3
```

I metodi creati in questo modo, tuttavia, non hanno accesso a eventuali proprietà o metodi privati della classe Protean. Inoltre, anche i riferimenti a proprietà o metodi pubblici della classe `Protean` devono essere qualificati con la parola chiave `this` o con il nome della classe. Nell'esempio seguente, il metodo `traceProtean()` tenta di accedere alle variabili private e pubbliche della classe `Protean`.

```
myProtean.traceProtean = function ()
{
  trace(myProtean.privateGreeting); // undefined
  trace(myProtean.publicGreeting); // hello
};
myProtean.traceProtean();
```

Descrizione dei tipi di dati

I tipi di dati di base sono Boolean, int, Null, Number, String, uint e void. Le classi ActionScript di base definiscono inoltre i seguenti tipi di dati complessi: Object, Array, Date, Error, Function, RegExp, XML e XMLList.

Tipo di dati Boolean

Il tipo di dati Boolean comprende due valori: `true` e `false`. Nessun altro valore è consentito per le variabili di questo tipo. Il valore predefinito di una variabile Boolean dichiarata ma non inizializzata è `false`.

Tipo di dati int

Il tipo di dati int è memorizzato internamente come numero intero a 32 bit e comprende la serie di numeri interi da -2.147.483.648 (-2^{31}) a 2.147.483.647 ($2^{31} - 1$). Le versioni precedenti di ActionScript offrivano solo il tipo di dati Number, utilizzato sia per i numeri interi che per quelli a virgola mobile. In ActionScript 3.0 è ora possibile accedere ai tipi macchina di basso livello per i numeri interi a 32 bit con e senza segno. Se la variabile non deve usare numeri a virgola mobile, l'impiego del tipo di dati int al posto di Number dovrebbe garantire maggiore velocità ed efficienza.

Per i numeri interi al di fuori dell'intervallo dei valori minimo e massimo, utilizzare il tipo di dati Number, che è in grado di gestire i valori compresi tra -9.007.199.254.740.992 e 9.007.199.254.740.992 (valori interi a 53 bit). Il valore predefinito per le variabili del tipo di dati int è 0.

Tipo di dati Null

Il tipo di dati Null contiene un solo valore, `null`. Questo è il valore predefinito per il tipo di dati String e per tutte le classi che definiscono tipi di dati complessi, compresa la classe Object. Nessuno degli altri tipi di dati di base, come Boolean, Number, int e uint, contiene il valore `null`. Se si tenta di assegnare `null` a variabili del tipo Boolean, Number, int o uint, Flash Player converte `null` nel valore predefinito appropriato. Non è possibile utilizzare questo tipo di dati come annotazione di tipo.

Tipo di dati Number

In ActionScript 3.0, il tipo di dati Number può rappresentare numeri interi, numeri interi senza segno e numeri a virgola mobile. Tuttavia, per ottimizzare le prestazioni, è meglio utilizzare il tipo di dati Number solo per i numeri interi maggiori dei numeri a 32 bit memorizzabili dai tipi `int` e `uint`, oppure per i numeri a virgola mobile. Per memorizzare un numero a virgola mobile, includere il punto dei decimali nel numero; se si omette il punto dei decimali, il valore viene memorizzato come numero intero.

Il tipo di dati Number utilizza il formato a doppia precisione a 64 bit specificato dallo standard IEEE 754 per i calcoli aritmetici binari a virgola mobile, che prescrive come devono essere memorizzati i numeri a virgola mobile utilizzando i 64 bit disponibili. Un bit serve per designare il numero come positivo o negativo, undici bit sono utilizzati per l'esponente (memorizzato come base 2) e i rimanenti 52 bit servono per memorizzare il *significante* (chiamato anche *mantissa*), ovvero il numero che viene elevato alla potenza indicata dall'esponente.

Utilizzando alcuni dei bit per memorizzare un esponente, il tipo di dati Number può memorizzare numeri a virgola mobile notevolmente più grandi che non se utilizzasse tutti i bit per il significante. Ad esempio, se il tipo di dati Number utilizzasse tutti i 64 bit per memorizzare il significante, il numero più grande che potrebbe memorizzare sarebbe 2^{65} - 1. Utilizzando 11 bit per memorizzare un esponente, il tipo di dati Number può elevare il significante fino a una potenza di 2^{1023}.

I valori massimo e minimo che il tipo Number può rappresentare sono memorizzati in proprietà statiche della classe Number denominate `Number.MAX_VALUE` e `Number.MIN_VALUE`.

```
Number.MAX_VALUE == 1.79769313486231e+308
Number.MIN_VALUE == 4.940656458412467e-324
```

Un intervallo di numeri così ampio va a scapito della precisione. Il tipo di dati Number utilizza 52 bit per memorizzare il significante, con il risultato che i numeri che richiedono più di 52 bit per essere rappresentati con precisione, ad esempio la frazione 1/3, sono solo approssimazioni. Se l'applicazione richiede una precisione assoluta per i numeri decimali, è necessario utilizzare un software in grado di implementare i calcoli aritmetici a virgola mobile decimali anziché quelli binari.

Quando si memorizzano valori interi con il tipo di dati Number, vengono utilizzati solo i 52 bit del significante. Il tipo di dati Number usa questi 52 bit e un bit speciale nascosto per rappresentare i numeri interi da -9.007.199.254.740.992 (-2^{53}) a 9.007.199.254.740.992 (2^{53}).

Flash Player utilizza il valore `NaN` non solo come valore predefinito per le variabili di tipo `Number`, ma anche come risultato di qualunque operazione che dovrebbe restituire un numero e invece restituisce un valore diverso. Ad esempio, se si tenta di calcolare la radice quadrata di un numero negativo, il risultato è `NaN`. Altri valori Number speciali sono *infinito positivo* e *infinito negativo*.

NOTA	Il risultato della divisione per `0` è `NaN` solo se anche il divisore è `0`. La divisione per `0` dà il risultato `infinito` se il dividendo è positivo oppure `-infinito` se è negativo.

Tipo di dati String

Il tipo di dati String rappresenta una sequenza di caratteri a 16 bit. Le stringhe vengono memorizzate internamente come caratteri Unicode, utilizzando il formato UTF-16. Come nel linguaggio di programmazione Java, le stringhe sono valori immutabili. Un'operazione su un valore String restituisce una nuova istanza della stringa. Il valore predefinito di una variabile dichiarata con il tipo di dati String è `null`. Il valore `null` non è equivalente a una stringa vuota (`""`), anche se entrambi rappresentano l'assenza di caratteri.

Tipo di dati uint

Il tipo di dati uint è memorizzato internamente come numero intero a 32 bit senza segno e comprende la serie di numeri interi da 0 a 4.294.967.295 (2^{32} - 1). Utilizzare il tipo di dati uint in circostanze speciali che richiedono numeri interi non negativi. Ad esempio, va necessariamente utilizzato per rappresentare i valori di colore dei pixel, perché il tipo di dati int ha un bit di segno interno che non è appropriato per la gestione dei valori di colore. Per i numeri interi maggiori del valore uint massimo, utilizzare il tipo di dati Number, che è in grado di gestire i valori interi a 53 bit. Il valore predefinito per le variabili del tipo di dati uint è 0.

Tipo di dati void

Il tipo di dati void contiene un solo valore, `undefined`. Nelle versioni precedenti di ActionScript, `undefined` era il valore predefinito per le istanze della classe Object.
In ActionScript 3.0, il valore predefinito delle istanze Object è `null`. Se si tenta di assegnare il valore `undefined` a un'istanza della classe Object, Flash Player converte il valore in `null`.
È possibile assegnare il valore `undefined` solo alle variabili senza tipo, ovvero quelle che sono prive di un'annotazione di tipo oppure utilizzano l'asterisco (`*`) come annotazione. È possibile utilizzare `void` solo come annotazione del tipo restituito.

Tipo di dati Object

Il tipo di dati Object è definito dalla classe Object, che viene utilizzata come classe di base per tutte le definizioni di classe in ActionScript. La versione ActionScript 3.0 del tipo di dati Object differisce dalle versioni precedenti per tre aspetti. Innanzi tutto, il tipo di dati Object non è più il tipo predefinito assegnato alle variabili prive di annotazione di tipo. In secondo luogo, il tipo Object non include più il valore `undefined`, che era il valore predefinito di tutte le istanze Object. Infine, in ActionScript 3.0 il valore predefinito delle istanze della classe Object è `null`.

Nelle versioni precedenti di ActionScript, a una variabile priva di annotazione di tipo veniva assegnato automaticamente il tipo di dati Object. ActionScript 3.0, al contrario, prevede la possibilità di variabili effettivamente prive di tipo. Pertanto, le variabili per le quali non viene fornita un'annotazione di tipo vengono ora considerate senza tipo. Se si preferisce rendere esplicito a chi leggerà il codice che l'intenzione è effettivamente quella di lasciare una variabile priva di tipo, è possibile utilizzare il nuovo simbolo di asterisco (`*`) come annotazione di tipo, che equivale a omettere l'annotazione. L'esempio seguente mostra due istruzioni equivalenti, che dichiarano entrambe una variabile senza tipo `x`:

```
var x
var x:*
```

Solo le variabili senza tipo possono contenere il valore `undefined`. Se si tenta di assegnare il valore `undefined` a una variabile che appartiene a un tipo di dati, Flash Player converte il valore `undefined` nel valore predefinito del tipo di dati in questione. Per le istanze del tipo di dati Object, il valore predefinito è `null`. Pertanto, se si tenta di assegnare `undefined` a un'istanza Object, Flash Player converte il valore `undefined` in `null`.

Conversione del tipo di dati

Una conversione del tipo di dati ha luogo quando un valore viene trasformato in un valore appartenente a un tipo di dati diverso. Le conversioni di tipo possono essere *implicite* o *esplicite*. Una conversione implicita, chiamata anche *assegnazione forzata* (in inglese, coercion), viene talvolta eseguita da Flash Player in fase di runtime. Ad esempio, se il valore 2 è assegnato a una variabile del tipo di dati Boolean, Flash Player converte il valore 2 nel valore booleano `true` prima di assegnare il valore alla variabile. La conversione esplicita, chiamata anche *inserimento* (in inglese, casting), ha luogo quando il codice passa al compilatore l'istruzione di elaborare una variabile di un tipo di dati particolare come se appartenesse a un tipo di dati diverso. Quando l'operazione riguarda valori di base, l'inserimento ha effettivamente il risultato di convertire i valori da un tipo di dati a un altro. Per inserire un oggetto in un tipo diverso, occorre racchiudere il nome dell'oggetto tra parentesi e farlo precedere dal nome del nuovo tipo. Il codice seguente, ad esempio, accetta un valore booleano e lo inserisce in un numero intero:

```
var myBoolean:Boolean = true;
var myINT:int = int(myBoolean);
trace(myINT); // 1
```

Conversione implicita

La conversione implicita viene eseguita in fase di runtime in una serie di casi:

- Nelle istruzioni di assegnazione
- Quando i valori vengono passati come argomenti di funzioni
- Quando i valori vengono restituiti da funzioni
- Nelle espressioni che utilizzano determinati operatori, ad esempio l'operatore di addizione (+)

Per i tipi definiti dall'utente, la conversione implicita ha luogo quando il valore da convertire è un'istanza della classe di destinazione o di una classe derivata da essa. Se una conversione implicita ha esito negativo, viene generato un errore. Ad esempio, il codice seguente contiene una conversione implicita corretta e una con esito negativo:

```
class A {}
class B extends A {}

var objA:A = new A();
var objB:B = new B();
var arr:Array = new Array();

objA = objB; // La conversione viene eseguita.
objB = arr; // La conversione non viene eseguita.
```

Per i tipi di base, le conversioni implicite vengono gestite chiamando gli stessi algoritmi di conversione interni che vengono chiamati dalle funzioni di conversione esplicita.
Le conversioni dei tipi di base sono descritte in dettaglio nelle sezioni che seguono.

Conversione esplicita

È utile ricorrere alla conversione esplicita (detta anche inserimento o casting) quando si compila il codice in modalità rigorosa, in particolare quando si vuole evitare che una mancata corrispondenza di tipo generi un errore in fase di compilazione. Questa situazione può verificarsi quando si ha la certezza che i valori saranno convertiti correttamente in fase di runtime mediante l'assegnazione forzata. Ad esempio, quando si utilizzano i dati ricevuti da un form, si può scegliere di ricorrere all'assegnazione forzata per convertire determinati valori di stringa in valori numerici. Il codice seguente genera un errore di compilazione anche se, in modalità standard, verrebbe eseguito senza problemi.

```
var quantityField:String = "3";
var quantity:int = quantityField; // Errore di compilazione in modalità
                                  // rigorosa
```

Se si desidera continuare a utilizzare la modalità rigorosa ma si vuole convertire la stringa in numero intero, è possibile utilizzare la conversione esplicita, come nell'esempio seguente:

```
var quantityField:String = "3";
var quantity:int = int(quantityField); // La conversione esplicita viene
                                       // eseguita.
```

Inserimento nei tipi int, uint e Number

È possibile inserire qualunque tipo di dati nei tre seguenti tipi numerici: int, uint e Number. Se Flash Player non è in grado di convertire il numero per qualunque motivo, viene assegnato il valore predefinito 0 per i tipi di dati int e uint e il valore predefinito `NaN` per il tipo di dati Number. Se si converte un valore booleano in numero, `true` diventa 1 e `false` diventa 0.

```
var myBoolean:Boolean = true;
var myUINT:uint = uint(myBoolean);
var myINT:int = int(myBoolean);
var myNum:Number = Number(myBoolean);
trace(myUINT, myINT, myNum); // 1 1 1
myBoolean = false;
myUINT = uint(myBoolean);
myINT = int(myBoolean);
myNum = Number(myBoolean);
trace(myUINT, myINT, myNum); // 0 0 0
```

I valori stringa che contengono solo cifre possono essere convertiti correttamente in uno dei tre tipi numerici, i quali consentono inoltre di convertire stringhe che assomigliano a numeri negativi o che rappresentano un valore esadecimale (ad esempio, `0x1A`). Il processo di conversione ignora gli eventuali caratteri di spazio vuoto presenti all'inizio e alla fine del valore stringa. È anche possibile convertire le stringhe che hanno l'aspetto di numeri a virgola mobile utilizzando `Number()`. Se viene incluso il punto dei decimali, `uint()` e `int()` restituiscono un numero intero in cui i caratteri che seguono il punto sono troncati. Ad esempio, i valori stringa seguenti possono essere inseriti in valori numerici:

```
trace(uint("5"));     // 5
trace(uint("-5"));    // 4294967291. Riprende da MAX_VALUE
trace(uint(" 27 "));  // 27
trace(uint("3.7"));   // 3
trace(int("3.7"));    // 3
trace(int("0x1A"));   // 26
trace(Number("3.7")); // 3.7
```

I valori stringa che contengono caratteri non numerici restituiscono 0 se vengono convertiti con `int()` o `uint()` oppure `NaN` se convertiti con `Number()`. Il processo di conversione ignora lo spazio vuoto all'inizio e alla fine, ma restituisce 0 o `NaN` se una stringa contiene uno spazio vuoto tra due numeri.

```
trace(uint("5a"));    // 0
trace(uint("ten"));   // 0
trace(uint("17 63")); // 0
```

In ActionScript 3.0, la funzione `Number()` non supporta più gli ottali, ovvero i numeri a base 8. Se si passa una stringa con uno zero iniziale alla funzione `Number()` di ActionScript 2.0, il numero viene interpretato come ottale e convertito nell'equivalente decimale. Lo stesso non vale per la funzione `Number()` di ActionScript 3.0, che invece ignora lo zero iniziale.
Ad esempio, il codice seguente genera un output diverso se viene compilato con versioni differenti di ActionScript:

```
trace(Number("044"));
// ActionScript 3.0 44
// ActionScript 2.0 36
```

L'inserimento non è necessario quando un valore di un tipo numerico viene assegnato a una variabile di un altro tipo numerico. Anche in modalità rigorosa, i tipi numerici vengono convertiti implicitamente in altri tipi numerici. Ciò significa che in alcuni casi si possono ottenere valori imprevisti quando viene superato l'intervallo di un tipo numerico. Gli esempi seguenti vengono compilati correttamente in modalità rigorosa, ma alcuni di essi generano valori imprevisti:

```
var myUInt:uint = -3; // Assegna un valore di tipo int/Number alla
                      // variabile uint
trace(myUInt); // 4294967293

var myNum:Number = sampleUINT; // Assegna un valore di tipo int/uint
                               // alla variabile Number
trace(myNum) // 4294967293

var myInt:int = uint.MAX_VALUE + 1; // Assegna un valore di tipo Number
                                    // alla variabile uint
trace(myInt); // 0

myInt = int.MAX_VALUE + 1; // Assegna un valore di tipo uint/Number
                           // alla variabile int
trace(myInt); // -2147483648
```

La tabella che segue riepiloga i risultati dell'inserimento nei tipi Number, int o uint da altri tipi di dati.

Tipo di dati o valore	Risultato della conversione in Number, int o uint
Boolean	Se il valore è `true`, 1; altrimenti, 0.
Date	La rappresentazione interna dell'oggetto Date, che corrisponde al numero di millisecondi trascorsi dalla mezzanotte del 1 gennaio 1970 (ora universale).
`null`	0
Object	Se l'istanza è `null` e viene convertita in Number, `NaN`; altrimenti, 0.
String	Un numero, se Flash Player è in grado di convertire la stringa in numero; altrimenti, `NaN` se la stringa viene convertita nel tipo Number oppure 0 se viene convertita in int o uint.
`undefined`	Se convertito in Number, `NaN`; se convertito in int o uint, 0.

Inserimento nel tipo Boolean

L'inserimento di un tipo di dati numerico (uint, int o Number) nel tipo Boolean restituisce `false` se il valore numerico è 0 e `true` in tutti gli altri casi. Per il tipo di dati Number, anche il valore `NaN` restituisce `false`. L'esempio seguente illustra i risultati di un inserimento dei numeri -1, 0 e 1:

```
var myNum:Number;
for (myNum = -1; myNum<2; myNum++)
{
  trace("Boolean(" + myNum +") is " + Boolean(myNum));
}
```

L'output dell'esempio mostra che dei tre numeri solo 0 restituisce il valore `false`:

```
Boolean(-1) is true
Boolean(0) is false
Boolean(1) is true
```

L'inserimento di un valore String nel tipo Boolean restituisce `false` se la stringa è `null` o vuota (`""`) e `true` in tutti gli altri casi.

```
var str1:String;       // La stringa non inizializzata è null.
trace(Boolean(str1));  // false

var str2:String = "";  // Stringa vuota
trace(Boolean(str2));  // false

var str3:String = " "; // Solo spazio vuoto
trace(Boolean(str3));  // true
```

L'inserimento di un'istanza della classe Object nel tipo Boolean restituisce `false` se l'istanza è `null` e `true` in tutti gli altri casi:

```
var myObj:Object;      // L'oggetto non inizializzato è null.
trace(Boolean(myObj)); // false

myObj = new Object();  // Crea un'istanza
trace(Boolean(myObj)); // true
```

Le variabili Boolean vengono elaborate in un modo speciale in modalità rigorosa, ovvero è possibile assegnare valori di qualunque tipo di dati a una variabile Boolean senza eseguire l'inserimento. L'assegnazione forzata implicita da tutti i tipi di dati al tipo Boolean viene eseguita anche in modalità rigorosa. In altri termini, contrariamente a quasi tutti gli altri tipi di dati, l'inserimento nel tipo Boolean non è necessario per evitare gli errori della modalità rigorosa. Gli esempi seguenti vengono compilati correttamente in modalità rigorosa e danno i risultati previsti anche in fase di runtime:

```
var myObj:Object = new Object();  // Crea un'istanza
var bool:Boolean = myObj;
trace(bool); // true
bool = "random string";
trace(bool); // true
bool = new Array();
trace(bool); // true
bool = NaN;
trace(bool); // false
```

La tabella che segue riepiloga i risultati dell'inserimento nel tipo Boolean da altri tipi di dati.

Tipo di dati o valore	Risultato della conversione in Boolean
String	`false` se il valore è `null` o una stringa vuota (`""`); `true` negli altri casi.
`null`	`false`
Number, int o uint	`false` se il valore è `NaN` o 0; `true` negli altri casi.
Object	`false` se l'istanza è `null`; `true` negli altri casi.

Inserimento nel tipo String

L'inserimento di un tipo di dati numerico nel tipo di dati String restituisce una rappresentazione del numero sotto forma di stringa. Se invece si inserisce un valore booleano nel tipo di dati String, viene restituita la stringa `"true"` se il valore è `true` e la stringa `"false"` se il valore è `false`.

Se si inserisce un'istanza della classe Object nel tipo di dati String, viene restituita la stringa `"null"` se l'istanza è `null`. In tutti gli altri casi, l'inserimento della classe Object nel tipo di dati String restituisce la stringa "`[object Object]`".

Quando si inserire un'istanza della classe Array nel tipo String, viene restituita una stringa composta da un elenco delimitato da virgole di tutti gli elementi dell'array. Ad esempio, il seguente inserimento nel tipo di dati String restituisce una sola stringa contenente tutti e tre gli elementi dell'array:

```
var myArray:Array = ["primary", "secondary", "tertiary"];
trace(String(myArray)); // primary,secondary,tertiary
```

Se si inserisce un'istanza della classe Date nel tipo di dati String, viene restituita una rappresentazione della data contenuta nell'istanza sotto forma di stringa. Ad esempio, il codice seguente restituisce una rappresentazione sotto forma di stringa dell'istanza della classe Date. L'output mostra i risultati per l'ora legale del Pacifico:

```
var myDate:Date = new Date(2005,6,1);
trace(String(myDate)); // Venerdì 1 luglio 00.00.00 GMT-0700 2005
```

La tabella che segue riepiloga i risultati dell'inserimento nel tipo String da altri tipi di dati.

Tipo di dati o valore	Risultato della conversione in String
Array	Una stringa con tutti gli elementi dell'array.
Boolean	`"true"` o `"false"`
Date	La rappresentazione in formato stringa dell'oggetto Date.
`null`	`"null"`
Number, int o uint	La rappresentazione in formato stringa del numero.
Object	Se l'istanza è null, `"null"`; negli altri casi, `"[object Object]"`.

Sintassi

La sintassi di un linguaggio definisce una serie di regole che devono essere rispettate quando si scrive codice eseguibile.

Distinzione tra maiuscole e minuscole

Nel linguaggio ActionScript 3.0 viene fatta distinzione tra maiuscole e minuscole. Gli identificatori che presentano gli stessi caratteri ma lettere minuscole e maiuscole differenti vengono considerati identificatori diversi. Ad esempio, il codice seguente crea due variabili diverse:

```
var num1:int;
var Num1:int;
```

Sintassi del punto

L'operatore punto (.) permette di accedere alle proprietà e ai metodi di un oggetto. Utilizzando la sintassi del punto, è possibile fare riferimento a una proprietà o a un metodo di una classe specificando un nome di istanza seguito dall'operatore punto e dal nome della proprietà o del metodo. Ad esempio, si consideri la seguente definizione di classe:

```
class DotExample
{
  public var prop1:String;
  public function method1():void {}
}
```

Mediante la sintassi del punto, è possibile accedere alla proprietà `prop1` e al metodo `method1()` utilizzando il nome di istanza creato nel codice seguente:

```
var myDotEx:DotExample = new DotExample();
myDotEx.prop1 = "hello";
myDotEx.method1();
```

Si può ricorrere alla sintassi del punto anche per la definizione dei pacchetti. L'operatore punto viene utilizzato per fare riferimento ai pacchetti nidificati. Ad esempio, la classe EventDispatcher si trova in un pacchetto denominato events che è nidificato all'interno del pacchetto flash. Per fare riferimento al pacchetto events, usare l'espressione seguente:

```
flash.events
```

È anche possibile fare riferimento alla classe EventDispatcher mediante l'espressione seguente:

```
flash.events.EventDispatcher
```

Sintassi della barra

La sintassi della barra non è supportata in ActionScript 3.0. Nelle versioni precedenti di ActionScript la barra inclinata indicava il percorso di un clip filmato o di una variabile.

Valori letterali

Per *valore letterale* si intende un valore che compare direttamente nel codice. I seguenti esempi sono valori letterali:

```
17
"hello"
-3
9.4
null
undefined
true
false
```

I valori letterali possono essere raggruppati a formare valori letterali composti. I valori letterali di array sono racchiusi tra parentesi quadre ([]) e utilizzano la virgola per separare gli elementi dell'array.

Un valore letterale array può essere utilizzato per inizializzare un array. Gli esempi seguenti mostrano due array inizializzati tramite valori letterali array. È possibile utilizzare l'istruzione new e passare il valore letterale composto come parametro alla funzione di costruzione della classe Array, ma è anche possibile assegnare valori letterali direttamente durante la creazione di istanze delle seguenti classi principali ActionScript: Object, Array, String, Number, int, uint, XML, XMLList e Boolean.

```
// Viene utilizzata l'istruzione new
var myStrings:Array = new Array(["alpha", "beta", "gamma"]);
var myNums:Array = new Array([1,2,3,5,8]);

// Il valore letterale viene assegnato direttamente
var myStrings:Array = ["alpha", "beta", "gamma"];
var myNums:Array = [1,2,3,5,8];
```

I valori letterali possono essere anche utilizzati per inizializzare un oggetto generico, ovvero un'istanza della classe Object. I valori letterali oggetto sono racchiusi tra parentesi graffe ({}) e utilizzano la virgola per separare le proprietà dell'oggetto. Ogni proprietà viene dichiarata con il carattere di due punti (:) che separa il nome della proprietà dal relativo valore.

È possibile creare un oggetto generico utilizzando l'istruzione new e passando il valore letterale dell'oggetto come parametro alla funzione di costruzione della classe Object, oppure assegnare il valore letterale dell'oggetto direttamente all'istanza che si dichiara. L'esempio seguente crea un nuovo oggetto generico e lo inizializza con tre proprietà (propA, propB e propC), con valori impostati rispettivamente su 1, 2 e 3:

```
// Viene utilizzata l'istruzione new
var myObject:Object = new Object({propA:1, propB:2, propC:3});

// Il valore letterale viene assegnato direttamente
var myObject:Object = {propA:1, propB:2, propC:3};
```

Per ulteriori informazioni, vedere "Elementi fondamentali delle stringhe" a pagina 217, "Nozioni di base delle espressioni regolari" a pagina 306 e "Inizializzazione delle variabili di XML" a pagina 378.

Punto e virgola

Il carattere di punto e virgola (`;`) può essere utilizzato per terminare un'istruzione. Se si omette il carattere del punto e virgola, il compilatore presuppone che ogni riga di codice rappresenti una singola istruzione. Poiché molti programmatori sono abituati a usare il punto e virgola per indicare la fine di un'istruzione, il codice può risultare più leggibile se si adotta questa convezione in modo omogeneo.

L'uso di un punto e virgola per terminare un'istruzione consente di includere più di un'istruzione su una sola riga; tuttavia, un "affollamento" eccessivo di istruzioni può rendere più difficoltosa la lettura del codice.

Parentesi

In ActionScript 3.0 le parentesi (`()`) hanno tre usi possibili. Innanzi tutto, possono essere utilizzate per cambiare l'ordine delle operazioni all'interno di un'espressione. Le operazioni raggruppate all'interno di parentesi vengono sempre eseguite per prime. Ad esempio, nel codice seguente le parentesi sono state utilizzate per cambiare l'ordine delle operazioni:

```
trace(2 + 3 * 4);    // 14
trace( (2 + 3) * 4); // 20
```

In secondo luogo, è possibile utilizzare le parentesi con l'operatore virgola (`,`) per valutare una serie di espressioni e restituire il risultato dell'espressione finale, come nell'esempio seguente:

```
var a:int = 2;
var b:int = 3;
trace((a++, b++, a+b)); // 7
```

Infine, è possibile utilizzare le parentesi per passare uno o più parametri a funzioni o metodi, come nell'esempio seguente, dove viene passato un valore String alla funzione `trace()`:

```
trace("hello"); // hello
```

Commenti

Il codice ActionScript 3.0 supporta due tipi di commenti: a riga singola e su più righe. Questi meccanismi per l'inserimento di commenti sono simili a quelli di C++ e Java. Tutto il testo contrassegnato come commento viene ignorato dal compilatore.

I commenti a riga singola iniziano con due caratteri di barra (`//`) e continuano fino alla fine della riga. Ad esempio, il codice seguente contiene un commento a riga singola:

```
var someNumber:Number = 3; // Commento a riga singola
```

I commenti su più righe invece iniziano con una barra seguita da un asterisco (/*) e finiscono con un asterisco seguito da una barra (*/).

```
/* Questo è un commento multiriga che può occupare
più di una riga di codice. */
```

Parole chiave e riservate

Per *parole riservate* si intendono parole che non possono essere utilizzate come identificatori nel codice perché sono riservate per l'uso in ActionScript. Le parole riservate comprendono anche le *parole chiave lessicali*, che vengono rimosse dallo spazio dei nomi del programma dal compilatore. Se si utilizza una parola chiave lessicale come identificatore, il compilatore genera un errore. Nella tabella seguente sono elencate tutte le parole chiave lessicali di ActionScript 3.0.

as	break	case	catch
class	const	continue	default
delete	do	else	extends
false	finally	for	function
if	implements	import	in
instanceof	interface	internal	is
native	new	null	package
private	protected	public	return
super	switch	this	throw
to	true	try	typeof
use	var	void	while
with			

Esiste inoltre un piccolo gruppo di parole chiave, chiamate *parole chiave sintattiche*, che possono essere utilizzate come identificatori ma hanno un significato speciale in determinati contesti. Nella tabella seguente sono elencate tutte le parole chiave sintattiche di ActionScript 3.0.

each	get	set	namespace
include	dynamic	final	native
override	static		

Infine, vi sono vari identificatori che talvolta vengono definiti *parole riservate future*. Si tratta di parole che non sono riservate in ActionScript 3.0, ma potrebbero essere considerate come parole chiave da software che incorporano ActionScript 3.0. Benché sia possibile che molti di questi identificatori possano essere utilizzati senza problemi nella maggior parte dei casi, Adobe consiglia di evitarne l'uso perché potrebbero diventare parole chiave in una versione successiva del linguaggio.

abstract	boolean	byte	cast
char	debugger	double	enum
export	float	goto	intrinsic
long	prototype	short	synchronized
throws	to	transient	type
virtual	volatile		

Costanti

ActionScript 3.0 supporta l'istruzione `const`, che permette di creare una costante. Le costanti sono proprietà il cui valore è fisso, ovvero non modificabile. È possibile assegnare un valore a una costante una sola volta, e l'assegnazione deve avvenire in prossimità della dichiarazione della costante. Ad esempio, se una costante viene dichiarata come membro di una classe, è possibile assegnare ad essa un valore solo nella dichiarazione stessa oppure nella funzione di costruzione della classe.

Il codice seguente dichiara due costanti. Alla prima, `MINIMUM`, viene assegnato un valore nell'istruzione della dichiarazione, mentre il valore della seconda costante, `MAXIMUM`, viene assegnato nella funzione di costruzione.

```
class A
{
  public const MINIMUM:int = 0;
  public const MAXIMUM:int;

  public function A()
  {
    MAXIMUM = 10;
  }
}

var a:A = new A();
trace(a.MINIMUM); // 0
trace(a.MAXIMUM); // 10
```

Se si tenta di assegnare un valore iniziale a una costante in qualunque altro modo, viene generato un errore. Ad esempio, se si imposta il valore iniziale di `MAXIMUM` all'esterno della classe, si verifica un errore runtime.

```
class A
{
  public const MINIMUM:int = 0;
  public const MAXIMUM:int;
}

var a:A = new A();
a["MAXIMUM"] = 10; // Errore runtime
```

L'API di Flash Player definisce un'ampia gamma di costanti utilizzabili dal programmatore. Per convenzione, le costanti in ActionScript contengono solo lettere maiuscole, con le parole separate dal carattere di sottolineatura (_). Ad esempio, la definizione della classe MouseEvent utilizza questa convenzione di denominazione per le proprie costanti, ciascuna delle quali rappresenta un evento relativo all'input del mouse:

```
package flash.events
{
  public class MouseEvent extends Event
  {
    public static const CLICK:String               = "click";
    public static const DOUBLE_CLICK:String        = "doubleClick";
    public static const MOUSE_DOWN:String          = "mouseDown";
    public static const MOUSE_MOVE:String          = "mouseMove";
    ...
  }
}
```

Operatori

Gli operatori sono funzioni speciali che accettano uno o più operandi e restituiscono un valore. Un *operando* è un valore (solitamente un valore letterale, una variabile o un'espressione) che svolge la funzione di input per un operatore. Ad esempio, nel seguente codice gli operatori di addizione (+) e moltiplicazione (*) sono utilizzati con tre operandi letterali (2, 3 e 4) per restituire un valore, il quale viene quindi utilizzato dall'operatore di assegnazione (=) per assegnare il valore restituito, 14, alla variabile `sumNumber`.

```
var sumNumber:uint = 2 + 3 * 4; // uint = 14
```

Gli operatori possono essere unari, binari o ternari. Un operatore *unario* accetta un solo operando. Ad esempio, l'operatore di incremento (++) è di tipo unario perché accetta un unico operando. Un operatore *binario* accetta due operandi. È il caso, ad esempio, dell'operatore di divisione (/), che accetta due operandi. Un operatore *ternario* accetta tre operandi. Ad esempio, l'operatore condizionale (?:) accetta tre operandi.

Alcuni operatori, definiti *overloaded*, si comportano in modo diverso a seconda del tipo o del numero di operandi che vengono specificati. L'operatore di addizione (+) è un esempio di operatore overloaded che presenta un comportamento diverso a seconda del tipo di dati degli operandi. Se entrambi gli operandi sono numeri, l'operatore di addizione restituisce la somma dei valori. Se invece entrambi gli operandi sono stringhe, l'operatore restituisce la concatenazione dei due operandi. L'esempio di codice seguente mostra come l'operatore si comporta in modo diverso a seconda degli operandi specificati.

```
trace(5 + 5);     // 10
trace("5" + "5"); // 55
```

Gli operatori possono presentare un comportamento diverso anche in base al numero di operandi specificati. L'operatore di sottrazione (-), ad esempio, è sia di tipo unario che binario. Se viene fornito un unico operando, l'operatore di sottrazione ne esegue la negazione e restituisce il risultato. Se gli operandi sono due, l'operatore ne restituisce la differenza. L'esempio seguente mostra l'operatore di sottrazione utilizzato prima nella versione unaria e poi in quella binaria.

```
trace(-3);  // -3
trace(7-2); // 5
```

Priorità e associatività degli operatori

La priorità e l'associatività degli operatori determina l'ordine in cui questi vengono elaborati. Se si ha familiarità con le operazioni matematiche può sembrare ovvio che il compilatore elabori l'operatore di moltiplicazione (*) prima di quello di addizione (+). Tuttavia, il compilatore necessita di istruzioni esplicite sull'ordine di elaborazione degli operatori. Queste istruzioni vengono dette globalmente *priorità degli operatori*. ActionScript definisce un ordine di priorità predefinito per gli operatori che è possibile modificare tramite l'operatore parentesi tonda (()). Ad esempio, il codice seguente modifica l'ordine di priorità predefinito dell'esempio precedente per forzare il compilatore a elaborare l'operatore di addizione prima di quello di moltiplicazione:

```
var sumNumber:uint = (2 + 3) * 4; // uint == 20
```

In alcune situazioni, nella stessa espressione possono essere presenti due o più operatori con la stessa priorità. In questi casi, il compilatore utilizza le regole di *associatività* per determinare l'operatore da elaborare per primo. Tutti gli operatori binari, ad eccezione degli operatori di assegnazione, hanno un'associatività *da sinistra a destra*, ovvero gli operatori a sinistra vengono elaborati prima di quelli a destra. Gli operatori binari di assegnazione e l'operatore condizionale (`?:`) hanno un'associatività *da destra a sinistra*, ovvero gli operatori a destra vengono elaborati prima di quelli a sinistra.

Considerare, ad esempio, gli operatori minore di (`<`) e maggiore di (`>`) che hanno la stessa priorità. Se entrambi gli operatori vengono utilizzati nella stessa espressione, l'operatore a sinistra viene elaborato per primo, perché l'associatività di entrambi è da sinistra a destra. Le due istruzioni seguenti producono pertanto lo stesso risultato:

```
trace(3 > 2 < 1);   // false
trace((3 > 2) < 1); // false
```

L'operatore maggiore di viene elaborato per primo e restituisce `true`, perché l'operando 3 è maggiore dell'operando 2. Il valore `true` viene quindi passato all'operatore minore di insieme all'operando 1. Il codice seguente illustra questo stato intermedio:

```
trace((true) < 1);
```

L'operatore minore di converte il valore `true` nel valore numerico 1 e confronta quest'ultimo con il secondo operando 1, restituendo il valore `false` (perché il valore di 1 non è minore di 1).

```
trace(1 < 1); // false
```

L'associatività a sinistra predefinita può essere modificata mediante l'operatore parentesi. È possibile istruire il compilatore a elaborare per primo l'operatore minore di racchiudendolo tra parentesi insieme ai relativi operandi. L'esempio seguente utilizza l'operatore parentesi per produrre un output diverso con gli stessi numeri dell'esempio precedente:

```
trace(3 > (2 < 1)); // true
```

L'operatore minore di viene elaborato per primo e restituisce il valore `false` perché l'operando 2 non è minore dell'operando 1. Il valore `false` viene quindi passato all'operatore maggiore di insieme all'operando 3. Il codice seguente illustra questo stato intermedio:

```
trace(3 > (false));
```

L'operatore maggiore di converte il valore `false` nel valore numerico 0 e confronta quest'ultimo con l'altro operando 3, restituendo il valore `true` (perché il valore di 3 è maggiore di 0).

```
trace(3 > 0); // true
```

La tabella seguente elenca gli operatori di ActionScript 3.0 in ordine di priorità decrescente. Ogni riga della tabella contiene operatori che hanno la stessa priorità. Ciascuna riga di operatori ha una priorità più alta della riga sottostante.

Gruppo	Operatori
Primari	`[] {x:y} () f(x) new x.y x[y] <></> @ :: ..`
Forma suffissa	`x++ x--`
Unari	`++x --x + - ~ ! delete typeof void`
Moltiplicativi	`* / %`
Additivi	`+ -`
Spostamento bit a bit	`<< >> >>>`
Relazionali	`< > <= >= as in instanceof is`
Uguaglianza	`== != === !==`
AND bit a bit	`&`
XOR bit a bit	`^`
OR bit a bit	`|`
AND logico	`&&`
OR logico	`||`
Condizionali	`?:`
Assegnazione	`= *= /= %= += -= <<= >>= >>>= &= ^= |=`
Virgola	`,`

Operatori primari

Gli operatori primari consentono di creare valori letterali Array e Object, raggruppare espressioni, chiamare funzioni, creare istanze di classe e accedere alle proprietà.

Tutti gli operatori primari, elencati nella tabella seguente, hanno priorità equivalente. Gli operatori inclusi nella specifica E4X sono contrassegnati con l'indicazione (E4X).

Operatore	Operazione eseguita
`[]`	Inizializza un array
`{x:y}`	Inizializza un oggetto
`()`	Raggruppa espressioni
`f(x)`	Chiama una funzione

Operatore	Operazione eseguita
`new`	Chiama una funzione di costruzione
`x.y x[y]`	Accede a una proprietà
`<></>`	Inizializza un oggetto XMLList (E4X)
`@`	Accede a un attributo (E4X)
`::`	Qualifica un nome (E4X)
`..`	Accede a un elemento XML discendente (E4X)

Operatori in forma suffissa

Gli operatori in forma suffissa accettano un operatore e ne incrementano o decrementano il valore. Sebbene si tratti di operatori unari, sono classificati separatamente dal resto degli operatori di questo tipo perché hanno una priorità maggiore e si comportano in modo particolare. Quando si utilizza un operatore in forma suffissa all'interno di un'espressione complessa, il valore dell'espressione viene restituito prima dell'elaborazione dell'operatore in forma suffissa. Nel codice seguente, ad esempio, il valore dell'espressione `xNum++` viene restituito prima che il valore venga incrementato:

```
var xNum:Number = 0;
trace(xNum++); // 0
trace(xNum);   // 1
```

Tutti gli operatori in forma suffissa, elencati nella tabella seguente, hanno priorità equivalente.

Operatore	Operazione eseguita
`++`	Incremento (in forma suffissa)
`--`	Decremento (in forma suffissa)

Operatori unari

Gli operatori unari accettano un unico operando. Gli operatori di incremento (`++`) e decremento (`--`) di questo gruppo sono operatori *in forma prefissa*, ovvero vengono utilizzati prima dell'operando in un'espressione. Gli operatori in forma prefissa sono diversi da quelli in forma suffissa perché l'operazione di incremento o decremento viene completata prima della restituzione del valore di tutta l'espressione. Nel codice seguente, ad esempio, il valore dell'espressione `++xNum` viene restituito dopo l'incremento del valore.

```
var xNum:Number = 0;
trace(++xNum); // 1
trace(xNum);   // 1
```

Tutti gli operatori unari, elencati nella tabella seguente, hanno priorità equivalente.

Operatore	Operazione eseguita
`++`	Incremento (in forma prefissa)
`--`	Decremento (in forma prefissa)
`+`	+ unario
`-`	- unario (negazione)
`!`	NOT logico
`~`	NOT bit a bit
`delete`	Elimina una proprietà
`typeof`	Restituisce informazioni sul tipo
`void`	Restituisce un valore indefinito

Operatori moltiplicativi

Gli operatori moltiplicativi accettano due operandi ed eseguono moltiplicazioni, divisioni e moduli.

Tutti gli operatori moltiplicativi, elencati nella tabella seguente, hanno priorità equivalente.

Operatore	Operazione eseguita
`*`	Moltiplicazione
`/`	Divisione
`%`	Modulo

Operatori additivi

Gli operatori additivi accettano due operandi ed eseguono addizioni o sottrazioni. Tutti gli operatori additivi, elencati nella tabella seguente, hanno priorità equivalente.

Operatore	Operazione eseguita
`+`	Addizione
`-`	Sottrazione

Operatori di spostamento bit a bit

Gli operatori di spostamento bit a bit accettano due operandi e spostano i bit del primo operando in base a quanto specificato dal secondo operando. Tutti gli operatori di spostamento bit a bit, elencati nella tabella seguente, hanno priorità equivalente.

Operatore	Operazione eseguita
`<<`	Spostamento a sinistra bit a bit
`>>`	Spostamento a destra bit a bit
`>>>`	Spostamento a destra bit a bit senza segno

Operatori relazionali

Gli operatori relazionali accettano due operandi, ne confrontano il valore e restituiscono un valore booleano. Tutti gli operatori relazionali, elencati nella tabella seguente, hanno priorità equivalente.

Operatore	Operazione eseguita
`<`	Minore di
`>`	Maggiore di
`<=`	Minore o uguale a
`>=`	Maggiore o uguale a
`as`	Verifica il tipo di dati
`in`	Verifica le proprietà dell'oggetto
`instanceof`	Controlla la catena di prototipi
`is`	Verifica il tipo di dati

Operatori di uguaglianza

Gli operatori di uguaglianza accettano due operandi, ne confrontano il valore e restituiscono un valore booleano. Tutti gli operatori di uguaglianza, elencati nella tabella seguente, hanno priorità equivalente.

Operatore	Operazione eseguita
`==`	Uguaglianza
`!=`	Diseguaglianza
`===`	Uguaglianza rigorosa
`!==`	Diseguaglianza rigorosa

Operatori logici bit a bit

Gli operatori logici bit a bit accettano due operandi ed eseguono operazioni logiche a livello di bit. Gli operatori logici bit a bit hanno priorità diversa. Nella tabella seguente, gli operatori sono elencati in ordine di priorità, da maggiore a minore:

Operatore	Operazione eseguita
`&`	AND bit a bit
`^`	XOR bit a bit
`|`	OR bit a bit

Operatori logici

Gli operatori logici accettano due operandi e restituiscono un valore booleano. Gli operatori logici hanno priorità diversa. Nella tabella seguente, gli operatori sono elencati in ordine di priorità, da maggiore a minore:

Operatore	Operazione eseguita
`&&`	AND logico
`||`	OR logico

Operatore condizionale

L'operatore condizionale è un operatore ternario, ovvero accetta tre operandi. Può essere utilizzato come metodo rapido per applicare l'istruzione condizionale `if...else`.

Operatore	Operazione eseguita
`?:`	Condizionale

Operatori di assegnazione

Gli operatori di assegnazione accettano due operandi e assegnano un valore a uno di essi in base al valore dell'altro operando. Tutti gli operatori di assegnazione, elencati nella tabella seguente, hanno priorità equivalente.

Operatore	Operazione eseguita
`=`	Assegnazione
`*=`	Assegnazione moltiplicazione
`/=`	Assegnazione divisione
`%=`	Assegnazione modulo
`+=`	Assegnazione addizione
`-=`	Assegnazione sottrazione
`<<=`	Assegnazione spostamento a sinistra bit a bit
`>>=`	Assegnazione spostamento a destra bit a bit
`>>>=`	Assegnazione spostamento a destra senza segno bit a bit
`&=`	Assegnazione AND bit a bit
`^=`	Assegnazione XOR bit a bit
`|=`	Assegnazione OR bit a bit

Istruzioni condizionali

ActionScript 3.0 offre tre istruzioni condizionali di base che possono essere utilizzate per controllare il flusso del programma.

if..else

L'istruzione condizionale `if..else` consente di provare una condizione e quindi eseguire un blocco di codice se la condizione è soddisfatta o un blocco di codice alternativo in caso contrario. Il codice seguente, ad esempio, verifica se il valore di `x` è maggiore di 20 e genera un'istruzione `trace()` in caso affermativo oppure un'istruzione `trace()` diversa in caso negativo.

```
if (x > 20)
{
  trace("x is > 20");
}
else
{
  trace("x is <= 20");
}
```

Se non si desidera eseguire un blocco di codice alternativo, è possibile utilizzare l'istruzione `if` senza l'istruzione `else`.

if..else if

L'istruzione condizionale `if..else if` permette di provare più di una condizione. Il codice seguente, ad esempio, non solo controlla se il valore di `x` è maggiore di 20, ma anche se è negativo:

```
if (x > 20)
{
  trace("x is > 20");
}
else if (x < 0)
{
  trace("x is negative");
}
```

Se un'istruzione `if` o `else` è seguita da un'unica istruzione, non è necessario racchiuderla tra parentesi graffe. Il codice seguente, ad esempio, non utilizza le parentesi graffe:

```
if (x > 0)
  trace("x is positive");
else if (x < 0)
  trace("x is negative");
else
  trace("x is 0");
```

Adobe tuttavia consiglia di includere sempre le parentesi graffe per evitare risultati imprevisti nel caso che in un secondo momento vengano aggiunte altre istruzioni a un'istruzione condizionale priva di parentesi graffe. Ad esempio, nel codice seguente il valore di `positiveNums` viene incrementato di 1 indipendentemente dal fatto che la condizione restituisce o meno il valore `true`:

```
var x:int;
var positiveNums:int = 0;

if (x > 0)
  trace("x is positive");
  positiveNums++;

trace(positiveNums); // 1
```

switch

L'istruzione `switch` è utile quando sono presente vari percorsi di esecuzione che dipendono dalla stessa espressione condizionale. Fornisce una funzionalità simile a una lunga serie di istruzioni `if..else if`, ma risulta di più facile lettura. Anziché verificare se una condizione ha un valore booleano, l'istruzione `switch` valuta un'espressione e usa il risultato per determinare quale blocco di codice deve essere eseguito. I blocchi di codice iniziano con un'istruzione `case` e terminano con un'istruzione `break`. Ad esempio, l'istruzione `switch` seguente genera il giorno della settimana in base al numero del giorno restituito dal metodo `Date.getDay()`:

```
var someDate:Date = new Date();
var dayNum:uint = someDate.getDay();
switch(dayNum)
{
  case 0:
    trace("Sunday");
    break;
  case 1:
    trace("Monday");
    break;
  case 2:
    trace("Tuesday");
    break;
  case 3:
    trace("Wednesday");
    break;
  case 4:
    trace("Thursday");
    break;
  case 5:
    trace("Friday");
    break;
  case 6:
    trace("Saturday");
    break;
  default:
    trace("Out of range");
    break;
}
```

Ripetizione ciclica

Le istruzioni di ripetizione ciclica permettono di eseguire ripetutamente un blocco di codice specifico utilizzando una serie di valori o di variabili. Adobe consiglia di racchiudere sempre il blocco di codice tra parentesi graffe (`{}`). Benché sia possibile omettere le parentesi graffe se il blocco di codice contiene una sola istruzione, questa pratica non è consigliata per lo stesso motivo per cui non è consigliata per le istruzioni condizionali: aumenta la probabilità che le istruzioni aggiunte in futuro vengano inavvertitamente escluse dal blocco di codice. Se, in un secondo momento, si aggiunge un'istruzione da includere nel blocco di codice ma si dimentica di aggiungere anche le necessarie parentesi graffe, l'istruzione verrà ignorata al momento dell'esecuzione del ciclo.

for

Il ciclo `for` permette di eseguire iterazioni su una variabile per verificare un intervallo di valori specifico. A un'istruzione `for` è necessario fornire tre espressioni: una variabile impostata su un valore iniziale, un'istruzione condizionale che determina quando il ciclo termina e un'espressione che cambia il valore della variabile a ogni ciclo. Il codice seguente, ad esempio, esegue cinque iterazioni. Il valore della variabile `i` inizia a 0 e termina a 4 e l'output è rappresentato dai numeri da 0 a 4, ciascuno su una riga separata.

```
var i:int;
for (i = 0; i < 5; i++)
{
  trace(i);
}
```

for..in

Il ciclo `for..in` esegue un'iterazione sulle proprietà di un oggetto o sugli elementi di un array. È possibile, ad esempio, ricorrere a un ciclo `for...in` per eseguire iterazioni sulle proprietà di un oggetto generico (le proprietà degli oggetti non vengono ordinate in base a criteri particolari, ma inserite in ordine casuale):

```
var myObj:Object = {x:20, y:30};
for (var i:String in myObj)
{
  trace(i + ": " + myObj[i]);
}
// output:
// x: 20
// y: 30
```

È possibile anche eseguire iterazioni sugli elementi di un array:

```
var myArray:Array = ["one", "two", "three"];
for (var i:String in myArray)
{
  trace(myArray[i]);
}
// output:
// uno
// due
// tre
```

Non è invece possibile eseguire iterazioni sulle proprietà degli oggetti che sono istanze di classi definite dall'utente, a meno che la classe non sia dinamica. Anche in quest'ultimo caso, comunque, è possibile eseguire iterazioni solo sulle proprietà aggiunte in modo dinamico.

for each..in

Il ciclo `for each..in` esegue un'iterazione sulle voci di una raccolta, che possono essere tag contenuti in un oggetto XML o XMLList, i valori delle proprietà di un oggetto o gli elementi di un array. Ad esempio, come illustra il codice seguente, è possibile utilizzare un ciclo `for each..in` per eseguire un'iterazione sulle proprietà di un oggetto generico, tuttavia, a differenza di quanto avviene con il ciclo `for..in`, la variabile di iterazione di un ciclo `for each..in` contiene il valore della proprietà anziché il suo nome:

```
var myObj:Object = {x:20, y:30};
for each (var num in myObj)
{
  trace(num);
}
// output:
// 20
// 30
```

È possibile eseguire l'iterazione su un oggetto XML o XMLList, come mostra l'esempio seguente:

```
var myXML:XML = <users>
                   <fname>Jane</fname>
                   <fname>Susan</fname>
                   <fname>John</fname>
                </users>;

for each (var item in myXML.fname)
{
  trace(item);
}
/* output
Jane
Susan
John
*/
```

È anche possibile eseguire iterazioni sugli elementi di un array, come nel codice seguente:

```
var myArray:Array = ["one", "two", "three"];
for each (var item in myArray)
{
  trace(item);
}
// output:
// uno
// due
// tre
```

Non è invece possibile eseguire iterazioni sulle proprietà degli oggetti che sono istanze di classi chiuse. Nel caso di classi dinamiche, non è possibile eseguire l'iterazione sulle proprietà fisse, ovvero le proprietà definite nella definizione della classe.

while

Il ciclo `while` è come un'istruzione `if` che viene ripetuta fintanto che la condizione è `true`. Ad esempio, il codice seguente produce lo stesso output dell'esempio di ciclo `for`:

```
var i:int = 0;
while (i < 5)
{
  trace(i);
  i++;
}
```

Il ciclo `while` presenta uno svantaggio rispetto al ciclo `for`: è più facile scrivere cicli infiniti. A differenza del ciclo `for`, l'esempio di codice del ciclo `while` viene compilato anche se si omette l'espressione che incrementa la variabile del contatore. Senza l'espressione che incrementa `i`, il ciclo diventa infinito.

do..while

Il ciclo `do..while` è un ciclo `while` che garantisce che il blocco di codice venga eseguito almeno una volta, perché la condizione viene verificata dopo l'esecuzione del blocco di codice. Il codice seguente mostra un esempio semplice di un ciclo `do..while` che genera un output anche se la condizione non è soddisfatta:

```
var i:int = 5;
do
{
  trace(i);
  i++;
} while (i < 5);
// output: 5
```

Funzioni

Le *funzioni* sono blocchi di codice che eseguono operazioni specifiche e possono essere riutilizzati all'interno del programma. In ActionScript 3.0 esistono due tipi di funzioni: i *metodi* e le *chiusure di funzione*. Una funzione viene chiamata metodo oppure chiusura di funzione a seconda del contesto nel quale è definita. Se la funzione viene definita all'interno di una definizione di classe o associata a un'istanza di un oggetto, prende il nome di metodo. Se viene definita in qualunque altro modo, viene chiamata chiusura di funzione.

Le funzioni hanno sempre avuto un ruolo estremamente importante in ActionScript. In ActionScript 1.0, ad esempio, la parola chiave `class` non esisteva, quindi le "classi" erano definite dalle funzioni di costruzione. Anche se nel frattempo la parola chiave `class` è stata aggiunta al linguaggio, una conoscenza approfondita delle funzioni è ancora importante per sfruttare nel modo migliore le possibilità offerte da ActionScript. Questo compito può risultare più impegnativo per i programmatori che si aspettano un comportamento delle funzioni ActionScript analogo a quello di linguaggi come C++ o Java. Anche se le procedure di base per definire e chiamare le funzioni non dovrebbero rappresentare un problema per i programmatori esperti, alcune delle caratteristiche più avanzate di ActionScript richiedono un approfondimento.

Concetti di base delle funzioni

Questa sezione descrive le tecniche di base utilizzate per definire e chiamare le funzioni.

Chiamate di funzione

È possibile chiamare una funzione specificando il relativo identificatore seguito dall'operatore parentesi (`()`). L'operatore parentesi ha il compito di racchiudere gli eventuali parametri che si desidera inviare alla funzione. Ad esempio, la funzione `trace()`, che è una funzione di primo livello nell'API di Flash Player, è utilizzata spessissimo in questo manuale:

```
trace("Use trace to help debug your script");
```

Se si chiama una funzione senza parametri, è necessario includere una coppia di parentesi vuote. Ad esempio, è possibile utilizzare il metodo `Math.random()`, che non accetta parametri, per generare un numero casuale:

```
var randomNum:Number = Math.random();
```

Definizione di funzioni personalizzate

In ActionScript 3.0 esistono due modi per definire una funzione: mediante un'istruzione di funzione oppure un'espressione di funzione. A seconda che si prediliga uno stile di programmazione più statico o dinamico, è possibile scegliere una o l'altra tecnica. In genere, chi usa le istruzioni per definire le funzioni preferisce la programmazione statica (modalità rigorosa). Le espressioni vengono invece utilizzate per definire le funzioni quando esiste un'esigenza specifica in questo senso, solitamente nella programmazione dinamica (modalità standard).

Istruzioni di funzione

Le istruzioni di funzione sono la tecnica preferita per definire le funzioni in modalità rigorosa. Un'istruzione di funzione inizia con la parola chiave `function`, seguita da:

- Il nome della funzione
- I parametri, separati da virgole e racchiusi tra parentesi
- Il corpo della funzione, ovvero il codice ActionScript da eseguire quando la funzione viene chiamata, racchiuso tra parentesi graffe

Ad esempio, il codice seguente crea una funzione che definisce un parametro, quindi chiama la funzione utilizzando la stringa `"hello"` come valore del parametro:

```
function traceParameter(aParam:String)
{
  trace(aParam);
}

traceParameter("hello"); // hello
```

Espressioni di funzione

Il secondo modo per dichiarare una funzione prevede l'uso di un'istruzione di assegnazione con un'espressione di funzione, che talvolta viene anche definita letterale di funzione o funzione anonima. Si tratta di un metodo più verboso, largamente utilizzato nelle versioni precedenti di ActionScript.

Un'istruzione di assegnazione con un'espressione di funzione inizia con la parola chiave `var`, seguita da:

- Il nome della funzione
- L'operatore due punti (`:`)
- La classe `Function` per indicare il tipo di dati
- L'operatore di assegnazione (=)
- La parola chiave `function`
- I parametri, separati da virgole e racchiusi tra parentesi
- Il corpo della funzione, ovvero il codice ActionScript da eseguire quando la funzione viene chiamata, racchiuso tra parentesi graffe

Ad esempio, il codice seguente dichiara la funzione `traceParameter` utilizzando un'espressione di funzione:

```
var traceParameter:Function = function (aParam:String)
{
  trace(aParam);
};
traceParameter("hello"); // hello
```

Si noti che non occorre specificare un nome di funzione come avviene nelle istruzioni di funzione. Un'altra importante differenza tra le espressioni di funzione e le istruzioni di funzione consiste nel fatto che un'espressione di funzione è appunto un'espressione e non un'istruzione. Ciò significa che un'espressione di funzione, a differenza di un'istruzione di funzione, non può esistere come elemento autonomo, bensì può essere utilizzata solo all'interno di un'istruzione, solitamente un'istruzione di assegnazione. L'esempio seguente mostra un'espressione di funzione assegnata a un elemento array:

```
var traceArray:Array = new Array();
traceArray[0] = function (aParam:String)
{
  trace(aParam);
};
traceArray[0]("hello");
```

Scelta tra istruzioni ed espressioni

Come regola generale, utilizzare un'istruzione di funzione a meno che le circostanze specifiche non suggeriscano l'uso di un'espressione. Le istruzioni di funzione sono meno verbose e, rispetto alle espressioni di funzione, producono risultati più omogenei tra modalità rigorosa e modalità standard.

Le istruzioni di funzione sono più facili da leggere delle istruzioni di assegnazione che contengono espressioni di funzione, consentono di scrivere codice più conciso e creano meno confusione delle espressioni di funzione, che richiedono l'uso delle due parole chiave `var` e `function`.

Inoltre, le istruzioni di funzione producono risultati più omogenei tra le due modalità del compilatore perché consentono di utilizzare la sintassi del punto sia in modalità rigorosa che standard per richiamare un metodo dichiarato mediante un'istruzione di funzione, il che non è sempre possibile per i metodi dichiarati con un'espressione di funzione. Ad esempio, il codice seguente definisce una classe denominata Example con due metodi: `methodExpression()`, dichiarato con un'espressione di funzione, e `methodStatement()`, dichiarato con un'istruzione di funzione. In modalità rigorosa non è possibile utilizzare la sintassi del punto per richiamare il metodo `methodExpression()`.

```
class Example
{
  var methodExpression = function() {}
  function methodStatement() {}
}

var myEx:Example = new Example();
myEx.methodExpression(); // Errore in modalità rigorosa;
                         // OK in modalità standard
myEx.methodStatement(); // OK in modalità rigorosa e standard
```

Le espressioni di funzione sono generalmente più indicate per la programmazione che privilegia il comportamento runtime, ovvero dinamico. Se si preferisce utilizzare la modalità rigorosa ma si ha anche la necessità di chiamare un metodo dichiarato con un'espressione di funzione, è possibile utilizzare entrambe le tecniche. Innanzi tutto, è possibile chiamare il metodo utilizzando le parentesi quadre (`[]`) invece dell'operatore punto (`.`). Il metodo seguente ha esito positivo sia in modalità rigorosa che standard:

```
myExample["methodLiteral"]();
```

In secondo luogo, è possibile dichiarare l'intera classe come classe dinamica. Sebbene in questo modo sia possibile chiamare il metodo utilizzando l'operatore punto, la funzionalità della modalità rigorosa viene parzialmente sacrificata per tutte le istanze di tale classe. Ad esempio, il compilatore non genera un errore se si tenta di accedere a una proprietà non definita su un'istanza di una classe dinamica.

In determinate circostanze le espressioni di funzione risultano utili. Un caso frequente è quello delle funzioni che vengono utilizzate una sola volta e quindi eliminate. Un altro utilizzo, meno comune, riguarda l'associazione di una funzione a una proprietà prototype. Per ulteriori informazioni, vedere "Oggetto prototype" a pagina 190.

Esistono due sottili differenze tra le istruzioni di funzione e le espressioni di funzione di cui va tenuto conto quando si sceglie la tecnica da utilizzare. La prima è che un'espressione di funzione non viene considerata come oggetto indipendente ai fini della gestione della memoria e del processo di garbage collection. In altre parole, quando si assegna un'espressione di funzione a un altro oggetto, ad esempio a un elemento di array o una proprietà di un oggetto, nel codice viene creato semplicemente un riferimento all'espressione di funzione. Se l'array o l'oggetto al quale è associata l'espressione di funzione esce dall'area di validità o comunque cessa di essere disponibile, non è più possibile accedere all'espressione. Se l'array o l'oggetto viene eliminato, la memoria utilizzata dall'espressione di funzione diventa disponibile per il processo di garbage collection, ovvero può essere riutilizzata per altri scopi.

L'esempio seguente mostra che, se viene eliminata la proprietà alla quale è assegnata un'espressione di funzione, la funzione non è più disponibile. La classe Test è dinamica e consente quindi di aggiungere una proprietà denominata `functionExp` che contiene un'espressione di funzione. La funzione `functionExp()` può essere chiamata con l'operatore punto, ma non è più accessibile dopo l'eliminazione della proprietà `functionExp`.

```
dynamic class Test {}
var myTest:Test = new Test();

// espressione di funzione
myTest.functionExp = function () { trace("Function expression") };
myTest.functionExp();   // Espressione di funzione
delete myTest.functionExp;
myTest.functionExp();   // Errore
```

Se, al contrario, la funzione viene inizialmente definita mediante un'istruzione di funzione, esiste come oggetto autonomo e continua a esistere anche dopo l'eliminazione della proprietà alla quale è associata. L'operatore `delete` funziona solo sulle proprietà degli oggetti, quindi non ha effetto se viene utilizzato per eliminare la funzione `stateFunc()`.

```
dynamic class Test {}
var myTest:Test = new Test();

// Istruzione di funzione
function stateFunc() { trace("Function statement") }
myTest.statement = stateFunc;
myTest.statement(); // Istruzione di funzione
delete myTest.statement;
delete stateFunc;   // Nessun effetto
stateFunc();        // Istruzione di funzione
myTest.statement(); // Errore
```

La seconda differenza tra un'istruzione di funzione e un'espressione di funzione consiste nel fatto che la prima esiste in tutta l'area di validità nella quale è definita, comprese le istruzioni che la precedono. Un'espressione di funzione, al contrario, viene definita solo per le istruzioni successive. Ad esempio, il codice seguente contiene una chiamata (che ha esito positivo) alla funzione `scopeTest()` prima che venga definita:

```
statementTest(); // statementTest

function statementTest():void
{
  trace("statementTest");
}
```

Le espressioni di funzione non sono disponibili prima della posizione in cui vengono definite, quindi il codice seguente genera un errore runtime:

```
expressionTest(); // Errore runtime

var expressionTest:Function = function ()
{
  trace("expressionTest");
}
```

Restituzione di valori da funzioni

Per restituire un valore da una funzione, utilizzare l'istruzione `return` seguita dall'espressione o dal valore letterale da restituire. Il seguente codice, ad esempio, restituisce un'espressione corrispondente al parametro:

```
function doubleNum(baseNum:int):int
{
  return (baseNum * 2);
}
```

Si noti che l'istruzione `return` termina la funzione, quindi eventuali istruzioni successive a un'istruzione `return` non vengono eseguite, come nell'esempio seguente:

```
function doubleNum(baseNum:int):int {
  return (baseNum * 2);
  trace("after return"); // Questa istruzione trace non viene eseguita.
}
```

In modalità rigorosa è necessario restituire un valore del tipo appropriato se si sceglie di specificare un tipo restituito. Ad esempio, il codice seguente genera un errore in modalità rigorosa perché non restituisce un valore valido:

```
function doubleNum(baseNum:int):int
{
  trace("after return");
}
```

Funzioni nidificate

È possibile nidificare le funzioni, cioè dichiararle all'interno di altre funzioni. Una funzione nidificata è disponibile solo all'interno della funzione principale in cui è contenuta, a meno che non venga passato al codice esterno un riferimento alla funzione. Ad esempio, il codice seguente dichiara due funzioni nidificate all'interno della funzione `getNameAndVersion()`:

```
function getNameAndVersion():String
{
  function getVersion():String
  {
    return "9";
  }
  function getProductName():String
  {
    return "Flash Player";
  }
  return (getProductName() + " " + getVersion());
}
trace(getNameAndVersion()); // Flash Player 9
```

Se passate al codice esterno, le funzioni nidificate vengono passate come chiusure di funzione, vale a dire che la funzione conserva le definizioni che si trovano nell'area di validità nel momento in cui viene definita. Per ulteriori informazioni, vedere "Chiusure di funzione" a pagina 144.

Parametri di funzione

ActionScript 3.0 introduce alcune funzionalità relative ai parametri di funzione che potrebbero sembrare inedite ai programmatori che iniziano a utilizzare questo linguaggio. Benché l'idea di passare i parametri mediante un valore o un riferimento risulti probabilmente familiare alla maggior parte dei programmatori, l'oggetto `arguments` e il parametro `. . . (rest)` potrebbero invece rappresentare una novità.

Passaggio di argomenti mediante un valore o un riferimento

In molti linguaggi di programmazione, è importante comprendere la distinzione che esiste tra passare gli argomenti mediante un valore oppure mediante un riferimento, poiché tale distinzione può influire su come viene progettato il codice.

Passare un argomento mediante un valore significa copiarne il valore in una variabile locale da utilizzare con la funzione. Al contrario, si specifica un argomento mediante un riferimento, viene passato solo un riferimento all'argomento e non il valore effettivo. Nel secondo caso non viene quindi creata una copia dell'argomento vero e proprio, bensì viene passato come argomento un riferimento alla variabile nel momento in cui l'argomento viene creato e assegnato a una variabile locale da utilizzare nella funzione. Poiché è un riferimento a una variabile esterna alla funzione, la variabile locale consente di modificare il valore della variabile originale.

In ActionScript 3.0, tutti gli argomento vengono passati mediante riferimento perché tutti i valori sono memorizzati come oggetti. Tuttavia, gli oggetti che appartengono ai tipi di dati di base (Boolean, Number, int, uint e String) prevedono l'uso di operatori speciali grazie ai quali possono comportarsi come se venissero passati mediante un valore. Ad esempio, il codice seguente crea una funzione denominata `passPrimitives()` che definisce due parametri chiamati `xParam` e `yParam`, entrambi del tipo int. Questi parametri sono simili a variabili locali dichiarate nel corpo della funzione `passPrimitives()`. Quando la funzione viene chiamata con gli argomenti `xValue` e `yValue`, i parametri `xParam` e `yParam` vengono inizializzati mediante riferimenti agli oggetti int rappresentati da `xValue` e `yValue`. Poiché gli argomenti appartengono a un tipo di base, si comportano come se fossero passati mediante valore. Benché `xParam` e `yParam` contengano inizialmente solo riferimenti agli oggetti `xValue` e `yValue`, qualunque modifica delle variabili all'interno del corpo della funzione genera nuove copie dei valori in memoria.

```
function passPrimitives(xParam:int, yParam:int):void
{
  xParam++;
  yParam++;
  trace(xParam, yParam);
}

var xValue:int = 10;
var yValue:int = 15;
trace(xValue, yValue);          // 10 15
passPrimitives(xValue, yValue); // 11 16
trace(xValue, yValue);          // 10 15
```

All'interno della funzione `passPrimitives()`, i valori di `xParam` e `yParam` vengono incrementati, tuttavia ciò non influisce sui valori di `xValue` e `yValue`, come mostra l'ultima istruzione `trace`. Lo stesso varrebbe anche nel caso in cui i parametri avessero gli stessi nomi delle variabili, `xValue` e `yValue`, perché i valori `xValue` e `yValue` all'interno della funzione farebbero riferimento a nuove posizioni di memoria, distinte dalle variabili omonime esterne alla funzione.

Tutti gli altri oggetti, ovvero gli oggetti che non appartengono ai tipi di dati primitivi, vengono sempre passati mediante riferimento, quindi con la possibilità di modificare il valore della variabile originale. Ad esempio, il codice seguente crea un oggetto denominato `objVar` con due proprietà, `x` e `y`. L'oggetto viene passato come argomento alla funzione `passByRef()`. Poiché non appartiene a un tipo di base, l'oggetto non viene semplicemente passato mediante un riferimento, ma rimane un riferimento. Ciò significa che le modifiche apportate ai parametri all'interno della funzione avranno effetto sulle proprietà dell'oggetto all'esterno della funzione.

```
function passByRef(objParam:Object):void
{
  objParam.x++;
  objParam.y++;
  trace(objParam.x, objParam.y);
}
var objVar:Object = {x:10, y:15};
trace(objVar.x, objVar.y); // 10 15
passByRef(objVar);         // 11 16
trace(objVar.x, objVar.y); // 11 16
```

Il parametro `objParam` fa riferimento allo stesso oggetto della variabile globale `objVar`. Come si può notare nelle istruzioni `trace` dell'esempio, le modifiche apportate alle proprietà `x` e `y` dell'oggetto `objParam` vengono applicate anche all'oggetto `objVar`.

Valori predefiniti dei parametri

In ActionScript 3.0 è stata introdotta la possibilità di dichiarare dei *valori di parametro predefiniti* per una funzione. Se in una chiamata a una funzione con parametri predefiniti viene omesso un parametro con valori predefiniti, viene utilizzato il valore specificato per quel parametro nella definizione della funzione. Tutti i parametri con valori predefiniti devono essere posizionati alla fine dell'elenco dei parametri. I valori assegnati come predefiniti devono essere costanti della fase di compilazione. L'esistenza di un valore predefinito per un parametro fa sì che quel parametro diventi un *parametro opzionale*, mentre un parametro privo di valore predefinito viene considerato un *parametro obbligatorio*.

Ad esempio, il codice seguente crea una funzione con tre parametri, due dei quali hanno valori predefiniti. Quando la funzione viene chiamata con un solo parametro, vengono utilizzati i valori predefiniti dei parametri.

```
function defaultValues(x:int, y:int = 3, z:int = 5):void
{
  trace(x, y, z);
}
defaultValues(1); // 1 3 5
```

L'oggetto arguments

Quando si passano dei parametri a una funzione, è possibile utilizzare l'oggetto `arguments` per accedere alle informazioni relative a tali parametri. Seguono alcune osservazioni importanti relative all'oggetto `arguments`:

- L'oggetto `arguments` è un array che include tutti i parametri passati alla funzione.
- La proprietà `arguments.length` segnala il numero di parametri passati alla funzione.
- La proprietà `arguments.callee` fornisce un riferimento alla funzione stessa, che è utile per le chiamate ricorsive alle espressioni di funzione.

NOTA L'oggetto `arguments` non è disponibile se è presente un parametro denominato `arguments` oppure se si utilizza il parametro ... (rest).

ActionScript 3.0 consente di includere nelle chiamate di funzione più parametri di quelli definiti nella definizione della funzione; tuttavia, in modalità rigorosa viene generato un errore del compilatore se il numero di parametri è inferiore a quello dei parametri obbligatori. È possibile ricorrere alla funzionalità di array dell'oggetto `arguments` per accedere a qualunque parametro passato alla funzione, a prescindere che sia o meno definito nella definizione della funzione. L'esempio seguente utilizza l'array `arguments` con la proprietà `arguments.length` per tracciare tutti i parametri passati alla funzione `traceArgArray()`:

```
function traceArgArray(x:int):void
{
  for (var i:uint = 0; i < arguments.length; i++)
  {
    trace(arguments[i]);
  }
}

traceArgArray(1, 2, 3);

// output:
// 1
// 2
// 3
```

La proprietà `arguments.callee` viene spesso utilizzata nelle funzioni anonime per creare la ricorsività e rendere il codice più flessibile. Se il nome di una funzione ricorsiva cambia durante il ciclo di sviluppo, non occorre modificare la chiamata ricorsiva nel corpo della funzione se si utilizza `arguments.callee` al posto del nome della funzione. Nell'espressione di funzione seguente, la proprietà `arguments.callee` viene utilizzata per abilitare la ricorsività:

```
var factorial:Function = function (x:uint)
{
  if(x == 0)
  {
    return 1;
  }
  else
  {
    return (x * arguments.callee(x - 1));
  }
}

trace(factorial(5)); // 120
```

Se si utilizza il parametro ... (rest) nella dichiarazione della funzione, l'oggetto `arguments` non è disponibile e per accedere ai parametri è necessario utilizzare i rispettivi nomi che sono stati dichiarati.

È inoltre importante evitare di utilizzare la stringa `"arguments"` come nome di parametro perché impedisce l'uso dell'oggetto `arguments`. Ad esempio, se la funzione `traceArgArray()` viene riscritta con l'aggiunta di un parametro `arguments`, i riferimenti a `arguments` nel corpo della funzione sono relativi al parametro anziché all'oggetto `arguments`. Il codice seguente non produce alcun output:

```
function traceArgArray(x:int, arguments:int):void
{
  for (var i:uint = 0; i < arguments.length; i++)
  {
    trace(arguments[i]);
  }
}

traceArgArray(1, 2, 3);

// Nessun output
```

Nelle versioni precedenti di ActionScript, l'oggetto `arguments` conteneva anche una proprietà denominata `caller`, che era un riferimento alla funzione che chiamava la funzione corrente. La proprietà `caller` non è presente in ActionScript 3.0, ma se occorre fare riferimento alla funzione chiamante, è possibile modificare quest'ultima in modo che passi un parametro supplementare contenente un riferimento a se stessa.

Il parametro ... (rest)

In ActionScript 3.0 è stata introdotta una nuova dichiarazione di parametro, il parametro ... (rest), che consente di specificare un parametro array che accetta qualunque numero di argomenti separati da virgole. Il parametro può avere qualsiasi nome che non corrisponda a una parola riservata e deve essere l'ultimo parametro specificato. L'uso di questo parametro rende indisponibile l'oggetto `arguments`. Anche se il parametro ... (rest) offre la stessa funzionalità dell'array `arguments` e della proprietà `arguments.length`, non fornisce invece una funzionalità simile a quella di `arguments.callee`. Prima di usare il parametro ... (rest), assicurarsi che non sia necessario utilizzare `arguments.callee`.

Il seguente esempio riscrive la funzione `traceArgArray()` utilizzando il parametro ... (rest) invece dell'oggetto `arguments`:

```
function traceArgArray(... args):void
{
  for (var i:uint = 0; i < args.length; i++)
  {
    trace(args[i]);
  }
}

traceArgArray(1, 2, 3);

// output:
// 1
// 2
// 3
```

Il parametro ... (rest) può anche essere utilizzato con altri parametri, a condizione che venga specificato per ultimo. L'esempio seguente modifica la funzione `traceArgArray()` in modo tale che il primo parametro, `x`, sia del tipo int, e il secondo utilizzi il parametro ... (rest). L'output ignora il primo valore perché il primo parametro non fa più parte dell'array creato dal parametro ... (rest).

```
function traceArgArray(x: int, ... args)
{
  for (var i:uint = 0; i < args.length; i++)
  {
    trace(args[i]);
  }
}

traceArgArray(1, 2, 3);

// output:
// 2
// 3
```

Funzioni come oggetti

In ActionScript 3.0, le funzioni sono oggetti. Quando si crea una funzione, ciò che viene creato è in realtà un oggetto che può non solo essere passato come parametro a un'altra funzione, ma anche disporre di proprietà e metodi.

Le funzioni specificate come argomenti per altre funzioni vengono passate mediante riferimento e non mediante un valore. Quando si passa una funzione come argomento, si utilizza solo l'identificatore e si omette l'operatore parentesi usato per chiamare il metodo. Ad esempio, il codice seguente passa una funzione denominata `clickListener()` come argomento al metodo `addEventListener()`:

```
addEventListener(MouseEvent.CLICK, clickListener);
```

Anche il metodo `Array.sort()` definisce un parametro che accetta una funzione. Per un esempio di funzione di ordinamento personalizzata utilizzata come argomento per la funzione `Array.sort()`, vedere "Ordinamento di un array" a pagina 246.

Per quanto possa sembrare strano ai programmatori che iniziano a utilizzare ActionScript, le funzioni possono avere proprietà e metodi come qualunque altro oggetto. In effetti, ogni funzione dispone di una proprietà di sola lettura denominata `length` che memorizza il numero di parametri definiti per la funzione. Questa proprietà è diversa dalla proprietà `arguments.length`, che indica il numero di argomenti passati alla funzione. È bene ricordare che in ActionScript il numero di argomenti inviati a una funzione può superare quello dei parametri definiti per la stessa funzione. L'esempio seguente, che viene compilato solo in modalità standard perché la modalità rigorosa richiede una corrispondenza esatta tra il numero di argomenti passati e il numero di parametri definiti, mostra la differenza tra le due proprietà:

```
function traceLength(x:uint, y:uint):void
{
  trace("arguments received: " + arguments.length);
  trace("arguments expected: " + traceLength.length);
}

traceLength(3, 5, 7, 11);
/* output:
arguments received: 4
arguments expected: 2 */
```

È possibile definire proprietà personalizzate per la funzione all'esterno del corpo della funzione. Le proprietà di funzione possono servire come proprietà "quasi statiche" che consentono di salvare lo stato di una variabile relativo alla funzione. Ad esempio, potrebbe essere utile registrare quante volte viene utilizzata una particolare funzione. Questa funzionalità può servire se si sta creando un videogame e si desidera registrare quante volte un utente utilizza un comando specifico (benché sia anche possibile utilizzare una proprietà di classe statica per lo stesso scopo). Il codice seguente crea una proprietà di funzione all'esterno della dichiarazione della funzione e incrementa tale proprietà ogni volta che la funzione viene chiamata:

```
someFunction.counter = 0;

function someFunction():void
{
  someFunction.counter++;
}

someFunction();
someFunction();
trace(someFunction.counter); // 2
```

Area di validità delle funzioni

L'area di validità di una funzione determina non solo l'area in cui, all'interno di un programma, quella funzione può essere chiamata, ma anche le definizioni alle quali la funzione può accedere. Le stesse regole dell'area di validità che valgono per gli identificatori delle variabili si applicano anche agli identificatori di funzione. Una funzione dichiarata nell'area di validità globale è disponibile in tutto il codice. Ad esempio, ActionScript 3.0 contiene funzioni globali, quali `isNaN()` e `parseInt()`, che sono disponibili in qualunque punto del codice. Una funzione nidificata (cioè dichiarata all'interno di un'altra funzione) può essere utilizzata in qualunque posizione all'interno della funzione in cui è stata dichiarata.

La catena dell'area di validità

Ogni volta che inizia l'esecuzione di una funzione, viene creata una serie di oggetti e di proprietà. Innanzi tutto, viene creato un oggetto speciale chiamato *oggetto di attivazione*, nel quale vengono memorizzati i parametri e le eventuali variabili locali o funzioni dichiarate nel corpo della funzione. Non è possibile accedere direttamente all'oggetto di attivazione perché è un meccanismo interno. In secondo luogo viene creata una *catena dell'area di validità* che contiene un elenco ordinato degli oggetti nei quali Flash Player cerca le dichiarazioni di identificazione (gli identificatori). Ogni funzione che viene eseguita ha una catena dell'area di validità che viene memorizzata in una proprietà interna. Nel caso di una funzione nidificata, la catena dell'area di validità inizia con il proprio oggetto di attivazione, seguito dall'oggetto di attivazione della relativa funzione principale. La catena prosegue nello stesso modo fino al raggiungimento dell'oggetto globale, ovvero l'oggetto che viene creato all'inizio di un programma ActionScript e che contiene tutte le variabili globali e le funzioni.

Chiusure di funzione

Una *chiusura di funzione* è un oggetto che contiene un'istantanea della funzione e il relativo *ambiente lessicale*, il quale comprende tutte le variabili, le proprietà, i metodi e gli oggetti inclusi nella catena dell'area di validità della funzione, con i rispettivi valori. Le chiusure di funzione vengono create ogni volta che una funzione viene eseguita indipendentemente da un oggetto o da una classe. Il fatto che una chiusura di funzione mantenga l'area di validità nella quale è stata definita produce risultati interessanti quando una funzione viene passata in un'area di validità diversa come argomento o come valore restituito.

Ad esempio, il codice seguente crea due funzioni: `foo()`, che restituisce una funzione nidificata di nome `rectArea()` che calcola l'area di un rettangolo, e `bar()`, che chiama `foo()` e memorizza la chiusura di funzione restituita in una variabile denominata `myProduct`. Anche se la funzione `bar()` definisce la propria variabile locale `x` (con valore 2), quando la chiusura di funzione `myProduct()` viene chiamata, essa mantiene la variabile `x` (con valore 40) definita nella funzione `foo()`. La funzione `bar()`, pertanto, restituisce il valore `160` anziché `8`.

```
function foo():Function
{
  var x:int = 40;
  function rectArea(y:int):int // Chiusura di funzione definita
  {
    return x * y
  }
  return rectArea;
}
```

```
function bar():void
{
  var x:int = 2;
  var y:int = 4;
  var myProduct:Function = foo();
  trace(myProduct(4)); // Chiusura di funzione chiamata
}
bar(); // 160
```

I metodi si comportano in modo analogo, perché conservano a loro volta le informazioni relative all'ambiente lessicale nel quale sono stati creati. Questa caratteristica è evidente soprattutto quando un metodo viene estratto dalla propria istanza per creare un metodo vincolato. La differenza principale tra una chiusura di funzione e un metodo vincolato consiste nel fatto che il valore della parola chiave `this` in un metodo vincolato fa sempre riferimento all'istanza alla quale è stata originariamente associata, mentre in una chiusura di funzione il valore di `this` può cambiare. Per ulteriori informazioni, vedere "Metodi vincolati" a pagina 165.

CAPITOLO 4

Programmazione orientata agli oggetti con ActionScript

4

In questo capitolo vengono descritti gli elementi di ActionScript che supportano la programmazione orientata agli oggetti (OOP). Non vengono descritti i principi generali della programmazione orientata agli oggetti, quali la progettazione degli oggetti, l'astrazione, l'incapsulamento, l'ereditarietà e il polimorfismo. Il capitolo spiega come è possibile applicare questi principi utilizzando ActionScript 3.0.

Poiché le basi di ActionScript sono quelle di un linguaggio per la creazione di script, il supporto della programmazione OOP da parte di ActionScript 3.0 è opzionale. I programmatori dispongono pertanto di una notevole flessibilità nella scelta dell'approccio migliore per i progetti, a seconda del livello di complessità e dell'ambito di applicazione. Per operazioni di portata limitata, l'uso di ActionScript con un paradigma di programmazione procedurale può risultare del tutto sufficiente. Per i progetti di grandi dimensioni, l'applicazione dei principi della programmazione a oggetti può rendere il codice più facile da comprendere, gestire ed estendere.

Sommario

Elementi fondamentali della programmazione orientata agli oggetti

Introduzione alla programmazione orientata agli oggetti

La programmazione orientata agli oggetti (OOP, Object Oriented Programming) è un sistema di organizzazione del codice di un programma mediante raggruppamento all'interno di oggetti, ovvero singoli elementi che includono informazioni (valori di dati) e funzionalità. L'uso di un approccio orientato agli oggetti per organizzare un programma consente di raggruppare particolari informazioni (ad esempio, informazioni sui brani musicali, quali titolo dell'album, titolo della traccia o nome dell'artista) insieme a funzionalità o azioni comuni associate a tali informazioni (quali "aggiungi traccia all'elenco di riproduzione" o "riproduci tutti i brani di questo artista"). Tali voci vengono combinate in un'unica voce, un oggetto (ad esempio, un oggetto "Album" o "MusicTrack"). Essere in grado di raggruppare insieme tali valori e funzioni offre vari vantaggi, inclusa la necessità di tenere traccia di una sola variabile, anziché di più variabili, organizzando tutte le funzioni correlate insieme e strutturando i programmi in modo più vicino al mondo reale.

Comuni attività di programmazione orientata agli oggetti

In pratica, la programmazione orientata agli oggetti consiste in due parti. Una parte riguarda le strategie e le tecniche di progettazione di un programma (spesso chiamata anche *progettazione orientata agli oggetti*). Si tratta di un argomento molto ampio, che non verrà trattato in questo capitolo. L'altra parte della programmazione a oggetti riguarda le strutture di programmazione vere e proprie disponibili in un determinato linguaggio di programmazione e necessarie per costruire un programma seguendo un approccio orientato agli oggetti. In questo capitolo vengono trattate le seguenti attività comuni della programmazione a oggetti:

- Definizione di classi
- Creazione di proprietà, metodi e proprietà accessor get e set (metodi supplementari)
- Controllo dell'accesso a classi, proprietà, metodi e proprietà accessor
- Creazione delle proprietà e dei metodi statici
- Creazione di strutture simili a enumerazioni
- Definizione e uso delle interfacce
- Uso dell'ereditarietà, inclusa la sostituzione degli elementi di classe

Concetti e termini importanti

Il seguente elenco di riferimento contiene termini importanti utilizzati nel presente capitolo:

- Attributo: caratteristica assegnata a un elemento di classe (quale una proprietà o un metodo) nella definizione di classe. Gli attributi vengono generalmente usati per definire se la proprietà o il metodo sarà accessibile dal codice in altre parti del programma. Ad esempio, `private` e `public` sono attributi. Un metodo private può essere chiamato solo dal codice all'interno della classe, mentre un metodo public può essere chiamato da qualsiasi codice del programma.
- Classe: definizione della struttura e comportamento degli oggetti di un determinato tipo (come un modello o progetto base per oggetti di quel tipo di dati).
- Gerarchia di classe: struttura di più classi correlate che specifica quali classi ereditano funzionalità da altre classi.
- Funzione di costruzione: particolare metodo definibile in una classe che viene chiamato quando viene creata un'istanza di tale classe. Una funzione di costruzione è comunemente utilizzata per specificare valori predefiniti o per eseguire operazioni di impostazione per l'oggetto.
- Tipo di dati: tipo di informazioni che una particolare variabile può memorizzare. In generale, *tipo di dati* ha lo stesso significato di *classe*.
- Operatore punto: segno di punto (`.`) utilizzato da ActionScript e da altri linguaggi di programmazione per indicare che un nome è riferito a un elemento secondario di un oggetto (quale una proprietà o un metodo). Ad esempio, nell'espressione `myObject.myProperty`, l'operatore punto indica che il termine `myProperty` si riferisce a un valore che è un elemento dell'oggetto denominato `myObject`.
- Enumerazione: serie di valori costanti correlati raggruppati per praticità come proprietà di un'unica classe.
- Ereditarietà: meccanismo della programmazione a oggetti che consente a una definizione di classe di includere tutte le funzionalità di una diversa definizione di classe (e, in generale, di aggiungere tale funzionalità).
- Istanza: oggetto vero e proprio creato in un programma.
- Spazio dei nomi: attributo personalizzato che consente un controllo più accurato su quale codice può accedere ad altro codice.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché tali esempi sono destinati principalmente alla definizione e alla manipolazione dei tipi di dati, la prova prevede la creazione di un'istanza della classe da definire, la manipolazione dell'istanza mediante le relative proprietà o i relativi metodi e infine la visualizzazione dei valori delle proprietà dell'istanza. Per visualizzare tali valori, scriverli in un'istanza di campo di testo sullo stage oppure utilizzare la funzione `trace()` per stampare i valori nel pannello Output. Queste tecniche sono descritte dettagliatamente nel capitolo "Prova degli esempi di codice contenuti nei capitoli" a pagina 68".

Classi

Una classe è una rappresentazione astratta di un oggetto. In una classe sono memorizzate informazioni sui tipi di dati che un oggetto può contenere e sui comportamenti in base ai quali un oggetto può funzionare. L'utilità di tale astrazione può non risultare evidente quando si creano brevi script contenenti solo pochi oggetti che interagiscono tra loro. Tuttavia, con l'ampliarsi dell'area di validità di un programma e con l'aumentare del numero di oggetti da gestire, le classi consentono di controllare più accuratamente la creazione degli oggetti e il modo in cui essi interagiscono.

Fino alla versione di ActionScript 1.0, i programmatori potevano utilizzare gli oggetti funzione per creare costrutti somiglianti a classi. Con ActionScript 2.0 è stato introdotto il supporto delle classi con parole chiave quali `class` ed `extends`. In ActionScript 3.0, non solo è previsto il supporto delle parole chiave introdotte in ActionScript 2.0, ma vengono introdotte anche nuove funzionalità, quale il controllo dell'accesso ottimizzato mediante gli attributi `protected` e `internal` e un migliore controllo dell'ereditarietà grazie all'uso delle parole chiave `final` e `override`.

Se si ha già esperienza nella creazione di classi con linguaggi di programmazione quali Java, C++ o C#, le procedure di ActionScript risulteranno famigliari. ActionScript utilizza infatti molte parole chiave e nomi di attributo comuni a tali linguaggi di programmazione, quali `class`, `extends` e `public`, che verranno illustrati nelle sezioni seguenti.

NOTA

In questo capitolo, il termine *proprietà* si riferisce a qualsiasi membro di una classe o di un oggetto, incluse variabili, costanti e metodi. Inoltre, anche se spesso i termini *classe* e *statica* vengono utilizzati in modo intercambiabile, in questo capitolo essi si riferiscono a concetti distinti. Ad esempio, in questo capitolo il termine *proprietà di classe* si riferisce a tutti i membri di una classe e non solo ai membri statici.

Definizioni delle classi

Le definizioni delle classi in ActionScript 3.0 impiegano una sintassi simile a quella utilizzata in ActionScript 2.0. La sintassi corretta per la definizione di una classe richiede la parola chiave `class` seguita dal nome della classe. Il corpo della classe, racchiuso tra parentesi graffe (`{}`), segue il nome della classe. Ad esempio, il seguente codice consente di creare una classe denominata Shape contenente una variabile denominata `visible`:

```
public class Shape
{
  var visible:Boolean = true;
}
```

Una modifica significativa della sintassi riguarda la definizione di classi che si trovano all'interno di un pacchetto. In ActionScript 2.0, se una classe si trovava all'interno di un pacchetto, il nome del pacchetto doveva essere incluso nella dichiarazione della classe. In ActionScript 3.0, con l'introduzione dell'istruzione `package`, il nome del pacchetto deve essere incluso nella dichiarazione del pacchetto, anziché nella dichiarazione della classe. Ad esempio, le seguenti dichiarazioni di classe mostrano come la classe BitmapData, che fa parte del pacchetto flash.display, viene definita ActionScript 2.0 e in ActionScript 3.0:

```
// ActionScript 2.0
class flash.display.BitmapData {}

// ActionScript 3.0
package flash.display
{
  public class BitmapData {}
}
```

Attributi di classe

ActionScript 3.0 consente di modificare le definizioni delle classi mediante l'impiego di uno dei seguenti attributi:

Attributo	Definizione
`dynamic`	Consente di aggiungere proprietà alle istanze in fase di runtime.
`final`	La classe non può essere estesa da un'altra classe.
`internal` (valore predefinito)	Visibile ai riferimenti che si trovano all'interno del pacchetto corrente.
`public`	Visibile a tutti i riferimenti.

In tutti questi casi, a eccezione di `internal`, è necessario includere espressamente l'attributo per ottenere il comportamento associato. Ad esempio, se durante la definizione di una classe non si include l'attributo `dynamic`, non sarà possibile aggiungere proprietà a un'istanza della classe in fase di runtime. Per assegnare esplicitamente un attributo è necessario collocarlo all'inizio della definizione della classe, come indicato nel codice seguente:

```
dynamic class Shape {}
```

Si tenga presente che nell'elenco non è incluso un attributo denominato `abstract`. Le classi astratte non sono infatti supportate in ActionScript 3.0. Si noti inoltre che l'elenco non contiene attributi denominati `private` e `protected`. Tali attributi hanno un significato unicamente all'interno di una definizione di classe e non possono essere applicati alle classi stesse. Per fare in modo che una classe non sia pubblicamente visibile al di fuori di un pacchetto, è necessario inserire la classe in un pacchetto e contrassegnarla con l'attributo `internal`. In alternativa, è possibile omettere sia l'attributo `internal` che `public`, in tal modo il compilatore inserirà automaticamente l'attributo `internal`. Se si desidera che una classe non sia visibile al di fuori del file di origine che la definisce, collocare la classe alla base del file di origine, sotto la parentesi graffa che chiude la definizione del pacchetto.

Corpo della classe

Il corpo della classe, racchiuso tra parentesi graffe, consente di definire le variabili, le costanti e i metodi della classe. Nell'esempio seguente è riportata la dichiarazione per la classe Accessibility nell'API di Adobe Flash Player:

```
public final class Accessibility
{
  public static function get active():Boolean;
  public static function updateProperties():void;
}
```

All'interno del corpo di una classe è possibile definire anche uno spazio dei nomi. Nell'esempio seguente è illustrato come uno spazio dei nomi può essere definito all'interno del corpo di una classe e utilizzato come attributo di un metodo di tale classe:

```
public class SampleClass
{
  public namespace sampleNamespace;
  sampleNamespace function doSomething():void;
}
```

ActionScript 3.0 consente di inserire nel corpo di una classe non solo definizioni, ma anche istruzioni. Le istruzioni che si trovano all'interno del corpo di una classe, ma al di fuori della definizione di un metodo, vengono eseguite una sola volta, vale a dire nel momento in cui la definizione della classe viene rilevata la prima volta e l'oggetto classe associato viene creato. L'esempio seguente include una chiamata a una funzione esterna, `hello()`, e un'istruzione `trace` che consente di produrre un messaggio di conferma quando la classe viene definita:

```
function hello():String
{
  trace("hola");
}
class SampleClass
{
  hello();
  trace("class created");
}
// output quando viene creata la classe
hola
class created
```

A differenza di quanto avveniva nelle precedenti versioni di ActionScript, in ActionScript 3.0 è consentito definire una proprietà statica e una proprietà di istanza con lo stesso nome all'interno di un medesimo corpo di classe. Ad esempio, il codice seguente indica una variabile statica denominata `message` e una variabile di istanza con lo stesso nome:

```
class StaticTest
{
  static var message:String = "static variable";
  var message:String = "instance variable";
}
// Nello script
var myST:StaticTest = new StaticTest();
trace(StaticTest.message); // output: variabile statica
trace(myST.message);       // output: variabile di istanza
```

Attributi delle proprietà di classe

In relazione al modello a oggetti di ActionScript, il termine *proprietà* si riferisce a un qualsiasi membro di un una classe, incluse variabili, costanti e metodi. Nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0* il termine viene invece utilizzato in senso più ristretto per indicare unicamente membri di classi corrispondenti a variabili o definiti mediante il metodo getter o setter. In ActionScript 3.0 è disponibile una serie di attributi utilizzabili con qualsiasi proprietà di classe. La tabella seguente riporta tali attributi.

Attributo	Definizione
`internal` (valore predefinito)	Visibile ai riferimenti che si trovano all'interno dello stesso pacchetto.
`private`	Visibile ai riferimenti che si trovano all'interno della stessa classe.
`protected`	Visibile ai riferimenti che si trovano all'interno della stessa classe e delle classi derivate.
`public`	Visibile a tutti i riferimenti.
`static`	Specifica che una proprietà appartiene alla classe e non alle sue istanze.
`UserDefinedNamespace`	Nome dello spazio dei nomi personalizzato definito dall'utente.

Attributi dello spazio dei nomi per il controllo dell'accesso

ActionScript 3.0 fornisce quattro attributi speciali per il controllo dell'accesso alle proprietà definite all'interno di una classe: `public`, `private`, `protected` e `internal`.

L'attributo `public` rende una proprietà visible in qualsiasi punto dello script. Ad esempio, per rendere un metodo disponibile al codice che si trova al di fuori del pacchetto, è necessario dichiarare il metodo con l'attributo `public`. Ciò vale per qualsiasi proprietà che sia stata dichiarata utilizzando le parole chiave `var`, `const` o `function`.

L'attributo `private` rende una proprietà visibile unicamente ai chiamanti entro la classe che definisce le proprietà. Tale comportamento si differenzia da quello dell'attributo `private` di ActionScript 2.0, che consentiva a una sottoclasse di accedere a una proprietà privata di una superclasse. Un'altra significativa modifica nel comportamento ha a che fare con l'accesso in fase di runtime. In ActionScript 2.0 la parola chiave `private` impediva l'accesso solo in fase di compilazione e poteva essere facilmente evitata in fase di runtime. In ActionScript 3.0 ciò non è più possibile. Le proprietà contrassegnate come `private` non risultano disponibili né in fase di compilazione né in fase di runtime.

Ad esempio, il codice seguente crea una semplice classe denominata PrivateExample con una variabile private, quindi tenta di accedere alla variabile private dall'esterno della classe. In ActionScript 2.0, l'accesso in fase di compilazione non era consentito, ma tale divieto veniva facilmente aggirato mediante l'impiego dell'operatore di accesso alle proprietà (`[]`), che esegue una ricerca delle proprietà in fase di runtime, anziché in fase di compilazione.

```
class PrivateExample
{
  private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);    // errore in fase di compilazione in modalità
                             // rigorosa
trace(myExample["privVar"]); // ActionScript 2.0 consente l'accesso, mentre
                             // in ActionScript 3.0 si verifica un errore
                             // di runtime.
```

In ActionScript 3.0, il tentativo di accedere a una proprietà privata mediante l'operatore punto (`myExample.privVar`) provoca un errore di compilazione, se ci si trova in modalità rigorosa. In caso contrario, l'errore viene riportato in fase di runtime, esattamente come accade quando si usa l'operatore di accesso alle proprietà (`myExample["privVar"]`).

Nella tabella seguente sono riportati i risultati di tentativi di accesso a una proprietà privata appartenente a una classe chiusa (non dinamica):

	Modalità rigorosa	**Modalità standard**
operatore punto (`.`)	errore di compilazione	errore di runtime
operatore parentesi (`[]`)	errore di runtime	errore di runtime

Nelle classi dichiarate con l'attributo `dynamic`, i tentativi di accesso a una variabile privata non provocano errori di runtime. Al contrario, la variabile risulta semplicemente invisibile, di conseguenza Flash Player restituisce il valore `undefined`. Un errore in fase di compilazione si verifica invece se si utilizza un operatore punto in modalità rigorosa. L'esempio seguente è uguale a quello precedente, con la sola differenza che la classe PrivateExample viene dichiarata come classe dinamica:

```
dynamic class PrivateExample
{
  private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);    // errore in fase di compilazione in modalità
                             // rigorosa
trace(myExample["privVar"]); // output: undefined
```

Le classi dinamiche restituiscono in genere il valore `undefined`, anziché generare un errore, se un codice esterno alla classe tenta di accedere a una proprietà privata. Nella tabella seguente si evidenzia che l'errore viene generato unicamente se viene utilizzato l'operatore punto per accedere a una proprietà privata in modalità rigorosa:

	Modalità rigorosa	**Modalità standard**
operatore punto (`.`)	errore di compilazione	`undefined`
operatore parentesi (`[]`)	`undefined`	`undefined`

L'attributo `protected`, nuovo in ActionScript 3.0, rende una proprietà visibile ai chiamanti della sua stessa classe o di una sottoclasse. In altre parole, una proprietà protetta è disponibile solo all'interno della propria classe o delle classi che si trovano sotto di essa nella gerarchia di ereditarietà. Ciò vale sia che la sottoclasse si trovi nello stesso pacchetto o in un pacchetto differente.

Per chi ha familiarità con ActionScript 2.0, questa funzionalità è simile all'attributo `private` di ActionScript 2.0. L'attributo `protected` di ActionScript 3.0 è simile anche all'attributo `protected` di Java, con la sola differenza che la versione per Java consente l'accesso anche ai chiamanti che si trovano all'interno dello stesso pacchetto. L'attributo `protected` è utile se si dispone di una variabile o di un metodo necessario alle sottoclassi, ma che si desidera tenere nascosto da codice esterno alla catena di ereditarietà.

L'attributo `internal`, nuovo in ActionScript 3.0, rende una proprietà visibile ai chiamanti che si trovano all'interno del suo stesso pacchetto. Si tratta dell'attributo predefinito del codice all'interno di un pacchetto e può essere applicato a qualsiasi proprietà che non presenta alcuno dei seguenti attributi:

- `public`
- `private`
- `protected`
- uno spazio dei nomi definito dall'utente

L'attributo `internal` è simile al controllo dell'accesso predefinito in Java, anche se in Java non esiste un nome esplicito per tale livello di accesso, che può essere ottenuto solo mediante l'omissione di un qualsiasi altro modificatore dell'accesso. L'attributo `internal` di ActionScript 3.0 consente all'utente di dimostrare esplicitamente l'intenzione di rendere una proprietà visibile unicamente ai chiamanti che si trovano all'interno del suo stesso pacchetto.

Attributo static

L'attributo `static`, utilizzabile con proprietà dichiarate con le parole chiave `var`, `const` o `function`, consente di associare una proprietà alla classe anziché a istanze della classe. Il codice esterno alla classe deve richiamare proprietà statiche utilizzando il nome della classe anziché il nome di un'istanza.

Le proprietà statiche non vengono ereditate dalle sottoclassi, ma fanno parte della catena delle aree di validità di una sottoclasse. In altre parole, all'interno del corpo di una sottoclasse, è possibile utilizzare una variabile o un metodo senza fare riferimento alla classe in cui essa o esso è stato creato. Per ulteriori informazioni, vedere "Proprietà statiche non ereditate" a pagina 181.

Attributi spazio dei nomi definiti dall'utente

In alternativa agli attributi di controllo dell'accesso predefiniti, è possibile creare uno spazio dei nomi personalizzato da utilizzare come attributo. È possibile utilizzare un solo attributo spazio dei nomi per definizione e non è possibile utilizzare un tale attributo in combinazione con un qualsiasi attributo di controllo dell'accesso (`public`, `private`, `protected`, `internal`). Per ulteriori informazioni sull'uso dello spazio dei nomi, vedere "Spazi dei nomi" a pagina 80.

Variabili

Le variabili possono essere dichiarate con le parole chiave `var` o `const`. I valori delle variabili dichiarate con la parola chiave `var` possono essere modificati più volte durante l'esecuzione di uno script. Le variabili dichiarate con la parola chiave `const` sono dette *costanti* e a esse possono essere assegnati valori una sola volta. Se si tenta di assegnare un nuovo valore a una costante inizializzata si verifica un errore. Per ulteriori informazioni, vedere "Costanti" a pagina 115.

Variabili statiche

Le variabili statiche vengono dichiarate utilizzando una combinazione della parola chiave `static` e dell'istruzione `var` o `const`. Le variabili statiche che vengono associate a una classe, anziché all'istanza di una classe, sono utili per memorizzare e condividere informazioni applicabili a un'intera classe di oggetti. Ad esempio, è appropriato l'impiego di una variabile statica per registrare quante volte viene creata un'istanza di una determinata classe oppure per memorizzare il numero massimo di istanze di una classe consentito.

Nell'esempio seguente viene creata una variabile `totalCount` per registrare il numero di volte in cui viene creata l'istanza di una classe e una costante `MAX_NUM` per memorizzare il numero massimo di istanze consentite. Le variabili `totalCount` e `MAX_NUM` sono statiche, in quanto contengono valori applicabili all'intera classe e non a una sua particolare istanza.

```
class StaticVars
{
  public static var totalCount:int = 0;
  public static const MAX_NUM:uint = 16;
}
```

Il codice esterno alla classe StaticVars e alle sue sottoclassi può fare riferimento alle proprietà `totalCount` e `MAX_NUM` solo attraverso la classe stessa. Ad esempio, il codice seguente funziona:

```
trace(StaticVars.totalCount); // output: 0
trace(StaticVars.MAX_NUM); // output: 16
```

Non è possibile accedere a variabili statiche attraverso un'istanza della classe; il codice seguente restituisce degli errori:

```
var myStaticVars:StaticVars = new StaticVars();
trace(myStaticVars.totalCount); // Errore
trace(myStaticVars.MAX_NUM); // Errore
```

Le variabili dichiarate sia con la parola chiave `static` che `const` devono essere inizializzate nel momento in cui viene dichiarata la costante, come avviene all'interno della classe StaticVars per `MAX_NUM`. Non è possibile assegnare un valore a `MAX_NUM` all'interno della funzione di costruzione o di un metodo di istanza. Il codice seguente genera un errore in quanto non è un metodo valido per inizializzare una costante statica:

```
// !! Errore; impossibile inizializzare una costante in questo modo
class StaticVars2
{
  public static const UNIQUESORT:uint;
  function initializeStatic():void
  {
    UNIQUESORT = 16;
  }
}
```

Variabili di istanza

Le variabili di istanza includono proprietà dichiarate con le parole chiave `var` e `const`, ma senza la parola chiave `static`. Le variabili di istanza associate a istanze di classe, anziché a un'intera casse, sono utili per la memorizzazione di valori specifici di una particolare istanza. Ad esempio, la classe Array presenta una proprietà di istanza denominata `length` nella quale è memorizzato il numero di elementi di array contenuti in una particolare istanza della classe Array.

Le variabili di istanza, dichiarate sia con la parola chiave `var` che `const`, non possono essere sovrascritte all'interno di una sottoclasse. È tuttavia possibile ottenere una funzionalità analoga sostituendo i metodi getter e setter. Per ulteriori informazioni, vedere "Metodi supplementari get e set" a pagina 164.

Metodi

I metodi sono funzioni che fanno parte di una definizione di classe. Una volta creata un'istanza della classe, un metodo viene associato a tale istanza. A differenza delle funzioni dichiarate all'esterno delle classi, i metodi possono essere utilizzati esclusivamente dall'istanza alla quale sono associati.

I metodi vengono definiti utilizzando la parola chiave `function`. È possibile utilizzare una funzione con istruzione, come nell'esempio seguente:

```
public function sampleFunction():String {}
```

Oppure è possibile utilizzare una variabile alla quale viene assegnata un'espressione della funzione, come nell'esempio seguente:

```
public var sampleFunction:Function = function () {}
```

Nella maggior parte dei casi, è consigliabile utilizzare una funzione con istruzione anziché un'espressione della funzione per le ragioni seguenti:

- Le funzioni con istruzione sono più concise e più facili da leggere.
- Le funzioni con istruzione consentono di utilizzare le parole chiave `override` e `final`. Per ulteriori informazioni, vedere "Sostituzione di metodi" a pagina 179.
- Le funzioni con istruzione consentono di creare un legame più solido tra l'identificatore (vale a dire, il nome della funzione) e il codice, all'interno del corpo del metodo. Poiché il valore di una variabile può essere modificato mediante un'istruzione di assegnazione, il legame tra la variabile e la relativa espressione della funzione può essere sciolto in qualsiasi momento. Nonostante sia possibile ovviare a questo problema dichiarando la variabile con la parola chiave `const` anziché `var`, tale sistema non è consigliato, in quanto rende il codice di difficile lettura e impedisce l'uso delle parole chiave `override` e `final`.

Un caso in cui è consigliabile l'uso di un'espressione della funzione è quando si decide di associare una funzione all'oggetto prototype. Per ulteriori informazioni, vedere "Oggetto prototype" a pagina 190.

Metodi delle funzioni di costruzione

I metodi delle funzioni di costruzione, talvolta semplicemente dette *funzioni di costruzione*, sono funzioni che condividono lo stesso nome della classe nella quale vengono definite. Qualsiasi codice incluso in un metodo di funzione di costruzione viene eseguito ogni volta che viene creata un'istanza della classe con la parola chiave `new`. Ad esempio, il seguente codice definisce una semplice classe chiamata Example, contenente un'unica proprietà denominata `status`. Il valore iniziale della variabile `status` viene definito all'interno della funzione di costruzione.

```
class Example
{
  public var status:String;
  public function Example()
  {
    status = "initialized";
  }
}

var myExample:Example = new Example();
trace(myExample.status); // output: inizializzato
```

I metodi delle funzioni di costruzione possono essere solo pubblici, tuttavia, l'impiego dell'attributo `public` è facoltativo. Nelle funzioni di costruzione non è possibile usare nessuno degli altri specificatori del controllo di accesso, incluso `private`, `protected` o `internal`. Con i metodi delle funzioni di costruzione non è inoltre possibile utilizzare spazi dei nomi definiti dall'utente.

Una funzione di costruzione può effettuare una chiamata esplicita alla funzione di costruzione della sua superclasse diretta mediante l'istruzione `super()`. Se la funzione di costruzione della superclasse non viene esplicitamente chiamata, il compilatore inserisce automaticamente una chiamata prima della prima istruzione nel corpo della funzione di costruzione. Per chiamare i metodi della superclasse, è inoltre possibile utilizzare il prefisso `super` come riferimento alla superclasse. Se si decide di utilizzare sia `super()` che `super` nello stesso corpo della funzione di costruzione, assicurarsi di chiamare prima `super()`. In caso contrario, il riferimento `super` non funzionerà correttamente. La funzione di costruzione `super()` deve inoltre essere chiamata prima di eventuali istruzioni `throw` o `return`.

Nell'esempio che segue è illustrato cosa accade se si tenta di utilizzare il riferimento `super` prima di chiamare la funzione di costruzione `super()`. Una nuova classe, ExampleEx, estende la classe Example. La funzione di costruzione ExampleEx tenta di accedere alla variabile status definita all'interno della sua superclasse, ma lo fa prima di chiamare `super()`. L'istruzione `trace()` che si trova all'interno della funzione di costruzione ExampleEx genera il valore `null`, in quanto la variabile `status` non risulta disponibile fino all'esecuzione della funzione di costruzione `super()`.

```
class ExampleEx extends Example
{
  public function ExampleEx()
  {
    trace(super.status);
    super();
  }
}

var mySample:ExampleEx = new ExampleEx(); // output: null
```

Nonostante sia possibile utilizzare l'istruzione `return` all'interno di una funzione di costruzione, non è consentito che venga restituito un valore. In altre parole, le istruzioni `return` non devono avere espressioni o valori associati. Di conseguenza, i metodi delle funzioni di costruzione non possono restituire valori, ovvero non è possibile specificare tipi restituiti.

Se non si definisce un metodo di funzione di costruzione nella classe, il compilatore crea automaticamente una funzione di costruzione vuota. Se la classe viene estesa a un'altra classe, il compilatore includerà una chiamata `super()` nella funzione di costruzione generata.

Metodi statici

I metodi statici, chiamati anche *metodi di classe*, sono metodi che vengono dichiarati con la parola chiave `static`. I metodi statici, generalmente associati a una classe anziché all'istanza di una classe, sono utili per incorporare funzionalità relative a qualcosa di diverso dallo stato di singole istanze. Poiché i metodi statici vengono associati a un'intera classe, è possibile accedervi solo attraverso la classe stessa e non attraverso un'istanza della classe.

I metodi statici sono utili per incorporare funzioni che non si limitano a modificare lo stato delle istanze di classe. In altre parole, un metodo si definisce statico se fornisce funzionalità che non intervengono direttamente sul valore di un'istanza di classe. Ad esempio, la classe Date presenta un metodo statico denominato `parse()` che converte una stringa in un numero. Tale metodo è statico in quanto non modifica una singola istanza della classe. Il metodo `parse()` prende invece una stringa che rappresenta un valore data, l'analizza e restituisce un numero in un formato compatibile con la rappresentazione interna dell'oggetto Date. Questo metodo non è un metodo di istanza, in quanto non avrebbe senso applicarlo a un'istanza della classe Date.

Il metodo statico `parse()` è in contrasto con i metodi di istanza della classe Date, quale `getMonth()`. Il metodo `getMonth()` è un metodo di istanza, in quanto opera direttamente sul valore recuperando un componente specifico, il mese, di un'istanza Date.

Poiché i metodi statici non sono associati a singole istanze, non è possibile utilizzare le parole chiave `this` o `super` all'interno del corpo di un metodo statico. Sia il riferimento `this` che `super` hanno significato solo nel contesto di un metodo di istanza.

Diversamente da quanto accade in altri linguaggi di programmazione basati su classi, i metodi statici in ActionScript 3.0 non vengono ereditati. Per ulteriori informazioni, vedere "Proprietà statiche non ereditate" a pagina 181.

Metodi di istanza

I metodi di istanza sono metodi dichiarati senza la parola chiave `static`. Questi metodi, che vengono associati alle istanze di una classe anziché all'intera classe, sono utili per implementare funzionalità che riguardano singole istanze di una classe. Ad esempio, la classe Array contiene un metodo di istanza denominato `sort()` che opera direttamente sulle istanze Array.

Nel corpo di un metodo di istanza, sono valide sia le variabili statiche che di istanza, il che significa che è possibile fare riferimento alle variabili definite all'interno della stessa classe mediante un semplice identificatore. Ad esempio, la classe CustomArray seguente estende la classe Array. La classe CustomArray definisce una variabile statica denominata `arrayCountTotal` che registra il numero totale di istanze della classe, una variabile di istanza denominata `arrayNumber` che registra l'ordine in cui le istanze sono state create e un metodo di istanza denominato `getPosition()` che restituisce i valori di tali variabili.

```
public class CustomArray extends Array
{
  public static var arrayCountTotal:int = 0;
  public var arrayNumber:int;

  public function CustomArray()
  {
    arrayNumber = ++arrayCountTotal;
  }

  public function getArrayPosition():String
  {
     return ("Array " + arrayNumber + " of " + arrayCountTotal);
  }
}
```

Se il codice esterno alla classe deve fare riferimento alla variabile statica `arrayCountTotal` mediante l'oggetto di classe, utilizzando `CustomArray.arrayCountTotal`, il codice che risiede nel corpo del metodo `getPosition()` può fare riferimento direttamente alla variabile statica `arrayCountTotal`. Ciò vale anche per le variabili statiche contenute nelle superclassi. Anche se le proprietà statiche non vengono ereditate in ActionScript 3.0, quelle presenti nelle superclassi sono nell'area di validità. Ad esempio, la classe Array presenta poche variabili statiche, una delle quali è una costante denominata `DESCENDING`. Il codice residente in una delle sottoclassi di Array può fare riferimento alla costante statica `DESCENDING` mediante un semplice identificatore.

```
public class CustomArray extends Array
{
  public function testStatic():void
  {
    trace(DESCENDING); // output: 2
  }
}
```

Il valore del riferimento `this` nel corpo di un metodo di istanza è un riferimento all'istanza alla quale il metodo è associato. Il codice seguente illustra come il riferimento `this` punti all'istanza contenente il metodo:

```
class ThisTest
{
  function thisValue():ThisTest
  {
    return this;
  }
}

var myTest:ThisTest = new ThisTest();
trace(myTest.thisValue() == myTest); // output: true
```

L'ereditarietà dei metodi di istanza può essere controllata con le parole chiave `override` e `final`. Utilizzare l'attributo `override` per ridefinire un metodo ereditato e l'attributo `final` per impedire alle sottoclassi di sostituire un metodo. Per ulteriori informazioni, vedere "Sostituzione di metodi" a pagina 179.

Metodi supplementari get e set

Le funzioni get e set, chiamate anche *getter* e *setter*, consentono di rispettare i principi di programmazione dell'information hiding e dell'incapsulamento, fornendo un'interfaccia di programmazione facile da usare per le classi create. Le funzioni get e set permettono di mantenere le proprietà della classe private all'interno della classe stessa, pur consentendo agli utenti della classe di accedere a tali proprietà come se stessero accedendo a una variabile di classe anziché chiamare un metodo di classe.

Il vantaggio di questo tipo di approccio è che consente di non utilizzare le funzioni accessor tradizionali con nomi poco pratici, quali `getPropertyName()` e `setPropertyName()`. Un altro vantaggio delle funzioni getter e setter è che evitano di gestire due funzioni pubbliche per ciascuna proprietà che consente l'accesso sia di lettura che di scrittura.

Nell'esempio seguente, la classe GetSet include le funzioni accessor get e set denominate `publicAccess()` che consentono l'accesso alla variabile privata denominata `privateProperty`:

```
class GetSet
{
  private var privateProperty:String;

  public function get publicAccess():String
  {
    return privateProperty;
  }
```

```
  public function set publicAccess(setValue:String):void
  {
    privateProperty = setValue;
  }
}
```

Se si tenta di accedere alla proprietà `privateProperty` direttamente, si verifica l'errore seguente:

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.privateProperty); // errore
```

Gli utenti della classe GetSet utilizzeranno invece qualcosa che appare come una proprietà denominata `publicAccess`, ma che in realtà è una coppia di funzioni get e set che operano sulla proprietà privata chiamata `privateProperty`. Nell'esempio seguente viene creata un'istanza della classe GetSet, quindi viene impostato il valore di `privateProperty` mediante l'accessor pubblico denominato `publicAccess`:

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.publicAccess); // output: null
myGetSet.publicAccess = "hello";
trace(myGetSet.publicAccess); // output: hello
```

Le funzioni getter e setter consentono inoltre di sostituire le proprietà ereditate da una superclasse, cosa non possibile se si utilizzano normali variabili membro di classe. Le variabili membro di classe dichiarate con la parola chiave `var` non possono essere sostituite in una sottoclasse. Le proprietà create con le funzioni getter e setter invece non presentano questo limite. È possibile utilizzare l'attributo `override` sulle funzioni getter e setter ereditate da una superclasse.

Metodi vincolati

Un metodo vincolato, o *chiusura di un metodo*, è semplicemente un metodo estratto dalla propria istanza. Tra gli esempi di metodi vincolati vi sono metodi passati come argomenti a una funzione o restituiti come valori da una funzione. Tra le novità di ActionScript 3.0 vi è un metodo vincolato simile a una chiusura di funzione, in quanto in grado di conservare il proprio ambiente lessicale anche se estratto dall'istanza a cui è associato. Tuttavia, la principale differenza tra un metodo vincolato e una chiusura di funzione è che il riferimento `this` del metodo vincolato resta collegato, o vincolato, all'istanza che implementa il metodo. In altre parole, il riferimento `this` in un metodo vincolato punta sempre all'oggetto originale cha ha implementato il metodo. Nelle funzioni di chiusura, il riferimento `this` è generico, vale a dire che punta a qualsiasi oggetto associato alla funzione nel momento in cui viene chiamato.

Una corretta comprensione dei metodi vincolati è importante se si utilizza la parola chiave `this`. Il richiamo della parola chiave `this` fornisce un riferimento a un oggetto principale del metodo. Per la maggior parte dei programmatori ActionScript la parola chiave `this` fa sempre riferimento all'oggetto o alla classe che contiene la definizione di un metodo. Senza i metodi vincolati, tuttavia, ciò non sarebbe sempre possibile. Nelle versioni precedenti di ActionScript, ad esempio, il riferimento `this` non era sempre riferito all'istanza che aveva implementato il metodo. Se in ActionScript 2.0 i metodi vengono estratti da un'istanza, non solo il riferimento `this` resta vincolato all'istanza originale, ma le variabili di membro e i metodi della classe dell'istanza non risultano disponibili. Il problema non esiste in ActionScript 3.0, in quanto i metodi vincolati vengono automaticamente creati quando un metodo viene passato come parametro. I metodi vincolati garantiscono che la parola chiave `this` faccia sempre riferimento all'oggetto o alla classe nella quale il metodo viene definito.

Il codice seguente definisce una classe denominata ThisTest, contenente un metodo chiamato `foo()` che definisce a sua volta il metodo vincolato e un metodo `bar()` che restituisce il metodo vincolato. Il codice esterno alla classe crea un'istanza della classe ThisTest, chiama il metodo `bar()` e memorizza il valore restituito in una variabile denominata `myFunc`.

```
class ThisTest
{
  private var num:Number = 3;
  function foo():void // metodo vincolato definito
  {
    trace("foo's this: " + this);
    trace("num: " + num);
  }
  function bar():Function
  {
    return foo; // metodo vincolato restituito
  }
}

var myTest:ThisTest = new ThisTest();
var myFunc:Function = myTest.bar();
trace(this); // output: [oggetto global]
myFunc();
/* output:
this di foo: [oggetto ThisTest]
output: num: 3 */
```

Le ultime due righe del codice mostrano che il riferimento `this` del metodo vincolato `foo()` continua a puntare a un'istanza della classe ThisTest, anche se il riferimento `this` che si trova nella linea precedente punta all'oggetto global. Inoltre, il metodo vincolato memorizzato nella variabile `myFunc` può ancora accedere alle variabili di membro della classe ThisTest. Se lo stesso codice viene eseguito in ActionScript 2.0, il riferimento `this` corrisponderebbe e la variabile `num` risulterebbe `undefined`.

Un'area in cui l'inserimento dei metodi vincolati è più evidente è quella dei gestori di eventi, in quanto il metodo `addEventListener()` richiede il passaggio di una funzione o di un metodo come argomento. Per ulteriori informazioni, vedere "Funzione listener definita come metodo di classe" a pagina 352.

Enumerazioni con classi

Le *enumerazioni* sono tipi di dati personalizzati creati per incapsulare un piccolo gruppo di valori. ActionScript 3.0 non supporta una funzionalità di enumerazione specifica, a differenza di C++, che presenta la parola chiave `enum`, o di Java, che ha un'interfaccia di enumerazione propria. È tuttavia possibile creare enumerazioni mediante le classi e le costanti statiche. Ad esempio, la classe PrintJob nell'API di Flash Player impiega un'enumerazione chiamata PrintJobOrientation per memorizzare il set di valori che comprende `"landscape"` e `"portrait"`, come illustrato nel codice seguente:

```
public final class PrintJobOrientation
{
  public static const LANDSCAPE:String = "landscape";
  public static const PORTRAIT:String = "portrait";
}
```

Per convenzione, una classe di enumerazione viene dichiarata con l'attributo `final` in quanto non vi è necessità di estendere la classe. La classe comprende solo membri statici, di conseguenza non si potranno creare istanze della classe. Al contrario, è possibile accedere ai valori di enumerazione direttamente tramite l'oggetto di classe, come illustrato nel seguente estratto di codice:

```
var pj:PrintJob = new PrintJob();
if(pj.start())
{
  if (pj.orientation == PrintJobOrientation.PORTRAIT)
  {
    ...
  }
  ...
}
```

Tutte le classi di enumerazione nell'API di Flash Player contengono solo variabili di tipo String, int o uint. Il vantaggio di utilizzare enumerazioni anziché stringhe di caratteri o valori numerici è che gli errori tipografici sono più facilmente rilevabili nelle enumerazioni. Se si digita in modo errato il nome di un'enumerazione, il compilatore di ActionScript genera un errore. Se si usano valori letterali, il compilatore non rileva gli errori ortografici o l'uso di un numero errato. Nell'esempio precedente, se il nome della costante di enumerazione non è corretto, il compilatore genera un errore, come illustrato nell'estratto seguente:

```
if (pj.orientation == PrintJobOrientation.PORTRAI) // errore del
                                                   // compilatore
```

Tuttavia, il compilatore non genera un errore se la stringa di caratteri digitata contiene un errore ortografico, come nel caso seguente:

```
if (pj.orientation == "portrai") // nessun errore del compilatore
```

Una seconda tecnica per la creazione di enumerazioni prevede anch'essa la creazione di una classe separata con proprietà statiche per l'enumerazione. Ciò che differenzia questa tecnica, tuttavia, è che le proprietà statiche contengono un'istanza della classe anziché una stringa o un valore intero. Ad esempio, il seguente codice consente di creare una classe di enumerazione per i giorni della settimana:

```
public final class Day
{
  public static const MONDAY:Day = new Day();
  public static const TUESDAY:Day = new Day();
  public static const WEDNESDAY:Day = new Day();
  public static const THURSDAY:Day = new Day();
  public static const FRIDAY:Day = new Day();
  public static const SATURDAY:Day = new Day();
  public static const SUNDAY:Day = new Day();
}
```

Questa tecnica non viene utilizzata dall'API di Flash Player, ma è impiegata da numerosi sviluppatori che ne apprezzano in particolare la funzionalità di verifica tipi ottimizzata. Ad esempio, un metodo in grado di restituire un valore di enumerazione può limitare il valore restituito al tipo di dati dell'enumerazione. Il codice seguente mostra una funzione in grado di restituire non solo un giorno della settimana, ma anche una chiamata a una funzione che impiega il tipo dell'enumerazione come annotazione di tipo:

```
function getDay():Day
{
  var date:Date = new Date();
  var retDay:Day;
  switch (date.day)
  {
    case 0:
      retDay = Day.MONDAY;
      break;
    case 1:
      retDay = Day.TUESDAY;
      break;
    case 2:
      retDay = Day.WEDNESDAY;
      break;
    case 3:
      retDay = Day.THURSDAY;
      break;
    case 4:
      retDay = Day.FRIDAY;
      break;
    case 5:
      retDay = Day.SATURDAY;
      break;
    case 6:
      retDay = Day.SUNDAY;
      break;
  }
  return retDay;
}

var dayOfWeek:Day = getDay();
```

È inoltre possibile ottimizzare la classe Day in modo che a ogni giorno della settimana venga associato un numero intero e fornire un metodo `toString()` che restituisca una rappresentazione del giorno sotto forma di stringa. È possibile migliorare in questo modo la classe Day come esercizio.

Classi delle risorse incorporate

In ActionScript 3.0 vengono impiegate classi speciali, chiamate *classi delle risorse incorporate*, per rappresentare risorse incorporate. Una *risorsa incorporata* è una risorsa, quale un suono, un'immagine o un carattere, che viene inclusa in un file SWF in fase di compilazione. L'incorporamento, anziché il caricamento dinamico, di una risorsa ne garantisce la disponibilità in fase di runtime, ma al costo di un incremento delle dimensioni del file SWF.

Uso delle classi delle risorse incorporate in Flash

Per incorporare una risorsa, in primo luogo inserirla nella libreria di un file FLA. In secondo luogo, utilizzare la finestra di dialogo Proprietà del concatenamento della risorsa per specificare il nome della classe della risorsa incorporata. Se una classe con questo nome non esiste nel percorso di classe, ne viene generata una automaticamente. A questo punto, è possibile creare un'istanza della classe della risorsa incorporata e utilizzare qualsiasi proprietà e metodo definito o ereditato dalla classe. Ad esempio, il codice seguente può essere utilizzato per riprodurre l'audio incorporato collegato alla classe di una risorsa incorporata di nome PianoMusic:

```
var piano:PianoMusic = new PianoMusic();
var sndChannel:SoundChannel = piano.play();
```

Interfacce

Un'interfaccia è una raccolta di dichiarazioni di metodi che consente a oggetti non correlati di comunicare tra loro. Ad esempio, l'API di Flash Player definisce l'interfaccia IEventDispatcher, che contiene dichiarazioni di metodi utilizzabili dalle classi per gestire gli oggetti evento. L'interfaccia IEventDispatcher stabilisce una modalità standard per il passaggio degli oggetti evento da un oggetto all'altro. Il codice seguente mostra la definizione dell'interfaccia IEventDispatcher:

```
public interface IEventDispatcher
{
  function addEventListener(type:String, listener:Function,
       useCapture:Boolean=false, priority:int=0,
       useWeakReference:Boolean = false):void;
  function removeEventListener(type:String, listener:Function,
       useCapture:Boolean=false):void;
  function dispatchEvent(event:Event):Boolean;
  function hasEventListener(type:String):Boolean;
  function willTrigger(type:String):Boolean;
}
```

Le interfacce si basano sulla distinzione tra l'interfaccia di un metodo e la sua implementazione. L'interfaccia di un metodo include tutte le informazioni necessarie per chiamare tale metodo, incluso il nome del metodo, tutti i suoi parametri e il tipo restituito. L'implementazione di un metodo include non solo le informazioni sull'interfaccia, ma anche le istruzioni eseguibili che attivano il comportamento del metodo. La definizione di un'interfaccia contiene soltanto interfacce di metodo e tutte le classi che implementano l'interfaccia sono responsabili della definizione delle implementazioni del metodo.

Nell'API di Flash Player, la classe EventDispatcher implementa l'interfaccia IEventDispatcher mediante la definizione di tutti i metodi dell'interfaccia IEventDispatcher e l'aggiunta di corpi di metodo a ognuno dei metodi. Il codice seguente è estratto dalla definizione della classe EventDispatcher:

```
public class EventDispatcher implements IEventDispatcher
{
  function dispatchEvent(event:Event):Boolean
  {
    /* istruzioni di implementazione */
  }

  ...
}
```

L'interfaccia IEventDispatcher serve da protocollo per le istanze di EventDispatcher che devono elaborare oggetti evento e passarli ad altri oggetti che hanno anch'essi implementato l'interfaccia IEventDispatcher.

Per descrivere un'interfaccia è anche possibile affermare che essa definisce un tipo di dati esattamente come fa una classe. Di conseguenza, un'interfaccia può essere usata come un'annotazione di tipo, allo stesso modo di una classe. Come tipo di dati, l'interfaccia può inoltre essere utilizzata con operatori, quali `is` e `as`, che richiedono un tipo di dati.
A differenza di quanto avviene per le classi, tuttavia, non è possibile creare istanze di un'interfaccia. Questa distinzione ha portato molti programmatori a considerare le interfacce come tipi di dati astratti e le classi come tipi di dati concreti.

Definizione di un'interfaccia

La struttura della definizione di un'interfaccia è simile a quella della definizione di una classe, a eccezione del fatto che l'interfaccia può contenere solo metodi e non corpi dei metodi. Le interfacce inoltre non possono includere variabili o costanti, ma possono incorporare funzioni getter e setter. Per definire un'interfaccia, utilizzare la parola chiave `interface`. Ad esempio, la seguente interfaccia, IExternalizable, fa parte del pacchetto flash.utils dell'API di Flash Player. L'interfaccia IExternalizable definisce un protocollo per la serializzazione di un oggetto, vale a dire la conversione di un oggetto in un formato idoneo per la memorizzazione su un dispositivo o per la trasmissione in rete.

```
public interface IExternalizable
{
  function writeExternal(output:IDataOutput):void;
  function readExternal(input:IDataInput):void;
}
```

L'interfaccia IExternalizable viene dichiarata con il modificatore del controllo di accesso `public`. Le definizioni di interfaccia possono essere modificate unicamente mediante gli specificatori del controllo di accesso `public` e `internal`. Le dichiarazioni dei metodi all'interno di una definizione di interfaccia non possono avere nessuno specificatore del controllo di accesso.

L'API di Flash Player applica una convenzione in base alla quale i nomi di interfaccia iniziano con una `I` maiuscola, ma è possibile utilizzare qualsiasi identificatore valido come nome di interfaccia. Le definizioni di interfaccia vengono spesso collocate nel livello più alto di un pacchetto. Le definizioni di interfaccia non possono essere collocate all'interno di una definizione di classe o di un'altra definizione di interfaccia.

Le interfacce possono estendere altre interfacce. Ad esempio, l'interfaccia IExample seguente estende l'interfaccia IExternalizable:

```
public interface IExample extends IExternalizable
{
  function extra():void;
}
```

Tutte le classi che implementano l'interfaccia IExample devono includere implementazioni non solo del metodo `extra()`, ma anche dei metodi `writeExternal()` e `readExternal()` ereditati dall'interfaccia IExternalizable.

Implementazione di un'interfaccia in una classe

La classe è il solo elemento del linguaggio di ActionScript 3.0 in grado di implementare un'interfaccia. Per implementare una o più interfacce utilizzare la parola chiave `implements` nella dichiarazione della classe. Negli esempi seguenti vengono definite due interfacce, IAlpha e IBeta e una classe, Alpha, che le implementa entrambe:

```
interface IAlpha
{
  function foo(str:String):String;
}

interface IBeta
{
  function bar():void;
}

class Alpha implements IAlpha, IBeta
{
  public function foo(param:String):String {}
  public function bar():void {}
}
```

In una classe che implementa un'interfaccia, i metodi implementati devono:

- Utilizzare l'identificatore del controllo di accesso `public`.
- Utilizzare lo stesso nome del metodo di interfaccia.
- Avere lo stesso numero di parametri, ciascuno con tipi di dati corrispondenti ai tipi di dati dei parametri del metodo di interfaccia.
- Utilizzare lo stesso tipo restituito.

È tuttavia possibile scegliere quale nome assegnare ai parametri dei metodi implementati. Anche se il numero dei parametri e il tipo di dati di ciascun parametro del metodo implementato devono corrispondere a quelli del metodo di interfaccia, i nomi dei parametri possono essere differenti. Nell'esempio precedente, il parametro del metodo `Alpha.foo()` è denominato `param`:

```
public function foo(param:String):String {}
```

Mentre è denominato `str` nel metodo di interfaccia `IAlpha.foo()`:

```
function foo(str:String):String;
```

È disponibile un po' di flessibilità anche per quanto riguarda i valori di parametro predefiniti. Una definizione di interfaccia può includere dichiarazioni di funzioni con valori di parametro predefiniti. Un metodo che implementa una tale dichiarazione di funzione deve avere un valore di parametro predefinito che sia membro dello stesso tipo di dati del valore specificato nella definizione di interfaccia, anche se il valore vero e proprio può essere differente.
Ad esempio, il codice seguente definisce un'interfaccia che contiene un metodo con un valore di parametro predefinito 3:

```
interface IGamma
{
  function doSomething(param:int = 3):void;
}
```

La definizione di classe seguente implementa l'interfaccia Igamma, ma impiega un valore di parametro predefinito differente:

```
class Gamma implements IGamma
{
  public function doSomething(param:int = 4):void {}
}
```

Il motivo di questa flessibilità sta nel fatto che le regole di implementazione dell'interfaccia sono state specificamente studiate per garantire la compatibilità dei tipi di dati e per raggiungere tale obiettivo non è richiesta una corrispondenza dei nomi dei parametri e dei valori predefiniti.

Ereditarietà

L'ereditarietà è una forma di riutilizzo del codice che consente ai programmatori di sviluppare nuove classi basate sulle classi esistenti. Le classi esistenti vengono spesso definite *classi base* o *superclassi*, mentre le nuove classi sono generalmente chiamate *sottoclassi*. Uno dei principali vantaggi dell'ereditarietà è che consente di riutilizzare il codice di una classe di base, senza modificare il codice esistente. Inoltre, l'ereditarietà non richiede di modificare il modo in cui le altre classi interagiscono con la classe di base. Anziché modificare una classe esistente già ampiamente testata o già in uso, l'impiego dell'ereditarietà consente di trattare tale classe come un modulo integrato da estendere con proprietà o metodi aggiuntivi. Di conseguenza, la parola chiave `extends` viene utilizzata per indicare che una classe eredita da un'altra classe.

L'ereditarietà consente inoltre di sfruttare i vantaggi del *polimorfismo* all'interno del codice. Si definisce polimorfismo la capacità di usare un unico nome metodo per un metodo in grado di comportarsi in modo differente se applicato a diversi tipi di dati. Un semplice esempio è costituito da una classe base denominata Shape con due sottoclassi denominate Circle e Square. La classe Shape definisce un metodo chiamato `area()` che restituisce l'area della figura geometrica. Se viene implementato il polimorfismo, è possibile chiamare il metodo `area()` sugli oggetti di tipo Circle e Square e ottenere i calcoli corretti. L'ereditarietà abilita il polimorfismo consentendo alle sottoclassi di ereditare e ridefinire (*sostituire*) i metodi della classe base. Nell'esempio seguente, il metodo `area()` viene ridefinito dalle classi Circle e Square:

```
class Shape
{
  public function area():Number
  {
    return NaN;
  }
}

class Circle extends Shape
{
  private var radius:Number = 1;
  override public function area():Number
  {
    return (Math.PI * (radius * radius));
  }
}

class Square extends Shape
{
  private var side:Number = 1;
  override public function area():Number
  {
    return (side * side);
  }
}

var cir:Circle = new Circle();
trace(cir.area()); // output: 3.141592653589793
var sq:Square = new Square();
trace(sq.area()); // output: 1
```

Poiché ogni classe definisce un tipo di dati, l'uso dell'ereditarietà crea una speciale relazione tra la classe base e la classe che la estende. Una sottoclasse possiede sempre tutte le proprietà della sua classe base, di conseguenza una qualsiasi istanza di una sottoclasse può sempre essere sostituita con un'istanza della classe base. Ad esempio, se un metodo definisce un parametro di tipo Shape, è consentito il passaggio di un argomento di tipo Circle, in quanto Circle è un'estensione di Shape, come indicato di seguito:

```
function draw(shapeToDraw:Shape) {}

var myCircle:Circle = new Circle();
draw(myCircle);
```

Proprietà di istanza ed ereditarietà

Una proprietà di istanza, sia che venga dichiarata con la parola chiave `function`, `var` o `const`, viene ereditata da tutte le sottoclassi a condizione che essa non sia stata dichiarata con l'attributo `private` nella classe base. Ad esempio, la classe Event nell'API di Flash Player ha un numero di sottoclassi che ereditano proprietà comuni a tutti gli oggetti evento.

Per alcuni tipi di eventi, la classe Event contiene tutte le proprietà necessarie per la definizione dell'evento. Questi tipi di eventi non richiedono proprietà di istanza oltre a quelle definite nella classe Event. Esempi di tali eventi sono l'evento `complete`, che si verifica quando i dati vengono caricati correttamente, e l'evento `connect`, che si verifica quando viene stabilita una connessione di rete.

L'esempio seguente è un estratto della classe Event che illustra alcune delle proprietà e dei metodi ereditati dalle sottoclassi. Poiché le proprietà vengono ereditate, esse sono accessibili da una qualsiasi istanza di una sottoclasse.

```
public class Event
{
  public function get type():String;
  public function get bubbles():Boolean;
  ...

  public function stopPropagation():void {}
  public function stopImmediatePropagation():void {}
  public function preventDefault():void {}
  public function isDefaultPrevented():Boolean {}
  ...
}
```

Vi sono poi altri tipi di eventi che richiedono proprietà univoche non disponibili nella classe Event. Tali eventi vengono definiti utilizzando le sottoclassi della classe Event, in modo da consentire l'aggiunta di nuove proprietà alle proprietà già definite nella classe Event. Un esempio di tali sottoclassi è la classe MouseEvent, che aggiunge proprietà univoche per gli eventi associate ai movimenti o ai clic del mouse, quali gli eventi `mouseMove` e `click`. L'esempio seguente è un estratto della classe MouseEvent che mostra la definizione di proprietà esistenti all'interno della sottoclasse, ma non nella classe base:

```
public class MouseEvent extends Event
{
  public static const CLICK:String      = "click";
  public static const MOUSE_MOVE:String = "mouseMove";
  ...

  public function get stageX():Number {}
  public function get stageY():Number {}
  ...
}
```

Specificatori del controllo di accesso ed ereditarietà

Se una proprietà viene dichiarata con la parola chiave `public`, essa risulta visibile al codice ovunque si trovi. Ciò significa che la parola chiave `public`, a differenza delle parole chiave `private`, `protected` e `internal`, non pone limiti all'ereditarietà delle proprietà.

Se una proprietà viene dichiarata con la parola chiave `private`, essa risulterà visibile solo nella classe che la definisce e non verrà ereditata da alcuna delle sottoclassi. Ciò non accadeva nelle versioni precedenti di ActionScript, dove la parola chiave `private` si comportava in modo simile alla parola chiave `protected` di ActionScript 3.0.

La parola chiave `protected` indica che una proprietà è visibile non solo all'interno della classe che la definisce, ma anche in tutte le sue sottoclassi. A differenza della parola chiave `protected` del linguaggio di programmazione Java, la parola chiave `protected` di ActionScript 3.0 non rende una proprietà visibile a tutte le altre classi dello stesso pacchetto. In ActionScript 3.0, solo le sottoclassi possono accedere a una proprietà dichiarata con la parola chiave `protected`. Inoltre, una proprietà protetta è visibile a una sottoclasse, sia che questa si trovi nello stesso pacchetto della classe base che in un pacchetto differente.

Per limitare la visibilità di una proprietà al pacchetto nel quale essa è stata definita, utilizzare la parola chiave `internal` oppure non utilizzare alcuno specificatore del controllo di accesso. Lo specificatore del controllo di accesso `internal` è lo specificatore predefinito che viene applicato quando non ne viene specificato alcuno. Se una proprietà viene contrassegnata come `internal` essa verrà ereditata unicamente dalle sottoclassi che risiedono nello stesso pacchetto.

L'esempio seguente mostra come ogni specificatore del controllo di accesso può modificare l'ereditarietà nell'ambito dei pacchetti. Il codice seguente definisce una classe di applicazione principale chiamata AccessControl e due altre classi chiamate Base ed Extender. La classe Base si trova in un pacchetto denominato foo, mentre la classe Extender, che è una sottoclasse della classe Base, si trova in un pacchetto chiamato bar. La classe AccessControl importa unicamente la classe Extender e crea un'istanza di Extender che tenta di accedere a una variabile chiamata `str` definita nella classe Base. La variabile `str` è dichiarata come `public`, di conseguenza il codice viene compilato ed eseguito come nell'estratto seguente:

```
// Base.as in una cartella chiamata foo
package foo
{
  public class Base
  {
    public var str:String = "hello"; // modifica public in questa riga
  }
}

// Extender.as in una cartella chiamata bar
package bar
{
  import foo.Base;
  public class Extender extends Base
  {
    public function getString():String {
      return str;
    }
  }
}

// classe applicazione principale in file chiamato ProtectedExample.as
  import flash.display.MovieClip;
  import bar.Extender;
  public class AccessControl extends MovieClip
  {
    public function AccessControl()
    {
      var myExt:Extender = new Extender();
      trace(myExt.testString);  // errore se str non è public
      trace(myExt.getString()); // errore se str è private o internal
    }
  }
}
```

Per vedere come gli altri specificatori del controllo di accesso possono modificare la compilazione e l'esecuzione dell'esempio precedente, modificare lo specificatore del controllo di accesso alla variabile `str` in `private`, `protected` o `internal`, dopo avere eliminato o escluso mediante commento la seguente riga dalla classe `AccessControl`:

```
trace(myExt.testString);  // errore se str non è public
```

Sostituzione delle variabili non consentita

Le proprietà dichiarate con le parole chiave `var` o `const` vengono ereditate, ma non possono essere sostituite. Sostituire una proprietà significa ridefinirla in una sottoclasse. Il solo tipo di proprietà che è possibile sostituire sono i metodi, vale a dire le proprietà dichiarate con la parola chiave `function`. Nonostante non sia possibile sostituire una variabile di istanza, si può ottenere un risultato simile mediante la creazione di metodi getter e setter per la variabile di istanza, quindi sostituire tali metodi. Per ulteriori informazioni, vedere "Sostituzione di getter e setter" a pagina 181.

Sostituzione di metodi

Sostituire un metodo significa ridefinire il comportamento di un metodo ereditato. I metodi statici non vengono ereditati e non possono essere sostituiti. I metodi di istanza, tuttavia, vengono ereditati dalle sottoclassi e possono essere sostituiti a condizione che vengano soddisfatti i seguenti criteri:

- Il metodo di istanza non deve essere dichiarato con la parola chiave `final` nella classe base. Se utilizzata con un metodo di istanza, la parola chiave `final` indica l'intenzione del programmatore di impedire alle sottoclassi di sostituire il metodo.
- Il metodo di istanza non deve essere dichiarato con lo specificatore del controllo di accesso `private` nella classe base. Se un metodo viene contrassegnato come `private` nella classe base, non è necessario utilizzare la parola chiave `override` per la definizione di un metodo con nome identico nella sottoclasse, in quanto il metodo della classe base non risulterà visibile alla sottoclasse.

Per sostituire un metodo di istanza che soddisfi i criteri di cui sopra, è necessario che nella definizione del metodo all'interno della sottoclasse venga utilizzata la parola chiave `override` e che tale definizione corrisponda alla versione del metodo della superclasse, nei modi indicati di seguito:

- Il metodo di sostituzione deve avere lo stesso livello di controllo di accesso del metodo della classe base. I metodi contrassegnati come interni hanno lo stesso livello di controllo di accesso dei metodi che non presentano alcuno specificatore del controllo di accesso.
- Il metodo di sostituzione deve avere lo stesso numero di parametri del metodo della classe base.

- I parametri del metodo di sostituzione devono avere le stesse annotazioni di tipo di dati dei parametri del metodo della classe base.
- Il metodo di sostituzione deve avere lo stesso tipo restituito del metodo della classe base.

I nomi dei parametri del metodo di sostituzione, tuttavia, non devono corrispondere ai nomi dei parametri del metodo della classe base, a condizione che il numero dei parametri e il tipo di dati di ciascun parametro corrisponda.

Istruzione super

Quando sostituiscono un metodo, i programmatori spesso intendono aggiungere qualcosa al comportamento del metodo della superclasse da sostituire, anziché rimpiazzare completamente tale comportamento. Ciò richiede un meccanismo che consenta a un metodo in una sottoclasse di richiamare la versione di se stesso presente nella superclasse. L'istruzione `super` consente tale operazione, in quanto contiene un riferimento all'immediata superclasse. Nell'esempio seguente viene definita una classe chiamata Base contenente un metodo chiamato `thanks()` e una sottoclasse della classe Base denominata Extender che sostituisce il metodo `thanks()`. Il metodo `Extender.thanks()` impiega l'istruzione `super` per chiamare `Base.thanks()`.

```
package {
  import flash.display.MovieClip;
  public class SuperExample extends MovieClip
  {
    public function SuperExample()
    {
      var myExt:Extender = new Extender()
      trace(myExt.thanks()); // output: Mahalo nui loa
    }
  }
}

class Base {
  public function thanks():String
  {
    return "Mahalo";
  }
}

class Extender extends Base
{
  override public function thanks():String
  {
    return super.thanks() + " nui loa";
  }
}
```

Sostituzione di getter e setter

Nonostante non sia possibile sostituire le variabili definite in una superclasse, è invece possibile sostituire le funzioni getter e setter. Ad esempio, il codice seguente consente di sostituire una funzione getter denominata `currentLabel` definita nella classe MovieClip dell'API di Flash Player.

```
package
{
  import flash.display.MovieClip;
  public class OverrideExample extends MovieClip
  {
    public function OverrideExample()
    {
      trace(currentLabel)
    }
    override public function get currentLabel():String
    {
      var str:String = "Override: ";
      str += super.currentLabel;
      return str;
    }
  }
}
```

Il risultato dell'istruzione `trace()` nella funzione di costruzione classe OverrideExample è `Override: null`, a indicare che nell'esempio è stato possibile sostituire la proprietà ereditata `currentLabel`.

Proprietà statiche non ereditate

Le proprietà statiche non vengono ereditate dalle sottoclassi. Ciò significa che non è possibile accedere alle proprietà statiche attraverso un'istanza di una sottoclasse. Una proprietà statica è accessibile unicamente attraverso l'oggetto di classe sul quale viene definita. Ad esempio, il codice seguente definisce una classe base chiamata Base e una sottoclasse che la estende denominata Extender. Nella classe Base viene definita una variabile statica chiamata `test`. Il codice riportato nell'estratto seguente non viene compilato in modalità rigorosa e genera un errore di runtime in modalità standard.

```
package {
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
      trace(myExt.test);  // Errore
    }
  }
}

class Base {
  public static var test:String = "static";
}

class Extender extends Base { }
```

Il solo modo per accedere alla variabile statica `test` è mediante l'oggetto di classe, come illustrato nel codice seguente:

```
Base.test;
```

È tuttavia consentito definire una proprietà di istanza con lo stesso nome della proprietà statica. Tale proprietà di istanza può essere definita nella stessa classe della proprietà statica o in una sua sottoclasse. Ad esempio, la classe Base dell'esempio precedente potrebbe avere una proprietà di istanza denominata `test`. Il codice seguente viene compilato ed eseguito perché la proprietà di istanza viene ereditata dalla classe Extender. Il codice verrebbe compilato ed eseguito anche se la definizione della variabile di istanza test venisse spostata, ma non copiata, nella classe Extender.

```
package
{
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
      trace(myExt.test);  // output: istanza
    }
  }
}

class Base
{
  public static var test:String = "static";
  public var test:String = "instance";
}

class Extender extends Base {}
```

Proprietà statiche e catena delle aree di validità

Nonostante le proprietà statiche non vengano ereditate, esse rientrano in una catena delle aree di validità della classe che definisce tali proprietà e tutte le sottoclassi della classe. Di conseguenza, le proprietà statiche vengono dette *nell'area di validità* della classe nella quale vengono definite e di tutte le sue sottoclassi. Ciò significa che una proprietà statica risulta direttamente accessibile all'interno del corpo della classe che definisce la proprietà statica e tutte le sottoclassi della classe.

Nell'esempio che segue vengono modificate le classi definite nell'esempio precedente, al fine di mostrare come la variabile statica `test` definita nella classe Base si trovi nell'area di validità della classe Extender. In altre parole, la classe Extender può accedere alla variabile statica `test` senza aggiungervi un prefisso corrispondente al nome della classe che definisce `test`.

```
package
{
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
    }
  }
}

class Base {
  public static var test:String = "static";
}

class Extender extends Base
{
  public function Extender()
  {
    trace(test); // output: static
  }

}
```

Se viene definita una proprietà di istanza che impiega lo stesso nome di una proprietà statica presente nella stessa classe o in una superclasse, la proprietà di istanza ha una precedenza maggiore nella catena delle aree di validità. La proprietà di istanza *prevarica* la proprietà statica, ovvero il valore della proprietà di istanza viene utilizzato al posto del valore della proprietà statica. Ad esempio, il codice seguente mostra che, se la classe Extender definisce una variabile di istanza denominata `test`, l'istruzione `trace()` impiega il valore della variabile di istanza anziché quello della variabile statica.

```
package
{
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
    }
  }
}

class Base
{
  public static var test:String = "static";
}

class Extender extends Base
{
  public var test:String = "instance";
  public function Extender()
  {
    trace(test); // output: istanza
  }

}
```

Argomenti avanzati

Questa sezione si apre con una breve storia di ActionScript e della programmazione a oggetti e continua con una discussione sul modello a oggetti di ActionScript 3.0 e su come esso consenta alla nuova AVM2 (ActionScript Virtual Machine) di funzionare molto più velocemente rispetto a versioni precedenti di Flash Player che contenevano AVM1.

Storia del supporto per la programmazione a oggetti in ActionScript

Poiché ActionScript 3.0 si basa sulle versioni precedenti di ActionScript, potrebbe risultare interessante comprendere come si è evoluto il modello a oggetti di ActionScript. ActionScript è nato come un semplice meccanismo per la creazione di script per le prime versioni dello strumento di creazione di Flash. Successivamente, i programmatori hanno iniziato a creare applicazioni sempre più complesse con ActionScript. In risposta alle esigenze di tali programmatori, in ogni versione successiva sono state aggiunte funzionalità di linguaggio finalizzate a facilitare la creazione di applicazioni complesse.

ActionScript 1.0

ActionScript 1.0 fa riferimento alla versione del linguaggio utilizzata in Flash Player 6 e precedenti. Già in questa prima fase della sua evoluzione, il modello a oggetti di ActionScript si basava sul concetto di oggetto come tipo di dati fondamentale. Un oggetto di ActionScript è un tipo di dati composti con un gruppo di *proprietà*. Quando si parla di modello a oggetti, il termine *proprietà* include tutto ciò che viene associato a un oggetto, ad esempio variabili, funzioni e metodi.

Nonostante la prima generazione di ActionScript non supportasse la definizione di classi con la parola chiave `class`, era possibile definire una classe impiegando un tipo di oggetto speciale denominato oggetto prototipo. Invece di utilizzare la parola chiave `class` per creare una definizione astratta di classe a partire dalla quale creare istanze in oggetti concreti, come accade nei linguaggi basati sulle classi, quali Java e C++, i linguaggi basati sui prototipi, quali ActionScript 1.0, impiegano un oggetto esistente come modello (o prototipo) per creare altri oggetti. Se nei linguaggi basati sulle classi gli oggetti puntano a una classe che funge da modello, nei linguaggi basati sui prototipi, gli oggetti puntano invece a un altro oggetto, il loro prototipo, che funge da modello.

Per creare una classe in ActionScript 1.0, è necessario definire una funzione di costruzione per tale classe. In ActionScript, le funzioni sono veri e propri oggetti, non solo definizioni astratte. La funzione di costruzione creata funge da oggetto prototipo per le istanze della classe.
Il codice seguente consente di creare una classe denominata Shape e di definire una proprietà chiamata `visible` configurata su `true` per impostazione predefinita:

```
// classe base
function Shape() {}
// Crea una proprietà denominata visible.
Shape.prototype.visible = true;
```

Questa funzione di costruzione definisce una classe Shape dalla quale è possibile creare un'istanza mediante l'operatore `new`, come segue:

```
myShape = new Shape();
```

L'oggetto funzione di costruzione `Shape()` oltre a servire da prototipo per la creazione di istanze della classe Shape, può inoltre fungere da prototipo per la creazione di sottoclassi di Shape, vale a dire di altre classi che estendono la classe Shape.

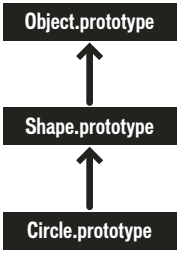
Per creare una sottoclasse della classe Shape è necessario seguire una procedura in due fasi. In primo luogo, si deve creare la classe mediante la definizione di una funzione di costruzione per la classe, come segue:

```
// classe secondaria
function Circle(id, radius)
{
    this.id = id;
    this.radius = radius;
}
```

Quindi, è necessario usare l'operatore `new` per dichiarare che la classe Shape è il prototipo della classe Circle. Per impostazione predefinita, qualsiasi classe creata impiega la classe Object come prototipo, di conseguenza, il valore `Circle.prototype` contiene un oggetto generico (un'istanza della classe Object). Per specificare che il prototipo di Circle deve essere Shape anziché Object, è necessario utilizzare il codice seguente per modificare il valore `Circle.prototype` in modo che contenga un oggetto Shape invece di un oggetto generico:

```
// Rende Circle una sottoclasse di Shape.
Circle.prototype = new Shape();
```

La classe Shape e la classe Circle sono ora collegate in una relazione di ereditarietà comunemente conosciuta come *catena di prototipi*. Il diagramma seguente illustra le varie relazioni in una catena di prototipi:



La classe alla base di ogni catena di prototipi è la classe Object. La classe Object contiene una proprietà statica denominata `Object.prototype` che punta all'oggetto prototipo di base di tutti gli oggetti creati in ActionScript 1.0. L'oggetto successivo nella catena di prototipi di esempio è l'oggetto Shape. La proprietà `Shape.prototype` infatti non è mai stata esplicitamente impostata e contiene ancora un oggetto generico (un'istanza della classe Object). Il collegamento finale della catena è costituito dalla classe Circle, collegata al suo prototipo, la classe Shape (la proprietà `Circle.prototype` contiene un oggetto Shape).

Se si crea un'istanza della classe Circle, come nell'esempio seguente, tale istanza eredita la catena di prototipi della classe Circle:

```
// Crea un'istanza della classe Circle.
myCircle = new Circle();
```

Si ricordi che è stata creata una proprietà denominata `visible`, membro della classe Shape. Nell'esempio, la proprietà `visible` non esiste come parte dell'oggetto `myCircle`, ma solo come membro dell'oggetto Shape, tuttavia, la riga di codice seguente risulta `vera`:

```
trace(myCircle.visible); // output: true
```

Flash Player è in grado di determinare che l'oggetto `myCircle` ha ereditato la proprietà `visible` risalendo la catena di prototipi. Quando esegue questo codice, Flash Player cerca in primo luogo all'interno delle proprietà dell'oggetto `myCircle` una proprietà chiamata `visible`, ma non la trova. Flash Player esegue quindi una ricerca all'interno dell'oggetto `Circle.prototype`, ma neppure lì trova una proprietà denominata `visible`. Risalendo lungo la catena dei prototipi, Flash Player trova infine la proprietà `visible` definita nell'oggetto `Shape.prototype` e restituisce il valore di tale proprietà.

Per una maggiore semplicità, in questa sezione sono stati omessi numerosi dettagli relativi alla catena di prototipi, per fornire solo le informazioni necessarie a comprendere il modello a oggetti di ActionScript 3.0.

ActionScript 2.0

In ActionScript 2.0 vengono introdotte nuove parole chiave, quali `class`, `extends`, `public` e `private`, che consentono di definire le classi in maniera simile ai linguaggi basati sulle classi, quali Java e C++. È importante tenere presente che il meccanismo di ereditarietà alla base non è cambiato con il passaggio da ActionScript 1.0 ad ActionScript 2.0. In ActionScript 2.0 è stato semplicemente aggiunto un nuovo tipo sintassi per la definizione delle classi. La catena dei prototipi funziona alla stessa maniera in entrambe le versioni del linguaggio.

La nuova sintassi introdotta da ActionScript 2.0, e illustrata nell'estratto seguente, consente di definire le classi in un modo che molti programmatori trovano più intuitivo:

```
// classe base
class Shape
{
    var visible:Boolean = true;
}
```

Si tenga inoltre presente che in ActionScript 2.0 sono state introdotte anche le annotazioni di tipo da utilizzare con la verifica del tipo in fase di compilazione, che consentono di dichiarare che la proprietà `visible` dell'esempio precedente contenga solamente un valore booleano. Anche la nuova parola chiave `extends` semplifica il processo di creazione delle sottoclassi. Nell'esempio seguente, la procedura in due fasi necessaria in ActionScript 1.0 viene portata a termine in una sola fase, grazie all'introduzione della parola chiave `extends`:

```
// classe secondaria
class Circle extends Shape
{
  var id:Number;
  var radius:Number;
  function Circle(id, radius)
  {
    this.id = id;
    this.radius = radius;
  }
}
```

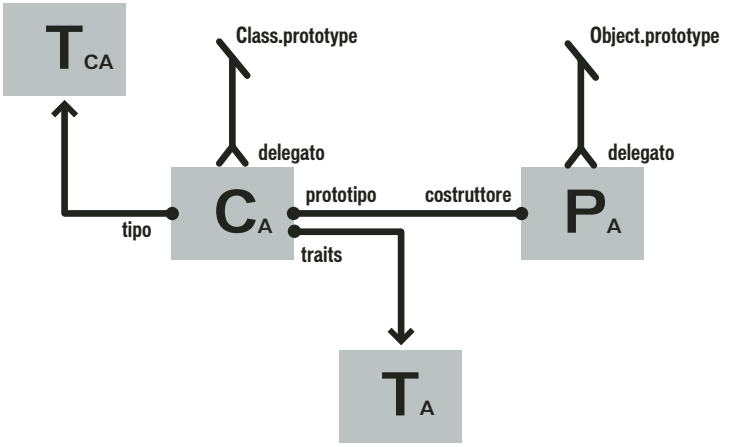
La funzione di costruzione viene ora dichiarata come parte della definizione della classe e le proprietà di classe `id` e `radius` devono anch'esse essere dichiarate esplicitamente.

In ActionScript 2.0 è stato inoltre introdotto il supporto della definizione di interfacce, che consente di rendere ancora più specifici i programmi orientati agli oggetti mediante protocolli formalmente definiti per la comunicazione tra oggetti.

Oggetto classe di ActionScript 3.0

Un comune paradigma di programmazione orientato agli oggetti, generalmente associato a Java e C++, impiega le classi per definire i tipi di oggetti. I linguaggi di programmazione che adottano questo paradigma tendono anch'essi a utilizzare le classi per costruire istanze del tipo di dati definiti dalle classi stesse. ActionScript impiega le classi per entrambi questi scopi, tuttavia, le sue origini di linguaggio basato sui prototipi gli attribuisce un'interessante caratteristica. Per ogni definizione di classe in ActionScript viene creato uno speciale oggetto di classe che consente la condivisione di comportamento e stato della classe. Per numerosi programmatori ActionScript, tuttavia, tale distinzione potrebbe non avere alcuna implicazione pratica in termini di codifica. ActionScript 3.0 è stato progettato in modo da consentire la creazione di sofisticate applicazioni ActionScript orientate agli oggetti senza la necessità di usare, o addirittura comprendere, tali particolari oggetti di classe. Per i programmatori avanzati che intendono sfruttare gli oggetti di classe, questa sezione approfondisce l'argomento.

Il diagramma seguente riporta la struttura di un oggetto di classe che rappresenta una semplice classe denominata A, definita con l'istruzione `class A {}`:



Ogni rettangolo del diagramma rappresenta un oggetto. Ogni oggetto del diagramma presenta una lettera A in carattere pedice a indicare che appartiene alla classe A. L'oggetto di classe (C_A) contiene riferimenti a vari altri oggetti importanti. L'oggetto traits dell'istanza (T_A) contiene in memoria le proprietà dell'istanza definite nella definizione della classe. L'oggetto traits della classe (T_{CA}) rappresenta il tipo interno della classe e contiene in memoria le proprietà statiche definite dalla classe (la lettera C in carattere pedice sta per "classe"). L'oggetto prototype (P_A) si riferisce sempre all'oggetto di classe al quale era originariamente associato mediante la proprietà `constructor`.

Oggetto traits

L'oggetto traits, una novità di ActionScript 3.0, è stato implementato ai fini delle prestazioni. Nelle versioni precedenti di ActionScript, il processo di ricerca dei nomi poteva risultare lento e laborioso, in quanto Flash Player doveva risalire l'intera catena di prototipi. In ActionScript 3.0, le operazioni di ricerca dei nomi sono molto più efficienti e veloci, poiché le proprietà ereditate vengono copiate dalle superclassi negli oggetti traits delle sottoclassi.

L'oggetto traits non è direttamente accessibile dal codice di programmazione, tuttavia, la sua presenza è riscontrabile in termini di miglioramenti delle prestazioni e dell'uso della memoria. L'oggetto traits fornisce ad AVM2 informazioni dettagliate sul layout e il contenuto delle classi. Grazie a tali informazioni, AVM2 è in grado di ridurre sensibilmente i tempi di esecuzione, in quanto può generare spesso istruzioni dirette alla macchina per l'accesso immediato a proprietà o il richiamo di metodi, senza che siano necessarie lente e laboriose ricerche di nomi.

L'oggetto traits consente inoltre di ridurre sensibilmente l'occupazione della memoria di un oggetto rispetto a quanto poteva occupare un oggetto simile nelle versioni precedenti di ActionScript. Ad esempio, se una classe è chiusa (vale a dire, non è dichiarata dinamica), un'istanza di tale classe non richiede una tabella hash per le proprietà aggiunte dinamicamente e può contenere qualcosa in più che un semplice puntatore agli oggetti traits e alcuni slot per le proprietà fisse definite nella classe. Di conseguenza, a un oggetto che richiedeva 100 byte di memoria in ActionScript 2.0 potrebbero bastare 20 byte in ActionScript 3.0.

NOTA	L'oggetto traits è un dettaglio interno di implementazione e potrebbe essere modificato o addirittura eliminato nelle versioni future di ActionScript.

Oggetto prototype

Ogni classe di ActionScript presenta una proprietà chiamata `prototype` che funge da riferimento all'oggetto prototype della classe. L'oggetto prototype è un retaggio delle origini di ActionScript come linguaggio basato sui prototipi. Per ulteriori informazioni, vedere "ActionScript 1.0" a pagina 185.

La proprietà `prototype` è una proprietà di sola lettura, vale a dire che non può essere modificata per puntare a oggetti differenti. Ciò la differenzia dalla proprietà di classe `prototype` delle versioni precedenti di ActionScript, dove era possibile riassegnare il prototipo in modo che puntasse a una classe diversa. Nonostante la proprietà `prototype` sia di sola lettura, l'oggetto prototype a cui fa riferimento non lo è. In altre parole, è possibile aggiungere nuove proprietà all'oggetto prototype. Le proprietà aggiunte all'oggetto prototype vengono condivise tra tutte le istanze della classe.

La catena di prototipi, che era il solo meccanismo di ereditarietà delle versioni precedenti di ActionScript, ha soltanto un ruolo secondario in ActionScript 3.0. Il sistema di ereditarietà primario, l'ereditarietà di proprietà fisse, viene gestito internamente dall'oggetto traits. Una proprietà fissa è una variabile o metodo definito in una definizione di classe. L'ereditarietà delle proprietà fisse è chiamata anche ereditarietà di classe, in quanto il meccanismo di ereditarietà è associato a parole chiave quali `class`, `extends` e `override`.

La catena di prototipi offre un meccanismo di ereditarietà alternativo molto più dinamico dell'ereditarietà di proprietà fisse. È possibile aggiungere proprietà all'oggetto prototype di una classe non solo come parte della definizione della classe, ma anche in fase di runtime mediante la proprietà `prototype` dell'oggetto di classe. Si tenga presente, tuttavia, che se si imposta il compilatore in modalità rigorosa, potrebbe essere possibile accedere a proprietà aggiunte a un oggetto prototype solo se la classe è stata dichiarata con la parola chiave `dynamic`.

Un buon esempio di classe con numerose proprietà associate all'oggetto prototype è costituito dalla classe Object. I metodi `toString()` e `valueOf()` della classe Object sono in realtà funzioni assegnate a proprietà dell'oggetto prototype della classe Object. L'esempio seguente illustra come la dichiarazione di tali metodi potrebbe, teoricamente, apparire (l'effettiva implementazione è leggermente differente a causa di dettagli di implementazione):

```
public dynamic class Object
{
  prototype.toString = function()
  {
    // istruzioni
  };
  prototype.valueOf = function()
  {
    // istruzioni
  };
}
```

Come accennato in precedenza, è possibile associare una proprietà all'oggetto prototype di una classe al di fuori della definizione della classe. Ad esempio, anche il metodo `toString()` può essere definito al di fuori della definizione della classe Object, come segue:

```
Object.prototype.toString = function()
{
  // istruzioni
};
```

A differenza dell'ereditarietà di proprietà fisse, tuttavia, l'ereditarietà di prototipi non richiede la parola chiave `override` per la ridefinizione di un metodo in una sottoclasse. Ad esempio, se si desidera ridefinire il metodo `valueOf()` in una sottoclasse della classe Object, sono disponibili tre diverse opzioni. In primo luogo, è possibile definire un metodo `valueOf()` sull'oggetto prototype della sottoclasse, all'interno della definizione della classe. Il codice seguente consente di creare una sottoclasse di Object chiamata Foo e di ridefinire il metodo `valueOf()` sull'oggetto prototype di Foo, nell'ambito della definizione della classe. Poiché ogni classe eredita da Object, non è necessario usare la parola chiave `extends`.

```
dynamic class Foo
{
  prototype.valueOf = function()
  {
    return "Instance of Foo";
  };
}
```

In secondo luogo, è possibile definire un metodo `valueOf()` sull'oggetto prototype di Foo al di fuori della definizione della classe, come illustrato nel codice seguente:

```
Foo.prototype.valueOf = function()
{
  return "Instance of Foo";
};
```

Infine, è possibile definire una proprietà fissa denominata `valueOf()` come parte della classe Foo. Questa tecnica si differenzia dalle altre in quanto mescola il sistema di ereditarietà di proprietà fisse con il sistema di ereditarietà di prototipi. Tutte le sottoclassi di Foo che desiderano ridefinire `valueOf()` devono utilizzare la parola chiave `override`. Nel codice seguente è illustrato `valueOf()` definito come proprietà fissa di Foo:

```
class Foo
{
  function valueOf():String
  {
    return "Instance of Foo";
  }
}
```

Spazio dei nomi di AS3

L'esistenza di due meccanismi di ereditarietà distinti, l'ereditarietà di prototipi e di proprietà fisse, crea un'interessante problema di compatibilità nell'ambito delle proprietà e dei metodi delle classi principali. La compatibilità con la bozza di specifica del linguaggio ECMAScript Edizione 4 prevede l'uso dell'ereditarietà di prototipi, vale a dire che le proprietà e i metodi di una classe principale devono essere definiti nell'oggetto prototype della classe. D'altro canto, la compatibilità con l'API di Flash Player richiede l'uso dell'ereditarietà di proprietà fisse, ovvero che le proprietà e i metodi di una classe principale vengano definite nell'ambito della definizione della classe mediante le parole chiave `const`, `var` e `function`. Inoltre, l'uso di proprietà fisse in luogo dei prototipi può portare a significativi incrementi delle prestazioni in fase di runtime.

ActionScript 3.0 risolve il problema utilizzando sia l'ereditarietà di prototipi che di proprietà fisse per le classi principali. Ogni classe principale contiene due serie di proprietà e metodi. Una serie viene definita sull'oggetto prototype per la compatibilità con la specifica ECMAScript, mentre l'altra serie viene definita con proprietà fisse e lo spazio dei nomi di AS3 per la compatibilità con l'API di Flash Player.

Lo spazio dei nomi di AS3 fornisce un pratico meccanismo che consente di scegliere tra le due serie di metodi e proprietà. Se non si utilizza lo spazio dei nomi di AS3, un'istanza di classe principale eredita le proprietà e i metodi definiti nell'oggetto prototype della classe principale. Se invece si decide di utilizzare lo spazio dei nomi di AS3, un'istanza di classe principale eredita le versioni di AS3, in quanto le proprietà fisse hanno sempre la precedenza rispetto alle proprietà prototype. In altre parole, se è disponibile una proprietà fissa, essa viene sempre utilizzata al posto di una proprietà prototype con nome identico.

Per scegliere di usare la versione dello spazio dei nomi di AS3 di una proprietà o di un metodo, è necessario qualificare la proprietà o il metodo con lo spazio dei nomi di AS3. Ad esempio, il codice seguente impiega la versione AS3 del metodo `Array.pop()`:

```
var nums:Array = new Array(1, 2, 3);
nums.AS3::pop();
trace(nums); // output: 1,2
```

Oppure, è possibile utilizzare la direttiva `use namespace` per aprire lo spazio dei nomi AS3 di tutte le definizioni racchiuse in un blocco di codice. Ad esempio, il codice seguente impiega la direttiva `use namespace` per aprire lo spazio dei nomi di AS3 dei metodi `pop()` e `push()`:

```
use namespace AS3;

var nums:Array = new Array(1, 2, 3);
nums.pop();
nums.push(5);
trace(nums) // output: 1,2,5
```

ActionScript 3.0 presenta inoltre opzioni del compilatore per ciascuna serie di proprietà, per consentire l'applicazione dello spazio dei nomi di AS3 all'intero programma. L'opzione del compilatore `-as3` rappresenta lo spazio dei nomi di AS3, mentre l'opzione del compilatore `-es` rappresenta l'opzione di ereditarietà dei prototipi (`es` sta per ECMAScript). Per aprire lo spazio dei nomi di AS3 per l'intero programma, impostare l'opzione del compilatore `-as3` su `true` e l'opzione del compilatore `-es` su `false`. Per utilizzare le versioni dei prototipi, impostare le opzioni del compilatore sui valori opposti. Le impostazioni predefinite del compilatore per Adobe Flex Builder 2 e Adobe Flash CS3 Professional sono `-as3 = true` e `-es = false`.

Se si ha l'intenzione di estendere una delle classi principali e di sostituire uno o più metodi, è necessario comprendere come lo spazio dei nomi di AS3 può influenzare la modalità di dichiarazione di un metodo sostituito. Se si utilizza lo spazio dei nomi di AS3, anche in ogni sostituzione di metodo di una classe principale è necessario utilizzare lo spazio dei nomi di AS3 insieme all'attributo `override`. Se non si usa lo spazio dei nomi di AS3 e si desidera ridefinire un metodo di classe principale all'interno di una sottoclasse, non è necessario utilizzare lo spazio dei nomi di AS3 né la parola chiave `override`.

Esempio: GeometricShapes

L'applicazione di esempio GeometricShapes illustra come è possibile applicare una serie di concetti e funzioni orientati agli oggetti utilizzando ActionScript 3.0, ad esempio:

- Definizione di classi
- Estensione di classi
- Polimorfismo e parola chiave `override`
- Definizione, estensione e implementazione delle interfacce

L'esempio include inoltre un "metodo factory" per la creazione di istanze di classi che illustra come dichiarare un valore restituito come un'istanza di un'interfaccia e utilizzare tale oggetto restituito in maniera generica.

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione GeometricShapes si trovano nella cartella Samples/GeometricShapes. L'applicazione è costituita dai seguenti file:

File	Descrizione
GeometricShapes.mxml o GeometricShapes.fla	Il file principale dell'applicazione in Flash (FLA) o Flex (MXML)
com/example/programmingas3/geometricshapes/IGeometricShape.as	Metodi di definizione interfaccia di base da implementare in tutte le classi dell'applicazione GeometricShapes.
com/example/programmingas3/geometricshapes/IPolygon.as	Metodo di definizione di interfaccia da implementare nelle classi dell'applicazione GeometricShapes che presentano più lati.
com/example/programmingas3/geometricshapes/RegularPolygon.as	Tipo di figura geometrica che presenta lati di uguale lunghezza posizionati simmetricamente attorno al centro della figura.
com/example/programmingas3/geometricshapes/Circle.as	Tipo di figura geometrica che definisce un cerchio.
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	Sottoclasse di RegularPolygon che definisce un triangolo con lati di pari lunghezza.
com/example/programmingas3/geometricshapes/IPolygon.as	Sottoclasse di RegularPolygon che definisce un rettangolo con quattro lati di pari lunghezza.
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	Classe contenente un metodo factory per la creazione di figure geometriche a partire da una forma e da una dimensione specificate.

Definizione delle classi GeometricShapes

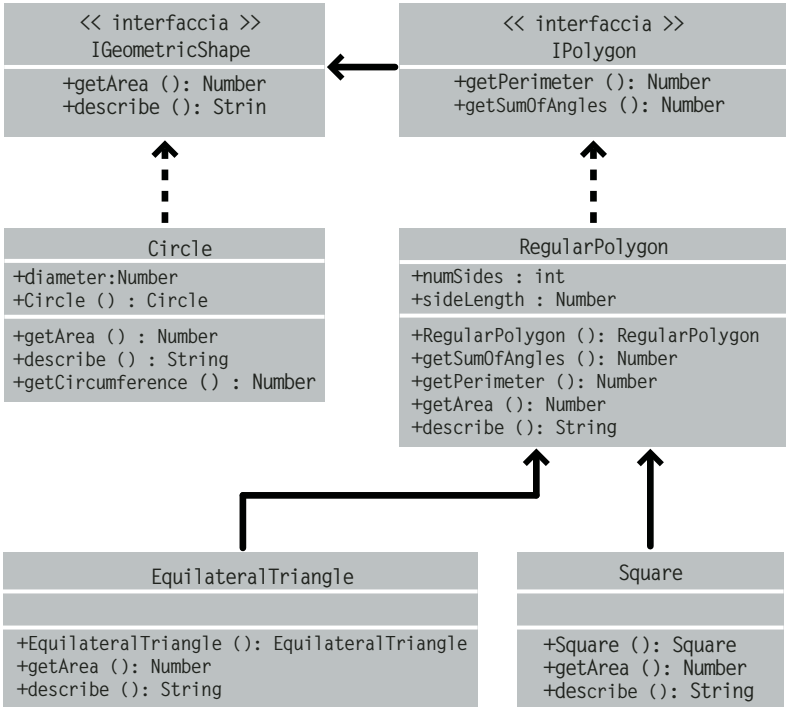
L'applicazione GeometricShapes consente di specificare un tipo di figura geometrica e una dimensione. Viene quindi restituita una descrizione della figura, la relativa area e il perimetro.

L'interfaccia utente dell'applicazione è molto semplice; essa include alcuni comandi per la selezione del tipo di figura geometrica, l'impostazione delle dimensione e la visualizzazione della descrizione. La parte più interessante di questa applicazione sta sotto la superficie, nella struttura delle classi e delle interfacce stesse.

Si tratta di un'applicazione che tratta di figure geometriche, senza visualizzarle graficamente. Viene fornita una piccola libreria di classi e interfacce che verranno utilizzate anche in un esempio di un capitolo successivo (vedere "Esempio: SpriteArranger" a pagina 454). L'esempio SpriteArranger consente di visualizzare graficamente le figure geometriche e di manipolarle, in base all'architettura delle classi fornita nell'applicazione GeometricShapes.

Le classi e le interfacce che definiscono le figure geometriche di questo esempio sono visualizzate nel diagramma seguente utilizzando il linguaggio UML (Unified Modeling Language):

Classi di esempio GeometricShape



Definizione di comportamenti comuni alle interfacce

L'applicazione GeometricShapes tratta tre diversi tipi di figure geometriche: cerchio, quadrato e triangolo equilatero. La struttura della classe GeometricShapes inizia con un'interfaccia molto semplice, IGeometricShape, che elenca una serie di metodi comuni a tutte e tre le figure geometriche:

```
package com.example.programmingas3.geometricshapes
{
  public interface IGeometricShape
  {
    function getArea():Number;
    function describe():String;
  }
}
```

L'interfaccia definisce due metodi: il metodo `getArea()`, che calcola e restituisce l'area della figura geometrica, e il metodo `describe()`, che assembla una descrizione in formato testo delle proprietà della figura.

Si desidera inoltre conoscere il perimetro di ciascuna figura. Tuttavia, il perimetro del cerchio viene definito circonferenza e viene calcolato in modo univoco, di conseguenza, in questo caso, il comportamento differisce da quello per calcolare il perimetro di un triangolo o un quadrato. Vi sono comunque sufficienti somiglianze tra triangoli, quadrati e altri poligoni da consentire la definizione di una nuova classe di interfaccia esclusiva per tali figure: IPolygon. Anche l'interfaccia IPolygon è piuttosto semplice, come illustrato di seguito:

```
package com.example.programmingas3.geometricshapes
{
  public interface IPolygon extends IGeometricShape
  {
    function getPerimeter():Number;
    function getSumOfAngles():Number;
  }
}
```

Questa interfaccia definisce due metodi comuni a tutti i poligoni: il metodo `getPerimeter()` che misura la distanza combinata di tutti i lati e il metodo `getSumOfAngles()` che somma tutti gli angoli interni.

L'interfaccia IPolygon estende l'interfaccia IGeometricShape, di conseguenza, tutte le classi che implementano l'interfaccia IPolygon devono dichiarare tutti e quattro i metodi, i due dell'interfaccia IGeometricShape e i due dell'interfaccia IPolygon.

Definizione delle classi delle figure geometriche

Una volta a conoscenza dei metodi comuni a ogni tipo di figura, è possibile passare alla definizione delle classi delle figure geometriche. In termini di quantità di metodi che è necessario implementare, la figura geometrica più semplice corrisponde alla classe Circle visualizzata di seguito:

```
package com.example.programmingas3.geometricshapes
{
  public class Circle implements IGeometricShape
  {
    public var diameter:Number;

    public function Circle(diam:Number = 100):void
    {
      this.diameter = diam;
    }

    public function getArea():Number
    {
      // The formula is Pi * radius * radius.
      var radius:Number = diameter / 2;
      return Math.PI * radius * radius;
    }

    public function getCircumference():Number
    {
      // The formula is Pi * diameter.
      return Math.PI * diameter;
    }

    public function describe():String
    {
      var desc:String = "This shape is a Circle.\n";
      desc += "Its diameter is " + diameter + " pixels.\n";
      desc += "Its area is " + getArea() + ".\n";
      desc += "Its circumference is " + getCircumference() + ".\n";
      return desc;
    }
  }
}
```

La classe Circle implementa l'interfaccia IGeometricShape, di conseguenza essa deve fornire codice sia per il metodo `getArea()` che per il metodo `describe()`. Inoltre, essa definisce il metodo `getCircumference()`, univoco per la classe Circle. Nella classe Circle viene infine dichiarata una proprietà, `diameter`, che non si trova nelle altre classi dei poligoni.

Le altre due figure geometriche, quadrato e triangolo equilatero, hanno alcune altre cose in comune: entrambe presentano lati di uguale lunghezza e vi sono formule comuni utilizzabili per calcolarne il perimetro e la somma degli angoli interni. In realtà, tali formule comuni sono applicabili a qualsiasi altro poligono regolare che sarà necessario definire in futuro.

La classe RegularPolygon sarà la superclasse sia della classe Square che della classe EquilateralTriangle. La superclasse consente di definire metodi comuni in una sola posizione, quindi non sarà necessario definirli separatamente in ogni sottoclasse. Segue il codice per la classe RegularPolygon:

```
package com.example.programmingas3.geometricshapes
{
  public class RegularPolygon implements IPolygon
  {
    public var numSides:int;
    public var sideLength:Number;

    public function RegularPolygon(len:Number = 100, sides:int = 3):void
    {
      this.sideLength = len;
      this.numSides = sides;
    }

    public function getArea():Number
    {
      // This method should be overridden in subclasses.
      return 0;
    }

    public function getPerimeter():Number
    {
      return sideLength * numSides;
    }

    public function getSumOfAngles():Number
    {
      if (numSides >= 3)
      {
        return ((numSides - 2) * 180);
      }
      else
      {
        return 0;
      }
    }
```

```
    public function describe():String
    {
      var desc:String = "Each side is " + sideLength + " pixels long.\n";
      desc += "Its area is " + getArea() + " pixels square.\n";
      desc += "Its perimeter is " + getPerimeter() + " pixels long.\n";
      desc += "The sum of all interior angles in this shape is " +
  getSumOfAngles() + " degrees.\n";
      return desc;
    }
  }
}
```

In primo luogo, la classe RegularPolygon dichiara due proprietà comuni a tutti i poligoni regolari: la lunghezza di ciascun lato (proprietà `sideLength`) e il numero di lati (proprietà `numSides`).

La classe RegularPolygon implementa l'interfaccia IPolygon e dichiara tutti e quattro i metodi dell'interfaccia IPolygon. Essa implementa inoltre due di essi (`getPerimeter()` e `getSumOfAngles()`) utilizzando formule comuni.

Poiché la formula del metodo `getArea()` è diversa in base al tipo di figura geometrica, la versione della classe base del metodo non può includere logica comune ereditabile dai metodi delle sottoclassi. Al contrario, essa restituisce semplicemente un valore 0 predefinito, a indicare che l'area non è stata calcolata. Per calcolare correttamente l'area di ogni figura geometrica, le sottoclassi della classe RegularPolygon devono sostituire il metodo `getArea()`.

Il codice seguente della classe EquilateralTriangle mostra come è possibile sostituire il metodo `getArea()`:

```
package com.example.programmingas3.geometricshapes
{
  public class EquilateralTriangle extends RegularPolygon
  {
    public function EquilateralTriangle(len:Number = 100):void
    {
      super(len, 3);
    }

    public override function getArea():Number
    {
        // The formula is ((sideLength squared) * (square root of 3)) / 4.
        return ( (this.sideLength * this.sideLength) * Math.sqrt(3) ) / 4;
    }
```

```
    public override function describe():String
    {
        /* starts with the name of the shape, then delegates the rest
        of the description work to the RegularPolygon superclass */
        var desc:String = "This shape is an equilateral Triangle.\n";
        desc += super.describe();
        return desc;
    }
  }
}
```

La parola chiave `override` indica che il metodo `EquilateralTriangle.getArea()` sostituisce intenzionalmente il metodo `getArea()` dalla superclasse RegularPolygon. Quando il metodo `EquilateralTriangle.getArea()` viene chiamato, esso calcola l'area utilizzando la formula del codice precedente, mentre il codice del metodo `RegularPolygon.getArea()` non viene mai eseguito.

Al contrario, la classe EquilateralTriangle non definisce una propria versione del metodo `getPerimeter()`. Quando il metodo `EquilateralTriangle.getPerimeter()` viene chiamato, la chiamata risale la catena di ereditarietà e viene eseguito il codice del metodo `getPerimeter()` della superclasse RegularPolygon.

La funzione di costruzione `EquilateralTriangle()` impiega l'istruzione `super()` per chiamare esplicitamente la funzione di costruzione `RegularPolygon()` della sua superclasse. Se entrambe le funzioni di costruzione presentano la stessa serie di parametri, anche se la funzione `EquilateralTriangle()` venisse omessa, la funzione di costruzione `RegularPolygon()` verrebbe eseguita al suo posto. Tuttavia, la funzione di costruzione `RegularPolygon()` richiede un parametro extra, `numSides`. Quindi, la funzione di costruzione `EquilateralTriangle()` richiama `super(len, 3)`, che trasmette il parametro di input `len` e il valore 3 per indicare che il triangolo ha tre lati.

Anche il metodo `describe()` impiega l'istruzione `super()`, ma in una maniera differente, cioè per richiamare la versione della superclasse RegularPolygon del metodo `describe()`. Il metodo `EquilateralTriangle.describe()` imposta in primo luogo la variabile di stringa `desc` in modo che indichi un'istruzione relativa al tipo di figura. Quindi, ottiene il risultato del metodo `RegularPolygon.describe()` richiamando `super.describe()` e aggiunge tale risultato alla stringa `desc`.

La classe Square non verrà descritta dettagliatamente in questa sezione, tuttavia, essa è simile alla classe EquilateralTriangle, in quanto fornisce una funzione di costruzione e modalità proprie di implementazione dei metodi `getArea()` e `describe()`.

Polimorfismo e metodo factory

Una serie di classe che impiega in modo proficuo interfacce ed ereditarietà può essere utilizzata in tanti modi interessanti. Ad esempio, tutte le classi delle figure geometriche descritte in precedenza o implementano l'interfaccia IGeometricShape o estendono una superclasse che lo fa. Di conseguenza, se si definisce una variabile come istanza di IGeometricShape, non è necessario sapere se si tratta di un'istanza della classe Circle o della classe Square per chiamare il suo metodo `describe()`.

Il codice seguente illustra come viene applicato questo principio:

```
var myShape:IGeometricShape = new Circle(100);
trace(myShape.describe());
```

Quando `myShape.describe()` viene chiamata, essa esegue il metodo `Circle.describe()` perché anche se la variabile è definita come un'istanza dell'interfaccia IGeometricShape, Circle è la sua classe sottostante.

L'esempio seguente mostra il principio del polimorfismo in azione: l'esatta chiamata di uno stesso metodo porta all'esecuzione di codici differenti, a seconda della classe dell'oggetto il cui metodo viene chiamato.

L'applicazione GeometricShapes applica questo tipo di polimorfismo basato sulle interfacce impiegando una versione semplificata di un modello di progettazione conosciuto come metodo factory. Il termine *metodo factory* si riferisce a una funzione che restituisce un oggetto il cui tipo di dati o contenuto sottostante può differire in base al contesto.

La classe GeometricShapeFactory illustrata qui definisce un metodo factory denominato `createShape()`:

```
package com.example.programmingas3.geometricshapes
{
  public class GeometricShapeFactory
  {
    public static var currentShape:IGeometricShape;

    public static function createShape(shapeName:String,
                                       len:Number):IGeometricShape
    {
      switch (shapeName)
      {
        case "Triangle":
          return new EquilateralTriangle(len);

        case "Square":
          return new Square(len);
```

```
            case "Circle":
               return new Circle(len);
         }
         return null;
      }

      public static function describeShape(shapeType:String,
   shapeSize:Number):String
      {
         GeometricShapeFactory.currentShape =
            GeometricShapeFactory.createShape(shapeType, shapeSize);
         return GeometricShapeFactory.currentShape.describe();
      }
   }
}
```

Il metodo factory `createShape()` consente alle funzioni di costruzione delle sottoclassi delle figure geometriche di definire i dettagli delle istanze da esse create, ma di restituire i nuovi oggetti come istanze di IGeometricShape, in modo che possano essere gestiti dall'applicazione in maniera più generica.

Il metodo `describeShape()` dell'esempio precedente mostra come un'applicazione può utilizzare il metodo factory per ottenere un riferimento generico a un oggetto più specifico. L'applicazione è in grado di ottenere la descrizione di un oggetto Circle appena creato come segue:

```
GeometricShapeFactory.describeShape("Circle", 100);
```

Il metodo `describeShape()` richiama quindi il metodo factory `createShape()` con gli stessi parametri, memorizzando il nuovo oggetto Circle in una variabile statica denominata `currentShape`, originariamente inserita come un oggetto di IGeometricShape. Quindi, viene chiamato il metodo `describe()` sull'oggetto `currentShape` e la chiamata viene automaticamente risolta per eseguire il metodo `Circle.describe()`, che restituisce una descrizione dettagliata del cerchio.

Ottimizzazione dell'applicazione di esempio

La reale portata di interfacce ed ereditarietà diventa evidente quando un'applicazione viene ottimizzata o modificata.

Si supponga di dover aggiungere una nuova figura geometrica, un pentagono, all'applicazione di esempio. Verrebbe creata una nuova classe Pentagon che estende la classe RegularPolygon e definisce versioni proprie dei metodi `getArea()` e `describe()`. Quindi verrebbe aggiunta una nuova opzione Pentagon alla casella combinata dell'interfaccia utente dell'applicazione. Niente altro. La classe Pentagon otterrebbe automaticamente le funzionalità del metodo `getPerimeter()` e del metodo `getSumOfAngles()` ereditandole dalla classe RegularPolygon. Poiché eredita da una classe che implementa l'interfaccia IGeometricShape, un'istanza di Pentagon può essere considerata anche un'istanza di IGeometricShape. Di conseguenza, ogni volta che si desidera aggiungere nuove figure geometriche, non sarà necessario modificare i metodi della classe GeometricShapeFactory.

È possibile aggiungere una classe Pentagon all'esempio Geometric Shapes come esercizio, per vedere come le interfacce e l'ereditarietà possono facilitare le operazioni di aggiunta di nuove funzioni a un'applicazione.

CAPITOLO 5

Operazioni con data e ora

5

L'aspetto della tempistica, seppur non primario, è generalmente un fattore importante nelle applicazioni software. ActionScript 3.0 offre una serie di interessanti funzioni per la gestione di date, orari e intervalli temporali. La maggior parte delle funzionalità legate alla gestione del tempo sono fornite da due classi principale: la classe Date e la nuova classe Timer del pacchetto flash.utils.

Sommario

Elementi fondamentali di date e orari

Introduzione all'uso di date e orari

Date e orari sono tipi di informazioni comunemente usate nei programmi ActionScript. Ad esempio, potrebbe essere necessario conoscere il giorno corrente della settimana o misurare quanto tempo trascorre un utente su una particolare schermata, tra le varie possibilità. In ActionScript, è possibile usare la classe Date per rappresentare un preciso momento nel tempo, con informazioni su data e ora. In un'istanza Date vi sono valori per singole unità di tempo e di data, quali anno, mese, giorno della settimana, ora, minuti, secondi, millisecondi e fuso orario. Per impieghi più avanzati, ActionScript include anche la classe Timer, che consente di eseguire azioni dopo un determinato intervallo di tempo o a intervalli regolari.

Operazioni comuni con date e orari

In questo capitolo vengono descritte le seguenti operazioni comuni eseguibili con le informazioni su date e orari:

- Operazioni con oggetti Date
- Determinazione del giorno e dell'orario corrente
- Accesso a singole unità di tempo e di data (giorni, anni, ore, minuti, ecc.)
- Esecuzione di operazioni aritmetiche con date e orari
- Conversione di fusi orari
- Esecuzione di azioni ripetute
- Esecuzione di azioni dopo un intervallo di tempo prestabilito

Concetti e termini importanti

Il seguente elenco di riferimento contiene termini importanti utilizzati nel presente capitolo:

- Ora universale UTC: Coordinated Universal Time, il fuso orario di riferimento o "ora zero". Tutti i fusi orari vengono espressi in numero di ore precedenti o successive l'ora universale UTC.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché tali esempi si concentrano principalmente sugli oggetti Date, la prova prevede la visualizzazione dei valori delle variabili utilizzate, sia mediante la scrittura dei valori in un'istanza di campo di testo sullo stage che mediante l'uso della funzione `trace()` per stampare i valori nel pannello Output. Queste tecniche sono descritte dettagliatamente nel capitolo "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Gestione di date e orari

Tutte le funzioni relative alla gestione di date di calendario e orari di ActionScript 3.0 sono concentrate nella classe Date di livello principale. La classe Date contiene metodi e proprietà che consentono di gestire date e orari sia nell'ora universale (UTC) che nell'ora locale specifica di un determinato fuso orario. UTC è una definizione di ora standard che corrisponde all'ora di Greenwich o GMT (Greenwich Mean Time).

Creazione di oggetti Date

La classe Date vanta uno dei metodi delle funzioni di costruzione più versatile di tutte le classi principali. È possibile chiamarlo in quattro diversi modi.

In primo luogo, se non vengono forniti parametri, la funzione di costruzione `Date()` restituisce un oggetto Date contenente la data e l'ora corrente, nell'ora locale basata sul fuso orario in cui ci si trova. Di seguito viene fornito un esempio:

```
var now:Date = new Date();
```

Secondo, se viene fornito un solo parametro numerico, la funzione di costruzione `Date()` lo interpreta come il numero di millesimi di secondo dal 1 gennaio 1970 e restituisce un oggetto Date corrispondente. Si noti che il valore in millesimi di secondo trasmesso viene interpretato come il numero di millisecondi dal 1 gennaio 1970, in UTC. Tuttavia, l'oggetto Date mostra i valori nel fuso orario locale, a meno che non si utilizzi metodi specifici di UTC per recuperarli e visualizzarli. Se si crea un nuovo oggetto Date utilizzando un singolo parametro in millesimi di secondo, accertarsi di calcolare la differenza di fuso orario tra l'ora locale e UTC. Le istruzioni seguenti consentono di creare un oggetto Date impostato sulla mezzanotte del 1 gennaio 1970, in UTC:

```
var millisecondsPerDay:int = 1000 * 60 * 60 * 24;
// ottieni un oggetto Date un giorno dopo la data di inizio 1/1/1970
var startTime:Date = new Date(millisecondsPerDay);
```

Terzo, è possibile trasmettere più parametri numerici alla funzione di costruzione `Date()`. Tali parametri vengono trattati rispettivamente come anno, mese, giorno, ora, minuto, secondo e millesimo di secondo e viene restituito un oggetto Date corrispondente. I seguenti parametri di input sono espressi in ora locale anziché in UTC. Le istruzioni seguenti consentono di creare un oggetto Date impostato sulla mezzanotte del 1 gennaio 2000, in UTC:

```
var millenium:Date = new Date(2000, 0, 1, 0, 0, 0, 0);
```

Infine, è possibile trasmettere un'unica stringa di parametri alla funzione di costruzione `Date()`. La funzione tenterà di analizzare la stringa in componenti di data o ora, quindi di restituire un oggetto Date corrispondente. Se si sceglie questo sistema, si consiglia di includere la funzione di costruzione `Date()` in un blocco `try..catch` per registrare eventuali errori di analisi. La funzione di costruzione `Date()` accetta vari formati di stringa, come illustrato in *Guida di riferimento del linguaggio e ai componenti ActionScript 3.0*. L'istruzione seguente consente di inizializzare un nuovo oggetto Date mediante un valore di stringa:

```
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");
```

Se la funzione di costruzione `Date()` non è in grado di analizzare il parametro di stringa, non verrà generata alcuna eccezione. Tuttavia, l'oggetto Date risultante conterrà un valore data non valido.

Determinazione dei valori di unità di tempo

È possibile estrarre i valori di varie unità di tempo da un oggetto Date mediante proprietà e metodi della classe Date. Ognuna delle seguenti proprietà consente di ottenere il valore di un'unità di tempo nell'oggetto Date:

- Proprietà `fullYear`
- Proprietà `month`, espressa in formato numerico (da 0 per gennaio a 11 per dicembre)
- Proprietà `date`, corrispondente al numero di calendario del giorno del mese, nell'intervallo da 1 a 31
- Proprietà `day`, giorno della settimana in formato numerico (0 per domenica)
- Proprietà `hours`, da 0 a 23
- Proprietà `minutes`
- Proprietà `seconds`
- Proprietà `milliseconds`

La classe Date offre vari metodi per determinare ciascuno di questi valori. Per ottenere il valore mese di un oggetto Date, ad esempio, sono disponibili quattro diversi modi:

- La proprietà `month`
- Il metodo `getMonth()`
- La proprietà `monthUTC`
- Il metodo `getMonthUTC()`

Tutte e quattro le modalità si equivalgono in termini di efficacia, quindi è possibile scegliere il metodo preferenziale in base ai requisiti dell'applicazione.

Le proprietà elencate rappresentano tutte componenti del valore data globale. Ad esempio, la proprietà dei millesimi di secondo non sarà mai superiore a 999, poiché, quando raggiunge il valore 1000, la proprietà secondi aumenta di 1 e la proprietà millesimi di secondo riparte dal valore 0.

Per ottenere il valore dell'oggetto Date in termini di millesimi di secondo dal 1 gennaio 1970 (UTC), è possibile utilizzare il metodo `getTime()`. La sua controparte, il metodo `setTime()`, consente di modificare il valore di un oggetto Date esistente utilizzando i millesimi di secondo dal 1 gennaio 1970 (UTC).

Operazioni aritmetiche con data e ora

La classe Date consente di eseguire addizioni e sottrazioni con i valori di data e ora. I valori di data sono memorizzati internamente in millesimi di secondo, di conseguenza, prima di aggiungere o sottrarre un valore da oggetti Date è necessario convertirlo in millesimi di secondo.

Se l'applicazione deve eseguire molte operazioni aritmetiche con date e orari, potrebbe essere utile creare costanti per conservare i valori di unità di tempo maggiormente utilizzati in millesimi di secondo, come nell'esempio seguente:

```
public static const millisecondsPerMinute:int = 1000 * 60;
public static const millisecondsPerHour:int = 1000 * 60 * 60;
public static const millisecondsPerDay:int = 1000 * 60 * 60 * 24;
```

Grazie alle unità di tempo standard, sarà più semplice eseguire operazioni aritmetiche con le date. Il codice seguente consente di impostare un valore di data su un'ora a partire dall'ora corrente mediante i metodi `getTime()` e `setTime()`:

```
var oneHourFromNow:Date = new Date();
oneHourFromNow.setTime(oneHourFromNow.getTime() + millisecondsPerHour);
```

Un altro modo per impostare un valore di data è creare un nuovo oggetto Date utilizzando un unico parametro di millesimi di secondo. Ad esempio, il codice seguente consente di aggiungere 30 giorni a una data per calcolarne un'altra:

```
// Imposta la data fattura sulla data corrente
var invoiceDate:Date = new Date();

// Aggiunge 30 giorni per ottenere la data di scadenza
var dueDate:Date = new Date(invoiceDate.getTime() + (30 *
  millisecondsPerDay));
```

Successivamente, la costante `millisecondsPerDay` viene moltiplicata per 30 per rappresentare l'intervallo di 30 giorni e il risultato viene sommato al valore `invoiceDate` utilizzato per impostare il valore `dueDate`.

Conversione di fusi orari

Le operazioni aritmetiche con data e ora risultano utili per convertire date da un fuso orario a un altro. Così funziona il metodo `getTimezoneOffset()`, che restituisce il valore in minuti corrispondente allo scarto del fuso orario dell'oggetto Date rispetto a UTC. Viene restituito un valore in minuti perché non tutti i fusi orari sono impostati in incrementi orari equivalenti, alcuni infatti presentano differenze di mezz'ora rispetto ai fusi orari adiacenti.

Nell'esempio seguente viene utilizzata la differenza di fuso orario per convertire una data dall'ora locale all'ora UTC. La conversione viene effettuata mediante il calcolo del valore del fuso orario in millesimi di secondo, quindi effettuando una regolazione del valore Date con tale valore:

```
// Crea un oggetto Date nell'ora locale
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");

// Converte Date in UTC aggiungendo o sottraendo la differenza di
// fuso orario
var offsetMilliseconds:Number = nextDay.getTimezoneOffset() * 60 * 1000;
nextDay.setTime(nextDay.getTime() + offsetMilliseconds);
```

Controllo degli intervalli di tempo

Quando si sviluppano applicazioni mediante Adobe Flash CS3 Professional, è possibile accedere alla linea temporale, che fornisce una progressione fotogramma per fotogramma dell'intera applicazione. In progetti esclusivi di ActionScript, tuttavia, è necessario affidarsi ad altri meccanismi di gestione temporale.

Cicli e timer

In alcuni linguaggi di programmazione, è necessario applicare schemi di gestione del tempo propri mediante istruzioni di cicliche quali `for` o `do..while`.

Le istruzioni cicliche vengono generalmente eseguite alla massima velocità consentita dal sistema, di conseguenza, l'applicazione risulterà più veloce su alcune macchine rispetto ad altre. Se l'applicazione che si sta eseguendo necessita di un intervallo di tempo regolare, è necessario collegarlo a un calendario o a un orologio. Molte applicazioni, quali giochi, animazioni e controller in tempo reale, richiedono meccanismi di scansione temporale regolari e omogenei tra un computer e l'altro.

La classe Timer di ActionScript 3.0 offre una soluzione efficace al problema. Grazie al modello eventi di ActionScript 3.0, la classe Timer è in grado di inviare eventi timer ogni volta che viene raggiunto un intervallo di tempo specificato.

Classe Timer

La modalità preferenziale di gestione delle funzioni temporali in ActionScript 3.0 è mediante la classe Timer (flash.utils.Timer), che consente di inviare eventi al trascorrere di intervalli di tempo specificati.

Per avviare un timer, è necessario creare in primo luogo un'istanza della classe Timer, quindi impostare la frequenza di creazione di un evento timer e la quantità di eventi da generare prima di arrestarsi.

Ad esempio, il codice seguente consente di creare un'istanza Timer che invia un evento ogni secondo e continua per 60 secondi:

```
var oneMinuteTimer:Timer = new Timer(1000, 60);
```

L'oggetto Timer invia un oggetto TimerEvent ogni volta che viene raggiunto l'intervallo di tempo specificato. Un tipo di evento dell'oggetto TimerEvent è `timer` (definito dalla costante `TimerEvent.TIMER`). Un oggetto TimerEvent contiene le stesse proprietà dell'oggetto standard Event.

Se l'istanza Timer viene impostata per un determinato numero di intervalli, essa invierà anche un evento `timerComplete` (definito dalla costante `TimerEvent.TIMER_COMPLETE`) quando viene raggiunto l'intervallo finale.

Segue una breve applicazione di esempio che illustra il funzionamento della classe Timer:

```
package
{
  import flash.display.Sprite;
  import flash.events.TimerEvent;
  import flash.utils.Timer;

  public class ShortTimer extends Sprite
  {
    public function ShortTimer()
    {
      // Crea un nuovo Timer di cinque secondi
      var minuteTimer:Timer = new Timer(1000, 5);

      // Designa listener per gli eventi intervallo e di completamento
      minuteTimer.addEventListener(TimerEvent.TIMER, onTick);
      minuteTimer.addEventListener(TimerEvent.TIMER_COMPLETE,
  onTimerComplete);

      // Avvia il conteggio del timer
      minuteTimer.start();
    }
```

```
    public function onTick(event:TimerEvent):void
    {
      // Visualizza il conteggio
      // L'obbiettivo di questo evento è l'istanza Timer stessa.
      trace("tick " + event.target.currentCount);
    }

    public function onTimerComplete(event:TimerEvent):void
    {
      trace("Time's Up!");
    }
  }
}
```

Quando viene creata la classe ShortTimer, viene creata anche un'istanza Timer che esegue un controllo una volta al secondo per cinque secondi. Quindi vengono aggiunti due listener al timer: uno che intercetta ogni conteggio di controllo e l'altro che resta in attesa dell'evento `timerComplete`.

Quindi, viene avviato il conteggio del timer e, da quel momento in poi, il metodo `onTick()` viene eseguito a intervalli di un secondo.

Il metodo `onTick()` visualizza semplicemente il conteggio temporale corrente. Dopo cinque secondi, viene eseguito il metodo `onTimerComplete()`, che avverte che il tempo è scaduto.

Quando si esegue questo esempio, sulla console o sulla finestra di traccia dovrebbero apparire le seguenti righe alla velocità di una riga al secondo:

```
tick 1
tick 2
tick 3
tick 4
tick 5
Time's Up!
```

Funzioni temporali del pacchetto flash.utils

ActionScript 3.0 contiene varie funzioni per la gestione del tempo simili a quelle disponibili in ActionScript 2.0. Tali funzioni vengono fornite a livello di pacchetto in flash.utils e possono essere utilizzate esattamente come in ActionScript 2.0.

Funzione	Descrizione
`clearInterval(id:uint):void`	Annulla una chiamata `setInterval()` specifica.
`clearTimeout(id:uint):void`	Annulla una chiamata `setTimeout()` specifica.

Funzione	Descrizione
`getTimer():int`	Restituisce il numero di millesimi di secondo trascorsi dall'inizializzazione di Adobe Flash Player.
`setInterval(closure:Function, delay:Number, ... arguments):uint`	Esegue una funzione a un intervallo specificato (in millisecondi).
`setTimeout(closure:Function, delay:Number, ... arguments):uint`	Esegue una funzione specifica dopo un ritardo prestabilito (in millisecondi).

Queste funzioni sono incluse in ActionScript 3.0 al fine di assicurare la compatibilità con le versioni precedenti. Adobe sconsiglia di utilizzare nelle nuove applicazioni di ActionScript 3.0. In generale, l'uso della classe Timer nelle applicazioni risulta più facile e più efficace.

Esempio: Orologio analogico semplice

L'esempio di un orologio analogico semplice illustra due dei concetti relativi a data e ora discussi in questo capitolo:

- Determinazione della data e dell'ora correnti ed estrazione dei valori di ore, minuti e secondi
- Uso di un timer per impostare la velocità di un'applicazione

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione SimpleClock si trovano nella cartella Samples/SimpleClock. L'applicazione è costituita dai seguenti file:

File	Descrizione
SimpleClockApp.mxml o SimpleClockApp.fla	Il file principale dell'applicazione in Flash (FLA) o Flex (MXML)
com/example/programmingas3/simpleclock/SimpleClock.as	File dell'applicazione principale.
com/example/programmingas3/simpleclock/AnalogClockFace.as	Consente di disegnare il quadrante di un orologio con lancette per ore, minuti e secondi.

Definizione della classe SimpleClock

L'esempio dell'orologio è molto semplice, tuttavia risulta molto efficace organizzare bene anche applicazioni banali, in modo da poterle espandere senza difficoltà in futuro. A tale scopo, l'applicazione SimpleClock impiega la classe SimpleClock per gestire attività di avvio e gestione del tempo e un'altra classe chiamata AnalogClockFace per visualizzare l'ora.

Il codice seguente consente di definire e inizializzare la classe SimpleClock (si noti che, nella versione Flash, SimpleClock estende invece la classe Sprite):

```
public class SimpleClock extends UIComponent
{
  /**
   * Componente per la visualizzazione dell'ora.
   */
  private var face:AnalogClockFace;

  /**
   * Timer che scandisce il tempo dell'applicazione.
   */
  private var ticker:Timer;
```

La classe presenta due importanti proprietà:

- La proprietà `face`, che è un'istanza della classe AnalogClockFace
- La proprietà `ticker`, che è un'istanza della classe Timer

La classe SimpleClock impiega una funzione di costruzione predefinita. Il metodo `initClock()` si occupa del lavoro di impostazione vero e proprio, in quanto crea il quadrante dell'orologio e avvia il conteggio dell'istanza Timer.

Creazione del quadrante dell'orologio

Nelle righe di codice SimpleClock seguenti viene creato il quadrante dell'orologio utilizzato per visualizzare l'ora:

```
  /**
   * Imposta un'istanza SimpleClock.
   */
  public function initClock(faceSize:Number = 200)
  {
    // Crea il quadrante dell'orologio e lo aggiunge all'elenco
    // di visualizzazione
    face = new AnalogClockFace(Math.max(20, faceSize));
    face.init();
    addChild(face);

    // Disegna la visualizzazione iniziale dell'orologio
    face.draw();
```

Le dimensioni del quadrante possono essere trasmesse al metodo `initClock()`. Se non viene trasmesso alcun valore `faceSize`, viene utilizzata la dimensione predefinita di 200 pixel.

Quindi, l'applicazione inizializza il quadrante e lo aggiunge all'elenco di visualizzazione utilizzando il metodo `addChild()` ereditato dalla classe DisplayObject. Successivamente, viene chiamato il metodo `AnalogClockFace.draw()` per visualizzare una volta il quadrante e l'ora corrente.

Avvio del timer

Dopo aver creato il quadrante dell'orologio, il metodo `initClock()` imposta un timer:

```
// Crea un Timer che attiva un evento al secondo
ticker = new Timer(1000);

// Designa il metodo onTick() per la gestione degli eventi Timer
ticker.addEventListener(TimerEvent.TIMER, onTick);

// Avvia il conteggio dell'orologio
ticker.start();
```

Per prima cosa viene creata un'istanza di Timer che invia un evento una volta al secondo (ogni 1000 millesimi di secondo). Poiché alla funzione di costruzione `Timer()` non viene trasmesso nessun secondo parametro `repeatCount`, il Timer continuerà a ripetere l'operazione all'infinito.

Il metodo `SimpleClock.onTick()` verrà eseguito una volta al secondo quando viene ricevuto l'evento `timer`:

```
public function onTick(event:TimerEvent):void
{
  // Aggiorna la visualizzazione dell'orologio
  face.draw();
}
```

Il metodo `AnalogClockFace.draw()` disegna semplicemente il quadrante e le lancette dell'orologio.

Visualizzazione dell'ora corrente

La maggior parte del codice della classe AnalogClockFace consente di impostare gli elementi visivi dell'orologio. Quando il metodo AnalogClockFace viene inizializzato, viene disegnata una forma circolare con un'etichetta di testo numerica in corrispondenza di ogni ora, quindi vengono creati tre oggetti Shape, uno per ogni lancetta, quella delle ore, dei minuti e dei secondi.

Una volta eseguita, l'applicazione SimpleClock chiama il metodo `AnalogClockFace.draw()` ogni secondo, come indicato di seguito:

```
/**
 * Chiamato dal contenitore principale quando il quadrante viene
disegnato.
 */
public override function draw():void
{
  // Memorizza la data e l'ora corrente in una variabile di istanza
  currentTime = new Date();
  showTime(currentTime);
}
```

Questo metodo consente di salvare l'ora corrente in una variabile, di modo che l'ora non venga modificata nella fase di disegno delle lancette. Quindi viene chiamato il metodo `showTime()` per visualizzare le lancette, come riportato di seguito:

```
/**
 * Visualizza data e ora nell'orologio analogico.
 */
public function showTime(time:Date):void
{
  // Determina i valori di ora
  var seconds:uint = time.getSeconds();
  var minutes:uint = time.getMinutes();
  var hours:uint = time.getHours();

  // Moltiplica per 6 per determinare i gradi
  this.secondHand.rotation = 180 + (seconds * 6);
  this.minuteHand.rotation = 180 + (minutes * 6);

  // Moltiplica per 30 per determinare i gradi base, quindi
  // aggiungi 29,5 gradi (59 * 0,5)
  // per calcolare i minuti.
  this.hourHand.rotation = 180 + (hours * 30) + (minutes * 0.5);
}
```

In primo luogo, questo metodo consente di estrarre i valori di ore, minuti e secondi dell'ora corrente. Quindi impiega questi valori per calcolare l'angolo di ogni lancetta. Poiché la lancetta dei secondi effettua una rotazione completa in 60 secondi, essa ruota di 6 gradi al secondo (360/60). La lancetta dei minuti effettua la stessa rotazione ogni minuto.

Anche la lancetta delle ore viene aggiornata ogni minuto, in modo che avanzi di un grado con trascorrere dei minuti. Essa ruota di 30 gradi ogni ora (360/12), ma anche di mezzo grado ogni minuto (30 gradi diviso 60 minuti).

CAPITOLO 6

Operazioni con le stringhe

6

La classe String contiene i metodi che consentono di utilizzare le stringhe di testo. Le stringhe sono importanti per eseguire operazioni in numerosi oggetti. I metodi descritti in questo capitolo sono utili per lavorare con le stringhe in oggetti quali TextField, StaticText, XML, ContextMenu e FileReference.

Le stringe sono sequenze di caratteri. ActionScript 3.0 supporta caratteri ASCII e Unicode.

Sommario

Elementi fondamentali delle stringhe

Introduzione alle operazioni con le stringhe

In termini di programmazione, una stringa è un valore di testo, vale a dire una sequenza di lettere, numeri o altri caratteri legati tra loro per formare un singolo valore. Ad esempio, la seguente riga di codice consente di creare una variabile con tipo di dati String e di assegnare un valore letterale di stringa a tale variabile:

```
var albumName:String = "Three for the money";
```

Come illustrato da questo esempio, in ActionScript è possibile indicare un valore di stringa racchiudendo il testo tra virgolette semplici o doppie. Seguono altri esempi di stringhe:

```
"Hello"
"555-7649"
"http://www.adobe.com/"
```

Ogni volta che si manipola una porzione di testo in ActionScript, si lavora con un valore di stringa. La classe String di ActionScript è il tipo di dati da utilizzare per eseguire operazioni con i valori di testo. Le istanze String vengono spesso impiegate per proprietà, metodi, parametri e così via, in molte altre classi di ActionScript.

Operazioni comuni con le stringhe

Nel presente capitolo verranno illustrate le seguenti attività comuni relative alle stringhe:

- Creazione di oggetti String
- Uso di caratteri speciali quali ritorno a capo, tabulazione e caratteri non di tastiera
- Misurazione della lunghezza stringa
- Isolamento di singoli caratteri in una stringa
- Unione di stringhe
- Confronto tra stringhe
- Ricerca, estrazione e sostituzione di porzioni di stringhe
- Generazione di stringhe maiuscole o minuscole

Concetti e termini importanti

L'elenco di riferimento seguente contiene dei termini importanti che vengono citati in questo capitolo:

- ASCII: sistema per la rappresentazione dei caratteri di testo e dei simboli nei programmi per computer. Il sistema ASCII supporta l'alfabeto inglese di 26 lettere, più un numero limitato di caratteri aggiuntivi.
- Carattere: la più piccola unita di dati di testo (un singolo carattere o simbolo).
- Concatenazione: unione di più valori di stringa mediante l'aggiunta di un valore alla fine di un altro, per creare un nuovo valore di stringa.
- Stringa vuota: stringa non contenente testo, spazi vuoto o altri caratteri, scritti come `""`. Un valore di stringa vuota è diverso da una variabile String con valore null. Una variabile String con valore null è una variabile alla quale non è stata assegnata alcuna istanza String, mentre una stringa vuota presenta un'istanza con un valore che non contiene caratteri.
- String: valore testuale (sequenza di caratteri).

- Valore letterale di stringa: valore di stringa esplicitamente scritto in codice come un valore di testo racchiuso da virgolette semplici o doppie.
- Sottostringa: stringa corrispondente a una porzione di un'altra stringa.
- Unicode: sistema standard per la rappresentazione dei caratteri di testo e dei simboli nei programmi per computer. Il sistema Unicode consente l'uso di qualsiasi carattere in qualsiasi sistema di scrittura.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché tali esempi si concentrano principalmente sulla manipolazione del testo, la prova prevede la visualizzazione dei valori delle variabili utilizzate, sia mediante la scrittura dei valori in un'istanza di campo di testo sullo stage che mediante l'uso della funzione `trace()` per stampare i valori nel pannello Output. Queste tecniche sono descritte dettagliatamente nel capitolo "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Creazione di stringhe

La classe String viene utilizzata per rappresentare dati di stringa (testuali) in ActionScript 3.0. Le stringhe di ActionScript 3.0 supportano sia i caratteri ASCII che Unicode. Il modo più semplice per creare una stringa è di usare un carattere letterale. Per dichiarare un valore letterale di stringa, utilizzare le virgolette doppie diritte (`"`) o le virgolette semplici (`'`). Ad esempio, le due stringhe seguenti sono equivalenti:

```
var str1:String = "hello";
var str2:String = 'hello';
```

Per dichiarare una stringa è inoltre possibile utilizzare l'operatore `new`, come di seguito:

```
var str1:String = new String("hello");
var str2:String = new String(str1);
var str3:String = new String();       // str3 == ""
```

Le due stringhe seguenti sono equivalenti:

```
var str1:String = "hello";
var str2:String = new String("hello");
```

Per utilizzare le virgolette semplici (`'`) all'interno di una stringa delimitata da virgolette semplici (`'`), utilizzare il carattere barra rovesciata (`\`). Analogamente, per utilizzare le virgolette doppie (`"`) all'interno di una stringa delimitata da virgolette doppie (`"`), utilizzare il carattere barra rovesciata (`\`). Le due stringhe seguenti sono equivalenti:

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

Scegliere di utilizzare le virgolette semplici o doppie in base quanto già presente nel valore letterale di stringa, come illustrato nell'esempio seguente:

```
var str1:String = "ActionScript <span class='heavy'>3.0</span>";
var str2:String = '<item id="155">banana</item>';
```

Tenere presente che ActionScript fa distinzione tra virgolette singole dritte (') e virgolette singole curve (‘ o ’). Lo stesso vale per le virgolette doppie. Per contraddistinguere i valori letterali di stringa utilizzare sempre le virgolette dritte. Se si incolla testo da una fonte esterna in ActionScript, verificare di utilizzare i caratteri corretti.

Come illustrato nella tabella seguente, per definire altri caratteri nei valori letterali di stringa, è possibile utilizzare il carattere barra rovesciata (\).

Sequenza di escape	Carattere
\b	Backspace
\f	Avanzamento pagina
\n	Nuova riga
\r	Ritorno a capo
\t	Tabulazione
\u*nnnn*	Carattere Unicode con il codice carattere specificato dal numero esadecimale *nnnn*; ad esempio, \u263a è il carattere smiley.
\x*nn*	Carattere ASCII con il codice di carattere specificato mediante il numero esadecimale *nn*.
\'	Virgolette semplici
\"	Virgolette doppie
\\	Barra rovesciata singola

Proprietà length

Ogni stringa dispone di una proprietà `length` che corrisponde al numero di caratteri contenuti nella stringa:

```
var str:String = "Adobe";
trace(str.length);            // output: 5
```

Se una stringa è nulla o vuota la sua lunghezza corrisponde a 0, come nell'esempio seguente:

```
var str1:String = new String();
trace(str1.length);           // output: 0

str2:String = '';
trace(str2.length);           // output: 0
```

Operazioni con i caratteri nelle stringhe

A ogni carattere di una stringa corrisponde una posizione di indice nella stringa (un numero intero). La posizione di indice del primo carattere è 0. Ad esempio, nella stringa seguente il carattere `y` si trova nella posizione 0 e il carattere `w` nella posizione 5:

```
"yellow"
```

È possibile esaminare singoli caratteri in varie posizioni di una stringa mediante il metodo `charAt()` e il metodo `charCodeAt()`, come nell'esempio seguente:

```
var str:String = "hello world!";
for (var:i = 0; i < str.length; i++)
{
  trace(str.charAt(i), "-", str.charCodeAt(i));
}
```

Quando viene eseguito questo codice, si ottiene il seguente risultato:

```
h - 104
e - 101
l - 108
l - 108
o - 111
  - 32
w - 119
o - 111
r - 114
l - 108
d - 100
! - 33
```

È inoltre possibile utilizzare codici di caratteri per definire una stringa mediante il metodo `fromCharCode()`, come nell'esempio seguente:

```
var myStr:String =
  String.fromCharCode(104,101,108,108,111,32,119,111,114,108,100,33);
    // Imposta myStr su "hello world!"
```

Confronto tra stringhe

È possibile utilizzare i seguenti operatori per confrontare le stringhe <, <=, !=, ==, => e >. Tali operatori possono essere utilizzati con istruzioni condizionali, quali `if` e `while`, come nell'esempio seguente:

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2)
{
  trace("A < a, B < b, C < c, ...");
}
```

Quando si usano questi operatori con le stringhe, ActionScript considera il valore del codice di ciascun carattere della stringa e confronta i caratteri da sinistra a destra, come nell'esempio seguente:

```
trace("A" < "B"); // true
trace("A" < "a"); // true
trace("Ab" < "az"); // true
trace("abc" < "abza"); // true
```

Utilizzare gli operatori == e != per confrontare le stringhe tra loro e per confrontare le stringhe con altri tipi di oggetti, come nell'esempio seguente:

```
var str1:String = "1";
var str1b:String = "1";
var str2:String = "2";
trace(str1 == str1b); // true
trace(str1 == str2); // false
var total:uint = 1;
trace(str1 == total); // true
```

Come ottenere rappresentazioni di altri oggetti sotto forma di stringa

È possibile ottenere la rappresentazione sotto forma di stringa di qualsiasi tipo di oggetto. Tutti gli oggetti dispongono, a questo scopo, di un metodo `toString()`:

```
var n:Number = 99.47;
var str:String = n.toString();
  // str == "99.47"
```

Quando si usa l'operatore di concatenazione + con una combinazione di oggetti String e di oggetti che non sono stringhe, non è necessario utilizzare il metodo `toString()`.
Per informazioni dettagliate sulla concatenazione, consultare la sezione successiva.

La funzione generale `String()` restituisce lo stesso valore di un determinato oggetto restituito dall'oggetto che chiama il metodo `toString()`.

Concatenazione di stringhe

Concatenare più stringhe significa prendere due stringhe e unirle in modo sequenziale per formare un'unica stringa. Ad esempio, è possibile utilizzare l'operatore + per concatenare due stringhe:

```
var str1:String = "green";
var str2:String = "ish";
var str3:String = str1 + str2; // str3 == "greenish"
```

È inoltre possibile ottenere lo stesso risultato con l'operatore +=, come nell'esempio seguente:

```
var str:String = "green";
str += "ish"; // str == "greenish"
```

Inoltre, la classe String include un metodo `concat()`, che può essere utilizzato come segue:

```
var str1:String = "Bonjour";
var str2:String = "from";
var str3:String = "Paris";
var str4:String = str1.concat(" ", str2, " ", str3);
// str4 == "Bonjour from Paris"
```

Se si usa l'operatore + (o l'operatore +=) con un oggetto String e un oggetto *diverso* da una stringa, ActionScript converte automaticamente l'oggetto non-stringa in un oggetto String per poter valutare l'espressione, come nell'esempio seguente:

```
var str:String = "Area = ";
var area:Number = Math.PI * Math.pow(3, 2);
str = str + area; // str == "Area = 28.274333882308138"
```

Tuttavia, è possibile utilizzare le parentesi per raggruppare gli oggetti e fornire contesto all'operatore +, come nell'esempio seguente:

```
trace("Total: $" + 4.55 + 1.45); // output: Total: $4.551.45
trace("Total: $" + (4.55 + 1.45)); // output: Total: $6
```

Ricerca di sottostringhe e modelli nelle stringhe

Le sottostringhe sono caratteri sequenziali all'interno di una stringa. Ad esempio, la stringa `"abc"` presenta le seguenti sottostringhe: `""`, `"a"`, `"ab"`, `"abc"`, `"b"`, `"bc"`, `"c"`. I metodi di ActionScript consentono di individuare le sottostringhe di una stringa.

I modelli vengono definiti in ActionScript da stringhe o da espressioni regolari. Ad esempio, la seguente espressione regolare definisce un modello specifico, le lettere A, B e C seguite da un carattere di digitazione (il carattere barra è un delimitatore delle espressioni regolari):

```
/ABC\d/
```

ActionScript include metodi per cercare i modelli nelle stringhe e per sostituire le corrispondenze trovate con sottostringhe. Tali metodi vengono descritti più dettagliatamente nelle sezioni che seguono.

Le espressioni regolari sono in grado di definire modelli intricati. Per ulteriori informazioni, vedere Capitolo 9, "Uso delle espressioni regolari" a pagina 305.

Ricerca di una sottostringa in base alla posizione del carattere

I metodi `substr()` e `substring()` sono simili. Entrambi restituiscono infatti una sottostringa di una stringa e accettano due parametri. In entrambi i metodi, il primo parametro corrisponde alla posizione del carattere iniziale della stringa. Nel metodo `substr()`, tuttavia, il secondo parametro corrisponde alla *lunghezza* della sottostringa da restituire, mentre nel metodo `substring()` il secondo parametro definisce la posizione del carattere alla *fine* della sottostringa (che non è incluso nella stringa restituita). Nell'esempio seguente viene illustrata la differenza tra questi due metodi:

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.substr(11,15)); // output: Paris, Texas!!!
trace(str.substring(11,15)); // output: Pari
```

Il funzionamento del metodo `slice()` è analogo al metodo `substring()`. Se si utilizzano come parametri due numeri interi non negativi, il risultato è identico. Con il metodo `slice()` è tuttavia possibile utilizzare come parametri numeri interi negativi, nel qual caso la posizione del carattere viene rilevata dalla fine della stringa, come nell'esempio seguente:

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.slice(11,15)); // output: Pari
trace(str.slice(-3,-1)); // output: !!
trace(str.slice(-3,26)); // output: !!!
trace(str.slice(-3,str.length)); // output: !!!
trace(str.slice(-8,-3)); // output: Texas
```

Come parametri del metodo `slice()`, è possibile combinare numeri interi non negativi e negativi.

Individuazione della posizione del carattere di una sottostringa corrispondente

I metodi `indexOf()` e `lastIndexOf()` possono essere utilizzati per individuare le sottostringhe corrispondenti in una stringa, come illustrato nell'esempio seguente:

```
var str:String = "The moon, the stars, the sea, the land";
trace(str.indexOf("the")); // output: 10
```

Il metodo `indexOf()` fa distinzione tra maiuscole e minuscole.

Per indicare la posizione di indice nella stringa dalla quale iniziare la ricerca, è possibile specificare un secondo parametro, come di seguito:

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.indexOf("the", 11)); // output: 21
```

Il metodo `lastIndexOf()` trova l'ultima occorrenza di una sottostringa nella stringa:

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the")); // output: 30
```

Se con il metodo `lastIndexOf()` si include un secondo parametro, la ricerca viene eseguita all'indietro (da destra a sinistra) a partire dalla posizione di indice nella stringa:

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the", 29)); // output: 21
```

Creazione di un array di sottostringhe segmentate da un delimitatore

È possibile utilizzare il metodo `split()` per creare un array di sottostringhe divise in base a un carattere delimitatore. Ad esempio, è possibile segmentare una stringa delimitata da virgole o da tabulazioni in più stringhe.

L'esempio seguente mostra come dividere un array in sottostringhe con la e commerciale (&) come delimitatore:

```
var queryStr:String = "first=joe&last=cheng&title=manager&StartDate=3/6/
  65";
var params:Array = queryStr.split("&", 2);
  // params == ["first=joe","last=cheng"]
```

Il secondo parametro del metodo `split()`, che è opzionale, definisce le dimensioni massime dell'array restituito.

È possibile utilizzare anche un'espressione regolare come carattere delimitatore:

```
var str:String = "Give me\t5."
var a:Array = str.split(/\s+/); // a == ["Give","me","5."]
```

Per ulteriori informazioni, vedere Capitolo 9, "Uso delle espressioni regolari" a pagina 305 e *Guida di riferimento del linguaggio e ai componenti ActionScript 3.0*.

Ricerca di modelli nelle stringhe e sostituzione di sottostringhe

La classe String include i seguenti metodi per eseguire operazioni con i modelli all'interno delle stringhe:

- Utilizzare i metodi `match()` e `search()` per trovare sottostringhe corrispondenti a un modello.
- Utilizzare il metodo `replace()` per trovare sottostringhe corrispondenti a un modello e sostituirle con una sottostringa specificata.

Tali metodi vengono descritti più dettagliatamente nelle sezioni che seguono.

È possibile utilizzare stringhe o espressioni regolari per definire i modelli utilizzati in questi metodi. Per ulteriori informazioni sulle espressioni regolari, vedere Capitolo 9, "Uso delle espressioni regolari" a pagina 305.

Ricerca di sottostringhe corrispondenti

Il metodo `search()` restituisce la posizione di indice della prima sottostringa corrispondente a un dato modello, come nell'esempio seguente:

```
var str:String = "The more the merrier.";
// (Nella ricerca si fa distinzione tra lettere maiuscole e minuscole.)
trace(str.search("the")); // output: 9
```

È possibile utilizzare espressioni regolari per definire il modello da utilizzare come corrispondenza, come illustrato nell'esempio seguente:

```
var pattern:RegExp = /the/i;
var str:String = "The more the merrier.";
trace(str.search(pattern)); // 0
```

Il risultato del metodo `trace()` è 0, in quanto il primo carattere della stringa si trova nella posizione di indice. L'indicatore `i` viene impostato nell'espressione regolare, in modo che nella ricerca non si faccia distinzione tra maiuscole e minuscole.

Il metodo `search()` consente di trovare solo una corrispondenza e restituisce la sua posizione di indice iniziale, anche se l'indicatore `g` (global) viene impostato nell'espressione regolare.

Nell'esempio seguente è illustrata un'espressione regolare più complessa, che deve corrispondere a una stringa racchiusa tra virgolette doppie:

```
var pattern:RegExp = /"[^"]*"/;
var str:String = "The \"more\" the merrier.";
trace(str.search(pattern)); // output: 4


str = "The \"more the merrier.";
trace(str.search(pattern)); // output: -1
// (Indica nessuna corrispondenza, in quanto mancano le virgolette doppie
// di chiusura.)
```

Il metodo `match()` funziona in modo simile. Esso consente di cercare una sottostringa corrispondente. Tuttavia, se si usa l'indicatore global in un modello di espressione regolare, come nell'esempio seguente, il metodo `match()` restituisce un array di sottostringhe corrispondenti:

```
var str:String = "bob@example.com, omar@example.org";
var pattern:RegExp = /\w*@\w*\.[org|com]+/g;
var results:Array = str.match(pattern);
```

L'array `results` è impostato come segue:

```
["bob@example.com","omar@example.org"]
```

Per ulteriori informazioni sulle espressioni regolari, vedere Capitolo 9, "Uso delle espressioni regolari" a pagina 305.

Sostituzione di sottostringhe corrispondenti

Utilizzare il metodo `replace()` per cercare un modello specifico all'interno di una stringa e sostituire le corrispondenze trovate con la stringa specificata, come illustrato nell'esempio seguente:

```
var str:String = "She sells seashells by the seashore.";
var pattern:RegExp = /sh/gi;
trace(str.replace(pattern, "sch"));
  //sche sells seaschells by the seaschore.
```

Si noti come, nell'esempio, nelle stringe corrispondenti non si fa distinzione tra lettere maiuscole e minuscole, in quanto è stato impostato l'indicatore `i` (`ignoreCase`) nell'espressione regolare e le corrispondenze multiple vengono sostituite in quanto è stato specificato l'indicatore `g` (`global`). Per ulteriori informazioni, vedere Capitolo 9, "Uso delle espressioni regolari" a pagina 305.

Nella stringa di sostituzione è possibile includere i seguenti codici di sostituzione `$`. Il testo di sostituzione visualizzato nella tabella seguente viene inserito al posto del codice di sostituzione `$` :

Codice $	Testo sostitutivo
`$$`	$
`$&`	Sottostringa di cui si è trovata la corrispondenza.
`$``	Porzione della stringa che precede la sottostringa di cui si è trovata la corrispondenza. In questo codice viene impiegato il carattere virgoletta singola dritta sinistra (`) e non la virgoletta dritta singola (') o la virgoletta singola curva sinistra (‘).
`$'`	La porzione della stringa che segue la sottostringa di cui si è trovata la corrispondenza. In questo codice viene utilizzata la virgoletta singola dritta (').
`$`*n*	Corrispondenza con l'*n*simo gruppo parentetico catturato, dove *n* è un numero a cifra singola compreso tra 1 e 9 e **$***n* non è seguito da una cifra decimale.
`$`*nn*	Corrispondenza con l'*nn*simo gruppo parentetico catturato, dove *nn* è un numero decimale a due cifre compreso tra 01 e 99. Se l'*nn*sima cattura non è definita, il testo di sostituzione è una stringa vuota.

Ad esempio, il codice seguente illustra l'utilizzo dei codici sostitutivi `$2` e `$1`, che rappresentano il primo e il secondo gruppo di cattura di cui è stata trovata una corrispondenza:

```
var str:String = "flip-flop";
var pattern:RegExp = /(\w+)-(\w+)/g;
trace(str.replace(pattern, "$2-$1")); // flop-flip
```

È possibile usare una funzione come secondo parametro del metodo `replace()`. Il testo corrispondente viene sostituito dal valore restituito dalla funzione.

```
var str:String = "Now only $9.95!";
var price:RegExp = /\$([\d,]+.\d+)+/i;
trace(str.replace(price, usdToEuro));

function usdToEuro(matchedSubstring:String,
                   capturedMatch1:String,
                   index:int,
                   str:String):String
{
  var usd:String = capturedMatch1;
  usd = usd.replace(",", "");
  var exchangeRate:Number = 0.853690;
  var euro:Number = usd * exchangeRate;
  const euroSymbol:String = String.fromCharCode(8364);
  return euro.toFixed(2) + " " + euroSymbol;
}
```

Se si usa una funzione come secondo parametro del metodo `replace()`, alla funzione vengono inviati i seguenti argomenti:

- La porzione corrispondente della stringa.
- Tutte le corrispondenze dei gruppi parentetici catturati. Il numero di argomenti passati in questo modo varia a seconda del numero di corrispondenze parentetiche. È possibile determinare il numero di corrispondenze parentetiche verificando il valore `arguments.length - 3` all'interno del codice della funzione.
- La posizione di indice nella stringa in cui inizia la corrispondenza.
- La stringa completa.

Conversione di stringhe da maiuscole a minuscole

Come illustrato nell'esempio seguente, i metodi `toLowerCase()` e `toUpperCase()` convertono i caratteri alfabetici contenuti nella stringa rispettivamente in caratteri minuscoli e maiuscoli:

```
var str:String = "Dr. Bob Roberts, #9."
trace(str.toLowerCase()); // dr. bob roberts, #9.
trace(str.toUpperCase()); // DR. BOB ROBERTS, #9.
```

Dopo l'esecuzione di questi metodi, la stringa di origine rimane invariata. Per trasformare la stringa di origine, utilizzare il codice seguente:

```
str = str.toUpperCase();
```

Questi metodi funzionano con caratteri estesi, non semplicemente a–z e A–Z:

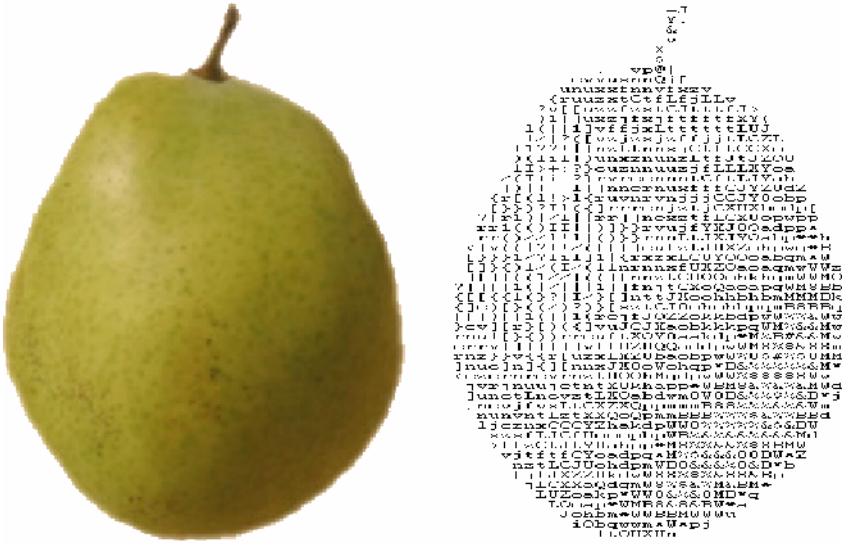
```
var str:String = "José Barça";
trace(str.toUpperCase(), str.toLowerCase()); // JOSÉ BARÇA josé barça
```

Esempio: ASCII Art

L'esempio ASCII Art illustra varie funzionalità della classe String in ActionScript 3.0, incluso le seguenti:

- Metodo `split()` della classe String, utilizzato per estrarre valori da una stringa delimitata da caratteri (informazioni di immagine in un file di testo delimitato da tabulazione).
- Varie tecniche di manipolazione delle stringhe, incluso il metodo `split()`, la concatenazione e l'estrazione di una porzione della stringa mediante i metodi `substring()` e `substr()`, che possono essere utilizzate per rendere maiuscola la prima lettera ogni parola nei titoli delle immagini.
- Metodo `getCharAt()`, che consente di ottenere un solo carattere da una stringa (per determinare il carattere ASCII corrispondente a un valore bitmap in scala di grigi).
- La concatenazione di stringhe, utilizzata per creare la rappresentazione ASCII art di un'immagine, un carattere per volta.

Il termine *ASCII art* si riferisce a rappresentazioni di immagini sotto forma di testo, nelle quali una griglia di caratteri a spaziatura fissa, quali Courier New, va a formare l'immagine. L'illustrazione seguente riporta un esempio di ASCII art prodotto dall'applicazione:



La versione ASCII art dell'immagine è riportata sulla destra.

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione ASCIIArt si trovano nella cartella Samples/AsciiArt. L'applicazione è costituita dai seguenti file:

File	Descrizione
AsciiArtApp.mxml o AsciiArtApp.fla	Il file dell'applicazione principale in Flash (FLA) o Flex (MXML)
com/example/programmingas3/asciiArt/AsciiArtBuilder.as	Classe che fornisce le funzionalità principali dell'applicazione, inclusa l'estrazione dei metadati dell'immagine da un file di testo, il caricamento delle immagini e la gestione del processo di conversione immagine-testo.
com/example/programmingas3/asciiArt/BitmapToAsciiConverter.as	Classe che fornisce il metodo `parseBitmapData()` per la conversione dei dati immagini in versione stringa.

File	Descrizione
com/example/programmingas3/asciiArt/Image.as	Classe che rappresenta un'immagine bitmap caricata.
com/example/programmingas3/asciiArt/ImageInfo.as	Classe che rappresenta metadati di un'immagine ASCII art (quali titolo, URL file di immagine e così via).
image/	Cartella contenente immagini utilizzate dall'applicazione.
txt/ImageData.txt	File di testo delimitato da tabulazione contenente informazioni sulle immagini da caricare mediante l'applicazione.

Estrazione di valori delimitati da tabulazione

In questo esempio viene utilizzato il sistema molto diffuso di memorizzare i dati dell'applicazione in un'ubicazione separata rispetto all'applicazione stessa; in questo modo, se i dati vengono modificati (ad esempio, se viene aggiunta una nuova immagine o se il titolo di un'immagine viene modificato), non sarà più necessario ricreare il file SWF. In questo caso, i metadati dell'immagine, incluso il titolo dell'immagine, l'URL del file dell'immagine vero e proprio e alcuni valori utilizzati per manipolare l'immagine sono memorizzati in un file di testo (il file txt/ImageData.txt del progetto). Il file di testo contiene quanto segue:

```
FILENAMETITLEWHITE_THRESHHOLDBLACK_THRESHHOLD
FruitBasket.jpgPear, apple, orange, and bananad810
Banana.jpgA picture of a bananaC820
Orange.jpgorangeFF20
Apple.jpgpicture of an apple6E10
```

Il file impiega un formato specifico delimitato da tabulazione. La prima riga è una riga di intestazione. Le restanti righe contengono i seguenti dati per ciascuna bitmap da caricare:

- Il nome file della bitmap.
- Il nome di visualizzazione della bitmap.
- I valori di soglia del bianco e soglia del nero per le bitmap. Si tratta di valori esadecimali al di sopra o al di sotto dei quali un pixel viene considerato completamente bianco o completamente nero.

Non appena l'applicazione viene avviata, la classe AsciiArtBuilder carica ed analizza il contenuto del file di testo per creare uno "stack" di immagini da visualizzare mediante il seguente codice del metodo `parseImageInfo()` della classe AsciiArtBuilder:

```
var lines:Array = _imageInfoLoader.data.split("\n");
var numLines:uint = lines.length;
for (var i:uint = 1; i < numLines; i++)
{
  var imageInfoRaw:String = lines[i];
  ...
  if (imageInfoRaw.length > 0)
  {
    // Crea un nuovo record di informazioni immagine e lo aggiunge
    // all'array delle informazioni immagine.
    var imageInfo:ImageInfo = new ImageInfo();

    // Divide la riga corrente in valori (separati da tabulazione (\t))
    // ed estrae singole proprietà:
    var imageProperties:Array = imageInfoRaw.split("\t");
    imageInfo.fileName = imageProperties[0];
    imageInfo.title = normalizeTitle(imageProperties[1]);
    imageInfo.whiteThreshold = parseInt(imageProperties[2], 16);
    imageInfo.blackThreshold = parseInt(imageProperties[3], 16);
    result.push(imageInfo);
  }
}
```

Tutto il contenuto del file di testo è racchiuso in una sola istanza di String, la proprietà `_imageInfoLoader.data`. Utilizzando il metodo `split()` con il carattere nuova riga (`"\n"`) come parametro, l'istanza String viene divisa in un array (`lines`) i cui elementi sono singole righe del file di testo. Quindi, il codice impiega un ciclo per lavorare con ogni singola riga (tranne la prima, che contiene solo l'intestazione). All'interno del ciclo, il metodo `split()` viene utilizzato di nuovo per dividere il contenuto della singola riga in una serie di valori (l'oggetto Array chiamato `imageProperties`). Il parametro utilizzato con il metodo `split()` in questo caso è il carattere tabulazione (`"\t"`), in quanto i valori di ogni riga sono delimitati dalla tabulazione.

Uso dei metodi della classe String per normalizzare i titoli delle immagini

Una delle decisioni di progettazione di questa applicazione è che tutti i titoli delle immagini vengano visualizzati in un formato standard, con la prima lettera di ogni parola in maiuscolo (tranne che per alcune parole che generalmente non vengono scritte in maiuscolo nei titoli). Anziché dare per scontato che il file di testo contenga titoli formattati correttamente, l'applicazione formatta i titoli mentre vengono estratti dal file di testo.

Nell'elenco di codice precedente, viene utilizzata la riga seguente per estrarre valori metadati dalle singole immagini:

```
imageInfo.title = normalizeTitle(imageProperties[1]);
```

In questo codice, il titolo dell'immagine viene trasmesso dal file di testo attraverso il metodo `normalizeTitle()` prima di essere memorizzato nell'oggetto ImageInfo:

```
private function normalizeTitle(title:String):String
{
  var words:Array = title.split(" ");
  var len:uint = words.length;
  for (var i:uint; i < len; i++)
  {
    words[i] = capitalizeFirstLetter(words[i]);
  }

  return words.join(" ");
}
```

Questo metodo impiega il metodo `split()` per dividere il titolo in singole parole (separate dal carattere spazio), passa ogni singola parola attraverso il metodo `capitalizeFirstLetter()`, quindi usa il metodo `join()` della classe Array per combinare le parole di nuovo in un'unica stringa.

Come suggerisce il nome, il metodo `capitalizeFirstLetter()` consente di rendere maiuscola la prima lettera di ogni parola:

```
/**
 * Rende maiuscola la prima lettera di una singola parola, eccetto
 * le parole che generalmente non sono maiuscole nei titoli.
 */
private function capitalizeFirstLetter(word:String):String
{
  switch (word)
  {
    case "and":
    case "the":
    case "in":
    case "an":
    case "or":
    case "at":
    case "of":
    case "a":
      // Non esegue alcuna operazione su queste parole.
      break;
    default:
      // In tutte le altre parole, scrive in maiuscolo
      // il primo carattere.
      var firstLetter:String = word.substr(0, 1);
      firstLetter = firstLetter.toUpperCase();
      var otherLetters:String = word.substring(1);
      word = firstLetter + otherLetters;
  }
  return word;
}
```

Nei titoli, il primo carattere delle seguenti parole *non* viene scritto in maiuscolo: “e”, “il”, “in”, “un”, “o”, “a”, “di”, “per”, ecc. (in genere, si tratta di articoli e preposizioni). Per eseguire questa logica, il codice impiega un’istruzione `switch` per verificare se la parola è tra quelle che non devono essere scritte in maiuscolo. In tal caso, il codice salta semplicemente l’istruzione `switch`. Se invece la parola deve essere scritta in maiuscolo, l’operazione di trasformazione viene eseguita in varie fasi:

1. La prima lettera della parola viene estratta mediante la funzione `substr(0, 1)`, che consente di estrarre una sottostringa che inizia con il carattere in posizione di indice 0 (la prima lettera della stringa, come indicato dal primo parametro `0`). La sottostringa è lunga un solo carattere (indicato dal secondo parametro `1`).
2. Tale carattere viene scritto in maiuscolo utilizzando il metodo `toUpperCase()`.

3. I caratteri rimanenti della parola originale vengono estratti mediante la funzione `substring(1)`, che consente di estrarre una sottostringa che inizia nella posizione di indice 1 (seconda lettera) fino alla fine della stringa (indicato dalla non impostazione del secondo parametro del metodo `substring()`).
4. L'ultima parola viene creata combinando la prima lettera trasformata in maiuscolo con le restanti lettere mediante concatenazione della stringa: `firstLetter + otherLetters`.

Creazione di testo ASCII art

La classe BitmapToAsciiConverter fornisce la funzionalità di conversione di un'immagine bitmap nella sua rappresentazione di testo ASCII. Tale procedura viene eseguita dal metodo `parseBitmapData()`, parzialmente illustrato di seguito:

```
var result:String = "";

// Esegue un ciclo tra le righe di pixel dall'inizio alla fine:
for (var y:uint = 0; y < _data.height; y += verticalResolution)
{
  // In ogni riga, esegue un ciclo tra i pixel da sinistra a destra:
  for (var x:uint = 0; x < _data.width; x += horizontalResolution)
  {
    ...

    // Converte il valore grigio nell'intervallo compreso tra 0 e 255 in
    // un valore compreso tra 0 e 64 (corrispondente al numero
    // di "sfumature di grigio" nel set di caratteri disponibili):
    index = Math.floor(grayVal / 4);
    result += palette.charAt(index);
  }
  result += "\n";
}
return result;
```

Questo codice definisce in primo luogo un'istanza di String denominata `result` che verrà utilizzata per creare la versione ASCII art dell'immagine bitmap. Quindi, esegue una funzione ciclica attraverso i singoli pixel dell'immagine bitmap di origine. Mediante una serie di tecniche di manipolazione del colore (qui omesse per ragioni di brevità), i valori di rosso, verde e blu di ogni singolo pixel vengono convertiti in un solo valore in scala di grigi (un numero compreso tra 0 e 255). Il codice divide quindi tale valore per 4 (come illustrato) per convertirlo in un valore della scala che va da 0 a 63, memorizzata nella variabile `index`. (Viene utilizzata una scala da 0 a 63 perché la "tavolozza" dei caratteri ASCII disponibili per questa applicazione contiene 64 valori.) La tavolozza di caratteri viene definita come un'istanza di String nella classe BitmapToAsciiConverter:

```
// I caratteri sono ordinati dal più scuro al più chiaro, in modo che
// la loro posizione (index) nella stringa corrisponda al relativo valore
// di colore (0 = nero).
private static const palette:String =
  "@#$%&8BMW*mwqpdbkhaoQOOZXYUJCLtfjzxnuvcr[]{}1()|/?Il!i><+_~-;,. ";
```

Poiché la variabile `index` definisce quale carattere ASCII nella tavolozza corrisponde al pixel corrente dell'immagine bitmap, tale carattere viene recuperato dalla stringa `palette` mediante il metodo `charAt()`. Quindi, esso viene aggiunto all'istanza di String `result` mediante l'operatore di assegnazione della concatenazione (+=). Inoltre, alla fine di ogni riga di pixel, un carattere nuova riga viene concatenato alla fine della stringa `result`, per obbligare la riga a tornare a capo e creare una nuova riga di "pixel" di caratteri.

CAPITOLO 7

Operazioni con gli array

7

Gli array consentono di memorizzare più valori in un'unica struttura di dati. È possibile utilizzare array indicizzati semplici per memorizzare valori mediante indici interi ordinali fissi oppure array associativi complessi per memorizzare valori utilizzando chiavi arbitrarie. Gli array possono inoltre essere multidimensionali e contenere elementi che sono al loro volta array. In questo capitolo viene illustrato come creare e manipolare vari tipi di array.

Sommario

Elementi fondamentali degli array

Introduzione alle operazioni con gli array

Spesso nella programmazione è necessario lavorare con una serie di elementi anziché con un singolo oggetto; ad esempio, in un'applicazione di riproduzione musicale, potrebbe essere necessario disporre di un elenco di brani da riprodurre. Per non dover creare una variabile distinta per ogni singolo brano dell'elenco, sarebbe preferibile poter disporre di tutti gli oggetti Song raggruppati in un singolo "pacchetto" e poter lavorare con essi come gruppo.

Un array è un elemento di programmazione che funge da contenitore per una serie di voci, quale un elenco di brani. Generalmente, tutti gli elemento di un array sono istanze di una stessa classe, ma ciò non è un requisito di ActionScript. Le singole voci di un array sono conosciute come gli *elementi* dell'array. Un array può essere considerato come una directory contenente variabili. Le variabili possono essere inserite come elementi nell'array, esattamente come si inserisce una cartella in una directory. Una volta inseriti vari file nella directory, è possibile lavorare con l'array come se fosse un'unica variabile (ad esempio, è possibile trasportare l'intera directory in un'alta posizione), è possibile lavorare con le variabili come se fossero un unico gruppo (ad esempio, è possibile sfogliare le cartelle una a una per cercare una particolare informazione), oppure è possibile accedere a esse individualmente (come aprire la directory e selezionare una cartella).

Ad esempio, si immagini di creare un'applicazione di riproduzione musicale in cui sia possibile selezionare più brani e aggiungerli a un elenco di riproduzione. Nel codice ActionScript, è possibile inserire un metodo chiamato `addSongsToPlaylist()` che accetta un singolo array come parametro. Indipendentemente dal numero di brani da inserire nell'elenco (pochi, molti o uno soltanto), è necessario chiamare il metodo `addSongsToPlaylist()` una sola volta, per passarlo nell'array contenente gli oggetti Song. All'interno del metodo `addSongsToPlaylist()`, è possibile usare una funzione ciclica per passare in rassegna i vari elementi dell'array (i brani) uno a uno e aggiungerli all'elenco di riproduzione.

Il tipo di array più comune in ActionScript è l'*array indicizzato*, un array dove ogni elemento viene memorizzato in uno slot numerato (detto *indice*) e le voci sono accessibili mediante numeri, come negli indirizzi. La classe Array viene impiegata per rappresentare array indicizzati. Gli array indicizzati sono in grado di soddisfare la maggior parte delle esigenze di programmazione. Un impiego speciale dell'array indicizzato è l'array multidimensionale, che consiste in un array indicizzato i cui elementi sono altri array indicizzati (contenenti a loro volta altri elementi). Un altro tipo di array è l'*array associativo*, che impiega una *chiave* in formato stringa anziché un indice numerico per identificare i vari elementi. Infine, per gli utenti avanzati, ActionScript 3.0 include anche la classe Dictionary, che rappresenta un vero e proprio *dizionario*, un array che consente di utilizzare qualsiasi tipo di oggetto come chiave di identificazione degli elementi.

Operazioni comuni con gli array

In questo capitolo vengono descritte le seguenti attività comuni eseguibili con gli array:

- Creazione di array indicizzati
- Aggiunta e rimozione di elementi di array
- Ordinamento di elementi di array
- Estrazione di porzioni di un array
- Uso degli array associativi e dei dizionari
- Uso degli array multidimensionali
- Copia di elementi di array
- Creazione di sottoclassi di un array

Concetti e termini importanti

Il seguente elenco di riferimento contiene termini importanti utilizzati nel presente capitolo:

- Array: oggetto che funge da contenitore per raggruppare più oggetti.
- Array associativo: array che impiega chiavi di stringa per identificare singoli elementi.
- Dizionario: array i cui elementi sono costituiti da coppie di oggetti, conosciuti come chiave e valore. La chiave viene utilizzata al posto dell'indice numerico per identificare singoli elementi.
- Elemento: singola voce di un array.
- Indice: "indirizzo" numerico utilizzato per identificare un singolo elemento in un array indicizzato.
- Array con indice: tipo di array standard nel quale gli elementi vengono memorizzati come elementi numerati e tale numerazione (indice) viene impiegata per identificare i singoli elementi.
- Chiave: stringa o oggetto utilizzato per identificare un singolo elemento in un array associativo o in un dizionario.
- Array multidimensionale: array contenente elementi che sono a loro volta array anziché singoli valori.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Tutti questi esempi includono la chiamata appropriata alla funzione `trace()`. Per provare gli esempi di codice contenuti in questo capitolo:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.
4. Eseguire il programma selezionando Controllo > Prova filmato.

   I risultati della funzione `trace()` vengono visualizzati nel pannello Output.

Per ulteriori informazioni sulle tecniche per la prova degli esempi di codice, vedere "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Array con indice

Gli array con indice sono in grado di memorizzare uno o più valori organizzati in modo che ciascun valore risulti accessibile mediante l'uso di un valore intero. Il primo indice è sempre il numero 0 e viene incrementato di un'unità a ogni elemento successivo aggiunto all'array. Come illustrato nel codice seguente, per creare un array indicizzato, è necessario chiamare la funzione di costruzione della classe Array o inizializzare l'array con un valore letterale array:

```
// Usa la funzione di costruzione Array.
var myArray:Array = new Array();
myArray.push("one");
myArray.push("two");
myArray.push("three");
trace(myArray); // output: one,two,three

// Usa il valore letterale dell'array.
var myArray:Array = ["one", "two", "three"];
trace(myArray); // output: one,two,three
```

La classe Array può inoltre contenere proprietà e metodi che consentono di modificare gli array indicizzati. Tali proprietà e metodi vengono applicati quasi esclusivamente ad array indicizzati piuttosto che ad array associativi.

Gli array con indice impiegano valori interi senza segni a 32 bit come numeri di indice.

La dimensione massima di un array con indice è $2^{32}1$ o 4.294.967.295. Qualsiasi tentativo di creare un array di dimensioni superiori genera un errore di runtime.

Un elemento di array può contenere valori di dati di qualsiasi tipo. ActionScript 3.0 non supporta il concetto di *array tipizzati*, vale a dire che non è possibile specificare che tutti gli elementi di un array appartengano a un solo tipo di dati.

In questa sezione è illustrato come creare e modificare array indicizzati mediante la classe Array, a partire dalla creazione di un array. I metodi che consentono di modificare gli array sono raggruppati in tre categorie, corrispondenti alle procedure per inserire elementi, rimuovere elementi e ordinare gli array. Esiste infine un gruppo di metodi che consente di trattare array esistenti come array di sola lettura; tali metodi si limitano a eseguire query sugli array. Anziché modificare un array esistente, i metodi di query restituiscono tutti un nuovo array. La sezione si conclude con una discussione sull'estensione della classe Array.

Creazione di array

La funzione di costruzione della classe Array può essere utilizzata in tre diversi modi. Primo: se si chiama la funzione di costruzione senza argomenti, viene restituito un array vuoto. Utilizzare la proprietà `length` della classe Array per verificare che l'array non contenga elementi. Ad esempio, il codice seguente consente di chiamare la funzione di costruzione della classe Array senza argomenti:

```
var names:Array = new Array();
trace(names.length); // output: 0
```

Secondo: se si usa un numero come unico parametro della funzione di costruzione della classe Array, viene creato un array di lunghezza corrispondente, con il valore di ciascun elemento impostato su `undefined`. L'argomento deve essere un numero intero senza segno compreso tra 0 e 4.294.967.295. Ad esempio, il codice seguente consente di chiamare la funzione di costruzione Array con un solo argomento numerico:

```
var names:Array = new Array(3);
trace(names.length); // output: 3
trace(names[0]); // output: undefined
trace(names[1]); // output: undefined
trace(names[2]); // output: undefined
```

Terzo: se si chiama la funzione di costruzione e si invia un elenco di elementi come parametri, viene creato un array con elementi corrispondenti a ognuno dei parametri. Il codice seguente consente di trasmettere tre argomenti alla funzione di costruzione Array:

```
var names:Array = new Array("John", "Jane", "David");
trace(names.length); // output: 3
trace(names[0]); // output: John
trace(names[1]); // output: Jane
trace(names[2]); // output: David
```

È inoltre possibile creare array con valori letterali di array o valori letterali oggetto. È possibile assegnare un valore letterale di array direttamente a una qualsiasi variabile di array, come illustrato nell'esempio seguente:

```
var names:Array = ["John", "Jane", "David"];
```

Inserimento di elementi di array

Tre dei metodi della classe Array (`push()`, `unshift()` e `splice()`) consentono di inserire elementi in un array. Il metodo `push()` consente di aggiungere uno o più elementi al termine di un array. In altre parole, l'ultimo elemento inserito nell'array mediante il metodo `push()` avrà il numero di indice più alto. Il metodo `unshift()` consente di inserire uno o più elementi all'inizio di un array, vale a dire in posizione di indice 0. Il metodo `splice()` consente di inserire un qualsiasi numero di elementi in una posizione di indice specificata dell'array.

L'esempio seguente illustra tutti e tre i metodi. Viene creato un array denominato `planets` per memorizzare i nomi dei pianeti in ordine di prossimità al sole. In primo luogo, viene chiamato il metodo `push()` per inserire l'elemento iniziale, `Mars`. Quindi viene chiamato il metodo `unshift()` per inserire l'elemento che appartiene alla prima parte dell'array, `Mercury`. Infine, viene chiamato il metodo `splice()` per inserire gli elementi `Venus` e `Earth` dopo `Mercury`, ma prima di `Mars`. Il primo argomento inviato a `splice()`, il valore intero 1, indica che l'inserimento deve iniziare dalla posizione di indice 1. Il secondo argomento inviato a `splice()`, il valore intero 0, indica che nessun elemento deve essere eliminato. Infine, il terzo e il quarto argomento inviati a `splice()`, `Venus` e `Earth`, rappresentano gli elementi da inserire.

```
var planets:Array = new Array();
planets.push("Mars");       // Contenuto array: Mars
planets.unshift("Mercury"); // Contenuto array: Mercury,Mars
planets.splice(1, 0, "Venus", "Earth");
trace(planets);             // Contenuto array: Mercury,Venus,Earth,Mars
```

I metodi `push()` e `unshift()` restituiscono entrambi un numero intero senza segno che rappresenta la lunghezza dell'array modificato. Il metodo `splice()` restituisce un array vuoto se utilizzato per inserire elementi, cosa che potrebbe sembrare strana, ma che acquista maggior senso alla luce della versatilità del metodo `splice()`. Il metodo `splice()` non consente solo di inserire elementi in un array, ma anche di rimuovere elementi. Se utilizzato per rimuovere elementi, il metodo `splice()` restituisce un array contenente gli elementi rimossi.

Rimozione di elementi da un array

Tre dei metodi della classe Array (`pop()`, `shift()` e `splice()`) consentono di rimuovere elementi da un array. Il metodo `pop()` consente di rimuovere un elemento dalla fine di un array. In altre parole, con questo metodo è possibile rimuovere l'elemento nella posizione di indice più alta. Il metodo `shift()` rimuove un elemento dall'inizio dell'array, vale a dire l'elemento che si trova in posizione di indice 0. Il metodo `splice()`, che può essere utilizzato anche per inserire elementi, consente di rimuovere un numero arbitrario di elementi a partire dalla posizione di indice specificata dal primo argomento inviato al metodo.

L'esempio seguente impiega tutti e tre i metodi per rimuovere elementi da un array. Viene creato un array denominato `oceans` per memorizzare i nomi degli oceani. Alcuni nomi dell'array corrispondono a nomi di laghi, quindi andranno rimossi.

In primo luogo, viene chiamato il metodo `splice()` per rimuovere gli elementi `Aral` e `Superior` e vengono inseriti gli elementi `Atlantic` e `Indian`. Il primo argomento inviato a `splice()`, il valore intero 2, indica che l'operazione deve iniziare con il terzo elemento dell'elenco, che si trova in posizione di indice 2. Il secondo argomento, 2, indica che le due voci devono essere rimosse. Gli argomenti rimanenti, `Atlantic` e `Indian`, sono i valori da inserire nella posizione di indice 2.

Quindi, il metodo `pop()` viene utilizzato per rimuovere l'ultimo elemento dell'array, `Huron`. E successivamente, il metodo `shift()` viene utilizzato per rimuovere il primo elemento dell'array, `Victoria`.

```
var oceans:Array = ["Victoria", "Pacific", "Aral", "Superior", "Indian",
  "Huron"];
oceans.splice(2, 2, "Arctic", "Atlantic"); // Sostituisce Aral e Superior
oceans.pop();   // Rimuove Huron
oceans.shift(); // Rimuove Victoria
trace(oceans);  // output: Pacific,Arctic,Atlantic,Indian
```

I metodi `pop()` e `shift()` restituiscono entrambi l'elemento che era stato rimosso. Il tipo di dati del valore restituito è Object perché gli array possono contenere valori di qualunque tipo di dati. Il metodo `splice()` restituisce un array contenente i valori rimossi. È possibile modificare l'esempio dell'array `oceans` in modo che la chiamata al metodo `splice()` assegni all'array una nuova variabile di array, come illustrato dall'esempio seguente:

```
var lakes:Array = oceans.splice(2, 2, "Arctic", "Atlantic");
trace(lakes); // output: Aral,Superior
```

È possibile incontrare del codice che impiega l'operatore `delete` su un elemento di array. L'operatore `delete` imposta il valore di un elemento di array su `undefined`, ma non rimuove l'elemento dall'array. Ad esempio, il codice seguente impiega l'operatore `delete` sul terzo elemento nell'array `oceans`, ma la lunghezza dell'array rimane 5:

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Indian", "Atlantic"];
delete oceans[2];
trace(oceans);        // output: Arctic,Pacific,,Indian,Atlantic
trace(oceans[2]);     // output: undefined
trace(oceans.length); // output: 5
```

È possibile troncare un array mediante la proprietà `length` dell'array. Se si imposta la proprietà `length` di un array su una lunghezza inferiore alla lunghezza corrente dell'array, l'array viene troncato e gli elementi memorizzati nelle posizioni di indice successive al nuovo valore di `length` meno 1 vengono rimossi. Ad esempio, se l'array `oceans` fosse ordinato in modo che tutte le voci valide si trovassero all'inizio, sarebbe possibile utilizzare la proprietà `length` per rimuovere le voci alla fine dell'array, come illustrato nel codice seguente:

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];
oceans.length = 2;
trace(oceans); // output: Arctic,Pacific
```

Ordinamento di un array

Sono disponibili tre diversi metodi di ordinamento: `reverse()`, `sort()` e `sortOn()`, che consentono di modificare l'ordine di un array, mediante inversione dell'ordine o ordinamento in base a un elemento specifico. Tutti i metodi consentono di modificare l'array esistente. Il metodo `reverse()` modifica l'ordine dell'array in modo che l'ultimo elemento diventi il primo, il penultimo elemento diventi il secondo e così via. Il metodo `sort()` consente di ordinare un array in vari modi predefiniti e di creare algoritmi di ordinamento predefiniti. Il metodo `sortOn()` consente di ordinare un array indicizzato di oggetti che presentano una o più proprietà comuni da utilizzare come chiavi di ordinamento.

Il metodo `reverse()` non presenta parametri e non restituisce valori, ma permette di alternare l'ordine dell'array dallo stato corrente a quello inverso. Nell'esempio seguente viene invertito l'ordine degli oceani elencati nell'array `oceans`:

```
var oceans:Array = ["Arctic", "Atlantic", "Indian", "Pacific"];
oceans.reverse();
trace(oceans); // output: Pacific,Indian,Atlantic,Arctic
```

Il metodo `sort()` consente di riordinare gli elementi di un array utilizzando l'*ordine predefinito*. L'ordine predefinito presenta le seguenti caratteristiche:

- Nell'ordinamento si fa distinzione tra maiuscole e minuscole, vale a dire che i caratteri maiuscoli precedono i minuscoli. Ad esempio, la lettera D precede la lettera b.
- L'ordinamento è crescente, vale a dire che un codice contenente le prime lettere dell'alfabeto (quale A) precede il codice contenente le lettere successive (quale B).
- L'ordinamento colloca valori identici l'uno accanto all'altro, ma in nessun ordine particolare.
- L'ordinamento è basato sulle stringhe, vale a dire che gli elementi vengono convertiti in stringhe prima di essere confrontati (ad esempio, 10 precede 3 perché la stringa `"1"` ha un codice di carattere inferiore rispetto alla stringa `"3"`).

Potrebbe essere necessario ordinare un array a prescindere dai caratteri o in ordine decrescente, oppure l'array potrebbe contenere valori da ordinare numericamente anziché alfabeticamente. Il metodo `sort()` presenta un parametro `options` che consente di modificare singole caratteristiche dell'ordine predefinito. Le opzioni vengono definite da una serie di costanti statiche della classe Array, come illustrato nell'elenco seguente:

- `Array.CASEINSENSITIVE`: questa opzione consente di ignorare i caratteri durante l'ordinamento. Ad esempio, la lettera minuscola b precede la lettera maiuscola D.
- `Array.DESCENDING`: questa opzione inverte l'ordine crescente predefinito. Ad esempio, la lettera B precede la lettera A.
- `Array.UNIQUESORT`: questa opzione interrompe l'ordinamento se vengono rilevati due valori identici.
- `Array.NUMERIC`: questa opzione consente un ordinamento numerico, ad esempio 3 precede 10.

Nell'esempio seguente sono evidenziate alcune di queste opzioni. Viene creato un array chiamato `poets` ordinato in base a una serie di opzioni differenti.

```
var poets:Array = ["Blake", "cummings", "Angelou", "Dante"];
poets.sort(); // ordinamento predefinito
trace(poets); // output: Angelou,Blake,Dante,cummings

poets.sort(Array.CASEINSENSITIVE);
trace(poets); // output: Angelou,Blake,cummings,Dante

poets.sort(Array.DESCENDING);
trace(poets); // output: cummings,Dante,Blake,Angelou

poets.sort(Array.DESCENDING | Array.CASEINSENSITIVE); // usa due opzioni
trace(poets); // output: Dante,cummings,Blake,Angelou
```

È inoltre possibile scrivere una propria funzione di ordinamento personalizzata da trasmettere come parametro al metodo `sort()`. Ad esempio, se si ha un elenco di nomi in cui ogni elemento contiene nome e cognome di una persona, ma si desidera ordinare l'elenco solo in base al cognome, è necessario utilizzare una funzione di ordinamento personalizzata per analizzare ogni elemento e usare solo il cognome nella funzione di ordinamento. Il codice seguente illustra come è possibile farlo con una funzione personalizzata utilizzata come parametro del metodo `Array.sort()`:

```
var names:Array = new Array("John Q. Smith", "Jane Doe", "Mike Jones");
function orderLastName(a, b):int
{
  var lastName:RegExp = /\b\S+$/;
  var name1 = a.match(lastName);
  var name2 = b.match(lastName);
  if (name1 < name2)
  {
    return -1;
  }
  else if (name1 > name2)
  {
    return 1;
  }
  else
  {
    return 0;
  }
}

trace(names); // output: John Q. Smith,Jane Doe,Mike Jones
names.sort(orderLastName);
trace(names); // output: Jane Doe,Mike Jones,John Q. Smith
```

La funzione di ordinamento personalizzata `orderLastName()` impiega un'espressione regolare per estrarre il cognome da ogni elemento da usare per l'operazione di confronto. L'identificatore della funzione `orderLastName` viene utilizzato come unico parametro durante la chiamata al metodo `sort()` per l'array `names`. La funzione di ordinamento accetta due parametri, `a` e `b`, in quanto funziona su due elementi di array per volta. Il valore restituito dalla funzione di ordinamento indica come devono essere ordinati gli elementi:

- Un valore restituito di -1 indica che il primo parametro, `a`, precede il secondo parametro, `b`.
- Un valore restituito di 1 indica che il secondo parametro, `b`, precede il primo parametro, `a`.
- Un valore restituito di 0 indica che gli elementi hanno precedenza di ordinamento uguale.

Il metodo `sortOn()` è destinato ad array indicizzati con elementi che contengono oggetti. Tali oggetti devono avere almeno una proprietà comune da utilizzare come chiave dell'ordinamento. L'uso del metodo `sortOn()` per array di qualsiasi altro tipo genera risultati inattesi.

Nell'esempio seguente l'array `poets` viene rivisto in modo che ciascun elemento sia un oggetto anziché una stringa. Ogni oggetto contiene il cognome del poeta e l'anno di nascita.

```
var poets:Array = new Array();
poets.push({name:"Angelou", born:"1928"});
poets.push({name:"Blake", born:"1757"});
poets.push({name:"cummings", born:"1894"});
poets.push({name:"Dante", born:"1265"});
poets.push({name:"Wang", born:"701"});
```

È possibile utilizzare il metodo `sortOn()` per ordinare l'array in base alla proprietà `born`. Il metodo `sortOn()` definisce due parametri, `fieldName` e `options`. L'argomento `fieldName` deve essere specificato come stringa. Nell'esempio seguente, `sortOn()` viene chiamato con due argomenti, `"born"` e `Array.NUMERIC`. L'argomento `Array.NUMERIC` viene utilizzato per fare in modo che l'ordinamento sia numerico anziché alfabetico. Si tratta di un sistema efficace quando tutti i valori presentano lo stesso numero di cifre, in quanto assicura che l'ordinamento continuerà a essere eseguito correttamente anche se nell'array viene inserito un valore con un numero di cifre inferiore o superiore.

```
poets.sortOn("born", Array.NUMERIC);
for (var i:int = 0; i < poets.length; ++i)
{
  trace(poets[i].name, poets[i].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

Generalmente, i metodi `sort()` e `sortOn()` consentono di modificare l'array. Se si desidera ordinare un array senza modificare l'array esistente, inviare la costante `Array.RETURNINDEXEDARRAY` come parte del parametro `options`. Questa opzione istruisce i metodi in modo che restituiscano un nuovo array che rifletta l'ordinamento richiesto e lasci invariato l'array originale. L'array restituito dai metodi è un semplice array di numeri di indice che riflette il nuovo ordinamento e non contiene elementi dell'array originale. Ad esempio, per ordinare l'array `poets` in base all'anno di nascita senza modificare l'array, includere la costante `Array.RETURNINDEXEDARRAY` nell'argomento trasmesso per il parametro `options`.

L'esempio seguente memorizza le informazioni di indice restituite in un array denominato `indices` e impiega l'array `indices` unitamente all'array `poets` non modificato per restituire i poeti in ordine di anno di nascita:

```
var indices:Array;
indices = poets.sortOn("born", Array.NUMERIC | Array.RETURNINDEXEDARRAY);
for (var i:int = 0; i < indices.length; ++i)
{
  var index:int = indices[i];
  trace(poets[index].name, poets[index].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

Query di un array

I restanti metodi della classe Array (`concat()`, `join()`, `slice()`, `toString()`) consentono di eseguire query nell'array per ottenere informazioni, senza modificare l'array. I metodi `concat()` e `slice()` restituiscono nuovi array, mentre i metodi `join()` e `toString()` restituiscono stringhe. Il metodo `concat()` impiega un nuovo array o un elenco di elementi come argomenti e li combina con l'array esistente per creare un nuovo array. Il metodo `slice()` impiega due parametri, denominati `startIndex` e `endIndex`, per restituire un nuovo array contenente una copia degli elementi a "sezionati" dall'array esistente. Il sezionamento inizia dall'elemento in posizione `startIndex` e termina con l'elemento che precede `endIndex`. Questo sistema prevede la ripetizione: l'elemento in posizione `endIndex` non è incluso nel valore restituito.

Nell'esempio seguente, vengono utilizzati i metodi `concat()` e `slice()` per creare nuovi array utilizzando elementi di altri array:

```
var array1:Array = ["alpha", "beta"];
var array2:Array = array1.concat("gamma", "delta");
trace(array2); // output: alpha,beta,gamma,delta

var array3:Array = array1.concat(array2);
trace(array3); // output: alpha,beta,alpha,beta,gamma,delta

var array4:Array = array3.slice(2,5);
trace(array4); // output: alpha,beta,gamma
```

È possibile utilizzare i metodi `join()` e `toString()` per eseguire query all'interno dell'array e restituire il contenuto sotto forma di stringa. Se non vengono utilizzati parametri per il metodo `join()`, i due metodi si comportano in maniera identica, essi restituiscono una stringa contenente un elenco delimitato da virgole di tutti gli elementi dell'array. Il metodo `join()`, a differenza del metodo `toString()`, accetta un parametro denominato `delimiter`, che consente di selezionare il simbolo da usare come separatore tra ogni elemento della stringa restituita.

Nell'esempio seguente viene creato un array denominato `rivers` e vengono chiamati sia il metodo `join()` che il metodo `toString()` per restituire i valori dell'array sotto forma di stringa. Il metodo `toString()` viene impiegato per restituire valori separati da virgola (`riverCSV`), mentre il metodo `join()` viene usato per restituire valori separati dal carattere +.

```
var rivers:Array = ["Nile", "Amazon", "Yangtze", "Mississippi"];
var riverCSV:String = rivers.toString();
trace(riverCSV); // output: Nile,Amazon,Yangtze,Mississippi
var riverPSV:String = rivers.join("+");
trace(riverPSV); // output: Nile+Amazon+Yangtze+Mississippi
```

Quando si usa il metodo `join()`, tenere presente che gli array nidificati vengono sempre restituiti con valori separati da virgola, a prescindere dal tipo di separatore specificato per gli elementi dell'array principale, come illustrato nell'esempio seguente:

```
var nested:Array = ["b","c","d"];
var letters:Array = ["a",nested,"e"];
var joined:String = letters.join("+");
trace(joined); // output: a+b,c,d+e
```

Array associativi

Gli array associativi, talvolta chiamati anche *hash* o *mappe*, impiegano *chiavi* anziché indici numerici per organizzare i valori memorizzati. Ogni chiave di un array associativo è una stringa univoca che consente di accedere a un valore memorizzato. Gli array associativi sono istanze della classe Object, di conseguenza, ogni chiave corrisponde a un nome di proprietà. Gli array associativi sono insiemi di coppie chiave/valore non ordinati. Le chiavi di un array associativo non sono mai in un ordine specifico.

ActionScript 3.0 introduce un tipo di array associativo avanzato denominato *dizionario*. I dizionari, che sono istanze della classe Dictionary nel pacchetto flash.utils, impiegano chiavi di qualsiasi tipo di dati, ma che sono in genere istanze della classe Object. In altre parole, le chiavi dei dizionari non soli limitate a valori di tipo String.

In questa sezione viene illustrato come creare array associativi che impiegano stringhe come chiavi e come usare la classe Dictionary.

Array associativi con chiavi stringa

In ActionScript 3.0 gli array associativi possono essere creati in due modi diversi. È possibile utilizzare la funzione di costruzione della classe Object, che presenta il vantaggio di consentire l'inizializzazione di un array con un valore letterale oggetto. Un'istanza della classe Object, detta anche *oggetto generico*, è identica dal punto di vista funzionale a un array associativo. Ogni nome di proprietà dell'oggetto generico funge da chiave che fornisce accesso a un valore memorizzato.

Nell'esempio seguente viene creato un array associativo denominato `monitorInfo` e viene utilizzato un valore letterale oggetto per inizializzare l'array con due coppie chiave/valore:

```
var monitorInfo:Object = {type:"Flat Panel", resolution:"1600 x 1200"};
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

Se non è necessario inizializzare l'array quando viene dichiarato, è possibile utilizzare la funzione di costruzione Object per creare l'array, come indicato di seguito:

```
var monitorInfo:Object = new Object();
```

Dopo che l'array viene creato utilizzando il valore letterale oggetto o la funzione di costruzione della classe Object, è possibile aggiungere nuovi valori all'array mediante l'operatore parentesi (`[]`) o l'operatore punto (`.`). Nell'esempio seguente vengono aggiunti due nuovi valori a `monitorArray`:

```
monitorInfo["aspect ratio"] = "16:10"; // Forma errata, non usare spazi
monitorInfo.colors = "16.7 million";
trace(monitorInfo["aspect ratio"], monitorInfo.colors);
// output: 16:10 16,7 milioni
```

Si noti come la chiave denominata `aspect ratio` contenga un carattere spazio. Questa operazione è possibile con l'operatore parentesi quadra, ma genera un errore se si tenta di effettuarla con l'operatore punto. L'uso di spazi nei nomi delle chiavi è sconsigliato.

In alternativa, per creare un array associativo è possibile utilizzare la funzione di costruzione Array e quindi l'operatore parentesi quadra (`[]`) o l'operatore punto (`.`) per aggiungere coppie chiave/valore all'array. Se si dichiara un array associativo di tipo Array, non è possibile utilizzare un valore letterale oggetto per inizializzare l'array. L'esempio seguente crea un array associativo denominato `monitorInfo` utilizzando la funzione di costruzione Array e aggiunge una chiave denominata `type` e una denominata `resolution` con i relativi valori.

```
var monitorInfo:Array = new Array();
monitorInfo["type"] = "Flat Panel";
monitorInfo["resolution"] = "1600 x 1200";
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

L'uso della funzione di costruzione Array per creare un array associativo non presenta vantaggi. Non è possibile usare la proprietà `Array.length` né alcuno dei metodi della classe Array con gli array associativi, anche se si usa la funzione di costruzione di Array o il tipo di dati di Array. L'impiego della funzione di costruzione della classe Array è meglio destinarlo alla creazione di array indicizzati.

Array associativi con chiavi oggetto

La classe Dictionary consente di creare array associativi che impiegano oggetti anziché stringhe come chiavi. Tali array vengono anche chiamati dizionari, hash o mappe.
Ad esempio, si consideri un'applicazione che consente di determinare la posizione di un oggetto Sprite in base alla sua associazione a un contenitore specifico. È possibile utilizzare un oggetto Dictionary per mappare ogni oggetto Sprite a un contenitore.

Nel codice seguente vengono create tre istanze della classe Sprite che servono da chiave per l'oggetto Dictionary. A ogni chiave viene assegnato un valore di `GroupA` o `GroupB`. Tali valori possono essere dati di qualsiasi tipo, tuttavia, nell'esempio, sia `GroupA` che `GroupB` sono istanze della classe Object. Successivamente, è possibile accedere al valore associato a ciascuna chiave con l'operatore (`[]`) di accesso alla proprietà, come illustrato dal codice seguente:

```
import flash.display.Sprite;
import flash.utils.Dictionary;

var groupMap:Dictionary = new Dictionary();

// oggetti da usare come chiavi
var spr1:Sprite = new Sprite();
var spr2:Sprite = new Sprite();
var spr3:Sprite = new Sprite();

// oggetti da usare come valori
var groupA:Object = new Object();
var groupB:Object = new Object();

// Crea una nuova coppia chiave/valore nel dizionario.
groupMap[spr1] = groupA;
groupMap[spr2] = groupB;
groupMap[spr3] = groupB;

if (groupMap[spr1] == groupA)
{
  trace("spr1 is in groupA");
}
```

```
if (groupMap[spr2] == groupB)
{
  trace("spr2 is in groupB");
}
if (groupMap[spr3] == groupB)
{
  trace("spr3 is in groupB");
}
```

Iterazioni con chiavi oggetto

È possibile eseguire iterazioni all'interno del contenuto di un oggetto Dictionary mediante le funzioni cicliche `for..in` o `for each..in`. Il ciclo `for..in` consente di eseguire iterazioni in base a chiavi, mentre il ciclo `for each..in` consente di eseguire iterazioni in base ai valori associati alle chiavi.

Utilizzare il ciclo `for..in` per accedere direttamente alle chiavi oggetto di un oggetto Dictionary. È inoltre possibile accedere ai valori dell'oggetto Dictionary con l'operatore di accesso alle proprietà (`[]`). Nel codice seguente viene utilizzato l'esempio precedente del dizionario `groupMap` per illustrare come eseguire un'iterazione in un oggetto Dictionary con il ciclo `for..in`:

```
for (var key:Object in groupMap)
{
  trace(key, groupMap[key]);
}
/* output:
[object Sprite] [object Object]
[object Sprite] [object Object]
[object Sprite] [object Object]
*/
```

Utilizzare il ciclo `for each..in` per accedere direttamente ai valori di un oggetto Dictionary. Anche nel codice seguente viene utilizzato il dizionario `groupMap` per illustrare come eseguire un'iterazione in un oggetto Dictionary con il ciclo `for each..in`:

```
for each (var item:Object in groupMap)
{
  trace(item);
}
/* output:
[oggetto Object]
[oggetto Object]
[oggetto Object]
*/
```

Chiavi oggetto e gestione della memoria

Adobe Flash Player impiega un sistema di garbage collection che consente di ripristinare memoria che non viene più utilizzata. Se un oggetto non presenta riferimenti associati, esso diventa idoneo per il processo di garbage collection e la memoria liberata viene ripristinata alla successiva esecuzione del sistema di garbage collection. Ad esempio, il codice seguente consente di creare un nuovo oggetto e di assegnare un riferimento a tale oggetto mediante la variabile `myObject`:

```
var myObject:Object = new Object();
```

Fintanto che esiste un riferimento all'oggetto, il sistema di garbage collection non ripristinerà la memoria occupata dall'oggetto. Se il valore di `myObject` viene modificato in modo da puntare a un oggetto differente o viene impostato sul valore `null`, la memoria occupata dall'oggetto originale diventa disponibile per la garbage collection, ma solo se non vi sono altri riferimenti che puntano all'oggetto originale.

Se si usa `myObject` come chiave in un oggetto Dictionary, si crea un altro riferimento all'oggetto originale. Ad esempio, il codice seguente consente di creare due riferimenti a un oggetto: la variabile `myObject` e la chiave nell'oggetto `myMap`:

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary();
myMap[myObject] = "foo";
```

Per fare in modo che l'oggetto a cui fa riferimento `myObject` diventi disponibile per la garbage collection, è necessario rimuovere tutti i riferimenti. In questo caso, è necessario modificare il valore di `myObject` ed eliminare la chiave `myObject` da `myMap`, come illustrato nel codice seguente:

```
myObject = null;
delete myMap[myObject];
```

In alternativa, è possibile utilizzare il parametro `useWeakReference` della funzione di costruzione Dictionary per rendere tutte le chiavi del dizionario *riferimenti deboli*. Il sistema di garbage collection ignora i riferimenti deboli, vale a dire che un oggetto che presenta solo riferimenti deboli è disponibile per la garbage collection. Ad esempio, nel codice seguente, non è necessario eliminare la chiave `myObject` da `myMap` per rendere l'oggetto disponibile per la garbage collection:

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary(true);
myMap[myObject] = "foo";
myObject = null; // Rende l'oggetto disponibile per il processo
                 // di garbage collection.
```

Array multidimensionali

Gli array multidimensionali contengono altri array come elementi. Ad esempio, si prenda in considerazione, un elenco di attività memorizzato come array indicizzato di stringhe:

```
var tasks:Array = ["wash dishes", "take out trash"];
```

Se si desidera memorizzare un elenco separato di attività per ogni giorno della settimana, è possibile creare un array multidimensionale con un elemento per ogni giorno della settimana. Ogni elemento contiene un array indicizzato simile all'array `tasks`, che a sua volta contiene l'elenco di attività. Negli array multidimensionali è possibile utilizzare qualsiasi combinazione di array indicizzati o associativi. Gli esempi nelle sezioni che seguono impiegano due array indicizzati o un array associativo di array indicizzati. È possibile provare altre combinazioni come esercizio.

Due array indicizzati

Se si usano due array indicizzati, è possibile visualizzare il risultato sotto forma di tabella o foglio di calcolo. Gli elementi del primo array rappresentano le righe della tabella, mentre gli elementi del secondo array rappresentano le colonne.

Ad esempio, l'array multidimensionale seguente impiega due array indicizzati per tenere traccia degli elenchi delle attività di ogni giorno della settimana. Il primo array, `masterTaskList`, viene creato mediante la funzione di costruzione della classe Array. Ogni elemento dell'array rappresenta un giorno della settimana, con indice 0 in corrispondenza di lunedì e indice 6 in corrispondenza di domenica. Tali elementi possono essere considerati come le righe della tabella. Per creare l'elenco delle attività di ogni giorno della settimana è possibile assegnare un valore letterale array a ciascuno dei sette elementi creati nell'array `masterTaskList`. I valori letterali di array rappresentano le colonne nella tabella.

```
var masterTaskList:Array = new Array();
masterTaskList[0] = ["wash dishes", "take out trash"];
masterTaskList[1] = ["wash dishes", "pay bills"];
masterTaskList[2] = ["wash dishes", "dentist", "wash dog"];
masterTaskList[3] = ["wash dishes"];
masterTaskList[4] = ["wash dishes", "clean house"];
masterTaskList[5] = ["wash dishes", "wash car", "pay rent"];
masterTaskList[6] = ["mow lawn", "fix chair"];
```

È possibile accedere a singole voci di qualsiasi elenco di attività mediante l'operatore parentesi. La prima serie di parentesi rappresenta il giorno della settimana, mentre la seconda serie rappresenta l'attività in elenco per tale giorno. Ad esempio, per recuperare la seconda attività dell'elenco di mercoledì, utilizzare in primo luogo l'indice 2 per mercoledì, quindi l'indice 1 per la seconda attività in elenco.

```
trace(masterTaskList[2][1]); // output: dentist
```

Per recuperare la prima attività dell'elenco di domenica, utilizzare l'indice 6 per domenica e l'indice 0 per la prima attività in elenco.

```
trace(masterTaskList[6][0]); // output: mow lawn
```

Array associativo con un array indicizzato

Per rendere più facilmente accessibili i singoli array, è possibile utilizzare un array associativo per i giorni della settimana e un array indicizzato per l'elenco delle attività. L'impiego di un array associativo consente di usare la sintassi del punto per fare riferimento a un particolare giorno della settimana; tuttavia, in tal caso è necessario un elaborazione extra in fase di runtime per accedere a ciascun elemento dell'array associativo. Nell'esempio seguente viene utilizzato un array associativo come base di un elenco attività, con una coppia chiave/valore per ogni giorno della settimana:

```
var masterTaskList:Object = new Object();
masterTaskList["Monday"] = ["wash dishes", "take out trash"];
masterTaskList["Tuesday"] = ["wash dishes", "pay bills"];
masterTaskList["Wednesday"] = ["wash dishes", "dentist", "wash dog"];
masterTaskList["Thursday"] = ["wash dishes"];
masterTaskList["Friday"] = ["wash dishes", "clean house"];
masterTaskList["Saturday"] = ["wash dishes", "wash car", "pay rent"];
masterTaskList["Sunday"] = ["mow lawn", "fix chair"];
```

La sintassi del punto rende il codice più facilmente leggibile, consentendo di evitare varie serie di parentesi.

```
trace(masterTaskList.Wednesday[1]); // output: dentist
trace(masterTaskList.Sunday[0]);    // output: mow lawn
```

È possibile eseguire un'iterazione attraverso l'elenco delle attività mediante un ciclo `for..in`, tuttavia, in tal caso è necessario usare le parentesi anziché la sintassi del punto per accedere al valore associato a ogni chiave. Poiché `masterTaskList` è un array associativo, gli elementi non vengono necessariamente recuperati nell'ordine che ci si attende, come illustrato dall'esempio seguente:

```
for (var day:String in masterTaskList)
{
  trace(day + ": " + masterTaskList[day])
}
/* output:
Sunday: mow lawn,fix chair
Wednesday: wash dishes,dentist,wash dog
Friday: wash dishes,clean house
Thursday: wash dishes
Monday: wash dishes,take out trash
Saturday: wash dishes,wash car,pay rent
Tuesday: wash dishes,pay bills
*/
```

Clonazione di array

La classe Array non presenta un metodo integrato per eseguire copie degli array. È possibile creare una *copia superficiale* di un array chiamando il metodo `concat()` o `slice()` senza argomenti. In una copia superficiale, se l'array originale presenta elementi che sono oggetti, solo i riferimenti agli oggetti vengono copiati e non gli oggetti stessi. La copia punta agli stessi oggetti dell'originale. Eventuali modifiche apportate agli oggetti vengono riprodotte in entrambi gli array.

In una *copia profonda*, tutti gli oggetti rilevati nell'array originale sono copiati in modo che il nuovo array non punti agli stessi oggetti dell'array originale. Per eseguire copie profonde è necessaria più di una singola riga di codice, che in genere richiede la creazione di una funzione. Tale funzione può essere creata come una funzione di utilità generale o come un metodo di una sottoclasse di Array.

Nell'esempio seguente viene definita una funzione denominata `clone()` che esegue una copia profonda. L'algoritmo viene preso a prestito da una tecnica di programmazione Java comune. La funzione crea una copia profonda serializzando l'array in un'istanza della classe ByteArray, quindi rileggendo l'array in un nuovo array. Tale funzione accetta un oggetto per essere utilizzato sia con array indicizzati che associativi, come illustrato dal codice seguente:

```
import flash.utils.ByteArray;

function clone(source:Object):*
{
  var myBA:ByteArray = new ByteArray();
  myBA.writeObject(source);
  myBA.position = 0;
  return(myBA.readObject());
}
```

Argomenti avanzati

Estensione della classe Array

La classe Array è una delle poche classi principali che non è finale, vale a dire che è possibile creare sottoclassi della classe Array. In questa sezione viene fornito un esempio che illustra come creare una sottoclasse di Array e vengono discussi alcuni dei problemi che potrebbero verificarsi durante tale procedura.

Come accennato precedentemente, gli array in ActionScript non sono tipizzati, tuttavia, è possibile creare sottoclassi di Array che accettano solo elementi di uno specifico tipo di dati. Nell'esempio delle sezioni che seguono viene definita una sottoclasse di Array denominata TypedArray che limita i suoi elementi a valori del tipo di dati specificato nel primo parametro. La classe TypedArray viene presentata solo come esempio di come è possibile estendere la classe Array e non è idonea a scopi produttivi per varie ragioni. In primo luogo, la verifica dei tipi viene eseguita in fase di runtime anziché in fase di compilazione. Secondo, quando un metodo TypedArray rileva un'errata corrispondenza, questa viene ignorata e non viene generata alcuna eccezione, anche se i metodi possono essere facilmente modificati per generare eccezioni. Terzo, la classe non è in grado di impedire l'uso dell'operatore di accesso all'array per l'inserimento di valori di qualunque tipo nell'array. Quarto, lo stile di codifica favorisce la semplicità rispetto all'ottimizzazione delle prestazioni.

Dichiarazione di una sottoclasse

Utilizzare la parola chiave `extends` per indicare che una classe è una sottoclasse di Array. Le sottoclassi di Array devono usare l'attributo `dynamic`, esattamente come la classe Array, altrimenti, non funzionano correttamente.

Il codice seguente illustra la definizione della classe TypedArray, contenente una costante per la memorizzazione del tipo di dati, un metodo della funzione di costruzione e i quattro metodi che consentono di aggiungere elementi all'array. Nell'esempio viene omesso il codice di ciascun metodo, che però verrà illustrato dettagliatamente nelle sezioni che seguono:

```
public dynamic class TypedArray extends Array
{
  private const dataType:Class;

  public function TypedArray(...args) {}

  AS3 override function concat(...args):Array {}

  AS3 override function push(...args):uint {}

  AS3 override function splice(...args) {}

  AS3 override function unshift(...args):uint {}
}
```

Tutti e quattro i metodi sostituiti impiegano lo spazio dei nomi di AS3 anziché l'attributo `public` perché in questo esempio si presuppone che l'opzione del compilatore `-as3` sia impostata su `true` e l'opzione del compilatore `-es` sia impostata su `false`. Queste sono le impostazioni predefinite per Adobe Flex Builder 2 e per Adobe Flash CS3 Professional. Per ulteriori informazioni, vedere "Spazio dei nomi di AS3" a pagina 192.

SUGGERIMENTO

Gli sviluppatori avanzati che preferiscono utilizzare l'ereditarietà di prototipi, possono apportare due piccole modifiche alla classe TypedArray, affinché venga compilata con l'opzione del compilatore `-es` impostata su `true`. In primo luogo, è necessario rimuovere tutte le occorrenze dell'attributo `override` e sostituire lo spazio dei nomi di AS3 con l'attributo `public`. Quindi, è necessario sostituire `Array.prototype` per tutte le occorrenze di `super`.

Funzione di costruzione TypedArray

La funzione di costruzione della sottoclasse pone un interessante problema in quanto la funzione di costruzione deve accettare un elenco di argomenti di lunghezza arbitraria. Il problema consiste in come passare gli argomenti al supercostruttore per creare l'array. Se si trasmette l'elenco di argomenti come un array, il supercostruttore lo considera come un unico argomento di tipo Array e l'array risultante contiene sempre un solo elemento. Per gestire in modo tradizionale elenchi di argomenti da trasmettere è possibile usare il metodo `Function.apply()`, che considera un array di argomenti come suo secondo parametro, ma lo converte poi in un elenco di argomenti durante l'esecuzione della funzione. Sfortunatamente, il metodo `Function.apply()` non può essere utilizzato con funzioni di costruzione.

La sola opzione rimasta è quella di ricreare la logica della funzione di costruzione di Array nella funzione di costruzione TypedArray. Il codice seguente illustra l'algoritmo utilizzato nella funzione di costruzione della classe Array, che è possibile riutilizzare nella funzione di costruzione della sottoclasse di Array:

```
public dynamic class Array
{
  public function Array(...args)
  {
    var n:uint = args.length
    if (n == 1 && (args[0] is Number))
    {
      var dlen:Number = args[0];
      var ulen:uint = dlen;
      if (ulen != dlen)
      {
        throw new RangeError("Array index is not a 32-bit unsigned integer
  ("+dlen+")");
      }
      length = ulen;
    }
    else
    {
      length = n;
      for (var i:int=0; i < n; i++)
      {
        this[i] = args[i]
      }
    }
  }
}
```

La funzione di costruzione TypedArray condivide la maggior parte del codice con la funzione di costruzione Array, con solo qualche leggera differenza. In primo luogo, l'elenco dei parametri include un nuovo parametro richiesto di tipo Class che consente di specificare il tipo di dati dell'array. Secondo, il tipo di dati trasmessi alla funzione di costruzione viene assegnato alla variabile `dataType`. Terzo, nell'istruzione `else`, il valore della proprietà `length` viene assegnato dopo il ciclo `for`, affinché la proprietà `length` includa unicamente gli argomenti del tipo corretto. Quarto, il corpo del ciclo `for` impiega la versione sostituita del metodo `push()`, in tal modo solo gli argomenti del tipo di dati corretto vengono aggiunti all'array. Nell'esempio seguente è illustrata la funzione di costruzione TypedArray:

```
public dynamic class TypedArray extends Array
{
  private var dataType:Class;
  public function TypedArray(typeParam:Class, ...args)
  {
    dataType = typeParam;
    var n:uint = args.length
    if (n == 1 && (args[0] is Number))
    {
      var dlen:Number = args[0];
      var ulen:uint = dlen
      if (ulen != dlen)
      {
        throw new RangeError("Array index is not a 32-bit unsigned integer
  ("+dlen+")")
      }
      length = ulen;
    }
    else
    {
      for (var i:int=0; i < n; i++)
      {
        // controllo tipo eseguito in push()
        this.push(args[i])
      }
      length = this.length;
    }
  }
}
```

Metodi sostituiti di TypedArray

La classe TypedArray sostituisce i quattro metodi della classe Array che consentono di aggiungere elementi a un array. Ognuno dei metodi sostituiti permette di aggiungere un controllo di tipo che impedisce l'inserimento di elementi appartenenti a un tipo di dati non corretto. Quindi, ogni metodo richiama la versione di se stesso della superclasse.

Il metodo `push()` esegue iterazioni all'interno dell'elenco degli argomenti con una funzione ciclica `for..in` ed effettua un controllo di tipo per ogni argomento. Se vengono rilevati argomenti di tipo non corretto, essi vengono rimossi dall'array `args` mediante il metodo `splice()`. Al termine della funzione ciclica `for..in`, l'array `args` conterrà valori solo di tipo `dataType`. La versione della superclasse del metodo `push()` viene quindi chiamata con l'array `args` aggiornato, come illustrato dal codice seguente:

```
AS3 override function push(...args):uint
{
  for (var i:* in args)
  {
    if (!(args[i] is dataType))
    {
      args.splice(i,1);
    }
  }
  return (super.push.apply(this, args));
}
```

Il metodo `concat()` consente di creare un array TypedArray temporaneo chiamato `passArgs` in cui vengono memorizzati gli argomenti che superano il controllo di tipo. Ciò consente di riutilizzare il codice per il controllo di tipo già esistente nel metodo `push()`. Una funzione ciclica `for..in` esegue un'iterazione attraverso l'array `args` e chiama il metodo `push()` su ciascun argomento. Poiché `passArgs` è tipizzato come TypedArray, viene eseguita la versione TypedArray di `push()`. Il metodo `concat()` chiama quindi la versione di se stesso della superclasse, come illustrato nel codice seguente:

```
AS3 override function concat(...args):Array
{
  var passArgs:TypedArray = new TypedArray(dataType);
  for (var i:* in args)
  {
    // controllo tipo eseguito in push()
    passArgs.push(args[i]);
  }
  return (super.concat.apply(this, passArgs));
}
```

Il metodo `splice()` impiega un elenco arbitrario di argomenti, tuttavia, i primi due argomenti fanno sempre riferimento a un numero di indice e al numero di elementi da eliminare. Ecco perché il metodo `splice()` sostituito esegue il controllo di tipo unicamente per gli elementi dell'array `args` nella posizione di indice 2 o superiore. Un aspetto interessante del codice è che sembra che all'interno della funzione ciclica `for` si verifichi una chiamata ricorsiva al metodo `splice()`; in realtà, non si tratta di una chiamata ricorsiva in quanto l'array`args` è di tipo Array e non di tipo TypedArray, quindi la chiamata ad `args.splice()` è una chiamata alla versione del metodo della superclasse. Una volta conclusa la funzione ciclica `for..in`, l'array `args` conterrà unicamente valori di tipo corretto in posizione di indice 2 o superiore e il metodo `splice()` chiamerà la versione di se stesso della superclasse, come illustrato nel codice seguente:

```
  AS3 override function splice(...args):*
  {
    if (args.length > 2)
    {
      for (var i:int=2; i< args.length; i++)
      {
        if (!(args[i] is dataType))
        {
          args.splice(i,1);
        }
      }
    }
    return (super.splice.apply(this, args));
  }
```

Anche il metodo `unshift()`, che consente di aggiungere elementi all'inizio di un array, accetta un elenco arbitrario di argomenti. Il metodo sostituito `unshift()` impiega un algoritmo molto simile a quello usato dal metodo `push()`, come illustrato nell'esempio di codice seguente:

```
  AS3 override function unshift(...args):uint
  {
    for (var i:* in args)
    {
      if (!(args[i] is dataType))
      {
        args.splice(i,1);
      }
    }
    return (super.unshift.apply(this, args));
  }
}
```

Esempio: PlayList

L'esempio PlayList illustra alcune tecniche di lavoro con gli array, nel contesto di un'applicazione per la gestione di sequenze di brani musicali. Le tecniche sono descritte di seguito.

- Creazione di un array indicizzato
- Aggiunta di elementi a un array indicizzato
- Ordinamento di un array di oggetti in base a varie proprietà, utilizzando varie opzioni di ordinamento
- Conversione di un array in una stringa delimitata da caratteri

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione PlayList si trovano nella cartella Samples/PlayList. L'applicazione è costituita dai seguenti file:

File	Descrizione
PlayList.mxml o PlayList.fla	Il file principale dell'applicazione in Flash (FLA) o Flex (MXML)
com/example/programmingas3/playlist/Song.as	Oggetto valore rappresentante informazioni su un singolo brano. Le voci gestite dalla classe PlayList sono istanze di Song.
com/example/programmingas3/playlist/SortProperty.as	Pseudo-enumerazione i cui valori disponibili rappresentano le proprietà della classe Song e in base alla quale è possibile ordinare un elenco di oggetti Song.

Panoramica della classe PlayList

La classe PlayList consente di gestire una serie di oggetti Song. Essa presenta una serie di metodi pubblici con funzionalità per l'aggiunta di brani all'elenco di riproduzione (metodo `addSong()`) e per l'ordinamento dei brani nell'elenco (metodo `sortList()`). Inoltre, la classe include una proprietà accessoria di sola lettura, `songList`, che consente l'accesso alla serie di brani effettivamente presenti nell'elenco di riproduzione. Internamente, la classe PlayList tiene traccia dei brani mediante una variabile privata di Array:

```
public class PlayList
{
  private var _songs:Array;
  private var _currentSort:SortProperty = null;
  private var _needToSort:Boolean = false;
  ...
}
```

Oltre alla variabile di Array _songs, utilizzata dalla classe PlayList per tenere traccia dell'elenco dei brani, vi sono altre due variabili private che segnalano se l'elenco deve essere ordinato o meno (_needToSort) e in base a quale proprietà l'elenco viene aggiornato in un dato momento (_currentSort).

Come avviene per tutti gli oggetti, la dichiarazione di un'istanza di Array è solo metà del lavoro di creazione di un Array. Prima di accedere alle proprietà o ai metodi di un'istanza di Array, è necessario creare l'istanza nella funzione di costruzione della classe PlayList.

```
public function PlayList()
{
  this._songs = new Array();
  // Imposta l'ordinamento iniziale.
  this.sortList(SortProperty.TITLE);
}
```

La prima riga della funzione di costruzione crea un'istanza della variabile _songs pronta per essere utilizzata. Inoltre, il metodo sortList() viene chiamato per impostare la proprietà iniziale in base alla quale effettuare l'ordinamento.

Aggiunta di un brano all'elenco

Se un utente aggiunge una nuova canzone nell'applicazione, il codice nel modulo di immissione dati richiama il metodo addSong() della classe PlayList.

```
/**
 * Aggiunge un brano all'elenco di riproduzione.
 */
public function addSong(song:Song):void
{
  this._songs.push(song);
  this._needToSort = true;
}
```

All'interno di addSong(), viene chiamato il metodo push() dell'array _songs e il nuovo oggetto Song trasmesso ad addSong() come nuovo elemento viene inserito nell'array. Con il metodo push(), il nuovo elemento viene aggiunto alla fine dell'array, a prescindere da qualsiasi ordinamento precedentemente applicato. Di conseguenza, dopo la chiamata al metodo push(), è possibile che l'elenco dei brani non sia più ordinato correttamente e che quindi la variabile _needToSort sia impostata su true. In teoria, il metodo sortList() potrebbe essere chiamato immediatamente, per eliminare la necessità di rilevare se l'elenco è stato o meno ordinato nel momento specificato. In pratica, tuttavia, è necessario che l'elenco dei brani venga ordinato solo immediatamente prima del suo recupero. Posticipando l'operazione di ordinamento, l'applicazione non esegue infatti gli ordinamenti non necessari, nel caso, ad esempio, vengano aggiunti vari brani all'elenco prima del suo recupero.

Ordinamento dell'elenco di brani

Poiché le istanze di Song gestite dall'elenco di riproduzione sono oggetti complessi, gli utenti dell'applicazione possono ordinare l'elenco in base a varie proprietà, quali titolo del brano o anno di pubblicazione. Nell'applicazione PlayList, l'attività di ordinamento dell'elenco dei brani è suddiviso in tre fasi: identificazione della proprietà in base alla quale l'elenco deve essere ordinato, indicazione delle opzioni di ordinamento da utilizzare ed esecuzione dell'operazione di ordinamento vera e propria.

Proprietà di ordinamento

Un oggetto Song è in grado di tenere traccia di varie proprietà, incluso titolo del brano, artista, anno di pubblicazione, nome del file e una serie di generi selezionati dall'utente associati a ciascun brano. Tra queste proprietà, solo le prime tre sono utili per l'ordinamento. Per ragioni di praticità, l'esempio include la classe SortProperty, che funge da elenco di valori che rappresentano le proprietà disponibili per eseguire l'ordinamento.

```
public static const TITLE:SortProperty = new SortProperty("title");
public static const ARTIST:SortProperty = new SortProperty("artist");
public static const YEAR:SortProperty = new SortProperty("year");
```

La classe SortProperty contiene tre costanti, `TITLE`, `ARTIST` e `YEAR`, ognuna delle quali contiene una stringa con il nome effettivo della proprietà della classe Song associata utilizzabile per l'ordinamento. Nella parte restante del codice, per indicare una proprietà di ordinamento viene utilizzato un membro dell'enumerazione. Ad esempio, nella funzione di costruzione PlayList, l'elenco viene ordinato chiamando il metodo `sortList()`, come indicato di seguito:

```
// Imposta l'ordinamento iniziale.
this.sortList(SortProperty.TITLE);
```

Poiché la proprietà in base alla quale eseguire l'ordinamento viene specificata come `SortProperty.TITLE`, i brani vengono ordinati in base al titolo.

Ordinamento in base a una proprietà e impostazione delle opzioni di ordinamento

Il lavoro di effettivo ordinamento dell'elenco dei brani viene eseguito dalla classe PlayList mediante il metodo `sortList()`, come indicato di seguito:

```
/**
 * Ordina l'elenco dei brani in base alla proprietà specificata.
 */
public function sortList(sortProperty:SortProperty):void
{
  ...
  var sortOptions:uint;
  switch (sortProperty)
  {
    case SortProperty.TITLE:
      sortOptions = Array.CASEINSENSITIVE;
      break;
    case SortProperty.ARTIST:
      sortOptions = Array.CASEINSENSITIVE;
      break;
    case SortProperty.YEAR:
      sortOptions = Array.NUMERIC;
      break;
  }

  // Esegue l'ordinamento effettivo dei dati.
  this._songs.sortOn(sortProperty.propertyName, sortOptions);

  // Salva la proprietà di ordinamento corrente.
  this._currentSort = sortProperty;

  // Registra che l'elenco è stato ordinato.
  this._needToSort = false;
}
```

Se si effettua l'ordinamento in base al titolo o all'artista, è possibile applicare un ordine alfabetico, mentre se si utilizza l'anno di pubblicazione, è più logico applicare un ordine numerico. L'istruzione `switch` viene utilizzata per definire l'opzione di ordinamento appropriata, memorizzata nella variabile `sortOptions`, in base al valore specificato nel parametro `sortProperty`. Anche in questo caso, per distinguere le varie proprietà vengono utilizzati i membri dell'enumerazione con nome anziché valori hard-coded.

Una volta impostati proprietà e opzioni di ordinamento, l'array `_songs` viene ordinato chiamando il metodo `sortOn()` e inviando tali valori come parametri. La proprietà di ordinamento corrente viene registrata, così come l'avvenuto ordinamento dell'elenco dei brani.

Combinazione di elementi di array in stringhe delimitate da caratteri

Oltre a essere utilizzati per gestire l'elenco dei brani nella classe PlayList, in questo esempio gli array vengono anche impiegati nella classe Song per la gestione degli elenchi dei generi da associare ai singoli brani. Si consideri il seguente snippet di codice tratto dalla definizione della classe Song:

```
private var _genres:String;

public function Song(title:String, artist:String, year:uint,
  filename:String, genres:Array)
{
  ...
  // I generi vengono trasmessi come un array,
  // ma memorizzati come stringhe separate de punto e virgola.
  this._genres = genres.join(";");
}
```

Quando si crea una nuova istanza di Song, il parametro `genres` utilizzato per specificare il genere (o i generi) a cui il brano appartiene viene definito come un'istanza di Array. Ciò consente di raggruppare facilmente più generi in una singola variabile da inviare alla funzione di costruzione. Tuttavia, internamente, la classe Song conserva i generi nella variabile privata `_genres` come un'istanza di String separata da punto e virgola. Il parametro Array viene convertito in una stringa separata da punto e virgola mediante chiamata al metodo `join()` con il valore letterale di stringa `";"` come delimitatore specificato.

Grazie allo stesso token, gli accessor `genres` consentono l'impostazione o il recupero dei generi come un array:

```
  public function get genres():Array
  {
    // I generi sono memorizzati come stringhe separate da punto e virgola,
    // di conseguenza, è necessario trasformarle in un Array per
  ritrasmetterle indietro.
    return this._genres.split(";");
  }
  public function set genres(value:Array):void
  {
    // I generi vengono trasmessi come un array,
    // ma memorizzati come stringhe separate da punto e virgola.
    this._genres = value.join(";");
  }
```

L'accessor `genres set` si comporta esattamente come la funzione di costruzione, in quanto accetta un Array e chiama il metodo `join()` per convertirlo in una stringa separata da punto e virgola. L'accessor `get` esegue l'operazione opposta: viene chiamato il metodo `split()` della variabile `_genres`, per suddividere la stringa in una array di valori mediante il delimitatore specificato (il valore letterale di stringa `";"` come sopra).

CAPITOLO 8

Gestione degli errori

8

La “gestione” degli errori si esegue aggiungendo una logica all’applicazione in grado di rispondere a o risolvere gli errori generati durante la compilazione dell’applicazione o durante la sua esecuzione. Se l’applicazione gestisce gli errori, quando ne incontra uno *attiva una risposta*; se la gestione degli errori non è attivata, invece, il processo che ha generato l’errore ha esito negativo ma l’utente non ne viene informato. Se usata correttamente, la gestione degli errori mette al riparo l’applicazione stessa e gli utenti da situazioni impreviste.

Tuttavia, la gestione degli errori riguarda un ambito di attività molto ampio che comprende errori generati durante la compilazione, ma anche errori in fase di runtime. Questo capitolo affronta la gestione degli errori generati in fase di runtime, descrive i diversi tipi di errore possibili e illustra i vantaggi offerti dal nuovo sistema di gestione degli errori di ActionScript 3.0. Inoltre, spiega come implementare le proprie strategie di gestione degli errori all’interno delle applicazioni create.

Sommario

Elementi fondamentali della gestione degli errori

Introduzione alla gestione degli errori

Un errore in fase di runtime è un problema presente nel codice ActionScript, che impedisce l'esecuzione del contenuto ActionScript da parte di Adobe Flash Player. Per fare in modo che un'applicazione ActionScript venga eseguita senza intoppi, è necessario prevedere al suo interno del codice in grado di gestire gli errori, di correggerli, di escogitare delle soluzioni alternative o perlomeno di comunicare agli utenti che si è verificato un problema. Questo processo è denominato *gestione degli errori*.

La gestione degli errori riguarda un ambito di attività molto ampio che comprende errori generati durante la compilazione, ma anche errori in fase di runtime. Gli errori generati durante la compilazione sono facili da rilevare e devono obbligatoriamente essere risolti per consentire la creazione del file SWF. Il presente capitolo non prende in considerazione gli errori di compilazione. Per ulteriori informazioni su come scrivere del codice privo di errori di compilazione, vedere Capitolo 3, "Linguaggio e sintassi ActionScript" a pagina 71 e Capitolo 4, "Programmazione orientata agli oggetti con ActionScript" a pagina 147. Il presente capitolo si concentra sugli errori in fase di runtime.

Gli errori in fase di runtime sono più difficili da rilevare perché si manifestano solo nel momento in cui si esegue il codice. Se un segmento del proprio programma prevede vari rami di codice, come un'istruzione `if..then..else`, per accertarsi che il codice non contenga degli errori è indispensabile testare ogni possibile condizione usando tutti i valori possibili che gli utenti potrebbero specificare.

Gli errori in fase di runtime possono appartenere a due categorie: gli *errori di programmazione* sono errori del codice ActionScript, quello che succede se si specifica un tipo di dati errato per il parametro di un metodo; gli *errori logici* sono errori nella logica (la verifica dei dati e la manipolazione dei valori) del programma, ad esempio se si utilizza la formula sbagliata per calcolare i tassi di interesse in un'applicazione bancaria. Entrambi questi tipi di errore possono essere rilevati e corretti per tempo testando scrupolosamente l'applicazione.

Idealmente, sarebbe opportuno individuare ed eliminare tutti gli errori dell'applicazione prima di distribuirla agli utenti, però non è possibile prevedere né prevenire tutti gli errori. Ad esempio, supponiamo che un'applicazione ActionScript carichi dei dati da un sito Web sul quale non è possibile esercitare alcun controllo e che a un certo punto tale sito risulti inaccessibile. In questo caso, la porzione di applicazione che deve elaborare i dati esterni non potrà funzionare correttamente. L'aspetto più importante della gestione degli errori riguarda la gestione dei casi imprevisti in modo che gli utenti possano continuare a usare l'applicazione o che perlomeno vengano informati da un messaggio del motivo per cui l'applicazione non funziona correttamente.

Gli errori in fase di runtime sono rappresentati in due modi in ActionScript:

- Classi errore: molti errori sono associati a una classe errore. Quando si verifica un errore, Flash Player crea un'istanza della specifica classe associata all'errore. Il codice elabora una risposta all'errore utilizzando le informazioni contenute nell'oggetto errore.
- Eventi errore: a volte si verifica un errore quando Flash Player dovrebbe attivare un evento. In questi casi, Flash Player genera un evento errore in luogo dell'evento originario. Come tutti gli altri eventi, ogni evento errore è associato a una classe e Flash Player passa un'istanza di tale classe ai metodi che hanno effettuato il subscribing all'evento errore.

Per stabilire se un particolare metodo può attivare un errore o un evento errore, vedere l'argomento che tratta i metodi nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Operazioni comuni per la gestione degli errori

Le operazioni più comuni da intraprendere per eseguire la gestione degli errori nel codice sono le seguenti:

- Scrittura di codice per la gestione degli errori
- Ricerca, rilevamento e riproduzione degli errori
- Definizione di classi errore
- Risposta a eventi errore e a eventi basati sullo stato

Concetti e termini importanti

L'elenco seguente contiene i termini più importanti che vengono citati in questo capitolo:

- Asincrono: comando di un programma, come la chiamata a un metodo, che non genera un risultato immediato ma fornisce un risultato (o un errore) sotto forma di un evento.
- Cattura: quando si verifica un'eccezione (un errore in fase di runtime) e il codice la rileva, si dice che il codice *cattura* l'eccezione. Una volta catturata un'eccezione, Flash Player cessa di comunicare l'eccezione al resto del codice ActionScript.
- Versione debugger: versione speciale di Flash Player che contiene del codice che notifica gli utenti della presenza di errori runtime. La versione standard di Flash Player, che la maggior parte degli utenti possiede, ignora gli errori che il codice ActionScript non è in grado di gestire. La versione debugger, inclusa in Adobe Flash CS3 Professional e Adobe Flex, genera un messaggio di avvertenza quando si verifica un errore che non è in grado di gestire.
- Eccezione: errore che si verifica durante l'esecuzione del programma e che l'ambiente runtime (Flash Player) non è in grado di risolvere da solo.
- Riproduzione: quando il codice cattura un'eccezione, Flash Player cessa di notificare tali oggetti dell'eccezione. Se è importante che gli altri oggetti siano al corrente dell'eccezione, il codice deve *riprodurre* l'eccezione in modo che il processo di notifica ricominci.
- Sincrono: comando di un programma, come la chiamata a un metodo, che genera un risultato immediato (o che genera immediatamente un errore) e la cui risposta può essere utilizzata nello stesso blocco di codice.
- Generare: l'atto di comunicare a Flash Player (e, di conseguenza, di comunicare ad altri oggetti e al codice ActionScript) che si è verificato un errore viene comunemente definito *generazione* di un errore.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Tutti questi esempi includono la chiamata appropriata alla funzione `trace()`. Per provare gli esempi di codice contenuti in questo capitolo:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.
4. Eseguire il programma selezionando Controllo > Prova filmato.

   Nel pannello Output vengono visualizzati i risultati della funzione `trace()` dell'esempio di codice.

Alcuni degli esempi di codice che seguono sono più complessi e sono scritti come una classe. Per provare questi esempi:

1. Creare un documento Flash vuoto e salvarlo nel computer.
2. Creare un nuovo file ActionScript e salvarlo nella stessa directory del documento Flash. Il nome file deve corrispondere al nome della classe presente nell'esempio di codice. Ad esempio, se il codice definisce una classe chiamata ErrorTest, per salvare il file ActionScript, utilizzare il nome ErrorTest.as.
3. Copiare l'esempio di codice nel file ActionScript e salvare il file.
4. Nel documento Flash, fare clic in una parte vuota dello stage oppure dell'area di lavoro per attivare la finestra di ispezione Proprietà del documento.
5. Nella finestra di ispezione Proprietà, nel campo Classe documento inserire il nome della classe ActionScript copiata dal testo.
6. Eseguire il programma selezionando Controllo > Prova filmato.

   I risultati vengono visualizzati nel pannello Output (se l'esempio utilizza la funzione `trace()`) oppure in un campo di testo creato mediante il codice di esempio.

Per ulteriori informazioni su queste tecniche per la prova degli esempi di codice, vedere "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Tipi di errore

Durante lo sviluppo e l'esecuzione di applicazioni si incontrano diversi tipi di errore e vari termini relativi agli errori. Ecco un elenco dei principali tipi di errore e dei termini a essi relativi:

- Gli *errori in fase di compilazione* vengono generati dal compilatore ActionScript durante la compilazione del codice e si verificano se sono presenti dei problemi di sintassi che impediscono la compilazione del codice.
- Gli *errori in fase di runtime* si verificano durante l'esecuzione dell'applicazione già compilata e sono errori generati durante la riproduzione di un file SWF in Adobe Flash Player 9. Nella maggior parte dei casi è possibile gestirli man mano che si incontrano, comunicandoli all'utente e intervenendo in modo che l'applicazione non si interrompa. Se l'errore è irreversibile, ad esempio se non è possibile collegarsi a un sito web remoto o caricare dei dati necessari, la gestione degli errori permette di interrompere l'applicazione normalmente.

- Gli *errori sincroni* sono errori in fase di runtime che si verificano quando si richiama una funzione, ad esempio, quando si prova a utilizzare un particolare metodo passando un argomento non valido. In questo caso, Flash Player genera un'eccezione. La maggior parte degli errori sono di tipo sincrono, cioè avvengono nel momento in cui l'istruzione viene eseguita, e il flusso di controllo passa immediatamente all'istruzione `catch` più adeguata.

  La seguente porzione di codice, ad esempio, genera un errore runtime perché il metodo `browse()` non viene chiamato prima che il programma tenti di caricare un file:

  ```
  var fileRef:FileReference = new FileReference();
  try
  {
    fileRef.upload("http://www.yourdomain.com/fileupload.cfm");
  }
  catch (error:IllegalOperationError)
  {
    trace(error);
    // Error #2037: Funzioni chiamate in sequenza errata oppure la
    // chiamata precedente non è andata a buon fine.
  }
  ```

  In questo caso, l'errore in fase di runtime viene generato in modo sincrono perché Flash Player determina che il metodo `browse()` non è stato chiamato prima del tentativo di caricamento del file.

  Per informazioni dettagliate sulla gestione degli errori sincroni, vedere "Gestione degli errori sincroni delle applicazioni" a pagina 281.

- Gli *errori asincroni* sono errori in fase di runtime che si verificano in momenti diversi dell'esecuzione e che generano eventi rilevati da listener di eventi. In un'operazione asincrona, una funzione avvia un'operazione senza attenderne il completamento. Si può creare un listener di eventi errore che attenda una reazione da parte dell'applicazione o dell'utente e, se l'operazione non va a buon fine, si rileva l'errore mediante il listener e si risponde all'errore. A questo punto, il listener di eventi chiama una funzione di gestione eventi che risponda all'errore in modo utile, per esempio, visualizzando una finestra di dialogo con una richiesta di risolvere il problema diretta all'utente.

  Consideriamo nuovamente l'esempio di errore sincrono presentato prima. Se si esegue una chiamata corretta al metodo `browse()` prima di avviare il caricamento del file, Flash Player invia vari eventi. Ad esempio, all'avvio del caricamento, viene inviato l'evento `open`. Quando l'operazione di caricamento del file viene completata correttamente, viene inviato l'evento `complete`. Poiché la gestione degli eventi è un'attività asincrona, cioè che non avviene in momenti prestabiliti e conosciuti, è necessario usare il metodo `addEventListener()` per intercettare questi eventi specifici, come illustra il codice seguente:

  ```
  var fileRef:FileReference = new FileReference();
  fileRef.addEventListener(Event.SELECT, selectHandler);
  ```

```
fileRef.addEventListener(Event.OPEN, openHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();

function selectHandler(event:Event):void
{
  trace("...select...");
  var request:URLRequest = new URLRequest("http://www.yourdomain.com/
  fileupload.cfm");
  request.method = URLRequestMethod.POST;
  event.target.upload(request.url);
}
function openHandler(event:Event):void
{
  trace("...open...");
}
function completeHandler(event:Event):void
{
  trace("...complete...");
}
```

Per informazioni dettagliate sulla gestione degli errori asincroni, vedere "Risposte a eventi errore e a errori basati sullo stato" a pagina 288.

- Le *eccezioni non rilevate* sono errori generati a cui non corrisponde una logica (come un'istruzione `catch`) di risposta. Se l'applicazione genera un errore per il quale non esiste un'espressione `catch` appropriata o per il quale non è disponibile un gestore, né al livello attuale né al superiore, l'errore viene considerato un'eccezione non rilevata.

  In fase di runtime, Flash Player ignora intenzionalmente gli errori non rilevati e, se può, continua a riprodurre il file SWF perché gli utenti non possono necessariamente essere in grado di risolvere l'errore. Un errore non rilevato che passa sotto silenzio, cioè che viene ignorato, complica il lavoro delle applicazioni di debug. La versione debugger di Flash Player risponde agli errori non rilevati terminando lo script corrente e visualizzando l'errore non rilevato nell'istruzione `trace` o annotando il messaggio di errore nel file di registro. Se l'oggetto eccezione è un'istanza della classe Error o di una sua sottoclasse, viene richiamato il metodo `getStackTrace()` e le informazioni sulla traccia di stack vengono visualizzate anche nell'output dell'istruzione trace o nel file di registro. Per ulteriori informazioni sull'utilizzo della versione debugger di Flash Player, vedere "Utilizzo della versione debugger di Flash Player" a pagina 280.

Gestione degli errori in ActionScript 3.0

Dato che molte applicazioni possono essere eseguite senza logica di gestione degli errori, gli sviluppatori sono tentati di rimandare la creazione della logica di gestione degli errori all'interno delle loro applicazioni. Tuttavia, in assenza di un'adeguata procedura di gestione degli errori, è facile che un'applicazione si blocchi e non funzioni come previsto, con il risultato di esasperare l'utente. ActionScript 2.0 è dotato della classe Error che permette di integrare la logica nelle funzioni personalizzate in modo che generi un'eccezione accompagnata da un messaggio specifico. Poiché la gestione degli errori è un requisito fondamentale per sviluppare applicazioni di facile utilizzo da parte degli utenti, ActionScript 3.0 mette a disposizione un'architettura estesa per la rilevazione degli errori.

NOTA	Nonostante la *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0* sia in grado di documentare le eccezioni generate da vari metodi, potrebbe non essere possibile prevedere tutte le eccezioni possibili per ogni metodo. Un metodo potrebbe generare un'eccezione per errori di sintassi o altri problemi non esplicitamente associati alla descrizione del metodo, anche se la descrizione comprende alcune eccezioni generate dal metodo.

Elementi utilizzati da ActionScript 3.0 per la gestione degli errori

ActionScript 3.0 utilizza vari strumenti per la gestione degli errori:

- Classi Error. In conformità con la bozza della specifica del linguaggio ECMAScript (ECMA-262) Edizione 4, ActionScript 3.0 contiene un'ampia gamma di classi Error che permettono di espandere l'ambito delle situazioni suscettibili di generare errori. Ogni classe Error aiuta le applicazioni a gestire e a rispondere a condizioni di errore specifiche, siano esse relative a errori di sistema (come la condizione MemoryError), errori del codice (come la condizione ArgumentError), errori di rete e comunicazione (come la condizione URIError) o ad altre situazioni. Per ulteriori informazioni sulle classi, vedere "Confronto tra le classi di tipo Error" a pagina 292.
- Meno errori non catturati. Nelle versioni precedenti di Flash Player, gli errori venivano generati e comunicati solo se si utilizzava esplicitamente l'istruzione `throw`. In Flash Player 9, le proprietà e i metodi nativi di ActionScript generano errori in fase di runtime affinché l'utente possa gestire in modo più efficace ogni singola eccezione quando si verifica.

- Visualizzazione di messaggi di errore chiari durante le operazioni di debug. Quando si utilizza la versione debugger di Flash Player, situazioni o righe di codice problematiche generano messaggi di errore significativi che permettono di individuare facilmente il problema. La risoluzione degli errori risulta, di conseguenza, più rapida ed efficace. Per ulteriori informazioni, vedere "Utilizzo della versione debugger di Flash Player" a pagina 280.
- Indicazione precisa degli errori nei messaggi visualizzati in fase di runtime. Nelle versioni precedenti di Flash Player, il metodo `FileReference.upload()` restituiva il valore booleano `false` quando la chiamata al metodo `upload()` non andava a buon fine e indicava cinque possibili errori. Se si verifica un errore legato alla chiamata del metodo `upload()` in ActionScript 3.0, viene generato uno dei quattro errori possibili, il che permette di presentare agli utenti messaggi di errore più accurati.
- Gestione degli errori sofisticata. In conseguenza a molte situazioni comuni, possono essere generati errori distinti. Ad esempio, in ActionScript 2.0, se l'oggetto FileReference non è ancora stato impostato, la proprietà `name` ha valore `null`, perciò, per poter utilizzare o visualizzare la proprietà `name`, è necessario verificare che il suo valore non corrisponda a `null`. In ActionScript 3.0, se si tenta di accedere alla proprietà `name` prima di averla impostata, Flash Player genera un errore IllegalOperationError che informa l'utente che il valore della proprietà non è impostato e che può utilizzare i blocchi `try..catch..finally` per gestire l'errore. Per ulteriori informazioni, vedere "Uso delle istruzioni try..catch..finally" a pagina 281.
- Nessuno svantaggio significativo in termini di prestazioni. L'utilizzo dei blocchi `try..catch..finally` per gestire gli errori non richiede quasi o affatto risorse supplementari rispetto alle versioni precedenti di ActionScript.
- Classe ErrorEvent che permette di generare listener per errori asincroni specifici. Per ulteriori informazioni, vedere "Risposte a eventi errore e a errori basati sullo stato" a pagina 288.

Strategie di gestione degli errori

Fintanto che non incontrano intoppi, le applicazioni possono essere eseguite correttamente anche se il loro codice non prevede una logica per la gestione degli errori. Tuttavia, quando si verifica un problema in un'applicazione che non prevede la gestione attiva degli errori, gli utenti non sapranno mai perché l'applicazione si blocca.

Si può eseguire la gestione degli errori delle applicazioni in base a strategie diverse. Ecco le tre principali opzioni di cui dispone lo sviluppatore:

- Uso di istruzioni `try..catch..finally`. Queste istruzioni rilevano gli errori sincroni nel momento in cui si verificano. Annidando le istruzioni a vari livelli della gerarchia si possono catturare le eccezioni generate ai vari livelli di esecuzione del codice. Per ulteriori informazioni, vedere "Uso delle istruzioni try..catch..finally" a pagina 281.
- Creazione di oggetti errore personalizzati. Utilizzare la classe Error per creare oggetti errore personalizzati che rilevino operazioni specifiche dell'applicazione non coperte dai tipi di errore incorporati. Su questi oggetti si possono quindi utilizzare istruzioni `try..catch..finally`. Per ulteriori informazioni, vedere "Creazione di classi di errore personalizzate" a pagina 287.
- Scrittura di listener e gestori di eventi che rispondano agli errori. Questa strategia consente di creare gestori di eventi globali che consentono di gestire eventi simili senza duplicare grandi porzioni di codice nei blocchi `try..catch..finally`. Questo approccio offre anche maggiori possibilità di catturare gli errori asincroni. Per ulteriori informazioni, vedere "Risposte a eventi errore e a errori basati sullo stato" a pagina 288.

Utilizzo della versione debugger di Flash Player

Adobe offre agli sviluppatori una versione speciale di Flash Player a supporto delle operazioni di debug. Si ottiene una copia della versione debugger di Flash Player quando si installa Adobe Flash CS3 Professional o Adobe Flex Builder 2.

Esiste una sostanziale differenza tra le modalità di segnalazione degli errori tra la versione debugger e la versione standard di Flash Player. La versione debugger comunica il tipo di errore (come GenericError, IOError o EOFError), il numero dell'errore e un messaggio di errore in formato leggibile dall'uomo. La versione standard comunica solo il tipo di errore e il numero. Consideriamo l'esempio offerto dal codice seguente:

```
try
{
  tf.text = myByteArray.readBoolean();
}
catch (error:EOFError)
{
  tf.text = error.toString();
}
```

Se il metodo `readBoolean()` ha generato un errore EOFError nella versione debugger di Flash Player, nel campo di testo viene visualizzato il messaggio seguente `tf`: "EOFError: Error #2030: È stata rilevata la fine del file."

Lo stesso esempio di codice gestito dalla versione standard di Flash Player genera il seguente messaggio: "EOFError: Error #2030."

Al fine di limitare il più possibile la dimensione e le risorse utilizzate da Flash Player, le stringhe dei messaggi di errore non sono state incluse. L'utente ha però la facoltà di leggere una descrizione dell'errore cercandone il numero nella documentazione (le appendici della *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*). In alternativa, si può riprodurre l'errore utilizzando la versione debugger di Flash Player allo scopo di leggere il messaggio completo.

Gestione degli errori sincroni delle applicazioni

La modalità di gestione degli errori più comune è costituita dalla logica di gestione degli errori sincroni, che consiste nell'inserire istruzioni nel codice per catturare gli errori sincroni in fase di runtime. Questo tipo di approccio permette all'applicazione di rilevare e risolvere i problemi in fase di runtime quando le funzioni non vengono eseguite. La logica impiegata per rilevare gli errori sincroni utilizza, tra l'altro, le istruzioni `try..catch..finally` che provano ad eseguire un'operazione (fase "try"), catturano gli errori generati da Flash Player (fase "catch") e, alla fine (fase "finally"), eseguono altre attività per gestire l'operazione non riuscita.

Uso delle istruzioni try..catch..finally

Per rilevare gli errori sincroni in fase di runtime, usare le istruzioni `try..catch..finally`. Quando si verifica un errore runtime, Flash Player genera un'eccezione, ovvero sospende la normale esecuzione del codice e crea un oggetto speciale del tipo Error. L'oggetto Error viene quindi rilevato dal primo blocco `catch` disponibile.

L'istruzione `try` racchiude istruzioni che hanno il potenziale di creare errori e viene combinata a un'istruzione `catch`. Se viene rilevato un errore in una delle istruzioni del blocco `try`, le istruzioni `catch` collegate a quella particolare istruzione `try` vengono eseguite.

L'istruzione `finally` racchiude istruzioni eseguite indipendentemente dalla presenza di errori nel blocco `try`. In assenza di errori, le istruzioni del blocco `finally` vengono eseguite al termine delle istruzioni del blocco `try`. In presenza di un errore, viene innanzitutto eseguita l'istruzione `catch` appropriata seguita dalle istruzioni del blocco `finally`.

Il codice seguente dimostra la sintassi da impiegare per le istruzioni `try..catch..finally`:

```
try
{
   // codice che potrebbe generare un errore
}
catch (err:Error)
{
   // codice per reagire all'errore
}
finally
{
  // Codice eseguito indipendentemente da errori. Può eseguire operazioni
  // di pulizia dopo l'errore o mantenere attiva l'applicazione.
}
```

Ogni istruzione `catch` identifica un tipo di eccezione specifica da gestire. L'istruzione `catch` può specificare solo sottoclassi della classe Error. Le istruzioni `catch` vengono verificate in sequenza, ma viene eseguita solo la prima istruzione `catch` che coincide al tipo di errore generato. In altre parole, se si controlla per prima la classe Error di livello più alto e quindi una sottoclasse di questa classe, solo la classe Error di livello più alto evidenzierà una corrispondenza. Il codice seguente illustra questo punto:

```
try
{
  throw new ArgumentError("I am an ArgumentError");
}
catch (error:Error)
{
  trace("<Error> " + error.message);
}
catch (error:ArgumentError)
{
  trace("<ArgumentError> " + error.message);
}
```

Il codice precedente visualizza il seguente output:

```
<Error> I am an ArgumentError
```

Per poter rilevare correttamente l'errore ArgumentError, è necessario che i tipi specifici di errore siano elencati per primi seguiti dai tipi di errore più generici, come illustra il codice seguente:

```
try
{
  throw new ArgumentError("I am an ArgumentError");
}
catch (error:ArgumentError)
{
  trace("<ArgumentError> " + error.message);
}
catch (error:Error)
{
  trace("<Error> " + error.message);
}
```

Vari metodi e proprietà dell'API di Flash Player generano errori in fase di runtime se rilevano errori durante la loro esecuzione. Ad esempio, il metodo `close()` della classe Sound genera un errore IOError se il metodo non riesce a chiudere lo streaming audio, come illustrato nel codice seguente:

```
var mySound:Sound = new Sound();
try
{
  mySound.close();
}
catch (error:IOError)
{
  // Error #2029: Per questo oggetto URLStream non è aperto alcun flusso.
}
```

Man mano che si acquisisce dimestichezza con la *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*, si imparano i metodi che generano eccezioni, come illustrato nella descrizione dei singoli metodi.

L’istruzione throw

Flash Player genera eccezioni quando rileva degli errori nella fase di runtime dell’applicazione. Inoltre, lo sviluppatore può espressamente generare delle eccezioni usando l’istruzione `throw`. Per generare esplicitamente delle eccezioni, Adobe consiglia di generare istanze della classe Error o di sue sottoclassi. Il codice seguente illustra il funzionamento di un’istruzione `throw` che genera un’istanza della classe Error, `MyErr`, quindi richiama la funzione `myFunction()` per rispondere all’errore generato:

```
var MyError:Error = new Error("Encountered an error with the numUsers
  value", 99);
var numUsers:uint = 0;
try
{
  if (numUsers == 0)
  {
    trace("numUsers equals 0");
  }
}
catch (error:uint)
{
  throw MyError; // Rileva gli errori generati da numeri interi
                 // senza segno.
}
catch (error:int)
{
  throw MyError; // Rileva gli errori generati da numeri interi.
}
catch (error:Number)
{
  throw MyError; // Rileva gli errori generati da numeri.
}
catch (error:*)
{
  throw MyError; // Rileva qualsiasi altro errore.
}
finally
{
  myFunction(); // Esegue eventuali operazioni di pulizia richieste.
}
```

Si noti che la sequenza delle istruzioni `catch` parte da quella contenente i dati più specifici. Se fosse stata scritta per prima l'istruzione `catch` per il tipo di dati Number, né l'istruzione `catch` per il tipo di dati uint né l'istruzione catch per il tipo di dati int sarebbero mai state eseguite.

NOTA

Nel linguaggio di programmazione Java, ogni funzione che ha la facoltà di generare un'eccezione deve dichiararlo e indicare le classi di eccezione che possono essere generate in un'apposita clausola `throws` collegata alla descrizione della funzione. ActionScript non pone questo vincolo perciò non è necessario dichiarare le eccezioni che una funzione può generare.

Visualizzazione di un messaggio di errore semplice

Uno dei principali vantaggi che offre il nuovo modello di generazione di eccezioni e errori è che permette di comunicare agli utenti quando e perché un'azione non è andata a buon fine. Lo sviluppatore si deve occupare di scrivere il codice necessario per visualizzare il messaggio e presentare le opzioni per l'utente.

Il codice seguente rappresenta una semplice istruzione `try..catch` che permette di visualizzare l'errore in un campo di testo:

```
package
{
  import flash.display.Sprite;
  import flash.text.TextField;

  public class SimpleError extends Sprite
  {
    public var employee:XML =
      <EmpCode>
        <costCenter>1234</costCenter>
        <costCenter>1-234</costCenter>
      </EmpCode>;

    public function SimpleError()
    {
      try
      {
        if (employee.costCenter.length() != 1)
        {
          throw new Error("Error, employee must have exactly one cost
  center assigned.");
        }
      }
```

```
        catch (error:Error)
        {
          var errorMessage:TextField = new TextField();
          errorMessage.autoSize = TextFieldAutoSize.LEFT;
          errorMessage.textColor = 0xFF0000;
          errorMessage.text = error.message;
          addChild(errorMessage);
        }
      }
    }
}
```

Poiché si avvale di una gamma più ampia di classi di errore e di errori del compilatore incorporati, ActionScript 3.0 consente di visualizzare maggiori dettagli sulle operazioni non eseguite rispetto alle versioni precedenti del programma. Questa novità permette di creare applicazioni più stabili con procedure di gestione degli errori più efficaci.

Generazione ripetuta di errori

Durante la creazione delle applicazioni, si verifica spesso la necessità di ripetere la generazione di un errore che non è stato possibile gestire correttamente. Ad esempio, il codice seguente contiene un blocco `try..catch` nidificato che genera nuovamente un errore ApplicationError personalizzato se il blocco `catch` non riesce a gestire l'errore:

```
try
{
  try
  {
    trace("<< try >>");
    throw new ArgumentError("some error which will be rethrown");
  }
  catch (error:ApplicationError)
  {
    trace("<< catch >> " + error);
    trace("<< throw >>");
    throw error;
  }
  catch (error:Error)
  {
    trace("<< Error >> " + error);
  }
}
catch (error:ApplicationError)
{
  trace("<< catch >> " + error);
}
```

L'output generato dallo snippet di codice precedente sarebbe il seguente:

```
<< try >>
<< catch >> ApplicationError: some error which will be rethrown
<< throw >>
<< catch >> ApplicationError: some error which will be rethrown
```

Il blocco `try` nidificato genera un errore ApplicationError personalizzato catturato dal blocco `catch` successivo. Questo blocco `catch` nidificato può provare a gestire l'errore ma, se l'esito è negativo, genera l'oggetto ApplicationError diretto al blocco `try..catch`.

Creazione di classi di errore personalizzate

È possibile creare le proprie classi personalizzate da utilizzare in ActionScript ampliando le classi di errore standard. Vari motivi giustificano la creazione di classi personalizzate:

- Individuare errori o gruppi di errore specifici alla propria applicazione.

  Ad esempio, se si vogliono intraprendere azioni diverse in risposta agli errori generati dal codice, oltre a quelle avviate da Flash Player, si può creare una sottoclasse della classe Error che tenga traccia del nuovo tipo di errore in blocchi `try..catch`.
- Progettare modalità di visualizzazione uniche per gli errori generati dalla propria applicazione.

  Ad esempio, si può creare un nuovo metodo `toString()` che formatti i messaggi di errore in un modo particolare. Oppure, si può definire un metodo `lookupErrorString()` che, in base al codice di errore, recuperi la versione estesa del messaggio nella lingua preferita dall'utente.

Una classe di errore speciale deve costituire un'estensione di una classe ActionScript di base. Ecco un esempio di classe AppError speciale creata a partire da una classe Error di base:

```
public class AppError extends Error
{
  public function AppError(message:String, errorID:int)
  {
    super(message, errorID);
  }
}
```

Il codice seguente illustra un esempio di utilizzo di AppError in un progetto:

```
try
{
  throw new AppError("Encountered Custom AppError", 29);
}
catch (error:AppError)
{
  trace(error.errorID + ": " + error.message)
}
```

NOTA

Se si desidera ignorare il metodo `Error.toString()` nella sottoclasse, assegnargli un parametro `...(rest)`. La specifica del linguaggio ECMAScript (ECMA-262) Edizione 3 definisce il metodo `Error.toString()` in questo modo e, per assicurare la compatibilità con le versioni precedenti, ActionScript 3.0 fa lo stesso. Di conseguenza, per ignorare il metodo `Error.toString()` si devono utilizzare esattamente gli stessi parametri. Non passare i parametri al metodo `toString()` in fase di runtime perché tali parametri vengono ignorati.

Risposte a eventi errore e a errori basati sullo stato

Uno dei miglioramenti più evidenti apportati alla funzionalità di gestione degli errori in ActionScript 3.0 è rappresentato dal supporto della gestione degli errori asincroni della fase di runtime. (Per una definizione di errore asincrono, vedere "Tipi di errore" a pagina 275.)

Per rispondere agli eventi errore si possono creare listener e gestori di eventi. Molte classi inviano gli eventi errore esattamente come qualsiasi altro tipo di evento. Ad esempio, un'istanza della classe XMLSocket generalmente invia tre tipi di eventi: `Event.CLOSE`, `Event.CONNECT` e `DataEvent.DATA`. Tuttavia, quando si verifica un problema, la classe XMLSocket può inviare l'errore `IOErrorEvent.IOError` o l'errore `SecurityErrorEvent.SECURITY_ERROR`. Per ulteriori informazioni sui listener e gestori di eventi, vedere Capitolo 10, "Gestione degli eventi" a pagina 335.

Gli eventi errore appartengono a due categorie:

- Eventi errore che ampliano la classe ErrorEvent

  La classe flash.events.ErrorEvent contiene le proprietà e i metodi per la gestione degli errori runtime di Flash Player relativi alle attività di rete e di comunicazione. Le classi AsyncErrorEvent, IOErrorEvent e SecurityErrorEvent sono evoluzioni della classe ErrorEvent. Quando si utilizza la versione debugger di Flash Player, una finestra di dialogo comunica, in fase di runtime, gli eventi errore senza funzione di listener che il programma incontra.

- Eventi errore basati sullo stato

  Gli eventi errore basati sullo stato sono correlati alle proprietà `netStatus` e `status` delle classi di rete e comunicazione. Se Flash Player incontra un problema durante la lettura o scrittura dei dati, il valore della proprietà `netStatus.info.level` o `status.level` (a seconda della classe utilizzata) viene impostato su `"error"`.
  Si risponde a questo errore verificando se la proprietà `level` contiene il valore `"error"` nella funzione gestore di eventi.

Operazioni con gli eventi errore

La classe ErrorEvent e le relative sottoclassi contengono tipi di errore che Flash Player invia mentre tenta di leggere o scrivere dati.

L'esempio seguente illustra l'uso di un'istruzione `try..catch` e di gestori di eventi errore per visualizzare eventuali errori rilevati durante la lettura di un file locale. Per presentare agli utenti più opzioni, è possibile creare codice di gestione più sofisticato oppure gestire l'errore automaticamente nei punti indicati dal commento "inserire qui il codice di gestione degli errori":

```
package
{
  import flash.display.Sprite;
  import flash.errors.IOError;
  import flash.events.IOErrorEvent;
  import flash.events.TextEvent;
  import flash.media.Sound;
  import flash.media.SoundChannel;
  import flash.net.URLRequest;
  import flash.text.TextField;

  public class LinkEventExample extends Sprite
  {
    private var myMP3:Sound;
    public function LinkEventExample()
    {
      myMP3 = new Sound();
      var list:TextField = new TextField();
      list.autoSize = TextFieldAutoSize.LEFT;
      list.multiline = true;
      list.htmlText = "<a href=\"event:track1.mp3\">Track 1</a><br>";
      list.htmlText += "<a href=\"event:track2.mp3\">Track 2</a><br>";
      addEventListener(TextEvent.LINK, linkHandler);
      addChild(list);
    }
```

```
    private function playMP3(mp3:String):void
    {
      try
      {
        myMP3.load(new URLRequest(mp3));
        myMP3.play();
      }
      catch (err:Error)
      {
        trace(err.message);
        // inserire qui il codice di gestione degli errori
      }
      myMP3.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
    }

    private function linkHandler(linkEvent:TextEvent):void
    {
      playMP3(linkEvent.text);
      // inserire qui il codice di gestione degli errori
    }

    private function errorHandler(errorEvent:IOErrorEvent):void
    {
      trace(errorEvent.text);
      // inserire qui il codice di gestione degli errori
    }
  }
}
```

Operazioni con gli eventi di variazione dello stato

Flash Player varia in modo dinamico il valore della proprietà `netStatus.info.level` o della proprietà `status.level` delle classi che supportano la proprietà `level`. Le classi dotate della proprietà `netStatus.info.level` sono NetConnection, NetStream e SharedObject. Le classi dotate della proprietà `status.level` sono HTTPStatusEvent, Camera, Microphone e LocalConnection. Tramite una funzione gestore si può rispondere alla modifica del valore `level` e tenere traccia degli errori di comunicazione.

L'esempio seguente illustra l'utilizzo di una funzione `netStatusHandler()` per verificare il valore della proprietà `level`. Se la proprietà `level` indica che si è verificato un errore, il codice produce il messaggio "Video stream failed".

```
package
{
  import flash.display.Sprite;
  import flash.events.NetStatusEvent;
  import flash.events.SecurityErrorEvent;
  import flash.media.Video;
```

```
  import flash.net.NetConnection;
  import flash.net.NetStream;

  public class VideoExample extends Sprite
  {
    private var videoUrl:String = "Video.flv";
    private var connection:NetConnection;
    private var stream:NetStream;

    public function VideoExample()
    {
      connection = new NetConnection();
      connection.addEventListener(NetStatusEvent.NET_STATUS,
  netStatusHandler);
      connection.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
  securityErrorHandler);
      connection.connect(null);
    }

    private function netStatusHandler(event:NetStatusEvent):void
    {
      if (event.info.level = "error")
      {
        trace("Video stream failed")
      }
      else
      {
        connectStream();
      }
    }

    private function securityErrorHandler(event:SecurityErrorEvent):void
    {
      trace("securityErrorHandler: " + event);
    }

    private function connectStream():void
    {
      var stream:NetStream = new NetStream(connection);
      var video:Video = new Video();
      video.attachNetStream(stream);
      stream.play(videoUrl);
      addChild(video);
    }
  }
}
```

Confronto tra le classi di tipo Error

ActionScript comprende varie classi Error predefinite. Molte tra queste sono utilizzate da Flash Player, ma le stesse classi possono essere impiegate anche nel codice scritto dallo sviluppatore. In ActionScript 3.0 sono presenti due tipi principali di classi Error: le classi Error di base ActionScript e le classi Error del pacchetto flash.error. Le classi Error di base si basano sulla bozza della specifica del linguaggio ECMAScript (ECMA-262) Edizione 4, mentre le classi del pacchetto flash.error sono ausili supplementari introdotti in ActionScript 3.0 per agevolare lo sviluppo e il debug delle applicazioni.

Classi Error di base ECMAScript

Le classi Error di base ECMAScript comprendono Error, EvalError, RangeError, ReferenceError, SyntaxError, TypeError e URIError. Ognuna di queste è posizionata nello spazio dei nomi di livello superiore.

Nome classe	Descrizione	Note
Error	La classe Error può essere utilizzata per generare eccezioni e rappresenta la classe base per tutte le altre classi di eccezioni definite in ECMAScript: EvalError, RangeError, ReferenceError, SyntaxError, TypeError e URIError.	La classe Error funge da classe base per tutti gli errori runtime generati da Flash Player e si consiglia di usarla come modello per tutte le classi errore personalizzate.
EvalError	Un'eccezione EvalError viene generata quando si passano dei parametri alla funzione di costruzione della classe Function o se il codice utente chiama la funzione `eval()`.	A differenza delle versioni precedenti, ActionScript 3.0 non supporta la funzione `eval()` e qualsiasi tentativo di utilizzarla genera un errore. Le versioni precedenti di Flash Player utilizzavano la funzione `eval()` per accedere per nome a variabili, proprietà, oggetti e clip filmato.
RangeError	Un'eccezione RangeError viene generata quando un valore numerico è al di fuori dell'intervallo accettabile.	Ad esempio, la classe Timer genera un'eccezione RangeError se il ritardo specificato ha un valore negativo o non finito. Un'eccezione RangeError viene generata anche se si tenta di aggiungere un oggetto di visualizzazione a una profondità non valida.

Nome classe	Descrizione	Note
ReferenceError	Un'eccezione ReferenceError viene generata quando si tenta un riferimento a una proprietà undefined su un oggetto chiuso (non dinamico). Le versioni precedenti del compilatore ActionScript non generavano un errore quando si tentava di accedere a una proprietà `undefined`. Tuttavia, poiché la nuova specifica ECMAScript indica che questa condizione deve generare un errore, ActionScript 3.0 è stato adeguato.	Le eccezioni relative a variabili undefined segnalano la presenza di potenziali problemi e aiutano a migliorare la qualità del software. Tuttavia, se non si è abituati a inizializzare le variabili, questo nuovo comportamento di ActionScript richiede di rivedere le proprie modalità di scrittura del codice.
SyntaxError	Un'eccezione SyntaxError viene generata quando si verifica un errore di analisi nel codice ActionScript. Per ulteriori informazioni, vedere la Sezione 15.11.6.4 della specifica del linguaggio ECMAScript (ECMA-262) Edizione 3 (finché non diventa disponibile l'edizione 4) all'indirizzo www.ecma-international.org/publications/standards/Ecma-262.htm e la Sezione 10.3.1 della specifica ECMAScript for XML (E4X) (ECMA-357, Edizione 2) all'indirizzo www.ecma-international.org/publications/standards/Ecma-357.htm.	Un errore SyntaxError viene generato nelle seguenti circostanze: • ActionScript genera eccezioni SyntaxError quando un'espressione regolare non valida viene analizzata dalla classe RegExp. • ActionScript genera eccezioni SyntaxError quando del codice XML non valido viene analizzato dalla classe XMLDocument.

Nome classe	Descrizione	Note
TypeError	Un'eccezione TypeError viene generata quando il tipo effettivo di un operando è diverso dal tipo previsto. Per ulteriori informazioni, vedere la Sezione 15.11.6.5 della specifica ECMAScript all'indirizzo www.ecma-international.org/publications/standards/Ecma-262.htm e la Sezione 10.3 della specifica E4X all'indirizzo www.ecma-international.org/publications/standards/Ecma-357.htm.	Un errore TypeError viene generato nelle seguenti circostanze: • Non è stato possibile assegnare forzatamente un parametro effettivo di una funzione o di un metodo al tipo di parametro formale. • Un valore assegnato a una variabile non può essere assegnato forzatamente al tipo della variabile. • Il lato destro dell'operatore `is` o `instanceof` non è di tipo valido. • La parola chiave `super` viene utilizzata in modo non valido. • La ricerca di una proprietà restituisce più di un'associazione e pertanto è ambigua. • È stato richiamato un metodo su un oggetto incompatibile. Ad esempio, un'eccezione TypeError viene generata se un metodo di classe RegExp viene associato a un oggetto generico e successivamente richiamato.
URIError	Un'eccezione URIError viene generata quando una delle funzioni di gestione URI globali viene utilizzata in modo incompatibile con la propria definizione. Per ulteriori informazioni, vedere la Sezione 15.11.6.6 della specifica ECMAScript all'indirizzo www.ecma-international.org/publications/standards/Ecma-262.htm.	Un errore URIError viene generato nelle seguenti circostanze: Viene specificato un URI non valido per una funzione dell'API di Flash Player che prevede un URI valido, come `Socket.connect()`.

Classi Error di base ActionScript

Oltre alle classi Error di base ECMAScript, ActionScript mette a disposizione varie classi specifiche per operazioni di rilevamento e gestione degli errori proprie di ActionScript.

In qualità di estensioni in linguaggio ActionScript della bozza della specifica del linguaggio ECMAScript Edizione 4 che potrebbero potenzialmente rappresentare interessanti aggiunte a tale bozza, queste classi sono state mantenute al livello principale anziché essere inserite in un pacchetto come flash.error.

Nome classe	Descrizione	Note
ArgumentError	La classe ArgumentError rappresenta un errore che si verifica quando i parametri forniti durante la chiamata di una funzione non corrispondono ai parametri definiti per la funzione stessa.	Di seguito sono riportati esempi di ArgumentError: • A un metodo vengono forniti troppi o troppo pochi argomenti. • L'argomento previsto doveva essere membro di un'enumerazione, ma non lo è.
SecurityError	Un'eccezione SecurityError viene generata quando si verifica una violazione della sicurezza e l'accesso viene negato.	Di seguito sono riportati esempi di SecurityError: • Viene effettuato un accesso non autorizzato a una proprietà o una chiamata non consentita a un metodo al di fuori di una funzione di sicurezza sandbox. • Viene effettuato un tentativo di accesso a un URL non autorizzato dalla funzione di sicurezza sandbox. • Viene tentata una connessione socket a un numero di porta non autorizzato, ad esempio una porta inferiore alla 1024, senza che sia presente un file dei criteri. • Viene effettuato un tentativo di accesso alla fotocamera o al microfono dell'utente e la richiesta di accedere al dispositivo viene rifiutata dall'utente.
VerifyError	Un'eccezione VerifyError viene generata quando si incontra un file SWF strutturato in modo errato o danneggiato.	Quando un file SWF carica un altro file SWF, il file SWF principale può rilevare un'eccezione VerifyError generata dal file SWF caricato.

Classi Error del pacchetto flash.error

Il pacchetto flash.error contiene classi Error considerate parte dell'API di Flash Player. Diversamente da quanto avviene per le classi Error presentate fin'ora, il pacchetto flash.error comunica eventi errore specifici a Flash Player.

Nome classe	Descrizione	Note
EOFError	L'eccezione EOFError viene generata se si tenta di leggere oltre la fine dei dati disponibili.	Ad esempio, quando uno dei metodi di lettura dell'interfaccia IDataInput viene chiamato e i dati non sono sufficienti per soddisfare la richiesta di lettura.
IllegalOperationError	Un'eccezione IllegalOperationError viene generata se un metodo non viene implementato oppure se l'implementazione non copre l'uso corrente.	Di seguito sono riportati esempi di eccezioni per operazioni non valide: • Una classe base, ad esempio DisplayObjectContainer, fornisce più funzionalità rispetto a quelle supportate dallo stage. Ad esempio, se si tenta di ottenere o impostare una maschera sullo stage (tramite `stage.mask`), Flash Player genera un'eccezione IllegalOperationError con il messaggio "La classe Stage non implementa questa proprietà o questo metodo". • Una sottoclasse eredita un metodo di cui non ha bisogno e che non vuole supportare. • Alcuni metodi di accessibilità vengono chiamati quando Flash Player viene compilato senza il supporto dell'accessibilità. • Le funzioni di sola creazione vengono chiamate da una versione runtime di Flash Player. • Si tenta di impostare il nome di un oggetto inserito nella linea temporale.
IOError	Un'eccezione IOError viene generata quando si verifica un'eccezione di tipo I/O.	Si verifica questo errore, ad esempio, se si tenta di eseguire un'operazione di lettura/scrittura su un socket non connesso o che è stato disconnesso.

Nome classe	Descrizione	Note
MemoryError	Un'eccezione MemoryError viene generata quando una richiesta di allocazione memoria fallisce.	Per impostazione predefinita, ActionScript Virtual Machine 2 non impone un limite sulla quantità di memoria che un programma ActionScript può allocare. Sui PC desktop, le richieste di allocazione memoria sono rare. Si verifica un errore quando il sistema non è in grado di allocare la memoria necessaria per un'operazione. Di conseguenza, su un PC desktop, questa eccezione è rara a meno che la richiesta di memoria non sia estremamente grande; per esempio, una richiesta di 3 miliardi di byte è impossibile perché un programma Microsoft® Windows® a 32 bit è in grado di accedere a uno spazio indirizzi di 2 GB.
ScriptTimeoutError	Un'eccezione ScriptTimeoutError viene generata quando viene raggiunto l'intervallo di timeout dello script di 15 secondi. Rilevando l'eccezione ScriptTimeoutError è possibile gestire il timeout dello script in modo appropriato. In mancanza di un gestore di eccezioni, viene visualizzata una finestra di dialogo con un messaggio di errore.	Per impedire che uno sviluppatore poco scrupoloso rilevi l'eccezione dando origine a un ciclo infinito, viene catturata solo la prima eccezione ScriptTimeoutError generata durante l'esecuzione di un particolare script. L'eccezione ScriptTimeoutError successiva non viene rilevata dal codice e passa immediatamente al gestore di eccezioni non rilevate.
StackOverflowError	Un'eccezione StackOverflowError viene generata quando lo stack disponibile per lo script è esaurito.	Un'eccezione StackOverflowError potrebbe indicare che si è verificata una ricorsività infinita.

Esempio: Applicazione CustomErrors

L'applicazione CustomErrors illustra le tecniche di gestione degli errori personalizzati durante la creazione di un'applicazione. Le tecniche sono descritte di seguito.

- Convalida di un pacchetto XML
- Scrittura di un errore personalizzato
- Generazione di errori personalizzati
- Notifica agli utenti di ogni errore generato

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione CustomErrors si trovano nella cartella Samples/CustomError. L'applicazione comprende i seguenti file:

File	Descrizione
CustomErrors.mxml o CustomErrors.fla	Il file principale dell'applicazione in Flash (FLA) o Flex (MXML)
com/example/programmingas3/errors/ApplicationError.as	Classe Error che funge da classe base per le classi FatalError e WarningError.
com/example/programmingas3/errors/FatalError.as	Classe che definisce l'errore FatalError che l'applicazione può generare. Si tratta di un'estensione della classe personalizzata ApplicationError.
com/example/programmingas3/errors/Validator.as	Classe che definisce il metodo di convalida del pacchetto XML dipendente fornito dall'utente.
com/example/programmingas3/errors/WarningError.as	Classe che definisce l'errore WarningError che l'applicazione può generare. Si tratta di un'estensione della classe personalizzata ApplicationError.

Panoramica dell'applicazione CustomErrors

Il file CustomErrors.mxml racchiude l'interfaccia utente e una parte della logica alla base dell'applicazione CustomErrors. Una volta inviato l'evento `creationComplete` dell'applicazione, viene chiamato il metodo `initApp()` che definisce un pacchetto XML campione da verificare tramite la classe Validator. Il codice seguente illustra il metodo `initApp()`:

```
private function initApp():void
{
  employeeXML =
    <employee id="12345">
      <firstName>John</firstName>
      <lastName>Doe</lastName>
      <costCenter>12345</costCenter>
      <costCenter>67890</costCenter>
    </employee>;
}
```

Il pacchetto XML viene in seguito visualizzato nell'istanza di un componente TextArea sullo stage. In questo modo è possibile modificare il pacchetto XML prima di sottoporlo a una seconda convalida.

La selezione del pulsante Validate da parte dell'utente richiama il metodo `validateData()`. Il metodo convalida il pacchetto XML del dipendente tramite il metodo `validateEmployeeXML()` della classe Validator. Il codice seguente illustra il metodo `validateData()`:

```
public function validateData():void
{
  try
  {
    var tempXML:XML = XML(xmlText.text);
    Validator.validateEmployeeXML(tempXML);
    status.text = "The XML was successfully validated.";
  }
  catch (error:FatalError)
  {
    showFatalError(error);
  }
  catch (error:WarningError)
  {
    showWarningError(error);
  }
  catch (error:Error)
  {
    showGenericError(error);
  }
}
```

Per cominciare, viene creato un oggetto XML temporaneo usando il contenuto dell'istanza `xmlText` del componente TextArea. In seguito, viene chiamato il metodo `validateEmployeeXML()` della classe personalizzata Validator (com.example.programmingas3/errors/Validator.as) che passa l'oggetto XML temporaneo come parametro. Se il pacchetto XML è valido, l'istanza `status` del componente Label visualizza un messaggio indicante che l'operazione ha avuto esito positivo e l'applicazione si chiude. Se il metodo `validateEmployeeXML()` ha generato un errore personalizzato (cioè, un errore FatalError, WarningError o GenericError), viene eseguita l'istruzione `catch` appropriata che richiama il metodo `showFatalError()`, `showWarningError()` oppure `showGenericError()`. Ognuno di questi metodi visualizza un messaggio pertinente in un componente Alert per informare l'utente dell'errore specifico che si è verificato. Ogni metodo aggiorna, inoltre, l'istanza `status` del componente Label con un messaggio appropriato.

Se si verifica un errore irreversibile durante il tentativo di convalidare il pacchetto XML del dipendente, il messaggio di errore viene visualizzato in un componente Alert e l'istanza `xmlText` del componente TextArea e l'istanza `validateBtn` del componente Button vengono disabilitate, come illustra il codice seguente:

```
public function showFatalError(error:FatalError):void
{
  var message:String = error.message + "\n\n" + "Click OK to end.";
  var title:String = error.getTitle();
  Alert.show(message, title);
  status.text = "This application has ended.";
  this.xmlText.enabled = false;
  this.validateBtn.enabled = false;
}
```

Se si verifica un'avvertenza in luogo di un errore irrecuperabile, il messaggio di errore viene comunque visualizzato nell'istanza di un componente Alert, ma le istanze dei componenti TextField e Button non vengono disabilitate. Il metodo `showWarningError()` visualizza il messaggio di errore personalizzato nell'istanza del componente Alert. Il messaggio chiede, inoltre, all'utente di decidere se desidera procedere con la convalida del pacchetto XML o annullare lo script. La seguente porzione di codice illustra il metodo `showWarningError()`:

```
public function showWarningError(error:WarningError):void
{
  var message:String = error.message + "\n\n" + "Do you want to exit this
  application?";
  var title:String = error.getTitle();
  Alert.show(message, title, Alert.YES | Alert.NO, null, closeHandler);
  status.text = message;
}
```

Quando l'utente chiude l'istanza del componente Alert mediante il pulsante Yes o No, viene chiamato il metodo `closeHandler()`. La seguente porzione di codice illustra il metodo `closeHandler()`:

```
private function closeHandler(event:CloseEvent):void
{
  switch (event.detail)
  {
    case Alert.YES:
      showFatalError(new FatalError(9999));
      break;
    case Alert.NO:
      break;
  }
}
```

Se l'utente sceglie di annullare lo script facendo clic su Yes nella finestra di avvertenza Alert, viene generato un errore FatalError e l'applicazione viene interrotta.

Creazione della classe Validator personalizzata

La classe Validator personalizzata contiene un unico metodo: `validateEmployeeXML()`. Il metodo `validateEmployeeXML()` accetta un solo argomento, `employee`, vale a dire il pacchetto XML da convalidare. Il metodo `validateEmployeeXML()` è il seguente:

```
public static function validateEmployeeXML(employee:XML):void
{
  // checks for the integrity of items in the XML
  if (employee.costCenter.length() < 1)
  {
    throw new FatalError(9000);
  }
  if (employee.costCenter.length() > 1)
  {
    throw new WarningError(9001);
  }
  if (employee.ssn.length() != 1)
  {
    throw new FatalError(9002);
  }
}
```

Per poter essere convalidato, un dipendente deve appartenere a un centro costi (e solo a uno). Se il dipendente non appartiene a nessun centro costi, il metodo genera un FatalError che si propaga al metodo `validateData()` nel file principale dell'applicazione. Se il dipendente appartiene a più di un centro costi, viene generato un WarningError. La verifica finale effettuata sul pacchetto XML controlla che per l'utente sia definito un numero di iscrizione alla previdenza sociale (il nodo `ssn` del pacchetto XML). Se non è disponibile esattamente un nodo `ssn`, viene generato un FatalError.

A discrezione dello sviluppatore, è possibile aggiungere verifiche supplementari al metodo `validateEmployeeXML()`, ad esempio, per controllare che il nodo `ssn` contenga un numero valido o che per il dipendente siano stati definiti almeno un numero di telefono e un indirizzo e-mail e che entrambi i valori siano validi. Si può inoltre modificare il codice XML in modo che ogni dipendente disponga di un ID univoco e indichi l'ID del proprio manager.

Definizione della classe ApplicationError

La classe ApplicationError funge da classe base per le classi FatalError e WarningError. La classe ApplicationError è un'estensione della classe Error che definisce i metodi e proprietà specifici, compresa la definizione di un ID errore, la gravità e l'oggetto XML che contiene i codici e i messaggi dell'errore personalizzato. La classe definisce anche due costanti statiche utilizzate per definire la gravità di ogni tipo di errore.

Il metodo di costruzione della classe ApplicationError è il seguente:

```
public function ApplicationError()
{
  messages =
    <errors>
      <error code="9000">
        <![CDATA[Employee must be assigned to a cost center.]]>
      </error>
      <error code="9001">
        <![CDATA[Employee must be assigned to only one cost center.]]>
      </error>
      <error code="9002">
        <![CDATA[Employee must have one and only one SSN.]]>
      </error>
      <error code="9999">
        <![CDATA[The application has been stopped.]]>
      </error>
    </errors>;
}
```

Ogni nodo errore dell'oggetto XML contiene un codice numerico e un messaggio univoco. I messaggi di errore possono essere facilmente consultati in base al codice di errore usando E4X, come illustrato nel metodo `getMessageText()` seguente:

```
public function getMessageText(id:int):String
{
  var message:XMLList = messages.error.(@code == id);
  return message[0].text();
}
```

Il metodo `getMessageText()` accetta un solo argomento costituito da un numero intero, `id`, e restituisce una stringa. L'argomento `id` è il codice dell'errore da cercare. Ad esempio, passando un `id` 9001 si recupera l'errore secondo il quale i dipendenti devono essere assegnati a un solo centro costi. Se più errori condividono lo stesso codice, ActionScript restituisce il messaggio di errore per il primo risultato trovato (`message[0]` nell'oggetto XMLList restituito).

Il metodo seguente di questa classe, `getTitle()`, non accetta parametri e restituisce una stringa che contiene l'ID specifico dell'errore. Questo valore viene utilizzato nel titolo del componente Alert per contribuire all'identificazione dell'errore esatto verificatosi durante la convalida del pacchetto XML. La seguente porzione di codice illustra il metodo `getTitle()`:

```
public function getTitle():String
{
  return "Error #" + id;
}
```

L'ultimo metodo della classe ApplicationError è `toString()` che sostituisce la funzione definita nella classe Error per rendere possibile la personalizzazione della presentazione del messaggio di errore. Il metodo restituisce una stringa che identifica un numero di errore specifico e il messaggio.

```
public override function toString():String
{
  return "[APPLICATION ERROR #" + id + "] " + message;
}
```

Definizione della classe FatalError

La classe FatalError è un'estensione della classe personalizzata ApplicationError e definisce tre metodi: la funzione di costruzione FatalError, `getTitle()` e `toString()`. Il primo metodo, la funzione di costruzione FatalError, accetta un solo argomento costituito da un numero intero, `errorID`, imposta la gravità dell'errore mediante valori costanti statici definiti nella classe ApplicationError e ottiene il messaggio di errore specifico chiamando il metodo `getMessageText()` della classe ApplicationError. La funzione di costruzione FatalError è la seguente:

```
public function FatalError(errorID:int)
{
  id = errorID;
  severity = ApplicationError.FATAL;
  message = getMessageText(errorID);
}
```

Il metodo seguente della classe FatalError, `getTitle()`, sostituisce il metodo `getTitle()` definito nella classe ApplicationError e aggiunge la stringa "-- FATAL" al titolo per informare l'utente che si è verificato un errore irrecuperabile. Il metodo `getTitle()` è il seguente:

```
public override function getTitle():String
{
  return "Error #" + id + " -- FATAL";
}
```

L'ultimo metodo di questa classe, `toString()`, sostituisce il metodo `toString()` definito nella classe ApplicationError. Il metodo `toString()` è il seguente:

```
public override function toString():String
{
  return "[FATAL ERROR #" + id + "] " + message;
}
```

Definizione della classe WarningError

La classe WarningError è un'estensione della classe ApplicationError ed è quasi identica alla classe FatalError, fatta eccezione per un paio di variazioni minime alle stringhe e per il fatto che imposta la gravità dell'errore su ApplicationError.WARNING in luogo di ApplicationError.FATAL, come illustrato nel codice seguente:

```
public function WarningError(errorID:int)
{
  id = errorID;
  severity = ApplicationError.WARNING;
  message = super.getMessageText(errorID);
}
```

CAPITOLO 9

Uso delle espressioni regolari

9

Le espressioni regolari sono modelli usati per cercare e modificare il testo delle stringhe. Un'espressione regolare assomiglia a una stringa, ma può contenere codici speciali che descrivono modelli o ripetizioni. Ad esempio, la seguente espressione regolare corrisponde a una stringa che inizia con la lettera A seguita da una o più cifre in sequenza:

```
/A\d+/
```

Il presente capitolo descrive la sintassi di base per creare le espressioni regolari. Tuttavia, le espressioni regolari possono presentare notevoli complessità e sfumature. Cercare informazioni più dettagliate sulle espressioni regolari su Internet o in libreria. Tenere presente che i vari ambienti di programmazione utilizzano le espressioni regolari in modi diversi. ActionScript 3.0 implementa le espressioni regolari in base a quanto definito nella specifica del linguaggio ECMAScript Edizione 3 (ECMA-262).

Sommario

Nozioni di base delle espressioni regolari

Introduzione all'uso delle espressioni regolari

Le espressioni regolari descrivono un modello di caratteri e vengono generalmente usate per verificare la conformità del testo a un modello specifico (per verificare, ad esempio, che un numero di telefono inserito dall'utente contenga il numero di cifre previsto) oppure per sostituire le porzioni di testo che coincidono con un particolare modello.

Le espressioni regolari possono essere semplici. Ad esempio, si supponga di voler verificare che una particolare stringa contenga il valore "ABC" o di voler sostituire ogni occorrenza di "ABC" di una stringa con un testo diverso. In questo caso, si potrebbe usare la seguente espressione regolare che definisce il modello composto dalle lettere A, B e C in sequenza:

```
/ABC/
```

Notare che l'inizio e la fine dell'espressione regolare sono indicati dalla barra (`/`).

La sintassi delle espressioni regolari può essere anche complessa, talvolta addirittura criptica, come dimostra l'esempio seguente di espressione che cerca un indirizzo e-mail valido:

```
/([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z]{2,6}/
```

Generalmente, le espressioni regolari vengono utilizzate per cercare dei modelli all'interno di stringhe e per sostituire dei caratteri. In questi casi, si crea un oggetto espressione regolare e lo si usa come parametro di uno dei vari metodi della classe String. I seguenti metodi della classe String accettano le espressioni regolari come parametri: `match()`, `replace()`, `search()` e `split()`. Per ulteriori informazioni su questi metodi, vedere "Ricerca di modelli nelle stringhe e sostituzione di sottostringhe" a pagina 226.

La classe RegExp comprende i seguenti metodi: `test()` e `exec()`. Per ulteriori informazioni, vedere "Metodi di impiego di espressioni regolari con stringhe" a pagina 327.

Operazioni comuni eseguite con le espressioni regolari

Le espressioni regolari hanno vari impieghi comuni, che il presente capitolo descrive nel dettaglio:

- Creazione di un modello di espressione regolare
- Uso di caratteri speciali nei modelli
- Individuazione di sequenze di più caratteri (come "numeri a due cifre" o "comprendenti tra sette e dieci lettere")
- Individuazione di qualsiasi carattere di un gruppo di lettere o numeri (come "qualsiasi lettera compresa tra *a* e *m*")
- Individuazione di un carattere in un set di caratteri possibili
- Individuazione di sottosequenze (segmenti all'interno di un modello)
- Individuazione e sostituzione di testo sulla base di modelli

Concetti e termini importanti

L'elenco seguente contiene i termini più importanti che vengono utilizzati in questo capitolo:

- Carattere escape: indica che il carattere che segue deve essere considerato un metacarattere e non il carattere nel suo valore letterale. Nella sintassi delle espressioni regolari, il carattere barra rovesciata (\) è il carattere escape quindi, una barra rovesciata seguita da un altro carattere rappresenta un codice speciale e non il semplice carattere.
- Flag: carattere che specifica come interpretare un modello di espressione regolare, ad esempio se vale la distinzione tra lettere maiuscole e minuscole.
- Metacarattere: carattere che ha un significato speciale nel modello di un'espressione regolare che travalica il suo significato letterale.
- Quantificatore: carattere (o più caratteri) che indicano il numero di ripetizioni del modello. Ad esempio, un quantificatore potrebbe essere usato per indicare che un codice postale degli USA può contenere cinque o nove numeri.
- Espressione regolare: istruzione di un programma che definisce un modello di caratteri che possono essere utilizzati per individuare stringhe che contengono quel modello o per sostituire porzioni di stringa.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché tali esempi consistono principalmente in modelli delle espressioni regolari, la prova comprende le operazioni seguenti:

1. Creare un nuovo documento Flash.
2. Selezionare un fotogramma chiave e aprire il pannello Azioni.
3. Creare una variabile RegExp (espressione regolare) come la seguente:

   ```
   var pattern:RegExp = /ABC/;
   ```

4. Copiare il modello dell'esempio e assegnarlo come valore della variabile RegExp. Ad esempio, nella riga di codice precedente, il modello è la parte di codice a destra del segno di uguale, escluso il punto e virgola (`/ABC/`).
5. Creare una o più variabili String appropriate per provare l'espressione regolare. Ad esempio, se si crea un'espressione regolare per cercare indirizzi e-mail validi, creare alcune variabili String contenenti indirizzi e-mail validi e non validi:

   ```
   var goodEmail:String = "bob@example.com";
   var badEmail:String = "5@$2.99";
   ```

6. Aggiungere righe di codice per provare le variabili String e stabilire se corrispondono al modello dell'espressione regolare. Questi sono i valori che verranno visualizzati sullo schermo utilizzando la funzione `trace()` oppure mediante la scrittura in un campo di testo sullo stage.

   ```
   trace(goodEmail, " is valid:", pattern.test(goodEmail));
   trace(badEmail, " is valid:", pattern.test(badEmail));
   ```

   Ad esempio, supponendo che `pattern` definisca il modello dell'espressione regolare per un indirizzo e-mail valido, le righe di codice precedenti scrivono nel pannello Output il testo seguente:

   ```
   bob@example.com is valid: true
   5@$2.99 is valid: false
   ```

   Per ulteriori informazioni su come provare i valori scrivendoli in un'istanza di campo di testo sullo stage oppure utilizzando la funzione `trace()` per stamparli nel pannello Output, vedere "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Sintassi delle espressioni regolari

Questa sezione descrive tutti gli elementi disponibili in ActionScript per la sintassi delle espressioni regolari. Come vedremo, le espressioni regolari possono presentare notevoli complessità e sfumature. Cercare informazioni più dettagliate sulle espressioni regolari su Internet o in libreria. Tenere presente che i vari ambienti di programmazione utilizzano le espressioni regolari in modi diversi. ActionScript 3.0 implementa le espressioni regolari in base a quanto definito nella specifica del linguaggio ECMAScript Edizione 3 (ECMA-262).

Generalmente, le espressioni regolari vengono utilizzate per cercare modelli più complicati di una semplice sequenza di caratteri. Ad esempio, la seguente espressione regolare definisce il modello composto dalle lettere A, B e C in sequenza seguito da una qualsiasi cifra:

```
/ABC\d/
```

Il codice `\d` rappresenta "qualsiasi cifra". La barra rovesciata (`\`) viene chiamata carattere escape e la sua combinazione con il carattere immediatamente seguente (in questo caso la lettera d) ha un significato speciale. Il presente capitolo descrive le sequenze composte con il carattere escape e altre funzioni di sintassi.

La seguente espressione regolare definisce il modello composto dalle lettere ABC seguito da un qualsiasi numero di cifre (notare l'asterisco):

```
/ABC\d*/
```

L'asterisco (`*`) è un *metacarattere*. vale a dire un carattere con un significato speciale all'interno delle espressioni regolari. L'asterisco è un tipo di metacarattere specifico, definito *quantificatore*, che consente di quantificare il numero di ripetizioni di un carattere o gruppo di caratteri. Per ulteriori informazioni, vedere "Quantificatori" a pagina 316.

Oltre a questo modello, un'espressione regolare può contenere dei flag che specificano la modalità di ricerca dell'espressione regolare. Ad esempio, nella seguente espressione si usa il flag `i` per indicare che l'espressione regolare cerca indistintamente le maiuscole e le minuscole nelle stringhe:

```
/ABC\d*/i
```

Per ulteriori informazioni, vedere "Flag e proprietà" a pagina 322.

È possibile usare le espressioni regolari con i seguenti metodi della classe String: `match()`, `replace()` e `search()`. Per ulteriori informazioni su questi metodi, vedere "Ricerca di modelli nelle stringhe e sostituzione di sottostringhe" a pagina 226.

Creazione di un'istanza di un'espressione regolare

Esistono due modi per creare un'istanza di un'espressione regolare. Uno consiste nell'usare le barre (/) per delimitare l'espressione; l'altro nell'impiegare la funzione di costruzione new. Ad esempio, le espressioni regolari seguenti sono equivalenti:

```
var pattern1:RegExp = /bob/i;
var pattern2:RegExp = new RegExp("bob", "i");
```

Le barre delimitano il valore letterale di un'espressione regolare come le virgolette doppie delimitano il valore letterale di una stringa. La porzione di espressione regolare tra le barre definisce il *modello*. L'espressione regolare può comprendere anche dei *flag* dopo la seconda barra. Tali flag vengono considerati come parti integranti dell'espressione regolare ma distinti dal suo modello.

Per definire l'espressione regolare usando la funzione di costruzione new, è necessario usare due stringhe: la prima per definire il modello e la seconda per definire i flag, come illustra l'esempio seguente:

```
var pattern2:RegExp = new RegExp("bob", "i");
```

Per includere una barra *all'interno* di un'espressione regolare che usa le barre come delimitatori, è necessario anteporre alla barra il carattere escape barra rovesciata (\). Ad esempio, la seguente espressione regolare definisce il modello 1/2:

```
var pattern:RegExp = /1\/2/;
```

Per includere delle virgolette doppie *all'interno* di un'espressione regolare definita mediante la funzione di costruzione new, è necessario aggiungere il carattere escape barra rovesciata (\) prima delle virgolette (come si fa per definire un valore letterale di stringa). Ad esempio, le seguenti espressioni letterali definiscono il modello eat at "joe's":

```
var pattern1:RegExp = new RegExp("eat at \"joe's\"", "");
var pattern2:RegExp = new RegExp('eat at "joe\'s"', "");
```

Non utilizzare il carattere escape barra rovesciata con virgolette doppie all'interno di espressioni regolari definite usando le barre come delimitatori. Analogamente, non usare il carattere escape con barre all'interno di espressioni regolari definite mediante la funzione di costruzione new. Le seguenti espressioni regolari si equivalgono e definiscono il modello 1/2 "joe's":

```
var pattern1:RegExp = /1\/2 "joe's"/;
var pattern2:RegExp = new RegExp("1/2 \"joe's\"", "");
var pattern3:RegExp = new RegExp('1/2 "joe\'s"', '');
```

Inoltre, in un'espressione regolare che è definita con la funzione di costruzione `new`, per utilizzare una metasequenza che inizia con la barra rovesciata (`\`), ad esempio `\d` (che corrisponde a qualsiasi cifra), digitare la barra rovesciata due volte:

```
var pattern:RegExp = new RegExp("\\d+", ""); // corrisponde a una
                                              // o più cifre
```

In questo caso è necessario digitare la barra rovesciata due volte perché il primo parametro del metodo di costruzione `RegExp()` è una stringa e nel valore letterale di una stringa è necessario digitare una barra rovesciata due volte affinché venga riconosciuta come singolo carattere di barra rovesciata.

Le sezioni seguenti descrivono la sintassi per la definizione dei modelli delle espressioni regolari.

Per ulteriori informazioni sui flag, vedere "Flag e proprietà" a pagina 322.

Caratteri, metacaratteri e metasequenze

L'espressione regolare più semplice è quella che cerca una sequenza di caratteri, come nell'esempio seguente:

```
var pattern:RegExp = /hello/;
```

Tuttavia, ci sono dei caratteri speciali, definiti metacaratteri, che hanno un significato speciale all'interno delle espressioni regolari:

```
^ $ \ . * + ? ( ) [ ] { } |
```

L'espressione regolare dell'esempio seguente cerca la lettera A seguita da zero o più istanze della lettera B (il metacarattere asterisco indica la ripetizione), seguita dalla lettera C:

```
/AB*C/
```

Per utilizzare un metacarattere in un'espressione regolare svuotandolo del suo significato speciale, è sufficiente farlo precedere dal carattere escape barra rovesciata (`\`). L'espressione regolare dell'esempio seguente cerca la lettera A, seguita dalla lettera B, seguita da un asterisco, seguita dalla lettera C:

```
var pattern:RegExp = /AB\*C/;
```

Una *metasequenza,* proprio come un metacarattere, ha un significato speciale in un'espressione regolare. Una metasequenza comprende più caratteri. La sezione seguente fornisce dettagli sull'uso dei metacaratteri e delle metasequenze.

Informazioni sui metacaratteri

La tabella seguente contiene i metacaratteri che si possono usare nelle espressioni regolari:

Metacarattere	Descrizione
^ (accento circonflesso)	Corrisponde all'inizio della stringa. Se si imposta anche il flag `m` (`multiline`), l'accento circonflesso corrisponde anche all'inizio di una riga (vedere "Flag m (multiline)" a pagina 324). Si noti che quando viene usato all'inizio di una classe di caratteri, l'accento circonflesso indica la negazione, non l'inizio di una stringa. Per ulteriori informazioni, vedere "Classi di caratteri" a pagina 314.
$ (simbolo del dollaro)	Corrisponde alla fine della stringa. Se si imposta anche il flag `m` (`multiline`), `$` cerca la corrispondenza anche prima di un carattere nuova riga (`\n`). Per ulteriori informazioni, vedere "Flag m (multiline)" a pagina 324.
\ (barra rovesciata)	Sospende il significato speciale dei metacaratteri. Anteporre la barra rovesciata anche se si vuole inserire la barra nel valore letterale di un'espressione regolare, come in `/1\/2/` (l'espressione cerca il carattere 1, seguito dal carattere barra seguito dal carattere 2).
. (punto)	Corrisponde a un carattere qualunque. Il punto corrisponde a un carattere nuova riga (`\n`) solo se è presente il flag `s` (`dotall`). Per ulteriori informazioni, vedere "Flag s (dotall)" a pagina 325.
* (asterisco)	Corrisponde a una ripetizione di zero o più volte del carattere precedente. Per ulteriori informazioni, vedere "Quantificatori" a pagina 316.
+ (più)	Corrisponde a una ripetizione di una o più volte del carattere precedente. Per ulteriori informazioni, vedere "Quantificatori" a pagina 316.
? (punto di domanda)	Corrisponde a una ripetizione di zero o una volta del carattere precedente. Per ulteriori informazioni, vedere "Quantificatori" a pagina 316.
(e)	Definiscono un raggruppamento all'interno dell'espressione regolare. Usare i gruppi per: • Limitare l'area di validità del carattere di alternanza |: `/(a|b|c)d/` • Definire l'area di validità di un quantificatore: `/(walla.){1,2}/` • Eseguire riferimenti a posteriori. Ad esempio, nell'espressione regolare seguente, `\1` corrisponde ai valori individuati dal primo gruppo parentetico del modello: `/(\w*) is repeated: \1/` Per ulteriori informazioni, vedere "Gruppi" a pagina 318.

Metacarattere	Descrizione
`[` e `]`	Definiscono una classe di caratteri, che a sua volta definisce possibili corrispondenze di un unico carattere: `/[aeiou]/` corrisponde a qualsiasi dei caratteri specificati. Usare il trattino (`-`) all'interno delle classi di caratteri per definire un intervallo di caratteri: `/[A-Z0-9]/` corrisponde ai caratteri maiuscoli da A a Z o da 0 a 9. All'interno delle classi di caratteri, inserire una barra rovesciata per proteggere il valore letterale dei caratteri] e -: `/[+\-]\d+/` corrisponde al segno `+` o `-` prima di uno o più caratteri. All'interno delle classi di caratteri, i cosiddetti metacaratteri vengono considerati caratteri normali senza il bisogno di anteporre la barra rovesciata: `/[$£]/` corrisponde a `$` o a `£`. Per ulteriori informazioni, vedere "Classi di caratteri" a pagina 314.
`|` *(pipe)*	Indica due possibilità alternative: può eseguire la ricerca del valore alla sua sinistra o alla sua destra: `/abc|xyz/` corrisponde a `abc` o a `xyz`.

Informazioni sulle metasequenze

Le metasequenze sono sequenze di caratteri con un significato speciale all'interno delle espressioni regolari. Le metasequenze sono descritte nella tabella seguente:

Metasequenza	Descrizione

`{`*n*`}`
`{`*n*`,}`
e
`{`*n*`,`*n*`}`
	Specifica una quantità numerica o un intervallo di numeri per l'elemento precedente:
	`/A{27}/` corrisponde al carattere `A` ripetuto `27` volte.
	`/A{3,}/` corrisponde al carattere `A` ripetuto `3` o più volte.
	`/A{3,5}/` corrisponde al carattere `A` ripetuto da `3` a `5` volte.
	Per ulteriori informazioni, vedere "Quantificatori" a pagina 316.

`\b`	Corrisponde a elementi tra un carattere di parola e un carattere non di parola. Se il primo o l'ultimo carattere di una stringa è un carattere di parola, corrisponde anche all'inizio o alla fine della stringa.
`B`	Corrisponde a elementi tra due caratteri di parola. Corrisponde anche a elementi tra due caratteri non di parola.
`\d`	Corrisponde a una cifra decimale.
`\D`	Corrisponde a qualsiasi carattere diverso da una cifra.
`\f`	Corrisponde al carattere di avanzamento pagina.

Metasequenza	Descrizione
`\n`	Corrisponde al carattere nuova riga.
`\r`	Corrisponde al carattere ritorno a capo.
`\s`	Corrisponde a qualsiasi carattere spazio bianco (spazio, tabulazione, nuova riga o ritorno a capo).
`\S`	Corrisponde a qualsiasi carattere diverso da spazio bianco.
`\t`	Corrisponde al carattere tabulazione.
`\u`*`nnnn`*	Corrisponde al carattere Unicode con il codice di carattere specificato mediante il numero esadecimale *`nnnn`*. Ad esempio, `\u263a` corrisponde al carattere smiley.
`\v`	Corrisponde al carattere di avanzamento verticale.
`\w`	Corrisponde a un carattere di parola (`A-Z`, `a-z`, `0-9`, o _). Si noti che `\w` non corrisponde a caratteri che non appartengono all'alfabeto inglese come `é`, `ñ` o `ç`.
`\W`	Corrisponde a qualsiasi carattere diverso da un carattere di parola.
`\x`*`nn`*	Corrisponde al carattere con il valore ASCII specificato, come specificato mediante il numero esadecimale *`nn`*.

Classi di caratteri

Le classi di caratteri permettono di specificare una serie di caratteri nelle espressioni regolari. Racchiudere le classi di caratteri tra parentesi angolari (`[` e `]`). Ad esempio, la seguente espressione regolare definisce una classe di caratteri che corrisponde a `bag`, `beg`, `big`, `bog` o `bug`:

```
/b[aeiou]g/
```

Sequenze di escape nelle classi di caratteri

La maggior parte dei metacaratteri e delle metasequenze che hanno un significato speciale nelle espressioni regolari *non* hanno lo stesso significato all'interno della classe di caratteri. Nelle espressioni regolari, ad esempio, l'asterisco indica una ripetizione, ma questo non vale nelle classi di caratteri. Nella seguente classe di caratteri l'asterisco ha il suo significato letterale, come tutti gli altri caratteri:

```
/[abc*123]/
```

Esistono, tuttavia, tre caratteri che le classi di caratteri interpretano come metacaratteri:

Metacarattere	Significato nelle classi di caratteri
`]`	Definisce la fine della classe di caratteri.
`-`	Definisce un intervallo di caratteri (vedere "Intervalli di caratteri nelle classi di caratteri" a pagina 315).
`\`	Definisce metasequenze e disabilita il significato speciale dei metacaratteri.

Per mantenere il significato letterale dei seguenti caratteri, vale a dire per non utilizzarli come metacaratteri, anteporre al carattere la barra rovesciata (il carattere escape). Ad esempio, la seguente espressione regolare comprende una classe di caratteri che corrisponde a uno dei quattro simboli (`$`, `\`, `]` o `-`):

```
/[$\\\]\-]/
```

Oltre ai metacaratteri che mantengono il loro significato speciale, le sequenze seguenti fungono da metasequenze nelle classi di caratteri:

Metasequenza	Significato nelle classi di caratteri
`\n`	Corrisponde al carattere nuova riga.
`\r`	Corrisponde al carattere ritorno a capo.
`\t`	Corrisponde al carattere tabulazione.
`\u`*`nnnn`*	Corrisponde al carattere con il punto di codifica Unicode specificato, come specificato mediante il numero esadecimale *`nnnn`*).
`\x`*`nn`*	Corrisponde al carattere con il valore ASCII specificato, come specificato mediante il numero esadecimale *`nn`*.

Le altre metasequenze e gli altri metacaratteri delle espressioni regolari vengono trattati come caratteri normali all'interno delle classi di caratteri.

Intervalli di caratteri nelle classi di caratteri

Usare il trattino per specificare un intervallo di caratteri, come `A-Z`, `a-z` o `0-9`. I caratteri devono costituire un intervallo valido. La classe di caratteri dell'esempio seguente corrisponde a qualsiasi carattere dell'intervallo `a-z` o a qualsiasi cifra:

```
/[a-z0-9]/
```

In alternativa, è possibile specificare un intervallo per valore ASCII usando il codice del carattere ASCII `\x`*`nn`*. La classe di caratteri dell'esempio seguente corrisponde a qualsiasi carattere del set di caratteri estesi ASCII (come `é` e `ê`):

```
/[\x80-\x9A]/
```

Negazione delle classi di caratteri

L'uso di un accento circonflesso (^) all'inizio di una classe di caratteri nega tale classe, vale a dire che la corrispondenza viene cercata tra qualsiasi carattere ad eccezione di quelli indicati. La seguente classe di caratteri corrisponde a qualsiasi carattere *eccetto* le lettere minuscole (a-z) e le cifre:

```
/[^a-z0-9]/
```

Per utilizzare l'accento circonflesso (^) per indicare una negazione è necessario scriverlo *all'inizio* della classe di caratteri. In caso contrario, l'accento circonflesso mantiene il suo valore letterale e viene aggiunto come normale carattere della classe. La classe di caratteri dell'esempio seguente corrisponde a qualsiasi simbolo, accento circonflesso compreso:

```
/[!.,#+*%$&^]/
```

Quantificatori

I quantificatori consentono di specificare ripetizioni di caratteri singoli o di sequenze di caratteri all'interno dei modelli, come illustrato di seguito:

Metacarattere quantificatore	Descrizione
* (asterisco)	Corrisponde a una ripetizione di zero o più volte del carattere precedente.
+ (più)	Corrisponde a una ripetizione di una o più volte del carattere precedente.
? (punto di domanda)	Corrisponde a una ripetizione di zero o una volta del carattere precedente.
{*n*} {*n*,} e {*n*,*n*}	Specifica una quantità numerica o un intervallo di numeri per l'elemento precedente: /A{27}/ corrisponde al carattere A ripetuto 27 volte. /A{3,}/ corrisponde al carattere A ripetuto 3 o più volte. /A{3,5}/ corrisponde al carattere A ripetuto da 3 a 5 volte.

Un quantificatore può essere applicato a un unico carattere, a una classe di caratteri o a un gruppo:

- /a+/ corrisponde al carattere a ripetuto una o più volte.
- /\d+/ corrisponde a una o più cifre.
- /[abc]+/ corrisponde alla ripetizione di uno o più caratteri a, b o c.
- /(very,)*/ corrisponde alla parola very seguita da una virgola e da uno spazio ripetuto zero o più volte.

I quantificatori possono usati anche all'interno di gruppi parentetici a cui sono applicati dei quantificatori. Nell'esempio seguente, il quantificatore corrisponde a stringhe del tipo `word` e `word-word-word`:

```
/\w+(-\w+)*/
```

Per impostazione predefinita, le espressioni regolari agiscono in modo *greedy*, vale a dire che ogni porzione di modello dell'espressione regolare (come `.*`) cerca una corrispondenza con quanti più caratteri possibili nella stringa prima di lasciare il posto alla porzione successiva dell'espressione. Si consideri, ad esempio, l'espressione regolare e la stringa seguenti:

```
var pattern:RegExp = /<p>.*<\/p>/;
str:String = "<p>Paragraph 1</p> <p>Paragraph 2</p>";
```

L'espressione regolare corrisponde all'intera stringa:

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

Supponiamo, tuttavia, che si desideri individuare una corrispondenza solo per un gruppo `<p>...</p>`. Per ottenere questo risultato, procedere come segue:

```
<p>Paragraph 1</p>
```

Aggiungere un punto di domanda (`?`) dopo i quantificatori che si desidera trasformare in quantificatori *lazy*. L'espressione regolare dell'esempio seguente, che usa il quantificatore lazy `*?`, cerca una corrispondenza con `<p>` seguito dal minimo numero di caratteri possibili, seguito da `</p>`:

```
/<p>.*?<\/p>/
```

Tenere presenti gli aspetti seguenti dei quantificatori:

- I quantificatori `{0}` e `{0,0}` non escludono un elemento da una corrispondenza.
- Non combinare insieme più quantificatori, come `/abc+*/`.
- Il punto (.) non considera più righe, a meno che non sia presente il flag `s` (`dotall`), anche se seguito dal quantificatore `*`. Consideriamo l'esempio del codice seguente:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*<\/p>/;
trace(str.match(re)); // null;

re = /<p>.*<\/p>/s;
trace(str.match(re));
  // output: <p>Test
  //         Multiline</p>
```

Per ulteriori informazioni, vedere "Flag s (dotall)" a pagina 325.

Alternative

Nelle espressioni regolari, il carattere | (pipe) consente di definire delle corrispondenze alternative. Ad esempio, la seguente espressione regolare corrisponde a ognuna delle seguenti parole `cat, dog, pig, rat`:

```
var pattern:RegExp = /cat|dog|pig|rat/;
```

Per definire gruppi allo scopo di restringere l'area di validità del carattere | si possono usare parentesi. La seguente espressione regolare corrisponde a `cat` seguito da `nap` o da `nip`:

```
var pattern:RegExp = /cat(nap|nip)/;
```

Per ulteriori informazioni, vedere "Gruppi" a pagina 318.

Le due espressioni regolari seguenti, una contenente il carattere di alternanza | e l'altra una classe di caratteri (definita mediante `[` e `]`), sono equivalenti:

```
/1|3|5|7|9/
/[13579]/
```

Per ulteriori informazioni, vedere "Classi di caratteri" a pagina 314.

Gruppi

Per specificare un gruppo all'interno di un'espressione regolare si possono usare le parentesi, come nell'esempio seguente:

```
/class-(\d*)/
```

Un gruppo è un segmento di un modulo. I gruppi permettono di:

- Applicare un quantificatore a più caratteri.
- Delineare moduli secondari da applicare alternativamente (usando il carattere `|`).
- Catturare corrispondenze a sottostringhe (ad esempio, usando `\1` in un'espressione regolare per trovare un gruppo trovato in precedenza o usando `$1` in modo simile al metodo `replace()` della classe String).

Le sezioni seguenti forniscono alcuni dettagli sull'uso dei gruppi.

Uso di gruppi e di quantificatori

Se non sono stati definiti dei gruppi, il quantificatore si riferisce al carattere o alla classe di caratteri che lo precede, come illustra l'esempio seguente:

```
var pattern:RegExp = /ab*/ ;
// corrisponde al carattere a seguito da
// zero o più occorrenze del carattere b

pattern = /a\d+/;
// corrisponde al carattere a seguito da
// una o più cifre

pattern = /a[123]{1,3}/;
// corrisponde al carattere a seguito da
// da una a tre occorrenze di 1, 2 o 3
```

Tuttavia, definendo un gruppo è possibile applicare un quantificatore a più di un carattere o a più di una classe di caratteri:

```
var pattern:RegExp = /(ab)*/;
// corrisponde a zero o a più occorrenze del carattere a
// seguito dal carattere b, come in abababab

pattern = /(a\d)+/;
// corrisponde a una o a più occorrenze del carattere a seguito da
// una cifra, come in a1a5a8a3

pattern = /(spam ){1,3}/;
// corrisponde a 1, 2 o 3 occorrenze della parola spam seguita da uno spazio
```

Per ulteriori informazioni sui quantificatori, vedere "Quantificatori" a pagina 316.

Uso dei gruppi con il carattere di alternanza (|)

I gruppi consentono di definire una serie di caratteri a cui applicare il carattere di alternanza (|), nel modo seguente:

```
var pattern:RegExp = /cat|dog/;
// corrisponde a cat o a dog

pattern = /ca(t|d)og/;
// corrisponde a catog o a cadog
```

Uso dei gruppi per rilevare corrispondenze a sottostringhe

Se all'interno di un modello si definisce un gruppo parentetico standard, è possibile aggiungere dei riferimenti al gruppo in un punto successivo dell'espressione regolare. Questo tipo riferimenti sono definiti *a posteriori* e questo tipo di gruppi sono conosciuti come *gruppi di cattura*. Ad esempio, nell'espressione regolare seguente, la sequenza `\1` corrisponde alla sottostringa individuati dal gruppo di cattura parentetico:

```
var pattern:RegExp = /(\d+)-by-\1/;
// corrisponde a: 48-by-48
```

All'interno di un'espressione regolare, si possono specificare fino a 99 riferimenti a posteriori digitando `\1`, `\2`, ... , `\99`.

Analogamente, nel metodo `replace()` della classe String, si può usare `$1-$99` per inserire nella stringa di sostituzione corrispondenze a sottostringhe dei gruppi catturati:

```
var pattern:RegExp = /Hi, (\w+)\./;
var str:String = "Hi, Bob.";
trace(str.replace(pattern, "$1, hello."));
// output: Bob, hello.
```

Inoltre, quando si usano gruppi di cattura, il metodo `exec()` della classe RegExp class e il metodo `match()` della classe String restituiscono sottostringhe che corrispondono ai gruppi di cattura:

```
var pattern:RegExp = /(\w+)@(\w+).(\w+)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
   // bob@test.com,bob,example,com
```

Uso dei gruppi di non cattura e dei gruppi lookahead

Un gruppo di non cattura è un semplice costrutto di raggruppamento che non viene raccolto e fa riferimento a corrispondenze precedenti. Usare `(?:` e `)` per definire i gruppi di non cattura come descritto di seguito:

```
var pattern = /(?:com|org|net);
```

Ad esempio, si noti la differenza che esiste tra inserire `(com|org)` in un gruppo di cattura e inserirlo in un gruppo di non cattura (il metodo `exec()` elenca i gruppi di cattura dopo una corrispondenza completa):

```
var pattern:RegExp = /(\w+)@(\w+).(com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
  // bob@test.com,bob,example,com

//noncapturing:
var pattern:RegExp = /(\w+)@(\w+).(?:com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
  // bob@test.com,bob,example
```

Un tipo speciale di gruppo di non cattura è il *gruppo lookahead* che, a sua volta, si declina in due tipi: *gruppo lookahead positivo* e *gruppo lookahead negativo*.

Per definire un gruppo lookahead positivo, usare `(?=` e `)`, che specifica che il sottomodello del gruppo deve corrispondere alla posizione. Tuttavia, la porzione di stringa che corrisponde al gruppo lookahead positivo può corrispondere agli altri modelli dell'espressione regolare. Ad esempio, dal momento che nel frammento di codice seguente `(?=e)` è un gruppo lookahead positivo, il carattere `e` individuato può essere individuato anche da porzioni successive dell'espressione regolare, in questo caso dal gruppo di cattura `\w*`):

```
var pattern:RegExp = /sh(?=e)(\w*)/i;
var str:String = "Shelly sells seashells by the seashore";
trace(pattern.exec(str));
// Shelly,elly
```

Per definire un gruppo lookahead negativo, usare `(?!` e `)` che specificano che il sottomodello del gruppo *non deve* corrispondere alla posizione. Ad esempio:

```
var pattern:RegExp = /sh(?!e)(\w*)/i;
var str:String = "She sells seashells by the seashore";
trace(pattern.exec(str));
// shore,ore
```

Uso di gruppi denominati

Un gruppo denominato di un'espressione regolare è un gruppo a cui è stato assegnato un nome. Per definire un gruppo denominato, usare `(?P<name>` e `)`. L'espressione regolare dell'esempio seguente contiene il gruppo denominato `digits`:

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

Quando si usa il metodo `exec()`, un gruppo denominato corrispondente viene aggiunto come proprietà dell'array `result`:

```
var myPattern:RegExp = /([a-z]+)(?P<digits>\d+)[a-z]+/;
var str:String = "a123bcd";
var result:Array = myPattern.exec(str);
trace(result.digits); // 123
```

Segue un altro esempio che contiene due gruppi denominati (`name` e `dom`):

```
var emailPattern:RegExp =
  /(?P<name>(\w|[_.\-])+)@(?P<dom>((\w|-)+))+\.\w{2,4}+/;
var address:String = "bob@example.com";
var result:Array = emailPattern.exec(address);
trace(result.name); // bob
trace(result.dom); // esempio
```

NOTA I gruppi denominati non fanno parte della specifica del linguaggio ECMAScript. Sono una funzione speciale di ActionScript 3.0.

Flag e proprietà

La tabella seguente elenca i cinque flag che si possono impostare per le espressioni regolari. È possibile accedere a ogni flag come proprietà dell'oggetto espressione regolare.

Flag	Proprietà	Descrizione
`g`	`global`	Cerca più corrispondenze.
`i`	`ignoreCase`	Cerca corrispondenze ignorando maiuscole e minuscole. Si riferisce ai caratteri `A` - `Z` e `a` - `z` senza includere i caratteri del set esteso (come `É` e `é`).
`m`	`multiline`	Quando questo flag è impostato, `$` e `^` corrispondono, rispettivamente, alla posizione iniziale di una riga e alla posizione finale.
`s`	`dotall`	Quando questo flag è impostato, `.` (punto) può corrispondere al carattere nuova riga (`\n`).
`x`	`extended`	Consente di usare espressioni regolari estese. L'espressione regolare può contenere spazi, che non vengono considerati come parte del modello, al fine di rendere il codice dell'espressione più leggibile.

Si noti che le proprietà sono di sola lettura. Si possono impostare i flag (`g`, `i`, `m`, `s`, `x`) quando si imposta la variabile di un'espressione regolare come segue:

```
var re:RegExp = /abc/gimsx;
```

Non è, tuttavia, possibile impostare direttamente le proprietà denominate. Ad esempio, il codice seguente genera un errore:

```
var re:RegExp = /abc/;
re.global = true; // Questo genera un errore.
```

Per impostazione predefinita, a meno che non siano stati specificati nella dichiarazione dell'espressione regolare, i flag non sono impostati e le proprietà corrispondenti sono impostate su `false`.

Inoltre, esistono altre due proprietà di un'espressione regolare:

- La proprietà `lastIndex` specifica la posizione di indice nella stringa da usare per la chiamata successiva del metodo `exec()` o `test()` dell'espressione regolare.
- La proprietà `source` specifica la stringa che definisce la porzione modello dell'espressione regolare.

Flag g (global)

Se il flag `g` (`global`) *non* è presente, l'espressione regolare ricerca una sola corrispondenza. Ad esempio, se il flag `g` non viene impostato nell'espressione regolare, il metodo `String.match()` restituisce una sola sottostringa corrispondente:

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // output: she
```

Quando il flag `g` è impostato, il metodo `Sting.match()` restituisce più corrispondenze, come nell'esempio seguente:

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/g;
// Stesso modello, ma con il flag g IMPOSTATO.
trace(str.match(pattern)); // output: she,shells,shore
```

Flag i (ignoreCase)

Per impostazione predefinita, le corrispondenze cercate dalle espressioni regolari distinguono tra maiuscole e minuscole. Quando si imposta il flag `i` (`ignoreCase`), la distinzione tra maiuscole e minuscole viene disattivata. Ad esempio, la lettera minuscola `s` dell'espressione regolare non rileva la lettera maiuscola `S`, il primo carattere della stringa:

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // output: 13 -- Il primo carattere è escluso
```

Quando il flag `i` è impostato, invece, l'espressione regolare rileva anche la lettera `S` maiuscola:

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // output: 0
```

Il flag `i` sospende la distinzione tra maiuscole e minuscole solo per i caratteri `A` – `Z` e `a` – `z`, non per i caratteri del set esteso come `É` e `é`.

Flag m (multiline)

Se il flag `m` (`multiline`) non è impostato, `^` corrisponde alla posizione iniziale della stringa e `$` corrisponde alla posizione finale. Quando il flag `m` è impostato, i caratteri corrispondono, rispettivamente, alla posizione iniziale di una riga e alla posizione finale. Si consideri la stringa seguente, che comprende il carattere nuova riga (\n):

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // Corrisponde a una parola all'inizio
                           // della stringa.
```

Anche se il flag `g` (`global`) è impostato nell'espressione regolare, il metodo `match()` ricerca solo una sottostringa, poiché è presente una sola corrispondenza per `^` - l'inizio della stringa. Il risultato è il seguente:

```
Test
```

Ecco lo stesso frammento di codice con il flag `m` impostato:

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // Cerca una parola all'inizio delle righe.
```

Questa volta, il risultato include le parole all'inizio delle due righe:

```
Test,Multiline
```

Si noti che solo il carattere `\n` indica la fine della riga. I caratteri seguenti non indicano la fine della riga:

- Ritorno a capo (`\r`)
- Separatore di riga Unicode (`\u2028`)
- Separatore di paragrafo Unicode (`\u2029`)

Flag s (dotall)

Se il flag `s` (`dotall`) non è impostato, nelle espressioni regolari un punto (`.`) non corrisponde a un carattere nuova riga (`\n`). Di conseguenza, nell'esempio seguente non viene evidenziata una corrispondenza:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

Tuttavia, se il flag `s` è impostato, il punto trova il carattere nuova riga:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

In questo caso, la corrispondenza riguarda l'intera sottostringa racchiusa tra i tag `<p>`, che comprende anche il carattere nuova riga:

```
<p>Test
Multiline</p>
```

Flag x (extended)

Le espressioni regolari possono risultare abbastanza oscure, specialmente se contengono molti metacaratteri e metasequenze. Ad esempio:

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

L'uso del flag `x` (`extended`) all'interno di un'espressione regolare permette di aggiungere spazi bianchi che non vengono considerati come parte del modello. L'espressione regolare dell'esempio seguente è identica a quella dell'esempio precedente:

```
/     <p    (>  |  (\s*  [^>]*  >))    .*?    <\/p>  /gix
```

Se un'espressione regolare comprende un flag `x` e si desidera cercare il valore letterale di uno spazio vuoto, anteporre allo spazio una barra rovesciata. Ad esempio, le due espressioni regolari seguenti sono equivalenti:

```
/foo bar/
/foo \ bar/x
```

Proprietà lastIndex

La proprietà `lastIndex` specifica la posizione di indice nella stringa in corrispondenza della quale cominciare la ricerca successiva. La proprietà influisce sui metodi `exec()` e `test()` chiamati da un'espressione regolare in cui il flag `g` è impostato su `true`. Consideriamo l'esempio del codice seguente:

```
var pattern:RegExp = /p\w*/gi;
var str:String = "Pedro Piper picked a peck of pickled peppers.";
trace(pattern.lastIndex);
var result:Object = pattern.exec(str);
while (result != null)
{
  trace(pattern.lastIndex);
  result = pattern.exec(str);
}
```

Per impostazione predefinita, la proprietà `lastIndex` è impostata su 0 (per cominciare le ricerche all'inizio della stringa). Dopo ogni corrispondenza rilevata, viene portata alla posizione di indice successiva. Il risultato per il frammento di codice precedente sarà, pertanto, il seguente:

```
0
5
11
18
25
36
44
```

Se il flag `global` è impostato su `false`, i metodi `exec()` e `test()` non usano né impostano la proprietà `lastIndex`.

I metodi `match()`, `replace()` e `search()` della classe String avviano tutte le ricerche dall'inizio della stringa, indipendentemente dall'impostazione della proprietà `lastIndex` dell'espressione regolare usata nella chiamata al metodo. (Tuttavia, il metodo `match()` imposta `lastIndex` su 0.)

È possibile impostare la proprietà `lastIndex` per regolare la posizione iniziale nella stringa per la corrispondenza dell'espressione regolare.

Proprietà source

La proprietà `source` specifica la stringa che definisce la porzione modello dell'espressione regolare. Ad esempio:

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

Metodi di impiego di espressioni regolari con stringhe

La classe RegExp comprende due metodi: `exec()` e `test()`.

Oltre ai metodi `exec()` e `test()` della classe RegExp, la classe String comprende i metodi seguenti per cercare espressioni regolari nelle stringhe: `match()`, `replace()`, `search()` e `splice()`.

Metodo test()

Il metodo `test()` della classe RegExp verifica se la stringa contiene una corrispondenza dell'espressione regolare, come illustra l'esempio seguente:

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // output: true
```

Metodo exec()

Il metodo `exec()` della classe RegExp verifica se la stringa contiene una corrispondenza dell'espressione regolare e restituisce un array con:

- La sottostringa di cui si è trovata la corrispondenza
- Sottostringhe che contengono corrispondenze ai gruppi parentetici dell'espressione regolare

L'array comprende anche la proprietà `index`, che indica la posizione di indice dell'inizio della corrispondenza nella sottostringa.

Consideriamo l'esempio del codice seguente:

```
var pattern:RegExp = /\d{3}\-\d{3}-\d{4}/; //Numero di telefono USA
var str:String = "phone: 415-555-1212";
var result:Array = pattern.exec(str);
trace(result.index, " - ", result);
// 7  -  415-555-1212
```

Usare il metodo `exec()` più volte per cercare più sottostringhe quando il flag `g` (`global`) è impostato nell'espressione regolare:

```
var pattern:RegExp = /\w*sh\w*/gi;
var str:String = "She sells seashells by the seashore";
var result:Array = pattern.exec(str);

while (result != null)
{
  trace(result.index, "\t", pattern.lastIndex, "\t", result);
  result = pattern.exec(str);
}
// output:
// 0  3  She
// 10  19  seashells
// 27  35  seashore
```

Metodi della classe String che usano parametri RegExp

I seguenti metodi della classe String accettano le espressioni regolari come parametri: `match()`, `replace()`, `search()` e `split()`. Per ulteriori informazioni su questi metodi, vedere "Ricerca di modelli nelle stringhe e sostituzione di sottostringhe" a pagina 226.

Esempio: Un parser Wiki

Questo esempio di conversione di un semplice testo Wiki illustra alcune applicazioni delle espressioni regolari:

- Conversione di righe di testo che associano un modello Wiki di origine alle stringhe di output HTML appropriate.
- Uso di un'espressione regolare per convertire modelli URL in tag di collegamento ipertestuale HTML `<a>`.
- Uso di un'espressione regolare per convertire stringhe in dollari USA (come `"$9.95"`) in stringhe in euro (come `"8.24 €"`).

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione WikiEditor si trovano nella cartella Samples/WikiEditor. L'applicazione comprende i seguenti file:

File	Descrizione
WikiEditor.mxml o WikiEditor.fla	Il file principale dell'applicazione in Flash (FLA) o Flex (MXML)
com/example/programmingas3/regExpExamples/WikiParser.as	Classe che comprende metodi che usano espressioni regolari per convertire modelli di testo di Wiki nell'output HTML equivalente.
com/example/programmingas3/regExpExamples/URLParser.as	Classe che comprende metodi che usano espressioni regolari per convertire stringhe URL in tag di collegamento ipertestuale HTML `<a>`.
com/example/programmingas3/regExpExamples/CurrencyConverter.as	Classe che comprende metodi che usano espressioni regolari per convertire stringhe in dollari in stringhe in euro.

Definizione della classe WikiParser

La classe WikiParser comprende metodi che convertono testo di Wiki nell'output HTML equivalente. Non si tratta di un'applicazione di conversione Wiki particolarmente versatile, ma illustra degnamente alcuni impieghi delle espressioni regolari per la ricerca di modelli e la conversione di stringhe.

La funzione di costruzione, insieme al metodo `setWikiData()`, si limita a inizializzare una stringa campione di testo Wiki nel modo seguente:

```
public function WikiParser()
{
  wikiData = setWikiData();
}
```

Quando l'utente seleziona il pulsante Test dell'applicazione, l'applicazione chiama il metodo `parseWikiString()` dell'oggetto WikiParser. Questo metodo chiama, a sua volta, altri metodi che si occupano di assemblare la stringa HTML convertita.

```
public function parseWikiString(wikiString:String):String
{
  var result:String = parseBold(wikiString);
  result = parseItalic(result);
  result = linesToParagraphs(result);
  result = parseBullets(result);
  return result;
}
```

Ognuno dei metodi chiamati, `parseBold()`, `parseItalic()`, `linesToParagraphs()` e `parseBullets()`, usa il metodo `replace()` della stringa per sostituire le corrispondenze dei modelli individuate, definite da un'espressione regolare, allo scopo di trasformare il testo Wiki in testo HTML.

Conversione dei modelli in grassetto e corsivo

Il metodo `parseBold()` cerca un modello di testo grassetto Wiki (come `'''foo'''`) e lo converte nell'equivalente HTML (come `<b>foo</b>`) nel modo seguente:

```
private function parseBold(input:String):String
{
  var pattern:RegExp = /'''(.*?)'''/g;
  return input.replace(pattern, "<b>$1</b>");
}
```

Si noti che la porzione `(.?*)` dell'espressione regolare cerca qualsiasi numero di caratteri (*) tra i due modelli di definizione `'''`. Il quantificatore `?` rende la corrispondenza "lazy", vale a dire che la prima corrispondenza trovata per la stringa `'''aaa''' bbb '''ccc'''` sarà `'''aaa'''` e non l'intera stringa (che inizia e finisce con `'''`).

Le parentesi all'interno dell'espressione regolare definiscono un gruppo di cattura e il metodo `replace()` fa riferimento a tale gruppo mediante il codice `$1` della stringa di sostituzione. Il flag `g` (`global`) dell'espressione regolare fa in modo che il metodo `replace()` sostituisca tutte le corrispondenze nella stringa (e non solo la prima).

Il metodo `parseItalic()` opera in modo analogo al metodo `parseBold()`, con l'eccezione che cerca due, e non tre, apostrofi (`''`) come delimitatori del testo in corsivo:

```
private function parseItalic(input:String):String
{
  var pattern:RegExp = /''(.*?)''/g;
  return input.replace(pattern, "<i>$1</i>");
}
```

Conversione di modelli puntati

Come illustra l'esempio seguente, il metodo `parseBullet()` cerca il modello riga puntata Wiki (come `* foo`) e lo converte nel suo equivalente HTML (come `<li>foo</li>`):

```
private function parseBullets(input:String):String
{
  var pattern:RegExp = /^\*(.*)/gm;
  return input.replace(pattern, "<li>$1</li>");
}
```

Il simbolo ^ all'inizio dell'espressione regolare corrisponde alla posizione iniziale di una riga. Il flag m (multiline) dell'espressione regolare fa in modo che il simbolo ^ corrisponda alla posizione iniziale di una riga, non semplicemente di una stringa.

Il modello * cerca il carattere asterisco (la barra rovesciata segnala l'impiego del valore letterale dell'asterisco e non del suo valore di quantificatore *).

Le parentesi all'interno dell'espressione regolare definiscono un gruppo di cattura e il metodo replace() fa riferimento a tale gruppo mediante il codice $1 della stringa di sostituzione. Il flag g (global) dell'espressione regolare fa in modo che il metodo replace() sostituisca tutte le corrispondenze nella stringa (e non solo la prima).

Conversione modelli Wiki di paragrafo

Il metodo linesToParagraphs() converte ogni riga della stringa di input Wiki in un tag di paragrafo HTML <p>. Queste righe del metodo eliminano le righe vuote dalla stringa di input Wiki:

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

I simboli ^ e $ dell'espressione regolare corrispondono alla posizione iniziale e finale di una riga. Il flag m (multiline) nell'espressione regolare fa in modo che il simbolo ^ corrisponda alla posizione iniziale di una riga, non semplicemente di una stringa.

Il metodo replace() sostituisce tutte le sottostringhe individuate (righe vuote) con una stringa vuota (""). Il flag g (global) dell'espressione regolare fa in modo che il metodo replace() sostituisca tutte le corrispondenze nella stringa (e non solo la prima).

Conversione degli URL in tag HTML <a>

Quando l'utente seleziona il pulsante Test dell'applicazione, se la casella urlToATag è selezionata, l'applicazione chiama il metodo statico URLParser.urlToATag() per convertire le stringhe URL della stringa di input Wiki in tag HTML <a>.

```
var protocol:String = "((?:http|ftp)://)";
var urlPart:String = "([a-z0-9_-]+\.[a-z0-9_-]+)";
var optionalUrlPart:String = "(\.[a-z0-9_-]*)";
var urlPattern:RegExp = new RegExp(protocol + urlPart + optionalUrlPart,
                    "ig");
var result:String = input.replace(urlPattern,
                "<a href='$1$2$3'><u>$1$2$3</u></a>");
```

La funzione di costruzione RegExp() viene usata per creare un'espressione regolare (urlPattern) assemblando vari componenti costituenti. Questi componenti sono le singole stringhe che definiscono una porzione del modello dell'espressione regolare.

La prima porzione del modello dell'espressione regolare, definito dalla stringa `protocol`, definisce il protocollo URL che può essere `http://` o `ftp://`. Le parentesi definiscono un gruppo di non cattura, indicato dal simbolo `?`. Ciò significa che le parentesi definiscono semplicemente un gruppo per il modello di alternanza `|` e che il gruppo non cerca riferimenti a posteriori (`$1`, `$2`, `$3`) nella stringa di sostituzione del metodo `replace()`.

Tutti gli altri componenti costituenti dell'espressione regolare usano gruppi di cattura (indicati dalle parentesi nel modello) che, a loro volta, vengono usati nei riferimenti a posteriori (`$1`, `$2`, `$3`) nella stringa di sostituzione del metodo `replace()`.

La porzione di modello definita dalla stringa `urlPart` corrisponde ad *almeno* uno dei seguenti caratteri: `a-z`, `0-9`, _ o `-`. Il quantificatore + indica che è stata individuata la corrispondenza ad almeno un carattere. `\.` indica un punto obbligatorio (`.`). Il resto corrisponde a un'altra stringa con almeno uno dei seguenti caratteri: `a-z`, `0-9`, _ o `-`.

La porzione di modello definita dalla stringa `optionalUrlPart` corrisponde a *zero o più* elementi seguenti: un punto (`.`) seguito da un numero qualsiasi di caratteri alfanumerici (compresi i caratteri _ e `-`). Il quantificatore * indica che è stata individuata la corrispondenza a zero o più caratteri.

La chiamata al metodo `replace()` impiega l'espressione regolare e assembla la stringa HTML di sostituzione servendosi di riferimenti a posteriori.

Il metodo `urlToATag()` chiama, quindi, il metodo `emailToATag()` che usa una tecnica simile per convertire modelli e-mail con collegamenti ipertestuali HTML `<a>`. Le espressioni regolari usate in questo file di esempio per trovare gli URL HTTP, FTP ed e-mail sono relativamente semplici per motivi di chiarezza. In realtà, esistono espressioni regolari molto più complesse che permettono di individuare gli URL con una maggiore precisione.

Conversione delle stringhe in dollari USA in stringhe in euro

Quando l'utente seleziona il pulsante Test dell'applicazione, se la casella `dollarToEuro` è selezionata, l'applicazione chiama il metodo statico `CurrencyConverter.usdToEuro()` per convertire le stringhe in dollari (come `"$9.95"`) in stringhe in euro (come `"8.24 €"`) nel modo seguente:

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

La prima riga definisce un semplice modello per la ricerca delle stringhe in dollari USA. Si noti che il carattere `$` è preceduto dal carattere di escape barra rovesciata (`\`).

Il metodo `replace()` usa l'espressione regolare come motore di ricerca del modello e chiama la funzione `usdStrToEuroStr()` per determinare la stringa di sostituzione, cioè il valore corrispondente in euro.

Quando il nome di una funzione viene usato come secondo parametro del metodo `replace()`, i seguenti elementi vengono passati come parametri della funzione chiamata:

- La porzione corrispondente della stringa.
- Qualsiasi corrispondenza a gruppi parentetici catturata. Il numero di argomenti passati in questo modo varia a seconda del numero di corrispondenze a gruppi parentetici catturate. È possibile determinare il numero di corrispondenze a gruppi parentetici catturate verificando `arguments.length - 3` all'interno del codice della funzione.
- La posizione di indice nella stringa in cui inizia la corrispondenza.
- La stringa completa.

Il metodo `usdStrToEuroStr()` converte le stringhe in dollari USA in stringhe in euro nel modo seguente:

```
private function usdToEuro(...args):String
{
  var usd:String = args[1];
  usd = usd.replace(",", "");
  var exchangeRate:Number = 0.828017;
  var euro:Number = Number(usd) * exchangeRate;
  trace(usd, Number(usd), euro);
  const euroSymbol:String = String.fromCharCode(8364); // €
  return euro.toFixed(2) + " " + euroSymbol;
}
```

Si noti che `args[1]` rappresenta il gruppo parentetico catturato trovato dall'espressione regolare `usdPrice`. Questa costituisce il segmento numerico della stringa in dollari: vale a dire l'importo in dollari senza il simbolo `$`. Il metodo applica un tasso di cambio per la conversione e restituisce un risultato numerico (seguito dal simbolo `€` anziché preceduto dal simbolo $).

CAPITOLO 10

Gestione degli eventi

10

Un sistema di gestione degli eventi permette ai programmatori di elaborare risposte adeguate all'input degli utenti e agli eventi del sistema. Il modello di gestione degli eventi di ActionScript 3.0 non è solo comodo, ma anche conforme agli standard del settore, oltre che strettamente integrato con il nuovo elenco di visualizzazione di Adobe Flash Player 9. Questo modello di eventi, che si basa sulla specifica DOM (Document Object Model) Level 3, un'architettura di gestione degli eventi considerata uno standard del settore, è uno strumento di gestione degli eventi efficace e al contempo intuitivo dedicato ai programmatori ActionScript.

Il presente capitolo è suddiviso in cinque sezioni. Le prime due sezioni contengono le nozioni di base relative alla gestione degli eventi in ActionScript. Le ultime tre sezioni affrontano gli aspetti principali del modello di gestione degli eventi: il flusso di eventi, gli oggetti evento e i listener di eventi. Poiché il sistema di gestione degli eventi di ActionScript 3.0 interagisce in modo molto stretto con l'elenco di visualizzazione, si presuppone che il lettore abbia una conoscenza di base dell'elenco di visualizzazione. Per ulteriori informazioni, vedere "Programmazione degli elementi visivi" a pagina 395.

Sommario

Nozioni di base sulla gestione degli eventi

Introduzione alla gestione degli eventi

Si considera un evento una qualsiasi attività che avviene nel file SWF e che riveste un qualche interesse per il programmatore. Ad esempio, la maggior parte dei file SWF supportano qualche tipo di interazione con l'utente, dalla semplice risposta a un clic del mouse fino a reazioni molto più complesse, come accettare ed elaborare i dati inseriti in un modulo. Tutte le interazioni di questo tipo che avvengono tra l'utente e il file SWF sono considerate eventi. Inoltre, si possono verificare degli eventi anche in assenza di interventi diretti da parte dell'utente. Esempi di queste occorrenze sono quando termina il caricamento di dati da un server o quando una fotocamera collegata al computer diventa attiva.

In ActionScript 3.0, ogni evento è rappresentato da un oggetto evento, cioè da un'istanza della classe Event o di una delle sue sottoclassi. Oltre a memorizzare informazioni relative a un evento specifico, un oggetto evento contiene metodi che facilitano la manipolazione dell'oggetto stesso. Quando Flash Player rileva un clic del mouse, ad esempio, l'applicazione crea un oggetto evento che rappresenti quel particolare evento clic del mouse.

Dopo aver creato un oggetto evento, Flash Player *invia* l'oggetto, vale a dire lo passa, o trasmette, all'oggetto che costituisce il destinatario dell'evento. Un oggetto che funge da destinazione per un oggetto evento inviato viene definito *destinatario dell'evento*. Ad esempio, quando una fotocamera collegata diventa attiva, Flash Player invia un oggetto evento direttamente all'oggetto che rappresenta la fotocamera, vale a dire al destinatario dell'evento. Tuttavia, se il destinatario dell'evento è presente nell'elenco di visualizzazione, l'oggetto evento viene fatto scorrere attraverso la gerarchia dell'elenco di visualizzazione finché non raggiunge il destinatario designato. In alcuni casi, l'oggetto evento ripercorre il suo cammino in senso inverso e risale la gerarchia dell'elenco di visualizzazione. Questo attraversamento nei due sensi della gerarchia dell'elenco di visualizzazione è denominato *flusso di eventi*.

L'aggiunta di listener di eventi al codice permette di "catturare" gli oggetti evento. I *listener di eventi* sono infatti funzioni o metodi che il programmatore scrive per rispondere a eventi specifici. Per fare in modo che il programma risponda agli eventi, è necessario aggiungere dei listener di eventi al destinatario dell'evento o agli oggetti dell'elenco di visualizzazione che fanno parte del flusso di eventi dell'oggetto.

Il codice per la creazione di listener di eventi deve sempre attenersi a questa struttura di base (gli elementi in grassetto sono segnaposti modificabili a seconda del caso specifico):

```
function eventResponse(eventObject:EventType):void
{
  // Le azioni di risposta agli eventi vanno definite qui.
}
```

eventTarget.`addEventListener(`**EventType**.**EVENT_NAME**, **eventResponse**`);`

Questo esempio di codice svolge due compiti. Definisce una funzione, vale a dire che specifica le azioni che devono avvenire in risposta all'evento e, in secondo luogo, chiama il metodo `addEventListener()` dell'oggetto source, in pratica associando la funzione all'evento, in modo che quando si verifica l'evento vengono attivate le azioni descritte nella funzione. Quando l'evento si verifica, il destinatario dell'evento esamina l'elenco delle funzioni e dei metodi registrati come listener dell'evento e chiama ognuno passando l'oggetto evento come parametro.

Per creare il proprio listener di eventi, ci sono quattro valori da modificare in questo codice. Per cominciare, sostituire il nome della funzione con il nome che si desidera usare (la modifica deve essere applicata in due punti in corrispondenza di `eventResponse`). In secondo luogo, specificare il nome della classe dell'oggetto inviato dall'evento che si intende monitorare (`EventType` nel codice) e specificare la costante appropriata per l'evento specifico (`EVENT_NAME` nell'elenco). Quindi, effettuare una chiamata al metodo `addEventListener()` sull'oggetto che invierà l'evento (`eventTarget` nel codice). Infine, in via facoltativa, si può cambiare il nome della variabile utilizzata come parametro della funzione (`eventObject` nel codice).

Operazioni comuni per la gestione degli eventi

Le operazioni più comuni per la gestione degli eventi descritte in questo capitolo sono le seguenti:

- Scrittura di codice per rispondere agli eventi
- Disattivazione della risposta agli eventi
- Operazioni con oggetti evento
- Operazioni con il flusso di eventi:
  - Identificazione delle informazioni sul flusso di eventi
  - Arresto del flusso di eventi
  - Arresto del comportamento predefinito
- Invio di eventi dalle classi
- Creazione di un tipo di evento personalizzato

Concetti e termini importanti

L'elenco seguente contiene i termini più importanti che vengono citati in questo capitolo:

- Comportamento predefinito: alcuni eventi sono configurati con un comportamento attivato automaticamente denominato comportamento predefinito. Ad esempio, se un utente digita un testo in un campo di testo, viene attivato un evento text input. Il comportamento predefinito di questo evento consiste nel visualizzare il carattere digitato sulla tastiera all'interno del campo di testo. Tale comportamento può essere disattivato se per qualche ragione non si desidera visualizzare sullo schermo il testo digitato.
- Invio: notifica spedita ai listener di eventi per comunicare che si è verificato un evento.
- Evento: qualsiasi avvenimento che si verifica su un oggetto e di cui l'oggetto può informare altri oggetti.
- Flusso di eventi: quando si verifica un evento su un oggetto dell'elenco di visualizzazione, cioè su un oggetto visualizzato sullo schermo, tutti gli oggetti che contengono quell'oggetto vengono informati dell'evento e, a loro volta, notificano i rispettivi listener di eventi. Questo processo parte dallo Stage e prosegue lungo l'elenco di visualizzazione fino a raggiungere l'oggetto in cui si è verificato l'evento e poi riparte per tornare allo Stage. Tale processo è definito flusso di eventi.
- Oggetto evento: oggetto che contiene informazioni sull'occorrenza di un particolare evento e che viene inviato a tutti i listener nel momento in cui un evento viene inviato.
- Destinatario dell'evento: oggetto che esegue l'invio dell'evento. Ad esempio, se l'utente seleziona un pulsante all'interno di un oggetto Sprite che a sua volta si trova sullo Stage, tutti questi oggetti inviano eventi, ma l'unico destinatario dell'evento è quello in cui l'evento si è verificato, vale a dire il pulsante selezionato.
- Listener: oggetto o funzione registrato presso l'oggetto a indicare che deve essere notificato se si verifica un evento specifico.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Tutti questi esempi includono la chiamata appropriata alla funzione `trace()`, che consente di provare i risultati del codice. Per provare gli esempi di codice contenuti in questo capitolo:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.

4. Eseguire il programma selezionando Controllo > Prova filmato.

   Nel pannello Output vengono visualizzati i risultati della funzione `trace()` dell'esempio di codice.

Alcuni degli esempi di codice sono più complessi e sono scritti come una classe. Per provare questi esempi:

1. Creare un documento Flash vuoto e salvarlo nel computer.
2. Creare un nuovo file ActionScript e salvarlo nella stessa directory del documento Flash. Il nome file deve corrispondere al nome della classe presente nell'esempio di codice. Ad esempio, se l'esempio di codice definisce una classe chiamata EventTest, per salvare il file ActionScript, utilizzare il nome EventTest.as.
3. Copiare l'esempio di codice nel file ActionScript e salvare il file.
4. Nel documento Flash, fare clic in una parte vuota dello stage oppure dell'area di lavoro per attivare la finestra di ispezione Proprietà del documento.
5. Nella finestra di ispezione Proprietà, nel campo Classe documento inserire il nome della classe ActionScript copiata dal testo.
6. Eseguire il programma selezionando Controllo > Prova filmato.

   I risultati dell'esempio vengono visualizzati nel pannello Output.

Per ulteriori informazioni su queste tecniche per la prova degli esempi di codice, vedere "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Novità della gestione degli eventi di ActionScript 3.0 rispetto alle versioni precedenti

La differenza più sostanziale tra la modalità di gestione degli eventi di ActionScript 3.0 e la modalità delle versioni precedenti dell'applicazione è rappresentata dal fatto che ActionScript 3.0 contiene un solo sistema di gestione degli eventi unificato, mentre le versioni precedenti di ActionScript comprendevano modalità diverse. La presente sezione descrive a grandi linee come avveniva la gestione degli eventi nelle versioni precedenti di ActionScript e quindi passa a descrivere la nuova modalità implementata in ActionScript 3.0.

Gestione degli eventi nelle versioni precedenti di ActionScript

Le versioni di ActionScript anteriori alla 3.0 mettevano a disposizione dei programmatori varie modalità di gestione degli eventi:

- Gestori di eventi `on()` che venivano inseriti direttamente sulle istanze di Button e MovieClip
- Gestori di eventi `onClipEvent()` che venivano inseriti direttamente sulle istanze di MovieClip
- Proprietà della funzione Callback, come `XML.onload` e `Camera.onActivity`
- Listener di eventi da registrare tramite il metodo `addListener()`
- La classe UIEventDispatcher che implementava parzialmente il modello eventi DOM.

Ognuno di questi meccanismi presentava dei vantaggi e dei limiti. I gestori di eventi `on()` e `onClipEvent()`, ad esempio, sono facili da usare, ma rendono complicata la gestione dei progetti completati perché il codice inserito direttamente sui pulsanti e filmati clip diventa difficile da reperire. Anche le funzioni callback sono di facile implementazione, ma ogni evento ne supporta solo una. I listener di eventi, d'altro canto, sono difficili da implementare perché non solo richiedono la creazione dell'oggetto listener e della relativa funzione, ma anche la registrazione del listener per l'oggetto che genera l'evento. A questo maggiore carico di lavoro corrisponde, però, la possibilità di creare vari oggetti listener e registrarli tutti per lo stesso evento.

Lo sviluppo di componenti per ActionScript 2.0 ha generato un ulteriore modello di eventi. Il nuovo modello, incorporato nella classe UIEventDispatcher, si ispirava alla specifica di eventi DOM. Gli sviluppatori che hanno dimestichezza con la gestione degli eventi dei componenti troveranno il passaggio al nuovo modello ideato per ActionScript 3.0 relativamente indolore.

Sfortunatamente, la sintassi usata dai vari modelli di eventi è sovrapponibile per vari aspetti ma differisce in alcuni punti. Ad esempio, in ActionScript 2.0 alcune proprietà, come `TextField.onChanged`, possono essere usate come funzione callback o listener di eventi. Tuttavia, la sintassi per registrare gli oggetti listener differisce a seconda che si usi una delle sei classi che supportano i listener o la classe UIEventDispatcher. Per le classi Key, Mouse, MovieClipLoader, Selection, Stage e TextField si usa il metodo `addListener()`, ma per la gestione degli eventi dei componenti si usa il metodo `addEventListener()`.

Un'ulteriore difficoltà introdotta dai vari modelli di gestione degli eventi è che l'area di validità della funzione di gestione differiva notevolmente a seconda del meccanismo utilizzato. In altre parole, il significato della parola chiave `this` non era costante nei vari sistemi di gestione degli eventi.

Gestione degli eventi in ActionScript 3.0

Con ActionScript 3.0 viene introdotto un unico modello di gestione degli eventi che sostituisce i vari meccanismi disponibili nelle versioni precedenti dell'applicazione. Il nuovo modello di gestione si basa sulla specifica DOM (Document Object Model) Level 3. Nonostante il formato dei file SWF non aderisca espressamente allo standard DOM, il livello di somiglianza tra l'elenco di visualizzazione e la struttura DOM è sufficiente per rendere possibile l'implementazione del modello DOM. Gli oggetti dell'elenco di visualizzazione sono analoghi ai nodi DOM nella struttura gerarchica e i termini *oggetto dell'elenco di visualizzazione* e *nodo* in questo documento sono interscambiabili.

L'implementazione in Flash Player del modello eventi DOM introduce un nuovo concetto: i comportamenti predefiniti. Un *comportamento predefinito* è un'azione che Flash Player esegue come normale conseguenza di determinati eventi.

Comportamenti predefiniti

Generalmente è compito degli sviluppatori scrivere il codice che attiva la risposta agli eventi. In alcuni casi, tuttavia, un comportamento è così "naturalmente" associato a un evento che Flash Player attiva automaticamente il comportamento, a meno che lo sviluppatore non modifichi il codice in modo da annullarlo. In quanto attivati automaticamente da Flash Player, questi comportamenti vengono definiti predefiniti.

Ad esempio, quando un utente inserisce un testo in un oggetto TextField, l'aspettativa che il testo venga visualizzato in quel particolare oggetto TextField è così alta che è sembrato appropriato impostare questo comportamento come predefinito in Flash Player. Se si desidera disattivare un comportamento predefinito, è sufficiente annullarlo utilizzando il nuovo sistema di gestione degli eventi. Quando un utente inserisce un testo in un oggetto TextField, Flash Player crea un'istanza della classe TextEvent che rappresenti l'input dell'utente. Per impedire che Flash Player visualizzi il testo nel campo TextField, è necessario accedere a quella precisa istanza di TextEvent e chiamare il metodo `preventDefault()` dell'istanza.

Non tutti i comportamenti predefiniti possono essere disattivati. Ad esempio, quando un utente fa doppio clic su una parola di un oggetto TextField, Flash Player genera un oggetto MouseEvent. Il comportamento predefinito che viene attivato, e che non può in alcun modo essere disattivato, prevede che la parola sotto il cursore sia evidenziata.

A molti tipi di oggetto evento non sono stati associati dei comportamenti predefiniti. Ad esempio, Flash Player invia un oggetto evento connect quando si attiva una connessione di rete, ma a questo oggetto non è associato nessun comportamento predefinito. La documentazione API per la classe Event e le sue sottoclassi elenca tutti i tipi di eventi, descrive il comportamento predefinito associato a ognuno e indica se il comportamento può essere disattivato.

È importante capire che i comportamenti predefiniti sono associati solo a oggetti evento inviati da Flash Player mentre non esistono per oggetti evento inviati dal codice tramite ActionScript. Ad esempio, è consentito usare i metodi della classe EventDispatcher per inviare un oggetto evento del tipo `textInput`, ma a tale oggetto non sarà associato un comportamento predefinito. In altre parole, Flash Player non visualizza un carattere in un oggetto TextField come risultato di un evento `textInput` inviato a livello di codice.

Novità per i listener di eventi in ActionScript 3.0

Gli sviluppatori che hanno dimestichezza con l'impiego del metodo `addListener()` di ActionScript 2.0 possono trovare utili le informazioni seguenti, che sottolineano le differenze tra il modello listener di eventi di ActionScript 2.0 e il modello eventi di ActionScript 3.0. Ecco le principali differenze che esistono tra i due approcci:

- Per aggiungere listener di eventi in ActionScript 2.0 si usa talvolta `addListener()` e talvolta `addEventListener()`, mentre in ActionScript 3.0 si usa `addEventListener()` in tutte le situazioni.
- In ActionScript 2.0 non esiste il flusso di eventi, il che significa che il metodo `addListener()` può essere chiamato soltanto sull'oggetto che trasmette l'evento mentre in ActionScript 3.0, il metodo `addEventListener()` può essere chiamato su qualsiasi oggetto che fa parte del flusso di eventi.
- In ActionScript 2.0, i listener di eventi possono essere funzioni, metodi oppure oggetti mentre in ActionScript 3.0 possono essere listener di eventi solo funzioni o metodi.

Il flusso di eventi

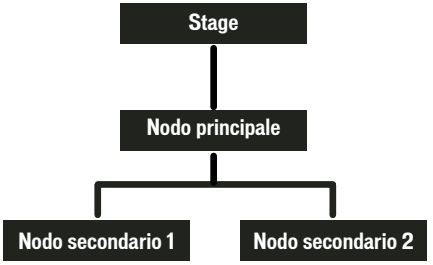
Ogni volta che si verifica un evento, Flash Player invia un oggetto evento. Se il destinatario dell'evento non fa parte dell'elenco di visualizzazione, Flash Player trasmette l'oggetto evento direttamente al destinatario dell'evento. Ad esempio, Flash Player invia l'oggetto progress direttamente a un oggetto URLStream. Se invece il destinatario dell'evento fa parte dell'elenco di visualizzazione, Flash Player invia l'oggetto evento all'elenco di visualizzazione e l'oggetto percorre l'elenco di visualizzazione fino ad arrivare al destinatario dell'evento.

Il *flusso di eventi* descrive il modo in cui l'oggetto evento percorre l'elenco di visualizzazione. L'elenco di visualizzazione è organizzato come una struttura gerarchica. In cima all'elenco si trova lo Stage, uno speciale contenitore di oggetti di visualizzazione che funge da radice dell'elenco di visualizzazione. Lo Stage è rappresentato dalla classe flash.display.Stage ed è accessibile solo attraverso un oggetto di visualizzazione. Ogni oggetto di visualizzazione è dotato di una proprietà denominata `stage` che fa riferimento allo Stage per quella particolare applicazione.

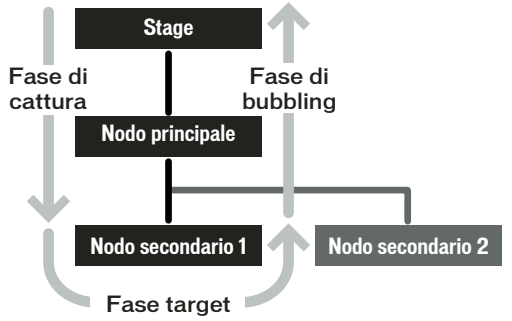
Quando Flash Player invia un oggetto evento, l'oggetto esegue un tragitto circolare dallo Stage al *nodo target*. La specifica degli eventi DOM definisce nodo target il nodo che rappresenta il destinatario dell'evento. In altre parole, il nodo target è l'oggetto dell'elenco di visualizzazione in cui si è verificato l'evento. Ad esempio, se un utente fa clic sull'oggetto `child1` dell'elenco di visualizzazione, Flash Player invierà un oggetto evento usando `child1` come nodo target.

Il flusso di eventi può essere suddiviso in tre parti. La prima parte si chiama fase di cattura e comprende tutti i nodi dallo Stage al livello superiore del nodo target. La seconda parte si chiama fase target e comprende solo il nodo target. La terza parte si chiama fase di bubbling e comprende i nodi incontrati nel tragitto inverso, cioè dal livello superiore del nodo target allo Stage.

I nomi delle fasi acquistano più senso se si concepisce l'elenco di visualizzazione come una struttura gerarchica verticale con lo Stage al vertice, come illustrato nel diagramma seguente:



Se un utente fa clic su `Child1 Node`, Flash Player invia un oggetto evento nel flusso di eventi. Come dimostra l'immagine seguente, il tragitto dell'oggetto parte da `Stage`, attraversa `Parent Node` e arriva a `Child1 Node`, per poi ripartire verso `Stage` attraversando di nuovo `Parent Node` e arrestarsi a `Stage`.



In questo esempio, la fase di cattura comprende `Stage` e `Parent Node` durante il tragitto verso il basso iniziale. La fase target comprende il tempo trascorso in corrispondenza di `Child1 Node`. La fase di bubbling comprende `Parent Node` e `Stage` che vengono incontrati durante il tragitto contrario intrapreso per tornare al nodo principale.

Il flusso di eventi contribuisce a rendere il sistema di gestione degli eventi più efficace per i programmatori ActionScript. Dal momento che nelle versioni precedenti di ActionScript il flusso di eventi non esiste, i listener di eventi possono essere aggiunti solo all'oggetto che genera l'evento. In ActionScript 3.0, è possibile aggiungere listener di eventi al nodo target ma anche a tutti i nodi del flusso di eventi.

La facoltà di aggiungere listener di eventi in vari punti del flusso di lavoro si rivela utile per i componenti dell'interfaccia utente che comprendono più oggetti. Ad esempio, un oggetto button spesso contiene un oggetto text che funge da etichetta del pulsante. In mancanza della possibilità di aggiungere un listener al flusso di eventi, per ricevere la notifica degli eventi click sul pulsante bisognerebbe aggiungere un listener all'oggetto button e uno all'oggetto text. La presenza del flusso di eventi permette invece di inserire un unico listener di eventi sull'oggetto button che gestisce gli eventi click che si verificano sia sull'oggetto text sia nelle aree di validità dell'oggetto button che non sono oscurate dall'oggetto text.

Non tutti gli oggetti evento attraversano le tre fasi del flusso di eventi. Alcuni tipi di evento, come `enterFrame` e `init` vengono inviati direttamente al nodo target e non partecipano alla fase di cattura né alla fase di bubbling. Altri eventi possono avere come destinatari oggetti che non rientrano nell'elenco di visualizzazione, come gli eventi inviati a un'istanza della classe Socket. Anche questi tipi di evento vengono trasmessi direttamente all'oggetto target senza partecipare alle fasi di cattura e di bubbling.

Per conoscere il comportamento di un tipo di evento specifico, consultare la documentazione API o analizzare le proprietà dell'oggetto evento. L'analisi delle proprietà dell'oggetto evento viene descritta nella sezione seguente.

Oggetti evento

Nel nuovo sistema di gestione degli eventi, gli oggetti evento svolgono due funzioni principali. Innanzi tutto, gli oggetti evento rappresentano veri e propri eventi in quanto ne memorizzano le proprietà. Secondariamente, gli oggetti evento contengono una serie di metodi che permettono di manipolare gli oggetti stessi e di influenzare il comportamento del sistema di gestione degli eventi.

Allo scopo di facilitare l'accesso a tali proprietà e metodi, l'API di Flash Player definisce una classe Event che funge da classe base per tutti gli oggetti evento. La classe Event definisce una serie di proprietà e metodi fondamentali comuni a tutti gli oggetti evento.

Questa sezione inizia presentando le proprietà della classe Event, passa alla descrizione dei metodi della classe Event e conclude spiegando lo scopo delle sottoclassi della classe Event.

Nozioni fondamentali sulle proprietà della classe Event

La classe Event definisce varie proprietà e costanti non modificabili che forniscono informazioni importanti sull'oggetto evento. Ecco le più importanti:

- I tipi di oggetto evento sono rappresentati da costanti e memorizzati nella proprietà `Event.type`.
- La possibilità di disattivare un comportamento predefinito di un evento è rappresentata da un valore booleano e registrata nella proprietà `Event.cancelable`.
- Le informazioni sul flusso di eventi sono contenute nelle proprietà rimanenti.

Tipi di oggetti evento

Ogni oggetto evento è associato a un tipo di evento. I tipi di evento sono registrati nella proprietà `Event.type` sotto forma di valori stringa. È utile conoscere il tipo a cui appartiene un oggetto evento per permettere al codice di distinguere tra loro gli oggetti che appartengono a tipi diversi. L'esempio di codice seguente specifica che la funzione listener `clickHandler()` deve rispondere a qualsiasi oggetto evento click del mouse passato a `myDisplayObject`:

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

Alla classe Event sono associate circa una ventina di tipi di evento che sono rappresentati da costanti della classe Event. Alcune di queste costanti sono indicate nella porzione di codice che rappresenta la definizione della classe Event:

```
package flash.events
{
  public class Event
  {
    // costanti della classe
    public static const ACTIVATE:String = "activate";
    public static const ADDED:String    = "added";
    // altre costanti omesse per ragioni di spazio
  }
}
```

Queste costanti rappresentano un modo semplice per fare riferimento a tipi di evento specifici. Si consiglia di usare le costanti in luogo delle stringhe che rappresentano perché il compilatore è in grado di rilevare un eventuale errore di ortografia nel nome di una costante. Se invece si inserisce un errore di ortografia in una stringa, l'errore potrebbe passare inosservato in fase di compilazione ma generare comportamenti imprevisti che sarebbero difficili da individuare in seguito. Per aggiungere un listener di eventi, ad esempio, servirsi del codice seguente:

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

piuttosto che di:

```
myDisplayObject.addEventListener("click", clickHandler);
```

Informazioni sui comportamenti predefiniti

Il codice può verificare se è possibile disattivare il comportamento predefinito di un evento analizzando la proprietà `cancelable`. La proprietà `cancelable` contiene un valore booleano che indica se un comportamento predefinito può essere disattivato. È consentito disattivare, cioè annullare, il comportamento predefinito associato a un piccolo numero di eventi usando il metodo `preventDefault()`. Per ulteriori informazioni, vedere "Disattivazione del comportamento predefinito di un evento" a pagina 349.

Informazioni sul flusso di eventi

Le proprietà rimanenti della classe Event contengono informazioni importanti sull'oggetto evento e il suo rapporto con il flusso di eventi, come descritto nell'elenco seguente:

- La proprietà `bubbles` contiene informazioni sulle parti del flusso a cui l'oggetto evento partecipa.
- La proprietà `eventPhase` indica la fase corrente del flusso di eventi.
- La proprietà `target` registra un riferimento al destinatario dell'evento.
- La proprietà `currentTarget` registra un riferimento all'oggetto dell'elenco di visualizzazione che sta elaborando l'oggetto evento.

La proprietà bubbles

Se un oggetto evento partecipa alla fase di bubbling del flusso di eventi significa che l'oggetto evento viene fatto risalire dal nodo target su fino allo Stage. La proprietà `Event.bubbles` registra un valore booleano che indica se l'oggetto evento partecipa alla fase di bubbling. Dal momento che gli eventi che partecipano alla fase di bubbling partecipano anche alle fasi di cattura e target, se il valore della proprietà è `true`, l'oggetto evento partecipa a tutte le tre fasi. Se il valore è `false`, l'oggetto evento non partecipa alla fase di bubbling.

La proprietà eventPhase

Per stabilire la fase in cui si trova un oggetto evento è sufficiente analizzarne la proprietà `eventPhase`. La proprietà `eventPhase` contiene un valore intero senza segno che rappresenta una delle tre fasi del flusso di eventi. L'API Flash Player definisce una classe EventPhase separata che contiene tre costanti che corrispondono ai tre valori interi senza segno, come dimostra la seguente porzione di codice:

```
package flash.events
{
  public final class EventPhase
  {
    public static const CAPTURING_PHASE:uint = 1;
    public static const AT_TARGET:uint       = 2;
    public static const BUBBLING_PHASE:uint  = 3;
  }
}
```

Queste costanti corrispondono ai tre valori validi della proprietà `eventPhase` e concorrono a rendere il codice più leggibile. Ad esempio, per fare in modo che la funzione `myFunc()` sia chiamata solo se il destinatario dell'evento si trova nella fase target, si può usare il codice seguente per verificare questa condizione:

```
if (event.eventPhase == EventPhase.AT_TARGET)
{
  myFunc();
}
```

La proprietà target

La proprietà `target` contiene un riferimento all'oggetto che rappresenta il destinatario dell'evento. In alcuni casi il destinatario è semplice: quando un microfono diventa attivo, il destinatario dell'evento è l'oggetto Microphone. Se il destinatario si trova nell'elenco di visualizzazione, tuttavia, è necessario prendere in considerazione anche la gerarchia dell'elenco di visualizzazione. Ad esempio, se un utente esegue un clic del mouse in un punto che comprende più oggetti dell'elenco di visualizzazione sovrapposti, Flash Player sceglie come destinatario dell'evento l'oggetto più distante dallo Stage.

Nei file SWF particolarmente complessi, in particolare quelli in cui i pulsanti sono normalmente decorati con piccoli oggetti secondari, la proprietà `target` spesso può non essere utilizzabile poiché punterebbe a un oggetto secondario e non al pulsante. Un metodo comunemente usato in questi casi consiste nell'aggiungere listener di eventi al pulsante e usare la proprietà `currentTarget` perché ha la particolarità di puntare al pulsante, mentre la proprietà `target` potrebbe puntare a un oggetto secondario del pulsante.

La proprietà currentTarget

La proprietà `currentTarget` contiene un riferimento all'oggetto che sta elaborando l'oggetto evento. Anche se può apparire strano non sapere quale nodo sta attualmente elaborando l'oggetto evento che si sta analizzando, si tenga presente che è possibile aggiungere una funzione listener a qualsiasi oggetto di visualizzazione del flusso dell'oggetto e che la funzione listener può essere inserita in qualsiasi posizione. Inoltre, la stessa funzione listener può essere aggiunta a vari oggetti di visualizzazione. Man mano che la dimensione e la complessità di un progetto crescono, la proprietà `currentTarget` diventa sempre più utile.

Nozioni fondamentali sui metodi della classe Event

I metodi della classe Event possono essere suddivisi in tre categorie:

- Metodi di utilità che creano copie di un oggetto evento o lo convertono in una stringa
- Metodi del flusso di eventi, che rimuovono gli oggetti evento dal flusso di eventi
- Metodi dei comportamenti predefiniti, che disattivano i comportamenti predefiniti o controllano se sono stati disattivati

Metodi di utilità della classe Event

Esistono due metodi di utilità nella classe Event: il metodo `clone()` permette di creare copie di un oggetto evento, mentre il metodo `toString()` permette di convertire in stringa le proprietà e i relativi valori di un oggetto evento. Entrambi questi metodi vengono utilizzati internamente dal sistema di gestione degli eventi ma sono disponibili anche per gli sviluppatori.

Per gli sviluppatori esperti che creano sottoclassi della classe Event: affinché la sottoclasse della classe Event funzioni correttamente, è necessario sostituire e implementare le versioni di entrambi i metodi di utilità.

Arresto del flusso di eventi

Per interrompere il tragitto dell'oggetto evento lungo il flusso di eventi si può chiamare il metodo `Event.stopPropogation()` o il metodo `Event.stopImmediatePropogation()`. L'unica differenza tra i due metodi è la possibilità di eseguire gli altri listener di eventi del nodo corrente:

- Il metodo `Event.stopPropogation()` impedisce all'oggetto evento di passare al nodo seguente solo una volta eseguiti gli altri listener di eventi del nodo corrente.
- Anche il metodo `Event.stopImmediatePropogation()` impedisce all'oggetto evento di passare al nodo seguente, ma non permette l'esecuzione degli altri listener di eventi del nodo corrente.

La chiamata a uno di questi due metodi non influisce sull'esecuzione del comportamento predefinito associato a un evento. Per impedire l'esecuzione di un comportamento predefinito, usare i metodi dei comportamenti predefiniti della classe Event.

Disattivazione del comportamento predefinito di un evento

I due metodi che agiscono sulla disattivazione dei comportamenti predefiniti sono il metodo `preventDefault()` e il metodo `isDefaultPrevented()`. Chiamare il metodo `preventDefault()` per annullare un comportamento predefinito associato a un evento. Per verificare se `preventDefault()` è già stato chiamato su un oggetto evento, chiamare il metodo `isDefaultPrevented()` che restituisce il valore `true` se il metodo è già stato chiamato oppure il valore `false` in caso contrario.

Il metodo `preventDefault()` funziona solo se il comportamento predefinito dell'evento può essere disattivato. Per accertarsi di questa possibilità, consultare la documentazione API relativa a quel tipo di evento o usare ActionScript per analizzare la proprietà `cancelable` dell'oggetto evento.

La disattivazione del comportamento predefinito non influisce sull'avanzamento dell'oggetto evento all'interno del flusso di eventi. Usare i metodi del flusso di eventi della classe Event per eliminare un oggetto evento dal flusso di eventi.

Sottoclassi della classe Event

Per la maggior parte di eventi, il gruppo di proprietà comuni definite nella classe Event è sufficiente. Esistono, tuttavia, alcuni eventi le cui caratteristiche peculiari non possono essere catturate dalle proprietà della classe Event. Per questi eventi speciali, l'API Flash Player definisce varie sottoclassi della classe Event.

Ogni sottoclasse comprende proprietà supplementari e tipi di evento riservati a quella categoria di eventi. Ad esempio, gli eventi associati agli input del mouse hanno delle caratteristiche che non possono essere catturate dalle proprietà definite nella classe Event. La classe MouseEvent rappresenta un ampliamento della classe Event e comprende dieci proprietà che contengono informazioni come la posizione dell'evento mouse e i tasti premuti durante l'evento mouse.

Una sottoclasse della classe Event contiene anche le costanti che rappresentano i tipi di evento associati alla sottoclasse. Ad esempio, la classe MouseEvent definisce costanti per vari tipi di evento mouse e comprende gli eventi del tipo `click`, `doubleClick`, `mouseDown` e `mouseUp`.

Come descritto nella sezione "Metodi di utilità della classe Event" a pagina 348, per creare una sottoclasse della classe Event è necessario sostituire i metodi `clone()` e `toString()` per attivare funzionalità specifiche per la sottoclasse.

Listener di eventi

I listener di eventi, detti anche gestori di eventi, sono funzioni che Flash Player esegue in risposta a eventi specifici. L'aggiunta di un listener di eventi è una procedura che si compone di due fasi. Si comincia creando una funzione o un metodo di classe che Flash Player deve eseguire in risposta all'evento. Questa funzione viene talvolta definita come funzione listener o funzione gestore di eventi. In secondo luogo, si usa il metodo `addEventListener()` per associare alla funzione listener il destinatario dell'evento o un oggetto dell'elenco di visualizzazione che appartiene al flusso di eventi appropriato.

Creazione di una funzione listener

La creazione di funzioni listener è un ambito in cui il modello eventi di ActionScript 3.0 si discosta dal modello eventi DOM. Nel modello eventi DOM, esiste una distinzione netta tra un listener di eventi e una funzione listener: listener di eventi è un'istanza di una classe che implementa l'interfaccia EventListener, mentre la funzione listener è un metodo di quella classe definito `handleEvent()`. Nel modello eventi DOM si registra l'istanza della classe che contiene la funzione listener e non la funzione listener in sé.

Nel modello eventi di ActionScript 3.0 non esiste una distinzione tra un listener di eventi e una funzione listener. ActionScript 3.0 non dispone dell'interfaccia EventListener e le funzioni listener possono essere definite al di fuori di una classe o come parte di una classe. Inoltre, le funzioni listener non devono necessariamente chiamarsi `handleEvent()` in quanto possono essere definite tramite un qualsiasi identificatore valido. In ActionScript 3.0, è lo sviluppatore che registra il nome della funzione listener.

Funzione listener definita al di fuori di una classe

L'esempio di codice seguente crea un file SWF semplice in cui viene visualizzato un quadrato rosso. La funzione listener `clickHandler()`, che non fa parte di una classe, monitora gli eventi click del mouse sul quadrato rosso.

```
package
{
  import flash.display.Sprite;

  public class ClickExample extends Sprite
  {
    public function ClickExample()
    {
      var child:ChildSprite = new ChildSprite();
      addChild(child);
    }
  }
}
```

```
import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
  public function ChildSprite()
  {
    graphics.beginFill(0xFF0000);
    graphics.drawRect(0,0,100,100);
    graphics.endFill();
    addEventListener(MouseEvent.CLICK, clickHandler);
  }
}

function clickHandler(event:MouseEvent):void
{
  trace("clickHandler detected an event of type: " + event.type);
  trace("the this keyword refers to: " + this);
}
```

Quando un utente interagisce con il file SWF facendo clic sul quadrato, Flash Player genera il seguente output di traccia:

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Si noti che l'oggetto evento viene passato come argomento a `clickHandler()`. Questo permette alla funzione listener di analizzare l'oggetto evento. In questo esempio, si usa la proprietà `type` dell'oggetto evento per determinare se l'evento è un clic del mouse.

L'esempio verifica anche il valore della parola chiave `this`. In questo caso, `this` rappresenta l'oggetto globale, il che è opportuno perché la funzione è definita al di fuori di qualsiasi classe predefinita o oggetto.

Funzione listener definita come metodo di classe

L'esempio seguente è identico a quello precedente che definisce la classe ClickExample, con la sola eccezione che la funzione `clickHandler()` è definita come metodo della classe ChildSprite:

```
package
{
  import flash.display.Sprite;

  public class ClickExample extends Sprite
  {
    public function ClickExample()
    {
      var child:ChildSprite = new ChildSprite();
      addChild(child);
    }
  }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
  public function ChildSprite()
  {
    graphics.beginFill(0xFF0000);
    graphics.drawRect(0,0,100,100);
    graphics.endFill();
    addEventListener(MouseEvent.CLICK, clickHandler);
  }
  private function clickHandler(event:MouseEvent):void
  {
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
  }
}
```

Quando un utente interagisce con il file SWF facendo clic sul quadrato rosso, Flash Player genera il seguente output di traccia:

```
clickHandler detected an event of type: click
the this keyword refers to: [object ChildSprite]
```

Si noti che la parola chiave `this` si riferisce all'istanza di ChildSprite denominata `child`. Questo rappresenta un cambiamento rispetto a quanto avveniva in ActionScript 2.0. Chi ha dimestichezza con i componenti in ActionScript 2.0 forse ricorderà che quando un metodo di classe veniva passato a `UIEventDispatcher.addEventListener()`, l'area di validità del metodo era legata al componente che aveva trasmesso l'evento anziché alla classe in cui il metodo listener era stato definito. In altre parole, usando questa tecnica in ActionScript 2.0, la parola chiave `this` farebbe riferimento al componente che trasmette l'evento anziché all'istanza di ChildSprite.

Questo rappresentava un problema notevole per i programmatori perché significava che non potevano accedere ad altri metodi e proprietà della classe che conteneva il metodo listener. Per aggirare il problema, i programmatori di ActionScript 2.0 potevano usare la classe `mx.util.Delegate` per cambiare l'area di validità del metodo listener. Questo espediente non è tuttavia più necessario perché ActionScript 3.0 crea un metodo vincolato quando viene chiamato `addEventListener()`. Di conseguenza, la parola chiave `this` fa riferimento all'istanza di ChildSprite denominata `child` e il programmatore ha accesso agli altri metodi e alle proprietà della classe ChildSprite.

Listener di eventi da evitare

Esiste una terza possibilità, sconsigliata, che consiste nel creare un oggetto generico dotato di una proprietà che punta a una funzione listener assegnata in modo dinamico. Questa tecnica viene comunque presentata qui perché veniva usata comunemente in ActionScript 2.0, ma se ne sconsiglia l'impiego in ActionScript 3.0, in quanto la parola chiave `this` farebbe riferimento all'oggetto globale e non all'oggetto listener.

L'esempio che segue è identico all'esempio precedente della classe ClickExample, con l'eccezione che qui la funzione listener è definita come parte di un oggetto generico denominato `myListenerObj`:

```
package
{
  import flash.display.Sprite;

  public class ClickExample extends Sprite
  {
    public function ClickExample()
    {
      var child:ChildSprite = new ChildSprite();
      addChild(child);
    }
  }
}
```

```
import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
  public function ChildSprite()
  {
    graphics.beginFill(0xFF0000);
    graphics.drawRect(0,0,100,100);
    graphics.endFill();
    addEventListener(MouseEvent.CLICK, myListenerObj.clickHandler);
  }
}

var myListenerObj:Object = new Object();
myListenerObj.clickHandler = function (event:MouseEvent):void
{
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
}
```

Il risultato dell'output di traccia è il seguente:

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Ci si aspetterebbe che `this` facesse riferimento a `myListenerObj` e che l'output di traccia fosse `[object Object]`, mentre invece fa riferimento all'oggetto globale. Quando si passa il nome di una proprietà dinamica come argomento a `addEventListener()`, Flash Player non è in grado di creare un metodo vincolato perché il parametro che viene passato come `listener` non è altro che l'indirizzo di memoria della funzione listener e Flash Player non ha modo di collegare quell'indirizzo all'istanza di `myListenerObj`.

Gestione dei listener di eventi

È possibile gestire le funzioni listener usando i metodi dell'interfaccia IEventDispatcher. IEventDispatcher è la versione di ActionScript 3.0 dell'interfaccia EventTarget del modello eventi DOM. Nonostante il nome IEventDispatcher sembri sottintendere che il suo scopo principale sia inviare (in inglese "dispatch") oggetti evento, in realtà, i metodi di questa classe sono usati in prevalenza per registrare i listener di eventi, verificare la presenza di listener di eventi e di rimuovere i listener di eventi. L'interfaccia IEventDispatcher definisce cinque metodi, come illustrato nel codice seguente:

```
package flash.events
{
  public interface IEventDispatcher
  {
    function addEventListener(eventName:String,
             listener:Object,
             useCapture:Boolean=false,
             priority:Integer=0,
             useWeakReference:Boolean=false):Boolean;

    function removeEventListener(eventName:String,
           listener:Object,
           useCapture:Boolean=false):Boolean;

    function dispatchEvent(eventObject:Event):Boolean;

    function hasEventListener(eventName:String):Boolean;
    function willTrigger(eventName:String):Boolean;
  }
}
```

L'API Flash Player implementa l'interfaccia IEventDispatcher con la classe EventDispatcher, che funge da classe base per tutte le classi suscettibili di essere destinatarie di eventi o parte del flusso di eventi. Ad esempio, la classe DisplayObject eredita dalla classe EventDispatcher. Ciò significa che qualsiasi oggetto dell'elenco di visualizzazione ha accesso ai metodi dell'interfaccia IEventDispatcher.

Aggiunta dei listener di eventi

Il metodo `addEventListener()` è il pezzo forte dell'interfaccia IEventDispatcher e consente di registrare le funzioni listener. I due parametri obbligatori sono `type` e `listener`. Con il parametro `type` si specifica il tipo di evento, mentre con il parametro `listener` si specifica la funzione listener da eseguire quando si verifica l'evento. Il parametro `listener` può essere un riferimento a una funzione o a un metodo di classe.

NOTA

Non utilizzare le parentesi per specificare il parametro `listener`. Ad esempio, la funzione `clickHandler()` è specificata senza parentesi nella seguente chiamata al metodo `addEventListener()`:
`addEventListener(MouseEvent.CLICK, clickHandler)`.

Il parametro `useCapture` del metodo `addEventListener()` consente di controllare la fase del flusso di eventi in cui il listener sarà attivo. Se `useCapture` è impostato su `true`, il listener sarà attivo durante la fase di cattura del flusso di eventi. Se `useCapture` è impostato su `false`, il listener sarà attivo durante la fase target e di bubbling del flusso di eventi. Per monitorare un evento durante tutte le fasi del flusso di eventi, chiamare `addEventListener()` due volte, una volta con `useCapture` impostato su `true` e una seconda volta con `useCapture` impostato su `false`.

Il parametro `priority` del metodo `addEventListener()` non è un elemento ufficiale del modello eventi DOM Level 3, però è stato incluso in ActionScript 3.0 per rendere più flessibile l'organizzazione dei listener di eventi. Quando si chiama `addEventListener()`, si può impostare la priorità per quel listener di eventi passando un valore intero come parametro `priority`. Il valore predefinito è 0, ma può essere modificato in un valore intero negativo o positivo. Più è alto il numero, prima verrà eseguito il listener di eventi. I listener di eventi con la stessa priorità vengono eseguiti nell'ordine in cui sono stati aggiunti, quindi prima è stato aggiunto un listener, prima viene eseguito.

Il parametro `useWeakReference` permette di specificare se il riferimento alla funzione listener è debole o normale. Impostando questo parametro su `true` si evitano situazioni in cui le funzioni listener rimangono in memoria anche quando diventano superflue. Flash Player usa la tecnica *garbage collection* per eliminare dalla memoria gli oggetti inutilizzati. Un oggetto viene considerato inutilizzato quando non esistono più riferimenti ad esso. L'operazione di garbage collection non tiene in considerazione i riferimenti deboli, vale a dire che una funzione listener per cui esiste solo un riferimento debole diventa una candidata per la garbage collection.

Una conseguenza importante di questo parametro riguarda l'uso degli eventi degli oggetti di visualizzazione. Normalmente, ci si aspetterebbe che un oggetto di visualizzazione venisse eliminato dalla memoria nel momento in cui viene eliminato dall'elenco di visualizzazione. Tuttavia, se altri oggetti hanno effettuato il subscribing come listener di quell'oggetto di visualizzazione e se il parametro `useWeakReference` è impostato su `false` (cioè sul valore predefinito), l'oggetto di visualizzazione continuerà ad esistere nella memoria di Flash Player pur non apparendo più sullo schermo. Per risolvere questo inconveniente si può effettuare il subscribing di tutti i listener all'oggetto di visualizzazione con il parametro `useWeakReference` impostato su `true` oppure rimuovere tutti i listener di eventi dall'oggetto di visualizzazione usando il metodo `removeEventListener()`.

Eliminazione dei listener di eventi

Per rimuovere i listener di eventi inutilizzati si può usare il metodo `removeEventListener()`. È buona norma rimuovere i listener diventati superflui. I parametri obbligatori sono `eventName` e `listener`, cioè gli stessi parametri del metodo `addEventListener()`. Ricordarsi che è possibile monitorare gli eventi durante tutte le fasi del flusso di eventi chiamando due volte `addEventListener()`: una con `useCapture` impostato su `true` e una seconda con il parametro impostato su `false`. Per eliminare entrambi i listener di eventi è necessario chiamare `removeEventListener()` due volte: una prima volta con `useCapture` impostato su `true` e una seconda volta con il parametro impostato su `false`.

Invio degli eventi

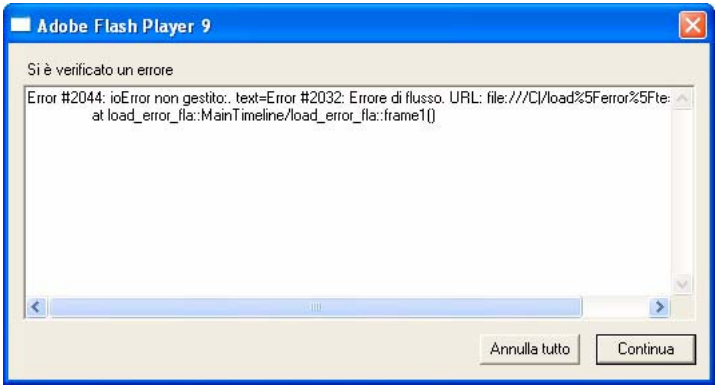
I programmatori esperti possono usare il metodo `dispatchEvent()` per inviare un oggetto evento personalizzato nel flusso di eventi. L'unico parametro accettato da questo metodo è un riferimento a un oggetto evento, che deve essere un'istanza o una sottoclasse della classe Event. Una volta inviato l'evento, la proprietà `target` dell'oggetto evento viene impostata sull'oggetto su cui il metodo `dispatchEvent()` è stato chiamato.

Verifica della presenza di listener di eventi

Gli ultimi due metodi dell'interfaccia IEventDispatcher forniscono informazioni utili per individuare l'esistenza di listener di eventi. Il metodo `hasEventListener()` restituisce il valore `true` se individua un listener di eventi per un tipo di evento specifico su un particolare oggetto dell'elenco di visualizzazione. Anche il metodo `willTrigger()` restituisce il valore `true` se individua un listener per un oggetto specifico dell'elenco di visualizzazione, ma `willTrigger()` verifica la presenza di listener non solo sull'oggetto di visualizzazione in sé ma anche su tutti gli antenati dell'oggetto per tutte le fasi del flusso di eventi.

Eventi errore senza listener

Sono le eccezioni, e non gli eventi, il meccanismo principale per la gestione degli errori in ActionScript 3.0, ma la gestione delle eccezioni non prende in considerazione le operazioni asincrone come il caricamento dei file. Se si verifica un errore durante un'operazione asincrona di questo tipo, Flash Player invia un oggetto evento errore. Se non è stato creato un listener per gli eventi errore, la versione debugger di Flash Player richiama una finestra di dialogo con alcune informazioni sull'errore. Ad esempio, l'uso di un URL non valido durante il caricamento di un file genera la seguente finestra di dialogo nella versione debugger di Flash Player:



La maggior parte degli eventi errore si basano sulla classe ErrorEvent e in quanto tali sono configurati con la proprietà `text` che viene utilizzata per memorizzare il messaggio di errore visualizzato da Flash Player. Le uniche due eccezioni sono costituite dalle classi StatusEvent e NetStatusEvent, che sono caratterizzate dalla proprietà `level` (`StatusEvent.level` e `NetStatusEvent.info.level`). Quando il valore della proprietà `level` è `"error"`, questi tipi di evento vengono considerati eventi errore.

Un evento errore non determina l'arresto della riproduzione di un file SWF e genera solo la visualizzazione di una finestra di dialogo nelle versioni debugger dei plug-in per browser e dei lettori autonomi, di un messaggio nel pannello Output nel player di creazione e l'aggiunta di una riga nel file di registro di Adobe Flex Builder 2. Non si manifesta affatto nelle versioni standard di Flash Player.

Esempio: Alarm Clock

L'esempio Alarm Clock rappresenta un orologio che permette all'utente di specificare l'ora in cui si deve attivare la sveglia e il messaggio da visualizzare in tale momento. Questo esempio si basa sull'applicazione SimpleClock di Capitolo 5, "Operazioni con data e ora" Alarm Clock illustra vari aspetti dell'uso di eventi in ActionScript 3.0, tra cui:

- Intercettazione e risposta a un evento
- Notifica dei listener di un evento
- Creazione di un tipo di evento personalizzato

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione Alarm Clock si trovano nella cartella Samples/AlarmClock. L'applicazione comprende i file seguenti:

File	Descrizione
AlarmClockApp.mxml o AlarmClockApp.fla	Il file principale dell'applicazione in Flash (FLA) o Flex (MXML)
com/example/programmingas3/clock/AlarmClock.as	Classe che estende la classe SimpleClock aggiungendo la funzionalità sveglia.
com/example/programmingas3/clock/AlarmEvent.as	Classe evento personalizzata (sottoclasse di flash.events.Event) che funge da oggetto evento per l'evento `alarm` della classe AlarmClock.
com/example/programmingas3/clock/AnalogClockFace.as	Disegna un orologio rotondo con lancette per le ore, i minuti e i secondi (descritto nell'esempio SimpleClock).
com/example/programmingas3/clock/SimpleClock.as	Componente di interfaccia clock con funzionalità di misurazione del tempo semplice (descritta nell'esempio SimpleClock).

Panoramica di Alarm Clock

Per assolvere alla funzionalità principale dell'orologio di questo esempio, cioè misurare il tempo e visualizzare il quadrante, viene utilizzato il codice dell'applicazione SimpleClock, descritto in "Esempio: Orologio analogico semplice" a pagina 213. La classe AlarmClock estende la classe SimpleClock aggiungendo la funzionalità sveglia, che permette di impostare l'ora in cui si deve attivare la sveglia e inviare una notifica quando la sveglia "suona".

Fornire una notifica per indicare che qualcosa avviene è il compito per cui sono stati inventati gli eventi. La classe AlarmClock espone l'evento Alarm, che altri oggetti possono intercettare al fine di eseguire le azioni desiderate. Inoltre, la classe AlarmClock usa un'istanza della classe Timer per determinare quando la sveglia deve essere attivata. Analogamente alla classe AlarmClock, la classe Timer genera un evento per avvertire altri oggetti (un'istanza di AlarmClock, in questo caso) quando è trascorsa una determinata quantità di tempo. Come per la maggior parte delle applicazioni ActionScript, gli eventi rappresentano un componente importante della funzionalità dell'applicazione di esempio Alarm Clock.

Attivazione della sveglia

Come accennato in precedenza, l'unica funzionalità a cui presiede la classe AlarmClock è l'impostazione e l'attivazione della sveglia. La classe incorporata Timer (flash.utils.Timer) rappresenta per lo sviluppatore un modo per definire il codice che deve essere eseguito una volta trascorsa la quantità di tempo specificata. La classe AlarmClock usa un'istanza di Timer per determinare quando la sveglia deve essere attivata.

```
import flash.events.TimerEvent;
import flash.utils.Timer;

/**
 * Timer che verrà usato per la sveglia.
 */
public var alarmTimer:Timer;
...
/**
 * Crea una nuova istanza di AlarmClock.
 */
public override function initClock(faceSize:Number = 200):void
{
  super.initClock(faceSize);
  alarmTimer = new Timer(0, 1);
  alarmTimer.addEventListener(TimerEvent.TIMER, onAlarm);
}
```

L'istanza di Timer definita nella classe AlarmClock è denominata `alarmTimer`. Il metodo `initClock()`, che esegue operazioni di configurazione necessarie per l'istanza di AlarmClock, esegue due operazioni con la variabile `alarmTimer`. Innanzi tutto, viene creata un'istanza della variabile con parametri che indicano all'istanza di Timer di attendere 0 millisecondi e di attivare l'evento timer una sola volta. Dopo la creazione dell'istanza di `alarmTimer`, il codice chiama il metodo `addEventListener()` della variabile per indicare che deve intercettare l'evento `timer` della variabile. Le istanze Timer operano inviando il proprio evento `timer` una volta trascorsa la quantità di tempo predefinita. Per poter attivare la sveglia, la classe AlarmClock ha bisogno di sapere quando l'evento `timer` viene inviato. Chiamando `addEventListener()`, il codice AlarmClock si registra come listener presso `alarmTimer`. I due parametri indicano che la classe AlarmClock intercetta l'evento `timer` (indicato dalla costante `TimerEvent.TIMER`) e quando l'evento si verifica, il metodo `onAlarm()` della classe AlarmClock deve essere chiamato in risposta all'evento.

Per eseguire l'impostazione della sveglia, viene chiamato il metodo `setAlarm()` della classe AlarmClock nel modo seguente:

```
/**
 * Imposta l'ora in cui si deve attivare la sveglia.
 * @param hour La porzione ora dell'ora della sveglia.
 * @param minutes La porzione minuti dell'ora della sveglia.
 * @param message Il messaggio da visualizzare quando viene attivata la
sveglia.
 * @return L'ora in cui si deve attivare la sveglia.
 */
public function setAlarm(hour:Number = 0, minutes:Number = 0,
message:String = "Alarm!"):Date
{
  this.alarmMessage = message;
  var now:Date = new Date();
  // Imposta l'ora per la data odierna.
  alarmTime = new Date(now.fullYear, now.month, now.date, hour, minutes);

  // Determina se oggi l'orario specificato è già passato.
  if (alarmTime <= now)
  {
    alarmTime.setTime(alarmTime.time + MILLISECONDS_PER_DAY);
  }

  // Arresta il timer della sveglia se già impostato.
  alarmTimer.reset();
  // Calcola quanti millisecondi devono trascorrere prima che la sveglia
  // si attivi (differenza tra ora della sveglia e ora attuale) e imposta
  // questo valore come ritardo del timer della sveglia.
  alarmTimer.delay = Math.max(1000, alarmTime.time - now.time);
  alarmTimer.start();

  return alarmTime;
}
```

Questo metodo esegue varie attività, compresa la registrazione del messaggio di sveglia e la creazione di un oggetto Date (`alarmTime`) che rappresenta il momento preciso in cui la sveglia si attiva. Di particolare rilevanza per questa discussione sono le ultime righe del metodo, la modalità in cui il timer della variabile `alarmTimer` viene impostato e attivato. Per cominciare, viene chiamato il metodo `reset()` che arresta il timer e lo azzera nel caso fosse già in esecuzione. In seguito, l'ora attuale (rappresentata dalla variabile `now`) viene sottratta dal valore della variabile `alarmTime` per determinare quanti millisecondi devono trascorrere prima che la sveglia si attivi. La classe Timer non attiva il suo evento `timer` a un'ora assoluta, quindi è la differenza oraria relativa che viene assegnata alla proprietà `delay` di `alarmTimer`. Infine, viene chiamato il metodo `start()` per avviare il timer.

Una volta trascorsa la quantità di tempo specificata, `alarmTimer` invia l'evento `timer`. Poiché la classe AlarmClock ha registrato il suo metodo `onAlarm()` come listener di quell'evento, quando l'evento `timer` si verifica, `onAlarm()` viene chiamato.

```
/**
 * Chiamato quando l'evento timer viene inviato.
 */
public function onAlarm(event:TimerEvent):void
{
  trace("Alarm!");
  var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
  this.dispatchEvent(alarm);
}
```

Un metodo registrato come listener di eventi deve essere definito con l'indicatore appropriato (cioè, l'insieme di parametri e il tipo restituito del metodo). Per poter essere impostato come listener dell'evento `timer` della classe Timer, un metodo deve definire un parametro il cui tipo di dati è TimerEvent (flash.events.TimerEvent), una sottoclasse della classe Event. Quando l'istanza di Timer chiama i suoi listener di eventi, passa un'istanza di TimerEvent come oggetto evento.

Invio di notifiche per l'evento alarm

Come la classe Timer, la classe AlarmClock fornisce un evento che permette ad altre porzioni di codice di ricevere notifiche dell'attivazione della sveglia. Per poter usare la struttura di gestione degli eventi incorporata nel linguaggio ActionScript, una classe deve implementare l'interfaccia flash.events.IEventDispatcher. Ciò avviene, comunemente, estendendo la classe flash.events.EventDispatcher, che fornisce un'implementazione standard di IEventDispatcher (o tramite l'estensione di una delle sottoclassi di EventDispatcher). Come descritto in precedenza, la classe AlarmClock estende la classe SimpleClock che, a sua volta, estende la classe Sprite, che estende (attraverso una catena di ereditarietà) la classe EventDispatcher. Questo significa che la classe AlarmClock è già intrinsecamente dotata della funzionalità necessaria alla generazione dei propri eventi.

Altre porzioni di codice possono essere registrate come destinatarie delle notifiche dell'evento `alarm` della classe AlarmClock chiamando il metodo `addEventListener()` che AlarmClock eredita da EventDispatcher. Quando è pronta per comunicare ad altre porzioni di codice che il suo evento `alarm` si è verificato, un'istanza di AlarmClock chiama il metodo `dispatchEvent()`, anch'esso ereditato da EventDispatcher.

```
var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
this.dispatchEvent(alarm);
```

Le seguenti righe di codice sono state estratte dal metodo `onAlarm()` della classe AlarmClock (riprodotto integralmente in precedenza). Viene chiamato il metodo `dispatchEvent()` dell'istanza AlarmClock, che notifica tutti i listener registrati che l'evento `alarm` dell'istanza di AlarmClock è stato attivato. Il parametro passato a `dispatchEvent()` è l'oggetto evento che viene trasmesso ai metodi del listener. In questo caso, si tratta di un'istanza della classe AlarmEvent, una sottoclasse di Event creata specificamente per questo esempio.

Creazione di un evento alarm personalizzato

Tutti i listener di eventi ricevono un parametro dell'oggetto evento che indica il particolare tipo di evento attivato. In molti casi, l'oggetto evento è un'istanza della classe Event. Tuttavia, talvolta, è opportuno fornire informazioni supplementari ai listener. Come descritto in precedenza in questo stesso capitolo, il modo più comune per svolgere questa operazione consiste nel definire una nuova classe (una sottoclasse della classe Event) e di usare una sua istanza come oggetto evento. In questo esempio, un'istanza di AlarmEvent viene usata come oggetto evento quando l'evento `alarm` della classe AlarmClock viene inviato. La classe AlarmEvent, riportata di seguito, fornisce informazioni supplementari sull'evento `alarm`, in particolare il messaggio di avvertenza:

```
import flash.events.Event;

/**
 * La classe Event personalizzata aggiunge una proprietà message alla
classe di base Event.
 */
public class AlarmEvent extends Event
{
  /**
   * Il nome del nuovo tipo di AlarmEvent.
   */
  public static const ALARM:String = "alarm";

  /**
   * Messaggio di testo che può essere passato a un gestore di eventi
   * insieme a questo oggetto evento.
   */
  public var message:String;

  /**
   *  Funzione di costruzione.
   * @param message Il testo da visualizzare all'attivazione della
sveglia.
   */
  public function AlarmEvent(message:String = "ALARM!")
  {
    super(ALARM);
    this.message = message;
  }
  ...
}
```

Il modo migliore per creare una classe oggetto evento personalizzata consiste nel definire una classe che estende la classe Event, come illustrato nell'esempio precedente. Per completare la funzionalità ereditata la classe AlarmEvent definisce la proprietà `message` che contiene il testo del messaggio associato all'evento; il valore `message` viene passato come parametro nella funzione di costruzione AlarmEvent. La classe AlarmEvent definisce anche la costante `ALARM`, che può essere usata come riferimento all'evento specifico (`alarm`) quando si chiama il metodo `addEventListener()` della classe AlarmClock.

Oltre ad aggiungere delle funzionalità personalizzate, ogni sottoclasse di Event deve sostituire il metodo `clone()` ereditato come parte della struttura di gestione degli eventi di ActionScript. In via facoltativa, le sottoclassi di Event possono anche sostituire il metodo ereditato `toString()` al fine di includere le proprietà dell'evento personalizzato nel valore restituito quanto il metodo `toString()` viene chiamato.

```
/**
 * Crea e restituisce una copia dell'istanza corrente.
 * @return Una copia dell'istanza corrente.
 */
public override function clone():Event
{
  return new AlarmEvent(message);
}

/**
 * Restituisce una stringa contenente tutte le proprietà
 * dell'istanza corrente.
 * @return Una rappresentazione sotto forma di stringa dell'istanza
corrente.
 */
public override function toString():String
{
  return formatToString("AlarmEvent", "type", "bubbles", "cancelable",
"eventPhase", "message");
}
```

Il metodo sostituito `clone()` deve restituire una nuova istanza della sottoclasse Event personalizzata con tutte le proprietà personalizzate impostate sull'istanza corrente. Nel metodo sostituito `toString()`, il metodo utilità `formatToString()` (ereditato da Event) viene usato per fornire una stringa con il nome del tipo personalizzato e i nomi e i valori di tutte le sue proprietà.

CAPITOLO 11

Operazioni con XML

11

ActionScript 3.0 comprende un gruppo di classi basate sulla specifica ECMAScript for XML (E4X) (ECMA-357 edizione 2) dotate di funzionalità potenti e di facile utilizzo per la gestione dei dati XML. Grazie a E4X, è possibile sviluppare codice con dati XML a una velocità impensabile con le precedenti tecniche di programmazione. Un ulteriore vantaggio è rappresentato dal fatto che il codice generato è anche più facile da leggere.

Il presente capitolo descrive l'uso di E4X per l'elaborazione dei dati XML.

Sommario

Nozioni di base su XML

Introduzione alle operazioni con XML

XML è una modalità standard di rappresentazione delle informazioni strutturate che permette ai computer di manipolarle con facilità e alle persone di scriverle e capirle senza eccessiva difficoltà. XML sta per eXtensible Markup Language. Una descrizione dello standard XML è disponibile all'indirizzo www.w3.org/XML/.

XML è un formato comodo e molto diffuso per la categorizzazione dei dati che li rende facili da leggere, individuare e manipolare. XML impiega una struttura ad albero e tag simile a quella di HTML. Ecco un piccolo esempio di dati in formato XML:

```
<song>
  <title>What you know?</title>
  <artist>Steve and the flubberblubs</artist>
  <year>1989</year>
  <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

I dati XML possono essere anche più complessi e contenere tag nidificati in altri tag, attributi e altri componenti strutturali. Ecco un esempio di dati più complessi in formato XML:

```
<album>
  <title>Questions, unanswered</title>
  <artist>Steve and the flubberblubs</artist>
  <year>1989</year>
  <tracks>
    <song tracknumber="1" length="4:05">
      <title>What do you know?</title>
      <artist>Steve and the flubberblubs</artist>
      <lastplayed>2006-10-17-08:31</lastplayed>
    </song>
    <song tracknumber="2" length="3:45">
      <title>Who do you know?</title>
      <artist>Steve and the flubberblubs</artist>
      <lastplayed>2006-10-17-08:35</lastplayed>
    </song>
    <song tracknumber="3" length="5:14">
      <title>When do you know?</title>
      <artist>Steve and the flubberblubs</artist>
      <lastplayed>2006-10-17-08:39</lastplayed>
    </song>
    <song tracknumber="4" length="4:19">
      <title>Do you know?</title>
      <artist>Steve and the flubberblubs</artist>
      <lastplayed>2006-10-17-08:44</lastplayed>
    </song>
  </tracks>
</album>
```

Si noti che questo documento XML comprende al suo interno altre strutture XML complete (come i tag `song` e i relativi oggetti secondari), e dimostra l'uso di varie strutture XML come gli attributi (`tracknumber` e `length` nei tag `song`) e di tag che, in luogo di dati, contengono altri tag (come il tag `tracks`).

Guida introduttiva a XML

Gli utenti che non hanno dimestichezza o una conoscenza limitata del formato XML possono leggere la breve descrizione dei suoi aspetti più salienti presentata di seguito. I dati XML vengono scritti in formato di testo normale rispettando una particolare sintassi che permette di organizzare le informazioni in una struttura significativa. Generalmente, un blocco di dati XML viene definito *documento XML*. I dati in formato XML sono organizzati in *elementi* (singole voci di dati o contenitori) disposti in una struttura gerarchica. In ogni documento XML è presente un elemento di livello superiore, cioè la voce principale, all'interno del quale può esserci una porzione di dati singola o, più verosimilmente, vari elementi che a loro volta ospitano altri elementi e così via. L'esempio di documento XML seguente contiene informazioni su un album musicale:

```
<song tracknumber="1" length="4:05">
  <title>What do you know?</title>
  <artist>Steve and the flubberblubs</artist>
  <mood>Happy</mood>
  <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

Ogni elemento è caratterizzato da una serie di *tag*: il nome dell'elemento racchiuso tra parentesi angolari (i simboli di minore di/maggiore di). Il tag di apertura, che indica l'inizio dell'elemento, contiene il nome dell'elemento:

```
<title>
```

Il tag di chiusura indica la fine dell'elemento e presenta una barra prima del nome dell'elemento:

```
</title>
```

Gli elementi che non presentano del contenuto possono essere scritti come elementi vuoti. Nel formato XML, l'elemento:

```
<lastplayed/>
```

è identico a questo elemento:

```
<lastplayed></lastplayed>
```

Oltre al proprio contenuto racchiuso tra i due tag di apertura e chiusura, un elemento può comprendere anche altri valori, denominati *attributi*, definiti nel tag di apertura. Ad esempio, il seguente elemento XML definisce un unico attributo `length` caratterizzato dal valore `"4:19"`:

```
<song length="4:19"></song>
```

Tutti gli elementi XML prevedono del contenuto, che può essere un valore singolo, uno o più elementi XML oppure niente (nel caso di elementi vuoti).

Per saperne di più su XML

Per chi desidera saperne di più sul formato XML sono disponibili vari libri e risorse supplementari e i seguenti siti Web:

- W3Schools XML Tutorial: http://w3schools.com/xml/
- XML.com: http://www.xml.com/
- Tutorial, discussioni e molto altro su XMLpitstop: http://xmlpitstop.com/

Classi ActionScript per il formato XML

ActionScript 3.0 comprende varie classi da utilizzare con i dati in formato XML. Le due classi principali sono le seguenti:

- XML: rappresenta un unico elemento XML che può essere un documento XML con vari elementi secondari o un elemento a un valore all'interno di un documento.
- XMLList: rappresenta un gruppo di elementi XML. Si usa l'oggetto XMLList quando sono presenti vari elementi XML di pari livello (cioè che appartengono allo stesso elemento principale all'interno della gerarchia del documento XML). Ad esempio, un'istanza di XMLList sarebbe il modo più semplice per gestire questo gruppo di elementi XML (presumibilmente appartenenti a un documento XML):

  ```
  <artist type="composer">Fred Wilson</artist>
  <artist type="conductor">James Schmidt</artist>
  <artist type="soloist">Susan Harriet Thurndon</artist>
  ```

Per operazioni di livello avanzato che comprendono spazi dei nomi XML, ActionScript mette a disposizione le classi Namespace e QName. Per ulteriori informazioni, vedere "Uso dello spazio dei nomi XML" a pagina 387.

Oltre alle classi incorporate specifiche per XML, ActionScript 3.0 comprende vari operatori che attivano funzionalità specifiche per l'accesso ai dati XML e la loro manipolazione. L'approccio che prevede la gestione dei dati XML tramite queste classi e questi operatori è conosciuto come ECMAScript for XML (E4X) ed è definito nella specifica ECMA-357 edizione 2.

Operazioni comuni con XML

Le operazioni più comuni che si eseguono in ActionScript manipolando dati in formato XML sono le seguenti:

- Costruzione di documenti XML (aggiungendo elementi e valori)
- Accesso a elementi, valori e attributi XML
- Ricerca all'interno di elementi XML
- Spostamenti all'interno di un gruppo di elementi XML.
- Conversione di dati da classi XML a String e viceversa
- Operazioni con gli spazi dei nomi XML
- Caricamento di file XML esterni

Concetti e termini importanti

L'elenco di riferimento seguente contiene dei termini importanti che vengono citati in questo capitolo:

- Elemento: voce singola di un documento XML che corrisponde a quanto contenuto tra un tag iniziale e un tag finale (i tag sono inclusi). Gli elementi XML possono consistere di dati o altri elementi oppure essere vuoti.
- Elemento vuoto: elemento XML che non comprende elementi di livello inferiore. Gli elementi vuoti sono spesso scritti come elementi a chiusura automatica (come nel caso di `<element/>`).
- Documento: struttura XML. Un documento XML può contenere un numero qualsiasi di elementi (o comprendere un unico elemento vuoto); tuttavia, in ogni documento XML deve necessariamente essere presente un solo elemento principale di livello superiore che funge da contenitore di tutti gli altri elementi.
- Nodo: sinonimo di elemento XML.
- Attributo: valore associato a un elemento scritto nel tag di apertura in formato `attributename="value"` piuttosto che scritto come elemento secondario separato nidificato all'interno dell'elemento.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Tutti questi esempi includono la chiamata appropriata alla funzione `trace()`. Per provare gli esempi di codice contenuti in questo capitolo:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.
4. Eseguire il programma selezionando Controllo > Prova filmato.

   I risultati della funzione `trace()` vengono visualizzati nel pannello Output.

Per ulteriori informazioni sulle tecniche per la prova degli esempi di codice, vedere "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

L'approccio di E4X all'elaborazione XML

La specifica ECMAScript for XML definisce una serie di classi e funzionalità per l'elaborazione dei dati XML. Tali classi e funzionalità sono definite globalmente *E4X*. ActionScript 3.0 comprende le seguenti classi E4X: XML, XMLList, QName e Namespace.

I metodi, le proprietà e gli operatori delle classi E4X sono stati progettati con gli obiettivi seguenti:

- Semplicità: laddove possibile, E4X rende il codice di gestione dei dati XML facile da scrivere e da capire.
- Uniformità: i metodi e la struttura fondanti di E4X sono uniformi tra loro e conformi alle altri porzioni di ActionScript.
- Familiarità: gli operatori per la manipolazione dei dati XML sono operatori conosciuti, come l'operatore punto (.).

NOTA	ActionScript 2.0 prevedeva una classe XML che in ActionScript 3.0 è stata ridenominata XMLDocument, allo scopo di non creare un conflitto con la classe XML di ActionScript 3.0 che fa parte di E4X. In ActionScript 3.0, le classi precedenti - XMLDocument, XMLNode, XMLParser e XMLTag - sono state inserite nel pacchetto flash.xml per assicurare il supporto retroattivo. Le nuove classi E4X sono classi principali che per essere utilizzate non richiedono l'importazione di un pacchetto. Il presente capitolo non approfondisce le classi XML usate da ActionScript 2.0. Per maggiori dettagli a questo riguardo, consultare il pacchetto flash.xml nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Ecco un esempio di manipolazione di dati con E4X:

```
var myXML:XML =
  <order>
    <item id='1'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>
```

Nonostante accada spesso che le applicazioni carichino i dati XML da un'origine esterna, come il servizio Web o un feed RSS, per motivi di chiarezza gli esempi presentati in questo capitolo assegnano valori letterali ai dati XML.

Come illustra la porzione di codice seguente, E4X comprende alcuni operatori intuitivi, come il punto (`.`) e l'identificatore di attributi (`@`) che vengono usati per accedere alle proprietà e agli attributi XML:

```
trace(myXML.item[0].menuName); // Output: burger
trace(myXML.item.(@id==2).menuName); // Output: fries
trace(myXML.item.(menuName=="burger").price); // Output: 3.95
```

Usare il metodo `appendChild()` per assegnare un nuovo nodo secondario al codice XML, come illustrato nello snippet di codice seguente:

```
var newItem:XML =
  <item id="3">
    <menuName>medium cola</menuName>
    <price>1.25</price>
  </item>

myXML.appendChild(newItem);
```

Usare gli operatori `@` e `.` non solo per leggere i dati, ma anche per assegnarli, come illustrato di seguito:

```
myXML.item[0].menuName="regular burger";
myXML.item[1].menuName="small fries";
myXML.item[2].menuName="medium cola";

myXML.item.(menuName=="regular burger").@quantity = "2";
myXML.item.(menuName=="small fries").@quantity = "2";
myXML.item.(menuName=="medium cola").@quantity = "2";
```

Usare un ciclo `for` per eseguire un'iterazione sui nodi XML, come illustrato di seguito:

```
var total:Number = 0;
for each (var property:XML in myXML.item)
{
  var q:int = Number(property.@quantity);
  var p:Number = Number(property.price);
  var itemTotal:Number = q * p;
  total += itemTotal;
  trace(q + " " + property.menuName + " $" + itemTotal.toFixed(2))
}
trace("Total: $", total.toFixed(2));
```

Oggetti XML

Un oggetto XML può rappresentare un elemento, un attributo, un commento, un'istruzione di elaborazione o un elemento di testo XML.

Gli oggetti XML sono classificati come oggetti con *contenuto semplice* o *contenuto complesso*. Un oggetto XML che comprende nodi secondari è classificato come oggetto con contenuto complesso. Gli oggetti XML con contenuto semplice sono gli attributi, i commenti, le istruzioni di elaborazione e i nodi di testo.

Ad esempio, nel seguente oggetto XML è presente contenuto complesso perché comprende un commento e un'istruzione di elaborazione:

```
XML.ignoreComments = false;
XML.ignoreProcessingInstructions = false;
var x1:XML =
  <order>
    <!--Questo è un commento. -->
    <?PROC_INSTR sample ?>
    <item id='1'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>
```

Come illustra l'esempio seguente, ora è possibile usare i metodi `comments()` e `processingInstructions()` per creare nuovi oggetti XML, un commento e una istruzione di elaborazione:

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

Proprietà di XML

La classe XML ha cinque proprietà statiche:

- Le proprietà `ignoreComments` e `ignoreProcessingInstructions` determinano se i commenti o le istruzioni di elaborazione devono essere ignorati nel momento in cui l'oggetto XML viene analizzato.
- La proprietà `ignoreWhitespace` determina se gli spazi vuoti devono essere ignorati nei tag degli elementi e nelle espressioni incorporate separate unicamente da spazi vuoti.
- Le proprietà `prettyIndent` e `prettyPrinting` vengono usate per formattare il testo restituito dai metodi `toString()` e `toXMLString()` della classe XML.

Per maggiori dettagli su queste proprietà, consultare la *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Metodi XML

I metodi seguenti permettono di eseguire operazioni con la struttura gerarchica degli oggetti XML:

- `appendChild()`
- `child()`
- `childIndex()`
- `children()`
- `descendants()`
- `elements()`
- `insertChildAfter()`
- `insertChildBefore()`
- `parent()`
- `prependChild()`

I metodi seguenti permettono di eseguire operazioni con gli attributi degli oggetti XML:

- `attribute()`
- `attributes()`

I metodi seguenti permettono di eseguire operazioni con le proprietà degli oggetti XML:

- `hasOwnProperty()`
- `propertyIsEnumerable()`
- `replace()`
- `setChildren()`

I metodi seguenti permettono di eseguire operazioni con i nomi completi e gli spazi dei nomi:

- `addNamespace()`
- `inScopeNamespaces()`
- `localName()`
- `name()`
- `namespace()`
- `namespaceDeclarations()`
- `removeNamespace()`
- `setLocalName()`
- `setName()`
- `setNamespace()`

I metodi seguenti permettono di eseguire operazioni con il contenuto XML e determinare alcuni tipi di contenuto XML:

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `nodeKind()`
- `processingInstructions()`
- `text()`

I metodi seguenti permettono di convertire in stringhe e formattare gli oggetti XML:

- `defaultSettings()`
- `setSettings()`
- `settings()`
- `normalize()`
- `toString()`
- `toXMLString()`

Rimangono alcuni metodi aggiuntivi:

- `contains()`
- `copy()`
- `valueOf()`
- `length()`

Per maggiori dettagli su questi metodi, consultare la *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Oggetti XMLList

Un'istanza di XMLList rappresenta una raccolta arbitraria di oggetti XML e può contenere documenti XML completi, frammenti XML o i risultati di una query XML.

I metodi seguenti permettono di eseguire operazioni con la struttura gerarchica degli oggetti XMLList:

- `child()`
- `children()`
- `descendants()`
- `elements()`
- `parent()`

I metodi seguenti permettono di eseguire operazioni con gli attributi degli oggetti XMLList:

- `attribute()`
- `attributes()`

I metodi seguenti permettono di eseguire operazioni con le proprietà degli oggetti XMLList:

- `hasOwnProperty()`
- `propertyIsEnumerable()`

I metodi seguenti permettono di eseguire operazioni con il contenuto XML e determinare alcuni tipi di contenuto XML:

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `processingInstructions()`
- `text()`

I metodi seguenti permettono di convertire in stringhe e formattare gli oggetti XMLList:

- `normalize()`
- `toString()`
- `toXMLString()`

Rimangono alcuni metodi aggiuntivi:

- `contains()`
- `copy()`
- `length()`
- `valueOf()`

Per maggiori dettagli su questi metodi, consultare la *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Gli oggetti XMLList che contengono esattamente un solo elemento XML possono usare tutte le proprietà e i metodi della classe XML in quanto un oggetto XMLList con un solo elemento XML viene considerato al pari di un oggetto XML. Nella porzione di codice dell'esempio seguente, poiché `doc.div` è un oggetto XMLList che contiene un solo elemento, è possibile usare il metodo `appendChild()` della classe XML:

```
var doc:XML =
    <body>
      <div>
        <p>Hello</p>
      </div>
    </body>;
doc.div.appendChild(<p>World</p>);
```

Per elenco delle proprietà e dei metodi di XML, vedere "Oggetti XML" a pagina 374.

Inizializzazione delle variabili di XML

Si può assegnare un letterale XML a un oggetto XML nel modo descritto di seguito:

```
var myXML:XML =
  <order>
    <item id='1'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>
```

Come illustrato nello snippet di codice seguente, è possibile usare anche la funzione di costruzione `new` per creare un'istanza di un oggetto XML partendo da una stringa che contiene dati XML:

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
               + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

Se i dati XML della stringa non hanno una sintassi corretta (se manca, ad esempio, un tag di chiusura), viene generato un errore in fase di runtime.

Si possono anche passare dati in base al riferimento (da altre variabili) a un oggetto XML, come illustrato nell'esempio seguente:

```
var tagname:String = "item";
var attributename:String = "id";
var attributevalue:String = “5”;
var content:String = "Chicken";
var x:XML = <{tagname} {attributename}={attributevalue}>{content}</
  {tagname}>;
trace(x.toXMLString())
  // Output: <item id="5">Chicken</item>
```

Per caricare dei dati XML da un URL, usare la classe URLLoader, come illustrato nell'esempio seguente:

```
import flash.events.Event;
import flash.net.URLLoader;
import flash.net.URLRequest;

var externalXML:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("xmlFile.xml");
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete);

function onComplete(event:Event):void
{
  var loader:URLLoader = event.target as URLLoader;
  if (loader != null)
  {
    externalXML = new XML(loader.data);
    trace(externalXML.toXMLString());
  }
  else
  {
    trace("loader is not a URLLoader!");
  }
}
```

Per leggere i dati XML da una connessione socket, usare la classe XMLSocket. Per ulteriori informazioni, vedere la voce relativa alla classe XMLSocket nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Assemblaggio e trasformazione di oggetti XML

Usare il metodo `prependChild()` o il metodo `appendChild()` per aggiungere una proprietà all'inizio o alla fine dell'elenco proprietà di un oggetto XML, come illustrato nell'esempio seguente:

```
var x1:XML = <p>Line 1</p>
var x2:XML = <p>Line 2</p>
var x:XML = <body></body>
x = x.appendChild(x1);
x = x.appendChild(x2);
x = x.prependChild(<p>Line 0</p>);
  // x == <body><p>Line 0</p><p>Line 1</p><p>Line 2</p></body>
```

Usare il metodo `insertChildBefore()` o il metodo `insertChildAfter()` per aggiungere una proprietà prima o dopo la proprietà specificata, nel modo seguente:

```
var x:XML =
  <body>
    <p>Paragraph 1</p>
    <p>Paragraph 2</p>
  </body>
var newNode:XML = <p>Paragraph 1.5</p>
x = x.insertChildAfter(x.p[0], newNode)
x = x.insertChildBefore(x.p[2], <p>Paragraph 1.75</p>)
```

Come illustrato dall'esempio seguente, è possibile utilizzare anche gli operatori parentesi graffe (`{` e `}`) per passare dati in base al riferimento (da altre variabili) durante la costruzione di oggetti XML:

```
var ids:Array = [121, 122, 123];
var names:Array = [["Murphy","Pat"], ["Thibaut","Jean"], ["Smith","Vijay"]]
var x:XML = new XML("<employeeList></employeeList>");

for (var i:int = 0; i < 3; i++)
{
  var newnode:XML = new XML();
  newnode =
    <employee id={ids[i]}>
      <last>{names[i][0]}</last>
      <first>{names[i][1]}</first>
    </employee>;

  x = x.appendChild(newnode)
}
```

Si possono assegnare proprietà e attributi a un oggetto XML usando l'operatore =, come nell'esempio seguente:

```
var x:XML =
  <employee>
    <lastname>Smith</lastname>
  </employee>
x.firstname = "Jean";
x.@id = "239";
```

In questo modo `x` dell'oggetto XML viene impostato su:

```
<employee id="239">
  <lastname>Smith</lastname>
  <firstname>Jean</firstname>
</employee>
```

È possibile usare gli operatori + e += per concatenare oggetti XMLList:

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

In questo modo `xList` dell'oggetto XMLList viene impostato su:

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

Lettura delle strutture XML

Una delle funzioni più sofisticate di XML è la sua abilità di creare strutture di dati complesse e nidificate mediante una stringa lineare di caratteri di testo. Durante il caricamento di dati di un oggetto XML, ActionScript analizza i dati e ne carica la struttura gerarchica in memoria (oppure genera un errore in fase di runtime se la sintassi dei dati XML contiene un errore).

Gli operatori e i metodi degli oggetti XML e XMLList rendono semplice la lettura della struttura dei dati XML.

Usare l'operatore punto (.) e l'operatore accessor discendente (..) per accedere alle proprietà secondarie di un oggetto XML. Si consideri l'oggetto XML seguente:

```
var myXML:XML =
  <order>
    <book ISBN="0942407296">
      <title>Baking Extravagant Pastries with Kumquats</title>
      <author>
        <lastName>Contino</lastName>
        <firstName>Chuck</firstName>
      </author>
      <pageCount>238</pageCount>
    </book>
    <book ISBN="0865436401">
      <title>Emu Care and Breeding</title>
      <editor>
        <lastName>Case</lastName>
        <firstName>Justin</firstName>
      </editor>
      <pageCount>115</pageCount>
    </book>
  </order>
```

L'oggetto `myXML.book` è un oggetto XMLList che contiene proprietà secondarie dell'oggetto `myXML` denominato `book`. Sono due oggetti XML, che corrispondono alle due proprietà `book` dell'oggetto `myXML`.

L'oggetto `myXML..lastName` è un oggetto XMLList contenente qualsiasi proprietà discendente denominata `lastName`. Sono due oggetti XML, che corrispondono ai due `lastName` dell'oggetto `myXML`.

L'oggetto `myXML.book.editor.lastName` è un oggetto XMLList che contiene qualsiasi elemento secondario denominato `lastName` di elementi secondari denominati `editor` di elementi secondari denominati `book` dell'oggetto `myXML`: in questo caso, un oggetto XMLList che contiene un solo oggetto XML (la proprietà `lastName` con il valore `"Case"`).

Accesso ai nodi principali e secondari

Il metodo `parent()` restituisce il nodo principale di un oggetto XML.

Per accedere a oggetti secondari specifici, si possono usare valori di indice ordinali di un elenco. Ad esempio, si consideri un oggetto XML `myXML` che dispone di due proprietà secondarie denominate `book`. A ogni proprietà `book` secondaria è associato un numero di indice:

```
myXML.book[0]
myXML.book[1]
```

Per accedere a un elemento secondario di secondo livello è possibile specificare il numero di indice sia dell'elemento secondario di primo livello che di quello di secondo livello:

```
myXML.book[0].title[0]
```

Tuttavia, se `x.book[0]` dispone di una sola proprietà secondaria denominata `title`, è possibile omettere il riferimento al numero di indice, come illustrato di seguito:

```
myXML.book[0].title
```

Analogamente, se esiste un solo oggetto secondario book dell'oggetto `x`, e se quell'oggetto secondario dispone di un solo oggetto title, si possono omettere entrambi i riferimenti di indice, nel modo seguente:

```
myXML.book.title
```

È possibile usare il metodo `child()` per andare agli elementi secondari con nomi basati su una variabile o un'espressione, come illustrato nell'esempio seguente:

```
var myXML:XML =
    <order>
      <book>
        <title>Dictionary</title>
      </book>
    </order>;

var childName:String = "book";

trace(myXML.child(childName).title) // output: Dictionary
```

Accesso agli attributi

Usare il simbolo @ (l'operatore identificatore di attributi) per accedere agli attributi di un oggetto XML o XMLList, come nel codice seguente:

```
var employee:XML =
  <employee id="6401" code="233">
    <lastName>Wu</lastName>
    <firstName>Erin</firstName>
  </employee>;
trace(employee.@id); // 6401
```

Per accedere a tutti gli attributi di un oggetto XML o XMLList si può usare il carattere jolly * combinato al simbolo @, come illustrato nel codice seguente:

```
var employee:XML =
  <employee id="6401" code="233">
    <lastName>Wu</lastName>
    <firstName>Erin</firstName>
  </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

Si può usare il metodo `attribute()` o il metodo `attributes()` per accedere a un attributo specifico o a tutti gli attributi di un oggetto XML o XMLList, come illustrato nel codice seguente:

```
var employee:XML =
  <employee id="6401" code="233">
    <lastName>Wu</lastName>
    <firstName>Erin</firstName>
  </employee>;
trace(employee.attribute("id")); // 6401
trace(employee.attribute("*").toXMLString());
// 6401
// 233
trace(employee.attributes().toXMLString());
// 6401
// 233
```

Si noti che per accedere agli attributi è anche possibile utilizzare la sintassi seguente, come illustrato nell'esempio seguente:

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

Tutte queste sono alternative equivalenti a `employee.@id`. La sintassi `employee.@id` rimane, tuttavia, l'approccio consigliato.

Filtraggio per attributo o elemento

Per filtrare gli elementi in base a un nome o a un valore di attributo specifico è possibile usare gli operatori parentesi: `(` e `)`. Osservare l'oggetto XML seguente:

```
var x:XML =
  <employeeList>
    <employee id="347">
      <lastName>Zmed</lastName>
      <firstName>Sue</firstName>
      <position>Data analyst</position>
    </employee>
    <employee id="348">
      <lastName>McGee</lastName>
      <firstName>Chuck</firstName>
      <position>Jr. data analyst</position>
    </employee>
  </employeeList>
```

Tutte le espressioni seguenti sono valide:

- `x.employee.(lastName == "McGee")` - Questo è il secondo nodo `employee`.
- `x.employee.(lastName == "McGee").firstName` - Questa è la proprietà `firstName` del secondo nodo `employee`.
- `x.employee.(lastName == "McGee").@id` - Questo è il valore dell'attributo `id` del secondo nodo `employee`.
- `x.employee.(@id == 347)` - Il primo nodo `employee`.
- `x.employee.(@id == 347).lastName` - Questa è la proprietà `lastName` del primo nodo `employee`.
- `x.employee.(@id > 300)` - Questo è un oggetto XMLList con entrambe le proprietà `employee`.
- `x.employee.(position.toString().search("analyst") > -1)` - Questo è un oggetto XMLList con entrambe le proprietà `position`.

Se si tenta di applicare un filtro usando attributi o elementi inesistenti, Adobe Flash Player genera un'eccezione. Ad esempio, l'ultima riga del codice seguente genera un errore perché non è presente un attributo `id` nel secondo elemento `p`:

```
var doc:XML =
       <body>
         <p id='123'>Hello, <b>Bob</b>.</p>
         <p>Hello.</p>
       </body>;
trace(doc.p.(@id == '123'));
```

Analogamente, l'ultima riga del codice seguente genera un errore perché non esiste una proprietà `b` del secondo elemento `p`:

```
var doc:XML =
       <body>
         <p id='123'>Hello, <b>Bob</b>.</p>
         <p>Hello.</p>
       </body>;
trace(doc.p.(b == 'Bob'));
```

Per evitare questo tipo di errori, è possibile identificare le proprietà per cui esistono attributi o elementi corrispondenti usando i metodi `attribute()` e `elements()`, come nel codice seguente:

```
var doc:XML =
       <body>
         <p id='123'>Hello, <b>Bob</b>.</p>
         <p>Hello.</p>
       </body>;
trace(doc.p.(attribute('id') == '123'));
trace(doc.p.(elements('b') == 'Bob'));
```

Inoltre, è possibile utilizzare anche il metodo `hasOwnProperty()`, come nel codice seguente:

```
var doc:XML =
       <body>
         <p id='123'>Hello, <b>Bob</b>.</p>
         <p>Hello.</p>
       </body>;
trace(doc.p.(hasOwnProperty('@id') && @id == '123'));
trace(doc.p.(hasOwnProperty('b') && b == 'Bob'));
```

Uso delle istruzioni for..in e for each..in

ActionScript 3.0 include le istruzioni `for..in` e `for each..in` per eseguire iterazioni sugli oggetti XMLList. Osservare, ad esempio, il seguente oggetto XML, `myXML`, e l'oggetto XMLList, `myXML.item`. L'oggetto XMLList, `myXML.item`, è composto dai due nodi `item` dell'oggetto XML.

```
var myXML:XML =
  <order>
    <item id='1' quantity='2'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2' quantity='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>;
```

L'istruzione `for..in` permette di eseguire iterazioni su una serie di proprietà di un XMLList:

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
  total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

L'istruzione `for each..in` permette di eseguire iterazioni sulle proprietà di XMLList:

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
  total2 += prop.@quantity * prop.price;
}
```

Uso dello spazio dei nomi XML

Lo spazio dei nomi di un oggetto (o documento) XML indica il tipo di dati che l'oggetto contiene. Ad esempio, per inviare e consegnare dati XML a un servizio Web che usa il protocollo SOAP, si dichiara lo spazio dei nomi nel tag di apertura di XML:

```
var message:XML =
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <soap:Body xmlns:w="http://www.test.com/weather/">
      <w:getWeatherResponse>
        <w:tempurature >78</w:tempurature>
      </w:getWeatherResponse>
    </soap:Body>
  </soap:Envelope>;
```

Lo spazio dei nomi è caratterizzato da un prefisso, `soap`, e da un URI che definisce lo spazio dei nomi, `http://schemas.xmlsoap.org/soap/envelope/`.

ActionScript 3.0 contiene la classe Namespace per le operazioni con gli spazi dei nomi XML. Per l'oggetto XML dell'esempio precedente, è possibile usare la classe Namespace come segue:

```
var soapNS:Namespace = message.namespace("soap");
trace(soapNS); // Output: http://schemas.xmlsoap.org/soap/envelope/

var wNS:Namespace = new Namespace("w", "http://www.test.com/weather/");
message.addNamespace(wNS);
var encodingStyle:XMLList = message.@soapNS::encodingStyle;
var body:XMLList = message.soapNS::Body;

message.soapNS::Body.wNS::GetWeatherResponse.wNS::tempurature = "78";
```

I seguenti metodi della classe XML permettono di eseguire operazioni con gli spazi dei nomi: `addNamespace()`, `inScopeNamespaces()`, `localName()`, `name()`, `namespace()`, `namespaceDeclarations()`, `removeNamespace()`, `setLocalName()`, `setName()` e `setNamespace()`.

La direttiva `default xml namespace` permette di assegnare uno spazio dei nomi predefinito agli oggetti XML. Nell'esempio seguente, sia `x1` che `x2` condividono il medesimo spazio dei nomi predefinito:

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");
default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

Conversione degli oggetti XML

Gli oggetti XML e XMLList possono essere convertiti in valori stringa. Analogamente, è possibile convertire stringhe in oggetti XML e XMLList. Tenere presente che tutti i valori di attributo, i nomi e i valori di testo XML sono stringhe. Le sezioni seguenti affrontano tutte queste forme di conversione XML.

Conversione degli oggetti XML e XMLList in stringhe

Le classi XML e XMLList comprendono i metodi `toString()` e `toXMLString()`. Il metodo `toXMLString()` restituisce una stringa che comprende tag, attributi, dichiarazioni di spazio dei nomi e contenuto dell'oggetto XML. Sugli oggetti XML dal contenuto complesso (cioè con elementi secondari), il metodo `toString()` agisce esattamente come il metodo `toXMLString()`. Per gli oggetti XML dal contenuto semplice (cioè quelli caratterizzati da un solo elemento di testo), il metodo `toString()` restituisce solo il contenuto di testo dell'elemento, come illustra l'esempio seguente:

```
var myXML:XML =
  <order>
    <item id='1' quantity='2'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
  <order>;

trace(myXML.item[0].menuName.toXMLString());
  // <menuName>burger</menuName>
trace(myXML.item[0].menuName.toString());
  // burger
```

Se si usa il metodo `trace()` senza specificare `toString()` o `toXMLString()`, i dati vengono automaticamente convertiti usando il metodo `toString()`, come dimostra il codice seguente:

```
var myXML:XML =
  <order>
    <item id='1' quantity='2'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
  <order>;

trace(myXML.item[0].menuName);
  // burger
```

Quando si usa il metodo `trace()` per le attività di debug del codice, spesso risulta utile usare il metodo `toXMLString()` per fare in modo che `trace()` restituisca dati più completi.

Conversione di stringhe in oggetti XML

Usare la funzione di costruzione `new XML()` per creare un oggetto XML da una stringa nel modo seguente:

```
var x:XML = new XML("<a>test</a>");
```

Se si tenta di convertire in XML una stringa che rappresenta dati XML non validi o con una sintassi XML errata, viene generato un errore in fase di runtime nel modo seguente:

```
var x:XML = new XML("<a>test"); // genera un errore
```

Conversione di valori attributo, nomi e valori di testo da stringhe

I valori attributo, i nomi e i valori di testo XML sono tipi di dati in formato stringa e potrebbe, pertanto, essere necessario convertirli in un tipo di dati diverso. Nel seguente esempio di codice viene utilizzata la funzione `Number()` per convertire i valori di testo in numeri:

```
var myXML:XML =
              <order>
                <item>
                  <price>3.95</price>
                </item>
                <item>
                  <price>1.00</price>
                </item>
              </order>;
```

```
var total:XML = <total>0</total>;
myXML.appendChild(total);

for each (var item:XML in myXML.item)
{
  myXML.total.children()[0] = Number(myXML.total.children()[0])
                            + Number(item.price.children()[0]);
}
trace(myXML.total); // 4.35;
```

Se nel codice non fosse stata aggiunta la funzione `Number()`, il codice avrebbe interpretato l'operatore + come l'operatore di concatenazione di stringhe e il metodo `trace()` dell'ultima riga avrebbe generato il valore seguente:

```
01.003.95
```

Lettura di documenti XML esterni

La classe URLLoader permette di caricare dati XML da un URL. Per usare il codice seguente all'interno delle applicazioni, sostituire il valore `XML_URL` dell'esempio con un URL valido:

```
var myXML:XML = new XML();
var XML_URL:String = "http://www.example.com/Sample3.xml";
var myXMLURL:URLRequest = new URLRequest(XML_URL);
var myLoader:URLLoader = new URLLoader(myXMLURL);
myLoader.addEventListener("complete", xmlLoaded);

function xmlLoaded(event:Event):void
{
  myXML = XML(myLoader.data);
  trace("Data loaded.");
}
```

È inoltre possibile usare la classe XMLSocket per impostare una connessione socket XML asincrona con un server. Per ulteriori informazioni, consultare la *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0* e

Esempio: Caricamento di dati RSS da Internet

L'applicazione di esempio RSSViewer dimostra come usare varie funzioni per manipolare dati XML in ActionScript, tra cui:

- Usare i metodi XML per leggere i dati XML sotto forma di feed RSS.
- Usare i metodi XML per assemblare i dati XML in formato HTML utilizzabile in campi di testo.

Il formato RSS è ampiamente utilizzato per raccogliere notizie tramite XML. Un file di dati RSS semplice ha un aspetto simile al seguente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
  <title>Alaska - Weather</title>
  <link>http://www.nws.noaa.gov/alerts/ak.html</link>
  <description>Alaska - Watches, Warnings and Advisories</description>

  <item>
    <title>
      Short Term Forecast - Taiya Inlet, Klondike Highway (Alaska)
    </title>
    <link>
      http://www.nws.noaa.gov/alerts/ak.html#A18.AJKNK.1900
    </link>
    <description>
      Short Term Forecast Issued At: 2005-04-11T19:00:00
      Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
      Homepage: http://pajk.arh.noaa.gov
    </description>
  </item>
  <item>
    <title>
      Short Term Forecast - Haines Borough (Alaska)
    </title>
    <link>
      http://www.nws.noaa.gov/alerts/ak.html#AKZ019.AJKNOWAJK.190000
    </link>
    <description>
      Short Term Forecast Issued At: 2005-04-11T19:00:00
      Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
      Homepage: http://pajk.arh.noaa.gov
    </description>
  </item>
</channel>
</rss>
```

L'applicazione SimpleRSS legge i dati RSS da Internet, li analizza per individuare i titoli, i collegamenti e le descrizioni, quindi restituisce i dati. La classe SimpleRSSUI costituisce l'interfaccia utente e chiama la classe SimpleRSS, che si occupa dell'elaborazione XML.

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione RSSViewer si trovano nella cartella Samples/RSSViewer. L'applicazione comprende i seguenti file:

File	Descrizione
RSSViewer.mxml o RSSViewer.fla	Il file principale dell'applicazione in Flash (FLA) o Flex (MXML)
com/example/programmingas3/rssViewer/RSSParser.as	Classe che contiene metodi che usano E4X per leggere i dati RSS (XML) e generare la rappresentazione HTML corrispondente.
RSSData/ak.rss	File RSS di esempio. L'applicazione è configurata in modo da leggere i dati RSS dal Web, presso un feed RSS Flex residente sul sito Adobe. In alternativa, non presenta difficoltà impostare l'applicazione in modo che legga i dati RSS da questo documento, che usa uno schema leggermente diverso rispetto a quello del feed RSS Flex.

Lettura e analisi dei dati XML

La classe RSSParser comprende il metodo `xmlLoaded()` che converte i dati RSS di input, memorizzati nella variabile `rssXML`, in una stringa che contiene dati in formato HTML, `rssOutput`.

Verso l'inizio del metodo, il codice indica lo spazio dei nomi XML predefinito se i dati RSS di origine comprendono uno spazio dei nomi predefinito:

```
if (rssXML.namespace("") != undefined)
{
  default xml namespace = rssXML.namespace("");
}
```

Le righe seguenti eseguono un ciclo tra il contenuto dei dati XML di origine ed esaminano tutte le proprietà discendenti denominate `item`:

```
for each (var item:XML in rssXML..item)
{
  var itemTitle:String = item.title.toString();
  var itemDescription:String = item.description.toString();
  var itemLink:String = item.link.toString();
  outXML += buildItemHTML(itemTitle,
              itemDescription,
              itemLink);
}
```

Le prime tre righe si limitano a impostare le variabili stringa che rappresentano le proprietà del titolo, della descrizione e del collegamento della proprietà `item` dei dati XML. La riga seguente chiama il metodo `buildItemHTML()` per ottenere i dati HTML sotto forma di oggetto XMLList, usando le tre nuove variabili come parametri.

Assemblaggio dei dati XMLList

I dati HTML (l'oggetto XMLList) hanno il seguente formato:

```
<b>itemTitle</b>
<p>
  itemDescription
  <br />
  <a href="link">
    <font color="#008000">More...</font>
  </a>
</p>
```

Le prime righe del metodo eliminano lo spazio dei nomi xml predefinito:

```
default xml namespace = new Namespace();
```

La direttiva `default xml namespace` ha un'area di validità a livello di blocco della funzione. Ciò significa che l'area di validità di questa dichiarazione è il metodo `buildItemHTML()`.

Le righe di codice che seguono assemblano l'oggetto XMLList basandosi sugli argomenti String passati alla funzione:

```
var body:XMLList = new XMLList();
body += new XML("<b>" + itemTitle + "</b>");
var p:XML = new XML("<p>" + itemDescription + "</p>");

var link:XML = <a></a>;
link.@href = itemLink; // <link href="itemLinkString"></link>
link.font.@color = "#008000";
     // <font color="#008000"></font></a>
     // 0x008000 = green
link.font = "More...";

p.appendChild(<br/>);
p.appendChild(link);
body += p;
```

Questo oggetto XMLList rappresenta dei dati in formato stringa adatti a un campo di testo HTML di ActionScript.

Il metodo `xmlLoaded()` usa il valore restituito dal metodo `buildItemHTML()` e lo converte in stringa:

```
XML.prettyPrinting = false;
rssOutput = outXML.toXMLString();
```

Estrazione del titolo del feed RSS e invio di un evento personalizzato

Il metodo `xmlLoaded()` imposta una variabile di stringa `rssTitle` basandosi sulle informazioni dei dati XML del feed RSS di origine:

```
rssTitle = rssXML.channel.title.toString();
```

Infine, il metodo `xmlLoaded()` genera un evento che notifica l'applicazione che i dati sono stati analizzati e sono disponibili:

```
       dataWritten = new Event("dataWritten", true);
```

CAPITOLO 12

Programmazione degli elementi visivi

12

La programmazione degli elementi visivi in ActionScript 3.0 consente di lavorare con elementi visualizzati sullo stage di Adobe Flash Player 9. In questo capitolo vengono descritti i concetti base del lavoro con gli elementi sullo schermo. Vengono inoltre fornite informazioni dettagliate sull'organizzazione programmatica degli elementi visivi e sulla creazione di classi personalizzate per la visualizzazione degli oggetti.

Sommario

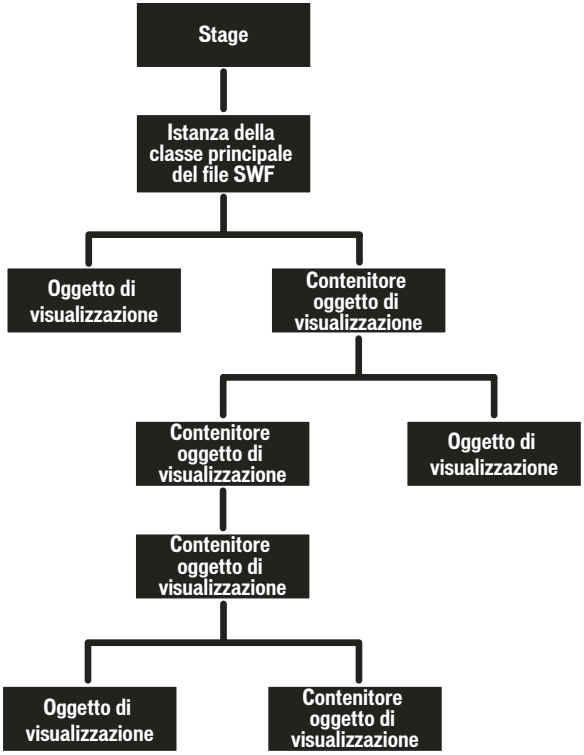
Elementi fondamentali della programmazione degli elementi visivi

Introduzione alla programmazione degli elementi visivi

Ogni applicazione costruita con ActionScript 3.0 presenta una gerarchia di oggetti visualizzati conosciuta come *elenco di visualizzazione*. L'elenco di visualizzazione contiene tutti gli elementi visibili dell'applicazione. Gli elementi di visualizzazione rientrano i uno dei seguenti gruppi:

- Stage

  Lo stage è un contenitore di base di oggetti di visualizzazione. Ogni applicazione presenta un oggetto stage contenente tutti gli oggetti visualizzati sullo schermo. Lo stage è il contenitore di primo livello e si trova in cima alla gerarchia dell'elenco di visualizzazione:



  Ogni file SWF presenta una classe ActionScript associata, conosciuta come *classe principale del file SWF*. Quando Flash Player apre un file SWF in una pagina HTML, Flash Player richiama la funzione di costruzione per quella classe e l'istanza che viene creata (che è sempre un tipo di oggetto di visualizzazione) viene aggiunta come elemento secondario dell'oggetto stage. La classe principale di un file SWF estende sempre la classe Sprite (Per ulteriori informazioni, vedere "Vantaggi dell'elenco di visualizzazione" a pagina 403).

  Per accedere allo stage è necessario passare dalla proprietà `stage` di una qualsiasi istanza di DisplayObject. Per ulteriori informazioni, vedere "Impostazione delle proprietà dello stage" a pagina 415.

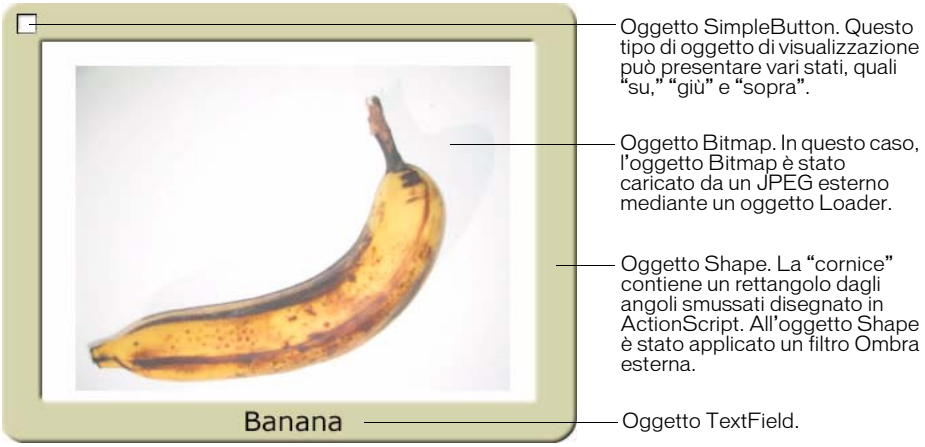
- Oggetti di visualizzazione

  In ActionScript 3.0, tutti gli elementi che appaiono sullo schermo di una applicazione sono tipi di *oggetti di visualizzazione*. Il pacchetto flash.display include una classe DisplayObject che è una classe base estesa da un numero di altre classi. Tali classi differenti rappresentano vari tipi di oggetti di visualizzazione, quali forme vettoriali, clip filmato e campi di testo, solo per citarne alcuni. Per una panoramica delle classi incorporate, vedere "Vantaggi dell'elenco di visualizzazione" a pagina 403.

- Contenitori di oggetti di visualizzazione

  I contenitori di oggetti di visualizzazione sono speciali tipi di oggetti che, oltre ad avere una propria rappresentazione visiva, possono contenere oggetti secondari che sono anch'essi oggetti di visualizzazione.

  La classe DisplayObjectContainer è una sottoclasse della classe DisplayObject. Un oggetto DisplayObjectContainer può contenere più oggetti di visualizzazione nel proprio *elenco secondario*. Ad esempio, l'illustrazione seguente mostra un tipo di oggetto DisplayObjectContainer conosciuto come Sprite contenente vari oggetti di visualizzazione:



  Nell'ambito della discussione sugli oggetti di visualizzazione, gli oggetti DisplayObjectContainer sono anche conosciuti come *contenitori di oggetti di visualizzazione* o più semplicemente *contenitori*.

  Nonostante tutti gli oggetti di visualizzazione visibili ereditino dalla classe DisplayObject, il tipo di ciascun oggetto corrisponde a quello di una specifica sottoclasse di DisplayObject. Ad esempio, esiste una funzione di costruzione per la classe Shape o per la classe Video, ma non vi è alcuna funzione di costruzione per la classe DisplayObject.

  Come accennato in precedenza, lo stage è un contenitore di oggetti di visualizzazione.

Attività comuni di programmazione degli elementi visivi

Poiché una parte consistente dalla programmazione in ActionScript prevede la creazione e la manipolazione di elementi visivi, le attività relative alla programmazione degli elementi visivi sono numerose. In questo capitolo sono descritte una serie di attività comuni a tutti gli oggetti di visualizzazione, quali:

- Uso dell'elenco di visualizzazione e dei contenitori degli oggetti di visualizzazione
  - Aggiunta di oggetti di visualizzazione all'elenco di visualizzazione
  - Rimozione di oggetti dall'elenco di visualizzazione
  - Spostamento degli oggetti tra i contenitori di visualizzazione
  - Spostamento degli oggetti davanti o dietro altri oggetti
- Operazioni con lo stage
  - Impostazione della frequenza fotogrammi
  - Controllo della modifica in scala dello stage
  - Uso della modalità a schermo intero
- Gestione di eventi di oggetti di visualizzazione
- Posizionamento degli oggetti di visualizzazione, inclusa creazione dell'interazione di trascinamento della selezione
- Ridimensionamento, modifica in scala e rotazione degli oggetti di visualizzazione
- Applicazione di metodi di fusione, trasformazioni di colore e trasparenza agli oggetti di visualizzazione
- Mascheratura degli oggetti di visualizzazione
- Animazione degli oggetti di visualizzazione
- Caricamento di contenuto di visualizzazione esterno (quali immagini o file SWF)

Nei capitoli successivi di questo manuale vengono descritte altre attività relative agli oggetti di visualizzazione, alcune delle quali sono applicabili a tutti i tipi di oggetti, mentre altre sono associate solo a tipi specifici di oggetti di visualizzazione:

- Disegno di grafica vettoriale con ActionScript sugli oggetti di visualizzazione, in Capitolo 14, "Uso dell'API di disegno" a pagina 481.
- Applicazione di trasformazioni geometriche agli oggetti di visualizzazione, in Capitolo 13, "Operazioni con le funzioni geometriche" a pagina 463.
- Applicazione di effetti grafici, quali sfocatura, alone, ombra esterna, ecc. agli oggetti di visualizzazione, in Capitolo 15, "Filtraggio degli oggetti di visualizzazione" a pagina 501.

- Uso delle caratteristiche specifiche di MovieClip, in Capitolo 16, “Operazioni con i clip filmato” a pagina 529
- Uso degli oggetti TextField, in Capitolo 17, “Operazioni con il testo” a pagina 547
- Uso delle immagini bitmap, in Capitolo 18, “Operazioni con le bitmap” a pagina 577.
- Uso delle immagini bitmap, in Capitolo 19, “Operazioni con i file video” a pagina 593.

Termini e concetti importanti

Il seguente elenco di riferimento contiene termini importanti utilizzati nel presente capitolo:

- Alfa: valore cromatico che rappresenta la quantità di trasparenza (o, più correttamente, il grado di opacità) di un colore. Ad esempio, un colore con un valore di canale alfa del 60% mostra solo il 60% della sua forza, mentre per il restante 40% è trasparente.
- Immagine bitmap: immagine definita nel computer come una griglia (righe e colonne) di pixel colorati. Generalmente, le immagini bitmap includono foto digitali e altre immagini simili.
- Modalità di fusione: specifica come deve interagire il contenuto di due immagini che si sovrappongono. Generalmente, un’immagine opaca posta sopra un’altra immagine blocca semplicemente l’immagine sottostante, che risulta completamente invisibile; tuttavia, vi sono varie modalità di fusione che consentono ai colori delle immagini di fondersi in diversi modi, così che il contenuto risultante corrisponda a una sorta di combinazione delle due immagini.
- Elenco di visualizzazione: gerarchia di oggetti di visualizzazione che verranno resi come contenuto visibile sullo schermo da Flash Player. Lo stage corrisponde al livello principale dell’elenco di visualizzazione e tutti gli oggetti di visualizzazione che sono associati allo stage o a uno dei suoi elementi secondari vanno a formare l’elenco di visualizzazione (anche se l’oggetto non è ancora stato sottoposto a rendering, ad esempio, si trova al di fuori dei confini dello stage).
- Oggetto di visualizzazione: oggetto che rappresenta contenuto visivo in Flash Player. Solo gli oggetti di visualizzazione possono essere inclusi nell’elenco di visualizzazione e tutte le classi degli oggetti di visualizzazione sono sottoclassi della classe DisplayObject.
- Contenitore di oggetti di visualizzazione: tipo speciale di oggetto di visualizzazione che può contenere oggetti di visualizzazione secondari oltre ad avere una propria rappresentazione visiva (generalmente).

- Classe principale del file SWF: classe che definisce il comportamento dell'oggetto di visualizzazione più esterno di un file SWF, che concettualmente è la classe dello stesso file SWF. Ad esempio, un file SWF creato con lo strumento di creazione di Flash presenta una "linea temporale principale" che contiene tutte le altre linee temporali; la classe principale del file SWF è la classe della quale la linea temporale principale è un'istanza.
- Effetto maschera: tecnica per nascondere dalla visualizzazione alcune porzioni di immagine (oppure per consentire di visualizzare solo certe parti di un'immagine). Le porzioni nascoste diventano trasparenti, in modo che il contenuto sottostante sia visibile. Il termine è riferito al nastro di mascheratura usato dai pittori per impedire al colore di venire applicato in alcune aree.
- Stage: contenitore visivo che costituisce la base o lo sfondo di tutto il contenuto visibile in un file SWF.
- Trasformazione: modifica di una caratteristica visiva di un'immagine, come rotazione di un oggetto, modifica della scala, inclinazione o distorsione della forma oppure alterazione del colore.
- Immagine vettoriale: immagine definita nel computer come linee e forme disegnate con particolari caratteristiche (quali spessore, lunghezza, dimensione, angolo e posizione).

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché il capitolo spiega come creare e manipolare il contenuto visivo, tutti gli esempi di codice in esso riportati creano oggetti visivi e li visualizzano sullo schermo; la prova dell'esempio prevede la visualizzazione del risultato in Flash Player invece della visualizzazione dei valori delle variabili come nei capitoli precedenti. Per provare gli esempi di codice contenuti in questo capitolo:

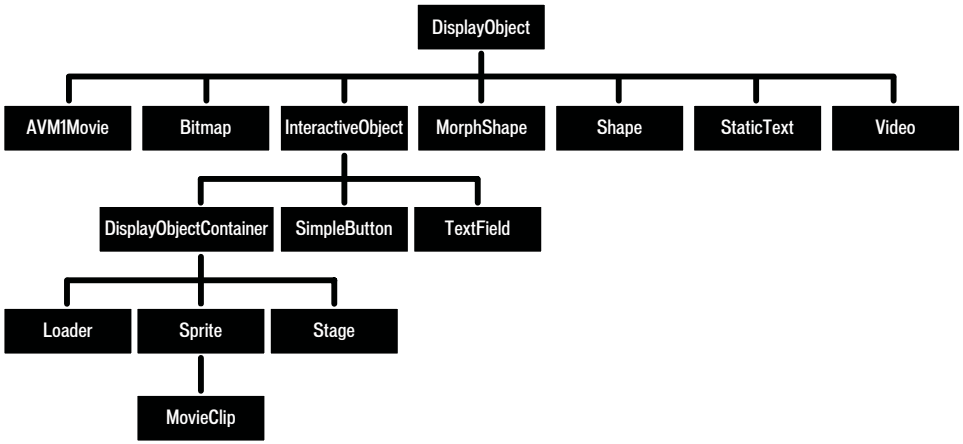
1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.
4. Eseguire il programma selezionando Controllo > Prova filmato.

   I risultati del codice vengono visualizzati sullo schermo e ogni eventuale chiamata alla funzione `trace()` viene visualizzata nel pannello Output.

Per ulteriori informazioni sulle tecniche per la prova degli esempi di codice, vedere "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Classi di visualizzazione di base

Il pacchetto flash.display di ActionScript 3.0 include una serie di classi per gli oggetti visivi che possono essere visualizzati in Flash Player. L'illustrazione seguente mostra le relazioni tra le sottoclassi di tali classi di oggetti di visualizzazione di base.



Nell'illustrazione è riportata l'ereditarietà di classe delle classi di oggetti di visualizzazione. Si noti che alcune di queste classi, in particolare StaticText, TextField e Video, pur non essendo nel pacchetto flash.display, ereditano comunque dalla classe DisplayObject.

Tutte le classi che estendono la classe DisplayObject ne ereditano i metodi e le proprietà. Per ulteriori informazioni, vedere "Proprietà e metodi della classe DisplayObject" a pagina 407.

È possibile creare un'istanza di oggetti appartenenti alle seguenti classi contenute nel pacchetto flash.display:

- Bitmap - La classe Bitmap consente di definire oggetti bitmap, sia caricati da file esterni che creati mediante rendering in ActionScript. Per caricare bitmap da file esterni utilizzare la classe Loader. È possibile caricare file GIF, JPG o PNG. È inoltre possibile creare un oggetto BitmapData contenente dati personalizzati e un oggetto Bitmap che impiega tali dati. I metodi della classe BitmapData possono essere utilizzati per alterare i file bitmap, sia che siano stati caricati o creati in ActionScript. Per ulteriori informazioni, vedere "Caricamento di oggetti di visualizzazione" a pagina 450 e il Capitolo 18, "Operazioni con le bitmap" a pagina 577.
- Loader - La classe Loader consente di caricare risorse esterne (sia file SWF che immagini). Per ulteriori informazioni, vedere "Caricamento dinamico di contenuto di visualizzazione" a pagina 450.

- Shape - La classe Shape consente di creare grafica vettoriale, quali rettangoli, linee, cerchi e così via. Per ulteriori informazioni, vedere Capitolo 14, “Uso dell’API di disegno” a pagina 481.
- SimpleButton - L’oggetto SimpleButton è la rappresentazione di ActionScript di un simbolo di pulsante di Flash. Un’istanza SimpleButton può avere tre diversi stati del pulsante: su, giù e sopra.
- Sprite - Un oggetto Sprite può contenere immagini proprie e oggetti di visualizzazione secondari. (La classe Sprite estende la classe DisplayObjectContainer). Per ulteriori informazioni, vedere “Uso dei contenitori degli oggetti di visualizzazione” a pagina 408 e il Capitolo 14, “Uso dell’API di disegno” a pagina 481.
- MovieClip - Un oggetto MovieClip è la versione di ActionScript di un simbolo di clip filmato creato in Flash. In pratica, un oggetto MovieClip è simile a un oggetto Sprite, con in più una linea temporale. Per ulteriori informazioni, vedere Capitolo 16, “Operazioni con i clip filmato” a pagina 529.

Le seguenti classi, che non si trovano nel pacchetto flash.display, sono sottoclassi della classe DisplayObject:

- La classe TextField, inclusa nel pacchetto flash.text, è un oggetto di visualizzazione per l’inserimento e la visualizzazione di testo. Per ulteriori informazioni, vedere Capitolo 17, “Operazioni con il testo” a pagina 547.
- La classe Video, inclusa nel pacchetto flash.media, è un oggetto di visualizzazione per l’inserimento e la visualizzazione di file video. Per ulteriori informazioni, consultare il Capitolo 19, “Operazioni con i file video” a pagina 593.

Le classi seguenti contenute nel pacchetto flash.display estendono la classe DisplayObject, tuttavia non è possibile creare istanze di tali classi. Al contrario, essere fungono da classi principali per altri oggetti di visualizzazione, combinando più funzionalità comuni in un’unica classe.

- AVM1Movie - La classe AVM1Movie viene utilizzata per rappresentare file SWF caricati creati in ActionScript 1.0 e 2.0.
- DisplayObjectContainer - Le classi Loader, Stage, Sprite e MovieClip possono estendere la classe DisplayObjectContainer. Per ulteriori informazioni, vedere “Uso dei contenitori degli oggetti di visualizzazione” a pagina 408.
- InteractiveObject - InteractiveObject è la classe base di tutti gli oggetti utilizzati per interagire con il mouse e la tastiera. Gli oggetti SimpleButton, TextField, Video, Loader, Sprite, Stage e MovieClip sono tutti sottoclassi di InteractiveObject. Per ulteriori informazioni sulla creazione di interazioni mediante mouse o tastiera, vedere Capitolo 21, “Rilevamento dell’input dell’utente” a pagina 671.

- MorphShape - Questi oggetti vengono creati quando si crea un'interpolazione di forma nello strumento di creazione di Flash. Non è possibile creare un'istanza di questi oggetti in ActionScript, tuttavia vi si può accedere dall'elenco di visualizzazione.
- Stage - La classe Stage estende la classe DisplayObjectContainer. Esiste un'istanza di Stage per ogni applicazione e tale istanza si trova al primo posto nella gerarchia dell'elenco di visualizzazione. Per accedere alla classe Stage utilizzare la proprietà `stage` di una qualsiasi istanza di DisplayObject. Per ulteriori informazioni, vedere "Impostazione delle proprietà dello stage" a pagina 415.

Anche la classe StaticText, contenuta nel pacchetto flash.text, estende la classe DisplayObject, ma non è possibile crearne un'istanza nel codice. I campi di testo statici vengono creati unicamente in Adobe Flash CS3 Professional.

Vantaggi dell'elenco di visualizzazione

In ActionScript 3.0 vi sono classi separate per vari tipi di oggetti di visualizzazione. In ActionScript 1.0 e 2.0, molti oggetti dello stesso tipo venivano inclusi tutti in un'unica classe: la classe MovieClip.

Una tale individualizzazione delle classe e la struttura gerarchica degli elenchi di visualizzazione presentano i seguenti vantaggi:

- Rendering più efficiente e impiego della memoria ridotto
- Gestione della profondità migliorata
- Ricerca completa dell'elenco di visualizzazione
- Oggetti di visualizzazione fuori elenco
- Più agevole creazione di sottoclassi degli oggetti di visualizzazione

Tali vantaggi vengono descritti più dettagliatamente nelle sezioni che seguono.

Rendering più efficiente e file di dimensioni più ridotte

In ActionScript 1.0 e 2.0, era possibile disegnare forme solo all'interno di oggetti MovieClip. ActionScript 3.0 offre classi di oggetti di visualizzazione più semplici nelle quali è possibile disegnare forme. Poiché queste classi di oggetti di visualizzazione di ActionScript 3.0 non includono la serie completa dei metodi e delle proprietà incluse negli oggetti MovieClip, l'impiego della memoria e delle risorse risulta più contenuto.

Ad esempio, ogni oggetto MovieClip include proprietà per la linea temporale del clip filmato, mentre l'oggetto Shape non le contiene. Le proprietà per la gestione della linea temporale possono richiedere grandi quantità di memoria e di risorse del processore. In ActionScript 3.0, l'uso dell'oggetto Shape consente di ottimizzare le risorse del sistema, in quanto è molto meno complesso dell'oggetto MovieClip. Flash Player non richiede la gestione di proprietà MovieClip inutilizzate, con conseguenti miglioramenti in termini di velocità e di impiego della memoria da parte degli oggetti.

Gestione della profondità migliorata

In ActionScript 1.0 e 2.0, la profondità veniva gestita mediante uno schema di gestione della profondità lineare e metodi quali `getNextHighestDepth()`.

ActionScript 3.0 include la classe DisplayObjectContainer, che presenta metodi e proprietà più pratici per la gestione della profondità e degli oggetti di visualizzazione.

In ActionScript 3.0, se un oggetto di visualizzazione viene spostato in una nuova posizione nell'elenco secondario di un'istanza di DisplayObjectContainer, gli altri elementi secondari presenti nel contenitore degli oggetti di visualizzazione vengono automaticamente riposizionati e reindicizzati all'interno del contenitore.

Inoltre, in ActionScript 3.0 è sempre possibile scoprire tutti gli oggetti secondari di un contenitore di oggetti di visualizzazione. Ogni istanza di DisplayObjectContainer presenta una proprietà `numChildren` che elenca il numero di elementi secondari presenti nel contenitore degli oggetti di visualizzazione. E poiché l'elenco secondario di un contenitore di oggetti di visualizzazione è sempre un elenco indicizzato, è possibile esaminare ogni singolo oggetto dell'elenco dalla posizione di indice 0 fino all'ultima posizione (`numChildren - 1`). Ciò non era supportato dai metodi e dalle proprietà degli oggetti MovieClip in ActionScript 1.0 e 2.0.

In ActionScript 3.0, gli elenchi di visualizzazione possono essere letti in modo sequenziale senza alcuna difficoltà, in quanto non vi sono spazi vuoti tra i numeri di indice degli elenchi secondari di contenitori di oggetti di visualizzazione. Leggere gli elenchi di visualizzazione e gestire la profondità degli oggetti è ora molto più semplice e immediato rispetto a quanto avveniva in ActionScript 1.0 e 2.0. In ActionScript 1.0 e 2.0, un clip filmato poteva contenere oggetti con spazi intermittenti nell'ordine di profondità che rendevano difficoltosa la lettura dell'elenco degli oggetti. In ActionScript 3.0, ogni elenco secondario di un contenitore di oggetti di visualizzazione è inserito internamente in una cache come un array, consentendo operazioni di ricerca (per indice) molto più rapide. Anche la ripetizione ciclica di tutti gli elementi secondari di un contenitore di oggetti di visualizzazione risulta molto veloce.

In ActionScript 3.0, è possibile accedere agli elementi secondari di un contenitore di oggetti di visualizzazione mediante il metodo `getChildByName()` della classe DisplayObjectContainer.

Ricerca completa dell'elenco di visualizzazione

In ActionScript 1.0 e 2.0, non era possibile accedere ad alcuni oggetti, quali forme vettoriali, disegnate nello strumento di creazione di Flash. In ActionScript 3.0, si può accedere a tutti gli oggetti presenti nell'elenco di visualizzazione, sia quelli creati in ActionScript che quelli realizzati con lo strumento di creazione di Flash. Per informazioni dettagliate, vedere "Lettura dell'elenco di visualizzazione" a pagina 413.

Oggetti di visualizzazione fuori elenco

In ActionScript 3.0, è possibile creare oggetti non visibili nell'elenco di visualizzazione. Tali oggetti sono conosciuti come oggetti di visualizzazione *fuori elenco*. Un oggetto di visualizzazione viene inserito nell'elenco degli oggetti di visualizzazione visibili solo se si richiama il metodo `addChild()` o `addChildAt()` di un'istanza di DisplayObjectContainer già inserita nell'elenco di visualizzazione.

Gli oggetti di visualizzazione fuori elenco possono essere utilizzati per assemblare oggetti di visualizzazione complessi, quali oggetti che presentano contenitori di oggetti di visualizzazione multipli contenenti più oggetti di visualizzazione. Mantenendo oggetti di visualizzazione fuori dall'elenco, si possono assemblare oggetti complicati senza che siano necessari i tempi di elaborazione normalmente richiesti per eseguire il rendering di tali oggetti. Gli oggetti di visualizzazione fuori elenco possono essere inseriti nell'elenco di visualizzazione quando necessario. Inoltre, è possibile spostare un elemento secondario di un contenitore di oggetti di visualizzazione fuori e dentro l'elenco di visualizzazione e in qualsiasi posizione dell'elenco in base alle necessità.

Creazione facilitata di sottoclassi degli oggetti di visualizzazione

In ActionScript 1.0 e 2.0, era spesso necessario inserire nuovi oggetti MovieClip in un file SWF per creare forme di base o per visualizzare bitmap. In ActionScript 3.0, la classe DisplayObject include numerose sottoclassi incorporate, quali Shape e Bitmap. Poiché le classi in ActionScript 3.0 sono più specifiche in base ai tipi di oggetti, la creazione di sottoclassi di base delle classi incorporate risulta più semplice.

Ad esempio, per disegnare un cerchio in ActionScript 2.0, era possibile creare una classe CustomCircle che estendesse la classe MovieClip ogni volta che veniva creata un'istanza di un oggetto della classe personalizzata. Tuttavia, tale classe includeva anche una serie di proprietà e metodi della classe MovieClip (quale `totalFrames`) non strettamente applicabili alla classe stessa. In ActionScript 3.0, è invece possibile creare una classe CustomCircle che estende l'oggetto Shape e che non include le proprietà e i metodi ad essa non pertinenti contenuti nella classe MovieClip. Nel codice riportato di seguito è illustrato un esempio di classe CustomCircle:

```
import flash.display.*;

private class CustomCircle extends Shape
{
  var xPos:Number;
  var yPos:Number;
  var radius:Number;
  var color:uint;
  public function CustomCircle(xInput:Number,
                      yInput:Number,
                      rInput:Number,
                      colorInput:uint)
  {
    xPos = xInput;
    yPos = yInput;
    radius = rInput;
    color = colorInput;
    this.graphics.beginFill(color);
    this.graphics.drawCircle(xPos, yPos, radius);
  }
}
```

Operazioni con gli oggetti di visualizzazione

Una volta acquisite nozioni di base relative a stage, oggetti di visualizzazione, contenitore degli oggetti di visualizzazione ed elenco di visualizzazione, è possibile passare alle sezioni seguenti, contenenti informazioni più specifiche sull'uso degli oggetti di visualizzazione in ActionScript 3.0.

Proprietà e metodi della classe DisplayObject

Tutti gli oggetti di visualizzazione sono sottoclassi della classe DisplayObject e, come tali, essi ereditano le proprietà e i metodi della classe DisplayObject. Le proprietà ereditate sono proprietà di base applicabili a tutti gli oggetti di visualizzazione. Ad esempio, ogni oggetto di visualizzazione presenta una proprietà `x` e una proprietà `y` che specificano la posizione dell'oggetto nel relativo contenitore degli oggetti di visualizzazione.

Non è possibile creare un'istanza di DisplayObject utilizzando la funzione di costruzione della classe DisplayObject. È invece necessario creare un altro tipo di oggetto (che sia una sottoclasse di DisplayObject), quale un oggetto Sprite, per creare un'istanza di un oggetto con l'operatore `new`. Inoltre, per creare una classe di oggetti di visualizzazione personalizzata, è necessario creare una sottoclasse di una delle sottoclassi degli oggetti di visualizzazione che presenta una funzione di costruzione utilizzabile (quale la classe Shape o la classe Sprite). Per ulteriori informazioni, vedere la descrizione della classe DisplayObject in *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Aggiunta di oggetti di visualizzazione all'elenco di visualizzazione

Quando si crea un'istanza di un oggetto di visualizzazione, essa non viene visualizzata sullo schermo (sullo stage) fino a quando l'istanza non viene inserita in un contenitore di oggetti di visualizzazione presente nell'elenco di visualizzazione. Ad esempio, nel codice seguente, l'oggetto TextField `myText` non risulta visibile se si omette l'ultima riga di codice. Nell'ultima riga di codice, la parola chiave `this` deve fare riferimento a un contenitore di oggetti di visualizzazione già inserito nell'elenco di visualizzazione.

```
import flash.display.*;
import flash.text.TextField;
var myText:TextField = new TextField();
myText.text = "Buenos dias.";
this.addChild(myText);
```

Quando si inserisce un qualsiasi elemento visivo sullo stage, tale elemento diventa un elemento *secondario* dell'oggetto Stage. Il primo file SWF caricato in un'applicazione (ad esempio, quello che viene incorporato in una pagina HTML) viene automaticamente aggiunto come elemento secondario dello stage. Può trattarsi di un oggetto di qualunque tipo che estende la classe Sprite.

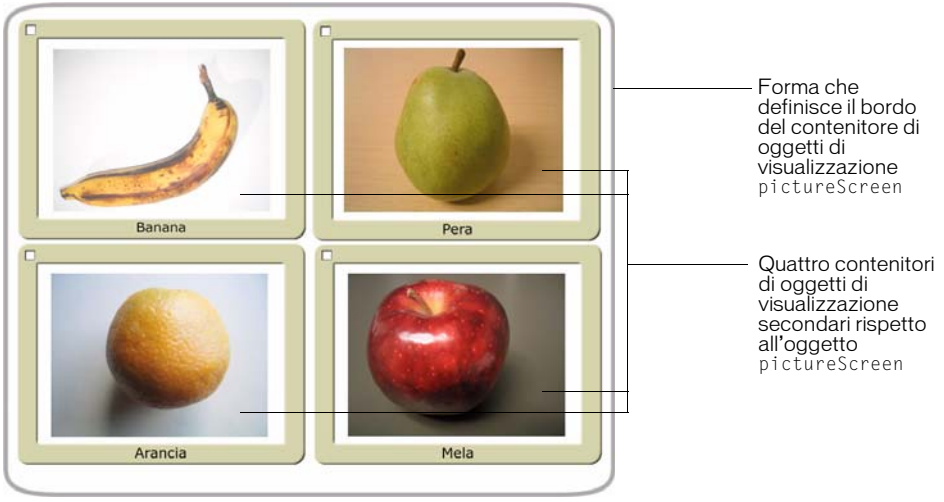
Tutti gli elementi di visualizzazione creati *senza* usare ActionScript (ad esempio, mediante l'aggiunta di un tag MXML in Adobe Flex Builder 2 o l'inserimento di un elemento sullo stage in Flash) vengono inseriti nell'elenco di visualizzazione. Anche se questi oggetti di visualizzazione non vengono inseriti mediante ActionScript, è possibile accedervi da ActionScript. Ad esempio, il codice seguente consente di regolare la larghezza di un oggetto denominato `button1` inserito mediante lo strumento di creazione (non mediante ActionScript):

```
button1.width = 200;
```

Uso dei contenitori degli oggetti di visualizzazione

Se un oggetto DisplayObjectContainer viene rimosso dall'elenco di visualizzazione, o se viene spostato o trasformato in qualunque altro modo, il relativo oggetto di visualizzazione presente in DisplayObjectContainer viene rimosso, spostato o trasformato.

Un contenitore di oggetti di visualizzazione è a sua volta un tipo di oggetto di visualizzazione e può essere inserito in un altro contenitore di oggetti di visualizzazione. Ad esempio, l'immagine seguente mostra un contenitore di oggetti di visualizzazione, `pictureScreen`, contenente una forma di contorno e quattro altri contenitori di oggetti di visualizzazione (di tipo PictureFrame):



Per fare in modo che un oggetto di visualizzazione venga riportato in un elenco di visualizzazione, è necessario inserirlo in un contenitore di oggetti di visualizzazione già presente nell'elenco di visualizzazione. Per fare ciò è necessario utilizzare il metodo `addChild()` o `addChildAt()` dell'oggetto contenitore. Ad esempio, senza la riga finale del codice seguente, l'oggetto `myTextField` non verrebbe visualizzato:

```
var myTextField:TextField = new TextField();
myTextField.text = "hello";
this.root.addChild(myTextField);
```

In questo codice di esempio, `this.root` punta al contenitore di oggetti di visualizzazione MovieClip che contiene il codice. Nel codice effettivo è necessario specificare un contenitore differente.

Utilizzare il metodo `addChildAt()` per inserire l'elemento secondario in una posizione specifica dell'elenco secondario del contenitore di oggetti di visualizzazione. Tali posizioni di indice a base zero dell'elenco secondario si riferiscono al livello (ordine di profondità) degli oggetti di visualizzazione. Ad esempio, si prendano in esame i seguenti tre oggetti di visualizzazione. Ogni oggetto è stato creato a partire da una classe personalizzata denominata Ball.

L'ordinamento su livelli di questi oggetti di visualizzazione all'interno del contenitore può essere modificato mediante il metodo `addChildAt()`. Esaminare, ad esempio, il codice seguente:

```
ball_A = new Ball(0xFFCC00, "a");
ball_A.name = "ball_A";
ball_A.x = 20;
ball_A.y = 20;
container.addChild(ball_A);

ball_B = new Ball(0xFFCC00, "b");
ball_B.name = "ball_B";
ball_B.x = 70;
ball_B.y = 20;
container.addChild(ball_B);

ball_C = new Ball(0xFFCC00, "c");
ball_C.name = "ball_C";
ball_C.x = 40;
ball_C.y = 60;
container.addChildAt(ball_C, 1);
```

Una volta eseguito il codice, gli oggetti di visualizzazione vengono disposti come segue nell'oggetto DisplayObjectContainer del `contenitore`. Si osservi il posizionamento degli oggetti.

Per riposizionare un oggetto all'inizio dell'elenco di visualizzazione, è sufficiente reinserirlo nell'elenco. Ad esempio, dopo l'esecuzione del codice precedente, per spostare `ball_A` sopra gli altri cerchi, eseguire questa riga di codice:

```
container.addChild(ball_A);
```

Questo codice non fa altro che rimuovere `ball_A` dalla posizione in cui si trova nell'elenco di visualizzazione del `contenitore` per reinserirlo in cima all'elenco, con il risultato di posizionare il cerchio sopra gli altri due.

È possibile utilizzare il metodo `getChildAt()` per verificare l'ordine dei livelli degli oggetti di visualizzazione. Il metodo `getChildAt()` restituisce oggetti secondari di un contenitore in base al numero di indice che si passa. Ad esempio, il codice seguente rivela nomi di oggetti di visualizzazione che si trovano in varie posizioni dell'elenco secondario dell'oggetto DisplayObjectContainer di `container`:

```
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_C
trace(container.getChildAt(2).name); // ball_B
```

Se si rimuove un oggetto di visualizzazione dall'elenco secondario del contenitore principale, gli elementi di rango superiore in elenco scalano di una posizione nell'elenco secondario. Ad esempio, continuando con il codice precedente, il codice che segue mostra come l'oggetto di visualizzazione che si trovava nella posizione 2 nell'elenco DisplayObjectContainer del `contenitore` passa alla posizione 1 se un oggetto di visualizzazione più in basso nell'elenco viene rimosso:

```
container.removeChild(ball_C);
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_B
```

I metodi `removeChild()` e `removeChildAt()` non consentono di eliminare completamente un'istanza di un oggetto di visualizzazione. Essi consentono solo di rimuoverla dall'elenco secondario del contenitore. L'istanza può ancora essere utilizzata come riferimento da un'altra variabile. (Per rimuovere completamente un oggetto, utilizzare l'operatore `delete`.)

Poiché un oggetto di visualizzazione presenta un solo contenitore principale, è possibile aggiungere un'istanza di un oggetto di visualizzazione a un solo contenitore di oggetti di visualizzazione. Ad esempio, il codice seguente mostra come l'oggetto di visualizzazione `tf1` possa esistere unicamente in un contenitore (in questo caso, un oggetto Sprite che estende la classe DisplayObjectContainer):

```
tf1:TextField = new TextField();
tf2:TextField = new TextField();
tf1.name = "text 1";
tf2.name = "text 2";

container1:Sprite = new Sprite();
container2:Sprite = new Sprite();

container1.addChild(tf1);
container1.addChild(tf2);
container2.addChild(tf1);
```

```
trace(container1.numChildren); // 1
trace(container1.getChildAt(0).name); // testo 2
trace(container2.numChildren); // 1
trace(container2.getChildAt(0).name); // testo 1
```

Se un oggetto di visualizzazione contenuto in un contenitore di oggetti di visualizzazione viene inserito in un altro contenitore di oggetti di visualizzazione, l'oggetto viene rimosso dal primo elenco secondario del contenitore di oggetti di visualizzazione.

Oltre ai metodi descritti sopra, la classe DisplayObjectContainer definisce vari altri metodi per utilizzare gli oggetti di visualizzazione secondari, quali:

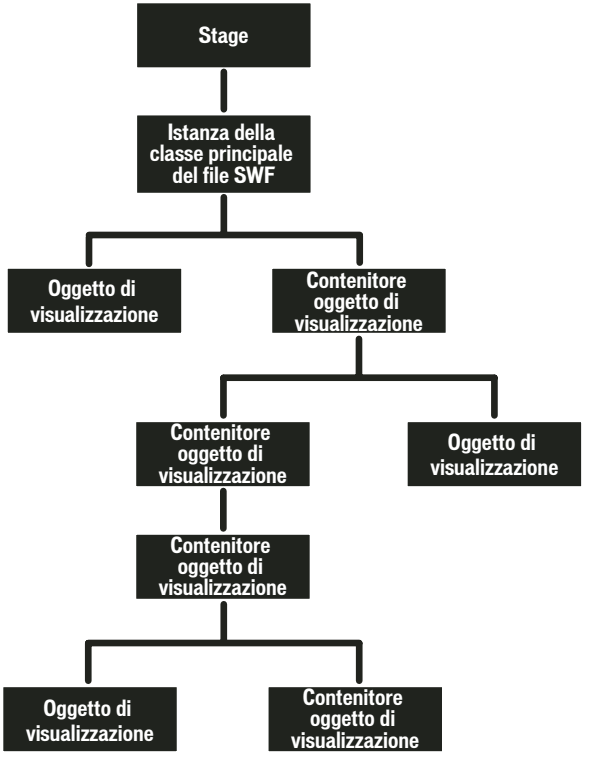
- `contains()`: consente di determinare se un oggetto di visualizzazione è un elemento secondario di DisplayObjectContainer.
- `getChildByName()`: consente di recuperare un oggetto in base al nome.
- `getChildIndex()`: restituisce la posizione di indice di un oggetto di visualizzazione.
- `setChildIndex()`: consente di modificare la posizione di un oggetto di visualizzazione secondario.
- `swapChildren()`: consente di invertire l'ordine di profondità di due oggetti di visualizzazione.
- `swapChildrenAt()`: consente di invertire l'ordine di profondità di due oggetti di visualizzazione, specificati in base al rispettivo valore di indice.

Per ulteriori informazioni, vedere le relative voci nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Si ricordi che un oggetto di visualizzazione che non si trova nell'elenco di visualizzazione (non incluso in un contenitore di oggetti di visualizzazione secondario allo stage) è conosciuto come oggetto di visualizzazione *fuori elenco*.

Lettura dell'elenco di visualizzazione

Come si è potuto osservare, l'elenco di visualizzazione è una struttura ad albero. Nella parte superiore di tale struttura si trova lo stage, che può contenere vari oggetti di visualizzazione. Tali oggetti sono essi stessi contenitori di oggetti di visualizzazione e possono contenere altri oggetti di visualizzazione o contenitori di oggetti di visualizzazione.



La classe DisplayObjectContainer include proprietà e metodi per la lettura dell'elenco di visualizzazione mediante gli elenchi secondari dei contenitori degli oggetti di visualizzazione. Ad esempio, si consideri il codice seguente, che consente di aggiungere due oggetti di visualizzazione, `title` e `pict`, all'oggetto `container` (che corrisponde a una classe Sprite che estende la classe DisplayObjectContainer):

```
var container:Sprite = new Sprite();
var title:TextField = new TextField();
title.text = "Hello";
var pict:Loader = new Loader();
var url:URLRequest = new URLRequest("banana.jpg");
pict.load(url);
pict.name = "banana loader";
container.addChild(title);
container.addChild(pict);
```

Il metodo `getChildAt()` restituisce l'elemento secondario dell'elenco di visualizzazione in una posizione di indice specifica:

```
trace(container.getChildAt(0) is TextField); // true
```

È possibile accedere agli oggetti secondari anche in base al nome. Ogni oggetto di visualizzazione presenta un nome proprietà; se tale nome non viene assegnato, Flash Player assegna un valore predefinito, quale `"instance1"`. Ad esempio, il codice seguente mostra come utilizzare il metodo `getChildByName()` per accedere a un oggetto di visualizzazione secondaria denominato `"banana loader"`:

```
trace(container.getChildByName("banana loader") is Loader); // true
```

L'uso del metodo `getChildByName()` può rallentare le prestazioni rispetto al metodo `getChildAt()`.

Poiché un contenitore di oggetti di visualizzazione può contenere altri contenitori di oggetti di visualizzazione come oggetti secondari nel proprio elenco di visualizzazione, è possibile leggere l'intero elenco di visualizzazione dell'applicazione come se fosse una struttura ad albero. Ad esempio, nell'estratto di codice di cui sopra, una volta completata l'operazione di caricamento dell'oggetto Loader `pict`, l'oggetto `pict` presenterà un oggetto di visualizzazione secondario (la bitmap) caricato. Per accedere all'oggetto di visualizzazione bitmap, è possibile scrivere `pict.getChildAt(0)`. È possibile scrivere anche `container.getChildAt(0).getChildAt(0)` (poiché `container.getChildAt(0) == pict`).

La funzione seguente fornisce un output `trace()` rientrato dell'elenco di visualizzazione da un contenitore di oggetti di visualizzazione:

```
function traceDisplayList(container:DisplayObjectContainer,
                          indentString:String = ""):void
{
  var child:DisplayObject;
  for (var i:uint=0; i < container.numChildren; i++)
  {
    child = container.getChildAt(i);
    trace(indentString, child, child.name);
    if (container.getChildAt(i) is DisplayObjectContainer)
    {
      traceDisplayList(DisplayObjectContainer(child), indentString + "")
    }
  }
}
```

Impostazione delle proprietà dello stage

La classe Stage sostituisce la maggior parte dei metodi e delle proprietà della classe DisplayObject. Se si chiama uno di questi metodi o proprietà sostituiti, Flash Player genera un'eccezione. Ad esempio, l'oggetto Stage non presenta proprietà `x` o `y`, in quanto la sua posizione è fissa, quale contenitore principale dell'applicazione. Le proprietà `x` e `y` si riferiscono alla posizione di un oggetto di visualizzazione in relazione al suo contenitore e, poiché lo stage non è contenuto in alcun contenitore, tali proprietà non sono applicabili.

NOTA

Alcune proprietà e metodi della classe Stage non sono disponibili per oggetti di visualizzazione che non presentano la stessa funzione di sicurezza sandbox del primo file SWF. Per informazioni dettagliate, vedere "Stage, sicurezza" a pagina 834.

Controllo della frequenza fotogrammi di riproduzione

La proprietà `framerate` della classe Stage viene utilizzata per impostare la frequenza fotogrammi di tutti i file SWF caricati nell'applicazione. Per ulteriori informazioni, consultare la *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0* e

Controllo della modifica in scala dello stage

Quando si ridimensiona una finestra di Flash Player, Flash Player regola automaticamente il contenuto dello stage per compensare la discrepanza. La proprietà `scaleMode` della classe Stage determina come il contenuto dello stage viene regolato. Questa proprietà può essere impostata su quattro diversi valori, definiti come costanti nella classe flash.display.StageScaleMode.

Per tre dei valori `scaleMode` (`StageScaleMode.EXACT_FIT`, `StageScaleMode.SHOW_ALL` e `StageScaleMode.NO_BORDER`), Flash Player modifica in scala il contenuto dello stage per adeguarlo ai limiti dell'area disponibile. Le tre opzioni differiscono nella modalità di esecuzione della modifica in scala:

- `StageScaleMode.EXACT_FIT` modifica in scala il file SWF in modo proporzionale.
- `StageScaleMode.SHOW_ALL` determina la visualizzazione del bordo, come le barre nere che appaiono sullo schermo di un televisore standard quando si guarda un film per schermi grandi.
- `StageScaleMode.NO_BORDER` determina se il contenuto può essere parzialmente ritagliato.

In alternativa, se `scaleMode` è impostata su `StageScaleMode.NO_SCALE`, quando la finestra di Flash Player viene ridimensionata il contenuto dello stage conserva le dimensioni definite. Esclusivamente in questa modalità scala, le proprietà `width` e `height` della classe Stage possono essere utilizzate per determinare le dimensioni in pixel effettive della finestra ridimensionata di Flash Player. (Nelle altre modalità, le proprietà `stageWidth` e `stageHeight` riflettono sempre la larghezza e l'altezza originali del file SWF.) Inoltre, se `scaleMode` è impostata su `StageScaleMode.NO_SCALE` e il file SWF viene ridimensionato, l'evento `resize` della classe Stage viene inviato, in modo che sia possibile effettuare le regolazioni appropriate.

Di conseguenza, il valore `StageScaleMode.NO_SCALE` garantisce un maggiore controllo sulla regolazione del contenuto dello schermo in base al ridimensionamento della finestra. Ad esempio, in un file SWF contenente un video e una barra di controllo, è possibile fare in modo che le dimensioni della barra di controllo rimangano inalterate quando si ridimensiona lo stage e modificare solo le dimensioni della finestra del video per adattarle a quelle dello stage. Questo comportamento viene illustrato nell'esempio seguente:

```
// videoScreen è un oggetto di visualizzazione (ad es. un'istanza video)
// che contiene un video; è posizionato nell'angolo superiore sinistro
// dello stage, e deve essere ridimensionato quando il file SWF viene
// ridimensionato.

// controlBar è un oggetto di visualizzazione (ad es. uno sprite) che
// contiene molti pulsanti; deve rimanere posizionato nell'angolo inferiore
// sinistro dello stage (sotto videoScreen) e non deve essere ridimensionato
// quando il file SWF viene ridimensionato.
```

```
import flash.display.Stage;
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.events.Event;

var swfStage:Stage = videoScreen.stage;
swfStage.scaleMode = StageScaleMode.NO_SCALE;
swfStage.align = StageAlign.TOP_LEFT;

function resizeDisplay(event:Event):void
{
  var swfWidth:int = swfStage.stageWidth;
  var swfHeight:int = swfStage.stageHeight;

  // Resize the video window.
  var newVideoHeight:Number = swfHeight - controlBar.height;
  videoScreen.height = newVideoHeight;
  videoScreen.scaleX = videoScreen.scaleY;

  // Reposition the control bar.
  controlBar.y = newVideoHeight;
}

swfStage.addEventListener(Event.RESIZE, resizeDisplay);
```

Uso della modalità a schermo intero

La modalità a schermo intero consente di fare occupare a un file SWF l'intero monitor, senza bordi, barre dei menu, ecc. La proprietà `displayState` della classe Stage consente di attivare e disattivare la modalità a schermo intero per un file SWF. La proprietà `displayState` può essere impostata su uno dei valori definiti dalle costanti nella classe flash.display.StageDisplayState. Per attivare la modalità a schermo intero, impostare `displayState` su `StageDisplayState.FULL_SCREEN`:

```
// mySprite è un'istanza di Sprite, già inserita nell'elenco di
// visualizzazione
mySprite.stage.displayState = StageDisplayState.FULL_SCREEN;
```

Per disattivare la modalità a schermo intero, impostare la proprietà `displayState` su `StageDisplayState.NORMAL`:

```
mySprite.stage.displayState = StageDisplayState.NORMAL;
```

Inoltre, un utente può scegliere di lasciare attiva la modalità a schermo intero e passare a un'altra finestra di visualizzazione oppure utilizzare una delle varie combinazioni di tasti disponibili: tasto Esc (tutte le piattaforme), Ctrl-W (Windows), Command-W (Mac) o Alt-F4 (Windows).

Il comportamento di ridimensionamento dello stage per la modalità a schermo intero è identico nella modalità normale; il ridimensionamento viene controllato dalla proprietà `scaleMode` della classe Stage. Come sempre, se la proprietà `scaleMode` è impostata su `State.NO_SCALE`, le proprietà `stageWidth` e `stageHeight` dello stage vengono modificate per riflettere le dimensioni dell'area dello schermo occupate dal file SWF (in questo caso, l'intero schermo).

Utilizzare l'evento `fullScreen` della classe Stage per rilevare se la modalità a schermo intero è attivata o disattivata e rispondere di conseguenza. Ad esempio, potrebbe essere necessario riposizionare, aggiungere o rimuovere elementi dallo schermo quando si attiva o disattiva la modalità a schermo intero, come nell'esempio seguente:

```
import flash.events.FullScreenEvent;

function fullScreenRedraw(event:FullScreenEvent):void
{
  if (event.fullScreen)
  {
    // Rimuove i campi di testo di input.
    // Aggiunge un pulsante che chiude la modalità a schermo intero.
  }
  else
  {
    // Riaggiunge i campi di testo di input.
    // Rimuove il pulsante che chiude la modalità a schermo intero.
  }
}

mySprite.stage.addEventListener(FullScreenEvent.FULL_SCREEN,
  fullScreenRedraw);
```

Come illustrato dal codice, l'oggetto evento dell'evento `fullScreen` è un'istanza della classe flash.events.FullScreenEvent, che include una proprietà `fullScreen` che indica se la modalità a schermo intero è attiva (`true`) o no (`false`).

Quando si lavora in modalità a schermo intero in ActionScript, tenere presente le seguenti considerazioni:

- La modalità a schermo intero può essere avviata mediante ActionScript in risposta a un clic del mouse (anche con il pulsante destro) o alla pressione di un tasto.
- Per gli utenti con più monitor, il contenuto SWF si espanderà fino a riempire un solo monitor. Flash Player impiega una metrica per determinare quale monitor contiene la porzione più grande di SWF, quindi usa tale monitor per la modalità a schermo intero.

- Per un file SWF incorporato in una pagina HTML, il codice HTML necessario per incorporare Flash Player deve includere un tag `param` e un attributo `embed` con nome `allowFullScreen` e valore `true`, come indicato di seguito:

```
<object>
  ...
  <param name="allowFullScreen" value="true" />
  <embed ... allowfullscreen="true" />
</object>
```

  Se si usa JavaScript in una pagina web per generare i tag di incorporamento SWF, è necessario alterare JavaScript per aggiungere l'attributo/tag `allowFullScreen param`. Ad esempio, se la pagina HTML impiega la funzione `AC_FL_RunContent()` (utilizzata dalle pagine HTML generate sia da Flex Builder che da Flash), è necessario aggiungere il parametro `allowFullScreen` alla chiamata di funzione, come indicato di seguito:

```
AC_FL_RunContent(
  ...
  'allowFullScreen','true',
  ...
  ); //end AC code
```

  Ciò non è applicabile a file SWF eseguiti nel lettore autonomo Flash Player.

- Tutte le operazioni di tastiera di ActionScript, quali eventi di tastiera e immissione testo nelle istanze TextFields, vengono disattivate in modalità a schermo intero. La sola eccezione sono le scelte rapide da tastiera che consentono di chiudere la modalità a schermo intero.

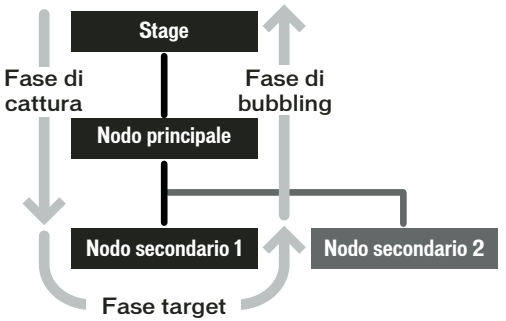
Vi sono inoltre alcune limitazioni di sicurezza da tenere presente. Tali limitazioni sono descritte alla sezione "Funzioni di sicurezza sandbox" a pagina 822.

Gestione degli eventi per gli oggetti di visualizzazione

La classe DisplayObject eredita dalla classe EventDispatcher. Ciò significa che ogni oggetto di visualizzazione può essere incluso completamente nel modello evento (descritto in Capitolo 10, "Gestione degli eventi" a pagina 335). Tutti gli oggetti di visualizzazione possono utilizzare il proprio metodo `addEventListener()`, ereditato dalla classe EventDispatcher, per intercettare particolari eventi, ma solo se l'oggetto che intercetta fa parte del flusso di eventi di tale evento.

Quando Flash Player invia un oggetto evento, tale oggetto esegue un percorso di andata e ritorno dallo stage all'oggetto di visualizzazione in cui si è verificato l'evento. Ad esempio, se un utente fa clic su un oggetto di visualizzazione denominato `child1`, Flash Player invia un oggetto evento che parte dallo stage, attraversa la gerarchia dell'elenco di visualizzazione fino a raggiungere l'oggetto di visualizzazione `child1`.

Il flusso di eventi è concettualmente suddiviso in tre fasi, come illustrato nel diagramma seguente:



Per ulteriori informazioni, vedere Capitolo 10, "Gestione degli eventi" a pagina 335.

Un dato importante da tenere presente quando si lavora con gli eventi degli oggetti di visualizzazione è l'effetto sui listener di eventi in relazione alla rimozione automatica degli oggetti di visualizzazione dalla memoria (garbage collection) quando tali oggetti vengono rimossi dall'elenco di visualizzazione. Se un oggetto di visualizzazione presenta oggetti registrati come listener ai propri eventi, tale oggetto di visualizzazione non verrà rimosso dalla memoria, anche se viene rimosso dall'elenco di visualizzazione, in quanto presenta riferimenti a tali oggetti listener. Per ulteriori informazioni, vedere "Gestione dei listener di eventi" a pagina 354.

Scelta di una sottoclasse DisplayObject

Con così tante opzioni a disposizione, una delle decisioni più importanti da prendere quando si lavora con gli oggetti di visualizzazione è quale oggetto utilizzare e per quale scopo. In questa sezione sono riportate alcune linee guida che dovrebbero aiutare l'utente nelle sue decisioni. Gli stessi suggerimenti sono applicabili sia che si debba scegliere un'istanza di una classe o una classe base per una classe da creare:

- Se non è necessario un oggetto che funga da contenitore per altri oggetti di visualizzazione (vale a dire, se è necessario un semplice elemento visivo autonomo), selezionare una delle seguenti sottoclassi di DisplayObject o InteractiveObject, in base all'uso che se ne deve fare:
  - Bitmap per la visualizzazione di un'immagine bitmap.
  - TextField per l'aggiunta di testo.
  - Video per la visualizzazione di video.
  - Shape per la visualizzazione di un'area di lavoro sullo schermo su cui disegnare contenuto. In particolare, se si desidera creare un'istanza per il disegno di forme sullo schermo, senza che sia necessario creare un contenitore per altri oggetti di visualizzazione, è possibile guadagnare in termini di prestazioni utilizzando la sottoclasse Shape al posto di Sprite o di MovieClip.
  - MorphShape, StaticText o SimpleButton per elementi specifici dello strumento di creazione di Flash. (Non è possibile creare istanze di queste classi in maniera programmatica, tuttavia è possibile creare variabili con questi tipi di dati per fare riferimento a elementi creati mediante lo strumento di creazione di Flash.)
- Se è necessario creare una variabile per fare riferimento allo stage principale, utilizzare la classe Stage come tipo di dati.
- Se è necessario un contenitore per il caricamento di un file di immagine o SWF esterno, utilizzare un'istanza Loader. Il contenuto caricato verrà inserito nell'elenco di visualizzazione come elemento secondario dell'istanza Loader. Il tipo di dati utilizzato dipenderà dalla natura del contenuto caricato, come indicato di seguito:
  - Un'immagine caricata corrisponderà a un'istanza Bitmap.
  - Un file SWF caricato scritto in ActionScript 3.0 corrisponderà a un'istanza Sprite o MovieClip (o a un'istanza di una sottoclasse di tali classi, come specificato dal creatore di contenuto).
  - Un file SWF caricato scritto in ActionScript 1.0 o in ActionScript 2.0 corrisponderà a un'istanza AVM1Movie.

- Se è necessario un oggetto che funga da contenitore per altri oggetti di visualizzazione (indipendentemente dal fatto che si disegni o meno sull'oggetto di visualizzazione mediante ActionScript), selezionare una delle sottoclassi DisplayObjectContainer:
  - Sprite se l'oggetto viene creato utilizzando solo ActionScript oppure come classe base per un oggetto di visualizzazione personalizzato da creare e manipolare unicamente mediante ActionScript.
  - MovieClip se si deve creare una variabile che faccia riferimento a un simbolo di clip filmato creato con lo strumento di creazione di Flash.
- Se si deve creare una classe da associare a un simbolo di clip filmato nella libreria Flash, selezionare una delle seguenti sottoclassi DisplayObjectContainer come classe base della classe:
  - MovieClip se il simbolo di clip filmato associato presenta contenuto su più di un fotogramma.
  - Sprite se il simbolo di clip filmato associato presenta contenuto solo sul primo fotogramma.

Manipolazione di oggetti di visualizzazione

Indipendentemente dal tipo di oggetto di visualizzazione che si decide di utilizzare, vi sono una serie di manipolazioni comuni a tutti gli oggetti quali elementi visualizzati sullo schermo. Ad esempio, tutti gli oggetti di visualizzazione possono essere collocati sullo schermo, spostati avanti o indietro nell'ordine di impilamento, ridimensionati, ruotati e così via. Poiché tutti gli oggetti di visualizzazione ereditano queste funzionalità dalla loro classe base comune (DisplayObject), tali funzionalità si comportano allo stesso modo sia che si stia manipolando un'istanza di TextField, Video, Shape o di qualsiasi altro oggetto. Nelle sezioni che seguono sono illustrate alcune di queste manipolazioni comuni a tutti gli oggetti di visualizzazione.

Modifica della posizione

Il tipo di manipolazione più banale per un oggetto di visualizzazione è il suo posizionamento sullo schermo. Per impostare la posizione di un oggetto di visualizzazione, modificare le proprietà `x` e `y` dell'oggetto.

```
myShape.x = 17;
myShape.y = 212;
```

Il sistema di posizionamento degli oggetti di visualizzazione tratta lo stage come un sistema di coordinate cartesiane (il comune sistema a griglia con un asse x orizzontale e un asse y verticale). L'origine del sistema di coordinate (la coordinata 0, 0, dove gli assi x e y si intersecano) si trova nell'angolo superiore sinistro dello stage. A partire da questo punto, i valori x sono positivi spostandosi verso destra e negativi verso sinistra, mentre (al contrario di ciò che avviene nei grafici tradizionali) i valori y sono positivi spostandosi verso il basso e negativi verso l'alto. Ad esempio, le righe di codice precedente consentono di spostare l'oggetto `myShape` alla coordinata x 17 (17 pixel a destra dell'origine) e alla coordinata y 212 (212 pixel al di sotto dell'origine).

Per impostazione predefinita, quando un oggetto di visualizzazione viene creato in ActionScript, le proprietà `x` e `y` sono impostate su 0, in modo da collocare l'oggetto nell'angolo superiore sinistro del suo contenitore principale.

Modifica della posizione in relazione allo stage

È importante ricordare che le proprietà `x` e `y` si riferiscono sempre alla posizione dell'oggetto di visualizzazione in relazione alla coordinate 0, 0 degli assi del suo oggetto di visualizzazione principale. Di conseguenza, nel caso di un'istanza Shape (quale un cerchio) contenuta in un'istanza Sprite, l'impostazione delle proprietà `x` e `y` dell'oggetto Shape su 0 consentirà di posizionare il cerchio nell'angolo superiore sinistro di Sprite, che non è necessariamente nell'angolo superiore sinistro dello stage. Per posizionare un oggetto in relazione alle coordinate globali dello stage, è possibile utilizzare il metodo `globalToLocal()` di un qualsiasi oggetto di visualizzazione e convertire le coordinate da globali (stage) a locali (contenitore dell'oggetto di visualizzazione), come indicato di seguito:

```
// Posiziona la forma nell'angolo superiore sinistro dello stage,
// indipendentemente dalla posizione del suo elemento principale.

// Crea un'istanza di Sprite, posizionata a x:200 e y:200.
var mySprite:Sprite = new Sprite();
mySprite.x = 200;
mySprite.y = 200;
this.addChild(mySprite);

// Disegna un punto alle coordinate 0, 0 di Sprite, come riferimento.
mySprite.graphics.lineStyle(1, 0x000000);
mySprite.graphics.beginFill(0x000000);
mySprite.graphics.moveTo(0, 0);
mySprite.graphics.lineTo(1, 0);
mySprite.graphics.lineTo(1, 1);
mySprite.graphics.lineTo(0, 1);
mySprite.graphics.endFill();
```

```
// Crea un'istanza Shape corrispondente a un cerchio.
var circle:Shape = new Shape();
mySprite.addChild(circle);

// Disegna un cerchio con raggio 50 e punto centrale a x:50, y:50
// all'interno di Shape.
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0xff0000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();

// Sposta Shape in modo che il suo angolo superiore sinistro corrisponda
// alle coordinate 0, 0 di stage.
var stagePoint:Point = new Point(0, 0);
var targetPoint:Point = mySprite.globalToLocal(stagePoint);
circle.x = targetPoint.x;
circle.y = targetPoint.y;
```

Allo stesso modo, è possibile utilizzare il metodo `localToGlobal()` della classe DisplayObject per convertire le coordinate locali nelle coordinate dello stage.

Creazione di un'interazione di trascinamento della selezione

Un motivo comune per lo spostamento di un oggetto di visualizzazione è la creazione di un'interazione di trascinamento della selezione, per fare in modo che, quando un utente fa clic su un oggetto, l'oggetto si muova insieme al puntatore del mouse, fino a quando il pulsante del mouse non viene rilasciato. L'interazione di trascinamento può essere creata in due diversi modi in ActionScript. In entrambi i casi, vengono utilizzati due eventi del mouse: quando il pulsante del mouse viene premuto, l'oggetto deve seguire il cursore del mouse, mentre quando il pulsante viene rilasciato, l'oggetto deve bloccarsi nella posizione in cui si trova.

Il primo modo, che prevede l'uso del metodo `startDrag()`, è il più semplice, ma presenta dei limiti. Quando il pulsante del mouse viene premuto, il metodo `startDrag()` dell'oggetto di visualizzazione da trascinare viene chiamato. Quando il pulsante del mouse viene rilasciato, viene chiamato il metodo `stopDrag()`.

```
// Questo codice consente di creare un'interazione di trascinamento della
// selezione mediante startDrag(). square è un oggetto DisplayObject
// (ad es. un'istanza di MovieClip o di Sprite).

import flash.events.MouseEvent;

// Questa funzione viene chiamata quando il pulsante del mouse viene
// premuto.
function startDragging(event:MouseEvent):void
{
  square.startDrag();
}
```

```
// Questa funzione viene chiamata quando il pulsante del mouse viene
// rilasciato.
function stopDragging(event:MouseEvent):void
{
  square.stopDrag();
}

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Questa tecnica presenta tuttavia un limite significativo: il metodo `startDrag()` consente di trascinare un solo elemento per volta. Se un oggetto di visualizzazione viene trascinato e il metodo `startDrag()` viene chiamato su un altro oggetto di visualizzazione, il primo oggetto cessa immediatamente di seguire il mouse. Ad esempio, se la funzione `startDragging()` viene modificata come illustrato di seguito, solo l'oggetto `circle` verrà trascinato, nonostante la chiamata del metodo `square.startDrag()`:

```
function startDragging(event:MouseEvent):void
{
  square.startDrag();
  circle.startDrag();
}
```

Come conseguenza del fatto che un solo oggetto per volta può essere trascinato con il metodo `startDrag()`, il metodo `stopDrag()` può essere chiamato per bloccare qualsiasi oggetto di visualizzazione in fase di trascinamento.

Se è necessario trascinare più di un oggetto di visualizzazione o per evitare conflitti quando più di un oggetto può potenzialmente usare il metodo `startDrag()`, si consiglia di utilizzare la tecnica di inseguimento del mouse per ottenere l'effetto di trascinamento. Con questa tecnica, quando il pulsante del mouse viene premuto, una funzione viene registrata come listener dell'evento `mouseMove` sullo stage. Tale funzione, che in seguito verrà chiamata ogni volta che il mouse si muove, provoca lo spostamento dell'oggetto trascinato in corrispondenza delle coordinate x, y del mouse. Quando il pulsante del mouse viene rilasciato, la registrazione della funzione come listener viene annullata, in modo che la funzione non venga più chiamata a ogni movimento del mouse e l'oggetto cessi di seguire il cursore. Il codice seguente illustra tale tecnica:

```
// Questo codice consente di creare un'interazione di trascinamento della
// selezione mediante inseguimento del mouse. circle è un oggetto
// DisplayObject (ad es. un'istanza di MovieClip o di Sprite).

import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;
```

```
// Questa funzione viene chiamata quando il pulsante del mouse viene
// premuto.
function startDragging(event:MouseEvent):void
{
  // Registra la differenza (spostamento) tra
  // il punto in cui si trovava il cursore quando il pulsante del mouse
  // è stato premuto e le coordinate x, y di circle nel momento in cui
  // il pulsante del mouse è stato premuto.
  offsetX = event.stageX - circle.x;
  offsetY = event.stageY - circle.y;

  // Indica a Flash Player di iniziare a intercettare l'evento mouseMove
  stage.addEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// Questa funzione viene chiamata quando il pulsante del mouse viene
// rilasciato.
function stopDragging(event:MouseEvent):void
{
  // Indica a Flash Player di cessare di intercettare l'evento mouseMove.
  stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// Questa funzione viene chiamata ogni volta che il mouse viene spostato,
// fino a quando il pulsante del mouse resta premuto.
function dragCircle(event:MouseEvent):void
{
  // Sposta il cerchio nella posizione del cursore, mantenendo
  // lo scarto tra la posizione del cursore e
  // la posizione dell'oggetto trascinato.
  circle.x = event.stageX - offsetX;
  circle.y = event.stageY - offsetY;

  // Istruisce Flash Player di aggiornare lo schermo dopo questo evento.
  event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Oltre a fare in modo che un oggetto di visualizzazione segua il cursore del mouse, una parte comune della funzione di interazione di trascinamento riguarda il movimento dell'oggetto trascinato sul livello superiore dello schermo, in modo che sembri fluttuare sopra gli altri oggetti. Si supponga, ad esempio, di avere due oggetti, un cerchio e un quadrato, entrambi da spostare mediante interazione di trascinamento. Se il cerchio si trova al di sotto del quadrato nell'elenco di visualizzazione e si fa clic sul cerchio per trascinarlo facendolo passare sopra il quadrato, il cerchio verrà fatto scorrere sotto il quadrato, rompendo l'illusione di trascinamento. Per evitare il problema, è possibile fare in modo che quando si fa clic sul cerchio, esso venga automaticamente spostato in cima all'elenco di visualizzazione, in modo che, quando viene trascinato, appaia sempre al di sopra di qualsiasi altro oggetto presente sullo stage.

Il codice seguente (adattato dall'esempio precedente) crea un'interazione di trascinamento per due oggetti di visualizzazione, un cerchio e un quadrato. Ogni volta che il pulsante del mouse viene premuto su uno dei due elementi, esso viene spostato al primo posto nell'elenco di visualizzazione dello stage, in modo che risulti sempre in primo piano quando viene trascinato. Il codice nuovo o modificato rispetto all'esempio precedente è visualizzato in grassetto.

```
// Questo codice consente di creare un'interazione di trascinamento della
// selezione mediante inseguimento del mouse. cerchio e quadrato sono
// oggetti di DisplayObjects (ad es istanze di MovieClip
// o di Sprite).

import flash.display.DisplayObject;
import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;
var draggedObject:DisplayObject;

// Questa funzione viene chiamata quando il pulsante del mouse viene
// premuto.
function startDragging(event:MouseEvent):void
{
  // ricorda quale oggetto viene trascinato
  draggedObject = DisplayObject(event.target);

  // Registra la differenza (spostamento) tra dove si trovava il cursore
  // quando il pulsante del mouse è stato premuto e le coordinate x, y
  // dell'oggetto nel momento in cui il pulsante del mouse è stato premuto.
  offsetX = event.stageX - draggedObject.x;
  offsetY = event.stageY - draggedObject.y;
```

```
    // Sposta l'oggetto selezionato al primo posto nell'elenco di
    visualizzazione
    stage.addChild(draggedObject);

    // Indica a Flash Player di iniziare ad intercettare l'evento mouseMove.
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// Questa funzione viene chiamata quando il pulsante del mouse viene
// rilasciato.
function stopDragging(event:MouseEvent):void
{
    // Indica a Flash Player di cessare di intercettare l'evento mouseMove.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// Questa funzione viene chiamata ogni volta che il mouse viene spostato,
// fino a quando il pulsante del mouse resta premuto.
function dragObject(event:MouseEvent):void
{
    // Sposta l'oggetto trascinato nella posizione del cursore, mantenendo
    // lo scarto tra la posizione del cursore e la posizione
    // dell'oggetto trascinato.
    draggedObject.x = event.stageX - offsetX;
    draggedObject.y = event.stageY - offsetY;

    // Istruisce Flash Player di aggiornare lo schermo dopo questo evento.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Per estendere ulteriormente l'effetto, come nel caso di un gioco in cui gettoni o carte vengono spostati da una pila a un'altra, è possibile inserire l'oggetto trascinato nell'elenco di visualizzazione dello stage quando viene "raccolto", quindi inserirlo in un altro elenco di visualizzazione (ad es. quello della "pila" dove viene depositato) al momento del rilascio del pulsante del mouse.

Infine, per migliorare l'effetto, è possibile applicare un filtro ombra esterna all'oggetto di visualizzazione quando viene selezionato mediante clic del mouse (all'inizio del trascinamento) per poi rimuoverlo quando l'oggetto viene rilasciato. Per ulteriori informazioni sull'uso del filtro ombra esterna e di altri filtri per gli oggetti di visualizzazione in ActionScript, vedere Capitolo 15, "Filtraggio degli oggetti di visualizzazione" a pagina 501.

Panoramica e scorrimento di oggetti di visualizzazione

Se un oggetto di visualizzazione è troppo grande per l'area in cui deve essere visualizzato, è possibile utilizzare la proprietà `scrollRect` per definire l'area visualizzabile dell'oggetto. Inoltre, modificando la proprietà `scrollRect` in risposta a un input dell'utente, è possibile fare scorrere il contenuto visualizzato orizzontalmente e verticalmente.

La proprietà `scrollRect` è un'istanza della classe Rectangle, in grado di combinare i valori necessari per definire un'area rettangolare come singolo oggetto. Per definire inizialmente l'area visualizzabile dell'oggetto, creare una nuova istanza di Rectangle e assegnarla alla proprietà `scrollRect` dell'oggetto di visualizzazione. Quindi, per fare scorrere l'oggetto, è necessario leggere la proprietà `scrollRect` in una variabile Rectangle separata e modificare la proprietà desiderata (ad esempio, modificare la proprietà `x` dell'istanza di Rectangle per lo scorrimento orizzontale e la proprietà `y` per lo scorrimento verticale). Infine, è necessario riassegnare l'istanza di Rectangle alla proprietà `scrollRect` per notificare all'oggetto di visualizzazione l'avvenuta modifica del valore.

Ad esempio, il codice seguente definisce l'area visualizzabile di un oggetto TextField denominato `bigText`, troppo grande per rientrare nei limiti fisici del file SWF. Quando i due pulsanti denominati `up` e `down` vengono selezionati, essi richiamano funzioni che provocano lo scorrimento dell'oggetto TextField verso l'alto o verso il basso, mediante la modifica della proprietà `y` dell'istanza di Rectangle `scrollRect`.

```
import flash.events.MouseEvent;
import flash.geom.Rectangle;

// Definisce l'area iniziale visualizzabile dell'istanza TextField:
// sinistra: 0, alto: 0, larghezza: larghezza, altezza TextField: 350 pixel.
bigText.scrollRect = new Rectangle(0, 0, bigText.width, 350);

// Salva nella cache TextField come bitmap per migliorare le prestazioni.
bigText.cacheAsBitmap = true;

// richiamato quando viene selezionato il pulsante "up"
function scrollUp(event:MouseEvent):void
{
  // Accede a rettangolo di scorrimento corrente.
  var rect:Rectangle = bigText.scrollRect;
  // Riduce il valore y del rettangolo di 20, per
  // spostare il rettangolo di 20 pixel verso il basso.
  rect.y -= 20;
  // Riassegna il rettangolo a TextField per rendere effettiva la modifica.
  bigText.scrollRect = rect;
}
```

```
// richiamato quando viene selezionato il pulsante "down"
function scrollDown(event:MouseEvent):void
{
  // Accede a rettangolo di scorrimento corrente.
  var rect:Rectangle = bigText.scrollRect;
  // Aumenta il valore y del rettangolo di 20, per
  // spostare il rettangolo di 20 pixel verso l'alto.
  rect.y += 20;
  // Riassegna il rettangolo a TextField per rendere effettiva la modifica.
  bigText.scrollRect = rect;
}

up.addEventListener(MouseEvent.CLICK, scrollUp);
down.addEventListener(MouseEvent.CLICK, scrollDown);
```

Come dimostra l'esempio, quando si lavora con la proprietà `scrollRect` di un oggetto di visualizzazione, è consigliabile fare in modo che Flash Player memorizzi nella cache il contenuto dell'oggetto di visualizzazione come bitmap, utilizzando la proprietà `cacheAsBitmap`. In tal modo, Flash Player non dovrà ridisegnare l'intero contenuto dell'oggetto di visualizzazione ogni volta che viene fatto scorrere e potrà invece utilizzare la bitmap memorizzata nella cache per eseguire il rendering della porzione necessaria direttamente sullo schermo. Per informazioni dettagliate, vedere "Memorizzazione nella cache di oggetti di visualizzazione" a pagina 434.

Manipolazione delle dimensioni e modifica in scala degli oggetti

È possibile misurare e manipolare le dimensioni di un oggetto di visualizzazione in due diversi modi, mediante le proprietà delle dimensioni (`width` e `height`) o le proprietà delle modifiche in scala (`scaleX` e `scaleY`).

Ogni oggetto di visualizzazione presenta una proprietà `width` e una proprietà `height`, inizialmente impostate sulle dimensioni dell'oggetto in pixel. È possibile leggere i valori di tali proprietà per misurare le dimensioni dell'oggetto di visualizzazione. È inoltre possibile specificare nuovi valori e modificare le dimensioni dell'oggetto, come segue:

```
// Ridimensiona un oggetto di visualizzazione.
square.width = 420;
square.height = 420;

// Determina il raggio di un oggetto di visualizzazione cerchio.
var radius:Number = circle.width / 2;
```

La modifica di `altezza` o `larghezza` di un oggetto di visualizzazione provoca una variazione in scala dell'oggetto; in altre parole, il contenuto viene ristretto o allargato per rientrare nella nuova area definita. Se l'oggetto di visualizzazione contiene solo forme vettoriali, tali forme verranno ridisegnate in base alla nuova scala, senza alcun peggioramento della qualità. Tutti gli elementi delle immagini bitmap dell'oggetto di visualizzazione verranno modificati in scala, anziché ridisegnati. Di conseguenza, ad esempio, una foto digitale la cui larghezza e altezza vengono incrementate oltre le effettive dimensioni in pixel dell'immagine, verrà "pixelizzata" e i suoi contorni risulteranno irregolari.

Se le proprietà `width` e `height` di un oggetto di visualizzazione vengono modificate, Flash Player aggiorna automaticamente le proprietà `scaleX` e `scaleY` dell'oggetto. Tali proprietà rappresentano le dimensioni relative dell'oggetto di visualizzazione rispetto alle sue dimensioni originali. Le proprietà `scaleX` e `scaleY` impiegano cifre decimali per rappresentare valori percentuali. Ad esempio, se la proprietà `width` di un oggetto di visualizzazione viene ridotta della metà rispetto alla sua dimensione originale, la proprietà `scaleX` dell'oggetto avrà un valore pari a `.5`, a indicare il 50 percento. Se invece l'altezza viene raddoppiata, la proprietà `scaleY` avrà il valore `2`, a indicare il 200 percento.

```
// circle è un oggetto di visualizzazione con proprietà width e height pari
// a 150 pixel.
// Nelle dimensioni originali, scaleX e scaleY sono pari a 1 (100%).
trace(circle.scaleX);// output: 1
trace(circle.scaleY);// output: 1

// Se le proprietà width e height vengono modificate,
// Flash Player modifica le proprietà scaleX e scaleY di conseguenza.
circle.width = 100;
circle.height = 75;
trace(circle.scaleX);// output: 0.6622516556291391
trace(circle.scaleY);// output: 0.4966887417218543
```

Le variazioni delle dimensioni non sono proporzionali. In altre parole, se si modifica il valore `height` di un quadrato senza modificare il valore `width`, le proporzioni della figura non verranno mantenute e il quadrato si trasformerà in un rettangolo. Per effettuare modifiche relative alle dimensioni di un oggetto di visualizzazione, è possibile impostare i valori delle proprietà `scaleX` e `scaleY` per ridimensionare l'oggetto, invece di impostare le proprietà `width` e `height`. Ad esempio, il codice seguente consente di modificare la proprietà `width` dell'oggetto denominato `square`, quindi di alterare la scala verticale (`scaleY`) per farla corrispondere a quella orizzontale, in modo da mantenere le corrette proporzioni dell'oggetto.

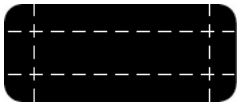
```
// Modifica direttamente width.
square.width = 150;

// Modifica la scala verticale per farla corrispondere a quella orizzontale
// e mantenere le proporzioni.
square.scaleY = square.scaleX;
```

Controllo della distorsione durante la modifica in scala

Normalmente, quando un oggetto di visualizzazione viene modificato in scala (ad esempio, allungato orizzontalmente), la distorsione che ne deriva viene equamente distribuita sull'intero oggetto, in modo che ogni sua parte venga allungata in modo uniforme. Per le immagini e gli elementi grafici, questa è probabilmente la soluzione migliore. Tuttavia, a volte è preferibile poter controllare quali porzioni dell'oggetto allungare e quali porzioni lasciare inalterate. Un comune esempio è costituito da un pulsante di forma rettangolare con gli angoli arrotondati. Con la normale modifica in scala, anche gli angoli del pulsante vengono allungati, in quanto il raggio degli angoli arrotondati viene modificato con il ridimensionamento dell'intero rettangolo.

Tuttavia, in questo caso, sarebbe preferibile poter controllare la modifica in scala e indicare le aree da modificare (i lati paralleli e il centro) e le aree da non modificare (gli angoli), in modo da non provocare una distorsione dell'oggetto.

È possibile utilizzare la modifica in scala a 9 porzioni (scala 9) per creare oggetti di visualizzazione in cui sia possibile controllare la modalità di modifica in scala dell'oggetto. Grazie alla modifica in scala 9 porzioni, l'oggetto di visualizzazione viene suddiviso in nove rettangoli distinti (una griglia 3 per 3, come quella del gioco del tris). I rettangoli non sono necessariamente delle stesse dimensioni ed è possibile scegliere dove posizionare le linee che formano la griglia. Il contenuto dei quattro rettangoli in corrispondenza degli angoli (ad esempio gli angoli smussati di un pulsante) non verrà allungato o compresso quando l'oggetto viene modificato in scala. I rettangoli centrale superiore e centrale inferiore verranno modificati in scala orizzontalmente, ma non verticalmente, mentre i rettangoli centrale sinistro e centrale destro verranno modificati verticalmente, ma non orizzontalmente. Il rettangolo centrale infine verrà modificato in scala sia in senso orizzontale che verticale.



Di conseguenza, quando si crea un oggetto di visualizzazione, se si desidera che una parte del suo contenuto non subisca modifiche in scala, è necessario posizionare le linee divisorie della griglia della modifica in scala a 9 porzioni in modo che il contenuto da non modificare si trovi in uno o più dei quattro rettangoli agli angoli.

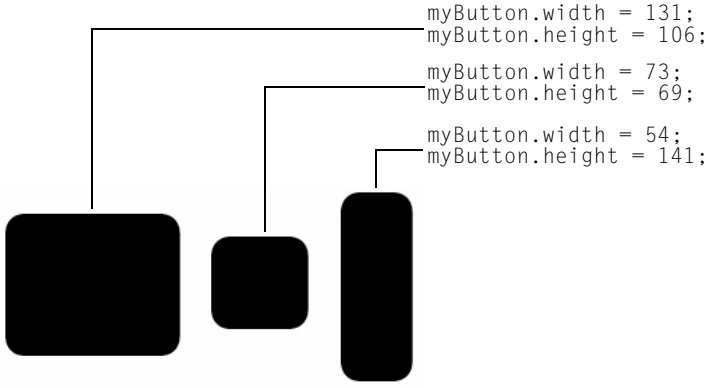
In ActionScript, se si imposta un valore per la proprietà `scale9Grid` di un oggetto di visualizzazione, viene attivata la modifica in scala a 9 porzioni per tale oggetto e viene definita la dimensione dei rettangoli che formano la griglia. È necessario utilizzare un'istanza della classe Rectangle come valore per la proprietà `scale9Grid`, come indicato di seguito:

```
myButton.scale9Grid = new Rectangle(32, 27, 71, 64);
```

I quattro parametri della funzione di costruzione di Rectangle sono la coordinata x, la coordinata y, larghezza e altezza. Nell'esempio, l'angolo superiore sinistro del rettangolo viene collocato nel punto x: 32, y: 27 dell'oggetto di visualizzazione denominato `myButton`. Il rettangolo è largo 71 pixel e alto 65 pixel (il suo bordo destro si trova alla coordinata x 103 dell'oggetto di visualizzazione, mentre il suo bordo inferiore si trova alla coordinata y 92).

L'area effettiva contenuta nell'area definita dall'istanza Rectangle rappresenta il rettangolo centrale della griglia a scala 9. I restanti rettangoli vengono calcolati da Flash Player mediante estensione dei lati dell'istanza Rectangle, come illustrato di seguito:

In questo caso, quando il pulsante viene modificato in scala, gli angoli smussati non verranno allungati o compressi, mentre tutte le altre parti verranno modificate per adattarsi al ridimensionamento.



Memorizzazione nella cache di oggetti di visualizzazione

Man mano che i progetti di Flash crescono di dimensioni, indipendentemente dal fatto che si stia creando un'applicazione o animazioni con script complesse, è necessario valutare gli aspetti legati alle prestazioni e all'ottimizzazione. Quando si dispone di contenuto che rimane statico (ad esempio un'istanza di Shape sotto forma di rettangolo), Flash non ottimizza il contenuto. Di conseguenza, quando si cambia la posizione del rettangolo, Flash deve ridisegnare l'intera istanza di Shape.

Per migliorare le prestazioni dei file SWF, è possibile memorizzare nella cache oggetti di visualizzazione specifici. L'oggetto di visualizzazione è una *superficie*, essenzialmente una versione bitmap dei dati vettoriali dell'istanza, vale a dire dei dati che non si intende modificare in modo significativo nel corso del file SWF. Pertanto, le istanze per cui è stata attivata la memorizzazione nella cache non vengono continuamente ridisegnate durante le riproduzione del file SWF, il che consente un rendering più rapido del file.

NOTA	I dati vettoriali possono essere aggiornati; quando si esegue l'aggiornamento, la superficie viene ricreata. Per questo motivo, i dati vettoriali memorizzati nella cache della superficie non devono restare immutati per tutto il file SWF.

L'impostazione della proprietà `cacheAsBitmap` di un oggetto di visualizzazione su `true` rende la cache dell'oggetto di visualizzazione una rappresentazione bitmap di se stessa. Flash crea un oggetto di superficie per l'istanza, rappresentato da una bitmap memorizzata nella cache invece che da dati vettoriali. Se si cambiano i limiti dell'oggetto di visualizzazione, la superficie viene ricreata anziché ridimensionata. Le superfici possono essere nidificate all'interno di altre superfici. La superficie secondaria copia la bitmap nella superficie principale. Per ulteriori informazioni, vedere "Attivazione della memorizzazione di bitmap nella cache" a pagina 438.

Le proprietà `opaqueBackground` e `scrollRect` della classe DisplayObject si riferiscono alla memorizzazione della bitmap nella cache mediante la proprietà `cacheAsBitmap`. Nonostante si tratti di proprietà indipendenti l'una dall'altra, `opaqueBackground` e `scrollRect` funzionano in modo ottimale quando un oggetto viene memorizzato nella cache come bitmap; in altre parole, è possibile valutare i vantaggi, in termini di prestazioni, dell'uso delle proprietà `opaqueBackground` e `scrollRect` solo se si imposta `cacheAsBitmap` su `true`. Per ulteriori informazioni sullo scorrimento del contenuto di oggetti di visualizzazione, vedere "Panoramica e scorrimento di oggetti di visualizzazione" a pagina 429. Per ulteriori informazioni sull'impostazione di uno sfondo opaco, vedere "Impostazione di un colore di sfondo opaco" a pagina 438.

Per informazioni sull'effetto maschera del canale alfa, che richiede l'impostazione della proprietà `cacheAsBitmap` su `true`, vedere "Applicazione dell'effetto maschera ai canali alfa" a pagina 446.

Quando attivare la memorizzazione nella cache

L'attivazione della funzione di memorizzazione nella cache di un oggetto di visualizzazione consente di creare una superficie, il che presenta diversi vantaggi, fra cui quello di consentire un rendering più rapido delle animazioni vettoriali complesse. L'attivazione della memorizzazione nella cache può essere utile in diversi casi. Nonostante questa funzione possa sembrare utile per migliorare le prestazioni dei file SWF in tutti i casi, ci sono situazioni in cui questa funzione non solo non migliora le prestazioni, ma, anzi, le può peggiorare. In questa sezione sono descritti i vari scenari in cui utilizzare la memorizzazione nella cache e i casi in cui è opportuno invece usare i normali oggetti di visualizzazione.

Le prestazioni generali dei dati memorizzati nella cache dipendono dalla complessità dei dati vettoriali delle istanze, dalla quantità di dati modificati e dal fatto che sia stata impostata o meno la proprietà `opaqueBackground`. Se si stanno modificando aree piccole, la differenza tra l'uso di una superficie e l'uso dei dati vettoriali è minima. Prima di distribuire l'applicazione, potrebbe essere utile provare entrambi gli scenari con il proprio lavoro.

Quando utilizzare la memorizzazione delle bitmap nella cache

Segue una descrizione degli scenari in cui l'attivazione della memorizzazione delle bitmap nella cache può comportare notevoli vantaggi.

- Immagine di sfondo complessa: un'applicazione contenente un'immagine di sfondo complessa e dettagliata di dati vettoriali (magari un'immagine in cui è stato applicato il comando Ricalca bitmap o un'immagine creata in Adobe Illustrator®). È possibile animare i personaggi sullo sfondo, operazione che però rallenta l'animazione perché lo sfondo deve continuamente rigenerare i dati vettoriali. Per migliorare le prestazioni, è possibile impostare la proprietà `opaqueBackground` dell'oggetto di visualizzazione dello sfondo su `true`. Viene eseguito il rendering dello sfondo come bitmap così che possa essere ridisegnato velocemente; in questo modo l'animazione viene riprodotta più rapidamente.
- Campo di testo a scorrimento: un'applicazione che visualizza una grande quantità di testo in un campo di testo a scorrimento. È possibile inserire il campo di testo in un oggetto di visualizzazione impostato come scorrevole con contorni a scorrimento (proprietà `scrollRect`). In questo modo si attiva lo scorrimento veloce dei pixel per l'istanza specificata. Quando un utente esegue lo scorrimento dell'istanza dell'oggetto di visualizzazione, Flash sposta verso l'alto i pixel già visualizzati e genera l'area che viene mostrata invece di rigenerare l'intero campo di testo.

- Sistema a finestre: un'applicazione con un complesso sistema di finestre che si sovrappongono. Ciascuna finestra può essere aperta o chiusa (per esempio, le finestre dei browser Web). Se si contrassegna ciascuna finestra come superficie (impostando la proprietà `cacheAsBitmap` su `true`), ogni finestra viene isolata e memorizzata nella cache. Gli utenti possono trascinare le finestre così che si sovrappongano tra loro, e ciascuna finestra non deve rigenerare il contenuto vettoriale.
- Applicazione dell'effetto maschera ai canali alfa: se si utilizza un'applicazione dell'effetto maschera ai canali alfa, è necessario impostare la proprietà `cacheAsBitmap` su `true`. Per ulteriori informazioni, vedere "Applicazione dell'effetto maschera ai canali alfa" a pagina 446.

L'attivazione della memorizzazione di bitmap nella cache in tutti questi scenari migliora la sensibilità e l'interattività dell'applicazione ottimizzando le immagini vettoriali.

Inoltre, ogni volta che si applica un filtro a un oggetto di visualizzazione, la proprietà `cacheAsBitmap` viene automaticamente impostata su `true` da Flash Player, anche è stata esplicitamente impostata su `false`. Se si cancellano tutti i filtri dall'oggetto di visualizzazione, la proprietà `cacheAsBitmap` ritorna sul valore sul quale era stata impostata.

Quando evitare di utilizzare la memorizzazione delle bitmap nella cache

L'uso scorretto di questa funzione può incidere in modo negativo sul file SWF. Quando si utilizza la memorizzazione delle bitmap nella cache, tenere presenti le seguenti linee guida:

- Non usare in modo eccessivo le superfici (oggetti di visualizzazione con memorizzazione nella cache abilitata). Ogni superficie usa più memoria di un oggetto di visualizzazione normale; pertanto, si consiglia di abilitare le superfici solo quando è necessario migliorare le prestazioni del rendering.

  Una bitmap memorizzata nella cache può utilizzare molta più memoria di una normale istanza di oggetto di visualizzazione. Per esempio, se l'oggetto di visualizzazione sullo stage misura 250 pixel per 250 pixel, quando memorizzato nella cache utilizza 250 KB invece di 1 KB, situazione che si verifica se si tratta di un'istanza di oggetto di visualizzazione normale (non memorizzata nella cache).
- Evitare di ingrandire le superfici memorizzate nella cache. Se si abusa della funzione di memorizzazione delle bitmap nella cache, si consuma una grande quantità di memoria (vedere il punto precedente) soprattutto se si aumenta il contenuto.

- Usare le superfici per le istanze di oggetti di visualizzazione che sono prevalentemente statiche (prive di animazione). È possibile trascinare o spostare l'istanza, ma il contenuto della stessa non dovrebbe animarsi né risultare molto modificato. (Animazione o modifica del contenuto sono più comuni con istanze di MovieClip contenenti animazione o un'istanza di Video.) Per esempio, se si ruota o si trasforma un'istanza, questa passa da superficie a dati vettoriali, cosa che rende difficile l'elaborazione e influisce negativamente sul file SWF.
- Se si mescolano le superfici con i dati vettoriali, si aumenta la quantità di elaborazione che Flash Player (e a volte il computer) deve eseguire. Si consiglia di raggruppare il più possibile insieme le superfici; per esempio, quando si creano applicazioni a finestra.

Attivazione della memorizzazione di bitmap nella cache

Per attivare il caching bitmap per un oggetto di visualizzazione, è necessario impostare la proprietà `cacheAsBitmap` dell'oggetto su `true`:

```
mySprite.cacheAsBitmap = true;
```

Può accadere che, dopo aver impostato la proprietà `cacheAsBitmap` su `true`, l'istanza dell'oggetto di visualizzazione esegua automaticamente l'aggancio ai pixel su coordinate intere. Quando si prova il file SWF, si può notare che il rendering delle animazioni vettoriali complesse viene eseguito più velocemente.

Se si verifica una delle condizioni elencate di seguito, la superficie (bitmap memorizzata nella cache) non viene creata anche se `cacheAsBitmap` è impostata su `true`:

- La larghezza o l'altezza della bitmap è superiore a 2880 pixel.
- La bitmap non viene assegnata (a causa di errore di memoria esaurita).

Impostazione di un colore di sfondo opaco

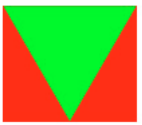
È possibile impostare uno sfondo opaco per un oggetto di visualizzazione. Per esempio, quando il file SWF dispone di uno sfondo contenente un'immagine vettoriale complessa, è possibile impostare la proprietà `opaqueBackground` su un colore specifico (solitamente lo stesso colore dello stage). Il colore viene specificato sotto forma di numero (in genere, un valore di colore esadecimale). Lo sfondo viene quindi trattato come una bitmap, che consente di ottimizzare le prestazioni.

Quando si imposta `cacheAsBitmap` su `true` e `opaqueBackground` su un colore specifico, la proprietà `opaqueBackground` consente alla bitmap interna di essere opaca e di velocizzare l'operazione di rendering. Se `cacheAsBitmap` non viene impostata su `true`, la proprietà `opaqueBackground` aggiunge una forma vettoriale quadrata opaca allo sfondo dell'oggetto di visualizzazione. Non crea automaticamente una bitmap.

L'esempio seguente mostra come impostare lo sfondo di un oggetto di visualizzazione per ottimizzare le prestazioni.

```
myShape.cacheAsBitmap = true;
myShape.opaqueBackground = 0xFF0000;
```

In questo caso, il colore di sfondo dell'oggetto Shape denominato `myShape` è impostato su rosso (`0xFF0000`). Se l'istanza di Shape contiene un triangolo verde e viene collocata su uno stage con sfondo bianco, verrebbe visualizzato un triangolo verde all'interno di un riquadro di delimitazione rosso (corrispondente al rettangolo che racchiude completamente l'istanza di Shape).



Ovviamente, questo codice avrebbe più senso se venisse utilizzato con uno stage che presenta uno sfondo rosso uniforme. Su uno sfondo colorato diverso, verrebbe invece specificato tale colore. Ad esempio, in un file SWF con sfondo bianco, la proprietà `opaqueBackground` verrebbe molto probabilmente impostata su `0xFFFFFF` o su bianco puro.

Applicazione dei metodi di fusione

I metodi di fusione consentono di combinare i colori di un'immagine (l'immagine di base) con quelli di un'altra immagine (l'immagine di fusione) per produrre una terza immagine, che viene visualizzata sullo schermo. Ogni valore di pixel di un'immagine viene elaborato con il valore di pixel corrispondente dell'altra immagine per produrre un valore di pixel per la stessa posizione all'interno del risultato.

Ogni oggetto di visualizzazione presenta una proprietà `blendMode` che può essere impostata su una delle seguenti modalità di fusione. Si tratta di costanti definite nella classe BlendMode. In alternativa, è possibile utilizzare i valori String (tra parentesi) che corrispondono ai valori effettivi delle costanti.

- `BlendMode.ADD` (`"add"`): generalmente utilizzato per creare una dissolvenza con effetto di schiarimento animato tra due immagini.
- `BlendMode.ALPHA` (`"alpha"`): generalmente utilizzato per applicare la trasparenza del primo piano sullo sfondo.
- `BlendMode.DARKEN` (`"darken"`): generalmente utilizzato per la sovrimpressione del testo.
- `BlendMode.DIFFERENCE` (`"difference"`): generalmente utilizzato per creare colori più vivaci.

- `BlendMode.ERASE` (`"erase"`): generalmente utilizzato per ritagliare (cancellare) parte dello sfondo mediante l'alfa di primo piano.
- `BlendMode.HARDLIGHT` (`"hardlight"`): generalmente utilizzato per creare effetti di ombreggiatura.
- `BlendMode.INVERT` (`"invert"`): utilizzato per invertire lo sfondo.
- `BlendMode.LAYER` (`"layer"`): utilizzato per imporre la creazione di un buffer temporaneo per la precomposizione di un oggetto di visualizzazione particolare.
- `BlendMode.LIGHTEN` (`"lighten"`): generalmente utilizzato per la sovrimpressione del tipo.
- `BlendMode.MULTIPLY` (`"multiply"`): generalmente utilizzato per creare ombre ed effetti di profondità.
- `BlendMode.NORMAL` (`"normal"`): utilizzato per specificare che i valori dei pixel dell'immagine a cui è stata applicata la fusione hanno la priorità su quelli dell'immagine di base.
- `BlendMode.OVERLAY` (`"overlay"`): generalmente utilizzato per creare effetti di ombreggiatura.
- `BlendMode.SCREEN` (`"screen"`): generalmente utilizzato per evidenziazioni ed effetti di riflesso lente.
- `BlendMode.SUBTRACT` (`"subtract"`): generalmente utilizzato per creare una dissolvenza con effetto di scurimento animato tra due immagini.

Regolazione dei colori di un oggetto di visualizzazione

È possibile utilizzare i metodi della classe ColorTransform incorporata (flash.geom.ColorTransform) per modificare il colore di un oggetto di visualizzazione. Ogni oggetto di visualizzazione presenta una proprietà `transform` che è un'istanza della classe Transform e che contiene informazioni sulle varie trasformazioni applicate all'oggetto di visualizzazione (quali rotazione, modifiche in scala o di posizione e così via). Oltre alle informazioni sulle trasformazioni geometriche, la classe Transform include anche una proprietà `colorTransform`, che è un'istanza della classe ColorTransform, e consente di modificare il colore dell'oggetto di visualizzazione. Per accedere alle informazioni sulle trasformazioni cromatiche di un oggetto di visualizzazione, è possibile usare un codice come quello riportato di seguito:

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;
```

Una volta creata un'istanza di ColorTransform, è possibile leggere i valori delle sue proprietà per scoprire quali trasformazioni cromatiche sono già state applicate oppure è possibile impostare tali valori per apportare modifiche di colore all'oggetto di visualizzazione. Per aggiornare l'oggetto di visualizzazione dopo le modifiche, è necessario riassegnare l'istanza di ColorTransform alla proprietà `transform.colorTransform`.

```
var colorInfo:ColorTransform = my DisplayObject.transform.colorTransform;

// Apportare qui alcune modifiche cromatiche.

// Salvare la modifica.
myDisplayObject.transform.colorTransform = colorInfo;
```

Impostazione dei valori di colore con il codice

La proprietà `color` della classe ColorTransform può essere utilizzata per assegnare un valore di colore rosso, verde, blu (RGB) specifico all'oggetto di visualizzazione. Nell'esempio seguente, la proprietà `color` viene utilizzata per modificare il colore dell'oggetto di visualizzazione denominato `square` in blu, quando l'utente fa clic su un pulsante denominato `blueBtn`:

```
// square è un oggetto di visualizzazione presente sullo stage.
// blueBtn, redBtn, greenBtn e blackBtn sono pulsanti presenti sullo stage.

import flash.events.MouseEvent;
import flash.geom.ColorTransform;

// Accede all'istanza ColorTransform associata a square.
var colorInfo:ColorTransform = square.transform.colorTransform;

// Questa funzione viene richiamata quando si fa clic su blueBtn.
function makeBlue(event:MouseEvent):void
{
  // Imposta il colore dell'oggetto ColorTransform.
  colorInfo.color = 0x003399;
  // Applica la modifica all'oggetto di visualizzazione
  square.transform.colorTransform = colorInfo;
}

blueBtn.addEventListener(MouseEvent.CLICK, makeBlue);
```

Si tenga presente che quando si modifica il colore di un oggetto di visualizzazione mediante la proprietà `color`, viene completamente modificato il colore dell'intero oggetto, anche se precedentemente l'oggetto era di più colori. Ad esempio, se un oggetto di visualizzazione contiene un cerchio verde con un testo nero sopra e si imposta la proprietà `color` dell'istanza di ColorTransform associata a tale oggetto su una tonalità di rosso, l'intero oggetto (cerchio e testo compresi) diventerà rosso (non sarà più possibile distinguere il testo dal resto dell'oggetto).

Modifica di effetti di colore e luminosità con il codice

Si supponga di avere un oggetto con più colori (ad esempio una foto digitale) e che non si desideri ricolorare completamente l'oggetto, ma soltanto modificare il colore di un oggetto, a partire dai colori esistenti. In uno scenario di questo tipo, la classe ColorTransform include una serie di proprietà di offset e moltiplicatrici che è possibile includere per effettuare questo tipo di modifica. Le proprietà moltiplicatrici denominate `redMultiplier`, `greenMultiplier`, `blueMultiplier` e `alphaMultiplier` funzionano come filtri fotografici colorati (o lenti colorate) e consentono di amplificare o smorzare alcuni colori dell'oggetto di visualizzazione. Le proprietà di offset (`redOffset`, `greenOffset`, `blueOffset` e `alphaOffset`) possono essere utilizzate per aggiungere quantità extra di un determinato colore all'oggetto oppure per specificare il valore minimo che un colore particolare può avere.

Queste proprietà di offset e moltiplicatrici sono identiche alle impostazione di colore avanzate disponibili per i simboli dei clip filmati nello strumento di creazione di Flash quando si seleziona Avanzato dal menu Colore della finestra di ispezione Proprietà.

Il codice seguente consente di caricare un'immagine JPEG e di applicare una trasformazione del colore che modifica i canali rosso e verde nel momento in cui il puntatore del mouse viene spostato lungo l'asse x e l'asse y. In questo caso, poiché non vengono specificati valori di offset, il valore del colore di ciascun canale visualizzato sullo schermo corrisponderà a una percentuale del valore del colore originale dell'immagine; in altre parole, la quantità massima di rosso o verde visualizzata in un dato pixel corrisponderà alla quantità originale di rosso o verde di tale pixel.

```
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.geom.Transform;
import flash.geom.ColorTransform;
import flash.net.URLRequest;

// Carica un'immagine sullo stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/
  images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Questa funzione viene chiamata quando il mouse viene posizionato
// sull'immagine caricata.
function adjustColor(event:MouseEvent):void
{
  // Accede all'oggetto ColorTransform dal Loader (contenente l'immagine)
  var colorTransformer:ColorTransform = loader.transform.colorTransform;
```

```
    // Imposta i moltiplicatori rosso e verde in base alla posizione
    // del mouse. Il valore rosso va da 0% (nessun rosso) quando il cursore
    // si trova a sinistra a 100% rosso (immagine normale) quando il cursore
    // si trova a destra. La stessa cosa viene applicata al canale verde,
    // controllato però dalla posizione del mouse sull'asse y.
    colorTransformer.redMultiplier = (loader.mouseX / loader.width) * 1;
    colorTransformer.greenMultiplier = (loader.mouseY / loader.height) * 1;

    // Applica le modifiche all'oggetto di visualizzazione
    loader.transform.colorTransform = colorTransformer;
}

loader.addEventListener(MouseEvent.MOUSE_MOVE, adjustColor);
```

Rotazione degli oggetti

Gli oggetti di visualizzazione possono essere ruotati mediante la proprietà `rotation`. Si può leggere il valore di questa proprietà per sapere se un oggetto è stato ruotato, oppure si può ruotare tale oggetto impostando la proprietà su un numero (in gradi) che rappresenta la rotazione da applicare. Ad esempio, questa riga di codice consente di ruotare l'oggetto denominato `square` di 45 gradi (un ottavo di rivoluzione completa):

```
square.rotation = 45;
```

In alternativa, è possibile ruotare un oggetto di visualizzazione utilizzando una matrice di trasformazione, come descritto in Capitolo 13, "Operazioni con le funzioni geometriche" a pagina 463.

Applicazione della dissolvenza agli oggetti

È possibile controllare la trasparenza di un oggetto per renderlo parzialmente o completamente trasparente, oppure modificare la trasparenza per creare un effetto di dissolvenza. La proprietà `alpha` della classe DisplayObject definisce la trasparenza (o, più precisamente, l'opacità) di un oggetto di visualizzazione. La proprietà `alpha` può essere impostata su un valore compreso tra 0 e 1, dove 0 corrisponde alla completa trasparenza e 1 alla completa opacità. Ad esempio, le linee di codice seguenti consentono di rendere un oggetto denominato `myBall` parzialmente trasparente (al 50 percento) quando viene selezionato con un clic del mouse:

```
function fadeBall(event:MouseEvent):void
{
  myBall.alpha = .5;
}
myBall.addEventListener(MouseEvent.CLICK, fadeBall);
```

È inoltre possibile modificare la trasparenza di un oggetto di visualizzazione mediante le impostazioni di regolazione dei colori accessibili dalla classe ColorTransform. Per ulteriori informazioni, vedere "Regolazione dei colori di un oggetto di visualizzazione" a pagina 440.

Mascheratura degli oggetti di visualizzazione

È possibile utilizzare un oggetto di visualizzazione come maschera per creare un'area trasparente attraverso cui sia visibile il contenuto di un altro oggetto di visualizzazione.

Definizione di una maschera

Per indicare che un oggetto di visualizzazione è la maschera di un altro oggetto, impostare l'oggetto di mascheratura come proprietà `mask` dell'oggetto di visualizzazione da mascherare:

```
// Rende l'oggetto maskSprite la maschera dell'oggetto mySprite.
mySprite.mask = maskSprite;
```

L'oggetto di visualizzazione mascherato verrà rivelato sotto tutte le aree opache (non trasparenti) dell'oggetto di visualizzazione che agisce da maschera. Ad esempio, il codice seguente crea un'istanza di Shape contenente un quadrato rosso di 100 per 100 pixel e un'istanza di Sprite contenente un cerchio blu con un raggio di 25 pixel. Se si fa clic sul cerchio, esso viene impostato come maschera del quadrato, in modo che la sola parte visibile di quest'ultimo sia la parte coperta dall'area piena del cerchio. In altre parole, risulterà visibile solo un cerchio rosso.

```
// Si presuppone che questo codice venga eseguito all'interno di
// un contenitore di oggetti di visualizzazione
// quale un'istanza di MovieClip o di Sprite.

import flash.display.Shape;

// Disegna un quadrato e lo aggiunge all'elenco di visualizzazione.
var square:Shape = new Shape();
square.graphics.lineStyle(1, 0x000000);
square.graphics.beginFill(0xff0000);
square.graphics.drawRect(0, 0, 100, 100);
square.graphics.endFill();
this.addChild(square);

// Disegna un cerchio e lo aggiunge all'elenco di visualizzazione.
var circle:Sprite = new Sprite();
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0x0000ff);
circle.graphics.drawCircle(25, 25, 25);
circle.graphics.endFill();
this.addChild(circle);
```

```
function maskSquare(event:MouseEvent):void
{
  square.mask = circle;
  circle.removeEventListener(MouseEvent.CLICK, maskSquare);
}

circle.addEventListener(MouseEvent.CLICK, maskSquare);
```

L'oggetto di visualizzazione che funge da maschera può essere trascinato, animato e ridimensionato dinamicamente; inoltre è possibile utilizzare forme separate in un'unica maschera. L'oggetto di visualizzazione di mascheratura non deve necessariamente essere inserito nell'elenco di visualizzazione. Tuttavia, se si desidera che l'oggetto maschera venga modificato in scala insieme allo stage o se si desidera abilitare l'interazione dell'utente con la maschera (ad es. mediante trascinamento o ridimensionamento controllati dall'utente), è necessario che l'oggetto maschera si trovi nell'elenco di visualizzazione. L'effettivo ordine di impilamento (o ordine di profondità) degli oggetti di visualizzazione non è importante, a condizione che l'oggetto maschera venga inserito nell'elenco di visualizzazione. (L'oggetto maschera verrà visualizzato sullo schermo esclusivamente come maschera.) Se l'oggetto maschera è un'istanza di MovieClip con più fotogrammi, tutti i fotogrammi vengono riprodotti nella sua linea temporale, esattamente come se l'istanza non fungesse da maschera. Per rimuovere la maschera, impostare la proprietà `mask` su `null`:

```
// Rimuove la maschera da mySprite
mySprite.mask = null;
```

Non è possibile utilizzare una maschera per mascherarne un'altra, né impostare la proprietà `alpha` di un oggetto di visualizzazione utilizzato come maschera. In un oggetto di visualizzazione impostato come maschera vengono utilizzati solo i riempimenti, mentre i tratti vengono ignorati.

Applicazione dell'effetto maschera ai caratteri dispositivo

È possibile utilizzare un oggetto di visualizzazione per mascherare il testo impostato con un carattere dispositivo. Se si utilizza un oggetto di visualizzazione per mascherare un testo impostato in un carattere dispositivo, il riquadro di delimitazione rettangolare della maschera viene utilizzato come area di mascheratura. Questo significa che se per un testo con un carattere dispositivo si crea una maschera di oggetto di visualizzazione di forma non rettangolare, la maschera che appare nel file SWF assume la forma del riquadro rettangolare di delimitazione e non quella della maschera stessa.

Applicazione dell'effetto maschera ai canali alfa

L'applicazione dell'effetto maschera ai canali alfa viene supportato se sia l'oggetto di visualizzazione di maschera che quelli mascherati utilizzano la memorizzazione delle bitmap nella cache, come illustrato di seguito.

```
// maskShape è un'istanza di Shape che include un riempimento del gradiente.
mySprite.cacheAsBitmap = true;
maskShape.cacheAsBitmap = true;
mySprite.mask = maskShape;
```

Ad esempio, un'applicazione dell'effetto maschera ai canali alfa deve utilizzare un filtro sull'oggetto maschera indipendentemente dal filtro applicato all'oggetto di visualizzazione mascherato.

Nell'esempio seguente, un file di immagine esterno viene caricato sullo stage. Tale immagine (o più precisamente, l'istanza di Loader nella quale viene caricata) corrisponderà all'oggetto di visualizzazione che viene mascherato. Un gradiente ovale (centro completamente nero che diventa trasparente sui bordi) viene disegnato sopra l'immagine; questa figura corrisponderà alla maschera alfa. Per entrambi gli oggetti di visualizzazione è stato attivato il caching bitmap. L'ovale viene impostato come maschera dell'immagine e viene successivamente reso trascinabile.

```
// Si presuppone che questo codice venga eseguito all'interno di
// un contenitore di oggetti di visualizzazione
// quale un'istanza di MovieClip o di Sprite.

import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Sprite;
import flash.geom.Matrix;
import flash.net.URLRequest;

// Carica un'immagine e aggiungila all'elenco di visualizzazione.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/
  images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Crea un'istanza di Sprite.
var oval:Sprite = new Sprite();
// Disegna un gradiente ovale.
var colors:Array = [0x000000, 0x000000];
var alphas:Array = [1, 0];
var ratios:Array = [0, 255];
var matrix:Matrix = new Matrix();
```

```
matrix.createGradientBox(200, 100, 0, -100, -50);
oval.graphics.beginGradientFill(GradientType.RADIAL,
                  colors,
                  alphas,
                  ratios,
                  matrix);
oval.graphics.drawEllipse(-100, -50, 200, 100);
oval.graphics.endFill();
// Aggiunge lo Sprite all'elenco di visualizzazione
this.addChild(oval);

// Imposta cacheAsBitmap = true per entrambi gli oggetti di visualizzazione.
loader.cacheAsBitmap = true;
oval.cacheAsBitmap = true;
// Imposta l'ovale come maschera di loader (e del suo elemento secondario,
// l'immagine caricata)
loader.mask = oval;

// Rende l'ovale trascinabile.
oval.startDrag(true);
```

Animazione di oggetti

Si definisce animazione il processo di mettere qualcosa in movimento, oppure di modificare qualcosa in un determinato arco di tempo. L'animazione con script è una parte fondamentale dei video giochi e viene spesso utilizzata per aggiungere un tocco di ricercatezza e utili indicazioni di interazione ad altre applicazioni.

L'idea fondamentale alla base dell'animazione con script è il verificarsi di un cambiamento suddiviso in incrementi lungo un intervallo temporale. In ActionScript, rendere un processo ripetitivo è facile, grazie a una comune istruzione ciclica. Tuttavia, un ciclo deve eseguire tutte le sue iterazioni prima di aggiornare lo schermo. Per creare un'animazione mediante script, è necessario scrivere un codice ActionScript che consenta di eseguire alcune azioni in modo ripetuto nel tempo, così come di aggiornare lo schermo ogni volta che tale script viene eseguito.

Ad esempio, si immagini di creare una semplice animazione, quale una palla che attraversa lo schermo. ActionScript include un semplice meccanismo che consente di tenere traccia del passaggio del tempo e di aggiornare lo schermo di conseguenza; vale a dire, è possibile scrivere un codice in grado di fare percorre alla palla una piccola porzione di spazio durante ogni porzione di tempo, fino a portarla a destinazione. Dopo ogni singolo movimento, lo schermo viene aggiornato, rendendo visibile un movimento che attraversa lo stage.

Da un punto di vista pratico, ha senso sincronizzare l'animazione mediante script con la velocità fotogrammi del file SWF (in altre parole, effettuare una variazione dell'animazione ogni volta che viene visualizzato un nuovo fotogramma), in quanto la velocità dei fotogrammi corrisponde alla velocità di aggiornamento dello schermo da parte di Flash Player. Ogni oggetto di visualizzazione presenta un evento `enterFrame` che viene inviato in base alla velocità fotogrammi del file SWF, a un evento per fotogramma. La maggior parte degli sviluppatori che creano animazioni mediante script impiegano l'evento `enterFrame` come strumento per creare azioni che si ripetono nel tempo. È possibile scrivere un codice listener per l'evento `enterFrame` che consenta di spostare la palla animata di una determinata porzione di spazio per ogni fotogramma, affinché, con l'aggiornamento dello schermo (a ogni fotogramma) la palla venga ridisegnata nella nuova posizione, creando un effetto di movimento.

NOTA

Per eseguire un'azione in modo ripetuto nel tempo, è anche possibile utilizzare la classe Timer. Un'istanza di Timer attiva una notifica evento al trascorre di ogni determinato intervallo di tempo. È possibile scrivere un codice che consenta di eseguire l'animazione mediante la gestione dell'evento timer della classe Timer, impostando l'intervallo di tempo su un valore molto basso (frazioni di secondo). Per ulteriori informazioni sull'uso della classe Timer, vedere "Controllo degli intervalli di tempo" a pagina 210.

Nell'esempio seguente, viene creata sullo stage un'istanza di Sprite a forma di cerchio denominata `circle`. Se si fa clic sul cerchio, viene avviata una sequenza animata mediante script che provoca la dissolvenza di `circle` (la sua proprietà `alpha` viene diminuita) fino a renderlo completamente trasparente:

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.MouseEvent;

// Disegna un cerchio e lo aggiunge all'elenco di visualizzazione.
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0x990000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();
addChild(circle);

// Quando l'animazione viene avviata, questa funzione viene richiamata a
// ogni fotogramma.
// La modifica apportata da questa funzione (aggiornamento dello schermo
// a ogni fotogramma) provoca il verificarsi dell'animazione.
```

```
function fadeCircle(event:Event):void
{
  circle.alpha -= .05;

  if (circle.alpha <= 0)
  {
    circle.removeEventListener(Event.ENTER_FRAME, fadeCircle);
  }
}

function startAnimation(event:MouseEvent):void
{
  circle.addEventListener(Event.ENTER_FRAME, fadeCircle);
}

circle.addEventListener(MouseEvent.CLICK, startAnimation);
```

Quando si fa clic sul cerchio, la funzione `fadeCircle()` viene registrata come listener dell'evento `enterFrame` e inizia a essere chiamata a ogni passaggio di fotogramma. Tale funzione provoca una dissolvenza di `circle` modificando la sua proprietà `alpha`; a ogni fotogramma, la proprietà `alpha` di circle viene diminuita di 0,05 (5 percento) e lo schermo viene aggiornato di conseguenza. Alla fine, quando il valore di `alpha` raggiunge 0 (e `circle` è completamente trasparente), la funzione `fadeCircle()` viene rimossa come listener dell'evento e l'animazione termina.

Lo stesso codice può essere utilizzato, ad esempio, per creare una vera e propria animazione, anziché una dissolvenza. Sostituendo la proprietà `alpha` con una proprietà differente nella funzione che viene registrata come listener dell'evento `enterFrame`, tale proprietà risulterà animata. Ad esempio, se si modifica la riga

```
  circle.alpha -= .05;
```

nel seguente codice

```
  circle.x += 5;
```

la proprietà `x` risulterà animata e il cerchio verrà spostato verso destra attraverso lo stage. La condizione che termina l'animazione (vale a dire, l'annullamento della registrazione come listener di `enterFrame`) può essere modificata quando viene raggiunta la coordinata x desiderata.

Caricamento dinamico di contenuto di visualizzazione

È possibile caricare una delle seguenti risorse di visualizzazione esterne in un'applicazione di ActionScript 3.0:

- File SWF creato in ActionScript 3.0 (Sprite, MovieClip o qualsiasi altra classe che estenda Sprite).
- File di immagine (inclusi file JPG, PNG e GIF).
- File SWF AVM1 (file SWF scritto in ActionScript 1.0 o 2.0).

Per caricare queste risorse è necessario utilizzare la classe Loader.

Caricamento di oggetti di visualizzazione

Gli oggetti Loader vengono utilizzati per caricare file SWF e di immagini nelle applicazioni. La classe Loader è una sottoclasse della classe DisplayObjectContainer. Un oggetto Loader può contenere nel proprio elenco di visualizzazione un solo oggetto di visualizzazione secondario, vale a dire l'oggetto di visualizzazione rappresentante il file SWF o di immagine che carica. Quando si carica un oggetto Loader nell'elenco di visualizzazione (come illustrato nel codice seguente), l'oggetto di visualizzazione secondario caricato viene anche aggiunto nell'elenco di visualizzazione:

```
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
this.addChild(pictLdr);
```

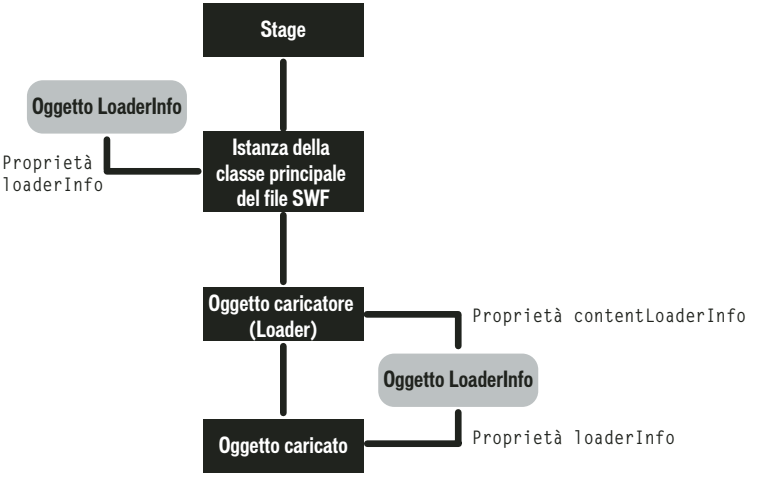
Una volta caricato il file SWF o l'immagine, è possibile spostare l'oggetto di visualizzazione caricato in un altro contenitore di oggetti di visualizzazione, come l'oggetto DisplayObjectContainer di `container` nell'esempio seguente:

```
import flash.display.*;
import flash.net.URLRequest;
import flash.events.Event;
var container:Sprite = new Sprite();
addChild(container);
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
pictLdr.contentLoaderInfo.addEventListener(Event.COMPLETE, imgLoaded);
function imgLoaded(event:Event):void
{
  container.addChild(pictLdr.content);
}
```

Monitoraggio dello stato di avanzamento del caricamento

Una volta avviato il caricamento del file, viene creato un oggetto LoaderInfo. L'oggetto LoaderInfo contiene informazioni che includono lo stato di avanzamento del caricamento, gli URL del loader e del contenuto caricato, il numero totale di byte del contenuto multimediale e l'altezza e larghezza nominali dello stesso. L'oggetto LoaderInfo invia inoltre eventi relativi al monitoraggio dello stato di avanzamento del caricamento.

Il diagramma seguente mostra i diversi usi dell'oggetto LoaderInfo: per l'istanza della classe principale del file SWF, per un oggetto Loader e per un oggetto caricato dall'oggetto Loader:



LoaderInfo è accessibile come una proprietà dell'oggetto Loader e dell'oggetto di visualizzazione caricato. Non appena il caricamento viene avviato, è possibile accedere all'oggetto LoaderInfo mediante la proprietà `contentLoaderInfo` dell'oggetto Loader. Al termine del caricamento dell'oggetto di visualizzazione, l'oggetto LoaderInfo risulta comunque accessibile come proprietà dell'oggetto di visualizzazione caricato mediante la proprietà `loaderInfo` dell'oggetto di visualizzazione. La proprietà `loaderInfo` dell'oggetto di visualizzazione caricato fa riferimento allo stesso oggetto LoaderInfo della proprietà `contentLoaderInfo` dell'oggetto Loader. In altre parole, un oggetto LoaderInfo è condiviso da un file di oggetto caricato e dall'oggetto Loader che lo ha caricato.

Per accedere alle proprietà del contenuto caricato, è necessario aggiungere un listener di evento all'oggetto LoaderInfo, come nel codice seguente:

```
import flash.display.Loader;
import flash.display.Sprite;
import flash.events.Event;

var ldr:Loader = new Loader();
var urlReq:URLRequest = new URLRequest("Circle.swf");
ldr.load(urlReq);
ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, loaded);
addChild(ldr);

function loaded(event:Event):void
{
  var content:Sprite = event.target.content;
  content.scaleX = 2;
}
```

Per ulteriori informazioni, vedere Capitolo 10, "Gestione degli eventi" a pagina 335.

Impostazione del contesto di caricamento

Quando si carica un file esterno in Flash Player mediante il metodo `load()` o `loadBytes()` della classe Loader, è possibile, se lo si desidera, specificare un parametro `context`. Tale parametro è un oggetto di LoaderContext.

La classe LoaderContext include tre proprietà che consentono di definire il contesto di utilizzo del contenuto caricato:

- `checkPolicyFile`: Utilizzare questa proprietà solo per caricare file di immagine (non file SWF). Se questa proprietà viene impostata su `true`, l'oggetto Loader cerca nel server di origine un file di criteri dei domini (vedere "Controlli del sito Web (file di criteri dei domini)" a pagina 816). Ciò è necessario unicamente per contenuto derivante da domini diversi da quello del file SWF contenente l'oggetto Loader. Se il server concede l'autorizzazione di accesso al dominio di Loader, ActionScript è in grado di accedere ai dati dell'immagine caricata dai file SWF presenti nel dominio di Loader; in altre parole, è possibile usare il comando `BitmapData.draw()` per accedere ai dati dell'immagine caricata.

  Si tenga presente che un file SWF appartenente a un dominio diverso da quello dell'oggetto Loader può chiamare il metodo `Security.allowDomain()` per consentire l'accesso a un dominio specifico.

- `securityDomain`: Utilizzare questa proprietà solo per caricare un file SWF (non un'immagine). Specificare questa proprietà per file SWF appartenenti a un dominio differente da quello del file contenente l'oggetto Loader. Se si specifica questa opzione, Flash Player verifica l'esistenza di un file dei criteri dei domini e, se lo trova, consente ai file SWF dei domini autorizzati nel file dei criteri dei domini di eseguire scambi di script con il contenuto SWF caricato. È possibile specificare `flash.system.SecurityDomain.currentDomain` per questo parametro.
- `applicationDomain`: Utilizzare questa proprietà solo per il caricamento di un file SWF scritto in ActionScript 3.0 (non un'immagine o un file SWF scritto in ActionScript 1.0 o 2.0). Quando si carica il file, è possibile specificare che il file venga incluso nello stesso dominio applicazione dell'oggetto Loader impostando il parametro `applicationDomain` su `flash.system.ApplicationDomain.currentDomain`. Se si inserisce il file SWF caricato nello stesso dominio applicazione, sarà possibile accedere direttamente alle sue classi. Ciò può risultare utile se si carica un file SWF con contenuti multimediali incorporati, ai quali si potrà accedere attraverso i relativi nomi di classi associati. Per ulteriori informazioni, vedere "Uso della classe ApplicationDomain" a pagina 745.

Segue un esempio di verifica di un file di criteri dei domini durante il caricamento di una bitmap da un altro dominio:

```
var context:LoaderContext = new LoaderContext();
context.checkPolicyFile = true;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/
  photo11.jpg");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

Segue un esempio di verifica di un file dei criteri dei domini durante il caricamento di un file SWF da un altro dominio, al fine di collocare il file nella stessa funzione di sicurezza sandbox dell'oggetto Loader. Inoltre, il codice inserisce le classi del file SWF caricato nello stesso dominio applicazione dell'oggetto Loader:

```
var context:LoaderContext = new LoaderContext();
context.securityDomain = SecurityDomain.currentDomain;
context.applicationDomain = ApplicationDomain.currentDomain;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/
  library.swf");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

Per ulteriori informazioni, vedere la classe LoaderContext nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Esempio: SpriteArranger

L'esempio SpriteArranger si basa sull'applicazione Geometric Shapes di esempio descritta a parte (vedere "Esempio: GeometricShapes" a pagina 194).

L'applicazione SpriteArranger di esempio illustra una serie di concetti relativi all'uso degli oggetti di visualizzazione, quali:

- Estensione delle classi degli oggetti di visualizzazione
- Aggiunta di oggetti all'elenco di visualizzazione
- Ordinamento su livelli degli oggetti di visualizzazione e uso dei contenitori degli oggetti di visualizzazione
- Risposta a eventi di oggetti di visualizzazione
- Uso di proprietà e metodi degli oggetti di visualizzazione

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione GeometricShapes si trovano nella cartella Examples/SpriteArranger. L'applicazione è costituita dai seguenti file:

File	Descrizione
SpriteArranger.mxml oppure SpriteArranger.fla	Il file principale dell'applicazione in Flash (FLA) o Flex (MXML)
com/example/programmingas3/SpriteArranger/CircleSprite.as	Classe che definisce un tipo di oggetto Sprite in grado di eseguire il rendering di un cerchio sullo schermo.
com/example/programmingas3/SpriteArranger/DrawingCanvas.as	Classe che definisce l'area di lavoro, ovvero un contenitore di oggetti di visualizzazione contenente oggetti GeometricSprite.
com/example/programmingas3/SpriteArranger/SquareSprite.as	Classe che definisce un tipo di oggetto Sprite in grado di eseguire il rendering di un quadrato sullo schermo.
com/example/programmingas3/SpriteArranger/TriangleSprite.as	Classe che definisce un tipo di oggetto Sprite in grado di eseguire il rendering di un triangolo sullo schermo.
com/example/programmingas3/SpriteArranger/GeometricSprite.as	Classe che estende l'oggetto Sprite, utilizzata per definire una figura geometrica sullo schermo. Anche CircleSprite, SquareSprite e TriangleSprite estendono questa classe.

File	Descrizione
com/example/programmingas3/geometricshapes/IGeometricShape.as	Metodi di definizione interfaccia di base da implementare in tutte le classi delle figure geometriche.
com/example/programmingas3/geometricshapes/IPolygon.as	Metodo di definizione di interfaccia da implementare nelle classi delle figure geometriche che presentano più lati.
com/example/programmingas3/geometricshapes/RegularPolygon.as	Tipo di figura geometrica che presenta lati di uguale lunghezza posizionati simmetricamente attorno al centro della figura.
com/example/programmingas3/geometricshapes/Circle.as	Tipo di figura geometrica che definisce un cerchio.
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	Sottoclasse di RegularPolygon che definisce un triangolo con lati di pari lunghezza.
com/example/programmingas3/geometricshapes/IPolygon.as	Sottoclasse di RegularPolygon che definisce un rettangolo con quattro lati di pari lunghezza.
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	Classe contenente un metodo factory per la creazione di figure geometriche a partire da una forma e da una dimensione specificate.

Definizione delle classi SpriteArranger

L'applicazione SpriteArranger consente di aggiungere una varietà di oggetti di visualizzazione all'area di lavoro visualizzata sullo schermo.

La classe DrawingCanvas definisce un'area di disegno, un tipo di contenitore di oggetti di visualizzazione al quale è possibile aggiungere figure geometriche visualizzate sullo schermo. Tali forme sono istanze di una delle sottoclassi della classe GeometricSprite.

Classe DrawingCanvas

La classe DrawingCanvas estende la classe Sprite e questa ereditarietà è definita nella dichiarazione della classe DrawingCanvas nel modo seguente:

```
public class DrawingCanvas extends Sprite
```

La classe Sprite è una sottoclasse delle classi DisplayObjectContainer e DisplayObject e la classe DrawingCanvas utilizza i metodi e le proprietà di queste classi.

Il metodo di costruzione `DrawingCanvas()` consente di impostare un oggetto rettangolo denominato `bounds` che verrà utilizzato successivamente per tracciare i contorni dell'area di lavoro. Il metodo di costruzione chiama quindi il metodo `initCanvas()`, come indicato di seguito:

```
this.bounds = new Rectangle(0, 0, w, h);
initCanvas(fillColor, lineColor);
```

Come indica l'esempio seguente, il metodo `initCanvas()` definisce varie proprietà dell'oggetto DrawingCanvas che sono state trasmesse come argomenti alla funzione di costruzione:

```
this.lineColor = lineColor;
this.fillColor = fillColor;
this.width = 500;
this.height = 200;
```

Il metodo `initCanvas()` richiama quindi il metodo `drawBounds()`, che disegna l'area di lavoro utilizzando la proprietà `graphics` della classe DrawingCanvas. La proprietà `graphics` viene ereditata dalla classe Shape.

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
this.graphics.drawRect(bounds.left - 1,
             bounds.top - 1,
             bounds.width + 2,
             bounds.height + 2);
this.graphics.endFill();
```

I seguenti metodi aggiuntivi della classe DrawingCanvas vengono richiamati in base alle interazioni dell'utente con l'applicazione:

- Metodi `addShape()` e `describeChildren()`, descritti nella sezione "Aggiunta di oggetti di visualizzazione all'area di lavoro" a pagina 458.
- Metodi `moveToBack()`, `moveDown()`, `moveToFront()` e `moveUp()`, descritti nella sezione "Modifica della disposizione dei livelli degli oggetti di visualizzazione" a pagina 461.
- Metodo `onMouseUp()`, descritto nella sezione "Selezione e trascinamento di oggetti di visualizzazione" a pagina 459.

Classe GeometricSprite e relative sottoclassi

Ogni oggetto di visualizzazione che l'utente può inserire nell'area di lavoro è un'istanza di una delle seguenti sottoclassi di GeometricSprite:

- CircleSprite
- SquareSprite
- TriangleSprite

La classe GeometricSprite estende la classe flash.display.Sprite:

```
public class GeometricSprite extends Sprite
```

La classe GeometricSprite include varie proprietà comuni a tutti gli oggetti GeometricSprite. Tali proprietà vengono impostate nella funzione di costruzione, in base a una serie di parametri trasmessi alla funzione. Ad esempio:

```
this.size = size;
this.lineColor = lColor;
this.fillColor = fColor;
```

La proprietà `geometricShape` della classe GeometricSprite definisce un'interfaccia IGeometricShape che definisce una serie di proprietà matematiche, ma non le proprietà visive, della figura. Le classi che implementano l'interfaccia IGeometricShape vengono definite nell'applicazione di esempio GeometricShapes (vedere "Esempio: GeometricShapes" a pagina 194).

La classe GeometricSprite definisce il metodo `drawShape()`, che viene ulteriormente perfezionato dalle definizioni di sostituzione di ciascuna sottoclasse di GeometricSprite. Per ulteriori informazioni, vedere la sezione "Aggiunta di oggetti di visualizzazione all'area di lavoro" di seguito.

La classe GeometricSprite fornisce inoltre i seguenti metodi:

- Metodi `onMouseDown()` e `onMouseUp()`, descritti nella sezione "Selezione e trascinamento di oggetti di visualizzazione" a pagina 459.
- Metodi `showSelected()` e `hideSelected()`, descritti nella sezione"Selezione e trascinamento di oggetti di visualizzazione" a pagina 459.

Aggiunta di oggetti di visualizzazione all'area di lavoro

Quando l'utente fa clic sul pulsante Aggiungi forma, l'applicazione richiama il metodo `addShape()` della classe DrawingCanvas. Tale metodo crea una nuova istanza di GeometricSprite richiamando la funzione di costruzione appropriata di una delle sottoclassi di GeometricSprite, come illustrato dal seguente esempio:

```
public function addShape(shapeName:String, len:Number):void
{
  var newShape:GeometricSprite;
  switch (shapeName)
  {
    case "Triangle":
      newShape = new TriangleSprite(len);
      break;

    case "Square":
      newShape = new SquareSprite(len);
      break;

    case "Circle":
      newShape = new CircleSprite(len);
      break;
  }
  newShape.alpha = 0.8;
  this.addChild(newShape);
}
```

Ciascun metodo della funzione di costruzione richiama il metodo `drawShape()`, che impiega la proprietà `graphics` della classe (ereditata dalla classe Sprite) per disegnare la grafica vettoriale appropriata. Ad esempio, il metodo `drawShape()` della classe CircleSprite include il seguente codice:

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
var radius:Number = this.size / 2;
this.graphics.drawCircle(radius, radius, radius);
```

La penultima riga della funzione `addShape()` imposta la proprietà `alpha` dell'oggetto di visualizzazione (ereditata dalla classe DisplayObject), per fare in modo che ogni oggetto di visualizzazione aggiunto all'area di lavoro risulti leggermente trasparente, affinché sia possibile vedere gli oggetti sotto di esso.

L'ultima riga del metodo `addChild()` aggiunge il nuovo oggetto di visualizzazione all'elenco secondario dell'istanza della classe DrawingCanvas, che si trova già nell'elenco di visualizzazione. In tal modo, il nuovo oggetto di visualizzazione viene visualizzato nello stage.

L'interfaccia dell'applicazione include due campi di testo, `selectedSpriteTxt` e `outputTxt`. Le proprietà di testo di tali campi vengono aggiornate con informazioni relative agli oggetti GeometricSprite che sono stati aggiunti all'area di lavoro oppure selezionati dall'utente. La classe GeometricSprite gestisce questa attività di informazione sostituendo il metodo `toString()`, come segue:

```
public override function toString():String
{
  return this.shapeType + " of size " + this.size + " at " + this.x + ", "
  + this.y;
}
```

La proprietà `shapeType` viene impostata sul valore appropriato nel metodo della funzione di costruzione di ogni sottoclasse di GeometricSprite. Ad esempio, il metodo `toString()` potrebbe restituire il seguente valore per un'istanza di CircleSprite recentemente aggiunta all'istanza DrawingCanvas:

```
Cerchio di dimensioni 50 a 0, 0
```

Il metodo `describeChildren()` della classe DrawingCanvas esegue un ciclo all'interno dell'elenco secondario dell'area di lavoro mediante la proprietà `numChildren` (ereditata dalla classe DisplayObjectContainer) al fine di impostare il limite del ciclo `for`. Viene generata una stringa che elenca ogni elemento secondario, come indicato di seguito:

```
var desc:String = "";
var child:DisplayObject;
for (var i:int=0; i < this.numChildren; i++)
{
    child = this.getChildAt(i);
    desc += i + ": " + child + '\n';
}
```

La stringa risultante viene utilizzata per impostare la proprietà `text` del campo di testo `outputTxt`.

Selezione e trascinamento di oggetti di visualizzazione

Se l'utente fa clic su un'istanza di GeometricSprite, l'applicazione chiama il gestore di eventi `onMouseDown()`. Come illustrato nell'esempio seguente, questo gestore di eventi è impostato per intercettare eventi di pressione mouse nella funzione di costruzione della classe GeometricSprite:

```
this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

Il metodo `onMouseDown()` richiama quindi il metodo `showSelected()` dell'oggetto GeometricSprite. Se si tratta della prima chiamata di tale metodo per l'oggetto, il metodo crea un nuovo oggetto Shape denominato `selectionIndicator` e impiega la proprietà `graphics` dell'oggetto Shape per disegnare un rettangolo di evidenziazione rosso, come indicato di seguito:

```
this.selectionIndicator = new Shape();
this.selectionIndicator.graphics.lineStyle(1.0, 0xFF0000, 1.0);
this.selectionIndicator.graphics.drawRect(-1, -1, this.size + 1,
  this.size + 1);
this.addChild(this.selectionIndicator);
```

Se non si tratta della prima chiamata del metodo `onMouseDown()`, il metodo si limita a impostare la proprietà `visible` della forma `selectionIndicator` (ereditata dalla classe DisplayObject) come segue:

```
this.selectionIndicator.visible = true;
```

Il metodo `hideSelected()` nasconde la forma `selectionIndicator` dell'oggetto precedentemente selezionato impostando la sua proprietà `visible` su `false`.

Il metodo del gestore di eventi `onMouseDown()` richiama anche il metodo `startDrag()` (ereditato dalla classe Sprite), che include il codice seguente:

```
var boundsRect:Rectangle = this.parent.getRect(this.parent);
boundsRect.width -= this.size;
boundsRect.height -= this.size;
this.startDrag(false, boundsRect);
```

Questo codice consente all'utente di trascinare l'oggetto selezionato per l'area di lavoro, entro i confini impostati dal rettangolo `boundsRect`.

Quando il pulsante del mouse viene rilasciato, viene trasmesso l'evento `mouseUp`. Il metodo della funzione di costruzione di DrawingCanvas imposta il seguente listener di eventi:

```
this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
```

Questo listener di eventi è configurato per l'oggetto DrawingCanvas, anziché per i singoli oggetti di GeometricSprite, perché se l'oggetto GeometricSprite viene trascinato, potrebbe finire dietro un altro oggetto di visualizzazione (un altro oggetto GeometricSprite) quando il mouse viene rilasciato. L'oggetto di visualizzazione in primo piano riceverebbe l'evento di rilascio del mouse, ma l'oggetto trascinato dall'utente no. L'aggiunta del listener all'oggetto DrawingCanvas garantisce che l'evento venga sempre gestito.

Il metodo `onMouseUp()` richiama il metodo `onMouseUp()` dell'oggetto GeometricSprite che, a sua volta, richiama il metodo `stopDrag()` dell'oggetto GeometricSprite.

Modifica della disposizione dei livelli degli oggetti di visualizzazione

L'interfaccia utente dell'applicazione include una serie di pulsanti chiamati Porta sullo sfondo, Sposta in basso, Sposta in alto e In primo piano. Quando l'utente fa clic su uno di questi pulsanti, l'applicazione richiama il metodo corrispondente della classe DrawingCanvas: `moveToBack()`, `moveDown()`, `moveUp()` o `moveToFront()`. Ad esempio, il metodo `moveToBack()` include il seguente codice:

```
public function moveToBack(shape:GeometricSprite):void
{
  var index:int = this.getChildIndex(shape);
  if (index > 0)
  {
    this.setChildIndex(shape, 0);
  }
}
```

Il metodo impiega il metodo `setChildIndex()` (ereditato dalla classe DisplayObjectContainer) per posizionare l'oggetto di visualizzazione nella posizione di indice 0 dell'elenco secondario dell'istanza di DrawingCanvas (`this`).

Il metodo `moveDown()` funziona in modo simile, tranne che decrementa la posizione di indice dell'oggetto di visualizzazione di 1 nell'elenco secondario dell'istanza DrawingCanvas:

```
public function moveDown(shape:GeometricSprite):void
{
  var index:int = this.getChildIndex(shape);
  if (index > 0)
  {
    this.setChildIndex(shape, index - 1);
  }
}
```

I metodi `moveUp()` e `moveToFront()` funzionano in modo simile ai metodi `moveToBack()` e `moveDown()`.

CAPITOLO 13

Operazioni con le funzioni geometriche

13

Il pacchetto flash.geom contiene classi per la definizione di oggetti geometrici, ad esempio punti, rettangoli e matrici di trasformazione, da usare per definire le proprietà di oggetti usati in altre classi.

Sommario

Nozioni di base sulle funzioni geometriche

Introduzione alle operazioni con le funzioni geometriche

La geometria è spesso una di quelle materie con cui le persone si scontrano a scuola memorizzando il meno possibile tuttavia, la conoscenza delle nozioni di base di geometria può rivelarsi molto preziosa in ActionScript.

Il pacchetto flash.geom contiene classi per la definizione di oggetti geometrici, ad esempio punti, rettangoli e matrici di trasformazione. Queste classi non attivano individualmente delle funzionalità, ma vengono utilizzate per definire le proprietà degli oggetti da usare in altre classi.

Tutte le classi che consentono di eseguire funzioni geometriche si basano sul principio che qualsiasi posizione dello schermo può essere rappresentata come su un piano bidimensionale. Lo schermo è considerato un grafico con un asse orizzontale (x) e un asse verticale (y). Qualsiasi *punto* dello schermo può essere rappresentato da una coppia di valori x e y, cioè da due *coordinate*.

Ogni oggetto di visualizzazione, Stage incluso, è dotato del proprio *spazio di coordinate*, sostanzialmente un proprio grafico in cui segnare la posizione degli oggetti di visualizzazione di livello secondario, i disegni e così via. Generalmente, *l'origine*, cioè il punto con coordinate 0, 0 dove si incontrano i due assi, corrisponde all'angolo superiore sinistro dell'oggetto di visualizzazione. Questi vale sempre per lo Stage, ma può variare per altri oggetti di visualizzazione. Come in tutti i normali sistemi di coordinate bidimensionali, i valori dell'asse delle x aumentano verso destra e diminuiscono verso sinistra. I punti che si trovano a sinistra dell'origine sono caratterizzati da una coordinata x negativa. Tuttavia, contrariamente a quanto avviene nei comuni sistemi di coordinate, in ActionScript i valori sull'asse delle y aumentano verso la parte inferiore dello schermo e diminuiscono verso la parte superiore. I valori che si trovano più in alto dell'origine avranno una coordinata y negativa. Dal momento che l'angolo superiore sinistro dello Stage rappresenta l'origine dello spazio di coordinate, tutti gli oggetti dello Stage avranno una coordinata x superiore a 0 e inferiore alla larghezza dello Stage e una coordinata y superiore a 0 e inferiore all'altezza dello Stage.

Per rappresentare singoli punti in uno spazio di coordinate si possono usare istanze della classe Point. È possibile creare un'istanza di Rectangle per rappresentare una zona rettangolare in uno spazio di coordinate. Gli utenti avanzati possono usare un'istanza di Matrix per applicare trasformazioni multiple o complesse a un oggetto di visualizzazione. Molte delle trasformazioni più semplici, come la rotazione, lo spostamento e la modifica in scala possono essere applicate direttamente a un oggetto di visualizzazione tramite le proprietà dell'oggetto. Per ulteriori informazioni sull'applicazione delle trasformazioni tramite le proprietà dell'oggetto di visualizzazione, vedere "Manipolazione di oggetti di visualizzazione" a pagina 422.

Operazioni comuni con le funzioni geometriche

Le operazioni più comuni che si possono eseguire usando le classi delle funzioni geometriche in ActionScript sono le seguenti:

- Calcolo della distanza tra due punti
- Determinazione delle coordinate di un punto in spazi di coordinate diversi
- Spostamento di un oggetto di visualizzazione della distanza e inclinazione specificate
- Operazioni con istanze di Rectangle:
  - Riposizionamento di un'istanza di Rectangle
  - Ridimensionamento di un'istanza di Rectangle
  - Determinazione della dimensione combinata o delle zone di sovrapposizione di istanze di Rectangle
- Creazione di oggetti Matrix
- Uso di un oggetto Matrix per applicare trasformazioni a un oggetto di visualizzazione

Concetti e termini importanti

L'elenco di riferimento seguente contiene dei termini importanti che vengono citati in questo capitolo:

- Coordinate cartesiane: le coordinate sono generalmente rappresentate da una coppia di numeri (come 5, 12 e 17, -23). I due numeri rappresentano, rispettivamente, la coordinata x e la coordinata y.
- Spazio di coordinate: grafico delle coordinate contenuto in un oggetto di visualizzazione su cui sono collocati gli elementi secondari.
- Origine: punto dello spazio di coordinate in cui si incontrano gli assi x e y. Questo punto ha coordinate 0, 0.
- Punto: posizione unica in uno spazio di coordinate. Nel sistema di coordinate bidimensionale usato in ActionScript, un punto è definito in base alla sua posizione rispetto all'asse delle x e all'asse delle y.
- Punto di registrazione: origine (coordinate 0, 0) dello spazio di coordinate di un oggetto di visualizzazione.
- Scala: dimensione di un oggetto rispetto alla dimensione originale. Il verbo "modificare in scala" significa ridimensionare un oggetto ingrandendolo o riducendolo.
- Conversione: modifica delle coordinate di un punto da uno spazio di coordinate a un altro.

- Trasformazione: modifica di una caratteristica visiva di un'immagine, come rotazione di un oggetto, modifica della scala, inclinazione o distorsione di un oggetto, alterazione del colore.
- Asse delle x: l'asse orizzontale nel sistema di coordinate bidimensionale utilizzato in ActionScript.
- Asse delle y: l'asse verticale nel sistema di coordinate bidimensionale utilizzato in ActionScript.

Operazioni con gli esempi contenuti nel capitolo

Molti degli esempi contenuti nel capitolo illustrano come eseguire calcoli o modificare valori; la maggior parte di essi include la chiamata appropriata alla funzione `trace()` per mostrare i risultati del codice. Per provare questi esempi, effettuare le operazioni seguenti:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.
4. Eseguire il programma selezionando Controllo > Prova filmato.

   Nel pannello Output vengono visualizzati i risultati della funzione `trace()` dell'esempio di codice.

Alcuni esempi mostrano come applicare trasformazioni agli oggetti di visualizzazione. I risultati di tali esempi vengono rappresentati visivamente invece che mediante un output di testo. Per provare questi esempi di trasformazione, effettuare le operazioni seguenti:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.
4. Creare un'istanza del simbolo di clip filmato sullo stage. Ad esempio, disegnare una forma, selezionarla, scegliere Elabora > Converti in simbolo e assegnare al simbolo un nome.
5. Con il clip filmato selezionato, nella finestra di ispezione Proprietà assegnare all'istanza un nome istanza. Questo nome deve corrispondere al nome utilizzato per l'oggetto di visualizzazione nell'esempio di codice, ad esempio, se il codice applica una trasformazione a un oggetto chiamato `myDisplayObject`, è necessario denominare `myDisplayObject` anche l'istanza del clip filmato.
6. Eseguire il programma selezionando Controllo > Prova filmato.

   Sullo schermo vengono visualizzati i risultati delle trasformazioni applicate all'oggetto secondo quanto specificato nell'esempio di codice.

Per ulteriori informazioni sulle tecniche per la prova degli esempi di codice, vedere "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Uso degli oggetti Point

Un oggetto Point definisce una coppia di coordinate cartesiane che rappresentano una posizione in un sistema di coordinate bidimensionale, dove *x* rappresenta l'asse orizzontale e *y* rappresenta l'asse verticale.

Per definire un oggetto Point è necessario impostarne le proprietà `x` e `y` come segue:

```
import flash.geom.*;
var pt1:Point = new Point(10, 20); // x == 10; y == 20
var pt2:Point = new Point();
pt2.x = 10;
pt2.y = 20;
```

Calcolo della distanza tra due punti

Usare il metodo `distance()` della classe Point per calcolare la distanza tra due punti in uno spazio di coordinate. Ad esempio, il codice seguente calcola la distanza tra i punti di registrazione dei due oggetti di visualizzazione, `circle1` e `circle2`, che risiedono nello stesso contenitore di oggetto di visualizzazione:

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
var pt2:Point = new Point(circle2.x, circle2.y);
var distance:Number = Point.distance(pt1, pt2);
```

Conversione degli spazi di coordinate

Se due oggetti di visualizzazione si trovano in diversi contenitori di oggetto di visualizzazione, è possibile che anche i loro spazi di coordinate differiscano. Usare il metodo `localToGlobal()` della classe DisplayObject per convertire le coordinate nello stesso spazio di coordinate (globale), quello dello Stage. Ad esempio, il codice seguente calcola la distanza tra i punti di registrazione dei due oggetti di visualizzazione, `circle1` e `circle2`, che risiedono in contenitori di oggetto di visualizzazione diversi:

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
pt1 = circle1.localToGlobal(pt1);
var pt2:Point = new Point(circle1.x, circle1.y);
pt2 = circle2.localToGlobal(pt2);
var distance:Number = Point.distance(pt1, pt2);
```

Analogamente, per calcolare la distanza del punto di registrazione di un oggetto di visualizzazione denominato `target` rispetto a un punto specifico dello Stage è possibile usare il metodo `localToGlobal()` della classe DisplayObject:

```
import flash.geom.*;
var stageCenter:Point = new Point();
stageCenter.x = this.stage.stageWidth / 2;
stageCenter.y = this.stage.stageHeight / 2;
var targetCenter:Point = new Point(target.x, target.y);
targetCenter = target.localToGlobal(targetCenter);
var distance:Number = Point.distance(stageCenter, targetCenter);
```

Spostamento di un oggetto di visualizzazione della distanza e inclinazione specificate

Usare il metodo `polar()` della classe Point per spostare un oggetto di visualizzazione in base al valore di distanza e inclinazione specificate. Ad esempio, il codice seguente sposta l'oggetto `myDisplayObject` di 100 pixel per 60 gradi:

```
import flash.geom.*;
var distance:Number = 100;
var angle:Number = 2 * Math.PI * (90 / 360);
var translatePoint:Point = Point.polar(distance, angle);
myDisplayObject.x += translatePoint.x;
myDisplayObject.y += translatePoint.y;
```

Altri impieghi della classe Point

Gli oggetti Point possono essere associati ai seguenti metodi e proprietà:

Classe	Metodi o proprietà	Descrizione
DisplayObjectContainer	`areInaccessibleObjectsUnderPoint()` `getObjectsUnderPoint()`	Restituisce un elenco di oggetti sotto un punto all'interno di un contenitore di oggetto di visualizzazione.
BitmapData	`hitTest()`	Permette di definire il pixel nell'oggetto BitmapData e il punto in cui si sta cercando un'attivazione.

Classe	Metodi o proprietà	Descrizione

BitmapData	`applyFilter()`
	`copyChannel()`
	`merge()`
	`paletteMap()`
	`pixelDissolve()`
	`threshold()`
		Definisce le posizioni dei rettangoli che definiscono le operazioni.

Matrix	`deltaTransformPoint()` `transformPoint()`	Permette di definire i punti a cui applicare una trasformazione.

Rectangle	`bottomRight`
	`size`
	`topLeft`
		Definisce queste proprietà.

Uso degli oggetti Rectangle

Un oggetto Rectangle definisce un'area rettangolare. Gli oggetti Rectangle hanno una posizione, definita dalle coordinate *x* e *y* dell'angolo superiore sinistro, e sono dotati delle proprietà `width` e `height`. Per definire le proprietà di un nuovo oggetto Rectangle richiamare la funzione di costruzione `Rectangle()`, come illustrato di seguito:

```
import flash.geom.Rectangle;
var rx:Number = 0;
var ry:Number = 0;
var rwidth:Number = 100;
var rheight:Number = 50;
var rect1:Rectangle = new Rectangle(rx, ry, rwidth, rheight);
```

Ridimensionamento e riposizionamento degli oggetti Rectangle

Gli oggetti Rectangle possono essere ridimensionati e riposizionati in vari modi.

Per riposizionare direttamente un oggetto Rectangle, è possibile modificarne le proprietà `x` e `y`. Questa operazione non influisce sui valori di altezza e larghezza dell'oggetto Rectangle.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.x = 20;
rect1.y = 30;
trace(rect1); // (x=20, y=30, w=100, h=50)
```

Come illustra il codice seguente, cambiando la proprietà `left` o `top` di un oggetto Rectangle, il rettangolo si sposta e le proprietà `x` e `y` ricalcano i valori delle proprietà `left` e `top`, rispettivamente. Tuttavia, poiché la posizione dell'angolo inferiore sinistro dell'oggetto Rectangle non viene cambiata, il rettangolo viene ridimensionato.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.left = 20;
rect1.top = 30;
trace(rect1); // (x=30, y=20, w=70, h=30)
```

Analogamente, come dimostra l'esempio che segue, se si modifica la proprietà `bottom` o `right` di un oggetto Rectangle, la posizione del suo angolo superiore sinistro non cambia e, di conseguenza, il rettangolo viene ridimensionato:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.right = 60;
trect1.bottom = 20;
trace(rect1); // (x=0, y=0, w=60, h=20)
```

In alternativa, si può riposizionare un oggetto Rectangle usando il metodo `offset()`, come illustrato di seguito:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.offset(20, 30);
trace(rect1); // (x=20, y=30, w=100, h=50)
```

Il metodo `offsetPt()` opera in modo simile, con la differenza che accetta come parametro un oggetto Point e non i valori di spostamento *x* e *y*.

Un oggetto Rectangle può essere ridimensionato anche usando il metodo `inflate()` che include due parametri: `dx` e `dy`. Il parametro `dx` rappresenta il numero di pixel rispetto al centro di cui si sposteranno i lati sinistro e destro del rettangolo; il parametro `dy` rappresenta il numero di pixel rispetto al centro di cui si sposteranno i lati superiore e inferiore dell'oggetto:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.inflate(6,4);
trace(rect1); // (x=-6, y=-4, w=112, h=58)
```

Il metodo `inflatePt()` opera in modo simile, con la differenza che accetta come parametro un oggetto Point e non i valori `dx` e `dy`.

Ricerca di unioni e intersezioni di oggetti Rectangle

Usare il metodo `union()` per individuare la zona del rettangolo formata dai perimetri di due rettangoli:

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(120, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.union(rect2)); // (x=0, y=0, w=220, h=160)
```

Usare il metodo `intersection()` per individuare la zona del rettangolo formata dall'area di sovrapposizione di due rettangoli:

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(80, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.intersection(rect2)); // (x=80, y=60, w=20, h=40)
```

Usare il metodo `intersects()` per scoprire se due rettangoli si intersecano. Inoltre, il metodo `intersects()` consente di scoprire se un oggetto di visualizzazione si trova in una determinata area dello Stage. Ad esempio, nel codice seguente si presume che lo spazio di coordinate del contenitore di oggetto di visualizzazione che contiene l'oggetto `circle` sia lo stesso di quello dello Stage. L'esempio illustra come usare il metodo `intersects()` per determinare se un oggetto di visualizzazione, `circle`, si interseca con aree particolari dello Stage, definite dagli oggetti Rectangle `target1` e `target2`:

```
import flash.display.*;
import flash.geom.Rectangle;
var circle:Shape = new Shape();
circle.graphics.lineStyle(2, 0xFF0000);
circle.graphics.drawCircle(250, 250, 100);
addChild(circle);
var circleBounds:Rectangle = circle.getBounds(stage);
var target1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(circleBounds.intersects(target1)); // false
var target2:Rectangle = new Rectangle(0, 0, 300, 300);
trace(circleBounds.intersects(target2)); // true
```

Analogamente, si può usare il metodo `intersects()` per scoprire se i perimetri dei rettangoli degli oggetti di visualizzazione si sovrappongono. È possibile usare il metodo `getRect()` della classe DisplayObject per includere lo spazio aggiuntivo che i tratti di un oggetto di visualizzazione potrebbero aggiungere a un'area di contorno.

Altri impieghi degli oggetti Rectangle

Gli oggetti Rectangle sono usati nei seguenti metodi e proprietà:

Classe	Metodi o proprietà	Descrizione
BitmapData	`applyFilter()`, `colorTransform()`, `copyChannel()`, `copyPixels()`, `draw()`, `fillRect()`, `generateFilterRect()`, `getColorBoundsRect()`, `getPixels()`, `merge()`, `paletteMap()`, `pixelDissolve()`, `setPixels()` e `threshold()`	Usato come tipo di alcuni parametri per la definizione di un'area dell'oggetto BitmapData.
DisplayObject	`getBounds()`, `getRect()`, `scrollRect`, `scale9Grid`	Usato come tipo di dati per la proprietà o come tipo di dati restituito.
PrintJob	`addPage()`	Usato per la definizione del parametro `printArea`.
Sprite	`startDrag()`	Usato per la definizione del parametro `bounds`.
TextField	`getCharBoundaries()`	Usato come tipo di valore restituito.
Transform	`pixelBounds`	Usato come tipo di dati.

Uso degli oggetti Matrix

La classe Matrix rappresenta una matrice di trasformazione che determina come mappare i punti da uno spazio di coordinate a un altro. È possibile eseguire diverse trasformazioni grafiche su un oggetto di visualizzazione impostando le proprietà di un oggetto Matrix, applicando l'oggetto Matrix alla proprietà `matrix` di un oggetto Transform e applicando quindi l'oggetto Transform come proprietà `transform` dell'oggetto di visualizzazione. Queste funzioni di trasformazione includono conversione (riposizionamento di *x* e *y*), rotazione, modifica in scala e inclinazione.

Definizione degli oggetti Matrix

Nonostante sia possibile definire una matrice modificando direttamente le proprietà (`a`, `b`, `c`, `d`, `tx`, `ty`) di un oggetto Matrix, è più semplice usare il metodo `createBox()`. Questo metodo comprende parametri che consentono di definire direttamente gli effetti di modifica in scala, rotazione e conversione della matrice risultante. Ad esempio, il codice seguente crea un oggetto Matrix con l'effetto di modificare in scala un oggetto orizzontalmente di 2.0, di modificarlo in scala verticalmente di 3.0, di ruotarlo di 45 gradi, di spostarlo (convertirlo) di 10 pixel a destra e di spostarlo di 20 pixel verso il basso:

```
var matrix:Matrix = new Matrix();
var scaleX:Number = 2.0;
var scaleY:Number = 3.0;
var rotation:Number = 2 * Math.PI * (45 / 360);
var tx:Number = 10;
var ty:Number = 20;
matrix.createBox(scaleX, scaleY, rotation, tx, ty);
```

È inoltre possibile modificare gli effetti di modifica in scala, rotazione e conversione di un oggetto Matrix usando i metodi `scale()`, `rotate()` e `translate()`. Si noti che i valori impostati per questi metodi si combinano con i valori già presenti nell'oggetto Matrix. Ad esempio, il codice seguente imposta un oggetto Matrix che modifica in scala un oggetto di 4 volte e lo ruota di 60 gradi, poiché i metodi `scale()` e `rotate()` sono chiamati due volte:

```
var matrix:Matrix = new Matrix();
var rotation:Number = 2 * Math.PI * (30 / 360); // 30°
var scaleFactor:Number = 2;
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);
matrix.scale(scaleX, scaleY);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

Per applicare un'inclinazione a un oggetto Matrix, modificarne le proprietà `b` o `c`. Modificando la proprietà `b` si imprime un'inclinazione verticale alla matrice; modificando la proprietà `c` si imprime un'inclinazione orizzontale. Il codice seguente inclina l'oggetto Matrix `myMatrix` in senso verticale di un fattore pari a 2:

```
var skewMatrix:Matrix = new Matrix();
skewMatrix.b = Math.tan(2);
myMatrix.concat(skewMatrix);
```

È possibile applicare una trasformazione di matrice alla proprietà `transform` di un oggetto di visualizzazione. Ad esempio, il codice seguente applica una trasformazione di matrice a un oggetto di visualizzazione denominato `myDisplayObject`:

```
var matrix:Matrix = myDisplayObject.transform.matrix;
var scaleFactor:Number = 2;
var rotation:Number = 2 * Math.PI * (60 / 360); // 60°
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

La prima riga imposta un oggetto Matrix sulla matrice di trasformazione esistente usata dall'oggetto di visualizzazione `myDisplayObject` (la proprietà `matrix` della proprietà `transformation` dell'oggetto di visualizzazione `myDisplayObject`). In questo modo, i metodi chiamati per la classe Matrix avranno un effetto cumulativo sulla posizione, la scala e la rotazione dell'oggetto di visualizzazione.

NOTA

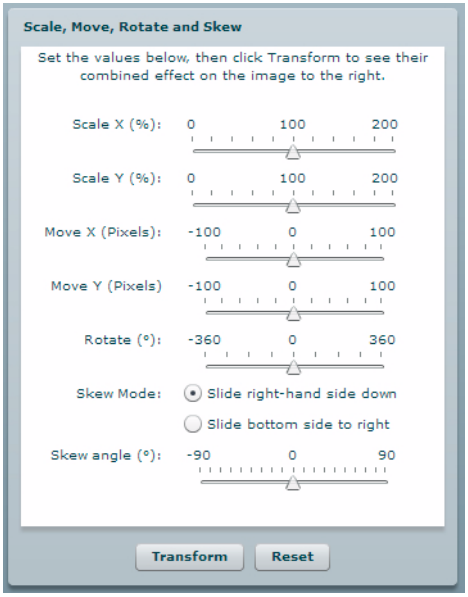
Anche la classe ColorTransform fa parte del pacchetto flash.geometry e viene usata per impostare la proprietà `colorTransform` di un oggetto Transform. Poiché non applica alcun tipo di trasformazione geometrica all'oggetto, il presente capitolo non la prende in esame. Per ulteriori informazioni, vedere la classe ColorTransform nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Esempio: Applicazione di una trasformazione di matrice a un oggetto di visualizzazione

L'applicazione di esempio DisplayObjectTransformer illustra una serie di impieghi della classe Matrix per trasformare un oggetto di visualizzazione nei modi seguenti:

- Rotazione dell'oggetto di visualizzazione
- Modifica in scala dell'oggetto di visualizzazione
- Conversione (riposizionamento) dell'oggetto di visualizzazione
- Inclinazione dell'oggetto di visualizzazione

L'applicazione comprende un'interfaccia che permette di modificare i parametri della trasformazione di matrice nel modo seguente:



Quando l'utente seleziona il pulsante Transform, l'applicazione applica la trasformazione impostata.

L'oggetto di visualizzazione originale e l'oggetto ruotato di -45° e ridotto del 50%

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione DisplayObjectTransformer si trovano nella cartella Samples/DisplayObjectTransformer. L'applicazione comprende i seguenti file:

File	Descrizione
DisplayObjectTransformer.mxml o DisplayObjectTransformer.fla	Il file principale dell'applicazione in Flash (FLA) o Flex (MXML)
com/example/programmingas3/geometry/MatrixTransformer.as	Classe che contiene metodi per applicare trasformazioni di matrice.
img/	Directory che contiene file immagine di esempio utilizzati dall'applicazione.

Definizione della classe MatrixTransformer

La classe MatrixTransformer comprende dei metodi statici che applicano trasformazioni geometriche degli oggetti Matrix.

Metodo transform()

Il metodo `transform()` comprende parametri per gli elementi seguenti:

- `sourceMatrix` - Matrice di partenza a cui il metodo applica le trasformazioni
- `xScale` e `yScale` - Fattori di scala di *x* e *y*
- `dx` e `dy` - Ammontare dello spostamento di *x* e *y*, espressi in pixel
- `rotation` - Ammontare della rotazione, espresso in gradi
- `skew` - Fattore di inclinazione, espresso in percentuale
- `skewType` - Direzione dell'inclinazione, può essere destra (`"right"`) o sinistra (`"left"`)

Il valore restituito è la matrice risultante.

Il metodo `transform()` chiama i seguenti metodi statici della classe:

- `skew()`
- `scale()`
- `translate()`
- `rotate()`

Ognuno restituisce la matrice di partenza con la trasformazione applicata.

Metodo skew()

Il metodo `skew()` inclina la matrice modificando le proprietà `b` e `c` della matrice stessa. Un parametro facoltativo, `unit`, stabilisce le unità in cui è definito l'angolo di inclinazione e, se necessario, il metodo converte il valore `angle` in radianti:

```
if (unit == "degrees")
{
  angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
  angle = Math.PI * 2 * angle / 100;
}
```

L'oggetto Matrix `skewMatrix` viene creato e modificato in base al valore di inclinazione. Inizialmente si tratta della matrice di identità:

```
var skewMatrix:Matrix = new Matrix();
```

Il parametro `skewSide` determina il lato a cui va applicata l'inclinazione. Se il parametro è impostato su `"right"`, il codice seguente imposta la proprietà `b` della matrice:

```
skewMatrix.b = Math.tan(angle);
```

Altrimenti, il lato inferiore viene inclinato modificando la proprietà `c` dell'oggetto Matrix nel modo seguente:

```
skewMatrix.c = Math.tan(angle);
```

L'inclinazione risultante viene applicata alla matrice esistente concatenando le due matrici, come illustra l'esempio seguente:

```
sourceMatrix.concat(skewMatrix);
return sourceMatrix;
```

Metodo scale()

Come illustra l'esempio seguente, il metodo `scale()` prima modifica il fattore di scala, se espresso in valore di percentuale, quindi usa il metodo `scale()` dell'oggetto Matrix:

```
if (percent)
{
  xScale = xScale / 100;
  yScale = yScale / 100;
}
sourceMatrix.scale(xScale, yScale);
return sourceMatrix;
```

Metodo translate()

Il metodo `translate()` semplicemente applica i fattori di conversione `dx` e `dy` chiamando il metodo `translate()` dell'oggetto Matrix come segue:

```
sourceMatrix.translate(dx, dy);
return sourceMatrix;
```

Metodo rotate()

Il metodo `rotate()` converte il fattore di rotazione specificato in radianti (se espresso in gradi o gradienti), quindi chiama il metodo `rotate()` dell'oggetto Matrix:

```
if (unit == "degrees")
{
  angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
  angle = Math.PI * 2 * angle / 100;
}
sourceMatrix.rotate(angle);
return sourceMatrix;
```

Chiamata al metodo MatrixTransformer.transform() dall'applicazione

L'applicazione comprende un'interfaccia utente tramite la quale l'utente può specificare i parametri di trasformazione. Tali parametri vengono quindi passati, insieme alla proprietà `matrix` della proprietà `transform` dell'oggetto di visualizzazione, al metodo `Matrix.transform()` nel modo seguente:

```
tempMatrix = MatrixTransformer.transform(tempMatrix,
                                         xScaleSlider.value,
                                         yScaleSlider.value,
                                         dxSlider.value,
                                         dySlider.value,
                                         rotationSlider.value,
                                         skewSlider.value,
                                         skewSide );
```

L'applicazione quindi applica il valore restituito alla proprietà `matrix` della proprietà `transform` dell'oggetto di visualizzazione, attivando in questo modo la trasformazione:

```
img.content.transform.matrix = tempMatrix;
```

CAPITOLO 14

Uso dell'API di disegno

14

Anche se le immagini e la grafica importate sono importanti, la funzionalità nota come API di disegno, che consente di disegnare linee e forme in ActionScript, offre la libertà di cominciare la creazione di un'applicazione con l'equivalente di una tela bianca su cui è possibile creare le immagini. La possibilità di utilizzare grafica personalizzata fornisce una vasta gamma di opportunità per le proprie applicazioni. Grazie alle tecniche illustrate in questo capitolo è possibile, tra le altre cose, sviluppare un programma di disegno, produrre grafica animata e interattiva oppure creare a livello di codice elementi personalizzati per l'interfaccia utente.

Sommario

Nozioni fondamentali sull'uso dell'API di disegno

Introduzione all'uso dell'API di disegno

API di disegno è il nome della funzionalità incorporata in ActionScript, che consente di creare elementi di grafica vettoriale (linee, curve, forme, riempimenti e gradienti) e di visualizzarli sullo schermo mediante ActionScript. Questa funzionalità è fornita dalla classe flash.display.Graphics. È possibile disegnare con ActionScript su qualunque istanza Shape, Sprite o MovieClip, mediante la proprietà `graphics` definita in ognuna di tali classi. (La proprietà `graphics` di ognuna delle classi è di fatto un'istanza della classe Graphics.)

Se ci si sta avvicinando per la prima volta al disegno mediante codice, la classe Graphics include diversi metodi che facilitano il disegno di forme comuni come cerchi, ellissi, rettangoli e rettangoli con angoli arrotondati. È possibile disegnarli come linee vuote o come forme piene. Se sono necessarie funzionalità più avanzate, la classe Graphics include anche dei metodi per disegnare linee e curve di Bézier quadratiche, che è possibile utilizzare in combinazione con le funzioni trigonometriche della classe Math per creare qualunque forma.

Operazioni comuni con l'API di disegno

L'API di disegno consente di effettuare le operazioni descritte di seguito:

- Definizione degli stili di linea e di riempimento per il disegno delle forme
- Disegno di linee rette e curve
- Uso dei metodi per disegnare forme come cerchi, ellissi e rettangoli
- Disegno con linee sfumate e riempimenti con gradiente
- Definizione di una matrice per la creazione di un gradiente
- Uso della trigonometria con l'API di disegno
- Incorporamento dell'API di disegno nell'animazione

Concetti e termini importanti

L'elenco di riferimento seguente contiene dei termini importanti che vengono citati in questo capitolo:

- Punto di ancoraggio: una delle due estremità di una curva di Bézier quadratica.
- Punto di controllo: il punto che definisce la direzione e la quantità di curvatura di una curva di Bézier quadratica. La linea curva non raggiunge mai il punto di controllo, tuttavia esegue una curvatura come se venisse attratta da esso.
- Spazio di coordinate: il grafico delle coordinate contenute in un oggetto di visualizzazione, su cui sono posizionati i relativi elementi secondari.
- Riempimento: la porzione interna uniforme di una forma che ha una linea riempita di colore oppure un'intera forma priva di contorno.
- Gradiente: un colore composto dalla transizione graduale da un colore a uno o più colori diversi (si contrappone al colore uniforme).
- Punto: una posizione singola in uno spazio di coordinate. Nel sistema di coordinate bidimensionale utilizzato in ActionScript, un punto è definito dalla sua posizione lungo l'asse x e l'asse y (le coordinate del punto).
- Curva di Bézier quadratica: un tipo di curva definito da una particolare formula matematica. In questo tipo di curva, la forma della curva viene calcolata in base alle posizioni dei punti di ancoraggio (le estremità della curva) e a un punto di controllo che definisce la quantità e la direzione della curvatura.
- Scala: le dimensioni di un oggetto rispetto alle proprie dimensioni originali. Quando si modifica in scala un oggetto si modificano le sue dimensioni allungandolo o riducendolo.
- Tratto: la porzione del contorno di una forma che ha una linea riempita di colore oppure le linee di una forma priva di riempimento.
- Conversione: modifica delle coordinate di un punto da uno spazio di coordinate a un altro.
- Asse X: l'asse orizzontale nel sistema di coordinate bidimensionale utilizzato in ActionScript.
- Asse Y: l'asse verticale nel sistema di coordinate bidimensionale utilizzato in ActionScript.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché questo capitolo descrive come disegnare contenuto visivo, la prova degli esempi di codice prevede l'esecuzione del codice e la visualizzazione dei risultati nel file SWF creato. Per provare gli esempi di codice:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.
4. Eseguire il programma selezionando Controllo > Prova filmato.

   I risultati dell'esempio di codice vengono visualizzati nel file SWF creato.

Nozioni fondamentali sulla classe Graphics

Ogni oggetto Shape, Sprite e MovieClip contiene una proprietà `graphics`, che è un'istanza della classe Graphics. La classe Graphics comprende proprietà e metodi per disegnare linee, riempimenti e forme. Se si desidera un oggetto di visualizzazione da utilizzare solo come tela per disegnare dei contenuti è possibile utilizzare un'istanza Shape. Un'istanza Shape offre prestazioni di disegno migliori rispetto agli altri oggetti di visualizzazione perché non ha il carico aggiuntivo delle funzionalità supplementari delle classi Sprite e MovieClip. Se si desidera un oggetto di visualizzazione su cui poter disegnare contenuti grafici ma che contenga anche altri oggetti di visualizzazione, è possibile utilizzare un'istanza Sprite. Per ulteriori informazioni su come determinare l'oggetto di visualizzazione più adatto alle varie operazioni, vedere "Scelta di una sottoclasse DisplayObject" a pagina 421.

Disegno di linee e curve

Tutto i disegni effettuati con un'istanza Graphics si basano sul disegno fondamentale di linee e curve. Di conseguenza, tutti il disegno creato con ActionScript deve essere eseguito seguendo la stessa serie di operazioni:

- Definizione degli stili di linea e di riempimento
- Impostazione della posizione di disegno iniziale
- Disegno di linee, curve e forme (facoltativamente spostando il punto di disegno)
- Se necessario, creazione di un riempimento (come ultima operazione)

Definizione degli stili di linea e di riempimento

Per disegnare con la proprietà `graphics` di un'istanza Shape, Sprite o MovieClip, innanzi tutto è necessario definire lo stile (dimensioni e colore della linea, colore del riempimento) da utilizzare nel disegno. Come accade quando si utilizzano gli strumenti di disegno in Adobe Flash CS3 Professional o in un altro programma di illustrazione, quando si utilizza ActionScript per disegnare si può utilizzare o meno un tratto o un colore di riempimento. L'aspetto del tratto viene specificato mediante il metodo `lineStyle()` o `lineGradientStyle()`. Per creare una linea uniforme, utilizzare il metodo `lineStyle()`. Quando si chiama questo metodo, i valori più comuni da specificare sono i primi tre parametri: spessore, colore e valore alfa della linea. Ad esempio, la riga di codice seguente indica all'istanza Shape di nome `myShape` di disegnare linee che sono spesse 2 pixel, sono di colore rosso (0x990000) e hanno il 75% di opacità:

```
myShape.graphics.lineStyle(2, 0x990000, .75);
```

Il valore predefinito per il parametro alfa è 1.0 (100%), pertanto è possibile lasciarlo disattivato se si desidera una linea completamente opaca. Il metodo `lineStyle()` accetta anche due parametri aggiuntivi per l'approssimazione dei pixel e la modalità scala; per ulteriori informazioni sull'utilizzo di questi parametri, vedere la descrizione del metodo `Graphics.lineStyle()` nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Per creare una linea sfumata, utilizzare il metodo `lineGradientStyle()`. Questo metodo viene descritto in "Creazione di linee sfumate e riempimenti con gradiente" a pagina 489.

Se si desidera creare una forma piena, chiamare il metodo `beginFill()`, `beginGradientFill()` o `beginBitmapFill()` prima di iniziare a disegnare. Il metodo `beginFill()` è il più elementare dei tre e accetta due parametri: il colore di riempimento e (facoltativamente) un valore alfa per il colore di riempimento. Ad esempio, se si desidera disegnare una forma con un riempimento uniforme verde, utilizzare il codice seguente (l'esempio presuppone che si stia disegnando su un oggetto di nome `myShape`):

```
myShape.graphics.beginFill(0x00FF00);
```

Quando si chiama un metodo di riempimento, prima di cominciare un nuovo riempimento viene implicitamente terminato qualunque eventuale riempimento precedente. Quando si chiama un metodo che specifica uno stile del tratto, il tratto precedente viene sostituito ma il riempimento specificato in precedenza non viene alterato, e viceversa.

Una volta specificati lo stile della linea e le proprietà del riempimento, è necessario determinare il punto iniziale del disegno. L'istanza Graphics ha un punto di disegno, simile alla punta di una penna su un foglio di carta. Il punto (qualsiasi) in cui si trova il punto di disegno indica dove comincerà la successiva azione di disegno. Inizialmente, un oggetto Graphics comincia con il punto di disegno in corrispondenza del punto 0, 0 nello spazio delle coordinate dell'oggetto su cui sta disegnando. Per cominciare a disegnare in un punto diverso, prima di chiamare uno dei metodi di disegno, chiamare il metodo `moveTo()`. Si tratta di un'operazione del tutto analoga a sollevare la punta della penna e spostarla in un altro punto.

Con il punto di disegno in posizione, si disegna mediante una serie di chiamate ai metodi di disegno `lineTo()` (per le linee rette) e `curveTo()` (per le linee curve).

SUGGERIMENTO	Mentre si disegna, è possibile chiamare il metodo `moveTo()` in qualsiasi momento per spostare il punto di disegno in un'altra posizione senza disegnare.

Mentre si disegna, se è stato specificato un colore di riempimento, è possibile specificare a Adobe Flash Player di chiudere il riempimento chiamando il metodo `endFill()`. Se non è stata disegnata una forma chiusa (in altre parole, se al momento della chiamata a `endFill()` il punto di disegno non si trova nel punto iniziale della forma), quando si chiama il metodo `endFill()` Flash Player chiude automaticamente la forma disegnando una linea retta dal punto di disegno corrente alla posizione specificata dalla chiamata più recente a `moveTo()`. Se è stato cominciato un riempimento e non è stato chiamato il metodo `endFill()`, chiamando `beginFill()` (o uno degli altri metodi di riempimento) il riempimento corrente viene chiuso e ne viene cominciato uno nuovo.

Disegno di linee rette

Quando si chiama il metodo `lineTo()`, l'oggetto Graphics disegna una linea retta dal punto di disegno corrente in corrispondenza delle coordinate specificate come due parametri nella chiamata al metodo, utilizzando lo stile di linea specificato. Ad esempio, la riga di codice seguente posiziona il punto di disegno in corrispondenza del punto 100, 100 e disegna una linea fino al punto 200, 200:

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.lineTo(200, 200);
```

L'esempio seguente disegna triangoli rossi e verdi con un'altezza di 100 pixel:

```
var triangleHeight:uint = 100;
var triangle:Shape = new Shape();

// red triangle, starting at point 0, 0
triangle.graphics.beginFill(0xFF0000);
triangle.graphics.moveTo(triangleHeight/2, 0);
triangle.graphics.lineTo(triangleHeight, triangleHeight);
triangle.graphics.lineTo(0, triangleHeight);
triangle.graphics.lineTo(triangleHeight/2, 0);

// green triangle, starting at point 200, 0
triangle.graphics.beginFill(0x00FF00);
triangle.graphics.moveTo(200 + triangleHeight/2, 0);
triangle.graphics.lineTo(200 + triangleHeight, triangleHeight);
triangle.graphics.lineTo(200, triangleHeight);
triangle.graphics.lineTo(200 + triangleHeight/2, 0);

this.addChild(triangle);
```

Disegno di curve

Il metodo `curveTo()` disegna una curva Bézier quadratica. Viene disegnato un arco che collega due punti (definiti punti di ancoraggio) piegandosi verso un terzo punto (definito punto di controllo). L'oggetto Graphics utilizza la posizione di disegno corrente come primo punto di ancoraggio. Quando si chiama il metodo `curveTo()`, si passano quattro parametri: le coordinate x e y del punto di controllo, seguite dalle coordinate x e y del secondo punto di ancoraggio. Ad esempio, il codice seguente disegna una curva che inizia nel punto 100, 100 e termina nel punto 200, 200. Dal momento che il punto di controllo si trova nel punto 175, 125, si ottiene una curva che si sposta verso destra e successivamente verso il basso:

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.curveTo(175, 125, 200, 200);
```

L'esempio seguente disegna degli oggetti circolari rossi e verdi con una larghezza e un'altezza di 100 pixel. Si noti che a causa della natura dell'equazione quadratica di Bézier, non si tratta di cerchi perfetti:

```
var size:uint = 100;
var roundObject:Shape = new Shape();

// red circular shape
roundObject.graphics.beginFill(0xFF0000);
roundObject.graphics.moveTo(size / 2, 0);
roundObject.graphics.curveTo(size, 0, size, size / 2);
roundObject.graphics.curveTo(size, size, size / 2, size);
roundObject.graphics.curveTo(0, size, 0, size / 2);
roundObject.graphics.curveTo(0, 0, size / 2, 0);
```

```
// green circular shape
roundObject.graphics.beginFill(0x00FF00);
roundObject.graphics.moveTo(200 + size / 2, 0);
roundObject.graphics.curveTo(200 + size, 0, 200 + size, size / 2);
roundObject.graphics.curveTo(200 + size, size, 200 + size / 2, size);
roundObject.graphics.curveTo(200, size, 200, size / 2);
roundObject.graphics.curveTo(200, 0, 200 + size / 2, 0);

this.addChild(roundObject);
```

Disegno di forme mediante metodi incorporati

Per comodità, quando si disegnano forme comuni come cerchi, ellissi, rettangoli e rettangoli arrotondati, ActionScript 3.0 fornisce dei metodi che disegnano tali forme automaticamente. Si tratta dei metodi `drawCircle()`, `drawEllipse()`, `drawRect()`, `drawRoundRect()` e `drawRoundRectComplex()` della classe Graphics. Questi metodi possono essere utilizzati al posto dei metodi `lineTo()` e `curveTo()`. Si noti tuttavia che è comunque necessario specificare gli stili di linea e di riempimento prima di chiamare questi metodi.

L'esempio seguente ricrea l'esempio dei quadrati rossi, verdi e blu con una larghezza e un'altezza di 100 pixel. Questo codice utilizza il metodo `drawRect()`, inoltre specifica che il colore di riempimento ha un valore alfa del 50% (0.5):

```
var squareSize:uint = 100;
var square:Shape = new Shape();
square.graphics.beginFill(0xFF0000, 0.5);
square.graphics.drawRect(0, 0, squareSize, squareSize);
square.graphics.beginFill(0x00FF00, 0.5);
square.graphics.drawRect(200, 0, squareSize, squareSize);
square.graphics.beginFill(0x0000FF, 0.5);
square.graphics.drawRect(400, 0, squareSize, squareSize);
square.graphics.endFill();
this.addChild(square);
```

In un oggetto Sprite o MovieClip, il contenuto del disegno creato con la proprietà `graphics` viene sempre visualizzato dietro tutti gli oggetti di visualizzazione secondari che sono contenuti nell'oggetto. Inoltre, il contenuto della proprietà `graphics` non è un oggetto di visualizzazione separato, pertanto non compare nell'elenco degli elementi secondari di un oggetto Sprite o MovieClip. Ad esempio, per l'oggetto Sprite seguente viene disegnato un cerchio con la relativa proprietà `graphics` e per esso è presente un oggetto TextField nell'elenco degli oggetti di visualizzazione secondari:

```
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0xFFCC00);
mySprite.graphics.drawCircle(30, 30, 30);
var label:TextField = new TextField();
label.width = 200;
label.text = "They call me mellow yellow...";
label.x = 20;
label.y = 20;
mySprite.addChild(label);
this.addChild(mySprite);
```

Si noti che l'oggetto TextField compare sopra il cerchio disegnato con l'oggetto Graphics.

Creazione di linee sfumate e riempimenti con gradiente

L'oggetto Graphics può disegnare tratti e riempimenti anche con gradienti anziché con colori uniformi. Per creare un tratto sfumato si utilizza il metodo `lineGradientStyle()`, mentre per il riempimento con gradiente si utilizza il metodo `beginGradientFill()`.

I due metodi accettano gli stessi parametri. I primi quattro sono richiesti: type, colors, alphas e ratios. I quattro parametri rimanenti sono opzionali ma utili nei casi di personalizzazione avanzata.

- Il primo parametro specifica il tipo di gradiente che si sta creando. I valori accettabili sono `GradientFill.LINEAR` o `GradientFill.RADIAL`.
- Il secondo parametro specifica l'array di valori di colore da utilizzare. In un gradiente lineare, i colori vengono disposti da sinistra a destra. In un gradiente radiale, i colori vengono disposti dall'interno verso l'esterno. L'ordine dei colori dell'array rappresenta l'ordine con cui i colori verranno disegnati nel gradiente.
- Il terzo parametro specifica i valori di trasparenza alfa per i colori corrispondenti nel parametro precedente.

- Il quarto parametro specifica la proporzione, o l'enfasi, di ciascun colore all'interno del gradiente. I valori accettabili sono compresi tra 0 e 255. Questi valori non rappresentano alcuna larghezza o altezza, bensì la posizione all'interno del gradiente; 0 rappresenta l'inizio del gradiente, 255 rappresenta la fine. L'array di proporzioni deve aumentare in sequenza e avere lo stesso numero di voci degli array di colori e di valori alfa specificati nel secondo e nel terzo parametro.

Benché sia opzionale, il quinto parametro (la matrice di trasformazione) viene utilizzata spesso, poiché fornisce un modo semplice e potente per controllare l'aspetto del gradiente. Questo parametro accetta un'istanza Matrix. Il modo più semplice per creare un oggetto Matrix per un gradiente consiste nell'utilizzare il metodo `createGradientBox()` della classe Matrix.

Definizione di un oggetto Matrix da usare con un gradiente

È possibile usare i metodi `beginGradientFill()` e `lineGradientStyle()` della classe flash.display.Graphics per la definizione di gradienti da usare nelle forme. Nella definizione di un gradiente, si fornisce una matrice come uno dei parametri di questi metodi.

Il modo più semplice per definire la matrice consiste nell'usare il metodo `createGradientBox()` della classe Matrix, che crea una matrice usata per definire il gradiente. È possibile definire la scala, la rotazione e la posizione del gradiente usando i parametri passati al metodo `createGradientBox()`. Il metodo `createGradientBox()` accetta i parametri seguenti:

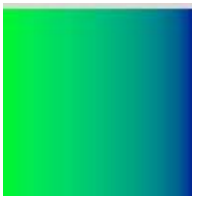
- Larghezza del riquadro del gradiente: la larghezza (espressa in pixel) dell'espansione del gradiente
- Altezza del riquadro del gradiente: l'altezza (espressa in pixel) dell'espansione del gradiente
- Rotazione del riquadro del gradiente: la rotazione (espressa in radianti) applicata al gradiente
- Conversione orizzontale: quantità di spazio (espressa in pixel) per cui un gradiente viene spostato in senso orizzontale
- Conversione verticale: quantità di spazio (espressa in pixel) per cui un gradiente viene spostato in senso verticale

Si consideri ad esempio un gradiente con le seguenti caratteristiche:

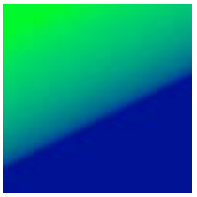
- `GradientType.LINEAR`
- Due colori, verde e blu, con gli array `ratios` impostati su `[0, 255]`
- `SpreadMethod.PAD`
- `InterpolationMethod.LINEAR_RGB`

Gli esempi seguenti contengono gradienti in cui il parametro `rotation` del metodo `createGradientBox()` differisce nel modo indicato e tutte le altre impostazioni rimangono invariate:

```
width = 100;
height = 100;
rotation = 0;
tx = 0;
ty = 0;
```



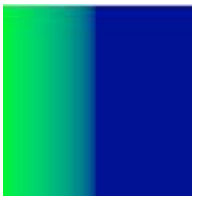
```
width = 100;
height = 100;
rotation = Math.PI/4; // 45°
tx = 0;
ty = 0;
```



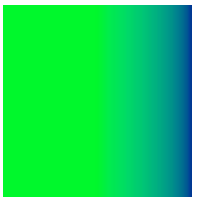
```
width = 100;
height = 100;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 0;
```

Gli esempi seguenti illustrano gli effetti su un gradiente lineare verde-blu in cui i parametri `rotation`, `tx` e `ty` del metodo `createGradientBox()` differiscono come indicato e tutte le altre impostazioni rimangono invariate:

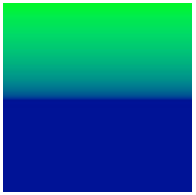
```
width = 50;
height = 100;
rotation = 0;
tx = 0;
ty = 0;
```



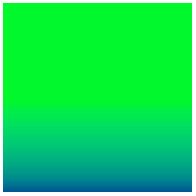
```
width = 50;
height = 100;
rotation = 0
tx = 50;
ty = 0;
```



```
width = 100;
height = 50;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 0;
```

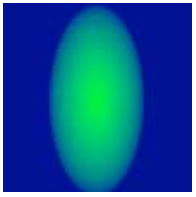


```
width = 100;
height = 50;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 50;
```



I parametri `width`, `height`, `tx` e `ty` del metodo `createGradientBox()` influenzano anche la dimensione e la posizione di un riempimento con gradiente *radiale*, come illustra l'esempio seguente:

```
width = 50;
height = 100;
rotation = 0;
tx = 25;
ty = 0;
```



Il codice seguente genera l'ultimo gradiente radiale illustrato:

```
import flash.display.Shape;
import flash.display.GradientType;
import flash.geom.Matrix;

var type:String = GradientType.RADIAL;
var colors:Array = [0x00FF00, 0x000088];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var spreadMethod:String = SpreadMethod.PAD;
var interp:String = InterpolationMethod.LINEAR_RGB;
var focalPtRatio:Number = 0;

var matrix:Matrix = new Matrix();
var boxWidth:Number = 50;
var boxHeight:Number = 100;
var boxRotation:Number = Math.PI/2; // 90°
var tx:Number = 25;
var ty:Number = 0;
matrix.createGradientBox(boxWidth, boxHeight, boxRotation, tx, ty);
```

```
var square:Shape = new Shape;
square.graphics.beginGradientFill(type,
                colors,
                alphas,
                ratios,
                matrix,
                spreadMethod,
                interp,
                focalPtRatio);
square.graphics.drawRect(0, 0, 100, 100);
addChild(square);
```

Si noti che la larghezza e l'altezza del riempimento con gradiente vengono determinate dalla larghezza e dall'altezza della matrice del gradiente anziché dalla larghezza o dall'altezza disegnate mediante l'oggetto Graphics. Quando si disegna mediante l'oggetto Graphics, si disegna ciò che esiste in corrispondenza di tali coordinate nella matrice del gradiente. Anche se si utilizza uno dei metodi shape di un oggetto Graphics come `drawRect()`, il gradiente non si allunga fino a coprire le dimensioni della forma che si sta disegnando: le dimensioni del gradiente devono essere specificate nella matrice del gradiente.

L'esempio seguente illustra la differenza visiva tra le dimensioni della matrice del gradiente e le dimensioni del disegno vero e proprio:

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(100, 40, 0, 0, 0);
myShape.graphics.beginGradientFill(GradientType.LINEAR, [0xFF0000,
  0x00FF00, 0x0000FF], [1, 1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 50, 40);
myShape.graphics.drawRect(0, 50, 100, 40);
myShape.graphics.drawRect(0, 100, 150, 40);
myShape.graphics.endFill();
this.addChild(myShape);
```

Questo codice disegna tre gradienti con lo stesso stile di riempimento, specificato con una distribuzione uniforme di rosso, verde e blu. I gradienti vengono disegnati mediante il metodo `drawRect()` con larghezze di rispettivamente di 50, 100 e 150 pixel. La matrice del gradiente specificata nel metodo `beginGradientFill()` viene creata con una larghezza di 100 pixel. Ciò significa che il primo gradiente comprende solo metà dello spettro del gradiente, il secondo lo comprende tutto e il terzo lo comprende tutto e ha 50 pixel aggiuntivi di blu che si estendono verso destra.

Il metodo `lineGradientStyle()` funziona in modo simile a `beginGradientFill()` con l'eccezione che, oltre a definire il gradiente, è necessario specificare lo spessore del tratto mediante il metodo `lineStyle()` prima di disegnare. Il codice seguente disegna un riquadro con un tratto sfumato rosso, verde e blu:

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(200, 40, 0, 0, 0);
myShape.graphics.lineStyle(5, 0);
myShape.graphics.lineGradientStyle(GradientType.LINEAR, [0xFF0000,
  0x00FF00, 0x0000FF], [1, 1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 200, 40);
this.addChild(myShape);
```

Per ulteriori informazioni sulla classe Matrix, vedere "Uso degli oggetti Matrix" a pagina 474.

Uso della classe Math con i metodi di disegno

Un oggetto Graphics disegna cerchi e quadrati ma è anche in grado di tracciare forme più complesse, in particolare quando i metodi di disegno vengono utilizzati in combinazione con le proprietà e i metodi della classe Math. La classe Math contiene costanti di interesse matematico comune, come `Math.PI` (circa 3.14159265...), una costante per il rapporto tra la circonferenza di un cerchio e il proprio diametro. Inoltre, contiene metodi per funzioni trigonometriche, tra cui `Math.sin()`, `Math.cos()` e `Math.tan()`. Disegnare forme utilizzando questi metodi e queste costanti consente di creare effetti visivi più dinamici, in particolare se impiegati in modo ripetuto o ricorsivo.

Molti metodi della classe Math prevedono misure circolari espresse in radianti anziché in gradi. La conversione tra questi due tipi di unità di misura rappresenta un uso comune della classe Math:

```
var degrees = 121;
var radians = degrees * Math.PI / 180;
trace(radians) // 2.111848394913139
```

L'esempio seguente crea una sinusoide e un'onda coseno per evidenziare la differenza tra i metodi `Math.sin()` e `Math.cos()` per un determinato valore.

```
var sinWavePosition = 100;
var cosWavePosition = 200;
var sinWaveColor:uint = 0xFF0000;
var cosWaveColor:uint = 0x00FF00;
var waveMultiplier:Number = 10;
var waveStretcher:Number = 5;

var i:uint;
for(i = 1; i < stage.stageWidth; i++)
{
  var sinPosY:Number = Math.sin(i / waveStretcher) * waveMultiplier;
  var cosPosY:Number = Math.cos(i / waveStretcher) * waveMultiplier;

  graphics.beginFill(sinWaveColor);
  graphics.drawRect(i, sinWavePosition + sinPosY, 2, 2);
  graphics.beginFill(cosWaveColor);
  graphics.drawRect(i, cosWavePosition + cosPosY, 2, 2);
}
```

Animazione mediante l'API di disegno

Un vantaggio offerto dalla creazione di contenuti con l'API di disegno consiste nel fatto che non si è limitati dal fatto di dover posizionare i contenuti una sola volta. Ciò che si disegna può essere cambiato mantenendo o modificando le variabili utilizzate per il disegno. È possibile trasmettere l'effetto dell'animazione modificando le variabili e ridisegnando, sia rispetto a un intervallo di fotogrammi sia utilizzando un timer.

Ad esempio, il codice seguente modifica la visualizzazione a ogni fotogramma (intercettando l'evento `Event.ENTER_FRAME`), incrementando il conteggio dei gradi corrente, e specifica all'oggetto Graphics di cancellare e ridisegnare con la posizione aggiornata.

```
stage.frameRate = 31;

var currentDegrees:Number = 0;
var radius:Number = 40;
var satelliteRadius:Number = 6;

var container:Sprite = new Sprite();
container.x = stage.stageWidth / 2;
container.y = stage.stageHeight / 2;
addChild(container);
var satellite:Shape = new Shape();
container.addChild(satellite);
```

```
addEventListener(Event.ENTER_FRAME, doEveryFrame);

function doEveryFrame(event:Event):void
{
  currentDegrees += 4;
  var radians:Number = getRadians(currentDegrees);
  var posX:Number = Math.sin(radians) * radius;
  var posY:Number = Math.cos(radians) * radius;
  satellite.graphics.clear();
  satellite.graphics.beginFill(0);
  satellite.graphics.drawCircle(posX, posY, satelliteRadius);
}
function getRadians(degrees:Number):Number
{
    return degrees * Math.PI / 180;
}
```

Per produrre un risultato significativamente diverso, è possibile sperimentare modificando le variabili seed iniziali all'inizio del codice, `currentDegrees`, `radius` e `satelliteRadius`. Ad esempio, provare a ridurre la variabile radius e/o a incrementare la variabile totalSatellites. Si tratta solo di un esempio di come l'API di disegno è in grado di creare una visualizzazione la cui complessità nasconde in realtà una semplicità di creazione.

Esempio: Algorithmic Visual Generator

L'esempio Algorithmic Visual Generator disegna dinamicamente sullo stage diversi "satelliti", cioè dei cerchi che si muovono seguendo un'orbita circolare. Le operazioni illustrate in questo esempio sono le seguenti:

- Uso dell'API di disegno per disegnare una forma di base con aspetti dinamici
- Collegamento dell'interazione dell'utente con le proprietà utilizzate in un disegno
- Trasmissione dell'effetto dell'animazione mediante la cancellazione dello stage in ciascun fotogramma e il ridisegno

Nell'esempio della sezione precedente veniva animato un "satellite" solitario utilizzando l'evento `Event.ENTER_FRAME`. Questo esempio espande l'esempio precedente, creando un pannello di controllo con una serie di dispositivi di scorrimento che aggiornano immediatamente la visualizzazione di diversi satelliti. Questo esempio formalizza il codice in classi esterne e racchiude il codice di creazione del satellite in un ciclo, memorizzando un riferimento a ogni satellite in un array `satellites`.

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione si trovano nella cartella Samples/AlgorithmicVisualGenerator. La cartella contiene i file seguenti:

File	Descrizione
AlgorithmicVisualGenerator.fla	Il file dell'applicazione principale in Flash (FLA).
com/example/programmingas3/algorithmic/AlgorithmicVisualGenerator.as	La classe che fornisce la funzionalità principale dell'applicazione, tra cui il disegno di satelliti sullo stage e la risposta agli eventi inviati dal pannello di controllo per aggiornare le variabili che influenzano il disegno dei satelliti.
com/example/programmingas3/algorithmic/ControlPanel.as	Una classe che gestisce l'interazione dell'utente con diversi dispositivi di scorrimento e, quando ciò avviene, l'invio di eventi.
com/example/programmingas3/algorithmic/Satellite.as	Una classe che rappresenta l'oggetto di visualizzazione che ruota in base a un'orbita attorno a un punto centrale e contiene le proprietà correlate al proprio stato di disegno.

Impostazione dei listener

L'applicazione crea innanzi tutto tre listener. Il primo intercetta un evento inviato dal pannello di controllo e che indica che è necessario ricreare i satelliti. Il secondo intercetta le modifiche apportate alle dimensioni dello stage del file SWF. Il terzo intercetta ogni fotogramma che scorre nel file SWF e indica di ridisegnare mediante la funzione `doEveryFrame()`.

Creazione dei satelliti

Una volta impostati i listener, viene chiamata la funzione `build()`. Questa funzione chiama dapprima la funzione `clear()`, che svuota l'array `satellites` e cancella tutti gli eventuali disegni precedenti sullo stage. Questa operazione è necessaria, dal momento che la funzione `build()` potrebbe essere richiamata ogni volta che il pannello di controllo invia un evento a tale scopo, ad esempio quando sono state modificate le impostazioni del colore. In tal caso, i satelliti devono essere rimossi e ricreati.

La funzione crea quindi i satelliti, impostando le proprietà iniziali necessarie per la creazione, quali la variabile `position`, che comincia in una posizione casuale all'interno dell'orbita, e la variabile `color`, che in questo esempio non viene modificata dopo che il satellite è stato creato.

Quando viene creato un satellite, viene aggiunto un riferimento a esso nell'array `satellites`. Quando viene chiamata la funzione `doEveryFrame()`, viene aggiornata per tutti i satelliti in questo array.

Aggiornamento della posizione di un satellite

La funzione `doEveryFrame()` è il cuore del processo di animazione dell'applicazione. Viene chiamata per ogni fotogramma, con una velocità identica alla frequenza di fotogrammi del file SWF. Dal momento che le variabili del disegno cambiano leggermente, si ottiene l'effetto del movimento.

La funzione dapprima cancella tutti i disegni precedenti e ridisegna lo sfondo. Quindi, scorre i contenitori di tutti i satelliti e incrementa la proprietà `position` di ciascun satellite, e aggiorna le proprietà `radius` e `orbitRadius` eventualmente modificate dall'interazione dell'utente con il pannello di controllo. Infine, il satellite viene aggiornato nella nuova posizione chiamando il metodo `draw()` della classe Satellite.

Si noti che il contatore `i` viene incrementato solo fino alla variabile `visibleSatellites`, poiché se l'utente ha limitato la quantità di satelliti visualizzati mediante il pannello di controllo, i satelliti rimanenti nel ciclo non devono essere ridisegnati bensì nascosti. Questa situazione si verifica in un ciclo che segue immediatamente il ciclo responsabile per il disegno.

Quando la funzione doEveryFrame() viene completata, il numero di `visibleSatellites` viene aggiornato in posizione sullo schermo.

Risposta all'interazione dell'utente

L'interazione dell'utente ha luogo mediante il pannello di controllo, che è gestito dalla classe ControlPanel. Questa classe imposta un listener insieme ai singoli valori minimo, massimo e predefinito di ciascun indicatore di scorrimento. Quando l'utente muove questi indicatori di scorrimento, viene chiamata la funzione `changeSetting()`. La funzione aggiorna le proprietà del pannello di controllo. Se la modifica richiede una ricreazione dello schermo, viene inviato un evento che viene gestito nel file principale dell'applicazione. Quando le impostazioni del pannello di controllo cambiano, la funzione `doEveryFrame()` disegna ogni satellite con le variabili aggiornate.

Ulteriori personalizzazioni

Questo esempio è solo uno schema essenziale di come generare elementi visivi mediante l'API di disegno. Utilizza un numero relativamente basso di righe di codice per creare un'esperienza interattiva che sembra invece piuttosto complessa. Tuttavia, anche questo esempio può essere esteso ricorrendo a modifiche di lieve entità. Ad esempio:

- La funzione `doEveryFrame()` può incrementare il valore del colore del satellite.
- La funzione `doEveryFrame()` può ridurre o espandere il raggio del satellite nel corso del tempo.
- Il raggio del satellite non deve necessariamente essere circolare; ad esempio, può utilizzare una classe Math per spostarsi in base a una sinusoide.
- I satelliti possono utilizzare il rilevamento a zone con altri satelliti.

L'API di disegno può essere utilizzata come alternativa alla creazione di effettivi visivi nell'ambiente di creazione di Flash e per disegnare forme di base in fase di runtime. Ma può anche essere usata per creare effetti visivi con varietà e aree di validità impossibili da creare a mano. Grazie all'API di disegno e a un po' di matematica, un autore ActionScript può dare vita a moltissime creazioni decisamente imprevedibili.

CAPITOLO 15

Filtraggio degli oggetti di visualizzazione

15

Per lungo tempo, l'applicazione di effetti di filtro alle immagini bitmap è stato monopolio di software specializzati nella modifica delle immagini quali Adobe Photoshop® e Adobe Fireworks®. Tuttavia, ActionScript 3.0 include il pacchetto flash.filters, che contiene una serie di classi di filtro per bitmap per consentire agli sviluppatori di applicare filtri alle bitmap a livello di codice e visualizzare oggetti di visualizzazione per ottenere molti degli effetti disponibili nelle applicazioni di manipolazione grafica.

Sommario

Elementi fondamentali del filtraggio degli oggetti di visualizzazione

Introduzione al filtraggio degli oggetti di visualizzazione

Uno dei metodi per aggiungere un tocco di ricercatezza a un'applicazione consiste nell'aggiungere dei semplici effetti grafici, ad esempio un'ombra esterna dietro una foto per creare l'illusione della tridimensionalità oppure un bagliore attorno a un pulsante per evidenziare che è attivo. ActionScript 3.0 include nove filtri applicabili a qualunque oggetto di visualizzazione o istanza BitmapData. Si tratta sia di filtri di base, quali l'ombra esterna e il bagliore, sia di filtri complessi per la creazione di vari effetti, quali il filtro mappa di spostamento e il filtro di convoluzione.

Operazioni di filtraggio comuni

L'uso dei filtri in ActionScript consente di effettuare le operazioni descritte di seguito:

- Creazione di un filtro
- Applicazione di un filtro a un oggetto di visualizzazione
- Applicazione di un filtro ai dati immagine in un'istanza BitmapData
- Rimozione dei filtri da un oggetto
- Creazione di vari effetti filtro, quali:
  - Bagliore
  - Sfocatura
  - Ombra esterna
  - Precisione
  - Spostamento
  - Rilevamento dei bordi
  - Rilievi
  - e altri effetti

Concetti e termini importanti

L'elenco di riferimento seguente contiene dei termini importanti che vengono citati in questo capitolo:

- Smussatura: un bordo creato schiarendo i pixel su due lati e scurendo i pixel sui due lati opposti, al fine di creare l'effetto di un bordo tridimensionale utilizzato generalmente per i pulsanti sollevati o rientrati e altri elementi grafici simili.
- Convoluzione: distorsione dei pixel in un'immagine mediante la combinazione del valore di ciascun pixel con i valori di alcuni o tutti i pixel vicini, utilizzando rapporti diversi.
- Spostamento: spostamento dei pixel di un'immagine in una nuova posizione.
- Matrice: una griglia di numeri utilizzata per eseguire alcuni calcoli matematici mediante l'applicazione dei numeri nella griglia a vari valori e la successiva combinazione dei risultati.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché questo capitolo descrive come creare e manipolare contenuto visivo, la prova degli esempi di codice prevede l'esecuzione del codice e la visualizzazione dei risultati nel file SWF creato. Quasi tutti gli esempi creano contenuto utilizzando l'API di disegno oppure caricano immagini a cui vengono applicati filtri.

Per provare il codice contenuto in questo capitolo:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare il codice nel riquadro dello script.
4. Eseguire il programma selezionando Controllo > Prova filmato.

   I risultati dell'esempio di codice vengono visualizzati nel file SWF creato.

Quasi tutti gli esempi di codice includono codice che crea un'immagine bitmap, pertanto è possibile provare il codice direttamente senza la necessità di fornire contenuto bitmap. In alternativa, è possibile modificare gli esempi in modo che carichino immagini personali e le utilizzino al posto degli esempi.

Creazione e applicazione di filtri

I filtri consentono di applicare alle bitmap e agli oggetti di visualizzazione una vasta gamma di effetti (ad esempio, ombre esterne, smussature e sfocature). Ogni filtro viene definito come classe, pertanto l'applicazione di filtri consiste nella creazione di istanze di oggetti filtro, un'operazione che non è diversa dalla creazione di qualunque altro oggetto. Una volta creata un'istanza di un oggetto filtro, può essere facilmente applicata a un oggetto di visualizzazione mediante la proprietà `filters` dell'oggetto o, nel caso di un oggetto BitmapData, mediante il metodo `applyFilter()`.

Creazione di un nuovo filtro

Per creare un nuovo oggetto filtro, è sufficiente chiamare il metodo di costruzione della classe di filtri scelta. Ad esempio, per creare un nuovo oggetto DropShadowFilter, utilizzare il codice seguente:

```
import flash.filters.DropShadowFilter;
var myFilter:DropShadowFilter = new DropShadowFilter();
```

Anche se non viene mostrato in questo esempio, la funzione di costruzione `DropShadowFilter()` (come tutte le funzioni di costruzione delle classi di filtri) accetta diversi parametri opzionali che è possibile utilizzare per personalizzare l'aspetto dell'effetto filtro.

Applicazione di un filtro

Una volta creato un oggetto filtro, è possibile applicarlo a un oggetto di visualizzazione o a un oggetto BitmapData; il modo in cui viene applicato dipende dall'oggetto di destinazione.

Applicazione di un filtro a un oggetto di visualizzazione

Quando si applica un filtro a un oggetto di visualizzazione, si utilizza la proprietà `filters`. La proprietà `filters` di un oggetto di visualizzazione è un'istanza Array, i cui elementi sono gli oggetti di filtro applicati all'oggetto di visualizzazione. Per applicare un solo filtro a un oggetto di visualizzazione, creare l'istanza del filtro, aggiungerla a un'istanza Array e assegnare tale oggetto Array alla proprietà `filters` dell'oggetto di visualizzazione:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.filters.DropShadowFilter;

// Crea un oggetto bitmapData e ne esegue il rendering sullo schermo.
var myBitmapData:BitmapData = new BitmapData(100,100,false,0xFFFF3300);
var myDisplayObject:Bitmap = new Bitmap(myBitmapData);
addChild(myDisplayObject);

// Crea un'istanza DropShadowFilter.
var dropShadow:DropShadowFilter = new DropShadowFilter();

// Crea l'array filters, aggiungendo il filtro all'array passandolo come
// parametro alla funzione di costruzione Array.
var filtersArray:Array = new Array(dropShadow);

// Assegna l'array filters all'oggetto di visualizzazione per applicare
// il filtro.
myDisplayObject.filters = filtersArray;
```

Se si desidera assegnare più filtri all'oggetto, è sufficiente aggiungere tutti i filtri all'istanza Array prima di assegnarla alla proprietà `filters`. È possibile aggiungere più oggetti a un array passandoli come parametri alla relativa funzione di costruzione. Ad esempio, questo codice applica un filtro smussatura e un filtro bagliore all'oggetto di visualizzazione creato in precedenza:

```
import flash.filters.BevelFilter;
import flash.filters.GlowFilter;

// Crea i filtri e li aggiunge a un array.
var bevel:BevelFilter = new BevelFilter();
var glow:GlowFilter = new GlowFilter();
var filtersArray:Array = new Array(bevel, glow);

// Assegna l'array filters all'oggetto di visualizzazione per applicare
// il filtro.
myDisplayObject.filters = filtersArray;
```

NOTA

Quando si crea l'array che contiene i filtri, è possibile crearlo mediante la funzione di costruzione `new Array()` (come mostrato negli esempi precedenti) oppure è possibile utilizzare la sintassi letterale di Array, racchiudendo i filtri tra parentesi quadre (`[]`). Ad esempio, la riga di codice seguente:

```
var filters:Array = new Array(dropShadow, blur);
```

ha lo stesso effetto della riga di codice seguente:

```
var filters:Array = [dropShadow, blur];
```

Se si applicano più filtri agli oggetti di visualizzazione, vengono applicati in modo cumulativo e in sequenza. Ad esempio, se un array filters contiene due elementi, un filtro di smussatura aggiunto per primo e un filtro ombra esterna aggiunto per secondo, il filtro ombra esterno viene applicato sia al filtro smussatura che all'oggetto di visualizzazione. Ciò avviene in virtù del fatto che il filtro ombra esterna occupa la seconda posizione nell'array filters. Se si desidera applicare i filtri in un modo non cumulativo, è necessario applicare ogni filtro a una nuova copia dell'oggetto di visualizzazione.

Se si assegnano solo pochi filtri a un oggetto di visualizzazione, è possibile creare l'istanza del filtro e assegnarla all'oggetto in una sola istruzione. Ad esempio, la riga di codice seguente applica un filtro sfocatura a un oggetto di visualizzazione denominato `myDisplayObject`:

```
myDisplayObject.filters = [new BlurFilter()];
```

Il codice precedente crea un'istanza Array mediante la sintassi letterale di Array (parentesi quadre), crea una nuova istanza BlurFilter come elemento dell'Array e assegna tale Array alla proprietà `filters` dell'oggetto di visualizzazione denominato `myDisplayObject`.

Rimozione dei filtri da un oggetto di visualizzazione

Rimuovere tutti i filtri da un oggetto di visualizzazione è un'operazione semplice che comporta l'assegnazione di un valore null alla proprietà `filters`:

```
myDisplayObject.filters = null;
```

Se sono stati applicati più filtri a un oggetto e si desidera rimuovere solo un filtro, è necessario eseguire diverse operazioni per modificare l'array della proprietà `filters`. Per ulteriori informazioni, vedere "Modifica dei filtri in fase di runtime" a pagina 507.

Applicazione di un filtro a un oggetto BitmapData

Per applicare un filtro a un oggetto BitmapData è necessario utilizzare il metodo `applyFilter()` dell'oggetto BitmapData:

```
myBitmapData.applyFilter(sourceBitmapData);
```

Il metodo `applyFilter()` applica un filtro a un oggetto BitmapData di origine, producendo una nuova immagine filtrata. Questo metodo non modifica l'immagine di origine; piuttosto, il risultato dell'applicazione del filtro all'immagine originale viene memorizzato nell'istanza BitmapData su cui viene chiamato il metodo `applyFilter()`.

Funzionamento dei filtri

Il filtraggio degli oggetti di visualizzazione funziona mediante la memorizzazione nella cache di una copia dell'oggetto originale sotto forma di bitmap trasparente.

Una volta applicato un filtro a un oggetto di visualizzazione, Adobe Flash Player memorizza l'oggetto nella cache sotto forma di bitmap fintanto che per l'oggetto è presente un elenco di filtri valido. Questa bitmap di origine viene quindi utilizzata come immagine di origine per tutti gli effetti filtro applicati in seguito.

Ogni oggetto di visualizzazione di solito contiene due bitmap: una con l'oggetto di visualizzazione di origine non filtrato originale e l'altra per l'immagine finale dopo l'applicazione del filtro. L'immagine finale viene utilizzata al momento del rendering. Se l'oggetto di visualizzazione non viene modificato, l'immagine finale non richiede alcun aggiornamento.

Problemi potenziali nelle operazioni con i filtri

Quando si utilizzano i filtri è opportuno tenere in considerazione una serie di potenziali fonti di confusione o problemi. Tali situazioni vengono descritte nelle sezioni che seguono.

Memorizzazione nella cache di filtri e bitmap

Per applicare un filtro a un oggetto di visualizzazione, è necessario attivare la memorizzazione nella cache per l'oggetto. Quando si applica un filtro a un oggetto di visualizzazione la cui proprietà `cacheAsBitmap` è impostata su `false`, Flash Player imposta automaticamente il valore della proprietà `cacheAsBitmap` dell'oggetto su `true`. Se successivamente si rimuovono tutti i filtri dall'oggetto di visualizzazione, la proprietà `cacheAsBitmap` ritorna sull'ultimo valore sul quale era stata impostata.

Modifica dei filtri in fase di runtime

Se a un oggetto di visualizzazione sono già applicati dei filtri, non è possibile aggiungere altri filtri all'array della proprietà `filters`. Per aggiungere o modificare il set di filtri applicati, creare una copia dell'intero array filters e apportare le modifiche a questo array temporaneo. Quindi, riassegnare l'array alla proprietà `filters` dell'oggetto di visualizzazione per applicare i filtri all'oggetto, come dimostrato nell'esempio seguente. Inizialmente, viene applicato un filtro bagliore all'oggetto di visualizzazione denominato `myDisplayObject`; quindi, quando viene fatto clic sull'oggetto di visualizzazione, viene chiamata la funzione `addFilters()`. In questa funzione, vengono applicati due filtri aggiuntivi a myDisplayObject:

```
import flash.events.MouseEvent;
import flash.filters.*;

myDisplayObject.filters = [new GlowFilter()];

function addFilters(event:MouseEvent):void
{
  // Crea una copia dell'array filters.
  var filtersCopy:Array = myDisplayObject.filters;

  // Apporta le modifiche desiderate ai filtri (in questo caso, aggiunge
  // i filtri).
  filtersCopy.push(new BlurFilter());
  filtersCopy.push(new DropShadowFilter());

  // Applica le modifiche riassegnando l'array alla proprietà filters.
  myDisplayObject.filters = filtersCopy;
}

myDisplayObject.addEventListener(MouseEvent.CLICK, addFilters);
```

Filtri e trasformazioni degli oggetti

Nella superficie relativa agli obiettivi del rilevamento per zone (operazione che consente di determinare se un'istanza si sovrappone o si interseca con un'altra) non sono presenti regioni filtrate (ad esempio, ombra esterna) al di fuori del rettangolo del riquadro di delimitazione dell'oggetto di visualizzazione. Poiché i metodi di rilevamento per zone della classe DisplayObject si basano sui vettori, non è possibile eseguirli sulla bitmap risultante.
Ad esempio, se si applica un filtro smussatura all'istanza di un pulsante, non è possibile eseguire il rilevamento per zone sulla parte smussata dell'istanza.

La modifica in scala, la rotazione e l'inclinazione non sono supportate da filtri. Se l'oggetto di visualizzazione filtrato viene modificato in scala (`scaleX` e `scaleY` non sono al 100%), l'effetto di filtro non viene modificato in scala con l'istanza. In altre parole, la forma originale dell'istanza viene ruotata, modificata in scala o inclinata; tuttavia, il filtro non viene ruotato, modificato in scala o inclinato con l'istanza.

È possibile animare un'istanza con un filtro per creare effetti realistici o nidificare istanze e utilizzare la classe BitmapData per animare i filtri in modo da ottenere l'effetto specificato.

Filtri e oggetti Bitmap

Quando si applica un filtro a un oggetto BitmapData, la proprietà `cacheAsBitmap` viene automaticamente impostata su `true`. In tal modo, il filtro viene di fatto applicato alla copia dell'oggetto e non all'originale.

Questa copia viene quindi posizionata nella vista principale (sopra l'oggetto originale) più vicino possibile al pixel più vicino. Se i limiti della bitmap originale cambiano, la copia filtrata viene ricreata anziché essere allungata o distorta.

Se tutti i filtri di un oggetto di visualizzazione vengono rimossi, la proprietà `cacheAsBitmap` viene reimpostata sul valore precedente all'applicazione del filtro.

Filtri di visualizzazione disponibili

ActionScript 3.0 include nove classi di filtro applicabili a qualunque oggetto di visualizzazione o BitmapData.

- Filtro smussatura (classe BevelFilter)
- Filtro sfocatura (classe BlurFilter)
- Filtro ombra esterna (classe DropShadowFilter)
- Filtro bagliore (classe GlowFilter)
- Filtro smussatura con gradiente (classe GradientBevelFilter)
- Filtro bagliore con gradiente (classe GradientGlowFilter)
- Filtro matrice colore (classe ColorMatrixFilter)
- Filtro convoluzione (classe ConvolutionFilter)
- Filtro mappa di spostamento (classe DisplacementMapFilter)

I primi sei filtri sono filtri semplici utilizzabili per creare un effetto specifico e applicare a esso alcune personalizzazioni. Questi sei filtri possono essere applicati mediante ActionScript oppure in Adobe Flash CS3 Professional mediante il pannello Filtri. Di conseguenza, anche se si applicano dei filtri mediante ActionScript, se si dispone dello strumento di creazione di Flash è possibile utilizzare l'interfaccia visiva per provare rapidamente i diversi filtri e le diverse impostazioni per determinare come creare l'effetto desiderato.

Gli ultimi tre filtri sono disponibili solo in ActionScript. Questi filtri (matrice colore, convoluzione e mappa di spostamento) sono molto più flessibili in virtù dei tipi di effetti che possono contribuire a creare; anziché essere ottimizzati per un solo effetto, forniscono un elevato livello di potenza e flessibilità. Ad esempio, selezionando diversi valori per la relativa matrice, il filtro convoluzione può essere utilizzato per creare effetti come le sfocature, i rilievi, l'aumento della nitidezza, il rilevamento dei bordi colorati, le trasformazioni e molti altri ancora.

Ognuno di tutti questi filtri può essere personalizzato mediante le relative proprietà. In genere sono disponibili due opzioni per l'impostazione delle proprietà di un filtro. Tutti i filtri consentono di impostare le proprietà passando dei valori di parametro alla funzione di costruzione dell'oggetto filtro. In alternativa, che si impostino o meno le proprietà del filtro passando dei parametri, è possibile regolare i filtri in un secondo momento impostando i valori delle proprietà dell'oggetto filtro. Nella maggior parte degli esempi di codice presentati in questa sezione le proprietà vengono impostate direttamente, ai fini di una maggiore comprensibilità. Ciononostante, è possibile ottenere lo stesso risultato con un numero inferiore di righe di codice passando i valori come parametri alla funzione di costruzione dell'oggetto filtro. Per maggiori dettagli sulle caratteristiche, sulle proprietà e sui parametri delle funzioni di costruzione di ogni filtro, vedere gli esempi del pacchetto flash.filters nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Filtro smussatura

La classe BevelFilter consente di aggiungere un effetto di smussatura tridimensionale all'oggetto filtrato. Questo filtro fa apparire gli spigoli e i bordi vivi dell'oggetto come se fossero stati cesellati, o smussati.

Le proprietà della classe BevelFilter consentono di personalizzare l'aspetto della smussatura. È possibile impostare i colori di evidenziazione e d'ombra, le sfocature dei bordi della smussatura, gli angoli della smussatura e la posizione del bordo della smussatura; è persino possibile creare un effetto di foratura.

Nell'esempio seguente viene caricata un'immagine esterna a cui viene applicato un filtro di smussatura.

```
import flash.display.*;
import flash.filters.BevelFilter;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.net.URLRequest;

// Carica un'immagine sullo stage.
var imageLoader:Loader = new Loader();
var url:String = "http://www.helpexamples.com/flash/images/image3.jpg";
var urlReq:URLRequest = new URLRequest(url);
imageLoader.load(urlReq);
addChild(imageLoader);
```

```
// Crea il filtro smussatura e ne imposta le proprietà.
var bevel:BevelFilter = new BevelFilter();

bevel.distance = 5;
bevel.angle = 45;
bevel.highlightColor = 0xFFFF00;
bevel.highlightAlpha = 0.8;
bevel.shadowColor = 0x666666;
bevel.shadowAlpha = 0.8;
bevel.blurX = 5;
bevel.blurY = 5;
bevel.strength = 5;
bevel.quality = BitmapFilterQuality.HIGH;
bevel.type = BitmapFilterType.INNER;
bevel.knockout = false;

// Applica il filtro all'immagine.
imageLoader.filters = [bevel];
```

Filtro sfocatura

La classe BlurFilter sfuma, o sfoca, un oggetto di visualizzazione e il relativo contenuto. Gli effetti di sfocatura sono utili per dare l'impressione che un oggetto non sia a fuoco o per simulare un rapido movimento. Impostando la proprietà `quality` del filtro sfocatura su un valore basso, è possibile simulare l'effetto di un obiettivo leggermente sfocato. Impostandola su un valore elevato si ottiene un effetto di sfocatura fluida simile alla sfocatura gaussiana.

L'esempio seguente crea un oggetto circle mediante il metodo `drawCircle()` della classe Graphics e a esso applica un filtro sfocatura:

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.BlurFilter;

// Disegna un cerchio.
var redDotCutout:Sprite = new Sprite();
redDotCutout.graphics.lineStyle();
redDotCutout.graphics.beginFill(0xFF0000);
redDotCutout.graphics.drawCircle(145, 90, 25);
redDotCutout.graphics.endFill();

// Aggiunge il cerchio all'elenco di visualizzazione.
addChild(redDotCutout);

// Applica il filtro sfocatura al rettangolo.
var blur:BlurFilter = new BlurFilter();
blur.blurX = 10;
blur.blurY = 10;
blur.quality = BitmapFilterQuality.MEDIUM;
redDotCutout.filters = [blur];
```

Filtro ombra esterna

Le ombre esterne danno l'impressione che sia presente una sorgente di luce separata al di sopra di un oggetto di destinazione. La posizione e l'intensità di questa sorgente di luce può essere modificata per produrre una vasta gamma di effetti di ombra esterna.

Il filtro ombra esterna utilizza un algoritmo simile a quello utilizzato dal filtro sfocatura. La differenza principale consiste nel fatto che il filtro ombra esterna contiene alcune proprietà aggiuntive che è possibile modificare per simulare diversi attributi della sorgente di luce (ad esempio, il valore alfa, il colore, l'offset e la luminosità).

Questo filtro consente anche di applicare delle opzioni di trasformazione personalizzate allo stile dell'ombra esterna, tra cui l'ombra interna o esterna e la modalità foratura.

Nell'esempio di codice seguente viene creato lo sprite di una casella quadrata a cui viene applicata un'ombra esterna:

```
import flash.display.Sprite;
import flash.filters.DropShadowFilter;

// Disegna una casella.
var boxShadow:Sprite = new Sprite();
boxShadow.graphics.lineStyle(1);
boxShadow.graphics.beginFill(0xFF3300);
boxShadow.graphics.drawRect(0, 0, 100, 100);
boxShadow.graphics.endFill();
addChild(boxShadow);

// Applica il filtro ombra esterna alla casella.
var shadow:DropShadowFilter = new DropShadowFilter();
shadow.distance = 10;
shadow.angle = 25;

// È anche possibile impostare altre proprietà, quali il colore d'ombra,
// l'alfa, la quantità di sfocatura, l'intensità, la qualità e le opzioni
// per gli effetti ombra interna e foratura.

boxShadow.filters = [shadow];
```

Filtro bagliore

La classe GlowFilter applica un effetto di illuminazione agli oggetti di visualizzazione, dando l'impressione che una luce venga diretta da sotto l'oggetto per creare un bagliore soffuso.

In modo simile al filtro ombra esterna, il filtro bagliore include delle proprietà per modificare la distanza, l'angolazione e il colore della sorgente di luce per produrre effetti variabili.
La classe GlowFilter contiene anche diverse opzioni per modificare lo stile del bagliore, tra cui il bagliore interno ed esterno e la modalità di foratura.

Nell'esempio di codice seguente viene creata una croce mediante la classe Sprite e a essa viene applicato un filtro bagliore:

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.GlowFilter;

// Crea una croce.
var crossGraphic:Sprite = new Sprite();
crossGraphic.graphics.lineStyle();
crossGraphic.graphics.beginFill(0xCCCC00);
crossGraphic.graphics.drawRect(60, 90, 100, 20);
crossGraphic.graphics.drawRect(100, 50, 20, 100);
crossGraphic.graphics.endFill();
addChild(crossGraphic);

// Applica il filtro bagliore alla croce.
var glow:GlowFilter = new GlowFilter();
glow.color = 0x009922;
glow.alpha = 1;
glow.blurX = 25;
glow.blurY = 25;
glow.quality = BitmapFilterQuality.MEDIUM;

crossGraphic.filters = [glow];
```

Filtro smussatura con gradiente

La classe GradientBevelFilter consente di applicare un effetto di smussatura con gradiente ottimizzato agli oggetti di visualizzazione o BitmapData. L'uso di un colore a gradiente sulla smussatura ne migliora notevolmente la profondità spaziale, poiché dà ai bordi un aspetto tridimensionale più realistico.

Il codice seguente crea un oggetto rectangle mediante il metodo `drawRect()` della classe Shape e applica a esso un filtro smussatura con gradiente.

```
import flash.display.Shape;
import flash.filters.BitmapFilterQuality;
import flash.filters.GradientBevelFilter;

// Disegna un rettangolo.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Applica una smussatura con gradiente al rettangolo.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter();

gradientBevel.distance = 8;
gradientBevel.angle = 225; // l'opposto di 45 gradi
gradientBevel.colors = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
gradientBevel.alphas = [1, 0, 1];
gradientBevel.ratios = [0, 128, 255];
gradientBevel.blurX = 8;
gradientBevel.blurY = 8;
gradientBevel.quality = BitmapFilterQuality.HIGH;

// Altre proprietà consentono di impostare l'intensità del filtro e di
// impostare le opzioni per gli effetti di smussatura interna e foratura.

box.filters = [gradientBevel];

// Aggiunge l'elemento grafico all'elenco di visualizzazione.
addChild(box);
```

Filtro bagliore con gradiente

La classe GradientGlowFilter consente di applicare un effetto di bagliore con gradiente ottimizzato agli oggetti di visualizzazione o BitmapData. L'effetto offre un maggiore controllo cromatico del bagliore e a propria volta produce un effetto bagliore più realistico. Inoltre, il filtro bagliore con gradiente consente di applicare un bagliore con gradiente ai bordi interni, esterni o superiore di un oggetto.

Nell'esempio seguente viene disegnato un cerchio sullo stage, a cui viene applicato un filtro bagliore con gradiente. Spostando il mouse più verso destra e verso il basso, la quantità di sfocatura aumenta rispettivamente in direzione orizzontale e verticale. Inoltre, ogni volta che si fa clic sullo stage, l'intensità della sfocatura aumenta.

```
import flash.events.MouseEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.GradientGlowFilter;

// Crea una nuova istanza Shape.
var shape:Shape = new Shape();

// Disegna la forma.
shape.graphics.beginFill(0xFF0000, 100);
shape.graphics.moveTo(0, 0);
shape.graphics.lineTo(100, 0);
shape.graphics.lineTo(100, 100);
shape.graphics.lineTo(0, 100);
shape.graphics.lineTo(0, 0);
shape.graphics.endFill();

// Posiziona la forma sullo stage.
addChild(shape);
shape.x = 100;
shape.y = 100;

// Definisce un bagliore con gradiente.
var gradientGlow:GradientGlowFilter = new GradientGlowFilter();
gradientGlow.distance = 0;
gradientGlow.angle = 45;
gradientGlow.colors = [0x000000, 0xFF0000];
gradientGlow.alphas = [0, 1];
gradientGlow.ratios = [0, 255];
gradientGlow.blurX = 10;
gradientGlow.blurY = 10;
gradientGlow.strength = 2;
gradientGlow.quality = BitmapFilterQuality.HIGH;
gradientGlow.type = BitmapFilterType.OUTER;
```

```
// Definisce le funzioni per intercettare due eventi.
function onClick(event:MouseEvent):void
{
  gradientGlow.strength++;
  shape.filters = [gradientGlow];
}

function onMouseMove(event:MouseEvent):void
{
  gradientGlow.blurX = (stage.mouseX / stage.stageWidth) * 255;
  gradientGlow.blurY = (stage.mouseY / stage.stageHeight) * 255;
  shape.filters = [gradientGlow];
}
stage.addEventListener(MouseEvent.CLICK, onClick);
stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
```

Esempio: Combinazione di filtri di base

Il codice seguente utilizza diversi filtri di base, combinati con un timer per creare azioni ripetute, al fine di creare la simulazione di un semaforo animato.

```
import flash.display.Shape;
import flash.events.TimerEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;
import flash.filters.GradientBevelFilter;
import flash.utils.Timer;

var count:Number = 1;
var distance:Number = 8;
var angleInDegrees:Number = 225; // l'opposto di 45 gradi
var colors:Array = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
var blurX:Number = 8;
var blurY:Number = 8;
var strength:Number = 1;
var quality:Number = BitmapFilterQuality.HIGH;
var type:String = BitmapFilterType.INNER;
var knockout:Boolean = false;

// Disegna lo sfondo rettangolare per il semaforo.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();
```

```
// Disegna 3 cerchi per le tre luci.
var stopLight:Shape = new Shape();
stopLight.graphics.lineStyle();
stopLight.graphics.beginFill(0xFF0000);
stopLight.graphics.drawCircle(145,90,25);
stopLight.graphics.endFill();

var cautionLight:Shape = new Shape();
cautionLight.graphics.lineStyle();
cautionLight.graphics.beginFill(0xFF9900);
cautionLight.graphics.drawCircle(145,150,25);
cautionLight.graphics.endFill();

var goLight:Shape = new Shape();
goLight.graphics.lineStyle();
goLight.graphics.beginFill(0x00CC00);
goLight.graphics.drawCircle(145,210,25);
goLight.graphics.endFill();

// Aggiunge gli elementi grafici all'elenco di visualizzazione.
addChild(box);
addChild(stopLight);
addChild(cautionLight);
addChild(goLight);

// Applica una smussatura con gradiente al rettangolo del semaforo.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(distance,
  angleInDegrees, colors, alphas, ratios, blurX, blurY, strength, quality,
  type, knockout);
box.filters = [gradientBevel];

// Crea l'ombra interna (per le luci spente) e il bagliore
// (per le luci accese).
var innerShadow:DropShadowFilter = new DropShadowFilter(5, 45, 0, 0.5, 3,
  3, 1, 1, true, false);
var redGlow:GlowFilter = new GlowFilter(0xFF0000, 1, 30, 30, 1, 1, false,
  false);
var yellowGlow:GlowFilter = new GlowFilter(0xFF9900, 1, 30, 30, 1, 1, false,
  false);
var greenGlow:GlowFilter = new GlowFilter(0x00CC00, 1, 30, 30, 1, 1, false,
  false);

// Imposta lo stato iniziale delle luci (verde acceso, rosso/giallo spento).
stopLight.filters = [innerShadow];
cautionLight.filters = [innerShadow];
goLight.filters = [greenGlow];
```

```
// Scambia i filtri in base al valore del conteggio.
function trafficControl(event:TimerEvent):void
{
  if (count == 4)
  {
    count = 1;
  }

  switch (count)
  {
    case 1:
      stopLight.filters = [innerShadow];
      cautionLight.filters = [yellowGlow];
      goLight.filters = [innerShadow];
      break;
    case 2:
      stopLight.filters = [redGlow];
      cautionLight.filters = [innerShadow];
      goLight.filters = [innerShadow];
      break;
    case 3:
      stopLight.filters = [innerShadow];
      cautionLight.filters = [innerShadow];
      goLight.filters = [greenGlow];
      break;
  }

  count++;
}

// Crea un timer per scambiare i filtri in base a un intervallo
// di 3 secondi.
var timer:Timer = new Timer(3000, 9);
timer.addEventListener(TimerEvent.TIMER, trafficControl);
timer.start();
```

Filtro matrice colore

La classe ColorMatrixFilter viene utilizzata per manipolare i valori di colore e alfa dell'oggetto filtrato. In tal modo, è possibile creare modifiche alla saturazione, rotazioni della tonalità (spostando una tavolozza da una gamma di colori a un'altra), modifiche dalla luminanza ad alfa e altri effetti di manipolazione del colore utilizzando i valori di un canale di colore e potenzialmente applicandoli ad altri canali.

Da un punto di vista concettuale, il filtro scorre uno per uno i pixel nell'immagine di origine e separa ogni pixel nei rispettivi componenti rosso, verde, blu e alfa. Quindi, moltiplica i valori forniti nella matrice colore per ognuno di questi valori, aggiungendo i risultati per determinare il valore di colore risultante che verrà visualizzato sullo schermo per tale pixel. La proprietà `matrix` del filtro è un array di 20 numeri che vengono utilizzati nel calcolo del colore finale. Per dettagli sull'algoritmo specifico utilizzato per calcolare i valori di colore, consultare la sezione che descrive la proprietà `matrix` della classe ColorMatrixFilter nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Altre informazioni e altri esempi del filtro matrice colore sono disponibili nell'articolo "Using Matrices for Transformations, Color Adjustments, and Convolution Effects in Flash" (Uso delle matrici per le trasformazioni, le regolazioni cromatiche e gli effetti di convoluzione in Flash) sul sito Web Adobe Developer Center.

Filtro convoluzione

La classe ConvolutionFilter può essere utilizzata per applicare una vasta gamma di trasformazioni dell'immagine agli oggetti BitmapData o di visualizzazione, quali le sfocature, il rilevamento dei bordi, l'aumento della nitidezza, i rilievi e le smussature.

Da un punto di vista concettuale, il filtro convoluzione scorre uno per uno i pixel nell'immagine di origine e determina il colore finale di ogni pixel mediante il valore del pixel e di quelli che lo circondano. Una matrice, specificata come array di valori numerici, indica quanto il valore di ogni pixel vicino influisce sul valore risultante finale.

Si prenda in considerazione il tipo di matrice utilizzato più frequentemente, e cioè una matrice tre per tre. La matrice include nove valori:

```
N N N
N P N
N N N
```

Quando Flash Player applica il filtro convoluzione a un determinato pixel, considera il valore di colore del pixel stesso ("P" nell'esempio), oltre che i valori dei pixel che lo circondano ("N" nell'esempio). Tuttavia, impostando i valori nella matrice, si specifica il livello di priorità di determinati pixel per quanto riguarda l'incidenza sull'immagine risultante.

Ad esempio, la matrice seguente, applicata mediante un filtro convoluzione, lascia l'immagine esattamente com'è:

```
0 0 0
0 1 0
0 0 0
```

Il motivo per cui l'immagine rimane invariata consiste nel fatto che il valore del pixel originale ha un'intensità relativa pari a 1 nel determinare il colore finale del pixel, mentre i valori dei pixel circostanti hanno un'intensità relativa pari a 0 (che significa che i colori di questi pixel non influiscono sull'immagine finale).

In modo analogo, questa matrice fa spostare di un pixel verso sinistra i pixel di un'immagine:

```
0 0 0
0 0 1
0 0 0
```

Si noti che in questo caso il pixel stesso non ha alcun effetto sul valore finale del pixel visualizzato in tale posizione dell'immagine finale: solo il valore del pixel sulla destra viene utilizzato per determinare il valore risultante del pixel.

In ActionScript, si crea la matrice come combinazione di un'istanza Array contenente i valori e due proprietà che specificano il numero di righe e colonne nella matrice. L'esempio seguente carica un'immagine e, quando il caricamento è completato, applica un filtro convoluzione all'immagine mediante la matrice dell'esempio precedente:

```
// Carica un'immagine sullo stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/
  images/image1.jpg");
loader.load(url);
this.addChild(loader);

function applyFilter(event:MouseEvent):void
{
  // Crea la matrice per la convoluzione.
  var matrix:Array = [ 0, 0, 0,
                       0, 0, 1,
                       0, 0, 0 ];

  var convolution:ConvolutionFilter = new ConvolutionFilter();
  convolution.matrixX = 3;
  convolution.matrixY = 3;
  convolution.matrix = matrix;
  convolution.divisor = 1;

  loader.filters = [convolution];
}

loader.addEventListener(MouseEvent.CLICK, applyFilter);
```

Una cosa che non appare ovvia in questo codice è l'effetto dell'uso nella matrice di valori diversi da 1 o 0. Ad esempio, la stessa matrice con il numero 8 anziché 1 nella posizione destra esegue la stessa azione (cioè sposta i pixel verso sinistra). Inoltre, influisce sui colori dell'immagine, rendendoli 8 volte più brillanti. Ciò accade perché i valori di colore del pixel finale vengono calcolati moltiplicando i valori della matrice per i colori dei pixel originali, combinando i valori e dividendo il risultato per il valore della proprietà `divisor` del filtro. Si noti che nel codice di esempio la proprietà `divisor` è impostata su 1. Come regola generale, se si desidera che la luminosità dei colori rimanga più o meno la stessa dell'immagine originale, è necessario rendere il divisore uguale alla somma dei valori di matrice. Pertanto, nel caso di una matrice in cui la somma dei valori è pari a 8 ed è presente un divisore pari a 1, l'immagine risultante sarà circa 8 volte più luminosa di quella originale.

Anche se l'effetto di questa matrice non è particolarmente distinguibile, è possibile utilizzare altri valori di matrice per creare vari effetti. Qui di seguito sono elencati alcuni set standard di valori di matrice per diversi effetti, che utilizzano una matrice tre per tre:

- Sfocatura di base (divisore 5):

```
0 1 0
1 1 1
0 1 0
```

- Aumento della nitidezza (divisore 1):

```
 0, -1, 0
-1, 5, -1
 0, -1, 0
```

- Rilevamento dei bordi (divisore 1):

```
 0, -1, 0
-1, 4, -1
 0, -1, 0
```

- Effetto rilievo (divisore 1):

```
-2, -1, 0
-1, 1, 1
 0, 1, 2
```

Si noti che per la maggior parte di questi effetti il divisore è 1. Infatti, i valori di matrice negativi aggiunti ai valori di matrice positivi producono il risultato 1 (oppure 0 nel caso del rilevamento dei bordi, ma il valore della proprietà `divisor` non può essere 0).

Filtro mappa di spostamento

La classe DisplacementMapFilter utilizza i valori dei pixel di un oggetto BitmapData (noto come immagine della mappa di spostamento) per eseguire un effetto spostamento su un nuovo oggetto. Di solito l'immagine della mappa di spostamento è diversa dall'oggetto di visualizzazione vero e proprio o dall'istanza BitmapData a cui si applica il filtro. Un effetto di spostamento richiede lo spostamento dei pixel nell'immagine filtrata (in altre parole, il loro spostamento dalla posizione originale). Questo filtro può essere utilizzato per creare un effetto spostato, deformato o screziato.

La posizione e la quantità dello spostamento applicato a un determinato pixel vengono determinate dal valore di colore dell'immagine della mappa di spostamento. Quando si utilizza il filtro, oltre a specificare l'immagine della mappa, si specificano i valori seguenti per controllare il modo in cui viene calcolato lo spostamento rispetto all'immagine della mappa:

- Punto mappa: la posizione dell'immagine filtrata in corrispondenza della quale viene applicato l'angolo superiore sinistro del filtro spostamento. È possibile utilizzarla solo se si desidera applicare il filtro a parte di un'immagine.
- Componente X: il canale colore dell'immagine della mappa che influisce sulla posizione x dei pixel.
- Componente Y: il canale colore dell'immagine della mappa che influisce sulla posizione y dei pixel.
- Scala X: un valore moltiplicatore che specifica l'intensità dello spostamento dell'asse x.
- Scala Y: un valore moltiplicatore che specifica l'intensità dello spostamento dell'asse y.
- Modalità filtro: determina l'operazione che deve effettuare Flash Player negli spazi vuoti creati dai pixel spostati. Le opzioni disponibili, definite come costanti nella classe DisplacementMapFilterMode, sono le seguenti: visualizzare i pixel originali (modalità filtro `IGNORE`), applicare i pixel dell'altro lato dell'immagine (modalità filtro `WRAP`, predefinita), utilizzare il pixel spostato più vicino (modalità filtro `CLAMP`) o riempire gli spazi con un colore (modalità filtro `COLOR`).

Per acquisire una conoscenza di base sul funzionamento del filtro mappa di spostamento, si consideri un semplice esempio. Nel codice seguente viene caricata un'immagine che, al termine del caricamento, viene centrata sullo stage. All'immagine viene quindi applicato un filtro mappa di spostamento, in base al quale i pixel dell'intera immagine vengono spostati in senso orizzontale verso sinistra.

```
import flash.display.BitmapData;
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.geom.Point;
import flash.net.URLRequest;
```

```
// Carica un'immagine sullo stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/
  images/image3.jpg");
loader.load(url);
this.addChild(loader);

var mapImage:BitmapData;
var displacementMap:DisplacementMapFilter;

// This function is called when the image finishes loading.
function setupStage(event:Event):void
{
  // Centra l'immagine caricata sullo stage.
  loader.x = (stage.stageWidth - loader.width) / 2;
  loader.y = (stage.stageHeight - loader.height) / 2;

  // Crea il filtro mappa di spostamento.
  mapImage = new BitmapData(loader.width, loader.height, false, 0xFF0000);

  // Crea il filtro spostamento.
  displacementMap = new DisplacementMapFilter();
  displacementMap.mapBitmap = mapImage;
  displacementMap.mapPoint = new Point(0, 0);
  displacementMap.componentX = BitmapDataChannel.RED;
  displacementMap.scaleX = 250;
  loader.filters = [displacementMap];
}

loader.contentLoaderInfo.addEventListener(Event.COMPLETE, setupStage);
```

Di seguito sono riportate le proprietà utilizzate per definire lo spostamento:

- Bitmap mappa: la bitmap di spostamento è una nuova istanza BitmapData creata dal codice. Le sue dimensioni corrispondono a quelle dell'immagine caricata (pertanto lo spostamento viene applicato all'intera immagine) ed è riempita con pixel rossi uniformi.
- Punto mappa: questo valore è impostato sul punto 0, 0 e anche in questo caso provoca l'applicazione dello spostamento all'intera immagine.
- Componente X: questo valore viene impostato sulla costante `BitmapDataChannel.RED`; ciò significa che il valore rosso della bitmap della mappa determina la quantità di spostamento dei pixel lungo l'asse x.
- Scala X: questo valore è impostato su 250. La quantità di spostamento (dipendente dal fatto che l'immagine della mappa è completamente rossa) sposta l'immagine solo di poco (circa mezzo pixel), pertanto se questo valore viene impostato su 1 l'immagine si sposta solo di 0,5 pixel in orizzontale. Se lo si imposta su 250, l'immagine si sposta di circa 125 pixel.

In base a queste impostazioni i pixel dell'immagine filtrata vengono spostati di 250 pixel verso sinistra. La direzione (sinistra o destra) e la quantità dello spostamento si basano sul valore di colore dei pixel nell'immagine della mappa. In teoria, Flash Player scorre uno per uno i pixel dell'immagine filtrata (perlomeno, i pixel nell'area in cui è applicato il filtro, che in questo caso significa tutti i pixel), e per ogni pixel effettua le operazioni seguenti:

1. Trova il pixel corrispondente nella mappa dell'immagine. Ad esempio, quando Flash Player calcola la quantità di spostamento per il pixel nell'angolo superiore sinistro dell'immagine filtrata, cerca il pixel nell'angolo superiore sinistro dell'immagine della mappa.
2. Determina il valore del canale di colore specificato nel pixel della mappa. In questo caso, il canale di colore del componente x è il canale rosso, pertanto Flash Player verifica se il canale rosso dell'immagine della mappa si trova in corrispondenza del pixel in questione. Dal momento che la mappa dell'immagine è rosso uniforme, il canale rosso del pixel è 0xFF, o 255. Questo valore viene utilizzato come valore di spostamento.
3. Confronta il valore di spostamento con il valore "medio" (ovvero 127, che si trova a metà tra 0 e 255). Se il valore di spostamento è inferiore al valore medio, il pixel si sposta in una direzione positiva (a destra per lo spostamento x, verso il basso per lo spostamento y). D'altro canto, se il valore di spostamento è superiore al valore medio (come accade in questo esempio), il pixel si sposta in una direzione negativa (a sinistra per lo spostamento x, verso l'alto per lo spostamento y). Per maggiore precisione, Flash Player sottrae il valore di spostamento da 127, e il risultato (positivo o negativo) rappresenta la quantità relativa di spostamento che viene applicata.
4. Infine, determina la quantità effettiva di spostamento determinando la percentuale di spostamento completo rappresentata dal valore di spostamento relativo. In questo caso, il rosso pieno significa uno spostamento del 100%. Tale percentuale viene quindi moltiplicata per il valore della scala x o della scala y per determinare il numero di pixel di spostamento da applicare. In questo esempio, 100% moltiplicato per un moltiplicatore di 250 determina la quantità di spostamento: circa 125 pixel verso sinistra.

Dal momento che non sono stati specificati valori per il componente y e la scala y, sono stati utilizzati i valori predefiniti (che non provocano alcuno spostamento); pertanto l'immagine non viene spostata in direzione verticale.

Nell'esempio viene utilizzata l'impostazione predefinita della modalità filtro, `WRAP`; pertanto mentre i pixel vengono spostati verso sinistra, lo spazio vuoto a destra viene riempito con i pixel che sono stati esclusi dallo spostamento del bordo sinistro dell'immagine. È possibile sperimentare con questo valore per osservare il comportamento dei diversi effetti. Ad esempio, se si aggiunge la riga seguente alla porzione di codice in cui sono impostate le proprietà dello spostamento (prima della riga `loader.filters = [displacementMap]`), l'effetto che si ottiene è quello di un'immagine sfumata su tutto lo stage:

```
displacementMap.mode = DisplacementMapFilterMode.CLAMP;
```

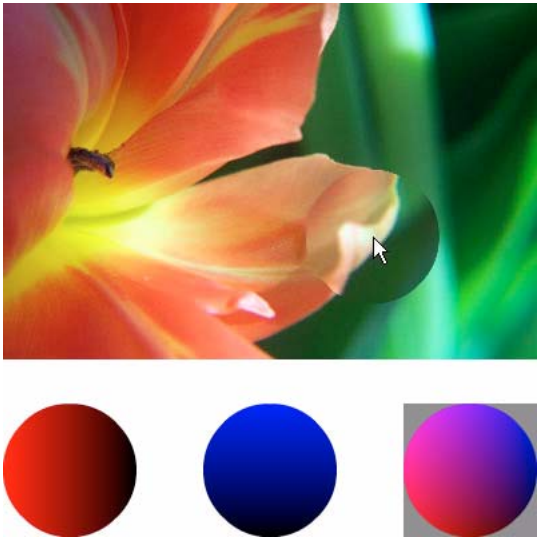
Per un esempio più complesso, il codice seguente utilizza un filtro mappa di spostamento per creare un effetto lente di ingrandimento su un'immagine:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.display.BitmapDataChannel;
import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Shape;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.filters.DisplacementMapFilterMode;
import flash.geom.Matrix;
import flash.geom.Point;
import flash.net.URLRequest;

// Crea i cerchi con gradiente che insieme formano
// l'immagine della mappa di spostamento.
var radius:uint = 50;

var type:String = GradientType.LINEAR;
var redColors:Array = [ 0xFF0000, 0x000000 ];
var blueColors:Array = [ 0x0000FF, 0x000000 ];
var alphas:Array = [ 1, 1 ];
var ratios:Array = [ 0, 255 ];
var xMatrix:Matrix = new Matrix();
xMatrix.createGradientBox(radius * 2, radius * 2);
var yMatrix:Matrix = new Matrix();
yMatrix.createGradientBox(radius * 2, radius * 2, Math.PI / 2);

var xCircle:Shape = new Shape();
xCircle.graphics.lineStyle(0, 0, 0);
xCircle.graphics.beginGradientFill(type, redColors, alphas, ratios,
  xMatrix);
xCircle.graphics.drawCircle(radius, radius, radius);

var yCircle:Shape = new Shape();
yCircle.graphics.lineStyle(0, 0, 0);
```

```
yCircle.graphics.beginGradientFill(type, blueColors, alphas, ratios,
  yMatrix);
yCircle.graphics.drawCircle(radius, radius, radius);

// Posiziona i cerchi nella parte inferiore dello schermo, per riferimento.
this.addChild(xCircle);
xCircle.y = stage.stageHeight - xCircle.height;
this.addChild(yCircle);
yCircle.y = stage.stageHeight - yCircle.height;
yCircle.x = 200;

// Carica un'immagine sullo stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/
  images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Crea l'immagine della mappa combinando i due cerchi con gradiente.
var map:BitmapData = new BitmapData(xCircle.width, xCircle.height, false,
  0x7F7F7F);
map.draw(xCircle);
var yMap:BitmapData = new BitmapData(yCircle.width, yCircle.height, false,
  0x7F7F7F);
yMap.draw(yCircle);
map.copyChannel(yMap, yMap.rect, new Point(0, 0), BitmapDataChannel.BLUE,
  BitmapDataChannel.BLUE);
yMap.dispose();

// Visualizza l'immagine della mappa sullo stage, per riferimento.
var mapBitmap:Bitmap = new Bitmap(map);
this.addChild(mapBitmap);
mapBitmap.x = 400;
mapBitmap.y = stage.stageHeight - mapBitmap.height;

// Questa funzione crea il filtro mappa di spostamento in corrispondenza
// del cursore.
function magnify():void
{
  // Posiziona il filtro
  var filterX:Number = (loader.mouseX) - (map.width / 2);
  var filterY:Number = (loader.mouseY) - (map.height / 2);
  var pt:Point = new Point(filterX, filterY);
  var xyFilter:DisplacementMapFilter = new DisplacementMapFilter();
  xyFilter.mapBitmap = map;
  xyFilter.mapPoint = pt;
  // Il rosso nell'immagine della mappa controlla lo spostamento x.
  xyFilter.componentX = BitmapDataChannel.RED;
  // Il blu nell'immagine della mappa controlla lo spostamento y.
  xyFilter.componentY = BitmapDataChannel.BLUE;
```

```
    xyFilter.scaleX = 35;
    xyFilter.scaleY = 35;
    xyFilter.mode = DisplacementMapFilterMode.IGNORE;
    loader.filters = [xyFilter];
}

// Questa funzione viene chiamata ogni volta che il mouse viene spostato.
// Se il mouse è sopra l'immagine caricata, applica il filtro.
function moveMagnifier(event:MouseEvent):void
{
    if (loader.hitTestPoint(loader.mouseX, loader.mouseY))
    {
        magnify();
    }
}
loader.addEventListener(MouseEvent.MOUSE_MOVE, moveMagnifier);
```

Il codice genera innanzi tutto due cerchi con gradiente, che vengono combinati per formare l'immagine della mappa di spostamento. Il cerchio rosso crea lo spostamento dell'asse x (`xyFilter.componentX = BitmapDataChannel.RED`), mentre quello blu crea lo spostamento dell'asse y (`xyFilter.componentY = BitmapDataChannel.BLUE`). Per semplificare l'aspetto dell'immagine della mappa di spostamento, il codice aggiunge sia i cerchi originali che i cerchi combinati che fungono da immagine della mappa nella parte inferiore dello schermo.



Quindi, il codice carica un'immagine e, quando il mouse si sposta, applica il filtro spostamento alla porzione dell'immagine che si trova sotto il mouse. I cerchi con gradiente utilizzati come immagine della mappa di spostamento provocano la diffusione dell'area spostata rispetto al puntatore del mouse. Si noti che le aree grigie dell'immagine della mappa di spostamento non provocano alcuno spostamento. Il colore grigio è `0x7F7F7F`. I canali blu e rosso di tale tonalità di grigio corrispondono esattamente alla tonalità media di tali canali di colore, pertanto non viene effettuato alcuno spostamento in un'area grigia dell'immagine della mappa. In modo analogo, al centro del cerchio non avviene alcuno spostamento. Benché in tale area il colore non sia grigio, i canali blu e rosso del colore sono identici ai canali blu e rosso del grigio medio, e poiché il blu e il rosso sono i colori che provocano lo spostamento, in tale area non si verifica alcuno spostamento.

Esempio: Filter Workbench

L'esempio Filter Workbench fornisce una semplice interfaccia utente per applicare diversi filtri a un'immagine e visualizzare il codice risultante che può essere utilizzato per generare lo stesso effetto mediante ActionScript. Per una descrizione di questo esempio e per scaricare il codice di origine, vedere http://www.adobe.com/go/learn_fl_filters_it.

CAPITOLO 16

Operazioni con i clip filmato

16

La classe MovieClip è la classe di base per l'animazione e i simboli di clip filmato creati in Adobe Flash CS3 Professional. Contiene tutti i comportamenti e le funzionalità degli oggetti di visualizzazione, ma con proprietà e metodi aggiuntivi per il controllo della linea temporale di un clip filmato. Questo capitolo descrive come utilizzare ActionScript per controllare la riproduzione di un clip filmato e come creare dinamicamente un clip filmato.

Sommario

Elementi fondamentali dei clip filmato

Introduzione alle operazioni con i clip filmato

I clip filmato sono un elemento fondamentale per la creazione di contenuti animati con lo strumento di creazione di Flash e per il controllo di tali contenuti mediante ActionScript. Ogni qual volta si crea un simbolo di un clip filmato in Flash, Flash aggiunge il simbolo alla libreria del documento Flash in questione. Per impostazione predefinita, questo simbolo diventa un'istanza della classe MovieClip e, come tale, contiene le proprietà e i metodi di tale classe.

Quando un'istanza di un clip filmato viene posizionata sullo stage, il clip filmato avanza automaticamente lungo la propria linea temporale (se è composto da più di un fotogramma), a meno che non se ne modifichi la riproduzione mediante ActionScript. È proprio questa linea temporale che contraddistingue la classe MovieClip, perché consente di creare animazioni mediante le interpolazioni di movimento e di forma con lo strumento di creazione di Flash. Al contrario, con un oggetto di visualizzazione che è un'istanza della classe Sprite, è possibile creare animazioni solo modificando a livello di codice i valori dell'oggetto.

Nelle precedenti versioni di ActionScript, la classe MovieClip costituiva la classe di base di tutte le istanze sullo stage. In ActionScript 3.0, un clip filmato è solo uno dei numerosi oggetti di visualizzazione che possono comparire sullo schermo. Se una linea temporale non è necessaria per la funzione di un oggetto di visualizzazione, l'utilizzo della classe Shape o della classe Sprite al posto della classe MovieClip può migliorare le prestazioni del rendering. Per ulteriori informazioni su come scegliere l'oggetto di visualizzazione più adatto a un'operazione, vedere "Scelta di una sottoclasse DisplayObject" a pagina 421.

Operazioni comuni con i clip filmato

Le seguenti operazioni comuni con i clip filmato sono descritte in questo capitolo:

- Riproduzione e interruzione dei clip filmato
- Riproduzione al contrario dei clip filmato
- Spostamento dell'indicatore di riproduzione in punti specifici della linea temporale del clip filmato
- Operazioni con le etichette di fotogrammi in ActionScript
- Accesso alle informazioni relative alle scene in ActionScript
- Creazione di istanze di simboli della libreria di un clip filmato mediante ActionScript
- Caricamento e controllo di file SWF esterni, compresi quelli creati per versioni precedenti di Flash Player
- Creazione di un sistema ActionScript per la creazione di risorse grafiche da caricare e utilizzare in fase di runtime

Concetti e termini importanti

L'elenco di riferimento seguente contiene dei termini importanti che vengono citati in questo capitolo:

- SWF AVM1: un file SWF creato mediante ActionScript 1.0 o ActionScript 2.0, generalmente ottimizzato per Flash Player 8 o versione precedente.
- SWF AVM2: un file SWF creato mediante ActionScript 3.0 per Adobe Flash Player 9.

- SWF esterno: un file SWF creato separatamente rispetto al file SWF di progetto e destinato a essere caricato e riprodotto nel file SWF di progetto.
- Fotogramma: la più piccola divisione temporale sulla linea temporale. Come accade con la pellicola cinematografica, ogni fotogramma è come un'istantanea dell'animazione nel tempo, e quando i fotogrammi vengono riprodotti rapidamente in sequenza viene creato l'effetto dell'animazione.
- Linea temporale: la rappresentazione metaforica della serie di fotogrammi che compongono una sequenza di animazione di un clip filmato. La linea temporale di un oggetto MovieClip è equivalente alla linea temporale nello strumento di creazione di Flash.
- Indicatore di riproduzione: un marcatore che identifica la posizione (fotogramma) nella linea temporale che viene visualizzata in un determinato momento.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché questo capitolo descrive le operazioni con i clip filmato in ActionScript, tutti gli esempi di codice riportati sono pensati per la manipolazione di un simbolo di clip filmato, che è stato creato e collocato sullo stage. La prova degli esempi prevede la visualizzazione del risultato in Flash Player per vedere gli effetti del codice sul simbolo. Per provare gli esempi di codice contenuti in questo capitolo:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.
4. Creare un'istanza del simbolo di clip filmato sullo stage. Ad esempio, disegnare una forma, selezionarla, scegliere Elabora > Converti in simbolo e assegnare al simbolo un nome.
5. Con il clip filmato selezionato, nella finestra di ispezione Proprietà assegnargli un nome istanza. Questo nome deve corrispondere al nome utilizzato per il clip filmato nell'esempio di codice, ad esempio, se il codice manipola un clip filmato chiamato `myMovieClip`, è necessario denominare `myMovieClip` anche l'istanza del clip filmato.
6. Eseguire il programma selezionando Controllo > Prova filmato.

   Sullo schermo vengono visualizzati i risultati della manipolazione del clip filmato secondo quanto specificato nell'esempio di codice.

Per informazioni su altre tecniche per la prova degli esempi di codice, vedere "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Operazioni con gli oggetti MovieClip

Quando si pubblica un file SWF, per impostazione predefinita Flash converte tutte le istanze dei simboli di clip filmato sullo stage in oggetti MovieClip. È possibile rendere il simbolo di un clip filmato disponibile per ActionScript assegnando a esso un nome di istanza nel campo Nome istanza della finestra di ispezione Proprietà. Quando viene creato il file SWF, Flash genera il codice che crea l'istanza MovieClip sullo stage e dichiara una variabile utilizzando il nome di istanza. Se sono stati assegnati dei nomi a dei clip filmato che sono nidificati in altri clip filmato, tali clip filmato secondari vengono trattati come proprietà del clip filmato principale: è possibile accedere al clip filmato secondario utilizzando la sintassi del punto. Ad esempio, se un clip filmato con il nome di istanza `childClip` è nidificato all'interno di un clip filmato di nome `parentClip`, è possibile riprodurre l'animazione della linea temporale del clip filmato secondario chiamando il codice seguente:

```
parentClip.childClip.play()
```

Mentre alcuni metodi e proprietà della classe MovieClip di ActionScript 2.0 rimangono uguali, altri sono cambiati. Tutte le proprietà precedute da un carattere di sottolineatura sono state ridenominate. Ad esempio, alle proprietà `_width` e `_height` si accede ora come `width` e `height`, mentre a `_xscale` e `_yscale` si accede come `scaleX` e `scaleY`. Per un elenco completo delle proprietà e dei metodi della classe MovieClip, consultare la *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Controllo della riproduzione di clip filmato

Flash utilizza la metafora della linea temporale per veicolare l'animazione o la modifica di uno stato. Qualunque elemento visivo che utilizza una linea temporale deve essere un oggetto MovieClip o essere un'estensione della classe MovieClip. Mentre ActionScript può indicare a qualunque filmato di fermarsi, avviare la riproduzione o passare a un altro punto della linea temporale, non può essere utilizzato per creare dinamicamente una linea temporale o aggiungere del contenuto in corrispondenza di determinati fotogrammi; ciò è possibile solo nello strumento di creazione di Flash.

Quando un oggetto MovieClip è in corso di riproduzione, avanza lungo la propria linea temporale a una velocità dettata dalla frequenza dei fotogrammi del file SWF. In alternativa, è possibile ignorare questa impostazione impostando la proprietà `Stage.frameRate` in ActionScript.

Riproduzione e interruzione di clip filmato

I metodi `play()` e `stop()` consentono il controllo di base di un clip filmato lungo tutta la linea temporale. Ad esempio, si supponga che un simbolo di clip filmato sullo stage contenga l'animazione di una bicicletta che attraversa lo schermo, con il nome di istanza impostato su `bicycle`. Se il codice seguente è associato a un fotogramma della linea temporale principale,

```
bicycle.stop();
```

la bicicletta non si muove (cioè, la sua animazione non viene riprodotta). Il movimento della bicicletta può essere avviato mediante altre interazioni dell'utente. Ad esempio, se è presente un pulsante di nome `startButton`, il codice seguente su un fotogramma chiave della linea temporale fa in modo che il clic sul pulsante avvii la riproduzione dell'animazione:

```
// Questa funzione viene chiamata quando si fa clic sul pulsante. Attiva la
// riproduzione dell'animazione della bicicletta.
function playAnimation(event:MouseEvent):void
{
  bicycle.play();
}
// Registra la funzione come listener con il pulsante.
startButton.addEventListener(MouseEvent.CLICK, playAnimation);
```

Avanzamento veloce e riavvolgimento

I metodi `play()` e `stop()` non sono l'unico modo per controllare la riproduzione in un clip filmato. È possibile muovere manualmente l'indicatore di riproduzione lungo la linea temporale utilizzando i metodi `nextFrame()` e `prevFrame()`. Se si chiama uno di questi metodi, la riproduzione viene interrotta e l'indicatore di riproduzione viene spostato rispettivamente in avanti o indietro.

L'utilizzo del metodo `play()` equivale a chiamare `nextFrame()` ogni volta che si attiva l'evento `enterFrame` dell'oggetto clip filmato. Lungo queste linee è possibile riprodurre il clip filmato `bicycle` al contrario creando un listener di eventi per l'evento `Event.ENTER_FRAME` e ordinando a `bicycle` di passare al proprio fotogramma precedente, come indicato di seguito:

```
// Questa funzione viene chiamata quando viene attivato l'evento
// enterFrame. In altre parole, viene chiamata una volta per fotogramma.
function everyFrame(event:Event):void
{
  if (bicycle.currentFrame == 1)
  {
    bicycle.gotoAndStop(bicycle.totalFrames);
  }
  else
  {
    bicycle.prevFrame();
  }
}
bicycle.addEventListener(Event.ENTER_FRAME, everyFrame);
```

Nella riproduzione normale, se un clip filmato contiene più di un solo fotogramma, viene ripetuto ciclicamente in modo indefinito durante la riproduzione; ovvero, ritorna al fotogramma 1 se avanza oltre il proprio fotogramma finale. Quando si utilizza `prevFrame()` o `nextFrame()`, questo comportamento non si verifica automaticamente (se si chiama `prevFrame()` quando l'indicatore di riproduzione si trova sul fotogramma 1, l'indicatore di riproduzione non viene spostato all'ultimo fotogramma). La condizione `if` nell'esempio precedente verifica se l'indicatore di riproduzione si è spostato all'indietro fino al primo fotogramma e imposta l'indicatore di riproduzione sul fotogramma finale, creando un ciclo continuo del clip filmato che viene eseguito all'indietro.

Passaggio a un fotogramma diverso e uso delle etichette di fotogramma

L'invio di un clip filmato a un nuovo fotogramma è un'operazione molto semplice. Se si chiama `gotoAndPlay()` o `gotoAndStop()`, il clip filmato passa al numero di fotogramma specificato come parametro. In alternativa, è possibile passare una stringa che corrisponde al nome dell'etichetta di fotogramma. È possibile assegnare un'etichetta a qualunque fotogramma sulla linea temporale. A tale scopo, selezionare un fotogramma sulla linea temporale, quindi immettere un nome nel campo Etichetta fotogramma della finestra di ispezione Proprietà.

I vantaggi dell'utilizzo delle etichette di fotogramma anziché dei numeri sono particolarmente evidenti quando si crea un clip filmato complesso. Quando il numero di fotogrammi, livelli e interpolazioni in un'animazione diventa piuttosto elevato, è opportuno considerare di assegnare delle etichette ai fotogrammi più importanti con delle descrizioni eloquenti che rappresentino delle variazioni nel comportamento del clip filmato (ad esempio, "off", "walking" o "running"). In questo modo si migliora la leggibilità del codice e si fornisce maggiore flessibilità, dal momento che le chiamate ActionScript utilizzate per passare a un fotogramma con un'etichetta sono dei puntatori a un singolo riferimento (l'etichetta) anziché a un numero di fotogramma specifico. Se successivamente si decide di spostare un segmento particolare dell'animazione in un fotogramma diverso, non è necessario modificare il codice ActionScript fintanto che si conserva la stessa etichetta per i fotogrammi nella nuova posizione.

Per rappresentare le etichette di fotogramma nel codice, ActionScript 3.0 include la classe FrameLabel. Ogni istanza di questa classe rappresenta una singola etichetta di fotogramma e ha una proprietà `name` che rappresenta il nome dell'etichetta di fotogramma specificato nella finestra di ispezione Proprietà e una proprietà `frame` che rappresenta il numero di fotogramma del fotogramma della linea temporale in cui si trova l'etichetta.

Per poter accedere alle istanze FrameLabel associate a un'istanza di clip filmato, la classe MovieClip include due proprietà che restituiscono direttamente degli oggetti FrameLabel. La proprietà `currentLabels` restituisce un array che è composto da tutti gli oggetti FrameLabel presenti sull'intera linea temporale di un clip filmato. La proprietà `currentLabel` restituisce un solo oggetto FrameLabel, che rappresenta l'etichetta di fotogramma incontrata più di recente lungo la linea temporale.

Si supponga di stare creando un clip filmato di nome robot e di aver assegnato delle etichette ai suoi vari stati di animazione. È possibile impostare una condizione che verifica la proprietà `currentLabel` per accedere allo stato corrente di robot, come nel codice seguente:

```
if (robot.currentLabel.name == "walking"
{
  // Esegue un'operazione
}
```

Operazioni con le scene

Nell'ambiente di creazione di Flash, è possibile utilizzare le scene per demarcare una serie di linee temporali che verranno attraversate da un file SWF. Utilizzando il secondo parametro del metodo `gotoAndPlay()` o `gotoAndStop()`, è possibile specificare una scena a cui deve essere inviato l'indicatore di riproduzione. Tutti i file FLA cominciano solo con la scena iniziale, ma è possibile creare delle nuove scene.

L'uso delle scene non è sempre l'approccio migliore, poiché le scene presentano una serie di svantaggi. Un documento Flash che contiene più scene può essere difficile da gestire, in particolare in presenza di più autori. Le scene multiple possono anche rivelarsi inefficienti dal punto di vista della larghezza di banda, poiché il processo di pubblicazione unisce tutte le scene in un'unica linea temporale. Ciò provoca uno scaricamento progressivo di tutte le scene, anche se non vengono mai riprodotte. Per questi motivi, l'uso di più scene è spesso scoraggiato a meno che non siano necessarie per organizzare animazioni basate su più linee temporali.

La proprietà `scenes` della classe MovieClip è un array di oggetti Scene che rappresentano tutte le scene nel file SWF. La proprietà `currentScene` restituisce un oggetto Scene che rappresenta la scena che è in corso di riproduzione.

La classe Scene ha diverse proprietà che forniscono informazioni su una scena. La proprietà `labels` restituisce un array di oggetti FrameLabel che rappresentano le etichette di fotogramma in tale scena. La proprietà `name` restituisce il nome della scena sotto forma di stringa. La proprietà `numFrames` restituisce un int che rappresenta il numero totale di fotogrammi nella scena.

Creazione di oggetti MovieClip mediante ActionScript

Un metodo per aggiungere contenuti allo schermo in Flash consiste nel trascinare le risorse dalla libreria allo stage; non si tratta tuttavia dell'unico flusso di lavoro. Per i progetti più complessi, gli sviluppatori esperti di solito preferiscono creare i clip filmato a livello di codice. Questo approccio offre vari vantaggi: un riutilizzo più facile del codice, una maggiore velocità in fase di compilazione e modifiche più sofisticate disponibili solo in ActionScript.

L'API dell'elenco di visualizzazione di ActionScript 3.0 semplifica il processo di creazione dinamica degli oggetti MovieClip. La capacità di creare direttamente un'istanza di un'istanza MovieClip, separandola dal processo di aggiunta all'elenco di visualizzazione, fornisce flessibilità e semplicità senza sacrificare il controllo.

In ActionScript 3.0, quando si crea un'istanza di un clip filmato (o di qualunque altro oggetto di visualizzazione) a livello di codice, non è visibile sullo schermo fino a quando non viene aggiunta all'elenco di visualizzazione chiamando il metodo `addChild()` o `addChildAt()` su un contenitore di un oggetto di visualizzazione. In questo modo è possibile creare un clip filmato, impostarne le proprietà e persino chiamarne i metodi ancora prima che ne venga effettuato il rendering sullo schermo. Per ulteriori informazioni sulle operazioni con l'elenco di visualizzazione, vedere "Uso dei contenitori degli oggetti di visualizzazione" a pagina 408.

Esportazione dei simboli della libreria per ActionScript

Per impostazione predefinita, le istanze dei simboli di clip filmato nella libreria di un documento Flash non possono essere create in modo dinamico (cioè solo mediante ActionScript), perché ogni simbolo che viene esportato per l'utilizzo in ActionScript viene aggiunto alle dimensioni del file SWF, e non tutti i simboli sono destinati all'uso sullo stage. Per tale motivo, per poter rendere un simbolo disponibile in ActionScript, è necessario specificare che deve essere esportato per ActionScript.

Per esportare un simbolo per ActionScript:

1. Selezionare il simbolo nel pannello Libreria e aprire la finestra di dialogo delle proprietà del simbolo.
2. Se necessario, attivare le impostazioni avanzate.
3. Nella sezione Concatenamento, selezionare Esporta per ActionScript.

   In questo modo vengono attivati i campi Classe e Classe base.

   Per impostazione predefinita, il campo Classe viene compilato con il nome del simbolo, senza spazi (ad esempio, un simbolo denominato "Tree House" diventa "TreeHouse"). Per specificare che il simbolo deve utilizzare una classe personalizzata per il proprio comportamento, immettere nel campo il nome completo della classe, compreso il relativo pacchetto. Se si desidera poter creare delle istanze del simbolo in ActionScript ma non è necessario aggiungere alcun comportamento aggiuntivo, è possibile lasciare invariato il nome della classe.

   Il valore predefinito del campo Base Class è `flash.display.MovieClip`. Se si desidera che il simbolo estenda la funzionalità di un'altra classe personalizzata, è possibile specificare il nome di tale classe, a condizione che estenda la classe Sprite (o MovieClip).

4. Fare clic su OK per salvare le modifiche.

   A questo punto, se Flash non riesce a trovare un file ActionScript esterno con una definizione della classe specificata (ad esempio, se non è necessario aggiungere dei comportamenti supplementari per il simbolo), viene visualizzato un avviso:

   Impossibile trovare una definizione per questa classe nel percorso di classe. Ne verrà generata una automaticamente nel file SWF al momento dell'esportazione.

   È possibile ignorare l'avviso se il simbolo della libreria non richiede altre funzionalità oltre quella della classe MovieClip.

Se non si fornisce una classe per il simbolo, Flash crea una classe per il simbolo equivalente alla seguente:

```
package
{
  import flash.display.MovieClip;

  public class ExampleMovieClip extends MovieClip
  {
    public function ExampleMovieClip()
    {
    }
  }
}
```

Se si desidera aggiungere delle funzionalità ActionScript supplementari al simbolo, aggiungere le proprietà e i metodi appropriati alla sua struttura. Ad esempio, si supponga che il simbolo di un clip filmato contenga un cerchio con una larghezza di 50 pixel e un'altezza di 50 pixel e che per il simbolo sia stata specificata l'esportazione per ActionScript con una classe di nome Circle. Il codice seguente, quando viene incluso in un file Circle.as, estende la classe MovieClip e fornisce al simbolo i metodi aggiuntivi `getArea()` e `getCircumference()`:

```
package
{
  import flash.display.MovieClip;

  public class Circle extends MovieClip
  {
    public function Circle()
    {
    }

    public function getArea():Number
    {
        // La formula è Pi volte il quadrato del raggio.
        return Math.PI * Math.pow((width / 2), 2);
    }
```

```
    public function getCircumference():Number
    {
        // La formula è Pi volte il diametro.
        return Math.PI * width;
    }
  }
}
```

Il codice seguente, posizionato in un fotogramma chiave sul fotogramma 1 del documento Flash, crea un'istanza del simbolo e la visualizza sullo schermo:

```
var c:Circle = new Circle();
addChild(c);
trace(c.width);
trace(c.height);
trace(c.getArea());
trace(c.getCircumference());
```

Questo codice illustra la creazione di istanze basata su ActionScript come alternativa al trascinamento delle singole risorse sullo stage. Crea un cerchio che ha tutte le proprietà di un clip filmato e i metodi personalizzati definiti nella classe Circle. L'esempio presentato è molto semplice, ma il simbolo della libreria può specificare un numero infinito di proprietà e metodi nella propria classe.

La creazione di istanze basata su ActionScript è uno strumento molto potente, poiché consente di creare dinamicamente grandi quantità di istanze che sarebbe noioso organizzare manualmente. Inoltre è molto flessibile, poiché consente di personalizzare le proprietà di ogni istanza nel momento in cui quest'ultima viene creata. Questi vantaggi sono evidenti se ad esempio di utilizza un ciclo per creare dinamicamente diverse istanze di Circle. Con il simbolo e la classe Circle descritti in precedenza nella libreria del documento Flash, collocare il codice seguente in un fotogramma chiave sul fotogramma 1:

```
import flash.geom.ColorTransform;

var totalCircles:uint = 10;
var i:uint;
for (i = 0; i < totalCircles; i++)
{
  // Crea una nuova istanza Circle.
  var c:Circle = new Circle();
  // Posiziona la nuova istanza Circle su una coordinata x che distanzia
  // i cerchi in modo uniforme sullo stage.
  c.x = (stage.stageWidth / totalCircles) * i;
  // Posiziona l'istanza Circle nel centro verticale dello stage.
  c.y = stage.stageHeight / 2;
  // Imposta un colore casuale per l'istanza Circle
  c.transform.colorTransform = getRandomColor();
  // Aggiunge l'istanza Circle alla linea temporale corrente.
  addChild(c);
}
```

```
function getRandomColor():ColorTransform
{
  // Genera valori casuali per i canali di colore rosso, verde e blu.
  var red:Number = (Math.random() * 512) - 255;
  var green:Number = (Math.random() * 512) - 255;
  var blue:Number = (Math.random() * 512) - 255;

  // Crea e restituisce un oggetto ColorTransform con i colori casuali.
  return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

L'esempio precedente mostra come creare e personalizzare rapidamente più istanze di un simbolo utilizzando il codice. Ogni istanza viene posizionata in base al conteggio corrente all'interno del ciclo, e a ogni istanza viene assegnato un colore casuale impostandone la proprietà `transform` (che Circle eredita estendendo la classe MovieClip).

Caricamento di un file SWF esterno

In ActionScript 3.0, i file SWF vengono caricati mediante la classe Loader. Per caricare un file SWF esterno, ActionScript deve eseguire quattro operazioni:

1. Creare un nuovo oggetto URLRequest con l'url del file.
2. Creare un nuovo oggetto Loader.
3. Chiamare il metodo `load()` dell'oggetto Loader, passando l'istanza URLRequest come parametro.
4. Chiamare il metodo `addChild()` su un contenitore di un oggetto di visualizzazione (ad esempio, la linea temporale principale di un documento Flash) per aggiungere l'istanza Loader all'elenco di visualizzazione.

Il codice dovrebbe essere simile al seguente:

```
var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
  externalSwf.swf");
var loader:Loader = new Loader()
loader.load(request);
addChild(loader);
```

Questo stesso codice può essere utilizzato per caricare un file di immagine esterno (JPEG, GIF o PNG) specificando l'url del file di immagine anziché quello di un file SWF. Un file SWF, a differenza di un file di immagine, può contenere codice ActionScript. Pertanto, benché il processo di caricamento di un file SWF sia identico al caricamento di un'immagine, quando si carica un file SWF esterno, sia il file SWF che effettua il caricamento che il file SWF che viene caricato devono trovarsi all'interno della stessa funzione di sicurezza sandbox se si prevede di utilizzare ActionScript per comunicare in qualche modo con il file SWF esterno. Inoltre, se il file SWF esterno contiene delle classi che condividono lo stesso spazio dei nomi delle classi nel file SWF in fase di caricamento, può essere necessario creare un nuovo dominio applicazione per il file SWF caricato per evitare eventuali conflitti tra gli spazi dei nomi. Per ulteriori informazioni sulla sicurezza e i domini applicazione, vedere "Uso della classe ApplicationDomain" a pagina 745 e "Caricamento di file SWF e di immagine" a pagina 828.

Quando il file SWF esterno è stato caricato correttamente, è possibile accedervi mediante la proprietà `Loader.content`. Se il file SWF esterno è pubblicato per ActionScript 3.0, si tratterà di un clip filmato o di uno sprite, a seconda della classe che estende.

Considerazioni sul caricamento di un file SWF di una versione precedente

Se il file SWF esterno è stato pubblicato con una versione precedente di ActionScript, è necessario prendere in considerazione alcune importanti limitazioni. A differenza di un file SWF di ActionScript 3.0 che viene eseguito in AVM2 (ActionScript Virtual Machine 2), un file SWF pubblicato per ActionScript 1.0 o 2.0 viene eseguito in AVM1 (ActionScript Virtual Machine 1).

Quando un file SWF AVM1 è stato caricato correttamente, l'oggetto caricato (la proprietà `Loader.content`) è un oggetto AVM1Movie. Un'istanza AVM1Movie è diversa da un'istanza MovieClip. Si tratta di un oggetto di visualizzazione ma, a differenza di un clip filmato, non include metodi o proprietà correlati alle linee temporali. Il file SWF AVM2 principale non ha accesso alle proprietà, ai metodi o agli oggetti dell'oggetto AVM1Movie caricato.

Sono previste altre limitazioni per un file SWF AVM1 caricato da un file SWF AVM2: Per maggiori dettagli, consultare la sezione relativa alla classe AVM1Movie nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Esempio: RuntimeAssetsExplorer

La funzionalità Esporta per ActionScript può rivelarsi particolarmente vantaggiosa per le librerie che possono essere utili in più progetti. I simboli che sono stati esportati in ActionScript sono disponibili non solo per tale file SWF ma per qualunque file SWF che si trova all'interno della stessa funzione di sicurezza sandbox che lo carica. In tal modo, un solo documento Flash può generare un file SWF che viene definito al solo scopo di contenere delle risorse grafiche. Questa tecnica è particolarmente utile per i progetti di grandi dimensioni in cui i designer grafici lavorano in parallelo agli sviluppatori; questi ultimi creano un file SWF "wrapper" che successivamente carica il file SWF delle risorse grafiche in fase di runtime. Questo metodo può essere utilizzato per mantenere una serie di versioni di un file in cui le risorse grafiche non dipendono dall'avanzamento dello sviluppo della programmazione.

L'applicazione RuntimeAssetsExplorer carica qualunque file SWF che sia una sottoclasse di RuntimeAsset e consente di sfogliare le risorse disponibili su tale file SWF. Questa situazione è mostrata nell'esempio seguente:

- Caricamento di un file SWF esterno mediante `Loader.load()`
- Creazione dinamica di un simbolo di libreria esportato per ActionScript
- Controllo ActionScript della riproduzione di MovieClip

Prima di cominciare, si noti che ciascuno dei file SWF deve trovarsi nella stessa funzione di sicurezza sandbox. Per ulteriori informazioni, vedere "Funzioni di sicurezza sandbox" a pagina 822.

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione RuntimeAssetsExplorer sono disponibili nella cartella Samples/RuntimeAssetsExplorer. L'applicazione è costituita dai file seguenti:

File	Descrizione
RuntimeAssetsExample.mxml oppure RuntimeAssetsExample.fla	L'interfaccia utente dell'applicazione per Flex (MXML) o Flash (FLA).
GeometricAssets.as	Una classe di esempio che implementa l'interfaccia RuntimeAsset.
GeometricAssets.fla	Un file FLA collegato a una classe GeometricAssets (classe documento del file FLA) che contiene i simboli che vengono esportati per ActionScript.

File	Descrizione
com/example/programmingas3/runtimeassetsexplorer/RuntimeLibrary.as	Un'interfaccia che definisce i metodi richiesti previsti per tutti i file SWF delle risorse di runtime che verranno caricati nel contenitore dell'applicazione di esplorazione.
com/example/programmingas3/runtimeassetsexplorer/AnimatingBox.as	La classe del simbolo di libreria, che ha la forma di una scatola rotante.
com/example/programmingas3/runtimeassetsexplorer/AnimatingStar.as	La classe del simbolo di libreria, che ha la forma di una stella rotante.

Definizione di un'interfaccia per la libreria di runtime

Per consentire all'applicazione di esplorazione di interagire correttamente con una libreria SWF, la struttura delle librerie delle risorse di runtime deve essere formalizzata. A tale scopo, viene creata un'interfaccia, che è simile a una classe nel senso che rappresenta una base di partenza dei metodi che contrassegnano una struttura prevista, ma che a differenza di una classe non contiene i corpi dei metodi stessi. L'interfaccia fornisce un modo per consentire la comunicazione tra la libreria di runtime e l'applicazione di esplorazione. Ogni file SWF di risorse di runtime che viene caricato nel browser implementa questa interfaccia. Per ulteriori informazioni sulle interfacce e la loro utilità, vedere "Interfacce" a pagina 170.

L'interfaccia RuntimeLibrary è molto semplice: è richiesta esclusivamente una funzione in grado di fornire all'applicazione di esplorazione un array di percorsi di classe per i simboli da esportare e disponibili nella libreria di runtime. A tale scopo, l'interfaccia contiene un solo metodo: `getAssets()`.

```
package com.example.programmingas3.runtimeassetexplorer
{
  public interface RuntimeLibrary
  {
    function getAssets():Array;
  }
}
```

Creazione del file SWF della libreria di risorse

Se si definisce l'interfaccia RuntimeLibrary, è possibile creare più file SWF di librerie di risorse che possono essere caricati in un altro file SWF. La creazione di una singola libreria SWF richiede quattro operazioni:

- Creazione di una classe per il file SWF della libreria di risorse
- Creazione delle classi per le singole risorse contenute nella libreria
- Creazione delle risorse grafiche vere e proprie
- Associazione degli elementi grafici alle classi e pubblicazione del file SWF della libreria

Creazione di una classe per l'implementazione dell'interfaccia RuntimeLibrary

A questo punto, verrà creata la classe GeometricAssets che implementa l'interfaccia RuntimeLibrary. Si tratta della classe document del file FLA. Il codice di questa classe è molto simile all'interfaccia RuntimeLibrary; la differenza consiste nel fatto che nella definizione della classe il metodo `getAssets()` ha un corpo del metodo.

```
package
{
  import flash.display.Sprite;
  import com.example.programmingas3.runtimeassetexplorer.RuntimeLibrary;

  public class GeometricAssets extends Sprite implements RuntimeLibrary
  {
    public function GeometricAssets() {

    }
    public function getAssets():Array {
      return [
  "com.example.programmingas3.runtimeassetexplorer.AnimatingBox",

  "com.example.programmingas3.runtimeassetexplorer.AnimatingStar" ];
    }
  }
}
```

Nel caso sia necessario creare una seconda libreria di runtime, è possibile creare un altro file FLA in base a un'altra classe (ad esempio, AnimationAssets) che fornisce la propria implementazione `getAssets()`.

Creazione delle classi per ciascuna risorsa di MovieClip

Nell'esempio in questione, viene semplicemente estesa la classe MovieClip senza aggiungere alcuna funzionalità alle risorse personalizzate. Il codice successivo per AnimatingStar è simile a quello di AnimatingBox:

```
package com.example.programmingas3.runtimeassetexplorer
{
  import flash.display.MovieClip;

  public class AnimatingStar extends MovieClip
  {
    public function AnimatingStar() {
    }
  }
}
```

Pubblicazione della libreria

A questo punto verranno collegate le risorse basate su MovieClip alla nuova classe creando un nuovo file FLA e immettendo GeometricAssets nel campo Classe documento della finestra di ispezione Proprietà. Ai fini dell'esempio, verranno create due forme molto elementari che utilizzano un'interpolazione di linea temporale per creare una rotazione oraria lungo 360 fotogrammi. Entrambi i simboli `animatingBox` e `animatingStar` sono impostati su Esporta per ActionScript e hanno il campo Classe impostato sui rispettivi percorsi di classe specificati nell'implementazione `getAssets()`. La classe di base predefinita di `flash.display.MovieClip` rimane, dal momento che l'obiettivo è estendere i metodi MovieClip standard.

Una volta specificate le impostazioni di esportazione del simbolo, pubblicare il file FLA. Si ottiene la prima libreria di runtime. Questo file SWF può essere caricato in un altro SWF AVM2 per rendere i simboli AnimatingBox e AnimatingStar disponibili per il nuovo file SWF.

Caricamento della libreria in un altro file SWF

L'ultima parte della funzione consiste nell'interfaccia utente per l'applicazione di esplorazione delle risorse. In questo esempio, il percorso della libreria di runtime è codificato nella variabile `ASSETS_PATH`. In alternativa, è possibile utilizzare la classe FileReference: ad esempio, per creare un'interfaccia che permetta di individuare un determinato file SWF sul disco rigido.

Quando la libreria di runtime è stata caricata correttamente, viene chiamato il metodo `runtimeAssetsLoadComplete()`.

```
private function runtimeAssetsLoadComplete(event:Event):void
{
  var rl:* = event.target.content;
  var assetList:Array = rl.getAssets();
  populateDropdown(assetList);
  stage.frameRate = 60;
}
```

In questo metodo, la variabile rl rappresenta il file SWF caricato. Il codice chiama il metodo `getAssets()` del file SWF caricato e ottiene l'elenco delle risorse disponibili, che utilizza per compilare un componente ComboBox con un elenco di risorse disponibili chiamando il metodo `populateDropDown()`. A propria volta, questo metodo memorizza il percorso di classe completo di ogni risorsa. Se si fa clic sul pulsante Aggiungi dell'interfaccia utente, viene attivato il metodo `addAsset()`:

```
private function addAsset():void
{
  var className:String = assetNameCbo.selectedItem.data;
  var AssetClass:Class = getDefinitionByName(className) as Class;
  var mc:MovieClip = new AssetClass();
  ...
}
```

che ottiene il percorso di classe della risorsa correntemente selezionata nel componente ComboBox (`assetNameCbo.selectedItem.data`) e utilizza la funzione `getDefinitionByName()` (del pacchetto flash.utils) per ottenere un riferimento reale alla classe della risorsa per poter creare una nuova istanza di tale risorsa.

CAPITOLO 17

Operazioni con il testo

17

In ActionScript 3.0, il testo viene solitamente visualizzato all'interno di un campo di testo. Tuttavia, di tanto in tanto può anche comparire sotto forma di una proprietà di un elemento nell'elenco di visualizzazione (ad esempio, come etichetta di un componente dell'interfaccia). Questo capitolo illustra come utilizzare il contenuto definito dagli script di un campo di testo, il testo dinamico immesso dall'utente proveniente da un file remoto o il testo statico definito in Adobe Flash CS3 Professional. Il programmatore di ActionScript 3.0 può definire dei contenuti specifici per i campi di testo oppure designare l'origine del testo e successivamente impostarne l'aspetto mediante gli stili e i formati. È anche possibile rispondere agli eventi utente quando l'utente immette un testo o fa clic su un collegamento ipertestuale.

Sommario

Nozioni fondamentali sulle operazioni con il testo

Introduzione alle operazioni con il testo

Per visualizzare qualunque testo sullo schermo in Adobe Flash Player si utilizza un'istanza della classe TextField. La classe TextField costituisce la base per altri componenti basati su testo, ad esempio i componenti TextArea o i componenti TextInput, che sono forniti nella struttura Adobe Flex e nell'ambiente di creazione di Flash. Per ulteriori informazioni sull'uso dei componenti di testo nell'ambiente di creazione di Flash, vedere "Informazioni sui campi di testo" nella guida *Uso di Flash*.

Il contenuto dei campi di testo può essere specificato in precedenza nel file SWF, caricato da un'origine esterna come un file di testo o un database oppure immesso dagli utenti che interagiscono con l'applicazione. All'interno di un campo di testo, il testo può comparire sotto forma di rendering HTML con incorporate le immagini. Una volta definita un'istanza di un campo di testo, è possibile utilizzare le classi del pacchetto flash.text (ad esempio, TextFormat e StyleSheet) per controllare l'aspetto del testo. Il pacchetto flash.text contiene quasi tutte le classi correlate alla creazione, alla gestione e alla formattazione del testo in ActionScript.

È possibile formattare il testo definendo la formattazione con un oggetto TextFormat e assegnando l'oggetto al campo di testo. Se il campo di testo contiene testo HTML, è possibile applicare un oggetto StyleSheet al campo di testo per assegnare gli stili a specifiche porzioni del contenuto del campo di testo. L'oggetto TextFormat o StyleSheet contiene le proprietà che definiscono l'aspetto del testo, quali il colore, le dimensioni e il peso. L'oggetto TextFormat assegna le proprietà a tutto il contenuto all'interno di un campo di testo, oppure a un intervallo di testo. Ad esempio, all'interno dello stesso campo di testo, una frase può essere in grassetto rosso e la successiva in blu corsivo.

Per ulteriori informazioni sui formati di testo, vedere "Assegnazione dei formati di testo" a pagina 559.

Per ulteriori informazioni sul testo HTML nei campi di testo, vedere "Visualizzazione del testo HTML" a pagina 553.

Per ulteriori informazioni su fogli di stile, vedere "Applicazione dei fogli di stile CSS" a pagina 560.

Oltre alle classi del pacchetto flash.text, è possibile utilizzare la classe flash.events.TextEvent per rispondere alle azioni relative al testo eseguite dall'utente.

Operazioni comuni con il testo

Le seguenti operazioni comuni con il testo sono descritte in questo capitolo:

- Modifica del contenuto di un campo di testo
- Uso dell'HTML nei campi di testo
- Uso delle immagini nei campi di testo
- Selezione del testo e operazioni con il testo selezionato dall'utente
- Rilevamento dell'input di testo
- Limitazione dell'input di testo
- Applicazione della formattazione e degli stili CSS al testo
- Regolazione della precisione, dello spessore e dell'antialiasing della visualizzazione del testo
- Accesso e uso dei campi di testo statici da ActionScript

Concetti e termini importanti

L'elenco di riferimento seguente contiene dei termini importanti che vengono citati in questo capitolo:

- CSS (Cascading Style Sheets): una sintassi standard per la definizione degli stili e della formattazione del contenuto strutturato in formato XML (o HTML).
- Carattere dispositivo: un carattere installato sul sistema dell'utente.
- Campo di testo dinamico: un campo di testo il cui contenuto può essere modificato da ActionScript ma non dall'input dell'utente.
- Carattere incorporato: un carattere i cui dati del contorno carattere sono memorizzati nel file SWF dell'applicazione.
- Testo HTML: contenuto testuale immesso in un campo di testo mediante ActionScript e che include tag di formattazione HTML insieme al contenuto testuale vero e proprio.
- Campo di testo di input: un campo di testo il cui contenuto può essere modificato dall'input dell'utente o da ActionScript.
- Campo di testo statico: un campo di testo statico creato nello strumento di creazione di Flash, il cui contenuto non può essere modificato mentre il file SWF è in esecuzione.
- Metriche righe di testo: misure delle dimensioni delle varie parti del contenuto testuale in un campo di testo, quali la linea di base del testo, l'altezza della parte superiore dei caratteri, la dimensioni dei discendenti (la parte di alcune lettere minuscole che si estende al di sotto della linea di base) e così via.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché questo capitolo descrive le operazioni con i campi di testo in ActionScript, tutti gli esempi di codice riportati prevedono la manipolazione di un oggetto TextField, che potrebbe essere stato sia creato e posizionato sullo stage nello strumento di creazione di Flash che creato mediante ActionScript. La prova degli esempi prevede la visualizzazione del risultato in Flash Player per vedere gli effetti del codice sul campo di testo.

Gli esempi contenuti in questo capitolo si dividono in due gruppi. Un tipo di esempio manipola un oggetto TextField senza creare l'oggetto esplicitamente. Per provare questi esempi di codice:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.
4. Utilizzare lo strumento Testo per creare un campo di testo dinamico sullo stage.
5. Con il campo di testo selezionato, nella finestra di ispezione Proprietà assegnargli un nome istanza. Questo nome deve corrispondere al nome utilizzato per il campo di testo nell'esempio di codice, ad esempio, se il codice manipola un campo di testo chiamato `myTextField`, è necessario denominare `myTextField` anche l'istanza del campo di testo.
6. Eseguire il programma selezionando Controllo > Prova filmato.

   Sullo schermo vengono visualizzati i risultati della manipolazione del campo di testo secondo quanto specificato nell'esempio di codice.

L'altro tipo di esempio di codice presente in questo capitolo consiste nella definizione di una classe da utilizzare come classe documento per il file SWF. Negli esempi di questo tipo, un'istanza TextField viene creata dal codice di esempio, in modo che non sia necessario crearne una separatamente. Per provare questo tipo di codice:

1. Creare un documento Flash vuoto e salvarlo nel computer.
2. Creare un nuovo file ActionScript e salvarlo nella stessa directory del documento Flash. Il nome file deve corrispondere al nome della classe presente nell'esempio di codice. Ad esempio, se l'esempio di codice definisce una classe chiamata TextFieldTest, per salvare il file ActionScript, utilizzare il nome TextFieldTest.as.
3. Copiare l'esempio di codice nel file ActionScript e salvare il file.
4. Nel documento Flash, fare clic in una parte vuota dello stage oppure dell'area di lavoro per attivare la finestra di ispezione Proprietà del documento.
5. Nella finestra di ispezione Proprietà, nel campo Classe documento inserire il nome della classe ActionScript copiata dal testo.

6. Eseguire il programma selezionando Controllo > Prova filmato.

   I risultati dell'esempio vengono visualizzati sullo schermo.

Per informazioni su altre tecniche per la prova degli esempi di codice, vedere "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Visualizzazione del testo

Benché strumenti di creazione come Adobe Flex Builder e lo strumento di creazione di Flash forniscano diverse opzioni per la visualizzazione del testo, tra cui i componenti e gli strumenti di testo, il metodo principale per impostare a livello di codice la visualizzazione del testo è rappresentato dai campi di testo.

Tipi di testo

Il tipo di testo all'interno di un campo di testo è caratterizzato dalla propria origine:

- Testo dinamico

  Il testo dinamico comprende i contenuti che vengono caricati da un'origine esterna, quale un file di testo, un file XML o un servizio Web remoto. Per ulteriori informazioni, vedere "Tipi di testo" a pagina 551.

- Testo di input

  Il testo di input è rappresentato dal testo immesso dall'utente o dal testo dinamico modificabile dall'utente. È possibile impostare un foglio di stile per la formattazione del testo di input oppure utilizzare la classe flash.text.TextFormat per assegnare delle proprietà al campo di testo per il contenuto di input. Per ulteriori informazioni, vedere "Rilevamento dell'input di testo" a pagina 557.

- Testo statico

  Il testo statico viene creato solo mediante lo strumento di creazione di Flash. Non è possibile creare un'istanza di testo statico mediante ActionScript: Tuttavia, è possibile utilizzare le classi ActionScript come StaticText e TextSnapshot per manipolare un'istanza di testo statico esistente. Per ulteriori informazioni, vedere "Operazioni con il testo statico" a pagina 566.

Modifica del contenuto di un campo di testo

Il testo dinamico può essere definito assegnando una stringa alla proprietà `flash.text.TextField.text`. È possibile assegnare una stringa direttamente alla proprietà, come segue:

```
myTextField.text = “Hello World”;
```

Inoltre, alla proprietà `text` è possibile assegnare un valore tratto da una variabile definita nello script, come indicato nell’esempio seguente:

```
package
{
  import flash.display.Sprite;
  import flash.text.*;

  public class TextWithImage extends Sprite
  {
    private var myTextBox:TextField = new TextField();
    private var myText:String = "Hello World";

    public function TextWithImage()
    {
      addChild(myTextBox);
      myTextBox.text = myText;
    }
  }
}
```

In alternativa, alla proprietà `text` è possibile assegnare un valore tratto da una variabile remota. Per caricare i valori di testo da origini remote sono disponibili tre opzioni:

- Le classi flash.net.URLLoader e flash.net.URLRequest caricano le variabili per il testo da una posizione locale o remota.
- L’attributo `FlashVars` è incorporato nella pagina HTML che include il file SWF e può contenere dei valori delle variabili di testo.
- La classe flash.net.SharedObject gestisce la memorizzazione persistente dei valori. Per ulteriori informazioni, vedere “Memorizzazione di dati locali” a pagina 711.

Visualizzazione del testo HTML

La classe flash.text.TextField ha una proprietà `htmlText` che è possibile utilizzare per identificare la stringa di testo come stringa che contiene tag HTML per la formattazione del contenuto. Come nell'esempio seguente, è necessario assegnare il valore di tipo stringa alla proprietà `htmlText` (non alla proprietà `text`) affinché Flash Player esegua il rendering del testo come HTML:

```
var myText:String = "<p>This is <b>some</b> content to <i>render</i> as
  <u>HTML</u> text.</p>";
myTextBox.htmlText = myText;
```

Flash Player supporta un sottoinsieme di tag ed entità HTML per la proprietà `htmlText`. La descrizione della proprietà `flash.text.TextField.htmlText` nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0* fornisce informazioni dettagliate sui tag e le entità HTML supportate.

Una volta designato il contenuto mediante la proprietà `htmlText`, è possibile utilizzare i fogli di stile o il tag `textformat` per gestire la formattazione del contenuto. Per ulteriori informazioni, vedere "Formattazione del testo" a pagina 559.

Uso delle immagini nei campi di testo

Un altro vantaggio della visualizzazione dei contenuti sotto forma di testo HTML consiste nel fatto che nel campo di testo è possibile includere delle immagini. È possibile fare riferimento a un'immagine (locale o remota) mediante il tag `img` e farla visualizzare all'interno del campo di testo associato.

L'esempio seguente crea un campo di testo di nome `myTextBox` e include un'immagine JPG di un occhio, memorizzata nella stessa directory del file SWF, all'interno del testo visualizzato:

```
package
{
  import flash.display.Sprite;
  import flash.text.*;

  public class TextWithImage extends Sprite
  {
    private var myTextBox:TextField;
    private var myText:String = "<p>This is <b>some</b> content to
  <i>test</i> and <i>see</i></p><p><img src='eye.jpg' width='20'
  height='20'></p><p>what can be rendered.</p><p>You should see an eye
  image and some <u>HTML</u> text.</p>";
```

```
    public function TextWithImage()
    {
      myTextBox.width = 200;
      myTextBox.height = 200;
      myTextBox.multiline = true;
      myTextBox.wordWrap = true;
      myTextBox.border = true;

      addChild(myTextBox);
      myTextBox.htmlText = myText;
    }
  }
}
```

Il tag `img` supporta i file JPEG, GIF, PNG e SWF.

Scorrimento del testo in un campo di testo

In molti casi, il testo risulta più lungo del campo di testo che lo visualizza. In altri, un campo di immissione testo consente all'utente di immettere più testo di quello visualizzabile in una sola volta. Per gestire i contenuti particolarmente lunghi (sia in verticale che in orizzontale), è possibile utilizzare le proprietà relative allo scorrimento della classe flash.text.TextField.

Alcune delle proprietà relative allo scorrimento sono `TextField.scrollV`, `TextField.scrollH` e `maxScrollV` e `maxScrollH`. Utilizzarle per rispondere a eventi quali un clic del mouse o la pressione di un tasto della tastiera.

Nell'esempio seguente viene creato un campo di testo che ha una dimensione fissa e contiene più testo di quello che il campo è in grado di visualizzare in una sola volta. Quando l'utente fa clic sul campo di testo, il testo viene fatto scorrere in verticale.

```
package
{
  import flash.display.Sprite;
  import flash.text.*;
  import flash.events.MouseEvent;

  public class TextScrollExample extends Sprite
  {
    private var myTextBox:TextField = new TextField();
    private var myText:String = "Hello world and welcome to the show. It's
  really nice to meet you. Take your coat off and stay a while. OK, show is
  over. Hope you had fun. You can go home now. Don't forget to tip your
  waiter. There are mints in the bowl by the door. Thank you. Please come
  again.";
```

```
    public function TextScrollExample()
    {
      myTextBox.text = myText;
      myTextBox.width = 200;
      myTextBox.height = 50;
      myTextBox.multiline = true;
      myTextBox.wordWrap = true;
      myTextBox.background = true;
      myTextBox.border = true;

      var format:TextFormat = new TextFormat();
      format.font = "Verdana";
      format.color = 0xFF0000;
      format.size = 10;

      myTextBox.defaultTextFormat = format;
      addChild(myTextBox);
      myTextBox.addEventListener(MouseEvent.MOUSE_DOWN, mouseDownScroll);
    }

    public function mouseDownScroll(event:MouseEvent):void
    {
      myTextBox.scrollV++;
    }
  }
}
```

Selezione ed elaborazione del testo

È possibile selezionare il testo dinamico o di input. Dal momento che le proprietà e i metodi di selezione del testo della classe TextField utilizzano le posizioni di indice per impostare l'intervallo di testo da elaborare, è possibile selezionare a livello di codice il testo dinamico o di input anche se non se ne conosce il contenuto.

NOTA

Nello strumento di creazione di Flash, se si sceglie l'opzione selezionabile su un campo di testo statico, il campo di testo che viene esportato e posizionato nell'elenco di visualizzazione è un campo di testo normale dinamico.

Selezione del testo

La proprietà `flash.text.TextField.selectable` è `true` per impostazione predefinita ed è possibile selezionare a livello di codice il testo mediante il metodo `setSelection()`.

Ad esempio, è possibile impostare un testo specifico all'interno di un campo di testo che è possibile selezionare quando l'utente fa clic sul campo di testo:

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN
  ALL CAPS is selected.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, selectText);

function selectText(event:MouseEvent):void
{
  myTextField.setSelection(49, 65);
}
```

In modo analogo, se si desidera che il testo all'interno di un campo di testo venga selezionato quando il testo viene inizialmente visualizzato, creare una funzione di gestore di eventi che viene chiamata non appena il campo di testo viene aggiunto all'elenco di visualizzazione.

Rilevamento del testo selezionato dall'utente

Le proprietà `selectionBeginIndex` e `selectionEndIndex` della classe TextField (che sono di sola lettura e non possono essere impostate per selezionare il testo a livello di codice) possono essere utilizzate anche per acquisire ciò che l'utente ha correntemente selezionato. Inoltre, i campi di testo di input possono utilizzare la proprietà `caretIndex`.

Il codice seguente, ad esempio, traccia i valori di indice del testo selezionato dall'utente:

```
var myTextField:TextField = new TextField();
myTextField.text = "Please select the TEXT IN ALL CAPS to see the index
  values for the first and last letters.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.MOUSE_UP, selectText);

function selectText(event:MouseEvent):void
{
  trace("First letter index position: " + myTextField.selectionBeginIndex);
  trace("Last letter index position: " + myTextField.selectionEndIndex);
}
```

È possibile applicare alla selezione una raccolta di proprietà di oggetto TextFormat per modificare l'aspetto del testo. Per ulteriori informazioni sull'applicazione di una raccolta di proprietà TextFormat al testo selezionato, vedere "Formattazione di intervalli di testo all'interno di un campo di testo" a pagina 563.

Rilevamento dell'input di testo

Per impostazione predefinita, la proprietà `type` di un campo di testo è impostata su `dynamic`. Se si imposta la proprietà `type` su `input` mediante la classe TextFieldType, è possibile raccogliere l'input dell'utente e salvare il valore per utilizzarlo in altre parti dell'applicazione. I campi di testo di input sono utili per i form e per tutte le applicazioni che richiedono all'utente di definire un valore di testo da utilizzare altrove nel programma.

Ad esempio, il codice seguente crea un campo di testo di input denominato `myTextBox`. Quando l'utente immette il testo nel campo, viene attivato l'evento `textInput`. Un gestore di eventi di nome `textInputCapture` acquisisce la stringa di testo immessa e le assegna una variabile. Flash Player visualizza il nuovo testo in un altro campo di testo, di nome `myOutputBox`.

```
package
{
  import flash.display.Sprite;
  import flash.display.Stage;
  import flash.text.*;
  import flash.events.*;

  public class CaptureUserInput extends Sprite
  {
    private var myTextBox:TextField = new TextField();
    private var myOutputBox:TextField = new TextField();
    private var myText:String = "Type your text here.";

    public function CaptureUserInput()
    {
      captureText();
    }

    public function captureText():void
    {
      myTextBox.type = TextFieldType.INPUT;
      myTextBox.background = true;
      addChild(myTextBox);
      myTextBox.text = myText;
      myTextBox.addEventListener(TextEvent.TEXT_INPUT, textInputCapture);
    }
```

```
    public function textInputCapture(event:TextEvent):void
    {
      var str:String = myTextBox.text;
      createOutputBox(str);
    }

    public function createOutputBox(str:String):void
    {
      myOutputBox.background = true;
      myOutputBox.x = 200;
      addChild(myOutputBox);
      myOutputBox.text = str;
    }

  }
}
```

Limitazione dell'input di testo

Dal momento che i campi di testo di input vengono spesso utilizzati per i form o le finestre di dialogo nelle applicazioni, è possibile limitare i tipi di carattere che l'utente può immettervi o persino mantenere nascosto il testo (ad esempio, quando si digita una password). La classe flash.text.TextField ha una proprietà `displayAsPassword` e una proprietà `restrict` che è possibile impostare per controllare l'input dell'utente.

La proprietà `displayAsPassword` si limita a nascondere il testo (visualizzandolo sotto forma di una serie di asterischi) mentre l'utente lo digita. Quando `displayAsPassword` è impostata su `true`, i comandi Taglia e Copia e i rispettivi tasti di scelta rapida non funzionano. Come illustra l'esempio seguente, l'assegnazione della proprietà `displayAsPassword` è uguale a quella di altre proprietà (ad esempio, quelle relative a sfondo e colore):

```
myTextBox.type = TextFieldType.INPUT;
myTextBox.background = true;
myTextBox.displayAsPassword = true;
addChild(myTextBox);
```

La proprietà `restrict` è un po' più complessa, dal momento che richiede di specificare quali caratteri l'utente è autorizzato a digitare in un campo di testo di input. È possibile consentire specificamente lettere, numeri o intervalli di lettere, numeri e caratteri. Il codice seguente consente all'utente di immettere solo lettere maiuscole (e non numeri o caratteri speciali) nel campo di testo:

```
myTextBox.restrict = "A-Z";
```

ActionScript 3.0 utilizza il trattino per definire gli intervalli e l'accento circonflesso per definire i caratteri esclusi. Per ulteriori informazioni sulla definizione delle limitazioni in un campo di testo di input, vedere la voce relativa alla proprietà `flash.text.TextField.restrict` nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Formattazione del testo

Sono disponibili diverse opzioni per formattare a livello di codice la visualizzazione del testo. È possibile impostare le proprietà direttamente sull'istanza TextField; ad esempio, le proprietà `TextFIeld.thickness`, `TextField.textColor` e `TextField.textHeight`. Oppure è possibile designare il contenuto del campo di testo mediante la proprietà `htmlText` e l'uso dei tag HTML supportati, quali `b`, `i` e `u`. Ma è anche possibile applicare degli oggetti TextFormat ai campi di testo che contengono testo semplice oppure oggetti StyleSheet ai campi di testo che contengono la proprietà `htmlText`. L'utilizzo degli oggetti TextFormat e StyleSheet fornisce il livello massimo di controllo e coerenza sull'aspetto del testo in tutta l'applicazione. È possibile definire un oggetto TextFormat o StyleSheet e applicarlo a molti o a tutti i campi di testo presenti nell'applicazione.

Assegnazione dei formati di testo

È possibile utilizzare la classe TextFormat per impostare una serie di proprietà di visualizzazione del testo e applicarle all'intero contenuto di un oggetto TextField o a un intervallo di testo.

L'esempio seguente applica un oggetto TextFormat a un intero oggetto TextField e applica un altro oggetto TextFormat a un intervallo di testo all'interno di tale oggetto TextField:

```
var tf:TextField = new TextField();
tf.text = "Hello Hello";

var format1:TextFormat = new TextFormat();
format1.color = 0xFF0000;

var format2:TextFormat = new TextFormat();
format2.font = "Courier";

tf.setTextFormat(format1);
var startRange:uint = 6;
tf.setTextFormat(format2, startRange);

addChild(tf);
```

Il metodo `TextField.setTextFormat()` agisce solo sul testo che è già visualizzato nel campo di testo. Se il contenuto dell'oggetto TextField cambia, è possibile che l'applicazione debba chiamare ancora il metodo `TextField.setTextFormat()` per applicare nuovamente la formattazione. È anche possibile impostare la proprietà `defaultTextFormat` dell'oggetto TextField per specificare il formato da utilizzare per il testo immesso dall'utente.

Applicazione dei fogli di stile CSS

I campi di testo possono contenere testo semplice o testo in formato HTML. Il testo semplice viene memorizzato nella proprietà `text` dell'istanza, mentre il testo HTML viene memorizzato nella proprietà `htmlText`.

È possibile utilizzare le dichiarazioni di stile CSS per definire gli stili di testo che è possibile applicare a molti campi di testo diversi. La dichiarazione di stile CSS può essere creata nel codice dell'applicazione oppure caricata in fase di runtime da un file CSS esterno.

La classe flash.text.StyleSheet gestisce gli stili CSS. La classe StyleSheet riconosce un set limitato di proprietà CSS. Per un elenco dettagliato delle proprietà di stile supportate dalla classe StyleSheet, consultare la sezione relativa a flash.textStylesheet nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

Come mostra l'esempio seguente, è possibile creare dei fogli di stile CSS (Cascading Style Sheets) nel codice e applicarli al testo HTML mediante un oggetto StyleSheet:

```
var style:StyleSheet = new StyleSheet();

var styleObj:Object = new Object();
styleObj.fontSize = "bold";
styleObj.color = "#FF0000";
style.setStyle(".darkRed", styleObj);

var tf:TextField = new TextField();
tf.styleSheet = style;
tf.htmlText = "<span class = 'darkRed'>Red</span> apple";

addChild(tf);
```

Una volta creato l'oggetto StyleSheet, il codice di esempio crea un oggetto semplice che contiene un set di proprietà di dichiarazioni di stile. Quindi, chiama il metodo `StyleSheet.setStyle()`, che aggiunge il nuovo stile al foglio di stile di nome ".darkred". Quindi, applica la formattazione del foglio di stile assegnando l'oggetto StyleSheet alla proprietà `styleSheet` dell'oggetto TextField.

Affinché gli stili CSS abbiano effetto, i fogli di stile devono essere applicati all'oggetto TextField prima di impostare la proprietà `htmlText`.

Per impostazione in fase di progettazione, un campo di testo al quale è applicato un foglio di stile non è modificabile. Se si assegna a un campo di testo di input un foglio di stile, il campo di testo mostra le proprietà del foglio di stile ma il campo di testo non consente agli utenti di immettervi del nuovo testo. Inoltre, non è possibile utilizzare le seguenti API ActionScript su un campo di testo con un foglio di stile:

- Il metodo `TextField.replaceText()`
- Il metodo `TextField.replaceSelectedText()`
- La proprietà `TextField.defaultTextFormat`
- Il metodo `TextField.setTextFormat()`

Se a un campo di testo è stato assegnato un foglio di stile ma successivamente la proprietà `TextField.styleSheet` viene impostata su `null`, il contenuto dei tag `TextField.text` e `TextField.htmlText` aggiunge dei tag e degli attributi al loro contenuto per incorporare la formattazione del foglio di stile precedentemente assegnato. Per conservare la proprietà `htmlText` originale, salvarla in una variabile prima di impostare il foglio di stile su `null`.

Caricamento di un file CSS esterno

L'uso dei fogli di stile CSS per la formattazione è particolarmente potente quando è possibile caricare le informazioni dei CSS da un file esterno in fase di runtime. Quando i dati CSS sono esterni all'applicazione, è possibile modificare lo stile visivo del testo nell'applicazione senza dover modificare il codice di origine ActionScript 3.0. Una volta implementata l'applicazione, è possibile modificare un file CSS esterno per modificare l'aspetto dell'applicazione senza dover implementare di nuovo il file SWF dell'applicazione.

Il metodo `StyleSheet.parseCSS()` converte una stringa che contiene i dati CSS in dichiarazioni di stile nell'oggetto StyleSheet. L'esempio seguente mostra come leggere un file CSS esterno e applicarne le dichiarazioni di stile a un oggetto TextField.

Innanzi tutto, ecco il contenuto del file CSS da caricare e denominato example.css:

```
p {
  font-family: Times New Roman, Times, _serif;
  font-size: 14;
}

h1 {
  font-family: Arial, Helvetica, _sans;
  font-size: 20;
  font-weight: bold;
}

.bluetext {
  color: #0000CC;
}
```

Quindi, ecco il codice ActionScript per una classe che carica il file example.css e applica gli stili al contenuto di TextField:

```
package
{
  import flash.display.Sprite;
  import flash.events.Event;
  import flash.net.URLLoader;
  import flash.net.URLRequest;
  import flash.text.StyleSheet;
  import flash.text.TextField;
  import flash.text.TextFieldAutoSize;

  public class CSSFormattingExample extends Sprite
  {
    var loader:URLLoader;
    var field:TextField;
    var exampleText:String = "<h1>This is a headline</h1>" +
      "<p>This is a line of text. <span class='bluetext'>" +
      "This line of text is colored blue.</span></p>";

    public function CSSFormattingExample():void
    {
      field = new TextField();
      field.width = 300;
      field.autoSize = TextFieldAutoSize.LEFT;
      field.wordWrap = true;
      addChild(field);

      var req:URLRequest = new URLRequest("example.css");

      loader = new URLLoader();
      loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
      loader.load(req);
    }

    public function onCSSFileLoaded(event:Event):void
    {
      var sheet:StyleSheet = new StyleSheet();
      sheet.parseCSS(loader.data);
      field.styleSheet = sheet;
      field.htmlText = exampleText;
    }
  }
}
```

Quando i dati CSS vengono caricati, il metodo `onCSSFileLoaded()` viene eseguito e chiama il metodo `StyleSheet.parseCSS()` per trasferire le dichiarazioni di stile all'oggetto StyleSheet.

Formattazione di intervalli di testo all'interno di un campo di testo

Un metodo particolarmente utile della classe flash.text.TextField è il metodo `setTextFormat()`. Mediante `setTextFormat()`, è possibile assegnare delle proprietà specifiche a una parte dei contenuti di un campo di testo per rispondere all'input dell'utente, ad esempio i moduli che devono ricordare agli utenti che determinate voci sono obbligatorie o per modificare l'enfasi di una sottosezione di un brano di testo all'interno di un campo di testo mentre un utente seleziona parti del testo.

L'esempio seguente utilizza `TextField.setTextFormat()` su un intervallo di caratteri per modificare l'aspetto di parte del contenuto di `myTextField` quando l'utente fa clic sul campo di testo:

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN
  ALL CAPS changes format.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, changeText);

var myformat:TextFormat = new TextFormat();
myformat.color = 0xFF0000;
myformat.size = 18;
myformat.underline = true;

function changeText(event:MouseEvent):void
{
  myTextField.setTextFormat(myformat, 49, 65);
}
```

Rendering avanzato del testo

ActionScript 3.0 fornisce una vasta gamma di classi nel pacchetto flash.text per controllare le proprietà del testo visualizzato, tra cui i caratteri incorporati, le impostazioni dell'antialiasing, il controllo del canale alfa e altre impostazioni specifiche. La *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0* fornisce descrizioni dettagliate di queste classi e proprietà, tra cui le classi CSMSettings, Font e TextRenderer.

Uso dei caratteri incorporati

Quando si specifica un carattere specifico per un TextField nell'applicazione, Flash Player cerca un carattere dispositivo (cioè un carattere che risiede sul computer dell'utente) con lo stesso nome. Se non lo trova sul sistema dell'utente o se l'utente ha una versione leggermente diversa del carattere con tale nome, la visualizzazione del testo può apparire molto diversa da quella progettata.

Per garantire che l'utente veda esattamente il carattere corretto, è possibile incorporarlo nel file SWF dell'applicazione. I caratteri incorporati offrono una serie di vantaggi:

- Ai caratteri incorporati viene applicata la funzione di antialiasing, che rende i bordi più morbidi, specialmente nel testo di grandi dimensioni.
- Il testo che utilizza i caratteri incorporati può essere ruotato.
- Il testo che usa caratteri incorporati può essere reso trasparente o semitrasparente.
- È possibile utilizzare lo stile CSS `kerning` (crenatura) con i caratteri incorporati.

La principale limitazione derivante dall'uso dei caratteri incorporati consiste nel fatto che aumentano le dimensioni del file o le dimensioni di scaricamento dell'applicazione.

Il metodo per incorporare un file audio nel file SWF dell'applicazione varia a seconda dell'ambiente di sviluppo.

Una volta incorporato un carattere, è possibile fare in modo che un TextField utilizzi il carattere incorporato corretto:

- Impostare la proprietà `embedFonts` dell'oggetto TextField su `true`.
- Creare un oggetto TextFormat, impostarne la proprietà `fontFamily` sul nome del carattere incorporato e applicare l'oggetto TextFormat all'oggetto TextField. Quando si specifica un carattere incorporato, la proprietà `fontFamily` deve contenere un solo nome; non può utilizzare un elenco delimitato da virgola di nomi di caratteri.
- Se si utilizzano gli stili CSS per impostare i caratteri per gli oggetti TextField o i relativi componenti, impostare la proprietà CSS `font-family` sul nome del carattere incorporato. Se si desidera specificare un carattere incorporato, la proprietà `font-family` deve contenere un solo nome e non un elenco di nomi.

Incorporamento di un carattere in Flash

Lo strumento di creazione di Flash consente di incorporare quasi tutti i caratteri installati nel sistema, compresi i caratteri TrueType e Type 1 PostScript.

Esistono diversi modi per incorporare i caratteri in un'applicazione Flash, ad esempio è possibile:

- Impostare le proprietà carattere e stile di un oggetto TextField sullo stage e fare clic sulla casella di controllo Incorpora caratteri
- Creare e fare riferimento a un simbolo di carattere
- Creare e utilizzare una libreria condivisa in runtime contenente simboli di carattere incorporati

Per ulteriori dettagli su come incorporare i caratteri nelle applicazioni Flash, vedere "Caratteri incorporati per i campi di testo dinamico o di input" nella guida *Uso di Flash*.

Controllo della precisione, dello spessore e dell'antialiasing

Per impostazione predefinita, Flash Player determina le impostazioni per i controlli di visualizzazione del testo come la precisione, lo spessore e l'antialiasing quando il testo viene ridimensionato, cambia colore o viene visualizzato su vari sfondi. In alcuni casi, ad esempio in presenza di testo molto piccolo o molto grande o di testo su una gamma di sfondi univoci, è possibile controllare queste impostazioni. È possibile ignorare le impostazioni di Flash Player mediante la classe `flash.text.TextRenderer` e le classi a essa associate, come la classe CSMSettings. Queste classi forniscono un controllo preciso sulla qualità di rendering del testo incorporato. Per ulteriori informazioni sui caratteri incorporati, vedere "Uso dei caratteri incorporati" a pagina 564.

NOTA

La proprietà `flash.text.TextField.antiAliasType` **deve contenere il valore** `AntiAliasType.ADVANCED` **(che è quello predefinito) per consentire di impostare la precisione, lo spessore o la proprietà** `gridFitType` **o per utilizzare il metodo** `TextRenderer.setAdvancedAntiAliasingTable()`.

L'esempio seguente applica delle proprietà personalizzate di modulazione continua del tratto (CSM, Continuous Stroke Modulation) e di formattazione al testo visualizzato mediante un carattere incorporato di nome `myFont`. Quando l'utente fa clic sul testo visualizzato, Flash Player applica le impostazioni personalizzate:

```
var format:TextFormat = new TextFormat();
format.color = 0x336699;
format.size = 48;
format.font = "myFont";

var myText:TextField = new TextField();
myText.embedFonts = true;
myText.autoSize = TextFieldAutoSize.LEFT;
myText.antiAliasType = AntiAliasType.ADVANCED;
myText.defaultTextFormat = format;
myText.selectable = false;
myText.mouseEnabled = true;
myText.text = "Hello World";
addChild(myText);
myText.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:Event):void
{
  var myAntiAliasSettings = new CSMSettings(48, 0.8, -0.8);
  var myAliasTable:Array = new Array(myAntiAliasSettings);
  TextRenderer.setAdvancedAntiAliasingTable("myFont", FontStyle.ITALIC,
  TextColorType.DARK_COLOR, myAliasTable);
}
```

Operazioni con il testo statico

Il testo statico viene creato solo mediante lo strumento di creazione di Flash. Non è possibile creare un'istanza di un testo statico a livello di codice mediante ActionScript. Il testo statico è utile se il testo è molto breve e non è destinato a essere modificato (a differenza del testo dinamico). Il testo statico è una sorta di elemento grafico, ad esempio un cerchio o un quadrato, disegnato sullo stage nello strumento di creazione di Flash. Mentre il testo statico è più limitato di quello dinamico, ActionScript 3.0 supporta la capacità di leggere i valori delle proprietà del testo statico mediante la classe flash.text.StaticText. Inoltre, è possibile utilizzare la classe flash.text.TextSnapshot per leggere i valori del testo statico.

Accesso ai campi di testo statici con la classe StaticText

Di solito, si utilizza la classe flash.text.StaticText nel pannello Azioni dello strumento di creazione di Flash per interagire con un'istanza di testo statico posizionata sullo stage. È anche possibile lavorare in file ActionScript che interagiscono con un file SWF che contiene testo statico. In entrambi i casi, non è possibile creare un'istanza a livello di codice di un testo statico. Il testo statico viene creato solo mediante lo strumento di creazione di Flash CS3.

Per creare un riferimento a un campo di testo statico esistente in ActionScript 3.0, è possibile eseguire un'iterazione su elementi nell'elenco di visualizzazione e assegnare una variabile. Ad esempio:

```
for (var i = 0; i < this.numChildren; i++) {
    var displayitem:DisplayObject = this.getChildAt(i);
    if (displayitem instanceof StaticText) {
        trace("a static text field is item " + i + " on the display list");
     var myFieldLabel:StaticText = StaticText(displayitem);
     trace("and contains the text: " + myFieldLabel.text);
    }
}
```

Una volta creato un riferimento a un campo di testo statico, è possibile utilizzare le proprietà di questo campo in ActionScript 3.0. Il codice che segue è associato a un fotogramma nella linea temporale e assume una variabile chiamata `myFieldLabel` assegnata a un riferimento di testo statico. Nell'esempio, un campo di testo dinamico di nome `myField` viene posizionato in relazione ai valori `x` e `y` di `myFieldLabel` e visualizza di nuovo il valore di `myFieldLabel`.

```
var myField:TextField = new TextField();
addChild(myField);
myField.x = myFieldLabel.x;
myField.y = myFieldLabel.y + 20;
myField.autoSize = TextFieldAutoSize.LEFT;
myField.text = "and " + myFieldLabel.text
```

Uso della classe TextSnapshot

Se si desidera lavorare a livello di codice con un'istanza di testo statico esistente, è possibile utilizzare la classe flash.text.TextSnapshot per interagire con la proprietà `textSnapshot` di un flash.display.DisplayObjectContainer. In altre parole, viene creata un'istanza TextSnapshot dalla proprietà `DisplayObjectContainer.textSnapshot`. A quel punto, è possibile applicare i metodi a tale istanza per recuperare i valori o selezionare delle parti del testo statico.

Ad esempio, posizionare sullo stage un campo di testo contenente il testo "TextSnapshot Example". Aggiungere il codice ActionScript seguente al fotogramma 1 della linea temporale:

```
var mySnap:TextSnapshot = this.getTextSnapshot();
var count:Number = mySnap.getCount();
mySnap.setSelected(0, 4, true);
mySnap.setSelected(1, 2, false);
var myText:String = mySnap.getSelectedText(false);
trace(myText);
```

La classe TextSnapshot è utile per ottenere il testo dai campi di testo statici di un file SWF caricato, nel caso si desideri utilizzare il testo come valore in un'altra parte dell'applicazione.

Esempio: Formattazione del testo in stile quotidiano

L'esempio News Layout formatta il testo in modo che abbia l'aspetto di un articolo di quotidiano. Il testo immesso può contenere un titolo, un sottotitolo e il corpo dell'articolo. Se sono date la larghezza e l'altezza dello schermo, l'esempio News Layout formatta il titolo e il sottotitolo in modo che occupino l'intera larghezza dell'area di visualizzazione. Il testo dell'articolo viene distribuito su due o più colonne.

Questo esempio illustra le seguenti tecniche di programmazione ActionScript:

- Estensione della classe TextField
- Caricamento e applicazione di un file CSS esterno
- Conversione degli stili CSS in oggetti TextFormat
- Uso della classe TextLineMetrics per ottenere informazioni sulle dimensioni di visualizzazione del testo

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione News Layout sono disponibili nella cartella Samples/NewsLayout. L'applicazione è costituita dai seguenti file:

File	Descrizione
NewsLayout.mxml oppure NewsLayout.fla	L'interfaccia utente dell'applicazione per Flex (MXML) o Flash (FLA).
StoryLayout.as	La classe ActionScript principale che organizza tutti i componenti di un articolo per la visualizzazione.

File	Descrizione
FormattedTextField.as	Una sottoclasse della classe TextField che gestisce il proprio oggetto TextFormat.
HeadlineTextField.as	Una sottoclasse della classe FormattedTextField che regola la dimensione dei caratteri per adattarsi a una larghezza desiderata.
MultiColumnTextField.as	Una classe ActionScript che distribuisce il testo su due o più colonne.
story.css	Un file CSS che definisce gli stili di testo per il layout.
newsconfig.xml	Un file XML con il contenuto dell'articolo.

Lettura di un file CSS esterno

L'applicazione News Layout comincia leggendo il testo dell'articolo da un file XML locale. Quindi, legge un file CSS esterno che fornisce le informazioni di formattazione per il titolo, il sottotitolo e il testo principale.

Il file CSS definisce tre stili: uno stile di paragrafo standard per l'articolo e gli stili h1 e h2, rispettivamente per il titolo e il sottotitolo.

```
p {
  font-family: Georgia, Times New Roman, Times, _serif;
  font-size: 12;
  leading: 2;
  text-align: justify;
}

h1 {
  font-family: Verdana, Arial, Helvetica, _sans;
  font-size: 20;
  font-weight: bold;
  color: #000099;
  text-align: left;
}

h2 {
  font-family: Verdana, Arial, Helvetica, _sans;
  font-size: 16;
  font-weight: normal;
  text-align: left;
}
```

La tecnica utilizzata per leggere il file CSS esterno è la stessa descritta in "Caricamento di un file CSS esterno" a pagina 561. Quando il file CSS è stato caricato, l'applicazione esegue il metodo `onCSSFileLoaded()`, mostrato qui sotto.

```
public function onCSSFileLoaded(event:Event):void
{
  this.sheet = new StyleSheet();
  this.sheet.parseCSS(loader.data);

  h1Format = getTextStyle("h1", this.sheet);
  if (h1Format == null)
  {
    h1Format = getDefaultHeadFormat();
  }
  h2Format = getTextStyle("h2", this.sheet);
  if (h2Format == null)
  {
    h2Format = getDefaultHeadFormat();
    h2Format.size = 16;
  }
  displayStory();
}
```

Il metodo `onCSSFileLoaded()` crea un nuovo oggetto StyleSheet e fa in modo che quest'ultimo analizzi i dati CSS di input. Il testo principale dell'articolo viene visualizzato in un oggetto MultiColumnTextField, che può utilizzare un oggetto StyleSheet direttamente. Tuttavia, i campi del titolo utilizzano la classe HeadlineTextField, per la cui formattazione viene usato un oggetto TextFormat.

Il metodo `onCSSFileLoaded()` chiama il metodo `getTextStyle()` due volte per convertire una dichiarazione di stile CSS in un oggetto TextFormat da utilizzare con ognuno dei due oggetti HeadlineTextField. Di seguito viene mostrato il metodo `getTextStyle()`:

```
public function getTextStyle(styleName:String, ss:StyleSheet):TextFormat
{
  var format:TextFormat = null;

  var style:Object = ss.getStyle(styleName);
  if (style != null)
  {
    var colorStr:String = style.color;
    if (colorStr != null && colorStr.indexOf("#") == 0)
    {
      style.color = colorStr.substr(1);
    }
```

```
    format = new TextFormat(style.fontFamily,
                  style.fontSize,
                  style.color,
                  (style.fontWeight == "bold"),
                  (style.fontStyle == "italic"),
                  (style.textDecoration == "underline"),
                  style.url,
                  style.target,
                  style.textAlign,
                  style.marginLeft,
                  style.marginRight,
                  style.textIndent,
                  style.leading);

    if (style.hasOwnProperty("letterSpacing"))
    {
      format.letterSpacing = style.letterSpacing;
    }
  }
  return format;
}
```

I nomi di proprietà e il significato dei valori di proprietà nelle dichiarazioni di stile CSS sono diversi rispetto agli oggetti TextFormat. Il metodo `getTextStyle()` converte i valori delle proprietà CSS nei valori previsti dall'oggetto TextFormat.

Disposizione degli elementi dell'articolo sulla pagina

La classe StoryLayout formatta e dispone i campi del titolo, del sottotitolo e del testo principale con un layout in stile quotidiano. Inizialmente, il metodo `displayText()` crea e posiziona i vari campi.

```
public function displayText():void
{
  headlineTxt = new HeadlineTextField(h1Format);
  headlineTxt.wordWrap = true;
  this.addChild(headlineTxt);
  headlineTxt.width = 600;
  headlineTxt.height = 100;
  headlineTxt.fitText(this.headline, 1, true);

  subtitleTxt = new HeadlineTextField(h2Format);
  subtitleTxt.wordWrap = true;
  subtitleTxt.y = headlineTxt.y + headlineTxt.height;
  this.addChild(subtitleTxt);
  subtitleTxt.width = 600;
  subtitleTxt.height = 100;
  subtitleTxt.fitText(this.subtitle, 1, false);
```

```
  storyTxt = new MultiColumnTextField(2, 10, 600, 200);
  storyTxt.y = subtitleTxt.y + subtitleTxt.height + 4;
  this.addChild(storyTxt);
  storyTxt.styleSheet = this.sheet;
  storyTxt.htmlText = loremIpsum;
}
```

Ogni campo viene semplicemente posizionato al di sotto del campo precedente impostandone la proprietà `y` sul valore della proprietà `y` del campo precedente più l'altezza del campo precedente. Questo calcolo di posizionamento dinamico è necessario poiché gli oggetti HeadlineTextField e gli oggetti MultiColumnTextField possono modificare la propria altezza per adattarsi ai contenuti.

Modifica della dimensione dei caratteri per adattarli alle dimensioni del campo

Se sono dati una larghezza in pixel e un numero massimo di righe da visualizzare, l'oggetto HeadlineTextField modifica la dimensione dei caratteri per adattare il testo al campo. Se il testo è breve, la dimensione del carattere è elevata, creando un titolo in stile tabloid. Se il testo è lungo, la dimensione del carattere è ovviamente più ridotta.

Il metodo `HeadlineTextField.fitText()` mostrato di seguito attiva il funzionamento del ridimensionamento dei caratteri:

```
public function fitText(msg:String, maxLines:uint = 1, toUpper:Boolean =
  false, targetWidth:Number = -1):uint
{
  this.text = toUpper ? msg.toUpperCase() : msg;

  if (targetWidth == -1)
  {
    targetWidth = this.width;
  }

  var pixelsPerChar:Number = targetWidth / msg.length;

  var pointSize:Number = Math.min(MAX_POINT_SIZE, Math.round(pixelsPerChar
  * 1.8 * maxLines));

  if (pointSize < 6)
  {
    // La dimensione in punti è troppo piccola
    return pointSize;
  }
```

```
    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
      return shrinkText(--pointSize, maxLines);
    }
    else
    {
      return growText(pointSize, maxLines);
    }
  }

  public function growText(pointSize:Number, maxLines:uint = 1):Number
  {
    if (pointSize >= MAX_POINT_SIZE)
    {
      return pointSize;
    }

    this.changeSize(pointSize + 1);

    if (this.numLines > maxLines)
    {
      // Riportarla all'ultima dimensione
      this.changeSize(pointSize);
      return pointSize;
    }
    else
    {
      return growText(pointSize + 1, maxLines);
    }
  }

  public function shrinkText(pointSize:Number, maxLines:uint=1):Number
  {
    if (pointSize <= MIN_POINT_SIZE)
    {
      return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
      return shrinkText(pointSize - 1, maxLines);
    }
    else
    {
      return pointSize;
    }
  }
```

Il metodo `HeadlineTextField.fitText()` utilizza una semplice tecnica ricorsiva per ridimensionare il carattere. Innanzi tutto, determina per supposizione un numero medio di pixel per carattere nel testo e da tale valore calcola una dimensione iniziale in punti. Quindi, modifica la dimensione del carattere del campo di testo e verifica se il testo utilizza il ritorno a capo automatico per creare più del numero massimo di righe di testo. Se sono presenti troppe righe, chiama il metodo `shrinkText()` per ridurre la dimensione del carattere e riprovare. Se non sono presenti troppe righe, chiama il metodo `growText()` per aumentare la dimensione del carattere e riprovare. Il processo termina quando l'incremento della dimensione del carattere di un altro punto creerebbe troppe righe di testo.

Distribuzione del testo su più colonne

La classe MultiColumnTextField distribuisce il testo su più oggetti TextField, che vengono successivamente riorganizzati come colonne di quotidiano.

La funzione di costruzione `MultiColumnTextField()` crea innanzi tutto un array di oggetti TextField, uno per ciascuna colonna, come mostrato di seguito:

```
for (var i:int = 0; i < cols; i++)
{
  var field:TextField = new TextField();
  field.autoSize = TextFieldAutoSize.NONE;
  field.wordWrap = true;
  field.styleSheet = this.styleSheet;

  this.fieldArray.push(field);
  this.addChild(field);
}
```

Ogni oggetto TextField viene aggiunto all'array e all'elenco di visualizzazione con il metodo `addChild()`.

Ogni qual volta la proprietà `text` o `styleSheet` dell'oggetto StoryLayout cambia, chiama il metodo `layoutColumns()` per rivisualizzare il testo. Il metodo `layoutColumns()` chiama il metodo `getOptimalHeight()`, mostrato di seguito, per determinare l'altezza in pixel necessaria per adattare tutto il testo all'interno della larghezza di layout data.

```
public function getOptimalHeight(str:String):int
{
  if (fieldArray.length == 0 || str == "" || str == null)
  {
    return this.preferredHeight;
  }
  else
  {
    var colWidth:int = Math.floor( (this.preferredWidth -
      ((this.numColumns - 1) * gutter)) / this.numColumns);
```

```
        var field:TextField = fieldArray[0] as TextField;
        field.width = colWidth;
        field.htmlText = str;

        var linesPerCol:int = Math.ceil(field.numLines / this.numColumns);
        var metrics:TextLineMetrics = field.getLineMetrics(0);
        var prefHeight:int = linesPerCol * metrics.height;
        return prefHeight + 4;
    }
}
```

Innanzi tutto, il metodo `getOptimalHeight()` calcola la larghezza di ogni colonna. Quindi, imposta la larghezza e la proprietà `htmlText` del primo oggetto TextField nell'array. Il metodo `getOptimalHeight()` utilizza tale oggetto TextField per determinare il numero totale di righe a cui è stato applicato l'a capo automatico nel testo, e in base a tale valore identifica il numero di righe che devono essere incluse in ogni colonna. Infine, chiama il metodo `TextField.getLineMetrics()` per recuperare un oggetto TextLineMetrics che contiene i dettagli relativi alle dimensioni del testo nella prima riga. La proprietà `TextLineMetrics.height` rappresenta l'altezza completa di una riga di testo, espressa in pixel, comprese le misure ascent, descent e leading. L'altezza ottimale per l'oggetto MultiColumnTextField è pertanto l'altezza della riga moltiplicata per il numero di righe per colonna, più 4 per il bordo di 2 pixel nella parte superiore e inferiore di un oggetto TextField.

Qui di seguito è riportato il codice per il metodo `layoutColumns()` completo:

```
public function layoutColumns():void
{
  if (this._text == "" || this._text == null)
  {
    return;
  }

  if (this.fitToText)
  {
    this.preferredHeight = this.getOptimalHeight(this._text);
  }

  var colWidth:int = Math.floor( (this.preferredWidth -
    ((numColumns - 1) * gutter)) / numColumns);
  var field:TextField;
  var remainder:String = this._text;
  var fieldText:String = "";
```

```
  for (var i:int = 0; i < fieldArray.length; i++)
  {
    field = this.fieldArray[i] as TextField;
    field.width = colWidth;
    field.height = this.preferredHeight;

    field.x = i * (colWidth + gutter);
    field.y = 0;

    field.htmlText = "<p>" + remainder + "</p>";

    remainder = "";
    fieldText = "";

    var linesRemaining:int = field.numLines;
    var linesVisible:int = field.numLines - field.maxScrollV + 1;
    for (var j:int = 0; j < linesRemaining; j++)
    {
      if (j < linesVisible)
      {
        fieldText += field.getLineText(j);
      }
      else
      {
        remainder +=  field.getLineText(j);
      }
    }

    field.htmlText = "<p>" + fieldText + "</p>";
  }
}
```

Dopo che la proprietà `preferredHeight` è stata impostata chiamando il metodo `getOptimalHeight()`, il metodo `layoutColumns()` esegue le iterazioni sugli oggetti TextField, impostando l'altezza di ciascuno sul valore `preferredHeight`. Il metodo `layoutColumns()` distribuisce quindi a ogni campo il numero di righe di testo sufficiente per non far verificare lo scorrimento in un singolo campo, e il testo in ogni campo successivo ha inizio dove termina il testo nel campo precedente.

CAPITOLO 18

Operazioni con le bitmap

18

Oltre alle funzioni di disegno vettoriale, ActionScript 3.0 include la capacità di creare immagini bitmap o di manipolare i dati pixel delle immagini bitmap esterne che vengono caricate in un file SWF. Grazie alla possibilità di accedere ai singoli valori di pixel e di modificarli, è possibile creare degli effetti simil-filtro personalizzati e utilizzare le funzioni di disturbo incorporate per creare delle texture e del disturbo casuale. Tutte queste tecniche sono descritte in questo capitolo.

Sommario

Nozioni fondamentali sulle operazioni con le bitmap

Introduzione alle operazioni con le bitmap

Quando si lavora con le immagini digitali, è probabile imbattersi in due tipi principali di elementi grafici: le immagini bitmap e le immagini vettoriali. La grafica bitmap, nota anche come grafica raster, è composta da piccoli quadrati (pixel) disposti con una formazione a griglia rettangolare. La grafica vettoriale è composta da forme geometriche (quali linee, curve e poligoni) generate matematicamente.

Le immagini bitmap sono definite dalla larghezza e dall'altezza dell'immagine, misurate in pixel, e dal numero di bit contenuti in ogni pixel, che rappresenta il numero di colori che un pixel è in grado di contenere. Nel caso di un'immagine bitmap che utilizza il modello di colore RGB, i pixel sono costituiti da tre byte: rosso, verde e blu. Ognuno di questi byte contiene un valore compreso tra 0 e 255. Quando i byte vengono combinati all'interno del pixel, producono un colore simile a quello ottenuto da un pittore quando miscela i colori. Ad esempio, un pixel che contiene valori in byte di rosso-255, verde-102 e blu-0 produce un arancione brillante.

La qualità di un'immagine bitmap viene determinata combinando la risoluzione dell'immagine con il relativo valore in bit della profondità di colore. La *risoluzione* fa riferimento al numero di pixel contenuti all'interno di un'immagine. Maggiore è il numero di pixel, più elevata è la risoluzione e più accurata appare l'immagine. La *profondità di colore* fa riferimento alla quantità di informazioni che un pixel è in grado di contenere. Ad esempio, un'immagine che ha un valore di profondità di colore di 16 bit per pixel non può rappresentare lo stesso numero di colori di un'immagine che ha una profondità di colore di 48 bit. Di conseguenza, l'immagine a 48 bit ha dei livelli di ombreggiatura più attenuati rispetto alla sua controparte a 16 bit.

Dal momento che gli elementi grafici bitmap dipendono dalla risoluzione, quando vengono modificati in scala non producono risultati ottimali. Questa situazione è particolarmente evidente quando le immagini bitmap vengono ingrandite in scala. L'ingrandimento in scala di solito produce una perdita di dettaglio e di qualità.

Formati di file bitmap

Le immagini bitmap sono raggruppate in una serie di formati di file comuni. Questi formati utilizzano diversi tipi di algoritmi di compressione per ridurre le dimensioni dei file, oltre che per ottimizzare la qualità dell'immagine in base allo scopo finale dell'immagine. I formati di immagine bitmap supportati da Adobe Flash Player sono GIF, JPG e PNG.

GIF

Il formato Graphics Interchange Format (GIF) è stato originariamente sviluppato da CompuServe nel 1987 come mezzo per trasmettere le immagini a 256 colori (colore a 8 bit). Il formato fornisce dimensioni di file ridotte ed è ideale per le immagini basate sul Web. A causa della tavolozza colori limitata, le immagini GIF di solito non sono adattate per le fotografie, che generalmente richiedono livelli elevati di ombreggiatura e gradienti di colore. Le immagini GIF consentono la trasparenza a livello di singolo bit, che permette di mappare i colori come trasparenti. Ciò comporta che il colore di sfondo di una pagina Web è visibile attraverso l'immagine nei punti in cui la trasparenza è stata mappata.

JPEG

Sviluppato dal Joint Photographic Experts Group (JPEG), il formato di immagine JPEG (spesso indicato come JPG) utilizza un algoritmo di compressione con perdita per riprodurre la profondità di colore a 24 bit con una dimensione ridotta di file. Nella compressione con perdita ogni volta che un'immagine viene salvata perde qualità ma produce un file di dimensioni inferiori. Il formato JPEG è ideale per le fotografie, poiché è in grado di visualizzare milioni di colori. La capacità di controllare il livello di compressione applicato a un'immagine consente di manipolare la qualità dell'immagine e le dimensioni del file.

PNG

Il formato Portable Network Graphics (PNG) è stato creato come alternativa open-source al formato GIF brevettato. I file PNG supportano una profondità di colore fino a 64 bit, consentendo la riproduzione fino a 16 milioni di colori. Dal momento che il PNG è un formato relativamente recente, non è supportato da alcuni dei browser più datati. A differenza dei JPG, i file PNG utilizzano una compressione senza perdita: in altre parole, quando l'immagine viene salvata non va perso alcun dato di immagine. I file PNG supportano anche la trasparenza alfa, che consente fino a 256 livelli di trasparenza.

Bitmap trasparenti e bitmap opache

Nelle immagini bitmap che utilizzano il formato GIF o PNG è possibile aggiungere un byte aggiuntivo (canale alfa) a ogni pixel. Questo byte di pixel aggiuntivo rappresenta il valore della trasparenza del pixel.

Le immagini GIF supportano la trasparenza a livello di singolo bit; in altre parole, è possibile specificare che sia trasparente un solo colore selezionato in una tavolozza di 256 colori. Le immagini PNG, d'altro canto, supportano fino a 256 livelli di trasparenza. Questa funzione è particolarmente utile quando è necessario fondere immagini o testo agli sfondi.

ActionScript 3.0 replica questo byte di pixel di trasparenza aggiuntivo all'interno della classe BitmapData. In modo simile al modello di trasparenza PNG, la costante `BitmapDataChannel.ALPHA` offre fino a 256 livelli di trasparenza.

Operazioni comuni con le bitmap

L'esempio seguente descrive diverse operazioni che è possibile eseguire quando si lavora con le immagini bitmap in ActionScript:

- Visualizzazione delle bitmap sullo schermo
- Recupero e impostazione dei valori di colore dei pixel
- Copia dei dati bitmap
  - Creazione della copia esatta di una bitmap
  - Copia di dati da un canale di colore di una bitmap a un canale di colore di un'altra bitmap
  - Copia in una bitmap di un'istantanea di un oggetto di visualizzazione sullo schermo
- Creazione di disturbo e di texture nelle immagini bitmap
- Scorrimento delle bitmap

Concetti e termini importanti

L'elenco seguente contiene dei termini importanti che vengono citati in questo capitolo:

- Alfa: il livello di trasparenza (o, più precisamente, di opacità) in un colore o in un'immagine. La quantità di alfa viene spesso descritta come valore del *canale alfa*.
- Colore ARGB: una combinazione di colori in cui il colore di ogni pixel è una miscela dei valori di colore rosso, verde e blu e la cui trasparenza viene specificata come valore alfa.
- Canale di colore: generalmente, i colori sono rappresentati come una miscela di alcuni colori di base: solitamente (per la grafica al computer) il rosso, il verde e il blu. Ogni colore di base viene considerato un canale di colore; la quantità di colore in ciascun canale di colore, combinato con gli altri, determina il colore finale.
- Profondità colore: noto anche come *profondità bit*, questo termine fa riferimento alla quantità di memoria del computer dedicata a ciascun pixel, il quale a propria volta determina il numero di colori possibili che è possibile rappresentare nell'immagine.
- Pixel: la più piccola unità di informazioni in un'immagine bitmap, in sostanza un punto di colore.
- Risoluzione: le dimensioni in pixel di un'immagine, che determina il livello di dettaglio contenuto nell'immagine. La risoluzione viene spesso espressa in termini di larghezza e altezza in pixel.
- Colore RGB: una combinazione di colori in cui il colore di ogni pixel viene rappresentato come una miscela dei valori di colore rosso, verde e blu.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché questo capitolo descrive come creare e manipolare contenuto visivo, la prova degli esempi di codice prevede l'esecuzione del codice e la visualizzazione dei risultati nel file SWF creato.

Per provare gli esempi di codice contenuti in questo capitolo:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare il codice nel riquadro dello script.
4. Eseguire il programma selezionando Controllo > Prova filmato.

   I risultati dell'esempio di codice vengono visualizzati nel file SWF creato.

Quasi tutti gli esempi di codice includono codice che crea un'immagine bitmap, pertanto è possibile provare il codice direttamente senza la necessità di fornire contenuto bitmap. In alternativa, se si desidera provare il codice su un'immagine personale, è possibile importare questa immagine in Adobe Flash CS3 Professional oppure caricare l'immagine esterna nel file SWF di prova e utilizzarne i dati bitmap con il codice di esempio. Per istruzioni sul caricamento di immagini esterne, vedere "Caricamento dinamico di contenuto di visualizzazione" a pagina 450.

Le classi Bitmap e BitmapData

Le principali classi ActionScript 3.0 dedicate alle operazioni con le immagini bitmap sono la classe Bitmap, che viene utilizzata per visualizzare le immagini bitmap sullo schermo, e la classe BitmapData, che viene utilizzata per accedere ai dati originari dell'immagine bitmap e manipolarli.

Nozioni fondamentali sulla classe Bitmap

In quanto sottoclasse della classe DisplayObject, la classe Bitmap è la classe ActionScript 3.0 principale per la visualizzazione delle immagini bitmap. Queste immagini possono essere caricate in Flash mediante la classe flash.display.Loader oppure create dinamicamente mediante la funzione di costruzione `Bitmap()`. Quando si carica un'immagine da un'origine esterna, un oggetto Bitmap può utilizzare solo immagini in formato GIF, JPEG o PNG. Una volta creata, l'istanza Bitmap può essere considerata come un wrapper di un oggetto BitmapData di cui è necessario effettuare il rendering sullo stage. Dal momento che un'istanza Bitmap è un oggetto di visualizzazione, per manipolarla è possibile utilizzare tutte le caratteristiche e le funzionalità degli oggetti di visualizzazione. Per ulteriori informazioni sulle operazioni con gli oggetti di visualizzazione, vedere Capitolo 12, "Programmazione degli elementi visivi" a pagina 395.

Aggancio ai pixel e attenuazione

Oltre alle funzionalità comuni a tutti gli oggetti di visualizzazione, la classe Bitmap fornisce alcune caratteristiche aggiuntive specifiche delle immagini bitmap.

In modo simile alla funzione di aggancio ai pixel disponibile nello strumento di creazione di Flash, la proprietà `pixelSnapping` della classe Bitmap determina se un oggetto Bitmap si aggancia o meno al pixel più vicino. Questa proprietà accetta una delle tre costanti definite nella classe PixelSnapping: `ALWAYS`, `AUTO` e `NEVER`.

La sintassi per applicare l'aggancio ai pixel è la seguente:

```
myBitmap.pixelSnapping = PixelSnapping.ALWAYS;
```

Spesso, quando le immagini bitmap vengono modificate in scala, diventano sfocate e distorte. Per ridurre questa distorsione, utilizzare la proprietà `smoothing` della classe BitmapData. Questa proprietà booleana, se impostata su `true`, attenua (mediante antialiasing) i pixel all'interno dell'immagine quando quest'ultima viene modificata in scala. In questo modo, l'aspetto dell'immagine risulta più chiaro e naturale.

Nozioni fondamentali sulla classe BitmapData

La classe BitmapData, che si trova nel pacchetto flash.display, può essere paragonata a una fotografia istantanea dei pixel contenuti in un'immagine bitmap caricata o creata dinamicamente. Questa istantanea viene rappresentata da un array di dati pixel all'interno dell'oggetto. La classe BitmapData contiene anche una serie di metodi incorporati utili per la creazione e la manipolazione dei dati pixel.

Per creare un'istanza di un oggetto BitmapData, utilizzare il codice seguente:

```
var myBitmap:BitmapData = new BitmapData(width:Number, height:Number,
  transparent:Boolean, fillColor:uinit);
```

I parametri `width` e `height` specificano le dimensioni della bitmap; il valore massimo di entrambi è 2880 pixel. Il parametro `transparent` specifica se i dati bitmap includono un canale alfa (`true`) o meno (`false`). Il parametro `fillColor` è un valore di colore a 32 bit che specifica il colore di sfondo, oltre che il valore della trasparenza (se è stato impostato su `true`). Nell'esempio seguente viene creato un oggetto BitmapData con uno sfondo arancione che è trasparente per il 50%.

```
var myBitmap:BitmapData = new BitmapData(150, 150, true, 0x80FF3300);
```

Per effettuare il rendering di un oggetto BitmapData appena creato sullo schermo, assegnarlo o racchiuderlo in un'istanza Bitmap. A tale scopo, è possibile passare l'oggetto BitmapData come parametro della funzione di costruzione dell'oggetto Bitmap oppure assegnarlo alla proprietà `bitmapData` di un'istanza Bitmap esistente. È anche necessario aggiungere l'istanza Bitmap all'elenco di visualizzazione chiamando i metodi `addChild()` o `addChildAt()` del contenitore dell'oggetto di visualizzazione che deve contenere l'istanza. Per ulteriori informazioni sulle operazioni con l'elenco di visualizzazione, vedere "Aggiunta di oggetti di visualizzazione all'elenco di visualizzazione" a pagina 408.

L'esempio seguente crea un oggetto BitmapData con un riempimento rosso e lo visualizza in un'istanza Bitmap:

```
var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false,
  0xFF0000);
var myImage:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myImage);
```

Manipolazione dei pixel

La classe BitmapData contiene una serie di metodi che consentono di manipolare i valori dei dati pixel.

Manipolazione dei singoli pixel

Quando si modifica l'aspetto di un'immagine bitmap a livello di pixel, innanzi tutto è necessario ottenere i valori di colore dei pixel contenuti all'interno dell'area che si desidera manipolare. Per leggere questi valori di pixel, si utilizza il metodo `getPixel()`.

Il metodo `getPixel()` recupera un valore RGB da una serie di coordinate x, y (pixel) che vengono passate come parametro. Se uno dei pixel che si desidera manipolare include delle informazioni sulla trasparenza (canale alfa), è necessario utilizzare il metodo `getPixel32()`. Questo metodo recupera un valore RGB, ma a differenza di `getPixel()`, il valore restituito da `getPixel32()` contiene dei dati aggiuntivi che rappresentano il valore del canale alfa (trasparenza) del pixel selezionato.

In alternativa, se si desidera semplicemente modificare il colore o la trasparenza di un pixel contenuto all'interno di una bitmap, è possibile utilizzare il metodo `setPixel()` o `setPixel32()`. Per impostare il colore di un pixel, è sufficiente passare le coordinate x, y e il valore di colore a uno di questi metodi.

L'esempio seguente utilizza il metodo `setPixel()` per disegnare una croce su uno sfondo BitmapData verde. Quindi, utilizza `getPixel()` per recuperare il valore di colore dal pixel in corrispondenza della coordinata 50, 50 e traccia il valore restituito.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 100, false, 0x009900);

for (var i:uint = 0; i < 100; i++)
{
  var red:uint = 0xFF0000;
  myBitmapData.setPixel(50, i, red);
  myBitmapData.setPixel(i, 50, red);
}

var myBitmapImage:Bitmap = new Bitmap(myBitmapData);
addChild(myBitmapImage);

var pixelValue:uint = myBitmapData.getPixel(50, 50);
trace(pixelValue.toString(16));
```

Se si desidera leggere il valore di un gruppo di pixel, anziché quello di un pixel singolo, utilizzare il metodo `getPixels()`. Questo metodo genera un array di byte da un'area rettangolare di dati pixel che viene passato come parametro. Ognuno degli elementi dell'array di byte (in altre parole, ognuno dei valori di pixel) è un intero senza segno, cioè un valore di pixel a 32 bit non moltiplicato.

Al contrario, per poter modificare (o impostare) il valore di un gruppo di pixel, utilizzare il metodo `setPixels()`. Questo metodo prevede due parametri (`rect` e `inputByteArray`), che vengono combinati per produrre un'area rettangolare (`rect`) di dati pixel (`inputByteArray`).

Quando i dati vengono letti (e scritti) da `inputByteArray`, il metodo `ByteArray.readUnsignedInt()` viene chiamato per ognuno dei pixel nell'array. Se per qualche motivo `inputByteArray` non contiene un rettangolo intero di dati pixel, il metodo interrompe in quel punto l'elaborazione dei dati dell'immagine.

È importante ricordare che, sia per ottenere che per impostare i dati pixel, l'array di byte prevede valori di pixel a 32 bit per l'alfa, il rosso, il verde e il blu (ARGB).

L'esempio seguente utilizza i metodi `getPixels()` e `setPixels()` per copiare un gruppo di pixel da un oggetto BitmapData a un altro.

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.utils.ByteArray;
import flash.geom.Rectangle;

var bitmapDataObject1:BitmapData = new BitmapData(100, 100, false,
  0x006666FF);
var bitmapDataObject2:BitmapData = new BitmapData(100, 100, false,
  0x00FF0000);

var rect:Rectangle = new Rectangle(0, 0, 100, 100);
var bytes:ByteArray = bitmapDataObject1.getPixels(rect);

bytes.position = 0;
bitmapDataObject2.setPixels(rect, bytes);

var bitmapImage1:Bitmap = new Bitmap(bitmapDataObject1);
addChild(bitmapImage1);
var bitmapImage2:Bitmap = new Bitmap(bitmapDataObject2);
addChild(bitmapImage2);
bitmapImage2.x = 110;
```

Rilevamento di collisioni a livello di pixel

Il metodo `BitmapData.hitTest()` effettua il rilevamento delle collisioni a livello di pixel tra i dati bitmap e un altro oggetto o punto.

Il metodo `BitmapData.hitTest()` accetta cinque parametri:

- `firstPoint` (Point): questo parametro fa riferimento alla posizione in pixel dell'angolo superiore sinistro del primo BitmapData su cui viene effettuata la verifica di rilevamento delle zone attive.
- `firstAlphaThreshold` (uint): questo parametro specifica il valore di canale alfa più alto che viene considerato opaco per la verifica di rilevamento delle zone attive.

- `secondObject` (Object): questo parametro rappresenta l'area di impatto. L'oggetto `secondObject` può essere un oggetto Rectangle, Point, Bitmap o BitmapData. Questo oggetto rappresenta l'area attiva su cui viene eseguito il rilevamento delle collisioni.
- `secondBitmapDataPoint` (Point): questo parametro opzionale viene utilizzato per definire la posizione di un pixel nel secondo oggetto BitmapData. Utilizzare questo parametro solo quando il valore di `secondObject` è un oggetto BitmapData. Il valore predefinito è `null`.
- `secondAlphaThreshold` (uint): questo parametro opzionale rappresenta il valore di canale alfa più alto che viene considerato opaco per il secondo oggetto BitmapData. Il valore predefinito è -1. Utilizzare questo parametro solo quando il valore di `secondObject` è un oggetto BitmapData ed entrambi gli oggetti BitmapData sono trasparenti.

Quando si esegue il rilevamento delle collisioni sulle immagini opache, è opportuno tenere in considerazione che ActionScript considera l'immagine come se fosse un rettangolo (o un riquadro di delimitazione) completamento opaco. In alternativa, quando si effettua il rilevamento per zone a livello di pixel su immagini trasparenti, è necessario che entrambe le immagini siano trasparenti. Inoltre, ActionScript utilizza i parametri di soglia dell'alfa per determinare in quale punto l'aspetto dei pixel passa da trasparente a opaco.

L'esempio seguente crea tre immagini bitmap e verifica la collisione dei pixel utilizzando due punti di collisione diversi (uno restituisce false, l'altro true):

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.geom.Point;

var bmd1:BitmapData = new BitmapData(100, 100, false, 0x000000FF);
var bmd2:BitmapData = new BitmapData(20, 20, false, 0x00FF3300);

var bm1:Bitmap = new Bitmap(bmd1);
this.addChild(bm1);

// Create a red square.
var redSquare1:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare1);
redSquare1.x = 0;

// Create a second red square.
var redSquare2:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare2);
redSquare2.x = 150;
redSquare2.y = 150;
```

```
// Define the point at the top-left corner of the bitmap.
var pt1:Point = new Point(0, 0);
// Define the point at the center of redSquare1.
var pt2:Point = new Point(20, 20);
// Define the point at the center of redSquare2.
var pt3:Point = new Point(160, 160);

trace(bmd1.hitTest(pt1, 0xFF, pt2)); // true
trace(bmd1.hitTest(pt1, 0xFF, pt3)); // false
```

Copia dei dati bitmap

Per copiare dati bitmap da un'immagine a un'altra, è possibile utilizzare diversi metodi: `clone()`, `copyPixels()`, `copyChannel()` e `draw()`.

Come suggerisce il nome stesso, il metodo `clone()` consente di clonare, o campionare, i dati bitmap da un oggetto BitmapData a un altro. Quando viene chiamato, il metodo restituisce un nuovo oggetto BitmapData che è un clone esatto dell'istanza originale da cui è stato copiato.

L'esempio seguente clona una copia di un quadrato arancione (principale) e posiziona il clone accanto al quadrato principale originale:

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myParentSquareBitmap:BitmapData = new BitmapData(100, 100, false,
  0x00ff3300);
var myClonedChild:BitmapData = myParentSquareBitmap.clone();

var myParentSquareContainer:Bitmap = new Bitmap(myParentSquareBitmap);
this.addChild(myParentSquareContainer);

var myClonedChildContainer:Bitmap = new Bitmap(myClonedChild);
this.addChild(myClonedChildContainer);
myClonedChildContainer.x = 110;
```

Il metodo `copyPixels()` rappresenta un modo rapido e facile per copiare i pixel da un oggetto BitmapData a un altro. Il metodo scatta un'istantanea rettangolare (definita dal parametro `sourceRect`) dell'immagine di origine e la copia in un'altra area rettangolare (di uguali dimensioni). La posizione del rettangolo incollato viene definita all'interno del parametro `destPoint`.

Il metodo `copyChannel()` campiona un valore di canale di colore predefinito (alfa, rosso, verde o blu) da un oggetto BitmapData di origine e lo copia in un canale di un oggetto BitmapData di destinazione. La chiamata a questo metodo non influisce sugli altri canali nell'oggetto BitmapData di destinazione.

Il metodo `draw()` disegna in una nuova bitmap il contenuto grafico (o ne esegue il rendering) da uno sprite, clip filmato o altro oggetto di visualizzazione di origine. Mediante i parametri `matrix`, `colorTransform`, `blendMode` e un parametro `clipRect` di destinazione è possibile modificare il modo di esecuzione del rendering. Questo metodo utilizza il renderer di vettori di Flash Player per generare i dati.

Quando si chiama `draw()`, si passa l'oggetto di origine (sprite, movie clip o altro oggetto di visualizzazione) come primo parametro, come mostrato di seguito:

```
myBitmap.draw(movieClip);
```

Se l'oggetto di origine ha subito delle trasformazioni (colore, matrice e così via) dopo che è stato originariamente caricato, queste trasformazioni non vengono copiate nel nuovo oggetto. Se si desidera copiare le trasformazioni nella nuova bitmap, è necessario copiare il valore della proprietà `transform` dall'oggetto originale alla proprietà `transform` dell'oggetto Bitmap che utilizza il nuovo oggetto BitmapData.

Creazione di texture mediante le funzioni di disturbo

Per modificare l'aspetto di una bitmap, è possibile applicarle un effetto di disturbo, utilizzando il metodo `noise()` o il metodo `perlinNoise()`. Un effetto di disturbo può essere paragonato alle interferenze statiche che appaiono su uno schermo televisivo non sintonizzato.

Per applicare un effetto di disturbo a una bitmap, utilizzare il metodo `noise()`. Questo metodo applica un valore di colore casuale ai pixel che si trovano all'interno di un'area specifica di un'immagine bitmap.

Il metodo accetta cinque parametri:

- `randomSeed` (int): il numero di scelta casuale che determina il motivo. Nonostante il nome, questo numero di fatto produce lo stesso risultato che si ottiene se viene passato lo stesso numero. Per poter ottenere un risultato veramente casuale, utilizzare il metodo `Math.random()` per passare un numero casuale per questo parametro.
- `low` (uint): questo parametro fa riferimento al valore più basso da generare per ogni pixel (da 0 a 255). Il valore predefinito è 0. Se si imposta un valore più basso si ottiene un motivo di disturbo più scuro, mentre un valore più alto produce un motivo più luminoso.
- `high` (uint): questo parametro fa riferimento al valore più alto da generare per ogni pixel (da 0 a 255). Il valore predefinito è 255. Se si imposta un valore più basso si ottiene un motivo di disturbo più scuro, mentre un valore più alto produce un motivo più luminoso.

- `channelOptions` (uint): questo parametro specifica il canale di colore dell'oggetto bitmap a cui viene applicato il motivo di disturbo. Il numero può essere costituito da una combinazione di qualunque valore dei quattro canali di colore ARGB. Il valore predefinito è 7.
- `grayScale` (Boolean): se impostato su `true`, questo parametro applica il valore `randomSeed` ai pixel della bitmap, eliminando in modo efficace tutti i colori dall'immagine. Il canale alfa non viene modificato da questo parametro. Il valore predefinito è `false`.

L'esempio seguente crea un'immagine bitmap e applica a essa un motivo di disturbo blu:

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmap:BitmapData = new BitmapData(250, 250,false, 0xff000000);
myBitmap.noise(500, 0, 255, BitmapDataChannel.BLUE,false);
var image:Bitmap = new Bitmap(myBitmap);
addChild(image);
```

Se si desidera creare una texture dall'aspetto più organico, utilizzare il metodo `perlinNoise()`. Questo metodo produce texture organiche molto realistiche ideali per riprodurre fumo, nuvole, acqua, fuoco o esplosioni.

Poiché viene generato da un algoritmo, il metodo `perlinNoise()` utilizza meno memoria rispetto alle texture basate su bitmap. Tuttavia, può comunque influire sull'uso del processore, rallentando il contenuto Flash e provocando una maggiore lentezza nel ridisegno della schermata rispetto alla frequenza di fotogrammi, in particolare su computer non recenti. Ciò è dovuto principalmente ai calcoli a virgola mobile necessari per elaborare gli algoritmi del disturbo Perlin.

Il metodo accetta nove parametri (i primi sei sono obbligatori):

- `baseX` (Number): determina il valore x (dimensione) dei motivi creati.
- `baseY` (Number): determina il valore y (dimensione) dei motivi creati.
- `numOctaves` (uint): il numero di ottave o singole funzioni di disturbo da combinare per creare questo disturbo. Un numero più elevato di ottave crea immagini con maggiore dettaglio ma richiede un tempo di elaborazione superiore.
- `randomSeed` (int): il numero di scelta casuale funziona allo stesso modo di quando viene utilizzato nella funzione `noise()`. Per ottenere un risultato veramente casuale, utilizzare il metodo `Math.random()` per passare un numero casuale per questo parametro.
- `stitch` (Boolean): se il valore è `true`, questo metodo tenta di attenuare i bordi di transizione dell'immagine per creare delle texture uniformi da utilizzare come riempimenti bitmap affiancati.

- `fractalNoise` (Boolean): questo parametro fa riferimento ai bordi dei gradienti che vengono generati dal metodo. Se è impostato su `true`, il metodo genera disturbo frattale che attenua i bordi dell'effetto. Se è impostato su `false`, genera turbolenza. Un immagine con turbolenza presenta delle discontinuità visibili nel gradiente che possono fornire una migliore approssimazione per effetti visivi più nitidi come le fiamme o le onde del mare.
- `channelOptions` (uint): il parametro `channelOptions` funziona allo stesso modo di quando viene utilizzato nel metodo `noise()`. Specifica il canale di colore (della bitmap) a cui viene applicato il motivo di disturbo. Il numero può essere costituito da una combinazione di qualunque valore dei quattro canali di colore ARGB. Il valore predefinito è 7.
- `grayScale` (Boolean): il parametro `grayScale` funziona allo stesso modo di quando viene utilizzato nel metodo `noise()`. Se è impostato su `true`, applica il valore `randomSeed` ai pixel della bitmap, eliminando in modo efficace tutti i colori dall'immagine. Il valore predefinito è `false`.
- `offsets` (Array): un array di punti che corrispondono agli offset x e y di ciascuna ottava. Se si modificano questi valori di offset, è possibile effettuare lo scorrimento fluido dell'immagine. Ogni punto nell'array di offset modifica la funzione di disturbo di un'ottava specifica. Il valore predefinito è `null`.

L'esempio seguente crea un oggetto BitmapData da 150 x 150 pixel che chiama il metodo `perlinNoise()` per generare un effetto nuvola verde e blu:

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false,
  0x00FF0000);

var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels,
  false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myBitmap);
```

Scorrimento delle bitmap

Si supponga di aver creato un'applicazione di generazione di mappe stradali in cui ogni volta che l'utente sposta la mappa è necessario aggiornare la visualizzazione (anche se la mappa è stata spostata di pochi pixel).

Un modo per creare questa funzionalità consiste nell'eseguire nuovamente un rendering di una nuova immagine che contiene la visualizzazione aggiornata della mappa ogni volta che l'utente la sposta. In alternativa, è possibile creare una singola immagine di grandi dimensioni e utilizzare il metodo `scroll()`.

Il metodo `scroll()` copia una bitmap presente sullo schermo e la incolla in una nuova posizione offset, specificata dai parametri (`x`, `y`). Se una porzione della bitmap si trova al di fuori dello stage, si ottiene l'effetto dello spostamento della mappa. Se si utilizza anche una funzione timer (o un evento `enterFrame`), è possibile dare l'impressione che l'immagine sia animata o scorrevole.

L'esempio seguente utilizza il precedente esempio del disturbo Perlin e genera un'immagine bitmap più grande (tre quarti della quale è al fuori dello stage). Quindi, viene applicato il metodo `scroll()`, insieme a un listener di eventi `enterFrame` che sposta l'immagine di un pixel in diagonale verso il basso. Questo metodo viene chiamato ogni volta che si accede al fotogramma e, di conseguenza, il rendering delle porzioni fuori schermo dell'immagine viene eseguito quando l'immagine scorre verso il basso.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(1000, 1000, false,
  0x00FF0000);
var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE;
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels,
  false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
myBitmap.x = -750;
myBitmap.y = -750;
addChild(myBitmap);

addEventListener(Event.ENTER_FRAME, scrollBitmap);

function scrollBitmap(event:Event):void
{
  myBitmapDataObject.scroll(1, 1);
}
```

Esempio: Animazione di sprite mediante una bitmap fuori schermo

Molti giochi sviluppato in Flash utilizzano centinaia di immagini animate contemporaneamente sullo schermo. Questo esempio di animazione bitmap disegna diverse centinaia di piccole bitmap o sprite su una bitmap fuori schermo di grandi dimensioni, quindi scrive tale bitmap sullo schermo accelerando notevolmente l'animazione. Per una descrizione di questo esempio e per scaricare il codice di origine, vedere www.adobe.com/go/learn_fl_bitmaps_it.

CAPITOLO 19

Operazioni con i file video

19

Flash Video è una delle tecnologie più diffuse su Internet. Tuttavia, la presentazione tradizionale del video (ovvero all'interno di uno schermo rettangolare dotato di una barra di avanzamento e di pulsanti di controllo) è solo uno dei suoi possibili utilizzi in un'applicazione Flash. Grazie ad ActionScript, è possibile avere un livello di accesso e di controllo molto sofisticato del caricamento, della presentazione e della riproduzione del video.

Sommario

Elementi fondamentali del video

Introduzione alle operazioni con i file video

Una funzione importante di Adobe Flash Player è rappresentata dalla capacità di visualizzare e manipolare le informazioni video con ActionScript esattamente come accade con altri contenuti visivi come le immagini, le animazioni, il testo e così via.

Quando si crea un file Flash Video (FLV) in Adobe Flash CS3 Professional, è possibile scegliere uno skin per il video che includa i controlli più comuni per la riproduzione. Tuttavia, non c'è alcun motivo di limitarsi all'uso delle opzioni disponibili. Grazie ad ActionScript, è possibile controllare in modo molto preciso il caricamento, la visualizzazione e la riproduzione del video: in altre parole è possibile creare skin personalizzati per il lettore video o utilizzare il video in molti modi non convenzionali.

Le operazioni con il video in ActionScript richiedono l'uso di una combinazione di diverse classi:

- Classe Video: il riquadro del contenuto video sullo stage è un'istanza della classe Video. Quest'ultima è un oggetto di visualizzazione, pertanto può essere manipolata con le stesse tecniche applicabili ad altri oggetti di visualizzazione, come il posizionamento, l'applicazione di trasformazioni e filtri, l'uso di metodi di fusione e così via.
- Classe NetStream: quando si carica un file video che si intende controllare da ActionScript, viene utilizzata un'istanza NetStream per rappresentare l'origine del contenuto video, in questo caso un flusso di dati video. L'uso di un'istanza NetStream comporta anche l'impiego di un oggetto NetConnection, che rappresenta il collegamento al file video, ovvero una sorta di tunnel attraverso il quale viene trasmesso il video.
- Classe Camera: quando si lavora con dati video provenienti da una videocamera collegata al computer dell'utente, un'istanza Camera rappresenta l'origine del contenuto video: la videocamera dell'utente e i dati video che rende disponibili.

Quando si carica un video esterno, è possibile caricare il file da un server Web standard per eseguire la riproduzione con scaricamento progressivo oppure utilizzare lo streaming video distribuito da un server specializzato come Macromedia® Flash® Media Server di Adobe.

Operazioni comuni con i file video

Questo capitolo descrive le seguenti attività relative al video:

- Visualizzazione e controllo del video sullo schermo
- Caricamento di file FLV esterni

- Gestione dei metadati e delle informazioni sui cue point in un file video
- Acquisizione e visualizzazione dell'input video dalla videocamera di un utente

Concetti e termini importanti

- Cue point: un marcatore che è possibile posizionare in un punto temporale specifico in un file video, ad esempio per fungere da segnalibro o per fornire dati aggiuntivi associati a tale punto temporale.
- Codifica: il processo che converte i dati video da un formato a un altro formato (ad esempio, la conversione di un'origine video ad alta risoluzione in un formato adatto alla distribuzione su Internet).
- Fotogramma: un segmento singolo di informazioni video; ogni fotogramma è come un'immagine statica che rappresenta un'istantanea di un punto temporale. Riproducendo ad alta velocità i fotogrammi in sequenza, viene creata l'illusione del movimento.
- Fotogramma chiave: un fotogramma video che contiene tutte le informazioni relative al fotogramma stesso. I fotogrammi che seguono un fotogramma chiave contengono solo le informazioni sul modo in cui differiscono rispetto al fotogramma chiave anziché contenere le informazioni integrali relative al fotogramma.
- Metadati: le informazioni su un file video che è possibile incorporare nel file stesso e che possono essere recuperate quando il caricamento del video è terminato.
- Scaricamento progressivo: quando un file video viene distribuito da un server Web standard, i dati video vengono caricati mediante lo scaricamento progressivo, ovvero le informazioni video vengono caricate in sequenza. Questa situazione offre il vantaggio che la riproduzione del video può iniziare prima che sia terminato lo scaricamento dell'intero file; tuttavia, impedisce di saltare a una porzione del video che non è stata ancora caricata.
- Streaming: in alternativa allo scaricamento progressivo, è possibile utilizzare un server video speciale per distribuire il video su Internet mediante una tecnica nota come streaming (definita anche "true streaming"). Con lo streaming, il computer di chi guarda il video non scarica mai il video intero. Per accelerare i tempi di scaricamento, in qualsiasi momento il computer richiede solo una porzione delle informazioni video totali. Poiché un server speciale controlla la distribuzione del contenuto video, è possibile accedere a qualunque porzione del video in qualsiasi momento anziché attendere che venga scaricato.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché questo capitolo descrive le operazioni con i file video in ActionScript, tutti gli esempi di codice riportati prevedono la manipolazione di un oggetto video, che potrebbe essere stato sia creato e posizionato sullo stage nello strumento di creazione di Flash che creato mediante ActionScript. La prova degli esempi prevede la visualizzazione del risultato in Flash Player per vedere gli effetti del codice sul video.

La maggior parte degli esempi manipola un oggetto Video senza creare l'oggetto esplicitamente. Per provare questi esempi di codice:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.
4. Se necessario, aprire il pannello Libreria.
5. Dal menu del pannello Libreria, scegliere Nuovo video.
6. Nella finestra di dialogo Proprietà video, specificare un nome per il nuovo simbolo video e scegliere Video (controllato da ActionScript) nel campo Tipo. Fare clic su OK per creare il nuovo simbolo Video.
7. Trascinare un'istanza del simbolo video dal pannello Libreria allo stage.
8. Con l'istanza video selezionata, nella finestra di ispezione Proprietà, assegnargli un nome istanza. Questo nome deve corrispondere al nome utilizzato per l'istanza Video nell'esempio di codice, ad esempio, se il codice manipola un oggetto Video chiamato `vid`, è necessario denominare `vid` anche l'istanza sullo stage.
9. Eseguire il programma selezionando Controllo > Prova filmato.

   Sullo schermo vengono visualizzati i risultati della manipolazione del video secondo quanto specificato nell'esempio di codice.

Alcuni degli esempi di codice contenuti nel capitolo includono la definizione di una classe oltre al codice di esempio. In tal caso, oltre a eseguire i passaggi precedenti, e prima di provare il file SWF, è necessario creare la classe che viene utilizzata nell'esempio. Per creare una classe definita nell'esempio di codice:

1. Verificare che il file FLA utilizzato per la prova sia stato salvato.
2. Dal menu principale, scegliere File > Nuovo.
3. Nella finestra di dialogo Nuovo documento, nella sezione Tipo, scegliere File ActionScript. Fare clic su OK per creare il nuovo file ActionScript.
4. Copiare il codice di definizione classe dell'esempio nel documento ActionScript.

5. Dal menu principale, scegliere File > Salva. Salvare il file nella stessa directory del documento Flash. Il nome file deve corrispondere al nome della classe presente nell'esempio di codice. Ad esempio, se l'esempio di codice definisce una classe chiamata "VideoTest", salvare il file ActionScript con il nome "VideoTest.as".
6. Tornare al documento Flash.
7. Eseguire il programma selezionando Controllo > Prova filmato.

   I risultati dell'esempio vengono visualizzati sullo schermo.

Per informazioni su altre tecniche per la prova degli esempi di codice, vedere "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Nozioni fondamentali sul formato Flash Video (FLV)

Il formato di file FLV contiene dati audio e video codificati per la distribuzione attraverso Flash Player. Ad esempio, se si dispone di un file video QuickTime o Windows Media, occorre utilizzare un codificatore (come Flash Video Encoder o Sorensen Squeeze) per convertirlo in un file FLV.

I file FLV possono essere creati importando un video nell'ambiente di creazione di Flash ed esportandolo come file FLV. È possibile utilizzare il plug-in FLV Export per esportare i file FLV dalle applicazioni di videomontaggio supportate.

Se si utilizzano file FLV esterni, sono disponibili alcune funzioni che non sono attivate quando si lavora con video importati:

- Nei documenti Flash è possibile utilizzare video clip di lunghezza maggiore senza alcun deterioramento delle prestazioni di riproduzione. I file FLV esterni vengono riprodotti mediante la memoria cache. Di conseguenza, i file di grandi dimensioni sono suddivisi in piccoli file e memorizzati; sono accessibili dinamicamente e richiedono una minore quantità di memoria rispetto ai file video incorporati.
- I file FLV esterni possono essere caratterizzati da una frequenza fotogrammi diversa rispetto a quella del documento Flash in cui sono riprodotti. Ad esempio, è possibile impostare la frequenza fotogrammi del documento Flash su 30 f/s e la frequenza fotogrammi video su 21 f/s. Questa impostazione offre un migliore controllo del video rispetto al video incorporato, per garantire riproduzioni più omogenee. Consente anche di riprodurre i file FLV con frequenze di fotogrammi diverse senza dover modificare il contenuto Flash esistente.

- Con i file FLV esterni non è necessario interrompere la riproduzione del documento Flash durante il caricamento del file video. In alcuni casi i file video importati possono interrompere la riproduzione del documento per eseguire alcune funzioni, come l'accesso a un'unità CD-ROM. I file FLV possono eseguire delle funzioni indipendentemente dal documento Flash, senza quindi interromperne la riproduzione.
- L'acquisizione di contenuto video risulta più semplice se si utilizzano file FLV esterni, perché è possibile utilizzare gestori di eventi per accedere ai metadati relativi al video.

SUGGERIMENTO

Per caricare file FLV da un server Web, potrebbe essere necessario registrare l'estensione e il tipo MIME del file con il server Web. A questo proposito, consultare la documentazione del server Web. Il tipo MIME per i file FLV è `video/x-flv`. Per ulteriori informazioni, vedere "Informazioni sulla configurazione di file FLV per l'hosting su un server" a pagina 624.

Nozioni fondamentali sulla classe Video

La classe Video consente di visualizzare lo streaming video dal vivo in un'applicazione senza incorporarlo nel file SWF. Per acquisire e riprodurre video dal vivo, utilizzare il metodo `Camera.getCamera()`. È possibile utilizzare la classe Video anche per riprodurre file Flash Video (FLV) su HTTP o dal file system locale. La classe Video può essere utilizzata in molti modi diversi:

- Caricare dinamicamente un file FLV mediante le classi NetConnection e NetStream e visualizzare il video in un oggetto Video.
- Acquisire l'input video dalla videocamera di un utente.
- Utilizzare il componente FLVPlayback.

NOTA

Le istanze di un oggetto Video sullo stage sono istanze della classe Video.

Anche se fa parte del pacchetto flash.media, la classe Video eredita dalla classe flash.display.DisplayObject e pertanto tutte le funzionalità relative agli oggetti di visualizzazione (ad esempio, le trasformazioni mediante matrice e i filtri) valgono anche per le istanze Video.

Per ulteriori informazioni, vedere "Manipolazione di oggetti di visualizzazione" a pagina 422, "Operazioni con le funzioni geometriche" a pagina 463 e "Filtraggio degli oggetti di visualizzazione" a pagina 501.

Caricamento di file video

Il caricamento di video mediante le classi NetStream e NetConnection è un processo suddiviso in più fasi.

1. La prima operazione da eseguire consiste nel creare un nuovo oggetto NetConnection. La classe NetConnection consente di riprodurre i file FLV in streaming da un indirizzo HTTP o da un'unità locale, passando il valore `null` al metodo `connect()`, se ci si connette a un file FLV locale che non utilizza un server come Adobe Flash Media Server 2 o Adobe Flex.

   ```
   var nc:NetConnection = new NetConnection();
   nc.connect(null);
   ```

2. La seconda operazione consiste nel creare un oggetto NetStream che riceve un oggetto NetConnection come parametro e nello specificare quale file FLV si desidera caricare. Lo snippet di codice seguente collega un oggetto NetStream all'istanza NetConnection specificata e carica un file FLV di nome video.flv nella stessa directory del file SWF:

   ```
   var ns:NetStream = new NetStream(nc);
   ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
   ns.play("video.flv");
   function asyncErrorHandler(event:AsyncErrorEvent):void
   {
     // Ignora l'errore
   }
   ```

3. La terza operazione consiste nel creare un nuovo oggetto Video e associare l'oggetto NetStream creato in precedenza mediante il metodo `attachNetStream()` della classe Video. Quindi è possibile aggiungere l'oggetto Video all'elenco di visualizzazione utilizzando il metodo `addChild()`, come mostra lo snippet di codice seguente:

   ```
   var vid:Video = new Video();
   vid.attachNetStream(ns);
   addChild(vid);
   ```

Una volta immesso il codice precedente, Flash Player tenta di caricare il file video video.flv nella stessa directory del file SWF.

SUGGERIMENTO

Per caricare file FLV da un server Web, potrebbe essere necessario registrare l'estensione e il tipo MIME del file con il server Web. A questo proposito, consultare la documentazione del server Web. Il tipo MIME per i file FLV è `video/x-flv`. Per ulteriori informazioni, vedere "Informazioni sulla configurazione di file FLV per l'hosting su un server" a pagina 624.

Controllo della riproduzione video

La classe NetStream fornisce quattro metodi principali per controllare la riproduzione video:

`pause()`: sospende la riproduzione di uno streaming video. Se il video è già in pausa, la chiamata a questo metodo non ha alcun effetto.

`resume()`: riprende la riproduzione di uno streaming video in pausa. Se il video è già in fase di riproduzione, la chiamata a questo metodo non ha alcun effetto.

`seek()`: cerca il fotogramma chiave più vicino alla posizione specificata (un offset, espresso in secondi, rispetto all'inizio dello streaming).

`togglePause()`: mette in pausa o riprende la riproduzione di uno streaming.

NOTA

Non è presente alcun metodo `stop()`. Per poter interrompere uno streaming, è necessario mettere in pausa la riproduzione e cercare l'inizio del flusso video.

NOTA

Il metodo `play()` non riprende la riproduzione; viene utilizzato per caricare i file video.

L'esempio seguente mostra come controllare un video mediante diversi pulsanti. Per eseguire l'esempio seguente, creare un nuovo documento e aggiungere quattro istanze di pulsante all'area di lavoro (`pauseBtn`, `playBtn`, `stopBtn` e `togglePauseBtn`):

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
  // Ignora l'errore
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

pauseBtn.addEventListener(MouseEvent.CLICK, pauseHandler);
playBtn.addEventListener(MouseEvent.CLICK, playHandler);
stopBtn.addEventListener(MouseEvent.CLICK, stopHandler);
togglePauseBtn.addEventListener(MouseEvent.CLICK, togglePauseHandler);

function pauseHandler(event:MouseEvent):void
{
  ns.pause();
}
```

```
function playHandler(event:MouseEvent):void
{
  ns.resume();
}
function stopHandler(event:MouseEvent):void
{
  // Mette in pausa lo streaming e riporta l'indicatore di riproduzione
  // all'inizio del flusso.
  ns.pause();
  ns.seek(0);
}
function togglePauseHandler(event:MouseEvent):void
{
  ns.togglePause();
}
```

Se si fa clic sull'istanza del pulsante `pauseBtn` mentre il video è in corso di riproduzione, il file video viene messo in pausa. Se il video è già in pausa, il clic su questo pulsante non ha alcun effetto. Se si fa clic sull'istanza del pulsante `playBtn`, viene ripresa la riproduzione del video se quest'ultima era stata precedentemente messa in pausa; in caso contrario, il pulsante non ha alcun effetto se il video era già in fase di riproduzione.

Rilevamento della fine di un flusso video

Per poter intercettare l'inizio e la fine di un flusso video, è necessario aggiungere un listener di eventi all'istanza NetStream che intercetti l'evento `netStatus`. Il codice seguente illustra come intercettare vari codici durante l'intera riproduzione del video:

```
ns.addEventListener(NetStatusEvent.NET_STATUS, statusHandler);
function statusHandler(event:NetStatusEvent):void
{
  trace(event.info.code)
}
```

Il codice precedente genera il seguente risultato:

```
NetStream.Play.Start
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Flush
NetStream.Play.Stop
NetStream.Buffer.Empty
NetStream.Buffer.Flush
```

I due codici che si desidera intercettare specificamente sono “NetStream.Play.Start” e “NetStream.Play.Stop”, che segnalano l’inizio e la fine della riproduzione del video. Lo snippet di codice seguente utilizza un’istruzione switch per filtrare i due codici e visualizzare un messaggio:

```
function statusHandler(event:NetStatusEvent):void
{
  switch (event.info.code)
  {
    case "NetStream.Play.Start":
      trace("Start [" + ns.time.toFixed(3) + " seconds]");
      break;
    case "NetStream.Play.Stop":
      trace("Stop [" + ns.time.toFixed(3) + " seconds]");
      break;
  }
}
```

Intercettando l’evento `netStatus` (`NetStatusEvent.NET_STATUS`), è possibile creare un lettore video che, al termine della riproduzione del video corrente, carica il video successivo all’interno di una playlist.

Streaming di file video

Per effettuare lo streaming di file da Flash Media Server, è possibile utilizzare le classi NetConnection e NetStream per collegarsi a un’istanza di un server remoto e riprodurre un flusso specifico. Per specificare un server RTMP (Real-Time Messaging Protocol), passare l’URL RTMP desiderato (ad esempio, “rtmp://localhost/appName/appInstance”) al metodo `NetConnection.connect()` anziché passare il valore `null`. Per riprodurre un determinato streaming dal vivo o registrato dal Flash Media Server specificato, passare un nome identificativo per i dati dal vivo pubblicati da `NetStream.publish()` o il nome di un file registrato da riprodurre al metodo `NetStream.play()`. Per ulteriori informazioni, consultare la documentazione di Flash Media Server.

Nozioni fondamentali sui cue point

Non tutti i file FLV contengono cue point. I cue point vengono generalmente incorporati in un file FLV durante la codifica FLV, ma esistono anche degli strumenti per incorporarli nei file FLV già esistenti.

Con Flash Video è possibile utilizzare diversi tipi di cue point. È possibile utilizzare ActionScript per interagire con i cue point incorporati in un file FLV (quando si crea il file FLV) o generati attraverso ActionScript.

- Cue point di navigazione: quando si codifica il file FLV i cue point di navigazione vengono incorporati nel flusso FLV e nel pacchetto di metadati FLV. Si utilizzano i cue point di navigazione per consentire agli utenti di eseguire una ricerca in una parte specifica di un file.
- Cue point di evento: quando si codifica il file FLV i cue point di evento vengono incorporati nel flusso FLV e nel pacchetto di metadati FLV. È possibile scrivere il codice per gestire gli eventi che vengono attivati in punti specifici durante la riproduzione FLV.
- Cue point di ActionScript: cue point esterni creati utilizzando codice ActionScript. È possibile scrivere il codice che attiva questi cue point in relazione alla riproduzione del video. Questi cue point sono meno accurati rispetto a quelli incorporati (fino a un decimo di secondo), perché il lettore video li traccia separatamente.

I cue point di navigazione creano un fotogramma chiave nella posizione del cue point specificata, quindi è possibile utilizzare il codice per spostare l'indicatore di riproduzione di un lettore video in quella posizione. In un file FLV è possibile impostare punti particolari in cui gli utenti possono eseguire una ricerca. Per esempio, il video può essere composto da più capitoli o segmenti e lo si può controllare incorporando i cue point di navigazione nel file.

Se si prevede di creare un'applicazione in cui si desidera che gli utenti si spostino su un cue point, occorre realizzare e incorporare i cue point quando si codifica il file invece di utilizzare i cue point ActionScript. Occorre incorporare i cue point nel file FLV, perché si può lavorare in modo più preciso. Per ulteriori informazioni sulla codifica dei file FLV con i cue point, vedere "Incorporamento dei cue point" nella guida *Uso di Flash*.

È possibile accedere ai parametri dei cue point scrivendo codice ActionScript. I parametri dei cue point fanno parte dell'oggetto evento ricevuto dal gestore di callback `onCuePoint`.

Per attivare determinate azioni nel codice quando il video raggiunge un cue point specifico, utilizzare il gestore di eventi `NetStream.onCuePoint`. Per ulteriori informazioni, vedere "Scrittura di metodi di callback per onCuePoint e onMetaData" a pagina 604.

Scrittura di metodi di callback per onCuePoint e onMetaData

È possibile attivare delle azioni nell'applicazione quando vengono raggiunti degli specifici cue point o quando determinati metadati vengono ricevuti dal lettore. Per attivare queste azioni, utilizzare i gestori di eventi `onCuePoint` e `onMetaData`. Per evitare che Flash Player generi degli errori, per questi gestori è necessario scrivere dei metodi di callback. Ad esempio, il codice seguente riproduce un file FLV di nome video.flv nella stessa cartella del documento SWF:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
  trace(event.text);
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Il codice precedente carica un file FLV locale di nome video.flv e intercetta l'evento `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`) da inviare. L'evento viene inviato quando viene generata un'eccezione da un codice asincrono nativo. In questo caso, viene inviato quando un file FLV contiene dei metadati o delle informazioni relative ai cue point e non sono stati definiti i listener appropriati. Il codice precedente gestisce l'evento `asyncError` e ignora l'errore se non si è interessati ai metadati o alle informazioni sui cue point presenti nel file video. Se fosse presente un file FLV contenente dei metadati e diversi cue point, verrebbero tracciate le informazioni seguenti:

```
Errore #2095: flash.net.NetStream non è riuscito a richiamare il callback
  onMetaData.
Errore #2095: flash.net.NetStream non è riuscito a richiamare il callback
  onCuePoint.
Errore #2095: flash.net.NetStream non è riuscito a richiamare il callback
  onCuePoint.
Errore #2095: flash.net.NetStream non è riuscito a richiamare il callback
  onCuePoint.
```

L'errore si verifica perché l'oggetto NetStream non è stato in grado di trovare un metodo di callback `onMetaData` o `onCuePoint`. Esistono vari modi per definire questi metodi di callback all'interno delle applicazioni:

- Impostare la proprietà client dell'oggetto su un oggetto
- Creare una classe personalizzata e definire i metodi per gestire i metodi di callback
- Estendere la classe NetStream e aggiungere i metodi per gestire i metodi di callback
- Estendere la classe NetStream e renderla dinamica
- Impostare la proprietà client dell'oggetto NetStream su questo valore

Impostare la proprietà client dell'oggetto su un oggetto

Impostando la proprietà `client` su un oggetto o su una sottoclasse di NetStream, è possibile reindirizzare i metodi di callback `onMetaData` e `onCuePoint` oppure ignorarli completamente. L'esempio seguente dimostra come utilizzare un oggetto vuoto per ignorare i metodi di callback senza intercettare l'evento `asyncError`:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var customClient:Object = new Object();

var ns:NetStream = new NetStream(nc);
ns.client = customClient;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Se si desidera intercettare uno solo tra i metodi di callback `onMetaData` o `onCuePoint`, è necessario definire i metodi per gestire tali metodi di callback, come mostrato nello snippet di codice seguente:

```
var customClient:Object = new Object();
customClient.onMetaData = metaDataHandler;
function metaDataHandler(infoObject:Object):void
{
  trace("metadata");
}
```

Il codice precedente intercetta il metodo di callback `onMetaData` e chiama il metodo `metaDataHandler()`, che traccia una stringa. Se Flash Player ha incontrato un cue point, non viene generato alcun errore anche se non è definito alcun metodo di callback `onCuePoint`.

Creare una classe personalizzata e definire i metodi per gestire i metodi di callback

Il codice seguente imposta la proprietà `client` dell'oggetto NetStream sulla classe personalizzata CustomClient, che definisce i gestori dei metodi di callback:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = new CustomClient();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

La classe CustomClient ha l'aspetto seguente:

```
package
{
  public class CustomClient
  {
    public function onMetaData(infoObject:Object):void
    {
      trace("metadata");
    }
  }
}
```

La classe CustomClient definisce un gestore per il gestore di callback `onMetaData`. Se si incontra un cue point e viene chiamato il metodo di callback `onCuePoint`, viene inviato un evento `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`) che visualizza il messaggio "flash.net.NetStream non è riuscito a richiamare il callback onCuePoint." Per impedire questo errore, è necessario definire un metodo di callback `onCuePoint` nella classe CustomClient oppure definire un gestore di eventi per l'evento `asyncError`.

Estendere la classe NetStream e aggiungere i metodi per gestire i metodi di callback

Il codice seguente crea un'istanza della classe CustomNetStream, che viene definita in un esempio di codice successivo:

```
var ns:CustomNetStream = new CustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

L'esempio di codice seguente definisce la classe CustomNetStream che estende la classe NetStream, gestisce la creazione dell'oggetto NetConnection necessario e gestisce i metodi del gestore di callback `onMetaData` e `onCuePoint`:

```
package
{
  import flash.net.NetConnection;
  import flash.net.NetStream;
  public class CustomNetStream extends NetStream
  {
    private var nc:NetConnection;
    public function CustomNetStream()
    {
      nc = new NetConnection();
      nc.connect(null);
      super(nc);
    }
    public function onMetaData(infoObject:Object):void
    {
      trace("metadata");
    }
    public function onCuePoint(infoObject:Object):void
    {
      trace("cue point");
    }
  }
}
```

Se si desidera rinominare i metodi `onMetaData()` e `onCuePoint()` nella classe CustomNetStream, è possibile utilizzare il codice seguente:

```
package
{
  import flash.net.NetConnection;
  import flash.net.NetStream;
  public class CustomNetStream extends NetStream
  {
    private var nc:NetConnection;
    public var onMetaData:Function;
    public var onCuePoint:Function;
    public function CustomNetStream()
    {
      onMetaData = metaDataHandler;
      onCuePoint = cuePointHandler;
      nc = new NetConnection();
      nc.connect(null);
      super(nc);
    }
    private function metaDataHandler(infoObject:Object):void
    {
      trace("metadata");
    }
    private function cuePointHandler(infoObject:Object):void
    {
      trace("cue point");
    }
  }
}
```

Estendere la classe NetStream e renderla dinamica

È possibile estendere la classe NetStream e rendere la sottoclasse dinamica così da poter aggiungere dinamicamente i gestori di callback `onCuePoint` e `onMetaData`. Questo comportamento viene illustrato nell'esempio seguente:

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

La classe DynamicCustomNetStream ha l'aspetto seguente:

```
package
{
  import flash.net.NetConnection;
  import flash.net.NetStream;
  public dynamic class DynamicCustomNetStream extends NetStream
  {
    private var nc:NetConnection;
    public function DynamicCustomNetStream()
    {
      nc = new NetConnection();
      nc.connect(null);
      super(nc);
    }
  }
}
```

Anche in assenza di gestori per i gestori di callback `onMetaData` e `onCuePoint` non vengono generati errori, poiché la classe DynamicCustomNetStream è dinamica. Se si desidera definire i metodi per i gestori di callback `onMetaData` e `onCuePoint`, è possibile utilizzare il codice seguente:

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.onMetaData = metaDataHandler;
ns.onCuePoint = cuePointHandler;
ns.play("http://www.helpexamples.com/flash/video/cuepoints.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function metaDataHandler(infoObject:Object):void
{
  trace("metadata");
}
function cuePointHandler(infoObject:Object):void
{
  trace("cue point");
}
```

Impostare la proprietà client dell'oggetto NetStream su questo valore

Impostando la proprietà `client` su `this`, Flash Player cerca nell'area di validità corrente i metodi `onMetaData()` e `onCuePoint()`. Il codice seguente contiene un esempio in tal senso:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Se viene chiamato il gestore di callback `onMetaData` o `onCuePoint` e non esiste alcun metodo per gestire il callback, non viene generato alcun errore. Per gestire questi gestori di callback, creare un metodo `onMetaData()` e `onCuePoint()` nel codice, come mostrato nello snippet seguente:

```
function onMetaData(infoObject:Object):void
{
  trace("metadata");
}
function onCuePoint(infoObject:Object):void
{
  trace("cue point");
}
```

Uso dei cue point

L'esempio seguente utilizza un ciclo `for..in` semplice per eseguire un'iterazione su ciascuna delle proprietà del parametro `infoObject` del gestore di callback `onCuePoint` e tracciare un messaggio quando vengono ricevuti i dati relativi ai cue point:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

```
function onCuePoint(infoObject:Object):void
{
  var key:String;
  for (key in infoObject)
  {
    trace(key + ": " + infoObject[key]);
  }
}
```

Appare l'output seguente:

```
parameters:
name: point1
time: 0.418
type: navigation
```

Questo codice utilizza una delle numerose tecniche disponibili per impostare l'oggetto su cui è stato richiamato il metodo di callback. Per informazioni sulle altre tecniche che è possibile utilizzare, vedere Scrittura di metodi di callback per onCuePoint e onMetaData.

Uso di metadati video

Per visualizzare le informazioni sui metadati nel file FLV è possibile utilizzare il gestore di callback `onMetaData`. I metadati comprendono informazioni sul file FLV come durata, larghezza, altezza e frequenza dei fotogrammi. Le informazioni sui metadati aggiunte al file FLV dipendono dal software utilizzato per codificare il file FLV o da quello impiegato per inserire le informazioni sui metadati.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onMetaData(infoObject:Object):void
{
  var key:String;
  for (key in infoObject)
  {
    trace(key + ": " + infoObject[key]);
  }
}
```

L'esempio precedente genera un codice simile a quello seguente, presupponendo che il file FLV contenga dei cue point e dell'audio:

```
width: 320
audiodelay: 0.038
canSeekToEnd: true
height: 213
cuePoints: ,,
audiodatarate: 96
duration: 16.334
videodatarate: 400
framerate: 15
videocodecid: 4
audiocodecid: 2
```

SUGGERIMENTO

Se il video non dispone dell'audio, le informazioni sui metadati relativi all'audio (come `audiodatarate`) restituiscono `undefined` perché durante la codifica non è stata aggiunta ai metadati alcuna informazione sull'audio.

Nel codice precedente, le informazioni sui cue point non erano visualizzate. Per poter visualizzare i metadati dei cue point, è possibile utilizzare la funzione seguente che visualizza in modo ricorsivo gli elementi in un oggetto:

```
function traceObject(obj:Object, indent:uint = 0):void
{
  var indentString:String = "";
  var i:uint;
  var prop:String;
  var val:*;
  for (i = 0; i < indent; i++)
  {
    indentString += "\t";
  }
  for (prop in obj)
  {
    val = obj[prop];
    if (typeof(val) == "object")
    {
      trace(indentString + " " + j + ": [Object]");
      traceObject(val, indent + 1);
    }
    else
    {
      trace(indentString + " " + prop + ": " + val);
    }
  }
}
```

Se si utilizza lo snippet di codice precedente per tracciare il parametro `infoObject` nel metodo `onMetaData()`, viene creato l'output seguente:

```
width: 320
audiodatarate: 96
audiocodecid: 2
videocodecid: 4
videodatarate: 400
canSeekToEnd: true
duration: 16.334
audiodelay: 0.038
height: 213
framerate: 15
cuePoints: [Object]
  0: [Object]
    parameters: [Object]
      lights: beginning
    name: point1
    time: 0.418
    type: navigation
  1: [Object]
    parameters: [Object]
      lights: middle
    name: point2
    time: 7.748
    type: navigation
  2: [Object]
    parameters: [Object]
      lights: end
    name: point3
    time: 16.02
    type: navigation
```

Oggetti info per onMetaData

Nella tabella seguente sono riportati i valori possibili per i metadati video:

Parametro	Descrizione
audiocodecid	Un numero che indica il codec audio (tecnica code/decode) utilizzato.
audiodatarate	Un numero che indica la frequenza con cui è stato codificato l'audio, espressa in kilobyte al secondo.
audiodelay	Un numero che indica il tempo del file FLV "time 0" del file FLV originale. Il contenuto video deve essere leggermente ritardato per sincronizzare correttamente l'audio.

Parametro	Descrizione
canSeekToEnd	Un valore booleano che è `true` se il file FLV viene codificato con un fotogramma chiave sull'ultimo fotogramma che permette di cercare la fine di un clip filmato a scaricamento progressivo. È `false` se il file FLV non viene codificato con un fotogramma chiave sull'ultimo fotogramma.
cuePoints	Un array di oggetti, uno per ciascun cue point incorporato nel file FLV. Il valore è undefined se il file FLV non contiene cue point. Ciascun oggetto ha le seguenti proprietà: ■ `type`: una stringa che specifica se il cue point è di tipo "navigation" o "event". ■ `name`: una stringa che indica il nome del cue point. ■ `time`: un numero che specifica il tempo del cue point in secondi e considera solo i primi tre decimali (millisecondi). ■ `parameters`: un oggetto opzionale in cui le coppie nome-valore vengono definite dall'utente durante la creazione dei cue point.
duration	Un numero che specifica la durata del file FLV, espressa in secondi.
framerate	Un numero che specifica la frequenza dei fotogrammi del file FLV.
height	Un numero che specifica l'altezza del file FLV, espressa in pixel.
videocodecid	Un numero che indica la versione codec utilizzata per codificare il video.
videodatarate	Un numero che specifica la velocità dati video del file FLV.
width	Un numero che specifica la larghezza del file FLV, espressa in pixel.

Nella tabella seguente sono riportati i valori possibili per il parametro `videocodecid`:

videocodecid	Nome del codec
2	Sorenson H.263
3	Screen video (solo SWF 7 e versioni successive)

videocodecid	Nome del codec
4	VP6 (solo SWF 8 e versioni successive)
5	Video VP6 con canale alfa (solo SWF 8 e versioni successive)

Nella tabella seguente sono riportati i valori possibili per il parametro `audiocodecid`:

audiocodecid	Nome del codec
0	uncompressed
1	ADPCM
2	mp3
5	Nellymoser 8kHz mono
6	Nellymoser

Rilevamento dell'input da videocamera

In aggiunta ai file video esterni, è possibile utilizzare una videocamera collegata al computer dell'utente come origine di dati video visualizzabili e manipolabili mediante ActionScript. La classe Camera è il meccanismo incorporato in ActionScript per l'utilizzo di una videocamera.

Nozioni fondamentali sulla classe Camera

L'oggetto Camera consente di collegarsi alla videocamera locale dell'utente e trasmettere il video a livello locale (cioè, all'utente stesso) oppure a livello remoto a un server (ad esempio, Flash Media Server).

Grazie alla classe Camera, è possibile accedere ai seguenti tipi di informazioni relative alla videocamera dell'utente:

- Quali videocamere installate sul computer dell'utente sono disponibili per Flash Player
- Se una videocamera è installata
- Se a Flash Player è consentito o negato l'accesso alla videocamera dell'utente
- Quale videocamera è attualmente attiva
- La larghezza e l'altezza del video in corso di acquisizione

La classe Camera fornisce diversi metodi e proprietà per eseguire operazioni con gli oggetti Camera. Ad esempio, la proprietà `Camera.names` statica contiene un array dei nomi di videocamera attualmente installati sul computer dell'utente. È anche possibile utilizzare la proprietà `name` per visualizzare il nome della videocamera attualmente attiva.

Visualizzazione sullo schermo del contenuto della videocamera

Il collegamento di una videocamera può richiedere una quantità di codice minore rispetto a quando si utilizzano le classi NetConnection e NetStream per caricare un file FLV. La classe Camera può tuttavia diventare rapidamente complicata, dal momento che per accedere a una videocamera è necessario prima collegarla a Flash Player e che per tale operazione è richiesta l'autorizzazione da parte dell'utente.

Il codice seguente dimostra come utilizzare la classe Camera per collegarsi alla videocamera locale di un utente:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

NOTA La classe Camera non contiene un metodo di costruzione. Per creare una nuova istanza Camera si utilizza il metodo statico `Camera.getCamera()`.

Progettazione di un'applicazione per la videocamera

Quando si crea un'applicazione che si collega alla videocamera di un utente, è necessario eseguire le seguenti operazioni relative al codice:

- Verificare se per l'utente è attualmente installata una videocamera.
- Verificare se l'utente ha autorizzato esplicitamente Flash Player ad accedere alla videocamera. Per motivi di sicurezza, il lettore visualizza la finestra di dialogo Impostazioni di Flash Player che consente all'utente di autorizzare o negare l'accesso alla propria videocamera. In questo modo si impedisce a Flash Player di collegarsi alla videocamera di un utente e di trasmettere uno streaming video senza il suo permesso. Se un utente consente l'accesso, l'applicazione può collegarsi alla videocamera dell'utente. Se un utente non consente l'accesso, l'applicazione non può collegarsi alla videocamera dell'utente. Le applicazioni devono sempre gestire entrambi i casi.

Collegamento alla videocamera di un utente

La prima operazione da eseguire quando ci si collega alla videocamera di un utente consiste nel creare una nuova istanza di una variabile di tipo Camera in base al valore restituito dal metodo statico `Camera.getCamera()`.

Quindi, è necessario creare un nuovo oggetto video e associare a esso l'oggetto Camera.

La terza operazione consiste nell'aggiungere l'oggetto video all'elenco di visualizzazione. Le operazioni 2 e 3 sono necessarie, dal momento che la classe Camera non estende la classe DisplayObject e pertanto non può essere aggiunta direttamente all'elenco di visualizzazione. Per visualizzare il video acquisito dalla videocamera, creare un nuovo oggetto video e chiamare il metodo `attachCamera()`.

Le tre operazioni sono illustrate nel codice seguente:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Si noti che se per un utente non è installata una videocamera, Flash Player non visualizza nulla.

Nella realtà, è necessario eseguire delle operazioni supplementari per l'applicazione. Per ulteriori informazioni, vedere Verifica dell'installazione delle videocamere e Rilevamento delle autorizzazioni per l'accesso alla videocamera.

Verifica dell'installazione delle videocamere

Prima di tentare di utilizzare qualunque metodo o proprietà su un'istanza Camera, è necessario verificare che per l'utente sia installata una videocamera. A tale scopo, è possibile utilizzare uno dei due metodi seguenti:

- Verificare la proprietà statica `Camera.names`, che contiene un array dei nomi di videocamera disponibili. Di solito, questo array ha una o meno stringhe, dal momento che di solito per la maggior parte degli utenti non è installata più di una sola videocamera per volta. Il codice seguente illustra come è possibile verificare la proprietà `Camera.names` per determinare se per l'utente sono disponibili delle videocamere:

```
if (Camera.names.length > 0)
{
  trace("User has no cameras installed.");
}
else
{
  var cam:Camera = Camera.getCamera(); // Get default camera.
}
```

- Verifica il valore restituito dal metodo statico `Camera.getCamera()`. Se non è disponibile o installata alcuna videocamera, questo metodo restituisce `null`, in caso contrario restituisce un riferimento a un oggetto Camera. Il codice seguente illustra come è possibile verificare il metodo `Camera.getCamera()` per determinare se per l'utente sono disponibili delle videocamere:

```
var cam:Camera = Camera.getCamera();
if (cam == null)
{
  trace("User has no cameras installed.");
}
else
{
  trace("User has at least 1 camera installed.");
}
```

Dal momento che la classe Camera non estende la classe DisplayObject, non può essere aggiunta direttamente all'elenco di visualizzazione mediante il metodo `addChild()`. Per visualizzare il video acquisito dalla videocamera, creare un nuovo oggetto Video e chiamare il metodo `attachCamera()` sull'istanza Video.

Questo snippet di codice mostra come associare la videocamera (se presente); in caso contrario, Flash Player non visualizza nulla:

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
  var vid:Video = new Video();
  vid.attachCamera(cam);
  addChild(vid);
}
```

Rilevamento delle autorizzazioni per l'accesso alla videocamera

Per poter visualizzare l'output della videocamera, l'utente deve prima consentire esplicitamente a Flash Player di accedere alla videocamera. Quando il metodo `attachCamera()` viene chiamato, Flash Player visualizza la finestra di dialogo Impostazioni di Flash Player, che richiede all'utente di consentire o negare a Flash Player di accedere alla videocamera e al microfono. Se l'utente ha scelto di consentire l'accesso, l'output della videocamera viene visualizzato nell'istanza Video sullo stage. Se l'utente non ha consentito l'accesso, Flash Player non è in grado di collegarsi alla videocamera e l'oggetto Video non visualizza nulla.

Se per l'utente non è installata una videocamera, Flash Player non visualizza nulla. Se per l'utente è installa una videocamera, Flash Player visualizza la finestra di dialogo Impostazioni di Flash Player, che richiede all'utente di consentire o negare a Flash Player di accedere alla videocamera. Se l'utente consente l'accesso alla propria videocamera, il video viene visualizzato per l'utente stesso; in caso contrario, non viene visualizzato nulla.

Se si desidera rilevare se l'utente ha consentito o negato l'accesso alla videocamera, è possibile intercettare l'evento `status` della videocamera (`StatusEvent.STATUS`), come mostrato nel codice seguente:

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
  cam.addEventListener(StatusEvent.STATUS, statusHandler);
  var vid:Video = new Video();
  vid.attachCamera(cam);
  addChild(vid);
}
function statusHandler(event:StatusEvent):void
{
  // Questo evento viene inviato quando l'utente fa clic sul pulsante
  // "Consenti" o "Nega" nella finestra di dialogo Impostazioni
  // di Flash Player.
  trace(event.code); // "Camera.Muted" o "Camera.Unmuted".
}
```

La funzione `statusHandler()` viene chiamata non appena l'utente fa clic su Consenti o Nega. Mediante uno dei due metodi seguenti è possibile rilevare su quale pulsante l'utente ha fatto clic:

- Il parametro `event` della funzione `statusHandler()` contiene una proprietà code che contiene la stringa "Camera.Muted" o "Camera.Unmuted". Se il valore è "Camera.Muted", l'utente ha fatto clic sul pulsante Nega e Flash Player non può accedere alla videocamera. Lo snippet di codice seguente contiene un esempio:

  ```
  function statusHandler(event:StatusEvent):void
  {
    switch (event.code)
    {
      case "Camera.Muted":
        trace("User clicked Deny.");
        break;
      case "Camera.Unmuted":
        trace("User clicked Accept.");
        break;
    }
  }
  ```

- La classe Camera contiene una proprietà di sola lettura di nome `muted` che specifica se l'utente ha negato (`true`) o consentito (`false`) l'accesso alla videocamera nel pannello Privacy di Flash Player. Lo snippet di codice seguente contiene un esempio:

```
function statusHandler(event:StatusEvent):void
{
  if (cam.muted)
  {
    trace("User clicked Deny.");
  }
  else
  {
    trace("User clicked Accept.");
  }
}
```

Verificando l'avvenuto invio dell'evento status, è possibile scrivere del codice che gestisce l'accettazione o il rifiuto da parte dell'utente dell'accesso alla videocamera ed eseguire la pulizia appropriata del codice. Ad esempio, se l'utente fa clic sul pulsante Nega, è possibile visualizzare un messaggio che avvisa che è necessario fare clic su Consenti per partecipare a una chat video oppure è possibile fare in modo che l'oggetto Video nell'elenco di visualizzazione venga eliminato per liberare risorse di sistema.

Ottimizzazione della qualità video

Per impostazione predefinita, le nuove istanze della classe Video sono larghe 320 pixel e alte 240 pixel. Per poter ottimizzare la qualità video, è necessario garantire sempre che l'oggetto video abbia le stesse dimensioni del video restituito dall'oggetto Camera. È possibile ottenere la larghezza e l'altezza dell'oggetto Camera utilizzando le proprietà `width` e `height` della classe Camera; quindi, è possibile impostare le proprietà `width` e `height` dell'oggetto Video in modo che corrispondano alle dimensioni dell'oggetto Camera oppure è possibile passare la larghezza e l'altezza della videocamera al metodo di costruzione della classe Video, come mostrato nello snippet di codice seguente:

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
  var vid:Video = new Video(cam.width, cam.height);
  vid.attachCamera(cam);
  addChild(vid);
}
```

Dal momento che il metodo `getCamera()` restituisce un riferimento a un oggetto Camera (oppure `null` se non è disponibile alcuna videocamera), è possibile accedere ai metodi e alle proprietà della videocamera anche se l'utente nega l'accesso alla propria videocamera. In questo modo è possibile impostare le dimensioni dell'istanza video mediante l'altezza e la larghezza native della videocamera.

```
var vid:Video;
var cam:Camera = Camera.getCamera();

if (cam == null)
{
  trace("Unable to locate available cameras.");
}
else
{
  trace("Found camera: " + cam.name);
  cam.addEventListener(StatusEvent.STATUS, statusHandler);
  vid = new Video();
  vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
  if (cam.muted)
  {
    trace("Unable to connect to active camera.");
  }
  else
  {
    // Ridimensiona l'oggetto Video per farlo corrispondere alle
    // impostazioni della videocamera e
    // aggiunge il video all'elenco di visualizzazione.
    vid.width = cam.width;
    vid.height = cam.height;
    addChild(vid);
  }
  // Rimuove il listener di eventi status.
  cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}
```

Monitoraggio delle condizioni di riproduzione

La classe Camera contiene alcune proprietà che consentono di monitorare lo stato corrente dell'oggetto Camera. Ad esempio, il codice seguente visualizza diverse delle proprietà della videocamera mediante un oggetto Timer e un'istanza di campo di testo nell'elenco di visualizzazione:

```
var vid:Video;
var cam:Camera = Camera.getCamera();
var tf:TextField = new TextField();
tf.x = 300;
tf.autoSize = TextFieldAutoSize.LEFT;
addChild(tf);

if (cam != null)
{
  cam.addEventListener(StatusEvent.STATUS, statusHandler);
  vid = new Video();
  vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
  if (!cam.muted)
  {
    vid.width = cam.width;
    vid.height = cam.height;
    addChild(vid);
    t.start();
  }
  cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}

var t:Timer = new Timer(100);
t.addEventListener(TimerEvent.TIMER, timerHandler);
function timerHandler(event:TimerEvent):void
{
  tf.text = "";
  tf.appendText("activityLevel: " + cam.activityLevel + "\n");
  tf.appendText("bandwidth: " + cam.bandwidth + "\n");
  tf.appendText("currentFPS: " + cam.currentFPS + "\n");
  tf.appendText("fps: " + cam.fps + "\n");
  tf.appendText("keyFrameInterval: " + cam.keyFrameInterval + "\n");
  tf.appendText("loopback: " + cam.loopback + "\n");
  tf.appendText("motionLevel: " + cam.motionLevel + "\n");
  tf.appendText("motionTimeout: " + cam.motionTimeout + "\n");
  tf.appendText("quality: " + cam.quality + "\n");
}
```

Ogni 1/10 di secondo (100 millisecondi) l'evento `timer` dell'oggetto Timer viene inviato e la funzione `timerHandler()` aggiorna il campo di testo nell'elenco di visualizzazione.

Invio del video a un server

Se si desidera creare applicazioni più complesse che utilizzino oggetti Video o Camera, Flash Media Server offre una combinazione di funzioni di streaming multimediale e un ambiente di sviluppo per la creazione e la distribuzione di applicazioni multimediali a un pubblico di vaste dimensioni. Questa combinazione consente agli sviluppatori di creare applicazioni Video on Demand, trasmissioni di eventi Web dal vivo e streaming di mp3, oltre che blog video, messaggistica video e ambienti chat multimediali. Per ulteriori informazioni, consultare la documentazione online di Flash Media Server all'indirizzo http://livedocs.macromedia.com/fms/2/docs/.

Argomenti avanzati

Gli argomenti seguenti illustrano alcuni particolari problemi che si verificano quando si lavora con il video.

Compatibilità di Flash Player con i file FLV codificati

Flash Player 7 supporta i file FLV codificati con il codec per video Sorenson™ Spark™. Flash Player 8 supporta i file FLV codificati con Sorenson Spark o On2 VP6 in Flash Professional 8. Il codec per video On2 VP6 supporta un canale alfa. Versioni diverse di Flash Player supportano i file FLV in modi differenti. Per ulteriori informazioni, consultare la tabella seguente:

Codec	Versione file SWF (versione di pubblicazione)	Versione Flash Player richiesta per la riproduzione
Sorenson Spark	6	6, 7 o 8.
	7	7, 8
On2 VP6	6	8*
	7	8
	8	8

* Se il file SWF carica un file FLV, è possibile utilizzare il video On2 VP6 con l'obbligo di ripubblicare il file SWF per Flash Player 8, sempre che gli utenti, per visualizzare il file SWF, utilizzino Flash Player 8. Solo Flash Player 8 supporta sia la pubblicazione che la riproduzione di video On2 VP6.

Informazioni sulla configurazione di file FLV per l'hosting su un server

Quando si lavora con i file FLV, è necessario configurare il server così che possa supportare il formato di file FLV. Multipurpose Internet Mail Extensions (MIME) è una specifica di dati standardizzata che consente di inviare file non ASCII attraverso connessioni Internet.
I browser Web e i client di posta elettronica sono configurati in modo da essere in grado di interpretare diversi tipi MIME, così che possano inviare e ricevere video, audio, immagini e testo formattato. Per caricare file FLV da un server Web, potrebbe essere necessario registrare l'estensione e il tipo MIME del file con il server Web. A questo proposito, consultare la documentazione del server Web. Il tipo MIME per i file FLV è `video/x-flv`. Informazioni complete per il tipo di file FLV sono riportate di seguito:

- Tipo Mime: video/x-flv
- Estensione del file: .flv
- Parametri necessari: nessuno
- Parametri facoltativi: nessuno
- Considerazioni sulla codifica: i file FLV sono file binari; alcune applicazioni possono richiedere l'impostazione del sottotipo application/octet-stream.
- Problemi di sicurezza: nessuno
- Specifiche pubblicate: www.adobe.com/go/flashfileformat_it.

Rispetto alle versioni precedenti, Microsoft ha cambiato il modo di gestire i contenuti multimediali in streaming nel server Web Microsoft Internet Information Services (IIS) 6.0. Le versioni precedenti di IIS non richiedono alcuna modifica per eseguire lo streaming di Flash Video. In IIS 6.0, il server Web predefinito fornito con Windows 2003, il server richiede un tipo MIME per riconoscere che i file FLV sono i contenuti multimediali in streaming.

Quando i file SWF che eseguono lo streaming di file FLV esterni vengono posizionati su un sistema operativo Microsoft Windows Server® 2003 e visualizzati in un browser, il file SWF viene riprodotto correttamente, mentre lo streaming del video FLV non viene eseguito. Questo problema riguarda tutti i file FLV posizionati su Windows Server 2003, compresi quelli realizzati con versioni precedenti dello strumento di creazione di Flash, Macromedia Flash Video Kit per Dreamweaver MX 2004 di Adobe. Questi file funzionano correttamente se provati su altri sistemi operativi.

Per informazioni sulla configurazione di Microsoft Windows 2003 e Microsoft IIS Server 6.0 per eseguire lo streaming di video FLV, vedere www.adobe.com/go/tn_19439_it.

Informazioni sull'indirizzamento di file FLV locali in Macintosh

Se ci cerca di riprodurre un file FLV locale da un'unità non di sistema su un computer Apple® Macintosh® utilizzando un percorso contenente una barra relativa (/), il video non verrà riprodotto. Le unità non di sistema comprendono, ma non sono limitate a, CD-ROM, dischi rigidi con partizioni, unità di memorizzazione rimovibili e dispositivi di memorizzazione connessi.

NOTA	Il motivo della mancata riproduzione è una limitazione del sistema operativo e non di Flash Player.

Per riprodurre un file FLV proveniente da un'unità non di sistema in Macintosh, fare riferimento a esso con un percorso assoluto utilizzando una notazione basata sui due punti (:) al posto di una notazione basata sulla barra laterale (/). L'elenco seguente mostra le differenze tra i due tipi di notazione:

- Notazione basata sulla barra laterale: myDrive/myFolder/myFLV.flv
- Notazione basata sui due punti: (Mac OS®) myDrive:myFolder:myFLV.flv

È anche possibile creare un file di proiettore per un CD-ROM che si prevede di utilizzare per la riproduzione in Macintosh. Per informazioni aggiornate sui CD-ROM Mac OS e i file FLV, vedere www.adobe.com/go/3121b301_it.

Esempio: Video Jukebox

L'esempio seguente crea un semplice video jukebox che carica in modo dinamico un elenco di video da riprodurre in sequenza. Viene pertanto creata un'applicazione che consente a un utente di sfogliare tra una serie di esercitazioni video o che specifica quali annunci pubblicitari devono essere riprodotti prima di distribuire il video richiesto dall'utente. L'esempio dimostra le seguenti funzioni di ActionScript 3.0:

- Aggiornamento di un indicatore di riproduzione in base all'avanzamento della riproduzione di un file video
- Intercettazione e analisi dei metadati di un file video
- Gestione di codici specifici in uno streaming di rete
- Caricamento, riproduzione, pausa e interruzione di un file FLV caricato dinamicamente
- Ridimensionamento di un oggetto video nell'elenco di visualizzazione in base ai metadati dello streaming di rete

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione Video Jukebox sono disponibili nella cartella Samples/VideoJukebox. L'applicazione è costituita dai seguenti file:

File	Descrizione
VideoJukebox.as	La classe che fornisce la funzionalità principale dell'applicazione.
VideoJukebox.fla	Il file dell'applicazione principale per Flash.
playlist.xml	Un file che elenca i file video che vengono caricati nel video jukebox.

Caricamento di un file di playlist video esterno

Il file playlist.xml esterno specifica quali video caricare e l'ordine in cui devono essere riprodotti. Per poter caricare il file XML, è necessario utilizzare un oggetto URLLoader e un oggetto URLRequest, come mostrato nel codice seguente:

```
uldr = new URLLoader();
uldr.addEventListener(Event.COMPLETE, xmlCompleteHandler);
uldr.load(new URLRequest(PLAYLIST_XML_URL));
```

Questo codice viene collocato nella funzione di costruzione della classe VideoJukebox in modo che il file venga caricato prima che venga eseguito qualunque altro codice. Non appena è terminato il caricamento del file XML, viene chiamato il metodo `xmlCompleteHandler()` che analizza il file esterno in un oggetto XML, come mostrato nel codice seguente:

```
private function xmlCompleteHandler(event:Event):void
{
  playlist = XML(event.target.data);
  videosXML = playlist.video;
  main();
}
```

L'oggetto playlist XML contiene i dati XML originari del file esterno, mentre videosXML è un oggetto XMLList che contiene solo i nodi video. Lo snippet di codice seguente contiene un esempio di un file playlist.xml:

```
<videos>
  <video url="video/caption_video.flv" />
  <video url="video/cuepoints.flv" />
  <video url="video/water.flv" />
</videos>
```

Infine, il metodo `xmlCompleteHandler()` chiama il metodo `main()` che imposta le istanze dei vari componenti nell'elenco di visualizzazione, oltre agli oggetti NetConnection e NetStream che vengono utilizzati per caricare i file FLV esterni.

Creazione dell'interfaccia utente

Per creare l'interfaccia utente è necessario trascinare cinque istanze Button nell'elenco di visualizzazione e assegnare a esse i seguenti nomi di istanza: `playButton`, `pauseButton`, `stopButton`, `backButton` e `forwardButton`.

Per ognuna di queste istanze Button, è necessario assegnare un gestore per l'evento `click`, come mostrato nello snippet di codice seguente:

```
playButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
pauseButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
stopButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
backButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
forwardButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
```

Il metodo `buttonClickHandler()` utilizza un'istruzione switch per determinare su quale istanza Button è stato fatto clic, come mostrato nel codice seguente:

```
private function buttonClickHandler(event:MouseEvent):void
{
  switch (event.currentTarget)
  {
    case playButton:
      ns.resume();
      break;
    case pauseButton:
      ns.togglePause();
      break;
    case stopButton:
      ns.pause();
      ns.seek(0);
      break;
    case backButton:
      playPreviousVideo();
      break;
    case forwardButton:
      playNextVideo();
      break;
  }
}
```

Quindi, aggiungere un'istanza Slider all'elenco di visualizzazione e assegnarle il nome di istanza `volumeSlider`. Il codice seguente imposta la proprietà `liveDragging` dell'istanza Slider su `true` e definisce per quest'ultima un listener di eventi per l'evento `change`:

```
volumeSlider.value = volumeTransform.volume;
volumeSlider.minimum = 0;
volumeSlider.maximum = 1;
volumeSlider.snapInterval = 0.1;
volumeSlider.tickInterval = volumeSlider.snapInterval;
volumeSlider.liveDragging = true;
volumeSlider.addEventListener(SliderEvent.CHANGE, volumeChangeHandler);
```

Aggiungere un'istanza ProgressBar all'elenco di visualizzazione e assegnarle il nome di istanza `positionBar`. Impostarne la proprietà `mode` su manual, come mostrato nello snippet di codice seguente:

```
positionBar.mode = ProgressBarMode.MANUAL;
```

Infine, aggiungere un'istanza Label all'elenco di visualizzazione e assegnarle il nome di istanza `positionLabel`. Il valore di questa istanza Label viene impostata dall'istanza Timer

Intercettazione dei metadati di un oggetto Video

Quando Flash Player incontra dei metadati per ognuno dei video caricati, il gestore di callback `onMetaData()` viene chiamato sulla proprietà `client` dell'oggetto NetStream. Il codice seguente inizializza un oggetto e imposta il gestore di callback specificato:

```
client = new Object();
client.onMetaData = metadataHandler;
```

Il metodo `metadataHandler()` ne copia i dati nella proprietà meta definita precedentemente nel codice. Ciò consente di accedere ai metadati del video corrente in qualunque momento e in qualsiasi parte dell'intera applicazione. Quindi, l'oggetto Video sullo stage viene ridimensionato affinché corrisponda alle dimensioni restituite dai metadati. Infine, l'istanza della barra di avanzamento positionBar viene spostata e ridimensionata in base alle dimensioni del video che è in corso di riproduzione. Il codice seguente contiene l'intero metodo `metadataHandler()`:

```
private function metadataHandler(metadataObj:Object):void
{
  meta = metadataObj;
  vid.width = meta.width;
  vid.height = meta.height;
  positionBar.move(vid.x, vid.y + vid.height);
  positionBar.width = vid.width;
}
```

Caricamento dinamico di un video Flash

Per caricare in modo dinamico ognuno dei video Flash, l'applicazione utilizza un oggetto NetConnection e un oggetto NetStream. Il codice seguente crea un oggetto NetConnection e passa il valore `null` al metodo `connect()`. Specificando `null`, Flash Player si collega a un video sul server locale anziché collegarsi a un server quale Flash Media Server.

Il codice seguente crea sia l'istanza NetConnection che l'istanza NetStream, definisce un listener di eventi per l'evento `netStatus` e assegna l'oggetto `client` alla proprietà `client`:

```
nc = new NetConnection();
nc.connect(null);

ns = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
ns.client = client;
```

Il metodo `netStatusHandler()` viene chiamato ogni qual volta lo stato del video viene modificato; ad esempio quando la riproduzione del video viene avviata o interrotta, quando il video è in fase di bufferizzazione o se non è possibile trovare uno streaming video. Il codice seguente elenca l'evento `netStatusHandler()`:

```
private function netStatusHandler(event:NetStatusEvent):void
{
  try
  {
    switch (event.info.code)
    {
      case "NetStream.Play.Start":
        t.start();
        break;
      case "NetStream.Play.StreamNotFound":
      case "NetStream.Play.Stop":
        t.stop();
        playNextVideo();
        break;
    }
  }
  catch (error:TypeError)
  {
    // Ignora qualunque errore.
  }
}
```

Il codice precedente valuta la proprietà code dell'oggetto info e filtra se il codice è "NetStream.Play.Start", "NetStream.Play.StreamNotFound" o "NetStream.Play.Stop". Tutti gli altri codici vengono ignorati. Se lo streaming di rete è in fase di avvio, il codice avvia l'istanza Timer che aggiorna l'indicatore di riproduzione. Se non è possibile trovare lo streaming di rete oppure è fermo, l'istanza Timer viene interrotta e l'applicazione tenta di riprodurre il video successivo all'interno della playlist.

Ogni volta che viene eseguita l'istanza Timer, la barra di avanzamento `positionBar` aggiorna la propria posizione corrente chiamando il metodo `setProgress()` della classe ProgressBar e l'istanza Label `positionLabel` viene aggiornata con il tempo trascorso e il tempo totale del video corrente.

```
private function timerHandler(event:TimerEvent):void
{
  try
  {
    positionBar.setProgress(ns.time, meta.duration);
    positionLabel.text = ns.time.toFixed(1) + " of "
  meta.duration.toFixed(1) + " seconds";
  }
  catch (error:Error)
  {
    // Ignora questo errore.
  }
}
```

Controllo del volume del video

È possibile controllare il volume del video caricato dinamicamente impostando la proprietà `soundTransform` sull'oggetto NetStream. L'applicazione del video jukebox consente di modificare il livello del volume modificando il valore dell'istanza Slider `volumeSlider`. Il codice seguente mostra come modificare il livello del volume assegnando il valore del componente Slider a un oggetto SoundTransform che è impostato sulla proprietà `soundTransform` dell'oggetto NetStream:

```
private function volumeChangeHandler(event:SliderEvent):void
{
  volumeTransform.volume = event.value;
  ns.soundTransform = volumeTransform;
}
```

Controllo della riproduzione video

Il resto dell'applicazione controlla la riproduzione video quando il video raggiunge la fine dello streaming video o l'utente salta al video precedente o successivo.

Il metodo seguente recupera l'URL del video dall'elemento XMLList dell'indice attualmente selezionato:

```
private function getVideo():String
{
  return videosXML[idx].@url;
}
```

Il metodo `playVideo()` chiama il metodo `play()` sull'oggetto NetStream per caricare il video attualmente selezionato:

```
private function playVideo():void
{
  var url:String = getVideo();
  ns.play(url);
}
```

Il metodo `playPreviousVideo()` decrementa l'indice corrente del video, chiama il metodo `playVideo()` per caricare il nuovo file video e imposta la barra di avanzamento come visibile:

```
private function playPreviousVideo():void
{
  if (idx > 0)
  {
    idx--;
    playVideo();
    positionBar.visible = true;
  }
}
```

L'ultimo metodo, `playNextVideo()`, incrementa l'indice del video e chiama il metodo `playVideo()`. Se il video corrente è l'ultimo della playlist, il metodo `clear()` viene chiamato sull'oggetto Video e la proprietà `visible` dell'istanza della barra di avanzamento viene impostata su `false`:

```
private function playNextVideo():void
{
  if (idx < (videosXML.length() - 1))
  {
    idx++;
    playVideo();
    positionBar.visible = true;
  }
  else
  {
    idx++;
    vid.clear();
    positionBar.visible = false;
  }
}
```

CAPITOLO 20

Operazioni con l'audio

20

ActionScript è stato creato per le applicazioni interattive di grande impatto visivo, e un elemento spesso trascurato di questo genere di applicazioni è l'audio. È possibile aggiungere effetti sonori a un videogioco o un feedback audio all'interfaccia utente di un'applicazione, o perfino creare un programma specifico per l'audio che analizzi i file mp3 caricati in Internet.

In questo capitolo vengono fornite informazioni relative al caricamento di file audio esterni e alle operazioni con l'audio incorporato in un file SWF. Inoltre, viene illustrato come controllare l'audio, come creare delle rappresentazioni visive delle informazioni relative ai suoni e come acquisire l'audio dal microfono di un utente.

Sommario

Nozioni fondamentali sulle operazioni con l'audio

Introduzione alle operazioni con l'audio

Esattamente come codificano le immagini in un formato digitale, le memorizzano e le recuperano per la visualizzazione sullo schermo, i computer sono in grado di acquisire e codificare l'audio digitale (cioè, la rappresentazione informatica delle informazioni relative ai suoni) e di memorizzarlo e recuperarlo per riprodurlo mediante gli altoparlanti collegati al computer. Un modo per riprodurre l'audio consiste nell'utilizzare Adobe Flash Player e ActionScript.

Quando vengono convertiti in formato digitale, i dati audio hanno diverse caratteristiche, tra cui il volume e l'indicazione se si tratta di audio stereo o mono. Quando si riproduce un suono in ActionScript, è possibile regolare queste caratteristiche: ad esempio, riprodurlo a un volume più elevato o fare in modo che sembri provenire da una determinata direzione.

Per poter controllare un suono in ActionScript, è necessario che le informazioni relative a esso siano state caricate in Flash Player. Sono disponibili quattro modi per ottenere i dati audio in Flash Player per utilizzarli mediante ActionScript. È possibile caricare un file audio esterno (ad esempio, un file mp3) nel file SWF; è possibile incorporare direttamente le informazioni relative al suono nel file SWF quando quest'ultimo viene creato; è possibile ricevere l'input audio mediante un microfono collegato al computer di un utente; infine, è possibile accedere ai dati audio che vengono trasmessi in streaming da un server.

Quando si caricano dei dati audio da un file audio esterno, è possibile iniziare la riproduzione dell'inizio del file mentre il resto dell'audio è ancora in fase di caricamento.

Benché esistano diversi formati di file audio per codificare l'audio digitale, ActionScript 3.0 e Flash Player supportano i file audio memorizzati in formato mp3. Non possono caricare o riprodurre direttamente i file audio in formati tipo WAV o AIFF.

Quando si lavora con l'audio in ActionScript, è probabile che vengano utilizzate diverse classi del pacchetto flash.media. La classe Sound è la classe utilizzata per ottenere l'accesso alle informazioni sull'audio caricando un file audio e iniziandone la riproduzione. Una volta iniziata la riproduzione di un suono, Flash Player fornisce l'accesso a un oggetto SoundChannel. Dal momento che un file audio caricato potrebbe essere un suono tra i tanti che vengono riprodotti sul computer di un utente, ogni singolo suono che viene riprodotto utilizza il proprio oggetto SoundChannel; ciò che viene riprodotto dagli altoparlanti del computer è l'output combinato di tutti gli oggetti SoundChannel combinati. Questa istanza SoundChannel viene utilizzata per controllare le proprietà del suono e per interromperne la riproduzione. Infine, se si desidera controllare l'audio combinato, la classe SoundMixer fornisce il controllo sull'output combinato.

È anche possibile utilizzare altre classi per eseguire operazioni più specifiche quando si lavora con l'audio in ActionScript; per ulteriori informazioni su tutte le classi relative all'audio, vedere "Nozioni fondamentali sull'architettura audio" a pagina 637.

Operazioni comuni con l'audio

Questo capitolo descrive le seguenti attività relative all'audio:

- Caricamento di file mp3 esterni e rilevamento dell'avanzamento del caricamento
- Riproduzione, pausa, ripresa della riproduzione e interruzione dell'audio
- Riproduzione dell'audio in streaming durante il caricamento
- Regolazione del volume e della panoramica dell'audio
- Recupero dei metadati ID3 da un file mp3
- Uso dei dati relativi alla forma d'onda priva di compressione
- Acquisizione e riproduzione dell'input audio dal microfono di un utente

Concetti e termini importanti

L'elenco di riferimento seguente contiene dei termini importanti che vengono citati in questo capitolo:

- Ampiezza: la distanza di un punto sulla forma d'onda sonora rispetto allo zero o alla linea di equilibrio.
- Velocità di trasferimento: la quantità di dati che vengono codificati o trasmessi in streaming per ciascun secondo di un file audio. Per i file mp3, la velocità di trasferimento viene di solito indicata in migliaia di bit al secondo, o kbps (kilo bits per second). Una velocità di trasferimento superiore di solito indica un'onda sonora di maggiore qualità.

- Buffering: il processo di ricezione e memorizzazione dei dati audio prima della riproduzione.
- mp3: il formato MPEG-1 Audio Layer 3, o mp3, è un formato di compressione audio molto diffuso.
- Panoramica (o panning): il posizionamento di un segnale audio rispetto ai canali sinistro e destro in un campo sonoro stereo.
- Picco: il punto più alto all'interno in una forma d'onda.
- Frequenza di campionamento: definisce il numero di campioni al secondo estratti da un segnale audio analogico per renderlo un segnale digitale. La frequenza di campionamento dell'audio di un compact disc standard è di 44.1 kHz o 44.100 campioni al secondo.
- Streaming: il processo in base al quale vengono riprodotte le porzioni iniziali di un file audio o video mentre le porzioni finali dello stesso file sono ancora in fase di caricamento da un server.
- Volume: la rumorosità di un suono.
- Forma d'onda: la forma di un grafico delle ampiezze di un segnale audio che variano nel tempo.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché questo capitolo descrive le operazioni con l'audio in ActionScript, molti esempi eseguono azioni che comprendono operazioni con un file audio, ad esempio l'avvio della riproduzione, l'interruzione della riproduzione oppure la regolazione dell'audio. Per provare gli esempi contenuti in questo capitolo:

1. Creare un nuovo documento Flash e salvarlo nel computer.
2. Selezionare il primo fotogramma chiave nella linea temporale e aprire il pannello Azioni.
3. Copiare l'esempio di codice nel riquadro dello script.
4. Se il codice prevede il caricamento di un file audio esterno, è presente una riga di codice simile alla seguente:

   ```
   var req:URLRequest = new URLRequest("click.mp3");
   var s:Sound = new Sound(req);
   ```

   dove "click.mp3" è il nome del file audio che verrà caricato. Per provare questi esempi, è necessario disporre di un file mp3 e collocarlo nella stessa cartella del documento Flash. In seguito, è necessario modificare il codice in modo che utilizzi il nome di questo file mp3 al posto del nome presente nell'esempio di codice (ad esempio, nel codice sopra occorre sostituire "click.mp3" con il nome del proprio file mp3).

5. Dal menu principale, selezionare Controllo > Prova filmato per creare un file SWF ed eseguire l'anteprima (e ascoltarla) del risultato dell'esempio.

Oltre a riprodurre l'audio, alcuni degli esempi visualizzano valori utilizzando la funzione `trace()`; quando si provano questi esempi, i risultati di questi valori vengono visualizzati nel pannello Output. Alcuni esempi inoltre disegnano contenuto sullo schermo, pertanto il loro risultato verrà visualizzato nella finestra di Flash Player.

Per ulteriori informazioni su come provare gli esempi di codice contenuti nel manuale, vedere "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Nozioni fondamentali sull'architettura audio

Le applicazioni possono caricare i dati audio da quattro origini principali:

- File audio esterni caricati in fase di runtime
- Risorse audio incorporate nel file SWF dell'applicazione
- Dati audio provenienti da un microfono collegato al sistema dell'utente
- Dati audio trasmessi in streaming da un server multimediale remoto, come Flash Media Server

I dati audio possono essere caricati completamente prima di essere riprodotti oppure possono essere trasmessi in streaming, ovvero essere riprodotti mentre sono ancora in fase di caricamento.

ActionScript 3.0 e Flash Player supportano i file audio codificati nel formato mp3. Non possono caricare o riprodurre direttamente i file audio in formati tipo WAV o AIFF.

Utilizzando Adobe Flash CS3 Professional, è possibile importare file audio WAV o AIFF e quindi incorporarli nei file SWF dell'applicazione in formato mp3. Lo strumento di creazione di Flash consente anche di comprimere i file audio incorporati per ridurne le dimensioni, tuttavia la riduzione delle dimensioni compromette la qualità audio. Per ulteriori informazioni, vedere "Importazione dell'audio" nella guida *Uso di Flash*.

L'architettura audio di ActionScript 3.0 utilizza le seguenti classi del pacchetto flash.media.

Classe	Descrizione
flash.media.Sound	La classe Sound gestisce il caricamento dell'audio e le proprietà audio di base e avvia la riproduzione dell'audio.
flash.media.SoundChannel	Quando un'applicazione riproduce un oggetto Sound, viene creato un nuovo oggetto SoundChannel per controllare la riproduzione. L'oggetto SoundChannel controlla il volume sia del canale sinistro che del canale destro di riproduzione dell'audio. Ogni suono che viene riprodotto ha un proprio oggetto SoundChannel.
flash.media.SoundLoaderContext	La classe SoundLoaderContext specifica quanti secondi di bufferizzazione utilizzare durante il caricamento di un file audio e se Flash Player cerca un file dei criteri validi su più domini durante il caricamento. Un oggetto SoundLoaderContext viene utilizzato come parametro per il metodo `Sound.load()`.
flash.media.SoundMixer	La classe SoundMixer controlla le proprietà di riproduzione e di sicurezza che riguardano tutti i suoni in un'applicazione. Nella realtà, più canali audio vengono combinati attraverso un oggetto SoundMixer comune, pertanto i valori delle proprietà nell'oggetto SoundMixer agiscono su tutti gli oggetti SoundChannel che sono in corso di riproduzione.
flash.media.SoundTransform	La classe SoundTransform contiene dei valori che controllano il volume e la panoramica dell'audio. Un oggetto SoundTransform può essere applicato, tra gli altri, a un singolo oggetto SoundChannel, all'oggetto SoundMixer globale o a un oggetto Microphone.

Classe	Descrizione
flash.media.ID3Info	Un oggetto ID3Info contiene le proprietà che rappresentano le informazioni dei metadati ID3 che vengono spesso memorizzati nei file audio mp3.
flash.media.Microphone	La classe Microphone rappresenta un microfono o un altro dispositivo di input audio collegato al computer dell'utente. L'input audio proveniente da un microfono può essere instradato agli altoparlanti o inviato a un server remoto. L'oggetto Microphone controlla il guadagno, la frequenza di campionamento e altre caratteristiche del proprio flusso audio.

Ogni risorsa audio che viene caricata e riprodotta richiede una propria istanza della classe Sound e della classe SoundChannel. L'output proveniente da più istanze SoundChannel viene quindi combinato dalla classe SoundMixer globale durante la riproduzione.

Le classi Sound, SoundChannel e SoundMixer non vengono utilizzate per i dati audio ottenuti da un microfono o dallo streaming di un server multimediale come Flash Media Server.

Caricamento di file audio esterni

Ogni istanza della classe Sound serve per caricare e attivare la riproduzione di una risorsa audio specifica. Un'applicazione non può riutilizzare un oggetto Sound per caricare più di un suono. Per creare una nuova risorsa audio, è necessario creare un nuovo oggetto Sound.

Se si carica un file audio di piccole dimensioni, ad esempio un clic da associare a un pulsante, l'applicazione può creare un nuovo oggetto Sound e fare in modo che carichi automaticamente il file audio, come mostrato di seguito:

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
```

La funzione di costruzione `Sound()` accetta un oggetto URLRequest come primo parametro. Quando viene fornito un valore per il parametro URLRequest, il nuovo oggetto Sound inizia automaticamente il caricamento della risorsa audio specificata.

A eccezione dei casi più semplici, l'applicazione deve tenere in considerazione l'avanzamento del caricamento del file audio e controllare il verificarsi di eventuali errori durante il caricamento. Ad esempio, se il suono di un clic ha dimensioni piuttosto elevate, potrebbe non essere stato caricato completamente quando l'utente fa clic sul pulsante che attiva il suono. Il tentativo di riprodurre un suono non caricato può provocare un errore in fase di runtime. Pertanto, è preferibile attendere che il suono venga caricato completamente prima di consentire all'utente di effettuare operazioni che possono avviare la riproduzione di tale suono.

Un oggetto Sound invia una serie di eventi diversi durante il processo di caricamento dell'audio. L'applicazione può intercettare tali eventi per tenere traccia dell'avanzamento del caricamento e garantire che l'audio venga caricato completamente prima di essere riprodotto. La tabella seguente elenca gli eventi che possono essere inviati da un oggetto Sound.

Evento	Descrizione
open (`Event.OPEN`)	Inviato appena prima dell'inizio dell'operazione di caricamento dell'audio.
progress (`ProgressEvent.PROGRESS`)	Inviato periodicamente durante il caricamento dell'audio quando vengono ricevuti i dati dal file o dal flusso.
id3 (`Event.ID3`)	Inviato quando sono disponibili dati ID3 per un file audio mp3.
complete (`Event.COMPLETE`)	Inviato quando sono stati caricati tutti i dati della risorsa audio.
ioError (`IOErrorEvent.IO_ERROR`)	Inviato quando non è possibile individuare un file audio o quando il processo di caricamento viene interrotto prima che siano stati ricevuti tutti i dati audio.

Il codice riportato di seguito illustra come riprodurre un file audio dopo che ne è terminato il caricamento:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(Event.COMPLETE, onSoundLoaded);
var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);
```

```
function onSoundLoaded(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
```

Come prima cosa, l'esempio di codice crea un nuovo oggetto Sound senza assegnare un valore iniziale per il relativo parametro URLRequest. Quindi, intercetta l'evento `Event.COMPLETE` dall'oggetto Sound, provocando l'esecuzione del metodo `onSoundLoaded()` quando sono stati caricati tutti i dati audio. Infine, chiama il metodo `Sound.load()` con un nuovo valore di URLRequest per il file audio.

Il metodo `onSoundLoaded()` viene eseguito quando il caricamento dell'audio è completato. La proprietà target dell'oggetto Event è un riferimento all'oggetto Sound. Quando si chiama il metodo `play()` dell'oggetto Sound, viene avviata la riproduzione dell'audio.

Monitoraggio del processo di caricamento dell'audio

I file audio possono avere dimensioni notevoli e il loro caricamento può richiedere parecchio tempo. Mentre Flash Player consente all'applicazione di riprodurre l'audio anche prima che ne sia terminato il caricamento, è possibile fornire all'utente un'indicazione di quanti dati audio sono stati caricati e di quanti sono già stati riprodotti.

La classe Sound invia due eventi che rendono relativamente facile la visualizzazione dell'avanzamento del caricamento di un file audio: `ProgressEvent.PROGRESS` e `Event.COMPLETE`. L'esempio seguente mostra come utilizzare questi eventi per visualizzare le informazioni sull'avanzamento del caricamento di un file:

```
import flash.events.Event;
import flash.events.ProgressEvent;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
s.addEventListener(Event.COMPLETE, onLoadComplete);
s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);

var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onLoadProgress(event:ProgressEvent):void
{
  var loadedPct:uint =
    Math.round(100 * (event.bytesLoaded / event.bytesTotal));
  trace("The sound is " + loadedPct + "% loaded.");
}
```

```
function onLoadComplete(event:Event):void
{
  var localSound:Sound = event.target as Sound;
  localSound.play();
}
function onIOError(event:IOErrorEvent)
{
  trace("The sound could not be loaded: " + event.text);
}
```

Questo codice crea innanzi tutto un oggetto Sound, quindi aggiunge dei listener a tale oggetto per gli eventi `ProgressEvent.PROGRESS` e `Event.COMPLETE`. Dopo che il metodo `Sound.load()` è stato chiamato e sono stati ricevuti i primi dati dal file audio, si verifica un evento `ProgressEvent.PROGRESS` che attiva il metodo `onSoundLoadProgress()`.

La percentuale dei dati audio caricati è uguale al valore della proprietà `bytesLoaded` dell'oggetto ProgressEvent divisa per il valore della proprietà `bytesTotal`. Le stesse proprietà `bytesLoaded` e `bytesTotal` sono disponibili anche nell'oggetto Sound. L'esempio precedente mostra semplicemente dei messaggi relativi all'avanzamento del caricamento dell'audio, ma è possibile utilizzare i valori `bytesLoaded` e `bytesTotal` per aggiornare i componenti della barra di avanzamento, ad esempio quelli forniti con la struttura Adobe Flex 2 o nello strumento di creazione di Flash.

Questo esempio mostra anche come un'applicazione riconosce e risponde a un errore mentre carica dei file audio. Ad esempio, se non è possibile individuare un file audio con un determinato nome, l'oggetto Sound invia un evento `Event.IO_ERROR`. Nel codice precedente, viene eseguito il metodo `onIOError()`, che visualizza un breve messaggio di errore quando si verifica un errore.

Operazioni con l'audio incorporato

L'uso dell'audio incorporato al posto del caricamento dell'audio da un file esterno è particolarmente utile per i file audio di piccole dimensioni utilizzati come indicatori acustici nell'interfaccia utente di un'applicazione (ad esempio, il suono riprodotto quando si fa clic su un pulsante).

Quando si incorpora un file audio nell'applicazione, le dimensioni del file SWF risultante aumentano della dimensione del file audio. In altre parole, se si incorporano file di grandi dimensioni, il file SWF può raggiungere dimensioni indesiderate.

Il metodo per incorporare un file audio nel file SWF dell'applicazione varia a seconda dell'ambiente di sviluppo.

Uso di un file audio incorporato in Flash

Lo strumento di creazione di Flash consente di importare file audio in diversi formati e di memorizzarli come simboli nella libreria. In seguito, è possibile assegnarli ai fotogrammi della linea temporale oppure ai fotogrammi dello stato di un pulsante, utilizzarli con i comportamenti oppure direttamente nel codice ActionScript. Questa sezione descrive come utilizzare l'audio incorporato nel codice ActionScript mediante lo strumento di creazione di Flash. Per informazioni su altri modi di utilizzare l'audio incorporato in Flash, vedere "Importazione dell'audio" nella guida *Uso di Flash*.

Per incorporare un file audio in un filmato Flash:

1. Selezionare File > Importa > Importa nella libreria, quindi selezionare un file audio e importarlo.
2. Fare clic con il pulsante destro del mouse sul nome del file importato nel pannello Libreria e selezionare Proprietà. Selezionare la casella di controllo Esporta per ActionScript.
3. Nel campo Classe, inserire un nome da utilizzare per l'audio incorporato in ActionScript. Per impostazione predefinita, in questo campo viene utilizzato il nome del file audio. Se il nome file include un punto, come in "DrumSound.mp3", è necessario modificarlo in un nome simile a "DrumSound"; ActionScript non accetta il carattere punto in un nome classe. Il campo Classe base deve rimanere impostato su flash.media.Sound.
4. Fare clic su OK. Potrebbe essere visualizzata una finestra di dialogo che segnala che è impossibile trovare una definizione per la classe nel percorso di classe. Fare clic su OK e continuare. Se è stato specificato un nome che non corrisponde a nessun nome classe del percorso di classe dell'applicazione, una nuova classe che eredita da flash.media.Sound viene generata automaticamente.
5. Per utilizzare l'audio incorporato, occorre fare riferimento al nome classe dell'audio in ActionScript. Ad esempio, il codice seguente inizia con la creazione di una nuova istanza della classe DrumSound generata automaticamente:

   ```
   var drum:DrumSound = new DrumSound();
   var channel:SoundChannel = drum.play();
   ```

   DrumSound è una sottoclasse della classe flash.media.Sound, pertanto eredita i metodi e le proprietà della classe Sound, incluso il metodo `play()` come mostrato sopra.

Operazioni con l'audio in streaming

La riproduzione di un file audio o video mentre i relativi dati sono ancora in fase di caricamento si definisce *streaming*. I file audio esterni che vengono caricati da un server remoto sono spesso in streaming per evitare all'utente di attendere che tutti i dati audio siano stati caricati prima di ascoltare l'audio.

La proprietà `SoundMixer.bufferTime` rappresenta il numero di millisecondi di dati audio che Flash Player deve raccogliere prima di consentire la riproduzione dell'audio. In altre parole, se la proprietà `bufferTime` è impostata su 5000, Flash Player carica almeno 5000 millisecondi di dati dal file audio prima che inizi la riproduzione dell'audio. Il valore predefinito di `SoundMixer.bufferTime` è 1000.

L'applicazione può ignorare il valore globale di `SoundMixer.bufferTime` per un determinato file audio specificando un nuovo valore per `bufferTime` durante il caricamento dell'audio. Per ignorare il tempo di bufferizzazione predefinito, innanzi tutto creare una nuova istanza della classe SoundLoaderContext, impostarne la proprietà `bufferTime` e passarla come parametro al metodo `Sound.load()`, come mostrato di seguito:

```
import flash.media.Sound;
import flash.media.SoundLoaderContext;
import flash.net.URLRequest;

var s:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
var context:SoundLoaderContext = new SoundLoaderContext(8000, true);
s.load(req, context);
s.play();
```

Mentre la riproduzione continua, Flash Player cerca di preservare la dimensione del buffer audio (o di aumentarla). Se i dati audio vengono caricati più rapidamente rispetto alla velocità di riproduzione, la riproduzione continua senza interruzioni. Tuttavia, se la velocità di caricamento dei dati rallenta a causa dei limiti della rete, è possibile che l'indicatore di riproduzione raggiunga la fine del buffer audio. In tal caso, la riproduzione viene sospesa e ripristinata automaticamente non appena sono stati caricati altri dati audio.

Per determinare se la riproduzione viene sospesa perché Flash Player attende il caricamento dei dati, utilizzare la proprietà `Sound.isBuffering`.

Riproduzione dell'audio

La riproduzione dell'audio caricato può consistere semplicemente nella chiamata al metodo `Sound.play()` di un oggetto Sound, come indicato di seguito:

```
var snd:Sound = new Sound(new URLRequest("smallSound.mp3"));
snd.play();
```

Quando si riproduce l'audio mediante ActionScript 3.0, è possibile eseguire le operazioni seguenti:

- Riprodurre un file audio da una posizione iniziale specifica
- Sospendere la riproduzione di un file audio e riprenderla dalla stessa posizione
- Sapere esattamente quando termina la riproduzione di un file audio
- Tenere traccia dell'avanzamento del caricamento di un file audio
- Modificare il volume o la panoramica dell'audio durante la riproduzione

Per eseguire queste operazioni durante la riproduzione, utilizzare le classi SoundChannel, SoundMixer e SoundTransform.

La classe SoundChannel controlla la riproduzione di un singolo suono. La proprietà `SoundChannel.position` può essere considerata una sorta di indicatore di riproduzione, che specifica il punto all'interno dei dati audio che sono attualmente in corso di riproduzione.

Quando un'applicazione chiama il metodo `Sound.play()`, viene creata una nuova istanza SoundChannel per controllare la riproduzione.

L'applicazione può riprodurre un suono da una posizione iniziale specifica passando tale posizione (sotto forma di millisecondi) come parametro `startTime` del metodo `Sound.play()`. Inoltre, può specificare un numero prefissato di ripetizioni del suono in rapida successione passando un valore numerico nel parametro loops del metodo `Sound.play()`.

Quando viene chiamato il metodo `Sound.play()` utilizzando sia un parametro `startTime` che un parametro `loops`, l'audio viene riprodotto ripetutamente ogni volta dallo stesso punto iniziale, come mostrato nel codice seguente:

```
var snd:Sound = new Sound(new URLRequest("repeatingSound.mp3"));
snd.play(1000, 3);
```

In questo esempio, l'audio viene riprodotto da un punto che si trova un secondo dopo l'inizio del suono, per tre volte in successione.

Sospensione e ripresa della riproduzione dell'audio

Se l'applicazione è destinata a riprodurre audio di una certa durata (ad esempio, canzoni o podcast), è consigliabile consentire fornire agli utenti la possibilità di sospendere e riprendere la riproduzione. In ActionScript, un suono non può essere letteralmente sospeso: può solo essere interrotto. Tuttavia, un suono può essere riprodotto a partire da qualunque punto. È possibile registrare la posizione del suono nel momento in cui viene interrotto e successivamente riavviare la riproduzione da tale posizione.

Si supponga di disporre di un codice che carica e riproduce un file audio nel modo seguente:

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var channel:SoundChannel = snd.play();
```

Durante la riproduzione dell'audio, la proprietà `SoundChannel.position` indica il punto all'interno del file audio che è in corso di riproduzione. L'applicazione può memorizzare il valore della posizione prima di interrompere la riproduzione dell'audio, come mostrato di seguito:

```
var pausePosition:int = channel.position;
channel.stop();
```

Per riprendere la riproduzione dell'audio, passare il valore della posizione precedentemente memorizzato per riavviare l'audio dallo stesso punto in cui è stato interrotto.

```
channel = snd.play(pausePosition);
```

Monitoraggio della riproduzione

Talvolta è necessario che l'applicazione sappia quando si interrompe la riproduzione di un suono per poter avviare la riproduzione di un altro suono o per eliminare le risorse utilizzate durante la riproduzione precedente. La classe SoundChannel invia un evento `Event.SOUND_COMPLETE` quando termina la riproduzione del relativo audio. L'applicazione può intercettare questo evento ed eseguire le operazioni appropriate, come mostrato di seguito:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound("smallSound.mp3");
var channel:SoundChannel = snd.play();
s.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

public function onPlaybackComplete(event:Event)
{
  trace("The sound has finished playing.");
}
```

La classe SoundChannel non invia eventi progress durante la riproduzione. Per generare un rapporto sull'avanzamento della riproduzione, l'applicazione può impostare un proprio meccanismo di gestione temporale e tenere traccia della posizione dell'indicatore di riproduzione dell'audio.

Per calcolare la percentuale di audio che è stata riprodotta, è possibile dividere il valore della proprietà `SoundChannel.position` per la lunghezza dei dati audio in corso di riproduzione:

```
var playbackPercent:uint = 100 * (channel.position / snd.length);
```

Tuttavia, questo codice riporta delle percentuali di riproduzione accurate solo se i dati audio sono stati interamente caricati prima dell'inizio della riproduzione. La proprietà `Sound.length` mostra le dimensioni dei dati audio in corso di caricamento, non le dimensioni finali dell'intero file audio. Per tenere traccia dell'avanzamento della riproduzione di un file audio in streaming che è ancora in fase di caricamento, l'applicazione deve stimare le dimensioni finali dell'intero file audio e utilizzare tale valore nei calcoli. È possibile stimare la lunghezza finale dei dati audio mediante le proprietà `bytesLoaded` e `bytesTotal` dell'oggetto Sound, come indicato di seguito:

```
var estimatedLength:int =
  Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
var playbackPercent:uint = 100 * (channel.position / estimatedLength);
```

Il codice seguente carica un file audio di dimensioni maggiori e utilizza l'evento `Event.ENTER_FRAME` come meccanismo di gestione temporale per la visualizzazione dell'avanzamento della riproduzione. Genera un report periodico della percentuale di riproduzione, che viene calcolata dividendo il valore della posizione corrente per la lunghezza totale del file audio:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new
  URLRequest("http://av.adobe.com/podcast/csbu_dev_podcast_epi_2.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
snd.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);
```

```
function onEnterFrame(event:Event):void
{
  var estimatedLength:int =
    Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
  var playbackPercent:uint =
    Math.round(100 * (channel.position / estimatedLength));
  trace("Sound playback is " + playbackPercent + "% complete.");
}

function onPlaybackComplete(event:Event)
{
  trace("The sound has finished playing.");
  removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Una volta avviato il caricamento dei dati audio, questo codice chiama il metodo `snd.play()` e memorizza l'oggetto SoundChannel risultante nella variabile `channel`. Quindi, aggiunge un listener di eventi all'applicazione principale per l'evento `Event.ENTER_FRAME` e un altro listener di eventi all'oggetto SoundChannel per l'evento `Event.SOUND_COMPLETE` che si verifica quando la riproduzione è completata.

Ogni volta che l'applicazione raggiunge un nuovo fotogramma della propria animazione, viene chiamato il metodo `onEnterFrame()`. Il metodo `onEnterFrame()` stima la lunghezza totale del file audio in base alla quantità di dati che sono già stati caricati, quindi calcola e visualizza la percentuale di riproduzione corrente.

Quando è stato riprodotto l'intero file audio, viene eseguito il metodo `onPlaybackComplete()` e viene rimosso il listener di eventi per l'evento `Event.ENTER_FRAME` in modo che non tenti di visualizzare gli aggiornamenti sull'avanzamento dopo che la riproduzione è terminata.

L'evento `Event.ENTER_FRAME` può essere inviato molte volte al secondo. In determinate circostanze, non è necessario visualizzare l'avanzamento della riproduzione con tale frequenza. In tali casi, l'applicazione può impostare il proprio meccanismo di gestione temporale mediante la classe flash.util.Timer (vedere "Operazioni con data e ora" a pagina 205).

Interruzione dell'audio in streaming

Nel processo di riproduzione dell'audio in streaming (cioè, per i file audio che sono ancora in fase di caricamento durante la riproduzione) si verifica una circostanza particolare. Quando l'applicazione chiama il metodo `SoundChannel.stop()` su un'istanza SoundChannel che sta riproducendo un file audio in streaming, la riproduzione si arresta per un fotogramma, quindi, in corrispondenza del fotogramma successivo, ricomincia dall'inizio del file audio. Questa situazione si verifica perché il processo di caricamento è ancora in corso. Per interrompere sia il caricamento che la riproduzione di un file audio in streaming, chiamare il metodo `Sound.close()`.

Considerazioni sulla sicurezza durante il caricamento e la riproduzione dell'audio

La capacità dell'applicazione di accedere ai dati audio può essere limitata in base al modello di sicurezza di Flash Player. Ogni suono è soggetto alle limitazioni di due diverse funzioni di sicurezza sandbox: quella per il contenuto stesso (definita "sandbox del contenuto") e quella per l'applicazione o l'oggetto che carica e riproduce il suono (definita "sandbox del titolare"). Per informazioni generali sul modello di sicurezza di Flash Player e sulla definizione delle funzioni di sicurezza sandbox, vedere "Sicurezza di Flash Player" a pagina 807.

La sandbox del contenuto determina se è possibile estrarre dei dati audio dettagliati dall'audio mediante la proprietà `id3` o il metodo `SoundMixer.computeSpectrum()`. Non limita il caricamento o la riproduzione del file audio.

Il dominio di origine del file audio definisce le limitazioni di sicurezza della sandbox del contenuto. In genere, se un file audio si trova nello stesso dominio o nella stessa cartella del file SWF dell'applicazione o dell'oggetto che lo carica, l'applicazione o l'oggetto ha l'accesso completo a tale file audio. Se l'audio proviene da un dominio diverso rispetto all'applicazione, può essere comunque incluso nella sandbox del contenuto utilizzando un file di criteri validi su più domini.

L'applicazione può passare un oggetto SoundLoaderContext con una proprietà `checkPolicyFile` come parametro per il metodo `Sound.load()`. Se la proprietà `checkPolicyFile` viene impostata su `true`, Flash Player verifica la presenza di un file di criteri validi per domini diversi sul server da cui viene caricato l'audio. Se è presente un file di criteri validi per domini diversi e consente l'accesso al file SWF in fase di caricamento, il file SWF può caricare il file audio, accedere alla proprietà `id3` dell'oggetto Sound e chiamare il metodo `SoundMixer.computeSpectrum()` per i file audio caricati.

La sandbox del titolare controlla la riproduzione locale dell'audio. L'applicazione o l'oggetto che avvia la riproduzione di un file audio definisce la sandbox del titolare.

Il metodo `SoundMixer.stopAll()` interrompe la riproduzione dell'audio in tutti gli oggetti SoundChannel che sono attualmente in corso di riproduzione, a condizione che soddisfino i criteri seguenti:

- I file audio sono stati avviati da oggetti all'interno della stessa sandbox del titolare.
- I file audio appartengono a un'origine con un file di criteri validi su più domini che consente l'accesso al dominio dell'applicazione o dell'oggetto che chiama il metodo `SoundMixer.stopAll()`.

Per determinare se il metodo `SoundMixer.stopAll()` interrompe effettivamente tutti i file audio in corso di riproduzione, l'applicazione può chiamare il metodo `SoundMixer.areSoundsInaccessible()`. Se tale metodo restituisce il valore `true`, alcuni dei file audio in corso di riproduzione si trovano fuori dal controllo della sandbox del titolare corrente e non vengono interrotti dal metodo `SoundMixer.stopAll()`.

Il metodo `SoundMixer.stopAll()` arresta anche l'indicatore di riproduzione, evitando che continui per tutti i file audio che sono stati caricati da file esterni. Tuttavia, è possibile che la riproduzione dei file audio che sono incorporati nei file FLA e associati ai fotogrammi nella linea temporale mediante lo strumento di creazione di Flash venga ripresa se l'animazione passa a un nuovo fotogramma.

Controllo del volume e della panoramica dell'audio

Un singolo oggetto SoundChannel controlla sia il canale sinistro che il canale destro di un file audio stereo. Se un file audio mp3 è mono, i canali sinistro e destro dell'oggetto SoundChannel contengono forme d'onda identiche.

È possibile determinare l'ampiezza di un ogni canale stereo dell'audio in corso di riproduzione mediante le proprietà `leftPeak` e `rightPeak` dell'oggetto SoundChannel. Queste proprietà mostrano l'ampiezza di picco della forma d'onda audio. Non rappresentano quindi il volume di riproduzione effettivo, che è invece una funzione dell'ampiezza della forma d'onda audio e dei valori di volume impostati nell'oggetto SoundChannel e nella classe SoundMixer.

La proprietà pan di un oggetto SoundChannel può essere utilizzata per specificare un livello di volume diverso per ciascuno dei canali (sinistro e destro) durante la riproduzione. La proprietà pan può avere un volume compreso tra -1 e 1, dove -1 indica che il canale sinistro viene riprodotto al volume massimo e il canale destro è silenzioso, mentre 1 indica la situazione opposta. I valori numerici compresi tra -1 e 1 impostano dei valori proporzionali per i canali sinistro e destro, mentre il valore 0 indica che entrambi i canali vengono riprodotti con un livello di volume medio e bilanciato.

Il codice di esempio seguente crea un oggetto SoundTransform con un valore di volume pari a 0.6 e un valore pan pari a -1 (volume massimo per il canale sinistro e canale destro silenzioso). Passa l'oggetto SoundTransform come parametro al metodo `play()`, che applica tale oggetto SoundTransform al nuovo oggetto SoundChannel che viene creato per controllare la riproduzione.

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var trans:SoundTransform = new SoundTransform(0.6, -1);
var channel:SoundChannel = snd.play(0, 1, trans);
```

È possibile alterare il volume e la panoramica dell'audio durante la riproduzione impostando le proprietà `pan` o `volume` di un oggetto SoundTransform e successivamente applicando tale oggetto come proprietà `soundTransform` di un oggetto SoundChannel.

È anche possibile impostare simultaneamente i valori globali per il volume e la panoramica per tutti i file audio mediante la proprietà `soundTransform` della classe SoundMixer, come mostra l'esempio seguente:

```
SoundMixer.soundTransform = new SoundTransform(1, -1);
```

È anche possibile utilizzare un oggetto SoundTransform per impostare i valori del volume e della panoramica per un oggetto Microphone (vedere "Rilevamento dell'input audio" a pagina 658) e per gli oggetti Sprite e SimpleButton.

L'esempio seguente alterna la panoramica di un file audio dal canale sinistro a quello destro e viceversa durante la riproduzione.

```
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var panCounter:Number = 0;
```

```
var trans:SoundTransform;
trans = new SoundTransform(1, 0);
var channel:SoundChannel = snd.play(0, 1, trans);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

addEventListener(Event.ENTER_FRAME, onEnterFrame);

function onEnterFrame(event:Event):void
{
  trans.pan = Math.sin(panCounter);
  channel.soundTransform = trans; // or SoundMixer.soundTransform = trans;
  panCounter += 0.05;
}

function onPlaybackComplete(event:Event):void
{
  removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Questo codice inizia caricando un file audio, quindi creando un nuovo oggetto SoundTransform con il volume impostato su 1 (volume massimo) e la panoramica impostata su 0 (bilanciamento uniforme tra il canale sinistro e destro). Quindi, chiama il metodo `snd.play()`, passando l'oggetto SoundTransform come parametro.

Durante la riproduzione del suono, il metodo `onEnterFrame()` viene eseguito ripetutamente. Il metodo `onEnterFrame()` utilizza la funzione `Math.sin()` per generare un valore compreso tra -1 e 1, un intervallo che corrisponde ai valori accettabili della proprietà `SoundTransform.pan`. La proprietà `pan` dell'oggetto SoundTransform viene impostata sul nuovo valore, quindi la proprietà `soundTransform` del canale viene impostata per utilizzare l'oggetto SoundTransform modificato.

Per eseguire questo esempio, sostituire il nome di file bigSound.mp3 con il nome di un file mp3 locale. Quindi eseguire l'esempio. Sarà possibile udire il volume del canale sinistro aumentare mentre quello del canale destro si attenua, e viceversa.

In questo esempio, lo stesso effetto può essere ottenuto impostando la proprietà `soundTransform` della classe SoundMixer. Tuttavia, in questo modo si agisce sulla panoramica di tutti i file audio in corso di riproduzione, non solo sul singolo file riprodotto dall'oggetto SoundChannel in questione.

Operazioni con i metadati audio

I file audio che utilizzano il formato mp3 possono contenere dati aggiuntivi sull'audio sotto forma di tag ID3.

Non tutti i file mp3 contengono metadati ID3. Quando un oggetto Sound carica un file audio mp3, invia un evento `Event.ID3` se il file audio contiene metadati ID3. Per impedire errori in fase di runtime, l'applicazione deve attendere di aver ricevuto l'evento `Event.ID3` prima di accedere alla proprietà `Sound.id3` di un file audio caricato.

Il codice seguente mostra come riconoscere quando sono stati caricati i metadati ID3 per un file audio:

```
import flash.events.Event;
import flash.media.ID3Info;
import flash.media.Sound;

var s:Sound = new Sound();
s.addEventListener(Event.ID3, onID3InfoReceived);
s.load("mySound.mp3");

function onID3InfoReceived(event:Event)
{
  var id3:ID3Info = event.target.id3;

  trace("Received ID3 Info:");
  for (var propName:String in id3)
  {
    trace(propName + " = " + id3[propName]);
  }
}
```

Questo codice comincia creando un oggetto Sound a cui specifica di intercettare l'evento `Event.ID3`. Quando i metadati ID3 del file audio sono stati caricati, viene chiamato il metodo `onID3InfoReceived()`. L'elemento di destinazione dell'oggetto Event passato al metodo `onID3InfoReceived()` è l'oggetto Sound originale, pertanto il metodo ottiene la proprietà `id3` dell'oggetto Sound e successivamente esegue le iterazioni su tutte le proprie proprietà indicate per tracciarne i valori.

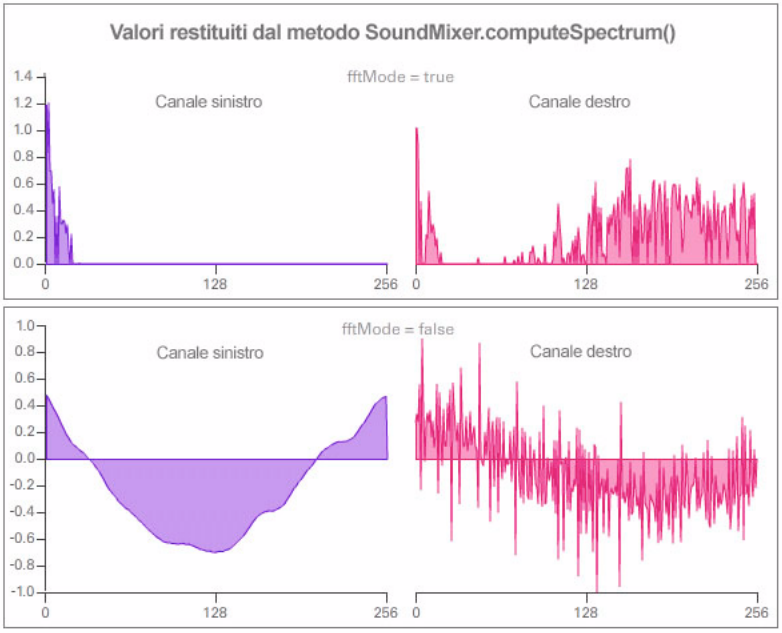
Accesso ai dati audio originari

Il metodo `SoundMixer.computeSpectrum()` consente a un'applicazione di leggere i dati audio originari per la forma d'onda in corso di riproduzione. Se al momento è in corso di riproduzione più di un oggetto SoundChannel, il metodo `SoundMixer.computeSpectrum()` mostra i dati audio combinati di tutti gli oggetti SoundChannel.

I dati audio vengono restituiti sotto forma di un oggetto ByteArray che contiene 512 byte di dati, ciascuno dei quali contiene un valore a virgola mobile compreso tra -1 e 1. Questi valori rappresentano l'ampiezza dei punti all'interno della forma d'onda audio in corso di riproduzione. I valori vengono distribuiti in due gruppi di 256 byte, il primo per il canale stereo sinistro e il secondo per il canale stereo destro.

Il metodo `SoundMixer.computeSpectrum()` restituisce i dati dello spettro di frequenze anziché i dati della forma d'onda se il parametro `FFTMode` è impostato su `true`. Lo spettro di frequenze mostra l'ampiezza organizzata in base alla frequenza audio, dalla più bassa alla più alta. Per convertire i dati della forma d'onda in dati dello spettro di frequenze viene utilizzato un algoritmo Fast Fourier Transform (FFT). I valori dello spettro di frequenze risultanti sono compreso tra 0 e circa 1,414 (la radice quadrata di 2).

Il diagramma seguente confronta i dati restituiti dal metodo `computeSpectrum()` quando il parametro `FFTMode` è impostato su `true` e quando è impostato su `false`. L'audio di cui sono stati utilizzati i dati in questo diagramma contiene un suono basso molto forte nel canale sinistro e il suono di un colpo di tamburo nel canale destro.



Il metodo `computeSpectrum()` può anche restituire dati che sono stati ricampionati a una frequenza di campionamento inferiore. Di solito, ciò produce dati della forma d'onda o della frequenza più attenuati ma anche meno dettagliati. Il parametro `stretchFactor` controlla la frequenza di campionamento con cui vengono campionati i dati del metodo `computeSpectrum()`. Quando il parametro `stretchFactor` è impostato su 0 (valore predefinito), i dati audio vengono campionati a una frequenza di campionamento di 44,1 kHz. La frequenza di campionamento viene dimezzata a ogni successivo valore del parametro stretchFactor, pertanto il valore 1 specifica una frequenza di 22,05 kHz, il valore 2 specifica una frequenza di 11,025 kHz e così via. Il metodo `computeSpectrum()` torna a restituire 256 byte per canale stereo quando si utilizza un valore `stretchFactor` maggiore.

Il metodo `SoundMixer.computeSpectrum()` presenta alcune limitazioni:

- Poiché i dati audio provenienti da un microfono o dai flussi RTMP non passano attraverso l'oggetto SoundMixer globale, il metodo `SoundMixer.computeSpectrum()` non restituisce i dati provenienti da tali origini.
- Se l'origine di uno o più file audio in corso di riproduzione si trova al di fuori della sandbox del contenuto corrente, per motivi di sicurezza il metodo `SoundMixer.computeSpectrum()` genera un errore. Per maggiori dettagli sulle limitazioni di sicurezza del metodo `SoundMixer.computeSpectrum()`, vedere "Considerazioni sulla sicurezza durante il caricamento e la riproduzione dell'audio" a pagina 649 e "Accesso a file multimediali caricati come dati" a pagina 836.

Creazione di un semplice visualizzatore audio

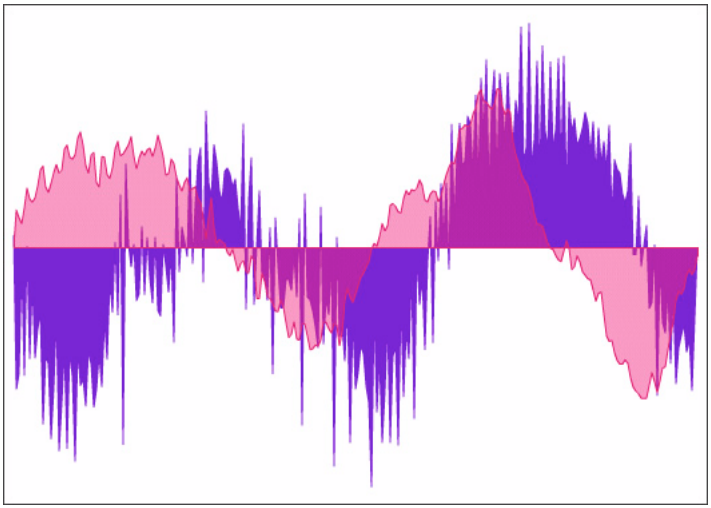
L'esempio seguente utilizza il metodo `SoundMixer.computeSpectrum()` per mostrare un grafico della forma d'onda audio che si anima in corrispondenza di ogni fotogramma:

```
import flash.display.Graphics;
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

const PLOT_HEIGHT:int = 200;
const CHANNEL_LENGTH:int = 256;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);
```

```
var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
snd.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

var bytes:ByteArray = new ByteArray();

function onEnterFrame(event:Event):void
{
  SoundMixer.computeSpectrum(bytes, false, 0);

  var g:Graphics = this.graphics;

  g.clear();
  g.lineStyle(0, 0x6600CC);
  g.beginFill(0x6600CC);
  g.moveTo(0, PLOT_HEIGHT);

  var n:Number = 0;

  // left channel
  for (var i:int = 0; i < CHANNEL_LENGTH; i++)
  {
    n = (bytes.readFloat() * PLOT_HEIGHT);
    g.lineTo(i * 2, PLOT_HEIGHT - n);
  }
  g.lineTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);
  g.endFill();

  // right channel
  g.lineStyle(0, 0xCC0066);
  g.beginFill(0xCC0066, 0.5);
  g.moveTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);

  for (i = CHANNEL_LENGTH; i > 0; i--)
  {
    n = (bytes.readFloat() * PLOT_HEIGHT);
    g.lineTo(i * 2, PLOT_HEIGHT - n);
  }
  g.lineTo(0, PLOT_HEIGHT);
  g.endFill();
}

function onPlaybackComplete(event:Event)
{
  removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Innanzi tutto, questo esempio carica e riproduce un file audio, quindi intercetta l'evento `Event.ENTER_FRAME`, il quale attiva il metodo `onEnterFrame()` durante la riproduzione dell'audio. Il metodo `onEnterFrame()` comincia chiamando il metodo `SoundMixer.computeSpectrum()`, che memorizza i dati della forma d'onda audio nell'oggetto `bytes` di ByteArray.

L'immagine della forma d'onda viene formata mediante l'API di disegno vettoriale. Un ciclo `for` accede in sequenza ai primi 256 valori di dati, che rappresentano il canale stereo sinistro, e disegna una linea da un punto a quello successivo mediante il metodo `Graphics.lineTo()`. Un secondo ciclo `for` accede in sequenza alla serie di 256 valori successiva, ma formandone l'immagine in ordine inverso, da destra a sinistra. Le immagini delle forme d'onda risultanti possono produrre un effetto speculare molto suggestivo, come mostrato nell'immagine seguente.



Rilevamento dell'input audio

La classe Microphone consente all'applicazione di connettersi a un microfono o a un altro dispositivo di input audio presente sul sistema dell'utente e di trasmettere l'input audio agli altoparlanti del sistema o di inviare i dati audio a un server remoto, quale Flash Media Server.

Accesso a un microfono

La classe Microphone non contiene un metodo di costruzione. Per ottenere una nuova istanza Microphone si utilizza piuttosto il metodo statico `Microphone.getMicrophone()`, come mostrato di seguito:

```
var mic:Microphone = Microphone.getMicrophone();
```

Una chiamata al metodo `Microphone.getMicrophone()` senza un parametro restituisce il primo dispositivo di input audio rilevato sul sistema dell'utente.

A un sistema può essere connesso più di un dispositivo di input audio. L'applicazione può utilizzare la proprietà `Microphone.names` per ottenere un array dei nomi di tutti i dispositivi di input audio disponibili. Quindi, chiama il metodo `Microphone.getMicrophone()` con un parametro `index` che corrisponde al valore di indice del nome di un dispositivo nell'array.

A un sistema può non essere connesso alcun microfono o dispositivo di input audio. Per verificare se l'utente ha installato un dispositivo di input audio, è possibile utilizzare la proprietà `Microphone.names` o il metodo `Microphone.getMicrophone()`. Se l'utente non ha installato alcun dispositivo, la lunghezza dell'array `names` è pari a 0 e il metodo `getMicrophone()` restituisce il valore `null`.

Quando l'applicazione chiama il metodo `Microphone.getMicrophone()`, Flash Player visualizza la finestra di dialogo Impostazioni di Flash Player, che richiede all'utente di consentire o negare a Flash Player di accedere alla videocamera e al microfono. Dopo che l'utente ha fatto clic sul pulsante Consenti o sul pulsante Nega in questa finestra di dialogo, viene inviato StatusEvent. La proprietà `code` di tale istanza StatusEvent indica se l'accesso al microfono è stato consentito o negato, come mostrato nell'esempio seguente:

```
import flash.media.Microphone;

var mic:Microphone = Microphone.getMicrophone();
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

function onMicStatus(event:StatusEvent):void
{
  if (event.code == "Microphone.Unmuted")
  {
    trace("Microphone access was allowed.");
  }
  else if (event.code == "Microphone.Muted")
  (
    trace("Microphone access was denied.");
  }
}
```

La proprietà `StatusEvent.code` contiene “Microphone.Unmuted” se l’accesso è stato consentito oppure “Microphone.Muted” se l’accesso è stato negato.

NOTA

La proprietà `Microphone.muted` viene impostata su `true` o `false` quando l’utente rispettivamente consente o nega l’accesso al microfono. Tuttavia, la proprietà `muted` non viene impostata sull’istanza Microphone fino a quando non viene inviato StatusEvent, pertanto l’applicazione deve anche attendere che venga inviato l’evento `StatusEvent.STATUS` prima di verificare la proprietà `Microphone.muted`.

Instradamento dell’audio del microfono agli altoparlanti locali

L’input audio di un microfono può essere instradato agli altoparlanti del sistema locale chiamando il metodo `Microphone.setLoopback()` con un parametro con valore `true`.

Tuttavia, quando l’audio proveniente da un microfono locale viene instradato agli altoparlanti locali, si rischia di produrre un ciclo di feedback audio che può provocare dei suoni fortissimi e stridenti che possono danneggiare l’hardware audio. La chiamata al metodo `Microphone.setUseEchoSuppression()` con un parametro con valore `true` riduce (ma non elimina del tutto) il rischio di feedback audio. Adobe consiglia di chiamare sempre `Microphone.setUseEchoSuppression(true)` prima di chiamare `Microphone.setLoopback(true)`, a meno che non si sia certi che l’utente stia riproducendo l’audio mediante le cuffie o un dispositivo diverso dagli altoparlanti.

Il codice seguente mostra come instradare l'audio da un microfono locale agli altoparlanti del sistema locale:

```
var mic:Microphone = Microphone.getMicrophone();
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

Modifica dell'audio del microfono

L'applicazione può modificare i dati audio provenienti da un microfono in due modi. Innanzi tutto, può modificare il guadagno dell'audio di ingresso, che moltiplica efficacemente i valori di input per una quantità specifica per ottenere un suono più forte o più tenue. La proprietà `Microphone.gain` accetta valori numerici compresi tra 0 e 100 (inclusi). Il valore 50 moltiplica per uno e specifica il volume normale. Il valore 0 moltiplica per zero e rende silenzioso il volume dell'audio in ingresso. I valori superiori a 50 specificano un volume superiore a quello normale.

L'applicazione può anche modificare la frequenza di campionamento dell'audio in ingresso. Le frequenze superiori aumentano la qualità del suono, ma producono flussi di dati più densi la cui trasmissione e memorizzazione richiede più risorse. La proprietà `Microphone.rate` rappresenta la frequenza di campionamento audio misurata in kilohertz (kHz). Il valore predefinito è 8 kHz. È possibile impostare la proprietà `Microphone.rate` su un valore maggiore di 8 kHz se il microfono lo supporta. Ad esempio, se si imposta `Microphone rate` sul valore 11, la frequenza di campionamento viene impostata su 11 kHz; se lo si imposta su 22, la frequenza di campionamento viene impostata su 22 kHz e così via.

Rilevamento dell'attività del microfono

Per preservare larghezza di banda e risorse di elaborazione, Flash Player tenta di rilevare se non viene trasmesso alcun suono da un microfono. Quando il livello dell'attività del microfono rimane al di sotto della soglia del livello di silenzio per un determinato periodo di tempo, Flash Player interrompe la trasmissione dell'input audio e invia un semplice evento ActivityEvent.

Tre proprietà della classe Microphone si occupano di monitorare e controllare il rilevamento dell'attività:

- La proprietà di sola lettura `activityLevel` indica la quantità di suono che viene rilevata dal microfono, su una scala da 0 a 100.
- La proprietà `silenceLevel` specifica la quantità di suono necessaria per attivare il microfono e inviare un evento `ActivityEvent.ACTIVITY`. Anche la proprietà `silenceLevel` utilizza una scala da 0 a 100, e il valore predefinito è 10.

- La proprietà `silenceTimeout` descrive il numero di millisecondi per cui il livello dell'attività deve rimanere al di sotto del livello di silenzio, fino a quando viene inviato un evento `ActivityEvent.ACTIVITY` per indicare che il microfono è silenzioso. Il valore predefinito di `silenceTimeout` è 2000.

Sia la proprietà `Microphone.silenceLevel` che la proprietà `Microphone.silenceTimeout` sono di sola lettura, ma è possibile modificarne i valori utilizzando il metodo `Microphone.setSilenceLevel()`.

In determinati casi, il processo di attivazione del microfono quando viene rilevata della nuova attività può provocare un lieve ritardo. Tale circostanza può essere evitata mantenendo sempre attivo il microfono. L'applicazione può chiamare il metodo `Microphone.setSilenceLevel()` con il parametro `silenceLevel` impostato su zero per specificare a Flash Player di mantenere attivo il microfono e di continuare a raccogliere dati anche se non viene rilevato alcun suono. Al contrario, se si imposta il parametro `silenceLevel` su 100 si impedisce che il microfono venga attivato del tutto.

L'esempio seguente visualizza le informazioni sul microfono e genera un report sugli eventi activity e sugli eventi status inviati da un oggetto Microphone:

```
import flash,events.ActivityEvent;
import flash,events.StatusEvent;
import flash.media.Microphone;

var deviceArray:Array = Microphone.names;
trace("Available sound input devices:");
for (var i:int = 0; i < deviceArray.length; i++)
{
  trace("   " + deviceArray[i]);
}

var mic:Microphone = Microphone.getMicrophone();
mic.gain = 60;
mic.rate = 11;
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
mic.setSilenceLevel(5, 1000);

mic.addEventListener(ActivityEvent.ACTIVITY, this.onMicActivity);
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

var micDetails:String = "Sound input device name: " + mic.name + '\n';
micDetails += "Gain: " + mic.gain + '\n';
micDetails += "Rate: " + mic.rate + " kHz" + '\n';
micDetails += "Muted: " + mic.muted + '\n';
micDetails += "Silence level: " + mic.silenceLevel + '\n';
micDetails += "Silence timeout: " + mic.silenceTimeout + '\n';
micDetails += "Echo suppression: " + mic.useEchoSuppression + '\n';
trace(micDetails);
```

```
function onMicActivity(event:ActivityEvent):void
{
  trace("activating=" + event.activating + ", activityLevel=" +
    mic.activityLevel);
}

function onMicStatus(event:StatusEvent):void
{
  trace("status: level=" + event.level + ", code=" + event.code);
}
```

Quando si esegue il suddetto esempio, parlare o produrre dei suoni nel microfono di sistema e osservare le istruzioni trace risultanti che vengono visualizzati sulla console o in una finestra di debug.

Invio di audio verso e da un server multimediale

Quando si utilizza ActionScript con un server per lo streaming di media audio e video come Flash Media Server (FMS) sono disponibili delle funzionalità audio aggiuntive.

In particolare, l'applicazione può associare un oggetto Microphone a un oggetto NetStream e trasmettere i dati direttamente dal microfono dell'utente al server. I dati audio possono anche essere inviati in streaming dal server a un'applicazione Flash o Flex e riprodotti come parte di un oggetto MovieClip o utilizzando un oggetto Video.

Per ulteriori informazioni, consultare la documentazione online di Flash Media Server all'indirizzo http://livedocs.adobe.com.

Esempio: Podcast Player

Un podcast è un file audio che viene distribuito su Internet, su richiesta ("on demand") o su abbonamento. I podcast vengono solitamente pubblicati come parti di una serie, definita anche canale podcast. Poiché possono avere una durata molto variabile (da un minuto a molte ore), gli episodi di un podcast vengono di solito riprodotti in streaming. Gli episodi di un podcast, definiti anche elementi, vengono generalmente distribuiti in formato mp3. Anche i podcast video sono molto popolari, ma questa applicazione di esempio riproduce solo podcast audio che utilizzano file mp3.

Questo esempio non rappresenta un aggregatore di podcast completo. Ad esempio, non gestisce gli abbonamenti a podcast specifici né memorizza quali podcast sono stati ascoltati dall'utente al successivo avvio dell'applicazione. Può tuttavia fungere da punto di partenza per un aggregatore di podcast dotato di maggiori funzionalità.

L'esempio Podcast Player illustra le seguenti tecniche di programmazione ActionScript:

- Lettura di un feed RSS esterno e analisi del suo contenuto XML
- Creazione di una classe SoundFacade per semplificare il caricamento e la riproduzione dei file audio
- Visualizzazione dell'avanzamento della riproduzione dell'audio
- Sospensione e ripresa della riproduzione dell'audio

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione Podcast Player sono disponibili nella cartella Samples/PodcastPlayer. L'applicazione è costituita dai seguenti file:

File	Descrizione
PodcastPlayer.mxml oppure PodcastPlayer.fla	L'interfaccia utente dell'applicazione per Flex (MXML) o Flash (FLA).
RSSBase.as	Una classe di base che fornisce le proprietà e i metodi comuni per le classi RSSChannel e RSSItem.
RSSChannel.as	Una classe ActionScript che contiene i dati relativi a un canale RSS.
RSSItem.as	Una classe ActionScript che contiene i dati relativi a un elemento RSS.
SoundFacade.as	La classe ActionScript principale dell'applicazione, che incorpora i metodi e gli eventi della classe Sound e della classe SoundChannel e che aggiunge il supporto per la sospensione e la ripresa della riproduzione.
URLService.as	Una classe ActionScript che recupera i dati da un URL remoto.
playerconfig.xml	Un file XML che contiene un elenco di feed RSS che rappresentano i canali podcast.

Lettura dei dati RSS per un canale podcast

L'applicazione Podcast Player comincia leggendo le informazioni relative a un numero di canali podcast e ai relativi episodi:

1. Innanzi tutto, l'applicazione legge un file di configurazione XML che contiene un elenco di canali podcast e visualizza l'elenco dei canali per l'utente.

2. Quando l'utente seleziona uno dei canali podcast, l'applicazione legge il feed RSS per il canale e visualizza un elenco dei relativi episodi.

Questo esempio utilizza la classe di utilità URLLoader per recuperare i dati basati su testo da una posizione remota o da un file locale. L'applicazione Podcast Player crea come prima cosa un oggetto URLLoader per ottenere un elenco di feed RSS in formato XML dal file playerconfig.xml. Quindi, quando l'utente seleziona un feed specifico dall'elenco, viene creato un nuovo oggetto URLLoader per leggere i dati RSS dall'URL del feed.

Semplificazione del caricamento e della riproduzione dell'audio mediante la classe SoundFacade

L'architettura audio di ActionScript 3.0 è potente ma complessa. Le applicazioni che richiedono solo funzioni di base di caricamento e riproduzione audio possono utilizzare una classe che nasconde alcune caratteristiche particolarmente complesse fornendo un set semplificato di chiamate a metodi ed eventi. Nel mondo della progettazione software, una classe di questo tipo viene definita *classe facade* (o "di facciata").

La classe SoundFacade presenta un'unica interfaccia per eseguire le operazioni seguenti:

- Caricamento dei file audio mediante un oggetto Sound, un oggetto SoundLoaderContext e la classe SoundMixer
- Riproduzione dei file audio mediante l'oggetto Sound e l'oggetto SoundChannel
- Invio di eventi progress per la riproduzione
- Sospensione e ripresa della riproduzione dell'audio mediante l'oggetto Sound e l'oggetto SoundChannel

La classe SoundFacade ha lo scopo di offrire la maggior parte delle funzionalità delle classi audio di ActionScript ma con un livello inferiore di complessità.

Il codice seguente illustra la dichiarazione della classe, le proprietà della classe e il metodo di costruzione `SoundFacade()`:

```
public class SoundFacade extends EventDispatcher
{
  public var s:Sound;
  public var sc:SoundChannel;
  public var url:String;
  public var bufferTime:int = 1000;

  public var isLoaded:Boolean = false;
  public var isReadyToPlay:Boolean = false;
  public var isPlaying:Boolean = false;
  public var isStreaming:Boolean = true;
  public var autoLoad:Boolean = true;
  public var autoPlay:Boolean = true;

  public var pausePosition:int = 0;

  public static const PLAY_PROGRESS:String = "playProgress";
  public var progressInterval:int = 1000;
  public var playTimer:Timer;

  public function SoundFacade(soundUrl:String, autoLoad:Boolean = true,
                  autoPlay:Boolean = true, streaming:Boolean = true,
                  bufferTime:int = -1):void
  {
    this.url = soundUrl;

    // Imposta i valori booleani che determinano il comportamento
    // di questo oggetto
    this.autoLoad = autoLoad;
    this.autoPlay = autoPlay;
    this.isStreaming = streaming;

    // Il valore predefinito è il valore bufferTime globale
    if (bufferTime < 0)
    {
      bufferTime = SoundMixer.bufferTime;
    }

    // Mantiene il tempo di bufferizzazione all'interno di un intervallo
    // ragionevole, tra 0 e 30 secondi
    this.bufferTime = Math.min(Math.max(0, bufferTime), 30000);

    if (autoLoad)
    {
      load();
    }
  }
```

La classe SoundFacade estende la classe EventDispatcher in modo da poter inviare degli eventi propri. Innanzi tutto, il codice della classe dichiara le proprietà di un oggetto Sound e di un oggetto SoundChannel. La classe memorizza anche il valore dell'URL del file audio e una proprietà `bufferTime` da utilizzare quando si utilizza audio in streaming. Inoltre, accetta alcuni valori di parametro booleani che influiscono sul comportamento del caricamento e della riproduzione:

- Il parametro `autoLoad` indica all'oggetto che il caricamento dell'audio deve iniziare non appena l'oggetto viene creato.
- Il parametro `autoPlay` indica che la riproduzione dell'audio deve iniziare non appena è stata caricata una quantità sufficiente di dati audio. Se si tratta di audio in streaming, la riproduzione ha inizio non appena è stata caricata la quantità di dati audio specificata dalla proprietà `bufferTime`.
- Il parametro `streaming` indica che la riproduzione di questo file audio può iniziare prima che ne sia stato completato il caricamento.

Il valore predefinito del parametro `bufferTime` è -1. Se il metodo di costruzione rileva un valore negativo nel parametro `bufferTime`, imposta la proprietà `bufferTime` sul valore di `SoundMixer.bufferTime`. In questo modo, l'applicazione può essere impostata in modo predefinito sul valore `SoundMixer.bufferTime` globale come desiderato.

Se il parametro `autoLoad` è impostato su `true`, il metodo di costruzione chiama immediatamente il metodo `load()` seguente per avviare il caricamento del file audio:

```
public function load():void
{
  if (this.isPlaying)
  {
    this.stop();
    this.s.close();
  }
  this.isLoaded = false;

  this.s = new Sound();

  this.s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
  this.s.addEventListener(Event.OPEN, onLoadOpen);
  this.s.addEventListener(Event.COMPLETE, onLoadComplete);
  this.s.addEventListener(Event.ID3, onID3);
  this.s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
  this.s.addEventListener(SecurityErrorEvent.SECURITY_ERROR, onIOError);

  var req:URLRequest = new URLRequest(this.url);

  var context:SoundLoaderContext = new SoundLoaderContext(this.bufferTime,
                                                          true);
  this.s.load(req, this.slc);
}
```

Il metodo `load()` crea un nuovo oggetto Sound e successivamente aggiunge i listener per tutti gli eventi audio importanti. Quindi, specifica all'oggetto Sound di caricare il file audio, utilizzando un oggetto SoundLoaderContext per passare il valore `bufferTime`.

Dal momento che la proprietà `url` può essere modificata, è possibile utilizzare un'istanza SoundFacade per riprodurre diversi file audio in successione: per caricare il nuovo file audio, è sufficiente modificare la proprietà `url` e chiamare il metodo `load()`.

I tre eventi del listener di eventi seguenti mostrano il modo in cui l'oggetto tiene traccia dell'avanzamento del caricamento e decide quando iniziare la riproduzione del suono:

```
public function onLoadOpen(event:Event):void
{
  if (this.isStreaming)
  {
    this.isReadyToPlay = true;
    if (autoPlay)
    {
      this.play();
    }
  }
  this.dispatchEvent(event.clone());
}

public function onLoadProgress(event:ProgressEvent):void
{
  this.dispatchEvent(event.clone());
}

public function onLoadComplete(event:Event):void
{
  this.isReadyToPlay = true;
  this.isLoaded = true;
  this.dispatchEvent(evt.clone());

  if (autoPlay && !isPlaying)
  {
    play();
  }
}
```

Il metodo `onLoadOpen()` viene eseguito quando inizia il caricamento dell'audio. Se l'audio può essere riprodotto in streaming, il metodo `onLoadComplete()` imposta immediatamente il flag `isReadyToPlay` su `true`. Il flag `isReadyToPlay` determina se l'applicazione può avviare la riproduzione dell'audio, ad esempio in risposta a un'azione dell'utente (come la selezione di un pulsante Riproduci). La classe SoundChannel gestisce la bufferizzazione dei dati audio, pertanto non è necessario verificare se sono stati caricati dati a sufficienza prima di chiamare il metodo `play()`.

Il metodo `onLoadProgress()` viene eseguito periodicamente durante il processo di caricamento. Si limita a inviare un clone del proprio oggetto ProgressEvent per essere utilizzato dal codice che utilizza l'oggetto SoundFacade.

Quando i dati audio sono stati completamente caricati, viene eseguito il metodo `onLoadComplete()` e, se necessario, viene chiamato il metodo `play()` per l'audio non in streaming. Di seguito viene mostrato il metodo `play()`.

```
public function play(pos:int = 0):void
{
  if (!this.isPlaying)
  {
    if (this.isReadyToPlay)
    {
      this.sc = this.s.play(pos);
      this.sc.addEventListener(Event.SOUND_COMPLETE, onPlayComplete);
      this.isPlaying = true;

      this.playTimer = new Timer(this.progressInterval);
      this.playTimer.addEventListener(TimerEvent.TIMER, onPlayTimer);
      this.playTimer.start();
    }
  }
}
```

Il metodo `play()` chiama il metodo `Sound.play()` se l'audio è pronto per essere riprodotto. L'oggetto SoundChannel risultante viene memorizzato nella proprietà `sc`. Il metodo `play()` crea quindi un oggetto Timer da utilizzare per inviare eventi progress per la riproduzione a intervalli regolari.

Visualizzazione dell'avanzamento della riproduzione

La creazione di un oggetto Timer per gestire il monitoraggio della riproduzione è un'operazione complessa che è consigliabile codificare una sola volta. L'incorporamento di questa logica Timer in una classe riutilizzabile come SoundFacade consente alle applicazioni di intercettare gli stessi eventi di avanzamento sia per il caricamento che per la riproduzione dell'audio.

L'oggetto Timer creato dal metodo `SoundFacade.play()` invia un'istanza TimerEvent ogni secondo. Il metodo `onPlayTimer()` mostrato di seguito viene eseguito ogni volta che viene ricevuta una nuova istanza TimerEvent.

```
public function onPlayTimer(event:TimerEvent):void
{
  var estimatedLength:int =
    Math.ceil(this.s.length / (this.s.bytesLoaded / this.s.bytesTotal));
  var progEvent:ProgressEvent =
    new ProgressEvent(PLAY_PROGRESS, false, false, this.sc.position,
  estimatedLength);
  this.dispatchEvent(progEvent);
}
```

Il metodo `onPlayTimer()` implementa la tecnica di stima delle dimensioni descritta nella sezione "Monitoraggio della riproduzione" a pagina 646. Quindi, crea una nuova istanza ProgressEvent con il tipo di evento `SoundFacade.PLAY_PROGRESS`, con la proprietà `bytesLoaded` impostata sulla posizione corrente dell'oggetto SoundChannel e la proprietà `bytesTotal` impostata sulla lunghezza stimata dei dati audio.

Sospensione e ripresa della riproduzione

Il metodo `SoundFacade.play()` mostrato in precedenza accetta un parametro `pos` che corrisponde a una posizione iniziale all'interno dei dati audio. Se il valore di `pos` è 0, la riproduzione dell'audio comincia dall'inizio.

Anche il metodo `SoundFacade.stop()` accetta un parametro `pos`, come mostrato di seguito:

```
public function stop(pos:int = 0):void
{
  if (this.isPlaying)
  {
    this.pausePosition = pos;
    this.sc.stop();
    this.playTimer.stop();
    this.isPlaying = false;
  }
}
```

Ogni qual volta viene chiamato, il metodo `SoundFacade.stop()` imposta la proprietà `pausePosition` per consentire all'applicazione di sapere dove posizionare l'indicatore di riproduzione se l'utente desidera riprendere la riproduzione dello stesso file audio.

I metodi `SoundFacade.pause()` e `SoundFacade.resume()` mostrati di seguito richiamano rispettivamente i metodi `SoundFacade.stop()` e `SoundFacade.play()`, passando ogni volta un parametro `pos`.

```
public function pause():void
{
  stop(this.sc.position);
}

public function resume():void
{
  play(this.pausePosition);
}
```

Il metodo `pause()` passa il valore `SoundChannel.position` corrente al metodo `play()`, che lo memorizza nella proprietà `pausePosition`. Il metodo `resume()` inizia a riprodurre nuovamente lo stesso file audio utilizzando il valore `pausePosition` come punto iniziale.

Estensione dell'esempio Podcast Player

Questo esempio presenta un Podcast Player molto semplice che illustra l'utilizzo della classe riutilizzabile SoundFacade. Tuttavia, altre funzionalità possono essere aggiunte a questa applicazione per aumentarne l'utilità. Ad esempio, è possibile:

- Memorizzare l'elenco dei feed e delle informazioni sull'utilizzo per ciascun episodio in un'istanza SharedObject che è possibile utilizzare al successivo avvio dell'applicazione.
- Consentire all'utente di aggiungere i propri feed RSS all'elenco dei canali podcast.
- Memorizzare la posizione dell'indicatore di riproduzione quando l'utente interrompe o abbandona un episodio, in modo da poterlo riavviare dallo stesso punto al successivo utilizzo dell'applicazione.
- Scaricare file mp3 di episodi per l'ascolto non in linea, cioè quando l'utente non è collegato a Internet.
- Aggiungere delle funzioni di abbonamento che verificano periodicamente la disponibilità di nuovi episodi in un canale podcast e aggiornano automaticamente l'elenco degli episodi.
- Aggiungere funzioni di ricerca e consultazione dei podcast mediante un'API da un servizio di hosting di podcast come Odeo.com.

CAPITOLO 21

Rilevamento dell'input dell'utente

21

Questo capitolo descrive come creare l'interattività utilizzando ActionScript 3.0 per rispondere all'attività da parte dell'utente. Descrive gli eventi da tastiera e del mouse, quindi passa ad argomenti più avanzati, tra cui la personalizzazione del menu di scelta di rapida e la gestione degli elementi attivi. Il capitolo si chiude con WordSearch, un esempio di applicazione che risponde all'input del mouse.

Il capitolo presuppone una certa dimestichezza con il modello di eventi ActionScript 3.0. Per ulteriori informazioni, vedere Capitolo 10, "Gestione degli eventi" a pagina 335.

Sommario

Elementi fondamentali dell'input dell'utente

Introduzione al rilevamento dell'input dell'utente

L'interazione dell'utente, attraverso la tastiera, il mouse, una video/fotocamera o una combinazione di tutti questi dispositivi costituisce la base dell'interattività. In ActionScript 3.0, l'identificazione e la risposta all'interazione dell'utente riguardano principalmente l'intercettazione di eventi.

La classe InteractiveObject è una sottoclasse della classe DisplayObject e fornisce la struttura comune di eventi e funzionalità necessaria per gestire l'interazione dell'utente. Non è possibile creare direttamente un'istanza della classe InteractiveObject. Gli oggetti di visualizzazione come SimpleButton, Sprite, TextField e vari componenti di Flash e Flex ereditano il modello di interazione dell'utente da questa classe e pertanto condividono una struttura comune. Ciò significa che le tecniche illustrate e il codice scritto per gestire l'interazione dell'utente in un oggetto derivato da InteractiveObject sono validi per tutti gli altri.

Le seguenti operazioni comuni correlate all'interazione dell'utente sono descritte in questo capitolo:

- Rilevamento dell'input da tastiera a livello dell'intera applicazione
- Acquisizione dell'input da tastiera in un oggetto di visualizzazione specifico
- Rilevamento dell'input da mouse a livello dell'intera applicazione
- Acquisizione dell'input da mouse in un oggetto di visualizzazione specifico
- Creazione di un'interattività di trascinamento della selezione
- Creazione di un cursore del mouse (puntatore del mouse) personalizzato
- Aggiunta di nuovi comportamenti al menu di scelta rapida
- Gestione degli elementi attivi

Concetti e termini importanti

Prima di proseguire, è importante sviluppare una certa familiarità con i seguenti termini relativi all'interazione dell'utente:

- Codice di carattere: un codice numerico che rappresenta un carattere nel set di caratteri corrente (associato a un tasto premuto sulla tastiera). Ad esempio, "D" e "d" hanno codici di carattere diversi anche se vengono creati dallo stesso tasto della tastiera.
- Menu di scelta rapida: il menu visualizzato quando un utente fa clic con il pulsante destro del mouse o utilizza una combinazione tastiera-mouse specifica. I comandi del menu di scelta rapida di solito agiscono direttamente sull'elemento su cui è stato fatto clic. Ad esempio, un menu di scelta rapida per un'immagine può contenere un comando per visualizzare l'immagine in una finestra separata e un comando per scaricarla.
- Attivazione: l'indicazione che un elemento selezionato è attivo ed è l'elemento di destinazione dell'interazione della tastiera o del mouse.
- Codice tasto: un codice numerico che corrisponde a un tasto fisico sulla tastiera.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Poiché questo capitolo descrive le operazioni con l'input dell'utente in ActionScript, tutti gli esempi di codice riportati prevedono la manipolazione di alcuni tipi di oggetti di visualizzazione, generalmente un campo di testo o qualsiasi sottoclasse InteractiveObject. Ai fini degli esempi, l'oggetto di visualizzazione potrebbe essere un oggetto creato e posizionato sullo stage in Adobe Flash CS3 Professional oppure un oggetto creato mediante ActionScript. La prova di un esempio prevede la visualizzazione del risultato in Flash Player e l'interazione con l'esempio per vedere gli effetti del codice.

Per provare gli esempi di codice contenuti in questo capitolo:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.
4. Creare un'istanza sullo stage:
   - Se il codice si riferisce a un campo di testo, utilizzare lo strumento Testo per creare un campo di testo dinamico sullo stage.
   - In alternativa, creare l'istanza di un simbolo di clip filmato o pulsante sullo stage.
5. Selezionare il campo di testo, pulsante o clip filmato e assegnarvi un nome di istanza nella finestra di ispezione Proprietà. Questo nome deve corrispondere al nome dell'oggetto di visualizzazione nell'esempio di codice, ad esempio, se il codice manipola un oggetto chiamato `myDisplayObject`, assegnare anche all'oggetto sullo stage il nome `myDisplayObject`.
6. Eseguire il programma selezionando Controllo > Prova filmato.

   Sullo schermo, l'oggetto di visualizzazione viene manipolato secondo quanto specificato nel codice.

Rilevamento dell'input da tastiera

Gli oggetti di visualizzazione che ereditano il modello di interazione dalla classe InteractiveObject possono rispondere agli eventi da tastiera mediante dei listener di eventi. Ad esempio, è possibile posizionare un listener di eventi sullo stage in modo che intercetti e risponda all'input dalla tastiera. Nel codice seguente, un listener di eventi rileva la pressione di un tasto e visualizza il nome del tasto e le proprietà del codice tasto:

```
function reportKeyDown(event:KeyboardEvent):void
{
  trace("Key Pressed: " + String.fromCharCode(event.charCode) +
    " (character code: " + event.charCode + ")");
}
stage.addEventListener(KeyboardEvent.KEY_DOWN, reportKeyDown);
```

Alcuni tasti, ad esempio Ctrl, generano eventi anche se sono privi di rappresentazione sotto forma di glifo.

Nel codice di esempio precedente, il listener di eventi da tastiera rileva l'input da tastiera per l'intero stage. È anche possibile scrivere un listener di eventi per un oggetto di visualizzazione specifico sullo stage; questo evento viene attivato quando l'oggetto è l'elemento attivo.

Nell'esempio seguente, i comandi da tastiera sono visualizzati nel pannello Output solo quando l'utente digita all'interno dell'istanza TextField. Se si tiene premuto il tasto Maiusc, il bordo dell'istanza TextField diventa temporaneamente rosso.

Questo codice presuppone che sullo stage sia presente un'istanza TextField di nome `tf`.

```
tf.border = true;
tf.type = "input";
tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
tf.addEventListener(KeyboardEvent.KEY_UP,reportKeyUp);

function reportKeyDown(event:KeyboardEvent):void
{
  trace("Key Pressed: " + String.fromCharCode(event.charCode) +
    " (key code: " + event.keyCode + " character code: "
    + event.charCode + ")");
  if (event.keyCode == Keyboard.SHIFT) tf.borderColor = 0xFF0000;
}

function reportKeyUp(event:KeyboardEvent):void
{
  trace("Key Released: " + String.fromCharCode(event.charCode) +
    " (key code: " + event.keyCode + " character code: " +
    event.charCode + ")");
  if (event.keyCode == Keyboard.SHIFT)
  {
    tf.borderColor = 0x000000;
  }
}
```

La classe TextField visualizza anche un evento `textInput` che è possibile intercettare quando un utente immette un testo. Per ulteriori informazioni, vedere "Rilevamento dell'input di testo" a pagina 557.

Nozioni fondamentali sui codici tasto e sui codici di carattere

È possibile accedere alle proprietà `keyCode` e `charCode` di un evento di tastiera per determinare quale tasto è stato premuto e successivamente attivare altre azioni. La proprietà `keyCode` è un valore numerico che corrisponde al valore di un tasto sulla tastiera. La proprietà `charCode` è il valore numerico di tale tasto nel set di caratteri corrente (il set di caratteri predefinito è UTF-8, che supporta l'ASCII).

La differenza principale tra i valori dei codici tasto e i valori dei caratteri consiste nel fatto che il valore di un codice tasto rappresenta un determinato tasto sulla tastiera (l'1 su un tastierino numerico è diverso dall'1 nella fila di tasti superiore, ma il tasto che genera "1" e quello che genera "!" è lo stesso), mentre il valore di un carattere rappresenta un particolare carattere (i caratteri R e r sono diversi).

NOTA

Per le mappature tra i tasti e i relativi valori dei codici di carattere in ASCII, vedere Appendice C, "Tasti della tastiera e valori dei codici tasto" a pagina 761.

Le mappature tra tasti e i relativi codici tasto dipendono dal dispositivo e dal sistema operativo. Per tale motivo, è consigliabile non utilizzare le mappature dei tasti per attivare azioni. Invece, utilizzare i valori di costante predefiniti forniti dalla classe Keyboard per fare riferimento alle proprietà `keyCode` appropriate. Ad esempio, anziché utilizzare la mappatura per il tasto Maiusc, utilizzare la costante `Keyboard.SHIFT` (come mostrato nel codice di esempio precedente).

Nozioni fondamentali sulla priorità di KeyboardEvent

Come accade con altri eventi, la sequenza degli eventi da tastiera viene determinata dalla gerarchia degli oggetti di visualizzazione e non dall'ordine con cui vengono assegnati i metodi `addEventListener()` nel codice.

Ad esempio, si supponga di posizionare un campo di testo di nome `tf` all'interno di un clip filmato di nome `container` e di aggiungere un listener di eventi per un evento da tastiera a entrambe le istanze, come mostra l'esempio seguente:

```
container.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
container.tf.border = true;
container.tf.type = "input";
container.tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);

function reportKeyDown(event:KeyboardEvent):void
{
  trace(event.currentTarget.name + " hears key press: " +
    String.fromCharCode(event.charCode) + " (key code: " +
    event.keyCode + " character code: " + event.charCode + ")");
}
```

Poiché è presente un listener sia sul campo di testo che sul relativo contenitore superiore, la funzione `reportKeyDown()` viene chiamata due volte per ogni comando da tastiera all'interno di TextField. Si noti che per ogni tasto premuto, il campo di testo invia un evento prima del clip filmato `container`.

Il sistema operativo e il browser Web elaborano gli eventi da tastiera prima di Adobe Flash Player. Ad esempio, in Microsoft Internet Explorer, se si preme Ctrl+W viene chiusa la finestra del browser prima che l'eventuale file SWF in essa contenuto invii un evento da tastiera.

Rilevamento dell'input da mouse

I clic del mouse creano degli eventi del mouse che è possibile utilizzare per attivare delle funzionalità interattive. Un listener di eventi può essere aggiunto allo stage per intercettare gli eventi del mouse che si verificano in qualunque punto all'interno del file SWF. È anche possibile aggiungere dei listener di eventi agli oggetti sullo stage che ereditano da InteractiveObject (ad esempio, Sprite o MovieClip); questi listener vengono attivati quando si fa clic sull'oggetto.

Come accade con gli eventi da tastiera, anche gli eventi del mouse si propagano. Nell'esempio seguente, dal momento che `square` è un elemento secondario dello stage, l'evento viene inviato sia dallo Sprite `square` che dall'oggetto Stage quando viene fatto clic sul quadrato:

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0xFF0000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.addEventListener(MouseEvent.CLICK, reportClick);
square.x =
square.y = 50;
addChild(square);

stage.addEventListener(MouseEvent.CLICK, reportClick);

function reportClick(event:MouseEvent):void
{
  trace(event.currentTarget.toString() +
    " dispatches MouseEvent. Local coords [" +
    event.localX + "," + event.localY + "] Stage coords [" +
    event.stageX + "," + event.stageY + "]");
}
```

Nell'esempio precedente, si noti che l'evento mouse contiene le informazioni sul clic. Le proprietà `localX` e `localY` contengono la posizione del clic sull'elemento di livello più basso nella catena di visualizzazione. Ad esempio, se si fa clic sull'angolo in alto a sinistra di `square` vengono visualizzate le coordinate locali [0,0], dal momento che si tratta del punto di registrazione di `square`. In alternativa, le proprietà `stageX` e `stageY` fanno riferimento alle coordinate globali del clic sullo stage. Lo stesso clic visualizza [50,50] per queste coordinate, dal momento che `square` è stato spostato in corrispondenza di queste coordinate. Entrambe queste coppie di coordinate possono essere utili a seconda del tipo di risposta che si desidera dare all'interazione dell'utente.

L'oggetto MouseEvent contiene le proprietà booleane `altKey`, `ctrlKey` e `shiftKey`, che possono essere utilizzate per verificare se al momento del clic del mouse viene premuto anche il tasto Alt, Ctrl o Maiusc.

Creazione della funzionalità di trascinamento della selezione

La funzionalità di trascinamento della selezione consente agli utenti selezionare un oggetto mentre premono il pulsante sinistro del mouse, spostarlo in una nuova posizione sullo schermo e qui posizionarlo rilasciando il pulsante sinistro del mouse. Nel codice seguente ne viene illustrato un esempio:

```
import flash.display.Sprite;
import flash.events.MouseEvent;

var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFFCC00);
circle.graphics.drawCircle(0, 0, 40);

var target1:Sprite = new Sprite();
target1.graphics.beginFill(0xCCFF00);
target1.graphics.drawRect(0, 0, 100, 100);
target1.name = "target1";

var target2:Sprite = new Sprite();
target2.graphics.beginFill(0xCCFF00);
target2.graphics.drawRect(0, 200, 100, 100);
target2.name = "target2";

addChild(target1);
addChild(target2);
addChild(circle);

circle.addEventListener(MouseEvent.MOUSE_DOWN, mouseDown)

function mouseDown(event:MouseEvent):void
{
  circle.startDrag();
}
circle.addEventListener(MouseEvent.MOUSE_UP, mouseReleased);

function mouseReleased(event:MouseEvent):void
{
  circle.stopDrag();
  trace(circle.dropTarget.name);
}
```

Per informazioni dettagliate, vedere “Creazione di un’interazione di trascinamento della selezione” a pagina 424.

Personalizzazione del cursore del mouse

Il cursore del mouse (puntatore) può essere nascosto o sostituito con qualunque oggetto di visualizzazione sullo stage. Per nascondere il cursore del mouse, chiamare il metodo `Mouse.hide()`. È possibile personalizzare il cursore chiamando `Mouse.hide()`, intercettando sullo stage l'evento `MouseEvent.MOUSE_MOVE` e impostando le coordinate di un oggetto di visualizzazione (il cursore personalizzato) sulle proprietà `stageX` e `stageY` dell'evento. Nell'esempio seguente viene illustrata un'esecuzione di base di questa operazione:

```
var cursor:Sprite = new Sprite();
cursor.graphics.beginFill(0x000000);
cursor.graphics.drawCircle(0,0,20);
cursor.graphics.endFill();
addChild(cursor);

stage.addEventListener(MouseEvent.MOUSE_MOVE,redrawCursor);
Mouse.hide();

function redrawCursor(event:MouseEvent):void
{
  cursor.x = event.stageX;
  cursor.y = event.stageY;
}
```

Personalizzazione del menu di scelta rapida

Ogni oggetto che eredita dalla classe InteractiveObject può avere un menu di scelta rapida univoco, visualizzato quando l'utente fa clic con il pulsante destro del mouse all'interno del file SWF. Alcuni comandi sono inclusi per impostazione predefinita: Avanti, Indietro, Stampa, Qualità e Zoom.

È possibile rimuovere i comandi predefiniti dal menu, a eccezione delle opzioni Impostazioni e Informazioni su. Se si imposta la proprietà dello stage `showDefaultContextMenu` su `false`, vengono rimossi questi comandi dal menu di scelta rapida.

Per creare un menu di scelta rapida personalizzato per un oggetto di visualizzazione specifico, creare una nuova istanza della classe ContextMenu, chiamare il metodo `hideBuiltInItems()` e assegnare tale istanza alla proprietà `contextMenu` di tale istanza DisplayObject. L'esempio seguente fornisce a un quadrato tracciato dinamicamente un comando di un menu di scelta rapida per convertirlo in un colore casuale:

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0x000000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.x =
square.y = 10;
addChild(square);

var menuItem:ContextMenuItem = new ContextMenuItem("Change Color");
menuItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT,changeColor);
var customContextMenu:ContextMenu = new ContextMenu();
customContextMenu.hideBuiltInItems();
customContextMenu.customItems.push(menuItem);
square.contextMenu = customContextMenu;

function changeColor(event:ContextMenuEvent):void
{
  square.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
  return new ColorTransform(Math.random(), Math.random(),
    Math.random(),1,(Math.random() * 512) - 255,
    (Math.random() * 512) -255, (Math.random() * 512) - 255, 0);
}
```

Gestione degli elementi attivi

Un oggetto interattivo può essere attivato, sia a livello di codice che attraverso un'azione dell'utente. In entrambi i casi, l'attivazione imposta la proprietà `focus` dell'oggetto su `true`. Inoltre, se la proprietà `tabEnabled` è impostata su `true`, l'utente può rendere attivo un nuovo oggetto premendo il tasto Tab. Si noti che il valore `tabEnabled` è `false` per impostazione predefinita, a eccezione dei casi seguenti:

- Per un oggetto SimpleButton, il valore è `true`.
- Per un campo di testo di input, il valore è `true`.
- Per un oggetto Sprite o MovieClip con `buttonMode` impostato su `true`, il valore è `true`.

In ognuna di queste situazioni, è possibile aggiungere un listener per `FocusEvent.FOCUS_IN` o `FocusEvent.FOCUS_OUT` per fornire un comportamento aggiuntivo quando cambia l'elemento attivo. Ciò è particolarmente utile per i campi di testo e i form, ma può essere utilizzato anche su sprite, clip filmato o qualunque oggetto che erediti dalla classe InteractiveObject. L'esempio seguente mostra come abilitare il cambiamento ciclico dell'attivazione con il tasto Tab e come rispondere al conseguente evento focus di attivazione. In questo caso, ogni quadrato cambia colore quando diventa l'elemento attivo.

NOTA

Lo strumento di creazione di Flash utilizza delle scelte rapide da tastiera per gestire gli elementi attivi; pertanto è preferibile provare i file SWF in un browser anziché in Flash per simulare correttamente la gestione degli elementi attivi.

```
var rows:uint = 10;
var cols:uint = 10;
var rowSpacing:uint = 25;
var colSpacing:uint = 25;
var i:uint;
var j:uint;
for (i = 0; i < rows; i++)
{
  for (j = 0; j < cols; j++)
  {
    createSquare(j * colSpacing, i * rowSpacing, (i * cols) + j);
  }
}

function createSquare(startX:Number, startY:Number, tabNumber:uint):void
{
  var square:Sprite = new Sprite();
  square.graphics.beginFill(0x000000);
  square.graphics.drawRect(0, 0, colSpacing, rowSpacing);
  square.graphics.endFill();
  square.x = startX;
  square.y = startY;
  square.tabEnabled = true;
  square.tabIndex = tabNumber;
  square.addEventListener(FocusEvent.FOCUS_IN, changeColor);
  addChild(square);
}
function changeColor(event:FocusEvent):void
{
  e.target.transform.colorTransform = getRandomColor();
}
```

```
function getRandomColor():ColorTransform
{
  // Genera valori casuali per i canali di colore rosso, verde e blu.
  var red:Number = (Math.random() * 512) - 255;
  var green:Number = (Math.random() * 512) - 255;
  var blue:Number = (Math.random() * 512) - 255;

  // Crea e restituisce un oggetto ColorTransform con i colori casuali.
  return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

Esempio: WordSearch

Questo esempio illustra l'interazione dell'utente mediante la gestione degli eventi del mouse. Gli utenti creano il maggior numero possibile di parole partendo da una griglia casuale di lettere e movendosi sia in orizzontale che in verticale all'interno della griglia, ma senza usare mai la stessa lettera due volte. L'esempio illustra le seguenti funzioni di ActionScript 3.0:

- Creazione dinamica di una griglia di componenti
- Risposta agli eventi del mouse
- Gestione di un punteggio in base all'interazione con l'utente

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione WordSearch si trovano nella cartella Samples/WordSearch. L'applicazione è costituita dai seguenti file:

File	Descrizione
WordSearch.as	La classe che fornisce la funzionalità principale dell'applicazione.
WordSearch.fla	Il file dell'applicazione principale per Flash.
dictionary.txt	Un file utilizzato per determinare se le parole create sono valide e ortograficamente corrette.

Caricamento di un dizionario

Per poter creare un gioco che si basa sulla ricerca di parole, naturalmente è necessario un dizionario. Nell'esempio è incluso un file di testo di nome dictionary.txt che contiene un elenco di parole separate da ritorni a capo. Una volta creato un array di nome `words`, la funzione `loadDictionary()` richiede questo file e, se viene caricato correttamente, il file diventa una stringa di notevole lunghezza. È possibile analizzare questa stringa in un array di parole utilizzando il metodo `split()`, e creando delle interruzioni in corrispondenza di ciascuna istanza di un ritorno a capo (codice carattere 10). Questa operazione viene effettuata nella funzione `dictionaryLoaded()`:

```
words = dictionaryText.split(String.fromCharCode(10));
```

Creazione dell'interfaccia utente

Una volta memorizzate le parole, è possibile impostare l'interfaccia utente. Creare due istanze Button: una per inviare una parola e l'altra per cancellare una parola che si sta sillabando. In ciascun caso, è necessario rispondere all'input dell'utente intercettando l'evento `MouseEvent.CLICK` trasmesso dal pulsante e successivamente chiamando una funzione. Nella funzione `setupUI()`, questo codice crea i listener sui due pulsanti:

```
submitWordButton.addEventListener(MouseEvent.CLICK,submitWord);
clearWordButton.addEventListener(MouseEvent.CLICK,clearWord);
```

Generazione di un tabellone

Il tabellone di gioco è una griglia di lettere casuali. Nella funzione `generateBoard()` viene creata una griglia bidimensionale nidificando un ciclo all'interno di un altro. Il primo ciclo incrementa le righe mentre il secondo incrementa il numero totale di colonne per riga. Ognuna delle celle create da queste righe e colonne contiene un pulsante che rappresenta una lettera sul tabellone.

```
private function generateBoard(startX:Number,
  startY:Number,
  totalRows:Number,
  totalCols:Number,
  buttonSize:Number):void
{
  buttons = new Array();
  var colCounter:uint;
  var rowCounter:uint;
  for (rowCounter = 0; rowCounter < totalRows; rowCounter++)
  {
```

```
    for (colCounter = 0; colCounter < totalCols; colCounter++)
    {
      var b:Button = new Button();
      b.x = startX + (colCounter*buttonSize);
      b.y = startY + (rowCounter*buttonSize);
      b.addEventListener(MouseEvent.CLICK, letterClicked);
      b.label = getRandomLetter().toUpperCase();
      b.setSize(buttonSize,buttonSize);
      b.name = "buttonRow"+rowCounter+"Col"+colCounter;
      addChild(b);

      buttons.push(b);
    }
  }
}
```

Benché venga aggiunto un listener per un evento `MouseEvent.CLICK` su una sola riga, dal momento che si trova in un ciclo `for`, viene assegnato a ogni istanza di Button. Inoltre, a ogni pulsante viene assegnato un nome derivante dalla sua posizione di riga e colonna, al fine di facilitare il riferimento alla riga e alla colonna di ogni pulsante più avanti nel codice.

Creazione di parole in base all'input dell'utente

Le parole possono essere sillabate selezionando le lettere adiacenti in orizzontale o in verticale, ma non è consentito utilizzare due volte la stessa lettera. Ogni volta che si fa clic viene generato un evento del mouse; a quel punto la parola che l'utente sta creando deve essere verificata per garantire che sia valida come continuazione delle lettere su cui è stato fatto clic in precedenza. Se non è valida, la parola precedente viene rimossa e ne viene iniziata una nuova. Questa verifica viene effettuata nel metodo `isLegalContinuation()`.

```
private function isLegalContinuation(prevButton:Button,
  currButton:Button):Boolean
{
  var currButtonRow:Number = Number(currButton.name.charAt(currButton.name.
  indexOf("Row") + 3));
  var currButtonCol:Number =
  Number(currButton.name.charAt(currButton.name.indexOf("Col") + 3));
  var prevButtonRow:Number =
  Number(prevButton.name.charAt(prevButton.name.indexOf("Row") + 3));
  var prevButtonCol:Number =
  Number(prevButton.name.charAt(prevButton.name.indexOf("Col") + 3));

  return ((prevButtonCol == currButtonCol && Math.abs(prevButtonRow -
  currButtonRow) <= 1) ||
       (prevButtonRow == currButtonRow && Math.abs(prevButtonCol -
  currButtonCol) <= 1));
}
```

I metodi `charAt()` e `indexOf()` della classe String recuperano le righe e le colonne appropriate rispetto sia al pulsante corrente su cui si fa clic sia a quello su cui è stato fatto clic in precedenza. Il metodo `isLegalContinuation()` restituisce `true` se la riga o la colonna è invariata e se la riga o la colonna che è stata modificata si trova all'interno di un singolo incremento rispetto a quella precedente. Se si desidera cambiare le regole del gioco e consentire ad esempio la sillabazione diagonale, è possibile rimuovere le ricerche di una riga o colonna invariata. La riga finale avrà un aspetto simile al seguente:

```
return (Math.abs(prevButtonRow - currButtonRow) <= 1) &&
  Math.abs(prevButtonCol - currButtonCol) <= 1));
```

Verifica delle parole immesse

Per completare il codice per il gioco sono necessari dei meccanismi per verificare le parole immesse e per calcolare il punteggio. Il metodo `searchForWord()` li contiene entrambi:

```
private function searchForWord(str:String):Number
{
  if (words && str)
  {
    var i:uint = 0
    for (i = 0; i < words.length; i++)
    {
      var thisWord:String = words[i];
      if (str == words[i])
      {
        return i;
      }
    }
    return -1;
  }
  else
  {
    trace("WARNING: cannot find words, or string supplied is null");
  }
  return -1;
}
```

Questa funzione scorre tutte le parole presenti nel dizionario. Se la parola dell'utente corrisponde a una parola presente nel dizionario, ne viene restituita la posizione nel dizionario. Il metodo `submitWord()` verifica a questo punto la risposta e, se la posizione è valida, aggiorna il punteggio.

Personalizzazione

All'inizio della classe sono presenti diverse costanti. Il gioco può essere cambiato modificando queste variabili. Ad esempio, è possibile cambiare il tempo a disposizione aumentando la variabile `TOTAL_TIME`. Oppure, è possibile aumentare leggermente la variabile `PERCENT_VOWELS` per incrementare la probabilità di trovare delle parole.

CAPITOLO 22

Connettività di rete e comunicazioni

22

In questo capitolo viene descritto come attivare il file SWF affinché possa comunicare con i file esterni e altre istanze di Adobe Flash Player 9. Viene inoltre spiegato come caricare dati da origini esterne, inviare messaggi tra un server Java e Flash Player ed eseguire il caricamento e lo scaricamento di file utilizzando le classi FileReference e FileReferenceList.

Sommario

Nozioni fondamentali sulla connettività di rete e le comunicazioni

Introduzione alla connettività di rete e alle comunicazioni

Quando si creano applicazioni ActionScript più complesse, spesso è necessario comunicare con script sul lato server o caricare i dati da file di testo o XML esterni. Il pacchetto flash.net contiene le classi necessarie per inviare e ricevere dati su Internet; ad esempio per caricare il contenuto dagli URL remoti, comunicare con altre istanze di Flash Player e connettersi ai siti Web remoti.

In ActionScript 3.0, è possibile caricare file esterni con le classi URLLoader e URLRequest. Quindi, si utilizza una classe specifica per accedere ai dati, a seconda del tipo di dati che sono stati caricati. Ad esempio, se il contenuto remoto è formattato come coppie nome-valore, utilizzare la classe URLVariables per analizzare i risultati del server. In alternativa, se il file caricato mediante le classi URLLoader e URLRequest è un documento XML remoto, sarà possibile analizzare il documento XML utilizzando la funzione di costruzione della classe XML, la funzione di costruzione della classe XMLDocument o il metodo `XMLDocument.parseXML()`. In questo modo è possibile semplificare il codice ActionScript perché il codice per caricare i file esterni è lo stesso, indipendentemente dal fatto che si utilizzi la classe URLVariables, la classe XML o un'altra classe per analizzare e utilizzare i dati remoti.

Il pacchetto flash.net contiene anche delle classi per altri tipi di comunicazione remota, tra cui la classe FileReference per caricare e scaricare file da un server, le classi Socket e XMLSocket che consentono di comunicare direttamente con i computer remoti mediante connessioni socket e le classi NetConnection eNetStream che vengono utilizzate per comunicare con risorse server specifiche di Flash (ad esempio, Flash Media Server e i server Flash Remoting) e per caricare i file video.

Infine, il pacchetto flash.net include le classi per la comunicazione sul computer locale degli utenti, tra cui la classe LocalConnection, che consente la comunicazione tra due o più file SWF in esecuzione su un solo computer, e la classe SharedObject, che consente di memorizzare i dati sul computer di un utente e recuperarli successivamente quando ritornano all'applicazione.

Operazioni comuni di connettività di rete e comunicazione

L'elenco seguente descrive le operazioni più comuni legate alla comunicazione esterna da ActionScript e descritte in questo capitolo:

- Caricamento di dati da un file esterno o da uno script server
- Invio di dati a uno script server
- Comunicazione con altri file SWF locali
- Operazioni con le connessioni socket binarie
- Comunicazione con i socket XML
- Memorizzazione di dati locali persistenti
- Caricamento di file su un server
- Scaricamento di file da un server nel computer di un utente

Concetti e termini importanti

L'elenco di riferimento seguente contiene dei termini importanti che vengono citati in questo capitolo:

- Dati esterni: dati memorizzati in qualche forma al di fuori del file SWF e caricati nel file SWF quando necessario. Questi dati possono essere caricati direttamente in un file che viene caricato oppure possono essere memorizzati in un database o in un'altra forma che può essere recuperata chiamando script o programmi in esecuzione su un server.
- Variabili con codifica URL: il formato con codifica URL fornisce un metodo per rappresentare diverse variabili (coppie di nomi e valori di variabili) in un'unica stringa di testo. Le singole variabili sono scritte nel formato `nome=valore`. Ogni variabile (cioè ogni coppia nome-variabile) è separata da un carattere di e commerciale, in questo modo: `variabile1=valore1&variabile2=valore2`. Ciò consente di inviare un numero indefinito di variabili sotto forma di un solo messaggio.
- Tipo MIME: un codice standard utilizzato per identificare il tipo di un determinato file nella comunicazione su Internet. Qualunque tipo di file ha un codice specifico che viene utilizzato per identificarlo. Quando si invia un file o un messaggio, un computer (ad esempio, un server Web o l'istanza di Flash Player di un utente) specifica il tipo di file che è in corso di invio.
- HTTP: acronimo di Hypertext Transfer Protocol, un formato standard per la distribuzione di pagine Web e vari altri tipi di contenuti inviati su Internet.
- Metodo di richiesta: quando un programma come Flash Player o un browser Web invia un messaggio (definito anche richiesta HTTP) a un server Web, qualunque dato inviato può essere incorporato nella richiesta utilizzando due *metodi di richiesta* definiti GET e POST. Sul lato server, il programma che riceve la richiesta deve cercare i dati nella porzione appropriata della richiesta, pertanto il metodo di richiesta utilizzato per l'invio dei dati da ActionScript deve corrispondere a quello utilizzato per leggere i dati sul server.
- Connessione socket: una connessione persistente che consente la comunicazione tra due computer.
- Caricamento: l'invio di un file a un altro computer.
- Scaricamento: il recupero di un file da un altro computer.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Molti di questi esempi caricano dati esterni oppure effettuano altri tipi di comunicazioni; spesso includono chiamate alla funzione `trace()`, che fanno in modo che i risultati dell'esecuzione dell'esempio vengano visualizzati nel pannello Output. Altri esempi eseguono effettivamente delle funzioni, come il caricamento di file in un server. La prova di tali esempi comprende l'interazione con il file SWF e la verifica dell'azione che eseguono.

Gli esempi di codice si dividono in due categorie. Alcuni esempi vengono scritti con il presupposto che il codice sia in uno script autonomo, ad esempio associato a un fotogramma chiave in un documento Flash. Per provare questi esempi:

1. Creare un nuovo documento Flash.
2. Selezionare il fotogramma chiave nel fotogramma 1 della linea temporale e aprire il pannello Azioni.
3. Copiare l'esempio di codice nel riquadro dello script.
4. Dal menu principale, selezionare Controllo > Prova filmato per creare un file SWF e provare l'esempio.

Altri esempi di codice vengono scritti come una classe; il risultato previsto è che la classe dell'esempio funga da classe documento per il documento Flash. Per provare questi esempi:

1. Creare un documento Flash vuoto e salvarlo nel computer.
2. Creare un nuovo file ActionScript e salvarlo nella stessa directory del documento Flash. Il nome file deve corrispondere al nome della classe presente nell'esempio di codice. Ad esempio, se il codice definisce una classe chiamata "UploadTest", salvare il file ActionScript con il nome "UploadTest.as".
3. Copiare l'esempio di codice nel file ActionScript e salvare il file.
4. Nel documento Flash, fare clic in una parte vuota dello stage oppure della tavola di montaggio per attivare la finestra di ispezione Proprietà del documento.
5. Nella finestra di ispezione Proprietà, nel campo Classe documento inserire il nome della classe ActionScript copiata dal testo.
6. Eseguire il programma selezionando Controllo > Prova filmato e provare l'esempio.

Infine, alcuni degli esempi nel capitolo prevedono l'interazione con un programma in esecuzione su un server. Questi esempi includono codice che può essere utilizzato per creare il programma server richiesto per la prova; è necessario configurare le applicazioni appropriate su un server Web per provare questi esempi.

Operazioni con i dati esterni

ActionScript 3.0 include dei meccanismi per caricare dati da origini esterne. Queste origini possono essere costituite da contenuti statici come i file di testo o da contenuti dinamici come gli script Web che recuperano i dati da un database. I dati possono essere formattati in molti modi e ActionScript fornisce le funzionalità per decodificare e accedere ai dati. Inoltre, è possibile inviare dati al server esterno come parte del processo di recupero dei dati.

Utilizzo delle classi URLLoader e URLVariables

ActionScript 3.0 utilizza le classi URLLoader e URLVariables per caricare i dati esterni. La classe URLLoader scarica i dati da un URL sotto forma di testo, dati binari o variabili con codifica URL. La classe URLLoader è utile per scaricare file di testo, XML o altre informazioni da utilizzare nelle applicazioni ActionScript dinamiche basate su dati. La classe URLLoader sfrutta il modello di gestione degli eventi avanzato di ActionScript 3.0, che consente l'intercettazione di eventi come `complete`, `httpStatus`, `ioError`, `open`, `progress` e `securityError`. Il nuovo modello di gestione degli eventi costituisce un miglioramento significativo rispetto al supporto di ActionScript 2.0 per i gestori di eventi `LoadVars.onData`, `LoadVars.onHTTPStatus` e `LoadVars.onLoad`, perché consente di gestire gli errori e gli eventi in modo più efficiente. Per ulteriori informazioni sulla gestione degli eventi, vedere Capitolo 10, "Gestione degli eventi".

In modo analogo alle classi XML e LoadVars presenti nelle versioni precedenti di ActionScript, i dati dell'URL di URLLoader non sono disponibili finché lo scaricamento non viene completato. È possibile verificare lo stato di avanzamento dello scaricamento (byte caricati e byte totali) intercettando gli eventi `flash.events.ProgressEvent.PROGRESS` da inviare, benché se un file viene caricato troppo velocemente, l'evento `ProgressEvent.PROGRESS` potrebbe non essere inviato. Dopo che un file è stato scaricato, viene inviato l'evento `flash.events.Event.COMPLETE`. I dati caricati vengono decodificati dalla codifica UTF-8 o UTF-16 in una stringa.

NOTA Se non è impostato alcun valore per `URLRequest.contentType`, i valori vengono inviati come `application/x-www-form-urlencoded`.

Il metodo `URLLoader.load()` (e facoltativamente la funzione di costruzione della classe URLLoader) accetta un solo parametro, `request`, ovvero un'istanza URLRequest. Un'istanza URLRequest contiene tutte le informazioni per una singola richiesta HTTP, ad esempio l'URL di destinazione, il metodo di richiesta (`GET` o `POST`), informazioni aggiuntive sull'intestazione e il tipo MIME (ad esempio, quando si carica il contenuto XML).

Ad esempio, per caricare un pacchetto XML in uno script sul lato server, è possibile utilizzare il codice ActionScript 3.0 riportato di seguito:

```
var secondsUTC:Number = new Date().time;
var dataXML:XML =
  <login>
    <time>{secondsUTC}</time>
    <username>Ernie</username>
    <password>guru</password>
  </login>;
var request:URLRequest = new URLRequest("http://www.yourdomain.com/
  login.cfm");
request.contentType = "text/xml";
request.data = dataXML.toXMLString();
request.method = URLRequestMethod.POST;
var loader:URLLoader = new URLLoader();
try
{
  loader.load(request);
}
catch (error:ArgumentError)
{
  trace("An ArgumentError has occurred.");
}
catch (error:SecurityError)
{
  trace("A SecurityError has occurred.");
}
```

Lo snippet di codice precedente crea un'istanza di XML di nome `dataXML` che contiene un pacchetto XML da inviare al server. Vengono quindi impostate la proprietà `contentType` di URLRequest su `"text/xml"` e la proprietà `data` di URLRequest sul contenuto del pacchetto XML, che viene convertito in una stringa mediante il metodo `XML.toXMLString()`. Infine, viene creata una nuova istanza di URLLoader e viene inviata la richiesta allo script remoto utilizzando il metodo `URLLoader.load()`.

Esistono tre modi per specificare i parametri da passare a una richiesta URL:

- All'interno della funzione di costruzione URLVariables
- All'interno del metodo `URLVariables.decode()`
- Come proprietà specifiche all'interno dell'oggetto stesso URLVariables

Quando si definiscono le variabili all'interno della funzione di costruzione URLVariables o del metodo `URLVariables.decode()`, è necessario assicurarsi di eseguire la codifica URL del carattere e commerciale, in quanto ha un significato particolare e agisce come delimitatore. Ad esempio, quando si passa una e commerciale, è necessario eseguire la codifica URL di questo carattere modificandolo da `&` in `%26`, in quanto la e commerciale agisce come delimitatore per i parametri.

Caricamento di dati da documenti esterni

Quando si creano applicazioni dinamiche con ActionScript 3.0, è consigliabile caricare i dati da file esterni o da script sul lato server. In questo modo è possibile creare applicazioni dinamiche senza modificare o ricompilare i file ActionScript. Ad esempio, se si crea un'applicazione di tipo "suggerimento utile", è possibile scrivere uno script sul lato server che recuperi un suggerimento casuale da un database e lo salvi in un file di testo una volta al giorno. L'applicazione ActionScript può quindi caricare il contenuto di un file di testo statico, anziché interrogare il database ogni volta.

Lo snippet di codice seguente crea un oggetto URLRequest e URLLoader che carica il contenuto di un file di testo esterno, params.txt:

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

È possibile semplificare lo snippet di codice precedente come segue:

```
var loader:URLLoader = new URLLoader(new URLRequest("params.txt"));
```

Per impostazione predefinita, se non si definisce un metodo di richiesta Flash Player carica il contenuto utilizzando il metodo HTTP `GET`. Se si desidera inviare i dati utilizzando il metodo `POST`, impostare la proprietà `request.method` su `POST` utilizzando la costante statica `URLRequestMethod.POST`, come descritto nel codice seguente:

```
var request:URLRequest = new URLRequest("sendfeedback.cfm");
request.method = URLRequestMethod.POST;
```

Il documento esterno, params.txt, che viene caricato in fase di runtime contiene i seguenti dati:

```
monthNames=January,February,March,April,May,June,July,August,September,Octo
  ber,November,December&dayNames=Sunday,Monday,Tuesday,Wednesday,Thursday,
  Friday,Saturday
```

Il file contiene due parametri, `monthNames` e `dayNames`. Ogni parametro contiene un elenco separato da virgole che viene analizzato come stringa. Questo elenco può essere suddiviso in un array utilizzando il metodo `String.split()`.

SUGGERIMENTO	Evitare di utilizzare parole riservate o costrutti del linguaggio come nomi di variabili nei file di dati esterni, perché ciò rende più difficile la lettura e il debug del codice.

Dopo che i dati sono stati caricati, viene inviato l'evento `Event.COMPLETE` e il contenuto del documento esterno diventa disponibile per l'utilizzo nella proprietà `data` di URLLoader, come illustrato nel codice riportato di seguito:

```
private function completeHandler(event:Event):void
{
  var loader2:URLLoader = URLLoader(event.target);
  trace(loader2.data);
}
```

Se il documento remoto contiene coppie nome-valore, è possibile analizzare i dati utilizzando la classe URLVariables passando il contenuto del file caricato, come riportato di seguito:

```
private function completeHandler(event:Event):void
{
  var loader2:URLLoader = URLLoader(event.target);
  var variables:URLVariables = new URLVariables(loader2.data);
  trace(variables.dayNames);
}
```

Ogni coppia nome-valore contenuta nel file esterno viene creata sotto forma di proprietà nell'oggetto URLVariables. Ogni proprietà all'interno dell'oggetto variables nell'esempio di codice precedente viene considerato come una stringa. Se il valore della coppia nome-valore è un elenco di elementi, è possibile convertire la stringa in un array chiamando il metodo `String.split()`, come riportato di seguito:

```
var dayNameArray:Array = variables.dayNames.split(",");
```

SUGGERIMENTO

Se si caricano dati numerici dai file di testo esterni, è necessario convertire i valori in valori numerici utilizzando una funzione di primo livello quale `int()`, `uint()` o `Number()`.

Anziché caricare il contenuto del file remoto come stringa e creare un nuovo oggetto URLVariables, è possibile impostare la proprietà `URLLoader.dataFormat` su una delle proprietà statiche presenti nella classe URLLoaderDataFormat. Di seguito sono riportati i tre valori possibili per la proprietà `URLLoader.dataFormat`:

- `URLLoaderDataFormat.BINARY` — La proprietà `URLLoader.data` conterrà dati binari memorizzati nell'oggetto ByteArray.
- `URLLoaderDataFormat.TEXT` — La proprietà `URLLoader.data` conterrà del testo in un oggetto String.
- `URLLoaderDataFormat.VARIABLES` — La proprietà `URLLoader.data` conterrà le variabili con codifica URL memorizzate in un oggetto URLVariables.

Il codice seguente dimostra come l'impostazione della proprietà `URLLoader.dataFormat` su `URLLoaderDataFormat.VARIABLES` consenta di analizzare automaticamente i dati caricati in un oggetto URLVariables:

```
package
{
  import flash.display.Sprite;
  import flash.events.*;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.net.URLRequest;

  public class URLLoaderDataFormatExample extends Sprite
  {
    public function URLLoaderDataFormatExample()
    {
      var request:URLRequest = new URLRequest("http://
      www.[yourdomain].com/params.txt");
      var variables:URLLoader = new URLLoader();
      variables.dataFormat = URLLoaderDataFormat.VARIABLES;
      variables.addEventListener(Event.COMPLETE, completeHandler);
      try
      {
        variables.load(request);
      }
      catch (error:Error)
      {
        trace("Unable to load URL: " + error);
      }
    }
    private function completeHandler(event:Event):void
    {
        var loader:URLLoader = URLLoader(event.target);
        trace(loader.data.dayNames);
    }
  }
}
```

NOTA Il valore predefinito per `URLLoader.dataFormat` è `URLLoaderDataFormat.TEXT`.

Come illustrato nell'esempio seguente, caricare contenuto XML da un file esterno è come caricare URLVariables. È possibile creare un'istanza di URLRequest e un'istanza di URLLoader e utilizzarle per scaricare un documento XML remoto. Dopo che il file è stato completamente scaricato, viene inviato l'evento `Event.COMPLETE` e il contenuto del documento esterno viene convertito in un'istanza di XML che può essere analizzata utilizzando i metodi e le proprietà XML.

```
package
{
  import flash.display.Sprite;
  import flash.errors.*;
  import flash.events.*;
  import flash.net.URLLoader;
  import flash.net.URLRequest;

  public class ExternalDocs extends Sprite
  {
    public function ExternalDocs()
    {
      var request:URLRequest = new URLRequest("http://
      www.[yourdomain].com/data.xml");
      var loader:URLLoader = new URLLoader();
      loader.addEventListener(Event.COMPLETE, completeHandler);
      try
      {
        loader.load(request);
      }
      catch (error:ArgumentError)
      {
        trace("An ArgumentError has occurred.");
      }
      catch (error:SecurityError)
      {
        trace("A SecurityError has occurred.");
      }
    }
    private function completeHandler(event:Event):void
    {
      var dataXML:XML = XML(event.target.data);
      trace(dataXML.toXMLString());
    }
  }
}
```

Comunicazione con script esterni

Oltre a caricare file di dati esterni, è possibile utilizzare la classe URLVariables per inviare variabili a uno script sul lato server ed elaborare la risposta del server. Questa operazione è utile, ad esempio, quando si programma un gioco e si desidera inviare il punteggio dell'utente a un server per calcolare se dovrà essere aggiunto all'elenco dei punteggi più alti oppure per inviare le informazioni di login di un utente a un server per la convalida. Uno script sul lato server può elaborare il nome utente e la password, convalidarli rispetto a un database e confermare se le credenziali fornite dall'utente sono valide.

Lo snippet di codice seguente crea un oggetto URLVariables di nome `variables`, che a propria volta crea una nuova variabile denominata `name`. Viene quindi creato un oggetto URLRequest che specifica l'URL dello script sul lato server a cui inviare le variabili. Si imposta quindi la proprietà `method` dell'oggetto URLRequest in modo che le variabili siano inviate come una richiesta `POST` HTTP. Per aggiungere l'oggetto URLVariables alla richiesta URL, impostare la proprietà `data` dell'oggetto URLRequest sull'oggetto URLVariables creato precedentemente. Infine, viene creata l'istanza di URLLoader e viene richiamato il metodo `URLLoader.load()`, che avvia la richiesta.

```
var variables:URLVariables = new URLVariables("name=Franklin");
var request:URLRequest = new URLRequest();
request.url = "http://www.[yourdomain].com/greeting.cfm";
request.method = URLRequestMethod.POST;
request.data = variables;
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.VARIABLES;
loader.addEventListener(Event.COMPLETE, completeHandler);
try
{
  loader.load(request);
}
catch (error:Error)
{
  trace("Unable to load URL");
}

function completeHandler(event:Event):void
{
  trace(event.target.data.welcomeMessage);
}
```

Il codice riportato di seguito include il contenuto del documento greeting.cfm di Adobe ColdFusion® utilizzato nell'esempio precedente:

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
  <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

Connessione ad altre istanze di Flash Player

La classe LocalConnection consente la comunicazione tra diverse istanze di Flash Player, come un file SWF in un contenitore HTML o in un lettore incorporato o autonomo. In questo modo è possibile creare applicazioni molto versatili, in grado di condividere i dati tra le istanze di Flash Player, ad esempio i file SWF in esecuzione in un browser Web o incorporati nelle applicazioni desktop.

Classe LocalConnection

La classe LocalConnection consente di sviluppare file SWF in grado di inviare istruzioni ad altri file SWF senza utilizzare il metodo `fscommand()` o JavaScript. Gli oggetti LocalConnection possono comunicare solo tra file SWF in esecuzione sullo stesso computer client, tuttavia possono essere eseguiti in applicazioni diverse. Ad esempio, un file SWF in esecuzione in un browser e un file SWF in esecuzione in un proiettore possono condividere le informazioni: il proiettore gestisce le informazioni locali, mentre il file SWF basato su browser si connette in remoto. Un proiettore è un file SWF salvato in un formato che può essere eseguito come applicazione autonoma, ovvero senza richiedere l'installazione di Flash Player essendo incorporato nell'eseguibile.

Gli oggetti LocalConnection possono essere utilizzati per la comunicazione tra file SWF che utilizzano versioni diverse di ActionScript:

- Gli oggetti LocalConnection di ActionScript 3.0 sono in grado di comunicare con gli oggetti LocalConnection creati in ActionScript 1.0 e 2.0.
- Gli oggetti LocalConnection di ActionScript 1.0 e 2.0 sono in grado di comunicare con gli oggetti LocalConnection creati in ActionScript 3.0.

Flash Player gestisce in modo automatico questa comunicazione tra oggetti LocalConnection di versioni diverse.

Il modo più semplice per utilizzare un oggetto LocalConnection consiste nel consentire la comunicazione solo tra gli oggetti LocalConnection di uno stesso dominio. In questo modo, si evitano problemi legati alla sicurezza. Tuttavia, se è necessario consentire la comunicazione tra domini diversi, esistono diversi modi per implementare delle misure di sicurezza. Per ulteriori informazioni, vedere la discussione sul parametro `connectionName` del metodo `send()` e le sezioni relative ad `allowDomain()` e `domain` nella sezione relativa alla classe LocalConnection nella *Guida di riferimento al linguaggio e ai componenti di ActionScript 3.0*.

SUGGERIMENTO	È possibile utilizzare gli oggetti LocalConnection per inviare e ricevere dati in un solo file SWF; Adobe consiglia tuttavia di non utilizzare questo metodo, bensì di utilizzare gli oggetti condivisi.

Sono disponibili tre modi per aggiungere metodi di callback agli oggetti LocalConnection:

- Creare una sottoclasse della classe LocalConnection e aggiungere dei metodi.
- Impostare la proprietà `LocalConnection.client` su un oggetto che implementi i metodi.
- Creare una classe dinamica per estendere LocalConnection e associare in modo dinamico i metodi.

Il primo modo per aggiungere i metodi di callback consiste nell'estendere la classe LocalConnection. Definire i metodi all'interno della classe personalizzata anziché aggiungerli in modo dinamico all'istanza di LocalConnection. Questo approccio viene illustrato nel codice riportato di seguito:

```
package
{
  import flash.net.LocalConnection;
  public class CustomLocalConnection extends LocalConnection
  {
    public function CustomLocalConnection(connectionName:String)
    {
      try
      {
        connect(connectionName);
      }
      catch (error:ArgumentError)
      {
        // Server già creato/connesso
      }
    }
```

```
    public function onMethod(timeString:String):void
    {
      trace("onMethod called at: " + timeString);
    }
  }
}
```

Per creare una nuova istanza della classe DynamicLocalConnection, è possibile utilizzare il codice riportato di seguito:

```
var serverLC:CustomLocalConnection;
serverLC = new CustomLocalConnection("serverName");
```

Il secondo modo per aggiungere i metodi di callback consiste nell'utilizzare la proprietà `LocalConnection.client`. In questa fase occorre creare una classe personalizzata e assegnare una nuova istanza alla proprietà `client` come illustrato nel codice riportato di seguito:

```
var lc:LocalConnection = new LocalConnection();
lc.client = new CustomClient();
```

La proprietà `LocalConnection.client` indica i metodi di callback dell'oggetto da richiamare. Nel codice precedente la proprietà `client` è stata impostata su una nuova istanza della classe personalizzata CustomClient. Il valore predefinito per la proprietà `client` è l'istanza di LocalConnection corrente. È possibile utilizzare la proprietà `client` se si dispone di due gestori di dati con lo stesso set di metodi ma con un funzionamento diverso; ad esempio, in un'applicazione dove un pulsante in una finestra attiva/disattiva la vista in una seconda finestra.

Per creare la classe CustomClient, è possibile utilizzare il seguente codice:

```
package
{
  public class CustomClient extends Object
  {
    public function onMethod(timeString:String):void
    {
      trace("onMethod called at: " + timeString);
    }
  }
}
```

Il terzo modo per aggiungere i metodi di callback, creando una classe dinamica e associando in modo dinamico i metodi, è molto simile all'utilizzo della classe LocalConnection nelle versioni precedenti di ActionScript, come illustrato nel codice riportato di seguito:

```
import flash.net.LocalConnection;
dynamic class DynamicLocalConnection extends LocalConnection {}
```

I metodi di callback possono essere aggiunti in modo dinamico a questa classe utilizzando il codice riportato di seguito:

```
var connection:DynamicLocalConnection = new DynamicLocalConnection();
connection.onMethod = this.onMethod;
// Aggiungere qui il codice.
public function onMethod(timeString:String):void
{
  trace("onMethod called at: " + timeString);
}
```

Il precedente modo di aggiungere i metodi di callback non è consigliato, perché il codice non è molto portabile. Inoltre, l'utilizzo di questo metodo per la creazione delle connessioni locali potrebbe causare problemi a livello di prestazioni perché l'accesso alle proprietà dinamiche è significativamente più lento dell'accesso alle proprietà chiuse.

Invio di messaggi tra due istanze di Flash Player

Utilizzare la classe LocalConnection per comunicare tra diverse istanze di Flash Player. Ad esempio, in una pagina Web potrebbero essere presenti più istanze di Flash Player oppure potrebbe essere presente un'istanza di Flash Player che recupera i dati da un'istanza di Flash Player in una finestra a comparsa.

Il codice riportato di seguito definisce un oggetto di connessione locale che agisce come un server e accetta le chiamate in arrivo da altre istanze di Flash Player:

```
package
{
  import flash.net.LocalConnection;
  import flash.display.Sprite;
  public class ServerLC extends Sprite
  {
    public function ServerLC()
    {
      var lc:LocalConnection = new LocalConnection();
      lc.client = new CustomClient1();
      try
      {
        lc.connect("conn1");
      }
      catch (error:Error)
      {
        trace("error:: already connected");
      }
    }
  }
}
```

Questo codice crea dapprima un oggetto LocalConnection di nome `lc` e imposta la proprietà `client` sulla classe personalizzata, CustomClient1. Quando un'altra istanza di Flash Player chiama un metodo in questa istanza della connessione locale, il metodo viene cercato nella classe CustomClient1.

Ogni volta che un'istanza di Flash Player si connette a questo file SWF e tenta di richiamare un metodo qualsiasi per la connessione locale specificata, la richiesta viene inviata alla classe specificata dalla proprietà `client`, che è impostata sulla classe CustomClient1:

```
package
{
  import flash.events.*;
  import flash.system.fscommand;
  import flash.utils.Timer;
  public class CustomClient1 extends Object
  {
    public function doMessage(value:String = ""):void
    {
      trace(value);
    }
    public function doQuit():void
    {
      trace("quitting in 5 seconds");
      this.close();
      var quitTimer:Timer = new Timer(5000, 1);
      quitTimer.addEventListener(TimerEvent.TIMER, closeHandler);
    }
    public function closeHandler(event:TimerEvent):void
    {
      fscommand("quit");
    }
  }
}
```

Per creare un server LocalConnection, chiamare il metodo `LocalConnection.connect()` e specificare un nome di connessione univoco. Se è già presente una connessione con il nome specificato, viene generato un errore ArgumentError, in cui è indicato che il tentativo di connessione non è riuscito in quanto l'oggetto è già connesso.

Lo snippet di codice seguente illustra come creare una nuova connessione socket con il nome `conn1`:

```
try
{
  connection.connect("conn1");
}
catch (error:ArgumentError)
{
  trace("Error! Server already exists\n");
}
```

NOTA

Nelle versioni precedenti di ActionScript, il metodo `LocalConnection.connect()` restituiva un valore booleano se il nome della connessione era già stato utilizzato. In ActionScript 3.0, se il nome è già stato utilizzato viene generato un errore.

La connessione al file SWF primario da un file SWF secondario richiede la creazione di un nuovo oggetto LocalConnection nell'oggetto LocalConnection mittente e quindi l'esecuzione di una chiamata al metodo `LocalConnection.send()` con il nome della connessione e il nome del metodo da eseguire. Ad esempio, per connettere l'oggetto LocalConnection creato in precedenza, utilizzare il codice riportato di seguito:

```
sendingConnection.send("conn1", "doQuit");
```

Questo codice stabilisce la connessione a un oggetto LocalConnection esistente con il nome di connessione `conn1` e richiama il metodo `doQuit()` nel file SWF remoto. Se si desidera inviare dei parametri al file SWF remoto, specificare gli argomenti aggiuntivi dopo il nome del metodo nel metodo `send()`, come illustrato nello snippet di codice seguente:

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

Connessione ai documenti SWF in domini diversi

Per consentire le comunicazioni solo da domini specifici, chiamare il metodo `allowDomain()` o `allowInsecureDomain()` della classe LocalConnection e passare un elenco di uno o più domini a cui è consentito accedere a questo oggetto LocalConnection.

Nelle versioni precedenti di ActionScript, `LocalConnection.allowDomain()` e `LocalConnection.allowInsecureDomain()` erano metodi di callback che dovevano essere implementati dagli sviluppatori e che dovevano restituire un valore booleano. In ActionScript 3.0, `LocalConnection.allowDomain()` e `LocalConnection.allowInsecureDomain()` sono metodi incorporati che gli sviluppatori possono chiamare esattamente come `Security.allowDomain()` e `Security.allowInsecureDomain()`, passando uno o più nomi di domini a cui consentire l'accesso.

Vi sono due valori speciali che è possibile passare ai metodi `LocalConnection.allowDomain()` e `LocalConnection.allowInsecureDomain()`: `*` e `localhost`. Il valore asterisco `(*)` consente l'accesso da tutti i domini. La stringa `localhost` consente le chiamate al file SWF da tutti i file SWF installati a livello locale.

In Flash Player 8 sono state introdotte delle limitazioni di sicurezza per i file SWF locali. Un file SWF autorizzato ad accedere a Internet non può avere accesso anche al file system locale. Se si specifica `localhost`, qualunque file SWF locale può accedere al file SWF. Se il metodo `LocalConnection.send()` tenta di comunicare con un file SWF da una funzione di sicurezza sandbox a cui il codice chiamante non ha accesso, viene inviato un evento `securityError` (`SecurityErrorEvent.SECURITY_ERROR`). Per risolvere questo errore, è possibile specificare il dominio del chiamante nel metodo `LocalConnection.allowDomain()` del ricevente.

Se si implementa la comunicazione solo tra i file SWF dello stesso dominio, è possibile specificare un parametro `connectionName` che non inizi con un carattere di sottolineatura (_) e che non specifichi un nome di dominio (ad esempio, `myDomain:connectionName`). Utilizzare la stessa stringa nel comando `LocalConnection.connect(connectionName)`.

Se si implementa la comunicazione tra i file SWF di domini diversi, specificare un parametro `connectionName` che inizi con un carattere di sottolineatura. L'utilizzo del carattere di sottolineatura rende il file SWF con l'oggetto LocalConnection ricevente più portabile tra i domini. Di seguito sono illustrati i due casi possibili:

- Se la stringa per `connectionName` non inizia con un carattere di sottolineatura (_), viene aggiunto un prefisso composto dal nome del superdominio e da un segno di due punti (ad esempio, `"myDomain:connectionName"`). Benché questo assicuri che la connessione non entri in conflitto con le connessioni di altri domini che hanno lo stesso nome, tutti gli oggetti LocalConnection mittenti devono specificare questo superdominio (ad esempio, `myDomain:connectionName`). Se il file SWF con l'oggetto LocalConnection ricevente viene spostato in un altro dominio, Flash Player modifica il prefisso per rispecchiare il nuovo superdominio (ad esempio, `anotherDomain:connectionName`). Tutti gli oggetti LocalConnection mittenti devono quindi essere modificati manualmente in modo da puntare al nuovo superdominio.
- Se la stringa per `connectionName` inizia con un carattere di sottolineatura (ad esempio, `_connectionName`), non viene aggiunto alcun prefisso alla stringa. Questo significa che gli oggetti LocalConnection riceventi e mittenti utilizzeranno stringhe identiche per `connectionName`. Se l'oggetto ricevente utilizza `LocalConnection.allowDomain()` per specificare che vengano accettate le connessioni da qualunque dominio, il file SWF con l'oggetto LocalConnection ricevente può essere spostato in un altro dominio senza modificare alcun oggetto LocalConnection mittente.

Connessioni socket

In ActionScript 3.0 sono disponibili due diversi tipi di connessioni socket: connessioni socket XML e connessioni socket binarie. Un socket XML consente di connettersi a un server remoto e di creare una connessione al server che rimane aperta finché non viene chiusa in modo esplicito. In questo modo è possibile scambiare dati XML tra un server e un client senza dover aprire continuamente nuove connessioni al server. Un ulteriore vantaggio legato all'uso di un server socket XML è rappresentato dal fatto che l'utente non deve richiedere esplicitamente i dati. È possibile inviare dati dal server senza richieste e a tutti i client connessi al server socket XML.

Una connessione socket binaria è simile a un socket XML, ma il client e il server non devono necessariamente scambiare pacchetti XML specifici. Al contrario, sulla connessione possono essere trasferiti dati come informazioni binarie. In questo modo è possibile connettersi a una vasta gamma di servizi, compresi i server di posta (POP3, SMTP e IMAP) e i server delle news (NNTP).

Classe Socket

Introdotta in ActionScript 3.0, la classe Socket consente di effettuare connessioni socket e di leggere e scrivere dati binari originari da ActionScript. È simile alla classe XMLSocket ma non impone il formato dei dati ricevuti e trasmessi. La classe Socket è utile per interagire con i server che utilizzano protocolli binari. Utilizzando le connessioni socket binarie è possibile scrivere codice che consenta l'interazione con più protocolli Internet diversi, ad esempio POP3, SMTP, IMAP e NNTP. Ciò consente a Flash Player di connettersi a propria volta ai server di posta e delle news.

Flash Player è in grado di interfacciarsi con un server utilizzando direttamente il protocollo binario di quel server. Alcuni server utilizzano l'ordine dei byte bigEndian e altri littleEndian. La maggior parte dei server in Internet utilizza l'ordine dei byte bigEndian perché questo è "l'ordine dei byte di rete". L'ordine dei byte littleEndian è conosciuto in quanto utilizzato dall'architettura Intel® x86. È necessario utilizzare l'ordine dei byte Endian corrispondente a quello del server che invia o riceve i dati. Tutte le operazioni eseguite dalle interfacce IDataInput e IDataOutput e le classi che implementano tali interfacce (ByteArray, Socket e URLStream) sono codificate per impostazione predefinita nel formato bigEndian, ovvero con il byte più significativo all'inizio. Ciò avviene ai fini della corrispondenza dell'ordine dei byte di rete ufficiale e Java. Per scegliere di utilizzare l'ordine dei byte bigEndian o littleEndian, è possibile impostare la proprietà `endian` su `Endian.BIG_ENDIAN` o `Endian.LITTLE_ENDIAN`.

SUGGERIMENTO

La classe Socket eredita tutti i metodi implementati dalle interfacce IDataInput e IDataOutput (incluse nel pacchetto flash.utils) e per scrivere e leggere da Socket dovranno essere utilizzati questi metodi.

Classe XMLSocket

ActionScript fornisce una classe XMLSocket incorporata che consente di stabilire una connessione continua con un server. La connessione aperta evita problemi di latenza e viene in genere utilizzata per applicazioni in tempo reale, ad esempio per le applicazioni chat o i giochi in modalità multiplayer. Una normale soluzione chat basata su HTTP esegue il polling del server e scarica i nuovi messaggi mediante una richiesta HTTP. Al contrario, una soluzione chat basata su XMLSocket mantiene una connessione aperta con il server, che consente a quest'ultimo di inviare immediatamente i messaggi in entrata senza che il client emetta una richiesta.

Per creare una connessione socket, è necessario creare un'applicazione sul lato server che attenda la richiesta della connessione socket e invii la risposta al file SWF. Questo tipo di applicazione sul lato server può essere scritta in un linguaggio di programmazione come Java, Python o Perl. Per utilizzare la classe XMLSocket, sul server deve essere in esecuzione un daemon in grado interpretare il protocollo utilizzato dalla classe XMLSocket. Il protocollo viene descritto nell'elenco seguente:

- I messaggi XML vengono inviati su una connessione socket di streaming TCP/IP full-duplex.
- Ogni messaggio XML è un documento XML completo, terminato da un byte zero (0).

- Su una singola connessione XMLSocket può essere inviato e ricevuto un numero illimitato di messaggi XML.

NOTA	La classe XMLSocket non può eseguire il tunneling automatico attraverso i firewall perché, a differenza del protocollo RTMP (Real-Time Messaging Protocol), XMLSocket non dispone di capacità di tunneling HTTP. Se è necessario il tunneling HTTP, valutare la possibilità di utilizzare Flash Remoting o Flash Media Server (che supporta RTMP).

Le seguenti limitazioni riguardano le modalità di connessione al server da parte di un oggetto XMLSocket:

- Il metodo `XMLSocket.connect()` può effettuare una connessione solo alle porte con numeri superiori o uguali a 1024. Una conseguenza di questa limitazione consiste nel fatto che anche i daemon dei server che comunicano con l'oggetto `XMLSocket` devono essere assegnati a numeri di porta superiori o uguali a 1024. I numeri di porta inferiori a 1024 vengono spesso utilizzati dai servizi di sistema come FTP (21), Telnet (23), SMTP (25), HTTP (80) e POP3 (110) e pertanto gli oggetti XMLSocket sono esclusi da queste porte per motivi di sicurezza. La restrizione dei numeri di porta riduce le possibilità di accesso inappropriato e abuso di queste risorse.
- Il metodo `XMLSocket.connect()` può connettersi solo ai computer dello stesso dominio su cui risiede il file SWF. Questa restrizione non vale per i file SWF in esecuzione su un disco locale. (Questa restrizione è identica alle regole di sicurezza per `URLLoader.load()`). Per connettersi a un daemon di un server in esecuzione su un dominio diverso da quello su cui risiede il file SWF, è possibile creare un file di criteri di sicurezza sul server che consenta l'accesso da alcuni domini specifici.

NOTA	La procedura di configurazione di un server per la comunicazione con l'oggetto XMLSocket può rivelarsi un'operazione complessa. Se l'applicazione in uso non richiede l'interattività in tempo reale, utilizzare la classe URLLoader anziché la classe XMLSocket.

Per trasferire il codice XML verso e dal server su una connessione socket, è possibile utilizzare i metodi `XMLSocket.connect()` e `XMLSocket.send()` della classe XMLSocket. Il metodo `XMLSocket.connect()` stabilisce una connessione socket con una porta di un server Web. Il metodo `XMLSocket.send()` passa un oggetto XML al server specificato nella connessione socket.

Quando si richiama il metodo `XMLSocket.connect()`, Flash Player apre una connessione TCP/IP al server e la mantiene aperta fino al verificarsi di uno dei seguenti eventi:

- Viene chiamato il metodo `XMLSocket.close()` della classe XMLSocket.
- Non sono più presenti riferimenti all'oggetto XMLSocket.
- Flash Player viene chiuso.
- La connessione viene interrotta, ad esempio il modem si disconnette.

Creazione e connessione a un server socket XML Java

Il codice riportato di seguito dimostra un semplice server XMLSocket scritto in Java che accetta le connessioni in entrata e visualizza i messaggi ricevuti nella finestra del prompt dei comandi. Per impostazione predefinita, viene creato un nuovo server sulla porta 8080 del computer locale, anche se è possibile specificare un numero di porta diverso quando si avvia il server dalla riga di comando.

Creare un nuovo documento di testo e aggiungere il codice riportato di seguito:

```
import java.io.*;
import java.net.*;

class SimpleServer
{
  private static SimpleServer server;
  ServerSocket socket;
  Socket incoming;
  BufferedReader readerIn;
  PrintStream printOut;

  public static void main(String[] args)
  {
    int port = 8080;

    try
    {
      port = Integer.parseInt(args[0]);
    }
    catch (ArrayIndexOutOfBoundsException e)
    {
      // Rileva l'eccezione e continua.
    }

    server = new SimpleServer(port);
  }

  private SimpleServer(int port)
  {
    System.out.println(">> Starting SimpleServer");
    try
    {
      socket = new ServerSocket(port);
      incoming = socket.accept();
      readerIn = new BufferedReader(new
      InputStreamReader(incoming.getInputStream()));
      printOut = new PrintStream(incoming.getOutputStream());
      printOut.println("Enter EXIT to exit.\r");
```

```
        out("Enter EXIT to exit.\r");
        boolean done = false;
        while (!done)
        {
          String str = readerIn.readLine();
          if (str == null)
          {
            done = true;
          }
          else
          {
            out("Echo: " + str + "\r");
            if(str.trim().equals("EXIT"))
            {
              done = true;
            }
          }
          incoming.close();
        }
      }
      catch (Exception e)
      {
        System.out.println(e);
      }
    }

    private void out(String str)
    {
      printOut.println(str);
      System.out.println(str);
    }
}
```

Salvare il documento sul disco rigido come SimpleServer.java e compilarlo con un compilatore Java, che crea un file di classe Java di nome SimpleServer.class.

È possibile avviare il server XMLSocket aprendo un prompt dei comandi e digitando `java SimpleServer`. Il file SimpleServer.class può trovarsi in qualsiasi percorso del computer locale o della rete; non è necessario collocarlo nella directory principale del server Web.

SUGGERIMENTO

Se non è possibile avviare il server perché i file non si trovano nel percorso di classe Java, provare ad avviarlo con `java -classpath . SimpleServer`.

Per connettersi a XMLSocket dall'applicazione ActionScript, è necessario creare una nuova istanza della classe XMLSocket e chiamare il metodo `XMLSocket.connect()` mentre vengono passati un nome host e un numero di porta, come riportato di seguito:

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

Si verifica un evento `securityError` (flash.events.SecurityErrorEvent) se una chiamata a `XMLSocket.connect()` tenta di connettersi a un server che si trova al di fuori della funzione di sicurezza sandbox del chiamante o a una porta con numero inferiore a 1024.

Quando si ricevono dati dal server, viene inviato l'evento `data` (`flash.events.DataEvent.DATA`):

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
  trace("[" + event.type + "] " + event.data);
}
```

Per inviare dati al server XMLSocket, utilizzare il metodo `XMLSocket.send()` e passare una stringa o un oggetto XML. Flash Player converte il parametro fornito in un oggetto String e invia il contenuto al server XMLSocket seguito da un byte zero (0):

```
xmlsock.send(xmlFormattedData);
```

Il metodo `XMLSocket.send()` non restituisce un valore che indica se i dati sono stati trasmessi correttamente. Se si è verificato un errore mentre si tenta di inviare i dati, viene generato un errore IOError.

SUGGERIMENTO	Ogni messaggio inviato al server socket XML deve essere terminato da un carattere newline (`\n`).

Memorizzazione di dati locali

Un oggetto condiviso, a volte definito come “cookie Flash”, è un file di dati che può essere creato sul computer locale dai siti visitati. Gli oggetti condivisi vengono comunemente utilizzati per migliorare la navigazione su Internet, ad esempio consentendo la personalizzazione dell’interfaccia di un sito Web visitato di frequente. Gli oggetti condivisi non sono di per sé in grado di effettuare operazioni o utilizzare i dati presenti nel computer. È importante sottolineare, inoltre, che gli oggetti condivisi non possono mai accedere o ricordare l’indirizzo di posta elettronica o altre informazioni personali dell’utente, a meno che tali informazioni non vengano fornite volontariamente.

È possibile creare nuove istanze di un oggetto condiviso utilizzando il metodo statico `SharedObject.getLocal()` o `SharedObject.getRemote()`. Il metodo `getLocal()` tenta di caricare un oggetto condiviso persistente a livello locale disponibile solo per il client corrente, mentre il metodo `getRemote()` tenta di caricare un oggetto condiviso remoto che può essere condiviso tra più client mediante un server, ad esempio Flash Media Server. Se l’oggetto condiviso locale o remoto non esiste, i metodi `getLocal()` e `getRemote()` creano una nuova istanza di SharedObject.

Il codice riportato di seguito tenta di caricare un oggetto condiviso locale di nome `test`. Se questo oggetto condiviso non esiste, viene creato un nuovo oggetto condiviso con lo stesso nome.

```
var so:SharedObject = SharedObject.getLocal("test");
trace("SharedObject is " + so.size + " bytes");
```

Se non è possibile trovare un oggetto condiviso di nome test, ne viene creato uno nuovo con dimensioni di 0 byte. Se l’oggetto condiviso esisteva in precedenza, vengono restituite le dimensioni correnti (espresse in byte).

È possibile memorizzare dati in un oggetto condiviso assegnando dei valori all’oggetto dati, come illustrato nell’esempio riportato di seguito:

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.now = new Date().time;
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

Se è già presente un oggetto condiviso con il nome `test` e il parametro `now`, il valore esistente viene sovrascritto. È possibile utilizzare la proprietà `SharedObject.size` per determinare se un oggetto condiviso esiste già, come illustrato nell'esempio riportato di seguito:

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
  // L'oggetto condiviso non esiste.
  trace("created...");
  so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

Il codice sopra riportato utilizza il parametro `size` per determinare se l'istanza dell'oggetto condiviso con il nome specificato esiste già. Se si prova il codice riportato di seguito, si noterà che l'oggetto condiviso viene ricreato ogni volta che si esegue il codice. Per salvare un oggetto condiviso sul disco rigido dell'utente, è necessario chiamare in modo esplicito il metodo `SharedObject.flush()` come illustrato nell'esempio riportato di seguito:

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
  // L'oggetto condiviso non esiste.
  trace("created...");
  so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

Quando si utilizza il metodo `flush()` per scrivere gli oggetti condivisi sul disco rigido dell'utente, verificare con attenzione se l'utente ha disattivato in modo esplicito la memorizzazione locale tramite Gestione impostazioni di Flash Player (www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager07.html), come illustrato nell'esempio riportato di seguito:

```
var so:SharedObject = SharedObject.getLocal("test");
trace("Current SharedObject size is " + so.size + " bytes.");
so.flush();
```

È possibile recuperare i valori da un oggetto condiviso specificando il nome della proprietà nella proprietà `data` dell'oggetto condiviso. Ad esempio, se si esegue il codice riportato di seguito, verrà indicato da quanti minuti è stata creata l'istanza di SharedObject:

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
  // L'oggetto condiviso non esiste.
  trace("created...");
  so.data.now = new Date().time;
}
var ageMS:Number = new Date().time - so.data.now;
trace("SharedObject was created " + Number(ageMS / 1000 /
  60).toPrecision(2) + " minutes ago");
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

La prima volta che si esegue il codice sopra riportato, viene creata una nuova istanza di SharedObject di nome `test` con dimensioni iniziali di 0 byte. Poiché le dimensioni iniziali sono pari a 0 byte, l'istruzione `if` restituisce `true` e viene aggiunta una nuova proprietà di nome `now` all'oggetto condiviso locale. La durata dell'oggetto condiviso viene calcolata sottraendo il valore della proprietà `now` dall'ora corrente. A ogni successiva esecuzione del codice sopra riportato, le dimensioni dell'oggetto condiviso dovrebbero essere maggiori di 0 e il codice registrerà da quanti minuti è stato creato tale oggetto.

Visualizzazione del contenuto di un oggetto condiviso

Nella proprietà `data` degli oggetti condivisi vengono memorizzati dei valori. È possibile eseguire un'elaborazione ciclica di ciascun valore contenuto nell'istanza di un oggetto condiviso utilizzando un ciclo `for..in`, come illustrato nell'esempio riportato di seguito:

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.hello = "world";
so.data.foo = "bar";
so.data.timezone = new Date().timezoneOffset;
for (var i:String in so.data)
{
  trace(i + ":\t" + so.data[i]);
}
```

Creazione di un oggetto SharedObject protetto

Quando si crea un oggetto SharedObject locale o remoto utilizzando `getLocal()` o `getRemote()`, è disponibile un parametro opzionale di nome `secure` che determina se l'accesso a questo oggetto condiviso è limitato ai file SWF inviati attraverso una connessione HTTPS. Se questo parametro viene impostato su `true` e il file SWF viene inviato tramite HTTPS, Flash Player crea un nuovo oggetto condiviso protetto oppure acquisisce un riferimento a un oggetto condiviso protetto esistente. Questo oggetto condiviso protetto può essere letto o scritto solo da/su file SWF distribuiti tramite HTTPS che chiamano `SharedObject.getLocal()` con il parametro secure impostato su `true`. Se questo parametro viene impostato su `false` e il file SWF viene inviato tramite HTTPS, Flash Player crea un nuovo oggetto condiviso protetto oppure acquisisce un riferimento a un oggetto condiviso esistente.

Questo oggetto condiviso può essere letto o scritto da/su file SWF distribuiti tramite connessioni non HTTPS. Se il file SWF viene distribuito tramite una connessione non HTTPS e si tenta di impostare il parametro su `true`, la creazione del nuovo oggetto condiviso (oppure l'accesso a un oggetto condiviso protetto esistente) non riesce, viene generato un errore e l'oggetto condiviso viene impostato su `null`. Se si tenta di eseguire lo snippet di codice seguente da una connessione non HTTPS, il metodo `SharedObject.getLocal()` genera un errore:

```
try
{
  var so:SharedObject = SharedObject.getLocal("contactManager", null,
  true);
}
catch (error:Error)
{
  trace("Unable to create SharedObject.");
}
```

Indipendentemente dal valore di questo parametro, gli oggetti condivisi creati vengono presi in considerazione per il conteggio dello spazio totale su disco concesso al dominio.

Operazioni di caricamento e scaricamento dei file

Tramite la classe FileReference è possibile aggiungere la capacità di caricare e scaricare file tra client e server. Agli utenti viene richiesto di selezionare un file da caricare, o un percorso in cui scaricarlo, tramite una finestra di dialogo, ad esempio Apri nel sistema operativo Windows.

Ogni oggetto FileReference che si crea con ActionScript fa riferimento a un singolo file sul disco rigido dell'utente. L'oggetto dispone di proprietà che contengono informazioni su dimensione, tipo, nome, data di creazione e data di modifica del file.

NOTA La proprietà `creator` è supportata solo in Mac OS. Tutte le altre piattaforme restituiscono `null`.

Un'istanza della classe FileReference può essere creata in due modi. È possibile utilizzare l'operatore `new` come illustrato nel codice riportato di seguito:

```
import flash.net.FileReference;
var myFileReference:FileReference = new FileReference();
```

In alternativa, è possibile chiamare il metodo `FileReferenceList.browse()` che apre una finestra di dialogo nel sistema dell'utente, nella quale viene richiesto di selezionare uno o più file da caricare, quindi crea un array di oggetti FileReference se l'utente riesce a selezionare uno o più file. Ogni oggetto FileReference rappresenta un file selezionato dall'utente nella finestra di dialogo. Nelle proprietà FileReference (ad esempio `name`, `size` o `modificationDate`) di un oggetto FileReference non sono contenuti dati, finché non si verifica una delle seguenti condizioni:

- È stato chiamato il metodo `FileReference.browse()` o `FileReferenceList.browse()` e l'utente ha selezionato un file dal selettore.
- È stato chiamato il metodo `FileReference.download()` e l'utente ha selezionato un file dal selettore.

NOTA Quando si esegue uno scaricamento, viene compilata solo la proprietà `FileReference.name` prima che l'operazione sia stata completata. Dopo lo scaricamento del file, saranno disponibili tutte le proprietà.

Mentre le chiamate al metodo `FileReference.browse()`, `FileReferenceList.browse()` o `FileReference.download()` sono in esecuzione, la maggior parte dei lettori continua la riproduzione del file SWF.

Classe FileReference

La classe FileReference consente di caricare e scaricare i file tra il computer di un utente e un server. Una finestra di dialogo del sistema operativo richiede all'utente di selezionare un file da caricare o un percorso per lo scaricamento. Ogni oggetto FileReference fa riferimento a un singolo file sul disco dell'utente e ha delle proprietà che contengono informazioni relative a dimensioni, tipo, nome, data di creazione, data di modifica e generatore del file.

Le istanze di FileReference possono essere create in due modi:

- Quando si utilizza l'operatore `new` con la funzione di costruzione FileReference, come nell'esempio riportato di seguito:

  ```
  var myFileReference:FileReference = new FileReference();
  ```

- Quando si chiama `FileReferenceList.browse()`, che crea un array di oggetti FileReference.

Per le operazioni di caricamento e scaricamento, un file SWF può accedere ai file solo dal proprio dominio, compresi gli eventuali domini specificati da un file dei criteri di accesso a più domini. È necessario collocare un file di criteri sul file server nel caso il file SWF che inizializza il caricamento o scaricamento non abbia lo stesso dominio del file server.

NOTA È possibile eseguire una sola azione `browse()` o `download()` alla volta, in quanto è possibile aprire una sola finestra di dialogo per volta.

Lo script server che gestisce il caricamento del file deve prevedere una richiesta `POST` HTTP con gli elementi seguenti:

- `Content-Type` con un valore `multipart/form-data`.
- `Content-Disposition` con un attributo `name` impostato su "`Filedata`" e un attributo `filename` impostato sul nome del file originale. È possibile specificare un attributo `name` personalizzato passando un valore per il parametro `uploadDataFieldName` nel metodo `FileReference.upload()`.
- Il contenuto binario del file.

Di seguito è riportato un esempio di richiesta HTPP `POST`:

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filedata"; filename="sushi.jpg"
Content-Type: application/octet-stream

Test File
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
(actual file data,,,)
```

Il seguente esempio di richiesta `POST` HTTP invia tre variabili `POST`: `api_sig`, `api_key` e `auth_token` e utilizza `"photo"` come valore personalizzato per il nome del campo dati di caricamento:

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_sig"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
```

```
Content-Disposition: form-data; name="api_key"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="auth_token"

XXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="photo"; filename="sushi.jpg"
Content-Type: application/octet-stream

(actual file data,,,)
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7--
```

Caricamento di file su un server

Per caricare dei file su un server, chiamare innanzi tutto il metodo `browse()` per consentire all'utente di selezionare uno o più file. Dopo che è stato chiamato il metodo `FileReference.upload()`, il file selezionato viene trasferito sul server. Se l'utente seleziona più file utilizzando il metodo `FileReferenceList.browse()`, viene creato automaticamente un array dei file selezionati denominato `FileReferenceList.fileList`. A questo punto è possibile utilizzare il metodo `FileReference.upload()` per caricare ogni file individualmente.

NOTA Il metodo `FileReference.browse()` consente di caricare solo file singoli. Per consentire a un utente di caricare più file, utilizzare il metodo `FileReferenceList.browse()`.

Per impostazione predefinita, la finestra di dialogo del selettore del sistema consente agli utenti di selezionare qualsiasi tipo di file dal computer locale, tuttavia gli sviluppatori possono specificare uno o più filtri per tipi di file personalizzati utilizzando la classe FileFilter e passando un array di istanze di FileFilter al metodo `browse()`:

```
var imageTypes:FileFilter = new FileFilter("Images (*.jpg, *.jpeg, *.gif,
  *.png)", "*.jpg; *.jpeg; *.gif; *.png");
var textTypes:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
  "*.txt; *.rtf");
var allTypes:Array = new Array(imageTypes, textTypes);
var fileRef:FileReference = new FileReference();
fileRef.browse(allTypes);
```

Dopo che l'utente ha selezionato i file e ha fatto clic sul pulsante Apri nel selettore del sistema, viene inviato l'evento `Event.SELECT`. Se per selezionare un file da caricare è stato utilizzato il metodo `FileReference.browse()`, per inviare il file a un server Web è necessario il codice riportato di seguito:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
try
{
  var success:Boolean = fileRef.browse();
}
catch (error:Error)
{
  trace("Unable to browse for files.");
}
function selectHandler(event:Event):void
{
  var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
  fileUploadScript.cfm")
  try
  {
    fileRef.upload(request);
  }
  catch (error:Error)
  {
    trace("Unable to upload file.");
  }
}
function completeHandler(event:Event):void
{
  trace("uploaded");
}
```

SUGGERIMENTO

È possibile inviare dati al server con il metodo `FileReference.upload()` utilizzando le proprietà `URLRequest.method` e `URLRequest.data` per inviare variabili con il metodo `POST` o `GET`.

Quando si tenta di caricare il file utilizzando il metodo `FileReference.upload()`, è possibile che venga inviato uno qualsiasi dei seguenti eventi:

- `Event.OPEN`: inviato quando inizia un'operazione di caricamento.
- `ProgressEvent.PROGRESS`: inviato periodicamente durante l'operazione di caricamento del file.
- `Event.COMPLETE`: inviato quando l'operazione di caricamento dei file viene completata correttamente.
- `SecurityErrorEvent.SECURITY_ERROR`: inviato quando un caricamento non riesce a causa di un violazione della sicurezza.
- `HTTPStatusEvent.HTTP_STATUS`: inviato quando un caricamento non riesce a causa di un errore HTTP.
- `IOErrorEvent.IO_ERROR`: inviato se il caricamento non viene completato per uno dei seguenti motivi:
  - Si verifica un errore di input/output durante la lettura, la scrittura o la trasmissione del file da parte del lettore.
  - Il file SWF ha tentato di caricare un file su un server che richiede l'autenticazione (ad esempio, un nome utente e una password). Durante il caricamento, Flash Player non fornisce all'utente un modo per immettere una password.
  - Il parametro `url` contiene un protocollo non valido. Il metodo `FileReference.upload()` deve utilizzare HTTP o HTTPS.

SUGGERIMENTO

Flash Player non offre il supporto completo per i server che richiedono l'autenticazione. Solo i file SWF in esecuzione in un browser (ovvero quelli che utilizzano il plug-in per il browser o il controllo Microsoft ActiveX®) possono fornire una finestra di dialogo per richiedere all'utente di immettere un nome utente e una password per l'autenticazione, e solo per gli scaricamenti. Per i caricamenti mediante il plug-in o il controllo ActiveX o per il caricamento/scaricamento mediante un lettore autonomo o esterno, il trasferimento dei file non è possibile.

Se si crea uno script server in ColdFusion per accettare il caricamento di un file da Flash Player, è possibile utilizzare codice analogo a quello riportato di seguito:

```
<cffile action="upload" filefield="Filedata" destination="#ExpandPath('./
  ')#" nameconflict="OVERWRITE" />
```

Questo codice ColdFusion carica il file inviato da Flash Player e lo salva nella stessa directory del modello di ColdFusion, sovrascrivendo gli eventuali file con lo stesso nome. Il codice sopra riportato rappresenta la quantità minima di codice necessaria per accettare il caricamento di un file; non utilizzare questo script in un ambiente di produzione. Idealmente è opportuno aggiungere la convalida dei dati per assicurare che gli utenti carichino solo i tipi di file accettati, ad esempio un'immagine anziché uno script sul lato server potenzialmente pericoloso.

Il codice riportato di seguito illustra il caricamento di file utilizzando PHP e include la convalida dei dati. Lo script limita a 10 il numero di file caricati nella directory di caricamento, assicura che le dimensioni del file siano inferiori a 200 KB e consente il caricamento e il salvataggio nel file system solo di file JPEG, GIF o PNG.

```
<?php
$MAXIMUM_FILESIZE = 1024 * 200; // 200KB
$MAXIMUM_FILE_COUNT = 10; // keep maximum 10 files on server
echo exif_imagetype($_FILES['Filedata']);
if ($_FILES['Filedata']['size'] <= $MAXIMUM_FILESIZE)
{
  move_uploaded_file($_FILES['Filedata']['tmp_name'], "./temporary/
  ".$_FILES['Filedata']['name']);
  $type = exif_imagetype("./temporary/".$_FILES['Filedata']['name']);
  if ($type == 1 || $type == 2 || $type == 3)
  {
    rename("./temporary/".$_FILES['Filedata']['name'], "./images/
  ".$_FILES['Filedata']['name']);
  }
  else
  {
    unlink("./temporary/".$_FILES['Filedata']['name']);
  }
}
$directory = opendir('./images/');
$files = array();
while ($file = readdir($directory))
{
  array_push($files, array('./images/'.$file, filectime('./images/
  '.$file')));
}
usort($files, sorter);
if (count($files) > $MAXIMUM_FILE_COUNT)
{
  $files_to_delete = array_splice($files, 0, count($files) -
  $MAXIMUM_FILE_COUNT);
  for ($i = 0; $i < count($files_to_delete); $i++)
  {
    unlink($files_to_delete[$i][0]);
  }
}
```

```
print_r($files);
closedir($directory);

function sorter($a, $b)
{
  if ($a[1] == $b[1])
  {
    return 0;
  }
  else
  {
    return ($a[1] < $b[1]) ? -1 : 1;
  }
}
?>
```

È possibile passare variabili aggiuntive allo script di caricamento utilizzando il metodo di richiesta `POST` o `GET`. Per inviare variabili `POST` aggiuntive allo script di caricamento, è possibile utilizzare il codice riportato di seguito:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();
function selectHandler(event:Event):void
{
  var params:URLVariables = new URLVariables();
  params.date = new Date();
  params.ssid = "94103-1394-2345";
  var request:URLRequest = new URLRequest("http://www.yourdomain.com/
  FileReferenceUpload/fileupload.cfm");
  request.method = URLRequestMethod.POST;
  request.data = params;
  fileRef.upload(request, "Custom1");
}
function completeHandler(event:Event):void
{
  trace("uploaded");
}
```

Nell'esempio precedente viene creato un nuovo oggetto URLVariables che viene passato allo script sul lato server remoto. Nelle versioni precedenti di ActionScript era possibile passare variabili allo script di caricamento sul server mediante valori inclusi nella stringa di query. ActionScript 3.0 consente di passare le variabili allo script remoto utilizzando un oggetto URLRequest, che consente di passare i dati utilizzando il metodo `POST` o `GET`; in questo modo diventa inoltre più semplice passare set di dati più estesi. Per specificare se le variabili vengono passate utilizzando il metodo di richiesta `GET` o `POST`, è possibile impostare la proprietà `URLRequest.method` rispettivamente su `URLRequestMethod.GET` o su `URLRequestMethod.POST`.

ActionScript 3.0 consente inoltre di ignorare il nome di campo del file di caricamento predefinito `Filedata` fornendo un secondo parametro al metodo `upload()`, come illustrato nell'esempio precedente (dove il valore predefinito `Filedata` è stato sostituito con `Custom1`).

Per impostazione predefinita, Flash Player non tenta di inviare un caricamento di prova, anche se è possibile ignorare questa impostazione passando il valore `true` come terzo parametro al metodo `upload()`. Lo scopo del caricamento di prova è di verificare se il caricamento del file vero e proprio darà esito positivo, così come l'autenticazione sul server (se richiesta).

NOTA	Attualmente il caricamento di prova è disponibile solo per Flash Player in ambiente Windows.

Scaricamento di file da un server

È possibile consentire agli utenti di scaricare i file da un server utilizzando il metodo `FileReference.download()`, che accetta due parametri: `request` e `defaultFileName`. Il primo parametro è l'oggetto URLRequest che contiene l'URL del file da scaricare. Il secondo parametro è facoltativo e consente di specificare un nome file predefinito che verrà visualizzato nella finestra di dialogo di scaricamento del file. Se si omette il secondo parametro, `defaultFileName`, viene utilizzato il nome file dall'URL specificato.

Il codice riportato di seguito consente di scaricare un file di nome index.xml dalla stessa directory del documento SWF:

```
var request:URLRequest = new URLRequest("index.xml");
var fileRef:FileReference = new FileReference();
fileRef.download(request);
```

Per impostare il nome predefinito su currentnews.xml anziché su index.xml, specificare il parametro `defaultFileName`, come illustrato nello snippet di codice seguente:

```
var request:URLRequest = new URLRequest("index.xml");
var fileToDownload:FileReference = new FileReference();
fileToDownload.download(request, "currentnews.xml");
```

La capacità di rinominare un file può risultare molto utile se il nome file del server non è intuitivo o è stato generato dal server. È inoltre opportuno specificare il parametro `defaultFileName` quando si scarica un file utilizzando uno script sul lato server, anziché direttamente. Ad esempio, è necessario specificare il parametro `defaultFileName` se si utilizza uno script sul lato server che scarica i file in base alle variabili URL passate. Diversamente, il nome predefinito del file scaricato corrisponde al nome dello script sul lato server.

È possibile inviare dati al server tramite il metodo `download()` aggiungendo dei parametri all'URL, che lo script server deve analizzare. Lo snippet di codice ActionScript 3.0 riportato di seguito consente di scaricare un documento in base ai parametri passati a uno script ColdFusion:

```
package
{
  import flash.display.Sprite;
  import flash.net.FileReference;
  import flash.net.URLRequest;
  import flash.net.URLRequestMethod;
  import flash.net.URLVariables;

  public class DownloadFileExample extends Sprite
  {
    private var fileToDownload:FileReference;
    public function DownloadFileExample()
    {
      var request:URLRequest = new URLRequest();
      request.url = "http://www.[yourdomain].com/downloadfile.cfm";
      request.method = URLRequestMethod.GET;
      request.data = new URLVariables("id=2");
      fileToDownload = new FileReference();
      try
      {
        fileToDownload.download(request, "file2.txt");
      }
      catch (error:Error)
      {
        trace("Unable to download file.");
      }
    }
  }
}
```

Il codice riportato di seguito illustra lo script ColdFusion, download.cfm, che scarica uno dei due file dal server, a seconda del valore di una variabile URL:

```
<cfparam name="URL.id" default="1" />
<cfswitch expression="#URL.id#">
  <cfcase value="2">
    <cfcontent type="text/plain" file="#ExpandPath('two.txt')#"
  deletefile="No" />
  </cfcase>
  <cfdefaultcase>
    <cfcontent type="text/plain" file="#ExpandPath('one.txt')#"
  deletefile="No" />
  </cfdefaultcase>
</cfswitch>
```

Classe FileReferenceList

La classe FileReferenceList consente all'utente di selezionare uno o più file da caricare in uno script sul lato server. Il caricamento del file viene gestito dal metodo `FileReference.upload()`, che deve essere chiamato su ogni file selezionato dall'utente.

Il codice riportato di seguito crea due oggetti FileFilter (`imageFilter` e `textFilter`) e li passa in un array al metodo `FileReferenceList.browse()`. Di conseguenza, nella finestra di dialogo dei file del sistema operativo vengono visualizzati due possibili filtri per i tipi di file.

```
var imageFilter:FileFilter = new FileFilter("Image Files (*.jpg, *.jpeg,
  *.gif, *.png)", "*.jpg; *.jpeg; *.gif; *.png");
var textFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
  "*.txt; *.rtf");
var fileRefList:FileReferenceList = new FileReferenceList();
try
{
  var success:Boolean = fileRefList.browse(new Array(imageFilter,
  textFilter));
}
catch (error:Error)
{
  trace("Unable to browse for files.");
}
```

Consentire all'utente di selezionare e caricare uno o più file utilizzando la classe FileReferenceList equivale a utilizzare `FileReference.browse()` per selezionare i file, anche se FileReferenceList consente di selezionare più di un file. Per caricare più file, è necessario utilizzare `FileReference.upload()` per caricare ogni file selezionato, come illustrato dal codice riportato di seguito:

```
var fileRefList:FileReferenceList = new FileReferenceList();
fileRefList.addEventListener(Event.SELECT, selectHandler);
fileRefList.browse();

function selectHandler(event:Event):void
{
  var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
  fileUploadScript.cfm");
  var file:FileReference;
  var files:FileReferenceList = FileReferenceList(event.target);
  var selectedFileArray:Array = files.fileList;
  for (var i:uint = 0; i < selectedFileArray.length; i++)
  {
    file = FileReference(selectedFileArray[i]);
    file.addEventListener(Event.COMPLETE, completeHandler);
    try
    {
      file.upload(request);
    }
```

```
      catch (error:Error)
      {
        trace("Unable to upload files.");
      }
    }
}
function completeHandler(event:Event):void
{
  trace("uploaded");
}
```

Poiché l'evento `Event.COMPLETE` viene aggiunto a ogni singolo oggetto FileReference nell'array, viene chiamato automaticamente il metodo `completeHandler()` al termine del caricamento di ogni file.

Esempio: Creazione di un client Telnet

L'esempio relativo a Telnet illustra le tecniche necessarie per connettersi a un server remoto e trasmettere i dati utilizzando la classe Socket, Nell'esempio vengono descritte le seguenti tecniche:

- Creazione di un client telnet personalizzato utilizzando la classe Socket
- Invio di testo al server remoto utilizzando un oggetto ByteArray
- Gestione dei dati ricevuti da un server remoto

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione Telnet sono disponibili nella cartella Samples/Telnet. L'applicazione è costituita dai seguenti file:

File	Descrizione
TelnetSocket.mxml	Il file dell'applicazione principale costituito dall'interfaccia utente MXML.
com/example/programmingas3/Telnet/Telnet.as	Fornisce la funzionalità del client Telnet per l'applicazione, ad esempio per connettersi a un server remoto e inviare, ricevere e visualizzare i dati.

Panoramica dell'applicazione socket Telnet

Il file TelnetSocket.mxml principale viene utilizzato per la creazione dell'interfaccia utente (UI) dell'applicazione.

Oltre all'interfaccia utente, questo file definisce due metodi, `login()` e `sendCommand()`, per connettere l'utente al server specificato.

Il codice riportato di seguito elenca il codice ActionScript nel file dell'applicazione principale:

```
import com.example.programmingas3.socket.Telnet;

private var telnetClient:Telnet;
private function connect():void
{
  telnetClient = new Telnet(serverName.text, int(portNumber.text), output);
  console.title = "Connecting to " + serverName.text + ":" +
  portNumber.text;
  console.enabled = true;
}
private function sendCommand():void
{
  var ba:ByteArray = new ByteArray();
  ba.writeMultiByte(command.text + "\n", "UTF-8");
  telnetClient.writeBytesToSocket(ba);
  command.text = "";
}
```

La prima riga di codice importa la classe Telnet dal pacchetto personalizzato com.example.programmingas.socket. La seconda riga di codice dichiara un'istanza della classe Telnet, `telnetClient`, che verrà inizializzata successivamente dal metodo `connect()`. Viene quindi dichiarato il metodo `connect()` e viene inizializzata la variabile `telnetClient` dichiarata in precedenza. Questo metodo passa il nome del server specificato dall'utente, la porta del server telnet e un riferimento a un componente TextArea nell'elenco di visualizzazione, che viene utilizzato per visualizzare le risposte in formato testo del server socket. Le ultime due righe del metodo `connect()` impostano la proprietà `title` per il pannello e attivano il componente pannello, che consente all'utente di inviare dati al server remoto. Il metodo finale nel file dell'applicazione principale, `sendCommand()`, viene utilizzato per inviare i comandi dell'utente al server remoto come oggetto ByteArray.

Panoramica della classe Telnet

La classe Telnet viene utilizzata per connettersi al server Telnet remoto e per inviare/ricevere i dati.

La classe Telnet dichiara le seguenti variabili private:

```
private var serverURL:String;
private var portNumber:int;
private var socket:Socket;
private var ta:TextArea;
private var state:int = 0;
```

La prima variabile, `serverURL`, contiene l'indirizzo del server a cui connettersi specificato dall'utente.

La seconda variabile, `portNumber`, specifica il numero di porta su cui è attualmente in esecuzione il server Telnet. Per impostazione predefinita, il servizio Telnet viene eseguito sulla porta 23e.

La terza variabile, `socket`, è un'istanza di Socket che tenterà di connettersi al server definito dalle variabili `serverURL` e `portNumber`.

La quarta variabile, `ta`, è un riferimento a un'istanza del componente TextArea sullo stage. Questo componente viene utilizzato per visualizzare le risposte dal server Telnet remoto o gli eventuali messaggi di errore.

La variabile finale, `state`, è un valore numerico che viene utilizzato per determinare le opzioni supportate dal client Telnet.

Come è già stato illustrato in precedenza, la funzione di costruzione della classe Telnet viene chiamata dal metodo `connect()` nel file dell'applicazione principale.

La funzione di costruzione Telnet accetta tre parametri: `server`, `port` e `output`. I parametri `server` e `port` specificano il nome del server e il numero di porta su cui è in esecuzione il server Telnet. Il parametro finale, `output`, è un riferimento a un'istanza del componente TextArea sullo stage dove verrà visualizzato l'output del server.

```
public function Telnet(server:String, port:int, output:TextArea)
{
  serverURL = server;
  portNumber = port;
  ta = output;
  socket = new Socket();
  socket.addEventListener(Event.CONNECT, connectHandler);
  socket.addEventListener(Event.CLOSE, closeHandler);
  socket.addEventListener(ErrorEvent.ERROR, errorHandler);
  socket.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
  socket.addEventListener(ProgressEvent.SOCKET_DATA, dataHandler);
  Security.loadPolicyFile("http://" + serverURL + "/crossdomain.xml");
  try
  {
    msg("Trying to connect to " + serverURL + ":" + portNumber + "\n");
    socket.connect(serverURL, portNumber);
  }
```

```
  catch (error:Error)
  {
    msg(error.message + "\n");
    socket.close();
  }
}
```

Scrittura dei dati su un socket

Per scrivere i dati su una connessione socket, chiamare uno dei metodi di scrittura nella classe Socket (ad esempio, `writeBoolean()`, `writeByte()`, `writeBytes()` o `writeDouble()`), quindi scaricare i dati nel buffer di output utilizzando il metodo `flush()`. Nel server Telnet, i dati vengono scritti sulla connessione socket utilizzando il metodo `writeBytes()` che utilizza l'array di byte come parametro e lo invia al buffer di output. Di seguito è riportato il metodo `writeBytesToSocket()`:

```
public function writeBytesToSocket(ba:ByteArray):void
{
  socket.writeBytes(ba);
  socket.flush();
}
```

Questo metodo viene chiamato dal metodo `sendCommand()` del file dell'applicazione principale.

Visualizzazione di messaggi dal server socket

Ogni volta che si riceve un messaggio dal server socket o si verifica un evento, viene chiamato il metodo personalizzato `msg()`. Questo metodo aggiunge una stringa al componente TextArea sullo stage e chiama un metodo personalizzato `setScroll()`, che scorre il componente TextArea fino alla fine. Di seguito è riportato il metodo `msg()`:

```
private function msg(value:String):void
{
  ta.text += value;
  setScroll();
}
```

Se il contenuto del componente TextArea non scorre automaticamente fino alla fine, gli utenti dovranno trascinare manualmente le barre di scorrimento nell'area di testo per visualizzare l'ultima risposta dal server.

Scorrimento di un componente TextArea

Il metodo `setScroll()` contiene un'unica riga di codice ActionScript che attiva lo scorrimento verticale del contenuto del componente TextArea, in modo che l'utente possa visualizzare l'ultima riga del testo restituito. Nello snippet di codice seguente, viene illustrato il metodo `setScroll()`:

```
public function setScroll():void
{
  ta.verticalScrollPosition = ta.maxVerticalScrollPosition;
}
```

Questo metodo imposta la proprietà `verticalScrollPosition` che corrisponde al numero di riga della prima riga di caratteri attualmente visualizzata e lo imposta come valore della proprietà `maxVerticalScrollPosition`.

Esempio: Caricamento e scaricamento di file

Nell'esempio FileIO vengono illustrate le tecniche necessarie per eseguire lo scaricamento e il caricamento di file in Flash Player, descritte di seguito:

- Scaricamento di file nel computer di un utente
- Caricamento di file dal computer di un utente su un server
- Annullamento di uno scaricamento in corso
- Annullamento di un caricamento in corso

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione FileIO sono disponibili nella cartella Samples/FileIO. L'applicazione è costituita dai seguenti file:

File	Descrizione
FileIO.fla oppure FileIO.mxml	Il file principale dell'applicazione in Flash (FLA) o Flex (MXML)
com/example/programmingas3/fileio/FileDownload.as	Una classe che include i metodi per scaricare i file da un server.
com/example/programmingas3/fileio/FileUpload.as	Una classe che include i metodi per caricare i file su un server.

Panoramica dell'applicazione FileIO

L'applicazione FileIO include l'interfaccia utente che consente a un utente di caricare o scaricare file utilizzando Flash Player. L'applicazione definisce innanzitutto due componenti personalizzati, FileUpload e FileDownload, contenuti nel pacchetto com.example.programmingas3.fileio. Dopo l'invio dell'evento `contentComplete` da parte di ogni componente personalizzato, viene chiamato il metodo `init()` del componente e vengono passati i riferimenti a un'istanza dei componenti ProgressBar e Button, che consente agli utenti di visualizzare l'avanzamento del caricamento o dello scaricamento di un file o l'annullamento del trasferimento di file in corso.

Il codice pertinente del file FileIO.mxml è il seguente (si noti che nella versione Flash, il file FLA contiene dei componenti posizionati sullo stage, i cui nomi corrispondono a quelli dei componenti Flex descritti in questa operazione):

```
<example:FileUpload id="fileUpload"
  creationComplete="fileUpload.init(uploadProgress, cancelUpload);" />
<example:FileDownload id="fileDownload"
  creationComplete="fileDownload.init(downloadProgress, cancelDownload);" />
```

Il codice riportato di seguito illustra il pannello di caricamento dei file che contiene una barra di avanzamento e due pulsanti. Il primo pulsante, `startUpload`, chiama il metodo `FileUpload.startUpload()`, che a propria volta chiama il metodo `FileReference.browse()`. Di seguito è riportato un estratto del codice per il pannello di caricamento dei file:

```
<mx:Panel title="Upload File" paddingTop="10" paddingBottom="10"
  paddingLeft="10" paddingRight="10">
  <mx:ProgressBar id="uploadProgress" label="" mode="manual" />
  <mx:ControlBar horizontalAlign="right">
    <mx:Button id="startUpload" label="Upload..."
  click="fileUpload.startUpload();" />
    <mx:Button id="cancelUpload" label="Cancel"
  click="fileUpload.cancelUpload();" enabled="false" />
  </mx:ControlBar>
</mx:Panel>
```

Questo codice colloca un'istanza del componente ProgressBar e due istanze del componente Button sullo stage. Quando l'utente fa clic sul pulsante di caricamento (`startUpload`), viene aperta una finestra di dialogo del sistema operativo che consente all'utente di selezionare un file da caricare su un server remoto. L'altro pulsante, `cancelUpload`, è disattivato per impostazione predefinita, tuttavia quando un utente inizia il caricamento di un file, il pulsante viene attivato e consente di interrompere il trasferimento del file in qualsiasi momento.

Di seguito è riportato il codice per il pannello di scaricamento dei file:

```
<mx:Panel title="Download File" paddingTop="10" paddingBottom="10"
  paddingLeft="10" paddingRight="10">
  <mx:ProgressBar id="downloadProgress" label="" mode="manual" />
  <mx:ControlBar horizontalAlign="right">
    <mx:Button id="startDownload" label="Download..."
  click="fileDownload.startDownload();" />
    <mx:Button id="cancelDownload" label="Cancel"
  click="fileDownload.cancelDownload();" enabled="false" />
  </mx:ControlBar>
</mx:Panel>
```

Questo codice è molto simile a quello utilizzato per il caricamento dei file. Quando l'utente fa clic sul pulsante di scaricamento, (`startDownload`), viene chiamato il metodo `FileDownload.startDownload()` che inizia lo scaricamento del file specificato nella variabile `FileDownload.DOWNLOAD_URL`. Mentre il file viene scaricato, la barra di avanzamento viene aggiornata e mostra la percentuale del file già scaricata. L'utente può annullare lo scaricamento in qualsiasi momento facendo clic sul pulsante `cancelDownload`, che interrompe immediatamente l'avanzamento dell'operazione.

Scaricamento di file da un server remoto

Lo scaricamento dei file da un server remoto viene gestito dalla classe flash.net.FileReference e dalla classe personalizzata com.example.programmingas3.fileio.FileDownload. Quando l'utente fa clic sul pulsante di scaricamento, inizia automaticamente lo scaricamento del file specificato nella variabile `DOWNLOAD_URL` della classe FileDownload.

La classe FileDownload definisce innanzitutto quattro variabili contenute nel pacchetto com.example.programmingas3.fileio, come illustrato dal codice riportato di seguito:

```
  /**
   * Specifica a livello di codice l'URL del file da scaricare sul computer
  dell'utente.
   */
  private const DOWNLOAD_URL:String = "http://www.yourdomain.com/
  file_to_download.zip";

  /**
   * Crea un'istanza di FileReference per gestire lo scaricamento del file.
   */
  private var fr:FileReference;

  /**
   * Definisce il riferimento al componente ProgressBar di scaricamento.
   */
  private var pb:ProgressBar;
```

```
/**
 * Definisce il riferimento al pulsante di annullamento che arresterà
 * immediatamente lo scaricamento attualmente in corso.
 */
private var btn:Button;
```

La prima variabile, `DOWNLOAD_URL`, contiene il percorso del file che viene scaricato nel computer dell'utente quando quest'ultimo fa clic sul pulsante di scaricamento nel file dell'applicazione principale.

La seconda variabile, `fr`, è un oggetto FileReference che viene inizializzato nel metodo `FileDownload.init()` e gestisce lo scaricamento del file remoto nel computer dell'utente.

Le ultime due variabili, `pb` e `btn`, contengono i riferimenti alle istanze dei componenti ProgressBar e Button sullo stage, che vengono inizializzate dal metodo `FileDownload.init()`.

Inizializzazione del componente FileDownload

Il componente FileDownload viene inizializzato mediante la chiamata al metodo `init()` nella classe FileDownload. Questo metodo accetta due parametri, `pb` e `btn`, che corrispondono rispettivamente alle istanze dei componenti ProgressBar e Button.

Di seguito è riportato il codice per il metodo `init()`:

```
/**
 * Imposta i riferimenti ai componenti e aggiunge i listener per gli
 * eventi OPEN, PROGRESS e COMPLETE.
 */
public function init(pb:ProgressBar, btn:Button):void
{
  // Imposta i riferimenti alla barra di avanzamento e al pulsante di
  // annullamento, che vengono passati dallo script chiamante.
  this.pb = pb;
  this.btn = btn;

  fr = new FileReference();
  fr.addEventListener(Event.OPEN, openHandler);
  fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
  fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

Inizio dello scaricamento del file

Quando l'utente fa clic sull'istanza del componente Download Button sullo stage, il metodo `startDownload()` inizia il processo di scaricamento del file. Nell'estratto di codice riportato di seguito viene illustrato il metodo `startDownload()`:

```
/**
 * Inizia a scaricare il file specificato nella costante DOWNLOAD_URL.
 */
public function startDownload():void
{
  var request:URLRequest = new URLRequest();
  request.url = DOWNLOAD_URL;
  fr.download(request);
}
```

Il metodo `startDownload()` crea innanzitutto un nuovo oggetto URLRequest e imposta l'URL di destinazione sul valore specificato dalla variabile `DOWNLOAD_URL`. Viene quindi chiamato il metodo `FileReference.download()` e l'oggetto URLRequest appena creato viene passato come parametro. Viene quindi aperta una finestra di dialogo del sistema operativo sul computer dell'utente che richiede di selezionare un percorso in cui salvare il documento richiesto. Dopo che l'utente ha selezionato un percorso, viene inviato l'evento `open` (`Event.OPEN`) e viene richiamato il metodo `openHandler()`.

Il metodo `openHandler()` imposta il formato di testo per la proprietà `label` del componente ProgressBar e attiva il pulsante di annullamento che consente all'utente di arrestare immediatamente lo scaricamento in corso. Di seguito è riportato il metodo `openHandler()`:

```
/**
 * Quando viene inviato l'evento OPEN, modificare l'etichetta della barra
 * di avanzamento e attivare il pulsante di annullamento che consente
 * all'utente di interrompere l'operazione di scaricamento.
 */
private function openHandler(event:Event):void
{
  pb.label = "DOWNLOADING %3%%";
  btn.enabled = true;
}
```

Controllo dell'avanzamento dell'operazione di scaricamento di un file

Mentre viene scaricato un file da un server remoto nel computer dell'utente, viene inviato a intervalli regolari l'evento `progress` (`ProgressEvent.PROGRESS`). Ogni volta che viene inviato l'evento `progress`, viene richiamato il metodo `progressHandler()` e l'istanza del componente ProgressBar sullo stage viene aggiornata. Di seguito è riportato il codice per il metodo `progressHandler()`:

```
/**
 * Durante lo scaricamento del file, aggiornare lo stato della barra di
avanzamento.
 */
private function progressHandler(event:ProgressEvent):void
{
  pb.setProgress(event.bytesLoaded, event.bytesTotal);
}
```

L'evento progress ha due proprietà, `bytesLoaded` e `bytesTotal`, che vengono utilizzate per aggiornare il componente ProgressBar sullo stage e mostrano all'utente la quantità di file scaricata rispetto alla quantità rimanente. L'utente può interrompere il trasferimento del file in qualsiasi momento facendo clic sul pulsante di annullamento sotto la barra di avanzamento.

Se il file viene scaricato correttamente, l'evento `complete` (`Event.COMPLETE`) richiama il metodo `completeHandler()`, che comunica all'utente che lo scaricamento del file è stato completato e disattiva il pulsante di annullamento. Di seguito è riportato il codice per il metodo `completeHandler()`:

```
/**
 * Una volta completato lo scaricamento, modificare per l'ultima volta
 * l'etichetta della barra di avanzamento e disabilitare il pulsante
 * "Annulla", poiché lo scaricamento è già stato completato.
 */
private function completeHandler(event:Event):void
{
  pb.label = "DOWNLOAD COMPLETE";
  btn.enabled = false;
}
```

Annullamento dello scaricamento del file

L'utente può interrompere il trasferimento del file e interrompere lo scaricamento di ulteriori byte in qualsiasi momento facendo clic sul pulsante di annullamento presente sullo stage. Di seguito è riportato un estratto del codice per annullare lo scaricamento:

```
/**
 * Annullare lo scaricamento corrente del file.
 */
public function cancelDownload():void
{
  fr.cancel();
  pb.label = "DOWNLOAD CANCELLED";
  btn.enabled = false;
}
```

Innanzi tutto, il codice interrompe immediatamente il trasferimento del file impedendo lo scaricamento di ulteriori dati. Quindi, la proprietà label della barra di avanzamento viene aggiornata per informare l'utente che lo scaricamento è stato annullato. Infine, il pulsante di annullamento viene disattivato, impedendo all'utente di fare clic sul pulsante finché non tenterà di nuovo di scaricare il file.

Caricamento di file su un server remoto

Il processo di caricamento dei file è molto simile a quello per lo scaricamento. La classe FileUpload dichiara le stesse quattro variabili, come illustrato nel codice seguente:

```
private const UPLOAD_URL:String = "http://www.yourdomain.com/
  your_upload_script.cfm";
private var fr:FileReference;
private var pb:ProgressBar;
private var btn:Button;
```

A differenza della variabile `FileDownload.DOWNLOAD_URL`, la variabile `UPLOAD_URL` contiene l'URL per lo script sul lato server che caricherà il file dal computer dell'utente. Il comportamento delle tre variabili è analogo a quello delle relative controparti nella classe FileDownload.

Inizializzazione del componente FileUpload

Il componente FileUpload contiene un metodo `init()`, che viene chiamato dall'applicazione principale. Questo metodo accetta due parametri, `pb` e `btn`, che fanno riferimento rispettivamente a un'istanza dei componenti ProgressBar e Button sullo stage. Quindi il metodo `init()` inizializza l'oggetto FileReference definito precedentemente nella classe FileUpload. Infine, il metodo assegna quattro listener di eventi all'oggetto FileReference. Di seguito è riportato il codice per il metodo `init()`:

```
public function init(pb:ProgressBar, btn:Button):void
{
  this.pb = pb;
  this.btn = btn;

  fr = new FileReference();
  fr.addEventListener(Event.SELECT, selectHandler);
  fr.addEventListener(Event.OPEN, openHandler);
  fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
  fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

Inizio del caricamento del file

Il caricamento del file ha inizio quando l'utente fa clic sul pulsante di caricamento sullo stage, che richiama il metodo the `FileUpload.startUpload()`. Questo metodo chiama il metodo `browse()` della classe FileReference, che attiva la visualizzazione di una finestra di dialogo del sistema operativo in cui viene richiesto all'utente di selezionare un file da caricare sul server remoto. Di seguito è riportato un estratto del codice per il metodo `startUpload()`:

```
public function startUpload():void
{
  fr.browse();
}
```

Dopo che l'utente ha selezionato un file da caricare, viene inviato l'evento `select` (`Event.SELECT`) che richiama il metodo `selectHandler()`. Il metodo `selectHandler()` crea un nuovo oggetto URLRequest e imposta la proprietà `URLRequest.url` sul valore dalla costante `UPLOAD_URL` definito precedentemente nel codice. Infine, l'oggetto FileReference carica il file selezionato nello script sul lato server specificato. Di seguito è riportato il codice per il metodo `selectHandler()`:

```
private function selectHandler(event:Event):void
{
  var request:URLRequest = new URLRequest();
  request.url = UPLOAD_URL;
  fr.upload(request);
}
```

Il codice rimanente nella classe FileUpload è lo stesso di quello definito nella classe FileDownload. Se in qualsiasi momento l'utente desidera terminare il caricamento, può fare clic sul pulsante di annullamento, che imposta l'etichetta sulla barra di avanzamento e interrompe immediatamente il trasferimento del file. La barra di avanzamento viene aggiornata ogni volta che viene inviato l'evento `progress` (`ProgressEvent.PROGRESS`). In modo analogo, una volta completato il caricamento, la barra di avanzamento viene aggiornata per informare l'utente che il file è stato caricato. Il pulsante di annullamento viene quindi disattivato finché l'utente non inizia un nuovo trasferimento di file.

CAPITOLO 23

Ambiente del sistema client

23

In questo capitolo vengono illustrate le procedure per l'interazione con il sistema dell'utente. Viene spiegato come determinare quali funzioni sono supportate e come creare file SWF in più lingue utilizzando l'IME (Input Method Editor) installato nel computer dell'utente (se disponibile). Vengono inoltre descritti gli usi più comuni per i domini delle applicazioni.

Sommario

Nozioni fondamentali sull'ambiente del sistema client

Introduzione all'ambiente del sistema client

Man mano che si creano applicazioni ActionScript più avanzate, può diventare necessario conoscere i dettagli e le funzioni di accesso del sistema operativo dei propri utenti. L'ambiente del sistema client è una raccolta di classi nel pacchetto flash.system che consentono di accedere a funzionalità a livello di sistema come le seguenti:

- Determinazione dell'applicazione e del dominio di sicurezza in cui è in esecuzione un file SWF
- Determinazione delle capacità di Flash Player dell'utente, ad esempio le dimensioni dello schermo (risoluzione) e se sono disponibili determinare funzionalità (come l'audio mp3)

- Creazione di siti multilingua mediante l'IME
- Interazione con il contenitore di Flash Player (che può essere una pagina HTML o un'applicazione contenitore)
- Salvataggio delle informazioni negli Appunti dell'utente

l pacchetto flash.system include inoltre le classi IMEConversionMode e SecurityPanel, che contengono delle costanti statiche utilizzate rispettivamente con le classi IME e Security.

Operazioni comuni con l'ambiente del sistema client

In questo capitolo sono descritte le seguenti operazioni comuni con il sistema client mediante ActionScript:

- Determinazione della quantità di memoria utilizzata dall'applicazione
- Copia del testo negli Appunti dell'utente
- Determinazione delle capacità del computer dell'utente, quali:
  - Risoluzione dello schermo, colori, DPI e rapporto proporzionale dei pixel
  - Sistema operativo
  - Supporto per lo streaming audio, lo streaming video e la riproduzione di file mp3
  - Se il Flash Player installato è in versione debugger
- Operazioni con i domini applicazione:
  - Definizione di un dominio applicazione
  - Separazione del codice dei file SWF nei domini applicazione
- Operazioni con un IME nell'applicazione:
  - Determinazione dell'installazione di un IME
  - Determinazione e impostazione della modalità di conversione IME
  - Disattivazione dell'IME per i campi di testo
  - Rilevamento della conversione IME

Concetti e termini importanti

L'elenco di riferimento seguente contiene dei termini importanti che vengono citati in questo capitolo:

- Sistema operativo: il programma principale (ad esempio, Microsoft Windows, Mac OS X o Linux®) che è in esecuzione su un computer e all'interno del quale sono in esecuzione altre applicazioni.
- Appunti: il contenitore del sistema operativo per il testo e gli elementi che vengono copiati/tagliati e incollati nelle applicazioni.
- Dominio applicazione: un meccanismo di separazione delle classi utilizzato in diversi file SWF che consente ai file SWF di includere classi diverse con lo stesso nome senza che si sovrascrivano a vicenda.
- IME (Input Method Editor): un programma (o uno strumento del sistema operativo) che viene utilizzato per immettere caratteri o simboli complessi mediante una tastiera standard.
- Sistema client: in termini di programmazione, un *client* è la parte di un'applicazione (o l'intera applicazione) che viene eseguita su un computer e utilizzata da un solo utente. Il *sistema client* è il sistema operativo sottostante sul computer dell'utente.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. Tutti questi esempi includono la chiamata appropriata alla funzione `trace()` per la scrittura dei valori da provare. Per provare gli esempi di codice contenuti in questo capitolo:

1. Creare un documento Flash vuoto.
2. Selezionare un fotogramma chiave nella linea temporale.
3. Aprire il pannello Azioni e copiare l'esempio di codice nel riquadro dello script.
4. Eseguire il programma selezionando Controllo > Prova filmato.

   Nel pannello Output vengono visualizzati i risultati della funzione `trace()` dell'esempio di codice.

Alcuni degli esempi di codice che seguono sono più complessi e sono scritti come una classe. Per provare questi esempi:

1. Creare un documento Flash vuoto e salvarlo nel computer.
2. Creare un nuovo file ActionScript e salvarlo nella stessa directory del documento Flash. Il nome file deve corrispondere al nome della classe presente nell'esempio di codice. Ad esempio, se l'esempio di codice definisce una classe chiamata SystemTest, per salvare il file ActionScript, utilizzare il nome SystemTest.as.
3. Copiare l'esempio di codice nel file ActionScript e salvare il file.
4. Nel documento Flash, fare clic in una parte vuota dello stage oppure dell'area di lavoro per attivare la finestra di ispezione Proprietà del documento.
5. Nella finestra di ispezione Proprietà, nel campo Classe documento inserire il nome della classe ActionScript copiata dal testo.
6. Eseguire il programma selezionando Controllo > Prova filmato.

   I risultati dell'esempio vengono visualizzati nel pannello Output.

Per ulteriori informazioni sulle tecniche per la prova degli esempi di codice, vedere "Prova degli esempi di codice contenuti nei capitoli" a pagina 68.

Uso della classe System

La classe System contiene metodi e proprietà che consentono di interagire con il sistema operativo dell'utente e di recuperare informazioni sull'utilizzo della memoria corrente per Adobe Flash Player. I metodi e le proprietà della classe System consentono inoltre di intercettare gli eventi `imeComposition`, di indicare a Flash Player di caricare i file di testo esterni utilizzando la tabella codici corrente dell'utente o nel formato Unicode oppure di impostare il contenuto degli Appunti dell'utente.

Richiesta di dati sul sistema dell'utente in fase di runtime

Mediante il controllo della proprietà `System.totalMemory` è possibile determinare la quantità di memoria (espressa in byte) attualmente utilizzata da Flash Player. Questa proprietà consente di verificare l'utilizzo della memoria e ottimizzare le applicazioni in base alle modifiche dei livelli di memoria. Ad esempio, se un particolare effetto visivo causa un significativo aumento dell'utilizzo della memoria, può essere utile modificare l'effetto o eliminarlo completamente.

La proprietà `System.ime` costituisce un riferimento all'editor del metodo di input (IME) installato. Questa proprietà consente di intercettare gli eventi `imeComposition` (`flash.events.IMEEvent.IME_COMPOSITION`) utilizzando il metodo `addEventListener()`.

La terza proprietà nella classe System è `useCodePage`. Quando è impostata su `true`, Flash Player utilizza la tabella codici tradizionale del sistema operativo su cui è in esecuzione il lettore per interpretare i file di testo esterni. Se si imposta questa proprietà su `false`, Flash Player interpreta i file di testo esterni come Unicode.

Se si imposta la proprietà `System.useCodePage` su `true`, tenere presente che la tabella codici tradizionale del sistema operativo su cui è in esecuzione il lettore deve includere i caratteri utilizzati nel file di testo esterno per consentire la visualizzazione del testo. Ad esempio, se viene caricato un file esterno contenente caratteri cinesi, questi caratteri non possono essere visualizzati su un sistema che utilizza la tabella codici di Windows per l'inglese, poiché questa tabella codici non include i caratteri cinesi.

Per garantire che gli utenti di tutte le piattaforme siano in grado di visualizzare i file di testo esterni utilizzati nei file SWF, codificare tutti i file di testo esterni come Unicode e lasciare la proprietà `System.useCodePage` impostata su `false` per impostazione predefinita. In questo modo Flash Player interpreta il testo come Unicode.

Salvataggio del testo negli Appunti

La classe System include il metodo `setClipboard()` che consente a Flash Player di impostare il contenuto degli Appunti dell'utente con una stringa specifica. Per motivi di sicurezza, non viene fornito un metodo `Security.getClipboard()` in quanto potrebbe consentire a siti potenzialmente dannosi di accedere ai dati copiati negli Appunti dell'utente.

Il codice riportato di seguito illustra come copiare un messaggio di errore negli Appunti dell'utente quando si verifica un errore di sicurezza. Il messaggio di errore può essere utile se l'utente desidera segnalare un potenziale bug di un'applicazione.

```
private function securityErrorHandler(event:SecurityErrorEvent):void
{
  var errorString:String = "[" + event.type + "] " + event.text;
  trace(errorString);
  System.setClipboard(errorString);
}
```

Uso della classe Capabilities

La classe Capabilities consente agli sviluppatori di determinare l'ambiente in cui viene eseguito il file SWF. Utilizzando le diverse proprietà della classe Capabilities è possibile determinare la risoluzione del sistema dell'utente, l'eventuale supporto di componenti software di accessibilità, la lingua del sistema operativo dell'utente, oltre alla versione di Flash Player attualmente installata.

Mediante il controllo delle proprietà della classe Capabilities è possibile personalizzare l'applicazione per ottimizzare l'interazione con l'ambiente specifico dell'utente. Ad esempio, controllando le proprietà `Capabilities.screenResolutionX` e `Capabilities.screenResolutionY` è possibile determinare la risoluzione dello schermo utilizzata dal sistema dell'utente e stabilire quali sono le dimensioni del video più appropriate. In alternativa, è possibile controllare la proprietà `Capabilities.hasMP3` per verificare se il sistema dell'utente supporta la riproduzione del formato mp3 prima di caricare un file mp3 esterno.

Il codice riportato di seguito utilizza un'espressione regolare per analizzare la versione di Flash Player utilizzata dal client:

```
var versionString:String = Capabilities.version;
var pattern:RegExp = /^(\w*) (\d*),(\d*),(\d*),(\d*)$/;
var result:Object = pattern.exec(versionString);
if (result != null)
{
  trace("input: " + result.input);
  trace("platform: " + result[1]);
  trace("majorVersion: " + result[2]);
  trace("minorVersion: " + result[3]);
  trace("buildNumber: " + result[4]);
  trace("internalBuildNumber: " + result[5]);
}
else
{
  trace("Unable to match RegExp.");
}
```

Se si desidera inviare le caratteristiche del sistema dell'utente a uno script sul lato server, in modo che le informazioni possano essere memorizzate in un database, è possibile utilizzare il codice ActionScript riportato di seguito:

```
var url:String = "log_visitor.cfm";
var request:URLRequest = new URLRequest(url);
request.method = URLRequestMethod.POST;
request.data = new URLVariables(Capabilities.serverString);
var loader:URLLoader = new URLLoader(request);
```

Uso della classe ApplicationDomain

Lo scopo della classe ApplicationDomain è quello di consentire la memorizzazione di una tabella di definizioni di ActionScript 3.0. Tutto il codice presente in un file SWF viene definito come esistente all'interno di un dominio dell'applicazione. I domini delle applicazioni sono utilizzati per separare in partizioni le classi che si trovano nello stesso dominio di sicurezza. Consentono di avere più definizioni per la stessa classe e di riutilizzare le definizioni degli elementi principali da parte degli elementi secondari.

I domini delle applicazioni possono essere utilizzati quando si carica un file SWF esterno scritto in ActionScript 3.0 utilizzando l'API della classe Loader. Non è possibile utilizzare i domini delle applicazioni quando si carica un'immagine o un file SWF scritto in ActionScript 1.0 o ActionScript 2.0. Tutte le definizioni di ActionScript 3.0 contenute nella classe Loader vengono salvate nel dominio dell'applicazione. Quando si carica il file SWF, è possibile specificare che venga incluso nello stesso dominio dell'applicazione che contiene l'oggetto Loader impostando il parametro `applicationDomain` dell'oggetto LoaderContext su `ApplicationDomain.currentDomain`. L'inserimento del file SWF caricato nello stesso dominio dell'applicazione consente di accedere direttamente alle relative classi. Ciò può essere utile se si carica un file SWF che include contenuto multimediale incorporato a cui è possibile accedere tramite i nomi delle classi associate oppure se si desidera accedere ai metodi del file SWF caricato, come illustrato nell'esempio riportato di seguito:

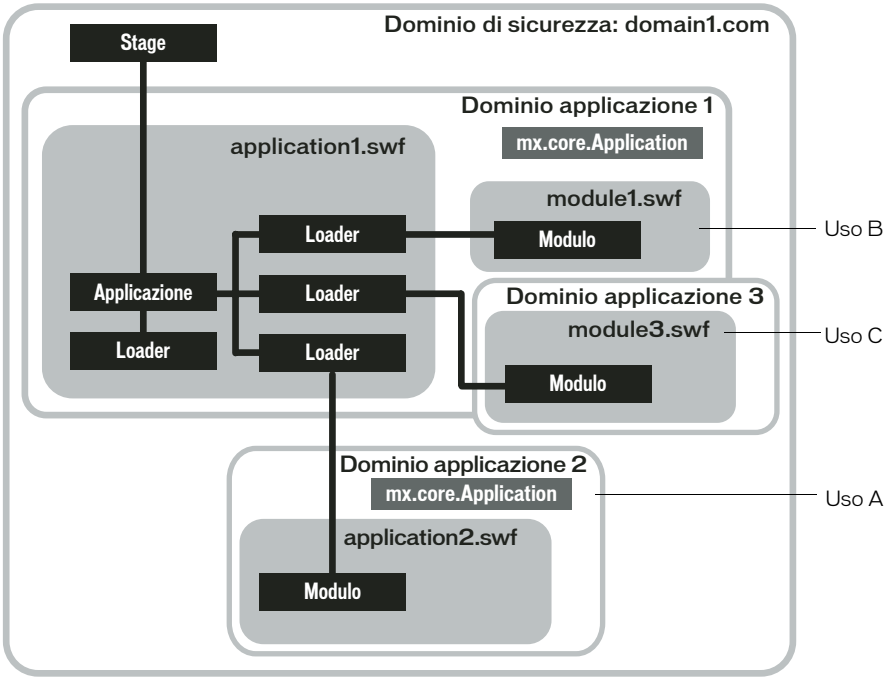
```
package
{
  import flash.display.Loader;
  import flash.display.Sprite;
  import flash.events.*;
  import flash.net.URLRequest;
  import flash.system.ApplicationDomain;
  import flash.system.LoaderContext;

  public class ApplicationDomainExample extends Sprite
  {
    private var ldr:Loader;
    public function ApplicationDomainExample()
    {
      ldr = new Loader();
      var req:URLRequest = new URLRequest("Greeter.swf");
      var ldrContext:LoaderContext = new LoaderContext(false,
  ApplicationDomain.currentDomain);
      ldr.contentLoaderInfo.addEventListener(Event.COMPLETE,
  completeHandler);
      ldr.load(req, ldrContext);
    }
```

```
    private function completeHandler(event:Event):void
    {
      ApplicationDomain.currentDomain.getDefinition("Greeter");
      var myGreeter:Greeter = Greeter(event.target.content);
      var message:String = myGreeter.welcome("Tommy");
      trace(message); // Hello, Tommy
    }
  }
}
```

Di seguito sono elencati altri aspetti da considerare quando si utilizzano i domini dell'applicazione:

- Tutto il codice presente in un file SWF viene definito come esistente all'interno di un dominio dell'applicazione. Il *dominio corrente* è quello in cui viene eseguita l'applicazione principale. Il *dominio di sistema* contiene tutti i domini dell'applicazione, compreso quello corrente; in altri termini, contiene tutte le classi di Flash Player.
- Tutti i domini dell'applicazione, eccetto il dominio di sistema, sono associati a un dominio principale. Il dominio principale del dominio applicazione dell'applicazione principale è il dominio di sistema. Le classi caricate vengono definite soltanto se non sono già definite dal rispettivo elemento principale. Non è consentito sostituire la definizione di una classe caricata.

Nel diagramma seguente è illustrata un'applicazione che carica il contenuto da diversi file SWF all'interno di un unico dominio, domain1.com. A seconda del contenuto caricato, possono essere utilizzati diversi domini dell'applicazione. Il testo seguente descrive la logica utilizzata per impostare il dominio dell'applicazione corretto per ogni file SWF incluso nell'applicazione.



Il file dell'applicazione principale è application1.swf. Contiene gli oggetti Loader che caricano il contenuto da altri file SWF. In questo scenario, il dominio corrente è Dominio applicazione 1. Uso A, Uso B e Uso C illustrano le diverse tecniche per impostare il dominio dell'applicazione appropriato per ogni file SWF incluso in un'applicazione.

Uso A: separazione in partizioni del file SWF secondario creando un dominio di sistema secondario. Nel diagramma viene creato Dominio applicazione 2 come elemento secondario del dominio di sistema. Il file application2.swf viene caricato nel Dominio applicazione 2 e le relative definizioni di classe vengono in tal modo separate in partizioni dalle classi definite nel file application1.swf.

Questa tecnica può essere utilizzata, ad esempio, per fare in modo che un'applicazione datata carichi dinamicamente una versione più recente della stessa applicazione senza creare conflitti. Il conflitto non avviene perché nonostante vengano utilizzati gli stessi nomi di classe, questi sono separati in partizioni in domini dell'applicazione diversi.

Il codice riportato di seguito crea un dominio dell'applicazione secondario rispetto al dominio di sistema:

```
request.url = "application2.swf";
request.applicationDomain = new ApplicationDomain();
```

Uso B: aggiunta di nuove definizioni di classe alle definizioni di classe correnti. Il dominio dell'applicazione del file module1.swf è impostato sul dominio corrente (Dominio applicazione 1). Ciò consente di aggiungere le nuove definizioni di classe all'attuale set di definizioni di classe dell'applicazione. Questo approccio può essere utilizzato per una libreria condivisa in runtime dell'applicazione principale. Il file SWF caricato viene gestito come una libreria condivisa remota (RSL). Utilizzare questa tecnica per caricare le RSL mediante un precaricatore prima dell'avvio dell'applicazione.

Il codice riportato di seguito imposta un dominio dell'applicazione sul dominio corrente:

```
request.url = "module1.swf";
request.applicationDomain = ApplicationDomain.currentDomain;
```

Uso C: utilizzo delle definizioni di classe dell'elemento principale mediante la creazione di un nuovo dominio secondario del dominio corrente. Il dominio dell'applicazione del file module3.swf è un elemento secondario del dominio corrente e tale elemento secondario utilizza le versioni di tutte le classi dell'elemento principale. Questa tecnica può inoltre essere utilizzata, ad esempio, per caricare un modulo di un'applicazione RIA (Rich Internet Application) con più schermate come elemento secondario dell'applicazione principale, che utilizza i tipi dell'applicazione principale. Se è possibile garantire che tutte le classi siano sempre aggiornate ai fini della compatibilità con le versioni precedenti e che l'applicazione che effettua il caricamento sia sempre più recente degli elementi caricati, l'elemento secondario utilizzerà le versioni principali. La disponibilità di un nuovo dominio dell'applicazione consente inoltre di scaricare tutte le definizioni di classe per il processo di garbage collection, a condizione che sia possibile garantire che non si continuerà a utilizzare riferimenti al file SWF secondario.

Questa tecnica consente ai moduli caricati di condividere gli oggetti singleton del caricatore e i membri delle classi statici.

Il codice riportato di seguito crea un nuovo dominio secondario del dominio corrente:

```
request.url = "module3.swf";
request.applicationDomain = new
  ApplicationDomain(ApplicationDomain.currentDomain);
```

Uso della classe IME

La classe IME consente di manipolare l'IME (Input Method Editor) del sistema operativo all'interno di Flash Player.

Utilizzando ActionScript è possibile determinare quanto segue:

- Se sul computer di un utente è installato un IME (`Capabilities.hasIME`)
- Se sul computer di un utente l'IME è abilitato o disabilitato (`IME.enabled`)
- La modalità di conversione utilizzata dall'IME corrente (`IME.conversionMode`)

È possibile associare un campo di testo di input con un contesto IME particolare. Quando si passa da un campo di input a un altro, è anche possibile passare a una modalità dell'IME diversa, ad esempio Hiragana (giapponese), numeri a larghezza intera, numeri a larghezza ridotta, input diretto e così via.

Un IME consente all'utente di digitare caratteri di testo non ASCII nelle lingue multibyte quali il cinese, il giapponese e il coreano.

Per ulteriori informazioni sull'uso degli IME, consultare la documentazione del sistema operativo per il quale si sta sviluppando l'applicazione. Per ulteriori risorse, vedere i siti Web seguenti:

- http://www.microsoft.com/globaldev/default.mspx
- http://developer.apple.com/documentation/
- http://java.sun.com/

NOTA

Se sul computer dell'utente non è stato attivato alcun IME, le chiamate ai metodi o alle proprietà IME diversi da `Capabilities.hasIME` avranno esito negativo. Dopo che un IME è stato attivato manualmente, le successive chiamate ActionScript ai metodi e alle proprietà IME funzioneranno nei modi previsti. Ad esempio, se si utilizza un IME giapponese, è necessario attivarlo prima di chiamare qualsiasi metodo o proprietà IME.

Verifica dell'installazione e attivazione di un IME

Prima di chiamare qualsiasi metodo o proprietà IME, è necessario verificare sempre se sul computer dell'utente è installato e attivato un IME. Il codice riportato di seguito illustra come verificare se l'utente ha installato e attivato un IME prima di chiamare qualsiasi metodo:

```
if (Capabilities.hasIME)
{
  if (IME.enabled)
  {
    trace("IME is installed and enabled.");
  }
  else
  {
    trace("IME is installed but not enabled. Please enable your IME and try
    again.");
  }
}
else
{
  trace("IME is not installed. Please install an IME and try again.");
}
```

Il codice precedente controlla innanzi tutto se sul computer dell'utente è installato un IME utilizzando la proprietà `Capabilities.hasIME`. Se questa proprietà è impostata su `true`, il codice controlla quindi se l'IME dell'utente è attualmente attivato utilizzando la proprietà `IME.enabled`.

Determinazione della modalità di conversione IME attivata

Quando si creano applicazioni in più lingue, può essere necessario determinare quale modalità di conversione è attualmente attivata per l'utente. Il codice riportato di seguito dimostra come verificare se l'utente ha attivato un IME e, in tal caso, quale modalità di conversione è attiva:

```
if (Capabilities.hasIME)
{
  switch (IME.conversionMode)
  {
    case IMEConversionMode.ALPHANUMERIC_FULL:
      tf.text = "Current conversion mode is alphanumeric (full-width).";
      break;
    case IMEConversionMode.ALPHANUMERIC_HALF:
      tf.text = "Current conversion mode is alphanumeric (half-width).";
      break;
```

```
    case IMEConversionMode.CHINESE:
      tf.text = "Current conversion mode is Chinese.";
      break;
    case IMEConversionMode.JAPANESE_HIRAGANA:
      tf.text = "Current conversion mode is Japananese Hiragana.";
      break;
    case IMEConversionMode.JAPANESE_KATAKANA_FULL:
      tf.text = "Current conversion mode is Japanese Katakana (full-
      width).";
      break;
    case IMEConversionMode.JAPANESE_KATAKANA_HALF:
      tf.text = "Current conversion mode is Japanese Katakana (half-
      width).";
      break;
    case IMEConversionMode.KOREAN:
      tf.text = "Current conversion mode is Korean.";
      break;
    default:
      tf.text = "Current conversion mode is " + IME.conversionMode + ".";
      break;
  }
}
else
{
  tf.text = "Please install an IME and try again.";
}
```

Il codice precedente verifica come prima cosa se l'utente ha installato un IME. Quindi, controlla quale modalità di conversione viene utilizzata dall'IME corrente, confrontando la proprietà `IME.conversionMode` con ognuna delle costanti della classe IMEConversionMode.

Impostazione della modalità di conversione IME

Quando si cambia la modalità di conversione IME dell'utente, è necessario verificare che il codice sia racchiuso in un blocco `try..catch`, perché se si imposta una modalità di conversione utilizzando la proprietà `conversionMode`, è possibile che venga generato un errore se l'IME non è in grado di impostare tale modalità di conversione. Il codice riportato di seguito dimostra come utilizzare il blocco `try..catch` quando si imposta la proprietà `IME.conversionMode`:

```
var statusText:TextField = new TextField;
statusText.autoSize = TextFieldAutoSize.LEFT;
addChild(statusText);
if (Capabilities.hasIME)
{
  try
  {
    IME.enabled = true;
    IME.conversionMode = IMEConversionMode.KOREAN;
    statusText.text = "Conversion mode is " + IME.conversionMode + ".";
  }
  catch (error:Error)
  {
    statusText.text = "Unable to set conversion mode.\n" + error.message;
  }
}
```

Il codice precedente crea innanzitutto un campo di testo, che viene utilizzato per visualizzare un messaggio di stato per l'utente. Quindi, se l'IME è installato, il codice attiva l'IME e imposta la modalità di conversione sulla lingua coreana. Se nel computer dell'utente non è installato l'IME per il coreano, viene generato un errore che viene rilevato dal blocco `try..catch`. Il blocco `try..catch` visualizza il messaggio di errore nel campo di testo creato in precedenza.

Disattivazione dell'IME per determinati campi di testo

In alcuni casi, può essere utile disattivare l'IME dell'utente mentre è in corso la digitazione dei caratteri. Ad esempio, se è stato creato un campo di testo che accetta solo input numerico, può non essere consigliabile attivare l'IME e rallentare l'immissione dei dati.

L'esempio riportato di seguito dimostra come impostare l'intercettazione degli eventi `FocusEvent.FOCUS_IN` e `FocusEvent.FOCUS_OUT` e disattivare di conseguenza l'IME dell'utente:

```
var phoneTxt:TextField = new TextField();
var nameTxt:TextField = new TextField();

phoneTxt.type = TextFieldType.INPUT;
phoneTxt.addEventListener(FocusEvent.FOCUS_IN, focusInHandler);
phoneTxt.addEventListener(FocusEvent.FOCUS_OUT, focusOutHandler);
phoneTxt.restrict = "0-9";
phoneTxt.width = 100;
phoneTxt.height = 18;
phoneTxt.background = true;
phoneTxt.border = true;
addChild(phoneTxt);

nameField.type = TextFieldType.INPUT;
nameField.x = 120;
nameField.width = 100;
nameField.height = 18;
nameField.background = true;
nameField.border = true;
addChild(nameField);

function focusInHandler(event:FocusEvent):void
{
  if (Capabilities.hasIME)
  {
    IME.enabled = false;
  }
}
function focusOutHandler(event:FocusEvent):void
{
  if (Capabilities.hasIME)
  {
    IME.enabled = true;
  }
}
```

Questo esempio crea due campi di testo, `phoneTxt` e `nameTxt`, quindi aggiunge due listener di eventi al campo di testo `phoneTxt`. Quando l'utente imposta l'attivazione sul campo di testo `phoneTxt`, viene inviato un evento `FocusEvent.FOCUS_IN` e l'IME viene disattivato. Quando il campo di testo `phoneTxt` non è più attivo, viene inviato l'evento `FocusEvent.FOCUS_OUT` per riattivare l'IME.

Intercettazione degli eventi di composizione IME

Gli eventi di composizione IME vengono inviati quando si imposta una stringa di composizione. Ad esempio, se l'IME dell'utente è attivato e in uso e viene digitata una stringa in giapponese, l'evento `IMEEvent.IME_COMPOSITION` verrà inviato non appena l'utente seleziona la stringa di composizione. Per impostare l'intercettazione dell'evento `IMEEvent.IME_COMPOSITION`, è necessario aggiungere un listener di eventi alla proprietà statica `ime` nella classe System (`flash.system.System.ime.addEventListener(...)`), come illustrato nell'esempio riportato di seguito:

```
var inputTxt:TextField;
var outputTxt:TextField;

inputTxt = new TextField();
inputTxt.type = TextFieldType.INPUT;
inputTxt.width = 200;
inputTxt.height = 18;
inputTxt.border = true;
inputTxt.background = true;
addChild(inputTxt);

outputTxt = new TextField();
outputTxt.autoSize = TextFieldAutoSize.LEFT;
outputTxt.y = 20;
addChild(outputTxt);

if (Capabilities.hasIME)
{
  IME.enabled = true;
  try
  {
    IME.conversionMode = IMEConversionMode.JAPANESE_HIRAGANA;
  }
  catch (error:Error)
  {
    outputTxt.text = "Unable to change IME.";
  }
  System.ime.addEventListener(IMEEvent.IME_COMPOSITION,
  imeCompositionHandler);
}
else
{
  outputTxt.text = "Please install IME and try again.";
}

function imeCompositionHandler(event:IMEEvent):void
{
  outputTxt.text = "you typed: " + event.text;
}
```

Il codice precedente crea due campi di testo e li aggiunge all'elenco di visualizzazione. Il primo, `inputTxt`, è un campo di testo di input che consente all'utente di immettere testo in giapponese. Il secondo, `outputTxt`, è un campo di testo dinamico che visualizza i messaggi di errore per l'utente o che riflette la stringa in giapponese digitata dall'utente nel campo di testo `inputTxt`.

Esempio: Rilevamento delle caratteristiche del sistema

L'esempio CapabilitiesExplorer dimostra come utilizzare la classe flash.system.Capabilities per determinare le caratteristiche supportate da Flash Player nel sistema dell'utente. Nell'esempio vengono descritte le seguenti tecniche:

- Rilevamento delle caratteristiche supportate da Flash Player nel sistema dell'utente utilizzando la classe Capabilities
- Utilizzo della classe ExternalInterface per rilevare le impostazioni del browser supportate dal browser dell'utente

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione CapabilitiesExplorer sono disponibili nella cartella Samples/CapabilitiesExplorer. L'applicazione è costituita dai seguenti file:

File	Descrizione
CapabilitiesExplorer.fla oppure CapabilitiesExplorer.mxml	Il file principale dell'applicazione in Flash (FLA) o Flex (MXML)
com/example/programmingas3/capabilities/CapabilitiesGrabber.as	La classe che fornisce la funzionalità principale dell'applicazione, compresa l'aggiunta delle caratteristiche del sistema a un array, l'ordinamento degli elementi e l'utilizzo della classe ExternalInterface per recuperare le caratteristiche del browser.
capabilities.html	Un contenitore HTML che include il codice JavaScript necessario per comunicare con l'API esterna.

Panoramica di CapabilitiesExplorer

Il file CapabilitiesExplorer.mxml viene utilizzato per impostare l'interfaccia utente dell'applicazione CapabilitiesExplorer. Le caratteristiche di Flash Player nel sistema dell'utente saranno visualizzate in un'istanza del componente DataGrid sullo stage. Se l'applicazione viene eseguita da un contenitore HTML e se l'API esterna è disponibile, verranno visualizzate anche le caratteristiche del browser.

Quando viene inviato l'evento `creationComplete` del file dell'applicazione principale, viene richiamato il metodo `initApp()`. Il metodo `initApp()` chiama il metodo `getCapabilities()` dall'interno della classe com.example.programmingas3.capabilities.CapabilitiesGrabber. Di seguito è riportato il codice per il metodo `initApp()`:

```
private function initApp():void
{
  var dp:Array = CapabilitiesGrabber.getCapabilities();
  capabilitiesGrid.dataProvider = dp;
}
```

Il metodo `CapabilitiesGrabber.getCapabilities()` restituisce un array ordinato delle caratteristiche di Flash Player e del browser, che viene quindi impostato sulla proprietà `dataProvider` dell'istanza del componente DataGrid `capabilitiesGrid` sullo stage.

Panoramica della classe CapabilitiesGrabber

Il metodo statico `getCapabilities()` della classe CapabilitiesGrabber aggiunge ogni proprietà della classe flash.system.Capabilities a un array (`capDP`). Quindi, chiama il metodo statico `getBrowserObjects()` nella classe CapabilitiesGrabber. Il metodo `getBrowserObjects()` utilizza l'API esterna per eseguire elaborazioni cicliche dell'oggetto navigator del browser, che contiene le caratteristiche del browser. Di seguito è riportato il metodo `getCapabilities()`:

```
public static function getCapabilities():Array
{
  var capDP:Array = new Array();
  capDP.push({name:"Capabilities.avHardwareDisable",
  value:Capabilities.avHardwareDisable});
  capDP.push({name:"Capabilities.hasAccessibility",
  value:Capabilities.hasAccessibility});
  capDP.push({name:"Capabilities.hasAudio", value:Capabilities.hasAudio});
  ...
  capDP.push({name:"Capabilities.version", value:Capabilities.version});
```

```
  var navArr:Array = CapabilitiesGrabber.getBrowserObjects();
  if (navArr.length > 0)
  {
    capDP = capDP.concat(navArr);
  }
  capDP.sortOn("name", Array.CASEINSENSITIVE);
  return capDP;
}
```

Il metodo `getBrowserObjects()` restituisce un array di ogni proprietà dell'oggetto navigator del browser. Se questo array ha una lunghezza di uno o più elementi, l'array delle caratteristiche del browser (`navArr`) viene aggiunto all'array delle caratteristiche di Flash Player (`capDP`), quindi l'intero array viene ordinato alfabeticamente. Infine, l'array ordinato viene restituito al file dell'applicazione principale, che quindi completa la griglia di dati.
Di seguito è riportato il codice per il metodo `getBrowserObjects()`:

```
private static function getBrowserObjects():Array
{
  var itemArr:Array = new Array();
  var itemVars:URLVariables;
  if (ExternalInterface.available)
  {
    try
    {
      var tempStr:String = ExternalInterface.call("JS_getBrowserObjects");
      itemVars = new URLVariables(tempStr);
      for (var i:String in itemVars)
      {
        itemArr.push({name:i, value:itemVars[i]});
      }
    }
    catch (error:SecurityError)
    {
      // ignore
    }
  }
  return itemArr;
}
```

Se nell'ambiente utente corrente è disponibile l'API esterna, Flash Player chiama il metodo JavaScript `JS_getBrowserObjects()`, che esegue delle elaborazioni cicliche dell'oggetto navigator del browser e restituisce ad ActionScript una stringa di valori con codifica URL. Questa stringa viene quindi convertita in un oggetto URLVariables (`itemVars`) e aggiunta all'array `itemArr`, che viene restituito allo script chiamante.

Comunicazione con JavaScript

La fase finale nella creazione dell'applicazione CapabilitiesExplorer consiste nello scrivere il codice JavaScript necessario per eseguire elaborazioni cicliche di ogni elemento contenuto nell'oggetto navigator del browser e nell'aggiungere una coppia nome-valore a un array temporaneo. Di seguito è riportato il codice per il metodo JavaScript `JS_getBrowserObjects()` nel file container.html:

```
<script language="JavaScript">
  function JS_getBrowserObjects()
  {
    // Crea un array che contiene ognuno degli elementi del browser.
    var tempArr = new Array();

    // Esegue delle elaborazioni cicliche di ciascun elemento nell'oggetto
    // navigator del browser.
    for (var name in navigator)
    {
      var value = navigator[name];

      // Se il valore corrente è una stringa o un oggetto Boolean,
      // lo aggiunge all'array; altrimenti ignora l'elemento.
      switch (typeof(value))
      {
        case "string":
        case "boolean":

          // Crea una stringa temporanea che verrà aggiunta all'array.
          // Verifica che ai valori sia applicata la codifica URL
          // utilizzando la funzione escape() di JavaScript.
          var tempStr = "navigator." + name + "=" + escape(value);
          //  Inserisce la coppia nome-valore con codifica URL nell'array.
          tempArr.push(tempStr);
          break;
      }
    }
    // Esegue elaborazioni cicliche di ciascun elemento nell'oggetto screen
    // del browser.
    for (var name in screen)
    {
      var value = screen[name];
```

```
        // Se il valore corrente è un numero, lo aggiunge all'array;
        // altrimenti ignora l'elemento.
        switch (typeof(value))
        {
          case "number":
            var tempStr = "screen." + name + "=" + escape(value);
            tempArr.push(tempStr);
            break;
        }
      }
      // Restituisce l'array come stringa con codifica URL delle coppie
      // nome-valore.
      return tempArr.join("&");
    }
</script>
```

Il codice inizia con la creazione di un array temporaneo che includerà tutte le coppie nome-valore nell'oggetto navigator. Quindi, viene eseguita l'elaborazione ciclica dell'oggetto navigator utilizzando un ciclo `for..in` e viene valutato il tipo di dati del valore corrente per filtrare i valori indesiderati. In questa applicazione vengono utilizzati solo i valori di tipo stringa o booleano, mentre gli altri tipi di dati (ad esempio, funzioni o array) vengono ignorati. Ogni valore stringa o booleano nell'oggetto navigator viene aggiunto all'array `tempArr`. Quindi, viene eseguita l'elaborazione ciclica dell'oggetto screen utilizzando un ciclo `for..in` e ogni valore numerico viene aggiunto all'array `tempArr`. Infine, l'array temporaneo viene convertito in una stringa utilizzando il metodo `Array.join()`. L'array utilizza la e commerciale (&) come delimitatore, consentendo ad ActionScript di analizzare facilmente i dati mediante la classe URLVariables.

CAPITOLO 24

Stampa

24

Adobe® Flash® Player 9 è in grado di comunicare con l'interfaccia di stampa di un sistema operativo per inviare le pagine allo spooler di stampa. Ogni pagina inviata da Flash Player allo spooler può includere contenuto visibile, dinamico o fuori schermo, inclusi i valori di database e il testo dinamico. Inoltre, Flash Player imposta le proprietà della classe flash.printing.PrintJob in base alle impostazioni della stampante dell'utente, in modo da consentire la formattazione corretta delle pagine.

In questo capitolo vengono descritte in dettaglio le strategie per l'utilizzo dei metodi e delle proprietà della classe flash.printing.PrintJob per creare un lavoro di stampa, leggere le impostazioni di stampa di un utente e apportare modifiche a un lavoro di stampa in base al feedback di Flash Player e del sistema operativo dell'utente.

Sommario

Elementi fondamentali della stampa

Introduzione alla stampa

In ActionScript 3.0, si utilizza la classe PrintJob per creare delle istantanee del contenuto visualizzato per convertirlo in una rappresentazione stampabile su carta. Per alcuni versi, l'impostazione di contenuti per la stampa è uguale all'impostazione di contenuti per la visualizzazione su schermo: si posizionano e dimensionano degli elementi per creare il layout desiderato. Tuttavia, la stampa presenta alcune idiosincrasie che la rendono diversa dal layout su schermo. Ad esempio, le stampanti utilizzano una risoluzione diversa rispetto ai monitor dei computer; il contenuto di una schermata è dinamico e può cambiare, mentre quello stampato è intrinsecamente statico; infine, quando si pianifica la stampa, è necessario considerare le limitazioni del formato della pagina e la possibilità di stampare su più pagine.

Anche se queste differenze possono sembrare ovvie, è importante tenerle in considerazione quando si imposta la stampa con ActionScript. Dal momento che la precisione della stampa dipende da una combinazione dei valori specificati in fase di creazione e dalle caratteristiche della stampante dell'utente, la classe PrintJob include delle proprietà che consentono di determinare le importanti caratteristiche che è necessario considerare.

Operazioni comuni con la stampa

Le seguenti operazioni comuni con la stampa sono descritte in questo capitolo:

- Avvio di un lavoro di stampa
- Aggiunta di pagine a un lavoro di stampa
- Determinazione dell'annullamento di un lavoro di stampa da parte dell'utente
- Possibilità di specificare se utilizzare il rendering bitmap o vettoriale
- Impostazione delle dimensioni, della scala e dell'orientamento della pagina
- Specificazione dell'area stampabile del contenuto
- Conversione delle dimensioni dello schermo nel formato della pagina
- Stampa di lavori di stampa con più pagine

Concetti e termini importanti

L'elenco di riferimento seguente contiene dei termini importanti che vengono citati in questo capitolo:

- Spooler: una porzione del sistema operativo o del driver della stampante che tiene traccia delle pagine mentre sono in attesa di essere stampate e le invia alla stampante quando quest'ultima si rende disponibile.
- Orientamento della pagina: la rotazione del contenuto stampato rispetto alla carta (orizzontale o verticale).
- Lavoro di stampa: la pagina o la serie di pagine che compongono un'unica stampa.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. In molti casi, si tratta di piccole porzioni di codice, invece di esempi completamente funzionanti di stampa o che controllano valori. La prova degli esempi prevede la creazione di elementi da stampare e l'uso di esempi di codice con questi elementi. Gli ultimi due esempi del capitolo sono esempi completi di stampa, che includono il codice che definisce il contenuto da stampare e quello che esegue le operazioni di stampa.

Per provare gli esempi di codice:

1. Creare un nuovo documento Flash.
2. Selezionare il fotogramma chiave nel fotogramma 1 della linea temporale e aprire il pannello Azioni.
3. Copiare l'esempio di codice nel riquadro dello script.
4. Dal menu principale, selezionare Controllo > Prova filmato per creare un file SWF e provare l'esempio.

Stampa di una pagina

Utilizzare un'istanza della classe PrintJob per gestire la stampa. Per stampare una pagina di base mediante Flash Player sono richieste quattro istruzioni principali in sequenza:

- `new PrintJob()`: crea una nuova istanza del lavoro di stampa con il nome specificato.
- `PrintJob.start()`: avvia il processo di stampa per il sistema operativo, richiamando la finestra di dialogo di stampa per l'utente, e compila le proprietà di sola lettura del lavoro di stampa.
- `PrintJob.addPage()`: contiene i dettagli del contenuto del lavoro di stampa, inclusi l'oggetto Sprite (e gli eventuali oggetti secondari in esso contenuti), le dimensioni dell'area di stampa e la modalità di stampa dell'immagine sulla stampante, ovvero grafica vettoriale o bitmap. È possibile utilizzare chiamate successive ad `addPage()` per stampare più sprite su diverse pagine.
- `PrintJob.send()`: invia le pagine alla stampante del sistema operativo.

Lo script per un lavoro di stampa molto semplice, ad esempio, potrebbe avere un aspetto simile al seguente (incluse le istruzioni `package`, `import` e `class` per la compilazione):

```
package
{
  import flash.printing.PrintJob;
  import flash.display.Sprite;

  public class BasicPrintExample extends Sprite
  {
    var myPrintJob:PrintJob = new PrintJob();
    var mySprite:Sprite = new Sprite();

    public function BasicPrintExample()
    {
      myPrintJob.start();
      myPrintJob.addPage(mySprite);
      myPrintJob.send();
    }
  }
}
```

NOTA Questo esempio è destinato a illustrare gli elementi di base di uno script per un lavoro di stampa e non contiene codice per la gestione degli errori. Per creare uno script che risponda correttamente all'operazione di annullamento di un lavoro di stampa, vedere "Operazioni relative alle eccezioni e ai valori restituiti" a pagina 765.

Se, per qualsiasi motivo, è necessario cancellare le proprietà di un oggetto PrintJob, impostare la variabile PrintJob su `null` (come in `myPrintJob = null`).

Attività di Flash Player e interfaccia di stampa del sistema operativo

Poiché Flash Player invia le pagine all'interfaccia di stampa del sistema operativo, è opportuno comprendere l'ambito delle attività gestite da Flash Player e quello delle attività gestite dall'interfaccia di stampa del sistema operativo. Flash Player può avviare un lavoro di stampa, leggere alcune delle impostazioni della pagina di una stampante, passare il contenuto per un lavoro di stampa al sistema operativo e verificare se l'utente o il sistema ha annullato un lavoro di stampa. Tutti gli altri processi, ad esempio la visualizzazione delle finestre di dialogo specifiche della stampante, l'annullamento di un lavoro di stampa elaborato dallo spooler o la segnalazione dello stato della stampante, vengono gestiti dal sistema operativo. Flash Player è in grado di rispondere se si verificano problemi all'avvio o durante la formattazione di un lavoro di stampa, ma può fornire una segnalazione solo su determinate proprietà o condizioni dall'interfaccia di stampa del sistema operativo. Gli sviluppatori dovranno fare in modo che il codice abbia la capacità di rispondere a queste proprietà o condizioni.

Operazioni relative alle eccezioni e ai valori restituiti

È opportuno verificare se il metodo `PrintJob.start()` restituisce `true` prima di eseguire le chiamate a `addPage()` e `send()`, qualora l'utente abbia annullato il lavoro di stampa. Un metodo semplice per verificare se questi metodi sono stati annullati, prima di continuare, consiste nell'inserirli in un'istruzione `if`, come riportato di seguito:

```
if (myPrintJob.start())
{
  // Inserire qui le istruzioni addPage() e send()
}
```

Se `PrintJob.start()` è `true`, cioè se l'utente ha selezionato Stampa (o Flash Player ha avviato il comando Stampa), possono essere chiamati i metodi `addPage()` e `send()`.

Inoltre, per facilitare la gestione del processo di stampa, Flash Player genera delle eccezioni per il metodo `PrintJob.addPage()`, così che sia possibile rilevare gli errori e presentare le informazioni e le opzioni all'utente. Se un metodo `PrintJob.addPage()` ha esito negativo, è possibile chiamare un'altra funzione oppure interrompere il lavoro di stampa corrente. Per rilevare queste eccezioni, incorporare le chiamate ad `addPage()` in un'istruzione `try..catch`, come nell'esempio riportato di seguito. Nell'esempio, `[params]` è un segnaposto per i parametri che specificano il contenuto effettivo da stampare.

```
if (myPrintJob.start())
{
  try
  {
    myPrintJob.addPage([params]);
  }
  catch (error:Error)
  {
    // Gestisce l'errore,
  }
  myPrintJob.send();
}
```

Una volta avviato il lavoro di stampa, è possibile aggiungere il contenuto utilizzando `PrintJob.addPage()` e verificare se viene generata un'eccezione (ad esempio, se l'utente ha annullato il lavoro di stampa). In caso contrario, è possibile aggiungere logica all'istruzione `catch` per fornire all'utente (o a Flash Player) le informazioni e le opzioni oppure è possibile interrompere il lavoro di stampa corrente. Se la pagina viene aggiunta correttamente, si può procedere all'invio delle pagine alla stampante utilizzando `PrintJob.send()`.

Se durante l'invio del lavoro di stampa alla stampante si verifica un problema (ad esempio, la stampante non è in linea), è possibile rilevare anche quell'eccezione e fornire all'utente (o a Flash Player) le informazioni o ulteriori opzioni (quale la visualizzazione del testo di un messaggio o la presentazione di un avviso nell'animazione di Flash). È possibile, ad esempio, assegnare nuovo testo a un campo di testo in un'istruzione `if..else`, come illustrato nel codice riportato di seguito:

```
if (myPrintJob.start())
{
  try
  {
    myPrintJob.addPage([params]);
  }
  catch (error:Error)
  {
    // Gestisce l'errore.
  }
  myPrintJob.send();
}
else
{
  myAlert.text = "Print job canceled";
}
```

Per un esempio funzionante, vedere "Esempio: Modifica in scala, ritaglio e risposta" a pagina 773.

Operazioni con le proprietà della pagina

Dopo che l'utente ha fatto clic su OK nella finestra di dialogo di stampa e `PrintJob.start()` ha restituito `true`, è possibile accedere alle proprietà definite nelle impostazioni della stampante, che includono la larghezza e l'altezza del foglio (`pageHeight` e `pageWidth`) e l'orientamento del contenuto sul foglio. Poiché queste sono impostazioni della stampante, che non vengono controllate da Flash Player, non è possibile modificarle; tuttavia, possono essere utilizzate per allineare il contenuto inviato alla stampante, in modo che corrisponda alle impostazioni correnti. Per ulteriori informazioni, vedere "Impostazione delle dimensioni, della scala e dell'orientamento" a pagina 768.

Impostazione del rendering vettoriale o bitmap

È possibile impostare manualmente il lavoro di stampa in modo che ogni pagina sia inviata allo spooler come grafica vettoriale o immagine bitmap. In alcuni casi, la stampa vettoriale produce file di spool di dimensioni inferiori e un'immagine migliore rispetto alla stampa bitmap. Se tuttavia nel contenuto è presente un'immagine bitmap e si desidera mantenere gli eventuali effetti di colore o di trasparenza alfa, stampare la pagina come immagine bitmap. Le stampanti non PostScript, inoltre, convertono automaticamente la grafica vettoriale in immagini bitmap. Specificare la stampa bitmap nel terzo parametro di `PrintJob.addPage()`,passando un oggetto PrintJobOptions con il parametro `printAsBitmap` impostato su `true`, come riportato di seguito:

```
var options:PrintJobOptions = new PrintJobOptions();
options.printAsBitmap = true;
myPrintJob.addPage(mySprite, null, options);
```

Se non si specifica un valore per il terzo parametro, il lavoro di stampa utilizza l'impostazione predefinita, ovvero la stampa vettoriale.

NOTA

Se non si desidera specificare un valore per `printArea` (il secondo parametro) ma un valore per la stampa bitmap, utilizzare `null` per `printArea`.

Tempistica delle istruzioni per i lavori di stampa

ActionScript 3.0 non limita un oggetto PrintJob a un fotogramma singolo (come accadeva nelle versioni precedenti di ActionScript). Tuttavia, poiché le informazioni sullo stato di stampa vengono visualizzate dopo aver fatto clic sul pulsante OK nella finestra di dialogo di stampa, è necessario chiamare `PrintJob.addPage()` e `PrintJob.send()` il più presto possibile per inviare le pagine allo spooler. Se il fotogramma contenente la chiamata `PrintJob.send()` viene raggiunto in ritardo, il processo di stampa viene ritardato.

In ActionScript 3.0 è presente un limite di 15 secondi al timeout dello script, pertanto l'intervallo tra ogni istruzione principale in una sequenza del lavoro di stampa non può superare 15 secondi. In altre parole, il limite di timeout dello script di 15 secondi si applica ai seguenti intervalli:

- Tra `PrintJob.start()` e il primo `PrintJob.addPage()`
- Tra `PrintJob.addPage()` e il successivo `PrintJob.addPage()`
- Tra l'ultimo `PrintJob.addPage()` e `PrintJob.send()`

Se uno di questi intervalli supera il limite di 15 secondi, la successiva chiamata a `PrintJob.start()` nell'istanza di PrintJob restituisce `false` e la successiva chiamata a `PrintJob.addPage()` nell'istanza di PrintJob causa la generazione di un'eccezione di runtime da parte di Flash Player.

Impostazione delle dimensioni, della scala e dell'orientamento

Nella sezione "Stampa di una pagina" a pagina 764 è descritta in dettaglio la procedura per un lavoro di stampa di base, in cui l'output riflette direttamente l'equivalente su carta delle dimensioni dello schermo e della posizione dello sprite specificato. Le stampanti utilizzano tuttavia risoluzioni diverse per la stampa e possono avere impostazioni che influiscono negativamente sull'aspetto dello sprite stampato.

Flash Player è in grado di leggere le impostazioni della stampante del sistema operativo, ma queste proprietà sono di sola lettura: è possibile rispondere ai relativi valori, ma non impostarli. Quindi è possibile, ad esempio, verificare l'impostazione del formato carta della stampante e adattare il contenuto al formato e, inoltre, determinare le impostazioni dei margini e l'orientamento della pagina della stampante. Per rispondere alle impostazioni della stampante, può essere necessario specificare un'area di stampa, regolare la differenza tra la risoluzione dello schermo e le misure in punti della stampante oppure trasformare il contenuto affinché corrisponda alle impostazioni relative alle dimensioni e all'orientamento della stampante in uso.

Utilizzo di rettangoli per l'area di stampa

Il metodo `PrintJob.addPage()` consente di specificare l'area di uno sprite che si desidera stampare. Il secondo parametro, `printArea`, è rappresentato da un oggetto Rectangle. Sono disponibili tre opzioni per fornire un valore a questo parametro:

- Creare un oggetto Rectangle con proprietà specifiche e quindi utilizzare il rettangolo nella chiamata ad `addPage()`, come illustrato nell'esempio riportato di seguito:

```
private var rect1:Rectangle = new Rectangle(0, 0, 400, 200);
myPrintJob.addPage(sheet, rect1);
```

- Se non è già stato specificato un oggetto Rectangle, è possibile effettuare questa operazione all'interno della chiamata, come illustrato nell'esempio riportato di seguito:

```
myPrintJob.addPage(sheet, new Rectangle(0, 0, 100, 100));
```

- Se si prevede di specificare dei valori per il terzo parametro nella chiamata a `addPage()`, ma senza specificare un rettangolo, è possibile utilizzare `null` come secondo parametro, come riportato di seguito:

```
myPrintJob.addPage(sheet, null, options);
```

NOTA

Se si prevede di specificare un rettangolo per le dimensioni di stampa, importare la classe `flash.display.Rectangle`.

Confronto tra punti e pixel

La larghezza e l'altezza del rettangolo sono espresse in pixel. L'unità di misura di stampa utilizzata da una stampante è il punto. I punti hanno dimensioni fisiche fisse (1/72 di pollice), mentre le dimensioni di un pixel dipendono dalla risoluzione specifica dello schermo. Il tasso di conversione tra pixel e punti varia a seconda delle impostazioni della stampante e se lo sprite è stato modificato in scala. Uno sprite non modificato in scala largo 72 pixel verrà stampato con una larghezza di un pollice, con un punto equivalente a un pixel, indipendentemente dalla risoluzione dello schermo.

È possibile utilizzare le seguenti equivalenze per convertire pollici o centimetri in twip o punti (un twip corrisponde a 1/20 di punto):

- 1 punto = 1/72 di pollice = 20 twip
- 1 pollice = 72 punti = 1440 twip
- 1 centimetro = 567 twip

Se si omette il parametro `printArea` oppure se questo viene passato in modo non corretto, viene stampata l'intera area dello sprite.

Modifica in scala

Per modificare in scala un oggetto Sprite prima della stampa, impostarne le proprietà (vedere "Manipolazione delle dimensioni e modifica in scala degli oggetti" a pagina 430) prima di chiamare il metodo `PrintJob.addPage()` e impostarle di nuovo sui rispettivi valori originali dopo la stampa. La scala di un oggetto Sprite non ha alcuna relazione con la proprietà `printArea`. In altre parole, se si specifica un'area di stampa di 50 x 50 pixel, vengono stampati 2500 pixel. Se l'oggetto Sprite è stato modificato in scala, vengono stampati gli stessi 2500 pixel, ma l'oggetto viene stampato nelle dimensioni modificate in scala.

Per un esempio, vedere "Esempio: Modifica in scala, ritaglio e risposta" a pagina 773.

Stampa con orientamento verticale o orizzontale

Poiché Flash Player è in grado di rilevare le impostazioni relative all'orientamento, è possibile inserire nel codice ActionScript la logica necessaria per regolare le dimensioni o la rotazione del contenuto in risposta alle impostazioni della stampante, come illustrato nell'esempio riportato di seguito:

```
if (myPrintJob.orientation == PrintJobOrientation.LANDSCAPE)
{
  mySprite.rotation = 90;
}
```

NOTA

Se si prevede di leggere l'impostazione del sistema relativa all'orientamento del contenuto sul foglio, impostare la classe PrintJobOrientation come illustrato di seguito:

```
import flash.printing.PrintJobOrientation;
```

La classe PrintJobOrientation fornisce i valori costanti che definiscono l'orientamento del contenuto nella pagina.

Risposta ai valori di altezza e larghezza della pagina

Utilizzando una strategia analoga alla gestione delle impostazioni di orientamento della stampante, è possibile leggere le impostazioni relative all'altezza e alla larghezza della pagina e rispondere incorporando la logica in un'istruzione `if`. Nel codice seguente viene illustrato un esempio:

```
if (mySprite.height > myPrintJob.pageHeight)
{
  mySprite.scaleY = .75;
}
```

Inoltre, è possibile determinare le impostazioni dei margini della pagina confrontando le dimensioni della pagina e del foglio, come nell'esempio riportato di seguito:

```
margin_height = (myPrintJob.paperHeight - myPrintJob.pageHeight) / 2;
margin_width = (myPrintJob.paperWidth - myPrintJob.pageWidth) / 2;
```

Esempio: Stampa su più pagine

Quando si stampa più di una pagina di contenuto, è possibile associare ogni pagina a un diverso sprite (in questo caso, `sheet1` e `sheet2`), quindi utilizzare `PrintJob.addPage()` per ogni sprite. Il codice riportato di seguito illustra questa tecnica:

```
package
{
  import flash.display.MovieClip;
  import flash.printing.PrintJob;
  import flash.printing.PrintJobOrientation;
  import flash.display.Stage;
  import flash.display.Sprite;
  import flash.text.TextField;
  import flash.geom.Rectangle;

  public class PrintMultiplePages extends MovieClip
  {
    private var sheet1:Sprite;
    private var sheet2:Sprite;

    public function PrintMultiplePages():void
    {
      init();
      printPages();
    }

    private function init():void
    {
      sheet1 = new Sprite();
      createSheet(sheet1, "Once upon a time...", {x:10, y:50, width:80,
      height:130});
      sheet2 = new Sprite();
      createSheet(sheet2, "There was a great story to tell, and it ended
      quickly.\n\nThe end.", null);
    }

    private function createSheet(sheet:Sprite, str:String,
    imgValue:Object):void
    {
      sheet.graphics.beginFill(0xEEEEEE);
      sheet.graphics.lineStyle(1, 0x000000);
      sheet.graphics.drawRect(0, 0, 100, 200);
```

```
    sheet.graphics.endFill();

    var txt:TextField = new TextField();
    txt.height = 200;
    txt.width = 100;
    txt.wordWrap = true;
    txt.text = str;

    if (imgValue != null)
    {
      var img:Sprite = new Sprite();
      img.graphics.beginFill(0xFFFFFF);
      img.graphics.drawRect(imgValue.x, imgValue.y, imgValue.width,
      imgValue.height);
      img.graphics.endFill();
      sheet.addChild(img);
    }
    sheet.addChild(txt);
  }

  private function printPages():void
  {
    var pj:PrintJob = new PrintJob();
    var pagesToPrint:uint = 0;
    if (pj.start())
    {
      if (pj.orientation == PrintJobOrientation.LANDSCAPE)
      {
        throw new Error("Page is not set to an orientation of
        portrait.");
      }

      sheet1.height = pj.pageHeight;
      sheet1.width = pj.pageWidth;
      sheet2.height = pj.pageHeight;
      sheet2.width = pj.pageWidth;

      try
      {
        pj.addPage(sheet1);
        pagesToPrint++;
      }
      catch (error:Error)
      {
      // Risponde all'errore.
      }
```

```
            try
            {
              pj.addPage(sheet2);
              pagesToPrint++;
            }
            catch (error:Error)
            {
              // Risponde all'errore.
            }

            if (pagesToPrint > 0)
            {
              pj.send();
            }
          }
        }
    }
}
```

Esempio: Modifica in scala, ritaglio e risposta

In alcuni casi, può essere utile modificare le dimensioni (o altre proprietà) di un oggetto di visualizzazione in fase di stampa per conciliare le differenze tra il modo in cui viene visualizzato sullo schermo e il modo in cui appare sul foglio stampato. Quando si modificano le proprietà di un oggetto di visualizzazione prima della stampa (ad esempio, utilizzando le proprietà `scaleX` e `scaleY`), si tenga presente che se le dimensioni dell'oggetto modificate in scala sono superiori a quelle del rettangolo definito per l'area di stampa, l'oggetto verrà ritagliato. Una volta stampate le pagine, è inoltre consigliabile reimpostare le proprietà.

Il codice riportato di seguito modifica in scala le dimensioni dell'oggetto di visualizzazione `txt` (ma non lo sfondo della casella verde) e di conseguenza il campo di testo viene ritagliato in base alle dimensioni del rettangolo specificato. Dopo la stampa, vengono ripristinate le dimensioni originali del campo di testo per la visualizzazione sullo schermo. Se il lavoro di stampa viene annullato dalla finestra di dialogo di stampa del sistema operativo, il contenuto in Flash Player viene modificato per informare l'utente che il lavoro è stato annullato.

```
package
{
  import flash.printing.PrintJob;
  import flash.display.Sprite;
  import flash.text.TextField;
  import flash.display.Stage;
  import flash.geom.Rectangle;
```

```
public class PrintScaleExample extends Sprite
{
  private var bg:Sprite;
  private var txt:TextField;

  public function PrintScaleExample():void
  {
    init();
    draw();
    printPage();
  }

  private function printPage():void
  {
    var pj:PrintJob = new PrintJob();
    txt.scaleX = 3;
    txt.scaleY = 2;
    if (pj.start())
    {
      trace(">> pj.orientation: " + pj.orientation);
      trace(">> pj.pageWidth: " + pj.pageWidth);
      trace(">> pj.pageHeight: " + pj.pageHeight);
      trace(">> pj.paperWidth: " + pj.paperWidth);
      trace(">> pj.paperHeight: " + pj.paperHeight);

      try
      {
        pj.addPage(this, new Rectangle(0, 0, 100, 100));
      }
      catch (error:Error)
      {
        // Non esegue alcuna azione.
      }
      pj.send();
    }
    else
    {
      txt.text = "Print job canceled";
    }
    // Reimposta le proprietà per la modifica in scala di txt.
    txt.scaleX = 1;
    txt.scaleY = 1;
  }

  private function init():void
  {
    bg = new Sprite();
    bg.graphics.beginFill(0x00FF00);
    bg.graphics.drawRect(0, 0, 100, 200);
    bg.graphics.endFill();
```

```
        txt = new TextField();
        txt.border = true;
        txt.text = "Hello World";
      }

      private function draw():void
      {
        addChild(bg);
        addChild(txt);
        txt.x = 50;
        txt.y = 50;
      }
    }
}
```

CAPITOLO 25

Uso dell'API esterna

25

L'API esterna di ActionScript 3.0 consente una comunicazione diretta tra ActionScript e l'applicazione contenitore in cui è in esecuzione Adobe Flash Player 9. Esistono diverse situazioni in cui è consigliabile utilizzarla; ad esempio quando si crea un'interazione tra un documento SWF e JavaScript in una pagina HTML o quando si crea un'applicazione desktop che utilizza Flash Player per visualizzare un file SWF.

Questo capitolo descrive come utilizzare l'API esterna per interagire con un'applicazione contenitore, come passare dei dati tra ActionScript e JavaScript in una pagina HTML e come stabilire la comunicazione e scambiare dati tra ActionScript e un'applicazione desktop.

NOTA	Questo capitolo si concentra solo sulla comunicazione tra ActionScript in un file SWF e l'applicazione contenitore che include un riferimento all'istanza di Flash Player in cui l'SWF è caricato. Qualsiasi altro uso di Flash Player all'interno di un'applicazione non è contemplato da questa documentazione. Flash Player è progettato per essere utilizzato come plug-in per browser o come proiettore (applicazione autonoma). Altri scenari di impiego potrebbero avere un supporto limitato.

Sommario

Nozioni fondamentali sull'uso dell'API esterna

Introduzione all'uso dell'API esterna

Benché in alcune circostanze un file SWF possa essere eseguito in modo indipendente (ad esempio, creando un proiettore SWF), nella maggior parte dei casi un'applicazione SWF viene eseguita come elemento in un'altra applicazione. Di solito, il contenitore in cui è incorporato il file SWF è un file HTML; meno frequentemente, un file SWF viene utilizzato integralmente o parzialmente come interfaccia utente di un'applicazione desktop.

Quando si lavora su applicazioni più avanzate, può essere necessario impostare la comunicazione tra un file SWF e l'applicazione contenitore. Ad esempio, è abbastanza comune che una pagina Web visualizzi testo o altre informazioni in HTML e includa un file SWF per visualizzare del contenuto visivo dinamico come un grafico o un video. In tali circostanze, è possibile fare in modo che, quando gli utenti fanno clic su un pulsante presente sulla pagina Web, qualcosa cambi nel file SWF. ActionScript contiene un meccanismo, definito API esterna, che facilita questo tipo di comunicazione tra il codice ActionScript presente in un file SWF e altro codice presente nell'applicazione contenitore.

Operazioni comuni con l'API esterna

Le seguenti operazioni comuni con l'API esterna sono descritte in questo capitolo:

- Accesso alle informazioni relative all'applicazione contenitore
- Uso di ActionScript per chiamare il codice in un'applicazione contenitore, quale una pagina Web o un'applicazione desktop
- Chiamate al codice ActionScript dal codice presente in un'applicazione contenitore
- Creazione di un proxy per semplificare le chiamate al codice ActionScript da un'applicazione contenitore

Concetti e termini importanti

L'elenco di riferimento seguente contiene dei termini importanti che vengono citati in questo capitolo:

- Contenitore ActiveX: un'applicazione contenitore (non un browser Web) che include un'istanza del controllo ActiveX di Flash Player per visualizzare il contenuto SWF nell'applicazione.
- Applicazione contenitore: l'applicazione al cui interno Flash Player esegue un file SWF (ad esempio, un browser Web e la pagina HTML che include i contenuti di Flash Player).
- Proiettore: un file SWF che è stato convertito in un file eseguibile autonomo che include sia Flash Player che il contenuto del file SWF. Un proiettore può essere creato in Adobe Flash CS3 oppure mediante la versione autonoma di Flash Player. I proiettori vengono generalmente utilizzati per distribuire i file SWF su CD-ROM o in situazioni simili in cui le dimensioni di scaricamento non sono un problema e l'autore del file SWF vuole essere certo che l'utente sia in grado di eseguire il file SWF, indipendentemente dal fatto che abbia installato Flash Player sul computer.
- Proxy: un'applicazione o un codice intermedio che chiama il codice in un'applicazione (definita "applicazione esterna") per conto di un'altra applicazione (definita "applicazione chiamante") e restituisce i valori all'applicazione chiamante. Un proxy può essere utilizzato tra l'altro per:
  - Semplificare il processo di esecuzione delle chiamate a funzioni esterne convertendo delle chiamate a funzioni native nell'applicazione chiamante nel formato interpretato dall'applicazione esterna
  - Superare le limitazioni di sicurezza e di altro tipo per evitare che il chiamante comunichi direttamente con l'applicazione esterna
- Serializzare: convertire i valori di oggetti o dati in un formato utilizzabile per passare i valori nei messaggi scambiati tra due sistemi di programmazione, ad esempio su Internet o tra due applicazioni diverse in esecuzione sullo stesso computer.

Operazioni con gli esempi contenuti nel capitolo

È consigliabile provare gli esempi di codice presenti in questo capitolo. In molti casi, si tratta di brevi codici a scopo puramente dimostrativo, invece di esempi completamente funzionanti o che controllano valori. Poiché l'uso di un'API esterna richiede (per definizione) la scrittura di codice ActionScript e del codice in un'applicazione contenitore, la prova degli esempi prevede la creazione di un contenitore (ad esempio, una pagina Web contenente il file SWF) e l'uso degli esempi di codice per interagire con il contenitore.

Per provare un esempio di comunicazione tra ActionScript e JavaScript:

1. Creare un nuovo documento Flash e salvarlo nel computer.
2. Dal menu principale, scegliere File > Impostazioni pubblicazione.
3. Nella finestra di dialogo Impostazioni pubblicazione, selezionare la scheda Formati e verificare che le caselle di controllo HTML e Flash siano selezionate.
4. Fare clic sul pulsante Pubblica. In questo modo, vengono generati un file SWF e un file HTML nella stessa cartella e con lo stesso nome utilizzato per salvare il documento Flash. Fare clic su OK per chiudere la finestra di dialogo Impostazioni pubblicazione.
5. Deselezionare la casella di controllo HTML. Ora che la pagina HTML è stata generata, occorre modificarla e aggiungervi il codice JavaScript appropriato. La deselezione della casella di controllo HTML garantisce che in seguito alla modifica della pagina HTML, Flash non sovrascriva le modifiche con una nuova pagina HTML al momento della pubblicazione del file SWF.
6. Fare clic su OK per chiudere la finestra di dialogo Impostazioni pubblicazione.
7. Mediante un'applicazione HTML o editor di testo, aprire il file HTML creato da Flash al momento della pubblicazione del file SWF. Nel codice di origine HTML, aggiungere un elemento script e copiare al suo interno il codice JavaScript contenuto nell'esempio di codice.
8. Salvare il file HTML e tornare a Flash.
9. Selezionare il fotogramma chiave nel fotogramma 1 della linea temporale e aprire il pannello Azioni.
10. Copiare l'esempio di codice ActionScript nel riquadro dello script.
11. Dal menu principale, scegliere File > Pubblica per aggiornare il file SWF con le modifiche effettuate.
12. Mediante un browser Web, aprire la pagina HTML modificata per visualizzarla e provarne la comunicazione con ActionScript.

Per provare un esempio di comunicazione tra ActionScript e un contenitore ActiveX:

1. Creare un nuovo documento Flash e salvarlo nel computer. È consigliabile salvarlo nella cartella in cui l'applicazione contenitore si aspetta di trovare il file SWF.
2. Dal menu principale, scegliere File > Impostazioni pubblicazione.
3. Nella finestra di dialogo Impostazioni pubblicazione, selezionare la scheda Formati e verificare che sia selezionata solo la casella di controllo Flash.

4. Nel campo di testo File accanto alla casella di controllo Flash, fare clic sull'icona della cartella per selezionare la cartella in cui pubblicare il file SWF. Quando si imposta la posizione del file SWF, è possibile (ad esempio) conservare il documento Flash in una cartella e collocare il file SWF pubblicato in un'altra, ad esempio nella cartella contenente il codice di origine dell'applicazione contenitore.
5. Selezionare il fotogramma chiave nel fotogramma 1 della linea temporale e aprire il pannello Azioni.
6. Copiare il codice ActionScript dell'esempio nel riquadro dello script.
7. Dal menu principale, scegliere File > Pubblica per pubblicare di nuovo il file SWF.
8. Creare ed eseguire l'applicazione contenitore per provare la comunicazione tra ActionScript e l'applicazione contenitore.

I due esempi alla fine di questo capitolo sono esempi completi che utilizzano l'API esterna per comunicare rispettivamente con una pagina HTML e un'applicazione desktop C#. Questi esempi sono costituiti da codici completi, che includono il codice di verifica degli errori del contenitore e ActionScript, necessari per la scrittura di codice che utilizza l'API esterna. Per un altro esempio di codice completo che utilizza l'API esterna, vedere l'esempio della classe ExternalInterface nella *Guida di riferimento del linguaggio e ai componenti ActionScript 3.0*.

Requisiti e vantaggi dell'API esterna

L'API esterna è una porzione di ActionScript che fornisce un meccanismo di comunicazione tra ActionScript e il codice in esecuzione su un'applicazione cosiddetta "esterna", ovvero un'applicazione che funge da contenitore per Flash Player (di solito, si tratta di un browser Web o di un'applicazione proiettore autonoma). In ActionScript 3.0, la funzionalità dell'API esterna è fornita dalla classe ExternalInterface. Nelle versioni di Flash Player precedenti a Flash Player 8, l'azione `fscommand()` veniva utilizzata per effettuare la comunicazione con l'applicazione contenitore. La classe ExternalInterface sostituisce `fscommand()` ed è consigliabile utilizzarla per tutte le comunicazioni tra JavaScript e ActionScript.

NOTA

Se è necessario utilizzare ancora `fscommand()` (ad esempio, per mantenere la compatibilità con applicazioni precedenti o per interagire con un'applicazione contenitore di terze parti o con la versione autonoma di Flash Player), la funzione è ancora disponibile come funzione a livello di pacchetto nel pacchetto flash.system.

La classe ExternalInterface è un sottosistema che consente facili comunicazioni da ActionScript e Flash Player a una pagina HTML contenente JavaScript o a un'applicazione desktop che incorpora un'istanza di Flash Player.

La classe ExternalInterface è disponibile solo nelle seguenti circostanze:

- In tutte le versioni supportate di Internet Explorer per Windows (5.0 e versioni successive)
- In un'applicazione contenitore, come un'applicazione desktop, che utilizza un'istanza del controllo ActiveX di Flash Player
- In tutti i browser che supportano l'interfaccia NPRuntime che comprende attualmente i seguenti browser:
  - Firefox 1.0 e versioni successive
  - Mozilla 1.7.5 e versioni successive
  - Netscape 8.0 e versioni successive
  - Safari 1.3 e versioni successive

In tutte le altre situazioni (ad esempio, nel caso di esecuzione in un lettore autonomo) in cui la proprietà `ExternalInterface.available` restituisce `false`.

Da ActionScript, è possibile chiamare una funzione JavaScript sulla pagina HTML. L'API esterna offre la seguente funzionalità perfezionata rispetto a `fscommand()`:

- È possibile utilizzare qualsiasi funzione JavaScript, non solo le funzioni utilizzabili con `fscommand()`.
- È possibile passare un numero qualsiasi di argomenti, con tutti i nomi; non ci sono limitazioni al passaggio di un comando e di un unico argomento String. In questo modo, l'API esterna risulta molto più flessibile rispetto a `fscommand()`.
- È possibile passare diversi tipi di dati (come Boolean, Number e String); non vi sono più limitazioni per i parametri String.
- È possibile ricevere il valore di una chiamata e restituire immediatamente tale valore ad ActionScript (come valore restituito della chiamata eseguita).

AVVERTENZA

Se il nome fornito all'istanza di Flash Player in una pagina HTML (l'attributo `id` del tag `object`) include un trattino (`-`) o altri caratteri che sono definiti come operatori in JavaScript (ad esempio, `+`, `*`, `/`, `\`, `.` e così via), le chiamate a ExternalInterface da ActionScript non funzionano quando la pagina Web contenitore viene visualizzata in Internet Explorer.
Inoltre, se i tag HTML che definiscono l'istanza di Flash Player (i tag `object` ed `embed`) sono nidificati in un tag `form` HTML, le chiamate ExternalInterface da ActionScript non funzionano.

Uso della classe ExternalInterface

La comunicazione tra ActionScript e l'applicazione contenitore può avere uno dei due formati seguenti: ActionScript è in grado di chiamare il codice (ad esempio, una funzione JavaScript) definito nel contenitore, oppure il codice nel contenitore può chiamare una funzione ActionScript che è stata progettata per essere chiamabile. In entrambi i casi, le informazioni possono essere inviate al codice chiamato e i risultati possono essere restituiti al codice che ha effettuato la chiamata.

Per facilitare questa comunicazione, la classe ExternalInterface include due proprietà statiche e due metodi statici. Queste proprietà e questi metodi servono per ottenere informazioni sulla connessione all'interfaccia esterna, per eseguire il codice nel contenitore da ActionScript e per rendere le funzioni ActionScript disponibili alle chiamate effettuate dal contenitore.

Accesso alle informazioni sul contenitore esterno

La proprietà `ExternalInterface.available` indica se la versione corrente di Flash Player si trova in un contenitore dotato di interfaccia esterna. Se l'interfaccia esterna è disponibile, questa proprietà è `true`; in caso contrario, è `false`. Prima di utilizzare qualunque altra funzionalità della classe ExternalInterface, verificare sempre che il contenitore corrente supporti la comunicazione con l'interfaccia esterna nel modo seguente:

```
if (ExternalInterface.available)
{
  // Esegue le chiamate al metodo ExternalInterface.
}
```

NOTA

La proprietà `ExternalInterface.available` riporta se il contenitore corrente è di un tipo che supporta la connettività ExternalInterface. Non indica se JavaScript è abilitato nel browser corrente.

La proprietà `ExternalInterface.objectID` consente di determinare l'identificatore univoco per l'istanza di Flash Player (nello specifico, l'attributo `id` del tag `object` in Internet Explorer o l'attributo `name` del tag `embed` nei browser che utilizzano l'interfaccia NPRuntime). Questo identificatore univoco rappresenta il documento SWF corrente nel browser e può essere utilizzato per fare riferimento al documento SWF (ad esempio, quando si chiama una funzione JavaScript in una pagina HTML contenitore). Quando il contenitore di Flash Player non è un browser Web, questa proprietà è `null`.

Chiamate al codice esterno da ActionScript

Il metodo `ExternalInterface.call()` esegue il codice nell'applicazione contenitore. Richiede almeno un parametro, ovvero una stringa che contiene il nome della funzione da chiamare nell'applicazione contenitore. Tutti gli eventuali parametri aggiuntivi passati al metodo `ExternalInterface.call()` vengono passati al contenitore come parametri della chiamata alla funzione.

```
// Chiama la funzione esterna "addNumbers"
// passando due parametri e assegnando il risultato della funzione
// alla variabile "result"
var param1:uint = 3;
var param2:uint = 7;
var result:uint = ExternalInterface.call("addNumbers", param1, param2);
```

Se il contenitore è una pagina HTML, questo metodo richiama la funzione JavaScript con il nome specificato, che deve essere definita in un elemento `script` presente nella pagina HTML contenitore. Il valore restituito dalla funzione JavaScript viene ripassato ad ActionScript.

```
<script language="JavaScript">
  // Aggiunge due numeri e reinvia il risultato ad ActionScript
  function addNumbers(num1, num2)
  {
    return (num1 + num2);
  }
</script>
```

Se il contenitore è un altro contenitore ActiveX, questo metodo fa in modo che il controllo ActiveX di Flash Player invii il proprio evento `FlashCall`. Il nome di funzione specificato e tutti gli eventuali parametri vengono serializzati in una stringa XML da Flash Player.
Il contenitore può accedere a queste informazioni nella proprietà `request` dell'oggetto evento e utilizzarle per determinare come eseguire il proprio codice. Per restituire un valore ad ActionScript, il codice del contenitore chiama il metodo `SetReturnValue()` dell'oggetto ActiveX, passando il risultato (serializzato in una stringa XML) come parametro di tale metodo. Per ulteriori informazioni sul formato XML utilizzato per questa comunicazione, vedere "Il formato XML dell'API esterna" a pagina 787.

Se il contenitore è un browser Web o un altro contenitore ActiveX, se la chiamata non riesce o se il metodo container non specifica un valore da restituire, viene restituito `null`. Il metodo `ExternalInterface.call()` genera un'eccezione SecurityError se l'ambiente contenitore appartiene a una funzione di sicurezza sandbox a cui il codice chiamante non ha accesso. Per ovviare a questo inconveniente è possibile impostare un valore adeguato per `allowScriptAccess` nell'ambiente contenitore. Ad esempio, per modificare il valore di `allowScriptAccess` in una pagina HTML, si modifica l'attributo appropriato nei tag `object` ed `embed`.

Chiamate al codice ActionScript dal contenitore

Un contenitore può chiamare esclusivamente il codice ActionScript presente in una funzione. Per chiamare una funzione ActionScript dall'applicazione contenitore, è necessario eseguire due operazioni: registrare la funzione con la classe ExternalInterface, quindi chiamarla dal codice del contenitore.

Come prima cosa, è necessario registrare la funzione ActionScript per indicare che deve essere resa disponibile al contenitore. Utilizzare il metodo `ExternalInterface.available` nel modo indicato di seguito:

```
function callMe(name:String):String
{
  return "busy signal";
}
ExternalInterface.addCallback("myFunction", callMe);
```

Il metodo `addCallback()` accetta due parametri: il primo, un nome di funzione quale una stringa, è il nome con cui la funzione sarà conosciuta dal contenitore. Il secondo parametro è la funzione ActionScript vera e propria che verrà eseguita quando il contenitore chiama il nome di funzione definito. Dal momento che questi nomi sono distinti, è possibile specificare un nome di una funzione che verrà utilizzata dal contenitore, anche se l'effettiva funzione ActionScript ha un nome diverso. Si tratta di una possibilità particolarmente utile se non si conosce il nome della funzione (ad esempio, se viene specificata una funzione anonima o se la funzione da chiamare viene determinata in fase di runtime).

Una volta che una funzione ActionScript è stata registrata con la classe ExternalInterface, il contenitore può chiamarla. Ciò può avvenire in diversi modi, a seconda del tipo di contenitore. Ad esempio, nel codice JavaScript in un browser Web, la funzione ActionScript viene chiamata utilizzando il nome della funzione registrata come se si trattasse di un metodo dell'oggetto browser Flash Player (ovvero, un metodo dell'oggetto JavaScript che rappresenta il tag `object` o `embed`). In altre parole, vengono passati i parametri e viene restituito un risultato come se venisse chiamata una funzione locale.

```
<script language="JavaScript">
  // callResult gets the value "busy signal"
  var callResult = flashObject.myFunction("my name");
</script>
...
<object id="flashObject"...>
  ...
  <embed name="flashObject".../>
</object>
```

In alternativa, quando si chiama una funzione ActionScript in un file SWF in esecuzione in un'applicazione desktop, il nome della funzione registrata e tutti gli eventuali parametri devono essere serializzati in una stringa in formato XML. Quindi, la chiamata viene eseguita realmente chiamando il metodo `CallFunction()` del controllo ActiveX con la stringa XML come parametro. Per ulteriori informazioni sul formato XML utilizzato per questa comunicazione, vedere "Il formato XML dell'API esterna" a pagina 787.

In entrambi i casi, il valore restituito dalla funzione ActionScript viene ripassato al codice del contenitore, sia direttamente sotto forma di valore quando il chiamante è il codice JavaScript in un browser sia serializzato come stringa in formato XML quando il chiamante è un contenitore ActiveX.

Il formato XML dell'API esterna

La comunicazione tra ActionScript e un'applicazione che contiene il controllo ActiveX di Shockwave Flash utilizza un formato XML specifico per codificare le chiamate alle funzioni e i valori. Il formato XML utilizzato dall'API esterna è suddiviso in due parti. Un formato viene utilizzato per rappresentare le chiamate alle funzioni. Un altro viene utilizzato per rappresentare i singoli valori; questo formato viene utilizzato per i parametri nelle funzioni, nonché per i valori restituiti da queste ultime. Il formato XML per le chiamate alle funzioni viene utilizzato per le chiamate verso e da ActionScript. Per una chiamata a una funzione proveniente da ActionScript, Flash Player passa l'XML al contenitore; per una chiamata proveniente dal contenitore, Flash Player prevede che l'applicazione contenitore passi una stringa XML in questo formato. Il frammento XML seguente mostra un esempio di chiamata a una funzione in formato XML:

```
<invoke name="functionName" returntype="xml">
  <arguments>
    ... (individual argument values)
  </arguments>
</invoke>
```

Il nodo radice è il nodo `invoke` e ha due attributi: `name` indica il nome della funzione da chiamare, mentre `returntype` è sempre `xml`. Se la chiamata alla funzione comprende dei parametri, il nodo `invoke` ha un nodo `arguments` secondario, i cui nodi secondari sono costituiti dai valori di parametro formattati utilizzando il formato dei singoli valori illustrato di seguito.

I singoli valori, compresi i parametri della funzione e i valori da essa restituiti, utilizzano uno schema di formattazione che oltre ai valori effettivi include le informazioni sul tipo di dati. L'elenco seguente illustra le classi ActionScript e il formato XML utilizzato per codificare i valori per tale tipo di dati:

Classe/valore ActionScript	Classe/ valore C#	Formato	Commenti
null	null	`<null/>`	
Boolean `true`	bool true	`<true/>`	
Boolean `false`	bool false	`<false/>`	
String	string	`<string>valore stringa</string>`	

Number, int, uint	single, double, int, uint	`<number>27.5</number>`
		
		`<number>-12</number>`	

Array (gli elementi possono essere di tipi diversi)	Una raccolta che consente l'uso di elementi di tipi diversi, quale ArrayList oppure object[]	`<array>`
		`  <property id="0">`
		`  <number>27.5</number>`
		`  </property>`
		`  <property id="1">`
		`  <string>Hello there!</string>`
		`  </property>`
		`  ...`
		`</array>`
			Il nodo `property` definisce i singoli elementi, mentre l'attributo `id` è l'indice numerico a base zero.

Object	Un dizionario con chiavi in formato stringa e valori oggetto, quale HashTable con chiavi in formato stringa	`<object>`
		`  <property id="name">`
		`  <string>John Doe</string>`
		`  </property>`
		`  <property id="age">`
		`  <string>33</string>`
		`  </property>`
		`  ...`
		`</object>`
			Il nodo `property` definisce le singole proprietà, mentre l'attributo `id` è il nome della proprietà (una stringa).

Altre classi incorporate o personalizzate		`<null/>`
		`  oppure`
		
		`<object></object>`
			ActionScript codifica gli altri oggetti come null o come un oggetto vuoto. In entrambi i casi, qualunque valore di proprietà va perso.

NOTA

A titolo di esempio, oltre alle classi ActionScript questa tabella mostra le classi C# equivalenti; tuttavia, l'API esterna può essere utilizzata per comunicare con qualunque linguaggio di programmazione o ambiente di runtime che supporti i controlli ActiveX e non si limita alle applicazioni C#.

Quando si creano delle applicazioni che utilizzano l'API esterna con un'applicazione contenitore ActiveX, probabilmente è più comodo scrivere un proxy che esegua l'operazione di conversione delle chiamate alle funzioni native nel formato XML serializzato. Per un esempio di una classe proxy scritta in C#, vedere "Funzionamento della classe ExternalInterfaceProxy" a pagina 803.

Esempio: Uso dell'API esterna con una pagina Web contenitore

Questo esempio di applicazione mostra le tecniche appropriate per abilitare la comunicazione tra ActionScript e JavaScript in un browser Web, nel contesto di un'applicazione di Instant Messaging che consente a un utente di chattare con se stesso (da cui il nome dell'applicazione: Introvert IM, IM introverso). I messaggi vengono scambiati tra un form HTML nella pagina Web e un'interfaccia SWF che utilizza l'API esterna. La tecnica mostrata nell'esempio consente di:

- Avviare correttamente la comunicazione dopo aver verificato che il browser sia pronto per comunicare prima di impostare la comunicazione
- Verificare che il contenitore supporti l'API esterna
- Chiamare le funzioni JavaScript da ActionScript, passare i parametri e ricevere dei valori in risposta
- Rendere i metodi ActionScript disponibili alle chiamate di JavaScript ed eseguire tali chiamate

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione Introvert IM sono disponibili nella cartella Samples/IntrovertIM_HTML. L'applicazione è costituita dai file seguenti:

File	Descrizione
IntrovertIMApp.fla oppure IntrovertIMApp.mxml	Il file principale dell'applicazione per Flash (FLA) o Flex (MXML).
com/example/programmingas3/introvertIM/IMManager.as	La classe che attiva e gestisce la comunicazione tra ActionScript e il contenitore.
com/example/programmingas3/introvertIM/IMMessageEvent.as	Il tipo di evento personalizzato, inviato dalla classe IMManager quando viene ricevuto un messaggio dal contenitore.
com/example/programmingas3/introvertIM/IMStatus.as	Un'enumerazione i cui valori rappresentano i diversi valori dello stato di "disponibilità" che è possibile selezionare nell'applicazione.
html-flash/IntrovertIMApp.html oppure html-template/index.template.html	La pagina HTML per l'applicazione per Flash (html-flash/IntrovertIMApp.html) o il modello utilizzato per creare la pagina HTML contenitore per l'applicazione per Adobe Flex (html-template/index.template.html). Questo file contiene tutte le funzioni JavaScript che compongono la parte contenitore dell'applicazione.

Preparazione di una comunicazione ActionScript-browser

L'API esterna viene molto spesso utilizzata per consentire alle applicazioni ActionScript di comunicare con un browser Web. Grazie all'API esterna, i metodi ActionScript possono chiamare il codice scritto in JavaScript e viceversa. In virtù della complessità dei browser e del modo in cui effettuano il rendering interno delle pagine, non è possibile garantire che un documento SWF registri le proprie funzioni di callback prima che venga eseguito il primo codice JavaScript della pagina HTML. Per tale motivo, prima di chiamare le funzioni nel documento SWF da JavaScript, il documento SWF deve sempre chiamare la pagina HTML per notificarle che il documento SWF è pronto per accettare le connessioni.

Ad esempio, attraverso una serie di operazioni eseguite dalla classe IMManager, l'applicazione Introvert IM determina se il browser è pronto per la comunicazione e prepara il documento SWF di conseguenza. La prima operazione, ovvero determinare quando il browser è pronto per la comunicazione, si svolge nella funzione di costruzione IMManager, come indicato di seguito:

```
public function IMManager(initialStatus:IMStatus)
{
  _status = initialStatus;

  // Verifica se il contenitore è in grado di utilizzare l'API esterna.
  if (ExternalInterface.available)
  {
    try
    {
      // Quest'ultima chiama il metodo isContainerReady(), che a propria
      // volta chiama il contenitore per verificare se Flash Player è stato
      // caricato e se il contenitore è pronto per ricevere le chiamate
      // dal documento SWF.
      var containerReady:Boolean = isContainerReady();
      if (containerReady)
      {
        // Se il contenitore è pronto, registra le funzioni del file SWF.
        setupCallbacks();
      }
      else
      {
        // Se il contenitore non è pronto, imposta un timer per chiamare
        // il contenitore a intervalli di 100 ms. Quando il contenitore
        // risponde che è pronto, il timer viene fermato.
        var readyTimer:Timer = new Timer(100);
        readyTimer.addEventListener(TimerEvent.TIMER, timerHandler);
        readyTimer.start();
      }
    }
    ...
  }
  else
  {
    trace("External interface is not available for this container.");
  }
}
```

Innanzitutto, il codice verifica se l'API esterna è almeno disponibile nel contenitore corrente utilizzando la proprietà `ExternalInterface.available`. In caso affermativo, inizia il processo di impostazione della comunicazione. Poiché quando si tenta la comunicazione con un'applicazione esterna possono verificarsi delle eccezioni di sicurezza e altri errori, il codice viene racchiuso in un blocco `try` (i blocchi `catch` corrispondenti sono stati omessi dall'elenco per ragioni di spazio).

Il codice successivo chiama il metodo `isContainerReady()`, rappresentato qui:

```
private function isContainerReady():Boolean
{
  var result:Boolean = ExternalInterface.call("isReady");
  return result;
}
```

Il metodo `isContainerReady()` utilizza a propria volta il metodo `ExternalInterface.call()` per chiamare la funzione JavaScript `isReady()`, come indicato di seguito:

```
<script language="JavaScript">
<!--
// ------- Private vars -------
var jsReady = false;
...
// ------- Funzioni chiamate da ActionScript -------
// Chiamate per verificare se la pagina è stata inizializzata e se
  JavaScript è disponibile
function isReady()
{
  return jsReady;
}
...
// Chiamate da un evento onload del tag <body>
function pageInit()
{
  // Registra che JavaScript è pronto.
  jsReady = true;
}
...
//-->
</script>
```

La funzione `isReady()` restituisce semplicemente il valore della variabile `jsReady`. Tale variabile inizialmente è `false`; quando l'evento `onload` della pagina Web è stato attivato, il valore della variabile diventa `true`. In altre parole, se ActionScript chiama la funzione `isReady()` prima che la pagina sia stata caricata, JavaScript restituisce `false` a `ExternalInterface.call("isReady")`, e di conseguenza il metodo ActionScript `isContainerReady()` restituisce `false`. Una volta caricata la pagina, la funzione JavaScript `isReady()` restituisce `true`, pertanto anche il metodo ActionScript `isContainerReady()` restituisce `true`.

Tornando alla funzione di costruzione IMManager, a seconda della prontezza del contenitore può verificarsi una delle due condizioni seguenti. Se `isContainerReady()` restituisce `true`, il codice chiama semplicemente il metodo `setupCallbacks()`, che completa il processo di impostazione della comunicazione con JavaScript. Se invece `isContainerReady()` restituisce `false`, il processo viene essenzialmente sospeso. Viene creato un oggetto Timer a cui viene ordinato di chiamare il metodo `timerHandler()` ogni 100 millisecondi, come indicato di seguito:

```
private function timerHandler(event:TimerEvent):void
{
  // Verifica se il contenitore è pronto.
  var isReady:Boolean = isContainerReady();
  if (isReady)
  {
    // Se nessun contenitore è pronto, non è più necessario verificare,
    // quindi arresta il timer.
    Timer(event.target).stop();
    // Imposta i metodi ActionScript che saranno disponibili per essere
    // chiamati dal contenitore.
    setupCallbacks();
  }
}
```

Ogni volta che il metodo `timerHandler()` viene chiamato, verifica ancora una volta il risultato del metodo `isContainerReady()`. Una volta inizializzato il contenitore, tale metodo restituisce il valore `true`. A quel punto, il codice arresta il timer e chiama il metodo `setupCallbacks()` per completare il processo di impostazione della comunicazione con il browser.

Esposizione dei metodi ActionScript a JavaScript

Come illustrava l'esempio precedente, quando il codice determina che il browser è pronto, viene chiamato il metodo `setupCallbacks()`. Questo metodo prepara ActionScript alla ricezione delle chiamate da JavaScript, come illustrato di seguito:

```
private function setupCallbacks():void
{
  // Registra le funzioni client SWF con il contenitore
  ExternalInterface.addCallback("newMessage", newMessage);
  ExternalInterface.addCallback("getStatus", getStatus);
  // Notifica al contenitore che il file SWF è pronto per essere chiamato.
  ExternalInterface.call("setSWFIsReady");
}
```

Il metodo `setCallBacks()` conclude la preparazione alla comunicazione chiamando `ExternalInterface.addCallback()` per registrare i due metodi che saranno disponibili per le chiamate da JavaScript. In questo codice, il primo parametro (il nome con cui il metodo è conosciuto da JavaScript, ovvero `"newMessage"` e `"getStatus"`) è uguale al nome del metodo in ActionScript. (Nel caso dell'esempio, poiché sarebbe stato ininfluente indicare nomi diversi, per semplicità è stato utilizzato lo stesso nome.) Infine, viene utilizzato il metodo `ExternalInterface.call()` per chiamare la funzione JavaScript `setSWFIsReady()`, che notifica al contenitore che le funzioni ActionScript sono state registrate.

Comunicazione da ActionScript al browser

L'applicazione Introvert IM dimostra una serie di esempi di chiamate a funzioni JavaScript nella pagina contenitore. Nel caso più semplice (un esempio tratto dal metodo `setupCallbacks()`), la funzione JavaScript `setSWFIsReady()` viene chiamata senza passare alcun parametro o ricevere alcun valore:

```
ExternalInterface.call("setSWFIsReady");
```

In un altro esempio tratto dal metodo `isContainerReady()`, ActionScript chiama la funzione `isReady()` e riceve come risposta un valore booleano:

```
var result:Boolean = ExternalInterface.call("isReady");
```

È possibile passare parametri alle funzioni JavaScript anche utilizzando l'API esterna. Ad esempio, considerare il metodo `sendMessage()` della classe IMManager, che viene chiamato quando l'utente sta inviando un nuovo messaggio al proprio interlocutore:

```
public function sendMessage(message:String):void
{
  ExternalInterface.call("newMessage", message);
}
```

Anche in questo caso, si utilizza `ExternalInterface.call()` per chiamare la funzione JavaScript designata, notificando al browser il nuovo messaggio. Inoltre, il messaggio stesso viene passato come parametro aggiuntivo a `ExternalInterface.call()`, e di conseguenza viene passato come parametro alla funzione JavaScript `newMessage()`.

Chiamate al codice ActionScript da JavaScript

Una comunicazione dovrebbe essere sempre biunivoca, e l'applicazione Introvert IM non fa eccezione. Non solo il client IM di Flash Player chiama JavaScript per inviare i messaggi, ma il form HTML chiama il codice JavaScript anche per inviare messaggi e chiedere informazioni al file SWF. Ad esempio, quando il file SWF notifica al contenitore che il contatto è stato stabilito ed è pronto per comunicare, come prima cosa il browser chiama il metodo `getStatus()` della classe IMManager per recuperare lo stato di disponibilità iniziale dell'utente dal client IM SWF. Questo avviene in una pagina Web, nella funzione `updateStatus()`, nel modo indicato di seguito:

```
<script language="JavaScript">
...
function updateStatus()
{
  if (swfReady)
  {
    var currentStatus = getSWF("IntrovertIMApp").getStatus();
    document.forms["imForm"].status.value = currentStatus;
  }
}
...
</script>
```

Il codice verifica il valore della variabile `swfReady`, che rileva se il file SWF ha notificato il browser che ne ha registrato i metodi con la classe ExternalInterface. Se il file SWF è pronto per ricevere la comunicazione, la riga successiva (`var currentStatus =` ...) chiama di fatto il metodo `getStatus()` nella classe IMManager. In questa riga di codice vengono eseguite tre operazioni:

- Viene chiamata la funzione JavaScript `getSWF()`, che restituisce un riferimento all'oggetto JavaScript che rappresenta il file SWF. Il parametro passato a `getSWF()` determina quale oggetto browser viene restituito nel caso in cui in una stessa pagina HTML sia presente più di un solo file SWF. Il valore passato a tale parametro deve corrispondere all'attributo `id` del tag `object` e all'attributo `name` del tag `embed` utilizzati per includere il file SWF.

- Utilizzando il riferimento al file SWF, viene chiamato il metodo `getStatus()` come se fosse un metodo dell'oggetto SWF. In questo caso, viene utilizzato il nome di funzione "`getStatus`" perché è il nome con cui la funzione ActionScript viene registrata mediante `ExternalInterface.addCallback()`.
- Il metodo ActionScript `getStatus()` restituisce un valore. Tale valore viene assegnato alla variabile `currentStatus`, che viene successivamente assegnata come contenuto (la proprietà `value`) del campo di testo `status`.

La funzione JavaScript `sendMessage()` dimostra il passaggio di un parametro a una funzione ActionScript. (`sendMessage()` è la funzione che viene chiamata quando l'utente preme il pulsante Invia presente nella pagina HTML.)

```
<script language="JavaScript">
...
function sendMessage(message)
{
  if (swfReady)
  {
    ...
    getSWF("IntrovertIMApp").newMessage(message);
  }
}
...
</script>
```

Il metodo ActionScript `newMessage()` prevede un solo parametro, pertanto la variabile JavaScript `message` viene passata ad ActionScript utilizzandola come parametro nella chiamata al metodo `newMessage()` nel codice JavaScript.

Rilevamento del tipo di browser

A causa delle differenze nel modo in cui i browser accedono ai contenuti, è importante utilizzare sempre JavaScript per rilevare quale browser è utilizzato dall'utente e per accedere al filmato in base alla sintassi specifica del browser, utilizzando l'oggetto window o document, come mostrato nella funzione JavaScript `getSWF()` dell'esempio seguente:

```
<script language="JavaScript">
...
function getSWF(movieName)
{
  if (navigator.appName.indexOf("Microsoft") != -1)
  {
    return window[movieName];
  }
  else
  {
    return document[movieName];
  }
}
...
</script>
```

Se lo script non rileva il tipo di browser dell'utente, quest'ultimo potrebbe sperimentare un comportamento imprevisto durante la riproduzione dei file SWF in un contenitore HTML.

Esempio: Uso dell'API esterna con un contenitore ActiveX

Questo esempio dimostra l'uso dell'API esterna per la comunicazione tra ActionScript e un'applicazione desktop che utilizza il controllo ActiveX. L'esempio riutilizza l'applicazione Introvert IM, compresi il codice ActionScript e persino lo stesso file SWF, e pertanto non descrive l'uso dell'API esterna in ActionScript. Per meglio comprendere questo esempio è consigliabile aver sviluppato una certa dimestichezza con l'esempio precedente.

L'applicazione desktop in questo esempio è scritta in C# mediante Microsoft Visual Studio .NET. La discussione è incentrata sulle tecniche specifiche per l'uso dell'API esterna con il controllo ActiveX. L'esempio seguente illustra:

- Le chiamate alle funzioni ActionScript da un'applicazione desktop su cui è presente il controllo ActiveX di Flash Player
- La ricezione delle chiamate alle funzioni provenienti da ActionScript e la loro elaborazione in un contenitore ActiveX

- L'uso della classe proxy per nascondere i dettagli del formato XML serializzato utilizzato da Flash Player per i messaggi inviati a un contenitore ActiveX

Per ottenere i file dell'applicazione per questo esempio, accedere all'indirizzo www.adobe.com/go/learn_programmingAS3samples_flash_it. I file dell'applicazione Introvert IM in C# sono disponibili nella cartella Samples/IntrovertIM_CSharp. L'applicazione è costituita dai file seguenti:

File	Descrizione
AppForm.cs	Il file dell'applicazione principale, con l'interfaccia C# Windows Forms.
bin/Debug/IntrovertIMApp.swf	Il file SWF caricato dall'applicazione.
ExternalInterfaceProxy/ ExternalInterfaceProxy.cs	La classe che funge da wrapper del controllo ActiveX per la comunicazione con l'interfaccia esterna. Fornisce i meccanismi per effettuare e ricevere le chiamate ActionScript.
ExternalInterfaceProxy/ ExternalInterfaceSerializer.cs	La classe che si occupa di convertire i messaggi in formato XML di Flash Player in oggetti .NET.
ExternalInterfaceProxy/ ExternalInterfaceEventArgs.cs	Questo file definisce due tipi (classi) C#: un delegato personalizzato e una classe arguments dell'evento, che vengono utilizzati dalla classe ExternalInterfaceProxy per notificare a un listener una chiamata a una funzione proveniente da ActionScript.
ExternalInterfaceProxy/ ExternalInterfaceCall.cs	Questa classe è un oggetto value che rappresenta una chiamata a una funzione da ActionScript al contenitore ActiveX, con le proprietà per il nome e i parametri della funzione.
bin/Debug/IntrovertIMApp.swf	Il file SWF caricato dall'applicazione.
obj/AxInterop.ShockwaveFlashObjects.dll, obj/Interop.ShockwaveFlashObjects.dll	Assembly di wrapper creati da Visual Studio .NET e richiesti per accedere al controllo ActiveX di Flash Player (Adobe Shockwave® Flash) dal codice gestito.

Panoramica dell'applicazione Introvert IM in C#

Questa applicazione di esempio rappresenta due programmi client di Instant Messaging (uno all'interno di un file SWF e un altro creato con Windows Forms) che comunicano l'uno con l'altro. L'interfaccia utente include un'istanza del controllo ActiveX Shockwave Flash, all'interno del quale viene caricato il file SWF che contiene il client IM ActionScript. L'interfaccia include anche diversi campi di testo che compongono il client IM di Windows Forms: un campo per l'immissione dei messaggi (`MessageText`), un altro che visualizza la trascrizione dei messaggi scambiati tra i client (`Transcript`) e un terzo (`Status`) che visualizza lo stato della disponibilità in base all'impostazione specificata nel client IM SWF.

Inclusione del controllo ActiveX di Shockwave Flash

Per includere il controllo ActiveX di Shockwave Flash nell'applicazione Windows Forms, come prima cosa è necessario aggiungerlo alla Casella degli strumenti di Microsoft Visual Studio.

Per aggiungere il controllo alla Casella degli strumenti:

1. Aprire la Casella degli strumenti di Visual Studio.
2. Fare clic con il pulsante destro del mouse nella sezione Windows Form di Visual Studio 2003 o in qualunque sezione di Visual Studio 2005. Dal menu di scelta rapida, selezionare Aggiungi/Rimuovi gli elementi in Visual Studio 2003 (Scegli elementi... in Visual Studio 2005).

   Viene aperta la finestra di dialogo Personalizza Casella degli strumenti (2003)/Scegli Casella degli strumenti (2005).
3. Selezionare la scheda Componenti COM, che elenca tutti i componenti COM disponibili sul computer, compreso il controllo ActiveX di Flash Player.
4. Scorrere fino a Oggetto Shockwave Flash (Shockwave Flash Object) e selezionarlo.

   Se l'elemento non è presente nell'elenco, assicurarsi che il controllo ActiveX di Flash Player sia installato sul sistema.

Nozioni fondamentali sulla comunicazione da ActionScript al contenitore ActiveX

Il funzionamento della comunicazione che utilizza l'API esterna con un'applicazione contenitore ActiveX è uguale alla comunicazione con un browser Web, ma con una differenza importante. Come descritto in precedenza, quando ActionScript comunica con un browser Web, in fase di sviluppo viene specificato che le funzioni vengano chiamate direttamente; i dettagli di come le chiamate e le risposte delle funzioni vengono formattate per essere passate tra il lettore e il browser sono nascosti. Tuttavia, quando si utilizza l'API esterna per comunicare con un'applicazione contenitore ActiveX, Flash Player invia dei messaggi (chiamate a funzioni e valori restituiti) all'applicazione in un formato XML specifico e prevede che le chiamate alle funzioni e i valori restituiti dall'applicazione contenitore utilizzino lo stesso formato XML. Lo sviluppatore dell'applicazione contenitore ActiveX deve scrivere del codice in grado di comprendere e creare le chiamate alle funzioni e le relative risposte in un formato appropriato.

L'esempio dell'applicazione Introvert IM in C# comprende un set di classi che consentono di evitare la formattazione dei messaggi; piuttosto, è possibile lavorare con tipi di dati standard quando si chiamano e ricevono funzioni ActionScript. La classe ExternalInterfaceProxy, insieme ad altre classi di supporto, fornisce questa funzionalità e può essere riutilizzata in qualunque progetto .NET per facilitare la comunicazione dell'API esterna.

Le sezioni di codice seguenti, estratte dal form dell'applicazione principale (AppForm.cs), dimostrano l'interazione semplificata che è possibile ottenere utilizzando la classe ExternalInterfaceProxy:

```
public class AppForm : System.Windows.Forms.Form
{
  ...
  private ExternalInterfaceProxy proxy;
  ...
  public AppForm()
  {
    ...
    // Registra questa applicazione per ricevere una notifica quando
    // il proxy riceve una chiamata da ActionScript.
    proxy = new ExternalInterfaceProxy(IntrovertIMApp);
    proxy.ExternalInterfaceCall += new
  ExternalInterfaceCallEventHandler(proxy_ExternalInterfaceCall);
    ...
  }
  ...
```

L'applicazione dichiara e crea un'istanza ExternalInterfaceProxy di nome `proxy`, passando un riferimento al controllo ActiveX di Shockwave Flash presente nell'interfaccia utente (`IntrovertIMApp`). Quindi, il codice registra il metodo `proxy_ExternalInterfaceCall()` per ricevere l'evento `ExternalInterfaceCall` del proxy. Questo evento viene inviato dalla classe ExternalInterfaceProxy quando una chiamata a una funzione proviene da Flash Player. La sottoscrizione a questo evento rappresenta il modo in cui il codice C# riceve le chiamate alle funzioni provenienti da ActionScript e risponde a esse.

Quando una chiamata a una funzione proviene da ActionScript, l'istanza ExternalInterfaceProxy (`proxy`) riceve la chiamata, la converte dal formato XML e invia una notifica agli oggetti che sono listener dell'evento `ExternalInterfaceCall` del proxy. Nel caso della classe AppForm, il metodo `proxy_ExternalInterfaceCall()` gestisce tale evento, come indicato di seguito:

```
/// <summary>
/// Chiamato dal proxy quando una chiamata ActionScript ExternalInterface
/// viene effettuata dal file SWF
/// </summary>
private object proxy_ExternalInterfaceCall(object sender,
ExternalInterfaceCallEventArgs e)
{
  switch (e.FunctionCall.FunctionName)
  {
    case "isReady":
      return isReady();
    case "setSWFIsReady":
      setSWFIsReady();
      return null;
    case "newMessage":
      newMessage((string)e.FunctionCall.Arguments[0]);
      return null;
    case "statusChange":
      statusChange();
      return null;
    default:
      return null;
  }
}
...
```

Al metodo viene passata un'istanza ExternalInterfaceCallEventArgs, che in questo esempio ha il nome `e`. Tale oggetto, a propria volta, contiene una proprietà `FunctionCall` che è un'istanza della classe ExternalInterfaceCall.

Un'istanza ExternalInterfaceCall è un oggetto value semplice con due proprietà. La proprietà `FunctionName` contiene il nome di funzione specificato nell'istruzione ActionScript `ExternalInterface.Call()`. Se in ActionScript vengono aggiunti dei parametri, questi ultimi vengono inclusi nella proprietà `Arguments` dell'oggetto ExternalInterfaceCall. In questo caso, il metodo che gestisce l'evento è semplicemente un'istruzione `switch` che funge da gestore del traffico. Il valore della proprietà `FunctionName` (`e.FunctionCall.FunctionName`) determina quale metodo della classe AppForm viene chiamato.

I rami dell'istruzione `switch` nel codice precedente dimostrano gli scenari più comuni di chiamate ai metodi. Ad esempio, qualunque metodo deve restituire un valore ad ActionScript (ad esempio, la chiamata al metodo `isReady()`), altrimenti deve restituire `null` (come mostrato nelle altre chiamate ai metodi). L'accesso ai parametri passati da ActionScript è mostrato nella chiamata al metodo `newMessage()`, che passa un parametro `e.FunctionCall.Arguments[0]`, il primo elemento dell'array `Arguments`.

La chiamata a una funzione ActionScript da C# mediante la classe ExternalInterfaceProxy è ancora più diretta della ricezione da ActionScript di una chiamata a una funzione. Per chiamare una funzione ActionScript, si utilizza il metodo `Call()` dell'istanza ExternalInterfaceProxy, come indicato di seguito:

```
  /// <summary>
  /// Chiamato quando si preme il pulsante "Invia"; il valore nel
  /// campo di testo MessageText viene passato come parametro.
  /// </summary>
  /// <param name="message">Il messaggio da inviare.</param>
  private void sendMessage(string message)
  {
    if (swfReady)
    {
      ...
      // Chiama la funzione newMessage in ActionScript.
      proxy.Call("newMessage", message);
    }
  }
  ...
  /// <summary>
  /// Chiama la funzione ActionScript per ottenere lo stato di
  "disponibilità"
  // corrente e scriverlo nel campo di testo.
  /// </summary>
  private void updateStatus()
  {
    Status.Text = (string)proxy.Call("getStatus");
  }
  ...
}
```

Come mostra l'esempio, il metodo `Call()` della classe ExternalInterfaceProxy è molto simile alla sua controparte ActionScript, `ExternalInterface.Call()`. Il primo parametro è una stringa che corrisponde al nome della proprietà da chiamare. Tutti gli eventuali parametri supplementari (non mostrati in questo caso) vengono passati alla funzione ActionScript. Se la funzione ActionScript restituisce un valore, tale valore viene restituito dal metodo `Call()` (come mostrato nell'esempio precedente).

Funzionamento della classe ExternalInterfaceProxy

L'uso di una wrapper proxy su un controllo ActiveX non sempre si dimostra pratico; in alcuni casi, può essere preferibile o necessario scrivere una classe proxy personalizzata (ad esempio, scritta in un linguaggio di programmazione diverso o destinata a una piattaforma diversa). Anche se in questa sede non vengono descritti tutti i dettagli della creazione di un proxy, comprendere il funzionamento interno di una classe proxy illustrato in questo esempio può rivelarsi utile.

Si utilizza il metodo `CallFunction()` del controllo ActiveX di Shockwave Flash per chiamare una funzione ActionScript dal contenitore ActiveX mediante l'API esterna. Questa operazione viene mostrata nel seguente estratto dal metodo `Call()` della classe ExternalInterfaceProxy:

```
// Chiama una funzione ActionScript sul file SWF in "_flashControl",
// che è un controllo ActiveX di Shockwave Flash.
string response = _flashControl.CallFunction(request);
```

In questa porzione di codice, `_flashControl` è il codice ActiveX di Shockwave Flash. Le chiamate alle funzioni ActionScript vengono effettuate mediante il metodo `CallFunction()`, il quale riceve un parametro (nell'esempio, `request`), che è una stringa che contiene le istruzioni in formato XML che comprendono il nome della funzione ActionScript da chiamare e gli eventuali parametri. Qualunque valore restituito da ActionScript viene codificato sotto forma di stringa in formato XML e reinviato come valore restituito dalla chiamata `CallFunction()`. In questo esempio, tale stringa XML viene memorizzata nella variabile `response`.

La ricezione di una chiamata a una funzione proveniente da ActionScript è un processo suddiviso in più fasi. Tali chiamate provocano l'invio dell'evento FlashCall da parte del controllo ActiveX di Shockwave Flash, pertanto una classe (ad esempio, ExternalInterfaceProxy) che intende ricevere le chiamate da un file SWF deve definire un gestore per tale evento. Nella classe ExternalInterfaceProxy, la funzione del gestore di eventi si chiama `_flashControl_FlashCall()` ed è registrata per intercettare l'evento nella funzione di costruzione della classe, come mostrato di seguito:

```
private AxShockwaveFlash _flashControl;

public ExternalInterfaceProxy(AxShockwaveFlash flashControl)
{
  _flashControl = flashControl;
  _flashControl.FlashCall += new
  _IShockwaveFlashEvents_FlashCallEventHandler(_flashControl_FlashCall);
}
...
private void _flashControl_FlashCall(object sender,
  _IShockwaveFlashEvents_FlashCallEvent e)
{
  // Usa la proprietà request dell'oggetto evento ("e.request")
  // per eseguire alcune azioni.
  ...
  // Restituisce un valore ad ActionScript;
  // il valore restituito deve prima essere codificato sotto forma di
  stringa in formato XML.
  _flashControl.SetReturnValue(encodedResponse);
}
```

L'oggetto evento (`e`) contiene una proprietà `request` (`e.request`) costituita da una stringa che contiene le informazioni sulla chiamata alla funzione (ad esempio il nome della funzione e i parametri) in formato XML. Queste informazioni possono essere utilizzate dal contenitore per determinare quale codice eseguire. Nella classe ExternalInterfaceProxy, la richiesta viene convertita dal formato XML a un oggetto ExternalInterfaceCall, che fornisce le stesse informazioni in una forma più accessibile. Il metodo `SetReturnValue()` del controllo ActiveX viene utilizzato per restituire il risultato di una funzione al chiamante ActionScript; ancora una volta, il parametro risultante deve essere codificato nello stesso formato XML utilizzato dall'API esterna.

La comunicazione tra ActionScript e un'applicazione che contiene il controllo ActiveX di Shockwave Flash utilizza un formato XML specifico per codificare le chiamate alle funzioni e i valori. Nell'esempio dell'applicazione Introvert IM in C#, la classe ExternalInterfaceProxy consente al codice presente nel form dell'applicazione di agire direttamente sui valori inviati o ricevuti da ActionScript e di ignorare il formato XML utilizzato da Flash Player. Per ottenere questo risultato, la classe ExternalInterfaceProxy utilizza i metodi della classe ExternalInterfaceSerializer per convertire i messaggi XML in oggetti .NET. La classe ExternalInterfaceSerializer ha quattro metodi pubblici:

- `EncodeInvoke()`: codifica un nome di una funzione e un ArrayList di argomenti C# nel formato XML appropriato.
- `EncodeResult()`: codifica il valore di un risultato in un formato XML appropriato.
- `DecodeInvoke()`: decodifica una chiamata a una funzione proveniente da ActionScript. La proprietà `request` dell'oggetto evento FlashCall viene passata al metodo `DecodeInvoke()` e converte la chiamata in un oggetto ExternalInterfaceCall.
- `DecodeResult()`: decodifica l'XML ricevuto come risultato di una chiamata a una funzione ActionScript.

Questi metodi codificano i valori C# nel formato XML dell'API esterna e decodificano l'XML in oggetti C#. Per dettagli sul formato XML utilizzato da Flash Player, vedere "Il formato XML dell'API esterna" a pagina 787.

CAPITOLO 26

Sicurezza di Flash Player

26

La sicurezza è uno dei problemi fondamentali di Adobe, utenti, proprietari di siti Web e sviluppatori di contenuto. Ecco perché Adobe Flash Player 9 include una serie di controlli e regole di sicurezza finalizzati a salvaguardare gli utenti, i proprietari di siti Web e gli sviluppatori di contenuto. In questo capitolo vengono illustrate le operazioni che è possibile effettuare con il modello di sicurezza di Flash Player durante lo sviluppo di applicazioni Flash. Tutti i file SWF presi in esame in questo capitolo sono considerati pubblicati in ActionScript 3.0 (e, di conseguenza, eseguiti in Flash Player 9 o successivo), se non indicato diversamente.

Questo capitolo è da intendersi come una panoramica sulle funzioni di sicurezza e non ha lo scopo di illustrare in modo esaustivo tutti i dettagli di implementazione, gli scenari di impiego o le possibili ramificazioni di alcune API. Per una discussione più dettagliata dei concetti di sicurezza di Flash Player, vedere il white paper sulla sicurezza di *Flash Player 9* all'indirizzo www.adobe.com/go/fp9_0_security_it.

Sommario

Panoramica sulla sicurezza di Flash Player

La maggior parte delle funzioni di sicurezza di Flash Player si basa sul dominio di origine dei file SWF, multimediali e delle risorse caricati. Un file SWF di un dominio Internet specifico, quale www.example.com, può sempre accedere ai dati di tale dominio. Tali risorse sono inserite nello stesso raggruppamento di sicurezza, conosciuto come *funzione di sicurezza sandbox*. Per ulteriori informazioni, vedere "Funzioni di sicurezza sandbox" a pagina 822.

Ad esempio, un file SWF può caricare altri file SWF, bitmap, file audio e di testo e qualsiasi altra risorsa appartenente al suo stesso dominio. Inoltre, lo scambio di script tra due file SWF dello stesso dominio è sempre consentito, a condizione che entrambi siano stati scritti con ActionScript 3.0. Lo *scambio di script* consiste nella capacità di un file SWF di utilizzare ActionScript per accedere alle proprietà, ai metodi e agli oggetti di un altro file SWF. Lo scambio di script non è supportato tra file SWF scritti con ActionScript 3.0 e file scritti con versioni precedenti di ActionScript; tuttavia, questi file sono in grado di comunicare tra loro mediante la classe LocalConnection. Per ulteriori informazioni, vedere "Scambio di script" a pagina 832.

Le seguenti regole di sicurezza di base vengono sempre applicate per impostazione predefinita:

- Le risorse che condividono la stessa funzione di sicurezza sandbox possono sempre accedere le une alle altre.
- I file SWF appartenenti a una funzione di sicurezza sandbox remota non possono accedere a file e dati locali.

Flash Player considera i seguenti domini come domini singoli e imposta per ciascuno di essi funzioni di sicurezza sandbox specifiche:

- `http://example.com`
- `http://www.example.com`
- `http://store.example.com`
- `https://www.example.com`
- `http://192.0.34.166`

Anche se un dominio con nome, quale http://example.com, viene mappato su un indirizzo IP specifico, quale http://192.0.34.166, Flash Player imposta una funzione di sicurezza sandbox specifica per ciascuno.

Sono disponibili due diversi metodi fondamentali per garantire a un file SWF l'accesso a risorse appartenenti a sandbox diverse da quelle del file stesso:

- Il metodo `Security.allowDomain()` (vedere "Controlli creatore (sviluppatore)" a pagina 820)
- Il file di criteri dei domini (vedere "Controlli del sito Web (file di criteri dei domini)" a pagina 816)

A un file SWF è proibito, per impostazione predefinita, scambiare script con file SWF di ActionScript 3.0 appartenenti ad altri domini, così come caricare dati da altri domini. Per poterlo fare, è necessario chiamare il metodo `Security.allowDomain()` nel file SWF caricato. Per informazioni dettagliate, vedere "Scambio di script" a pagina 832.

Nel modello di sicurezza di Flash Player si fa distinzione tra caricamento di *contenuto* e accesso a o caricamento di *dati*.

- Caricamento di contenuto — Viene definito *contenuto* qualsiasi tipo di contenuto multimediale, inclusi file multimediali visivi visualizzati da Flash Player, file audio, video o file SWF contenenti file multimediali visualizzati. Si definisce *dati* qualcosa di accessibile unicamente al codice di ActionScript. È possibile caricare contenuto mediante classi quali Loader, Sound e NetStream.
- Accesso a contenuto come dati o caricamento di dati — È possibile accedere ai dati in due diversi modi: mediante estrazione dei dati dal contenuto multimediale caricato o mediante caricamento diretto dei dati da un file esterno (quale un file XML). Per estrarre dati da contenuto multimediale caricato è possibile utilizzare oggetti Bitmap, il metodo `BitmapData.draw()`, la proprietà `Sound.id3` o il metodo `SoundMixer.computeSpectrum()`. Per caricare i dati è possibile utilizzare classi quali URLStream, URLLoader, Socket e XMLSocket.

Il modello di sicurezza di Flash Player definisce varie regole per il caricamento di contenuto e l'accesso ai dati. In generale, vi sono meno limitazioni al caricamento del contenuto rispetto all'accesso ai dati.

Il contenuto (file SWF, bitmap, file mp3 e video) può essere caricato ovunque, tuttavia, se si tratta di contenuto appartenente a un dominio diverso da quello del file SWF che carica, esso verrà suddiviso in partizioni in una funzione di sicurezza sandbox separata.

Esistono alcune barriere al caricamento di contenuto:

- Per impostazione predefinita, i file SWF locali (quelli caricati da indirizzi non di rete, quale un disco rigido di un utente) sono classificati nella sandbox locale con file system. Questi file non possono caricare contenuto dalla rete. Per ulteriori informazioni, vedere "Funzioni di sicurezza sandbox locali" a pagina 822.
- I server RTMP (Real-Time Messaging Protocol) sono in grado di limitare l'accesso al contenuto. Per ulteriori informazioni, vedere "Contenuto distribuito mediante server RTMP" a pagina 832.

Se il contenuto caricato è un file di immagine, audio o video, un file SWF esterno alla sua sicurezza sandbox non può accedere ai dati di tale file, quali dati pixel o dati audio, a meno che il dominio del file SWF non sia stato incluso in un file di criteri dei domini nel dominio di origine del file multimediale. Per informazioni dettagliate, vedere "Accesso a file multimediali caricati come dati" a pagina 836.

Tra le altre forme di dati caricati vi sono file di testo o XML caricati con un oggetto URLLoader. Anche in questo caso, per accedere a dati appartenenti a una funzione di sicurezza sandbox diversa, è necessario ottenere un'autorizzazione mediante un file di criteri dei domini nel dominio di origine. Per informazioni dettagliate, vedere "Uso di URLLoader e URLStream" a pagina 840.

Panoramica dei controlli di autorizzazione

Il modello di sicurezza della fase di runtime del client di Flash Player è stato progettato a partire da risorse quali file SWF, dati locali e URL Internet. Gli *stakeholder* sono le parti che possiedono o utilizzano tali risorse. Gli stakeholder possono controllare le proprie risorse (mediante impostazioni di sicurezza) e ogni risorsa presenta quattro stakeholder. Flash Player applica in maniera rigorosa una gerarchia di autorità per tali controlli, come illustrato nel grafico seguente:

Gerarchia dei controlli di sicurezza

Ciò significa, ad esempio, che se un amministratore limita l'accesso a una risorsa, nessun altro stakeholder può ignorare tale restrizione.

I controlli amministratore, utente e sito Web vengono illustrati nelle sezioni che seguono. Le impostazioni del creatore (sviluppatore) vengono descritte nella parte restante di questo capitolo.

Controlli amministratore

L'amministratore di un computer (utente che ha eseguito l'accesso con privilegi amministrativi) può applicare impostazioni di sicurezza di Flash Player valide per tutti gli utenti del computer. In un ambiente non aziendale, come nel caso di un computer domestico, vi è in genere un utente con accesso di amministratore. Anche in un ambiente aziendale singoli utenti possono disporre di diritti amministrativi.

Esistono due tipi di controlli amministratore:

- Il file mms.cfg
- La directory Global Flash Player Trust

File mms.cfg

Nei sistemi Mac OS X, il file mms.cfg si trova in /Library/Supporto Applicazioni/Macromedia/mms.cfg. Nei sistemi Microsoft Windows, tale file si trova nella cartella Macromedia Flash Player, nella directory di sistema (ad esempio, C:\windows\system32\macromed\flash\mms.cfg, in un'installazione di Windows XP predefinita).

All'avvio, Flash Player legge le impostazioni di sicurezza da questo file e le impiega per limitare le funzionalità.

Il file mms.cfg include impostazioni utilizzate dall'amministratore per eseguire le seguenti attività:

- **Caricamento dati** — Limita la lettura dei file SWF locali, disabilita lo scaricamento e il caricamento dei file e imposta il limite di memorizzazione per oggetti condivisi persistenti.
- **Controlli per la riservatezza** — Disattiva l'accesso a microfono e fotocamera, nega ai file SWF di riprodurre contenuto senza finestre e impedisce ai file SWF di un dominio che non corrisponde all'URL visualizzato in una finestra di browser di accedere a oggetti condivisi persistenti.
- **Aggiornamenti di Flash Player** — Imposta l'intervallo di verifica di versioni aggiornate di Flash Player, specifica l'URL per la ricerca di informazioni sugli aggiornamenti di Flash Player, imposta l'URL da cui scaricare le versioni aggiornate di Flash Player e disattiva completamente gli aggiornamenti automatici di Flash Player.
- **Supporto di file legacy** — Specifica se collocare file SWF di versioni precedenti nella sandbox locale affidabile.
- **Sicurezza dei file locali** — Specifica se i file locali possono essere collocati nella sandbox locale affidabile.
- **Modalità a schermo intero** — Disabilita la modalità a schermo intero.

I file SWF possono accedere ad alcune informazioni sulle funzionalità che sono state disattivate mediante chiamata delle proprietà `Capabilities.avHardwareDisable` e `Capabilities.localFileReadDisable`. Tuttavia, la maggior parte delle informazioni contenute nel file mms.cfg non possono essere interrogate da ActionScript.

Per imporre criteri di riservatezza e sicurezza indipendenti dall'applicazione a un computer, il file mms.cfg dovrebbe essere modificato unicamente dagli amministratori di sistema. Il file mms.cfg non deve essere utilizzato dagli installatori di applicazioni. Nonostante sia possibile per un installatore con privilegi di amministratore modificare il contenuto del file mms.cfg file, Adobe considera tale comportamento una violazione della fiducia dell'utente ed esorta pertanto gli installatori a non modificare tale file.

Directory Global Flash Player Trust

Le applicazioni di utenti e installatori con privilegi di amministratore sono in grado di registrare file SWF locali specifici come affidabili. Tali file SWF vengono assegnati alla sandbox locale affidabile. Questi file possono interagire con qualunque altro file SWF e possono caricare dati sia in remoto che in locale. I file vengono designati come affidabili all'interno della directory Global Flash Player Trust, che si trova nella stessa directory del mms.cfg file, nei seguenti percorsi (specifici in base all'utente corrente):

- Windows: system\Macromed\Flash\FlashPlayerTrust
  (ad esempio, C:\windows\system32\Macromed\Flash\FlashPlayerTrust)
- Mac: app support/Macromedia/FlashPlayerTrust
  (ad esempio, /Library/Supporto Applicazioni/Macromedia/FlashPlayerTrust)

La directory Flash Player Trust può contenere un numero indefinito di file di testo, ciascuno con un elenco dei percorsi affidabili, un percorso per riga. Ogni percorso può corrispondere a un singolo file SWF, un file HTML o una directory. Le righe di commento iniziano con il simbolo #. Ad esempio, un file di configurazione affidabile di Flash Player contenente il seguente testo garantisce uno stato affidabile a tutti i file contenuti nella directory specificata e in tutte le relative sottodirectory:

```
# Considera affidabili i file delle directory seguenti:
C:\Documents and Settings\All Users\Documenti\SampleApp
```

I percorsi elencati in un file di configurazione affidabile devono sempre essere percorsi locali o percorsi di rete SMB. I percorsi HTTP sono ignorati nel file di configurazione affidabile; solo i file locali possono essere considerati affidabili.

Per evitare conflitti, assegnare a ogni file di configurazione affidabile un nome corrispondente all'applicazione di installazione e utilizzare l'estensione .cfg.

Gli sviluppatori che distribuiscono un file SWF eseguito a livello locale mediante un'applicazione di installazione possono fare in modo che tale applicazione aggiunga un file di configurazione alla directory Global Flash Player Trust, al fine di garantire la totale affidabilità del file che stanno distribuendo. L'applicazione di installazione deve essere eseguita da un utente con privilegi amministrativi. A differenza del file mms.cfg, la directory Global Flash Player Trust viene inclusa allo scopo di garantire autorizzazioni di affidabilità mediante applicazioni di installazione. Sia gli utenti amministrativi che le applicazioni di installazione possono designare applicazioni locali come affidabili mediante la directory Global Flash Player Trust.

Esistono anche directory Flash Player Trust per singoli utenti (vedere la sezione seguente, "Controlli utente").

Controlli utente

Flash Player offre tre diversi meccanismi a livello utente per l'impostazione delle autorizzazioni: l'interfaccia utente Impostazioni, Gestione impostazioni e la directory User Flash Player Trust.

Interfaccia utente Impostazioni e Gestione impostazioni

L'interfaccia utente Impostazioni è un meccanismo rapido e interattivo per la configurazione delle impostazioni di un dominio specifico. Gestione impostazioni presenta un'interfaccia più dettagliata e consente di effettuare modifiche a livello generale in grado di alterare le autorizzazioni di molti o di tutti i domini. Inoltre, quando viene richiesta una nuova autorizzazione da parte di un file SWF e sono necessarie decisioni in fase di runtime relative alla sicurezza o alla riservatezza, vengono visualizzate finestre di dialogo nelle quali è possibile modificare alcune impostazioni di Flash Player.

Gestione impostazioni e l'interfaccia utente Impostazioni offrono le seguenti opzioni di sicurezza:

- Impostazioni di fotocamera e microfono — L'utente è in grado di controllare l'accesso di Flash Player alla fotocamera e al microfono del computer. L'utente è in grado di concedere o negare l'accesso a tutti i siti o a siti specifici. Se non viene specificata un'impostazione per tutti i siti o per un sito specifico, viene visualizzata una finestra di dialogo ogni volta che un file SWF tenta di accedere alla fotocamera o al microfono, per permettere all'utente di scegliere se consentire o meno l'accesso a tale dispositivo da parte del file SWF. È inoltre possibile specificare la fotocamera o il microfono da utilizzare e impostare la sensibilità del microfono.

- Impostazione della memorizzazione degli oggetti condivisi — L'utente è in grado di selezionare la quantità di spazio su disco che un dominio può utilizzare per memorizzare oggetti condivisi persistenti. Tali impostazioni possono essere specificate per un numero indefinito di domini specifici, inoltre è possibile configurare impostazioni predefinite per i nuovi domini. Il limite predefinito è 100 KB di spazio su disco. Per ulteriori informazioni sugli oggetti condivisi persistenti, vedere la classe SharedObject in *Guida di riferimento del linguaggio e ai componenti ActionScript 3.0*.

NOTA

Qualsiasi impostazione che riguarda il file mms.cfg (vedere "Controlli amministratore" a pagina 812) non viene riportata in Gestione impostazioni.

Per ulteriori informazioni su Gestione impostazioni, vedere www.adobe.com/go/settingsmanager_it.

Directory User Flash Player Trust

Le applicazioni di utenti e installatori sono in grado di registrare file SWF locali specifici come affidabili. Tali file SWF vengono assegnati alla sandbox locale affidabile. Questi file possono interagire con qualunque altro file SWF e possono caricare dati sia in remoto che in locale. I file vengono designati dall'utente come affidabili all'interno della directory User Flash Player Trust, che si trova nella stessa directory dell'area di memorizzazione degli oggetti condivisi di Flash, nei seguenti percorsi (specifici in base all'utente corrente):

- Windows: app data\Macromedia\Flash Player\#Security\FlashPlayerTrust

  (ad esempio, C:\Documents and Settings\JohnD\Dati applicazioni\Adobe\Flash Player\#Security\FlashPlayerTrust)
- Mac: app data/Macromedia/Flash Player/#Security/FlashPlayerTrust

  (ad esempio, /Users/JohnD/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust)

Tali impostazioni riguardano unicamente l'utente corrente e non gli altri utenti che accedono al computer. Se un utente che non dispone di privilegi amministrativi installa un'applicazione nella propria porzione del sistema, la directory User Flash Player Trust consente al programma di installazione di registrare l'applicazione come affidabile per tale utente.

Gli sviluppatori che distribuiscono un file SWF eseguito a livello locale mediante un'applicazione di installazione possono fare in modo che tale applicazione aggiunga un file di configurazione alla directory User Flash Player Trust, al fine di garantire la totale affidabilità del file che stanno distribuendo. Anche in questa situazione, la directory User Flash Player Trust è considerata un controllo utente, in quanto avviata da un'azione (installazione) eseguita dall'utente.

Esiste anche una directory Global Flash Player Trust utilizzata da utenti o installatori con privilegi amministrativi per registrare applicazioni affidabili per tutti gli utenti di un computer (vedere "Controlli amministratore" a pagina 812).

Controlli del sito Web (file di criteri dei domini)

Per rendere i dati di un server Web disponibili ai file SWF di altri domini, è possibile creare un file di criteri dei domini sul server. Il *file di criteri dei domini* è un file XML che fornisce al server la possibilità di indicare che i propri dati e documenti sono disponibili per i file SWF provenienti da specifici domini o da tutti i domini. A qualsiasi file SWF gestito da un dominio specificato dal file di criteri del server è consentito l'accesso a dati o alle risorse provenienti da tale server.

I file di criteri dei domini influenzano l'accesso a vari tipi di risorse, incluso le seguenti:

- Dati in bitmap, file audio e video
- Caricamento di file XML e di testo
- Accesso a connessioni socket e connessioni socket XML
- Importazione di file SWF da altri domini di sicurezza nel dominio di sicurezza del file SWF che esegue il caricamento

Maggiori informazioni a riguardo sono contenute nella parte rimanente di questo capitolo.

Sintassi del file dei criteri

Nell'esempio seguente è riportato un file di criteri che consente l'accesso a file SWF appartenenti a `*.example.com`, `www.friendOfExample.com` e `192.0.34.166`:

```
<?xml version="1.0"?>
<cross-domain-policy>
  <allow-access-from domain="*.example.com" />
  <allow-access-from domain="www.friendOfExample.com" />
  <allow-access-from domain="192.0.34.166" />
</cross-domain-policy>
```

Quando un file SWF tenta di accedere a dati da un dominio diverso, Flash Player tenta automaticamente di caricare un file di criteri dal quel dominio. Se il dominio del file SWF che tenta di accedere ai dati è compreso nel file dei criteri, i dati sono accessibili automaticamente.

Per impostazione predefinita, i file di criteri devono essere denominati `crossdomain.xml` e devono risiedere nella directory principale del server. Tuttavia, un file SWF è in grado di cercare un nome differente o all'interno di una directory differente mediante il metodo `Security.loadPolicyFile()`. Un file di criteri dei domini è applicabile unicamente alla directory dalla quale è stato caricato e alle sue directory secondarie. Di conseguenza, un file di criteri che si trova nella directory principale è applicabile all'intero server, mentre un file di criteri caricato da una qualsiasi sottodirectory è applicabile unicamente a tale directory e alle sue directory secondarie.

Un file di criteri influisce solo sull'accesso al server in cui risiede. Un file di criteri che si trova, ad esempio, all'indirizzo https://www.adobe.com:8080/crossdomain.xml sarà valido solo per le chiamate per il caricamento di dati eseguite a www.adobe.com tramite HTTPS sulla porta 8080.

Un file di criteri dei domini contiene un solo tag `<cross-domain-policy>` che, a sua volta, contiene zero o più tag `<allow-access-from>`. Ogni tag `<allow-access-from>` contiene un attributo `domain` che specifica un indirizzo IP esatto, un dominio esatto o un dominio carattere jolly (ovvero qualsiasi dominio). I domini carattere jolly sono indicati da un solo asterisco (*) che corrisponde a tutti i domini e a tutti gli indirizzi IP oppure da un asterisco seguito da un suffisso che corrisponde a tutti i domini che terminano con il suffisso specificato. I suffissi devono iniziare con un punto. I domini carattere jolly con suffissi possono tuttavia essere rappresentati da domini costituiti solo dal suffisso senza il punto iniziale, ad esempio, foo.com è considerato parte di *.foo.com. I caratteri jolly non sono consentiti quando vengono specificati domini IP.

Se si specifica un indirizzo IP, l'accesso è consentito solo ai file SWF caricati da quell'indirizzo IP utilizzando la sintassi IP (ad esempio http://65.57.83.12/flashmovie.swf) e non ai file caricati utilizzando la sintassi del nome del dominio. Flash Player non esegue la risoluzione DNS.

È possibile consentire l'accesso a documenti appartenenti a qualsiasi dominio, come illustrato di seguito:

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```

Ogni tag `<allow-access-from>` dispone dell'attributo opzionale `secure` che ha come valore predefinito `true`. È possibile impostarlo su `false` se il file di criteri si trova su un server HTTPS e si desidera consentire ai file SWF di un server non HTTPS di caricare dati dal server HTTPS.

L'impostazione dell'attributo `secure` su `false` potrebbe compromettere la sicurezza garantita dal protocollo HTTPS. In particolare, l'impostazione di questo attributo su `false` consente di aprire il contenuto sicuro ad attacchi di snooping e spoofing. Adobe sconsiglia vivamente di impostare l'attributo `secure` su `false`.

Se i dati da caricare si trovano su un server HTTPS, ma il file SWF che esegue il caricamento si trova su un server HTTP, Adobe consiglia di spostare il file SWF su un server HTTPS, in modo da poter conservare tutte le copie dei dati sotto la protezione del server HTTPS. Tuttavia, se si decide di mantenere il file SWF su un server HTTP, aggiungere l'attributo `secure="false"` al tag `<allow-access-from>`, come riportato nel codice seguente:

```
<allow-access-from domain="www.example.com" secure="false" />
```

Utilizzare un file di criteri che non contiene tag `<allow-access-from>` equivale a non utilizzare alcun criterio su un server.

File dei criteri socket

Gli oggetti di ActionScript sono in grado di creare istanze di due diversi tipi di connessione server: connessioni server basate su documenti e connessioni socket. Gli oggetti ActionScript quali Loader, Sound, URLLoader e URLStream sono in grado di creare istanze di connessioni server bastate su documenti che, a loro volta, possono caricare file da un URL. Gli oggetti ActionScript Socket e XMLSocket sono invece in grado di effettuare connessioni socket, che operano con streaming di dati e non con documenti caricati. Flash Player supporta due tipi di file di criteri: i file di criteri basati su documenti e i file di criteri socket. Le connessioni basate su documenti necessitano di file di criteri basati su documenti, mentre le connessioni socket richiedono file di criteri socket.

Per Flash Player è necessario che un file di criteri venga trasmesso con lo stesso tipo di protocollo utilizzato dalla connessione. Ad esempio, se si colloca un file di criteri sul server HTTP, i file SWF di altri domini possono caricare dati da esso come da un qualsiasi server HTTP. Tuttavia, se non si fornisce un file di criteri socket allo stesso server, ai file SWF di altri domini non sarà consentito connettersi al server a livello di socket. Il sistema mediante il quale un file di criteri socket viene recuperato deve corrispondere al sistema mediante il quale viene eseguita la connessione.

Un file dei criteri gestito da un server socket ha la stessa sintassi di qualsiasi altro file dei criteri, con la differenza che questo file specifica anche le porte a cui viene garantito l'accesso. Un file dei criteri proveniente da una porta con numero inferiore a 1024 può autorizzare l'accesso a qualsiasi porta; un file proveniente da una porta 1024 o superiore può autorizzare l'accesso alle porte 1024 e superiori. Le porte consentite vengono specificate in un attributo `to-ports` del tag `<allow-access-from>`. Sono consentiti i numeri di porta singoli, gli intervalli di porte e i caratteri jolly.

Segue un esempio di file dei criteri XMLSocket:

```
<cross-domain-policy>
   <allow-access-from domain="*" to-ports="507" />
   <allow-access-from domain="*.example.com" to-ports="507,516" />
   <allow-access-from domain="*.example2.com" to-ports="516-523" />
   <allow-access-from domain="www.example2.com" to-ports="507,516-523" />
   <allow-access-from domain="www.example3.com" to-ports="*" />
</cross-domain-policy>
```

Quando i file di criteri sono stati introdotti per la prima volta in Flash Player 6, i file di criteri socket non erano supportati. Le connessioni ai server socket venivano autorizzate da un file di criteri contenuto nel percorso predefinito del file di criteri dei domini su un server HTTP collegato alla porta 80 dello stesso host del server socket. Per consentire la conservazione delle disposizioni server esistenti, Flash Player 9 supporta ancora tale funzionalità. Tuttavia, l'impostazione predefinita di Flash Player ora prevede il recupero di un file di criteri socket sulla stessa porta della connessione socket. Se si desidera utilizzare un file di criteri basato su HTTP per autorizzare connessioni socket, è necessario richiedere esplicitamente il file di criteri HTTP mediante un codice simile al seguente:

```
Security.loadPolicyFile("http://socketServerHost.com/crossdomain.xml")
```

Inoltre, per autorizzare connessioni socket, il file di criteri HTTP deve provenire unicamente dallo stesso percorso predefinito del file di criteri dei domini e non da altri percorsi HTTP. Un file di criteri ottenuto da un server HTTP autorizza implicitamente l'accesso socket a tutte le porte 1024 e superiori; qualsiasi attributo `to-ports` presente nel file di criteri HTTP viene ignorato.

Per ulteriori informazioni sui file di criteri socket, vedere "Connessione a socket" a pagina 840.

Precaricamento dei file di criteri

Caricare dati da un server o connettersi a un socket è un'operazione asincrona, di conseguenza Flash Player attende che il file di criteri dei domini termini di scaricare prima di iniziare l'operazione principale. Tuttavia, l'estrazione di dati pixel da immagini o di dati campione da file audio è un'operazione sincrona; per poter estrarre i dati, è necessario che il file di criteri dei domini venga prima caricato. Quando si caricano i file multimediali, è necessario specificare che venga verificato un file di criteri dei domini:

- Se si usa il metodo `Loader.load()`, impostare la proprietà `checkPolicyFile` del parametro `context`, che è un oggetto LoaderContext.
- Se si incorpora un'immagine in un campo di testo mediante il tag `<img>`, impostare l'attributo `checkPolicyFile` del tag `<img>` su `"true"`, come nell'esempio seguente: `<img checkPolicyFile = "true" src = "example.jpg">`.

- Se si usa il metodo `Sound.load()`, impostare la proprietà `checkPolicyFile` del parametro `context`, che è un oggetto SoundLoaderContext.
- Se si usa la classe NetStream, impostare la proprietà `checkPolicyFile` dell'oggetto NetStream.

Quando si imposta uno di questi parametri, Flash Player verifica per prima cosa la presenza di file di criteri già scaricati per tale dominio. Quindi prende in esame eventuali chiamate in attesa al metodo `Security.loadPolicyFile()` per vedere se sono nell'area di validità e, in caso affermativo, attende che vengano eseguite. Infine cerca il file di criteri dei domini nel percorso predefinito all'interno del server.

Controlli creatore (sviluppatore)

L'API di ActionScript principale utilizzata per concedere privilegi di sicurezza è il metodo `Security.allowDomain()`, che consente di concedere privilegi a file SWF nel dominio specificato. Nell'esempio seguente, un file SWF concede l'accesso a file SWF gestiti dal dominio www.example.com:

```
Security.allowDomain("www.example.com")
```

Questo metodo consente di concedere autorizzazioni per:

- Scambio di script tra file SWF (vedere "Scambio di script" a pagina 832)
- Accesso a elenchi di visualizzazione (vedere "Lettura dell'elenco di visualizzazione" a pagina 835)
- Rilevamento di eventi (vedere "Sicurezza eventi" a pagina 836)
- Accesso completo a proprietà e metodi dell'oggetto Stage (vedere "Stage, sicurezza" a pagina 834)

Lo scopo primario di una chiamata del metodo `Security.allowDomain()` è concedere a file SWF di un dominio esterno l'autorizzazione a inviare script al file SWF che chiama il metodo `Security.allowDomain()`. Per ulteriori informazioni, vedere "Scambio di script" a pagina 832.

Se si specifica un indirizzo IP come parametro del metodo `Security.allowDomain()`, non viene consentito l'accesso a tutte le parti che accedono dall'indirizzo IP specificato. Al contrario, viene consentito solo l'accesso da una parte che contiene l'indirizzo IP specificato nel proprio URL, anziché il nome di dominio mappato sull'indirizzo IP. Ad esempio, se il nome dominio www.example.com viene mappato sull'indirizzo IP 192.0.34.166, una chiamata a `Security.allowDomain("192.0.34.166")` non garantisce l'accesso a www.example.com.

Per consentire l'accesso da tutti i domini, è possibile trasmettere il carattere jolly `"*"` al metodo `Security.allowDomain()`. Poiché in tal modo si consente a file SWF di *tutti* i domini di inviare script al file SWF chiamante, si raccomanda di utilizzare il carattere jolly `"*"` con estrema cautela.

ActionScript include un'API di autorizzazione secondaria chiamata `Security.allowInsecureDomain()`. Questo metodo funziona esattamente come il metodo `Security.allowDomain()`, con la differenza che, se chiamato da un file SWF gestito da una connessione HTTPS protetta, esso consente anche l'accesso al file SWF chiamante da parte di altri file SWF gestiti da un protocollo non protetto, quale HTTP. Tuttavia, non è consigliabile consentire lo scambio di script tra file appartenenti a un protocollo protetto (HTTPS) e file appartenenti a protocolli non protetti (come HTTP), in quanto si potrebbe aprire contenuto protetto ad attacchi di snooping e spoofing. Ecco come funzionano tali attacchi: poiché il metodo `Security.allowInsecureDomain()` consente l'accesso ai dati HTTPS protetti da parte di file SWF gestiti su connessioni HTTP, chi effettua l'attacco si interpone tra il server HTTP e gli utenti ed è in grado di sostituire il file SWF HTTP con un suo file, che può così accedere ai dati HTTPS protetti.

Un altro metodo di sicurezza importante è il metodo `Security.loadPolicyFile()`, che fa in modo che Flash Player verifichi un file di criteri dei domini in un percorso non standard. Per ulteriori informazioni, vedere "Controlli del sito Web (file di criteri dei domini)" a pagina 816.

Funzioni di sicurezza sandbox

I computer client possono ottenere singoli file SWF da varie fonti, quali siti Web esterni o file system locale. Flash Player assegna singolarmente file SWF e altre risorse, quali oggetti condivisi, bitmap, file audio, video e di dati, a funzioni di sicurezza sandbox in base alla loro origine, nel momento in cui vengono caricati in Flash Player. Nelle sezioni che seguono vengono descritte le regole, applicate da Flash Player, che determinano a che cosa può accedere un file SWF contenuto in una data sandbox.

Per ulteriori informazioni sulle funzioni di sicurezza sandbox, vedere il white paper sulla sicurezza di *Flash Player 9*.

Funzioni di sicurezza sandbox remote

Flash Player classifica le risorse (inclusi i file SWF) provenienti da Internet in sandbox separate corrispondenti ai relativi domini di origine del sito Web. Per impostazione predefinita, questi file sono autorizzati ad accedere a risorse appartenenti al proprio server. Ai file SWF remoti può essere consentito l'accesso a dati aggiuntivi appartenenti ad altri domini da siti Web e autorizzazioni del creatore specifiche, quali file di criteri dei domini e il metodo `Security.allowDomain()`. Per ulteriori informazioni, vedere "Controlli del sito Web (file di criteri dei domini)" a pagina 816 e "Controlli creatore (sviluppatore)" a pagina 820.

I file SWF remoti non possono caricare file o risorse locali.

Per ulteriori informazioni, vedere il white paper sulla sicurezza di *Flash Player 9*.

Funzioni di sicurezza sandbox locali

Si definisce *file locale* qualunque file al quale si fa riferimento mediante il protocollo `file:` o un percorso UNC (Universal Naming Convention). I file SWF locali si trovano in una delle tre sandbox locali:

- Sandbox locale con file system — Per ragioni di sicurezza, Flash Player colloca tutte le risorse e i file SWF locali nella sandbox locale con file system per impostazione predefinita. Da questa sandbox, i file SWF sono in grado di leggere file locali (mediante la classe URLLoader, ad esempio), ma non possono comunicare in alcun modo con la rete. Ciò garantisce all'utente che i dati locali non possano essere diffusi nella rete o altrimenti condivisi in modo inadeguato.

- Sandbox locale con rete — Quando si compila un file SWF, è possibile specificare che esso abbia accesso alla rete se eseguito come file locale (vedere "Impostazione del tipo di sandbox per file SWF locali" a pagina 824). Questi file vengono collocati nella sandbox locale con rete. I file SWF assegnati alla sandbox locale con rete rinunciano all'accesso ai file locali. In compenso, ai file SWF è consentito l'accesso ai dati della rete. Tuttavia, un file SWF locale con rete non può comunque leggere dati ottenuti dalla rete se non sono presenti autorizzazioni per tale operazione, quali un file di criteri dei domini o una chiamata al metodo `Security.allowDomain()`. Per concedere tale autorizzazione, un file di criteri dei domini deve concedere l'autorizzazione di accesso a *tutti* i domini mediante `<allow-access-from domain="*"/>` o `Security.allowDomain("*")`. Per ulteriori informazioni, vedere "Controlli del sito Web (file di criteri dei domini)" a pagina 816 e "Controlli creatore (sviluppatore)" a pagina 820.
- Sandbox locale affidabile — I file SWF locali registrati come affidabili (da programmi di utenti o installatori) vengono collocati nella sandbox locale affidabile. Gli amministratori di sistema e gli utenti possono riassegnare o spostare un file SWF locale alla o dalla sandbox locale affidabile in base a considerazioni di sicurezza (vedere "Controlli amministratore" a pagina 812 e "Controlli utente" a pagina 814). I file SWF assegnati alla sandbox locale affidabile possono interagire con qualunque altro file SWF e caricare i dati da qualsiasi postazione (remota o locale).

Le comunicazioni tra la sandbox locale con rete e la sandbox locale con file system, così come le comunicazioni tra sandbox locale con file system e sandbox remote, sono assolutamente proibite. Le autorizzazioni per tali comunicazioni non possono essere concesse dall'applicazione Flash o da utenti o amministratori.

L'invio di script in entrambe le direzioni tra file HTML locali e file SWF locali (ad esempio, mediante la classe ExternalInterface) richiede che sia il file HTML che il file SWF si trovino nella stessa sandbox locale affidabile. I modelli di sicurezza locali dei browser infatti sono differenti dal modello di sicurezza locale di Flash Player.

I file SWF nella sandbox locale con rete non possono caricare file SWF nella sandbox locale con file system. I file SWF nella sandbox locale con file system non possono caricare file SWF nella sandbox locale con rete.

Impostazione del tipo di sandbox per file SWF locali

È possibile configurare un file SWF per la sandbox locale con file system o per la sandbox locale con rete effettuando le impostazioni di pubblicazione del documento nello strumento di creazione Adobe Flash CS3 Professional. Per ulteriori informazioni, vedere "Impostazione delle opzioni di pubblicazione per il formato file SWF di Flash" nella guida *Uso di Flash*.

Un utente finale o l'amministratore di un computer possono classificare un file SWF locale come affidabile e consentire a tale file di caricare dati da tutti i domini, sia locali che di rete. Tale informazione viene specificata nelle directory Global Flash Player Trust e User Flash Player Trust. Per ulteriori informazioni, vedere "Controlli amministratore" a pagina 812 e "Controlli utente" a pagina 814.

Per ulteriori informazioni sulle funzioni di sicurezza sandbox locali, vedere "Funzioni di sicurezza sandbox locali" a pagina 822.

Proprietà Security.sandboxType

Il creatore di un file SWF può utilizzare la proprietà statica di sola lettura `Security.sandboxType` per determinare a quale tipo di sandbox Flash Player ha assegnato il file SWF. La classe Security include costanti che rappresentano possibili valori della proprietà `Security.sandboxType`, come indicato di seguito:

- `Security.REMOTE` — Il file SWF deriva da un URL Internet e viene gestito tramite regole sandbox basate sul dominio.
- `Security.LOCAL_WITH_FILE` — Questo file SWF è un file locale che non è considerato affidabile dall'utente e non è stato pubblicato con una designazione di rete. Il file SWF può leggere le origini dati locali, ma non può comunicare con Internet.
- `Security.LOCAL_WITH_NETWORK` — Questo file SWF è un file locale che non è considerato affidabile dall'utente, ma è stato pubblicato con una designazione di rete. Il file SWF può comunicare con Internet, ma non può leggere da origini dati locali.
- `Security.LOCAL_TRUSTED` — Questo file SWF è un file locale che è stato considerato affidabile dall'utente tramite Gestione impostazioni o il file di configurazione Flash Player Trust. Il file SWF può leggere le origini dati locali e comunicare con Internet.

Limitazioni delle API di connettività di rete

È possibile controllare l'accesso di un file SWF alle funzionalità di rete impostando il parametro `allowNetworking` dei tag `<object>` e `<embed>` nella pagina HTML in cui si trova il contenuto SWF.

I valori possibili di `allowNetworking` sono:

- `"all"` (predefinito) — Tutte le API di connettività sono ammesse nel file SWF.
- `"internal"` — Il file SWF non può chiamare le API di navigazione o interazione del browser, elencate più avanti in questa sezione, ma può chiamare qualunque altra API di connettività di rete.
- `"none"` — Il file SWF non può chiamare le API di navigazione o interazione del browser, elencate più avanti in questa sezione, e non può utilizzare API di comunicazione tra file SWF, anch'esse descritte più avanti.

La chiamata di un'API non consentita provoca un'eccezione SecurityError.

Per impostare il parametro `allowNetworking`, aggiungere, nei tag `<object>` e `<embed>` della pagina HTML che include un riferimento al file SWF, il parametro `allowNetworking` e impostarne il valore, come nell'esempio seguente:

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
 codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
  swflash.cab#version=9,0,18,0"
 width="600" height="400" id="test" align="middle">
<param name="allowNetworking" value="none" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowNetworking="none" bgcolor="#333333"
  width="600" height="400"
  name="test" align="middle" type="application/x-shockwave-flash"
  pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

Una pagina HTML può anche usare uno script per generare tag di incorporamento SWF. È necessario modificare lo script in modo da inserire le impostazioni `allowNetworking` corrette. Le pagine HTML generate da Flash e Adobe Flex Builder impiegano la funzione `AC_FL_RunContent()` per incorporare riferimenti a file SWF; l'utente deve aggiungere le impostazioni del parametro `allowNetworking` come indicato di seguito:

```
AC_FL_RunContent( ... "allowNetworking", "none", ...)
```

Le seguenti API sono disabilitate quando `allowNetworking` è impostato su `"internal"`:

- `navigateToURL()`
- `fscommand()`
- `ExternalInterface.call()`

Oltre alle API elencate sopra, le seguenti API sono disabilitate quando `allowNetworking` è impostato su `"none"`:

- `sendToURL()`
- `FileReference.download()`
- `FileReference.upload()`
- `Loader.load()`
- `LocalConnection.connect()`
- `LocalConnection.send()`
- `NetConnection.connect()`
- `NetStream.play()`
- `Security.loadPolicyFile()`
- `SharedObject.getLocal()`
- `SharedObject.getRemote()`
- `Socket.connect()`
- `Sound.load()`
- `URLLoader.load()`
- `URLStream.load()`
- `XMLSocket.connect()`

Anche se l'impostazione `allowNetworking` selezionata consente a un file SWF di utilizzare un'API di connettività di rete, potrebbero essere attive altre restrizioni dovute ai limiti della sicurezza sandbox, come descritto in questo capitolo.

Se `allowNetworking` è impostato su `"none"`, non è possibile fare riferimento a file multimediali esterni in un tag `<img>` nella proprietà `htmlText` di un oggetto TextField (viene generata un'eccezione SecurityError).

Se `allowNetworking` è impostato su `"none"`, un simbolo appartenente a una libreria condivisa importata inserito nello strumento di creazione di Flash (non in ActionScript) viene disabilitato in fase di runtime.

Sicurezza modalità a schermo intero

Flash Player 9.0.27.0 e successivi supportano la modalità a schermo intero, che, se attivata, consente di visualizzare il contenuto di Flash nell'intero schermo. Per attivare la modalità a schermo intero, impostare la proprietà `displayState` dello stage sulla costante `StageDisplayState.FULL_SCREEN`. Per ulteriori informazioni, vedere "Uso della modalità a schermo intero" a pagina 417.

Per i file SWF eseguiti in un browser, è necessario tenere presente alcune considerazioni relative alla sicurezza.

Per attivare la modalità a schermo intero nei tag `<object>` e `<embed>` di una pagina HTML contenente un riferimento al file SWF, aggiungere il parametro `allowFullScreen` con il valore impostato su `"true"` (il valore predefinito è `"false"`), come illustrato nell'esempio seguente:

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
 codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
  swflash.cab#version=9,0,18,0"
 width="600" height="400" id="test" align="middle">
<param name="allowFullScreen" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
  width="600" height="400"
  name="test" align="middle" type="application/x-shockwave-flash"
  pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

Una pagina HTML può anche usare uno script per generare tag di incorporamento SWF. È necessario modificare lo script in modo da inserire le impostazioni `allowFullScreen` corrette. Le pagine HTML generate da Flash e Flex Builder impiegano la funzione `AC_FL_RunContent()` per incorporare riferimenti a file SWF; l'utente deve aggiungere le impostazioni del parametro `allowFullScreen` come indicato di seguito:

```
AC_FL_RunContent( ... "allowFullScreen", "true", ...)
```

Il parametro ActionScript che avvia la modalità a schermo intero può essere chiamato solo in risposta a un evento associato al mouse o alla tastiera. Se viene chiamato in altre situazione, Flash Player genera un'eccezione.

In modalità a schermo intero, gli utenti non possono immettere testo nei campi di inserimento testo. Tutti gli input di tastiera e le operazioni di tastiera di ActionScript sono disabilitate in modalità a schermo intero, a eccezione delle scelte rapide da tastiera (quale il tasto Esc) che consentono di riportare l'applicazione in modalità normale.

Quando si entra in modalità a schermo intero, viene visualizzato un messaggio che istruisce l'utente su come uscire e tornare in modalità normale. Il messaggio viene visualizzato solo per alcuni secondi, quindi scompare.

Una chiamata alla proprietà `displayState` di un oggetto Stage genera un'eccezione per qualunque chiamante che non si trova nella stessa sicurezza sandbox del titolare dello stage (il file SWF principale). Per ulteriori informazioni, vedere "Stage, sicurezza" a pagina 834.

Gli amministratori possono disattivare la modalità a schermo intero per i file SWF eseguiti nei browser mediante l'impostazione `FullScreenDisable = 1` nel file mms.cfg. Per informazioni dettagliate, vedere "Controlli amministratore" a pagina 812.

Per attivare la modalità a schermo intero in un browser, è necessario che il file SWF sia contenuto in una sola pagina HTML.

La modalità a schermo intero è sempre consentita in lettori autonomi o in un file del proiettore.

Caricamento di contenuto

Un file SWF può caricare i seguenti tipi di contenuto:

- File SWF
- Immagini
- Audio
- Video

Caricamento di file SWF e di immagine

Utilizzare la classe Loader per caricare file SWF e di immagine (file JPG, GIF o PNG). Tutti i file SWF, eccetto quelli che si trovano nella sandbox locale con file system, possono caricare file SWF e di immagine di qualsiasi dominio di rete. Solo i file SWF contenuti nelle sandbox locali possono caricare file SWF e di immagine del file system locale. Tuttavia, i file nella sandbox locale con rete possono caricare unicamente file SWF locali che si trovano nella sandbox locale affidabile o locale con rete. I file SWF nella sandbox locale con rete possono caricare contenuto locale diverso da file SWF (quali immagini), tuttavia, essi non possono accedere ai dati del contenuto caricato.

Quando si carica un file SWF da un'origine non affidabile (ad esempio un dominio diverso da quello del file SWF principale dell'oggetto Loader), può essere opportuno definire una maschera per l'oggetto Loader, in modo da impedire che il contenuto caricato (che è un elemento secondario dell'oggetto Loader) possa essere disegnato in parti dello stage al di fuori della maschera, come nel codice seguente:

```
import flash.display.*;
import flash.net.URLRequest;
var rect:Shape = new Shape();
rect.graphics.beginFill(0xFFFFFF);
rect.graphics.drawRect(0, 0, 100, 100);
addChild(rect);
var ldr:Loader = new Loader();
ldr.mask = rect;
var url:String = "http://www.unknown.example.com/content.swf";
var urlReq:URLRequest = new URLRequest(url);
ldr.load(urlReq);
addChild(ldr);
```

Quando si chiama il metodo `load()` dell'oggetto Loader, è possibile specificare un parametro `context`, che è un oggetto LoaderContext. La classe LoaderContext include tre proprietà che consentono di definire il contesto di utilizzo del contenuto caricato:

- `checkPolicyFile`: Utilizzare questa proprietà solo per caricare file di immagine (non file SWF). Specificare questa proprietà per file di immagine appartenenti a un dominio differente da quello del file contenente l'oggetto Loader. Se questa proprietà viene impostata su `true`, l'oggetto Loader cerca nel server di origine un file di criteri dei domini (vedere "Controlli del sito Web (file di criteri dei domini)" a pagina 816). Se il server concede l'autorizzazione di accesso al dominio Loader, i file SWF del dominio Loader possono accedere ai dati contenuti nell'immagine caricata. In altre parole, è possibile utilizzare la proprietà `Loader.content` per ottenere un riferimento all'oggetto Bitmap che rappresenta l'immagine caricata oppure il metodo `BitmapData.draw()` per accedere ai pixel dell'immagine caricata.

- `securityDomain`: Utilizzare questa proprietà solo per caricare un file SWF (non un'immagine). Specificare questa proprietà per file SWF appartenenti a un dominio differente da quello del file contenente l'oggetto Loader. Per la proprietà `securityDomain` sono al momento supportati solo due valori: `null` (predefinito) e `SecurityDomain.currentDomain`. Se si specifica `SecurityDomain.currentDomain`, il file SWF caricato viene *importato* nella stessa sandbox del file SWF che esegue il caricamento, ovvero funziona come se fosse stato caricato dallo stesso server del file SWF caricante. Ciò è consentito solo se nel server del file SWF caricato si trova un file di criteri dei domini che consente l'accesso da parte del dominio del file SWF che esegue il caricamento. Se il file di criteri non viene trovato, caricante e caricato possono scambiarsi liberamente script una volta avviato il caricamento, in quanto si trovano nella stessa sandbox. Si tenga presente che l'importazione di sandbox può essere sostituita mediante l'esecuzione di un normale caricamento, quindi facendo in modo che il file SWF caricato chiami il metodo `Security.allowDomain()`. Quest'ultimo metodo potrebbe risultare più semplice da usare, in quanto il file SWF caricato si troverebbe nella propria sandbox e sarebbe in grado di accedere alle risorse presenti nel proprio server.
- `applicationDomain`: Utilizzare questa proprietà solo per il caricamento di un file SWF scritto in ActionScript 3.0 (non un'immagine o un file SWF scritto in ActionScript 1.0 o 2.0). Quando si carica il file, è possibile specificare che venga collocato in un particolare dominio dell'applicazione, anziché, per impostazione predefinita, essere inserito nel dominio di una nuova applicazione, che è un dominio secondario del domino dell'applicazione del file SWF che esegue il caricamento. Si tenga presente che i domini di applicazione sono sottounità di domini di sicurezza, quindi è possibile specificare un dominio di applicazione di destinazione solo se il file SWF che si sta caricando appartiene al proprio dominio di sicurezza, o perché proviene dal proprio server o perché è stato importato nel proprio dominio di sicurezza mediante la proprietà `securityDomain`. Se si specifica un dominio di applicazione, ma il file SWF caricato fa parte di un diverso dominio di sicurezza, il dominio specificato in `applicationDomain` viene ignorato. Per ulteriori informazioni, vedere "Uso della classe ApplicationDomain" a pagina 745.

Per informazioni dettagliate, vedere "Impostazione del contesto di caricamento" a pagina 452.

Un'importante proprietà dell'oggetto Loader è la proprietà `contentLoaderInfo`, che è un oggetto LoaderInfo. A differenza della maggior parte degli oggetti, l'oggetto LoaderInfo viene condiviso tra il file SWF che carica e il contenuto caricato ed è sempre accessibile da entrambe le parti. Se il contenuto caricato è un file SWF, esso può accedere all'oggetto LoaderInfo mediante la proprietà `DisplayObject.loaderInfo`. Gli oggetti LoaderInfo includono informazioni quali stato di avanzamento del caricamento, URL di caricante e caricato, relazione di affidabilità tra caricante e caricato e altre informazioni. Per ulteriori informazioni, vedere "Monitoraggio dello stato di avanzamento del caricamento" a pagina 451.

Caricamento di audio e video

Tutti i file SWF, a eccezione di quelli nella sandbox locale con file system, possono caricare audio e video da origini di rete mediante i metodi `Sound.load()`, `NetConnection.connect()` e `NetStream.play()`.

Solo i file SWF locali sono in grado di caricare file multimediali dal file system locale. E solo i file SWF nella sandbox locale con file system o nella sandbox locale affidabile possono accedere ai dati contenuti in tali file caricati.

Esistono altre restrizioni relative all'accesso ai dati dei file multimediali caricati. Per informazioni dettagliate, vedere "Accesso a file multimediali caricati come dati" a pagina 836.

Caricamento di file SWF e di immagini mediante il tag <img> in un campo di testo

È possibile caricare file SWF e bitmap in campi di testo mediante il tag `<img>`, come illustrato dal codice seguente:

```
<img src = 'filename.jpg' id = 'instanceName' >
```

Per accedere al contenuto caricato in questo modo, utilizzare il metodo `getImageReference()` dell'istanza TextField, come nel codice seguente:

```
var loadedObject:DisplayObject =
  myTextField.getImageReference('instanceName');
```

Si noti, tuttavia, che i file SWF e le immagini caricate in questo modo vengono inseriti nella sandbox corrispondente alla rispettiva origine.

Quando si carica un file di immagine utilizzando un tag `<img>` in un campo di testo, l'accesso ai dati di tale file può essere autorizzato da un file di criteri dei domini. Per verificare la presenza di un file di criteri, aggiungere l'attributo `checkPolicyFile` al tag `<img>`, come nel codice seguente:

```
<img src = 'filename.jpg' checkPolicyFile = 'true' id = 'instanceName' >
```

Quando si carica un file SWF mediante un tag `<img>` in un campo di testo, è possibile consentire l'accesso ai dati di tale file SWF chiamando il metodo `Security.allowDomain()`.

Quando si utilizza un tag `<img>` in un campo di testo per caricare un file esterno (anziché utilizzare una classe Bitmap incorporata al file SWF), viene automaticamente creato un oggetto Loader come elemento secondario dell'oggetto TextField e il file esterno viene caricato nel Loader come se fosse stato caricato in ActionScript mediante l'oggetto Loader. In questo caso, il metodo `getImageReference()` restituisce l'oggetto Loader creato automaticamente. Non è necessaria alcuna verifica di sicurezza per accedere a questo Loader poiché si trova nella stessa sandbox di sicurezza del codice chiamante.

Tuttavia, quando si fa riferimento alla proprietà `content` dell'oggetto Loader per accedere ai contenuti multimediali caricati, le regole di sicurezza vengono applicate. Se il contenuto è un'immagine, è necessario implementare un file di criteri dei domini, mentre se il contenuto è un file SWF, è necessario che il codice del file chiami il metodo `allowDomain()`.

Contenuto distribuito mediante server RTMP

Flash Media Server impiega il protocollo RTMP (Real-Time Media Protocol) per gestire dati, audio e video. I file SWF caricano questi file multimediali mediante il metodo `connect()` della classe NetConnection, trasmettendo un URL RTMP come parametro. Flash Media Server è in grado di limitare le connessioni e impedire lo scaricamento di contenuto, in base al dominio del file richiedente. Per ulteriori informazioni, consultare la documentazione di Flash Media Server.

Per i file multimediali caricati da fonti RTMP, non è possibile utilizzare i metodi `BitmapData.draw()` e `SoundMixer.computeSpectrum()` per estrarre dati audio e di immagine in fase di runtime.

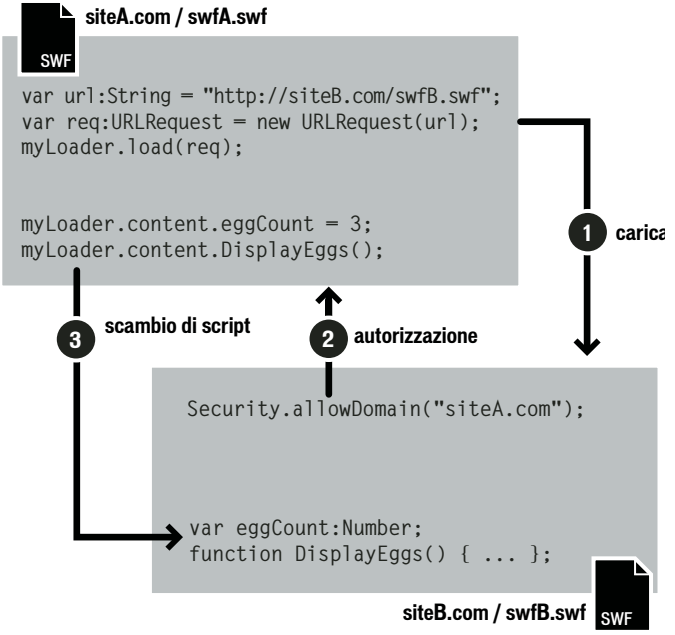
Scambio di script

Se due file SWF sono stati creati con ActionScript 3.0 e sono gestiti dallo stesso dominio (ad esempio, l'URL di un file SWF è http://www.example.com/swfA.swf e l'URL dell'altro è http://www.example.com/swfB.swf) un file SWF può esaminare e modificare variabili, oggetti, proprietà e metodi dell'altro e vice versa. Questa operazione è detta *scambio di script*.

Lo scambio di script non è supportato tra file SWF AVM1 e AVM2. I file SWF AVM1 sono file creati con ActionScript 1.0 o ActionScript 2.0 (la dicitura AVM1 e AVM2 è riferita ad ActionScript Virtual Machine). È tuttavia possibile utilizzare la classe LocalConnection per scambiare dati tra AVM1 e AVM2.

Se due file SWF scritti con ActionScript 3.0 appartengono a domini diversi, ad esempio http://siteA.com/swfA.swf e http://siteB.com/siteB.swf, per impostazione predefinita Flash Player non permette a swfA.swf di inviare script a swfB.swf e a swfB.swf di inviare script a swfA.swf. Il file SWF permette ai file SWF di altri domini di scambiare script tramite la chiamata alla funzione `Security.allowDomain()`. La chiamata a `Security.allowDomain("siteA.com")` consente a swfB.swf di ricevere script dai file SWF di siteA.com.

In qualsiasi configurazione tra domini, sono coinvolte due parti ed è importante chiarire il ruolo di ognuna. Ai fini di questa discussione, il lato che esegue lo scambio di script è detto *parte che accede* (di solito il file SWF che esegue l'accesso), mentre l'altro lato è detto *parte a cui si accede* (di solito il file SWF a cui si accede). Quando siteA.swf invia script a siteB.swf, siteA.swf è la parte che accede, mentre siteB.swf la parte a cui si accede, come illustrato dal diagramma seguente:



Le autorizzazioni tra domini definite tramite il metodo `Security.allowDomain()` sono asimmetriche. Nell'esempio precedente, siteA.swf può inviare script a siteB.swf, ma siteB.swf non può inviare script a siteA.swf, poiché siteA.swf non ha eseguito la chiamata al metodo `Security.allowDomain()` per dare ai file SWF di siteB.com l'autorizzazione a inviargli script. Per impostare autorizzazioni simmetriche, è necessario che entrambi i file SWF chiamino il metodo `Security.allowDomain()`.

Oltre a proteggere i file SWF dall'esecuzione di script tra domini generati da altri file SWF, Flash Player protegge i file SWF dall'esecuzione di script tra domini generati da file HTML. La trasmissione di script da file HTML a SWF avviene tramite callback gestiti mediante il metodo `ExternalInterface.addCallback()`. Quando gli script da HTML a SWF superano il dominio, il file SWF a cui si accede deve chiamare il metodo `Security.allowDomain()`, esattamente come quando la parte che accede è un file SWF; in caso contrario, l'operazione non viene completata. Per ulteriori informazioni, vedere "Controlli creatore (sviluppatore)" a pagina 820.

Flash Player prevede inoltre controlli di sicurezza per la trasmissione di script da SWF a HTML. Per ulteriori informazioni, vedere "Controllo dell'accesso a script in una pagina Web host" a pagina 845.

Stage, sicurezza

Alcune proprietà e metodi dell'oggetto Stage sono disponibili a tutti gli oggetti sprite o clip filmato presenti nell'elenco di visualizzazione.

Tuttavia, l'oggetto Stage presenta un proprio titolare: vale a dire il primo file SWF caricato. Per impostazione predefinita, le proprietà e i metodi seguenti dell'oggetto Stage sono disponibili unicamente ai file SWF che si trovano nella stessa sandbox di sicurezza del titolare dello stage:

Proprietà		Metodi
`align`	`showDefaultContextMenu`	`addChild()`
`displayState`	`stageFocusRect`	`addChildAt()`
`frameRate`	`stageHeight`	`addEventListener()`
`height`	`stageWidth`	`dispatchEvent()`
`mouseChildren`	`tabChildren`	`hasEventListener()`
`numChildren`	`textSnapshot`	`setChildIndex()`
`quality`	`width`	`willTrigger()`
`scaleMode`		

Perché un file SWF in una sandbox diversa da quella del titolare dello stage possa accedere a queste proprietà e metodi, il file SWF titolare dello stage deve chiamare il metodo `Security.allowDomain()` per consentire l'accesso a domini di una sandbox esterna. Per ulteriori informazioni, vedere "Controlli creatore (sviluppatore)" a pagina 820.

La proprietà `frameRate` rappresenta un caso particolare, in quanto qualsiasi file SWF è in grado di leggere la proprietà `frameRate`. Tuttavia, solo i file che si trovano nella stessa funzione di sicurezza sandbox del titolare dello stage (o i file ai quali è stata concessa l'autorizzazione di accesso mediante il metodo `Security.allowDomain()`) possono modificare tale proprietà.

Esistono inoltre limitazioni relative ai metodi `removeChildAt()` e `swapChildrenAt()` dell'oggetto stage, ma si tratta di restrizioni di tipo differente. Anziché essere nello stesso dominio del titolare dello stage, per chiamare questi metodi è necessario che il codice si trovi nello stesso dominio del titolare degli oggetti secondari interessati, oppure tali oggetti secondari possono chiamare il metodo `Security.allowDomain()`.

Lettura dell'elenco di visualizzazione

La capacità di un file SWF di accedere a oggetti di visualizzazione caricati da altre sandbox è limitata. Perché un file SWF possa accedere a oggetti di visualizzazione creati da un altro file SWF in una sandbox differente, è necessario che il file SWF al quale si accede chiami il metodo `Security.allowDomain()` per consentire l'accesso da parte del dominio del file SWF che accede. Per ulteriori informazioni, vedere "Controlli creatore (sviluppatore)" a pagina 820.

Per accedere a un oggetto Bitmap che è stato caricato da un oggetto Loader, è necessario che ci sia un file di criteri dei domini nel server di origine del file di immagine e che tale file di criteri dei domini conceda l'autorizzazione di accesso al dominio del file SWF che tenta di accedere all'oggetto Bitmap (vedere "Controlli del sito Web (file di criteri dei domini)" a pagina 816).

L'oggetto LoaderInfo corrispondente al file caricato (e all'oggetto Loader) include le tre seguenti proprietà, che definiscono la relazione tra l'oggetto caricato e l'oggetto Loader: `childAllowsParent`, `parentAllowsChild` e `sameDomain`.

Sicurezza eventi

Gli eventi collegati all'elenco di visualizzazione presentano limitazioni di accesso di sicurezza, in base alla sandbox dell'oggetto di visualizzazione che invia l'evento. Un evento nell'elenco di visualizzazione presenta fasi di bubbling e di cattura (descritte in Capitolo 10, "Gestione degli eventi" a pagina 335). Durante le fasi di bubbling e di cattura, l'evento migra dall'oggetto di visualizzazione di origine attraverso oggetti di visualizzazione principali nell'elenco di visualizzazione. Se un oggetto principale si trova in una funzione di sicurezza sandbox diversa da quella dell'oggetto di visualizzazione di origine, le fasi di cattura e bubbling si interrompono al di sotto dell'oggetto principale, a meno che non vi sia un rapporto di mutua fiducia tra il titolare dell'oggetto principale e il titolare dell'oggetto di origine. Tale rapporto di mutua fiducia può essere ottenuto come segue:

1. Il file SWF titolare dell'oggetto principale deve chiamare il metodo `Security.allowDomain()` per considerare attendibile il dominio del file SWF titolare dell'oggetto di origine.
2. Il file SWF titolare dell'oggetto di origine deve chiamare il metodo `Security.allowDomain()` per considerare attendibile il dominio del file SWF titolare dell'oggetto di principale.

L'oggetto LoaderInfo corrispondente al file caricato (e all'oggetto Loader) include le due seguenti proprietà, che definiscono la relazione tra l'oggetto caricato e l'oggetto Loader: `childAllowsParent` e `parentAllowsChild`.

Per gli eventi che vengono inviati da oggetti diversi da oggetti di visualizzazione, non sono previste verifiche o altre implicazioni di sicurezza.

Accesso a file multimediali caricati come dati

Per accedere ai dati caricati è possibile utilizzare metodi quali `BitmapData.draw()` e `SoundMixer.computeSpectrum()`. Per impostazione predefinita, un file SWF appartenente a una data funzione di sicurezza sandbox non può ottenere dati pixel o audio da oggetti grafici o audio di cui è stato eseguito il rendering o che sono stati riprodotti da file multimediali di un'altra sandbox. Tuttavia, per ottenere tale autorizzazione è possibile utilizzare i seguenti metodi:

- In un file SWF caricato, chiamare il metodo `Security.allowDomain()` per consentire l'accesso ai dati da parte di file SWF di altri domini.

- Per immagini, audio o video caricati, aggiungere un file di criteri dei domini nel server del file caricato. Tale file di criteri concede l'accesso al dominio del file SWF che tenta di chiamare i metodi `BitmapData.draw()` o `SoundMixer.computeSpectrum()` per estrarre dati dal file.

Nelle sezioni seguenti sono contenute maggiori informazioni sull'accesso a dati bitmap, audio e video.

Accesso ai dati bitmap

Il metodo `draw()` di un oggetto BitmapData consente di disegnare i pixel visualizzati di un qualsiasi oggetto di visualizzazione sull'oggetto BitmapData. Sono inclusi i pixel di oggetti MovieClip, Bitmap e di qualsiasi oggetto di visualizzazione. Perché il metodo `draw()` sia in grado di disegnare pixel sull'oggetto BitmapData, è necessario che vengano soddisfatte le seguenti condizioni:

- Nel caso di un oggetto di origine diverso da una bitmap caricata, l'oggetto di origine e (nel caso di un oggetto Sprite o MovieClip) tutti i suoi oggetti secondari devono provenire dallo stesso dominio dell'oggetto che chiama il metodo `draw()` o devono trovarsi in un file SWF accessibile al chiamante del metodo `Security.allowDomain()`.
- Nel caso di un oggetto di origine bitmap caricata, l'oggetto di origine deve provenire dallo stesso dominio dell'oggetto che chiama il metodo `draw()` oppure il suo server di origine deve contenere un file di criteri dei domini che conceda l'autorizzazione al dominio del chiamante.

Se queste condizioni non vengono soddisfatte, viene generata un'eccezione SecurityError.

Quando si carica l'immagine mediante il metodo `load()` della classe Loader, è possibile specificare un parametro `context`, che è un oggetto LoaderContext. Se la proprietà `checkPolicyFile` dell'oggetto LoaderContext viene impostata su `true`, Flash Player verifica la presenza di un file di criteri validi per domini diversi sul server da cui viene caricata l'immagine. Se esiste un file di criteri dei domini che consente l'accesso al dominio del file SWF da caricare, il file può accedere ai dati dell'oggetto Bitmap; in caso contrario, non vi può accedere.

È inoltre possibile specificare una proprietà `checkPolicyFile` in un'immagine caricata mediante il tag`<img>` inserito in un campo di testo. Per informazioni dettagliate, vedere "Caricamento di file SWF e di immagini mediante il tag <img> in un campo di testo" a pagina 831.

Accesso a dati audio

Le seguenti API relative all'audio di ActionScript 3.0 presentano le seguenti limitazioni di sicurezza:

- Il metodo `SoundMixer.computeSpectrum()` è sempre consentito per i file SWF che si trovano nella stessa funzione di sicurezza sandbox del file audio. Per i file di altre sandbox esistono invece dei controlli di sicurezza.
- Il metodo `SoundMixer.stopAll()` è sempre consentito per i file SWF che si trovano nella stessa funzione di sicurezza sandbox del file audio. Per i file di altre sandbox esistono invece dei controlli di sicurezza.
- La proprietà `id3` della classe Sound è sempre consentita per i file SWF che si trovano nella stessa funzione di sicurezza sandbox del file audio. Per i file di altre sandbox esistono invece dei controlli di sicurezza.

Ogni file audio presenta due tipi di sandbox associate, una per il contenuto e una per il titolare:

- Il dominio di origine del file audio determina la sandbox del contenuto, che, a sua volta, determina se i dati audio possono essere estratti mediante la proprietà `id3` e il metodo `SoundMixer.computeSpectrum()`.
- L'oggetto che avvia la riproduzione audio determina la sandbox del titolare, che, a sua volta, determina se la riproduzione del file audio può essere interrotta mediante il metodo `SoundMixer.stopAll()`.

Quando si carica l'audio mediante il metodo `load()` della classe Sound, è possibile specificare un parametro `context`, che è un oggetto SoundLoaderContext. Se la proprietà `checkPolicyFile` dell'oggetto SoundLoaderContext viene impostata su `true`, Flash Player verifica la presenza di un file di criteri validi per domini diversi sul server da cui viene caricato l'audio. Se esiste un file di criteri dei domini che consente l'accesso al dominio del file SWF da caricare, il file può accedere alla proprietà `id` dell'oggetto Sound; in caso contrario, non vi può accedere. Inoltre, l'impostazione della proprietà `checkPolicyFile` può attivare il metodo `SoundMixer.computeSpectrum()` per i file audio caricati.

È possibile utilizzare il metodo `SoundMixer.areSoundsInaccessible()` per sapere se una chiamata al metodo `SoundMixer.stopAll()` non consente l'interruzione della riproduzione audio perché la sandbox di uno o più titolari non è accessibile al chiamante.

Mediante la chiamata al metodo `SoundMixer.stopAll()` viene interrotta la riproduzione dei file audio la cui sandbox del titolare corrisponde al quella del chiamate di `stopAll()`. Tale metodo consente inoltre di interrompere la riproduzione audio avviata da file SWF che hanno chiamato il metodo `Security.allowDomain()` per consentire l'accesso da parte del dominio del file SWF che chiama il metodo `stopAll()`. Tutte le altre riproduzioni audio non vengono interrotte e la presenza di tali file audio può essere rilevata chiamando il metodo `SoundMixer.areSoundsInaccessible()`.

Se si chiama il metodo `computeSpectrum()`, tutti i file audio in riproduzione devono appartenere alla stessa sandbox dell'oggetto che chiama il metodo oppure provenire da un'origine che ha acquisito l'autorizzazione di accesso alla sandbox del chiamante; in caso contrario, viene generata un'eccezione SecurityError. Per file audio caricati da audio incorporati in una libreria di un file SWF, l'autorizzazione viene concessa mediante chiamata al metodo `Security.allowDomain()` nel file SWF caricato. Per file audio caricati da fonti diverse dai file SWF (derivanti da file mp3 caricati o da video Flash), è necessario un file di criteri dei domini che conceda l'autorizzazione di accesso ai dati del file multimediale caricato. Non è possibile utilizzare il metodo `computeSpectrum()` se un file audio viene caricato da streaming RTMP.

Per ulteriori informazioni, vedere "Controlli creatore (sviluppatore)" a pagina 820 e "Controlli del sito Web (file di criteri dei domini)" a pagina 816.

Accesso a dati video

È possibile utilizzare il metodo `BitmapData.draw()` per catturare i dati pixel del fotogramma corrente di un video.

Esistono due diversi tipi di video:

- Video RTMP
- Video progressivi, caricati da un file FLV senza un server RTMP

Non è possibile utilizzare il metodo `BitmapData.draw()` per accedere a video RTMP.

Quando si chiama il metodo `BitmapData.draw()` con video progressivi come il parametro `source`, il chiamante di `BitmapData.draw()` deve appartenere alla stessa sandbox del file FLV oppure il server del file FLV deve contenere un file di criteri che conceda l'autorizzazione di accesso al dominio del file SWF chiamante. È possibile richiedere che il file dei criteri venga scaricato impostando la proprietà `checkPolicyFile` dell'oggetto NetStream su `true`.

Caricamento di dati

I file SWF possono caricare dati da server in ActionScript e inviare dati da ActionScript ai server. Il caricamento dei dati è un'operazione differente dal caricamento di file multimediali, in quanto le informazioni caricate vengono direttamente visualizzate in ActionScript, anziché essere riprodotte come file multimediali. Generalmente, i file SWF possono caricare dati dai propri domini. Tuttavia, essi richiedono file di criteri dei domini per caricare dati da altri domini.

Uso di URLLoader e URLStream

È possibile caricare dati, quali file XML o file di testo. I metodi `load()` delle classi URLLoader e URLStream sono governati da autorizzazioni di file di criteri dei domini.

Se si usa il metodo `load()` per caricare contenuto da un dominio diverso da quello del file SWF che chiama il metodo, Flash Player verifica la presenza di un file di criteri dei domini sul server delle risorse caricate. Se il file di criteri dei domini è presente e garantisce l'accesso al dominio del file SWF caricante, è possibile procedere al caricamento dei dati.

Connessione a socket

Per impostazione predefinita, l'accesso a socket e socket XML appartenenti a domini differenti non è consentito. È inoltre disabilitato, per impostazione predefinita, l'accesso a connessioni socket dello stesso dominio del file SWF su porte inferiori a 1024. Per consentire l'accesso a tali porte è necessario gestire un file di criteri dei domini da una delle seguenti posizioni:

- La stessa porta della connessione socket principale
- Una porta differente
- Un server HTTP sulla porta 80 nello stesso dominio del server socket

Se si gestisce il file di criteri dei domini dalla stessa porta della connessione socket principale, o da una porta differente, le porte autorizzate vengono enumerate mediante l'attributo `to-ports` nel file di criteri dei domini, come illustrato nell'esempio seguente:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy
  SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<!-- Policy file for xmlsocket://socks.mysite.com -->
<cross-domain-policy>
   <allow-access-from domain="*" to-ports="507" />
   <allow-access-from domain="*.example.com" to-ports="507,516" />
   <allow-access-from domain="*.example.org" to-ports="516-523" />
   <allow-access-from domain="adobe.com" to-ports="507,516-523" />
   <allow-access-from domain="192.0.34.166" to-ports="*" />
</cross-domain-policy>
```

Per recuperare un file di criteri socket dalla stessa porta di una connessione socket principale, è sufficiente chiamare il metodo `Socket.connect()` o `XMLSocket.connect()` e, se il dominio specificato è diverso da quello del file SWF chiamante, Flash Player tenterà automaticamente di recuperare un file di criteri dalla stessa porta della connessione principale alla quale si sta tentando di accedere. Per recuperare un file di criteri socket da una porta differente sullo stesso server della connessione principale, chiamare il metodo `Security.loadPolicyFile()` con la sintassi speciale `"xmlsocket"`, come illustrato di seguito:

```
Security.loadPolicyFile("xmlsocket://server.com:2525");
```

Chiamare il metodo `Security.loadPolicyFile()` prima del metodo `Socket.connect()` o `XMLSocket.connect()`. Flash Player attenderà fino al completamento della richiesta di recupero del file di criteri prima di acconsentire o meno l'accesso alla connessione principale.

Se si implementa un server socket ed è necessario fornire un file di criteri socket, decidere se fornire il file di criteri utilizzando la stessa porta che accetta connessioni principali o un'altra porta. In entrambi i casi, il server dovrà attendere la prima trasmissione dal client prima di decidere se inviare un file di criteri o impostare una connessione principale. Quando Flash Player richiede un file di criteri, viene sempre trasmessa la seguente stringa non appena si stabilisce una connessione:

```
<policy-file-request/>
```

Quando il server riceve la stringa, il file di criteri può essere trasmesso. Non è possibile riutilizzare la stessa connessione per la richiesta di un file di criteri e per la connessione principale; la connessione deve essere chiusa dopo la trasmissione del file di criteri. In caso contrario, Flash Player chiude la connessione per la trasmissione del file di criteri prima di riconnettersi per l'impostazione della connessione principale.

Per ulteriori informazioni, vedere "File dei criteri socket" a pagina 818.

Invio di dati

L'invio di dati si verifica quando il codice ActionScript invia dati da un file SWF a un server o a una risorsa. L'invio di dati è sempre consentito per file SWF di domini della rete. Un file SWF locale può inviare dati a indirizzi di rete solo se si trova nella sandbox locale affidabile o locale con rete. Per ulteriori informazioni, vedere "Funzioni di sicurezza sandbox locali" a pagina 822.

È possibile utilizzare la funzione `flash.net.sendToURL()` per inviare dati a un URL. Vi sono anche altri metodi per inviare richieste a URL. Tra essi vi sono metodi di caricamento, quali `Loader.load()` e `Sound.load()` e metodi di caricamento dati, quali `URLLoader.load()` e `URLStream.load()`.

Caricamento e scaricamento di file

Il metodo `FileReference.upload()` avvia il caricamento di un file selezionato da un utente su un server remoto. È necessario chiamare il metodo `FileReference.browse()` o `FileReferenceList.browse()` prima di chiamare il metodo `FileReference.upload()`.

La chiamata al metodo `FileReference.download()` apre una finestra di dialogo nella quale è possibile scaricare un file da un server remoto.

NOTA	Sui server che richiedono l'autenticazione dell'utente, solo i file SWF in esecuzione in un browser (ovvero quelli che utilizzano il plug-in per il browser o il controllo ActiveX) possono fornire una finestra di dialogo per richiedere all'utente di immettere un nome utente e una password per l'autenticazione, e solo per gli scaricamenti. Flash Player non consente il caricamento su server che richiedono l'autenticazione utente.

Le operazioni di caricamento e scaricamento non sono consentite se il file SWF che effettua la chiamata si trova in nella sandbox locale con file system.

Per impostazione predefinita, un file SWF non può avviare un caricamento su o uno scaricamento da un server diverso dal proprio. Un file SWF può eseguire operazioni di caricamento su o scaricamento da server differenti, se tale server contiene un file di criteri dei domini in grado di concedere l'autorizzazione di accesso al dominio del file SWF richiedente.

Caricamento di contenuto incorporato da file SWF importati in un dominio di sicurezza

Quando si carica un file SWF, è possibile impostare il parametro `context` del metodo `load()` dell'oggetto Loader utilizzato per caricare il file. Tale parametro impiega un oggetto di LoaderContext. Se la proprietà `securityDomain` di tale oggetto LoaderContext viene impostata su `Security.currentDomain`, Flash Player verifica la presenza di un file di criteri validi per domini diversi sul server del file SWF caricato. Se il file di criteri dei domini è presente e garantisce l'accesso al dominio del file SWF caricante, è possibile procedere al caricamento del file come contenuto multimediale importato. In questo modo, il file che esegue caricamento potrà accedere agli oggetti contenuti nella libreria del file SWF.

Un altro modo per un file SWF di accedere alle classi dei file SWF caricati appartenenti una funzione di sicurezza sandbox differente è mediante chiamata al metodo `Security.allowDomain()` da parte del file SWF caricato, per ottenere l'accesso al dominio del file SWF chiamante. È possibile aggiungere la chiamata al metodo `Security.allowDomain()` al metodo della funzione di costruzione della classe principale del file SWF caricato, quindi fare in modo che il file SWF che esegue il caricamento aggiunga un listener eventi che risponda all'evento `init` inviato dalla proprietà `contentLoaderInfo` dell'oggetto Loader. Quando questo evento viene inviato, il file SWF caricato ha già chiamato il metodo `Security.allowDomain()` nel metodo della funzione di costruzione e le classi del file SWF caricato risultano accessibili al file SWF che esegue il caricamento. Il file SWF caricante può recuperare le classi dal file SWF caricato mediante il metodo `Loader.contentLoaderInfo.applicationDomain.getDefinition()`.

Operazioni con contenuto precedente

In Flash Player 6, il dominio utilizzato per alcune impostazioni del lettore si basa sulla porzione finale del dominio del file SWF. Tali impostazioni includono autorizzazioni di accesso per fotocamera e microfono, quote di archiviazione e archiviazione di oggetti condivisi persistenti.

Se il dominio di un file SWF include più di due segmenti, come www.example.com, il primo segmento (www) viene rimosso e viene utilizzata la porzione restante del dominio. Quindi, in Flash Player 6, www.example.com e store.example.com utilizzano entrambi example.com come dominio per queste impostazioni. Allo stesso modo, www.example.co.uk e store.example.co.uk utilizzano entrambi example.co.uk come dominio per le impostazioni. Ciò può provocare problemi, ad esempio nel caso file SWF da domini non correlati, quali example1.co.uk e example2.co.uk, accedano agli stessi oggetti condivisi.

In Flash Player 7 e successivi, le impostazioni del lettore vengono scelte per impostazione predefinita in accordo al dominio esatto del file SWF. Ad esempio, un file SWF appartenente a www.example.com utilizzerebbe le impostazioni del lettore per www.example.com, mentre un file SWF di store.example.com utilizzerebbe le impostazioni del lettore separate per store.example.com.

In un file SWF scritto con ActionScript 3.0, se `Security.exactSettings` è impostato su `true` (impostazione predefinita), Flash Player impiega i domini esatti per le impostazioni del lettore. Se è impostato su `false`, Flash Player impiega le impostazioni di dominio utilizzate in Flash Player 6. Se si modifica il valore predefinito di `exactSettings`, è necessario farlo prima che si verifichino eventi che richiedono la scelta di impostazioni del lettore da parte di Flash Player (ad esempio, prima di usare una fotocamera o un microfono o di recuperare un oggetto condiviso persistente).

Se si è pubblicato un file SWF con la versione 6 e a partire da esso si sono creati oggetti condivisi persistenti, per recuperare tali oggetti da un file SWF che usa ActionScript 3.0, è necessario impostare `Security.exactSettings` su `false` prima di chiamare `SharedObject.getLocal()`.

Impostazione di autorizzazioni LocalConnection

La classe LocalConnection consente di sviluppare file SWF che inviano istruzioni gli uni agli altri. Gli oggetti LocalConnection possono comunicare solo tra i file SWF in esecuzione sullo stesso client, ma possono essere eseguiti in applicazioni diverse (ad esempio, un file SWF in esecuzione in un browser e un file SWF in esecuzione in un proiettore).

Per ogni comunicazione LocalConnection, esiste un file SWF mittente e un file SWF listener. Per impostazione predefinita, Flash Player consente comunicazioni LocalConnection tra file SWF dello stesso dominio. Per file SWF che si trovano in sandbox differenti, il file listener deve concedere l'autorizzazione al file mittente mediante il metodo `LocalConnection.allowDomain()`. La stringa trasmessa come argomento al metodo `LocalConnection.allowDomain()` può contenere: nomi di dominio esatti, indirizzi IP e il carattere jolly `*`.

NOTA

Il metodo `allowDomain()` è stato modificato rispetto alla forma che aveva in ActionScript 1.0 e 2.0. Nelle versioni precedenti, `allowDomain()` era un metodo di callback implementato dall'utente. In ActionScript 3.0, `allowDomain()` è un metodo incorporato della classe LocalConnection che viene chiamato. Grazie a questa modifica, `allowDomain()` funziona in modo molto simile a `Security.allowDomain()`.

Un file SWF può utilizzare la proprietà `domain` della classe LocalConnection per determinare il proprio dominio.

Controllo dell'accesso a script in una pagina Web host

La creazione di script in uscita viene eseguita mediante l'uso delle seguenti API di ActionScript 3.0:

- Funzione `flash.system.fscommand()`
- La funzione `flash.net.navigateToURL()` (se si specifica un'istruzione di script, quale `navigateToURL("javascript: alert('Hello from Flash Player.')")`)
- La funzione `flash.net.navigateToURL()` (quando il parametro `window` è impostato su `"_top"`, `"_self"` o `"_parent"`)
- Il metodo `ExternalInterface.call()`

Per i file SWF eseguiti localmente, le chiamate a questi metodi hanno esito positivo solo se il file SWF e la pagina Web che lo contiene (se ne esiste una) si trovano nella funzione di sicurezza sandbox locale affidabile. Le chiamate a questi metodi non hanno esito positivo se il contenuto si trova nella sandbox locale con rete o locale con file system.

Il parametro `AllowScriptAccess` del codice HTML che carica un file SWF controlla la capacità di eseguire script in uscita da un file SWF.

Impostare tale parametro nel codice HTML della pagina Web che ospita un file SWF. Il parametro deve essere impostato nel tag `PARAM` o `EMBED`.

Il parametro `AllowScriptAccess` può avere uno dei tre seguenti possibili valori: `"always"`, `"sameDomain"` o `"never"`:

- Se `AllowScriptAccess` corrisponde a `"sameDomain"`, la creazione di script in uscita è consentita solo se il file SWF e la pagina Web si trovano nello stesso dominio. Si tratta dell'impostazione predefinita per il contenuto AVM2.
- Se `AllowScriptAccess` è impostato su `"never"`, la creazione di script in uscita non viene mai eseguita.
- Se `AllowScriptAccess` è impostato su `"always"`, la creazione di script in uscita viene sempre eseguita.

Se il parametro `AllowScriptAccess` non viene specificato per un file SWF in una pagina HTML, esso viene automaticamente impostato su `"sameDomain"` per il contenuto AVM2.

Segue un esempio di impostazione del tag `AllowScriptAccess` in una pagina HTML:

```
<object id='MyMovie.swf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-
  444553540000' codebase='http://download.adobe.com/pub/shockwave/cabs/
  flash/swflash.cab#version=9,0,0,0' height='100%' width='100%'>
<param name='AllowScriptAccess' value='never'/>
<param name='src' value=''MyMovie.swf'/>
<embed name='MyMovie.swf' pluginspage='http://www.adobe.com/go/
  getflashplayer' src='MyMovie.swf' height='100%' width='100%'
  AllowScriptAccess='never'/>
</object>
```

Il parametro `AllowScriptAccess` può impedire a un file SWF ospitato da un dominio di accedere a uno script di una pagina HTML proveniente da un altro dominio. Se si usa `AllowScriptAccess="never"` per tutti i file SWF ospitati da domini differenti, si può garantire la sicurezza degli script contenuti in una pagina HTML.

Per ulteriori informazioni, vedere le seguenti voci in *Guida di riferimento del linguaggio e ai componenti ActionScript 3.0.*

- Funzione `flash.system.fscommand()`
- Funzione `flash.net.navigateToURL()`
- Metodo `call()` della classe ExternalInterface

Oggetti condivisi

Flash Player consente di usare *oggetti condivisi*, vale a dire oggetti di ActionScript che persistono al di fuori dei file SWF, sia a livello locale su un file system di un utente o in remoto su un server RTMP. Gli oggetti condivisi, come altri file multimediali di Flash Player, sono suddivisi in funzioni di sicurezza sandbox. Tuttavia, il modello di sandbox degli oggetti condivisi è un po' differente, in quanto gli oggetti condivisi non sono risorse accessibili da un dominio a un altro. Al contrario, gli oggetti condivisi vengono sempre recuperati da un apposito archivio specifico del dominio di ciascun file SWF che chiama i metodi della classe SharedObject. Generalmente, l'archivio degli oggetti condivisi è ancora più particolare di un dominio di file SWF: per impostazione predefinita, ogni file SWF impiega un archivio di oggetti condivisi particolare in base al proprio URL di origine.

Un file SWF può utilizzare il parametro `localPath` dei metodi `SharedObject.getLocal()` e `SharedObject.getRemote()` per usare un archivio di oggetti condivisi associato con solo una parte del proprio URL. In questo modo, il file SWF può consentire la condivisione con altri file SWF di altri URL. Anche se si trasmette `'/'` come parametro `localPath`, ciò specifica ancora un archivio di oggetti condivisi particolare del dominio.

Gli utenti possono limitare l'accesso agli oggetti condivisi tramite la finestra di dialogo Impostazioni o Gestione impostazioni di Flash Player. Per impostazione predefinita, è possibile creare oggetti condivisi che contengono un massimo di 100 KB di dati per dominio. Gli utenti con privilegi amministrativi e gli utenti standard possono anche limitare la possibilità di scrivere sul file system. Per ulteriori informazioni, vedere "Controlli amministratore" a pagina 812 e "Controlli utente" a pagina 814.

Per impostare la protezione di un oggetto condiviso, specificare `true` per il parametro `secure` del metodo `SharedObject.getLocal()` o del metodo `SharedObject.getRemote()`. Per quanto riguarda il parametro `secure`, si tenga presente quanto segue:

- Se il parametro viene impostato su `true`, Flash Player crea un nuovo oggetto condiviso protetto oppure acquisisce un riferimento all'oggetto condiviso protetto esistente. Questo oggetto condiviso protetto può essere letto o scritto solo da/su file SWF distribuiti tramite HTTPS che chiamano `SharedObject.getLocal()` con il parametro`secure` impostato su `true`.
- Se il parametro viene impostato su `false`, Flash Player crea un nuovo oggetto condiviso oppure ottiene un riferimento a un oggetto condiviso esistente che può essere letto o scritto dai file SWF distribuiti mediante connessioni non HTTPS.

Se il file SWF chiamante non proviene da un URL HTTPS e si specifica `true` per il parametro `secure` del metodo `SharedObject.getLocal()` o del metodo `SharedObject.getRemote()`, viene generata un'eccezione SecurityError.

La scelta di un archivio di oggetti condivisi si basa sull'URL di origine del file SWF. Ciò si verifica anche nelle due situazioni in cui un file SWF non ha origine da un semplice URL, vale a dire nei casi di caricamento mediante importazione e di caricamento dinamico. Il caricamento mediante importazione avviene quando un file SWF viene caricato con la proprietà `LoaderContext.securityDomain` impostata su `SecurityDomain.currentDomain`. In questa situazione, il file SWF caricato presenta uno pseudo URL che inizia con il dominio del file SWF caricante, quindi specifica l'effettivo URL di origine. Il caricamento dinamico si riferisce al caricamento di un file SWF mediante il metodo `Loader.loadBytes()`. In questa situazione, il file SWF caricato avrà uno pseudo URL che inizia con l'intero URL del file SWF che esegue il caricamento, seguito da un ID di numero intero. In entrambi i casi, è possibile esaminare lo pseudo URL del file SWF utilizzando la proprietà `LoaderInfo.url`. Lo pseudo URL viene trattato esattamente come un vero URL allo scopo di scegliere un archivio di oggetti condivisi. È possibile specificare un parametro `localPath` di oggetto condiviso che impiega in parte o per intero lo pseudo URL.

Utenti e amministratori possono scegliere di disabilitare l'uso di *oggetti condivisi di terze parti*. Si tratta dell'impiego di oggetti condivisi da parte di qualsiasi file SWF che viene eseguito in un browser Web, quando l'URL di origine di tale file SWF appartiene a un dominio diverso dall'URL visualizzato nella barra degli indirizzi del browser. Utenti e amministratori possono scegliere di disabilitare l'uso di oggetti condivisi di terze parti per motivi di riservatezza, per evitare la cattura di dati di traccia tra domini. Per evitare questa limitazione, è possibile fare in modo che qualsiasi file SWF che usa oggetti condivisi venga caricato unicamente all'interno delle strutture di pagine HTML, al fine di garantire che il file SWF provenga dallo stesso dominio visualizzato nella barra degli indirizzi del browser. Quando si cerca di usare oggetti condivisi di un file SWF di terze parti, e l'uso di oggetti condivisi di terze parti è disabilitato, i metodi `SharedObject.getLocal()` e `SharedObject.getRemote()` restituiscono il valore `null`. Per ulteriori informazioni, vedere www.adobe.com/products/flashplayer/articles/thirdpartylso.

Accesso a fotocamera, microfono, Appunti, mouse e tastiera

Quando un file SWF tenta di accedere alla fotocamera o al microfono di un utente utilizzando il metodo `Camera.get()` o `Microphone.get()`, Flash Player visualizza una finestra di dialogo relativa alla riservatezza, nella quale l'utente può concedere o meno l'accesso alla sua fotocamera e microfono. Utenti e utenti amministrativi sono inoltre in grado di disattivare l'accesso alla fotocamera globalmente o per sito, mediante i controlli del file mms.cfg, l'interfaccia utente Impostazioni e Gestione impostazioni (vedere "Controlli amministratore" a pagina 812 e "Controlli utente" a pagina 814). Se vengono impostati tali limitazioni per gli utenti, i metodi `Camera.get()` e `Microphone.get()` restituiscono il valore `null`. È possibile utilizzare la proprietà `Capabilities.avHardwareDisable` per determinare se l'accesso a fotocamera e microfono è stato negato (`true`) o consentito (`false`) dall'amministratore.

Il metodo `System.setClipboard()` consente a un file SWF di sostituire il contenuto degli Appunti con una stringa di testo normale. Ciò non presenta rischi di sicurezza. Per proteggersi dal rischio di tagliare o copiare negli Appunti password e altri dati sensibili, non esiste un metodo "getClipboard" (lettura) corrispondente.

Un'applicazione Flash può monitorare solo gli eventi di tastiera e del mouse che si verificano nell'ambito del proprio stato attivo ma non è in grado di rilevare eventi tastiera e mouse in altre applicazioni.

Indice analitico

Simboli

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X