學習

ACTIONSCRIPT® 3.0

上次更新 **2011/5/17**

法律聲明

目錄

第 1 章 ActionScript 3.0 簡介

關於 ActionScript

ActionScript 是 Adobe® Flash® Player 和 Adobe® AIR™ 執行階段環境所使用的程式語言。這套語言可為 Flash、Flex 與 AIR 內容和應用程式提供互動性、資料處理和更多功能。

ActionScript 會在 ActionScript Virtual Machine (AVM，是 Flash Player 和 AIR 的一部分) 中執行。編譯器通常會將 ActionScript 程式碼轉換成位元組碼格式 (位元組碼是一種由電腦撰寫而且電腦可以理解的程式設計語言)。編譯器範例包括一種 Adobe® Flash® Professional 內建的編譯器以及一種 Adobe® Flash® Builder™ 內建並在 Adobe® Flex™ SDK 中可以用到的編譯器。位元組碼內嵌於 Flash Player 和 AIR 都會執行的 SWF 檔案中。

ActionScript 3.0 提供強大的程式設計模型，對於物件導向程式設計有基本認識的開發人員將能很快熟悉。ActionScript 3.0 中針對舊版 ActionScript 改善的部分重要功能包括：

- 名為 AVM2 的新 ActionScript Virtual Machine，它使用了全新的位元組碼指令集，可大幅提高效能
- 更新穎的編譯器程式碼基底，相較於舊版編譯器，可以執行更深入的最佳化作業
- 經過擴充和改良的應用程式設計介面 (API)，對物件具備較低的控制度，是真正的物件導向模型
- 以 ECMAScript for XML (E4X) 規格 (ECMA-357 第 2 版) 為基礎的 XML API。E4X 是 ECMAScript 的語言擴充功能，將 XML 新增為該語言的原生資料類型。
- 以「文件物件模型 (DOM) 第 3 層事件規格」為基礎的事件模型

ActionScript 3.0 的優點

ActionScript 3.0 的 Script 編寫功能比舊版 ActionScript 更強大。這套語言能建立高度複雜的應用程式，其中包含大型資料集和物件導向、重複使用的程式碼基底。在 Adobe Flash Player 中執行的內容並不需要 ActionScript 3.0。不過，它只有透過 AVM2 (ActionScript 3.0 虛擬機器)，才能改善效能。與舊版 ActionScript 程式碼相較，ActionScript 3.0 程式碼的執行速度能快上十倍。

舊版 ActionScript Virtual Machine (AVM1) 可執行 ActionScript 1.0 和 ActionScript 2.0 程式碼。Flash Player 9 和 10 支援 AVM1 的回溯相容。

ActionScript 3.0 的新增功能

ActionScript 3.0 包含許多與 ActionScript 1.0 和 2.0 相似的類別和功能。不過，ActionScript 3.0 在架構和概念上與舊版 ActionScript 不同。ActionScript 3.0 中的增強項目包括核心語言的新功能，以及經過改良的 API，可加強控制低階物件。

核心語言功能

核心語言會定義程式語言的基本建構區塊，例如陳述式、運算式、條件、迴圈和類型。ActionScript 3.0 包含許多可加速開發程序的功能。

執行階段例外

與舊版 ActionScript 相較，ActionScript 3.0 會通報更多的錯誤狀況。執行階段例外適用於常見的錯誤狀況，可加強除錯效果，讓您開發有效處理錯誤的應用程式。執行階段錯誤可提供堆疊追蹤，其中會加註來源檔案和行號資訊，協助您快速找出錯誤所在位置。

執行階段類型

在 ActionScript 3.0 中，類型資訊會在執行階段中予以保留。此資訊是用於執行執行階段類型檢查，藉以改善系統的檔案安全。類型資訊也可用來以原生機器的表示方式代表變數，這樣可以提高效能並減少記憶體用量。透過比較的方式，在 ActionScript 2.0 中，類型註釋主要是一種開發人員輔助，而且所有的值都會在執行階段中動態輸入。

密封類別

ActionScript 3.0 包括密封類別的概念。密封類別只擁有一組在編譯階段定義的固定屬性和方法，不能加入其它屬性和方法。在執行階段不可以變更類別，這樣能夠進行更嚴格的編譯階段檢查，產生更強大的程式。此外，由於每個物件實體不需要內部雜湊表，因此這也會減少記憶體的用量。只要使用 dynamic 關鍵字，就可以宣告動態類別。ActionScript 3.0 中的所有類別預設都是密封的，但是您可以使用 dynamic 關鍵字將類別宣告為動態類別。

方法結束項

ActionScript 3.0 可讓方法終止自動記憶其原始物件實體。這項功能對事件處理很有用。在 ActionScript 2.0 中，方法終止不會記憶當初是從哪一個來源物件實體擷取，因此當叫用方法終止時，會導致無法預期的行為。

ECMAScript for XML (E4X)

ActionScript 3.0 實作了最近才標準化為 ECMA-357 的 ECMAScript for XML (E4X)。E4X 可提供一組自然、順暢的語言建構，可用來操作 XML。與傳統 XML 剖析 API 不同的是，使用 E4X 的 XML 就像是語言的原生資料類型。E4X 可大幅減少所需的程式碼，簡化使用 XML 開發應用程式的工作。

若要檢視 ECMA E4X 規格，請前往 www.ecma-international.org。

規則運算式

ActionScript 3.0 包含規則運算式的原生支援，讓您可以快速搜尋和操作字串。ActionScript 3.0 會依照 ECMAScript (ECMA-262) 第 3 版語言規格中的定義，實作規則運算式的支援。

命名空間

命名空間類似於用來控制宣告可見性的傳統存取指定字 (public、private、protected)。但命名空間是自訂存取指定字，可以讓您自行選擇名稱。命名空間會使用通用資源識別項 (URI) 來避免發生衝突，而當您使用 E4X 時，命名空間會用來代表 XML 命名空間。

新的基本類型

ActionScript 3.0 包含 3 種數字類型：Number、int 以及 uint。數字代表一個雙倍精確、浮點數字。int 類型是 32 位元具有正負號的整數，可讓 ActionScript 程式碼利用 CPU 的快速整數算術能力。int 類型適用於使用整數的迴圈計數器和變數。uint 類型是無正負號的 32 位元整數，適用於 RGB 顏色值、位元組計數等用途。相反的，ActionScript 2.0 只有一種數字類型 - Number。

API 功能

ActionScript 3.0 中的 API 包含許多類別，可讓您控制低階物件。與舊版相比，這套語言的架構在設計上更為直覺。由於新類別太多，無法逐一詳細說明，不過某些巨大的差異值得一提。

DOM3 事件模型

Document Object Model Level 3 事件模型 (DOM3) 提供一種產生以及處理事件訊息的方法。這種事件模型的設計是為了允許物件應用程式中的物件，可以互動以及通訊、維護其狀態以及對任何變更做出回應。ActionScript 3.0 事件模型是以「全球資訊網協會 DOM 第 3 層事件規格」為基礎，提供的機制比舊版 ActionScript 的事件系統更為清楚，也更有效率。這種模型提供的機制遠比舊版 ActionScript 提供的事件系統還要明確且更有效率。

事件和錯誤事件都可在 flash.events 套件中找到。Flash Professional 組件使用的事件模型與 Flex 架構相同，因此事件系統在 Flash Platform 上是統一的。

顯示清單 **API**

這是存取顯示清單 (即包含應用程式中任何視覺元素的樹狀結構) 的 API，是由處理 Flash 視覺基本元素的類別所組成。

Sprite 類別是一種輕量型建構區塊，其設計是做為使用者介面組件之類視覺元素的基底類別。Shape 類別則代表原始向量形狀。這些類別都可以使用 new 運算子加以自然實體化，而且可以隨時動態改變父系。

自動深度管理。您可以使用方法來指定和管理物件的疊置順序。

處理動態資料和內容

ActionScript 3.0 包含多項可用來載入和處理應用程式中資源和資料的機制，這些機制不但直覺，而且在 API 中一致。Loader 類別提供載入 SWF 檔和影像資源的單一機制，並能夠存取所載入內容的詳細資訊。URLLoader 類別可提供另一種機制，將文字和二進位資料載入資料驅動應用程式。Socket 類別可用任何形式讀取和寫入二進位資料至伺服器通訊端。

低階資料存取

各種 API 提供低階資料存取。針對下載的資料，URLStream 類別可在資料下載的同時，以原始二進位資料的形式存取資料。ByteArray 類別可讓您最佳化讀取、寫入和使用二進位資料。聲音 API 透過 SoundChannel 和 SoundMixer 類別，可讓您精細控制聲音。安全性 API 可以提供 SWF 檔或所載入內容的安全性權限資訊，讓您處理安全性錯誤。

使用文字

ActionScript 3.0 為所有與文字相關的 API 提供 flash.text 套件。TextLineMetrics 類別可為文字欄位中的文字行提供詳細的公制字。它取代了 ActionScript 2.0 中的 TextFormat.getTextExtent() 方法。TextField 類別包含低階方法，可以為文字欄位中的文字行或單一字元提供特定資訊。例如，getCharBoundaries() 方法會傳回代表字元範圍框的矩形。getCharIndexAtPoint() 方法會傳回特定地方的字元索引。getFirstCharInParagraph() 方法會傳回段落中第一個字元的索引。行階層方法則包含 getLineLength() (傳回指定文字行中的字元數) 以及 getLineText() (傳回指定行的文字)。Font 類別可管理 SWF 檔中的內嵌字體。

為了對文字進行平均低階控制，flash.text.engine 套件中的類別會組成 Flash 文字引擎。這一組類別提供文字的低階控制，而且其設計是用於建立文字架構和組件。

第 2 章 ActionScript 快速入門

程式設計的基本概念

由於 ActionScript 是一種程式語言，因此如能先瞭解一些基本電腦程式設計概念，將有助於快速熟悉 ActionScript。

電腦程式的作用

首先，我們可以先瞭解什麼是電腦程式以及其作用。電腦程式可分為兩點進行說明：

- 程式是讓電腦執行的一系列指示或步驟。
- 每個步驟最終都會操作某些資訊或資料。

一般來說，電腦程式就是提供給電腦的逐步指示清單，讓電腦逐一執行指示。每個個別的指示稱為 「陳述式」。在 ActionScript 中，每一個陳述式在撰寫最後都會加上一個分號。

基本上，程式中的指示就是用來操作儲存在電腦記憶體中的一些資料。簡單的範例像是指示電腦將兩個數字相加，並將結果儲存在記憶體。而複雜的範例則像是螢幕上畫了一個矩形，而您想寫一個程式，將這個矩形移到螢幕上的其它位置。電腦會記住矩形的特定資訊：矩形所在的 x 和 y 座標、寬度和高度、顏色等。這些資訊都會儲存在電腦記憶體中。程式將矩形移至不同的位置時，應採取的步驟會類似「將 x 座標變更成 200、將 y 座標變更成 150」。換言之，它應該為 x 和 y 座標指定新的值。電腦在幕後會對此資料做一些處理，以便將這些數字實際轉換成顯示在電腦螢幕上的影像。不過，就基本的細節而言，我們只要了解 「移動螢幕上的矩形」，只是涉及變更電腦記憶體中的資料位元就可以了。

變數與常數

程式設計主要涉及變更電腦記憶體中的資訊部分。因此，能夠在程式中呈現一小部分資訊的方式十分重要。「變數」是一種名稱，代表電腦記憶體中的值。當您撰寫陳述式來操作各個值時，會以變數的名稱代替每一個值。當電腦在程式中看到變數名稱時，就會搜尋記憶體，然後使用所找到的值。例如，假設您有兩個變數名為 value1 和 value2，各自代表一個數字，若要將這兩個數字相加，可以將陳述式撰寫如下：

```
value1 + value2
```

實際執行步驟時，電腦會先尋找每個變數中的值，然後再將值相加起來。

在 ActionScript 3.0 中，變數實際上是由三個不同部分組成的：

- 變數的名稱
- 可以儲存在變數中的資料類型
- 儲存在電腦記憶體中的實際值

您已經明白了電腦如何使用名稱做為值的預留位置。不過，資料類型也十分重要。當您在 ActionScript 建立變數後，要指定保存變數的資料類型。從現在開始，程式的指示只能在變數中儲存該種資料類型。您可以使用與資料類型相關的特性，操控這個值。在 ActionScript 中，若要建立變數 (稱為 「宣告」變數)，便需要使用 var 陳述式：

```
var value1:Number;
```

此範例會告訴電腦建立一個名稱為 value1 的變數，它只會保存 Number 資料。("Number" 是 ActionScript 中定義的一種特定資料類型) 。您也可以立即在變數中儲存值：

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

Adobe Flash Professional

在 Flash Professional 中，會使用另一種方式宣告變數。當您在「舞台」上放置影片片段元件、按鈕元件或文字欄位時，會在「屬性」檢測器中指定實體名稱。Flash Professional 在幕後會建立一個變數，其名稱與實體名稱相同。您可以在 ActionScript 程式碼中使用這個名稱，用來代表「舞台」項目。舉個例子，假設您在「舞台」上有一個影片片段元件，您將它的實體名稱命名為 rocketShip。任何時候在 ActionScript 程式碼中使用變數 rocketShip，實際上您就是在操控這個影片片段。

常數和變數很類似。它是一個名稱，以指定的資料類型來代表電腦記憶體中的值。不同的是，常數在 ActionScript 應用程式的過程中，一次只能被指定一個值。一旦指定常數的值之後，該值在整個應用程式中都相同。常數的宣告語法和變數的宣告語法幾乎一樣。唯一的差異就是您使用 const 關鍵字取代 var 關鍵字：

```
const SALES_TAX_RATE:Number = 0.07;
```

定義的值會在整個專案多個位置使用時，常數相當實用，因為在正常情況下，常數不會改變。使用常數而非常值的話，可讓您的程式碼更容易讀取。例如，假設相同程式碼的兩個版本。一個版本是將價格乘以 SALES_TAX_RATE。另一個版本是將價格乘以 0.07。使用 SALES_TAX_RATE 常數的那一個版本更容易明白。此外，常數定義的值預設不會變更。如果您使用常數來代表專案中的值，只需在一個地方變更該值即可。相反的，如果使用硬式編碼的數值，就必須到每個地方進行變更。

資料類型

在 ActionScript 中，您可以使用許多資料類型做為所建立變數的資料類型。部分資料類型可視為「簡單」或「基本」資料類型：

- String：文字值，例如某個名稱或書籍章節文字
- Numeric：ActionScript 3.0 針對數值資料提供三種特定資料類型 -
  - Number：任何數值，包括有小數或沒有小數的值
  - int：整數 (即沒有小數的數字)
  - uint：「無正負號」的整數，表示這個整數不能是負數
- Boolean：true 或 false 值，例如切換是否開啟，或兩個值是否相等

簡單資料類型代表單一資訊，例如一個數字或一段文字。不過，ActionScript 大部分定義的資料類型都是複雜資料類型。它們代表單一容器中的一組值。例如，資料類型為 Date 的變數代表單一值 (即某個時間)。但是，日期值是由幾個值所組成，包括日、月、年、時、分、秒等，這些都是個別的數字。人們一般會將日期視為單一值，而且您可以建立 Date 變數，就可以將日期當做是單一值。不過，電腦內部會將它視為一群值，整體定義為單一的日期。

大多數內建資料類型和程式設計人員定義的資料類型，都屬於複雜資料類型。一些您或許認得的複雜資料類型包括：

- MovieClip：影片片段元件
- TextField：動態或輸入文字欄位
- SimpleButton：按鈕元件
- Date：單一時間的資訊 (日期和時間)

資料類型的兩個常見同義詞是類別和物件。類別就是資料類型的定義。它像是一個範本，用於這種資料類型的所有物件，好比說「Example 資料類型的所有變數都具備以下特性：A、B 和 C」；而「物件」則是類別的實際實體。例如，資料類型為 MovieClip 的變數可描述為 MovieClip 物件。因此，下面的說法基本上都是相同的：

- 變數 myVariable 的資料類型為 Number。
- 變數 myVariable 是 Number 實體。
- 變數 myVariable 是 Number 物件。
- 變數 myVariable 是 Number 類別的實體。

使用物件

ActionScript 是一種物件導向的程式語言。物件導向程式設計只是一種程式設計的方式，並使用物件來組織程式中的程式碼。

以前 「電腦程式」這個名詞是定義為電腦執行的一系列步驟或指示。就概念上來說，可以將電腦程式視為一長串指示的清單。不過，在物件導向程式設計中，程式指示分散到不同的物件。程式碼會組成不同的功能區塊，因此相關的功能類型或相關的資訊片段會群組在一個容器中。

Adobe Flash Professional

如果使用過 Flash Professional 的元件，您就已經使用過物件了。假設您定義一個矩形的影片片段元件，然後將其副本放在「舞台」上。這個影片片段元件等於 (基本上) 就是 ActionScript 中的物件，即 MovieClip 類別的實體。

您可以修改影片片段的多個特性。選取它之後，您可以在 「屬性」檢測器中變更值，例如它的 x 座標或寬度。您也可以做各種顏色調整，例如變更它的 Alpha (透明度) 或者套用投影濾鏡。您還可以使用其它 Flash Professional 工具進行變更，例如使用 「自由變形」工具旋轉矩形。您可以在 Flash Professional 修改影片片段元件的所有方法，ActionScript 也都可以使用。您只需變更組合在 MovieClip 物件中的所有資料，就可以在 ActionScript 修改電影片段 。

在 ActionScript 物件導向程式設計中，任何類別都可能具有三種特性：

- 屬性
- 方法
- 事件

這些特性可用來管理程式所使用的資料，以及決定要執行的動作和及其順序。

屬性

屬性是物件中所合併多項資料的其中一項資料。範例歌曲物件可能具有名為 artist 和 title 的屬性，而 MovieClip 類別則具有像是 rotation、x、width 和 alpha 的屬性。屬性的使用方式與個別變數相同。事實上，您可以將屬性當做是物件所包含的 「子」變數。

下面是 ActionScript 程式碼使用屬性的幾個範例。這一行程式碼會將名為 square 的 MovieClip 移至 x 座標 100 像素的位置：

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

此程式碼會使用 rotation 屬性來旋轉 square MovieClip，以便與 triangle MovieClip 的旋轉角度相符：

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

此程式碼會變更 square MovieClip 的水平縮放，讓寬度成為原來的 1.5 倍：

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

請注意這幾段程式碼的共通結構：首先會使用變數 (square、triangle) 做為物件的名稱，後面接著一個英文句點 (.)，然後再加上屬性的名稱 (x、rotation、scaleX)。這個英文句點稱為 「點運算子」，用來表示存取物件的其中一個子元素。這整個結構 (「變數名稱 - 點 - 屬性名稱」) 可當做單一變數，代表電腦記憶體中單一值的名稱。

方法

「方法」是物件可以執行的動作。例如，假設您在 Flash Professional 建立影片片段元件，它的時間軸有數個關鍵影格和動畫。影片片段就可以播放、停止或指示將播放磁頭移至特定影格。

下面這行程式碼指示名為 shortFilm 的 MovieClip 開始播放：

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

這行程式碼會指示名為 shortFilm 的 MovieClip 停止播放 (播放磁頭會立即停止，就像暫停視訊一樣)：

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

這行程式碼會指示名為 shortFilm 的 MovieClip 將播放磁頭移至 「影格 1」，然後停止播放 (就像倒帶一樣)：

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

方法與屬性的存取方式類似，一樣是先寫物件的名稱 (變數)，後面接著一個英文句點，然後加上方法的名稱，後面接著括號。括號的作用是 「呼叫」方法，也就是指示物件執行動作。有時候括號內會放入值 (或變數)，這表示會一起傳遞執行動作所需的其它資訊。這些值稱為方法 「參數」。例如，gotoAndStop() 方法就必須知道要移至哪一個影格，因此需要在括號內加上單一參數。其它像是 play() 和 stop() 等方法的本身指示就很清楚，因此不需要額外的資訊。不過，這些方法後面還是要加上括號。

與屬性 (和變數) 不同的是，方法不能當做值預留位置。不過有些方法可以執行計算，然後傳回的結果可當做變數。例如，Number 類別的 toString() 方法會將數值轉換為文字表示：

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

假設您要將 Number 變數的值顯示在螢幕上的文字欄位中，就必須使用 toString() 方法。TextField 類別的 text 屬性定義為 String，表示只能包含文字值。(text 屬性代表實際顯示在螢幕的文字內容)。這行程式碼會將變數 numericData 中的數值轉換為文字。然後顯示在螢幕上名為 calculatorDisplay 的 TextField 物件中：

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

事件

電腦程式為電腦逐步執行的一系列指示。某些簡單的電腦程式確實如此，電腦只需執行幾個步驟就結束程式。但是，ActionScript 程式會持續執行，等待使用者輸入或其它動作。而 「事件」就是決定電腦應執行哪些指示以及何時執行的機制。

基本上，「事件」是 ActionScript 知道而且可以回應的情況。很多事件是與使用者互動有關，例如使用者按一下某個按鈕或者按下鍵盤上的某個按鍵。事件還有其他類型。舉例來說，假設您使用 ActionScript 載入外部影像，就會有事件讓您知道影像何時載入完畢。當 ActionScript 程式執行時，就概念上，它只是在等候特定的事情發生而已。當事情發生後，就會執行您為這些事件指定的特定 ActionScript 程式碼。

基本事件處理

指定執行特定動作以回應特定事件的技巧稱為 「事件處理」。當撰寫 ActionScript 程式碼來執行事件處理時，需要先找出三個重要元素：

- 事件來源：事件會發生在哪一個物件？例如，按下哪一個按鈕，或者是由哪一個 Loader 物件載入影像？事件來源又稱為 「事件目標」。因為電腦就是將這個物件當做是事件的目標 (也就是，事件實際發生的地方)，因此這樣稱呼它。
- 事件：會發生什麼情況，也就是要回應的情況為何？這是必須指出的特定項目，因為很多物件會觸發多個事件。
- 回應：事件發生時，要執行的步驟？

當您撰寫 ActionScript 程式碼來處理事件時，便需要這三個元素。程式碼會遵循這個基本結構 (粗體字的元素是預留位置，您要視特定情況填寫)：

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

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

此程式碼會執行兩個動作：首先會定義函數，以指定動作來回應事件。接下來，它會呼叫來源物件的 addEventListener() 方法。呼叫 addEventListener() 基本上會將這個函數 「訂閱」給指定的事件使用。當事件發生後，就會執行函數的動作。下面將針對各個部分詳細說明。

「函數」是由多個動作所組成，用單一名稱來代表執行這些動作的捷徑名稱。函數與方法大致相同，但函數不一定要與特定類別相關聯。(實際上，您可將 「方法」定義為與特定類別相關聯的函數。) 當您建立函數來處理事件時，可以選擇函數的名稱 (在此例中，函數名稱為 eventResponse)，您也指定一個參數 (在此例中，參數名稱為 eventObject)。指定函數參數就像宣告變數，因此您也必須指定參數的資料類型 (在此範例中，參數的資料類型為 EventType)。

您要偵聽的每種事件都有一個相關聯的 ActionScript 類別。您為函數參數所指定的資料類型也固定會是與要回應之特定事件相關的類別。例如，click 事件 (當使用者以滑鼠按一下某個項目時觸發) 與 MouseEvent 類別相關聯。若要針對 click 事件撰寫偵聽程式函數，您需要以資料類型為 MouseEvent 的參數定義偵聽程式函數。最後，請在左右大括號 ({ ... }) 之間，撰寫事件發生時，電腦要執行的指示。

撰寫事件處理函數。接下來就需要指示事件來源物件 (即發生事件的物件，例如按鈕)，在事件發生時必須呼叫函數。呼叫該物件的 addEventListener() 方法 (具有事件的所有物件都會有 addEventListener() 方法)，將您的函數註冊至事件來源物件。addEventListener() 方法使用兩個參數：

- 第一個是要回應的特定事件名稱。每個事件都會與特定類別相關聯。每個事件類別會為它的每個事件定義特殊的值，就像是獨一無二的名稱。第一個參數會使用該值。
- 第二個是事件回應函數的名稱。請注意，當函數當做參數傳遞時，其名稱在撰寫時是不加括號的。

事件處理程序

以下是您在建立事件偵聽程式時的逐步程序說明。這個範例是建立偵聽程式函數，當您按下 myButton 物件時，就會呼叫這個函數。

程式設計人員所撰寫的實際程式碼如下所示：

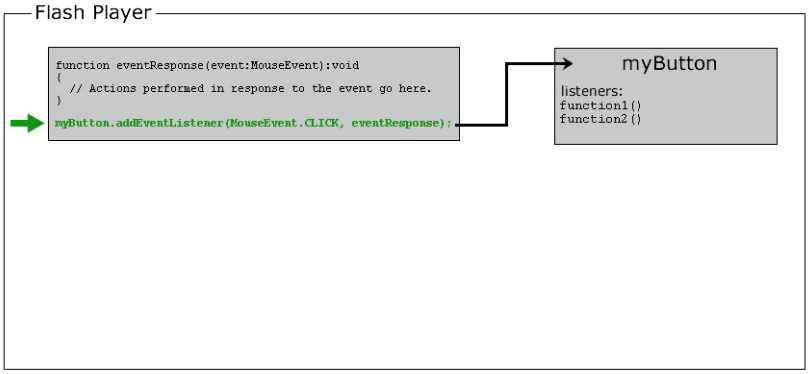
```
function eventResponse(event:MouseEvent):void
{
    // Actions performed in response to the event go here.
}

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

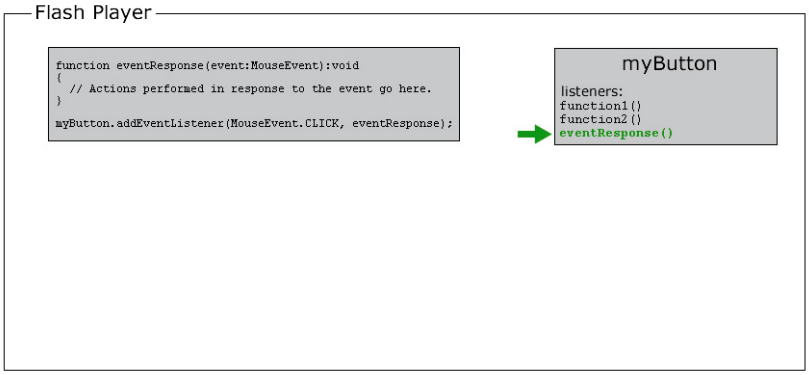
以下是此程式碼執行時的實際運作情形。

1 SWF 檔載入時，電腦就會記下有個名叫 eventResponse() 函數的事實。

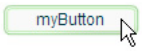
2 接著，電腦會執行程式碼 (明確地說，不屬於函數的程式碼)。在這裡只有一行程式碼，它會針對事件來源物件 (名為 myButton) 呼叫 addEventListener() 方法，並傳遞 eventResponse 函數當做參數。



在內部，myButton 會記錄一份可以偵聽各個事件的函數清單。呼叫它的 addEventListener() 方法時，myButton 會將 eventResponse() 函數儲存至它的事件偵聽程式清單。



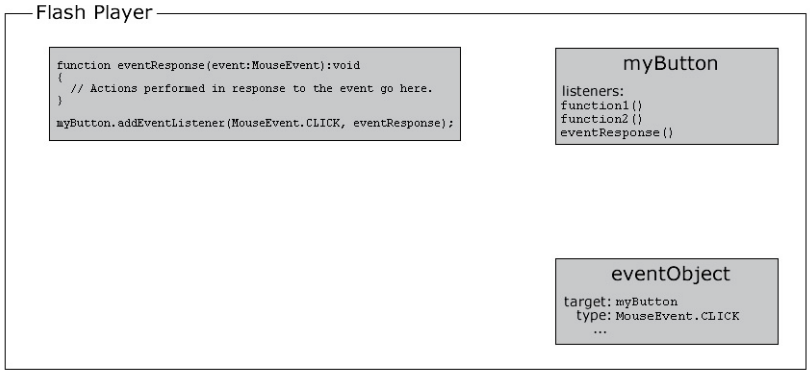
3 當使用者按下 myButton 物件時，就會觸發其 click 事件 (就是程式碼中的 MouseEvent.CLICK)。



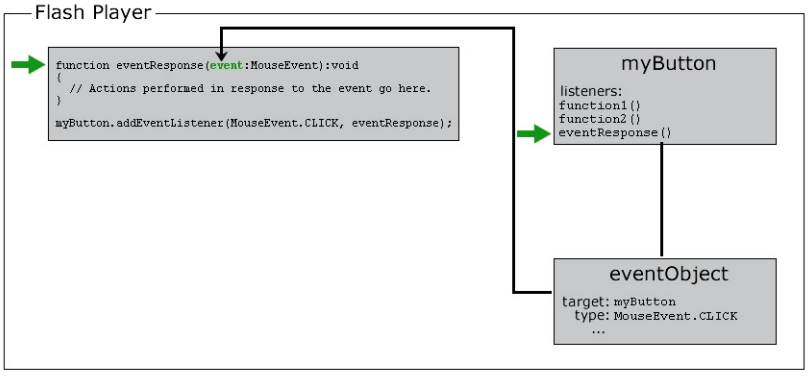
此時，會發生下列情況：

a 建立一個物件，也就是與上述事件相關聯的類別實體 (在這個範例中是指 MouseEvent)。對很多事件而言，這個物件是 Event 類別的實體。對滑鼠事件而言，它就是 MouseEvent 實體。對其它事件而言，則是與該事件相關的類別實體。建

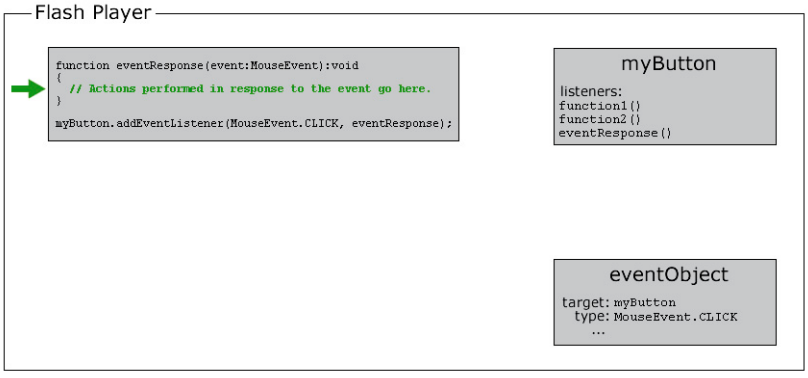
立的這個物件就是所謂的「事件物件」，會包含所發生事件的相關特定資訊：事件的類型、發生位置以及其它事件特有的資訊 (如果有的話)。



b 電腦接著會檢閱 myButton 儲存的事件偵聽程式清單。它會逐一檢視這些函數、呼叫每個函數，並將事件物件當做參數傳遞至函數。由於 eventResponse() 函數是 myButton 的偵聽程式之一，因此電腦在執行這個程序時，也會呼叫 eventResponse() 函數。



c 呼叫 eventResponse() 函數時，該函數的程式碼就會執行，進而執行您所指定的動作。



事件處理範例

以下是一些更具體的事件處理程式碼範例。這些範例可以協助您瞭解一些常見的事件元素，以及撰寫事件處理程式碼時可能的變化：

- 按一下按鈕，讓目前的影片片段開始播放。在下列範例中，playButton 是按鈕的實體名稱，而 this 是代表「目前物件」的特殊名稱：

```
this.stop();

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

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

- 偵測文字欄位的輸入內容。在這個範例中，entryText 是輸入文字欄位，而 outputText 是動態文字欄位：

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

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

- 按一下按鈕以瀏覽至 URL。在此例中，linkButton 是按鈕的實體名稱：

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

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

建立物件實體

物件必須存在，才能在 ActionScript 中使用該物件。建立物件需要先宣告變數，但宣告變數只會在電腦記憶體中建立一個空位置。您必須將實際值指派給變數 (建立物件並將物件儲存在變數中)，才能使用或操作變數。建立物件的程序稱為「實體化」物件。換言之，就是建立特定類別的實體。

若要建立物件實體，有一個簡單的方法完全不需要使用 ActionScript。在 Flash Professional 中，將影片片段元件、按鈕元件或文字欄位放在「舞台」上，然後指定其實體名稱。Flash Professional 就會自動宣告同名變數、建立物件實體，然後將該物件儲存在變數中。同樣地，在 Flex 中，當您編寫 MXML 標籤或將組件放在 Flash Builder 「設計」模式的編輯器時，會在 MXML 中建立組件。為該組件指定 ID，該 ID 就會成為內含該組件實體的 ActionScript 變數的名稱。

不過，您不一定都會建立視學上的物件，對於非視覺性的物件就不可能這麼做。您還可以使用其他方法，只需使用 ActionScript 便可以建立物件實體。

針對幾種 ActionScript 資料類型，您可以使用「常值運算式」(即直接寫入 ActionScript 程式碼的值) 建立實體。下面是一些範例：

- 常值數值 (直接輸入數字)：

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

- 常值 String 值 (用雙引號括住文字)：

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

- 常值 Boolean 值 (使用常值 true 或 false)：

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

- 常值 Array 值 (以方括號括住以逗號分隔的值清單)：

```
var seasons:Array = ["spring", "summer", "autumn", "winter"];
```

- 常值 XML 值 (直接輸入 XML)：

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

ActionScript 也可針對 Array、RegExp、Object 和 Function 資料類型定義常值運算式。

為任何資料類型建立實體，最常用的方法就是使用 new 運算子加上類別名稱，如下所示：

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

使用 new 運算子建立物件通常描述為 「呼叫類別的建構函式」。「建構函式」是特殊的方法，會在建立類別的實體時呼叫。請注意，當您使用這種方法建立實體的時候，要在類別名稱後面加上圓括弧。有時候您會在圓括弧中指定參數值。建立方法時，也會執行以下兩件事。

即使是使用常值運算式建立實體的資料類型，您也可以使用 new 運算子建立物件實體。例如，下面兩行程式碼的作用相同：

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

請務必熟悉使用 new ClassName() 來建立物件的方式。很多 ActionScript 資料類型不會有視覺呈現。因此，利用在 Flash Professional 「舞台」或 Flash Builder 的 MXML 編輯器「設計」模式中放上項目，並無法建立它們。您只能使用 new 運算子，在 ActionScript 建立這些資料類型的實體。

Adobe Flash Professional

在 Flash Professional 中，如果是在 「元件庫」中定義但未放在 「舞台」上的影片片段元件，也可以使用 new 運算子建立其實體。

更多說明主題

使用陣列

使用規則運算式

使用 ActionScript 建立 MovieClip 物件

常見的程式元素

您可以使用一些其他建構區塊來建立 ActionScript 程式。

運算子

「運算子」是用來執行計算的特殊符號 (或文字)。運算子多半用於數學運算，也可以用來比較兩個值。一般來說，運算子會使用一或多個值，然後 「算出」單一結果。例如：

- 加法運算子 (+) 會將兩個值相加，然後產生單一數字：

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

- 乘法運算子 (*) 會將兩個值相乘，然後產生單一數字：

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

- 相等運算子 (==) 會比較兩個值，判斷兩值是否相等，然後產生單一 true 或 false (Boolean) 值：

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

如上所示，相等運算子和其它「比較」運算子常與 if 陳述式搭配使用，來決定是否應執行特定指示。

註解

當您撰寫 ActionScript 時，通常想給自己留下一些注意事項。例如，有時候您想解釋特定幾行程式碼的工作原理以及您做出特殊選擇的原因。「程式碼註解」是在程式碼中加註文字的工具，電腦會忽略這些文字。ActionScript 包含兩種註解：

- 單行註解：單行註解是在一行中的任何位置輸入兩個斜線來表示。電腦會忽略從斜線開始一直到該行結尾之間的內容：

```
// This is a comment; it's ignored by the computer.
var age:Number = 10; // Set the age to 10 by default.
```

- 多行註解：多行註解是由開頭註解記號 (/*)、註解內容，以及結尾註解記號 (*/) 所組成。電腦會忽略開頭和結尾註解記號之間的任何內容 (無論註解有多少行)：

```
/*
This is a long description explaining what a particular
function is used for or explaining a section of code.

In any case, the computer ignores these lines.
*/
```

註解也常用來暫時「關閉」一或多行程式碼。例如，如果您測試不同方法來執行相同動作，就可以使用註解。或者使用註解來嘗試瞭解為什麼某段 ActionScript 程式碼未如預期般運作。

流程控制

很多時候，您會想在程式中重複某些動作、只執行某些動作，或根據特定條件執行其它動作等。「流程控制」會控制哪些動作將會執行。ActionScript 提供幾種流程控制元素。

- 函數：函數就像捷徑一樣。用單一名稱來代表一系列的動作，而且可用來執行計算。函數對處理事件很重要，但也可用來當做一般工具，組合一系列指示。
- 迴圈：迴圈結構可讓您指定一組指示，讓電腦執行多次或直到某個狀況改變為止。迴圈通常用來操作幾個相關的項目，所使用變數的值會隨著電腦每執行一次迴圈而改變。
- 條件陳述式：條件陳述式可指定某些指示，只在特定情況下才執行。條件陳述式也可以用為不同條件提供其他的指示。最常見的條件陳述式是 if 陳述式。if 陳述式會檢查後面括號內的值或運算式。如果值為 true，則會執行大括號內的程式碼。否則會予以忽略。例如：

```
if (age < 20)
{
    // show special teenager-targeted content
}
```

伴隨 if 陳述式的 else 陳述式，可讓您指定如果條件不是 true 時，要執行的其它指示：

```
if (username == "admin")
{
    // do some administrator-only things, like showing extra options
}
else
{
    // do some non-administrator things
}
```

範例：動畫作品集 (Flash Professional)

這個範例設計的目的，是要讓您有機會一睹 ActionScript 程式碼片段能如何拼湊成一個完整應用程式。「動畫作品集」這個範例會示範如何取用現有的線性動畫，並加入其他次要的互動式元素。例如，為客戶建立的動畫，可以整合為線上作品集。您在動畫中加入的互動式行為，會包含兩個可供觀賞者按下的按鈕：一個用來啟動動畫，另一個用來瀏覽至不同的 URL (如作品集選單或作者的首頁)。

建立這個作品項目的程序可以分成下列幾個主要部分：

1 準備 FLA 檔以便用來加入 ActionScript 和互動式元素。

2 建立和加入按鈕。

3 撰寫 ActionScript 程式碼。

4 測試應用程式 .

準備加入互動性

若能在加入互動式元素至動畫中之前，先建立一些容納新增內容的位置，以設定 FLA 檔，相信會提供很大的幫助。這項工作包括：建立可在「舞台」上放置按鈕的實際空間。它也包括在 FLA 檔中建立可分別保存不同項目的「空間」

設定 **FLA** 以便加入互動式元素：

1 建立包含簡單動畫 (如單一移動補間動畫或形狀補間動畫) 的新 FLA 檔。如果您已經有一個內含您在專案中展示的動畫 FLA 檔案，請開啟這個檔案，然後儲存成新的名稱。

2 決定兩個按鈕要顯示在螢幕上的位置。其中一個按鈕會啟動動畫，另一個按鈕則會連結至作者作品集或首頁。如有必要，請在「舞台」上為這個新內容清除或增加一些空間。如果動畫的位置尚未完成，您可以在第一個影格上建立開頭畫面。在此情況下，您可能需要改變動畫的位置，讓動畫在「影格 2」或之後開始播放。

3 在「時間軸」中的其它圖層上方加入新圖層，並命名為 **buttons**。這個圖層就是您新增按鈕的地方。

4 在這個 buttons 圖層上方加入新圖層，並命名為 **actions**。這個圖層是您將 ActionScript 程式碼加入應用程式的地方。

建立並加入按鈕

接下來，您會實際建立按鈕並設定位置，形成互動式應用程式的中心。

建立按鈕並將它加入 **FLA**：

1 使用繪圖工具，在 buttons 圖層上建立第一個按鈕 (「播放」按鈕) 的視覺外觀。例如，繪製上面有文字的水平橢圓形。

2 使用 「選取」工具選取這一個按鈕的所有圖像部分。

3 從主選單中選擇 「修改 > 轉換成元件」。

4 在對話方塊中選擇 「按鈕」做為元件類型，指定元件的名稱，然後按一下 「確定」。

5 選取按鈕之後，在 「屬性」檢測器中將按鈕的實體名稱指定為 **playButton**。

6 重複步驟 1 到 5，建立的按鈕會將觀賞者帶往作者首頁。將這個按鈕命名為 **homeButton**。

撰寫程式碼

這個應用程式的 ActionScript 程式碼可以分成三組功能，但全部都是輸入在同一個位置。程式碼做的三件事分別為：

- 一旦 SWF 檔載入 (當播放磁頭進入 「影格 1」) 時，立即停止播放磁頭。
- 偵聽事件，以便在使用者按一下 「播放」按鈕時開始播放 SWF 檔。
- 偵聽事件，以便在使用者按一下作者首頁按鈕時，讓瀏覽器進入適當的 URL。

建立程式碼，以便在播放磁頭進入 「影格 **1**」時停止它：

1 選取 actions 圖層 「影格 1」上的關鍵影格。

2 若要開啟 「動作」面板，請從主選單中選擇 「視窗 > 動作」。

3 在 Script 窗格中，輸入下列程式碼：

```
stop();
```

撰寫程式碼，以便在按一下 「播放」按鈕時啟動動畫：

1 在上個步驟中輸入的程式碼結尾處，加入兩行空行。

2 在 Script 底部輸入下列程式碼：

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

這個程式碼會定義名為 startMovie() 的函數。當呼叫 startMovie() 時，就會讓主要時間軸開始播放。

3 在緊接上個步驟所加入程式碼之後的那一行，輸入下面這行程式碼：

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

這行程式碼會將 startMovie() 函數註冊為 playButton 之 click 事件的偵聽程式。換句話說，程式碼會設定成每當按一下 playButton 這個按鈕時，便呼叫 startMovie() 函數。

撰寫程式碼，以便在按一下首頁按鈕時，讓瀏覽器瀏覽至某個 **URL**：

1 在上個步驟中輸入的程式碼結尾處，加入兩行空行。

2 在 Script 底部輸入下列程式碼：

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

這個程式碼會定義名為 gotoAuthorPage() 的函數。這個函數會先建立一個 URLRequest 實體，代表 URL http://example.com/。然後它會將 URL 傳遞至 navigateToURL() 函數，讓使用者的瀏覽器開啟這個 URL。

3 在緊接上個步驟所加入程式碼之後的那一行，輸入下面這行程式碼：

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

這行程式碼會將 gotoAuthorPage() 函數註冊為 homeButton 之 click 事件的偵聽程式。換句話說，程式碼會設定成每當按一下 homeButton 這個按鈕時，便呼叫 gotoAuthorPage() 函數。

測試應用程式

應用程式現在具有完整功能了。讓我們來進行測試，確定一下情況。

測試應用程式：

1. 從主選單中選擇 「控制 > 測試影片」。Flash Professional 便會建立 SWF 檔，並在 Flash Player 視窗開啟它。
2. 試著按一下這兩個按鈕，確定它們會依照您所預期的執行。
3. 如果按鈕沒有作用，則需要檢查下列事項：
   - 按鈕的實體名稱是否彼此不同？
   - addEventListener() 方法呼叫所用的名稱是否與按鈕的實體名稱相同？
   - addEventListener() 方法呼叫使用的事件名稱是否正確？
   - 是否為每個函數指定了正確的參數 (兩個方法都需要一個資料類型為 MouseEvent 的參數。)

   以上所有錯誤以及其他最常見的錯誤，都會引發錯誤訊息。當您選擇 「測試影片」命令，或者您在測試專案時按一下按鈕，都會顯示錯誤訊息。請查看 「編譯器錯誤」面板中的編譯器錯誤 (第一次選擇 「測試影片」時發生的錯誤)。檢查 「輸出」面板有無執行階段錯誤 (當內容播放時發生的錯誤；例如，按一下按鈕時)。

使用 **ActionScript** 建立應用程式

撰寫 ActionScript 來建立應用程式時，熟悉語法和要使用的類別名稱僅是整個程序中最為基本的條件而已。大部分的 Flash Platform 文件都包含這兩個主題 (語法和使用 ActionScript 類別)。不過，若要建立 ActionScript 應用程式，您同樣需要了解知道以下資訊：

- 撰寫 ActionScript 時可以使用什麼程式？
- 如何編排 ActionScript 程式碼？
- 如何在應用程式中包括 ActionScript 程式碼？
- 開發 ActionScript 應用程式時要採取的步驟？

組織程式碼的選項

您可以使用 ActionScript 3.0 程式碼來建立許多程式，從簡單的動畫，一直到複雜的主從式交易處理系統。視所要建立的應用程式類型而定，請使用下面的一或多個不同方法，將 ActionScript 加入專案。

將程式碼儲存在 **Flash Professional** 時間軸的影格

在 Flash Professional 中，您可以將 ActionScript 程式碼加入時間軸的任何影格。這個程式碼會在影片播放 (也就是播放磁頭進入該影格) 的同時執行。

將 ActionScript 程式碼放入影格，便可以輕鬆地將行為加入 Flash Professional 建立的應用程式。您可以將程式碼加入主要時間軸的任何影格，或任何 MovieClip 元件時間軸的任何影格。但是，這種彈性有其代價。當您建立較大的應用程式時，很容易就會忘記哪些影格儲存哪些 Script。這種複雜結構會使得應用程式在日後難以維護。

很多開發人員只會將程式碼放在時間軸的第一個影格，或 Flash 文件的特定圖層，藉此在 Flash Professional 中簡化 ActionScript 程式碼的組織。區隔程式碼比較容易從 Flash FLA 檔找出和維護程式碼。不過，如果未經複製程式碼並貼到新的檔案中，相同的程式碼不可以用於另一個 Flash Professional 專案。

如果以後想更容易在其他 Flash Professional 專案中使用您的 ActionScript，請將程式碼儲存至外部的 ActionScript 檔案 (副檔名為 .as 的文字檔)。

在 **Flex MXML** 檔案中嵌入程式碼

在 Flash Builder 之類的 Flex 開發環境中，您可以在 Flex MXML 檔案的 <fx:Script> 標籤中包含 ActionScript 程式碼。不過，這種技術會增加大型專案的複雜度，會更不容易在另一個 Flex 專案中使用相同的程式碼。為了以後更易於在其他 Flex 專案使用您的 ActionScript 程式碼，請將程式碼儲存至外部的 ActionScript 檔案。

備註：您可以為 <fx:Script> 標籤指定來源參數。使用來源參數可以讓您從外部檔案 「匯入」 ActionScript 程式碼，就像是直接在 <fx:Script> 標籤中輸入一樣。不過，您所使用的來源檔案無法定義自己的類別，這樣會限制其重複使用的可能性。

將程式碼儲存在 **ActionScript** 檔案中

如果您的專案有大量的 ActionScript 程式碼，最好能將程式碼組織在不同的 ActionScript 來源檔案 (即副檔名為 .as 的文字檔) 中。ActionScript 檔案的結構有兩種，您可以視其在應用程式中的用途來決定使用哪一種。

- 非結構化 ActionScript 程式碼：ActionScript 程式碼行 (包括陳述式或函數定義) 的寫法就像直接輸入時間軸 Script、MXML 檔案一樣。

  以這種方式撰寫的 ActionScript 可以用 ActionScript 中的 include 陳述式存取，或 Flex MXML 中的 <fx:Script> 標籤存取。ActionScript include 陳述式會告訴編譯器，在特定位置以及程式檔中既有的範圍包含外部 ActionScript 檔案的內容。結果與直接在該處輸入程式碼相同。在 MXML 語言中，使用 <fx:Script> 標籤加上一個來源特質可以識別編譯器在應用程式中該處載入的外部 ActionScript。例如，下列標籤會載入名為 Box.as 的外部 ActionScript 檔案：

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

- ActionScript 類別定義：ActionScript 類別的定義，包括其方法和屬性定義。

  當您定義類別時，您可以建立類別的實體並使用其屬性、方法和事件，就可以存取類別中的 ActionScript 程式碼。使用自己的類別和使用任何內建的 ActionScript 類別並無不同，而且需要兩個部分：

  - 使用 import 陳述式指定類別的完整名稱，讓 ActionScript 編譯器知道從何處找到類別。例如，要在 ActionScript 中使用 MovieClip 類別，可以使用它的完整名稱 (包括套件和類別) 匯入類別。

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

    或者，您也可以匯入包含 MovieClip 類別的套件，就等於是針對套件的各類別分別寫入 import 陳述式：

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

    對於必須匯入類別，才能在程式碼中使用該類別的規則而言，最上層的類別是唯一的例外。這些類別並未定義在套件之中。

  - 撰寫使用特定類別名稱的程式碼。例如，宣告類別的變數及其資料類型，同時建立類別的實體，以便儲存至變數。藉由在 ActionScript 程式碼中使用類別，您便可以指示編譯器載入該類別的定義。例如，已知有一個稱為 Box 的外部類別，這個陳述式會建立 Box 類別的實體：

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

    編譯器第一次參照 Box 類別時，會搜尋可用的原始程式碼，以便找到 Box 類別定義。

選擇正確的工具

您可以使用多個工具的其中一個 (或同時使用多個工具)，撰寫以及編輯自己的 ActionScript 程式碼。

Flash Builder

使用 Flex 框架建立專案或建立主要由 ActionScript 程式碼構成的專案時，Adobe Flash Builder 是首選工具。Flash Builder 也包含功能齊全的 ActionScript 編輯器、視覺版面以及 MXML 編輯功能。它可以用來建立 Flex 或純 ActionScript 專案。Flex 具備多項優點，包括有許多預先建立的使用者介面控制項、彈性的動態版面配置控制項，以及內建多項機制，可讓您使用遠端資料並將外部資料連結到使用者介面元素。不過，由於需要額外的程式碼，才能提供這些功能，因此使用 Flex 的專案所擁有的 SWF 檔案，其大小會比非 Flex 的 SWF 專案更大。

如果您使用 Flex 建立功能完整的資料導向網際網路應用程式時，請使用 Flash Builder。當您想編輯 ActionScript 程式碼、編輯 MXML 程式碼，以及編排應用程式視覺版面時，都可以使用這個工具。

很多建立大量涉及 ActionScript 專案的使用者，都會使用 Flash Professional 來建立視覺資源，並使用 Flash Builder 做為 ActionScript 程式碼的編輯器。

Flash Professional

除了圖形和動畫建立功能之外，Flash Professional 還包括各種處理 ActionScript 程式碼的工具。程式碼可以附加至 FLA 檔案中的元素或附加至外部的純 ActionScript 檔案。Flash Professional 最適用於處理涉及大量動畫或視訊的專案。當您想自行建立大部分的圖形資源時，就可瞭解其價值。使用 Flash Professional 開發 ActionScript 專案的另一個理由，就是在相同的應用程式中建立視覺資源以及撰寫程式碼。Flash Professional 也包括預先建立的使用者介面組件。您可以使用這些組件，讓 SWF 檔案變小一些，以及使用視覺工具，建立這些組件的視覺效果，使用於專案中。

Flash Professional 提供兩種工具來撰寫 ActionScript 程式碼：

- 「動作」面板：處理 FLA 檔時使用，這個面板可讓您撰寫加入時間軸影格的 ActionScript 程式碼。
- 「Script」視窗：「Script」視窗是專門處理 ActionScript (.as) 程式碼檔案的文字編輯器。

協力廠商 **ActionScript** 編輯器

由於 ActionScript (.as) 檔案是簡單的文字檔，因此只要程式能夠編輯純文字檔，就可用來撰寫 ActionScript 檔案。除了 Adobe 的 ActionScript 產品，也有幾家協力廠商推出具備 ActionScript 專屬功能的文字編輯程式。您可以使用任何文字編輯器程式撰寫 MXML 檔案或 ActionScript 類別。然後您可以使用 Flex SDK，從這些檔案建立應用程式。專案可以使用 Flex 或是純 ActionScript 應用程式。也有部分開發人員使用 Flash Builder 或協力廠商 ActionScript 編輯器來撰寫 ActionScript 類別，再結合 Flash Professional 用於建立圖形內容。

選擇協力廠商 ActionScript 編輯器的理由包括以下幾項：

- 您偏好在個別程式中撰寫 ActionScript 程式碼，並於 Flash Professional 中設計視覺元素。
- 在設計非 ActionScript 程式時使用了某項應用程式 (例如用其它程式語言建立 HTML 網頁或建立應用程式)。您想使用同樣的應用程式來撰寫 ActionScript 程式碼。
- 想要使用 Flex SDK 建立純 ActionScript 或 Flex 專案，但不想用到 Flash Professional 或 Flash Builder。

提供 ActionScript 專屬支援的幾個重要程式碼編輯器包括：

- Adobe Dreamweaver® CS4
- ASDT
- FDT
- FlashDevelop
- PrimalScript
- SE|PY
- TextMate (搭配 ActionScript 和 Flex 組合)

ActionScript 開發程序

不論您的 ActionScript 專案大小為何，利用一種程序來設計以及開發應用程式讓工作更有效率、更實用。下列步驟說明使用 ActionScript 3.0 建立應用程式的基本開發程序：

1 設計應用程式。

建立應用程式之前，先用某種方式描述要建立的應用程式。

2 撰寫 ActionScript 3.0 程式碼。

您可以使用 Flash Professional、Flash Builder、Dreamweaver 或文字編輯器建立 ActionScript 程式碼。

3 建立 Flash 或 Flex 專案來執行程式碼。

在 Flash Professional 中，建立一個 FLA 檔案、指定發佈設定值、將使用者介面組件新增至應用程式，以及參照 ActionScript 程式碼。在 Flex 中，定義應用程式、使用 MXML 新增使用者介面組件，以及參照 ActionScript 程式碼。

4 發佈並測試 ActionScript 應用程式。

測試應用程式涉及到從開發環境中執行應用程式，同時確定您希望做到的每一件事，統統都做到了。

您不一定需要照著上述步驟的順序，或等到某項步驟完成再進行下一步。例如，您可以設計應用程式的某個畫面 (步驟 1)，接著建立圖形、按鈕等 (步驟 3)，然後再撰寫 ActionScript 程式碼 (步驟 2) 和測試 (步驟 4)。或者，您可以先設計部分內容，接著一次加入一個按鈕或介面元素，並分別撰寫其 ActionScript，然後在建立之後進行測試。記住開發程序的這 4 個階段，會很有幫助。不過，在實際的開發中，在各個階段之間適當地往返會更有效率。

建立自己的類別

建立專案所需的類別看起來可能相當不容易。不過，建立類別時，比較困難的部分是設計類別的方法、屬性和事件。

設計類別的策略

物件導向設計是相當複雜的一項主題，而關於這項主題的學術研究和應用實務不勝枚舉。不過，下面幾個建議方法可幫助您入門。

1 考量這個類別的實體將在應用程式中扮演什麼角色。一般來說，物件的角色包括：

- 值物件：這些物件主要用途是做為資料的容器。它們的可能有幾個屬性和方法 (或者有時候沒有任何方法)。它們一般是明確定義之項目的程式碼。例如，音樂播放程式可以包含 Song 類別，表示實際一首歌，另外有一個 Playlist 類別，表示概念上的歌曲匯總。
- 顯示物件：這些是實際出現在畫面上的物件。例如下拉式清單或狀態報告等使用者介面元素，或像是電玩人物的圖形元素等。
- 應用程式結構：這些物件在應用程式執行的邏輯或處理中扮演各種支援角色。例如，您可以建立一個物件，在模擬生物學中執行特定計算。您可以建立一個物件，負責同步音樂播放程式的刻度控制與音量輸出。另一個物件可以管理電玩中的規則。或者您可以建立一個類別，在繪圖應用程式中載入儲存的相片。

2 決定類別需要的特定功能。不同類型的功能通常會成為類別的方法。

3 如果類別要當做值物件使用，則需決定實體將包含哪些資料。這些項目可以當做屬性。

4 由於類別是特別為專案所設計，因此重點在於提供應用程式所需的功能。試著自行回答這些問題：

- 應用程式將儲存、追蹤和操作哪些資訊？回答這個問題可以幫助您找出需要的任何值物件和屬性。
- 應用程式會執行哪些動作？例如，應用程式第一次載入時、按下特定按鈕後、電影停止播放時，會發生什麼事？這些項目都可以當做方法。如果「動作」涉及變更個別的值，也可以是屬性。
- 至於任何指定的動作，執行動作時需要什麼資訊？這些資訊就是方法的參數。
- 隨著應用程式執行，哪些情況會使類別產生變化，而且要讓應用程式的其它部分知道？這些項目就可以當做事件。

5 現在是否有任何物件和您需要的物件類似，但缺少一些您想添加的額外功能？考慮建立一個子類別。(「子類別」是一種以現有類別的功能為基礎而建立的類別，不是定義它自己所有的功能。) 例如，如需建立屬於螢幕視覺物件的類別，可以將現有的顯示物件當做是類別的基礎。在這種情況下，顯示物件 (例如 MovieClip 或 Sprite) 會是「基底類別」，而您的類別將延伸該類別。

撰寫類別的程式碼

當您對類別有了某種設計，或者至少對於它應該儲存的資訊以及要執行的動作有了一些構思之後，實際的類別撰寫語法就簡單容易多了。

下面是建立 ActionScript 類別的基本步驟：

1 使用您的 ActionScript 文字編輯器程式開啟新的純文字文件。

2 輸入 class 陳述式來定義類別的名稱。若要新增 class 陳述式，請輸入 public class 這兩個字，然後輸入類別的名稱。新增左、右方大括號來包含類別的內容 (方法和屬性定義)。例如：

```
public class MyClass
{
}
```

public 一字代表可以從任何其它程式碼存取類別。如需其它替代方式，請參閱存取控制命名空間特質。

3 輸入 package 陳述式，指示包含類別的套件名稱。語法是指 package 這個字，後面加上完整的套件名稱，接下來是 class 陳述式周圍的左、右大括號)。例如，將前一個步驟的程式碼變更成以下內容：

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

4 在類別主體內，使用 var 陳述式定義類別中的每個屬性。語法就跟宣告任何變數的語法一樣 (加上 public 修飾詞)。例如，在類別定義的左右大括號之間加入下面幾行，會建立名為 textProperty、numericProperty 以及 dateProperty 的屬性：

```
public var textProperty:String = "some default value";
public var numericProperty:Number = 17;
public var dateProperty:Date;
```

5 使用定義函數的相同語法來定義類別中的每個方法。例如：

- 若要建立 myMethod() 方法，請輸入：

```
public function myMethod(param1:String, param2:Number):void
{
    // do something with parameters
}
```

- 若要建立建構函式 (在建立類別的實體時呼叫的特殊方法)，請建立名稱與類別完全相符的方法：

```
public function MyClass()
{
    // do stuff to set initial values for properties
    // and otherwise set up the object
    textVariable = "Hello there!";
    dateVariable = new Date(2001, 5, 11);
}
```

如果您未在類別中包括建構函式方法，編譯器會自動在類別中建立一個空白的建構函式。(也就是，不含參數而且沒有陳述式的建構函式。)

您可以定義一些其他類別元素。這些元素更複雜一些。

- 「存取子」是介於方法與屬性之間的特殊混合體。當您撰寫程式碼來定義類別時，可以將存取子寫成方法。您可以執行多個動作，而不是像定義屬性一樣只能讀取或指定值。但是，當您建立類別的實體時，會將存取子視為屬性，並使用名稱來讀取或指定值。
- ActionScript 中的事件不是使用特定語法定義的。您反而應該使用 EventDispatcher 類別的功能，在您的類別中定義事件。

更多說明主題

處理事件

範例：建立基本應用程式

ActionScript 3.0 可用於多種應用程式開發環境中，其中包括 Flash Professional 和 Flash Builder 工具或者任何文字編輯器。

本範例將逐步說明如何使用 Flash Professional 或 Flash Builder 工具，建立和增強簡單的 ActionScript 3.0 應用程式。您將建立的應用程式，會顯示在 Flash Professional 和 Flex 應用程式中使用外部 ActionScript 3.0 類別檔案的簡單模式。

設計 ActionScript 應用程式

此範例 ActionScript 應用程式是標準的 "Hello World" 應用程式，所以它的設計很簡單：

- 這個應用程式稱為 HelloWorld。
- 它會顯示單一文字欄位，其中包含文字 "Hello World!"
- 這個應用程式使用一個稱為 Greeter 的單一物件導向類別。這種設計可以使用 Flash Professional 或 Flex 專案中的類別。
- 在這個範例中，您會先建立應用程式的基本版本。然後新增功能，讓使用者輸入使用名稱，並讓應用程式在已知的使用者清單中檢查這個名稱。

有了清楚的定義之後，接下來就可以開始建立應用程式。

建立 HelloWorld 專案和 Greeter 類別

Hello World 應用程式的設計說明指出程式碼應方便重複使用。為了達到這個目標，應用程式會使用一個稱為 Greeter 的單一物件導向類別。您從利用 Flash Builder 或 Flash Professional 建立的應用程式中使用這個類別。

利用 **Flex** 建立 **HelloWorld** 專案和 **Greeter** 類別：

1. 在 Flash Builder 中，選取 「File > New > Flex Project」。
2. 輸入 HelloWorld 做為「Project Name」(專案名稱)。確定 「Application」(應用程式) 類型設定成「Web (runs in Adobe Flash Player)」，然後按一下 「Finish」。

   Flash Builder 會建立您的專案並顯示在 Package Explorer 之中。根據預設，此專案應該已經包含一個名為 HelloWorld.mxml 的檔案，而且編輯器已經開啟該檔案。
3. 現在，若要使用 Flash Builder 建立自訂的 ActionScript 類別檔案，請選取 「File > New > ActionScript Class」。
4. 在 「New ActionScript Class」對話方塊的 「Name」欄位中，輸入 **Greeter** 做為類別名稱，然後按一下 「Finish」。

   新的 ActionScript 編輯視窗隨即顯示。

   後續步驟請參閱下一節將程式碼加入 Greeter 類別。

在 **Flash Professional** 建立 **Greeter** 類別：

1. 在 Flash Professional 中，選取 「檔案 > 新增」。
2. 在 「新增文件」對話方塊中選取 「ActionScript 檔案」，然後按一下 「確定」。

   新的 ActionScript 編輯視窗隨即顯示。
3. 選取 「檔案 > 儲存」。選取一個資料夾來存放應用程式，將 ActionScript 檔案命名為 **Greeter.as**，然後按一下 「確定」。

   後續步驟請參閱下一節將程式碼加入 Greeter 類別。

將程式碼加入至 Greeter 類別

Greeter 類別會定義物件 Greeter，您可以用於 HelloWorld 應用程式。

將程式碼加入 **Greeter** 類別：

1 將下列程式碼輸入新的檔案中 (某些程式碼可能已經新增)：

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

Greeter 類別包含一個單一的 sayHello() 方法，此方法會傳回一個 "Hello World!" 的字串。

2 選取 「檔案 > 儲存檔案」來儲存這個 ActionScript 檔案。

現在，Greeter 類別已經就緒，可以在應用程式中使用了。

建立使用 ActionScript 程式碼的應用程式

前面所建立的 Greeter 類別定義了一組獨立的軟體功能，但這還不是完整的應用程式。若要使用這個類別，您可以建立 Flash Professional 文件或 Flex 專案。

程式碼需要 Greeter 類別的實體。下面顯示如何在應用程式使用 Greeter 類別。

使用 **Flash Professional** 建立 **ActionScript** 應用程式：

1 請選取 「檔案 > 新增」。

2 在 「新增文件」對話方塊中選取 「Flash 檔案 (ActionScript 3.0)」，然後按一下 「確定」。

新的文件視窗隨即顯示。

3 選取 「檔案 > 儲存」。選取存放 Greeter.as 類別檔案所在的同一個資料夾，將 Flash 文件命名為 **HelloWorld.fla**，然後按一下 「確定」。

4 在 Flash Professional 工具面板中，選取 「文字」工具。在 「舞台」中拖曳，以便定義寬度大約 300 像素、寬度約 100 像素的新文字欄位。

5 在 「屬性」面板中，當 「舞台」上的文字欄位依然選取時，將文字類型設定為 「動態文字」。輸入 **mainText** 做為文字欄位的實體名稱。

6 按一下主要時間軸的第一個影格。選擇 「視窗 > 動作」來開啟 「動作」面板。

7 在 「動作」面板中，輸入下列 Script：

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

8 儲存檔案。

後續步驟請參閱發佈並測試 ActionScript 應用程式。

使用 **Flash Builder** 建立 **ActionScript** 應用程式：

1 開啟 HelloWorld.mxml 檔案，新增程式碼以符合下列內容：

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/halo"
    minWidth="1024"
    minHeight="768""
    creationComplete="initApp()">

    <fx:Script>
        <![CDATA[
            private var myGreeter:Greeter = new Greeter();

            public function initApp():void
            {
                // says hello at the start, and asks for the user's name
                mainTxt.text = myGreeter.sayHello();
            }
        ]]>
    </fx:Script>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:TextArea id="mainTxt" width="400"/>

</s:Application>
```

這個 Flex 專案包含四個 MXML 標籤：

- <s:Application> 標籤，定義應用程式容器
- <s:layout> 標籤，定義 Application 標籤的版面樣式 (垂直版面)
- <fx:Script> 標籤，其中包含特定 ActionScript 程式碼
- <s:TextArea> 標籤，定義一個欄位來顯示文字訊息給使用者

<fx:Script> 標籤中的程式碼會定義應用程式載入時呼叫的 initApp() 方法。initApp() 方法會將 mainTxt TextArea 的文字值設為 "Hello World!" 字串，這是由您剛撰寫之 Greeter 自訂類別的 sayHello() 方法所傳回的字串。

2 選取「File > Save」，儲存應用程式。

後續步驟請參閱發佈並測試 ActionScript 應用程式。

發佈並測試 ActionScript 應用程式

軟體開發是不斷重複同樣步驟的過程。首先是撰寫程式碼、嘗試編譯程式碼，然後編輯程式碼，直到編譯沒問題。您將會執行編譯後的應用程式並加以測試，進而查看該應用程式是否具備您想要的設計。如果沒有，您可以再次編輯程式碼，直到達到想要的設計為止。Flash Professional 和 Flash Builder 開發環境提供多種方法來發佈、測試和除錯應用程式。

下面是在各環境中測試 HelloWorld 應用程式的基本步驟。

使用 **Flash Professional** 發佈和測試 **ActionScript** 應用程式：

1 發佈應用程式，並注意是否發生編譯錯誤。在 Flash Professional 中選取「控制 > 測試影片」，編譯 ActionScript 程式碼並執行 HelloWorld 應用程式。

2 如果在您測試應用程式時，「輸出」視窗顯示任何錯誤或警告，請在 HelloWorld.fla 或 HelloWorld.as 檔案中修正這些錯誤。然後再次測試應用程式。

3 如果沒有發生編譯錯誤，您就會看到 Hello World 應用程式出現在 Flash Player 視窗中。

您剛才建立了採用 ActionScript 3.0 的物件導向應用程式，雖然簡單，卻很完整。接下來，請繼續強化 HelloWorld 應用程式。

使用 **Flash Builder** 發佈和測試 **ActionScript** 應用程式：

1 選取「Run > Run HelloWorld」。

2 HelloWorld 應用程式隨即啟動。

- 如果在您測試應用程式時，「Output」視窗顯示任何錯誤或警告，請在 HelloWorld.mxml 或 Greeter.as 檔案中修正錯誤。然後再次測試應用程式。
- 如果沒有發生編譯錯誤，您就會看到 Hello World 應用程式出現在瀏覽器視窗中。文字 "Hello World!" 會接著顯示出來。

您剛才建立了採用 ActionScript 3.0 的物件導向應用程式，雖然簡單，卻很完整。接下來，請繼續強化 HelloWorld 應用程式。

強化 HelloWorld 應用程式

為了讓應用程式多點變化，您現在可以設定應用程式要求使用者輸入名稱，然後根據預先定義的名稱清單進行驗證。

首先需要更新 Greeter 類別來加入新功能。接著再更新應用程式來使用新功能。

更新 **Greeter.as** 檔案：

1 開啟 Greeter.as 檔案。

2 將檔案內容變更如下 (新增和變更的行會以粗體顯示)：

```
package
{
    public class Greeter
    {
        /**
         * Defines the names that 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;
            }
        }
    }
}
```

Greeter 類別現在有了數項新功能：

- validNames 陣列會列出有效的使用者名稱。當載入 Greeter 類別時，這個陣列會初始化為包含三個名稱的清單。
- sayHello() 方法現在可接受使用者名稱，並視情況變更問候內容。如果 userName 是空字串 ("")，greeting 屬性會設定提示使用者輸入名稱。如果使用者名稱有效，問候內容會變成 "Hello, userName"。如果最後這兩個條件都不成立，greeting 變數會設為 "Sorry userName, you are not on the list"。
- 如果在 validNames 陣列找到 inputName，validName() 方法會傳回 true，如果找不到，則會傳回 false。陳述式 validNames.indexOf(inputName) 會檢查 validNames 陣列中的每個字串來比對 inputName 字串。Array.indexOf() 方法會傳回陣列中某個物件的第一個實體的索引位置。如果在陣列中找不到物件，就會傳回值 -1。

接下來，編輯參照此 ActionScript 類別的應用程式檔案。

使用 **Flash Professional** 修改應用程式：

1. 開啟 HelloWorld.fla 檔案。
2. 修改「影格 1」中的 Script，改成傳遞空字串 ("") 給 Greeter 類別的 sayHello() 方法：

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

3. 選取工具面板中的「文字」工具。在「舞台」上建立兩個新的文字欄位。將這兩個欄位並排放到現有的 mainText 文字欄位下方。
4. 在第一個新的文字欄位中 (它是標籤)，輸入 **User Name:**。
5. 選取另一個新的文字欄位，並在「屬性」檢測器中選取「Input Text」做為文字欄位的類型。選取「單行」做為「字行類型」。輸入 **textIn** 當做實體名稱。
6. 按一下主要時間軸的第一個影格。
7. 在「動作」面板中，將下面幾行加在現有 Script 的最後面：

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

textIn.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);

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

新的程式碼增加了下列功能：

- 頭兩行只是定義兩個文字欄位的邊框。
- 文字輸入欄位如 textIn 欄位，有一組可供它傳送的事件。您可以使用 addEventListener() 方法，定義會在發生某類型事件時執行的函數。在此例中，這個事件即是在鍵盤上按一個按鍵。
- keyPressed() 自訂函數會檢查按下的按鍵是否為 Enter 鍵。如果是 Enter 鍵，它會呼叫 myGreeter 物件的 sayHello() 方法，將 textIn 文字欄位中的文字當做參數傳遞。這個方法會根據傳入的值傳回問候的字串，而傳回的字串則接著指定給 mainText 文字欄位的 text 屬性。

「影格 1」的完整 Script 如下：

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

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

textIn.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);

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

8 儲存檔案。

9 選取「控制 > 測試影片」，執行應用程式。

當您執行應用程式時，應用程式會提示您輸入使用者名稱。如果您輸入有效的名稱 (Sammy、Frank 或 Dean)，應用程式便會顯示確認訊息 "hello"。

使用 **Flash Builder** 修改應用程式：

1 開啟 HelloWorld.mxml 檔案。

2 接下來，修改 <mx:TextArea> 標籤，向使用者說明只供顯示。將背景顏色變更成淺灰色，並將 editable 特質設定成 false：

```
    <s:TextArea id="mainTxt" width="400" backgroundColor="#DDDDDD" editable="false" />
```

3 現在，在 <s:TextArea> 結束標籤正後方加上下列幾行。這幾行會建立 TextInput 組件，讓使用者輸入使用者名稱值：

```
    <s:HGroup width="400">
        <mx:Label text="User Name:"/>
        <s:TextInput id="userNameTxt" width="100%" enter="mainTxt.text =
myGreeter.sayHello(userNameTxt.text);" />
    </s:HGroup>
```

enter 特質會定義使用者在 userNameTxt 欄位中按下 Enter 鍵所發生的事。在這個範例中，程式碼會將欄位中的文字傳遞至 Greeter.sayHello() 方法。mainTxt 欄位中的問候內容會跟著一起變更。

HelloWorld.mxml 檔案類似如下：

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/halo"
    minWidth="1024"
    minHeight="768""
    creationComplete="initApp()">

    <fx:Script>
        <![CDATA[
            private var myGreeter:Greeter = new Greeter();

            public function initApp():void
            {
                // says hello at the start, and asks for the user's name
                mainTxt.text = myGreeter.sayHello();
            }
        ]]>
    </fx:Script>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:TextArea id="mainTxt" width="400" backgroundColor="#DDDDDD" editable="false"/>

    <s:HGroup width="400">
        <mx:Label text="User Name:"/>
        <s:TextInput id="userNameTxt" width="100%" enter="mainTxt.text =
myGreeter.sayHello(userNameTxt.text);" />
    </s:HGroup>

</s:Application>
```

4 儲存編輯過的 HelloWorld.mxml 檔案。選取「Run > Run HelloWorld」，執行應用程式。

當您執行應用程式時，應用程式會提示您輸入使用者名稱。如果您輸入有效的名稱 (Sammy、Frank 或 Dean)，應用程式便會顯示 " Hello, userName" 確認訊息。

第 3 章 ActionScript 語言和語法

ActionScript 3.0 包含核心 ActionScript 語言和 Adobe Flash Platform 應用程式設計介面 (API)。核心語言是 ActionScript 的一部分，用於定義語言的語法以及最上層的資料類型。ActionScript 3.0 可以利用程式的方式來存取 Adobe Flash Platform 執行階段：Adobe Flash Player 以及 Adobe AIR。

語言概觀

物件是 ActionScript 3.0 語言的核心要素，也是基本的建構單元。您所宣告的每一個變數、所撰寫的每一個函數，以及所建立的每一個類別實體都是物件。您可以將 ActionScript 3.0 程式視為一組執行工作的物件，這些物件會回應事件，並且彼此進行通訊。

熟悉 Java 或 C++ 物件導向程式設計 (OOP) 的程式設計人員可能會將物件視為包含以下兩種成員的模組：儲存在成員變數或屬性的資料，以及可透過方法存取的行為指令。ActionScript 3.0 會以相似但略有不同的方式來定義物件。在 ActionScript 3.0 中，物件只是屬性的集合。這些屬性是容器，不但能保存資料，也能保存函數或其它物件。若函數是以這種方式附加到物件上，則稱為方法。

雖然具有 Java 或 C++ 背景的程式設計人員可能會覺得 ActionScript 3.0 的定義有些奇怪，但是在實際作業時，以 ActionScript 3.0 類別定義物件類型其實與 Java 或 C++ 中定義類別的方式非常類似。兩種物件定義的區別只有在討論 ActionScript 物件模型及其它進階主題時才會有意義，但在其它大部分情況下，所謂「屬性」一詞是指類別成員變數，與方法相對。例如，「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考 」這本書使用 「屬性」這個詞彙，來表示變數或 getter-setter 屬性。並使用 「方法」一詞代表屬於類別之一部分的函數。

ActionScript 中的類別與 Java 或 C++ 中的類別之間有一項微妙的差異，就是在 ActionScript 中，類別不僅只是抽象實體。ActionScript 類別是以儲存類別之屬性和方法的「類別物件」來表示，因此，您便能夠運用 Java 和 C++ 程式設計人員可能會覺得性質很不相同的技巧 (例如，在類別或套件的最上層包含陳述式或可執行的程式碼)。

ActionScript 類別與 Java 或 C++ 類別之間還有另外一項差異，就是每一個 ActionScript 類別都有所謂的 「原型物件」。在舊版 ActionScript 中，原型物件連結在一起形成「原型鏈」，而整體做為整個類別繼承階層的基礎；但是在 ActionScript 3.0 中，原型物件在繼承系統中只扮演份量很輕的小角色，原型物件依舊十分有用，它可以替代靜態屬性和方法，在類別的所有實體間共享屬性及其值。

在過去，進階 ActionScript 程式設計人員都可以利用特殊的內建語言元素，直接操控屬性鏈；既然語言現在可提供更成熟的類別架構式程式設計介面實作，這些特殊語言元素中有很多 (例如 __proto__ 和 __resolve) 已不再是語言的一部分。而且，將內部繼承機制最佳化的作法讓效能大幅提升，可防止直接存取繼承機制。

物件和類別

在 ActionScript 3.0 中，每一個物件都是由類別所定義。類別可視為物件類型的範本或藍圖。類別定義可以包含變數和常數 (保存資料值) 以及方法 (封裝繫結至類別之行為的函數)。儲存在屬性中的值可以是 「基本值」或其它物件。基本值是數字、字串或 Boolean 值。

ActionScript 包含一些屬於核心語言的內建類別。這些內建類別中，某些類別 (例如 Number、Boolean 和 String) 代表 ActionScript 中可用的基本值；其它例如 Array、Math 和 XML 類別，則會定義更為複雜的物件。

所有類別，不論是內建或使用者定義，都是衍生自 Object 類別。具有舊版 ActionScript 經驗的程式設計人員一定要注意：即使所有其它類別仍然衍生自 Object 資料類型，Object 資料類型已不再是預設資料類型。在 ActionScript 2.0 中，下列兩行程式碼是相等的，因為如果沒有類型註釋，就代表變數是 Object 類型：

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

然而 ActionScript 3.0 則導入不具類型的變數概念，可以用下列兩種方式指定：

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

不具類型的變數與 Object 類型變數不同，最主要差別在於不具類型的變數可以保存特殊值 undefined，而 Object 類型的變數則不能保存該值。

您可以使用 class 關鍵字，自行定義類別。有三種方式可以用來宣告類別屬性：常數可以用 const 關鍵字定義、變數可以用 var 關鍵字定義，而 getter 和 setter 屬性則是在方法宣告中使用 get 和 set 特質定義。此外，您也可以 function 關鍵字宣告方法。

類別實體是使用 new 運算子所建立。下列範例會建立 Date 類別的實體，稱為 myBirthday。

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

套件和命名空間

套件和命名空間是相關的概念。套件可以讓您將類別定義合併在一起，以加強程式碼共享，並降低命名衝突；命名空間可以讓您控制識別名稱的可見性，例如屬性和方法名稱，而且不管位於套件之內或之外，都可以套用至程式碼。套件可以讓您組織類別檔案，而命名空間可以讓您管理各個屬性和方法的可見性。

套件

在 ActionScript 3.0 中，套件都會使用命名空間實作，但並非與命名空間同義。宣告套件時，會明確地建立特殊類型的命名空間，保證在編譯階段為已知；而命名空間雖然明確建立，卻並不一定保證能在編譯階段為已知。

下列範例會使用 package 指令建立簡單的套件，其中包含一個類別：

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

在此範例中的類別名稱是 SampleCode。由於此類別是在 samples 套件內，因此編譯器會在編譯階段自動限定類別名稱，成為完整名稱：samples.SampleCode。編譯器也會限定任何屬性或方法的名稱，讓 sampleGreeting 和 sampleFunction() 分別成為 samples.SampleCode.sampleGreeting 和 samples.SampleCode.sampleFunction()。

許多開發人員，尤其是具有 Java 程式設計背景的開發人員，可能會選擇只將類別放置於套件的最上層。但是 ActionScript 3.0 不但支援位於套件最上層的類別，也支援變數、函數，甚至還支援陳述式。這項功能的進階用法是在套件最上層定義命名空間，讓它能夠供該套件中的所有類別使用。但是請注意，套件最上層只允許兩個存取指定字 public 和 internal。與 Java 不同的是，ActionScript 3.0 不支援巢狀類別，也不支援私有類別，而 Java 允許您將巢狀類別宣告為私有。

但是在其它許多方面，ActionScript 3.0 套件與 Java 程式語言中的套件很類似。在上一個範例中您可以看到，完整的套件參考是使用點運算子 (.) 來表示，在 Java 中也一樣。您可以使用套件將程式碼組織成直覺式的階層結構，供其他程式設計人員使用。如此一來，您可以自行建立套件與他人共享，並能在您的程式碼中使用他人建立的套件，對共享程式碼方面來說頗有助益。

使用套件也有助確保您所使用的識別名稱為唯一，而不會與其它識別名稱相衝突。事實上，有些人認為這是套件最大的好處。例如，兩個程式設計人員想要彼此共享程式碼，便可以各自建立名為 SampleCode 的類別。若沒有套件，就會造成名稱衝突，而且唯一的解決辦法就是重新命名其中一個類別；但是有了套件，名稱衝突很容易避免，只要將其中一個類別命名為唯一的名稱並放在套件中 (最好是兩個同時做)，便可解決。

您也可以在套件名稱中包含嵌入的點，以建立巢狀套件；如此可以讓您建立套件的階層式組織。例如 ActionScript 3.0 中提供的 flash.display，便是很好的範例。flash.display 套件以巢狀方式位於 flash 套件中。

大多數的 ActionScript 3.0 都是在 flash 套件之下加以組織。例如，flash.display 套件包含顯示清單 API，而 flash.events 套件則包含新的事件模式。

建立套件

ActionScript 3.0 在組織套件、類別和來源檔案的方式上，提供相當大的彈性。舊版 ActionScript 對每個來源檔案只允許一個類別，且要求來源檔案的名稱與類別名稱相符。ActionScript 3.0 則可以讓您在一個來源檔案中包含多個類別，但每個檔案中只有一個類別可供該檔案外部的程式碼使用。也就是說，每個檔案中有一個類別可以在套件宣告之中進行宣告。您必須在套件定義之外宣告其它任何類別，而讓位於來源檔案之外的程式碼無法看見這些類別。在套件定義之內宣告的類別名稱必須與來源檔案的名稱相符。

ActionScript 3.0 在宣告套件的方式上，也提供更大的彈性。在舊版的 ActionScript 中，套件只代表放置來源檔案的目錄，且不是用 package 陳述式宣告套件，而是在類別宣告中包含套件名稱，做為完整類別名稱的一部分。雖然套件仍然代表 ActionScript 3.0 中的目錄，但套件可以包含的不只是類別而已。ActionScript 3.0 中使用 package 陳述式宣告套件。也就是說，您也可以在套件最上層宣告變數、函數和命名空間，甚至可以在套件最上層納入可執行陳述式。若確實在套件最上層宣告變數、函數或命名空間，則在該階層唯一可用的特質是 public 和 internal，而且每個檔案只有一個套件層級宣告可以使用 public 特質，不管該宣告是類別、變數、函數或命名空間都可以。

套件對組織程式碼及防止名稱衝突都很有用。請勿混淆了套件的概念與不相關的類別繼承概念。位於相同套件中的兩個類別將會有相同的命名空間，但彼此不一定會在其它任何方面相關；同樣地，巢狀套件與其父套件也可能沒有任何語意關聯性。

匯入套件

若要使用位於套件中的類別，您必須匯入套件或特定的類別。這與 ActionScript 2.0 不同，在該版本中，匯入類別是選擇性的。

例如，考慮前面提出的 SampleCode 類別範例。若類別位於名為 samples 的套件中，您必須先使用下列其中一項 import 陳述式，才能使用 SampleCode 類別：

```
import samples.*;
```

或

```
import samples.SampleCode;
```

一般來說，import 陳述式應該盡可能明確。若想要只使用 samples 套件的 SampleCode 類別，應該只匯入 SampleCode 類別，而不是匯入類別所屬的整個套件。匯入整個套件可能會導致無法預期的名稱衝突。

您也必須將定義套件或類別的原始碼放入 「類別路徑」中。類別路徑是使用者定義的清單，可決定編譯器所要搜尋匯入套件及類別的本機目錄路徑。類別路徑有時也稱為 「組建路徑」或 「來源路徑」。

在您正確匯入類別或套件之後，就可以使用類別的完整名稱 (samples.SampleCode)，也可以僅使用類別名稱本身 (SampleCode)。

在以完全相同的名稱命名類別、方法或屬性而形成模棱兩可的程式碼時，完整名稱會很有用，但是若用於所有識別名稱，則可能會很難管理。例如，在實體化 SampleCode 類別實體時，使用完整名稱會造成冗長的程式碼：

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

隨著巢狀套件的階層增加，程式碼的可讀性也跟著降低。在您確信模棱兩可的識別名稱不會成為問題時，可以使用簡單的識別名稱，讓程式碼更容易閱讀。例如，若只使用類別識別名稱，則實體化 SampleCode 類別的新實體就不致於太過冗長：

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

若嘗試使用識別名稱，但並未先匯入適當的套件或類別，編譯器將找不到類別定義；另一方面，若確實匯入套件或類別，任何嘗試定義與匯入名稱造成衝突的名稱都會產生錯誤。

建立套件時，該套件中所有成員的預設存取指定字是 internal，也就是說，根據預設，只有該套件的其它成員才看得見套件成員。若要讓類別供套件外的程式碼使用，必須將類別宣告為 public。例如，下列套件包含 SampleCode 和 CodeFormatter 兩個類別：

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

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

SampleCode 類別可以在套件外看見，因為它是宣告為 public 類別；但是 CodeFormatter 類別則只能在 samples 套件本身之中才能看見。若嘗試在 samples 套件外存取 CodeFormatter 類別，將產生錯誤，如下列範例所示：

```
import samples.SampleCode;
import samples.CodeFormatter;
var mySample:SampleCode = new SampleCode(); // okay, public class
var myFormatter:CodeFormatter = new CodeFormatter(); // error
```

若要讓兩個類別都能供套件外的程式碼使用，必須將這兩個類別都宣告為 public。您不能將 public 特質套用至套件宣告。

若要解決使用套件時可能會發生的名稱衝突，完整名稱就很有用。匯入兩個用相同識別名稱定義類別的套件，就可能會發生名稱衝突。例如，以下列套件為例，它也包含名為 SampleCode 的類別：

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

若匯入這兩個類別 (如下所示)，在使用 SampleCode 類別時就會發生名稱衝突：

```
import samples.SampleCode;
import langref.samples.SampleCode;
var mySample:SampleCode = new SampleCode(); // name conflict
```

編譯器無法得知要使用哪一個 SampleCode 類別。若要解決這項衝突，必須使用各個類別的完整名稱，如下所示：

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

備註：具有 C++ 背景的程式設計人員經常將 import 陳述式與 #include 搞混。在 C++ 中必須使用 #include 指令，因為 C++ 編譯器一次只處理一個檔案，而且除非明確包含標頭檔，否則不會在其它檔案中搜尋類別定義。ActionScript 3.0 中包含 include 指令，但它並不是設計用來匯入類別和套件。若要將類別或套件匯入 ActionScript 3.0 中，您必須使用 import 陳述式，並將包含套件的來源檔案放入類別路徑中。

命名空間

命名空間可讓您控制所建立屬性和方法的可見性。請將 public、private、protected 和 internal 存取控制指定字視為內建的命名空間。若這些預先定義的控制指定字不能配合您的需求，您可以自行建立命名空間。

您若熟悉 XML 命名空間，您對此處所列的大部分的討論勢必不陌生，不過，ActionScript 實作的語法和細節都會與 XML 稍有差異；若您從未使用過命名空間，概念本身簡單明瞭，但您必須學習有關實作的特定專門用語。

若要瞭解命名空間的運作方式，則知道屬性或方法的名稱永遠包含識別名稱和命名空間兩個部分將會有幫助。識別名稱是您一般所想的名稱。例如，下列類別定義中的識別名稱是 sampleGreeting 和 sampleFunction()：

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

只要定義之前沒有命名空間特質，它們的名稱就會以預設 internal 命名空間加以限定，也就是說，只有在相同套件中的呼叫者才看得見。若編譯器設定為嚴謹模式，編譯器就會發出警告，說明 internal 命名空間會在沒有命名空間特質的情況下套用至任一識別名稱。為確保識別名稱於何處都可供使用，您必須特地在識別名稱之前加上 public 特質。在舊版範例程式碼中，sampleGreeting 和 sampleFunction() 都有 internal 命名空間值。

使用命名空間時，要遵循三個基本步驟。首先，您必須使用 namespace 關鍵字來定義命名空間。例如，下列程式碼會定義 version1 命名空間：

```
namespace version1;
```

接著，在屬性或方法宣告中使用命名空間而非存取控制指定字，以套用命名空間。下列範例會將名為 myFunction() 的函數放入 version1 命名空間中：

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

一旦套用命名空間之後，您就可以使用 use 指令，或是使用命名空間限定識別名稱的名稱來進行參考。下列範例會透過 use 指令，參考 myFunction() 函數：

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

您也可以使用完整名稱來參考 myFunction() 函數，如下列範例所示：

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

定義命名空間

命名空間包含一個值，也就是「統一資源識別名稱」(URI)，它有時候也稱為「命名空間名稱」。URI 可以讓您確保您的命名空間為唯一。

您是透過兩種方式的其中一種，宣告命名空間定義來建立命名空間。您可以用明確的 URI 定義命名空間，就像定義 XML 命名空間一樣，或者也可以省略 URI。下列範例將說明如何使用 URI 定義命名空間：

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

URI 是做為該命名空間的唯一識別字串。若省略 URI (如下列範例所示)，則編譯器將建立唯一的內部識別字串以取代 URI。您無法存取這個內部識別字串。

```
namespace flash_proxy;
```

一旦定義命名空間之後，不管有沒有 URI，命名空間都不能在相同範圍中重新定義。嘗試定義先前在相同範圍中定義的命名空間會導致編譯器錯誤。

若命名空間是在套件或類別中定義，除非使用了適當的存取控制指定字，否則套件或類別之外的程式碼可能會看不見命名空間。舉例來說，下列程式碼會顯示在 flash.utils 套件中定義的 flash_proxy 命名空間。在下列範例中沒有存取控制指定字，也就是說，只有 flash.utils 套件之內的程式碼才能看見 flash_proxy 命名空間，而在套件之外的任何程式碼都無法看見該命名空間：

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

下列程式碼會使用 public 特質，讓套件之外的程式碼可以看見 flash_proxy 命名空間：

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

套用命名空間

套用命名空間就是將定義放入命名空間中。可以放入命名空間中的定義包括函數、變數和常數 (您不能將類別放入自訂命名空間中)。

例如，請考慮使用 public 存取控制命名空間宣告的函數。使用函數定義內的 public 特質將函數放入公用命名空間中，可讓所有程式碼都能使用該函數。一旦定義命名空間之後，使用所定義命名空間的方式與使用 public 特質相同，可參考您自訂命名空間的程式碼都能使用該定義。例如，若您定義命名空間 example1，則可使用 example1 做為特質來加入名為 myFunction() 的方法，如下列範例所示：

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

使用命名空間 example1 做為特質來宣告 myFunction() 方法，表示此方法屬於 example1 命名空間。

套用命名空間時，應該牢記以下事項：

- 每一個宣告只能套用一個命名空間。
- 您無法一次將一個命名空間特質套用到多個定義。也就是說，如果要將您的命名空間套用到十個不同的函數，就必須以命名空間做為特質，加入這十個函數的每一個函數定義中。
- 如果您套用命名空間，那麼也不能指定存取控制指定字，因為命名空間和存取控制指定字是互相排斥的。換句話說，除了套用您的命名空間之外，您不能將函數或屬性宣告為 public、private、protected 或 internal。

參考命名空間

使用以任何存取控制命名空間 (例如 public、private、protected 和 internal) 宣告的方法或屬性時，不需要明確地參考命名空間。這是因為這些特殊命名空間的存取權限是由內容所控制。例如，放入 private 命名空間的定義會自動提供給相同類別內的程式碼使用。但是對於您所定義的命名空間，並沒有這種內容感應式功能。若要使用您放入自訂命名空間的方法或屬性，您必須參考該命名空間。

您可以用 use namespace 指令參考命名空間，或透過使用名稱修飾語 (::) 標點符號的命名空間來限定名稱。用 use namespace 指令參考命名空間，會「開啟」命名空間，以便套用至未限定的任何識別名稱。例如，若已定義 example1 命名空間，則可以使用 use namespace example1 存取該命名空間中的名稱：

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

您一次可以開啟多個命名空間。用 use namespace 開啟了命名空間以後，它會在所開啟的程式碼區塊之中保持為開放。您無法明確地關閉命名空間。

但是，具有多個開啟的命名空間，會增加名稱衝突的可能性。若您不想開啟命名空間，可以使用命名空間和名稱修飾語標點符號來限定方法或屬性名稱，以避免使用 use namespace 指令。例如，下列程式碼將示範如何用 example1 命名空間來限定名稱 myFunction()：

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

使用命名空間

您可以在屬於 ActionScript 3.0 一部分的 flash.utils.Proxy 類別中，找到用來防止名稱衝突之命名空間的實際範例。Proxy 類別是用來取代 ActionScript 2.0 中的 Object.__resolve 屬性，可以讓您在錯誤發生之前，先攔截未定義屬性或方法的參考。為了避免名稱衝突，Proxy 類別的所有方法都位於 flash_proxy 命名空間。

若要更瞭解如何使用 flash_proxy 命名空間，您必須先瞭解如何使用 Proxy 類別。Proxy 類別的功能只適用於繼承自它的類別。也就是說，若要針對物件使用 Proxy 類別的方法，該物件的類別定義必須擴充 Proxy 類別。例如，若要攔截對未定義方法的嘗試呼叫，您必須先擴充 Proxy 類別，然後覆寫 Proxy 類別的 callProperty() 方法。

您可能還記得，實作命名空間通常要經過定義、套用，然後參考命名空間的三步驟程序，但是由於您從未明確地呼叫任何 Proxy 類別方法，所以只會定義和套用 flash_proxy 命名空間，而不會有參考的程序。ActionScript 3.0 會定義 flash_proxy 命名空間，並將在 Proxy 類別中套用這個命名空間。您的程式碼只需要將 flash_proxy 命名空間套用至擴充 Proxy 類別的類別即可。

flash_proxy 命名空間在 flash.utils 套件中定義的方式與下列程序類似：

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

命名空間套用至 Proxy 類別的方法，如下列 Proxy 類別中的摘錄所示：

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

如下列程式碼所示，您必須先匯入 Proxy 類別和 flash_proxy 命名空間。然後，您必須宣告類別以擴充 Proxy 類別 (若要在嚴謹模式中編譯，也必須加入 dynamic 特質)。當您覆寫 callProperty() 方法時，必須使用 flash_proxy 命名空間。

```
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);
        }
    }
}
```

若建立 MyProxy 類別的實體，並呼叫未定義的方法 (如下列範例中呼叫的 testing() 方法)，則您的 Proxy 物件會攔截方法呼叫，並執行已遭覆寫之 callProperty() 方法中的陳述式 (在此範例中，則是簡單的 trace() 陳述式)。

```
var mySample:MyProxy = new MyProxy();
mySample.testing(); // method call intercepted: testing
```

在 flash_proxy 命名空間中擁有 Proxy 類別的方法，將有兩大優點：首先，獨立的命名空間可讓擴充 Proxy 類別的任何類別之公用介面減少堆積現象 (Proxy 類別中可供您覆寫的方法大約有十來個，這些全都不是設計用來直接進行呼叫的。因此，將它們全都放在公用命名空間中可能會造成混淆)。第二，如果 Proxy 子類別中所包含的實體方法名稱與任何 Proxy 類別方法完全相符，使用 flash_proxy 命名空間可以避免名稱衝突。例如，您可能要將自己的方法其中一個命名為 callProperty()。下列為可接受的程式碼，因為您的 callProperty() 方法版本是位於不同的命名空間中：

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

當您想要以四個存取控制指定字 (public、private、internal 和 protected) 無法達到的方式，提供方法或屬性的存取權限時，命名空間可能也有幫助。舉例來說，您可能有一些公用程式方法，分散在幾個不同的套件中。您想要讓所有套件都能使用這些方法，但又不想讓方法變成公用方法。若要達到這項目的，您可以建立命名空間，然後用它做為您自己的特殊存取控制指定字。

下列範例會利用使用者定義的命名空間，將位於不同套件中的兩個函數組合在一起。藉由將這兩個函數組合在相同的命名空間中，您可以讓類別或套件透過單一 use namespace 陳述式同時看見兩個函數。

本範例會使用四個檔案來示範此技巧。所有檔案都必須位於您的類別路徑中。第一個檔案是 myInternal.as，用來定義 myInternal 命名空間。因為檔案是在名為 example 的套件中，所以您必須將檔案放入名為 example 的檔案夾中。該命名空間會標示為 public，以便能匯入其它套件中。

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

第二個和第三個檔案分別是 Utility.as 和 Helper.as，定義包含必須供其它套件使用之方法的類別。Utility 類別是在 example.alpha 套件中，也就是說，檔案應該是在 example 檔案夾下名為 alpha 的子檔案夾中。Helper 類別是在 example.beta 套件中，也就是說，檔案應該是在 example 檔案夾下名為 beta 的子檔案夾之中。example.alpha 和 example.beta 這兩個套件都必須先匯入命名空間，才能加以使用。

```
// Utility.as in the example/alpha folder
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 in the example/beta folder
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;
        }
    }
}
```

第四個檔案 NamespaceUseCase.as 是主要的應用程式類別，必須是 example 檔案夾的同級節點。在 Flash Professional 中，這個類別是用來當做 FLA 的文件類別使用。NamespaceUseCase 類別也會匯入 myInternal 命名空間，並用它來呼叫兩個位於其它套件中的靜態方法。本範例之所以使用靜態方法，只是為了簡化程式碼而已。靜態和實體方法都可以放在 myInternal 命名空間中。

```
// NamespaceUseCase.as
package
{
    import flash.display.MovieClip;
    import example.myInternal; // import namespace
    import example.alpha.Utility;// import Utility class
    import example.beta.Helper;// import Helper class

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

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

            Helper.someTask();
            trace(Helper.lastCalled); // [time someTask() was last called]
        }
    }
}
```

變數

變數可以讓您儲存在程式中使用的值。若要宣告變數，必須使用 var 陳述式加上變數名稱。而在 ActionScript 3.0 中，則需要一律必須使用 var 陳述式。例如，下列 ActionScript 程式碼行會宣告名為 i 的變數：

```
var i;
```

如果宣告變數時省略 var 陳述式，則在嚴謹模式會發生編譯器錯誤，而在標準模式則會發生執行階段錯誤。例如，如果未事先定義 i 變數，下列程式碼會產生錯誤：

```
i; // error if i was not previously defined
```

若要使變數與資料類型產生關聯時，您必須在宣告變數時就這麼做。在未指定變數的類型時宣告變數是合乎規定的作業，但在嚴謹模式中將產生編譯器警告。您可以在變數名稱後面加上冒號 (:) 接著加上變數的類型，以指定變數的類型。例如，下列程式碼會宣告類型為 int 的變數 i：

```
var i:int;
```

您可以使用指定運算子 (=)，將值指定給變數。例如，下列程式碼會宣告變數 i，並將此變數的值指定為 20：

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

您可能會發現，在宣告變數時同時指定值會比較方便，如下列範例所示：

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

在宣告變數時同時指定值的作法，不僅可運用於指定如整數或字串等基本值，也能運用於建立陣列或實體化類別實體。下列範例會示範使用一行程式碼，宣告陣列同時也指定其值。

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

您可以使用 new 運算子，建立類別的實體。下列範例會建立名為 CustomClass 的實體，同時將新建立之類別實體的參考指派給名為 customItem 的變數：

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

如果您要宣告多個變數，可以使用逗號運算子 (,) 分隔變數，在一行程式碼中宣告所有變數。例如，下列程式碼會在一行程式碼中宣告三個變數：

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

您可以在同一行程式碼中為每一個變數指定值。例如，下列程式碼會宣告三個變數 (a、b 和 c)，並為每個變數指定一個值：

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

雖然您可以使用逗號運算子將變數宣告組成一個陳述式，但是這種作法可能會降低程式碼的可讀性。

瞭解變數範圍

變數的 「範圍」是語彙參考可以在其中存取變數的程式碼區域。「全域」變數是定義於程式碼所有區域中的一種變數，而 「區域」變數則是僅定義於程式碼一部分的一種變數。在 ActionScript 3.0 中，指定給變數的範圍僅限於在其中宣告這些變數的函數或類別。全域變數是您在任何函數或類別定義之外定義的變數。例如，下列程式碼會在任何函數之外宣告，建立全域變數 strGlobal。這個範例顯示，全域變數可以同時在函數定義之內和之外使用。

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

您可以在函數定義之內宣告變數，藉以宣告區域變數。可供您定義區域變數的最小程式碼區域是函數定義。在函數之內宣告的區域變數，只存在於該函數之中。例如，如果您在名為 localScope() 的函數中宣告名為 str2 的變數，該變數無法在此函數之外使用。

```
function localScope()
{
    var strLocal:String = "local";
}
localScope();
trace(strLocal); // error because strLocal is not defined globally
```

如果您用於區域變數的變數名稱已宣告為全域變數，當區域變數位於範圍內時，區域定義會隱藏 (或遮蔽) 全域定義。全域變數仍會存在於函數之外。例如，下列程式碼會建立名為 str1 的全域字串變數，然後在 scopeTest() 函數內建立相同名稱的區域變數。函數之內的 trace 陳述式會輸出該變數的區域值，但函數之外的 trace 陳述式則會輸出該變數的全域值。

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

ActionScript 變數與 C++ 和 Java 中的變數不同，不會有區塊層級範圍。程式碼區塊是介於左大括號 ({) 與右大括號 (}) 之間的任何一組陳述式。在像是 C++ 和 Java 這類程式語言中，於程式碼區塊之內宣告的變數無法在該程式碼區塊之外使用。這項範圍限制稱為區塊層級範圍，但在 ActionScript 中並沒有這項限制。如果在程式碼區塊之內宣告變數，則該變數不僅可在該程式碼區塊中使用，也可以在程式碼所屬函數的任何其它部分使用。例如，下列函數包含在各種不同區塊範圍中定義的變數。所有變數都可以透過函數使用。

```
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); // all still defined
    }
    trace(elemStr, valueStr, i); // all defined if numElements > 0
}

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

沒有區塊層級範圍代表一項有趣的含意：您可以在宣告變數之前讀取或寫入，只要在函數結束前宣告即可。這是因為有一個稱為「升舉」的技術，會讓編譯器將所有變數宣告移至函數的最上層。例如，下列程式碼即使在 num 變數的初始 trace() 函數於宣告 num 變數之前發生，也會進行編譯：

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

但是，編譯器並不會升舉任何指定陳述式。這說明了 num 之初始 trace() 產生 NaN (非數字) 的原因，此為 Number 資料類型的變數預設值。這表示即使在宣告變數之前，您也可以指定變數的值，如下列範例所示：

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

預設值

「預設值」是您設定變數值之前，變數所包含的值。首次設定變數值時，就是在對變數進行「初始化」。如果宣告變數但未設定變數值，該變數就是「未初始化」。未初始化的變數值是依其資料類型而定。下表說明變數的預設值，依資料類型組織整理：

資料類型	預設值
Boolean	False
int	0
Number	NaN
物件	null
String	null
uint	0
未宣告 (相當於類型註釋 *)	undefined
所有其它類別，包括使用者定義的類別。	null

若是 Number 類型的變數，預設值為 NaN (非數字)，這是以 IEEE-754 標準定義的特殊值，表示不代表數字的值。

如果您宣告變數，但並未宣告其資料類型，則會套用預設資料類型 *，這其實是表示該變數不具類型。如果也不用值初始化不具類型的變數，其預設值即為 undefined。

若是 Boolean、Number、int 和 uint 以外的資料類型，任何未初始化之變數的預設值都是 null。這適用於所有由 ActionScript 3.0 定義的類別，以及您所建立的任何自訂類別。

null 值並不是類型為 Boolean、Number、int 或 uint 等變數的有效值。若嘗試指定 null 的值給上述變數，該值即轉換為該資料類型的預設值。若是 Object 類型變數，您可以指派 null 的值。若嘗試指定 undefined 的值給 Object 類型變數，該值即轉換為 null。

若是 Number 類型變數，會有名為 isNaN() 的特殊最上層函數，若變數不是數字，則會傳回 Boolean 值 true；否則會傳回 false。

資料類型

「資料類型」會定義一組值。例如，Boolean 資料類型是只有 true 和 false 兩個值的集合。除了 Boolean 資料類型以外，ActionScript 3.0 另外還定義了數個常用的資料類型，例如 String、Number 和 Array。您可以透過使用類別或介面定義自訂的值集合，自行定義資料類型。ActionScript 3.0 中所有的值都是物件，不管是基本值或複雜值都一樣。

「基本值」是屬於下列其中一個資料類型的值：Boolean、int、Number、String 和 uint。使用基本值通常會比使用複雜值快速，因為 ActionScript 是以特殊方式儲存基本值，而能夠讓記憶體和速度最佳化。

備註：讀者若對技術層面的細節感興趣的話，應該會想知道 ActionScript 其實是將基本值以永遠不變的物件儲存於內部。儲存為永遠不變的物件，表示由參考傳遞與由值傳遞的效率都相同。因為參考通常比值本身小很多，所以這麼做便能降低記憶體的用量，同時也加快執行速度。

「複雜值」是非基本值的值。定義複雜值集的資料類型包括 Array、Date、Error、Function、RegExp、XML 和 XMLList。

許多程式語言會區分基本值及其包裝函式物件。例如，Java 有 int 基本值及包裝它的 java.lang.Integer 類別。Java 基本值並非物件，但其包裝函式是物件，使得基本值適用於某些作業，而包裝函式物件則較適合其它作業。在 ActionScript 3.0 中，由於考慮到實際應用目的，因此基本值及其包裝函式物件是不做區分的。所有值都是物件，甚至包括基本值也是。執行階段會將這些基本類型視為行為方式類似物件的特殊狀況，但並不需要一般建立物件所需的額外負荷。這表示下列兩行程式碼是相等的：

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

上面列出的所有基本和複雜資料類型都是以 ActionScript 3.0 核心類別定義的。核心類別可以讓您使用常值建立物件，而不是使用 new 運算子。例如，您可以使用常值或 Array 類別建構函式來建立陣列，如下所示：

```
var someArray:Array = [1, 2, 3]; // literal value
var someArray:Array = new Array(1,2,3); // Array constructor
```

類型檢查

類型檢查可能會在編譯階段或執行階段發生。靜態產生類型的語言 (如 C++ 和 Java) 會在編譯階段進行類型檢查；動態產生類型的語言 (如 Smalltalk 和 Python) 會在執行階段處理類型檢查。ActionScript 3.0 是動態產生類型的語言，具有執行階段類型檢查功能，但也能以特殊的編譯器模式 (稱為 「嚴謹模式」) 支援編譯階段類型檢查功能。在嚴謹模式中，類型檢查會在編譯器階段及執行階段發生；但是在標準模式中，類型檢查只在執行階段中發生。

動態產生類型的語言能在建構程式碼時提供極大的彈性，但是卻必須允許類型錯誤在執行階段顯現。靜態產生類型的語言會在編譯階段報告類型錯誤，但是在編譯階段就必須要知道類型資訊。

編譯階段類型檢查

大型專案經常偏好使用編譯階段類型檢查，因為隨著專案的大小成長，資料類型彈性通常變得比較不重要，盡早發現類型錯誤反而顯得更重要。因此才會在預設時將 Flash Professional 和 Flash Builder 中的 ActionScript 編譯器設定為在嚴謹模式中執行。

Adobe Flash Builder

您可以在 Flash Builder 中，透過 「Project Properties」對話方塊中的 ActionScript 編譯器設定來停用嚴謹模式。

為了提供編譯階段類型檢查，編譯器必須知道程式碼中變數或運算式的資料類型資訊。若要明確地宣告變數的資料類型，請加入冒號運算子 (:)，之後再用資料類型做為變數名稱的字尾。若要使資料類型與參數產生關聯，請使用冒號運算子，之後再加上資料類型。例如，下列程式碼會將資料類型資訊加入 xParam 參數，而用明確的資料類型宣告變數 myParam：

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

在嚴謹模式中，ActionScript 編譯器會將類型不相符回報為編譯器錯誤。例如，下列程式碼會宣告 Object 類型的函數參數 xParam，但稍後會嘗試將 String 和 Number 類型的值指定給該參數。如此會在嚴謹模式中產生編譯器錯誤。

```
function dynamicTest(xParam:Object)
{
    if (xParam is String)
    {
        var myStr:String = xParam; // compiler error in strict mode
        trace("String: " + myStr);
    }
    else if (xParam is Number)
    {
        var myNum:Number = xParam; // compiler error in strict mode
        trace("Number: " + myNum);
    }
}
```

但是，即使在嚴謹模式中，您也可以選擇性地將指定陳述式的右邊保留為不具類型，決定不執行編譯階段類型檢查。您可以省略類型註釋，或使用特殊的星號 (*) 類型註釋，讓變數或運算式成為不具類型。例如，若上一個範例中的 xParam 參數已經過修改，使它不再有類型註釋，則程式碼將在嚴謹模式中進行編譯：

```
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");
```

執行階段類型檢查

無論您在嚴謹模式或標準模式中進行編譯，ActionScript 3.0 中都會發生執行階段類型檢查。請考慮一種情況，在此情況下，3 這個值是傳遞做為預期有陣列的函數之引數。在嚴謹模式中，編譯器將產生錯誤，因為 3 這個值與資料類型 Array 不相容。若停用嚴謹模式，而且在標準模式中執行，則編譯器會回報類型不相符，而執行階段類型檢查則會導致執行階段錯誤。

下列範例會顯示名為 typeTest() 的函數，它預期會收到 Array 引數，但收到的卻是 3 這個值。如此便會在標準模式中導致執行階段錯誤，因為 3 這個值並不是該參數之宣告資料類型 (Array) 的成員。

```
function typeTest(xParam:Array)
{
    trace(xParam);
}
var myNum:Number = 3;
typeTest(myNum);
// run-time error in ActionScript 3.0 standard mode
```

在某些情況下，您可能也會收到執行階段錯誤，甚至在嚴謹模式中操作時也一樣。如果使用嚴謹模式，但使用不具類型的變數，決定不執行編譯階段類型檢查，就可能會發生這種情況。當您使用不具類型的變數時，並不會消除類型檢查，而是將檢查延遲到執行階段再進行。例如，假設上一個範例中的 myNum 變數並未宣告資料類型，編譯器無法偵測類型不相符的情形，但程式碼會產生執行階段錯誤，因為它會比較 myNum 的執行階段值 (由於指定陳述式的結果而設定為 3) 與 xParam 的類型 (設定為 Array 資料類型)。

```
function typeTest(xParam:Array)
{
    trace(xParam);
}
var myNum = 3;
typeTest(myNum);
// run-time error in ActionScript 3.0
```

相較於編譯階段檢查，執行階段類型檢查也允許以更具彈性的方式使用繼承。藉由將類型檢查延遲到執行階段，即使您「向上轉型」，標準模式也可以讓您參考子類別的屬性。使用基底類別宣告類別實體的類型，但是用子類別進行實體化時，就會發生向上轉型。例如，您可以建立可擴充的 ClassBase 類別 (具有 final 特質的類別無法進行擴充)：

```
class ClassBase
{
}
```

接著，您可以建立 ClassBase 的子類別 ClassExtender，它具有一個 someString 屬性，如下所示：

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

您可以使用這兩個類別，建立使用 ClassBase 資料類型宣告的類別實體，但使用 ClassExtender 建構函式進行實體化。向上轉型被視為安全作業，因為基底類別只包含子類別中的屬性或方法。

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

但是子類別確實包含基底類別所不包含的屬性或方法。例如，ClassExtender 類別包含 someString 屬性，而 ClassBase 類別中並沒有此屬性。在 ActionScript 3.0 標準模式中，您可以使用 myClass 實體參考此屬性，而不會產生編譯階段錯誤，如下列範例所示：

```
var myClass:ClassBase = new ClassExtender();
myClass.someString = "hello";
// no error in ActionScript 3.0 standard mode
```

is 運算子

is 運算子可以讓您測試變數或運算式是否為指定資料類型的成員。在舊版 ActionScript 中，是由 instanceof 運算子提供此功能，但在 ActionScript 3.0 中，則不應該使用 instanceof 運算子來測試資料類型的成員資格。進行手動類型檢查時，請勿使用 instanceof 運算子，而應該改用 is 運算子，因為運算式 x instanceof y 只會檢查 x 的原型鏈是否有 y 存在 (而在 ActionScript 3.0 中，原型鏈並不提供完整的繼承階層狀況)。

is 運算子會檢查正確的繼承階層，且不但可用來檢查物件是否為特定類別的實體，並可用來檢查物件是否為實作特定介面的類別實體。下列範例會建立名為 mySprite 的 Sprite 類別實體，並使用 is 運算子測試 mySprite 是否為 Sprite 和 DisplayObject 類別的實體，以及它是否會實作 IEventDispatcher 介面：

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

is 運算子會檢查繼承階層，並正確報告 mySprite 與 Sprite 和 DisplayObject 類別相容 (Sprite 類別是 DisplayObject 類別的子類別)。is 運算子也會檢查 mySprite 是否繼承自任何實作 IEventDispatcher 介面的類別。由於 Sprite 類別是繼承自 EventDispatcher 類別，此類別會實作 IEventDispatcher 介面，所以 is 運算子會正確報告 mySprite 實作相同的介面。

下列範例會示範上個範例的相同測試，但是使用的是 instanceof 而不是 is 運算子。instanceof 運算子會正確地識別 mySprite 為 Sprite 或 DisplayObject 的實體，但在用來測試 mySprite 是否實作 IEventDispatcher 介面時，則會傳回 false。

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

as 運算子

as 運算子也可以讓您檢查運算式是否為指定資料類型的成員。與 is 運算子不同，as 運算子不會傳回 Boolean 值。as 運算子是傳回運算式的值而非 true；傳回 null 而非 false。下列範例會示範不使用 is 而改用 as 運算子，在簡單的範例中檢查 Sprite 實體是否為 DisplayObject、IEventDispatcher 和 Number 資料類型的成員所得出的結果。

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

當您使用 as 運算子時，右邊的運算元必須是資料類型。若嘗試在右邊使用運算式而不用資料類型做為運算元，將會導致錯誤。

動態類別

「動態」類別會定義物件，此物件可以在執行階段，透過加入或變更屬性及方法進行改變。不是動態的類別 (例如 String 類別) 屬於 「密封」類別。您不能在執行階段將屬性或方法加入至密封類別。

您可以在宣告類別時，使用 dynamic 特質，建立動態類別。例如，下列程式碼會建立名為 Protean 的動態類別：

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

若以後實體化 Protean 類別的實體，您可以在類別定義之外，將屬性或方法加入至此實體。例如，下列程式碼會建立 Protean 類別的實體，並將名為 aString 和 aNumber 的屬性加入至此實體：

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

您加入至動態類別實體的屬性是執行階段實體，因此任何類型的檢查都是在執行階段進行。您不能將類型註釋加入至以此方式新增的屬性。

您也可以透過定義函數，然後將函數附加至 myProtean 實體的屬性，將方法加入至 myProtean 實體。下列程式碼會將 trace 陳述式移入名為 traceProtean() 的方法中：

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

但是，以這種方式建立的方法，無法存取 Protean 類別的任何私有屬性或方法。而且，即使參考 Protean 類別的公用屬性或方法，也必須利用 this 關鍵字或類別名稱加以限定。下列範例會示範 traceProtean() 方法嘗試存取 Protean 類別的私有和公用變數。

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

資料類型說明

基本資料類型包括：Boolean、int、Null、Number、String、uint 和 void。ActionScript 核心類別也會定義下列複雜資料類型：Object、Array、Date、Error、Function、RegExp、XML 和 XMLList。

Boolean 資料類型

Boolean 資料類型包含兩個值：true 和 false。Boolean 類型的變數沒有其它有效值。已宣告但未初始化的 Boolean 變數預設值為 false。

int 資料類型

int 資料類型在內部是儲存為 32 位元的整數，而包含下列範圍的一組整數：

-2,147,483,648 (-2^{31}) 至 2,147,483,647 ($2^{31} - 1$) (含)。舊版 ActionScript 僅提供 Number 資料類型，同時供整數與浮點數使用。而在 ActionScript 3.0 中，您可以存取低階電腦類型 32 位元具有正負號和無正負號的整數。如果變數不需要浮點數，不使用 Number 資料類型，而改用 int 資料類型應該會更快、更有效率。

對於在最小及最大 int 值範圍之外的整數值來說，使用 Number 資料類型可以處理介於正負 9,007,199,254,740,992 (53 位元整數值) 之間的值。int 資料類型變數的預設值為 0。

Null 資料類型

Null 資料類型僅包含 null 這個值。這是 String 資料類型及定義複雜資料類型之所有類別 (包括 Object 類別) 的預設值。其它任何基本資料類型，例如 Boolean、Number、int 和 uint，都不包含 null 這個值。在執行階段中，如果您嘗試將 null 指派給類型為 Boolean、Number、int 或 uint 的變數，null 這個值會轉換成適當的預設值。您無法使用此資料類型做為類型註釋。

Number 資料類型

在 ActionScript 3.0 中，Number 資料類型可以代表整數、無正負號的整數和浮點數值。但是若要得到最高效能，應該只在整數值大於 32 位元 int 和 uint 類型所能儲存的數值時，或數值為浮點數時，才使用 Number 資料類型。若要儲存浮點數值，必須在數值中包含小數點。如果省略小數點，數值會儲存為整數。

Number 資料類型使用的是 IEEE 二進位浮點數運算標準 (IEEE-754) 所指定的 64 位元雙精度格式。這項標準規定使用 64 個可用位元儲存浮點數值的方式。其中，1 個位元是用來指定數值是正值或負值；11 個位元可供指數使用，儲存時以 2 為底；而其餘 52 個位元則是用來儲存「有效位數」(也稱為「尾數」)，它是自乘至指數所指示的次方值。

Number 資料類型透過使用部分位元儲存指數，比使用其所有位元儲存有效位數可儲存的浮點數值大得多。例如，若 Number 資料類型將 64 個位元全部用來儲存有效位數，那麼可儲存的最大數目為 $2^{65} - 1$。若使用 11 位元來儲存指數，則 Number 資料類型可將有效位數增加為 2^{1023}。

Number 類型可代表的最大及最小值是儲存在 Number 類別的靜態屬性 Number.MAX_VALUE 和 Number.MIN_VALUE 之中。

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

雖然這個數值的範圍非常之大，但是精確度相較之下則差了許多。Number 資料類型會使用 52 個位元儲存有效位數，其結果是需要 52 個位元以上才能精確地代表的數值 (例如，分數 1/3) 只是近似值而已。如果應用程式需要絕對精確度的小數，就必須使用實作小數浮點運算的軟體，而不能使用二進位浮點數運算。

以 Number 資料類型儲存整數值時，僅會用到有效位數的 52 個位元。Number 資料類型會使用這 52 個位元及特殊的隱藏位元代表自 -9,007,199,254,740,992 (-2^{53}) 至 9,007,199,254,740,992 (2^{53}) 的整數值。

NaN 值不只是做為 Number 類型變數的預設值而已，還可以做為任何應傳回但未傳回數值之運算作業的結果。例如，若嘗試計算負數的平方根，則結果會是 NaN。其它特殊 Number 值包括「正無限大」和「負無限大」。

備註：若除數也是 0，則除以 0 的結果只有 NaN。當被除數是正數時，除以 0 會產生 infinity；而當被除數是負數時，則產生 -infinity。

String 資料類型

String 資料類型代表 16 位元字元的序列。字串會使用 UTF-16 格式，在內部儲存為 Unicode 字元。字串是永遠不變的值，與在 Java 程式語言中的性質相同。String 值的操作會傳回字串的新實體。使用 String 資料類型宣告的變數預設值為 null。null 這個值和空字串 ("") 不一樣。null 這個值表示變數未儲存任何值，而空字串表示變數有一個值，也就是一個不包含任何字元的「字串」。

uint 資料類型

uint 資料類型在內部是儲存為 32 位元無正負號整數，而包含自 0 至 4,294,967,295 ($2^{32} - 1$) 的一組整數 (包含 0 和 4,294,967,295)。在呼叫非負數整數的特殊狀況下，才會使用 uint 資料類型。例如，您必須使用 uint 資料類型代表像素顏色值，因為 int 資料類型所具有的內部正負號位元並不適合處理顏色值。對於比最大 uint 值還要大的整數值來說，使用 Number 資料類型可以處理 53 位元整數值。uint 資料類型變數的預設值為 0。

void 資料類型

void 資料類型只包含 undefined 這個值。在舊版 ActionScript 中，undefined 是 Object 類別之實體的預設值；在 ActionScript 3.0 中，Object 實體的預設值為 null。若嘗試指定 undefined 的值給 Object 類別實體，該值會轉換為 null。您只能將 undefined 值指定給不具類型的變數。不具類型變數是沒有任何類型註釋，或使用星號 (*) 類型註釋的變數。您只能使用 void 做為傳回類型註釋。

Object 資料類型

Object 資料類型由 Object 類別所定義。Object 類別可做為 ActionScript 中所有類別定義的基底類別。與舊版 ActionScript 相較之下，ActionScript 3.0 版 Object 資料類型的不同之處包含以下三方面：第一，Object 資料類型不再是指定給沒有類型註釋之變數的預設資料類型；第二，Object 資料類型不再包含 undefined 值，過去這是 Object 實體的預設值；第三，在 ActionScript 3.0 中，Object 類別之實體的預設值是 null。

在舊版 ActionScript 中，沒有類型註釋的變數會自動指定為 Object 資料類型；而在 ActionScript 3.0 中已非如此，現在 ActionScript 3.0 包含真正不具類型變數的概念。沒有類型註釋的變數現在會視為不具類型。若要讓程式碼的讀者清清楚楚地知道，您是刻意要讓變數保持為不具類型，則可以使用類型註釋的星號 (*) 符號，這個符號等於省略類型註釋。下列範例會示範兩個相等的陳述式，兩個都會宣告不具類型的變數 x：

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

只有不具類型的變數可以存放 undefined 值。若嘗試將 undefined 值指定給具有資料類型的變數，執行階段會將 undefined 值轉換為該資料類型的預設值。例如 Object 資料類型 (預設值為 null)，表示如果您嘗試將 undefined 指派為 Object 實體，則值會轉換成 null。

類型轉換

類型轉換可以說是在值轉換成不同資料類型的值時發生。類型轉換可以是 「隱含」或 「明確」。隱含轉換，又稱為 「強制」，有時候會在執行階段中執行。例如，如果將 2 這個值指定給資料類型為 Boolean 的變數，就會先將 2 這個值轉換成 Boolean 值 true，再將此值指定給變數。明確轉換也稱為 「轉型」，是在您的程式碼指示編譯器將變數視為一種資料類型，而似乎是屬於不同的資料類型時發生。當涉及基本值時，轉型其實是從一種資料類型轉換成另一種資料類型。若要將物件轉換為另一種類型，您可以用括號括住物件名稱，並在它前面放置新類型的名稱。例如，下列程式碼會使用 Boolean 值進行轉型，將它轉換為整數：

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

隱含轉換

隱含轉換在執行階段時發生於幾種不同狀況下：

- 在指定陳述式中
- 當值做為函數引數傳遞時
- 當值自函數傳回時
- 在使用某些運算子 (如，加法 (+) 運算子) 的運算式中

  至於使用者定義的類型，在要轉換的值為目標類別之實體時，或衍生自目標類別的類別時，會順利進行隱含轉換。若隱含轉換不成功，就會發生錯誤。例如，下列程式碼包含成功的隱含轉換和不成功的隱含轉換：

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

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

objA = objB; // Conversion succeeds.
objB = arr; // Conversion fails.
```

對於基本類型，隱含轉換是透過呼叫與明確轉換函數所呼叫相同的內部轉換演算法來處理。

明確轉換

在嚴謹模式中進行編譯時，使用明確轉換或轉型會很有幫助，因為有時您不希望因為類型不相符而產生編譯階段錯誤。當您知道強制轉換將在執行階段正確地轉換值時，可能就是這種情況。例如，使用自表單接收的資料時，可能要靠強制轉換將某些字串值轉換成數值。即使程式碼在標準模式中正確執行，下列程式碼也會產生編譯階段錯誤：

```
var quantityField:String = "3";
var quantity:int = quantityField; // compile time error in strict mode
```

若要繼續使用嚴謹模式，但想要將字串轉換成整數，您可以使用明確轉換，如下所示：

```
var quantityField:String = "3";
var quantity:int = int(quantityField); // Explicit conversion succeeds.
```

轉型成 **int**、**uint** 和 **Number**

您可以將任何資料類型轉型成為下列三個數值類型之一：int、uint 和 Number。如果因為某些原則造成無法轉換數字，則會為 int 與 uint 資料類型指定預設值 0，以及為 Number 資料類型指定預設值 NaN。如果將 Boolean 值轉換成數值，true 會成為 1 這個值；而 false 會成為 0 這個值。

```
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
```

僅包含數字的字串值可以成功地轉換成為其中一種數值類型。數值類型也可以轉換成看起來像負數值的字串，或代表十六進位值的字串 (例如，0x1A)。轉換程序會忽略字串值中的開頭或結尾空白字元。您也可以使用 Number()，對看起來像浮點數值的字串進行轉型。如果包含小數點，uint() 和 int() 會傳回整數，並將小數點和小數點之後的字元截斷。例如，下列字串值可以轉型成為數值：

```
trace(uint("5")); // 5
trace(uint("-5")); // 4294967291. It wraps around from 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
```

包含非數值字元的字串值會在以 int() 或 uint() 轉型時傳回 0；而在以 Number() 轉型時傳回 NaN。轉換程序會忽略開頭或結尾空白字元，但如果字串有空白分隔兩個數值時，則會傳回 0 或 NaN。

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

在 ActionScript 3.0 中，Number() 函數不再支援八進位或以 8 為基底的數字。若提供有開頭零的字串給 ActionScript 2.0 Number() 函數，則數字會截斷為八進位數字，並轉換成相等的十進位數。但是 Number() 函數在 ActionScript 3.0 中則不再是如此，而是會忽略開頭的零。例如，下列程式碼在使用不同版本的 ActionScript 編譯時，會產生不同的輸出：

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

將其中一個數值類型的值指定給不同數值類型的變數時，並不需要進行轉型作業。即使是在嚴謹模式中，數值類型也是以隱含方式轉換成其它數值類型。也就是說，在某些情況下，超過類型範圍時，可能會產生未預期的值。下列範例都會在嚴謹模式中編譯，但是有一些會產生未預期的值：

```
var myUInt:uint = -3; // Assign int/Number value to uint variable
trace(myUInt); // 4294967293

var myNum:Number = sampleUINT; // Assign int/uint value to Number variable
trace(myNum) // 4294967293

var myInt:int = uint.MAX_VALUE + 1; // Assign Number value to uint variable
trace(myInt); // 0

myInt = int.MAX_VALUE + 1; // Assign uint/Number value to int variable
trace(myInt); // -2147483648
```

下表摘要列出從其它資料類型轉型成 Number、int 或 uint 資料類型的結果。

資料類型或值	轉換成 **Number**、**int** 或 **uint** 的結果
Boolean	若值為 true，則為 1；否則為 0。
Date	Date 物件的內部形式，從全球時間 1970 年 1 月 1 日午夜開始計算的毫秒數。
null	0
物件	若實體為 null 並且轉換成 Number，則為 NaN；否則為 0。
String	如果字串可以轉換成數字，則為一個數字；如果轉換成 Number，則為 NaN，或者如果轉換成 int 或 uint，則為 0。
undefined	若轉換成 Number，則為 NaN；若轉換成 int 或 uint，則為 0。

轉型成為 **Boolean**

從任何數值資料類型 (uint、int 和 Number) 轉型成為 Boolean，若數值為 0，則結果為 false，否則為 true。若是 Number 資料類型，則 NaN 這個值也會產生 false 的結果。下列範例會示範數字 -1、0 和 1 的轉型結果：

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

來自範例的輸出會示範三個數字的轉型結果，只有 0 會傳回 false 的值：

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

若字串是 null 或空字串 ("")，則從 String 值轉型成 Boolean 會傳回 false；否則，它會傳回 true。

```
var str1:String; // Uninitialized string is null.
trace(Boolean(str1)); // false

var str2:String = ""; // empty string
trace(Boolean(str2)); // false

var str3:String = " "; // white space only
trace(Boolean(str3)); // true
```

從 Object 類別實體轉型成為 Boolean，若實體為 null，則傳回 false，否則傳回 true：

```
var myObj:Object; // Uninitialized object is null.
trace(Boolean(myObj)); // false

myObj = new Object(); // instantiate
trace(Boolean(myObj)); // true
```

Boolean 變數在嚴謹模式中會有特殊處理，您可以在其中指定任何資料類型的值給 Boolean 變數而無需轉型。即使在嚴謹模式中，也會從所有資料類型隱含強制轉換成 Boolean 資料類型。換句話說，與幾乎所有其它資料類型都不同的是，不必轉型成 Boolean 就可避免發生嚴謹模式錯誤。下列範例都會在嚴謹模式中編譯，而且在執行階段的行為方式將如預期：

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

下表摘要列出從其它資料類型轉型成 Boolean 資料類型的結果：

資料類型或值	轉換成 **Boolean** 的結果
String	若值為 null 或空字串 ("")，則為 false；否則為 true。
null	False
Number、int 或 uint	若值為 NaN 或 0，則為 false；否則為 true。
物件	若實體為 null，則為 false；否則為 true。

轉型成 **String**

從任何數值資料類型轉型成 String 資料類型，都會傳回數值的字串形式。從 Boolean 值轉型成 String 資料類型，若值為 true 會傳回字串 "true"；若值為 false，則傳回字串 "false"。

從 Object 類別的實體轉型成 String 資料類型，若實體為 null，會傳回字串 "null"。否則，從 Object 類別的實體轉型成 String 資料類型，會傳回字串 "[object Object]"。

從 Array 類別的實體轉型成 String 時，會傳回包含所有陣列元素之逗號分隔清單的字串。例如，下列轉型成 String 資料類型會傳回一個包含陣列之全部三個元素的字串：

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

從 Date 類別的實體轉型成 String 資料類型，會傳回該實體包含的日期字串形式。例如，下列範例會傳回 Date 類別實體的字串形式 (輸出會顯示太平洋日光節約時區的結果)：

```
var myDate:Date = new Date(2005,6,1);
trace(String(myDate)); // Fri Jul 1 00:00:00 GMT-0700 2005
```

下表摘要列出從其它資料類型轉型成 String 資料類型的結果。

資料類型或值	轉換成字串的結果
Array	由所有陣列元素組成的字串。
Boolean	"true" 或 "false"
Date	Date 物件的字串形式。
null	"null"
Number、int 或 uint	數字的字串形式。
物件	若實體為 null，則為 "null"；否則為 "[object Object]"。

語法

語言的語法會定義一組規則，在撰寫可執行的程式碼時您必須加以遵守。

區分大小寫

ActionScript 3.0 是區分大小寫的語言。僅有大小寫差異的識別名稱會視為不同的識別名稱。例如，下列程式碼會建立兩個不同的變數：

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

點語法

點運算子 (.) 可用來存取物件的屬性和方法。使用點語法時，您可以使用實體名稱參考類別屬性或方法，作法是在實體名稱後面接一個點 (.) 運算子，再加上屬性或方法的名稱。例如，請考慮下列類別定義：

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

使用點語法時，您可以使用下列程式碼建立的實體名稱，存取 prop1 屬性和 method1() 方法。

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

定義套件時，您可以使用點語法。您可以使用點運算子來參考巢狀套件。例如，EventDispatcher 類別位於名為 events 的套件中，而該套件又以巢狀方式位於名為 flash 的套件中。您可以使用下列運算式，參考 events 套件：

```
flash.events
```

您也可以使用此運算式，參考 EventDispatcher 類別：

```
flash.events.EventDispatcher
```

斜線語法

ActionScript 3.0 不支援斜線語法。舊版的 ActionScript 會使用斜線語法，表示影片片段或變數的路徑。

常值

「常值」是直接在程式碼中出現的值。下列範例全部都是常值：

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

您可以將多個常值組成複合常值。陣列常值要用方括號字元 ([]) 括住，並使用逗號來分隔陣列元素。

陣列常值可以用來初始化陣列。下列範例顯示兩個使用陣列常值來初始化的陣列。您可以使用 new 陳述式，並傳遞複合常值做為 Array 類別建構函式的參數，但是，您也可以在實體化下列 ActionScript 核心類別的實體時，直接指定常值：Object、Array、String、Number、int、uint、XML、XMLList 和 Boolean。

```
// Use new statement.
var myStrings:Array = new Array(["alpha", "beta", "gamma"]);
var myNums:Array = new Array([1,2,3,5,8]);

// Assign literal directly.
var myStrings:Array = ["alpha", "beta", "gamma"];
var myNums:Array = [1,2,3,5,8];
```

常值也可以用來初始化一般物件。一般物件是 Object 類別的實體。物件常值要用大括號 ({}) 括住，並使用逗號來分隔物件屬性。每個屬性都是使用冒號字元 (:) 加以宣告，冒號會分隔屬性名稱與屬性值。

您可以使用 new 陳述式來建立一般物件，然後將物件常值當做參數傳遞給 Object 類別建構函式；或者，也可以將物件常值直接指定給您所宣告的實體。下列範例示範建立新的一般物件，並將具有三個屬性 (propA、propB 和 propC) 的物件初始化，再分別將其值設定為 1、2 和 3 的兩個替代方式。

```
// Use new statement and add properties.
var myObject:Object = new Object();
myObject.propA = 1;
myObject.propB = 2;
myObject.propC = 3;

// Assign literal directly.
var myObject:Object = {propA:1, propB:2, propC:3};
```

更多說明主題

使用字串

使用規則運算式

初始化 XML 變數

分號

您可以使用分號字元 (;) 終止陳述式。或者，您也可以省略分號字元，編譯器會假設每一行程式碼代表一個陳述式。由於許多程式設計人員習慣使用分號來代表陳述式的結束，因此如果一致地使用分號來終止陳述式，您的程式碼可能會更易於閱讀。

使用分號來終止陳述式可讓您在同一行放置一個以上的陳述式，但這樣做可能會讓程式碼難以閱讀。

括號

在 ActionScript 3.0 中，括號 (()) 有三種使用方式。第一，您可以使用括號來變更運算式中的作業順序。組合在括號之中的作業永遠都會先執行。例如，在下列程式碼中，會使用括號來改變作業的順序：

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

第二，您可以搭配括號使用逗號 (,) 運算子，以評估一系列運算式，並傳回最終運算式的結果，如下列範例所示：

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

第三，您可以使用括號來傳遞一個或多個參數給函數或方法，如下列範例所示，它會傳遞 String 值給 trace() 函數：

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

註解

ActionScript 3.0 程式碼支援兩種類型的註解：單行註解和多行註解。這些註解機制類似於 C++ 和 Java 中的註解機制。編譯器會忽略標示為註解的文字。

單行註解是以兩個正斜線字元 (//) 開頭，然後繼續直到該行結束。例如，下列程式碼包含單行註解：

```
var someNumber:Number = 3; // a single line comment
```

多行註解是以一個正斜線字元和星號 (/*) 開頭，然後以星號和正斜線 (*/) 結尾。

```
/* This is multiline comment that can span
more than one line of code. */
```

關鍵字與保留字

「保留字」是不可以在程式碼中用來當做識別名稱的字詞，因為這些字詞是保留給 ActionScript 使用的。保留字包括 「語彙關鍵字」，是由編譯器從程式命名空間中移除。若使用語彙關鍵字做為識別名稱，則編譯器會報告錯誤。下表列出 ActionScript 3.0 語彙關鍵字。

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			

有一小組關鍵字稱為 「語法關鍵字」，可用來做為識別名稱，但在某些內容中會有特殊的意義。下表列出 ActionScript 3.0 語法關鍵字。

each	get	set	namespace
include	dynamic	final	native
override	static		

也有一些識別名稱有時稱為「未來保留字」。雖然與 ActionScript 3.0 結合的軟體可能將其中部分識別名稱視為關鍵字，但是，ActionScript 3.0 並未保留這些識別名稱。雖然您可以在程式碼中任意使用這些識別名稱，不過 Adobe 建議您不要使用這些識別名稱，因為在後續的語言版本中，它們可能是關鍵字。

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

常數

ActionScript 3.0 支援 const 陳述式，讓您可以用來建立常數。常數是有無法改變之固定值的屬性。您只能將值指定給常數一次，指定作業必須在非常接近常數宣告時發生。例如，如果常數是宣告為類別的成員，您只能將值指定給該常數做為宣告的一部分，或是在類別建構函式中指定值。

下列程式碼會宣告兩個常數：第一個常數是 MINIMUM，具有指定做為宣告陳述式之一部分的值；第二個常數是 MAXIMUM，具有在建構函式中指定的值。請注意，此範例僅能在標準模式下編譯，因為嚴謹模式僅允許在初始化階段指定常數的值。

```
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
```

若嘗試以其它方式將初始值指定給常數，就會發生錯誤。例如，若嘗試在類別之外設定 MAXIMUM 的初始值，就會發生執行階段錯誤。

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

var a:A = new A();
a["MAXIMUM"] = 10; // run-time error
```

ActionScript 3.0 會定義廣泛的常數範圍供您使用。按照慣例，ActionScript 中的常數全都是大寫字母，並以底線字元 (_) 來分隔單字。例如，MouseEvent 類別定義會使用這種命名慣例為其常數命名，每個都代表與滑鼠輸入相關的事件：

```
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";
    ...
    }
}
```

運算子

運算子是特殊的函數，可以有一個或多個運算元，並傳回值。「運算元」是供運算子用來做為輸入的值，通常是常值、變數或運算式。例如，在下列程式碼中，加法 (+) 和乘法 (*) 運算子是搭配三個常值運算元 (2、3, 和 4) 以傳回值。然後此值再由指定 (=) 運算子用來指定傳回的值 14 給變數 sumNumber。

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

運算子可以是一元、二元或三元。「一元」運算子可接受一個運算元。例如，遞增 (++) 運算子就是一元運算子，因為它只接受一個運算元。「二元」運算子可接受兩個運算元。例如，除法 (/) 運算子就接受兩個運算元。「三元」運算子可接受三個運算元。例如，條件 (?:) 運算子就接受三個運算元。

有些運算子是 「多載」，也就是說，它們的行為方式會依傳遞給它們的運算元類型或數量而不同。加法 (+) 運算子就是多載運算子的範例，會依運算元的資料類型而有不同的行為方式。如果兩個運算元都是數值，加法運算子會傳回值的總和。如果兩個運算元都是字串，加法運算子會將兩個運算元連接在一起傳回。下列範例程式碼將示範運算子如何依運算元而有不同的行為方式：

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

運算子也可以根據所提供運算元的數目而有不同的行為方式。減法 (-) 運算子可同時做為一元和二元運算子。只提供一個運算元時，減法運算子會將運算元變為負數，然後傳回結果；當提供兩個運算元時，減法運算子會傳回兩個運算元之間的差異。下列範例會示範，先使用減法運算子做為一元運算子，然後做為二元運算子。

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

運算子優先順序和關聯性

運算子優先順序和關聯性決定處理運算子時所依照的順序。對熟悉算術的人而言，編譯器會在加法 (+) 運算子之前優先處理乘法 (*) 運算子是理所當然的，但編譯器仍需要優先處理哪些運算子的明確指示。這類指示統稱為 「運算子優先順序」。ActionScript 定義了預設的運算子優先順序，您可使用括號 (()) 運算子來變更順序。例如，下列程式碼會改變上一個範例中的預設優先順序，以強迫編譯器先處理加法運算子，再處理乘法運算子：

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

您可能會遇到兩個或更多具有相同優先順序的運算子出現在同一運算式的情形。在這種情況下，編譯器會使用 「關聯性」的規則來決定優先處理的運算子。除了指定運算子以外，所有的二元運算子都是 「左關聯」，這表示會在右邊運算子之前優先處理左邊的運算子。指定運算子和條件 (?:) 運算子是 「右關聯」，這表示會在左邊運算子之前優先處理右邊的運算子。

以小於 (<) 和大於 (>) 運算子為例，它們有著相同的優先順序。如果兩個運算子都使用於相同的運算式，則會先處理左邊的運算子，因為兩個運算子都屬於左關聯性。這表示下列兩個陳述式會產生相同的輸出：

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

大於運算子會先行處理而產生 true 的值，因為運算元 3 大於運算元 2；然後 true 值會傳遞給小於運算子，同時附上運算元 1。下列程式碼代表這種中間狀態：

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

小於運算子會將 true 值轉換為數值 1，再將該數值與第二個運算元 1 做比較，最後傳回 false (值 不小於 1)。

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

您可以用括號運算子，改變預設的左關聯性。將運算子及其運算元括在括號之間，便可以指示編譯器先處理小於運算子。下列範例會使用前一個範例中相同的成員，並使用括號運算子產生不同的輸出：

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

小於運算子會優行處理，其會產生 false 的值，因為運算元 2 不小於運算元 1。false 值接著將連同運算元 3 一併傳遞給大於運算子。下列程式碼代表這種中間狀態：

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

大於運算子會將 false 值轉換為數值 0，再將該數值與另一個運算元 3 做比較，以傳回 true (值 3 大於 0)。

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

下表列出可在 ActionScript 3.0 中用來遞減優先順序的運算子。表中的每一行都包含具有相同優先順序的運算子。每一行運算子的優先順序都高於表中位於其下方的運算子。

群組	運算子
主要	[] {x:y} () f(x) new x.y x[y] <></> @ :: ..
後置	x++ x--
一元	++x --x + - ~ ! delete typeof void
乘法	* / %
加法	+ -
位元位移	<< >> >>>
關係	< > <= >= as in instanceof is
相等	== != === !==
位元 AND	&
位元 XOR	^
位元 OR	|
邏輯 AND	&&
邏輯 OR	||
條件	?:
指定	= *= /= %= += -= <<= >>= >>>= &= ^= |=
逗號	,

主要運算子

主要運算子包括用來建立 Array 和 Object 常值、群組運算式、呼叫函數、實體化類別實體，以及存取屬性的運算子。

如下表中列出的所有主要運算子都有相等的優先順序。屬於 E4X 規格一部分的運算子是以 (E4X) 標記法表示。

運算子	執行的運算
[]	初始化陣列
{x:y}	初始化物件
()	群組運算式
f(x)	呼叫函數
new	呼叫建構函式
x.y x[y]	存取屬性
<></>	初始化 XMLList 物件 (E4X)
@	存取特質 (E4X)
::	限定名稱 (E4X)
..	存取後代 XML 元素 (E4X)

後置運算子

後置運算子可接受一個運算子，並且會遞增或遞減值。雖然這些運算子是一元運算子，但具有較高的優先順序和特殊的行為，因此不同於其它一元運算子而自成一類。使用後置運算子做為較大運算式的一部分時，會先傳回運算式的值，然後再處理後置運算子。例如，下列程式碼顯示運算式 xNum++ 的值會在值遞增之前先傳回：

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

如下表中列出的所有後置運算子都有相等的優先順序：

運算子	執行的運算
++	遞增 (後置)
--	遞減 (後置)

一元運算子

一元運算子可接受一個運算元。這個群組中的遞增 (++) 和遞減 (--) 運算子是「前置運算子」，也就是說，在運算式中，它們會出現於運算元之前。前置運算子與後置運算子不同，因為其遞增或遞減運算會在整個運算式的值傳回之前完成。例如，下列程式碼會示範如何先遞增值，再傳回運算式 ++xNum 的值：

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

如下表中列出的所有一元運算子都有相等的優先順序：

運算子	執行的運算
++	遞增 (前置)
--	遞減 (前置)
+	一元 +
-	一元 - (負操作)

運算子	執行的運算
!	邏輯 NOT
~	位元 NOT
delete	刪除屬性
typeof	傳回類型資訊
void	傳回未定義的值

乘法運算子

乘法運算子可接受兩個運算元，並執行乘法、除法或模除計算。

如下表中列出所有的乘法運算子都有相等的優先順序：

運算子	執行的運算
*	乘法
/	除法
%	模除

加法運算子

加法運算子接受兩個運算元，並執行加法或減法計算。如下表中列出的所有加法運算子都有相等的優先順序：

運算子	執行的運算
+	加法
-	減法

位元位移運算子

位元位移運算子接受兩個運算元，並根據第二個運算元指定的位移數，對第一個運算元的位元進行對應的位移。如下表中列出的所有位元位移運算子都有相等的優先順序：

運算子	執行的運算
<<	位元左移
>>	位元右移
>>>	無正負號的位元右移

關係運算子

關係運算子接受兩個運算元，再比較其值，然後傳回 Boolean 值。如下表中列出的所有關係運算子都有相等的優先順序：

運算子	執行的運算
<	小於
>	大於
<=	小於或等於
>=	大於或等於
as	檢查資料類型
in	檢查物件屬性
instanceof	檢查原型鏈
is	檢查資料類型

相等運算子

相等運算子接受兩個運算元，再比較其值，然後傳回 Boolean 值。如下表中列出的所有相等運算子都有相等的優先順序：

運算子	執行的運算
==	相等
!=	不相等
===	嚴謹相等
!==	嚴謹不相等

位元邏輯運算子

位元邏輯運算子接受兩個運算元，並執行位元層級的邏輯運算。位元邏輯運算子在優先順序上有差異，下表即依遞減的優先順序列出這些運算子：

運算子	執行的運算
&	位元 AND
^	位元 XOR
|	位元 OR

邏輯運算子

邏輯運算子接受兩個運算元，並傳回 Boolean 結果。邏輯運算子在優先順序上有差異，下表即依遞減的優先順序列出這些運算子：

運算子	執行的運算
&&	邏輯 AND
||	邏輯 OR

條件運算子

條件運算子是三元運算子，這表示它會接受三個運算元。條件運算子是套用 if.. 的速記方法 else 條件陳述式的速記方法。

運算子	執行的運算
?:	條件

指定運算子

指定運算子接受兩個運算元，並根據其它運算元的值，將值指定給一個運算元。如下表中列出的所有指定運算子都有相等的優先順序：

運算子	執行的運算
=	指定
*=	乘法指定
/=	除法指定
%=	模除指定
+=	加法指定
-=	減法指定
<<=	位元左移指定
>>=	位元右移指定
>>>=	無正負號的位元右移指定
&=	位元 AND 指定
^=	位元 XOR 指定
|=	位元 OR 指定

條件

ActionScript 3.0 提供三個基本的條件陳述式，可讓您用來控制程式流程。

if..else

if..else 條件陳述式可讓您測試條件，並在該條件成立時執行程式碼區塊，或是在條件不成立時，執行另一個程式碼區塊。例如，下列程式碼會測試 x 的值是否超過 20。如果超過就產生一個 trace() 函數；否則就產生另外一個 trace() 函數：

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

如果不要執行另一個程式碼區塊，則可以只用 if 陳述式，而不使用 else 陳述式。

if..else if

您可以使用 if..else if 條件陳述式來測試一個以上的條件。例如，下列程式碼不僅會測試 x 的值是否超過 20，也會測試 x 的值是否為負數：

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

如果 if 或 else 陳述式後面只跟著一個陳述式，該陳述式便不需要用大括號括住。例如，下列程式碼就不使用大括號：

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

但是，Adobe 建議您永遠都使用大括號，因為如果後來陳述式加入至沒有大括號的條件陳述式，可能會產生未預期的行為。例如，在下列程式碼中，不管條件是否評估為 true，positiveNums 的值都會加 1：

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

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

trace(positiveNums); // 1
```

switch

如果您有多個執行路徑均相依於相同的條件運算式，switch 陳述式就很有用。它提供的功能類似於一段很長的 if..else if 陳述式，但比較容易閱讀。switch 陳述式不會測試 Boolean 值的條件，而是會評估運算式，然後使用結果決定要執行的程式碼區塊。以 case 陳述式開頭，而以 break 陳述式結尾的程式碼區塊。例如，下列 switch 陳述式會根據 Date.getDay() 方法傳回的天數來列印一週當中的日子：

```
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;
}
```

迴圈

迴圈陳述式可以讓您使用一串值或變數，重複執行特定的程式碼區塊。Adobe 建議您永遠使用大括號 ({}) 括住程式碼區塊。雖然程式碼區塊僅包含一個陳述式時，您可以省略大括號，但是不建議採用這種作法，理由和不建議條件陳述式省去大括號相同：將來若加入其它陳述式，會提高它意外從程式碼區塊排除的可能風險。若將來在程式碼區塊中加入想要包含的陳述式，但忘了加上必要的大括號，則該陳述式就不會成為迴圈的一部分執行。

for

for 迴圈可以讓您針對特定數值範圍的變數重複執行。您必須在 for 陳述式中提供三個運算式：設定為初始值的變數、決定迴圈何時結束的條件陳述式，以及隨每次迴圈變更變數值的運算式。例如，下列程式碼會重複迴圈五次。變數 i 的值開始於 0 而結束於 4，而輸出則是從 0 到 4，每個值都各自佔有一行。

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

for..in

for..in 迴圈會重複執行物件的屬性或陣列的元素。例如，您可以使用 for..in 迴圈可重複執行一般物件的屬性 (物件屬性並沒有任何特定順序，因此屬性可能會以隨機順序顯示)：

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

您也可以重複執行陣列的元素：

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

如果物件是密封類別的實體 (包括內建的類別和使用者定義的類別)，您將無法重複執行其屬性。您只能重複動態類別的屬性。即使有動態類別的實體，您也只能重複動態新增的屬性。

for each..in

for each..in 迴圈會重複執行集合中的項目，這些項目可能是 XML 或 XMLList 物件中的標籤、物件屬性保留的值，或陣列的元素。例如，如下列摘錄所示，您可以使用 for each..in 迴圈重複執行一般物件的屬性，但與 for..in 迴圈不同的是，for each..in 迴圈中的迴圈指標變數包含由屬性保存的值，而不是屬性名稱：

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

您可以重複執行 XML 或 XMLList 物件，如下列範例所示：

```
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
*/
```

您也可以重複執行陣列的元素，如下列範例所示：

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

如果物件是密封類別的實體，您將無法重複執行其屬性。即使是動態類別的實體，您也無法重複執行任何固定的屬性，這些都是定義為類別定義之一部分的屬性。

while

while 迴圈與 if 陳述式相同，只要條件為 true，就會不斷重複。例如，下列程式碼會產生與 for 迴圈範例相同的輸出：

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

使用 while 迴圈而不使用 for 迴圈的一項缺點，是利用 while 迴圈編寫無限迴圈較為容易。如果省略遞增計數器變數的運算式，for 迴圈範例程式碼就不會編譯，但若省略該步驟，while 迴圈範例卻能進行編譯。如果沒有遞增 i 的運算式，該迴圈就會變成無限迴圈。

do..while

do..while 迴圈是保證程式碼區塊至少執行一次的 while 迴圈，因為其條件是在程式碼區塊執行之後才檢查。下列程式碼會示範 do..while 迴圈的簡單範例，此範例即使在條件不符的情況下也會產生輸出：

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

函數

「函數」是可以執行特定工作，且可在程式中重複使用的程式碼區塊。ActionScript 3.0 中有兩種類型的函數：「方法」和「函數結束項」。依函數定義的內容，便可決定函數該稱為方法或函數結束項。若函數是定義為類別定義的一部分，或附加至物件的實體，則稱為方法；若函數是以任何其它方式定義，則稱為函數結束項。

函數在 ActionScript 中一直都相當重要。例如，ActionScript 1.0 中沒有 class 關鍵字，所以「類別」就由建構函數加以定義。雖然已將 class 關鍵字加入此語言，但若要完全發揮此語言所能提供的功能，徹底瞭解函數仍然很重要。對於預期 ActionScript 的函數行為會與 C++ 或 Java 這類語言中的函數行為相似的程式設計人員來說，這可能會是一項挑戰。雖然基本的函數定義和叫用過程對有經驗的程式設計人員不會有什麼困難，但是 ActionScript 函數中有些比較進階的功能特性可能就需要多加解釋。

基本的函數觀念

呼叫函數

呼叫函數時，必須使用函數的識別名稱，後面跟著括號運算子 (())。括號運算子必須括住要傳送給該函數的任何函數參數。例如，trace() 函數是 ActionScript 3.0 中最上層的函數：

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

若要在不使用參數的情況下呼叫函數，必須使用一對空的括號。舉例來說，您可以使用 Math.random() 方法產生隨機數字，此方法不會使用任何參數：

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

定義您自己的函數

在 ActionScript 3.0 中有兩種方式可以定義函數：您可以使用函數陳述式，或是使用函數運算式。所選擇的技巧需視您偏好較為靜態或較為動態的程式設計方式而定。若您偏好靜態或嚴謹模式的程式設計方式，則可以用函數陳述式定義函數；若有特別需求，則可以用函數運算定義函數，函數運算式比較常用在動態或標準模式的程式設計中。

函數陳述式

在嚴謹模式中，較為偏好使用函數陳述式來定義函數。函數陳述式是以 function 關鍵字做為開頭，後面再加上：

- 函數名稱
- 參數，形成以括號括住的逗號分隔清單
- 函數主體，即叫用函數時所執行的 ActionScript 程式碼，以大括號括住

例如，下列程式碼會建立定義參數的函數，然後使用 "hello" 字串做為參數值來叫用該函數：

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

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

函數運算式

宣告函數的第二種方式是使用具有函數運算式的指定陳述式，而函數運算式有時又稱為函數常值或匿名函數。這是比較詳細而冗長的方法，在舊版 ActionScript 中廣泛地使用。

具有函數運算式的指定陳述式是以 var 關鍵字做為開頭，後面再加上：

- 函數名稱
- 冒號運算子 (:)
- Function 類別，以指出資料類型
- 指定運算子 (=)
- function 關鍵字
- 參數，形成以括號括住的逗號分隔清單
- 函數主體，即叫用函數時所執行的 ActionScript 程式碼，以大括號括住

  例如，下列程式碼會使用函數運算式宣告 traceParameter 函數：

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

請注意，就像在函數陳述式中一樣，您不用指定函數名稱。函數運算式與函數陳述式之間的另一個重大差異是，函數運算式是運算式，而不是陳述式。也就是說，函數運算式不能獨自存在，而函數陳述式則可單獨成立。函數運算式只能做為陳述式 (通常是指定陳述式) 的一部分。下列範例會示範指定給陣列元素的函數運算式：

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

選擇陳述式或運算式

一般原則是，除非有特定情況需要使用運算式，否則就使用函數陳述式。函數陳述式不那麼詳細而冗長，而且在嚴謹模式與標準模式之間可提供比函數運算式更為一致的體驗。

函數陳述式比包含函數運算式的指定陳述式更容易閱讀。函數陳述式可讓您的程式碼更簡潔，也沒有函數運算式那麼複雜，因為函數運算式需要同時使用 var 和 function 兩個關鍵字。

函數陳述式之所以能在兩個編譯器模式之間提供更一致的體驗，在於您可以同時在嚴謹模式與標準模式使用點語法，叫用使用函數陳述式宣告的方法。而用函數運算式宣告的方法就不一定能夠這樣。例如，下列程式碼會定義名為 Example 的類別，此類別具有兩個方法：用函數運算式宣告的 methodExpression() 以及用函數陳述式宣告的 methodStatement()。在嚴謹模式中，您不能使用點語法叫用 methodExpression() 方法。

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

var myEx:Example = new Example();
myEx.methodExpression(); // error in strict mode; okay in standard mode
myEx.methodStatement(); // okay in strict and standard modes
```

一般都認為，函數運算式更適合用來進行著重執行階段或動態行為方式的程式設計。若偏好使用嚴謹模式，但也需要呼叫用函數運算式宣告的方法，您可以使用兩種技巧其中任一種。首先，您可以使用方括號 ([]) 而不使用點 (.) 運算子來呼叫方法。下列方法呼叫在嚴謹模式和標準模式中都會成功：

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

接著，您可以將整個類別宣告為動態類別。雖然這樣可以讓您使用點運算子呼叫方法，不過缺點是犧牲了嚴謹模式中該類別所有實體的一些功能。例如，如果嘗試在動態類別的實體上存取未定義的屬性，編譯器不會產生錯誤。

在特定情況下，函數運算式很有用。函數運算式的其中一個常用用法，就是供只用一次就捨棄的函數使用。另外，較不常用的用法則是用來將函數附加至原型屬性。如需詳細資訊，請參閱原型物件。

在函數陳述式與函數運算式之間有兩個微小的差異，您應該在選擇所要使用的技巧時將其納入考量。第一項差異是，在記憶體管理和記憶體回收方面，函數運算式不會做為物件獨立存在。換句話說，當您將函數運算式指定至另一個物件，例如陣列元素或物件屬性，只是在程式碼中建立該函數運算式的參考。若函數運算式所附加的陣列或物件超出範圍，或是因其它原因無法再使用，您無法再存取該函數運算式。若刪除該陣列或物件，函數運算式所使用的記憶體可供進行記憶體回收，也就是說，該記憶體可開始回收，重新做為其它用途。

下列範例將會為您進行示範，就函數運算式而言，一旦刪除了運算式所指定的屬性，就不能再使用該函數。Test 類別是動態的，也就是說，您可以加入名為 functionExp 的屬性，此屬性會存放函數運算式。functionExp() 函數可以用點運算子加以呼叫，不過一旦刪除 functionExp 屬性，就無法再存取此函數。

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

// function expression
myTest.functionExp = function () { trace("Function expression") };
myTest.functionExp(); // Function expression
delete myTest.functionExp;
myTest.functionExp(); // error
```

在另一方面，如果函數先用函數陳述式加以定義，則可以做為自身的物件而存在，而且即使刪除所附加的屬性之後，也會繼續存在。delete 運算子只會針對物件的屬性作用，因此即使是刪除函數 stateFunc() 本身的呼叫也沒有作用。

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

// function statement
function stateFunc() { trace("Function statement") }
myTest.statement = stateFunc;
myTest.statement(); // Function statement
delete myTest.statement;
delete stateFunc; // no effect
stateFunc();// Function statement
myTest.statement(); // error
```

函數陳述式與函數運算式之間的第二個差異是，函數陳述式在所定義的範圍中一直都存在，包括出現在函數陳述式之前的陳述式中。對照之下，函數運算式則只為後續陳述式定義。例如，下列程式碼在 scopeTest() 函數定義之前順利地呼叫該函數：

```
statementTest(); // statementTest

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

函數運算式在定義之前無法使用，因此下列程式碼就會導致執行階段錯誤：

```
expressionTest(); // run-time error

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

從函數傳回值

若要從函數傳回值，請使用 return 陳述式，後面加上您要傳回的運算式或常值。例如，下列程式碼會傳回代表參數的運算式：

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

請注意，return 陳述式會終止函數，因此 return 陳述式之下的任何陳述式都不會執行，如下所示：

```
function doubleNum(baseNum:int):int {
    return (baseNum * 2);
    trace("after return"); // This trace statement will not be executed.
}
```

在嚴謹模式中，若選擇指定傳回類型，則必須傳回適當類型的值。例如，下列程式碼會在嚴謹模式中產生錯誤，因為它不會傳回有效值：

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

巢狀函數

您可以讓函數形成巢狀結構，也就是說，可以在其它函數之內宣告函數。除非將巢狀函數參考傳遞至外部程式碼，否則一個巢狀函數只能在其父函數內使用。例如，下列程式碼會在 getNameAndVersion() 函數內宣告兩個巢狀函數：

```
function getNameAndVersion():String
{
    function getVersion():String
    {
        return "10";
    }
    function getProductName():String
    {
        return "Flash Player";
    }
    return (getProductName() + " " + getVersion());
}
trace(getNameAndVersion()); // Flash Player 10
```

當巢狀函數傳遞至外部程式碼時，會做為函數結束項傳遞，也就是說，函數會保留定義函數時在範圍內的任何定義。如需詳細資訊，請參閱函數範圍。

函數參數

ActionScript 3.0 提供函數參數一些功能，對剛開始使用此語言的程式設計人員會顯得很新奇。雖然以傳值或傳址的方式傳遞參數的概念對大部分程式設計人員而言應該很熟悉，但 arguments 物件及 ... (rest) 參數對很多人而言可能是新的概念。

以傳值或以傳址方式來傳遞引數

在許多程式語言中，瞭解以傳值或以傳址方式傳遞引數之間的區別相當重要，因為這項區別可能會影響程式碼設計的方式。

以傳值方式來傳遞表示，引數的值會複製到區域變數內，以便在函數中使用。以傳址方式來傳遞表示，只傳遞了引數的參考，而沒有傳遞實際的值；沒有製作實際引數的副本，而是建立了傳遞做為引數的變數參考，並指定給區域變數，以便在函數中使用。區域變數是函數之外的變數參考，讓您能夠變更原始的變數值。

在 ActionScript 3.0 中，所有引數都是以傳址方式傳遞，因為所有的值都會儲存為物件。但是，屬於基本資料類型的物件 (包括 Boolean、Number、int、uint 和 String) 都有特殊運算子，使它們能夠表現得像是以傳值方式傳遞。例如，下列程式碼會建立名為 passPrimitives() 的函數，以定義兩個同樣是 int 類型的參數：xParam 和 yParam。這些參數類似於在 passPrimitives() 函數主體內部宣告的區域變數。當函數是以引數 xValue 和 yValue 呼叫時，參數 xParam 和 yParam 都是用以 xValue 和 yValue 代表的 int 物件參考進行初始化。因為引數是基本型，所以表現得就像是以傳值方式傳遞一樣。雖然 xParam 和 yParam 最初只包含 xValue 和 yValue 物件的參考，但是在函數主體之內的任何變數變更都會在記憶體中產生新的值副本。

```
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
```

在 passPrimitives() 函數內，xParam 和 yParam 的值是遞增的，但這並不會影響 xValue 和 yValue 的值，如上一個 trace 陳述式所示。即使參數名稱與變數名稱 xValue 和 yValue 完全相同也是如此，因為函數之內的 xValue 和 yValue 會指向記憶體中的新位置，這一點與函數之外名稱相同的變數不同。

所有其它物件 (也就是不屬於基本資料類型的物件) 都是依傳址方式傳遞，讓您能夠改變原始變數的值。例如，下列程式碼會建立名為 objVar 的物件，具有 x 和 y 兩個屬性。物件會以引數形式傳遞給 passByRef() 函數。因為該物件不是基本類型，所以該物件不但是以傳址方式傳遞，而且還會一直保持為參考。這表示，在函數內對參數所做的變更會影響函數外的物件屬性。

```
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
```

objParam 參數會參考全域 objVar 變數所參考的相同物件。如範例中的 trace 陳述式所示，對 objParam 物件之 x 和 y 屬性所做的變更已反映在 objVar 物件中。

預設參數值

在 ActionScript 3.0 中，您可以宣告函數的預設參數值。若呼叫有預設參數值的函數會省略有預設值的參數，就使用該參數在函數定義中指定的值。所有具預設值的參數都必須放在參數清單最後。指定做為預設值的值必須是編譯階段常數。參數若有預設值存在，便可以有效地讓參數成為 「選擇性的參數」。沒有預設值的參數即視為 「必要參數」。

例如，下列程式碼會建立具有三個參數的函數，其中有兩個有預設值。當函數只用一個參數呼叫時，就會使用參數的預設值。

```
function defaultValues(x:int, y:int = 3, z:int = 5):void
{
    trace(x, y, z);
}
defaultValues(1); // 1 3 5
```

arguments 物件

當參數傳遞給函數時，您可以使用 arguments 物件，存取有關傳遞給函數之參數的資訊。arguments 物件的重要性如下：

- arguments 物件是陣列，其中包含傳遞給函數的所有參數。
- arguments.length 屬性會報告要傳遞給函數的參數數目。
- arguments.callee 屬性會提供參考給函數本身，對函數運算式的遞迴呼叫很有用。

  備註：您無法在下列情況下使用 arguments 物件 - 如果有任何參數命名為 arguments，或是如果使用 ... (rest) 參數。

  如果在函數主體中參考 arguments 物件，ActionScript 3.0 就允許函數呼叫包含比函數定義中所定義更多的參數，但若參數的數目與必要參數 (並與任何選用的參數) 的數目不符，就會在嚴謹模式中產生編譯器錯誤。您可以使用 arguments 物件的陣列方面，存取傳遞給函數的任何參數，而不管該參數是否在函數定義中定義。僅能在標準模式下編譯的下列範例會使用 arguments 陣列與 arguments.length 屬性，以追蹤傳遞給 traceArgArray() 函數的所有參數：

```
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
```

arguments.callee 屬性經常用於匿名函數中，以建立遞迴。您可以用它來增加程式碼的彈性。如果您在開發週期的過程中變更了遞迴函數的名稱，當您使用 arguments.callee 而不用函數名稱時，就不需要擔心變更函數主體中的遞迴呼叫。arguments.callee 屬性用於下列函數運算式中，以啟用遞迴：

```
var factorial:Function = function (x:uint)
{
    if(x == 0)
    {
        return 1;
    }
    else
    {
        return (x * arguments.callee(x - 1));
    }
}

trace(factorial(5)); // 120
```

如果您在函數宣告中使用 ... (rest) 參數，就無法使用 arguments 物件。而必須使用為它們宣告的參數名稱來存取參數。

您也應該小心避免使用字串 "arguments" 做為參數名稱，因為它會遮蔽 arguments 物件。例如，若重新撰寫 traceArgArray() 函數以加入 arguments 參數，則函數主體中的 arguments 參考的是參數，而不是 arguments 物件。下列程式碼不會產生輸出：

```
function traceArgArray(x:int, arguments:int):void
{
    for (var i:uint = 0; i < arguments.length; i++)
    {
        trace(arguments[i]);
    }
}

traceArgArray(1, 2, 3);

// no output
```

在舊版 ActionScript 中，arguments 物件也包含名為 caller 的屬性，它是參考呼叫目前函數的函數；在 ActionScript 3.0 中則沒有 caller 屬性，但是如果您需要參考呼叫函數，可以改變呼叫函數，以便傳遞參考它本身的額外參數。

... (rest) 參數

ActionScript 3.0 中加入了新的參數宣告，稱為 ... (rest) 參數。此參數可以讓您指定接受任何以逗號分隔引數數目的陣列參數。該參數可以包含任何不是保留字的名稱。此參數宣告必須是最後指定的參數。使用此參數將會使得 arguments 物件無法使用。雖然 ... (rest) 參數可提供您等同 arguments 陣列與 arguments.length 屬性的功能，但卻無法提供類似 arguments.callee 所提供的功能。您應該先確認不需要使用 arguments.callee，之後再使用 ... (rest) 參數。

下列範例會使用 ... (rest) 參數 (而非 arguments 物件) 重新撰寫 traceArgArray() 函數。

```
function traceArgArray(... args):void
{
    for (var i:uint = 0; i < args.length; i++)
    {
        trace(args[i]);
    }
}

traceArgArray(1, 2, 3);

// output:
// 1
// 2
// 3
```

... (rest) 參數也可與其他參數一併使用，只要該參數是列出的最後一個參數即可。下列範例會修改 traceArgArray() 函數，讓它的第一個參數 x 為 int 類型，而第二個參數再使用 ... (rest) 參數。輸出會略過第一個值，因為第一個參數不再是由 ... (rest) 參數建立的陣列一部分。

```
function traceArgArray(x: int, ... args)
{
    for (var i:uint = 0; i < args.length; i++)
    {
        trace(args[i]);
    }
}

traceArgArray(1, 2, 3);

// output:
// 2
// 3
```

函數為物件

在 ActionScript 3.0 中，函數就是物件。當您建立函數時，所建立的物件不但能當做參數傳遞給另一個函數，而且當中還會附加屬性和方法。

當做引數傳遞至另一個函數的函數是以傳址方式傳遞，而不是以傳值方式傳遞。當您傳遞函數做為引數時，只需使用識別名稱即可，而不需使用用來呼叫方法的括號運算子。例如，下列程式碼會將名為 clickListener() 的函數做為引數，傳遞至 addEventListener() 方法：

```
addEventListener(MouseEvent.CLICK, clickListener);
```

雖然對剛使用 ActionScript 的程式設計人員而言可能有點陌生，但函數就像任何其他物件一樣，可以具有屬性和方法。事實上，每個函數都有名為 length 的唯讀屬性，其儲存為函數定義的參數數目。這與 arguments.length 屬性不同，此屬性會報告傳送給函數的引數數目。不知您是否還記得，在 ActionScript 中，傳遞給函數的引數數目可以超過為該函數所定義的參數數目。下列範例只在標準模式中編譯，因為嚴謹模式要求傳遞的引數數目與定義的參數數目必須完全相符，此範例會顯示這兩個屬性之間的差異：

```
// Compiles only in standard mode
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 */
```

在標準模式中，您可以定義函數主體之外的函數屬性，藉以定義自己的函數屬性。函數屬性可以做為準靜態屬性，讓您儲存與函數相關之變數的狀態。例如，您可能要追蹤呼叫特定函數的次數。如果是撰寫遊戲，而要追蹤使用者使用特定命令的次數，這種功能可能會很有用；不過您也可以使用靜態類別屬性來執行這項作業。因為嚴謹模式不會讓您將動態屬性加入至函數，而僅能在標準模式中編譯的下列範例，會在函數宣告之外建立函數屬性，並在每次呼叫函數時遞增屬性：

```
// Compiles only in standard mode
var someFunction:Function = function ():void
{
    someFunction.counter++;
}

someFunction.counter = 0;

someFunction();
someFunction();
trace(someFunction.counter); // 2
```

函數範圍

函數的範圍不僅決定程式中可以呼叫函數之處，而且可以決定函數能存取的定義。套用至變數識別名稱的範圍規則也可套用至函數識別名稱。全域範圍中的函數宣告在整個程式碼中都可以使用。例如，ActionScript 3.0 包含全域函數 (例如 isNaN() 和 parseInt())，可在程式碼任一處使用。巢狀函數 (在另一個函數內宣告的函數) 可用在其宣告函數中的任何一處。

範圍鏈

只要函數一開始執行，就會建立一些物件和屬性。首先建立稱為 「啟動物件」的特殊物件，其中儲存參數及任何區域變數，或是在函數主體中宣告的函數。您不能直接存取啟動物件，因為它是內部機制。接著會建立 「範圍鏈」，其中包含執行階段檢查會識別名稱宣告的物件順序清單。每一個執行的函數都有儲存在內部屬性中的範圍鏈。對巢狀函數來說，範圍鏈是自其本身的啟動物件開始，後面加上其父函數的啟動物件。範圍鏈以這種方式繼續，直至達到全域物件為止。全域物件是在 ActionScript 程式開始時建立，並包含所有全域變數和函數。

函數結束項

「函數結束項」是一種物件，其中包含函數及其 「語彙環境」的快照。函數的語彙環境包含函數之範圍鏈中的所有變數、屬性、方法和物件，以及其值。只要在物件或類別之外執行函數，就會建立函數結束項。由於函數結束項保留其定義範圍的情況，當函數傳遞為引數或傳回值至不同範圍中時，就會產生有趣的結果。

例如，下列程式碼會建立兩個函數：foo() 會傳回名為 rectArea() 的巢狀函數，可計算矩形區域；而 bar() 會呼叫 foo() 並將傳回的函數結束項儲存在名為 myProduct 的變數中。即使 bar() 函數會定義其自身的區域變數 x (值為 2)，當呼叫函數結束項 myProduct() 時，它會保留於函數 foo() 中定義的變數 x (值為 40)。因此 bar() 函數會傳回 160 這個值，而不是 8。

```
function foo():Function
{
    var x:int = 40;
    function rectArea(y:int):int // function closure defined
    {
        return x * y
    }
    return rectArea;
}
function bar():void
{
    var x:int = 2;
    var y:int = 4;
    var myProduct:Function = foo();
    trace(myProduct(4)); // function closure called
}
bar(); // 160
```

方法表現的行為也類似，它們也會保留有關建立其自身的語彙環境資訊。這項特性在從其實體擷取方法時最顯著，它會建立繫結方法。函數結束項與繫結方法之間的主要差異在於，繫結方法中 this 關鍵字的值一律會參考原始附加的實體；而在函數結束項中，this 關鍵字的值則可變更。

第 4 章 使用 ActionScript 設計物件導向程式

物件導向程式設計簡介

物件導向程式設計 (OOP) 是利用程式來整理程式碼的方式，方法就是按物件分成群組。如此看來，「物件」這個名詞表示個別的元素，裡面包含資訊 (資料值) 和功能。使用物件導向做法來組織程式，可以讓您將特定資訊與共通功能或是該資訊的相關聯動作組合在一起。例如，將專輯標題、樂曲標題或作曲家名稱等音樂資訊，與功能 「新增樂曲至播放清單」或是 「播放此作曲家的所有歌曲」組合在一起。這些部分會組合成單一項目，即物件 (例如 「Album」或 「MusicTrack」)。結合這些值和函數的功能具有多項優點。最重要的好處就是您只需使用一個變數，不用再使用多個變數了。此外，它會將相關功能放在一起。最後，將資訊與功能結合一起可以讓您使用與真實情況更接近的方式來設計程式架構。

類別

類別是物件的抽象形式，類別會儲存有關物件可保存之資料類型以及物件所能展現之行為方式的資訊。當您撰寫小型 Script，而其中只包含彼此互動的少數幾個物件時，這種抽象觀念的用處也許不太明顯。不過，程式的範圍會讓必須管理的物件數量增加。在這個情況下，類別可以讓您更好控制物件的建立方式以及彼此的互動方式。

早在 ActionScript 1.0 時，ActionScript 程式設計人員就能使用 Function 物件來建立類似類別的建構；ActionScript 2.0 中更將 class 和 extends 等關鍵字正式加入類別的支援；ActionScript 3.0 不但繼續支援 ActionScript 2.0 中導入的關鍵字，而且還添加一些新功能。例如，ActionScript 3.0 以 protected 和 internal 特質加強存取控制。它也使用 final 和 override 關鍵字更能夠控制繼承。

若開發人員曾經以程式設計語言 (如 Java、C++ 或 C# 等) 建立類別，就會發現 ActionScript 可提供類似的體驗。ActionScript 有很多相同的關鍵字和特質名稱，例如 class、extends 和 public。

備註：在 Adobe ActionScript 文件中，「屬性」一詞代表物件或類別的任何成員，包括變數、常數和方法。此外，雖然 「類別」和 「靜態」這兩個詞經常互換代用，但在此處卻是兩個不同的詞。例如，「類別屬性」是用來表示類別的所有成員，而不只是靜態成員。

類別定義

ActionScript 3.0 類別定義使用的語法類似於 ActionScript 2.0 中類別定義所使用的語法。正確的類別定義語法必須要有 class 關鍵字，後面再加上類別名稱。類別主體是以大括號 ({}) 括住，後面再加上類別名稱。例如，下列程式碼會建立名為 Shape 的類別，其中包含一個變數 visible：

```
public class Shape
{
    var visible:Boolean = true;
}
```

有一項重大的語法變更與套件中的類別定義息息相關。在 ActionScript 2.0 中，若類別位於套件之中，則套件名稱必須包含於類別定義中。而在 ActionScript 3.0 中，由於導入 package 陳述式，套件名稱必須包含於套件宣告中，而不是包含於類別宣告中。例如，下列類別宣告會示範，BitmapData 類別 (屬於 flash.display 套件的一部分) 如何分別在 ActionScript 2.0 和 ActionScript 3.0 中進行定義：

```
// ActionScript 2.0
class flash.display.BitmapData {}

// ActionScript 3.0
package flash.display
{
    public class BitmapData {}
}
```

Class 特質

ActionScript 3.0 可以讓您使用下列四個特質的其中一個，修改類別定義：

特質	定義
dynamic	可以讓您在執行階段將屬性加入實體。
final	絕對不可以由另一個類別加以擴充。
internal (預設值)	目前套件之內的參考可以看見。
public	任何一處的參考都可以看見。

針對上述每一個特質 (除了 internal 以外)，您都必須明確包含該特質，才能取得與其關聯的行為方式。例如，定義類別時，若沒有包含 dynamic 特質，就無法在執行階段將屬性加入至類別實體。您可以將特質放在類別定義的開頭，明確地加以指定，如下列程式碼所示範：

```
dynamic class Shape {}
```

請注意，清單中並未包含名為 abstract 的特質，ActionScript 3.0 不支援抽象類別。另外，也請注意，清單不包含名為 private 和 protected 的特質。這兩個特質只有在類別定義之內才有意義，而無法套用至類別本身。若不要讓類別在套件之外公開可見，請將類別放入套件之內，並將該類別標示為 internal 特質；或者，您也可以同時省略 internal 和 public 特質，編譯器會自動為您加入 internal 特質。您也可以定義類別，讓類別只能在定義的來源檔案中看到。請將類別放在來源檔案最底端，套件定義的結尾大括號之下。

類別主體

類別主體是以大括號括住。它會定義類別的變數、常數和方法。下列範例顯示 ActionScript 3.0 中 Accessibility 類別的宣告：

```
public final class Accessibility
{
    public static function get active():Boolean;
    public static function updateProperties():void;
}
```

您也可以在類別主體之內定義命名空間。下列範例示範如何在類別主體之中定義命名空間，然後在該類別中用來做為方法的特質：

```
public class SampleClass
{
    public namespace sampleNamespace;
    sampleNamespace function doSomething():void;
}
```

ActionScript 3.0 不但可以讓您在類別主體中包含定義，也可以包含陳述式。位於類別主體但位於方法定義之外的陳述式，只會執行一次。執行的時機為初次遇到類別定義，並建立關聯的類別物件時。下列範例包含外部函數 hello() 的呼叫，以及在定義類別時，輸出確認訊息的 trace 陳述式：

```
function hello():String
{
    trace("hola");
}
class SampleClass
{
    hello();
    trace("class created");
}
// output when class is created
hola
class created
```

ActionScript 3.0 允許在相同類別主體中以相同名稱定義靜態屬性和實體屬性。例如，下列程式碼會宣告名為 message 的靜態變數，以及名稱相同的實體變數：

```
class StaticTest
{
    static var message:String = "static variable";
    var message:String = "instance variable";
}
// In your script
var myST:StaticTest = new StaticTest();
trace(StaticTest.message); // output: static variable
trace(myST.message); // output: instance variable
```

類別屬性特質

在探討 ActionScript 物件模型中，「屬性」一詞可以代表類別的任何成員，包括變數、常數和方法，不過在 「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考 」書中，這個詞彙適用的範圍較窄。就上下文來說，屬性這個詞彙只包括屬於變數的類別成員，或由 getter、setter 方法定義的類別成員。在 ActionScript 3.0 中，有一組特質可配合任何類別的屬性使用，下表列出這組特質的清單。

特質	定義
internal (預設值)	相同套件之內的參考可以看見。
private	相同類別之中的參考可以看見。
protected	相同類別及衍生類別之中的參考可以看見。
public	任何一處的參考都可以看見。
static	指定屬性屬於類別，而不屬於該類別的實體。
UserDefinedNamespace	由使用者定義的自訂命名空間名稱。

存取控制命名空間特質

ActionScript 3.0 提供四個特殊的特質，可控制類別之內所定義屬性的存取：public、private、protected 和 internal。

public 特質可讓屬性在您的 Script 中隨處可見。例如，若要讓方法可供套件之外的程式碼使用，您必須以 public 特質宣告該方法。這對任何屬性都成立，不管是使用 var、const 或 function 關鍵字宣告都一樣。

private 特質可以讓一個屬性只有其定義類別的呼叫者可以看見；這種行為方式與 ActionScript 2.0 中的 private 特質不同，在後者中可以讓子類別存取父類別中的私有屬性。另外一項重要的行為方式改變則與執行階段存取有關：在 ActionScript 2.0 中，private 關鍵字只在編譯階段才禁止存取，而在執行階段則很容易規避限制加以運用；在 ActionScript 3.0 中，情形已經改變，標記為 private 的屬性在編譯階段或執行階段都無法使用。

例如，下列程式碼會建立名為 PrivateExample 而有一個私有變數的簡單類別，然後會嘗試從類別之外存取此私有變數。

```
class PrivateExample
{
    private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);// compile-time error in strict mode
trace(myExample["privVar"]); // ActionScript 2.0 allows access, but in ActionScript 3.0, this is a run-time
error.
```

在 ActionScript 3.0 中，若使用嚴謹模式，嘗試使用點運算子 (myExample.privVar) 存取私有屬性會導致編譯階段錯誤；否則，就會在執行階段報告錯誤，跟使用屬性存取運算子 (myExample["privVar"]) 時的情形一樣。

下表摘要列出嘗試存取屬於密封 (而非動態) 類別之私有屬性的結果：

	嚴謹模式	標準模式
點運算子 (.)	編譯階段錯誤	執行階段錯誤
方括號運算子 ([])	執行階段錯誤	執行階段錯誤

在以 dynamic 特質宣告的類別中，嘗試存取私有變數並不會產生執行階段錯誤，相反的，變數會看不到，因此傳回 undefined。但是如果在嚴謹模式中使用點運算子，則會發生編譯階段錯誤。下列範例基本上與上一個範例完全相同，只不過 PrivateExample 類別是宣告為動態類別：

```
dynamic class PrivateExample
{
    private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);// compile-time error in strict mode
trace(myExample["privVar"]); // output: undefined
```

一般來說，類別之外的程式碼嘗試存取私有屬性時，動態類別會傳回值 undefined 而不會產生錯誤。下表顯示，只有使用了點運算子，在嚴謹模式中存取私有屬性時，才會產生錯誤：

	嚴謹模式	標準模式
點運算子 (.)	編譯階段錯誤	undefined
方括號運算子 ([])	undefined	undefined

protected 特質是 ActionScript 3.0 中新增的特質，它會讓位於屬性本身所在類別或子類別之中的呼叫者看見該屬性；也就是說，一個保護的屬性是在本身類別之內，或是對於位在其下繼承階層中任何一處的類別，才能使用，不管子類別是位於相同或不同的套件中都一樣。

至於熟悉 ActionScript 2.0 的人，這個功能和 ActionScript 2.0 中的 private 特質很類似。ActionScript 3.0 protected 特質也和 Java 中的 protected 特質類似。不同的是，Java 版也允許在相同套件之內存取呼叫者。當您的子類別需要變數或方法，而您要隱藏此變數或方法，不要讓繼承鏈之外的程式碼看見時，protected 特質就很有用。

internal 特質是 ActionScript 3.0 中新增的特質，可以讓位於屬性本身套件中的呼叫者看見該屬性。這是套件之內程式碼的預設特質，會套用至沒有下列其它特質的任何屬性：

- public
- private
- protected
- 使用者定義的命名空間

internal 特質與 Java 中的預設存取控制類似，不過在 Java 中此層級的存取並沒有明確的名稱，而僅能透過省略其它任何存取修飾語來完成這項操作。您可以在 ActionScript 3.0 中使用 internal 特質，讓您能夠選擇明確地表示，是刻意要讓屬性僅供在本身套件之內的呼叫者看見。

static 特質

static 特質可配合以 var、const 或 function 關鍵字宣告的屬性一併使用，讓您將屬性附加至類別，而不是附加至類別的實體。類別外部的程式碼必須使用類別名稱 (而不是使用實體名稱) 呼叫靜態屬性。

靜態屬性並不會由子類別加以繼承，但這種屬性是子類別範圍鏈的一部分；也就是說，在子類別的主體之內，不必參考定義此變數或方法的類別，就可以使用靜態變數或方法。

使用者定義的命名空間特質

若不使用預先定義的存取控制特質，另外還有一個方法，就是建立自訂命名空間做為特質使用。每個定義只有一個命名空間特質可用，而且不能結合命名空間特質與任何存取控制特質 (public、private、protected、internal) 同時使用。

變數

變數可以用 var 或 const 關鍵字進行宣告。以 var 關鍵字宣告的變數可以在整個 Script 執行期間多次變更其值；以 const 關鍵字宣告的變數稱為 「常數」，則僅能指定一次值，若嘗試指定新的值給已初始化的常數，會導致錯誤。

靜態變數

靜態變數是結合使用 static 關鍵字及 var 或 const 陳述式所宣告的。靜態變數是附加至類別而不是附加至類別實體，在儲存及共用適用於整個物件的類別資訊時很有用。例如，若要保持類別實體化的次數記錄，或者，若要儲存可允許的最大類別實體數目，靜態變數就很適用。

下列範例會建立 totalCount 變數，以追蹤類別實體化次數，也會建立 MAX_NUM 常數，以儲存最大的實體數目。totalCount 和 MAX_NUM 是靜態變數，因為它們所包含的值可套用至整個類別而不是特定的實體。

```
class StaticVars
{
    public static var totalCount:int = 0;
    public static const MAX_NUM:uint = 16;
}
```

在 StaticVars 類別之外的程式碼及其任何子類別都是只能透過類別本身來參考 totalCount 和 MAX_NUM 屬性。例如，下列程式碼可以正確運作：

```
trace(StaticVars.totalCount); // output: 0
trace(StaticVars.MAX_NUM); // output: 16
```

您不能透過類別的實體來存取靜態變數，因此下列程式碼會傳回錯誤：

```
var myStaticVars:StaticVars = new StaticVars();
trace(myStaticVars.totalCount); // error
trace(myStaticVars.MAX_NUM); // error
```

同時以 static 和 const 關鍵字宣告的變數必須在宣告常數的同時進行初始化，StaticVars 類別的 MAX_NUM 也是一樣。您不能指定值給建構函式或實體方法之內的 MAX_NUM。下列程式碼會產生錯誤，因為它不是初始化靜態常數的有效方式：

```
// !! Error to initialize static constant this way
class StaticVars2
{
    public static const UNIQUESORT:uint;
    function initializeStatic():void
    {
        UNIQUESORT = 16;
    }
}
```

實體變數

實體變數包含以 var 和 const 關鍵字而不用 static 關鍵字宣告的屬性。實體變數會附加至類別實體而不是附加至整個類別，在儲存實體特有的值時很有用。例如，Array 類別具有名為 length 的實體屬性，該實體屬性儲存 Array 類別之特定實體所保存的陣列元素數目。

實體變數不管是宣告為 var 或 const，都不能在子類別中加以覆寫，但是您可以透過覆寫 getter 和 setter 方法，達到類似於覆寫變數的功能。

方法

方法是類別定義之一部分的函數，一旦建立了類別的實體之後，方法即繫結至該實體。方法與在類別之外宣告的函數不同，它不能離開所附加之實體單獨使用。

方法是使用 function 關鍵字所定義的。如同任何類別屬性一樣，您也可以將任何類別屬性特質套用至方法，其中包括 private、protected、public、internal、static 或自訂命名空間。您可以使用如下所示的函數陳述式：

```
public function sampleFunction():String {}
```

也可以使用您指定了函數運算式的變數，如下所示：

```
public var sampleFunction:Function = function () {}
```

一般來說，您要使用函數陳述式，而不要使用函數運算式，理由如下：

- 函數陳述式會更簡潔，也更容易閱讀。
- 函數陳述式可以讓您使用 override 和 final 關鍵字。
- 函數陳述式會在識別名稱 (函數的名稱) 與方法主體之內的程式碼之間建立更強的繫結。由於變數的值可以用指定陳述式加以變更，變數及其函數運算式之間的連線隨時都可能會被切斷，雖然可以用 const (而不是用 var) 宣告變數的方式來解決這個問題，但一般並不認為這種技巧是最佳做法。它會讓程式碼很難閱讀，而且無法使用 override 和 final 關鍵字。

但是，在某種情況下您必須使用函數運算式：就是當您選擇將函數附加至原型物件時。

建構函式方法

建構函式方法有時稱為「建構函式」，是與它們的定義類別共用相同名稱的函數。只要類別的實體以 new 關鍵字建立，就會執行您包含在建構函式方法中的任何程式碼。例如，下列程式碼會定義名為 Example 的簡單類別，其中只包含一個名為 status 的屬性，status 變數的初始值是在建構函數中設定。

```
class Example
{
    public var status:String;
    public function Example()
    {
        status = "initialized";
    }
}

var myExample:Example = new Example();
trace(myExample.status); // output: initialized
```

建構函式方法只能是公用的，但是否使用 public 特質卻是選擇性的。您不能在建構函式上使用任何其它存取控制指定字，包括 private、protected 或 internal。您也不能使用具有建構函式方法的使用者定義命名空間。

建構函式可以使用 super() 陳述式，明確呼叫其直屬父類別的建構函式；如果沒有明確地呼叫父類別建構函式，編譯器會自動在建構函式主體的第一個陳述式之前插入呼叫。您也可以使用 super 前置詞做為父類別的參考，以呼叫父類別的方法。若決定在相同的建構函式主體中，同時使用 super() 和 super，一定要先呼叫 super()；否則，super 參考就不會表現出預期的行為方式。super() 建構函式也應該在任何 throw 或 return 陳述式之前進行呼叫。

下列範例會示範若嘗試在呼叫 super() 建構函式之前使用 super 參考，所發生的情況。新類別 ExampleEx 會擴充 Example 類別，ExampleEx 建搆函式會嘗試存取定義於其父類別的狀態變數，但會在呼叫 super() 之前進行。在 ExampleEx 建構函式之內的 trace() 陳述式會產生 null 值，因為 status 變數要在執行 super() 建構函式之後才能使用。

```
class ExampleEx extends Example
{
    public function ExampleEx()
    {
        trace(super.status);
        super();
    }
}

var mySample:ExampleEx = new ExampleEx(); // output: null
```

雖然在建構函式之內使用 return 陳述式是合法的，但卻不允許傳回值；換句話說，return 陳述式絕對不能有相關聯的運算式或值，因此，建構函式方法也不允許傳回值。也就是說，無法指定任何傳回類型。

若不在類別中定義建構函式方法，編譯器會自動為您建立空白的建構函式。若您的類別擴充另一個類別，編譯器會在所產生的建構函式中包含 super() 呼叫。

靜態方法

靜態方法也稱為「類別方法」，是以 static 關鍵字宣告的方法。靜態方法是附加至類別而不是附加至類別的實體，適用於封裝影響個別實體狀態以外的功能。由於靜態方法是附加至類別整體，靜態方法只能透過類別存取，而無法透過類別的實體存取。

靜態方法適用於封裝不限於影響類別實體狀態的功能；換句話說，若方法提供的功能不會直接影響類別實體的功能，則方法應該是靜態的。例如，Date 類別具有名為 parse() 的靜態方法，它接受字串並轉換成數字；方法是靜態的，因為它不會影響類別的個別實體，而是由 parse() 方法接受代表日期值的字串，解析該字串，然後以與 Date 物件之內部形式相容的格式傳回數字。這個方法不是實體方法，因為將此方法套用至 Date 類別的實體沒有意義。

若將靜態 parse() 方法與 Date 類別的其中一個實體方法 (例如 getMonth()) 對照比較，就會發現到：getMonth() 方法是實體方法，因為它是透過擷取 Date 實體的特定組件 (月份)，直接在實體的值上運作。

由於靜態方法並不與個別實體繫結，您不能在靜態方法的主體之中使用關鍵字 this 或 super，this 參考和 super 參考都是只在實體方法的內容之中才有意義。

靜態方法與其它以類別為基礎的程式設計語言不同，在 ActionScript 3.0 中的靜態方法是不繼承的。

實體方法

實體方法是不使用 static 關鍵字宣告的方法。實體方法會附加至類別的實體而不是類別整體，適用於實作會影響類別之個別實體的功能。例如，Array 類別包含名為 sort() 的實體方法，它是直接在 Array 實體上運作。

在實體方法的主體之內，靜態和實體變數都在範圍中，也就是說，在相同類別中定義的變數可以使用簡單的識別名稱進行參考。舉例來說，下列類別 CustomArray 會擴充 Array 類別，CustomArray 類別會定義名為 arrayCountTotal 的靜態變數以追蹤類別實體的總數、定義名為 arrayNumber 的實體變數以追蹤實體建立順序，以及定義名為 getPosition() 的實體方法傳回這些變數值。

```
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);
    }
}
```

雖然類別之外的程式碼必須使用 CustomArray.arrayCountTotal，透過類別物件來存取 arrayCountTotal 靜態變數，但位於 getPosition() 方法主體之內的程式碼則可以直接參考 arrayCountTotal 靜態變數。即使是在父類別中的靜態變數也是如此。雖然靜態屬性在 ActionScript 3.0 中並不繼承，但父類別中的靜態屬性卻是在範圍中。例如，Array 類別有一些靜態變數，其中一個是名為 DESCENDING 的常數；位於 Array 子類別中的程式碼可以使用簡單的識別名稱，存取靜態常數 DESCENDING。

```
public class CustomArray extends Array
{
    public function testStatic():void
    {
        trace(DESCENDING); // output: 2
    }
}
```

實體方法主體之中 this 參考的值是方法所附加實體的參考。下列程式碼會示範 this 參考指向包含該方法的實體：

```
class ThisTest
{
    function thisValue():ThisTest
    {
        return this;
    }
}

var myTest:ThisTest = new ThisTest();
trace(myTest.thisValue() == myTest); // output: true
```

實體方法的繼承可以用 override 和 final 關鍵字加以控制；您可以使用 override 特質，重新定義繼承的方法，而用 final 特質防止子類別覆寫方法。

get 和 set 存取子方法

Get 和 set 存取子函數也稱為 getter 和 setter，讓您在為所建立類別提供容易使用之程式設計介面時，也遵守資訊隱藏和封裝的程式設計原則。Get 和 set 函數可以讓您將類別屬性保持為類別私有，但可以讓類別的使用者以存取類別變數的方式來存取這些屬性，而不是呼叫類別方法來存取。

這種做法的優點是可以讓您避免名稱大而無當的傳統存取子函數，例如 getPropertyName() 和 setPropertyName()。使用 getter 和 setter 的另一項好處是可以避免每一個屬性都具有同時允許讀取及寫入存取的兩個公用外表的函數。

下列範例類別名為 GetSet，包括名為 publicAccess() 的 get 和 set 存取子函數，可讓您存取名為 privateProperty 的私有變數：

```
class GetSet
{
    private var privateProperty:String;

    public function get publicAccess():String
    {
        return privateProperty;
    }

    public function set publicAccess(setValue:String):void
    {
        privateProperty = setValue;
    }
}
```

若嘗試直接存取 privateProperty 屬性，將會導致錯誤，如下所示：

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.privateProperty); // error occurs
```

GetSet 類別的使用者則會使用看起來像名為 publicAccess 的屬性，但其實是一對 get 和 set 存取子函數，在名為 privateProperty 的私有屬性上運作。下列範例會將 GetSet 類別實體化，然後使用名為 publicAccess 的公用存取子設定 privateProperty 的值：

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.publicAccess); // output: null
myGetSet.publicAccess = "hello";
trace(myGetSet.publicAccess); // output: hello
```

Getter 和 setter 函數也可以讓您覆寫繼承自父類別的屬性，這是使用一般類別成員變數時不可能執行的作業。使用 var 關鍵字宣告的類別成員變數無法在子類別中加以覆寫；但是使用 getter 和 setter 函數建立的屬性卻沒有這項限制。您可以在繼承自父類別的 getter 和 setter 函數上使用 override 特質。

繫結方法

繫結方法有時稱為「方法結束項」，其實就是自方法實體擷取的方法。繫結方法的範例包括傳遞給函數做為引數的方法，或從函數傳回做為值的方法。繫結方法是 ActionScript 3.0 中新增的方法，類似於函數結束項，即使自方法實體擷取時都會保留其語彙環境。但是，繫結方法與函數結束項之間最主要的差異在於繫結方法的 this 參考會保持連結 (或稱為繫結) 至實作方法的實體，換句話說，繫結方法中的 this 參考永遠都指向實作方法的原始物件；而函數結束項的 this 參考則是一般參考。也就是說，它會指向叫用時與函數關聯的任何物件。

若使用 this 關鍵字，則一定要瞭解繫結方法。請回想 this 關鍵字提供方法之父物件的參考，大部分 ActionScript 程式設計人員，都會預期 this 關鍵字永遠都代表包含方法的定義物件或類別；但是若沒有方法繫結，則不可能永遠都是如此。例如，在舊版 ActionScript 中，this 參考就不會永遠都指向實作方法的實體。在 ActionScript 2.0 中，當方法是從實體中擷取時，不但 this 參考不繫結至原始實體，成員變數和實體類別的方法也無法使用；在 ActionScript 3.0 中這不是問題，因為當您傳遞方法做為參數時，會自動建立繫結方法，繫結方法會確保 this 關鍵字永遠參考定義方法的物件或類別。

下列程式碼會定義名為 ThisTest 的類別，其中包含定義繫結方法而名為 foo() 的方法，以及傳回繫結方法而名為 bar() 的方法；類別之外的程式碼會建立 ThisTest 類別的實體，呼叫 bar() 方法，然後將傳回值儲存在名為 myFunc 的變數中。

```
class ThisTest
{
    private var num:Number = 3;
    function foo():void // bound method defined
    {
        trace("foo's this: " + this);
        trace("num: " + num);
    }
    function bar():Function
    {
        return foo; // bound method returned
    }
}

var myTest:ThisTest = new ThisTest();
var myFunc:Function = myTest.bar();
trace(this); // output: [object global]
myFunc();
/* output:
foo's this: [object ThisTest]
output: num: 3 */
```

程式碼的最後兩行會示範，即使該程式碼行中就在它之前的 this 參考指向全域物件，繫結方法 foo() 中的 this 參考仍然指向 ThisTest 類別的實體；而且，儲存在 myFunc 變數中的繫結方法也仍然可以存取 ThisTest 類別的成員變數。如果在 ActionScript 2.0 中執行相同的程式碼，this 參考會相符，而 num 變數就會成為 undefined。

加上繫結方法最顯著之處與事件處理常式相關，因為 addEventListener() 方法會要求您傳遞函數或方法做為引數。

列舉與類別

列舉是您建立的自訂資料類型，可用來封裝一小組值。ActionScript 3.0 不支援特定列舉功能，這一點跟 C++ 及其 enum 關鍵字或 Java 及其 Enumeration 介面都不同，但是，您可以使用類別和靜態常數建立列舉項目。例如，ActionScript 3.0 中的 PrintJob 類別會使用名為 PrintJobOrientation 的列舉項目，儲存 "landscape" 和 "portrait" 這兩個值，如下列程式碼所示：

```
public final class PrintJobOrientation
{
    public static const LANDSCAPE:String = "landscape";
    public static const PORTRAIT:String = "portrait";
}
```

依照慣例，列舉類別會以 final 特質宣告，因為不必再擴充該類別。此類別僅包含靜態成員。也就是說，您不用建立類別的實體。而是直接透過類別物件存取列舉值，如下列程式碼摘錄所示：

```
var pj:PrintJob = new PrintJob();
if(pj.start())
{
    if (pj.orientation == PrintJobOrientation.PORTRAIT)
    {
        ...
    }
    ...
}
```

ActionScript 3.0 中所有列舉類別都只包含 String、int 或 uint 類型的變數。使用列舉而不用常值字串或數字值的優點是使用列舉時很容易找出拼寫錯誤；如果列舉名稱有打字錯誤時，ActionScript 編譯器會產生錯誤。若使用常值，如果一個字的拼法不正確，或使用的數字錯誤，編譯器不會報告。在上一個範例中，如果列舉常數的名稱不正確，編譯器就會產生錯誤，如下列摘錄所示：

```
    if (pj.orientation == PrintJobOrientation.PORTRAI) // compiler error
```

但是，如果字串常值的拼寫錯誤，則編譯器不會產生錯誤，如下所示：

```
    if (pj.orientation == "portrai") // no compiler error
```

第二種建立列舉的技巧也包含用列舉的靜態屬性另外建立類別。這種技巧的差別在於：每一個靜態屬性都包含類別的實體，而不包含字串或整數值。例如，下列程式碼會為星期中的日期建立列舉類別：

```
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();
}
```

ActionScript 3.0 不會使用這項技巧，但很多開發人員喜歡這項技巧所提供的改良類型檢查，因此都會加以利用。例如，傳回列舉值的方法可以將傳回值限定於 enumeration 資料類型。下列程式碼不但示範會傳回星期別的函數，也示範使用列舉類型做為類型附註的函數呼叫：

```
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();
```

您也可以加強 Day 類別，讓它將整數與星期中的每一天關聯，然後提供傳回代表各天之字串的 toString() 方法。

內嵌資源類別

ActionScript 3.0 使用稱為「內嵌資源類別」的特殊類別，以代表內嵌的資源。「內嵌的資源」是在編譯階段內含於 SWF 檔中的一種資源，例如聲音、影像或字型。嵌入資源而不動態載入資源，可確保在執行階段可以使用該資源，但是會增加 SWF 檔的大小。

使用 Flash Professional 中的內嵌資源類別

若要嵌入資源，請先將資源放在 FLA 檔的元件庫中。接下來，使用資源的連結屬性來提供資源之內嵌資源類別的名稱。如果在類別路徑中找不到這個名稱的類別，便會自動產生一個類別。接著，您可以建立內嵌資源類別的實體，並使用由該類別所定義或繼承的任何屬性和方法。例如，下列程式碼可以用來播放連結到內嵌資源類別 (名為 PianoMusic) 的內嵌聲音：

```
var piano:PianoMusic = new PianoMusic();
var sndChannel:SoundChannel = piano.play();
```

或者，您可以按照下一節所述，使用 [Embed] 中繼資料標籤，將資源嵌入 Flash Professional 專案。如果在程式碼中使用 [Embed] 中繼資料標籤，Flash Professional 會使用 Flex 編譯器來編譯專案，而不會使用 Flash Professional 編譯器。

透過 Flex 編譯器來使用嵌入的資源類別

如果您使用 Flex 編譯器編譯自己的程式碼，要在 ActionScript 程式碼中嵌入一個資源，請使用 [Embed] 中繼資料標籤。將資源放在主要來源資料夾或專案組建路徑中的其他資料夾。當 Flex 編譯器遇到 Embed 中繼資料標籤時，就會為您增加內嵌資源類別。您可以透過資料類型 Class 的變數存取類別，變數必須在 [Embed] 中繼資料標籤之後立即宣告。例如，下列程式碼嵌入稱為 sound1.mp3 的聲音，然後使用 soundCls 變數，將參考儲存至與該聲音關聯的內嵌資源類別。然後，此範例再建立內嵌資源類別的實體，並在該實體上呼叫 play() 方法：

```
package
{
    import flash.display.Sprite;
    import flash.media.SoundChannel;
    import mx.core.SoundAsset;

    public class SoundAssetExample extends Sprite
    {
        [Embed(source="sound1.mp3")]
        public var soundCls:Class;

        public function SoundAssetExample()
        {
            var mySound:SoundAsset = new soundCls() as SoundAsset;
            var sndChannel:SoundChannel = mySound.play();
        }
    }
}
```

Adobe Flash Builder

若要在 Flash Builder ActionScript 專案中使用 [Embed] 中繼資料標籤，請從 Flex 架構匯入所需的任何類別。例如，若要嵌入聲音，請匯入 mx.core.SoundAsset 類別。若要使用 Flex 架構，請在 ActionScript 組建路徑中包含 framework.swc 檔案。這樣會增加 SWF 檔的大小。

Adobe Flex

或者，在 Flex 中，您也可以在 MXML 標籤定義中使用 @Embed() 指令內嵌資源。

介面

介面是方法宣告的集合，可讓不相關的物件彼此互相通訊。例如，ActionScript 3.0 可以定義 IEventDispatcher 介面，其中包含類別可用來處理事件物件的方法宣告；IEventDispatcher 介面會為物件建立標準方式，以便彼此互相傳遞事件物件。下列程式碼會示範 IEventDispatcher 介面的定義：

```
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;
}
```

介面是根據方法的介面及其實作之間的區別而來。方法的介面包含叫用該方法的所有必要資訊，包括方法的名稱、其所有參數，以及其傳回類型；方法的實作則不僅包括介面資訊，更包括可執行的陳述式，以實踐方法的行為。介面定義則只包含方法介面，以及實作介面的任何類別，此類別負責定義方法實作。

在 ActionScript 3.0 中，EventDispatcher 類別會透過定義所有 IEventDispatcher 介面的方法，並將方法主體加入各方法中，以實作 IEventDispatcher 介面。下列程式碼摘錄自 EventDispatcher 類別定義：

```
public class EventDispatcher implements IEventDispatcher
{
    function dispatchEvent(event:Event):Boolean
    {
        /* implementation statements */
    }

    ...
}
```

IEventDispatcher 介面是做為通訊協定，供 EventDispatcher 實體用來處理事件物件，並將它們傳遞給也是實作 IEventDispatcher 介面的其它物件。

介面可以用另外一種方式來描述：它像類別一樣定義資料類型，因此介面可用來做為類型附註，跟類別一樣。做為資料類型的介面也可以配合需要資料類型的運算子 (例如 is 和 as 運算子) 一併使用；但是跟類別不同的是，介面不能實體化。這項區別讓很多程式設計人員將介面視為抽象資料類型，而將類別視為具體資料類型。

定義介面

介面定義的結構類似於類別定義，只不過介面只能包含沒有方法主體的方法。介面不能包含變數或常數，但可以包含 getter 和 setter。若要定義介面，請使用 interface 關鍵字。例如，下列 IExternalizable 介面是 ActionScript 3.0 中 flash.utils 套件的一部分。IExternalizable 介面會定義序列化物件的通訊協定，也就是說，將物件轉換成適合儲存在裝置上或進行跨網路傳輸的格式。

```
public interface IExternalizable
{
    function writeExternal(output:IDataOutput):void;
    function readExternal(input:IDataInput):void;
}
```

IExternalizable 介面是以 public 存取控制修飾詞宣告。介面定義僅能由 public 和 internal 存取控制指定字加以修飾。介面定義之內的方法宣告不能有任何存取控制指定字。

ActionScript 3.0 遵循介面名稱以大寫 I 開頭的慣例，但您可以使用任何合法識別名稱做為介面名稱。介面定義經常是放在套件最上層。介面定義不能放在類別定義或其它介面定義之中。

介面可以擴充一個或多個其它介面。例如，下列介面 IExample 會擴充 IExternalizable 介面：

```
public interface IExample extends IExternalizable
{
    function extra():void;
}
```

任何實作 IExample 介面的類別都必須包含 extra() 方法，以及繼承自 IExternalizable 介面的 writeExternal() 和 readExternal() 方法之實作。

在類別中實作介面

類別是 ActionScript 3.0 中唯一能夠實作介面的語言元素，在類別宣告中使用 implements 關鍵字，便能實作一個或多個介面。下列範例會定義兩個介面，IAlpha 和 IBeta，以及一個實作這兩個介面的類別 Alpha：

```
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 {}
}
```

在實作介面的類別中，實作方法時必須具備下列條件：

- 使用 public 存取控制識別名稱。
- 使用與介面方法相同的名稱。
- 必須具有相同數目的參數，每個參數都有與介面方法參數相同的資料類型。
- 使用相同的傳回類型。

```
public function foo(param:String):String {}
```

但是對於所實作方法參數的命名方式，還是有一些彈性。雖然已實作方法中的參數數目和各參數的資料類型都必須與介面方法相符，但是參數名稱並不需要相符。例如，在上一個範例中，Alpha.foo() 方法的參數就命名為 param：

但是，在 IAlpha.foo() 介面方法中，參數是命名為 str：

```
function foo(str:String):String;
```

在預設參數值上，也有一些彈性。介面定義可以包含有預設參數值的函數宣告。實作這種函數宣告之方法的預設參數值必須與介面定義中所指定值具有相同的資料類型，但實際值並不需要相同。例如，下列程式碼會定義包含具有預設參數值 3 之方法的介面：

```
interface IGamma
{
    function doSomething(param:int = 3):void;
}
```

下列類別定義會實作 IGamma 介面，但使用不同的預設參數值：

```
class Gamma implements IGamma
{
    public function doSomething(param:int = 4):void {}
}
```

具有這種彈性的原因在於：實作介面的規則是特別為確保資料類型相容性而設計的，因而需要完全相同的參數名稱，但預設參數值並不需要達到相同的目標。

繼承

繼承是一種重複使用程式碼的形式，可以讓程式設計人員根據現有的類別開發新類別。現有的類別通常稱為 「基底類別」或「父類別」，而新類別則稱為 「子類別」。繼承的主要優點是可以讓您重複使用基底類別的程式碼，但是讓現有程式碼保持原狀不變；而且，繼承不需要變更其它類別與基底類別互動的方式。您不必修改已經過徹底測試或可能已經在使用的現有類別而使用繼承，而可以將該類別視為可用其它屬性或方法擴充的整合式模組。因此，要使用 extends 關鍵字，指出類別是從另一個類別繼承而來。

繼承也讓您在程式碼中發揮利用 「多型」。多型能夠讓方法使用單一方法名稱，而在套用至不同資料類型時，表現出不同的行為方式。簡單的範例是名為 Shape 的基底類別，具有兩個名為 Circle 和 Square 的子類別。Shape 類別會定義名為 area() 的方法，傳回形狀的面積。若已實作多型，您可以在 Circle 和 Square 類型的物件上呼叫 area() 方法，為您完成正確的計算。繼承可以透過允許子類別從基底類別繼承及重新定義或 「覆寫」方法，啟用多型。在下列範例中，area() 方法是由 Circle 和 Square 類別重新定義：

```
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
```

由於每個類別都會定義資料類型，使用繼承會在基底類別與其擴充類別之間建立特殊關係。子類別保證擁有其基底類別的所有屬性，也就是說，子類別的實體永遠都能取代基底類別的實體。例如，如果方法定義 Shape 類型的參數，傳遞 Circle 類型的引數是合法的，因為 Circle 擴充 Shape，如下所示：

```
function draw(shapeToDraw:Shape) {}

var myCircle:Circle = new Circle();
draw(myCircle);
```

實體屬性和繼承

實體屬性不管是以 function、var 或 const 關鍵字定義，只要屬性不在基底類別中以 private 特質宣告，所有子類別都會加以繼承。例如，ActionScript 3.0 中的 Event 類別有一些子類別繼承了所有事件物件都相同的繼承屬性。

在一些事件類型上，Event 類別會包含所有必要屬性，以定義事件；這些事件類型不需要定義於 Event 類別以外的實體屬性。這種事件的範例有：complete 事件 (會在順利載入資料時發生)；以及 connect 事件 (會在建立網路連線以後發生)。

下列範例摘錄自 Event 類別，其中有一些繼承自子類別的屬性和方法。由於屬性是繼承的，任何子類別的實體都可以存取這些屬性。

```
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 {}
    ...
}
```

其它事件類型需要唯一屬性，這些屬性不能在 Event 類別中使用。這些事件是使用 Event 類別之子類別定義的，因此新屬性可以加入至定義於 Event 類別中的屬性。這種子類別的範例是 MouseEvent 類別，它會加入與滑鼠動作或按滑鼠關聯之事件 (如 mouseMove 和 click 事件) 特有的屬性。下列範例摘錄自 MouseEvent 類別，其中有一些存在於子類別上但不在基底類別上的屬性定義：

```
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 {}
    ...
}
```

存取控制指定字和繼承

若屬性是以 public 關鍵字宣告，則此屬性可在程式碼任何一處看見，也就是說，public 關鍵字與 private、protected 和 internal 等關鍵字不同，並沒有對屬性繼承施加任何限制。

若屬性是以 private 關鍵字宣告，則只能在定義此屬性的類別中看見，也就是說，它不會由任何子類別繼承的。這種行為方式與舊版 ActionScript 不同，舊版 private 關鍵字的行為比較像 ActionScript 3.0 的 protected 關鍵字。

protected 關鍵字表示屬性不但能在定義的類別中看見，也能在所有子類別中看見。跟 Java 程式設計語言中的 protected 關鍵字不同，ActionScript 3.0 中的 protected 關鍵字不會讓屬性供相同套件中的所有其它類別看見；在 ActionScript 3.0 中，只有子類別可以存取用 protected 關鍵字宣告的屬性。而且，不管子類別是在與基底類別相同的套件或是不同的套件中，保護的屬性都可讓子類別看見。

若要限制屬性在定義該屬性的套件中之可見性，請使用 internal 關鍵字，或者不要使用任何存取控制指定字。internal 存取控制指定字是在未指定時所套用的預設存取控制指定字。相同套件中的子類別才能繼承標記為 internal 的屬性。

您可以使用下列範例，查看每一個存取控制指定字如何跨越套件界限影響繼承。下列程式碼定義名為 AccessControl 的主應用程式類別，以及另外兩個名為 Base 和 Extender 的類別：Base 類別在名為 foo 的套件中，而 Extender 類別是 Base 類別的子類別，在名為 bar 的套件中。AccessControl 類別僅匯入 Extender 類別，並建立 Extender 的實體，嘗試存取在 Base 類別中定義的 str 變數；str 變數是宣告為 public，因此程式碼編譯及執行如下列摘錄中所示：

```
// Base.as in a folder named foo
package foo
{
    public class Base
    {
        public var str:String = "hello"; // change public on this line
    }
}

// Extender.as in a folder named bar
package bar
{
    import foo.Base;
    public class Extender extends Base
    {
        public function getString():String {
            return str;
        }
    }
}

// main application class in file named AccessControl.as
package
{
    import flash.display.MovieClip;
    import bar.Extender;
    public class AccessControl extends MovieClip
    {
        public function AccessControl()
        {
            var myExt:Extender = new Extender();
            trace(myExt.str);// error if str is not public
            trace(myExt.getString()); // error if str is private or internal
        }
    }
}
```

若要查看其它存取控制指定字如何影響上一個範例的編譯和執行，請在刪除下列 AccessControl 類別這行或將它註解化之後，將 str 變數的存取控制指定字變更為 private、protected 或 internal：

```
trace(myExt.str);// error if str is not public
```

不允許覆寫變數

以 var 或 const 關鍵字宣告的屬性可以繼承，但不能加以覆寫。所謂覆寫屬性，就是要在子類別中重新定義屬性。唯一可以覆寫的屬性類型，是取得及設定存取子 (使用 function 關鍵字宣告的屬性)。雖然您不能覆寫實體變數，但是您可以透過為實體變數建立 getter 和 setter 方法，然後覆寫方法，達到類似的功能。

覆寫方法

所謂覆寫方法，就是要重新定義繼承的方法之行為方式。靜態方法不是繼承的，所以不能覆寫。但是實體方法是由子類別加以繼承，只要符合下列兩項條件，即可進行覆寫：

- 實體方法不用 final 關鍵字在基底類別中宣告。final 關鍵字配合實體方法使用時，表示程式設計人員意圖阻止子類別覆寫方法。
- 實體方法不會使用 private 存取控制指定字在基底類別中宣告。如果方法在基底類別中標記為 private，就不需要在子類別定義名稱完全相同的方法時，使用 override 關鍵字，因為子類別無法看見基底類別方法。

  若要覆寫符合這些條件的實體方法，子類別中的方法定義必須使用 override 關鍵字，也必須在下列各方面與方法的父類別版本相符：

- 覆寫方法必須具有與基底類別方法相同的存取控制層級。標示為 internal 的方法必須具有與無存取控制指定字的方法相同之存取控制層級。
- 覆寫方法必須具有與基底類別方法相同的參數數目。
- 覆寫方法參數必須具有與基底類別方法中參數相同的資料類型。
- 覆寫方法必須具有與基底類別方法相同的傳回類型。

但是，在覆寫方法中的參數名稱不必與基底類別中參數名稱相同，只要各參數的參數數目及資料類型相符即可。

super 陳述式

覆寫方法時，程式設計人員經常是要新增所要覆寫父類別方法的行為方式，而不是完全取代其行為方式。這需要一種機制，可以讓子類別中的方法呼叫本身的父類別版，super 陳述式就會提供這種機制，因此其中包含直屬父類別的參考。下列範例會定義名為 Base 的類別，其中包含名為 thanks() 的方法，以及名為 Extender 而覆寫 thanks() 方法的 Base 類別之子類別。Extender.thanks() 方法會使用 super 陳述式，呼叫 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";
    }
}
```

覆寫 getter 和 setter

雖然您不能覆寫定義於父類別中的變數，但卻可以覆寫 getter 和 setter。例如，下列程式碼會覆寫名為 currentLabel 的 getter，它定義於 ActionScript 3.0 的 MovieClip 類別中：

```
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;
        }
    }
}
```

OverrideExample 類別建構函式中 trace() 陳述式的輸出是 Override: null，顯示出此範例能夠覆寫繼承的 currentLabel 屬性。

不繼承的靜態屬性

靜態屬性無法由子類別加以繼承，也就是說，靜態屬性不能透過子類別的實體進行存取；唯有透過定義該屬性的類別物件，才能存取靜態屬性。例如，下列程式碼會定義名為 Base 的基底類別，以及擴充 Base 而名為 Extender 的子類別；名為 test 的靜態變數是定義於 Base 類別中。如下列摘錄中所撰寫的程式碼在嚴謹模式中不會進行編譯，而在標準模式中會產生執行階段錯誤。

```
package {
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
            trace(myExt.test);// error
        }
    }
}

class Base {
    public static var test:String = "static";
}

class Extender extends Base { }
```

存取靜態變數 test 的唯一方法是透過類別物件存取，如下列程式碼所示：

```
Base.test;
```

但使用與靜態屬性相同的名稱來定義實體屬性是受到允許的，這種實體屬性可以在與靜態屬性相同的類別中，或是子類別中進行定義。例如，上一個範例中的 Base 類別可以有名為 test 的實體屬性。下列程式碼會編譯並執行，因為實體屬性是由 Extender 類別加以繼承。若將 test 實體變數的定義移至 (而不是複製至) Extender 類別，則此程式碼也會編譯並執行。

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
            trace(myExt.test);// output: instance
        }
    }
}

class Base
{
    public static var test:String = "static";
    public var test:String = "instance";
}

class Extender extends Base {}
```

靜態屬性與範圍鏈

雖然靜態屬性不繼承，但它們是在定義該屬性之類別及該類別之任何子類別的範圍鏈內，因此靜態屬性是同時位於其定義類別及任何子類別的「範圍中」。也就是說，靜態屬性可以在定義靜態屬性的類別主體及其任何子類別之內直接存取。

下列範例修改了上一個範例中所定義的類別，以示範定義於 Base 類別中的靜態 test 變數是在 Extender 類別的範圍中。換句話說，Extender 類別可以存取靜態 test 變數，而不需要以定義 test 的類別名稱做為變數的前置詞。

```
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
    }

}
```

若定義實體屬性時，所使用名稱與同類別或父類別中的靜態屬性名稱相同，則實體屬性在範圍鏈中具有較高優先順序。實體屬性即稱為「遮蔽」靜態屬性，也就是說，使用了實體屬性的值，而不是使用靜態屬性的值。如下列範例所示範，若 Extender 類別定義名為 test 的實體變數，則 trace() 陳述式會使用實體變數的值，而不使用靜態變數的值。

```
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: instance
    }

}
```

進階主題

ActionScript 支援 OOP 的歷史背景

由於 ActionScript 3.0 是以舊版 ActionScript 為基礎所建立，瞭解 ActionScript 物件模型的演進情況可能會有幫助。ActionScript 最初是針對早期 Flash Professional 版本提供的簡單 Script 撰寫機制。後來，程式設計人員開始使用 ActionScript 建立日益複雜的應用程式。為了回應這些程式設計人員的需求，每一個後續版本都會有新增的語言功能，以便協助建立複雜的應用程式。

ActionScript 1.0

ActionScript 1.0 是用於 Flash Player 6 及更早版本中的語言版本。即使在這麼早期的開發階段，ActionScript 物件模型就是建立在以物件做為基礎資料類型的概念上。ActionScript 物件是具有一組 「屬性」的複合資料類型；討論物件模型時，「屬性」一詞包含附加至物件的一切項目，如變數、函數或方法。

雖然這第一代 ActionScript 並不支援使用 class 關鍵字的類別定義，但可以使用稱為原型 (Prototype) 物件的特殊物件定義類別。ActionScript 1.0 原型語言跟 Java 和 C++ 類別語言不同：後兩者都使用 class 關鍵字建立抽象類別定義，以便實體化成具體物件，而 ActionScript 1.0 則是使用現有物件做為其它物件的模型 (或原型)。類別語言中的物件可以指向類別做為其範本，而原型語言中的物件卻是指向其它物件 (即其原型) 做為其範本。

若要在 ActionScript 1.0 中建立類別，必須為該類別定義建構函數。在 ActionScript 中，函數實際上就是物件，而不只是抽象定義。您所建立的建構函數是做為該類別之實體的原型物件。下列程式碼會建立名為 Shape 的類別，並定義一個名為 visible 的屬性，該屬性是預設為 true：

```
// base class
function Shape() {}
// Create a property named visible.
Shape.prototype.visible = true;
```

此建構函數會定義您可以用 new 運算子實體化的 Shape 類別，如下所示：

```
myShape = new Shape();
```

就像 Shape() 建構函數物件可以做為 Shape 類別的實體原型，它也可以做為 Shape 的子類別 (也就是擴充 Shape 類別的其它類別) 之原型。

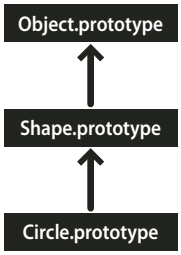
建立 Shape 類別的子類別要執行兩步驟程序。首先，定義類別的建構函數以建立該類別，如下所示：

```
// child class
function Circle(id, radius)
{
this.id = id;
this.radius = radius;
}
```

接下來，使用 new 運算子，以宣告 Shape 類別為 Circle 類別的原型。根據預設，您所建立的任何類別都會使用 Object 類別做為其原型，也就是說，Circle.prototype 目前包含一般物件 (Object 類別的實體)。若要指定 Circle 的原型為 Shape 而不是 Object，請使用下列程式碼變更 Circle.prototype 的值，讓它包含 Shape 物件，而不是包含一般物件：

```
// Make Circle a subclass of Shape.
Circle.prototype = new Shape();
```

Shape 類別和 Circle 類別現在會以繼承關係連結成一體，這關係一般稱為 「原型鏈」。下圖可說明原型鏈中的關係：



每一個原型鏈尾端的基底類別都是 Object 類別。Object 類別包含名為 Object.prototype 的靜態屬性，它指向 ActionScript 1.0 建立之所有物件的基底原型物件。範例原型鏈中的下一個物件是 Shape 物件。這是因為 Shape.prototype 屬性從未明確地設定，因此它仍然保存一般物件 (Object 類別的實體)。這個鏈中最後一個連結是 Circle 類別，它是連結至其原型 - Shape 類別 (Circle.prototype 原型保存了 Shape 物件)。

如果您建立 Circle 類別的實體 (如同下列範例一樣)，此實體會繼承 Circle 類別的原型鏈：

```
// Create an instance of the Circle class.
myCircle = new Circle();
```

不知您是否還記得，範例包含名為 visible 的屬性做為 Shape 類別的成員。在這個範例中，visible 屬性不是 myCircle 物件的一部分，而只是做為 Shape 物件的成員，然而下列程式碼行卻輸出 true：

```
trace(myCircle.visible); // output: true
```

執行階段只要順著原型鏈往上走，就能查明是 myCircle 物件繼承了 visible 屬性。執行此程式碼時，執行階段會先搜尋 myCircle 物件的所有屬性，尋找名為 visible 的屬性，但找不到這個屬性。接著，它就在 Circle.prototype 物件中尋找，但還是找不到名為 visible 的屬性。繼續順著原型鏈往上走，它終於找到在 Shape.prototype 物件上定義的 visible 屬性，並輸出該屬性的值。

由於想要盡可能地簡單明瞭，因此本節內容省略了原型鏈的許多細節和錯綜複雜的關係。而是將重點放在提供足夠資訊，希望能協助您瞭解 ActionScript 3.0 物件模型。

ActionScript 2.0

ActionScript 2.0 導入新的關鍵字，如 class、extends、public 和 private，可讓您以使用 Java 和 C++ 等類別語言者所熟悉的方式來定義類別。請務必要了解，基礎繼承機制在 ActionScript 1.0 與 ActionScript 2.0 之間並沒有任何改變，ActionScript 2.0 只是加了新語法，可供定義類別而已。原型鏈在這兩版語言中的運作方式完全一樣。

由 ActionScript 2.0 導入的新語法 (如下列摘錄所示)，讓您能夠以許多程式設計人員覺得更為直覺的方式來定義類別：

```
// base class
class Shape
{
var visible:Boolean = true;
}
```

請注意，ActionScript 2.0 也導入類型附註，可用來進行編譯階段類型檢查。如此一來，您便可以宣告上一個範例中的 visible 屬性應該只包含 Boolean 值；新的 extends 關鍵字也簡化了建立子類別的程序。在下列範例中，ActionScript 1.0 中必須要有兩個步驟的程序，而使用 extends 關鍵字只需一個步驟就可完成：

```
// child class
class Circle extends Shape
{
    var id:Number;
    var radius:Number;
    function Circle(id, radius)
    {
        this.id = id;
        this.radius = radius;
        }
}
```

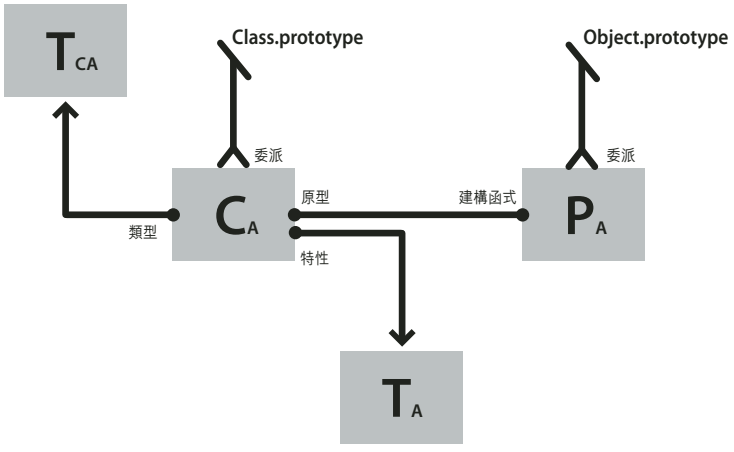
建構函式現在是宣告為類別定義的一部分，而類別屬性 id 和 radius 都必須明確地進行宣告。

ActionScript 2.0 也新增介面定義的支援，可讓您利用正式為物件間通訊定義的通訊協定，更進一步修改您的物件導向程式。

ActionScript 3.0 類別物件

常見的物件導向程式設計範式通常是與 Java 和 C++ 相關聯，會使用類別來定義物件的類型。採用這種範式的程式設計語言也傾向於使用類別來建構類別所定義資料類型的實體。ActionScript 同時將類別用於這兩種用途，不僅紮根於原型語言，更增添了值得玩味的特性。ActionScript 為每一個定義建立特殊的類別物件，而允許同時共用行為和狀態。但是對許多 ActionScript 程式設計人員來說，這項區別沒有實際的程式含意。ActionScript 3.0 的設計方式能讓您建立更複雜的物件導向 ActionScript 應用程式，而不需要使用 (甚至也不需要瞭解) 這些特殊類別物件。

下圖會顯示類別物件的結構，它代表名為 A 而用陳述式 class A {} 定義的簡單類別：



圖中的每一個矩形都代表一個物件。圖中每一個物件都有下標字元 A，以代表屬於類別 A，類別物件 (CA) 包含其它重要物件的數目參考。實體特性物件 (TA) 會儲存在類別定義之內定義的實體屬性。類別特性物件 (TCA) 代表類別的內部類型，並儲存由類別定義的靜態屬性 (下標字元 C 代表 「類別」)。原型物件 (PA) 永遠都是指原先透過 constructor 屬性附加其上的類別物件。

特性物件

特性物件是 ActionScript 3.0 中的新增物件，實作時會以效能為考量。在舊版 ActionScript 中，名稱查閱程序可能會相當耗費時間，因為 Flash Player 會順著原型鏈查詢；在 ActionScript 3.0 中，名稱查閱會更有效率，所耗費時間較少，因為繼承的屬性是從父類別往下複製，一直到子類別的特性物件。

特性物件不能直接由程式設計人員編寫的程式碼存取，但可從效能改善及記憶體使用情形感覺到它的存在。特性物件提供 AVM2 有關類別版面及內容的詳細資訊。掌握了這些資訊之後，AVM2 就能大幅減少執行時間，因為它可以經常直接由機器產生指示以存取屬性，或是直接呼叫方法，而不需進行耗費時間的名稱查閱。

使用了特性物件的話，一個物件的記憶體使用量便會比舊版 ActionScript 中的類似物件減少相當多。例如，若是類別被密封 (也就是，類別不是動態宣告)，類別的實體就不需要供動態加入屬性使用的雜湊表，而且所包含的只是特性物件的指標加上一些類別中所定義固定屬性的空間位置而已。其結果就是，在 ActionScript 2.0 中需要 100 位元組記憶體的物件，在 ActionScript 3.0 中可能只需要 20 位元組左右的記憶體就夠了。

備註：特性物件是內部實作詳細資訊，並不保證在將來新版 ActionScript 中不會改變或甚至完全消失。

原型物件

每個 ActionScript 類別物件都有名為 prototype 的屬性，是類別之原型物件的參考，原型物件是舊版 ActionScript 原型語言根源留下的產物。如需詳細資訊，請參閱 ActionScript 支援 OOP 的歷史背景。

prototype 屬性是唯讀屬性，也就是不能進行修改以另外指向不同的物件。這與舊版 ActionScript 中的類別 prototype 屬性不同，在舊版中，原型可以重新指定，以指向不同的類別。雖然 prototype 屬性是唯讀性質，但所參考原型物件並不是唯讀的，換句話說，新的屬性可以加入至原型物件中，加入原型物件的屬性可以供類別中所有實體共用。

原型鏈是舊版 ActionScript 中的唯一繼承機制，但在 ActionScript 3.0 中則只是扮演了次要的角色，主要的繼承機制為固定的屬性繼承，是由特性物件在內部進行處理。固定的屬性是定義為類別定義一部分的變數或方法。固定的屬性繼承也稱為類別繼承，因為它是與關鍵字 class、extends 和 override 等關聯的繼承機制。

原型鏈提供另外一種形式的繼承機制，比固定的屬性繼承更具彈性。不但可以將屬性做為類別定義的一部分加入至類別的原型物件，也可以在執行階段透過類別物件的 prototype 屬性加入。但請注意，如果將編譯器設定為嚴謹模式，可能就無法存取加入至原型物件的屬性，除非以 dynamic 關鍵字進行宣告。

Object 類別是有某些屬性附加至原型物件之類別的好範例。Object 類別的 toString() 和 valueOf() 方法其實都是指定給 Object 類別之原型物件屬性的函數。下列範例示範這些方法的宣告在理論上的情況 (實際實作會因為實作細節的關係而稍微有些差異)：

```
public dynamic class Object
{
    prototype.toString = function()
    {
        // statements
    };
    prototype.valueOf = function()
    {
        // statements
    };
}
```

上文已經討論過，您可以在類別定義之外，將屬性附加至類別的原型物件。例如，toString() 方法也可以在 Object 類別定義之外定義，如下所示：

```
Object.prototype.toString = function()
{
    // statements
};
```

但是，原型繼承與固定的屬性繼承不同，若要在子類別中重新定義方法，並不需要 override 關鍵字。例如：若要在 Object 類別的子類別中重新定義 valueOf() 方法，您有三種選擇：第一，您可以在類別定義中的子類別之原型物件上定義 valueOf() 方法。下列程式碼會建立名為 Foo 的 Object 之子類別，然後在 Foo 的原型物件上重新定義 valueOf() 方法，做為類別定義的一部分。由於每一個物件都是繼承自 Object，所以並不需要使用 extends 關鍵字。

```
dynamic class Foo
{
    prototype.valueOf = function()
    {
        return "Instance of Foo";
    };
}
```

第二，您可以在類別定義之外，於 Foo 的原型物件上定義 valueOf() 方法，如下列程式碼所示：

```
Foo.prototype.valueOf = function()
{
    return "Instance of Foo";
};
```

第三，您可以定義名為 valueOf() 的固定屬性做為 Foo 類別的一部分。這種技巧與其它方法都不同，它混合了固定的屬性繼承與原型繼承。任何要重新定義 valueOf() 的 Foo 子類別都必須使用 override 關鍵字。下列程式碼示範如何將 valueOf() 定義為 Foo 中的固定屬性：

```
class Foo
{
    function valueOf():String
    {
        return "Instance of Foo";
    }
}
```

AS3 命名空間

兩種不同繼承機制 (固定的屬性繼承和原型繼承) 的存在，造就了在核心類別的屬性及方法方面的相容性挑戰。與 ECMAScript 語言規格 (ActionScript 的撰寫依據) 的相容性需要使用原型繼承，也就是說，核心類別的屬性和方法都是在該類別的原型物件上定義。另一方面，與 ActionScript 3.0 的相容性則需要使用固定的屬性繼承，也就是說，核心類別的屬性和方法是使用 const、var 和 function 等關鍵字在類別定義中定義。而且，使用固定的屬性而不使用原型版本，可能會讓執行階段效能大幅提升。

ActionScript 3.0 是透過同時使用核心類別的原型繼承和固定屬性繼承，來解決這個問題。每一個核心類別都包含兩組屬性和方法：一組是在原型物件上為 ECMAScript 規格之相容性所定義，另一組則是以固定的屬性和 AS3 命名空間，為 ActionScript 3.0 之相容性所定義。

AS3 命名空間提供在兩組屬性和方法之間選擇的便利機制。若不使用 AS3 命名空間，核心類別的實體會繼承在核心類別之原型物件上定義的屬性和方法。若決定使用 AS3 命名空間，則核心類別的實體會繼承 AS3 版，因為固定的屬性總是比原型屬性優先繼承。換句話說，只要可以使用固定的屬性，一定會使用此屬性，而不使用名稱完全相同的原型屬性。

您可以透過用 AS3 命名空間加以限定，選擇性地使用 AS3 命名空間版屬性或方法。例如，下列程式碼會使用 AS3 版 Array.pop() 方法：

```
var nums:Array = new Array(1, 2, 3);
nums.AS3::pop();
trace(nums); // output: 1,2
```

另外，您也可以使用 use namespace 指令，開啟在程式碼區塊之內所有定義的 AS3 命名空間。例如，下列程式碼會使用 use namespace 指令，同時開啟 pop() 和 push() 方法的 AS3 命名空間：

```
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 也為各組屬性提供編譯器選項，以便讓您將 AS3 命名空間套用至整個程式。-as3 編譯器選項代表 AS3 命名空間，而 -es 編譯器選項則代表原型繼承選項 (es 代表 ECMAScript)。若要為整個程式開啟 AS3 命名空間，請將 -as3 編譯器選項設定為 true，而將 -es 編譯器選項設定為 false。若要使用原型版本，請將編譯器選項設定為相反的值。Flash Builder 和 Flash Professional 的預設編譯器設定是 -as3 = true 和 -es = false。

若計劃擴充任何核心類別及覆寫任何方法，應該要瞭解 AS3 命名空間會如何影響您必須宣告被覆寫方法的方式。若要使用 AS3 命名空間，核心類別的任何方法覆寫也必須使用 AS3 命名空間，以及 override 特質。若不要使用 AS3 命名空間，而要在子類別中重新定義核心類別方法，就不應該使用 AS3 命名空間或 override 關鍵字。

範例：GeometricShapes

GeometricShapes 樣本應用程式會示範一些物件導向的概念，及可以使用 ActionScript 3.0 套用的功能，包括：

- 定義類別
- 擴充類別
- 多型和 override 關鍵字
- 定義、擴充及實作介面

範例中也包括建立類別實體的 「原廠方法」，示範如何宣告傳回值做為介面的實體，然後以一般方式使用該傳回的物件。

若要取得此樣本的應用程式檔案，請參閱 www.adobe.com/go/learn_programmingAS3samples_flash_tw。GeometricShapes 應用程式檔案可以在 Samples/GeometricShapes 資料夾中找到。此應用程式是由下列檔案組成：

檔案	說明
GeometricShapes.mxml 或 GeometricShapes.fla	主應用程式檔案，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/geometricshapes/IGeometricShape.as	要由所有 GeometricShapes 應用程式類別實作的基底介面定義方法。
com/example/programmingas3/geometricshapes/IPolygon.as	要由具有多邊的 GeometricShapes 應用程式類別實作的介面定義方法。
com/example/programmingas3/geometricshapes/RegularPolygon.as	一種幾何圖形，將等長的邊對稱地放置在圖形中心的四周。
com/example/programmingas3/geometricshapes/Circle.as	定義圓形的一種幾何圖形。
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	定義等邊三角形的 RegularPolygon 子類別。
com/example/programmingas3/geometricshapes/Square.as	定義四邊全都等長之正方形的 RegularPolygon 子類別。
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	包含 「原廠方法」的類別，該方法可以用來建立具有指定形狀類型及大小的形狀。

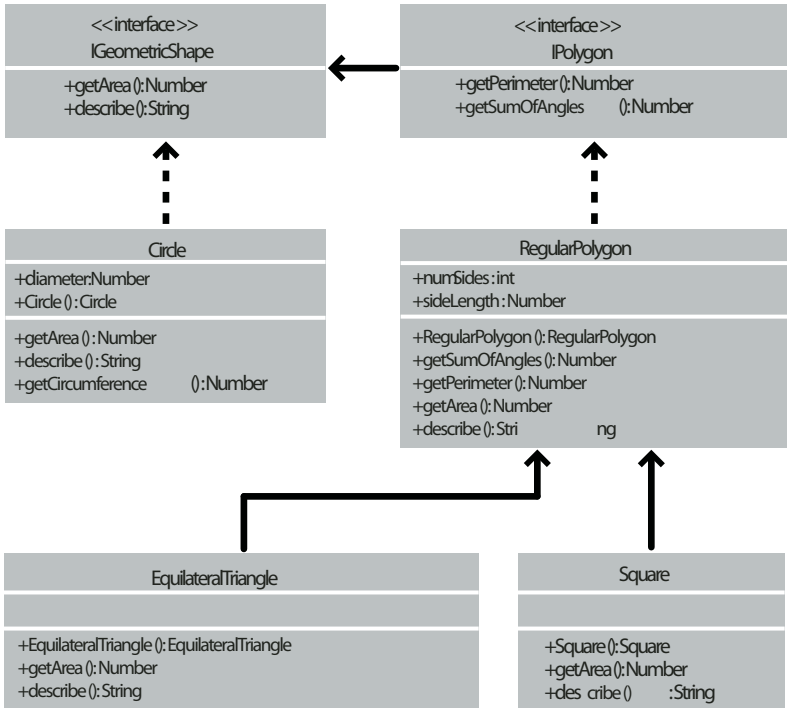
定義 GeometricShapes 類別

GeometricShapes 應用程式可以讓使用者指定一種幾何圖形和大小，然後以圖形的描述、面積及周圍距離來回應。

應用程式使用者介面 (包括選取形狀類型、設定大小，及顯示描述等少數控制項) 並不重要。這個應用程式最有趣的部分是在其核心，也就是其類別和介面本身的結構。

這個應用程式會處理幾何圖形，但卻不會以圖像顯示圖形。

在這個範例中用來定義幾何圖形的類別和介面是使用統一模組化語言 (UML) 標記法，以下面圖解顯示：



GeometricShapes Example 類別

以介面定義共通的行為

這個 GeometricShapes 應用程式處理三種形狀：圓形、方形和等邊三角形。GeometricShapes 類別結構以非常簡單的介面 IGeometricShape 開始，當中列出三種形狀都有的方法：

```
package com.example.programmingas3.geometricshapes
{
    public interface IGeometricShape
    {
        function getArea():Number;
        function describe():String;
    }
}
```

此介面定義兩個方法：getArea() 方法會計算並傳回圖形面積，而 describe() 方法會組合描述形狀屬性的文字。

我們也想知道各圖形周圍的距離；但是，稱為圓周的圓形周圍會以獨特的方式計算，因此行為跟三角形或方形很不同。不過，三角形、方形及其它多邊形還是有足夠的相似性，因此值得特別為它們定義新介面類別：IPolygon。IPolygon 介面也相當簡單，如下所示：

```
package com.example.programmingas3.geometricshapes
{
    public interface IPolygon extends IGeometricShape
    {
        function getPerimeter():Number;
        function getSumOfAngles():Number;
    }
}
```

這個介面定義所有多邊形都有的兩個方法：getPerimeter() 方法會測量所有各邊加起來的距離，而 getSumOfAngles() 方法會將所有內角相加。

IPolygon 介面會擴充 IGeometricShape 介面，也就是說，任何實作 IPolygon 介面的類別都必須宣告四個方法：兩個來自 IGeometricShape 介面，兩個來自 IPolygon 介面。

定義形狀類別

一旦掌握了各種形狀都有的共通方法，就可以定義形狀類別本身了。就必須實作的方法數目而言，最簡單的形狀是 Circle 類別，如下所示：

```
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;
        }
    }
}
```

Circle 類別會實作 IGeometricShape 介面，因此它必須同時為 getArea() 方法和 describe() 方法提供程式碼。此外，它也定義 getCircumference() 方法，這是 Circle 類別特有的方法；Circle 類別也宣告屬性 diameter，也是其它多邊形類別中不會有的屬性。

另外兩種形狀 - 方形和等邊三角形，卻另外有一些共通特性：它們都各有等長的邊，而且兩者都有共通的公式可用來計算周長和內角總和。事實上，這些共通的公式也適用於以後您會定義的任何其它正多邊形。

RegularPolygon 類別同時是 Square 類別和 EquilateralTriangle 類別的父類別。父類別可以讓您在一處定義共通的方法，而不必分別在各個子類別中定義。下面是 RegularPolygon 類別的程式碼：

```
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;
        }
    }
}
```

首先，RegularPolygon 類別宣告所有正多邊形都有的兩個屬性：各邊的長度 (sideLength 屬性) 和邊的數目 (numSides 屬性)。

RegularPolygon 類別實作 IPolygon 介面，然後將 IPolygon 介面的四個方法都一起宣告。它使用共通的公式，實作其中兩個方法：getPerimeter() 和 getSumOfAngles() 方法。

由於 getArea() 方法的公式，每一種形狀都不同，因此方法的基底類別版本不能包括子類別方法可繼承的共通邏輯。而只是傳回 0 預設值，指出並未計算面積。若要正確計算各圖形的面積，RegularPolygon 類別的子類別便需要覆寫 getArea() 方法本身。

下列 EquilateralTriangle 類別的程式碼示範 getArea() 方法是如何覆寫的：

```
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;
        }
    }
}
```

override 關鍵字指出 EquilateralTriangle.getArea() 方法會刻意從 RegularPolygon 父類別覆寫 getArea() 方法。呼叫 EquilateralTriangle.getArea() 方法時，它會使用上一個程式碼中的公式計算面積，而 RegularPolygon.getArea() 方法中的程式碼則永遠不會執行。

對照之下，EquilateralTriangle 類別不會定義其本身版本的 getPerimeter() 方法。呼叫 EquilateralTriangle.getPerimeter() 方法時，呼叫會順著繼承鏈往上，並執行 RegularPolygon 父類別之 getPerimeter() 方法中的程式碼。

EquilateralTriangle() 建構函式則會使用 super() 陳述式，以明確叫用其父類別的 RegularPolygon() 建構函式。若兩個建構函式都有相同的一組參數，您可以完全省略 EquilateralTriangle() 建構函式，而改為執行 RegularPolygon() 建構函式。但是 RegularPolygon() 建構函式需要額外的參數 numSides。因此 EquilateralTriangle() 建構函式呼叫 super(len,)，跟 len 輸入參數和值 3 一起傳遞，指出三角形會有 3 個邊。

describe() 方法也使用 super() 敘述式，不過方式不同。這種方式會使用這個敘述句來呼叫 describe() 方法的 RegularPolygon 父類別版本。EquilateralTriangle.describe() 方法先將 desc 字串變數設定至有關形狀類型的陳述式。然後，它會呼叫 super.describe()，以取得 RegularPolygon.describe() 方法的結果，而且將該結果附加至 desc 字串。

這裡不會詳細描述 Square 類別，但它與 EquilateralTriangle 類別類似，可提供建構函式及其本身 getArea() 和 describe() 方法的實作。

多型和原廠方法

一組發揮運用介面和繼承的類別，可以透過許多有趣的方式使用。例如，到目前為止所描述的全部形狀類別，不是實作 IGeometricShape 介面，就是擴充執行實作的父類別。因此，若要將變數定義為 IGeometricShape 的實體，不必知道它實際上是 Circle 或是 Square 類別的實體，就能呼叫其 describe() 方法。

下列程式碼會示範這種運作方式：

```
var myShape:IGeometricShape = new Circle(100);
trace(myShape.describe());
```

呼叫 myShape.describe() 時，它會執行 Circle.describe() 方法，因為雖然變數是定義為 IGeometricShape 介面的實體，但 Circle 是其基礎類別。

這個範例示範了運作中的多型原則：完全相同的方法呼叫會導致執行不同的程式碼，依被叫用方法的物件類別而不同。

GeometricShapes 應用程式使用簡化版設計模式，稱為原廠方法，來套用這種介面多型。「原廠方法」一詞是指傳回物件的函數，其基礎資料類型或內容會依背景內容而不同。

這裡示範的 GeometricShapeFactory 類別會定義名為 createShape() 的原廠方法：

```
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();
        }
    }
}
```

createShape() 原廠方法可以讓形狀子類別建構函式定義所建立實體的詳細資訊，而同時傳回新的物件做為 IGeometricShape 實體，以便由應用程式以更普遍的方式加以處理。

上一個範例中的 describeShape() 方法示範了應用程式如何使用原廠方法以更精確地取得特定物件的一般性參考。應用程式可以取得新建立 Circle 物件的描述，如下所示：

```
GeometricShapeFactory.describeShape("Circle", 100);
```

describeShape() 方法會接著以相同的參數呼叫 createShape() 原廠方法，將新的 Circle 物件儲存在名為 currentShape 的靜態變數中，該變數則轉換類型成為 IGeometricShape 物件。接下來，在 currentShape 物件上呼叫 describe() 方法，而該呼叫會自動解析，以執行 Circle.describe() 方法，傳回圓形的詳細描述。

增強樣本應用程式

介面和繼承的真正功能會在您增強或變更應用程式時真正顯現出來。

假設您要將新形狀「五邊形」，加入至此樣本應用程式。您要建立擴充 RegularPolygon 類別的 Pentagon 類別，然後定義其自身版本的 getArea() 和 describe() 方法。然後，再將新的 Pentagon 選項加入應用程式使用者介面中的下拉式清單方塊。這樣一來，整個作業就完成了。Pentagon 類別會透過繼承，自動從 RegularPolygon 類別取得 getPerimeter() 方法和

getSumOfAngles() 方法的功能。因為 Pentagon 實體會從實作 IGeometricShape 介面的類別繼承，所以也可被視為 IGeometricShape 實體；也就是說，若要加入新的形狀類型，不必變更任何 GeometricShapeFactory 中方法的方法使用方式 (因此，您也不必變更任何使用 GeometricShapeFactory 類別的程式碼)。

您可以練習在此「幾何圖形」範例中新增 Pentagon (五角形) 類別，以真正瞭解介面和繼承如何能減輕在應用程式中增加新功能的工作負擔。