ACTIONSCRIPT® 3.0

開發人員指南

上次更新 **2015/10/14**

法律聲明

如需法律聲明，請參閱 http://help.adobe.com/zh_TW/legalnotices/index.html。

內容

第 1 章 使用日期與時間

管理月曆日期和時間 1

控制時間間隔 3

日期與時間範例：簡易類比時鐘 5

第 2 章 使用字串

字串的基本概念 9

建立字串 10

length 屬性 11

使用字串中的字元 11

比較字串 12

取得其它物件的字串形式 12

連接字串 12

尋找字串中的子字串和樣式 13

轉換字串大小寫 16

字串範例：ASCII 藝術 17

第 3 章 使用陣列

陣列的基本概念 21

索引陣列 22

關聯陣列 31

多維度陣列 34

複製陣列 35

擴充 Array 類別 36

陣列範例：PlayList 39

第 4 章 處理錯誤

錯誤處理的基本概念 44

錯誤類型 45

ActionScript 3.0 中的錯誤處理程序 46

使用 Flash 執行階段的除錯程式版本 48

處理應用程式中的同步錯誤 48

建立自訂的錯誤類別 52

對錯誤事件和狀態做出回應 53

比較各種 Error 類別 56

處理錯誤範例：CustomErrors 應用程式 58

第 5 章 使用規則運算式

規則運算式的基本概念 64

規則運算式語法 65

搭配字串使用規則運算式的方法 76
一般表示式範例：Wiki 剖析器 77

第 6 章 使用 XML
XML 的基本概念 81
用來處理 XML 的 E4X 83
XML 物件 85
XMLList 物件 87
初始化 XML 變數 88
組合及轉換 XML 物件 89
移動 XML 結構 91
使用 XML 命名空間 94
XML 類型轉換 95
讀取外部 XML 文件 97
ActionScript 中的 XML 範例：從網際網路載入 RSS 資料 97

第 7 章 使用原生 JSON 功能
JSON API 概觀 101
定義自訂 JSON 行為 102

第 8 章 處理事件
處理事件的基本概念 108
ActionScript 3.0 與舊版在事件處理方面的差異 109
事件流程 111
事件物件 112
事件偵聽程式 115
事件處理範例：鬧鐘 120

第 9 章 使用應用程式網域

第 10 章 顯示程式設計
顯示程式設計的基本概念 128
核心顯示類別 130
採用顯示清單的優點 132
處理顯示物件 134
操作顯示物件 146
將物件製成動畫 162
舞台方向 163
以動態方式載入顯示內容 167
顯示物件範例：SpriteArranger 171

第 11 章 處理幾何
幾何的基本概念 177
使用 Point 物件 178

使用 Rectangle 物件 179
使用 Matrix 物件 182
幾何範例：將矩陣變形套用至顯示物件 183

第 12 章 使用繪圖 API
繪圖 API 的基本概念 187
Graphics 類別 188
繪製線段和曲線 188
使用內建方法來繪製形狀 190
建立漸層線段與填色 191
搭配繪圖方法使用 Math 類別 195
使用繪圖 API 來建立動畫 195
繪製 API 範例：演算法視覺產生器 196
繪圖 API 的進階用法 198

第 13 章 使用點陣圖
使用點陣圖的基本概念 205
Bitmap 與 BitmapData 類別 207
操作像素 208
複製點陣圖資料 210
壓縮點陣圖資料 211
使用雜訊函數製作紋理 212
捲動點陣圖 213
善用 MIP 映射的優點 214
點陣圖範例：動畫轉動的月球 215
點陣圖影像的非同步解碼 223

第 14 章 以濾鏡處理顯示物件
以濾鏡處理顯示物件的基本概念 226
建立與套用濾鏡 226
可用的顯示濾鏡 232
濾鏡處理顯示物件範例：濾鏡工作台 246

第 15 章 使用 Pixel Bender 著色器
Pixel Bender 著色的基本概念 253
載入或嵌入著色器 255
存取著色器中繼資料 256
指定著色器輸入和參數值 256
使用著色器 261

第 16 章 使用影片片段
影片片段的基本觀念 272
使用 MovieClip 物件 272
控制影片片段播放作業 273

使用 ActionScript 建立 MovieClip 物件 275
載入外部 SWF 檔 277
移動影片片段範例：RuntimeAssetsExplorer 279

第 17 章 使用移動補間動畫
移動補間動畫的基本概念 283
在 Flash 中複製移動補間動畫指令碼 283
整合移動補間動畫指令碼 284
說明動畫 285
新增濾鏡 287
將移動補間動畫與其顯示物件建立關聯 289

第 18 章 使用反向運動
反向運動的基本概念 290
將 IK 骨架製成動畫概觀 291
取得 IK 骨架的相關資訊 292
初始化 IKMover 和限制其移動方式 293
移動 IK 骨架 293
使用彈起 294
使用 IK 事件 294

第 19 章 在三維環境 (3D) 中工作
3D 顯示物件的基本概念 296
瞭解 Flash Player 和 AIR 執行階段中的 3D 顯示物件 297
建立和移動 3D 顯示物件 298
在 2D 檢視上投影 3D 物件 300
範例：透視投影 301
執行複雜的 3D 變形 303
使用三角形產生 3D 特效 306

第 20 章 使用文字的基本概念

第 21 章 使用 TextField 類別
顯示文字 315
選取和處理文字 318
擷取文字輸入 319
限制文字輸入 320
格式化文字 321
進階文字顯示 324
使用靜態文字 326
TextField 範例 : 報紙樣式文字格式 327

第 22 章 使用 Flash 文字引擎

建立和顯示文字 336
處理 FTE 中的事件 340
格式化文字 344
使用字體 347
控制文字 349
Flash Text Engine 範例：新聞版面 353

第 23 章 使用 Text Layout Framework

Text Layout Framework 的概觀 362
使用 Text Layout Framework 363
使用 TLF 組織文字 367
使用 TLF 格式化文字 371
使用 TLF 匯入和匯出文字 372
使用 TLF 管理文字容器 372
使用 TLF 啟用文字選取範圍、編輯和還原 373
使用 TLF 處理事件 374
將影像定位在文字內部 374

第 24 章 使用聲音

使用聲音的基本概念 375
瞭解聲音架構 376
載入外部聲音檔案 377
使用內嵌聲音 379
使用串流聲音檔案 380
使用動態產生的音效 381
播放聲音 383
載入和播放聲音時的安全性考量 386
控制聲音音量和左右相位 386
使用聲音中繼資料 388
存取原始聲音資料 388
擷取聲音輸入 391
聲音範例：Podcast Player 396

第 25 章 使用視訊

視訊的基本觀念 403
瞭解視訊格式 404
瞭解 Video 類別 407
載入視訊檔案 407
控制視訊播放 408
以全螢幕模式播放視訊 410
串流視訊檔案 413
瞭解提示點 413

為中繼資料和提示點撰寫回呼方法 414
使用提示點和中繼資料 418
監視 NetStream 活動 427
視訊檔案的進階主題 430
視訊範例：視訊點唱機 431
使用硬體加速呈現方式的 StageVideo 類別 435

第 26 章 使用攝影機
瞭解 Camera 類別 443
在螢幕上顯示攝影機內容 443
設計攝影機應用程式 444
連接至使用者的攝影機 444
確定已安裝攝影機 445
偵測存取攝影機的權限 446
最佳化攝影機視訊品質 447
監控攝影機狀態 448

第 27 章 使用數位版權管理
瞭解保護的內容工作流程 450
NetStream 類別的 DRM 相關成員和事件 456
使用 DRMStatusEvent 類別 458
使用 DRMAuthenticateEvent 類別 459
使用 DRMErrorEvent 類別 462
使用 DRMManager 類別 463
使用 DRMContentData 類別 464
更新 Flash Player 以支援 Adobe Access 464
頻外授權 465
網域支援 466
使用網域支援功能播放加密的內容 467
授權預覽 467
傳遞內容 467
開放原始碼媒體架構 468

第 28 章 在 AIR 中新增 PDF 內容
偵測 PDF 功能 471
載入 PDF 內容 472
為 PDF 內容編寫指令碼 472
AIR 中 PDF 內容的已知限制 474

第 29 章 使用者互動的基本概念
擷取使用者輸入 475
管理焦點 475
探索輸入類型 477

第 30 章 鍵盤輸入
擷取鍵盤輸入 479
使用 IME 類別 481
虛擬鍵盤 485

第 31 章 滑鼠輸入
擷取滑鼠輸入 492
滑鼠輸入範例：WordSearch 494

第 32 章 觸控、多點觸控和手勢輸入
觸控輸入的基本概念 498
觸控支援探索 500
觸控事件處理 501
觸控和拖曳 503
手勢事件處理 504
疑難排解 508

第 33 章 複製和貼上
複製及貼上基本概念 511
讀取與寫入系統剪貼簿 511
在 AIR 中 HTML 的複製和貼上 512
Clipboard 資料格式 514

第 34 章 加速計輸入
檢查加速計支援 519
偵測加速計變更 519

第 35 章 在 AIR 中拖放
在 AIR 中拖放的基本概念 521
支援拖出手勢 523
支援拖入手勢 525
在 HTML 中拖放 527
將 HTML 元素中的資料拖曳至目標 530
將資料拖曳到 HTML 元素中 531
範例：覆寫預設的 HTML 拖入行為 531
在非應用程式的 HTML 安全執行程序內處理檔案放下 533
放下檔案承諾 534

第 36 章 使用選單
選單的基本概念 542
建立原生選單 (AIR) 548
關於 HTML 中的快顯選單 (AIR) 550
顯示彈出式原生選單 (AIR) 551

處理選單事件 551
原生選單範例：視窗與應用程式選單 (AIR) 553

第 37 章 AIR 中的工作列圖示
關於工作列圖示 556
停駐圖示 556
系統匣圖示 557
視窗工作列圖示和按鈕 559

第 38 章 使用檔案系統
使用 FileReference 類別 561
使用 AIR 檔案系統 API 573

第 39 章 儲存本機資料
共享物件 604
加密本機儲存 612

第 40 章 在 AIR 中使用本機 SQL 資料庫
關於本機 SQL 資料庫 615
建立及修改資料庫 618
操控 SQL 資料庫資料 624
使用同步和非同步資料庫作業 648
為 SQL 資料庫加密 652
使用 SQL 資料庫的策略 667

第 41 章 使用位元組陣列
讀取與寫入 ByteArray 670
ByteArray 範例：讀取 .zip 檔 675

第 42 章 網路作業與通訊的基本概念
Network 介面 681
網路連線變更 682
網域名稱系統 (DNS) 記錄 684

第 43 章 通訊端
TCP 通訊端 687
UDP 通訊端 (AIR) 697
IPv6 位址 698

第 44 章 HTTP 通訊
載入外部資料 700
網路服務要求 707
在其他應用程式中開啟 URL 714

第 45 章 與其他 Flash Player 和 AIR 實體進行通訊
關於 LocalConnection 類別 716
在兩個應用程式之間傳送訊息 718
連線至不同網域和 AIR 應用程式中的內容 719

第 46 章 與 AIR 中的原生程序通訊
原生程序通訊概觀 721
啟動以及關閉原生程序 722
與原生程序通訊 722
原生程序通訊安全性注意事項 724

第 47 章 使用外部 API
使用外部 API 的基本概念 725
外部 API 需求與優點 727
使用 ExternalInterface 類別 728
外部 API 範例：使用瀏覽器在 ActionScript 和 JavaScript 之間通訊 731

第 48 章 AIR 中的 XML 簽名驗證
XML 簽名驗證的基本概念 737
關於 XML 簽名 741
實作 IURIDereferencer 介面 743

第 49 章 用戶端系統環境
用戶端系統環境的基本概念 750
使用 System 類別 751
使用 Capabilities 類別 752
功能範例：偵測系統功能 752

第 50 章 AIR 應用程式叫用與終止
叫用應用程式 756
擷取命令列引數 757
於使用者登入時叫用 AIR 應用程式 760
從瀏覽器叫用 AIR 應用程式 761
終止應用程式 762

第 51 章 使用 AIR 執行階段和作業系統資訊
管理檔案關聯 764
取得執行階段版本的修補程式層級 764
偵測 AIR 功能 765
追蹤使用者狀態 765

第 52 章 使用 AIR 原生視窗
AIR 中的原生視窗基本概念 766
建立視窗 772
管理視窗 780

偵聽視窗事件 788
顯示全螢幕視窗 789

第 53 章 AIR 的顯示螢幕
AIR 顯示螢幕基本概念 791
列舉螢幕 792

第 54 章 列印
列印的基本概念 795
列印頁面 795
Flash 執行階段工作與系統列印 796
設定大小、縮放與列印方向 798
進階列印技術 800
列印範例：多重頁面列印 802
列印範例：縮放、裁切和回應 804
列印範例：頁面設定和列印選項 805

第 55 章 Geolocation
偵測地理位置變更 808

第 56 章 國際化應用程式
國際化應用程式的基本概念 811
flash.globalization 套件概觀 812
判斷地區設定 813
格式化數字 814
格式化貨幣值 816
格式化日期和時間 818
排序和比較字串 819
大小寫轉換 820
範例：國際化股票行情應用程式 821

第 57 章 當地語系化應用程式
選擇地區 825
當地語系化 Flex 內容 825
當地語系化 Flash 內容 826
當地語系化 AIR 應用程式 826
當地語系化日期、時間和貨幣 826

第 58 章 關於 HTML 環境
HTML 環境概觀 829
AIR 和 WebKit 831

第 59 章 在 AIR 中撰寫 HTML 和 JavaScript 程式
關於 HTMLLoader 類別 845
避免發生與安全性有關的 JavaScript 錯誤 847

透過 JavaScript 存取 AIR API 類別 851
關於 AIR 中的 URL 852
讓 JavaScript 可以使用 ActionScript 物件 853
透過 ActionScript 存取 HTML DOM 和 JavaScript 物件 854
在 HTML 中嵌入 SWF 內容 855
在 HTML 網頁中使用 ActionScript 元件庫 858
轉換 Date 和 RegExp 物件 859
透過 ActionScript 操作 HTML 樣式表 860
跨指令碼處理不同安全執行程序中內容 861

第 60 章 為 AIR HTML 容器編寫指令碼
顯示 HTMLLoader 物件的屬性 865
捲動 HTML 內容 867
存取 HTML 操作記錄清單 868
設定載入 HTML 內容時所用的使用者代理程式 869
設定字元編碼以供 HTML 內容使用 869
為 HTML 內容定義類似瀏覽器的使用者介面 869
建立 HTMLLoader 類別的子類別 878

第 61 章 在 AIR 中處理 HTML 相關事件
HTMLLoader 事件 880
使用 ActionScript 處理 DOM 事件 880
回應未捕捉到的 JavaScript 例外 881
使用 JavaScript 處理執行階段事件 883

第 62 章 在行動應用程式中顯示 HTML 內容
StageWebView 物件 886
內容 886
瀏覽事件 888
步驟記錄 889
焦點 890
點陣圖捕捉 892

第 63 章 將 Worker 用於並行
瞭解 Worker 與並行 894
建立及管理 Worker 895
Worker 之間的通訊 897

第 64 章 安全性
Flash 平台安全性概觀 900
安全執行程序 901
權限控制 905
限制網路 API 911
全螢幕模式安全性 913

全螢幕互動模式安全性 914
載入內容 914
跨指令碼處理 917
將載入的媒體當做資料加以存取 919
載入資料 921
從匯入安全性網域中的 SWF 檔載入內嵌內容 924
使用舊版內容 924
設定 LocalConnection 連線權限 924
控制外傳 URL 存取 925
共享物件 926
攝影機、麥克風、剪貼簿、滑鼠和鍵盤存取 927
AIR 安全性 928

第 65 章 如何使用 ActionScript 範例
範例類型 945
在 Flash Professional 執行 ActionScript 3.0 範例 946
在 Flash Builder 執行 ActionScript 3.0 範例 948
在行動裝置執行 ActionScript 3.0 範例 948

第 66 章 本機資料庫內的 SQL 支援
支援的 SQL 語法 952
資料類型支援 969

第 67 章 SQL 錯誤詳細訊息、ID 和引數

第 68 章 Adobe Graphics Assembly Language (AGAL)
AGAL 位元組碼格式 977

第 1 章 使用日期與時間

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

時間雖不能代表一切，但這卻常是軟體應用程式不可或缺的要素。ActionScript 3.0 提供強大的方法供您管理月曆日期、時間以及時間間隔。提供時間功能的兩個主要類別是 flash.utils 套件中的 Date 類別和新的 Timer 類別。

日期與時間是 ActionScript 程式所使用的一般資訊。例如，您可能需要知道目前是星期幾，或是測定使用者在特定畫面所花費的時間等。在 ActionScript 中，您可以使用 Date 類別來代表單一時間片刻，包括日期和時間資訊。Date 實體內有各個日期和時間單位的值，包括年、月、日、星期別、時、分、秒、毫秒和時區。在進階用途方面，ActionScript 還包括 Timer 類別，可用來在特定延遲之後或是於重複間隔執行動作。

更多說明主題

Date

flash.utils.Timer

管理月曆日期和時間

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 中的月曆日期和時間管理功能都集中在最上層的 Date 類別。Date 類別包含方法和屬性，可讓您以 Coordinated Universal Time (UTC) 或某個時區的本地時間處理日期和時間。UTC 是標準時間定義，基本上與格林威治標準時間 (GMT) 相同。

建立 Date 物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Date 類別可說是所有核心類別當中，用途最廣的建構函式方法之一。您可以用四種不同方式叫用這個類別。

首先，如果不提供參數，Date() 建構函式會傳回 Date 物件，其中包含目前日期和時間，以您的時區的本地時間表示。以下是範例：

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

第二，如果指定一個數字參數，Date() 建構函式會將該數字視為是自 1970 年 1 月 1 日起算的毫秒數，然後傳回相對應的 Date 物件。請注意，您所傳遞的毫秒值會視為是自 1970 年 1 月 1 日 (UTC) 起算的毫秒數。但是，Date 物件會以您的本地時區來表示值，除非您使用 UTC 專屬方法來擷取和顯示值。如果是使用單一毫秒參數來建立新的 Date 物件，請務必將您的本地時間與 UTC 的時差列入計算。下列陳述式會建立 Date 物件，設為 1970 年 1 月 1 日當天午夜 (UTC)：

```
var millisecondsPerDay:int = 1000 * 60 * 60 * 24;
// gets a Date one day after the start date of 1/1/1970
var startTime:Date = new Date(millisecondsPerDay);
```

第三，您可以將多個數字參數傳遞至 Date() 建構函式。建構函式會將這些參數分別視為年、月、日、時、分、秒及毫秒，然後傳回相對應的 Date 物件。這些輸入參數會假設是以本地時間而非 UTC 來表示。下列陳述式會取得 Date 物件，設為 2000 年 1 月 1 日開始的午夜 (本地時間)：

```
var millenium:Date = new Date(2000, 0, 1, 0, 0, 0, 0);
```

第四，您可以將單一字串參數傳遞至 Date() 建構函式。建構函式會嘗試將字串剖析為日期或時間組件，然後傳回相對應的 Date 物件。如果您使用這種方法，最好將 Date() 建構函式放在 try..catch 區塊內，以截獲任何剖析錯誤。Date() 建構函式接受多種不同的字串格式 (如需相關資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考)。下列陳述式會使用字串值初始化新的 Date 物件：

```
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");
```

如果 Date() 建構函式無法成功剖析字串參數，就不會引發例外。但是，產生的 Date 物件會包含無效的日期值。

取得時間單位值

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 Date 類別的屬性或方法，從 Date 物件擷取各種時間單位的值。下列每個屬性都可以提供 Date 物件中的時間單位值：

- fullYear 屬性
- month 屬性：以數字格式表示，0 代表 1 月，一直到 11 代表 12 月
- date 屬性：月份中日期別的月曆數字，範圍在 1 到 31 之間
- day 屬性：以數字格式表示的星期別，0 代表星期日
- hours 屬性：範圍在 0 到 23 之間
- minutes 屬性
- seconds 屬性
- milliseconds 屬性

  實際上，Date 類別可提供多種方式來擷取每一個值。例如，您可以用四種不同方式取得 Date 物件的月份值：

- month 屬性
- getMonth() 方法
- monthUTC 屬性
- getMonthUTC() 方法

  這四種方式的效率其實都相同，因此您可以視應用程式所需，使用其中一種方式。

  上面所列的屬性都是總日期值的組件。例如，milliseconds 屬性不會超過 999，因為一到 1000，seconds 值就會增加 1，而 milliseconds 屬性會重設為 0。

  如果您想要以自 1970 年 1 月 1 日 (UTC) 起算的毫秒數取得 Date 物件的值，可以使用 getTime() 方法。這個方法的對應方法為 setTime()，可讓您使用自 1970 年 1 月 1 日 (UTC) 起算的毫秒數變更現有 Date 物件的值。

執行日期與時間計算

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以加減 Date 類別的日期與時間。由於日期值在內部是以毫秒計算，因此在從 Date 物件加減值時，必須先將值轉換為毫秒。

如果您的應用程式會執行大量的日期與時間計算，那麼最好能建立常數，以毫秒為單位存放常用的時間單位值，如下所示：

```
public static const millisecondsPerMinute:int = 1000 * 60;
public static const millisecondsPerHour:int = 1000 * 60 * 60;
public static const millisecondsPerDay:int = 1000 * 60 * 60 * 24;
```

現在，您就可以輕鬆地使用標準時間單位來執行日期計算。下列程式碼會使用 getTime() 和 setTime() 方法，將日期值設為目前時間過後一小時：

```
var oneHourFromNow:Date = new Date();
oneHourFromNow.setTime(oneHourFromNow.getTime() + millisecondsPerHour);
```

設定日期值的另一個方式是使用單一毫秒參數建立新的 Date 物件。例如，下列程式碼會將 30 天加至某個日期，然後計算出另一個日期：

```
// sets the invoice date to today's date
var invoiceDate:Date = new Date();

// adds 30 days to get the due date
var dueDate:Date = new Date(invoiceDate.getTime() + (30 * millisecondsPerDay));
```

接著，millisecondsPerDay 會乘以 30 表示 30 天的時間，而且結果會加入至 invoiceDate 值，並且用於設定 dueDate 值。

轉換時區

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

日期與時間計算可以很容易讓您將日期從某個時區轉換為另一個時區。getTimezoneOffset() 方法也很方便，這個方法會傳回 Date 物件的時區與 UTC 相差的分鐘值。傳回分鐘值的原因是，並非所有時區都是以整點方式遞增，部分時區與相鄰時區只相差半小時。

下列範例會使用時區偏移，將日期從本地時間轉換為 UTC。首先會以毫秒算出時區值，然後根據該值調整 Date 值：

```
// creates a Date in local time
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");

// converts the Date to UTC by adding or subtracting the time zone offset
var offsetMilliseconds:Number = nextDay.getTimezoneOffset() * 60 * 1000;
nextDay.setTime(nextDay.getTime() + offsetMilliseconds);
```

控制時間間隔

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您使用 Adobe Flash CS4 Professional 開發應用程式時，可以存取時間軸，這會為應用程式提供穩定且以影格為單位的進度。但是，在完全使用 ActionScript 的專案中，就必須依賴其它時間機制。

迴圈與計時器的比較

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在某些程式語言中，您必須使用迴圈陳述式 (如 for 或 do..while) 來自行設計計時機制。

迴圈陳述式的執行速度一般會視本機電腦而定，因此應用程式在某些電腦上的執行速度較快，某些則較慢。如果應用程式需要一致的時間間隔，您就必須將其與實際月曆或時鐘的時間繫結。許多應用程式，例如遊戲、動畫和即時控制器都需要規律、以時間驅動而且在每一部機器上都一致的計時機制。

ActionScript 3.0 的 Timer 類別即是強大的解決方案。使用 ActionScript 3.0 事件模型，每當達到指定的時間間隔，Timer 類別就會傳送計時器事件。

Timer 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 ActionScript 3.0 中處理計時功能時，一般會使用 Timer 類別 (flash.utils.Timer)，每當達到指定的時間間隔，這個類別就會傳送事件。

若要啟動計時器，您必須先建立 Timer 類別的實體，並指示產生計時器事件的頻率，以及要產生多少次事件才停止。

例如，下列程式碼會建立 Timer 實體，每秒傳送一次事件，持續 60 秒：

```
var oneMinuteTimer:Timer = new Timer(1000, 60);
```

每當達到指定的間隔時，Timer 物件就會傳送 TimerEvent 物件。TimerEvent 物件的事件類型為 timer (由常數 TimerEvent.TIMER 所定義)。TimerEvent 物件包含的屬性與標準 Event 物件相同。

如果將 Timer 實體設定為固定次數的間隔，那麼當達到最後的間隔時，也會傳送 timerComplete 事件 (由常數 TimerEvent.TIMER_COMPLETE 所定義)。

下面是一個小型樣本應用程式，顯示如何使用 Timer 類別：

```
package
{
    import flash.display.Sprite;
    import flash.events.TimerEvent;
    import flash.utils.Timer;

    public class ShortTimer extends Sprite
    {
        public function ShortTimer()
        {
            // creates a new five-second Timer
            var minuteTimer:Timer = new Timer(1000, 5);

            // designates listeners for the interval and completion events
            minuteTimer.addEventListener(TimerEvent.TIMER, onTick);
            minuteTimer.addEventListener(TimerEvent.TIMER_COMPLETE, onTimerComplete);

            // starts the timer ticking
            minuteTimer.start();
        }

        public function onTick(event:TimerEvent):void
        {
            // displays the tick count so far
            // The target of this event is the Timer instance itself.
            trace("tick " + event.target.currentCount);
        }

        public function onTimerComplete(event:TimerEvent):void
        {
            trace("Time's Up!");
        }
    }
}
```

建立 ShortTimer 類別時，會建立 Timer 實體，此實體會每秒計時一次，持續五秒。接著會將兩個偵聽程式加入計時器，一個用來偵聽每次計時，另一個則偵聽 timerComplete 事件。

接下來會啟動計時器，從現在開始，onTick() 方法就會每一秒執行一次。

onTick() 方法只會顯示目前的計時計數。等到五秒一過，onTimerComplete() 方法會執行，指出時間已到。

當您執行這個樣本時，會在主控台或追蹤視窗看到下面這幾行，每秒出現一行：

```
tick 1
tick 2
tick 3
tick 4
tick 5
Time's Up!
```

flash.utils 套件中的計時函數

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 包含幾個計時函數，這些計時函數和 ActionScript 2.0 所提供的類似。這些函數在 flash.utils 套件中是以套件層級的函數提供，其運作方式和在 ActionScript 2.0 中完全相同。

函數	說明
clearInterval(id:uint):void	取消指定的 setInterval() 呼叫。
clearTimeout(id:uint):void	取消指定的 setTimeout() 呼叫。
getTimer():int	傳回從初始化 Adobe® Flash® Player 或 Adobe® AIR™ 以後所經過的毫秒數。
setInterval(closure:Function, delay:Number, ... arguments):uint	在指定的間隔 (以毫秒為單位) 執行函數。
setTimeout(closure:Function, delay:Number, ... arguments):uint	在指定的延遲時間 (以毫秒為單位) 後執行指定的函數。

ActionScript 3.0 保留這些函數以提供舊版相容性。Adobe 不建議您在新的 ActionScript 3.0 應用程式中使用這些函數。一般來說，在應用程式中使用 Timer 類別會較容易且具備效率。

日期與時間範例：簡易類比時鐘

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

有個簡單的類比時鐘範例可說明這兩個日期與時間概念：

- 取得目前的日期與時間，然後擷取時、分和秒的值
- 使用 Timer 設定應用程式的步調

  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。您可以在 Samples/SimpleClock 檔案夾中找到 SimpleClock 應用程式檔案，此應用程式是由下列檔案組成：

檔案	說明
SimpleClockApp.mxml 或 SimpleClockApp.fla	主應用程式檔案，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/simpleclock/SimpleClock.as	主應用程式檔案。
com/example/programmingas3/simpleclock/AnalogClockFace.as	繪製圓形鐘面，並根據時間繪製時針、分針和秒針。

定義 SimpleClock 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

這個時鐘範例很簡單，但即使是簡單的應用程式，最好還是能組織清楚，以便日後擴充。因此，SimpleClock 應用程式會使用 SimpleClock 類別處理啟動和計時工作，然後使用另一個名為 AnalogClockFace 的類別來顯示時間。

以下是定義並初始化 SimpleClock 類別的程式碼 (請注意，在 Flash 版本中，SimpleClock 會改為擴充 Sprite 類別)：

```
public class SimpleClock extends UIComponent
{
    /**
     * The time display component.
     */
    private var face:AnalogClockFace;

    /**
     * The Timer that acts like a heartbeat for the application.
     */
    private var ticker:Timer;
```

這個類別有兩個重要屬性：

- face 屬性：這是 AnalogClockFace 類別的實體
- ticker 屬性：這是 Timer 類別的實體

SimpleClock 類別會使用預設建構函式。initClock() 方法負責實際的設定工作，包括建立鐘面和啟動 Timer 實體計時。

建立鐘面

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

SimpleClock 程式碼中的下面幾行會建立用來顯示時間的鐘面：

```
    /**
     * Sets up a SimpleClock instance.
     */
    public function initClock(faceSize:Number = 200)
    {
        // creates the clock face and adds it to the display list
        face = new AnalogClockFace(Math.max(20, faceSize));
        face.init();
        addChild(face);

        // draws the initial clock display
        face.draw();
```

您可將鐘面的大小傳遞至 initClock() 方法。如果沒有傳遞 faceSize 值，則預設大小為 200 像素。

接下來，應用程式會初始化鐘面，然後使用從 DisplayObjectContainer 類別繼承的 addChild() 方法，將鐘面加入顯示清單。之後則會呼叫 AnalogClockFace.draw() 方法顯示一次鐘面，在其中顯示目前的時間。

啟動計時器

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在建立鐘面之後，initClock() 方法會設定計時器：

```
        // creates a Timer that fires an event once per second
        ticker = new Timer(1000);

        // designates the onTick() method to handle Timer events
        ticker.addEventListener(TimerEvent.TIMER, onTick);

        // starts the clock ticking
        ticker.start();
```

首先，這個方法會建立 Timer 實體，設定每秒 (即每 1000 毫秒) 傳送一次事件。由於沒有第二個 repeatCount 參數傳遞至 Timer() 建構函式，所以 Timer 會不斷重複。

SimpleClock.onTick() 方法會在每秒收到 timer 事件時執行一次：

```
    public function onTick(event:TimerEvent):void
    {
        // updates the clock display
        face.draw();
    }
```

AnalogClockFace.draw() 方法只會繪製鐘面和指針。

顯示目前時間

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

AnalogClockFace 類別中的大部分程式碼都是用來設定鐘面的顯示元素。初始化 AnalogClockFace 時，會繪製一個圓形、將數字文字標籤放在每個小時刻度，然後建立三個 Shape 物件，分別代表時鐘上的時針、分針和秒針。

當 SimpleClock 應用程式執行時，會每秒呼叫一次 AnalogClockFace.draw() 方法，如下所示：

```
    /**
     * Called by the parent container when the display is being drawn.
     */
    public override function draw():void
    {
        // stores the current date and time in an instance variable
        currentTime = new Date();
        showTime(currentTime);
    }
```

這個方法會將目前時間儲存在變數中，使時間無法在繪製指針的期間改變。然後會呼叫 showTime() 方法顯示指針，如下所示：

```
/**
 * Displays the given Date/Time in that good old analog clock style.
 */
public function showTime(time:Date):void
{
    // gets the time values
    var seconds:uint = time.getSeconds();
    var minutes:uint = time.getMinutes();
    var hours:uint = time.getHours();

    // multiplies by 6 to get degrees
    this.secondHand.rotation = 180 + (seconds * 6);
    this.minuteHand.rotation = 180 + (minutes * 6);

    // Multiply by 30 to get basic degrees, then
    // add up to 29.5 degrees (59 * 0.5)
    // to account for the minutes.
    this.hourHand.rotation = 180 + (hours * 30) + (minutes * 0.5);
}
```

首先，這個方法會擷取目前時間的時、分和秒值。接著，則會使用這些值來計算每個指針的角度。由於秒針會 60 秒旋轉一圈，表示每秒旋轉的角度為 6 度 (360/60)。分針也是每分鐘旋轉 6 度。

時針也會每分鐘更新一次，以跟著分針走。時針會每小時旋轉 30 度 (360/12)，但也會每分鐘旋轉半度 (30 度除以 60 分鐘)。

第 2 章 使用字串

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

String 類別中包含的方法，可讓您使用文字字串。字串在處理許多物件時相當重要。這裡說明的方法在處理物件中使用的字串時很有用，例如 TextField、StaticText、XML、ContextMenu 和 FileReference 物件。

字串是一連串的字元，ActionScript 3.0 支援 ASCII 和 Unicode 字元。

更多說明主題

String

RegExp

parseFloat()

parseInt()

字串的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

以程式設計的術語來說，字串是文字值，也就是由一串字母、數字或其它字元串連而成的單一值。例如，這行程式碼會建立資料類型為 String 的變數，並為該變數指定常值字串值：

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

如本範例所示，在 ActionScript 中，只要在文字前後標上雙引號或單引號，就能用來代表字串值。以下列舉幾個字串範例：

```
"Hello"
"555-7649"
"http://www.adobe.com/"
```

任何時候，當您在 ActionScript 中處理一段文字時，您所處理的文字就是字串值。ActionScript 的 String 類別是您在處理文字值時所能使用的資料類別。String 實體經常用於其它許多 ActionScript 類別中的屬性、方法參數。

重要概念與術語

下列參考清單包含您將會遇到的字串重要術語：

ASCII 在電腦程式中，用來代表文字字元與符號的系統。ASCII 系統支援 26 個英字文母以及有限的額外字元集。

字元 文字資料的最小單位 (單一字母或符號)。

連接 在字串值之後再加上另一個字串值，將多個字串值結合起來，以建立新的字串值。

空字串 未包含文字、空格或其它字元的字串，撰寫方式為 ""。空字串值與具有 null 值的 String 變數不同，後者是未指定 String 實體的變數，而空字串則具有實體，只是實體的值並未包含任何字元。

String 文字值 (一連串的字元)。

字串常值 (或稱「常值字串」) 在程式碼中明確撰寫的字串值，其撰寫方式為文字值前後加上雙引號或單引號。

子字串 為另一字串中一部分的字串。

Unicode 在電腦程式中，用來代表文字字元與符號的標準系統。Unicode 系統允許在任何撰寫系統中使用任何字元。

建立字串

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

String 類別是用來代表 ActionScript 3.0 中的字串 (文字) 資料，ActionScript 字串支援 ASCII 和 Unicode 字元。建立字串最簡單的方式就是使用字串常值。若要宣告字串常值，請使用半形雙引號 (") 或單引號 (') 字元。例如，下列兩個字串是相等的：

```
var str1:String = "hello";
var str2:String = 'hello';
```

您也可以使用 new 運算子宣告字串，如下所示：

```
var str1:String = new String("hello");
var str2:String = new String(str1);
var str3:String = new String();       // str3 == ""
```

下列這兩個字串是相等的：

```
var str1:String = "hello";
var str2:String = new String("hello");
```

若要在單引號 (') 分隔符號定義的字串常值內使用單引號 (")，請使用反斜線跳脫字元 (\)。同樣地，若要在雙引號 (") 分隔符號定義的字串常值內使用雙引號 (")，也請使用反斜線跳脫字元 (\)。下列這兩個字串是相等的：

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

您可以根據字串常值中存在的任何單或雙引號，選擇使用單引號或雙引號，如下所示：

```
var str1:String = "ActionScript <span class='heavy'>3.0</span>";
var str2:String = '<item id="155">banana</item>';
```

請注意，ActionScript 會區分半形單引號 (') 與左右單引號 (' 或 ')，而且也會如此區分雙引號。請使用半形引號來表示字串常值。當從其它來源貼上文字到 ActionScript 時，請務必使用正確的字元。

如下表所示，您可以使用反斜線跳脫字元 (\) 來定義字串常值中的其它字元：

跳脫序列	字元
\b	退格字元
\f	換頁字元
\n	換行字元
\r	歸位字元
\t	Tab 鍵
\u*nnnn*	Unicode 字元加上十六進位數字 nnnn 指定的字元碼，例如，\u263a 是笑臉字元
\\x*nn*	ASCII 字元加上十六進位數字 nn 指定的字元碼
\'	單引號
\"	雙引號
\\	單反斜線字元

length 屬性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

每個字串都有 length 屬性，其值等於字串中的字元數目：

```
var str:String = "Adobe";
trace(str.length);            // output: 5
```

空字串和 null 字串的長度都是 0，如下列範例所示：

```
var str1:String = new String();
trace(str1.length);           // output: 0

str2:String = '';
trace(str2.length);           // output: 0
```

使用字串中的字元

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

字串中的每個字元在字串中都有索引位置 (整數)。第一個字元的索引位置為 0。例如，在下列字串中，字元 y 位於位置 0，而字元 w 位於位置 5：

```
"yellow"
```

您可以使用 charAt() 方法和 charCodeAt() 方法，檢查字串中各個位置的個別字元，如下列範例所示：

```
var str:String = "hello world!";
for (var i:int = 0; i < str.length; i++)
{
    trace(str.charAt(i), "-", str.charCodeAt(i));
}
```

當您執行這段程式碼時，會產生下列輸出：

```
h - 104
e - 101
l - 108
l - 108
o - 111
- 32
w - 119
o - 111
r - 114
l - 108
d - 100
! - 33
```

您也可以使用 fromCharCode() 方法，以字元碼定義字串，如下列範例所示：

```
var myStr:String = String.fromCharCode(104,101,108,108,111,32,119,111,114,108,100,33);
        // Sets myStr to "hello world!"
```

比較字串

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用下列運算子比較字串：<、<=、!=、==、=> 和 >。這些運算子可以和條件陳述式搭配使用，如 if 和 while，如下列範例所示：

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2)
{
    trace("A < a, B < b, C < c, ...");
}
```

當在字串使用這些運算子時，ActionScript 會計算字串中每個字元的字元碼值，然後從左而右比較字元，如下所示：

```
trace("A" < "B"); // true
trace("A" < "a"); // true
trace("Ab" < "az"); // true
trace("abc" < "abza"); // true
```

使用 == 和 != 運算子可以比較兩個字串，以及將字串與其它類型的物件比較，如下列範例所示：

```
var str1:String = "1";
var str1b:String = "1";
var str2:String = "2";
trace(str1 == str1b); // true
trace(str1 == str2); // false
var total:uint = 1;
trace(str1 == total); // true
```

取得其它物件的字串形式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以取得任何類型物件的字串形式。所有物件都有 toString() 方法以供轉換：

```
var n:Number = 99.47;
var str:String = n.toString();
    // str == "99.47"
```

當使用 + 連接運算子來組合 String 物件和非字串物件時，就不需要使用 toString() 方法。如需有關連接字串的詳細資訊，請參閱下一節。

String() 全域函數針對特定物件傳回的值，會與物件呼叫 toString() 方法傳回的值相同。

連接字串

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

連接字串表示會使用兩個字串，然後依序排列後連接為一個字串。例如，您可以使用 + 運算子連接兩個字串：

```
var str1:String = "green";
var str2:String = "ish";
var str3:String = str1 + str2; // str3 == "greenish"
```

您也可以使用 += 運算子產生相同的結果，如下列範例所示：

```
var str:String = "green";
str += "ish"; // str == "greenish"
```

此外，String 類別還包含 concat() 方法，其使用方式如下所示：

```
var str1:String = "Bonjour";
var str2:String = "from";
var str3:String = "Paris";
var str4:String = str1.concat(" ", str2, " ", str3);
// str4 == "Bonjour from Paris"
```

如果您使用 + 運算子 (或 += 運算子) 連接 String 物件和 「非」字串物件，ActionScript 會自動將非字串物件轉換為 String 物件，這樣才能評估運算式，如下列範例所示：

```
var str:String = "Area = ";
var area:Number = Math.PI * Math.pow(3, 2);
str = str + area; // str == "Area = 28.274333882308138"
```

但是，您可以使用括號進行群組，為 + 運算子提供內容，如下列範例所示：

```
trace("Total: $" + 4.55 + 1.45); // output: Total: $4.551.45
trace("Total: $" + (4.55 + 1.45)); // output: Total: $6
```

尋找字串中的子字串和樣式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

子字串是字串內的連續字元。例如，字串 "abc" 有下列子字串：""、"a"、"ab"、"abc"、"b"、"bc"、"c"。您可以使用 ActionScript 方法找出字串的子字串。

樣式在 ActionScript 中是由字串或規則運算式所定義的。例如，下列規則運算式會定義特定樣式，即字母 A、B 和 C，後面接著一個數字字元 (正斜線是規則運算式的分隔符號)：

```
/ABC\d/
```

ActionScript 提供方法來尋找字串中的樣式，以及用取代子字串來取代找到的符合項目。下列各節會說明這些方法。

規則運算式可定義複雜樣式。如需詳細資訊，請參閱第 64 頁 「使用規則運算式」。

依字元位置找出子字串

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

substr() 和 substring() 方法很類似。這兩個方法都會傳回字串的子字串，而且都會使用兩個參數。在這兩種方法中，第一個參數都是指定字串中開頭字元的位置。但是，substr() 方法的第二個參數是要傳回的子字串 「長度」，而 substring() 方法的第二個參數則是子字串 「結尾」字元的位置 (傳回的字串不含在此位置的字元)。本範例展示這兩種方法的差異：

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.substr(11,15)); // output: Paris, Texas!!!
trace(str.substring(11,15)); // output: Pari
```

slice() 方法的作用與 substring() 方法類似。指定兩個非負數的整數當成參數時，結果是相同的。但是，slice() 方法可以使用負值整數做為參數，在這種情況下，字元位置會從字串結尾算起，如下列範例所示：

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.slice(11,15)); // output: Pari
trace(str.slice(-3,-1)); // output: !!
trace(str.slice(-3,26)); // output: !!!
trace(str.slice(-3,str.length)); // output: !!!
trace(str.slice(-8,-3)); // output: Texas
```

您可以結合非負值和負值整數，當成 slice() 方法的參數。

找出相符子字串的字元位置

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 indexOf() 和 lastIndexOf() 方法，找出字串內相符的子字串，如下列範例所示：

```
var str:String = "The moon, the stars, the sea, the land";
trace(str.indexOf("the")); // output: 10
```

請注意，indexOf() 方法會區分大小寫。

您可以指定第二個參數，指出要在字串中開始搜尋的索引位置，如下所示：

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.indexOf("the", 11)); // output: 21
```

lastIndexOf() 方法會找出子字串在字串中最後出現的位置：

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the")); // output: 30
```

如果您指定 lastIndexOf() 方法的第二個參數，會從該索引位置以回溯方式搜尋字串 (從右到左)：

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the", 29)); // output: 21
```

建立由分隔符號區分的子字串陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 split() 方法，依照分隔符號來拆解字串，據以建立子字串陣列。例如，您可以將以逗點分隔和定位點分隔的字串，區分為數個字串。

下列範例顯示如何使用 and 符號 (&) 字元當成分隔符號，將陣列拆解為子字串：

```
var queryStr:String = "first=joe&last=cheng&title=manager&StartDate=3/6/65";
var params:Array = queryStr.split("&", 2); // params == ["first=joe","last=cheng"]
```

split() 方法的第二個參數是選擇性的，會定義所傳回陣列的大小上限。

您也可以使用規則運算式當做分隔符號字元：

```
var str:String = "Give me\t5."
var a:Array = str.split(/\s+/); // a == ["Give","me","5."]
```

如需詳細資訊，請參閱第 64 頁「使用規則運算式」和適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。

尋找字串中的樣式並取代子字串

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

String 類別包含下列方法，可處理字串中的樣式：

- 使用 match() 和 search() 方法，可以找出符合樣式的子字串。
- 使用 replace() 方法可以尋找符合樣式的子字串，然後使用指定的子字串取代這些子字串。

下列各節會說明這些方法。

您可以使用字串或規則運算式定義這些方法所用的樣式。如需有關規則運算式的詳細資訊，請參閱第 64 頁 「使用規則運算式」。

尋找符合的子字串

search() 方法會傳回符合特定樣式的第一個子字串的索引位置，如下列範例所示：

```
var str:String = "The more the merrier.";
// (This search is case-sensitive.)
trace(str.search("the")); // output: 9
```

您也可以使用規則運算式定義要比對的樣式，如下列範例所示：

```
var pattern:RegExp = /the/i;
var str:String = "The more the merrier.";
trace(str.search(pattern)); // 0
```

trace() 方法的輸出為 0，因為字串中第一個字元的索引位置為 0。規則運算式中設定了 i 旗標，因此搜尋不會區分大小寫。

search() 方法只找到一個符合的項目，並傳回其開頭索引位置，即使規則運算式設定了 g (全域) 旗標也會傳回。

下列範例顯示一個更複雜的規則運算式，這個規則運算式會比對雙引號中的字串：

```
var pattern:RegExp = /"[^"]*"/;
var str:String = "The \"more\" the merrier.";
trace(str.search(pattern)); // output: 4

str = "The \"more the merrier.";
trace(str.search(pattern)); // output: -1
// (Indicates no match, since there is no closing double quotation mark.)
```

match() 方法有類似的運作方式，也會搜尋相符的子字串。但是，當您在規則運算式樣式中使用全域旗標時 (如下列範例所示)，match() 會傳回符合子字串的陣列：

```
var str:String = "bob@example.com, omar@example.org";
var pattern:RegExp = /\w*@\w*\.[org|com]+/g;
var results:Array = str.match(pattern);
```

results 陣列會設定如下：

```
["bob@example.com","omar@example.org"]
```

如需有關規則運算式的詳細資訊，請參閱第 64 頁 「使用規則運算式」。

取代符合的子字串

您可以使用 replace() 方法在字串中搜尋指定的樣式，然後使用指定的取代字串來取代符合的項目，如下列範例所示：

```
var str:String = "She sells seashells by the seashore.";
var pattern:RegExp = /sh/gi;
trace(str.replace(pattern, "sch")); //sche sells seaschells by the seaschore.
```

請注意，在這個範例中，符合的字串不區分大小寫，因為規則運算式中設定了 i (ignoreCase) 旗標，此外也取代多個符合的項目，這是因為設定了 g (global) 旗標。如需詳細資訊，請參閱第 64 頁 「使用規則運算式」。

您可以在取代字串中加入下列 $ 取代程式碼。下表顯示的取代文字會插入以代替 $ 取代程式碼：

$ 程式碼	取代文字
$$	$
$&	符合的子字串。
$`	符合之子字串前面的字串部分。此程式碼會使用半形左單引號字元 (`)，而非半形單引號 (') 或左智慧單引號 (')。
$'	符合之子字串後面的字串部分。此程式碼會使用半形單引號 (').
$n	第 n 個捕捉到的括號群組相符項目，其中 n 是 1 到 9 的單一位數，而且 $ n 的後面沒有小數位數。
$nn	第 nn 個捕捉到的括號群組相符項目，其中 nn 是兩位數的十進位數字 (01 到 99)。如果第 nn 個捕捉未定義，取代文字便會是空字串。

例如，以下示範 $2 與 $1 取代程式碼的用法，這些程式碼代表捕捉到的第一個與第二個相符群組：

```
var str:String = "flip-flop";
var pattern:RegExp = /(\w+)-(\w+)/g;
trace(str.replace(pattern, "$2-$1")); // flop-flip
```

您也可以使用函數當做 replace() 方法的第二個參數。符合的文字會由函數的傳回值取代。

```
var str:String = "Now only $9.95!";
var price:RegExp = /\$([\d,]+.\d+)+/i;
trace(str.replace(price, usdToEuro));

function usdToEuro(matchedSubstring:String,  capturedMatch1:String,  index:int,
str:String):String
{
    var usd:String = capturedMatch1;
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.853690;
    var euro:Number = parseFloat(usd) * exchangeRate;
    const euroSymbol:String = String.fromCharCode(8364);
    return euro.toFixed(2) + " " + euroSymbol;
}
```

當您使用函數做為 replace() 方法的第二個參數時，下列引數會傳遞至函數：

- 字串的符合部分。
- 任何擷取到的括號群組相符項目。以此方式傳遞的引數將取決於括號群組的相符項目數而有所不同。您可以檢查函數程式碼內的 arguments.length - 3，判斷括號符合項目的數目。
- 相符項目在字串中開始的索引位置。
- 完整的字串。

轉換字串大小寫

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如下列範例所示，toLowerCase() 方法和 toUpperCase() 方法，會分別將字串中的字母轉換為小寫和大寫：

```
var str:String = "Dr. Bob Roberts, #9."
trace(str.toLowerCase()); // dr. bob roberts, #9.
trace(str.toUpperCase()); // DR. BOB ROBERTS, #9.
```

執行這些方法之後，原始字串仍會維持不變。若要轉換原始字串，請使用下列程式碼：

```
str = str.toUpperCase();
```

這些方法可用於延伸字元，而不只是 a–z 和 A–Z：

```
var str:String = "José Barça";
trace(str.toUpperCase(), str.toLowerCase()); // JOSÉ BARÇA josé barça
```

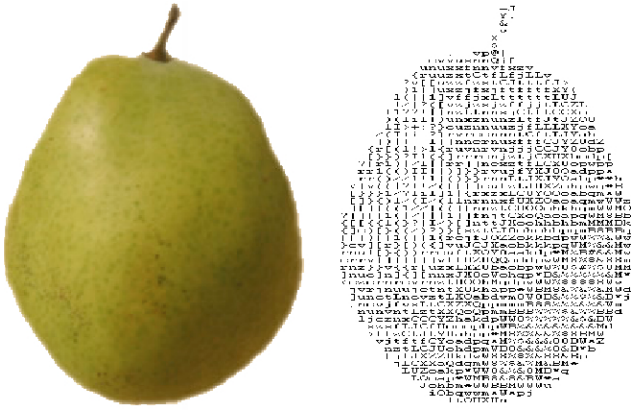
字串範例：ASCII 藝術

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

這個 「ASCII 藝術」範例會說明在 ActionScript 3.0 中使用 String 類別的多項功能，其中包括：

- 使用 String 類別的 split() 方法，從字元分隔字串 (定位點分隔文字檔中的影像資訊) 擷取值。
- 使用多種字串操作技巧 (包括 split()、連接，以及使用 substring() 和 substr() 擷取字串的部分)，將影像標題中每個字的第一個字母改為大寫。
- 使用 getCharAt() 方法，從字串取得單一字元 (以決定與灰階點陣圖值相對應的 ASCII 字元)。
- 使用字串連接，以一次一個字元的方式建立影像的 ASCII 藝術圖像。

「ASCII 藝術」是指用文字來呈現影像，利用一組固定間距字體的字元 (如 Courier New 字元) 來繪製影像。下圖便是應用程式所產生的 ASCII 藝術範例：



右邊顯示的是圖片的 ASCII 藝術版本。

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。您可以在 Samples/AsciiArt 檔案夾中找到 ASCIIArt 應用程式檔案，此應用程式是由下列檔案組成：

檔案	說明
AsciiArtApp.mxml 或 AsciiArtApp.fla	主應用程式檔案，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/asciiArt/AsciiArtBuilder.as	提供應用程式主要功能的類別，這些功能包括從文字檔擷取影像中繼資料、載入影像，以及管理影像轉為文字的程序。
com/example/programmingas3/asciiArt/BitmapToAsciiConverter.as	提供 parseBitmapData() 方法的類別，這個方法會將影像資料轉換為 String 版本。
com/example/programmingas3/asciiArt/Image.as	代表所載入點陣圖影像的類別。

檔案	說明
com/example/programmingas3/asciiArt/ImageInfo.as	代表 ASCII 藝術影像中繼資料 (如標題、影像檔 URL 等) 的類別。
image/	其中包含應用程式所使用影像的資料夾。
txt/ImageData.txt	以定位點分隔的文字檔，其中包含應用程式將載入的影像的相關資訊。

擷取定位點分隔值

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

這個範例會採取一般將應用程式資料與應用程式本身分開儲存的做法，這樣一來，如果資料變更 (例如加入其它影像或影像的標題變更)，就不需要重新建立 SWF 檔。在此範例中，影像中繼資料 (包括影像標題、實際影像檔的 URL，以及用來操作影像的一些值) 會儲存在文字檔中 (即專案中的 txt/ImageData.txt 檔案)。文字檔的內容如下：

```
FILENAMETITLEWHITE_THRESHHOLDBLACK_THRESHHOLD
FruitBasket.jpgPear, apple, orange, and bananad810
Banana.jpgA picture of a bananaC820
Orange.jpgorangeFF20
Apple.jpgpicture of an apple6E10
```

檔案使用特定的定位點分隔格式。第一行 (列) 是標題列。其它行則包含要載入的每個點陣圖的下列資料：

- 點陣圖的檔名。
- 點陣圖的顯示名稱。
- 點陣圖的白色臨界值和黑色臨界值。這些是十六進位值，如果像素超過或低於這些值，將會視為全白或全黑。

應用程式一啟動，AsciiArtBuilder 類別就會使用 AsciiArtBuilder 類別的 parseImageInfo() 方法的下列程式碼，載入並解析文字檔的內容，以建立要顯示的影像「堆疊」：

```
var lines:Array = _imageInfoLoader.data.split("\n");
var numLines:uint = lines.length;
for (var i:uint = 1; i < numLines; i++)
{
    var imageInfoRaw:String = lines[i];
    ...
    if (imageInfoRaw.length > 0)
    {
        // Create a new image info record and add it to the array of image info.
        var imageInfo:ImageInfo = new ImageInfo();

        // Split the current line into values (separated by tab (\t)
        // characters) and extract the individual properties:
        var imageProperties:Array = imageInfoRaw.split("\t");
        imageInfo.fileName = imageProperties[0];
        imageInfo.title = normalizeTitle(imageProperties[1]);
        imageInfo.whiteThreshold = parseInt(imageProperties[2], 16);
        imageInfo.blackThreshold = parseInt(imageProperties[3], 16);
        result.push(imageInfo);
    }
}
```

文字檔的整個內容都包含在單一 String 實體 (即 _imageInfoLoader.data 屬性) 中。使用 split() 方法並將換行字元 ("\n") 設為參數，String 實體就會分割成 Array (lines)，其元素即為文字檔中的每一行。程式碼接著會使用迴圈來處理每一行 (第一行除外，因為這只是標題，無實質內容)。在迴圈內，會再次使用 split() 方法，將單行內容分成一組值 (即名為 imageProperties 的 Array 物件)。split() 方法這次使用的參數是定位點 ("\t") 字元，因為各行的值都是用定位點字元隔開。

使用 String 方法標準化影像標題

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

這個應用程式的其中一項設計決定，便是讓所有影像標題都以標準格式顯示，也就是將每個字的第一個字母都改為大寫 (只有少數通常在英文標題裡不大寫的字不改)。應用程式不會假設文字檔包含格式正確的標題，而會在擷取文字檔中的標題時予以格式化。

在前面列出的程式碼中，當擷取個別影像中繼資料值時，會使用下面這一行程式碼：

```
imageInfo.title = normalizeTitle(imageProperties[1]);
```

在此程式碼中，從文字檔擷取的影像標題會透過 normalizeTitle() 方法傳遞，然後再儲存到 ImageInfo 物件中：

```
private function normalizeTitle(title:String):String
{
    var words:Array = title.split(" ");
    var len:uint = words.length;
    for (var i:uint; i < len; i++)
    {
        words[i] = capitalizeFirstLetter(words[i]);
    }

    return words.join(" ");
}
```

這個方法會使用 split() 方法，將標題分為獨立的單字 (用空格字元隔開)、將每個字透過 capitalizeFirstLetter() 方法傳遞，然後再使用 Array 類別的 join() 方法，將這些字再次組合成單一字串。

顧名思義，capitalizeFirstLetter() 方法就是將每個字的第一個字母改為大寫：

```
/**
 * Capitalizes the first letter of a single word, unless it's one of
 * a set of words that are normally not capitalized in English.
 */
private function capitalizeFirstLetter(word:String):String
{
    switch (word)
    {
        case "and":
        case "the":
        case "in":
        case "an":
        case "or":
        case "at":
        case "of":
        case "a":
            // Don't do anything to these words.
            break;
        default:
            // For any other word, capitalize the first character.
            var firstLetter:String = word.substr(0, 1);
            firstLetter = firstLetter.toUpperCase();
            var otherLetters:String = word.substring(1);
            word = firstLetter + otherLetters;
    }
    return word;
}
```

在英文中，如果是下列的其中一個字，則在標題中這些字的第一個字元就「不」會大寫：“and”、“the”、“in”、“an”、“or”、“at”、“of” 或 “a” (這是簡化的規則)。為執行此邏輯，程式碼會先使用 switch 陳述式，檢查單字是否為其中一個首字母不應大寫的字。如果是的話，程式碼就會跳出 switch 陳述式。相反地，如果單字的首字母應大寫，那麼就會執行下列步驟來改為大寫：

1 使用 substr(0, 1) 擷取單字的第一個字母，這個方法會從索引位置為 0 的字元 (即字串中的第一個字母，因為第一個參數為 0) 開始擷取子字串。子字串的長度為一個字元 (因為第二個參數為 1)。

2 使用 toUpperCase() 方法，將該字母改為大寫。

3 原字的其它字元則使用 substring(1) 擷取，這個方法會從索引位置為 1 的字元 (即第二個字母) 開始擷取子字串，一直擷取到字串的結尾 (因為 substring() 方法的第二個參數沒有指定)。

4 最後會使用字串連接，將剛改為大寫的首字母與其它字母結合來產生單字：firstLetter + otherLetters。

產生 ASCII 藝術文字

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

BitmapToAsciiConverter 類別可將點陣圖影像轉換為其 ASCII 文字形式。這是由 parseBitmapData() 方法執行，其部分內容顯示如下：

```
var result:String = "";

// Loop through the rows of pixels top to bottom:
for (var y:uint = 0; y < _data.height; y += verticalResolution)
{
    // Within each row, loop through pixels left to right:
    for (var x:uint = 0; x < _data.width; x += horizontalResolution)
    {
        ...

        // Convert the gray value in the 0-255 range to a value
        // in the 0-64 range (since that's the number of "shades of
        // gray" in the set of available characters):
        index = Math.floor(grayVal / 4);
        result += palette.charAt(index);
    }
    result += "\n";
}
return result;
```

這段程式碼先定義一個名為 result 的 String 實體，以用來建立點陣圖影像的 ASCII 藝術版本。接下來便循序處理原始點陣圖影像的個別像素。利用多項色彩操作技巧 (因篇幅有限，不多詳述)，程式碼將個別像素的紅、綠和藍色值轉換為單一灰階值 (0 到 255 之間的數字)。接著程式碼再將該值除以 4 (如上所示)，轉換為 0 到 63 色階之間的值，然後將值儲存在變數 index 中 (使用 0-63 色階的原因是，這個應用程式所用 ASCII 字元的「面板」包含 64 個值)。字元面板定義為 BitmapToAsciiConverter 類別中的 String 實體：

```
// The characters are in order from darkest to lightest, so that their
// position (index) in the string corresponds to a relative color value
// (0 = black).
private static const palette:String = "@#$%&8BMW*mwqpdbkhaoQ0OZXYUJCLtfjzxnuvcr[]{}1()|/?Il!i><+_~-;,. ";
```

由於 index 變數會定義面板中的哪一個 ASCII 字元與點陣圖影像中的目前像素相對應，因此便使用 charAt() 方法，從 palette String 擷取該字元。這個字元接著會使用連接指定運算子 (+=)，附加到 result String 實體。此外，在每列像素結尾，會有換行字元連接到 result String 的結尾，以強制換行，產生一列新的字元「像素」。

第 3 章 使用陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

陣列可以讓您在單一資料結構中儲存多個值。您可以使用簡單的索引陣列，以固定的序數整數索引儲存值，或是使用複雜的關聯陣列，以任意鍵儲存值。陣列也可以是多維度的，包含本身也是陣列的元素。最後，您可以使用 Vector 做為其元素都是具有相同資料類型之所有實體的陣列。

更多說明主題

Array

Vector

陣列的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

進行程式設計時，您要處理的往往不是單一物件，而是一組項目。例如，在音樂播放應用程式中，可能會有包含多首歌曲並準備播放的清單。您不會希望為清單上的每首歌曲建立不同的變數，而是希望這些 Song 物件最好能全部結合起來，並且能夠以群組的方式加以處理。

陣列是程式設計的元素，可做為一組項目的容器，如歌曲清單等。在大部分情況下，陣列中的所有項目都是相同類別的實體，但是對 ActionScript 來說，這並非必要條件。陣列中的各個項目稱為陣列的 「元素」。您可以將陣列想成是變數的檔案抽屜。變數可當做元素以新增至陣列中，就像把資料夾放進檔案抽屜一樣。您可以將陣列當做單一變數來使用 (如同將整個抽屜搬移至不同位置)，或將變數當做群組來使用 (就像逐一翻閱資料夾以尋找資訊)，或者是個別存取這些檔案 (如同打開抽屜，選取單一資料夾)。

舉例來說，假設您要建立音樂播放應用程式，讓使用者可以選取多首曲目並新增至播放清單。那麼，在 ActionScript 程式碼中就要有名為 addSongsToPlaylist() 的方法，這個方法會使用單一陣列做為參數。無論要在清單中新增多少歌曲 (少數幾首、多首，或甚至只有一首)，都只要呼叫 addSongsToPlaylist() 方法一次，將內含 Song 物件的陣列傳送過去即可。在 addSongsToPlaylist() 方法中，您可以使用迴圈逐一瀏覽陣列的元素 (歌曲)，並將這些元素實際新增至播放清單。

最常見的 ActionScript 陣列類型是 「索引陣列」。在索引陣列中，每個項目都會儲存在具有編號的位置 (稱為 「索引」)，而編號就像地址一樣，可以用來存取項目。索引陣列的運作方式可以符合大部分程式設計的需求。Array 是用來表示索引陣列的常見類別。

索引陣列時常用於儲存多個相同類型的項目 (本身為相同類別之實體的物件)。Array 類別不表示會限制本身所包含的項目類型。Vector 類別的類型是索引陣列，其中單一陣列內的所有項目都具有相同類型。使用 Vector 實體 (而非 Array 實體) 時，也能改善效能並具有其它優點。從 Flash Player 10 和 Adobe AIR 1.5 開始，Vector 類別已可供使用。

索引陣列的其中一種特殊用法，就是 「多維度陣列」。所謂多維度陣列，就是其元素都是索引陣列的索引陣列 (其中前者又包含其它元素)。

另一種陣列是 「關聯陣列」，這種陣列會以字串 「索引鍵」而非數值索引來識別各個元素。最後，ActionScript 3.0 也包含了 Dictionary 類別，這個類別表示 「字典」。字典是一個陣列，可讓您將任何類型的物件當做索引鍵來使用，以區分各個元素。

重要概念與術語

下列參考清單包含撰寫陣列和向量處理常式時會遇到的相關重要術語：

Array 一種做為容器的物件，可將多個物件群組起來。

陣列存取 ([]) 運算子 一對括住索引或索引鍵 (可唯一識別陣列元素) 的方括號。這個語法使用時會加在陣列變數名稱後方，以指定陣列的單一元素 (而非整個陣列)

關聯陣列 使用字串索引鍵來識別各個元素的陣列。

基底類型 Vector 實體可以儲存的物件資料類型。

Dictionary 其項目是由物件配對 (即索引鍵與值) 所組成的陣列。此索引鍵會取代數值索引來識別單一元素

元素 陣列中的單一項目。

索引 即數值 「位址」，用來識別索引陣列中的單一元素。

索引陣列 標準的陣列類型，它會將每個元素儲存在具有編號的位置，並使用此編號 (索引) 來識別各個元素。

按鍵 在關聯陣列或字典中，用來識別單一元素的字串或物件。

多維陣列 這種陣列所包含的項目是陣列，而非單一值。

T 標準慣例，在本說明文件中用於表示 Vector 實體的基底類型 (無論此基底類型為何都一樣)。T 慣例可用於表示類別名稱，您可以在類型參數的說明中找到相關描述 (「T」表示 「類型」，就像 「資料類型」一樣)。

類型參數 搭配 Vector 類別名稱所使用的語法，以指定 Vector 的基底類型 (此類別所儲存的物件資料類型)。這個語法的開頭為一個點 (.)，後面再接著由角括號 (<>) 括住的資料類型名稱。完成之後，看起來就像：Vector.<T>。在本說明文件中，類型參數內指定的類別通常會表示為 T。

Vector 一種陣列類型，陣列中的所有元素都是具有相同資料類型的實體。

索引陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

索引陣列會儲存一系列一或多個值，這些值經過組織，可以使用無正負號的整數值加以存取。第一個索引一定是數字 0，而每個相繼新增到陣列之元素的索引則會加 1 來遞增。在 ActionScript 3.0 中，會使用 Array 類別和 Vector 類別當做索引陣列。

索引陣列會使用無正負號的 32 位元整數做為索引編號。索引陣列的大小上限為 2^{32} - 1 或 4,294,967,295。嘗試建立大於此上限的陣列會導致發生執行階段錯誤。

若要存取索引陣列的個別元素，您可以使用陣列存取 ([]) 運算子，指定要存取之元素的索引位置。例如，下列程式碼表示陣列索引 (名為 songTitles) 中的第一個陣列 (位於索引 0 的元素)：

```
songTitles[0]
```

陣列變數名稱後面接著括住索引的方括號，這種組合的作用是當做單一識別名稱 (換句話說，變數名稱可以利用各種方式來使用這種組合)。您可以在指定陳述式左邊使用名稱和索引，將某個值指定給索引陣列元素：

```
songTitles[1] = "Symphony No. 5 in D minor";
```

同樣地，您也可以在指定陳述式右邊使用名稱和索引，擷取索引陣列元素的值。

```
var nextSong:String = songTitles[2];
```

此外，還可以在方括號中使用變數，而不需提供明確的值 (變數必須包含非負數整數值，例如 uint、正 int 或正整數 Number 實體)。需要 「重複執行」索引陣列中的元素，或針對部分或所有元素執行某項作業時，往往都會用到這項技巧。下列程式碼將示範這項技巧的操作方式。在程式碼中，會使用迴圈存取 Array 物件 (名為 oddNumbers) 內的每個值。然後，再使用 trace() 陳述式，依照 「oddNumber[索引] = 值」格式輸出每個值：

```
var oddNumbers:Array = [1, 3, 5, 7, 9, 11];
var len:uint = oddNumbers.length;
for (var i:uint = 0; i < len; i++)
{
    trace("oddNumbers[" + i.toString() + "] = " + oddNumbers[i].toString());
}
```

Array 類別

索引陣列的第一個類型是 Array 類別。Array 實體可以存放任何資料類型的值。同一個 Array 物件可以存放不同資料類型的物件。例如，一個 Array 實體可以在索引 0 存放 String 值、在索引 1 存放 Number 實體，以及在索引 2 存放 XML 物件。

Vector 類別

您可以在 ActionScript 3.0 中使用的另一種索引陣列是 Vector 類別。Vector 實體是「類型陣列」，表示 Vector 實體中所有元素的資料類型都相同。

備註：從 Flash Player 10 和 Adobe AIR 1.5 開始，Vector 類別已可供使用。

當您宣告 Vector 變數或實體化 Vector 物件時，必須明確指定 Vector 可包含的物件資料類型。所指定的資料類型稱為 Vector 的「基底類型」。在執行階段和編譯階段 (在嚴謹模式中)，會檢查任何設定 Vector 元素的值或從 Vector 擷取值的程式碼。如果所加入或擷取之物件的資料類型與 Vector 的基底類型不符，便會發生錯誤。

除了資料類型限制之外，Vector 類別的其他限制與 Array 類別還是有所區別的：

- Vector 是密集陣列。Array 物件可以在索引 0 和 7 具有值，即使索引 1 到 6 都沒有值也無妨。不過，Vector 的每個索引則都必須具有值 (或 null)。
- 您可以選擇是否要固定 Vector 長度。這表示，您無法變更 Vector 包含的元素數目。
- 存取 Vector 的元素時，會經過邊界檢查。您永遠都不能讀取大於最後一個元素 (length - 1) 之索引的值。您永遠都不能設定超出目前最後一個索引之索引的值 (換句話說，只能設定位於現有索引或索引 [length] 的值)。

由於這些限制，比起元素都是單一類別實體的 Array 實體，Vector 有三大優點：

- 效能：使用 Vector 實體的陣列元素存取和循環遠比使用 Array 實體時快得多。
- 類型安全：在嚴謹模式中，編譯器可以識別資料類型錯誤。這類錯誤的範例包含指定具有不正確資料類型的值給 Vector，或者讀取來自 Vector 的值時，預期該值具有不正確的資料類型。在執行階段，當加入資料或讀取來自 Vector 物件的資料時，都會檢查資料類型。但是請注意，當您使用 push() 或 unshift() 方法將值加入 Vector 時，在編譯階段並不會檢查引數的資料類型。使用這些方法時，在執行階段仍然會檢查所有的值。
- 可靠性：與 Array 相比，執行階段範圍檢查 (或固定長度檢查) 會大幅增加可靠性。

除了額外的限制和優點之外，Vector 類別和 Array 類別可以說極為類似。Vector 物件的屬性和方法與 Array 的屬性和方法很類似 (在大部分情況下為完全相同)。在您想要使用所有元素都有相同資料類型之 Array 的大部分情況下，Vector 實體是比較好的方式。

建立陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用好幾種技巧來建立 Array 實體或 Vector 實體。不過，建立每一種陣列類型的技巧都有些許不同。

建立 Array 實體

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以呼叫 Array() 建構函式或使用 Array 常值語法，建立 Array 物件。

Array() 建構函數有三種使用方式。第一，假設您呼叫不含引數的建構函式，會得到一個空陣列。您可以使用 Array 類別的 length 屬性來確認陣列中不包含元素。例如，下列程式碼會呼叫 Array() 建構函式，並且不搭配任何引數：

```
var names:Array = new Array();
trace(names.length); // output: 0
```

第二，假設您使用數字做為 Array() 建構函式的唯一參數，則會建立具有該長度的陣列，而且每個元素的值會設定為 undefined。引數必須是介於 0 到 4,294,967,295 之間無正負號的整數。例如，下列程式碼會呼叫 Array() 建構函式，並搭配單一數值引數：

```
var names:Array = new Array(3);
trace(names.length); // output: 3
trace(names[0]); // output: undefined
trace(names[1]); // output: undefined
trace(names[2]); // output: undefined
```

第三，假設您呼叫建構函式並傳遞元素清單做為參數，則會以對應於每個參數的元素來建立陣列。下列程式碼會將三個引數傳遞給 Array() 建構函式：

```
var names:Array = new Array("John", "Jane", "David");
trace(names.length); // output: 3
trace(names[0]); // output: John
trace(names[1]); // output: Jane
trace(names[2]); // output: David
```

您也可以使用 Array 常值來建立陣列。Array 常值可以直接指定給陣列變數，如下列範例所示：

```
var names:Array = ["John", "Jane", "David"];
```

建立 Vector 實體

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

您可以呼叫 Vector.<T>() 建構函式來建立 Vector 實體，也可以呼叫 Vector.<T>() 全域函數來建立 Vector。這個函數會將指定的物件轉換成 Vector 實體。在 Flash Professional CS5 (以及更新的版本)、Flash Builder 4 (以及更新的版本) 以及 Flex 4 (以及更新的版本) 中，您也可以使用 Vector 常值語法建立 Vector 實體。

每當宣告 Vector 變數 (或類似的 Vector 方法參數或方法傳回類型) 時，您都必須指定 Vector 變數的基底類型。當您藉由呼叫 Vector.<T>() 建構函式來建立 Vector 實體，也就是說，每當您在 ActionScript 中使用 Vector 一詞，就必須指定基底類型。

您可以使用類型參數語法，指定 Vector 的基底類型。在程式碼中，類型參數必須緊接在 Vector 這個字之後。這個語法的開頭為一個點 (.)，後面再接著由角括號 (<>) 括住的基底類別名稱，如下列範例所示：

```
var v:Vector.<String>;
v = new Vector.<String>();
```

在範例的第一行中，變數 v 被宣告為 Vector.<String> 實體。換句話說，這個變數就表示只能夠存放 String 實體的索引陣列。第二行會呼叫 Vector() 建構函式，建立具有相同 Vector 類型的實體 (也就是其元素都是 String 物件的 Vector)，並且將這個物件指定給 v。

使用 Vector.<T>() 建構函式

如果您使用 Vector.<T>() 建構函式時未搭配任何引數，那麼就會建立空的 Vector 實體。您可以檢查 Vector 的 length 屬性，測試是否為空實體。例如，下列程式碼會呼叫 Vector.<T>() 建構函式，並且不搭配任何引數：

```
var names:Vector.<String> = new Vector.<String>();
trace(names.length); // output: 0
```

如果您事先知道 Vector 一開始所需的元素數目，即可預先定義 Vector 中的元素數目。若要建立具有數個元素的 Vector，請傳遞元素的數目當做第一個參數 (length 參數)。由於 Vector 的元素都不能是空白，因此這些元素會由基底類型的實體填滿。如果基底類型是允許 null 值的參考類型，則這些元素全部都會包含 null。否則，這些元素全部都會包含該類別的預設值。例如，uint 變數就不能是 null。因此，在下列程式碼中，會建立具有七個元素的 Vector (名為 ages)，其中每個元素都包含 0 這個值：

```
var ages:Vector.<uint> = new Vector.<uint>(7);
trace(ages); // output: 0,0,0,0,0,0,0
```

最後，使用 Vector.<T>() 建構函式，並傳遞 true 做為第二個參數 (fixed 參數) 的值，即可建立長度固定的 Vector。在這種情況下，Vector 在建立時就已經具有指定的元素數目，而且您無法變更這個數目。但是請注意，您還是可以變更長度固定之 Vector 的元素值。

使用 **Vector** 常值語法建構函式

在 Flash Professional CS5 (以及更新的版本)、Flash Builder 4 (以及更新的版本) 以及 Flex 4 (以及更新的版本) 中，您可以傳送值清單給 Vector.<T>() 建構函式，以指定 Vector 一開始的值：

```
// var v:Vector.<T> = new <T>[E0, ..., En-1 ,];
// For example:
var v:Vector.<int> = new <int>[0,1,2,];
```

以下資訊適用於此語法：

- 可省略最後加上的逗點。
- 不支援陣列中空白的項目；像 var v:Vector.<int> = new <int>[0,,2,] 的陳述式會發生編譯器錯誤。
- 您無法指定 Vector 實體的預設長度。相反地，長度會與初始化清單中的元素數目相同。
- 您無法指定 Vector 實體是否為固定長度。但可使用 fixed 屬性。
- 如果以值的形式所傳遞之項目不符合指定的類型，則可能會發生資料遺失或錯誤。例如：

  ```
  var v:Vector.<int> = new <int>[4.2]; // compiler error when running in strict mode
  trace(v[0]); //returns 4 when not running in strict mode
  ```

使用 **Vector.<T>()** 全域函數

除了 Vector.<T>() 和 Vector 常值語法建構函式，您也可以使用 Vector.<T>() 全域函數來建立 Vector 物件。Vector.<T>() 全域函數是轉換函數。當您呼叫 Vector.<T>() 全域函數時，必須指定方法所傳回之 Vector 的基底類型。您可以傳遞單一索引陣列 (Array 或 Vector 實體) 做為引數，然後方法就會傳回具有指定之基底類型的 Vector，其中包含來源陣列引數中的值。下列程式碼會列出呼叫 Vector.<T>() 全域函數所需的語法：

```
var friends:Vector.<String> = Vector.<String>(["Bob", "Larry", "Sarah"]);
```

Vector.<T>() 全域函數會執行兩項資料類型轉換。首先，當 Array 實體傳遞至函數時，便會傳回 Vector 實體。接著，無論來源陣列是 Array 還是 Vector 實體，函數都會嘗試將來源陣列的元素轉換成基底類型的值。轉換時，會使用標準的 ActionScript 資料類型轉換規則。例如，下列程式碼會將來源 Array 中的 String 值轉換成所產生 Vector 中的整數。第一個值 ("1.5") 的小數點部分會遭截斷，而且非數字的第三個值 ("Waffles") 在結果中會轉換成 0：

```
var numbers:Vector.<int> = Vector.<int>(["1.5", "17", "Waffles"]);
trace(numbers); // output: 1,17,0
```

如果來源中有任何元素無法轉換，便會發生錯誤。

當程式碼呼叫 Vector.<T>() 全域函數時，如果來源陣列中某一元素是所指定基底類型之子類別的實體，這個元素就會加入至產生的 Vector (不會發生錯誤)。使用 Vector.<T>() 全域函數是將基底類型為 T 的 Vector 轉換成基底類型為 T 之子類別的 Vector 的唯一方式。

插入陣列元素

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要在索引陣列中加入元素，最基本的方式就是使用陣列存取 ([]) 運算子。若要設定索引陣列元素的值，請在指定陳述式左邊使用 Array 或 Vector 物件名稱和索引：

```
songTitles[5] = "Happy Birthday";
```

如果 Array 或 Vector 本來就沒有該索引及在該處存放的元素，則會建立該索引並於該處儲存這個值。如果該索引已經有值，則新的值會取代現有的值。

Array 物件可讓您在任何索引建立元素。不過如果是 Vector 物件，您就只能將值指定給現有的索引，或者指定給下一個可用的索引。下一個可用的索引會對應至 Vector 物件的 length 屬性。若要在 Vector 物件中加入新元素，最安全的方式就是使用如下所示的程式碼：

```
myVector[myVector.length] = valueToAdd;
```

您可以利用 push()、unshift() 和 splice() 這三種 Array 和 Vector 類別方法，將元素插入索引陣列中。push() 方法會將一或多個元素附加到陣列結尾。換句話說，以 push() 方法插入的最後一個元素具有最高的索引編號。unshift() 方法會在陣列開頭 (索引編號一定是 0) 插入一或多個元素；而 splice() 方法則會在陣列的指定索引位置插入任意數量的項目。

下列範例將示範這三種方法。在範例中，會建立名為 planets 的陣列，依照距離 「太陽」的遠近順序儲存行星名稱。首先，會呼叫 push() 方法加入第一個項目 Mars。其次，呼叫 unshift() 方法，插入屬於陣列前方的項目 Mercury。最後，則會呼叫 splice() 方法將 Venus 和 Earth 項目插入 Mercury 之後、Mars 之前。傳送給 splice() 的第一個引數 (整數 1) 會指示從索引 1 開始插入；傳送給 splice() 的第二個引數 (整數 0) 則會指示不要刪除任何項目；傳送給 splice() 的第三和第四個引數 Venus 和 Earth 則是要插入的項目。

```
var planets:Array = new Array();
planets.push("Mars"); // array contents: Mars
planets.unshift("Mercury"); // array contents: Mercury,Mars
planets.splice(1, 0, "Venus", "Earth");
trace(planets); // array contents: Mercury,Venus,Earth,Mars
```

push() 和 unshift() 方法都會傳回無正負號的整數，代表修改過的陣列長度。以 splice() 方法來插入元素時會傳回空陣列，乍看之下好像有點奇怪，但是從 splice() 方法的多功能性來看，就顯得相當合理。您不僅可以使用 splice() 方法將元素插入陣列，也能移除陣列中的元素。當您使用 splice() 方法移除元素時，會傳回包含所移除元素的陣列。

備註：如果 Vector 物件的 fixed 屬性為 true，Vector 中的元素總數就無法變更。如果您嘗試使用上述技巧在固定長度的 Vector 中加入新元素，就會發生錯誤。

擷取值以及移除陣列元素

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要從索引陣列擷取元素的值，最簡單的方式就是使用陣列存取 ([]) 運算子。若要擷取索引陣列元素的值，請在指定陳述式右邊使用 Array 或 Vector 物件名稱和索引：

```
var myFavoriteSong:String = songTitles[3];
```

如果某一索引沒有元素，您仍然可以嘗試使用該索引從 Array 或 Vector 擷取值。在這種情況下，Array 物件會傳回 undefined 值，而且 Vector 會擲回 RangeError 例外。

您可以利用 pop()、shift() 和 splice() 這三種 Array 和 Vector 類別方法來移除元素。pop() 方法會從陣列結尾移除元素。換句話說，它會移除索引編號最高的元素。shift() 方法會從陣列開頭移除元素，也就是說，它固定會移除索引編號為 0 的元素。可用來插入元素的 splice() 方法，則會從傳送至這個方法的第一個引數所指定的索引編號開始，移除任意數目的元數。

下列範例會使用這三種方法來移除 Array 實體中的元素。在範例中，會建立名為 oceans 的 Array 來儲存大型水域的名稱。Array 中有些名稱是湖泊的名稱，而非海洋的名稱，因此這些屬於湖泊的名稱必須加以移除。

首先，會使用 splice() 方法移除 Aral 和 Superior 項目，並插入 Atlantic 和 Indian 這兩個項目。傳送至 splice() 的第一個引數 (整數 2) 會指出這項作業是從清單的第三個項目開始，也就是位於索引 2 的項目。第二個引數 2 則會指出要移除二個項目。其它引數 Atlantic 和 Indian 則是要插入索引 2 位置的值。

接下來，會使用 pop() 方法移除陣列中的最後一個元素 Huron。之後，會使用 shift() 方法移除陣列的第一個項目 Victoria。

```
var oceans:Array = ["Victoria", "Pacific", "Aral", "Superior", "Indian", "Huron"];
oceans.splice(2, 2, "Arctic", "Atlantic"); // replaces Aral and Superior
oceans.pop(); // removes Huron
oceans.shift(); // removes Victoria
trace(oceans);// output: Pacific,Arctic,Atlantic,Indian
```

pop() 和 shift() 方法都會傳回所移除的項目。對 Array 實體而言，傳回值的資料類型是 Object，因為陣列可以保留任何資料類型的值。對於 Vector 實體而言，傳回值的資料類型則是 Vector 的基底類型。splice() 方法會傳回包含所移除之值的 Array 或 Vector。您可以變更 oceans Array 範例，這樣對 splice() 的呼叫就會將傳回的 Array 指定給新的 Array 變數，如下列範例所示：

```
var lakes:Array = oceans.splice(2, 2, "Arctic", "Atlantic");
trace(lakes); // output: Aral,Superior
```

您可能會遇到針對 Array 物件元素使用 delete 運算子的程式碼。delete 運算子會將 Array 元素的值設定為 undefined，但是不會將元素從 Array 中移除。例如，下列程式碼會將 delete 運算子用於 oceans Array 的第三個元素，但是 Array 的長度仍然是 5：

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Indian", "Atlantic"];
delete oceans[2];
trace(oceans);// output: Arctic,Pacific,,Indian,Atlantic
trace(oceans[2]); // output: undefined
trace(oceans.length); // output: 5
```

您可以使用陣列的 length 屬性截斷 Array 或 Vector。如果您為索引陣列之 length 屬性設定的長度比這個陣列的目前長度還小，這個陣列便會遭截斷，而且大於新的 length 值減 1 之索引編號的元素會遭移除。例如，如果 oceans 陣列在經過排序之後，所有有效的項目都位於陣列的開頭，您即可使用 length 屬性移除位於陣列結尾的項目，如下列程式碼所示：

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];
oceans.length = 2;
trace(oceans); // output: Arctic,Pacific
```

備註：如果 Vector 物件的 fixed 屬性為 true，Vector 中的元素總數就無法變更。如果您嘗試使用上述技巧在固定長度的 Vector 中移除元素或截斷陣列，就會發生錯誤。

排序陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 reverse()、sort() 及 sortOn() 這三種方法，藉由排序或反向順序排序來變更索引陣列的順序。這三種方法都會修改現有的陣列。下表摘要列出這些方法，以及這些方法針對 Array 和 Vector 物件的行為：

方法	針對 **Array** 的行為	針對 **Vector** 的行為
reverse()	變更元素的順序，使最後一個元素變成第一個元素，倒數第二個元素變成第二個元素，依此類推	與針對 Array 的行為相同
sort()	可讓您依照各種預先定義的方式 (例如，按照字母或數字順序)，為 Array 的元素排序。您也可以指定自訂排序演算法	根據指定的自訂排序演算法，為元素排序
sortOn()	可讓您為具有一個或多個常用屬性的物件排序，以指定一個或多個屬性當做排序索引鍵使用	在 Vector 類別中無法使用

reverse() 方法

reverse() 方法既不使用參數也不傳回值，但是可以讓您將陣列順序從目前的狀態切換為反向順序。下列範例會反轉 oceans 陣列所列的海洋順序：

```
var oceans:Array = ["Arctic", "Atlantic", "Indian", "Pacific"];
oceans.reverse();
trace(oceans); // output: Pacific,Indian,Atlantic,Arctic
```

使用 **sort()** 方法進行基本排序 **(** 僅適用於 **Array** 類別 **)**

針對 Array 實體，sort() 方法會使用 「預設的排序順序」重新排列陣列中的元素。預設的排序順序具有下列特性：

- 排序動作有區分大小寫，也就是說，大寫字元會排在小寫字元前面。例如，字母 D 會排在字母 b 之前。
- 排序動作採取升冪方式，也就是說，較低的字元碼 (例如 A) 會排在較高字元碼的前面 (例如 B)。
- 排序動作會將相同的值排在相鄰位置，但沒有一定的順序。
- 排序是以字串為基礎，這表示元素在進行比較前，會先轉換成字串 (例如，10 會排在 3 之前，因為字串 "1" 的字元碼較字串 "3" 低)。

您可能需要忽略大小寫限制為 Array 排序，或以遞減順序排序，或者，您可能要以陣列所包含的數字 (而非按英文字母順序) 來排序。只要利用 Array 類別之 sort() 方法的 options 參數，您就可以變更預設排序順序的每個特徵。這些選項是由 Array 類別中的一組靜態常數所定義，如下列清單所示：

- Array.CASEINSENSITIVE：這個選項會忽略大小寫限制加以排序。例如，小寫字母 b 會排在大寫字母 D 之前。
- Array.DESCENDING：這會反轉預設的依遞增順序排序。例如，字母 B 會排在字母 A 之前。
- Array.UNIQUESORT：這會在發現兩個相同的值時，中止排序作業。
- Array.NUMERIC：這會進行數值排序，因此 3 會排在 10 之前。

下列範例會加強說明上述幾個選項。在範例中，會建立名為 poets 的 Array，並以幾個不同的選項加以排序。

```
var poets:Array = ["Blake", "cummings", "Angelou", "Dante"];
poets.sort(); // default sort
trace(poets); // output: Angelou,Blake,Dante,cummings

poets.sort(Array.CASEINSENSITIVE);
trace(poets); // output: Angelou,Blake,cummings,Dante

poets.sort(Array.DESCENDING);
trace(poets); // output: cummings,Dante,Blake,Angelou

poets.sort(Array.DESCENDING | Array.CASEINSENSITIVE); // use two options
trace(poets); // output: Dante,cummings,Blake,Angelou
```

使用 **sort()** 方法進行自訂排序 **(** 適用於 **Array** 和 **Vector** 類別 **)**

除了適用於 Array 物件的基本排序方式以外，您還可以定義自訂排序規則。這項技巧是唯一一個能夠透過 sort() 方法使用 Vector 類別的方式。若要定義自訂排序，請撰寫自訂排序函數，並將它傳遞至 sort() 方法當做引數。

例如，假設您有一份名稱清單，其中每個清單元素都包含人員的姓氏和名字，若您要依照姓氏排序清單，就必須使用自訂排序函數剖析每個元素，並在排序函數中使用姓氏。下列程式碼將說明如何以自訂函數進行上述排序，而此自訂函數是當做參數傳遞給 Array.sort() 方法：

```
var names:Array = new Array("John Q. Smith", "Jane Doe", "Mike Jones");
function orderLastName(a, b):int
{
    var lastName:RegExp = /\b\S+$/;
    var name1 = a.match(lastName);
    var name2 = b.match(lastName);
    if (name1 < name2)
    {
        return -1;
    }
    else if (name1 > name2)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
trace(names); // output: John Q. Smith,Jane Doe,Mike Jones
names.sort(orderLastName);
trace(names); // output: Jane Doe,Mike Jones,John Q. Smith
```

自訂排序函數 orderLastName() 會使用規則運算式擷取每個元素的姓氏，以用於比較運算作業。針對 names 陣列呼叫 sort() 方法時，是以函數的識別名稱 orderLastName 做為唯一參數。排序函數會採用 a 和 b 參數，因為它能同時作用於兩種陣列元素。排序函數的傳回值會指出元素是以何種方式排序：

- 傳回值 -1 表示第一個參數 a 是在第二個參數 b 之前。
- 傳回值 1 表示第二個參數 b 是在第一個參數 a 之前。
- 傳回值 0 指出元素都具有相同的排序優先順序。

sortOn() 方法 **(** 僅適用於 **Array** 類別 **)**

sortOn() 方法是針對具有內含物件之元素的 Array 物件所設計。這些物件預期會有至少一個共同屬性，以做為排序索引鍵。將 sortOn() 方法用於其它類型的陣列時，會產生無法預期的結果。

備註：Vector 類別並不包括 sortOn() 方法。這個方法只適用於 Array 物件。

下列範例會修改 poets Array，因此每個元素都會是物件而非字串。每個物件都保留了詩人的姓氏和出生年份。

```
var poets:Array = new Array();
poets.push({name:"Angelou", born:"1928"});
poets.push({name:"Blake", born:"1757"});
poets.push({name:"cummings", born:"1894"});
poets.push({name:"Dante", born:"1265"});
poets.push({name:"Wang", born:"701"});
```

您可以使用 sortOn() 方法，依照 born 屬性為 Array 排序。sortOn() 方法會定義 fieldName 和 options 參數。指定 fieldName 引數時，必須使用字串的形式。在下列範例中，會搭配兩個引數 born 和 Array.NUMERIC 呼叫 sortOn()。Array.NUMERIC 引數是用來確認排序是按數值順序，而非字母順序進行。即使當所有數值都具有相同數字時，這仍是相當好的作法，因為當您在稍後將具有較多或較少數字的數值加入陣列時，這種方法可以確保排序仍依照預期方式進行。

```
poets.sortOn("born", Array.NUMERIC);
for (var i:int = 0; i < poets.length; ++i)
{
    trace(poets[i].name, poets[i].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

在不修改原始陣列的情況下進行排序 **(** 僅適用於 **Array** 類別 **)**

一般來說，sort() 和 sortOn() 方法都會修改 Array。若您希望排序 Array 但不修改現有陣列，請傳遞 Array.RETURNINDEXEDARRAY 常數做為 options 參數的一部分。這個選項會指示方法傳回能夠反應排序的新 Array，並將原始 Array 保留不變。方法所傳回的 Array 是可以反映新排序順序的索引編號簡單 Array，其中不包含原始 Array 的任何元素。例如，若要依出生年份排序 poets Array 而不修改 Array，請將 Array.RETURNINDEXEDARRAY 常數納入要傳遞做為 options 參數之引數的一部分。

下列範例會將傳回的索引資訊儲存在名為 indices 的 Array 中，並使用 indices 陣列和未修改的 poets 陣列依出生年份輸出詩人名稱：

```
var indices:Array;
indices = poets.sortOn("born", Array.NUMERIC | Array.RETURNINDEXEDARRAY);
for (var i:int = 0; i < indices.length; ++i)
{
    var index:int = indices[i];
    trace(poets[index].name, poets[index].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

查詢陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

concat()、join()、slice() 和 toString() 四種 Array 和 Vector 類別的方法都會向陣列查詢資訊，但是不會修改陣列。concat() 和 slice() 方法會傳回新的陣列，而 join() 和 toString() 方法則會傳回字串。concat() 方法會採用新的陣列或元素清單做為引數，並將它合併至現有陣列以建立新的陣列。slice() 方法有兩個參數，即名符其實的 startIndex 和 endIndex，它們會傳回新的陣列，當中包含自現有陣列「分割」出來的元素副本。此分割是從 startIndex 的元素開始，並結束於 endIndex 之前的元素。在此要重複提醒的是，位於 endIndex 的元素不會包含在傳回值中。

下列範例會使用 concat() 和 slice()，利用其它陣列的元素來建立新的陣列：

```
var array1:Array = ["alpha", "beta"];
var array2:Array = array1.concat("gamma", "delta");
trace(array2); // output: alpha,beta,gamma,delta

var array3:Array = array1.concat(array2);
trace(array3); // output: alpha,beta,alpha,beta,gamma,delta

var array4:Array = array3.slice(2,5);
trace(array4); // output: alpha,beta,gamma
```

您可以使用 join() 和 toString() 方法來查詢陣列，並傳回陣列內容做為字串。如果沒有對 join() 方法使用任何參數，這兩種方法的行為方式是相同的，亦即它們都會傳回字串，其中包含陣列內以逗點分隔的所有元素清單。join() 方法與 toString() 方法不同，前者會採用名為 delimiter 的參數，可以讓您選擇符號，以做為傳回字串中每個元素之間的分隔符號。

下列範例會建立名為 rivers 的 Array，並同時呼叫 join() 和 toString() 以字串型式傳回 Array 中的值。使用 toString() 方法可傳回以逗點分隔的值 (riverCSV)，而使用 join() 方法則可傳回以 + 字元分隔的值。

```
var rivers:Array = ["Nile", "Amazon", "Yangtze", "Mississippi"];
var riverCSV:String = rivers.toString();
trace(riverCSV); // output: Nile,Amazon,Yangtze,Mississippi
var riverPSV:String = rivers.join("+");
trace(riverPSV); // output: Nile+Amazon+Yangtze+Mississippi
```

有關 join() 方法要注意的一點是，不管您為主要陣列元素指定了何種分隔符號，所有巢狀 Array 或 Vector 實體一律都會傳回以逗點分隔的值，如下列範例所示：

```
var nested:Array = ["b","c","d"];
var letters:Array = ["a",nested,"e"];
var joined:String = letters.join("+");
trace(joined); // output: a+b,c,d+e
```

關聯陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

關聯陣列有時也稱為 「雜湊」或 「對應」，它會使用 「索引鍵」而非數值索引來組織儲存的值。關聯陣列中的每個索引鍵都是唯一的字串，可用來存取儲存的值。關聯陣列是 Object 類別的實體，也就是說，每個索引鍵都會對應到一個屬性名稱。關聯陣列是索引鍵與值配對的無順序集合。程式碼不應該寄望關聯陣列的索引鍵會有一定的順序。

ActionScript 3.0 也包括名為 「字典」的進階類型關聯陣列。字典是 Dictionary 類別在 flash.utils 套件中的實體，它所使用的索引鍵可以是任何資料類型。換句話說，字典的索引鍵不限於 String 類型的值。

具有字串索引鍵的關鍵陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

有兩種方式可以在 ActionScript 3.0 中建立關聯陣列，第一種方式是使用 Object 實體。使用 Object 實體，您即可以物件常值初始化陣列。Object 類別的實體 (也稱為一般物件) 在功能上和關聯陣列完全相同。一般物件的每個屬性名稱都能當做索引鍵，可用來存取儲存的值。

下列範例會建立名為 monitorInfo 的關聯陣列，並使用物件常值搭配兩個索引鍵和值配對來初始化這個陣列：

```
var monitorInfo:Object = {type:"Flat Panel", resolution:"1600 x 1200"};
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

如果您不需要在宣告階段初始化陣列，則可以使用 Object 建構函式來建立陣列，如下所示：

```
var monitorInfo:Object = new Object();
```

使用物件常值或 Object 類別建構函式建立陣列之後，您可以使用陣列存取 ([]) 運算子或點運算子 (.)。下列範例會將兩個新的值加入 monitorArray：

```
monitorInfo["aspect ratio"] = "16:10"; // bad form, do not use spaces
monitorInfo.colors = "16.7 million";
trace(monitorInfo["aspect ratio"], monitorInfo.colors);
// output: 16:10 16.7 million
```

請注意，名為 aspect ratio 的索引鍵包含空格字元。這對於陣列存取 ([]) 運算子是可行的，但要是使用點運算子，就會產生錯誤。因此，建議您不要在索引鍵名稱中使用空格。

第二種建立關聯陣列的方式就是使用 Array 建構函式 (或任何動態類別的建構函式)，然後再使用陣列存取 ([]) 運算子或點運算子 (.)，將索引鍵與值配對加入陣列中。如果將關聯陣列宣告為 Array 類型，就無法使用物件常值來初始化陣列。下列範例會使用 Array 建構函式來建立名為 monitorInfo 的關聯陣列，然後將 type 索引鍵和 resolution 索引鍵，連同其各自的值一併加入該陣列中：

```
var monitorInfo:Array = new Array();
monitorInfo["type"] = "Flat Panel";
monitorInfo["resolution"] = "1600 x 1200";
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

使用 Array 建構函式來建立關聯陣列，並不具備任何優勢。即使您使用 Array 建構函式或 Array 資料類型，也無法搭配關聯陣列使用 Array.length 屬性或任何 Array 類別的方法。由此可見，Array 建構函式最適合用來建立索引陣列。

具有物件索引鍵的關聯陣列 (字典)

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 Dictionary 類別來建立關聯陣列，此陣列會使用物件而非字串做為索引鍵。這種陣列有時也稱為字典、雜湊或對應。舉例來說，就好比一個應用程式可以根據 Sprite 物件與特定容器的關聯來找到該物件的位置一樣。您可以使用 Dictionary 物件，將每個 Sprite 物件對應到容器。

下列程式碼會建立 Sprite 類別的三個實體，做為 Dictionary 物件的索引值。在程式碼中，會為每個索引鍵分別指定 GroupA 或 GroupB 的值。這些值可以是任何資料類型，但是在本範例中，GroupA 和 GroupB 都是 Object 類別的實體。接著，您可以使用陣列存取 ([]) 運算子，存取和每個索引鍵相關聯的值，如下列程式碼所示：

```
import flash.display.Sprite;
import flash.utils.Dictionary;

var groupMap:Dictionary = new Dictionary();

// objects to use as keys
var spr1:Sprite = new Sprite();
var spr2:Sprite = new Sprite();
var spr3:Sprite = new Sprite();

// objects to use as values
var groupA:Object = new Object();
var groupB:Object = new Object();

// Create new key-value pairs in dictionary.
groupMap[spr1] = groupA;
groupMap[spr2] = groupB;
groupMap[spr3] = groupB;

if (groupMap[spr1] == groupA)
{
    trace("spr1 is in groupA");
}
if (groupMap[spr2] == groupB)
{
    trace("spr2 is in groupB");
}
if (groupMap[spr3] == groupB)
{
    trace("spr3 is in groupB");
}
```

重複物件索引鍵

您可以使用 for..in 迴圈或 for each..in 迴圈，重複執行 Dictionary 物件的內容。for..in 迴圈可以讓您根據索引鍵來重複執行，而 for each..in 迴圈則可以讓您根據與每個索引鍵相關聯的值來重複執行。

您可以使用 for..in 迴圈，直接存取 Dictionary 物件的物件索引鍵。您也可以使用陣列存取 ([]) 運算子，存取 Dictionary 物件的值。下列程式碼會使用之前的 groupMap 字典範例，顯示如何以 for..in 迴圈重複執行 Dictionary 物件：

```
for (var key:Object in groupMap)
{
    trace(key, groupMap[key]);
}
/* output:
[object Sprite] [object Object]
[object Sprite] [object Object]
[object Sprite] [object Object]
*/
```

您可以使用 for each..in 迴圈，直接存取 Dictionary 物件的值。下列程式碼也會使用 groupMap 字典，說明如何以 for each..in 迴圈重複執行 Dictionary 物件：

```
for each (var item:Object in groupMap)
{
    trace(item);
}
/* output:
[object Object]
[object Object]
[object Object]
*/
```

物件索引鍵和記憶體管理

Adobe® Flash® Player 和 Adobe® AIR™ 都會使用記憶體回收系統，復原不再使用的記憶體。當物件沒有指向本身的參考時，即可進行記憶體回收，而記憶體將在下次記憶體回收系統執行時復原。例如，下列程式碼會建立新的物件，並且將物件的參考指定給 myObject 變數：

```
var myObject:Object = new Object();
```

只要有任何物件的參考存在，記憶體回收系統就不會復原該物件所佔用的記憶體。如果變更 myObject 的值使它指向不同的物件，或是設定為 null 值，原始物件所佔的記憶體就可以進行記憶體回收，但前提是，該原始物件已沒有其它參考。

如果您將 myObject 當做 Dictionary 物件中的索引鍵，表示您正為原始物件建立另一個參考。例如，下列程式碼會為物件建立兩個參考，它們分別是 myObject 變數以及 myMap 物件中的索引鍵：

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary();
myMap[myObject] = "foo";
```

若要使 myObject 所參考的物件可進行記憶體回收，您必須移除該物件的所有參考。在這種情況下，您必須變更 myObject 的值，並刪除 myMap 中的 myObject 索引鍵，如下列程式碼所示：

```
myObject = null;
delete myMap[myObject];
```

或者，您也可以使用 Dictionary 建構函式的 useWeakReference 參數，將所有的字典索引鍵變成 「弱參考」。記憶體回收系統會忽略弱參考，這表示只具有弱參考的物件可以進行記憶體回收。例如在下列程式碼中，您不需要為了使物件可進行記憶體回收而刪除 myMap 中的 myObject 索引鍵：

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary(true);
myMap[myObject] = "foo";
myObject = null; // Make object eligible for garbage collection.
```

多維度陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

多維度陣列包含其它做為元素的陣列。舉例來說，假如有一份工作清單，其儲存為字串的索引陣列：

```
var tasks:Array = ["wash dishes", "take out trash"];
```

如果要為一週中每一天的工作個別儲存一份清單，可以建立多維度陣列，讓一週中的每一天各用一個元素來代表。每個元素都包含一個類似於 tasks 陣列的索引陣列，以儲存工作清單。您可以在多維度陣列中，使用任意組合的索引或關聯陣列。下一節的範例會使用兩個索引陣列，或是索引陣列的一個關聯陣列。您也可以嘗試其它組合做為練習。

兩個索引陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您使用兩個索引陣列時，可以用表格或試算表將結果具體呈現出來。第一個陣列的元素代表表格的列，而第二個陣列的元素則代表欄。

例如，下列多維度陣列會使用兩個索引陣列來追蹤一週中每一天的工作清單。第一個陣列 masterTaskList 是使用 Array 類別建構函式所建立。陣列中的每個元素都代表一週中的一天，其中索引 0 代表星期一，索引 6 代表星期日。我們可以把這些元素想像成表格中的列。您可以為 masterTaskList 陣列中所建立的七個元素各指定一個陣列常值，以建立每天的工作清單。這些陣列常值都代表表格中的欄。

```
var masterTaskList:Array = new Array();
masterTaskList[0] = ["wash dishes", "take out trash"];
masterTaskList[1] = ["wash dishes", "pay bills"];
masterTaskList[2] = ["wash dishes", "dentist", "wash dog"];
masterTaskList[3] = ["wash dishes"];
masterTaskList[4] = ["wash dishes", "clean house"];
masterTaskList[5] = ["wash dishes", "wash car", "pay rent"];
masterTaskList[6] = ["mow lawn", "fix chair"];
```

您可以使用陣列存取 ([]) 運算子，存取任何工作清單中的個別項目。第一組方括號代表一週中的日子，而第二組方括號則代表當天的工作清單。例如，若要擷取星期三清單中的第二項工作，請先使用代表星期三的索引 2，再使用代表第二項工作的索引 1。

```
trace(masterTaskList[2][1]); // output: dentist
```

若要擷取星期天清單的第一項工作，請先使用代表星期天的索引 6，再使用代表第一項工作的索引 0。

```
trace(masterTaskList[6][0]); // output: mow lawn
```

具有索引陣列的關聯陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

為使個別陣列變得更容易存取，您可以將關聯陣列用於一週的日子，將索引陣列用於工作清單。使用關聯陣列，可讓您在要參考一週中的特定日子時使用點語法，但是在存取關聯陣列的每個元素時，就需要更多的執行處理時間。下列範例會使用關聯陣列做為工作清單的基礎，將索引鍵和值配對用於一週中的每一天：

```
var masterTaskList:Object = new Object();
masterTaskList["Monday"] = ["wash dishes", "take out trash"];
masterTaskList["Tuesday"] = ["wash dishes", "pay bills"];
masterTaskList["Wednesday"] = ["wash dishes", "dentist", "wash dog"];
masterTaskList["Thursday"] = ["wash dishes"];
masterTaskList["Friday"] = ["wash dishes", "clean house"];
masterTaskList["Saturday"] = ["wash dishes", "wash car", "pay rent"];
masterTaskList["Sunday"] = ["mow lawn", "fix chair"];
```

點語法藉由避免使用多組方括號，使得程式碼更容易閱讀。

```
trace(masterTaskList.Wednesday[1]); // output: dentist
trace(masterTaskList.Sunday[0]);// output: mow lawn
```

您可以使用 for..in 迴圈重複執行工作清單，但是必須使用陣列存取 ([]) 運算子 (而非點語法) 來存取與每個索引鍵相關聯的值。由於 masterTaskList 是關聯陣列，因此元素不一定會按照您所預期的順序加以擷取，如下列範例所示：

```
for (var day:String in masterTaskList)
{
    trace(day + ": " + masterTaskList[day])
}
/* output:
Sunday: mow lawn,fix chair
Wednesday: wash dishes,dentist,wash dog
Friday: wash dishes,clean house
Thursday: wash dishes
Monday: wash dishes,take out trash
Saturday: wash dishes,wash car,pay rent
Tuesday: wash dishes,pay bills
*/
```

複製陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Array 類別沒有內建的方法可製作陣列副本。您可以呼叫 concat() 或 slice() 方法且不搭配引數，建立陣列的 「表層副本」。在表層副本中，如果原始陣列具有物件元素，那麼只會複製物件參考，而不會複製物件本身。副本和原始陣列都會指向相同的物件。對物件所做的任何變更，都會反應在這兩個陣列中。

在 「深層副本」中，任何在原始陣列中找到的物件也會同時加以複製，因此新的陣列和原始陣列並不會指向相同的物件。深層複製需要一行以上的程式碼，這通常需要建立函數。這種函數可以建立為全域公用程式函數，或是 Array 子類別的方法。

下列範例會定義名為 clone() 的函數，以執行深層複製。此演算法是借自一般的 Java 程式設計技術。此函數會將陣列序列化至 ByteArray 類別的實體中以建立深層副本，然後將陣列讀回新的陣列中。此函數會採用物件，這樣就能與索引陣列和關聯陣列搭配使用，如下列程式碼所示：

```
import flash.utils.ByteArray;

function clone(source:Object):*
{
    var myBA:ByteArray = new ByteArray();
    myBA.writeObject(source);
    myBA.position = 0;
    return(myBA.readObject());
}
```

擴充 Array 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Array 類別是少數並非 final 的核心類別之一，這表示您可以建立自己的 Array 子類別。本節範例說明如何建立 Array 子類別，並討論幾個可能會在處理過程中產生的問題。

如同前面已敘述，ActionScript 中的陣列並非類別陣列，但是您可以建立 Array 子類別，只接受特定資料類型的元素。下一節的範例會定義名為 TypedArray 的 Array 子類別，此一子類別會將其元素限制為第一個參數所指定之資料類型的值。TypedArray 類別只是為了說明如何擴充 Array 類別而提供的範例，基於下列幾個原因，可能不適用於生產目的：第一，類型檢查是在執行階段而非編譯階段發生。第二，當 TypedArray 方法遇到不符的情況時，雖然此方法可輕易加以修改以擲回例外，但是卻不會忽略該情況，而且也不會擲回例外。第三，此類別無法避免使用陣列存取運算子，將任何類型的值插入陣列中。第四，編寫樣式偏重於簡化而非效能最佳化。

備註：您可以使用上述技巧來建立類型陣列，但是使用 Vector 物件會是比較好的方式。Vector 實體則是真正的類型陣列，所提供的效能與其它方面都比 Array 類別或任何子類別改善了許多。在此進行討論的目的，是要示範如何建立 Array 子類別。

宣告子類別

您可以使用 extends 關鍵字，指出某個類別為 Array 的子類別。Array 的子類別應該和 Array 類別一樣都使用 dynamic 特質。否則，您的子類別將無法正常運作。

下列程式碼會顯示 TypedArray 類別的定義，此類別包含一常數，可用來保留資料類型、建構函式方法，以及四種能夠將元素新增至陣列的方法。這個範例雖然省略了每個方法的程式碼，但是在下一節中有充分而詳細的說明：

```
public dynamic class TypedArray extends Array
{
    private const dataType:Class;

    public function TypedArray(...args) {}

    AS3 override function concat(...args):Array {}

    AS3 override function push(...args):uint {}

    AS3 override function splice(...args) {}

    AS3 override function unshift(...args):uint {}
}
```

這四種覆寫方法都會使用 AS3 命名空間而非 public 特質，因為此範例會假設編譯器選項 -as3 已設定為 true，而且編譯器選項 -es 已設定為 false。這些都是 Adobe Flash Builder 和 AdobeFlashProfessional 的預設設定。

若您是偏好使用原型繼承的進階開發人員，可以對 TypedArray 類別做兩個小變更，使它在編譯器選項 -es 設定為 true 的情況下進行編譯。首先移除所有的 override 特質，並將 AS3 命名空間更換為 public 特質。其次，以 Array.prototype 代替出現四次的 super。

TypedArray 建構函式

子類別建構函式提出相當有趣的挑戰，因為建構函式必須接受任意長度的引數清單。這個挑戰是如何將引數傳遞至父建構函式以建立陣列。如果您將引數清單當做陣列來傳遞，父建構函式會認為它是類型 Array 的單一引數，所產生的陣列永遠只有 1 個元素的長度。處理傳遞引數清單的傳統方式是使用 Function.apply() 方法，這種方法會將引數陣列當做其第二個參數，但是會在執行函數時，將它轉換為引數清單。遺憾的是，Function.apply() 方法不能搭配建構函式使用。

所剩下的唯一選項是在 TypedArray 建構函式中，重新建立 Array 建構函式的邏輯。下列程式碼會顯示 Array 類別建構函式所使用的演算法，您可以將此演算法重新用於 Array 子類別建構函式中：

```
public dynamic class Array
{
    public function Array(...args)
    {
        var n:uint = args.length
        if (n == 1 && (args[0] is Number))
        {
            var dlen:Number = args[0];
            var ulen:uint = dlen;
            if (ulen != dlen)
            {
                throw new RangeError("Array index is not a 32-bit unsigned integer ("+dlen+")");
            }
            length = ulen;
        }
        else
        {
            length = n;
            for (var i:int=0; i < n; i++)
            {
                this[i] = args[i]
            }
        }
    }
}
```

TypedArray 建構函式會共用 Array 建構函式大部分的程式碼，其中只有四個地方不同。首先，參數清單包含新加入之類別為 Class 的必要參數，可讓您指定陣列的資料類型。第二，傳遞至建構函式的資料類型會指定給 dataType 變數。第三，在 else 陳述式中，length 屬性的值是指定在 for 迴圈之後，因此 length 只包含適當類型的引數。第四，for 迴圈的主體會使用 push() 方法的覆寫版本，因此只有正確資料類型的引數才會新增至陣列中。下列範例會顯示 TypedArray 建構函數：

```
public dynamic class TypedArray extends Array
{
    private var dataType:Class;
    public function TypedArray(typeParam:Class, ...args)
    {
        dataType = typeParam;
        var n:uint = args.length
        if (n == 1 && (args[0] is Number))
        {
            var dlen:Number = args[0];
            var ulen:uint = dlen
            if (ulen != dlen)
            {
                throw new RangeError("Array index is not a 32-bit unsigned integer ("+dlen+")")
            }
            length = ulen;
        }
        else
        {
            for (var i:int=0; i < n; i++)
            {
                // type check done in push()
                this.push(args[i])
            }
            length = this.length;
        }
    }
}
```

TypedArray 覆寫方法

TypedArray 類別會覆寫 Array 類別的四種方法，而這四種方法都能新增元素到陣列中。在每種情況中，覆寫方法都會新增類型檢查，以防止新增資料類型不正確的元素。接著，每種方法都會呼叫其本身的父類別版本。

push() 方法會以 for..in 迴圈重複執行引數清單，並針對每個引數執行類型檢查。類型不正確的引數會以 splice() 方法從 args 陣列中加以移除。for..in 迴圈結束之後，args 陣列便僅包含類型為 dataType 的值。接著會以更新過的 args 陣列呼叫 push() 的父類別版本，如下列程式碼所示：

```
AS3 override function push(...args):uint
{
    for (var i:* in args)
    {
        if (!(args[i] is dataType))
        {
            args.splice(i,1);
        }
    }
    return (super.push.apply(this, args));
}
```

concat() 方法會建立名為 passArgs 的暫時性 TypedArray，以儲存通過類型檢查的引數。這容許重新使用存在於 push() 方法中的類型檢查程式碼。for..in 迴圈會重複執行 args 陣列，並且針對每個引數呼叫 push()。由於 passArgs 的類型是 TypedArray，因此會執行 push() 的 TypedArray 版本。接著，concat() 方法會呼叫本身的父類別版本，如下列程式碼所示：

```
    AS3 override function concat(...args):Array
    {
        var passArgs:TypedArray = new TypedArray(dataType);
        for (var i:* in args)
        {
            // type check done in push()
            passArgs.push(args[i]);
        }
        return (super.concat.apply(this, passArgs));
    }
```

splice() 方法會採用任意引數清單，但是前兩個引數一定會參考索引編號以及要刪除的元素數目。這就是覆寫的 splice() 方法只會對索引位置 2 或以上的 args 陣列元素進行類型檢查的原因。程式碼中有趣的是，在 for 迴圈中似乎有對 splice() 發出的遞迴呼叫，但是這並非遞迴呼叫，因為 args 的類型屬於 Array 而非 TypedArray，這表示對 args.splice() 的呼叫就是對該方法之父類別版本的呼叫。for..in 迴圈結束後，args 陣列在索引陣列位置 2 或以上只會包含類型正確的值，而 splice() 會呼叫本身的父類別版本，如下列程式碼所示：

```
    AS3 override function splice(...args):*
    {
        if (args.length > 2)
        {
            for (var i:int=2; i< args.length; i++)
            {
                if (!(args[i] is dataType))
                {
                    args.splice(i,1);
                }
            }
        }
        return (super.splice.apply(this, args));
    }
```

將元素新增至陣列開頭的 unshift() 方法也會採用任意引數清單。遭到覆寫之 unshift() 方法所使用的演算法，和 push() 方法所使用的演算法非常類似，如下列範例程式碼所示：

```
    AS3 override function unshift(...args):uint
    {
        for (var i:* in args)
        {
            if (!(args[i] is dataType))
            {
                args.splice(i,1);
            }
        }
        return (super.unshift.apply(this, args));
    }
}
```

陣列範例：PlayList

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

PlayList 範例會在管理歌曲清單的音樂播放清單應用程式中，示範使用陣列的技巧。這些技巧包括：

- 建立索引陣列
- 新增項目至索引陣列
- 使用不同的排序選項，依照不同的屬性排序物件陣列

- 將陣列轉換為以字元分隔的字串

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。您可以在 Samples/PlayList 檔案夾中找到 PlayList 應用程式檔案，此應用程式是由下列檔案組成：

檔案	說明
PlayList.mxml 或 PlayList.fla	主應用程式檔案，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/playlist/PlayList.as	代表歌曲清單的類別，使用 Array 來儲存清單，並管理清單項目的排序。
com/example/programmingas3/playlist/Song.as	代表單一歌曲資訊的值物件。由 PlayList 類別所管理的項目都是 Song 實體。
com/example/programmingas3/playlist/SortProperty.as	虛擬列舉項目可供使用的值代表 Song 類別的屬性，而 Song 物件清單就是依此屬性加以排序。

PlayList 類別概觀

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

PlayList 類別會管理一組 Song 物件。它所具備的公用方法，能夠將歌曲新增至播放清單 (addSong() 方法)，並為清單中的歌曲排序 (sortList() 方法)。另外，此類別還包含唯讀的存取子屬性 songList，可以提供對播放清單中該組實際歌曲的存取。在 PlayList 類別內部，則會使用私有 Array 變數來追蹤歌曲：

```
public class PlayList
{
    private var _songs:Array;
    private var _currentSort:SortProperty = null;
    private var _needToSort:Boolean = false;
    ...
}
```

除了由 PlayList 類別用來追蹤歌曲清單的 _songs Array 變數之外，其它兩個私有變數則會追蹤清單是否需要排序 (_needToSort)，以及在指定時間要以何種屬性為歌曲清單排序 (_currentSort)。

如同所有物件一樣，宣告 Array 實體只能算完成 Array 建立工作的一半而已。存取 Array 實體的屬性或方法之前，必須先加以實體化，而這是由 PlayList 類別的建構函式來進行。

```
    public function PlayList()
    {
        this._songs = new Array();
        // Set the initial sorting.
        this.sortList(SortProperty.TITLE);
    }
```

建構函式的第一行會將 _songs 變數實體化，以供稍後使用。除此之外，並呼叫 sortList() 方法來設定初始的排序方式屬性。

在播放清單中新增歌曲

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當使用者輸入新的歌曲至應用程式時，資料項目表格中的程式碼會呼叫 PlayList 類別的 addSong() 方法。

```
/**
 * Adds a song to the playlist.
 */
public function addSong(song:Song):void
{
    this._songs.push(song);
    this._needToSort = true;
}
```

在 addSong() 中，會呼叫 _songs 陣列的 push() 方法，以加入傳遞至 addSong() 的 Song 物件，做為該陣列中的新元素。不管之前是否曾經套用任何排序，藉由 push() 方法，便可以將此新元素加入至陣列結尾。這表示在呼叫 push() 方法之後，歌曲清單的排序可能不再正確，因此將 _needToSort 變數設定為 true。理論上來說，您可以立即呼叫 sortList() 方法，無須追蹤清單在特定時間是否已排序。然而實際上，只有在即將擷取歌曲之前，才需要排序歌曲清單。應用程式可以藉由延遲排序作業，避免執行不必要的排序動作，並在擷取歌曲清單前，才為新加入清單的歌曲進行排序。

排序歌曲清單

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

由於播放清單所管理的 Song 實體是相當複雜的物件，因此應用程式的使用者可能會希望根據不同的屬性為播放清單排序，例如歌曲名稱或出版年份等。在 PlayList 應用程式中，歌曲清單的排序工作可分為三個部分：識別清單應該以何種屬性排序、指出依照該屬性排序時需要使用哪些排序選項，以及執行實際的排序作業。

用於排序的屬性

Song 物件會追蹤數個屬性，包括歌曲名稱、作者、出版年份、檔案名稱、以及使用者所選取的歌曲類型。其中只有前三個項目才適用於排序。為方便開發人員使用，此範例包含 SortProperty 類別，此類別可做為列舉項目，而它的值則代表可用來進行排序的屬性。

```
public static const TITLE:SortProperty = new SortProperty("title");
public static const ARTIST:SortProperty = new SortProperty("artist");
public static const YEAR:SortProperty = new SortProperty("year");
```

SortProperty 類別包含 TITLE、ARTIST 和 YEAR 常數，每個常數都會儲存一個 String (內含相關聯之 Song 類別屬性的實際名稱，可用於排序)。在程式碼的其它部分，每當指定排序屬性時，就會使用列舉項目的成員來進行。例如，在 PlayList 建構函式中，最初是藉由呼叫 sortList() 方法為清單排序，如下所示：

```
    // Set the initial sorting.
    this.sortList(SortProperty.TITLE);
```

由於用於排序的屬性已指定為 SortProperty.TITLE，因此歌曲會依照其標題加以排序。

依照屬性排序並指定排序選項

歌曲清單實際的排序工作是由 PlayList 類別在 sortList() 方法中執行，如下所示：

```
    /**
     * Sorts the list of songs according to the specified property.
     */
    public function sortList(sortProperty:SortProperty):void
    {
        ...
        var sortOptions:uint;
        switch (sortProperty)
        {
            case SortProperty.TITLE:
                sortOptions = Array.CASEINSENSITIVE;
                break;
            case SortProperty.ARTIST:
                sortOptions = Array.CASEINSENSITIVE;
                break;
            case SortProperty.YEAR:
                sortOptions = Array.NUMERIC;
                break;
        }

        // Perform the actual sorting of the data.
        this._songs.sortOn(sortProperty.propertyName, sortOptions);

        // Save the current sort property.
        this._currentSort = sortProperty;

        // Record that the list is sorted.
        this._needToSort = false;
    }
```

依標題或藝人排序時，可以按照字母順序排序，但是依年份排序時，以數值排序是最合邏輯的方式。switch 陳述式可依據 sortProperty 參數所指定的值，用來定義適當的排序選項，而此排序選項則是儲存於變數 sortOptions 中。這裡又再次使用命名的列舉項目成員，而非硬式編碼值來區別屬性。

排序屬性和排序選項決定好之後，_songs 陣列就會藉由呼叫其 sortOn() 方法實際進行排序，並將這兩個值傳遞為參數。目前的排序屬性會記錄下來，以證明歌曲清單已完成排序。

將陣列元素合併成以字元分隔的字串

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

除了使用陣列來維護 PlayList 類別中的歌曲清單外，此範例中陣列也會用於 Song 類別，以協助管理特定歌曲所屬的類型清單。請考慮以下來自 Song 類別定義的程式碼片段：

```
private var _genres:String;

public function Song(title:String, artist:String, year:uint, filename:String, genres:Array)
{
    ...
    // Genres are passed in as an array
    // but stored as a semicolon-separated string.
    this._genres = genres.join(";");
}
```

在建立新的 Song 實體時，用來指定歌曲所屬類型的 genres 參數會定義為 Array 實體。這樣可以方便將多個類型分組成可傳遞至建構函式的單一變數。然而在內部，Song 類別會將這些類型保留於私有 _genres 變數中，做為以分號區隔的 String 實體。Array 參數會藉由呼叫其 join() 方法，並以常值字串值 ";" 做為指定的分隔符號，以便轉換為以分號區隔的字串。

藉由相同的字符，genres 存取子允許將類型設定或擷取為 Array：

```
public function get genres():Array
{
    // Genres are stored as a semicolon-separated String,
    // so they need to be transformed into an Array to pass them back out.
    return this._genres.split(";");
}
public function set genres(value:Array):void
{
    // Genres are passed in as an array,
    // but stored as a semicolon-separated string.
    this._genres = value.join(";");
}
```

genresset 存取子的表現方式和建構函式相同，它會採用 Array，並呼叫 join() 方法，將 Array 轉換為以分號區隔的 String。get 存取子則執行反向作業：呼叫 _genres 變數的 split() 方法，使用指定的分隔符號 (即之前的常值字串值 ";")，將 String 分割為值陣列。

第 4 章 處理錯誤

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

「處理」錯誤表示您會在應用程式中建立邏輯，用於回應或修正錯誤。編譯應用程式時或編譯的應用程式執行時，所發生的錯誤。應用程式在處理錯誤時，會在遇到錯誤時發生 「某些狀況」以做為回應，而與此相對的就是沒有任何回應 (無論程序造成何種情況，都不會產生錯誤訊息)。若使用得當，錯誤處理可以保護您的應用程式和應用程式的使用者，避免遭遇無法預期的行為。

然而，「錯誤處理」這個類別極為廣泛，包含對各種錯誤的回應，而這些錯誤都是在編譯或應用程式執行過程所擲回。本文將著重在處理執行階段錯誤的方式、可能擲回的各種錯誤類型，以及在 ActionScript 3.0 中的錯誤處理系統優勢。

錯誤處理的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

執行階段錯誤指的是讓 ActionScript 內容無法繼續預期執行的 ActionScript 程式碼錯誤。為了確保您的 ActionScript 程式碼能夠讓使用者順利執行，請在應用程式中撰寫能夠處理錯誤的程式碼，也就是必須能夠修正錯誤、提出替代方案，或是至少讓使用者知道已經發生錯誤。這項程序我們稱為 「錯誤處理」。

「錯誤處理」這個類別極為廣泛，包含對各種錯誤的回應，而這些錯誤都是在編譯或應用程式執行過程所擲回。發生在編譯階段的錯誤通常比較容易分辨出來，請加以修正，才能完成 SWF 檔的建立程序。

執行階段錯誤比較不容易偵測，因為往往需要實際執行錯誤的程式碼才會出現此類錯誤。如果程式某個區段包含數個程式碼分支，像是 if..then..else 陳述式，請代入實際應用中使用者可能會使用的所有可能輸入值來測試每個可能的情況，以便確定程式碼完全沒有錯誤。

執行階段錯誤可分為兩個類別：「程式錯誤」指的是 ActionScript 程式碼中的錯誤，例如在方法參數中指定了錯誤的資料類型；「邏輯錯誤」則是程式中的邏輯錯誤 (資料檢查與數值操作)，例如在計算銀行應用程式中的利率時，使用了錯誤的公式。同樣地，通常您只需小心測試應用程式，就能夠事先偵測並修正這兩種類型的錯誤。

最理想的情況，就是在您將應用程式發佈給一般使用者之前，先從應用程式中找出並移除所有的錯誤。然而，並非所有的錯誤都可以事先察覺或是加以預防。例如，假設您的 ActionScript 應用程式從您掌控範圍外的特定網站載入一些資訊。如果在載入過程中網站無法使用，則仰賴該外部資料的部分應用程式也會跟著出現異常行為。錯誤處理最重要的層面涉及這些不明的狀況以及順利地予以處理。使用者需要繼續使用您的應用程式，或者至少得到一則友善的錯誤訊息，告知為何無效。

ActionScript 以下列兩種方式來表示執行階段錯誤：

- 錯誤類別：許多錯誤都有相關的錯誤類別。一旦發生某種錯誤，Flash 執行階段 (如 Flash Player 或 Adobe AIR) 就會建立與該特定錯誤相關聯的特定錯誤類別實體。您的程式碼可以運用包含在該錯誤物件中的資訊，以便針對錯誤做出適當的回應。
- 錯誤事件：有時當 Flash 執行階段要按照正常程序觸發事件時，就會出現錯誤。在這些情況下，會改為觸發錯誤事件。每個錯誤事件都有相關聯的類別，而 Flash 執行階段會將該類別的實體傳遞至已訂閱錯誤事件的方法中。

若要判斷特定方法是否會觸發錯誤或錯誤事件，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中的方法項目。

重要概念與術語

以下參考清單包含程式設計錯誤處理途徑的重要術語：

非同步 一種程式命令，例如不會立即提供結果的方法呼叫；它將以事件形式提供結果 (或錯誤)。

捕捉 一旦發生例外 (執行階段錯誤) 而您的程式碼察覺例外情況時，就可以說該程式碼 「捕捉」到例外。一旦捕捉到例外，Flash 執行階段就會停止通知其它 ActionScript 程式碼有關此例外情況。

除錯程式版本 包含用來通知使用者執行階段錯誤之程式碼的 Flash 執行階段特殊版本，例如 Flash Player 除錯程式版本或 AIR Debug Launcher (ADL)。在 Flash Player 或 Adobe AIR 標準版中 (就是大多數使用者所使用的版本)，如果錯誤未經過您的 ActionScript 程式碼處理，就會被 Flash Player 忽略。在除錯程式版本中 (隨附於 Adobe Flash CS4 Professional 和 Adobe Flash Builder)，一旦出現未處理的錯誤，就會出現警告訊息。

例外 程式執行時所出現的錯誤，且 Flash 執行階段無法自行解決這個錯誤。

重新擲回 一旦您的程式碼捕捉到例外，Flash 執行階段就不會再通知其它物件有關此例外情況。如果其它物件必須得收到此例外情況，則您的程式碼必須 「重新擲回」例外，以便再次啟動通知程序。

同步 一種程式命令，例如方法呼叫，用來提供立即的結果 (或立即擲回錯誤)，意思就是可以在相同的程式碼區塊中使用該回應。

擲回 通知 Flash 執行階段 (並緊接著通知其它物件與 ActionScript 程式碼) 已經發生錯誤的行為，亦稱為 「擲回」錯誤。

錯誤類型

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您在開發和執行應用程式時，會遇到不同類型的錯誤和錯誤術語。下列清單會介紹主要的錯誤類型和術語：

- 「編譯階段錯誤」是由 ActionScript 編譯器在程式碼編譯期間提出的。當程式碼出現語法問題，使您無法建立應用程式時，就會發生編譯階段錯誤。
- 「執行階段錯誤」是在應用程式編譯完成後，於執行時所發生的錯誤。執行階段錯誤是 SWF 檔在 Flash 執行階段 (如 Adobe Flash Player 或 Adobe AIR) 中播放時所造成的錯誤。在大部分情況下，您可以在發生執行階段錯誤時自行處理，並向使用者報告，同時採取適當步驟，讓應用程式順利執行。如果錯誤相當嚴重，例如無法連線至遠端網站或載入必要資料，則可以使用錯誤處理功能，讓應用程式能夠順利完成。
- 「同步錯誤」是在呼叫函數時所發生的執行階段錯誤。例如，當您嘗試使用特定方法，但是傳遞給方法的引數無效時，Flash 執行階段就會擲回例外。大部分的錯誤都會同步發生 (也就是在執行陳述式時)，而且控制流程會立即傳遞至最適用的 catch 陳述式。

  例如，下列摘錄的程式碼會擲回執行階段錯誤，因為在程式嘗試上傳檔案之前，未先呼叫 browse() 方法：

  ```
  var fileRef:FileReference = new FileReference();
  try
  {
      fileRef.upload(new URLRequest("http://www.yourdomain.com/fileupload.cfm"));
  }
  catch (error:IllegalOperationError)
  {
      trace(error);
      // Error #2037: Functions called in incorrect sequence, or earlier
      // call was unsuccessful.
  }
  ```

  在這種情況下，會同步擲回執行階段錯誤，因為 Flash Player 判斷在嘗試上傳檔案前，未先呼叫 browse() 方法。

  如需有關同步錯誤處理的詳細資訊，請參閱第 48 頁 「處理應用程式中的同步錯誤」。

- 「非同步錯誤」是發生在一般程式流程之外的執行階段錯誤。這些錯誤會產生事件，而事件偵聽程式會捕捉它們。非同步作業是由函數所啟動的作業，但是函數不會等它完成。您可以建立錯誤事件偵聽程式，以等候應用程式或使用者嘗試部分作業。如果作業失敗，您就以事件偵聽程式捕捉錯誤並回應錯誤事件。接著，事件偵聽程式會呼叫事件處理常式函數，以實用的方式回應錯誤事件。例如，事件處理常式可以啟動對話方塊，提示使用者解決錯誤。

請回想之前的檔案上傳同步錯誤範例。如果您在檔案開始上傳前順利呼叫 browse() 方法，Flash Player 便會傳送數個事件。例如，在上傳開始時會傳送 open 事件。檔案上傳作業順利完成時會傳送 complete 事件。由於事件處理是非同步的 (也就是說，不在特定、已知、預先指定的時間發生)，因此請使用 addEventListener() 方法偵聽這些特定事件，如下列程式碼所示：

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.OPEN, openHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();

function selectHandler(event:Event):void
{
    trace("...select...");
    var request:URLRequest = new URLRequest("http://www.yourdomain.com/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    event.target.upload(request);
}
function openHandler(event:Event):void
{
    trace("...open...");
}
function completeHandler(event:Event):void
{
    trace("...complete...");
}
```

如需有關非同步錯誤處理的詳細資訊，請參閱第 53 頁「對錯誤事件和狀態做出回應」。

- 「未捕捉的例外」是在擲回時，沒有對應邏輯 (例如 catch 陳述式) 可以回應的錯誤。當應用程式擲回錯誤，而且在目前或更高層級找不到能夠處理該錯誤的適當 catch 陳述式或事件處理常式時，這種錯誤就視為未捕捉的例外。

  當發生未捕捉的錯誤時，執行階段會傳送 uncaughtError 事件。此事件又稱為「全域錯誤處理常式」。此事件是由 SWF 的 UncaughtErrorEvents 物件所傳送，它可以透過 LoaderInfo.uncaughtErrorEvents 屬性取得。如果未替 uncaughtError 事件註冊偵聽程式，執行階段就會略過未捕捉的錯誤，然後只要錯誤並未停止 SWF，就會嘗試繼續執行。

  除了傳送 uncaughtError 事件之外，Flash 執行階段的除錯程式版本會終止目前的指令碼，以回應未捕捉的錯誤。然後，它們會在 trace 陳述式輸出中顯示未捕捉的錯誤，或者將錯誤訊息寫入記錄檔。若例外物件是 Error 類別的實體或者是它的子類別之一，則堆疊追蹤資訊也會出現在輸出中。如需有關使用 Flash 執行階段除錯程式版本的詳細資訊，請參閱第 48 頁「使用 Flash 執行階段的除錯程式版本」。

  備註：在處理 uncaughtError 事件時，如果從 uncaughtError 處理常式擲出錯誤事件，會多次呼叫事件處理常式。這會造成無限迴圈的例外。建議您避免這種情況。

ActionScript 3.0 中的錯誤處理程序

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

由於許多應用程式即使沒有建立錯誤處理邏輯還是可以執行，因此開發人員往往會延緩為應用程式建立錯誤處理程序。然而，如果沒有建立錯誤處理程序，應用程式可能會在發生非預期狀況時，拖慢執行速度或讓使用者遭遇挫折。ActionScript 2.0 具有 Error 類別，可以讓您在自訂函數中建立邏輯，擲回包含特定訊息的例外。由於錯誤處理是製作易於使用之應用程式的重要關鍵，因此 ActionScript 3.0 便納入更大的架構以捕捉錯誤。

備註：雖然適用於 Adobe Flash Platform 的 ActionScript 3.0 參考記載很多方法擲回的例外，不過它可能未包含每個方法所有可能的例外。方法所擲回的例外可能有語法錯誤，或是其它未在方法說明中明確註明的問題 (即使當說明中確實列出方法會擲回的幾種例外)。

ActionScript 3.0 錯誤處理元素

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 包含多種錯誤處理工具，其中包括：

- Error 類別：ActionScript 3.0 包含範圍廣泛的 Error 類別，以擴充可能產生錯誤物件的狀況範疇。每種 Error 類別都能協助應用程式處理並回應特定的錯誤狀況，不論它們是與系統錯誤 (如 MemoryError 狀況)、編碼錯誤 (如 ArgumentError 狀況)、網路和通訊錯誤 (如 URIError 狀況) 或其它狀況有關。如需有關每個類別的詳細資訊，請參閱第 56 頁 「比較各種 Error 類別」。
- 減少無訊息的失敗情況：在 Flash Player 較早的版本中，您必須明確使用 throw 陳述式，才會產生並回報錯誤。至於 Flash Player 9 以及更新的 Flash 執行階段，原生 ActionScript 方法和屬性會擲回執行階段錯誤。發生這些錯誤時，它們可以讓您更有效的處理這些例外，然後個別處理每個例外。
- 清除在除錯期間所顯示的錯誤訊息：當您使用 Flash 執行階段的除錯程式版本時，有問題的程式碼或狀況會產生完善的錯誤訊息，讓您輕鬆找出特定區段的程式碼失敗的原因。這些訊息會讓錯誤的修正變得更有效率。如需詳細資訊，請參閱第 48 頁 「使用 Flash 執行階段的除錯程式版本」。
- 精確的錯誤可向使用者顯示明確的錯誤訊息。在先前的 Flash Player 版本中，FileReference.upload() 方法會在 upload() 呼叫失敗時，傳回 Boolean 值 false，指出可能發生五種錯誤的其中一種。如果您呼叫 ActionScript 3.0 的 upload() 方法時發生錯誤，有四種特定錯誤中可以協助您向使用者顯示更正確的錯誤訊息。
- 改良後的錯誤處理程序：許多常見狀況都會擲回明確的錯誤。例如，在 ActionScript 2.0 中，在 FileReference 物件遭到填入之前，name 屬性的值是 null (因此，在您能夠使用或顯示 name 屬性之前，請確實設定它的值，不可以是 null)。在 ActionScript 3.0 中，如果您嘗試在 name 屬性有值填入前加以存取，Flash Player 或 AIR 便會擲回 IllegalOperationError，通知您值尚未設定，而且您可以使用 try..catch..finally 區塊來處理此錯誤。如需詳細資訊，請參閱第 48 頁 「使用 try..catch..finally 陳述式」。
- 不會對效能造成重大的影響。與 ActionScript 之前的版本相比，使用 try..catch..finally 區塊來處理錯誤只需要很少資源，或甚至不需要額外的資源。
- ErrorEvent 類別可以讓您針對特定的非同步錯誤事件建立偵聽程式。如需詳細資訊，請參閱第 53 頁 「對錯誤事件和狀態做出回應」。

錯誤處理策略

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

只要不遇到有問題的狀況，即使您未在程式碼中建立錯誤處理邏輯，應用程式還是可以順利執行。然而，如果您不主動處理錯誤，當應用程式遇到問題時，使用者將無從得知應用程式失敗的原因。

應用程式遇到問題時，有幾種不同的處理方式。下列清單摘要說明三種主要的錯誤處理選項：

- 使用 try..catch..finally 陳述式。這些陳述式會在發生同步錯誤時加以捕捉。您可以將陳述式巢狀化於階層架構中，以便在程式碼執行的不同層級捕捉例外。如需詳細資訊，請參閱第 48 頁 「使用 try..catch..finally 陳述式」。
- 建立自己的自訂錯誤物件。您可以使用 Error 類別建立自己的自訂錯誤物件，以追蹤應用程式中，內建錯誤類型未涵蓋的特定作業。接著，便可以針對您的自訂錯誤物件，使用 try..catch..finally 陳述式。如需詳細資訊，請參閱第 52 頁 「建立自訂的錯誤類別」。
- 寫入事件偵聽程式與處理常式以回應錯誤事件。您可以使用這個策略來建立全域錯誤處理常式，這樣在處理類似的事件時，就無須在 try..catch..finally 區塊中複製大量的程式碼。使用這個方法捕捉到非同步錯誤的機率也比較高。如需詳細資訊，請參閱第 53 頁 「對錯誤事件和狀態做出回應」。

使用 Flash 執行階段的除錯程式版本

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Adobe 為開發人員提供特殊版本的 Flash 執行階段以協助進行除錯。在您安裝 Adobe Flash Professional 或 Adobe Flash Builder 時，會得到一份 Flash Player 除錯程式版本。安裝上述其中一種工具，或安裝做為 Adobe AIR SDK 的其中一部分時，您也可以取得 Adobe AIR 應用程式的除錯版本 (稱為 ADL) 公用程式。

Flash Player 和 Adobe AIR 的除錯版本和發行版本在指出錯誤的方式上有顯著的差異。除錯版本會顯示錯誤類型 (例如一般 Error、IOError 或 EOFError)、錯誤編號和人們可讀取的錯誤訊息。發行版本只會顯示錯誤類型和錯誤編號。例如，以下列程式碼為例：

```
try
{
    tf.text = myByteArray.readBoolean();
}
catch (error:EOFError)
{
    tf.text = error.toString();
}
```

假設 readBoolean() 方法在 Flash Player 除錯版本中擲回 EOFError，則 tf 文字欄位中會顯示下列訊息：EOFError：錯誤 #2030：遇到檔案的結尾。

Flash Player 或 Adobe AIR 發行版本中相同的程式碼則會顯示下列文字：「EOFError：錯誤 #2030」。

備註：除錯程式播放程式會廣播名稱為 「allComplete」的事件，因此請避免建立名稱為 「allComplete」的自訂事件。否則，您會在偵錯時發生無法預期的行為。

為了將發行版本中的資源和大小維持在最小，系統不會顯示錯誤訊息字串。您可以在文件 (適用於 Adobe Flash Platform 的 ActionScript 3.0 參考的附錄) 中查詢錯誤編號以找出相關的錯誤訊息。或者，您也可以使用 Flash Player 和 AIR 除錯版本複製錯誤，以查閱完整訊息。

處理應用程式中的同步錯誤

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

最常用的錯誤處理程序是同步錯誤處理邏輯，也就是將陳述式插入程式碼，在應用程式執行時捕捉同步錯誤。這種類型的錯誤處理方式可以讓您的應用程式在功能失敗時，注意到執行階段錯誤並加以復原。捕捉同步錯誤的邏輯包括 try..catch..finally 陳述式，此陳述式基本上會重複嘗試執行作業、捕捉來自 Flash 執行階段的錯誤回應，最後再執行部分其它作業以處理失敗的作業。

使用 try..catch..finally 陳述式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您在處理同步執行階段錯誤時，請使用 try..catch..finally 陳述式來捕捉錯誤。一旦發生執行階段錯誤，Flash Player 或 Adobe AIR 就會擲回例外。也就是說，Flash 執行階段會暫停正常的執行作業，並建立類型為 Error 的特殊物件。接著該 Error 物件會擲向第一個可用的 catch 區塊。

try 陳述式包含可能會造成錯誤的陳述式。使用 catch 陳述式時，請務必搭配 try 陳述式。如果在 try 陳述式區塊的其中一個陳述式內偵測到錯誤，就會執行附加到 try 陳述式的 catch 陳述式。

不管 try 區塊中是否發生錯誤，都會執行 finally 陳述式所包含的陳述式。如果沒有錯誤，finally 區域中的陳述式就會在 try 區塊陳述式完成後執行。如果有錯誤，則會先執行適當的 catch 陳述式，接著再執行 finally 區塊中的陳述式。

下列程式碼將示範使用 try..catch..finally 陳述式的語法：

```
try
{
    // some code that could throw an error
}
catch (err:Error)
{
    // code to react to the error
}
finally
{
    // Code that runs whether an error was thrown. This code can clean
    // up after the error, or take steps to keep the application running.
}
```

每個 catch 陳述式都會識別它所處理的特定例外類型。catch 陳述式可以指定只有 Error 類別之子類別的錯誤類別。每個 catch 陳述式都會依序檢查。只有符合所擲回之錯誤類型的第一個 catch 陳述式會執行。換句話說，假設您先檢查較高層級的 Error 類別，然後再檢查 Error 類別的子類別，只有較高層級的 Error 類別會符合。下列程式碼會說明這個觀點：

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
```

上一段程式碼會顯示下列輸出：

```
<Error> I am an ArgumentError
```

若要正確捕捉 ArgumentError，請確認最明確的錯誤類型排在第一，接著再列出一般性的錯誤類型，如下列程式碼所示：

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
```

ActionScript API 中的幾個方法和屬性如果在執行時遇到錯誤，則會擲回執行階段錯誤。例如，當 Sound 類別中的 close() 方法無法關閉聲音串流時，便會擲回 IOError，如下列程式碼所示：

```
var mySound:Sound = new Sound();
try
{
    mySound.close();
}
catch (error:IOError)
{
    // Error #2029: This URLStream object does not have an open stream.
}
```

當您更熟悉適用於 Adobe Flash Platform 的 ActionScript 3.0 參考之後，會注意到哪些方法會擲回例外，這些都會在每個方法的說明中詳述。

throw 陳述式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您遇到應用程式中的錯誤時，Flash 執行階段就會擲回例外。另外，您本身也可以使用 throw 陳述式明確擲回例外。明確擲出錯誤時，Adobe 建議您擲回 Error 類別或是其子類別的實體。下列程式碼將示範 throw 陳述式擲回 Error 類別 MyErr 的實體，並在最後擲回錯誤後，呼叫函數 myFunction() 加以回應：

```
var MyError:Error = new Error("Encountered an error with the numUsers value", 99);
var numUsers:uint = 0;
try
{
    if (numUsers == 0)
    {
        trace("numUsers equals 0");
    }
}
catch (error:uint)
{
    throw MyError; // Catch unsigned integer errors.
}
catch (error:int)
{
    throw MyError; // Catch integer errors.
}
catch (error:Number)
{
    throw MyError; // Catch number errors.
}
catch (error:*)
{
    throw MyError; // Catch any other error.
}
finally
{
    myFunction(); // Perform any necessary cleanup here.
}
```

請注意，catch 陳述式是依序排列的，因此最明確的資料類型會排在第一。如果 Number 資料類型的 catch 陳述式排在第一，則 uint 資料類型的 catch 陳述式和 int 資料類型的 catch 陳述式都不會執行。

備註：在 Java 程式語言中，每個可以擲回例外的函數都必須宣告這項事實，並將它可以擲回的例外類別，列在附加於函數宣告的 throws 子句中。ActionScript 並不會要求您宣告函數所擲回的例外。

顯示簡單的錯誤訊息

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

新的例外和錯誤事件模型的最大好處之一，就是能夠讓您告訴使用者動作失敗的時間和原因。您的責任是撰寫程式碼以顯示訊息，並提供回應的選項。

下列程式碼將顯示簡單的 try..catch 陳述式，在文字欄位中顯示錯誤：

```
package
{
    import flash.display.Sprite;
    import flash.text.TextField;

    public class SimpleError extends Sprite
    {
        public var employee:XML =
            <EmpCode>
                <costCenter>1234</costCenter>
                <costCenter>1-234</costCenter>
            </EmpCode>;

        public function SimpleError()
        {
            try
            {
                if (employee.costCenter.length() != 1)
                {
                    throw new Error("Error, employee must have exactly one cost center assigned.");
                }
            }
            catch (error:Error)
            {
                var errorMessage:TextField = new TextField();
                errorMessage.autoSize = TextFieldAutoSize.LEFT;
                errorMessage.textColor = 0xFF0000;
                errorMessage.text = error.message;
                addChild(errorMessage);
            }
        }
    }
}
```

藉由使用更大範圍的錯誤類別和內建的編譯器錯誤，ActionScript 3.0 對失敗發生的原因，可提供較先前 ActionScript 版本更多的資訊。此資訊可以讓您以更好的錯誤處理程序，建立更穩定的應用程式。

重新擲回錯誤

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您在建立應用程式時，在幾種情況下，若您無法妥善處理錯誤，則必須重新擲回錯誤。例如，下列程式碼會顯示巢狀 try..catch 區塊，此區塊會在巢狀 catch 區塊無法處理錯誤時，重新擲回自訂的 ApplicationError：

```
try
{
    try
    {
        trace("<< try >>");
        throw new ApplicationError("some error which will be rethrown");
    }
    catch (error:ApplicationError)
    {
        trace("<< catch >> " + error);
        trace("<< throw >>");
        throw error;
    }
    catch (error:Error)
    {
        trace("<< Error >> " + error);
    }
}
catch (error:ApplicationError)
{
    trace("<< catch >> " + error);
}
```

前述程式碼片段的輸出如下所示：

```
<< try >>
<< catch >> ApplicationError: some error which will be rethrown
<< throw >>
<< catch >> ApplicationError: some error which will be rethrown
```

巢狀 try 區塊會擲回自訂的 ApplicationError 錯誤，而此錯誤是由後續的 catch 區域所捕捉。此巢狀 catch 區塊可以嘗試處理錯誤，如果處理失敗，則會將 ApplicationError 物件擲出至含括的 try..catch 區塊。

建立自訂的錯誤類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以擴充其中一個標準 Error 類別，在 ActionScript 中建立您專用的錯誤類別。建立自己的錯誤類別包含了幾個原因：

- 為找出您的應用程式特有的錯誤或錯誤群組。

  例如，除了 Flash 執行階段所捕捉的錯誤外，也可以對自己程式碼所擲回的錯誤採取不同的動作。您可以建立 Error 類別的子類別，追蹤 try..catch 區塊中新的錯誤資料類型。

- 針對應用程式所產生的錯誤，提供獨特的錯誤顯示能力。

  例如，您可以建立新的 toString() 方法，以特定方式來格式化錯誤訊息。您也可以定義 lookupErrorString() 方法，根據使用者的語言偏好設定，採用錯誤代碼並擷取適當的訊息。

專用的 error 類別必須擴充核心 ActionScript Error 類別。以下是專用的 AppError 類別擴充 Error 類別的範例：

```
public class AppError extends Error
{
    public function AppError(message:String, errorID:int)
    {
        super(message, errorID);
    }
}
```

以下是在您的專案中使用 AppError 的範例：

```
try
{
    throw new AppError("Encountered Custom AppError", 29);
}
catch (error:AppError)
{
    trace(error.errorID + ": " + error.message)
}
```

備註：若您要覆寫子類別中的 Error.toString() 方法，請指定一個 ...(rest) 參數。ActionScript 3.0 所依據的 ECMAScript 語言規格會以上述方式定義 Error.toString() 方法，因此基於舊版相容性，ActionScript 3.0 也會以相同方式定義這個方法。因此，當您覆寫 Error.toString() 方法時，請完全符合這些參數。您不需在執行階段傳遞任何參數至 toString() 方法，因為這些參數都會遭到忽略。

對錯誤事件和狀態做出回應

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 在錯誤處理方面最重要的一項改良，就是在應用程式執行時，支援對非同步執行階段錯誤回應的錯誤事件處理。(如需非同步錯誤的定義，請參閱第 45 頁 「錯誤類型」)。

您可以建立事件偵聽程式與事件處理常式，回應錯誤事件。許多類別在傳送錯誤事件時，所採用的方式都和傳送其它事件相同。例如，XMLSocket 類別的實體通常會傳送三種類型的事件：Event.CLOSE、Event.CONNECT 和 DataEvent.DATA。然而，當問題發生時，XMLSocket 類別可以傳送 IOErrorEvent.IOError 或 SecurityErrorEvent.SECURITY_ERROR。如需有關事件偵聽程式和事件處理常式的詳細資訊，請參閱第 108 頁 「處理事件」。

錯誤事件可區分為下列兩種：

- 擴充 ErrorEvent 類別的 Error 事件

  flash.events.ErrorEvent 類別所包含的屬性和方法，可以用來管理和網路及通訊作業有關的執行階段錯誤。AsyncErrorEvent、IOErrorEvent 和 SecurityErrorEvent 類別都會擴充 ErrorEvent 類別。如果您使用的是 Flash 執行階段除錯程式版本，則無須使用偵聽程式函數，在執行階段便會出現對話方塊，通知您播放程式所遇到的任何錯誤事件。

- 以狀態為架構的錯誤事件

  以狀態為架構的錯誤事件和網路與通訊類別的 netStatus 及 status 屬性有關。如果 Flash 執行階段在讀取或寫入資料時遇到問題，netStatus.info.level 或 status.level 屬性的值 (視您所使用的類別物件而定) 便會設定為 "error" 值。如果您要回應這個錯誤，請檢查事件處理常式函數中的 level 屬性是否包含 "error" 這個值。

處理錯誤事件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ErrorEvent 類別及其子類別所包含的錯誤類型，可用來處理 Flash 執行階段在嘗試讀取或寫入資料時所傳送的錯誤。

下列範例會使用 try..catch 陳述式和錯誤事件處理常式，顯示在嘗試讀取本機檔案時所偵測到的任何錯誤。您可以加入較複雜的處理程式碼，以便為使用者提供選項，或是在註解 「您的錯誤處理程式碼在這裡」所指示的地方自動處理錯誤：

```
package
{
    import flash.display.Sprite;
    import flash.errors.IOError;
    import flash.events.IOErrorEvent;
    import flash.events.TextEvent;
    import flash.media.Sound;
    import flash.media.SoundChannel;
    import flash.net.URLRequest;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;

    public class LinkEventExample extends Sprite
    {
        private var myMP3:Sound;
        public function LinkEventExample()
        {
            myMP3 = new Sound();
            var list:TextField = new TextField();
            list.autoSize = TextFieldAutoSize.LEFT;
            list.multiline = true;
            list.htmlText = "<a href=\"event:track1.mp3\">Track 1</a><br>";
            list.htmlText += "<a href=\"event:track2.mp3\">Track 2</a><br>";
            addEventListener(TextEvent.LINK, linkHandler);
            addChild(list);
        }

        private function playMP3(mp3:String):void
        {
            try
            {
                myMP3.load(new URLRequest(mp3));
                myMP3.play();
            }
            catch (err:Error)
            {
                trace(err.message);
                // your error-handling code here
            }
            myMP3.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
        }

        private function linkHandler(linkEvent:TextEvent):void
        {
            playMP3(linkEvent.text);
            // your error-handling code here
        }

        private function errorHandler(errorEvent:IOErrorEvent):void
        {
            trace(errorEvent.text);
            // your error-handling code here
        }
    }
}
```

處理狀態變更事件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash 執行階段會針對支援 level 屬性的類別，動態變更 netStatus.info.level 或 status.level 屬性的值。具有 netStatus.info.level 屬性的類別包括 NetConnection、NetStream 和 SharedObject。具有 status.level 屬性的類別則包括 HTTPStatusEvent、Camera、Microphone 和 LocalConnection。您可以撰寫處理常式函數，以回應 level 值的變更並追蹤通訊錯誤。

下列範例會使用 netStatusHandler() 函數來測試 level 屬性的值。如果 level 屬性指出遇到了錯誤，程式碼便會追蹤訊息「Video stream failed」。

```
package
{
    import flash.display.Sprite;
    import flash.events.NetStatusEvent;
    import flash.events.SecurityErrorEvent;
    import flash.media.Video;
    import flash.net.NetConnection;
    import flash.net.NetStream;

    public class VideoExample extends Sprite
    {
        private var videoUrl:String = "Video.flv";
        private var connection:NetConnection;
        private var stream:NetStream;

        public function VideoExample()
        {
            connection = new NetConnection();
            connection.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
            connection.addEventListener(SecurityErrorEvent.SECURITY_ERROR, securityErrorHandler);
            connection.connect(null);
        }

        private function netStatusHandler(event:NetStatusEvent):void
        {
            if (event.info.level == "error")
            {
                trace("Video stream failed")
            }
            else
            {
                connectStream();
            }
        }

        private function securityErrorHandler(event:SecurityErrorEvent):void
        {
            trace("securityErrorHandler: " + event);
        }

        private function connectStream():void
        {
            var stream:NetStream = new NetStream(connection);
            var video:Video = new Video();
            video.attachNetStream(stream);
            stream.play(videoUrl);
            addChild(video);
        }
    }
}
```

比較各種 Error 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 可提供幾種預先定義的 Error 類別。但您也可以在自己的程式碼中，使用相同的 Error 類別。ActionScript 3.0 中共有兩種主要的 Error 類別：ActionScript 核心 Error 類別及 flash.error 套件 Error 類別。flash.error 套件包含其他類別，用來協助進行 ActionScript 3.0 應用程式的開發與除錯工作。

核心 Error 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

核心錯誤類別包括 Error、ArgumentError、EvalError、RangeError、ReferenceError、SecurityError、SyntaxError、TypeError、URIError 以及 VerifyError 類別。這些類別全部都位於命名空間的最上層。

類別名稱	說明	備註
Error	Error 類別是用於擲出例外，同時也是在 ECMAScript 中定義之其它例外類別 (如 EvalError、RangeError、ReferenceError、SyntaxError、TypeError 和 URIError) 的基底類別。	Error 類別可以做為所有執行階段錯誤的基底類別，同時也是任何自訂錯誤類別的建議基底類別。
ArgumentError	ArgumentError 類別所代表的，是在進行函數呼叫期間，所提供的參數值不符合該函數的定義參數時會出現的錯誤。	引數錯誤的幾個範例如下所示： • 對方法所提供的引數太少或太多。 • 所預期的引數是列舉項目，但實際卻不是。
EvalError	當有任何參數傳遞至 Function 類別的建構函式，或是使用者程式碼呼叫 eval() 函數時，就會擲回 EvalError 例外。	在 ActionScript 3.0 中，已經移除了對 eval() 函數的支援，而且嘗試使用該函數，會導致錯誤。 Flash Player 之前的版本使用的是 eval() 函數，依名稱來存取變數、屬性、物件或影片片段。
RangeError	當數值位於可接受的範例之外時，便會擲回 RangeError 例外。	例如，假設延遲是負值，或者不是有限的值，就會擲回 RangeError。若您嘗試在無效的深度新增顯示物件，也會擲回 RangeError。
ReferenceError	嘗試對密封 (非動態) 的物件參照未定義的屬性時，便會擲回 ReferenceError 例外。當嘗試存取的屬性為 undefined 時，ActionScript 3.0 之前的 ActionScript 編譯器版本並不會擲回錯誤。不過，在這種情況下，ActionScript 3.0 會擲出 ReferenceError 例外。	未定義的變數例外會指向可能的錯誤，以協助您改善軟體品質。然而，如果您不習慣初始化變數的要求，ActionScript 這項新的行為可能會要求您略微變更您的編碼習慣。
SecurityError	發生安全性違規且存取遭到拒絕時，便會擲回 SecurityError 例外。	安全性錯誤的幾個範例如下所示： • 跨越安全執行程序邊界，並進行未授權的屬性存取或方法呼叫。 • 嘗試存取安全執行程序不允許的 URL。 • 已嘗試透過通訊端連線連接至某一通訊埠，但是沒有必要的通訊端原則檔案。 • 已經嘗試存取使用者的攝影機或麥克風，但使用者拒絕裝置的存取。

類別名稱	說明	備註
SyntaxError	ActionScript 程式碼發生剖析錯誤時，會擲回 SyntaxError 例外。	在下列情況下會擲回 SyntaxError： • 當 RegExp 類別剖析無效的規則運算式時，ActionScript 會擲回 SyntaxError 例外。 • 當 XMLDocument 類別剖析無效的 XML 時，ActionScript 會擲回 SyntaxError 例外。
TypeError	當運算元的實際類型不同於預期的類型時，便會擲回 TypeError 例外。	在下列情況下會擲回 TypeError： • 無法將函數或方法的實際參數強制轉型為正式的參數類型。 • 將某個值指定給變數，而且無法將該值強制轉型為變數的類型。 • is 或 instanceof 運算子的右側不是有效的類型。 • 以不合法的方式使用 super 關鍵字。 • 屬性查閱造成一個以上的繫結，因而模糊不清。 • 呼叫不相容物件上的方法。例如，如果將 RegExp 類別中的某個方法「移植」到一般物件，然後再呼叫此方法，便會擲回 TypeError 例外。
URIError	當其中一個全域 URI 處理函數以不相容於其定義的方式使用時，便會擲回 URIError 例外。	在下列情況下會擲回 URIError： 將無效的 URI 指定給期待有效 URI 的 Flash Player API 函數，例如 Socket.connect()。
VerifyError	遭遇格式不正確或損毀的 SWF 檔時，便會擲回 VerifyError 例外。	當某個 SWF 檔載入另一個 SWF 檔時，父 SWF 檔案會捕捉載入之 SWF 檔案所產生的 VerifyError。

flash.error 套件 Error 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

flash.error 套件包含 Error 類別，這些類別都視為是 Flash 執行階段 API 的一部分。與前述的 Error 類別相反，flash.error 套件會與 Flash 執行階段 (如 Flash Player 和 Adobe AIR) 專有的錯誤事件進行通訊。

類別名稱	說明	備註
EOFError	當您嘗試讀取可用資料結尾以外範圍時，會擲出 EOFError 例外。	例如，當您在 IDataInput 介面中呼叫其中一個讀取方法，但資料不足，無法滿足讀取要求時，就會擲出 EOFError。
IllegalOperationError	當方法未實作，或者實作不適用於目前的用法時，會擲回 IllegalOperationError 例外。	不合規定的操作錯誤例外範例包括下列項目： • 基底類別 (例如 DisplayObjectContainer) 提供的功能比 Stage 可支援的更多。例如，若您嘗試針對 Stage 取得或設定遮色片 (使用 stage.mask)，Flash 執行階段便會擲回 IllegalOperationError 以及「Stage 類別未實作這個屬性或方法」的訊息。 • 子類別會繼承它不需要且不要支援的方法。 • 在沒有輔助功能的支援下編譯 Flash Player 時，會呼叫特定輔助功能方法。 • 從 Flash Player 的執行階段版本呼叫僅供編寫的功能。 • 您嘗試為時間軸上的物件設定名稱。
IOError	發生某種 I/O 例外時，就會擲回 IOError 例外。	例如，如果嘗試在尚未連線或已經斷線的通訊端上進行讀取 - 寫入操作，就會收到這個錯誤。
MemoryError	當記憶體配置要求失敗時，會擲回 MemoryError 例外。	根據預設，ActionScript Virtual Machine 2 不會限制 ActionScript 程式可以配置的記憶體數量。桌上型系統很少發生記憶體配置失敗的情形。當系統無法配置作業所需要的記憶體時，就會擲回錯誤。因此桌上型系統很少出現此一例外，除非所要求的配置數量過大，例如要求 30 億位元組 (因為 32 位元的 Microsoft® Windows® 程式只能存取 2 GB 的位址空間)。
ScriptTimeoutError	達到 15 秒的指令碼逾時間隔時，就會擲回 ScriptTimeoutError 例外。您可以藉由捕捉 ScriptTimeoutError 例外，更從容不迫地處理指令碼逾時狀況。如果沒有例外處理常式，未捕捉到的例外處理常式會顯示對話方塊和錯誤訊息。	為避免惡意的開發人員捕捉例外並停留在無限迴圈上，因此只會捕捉特定指令碼過程中所擲出的第一個 ScriptTimeoutError 例外。後續的 ScriptTimeoutError 例外無法由您的指令碼捕捉，而且會立即跳至未捕捉到的例外處理常式。
StackOverflowError	當耗盡指令碼可用的堆疊時，就會擲回 StackOverflowError 例外。	StackOverflowError 例外代表可能發生了無窮遞迴。

處理錯誤範例：CustomErrors 應用程式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

CustomErrors 應用程式會示範在建立應用程式時，處理自訂錯誤的技巧。這些技巧包括：

- 驗證 XML 封包
- 編寫自訂錯誤
- 擲回自訂錯誤
- 於錯誤擲回時通知使用者

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。您可以在 Samples/CustomError 檔案夾中找到 CustomErrors 應用程式檔案，此應用程式是由下列檔案組成：

檔案	說明
CustomErrors.mxml 或 CustomErrors.fla	主應用程式檔案，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/errors/ApplicationError.as	類別，可做為 FatalError 和 WarningError 類別的基底錯誤類別。
com/example/programmingas3/errors/FatalError.as	類別，定義應用程式擲回的 FatalError 錯誤。此類別會擴充自訂的 ApplicationError 類別。
com/example/programmingas3/errors/Validator.as	類別，會定義用來驗證使用者所提供之 employee XML 封包的單一方法。
com/example/programmingas3/errors/WarningError.as	類別，定義應用程式擲回的 WarningError 錯誤。此類別會擴充自訂的 ApplicationError 類別。

CustomErrors 應用程式概觀

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當應用程式載入時，會在 Flex 應用程式中呼叫 initApp() 方法，或在 Flash Professional 應用程式中執行時間軸 (非函數) 程式碼。此程式碼會定義 Validator 類別所要驗證的樣本 XML 封包。此時會執行下列程式碼：

```
employeeXML =
    <employee id="12345">
        <firstName>John</firstName>
        <lastName>Doe</lastName>
        <costCenter>12345</costCenter>
        <costCenter>67890</costCenter>
    </employee>;
}
```

XML 封包稍後將顯示在 「舞台」的 TextArea 組件實體上。這個步驟可以讓您在嘗試重新驗證 XML 封包前加以修改。

當使用者按下「驗證」按鈕時，就會呼叫 validateData() 方法。這個方法會使用 Validator 類別中的 validateEmployeeXML() 方法，驗證 employee XML 封包。下列程式碼會顯示 validateData() 方法：

```
function validateData():void
{
    try
    {
        var tempXML:XML = XML(xmlText.text);
        Validator.validateEmployeeXML(tempXML);
        status.text = "The XML was successfully validated.";
    }
    catch (error:FatalError)
    {
        showFatalError(error);
    }
    catch (error:WarningError)
    {
        showWarningError(error);
    }
    catch (error:Error)
    {
        showGenericError(error);
    }
}
```

首先，會使用 TextArea 組件實體 xmlText 的內容，建立暫時性的 XML 物件。接著呼叫自訂 Validator 類別 (com.example.programmingas3/errors/Validator.as) 中的 validateEmployeeXML() 方法，並將暫時性的 XML 物件當做參數來傳遞。如果 XML 封包有效，status Label 組件實體就會顯示成功的訊息，而應用程式也會結束。如果 validateEmployeeXML() 方法擲回自訂錯誤 (也就是 FatalError、WarningError 或一般 Error)，則會執行適當的 catch 陳述式，並呼叫 showFatalError()、showWarningError() 或 showGenericError() 方法。這些方法都會在名為 statusText 文字區域中顯示適當的訊息，明確通知使用者發生了什麼錯誤。每一種方法也都會以明確的訊息，更新 status Label 組件實體。

如果在嘗試驗證 employee XML 封包時發生嚴重錯誤，則會在 statusText 文字區域中顯示錯誤訊息，而且 xmlText TextArea 組件實體和 validateBtn Button 組件實體都會停用，如下列程式碼所示：

```
function showFatalError(error:FatalError):void
{
    var message:String = error.message + "\n\n";
    var title:String = error.getTitle();
    statusText.text = message + " " + title + "\n\nThis application has ended.";
    this.xmlText.enabled = false;
    this.validateBtn.enabled = false;
    hideButtons();
}
```

如果發生的是警告錯誤而非嚴重錯誤，錯誤訊息同樣會顯示在 statusText TextArea 實體中，但是並不會停用 xmlText TextField 和 Button 組件實體。showWarningError() 方法會在 statusText 文字區域中顯示自訂錯誤訊息。此訊息也會要求使用者決定，看是要繼續驗證 XML 還是要取消指令碼。下列摘錄會顯示 showWarningError() 方法：

```
function showWarningError(error:WarningError):void
{
    var message:String = error.message + "\n\n" + "Do you want to exit this application?";
    showButtons();
    var title:String = error.getTitle();
    statusText.text = message;
}
```

當使用者按一下 「是」或 「否」按鈕時，會呼叫 closeHandler() 方法。下列摘錄會顯示 closeHandler() 方法：

```
function closeHandler(event:CloseEvent):void
{
    switch (event.detail)
    {
        case yesButton:
            showFatalError(new FatalError(9999));
            break;
        case noButton:
            statusText.text = "";
            hideButtons();
            break;
    }
}
```

如果使用者選擇按一下 「是」來中止指令碼，則會擲回 FatalError，並造成應用程式終止。

建立自訂的驗證工具

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

自訂的 Validator 類別包含單一方法 validateEmployeeXML()。validateEmployeeXML() 方法接受單一引數 employee，而這就是您要驗證的 XML 封包。validateEmployeeXML() 方法如下所示：

```
public static function validateEmployeeXML(employee:XML):void
{
    // checks for the integrity of items in the XML
    if (employee.costCenter.length() < 1)
    {
        throw new FatalError(9000);
    }
    if (employee.costCenter.length() > 1)
    {
        throw new WarningError(9001);
    }
    if (employee.ssn.length() != 1)
    {
        throw new FatalError(9002);
    }
}
```

您要驗證的員工必須屬於一個 (唯一一個) 成本中心。如果員工不屬於任何成本中心，此方法便會擲回 FatalError，並反昇至主應用程式檔案中的 validateData() 方法。如果員工屬於一個以上的成本中心，則會擲回 WarningError。XML 驗證工具的最終檢查，是使用者只有一個所定義的社會安全號碼 (XML 封包中的 ssn 節點)。如果 ssn 節點不是剛好一個，則會擲回 FatalError 錯誤。

您可以為 validateEmployeeXML() 方法新增其它檢查。例如，為確定 ssn 節點包含有效數字，或是員工至少定義了一個電話號碼和電子郵件位址，而且這兩個值都是有效的。您也可以修改 XML，讓每位員工都有唯一的 ID，並指定其經理的 ID。

定義 ApplicationError 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ApplicationError 類別可做為 FatalError 和 WarningError 類別的基底類別。ApplicationError 類別會擴充 Error 類別，同時定義自己的自訂方法和屬性，包括定義錯誤 ID、嚴重性，以及包含自訂錯誤代碼和訊息的 XML 物件。此類別也會定義兩個靜態常數，用來定義每種錯誤類型的嚴重性。

ApplicationError 類別的建構函式方法如下所示：

```
public function ApplicationError()
{
    messages =
        <errors>
            <error code="9000">
                <![CDATA[Employee must be assigned to a cost center.]]>
            </error>
            <error code="9001">
                <![CDATA[Employee must be assigned to only one cost center.]]>
            </error>
            <error code="9002">
                <![CDATA[Employee must have one and only one SSN.]]>
            </error>
            <error code="9999">
                <![CDATA[The application has been stopped.]]>
            </error>
        </errors>;
}
```

XML 物件中的每個錯誤節點都包含唯一的數值代碼和錯誤訊息。您可以使用 E4X，藉由錯誤碼輕鬆地查詢錯誤訊息，如下列 getMessageText() 方法所示：

```
public function getMessageText(id:int):String
{
    var message:XMLList = messages.error.(@code == id);
    return message[0].text();
}
```

getMessageText() 方法會採用單一整數引數 id，並且傳回字串。id 引數就是您要查詢的錯誤代碼。例如，若您傳遞的 id 為 9001，則會擷取到一個錯誤，告訴您只能為員工指定一個成本中心。如果有一個以上的錯誤具有相同的錯誤代碼，ActionScript 只會針對第一個找到的結果 (所傳回 XMLList 物件中的 message[0]) 傳回錯誤訊息。

此 getTitle() 類別中的下一個方法不會採用任何參數，並且會傳回字串值，其中內含此一特定錯誤的錯誤 ID。此值用於協助您輕鬆識別在驗證 XML 封包期間，所發生的明確錯誤。下列摘錄會顯示 getTitle() 方法：

```
public function getTitle():String
{
    return "Error #" + id;
}
```

ApplicationError 類別中的最後一個方法是 toString()。這個方法會覆寫 Error 類別所定義的函數，讓您自訂錯誤訊息的顯示方式。方法會傳回字串，用以識別特定的錯誤號碼以及所發生的訊息。

```
public override function toString():String
{
    return "[APPLICATION ERROR #" + id + "] " + message;
}
```

定義 FatalError 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

FatalError 類別會擴充自訂的 ApplicationError 類別，並定義三種方法：FatalError 建構函式、getTitle() 以及 toString()。第一種方法是 FatalError 建構函式，它會採用單一整數引數 errorID，並使用 ApplicationError 類別中定義的靜態常數值來設定錯誤的嚴重性，以及呼叫 ApplicationError 類別的 getMessageText() 方法，取得特定錯誤的錯誤訊息。FatalError 建構函式如下所示：

```
public function FatalError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.FATAL;
    message = getMessageText(errorID);
}
```

FatalError 類別的下一個方法是 getTitle()，它會覆寫之前在 ApplicationError 類別中定義的 getTitle() 方法，並將文字「-- FATAL」附加到標題中，通知使用者發生了嚴重錯誤。getTitle() 方法如下所示：

```
public override function getTitle():String
{
    return "Error #" + id + " -- FATAL";
}
```

此類別最後一個方法是 toString()，會覆寫 ApplicationError 類別中定義的 toString() 方法。toString() 方法是

```
public override function toString():String
{
    return "[FATAL ERROR #" + id + "] " + message;
}
```

定義 WarningError 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

WarningError 類別會擴充 ApplicationError 類別，而且幾乎和 FatalError 類別相同，只有幾個字串的小變動，以及將錯誤嚴重性設定為 ApplicationError.WARNING 而非 ApplicationError.FATAL，如下列程式碼所示：

```
public function WarningError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.WARNING;
    message = super.getMessageText(errorID);
}
```

第 5 章 使用規則運算式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

規則運算式說明一種可用來尋找並操作字串中相符文字的模式。規則運算式與字串很類似，但是包含可用來說明模式和重複的特殊程式碼。例如，下列規則運算式比對以 A 字元開頭，而後面跟著一個或多個連續數字的字串：

```
/A\d+/
```

下列主題將說明建構規則運算式的基本語法。然而，規則運算式相當複雜，而且有細微的差別。您可以在網路或書店中，找到詳細說明規則運算式的資源。請記住，不同的程式設計環境會以不同的方式實作規則運算式；ActionScript 3.0 會依照 ECMAScript 第 3 版語言規格 (ECMA-262) 中的定義實作規則運算式。

更多說明主題

RegExp

規則運算式的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

規則運算式可描述字元模式。規則運算式通常會用來驗證文字值是否符合特定模式 (如驗證使用者輸入的電話號碼是否包含正確的位數)，或是用來取代符合特定模式的部分文字值。

規則運算式可以很簡單。例如，假定您要確認特定字串與 「ABC」相符，或是要以其它文字取代字串中所有的 「ABC」。在這種情況下，您可以透過下列規則運算式來定義由字母 A、B、C 依序組成的模式：

```
/ABC/
```

請注意，規則運算式常值是以正斜線 (/) 字元加以區隔。

規則運算式的模式也可以很複雜，而且有時看起來含意模糊，如下列與有效電子郵件地址相符的運算式：

```
/([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z]{2,6}/
```

大部分情況下，您可以使用規則運算式來搜尋字串中的各種模式，並取代其中一些字元。在這些情況下，您要建立規則運算式物件，並用來做為多種 String 類別方法的其中一個參數。下列 String 類別的方法會將規則運算式當做參數使用：match()、replace()、search() 和 split()。如需有關這些方法的詳細資訊，請參閱第 15 頁 「尋找字串中的樣式並取代子字串」。

RegExp 類別包含下列方法：test() 和 exec()。如需詳細資訊，請參閱第 76 頁 「搭配字串使用規則運算式的方法」。

重要概念與術語

下列參考清單包含了與此功能相關的重要術語：

跳脫字元 表示應將接續的字元視為中繼字元，而非以常值字元看待的字元。在規則運算式語法中，反斜線字元 (\) 就是跳脫字元，所以如果反斜線後面接著另一個字元，則該反斜線就是一個特殊程式碼，而不只是一個字元。

旗標 用來指定關於如何使用規則運算式模式的一些選項的字元，例如是否要區分大小寫字元。

中繼字元 在規則運算式模式中具有特殊意義的字元 (不同於實際代表模式中的字元)。

數量詞 表示模式中某個部分應該重複多少次的一個字元 (或多個字元)。例如，數量詞可用來指定美國郵遞區號應該包含 5 或 9 位數字。

規則運算式 用來定義字元模式，以便確認其它字串是否符合該模式，或是應該取代部分字串的程式陳述式。

規則運算式語法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

本節說明 ActionScript 規則運算式語法的所有元素。如下所示，規則運算式有許多複雜及細微差別的地方。您可以在網路或書店中，找到詳細說明規則運算式的資源。請記住，不同的程式設計環境會以不同的方式實作規則運算式；ActionScript 3.0 會依照 ECMAScript 第 3 版語言規格 (ECMA-262) 中的定義實作規則運算式。

一般來說，您所使用的規則運算式通常比對較複雜的模式，而非簡單的字元字串。例如，下列規則運算式定義由字母 A、B、C 依序組成、後面再跟著數字的模式：

```
/ABC\d/
```

\d 程式碼代表 「任何數字」。反斜線 (\) 字元稱為跳脫字元，與後面的字元結合在一起 (在這個例子中就是 d) 時，在規則運算式中便具有特殊意義。

下列規則運算式定義字元 ABC、後面跟著任何位數數目的模式 (請注意星號)：

```
/ABC\d*/
```

星號字元 (*) 是 「中繼字元」。中繼字元是在規則運算式中具有特殊意義的字元。星號是一種稱為 「數量詞」的特定類型中繼字元，可用來量化字元或字元群組的重複次數。如需詳細資訊，請參閱第 69 頁 「數量詞」。

規則運算式除了其模式之外，還能包含旗標，以指定規則運算式的比對方式。例如，下列規則運算式使用 i 旗標，指定規則運算式會忽略相符字串中的大小寫區別：

```
/ABC\d*/i
```

如需詳細資訊，請參閱第 73 頁 「旗標和屬性」。

您可以透過下列 String 類別的方法來使用規則運算式：match()、replace() 以及 search()。如需有關這些方法的詳細資訊，請參閱第 15 頁 「尋找字串中的樣式並取代子字串」。

建立規則運算式的實體

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

有兩個方法可以建立規則運算式實體。其中一個方法是使用正斜線字元 (/) 來區隔規則運算式，另一個方法則是使用 new 建構函式。例如，下列規則運算式是相等的：

```
var pattern1:RegExp = /bob/i;
var pattern2:RegExp = new RegExp("bob", "i");
```

正斜線區隔規則運算式常值的方式與引號區隔字串常值的方式相同。規則運算式在正斜線內的部分定義了 「模式」。規則運算式在最後的區隔斜線之後，也能包含 「旗標」。這些旗標被視為規則運算式的一部分，但是與其模式分隔開來。

當您使用 new 建構函式時，要使用兩個字串來定義規則運算式。第一個字串定義模式，第二個字串則定義旗標，如下列範例所示：

```
var pattern2:RegExp = new RegExp("bob", "i");
```

當您將正斜線納入以正斜線分隔符號定義的規則運算式 「之中」時，必須在正斜線之前加上反斜線 (\) 跳脫字元。例如，下列規則運算式比對模式 1/2：

```
var pattern:RegExp = /1\/2/;
```

若要將引號納入以 new 建構函式定義的規則運算式 「之中」，您必須在引號前加上反斜線 (\) 跳脫字元 (如同您在定義任何 String 常值一樣)。例如，下列規則運算式比對 eat at "joe's" 的模式：

```
var pattern1:RegExp = new RegExp("eat at \"joe's\"", "");
var pattern2:RegExp = new RegExp('eat at "joe\'s"', "");
```

請不要在以正斜線分隔符號定義的規則運算式中，同時使用反斜線跳脫字元和引號。同樣的，也請不要在以 new 建構函式定義的規則運算式中，同時使用跳脫字元和正斜線。下列規則運算式是相等的，都是定義 1/2 "joe's" 模式：

```
var pattern1:RegExp = /1\/2 "joe's"/;
var pattern2:RegExp = new RegExp("1/2 \"joe's\"", "");
var pattern3:RegExp = new RegExp('1/2 "joe\'s"', '');
```

另外，在以 new 建構函式定義的規則運算式中，若要使用以反斜線 (\) 字元開頭的中繼序列，如 \d (比對任何數字)，請輸入反斜線字元兩次：

```
var pattern:RegExp = new RegExp("\\d+", ""); // matches one or more digits
```

在這種情況下，您必須輸入反斜線字元兩次，因為 RegExp() 建構函式方法的第一個參數是字串，而在字串常值中，您必須輸入反斜線字元兩次，才能將其辨識為單一反斜線字元。

本節以下部分將說明規則運算式模式的定義語法。

如需有關旗標的詳細資訊，請參閱第 73 頁 「旗標和屬性」。

字元、中繼字元與中繼序列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

最簡單的規則運算式是比對字元序列的規則運算式，如下列範例所示：

```
var pattern:RegExp = /hello/;
```

然而，下列稱為中繼字元的字元，在規則運算式中具有特殊意義：

```
^ $ \ . * + ? ( ) [ ] { } |
```

例如，下列規則運算式比對字母 A 後面跟著零或更多的字母 B (星號中繼字元表示此種重複現象)，後面再跟著字母 C：

```
/AB*C/
```

若要在規則運算式的模式中納入中繼字元並排除其特殊意義，必須使用反斜線 (\) 跳脫字元。例如，下列規則運算式依序比對字母 A、字母 B、星號及字母 C：

```
var pattern:RegExp = /AB\*C/;
```

「中繼序列」和中繼字元一樣，在規則運算式中都具有特殊意義。中繼序列是由一個以上的字元所組成。下列章節內容提供使用中繼字元和中繼序列的詳細資訊。

關於中繼字元

下表摘要說明可用於規則運算式的中繼字元：

中繼字元	說明
^ (跳脫字元)	比對字串的開頭。設定好 m (multiline) 旗標，跳脫字元也會比對行開頭 (請參閱第 73 頁 「旗標和屬性」)。請注意，跳脫字元用於字元類別開頭時，所代表的是負值，而非字串開頭。如需詳細資訊，請參閱第 68 頁 「字元類別」。
$(貨幣符號)	比對字串結尾。設定好 m (multiline) 旗標之後，$ 也會比對新行 (\n) 字元之前的位置。如需詳細資訊，請參閱第 73 頁 「旗標和屬性」。
\ (反斜線)	跳脫特殊字元的特殊中繼字元意義。 另外，若要在規則運算式常值中使用正斜線字元，請同時使用反斜線字元，如 /1\/2/ 中所示 (以比對字元 1，後面依序跟著正斜線字元與字元 2)。
. (點)	比對任何單一字元。 只有在設定 s (dotall) 旗標之後，點才會比對新行字元 (\n)。如需詳細資訊，請參閱第 73 頁 「旗標和屬性」。

中繼字元	說明
* (星號)	比對之前重複零次或多次的項目。 如需詳細資訊，請參閱第 69 頁 「數量詞」。
+ (加號)	比對之前重複一或多次的項目。 如需詳細資訊，請參閱第 69 頁 「數量詞」。
? (問號)	比對之前重複零次或一次的項目。 如需詳細資訊，請參閱第 69 頁 「數量詞」。
(和)	定義規則運算式中的群組。將群組用於下列用途： • 限制 | 替代字元的範圍：/(a|b|c)d/ • 定義數量詞的範圍：/(walla.){1,2}/ • 用於後參考中。例如，下列規則運算式中的 \1，比對與模式第一個括號群組相符的任何項目： • /(\w*) 重複：\1/ 如需詳細資訊，請參閱第 71 頁 「群組」。
[和]	定義字元類別，此字元類別定義單一字元可能的相符項目： /[aeiou]/ 比對任一指定字元。 在字元類別中，使用連字符號 (-) 來指定字元範圍： /[A-Z0-9]/ 比對大寫字元 A 到 Z 或 0 到 9。 在字元類別中，插入反斜線以跳脫] 和 - 字元： /[+\-]\d+/ 比對在一或多個數字之前的 + 或 -。 在字元類別中，其它字元 (通常是中繼字元) 會被視為一般字元 (非中繼字元)，因此無須加上反斜線： /[$]/£ 比對 $ 或 £。 如需詳細資訊，請參閱第 68 頁 「字元類別」。
| (管道)	做為替代，以比對左側部分或右側部分： /abc|xyz/ 比對 abc 或 xyz。

關於中繼序列

中繼序列是在規則運算式模式中具有特殊意義的字元序列。下表說明這些中繼序列：

中繼序列	說明
{*n*} {*n*,} 和 {*n*,*n*}	指定前述項目的數值數量詞或數量詞範圍： /A{27}/ 比對字元 A 重複 27 次。 /A{3,}/ 比對字元 A 重複 3 次或 3 次以上。 /A{3,5}/ 比對字元 A 重複 3 到 5 次。 如需詳細資訊，請參閱第 69 頁 「數量詞」。
\b	比對文字字元和非文字字元之間的位置。如果字串中第一個或最後一個字元是文字字元，那麼也比對字串的開頭或結尾。
\B	比對兩個文字字元之間的位置。也比對兩個非文字字元之間的位置。

中繼序列	說明
\d	比對十進位數字。
\D	比對數字以外的任何字元。
\f	比對換頁字元。
\n	比對新行字元。
\r	比對歸位字元。
\s	比對任何空白字元 (空格、定位點、新行或歸位字元)。
\S	比對空白字元以外的任何字元。
\t	比對定位點字元。
\u*nnnn*	比對 Unicode 字元，其字元代碼是以十六進位數字 nnnn 指定。例如，\u263a 是笑臉字元。
\v	比對垂直換頁字元。
\w	比對文字字元 (AZ–、az–、0-9 或 _)。請注意，\w 不比對非英文字元，例如 é、ñ 或 ç。
\W	比對文字字元以外的任何字元。
\\x*nn*	比對帶有指定的 ASCII 值的字元，以十六進位數字 nn 定義。

字元類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您使用字元類別來指定字元清單，以比對規則運算式中的一個位置。您以方括號定義字元類別 ([和])。例如，下列規則運算式定義符合 bag、beg、big、bog 或 bug 的字元類別：

```
/b[aeiou]g/
```

字元類別中的跳脫序列

通常，在規則運算式中具有特殊意義的大部分中繼字元和中繼序列，在字元類別中 「並不具有」相同的意義。例如，在規則運算式中，星號是用於重複，但是在字元類別中則具有不同的功用。下列字元類別會比對星號本身，跟所列出的任何其它字元一起進行比對：

```
/[abc*123]/
```

然而，下表所列的三個字元確實是做為字元類別中的中繼字元進行運作，而具有特殊意義：

中繼字元	在字元類別中的意義
]	定義字元類別的結尾。
-	定義字元範圍 (請參閱下面 「字元類別中的字元範圍」一節)。
\	定義中繼序列並復原中繼字元的特殊意義。

如果要將這些字元識別為常值字元 (沒有特殊的中繼字元意義)，您必須在字元前加上反斜線跳脫字元。例如，下列規則運算式包含與如下四種符號其中一種相符的字元類別 ($、\、] 或 -)：

```
/[$\\\]\-]/
```

除了保留本身之特殊意義的中繼字元外，下列中繼序列會在字元類別內以中繼序列方式運作：

中繼序列	在字元類別中的意義
\n	與新行字元相符。
\r	與歸位字元相符。
\t	與定位點字元相符。
\u*nnnn*	與帶有指定 Unicode 碼位值的字元相符 (以十六進位數字 nnnn 指定)。
\\x*nn*	與帶有指定 ASCII 值的字元相符 (以十六進位數字 nn 指定)。

在字元類別中，其它規則運算式中繼序列和中繼字元都會視為一般字元處理。

字元類別中的字元範圍

使用連字符號來指定字元範圍，如 A-Z、a-z 或 0-9。這些字元必須在字元集中構成有效範圍。例如，下列字元類別符合 a-z 範圍中的任一字元或任何數字：

```
/[a-z0-9]/
```

您也可以使用 \\xnn ASCII 字元碼，以 ASCII 值指定範圍。例如，下列字元類別與擴充 ASCII 字元集的任何字元相符 (如 é 和 ê)：

```
\\x
```

負的字元類別

當您在字元類別開頭使用跳脫 (^) 字元時，它會使該類別變為負數，而未列示的所有字元都會視為相符。下列字元類別與任何字元相符，「除了」小寫字母 (a–z) 或數字以外：

```
/[^a-z0-9]/
```

您必須在字元類別的 「開頭」輸入跳脫 (^) 字元，以表示負值。否則，您只是將跳脫字元新增到字元類別的字元中。例如，下列字元類別與一些符號字元的其中任何一個相符，包括跳脫字元：

```
/[!.,#+*%$&^]/
```

數量詞

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用數量詞來指定字元或序列在模式中的重複情形，如下所示：

數量詞中繼字元	說明
* (星號)	比對之前重複零次或多次的項目。
+ (加號)	比對之前重複一或多次的項目。
? (問號)	比對之前重複零次或一次的項目。
{*n*} {*n*,} 和 {*n*,*n*}	指定前述項目的數值數量詞或數量詞範圍： /A{27}/ 比對字元 A 重複 27 次。 /A{3,}/ 比對字元 A 重複 3 次或 3 次以上。 /A{3,5}/ 比對字元 A 重複 3 到 5 次。

您可以將數量詞套用至單一字元、字元類別或群組上：

- /a+/ 比對字元 a 重複一次或一次以上。

- /\d+/ 比對一個或多個數字。
- /[abc]+/ 比對一個或多個字元的重複，這些字元是 a、b 或 c。
- /(very,)*/ 比對文字 very，後面跟著逗號以及重複零次或多次的空格。

您可以在已經套用數量詞的括號群組之中使用數量詞。例如，下列數量詞會比對 word 和 word-word-word 等字串：

```
/\w+(-\w+)*/
```

根據預設，規則運算式所執行的比對，稱為 「貪婪比對」。規則運算式中的任何子模式 (例如 .*) 都會在移動至規則運算式下個部分之前，儘可能比對字串中的字元。例如，以下列規則運算式和字串為例：

```
var pattern:RegExp = /<p>.*<\/p>/;
str:String = "<p>Paragraph 1</p> <p>Paragraph 2</p>";
```

規則運算式比對整個字串：

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

但是，假設您只要比對一個 ... 群組。您可以藉由下列方式執行這項作業：

```
<p>Paragraph 1</p>
```

在任何數量詞後面加上問號 (?)，將它變更為所謂的 「緩式數量詞」。例如，下列使用緩式 *? 數量詞的規則運算式，會比對 後面接著最少可能的字元數 (緩式)，再接著 ：

```
/<p>.*?<\/p>/
```

請記住下列與數量詞有關的幾個重點：

- 數量詞 {0} 和 {0,0} 不會從相符項目中排除項目。
- 請不要結合使用多個數量詞，如 /abc+*/ 所示。
- 點 (.) 不會合併行，除非設定 s (dotall) 旗標，即使它後面跟著 * 數量詞，也不會合併。例如，以下列程式碼為例：

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*<\/p>/;
trace(str.match(re)); // null;

re = /<p>.*<\/p>/s;
trace(str.match(re));
    // output: <p>Test
    //                  Multiline</p>
```

如需詳細資訊，請參閱第 73 頁 「旗標和屬性」。

替代

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在規則運算式中使用 | (管道) 字元，讓規則運算式引擎考慮比對的替代項目。例如，下列規則運算式會比對下列其中任一文字 cat、dog、pig、rat：

```
var pattern:RegExp = /cat|dog|pig|rat/;
```

您可以使用括號來定義群組，以限制 | 替代字元的範圍。下列規則運算式會比對 cat，後面跟著 nap 或 nip：

```
var pattern:RegExp = /cat(nap|nip)/;
```

如需詳細資訊，請參閱第 71 頁 「群組」。

下列兩個規則運算式，雖然一個使用 | 替代字元，另一個使用字元類別 (以 [和] 定義)，這兩個規則運算式其實是相等的：

```
/1|3|5|7|9/
/[13579]/
```

如需詳細資訊，請參閱第 68 頁 「字元類別」。

群組

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用括號在規則運算式中指定群組，如下所示：

```
/class-(\d*)/
```

群組是模式的子區段。您可以使用群組來執行下列作業：

- 將數量詞套用至一個以上的字元。
- 區隔要套用替代的子模式 (藉由使用 | 字元)。
- 擷取子字串相符項目 (例如，藉著在規則運算式中使用 \1 以比對之前相符的群組，或是在 String 類別的 replace() 方法中以類似方式使用 $1)。

下列章節提供有關群組使用方式的詳細資訊。

搭配數量詞使用群組

如果您不使用群組，數量詞會套用至其前面的字元或字元類別，如下所示：

```
var pattern:RegExp = /ab*/ ;
// matches the character a followed by
// zero or more occurrences of the character b

pattern = /a\d+/;
// matches the character a followed by
// one or more digits

pattern = /a[123]{1,3}/;
// matches the character a followed by
// one to three occurrences of either 1, 2, or 3
```

然而，您可以使用群組，將數量詞套用至一個以上的字元或字元類別：

```
var pattern:RegExp = /(ab)*/;
// matches zero or more occurrences of the character a
// followed by the character b, such as ababab

pattern = /(a\d)+/;
// matches one or more occurrences of the character a followed by
// a digit, such as a1a5a8a3

pattern = /(spam ){1,3}/;
// matches 1 to 3 occurrences of the word spam followed by a space
```

如需有關數量詞的詳細資訊，請參閱第 69 頁 「數量詞」。

使用帶有替代 (|) 字元的群組

您可以使用群組來定義您要套用替代 (|) 字元的字元群組，如下所示：

```
var pattern:RegExp = /cat|dog/;
// matches cat or dog

pattern = /ca(t|d)og/;
// matches catog or cadog
```

使用群組來擷取子字串相符項目

若您在模式中定義標準的括號群組，那麼稍後就可以在規則運算式中加以參考。這就稱為 「後參考」，而這類群組稱為 「擷取群組」。例如，在下列規則運算式中，序列 \1 會比對與擷取括號群組相符的任何子字串：

```
var pattern:RegExp = /(\d+)-by-\1/;
// matches the following: 48-by-48
```

您可以在規則運算式中指定最多 99 個這種後參考，只要輸入 \1、\2、...、\99 即可。

同樣的，在 String 類別的 replace() 方法中，您可以使用 $1–$99，在取代字串中插入所擷取的群組子字串相符項目：

```
var pattern:RegExp = /Hi, (\w+)\./;
var str:String = "Hi, Bob.";
trace(str.replace(pattern, "$1, hello."));
    // output: Bob, hello.
```

另外，若使用擷取群組，RegExp 類別的 exec() 方法和 String 類別的 match() 方法會傳回與擷取群組相符的子字串：

```
var pattern:RegExp = /(\w+)@(\w+).(\w+)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@example.com,bob,example,com
```

使用非擷取群組和 **lookahead** 群組

非擷取群組僅用於群組，而不是 「收集而來」的，而且不符合已編號的後參考，使用 (?: 和) 來定義非擷取群組的情況將如下所示：

```
var pattern = /(?:com|org|net);
```

舉例來說，請注意在擷取群組與非擷取群組中放置 (com|org) 的差異 (exec() 方法將擷取群組列在完全相符的項目之後)：

```
var pattern:RegExp = /(\w+)@(\w+).(com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
// bob@example.com,bob,example,com

//noncapturing:
var pattern:RegExp = /(\w+)@(\w+).(?:com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@example.com,bob,example
```

lookahead 群組是一種特殊類型的非擷取群組，這種群組共有兩種類型：「正 lookahead 群組」和 「負 lookahead 群組」。

使用 (?= 和) 來定義正 lookahead 群組的話，此群組指定群組中的子模式必須在位置上相符。然而，字串中符合正 lookahead 群組的部分，可以比對規則運算式中其餘模式。例如，由於 (?=e) 在下列程式碼中是正 lookahead 群組，它所比對的字元 e 可藉由規則運算式的後續部分進行比對。在此範例中，是擷取群組 \w*)：

```
var pattern:RegExp = /sh(?=e)(\w*)/i;
var str:String = "Shelly sells seashells by the seashore";
trace(pattern.exec(str));
// Shelly,elly
```

使用 (?! 和) 來定義負 lookahead 群組的話，此群組指定群組中的子模式 「不得」在位置上相符。例如：

```
var pattern:RegExp = /sh(?!e)(\w*)/i;
var str:String = "She sells seashells by the seashore";
trace(pattern.exec(str));
// shore,ore
```

使用已命名群組

已命名群組是規則運算式中一種群組類型，具有已命名的識別名稱。請使用 (?P<name> 和) 來定義已命名群組。例如，下列規則運算式包含帶有名為 digits 識別名稱的已命名群組：

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

當您使用 exec() 方法時，會新增相符的已命名群組，以做為 result 陣列的屬性：

```
var myPattern:RegExp = /([a-z]+)(?P<digits>\d+)[a-z]+/;
var str:String = "a123bcd";
var result:Array = myPattern.exec(str);
trace(result.digits); // 123
```

以下是另一個範例，它使用兩個已命名群組，識別名稱是 name 和 dom：

```
var emailPattern:RegExp =
    /(?P<name>(\w|[_.\-])+)@(?P<dom>((\w|-)+))+\.\w{2,4}+/;
var address:String = "bob@example.com";
var result:Array = emailPattern.exec(address);
trace(result.name); // bob
trace(result.dom); // example
```

備註：已命名群組並非 ECMAScript 語言規格的一部分，而是 ActionScript 3.0 的附加功能。

旗標和屬性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下表列出您可以為規則運算式設定的五種旗標。每種旗標都可當做規則運算式物件的屬性加以存取。

旗標	屬性	說明
g	global	比對一個以上的相符項目。
i	ignoreCase	不區分大小寫的相符。套用至 A—Z 以及 a—z 字元，但不套用至擴充字元，如 É 和 é。
m	multiline	若設定這個旗標，$ 和 ^ 就能分別比對行的開頭與結尾。
s	dotall	若設定這個旗標，. (點) 就能比對新行字元 (\n)。
x	extended	允許擴充的規則運算式。您可以在規則運算式中輸入空格，這些空格會被視為模式的一部分而予以忽略。這樣可以讓您以更容易辨識的方式輸入規則運算式程式碼。

請注意，這些是唯讀屬性。在您設定規則運算式變數時，可以設定旗標 (g、i、m、s 以及 x)，如下所示：

```
var re:RegExp = /abc/gimsx;
```

然而，您不能直接設定已命名的屬性。例如，下列程式碼會造成錯誤：

```
var re:RegExp = /abc/;
re.global = true; // This generates an error.
```

根據預設，您必須在規則運算式宣告中加以設定，否則便無法設定旗標，而相對應的屬性也會設定為 false。

另外，規則運算式還有兩項屬性：

- lastIndex 屬性指定字串中的索引位置，以用於下次呼叫規則運算式的 exec() 或 test() 方法。
- source 屬性指定用來定義規則運算式模式部分的字串。

g (global) 旗標

如果 g (global) 旗標 「並未」納入其中，規則運算式最多只比對一個項目。例如，如果 g 旗標未納入規則運算式中，String.match() 方法只會傳回一個相符的子字串：

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // output: she
```

設定 g 旗標之後，Sting.match() 方法會傳回多個相符項目，如下所示：

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/g;
// The same pattern, but this time the g flag IS set.
trace(str.match(pattern)); // output: she,shells,shore
```

i (ignoreCase) 旗標

根據預設，規則運算式的相符項目會區分大小寫。若您設定 i (ignoreCase) 旗標，則會忽略大小寫。例如，規則運算式中的小寫 s 不會比對出字串第一個字元的大寫 S：

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // output: 13 -- Not the first character
```

然而設定 i 旗標之後，規則運算式就會比對出大寫字母 S：

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // output: 0
```

i 旗標只會忽略 A–Z 和 a–z 字元的大小寫，但不包括擴充字元的大小寫，如 É 和 é。

m (multiline) 旗標

若未設定 m (multiline) 旗標，^ 會比對字串開頭，而 $ 會比對字串結尾。若設定了 m 旗標，這些字元會分別比對行的開頭與結尾。以下列包含新行字元的字串為例：

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // Match a word at the beginning of the string.
```

即使在規則運算式中設定了 g (global) 旗標，match() 方法仍然只比對出一個子字串，因為 ^ 只有一個相符項目 (即字串開頭)。輸出是：

```
Test
```

以下是設定了 m 旗標的相同程式碼：

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // Match a word at the beginning of lines.
```

這次輸出會包含兩行開頭的文字：

```
Test,Multiline
```

請注意，只有 \n 字元會指出行的結尾。下列字元則不會：

- 歸位 (\r) 字元
- Unicode 行分隔 (\u2028) 字元
- Unicode 段落分隔 (\u2029) 字元

s (dotall) 旗標

若未設定 s (dotall 或「dot all」) 旗標，規則運算式模式中的點 (.) 不會比對新行字元 (\n)。因此，下列範例中沒有相符項目：

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

但是若設定了 s 旗標，點就會比對新行字元：

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

在此範例中，相符部分是 <p> 標籤中的整個子字串，包括新行字元：

```
<p>Test
Multiline</p>
```

x (extended) 旗標

規則運算式可能不易閱讀，尤其當包含大量中繼符號和中繼序列時。例如：

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

若您在規則運算式中使用 x (extended) 旗標，您在模式中輸入的任何空格都會被忽略。例如，下列規則運算式和上個範例完全相同：

```
/     <p    (>  |  (\s*  [^>]*  >))    .*?    <\/p>  /gix
```

若您已設定 x 旗標，並且確實要比對空格字元，請在空格前加上反斜線。例如，下列兩個運算式是相等的：

```
/foo bar/
/foo \ bar/x
```

lastIndex 屬性

lastIndex 屬性指定字串中的索引位置，下次搜尋就是要從這個位置開始。對於將 g 旗標設定為 true 的規則運算式而言，這個屬性會影響在此規則運算式上呼叫的 exec() 和 test() 方法。例如，以下列程式碼為例：

```
var pattern:RegExp = /p\w*/gi;
var str:String = "Pedro Piper picked a peck of pickled peppers.";
trace(pattern.lastIndex);
var result:Object = pattern.exec(str);
while (result != null)
{
    trace(pattern.lastIndex);
    result = pattern.exec(str);
}
```

根據預設，lastIndex 屬性設定為 0 (從字串開頭開始搜尋)。在每次比對相符之後，就會設定到相符項目後的索引位置。因此，上述程式碼的輸出如下所示：

```
0
5
11
18
25
36
44
```

如果 global 旗標設定為 false，exec() 和 test() 方法就不會使用或設定 lastIndex 屬性。

String 類別的 match()、replace() 和 search() 方法都會從字串開頭展開所有搜尋，不論用於呼叫方法的規則運算式中的 lastIndex 屬性設定為何 (但是 match() 方法會將 lastIndex 設定為 0)。

您可以設定 lastIndex 屬性，以調整規則運算式之相符字串中的開始位置。

source 屬性

source 屬性指定用來定義規則運算式模式部分的字串。例如：

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

搭配字串使用規則運算式的方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

RegExp 類別包含兩種方法：exec() 和 test()。

除了 RegExp 類別的 exec() 和 test() 方法以外，String 類別也包含下列方法，讓您在字串中比對規則運算式：match()、replace()、search() 和 splice()。

test() 方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

RegExp 類別的 test() 方法只會檢查所提供的字串是否包含規則運算式的相符項目，如下列範例所示：

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // output: true
```

exec() 方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

RegExp 類別的 exec() 方法會檢查對規則運算式相符項目所提供的字串，並傳回帶有下列項目的陣列：

- 符合的子字串
- 子字串會比對規則運算式中的任何括號群組

陣列也包含 index 屬性，指出子字串相符項目開頭的索引位置。

例如，以下列程式碼為例：

```
var pattern:RegExp = /\d{3}\-\d{3}-\d{4}/; //U.S phone number
var str:String = "phone: 415-555-1212";
var result:Array = pattern.exec(str);
trace(result.index, " - ", result);
// 7-415-555-1212
```

當規則運算式設定了 g (global) 旗標時，請使用 exec() 方法多次，以比對多個子字串：

```
var pattern:RegExp = /\w*sh\w*/gi;
var str:String = "She sells seashells by the seashore";
var result:Array = pattern.exec(str);

while (result != null)
{
    trace(result.index, "\t", pattern.lastIndex, "\t", result);
    result = pattern.exec(str);
}
//output:
// 0  3  She
// 10  19  seashells
// 27  35  seashore
```

使用 RegExp 參數的 String 方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列 String 類別的方法會將規則運算式當做參數使用：match()、replace()、search() 和 split()。如需有關這些方法的詳細資訊，請參閱第 15 頁「尋找字串中的樣式並取代子字串」。

一般表示式範例：Wiki 剖析器

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

這個簡單的 Wiki 文字轉換範例說明規則運算式的一些使用方式：

- 將符合來源 Wiki 模式的文字行，轉換為適當的 HTML 輸出字串。
- 使用規則運算式，將 URL 模式轉換為 HTML <a> 超連結標籤。
- 使用規則運算式，將美元字串 (如 "$9.95") 轉換為歐元字串 (如 "8.24 € "")。

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。您可以在 Samples/WikiEditor 資料夾中找到 WikiEditor 應用程式檔案。此應用程式是由下列檔案組成：

檔案	說明
WikiEditor.mxml 或 WikiEditor.fla	主應用程式檔案，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/regExpExamples/WikiParser.as	類別，其中所包含的方法可使用規則運算式，將 Wiki 輸入文字模式轉換為對等的 HTML 輸出。
com/example/programmingas3/regExpExamples/URLParser.as	類別，其中所包含的方法可使用規則運算式，將 URL 字串轉換為 HTML <a> 超連結標籤。
com/example/programmingas3/regExpExamples/CurrencyConverter.as	類別，其中所包含的方法可使用規則運算式，將美元字串轉換為歐元字串。

定義 WikiParser 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

WikiParser 類別所包含的方法，可以將 Wiki 輸入文字轉換為對等的 HTML 輸出。這並不是非常完備的 Wiki 轉換應用程式，但是卻示範出規則運算式在模式比對與字串轉換上，一些不錯的使用方式。

建構函數以及 setWikiData() 方法只會初始化 Wiki 輸入文字的樣本字串，如下所示：

```
public function WikiParser()
{
    wikiData = setWikiData();
}
```

當使用者按下樣本應用程式的「測試」按鈕時，應用程式就會叫用 WikiParser 物件的 parseWikiString() 方法。這個方法會呼叫其它幾個方法，而這幾個方法則進而組合所產生的 HTML 字串。

```
public function parseWikiString(wikiString:String):String
{
    var result:String = parseBold(wikiString);
    result = parseItalic(result);
    result = linesToParagraphs(result);
    result = parseBullets(result);
    return result;
}
```

所呼叫的每個方法 —parseBold()、parseItalic()、linesToParagraphs() 和 parseBullets()— 都使用字串的 replace() 方法取代由規則運算式所定義的比對模式，以便將輸入 Wiki 文字轉換成 HTML 格式的文字。

轉換粗體和斜體模式

parseBold() 方法會尋找 Wiki 粗體文字模式 (例如 '''foo''')，並將其轉換為 HTML 的對等模式 (如 foo)，如下所示：

```
private function parseBold(input:String):String
{
    var pattern:RegExp = /'''(.*?)'''/g;
    return input.replace(pattern, "<b>$1</b>");
}
```

請注意，規則運算式的 (.?*) 部分會比對兩個定義 ''' 模式之間，任何數目的字元 (*)。此 ? 數量詞讓比對變成非貪婪，因此對於 '''aaa''' bbb '''ccc''' 這樣的字串而言，第一個相符的字串會是 '''aaa'''，而非整個字串 (以 ''' 模式開始和結束)。

規則運算式中的括號定義擷取群組，而 replace() 方法則在取代字串中使用 $1 程式碼，以參考這個群組。規則運算式中的 g (global) 旗標會確保 replace() 方法取代字串中所有相符項目 (不只是第一個)。

parseItalic() 方法運作方式與 parseBold() 方法類似，只不過它會查找兩個 (而不是三個) 所有格符號 ('') 做為斜體文字分隔符號：

```
private function parseItalic(input:String):String
{
    var pattern:RegExp = /''(.*?)''/g;
    return input.replace(pattern, "<i>$1</i>");
}
```

轉換項目符號模式

如下列範例所示，parseBullet() 方法會尋找 Wiki 項目符號行模式 (如 * foo)，並將其轉換為 HTML 對等模式 (如 foo)：

```
private function parseBullets(input:String):String
{
    var pattern:RegExp = /^\*(.*)/gm;
    return input.replace(pattern, "<li>$1</li>");
}
```

位於規則運算式開頭的 ^ 符號與行開頭相符。規則運算式中的 m (multiline) 旗標會造成規則運算式比對 ^ 行開頭，而不僅是字串開頭。

* 模式會比對星號字元 (用反斜線來表示常值星號，而非 * 數量詞)。

規則運算式中的括號定義擷取群組，而 replace() 方法則在取代字串中使用 $1 程式碼，以參考這個群組。規則運算式中的 g (global) 旗標會確保 replace() 方法取代字串中所有相符項目 (不只是第一個)。

轉換段落 **Wiki** 模式

linesToParagraphs() 方法會將輸入 Wiki 字串中的每一行，轉換為 HTML <p> 段落標籤。方法中的這幾行會去除輸入 Wiki 字串的空行：

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

規則運算式的 ^ 和 $ 符號會比對行的開頭和結尾。規則運算式中的 m (multiline) 旗標會造成規則運算式比對 ^ 行開頭，而不僅是字串開頭。

replace() 方法會以空字串 ("") 取代所有相符的子字串 (空行)。規則運算式中的 g (global) 旗標會確保 replace() 方法取代字串中所有相符項目 (不只是第一個)。

將 URL 轉換為 HTML <a> 標籤

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當使用者按下樣本應用程式中的 「測試」按鈕時，如果使用者選取 urlToATag 核取方塊，應用程式會呼叫 URLParser.urlToATag() 靜態方法，將 URL 字串從輸入 Wiki 字串轉換為 HTML <a> 標籤。

```
var protocol:String = "((?:http|ftp)://)";
var urlPart:String = "([a-z0-9_-]+\.[a-z0-9_-]+)";
var optionalUrlPart:String = "(\.[a-z0-9_-]*)";
var urlPattern:RegExp = new RegExp(protocol + urlPart + optionalUrlPart, "ig");
var result:String = input.replace(urlPattern, "<a href='$1$2$3'><u>$1$2$3</u></a>");
```

使用 RegExp() 建構函數，將規則運算式 (urlPattern) 從幾個組成部分組合起來。這些組成部分就是定義規則運算式模式部分的各個字串。

規則運算式模式由 protocol 字串定義的第一部分，定義了 URL 通訊協定：也就是 http:// 或 ftp://。括號定義了非擷取群組，以 ? 符號來表示。也就是說，括號只是用來定義 | 替代模式的群組，群組並不會符合 replace() 方法替代字串中的後參考代碼 ($1、$2、$3)。

規則運算式的其它每個組成部分都各自使用擷取群組 (由模式中的括號表示)，這些群組再用於 replace() 方法之取代字串中的後參考代碼 ($1、$2、$3)。

模式中由 urlPart 字串定義的部分 「至少」比對下列其中一個字元：a-z、0-9、_ 或 -。+ 數量詞指出至少有一個字元相符。此 \. 表示需要點 (.) 字元。而剩餘部分則比對至少下列一個字元的其它字串：a-z、0-9、_ 或 -。

模式中由 optionalUrlPart 字串定義的部分「不符合或符合多個」下列項目：點 (.) 字元，後面跟著任何數目的英數字元 (包括 _ 和 -)。* 數量詞指出有零個或多個字元相符。

對 replace() 方法的呼叫採用規則運算式，並以後參考組合取代 HTML 字串。

urlToATag() 方法接著呼叫 emailToATag() 方法，後者使用類似的技巧，以 HTML <a> 超連結字串取代電子郵件模式。在此樣本檔案中，用來比對 HTTP、FTP 和電子郵件 URL 的規則運算式相當簡單，這是為了方便說明的緣故。若要更正確比對這類 URL，必須要有更加複雜的規則運算式。

將美元字串轉換為歐元字串

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當使用者按下樣本應用程式的 「測試」按鈕時，如果使用者選取 dollarToEuro 核取方塊，應用程式會呼叫 CurrencyConverter.usdToEuro() 靜態方法，將美元字串 (如 "$9.95") 轉換為歐元字串 (如 "8.24 € ")，如下所示：

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

第一行定義了比對美元字串的簡單模式。請注意，$ 字元前面有反斜線 (\) 跳脫字元。

replace() 方法將規則運算式當做模式比對項使用，它呼叫 usdStrToEuroStr() 函數來決定取代字串 (以歐元為單位的值)。

當您使用函數名稱做為 replace() 方法的第二參數時，下列項目會被當成參數傳遞至所呼叫的函數：

- 字串的符合部分。

- 任可擷取的括號群組相符項目。以這種方式傳遞的引數數目，會隨著所擷取括號群組相符項目的數量而改變。您可以檢查函數程式碼內的 arguments.length - 3 ，判斷所擷取括號群組相符項目的數量。
- 相符項目在字串中開始的索引位置。
- 完整的字串。

usdStrToEuroStr() 方法會將美元字串模式轉換為歐元字串，如下所示：

```
private function usdToEuro(...args):String
{
    var usd:String = args[1];
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.828017;
    var euro:Number = Number(usd) * exchangeRate;
    trace(usd, Number(usd), euro);
    const euroSymbol:String = String.fromCharCode(8364); // €
    return euro.toFixed(2) + " " + euroSymbol;
}
```

請注意，args[1] 代表與 usdPrice 規則運算式相符的已擷取括號群組。這是美元字串的數值部分，也就是只有金額數字，而沒有 $ 符號。此方法可用於匯率轉換，並會傳回所得到的字串 (後面會有結尾的 € 符號，而非開頭有 $ 符號)。

第 6 章 使用 XML

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 包含一組以 ECMAScript for XML (E4X) 規格 (ECMA-357 第 2 版) 為基礎的類別。這些類別都包含了強大且易於使用的功能，可供您用來處理 XML 資料。使用 E4X，您將能搭配 XML 資料來開發程式碼，並且效率將比使用先前的程式設計技巧還要高。而且，您所撰寫的程式碼也會更易於閱讀，這也是它的附加益處。

更多說明主題

XML 類別

ECMA-357 規格

XML 的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

XML 是一種用來表示結構化資訊的標準方式，不但可方便電腦處理，同時能讓開發人員與讀者輕鬆撰寫與瞭解。XML 是可擴展標記語言 (eXtensible Markup Language) 的縮寫。XML 標準可以在 www.w3.org/XML/ 上找到。

XML 針對資料提供了一項標準且方便的分類方式，讓使用者輕鬆讀取、存取與操作。XML 採用樹狀結構及與 HTML 類似的標籤結構。下列是一則簡易的 XML 資料範例：

```
<song>
    <title>What you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

XML 資料也可能因為巢狀式標籤結構，以及一些特質及其它結構組件而變得更複雜。下列是一則較為複雜的 XML 資料範例：

```
<album>
    <title>Questions, unanswered</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <tracks>
        <song tracknumber="1" length="4:05">
            <title>What do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:31</lastplayed>
        </song>
        <song tracknumber="2" length="3:45">
            <title>Who do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:35</lastplayed>
        </song>
        <song tracknumber="3" length="5:14">
            <title>When do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:39</lastplayed>
        </song>
        <song tracknumber="4" length="4:19">
            <title>Do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:44</lastplayed>
        </song>
    </tracks>
</album>
```

請注意，這個 XML 文件不但包含了其它完整的 XML 結構 (例如 song 標籤及其子系)，同時示範了其它 XML 結構，例如一些特質 (song 標籤中的 tracknumber 與 length)，以及一些包含其它標籤，而非包含資料 (例如 tracks 標籤) 的標籤。

XML 快速入門

如果您對 XML 僅有少許經驗，或是第一次使用，下列常見的 XML 資料簡報可供您參考。XML 資料是以純文字形式所撰寫，所使用的特殊語法可將資訊組織為結構化格式。一組 XML 資料通常稱為「XML 文件」。在 XML 格式裡，資料將透過階層結構分為各種「元素」(可能是單一資料項目或是其它元素的容器) 來加以組織。每一個 XML 文件都具有一個做為上層或主要項目的單一元素；在此根元素之內，可能會存在單一資訊，但是也很可能存在其它元素，且這些元素下層又會包含其它元素，以此類推。例如，這個 XML 文件就包含了音樂專輯的相關資訊：

```
<song tracknumber="1" length="4:05">
    <title>What do you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <mood>Happy</mood>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

每個元素都可由一組「標籤」來辨別，也就是將元素名稱包含在角括弧 (小於和大於符號) 之內。開頭的標籤代表元素的開始，內含元素名稱：

```
<title>
```

結束的標籤代表元素的結尾，內含正斜線加上元素名稱：

```
</title>
```

如果元素未包含任何內容，則可以寫成空白元素 (亦稱為自我結束元素)。在 XML 中，下列元素：

```
<lastplayed/>
```

與下列元素相似：

```
<lastplayed></lastplayed>
```

除了元素的內容會包含在開頭與結尾標籤之內以外，元素同時能夠包含其它稱為「特質」的值 (在元素的開頭標籤中定義)。例如，下列 XML 元素會定義名為 length 的單一特質，其值為 "4:19"：

```
<song length="4:19"></song>
```

每個 XML 元素都會包含內容，內容有可能是單一數值、一或多個 XML 元素，或是無 (表示為空白元素)。

深入瞭解 **XML**

若要深入瞭解使用 XML 的方式，可以參考其它一些書籍及資源，包括下列網站：

- W3Schools XML 教學課程：http://w3schools.com/xml/
- XMLpitstop 教學課程、討論清單等資料：http://xmlpitstop.com/

用來使用 **XML** 的 **ActionScript** 類別

ActionScript 3.0 包含好幾種可用來使用 XML 結構資訊的類別。下列是兩大主要類別：

- XML：代表單一 XML 元素，可能是包含多個子系的 XML 文件，或是文件內具有單一值的元素。
- XMLList：代表一組 XML 元素。當存在多個 XML 元素「旁系」(位於 XML 文件階層架構中的相同階層，且由相同的父系所包含) 時，就會用到 XMLList 物件。對實體來說，要處理此類 XML 元素集合時 (假定包含在 XML 文件中)，使用 XMLList 實體將是最簡便的方式：

```
<artist type="composer">Fred Wilson</artist>
<artist type="conductor">James Schmidt</artist>
<artist type="soloist">Susan Harriet Thurndon</artist>
```

若要納入 XML 命名空間做進階應用，則 ActionScript 亦能包含 Namespace 與 QName 類別。如需詳細資訊，請參閱第 94 頁「使用 XML 命名空間」。

除了使用 XML 時內建的類別之外，ActionScript 3.0 同時包含了好幾個運算子，特別針對您在存取與操作 XML 資料時所需而提供。利用這些類別與運算子來使用 XML 的方式我們稱為 ECMAScript for XML (E4X)，是由 ECMA-357 第 2 版規格所定義。

重要概念與術語

下列參考清單包含程式設計 XML 處理常式時會碰到的重要術語：

元素 XML 文件中的單一項目，也就是開頭標籤與結尾標籤之間所包含的內容 (包括標籤本身)。XML 元素可以包含文字資料或其它元素，也可以空白。

空白元素 不包含任何子元素的 XML 元素。空白元素通常以自我結束標籤 (例如 <element/>) 來代表。

文件 單一 XML 結構。XML 文件可以包含任何數量的元素 (或是僅包含單一空白元素)；然而，XML 文件必須具有單一上層元素，其中包含文件中的所有其它元素。

節點 XML 元素的別名。

特質 與元素相關聯的具名值，並以 attributename="value" 格式寫在元素的開頭標籤中，而不是寫成套疊在元素內的個別子元素。

用來處理 XML 的 E4X

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ECMAScript for XML 規格定義了一組類別與功能，可供您用來使用 XML 資料。這些類別與功能統稱為 E4X。ActionScript 3.0 包含下列 E4X 類別：XML、XMLList、QName 和 Namespace。

E4X 類別的各項方法、屬性與運算子都是針對下列目標而設計：

- 簡易性 — 在使用 XML 資料時，E4X 可讓撰寫及閱讀程式碼變得更加容易。
- 一致性 — E4X 的方式與思考是要達到內部的一致性，也同時要能與 ActionScript 的其它部分一致。
- 熟悉度 — 您可以使用眾所周知的運算子來處理 XML 資料，例如點 (.) 運算子。

備註：在 ActionScript 2.0 中是不同的 XML 類別；而在 ActionScript 3.0 中，這個類別已經重新命名為 XMLDocument，以免名稱與屬於 E4X 的 ActionScript 3.0 XML 類別發生衝突。在 ActionScript 3.0 中，從先前版本遺留下來的 XMLDocument、XMLNode、XMLParser 以及 XMLTag 類別都會包含在 flash.xml 套件內，主要用來提供與先前版本的相容性支援。新的 E4X 類別都是核心類別，因此您不需匯入套件就能加以使用。如需傳統 ActionScript 2.0 XML 類別的詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中的 flash.xml 套件。

以下是使用 E4X 來處理資料的範例：

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

您的應用程式經常會從外部來源 (例如網路服務或是 RSS Feed) 載入 XML 資料。不過，為了清楚起見，此處提供的程式碼範例會將 XML 資料指定為常值。

如下列程式碼所示，E4X 包含某些直覺式的運算子，例如點 (.) 和特質識別名稱 (@) 運算子，以便在 XML 中存取屬性與特質：

```
trace(myXML.item[0].menuName); // Output: burger
trace(myXML.item.(@id==2).menuName); // Output: fries
trace(myXML.item.(menuName=="burger").price); // Output: 3.95
```

您可以使用 appendChild() 方法，將新的子節點指定給 XML，如下列程式碼片段所示：

```
var newItem:XML =
    <item id="3">
        <menuName>medium cola</menuName>
        <price>1.25</price>
    </item>

myXML.appendChild(newItem);
```

@ 和 . 運算子不僅可以用來讀取資料，也可以用來指定資料，如下所示：

```
myXML.item[0].menuName="regular burger";
myXML.item[1].menuName="small fries";
myXML.item[2].menuName="medium cola";

myXML.item.(menuName=="regular burger").@quantity = "2";
myXML.item.(menuName=="small fries").@quantity = "2";
myXML.item.(menuName=="medium cola").@quantity = "2";
```

您可以使用 for 迴圈重複執行 XML 的節點，如下所示：

```
var total:Number = 0;
for each (var property:XML in myXML.item)
{
    var q:int = Number(property.@quantity);
    var p:Number = Number(property.price);
    var itemTotal:Number = q * p;
    total += itemTotal;
    trace(q + " " + property.menuName + " $" + itemTotal.toFixed(2))
}
trace("Total: $", total.toFixed(2));
```

XML 物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

XML 物件可以代表 XML 元素、特質、註解、處理指示，或文字元素。

XML 物件可以分類為具有 「簡單內容」或 「複雜內容」。具有子節點的 XML 物件是分類為具有複雜內容；而當 XML 物件為下列任一項目時，這個物件即具有簡單內容：特質、註解、處理指示或文字節點。

例如，下列 XML 物件包含複雜內容，其中包括了註解和處理指示：

```
XML.ignoreComments = false;
XML.ignoreProcessingInstructions = false;
var x1:XML =
    <order>
        <!--This is a comment. -->
        <?PROC_INSTR sample ?>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

如下列範例所示，您現在可以使用 comments() 和 processingInstructions() 方法來建立新的 XML 物件、註解與處理指示：

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

XML 屬性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

XML 類別具有五個靜態屬性：

- 剖析 XML 物件時，ignoreComments 和 ignoreProcessingInstructions 屬性會判斷是否忽略註解和處理指示。
- ignoreWhitespace 屬性會判斷是否要在元素標籤與內嵌運算式 (只依據空白字元來分隔) 中忽略空白字元。
- prettyIndent 和 prettyPrinting 屬性是用來格式化 XML 類別的 toString() 和 toXMLString() 方法所傳回的文字。

如需這些屬性的詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。

XML 方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列方法可讓您使用 XML 物件的階層架構：

- appendChild()
- child()
- childIndex()
- children()
- descendants()
- elements()
- insertChildAfter()
- insertChildBefore()
- parent()
- prependChild()

下列方法可讓您使用 XML 物件特質：

- attribute()
- attributes()

下列方法可讓您使用 XML 物件屬性：

- hasOwnProperty()
- propertyIsEnumerable()
- replace()
- setChildren()

下列方法可讓您處理限定名稱與命名空間：

- addNamespace()
- inScopeNamespaces()
- localName()
- name()
- namespace()
- namespaceDeclarations()
- removeNamespace()
- setLocalName()
- setName()
- setNamespace()

下列方法可讓您使用並判斷特定類型的 XML 內容：

- comments()
- hasComplexContent()
- hasSimpleContent()

- nodeKind()
- processingInstructions()
- text()

下列方法可供用來轉換成字串以及格式化 XML 物件：

- defaultSettings()
- setSettings()
- settings()
- normalize()
- toString()
- toXMLString()

此外，還有幾個額外的方法：

- contains()
- copy()
- valueOf()
- length()

如需這些方法的詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。

XMLList 物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

XMLList 實體代表 XML 物件的任意集合。它可以包含完整的 XML 文件、XML 片段或 XML 查詢的結果。

下列方法可讓您使用 XMLList 物件的階層架構：

- child()
- children()
- descendants()
- elements()
- parent()

下列方法可讓您使用 XMLList 物件特質：

- attribute()
- attributes()

下列方法可讓您使用 XMLList 物件屬性：

- hasOwnProperty()
- propertyIsEnumerable()

下列方法可讓您使用並判斷特定類型的 XML 內容：

- comments()
- hasComplexContent()

- hasSimpleContent()
- processingInstructions()
- text()

下列方法可供用來轉換成字串以及格式化 XMLList 物件：

- normalize()
- toString()
- toXMLString()

此外，還有幾個額外的方法：

- contains()
- copy()
- length()
- valueOf()

如需這些方法的詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。

對於確實包含一個 XML 元素的 XMLList 物件，由於包含一個 XML 元素的 XMLList 會視為與 XML 物件相同，因此您可以使用此 XML 類別的所有屬性與方法。例如，在下列程式碼中，由於 doc.div 是只包含一個元素的 XMLList 物件，因此您可以使用此 XML 類別的 appendChild() 方法：

```
var doc:XML =
        <body>
            <div>
                <p>Hello</p>
            </div>
        </body>;
doc.div.appendChild(<p>World</p>);
```

如需 XML 屬性與方法的清單，請參閱：第 85 頁 「XML 物件」。

初始化 XML 變數

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以將 XML 常值指派給 XML 物件，如下所示：

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

如下列程式碼片段所示，您也可以使用 new 建構函式，從包含 XML 資料的字串建立 XML 實體：

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                            + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

如果此字串中的 XML 資料並非制式格式 (例如，結束標籤遺失)，您便會看到執行階段錯誤。

您也能以傳址方式，從其它變數將資料傳入 XML 物件中，如下列範例所示：

```
var tagname:String = "item";
var attributename:String = "id";
var attributevalue:String = "5";
var content:String = "Chicken";
var x:XML = <{tagname} {attributename}={attributevalue}>{content}</{tagname}>;
trace(x.toXMLString())
    // Output: <item id="5">Chicken</item>
```

若要透過 URL 載入 XML 資料，請使用 URLLoader 類別，如下列範例所示：

```
import flash.events.Event;
import flash.net.URLLoader;
import flash.net.URLRequest;

var externalXML:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("xmlFile.xml");
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete);

function onComplete(event:Event):void
{
    var loader:URLLoader = event.target as URLLoader;
    if (loader != null)
    {
        externalXML = new XML(loader.data);
        trace(externalXML.toXMLString());
    }
    else
    {
        trace("loader is not a URLLoader!");
    }
}
```

若要透過通訊端連線讀取 XML 資料，請使用 XMLSocket 類別。如需詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考 中的 XMLSocket 類別。

組合及轉換 XML 物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 prependChild() 或 appendChild() 方法，將某個屬性加入 XML 物件之屬性清單的開頭或結尾處，如下列範例所示：

```
var x1:XML = <p>Line 1</p>
var x2:XML = <p>Line 2</p>
var x:XML = <body></body>
x = x.appendChild(x1);
x = x.appendChild(x2);
x = x.prependChild(<p>Line 0</p>);
    // x == <body><p>Line 0</p><p>Line 1</p><p>Line 2</p></body>
```

您可以使用 insertChildBefore() 或 insertChildAfter() 方法，在所指定屬性之前或之後加入某個屬性，如下所示：

```
var x:XML =
    <body>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </body>
var newNode:XML = <p>Paragraph 1.5</p>
x = x.insertChildAfter(x.p[0], newNode)
x = x.insertChildBefore(x.p[2], <p>Paragraph 1.75</p>)
```

如下列範例所示，您也可以使用大括號運算子 ({ 和 })，在建構 XML 物件時，以傳址方式從其它變數傳遞資料：

```
var ids:Array = [121, 122, 123];
var names:Array = [["Murphy","Pat"], ["Thibaut","Jean"], ["Smith","Vijay"]]
var x:XML = new XML("<employeeList></employeeList>");

for (var i:int = 0; i < 3; i++)
{
    var newnode:XML = new XML();
    newnode =
        <employee id={ids[i]}>
            <last>{names[i][0]}</last>
            <first>{names[i][1]}</first>
        </employee>;

    x = x.appendChild(newnode)
}
```

您可以使用 = 運算子，將屬性與特質指派給 XML 物件，如下所示：

```
var x:XML =
    <employee>
        <lastname>Smith</lastname>
    </employee>
x.firstname = "Jean";
x.@id = "239";
```

這會將 XML 物件 x 設定為如下：

```
<employee id="239">
    <lastname>Smith</lastname>
    <firstname>Jean</firstname>
</employee>
```

您可以使用 + 與 += 運算子，連接多個 XMLList 物件：

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

這會將 XMLList 物件 xList 設定為如下：

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

移動 XML 結構

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

XML 的其中一項強大功能，就是可透過文字字元的線性字串，提供複雜的巢狀結構資料。當您將資料載入 XML 物件中，ActionScript 就會剖析這項資料，並將其階層架構載入至記憶體中 (如果此 XML 資料並非制式格式，則會傳送執行階段錯誤)。

XML 和 XMLList 物件的運算子與方法可讓移動 XML 資料架構的工作變得輕鬆容易。

您可以使用點 (.) 運算子及後代存取子 (..) 運算子來存取 XML 物件的子屬性。以下列 XML 物件為例：

```
var myXML:XML =
    <order>
        <book ISBN="0942407296">
            <title>Baking Extravagant Pastries with Kumquats</title>
            <author>
                <lastName>Contino</lastName>
                <firstName>Chuck</firstName>
            </author>
            <pageCount>238</pageCount>
        </book>
        <book ISBN="0865436401">
            <title>Emu Care and Breeding</title>
            <editor>
                <lastName>Case</lastName>
                <firstName>Justin</firstName>
            </editor>
            <pageCount>115</pageCount>
        </book>
    </order>
```

myXML.book 物件是其中包含 myXML 物件之子屬性 (名為 book) 的 XMLList 物件。有兩個 XML 物件，而且這兩個物件符合 myXML 物件的兩個 book 屬性。

myXML..lastName 物件是包含所有名為 lastName 之後代屬性的 XMLList 物件。有兩個 XML 物件，而且這兩個物件符合 myXML 物件的兩個 lastName。

myXML.book.editor.lastName 物件是包含 myXML 物件之所有曾孫系 (名為 lastName) 的 XMLList 物件 (其中子系名為 book 且孫系名為 editor)：在此範例中，則是只包含一個 XML 物件 (值為 「Case」的 lastName 屬性) 的 XMLList 物件。

存取父節點及子節點

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

parent() 方法會傳回 XML 物件的父輩。

您可以使用子清單的序數索引值，存取特定的子物件。例如，假設某個 XML 物件 myXML 包含兩個名為 book 的子屬性。每一個名為 book 的子屬性都具有與本身相關聯的索引編號：

```
myXML.book[0]
myXML.book[1]
```

若要存取特定的孫系，您可以為子系與孫系名稱指定索引編號：

```
myXML.book[0].title[0]
```

然而，如果只有一個名為 title 的子系 x.book[0]，您就可以省略此索引參考，如下所示：

```
myXML.book[0].title
```

同樣地，如果 x 物件只有一個 book 子系，而且這個子物件也只有一個 title 物件，您就可以省略這兩者的索引參考，如下所示：

```
myXML.book.title
```

您可以使用 child() 方法，瀏覽至根據變數或運算式所命名的子系，如下列範例所示：

```
var myXML:XML =
        <order>
            <book>
                <title>Dictionary</title>
            </book>
        </order>;

var childName:String = "book";

trace(myXML.child(childName).title) // output: Dictionary
```

存取特質

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 @ 符號 (特質識別名稱運算子)，存取 XML 或 XMLList 物件中的特質，如下列程式碼中所示：

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@id); // 6401
```

您可以使用 * 萬用字元符號搭配 @ 符號，存取 XML 或 XMLList 物件中的所有特質，如下列程式碼中所示：

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

您可以使用 attribute() 或 attributes() 方法，存取 XML 或 XMLList 物件的特定特質或所有特質，如下列程式碼中所示：

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.attribute("id")); // 6401
trace(employee.attribute("*").toXMLString());
// 6401
// 233
trace(employee.attributes().toXMLString());
// 6401
// 233
```

請注意，您也可以使用下列語法來存取特質，如下列範例所示：

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

這些每一個都是等同於 employee.@id 的語法。然而，employee.@id 則是慣用的語法。

藉由特質或元素值進行篩選

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用括號運算子 (和)，篩選具有特定名稱或特質值的元素，以下列 XML 物件為例：

```
var x:XML =
    <employeeList>
        <employee id="347">
            <lastName>Zmed</lastName>
            <firstName>Sue</firstName>
            <position>Data analyst</position>
        </employee>
        <employee id="348">
            <lastName>McGee</lastName>
            <firstName>Chuck</firstName>
            <position>Jr. data analyst</position>
        </employee>
    </employeeList>
```

以下都是有效的運算式：

- x.employee.(lastName == "McGee") — 這是第二個 employee 節點。
- x.employee.(lastName == "McGee").firstName — 這是第二個 employee 節點的 firstName 屬性。
- x.employee.(lastName == "McGee").@id — 這是第二個 employee 節點之 id 特質的值。
- x.employee.(@id == 347) — 第一個 employee 節點。
- x.employee.(@id == 347).lastName — 這是第一個 employee 節點的 lastName 屬性。
- x.employee.(@id > 300) — 這是具有兩個 employee 屬性的 XMLList。
- x.employee.(position.toString().search("analyst") > -1) — 這是具有兩個 position 屬性的 XMLList。

如果您嘗試篩選不存在的特質或元素，就會擲出例外。例如，下列程式碼的最後一行會產生錯誤，因為第二個 p 元素中並沒有 id 特質：

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(@id == '123'));
```

同樣地，下列程式碼的最後一行也會產生錯誤，因為第二個 p 元素中並沒有 b 屬性：

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(b == 'Bob'));
```

若要避免這些錯誤發生，您可以使用 attribute() 和 elements() 方法，辨識具有相符特質或元素的特質，如下列程式碼中所示：

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(attribute('id') == '123'));
trace(doc.p.(elements('b') == 'Bob'));
```

您也可以使用 hasOwnProperty() 方法，如下列程式碼中所示：

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(hasOwnProperty('@id') && @id == '123'));
trace(doc.p.(hasOwnProperty('b') && b == 'Bob'));
```

使用 for..in 和 for each..in 陳述式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 包含 for..in 陳述式和 for each..in 陳述式，可用來重複執行 XMLList 物件。例如，以下列 XML 物件 myXML 以及 XMLList 物件 myXML.item 為例：XMLList 物件 myXML.item 由 XML 物件的兩個 item 節點組成。

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2' quantity='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>;
```

for..in 陳述式可讓您重複執行 XMLList 中的一組屬性名稱：

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
    total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

for each..in 陳述式可讓您重複執行 XMLList 中的屬性：

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
    total2 += prop.@quantity * prop.price;
}
```

使用 XML 命名空間

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

XML 物件 (或文件) 中的命名空間可以辨識該物件所包含的資料類型。例如，將 XML 資料發送或遞送至使用 SOAP 訊息通訊協定的網路服務時，您必須在 XML 的開始標籤中宣告命名空間：

```
var message:XML =
    <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <soap:Body xmlns:w="http://www.test.com/weather/">
            <w:getWeatherResponse>
                <w:tempurature >78</w:tempurature>
            </w:getWeatherResponse>
        </soap:Body>
    </soap:Envelope>;
```

這個命名空間具有 soap 前置詞以及定義此命名空間的 URI http://schemas.xmlsoap.org/soap/envelope/。

ActionScript 3.0 包含 Namespace 類別，可用來使用 XML 命名空間。對於先前範例中的 XML 物件，您可以依照下列方式使用 Namespace 類別：

```
var soapNS:Namespace = message.namespace("soap");
trace(soapNS); // Output: http://schemas.xmlsoap.org/soap/envelope/

var wNS:Namespace = new Namespace("w", "http://www.test.com/weather/");
message.addNamespace(wNS);
var encodingStyle:XMLList = message.@soapNS::encodingStyle;
var body:XMLList = message.soapNS::Body;

message.soapNS::Body.wNS::GetWeatherResponse.wNS::tempurature = "78";
```

XML 類別包含下列方法，可用來使用命名空間：addNamespace()、inScopeNamespaces()、localName()、name()、namespace()、namespaceDeclarations()、removeNamespace()、setLocalName()、setName() 和 setNamespace()。

default xml namespace 指令可讓您為 XML 物件指定預設的命名空間。例如下列程式碼中，x1 和 x2 都有相同的預設命名空間：

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");
default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

XML 類型轉換

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以將 XML 和 XMLList 物件轉換為 String 值。同樣地，您也可以將字串轉換為 XML 和 XMLList 物件。此外，也請記住，所有的 XML 特質值、名稱和文字值都是字串。下列幾節將會討論所有這些 XML 類型轉換的格式。

將 XML 和 XMLList 物件轉換為字串

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

XML 和 XMLList 類別都包含 toString() 和 toXMLString() 方法。toXMLString() 方法會傳回字串，其中包含 XML 物件的所有標籤、特質、命名空間宣告以及內容。對於具有複雜內容 (子元素) 的 XML 物件而言，toString() 方法的功能與 toXMLString() 方法完全相同；而對於具有簡單內容 (只包含一個文字元素) 的 XML 物件而言，toString() 方法則只會傳回該元素的文字內容，如下列範例所示：

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName.toXMLString());
    // <menuName>burger</menuName>
trace(myXML.item[0].menuName.toString());
    // burger
```

如果您使用 trace() 方法而不指定 toString() 或 toXMLString()，則預設會使用 toString() 方法來轉換資料，如下列程式碼所示：

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName);
    // burger
```

使用 trace() 方法為程式碼進行除錯時，您將會經常用到 toXMLString() 方法，因此 trace() 方法會輸出更完整的資料。

將字串轉換為 XML 物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 new XML() 建構函式，從字串建立 XML 物件，如下所示：

```
var x:XML = new XML("<a>test</a>");
```

如果您嘗試將代表無效 XML 或 XML (非制式格式) 的字串轉換為 XML，就會擲回執行階段錯誤，如下所示：

```
var x:XML = new XML("<a>test"); // throws an error
```

從字串轉換為特質值、名稱和文字值

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

因為所有 XML 特質值、名稱和文字值的資料類型都是 String，所以您必須將它們轉換為其它的資料類型。例如，下列程式碼會使用 Number() 函數，將文字值轉換為數字：

```
var myXML:XML =
                        <order>
                            <item>
                                <price>3.95</price>
                            </item>
                            <item>
                                <price>1.00</price>
                            </item>
                        </order>;

var total:XML = <total>0</total>;
myXML.appendChild(total);

for each (var item:XML in myXML.item)
{
    myXML.total.children()[0] = Number(myXML.total.children()[0])
                                                + Number(item.price.children()[0]);
}
trace(myXML.total); // 4.95;
```

如果上述程式碼沒有使用 Number() 函數，則會將 + 運算子解譯為字串連接運算子，而且最後一行的 trace() 方法會輸出：

```
01.003.95
```

讀取外部 XML 文件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 URLLoader 類別，從 URL 載入 XML 資料。若要在您的應用程式中使用下列程式碼，請將範例中的 XML_URL 值取代為有效的 URL：

```
import flash.events.Event;
import flash.net.URLLoader;

var myXML:XML = new XML();
var XML_URL:String = "http://www.example.com/Sample3.xml";
var myXMLURL:URLRequest = new URLRequest(XML_URL);
var myLoader:URLLoader = new URLLoader(myXMLURL);
myLoader.addEventListener(Event.COMPLETE, xmlLoaded);

function xmlLoaded(event:Event):void
{
    myXML = XML(myLoader.data);
    trace("Data loaded.");
}
```

您也可以使用 XMLSocket 類別，設定 XML 與伺服器之間非同步的通訊端連線。如需詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。

ActionScript 中的 XML 範例：從網際網路載入 RSS 資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

RSSViewer 樣本應用程式示範在 ActionScript 中使用 XML 的多項功能，其中包括：

- 使用 XML 方法，移動格式為 RSS Feed 的 XML 資料。
- 使用 XML 方法，組合格式為 HTML 的 XML 資料，以便在文字欄位中使用。

RSS 格式已經廣泛用於透過 XML 提供新資訊。簡單的 RSS 資料檔看起來如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
    <title>Alaska - Weather</title>
    <link>http://www.nws.noaa.gov/alerts/ak.html</link>
    <description>Alaska - Watches, Warnings and Advisories</description>

    <item>
        <title>
            Short Term Forecast - Taiya Inlet, Klondike Highway (Alaska)
        </title>
        <link>
            http://www.nws.noaa.gov/alerts/ak.html#A18.AJKNK.1900
        </link>
        <description>
            Short Term Forecast Issued At: 2005-04-11T19:00:00
            Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
            Homepage: http://pajk.arh.noaa.gov
        </description>
</item>
    <item>
        <title>
            Short Term Forecast - Haines Borough (Alaska)
        </title>
            <link>
            http://www.nws.noaa.gov/alerts/ak.html#AKZ019.AJKNOWAJK.190000
        </link>
        <description>
            Short Term Forecast Issued At: 2005-04-11T19:00:00
            Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
            Homepage: http://pajk.arh.noaa.gov
        </description>
    </item>
</channel>
</rss>
```

SimpleRSS 應用程式會從網際網路讀取 RSS 資料，並剖析此資料以找出其中的標題、連結及說明，然後再傳回這些資料。SimpleRSSUI 類別可提供 UI 並呼叫 SimpleRSS 類別，如此便會執行所有的 XML 處理作業。

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。您可以在 Samples/RSSViewer 檔案夾中找到 RSSViewer 應用程式檔案，此應用程式是由下列檔案組成：

檔案	說明
RSSViewer.mxml 或 RSSViewer.fla	主應用程式檔案，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/rssViewer/RSSParser.as	類別，其中包含使用 E4X 來移動 RSS (XML) 資料，並產生對應之 HTML 呈現格式的方法。
RSSData/ak.rss	樣本 RSS 檔案。這個應用程式是設定用來從網路 (Adobe 所提供的 Flex RSS Feed) 讀取 RSS 資料。然而，您可以輕鬆地將應用程式變更為從這份文件讀取 RSS 資料，它所使用的結構描述與 Flex RSS Feed 的結構描述不同。

讀取並剖析 XML 資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

RSSParser 類別包含 xmlLoaded() 方法，可將輸入 RSS 資料 (儲存在 rssXML 變數中) 轉換為包含 HTML 格式之輸出內容的字串 rssOutput。

如果來源 RSS 資料包含預設的命名空間，則在開始執行此方法之前，程式碼會設定此預設的 XML 命名空間：

```
if (rssXML.namespace("") != undefined)
{
    default xml namespace = rssXML.namespace("");
}
```

然後，下一行會重複執行來源 XML 資料的內容，以檢查每一個名為 item 的後代屬性：

```
for each (var item:XML in rssXML..item)
{
    var itemTitle:String = item.title.toString();
    var itemDescription:String = item.description.toString();
    var itemLink:String = item.link.toString();
    outXML += buildItemHTML(itemTitle,
                            itemDescription,
                            itemLink);
}
```

前三行只會設定字串變數，以代表 XML 資料之 item 屬性的 title、description 及 link 屬性。第四行便會呼叫 buildItemHTML() 方法，使用這三個新的字串變數做為參數，以取得格式為 XMLList 物件的 HTML 資料。

組合 XMLList 資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

HTML 資料 (XMLList 物件) 的格式如下：

```
<b>itemTitle</b>
<p>
    itemDescription
    <br />
    <a href="link">
        <font color="#008000">More...</font>
    </a>
</p>
```

方法的第一行會清除預設的 xml 命名空間：

```
default xml namespace = new Namespace();
```

default xml namespace 指令的範圍為函數區塊層級，這也就表示，此宣告的範圍是 buildItemHTML() 方法。

下列程式碼會根據傳遞給此函數的字串引數來組合 XMLList：

```
var body:XMLList = new XMLList();
body += new XML("<b>" + itemTitle + "</b>");
var p:XML = new XML("<p>" + itemDescription + "</p>");

var link:XML = <a></a>;
link.@href = itemLink; // <link href="itemLinkString"></link>
link.font.@color = "#008000";
        // <font color="#008000"></font></a>
        // 0x008000 = green
link.font = "More...";

p.appendChild(<br/>);
p.appendChild(link);
body += p;
```

這個 XMLList 物件代表適用於 ActionScript HTML 文字欄位的字串資料。

xmlLoaded() 方法會使用 buildItemHTML() 方法的傳回值，並將此值轉換為字串：

```
XML.prettyPrinting = false;
rssOutput = outXML.toXMLString();
```

摘取 RSS Feed 的標題及傳送自訂的事件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

xmlLoaded() 方法會根據來源 RSS XML 資料中的資訊，設定 rssTitle 字串變數：

```
rssTitle = rssXML.channel.title.toString();
```

最後，xmlLoaded() 方法會產生事件，通知該應用程式此資料已經剖析，並且可供使用：

```
            dataWritten = new Event("dataWritten", true);
```

第 7 章 使用原生 JSON 功能

ActionScript 3.0 提供原生 API，用於編碼和解碼使用 JavaScript Object Notation (JSON) 格式的 ActionScript 物件。JSON 類別與支援成員函數遵循 ECMA-262 第 5 版規格，但有一些不同之處。

社群成員 Todd Anderson 提供一份原生 JSON API 與協力廠商 as3corelib JSON 類別的比較。請參閱在 Flash Player 11 使用原生 JSON。

更多說明主題

JSON

JSON API 概觀

ActionScript JSON API 包含 JSON 類別與一些原生類別的 toJSON() 成員函數。應用程式若有任何類別需要自訂 JSON 編碼，ActionScript 架構可提供覆寫預設編碼的途徑。

JSON 類別會在內部針對所有未提供 toJSON() 成員的 ActionScript 類別處理匯入和匯出。對於這類情形，JSON 會移動所遇到每個物件的公用屬性。如果物件內包含其它物件，JSON 會遞迴至巢狀物件並執行相同的移動方式。若有任何物件提供 toJSON() 方法，JSON 會使用該自訂方法，而非內部的演算法。

JSON 介面包含一種編碼方法 stringify() 以及一種解碼方法 parse()。這些方法各自提供一個參數，讓您將自己的邏輯插入 JSON 編碼或解碼工作流程。若為 stringify()，此參數為 replacer；若為 parse()，則該參數為 reviver。這些參數會接受使用下列簽名且具有兩個引數的函數定義：

```
function(k, v):*
```

toJSON() 方法

toJSON() 方法的簽名為

```
public function toJSON(k:String):*
```

JSON.stringify() 會在物件移動期間，針對遇到的每個公用屬性呼叫 toJSON() (若存在的話)。屬性包含索引鍵值配對。當 stringify() 呼叫 toJSON() 時，該方法會傳遞其目前檢查屬性的索引鍵 k。一般的 toJSON() 實作會評估每個屬性名稱，並傳回屬性值所需的編碼。

toJSON() 方法可以傳回任何類型的值 (以 * 表示)— 而不只是字串。此變數傳回類型可讓 toJSON() 在適當時傳回物件。例如，自訂類別的屬性若包含來自其他協力廠商元件庫的物件，您可以在 toJSON() 遇到該屬性時傳回那個物件。JSON 便會遞迴至該協力廠商物件。編碼處理流程的運作方式如下：

- 如果 toJSON() 傳回的物件未評估為字串，stringify() 便會遞迴至該物件。
- 如果 toJSON() 傳回字串，stringify() 便會將該值包覆在另一個字串中，傳回包覆的字串，然後移至下一個值。

許多情況都會傳回物件，以便傳回您的應用程式所建立的 JSON 字串。傳回物件會運用內建的 JSON 編碼演算法，同時允許 JSON 遞迴至巢狀物件。

在 Object 類別或大部分其它原生類別中，均未定義 toJSON() 方法。這告訴 JSON 要在物件的公用屬性上執行標準移動方式。如果有意願的話，您也可以使用 toJSON() 來公開物件的私有屬性。

有一些原生類別連 ActionScript 元件庫也無法有效解決所有的使用情況。對於這些類別，ActionScript 提供一種簡單的實作方式，用戶端能夠重複實作以符合其需求。提供簡單 toJSON() 成員的類別包括：

- ByteArray

- Date
- Dictionary
- XML

您可以利用 ByteArray 類別的子類別來覆寫 toJSON() 方法，或是重新定義它的原型。宣告為 final 的 Date 和 XML 類別會要求您使用類別原型來重新定義 toJSON()。Dictionary 類別宣告為 dynamic，讓您有額外的自由來覆寫 toJSON()。

定義自訂 JSON 行為

若要針對原生類別實作自己的 JSON 編碼和解碼，您可以從幾個選項選擇：

- 定義或覆寫非 final 原生類別之自訂子類別中的 toJSON()
- 定義或重新定義類別原型上的 toJSON()
- 定義 dynamic 類別上的 toJSON 屬性
- 使用 JSON.stringify() replacer 和 JSON.parser() reviver 參數

更多說明主題

ECMA-262 第 5 版

在內建類別的原型定義 toJSON()

ActionScript 中的原生 JSON 實作方式反映出 ECMA-262 第 5 版中定義的 ECMAScript JSON 機制。因為 ECMAScript 不支援類別，所以 ActionScript 根據原型傳送方式來定義 JSON 行為。原型是 ActionScript 3.0 類別的前身，允許模擬繼承以及新增和重新定義成員。

ActionScript 可讓您在任何類別的原型定義或重新定義 toJSON()。這項特權甚至適用於標示 final 的類別。當您在類別原型定義 toJSON() 時，在應用程式範圍內該類別的所有實體都會採用您的定義。例如，以下是您可在 MovieClip 原型定義 toJSON() 方法的方式：

```
MovieClip.prototype.toJSON = function(k):* {
    trace("prototype.toJSON() called.");
    return "toJSON";
}
```

然後，當應用程式在任何 MovieClip 實體呼叫 stringify() 時，stringify() 便會傳回 toJSON() 方法的輸出：

```
var mc:MovieClip = new MovieClip();
var js:String = JSON.stringify(mc); //"prototype toJSON() called."
trace("js: " + js); //"js: toJSON"
```

您也可以在定義 toJSON() 方法的原生類別中覆寫該方法。例如，下列程式碼會覆寫 Date.toJSON()：

```
Date.prototype.toJSON = function (k):* {
    return "any date format you like via toJSON: "+
        "this.time:"+this.time + " this.hours:"+this.hours;
}
var dt:Date = new Date();
trace(JSON.stringify(dt));
// "any date format you like via toJSON: this.time:1317244361947 this.hours:14"
```

定義或覆寫類別層級的 toJSON()

應用程式不需要永遠使用原型來重新定義 toJSON()。您也可以將 toJSON() 定義為子類別的成員 (如果父類別不是標記為 final 的話)。例如，您可以擴充 ByteArray 類別並定義公用 toJSON() 函數：

```
package {

    import flash.utils.ByteArray;
    public class MyByteArray extends ByteArray
    {
        public function MyByteArray() {
        }

        public function toJSON(s:String):*
        {
            return "MyByteArray";
        }

    }
}


var ba:ByteArray = new ByteArray();
trace(JSON.stringify(ba)); //"ByteArray"
var mba:MyByteArray = new MyByteArray(); //"MyByteArray"
trace(JSON.stringify(mba)); //"MyByteArray"
```

如果類別是 dynamic，您可以依照下列方式將 toJSON 屬性加入該類別的物件，並指定函數給它：

```
var d:Dictionary = new Dictionary();
trace(JSON.stringify((d))); // "Dictionary"
d.toJSON = function(){return {c : "toJSON override."};} // overrides existing function
trace(JSON.stringify((d))); // {"c":"toJSON override."}
```

您可以在任何 ActionScript 類別覆寫、定義或重新定義 toJSON()。不過，大部分內建的 ActionScript 類別都未定義 toJSON()。Object 類別在其預設原型中未定義 toJSON，也未將它宣告為類別成員。只有少數原生類別會將該方法定義為原型函數。因此在大部分情況下，您一般無法覆寫 toJSON()。

未定義 toJSON() 的原生類別會由內部 JSON 實作序列化為 JSON。可以的話，請避免取代這個內建功能。如果您定義 toJSON() 成員，JSON 類別會使用您的邏輯，而非它自己的功能。

使用 JSON.stringify() replacer 參數

在原型上覆寫 toJSON() 適用於變更整個應用程式中類別的 JSON 匯出行為。不過，在某些情況中，您的匯出邏輯可能只會短暫套用到特殊案例。若要容納這類小範圍的變更，您可以使用 JSON.stringify() 方法的 replacer 參數。

stringify() 方法會將透過 replacer 參數傳遞的函數套用到所編碼的物件。這個函數的簽名類似於 toJSON()：

```
function (k,v):*
```

與 toJSON() 不同的是，replacer 函數需要值 v 與索引鍵 k。因為 stringify() 是在 JSON 物件上定義，而不是在所編碼的物件，所以會有此差異。當 JSON.stringify() 呼叫 replacer(k,v) 時，其會在原始的輸入物件移動。傳遞給 replacer 函數的隱含 this 參數是指保留索引鍵與值的物件。因為 JSON.stringify() 不會修改原始的輸入物件，所以在容器中移動的該物件保持不變。因此，您可以使用程式碼 this[k] 查詢原始物件上的索引鍵。v 參數會保留 toJSON() 轉換的值。

和 toJSON() 一樣，replacer 函數能夠傳回任何類型的值。如果 replacer 傳回字串，則 JSON 引擎會跳脫引號中的內容，然後也以引號包覆那些跳脫的內容。此包覆可確保 stringify() 收到有效的 JSON 字串物件，即物件保留 JSON.parse() 後續呼叫中的字串。

下列程式碼使用 replacer 參數與隱含的 this 參數，傳回 Data 物件的 time 與 hours 值：

```
JSON.stringify(d, function (k,v):* {
    return "any date format you like via replacer: "+
        "holder[k].time:"+this[k].time + " holder[k].hours:"+this[k].hours;
});
```

使用 JSON.parse() reviver 參數

JSON.parse() 方法的 reviver 參數的作用與 replacer 函數相反：它會將 JSON 字串轉換成可用的 ActionScript 物件。reviver 引數是一個函數，可接受兩個參數並傳回任何類型：

```
function (k,v):*
```

在這個函數中，k 是索引鍵，而 v 是 k 的值。和 stringify() 一樣，parse() 會移動 JSON 索引鍵值配對並將 reviver 函數 (如果有) 套用到每個配對。潛在的問題是，JSON 類別不會輸出物件的 ActionScript 類別名稱。因此，知道何種物件可以復原會是個挑戰。當物件是巢狀結構時，這個問題特別棘手。在設計 toJSON()、replacer 和 reviver 函數時，您可以想出識別所匯出 ActionScript 物件的方法，同時保持原始物件完整性。

剖析範例

下列範例說明復原從 JSON 字串剖析之物件的策略。這個範例會定義兩個類別：JSONGenericDictExample 和 JSONDictionaryExtnExample。類別 JSONGenericDictExample 是自訂 dictionary 類別。每個記錄會包含每人的名字和生日，以及唯一的 ID。每次呼叫 JSONGenericDictExample 建構函式時，都會將新建的物件以及靜態遞增整數 (例如它的 ID) 加入內部靜態陣列中。此外，類別 JSONGenericDictExample 還會定義 revive() 方法，以便從較長的 id 成員中只擷取整數部分。revive() 方法會使用這個整數來尋找並傳回正確的可復原物件。

類別 JSONDictionaryExtnExample 會擴充 ActionScript Dictionary 類別。它的記錄沒有集合結構，因此可以包含任何資料。資料是在建構 JSONDictionaryExtnExample 物件之後指定的，而不是指定做為類別定義的屬性。JSONDictionaryExtnExample 記錄會將 JSONGenericDictExample 物件當做索引鍵使用。當復原 JSONDictionaryExtnExample 物件時，JSONGenericDictExample.revive() 函數會使用與 JSONDictionaryExtnExample 關聯的 ID 來擷取正確的索引鍵物件。

最重要的是，除了 JSONDictionaryExtnExample 物件，JSONDictionaryExtnExample.toJSON() 方法還會傳回標記字串。這個字串會將 JSON 輸出識別為屬於 JSONDictionaryExtnExample 類別。這個標記毫無疑問地成為 JSON.parse() 期間處理的物件類型。

```
package {
    // Generic dictionary example:
    public class JSONGenericDictExample {
        static var revivableObjects = [];
        static var nextId = 10000;
        public var id;
        public var dname:String;
        public var birthday;

        public function JSONGenericDictExample(name, birthday) {
            revivableObjects[nextId] = this;
            this.id       = "id_class_JSONGenericDictExample_" + nextId;
            this.dname     = name;
            this.birthday = birthday;
            nextId++;
        }
        public function toString():String { return this.dname; }
        public static function revive(id:String):JSONGenericDictExample {
            var r:RegExp = /^id_class_JSONGenericDictExample_([0-9]*)$/;
            var res = r.exec(id);
            return JSONGenericDictExample.revivableObjects[res[1]];
        }
    }
```

```
}

package {
    import flash.utils.Dictionary;
    import flash.utils.ByteArray;

    // For this extension of dictionary, we serialize the contents of the
    // dictionary by using toJSON
    public final class JSONDictionaryExtnExample extends Dictionary {
        public function toJSON(k):* {
            var contents = {};
            for (var a in this) {
                contents[a.id] = this[a];
            }

            // We also wrap the contents in an object so that we can
            // identify it by looking for the marking property "class E"
            // while in the midst of JSON.parse.
            return {"class JSONDictionaryExtnExample": contents};
        }

        // This is just here for debugging and for illustration
        public function toString():String {
            var retval = "[JSONDictionaryExtnExample <";
            var printed_any = false;
            for (var k in this) {
                retval += k.toString() + "=" +
                "[e="+this[k].earnings +
                ",v="+this[k].violations + "], "
                printed_any = true;
            }
            if (printed_any)
                retval = retval.substring(0, retval.length-2);
            retval += ">]"
            return retval;
        }
    }
}
```

當下列執行階段指令碼呼叫 JSONDictionaryExtnExample 物件上的 JSON.parse() 時，reviver 函數會呼叫 JSONDictionaryExtnExample 中每個物件上的 JSONGenericDictExample.revive()。這個呼叫會擷取代表物件索引鍵的 ID。JSONGenericDictExample.revive() 函數會使用這個 ID 來擷取並傳回私有靜態陣列中儲存的 JSONDictionaryExtnExample 物件。

```
import flash.display.MovieClip;
import flash.text.TextField;

var a_bob1:JSONGenericDictExample = new JSONGenericDictExample("Bob", new Date(Date.parse("01/02/1934")));
var a_bob2:JSONGenericDictExample = new JSONGenericDictExample("Bob", new Date(Date.parse("05/06/1978")));
var a_jen:JSONGenericDictExample = new JSONGenericDictExample("Jen", new Date(Date.parse("09/09/1999")));

var e = new JSONDictionaryExtnExample();
e[a_bob1] = {earnings: 40, violations: 2};
e[a_bob2] = {earnings: 10, violations: 1};
e[a_jen]  = {earnings: 25, violations: 3};

trace("JSON.stringify(e): " + JSON.stringify(e)); // {"class JSONDictionaryExtnExample":
                        //{"id_class_JSONGenericDictExample_10001":
                        //{"earnings":10,"violations":1},
                        //"id_class_JSONGenericDictExample_10002":
                        //{"earnings":25,"violations":3},
                        //"id_class_JSONGenericDictExample_10000":
                        // {"earnings":40,"violations":2}}}

var e_result = JSON.stringify(e);

var e1 = new JSONDictionaryExtnExample();
var e2 = new JSONDictionaryExtnExample();

// It's somewhat easy to convert the string from JSON.stringify(e) back
// into a dictionary (turn it into an object via JSON.parse, then loop
// over that object's properties to construct a fresh dictionary).
//
// The harder exercise is to handle situations where the dictionaries
// themselves are nested in the object passed to JSON.stringify and
// thus does not occur at the topmost level of the resulting string.
//
// (For example: consider roundtripping something like
//   var tricky_array = [e1, [[4, e2, 6]], {table:e3}]
// where e1, e2, e3 are all dictionaries.  Furthermore, consider
// dictionaries that contain references to dictionaries.)
//
// This parsing (or at least some instances of it) can be done via
// JSON.parse, but it's not necessarily trivial.  Careful consideration
// of how toJSON, replacer, and reviver can work together is
// necessary.

var e_roundtrip =
    JSON.parse(e_result,
               // This is a reviver that is focused on rebuilding JSONDictionaryExtnExample objects.
               function (k, v) {
                     if ("class JSONDictionaryExtnExample" in v) { // special marker tag;
                         //see JSONDictionaryExtnExample.toJSON().
                       var e = new JSONDictionaryExtnExample();
                       var contents = v["class JSONDictionaryExtnExample"];
                       for (var i in contents) {
                           // Reviving JSONGenericDictExample objects from string
                           // identifiers is also special;
                           // see JSONGenericDictExample constructor and
                           // JSONGenericDictExample's revive() method.
                           e[JSONGenericDictExample.revive(i)] = contents[i];
                       }
                       return e;
                     } else {
```

```
                        return v;
                    }
                });

trace("// == Here is an extended Dictionary that has been round-tripped  ==");
trace("// == Note that we have revived Jen/Jan during the roundtrip.    ==");
trace("e:           " + e); //[JSONDictionaryExtnExample <Bob=[e=40,v=2], Bob=[e=10,v=1],
                            //Jen=[e=25,v=3]>]
trace("e_roundtrip: " + e_roundtrip); //[JSONDictionaryExtnExample <Bob=[e=40,v=2],
                                      //Bob=[e=10,v=1], Jen=[e=25,v=3]>]
trace("Is e_roundtrip a JSONDictionaryExtnExample? " + (e_roundtrip is JSONDictionaryExtnExample)); //true
trace("Name change: Jen is now Jan");
a_jen.dname = "Jan"

trace("e:           " + e); //[JSONDictionaryExtnExample <Bob=[e=40,v=2], Bob=[e=10,v=1],
                            //Jan=[e=25,v=3]>]
trace("e_roundtrip: " + e_roundtrip); //[JSONDictionaryExtnExample <Bob=[e=40,v=2],
                                      //Bob=[e=10,v=1], Jan=[e=25,v=3]>]
```

第 8 章 處理事件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

事件處理系統可讓程式設計人員很方便地針對使用者輸入動作與系統事件做出回應。ActionScript 3.0 事件模型不僅方便好用，更符合業界標準，並與顯示清單密切整合。根據「文件物件模型第 3 層事件規格」(Document Object Model (DOM) Level 3 Events Specification，這是遵循業界標準的事件處理架構)，這個新的事件模型可提供 ActionScript 程式設計人員強大且直覺式的事件處理工具。

ActionScript 3.0 事件處理系統與顯示清單的關係密不可分。若要取得顯示清單的基本瞭解，請參閱第 128 頁「顯示程式設計」。

更多說明主題

flash.events 套件

文件物件模型 (DOM) 第 3 層事件規格

處理事件的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以將事件想成是 SWF 檔中發生的任何狀況，它們都想要向身為程式設計人員的您傳達一些重要訊息。例如，大多數的 SWF 檔都支援某些使用者互動 (包括像是回應滑鼠按一下的動作這種簡單的互動，或像是接受並處理輸入至表單中的資料這種較為複雜的互動) 功能。因此，任何這類與 SWF 檔進行的使用者互動，都視為一個事件。事件也會在不直接與使用者進行互動的情況下發生，例如從伺服器完成載入資料，或連接的攝影機變成使用中狀態。

在 ActionScript 3.0 中，每一個事件都會由事件物件所表示，而事件物件就是 Event 類別或其子類別的實體。事件物件不只會儲存關於特定事件的資訊，也包含了協助處理此事件物件的方法。例如，當 Flash Player 或 AIR 偵測到滑鼠按一下的動作時，便會建立事件物件 (MouseEvent 類別的實體) 來表示該特定的滑鼠按一下事件。

在建立事件物件之後，Flash Player 或 AIR 便會「傳送」該事件物件，這表示會將該事件物件傳遞給做為該事件之目標的物件。其中，做為所傳送事件物件之目的地的物件，便稱為「事件目標」。例如，當連接的攝影機成為使用中狀態，Flash Player 便會直接將事件物件傳送給事件目標 (在此情況下則是代表攝影機的物件)。但是，如果該事件目標位於顯示清單中，則事件物件會向下傳遍顯示清單架構，直到到達該事件目標為止。在某些情況下，該事件物件接著便會沿著相同的路線，在顯示清單架構中向上「反昇」。這種顯示清單架構的移動方式便稱為「事件流程」。

您可以使用事件偵聽程式，「偵聽」程式碼中的事件物件。「事件偵聽程式」是您撰寫用來回應特定事件的函數或方法。若要確保程式會對事件做出回應，您必須將事件偵聽程式加入至事件目標，或加入至任何顯示清單物件 (為事件物件之事件流程的一部分) 中。

不管您在何時撰寫事件偵聽程式程式碼，它都會遵循此基本結構 (呈現粗體的元素代表您將在特定個案中填入的預留位置)：

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

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

此程式碼會執行兩個動作：首先會定義函數，以指定要執行的動作來回應事件。接著會呼叫來源物件的 addEventListener() 方法，即由指定的事件 「訂閱」函數，當事件發生時，就會執行函數的動作。當事件真的發生時，事件目標就會檢查已註冊為事件偵聽程式的所有函數與方法清單。接著，它會輪流呼叫每個函數與方法，並將事件物件做為參數傳遞過去。

您需要更改此程式碼中的四個項目，以建立自己的事件偵聽程式。首先，您必須將函數名稱變更為自己想要使用的名稱 (您必須在兩個地方找到有 「eventResponse」字樣的程式碼並加以變更)。再者，您必須為事件物件指定適當的類別名稱，而該事件物件是由您想要偵聽的事件負責傳送 (就是程式碼中的 「EventType」)；同時，您必須為特定事件 (清單中的 「EVENT_NAME」) 指定適當的常數。第三，您必須在即將傳送事件的物件上呼叫 addEventListener() 方法 (在此程式碼中為 「eventTarget」)。或者，您也可以變更當做函數參數使用的變數名稱 (在此程式碼中為 「eventObject」)。

重要概念與術語

下列參考清單包括撰寫事件處理常式時會遇到的相關重要術語：

反昇 某些事件會有反昇現象，讓父顯示物件可回應子項傳送的事件。

反昇階段 事件向上傳播到父顯示物件的事件流程一部分。反昇階段發生於捕捉及目標階段之後。

捕捉階段 事件從最平常的目標向下傳播至最特定目標的事件流程一部分。捕捉階段發生於目標與反昇階段之前。

預設行為 某些事件會包含一種通常會伴隨事件一起發生的行為，稱為預設行為。例如，當使用者在文字欄位中輸入文字，就會引發文字輸入事件。該事件的預設行為就是實際顯示輸入到文字欄位中的字元，但是如果因為某些理由您不希望讓輸入的字元顯示出來的話，也可以覆寫該預設行為。

傳送 通知事件偵聽程式發生了某個事件。

事件 發生在某個物件上的事情，且此物件可告知其它物件關於所發生的事情。

事件流程 當事件發生在顯示清單上的物件時 (顯示在螢幕上的物件)，則包含該物件的所有物件都會收到事件通知，並反過來通知所屬的事件偵聽程式有關該事件。此程序將由 「舞台」開始，並通過顯示清單直到發生事件的實際物件，最後重新回到 「舞台」上。我們將這個程序稱為事件流程。

事件物件 包含特定事件之相關發生資訊的物件；傳送事件時，會將此特定事件傳送給所有偵聽程式。

事件目標 實際傳送事件的物件。例如，如果使用者按一下位於 Sprite (位於 「舞台」內) 內的按鈕時，所有的這些物件都會傳送事件，但是事件目標則是實際發生事件的物件，在這裡指的是按下的按鈕。

偵聽程式 將自己向物件註冊的物件或函數，以表示當發生特定事件時，自己應該收到通知。

目標階段 事件達到最特定可能目標時的事件流程點。目標階段發生於捕捉階段和反昇階段之間。

ActionScript 3.0 與舊版在事件處理方面的差異

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 與舊版 ActionScript 在事件處理方面，最明顯的差異就是 ActionScript 3.0 中只有一種事件處理系統，而舊版 ActionScript 中卻有數種不同的事件處理系統。本節將以概述舊版 ActionScript 中事件處理的運作方式做為開頭，然後再討論 ActionScript 3.0 中在事件處理方面的變更。

舊版 ActionScript 中的事件處理方式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 之前的 ActionScript 版本是提供多種不同的方式來處理事件：

- 可以直接置於 Button 和 MovieClip 實體上的 on() 事件處理常式
- 可以直接置於 MovieClip 實體上的 onClipEvent() 處理常式

- 回呼函數屬性，例如 XML.onload 和 Camera.onActivity
- 使用 addListener() 方法註冊的事件偵聽程式
- 部分實作 DOM 事件模型的 UIEventDispatcher 類別

上述每一種機制都代表了它自己的各項優點與限制。on() 和 onClipEvent() 處理常式都很容易使用，但是在專案的後續維護方面卻較為困難，因為要找出置於按鈕與影片片段上的程式碼可能會有困難。此外，實作回呼函數也很簡單，但卻會限制您只能針對一個指定的事件來實作一個回呼函數。與之相較，實作事件偵聽程式則較為困難，因為您不僅要建立偵聽程式物件與函數，還必須使用可產生事件的物件來註冊此偵聽程式。不過，這些額外的步驟卻能讓您針對相同的事件，建立多個偵聽程式物件並加以註冊。

ActionScript 2.0 的組件開發工作則產生了另一種事件模型，這個包含在 UIEventDispatcher 類別中的新事件模型是以「DOM 事件規格」的子集為基礎，讓熟悉組件事件處理的開發人員都能輕鬆轉換到新的 ActionScript 3.0 事件模型概念。

可惜的是，不同事件模型使用的語法在多方面都有所重疊，而且各有不同。例如，在 ActionScript 2.0 中，某些屬性 (例如 TextField.onChanged) 可以當做回呼函數或事件偵聽程式來使用。然而，視您使用的是支援偵聽程式的六個類別之一或是 UIEventDispatcher 類別，用來註冊偵聽程式物件的語法會有所不同。若是 Key、Mouse、MovieClipLoader、Selection、Stage 和 TextField 類別，您可以使用 addListener() 方法；但為了組件事件處理作業，則必須使用名為 addEventListener() 的方法。

不同的事件處理模型還會產生另一個複雜的情況，就是事件處理常式函數的範圍會根據所使用的機制大相逕庭。換句話說，在多個事件處理系統中，this 關鍵字的意義並不一致。

ActionScript 3.0 中的事件處理方式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 導入了單一事件處理模型，它取代了舊版程式語言中許多不同的事件處理機制。這個新的事件模型是以「文件物件模型第 3 層事件規格」(Document Object Model (DOM) Level 3 Events Specification) 為基礎。雖然 SWF 檔格式並非特別遵守「文件物件模型」標準，然而在顯示清單與 DOM 結構之間，還是有足夠的相似性，讓實作 DOM 事件模型這件事成為可能。顯示清單中的物件就像是 DOM 階層架構中的節點，而且本段論述中的「顯示清單物件」和「節點」都可以互換使用。

Flash Player 和 AIR 在實作 DOM 事件模型時，加入了名為預設行為的概念。所謂「預設行為」，就是 Flash Player 或 AIR 會當做特定事件之一般結果來執行的動作。

預設行為

通常，開發人員必須負責撰寫可回應事件的程式碼。然而，在某些情況下，某個行為常常會與事件 (除非開發人員新增程式碼來取消該行為，否則 Flash Player 或 AIR 會自動執行該行為) 產生關聯。因為 Flash Player 或 AIR 會自動顯示該行為，所以這種行為便稱為預設行為。

例如，當使用者在 TextField 物件中輸入文字，則通常會期待 TextField 物件中出現該文字，因此這項行為會內建在 Flash Player 或 AIR 中。如果您不要這項預設行為發生，可以使用新的事件處理系統來將其取消。當使用者在 TextField 物件中輸入文字，Flash Player 或 AIR 便會建立 TextEvent 類別的實體，以代表該使用者的輸入動作。若要避免 Flash Player 或 AIR 在 TextField 物件中顯示該文字，您必須存取此特定的 TextEvent 實體，並呼叫此實體的 preventDefault() 方法。

請注意，不是所有的預設行為都能加以避免。例如，當使用者按兩下 TextField 物件中的某個單字時，Flash Player 或 AIR 便會產生 MouseEvent 物件。這項無法避免的預設行為，就是游標下方的單字會呈現反白顯示。

許多類型的事件物件都沒有相關聯的預設行為。例如，當建立網路連線時，Flash Player 便會傳送連線物件，但是並沒有預設行為與它有關聯。Event 類別及其子類別的 API 說明文件會列出每一個事件的類型，並描述任何相關聯的預設行為，以及該行為是否能避免。

您必須知道，預設行為只會與 Flash Player 或 AIR 所傳送的事件物件有關聯，而且透過 ActionScript 以程式設計方式傳送之事件物件的預設行為則不存在。例如，您可以使用 EventDispatcher 類別的方法，傳送類型為 textInput 的事件物件，但是該事件物件將不具有與它相關聯的預設行為。換句話說，Flash Player 和 AIR 將不會在 TextField 物件中顯示字元做為 textInput 事件 (以程式設計的方式傳送) 的結果。

ActionScript 3.0 中事件偵聽程式的新增功能

新的規格對熟悉使用 ActionScript 2.0 addListener() 方法的開發人員而言，將有助於他們瞭解 ActionScript 2.0 事件偵聽程式模型與 ActionScript 3.0 事件模型之間的差異。下表說明這兩個事件模型的幾個主要差異點：

- 若要在 ActionScript 2.0 中新增事件偵聽程式，那麼在某些情況下您必須使用 addListener()，而其它情況下則必須使用 addEventListener()；而在 ActionScript 3.0 中，針對所有的情況您只需使用 addEventListener() 即可。
- ActionScript 2.0 中沒有事件流程，這表示您只能針對廣播事件的物件呼叫 addListener() 方法；而在 ActionScript 3.0 中，則可以針對屬於事件流程一部分的任何物件呼叫 addEventListener() 方法。
- 在 ActionScript 2.0 中，事件偵聽程式可以是函數、方法或物件，而在 ActionScript 3.0 中，只有函數或方法可以做為事件偵聽程式。

事件流程

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

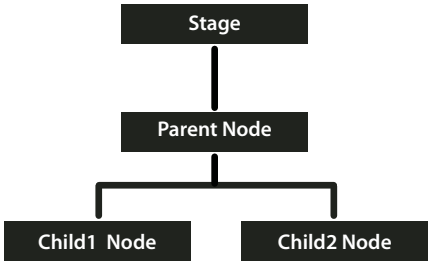
每當發生事件時，Flash Player 或 AIR 都會傳送事件物件。如果事件目標不是顯示清單，Flash Player 或 AIR 就會直接將事件物件傳送至該事件目標。例如，Flash Player 會將進度事件物件直接傳送至 URLStream 物件。但是，如果該事件目標位於顯示清單中，則 Flash Player 會將事件物件傳送至顯示清單，而且該事件物件會傳遍顯示清單，直到到達該事件目標為止。

「事件流程」會說明事件物件在顯示清單中移動的方式。顯示清單是以階層的形式所組成，能以樹狀結構來說明。顯示清單階層的頂端就是 Stage，它是特殊的顯示物件容器，作用就像是顯示清單的根。這個 Stage 是由 flash.display.Stage 類別來表示，而且只能透過顯示物件來存取。每一個顯示物件都有名為 stage 的屬性，此屬性指的就是該應用程式的 「舞台」。

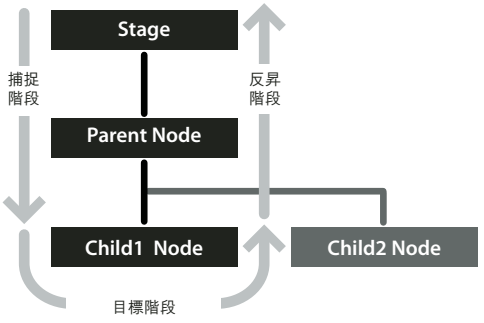
當 Flash Player 或 AIR 針對與顯示清單相關的事件傳送事件物件時，會將該事件物件從 Stage 傳送至 「目標節點」，然後再回到 Stage。「DOM 事件規格」是將目標節點定義為代表事件目標的節點。換句話說，目標節點也就是在其位置發生該事件的顯示清單物件。例如，假設使用者按一下名為 child1 的顯示清單物件，Flash Player 或 AIR 便會以 child1 做為目標節點來傳送事件物件。

事件流程在概念上可分為三個階段：第一個階段稱為 「捕捉階段」，這個階段是由從 Stage 到目標節點之間的所有節點組成。第二個階段稱為「目標階段」，這個階段是由目標節點單獨組成。第三個階段稱為 「反昇階段」 (Bubbling)，這個階段是由事件物件從目標節點的父輩回到 Stage 途中所遇到的節點組成。

如果您將顯示清單認為是 Stage 在頂端的垂直式階層架構，則這些階段的名稱可以幫助您更容易瞭解，如下圖所示：



如果使用者按一下「Child1 Node」，Flash Player 或 AIR 便會將事件物件傳送至事件流程中。如下圖所示，該物件會從 Stage 向下移動到 Parent Node，接著再移動到 Child1 Node，然後「反昇」回到 Stage。在返回 Stage 期間，該物件會再次經過 Parent Node。



在此範例中，捕捉階段包含在首次向下移動期間經過的 Stage 和 Parent Node；目標階段包含在 Child1 Node 所耗費的時間；而反昇階段則包含在向上移動回到根節點期間所遇到的 Parent Node 和 Stage。

這種事件流程可讓 ActionScript 程式設計人員使用功能更強大的事件處理系統。在舊版 ActionScript 中並沒有事件流程，這表示您只能將事件偵聽程式加入至可產生事件的物件；而在 ActionScript 3.0 中，您不僅可以將事件偵聽程式加入至目標節點，也可以加入至事件流程中的任一節點。

當使用者介面組件包含一個以上的物件時，將事件偵聽程式加入至事件流程是很有用的功能。例如，按鈕物件通常會包含當做該按鈕物件之標籤的文字物件。如果不能將事件偵聽程式加入至事件串流，您就必須將偵聽程式加入至該按鈕與文字物件，以確保按一下按鈕時，能夠收到按一下事件。然而，事件物件能夠讓您將單一事件偵聽程式置於按鈕物件上，以處理發生在文字物件、或是文字物件未遮蔽到之按鈕物件區域的按一下事件。

然而，不是每一個事件物件都會參與事件流程的所有三個階段。某些類型 (例如 enterFrame 和 init 事件類型) 的事件會直接傳送至目標節點，並且不會參與捕捉階段或反昇階段；而其它事件則可能會將目標設定為不在顯示清單中的物件，例如傳送至 Socket 類別之實體的事件。這些事件物件也將會直接傳送至目標物件，而不會參與捕捉階段與反昇階段。

若要得知某個特定事件類型的行為，您可以檢查 API 說明文件，或檢查該事件物件的屬性。下節將針對檢查事件物件的屬性進行說明。

事件物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在新的事件處理系統中，事件物件的主要作用有兩種：第一，事件物件會將特定事件的相關資訊儲存在一組屬性中，以代表實際的事件。第二，事件物件包含一組方法，可讓您處理這些事件物件，並影響事件處理系統的行為。

為了能方便存取這些屬性和方法，Flash Player API 會定義做為所有事件物件之基底類別的 Event 類別。Event 類別會定義一組對所有事件物件都通用的基本屬性與方法。

本節將以對 Event 類別屬性的論述做為開頭，並進而說明 Event 類別方法，然後再以 Event 類別的子類別為何存在做為結語。

瞭解 Event 類別的屬性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Event 類別會定義數個唯讀屬性及常數，它們可提供關於事件物件的重要資訊。其中，下列項目更是特別重要：

- 事件物件類型會以常數表示，並且會儲存在 Event.type 屬性中。

- 是否能避免事件的預設行為會以 Boolean 值表示，並且會儲存在 Event.cancelable 屬性中。
- 事件流程資訊會包含在其餘的屬性中。

事件物件類型

每一個事件物件都有相關聯的事件類型，而且這些事件類型都會以字串值的形式儲存在 Event.type 屬性中。知道事件物件的類型會很有用，因為這樣可以讓您的程式碼能分辨不同類型的物件。例如，下列程式碼會指定 clickHandler() 偵聽程式函數必須對任何傳遞給 myDisplayObject 的滑鼠按一下事件物件做出回應：

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

約有 24 種事件類型與 Event 類別本身相關聯，並且會以 Event 類別常數來表示。以下顯示其中某些類型 (摘錄自 Event 類別定義)：

```
package flash.events
{
    public class Event
    {
        // class constants
        public static const ACTIVATE:String = "activate";
        public static const ADDED:String= "added";
        // remaining constants omitted for brevity
    }
}
```

這些常數可提供簡單的方式，讓您參考特定的事件類型。您應該使用這些常數，而非這些常數所代表的字串。如果您在程式碼中拼錯了常數名稱，編譯器將能捕捉到錯誤；但是如果您使用的是字串，則在編譯期間可能就不會出現打字上的錯誤，進而造成難以除錯的未預期行為。例如，新增事件偵聽程式時，請使用下列程式碼：

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

而不要使用下列程式碼：

```
myDisplayObject.addEventListener("click", clickHandler);
```

預設行為資訊

您的程式碼會存取 cancelable 屬性，檢查所有特定事件物件的預設行為是否能加以避免。cancelable 屬性具有 Boolean 值，可指出是否能避免某項預設行為。您可以使用 preventDefault() 方法，避免 (或取消) 與一小部分事件相關聯的預設行為。如需詳細資訊，請參閱第 114 頁 「瞭解 Event 類別的方法」下的 「取消預設事件行為」。

事件流程資訊

其餘的 Event 類別屬性不但包含關於事件物件的重要資訊，也包含了事件物件與事件流程的關聯性資訊。下表將會加以說明：

- bubbles 屬性包含關於事件物件有參與其中之事件流程的資訊。
- eventPhase 屬性代表目前事件流程中的階段。
- target 屬性會儲存事件目標的參考。
- currentTarget 屬性會儲存目前正在處理事件物件之顯示清單物件的參考。

bubbles 屬性

如果某個事件的事件物件參與事件流程的反昇階段，該稱為該事件在 「反昇」。這表示該事件物件是從目標節點傳遞經過它的祖先，直到回到 Stage 為止。Event.bubbles 會儲存 Boolean 值，它會指出該事件物件是否參與反昇階段。因為所有反昇的事件也都有參與捕捉及目標階段，這就表示任何反昇的事件都會參與全部三個事件流程階段。如果值為 true，就表示該事件物件有參與全部三個階段。如果值為 false，就表示該事件物件沒有參與反昇階段。

eventPhase 屬性

您可以調查任何事件物件的 eventPhase 屬性，判斷該事件物件的事件階段。eventPhase 屬性包含無正負號的整數值，表示事件流程中的其中一個階段。Flash Player API 會定義不同的 EventPhase 類別，它包含三個常數，分別對應至三個無正負號的整數值，如下列摘錄的程式碼所示：

```
package flash.events
{
    public final class EventPhase
    {
        public static const CAPTURING_PHASE:uint = 1;
        public static const AT_TARGET:uint = 2;
        public static const BUBBLING_PHASE:uint= 3;
    }
}
```

這些常數會對應至 eventPhase 屬性的三個有效值。您可以使用這些常數，讓程式碼更易於閱讀。如果要確保只有在事件目標位於目標舞台中時，才呼叫名為 myFunc() 的函數，您可以使用下列程式碼來測試這個條件：

```
if (event.eventPhase == EventPhase.AT_TARGET)
{
    myFunc();
}
```

target 屬性

target 屬性具有物件的參考，而該物件為事件的目標。在某些情況下，這會是直接參考，例如當麥克風成為使用中狀態時，事件物件的目標就是 Microphone 物件。不過，如果該目標位於顯示清單中，您就必須將顯示清單階層列入考慮。例如，假設使用者以滑鼠按一下某個點，而且這個點包含重疊的顯示清單物件，則 Flash Player 和 AIR 就一定會選擇距離 Stage 最遠的物件做為事件目標。

對於複雜的 SWF 檔，特別是其中通常以較小子物件來裝飾的檔案，可能就不會經常使用 target 屬性，因為這個屬性通常是指向某個按鈕的子物件，而非該按鈕。在這些情況下，一般的作法就是將事件偵聽程式加入至該按鈕，並使用 currentTarget 屬性 (因為它會指向該按鈕，而 target 屬性則是指向該按鈕的子系)。

currentTarget 屬性

currentTarget 屬性包含正在處理事件物件之物件的參考。雖然不去瞭解目前是哪個節點正在處理您正在檢查的事件物件似乎很奇怪，但是請記住，您可以將偵聽程式函數加入至該事件物件之事件流程中的任何顯示物件，並且可以將該偵聽程式函數置於任意位置。此外，您也可以將相同的偵聽程式函數加入至不同的顯示物件。隨著專案的大小和複雜程度提高，currentTarget 屬性也會變得愈來愈有用。

瞭解 Event 類別的方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Event 類別方法共有三種：

- 公用程式方法，這些方法可以建立事件物件的副本，或將事件物件轉換為字串
- 事件流程方法，這些方法會從事件流程移除事件物件
- 預設行為方法，這些方法會避免預設行為，或檢查是否已避免該行為

Event 類別的公用程式方法

Event 類別中有兩種公用程式方法。clone() 方法可讓您建立事件物件的副本。toString() 方法可讓您產生事件物件之屬性的字串表示，以及這些屬性的值。這兩種方法都是由事件模型系統在內部使用，但是會公開讓開發人員做為一般用途。

對於建立 Event 類別之子類別的進階開發人員，您必須覆寫並實作這兩種公用程式方法的版本，以確保此事件子類別能正常運作。

停止事件流程

您可以呼叫 Event.stopPropagation() 或 Event.stopImmediatePropagation() 方法，避免事件物件在事件流程中持續傳遞。這兩個方法除了是否允許目前節點的其它事件偵聽程式執行之外，幾乎完全相同：

- Event.stopPropagation() 方法可避免事件物件移動到下一個節點，但是只有在允許執行目前節點上的其它任何事件偵聽程式之後，才能這麼做。
- Event.stopImmediatePropagation() 方法也能避免事件物件移動到下一個節點，但是卻不會允許執行目前節點上的其它任何事件偵聽程式。

呼叫上述兩方法的其中一個，並不會影響與某個事件相關聯的預設行為是否會發生。您可以使用 Event 類別的預設行為方法，避免預設行為發生。

取消預設事件行為

與取消預設行為有關的兩個方法為 preventDefault() 和 isDefaultPrevented() 方法。您可以呼叫 preventDefault() 方法，取消與某個事件相關聯的預設行為。若要檢查是否已針對事件物件呼叫 preventDefault()，請呼叫 isDefaultPrevented() 方法，如果已經呼叫該方法，它會傳回 true 值；否則會傳回 false。

只有在可以取消事件的預設行為時，preventDefault() 方法才能運作。您可以參考該事件類型的 API 說明文件，或是使用 ActionScript 檢查該事件物件的 cancelable 屬性，藉以得知目前的情況是否符合上述條件。

取消預設行為並不會對事件流程中事件物件的傳遞進度造成影響。您可以使用 Event 類別的事件流程方法，從事件流程中移除事件物件。

Event 類別的子類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

對許多事件來說，Event 類別中所定義的一般屬性集已經足夠使用。不過，其它事件卻都具有無法以 Event 類別中可用之屬性所捕捉的唯一特性。針對這些事件，ActionScript 3.0 會定義數個 Event 類別的子類別。

每一個子類別都會提供額外的屬性和事件類型，它們對於這些事件的分類而言，都是獨一無二的。例如，與滑鼠輸入動作有關的事件都具有數個唯一特性，這些特性都無法以 Event 類別中所定義的屬性加以捕捉。藉由新增包含像是滑鼠事件位置，以及在滑鼠事件期間是否有按下特定按鍵這類資訊的十項屬性，MouseEvent 類別便可擴充 Event 類別。

Event 子類別也包含代表與該子類別相關聯之事件類型的常數。例如，MouseEvent 類別會為數個滑鼠事件類型定義常數，這些事件類型包括 click、doubleClick、mouseDown 和 mouseUp。

如第 112 頁 「事件物件」底下關於 Event 類別公用程式方法的內容所述，建立 Event 子類別時，您必須覆寫 clone() 和 toString() 方法，以提供該子類別專屬的功能。

事件偵聽程式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

事件偵聽程式 (也稱為事件處理常式) 是 Flash Player 和 AIR 為回應特定事件而執行的函數。新增事件偵聽程式需要兩個步驟。首先，您必須建立函數或類別方法，讓 Flash Player 或 AIR 可用來回應事件。這有時又稱為偵聽程式函數或事件處理常式函數。接著，您必須使用 addEventListener() 方法註冊偵聽程式函數，並設定該事件的目標或位於適當事件流程中的顯示清單物件。

建立偵聽程式函數

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 事件模型與 DOM 事件模型不同的其中一個部分，就是在建立偵聽程式函數這方面。在 DOM 事件模型中，事件偵聽程式與偵聽程式函數之間有著清楚的區別：事件偵聽程式是會實作 EventListener 介面之類別的實體，而偵聽程式函數則是名為 handleEvent() 之類別的方法。在 DOM 事件模型中，您註冊的是包含偵聽程式函數的類別實體，而非實際的偵聽程式函數。

在 ActionScript 3.0 事件模型中，事件偵聽程式與偵聽程式函數之間並沒有清楚的區別。ActionScript 3.0 並沒有 EventListener 介面，因此偵聽程式函數可以在類別之外定義，或是定義為類別的一部分。此外，偵聽程式函數不一定要名為 handleEvent()，而是能以任何有效的識別項來命名。在 ActionScript 3.0 中，您註冊的是實際偵聽程式函數的名稱。

在類別之外定義的偵聽程式函數

下列程式碼會建立簡單的 SWF 檔，並顯示紅色正方形。名為 clickHandler() 的偵聽程式函數 (不屬於類別的一部分) 會偵聽紅色正方形上的滑鼠按一下事件。

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
}

function clickHandler(event:MouseEvent):void
{
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
}
```

當使用者按一下該正方形，並與產生的 SWF 檔進行互動時，Flash Player 或 AIR 便會產生下列追蹤輸出：

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

請注意，該事件物件會當做引數傳遞給 clickHandler()。這可讓您的偵聽程式函數檢查該事件物件。在此範例中，您會使用該事件物件的 type 屬性，查明該事件為 click 事件。

此範例也會檢查 this 關鍵字的值。在此範例中，this 代表全域物件，這是可以理解的，因為這個函數是在任何自訂類別或物件之外所定義。

定義為類別方法的偵聽程式函數

下列範例與定義 ClickExample 類別的上一個範例完全相同，但是 clickHandler() 函數卻定義為 ChildSprite 類別的方法：

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
    private function clickHandler(event:MouseEvent):void
    {
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
    }
}
```

當使用者按一下該紅色正方形，並與產生的 SWF 檔進行互動時，Flash Player 或 AIR 便會產生下列追蹤輸出：

```
clickHandler detected an event of type: click
the this keyword refers to: [object ChildSprite]
```

請注意，this 會參考名為 child 的 ChildSprite 實體。這是與 ActionScript 2.0 不同的行為變更。如果您使用 ActionScript 2.0 中的組件，就可能還記得，當類別方法傳遞至 UIEventDispatcher.addEventListener() 中時，該方法的範圍就會限定為廣播該事件的組件，而非在其中定義偵聽程式方法的類別。換句話說，如果您使用這項 ActionScript 2.0 的技巧，this 關鍵字就會參考廣播該事件的組件，而非 ChildSprite 實體。

對某些程式設計人員而言，這是個重大的問題，因為這表示他們無法存取該類別 (包含偵聽程式方法) 的其它方法和屬性。為瞭解決此問題，ActionScript 2.0 的程式設計人員會使用 mx.util.Delegate 類別，變更偵聽程式方法的範圍。不過，因為 ActionScript 3.0 會在呼叫 addEventListener() 方法時建立繫結方法，所以便不再需要上述作法。結果，this 關鍵字便會參考名為 child 的 ChildSprite 實體，如此程式設計人員便能存取 ChildSprite 類別的其它方法和屬性。

不應該使用的事件偵聽程式

此外，還有第三種技巧，您可以使用這項技巧建立一般物件，而且該物件的屬性會指向以動態方式指派的偵聽程式函數，但是這項技巧並不建議使用。我們在此討論它的原因是因為 ActionScript 2.0 常常會用到它，但是在 ActionScript 3.0 請勿使用它。不建議使用這個技巧的原因，是因為 this 關鍵字會參考全域物件而不是您的偵聽程式物件。

下列範例與上一個 ClickExample 類別範例完全相同，但是偵聽程式函數卻定義為名為 myListenerObj 之一般物件的一部分：

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, myListenerObj.clickHandler);
    }
}

var myListenerObj:Object = new Object();
myListenerObj.clickHandler = function (event:MouseEvent):void
{
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
}
```

追蹤的結果，看起來如下所示：

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

您期望 this 會參考 myListenerObj，並且追蹤輸出會是 [object Object]，但是它卻改為參考全域物件。當您將動態屬性名稱當做引數傳遞給 addEventListener() 時，Flash Player 或 AIR 便無法建立繫結方法。這是因為您傳遞做為 listener 參數的只不過是偵聽程式函數的記憶體位址，而且 Flash Player 和 AIR 無法將這個記憶體位址與 myListenerObj 實體加以連結。

管理事件偵聽程式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 IEventDispatcher 介面的方法，管理您的偵聽程式函數。IEventDispatcher 介面是 DOM 事件模型之 EventTarget 介面的 ActionScript 3.0 版本。雖然 IEventDispatcher 這個名稱看起來暗指它的主要目的是要發送 (或傳送) 事件物件，但是此類別的方法卻更常用於註冊、檢查及移除事件偵聽程式。IEventDispatcher 介面會定義五個方法，如下列程式碼中所示：

```
package flash.events
{
    public interface IEventDispatcher
    {
        function addEventListener(eventName:String,
                        listener:Object,
                        useCapture:Boolean=false,
                        priority:Integer=0,
                        useWeakReference:Boolean=false):Boolean;

        function removeEventListener(eventName:String,
                    listener:Object,
                    useCapture:Boolean=false):Boolean;

        function dispatchEvent(eventObject:Event):Boolean;

        function hasEventListener(eventName:String):Boolean;
        function willTrigger(eventName:String):Boolean;
    }
}
```

Flash Player API 會使用 EventDispatcher 類別實作 IEventDispatcher 介面，這個類別是做為所有可以是事件目標或事件流程之一部分類別的基底類別來使用。例如，DisplayObject 類別便是繼承自 EventDispatcher 類別。這表示顯示清單中的任何物件都可以存取 IEventDispatcher 介面的方法。

新增事件偵聽程式

addEventListener() 對 IEventDispatcher 介面而言是非常有用的方法，您可以使用這個方法來註冊偵聽程式函數。使用這個方法需要搭配 type 和 listener 參數。type 參數是用來指定事件的類型，而 listener 參數則是用來指定當事件發生時，將會執行的偵聽程式函數。listener 參數可以是某個函數或類別方法的參考。

當您指定 listener 參數時，請勿使用括號。例如，在下列對 addEventListener() 方法的呼叫中，指定 clickHandler() 函數時是不使用括號的：

```
addEventListener(MouseEvent.CLICK, clickHandler)
```

addEventListener() 方法的 useCapture 參數可讓您控制偵聽程式將在其中成為使用中狀態的事件流程階段。如果將 useCapture 設定為 true，您的偵聽程式在事件流程的捕捉階段期間將會成為使用中狀態。如果將 useCapture 設定為 false，您的偵聽程式在事件流程的目標與反昇階段期間將會成為使用中狀態。若要在事件流程的所有階段都偵聽某個事件，您必須呼叫 addEventListener() 兩次，第一次先將 useCapture 設定為 true，另一次再將 useCapture 設定為 false。

addEventListener() 方法的 priority 參數並不屬於正式的 DOM 第 3 層事件模型的一部分。它包含在 ActionScript 3.0 中，以便讓您在組織事件偵聽程式時能更具有彈性。在呼叫 addEventListener() 時，您便可以傳遞整數值做為 priority 參數，藉以設定該事件偵聽程式的優先順序。預設值為 0，但是您可以將這個值設為負整數或正整數。數字愈高代表會愈快執行該事件偵聽程式。具有相同優先順序的事件偵聽程式則會依照它們加入時的順序來執行，所以愈早加入的偵聽程式就會愈早執行。

useWeakReference 參數可讓您指定偵聽程式函數的參考為弱參考或是一般參考。將此參數設定為 true 可讓您避免偵聽程式函數一直存在於記憶體中 (即使不再需要這些函數也一樣) 的情況。Flash Player 和 AIR 會使用名為「記憶體回收」的技巧，清理記憶體中不再使用的物件。如果對某個物件的參考不存在，就表示已將該物件視為不再使用。垃圾回收器並不在乎弱參考，這表示具有指向本身之參考的偵聽程式函數也符合垃圾收集的條件。

移除事件偵聽程式

您可以使用 removeEventListener() 方法，移除不再需要的事件偵聽程式。移除不再用到的事件偵聽程式是很好的作法。與 addEventListener() 方法所需的參數一樣，使用上述方法必須搭配 eventName 和 listener 參數。請回想之前的內容，您可以呼叫 addEventListener() 兩次，第一次先將 useCapture 設定為 true，另一次再將它設定為 false，在所有事件階段都偵聽事件。若要移除這兩個事件偵聽程式，您必須呼叫 removeEventListener() 兩次，第一次先將 useCapture 設定為 true，另一次再將它設定為 false。

傳送事件

進階程式設計人員可以使用 dispatchEvent() 方法，將自訂事件物件傳送到事件流程中。此方法唯一會採用的參數，就是事件物件的參考，這個參考必須是 Event 類別的實體或是子類別。一旦傳送之後，此事件物件的 target 屬性會設定為當做 dispatchEvent() 所呼叫之目標的物件。

檢查現有的事件偵聽程式

IEventDispatcher 介面的最後兩個方法可提供關於事件偵聽程式存在狀況的有用資訊。如果在某個清單物件上有找到特定事件類型的事件偵聽程式，hasEventListener() 方法就會傳回 true。如果有找到特定顯示清單物件的偵聽程式，willTrigger() 方法也會傳回 true，但是 willTrigger() 不僅會檢查該顯示物件上的偵聽程式，也會檢查事件流程之所有階段的顯示清單物件之全部祖先上的偵聽程式。

不含偵聽程式的錯誤事件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

例外 (而非事件) 是 ActionScript 3.0 中錯誤處理的主要機制，但是例外處理並不適用於非同步作業 (例如載入檔案)。如果在這種非同步作業期間發生錯誤，Flash Player 和 AIR 便會傳送錯誤事件物件。如果您沒有為此錯誤事件建立偵聽程式，Flash Player 和 AIR 的除錯程式版本便會顯示包含該項錯誤資訊的對話方塊。例如，Flash Player 的除錯程式版本會產生下列對話方塊，描述當應用程式嘗試從無效的 URL 載入檔案時，所發生的錯誤：

大多數錯誤事件都是以 ErrorEvent 類別為基礎，而且都具有名為 text 的屬性，它會用來儲存 Flash Player 或 AIR 顯示的錯誤訊息。但是，StatusEvent 和 NetStatusEvent 類別則是例外。這兩個類別都有 level 屬性 (StatusEvent.level 和 NetStatusEvent.info.level)。當 level 屬性的值為 「error」時，就會將這些事件類型視為錯誤事件。

錯誤事件並不會導致 SWF 檔停止執行，它會在瀏覽器外掛程式和獨立播放程式的除錯程式版本中顯示為對話方塊、在編寫播放程式的輸出面板中顯示為訊息，而在 Adobe Flash Builder 的記錄檔則顯示為其中的一個項目。而在 Flash Player 或 AIR 的發行版本中，它則完全不會顯示。

事件處理範例：鬧鐘

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

「鬧鐘」範例包含可讓使用者指定響鈴時間的時鐘，以及要在該時間顯示的訊息。「鬧鐘」範例是根據第 1 頁 「使用日期與時間」的 SimpleClock 應用程式所建置，它會說明在 ActionScript 3.0 中處理事件的數項觀點，其中包括：

- 偵聽及回應事件
- 通知事件的偵聽程式

- 建立自訂事件類型

若要取得此樣本的 Flash Professional 應用程式檔案，請參閱 http://www.adobe.com/go/learn_programmingAS3samples_flash_tw。若要取得此樣本的 Flex 應用程式檔案，請參閱 http://www.adobe.com/go/as3examples_tw。您可以在 Samples/AlarmClock 檔案夾中找到 「鬧鐘」應用程式檔案。這個應用程式包含下列檔案：

檔案	說明
AlarmClockApp.mxml 或 AlarmClockApp.fla	主應用程式檔案，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/clock/AlarmClock.as	會擴充 SimpleClock 類別的類別，以增加鬧鐘的功能。
com/example/programmingas3/clock/AlarmEvent.as	自訂的事件類別 (flash.events.Event 的子類別)，當做 AlarmClock 類別之 alarm 事件的事件物件來使用。
com/example/programmingas3/clock/AnalogClockFace.as	繪製圓形時鐘外觀，並根據時間繪製時針、分針和秒針 (已在 SimpleClock 範例中加以說明)。
com/example/programmingas3/clock/SimpleClock.as	具有簡單計時功能的時鐘介面組件 (已在 SimpleClock 範例中加以說明)。

鬧鐘概觀

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

本範例中鬧鐘的主要功能 (包含追蹤時間及顯示鬧鐘外觀) 會重複使用 SimpleClock 應用程式程式碼，這已在第 5 頁 「日期與時間範例：簡易類比時鐘」中加以說明。AlarmClock 類別會從該範例中，藉由加入鬧鐘所需的功能來擴充 SimpleClock 類別，這些功能包括設定響鈴時間，以及當響鈴 「響起」時提供通知。

建立事件的原因，就是為了工作中有事發生時能夠提供通知。AlarmClock class 會公開 Alarm 事件，為了執行所需的動作，其它物件都可以偵聽此事件。此外，AlarmClock 類別還會使用 Timer 類別的實體，以判斷何時要觸發它的響鈴。與 AlarmClock 類別相似，Timer 類別也可以在經過特定時間之後，提供事件來通知其它物件 (在此範例中，則為 AlarmClock 實體)。和大多數 ActionScript 應用程式一樣，事件構成了 「鬧鐘」樣本應用程式的一部分重要功能。

觸發響鈴

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如先前內容所提及，AlarmClock 類別實際提供的唯一功能會與設定和觸發響鈴有關。內建的 Timer 類別 (flash.utils.Timer) 可提供一種方式，讓開發人員定義在指定的時間量之後執行的程式碼。AlarmClock 類別會使用 Timer 實體，判斷何時要觸發響鈴。

```
import flash.events.TimerEvent;
import flash.utils.Timer;

/**
 * The Timer that will be used for the alarm.
 */
public var alarmTimer:Timer;
...
/**
 * Instantiates a new AlarmClock of a given size.
 */
public override function initClock(faceSize:Number = 200):void
{
    super.initClock(faceSize);
    alarmTimer = new Timer(0, 1);
    alarmTimer.addEventListener(TimerEvent.TIMER, onAlarm);
}
```

在 AlarmClock 類別中定義的 Timer 實體名為 alarmTimer。initClock() 方法 (會執行 AlarmClock 實體所需的設定作業) 搭配 alarmTimer 變數時，會執行兩件事。首先，它會搭配參數 (指示 Timer 實體等待 0 毫秒) 以實體化該變數，並且只觸發其 timer 事件一次。實體化 alarmTimer 之後，程式碼便會呼叫該變數的 addEventListener() 方法，表示它要偵聽該變數的 timer 事件。在經過指定的時間量之後，藉由傳送 Timer 實體的 timer 事件，讓此實體得以運作。AlarmClock 類別必須知道傳送 timer 事件的時間，以便觸發它自己的響鈴。藉由呼叫 addEventListener()，AlarmClock 程式碼便可以使用 alarmTimer 將本身註冊為偵聽程式。這兩個參數表示 AlarmClock 類別要偵聽 timer 事件 (由 TimerEvent.TIMER 常數所表示)，以及當事件發生時，要呼叫 AlarmClock 類別的 onAlarm() 方法來回應該事件。

為了要實際設定響鈴，會呼叫 AlarmClock 類別的 setAlarm() 方法，如下所示：

```
/**
 * Sets the time at which the alarm should go off.
 * @param hour The hour portion of the alarm time.
 * @param minutes The minutes portion of the alarm time.
 * @param message The message to display when the alarm goes off.
 * @return The time at which the alarm will go off.
 */
public function setAlarm(hour:Number = 0, minutes:Number = 0, message:String = "Alarm!"):Date
{
    this.alarmMessage = message;
    var now:Date = new Date();
    // Create this time on today's date.
    alarmTime = new Date(now.fullYear, now.month, now.date, hour, minutes);

    // Determine if the specified time has already passed today.
    if (alarmTime <= now)
    {
        alarmTime.setTime(alarmTime.time + MILLISECONDS_PER_DAY);
    }

    // Stop the alarm timer if it's currently set.
    alarmTimer.reset();
    // Calculate how many milliseconds should pass before the alarm should
    // go off (the difference between the alarm time and now) and set that
    // value as the delay for the alarm timer.
    alarmTimer.delay = Math.max(1000, alarmTime.time - now.time);
    alarmTimer.start();

    return alarmTime;
}
```

這個方法會執行數項工作，其中包含儲存響鈴訊息以及建立 Date 物件 (alarmTime) 代表響鈴要響起時的實際時間。與目前論述最有關聯的部分，也就是此方法的最後幾行，會設定並啟動 alarmTimer 變數的計時器。首先，呼叫它的 reset() 方法，如果計時器正在執行的話，這麼做便會將計時器停止並重新設定。接著，再從 alarmTime 變數的值中減去目前的時間 (以 now 變數表示)，以判斷在響鈴響起之前還需多少毫秒。由於 Timer 類別不會在絕對時間觸發它的 timer 事件，所以這會是指定給 alarmTimer 之 delay 屬性的這個相對時間差。最後會呼叫 start() 方法，實際啟動計時器。

一旦經過指定的時間量之後，alarmTimer 便會傳送 timer 事件。由於 AlarmClock 類別已經將本身的 onAlarm() 方法註冊為該事件的偵聽程式，所以當 timer 事件發生時，就會呼叫 onAlarm()。

```
/**
 * Called when the timer event is dispatched.
 */
public function onAlarm(event:TimerEvent):void
{
    trace("Alarm!");
    var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
    this.dispatchEvent(alarm);
}
```

註冊為事件偵聽程式的方法必須以適當的記號 (亦即此方法的參數集與傳回類型) 來定義。若要成為 Timer 類別之 timer 事件的偵聽程式，方法必須定義一個參數，而且此參數的資料類型必須是 TimerEvent (flash.events.TimerEvent)，它是 Event 類別的子類別。當 Timer 實體呼叫本身的事件偵聽程式時，它會傳遞 TimerEvent 實體做為事件物件。

通知其它偵聽程式有關此響鈴事件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

與 Timer 類別相似，AlarmClock 類別也可以在響鈴響起時，提供可允許其它程式碼接收通知的事件。對於要使用內建於 ActionScript 中之事件處理架構的類別而言，此類別必須實作 flash.events.IEventDispatcher 介面。最常見的作法是藉由擴充 flash.events.EventDispatcher 類別的方式來完成上述實作，這種方式能提供 IEventDispatcher 的標準實作 (或是藉由擴充 EventDispatcher 的其中一項子類別來完成)。如先前所述，AlarmClock 類別延伸出 SimpleClock 類別，而後者又 (透過一連串繼承) 延伸出 EventDispatcher 類別。這些全部都表示 AlarmClock 類別已經有內建的功能來提供它自己的事件。

藉由呼叫 AlarmClock 從 EventDispatcher 繼承的 addEventListener() 方法，其它程式碼便可以註冊以接收 AlarmClock 類別之 alarm 事件通知。當 AlarmClock 實體已準備好通知其它程式碼該實體的 alarm 已經引發時，就會藉由呼叫 dispatchEvent() 方法來完成，這個方法也是繼承自 EventDispatcher。

```
var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
this.dispatchEvent(alarm);
```

下列程式碼是取自 AlarmClock 類別的 onAlarm() 方法 (已顯示在它之前的整個程式碼中)。呼叫 AlarmClock 實體的 dispatchEvent() 方法，這樣接著便會通知所有已觸發 AlarmClock 實體之 alarm 事件的已註冊偵聽程式。傳遞給 dispatchEvent() 的參數，就是將一併傳遞給偵聽程式方法的事件物件。在此範例中，它是 AlarmEvent 類別的實體，此類別是特別建立供這個範例使用的 Event 子類別。

提供自訂的 alarm 事件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

所有事件偵聽程式都會接收事件物件參數 (包含遭觸發之特定事件的相關資訊)。在許多情況下，這個事件物件都會是 Event 類別的實體。然而，在某些情況下，提供事件偵聽程式額外的資訊會很有用。通常若要達到這個目的，您就必須定義新的類別 (Event 類別的子類別)，並將此新類別的實體當做事件物件來使用。在此範例中，當傳送 AlarmClock 類別的 alarm 事件時，AlarmEvent 實體便會當做事件物件來使用。這裡所顯示的 AlarmEvent 類別會提供關於 alarm 事件的額外資訊，特別是響鈴訊息：

```
import flash.events.Event;

/**
 * This custom Event class adds a message property to a basic Event.
 */
public class AlarmEvent extends Event
{
    /**
     * The name of the new AlarmEvent type.
     */
    public static const ALARM:String = "alarm";

    /**
     * A text message that can be passed to an event handler
     * with this event object.
     */
    public var message:String;

    /**
     *Constructor.
     *@param message The text to display when the alarm goes off.
     */
    public function AlarmEvent(message:String = "ALARM!")
    {
        super(ALARM);
        this.message = message;
    }
    ...
}
```

建立自訂事件物件類別的最佳方式，就是定義會擴充 Event 類別的類別，如先前的範例所示。為了補充所繼承的功能，AlarmEvent 類別會定義 message 屬性，此屬性包含了與該事件有關聯的響鈴訊息文字；message 值則會當做 AlarmEvent 建構函式中的參數來傳入。AlarmEvent 也會定義 ALARM 常數，此常數可以用來在呼叫 AlarmClock 類別的 addEventListener() 方法時，參考特定的事件 (alarm)。

除了加入自訂的功能外，每一個 Event 子類別都必須覆寫所繼承的 clone() 方法，做為 ActionScript 事件處理架構的一部分。Event 子類別也可以選擇性地覆寫所繼承的 toString() 方法，以便在呼叫 toString() 方法時，將自訂事件的屬性包含在傳回的值中。

```
/**
 * Creates and returns a copy of the current instance.
 * @return A copy of the current instance.
 */
public override function clone():Event
{
    return new AlarmEvent(message);
}

/**
 * Returns a String containing all the properties of the current
 * instance.
 * @return A string representation of the current instance.
 */
public override function toString():String
{
    return formatToString("AlarmEvent", "type", "bubbles", "cancelable", "eventPhase", "message");
}
```

遭到覆寫的 clone() 方法必須傳回自訂 Event 子類別的新實體，並設定所有的自訂屬性以符合目前的實體。在遭到覆寫的 toString() 方法中，會使用公用程式方法 formatToString() (繼承自 Event) 以提供具有自訂類型名稱的字串，以及所有其屬性的名稱與值。

第 9 章 使用應用程式網域

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ApplicationDomain 類別的目的是用於儲存 ActionScript 3.0 的定義表格。SWF 檔中的所有程式碼都定義為存在應用程式網域中。您可以使用應用程式網域，分割位於相同安全性網域中的類別。這麼做能允許相同的類別具有多項定義，並讓子系重新使用父輩定義。

您可以在使用 Loader 類別 API 載入以 ActionScript 3.0 撰寫的外部 SWF 檔時，使用應用程式網域 (請注意，當您載入以 ActionScript 1.0 或 ActionScript 2.0 撰寫的影像或 SWF 檔時，將無法使用應用程式網域)。所有包含在載入類別中的 ActionScript 3.0 定義都會儲存在應用程式網域中。載入 SWF 檔時，您可以將 LoaderContext 物件的 applicationDomain 參數設定為 ApplicationDomain.currentDomain，指定將該檔案包含在與 Loader 物件檔案相同的應用程式網域中。藉由將載入的 SWF 檔置於相同的應用程式網域中，便可以直接存取其類別。如果您要載入包含內嵌媒體 (透過媒體相關類別名稱進行存取) 的 SWF 檔，或是要存取所載入 SWF 檔的方法，這就十分有用。

下列範例假設它可以存取不同的 Greeter.swf 檔案，該檔案定義公用方法 welcome()：

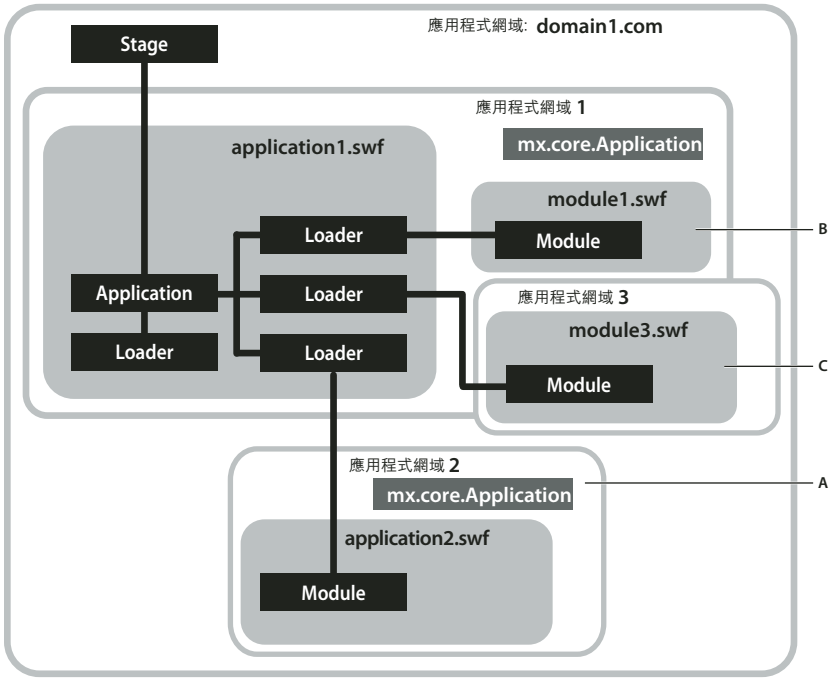
```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLRequest;
    import flash.system.ApplicationDomain;
    import flash.system.LoaderContext;

    public class ApplicationDomainExample extends Sprite
    {
        private var ldr:Loader;
        public function ApplicationDomainExample()
        {
            ldr = new Loader();
            var req:URLRequest = new URLRequest("Greeter.swf");
            var ldrContext:LoaderContext = new LoaderContext(false, ApplicationDomain.currentDomain);
            ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, completeHandler);
            ldr.load(req, ldrContext);
        }
        private function completeHandler(event:Event):void
        {
            var myGreeter:Class = ApplicationDomain.currentDomain.getDefinition("Greeter") as Class;
            var myGreeter:Greeter = Greeter(event.target.content);
            var message:String = myGreeter.welcome("Tommy");
            trace(message); // Hello, Tommy
        }
    }
}
```

另請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中的 ApplicationDomain 類別範例。

當您處理應用程式網域時，其它要注意的事項包括：

- SWF 檔中的所有程式碼都定義為存在應用程式網域中。「目前的網域」是指您的主應用程式在其中執行的網域。「系統網域」則包含所有的應用程式網域，其中包括目前的網域，也就是包含了所有的 Flash Player 類別。
- 所有的應用程式網域，除了系統網域之外，都具有相關的父網域。您的主應用程式之應用程式網域的父網域，就是系統網域。只有在其父輩尚未定義載入的類別時，才會定義這些類別。因此，您不能以較新的定義覆寫載入類別的定義。

下圖顯示在單一網域 domain1.com 中，從各種不同 SWF 檔載入內容的應用程式。根據所載入的內容而定，可能會用到不同的應用程式網域。下方的文字說明在應用程式中用來為每一個 SWF 檔設定適當應用程式網域的邏輯。



A. 用法 A **B.** 用法 B **C.** 用法 C

主應用程式檔案為 application1.swf，它包含會從其它 SWF 檔載入內容的 Loader 物件。在此案例中，目前的網域是 「應用程式網域 1」。用法 A、用法 B 和用法 C 各說明不同的技巧，可用來在應用程式中為每一個 SWF 檔設定適當的應用程式網域。

用法 A 藉由建立系統網域的子系，分割子 SWF 檔。在圖中，「應用程式網域 2」是建立做為系統網域的子網域，application2.swf 檔會載入至 「應用程式網域 2」中，而其類別定義就會從 application1.swf 所定義的類別中分割出來。

此技巧的其中一項用法，就是讓舊的應用程式以動態方式載入較新版的相同應用程式，而不造成衝突。沒有發生衝突的原因，是因為雖然使用了相同的類別名稱，但是類別都已經分割而位於不同的應用程式網域中。

下列程式碼會建立系統網域子系的應用程式網域，並開始使用該應用程式網域載入 SWF：

```
var appDomainA:ApplicationDomain = new ApplicationDomain();

var contextA:LoaderContext = new LoaderContext(false, appDomainA);
var loaderA:Loader = new Loader();
loaderA.load(new URLRequest("application2.swf"), contextA);
```

用法 B: 將新的類別定義新增至目前的類別定義。module1.swf 的應用程式網域設定為目前的網域 (應用程式網域 1)，這可讓您在應用程式目前的類別定義集中，加入新的類別定義。這可以用來做為主應用程式的執行階段共享元件庫。所載入的 SWF 會視為執行階段共享元件庫 (RSL)。您可以使用這項技巧，在應用程式開始之前，預先載入器載入 RSL。

下列程式碼會載入 SWF，並將其應用程式網域設定為目前的網域：

```
var appDomainB:ApplicationDomain = ApplicationDomain.currentDomain;

var contextB:LoaderContext = new LoaderContext(false, appDomainB);
var loaderB:Loader = new Loader();
loaderB.load(new URLRequest("module1.swf"), contextB);
```

用法 C: 透過建立目前網域的新子網域，使用父項目的類別定義。module3.swf 的應用程式網域是目前網域的子網域，這個子網域會使用父輩所有類別的版本。此技巧的其中一項用法，是可以做為具有多重螢幕之多樣化網頁應用程式 (Rich Internet

Application，RIA) 的模組，此模組會載入為主應用程式的子系，並使用主應用程式的類型。如果您能確保所有類別都一定會升級以與舊版相容，而且載入內容的應用程式一定會比它所載入的內容更新，則子系將會使用父輩版本。如果您確定不會繼續參考子 SWF，則具有新的應用程式網域也可讓您取消載入所有的類別定義，以便進行記憶體回收。

這項技巧可讓載入模組共享載入器的 Singleton 物件和靜態類別成員。

下列程式碼會建立目前網域的新子網域，並使用該應用程式網域開始載入 SWF：

```
var appDomainC:ApplicationDomain = new ApplicationDomain(ApplicationDomain.currentDomain);

var contextC:LoaderContext = new LoaderContext(false, appDomainC);
var loaderC:Loader = new Loader();
loaderC.load(new URLRequest("module3.swf"), contextC);
```

第 10 章 顯示程式設計

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

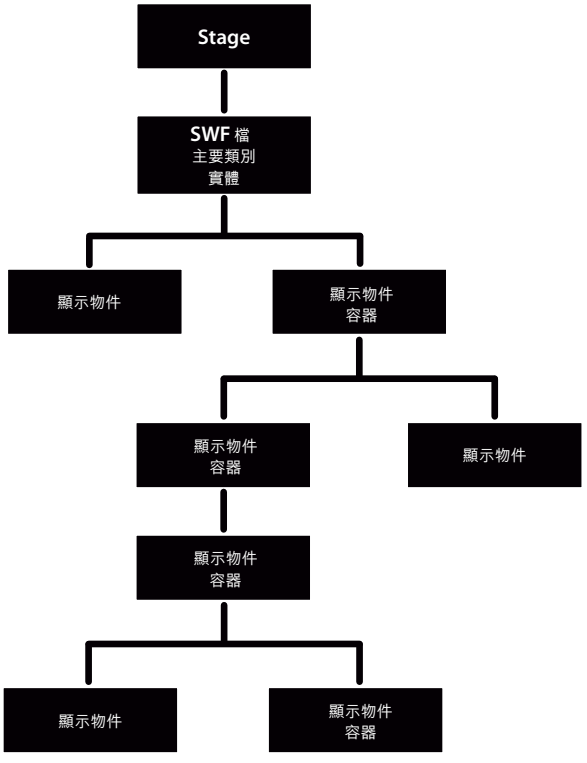
在 Adobe® ActionScript® 3.0 中的視覺元素程式設計，會搭配顯示舞台中的顯示物件。例如，您可使用 ActionScript 顯示程式設計 API，新增、移動、移除與排序顯示物件、套用濾鏡與遮色片、繪製向量與點陣圖形、執行三維變形。用於顯示程式設計的主要類別是 flash.display 套件的一部分。

備註：Adobe® AIR™ 提供 HTMLLoader 物件，以顯示 HTML 內容。HTMLLoader 會將 HTML DOM 視覺元素顯示為單一顯示物件。您無法透過 ActionScript 顯示清單階層，直接存取 DOM 的個別元件。而必須使用 HTMLLoader 提供的 DOM API，存取這些 DOM 元件。

顯示程式設計的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

利用 ActionScript 3.0 建立的每一個應用程式都有顯示物件的階層架構，稱為 「顯示清單」。如下圖所示：顯示清單包含應用程式中的所有可見元素，



如上圖所示，顯示元素會落在下列其中一個或多個群組：

- Stage

  Stage (舞台) 是顯示物件的基礎容器。每一個應用程式都有一個 Stage 物件，其中包含所有螢幕上的顯示物件。Stage 是最高階容器，位於顯示清單階層架構的頂端：

  每個 SWF 檔都有相關聯的 ActionScript 類別，稱為「SWF 檔的主要類別」。當 Flash Player 或 Adobe AIR 開啟 SWF 檔時，Flash Player 或 AIR 會呼叫該類別的建構函式，而所建立的實體 (永遠都是一種顯示物件) 就會做為 Stage 物件的子系來加入。SWF 檔的主要類別一律會擴充 Sprite 類別 (如需詳細資訊，請參閱第 132 頁 「採用顯示清單的優點」)。

  您可以透過任何 DisplayObject 實體的 stage 屬性來存取此 Stage。如需詳細資訊，請參閱第 139 頁 「設定 Stage 的屬性」。

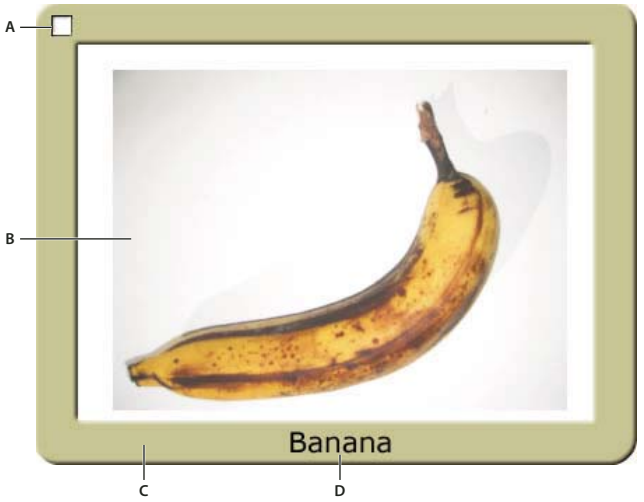
- 顯示物件

  在 ActionScript 3.0 中，出現在應用程式中螢幕上的所有元素都是一種 「顯示物件」。flash.display 套件包含一個 DisplayObject 類別，這是由其他數個類別擴充的基礎的類別。這些不同的類別都各代表不同的顯示物件類型，例如，向量形狀、影片片段，以及文字欄位，這只是列舉其中幾個而已。如需這些類別的一覽表，請參閱第 132 頁 「採用顯示清單的優點」。

- 顯示物件容器

  顯示物件容器是特殊類型的顯示物件，除了具有自己的視覺呈現以外，也可以包含本身也是顯示物件的子物件。

  DisplayObjectContainer 類別是 DisplayObject 類別的子類別。DisplayObjectContainer 物件可以在其 「子清單」中包含多個顯示物件。例如，下圖顯示一種 DisplayObjectContainer 物件，又稱為 Sprite，其中包含各種顯示物件：



A. SimpleButton 物件。這種顯示物件有不同的 「一般」、「按下」和 「滑入」狀態。 **B.** Bitmap 物件。在此範例中，Bitmap 物件是透過 Loader 物件從外部 JPEG 載入。 **C.** Shape 物件。「圖片邊框」包含在 ActionScript 中繪製的圓角矩形。這個 Shape 物件套用了 「投影」濾鏡。 **D.** TextField 物件。

在討論顯示物件的內容中，DisplayObjectContainer 物件也稱為 「顯示物件容器」，或簡稱 「容器」。上文中也已提及，Stage 是顯示物件容器。

雖然所有可見的顯示物件都是從 DisplayObject 類別繼承的，每個物件的類型都是 DisplayObject 類別的特定子類別。例如，Shape 類別或 Video 類別有建構函式，但 DisplayObject 類別並沒有建構函式。

重要概念與術語

下列參考清單包含 ActionScript 圖形程式設計時會遇到的相關重要術語：

Alpha 代表色彩透明度份量的顏色值 (或者更正確地說，不透明度的份量)。例如，Alpha 色版值為 60% 的顏色會顯示完整色調的 60% 以及 40% 的透明度。

點陣圖像 在電腦上定義為彩色像素格線 (列與欄) 的圖像。點陣圖像包含數位相片和類似的影像。

混合模式 兩個重疊影像的內容互動方式的規格。上方的不透明影像通常會遮住下方影像，導至下方影像無法顯示出來，然而混合模式可以讓影像的顏色以不同方式相互混合，以產生結合兩個影像的內容。

顯示清單 會由 Flash Player 和 AIR 呈現為可見畫面內容的顯示物件階層。Stage 是顯示清單的根，而附加在 Stage 或是其子系之一的所有顯示物件會形成顯示清單 (即使未實際呈現的物件也是如此，如位在 Stage 邊緣之外的物件)。

顯示物件 在 Flash Player 或 AIR 中代表某種可見內容的物件。顯示清單只能包含顯示物件，而所有的顯示物件類別都是 DisplayObject 類別的子類別。

顯示物件容器 顯示物件的一種特殊類型，(通常) 除了具有本身的視覺表示之外，還包含子顯示物件。

SWF 檔的主要類別 此為定義 SWF 檔中最外層顯示物件之行為的類別，這在概念上即是 SWF 檔本身的類別。例如，在以 Flash 編寫工具所建立的 SWF 中，主要類別是文件類別。它具有包含其它所有時間軸的「主要時間軸」，而時間軸為實體的類別即為 SWF 檔的主要類別。

遮色片 遮住影像特定部分的技巧 (或者反過來，僅顯示影像的特定部分)。影像被遮蓋的部分會變成透明，使底下的內容能夠顯示出來。這個術語和遮色帶有關，也就是畫家避免將顏料畫到特定區域時所使用的工具。

舞台 視覺容器，為 SWF 中所有視覺內容的基底或背景。

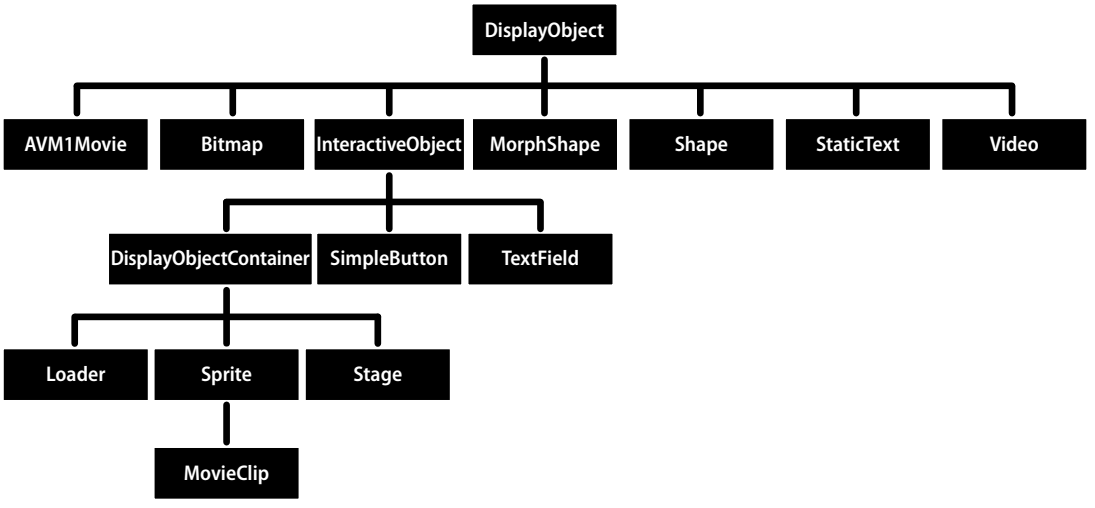
變形 調整圖像的視覺特性，如旋轉物件、改變比例、傾斜或扭曲物件形狀或改變顏色。

向量圖像 在電腦上，以特定特性 (如粗細、長度、尺寸、角度和位置) 繪製的線條和形狀所定義的圖像。

核心顯示類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 的 flash.display 套件包含可出現在 Flash Player 或 AIR 中的視覺物件類別。下圖顯示這些核心顯示物件類別的子類別關係。



下圖示範顯示物件類別的類別繼承。請注意，這些類別之中有一些並未包含在 flash.display 套件中，但仍然是繼承自 DisplayObject 類別，尤其是 StaticText、TextField 和 Video。

擴充 DisplayObject 類別的所有類別也會同時繼承其方法和屬性。如需詳細資訊，請參閱：第 134 頁 「DisplayObject 類別的屬性和方法」。

您可以對包含於 flash.display 中的下列類別之物件進行實體化：

- Bitmap：您可以使用 Bitmap 類別定義點陣圖物件，方式可以是從外部檔案載入，或透過 ActionScript 呈現。您可以透過 Loader 類別，從外部檔案載入點陣圖。您可以載入 GIF、JPG 或 PNG 檔案；也可以用自訂資料建立 BitmapData 物件，然後再使用該資料建立 Bitmap 物件。不管是載入或是在 ActionScript 中建立的點陣圖，您都可以使用 BitmapData 類別的方法加以改變。如需詳細資訊，請參閱第 167 頁 「載入顯示物件」和第 205 頁 「使用點陣圖」。
- Loader：您可以使用 Loader 類別，載入外部資源 (SWF 檔或圖像)。如需詳細資訊，請參閱第 167 頁 「以動態方式載入顯示內容」。
- Shape：您可以使用 Shape 類別建立向量圖像，例如矩形、線段、圓形等等。如需詳細資訊，請參閱第 187 頁 「使用繪圖 API」。
- SimpleButton：SimpleButton 物件是以 Flash 編寫工具建立之按鈕元件的 ActionScript 形式。SimpleButton 實體有四種按鈕狀態：一般、按下、滑入和碰撞測試 (回應滑鼠和鍵盤事件的區域)。
- Sprite：Sprite 物件可以包含本身的圖像，也可以包含子顯示物件 (Sprite 類別會擴充 DisplayObjectContainer 類別)。如需詳細資訊，請參閱第 135 頁 「使用顯示物件容器」和第 187 頁 「使用繪圖 API」。
- MovieClip：MovieClip 物件是以 Flash 編寫工具建立之 ActionScript 形式的影片片段元件。實際作業時，MovieClip 類似於 Sprite 物件，只不過它也有時間軸。如需詳細資訊，請參閱第 272 頁 「使用影片片段」。

下列不包含於 flash.display 套件中的類別都是 DisplayObject 類別的子類別：

- TextField 類別包含於 flash.text 套件中，是文字顯示及輸入的顯示物件。如需詳細資訊，請參閱第 313 頁 「使用文字的基本概念」。
- flash.text.engine 套件所包括的 TextLine 類別，是用以顯示 Flash 文字引擎與 Text Layout Framework 所編輯文字行的顯示物件。如需詳細資訊，請參閱第 336 頁 「使用 Flash 文字引擎」與第 362 頁 「使用 Text Layout Framework」。
- Video 類別包含於 flash.media 套件中，是用來顯示視訊檔案的顯示物件。如需詳細資訊，請參閱第 403 頁 「使用視訊」。

下列 flash.display 套件中的類別會擴充 DisplayObject 類別，但不能您建立這些類別的實體。這些類別只能做為其它顯示物件的父類別，組合一般功能成為單一類別。

- AVM1Movie：AVM1Movie 類別是用來代表載入用 ActionScript 1.0 和 2.0 編寫的 SWF 檔。
- DisplayObjectContainer：Loader、Stage、Sprite 和 MovieClip 類別都會擴充 DisplayObjectContainer 類別。如需詳細資訊，請參閱第 135 頁 「使用顯示物件容器」。
- InteractiveObject：InteractiveObject 是用來與滑鼠及鍵盤互動之所有物件的基底類別。SimpleButton、TextField、Loader、Sprite、Stage 和 MovieClip 物件全部都是 InteractiveObject 類別的子類別。如需有關建立滑鼠及鍵盤互動的詳細資訊，請參閱第 475 頁 「使用者互動的基本概念」。
- MorphShape：這些物件是在以 Flash 編寫工具建立形狀補間動畫時建立。您不能使用 ActionScript 將它們實體化，但可以從顯示清單中存取。
- Stage：Stage 類別會擴充 DisplayObjectContainer 類別。一個應用程式中有一個 Stage 實體，位於顯示清單階層架構的頂端。若要存取 Stage，請使用任何 DisplayObject 實體的 stage 屬性。如需詳細資訊，請參閱第 139 頁 「設定 Stage 的屬性」。

flash.text 套件中的 StaticText 類別也會擴充 DisplayObject 類別，但您不能用程式碼建立它的實體；只有使用 Flash 才能建立靜態文字欄位。

下列類別不是顯示物件或顯示物件容器，而且不會出現在顯示清單中，但是會在舞台上顯示圖形。這些類別會繪製成稱為「檢視區域」的矩形，其位置與舞台相對。

- StageVideo：StageVideo 類別會盡可能使用硬體加速以顯示視訊內容。此類別從 Flash Player 10.2 開始提供。如需詳細資訊，請參閱第 435 頁「使用硬體加速呈現方式的 StageVideo 類別」。
- StageWebView：StageWebView 類別會顯示 HTML 內容。此類別從 AIR 2.5 開始提供。如需詳細資訊，請參閱 第 886 頁「StageWebView 物件」。

下列 fl.display 類別提供使 flash.display.Loader 和 LoaderInfo 類別平行的功能。如果您在 Flash Professional 環境 (CS5.5 或更新的版本) 中開發，請使用這些類別，而不要使用它們的 flash.display 對應類別。在該環境中，這些類別有助於解決 RSL 預先載入時與 TLF 相關的問題。如需詳細資訊，請參閱第 170 頁「使用 ProLoader 與 ProLoaderInfo 類別」。

- fl.display.ProLoader— 類似於 flash.display.Loader
- fl.display.ProLoaderInfo— 類似於 flash.display.LoaderInfo

採用顯示清單的優點

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 ActionScript 3.0 中，不同類型的顯示物件有不同的類別；在 ActionScript 1.0 和 2.0 中，許多相同類型的物件全部都會包含在 MovieClip 類別中。

這種類別的個別性及顯示清單的階層架構有下列優點：

- 顯示呈現更有效率，並減少記憶體用量
- 更完善的深度管理
- 完整的顯示清單移動方式
- 清單外顯示物件
- 更容易產生顯示物件的子類別

顯示呈現更有效率，而且檔案更小

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 ActionScript 1.0 和 2.0 中，您只能在 MovieClip 物件中繪製形狀；在 ActionScript 3.0 中，則有更簡單的顯示物件類別，您可以在其中繪製形狀。由於這些 ActionScript 3.0 顯示物件類別都沒有 MovieClip 物件所包含的完整方法和屬性，因此記憶體和處理器資源的負擔較輕。

例如，每一個 MovieClip 物件都包含影片片段時間軸的屬性，而 Shape 物件則不包含這些屬性。管理時間軸的屬性可能會耗用許多記憶體及處理器資源，在 ActionScript 3.0 中，使用 Shape 物件可以改善效能，Shape 物件的額外負荷比複合 MovieClip 物件少。Flash Player 和 AIR 都不需要管理未使用的 MovieClip 屬性，因此可以提升速度，並減少物件所耗費的記憶體使用量。

更完善的深度管理

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 ActionScript 1.0 和 2.0 中，深度會透過線性深度管理配置和 getNextHighestDepth() 之類的方法來管理。

ActionScript 3.0 則包含 DisplayObjectContainer 類別，其中有更便利的方法和屬性可用來管理顯示物件的深度。

在 ActionScript 3.0 中，當您將顯示物件移至 DisplayObjectContainer 實體之子清單中的新位置時，顯示物件容器中的其它子系都會自動重新放置，並在顯示物件容器中指定適當的子索引位置。

同時在 ActionScript 3.0 中，一定可以找到所有顯示物件容器的所有子物件。每一個 DisplayObjectContainer 實體都有 numChildren 屬性，其中列出顯示物件容器中的子系數目，而由於顯示物件容器的子清單永遠都是索引清單，您可以從索引位置 0 檢查清單中的每一個物件，一直檢查到最後的索引位置 (numChildren - 1)；而在 ActionScript 1.0 和 2.0 中，MovieClip 物件的方法和屬性就不可能這樣。

在 ActionScript 3.0 中，您可以輕易地依序移動經過顯示清單。顯示物件容器子清單的索引數目之間沒有空隙。在移動顯示清單及管理物件深度方面，也較 ActionScript 1.0 和 2.0 更簡單。在 ActionScript 1.0 及 2.0 中，影片片段可能包含在深度順序有斷續間隔的物件，使得物件清單的移動變得相當困難。而在 ActionScript 3.0 中，顯示物件容器的每一份子清單都是由內部快取為陣列，因此在 (依索引) 查閱時非常快速，循環執行顯示物件容器的所有子系迴圈也一樣非常快速。

在 ActionScript 3.0 中，您也可以使用 DisplayObjectContainer 類別的 getChildByName() 方法，存取顯示物件容器中的子系。

完整的顯示清單移動方式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 ActionScript 1.0 和 2.0 中，您無法存取一些以 Flash 編寫工具繪製的物件，例如向量形狀；而在 ActionScript 3.0 中，您可以存取顯示清單上的所有物件，包括使用 ActionScript 及以 Flash 編寫工具建立的所有顯示物件。如需詳細資訊，請參閱第 138 頁「在顯示清單中移動」。

清單外顯示物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 ActionScript 3.0 中，您可以建立不在可見顯示清單上的顯示物件，這些物件稱為「清單外」顯示物件。顯示物件只有在您呼叫已加入顯示清單之 DisplayObjectContainer 實體的 addChild() 或 addChildAt() 方法時，才會加入可見的顯示清單中。

您可以使用清單外顯示物件組合成複雜的顯示物件，如具有包含多個顯示物件的多個顯示物件容器的顯示物件。您可以透過將顯示物件放在清單外來組合複雜的物件，而不必耗費處理時間來呈現這些顯示物件，然後在需要時，再將清單外顯示物件加入至顯示清單。而且，您也可以隨意將顯示物件容器的子系移入或移出顯示清單，或移至顯示清單中任何想要的位置。

更容易產生顯示物件的子類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 ActionScript 1.0 和 2.0 中，您必須經常將新的 MovieClip 物件加入 SWF 檔中，以建立基本的形狀或顯示點陣圖；而在 ActionScript 3.0 中，DisplayObject 類別包含許多內建的子類別，其中就包括了 Shape 和 Bitmap。由於 ActionScript 3.0 中的類別更專屬於特定物件類型，因此更容易建立內建類別的基本子類別。

例如，若要在 ActionScript 2.0 中繪製圓形，您可以建立 CustomCircle 類別，在自訂類別的物件實體化時，擴充 MovieClip 類別。但是，該類別也會包含一些來自 MovieClip 類別的屬性和方法 (例如 totalFrames)，其實並不適用於此類別；而在 ActionScript 3.0 中，您可以建立擴充 Shape 物件的 CustomCircle 類別，因此就不會納入 MovieClip 類別中所含不相關的屬性和方法。下列程式碼會示範 CustomCircle 類別的範例：

```
import flash.display.*;

public class CustomCircle extends Shape
{
    var xPos:Number;
    var yPos:Number;
    var radius:Number;
    var color:uint;
    public function CustomCircle(xInput:Number,
                                     yInput:Number,
                                     rInput:Number,
                                     colorInput:uint)
    {
        xPos = xInput;
        yPos = yInput;
        radius = rInput;
        color = colorInput;
        this.graphics.beginFill(color);
        this.graphics.drawCircle(xPos, yPos, radius);
    }
}
```

處理顯示物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

現在，既然已經瞭解 Stage、顯示物件、顯示物件容器及顯示清單的基本概念，本節要提供您一些有關處理 ActionScript 3.0 中顯示物件的更明確資訊。

DisplayObject 類別的屬性和方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

所有顯示物件都是 DisplayObject 類別的子類別，因此它們都繼承了 DisplayObject 類別的屬性和方法，繼承的屬性是適用於所有顯示物件的基本屬性。例如，每一個顯示物件都有 x 屬性和 y 屬性，指定物件在其顯示物件容器中的位置。

您不能使用 DisplayObject 類別建構函式建立 DisplayObject 實體，而是必須建立如 Sprite 這種另一類型的物件 (屬於 DisplayObject 類別之子類別的物件)，以便用 new 運算子將物件實體化。而且，若要建立自訂顯示物件類別，您必須建立顯示物件的其中一個子類別，此子類別具有可使用的建構函式 (如 Shape 類別或 Sprite 類別)。如需詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中的 DisplayObject 類別說明。

將顯示物件加入顯示清單

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

實體化顯示物件時，它並不會出現在螢幕上 (在「舞台」上)，一直要到將顯示物件實體加入至顯示清單上的顯示物件容器時才會出現。例如，在下列程式碼中，若省略程式碼最後一行，就看不見 myText TextField 物件。在最後一行程式碼中，this 關鍵字必須參考已經加入顯示清單中的顯示物件容器。

```
import flash.display.*;
import flash.text.TextField;
var myText:TextField = new TextField();
myText.text = "Buenos dias.";
this.addChild(myText);
```

當您將任何視覺元素加入 Stage 時，該元素即成為此 Stage 物件的「子系」。載入應用程式中的第一個 SWF 檔 (例如，嵌入 HTML 網頁中的檔案) 會自動加入成為 Stage 的子系，它可以是擴充 Sprite 類別的任何類型物件。

您「不使用」ActionScript 所建立的任何顯示物件 (例如，透過在 Flex MXML 檔案中加入 MXML 標籤，或透過將項目放在 Flash Professional 中的「舞台」) 都會加入至顯示清單。雖然您並沒有透過 ActionScript 加入這些顯示物件，但是卻可以透過 ActionScript 加以存取。例如，下列程式碼會調整 (未透過 ActionScript) 加入編寫工具中而名為 button1 的物件寬度：

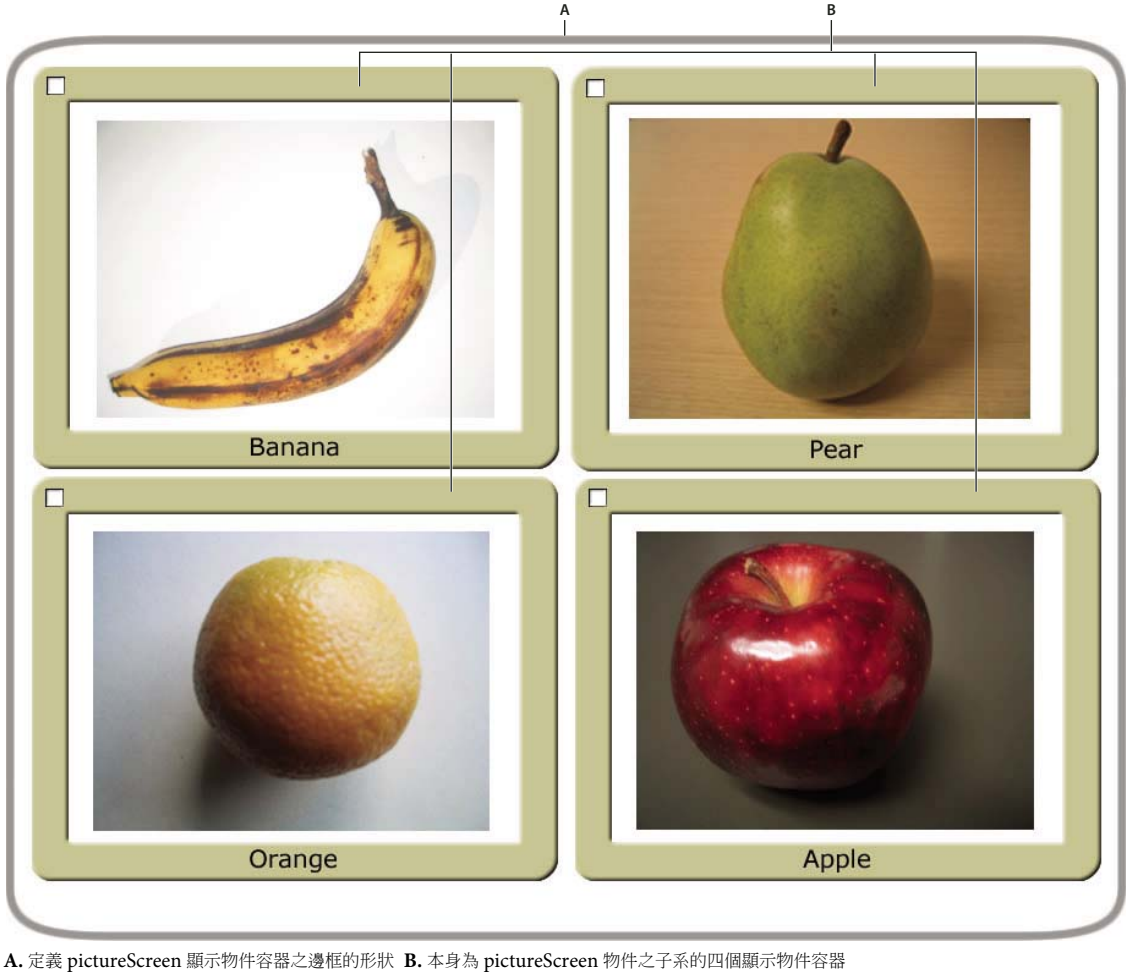
```
button1.width = 200;
```

使用顯示物件容器

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若從顯示清單刪除 DisplayObjectContainer 物件，或是以其它方式移動或轉換，則也會刪除、移動或轉換 DisplayObjectContainer 中的各個顯示物件。

顯示物件容器本身就是一種顯示物件，它可以加入至另一個顯示物件容器中。例如，下列影像會顯示一個顯示物件容器 pictureScreen，其中包含一個邊框形狀及另外四個顯示物件容器 (屬 PictureFrame 類型)：



A. 定義 pictureScreen 顯示物件容器之邊框的形狀 **B.** 本身為 pictureScreen 物件之子系的四個顯示物件容器

若要讓顯示物件出現在顯示清單中，您必須將它加入顯示清單的顯示物件容器中。您可以使用 addChild() 方法或容器物件的 addChildAt() 方法，執行這項作業。例如，若沒有下列程式碼的最後一行，就不會顯示 myTextField 物件。

```
var myTextField:TextField = new TextField();
myTextField.text = "hello";
this.root.addChild(myTextField);
```

在樣本程式碼中，this.root 會指向包含程式碼的 MovieClip 顯示物件容器；而在實際的程式碼中，您可以指定不同的容器。

您可以使用 addChildAt() 方法，將子系加入顯示物件容器子清單中的特定位置。子清單中這些從零開始的索引位置與顯示物件的圖層 (由前至後順序) 相關。例如，假設有下列三個顯示物件。每一個物件都是透過稱為 Ball 的自訂類別加以建立。

您可以使用 addChildAt() 方法，調整這些顯示物件在容器中的圖層。例如，以下列程式碼為例：

```
ball_A = new Ball(0xFFCC00, "a");
ball_A.name = "ball_A";
ball_A.x = 20;
ball_A.y = 20;
container.addChild(ball_A);

ball_B = new Ball(0xFFCC00, "b");
ball_B.name = "ball_B";
ball_B.x = 70;
ball_B.y = 20;
container.addChild(ball_B);

ball_C = new Ball(0xFFCC00, "c");
ball_C.name = "ball_C";
ball_C.x = 40;
ball_C.y = 60;
container.addChildAt(ball_C, 1);
```

執行這段程式碼之後，顯示物件就會如下所示放置於 container DisplayObjectContainer 物件中 (請注意物件的圖層)：

若要將物件重新放置在顯示清單頂端，只要重新將它加入清單中即可。例如，執行上一個程式碼之後，若要將 ball_A 移至堆疊頂端，請使用下列這行程式碼：

```
container.addChild(ball_A);
```

這段程式碼會有效地自其 container 顯示清單中的位置移除 ball_A，再重新將它加入清單的頂端，最後結果是將它移至堆疊的頂端。

您可以使用 getChildAt() 方法，確認顯示物件的圖層順序。getChildAt() 方法會根據您傳遞給它的索引編號，傳回容器的子物件。例如，下列程式碼會在 container DisplayObjectContainer 物件的子清單中之不同位置上顯示出顯示物件的名稱：

```
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_C
trace(container.getChildAt(2).name); // ball_B
```

如果您從父容器的子清單移除顯示物件，清單上高於此物件的元素會在子索引中各自下移一個位置。例如，接續上一段程式碼的下列程式碼會示範，如果將子清單中下面一個顯示物件移除之後，container DisplayObjectContainer 中在位置 2 的顯示物件如何移至位置 1：

```
container.removeChild(ball_C);
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_B
```

removeChild() 和 removeChildAt() 方法不會完全刪除顯示物件實體，而只是將它們從容器的子清單中移除。此實體仍然可以由另一個變數進行參考 (使用 delete 運算子，可將物件完全移除)。

由於顯示物件只有一個父容器，所以您只能將顯示物件的實體加入一個顯示物件容器。例如，下列程式碼會示範顯示物件 tf1 只能存在於一個容器中 (在此範例中是 Sprite，它可擴充 DisplayObjectContainer 類別)：

```
tf1:TextField = new TextField();
tf2:TextField = new TextField();
tf1.name = "text 1";
tf2.name = "text 2";

container1:Sprite = new Sprite();
container2:Sprite = new Sprite();

container1.addChild(tf1);
container1.addChild(tf2);
container2.addChild(tf1);

trace(container1.numChildren); // 1
trace(container1.getChildAt(0).name); // text 2
trace(container2.numChildren); // 1
trace(container2.getChildAt(0).name); // text 1
```

若將包含於一個顯示物件容器中的顯示物件加入至另一個顯示物件容器中，它會自第一個顯示物件容器的子清單中移除。

除了上述方法外，DisplayObjectContainer 類別也會定義一些方法以配合子顯示物件使用，包括下列方法：

- contains()：判斷顯示物件是否為 DisplayObjectContainer 的子系。
- getChildByName()：依名稱擷取顯示物件。
- getChildIndex()：傳回顯示物件的索引位置。
- setChildIndex()：變更子顯示物件的位置。
- removeChildren()：移除多個子顯示物件。
- swapChildren()：替換兩個顯示物件的由前至後順序。
- swapChildrenAt()：替換由索引值所指定之兩個顯示物件的由前至後順序。

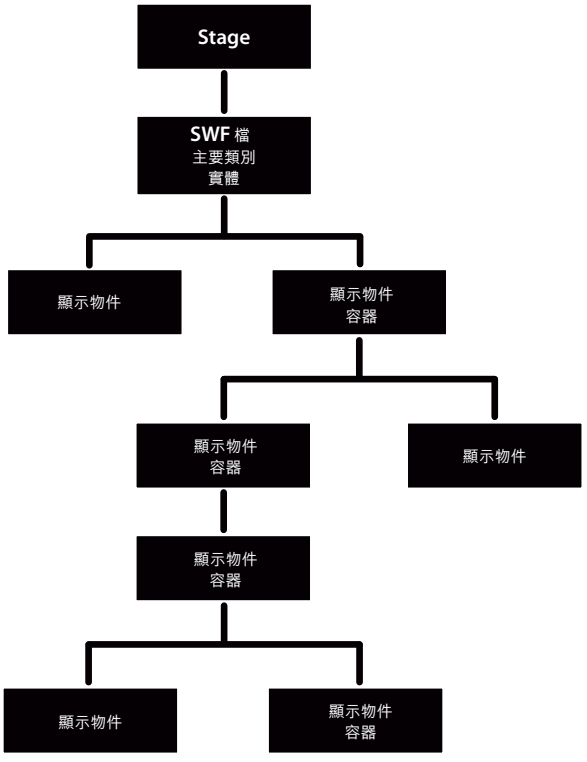
如需詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考的相關項目。

請記住，顯示清單外的顯示物件，也就是不包含於 Stage 子系顯示物件容器中的顯示物件，稱為 「清單外」顯示物件。

在顯示清單中移動

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如您所見，顯示清單是樹狀結構。在樹狀結構頂端的是 Stage，它可以包含多個顯示物件。這些本身是顯示物件容器的顯示物件可以包含其它顯示物件或顯示物件容器。



DisplayObjectContainer 類別包含藉由顯示物件容器之子清單，在顯示清單中移動的屬性和方法。例如，以下列程式碼為例，它會將兩個顯示物件 title 和 pict 加入至 container 物件 (這是 Sprite，而此 Sprite 類別會擴充 DisplayObjectContainer 類別)：

```
var container:Sprite = new Sprite();
var title:TextField = new TextField();
title.text = "Hello";
var pict:Loader = new Loader();
var url:URLRequest = new URLRequest("banana.jpg");
pict.load(url);
pict.name = "banana loader";
container.addChild(title);
container.addChild(pict);
```

getChildAt() 方法會傳回位於指定索引位置之顯示清單的子系：

```
trace(container.getChildAt(0) is TextField); // true
```

您也可以依名稱存取子物件。每一個顯示物件都有名稱屬性，如果您不指定，Flash Player 或 AIR 就會指定預設值，如 "instance1"。例如，下列程式碼會示範如何使用 getChildByName() 方法，存取名稱為 "banana loader" 的子顯示物件：

```
trace(container.getChildByName("banana loader") is Loader); // true
```

使用 getChildByName()，可能會導致執行效能比使用 getChildAt() 方法時緩慢。

由於顯示物件可以在其顯示清單中包含其它顯示物件容器做為子物件，所以您可以將應用程式的完整顯示清單當做樹狀結構在其中移動。例如，在上文所示的程式碼摘錄中，pict Loader 物件的載入作業完成以後，pict 物件將會載入一個點陣圖子顯示物件。若要存取這個點陣圖顯示物件，您可以撰寫 pict.getChildAt(0)，也可以撰寫 container.getChildAt(0).getChildAt(0) (因為 container.getChildAt(0) == pict)。

下列函數會從顯示物件容器提供顯示清單的縮排 trace() 輸出：

```
function traceDisplayList(container:DisplayObjectContainer,indentString:String = ""):void
{
    var child:DisplayObject;
    for (var i:uint=0; i < container.numChildren; i++)
    {
        child = container.getChildAt(i);
        trace(indentString, child, child.name);
        if (container.getChildAt(i) is DisplayObjectContainer)
        {
            traceDisplayList(DisplayObjectContainer(child), indentString + "")
        }
    }
}
```

Adobe Flex

如果您使用 Flex，就應該知道 Flex 可以定義許多組件顯示物件類別，而且這些類別都會覆寫 DisplayObjectContainer 類別的顯示清單存取方法。例如，mx.core 套件的 Container 類別會覆寫 DisplayObjectContainer 類別 (Container 所擴充的類別) 的 addChild() 方法和其它方法。這個類別覆寫 addChild() 方法的方式，會讓您無法將所有類型的顯示物件加入 Flex 中的 Container 實體。如此一來，被覆寫的方法就會要求您要加入的子物件類型必須是 mx.core.UIComponent 物件。

設定 Stage 的屬性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Stage 類別會覆寫 DisplayObject 類別的大部分屬性和方法。若您呼叫其中一個被覆寫的屬性或方法，Flash Player 和 AIR 都會擲回例外。例如，Stage 物件並沒有 x 或 y 屬性，因為它的位置是固定為應用程式的主要容器。x 和 y 屬性是指顯示物件相對於其容器的位置，而由於 Stage 不包含於其它顯示物件容器之中，所以這些屬性不適用。

備註：只有與第一個載入的 SWF 檔不在相同安全執行程序中的顯示物件，才能使用 Stage 類別的一些屬性和方法。如需詳細資訊，請參閱第 918 頁「Stage 安全性」。

控制播放影格速率

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Stage 類別的 frameRate 屬性可設定載入應用程式中所有 SWF 檔的影格速率。如需詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。

控制舞台縮放

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

調整呈現 Flash Player 或 AIR 畫面部分的大小時，執行階段會自動調整「舞台」內容以進行補償。Stage 類別的 scaleMode 屬性會決定 「舞台」內容的調整方式。此屬性可以設定成四種不同的值，這些值在 flash.display.StageScaleMode 類別中定義為常數值：

- StageScaleMode.EXACT_FIT 會縮放 SWF 以符合新舞台尺寸，而不管原始內容外觀比例為何。寬度和高度的縮放因數可能不相同，因此如果舞台的外觀比例已變更，內容看起來可能會受到擠壓或延伸。
- StageScaleMode.SHOW_ALL 會縮放 SWF 以完全符合新舞台尺寸，而不會變更內容外觀比例。此縮放模式會顯示所有的內容，但是可能會造成「上下黑邊」框線，就像是在標準規格電視上觀看寬螢幕影片時出現的黑色邊條。
- StageScaleMode.NO_BORDER 會縮放 SWF 以完全符合新舞台尺寸，而不會變更內容的外觀比例。此縮放模式會充分利用舞台顯示區域，但是可能會造成裁切。
- StageScaleMode.NO_SCALE 則不會縮放 SWF。如果新舞台尺寸較小，會裁切內容；如果較大，新增的空間為空白。

唯有在 StageScaleMode.NO_SCALE 縮放模式下，才可以使用 Stage 類別的 stageWidth 與 stageHeight 屬性，來決定舞台在調整大小後的實際像素尺寸。(在其它縮放模式中，stageWidth 和 stageHeight 屬性永遠反映 SWF 的原始寬度和高度。) 此外，如果將 scaleMode 設定為 StageScaleMode.NO_SCALE，再調整 SWF 檔的大小，則還會傳送 Stage 類別的 resize 事件，讓您可以進行相應的調整。

因此，將 scaleMode 設定為 StageScaleMode.NO_SCALE，讓您更能隨心所欲地控制螢幕內容如何依據視窗的大小調整進行調整。例如，在包含視訊和控制列的 SWF 中，您可能會想讓控制列在調整 「舞台」大小時保持同樣的大小，而只要變更視訊的大小來配合 「舞台」大小的改變。這會在下列範例中示範：

```
// mainContent is a display object containing the main content;
// it is positioned at the top-left corner of the Stage, and
// it should resize when the SWF resizes.

// controlBar is a display object (e.g. a Sprite) containing several
// buttons; it should stay positioned at the bottom-left corner of the
// Stage (below mainContent) and it should not resize when the SWF
// resizes.

import flash.display.Stage;
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.events.Event;

var swfStage:Stage = mainContent.stage;
swfStage.scaleMode = StageScaleMode.NO_SCALE;
swfStage.align = StageAlign.TOP_LEFT;
swfStage.addEventListener(Event.RESIZE, resizeDisplay);

function resizeDisplay(event:Event):void
{
    var swfWidth:int = swfStage.stageWidth;
    var swfHeight:int = swfStage.stageHeight;

    // Resize the main content area
    var newContentHeight:Number = swfHeight - controlBar.height;
    mainContent.height = newContentHeight;
    mainContent.scaleX = mainContent.scaleY;

    // Reposition the control bar.
    controlBar.y = newContentHeight;
}
```

設定 **AIR** 視窗的舞台縮放模式

舞台 scaleMode 屬性會決定調整視窗大小後，舞台縮放和剪輯子顯示物件的方式。AIR 應該只使用 noScale 模式。在這個模式下，舞台不會縮放。反而，舞台的大小會隨著視窗的邊緣而直接變更。若視窗變小，可能會剪輯物件。

舞台縮放模式是針對網路瀏覽器之類的環境所設計，因為在這種環境中您有時無法控制舞台的大小或比例。當舞台不符合應用程式的真實大小或者長寬比，這些模式可以讓您選擇最接近需要的大小或比例。在 AIR 中，您永遠都擁有舞台的控制權，因此在大部分情況下，重新配置您的內容或調整視窗的尺寸，將會比啟用舞台縮放得到更佳的結果。

在瀏覽器以及初始 AIR 視窗中，視窗大小與初始縮放系數之間的關係，會從載入的 SWF 檔案讀取。不過，當您建立 NativeWindow 物件，AIR 會在視窗大小以及縮放系數 72:1 之間任意選擇一種關係。因此，若您的視窗為 72x72 個像素，則新增至視窗的 10x10 矩形所繪製的大小就是 10x10 個像素。不過，若視窗為 144x144 個像素，則 10x10 個像素矩形會被縮放成 20x20 個像素。若您堅持視窗舞台使用 scaleMode 而不用 noScale，則可以將視窗中任何顯示物件的縮放係數設定為 72 個像素 (與舞台目前的寬度和高度比例)。例如，以下程式碼會為 client 顯示物件計算必要的縮放係數：

```
if(newWindow.stage.scaleMode != StageScaleMode.NO_SCALE){
client.scaleX = 72/newWindow.stage.stageWidth;
client.scaleY = 72/newWindow.stage.stageHeight;
}
```

備註：Flex 和 HTML 視窗會自動將舞台 scaleMode 設定成 noScale。變更 scaleMode 會干擾這些視窗類型所使用的版面機制。

使用全螢幕模式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

全螢幕模式可讓您將影片的舞台設定為符合檢視者的整個螢幕，而不含任何容器邊框或選單。Stage 類別的 displayState 屬性是用來為 SWF 切換開啟及關閉全螢幕模式。displayState 屬性可以設定為由 flash.display.StageDisplayState 類別中常數所定義的其中一個值。若要開啟全螢幕模式，請將 displayState 屬性設定為 StageDisplayState.FULL_SCREEN：

```
stage.displayState = StageDisplayState.FULL_SCREEN;
```

若要開啟全螢幕互動模式 (Flash Player 11.3 的新功能)，請將 displayState 屬性設定為 StageDisplayState.FULL_SCREEN_INTERACTIVE：

```
stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
```

在 Flash Player 中，全螢幕模式只能透過 ActionScript 回應按一下滑鼠動作 (包括按一下右鍵) 或按下按鍵，才能進行初始化。對於在應用程式安全執行程序中執行的 AIR 內容，則不會要求必須進入全螢幕模式才能回應使用者手勢。

若要離開全螢幕模式，請將 displayState 屬性設定為 StageDisplayState.NORMAL。

```
stage.displayState = StageDisplayState.NORMAL;
```

此外，使用者可以將焦點切換至不同的視窗，或使用一些按鍵組合的其中一種，選擇離開全螢幕模式，可使用的按鍵組合包括：Esc 鍵 (所有平台)、Ctrl+W (Windows)、Command+W (Mac) 或 Alt+F4 (Windows)。

在 **Flash Player** 中啟用全螢幕模式

若要為嵌入 HTML 網頁中的 SWF 檔啟用全螢幕模式，嵌入 Flash Player 的 HTML 程式碼必須包括 param 標籤以及名稱為 allowFullScreen 和值為 true 的 embed 特質，如下所示：

```
<object>
    ...
    <param name="allowFullScreen" value="true" />
    <embed ... allowFullScreen="true" />
</object>
```

在 Flash 編寫工具中選取 「檔案 > 發佈設定」，然後在 「發佈設定」對話方塊的 「HTML」索引標籤上，選取 「僅適用於 Flash - 允許全螢幕」範本。

在 Flex 中，確認 HTML 範本包含支援全螢幕的 <object> 和 <embed> 標籤。

若在網頁中使用 JavaScript，以產生 SWF 嵌入標籤，則必須更改 JavaScript，在其中加入 allowFullScreen param 標籤和特質。例如，若您的 HTML 頁面使用 AC_FL_RunContent() 函數 (用於 Flash Professional 和 Flash Builder 所產生的 HTML 檔)，應將 allowFullScreen 參數加入該函數，如下所示：

```
AC_FL_RunContent(
    ...
    'allowFullScreen','true',
    ...
    ); //end AC code
```

這並不適用於在獨立的 Flash Player 中執行的 SWF 檔。

備註：無論您將「視窗模式」(HTML 中的 wmode) 設定為「不透明無視窗」(opaque) 或「透明無視窗」(transparent)，全螢幕視窗一定是不透明的。

在瀏覽器中，Flash Player 若要使用全螢幕模式，也有與安全性相關的限制。這些限制會在第 900 頁「安全性」中說明。

啟用 **Flash Player 11.3** 及更新版本中的全螢幕互動模式

Flash Player 11.3 及更新版本支援全螢幕互動模式，可對所有鍵盤按鍵提供完整支援 (**Esc** 除外，這個按鍵會結束全螢幕互動模式)。全螢幕互動模式適合在進行遊戲時使用 (例如，可以與多人遊戲中的玩家聊天，或者在第一人的射擊遊戲中啟用 WASD 鍵盤控制項)。

若要為內嵌於 HTML 頁面中的 SWF 檔案啟用全螢幕互動模式，要嵌入 Flash Player 的 HTML 程式碼必須包括 param 標籤以及名稱為 allowFullScreenInteractive 和值為 true 的 embed 特質，如下所示：

```
<object>
    ...
    <param name="allowFullScreenInteractive" value="true" />
    <embed ... allowFullScreenInteractive="true" />
</object>
```

在 Flash 編寫工具中選取「檔案 > 發佈設定」，然後在「發佈設定」對話方塊的「HTML」索引標籤上，選取「僅適用於 Flash - 允許全螢幕」範本。

在 Flash Builder 和 Flex 中，確定 HTML 範本包含支援全螢幕互動模式的 <object> 和 <embed> 標籤。

若要在網頁中使用 JavaScript，以產生 SWF 內嵌標籤，您必須更改 JavaScript，在其中加入 allowFullScreenInteractive param 標籤和特質。例如，若您的 HTML 頁面使用 AC_FL_RunContent() 函數 (用於 Flash Professional 和 Flash Builder 所產生的 HTML 頁面)，應將 allowFullScreenInteractive 參數加入該函數呼叫，如下所示：

```
AC_FL_RunContent(
    ...
    'allowFullScreenInteractive','true',
    ...
    ); //end AC code
```

這並不適用於在獨立的 Flash Player 中執行的 SWF 檔。

全螢幕舞台大小與縮放

Stage.fullScreenHeight 和 Stage.fullScreenWidth 屬性會傳回進入全螢幕大小時 (在立即進入的情況下)，顯示器的高度和寬度。當您擷取這些值之後，如果使用者在進入全螢幕模式之前，將瀏覽器從一個顯示器移至其它顯示器，這些值就可能會不正確。如果您從 Stage.displayState 屬性設定為 StageDisplayState.FULL_SCREEN 的事件處理常式中擷取這些值，這些值就是正確的。若使用者有多個顯示器，SWF 內容只會擴展填滿一個顯示器畫面。Flash Player 和 AIR 會量測並判斷包含最大部分 SWF 的顯示器，然後使用該顯示器執行全螢幕模式。fullScreenHeight 和 fullScreenWidth 屬性只會反映用於全螢幕模式的監視器大小。如需詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中的 Stage.fullScreenHeight 與 Stage.fullScreenWidth。

全螢幕模式的舞台縮放行為與在一般模式下相同：縮放是由 Stage 類別的 scaleMode 屬性控制。如果將 scaleMode 屬性設定為 StageScaleMode.NO_SCALE，Stage 的 stageWidth 和 stageHeight 屬性會變更，以反映由 SWF 所佔據之螢幕區域大小 (在此例中為整個螢幕)。如果是在瀏覽器中檢視，則 HTML 參數會控制這項設定。

您可以使用 Stage 類別的 fullScreen 事件，在開啟或關閉全螢幕模式時，進行偵測及回應。例如，您可能要在進入或離開全螢幕模式時，重新定位、加入或從螢幕移除項目，如下列範例所示：

```
import flash.events.FullScreenEvent;

function fullScreenRedraw(event:FullScreenEvent):void
{
    if (event.fullScreen)
    {
        // Remove input text fields.
        // Add a button that closes full-screen mode.
    }
    else
    {
        // Re-add input text fields.
        // Remove the button that closes full-screen mode.
    }
}

mySprite.stage.addEventListener(FullScreenEvent.FULL_SCREEN, fullScreenRedraw);
```

如這段程式碼所示，fullScreen 事件的事件物件是 flash.events.FullScreenEvent 類別的實體，其中包含 fullScreen 屬性，指出已啟用全螢幕模式 (true) 或未啟用 (false)。

全螢幕模式中的鍵盤支援

當 Flash Player 在瀏覽器中執行時，所有與鍵盤相關的 ActionScript (如鍵盤事件和 TextField 實體中的輸入文字)，在全螢幕模式下都會遭到停用，例外情況 (啟用的按鍵) 如下：

- 一些非列印按鈕，也就是方向鍵、空格鍵和 Tab 鍵
- 終止全螢幕模式的鍵盤快速鍵是：Esc (Windows 和 Mac)、Control+W (Windows)、Command+W (Mac) 以及 Alt+F4

這些限制「不適用」於在獨立 Flash Player 或 AIR 中執行的 SWF 內容。AIR 支援互動式全螢幕模式，允許鍵盤輸入。

全螢幕模式中的滑鼠支援

根據預設，全螢幕模式中的滑鼠事件運作方式與非全螢幕模式相同。然而，在全螢幕模式中，您可以選擇性地設定 Stage.mouseLock 屬性以啟用滑鼠鎖定。滑鼠鎖定會停用游標並啟用未繫結的滑鼠移動方式。

備註：您只能在桌面應用程式的全螢幕模式中啟用滑鼠鎖定。若未在應用程式的全螢幕模式設定滑鼠鎖定，或是針對行動裝置上的應用程式設定滑鼠鎖定，都將擲回例外狀況。

出現下列情況時會自動停用滑鼠鎖定，並且再次顯示滑鼠游標：

- 使用者使用 Escape 鍵 (所有平台)、Control-W (Windows)、Command-W (Mac) 或 Alt-F4 (Windows) 結束全螢幕模式。
- 應用程式視窗失去焦點。
- 任何設定 UI 均為可見，包括所有隱私權對話方塊。
- 出現原生對話方塊，例如檔案上傳對話方塊。

與滑鼠移動相關聯的事件 (例如 mouseMove 事件) 使用 MouseEvent 類別表示事件物件。當滑鼠鎖定停用時，請使用 MouseEvent.localX 和 MouseEvent.localY 屬性判斷滑鼠的位置。當滑鼠鎖定啟用時，請使用 MouseEvent.movementX 和 MouseEvent.movementY 屬性判斷滑鼠的位置。movementX 和 movementY 屬性包含自上次事件以來滑鼠位置的變更，而非滑鼠位置的絕對座標。

全螢幕模式中的硬體縮放

在 Flash Player 或 AIR 中，您可以使用 Stage 類別的 fullScreenSourceRect 屬性，將舞台的特定區域設定縮放為全螢幕模式。Flash Player 和 AIR 會使用硬體縮放 (如果可供使用的話)，利用使用者電腦上的顯示卡和視訊卡，而且一般來說，透過硬體縮放顯示內容的速度通常會比軟體縮放快。

若要利用硬體縮放，請將整個舞台或舞台的一部分設定為全螢幕模式。下列 ActionScript 3.0 程式碼將整個舞台設定為全螢幕模式：

```
import flash.geom.*;
{
    stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

當這個屬性設定為有效的矩形，而且 displayState 屬性設定為全螢幕模式時，Flash Player 和 AIR 會縮放指定的區域。ActionScript 中實際的「舞台」大小 (以像素為單位) 不會改變。Flash Player 和 AIR 會強制最小限制，使矩形的大小放得下標準的「按 Esc 離開全螢幕模式」訊息。這個限制通常大約是 260 x 30 像素，但可能因平台和 Flash Player 版本的不同而異。

只有當 Flash Player 或 AIR 不是處於全螢幕模式時，才能設定 fullScreenSourceRect 屬性。若要正確使用這個屬性，請先設定這個屬性，然後再將 displayState 屬性設定為全螢幕模式。

若要啟用縮放，請將 fullScreenSourceRect 屬性設定為矩形物件。

```
stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
```

若要停用縮放，請將 fullScreenSourceRect 屬性設定為 null。

```
stage.fullScreenSourceRect = null;
```

若要在 Flash Player 中利用所有硬體加速功能，請透過 「Flash Player 設定」對話方塊來啟用這些功能。若要載入此對話方塊，請在瀏覽器中的 Flash Player 內容之中按一下滑鼠右鍵 (Windows) 或 Control+ 按一下 (Mac)。選取「顯示」索引標籤 (第一個索引標籤)，然後按一下 「啟用硬體加速」核取方塊。

直接與 **GPU** 複合視窗模式

Flash Player 10 引進兩種新的視窗模式：直接模式及 GPU 複合模式，您可以透過 Flash 編寫工具中的發佈設定加以啟用。AIR 不支援這些模式。若要利用這些模式的優點，您必須為 Flash Player 啟用硬體加速。

直接模式會以最快、最直接的路徑將圖像送上螢幕，非常適合用來播放視訊。

GPU 複合模式會使用視訊卡上的圖像處理單元來加速複合作業。視訊複合是一項排列多個影像圖層以建立單一視訊影像的程序。使用 GPU 加速複合作業時，可以改善 YUV 轉換、色彩校正、旋轉或縮放以及混合時的效能。YUV 轉換就是從複合類比訊號 (用於傳輸) 轉換成 RGB (紅色、綠色、藍色) 顏色模型 (視訊攝影機和顯示器所使用) 的色彩轉換。使用 GPU 加速複合作業時，可以減少記憶體的使用量和 CPU 的運算需求。此外，也可以讓具有標準定義的視訊在播放時更為順暢。

實作這些視窗模式時，請務必謹慎。使用 GPU 複合時，也可能會耗用許多記憶體和 CPU 資源。如果 GPU 無法處理某些作業 (例如混合模式、濾鏡、裁剪或遮色)，就必須靠軟體來完成。使用這些模式時，Adobe 建議您每一個 HTML 網頁只播放一個 SWF 檔，並且不要為橫幅啟用這些模式。Flash 的 「測試影片」功能不會使用硬體加速，但是您可以透過 「發佈預覽」選項來使用。

在 SWF 檔中設定 60 (最高螢幕重新整理速率) 以上的影格速率並沒有用。將影格速率設定為介於 50 和 55 之間的值可以容許丟棄影格的情形，這種情形有時會因為各種原因而發生。

若要使用直接模式，電腦必須配備 Microsoft DirectX 9 和 VRAM 128 MB (Windows) 以及 OpenGL (Apple Macintosh、Mac OS X v10.2 或更新版本)。若要使用 GPU 複合模式，電腦必須具備 Microsoft DirectX 9 和 Pixel Shader 2.0 支援 (Windows) 以及 VRAM 128 MB。對於 Mac OS X 和 Linux 而言，若要使用 GPU 複合模式，則必須具備 OpenGL 1.5 和數個 OpenGL 擴充功能 (framebuffer 物件、多重紋理、著色器物件、陰影語言、片段著色器)。

您可以透過「Flash 發佈設定」對話方塊，利用「Flash」索引標籤上的「硬體加速」選單，為每個 SWF 檔啟動 direct 和 gpu 加速模式。如果您選擇 「無」，則如 「HTML」索引標籤上的 「視窗模式」設定所指定，視窗模式會回復至 default、transparent 或 opaque。

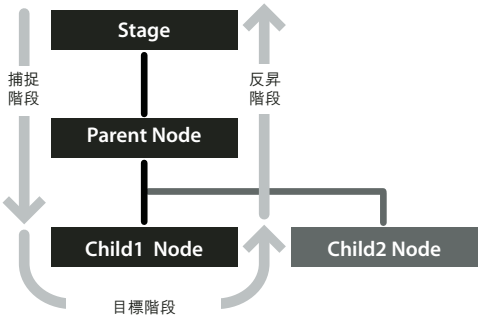
處理顯示物件的事件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

DisplayObject 類別是繼承自 EventDispatcher 類別，也就是說，每一個顯示物件都可以完全參與事件模型 (請參閱第 108 頁「處理事件」)。每一個顯示物件都可以使用其 addEventListener() 方法 (繼承自 EventDispatcher 類別) 以偵聽特定事件，但只有在偵聽物件屬於該事件的事件流程時才能使用。

當 Flash Player 或 AIR 傳送事件物件時，該事件物件會從 Stage 傳送至事件發生的顯示物件，然後再回到 Stage。例如，如果使用者在名為 child1 的顯示物件上按一下，Flash Player 會從 Stage 傳送事件物件，一直沿著顯示清單階層架構往下傳送到 child1 顯示物件。

事件流程在概念上可分成三個階段，如下列圖例所示：



如需詳細資訊，請參閱第 108 頁 「處理事件」。

使用顯示物件事件時心中要謹記一個重點，就是當顯示物件從顯示清單移除時，事件偵聽程式對是否自動從記憶體中移除顯示物件 (遭到回收) 所能產生的影響。若顯示物件含有訂閱為其事件之偵聽程式的物件，則即使自顯示清單中移除該顯示物件，也不會自記憶體中移除，因為它仍將具有這些偵聽程式物件的參考。如需詳細資訊，請參閱第 118 頁 「管理事件偵聽程式」。

選擇 DisplayObject 子類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用顯示物件時，您有一些選擇，其中一項重要決定就是：使用哪個物件做何種用途。這裡有一些原則可以幫助您做決定。不管您是需要類別的實體，或是為您要建立的類別選擇基底類別，這些建議同樣也適用：

- 若不需要做為其它顯示物件之容器的物件 (也就是，您只需要做為獨立螢幕元素的物件)，請視其用途，選擇下列 DisplayObject 或 InteractiveObject 子類別的其中一個：
  - 用來顯示點陣圖影像的 Bitmap。
  - 用來加入文字的 TextField。
  - 用來顯示視訊的 Video。
  - 用來在螢幕上繪製內容之 「畫布」的 Shape。尤其是，若要在螢幕上建立所繪製形狀的實體，而且它不會做為其它顯示物件的容器，那麼使用 Shape 而不使用 Sprite 或 MovieClip 將能讓您取得相當大的效能效益。
  - 以 Flash 編寫工具建立之項目所需的 MorphShape、StaticText 或 SimpleButton (您不能以程式設計方式建立這些類別的實體，但是可以用這些資料類型建立變數，參考使用 Flash 編寫工具所建立的項目)。
- 若需要變數以參考主要的 Stage，請使用 Stage 類別做為其資料類型。

- 若您需要容器來載入外部 SWF 檔或影像檔，請使用 Loader 實體。已載入的內容將加入顯示清單中，做為 Loader 實體的子系，其資料類型將視所載入內容的性質而定，如下所示：
  - 載入的影像將會是 Bitmap 實體。
  - 在 ActionScript 3.0 中撰寫之被載入端 SWF 檔將會是 Sprite 或 MovieClip 實體 (或這些類別的子類別實體，由內容建立者指定)。
  - 在 ActionScript 1.0 或 ActionScript 2.0 中撰寫之被載入端 SWF 檔將會是 AVM1Movie 實體。
- 若需要物件做為其它顯示物件的容器 (不論您是否也需要使用 ActionScript 繪製到顯示物件上)，請選擇 DisplayObjectContainer 子類別的其中一個：
  - 若物件將只使用 ActionScript 建立，或是做為將僅用 ActionScript 建立及操作之自訂顯示物件的基底類別，請選擇 Sprite。
  - 若要建立變數以參考用 Flash 編寫工具建立的影片片段元件，請選擇 MovieClip。
- 若要建立將與 Flash 元件庫中的影片片段元件相關聯的類別，請選擇下列其中一個 DisplayObjectContainer 子類別做為該類別的基底類別：
  - 若相關聯的影片片段元件具有一個以上的影格，請選擇 MovieClip
  - 若相關聯的影片片段元件只在第一個影格中含有內容，請選擇 Sprite

操作顯示物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

不管您選擇使用何種顯示物件，所有顯示物件做為螢幕上顯示的元素都會有一些共同的操作功能。例如，它們都可以放在螢幕上，在顯示物件的堆疊順序中向前或向後移動、縮放、旋轉，諸如此類的動作。因為所有顯示物件都自其共用的基底類別 (DisplayObject) 繼承這項功能，這項功能不管您是在操作 TextField 實體、Video 實體、Shape 實體，或任何其它顯示物件，都會表現出相同的行為。下列各節將詳細探討其中一些共通的顯示物件操作。

變更位置

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

任何顯示物件的最基本操作就是放置在螢幕上。若要設定顯示物件的位置，請變更物件的 x 和 y 屬性。

```
myShape.x = 17;
myShape.y = 212;
```

顯示物件放置系統是以迪卡兒座標系統 (有水平 x 軸和垂直 y 軸的常用網格系統) 處理 「舞台」。座標系統的原點 (0,0 座標，代表 x 和 y 軸相交處) 是在「舞台」的左上角。從此處，x 值是向右為正值而向左為負值，而 (與一般製圖系統相反) y 值是向下為正值而向上為負值。例如，上面的幾行程式碼會將物件 myShape 移至 x 座標 17 (原點向右 17 個像素) 而 y 座標 212 (原點向下 212 個像素)。

根據預設，當顯示物件是使用 ActionScript 建立時，x 和 y 屬性同時都設定為 0，而將物件放在其父內容的左上角。

變更相對於舞台的位置

請務必記得 x 和 y 屬性永遠都指向相對於其父顯示物件軸之 0,0 座標的顯示物件位置，因此，對包含於 Sprite 實體內的 Shape 實體 (如圓形) 而言，將 Shape 物件的 x 和 y 屬性設定為 0 會將該圓形放置於 Sprite 的左上角，而不一定是 「舞台」的左上角。若要將物件依相對於全域 「舞台」座標放置，則可以使用任何顯示物件的 globalToLocal() 方法，將座標從全域 (「舞台」) 座標轉換成區域 (顯示物件容器) 座標，如下所示：

```
// Position the shape at the top-left corner of the Stage,
// regardless of where its parent is located.

// Create a Sprite, positioned at x:200 and y:200.
var mySprite:Sprite = new Sprite();
mySprite.x = 200;
mySprite.y = 200;
this.addChild(mySprite);

// Draw a dot at the Sprite's 0,0 coordinate, for reference.
mySprite.graphics.lineStyle(1, 0x000000);
mySprite.graphics.beginFill(0x000000);
mySprite.graphics.moveTo(0, 0);
mySprite.graphics.lineTo(1, 0);
mySprite.graphics.lineTo(1, 1);
mySprite.graphics.lineTo(0, 1);
mySprite.graphics.endFill();

// Create the circle Shape instance.
var circle:Shape = new Shape();
mySprite.addChild(circle);

// Draw a circle with radius 50 and center point at x:50, y:50 in the Shape.
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0xff0000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();

// Move the Shape so its top-left corner is at the Stage's 0, 0 coordinate.
var stagePoint:Point = new Point(0, 0);
var targetPoint:Point = mySprite.globalToLocal(stagePoint);
circle.x = targetPoint.x;
circle.y = targetPoint.y;
```

同樣地，您也可以使用 DisplayObject 類別的 localToGlobal() 方法，將區域座標轉換成 「舞台」座標。

使用滑鼠移動顯示物件

您可以讓使用者利用 ActionScript 中的兩種不同技術，以滑鼠移動顯示物件。在這兩種方式中，都會使用兩個滑鼠事件：按下滑鼠按鍵時，即告知物件跟隨滑鼠游標，以及放開滑鼠按鍵時，即告知物件停止跟隨滑鼠游標。

備註：Flash Player 11.3 及更新版本、AIR 3.3 及更新版本：您也可以使用 MouseEvent.RELEASE_OUTSIDE 事件涵蓋使用者在包含的 Sprite 邊界外放開滑鼠按鈕的情形。

第一種技術是使用 startDrag() 方法，比較簡單，但是也具有較多限制。按下滑鼠按鍵時，即呼叫所要拖曳之顯示物件的 startDrag() 方法；放開滑鼠按鍵時，則呼叫 stopDrag() 方法。Sprite 類別定義這兩個函數，這樣移動的物件必須是 Sprite 或其子類別之一。

```
// This code creates a mouse drag interaction using the startDrag()
// technique.
// square is a MovieClip or Sprite instance).

import flash.events.MouseEvent;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    square.startDrag();
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    square.stopDrag();
}

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

這項技巧有一項相當大的限制，就是一次只能使用 startDrag() 拖曳一個項目。如果正在拖曳一個顯示物件，而在另一個顯示物件上呼叫 startDrag() 方法，第一個顯示物件就會立即停止跟隨滑鼠移動。例如，如果變更 startDragging() 函數 (如下所示)，則將只拖曳 circle 物件，而不管 square.startDrag() 方法呼叫：

```
function startDragging(event:MouseEvent):void
{
    square.startDrag();
    circle.startDrag();
}
```

由於在使用 startDrag() 時只能拖曳一個物件，結果針對任何顯示物件隨時都可能呼叫 stopDrag() 方法，而導致停止目前正在拖曳的任何物件。

若需要拖曳一個以上的顯示物件，或是要避免可能會有一個以上的物件使用 startDrag() 所可能產生的衝突，最好是使用滑鼠跟隨技巧來建立拖曳作用。使用這項技巧並按下滑鼠按鍵，即會將函數訂閱為「舞台」之 mouseMove 事件的偵聽程式，然後再於每次滑鼠移動時呼叫這個函數，讓拖曳的物件跳到滑鼠的 x,y 座標；放開滑鼠按鍵之後，即會將函數取消訂閱為偵聽程式，表示它不再於滑鼠移動時進行呼叫，而且物件會停止跟隨滑鼠游標。接下來這段程式碼會示範這項技巧。

```
// This code moves display objects using the mouse-following
// technique.
// circle is a DisplayObject (e.g. a MovieClip or Sprite instance).

import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // Record the difference (offset) between where
    // the cursor was when the mouse button was pressed and the x, y
    // coordinate of the circle when the mouse button was pressed.
    offsetX = event.stageX - circle.x;
    offsetY = event.stageY - circle.y;

    // tell Flash Player to start listening for the mouseMove event
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called when the mouse button is released.
```

```
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragCircle(event:MouseEvent):void
{
    // Move the circle to the location of the cursor, maintaining
    // the offset between the cursor's location and the
    // location of the dragged object.
    circle.x = event.stageX - offsetX;
    circle.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

除了讓顯示物件跟著滑鼠游標移動外，最好是將拖曳的物件移至畫面前面，讓它浮動出現在所有其它物件之上。例如，假設您有兩個物件，一個是圓形，另一個是方形，兩者都可以使用滑鼠移動它們。若在顯示清單上，圓形正好在正方形之下，而且您按一下並拖曳圓形，要讓游標在正方形之上，圓形就會顯得好像是滑到正方形之後，打斷了拖放的感覺。其實您可以做成按一下圓形時，它會移到顯示清單頂端，然後永遠顯示在任何其它內容之上。

以下的程式碼 (從前面的範例中改編) 允許使用滑鼠移動兩個顯示物件，也就是圓形和矩形。只要滑鼠按鍵在其中一個物件之上按一下，該項目就會移到「舞台」的顯示清單頂端，讓拖曳的項目永遠都顯示在最上面 (新的程式碼或與以前列出的程式碼不同時會以粗體顯示)。

```
// This code creates a drag-and-drop interaction using the mouse-following
// technique.
// circle and square are DisplayObjects (e.g. MovieClip or Sprite
// instances).

import flash.display.DisplayObject;
import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;
var draggedObject:DisplayObject;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // remember which object is being dragged
    draggedObject = DisplayObject(event.target);

    // Record the difference (offset) between where the cursor was when
    // the mouse button was pressed and the x, y coordinate of the
    // dragged object when the mouse button was pressed.
    offsetX = event.stageX - draggedObject.x;
    offsetY = event.stageY - draggedObject.y;

    // move the selected object to the top of the display list
    stage.addChild(draggedObject);

    // Tell Flash Player to start listening for the mouseMove event.
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}
```

```
// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragObject(event:MouseEvent):void
{
    // Move the dragged object to the location of the cursor, maintaining
    // the offset between the cursor's location and the location
    // of the dragged object.
    draggedObject.x = event.stageX - offsetX;
    draggedObject.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

若要更進一步擴充這種特效，例如在遊戲中移動數堆代幣或卡片，您可以在「撿起」時，將拖曳的物件加入「舞台」的顯示清單中，然後在放開滑鼠時，將它加入另一份顯示清單 (例如位於丟下位置的代幣或卡片堆) 中。

最後，若要加強特效，您可以在按一下 (開始拖曳) 時，將「投影」濾鏡套用至顯示物件，然後在放開物件時，移除投影。如需有關在 ActionScript 中使用投影濾鏡及其它顯示物件濾鏡的詳細資訊，請參閱第 226 頁「以濾鏡處理顯示物件」。

左右移動和捲動顯示物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若有對所要顯示的區域來說太大的顯示物件，您可以使用 scrollRect 屬性，定義顯示物件的可見區域。此外，透過變更 scrollRect 屬性以回應使用者輸入，您可以讓內容向左右移動或向上下捲動。

scrollRect 屬性是 Rectangle 類別的實體，是組合定義矩形區域為單一物件所需值的類別。一開始若要定義顯示物件的可見區域，請建立新的 Rectangle 實體，並指定給顯示物件的 scrollRect 屬性。接著，若要上下捲動或左右移動，可將 scrollRect 屬性讀入另外一個 Rectangle 變數中，再變更所需的屬性 (例如，變更 Rectangle 實體的 x 屬性以左右移動，或變更 y 屬性以上下捲動)，然後重新指定 Rectangle 實體至 scrollRect 屬性，以通知已變更值的顯示物件。

例如，下列程式碼會定義名為 bigText 之 TextField 物件的可見區域，此物件太大而無法置入 SWF 檔範圍之中。當按一下名為 up 和 down 的兩個按鈕時，就會透過修改 scrollRect Rectangle 實體的 y 屬性，呼叫讓 TextField 物件的內容上下捲動的函數。

```
import flash.events.MouseEvent;
import flash.geom.Rectangle;

// Define the initial viewable area of the TextField instance:
// left: 0, top: 0, width: TextField's width, height: 350 pixels.
bigText.scrollRect = new Rectangle(0, 0, bigText.width, 350);

// Cache the TextField as a bitmap to improve performance.
bigText.cacheAsBitmap = true;

// called when the "up" button is clicked
function scrollUp(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Decrease the y value of the rectangle by 20, effectively
    // shifting the rectangle down by 20 pixels.
    rect.y -= 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

// called when the "down" button is clicked
function scrollDown(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Increase the y value of the rectangle by 20, effectively
    // shifting the rectangle up by 20 pixels.
    rect.y += 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

up.addEventListener(MouseEvent.CLICK, scrollUp);
down.addEventListener(MouseEvent.CLICK, scrollDown);
```

如上述範例所示，當您使用顯示物件的 scrollRect 屬性時，最好是使用 cacheAsBitmap 屬性，指定 Flash Player 或 AIR 應該將顯示物件的內容快取為點陣圖。如此一來，Flash Player 和 AIR 就不必在每次捲動時重新繪製顯示物件的全部內容，而且能夠使用快取的點陣圖，直接在螢幕上呈現必要的部分。如需詳細資訊，請參閱第 154 頁 「快取顯示物件」。

調整大小及縮放物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以用兩種方式量測及調整顯示物件的大小：使用維度屬性 (width 和 height)，或縮放屬性 (scaleX 和 scaleY)。

每一個顯示物件都有 width 屬性和 height 屬性，起初都以像素為單位設定為物件的大小。您可以讀取這些屬性的值，以量測顯示物件的大小。您也可以指定新值，以變更物件的大小，如下所示：

```
// Resize a display object.
square.width = 420;
square.height = 420;

// Determine the radius of a circle display object.
var radius:Number = circle.width / 2;
```

變更顯示物件的 height 或 width 會導致縮放物件的結果，表示其內容會擴展或壓縮，以配合新區域的大小。若顯示物件只包含向量形狀，這些形狀將會以新比例重新繪製，而保持原有的品質。顯示物件中任何點陣圖圖像元素都將進行縮放而不是重新繪製。例如，寬度和高度加大到超過影像中像素資訊的實際維度時，數位相片會像素化而呈現鋸齒狀。

當您變更顯示物件的 width 或 height 屬性時，Flash Player 和 AIR 也會更新該物件的 scaleX 和 scaleY 屬性。

備註： TextField 物件對於這個縮放行為指令而言都屬於例外。文字欄位必須能自行調整大小來符合文字換行和字體大小，而在重新調整大小之後，文字欄位會將本身的 scaleX 或 scaleY 值重設為 1。不過，如果您調整 TextField 物件的 scaleX 或 scaleY 值，寬度和高度值也會隨之變更，以符合所提供的縮放值。

這些屬性代表顯示物件與其原始大小比較下的相對大小。scaleX 和 scaleY 屬性會使用分數 (小數) 值代表百分比。例如，若變更了顯示物件的 width，而使寬度成為原始大小的一半，則物件的 scaleX 屬性的值將會是 .5，表示 50%；若其高度加大成兩倍，則其 scaleY 屬性的值將會是 2，表示 200%。

```
// circle is a display object whose width and height are 150 pixels.
// At original size, scaleX and scaleY are 1 (100%).
trace(circle.scaleX); // output: 1
trace(circle.scaleY); // output: 1

// When you change the width and height properties,
// Flash Player changes the scaleX and scaleY properties accordingly.
circle.width = 100;
circle.height = 75;
trace(circle.scaleX); // output: 0.6622516556291391
trace(circle.scaleY); // output: 0.4966887417218543
```

大小並不是依比例變更，換句話說，若變更正方形的 height，但未變更其 width，其比例將不會再相同，正方形將變成矩形。若要對顯示物件的大小進行相對性變更，則可以設定 scaleX 和 scaleY 屬性的值來調整物件的大小，做為設定 width 或 height 屬性的另一種替代方式。例如，這段程式碼會變更名為 square 之顯示物件的 width，然後改變垂直縮放比例 (scaleY) 以配合水平縮放比例，讓正方形大小維持原來比例。

```
// Change the width directly.
square.width = 150;

// Change the vertical scale to match the horizontal scale,
// to keep the size proportional.
square.scaleY = square.scaleX;
```

縮放時控制扭曲情形

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

一般來說，縮放顯示物件時 (例如，以水平方向延伸)，所產生的扭曲是跨整個物件平均散佈，因此每一個部分的延伸量都相同。對於圖像和設計元素，這大概是您所要的結果，但是有時會寧可控制顯示物件中要延伸的部位，以及要保持不變的部位。這種情形的其中一個常見範例是圓角矩形按鈕。在一般縮放下，按鈕的四角會延伸，使得四角半徑依按鈕調整大小而變更。

但是，在本範例中，卻最好能控制縮放 (能夠指定某些區域 (直邊和中央) 要縮放，以及不應該縮放的區域 (四角))，以便在沒有可見的扭曲下進行縮放。

您可以使用 9 分割縮放 (9 倍縮放) 來建立顯示物件，就能夠控制物件縮放的方式。在 9 分割縮放下，顯示物件是分成 9 個不同的矩形 (即 3 x 3 的網格，就像井字遊戲棋盤的網格)。這些矩形並不一定要有相同大小，由您指定網格線所在位置，位於四角矩形中的任何內容 (如按鈕的圓角) 將不會在縮放顯示物件時擴展或壓縮；中央上下矩形將依水平方向縮放，但不依垂直方向縮放，而同時左右中間矩形將依垂直方向縮放而不依水平方向縮放；中央矩形則將同時依水平及垂直方向縮放。

請記住，若是建立顯示物件，而您要某些內容永遠不縮放，則只需確實放置 9 分割縮放網格的分割線，以便讓內容落在其中一個角上矩形中。

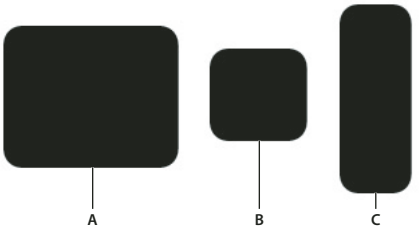
在 ActionScript 中設定顯示物件的 scale9Grid 屬性值會開啟物件的 9 分割縮放功能，並在物件的 9 倍縮放網格中定義矩形大小。您可以使用 Rectangle 類別的實體，做為 scale9Grid 屬性的值，如下所示：

```
myButton.scale9Grid = new Rectangle(32, 27, 71, 64);
```

Rectangle 建構函式的四個參數是 x 座標、y 座標、寬度和高度。在此範例中，矩形的左上角是放置於名為 myButton 之顯示物件的 x: 32, y: 27 這個點上，矩形是 71 個像素寬，及 64 個像素高 (因此右邊位於顯示物件上 x 座標 103，而其底邊則位於顯示物件上 y 座標 92)。

包含於由 Rectangle 實體所定義區域中的實際區域代表 9 倍縮放網格的中央矩形，其它矩形是由 Flash Player 和 AIR 擴充 Rectangle 實體的各邊加以計算，如下所示：

在此範例中，當按鈕進行大小縮放，圓角將不會延伸或壓縮，但其它區域則將隨著調整，配合縮放。



A. myButton.width = 131；myButton.height = 106； **B.** myButton.width = 73；myButton.height = 69； **C.** myButton.width = 54；myButton.height = 141；

快取顯示物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

不論您正在建立的是應用程式還是以指令碼建立的複雜動畫，當 Flash 中的設計程式愈來愈龐大時，就必須同時考慮到效能和最佳化的問題。當您有內容保持為靜態 (如矩形 Shape 實體) 時，Flash Player 和 AIR 不會將內容最佳化，因此，當您變更矩形的位置時，Flash Player 或 AIR 便會重新繪製整個 Shape 實體。

您可以快取指定的顯示物件，以改善 SWF 檔的效能。顯示物件是 「表面」，基本上是實體之向量資料的點陣圖版本，這些是您不想要在 SWF 檔的處理過程中變更太多的資料。因此，開啟了快取的實體並不是隨著 SWF 檔播放連續重新繪製，而是讓 SWF 檔快速呈現。

備註：您可以在重新建立表面時更新向量資料。因此，表面中快取的向量資料不需要與整個 SWF 檔相同。

將顯示物件的 cacheAsBitmap 屬性設定為 true 可讓顯示物件快取本身的點陣圖形式。Flash Player 或 AIR 會建立該實體的表面物件，這是一個快取點陣圖而非向量資料。如果您變更顯示物件的範圍，則會重新建立表面，而非重新調整表面的大小。表面可以巢狀結構出現在其它表面中。子表面會複製其點陣圖至其父表面上。如需詳細資訊，請參閱第 155 頁 「啟用點陣圖快取」。

DisplayObject 類別的 opaqueBackground 屬性和 scrollRect 屬性是與使用 cacheAsBitmap 屬性快取的點陣圖相關。雖然這三個屬性彼此獨立，但當物件快取為點陣圖時，opaqueBackground 和 scrollRect 屬性運作最順暢，只有在將 cacheAsBitmap 設定為 true 時，才會看到 opaqueBackground 和 scrollRect 屬性的效能優點。如需有關捲動顯示物件內容的詳細資訊，請參閱第 150 頁 「左右移動和捲動顯示物件」。如需有關設定不透明背景的詳細資訊，請參閱第 156 頁 「設定不透明背景顏色」。

如需有關 Alpha 色版遮色片 (必須將 cacheAsBitmap 屬性設定為 true) 的詳細資訊，請參閱第 160 頁 「遮蓋顯示物件」。

啟用快取的時機

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

啟用顯示物件的快取會建立表面，這會有幾項好處，例如幫助複雜的向量動畫更快速呈現。多數情況下，您都會想啟用快取，而且可能會想要一直啟用快取來改善 SWF 檔的效能，但是在某些情況下，啟用快取並不能改善效能，有的情況甚至會降低效能。本節會說明在哪些情況下應該使用快取，哪些情況下則使用一般的顯示物件即可。

快取資料的整體效能是依據實體向量資料的複雜程度、變更的資料多寡，以及是否設定 opaqueBackground 屬性而定。如果您只變更小區域，則使用表面和使用向量資料的差異不大。在您部署應用程式之前，可以先測試這兩種情況。

使用點陣圖快取的時機

以下是在啟用點陣圖快取時，會產生明顯優勢的典型狀況：

- 複雜的背景影像：含有複雜且詳盡向量資料之背景影像的應用程式，這個向量資料或許是套用了追蹤點陣圖命令後的影像，或是以 Adobe Illustrator® 建立的圖像。這個背景上或許還有動畫字元，由於背景必須持續地重新產生向量資料，因此會降低這個動畫的速度。若要改善效能，可以將背景顯示物件的 opaqueBackground 屬性設定為 true，如此這個背景便會以點陣圖的方式呈現，並且可以快速地重新繪製，增加播放動畫的速度。
- 捲動的文字欄位：在捲動的文字欄位中顯示大量文字的應用程式。您可以將文字欄位放置於設定為可捲動並具有捲動範圍 (scrollRect 屬性) 的顯示物件中。這會啟用所指定實體的快速像素捲動功能。當使用者捲動這個顯示物件實體時，Flash Player 或 AIR 便會移開已捲動的像素，並產生新顯露區域的內容，而不是重新產生整個文字欄位。
- 重疊視窗的系統：具有重疊視窗之複雜系統的應用程式。每個視窗都可以加以開啟或關閉 (例如，網頁瀏覽器視窗)。如果將每個視窗都標示為 「表面」 (將 cacheAsBitmap 屬性設定為 true)，就會分離並快取每個視窗。如此使用者便可以拖曳視窗讓這些視窗互相重疊，而且每個視窗並不需要重新產生向量內容。
- Alpha 色版遮色片：當您使用 Alpha 色版遮色片時，必須將 cacheAsBitmap 屬性設定為 true。如需詳細資訊，請參閱第 160 頁 「遮蓋顯示物件」。

在上述的情況中啟用點陣圖快取，都可以藉由最佳化向量圖形，改善應用程式的回應和互動品質。

此外，只要將濾鏡套用至顯示物件，即使明確將 cacheAsBitmap 設定為 false，它也會自動設定為 true。如果您將顯示物件的所有濾鏡都清除，cacheAsBitmap 屬性就會還原為上一次的設定。

避免使用點陣圖快取的時機

在錯誤的情況下使用此功能，可能會對 SWF 檔案的效能造成負面的影響。使用點陣圖快取時，請切記下列原則：

- 請勿過度使用表面 (啟用快取的顯示物件)。每個表面都會比一般顯示物件使用更多的記憶體，這表示您應該只在需要改善成像效能時，才啟用這些表面。

  每個快取的點陣圖都會比一般顯示物件使用更多的記憶體，例如，如果「舞台」上 Sprite 實體的大小為 250 x 250 像素，在快取時可能會使用 250 KB 的記憶體資源，而一般 (未快取) 的 Sprite 實體則僅使用 1 KB 的記憶體而已。
- 請避免放大快取的表面。如果您過度使用點陣圖快取，便會消耗掉大量的記憶體 (請參閱上一個標題)，尤其是在放大內容時。
- 在大部分顯示物件實體都是靜態 (非動畫) 的情況下，請使用表面。您可以拖曳或移動實體，但實體的內容不應過度被製成動畫或變更。動畫或變更內容比較可能在包含動畫或 Video 實體的 MovieClip 實體發生。例如，如果您旋轉或轉換某個實體，這個實體便會在表面和向量資料之間進行變更，如此不僅難以處理，也會對 SWF 檔產生負面的影響。
- 如果您將表面與向量資料混合，會增加 Flash Player 和 AIR (有時候是電腦) 需要處理的資料量。所以，請盡量將多個表面組織在一起 (例如，在建立重疊視窗的應用程式時)。
- 請勿快取圖形經常變更的物件。每次您縮放、傾斜、旋轉顯示物件；變更 Alpha 或顏色變化；移動子顯示物件；或是使用圖形屬性繪製時，就會重繪點陣圖快取。如果每個影格都發生此狀況，執行階段必須將物件繪製成點陣圖，然後將該點陣圖複製到舞台，相較於只將未快取的物件繪製到舞台，這將會造成額外的工作。快取與更新頻率之間的效能問題須視顯示物件的複雜度與大小而定，而且只能由測試特定內容決定。

啟用點陣圖快取

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要啟用顯示物件的點陣圖快取，請將其 cacheAsBitmap 屬性設定為 true：

```
mySprite.cacheAsBitmap = true;
```

將 cacheAsBitmap 屬性設定為 true 之後，您可能會發現顯示物件會以像素為座標值，自動對齊整個座標。當您測試 SWF 檔時，應該會注意到任何動畫在複雜的向量影像上會呈現得更快。

如果發生下列其中一或多種情況，即使將 cacheAsBitmap 設為 true，也不會建立表面 (快取的點陣圖)：

- 點陣圖的寬度或高度大於 2880 像素。
- 點陣圖無法配置記憶體 (因為發生記憶體不足的錯誤)。

快取的點陣圖變形矩陣

Adobe AIR 2.0 及更新的版本 **(** 行動描述檔 **)**

在行動裝置的 AIR 應用程式中，您應該在每次設定 cacheAsBitmap 屬性時，同時設定 cacheAsBitmapMatrix 屬性。設定此屬性可讓您將範圍較廣的變形套用至顯示物件，而不會觸發重新顯示。

```
mySprite.cacheAsBitmap = true;
mySprite.cacheAsBitmapMatrix = new Matrix();
```

當您設定此矩陣屬性時，可以將下列其他變形套用至顯示物件，而不必到達物件：

- 不進行像素貼齊即移動或轉譯
- 旋轉
- 縮放

- 傾斜
- 變更 Alpha (在 0 與 100% 透明度之間)

這些變形會直接套用至快取的點陣圖。

設定不透明背景顏色

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以為顯示物件設定不透明的背景。例如，當 SWF 具有含複雜的向量圖像背景時，您可以將 opaqueBackground 屬性設為指定的顏色 (通常是和 「舞台」相同的顏色)，顏色會以數字 (一般是十六進位顏色值) 來指定，然後這個背景便會被視為點陣圖，這可以幫助最佳化效能。

當您將 cacheAsBitmap 設為 true，並將 opaqueBackground 屬性設為指定的顏色時，opaqueBackground 屬性便會允許內部點陣圖成為不透明，並加快成像的速度。如果您不將 cacheAsBitmap 設為 true，opaqueBackground 屬性便會將不透明的向量正方形形狀加入至此顯示物件的背景。如此一來，就不會自動建立點陣圖。

下列範例會示範如何設定顯示物件的背景，以便最佳化效能。

```
myShape.cacheAsBitmap = true;
myShape.opaqueBackground = 0xFF0000;
```

在此範例中，名為 myShape 的 Shape 背景顏色是設定為紅色 (0xFF0000)。假設 Shape 實體包含在 「舞台」上繪有白色背景的綠色三角形，就會顯示為 Shape 實體的範圍框 (完全包住 Shape 的矩形) 中空白區域是紅色的綠色三角形。

當然，若這段程式碼是搭配純紅色背景的 「舞台」來使用則會更合理：在另一個顏色背景上，就會指定為該顏色。例如，在背景為白色的 SWF 檔中，opaqueBackground 屬性很可能就設定為 0xFFFFFF 純白色。

套用混合模式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

混合模式是將一個影像 (基底影像) 的顏色與另一個影像 (混合影像) 的顏色相結合，來產生第三個影像，所形成的影像才是實際顯示在螢幕上的影像。影像中的每個像素值都會使用另一個影像的對應像素值來處理，而在原位置產生像素值結果。

每一個顯示物件都有 blendMode 屬性，可以設定為下列其中一個混合模式，這些是在 BlendMode 類別中定義的常數。另外，您也可以使用為常數實際值的 String 值 (在括號中)。

- BlendMode.ADD ("add")：通常用於在兩個影像之間建立漸亮溶解的動畫特效。
- BlendMode.ALPHA ("alpha")：通常用於將前景的透明度套用在背景上。(在 GPU 顯示下不支援。)
- BlendMode.DARKEN ("darken")：通常用於疊置類型。(在 GPU 顯示下不支援。)
- BlendMode.DIFFERENCE ("difference")：通常用於建立更鮮明的顏色。
- BlendMode.ERASE ("erase")：通常用於施加前景 Alpha 透明度，以便刪去 (擦除) 背景的某部位。(在 GPU 顯示下不支援。)
- BlendMode.HARDLIGHT ("hardlight")：通常用於建立陰影特效。(在 GPU 顯示下不支援。)
- BlendMode.INVERT ("invert")：用於反轉背景。

- BlendMode.LAYER ("layer")：用於在特定顯示物件的預先合成階段強制建立暫時緩衝。(在 GPU 顯示下不支援。)
- BlendMode.LIGHTEN ("lighten")：通常用於疊置類型。(在 GPU 顯示下不支援。)
- BlendMode.MULTIPLY ("multiply")：通常用於建立陰影和深度特效。
- BlendMode.NORMAL ("normal")：用於指定混合影像的像素值應覆蓋基底影像的像素值。
- BlendMode.OVERLAY ("overlay")：通常用於建立陰影特效。(在 GPU 顯示下不支援。)
- BlendMode.SCREEN ("screen")：通常用於建立反白和光暈。
- BlendMode.SHADER ("shader")：用於指定使用 Pixel Bender 著色器建立自訂混合特效。如需有關使用著色器的詳細資訊，請參閱第 253 頁「使用 Pixel Bender 著色器」。(在 GPU 顯示下不支援。)
- BlendMode.SUBTRACT ("subtract")：通常用於建立兩個影像漸暗溶解的動畫特效。

調整 DisplayObject 顏色

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 ColorTransform 類別 (flash.geom.ColorTransform) 的方法，調整顯示物件的顏色。每一個顯示物件都有 transform 屬性，它是 Transform 類別的實體，其中包含有關套用至顯示物件之各種變形的資訊 (如旋轉、縮放或位置變更，諸如此類)。除了有關幾何變形的資訊以外，Transform 類別也包含 colorTransform 屬性，它是 ColorTransform 類別的實體，可供對顯示物件進行顏色調整。若要存取顯示物件的顏色轉換資訊，可以使用如下的程式碼：

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;
```

建立了 ColorTransform 實體之後，您可以讀取其屬性值，找出已經套用的顏色轉換，也可以設定這些值，對顯示物件進行顏色變更。若要在任何變更之後更新顯示物件，必須重新將 ColorTransform 實體指定回 transform.colorTransform 屬性。

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;

// Make some color transformations here.

// Commit the change.
myDisplayObject.transform.colorTransform = colorInfo;
```

使用程式碼設定顏色值

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ColorTransform 類別的 color 屬性可用來將特定紅、綠、藍 (RGB) 顏色值指定給顯示物件。下列範例會使用 color 屬性，當使用者按一下名為 blueBtn 的按鈕時，將名為 square 的顯示物件顏色變更為藍色：

```
// square is a display object on the Stage.
// blueBtn, redBtn, greenBtn, and blackBtn are buttons on the Stage.

import flash.events.MouseEvent;
import flash.geom.ColorTransform;

// Get access to the ColorTransform instance associated with square.
var colorInfo:ColorTransform = square.transform.colorTransform;

// This function is called when blueBtn is clicked.
function makeBlue(event:MouseEvent):void
{
    // Set the color of the ColorTransform object.
    colorInfo.color = 0x003399;
    // apply the change to the display object
    square.transform.colorTransform = colorInfo;
}

blueBtn.addEventListener(MouseEvent.CLICK, makeBlue);
```

請注意，當您使用 color 屬性變更顯示物件的顏色時，會完全變更整個物件的顏色，不管物件在之前是否有多個顏色都一樣。例如，如果有顯示物件包含頂端有黑色文字的綠色圓形，將該物件的相關聯 ColorTransform 實體的 color 屬性設定為紅色形狀會讓整個物件、圓形和文字都變為紅色 (因此文字將無法再與其餘物件區別)。

使用程式碼改變顏色和亮度特效

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

假設有包含多個顏色的顯示物件 (例如，數位相片)，而且您不要完全重新為物件上色，則只要根據現有顏色調整顯示物件的顏色即可。在這種情況下，ColorTransform 類別包含一系列倍數及偏移屬性，可供您用來進行這類調整。名為 redMultiplier、greenMultiplier、blueMultiplier 和 alphaMultiplier 的倍數屬性運作方式與彩色照相濾鏡 (或彩色太陽眼鏡) 類似，會增加或減少顯示物件中的某些顏色。偏移屬性 (redOffset、greenOffset、blueOffset 和 alphaOffset) 可以用來將某些顏色的額外色量加入物件，或指定該特定顏色可有的最小值。

當您在 「屬性」檢測器上的 「顏色」彈出式選單中選擇 「進階」時，這些倍數和偏移屬性會與 Flash 編寫工具中影片片段元件可用的進階顏色設定完全相同。

下列程式碼會載入 JPEG 影像，然後套用顏色轉換，當滑鼠指標沿著 x 軸和 y 軸移動時修改紅色和綠色色版。在此範例中，因為已指定了偏移值，顯示在螢幕上每一個顏色色版的顏色值都是影像中原始顏色的百分比，也就是說，顯示在任何指定像素中最紅或最綠的將是該像素中的原始紅色或綠色量。

```
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.geom.Transform;
import flash.geom.ColorTransform;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// This function is called when the mouse moves over the loaded image.
function adjustColor(event:MouseEvent):void
{
    // Access the ColorTransform object for the Loader (containing the image)
    var colorTransformer:ColorTransform = loader.transform.colorTransform;

    // Set the red and green multipliers according to the mouse position.
    // The red value ranges from 0% (no red) when the cursor is at the left
    // to 100% red (normal image appearance) when the cursor is at the right.
    // The same applies to the green channel, except it's controlled by the
    // position of the mouse in the y axis.
    colorTransformer.redMultiplier = (loader.mouseX / loader.width) * 1;
    colorTransformer.greenMultiplier = (loader.mouseY / loader.height) * 1;

    // Apply the changes to the display object.
    loader.transform.colorTransform = colorTransformer;
}

loader.addEventListener(MouseEvent.MOUSE_MOVE, adjustColor);
```

旋轉物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 rotation 屬性，旋轉顯示物件。您可以讀取此值，得知物件是否已旋轉，也可以設定此屬性為數字 (度數) 代表要套用至物件的旋轉量以旋轉該物件。例如，下面這行程式碼會將名為 square 的物件旋轉 45 度 (一個整圈的八分之一)：

```
square.rotation = 45;
```

另外，您也可以使用變形矩陣旋轉顯示物件，請參閱第 177 頁 「處理幾何」。

淡化物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以控制顯示物件的透明度，讓該物件部分或全部透明，也可以變更透明度讓該物件以淡入或淡出方式顯示。DisplayObject 類別的 alpha 屬性會定義顯示物件的透明度 (其實更準確的說法是，不透明度)。alpha 屬性可設定為 0 與 1 之間的任何值，其中 0 是完全透明，而 1 則是完全不透明。例如，在下列程式碼行中，會示範用滑鼠按一下名為 myBall 的物件時，讓物件呈現部分 (50%) 透明：

```
function fadeBall(event:MouseEvent):void
{
    myBall.alpha = .5;
}
myBall.addEventListener(MouseEvent.CLICK, fadeBall);
```

您也可以利用可透過 ColorTransform 類別使用的顏色調整，改變顯示物件的透明度。如需詳細資訊，請參閱第 157 頁「調整 DisplayObject 顏色」。

遮蓋顯示物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以將顯示物件當做遮色片使用以建立缺口，透過這個缺口可以看到另一個顯示物件的內容。

定義遮色片

若要指出顯示物件將成為另一個顯示物件的遮色片，請將該遮色片物件設定為要遮蓋之顯示物件的 mask 屬性：

```
// Make the object maskSprite be a mask for the object mySprite.
mySprite.mask = maskSprite;
```

被遮蓋的顯示物件會在做為遮色片的顯示物件之所有不透明 (非透明) 的區域下呈現。例如，下列程式碼會建立 Shape 實體 (其中包含紅色 100 x 100 像素方形)，以及 Sprite 實體 (其中包含半徑為 25 像素的藍色圓形)。按一下圓形時，它會設定為正方形的遮色片，使得正方形只顯示由實心圓形部分所遮蓋的地方，換句話說，將只能看見紅色圓形。

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.Shape;

// Draw a square and add it to the display list.
var square:Shape = new Shape();
square.graphics.lineStyle(1, 0x000000);
square.graphics.beginFill(0xff0000);
square.graphics.drawRect(0, 0, 100, 100);
square.graphics.endFill();
this.addChild(square);

// Draw a circle and add it to the display list.
var circle:Sprite = new Sprite();
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0x0000ff);
circle.graphics.drawCircle(25, 25, 25);
circle.graphics.endFill();
this.addChild(circle);

function maskSquare(event:MouseEvent):void
{
    square.mask = circle;
    circle.removeEventListener(MouseEvent.CLICK, maskSquare);
}

circle.addEventListener(MouseEvent.CLICK, maskSquare);
```

做為遮色片的顯示物件可以拖曳、製成動畫、可動態調整大小，並可在單一遮色片中使用不同的形狀。遮色片顯示物件不一定要加入顯示清單中，但是若要遮色片物件在「舞台」縮放時也跟著縮放，或是若要讓使用者與遮色片互動 (例如使用者控制拖曳及調整大小)，則必須將遮色片物件加入顯示清單中。顯示清單的實際 z 索引 (由前至後順序) 沒關係，只要遮色片物件加入顯示清單即可 (遮色片物件只會在螢幕上顯示為遮色片)。若遮色片物件是有多個影格的 MovieClip 實體，就會在其時間軸上播放所有影格，若不做為遮色片，也會有相同情況。您可以將 mask 屬性設定為 null，藉以移除遮色片：

```
// remove the mask from mySprite
mySprite.mask = null;
```

您不能使用遮色片來遮罩另一個遮色片。您不能設定遮色片顯示物件的 alpha 屬性。用來做為遮色片的顯示物件中，只能使用填色，而會忽略筆畫。

AIR 2

若被遮色的顯示物件是透過設定 cacheAsBitmap 與 cacheAsBitmapMatrix 屬性加以快取，則遮色片必須是被遮色的顯示物件的子系。同樣地，若被遮色的顯示物件是快取之顯示物件容器的子系，則遮色片與顯示物件都必須是該容器的子系。若被遮色的物件是一個以上的已快取顯示物件容器的子系，則遮色片必須是在顯示清單中最接近被遮色物件的已快取容器的子系。

關於遮蓋裝置字體

您可以使用顯示物件，遮蓋採用裝置字體的文字內容。使用顯示物件遮蓋裝置字體中的文字集時，會使用遮色片的矩形範圍框做為遮蓋形狀；也就是說，如果您為裝置字體文字建立了非矩形的顯示物件遮色片，則出現在 SWF 檔中的遮色片會顯示為遮色片的矩形範圍框形狀，而不是遮色片本身的形狀。

Alpha 色版遮色片

如果遮色片和受到遮色的顯示物件都使用點陣圖快取時，便會支援 Alpha 色版遮色片，如下所示：

```
// maskShape is a Shape instance which includes a gradient fill.
mySprite.cacheAsBitmap = true;
maskShape.cacheAsBitmap = true;
mySprite.mask = maskShape;
```

例如，Alpha 色板遮色片的一種應用是在套用於已遮蓋顯示物件的濾鏡之外、另外在遮色片物件上使用濾鏡。

在下列範例中，會在「舞台」上載入外部影像檔，該影像 (更準確地說，是此影像所載入的目的地 Loader 實體) 將會是遭到遮蓋的顯示物件；漸層橢圓形 (純黑色中心逐漸淡出透明邊緣) 會繪製在影像之上，這將是 Alpha 遮色片。兩個顯示物件都已開啟點陣圖快取，橢圓形是設定為影像的遮色片，然後將它變成可拖曳。

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Sprite;
import flash.geom.Matrix;
import flash.net.URLRequest;

// Load an image and add it to the display list.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create a Sprite.
var oval:Sprite = new Sprite();
// Draw a gradient oval.
var colors:Array = [0x000000, 0x000000];
var alphas:Array = [1, 0];
var ratios:Array = [0, 255];
```

```
var matrix:Matrix = new Matrix();
matrix.createGradientBox(200, 100, 0, -100, -50);
oval.graphics.beginGradientFill(GradientType.RADIAL,
                                colors,
                                alphas,
                                ratios,
                                matrix);
oval.graphics.drawEllipse(-100, -50, 200, 100);
oval.graphics.endFill();
// add the Sprite to the display list
this.addChild(oval);

// Set cacheAsBitmap = true for both display objects.
loader.cacheAsBitmap = true;
oval.cacheAsBitmap = true;
// Set the oval as the mask for the loader (and its child, the loaded image)
loader.mask = oval;

// Make the oval draggable.
oval.startDrag(true);
```

將物件製成動畫

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

動畫是讓事物移動的程序，也可以說，是讓事物隨著時間變更。以指令碼編寫的動畫是電玩遊戲的基礎部分，而且經常會用來修飾或是將有用的互動線索加入至其它應用程式。

以指令碼編寫之動畫背後的基本構想是畫面必須要產生變化，而且這個變化必須依時間劃分增量。在 ActionScript 中，只要使用一般迴圈陳述式便可以輕鬆地重複動作。但是，迴圈會在執行循環中所有項目之後，才會更新顯示畫面。因此若要以程式碼建立動畫，您必須撰寫 ActionScript，在一段時間內重複執行一些動作，而且每次執行時都會更新螢幕。

例如，想像一下您要建立簡單的動畫，如讓一個球滾過整個螢幕。ActionScript 包含簡單的機制，可以讓您追蹤時間的推移，並據此更新螢幕。也就是說，您可以撰寫每次將球移動一點點的程式碼，一直到它到達目的地為止。每次移動過後，螢幕都會更新，讓檢視者看見橫越「舞台」的動作。

從實際的觀點來看，將以程式碼建立的動畫與 SWF 檔的影格速率同步化是很合理的 (換句話說，讓一個動畫在每次顯示或要顯示新影格時變更)，因為這會定義 Flash Player 或 AIR 更新螢幕的頻率。每個顯示物件都有 enterFrame 事件，根據 SWF 檔的影格速率傳送，一個事件一個影格。大部分開發人員以程式碼建立動畫時，都會使用 enterFrame 事件，建立在一段時間內重複的動作。您可以撰寫偵聽 enterFrame 事件的程式碼，將製成動畫的球在每個影格中移動一定的量，而且當螢幕 (每個影格) 更新時，球會在新位置上重新繪製，而產生動作。

備註：另外還有一種方式可以在一段時間內重複執行動作，就是使用 Timer 類別。Timer 實體會在每次經過指定的時間後觸發事件通知。您可以撰寫程式碼，透過處理 Timer 類別的 timer 事件，將時間間隔設定為很小的值 (幾分之一秒) 來執行動畫。如需有關使用 Timer 類別的詳細資訊，請參閱第 3 頁「控制時間間隔」。

在下列範例中，會在「舞台」上建立名為 circle 的圓形 Sprite 實體。當使用者按一下圓形時，以指令碼編寫的動畫序列即開始進行，使得 circle 淡出 (其 alpha 屬性會減少) 直到完全透明為止：

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.MouseEvent;

// draw a circle and add it to the display list
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0x990000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();
addChild(circle);

// When this animation starts, this function is called every frame.
// The change made by this function (updated to the screen every
// frame) is what causes the animation to occur.
function fadeCircle(event:Event):void
{
    circle.alpha -= .05;

    if (circle.alpha <= 0)
    {
        circle.removeEventListener(Event.ENTER_FRAME, fadeCircle);
    }
}

function startAnimation(event:MouseEvent):void
{
    circle.addEventListener(Event.ENTER_FRAME, fadeCircle);
}

circle.addEventListener(MouseEvent.CLICK, startAnimation);
```

當使用者按一下圓形時，函數 fadeCircle() 會訂閱做為 enterFrame 事件的偵聽程式，表示從這時開始，每進入一個影格都會呼叫一次這個函數。該函數會透過變更其 alpha 屬性淡出 circle，因此每個影格圓形的 alpha 減少 .05 (5%) 以後，螢幕就會更新。最後，當 alpha 值為 0 (circle 完全透明) 時，fadeCircle() 函數即移除，不再為事件偵聽程式，而結束動畫。

例如，您可以使用相同的程式碼，建立動畫動作，而不進行淡出。在成為 enterFrame 事件偵聽程式的函數中，以另一個屬性取代 alpha，就會改為建立該屬性的動畫。例如，將下列這一行

```
circle.alpha -= .05;
```

變更為下面這行程式碼

```
circle.x += 5;
```

就會建立 x 屬性的動畫，使得圓形向右移過 「舞台」。結束動畫的條件可以變更為在達到所要的 x 座標時結束動畫 (也就是，取消訂閱 enterFrame 偵聽程式)。

舞台方向

AIR 2.0 及更新的版本

當使用者旋轉裝置時，行動裝置通常會調整使用者介面的方向，以保持顯示為直立。如果您在應用程式中啟用自動調整方向，裝置會將顯示保持在正確的方向，但是當舞台的外觀比例變更時，您可以自行決定以確定內容看起來是否妥當。如果您停用自動調整方向，則裝置顯示會保持在固定的方向，除非您手動變更方向。

AIR 應用程式會在不同的行動裝置與作業系統上執行。基礎方向行為會隨不同作業系統而異，甚至相同作業系統在不同裝置上也會不一樣。在所有裝置與作業系統都運作良好的簡單設計策略，就是啟用自動方向並偵聽 「舞台」 resize 事件，以判斷何時需要重新整理應用程式的版面。

或者，如果您的應用程式只支援縱向外觀比例或只支援橫向外觀比例，則可以關閉自動方向，並在 AIR 應用程式描述器中設定支援的外觀比例。此設計策略提供一致的行為，並且會為選取的外觀比例選取 「最佳的方向」。例如，如果您指定橫向外觀比例，則所選的方向適用於橫向模式且具備外滑鍵盤的裝置。

取得目前的 「舞台」方向和外觀比例

報告的方向會相對於裝置的正常位置。在大部分的裝置上，有一個明確且直立的位置。此位置被視為 「預設」方向。另外三個可能的方向為：「往左旋轉」、「往右旋轉」以及 「上下顛倒」。StageOrientation 類別會定義設定或比較方向值時要使用的字串常數。

Stage 類別會定義報告方向的兩個屬性：

- Stage.deviceOrientation — 報告相對於預設位置的裝置實體方向。

  備註：當您的應用程式剛啟動或是裝置平放時，不一定可以使用 deviceOrientation。在這些情況下，會將裝置方向報告成「不明」。

- Stage.orientation — 報告相對於預設位置的舞台方向。啟用自動方向時，舞台會以裝置相反的方向旋轉以保持直立。因此，orientation 屬性報告的右邊與左邊位置與 deviceOrientation 屬性所報告的位置相反。例如，當 deviceRotation 報告 「往右旋轉」時，orientation 會報告 「往左旋轉」。

只要比較舞台目前的寬度與高度，就可以得到舞台的外觀比例：

```
var aspect:String = this.stage.stageWidth >= this.stage.stageHeight ? StageAspectRatio.LANDSCAPE :
StageAspectRatio.PORTRAIT;
```

自動方向

當自動方向開啟且使用者旋轉其裝置時，作業系統會調整整個使用者介面的方向，包括系統工具列與您的應用程式。因此，舞台的外觀比例會從縱向變更為橫向，從橫向變更為縱向。外觀比例變更時，舞台尺寸也會變更。

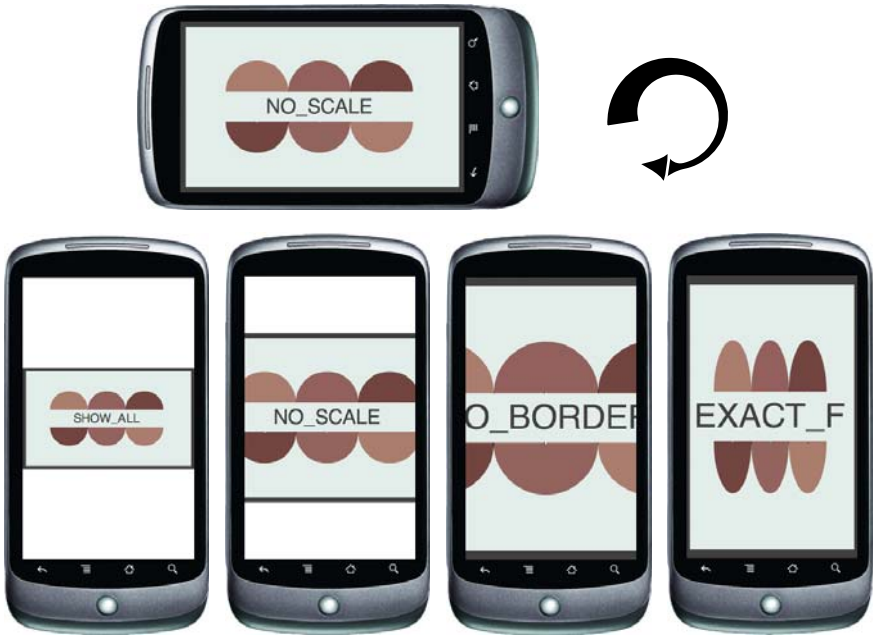
將 「舞台」的 autoOrients 屬性設為 true 或 false，會在執行階段啟用或停用自動方向。您可以使用 <autoOrients> 元素，在 AIR 應用程式描述器中設定此屬性的初始值。(請注意，在 AIR 2.6 之前，autoOrients 是唯讀屬性，而且只能在應用程式描述器中設定)。

如果您指定橫向或縱向外觀比例，同時也啟用自動調整方向，則 AIR 會將自動調整方向限制為指定的外觀比例。

舞台尺寸變更

當舞台尺寸變更時，舞台內容會按照 「舞台」物件指定的 scaleMode 與 align 屬性來縮放和重新放置。在大部分的情況下，依賴舞台 scaleMode 設定所提供的自動行為不會產生良好的結果。而是您必須重新配置或重繪圖形與元件，以支援多個外觀比例。(提供彈性的版面邏輯，同時也表示您的應用程式可在不同的螢幕大小與外觀比例的裝置上運作地更好)。

下列說明示範旋轉一般行動裝置時，不同 scaleMode 設定的效果：



從橫向旋轉到縱向外觀比例

此說明示範使用不同縮放模式從橫向外觀比例旋轉至縱向外觀比例時，所發生的縮放行為。從縱向旋轉到橫向會造成一組類似的效果。

方向變更事件

Stage 物件會傳送兩個事件類型，您可以偵測此事件，並對方向變更做出反應。啟用自動方向時，會同時傳送舞台的 resize 與 orientationChange 事件。

當您依賴自動方向以保持 ⊠ 示直立時，resize 事件是可使用的最佳事件。當舞台傳送 resize 事件時，請視需要重新配置或重繪您的內容。將舞台縮放模式設為 noScale 時，只會傳送 resize 事件。

orientationChange 事件也可用以偵測方向變更。啟用自動方向時，只會傳送 orientationChange 事件。

備註：在一些行動平台上，舞台會先傳送可取消的 orientationChanging 事件，再傳送 resize 或 orientationChange 事件。因為並非所有的平台都支援此事件，所以請避免依賴它。

手動方向

AIR 2.6 及更新的版本

您可以使用舞台 setOrientation() 或 setAspectRatio() 方法來控制舞台方向。

設定舞台方向

您可以使用 Stage 物件的 setOrientation() 方法，在執行階段設定舞台方向。使用 StageOrientation 類別定義的字串常數可指定想要的方向。

```
this.stage.setOrientation( StageOrientation.ROTATED_RIGHT );
```

並非所有裝置和作業系統都支援所有可能的方向。例如，Android 2.2 不支援以程式設計方式在縱向標準裝置上選擇向左旋轉，也不支援倒轉方向。舞台的 supportedOrientations 屬性提供可傳送到 setOrientation() 方法的方向清單：

```
var orientations:Vector.<String> = this.stage.supportedOrientations;
for each( var orientation:String in orientations )
{
    trace( orientation );
}
```

設定舞台外觀比例

如果您主要的考量是舞台外觀比例，可以將外觀比例設定為縱向或橫向。您可以在 AIR 應用程式描述器設定外觀比例或在執行階段使用舞台 setAspectRatio() 方法：

```
this.stage.setAspectRatio( StageAspectRatio.LANDSCAPE );
```

執行階段選擇指定外觀比例的兩個可能方向的其中之一。這可能不符合目前裝置方向。例如，偏好設定中的預設方向會選擇為倒轉方向 (AIR 3.2 和更早版本)，而偏好設定中外滑鍵盤適用的方向為相反方向。

(AIR 3.3 及更新版本 **)** 從 AIR 3.3 開始 (SWF 版本 16)，您也可以使用 StageAspectRatio.ANY 常數。如果 Stage.autoOrients 設定為 true，並且您呼叫 setAspectRatio(StageAspectRatio.ANY)，您的應用程式會具有重新沿向所有方向的能力 (landscape-left、landscape-right、portait 和 portrait-upside-down)。這也是 AIR 3.3 的新功能，外觀比例具永續性，並且限制裝置在進一步旋轉時只能旋轉至指定的方向。

範例：設定舞台方向以符合裝置方向

以下範例說明更新舞台方向以符合目前裝置方向的功能。舞台 deviceOrientation 屬性表示裝置的實際方向，即使自動調整方向已關閉。

```
function refreshOrientation( theStage:Stage ):void
{
    switch ( theStage.deviceOrientation )
    {
        case StageOrientation.DEFAULT:
            theStage.setOrientation( StageOrientation.DEFAULT );
            break;
        case StageOrientation.ROTATED_RIGHT:
            theStage.setOrientation( StageOrientation.ROTATED_LEFT );
            break;
        case StageOrientation.ROTATED_LEFT:
            theStage.setOrientation( StageOrientation.ROTATED_RIGHT );
            break;
        case StageOrientation.UPSIDE_DOWN:
            theStage.setOrientation( StageOrientation.UPSIDE_DOWN );
            break;
        default:
            //No change
    }
}
```

方向更改為非同步。您可以偵聽舞台傳送的 orientationChange 事件以偵測更改的完成。如果裝置不支援方向，setOrientation() 呼叫會失敗，且不會擲出錯誤。

以動態方式載入顯示內容

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以將下列任何外部顯示資源載入 ActionScript 3.0 應用程式中：

- 以 ActionScript 3.0 編寫的 SWF 檔 - 這個檔案可以是 Sprite、MovieClip 或擴充 Sprite 的任何類別。在 iOS 上的 AIR 應用程式 r，僅可載入不含 ActionScript 位元碼的 SWF 檔。這表示可載入包含內嵌資料，如影像和聲音的 SWF 檔，但不可載入包含可執行程式碼的 SWF 檔。
- 影像檔 - 包括 JPG、PNG 和 GIF 檔。
- AVM1 SWF 檔 - 這是以 ActionScript 1.0 或 2.0 撰寫的 SWF 檔 (行動 AIR 應用程式不支援)。

您可以使用 Loader 類別載入這些資源。

載入顯示物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Loader 物件是用來將 SWF 檔及圖像檔載入應用程式中。Loader 類別是 DisplayObjectContainer 類別的子類別。Loader 物件只能在其顯示清單中包含一個子顯示物件，顯示物件代表它所載入的 SWF 或圖像檔。當您將 Loader 物件加入顯示清單時 (如下列程式碼所示)，也會在載入子顯示物件後，將其加入至顯示清單：

```
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
this.addChild(pictLdr);
```

載入 SWF 檔或影像以後，您可以將載入的顯示物件移至另一個顯示物件容器，如本範例中的 container DisplayObjectContainer 物件：

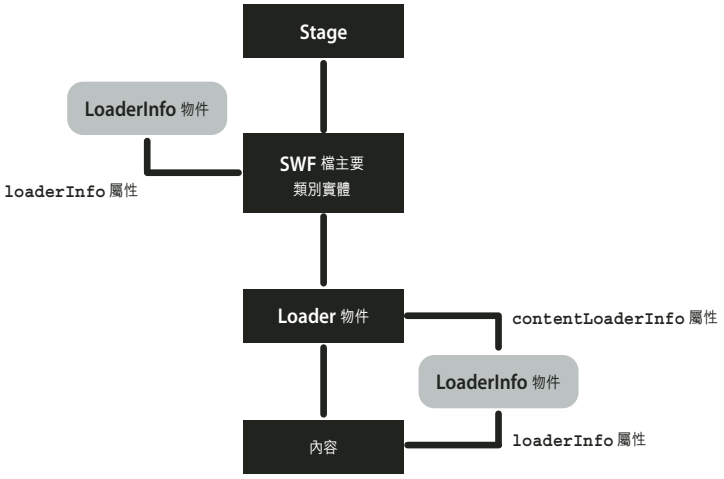
```
import flash.display.*;
import flash.net.URLRequest;
import flash.events.Event;
var container:Sprite = new Sprite();
addChild(container);
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
pictLdr.contentLoaderInfo.addEventListener(Event.COMPLETE, imgLoaded);
function imgLoaded(event:Event):void
{
    container.addChild(pictLdr.content);
}
```

監視載入進度

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

檔案開始載入之後，就會建立 LoaderInfo 物件。LoaderInfo 物件會提供資訊，如載入進度、載入器與被載入者的 URL、媒體的位元組總和，以及媒體的表面高度和寬度。LoaderInfo 也會傳送事件，以供監視載入進度。

下圖顯示 LoaderInfo 物件的不同用法，分別針對：SWF 檔之主要類別的實體、Loader 物件，以及由 Loader 物件所載入的物件：



LoaderInfo 物件可以同時做為 Loader 物件和已載入顯示物件的屬性加以存取。載入作業一開始，LoaderInfo 物件就可以透過 Loader 物件的 contentLoaderInfo 屬性加以存取。顯示物件完成載入以後，LoaderInfo 物件也可以做為已載入顯示物件的屬性，透過顯示物件的 loaderInfo 屬性加以存取。已載入顯示物件的 loaderInfo 屬性所參考的 LoaderInfo 物件與 Loader 物件的 contentLoaderInfo 屬性相同；換句話說，已載入的物件與載入該物件的 Loader 物件 (亦即，載入器與被載入者) 會共用一個 LoaderInfo 物件。

若要存取已載入內容的屬性，就要將事件偵聽程式加入至 LoaderInfo 物件，如下列程式碼所示：

```
import flash.display.Loader;
import flash.display.Sprite;
import flash.events.Event;

var ldr:Loader = new Loader();
var urlReq:URLRequest = new URLRequest("Circle.swf");
ldr.load(urlReq);
ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, loaded);
addChild(ldr);

function loaded(event:Event):void
{
    var content:Sprite = event.target.content;
    content.scaleX = 2;
}
```

如需詳細資訊，請參閱第 108 頁「處理事件」。

指定載入內容

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

將外部檔案透過 Loader 類別的 load() 或 loadBytes() 方法載入 Flash Player 或 AIR 之中時，您可以選擇指定 context 參數，這個參數是 LoaderContext 物件。

LoaderContext 類別包括三個屬性，可以讓您定義如何使用載入內容的環境：

- checkPolicyFile：請只在載入影像檔 (而不是 SWF 檔) 時使用這個屬性。如果將這個屬性設定為 true，Loader 會檢查原則檔的原始伺服器 (請參閱第 907 頁「網站控制 (原則檔)」)。只有在內容不是源自包含 Loader 物件的 SWF 檔時，才需要

執行這項步驟。如果伺服器授予 Loader 網域權限，Loader 網域中 SWF 檔的 ActionScript 就可以存取已載入影像中的資料；換句話說，您可以使用 BitmapData.draw() 命令，存取載入影像中的資料。

請注意，來自 Loader 物件以外網域的 SWF 檔可以呼叫 Security.allowDomain()，以允許特定網域。

- securityDomain：請只在載入 SWF 檔 (而不是影像) 時使用這個屬性。為不是源自包含 Loader 物件之檔案所屬網域的 SWF 檔指定這個方法。指定這個選項時，Flash Player 會檢查是否有原則檔，如果有，則來自跨網域原則檔中所允許網域的 SWF 檔就可以對載入的 SWF 內容進行跨 Script 處理。您可以將這個參數設定為 flash.system.SecurityDomain.currentDomain。
- applicationDomain：請只在載入以 ActionScript 3.0 撰寫的 SWF 檔 (而非影像或以 ActionScript 1.0 或 2.0 撰寫的 SWF 檔) 時使用此屬性。載入檔案時，您可以透過將 applicationDomain 參數設定為 flash.system.ApplicationDomain.currentDomain，指定檔案要包含在與 Loader 物件相同的應用程式網域中。藉由將載入的 SWF 檔置於相同的應用程式網域中，便可以直接存取其類別。如果您要載入的是包含內嵌媒體 (可以透過它們的相關類別名稱進行存取) 的 SWF 檔，這種做法就很有用。如需詳細資訊，請參閱第 125 頁 「使用應用程式網域」。

在下列範例中，會示範從另一個網域載入點陣圖時，檢查原則檔的情形：

```
var context:LoaderContext = new LoaderContext();
context.checkPolicyFile = true;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/photo11.jpg");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

在下列範例中，會示範從另一個網域載入 SWF 時，檢查原則檔，以便將檔案放入與 Loader 物件相同的安全執行程序中。此外，程式碼也會將已載入 SWF 檔中的類別加入與 Loader 物件相同的應用程式網域中：

```
var context:LoaderContext = new LoaderContext();
context.securityDomain = SecurityDomain.currentDomain;
context.applicationDomain = ApplicationDomain.currentDomain;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/library.swf");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

如需詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中的 LoaderContext 類別。

在 AIR for iOS 中載入 SWF 檔案

Adobe AIR 3.6 及更新的版本、僅 **iOS**

在 iOS 裝置上，於執行階段載入並編譯程式碼時會有一些限制。由於這些限制，將外部 SWF 檔案載入應用程式的工作會有一些必然的差異：

- 所有包含 ActionScript 程式碼的 SWF 檔案必須包含在應用程式套件中。包含程式碼的 SWF 不得從外部來源 (例如透過網路) 載入。封裝應用程式期間，應用程式套件中所有 SWF 檔案內的所有 ActionScript 程式碼都會編譯為適用於 iOS 裝置的原生程式碼。
- 您無法載入再取消載入，然後重新載入 SWF 檔案。如果您嘗試執行此操作，就會發生錯誤。
- 載入再取消載入記憶體的行為，與在桌面平台中相同。如果您載入再取消載入 SWF 檔案，SWF 中的所有視覺資源都會從記憶體中取消載入。然而，載入之 SWF 中 ActionScript 類別的任何類別參照會留在記憶體中，並且可以在 ActionScript 程式碼中存取。
- 所有載入的 SWF 檔案必須在與主要 SWF 檔案相同的應用程式網域中載入。這不是預設行為，因此，對於所載入的每個 SWF，您必須建立指定主要應用程式網域的 LoaderContext 物件，並且將該 LoaderContext 物件傳遞至 Loader.load() 方法呼叫。如果您嘗試在主要 SWF 應用程式網域以外的應用程式網域中載入 SWF 檔案，就會發生錯誤。即使載入的 SWF 僅包含視覺資源而沒有 ActionScript 程式碼，也是如此。

  下列範例說明用來將 SWF 從應用程式套件載入主要 SWF 應用程式網域的程式碼：

```
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("swfs/SecondarySwf.swf");
var loaderContext:LoaderContext = new LoaderContext(false, ApplicationDomain.currentDomain, null);
loader.load(url, loaderContext);
```

僅包含資源而沒有程式碼的 SWF 檔案可以從應用程式套件或透過網路載入。在任一種情況中，SWF 檔案仍然必須載入主要應用程式網域。

對於 AIR 3.6 以前的 AIR 版本，所有程式碼會在編譯過程中，從主要應用程式 SWF 以外的 SWF 中刪除。僅包含視覺資源的 SWF 檔案可以在執行階段隨附於應用程式套件並載入，但不可以有程式碼。如果您嘗試載入包含 ActionScript 程式碼的 SWF，就會發生錯誤。此錯誤會造成「未編譯的 ActionScript」錯誤對話方塊出現在應用程式中。

請參閱

在 iOS 的 AIR 應用程式中封裝並載入多個 SWF

使用 ProLoader 與 ProLoaderInfo 類別

Flash Player 9 以及更新的版本、**Adobe AIR 1.0** 以及更新的版本，並且需要 **Flash Professional CS5.5**

為了協助預先載入執行階段共享元件庫 (RSL)，Flash Professional CS5.5 會導入 fl.display.ProLoader 和 fl.display.ProLoaderInfo 類別。這些類別會鏡像 flash.display.Loader 和 flash.display.LoaderInfo 類別，但可提供更一致性的載入經驗。

此外，ProLoader 還可協助您使用 RSL 預先載入的 Text Layout Framework (TLF)，載入 SWF 檔。在執行階段中，預先載入其他 SWF 檔或 SWZ 檔的 SWF 檔 (例如 TLF)，需要僅供內部使用的 SWF 包裝檔。由 SWF 包裝檔附加之額外圖層的複雜度，會導致不必要的行為。由於檔案均為一般的 SWF 檔，ProLoader 可解決載入這些檔案的複雜度。ProLoader 類別使用的解決方案對使用者而言一目瞭然，且無需特別在 ActionScript 中處理。此外，ProLoader 會載入正確的一般 SWF 內容。

在 Flash Professional CS 5.5 和更新的版本中，您可以更安全地以 ProLoader 類別，取代 Loader 類別的所有用法。接著，將您的應用程式匯出至 Flash Player 10.2 或更新的版本，讓 ProLoader 可存取所需的 ActionScript functionality。若您使用較舊版的 Flash Player (支援 ActionScript 3.0)，也可以使用 ProLoader。不過，使用 Flash Player 10.2 或更新的版本可讓您充分利用 ProLoader 的功能。當您在 Flash Professional CS5.5 或更新的版本中使用 TLF 時，永遠使用 ProLoader。除了 Flash Professional 以外，其他環境均無需使用 ProLoader。

重要事項：對於在 Flash Professional CS5.5 和更新的版本中發佈的 SWF 檔案，您可以一律使用 fl.display.ProLoader 和 fl.display.ProLoaderInfo 類別，不要使用 flash.display.Loader 和 flash.display.LoaderInfo。

ProLoader 類別提到的問題

ProLoader 類別提到的問題，包括無法處理 legacy Loader 類別。這些問題起因於 TLF 元件庫預先載入 RSL。特別是，套用至使用 Loader 物件來載入其他 SWF 檔案的 SWF 檔案。提到的問題如下：

- 要載入的檔案和已載入的檔案間之指令碼並非預期。ProLoader 類別會自動將要載入的 SWF 檔設定為已載入的 SWF 檔的父系。因此，要載入的 SWF 檔之通訊，會直接進入已載入的 SWF 檔。
- **SWF** 應用程式必須主動管理載入程序。如此一來，則需要實作額外事件，例如 added、removed、addedToStage 以及 removedFromStage。如果您的應用程式使用 Flash Player 10.2 或更新的版本，ProLoader 則無需進行此額外作業。

更新程式碼以使用 ProLoader 而非 Loader

因為 ProLoader 會鏡像 Loader 類別，所以，您可以輕鬆地在程式碼中切換這兩個類別。以下範例說明如何更新現有程式碼，以使用新類別：

```
import flash.display.Loader;
import flash.events.Event;
var l:Loader = new Loader();

addChild(l);
l.contentLoaderInfo.addEventListener(Event.COMPLETE, loadComplete);
l.load("my.swf");
function loadComplete(e:Event) {
    trace('load complete!');
}
```

此程式碼可以更新為使用 ProLoader，如下所示：

```
import fl.display.ProLoader;
import flash.events.Event;
var l:ProLoader = new ProLoader();

addChild(l);
l.contentLoaderInfo.addEventListener(Event.COMPLETE, loadComplete);
l.load("my.swf");
function loadComplete(e:Event) {
    trace('load complete!');
}
```

顯示物件範例：SpriteArranger

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

SpriteArranger 樣本應用程式是以 Geometric Shapes 樣本應用程式 (於 「學習 ActionScript 3.0」中另外說明) 為基礎而建立。

SpriteArranger 樣本應用程式將說明處理顯示物件的一些概念：

- 擴充顯示物件類別
- 將物件加入顯示清單
- 建立顯示物件圖層及使用顯示物件容器
- 回應顯示物件事件
- 使用顯示物件的屬性和方法

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。您可以在 Examples/SpriteArranger 檔案夾中找到 SpriteArranger 應用程式檔案，此應用程式是由下列檔案組成：

檔案	說明
SpriteArranger.mxml 或 SpriteArranger.fla	主應用程式檔案，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/SpriteArranger/CircleSprite.as	定義在螢幕上呈現圓形之 Sprite 物件類型的類別。
com/example/programmingas3/SpriteArranger/DrawingCanvas.as	定義畫布的類別，畫布是包含 GeometricSprite 物件的顯示物件容器。
com/example/programmingas3/SpriteArranger/SquareSprite.as	定義在螢幕上呈現方形之 Sprite 物件類型的類別。
com/example/programmingas3/SpriteArranger/TriangleSprite.as	定義在螢幕上呈現三角形之 Sprite 物件類型的類別。

檔案	說明
com/example/programmingas3/SpriteArranger/GeometricSprite.as	擴充 Sprite 物件的類別，用來定義螢幕上的形狀。CircleSprite、SquareSprite 和 TriangleSprite 都各會擴充此類別。
com/example/programmingas3/geometricshapes/IGeometricShape.as	要由所有幾何圖形類別實作的基底介面定義方法。
com/example/programmingas3/geometricshapes/IPolygon.as	要由具有多邊的幾何圖形類別實作的介面定義方法。
com/example/programmingas3/geometricshapes/RegularPolygon.as	一種幾何圖形，將等長的邊以對稱方式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 SpriteArranger 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

SpriteArranger 應用程式可以讓使用者將各種不同的顯示物件加入至螢幕上的「畫布」。

DrawingCanvas 類別會定義繪圖區域，是一種顯示物件容器，使用者可以在其中加入螢幕上的形狀。這些螢幕上的形狀都是 GeometricSprite 類別的其中一個子類別之實體。

DrawingCanvas 類別

在 Flex 中，所有加入 Container 物件的子顯示物件都必須是傳承自 mx.core.UIComponent 類別的類別。這個應用程式會加入 DrawingCanvas 類別的實體做為 mx.containers.VBox 物件的子系 ,，如同在 SpriteArranger.mxml 檔案的 MXML 程式碼中所定義。這項繼承是在 DrawingCanvas 類別宣告中定義，如下所示：

```
public class DrawingCanvas extends UIComponent
```

UIComponent 類別繼承自 DisplayObject、DisplayObjectContainer 和 Sprite 類別，而且 DrawingCanvas 類別中的程式碼會使用這些類別的方法和屬性。

DrawingCanvas 類別會擴充 Sprite 類別，而且這是 DrawingCanvas 類別宣告中定義的繼承，如下所示：

```
public class DrawingCanvas extends Sprite
```

Sprite 類別是 DisplayObjectContainer 及 DisplayObject 類別的子類別，而且 DrawingCanvas 類別會使用這些類別的方法和屬性。

DrawingCanvas() 建構函式方法會設定 Rectangle 物件 bounds (這是稍後要用在畫布外框繪圖中的屬性)，然後再呼叫 initCanvas() 方法，如下所示：

```
this.bounds = new Rectangle(0, 0, w, h);
initCanvas(fillColor, lineColor);
```

如下列範例所示，initCanvas() 方法會定義 DrawingCanvas 物件的各種屬性，傳遞給建構函式做為引數：

```
this.lineColor = lineColor;
this.fillColor = fillColor;
this.width = 500;
this.height = 200;
```

然後 initCanvas() 方法會再呼叫 drawBounds() 方法，在畫布上使用 DrawingCanvas 類別的 graphics 屬性繪圖。graphics 屬性是繼承自 Shape 類別。

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
this.graphics.drawRect(bounds.left - 1,
                       bounds.top - 1,
                       bounds.width + 2,
                       bounds.height + 2);
this.graphics.endFill();
```

下列 DrawingCanvas 類別的其它方法是根據使用者與應用程式的互動而叫用：

- addShape() 和 describeChildren() 方法，請參閱第 173 頁「將顯示物件加入畫布」
- moveToBack()、moveDown()、moveToFront() 和 moveUp() 方法，請參閱第 175 頁「重新安排顯示物件圖層」
- onMouseUp() 方法，請參閱第 175 頁「點選及拖曳顯示物件」

GeometricSprite 類別及其子類別

使用者可以加入至畫布的每一個顯示物件都是下列 GeometricSprite 類別其中一個子類別的實體：

- CircleSprite
- SquareSprite
- TriangleSprite

GeometricSprite 類別會擴充 flash.display.Sprite 類別：

```
public class GeometricSprite extends Sprite
```

GeometricSprite 類別包含一些所有 GeometricSprite 物件都共通的屬性。這些都是根據傳遞給函數的參數，在建構函式中設定。例如：

```
this.size = size;
this.lineColor = lColor;
this.fillColor = fColor;
```

GeometricSprite 類別的 geometricShape 屬性會定義 IGeometricShape 介面，此介面則會定義形狀的數學屬性，而不是視覺屬性。實作 IGeometricShape 介面的類別定義於「學習 ActionScript 3.0」說明的 GeometricShapes 樣本應用程式。

GeometricSprite 類別會定義 drawShape() 方法，再進一步於 GeometricSprite 的各個子類別之覆寫定義中修改。如需詳細資訊，請參閱稍後說明的「將顯示物件加入畫布」一節。

GeometricSprite 類別也提供下列方法：

- onMouseDown() 和 onMouseUp() 方法，請參閱第 175 頁「點選及拖曳顯示物件」
- showSelected() 和 hideSelected() 方法，請參閱第 175 頁「點選及拖曳顯示物件」

將顯示物件加入畫布

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當使用者按一下 Add Shape 按鈕時，應用程式會呼叫 DrawingCanvas 類別的 addShape() 方法。它會呼叫 GeometricSprite 子類別的其中一個適當建構函式，實體化新的 GeometricSprite，如下列範例所示：

```
public function addShape(shapeName:String, len:Number):void
{
    var newShape:GeometricSprite;
    switch (shapeName)
    {
        case "Triangle":
            newShape = new TriangleSprite(len);
            break;

        case "Square":
            newShape = new SquareSprite(len);
            break;

        case "Circle":
            newShape = new CircleSprite(len);
            break;
    }
    newShape.alpha = 0.8;
    this.addChild(newShape);
}
```

每一個建構函式方法都會呼叫 drawShape() 方法，使用類別的 graphics 屬性 (繼承自 Sprite 類別)，繪製適當的向量圖形。例如，CircleSprite 類別的 drawShape() 方法包含下列程式碼：

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
var radius:Number = this.size / 2;
this.graphics.drawCircle(radius, radius, radius);
```

addShape() 函數的倒數第二行會設定顯示物件的 alpha 屬性 (自 DisplayObject 類別繼承)，以便讓加入畫布的每一個顯示物件都有些微透明，讓使用者能夠看到背後的物件。

addChild() 方法的最後一行會將新的顯示物件加入 DrawingCanvas 類別實體的子清單中，它本來就已經在顯示清單上，如此會讓新的顯示物件出現在「舞台」上。

應用程式的介面包括兩個文字欄位，selectedSpriteTxt 和 outputTxt。這兩個文字欄位的 text 屬性是以有關 (加入至畫布或由使用者選取之) GeometricSprite 物件的資訊加以更新。GeometricSprite 類別是透過覆寫 toString() 方法，處理此資訊報告工作，如下所示：

```
public override function toString():String
{
    return this.shapeType + " of size " + this.size + " at " + this.x + ", " + this.y;
}
```

shapeType 屬性會在每個 GeometricSprite 子類別的建構函式方法中設定為適當的值。例如，toString() 方法可能會傳回下列最近加入 DrawingCanvas 實體之 CircleSprite 實體的值：

```
Circle of size 50 at 0, 0
```

DrawingCanvas 類別的 describeChildren() 方法會使用 numChildren 屬性 (繼承自 DisplayObjectContainer 類別) 設定 for 迴圈的限制，循序處理畫布的子清單。它會產生列出各子系的字串，如下所示：

```
var desc:String = "";
var child:DisplayObject;
for (var i:int=0; i < this.numChildren; i++)
{
child = this.getChildAt(i);
desc += i + ": " + child + '\n';
}
```

所產生的結果字串是用來設定 outputTxt 文字欄位的 text 屬性。

點選及拖曳顯示物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當使用者在 GeometricSprite 實體上按一下時，應用程式會呼叫 onMouseDown() 事件處理常式。如下所示，此事件處理常式是在 GeometricSprite 類別的建構函式中設定為按一下滑鼠事件的偵聽程式：

```
this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

然後 onMouseDown() 方法再呼叫 GeometricSprite 物件的 showSelected() 方法。如果是初次為物件呼叫此方法，則此方法會建立名為 selectionIndicator 的新 Shape 物件，並使用 Shape 物件的 graphics 屬性，繪製紅色的反白標示矩形，如下所示：

```
this.selectionIndicator = new Shape();
this.selectionIndicator.graphics.lineStyle(1.0, 0xFF0000, 1.0);
this.selectionIndicator.graphics.drawRect(-1, -1, this.size + 1, this.size + 1);
this.addChild(this.selectionIndicator);
```

如果不是初次呼叫 onMouseDown() 方法，此方法會直接設定 selectionIndicator 形狀的 visible 屬性 (繼承自 DisplayObject 類別)，如下所示：

```
this.selectionIndicator.visible = true;
```

hideSelected() 方法會將其 visible 屬性設定為 false，隱藏先前所選取物件的 selectionIndicator 形狀。

onMouseDown() 事件處理常式也會呼叫 startDrag() 方法 (繼承自 Sprite 類別)，其中包含下列程式碼：

```
var boundsRect:Rectangle = this.parent.getRect(this.parent);
boundsRect.width -= this.size;
boundsRect.height -= this.size;
this.startDrag(false, boundsRect);
```

如此可讓使用者在畫布上於 boundsRect 矩形設定的範圍內，四處拖曳物件。

當使用者放開滑鼠按鍵時，即傳送 mouseUp 事件。DrawingCanvas 的建構函式方法會設置下列事件偵聽程式：

```
this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
```

這個事件偵聽程式是為 DrawingCanvas 物件設定，而不是為各個 GeometricSprite 物件設定的。這是因為拖曳 GeometricSprite 物件時，如果放開滑鼠，此物件可能會跑到另外一個顯示物件 (另外一個 GeometricSprite 物件) 的背後；使得在前景中的顯示物件接到放開滑鼠事件，而使用者拖曳的顯示物件反而接不到這個事件。將偵聽程式加入至 DrawingCanvas 物件可確保永遠都會處理該事件。

onMouseUp() 方法會呼叫 GeometricSprite 物件的 onMouseUp() 方法，此方法接著再轉而呼叫 GeometricSprite 物件的 stopDrag() 方法。

重新安排顯示物件圖層

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

應用程式的使用者介面包含標示為 「向後」、「向下」、「向上」和 「向前」等按鈕。當使用者按一下上述其中一個按鈕時，應用程式就會呼叫相對應的 DrawingCanvas 類別方法：moveToBack()、moveDown()、moveUp() 或 moveToFront()。例如，moveToBack() 方法包含下列程式碼：

```
public function moveToBack(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, 0);
    }
}
```

此方法會使用 setChildIndex() 方法 (繼承自 DisplayObjectContainer 類別)，將顯示物件放置在 DrawingCanvas 實體 (this) 之子清單中的索引位置 0。

moveDown() 方法也以類似方式運作，只不過它是在 DrawingCanvas 實體的子清單中，將顯示物件的索引位置以 1 遞減：

```
public function moveDown(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, index - 1);
    }
}
```

moveUp() 和 moveToFront() 方法的運作方式與 moveToBack() 和 moveDown() 方法相同。

第 11 章 處理幾何

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

flash.geom 套件內含數個類別，可定義像是點、矩形及變形矩陣這類幾何物件。您可以使用這些類別，定義在其它類別中使用之物件的屬性。

更多說明主題
flash.geom 套件

幾何的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

flash.geom 套件內含數個類別，可定義像是點、矩形及變形矩陣這類幾何物件。這些類別本身不一定會提供一些功能；然而，卻可透過這些類別來定義物件屬性，以便運用在其它類別上。

所有的幾何類別都是基於螢幕上的位置是由二維平面所代表的觀念來運作。螢幕將被視為帶有水平 (x) 軸與垂直 (y) 軸的平面圖表。螢幕上的任何位置 (或 「點」) 將以一對 (x, y) 值，也就是該位置的 「座標」來代表。

每一個顯示物件 (包括 「舞台」)，都有自己的 「座標空間」。座標空間是物件的自屬圖形，用於繪製子顯示物件、繪圖等等的位置。「原點」位於座標位置 0, 0 (即 x 和 y 軸交叉處)，而且是放在顯示物件的左上角。雖然這個原點對「舞台」來說永遠適用，但是對其它顯示物件卻不一定適用。x 軸的值越往右會變得越大，而越往左會變得越小。至於原點左方的位置，x 座標為負值。不過，和傳統座標系統相反的是，y 軸上的 Flash 執行階段座標值越往螢幕下方就變得越大，而越往螢幕上方則變得越小。原點上方的值的 y 座標值為負數)。因為舞台左上角是其座標空間的原點，所以「舞台」上大部分物件的 x 座標都大於 0，而且比 「舞台」寬度小。而且同一個物件的 y 座標大於 0，而且比 「舞台」高度小。

您可以使用 Point 類別實體來代表座標空間中的個別座標點。您可以建立 Rectangle 實體來代表座標空間中的矩形區域。如果您是進階使用者，可以使用 Matrix 實體，將多個或複雜的變形套用到顯示物件。許多簡單的變形，例如旋轉、位置與縮放變更，都可以使用物件的屬性，直接套用至該顯示物件。如需有關使用顯示物件屬性套用變形的詳細資訊，請參閱第 146 頁 「操作顯示物件」。

重要概念與術語
下列參考清單包含重要幾何術語：

迪卡兒座標 座標一般都是以成對的數字來表示 (像是 5, 12 或 17, -23)。這兩個數字分別為 x 座標與 y 座標。

座標空間 包含在顯示物件中的座標圖表，且圖表上放置了該物件的子元素。

原點 座標空間中 x 軸與 y 軸交叉的那一點。此點的座標為 0, 0。

點 座標空間中的單一位置。在 ActionScript 使用的二維座標系統中，沿著 x 軸和 y 軸的位置 (點的座標) 會定義這一點。

註冊點 在顯示物件中，座標空間的原點 (0, 0 座標)。

縮放 物件大小 (相對於原始大小)。當做為動詞使用時，縮放物件表示藉由延伸或縮小物件來變更其尺寸。

轉移 將點的座標從某個座標空間變換至另一個座標空間。

變形 調整圖像的視覺特性，如旋轉物件、改變比例、傾斜或扭曲物件形狀或改變顏色。

X 軸 ActionScript 中所使用的二維系統之水平軸。

Y 軸 ActionScript 中所使用的二維系統之垂直軸。

使用 Point 物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Point 物件會定義一對迪卡兒座標。它代表二維座標系統的位置，其中 x 代表水平軸，而 y 代表垂直軸。

若要定義 Point 物件，您必須設定它的 x 和 y 屬性，如下所示：

```
import flash.geom.*;
var pt1:Point = new Point(10, 20); // x == 10; y == 20
var pt2:Point = new Point();
pt2.x = 10;
pt2.y = 20;
```

找出兩點之間的距離

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 Point 類別的 distance() 方法，找出座標空間中兩點之間的距離。例如，下列程式碼會在相同的顯示物件容器中，找出 circle1 和 circle2 這兩個顯示物件的註冊點之間的距離：

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
var pt2:Point = new Point(circle2.x, circle2.y);
var distance:Number = Point.distance(pt1, pt2);
```

轉移座標空間

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如果兩個顯示物件位於不同的顯示物件容器中，表示它們可能位於不同的座標空間中。您可以使用 DisplayObject 類別的 localToGlobal() 方法，將不同的座標轉移為 「舞台」上相同 (全域) 的座標空間。例如，下列程式碼會在不同的顯示物件容器中，找出 circle1 和 circle2 這兩個顯示物件的註冊點之間的距離：

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
pt1 = circle1.localToGlobal(pt1);
var pt2:Point = new Point(circle2.x, circle2.y);
pt2 = circle2.localToGlobal(pt2);
var distance:Number = Point.distance(pt1, pt2);
```

同樣地，若要找出名為 target 之顯示物件的註冊點與 「舞台」上特定點之間的距離，請使用 DisplayObject 類別的 localToGlobal() 方法：

```
import flash.geom.*;
var stageCenter:Point = new Point();
stageCenter.x = this.stage.stageWidth / 2;
stageCenter.y = this.stage.stageHeight / 2;
var targetCenter:Point = new Point(target.x, target.y);
targetCenter = target.localToGlobal(targetCenter);
var distance:Number = Point.distance(stageCenter, targetCenter);
```

依指定的角度與距離移動顯示物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 Point 類別的 polar() 方法，依特定的角度將顯示物件移動特定的距離。例如，下列程式碼會將 myDisplayObject 物件以 60° 移動 100 個像素：

```
import flash.geom.*;
var distance:Number = 100;
var angle:Number = 2 * Math.PI * (90 / 360);
var translatePoint:Point = Point.polar(distance, angle);
myDisplayObject.x += translatePoint.x;
myDisplayObject.y += translatePoint.y;
```

Point 類別的其它用法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以搭配下列方法與屬性使用 Point 物件：

類別	方法或屬性	說明
DisplayObjectContainer	areInaccessibleObjectsUnderPoint()getObjectsUnderPoint()	用來傳回顯示物件容器中某一點下的物件清單。
BitmapData	hitTest()	用來定義 BitmapData 物件中的像素，以及定義要進行碰撞測試的點。
BitmapData	applyFilter() copyChannel() merge() paletteMap() pixelDissolve() threshold()	用來定義矩形的位置，這些矩形都會定義作業。
矩陣	deltaTransformPoint() transformPoint()	用來定義您要針對其套用變形的點。
Rectangle	bottomRight size topLeft	用來定義這些屬性。

使用 Rectangle 物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Rectangle 物件會定義矩形區域，它的位置是由本身左上角的 x 和 y 座標，以及 width 和 height 屬性所定義。您可以呼叫 Rectangle() 建構函式，定義新 Rectangle 物件的這些屬性，如下所示：

```
import flash.geom.Rectangle;
var rx:Number = 0;
var ry:Number = 0;
var rwidth:Number = 100;
var rheight:Number = 50;
var rect1:Rectangle = new Rectangle(rx, ry, rwidth, rheight);
```

調整 Rectangle 物件的大小及位置

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用多種方式來調整 Rectangle 物件的大小及位置。

您可以直接變更 Rectangle 物件的 x 和 y 屬性，來調整該物件的位置。此變更並不會影響 Rectangle 物件的寬度或高度。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.x = 20;
rect1.y = 30;
trace(rect1); // (x=20, y=30, w=100, h=50)
```

顯示以下程式碼時，在您變更 Rectangle 物件的 left 或 top 屬性後，就會重新定位矩形。矩形的 x 和 y 屬性分別符合 left 和 top 屬性。但是，由於 Rectangle 物件的左下角位置不會更改，因此它的大小會改變：

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.left = 20;
rect1.top = 30;
trace(rect1); // (x=20, y=30, w=80, h=20)
```

同樣地，如下列範例所示，若您變更 Rectangle 物件的 bottom 或 right 屬性，它的左上角位置不會更改。矩形會隨之調整大小。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.right = 60;
trect1.bottom = 20;
trace(rect1); // (x=0, y=0, w=60, h=20)
```

您也可以使用 offset() 方法，重新調整 Rectangle 物件的位置，如下所示：

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.offset(20, 30);
trace(rect1); // (x=20, y=30, w=100, h=50)
```

offsetPt() 方法具有相似的運作方式，不過它會使用 Point 物件做為它的參數，而非使用 x 和 y 偏移值。

您也可以使用 inflate() 方法 (包含 dx 和 dy 兩個參數)，調整 Rectangle 物件的大小。dx 參數代表像素的數目，也就是矩形從中心點往左方或右方移動的距離。dy 參數代表像素的數目，也就是矩形從中心點往上方或下方移動的距離。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.inflate(6,4);
trace(rect1); // (x=-6, y=-4, w=112, h=58)
```

inflatePt() 方法也具有相似的運作方式，不過它會使用 Point 物件做為它的參數，而非使用 dx 和 dy 值。

找出 Rectangle 物件的聯集與交集

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 union() 方法，找出兩個矩形邊界所聯合形成的矩形區域：

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(120, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.union(rect2)); // (x=0, y=0, w=220, h=160)
```

您可以使用 intersection() 方法，找出兩個矩形互相重疊所形成的矩形區域：

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(80, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.intersection(rect2)); // (x=80, y=60, w=20, h=40)
```

您可以使用 intersects() 方法，得知兩個矩形是否互相交集。您也可以使用 intersects() 方法，得知某個顯示物件是否位於 「舞台」的特定區域中。至於以下的程式碼範例，假設顯示物件容器 (其中包含 circle 物件) 的座標空間與 「舞台」的座標空間相同。此範例將示範如何使用 intersects() 方法，判斷顯示物件 circle 是否與 「舞台」上 target1 和 target2 Rectangle 物件所指定的區域有交集：

```
import flash.display.*;
import flash.geom.Rectangle;
var circle:Shape = new Shape();
circle.graphics.lineStyle(2, 0xFF0000);
circle.graphics.drawCircle(250, 250, 100);
addChild(circle);
var circleBounds:Rectangle = circle.getBounds(stage);
var target1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(circleBounds.intersects(target1)); // false
var target2:Rectangle = new Rectangle(0, 0, 300, 300);
trace(circleBounds.intersects(target2)); // true
```

同樣地，您可以使用 intersects() 方法，得知兩個顯示物件的矩形邊界是否重疊。請使用 DisplayObject 類別的 getRect() 方法，將其它任何空間 (顯示物件的筆畫會將此空間加入至區域邊界) 納入。

Rectangle 物件的其它用法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列方法和屬性都會用到 Rectangle 物件：

類別	方法或屬性	說明
BitmapData	applyFilter()、colorTransform()、copyChannel()、copyPixels()、draw()、drawWithQuality()、encode()、fillRect()、generateFilterRect()、getColorBoundsRect()、getPixels()、merge()、paletteMap()、pixelDissolve(), setPixels() 以及 threshold()	當做某些參數的類型來使用，以定義 BitmapData 物件的區域。
DisplayObject	getBounds()、getRect()、scrollRect、scale9Grid	當做屬性的資料類型或是傳回的資料類型來使用。
PrintJob	addPage()	用來定義 printArea 參數。
Sprite	startDrag()	用來定義 bounds 參數。
TextField	getCharBoundaries()	當做傳回值類型來使用。
Transform	pixelBounds	當做資料類型來使用。

使用 Matrix 物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Matrix 類別代表變形矩陣，可決定如何從一個座標空間，將各點對應到另一個空間。您可以透過設定 Matrix 物件的屬性、將該 Matrix 物件套用至 Transform 物件的 matrix 屬性，然後將該 Transform 物件套用成顯示物件的 transform 屬性，藉以在顯示物件上執行各種圖像轉換。這些轉換函數包括轉移 (x 及 y 重新定位)、旋轉、縮放及傾斜。

雖然您可以透過直接調整 Matrix 物件之屬性 (a、b、c、d、tx、ty) 的方式定義矩陣，但是使用 createBox() 方法則會更加容易。這個方法包含可讓您直接定義所產生矩陣之縮放、旋轉與轉移效果的參數。例如，下列程式碼會建立 Matrix 物件，此物件具有水平縮放係數 2.0、垂直縮放係數 3.0、旋轉 45°，以及向右移動 (轉移) 10 個像素並向下移動 20 個像素的效果：

```
var matrix:Matrix = new Matrix();
var scaleX:Number = 2.0;
var scaleY:Number = 3.0;
var rotation:Number = 2 * Math.PI * (45 / 360);
var tx:Number = 10;
var ty:Number = 20;
matrix.createBox(scaleX, scaleY, rotation, tx, ty);
```

您也可以使用 scale()、rotate() 與 translate() 方法，調整 Matrix 物件的縮放、旋轉及轉譯效果。請注意，這些方法必須搭配現有之 Matrix 物件的值來使用。例如，下列程式碼會設定 Matrix 物件，因為呼叫 scale() 和 rotate() 方法兩次的緣故，所以此物件會以 4 為係數縮放物件，並將其旋轉 60°：

```
var matrix:Matrix = new Matrix();
var rotation:Number = 2 * Math.PI * (30 / 360); // 30°
var scaleFactor:Number = 2;
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);
matrix.scale(scaleX, scaleY);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

若要將傾斜變形套用至 Matrix 物件，請調整該物件的 b 或 c 屬性。其中，調整 b 屬性會垂直傾斜矩陣，而調整 c 屬性則會水平傾斜矩陣。下列程式碼會以 2 為係數，垂直傾斜 myMatrix Matrix 物件：

```
var skewMatrix:Matrix = new Matrix();
skewMatrix.b = Math.tan(2);
myMatrix.concat(skewMatrix);
```

您可以將 Matrix 變形套用至顯示物件的 transform 屬性。例如，下列程式碼會將矩陣變形套用至名為 myDisplayObject 的顯示物件：

```
var matrix:Matrix = myDisplayObject.transform.matrix;
var scaleFactor:Number = 2;
var rotation:Number = 2 * Math.PI * (60 / 360); // 60°
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

第一行會將 Matrix 物件設定為 myDisplayObject 顯示物件所使用的現有變形矩陣 (myDisplayObject 顯示物件之 transformation 屬性的 matrix 屬性)。如此一來，您所呼叫的 Matrix 類別方法便會在顯示物件的現有位置呈現縮放與旋轉的累加效果。

備註：flash.geometry 套件中也包含 ColorTransform 類別，它是用來設定 Transform 物件的 colorTransform 屬性。由於此類別不會套用任何幾何變形，所以在此處不討論細節。如需詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中的 ColorTransform 類別。

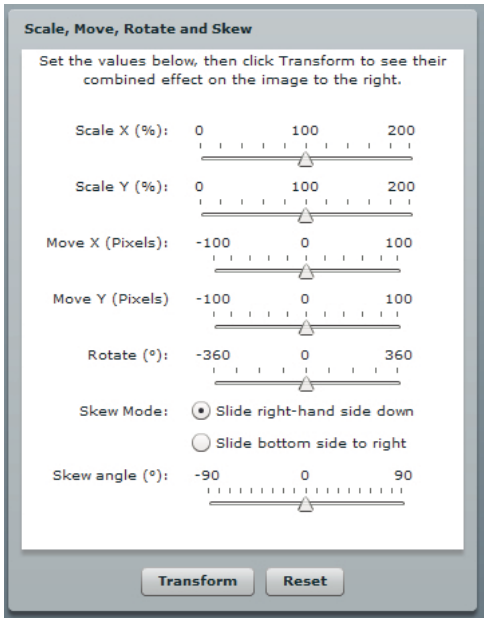
幾何範例：將矩陣變形套用至顯示物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

DisplayObjectTransformer 樣本應用程式會示範使用 Matrix 類別進行顯示物件之變形作業的數項功能，其中包括：

- 旋轉顯示物件
- 縮放顯示物件
- 轉譯 (重新定位) 顯示物件
- 傾斜顯示物件

此應用程式可提供介面，用來調整矩陣變形的參數，如下所示：



當使用者按一下 「變形」按鈕時，應用程式便會套用正確的變形作業。

原始的顯示物件，以及旋轉 -45° 並縮放 50% 之後的顯示物件

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。您可以在 Samples/DisplayObjectTransformer 檔案夾中找到 DisplayObjectTransformer 應用程式檔案，此應用程式是由下列檔案組成：

檔案	說明
DisplayObjectTransformer.mxml 或 DisplayObjectTransformer.fla	主應用程式檔案，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/geometry/MatrixTransformer.as	類別，其中包含要用來套用矩陣變形的方法。
img/	目錄，其中包含此應用程式所使用的樣本影像檔。

定義 MatrixTransformer 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

MatrixTransformer 類別包含靜態方法，可套用 Matrix 物件的幾何變形。

transform() 方法

transform() 方法包含下列每一個項目的參數：

- sourceMatrix — 輸入矩陣，也就是此方法要加以變形的矩陣
- xScale 和 yScale — x 和 y 縮放係數
- dx 和 dy — x 和 y 轉譯的像素量
- rotation — 旋轉的角度量
- skew — 傾斜係數 (百分比)
- skewType — 傾斜方向，為 "right" 或 "left"

傳回值為所產生的矩陣。

transform() 方法會呼叫此類別的下列靜態方法：

- skew()
- scale()
- translate()
- rotate()

其中每一個都會傳回已套用變形的來源矩陣。

skew() 方法

skew() 方法會調整矩陣的 b 和 c 屬性，藉以傾斜矩陣。選擇性參數 unit 會判斷用來定義傾斜角度的單位。如有必要，這個方法會將 angle 值轉換為弧度：

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
```

建立 skewMatrix Matrix 物件並加以調整，以便套用傾斜變形。一開始，它是一個單位矩陣，如下所示：

```
var skewMatrix:Matrix = new Matrix();
```

skewSide 參數會決定是哪一邊套用了傾斜變形。如果將它設定為 "right"，下列程式碼就會設定該矩陣的 b 屬性：

```
skewMatrix.b = Math.tan(angle);
```

否則，就會調整 Matrix 的 c 屬性來傾斜底端，如下所示：

```
skewMatrix.c = Math.tan(angle);
```

接著這兩個矩陣就會結合，將產生的傾斜變形套用至現有的矩陣，如下列範例所示：

```
sourceMatrix.concat(skewMatrix);
return sourceMatrix;
```

scale() 方法

如下列範例所示，如果所提供的縮放係數為百分比，則 scale() 方法首先會調整此係數，然後再使用此矩陣物件的 scale() 方法：

```
if (percent)
{
    xScale = xScale / 100;
    yScale = yScale / 100;
}
sourceMatrix.scale(xScale, yScale);
return sourceMatrix;
```

translate() 方法

translate() 方法只會藉由呼叫此矩陣物件的 translate() 方法來套用 dx 和 dy 轉譯係數，如下所示：

```
sourceMatrix.translate(dx, dy);
return sourceMatrix;
```

rotate() 方法

rotate() 方法會將輸入旋轉係數轉換為弧度 (如果所提供的係數為角度或斜率)，然後呼叫此矩陣物件的 rotate() 方法：

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
sourceMatrix.rotate(angle);
return sourceMatrix;
```

從應用程式呼叫 MatrixTransformer.transform() 方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

此應用程式包含使用者介面，可用來從使用者處取得變形參數。然後，再將這些值以及顯示物件之 transform 屬性的 matrix 屬性傳遞給 Matrix.transform() 方法，如下所示：

```
tempMatrix = MatrixTransformer.transform(tempMatrix,
    xScaleSlider.value,
    yScaleSlider.value,
    dxSlider.value,
    dySlider.value,
    rotationSlider.value,
    skewSlider.value,
    skewSide );
```

接下來，此應用程式會將傳回值套用至顯示物件之 transform 屬性的 matrix 屬性，觸發變形作業：

```
img.content.transform.matrix = tempMatrix;
```

第 12 章 使用繪圖 API

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

雖然匯入的影像與圖像很重要，您還是可以透過稱為繪圖 API 的功能在 ActionScript 中繪製線段與形狀，在電腦中自由地開啟包含相當於空白畫布的應用程式，盡情地創造想像中的影像。這項可建立專屬圖像的功能為您的應用程式開拓了無限的可能性。您可以運用此處介紹的技巧，建立繪圖程式、創作互動式動畫，或是透過程式設計方式建立專屬的使用者介面元素，並探索其他更多的發展潛能。

更多說明主題
flash.display.Graphics

繪圖 API 的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

「繪圖 API」這個 ActionScript 的內建功能可讓您建立向量圖像，包括線段、曲線、形狀、填色與漸層等，並使用 ActionScript 將這些圖像顯示在螢幕上。flash.display.Graphics 類別可提供此功能。您可以在任何 Shape、Sprite，或 MovieClip 實體上使用 ActionScript 來繪圖，方法是使用這些類別所個別定義的 graphics 屬性 (這些類別的個別 graphics 屬性事實上都是 Graphics 類別的實體)。

如果您剛開始使用程式碼來繪圖，可利用 Graphics 類別所包含的數種方法輕鬆地繪製像是圓形、橢圓形、矩形，以及圓角矩形之類的常見形狀。您可以將這些形狀繪製成空白線段或是填色形狀。當您需要更進階的功能時，Graphics 類別同樣包含了一些可用來繪製線段與二次方貝茲曲線的方法，方便您搭配 Math 類別中的三角函數建立所需要的任何形狀。

Flash 執行階段 (例如 Flash Player 10 和 Adobe AIR 1.5 及更新版本) 中另外加入了供繪圖使用的 API，可讓您以程式設計方式，透過單一命令來繪製整個形狀。當您熟悉 Graphics 類別和 「使用繪圖 API 基本課程」中涵蓋的工作之後，請參閱 第 198 頁 「繪圖 API 的進階用法」，進一步瞭解這些繪圖 API 功能。

重要概念與術語
下列參考清單包含使用繪圖 API 時會碰到的重要術語：

錨點 在二次方貝茲曲線中，兩個端點的其中一個。

控制點 定義二次方貝茲曲線之曲線方向與數量的點。曲線永遠無法連接控制點；然而，曲線的繪製方式看起來就像是朝著控制點走。

座標空間 包含在顯示物件中的座標圖表，且圖表上放置了該物件的子元素。

填色 在形狀內部的實心部分填入了有顏色的線段，或是沒有外框的整個形狀。

漸層 一種包含由某種顏色逐漸轉換為另一個或其它多種顏色的顏色空間 (與純色不同)。

點 座標空間中的單一位置。在 ActionScript 所使用的二維座標系統中，沿著 x 軸與 y 軸的位置可用來定義點 (點的座標)。

二次方貝茲曲線 由特殊數學公式定義的曲線類型。在這種曲線類型中，曲線的形狀是依據錨點 (曲線的兩個端點) 位置以及用來定義曲線數量與方向的控制點所計算出來的。

縮放 物件大小 (相對於原始大小)。當做為動詞使用時，縮放物件表示藉由延伸或縮小物件來變更其尺寸。

Stroke 形狀的外框部分 (其中包含填上顏色的線段)，或是未填上顏色的形狀線段。

轉移 將點的座標從某個座標空間變換至另一個座標空間。

X 軸 ActionScript 中所使用的二維系統之水平軸。

Y 軸 ActionScript 中所使用的二維系統之垂直軸。

Graphics 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

每個 Shape、Sprite 與 MovieClip 物件都具有 graphics 屬性，這個屬性是 Graphics 類別的實體。Graphics 類別包含一些可用來繪製線段、填色與形狀的屬性和方法。如果您希望讓顯示物件單純當做繪製內容的畫布來使用，就可以使用 Shape 實體。Shape 實體在繪圖效果上比其它顯示物件來得好，因為它不像 Sprite 與 MovieClip 類別還擔負執行其它功能的責任。如果您想要一個不只可用來繪製圖像內容的顯示物件，同時希望該物件也包含其它顯示物件的話，可以使用 Sprite 實體。如需有關決定各種工作所需不同顯示物件的詳細資訊，請參閱第 145 頁「選擇 DisplayObject 子類別」。

繪製線段和曲線

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您使用 Graphics 實體來繪圖時，將一律依據線段與曲線的基本繪圖概念來完成。如此一來，所有的 ActionScript 繪圖工作都必須透過下列相同的步驟順序來完成：

- 定義線段與填色樣式
- 設定初始繪圖位置
- 繪製線段、曲線與形狀 (或是移動繪圖點)
- 必要時，完成建立填色工作

定義線段與填色樣式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要使用 Shape、Sprite 或 MovieClip 實體的 graphics 屬性來繪圖，首先必須先定義繪圖時要使用的樣式 (線段大小與顏色、填色顏色)。就像您使用 Adobe® Flash® Professional 或其它繪圖應用程式中的繪圖工具一樣，當您使用 ActionScript 繪圖時，您可以選擇要不要加上筆畫以及填色顏色。您可以使用 lineStyle() 或 lineGradientStyle() 方法來指定筆畫外觀。若要建立實線，請使用 lineStyle() 方法。呼叫此方法時，最常指定的值是前三個參數：線段粗細、顏色和 Alpha。例如，這行程式碼會告訴名為 myShape 的 Shape 來繪製粗細為 2 個像素、紅色 (0x990000)，以及 75% 不透明度的線段：

```
myShape.graphics.lineStyle(2, 0x990000, .75);
```

Alpha 參數的預設值為 1.0 (100%)，因此假如您需要完全不透明的線段，可以捨棄使用此參數。lineStyle() 方法還可接受其他兩個用於像素提示與縮放模式的參數：如需有關使用這些參數的詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中 Graphics.lineStyle() 方法的說明。

若要建立漸層線段，請使用 lineGradientStyle() 方法。第 191 頁「建立漸層線段與填色」中將詳細介紹這個方法。

如果您想要建立填色形狀，可以在開始繪圖之前，先呼叫 beginFill()、beginGradientFill()、beginBitmapFill() 或 beginShaderFill() 方法。其中最基本的方法是 beginFill()，它接受兩個參數：填色顏色和 (選擇性) 填色顏色的 Alpha 值。例如，如果您想要用一個純綠色填色的形狀來繪圖，就要使用下列程式碼 (假定您正在繪製名為 myShape 的物件)：

```
myShape.graphics.beginFill(0x00FF00);
```

在開始新的填色工作之前，呼叫任何填色方法會暗中結束先前的任何填色工作。呼叫任何可指定筆畫樣式的方法會取代上一個筆畫，但是不會改變先前指定的填色，反之亦然。

一旦您指定了線段樣式與填色屬性，下一步就是指出繪圖的開始點。Graphics 實體具有一個繪圖點，就像紙張上的鋼筆筆尖。不管繪圖點的位置在哪，那個位置就是下一個繪圖動作開始的地方。一開始，Graphics 物件會將繪圖點放在物件座標空間的 0,0 繪圖位置上。若要在不同點開始繪圖，則必須先呼叫 moveTo() 方法，再呼叫其中一個繪圖方法。這種方式就好比將鋼筆筆尖從紙張上提起，然後放到新的位置一樣。

隨著繪圖點就定位，您就可以連續呼叫 lineTo() (用於繪製直線) 與 curveTo() (用於繪製曲線) 等繪圖方法來開始繪圖。

在繪圖過程中，您可以隨時呼叫 moveTo() 方法，停止繪圖，並將繪圖點移至新的位置。

在繪圖過程中，如果已經指定填色顏色，則可以呼叫 endFill() 方法來封閉填色。如果您尚未繪製封閉形狀 (也就是說，如果當您呼叫 endFill() 時，繪圖點不是位於形狀的開始點)，則當您呼叫 endFill() 方法時，Flash 執行階段會從目前的繪圖點繪製一條直線到最近一次 moveTo() 呼叫所指定的位置，自動將形狀封閉起來。如果您已經開始填色，但尚未呼叫 endFill()，則呼叫 beginFill() (或其它任何一種填色方法) 會封閉目前的填色工作，並開始新的填色工作。

繪製直線

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您呼叫 lineTo() 方法，Graphics 物件會從目前的繪圖點繪製一條直線到您在方法呼叫中指定做為兩個參數的座標為止，並使用您所指定的線段樣式來繪製直線。例如，這一行程式碼會將繪圖點放在 100, 100 的點座標，然後繪製一條到達 200, 200 點座標的直線：

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.lineTo(200, 200);
```

下列範例將以 100 像素的高度繪製紅色和綠色的三角形：

```
var triangleHeight:uint = 100;
var triangle:Shape = new Shape();

// red triangle, starting at point 0, 0
triangle.graphics.beginFill(0xFF0000);
triangle.graphics.moveTo(triangleHeight / 2, 0);
triangle.graphics.lineTo(triangleHeight, triangleHeight);
triangle.graphics.lineTo(0, triangleHeight);
triangle.graphics.lineTo(triangleHeight / 2, 0);

// green triangle, starting at point 200, 0
triangle.graphics.beginFill(0x00FF00);
triangle.graphics.moveTo(200 + triangleHeight / 2, 0);
triangle.graphics.lineTo(200 + triangleHeight, triangleHeight);
triangle.graphics.lineTo(200, triangleHeight);
triangle.graphics.lineTo(200 + triangleHeight / 2, 0);

this.addChild(triangle);
```

繪製曲線

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

curveTo() 方法可繪製一個二次方貝茲曲線。這個方法所繪製的弧形不只能夠連接兩個點 (稱為錨點)，還能同時折彎線段以連接第三個點 (稱為控制點)。Graphics 物件會使用目前的繪圖位置做為第一個錨點。呼叫 curveTo() 方法時，您可以傳遞四個參數：控制點的 x 和 y 座標，後面接著第二個錨點的 x 和 y 座標。例如，下列程式碼會於 100, 100 點座標開始繪製曲線，然後在 200, 200 點座標結束繪圖。由於控制點位於 175, 125 的點座標位置，如此一來便產生了先移到右邊，再向下移動的曲線：

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.curveTo(175, 125, 200, 200);
```

下列範例將以 100 像素的寬度與高度繪製紅色和綠色的圓形物件：請注意，由於二次方貝茲曲線方程式的限制使然，因此無法繪製出完美的圓形：

```
var size:uint = 100;
var roundObject:Shape = new Shape();

// red circular shape
roundObject.graphics.beginFill(0xFF0000);
roundObject.graphics.moveTo(size / 2, 0);
roundObject.graphics.curveTo(size, 0, size, size / 2);
roundObject.graphics.curveTo(size, size, size / 2, size);
roundObject.graphics.curveTo(0, size, 0, size / 2);
roundObject.graphics.curveTo(0, 0, size / 2, 0);

// green circular shape
roundObject.graphics.beginFill(0x00FF00);
roundObject.graphics.moveTo(200 + size / 2, 0);
roundObject.graphics.curveTo(200 + size, 0, 200 + size, size / 2);
roundObject.graphics.curveTo(200 + size, size, 200 + size / 2, size);
roundObject.graphics.curveTo(200, size, 200, size / 2);
roundObject.graphics.curveTo(200, 0, 200 + size / 2, 0);

this.addChild(roundObject);
```

使用內建方法來繪製形狀

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

為了方便您繪製如圓形、橢圓形、矩形與圓角矩形之類的常見形狀，ActionScript 3.0 特地提供一些可為您繪製這些常見形狀的方法。這些方法分別是 Graphics 類別的 drawCircle()、drawEllipse()、drawRect() 以及 drawRoundRect() 方法。這些方法也可以用來取代 lineTo() 與 curveTo() 方法。然而，在您呼叫這些方法之前，仍舊必須指定線段與填色樣式，這點請您特別留意。

下列範例將以 100 的像素寬度與高度來重新建立紅色、綠色與藍色的正方形繪圖範例。下列程式碼使用 drawRect() 方法，並另外再指定含 50% (0.5) Alpha 值的填色顏色：

```
var squareSize:uint = 100;
var square:Shape = new Shape();
square.graphics.beginFill(0xFF0000, 0.5);
square.graphics.drawRect(0, 0, squareSize, squareSize);
square.graphics.beginFill(0x00FF00, 0.5);
square.graphics.drawRect(200, 0, squareSize, squareSize);
square.graphics.beginFill(0x0000FF, 0.5);
square.graphics.drawRect(400, 0, squareSize, squareSize);
square.graphics.endFill();
this.addChild(square);
```

在 Sprite 或 MovieClip 物件中，使用 graphics 屬性所建立的繪圖內容一律會出現在包含於物件中的所有子顯示物件後方。而且，graphics 屬性內容並不是另外一個顯示物件，因此不會出現在 Sprite 或 MovieClip 物件子系清單中。例如，下列 Sprite 物件包含一個以其 graphics 屬性繪製的圓形，並在其子顯示物件清單中包含一個 TextField 物件：

```
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0xFFCC00);
mySprite.graphics.drawCircle(30, 30, 30);
var label:TextField = new TextField();
label.width = 200;
label.text = "They call me mellow yellow...";
label.x = 20;
label.y = 20;
mySprite.addChild(label);
this.addChild(mySprite);
```

請注意，TextField 會出現在圖像物件所繪製的圓形上方。

建立漸層線段與填色

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

圖像物件同時可以繪製筆畫與加上漸層 (不只是純色) 的填色。您可透過 lineGradientStyle() 方法來建立漸層筆畫，並使用 beginGradientFill() 方法來建立漸層填色。

兩種方法都可接受使用相同的參數。開頭的四個參數是必要參數：類型、顏色、Alpha 和比例。剩下的四個參數則是選用參數，但是對進階自訂作業卻很有幫助。

- 前四個參數可指定您所建立的漸層類型。可接受值 GradientType.LINEAR 或 GradientType.RADIAL。
- 第二個參數可指定要使用的顏色值陣列。在線性漸層中，顏色會由左至右依序排列。在放射狀漸層中，顏色則是由內向外排列。陣列顏色的排列方式代表了漸層中的顏色繪製順序。
- 第三個參數可指定先前參數中對應顏色的 Alpha 透明值。
- 第四個參數則是指定比例，或是漸層中每個顏色的比重。可接受的數值為 0-255，這些值並非代表任何寬度或高度，而是代表漸層中的顏色位置；0 代表位於漸層的開頭，而 255 則代表位於漸層的結尾。比例陣列必須依序增加，並與第二及第三個參數中所指定的顏色與 Alpha 陣列擁有相同的項目數量。

雖然第五個參數 (也就是變形矩陣) 是選用的參數，還是很常用，因為它所提供的漸層外觀控制功能不但容易使用，威力也相當強大。這個參數可接受使用 Matrix 實體。建立漸層的 Matrix 物件最簡便的方式就是，使用 Matrix 類別的 createGradientBox() 方法。

定義 Matrix 物件以搭配漸層使用

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 flash.display.Graphics 類別的 beginGradientFill() 和 lineGradientStyle() 方法，定義在形狀中使用的漸層。定義漸層時，您必須提供矩陣做為這些方法的其中一個參數。

定義矩陣最簡單的方式，就是使用 Matrix 類別的 createGradientBox() 方法，它會建立用來定義漸層的矩陣。您可以使用傳遞給 createGradientBox() 方法的參數，定義漸層的縮放、旋轉及位置。createGradientBox() 方法接受下列參數：

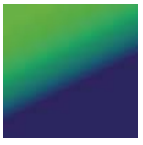
- 漸層方塊寬度：漸層擴散的寬度 (以像素為單位)
- 漸層方塊高度：漸層擴散的高度 (以像素為單位)
- 漸層方塊旋轉：將套用至漸層的旋轉方式 (以放射狀進行)
- 水平轉譯：漸層的水平平移距離 (以像素為單位)
- 垂直轉譯：漸層的垂直平移距離 (以像素為單位)

例如，假設漸層具有下列特性：

- GradientType.LINEAR
- 兩種顏色 (綠色和藍色)，並將 ratios 陣列設定為 [0, 255]
- SpreadMethod.PAD
- InterpolationMethod.LINEAR_RGB

下列範例所顯示的漸層，其 createGradientBox() 方法的 rotation 參數都不同，但是其它設定則不變：

width = 100;
height = 100;
rotation = 0;
tx = 0;
ty = 0;	

width = 100;
height = 100;
rotation = Math.PI/4; // 45°
tx = 0;
ty = 0;
	

width = 100;
height = 100;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 0;	

下列範例會顯示從綠色到藍色之線性漸層的效果，其中 createGradientBox() 方法的 rotation、tx 和 ty 參數都不同，但是其它設定則不變：

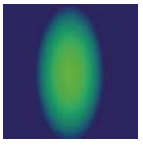
width = 50;
height = 100;
rotation = 0;
tx = 0;
ty = 0;	

width = 50;
height = 100;
rotation = 0
tx = 50;
ty = 0;	

width = 100;
height = 50;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 0;	

width = 100;
height = 50;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 50;	

createGradientBox() 方法的 width、height、tx 和 ty 參數也都會影響 「放射狀」漸層填色的大小與位置，如下列範例所示：

width = 50;
height = 100;
rotation = 0;
tx = 25;
ty = 0;
	

下列程式碼會產生上圖所示的放射狀漸層：

```
import flash.display.Shape;
import flash.display.GradientType;
import flash.geom.Matrix;

var type:String = GradientType.RADIAL;
var colors:Array = [0x00FF00, 0x000088];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var spreadMethod:String = SpreadMethod.PAD;
var interp:String = InterpolationMethod.LINEAR_RGB;
var focalPtRatio:Number = 0;

var matrix:Matrix = new Matrix();
var boxWidth:Number = 50;
var boxHeight:Number = 100;
var boxRotation:Number = Math.PI/2; // 90°
var tx:Number = 25;
var ty:Number = 0;
matrix.createGradientBox(boxWidth, boxHeight, boxRotation, tx, ty);

var square:Shape = new Shape;
square.graphics.beginGradientFill(type,
                            colors,
                            alphas,
                            ratios,
                            matrix,
                            spreadMethod,
                            interp,
                            focalPtRatio);
square.graphics.drawRect(0, 0, 100, 100);
addChild(square);
```

請注意，漸層填色的寬度與高度是由漸層矩陣的寬度與高度決定，而不是由 Graphics 物件繪製的寬度與高度來決定。使用 Graphics 物件時，您可以在漸層矩陣中的這些座標點上繪圖。即使您使用的是 Graphics 物件的其中一個形狀方法 (如 drawRect())，漸層並不會自己延伸為所繪製的形狀大小，漸層大小必須透過漸層矩陣本身才能加以指定。

下列將說明漸層矩陣尺寸以及繪圖尺寸本身，兩者之間在視覺上的差異：

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(100, 40, 0, 0, 0);
myShape.graphics.beginGradientFill(GradientType.LINEAR, [0xFF0000, 0x00FF00, 0x0000FF], [1, 1, 1], [0, 128,
255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 50, 40);
myShape.graphics.drawRect(0, 50, 100, 40);
myShape.graphics.drawRect(0, 100, 150, 40);
myShape.graphics.endFill();
this.addChild(myShape);
```

上述程式碼使用相同的填色樣式，並指定平均分配紅色、綠色與藍色來繪製三個漸層。繪製漸層時，分別使用了 50、100，與 150 像素寬度的 drawRect() 方法。beginGradientFill() 方法中指定的漸層矩陣會以 100 像素寬度來建立。也就是說，第一個漸層將僅涵蓋一半的漸層光譜，而第二個漸層則將涵蓋整個漸層，第三個漸層則是不僅僅涵蓋所有漸層，同時還向右側延伸了 50 像素的藍色光譜。

lineGradientStyle() 方法的作用與 beginGradientFill() 非常類似，差別在於前者可以定義漸層，而且在繪圖前必須先使用 lineStyle() 方法指定筆畫粗細。下列程式碼將以紅色、綠色與藍色漸層筆畫來繪製方塊：

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(200, 40, 0, 0, 0);
myShape.graphics.lineStyle(5, 0);
myShape.graphics.lineGradientStyle(GradientType.LINEAR, [0xFF0000, 0x00FF00, 0x0000FF], [1, 1, 1], [0, 128,
255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 200, 40);
this.addChild(myShape);
```

如需使用 Matrix 類別的詳細資訊，請參閱第 182 頁「使用 Matrix 物件」。

搭配繪圖方法使用 Math 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Graphics 物件不只可以繪製圓形與正方形，還可以繪製更複雜的形狀，尤其當您合併使用繪圖方法與 Math 類別的屬性與方法時，將可發揮更強大的功能。Math 類別包含了數學運算常用的常數，例如 Math.PI (約為 3.14159265...)，此常數可用來計算圓周與其直徑的比例 (圓周率)。它同時包含了三角函數中的一些方法，特別是 Math.sin()、Math.cos() 與 Math.tan()。使用這些方法與常數來繪製形狀時，如果合併使用重複與遞迴，則可以建立更加生動的視覺效果。

許多 Math 類別方法在測量圓形時，都會預期得到弧形單位而不是角度單位。Math 類別的常見用途，就是在這兩種單位之間進行轉換：

```
var degrees = 121;
var radians = degrees * Math.PI / 180;
trace(radians) // 2.111848394913139
```

下列範例將建立一個正弦波形與一個餘弦波形，以便凸顯某個值對 Math.sin() 與 Math.cos() 方法的不同作用。

```
var sinWavePosition = 100;
var cosWavePosition = 200;
var sinWaveColor:uint = 0xFF0000;
var cosWaveColor:uint = 0x00FF00;
var waveMultiplier:Number = 10;
var waveStretcher:Number = 5;

var i:uint;
for(i = 1; i < stage.stageWidth; i++)
{
    var sinPosY:Number = Math.sin(i / waveStretcher) * waveMultiplier;
    var cosPosY:Number = Math.cos(i / waveStretcher) * waveMultiplier;

    graphics.beginFill(sinWaveColor);
    graphics.drawRect(i, sinWavePosition + sinPosY, 2, 2);
    graphics.beginFill(cosWaveColor);
    graphics.drawRect(i, cosWavePosition + cosPosY, 2, 2);
}
```

使用繪圖 API 來建立動畫

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用繪圖 API 來建立內容的其中一項好處，就是您可以隨時移動內容位置。您可以維護並修改用來繪圖的變數，任意修改您的繪圖。您可以變更變數並在一段影格上或是透過計時器來重新繪製，以加上動畫。

例如，下列程式碼將隨著每個通過的影格變更顯示 (方法是偵聽 Event.ENTER_FRAME 事件)、遞增目前的角度計數，最後指引圖像物件清除繪圖，並在更新的位置上重新繪圖。

```
stage.frameRate = 31;

var currentDegrees:Number = 0;
var radius:Number = 40;
var satelliteRadius:Number = 6;

var container:Sprite = new Sprite();
container.x = stage.stageWidth / 2;
container.y = stage.stageHeight / 2;
addChild(container);
var satellite:Shape = new Shape();
container.addChild(satellite);

addEventListener(Event.ENTER_FRAME, doEveryFrame);

function doEveryFrame(event:Event):void
{
    currentDegrees += 4;
    var radians:Number = getRadians(currentDegrees);
    var posX:Number = Math.sin(radians) * radius;
    var posY:Number = Math.cos(radians) * radius;
    satellite.graphics.clear();
    satellite.graphics.beginFill(0);
    satellite.graphics.drawCircle(posX, posY, satelliteRadius);
}
function getRadians(degrees:Number):Number
{
return degrees * Math.PI / 180;
}
```

若要產生明顯不同的結果，可以多方嘗試修改程式碼開頭的初始 seed 變數 (currentDegrees、radius，和 satelliteRadius)。例如，您可以嘗試縮減 radius 變數，並且 / 或是增加 totalSatellites 變數。上述僅就繪圖 API 如何建立一個看起來複雜，實則非常容易建立的視覺顯示效果，做單一範例說明。

繪製 API 範例：演算法視覺產生器

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

演算法視覺產生器範例會以動態方式在舞台上繪製數個 「衛星」，或是讓幾個圓形繞著圓形軌道運行。這裡要特別探討的重點包括：

- 使用繪圖 API 來繪製具有動態外觀的基本形狀
- 連接使用者互動與繪圖中所使用的屬性
- 在每個影格中清除舞台並重新繪製來表現動畫

上一節中的範例使用 Event.ENTER_FRAME 事件，製作單個 「衛星」的動畫。本範例則以此為基礎加以擴充，建立包含一系列滑動軸的控制面板，可以立即更新數個衛星的視覺顯示。此範例將程式碼寫成外部類別，然後將每個衛星的參考儲存在 satellites 陣列中，以將建立衛星的程式碼納入迴圈中。

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。您可以在 Samples/AlgorithmicVisualGenerator 檔案夾中找到應用程式檔案。此資料夾包含下列檔案：

檔案	說明
AlgorithmicVisualGenerator.fla	Flash Professional 中的主要應用程式檔案 (FLA)。
com/example/programmingas3/algorithmic/AlgorithmicVisualGenerator.as	此類別可提供應用程式的主要功能，包括在舞台上繪製衛星並回應來自控制面板的事件，以便更新會影響衛星繪製效果的變數。
com/example/programmingas3/algorithmic/ControlPanel.as	此類別可透過數個滑動軸來管理使用者互動，並在發生互動時傳送事件。
com/example/programmingas3/algorithmic/Satellite.as	此類別不只可用來代表繞著軌道中心點旋轉的顯示物件，其中更包含與其現有繪圖狀態相關的屬性。

設定偵聽程式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

應用程式首先會建立三個偵聽程式。第一個會偵聽來自控制面板的傳送事件，此事件告知需要重新建立衛星。第二個會偵聽 SWF 檔的舞台大小變更。第三個則是偵聽在 SWF 檔中每個通過的影格，並使用 doEveryFrame() 函數來重新繪圖。

建立衛星

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

一旦設定好這些偵聽程式，就會呼叫 build() 函數。這個函數會先呼叫 clear() 函數，清空 satellites 陣列並清除舞台上先前完成的所有繪圖。這是必要的步驟，因為每當控制面板傳送事件時，就會重新呼叫 build() 函數，例如當顏色設定已經變更時，執行這項作業。在這種情況下，必須移除所有的衛星並重新建立衛星。

函數會接著建立衛星、設定建立時所需的初始屬性 (例如可在軌道的任意點位置上開始的 position 變數，以及在此範例中，就算建立了衛星，也不會改變的 color 變數)。

每建立一個衛星，就會將相關參考加入 satellites 陣列中。呼叫 doEveryFrame() 函數時，該函數就會更新陣列中的所有衛星。

更新衛星位置

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

doEveryFrame() 函數是應用程式動畫程序中的核心。每個影格都會呼叫它，其呼叫速率等於 SWF 檔的影格速率。由於繪圖的變數會稍微變更，這個函數可用來傳達動畫的外觀。

此函數首先會清除先前的所有繪圖，並在背景重新繪圖。然後，它會重複執行每個衛星容器，並遞增每個衛星的 position 屬性，接著更新在使用者與控制面板互動中可能發生變化的 radius 與 orbitRadius 屬性。最後，衛星會呼叫 Satellite 類別的 draw() 方法，將自己更新至新的位置。

請注意，i 這個計數單位，只能遞增到 visibleSatellites 變數為止。這是因為假如使用者已經透過控制面板限制了要顯示的衛星數量，則不應該重新繪製迴圈中剩餘的衛星，而是應該加以隱藏起來。如果負責繪圖的迴圈後面馬上跟著另一個迴圈，就會出現這個情況。

當 doEveryFrame() 函數完成工作後，螢幕上的 visibleSatellites 數量就會隨著位置更新。

回應使用者互動

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用者互動必須透過 ControlPanel 類別所管理的控制面板才能產生。此類別會在設定偵聽程式時，一併設定每個滑動軸的個別最小、最大與預設值。每當使用者移動這些滑動軸時，就會呼叫 changeSetting() 函數。此函數會更新控制面板的屬性。如果變更要求重新建立顯示，則會傳送事件並在主應用程式檔中處理該事件。隨著控制面板的設定變更，doEveryFrame() 函數也會使用更新的變數來繪製每個衛星。

進一步自訂

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

此範例僅簡單說明如何使用繪圖 API 來產生影像。它使用了行數相對較少的程式碼，來建立可顯示相當複雜的互動式視覺效果。即使如此，此範例仍舊可透過些微更改來延伸其效果。下列是幾個可行的作法：

- 使用 doEveryFrame() 函數來遞增衛星的顏色值。
- 使用 doEveryFrame() 函數，隨著時間變動縮減或展開衛星半徑。
- 衛星半徑不一定非得沿著圓形軌道移動；例如，它可以使用 Math 類別沿著正弦波形移動。
- 衛星可以使用感應區偵測功能來偵測其它衛星。

當您想要在 Flash 編寫環境中建立視覺效果時，另一個作法就是使用繪圖 API 在執行階段繪製基本形狀。但是這種作法也可運用在建立手繪所無法辦到的各種不同視覺效果。ActionScript 作者只要運用繪圖 API 以及一點點的數學知識，就可以將生命注入許多令人耳目一新的創作作品中。

繪圖 API 的進階用法

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

Flash Player 10、Adobe AIR 1.5 和更新版本的 Flash 執行階段皆支援一組進階的繪圖功能。這些執行階段的繪圖 API 增強功能擴充了舊版的繪圖方法，讓您能夠建立資料集來產生形狀、在執行階段變更形狀，並建立 3D 立體效果。繪圖 API 增強功能將現有方法合併成交替運用的命令。這些命令會運用向量陣列和列舉類別，提供繪圖方法所需的資料集。使用向量陣列可讓更複雜的形狀更快呈現，並且讓開發人員以程式設計方式變更陣列值，以便於執行階段呈現動態形狀。

您可以在下列章節內，找到 Flash Player 10 中加入的繪圖功能：第 199 頁 「繪製路徑」、第 200 頁 「定義迂迴規則」、第 202 頁 「使用圖形資料類別」和第 204 頁 「關於使用 drawTriangles()」。

下列是您常會透過 ActionScript 的進階繪圖 API 來完成的幾項工作：

- 使用 Vector 物件儲存繪圖方法所需的資料
- 定義路徑，在單一作業中以程式設計方式繪製形狀
- 定義迂迴規則，判斷重疊圖形的填色方式
- 讀取顯示物件的向量圖像內容，例如序列化並儲存圖形資料、在執行階段產生 Sprite 工作表，或者繪製向量圖像內容的副本
- 使用三角形和繪圖方法產生 3D 特效

重要概念與術語

下列參考清單包含您將在本章碰到的重要術語：

- Vector：具有相同資料類型之值的陣列。Vector 物件可以儲存值的陣列，其中繪圖方法會使用這些值，透過單一命令建構線條和形狀。如需有關 Vector 物件的詳細資訊，請參閱第 22 頁「索引陣列」。
- 路徑：路徑是由一個或數個直線或曲線區段所構成。每個區段的起始和結束，是由座標標示，其作用就像固定纜繩的繩栓。路徑可以是封閉的 (例如圓形)；也可以是開放的，也就是有明確的端點 (例如波浪線)。
- 迂迴：路徑的方向 (由描繪器所解譯)，可以是正向 (順時針) 或負向 (逆時針)。
- GraphicsStroke：用於設定線段樣式的類別。使用類別以自己的填色屬性來指定線條樣式，是繪圖 API 增強功能的一部分，但「筆畫」不是。您可以使用 GraphicsStroke 類別動態調整線段的樣式。
- Fill 物件：使用顯示類別 (例如，傳遞至繪圖命令 Graphics.drawGraphicsData() 的 flash.display.GraphicsBitmapFill 和 flash.display.GraphicsGradientFill) 所建立的物件。Fill 物件和增強的繪圖命令採用了更為物件導向的程式設計方式來複製 Graphics.beginBitmapFill() 和 Graphics.beginGradientFill()。

繪製路徑

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

有關繪製線段和曲線的章節 (請參閱第 188 頁「繪製線段和曲線」) 中介紹了下列命令：用於繪製單一線段的命令 (Graphics.lineTo())、用於繪製單一曲線的命令 (Graphics.curveTo())，以及將線段移至另一點以形成圖形的命令 (Graphics.moveTo())。Graphics.drawPath() 和 Graphics.drawTriangles() 方法都可接受一組代表那些相同繪圖命令的物件做為參數。透過這些方法，您可以提供一連串的 Graphics.lineTo()、Graphics.curveTo() 或 Graphics.moveTo() 命令給 Flash 執行階段在單一陳述式中執行。

GraphicsPathCommand 列舉類別會定義一組對應至繪圖命令的常數。您可以傳遞一連串的常數 (包裝在 Vector 實體中) 做為 Graphics.drawPath() 方法的參數。然後再透過單一命令，顯示整個形狀或數個形狀。此外，您也可以改變傳遞至這些方法的值，以變更現有形狀。

除了繪圖命令的向量以外，drawPath() 方法還需要一組座標，而且該座標對應至每個繪圖命令的座標。請建立包含座標 (Number 實體) 的 Vector 實體，並傳遞至 drawPath() 方法做為第二個 (data) 引數。

備註：向量中的這些值都不是 Point 物件，因為向量是一系列數字，其中每兩個數字一組，即可表示一組 x/y 座標。

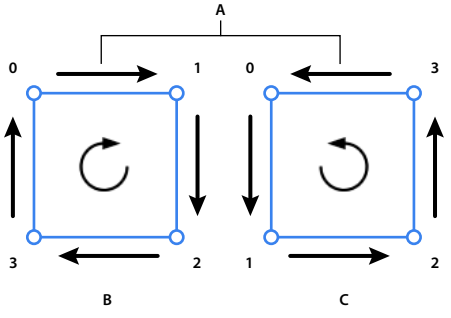
Graphics.drawPath() 方法會使每個命令符合其個別的座標點值 (兩個或四個為一組的數字)，以便於 Graphics 物件中產生路徑：

```
package
{
    import flash.display.*;

    public class DrawPathExample extends Sprite
    {
        public function DrawPathExample(){

            var squareCommands:Vector.<int> = new Vector.<int>(5, true);
            squareCommands[0] = GraphicsPathCommand.MOVE_TO;
            squareCommands[1] = GraphicsPathCommand.LINE_TO;
            squareCommands[2] = GraphicsPathCommand.LINE_TO;
            squareCommands[3] = GraphicsPathCommand.LINE_TO;
            squareCommands[4] = GraphicsPathCommand.LINE_TO;

            var squareCoord:Vector.<Number> = new Vector.<Number>(10, true);
            squareCoord[0] = 20; //x
            squareCoord[1] = 10; //y
            squareCoord[2] = 50;
            squareCoord[3] = 10;
            squareCoord[4] = 50;
            squareCoord[5] = 40;
            squareCoord[6] = 20;
            squareCoord[7] = 40;
            squareCoord[8] = 20;
            squareCoord[9] = 10;

            graphics.beginFill(0x442266);//set the color
            graphics.drawPath(squareCommands, squareCoord);
        }
    }
}
```

定義迂迴規則

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

增強的繪圖 API 還加入了路徑「迂迴」的概念：路徑的方向。路徑的迂迴可以是正向 (順時針) 或負向 (逆時針)。描繪器解譯座標 (由向量所提供，做為 data 參數) 的順序會決定迂迴。



正向和負向迂迴
A. 箭頭表示繪製的方向 **B.** 正向迂迴 (順時針) **C.** 負向迂迴 (逆時針)

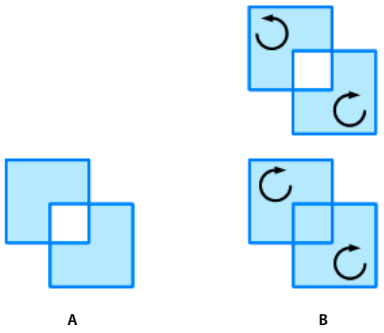
另外，請注意 Graphics.drawPath() 方法還有第三個選擇性參數，名為「winding」：

```
drawPath(commands:Vector.<int>, data:Vector.<Number>, winding:String = "evenOdd"):void
```

在本段內容中，第三個參數是字串或常數，可以指定交叉路徑的迂迴或填色規則 (這些常數值是在 GraphicsPathWinding 類別中定義為 GraphicsPathWinding.EVEN_ODD 或 GraphicsPathWinding.NON_ZERO)。當路徑交叉時，迂迴規則便很重要。

奇偶規則是標準的迂迴規則，也是舊版繪圖 API 所使用的規則。此外，奇偶規則也是 Graphics.drawPath() 方法的預設規則。使用奇偶規則時，所有交叉的路徑都會交替使用開啟和結束填色。如果兩個使用相同填色繪製的正方形交集，交集區域會填色。一般而言，相鄰區域既不會兩者皆填色也不會兩者皆未填色。

另一方面，非零規則需視迂迴 (繪製方向) 來決定交叉路徑所定義的區域是否要填色。當具有相反迂迴的路徑交叉時，所定義的區域就不會填色，這和奇偶規則很像。如果是相同迂迴的路徑，則不會填色的區域便會填色：

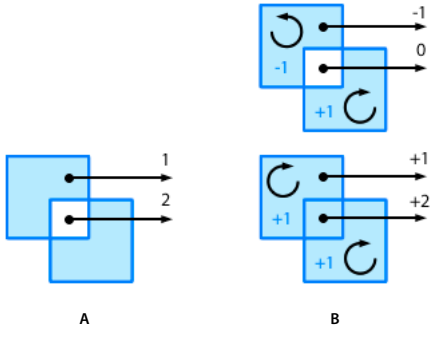


交集區域的迂迴規則
A. 奇偶迂迴規則 **B.** 非零迂迴規則

迂迴規則名稱

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

名稱會用來表示更加特殊的規則，這個規則會定義管理這些填色的方式。對於正向迂迴路徑，所指定的值為 +1；而對於負向迂迴路徑，所指定的值則為 -1。從形狀中封閉區域的某一點開始，繪製一條向外無限延伸的線條。該線條與某一路徑交叉的次數以及這些路徑的合併值都會用於決定填色。對於奇偶迂迴而言，會使用該線條與某一路徑的交叉次數。當次數為奇數時，區域就會填色。當次數為偶數時，區域則不會填色。對於非零迂迴而言，則會使用指定給這些路徑的值。當路徑的合併值為 0 以外的數字時，區域就會填色。當合併值為 0 時，區域就不會填色。



迂迴規則次數與填色
A. 奇偶迂迴規則 **B.** 非零迂迴規則

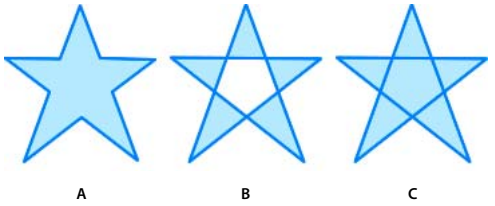
使用迂迴規則

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

雖然這些填色規則相當複雜，但是在某些情況下是不可或缺的。例如，以繪製星形形狀為例。使用標準的奇偶規則時，需要十個不同的線段才能繪製這個形狀。而使用非零迂迴規則時，十條線則減少為五條。以下是使用五條線及非零迂迴規則繪製星形的 ActionScript：

```
graphics.beginFill(0x60A0FF);
graphics.drawPath( Vector.<int>([1,2,2,2,2]),  Vector.<Number>([66,10, 23,127, 122,50, 10,49, 109,127]),
GraphicsPathWinding.NON_ZERO);
```

以下為星形形狀：



使用不同迂迴所繪製的星形形狀
A. 使用奇偶規則，以 10 條線繪製 **B.** 使用奇偶規則，以 5 條線繪製 **C.** 使用非零規則，以 5 條線繪製

此外，當影像在製作成動畫或做為立體物件和重疊的紋理使用時，迂迴規則便亦顯重要。

使用圖形資料類別

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

增強的繪圖 API 包括 flash.display 套件中的一組類別，這些類別會實作 IGraphicsData 介面。這些類別可當做 value 物件 (資料容器) 使用，代表繪圖 API 的繪圖方法。

下列類別會實作 IGraphicsData 介面：

- GraphicsBitmapFill
- GraphicsEndFill
- GraphicsGradientFill
- GraphicsPath
- GraphicsShaderFill
- GraphicsSolidFill
- GraphicsStroke
- GraphicsTrianglePath

透過上述類別，您就可以將完整繪圖儲存在 IGraphicsData 類型的 Vector 物件中 (Vector.<IGraphicsData>)。然後，您可以將圖形資料當做資料來源，重複用於其他形狀實體，或者儲存繪圖資訊以供日後使用。

請注意，每個填色樣式都有多個填色類別可供使用，但是您只能使用一個筆畫類別。ActionScript 只有一個筆畫 IGraphicsData 類別，因為這個筆畫類別可以使用多個填色類別定義其樣式。因此，每個筆畫實際是由筆畫類別和填色類別的組合所定義。另外，這些圖形資料類別的 API 會反映其在 flash.display.Graphics 類別中的表示方法：

圖形方法	對應的類別
beginBitmapFill()	GraphicsBitmapFill
beginFill()	GraphicsSolidFill
beginGradientFill()	GraphicsGradientFill
beginShaderFill()	GraphicsShaderFill
lineBitmapStyle()	GraphicsStroke + GraphicsBitmapFill

圖形方法	對應的類別
lineGradientStyle()	GraphicsStroke + GraphicsGradientFill
lineShaderStyle()	GraphicsStroke + GraphicsShaderFill
lineStyle()	GraphicsStroke + GraphicsSolidFill
moveTo() lineTo() curveTo() drawPath()	GraphicsPath
drawTriangles()	GraphicsTrianglePath

此外，GraphicsPath 類別有自己的 GraphicsPath.moveTo()、GraphicsPath.lineTo()、GraphicsPath.curveTo()、GraphicsPath.wideLineTo() 以及 GraphicsPath.wideMoveTo() 公用程式方法，可用來輕易地為 GraphicsPath 實體定義這些命令。這些公用程式方法可以簡化直接定義或更新命令和資料值的工作。

使用向量圖像資料來繪圖

當您擁有一組 IGraphicsData 實體之後，可以使用 Graphics 類別的 drawGraphicsData() 方法來顯示這些圖形。drawGraphicsData() 方法會從 IGraphicsData 實體的向量依序執行一組繪圖指示：

```
// stroke object
var stroke:GraphicsStroke = new GraphicsStroke(3);
stroke.joints = JointStyle.MITER;
stroke.fill = new GraphicsSolidFill(0x102020);// solid stroke

// fill object
var fill:GraphicsGradientFill = new GraphicsGradientFill();
fill.colors = [0x0000FF, 0xEEFFEE];
fill.matrix = new Matrix();
fill.matrix.createGradientBox(70, 70, Math.PI/2);
// path object
var path:GraphicsPath = new GraphicsPath(new Vector.<int>(), new Vector.<Number>());
path.commands.push(GraphicsPathCommand.MOVE_TO, GraphicsPathCommand.LINE_TO, GraphicsPathCommand.LINE_TO);
path.data.push(125,0, 50,100, 175,0);

// combine objects for complete drawing
var drawing:Vector.<IGraphicsData> = new Vector.<IGraphicsData>();
drawing.push(stroke, fill, path);

// draw the drawing
graphics.drawGraphicsData(drawing);
```

藉由修改上述範例中繪圖所使用之路徑內的其中一個值，即可針對更複雜的影像多次重繪圖形：

```
// draw the drawing multiple times
// change one value to modify each variation
graphics.drawGraphicsData(drawing);
path.data[2] += 200;
graphics.drawGraphicsData(drawing);
path.data[2] -= 150;
graphics.drawGraphicsData(drawing);
path.data[2] += 100;
graphics.drawGraphicsData(drawing);
path.data[2] -= 50;graphicsS.drawGraphicsData(drawing);
```

透過 IGraphicsData 物件即可定義填色和筆畫樣式，但是您不一定要使用這些填色和筆畫樣式。換句話說，Graphics 類別方法可用於設定樣式，而 IGraphicsData 物件則可用於繪製已儲存的一組路徑，反之亦然。

備註：除非您是要在原始繪圖上新增內容 (如上述範例中所示)，否則，在開始新的繪圖之前，請使用 Graphics.clear() 方法清除舊的繪圖。當您變更某一路徑或一組 IGraphicsData 物件的單一部分時，請重繪整個繪圖以查看變更內容。

使用圖形資料類別時，只要繪製三個以上的點就會呈現填色，這是因為形狀在本質上會在該點結束的緣故。即使填色結束了，但筆畫仍未結束，因此這項行為指令與使用 Graphics.lineTo() 或 Graphics.moveTo() 命令時不同。

讀取向量圖像資料

Flash Player 11.6 以及更新的版本，**Adobe AIR 3.6** 以及更新的版本

除了繪製顯示物件的向量內容以外，在 Flash Player 11.6 和 Adobe AIR 3.6 及更新版本中，您還可以使用 Graphics 類別的 readGraphicsData() 方法取得顯示物件之向量圖像內容的資料形式。這可用來建立圖形的快照，以便在執行階段儲存、複製、建立 Sprite 工作表等等。

呼叫 readGraphicsData() 方法會傳回包含 IGraphicsData 物件的 Vector 實體。這些是與搭配 drawGraphicsData() 方法繪製向量圖像時所用的相同物件。

搭配 readGraphicsData() 方法讀取向量圖像會有一些限制。如需詳細資訊，請參閱 ActionScript 語言參考中的 readGraphicsData() 項目。

關於使用 drawTriangles()

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

Flash Player 10 和 Adobe AIR 1.5 引進的另一個進階方法，Graphics.drawTriangles() 與 Graphics.drawPath() 命令類似。Graphics.drawTriangles() 方法也會使用 Vector.<Number> 物件，指定繪製路徑所需的座標點位置。

不過，Graphics.drawTriangles() 方法的實際目的，是要透過 ActionScript 協助產生 3D 特效。如需有關使用 Graphics.drawTriangles() 以產生立體特效的詳細資訊，請參閱第 306 頁「使用三角形產生 3D 特效」。

第 13 章 使用點陣圖

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 除了具有向量繪圖功能之外，還包括建立點陣圖影像或操作已載入 SWF 的外部點陣圖影像之像素資料的功能。透過存取和變更個別像素值的功能，您就可以建立自己的類似濾鏡之影像效果，並使用內建的雜訊功能以建立紋理與隨機雜訊。

- Renaun Erickson：使用位圖傳送技術在 ActionScript 中呈現遊戲資源 (英文)
- 點陣圖程式設計：Essential ActionScript 3 第 26 章，作者 Colin Moock (O'Reilly Media，2007) (英文)
- Mike Jones：在 Pushbutton Engine 中使用 Sprite (英文)
- Flash & Math：像素粒子變得簡單 (英文)
- Flixel

使用點陣圖的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用數位影像時，您多半會遇到兩種主要類型的圖像：點陣圖和向量。點陣圖 (Bitmap) 圖像又稱為點陣 (Raster) 圖像，是由極小的矩形 (像素) 所組成，它們是以矩形格線形式排列。向量圖像是由以數學方式產生的幾何形狀 (例如線條、曲線和多邊形) 所組成。

點陣圖影像是由影像的寬度與高度 (以像素為單位) 所定義，而且每個像素中所包含的位元數目，代表像素可以包含的顏色數目。在使用 RGB 色彩模式的點陣圖影像中，像素是由三個位元組所組成：代表紅色、綠色和藍色。其中每一個位元組都包含範圍介於 0 到 255 之間的值。當這些位元組結合在像素中時，就會產生顏色類似於混合繪製顏色的藝術效果。例如，由紅色值為 255、綠色值為 102，以及藍色值為 0 組成之位元組值之像素，會產生鮮明的橘色。

點陣圖影像的品質是由結合影像解析度及其顏色深度位元值所決定。「解析度」是指包含在影像中的像素數目。像素的數目愈大，解析度就愈高，也就會顯示較細緻的影像。「顏色深度」是指像素可以包含的資訊量。例如，每個像素有 16 位元顏色深度值的影像，無法將相同數目的顏色顯示為具有 48 位元顏色深度的影像。因此，48 位元的影像將會比 16 位元的影像有更平滑的陰影。

因為點陣圖圖像與解析度有關，所以它們在縮放時的表現並不出色。這一點在點陣圖影像放大時最為明顯。放大點陣圖，通常會犧牲細部畫質與品質。

點陣圖檔案格式

點陣圖影像可區分為數種常見的檔案格式。這些格式會各自使用不同類型的壓縮演算法以減少檔案大小，並根據影像的最終目的來最佳化影像品質。Adobe 執行階段支援的點陣圖影像有 BMP、GIF、JPG、PNG 和 TIFF。

BMP

BMP (點陣) 格式為 Microsoft Windows 作業系統預設使用的影像格式。這種格式不會使用任何壓縮演算法形式，因此本身產生的檔案通常相當龐大。

GIF

「圖形交替格式」(Graphics Interchange Format，GIF) 原本是由 CompuServe 在 1987 年所開發，做為以 256 色 (8 位元顏色) 傳輸影像的方法。此格式可提供小型檔案，適用於網頁用的影像。因為這個格式的色盤有限，所以 GIF 影像通常不適用於相片，相片通常需要使用較高級的陰影與漸層。GIF 影像允許單一位元的透明度，它允許將顏色對應為清晰的 (或透明的)。這會導致從已對應透明度的影像可直接看到網頁所顯示的背景顏色。

JPEG

JPEG 格式是由「聯合圖像開發小組」(Joint Photographic Experts Group，JPEG) 所開發，JPEG (通常寫成 JPG) 影像格式會使用失真的壓縮演算法，允許小型檔案具有 24 位元顏色深度。失真的壓縮表示每次儲存影像時，影像會損失品質與資料，但是可使檔案大小更為精簡。JPEG 格式適用於相片，因為它可以顯示數百萬種顏色。將控制壓縮程度的功能套用至影像，可讓您操作影像品質與檔案大小。

PNG

由於 GIF 檔案格式具有專利，因此做為替代方案、開放原始碼的 「可攜式網路圖案」 (Portable Network Graphics，PNG) 格式便應運而生。PNG 支援高達 64 位元的顏色深度，最多允許 1 千 6 百萬種顏色。因為 PNG 相對而言是相當新的格式，所以有些較舊的瀏覽器並不支援 PNG 檔案。與 JPG 不同的是，PNG 會使用不失真的壓縮方式，這表示在儲存影像時，不會損失任何影像資料。PNG 檔案也支援 Alpha 透明度，最多允許 256 層級的透明度。

TIFF

在 PNG 格式發表之前，「標籤式影像檔案格式」(TIFF) 曾經是各種平台所選擇使用的格式。TIFF 格式的缺點在於 TIFF 具有多種不同的變化，因此只靠單一讀取程式無法處理所有的版本。此外，目前也沒有網頁瀏覽器支援這種格式。TIFF 可以使用失真或非失真的壓縮，以及處理裝置專屬的顏色空間 (例如 CMYK)。

透明的點陣圖與不透明的點陣圖

使用 GIF 或 PNG 格式的點陣圖影像，其每個像素都可以增加額外的位元組 (Alpha 色版)。這個額外像素位元組代表像素的透明度值。

GIF 影像允許單一位元透明度，這表示您可以從 256 色的面板中，指定要成為透明色的單一顏色。另一方面，PNG 影像最多可以有 256 層級的透明度。當影像或文字需要融入背景時，這個功能就特別有用。

ActionScript 3.0 會在 BitmapData 類別中複製此額外透明度像素位元組。ActionScript 類似於 PNG 透明度模型，提供最多 256 層級的透明度。

重要概念與術語

下列清單包含學習點陣圖圖形時會遇到的相關重要術語：

Alpha 在顏色或影像中透明度 (更精確地說，是不透明度) 的層級。Alpha 量通常描述為 「Alpha 色版」值。

ARGB 顏色 一種顏色配置，其中每個像素的顏色都是紅色、綠色和藍色值的混合，而其透明度是以 Alpha 值指定。

顏色色版 通常，顏色會以一些基本顏色的混合來表示，一般 (對於電腦圖像而言) 為紅色、綠色和藍色。每個基本顏色都視為一個顏色色版。將每個顏色色版中的顏色量混合在一起，就會決定最終的顏色。

顏色深度 又稱為 「位元深度」，這是指用於每個像素的電腦記憶體量，這會決定可在影像中顯示的可能顏色數目。

像素 在點陣圖影像中的最小單位資訊，基本上是一個顏色點。

解析度 影像的像素尺寸，可決定影像微粒的細緻程度。解析度通常是根據寬度與高度的像素數目表示。

RGB 顏色 一種顏色配置，其中每個像素的顏色是以紅色、綠色和藍色值的混合來呈現。

Bitmap 與 BitmapData 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

搭配點陣圖影像使用的主要 ActionScript 3.0 類別，包括用來在螢幕上顯示點陣圖影像的 Bitmap 類別，以及用來存取和操作點陣圖原始影像資料的 BitmapData 類別。

更多說明主題

flash.display.Bitmap

flash.display.BitmapData

瞭解 Bitmap 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

做為 DisplayObject 類別的子類別，Bitmap 類別是用來顯示點陣圖影像的主要 ActionScript 3.0 類別。這些影像可能已經透過 flash.display.Loader 類別載入，或使用 Bitmap() 建構函式以動態方式建立。從外部來源載入影像時，Bitmap 物件只能使用 GIF、JPEG 或 PNG 格式的影像。一旦經過初始化，就可將 Bitmap 實體視為必須在「舞台」上顯示之 BitmapData 物件的包裝函式。因為 Bitmap 實體是一個顯示物件，所以顯示物件的所有特色與功能也可用來操作 Bitmap 實體。如需有關使用顯示物件的詳細資訊，請參閱第 128 頁「顯示程式設計」。

像素貼齊與平滑化功能

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

除了所有顯示物件共用的功能之外，Bitmap 類別還會提供點陣圖影像特有的一些其它功能。

Bitmap 類別中的 pixelSnapping 屬性可決定 Bitmap 物件是否會貼齊其最近的像素。這個屬性會採用在 PixelSnapping 類別中定義的下列三個常數之一：ALWAYS, AUTO 和 NEVER。

套用像素貼齊的語法如下：

```
myBitmap.pixelSnapping = PixelSnapping.ALWAYS;
```

通常在縮放點陣圖影像時，會造成模糊且扭曲的影像。為了減少此扭曲的情形，請使用 BitmapData 類別的 smoothing 屬性。將此 Boolean 屬性設定為 true 時，只要縮放影像，就會對影像中的像素進行平滑化 (或消除鋸齒)。這可以讓影像呈現更清晰而且更自然的外觀。

瞭解 BitmapData 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

BitmapData 類別 (在 flash.display 套件中) 可以比喻為包含在已載入或動態建立的點陣圖影像中，像素的相片快照。此快照是由物件中的像素資料陣列所呈現。BitmapData 類別也包含一系列的內建方法，這些方法對於建立和操作像素資料非常有用。

若要初始化 BitmapData 物件，請使用下列程式碼：

```
var myBitmap:BitmapData = new BitmapData(width:Number, height:Number, transparent:Boolean,
fillColor:uinit);
```

width 和 height 參數會指定點陣圖的大小。從 AIR 3 和 Flash Player 11 開始，已移除 BitmapData 物件的大小限制。點陣圖的大小上限取決於作業系統。

在 AIR 1.5 和 Flash Player 10 中，BitmapData 物件的大小上限為 8,191 像素寬度或高度，且像素總計不得超過 16,777,215 像素 (因此，如果 BitmapData 物件的寬度為 8,191 像素，則其高度上限為 2,048 像素)。若為 Flash Player 9 和 AIR 1.1 及其更早版本，高度和寬度的上限都是 2,880 像素。

transparent 參數會指定點陣圖資料是 (true) 否 (false) 包括 Alpha 色版。fillColor 參數是 32 位元顏色值，它會指定背景顏色及透明度值 (如果已經設定為 true)。下列範例會建立 BitmapData 物件，並設定具有 50% 透明度的橘色背景：

```
var myBitmap:BitmapData = new BitmapData(150, 150, true, 0x80FF3300);
```

若要在螢幕上顯示新建立的 BitmapData 物件，請將它指定給 Bitmap 實體，或將它包裝在 Bitmap 實體中。若要這麼做，您可以將 BitmapData 物件當做 Bitmap 物件之建構函式的參數來傳遞，或是可以將它指定給現有之 Bitmap 實體的 bitmapData 屬性。您也必須呼叫容納 Bitmap 實體之顯示物件容器的 addChild() 或 addChildAt() 方法，將 Bitmap 實體加入顯示清單。如需有關使用顯示清單的詳細資訊，請參閱第 134 頁 「將顯示物件加入顯示清單」。

下列範例會建立 BitmapData 物件並以紅色做為填色，再將它顯示在 Bitmap 實體中：

```
var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0xFF0000);
var myImage:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myImage);
```

操作像素

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

BitmapData 類別包含一組方法，可讓您操作像素資料值。

操作個別像素

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在像素層級變更點陣圖影像的外觀時，必須先取得包含在您要操作的區域中之像素顏色值。您可以使用 getPixel() 方法來讀取這些像素值。

getPixel() 方法會從一組 x, y (像素) 座標擷取 RGB 值，這些座標是當做參數傳遞。如果想要操作的任何像素包含透明度 (Alpha 色版) 資訊，您就必須使用 getPixel32() 方法。此方法也會擷取 RGB 值，但是與使用 getPixel() 不同的是，getPixel32() 傳回的值還包含其它資料，代表所選取像素的 Alpha 色版 (透明度) 值。

或者，若只要變更包含在點陣圖中的像素顏色或透明度，您可以使用 setPixel() 或 setPixel32() 方法。若要設定像素的顏色，只要將 x, y 座標及顏色值傳遞給下列其中一項方法即可。

下列範例會使用 setPixel() 在綠色的 BitmapData 背景上繪製叉號。然後再使用 getPixel()，從位於座標 50, 50 的像素擷取顏色值，並追蹤傳回值。

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 100, false, 0x009900);

for (var i:uint = 0; i < 100; i++)
{
    var red:uint = 0xFF0000;
    myBitmapData.setPixel(50, i, red);
    myBitmapData.setPixel(i, 50, red);
}

var myBitmapImage:Bitmap = new Bitmap(myBitmapData);
addChild(myBitmapImage);

var pixelValue:uint = myBitmapData.getPixel(50, 50);
trace(pixelValue.toString(16));
```

如果您要讀取一組像素的值 (相對於單一像素)，請使用 getPixels() 方法。這個方法會從像素資料的矩形區域產生位元組陣列，該像素資料區域會當做參數傳遞。該位元組陣列 (換句話說，就是像素值) 的每一個元素都是無正負號整數：32 位元、未相乘的像素值。

反過來說，為了變更 (或設定) 一組像素的值，請使用 setPixels() 方法。此方法需要兩個參數 (rect 與 inputByteArray)，結合這兩個參數就能輸出像素資料 (inputByteArray) 的矩形區域 (rect)。

從 inputByteArray 以外讀取 (和寫入) 資料時，會針對陣列中的每個像素呼叫 ByteArray.readUnsignedInt() 方法。基於某些原因，如果 inputByteArray 不包含完整矩形的像素資料，該方法就會停止處理在該點的影像資料。

因此，請務必記住在取得和設定像素資料時，位元組陣列需要 32 位元的 Alpha、紅色、綠色、藍色 (ARGB) 像素值。

下列範例會使用 getPixels() 和 setPixels() 方法，將一組像素從某個 BitmapData 物件複製到另一個 BitmapData 物件：

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.utils.ByteArray;
import flash.geom.Rectangle;

var bitmapDataObject1:BitmapData = new BitmapData(100, 100, false, 0x006666FF);
var bitmapDataObject2:BitmapData = new BitmapData(100, 100, false, 0x00FF0000);

var rect:Rectangle = new Rectangle(0, 0, 100, 100);
var bytes:ByteArray = bitmapDataObject1.getPixels(rect);

bytes.position = 0;
bitmapDataObject2.setPixels(rect, bytes);

var bitmapImage1:Bitmap = new Bitmap(bitmapDataObject1);
addChild(bitmapImage1);
var bitmapImage2:Bitmap = new Bitmap(bitmapDataObject2);
addChild(bitmapImage2);
bitmapImage2.x = 110;
```

像素層級衝突偵測

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

BitmapData.hitTest() 方法會在點陣圖資料與其它物件或點之間執行像素層級衝突偵測。

BitmapData.hitTest() 方法會採用五個參數：

- firstPoint (Point)：此參數會參考第一個 BitmapData 的左上角像素位置，在這個位置上會執行碰撞測試。

- firstAlphaThreshold (uint)：此參數指定會在此碰撞測試中視為不透明的最高 Alpha 色版值。
- secondObject (Object)：此參數代表影響的區域。secondObject 物件可以是 Rectangle、Point、Bitmap 或 BitmapData 物件。此物件代表執行衝突偵測的作用區域。
- secondBitmapDataPoint (Point)：此選擇性參數可以用來在第二個 BitmapData 物件中定義像素位置。只有在 secondObject 的值是 BitmapData 物件時，才會使用此參數。預設值是 null。
- secondAlphaThreshold (uint)：此選擇性參數代表在第二個 BitmapData 物件中視為不透明的最高 Alpha 色版值。預設值為 1。只有當 secondObject 的值為 BitmapData 物件，而且兩個 BitmapData 物件都為透明時，才能使用此參數。

針對不透明的影像執行衝突偵測時，請記住 ActionScript 會將影像視為彷彿它是完全不透明的矩形 (或是範圍框)。或者，針對透明的影像執行像素層級碰撞測試時，這兩個影像都必須是透明的。除此之外，ActionScript 還會使用 Alpha 臨界值參數，決定在哪一點像素會從透明變成不透明。

下列範例會建立三個點陣圖影像，並使用兩個不同的衝突點 (一個會傳回 false，另一個會傳回 true) 來檢查像素衝突：

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.geom.Point;

var bmd1:BitmapData = new BitmapData(100, 100, false, 0x000000FF);
var bmd2:BitmapData = new BitmapData(20, 20, false, 0x00FF3300);

var bm1:Bitmap = new Bitmap(bmd1);
this.addChild(bm1);

// Create a red square.
var redSquare1:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare1);
redSquare1.x = 0;

// Create a second red square.
var redSquare2:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare2);
redSquare2.x = 150;
redSquare2.y = 150;

// Define the point at the top-left corner of the bitmap.
var pt1:Point = new Point(0, 0);
// Define the point at the center of redSquare1.
var pt2:Point = new Point(20, 20);
// Define the point at the center of redSquare2.
var pt3:Point = new Point(160, 160);

trace(bmd1.hitTest(pt1, 0xFF, pt2)); // true
trace(bmd1.hitTest(pt1, 0xFF, pt3)); // false
```

複製點陣圖資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要將一個影像中的點陣圖資料複製到另一個影像中，您可以使用數種方法：clone()、copyPixels()、copyChannel()、draw() 和 drawWithQuality() (drawWithQuality 方法可以在 Flash Player 11.3 及更新版本，以及 AIR 3.3 及更新版本中取得)。

如其名稱所建議，clone() 方法可讓您將點陣圖資料從某個 BitmapData 物件複製或取樣到另一個 BitmapData 物件。呼叫這個方法時，它會傳回新的 BitmapData 物件，此物件是與原始實體完全相同的副本。

下列範例會複製橘色 (父輩) 矩形的副本，並將此副本置於原始父輩矩形旁邊：

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myParentSquareBitmap:BitmapData = new BitmapData(100, 100, false, 0x00ff3300);
var myClonedChild:BitmapData = myParentSquareBitmap.clone();

var myParentSquareContainer:Bitmap = new Bitmap(myParentSquareBitmap);
this.addChild(myParentSquareContainer);

var myClonedChildContainer:Bitmap = new Bitmap(myClonedChild);
this.addChild(myClonedChildContainer);
myClonedChildContainer.x = 110;
```

copyPixels() 方法是一種既快速又簡單的方式，可以從某個 BitmapData 物件將像素複製到另一個 BitmapData 物件。該方法會拍攝來源影像的矩形快照 (由 sourceRect 參數所定義)，並將它複製到另一個矩形區域 (大小相等)。新「貼上」的矩形位置是在 destPoint 參數中加以定義。

copyChannel() 方法會從來源 BitmapData 物件取樣預先定義的顏色色版值 (Alpha、紅色、綠色或藍色)，並將它複製到目標 BitmapData 物件的色版中。呼叫此方法並不會影響目標 BitmapData 物件中的其它色版。

draw() 和 drawWithQuality() 方法會從來源 Sprite、影片片段或其他顯示物件，將圖像內容繪製或顯示到新點陣圖。使用 matrix、colorTransform、blendMode 以及目標 clipRect 參數，您就可以修改顯示新點陣圖的方式。此方法會使用 Flash Player 或 AIR 中的向量描繪器來產生資料。

當您呼叫 draw() 或 drawWithQuality() 時，可以將來源物件 (Sprite、影片片段或其他顯示物件) 當做第一個參數傳遞，如以下所示：

```
myBitmap.draw(movieClip);
```

如果來源物件在起初載入之後套用了任何變形 (顏色、矩陣等等)，並不會將這些變形複製到新物件。如果您要將變形複製到新點陣圖，則需要將 transform 屬性的值從原始物件複製到使用新 BitmapData 物件的 Bitmap 物件之 transform 屬性。

壓縮點陣圖資料

Flash Player 11.3 以及更新的版本，**AIR 3.3** 以及更新的版本

flash.display.BitmapData.encode() 方法可讓您以原生方式將點陣圖資料壓縮至下列其中一個影像壓縮格式：

- **PNG** - 使用 PNG 壓縮即選擇性地使用快速壓縮，這種壓縮方法偏重壓縮速度，而非檔案大小。若要使用 PNG 壓縮，請傳遞新的 flash.display.PNGEncoderOptions 物件做為 BitmapData.encode() 方法的第二個參數。
- **JPEG** - 使用 JPEG 壓縮即選擇性地指定影像品質。若要使用 JPEG 壓縮，請傳遞新的 flash.display.JPEGEncoderOptions 物件做為 BitmapData.encode() 方法的第二個參數。
- **JPEGXR** - 使用 JPEG Extended Range (XR) 壓縮，即選擇性地指定顏色色版、失真和 Entropy 設定。若要使用 JPEGXR 壓縮，請傳遞新的 flash.display.JPEGXREncoderOptions 物件做為 BitmapData.encode() 方法的第二個參數。

您可以將這個功能用於影像處理，做為伺服器上傳或下載工作流程的一部分。

下列範例程式碼片段使用 JPEGEncoderOptions 壓縮 BitmapData 物件：

```
// Compress a BitmapData object as a JPEG file.
var bitmapData:BitmapData = new BitmapData(640,480,false,0x00FF00);
var byteArray:ByteArray = new ByteArray();
bitmapData.encode(new Rectangle(0,0,640,480), new flash.display.JPEGEncoderOptions(), byteArray);
```

使用雜訊函數製作紋理

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要修改點陣圖的外觀，您可以使用 noise() 方法或是 perlinNoise() 方法，在點陣圖上套用雜訊效果。雜訊效果可以比喻為出現在未調頻之電視螢幕上的電波干擾。

若要將雜訊效果套用到點陣圖，請使用 noise() 方法。這個方法會將隨機的顏色值套用到點陣圖影像的指定區域內。

此方法可接受五個參數：

- randomSeed (int)：決定圖樣的隨機種子數值。雖然其名稱為如此，如果傳遞相同的數目，此數目實際上會建立相同的結果。為了取得真正隨機的結果，請使用 Math.random() 方法為此參數傳遞隨機數值。
- low (uint)：此參數指的是要為每個像素所產生的最低值 (0 到 255)，預設值為 0。設定此值會降低在較黑的雜訊圖樣中的結果，而將它設定為較高的值會導致較亮的圖樣。
- high (uint)：此參數指的是要為每個像素所產生的最高值 (0 到 255)，預設值為 255。設定為較小的值時，會產生較暗的雜訊圖樣；而設定為較大的值則會產生較亮的圖樣。
- channelOptions (uint)：此參數會指定雜訊圖樣將套用的點陣圖物件之顏色色版。此數字可以是任何四個顏色色版 ARGB 值的組合，預設值為 7。
- grayScale (Boolean)：設定為 true 時，此參數會將 randomSeed 值套用至點陣圖像素，可有效地將影像的所有顏色洗掉。此參數不受 Alpha 色版的影響。預設值為 false。

下列範例會建立點陣圖影像，並將藍色的雜訊圖樣套用至該影像：

```
package
{
    import flash.display.Sprite;
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.BitmapDataChannel;

    public class BitmapNoise1 extends Sprite
    {
        public function BitmapNoise1()
        {
            var myBitmap:BitmapData = new BitmapData(250, 250,false, 0xff000000);
            myBitmap.noise(500, 0, 255, BitmapDataChannel.BLUE,false);
            var image:Bitmap = new Bitmap(myBitmap);
            addChild(image);
        }
    }
}
```

如果您想要建立看起來更有組織的紋理，請使用 perlinNoise() 方法。perlinNoise() 方法會產生寫實且具組織性的紋理，適用於煙霧、雲狀、水、火或甚至是爆炸效果。

因為它是由演算法所產生，所以比起點陣圖紋理，perlinNoise() 方法會使用更少的記憶體。然而，它對於處理器使用率仍然有影響，因此會使內容變慢，造成重繪螢幕的速度比影格速率更慢，特別是較舊的電腦更是如此。這主要是為了處理 Perlin 雜訊演算法，因此需要執行浮點數計算。

此方法會採用九個參數 (前六個為必要參數)：

- baseX (Number)：決定所建立圖樣的 x (大小) 值。
- baseY (Number)：決定所建立圖樣的 y (大小) 值。
- numOctaves (uint)：建立此雜訊所需結合的 octave 或是個別的雜訊函數數量。較大的 octave 數字所建立的影像能呈現更細膩的畫質，但是也需要更多的處理時間。

- randomSeed (int)：隨機種子數字的運作方式與它在 noise() 函數中的運作方式完全相同。為了取得真正隨機的結果，請使用 Math.random() 方法為此參數傳遞隨機號碼。
- stitch (Boolean)：如果設定為 true，則此方法會嘗試柔合 (平滑化) 影像的轉場邊緣，以建立無接縫的拼貼紋理做為點陣圖填色。
- fractalNoise (Boolean)：此參數與此方法所產生的漸層邊緣有關。如果設定為 true，則此方法會產生不規則雜訊，以平滑化該效果的邊緣。如果設定為 false，則會造成湍流化。帶有亂流的影像具有明顯不連續的漸層，可讓它產生較逼近、鮮明的視覺特效，例如火焰和海浪。
- channelOptions (uint)：channelOptions 參數的運作方式與它在 noise() 方法中的運作方式完全相同。它會指定雜訊圖樣所套用的點陣圖顏色色版。此數字可以是任何四個顏色色版 ARGB 值的組合，預設值為 7。
- grayScale (Boolean)：grayScale 參數的運作方式與它在 noise() 方法中的運作方式完全相同。如果設定為 true，它會將 randomSeed 值套用至點陣圖像素，可有效地將影像的所有顏色洗掉。預設值為 false。
- offsets (Array)：對應至每個 octave 之 x 和 y 偏移的點陣列。藉由操作偏移值，您可以平順地捲動影像的圖層。在偏移陣列中的每個點都會影響特定的 octave 雜點函數。預設值是 null。

下列範例將建立一個 150 x 150 像素的 BitmapData 物件，它會呼叫 perlinNoise() 方法來產生綠色與藍色雲狀效果：

```
package
{
    import flash.display.Sprite;
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.BitmapDataChannel;

    public class BitmapNoise2 extends Sprite
    {
        public function BitmapNoise2()
        {
            var myBitmapDataObject:BitmapData =
                new BitmapData(150, 150, false, 0x00FF0000);

            var seed:Number = Math.floor(Math.random() * 100);
            var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE
            myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

            var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
            addChild(myBitmap);
        }
    }
}
```

捲動點陣圖

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

想像一下您已建立街道地圖應用程式，每次使用者移動地圖，就需要更新檢視 (即使只是將地圖移動幾個像素也一樣)。

建立此功能的其中一個方法就是每次使用者移動地圖時，重新顯示包含已更新地圖檢視的新影像。或者，您可以建立較大的單一影像並使用 scroll() 方法。

scroll() 方法會複製螢幕上的點陣圖，然後將它貼到新的偏移位置，這個位置是由 (x, y) 參數指定。如果點陣圖的部分剛好在舞台以外，這將造成影像位移的效果。當結合計時器函數 (或是 enterFrame 事件) 時，您可以讓影像以動畫或捲動的方式顯示。

下列範例採用前面的 perlin 雜訊範例，並產生較大的點陣圖影像 (影像的四分之三顯示在舞台以外)。接著會套用 scroll() 方法以及 enterFrame 事件偵聽程式，它會沿著對角線的方向，將影像向下偏移一個像素。每次進入影格時就會呼叫此方法，因此當影像向下捲動時，就會將影像在螢幕外的部分顯示至 「舞台」。

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(1000, 1000, false, 0x00FF0000);
var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE;
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
myBitmap.x = -750;
myBitmap.y = -750;
addChild(myBitmap);

addEventListener(Event.ENTER_FRAME, scrollBitmap);

function scrollBitmap(event:Event):void
{
    myBitmapDataObject.scroll(1, 1);
}
```

善用 MIP 映射的優點

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

「MIP 映射」(MIP Map，亦稱為「Mipmap」) 是與紋理有關的一組點陣圖，可提高執行階段的顯示品質和效能。MIP 映射中的每個點陣圖影像是主要點陣圖影像的版本，但不如主要影像細緻。

例如，您的 MIP 映射包含最高品質、64 × 64 像素的主要影像。MIP 映射中較低品質的影像為 32 × 32、16 × 16、8 × 8、4 × 4、2 × 2 和 1 × 1 像素。

「紋理串流」是一種功能，可優先載入最低品質的點陣圖，然後以漸進方式一邊載入點陣圖一邊顯示較高品質的點陣圖。由於較低品質的點陣圖較小，所以載入的速度比主要影像來得快。因此，應用程式使用者可以在主要、高品質點陣圖載入之前，先在應用程式中檢視影像。

Flash Player 9.115.0 和更新版本以及 AIR 都會為每個點陣圖建立各種縮放 (從 50% 開始) 的最佳化版本，以實作這項技術 (此一實作程序稱為 「MIP 映射」)。

Flash Player 11.3 和 AIR 3.3 支援透過 Context3D.createCubeTexture() 及 Context3D.createTexture() 方法的 streamingLevels 參數進行紋理串流。

紋理壓縮可讓您以壓縮格式直接將紋理影像儲存在 GPU 上，以便節省 GPU 記憶體和記憶體頻寬。一般來說，壓縮的紋理是以壓縮格式離線壓縮後上傳到 GPU 的。然而，Flash Player 11.4 和 AIR 3.4 支援執行階段紋理壓縮，在特定情況下，例如從向量圖像顯示動態紋理時，此壓縮會很有用。若要使用執行階段紋理壓縮，請執行下列步驟：

- 呼叫 Context3D.createTexture() 方法、以第三個參數傳遞 flash.display3D.Context3DTextureFormat.COMPRESSED 或 flash.display3D.Context3DTextureFormat.COMPRESSED_ALPHA，藉此建立紋理物件。
- 使用 createTexture() 所傳回的 flash.display3D.textures.Texture 實體，呼叫 flash.display3D.textures.Texture.uploadFromBitmapData() 或 flash.display3D.textures.Texture.uploadFromByteArray()。這些方法會一次上傳並壓縮紋理。

會針對下列類型的點陣圖影像建立 MIP 對應：

- 使用 ActionScript 3.0 Loader 類別顯示的點陣圖 (JPEG、GIF 或 PNG 檔案)

- Flash Professional 文件的文件庫中的點陣圖
- BitmapData 物件
- 使用 ActionScript 2.0 loadMovie() 函數顯示的點陣圖

MIP 映射不會套用至濾鏡的物件或已快取點陣圖的影片片段。不過，如果濾鏡的顯示物件中有點陣圖變形，那麼即使點陣圖位於被遮蔽的內容中，還是可以套用 MIP 映射。

MIP 映射會自動建立，但是您也可以根據下列幾項原則，確保影像能充分利用此最佳化的優點：

- 針對視訊播放，將 Video 物件的 smoothing 屬性設定為 true (請參閱 Video 類別)。
- 針對點陣圖，smoothing 屬性則不需要設定為 true，但是當點陣圖使用平滑化功能時，品質的改善效果較為明顯。
- 對於二維影像，使用可被 4 或 8 除盡的點陣圖大小 (例如 640 x 128，可以縮小為 320 x 64 > 160 x 32 > 80 x 16 > 40 x 8 > 20 x 4 > 10 x 2 > 5 x 1)。

  對於三維紋理，請使用每個影像的解析度為 2 的倍數 (即 2^n) 的 MIP 映射。例如，主要影像的解析度為 1024 x 1024 像素。然後，MIP 映射中較低品質的影像會是 512 x 512、256 x 256、128 x 128 像素，降到 MIP 映射中 11 個影像共 1 x 1 像素。

  請注意，若點陣圖內容的高度或寬度是奇數，則不會進行 MIP 映射。

點陣圖範例：動畫轉動的月球

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

轉動的月球動畫範例示範了使用 Bitmap 物件和點陣圖影像資料 (BitmapData 物件) 的技術。該範例會使用月球表面的平面化影像做為原始影像資料，建立轉動、球面的月球動畫。當中將示範下列技術：

- 載入外部影像並存取其原始影像資料
- 從來源影像的不同部分重複複製像素以建立動畫
- 設定像素值來建立點陣圖影像

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。您可以在 Samples/SpinningMoon 資料夾中找到轉動的月球動畫之應用程式檔案。此應用程式是由下列檔案組成：

檔案	說明
SpinningMoon.mxml 或 SpinningMoon.fla	主應用程式檔案，在 Flex 中為 MXML，在 Flash 中為 FLA。
com/example/programmingas3/moon/MoonSphere.as	此類別會執行月球的載入、顯示和製成動畫等功能。
moonMap.png	內含月球表面相片的影像檔，該檔案會載入並用於建立旋轉的月球動畫。

將外部影像載入為點陣圖資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

這個樣本執行的第一項主要工作是載入一個外部影像檔，即月球表面的相片。載入作業會在 MoonSphere 類別內以兩種方法處理：在 MoonSphere() 建構函式中初始化載入程序，以及在外部影像完全載入時呼叫 imageLoadComplete() 方法。

載入外部影像的方法與載入外部 SWF 的方法類似；兩者皆使用 flash.display.Loader 類別的實體執行載入作業。MoonSphere() 方法內開始載入影像的實際程式碼如下：

```
var imageLoader:Loader = new Loader();
imageLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, imageLoadComplete);
imageLoader.load(new URLRequest("moonMap.png"));
```

在第一行中，宣告名為 imageLoader 的 Loader 實體。在第三行中，藉由呼叫 Loader 物件的 load() 方法，傳遞一個 URLRequest 實體以代表要載入之影像的 URL，實際啟動載入程序。在第二行中，設定將在影像完全載入時觸發的事件偵聽程式。請注意，addEventListener() 方法不是對 Loader 實體本身呼叫，而是對 Loader 物件的 contentLoaderInfo 屬性呼叫。Loader 實體本身不會傳送與載入之內容相關的事件。不過，其 contentLoaderInfo 屬性內含 LoaderInfo 物件的參考，而該物件與正在載入至 Loader 物件的內容 (此例中為外部影像) 的內容相關。LoaderInfo 物件提供了數個與載入外部內容的進度和完成相關的事件，包括 complete 事件 (Event.COMPLETE)，該事件將會在影像完全載入時觸發對 imageLoadComplete() 方法的呼叫。

雖然啟動外部影像載入是程序中的重要部分，不過瞭解載入完成時應執行什麼動作也同樣重要。如上面的程式碼中所示，影像載入完成時，便會呼叫 imageLoadComplete() 函數。該函數會對載入的影像資料執行幾項作業，我們將於陸續說明。不過，若要使用影像資料，就必須存取該資料。當使用 Loader 物件載入外部影像時，載入的影像會成為 Bitmap 實體，此實體會附加為 Loader 物件的子顯示物件。在這個情況下，Loader 實體可以讓做為事件物件一部分的事件偵聽程式方法使用，將其當做參數以傳遞至方法。imageLoadComplete() 方法的前幾行程式碼如下：

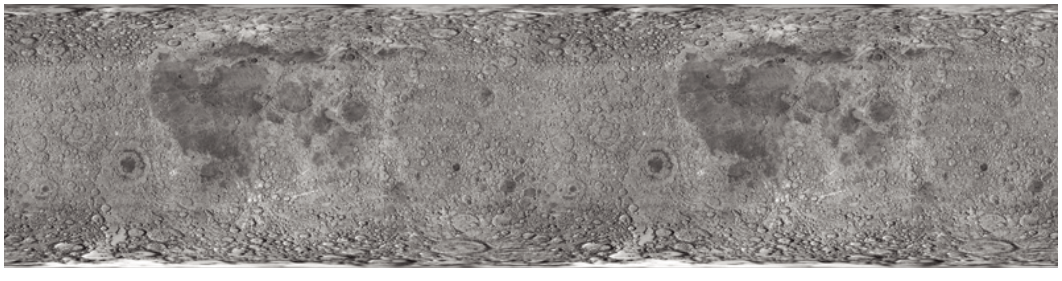
```
private function imageLoadComplete(event:Event):void
{
    textureMap = event.target.content.bitmapData;
    ...
}
```

請注意，事件物件的參數名為 event，它是 Event 類別的實體。Event 類別的每個實體都有一個 target 屬性，參照到觸發事件的物件 (此例中，即為 addEventListener() 方法所呼叫的 LoaderInfo 實體，如先前的內容所提及)。接著，LoaderInfo 物件內會有一個 content 屬性 (載入程序完成後)，內含載入之點陣圖影像的 Bitmap 實體。如果您想要直接在螢幕上顯示影像，則可以將此 Bitmap 實體 (event.target.content) 附加到顯示物件容器 (您也可以將 Loader 物件附加到顯示物件容器)。不過，在此樣本中，載入的內容會當做原始影像資料的來源使用，而不會顯示在螢幕上。因此，imageLoadComplete() 方法的第一行會讀取所載入 Bitmap 實體的 bitmapData 屬性 (event.target.content.bitmapData)，並將其儲存到名為 textureMap 的實體變數中，在建立旋轉的月球動畫時，該變數會做為影像資料的來源。本項於下列說明。

複製像素以建立動畫

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

動畫的基本定義是隨著時間而變更影像所產生的動態效果或變化。在此樣本中，目標是要建立依垂直軸旋轉的球面月球效果。不過，為了要瞭解動畫，您可以忽略樣本內球面扭曲的部分。考量實際載入以及做為月球影像資料來源的影像：



如您所見，影像並不是一或多個球體，而是月球表面的矩形相片。由於相片正好是在月球的赤道上拍攝，因此靠近影像頂端和底端的影像部分便會遭到延伸及扭曲。為了移除影像的扭曲效果，並使其顯示為球面，我們將會使用一個置換對應濾鏡，此部分將於稍後說明。不過，由於來源影像為矩形，因此若要建立球體旋轉的效果，程式碼只需要水平滑動月球表面相片即可。

請注意，影像實際包含了兩張緊鄰著彼此的月球表面相片之副本。這個影像為來源影像，影像資料會為了建立動畫的外觀而重複複製此來源影像。兩個影像副本若緊鄰著彼此，便可以更輕鬆地建立一個連續、不中斷的捲動效果。讓我們來逐步進行動作的處理程序，以瞭解其運作方式。

這個程序實際上包含了兩個不同的 ActionScript 物件。首先，其中有一個載入的來源影像，在程式碼內是以名為 textureMap 的 BitmapData 實體來表示。如同先前的內容所提及，textureMap 會在載入外部影像後，立即填入影像資料，當中使用的程式碼如下：

```
textureMap = event.target.content.bitmapData;
```

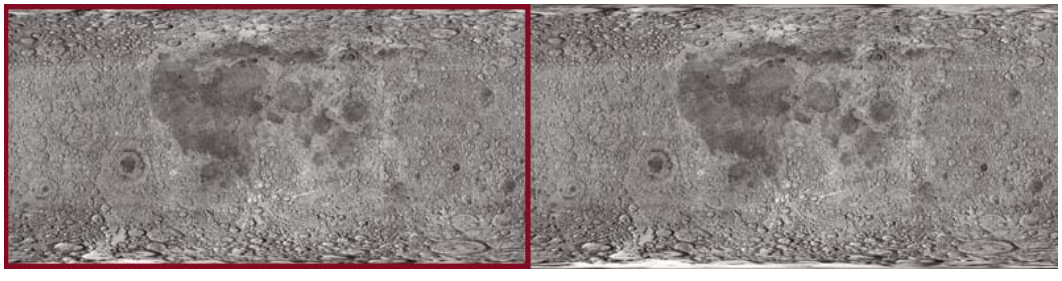
textureMap 的內容是矩形的月球影像。此外，為了建立旋轉的動畫，程式碼中會使用名為 sphere 的 Bitmap 實體，該實體是實際在螢幕上顯示月球影像的顯示物件。和 textureMap 一樣，在 imageLoadComplete() 方法內會建立 sphere 物件，並填入其初始的影像資料，其中使用了下列程式碼：

```
sphere = new Bitmap();
sphere.bitmapData = new BitmapData(textureMap.width / 2, textureMap.height);
sphere.bitmapData.copyPixels(textureMap,
                        new Rectangle(0, 0, sphere.width, sphere.height),
                        new Point(0, 0));
```

如程式碼中所示，sphere 已初始化。其 bitmapData 屬性 (由 sphere 顯示的原始影像資料) 會建立與 textureMap 具有相同的高度和其一半的寬度。換句話說，sphere 的內容將會是一個月球相片的大小 (因為 textureMap 影像內含兩個相連的月球相片)。接著，bitmapData 屬性會使用其 copyPixels() 方法填入影像資料。copyPixels() 方法呼叫內的參數會指定以下幾項作業：

- 第一個參數指定影像資料是從 textureMap 複製而來。
- 第二個參數 (即 new Rectangle 實體) 會指定要從 textureMap 的哪個部分拍攝影像快照。在此例中，該快照為起始自 textureMap 左上角的矩形 (由前兩個 Rectangle() 參數 0, 0 表示)，而此矩形快照的寬度和高度都符合 sphere 的 width 和 height 屬性。
- 第三個參數 (即 x 和 y 值都為 0 的新 Point 實體) 會定義像素資料的目標。在此例中，即為 sphere.bitmapData 的左上角 (0, 0)。

以視覺化的方式呈現時，程式碼會從下列影像以外框框住的部分複製 textureMap 像素，並將它們貼到 sphere 中。換句話說，sphere 的 BitmapData 內容即是此處強調的 textureMap 部分：



不過仍請記住，這只是 sphere 的初始狀態，也就是複製到 sphere 的第一個影像。

載入來源影像及建立 sphere 之後，由 imageLoadComplete() 方法執行的最後一項工作即是設定動畫。動畫由名為 rotationTimer 的 Timer 實體所驅動，這個實體是使用下列程式碼建立及啟動：

```
var rotationTimer:Timer = new Timer(15);
rotationTimer.addEventListener(TimerEvent.TIMER, rotateMoon);
rotationTimer.start();
```

程式碼會先建立名為 rotationTimer 的 Timer 實體，而傳遞給 Timer() 建構函式的參數表示 rotationTimer 應該每 15 毫秒就觸發其 timer 事件。接下來，則會呼叫 addEventListener() 方法，指定在 timer 事件 (TimerEvent.TIMER) 發生時呼叫 rotateMoon() 方法。最後，timer 實際上是由呼叫其 start() 方法所啟動的。

受到 rotationTimer 方法的定義方式之影響，因此大約每 15 毫秒 Flash Player 就會呼叫 MoonSphere 類別內的 rotateMoon() 方法，此處即是月球的動畫產生的地點。rotateMoon() 方法的原始碼如下：

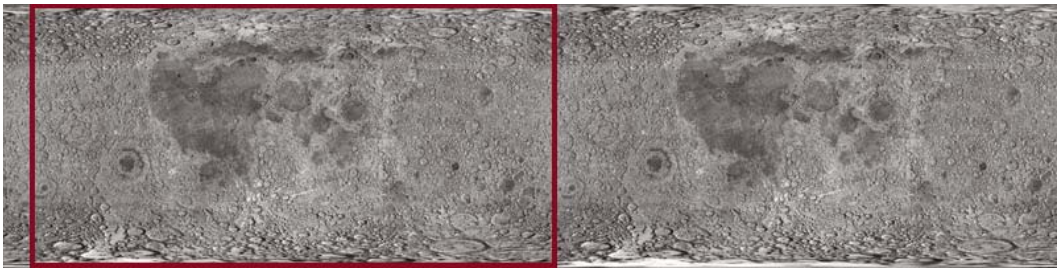
```
private function rotateMoon(event:TimerEvent):void
{
    sourceX += 1;
    if (sourceX > textureMap.width / 2)
    {
        sourceX = 0;
    }

    sphere.Data.copyPixels(textureMap,
                                    new Rectangle(sourceX, 0, sphere.width, sphere.height),
                                    new Point(0, 0));

    event.updateAfterEvent();
}
```

程式碼會執行下列三項作業：

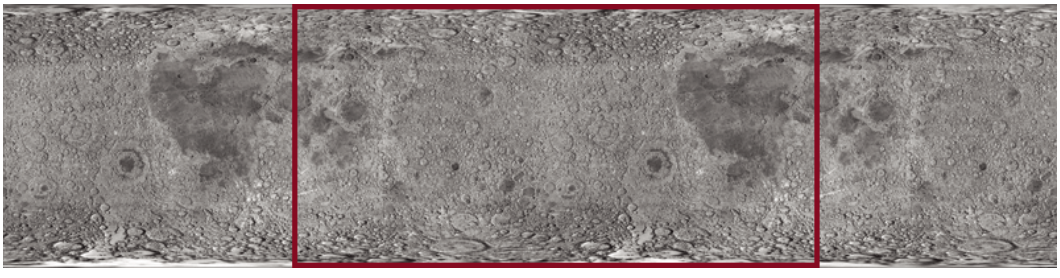
1 變數 sourceX 的值 (最初設定為 0) 會遞增 1。

```
sourceX += 1;
```

如您所見，sourceX 會用來決定像素將被複製到 sphere 上的 textureMap 內的哪個位置，因此這個程式碼具有在 textureMap 上將矩形向右移動一個像素的效果。回到視覺化呈現的部分，在幾個動畫循環後，來源矩形將會向右移動數個像素，就像這樣：

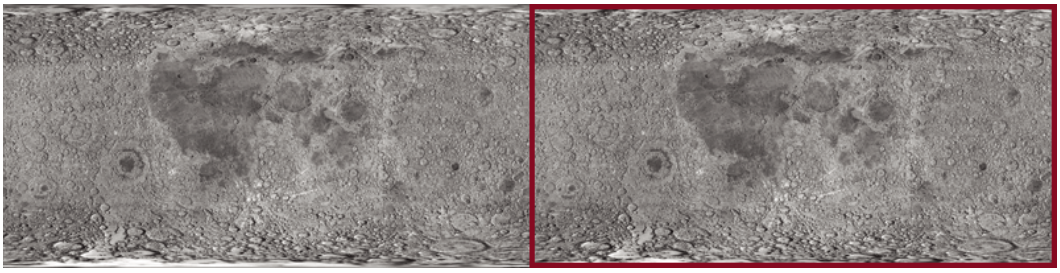


在更多個循環後，矩形將會移動得更遠：



在那些複製之像素中，這個逐步、平緩的位置移動即是動畫的關鍵。藉由緩慢及連續地將來源位置向右移動，顯示在螢幕上的影像會以 sphere 顯示為連續向左滑動的情形。這就是為什麼來源影像 (textureMap) 需要使用兩個月球表面相片副本的原因。由於矩形會連續向右移動，因此大部分的時間它不會覆蓋在單一月球相片上，而是與兩張月球相片重疊。

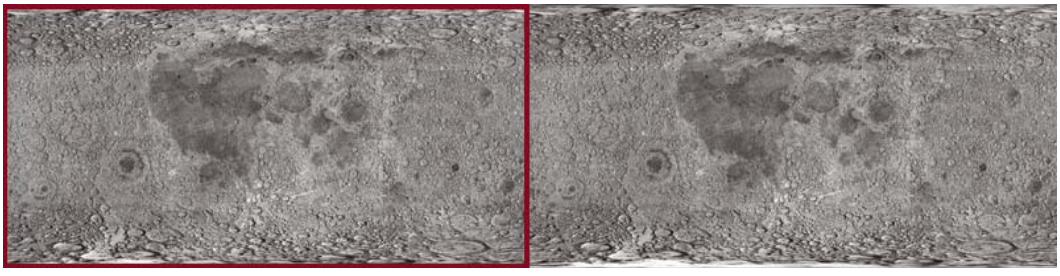
2 隨著來源矩形緩慢地向右移動，將出現一個問題。最後，矩形將會到達 textureMap 的右邊緣，並且將會耗盡可複製到 sphere 上的月球相片像素：



接下來的程式碼將可解決此問題：

```
if (sourceX >= textureMap.width / 2)
{
    sourceX = 0;
}
```

程式碼會檢查 sourceX (矩形的左邊緣) 是否已到達 textureMap 的中間。如果是的話，便會將 sourceX 重設回 0，使其移回 textureMap 的左邊緣，並再次啟動循環：



3 有了適當計算的 sourceX 值，建立動畫的最後一個步驟便是實際將新來源矩形像素複製到 sphere 上。執行此動作的程式碼非常類似於最初填入的 sphere (先前的內容中已提及)；唯一的差別是，在此例中，在 new Rectangle() 建構函式呼叫內，矩形的左邊緣會放置在 sourceX 上：

```
sphere.bitmapData.copyPixels(textureMap,
                             new Rectangle(sourceX, 0, sphere.width, sphere.height),
                             new Point(0, 0));
```

請記得，每隔 15 毫秒就會重複呼叫此程式碼。隨著來源矩形的位置連續地移動，並將像素複製到 sphere 上，畫面上會顯示由 sphere 代表的月球相片影像連續滑入之效果。換句話說，月球看起來就像在連續旋轉。

建立球面外觀

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當然，月球是一個球體而不是一個矩形。因此，樣本需要使用矩形的月球表面相片，將其製作成連續的動畫，並將其轉換為球體。這包含兩個不同的步驟：使用遮罩隱藏除了月球表面相片之圓形區域以外的所有內容，使用置換對應濾鏡扭曲月球相片外觀，使其看起來有立體感。

首先，使用一個圓形的遮罩來隱藏 MoonSphere 物件的所有內容，除了由濾鏡所建立的球體之外。下列程式碼會將遮罩建立為一個 Shape 實體，並將其套用為 MoonSphere 實體的遮罩：

```
moonMask = new Shape();
moonMask.graphics.beginFill(0);
moonMask.graphics.drawCircle(0, 0, radius);
this.addChild(moonMask);
this.mask = moonMask;
```

請注意，由於 MoonSphere 是顯示物件 (以 Sprite 類別為基礎)，因此可以使用所繼承的 mask 屬性將遮罩直接套用到 MoonSphere 實體上。

只使用圓形遮罩來隱藏相片的某些部分並不足以建立逼真的旋轉球體效果。由於月球表面的相片拍攝方式，其尺寸並未對稱；越接近影像頂端或底端的部分會越扭曲，相對於赤道部分會更具拉伸的感覺。為了要扭曲月球相片的外觀，使其看起來具有立體感，我們將會使用置換對應濾鏡。

置換對應濾鏡是一種用於扭曲影像的濾鏡類型。在此例中，將會 「扭曲」月球相片，藉由水平地擠壓影像的頂端和底端，同時保持中間部分不變動，使其看起來更為逼真。假設濾鏡在相片的方形部分上運作，則會擠壓頂端和底端而非中間部分，這會把方形轉換為圓形。將此扭曲的影像製作成動畫的副作用之一是，影像的中間部分實際移動的距離似乎比接近頂端和底端的部分更遠，如此將會產生圓形為立體 (球體) 的效果。

下列程式碼會用來建立置換對應濾鏡，其名為 displaceFilter：

```
var displaceFilter:DisplacementMapFilter;
displaceFilter = new DisplacementMapFilter(fisheyeLens,
                                new Point(radius, 0),
                                BitmapDataChannel.RED,
                                BitmapDataChannel.GREEN,
                                radius, 0);
```

第一個參數 fisheyeLens 稱為對應影像；在此例中是以程式設計方式建立的 BitmapData 物件。該影像的建立方式將於第 221 頁 「設定像素值來建立點陣圖影像」中說明。其他參數描述在已套用濾鏡影像中濾鏡的套用位置、將使用哪些顏色色版來控制置換效果，以及它們影響置換的程度。一旦建立了置換對應濾鏡，便會將其套用至 sphere (仍舊在 imageLoadComplete() 方法內)：

```
sphere.filters = [displaceFilter];
```

最終的影像 (即套用遮罩及置換對應濾鏡的影像) 看起來就像這樣：

隨著每個旋轉月球動畫的循環，來源影像資料的新快照會覆寫 sphere 的 BitmapData 內容。不過，並不需要每一次都重新套用濾鏡。這是因為濾鏡會套用至 Bitmap 實體上 (即顯示物件)，而不是點陣圖資料上 (即原始像素資訊)。請記住，Bitmap 實體不是實際的點陣圖資料；而是一個在螢幕上顯示點陣圖資料的顯示物件。舉例來說，Bitmap 實體就像是幻燈片放映機，可在螢幕上顯示相片幻燈片，而 BitmapData 物件則像是實際的相片幻燈片，可以透過幻燈片放映檔呈現。濾鏡可以直接套用至 BitmapData 物件，可與直接在相片幻燈片上繪圖以修改影像的動作進行做比較。濾鏡也可以套用到任何顯示物件上，包括 Bitmap 實體；這種情形類似於將濾鏡放在幻燈片放映機的鏡頭前方，以扭曲顯示在螢幕上的輸出 (不會改變原始的幻燈片)。由於原始點陣圖資料可透過 Bitmap 實體的 bitmapData 屬性存取，濾鏡便會直接套用至原始點陣圖資料上。不過在此例中，將濾鏡套用至 Bitmap 顯示物件比套用至點陣圖資料更為合理。

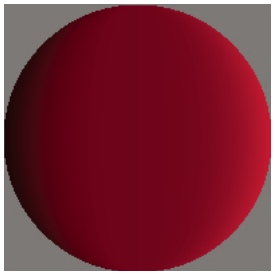
如需有關在 ActionScript 中使用置換對應的詳細資訊，請參閱第 226 頁「以濾鏡處理顯示物件」。

設定像素值來建立點陣圖影像

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

置換對應濾鏡的重要觀點是，濾鏡實際上包含了兩個影像。其中一個影像 (即來源影像) 是實際由濾鏡修改的影像。在此樣本中，其來源影像是名為 sphere 的 Bitmap 實體。由濾鏡使用的其它影像稱為對應影像。對應影像實際上並未顯示在螢幕上。每個像素的顏色都會用來當做是對置換函數的輸入。也就是說，對應影像內的某些 x, y 座標會決定要在來源影像內的該 x,y 座標上套用多少置換量 (在適當位置的實體移動)。

因此，若要使用置換對應濾鏡來建立球體效果，樣本需要適當的對應影像：一個具有灰色背景和一個以單一顏色 (紅色) 漸層填滿的圓形，水平地由暗顯示到亮，如此處所示：



由於此樣本內只使用了一個對應影像和濾鏡，因此對應影像只會在 imageLoadComplete() 方法中建立一次 (換句話說，即是在外部影像完成載入時)。在樣本中，會呼叫 MoonSphere 類別的 createFisheyeMap() 方法，建立名為 fisheyeLens 的對應影像：

```
var fisheyeLens:BitmapData = createFisheyeMap(radius);
```

在 createFisheyeMap() 方法內部，對應影像會使用 BitmapData 類別的 setPixel() 方法一次繪製一個像素。createFisheyeMap() 方法的完整程式碼列示如下，隨後會逐步討論其運作的方式：

```
private function createFisheyeMap(radius:int):BitmapData
{
    var diameter:int = 2 * radius;

    var result:BitmapData = new BitmapData(diameter,
                                           diameter,
                                           false,
                                           0x808080);

    // Loop through the pixels in the image one by one
    for (var i:int = 0; i < diameter; i++)
    {
        for (var j:int = 0; j < diameter; j++)
        {
            // Calculate the x and y distances of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctX:Number = (i - radius) / radius;
            var pctY:Number = (j - radius) / radius;

            // Calculate the linear distance of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);

            // If the current pixel is inside the circle,
            // set its color.
            if (pctDistance < 1)
            {
                // Calculate the appropriate color depending on the
                // distance of this pixel from the center of the circle.
                var red:int;
                var green:int;
                var blue:int;
                var rgb:uint;
                red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
                green = 0;
                blue = 0;
                rgb = (red << 16 | green << 8 | blue);
                // Set the pixel to the calculated color.
                result.setPixel(i, j, rgb);
            }
        }
    }
    return result;
}
```

首先，當在接收所呼叫方法時，參數 radius 會指定要建立圓形影像的半徑。接著，程式碼便會在將要繪製的圓形上建立 BitmapData 物件。該物件 (名為 result) 最終會當做方法的傳回值傳回。如下列程式碼片段所示，這個 result BitmapData 實體是以圓形直徑大小的寬度和高度所建立，不具透明度 (第三個參數值為 false)，並已預先填入顏色 0x808080 (中級的灰色)：

```
var result:BitmapData = new BitmapData(diameter,
                                       diameter,
                                       false,
                                       0x808080);
```

接下來，程式碼會使用兩個迴圈來重複影像的每個像素。外部迴圈會從左至右通過影像的每一欄 (使用變數 i 代表目前操作的像素水平位置)，而內部迴圈則從上到下通過目前欄的每一個像素 (使用變數 j 表示目前像素的垂直位置)。迴圈的程式碼 (包含省略的內部迴圈內容) 如下：

```
for (var i:int = 0; i < diameter; i++)
{
    for (var j:int = 0; j < diameter; j++)
    {
        ...
    }
}
```

隨著迴圈一個接一個對像素執行循環，使得每一個像素都會計算出一個值 (對應影像內像素的顏色值)。此程序包含四個步驟：

1 程式碼會沿著 x 軸計算目前像素與圓形中心點的距離 (i - radius)。該值可除以半徑值，使其以半徑的百分比表示，而不是以絕對距離表示 ((i - radius) / radius)。該百分比值會儲存在名為 pctX 的變數內，而 y 軸的相等值會經過計算並儲存在變數 pctY 中，如同此程式碼所示：

```
var pctX:Number = (i - radius) / radius;
var pctY:Number = (j - radius) / radius;
```

2 使用了標準的三角函數公式 (畢氏定理)，圓形中心和點目前所在位置間的線性距離皆是從 pctX 和 pctY 計算而來。該值會儲存在名為 pctDistance 的變數內，如此處所示：

```
var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);
```

3 接下來，程式碼會檢查距離百分比是否小於 1 (代表半徑的 100%，或換句話說，如果考慮的像素位於圓形半徑內)。如果像素落在圓形的內部，則會為其指定已計算的顏色值 (因篇幅有限故省略，但已在步驟 4 中說明)；如果像素未落在圓形的內部，則該像素不會有任何變化，因此其顏色會保留為預設的中級灰色：

```
if (pctDistance < 1)
{
    ...
}
```

4 對於落於圓形內部的像素而言，其顏色值已計算出。最終的顏色將是紅色的陰影部分，範圍從圓形左邊緣的黑色 (0% 的紅色) 到圓形右邊緣的亮紅色 (100%)。顏色值最初會以三個部分進行計算 (紅色、綠色和藍色)，如此處所示：

```
red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
green = 0;
blue = 0;
```

請注意，只有顏色的紅色部分 (即變數 red) 才會真正擁有值。為了在此處說明而顯示的綠色和藍色值 (即變數 green 和 blue) 可被省略。由於此方法的用途是要建立一個內含紅色漸層的圓形，因此不需要使用綠色或藍色值。

一旦確定了三種個別的顏色值，便會使用標準的位元轉換演算法將其結合為單一的整數顏色值，如此程式碼中顯示：

```
rgb = (red << 16 | green << 8 | blue);
```

最後，便會得到已計算的顏色值 (該值實際上是由目前的像素使用 result BitmapData 物件的 setPixel() 方法所指定)，如此處所示：

```
result.setPixel(i, j, rgb);
```

點陣圖影像的非同步解碼

Flash Player 11 以及更新的版本，**Adobe AIR 2.6** 以及更新的版本

當您使用點陣圖影像時，可以用非同步的方式解碼和載入點陣圖影像，以改善應用程式的認知效能。在許多情況下，以非同步方式解碼點陣圖影像可能需要花費與同步解碼影像相同的時間。不過，點陣圖影像會在關聯的 Loader 物件傳送 COMPLETE 事件之前，在不同的執行緒中解碼。因此，您可以在載入它們後，以非同步方式解碼較大的影像。

flash.system 套件中的 ImageDecodingPolicy 類別，可讓您指定點陣圖載入配置。預設載入配置為同步。

點陣圖解碼原則	點陣圖載入配置	說明
ImageDecodingPolicy.ON_DEMAND	同步	當存取影像資料時會解碼載入的影像。 使用此原則以解碼較小的影像。當您的應用程式不依賴複雜的特效和轉場時，也可以使用此原則。
ImageDecodingPolicy.ON_LOAD	非同步	在傳送 COMPLETE 事件前，會在加載時解碼載入的影像。 適用於較大的影像 (大於 10 MP)。您開發的 AIR 行動應用程式使用頁面轉變時，請使用此點陣圖載入原則以改善應用程式效能。

備註：如果要載入的檔案是點陣圖影像，而使用的解碼原則是 ON_LOAD，則在傳送 COMPLETE 事件前，會以非同步方式解碼影像。

下列程式碼顯示 ImageDecodingPolicy 類別的用法：

```
var loaderContext:LoaderContext = new LoaderContext();
loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_LOAD
var loader:Loader = new Loader();
loader.load(new URLRequest("http://www.adobe.com/myimage.png"), loaderContext);
```

您仍然可以透過 Loader.load() 與 Loader.loadBytes() 方法使用 ON_DEMAND 解碼。不過，所有將 LoaderContext 物件當做引數使用的其他方法，都會忽略任何傳回的 ImageDecodingPolicy 值。

下列範例顯示同步和非同步解碼點陣圖影像的差異：

```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.system.ImageDecodingPolicy;
    import flash.system.LoaderContext;

    public class AsyncTest extends Sprite
    {
        private var loaderContext:LoaderContext;
        private var loader:Loader;
        private var urlRequest:URLRequest;
        public function AsyncTest()
        {
            //Load the image synchronously
            loaderContext = new LoaderContext();
            //Default behavior.
            loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_DEMAND;
            loader = new Loader();
            loadImageSync();

            //Load the image asynchronously
            loaderContext = new LoaderContext();
            loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_LOAD;
            loader = new Loader();
            loadImageASync();
        }

        private function loadImageASync():void{
            trace("Loading image asynchronously...");
            urlRequest = new URLRequest("http://www.adobe.com/myimage.png");
            urlRequest.useCache = false;
```

```
            loader.load(urlRequest, loaderContext);
            loader.contentLoaderInfo.addEventListener
                (Event.COMPLETE, onAsyncLoadComplete);
        }

        private function onAsyncLoadComplete(event:Event):void{
            trace("Async. Image Load Complete");
        }

        private function loadImageSync():void{
            trace("Loading image synchronously...");
            urlRequest = new URLRequest("http://www.adobe.com/myimage.png");
            urlRequest.useCache = false;
            loader.load(urlRequest, loaderContext);
            loader.contentLoaderInfo.addEventListener
                (Event.COMPLETE, onSyncLoadComplete);
        }

        private function onSyncLoadComplete(event:Event):void{
            trace("Sync. Image Load Complete");
        }
    }
}
```

如需不同解碼原則效果的示範，請參閱 Thibaud Imbert：Adobe Flash 執行階段中的非同步點陣圖解碼

第 14 章 以濾鏡處理顯示物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

根據以往的經驗，如果要將濾鏡效果套用到點陣圖影像中，得靠 Adobe Photoshop® 和 Adobe Fireworks® 之類的專業影像編輯軟體才辦得到。ActionScript 3.0 包含 flash.filters 封裝，含有一系列的點陣圖效果濾鏡類別。這些效果可讓開發人員以程式設計方式，將濾鏡套用至點陣圖與顯示物件上，達到圖像處理應用程式所能提供的許多相同效果。

以濾鏡處理顯示物件的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

凸顯應用程式的其中一種方式，就是加入簡單的圖像效果。您可在相片背景加上投影以建立 3D 幻覺，或是在按鈕四周加上光暈來代表按鈕為作用中。ActionScript 3.0 包含了十個濾鏡，可讓您套用至任何顯示物件或是 BitmapData 實體。內建濾鏡包括基本濾鏡，例如投影與光暈濾鏡，以及複雜濾鏡，例如置換對應濾鏡與迴旋濾鏡。

備註：除了內建濾鏡之外，您亦可使用 Pixel Bender 設計自訂濾鏡與效果。請參閱第 253 頁 「使用 Pixel Bender 著色器」。

重要概念與術語

下列參考清單包含您在建立濾鏡時可能會遇到的相關重要術語：

斜角 在邊框的某兩邊加上會變亮的像素，並在相對的另外兩邊加上會變暗的像素。此效果可建立三維邊框外觀。此效果讓按鈕之類圖像看起來就像是突起或是凹下。

迴旋 將每個像素的值與鄰近像素的部分或全部值加起來，透過不同的比例來扭曲影像中的像素。

置換 將影像中的像素位移或移動至新的位置上。

矩陣 一個包含數字的方格，可用來執行特定的數學計算，例如將方格中的數字套用至不同的值中，再加總結果。

更多說明主題

flash.filters package

flash.display.DisplayObject.filters

flash.display.BitmapData.applyFilter()

建立與套用濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

濾鏡可讓您將一些包括投影、斜角與模糊的效果套用至點陣圖與顯示物件上。每個濾鏡都會定義為一個類別，因此套用濾鏡時會與建立濾鏡物件實體有關，與建構其它物件是一樣的意思。一旦您建立了濾鏡物件實體，就可以使用物件的 filters 屬性輕易地將它套用至顯示物件上，如果是 BitmapData 物件，則請改用 applyFilter() 方法。

建立濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您只需呼叫所選濾鏡類別的建構函式方法，就可以建立濾鏡物件。例如，您可以使用下列程式碼來建立 DropShadowFilter 物件：

```
import flash.filters.DropShadowFilter;
var myFilter:DropShadowFilter = new DropShadowFilter();
```

雖然 DropShadowFilter() 建構函式並未出現在上述程式碼中，但是它就像其它所有的濾鏡類別建構函式一樣，可接受好幾種選用的參數，以便用來自訂濾鏡效果的外觀。

套用濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

一旦您建構了濾鏡物件，就可以將之套用至顯示物件或 BitmapData 物件上；套用濾鏡的方法取決於您要套用濾鏡的目標物件。

將濾鏡套用至顯示物件

當您將濾鏡效果套用至顯示物件時，事實上是透過 filters 屬性來套用這些濾鏡。顯示物件的 filters 屬性是一個 Array 實體，當中的元素就是套用至顯示物件中的濾鏡物件。若要將單一濾鏡套用至顯示物件，首先要建立濾鏡實體、將它加入 Array 實體，接著將該 Array 物件指定給顯示物件的 filters 屬性：

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.filters.DropShadowFilter;

// Create a bitmapData object and render it to screen
var myBitmapData:BitmapData = new BitmapData(100,100,false,0xFFFF3300);
var myDisplayObject:Bitmap = new Bitmap(myBitmapData);
addChild(myDisplayObject);

// Create a DropShadowFilter instance.
var dropShadow:DropShadowFilter = new DropShadowFilter();

// Create the filters array, adding the filter to the array by passing it as
// a parameter to the Array() constructor.
var filtersArray:Array = new Array(dropShadow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

如果您想要為物件指定多個濾鏡，只需在為 filters 屬性指定濾鏡之前，先將所有的濾鏡加入 Array 實體即可。您可以將多個物件當做參數傳遞至其建構函式，藉以將這些物件加入 Array 實體中。例如，下列程式碼將一個斜角濾鏡與一個光暈濾鏡套用至先前所建立的顯示物件上：

```
import flash.filters.BevelFilter;
import flash.filters.GlowFilter;

// Create the filters and add them to an array.
var bevel:BevelFilter = new BevelFilter();
var glow:GlowFilter = new GlowFilter();
var filtersArray:Array = new Array(bevel, glow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

當您想要建立包含濾鏡的陣列時，可以使用 new Array() 建構函式 (如上一個範例所示) 來建立，或者使用 Array 常值語法，以方括弧 ([]) 括住這些濾鏡。例如，下列這行程式碼：

```
var filters:Array = new Array(dropShadow, blur);
```

與接下來的程式碼作用相同：

```
var filters:Array = [dropShadow, blur];
```

如果您將多個濾鏡套用至顯示物件，則這些濾鏡會以累加、連續的方式來套用。例如，假設某個濾鏡陣列具有兩個元素，則會先加入斜角濾鏡，再加入投影濾鏡，而投影濾鏡會同時套用至斜角濾鏡與顯示物件上。這是因為投影濾鏡位於濾鏡陣列的第二個位置。如果您想要以非累加的方式來套用濾鏡，請將每個濾鏡分別套用到新的顯示物件副本。

如果您只是單純地將一或幾個濾鏡指定給顯示物件，則可以先建立濾鏡實體，並以單一陳述式將它指定給物件。例如，下列這一行程式碼會將模糊濾鏡套用至名為 myDisplayObject 的顯示物件：

```
myDisplayObject.filters = [new BlurFilter()];
```

上一段程式碼採用 Array 常值語法 (方括弧) 來建立 Array 實體，並建立一個 BlurFilter 實體做為 Array 的元素，接著將該 Array 指定給名為 myDisplayObject 之顯示物件的 filters 屬性。

移除顯示物件上的濾鏡

移除顯示物件上的所有濾鏡，作法與指定一個 null 值給 filters 屬性一樣容易：

```
myDisplayObject.filters = null;
```

如果您已經將好幾個濾鏡套用至物件上，而且只想要移除其中一個濾鏡，則必須透過幾個步驟，變更 filters 屬性陣列。如需詳細資訊，請參閱第 228 頁 「使用濾鏡的潛在問題」。

將濾鏡套用至 **BitmapData** 物件

將濾鏡套用至 BitmapData 物件需要搭配使用 BitmapData 物件的 applyFilter() 方法：

```
var rect:Rectangle = new Rectangle();
var origin:Point = new Point();
myBitmapData.applyFilter(sourceBitmapData, rect, origin, new BlurFilter());
```

applyFilter() 方法可以將濾鏡套用至來源 BitmapData 物件，以產生一個以濾鏡處理的新影像。此方法並不會修改原始來源影像；反之，將濾鏡套用到來源影像後的結果會儲存在用來呼叫 applyFilter() 方法的 BitmapData 實體。

濾鏡的運作方式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

顯示物件會藉由將原始物件的副本快取為透明點陣圖來加上濾鏡。

一旦將濾鏡套用至顯示物件，只要物件包含有效的濾鏡清單，執行階段就會將物件快取為點陣圖。接著，這個來源點陣圖就會被當成所有後續套用之濾鏡效果的原始影像來使用。

每個顯示物件通常包含兩個點陣圖：其中一個是原始、未經濾鏡處理的來源顯示物件，另一個是經過濾鏡處理後的最終影像。顯示影像時是使用最終影像。只要不變更顯示物件，就不需要更新最終影像。

使用濾鏡的潛在問題

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用濾鏡時，請注意幾點可能讓您混淆或困擾的問題成因。

濾鏡和點陣圖快取

若要將濾鏡套用至顯示物件，必須啟用該物件的點陣圖快取功能。若您套用濾鏡的物件，其 cacheAsBitmap 屬性設為 false，則該物件的 cacheAsBitmap 屬性會自動設為 true。若您稍後從顯示物件移除所有濾鏡，cacheAsBitmap 屬性會重設為原本的設定。

在執行階段變更濾鏡

如果顯示物件已經套用了一個或多個濾鏡，您就不可以將其它濾鏡加入 filters 屬性陣列中，或從中移除濾鏡來變更濾鏡組。若要加入或變更要套用的濾鏡組，您必須變更個別的陣列，然後將該陣列指定給顯示物件的 filters 屬性，才能讓濾鏡套用到物件上。最簡單的方式是將 filters 屬性陣列讀入 Array 變數，然後修改這個暫存陣列。接著，您可以將此陣列重新指定給顯示物件的 filters 屬性。在較為複雜的案例中，您可能需要保留濾鏡的個別主陣列。請對該主濾鏡陣列進行任何變更，然後在每個變更後，將主陣列重新指定給顯示物件的 filters 屬性。

加入其它濾鏡

下列程式碼示範將其它濾鏡加入已經套用一或多個濾鏡之顯示物件的程序。首先，將光暈濾鏡套用至名為 myDisplayObject 的顯示物件；之後，每當按一下顯示物件時，就呼叫 addFilters() 函數。兩個額外的濾鏡就會透過這個函數套用至 myDisplayObject：

```
import flash.events.MouseEvent;
import flash.filters.*;

myDisplayObject.filters = [new GlowFilter()];

function addFilters(event:MouseEvent):void
{
    // Make a copy of the filters array.
    var filtersCopy:Array = myDisplayObject.filters;

    // Make desired changes to the filters (in this case, adding filters).
    filtersCopy.push(new BlurFilter());
    filtersCopy.push(new DropShadowFilter());

    // Apply the changes by reassigning the array to the filters property.
    myDisplayObject.filters = filtersCopy;
}

myDisplayObject.addEventListener(MouseEvent.CLICK, addFilters);
```

從一組濾鏡移除一個濾鏡

如果顯示物件已經套用多個濾鏡，而且您想要在繼續將其它濾鏡套用到該物件時移除其中一個濾鏡，您可以將濾鏡複製到暫存陣列、從該陣列移除不想要的濾鏡，然後將暫存陣列重新指定給顯示物件的 filters 屬性。從任何陣列移除一個或多個元素的數種方式詳述於第 26 頁「擷取值以及移除陣列元素」中。

最直接的方式，便是移除物件最上層的濾鏡 (套用到物件的最後一個濾鏡)。您可以使用 Array 類別的 pop() 方法來移除陣列中的濾鏡：

```
// Example of removing the top-most filter from a display object
// named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the last element from the Array (the top-most filter).
tempFilters.pop();

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

同樣地，若要移除最下層的濾鏡 (套用到物件的第一個濾鏡)，您可以使用相同的程式碼替代 Array 類別的 shift() 方法來取代 pop() 方法。

若要從濾鏡陣列的中間移除濾鏡 (假設該陣列有兩個以上的濾鏡)，您可以使用 splice() 方法。您必須知道要移除之濾鏡的索引 (陣列中的位置)。例如，下列程式碼會從顯示物件移除第二個濾鏡 (索引 1 的濾鏡)：

```
// Example of removing a filter from the middle of a stack of filters
// applied to a display object named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the second filter from the array. It's the item at index 1
// because Array indexes start from 0.
// The first "1" indicates the index of the filter to remove; the
// second "1" indicates how many elements to remove.
tempFilters.splice(1, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

判斷濾鏡的索引

您必須知道要從陣列移除哪個濾鏡，才會知道該濾鏡的索引。此外，您也需要知道 (根據設計應用程式的方式) 或計算要移除的濾鏡索引。

最好的方法是在設計您的應用程式時，讓您要移除的濾鏡永遠位於濾鏡組中的相同位置。例如，如果所擁有的單一顯示物件已套用迴旋濾鏡與投影濾鏡 (以該順序)，而且您想要移除投影濾鏡但保留迴旋濾鏡，濾鏡會位於已知的位置 (最上層的濾鏡)，因此您可以事先知道要使用的 Array 方法 (此範例是使用 Array.pop() 來移除投影濾鏡)。

如果您要移除的濾鏡永遠是特定類型，但不一定位於濾鏡組中的相同位置，您可以檢查陣列中的每個濾鏡資料類型來決定要移除的濾鏡。例如，下列程式碼會判斷哪個濾鏡組是光暈濾鏡，並從濾鏡組中移除該濾鏡。

```
// Example of removing a glow filter from a set of filters, where the
//filter you want to remove is the only GlowFilter instance applied
// to the filtered object.

var tempFilters:Array = filteredObject.filters;

// Loop through the filters to find the index of the GlowFilter instance.
var glowIndex:int;
var numFilters:int = tempFilters.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (tempFilters[i] is GlowFilter)
    {
        glowIndex = i;
        break;
    }
}

// Remove the glow filter from the array.
tempFilters.splice(glowIndex, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

在較為複雜的案例中，例如，如果要移除的濾鏡是在執行階段選取的，最好的方法是保留當做主濾鏡清單之濾鏡陣列的個別、持續副本。每當您要變更濾鏡組時，請變更主清單，然後套用該濾鏡陣列，做為顯示物件的 filters 屬性。

例如，下面列出的程式碼會將多個迴旋濾鏡套用至顯示物件來建立不同的視覺效果，並在稍後於應用程式中移除其中一個濾鏡，而保留其它的濾鏡。在此案例中，程式碼會保留濾鏡陣列的主副本，以及要移除之濾鏡的參考。尋找並移除特定的濾鏡與前述方法類似，但是不會製作濾鏡陣列的暫存副本，而是直接操作主副本，然後套用到顯示物件。

```
// Example of removing a filter from a set of
// filters, where there may be more than one
// of that type of filter applied to the filtered
// object, and you only want to remove one.

// A master list of filters is stored in a separate,
// persistent Array variable.
var masterFilterList:Array;

// At some point, you store a reference to the filter you
// want to remove.
var filterToRemove:ConvolutionFilter;

// ... assume the filters have been added to masterFilterList,
// which is then assigned as the filteredObject.filters:
filteredObject.filters = masterFilterList;

// ... later, when it's time to remove the filter, this code gets called:

// Loop through the filters to find the index of masterFilterList.
var removeIndex:int = -1;
var numFilters:int = masterFilterList.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (masterFilterList[i] == filterToRemove)
    {
        removeIndex = i;
        break;
    }
}

if (removeIndex >= 0)
{
    // Remove the filter from the array.
    masterFilterList.splice(removeIndex, 1);

    // Apply the new set of filters to the display object.
    filteredObject.filters = masterFilterList;
}
```

利用這個方法 (在您比較已儲存在濾鏡陣列中的項目濾鏡參考來判斷要移除的濾鏡時)，您「必須」保留濾鏡陣列的個別副本；如果您是在從顯示物件之 filters 屬性複製的暫存陣列中，比較已儲存的元素濾鏡參考，則該程式碼沒有作用。這是因為當您在內部將陣列指定給 filters 屬性時，執行階段會製作陣列中每個濾鏡物件的副本。這些副本 (而非原始物件) 會套用到顯示物件，而且當您將 filters 屬性讀入暫存陣列時，暫存陣列中會包含已複製之濾鏡物件的參考，而非原始濾鏡物件的參考。因此，如果您在上述範例中，透過比較 filterToRemove 與暫存濾鏡陣列中的濾鏡來嘗試判斷其索引，則不會找到任何相符項目。

濾鏡與物件變形

位於顯示物件的矩形範圍框外面的無濾鏡區域 (如投影) 會被視為具有感應區偵測用途 (判斷實體與其它實體重疊或相交) 的表面之一部分。由於 DisplayObject 類別的感應區偵測方法屬於向量架構，您無法在點陣圖結果上執行感應區偵測。例如，如果您將斜角濾鏡套用到按鈕實體，實體的斜角部分就沒有感應區偵測作用。

濾鏡不支援縮放、旋轉和傾斜；當有濾鏡的顯示物件本身縮放時 (scaleX 和 scaleY 不是 100%)，濾鏡效果並不會跟著實體一起縮放。這表示實體的原始形狀會旋轉、縮放或傾斜，但是濾鏡並未跟著實體一起旋轉、縮放或傾斜。

您可以將含有濾鏡的實體做成動畫以建立具真實感的特效，或巢狀放置實體並使用 BitmapData 類別進行濾鏡的動畫以達到此效果。

濾鏡和點陣圖物件

當您將任何濾鏡套用至 BitmapData 物件時，cacheAsBitmap 屬性就會自動設為 true。這樣一來，濾鏡會實際套用至物件副本上，而不是套用至原始物件上。

接著，此副本會放置到主要顯示中 (於原始物件上) 且儘可能地接近最靠近的像素。如果原始點陣圖的邊界變更，就會從原始物件重新建立含濾鏡的點陣圖副本，而不是加以延伸或扭曲。

如果您清除了顯示物件的所有濾鏡，會將 cacheAsBitmap 屬性重設為套用濾鏡之前的屬性。

可用的顯示濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 包含了十個濾鏡類別，可讓您套用至顯示物件及 BitmapData 物件中：

- 斜角濾鏡 (BevelFilter 類別)
- 模糊濾鏡 (BlurFilter 類別)
- 投影濾鏡 (DropShadowFilter 類別)
- 光暈濾鏡 (GlowFilter 類別)
- 漸層斜角濾鏡 (GradientBevelFilter 類別)
- 漸層光暈濾鏡 (GradientGlowFilter 類別)
- 顏色矩陣濾鏡 (ColorMatrixFilter 類別)
- 迴旋濾鏡 (ConvolutionFilter 類別)
- 置換對應濾鏡 (DisplacementMapFilter 類別)
- 著色器濾鏡 (ShaderFilter 類別)

前 6 個濾鏡都是用來建立單一特定效果簡單濾鏡，可讓您針對這些效果進行自訂。您可以使用 ActionScript 來應用這 6 個濾鏡，也可以透過 Flash Professional 中的 「濾鏡」面板來將濾鏡套用至物件。如此一來，就算您正透過 ActionScript 來套用這些濾鏡，如果手邊剛好有 Flash Professional，也可以使用其視覺介面，快速地嘗試各種不同的濾鏡與設定，找出建立所需效果的方式。

後面 4 個濾鏡則是只能透過 ActionScript 來使用。顏色矩陣濾鏡、迴旋濾鏡、置換對應濾鏡以及著色器濾鏡這些濾鏡在可用來建立效果的類型中更有彈性。這些濾鏡提供強大的功能和彈性，而不是只針對單一效果進行最佳化。例如，如果您想要建立如模糊、浮雕、銳利化、尋找顏色邊緣、變形等效果，只需選取不同的迴旋濾鏡矩陣值即可。

不管是簡單或是複雜的濾鏡，每一種都可以透過各自的屬性來自訂。一般來說，設定濾鏡屬性的方式有兩種。您可以將參數值傳遞給濾鏡物件的建構函式，設定所有的濾鏡屬性。或者，不管您是否藉由傳遞參數的方式來設定濾鏡屬性，稍後都可以再設定濾鏡物件的屬性值以調整濾鏡。列出的程式碼範例大部分都會直接設定屬性，方便使用者跟著學習。然而，您也可以將屬性值做為參數傳遞至濾鏡物件的建構函式中，通常不只能夠產生相同的結果，程式碼也會比較精簡。如需每個濾鏡、屬性及其建構函式參數特定的詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中有關 flash.filters 套件的列示項目。

斜角濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

BevelFilter 類別可讓您將 3D 斜角化邊緣加入至已套用濾鏡的物件。這個濾鏡可讓物件的直角或邊緣看起來像是經過雕鑿或加上斜角處理一樣。

BevelFilter 類別屬性可讓您自訂斜角的外觀。您可以設定反白標示與陰影顏色、斜角邊緣模糊、斜角角度，以及斜角邊緣置放等；您甚至可以建立去底色效果。

下列範例將載入一個外部影像，並在其上套用斜角濾鏡。

```
import flash.display.*;
import flash.filters.BevelFilter;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.net.URLRequest;

// Load an image onto the Stage.
var imageLoader:Loader = new Loader();
var url:String = "http://www.helpexamples.com/flash/images/image3.jpg";
var urlReq:URLRequest = new URLRequest(url);
imageLoader.load(urlReq);
addChild(imageLoader);

// Create the bevel filter and set filter properties.
var bevel:BevelFilter = new BevelFilter();

bevel.distance = 5;
bevel.angle = 45;
bevel.highlightColor = 0xFFFF00;
bevel.highlightAlpha = 0.8;
bevel.shadowColor = 0x666666;
bevel.shadowAlpha = 0.8;
bevel.blurX = 5;
bevel.blurY = 5;
bevel.strength = 5;
bevel.quality = BitmapFilterQuality.HIGH;
bevel.type = BitmapFilterType.INNER;
bevel.knockout = false;

// Apply filter to the image.
imageLoader.filters = [bevel];
```

模糊濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

BlurFilter 類別可針對顯示物件及其內容加以塗抹或進行模糊處理。如果您想讓某個物件看起來失焦，或是想要模擬類似動態模糊的物件快速移動狀態，則使用模糊濾鏡就可以達到不錯的效果。您可以將模糊濾鏡的 quality 屬性設為 「低」，模擬淡淡的鏡頭失焦效果。如果將 quality 屬性設為 「高」，則會得出近似高斯模糊的平順模糊效果。

下列範例將使用 Graphics 類別的 drawCircle() 方法來建立一個圓形物件，並在其上套用模糊濾鏡：

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.BlurFilter;

// Draw a circle.
var redDotCutout:Sprite = new Sprite();
redDotCutout.graphics.lineStyle();
redDotCutout.graphics.beginFill(0xFF0000);
redDotCutout.graphics.drawCircle(145, 90, 25);
redDotCutout.graphics.endFill();

// Add the circle to the display list.
addChild(redDotCutout);

// Apply the blur filter to the rectangle.
var blur:BlurFilter = new BlurFilter();
blur.blurX = 10;
blur.blurY = 10;
blur.quality = BitmapFilterQuality.MEDIUM;
redDotCutout.filters = [blur];
```

投影濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

投影濾鏡會讓目標物件上方看起來好像多出一個光源。您可以修改此光源的位置與明暗度，產生不同的投影效果。

DropShadowFilter 類別所使用的演算法與模糊濾鏡所使用的類似。最大的差異在於投影濾鏡多出了幾項屬性，可讓您加以修改以模擬不同的光源特質，例如 Alpha、顏色、偏移與亮度。

投影濾鏡同時可讓您將自訂的變形選項套用到投影樣式中，包括內 / 外陰影與去底色模式 (亦稱為挖空)。

下列程式碼將建立一個方塊 Sprite，並在其上套用投影：

```
import flash.display.Sprite;
import flash.filters.DropShadowFilter;

// Draw a box.
var boxShadow:Sprite = new Sprite();
boxShadow.graphics.lineStyle(1);
boxShadow.graphics.beginFill(0xFF3300);
boxShadow.graphics.drawRect(0, 0, 100, 100);
boxShadow.graphics.endFill();
addChild(boxShadow);

// Apply the drop shadow filter to the box.
var shadow:DropShadowFilter = new DropShadowFilter();
shadow.distance = 10;
shadow.angle = 25;

// You can also set other properties, such as the shadow color,
// alpha, amount of blur, strength, quality, and options for
// inner shadows and knockout effects.

boxShadow.filters = [shadow];
```

光暈濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

GlowFilter 類別會將光源效果套用至顯示物件上，讓它看起來像是背後有光照射過來，而在物件周圍形成淡淡的光暈。

光暈濾鏡與投影濾鏡很類似，兩者都包含可用來修改光源距離、角度與顏色，以產生各種不同效果的屬性。GlowFilter 同時提供數種可供修改光暈樣式的選項，包含內或外光暈以及去底色模式。

下列程式碼將使用 Sprite 類別來建立一個十字架，並在其上套用光暈濾鏡：

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.GlowFilter;

// Create a cross graphic.
var crossGraphic:Sprite = new Sprite();
crossGraphic.graphics.lineStyle();
crossGraphic.graphics.beginFill(0xCCCC00);
crossGraphic.graphics.drawRect(60, 90, 100, 20);
crossGraphic.graphics.drawRect(100, 50, 20, 100);
crossGraphic.graphics.endFill();
addChild(crossGraphic);

// Apply the glow filter to the cross shape.
var glow:GlowFilter = new GlowFilter();
glow.color = 0x009922;
glow.alpha = 1;
glow.blurX = 25;
glow.blurY = 25;
glow.quality = BitmapFilterQuality.MEDIUM;

crossGraphic.filters = [glow];
```

漸層斜角濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

GradientBevelFilter 類別可讓您對顯示物件或 BitmapData 物件套用增強的斜角特效。在斜角上使用漸層顏色可大大改善斜角的空間深度，讓邊緣看起來更真實、更立體。

下列程式碼將使用 Shape 類別的 drawRect() 方法來建立一個矩形物件，並在其上套用漸層斜角濾鏡。

```
import flash.display.Shape;
import flash.filters.BitmapFilterQuality;
import flash.filters.GradientBevelFilter;

// Draw a rectangle.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Apply a gradient bevel to the rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter();

gradientBevel.distance = 8;
gradientBevel.angle = 225; // opposite of 45 degrees
gradientBevel.colors = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
gradientBevel.alphas = [1, 0, 1];
gradientBevel.ratios = [0, 128, 255];
gradientBevel.blurX = 8;
gradientBevel.blurY = 8;
gradientBevel.quality = BitmapFilterQuality.HIGH;

// Other properties let you set the filter strength and set options
// for inner bevel and knockout effects.

box.filters = [gradientBevel];

// Add the graphic to the display list.
addChild(box);
```

漸層光暈濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

GradientGlowFilter 類別可讓您對顯示物件或 BitmapData 物件套用增強的光暈特效。此效果可讓您更容易掌控光暈顏色，進而產生更真實的光暈效果。此外，漸層光暈濾鏡可讓您將漸層光暈套用至物件的邊緣內部、外部，或邊緣上方。

下列範例會在「舞台」上繪製一個圓形，並將漸層光暈濾鏡套用至此圓形。當您將滑鼠移至右側並接著往下移時，模糊量就會分別往水平及垂直方向增加。此外，任何時候，只要您在「舞台」上按一下，就可增加模糊的強度。

```
import flash.events.MouseEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.GradientGlowFilter;

// Create a new Shape instance.
var shape:Shape = new Shape();

// Draw the shape.
shape.graphics.beginFill(0xFF0000, 100);
shape.graphics.moveTo(0, 0);
shape.graphics.lineTo(100, 0);
shape.graphics.lineTo(100, 100);
shape.graphics.lineTo(0, 100);
shape.graphics.lineTo(0, 0);
shape.graphics.endFill();

// Position the shape on the Stage.
addChild(shape);
shape.x = 100;
```

```
shape.y = 100;

// Define a gradient glow.
var gradientGlow:GradientGlowFilter = new GradientGlowFilter();
gradientGlow.distance = 0;
gradientGlow.angle = 45;
gradientGlow.colors = [0x000000, 0xFF0000];
gradientGlow.alphas = [0, 1];
gradientGlow.ratios = [0, 255];
gradientGlow.blurX = 10;
gradientGlow.blurY = 10;
gradientGlow.strength = 2;
gradientGlow.quality = BitmapFilterQuality.HIGH;
gradientGlow.type = BitmapFilterType.OUTER;

// Define functions to listen for two events.
function onClick(event:MouseEvent):void
{
    gradientGlow.strength++;
    shape.filters = [gradientGlow];
}

function onMouseMove(event:MouseEvent):void
{
    gradientGlow.blurX = (stage.mouseX / stage.stageWidth) * 255;
    gradientGlow.blurY = (stage.mouseY / stage.stageHeight) * 255;
    shape.filters = [gradientGlow];
}
stage.addEventListener(MouseEvent.CLICK, onClick);
stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
```

範例：結合基本濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列程式碼範例將使用幾個基本濾鏡，再結合用來建立重複動作的計時器，共同建立模擬紅綠燈的動畫。

```
import flash.display.Shape;
import flash.events.TimerEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;
import flash.filters.GradientBevelFilter;
import flash.utils.Timer;

var count:Number = 1;
var distance:Number = 8;
var angleInDegrees:Number = 225; // opposite of 45 degrees
var colors:Array = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
var blurX:Number = 8;
var blurY:Number = 8;
var strength:Number = 1;
var quality:Number = BitmapFilterQuality.HIGH;
var type:String = BitmapFilterType.INNER;
var knockout:Boolean = false;

// Draw the rectangle background for the traffic light.
var box:Shape = new Shape();
```

```
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Draw the 3 circles for the three lights.
var stopLight:Shape = new Shape();
stopLight.graphics.lineStyle();
stopLight.graphics.beginFill(0xFF0000);
stopLight.graphics.drawCircle(145,90,25);
stopLight.graphics.endFill();

var cautionLight:Shape = new Shape();
cautionLight.graphics.lineStyle();
cautionLight.graphics.beginFill(0xFF9900);
cautionLight.graphics.drawCircle(145,150,25);
cautionLight.graphics.endFill();

var goLight:Shape = new Shape();
goLight.graphics.lineStyle();
goLight.graphics.beginFill(0x00CC00);
goLight.graphics.drawCircle(145,210,25);
goLight.graphics.endFill();

// Add the graphics to the display list.
addChild(box);
addChild(stopLight);
addChild(cautionLight);
addChild(goLight);

// Apply a gradient bevel to the traffic light rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(distance, angleInDegrees, colors, alphas,
ratios, blurX, blurY, strength, quality, type, knockout);
box.filters = [gradientBevel];

// Create the inner shadow (for lights when off) and glow
// (for lights when on).
var innerShadow:DropShadowFilter = new DropShadowFilter(5, 45, 0, 0.5, 3, 3, 1, 1, true, false);
var redGlow:GlowFilter = new GlowFilter(0xFF0000, 1, 30, 30, 1, 1, false, false);
var yellowGlow:GlowFilter = new GlowFilter(0xFF9900, 1, 30, 30, 1, 1, false, false);
var greenGlow:GlowFilter = new GlowFilter(0x00CC00, 1, 30, 30, 1, 1, false, false);

// Set the starting state of the lights (green on, red/yellow off).
stopLight.filters = [innerShadow];
cautionLight.filters = [innerShadow];
goLight.filters = [greenGlow];

// Swap the filters based on the count value.
function trafficControl(event:TimerEvent):void
{
    if (count == 4)
    {
        count = 1;
    }

    switch (count)
    {
        case 1:
            stopLight.filters = [innerShadow];
            cautionLight.filters = [yellowGlow];
            goLight.filters = [innerShadow];
```

```
            break;
        case 2:
            stopLight.filters = [redGlow];
            cautionLight.filters = [innerShadow];
            goLight.filters = [innerShadow];
            break;
        case 3:
            stopLight.filters = [innerShadow];
            cautionLight.filters = [innerShadow];
            goLight.filters = [greenGlow];
            break;
    }

    count++;
}

// Create a timer to swap the filters at a 3 second interval.
var timer:Timer = new Timer(3000, 9);
timer.addEventListener(TimerEvent.TIMER, trafficControl);
timer.start();
```

顏色矩陣濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ColorMatrixFilter 類別可用來操作已套用濾鏡之物件的顏色和 Alpha 值。它允許您使用某個顏色色版上的值，並讓您將這些色版值套用至其它色版上，來建立飽和度變化、色相旋轉 (在面板上從某個顏色範圍變換至另一個顏色範圍)、明亮度到 Alpha 透明度和各種其它效果。

在概念上，濾鏡會依序檢查來源影像中的所有像素，並將每個像素分為紅、綠、藍與 Alpha 組件。接著，它會將顏色矩陣中的值乘以上述各個值，並將結果加總在一起，判斷將顯示在螢幕上的像素顏色值。濾鏡的 matrix 屬性是一個包含 20 的數字的陣列，可用來計算最後的顏色。如需計算顏色值所使用的特定演算法詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中描述的 ColorMatrixFilter 類別 matrix 屬性項目。

迴旋濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ConvolutionFilter 類別可應用在廣泛的 BitmapData 物件或顯示物件的影像變形作業中，例如模糊化、邊緣偵測、銳利化、浮雕與斜角處理等。

在概念上，迴旋濾鏡會依序檢查來源影像中的每個像素，並依據該像素及其周圍像素的值，來決定該像素的最後顏色。被指定為數值陣列的矩陣，將指出每個特定的相鄰像素值影響最後結果值的程度。

以最常見的 3 x 3 矩陣類型做為考量範例。此矩陣包含 9 個數值：

```
N   N   N
N   P   N
N   N   N
```

將迴旋濾鏡套用至特定像素時，它會檢查像素本身的顏色值 (在此範例中為 「P」)，以及周圍像素的值 (在此範例中標示為 「N」)。然而，當您在矩陣中設定這些值之後，等於指定了特定像素在影響結果影像上的優先順序。

例如，當您使用迴旋濾鏡來套用下列矩陣時，所產生的影像將與之前的影像一模一樣：

```
0   0   0
0   1   0
0   0   0
```

影像沒有出現任何變化的原因在於，原始像素值在決定最後像素顏色上具有 1 的相對強度，而周圍的像素值則具有 0 的相對強度，代表後者的顏色無法影響最後的影像。

同理，這個矩陣也會導致影像的像素向左偏移一個像素：

```
0   0   0
0   0   1
0   0   0
```

請注意，在此情況中，像素本身對該位置上所顯示的最後像素值沒有任何影響，而且只有用到右邊的像素值來決定像素的結果值。

在 ActionScript 中，您可以將矩陣建立為 Array 實體組合，內含一些用來指定矩陣列數與欄數的值與兩個屬性。下列範例將載入影像，並在影像完成載入時，使用先前列出的程式碼中的矩陣將迴旋濾鏡套用至影像中：

```
// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

function applyFilter(event:MouseEvent):void
{
    // Create the convolution matrix.
    var matrix:Array = [0, 0, 0,
                                        0, 0, 1,
                                        0, 0, 0];

    var convolution:ConvolutionFilter = new ConvolutionFilter();
    convolution.matrixX = 3;
    convolution.matrixY = 3;
    convolution.matrix = matrix;
    convolution.divisor = 1;

    loader.filters = [convolution];
}

loader.addEventListener(MouseEvent.CLICK, applyFilter);
```

如果在此程式碼中有些東西看不清楚，那是因為在矩陣中使用了 1 或 0 以外的值所產生的效果。例如，在相同的矩陣中，右邊位置上的 1 如果以 8 取代，還是會執行相同的動作 (但是會將像素向左偏移)。此外，這個數值還會影響影像的顏色，讓顏色增亮 8 倍。這是因為最終的像素顏色值是由矩陣值乘以原始像素顏色，然後將所有數值加總之後，再除以濾鏡的 divisor 屬性值而產生。請注意，在範例程式碼中，divisor 屬性已設為 1。按常理來說，如果您想讓顏色亮度與原始影像的亮度保持一樣，應該將除數設為等於矩陣數值的總和。這樣一來，如果矩陣數值總和為 8，而除數為 1 的話，得出的影像亮度大約會比原始影像高出 8 倍之多。

雖然這個矩陣的效果並不明顯，還是可以使用其它的矩陣數值來建立不同的效果。下列是一些透過 3 x 3 矩陣來產生不同效果的標準矩陣數值組合：

- 基本模糊 (除數為 5)：

```
0 1 0
1 1 1
0 1 0
```

- 銳利化 (除數為 1)：

```
 0, -1, 0
-1, 5, -1
 0, -1, 0
```

- 邊緣偵測 (除數為 1)：

```
 0, -1, 0
-1, 4, -1
 0, -1, 0
```

- 浮雕效果 (除數為 1)：

```
-2, -1, 0
-1, 1, 1
 0, 1, 2
```

請注意，在這些效果當中，大部分的除數都是 1。這是因為將負的矩陣值加入正的矩陣值會變成 1 (如果是邊緣偵測，則為 0，但其 divisor 屬性值不得為 0)。

置換對應濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

DisplacementMapFilter 類別會使用 BitmapData 物件 (稱為置換對應影像) 的像素值，對新的物件執行置換特效。置換對應影像通常與實際顯示物件或是套用了濾鏡的 BitmapData 實體不同。置換效果牽涉到置換含濾鏡的影像像素，換句話說，就是在某種程度上將這些像素從原始位置偏移到其它位置。您可以使用這個濾鏡來建立偏移、彎曲或斑駁的效果。

套用至特定像素的置換位置與數量，需取決於置換對應影像的顏色值。在您使用此濾鏡時，除了指定對應影像外，還需指定下列數值，以控制對應影像中計算置換的方式：

- 對應點：在含濾鏡的影像中，用來對應至所套用之置換濾鏡左上角的位置。如果您只想要將濾鏡套用至部分影像，就可以使用這個方式。
- X 組件：其對應影像的顏色色版會影響像素的 x 位置。
- Y 組件：其對應影像的顏色色版會影響像素的 y 位置。
- X 軸比例：指定 x 軸置換強度的倍數值。
- Y 軸比例：指定 y 軸置換強度的倍數值。
- 濾鏡模式：針對因為像素偏移所產生的空白空間，決定處理方式。這些在 DisplacementMapFilterMode 類別中定義為常數的選項，可用來顯示原始像素 (濾鏡模式為 IGNORE)、從影像的另一側開始圍繞像素 (濾鏡模式為 WRAP，此模式也是預設值)、運用最近的偏移像素 (濾鏡模式為 CLAMP)，或是將空間填滿顏色 (濾鏡模式為 COLOR)。

若要瞭解置換對應濾鏡的運作方式，請以下列基本範例為例。下列程式碼會載入一個影像，並在影像完成載入時將影像放在「舞台」的中央，接著套用一個置換對應濾鏡，讓整個影像中的像素都向左水平平移。

```
import flash.display.BitmapData;
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.geom.Point;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image3.jpg");
loader.load(url);
this.addChild(loader);

var mapImage:BitmapData;
var displacementMap:DisplacementMapFilter;

// This function is called when the image finishes loading.
function setupStage(event:Event):void
{
    // Center the loaded image on the Stage.
    loader.x = (stage.stageWidth - loader.width) / 2;
    loader.y = (stage.stageHeight - loader.height) / 2;

    // Create the displacement map image.
    mapImage = new BitmapData(loader.width, loader.height, false, 0xFF0000);

    // Create the displacement filter.
    displacementMap = new DisplacementMapFilter();
    displacementMap.mapBitmap = mapImage;
    displacementMap.mapPoint = new Point(0, 0);
    displacementMap.componentX = BitmapDataChannel.RED;
    displacementMap.scaleX = 250;
    loader.filters = [displacementMap];
}

loader.contentLoaderInfo.addEventListener(Event.COMPLETE, setupStage);
```

用來定義置換的屬性如下所示：

- 對應點陣圖：置換點陣圖是一個由程式碼所建立的全新 BitmapData 實體，其尺寸符合載入影像的尺寸 (因此會將置換效果套用至整個影像中)，並以純紅色像素填滿影像。
- 對應點：此值目前設為 0, 0 的點座標，並再一次將置換效果套用至整個影像中。
- X 組件：此值目前設為常數 BitmapDataChannel.RED，代表對應點陣圖的紅色值將決定要沿著 x 軸置換的像素量 (也就是像素移動的數量)。
- X 軸比例：此值目前設為 250。置換的總量 (對象是完全置換為紅色的對應影像) 並不會全部用來置換影像，而是約為 1/2 個像素的小量置換，因此如果此值設為 1，則影像只會水平偏移 0.5 個像素。當數值設為 250 時，則影像大約會偏移 125 個像素。

這些設定都會導致含濾鏡的影像像素向左偏移 250 個像素。像素的偏移方向 (向左或向右) 與數量，取決於對應影像中的像素顏色值。在概念上，濾鏡會依序檢查含濾鏡影像的每個像素 (至少會針對套用濾鏡區域中的像素進行檢查，在這裡指的是所有像素)，並對每個像素執行下列動作：

1. 它會在對應影像中尋找相符的像素。例如，當濾鏡準備針對含濾鏡影像的左上角像素計算其置換量時，就會尋找對應影像的左上角像素。
2. 這項作業決定了對應像素中的指定顏色色版值。在這個情況中，x 組件的顏色色版是紅色色版，因此濾鏡會查看有問題的像素，判斷其對應影像的紅色色版值是多少。由於對應影像使用純紅色，因此像素的紅色色版值為 0xFF，或為 255。此值將做為置換值來使用。

3 它會比較置換值與「中間」值 (也就是介於 0 與 255 中間的 127)。如果置換值小於中間值，則像素會往正數方向偏移 (x 置換會向右偏移，y 置換則向下偏移)。另一方面，如果置換值大於中間值 (如此範例所示)，則像素會往負數方向偏移 (x 置換會向左偏移，y 置換則向上偏移)。精確一點的說法是，濾鏡會從 127 減去置換值，結果 (正數或負數) 就是要套用的相對置換量。

4 最後，它會判斷相對置換值所代表的完整置換百分比，來得出實際的置換量。在這個範例中，全紅色代表 100% 置換。接著再以 x 軸比例或 y 軸比例乘以該百分比，得出要套用的置換像素量。在這個範例中，100% 乘以 250 這個倍數就會得到置換量，大約是向左偏移 125 個像素。

由於並未指定任何有關 y 組件與 y 軸比例的值，所以會使用預設值 (也就不會置換)，這就是為何影像不會沿著垂直方向做任何偏移的原因。

這個範例採用預設的濾鏡模式設定 WRAP，因此當像素向左偏移時，右側的空白空間就會填滿由影像左側邊緣偏移過來的像素。您可以使用這個值來測試一下，就可以看到不同的效果。例如，假如您將下列這一行程式碼加入設定了置換屬性的程式碼區段中 (在 loader.filters = [displacementMap] 這一行之前)，會讓 「舞台」上的影像看起來好像一大片塗鴉：

```
displacementMap.mode = DisplacementMapFilterMode.CLAMP;
```

為了進一步說明較為複雜的案例，下列範例採用置換對應濾鏡，在影像上建立放大鏡效果：

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.display.BitmapDataChannel;
import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Shape;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.filters.DisplacementMapFilterMode;
import flash.geom.Matrix;
import flash.geom.Point;
import flash.net.URLRequest;

// Create the gradient circles that will together form the
// displacement map image
var radius:uint = 50;

var type:String = GradientType.LINEAR;
var redColors:Array = [0xFF0000, 0x000000];
var blueColors:Array = [0x0000FF, 0x000000];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var xMatrix:Matrix = new Matrix();
xMatrix.createGradientBox(radius * 2, radius * 2);
var yMatrix:Matrix = new Matrix();
yMatrix.createGradientBox(radius * 2, radius * 2, Math.PI / 2);

var xCircle:Shape = new Shape();
xCircle.graphics.lineStyle(0, 0, 0);
xCircle.graphics.beginGradientFill(type, redColors, alphas, ratios, xMatrix);
xCircle.graphics.drawCircle(radius, radius, radius);

var yCircle:Shape = new Shape();
yCircle.graphics.lineStyle(0, 0, 0);
yCircle.graphics.beginGradientFill(type, blueColors, alphas, ratios, yMatrix);
yCircle.graphics.drawCircle(radius, radius, radius);

// Position the circles at the bottom of the screen, for reference.
this.addChild(xCircle);
xCircle.y = stage.stageHeight - xCircle.height;
this.addChild(yCircle);
yCircle.y = stage.stageHeight - yCircle.height;
```

```
yCircle.x = 200;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create the map image by combining the two gradient circles.
var map:BitmapData = new BitmapData(xCircle.width, xCircle.height, false, 0x7F7F7F);
map.draw(xCircle);
var yMap:BitmapData = new BitmapData(yCircle.width, yCircle.height, false, 0x7F7F7F);
yMap.draw(yCircle);
map.copyChannel(yMap, yMap.rect, new Point(0, 0), BitmapDataChannel.BLUE, BitmapDataChannel.BLUE);
yMap.dispose();

// Display the map image on the Stage, for reference.
var mapBitmap:Bitmap = new Bitmap(map);
this.addChild(mapBitmap);
mapBitmap.x = 400;
mapBitmap.y = stage.stageHeight - mapBitmap.height;

// This function creates the displacement map filter at the mouse location.
function magnify():void
{
    // Position the filter.
    var filterX:Number = (loader.mouseX) - (map.width / 2);
    var filterY:Number = (loader.mouseY) - (map.height / 2);
    var pt:Point = new Point(filterX, filterY);
    var xyFilter:DisplacementMapFilter = new DisplacementMapFilter();
    xyFilter.mapBitmap = map;
    xyFilter.mapPoint = pt;
    // The red in the map image will control x displacement.
    xyFilter.componentX = BitmapDataChannel.RED;
    // The blue in the map image will control y displacement.
    xyFilter.componentY = BitmapDataChannel.BLUE;
    xyFilter.scaleX = 35;
    xyFilter.scaleY = 35;
    xyFilter.mode = DisplacementMapFilterMode.IGNORE;
    loader.filters = [xyFilter];
}

// This function is called when the mouse moves. If the mouse is
// over the loaded image, it applies the filter.
function moveMagnifier(event:MouseEvent):void
{
    if (loader.hitTestPoint(loader.mouseX, loader.mouseY))
    {
        magnify();
    }
}
loader.addEventListener(MouseEvent.MOUSE_MOVE, moveMagnifier);
```

程式碼首先會產生兩個漸層圓形，以便加以結合並形成一個置換對應影像。紅色圓形會建立 x 軸置換 (xyFilter.componentX = BitmapDataChannel.RED)，而藍色圓形則會建立 y 軸置換 (xyFilter.componentY = BitmapDataChannel.BLUE)。為了協助您認識置換對應影像的外觀，程式碼加入了原始圓形以及結合後放在畫面底部做為對應影像的圓形。

程式碼會接著載入影像，並在滑鼠移動時，將置換濾鏡套用至位於滑鼠底下的影像部分。用來做為置換對應影像的漸層圓形，會讓置換的區域從游標往外散開。請注意，置換對應影像的灰色區域不會產生任何置換效果。灰色值為 0x7F7F7F。由於此灰色色調的藍色與紅色色版剛好與這些顏色色版的中間色調一樣，因此在對應影像的灰色區域中不會產生任何置換效果。同理，圓形的中心位置也不會產生任何置換效果。雖然該處的顏色不是灰色，然而該顏色的藍色色版與紅色色版卻與淺灰色的藍色色版及紅色色版一模一樣，而且由於藍色與紅色都是會產生置換效果的顏色，因此該處不會有任何置換情況。

著色器濾鏡

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

ShaderFilter 類別可讓您自訂定義為 Pixel Bender 著色器的濾鏡特效。由於濾鏡效果會撰寫為 Pixel Bender 著色器，因此效果完全可供自訂。套用濾鏡後的內容會被傳入到著色器當做影像輸入，而且著色器作業的結果會變成濾鏡結果。

備註：從 Flash Player 10 和 Adobe AIR 1.5 開始，已可以在 ActionScript 中使用著色器濾鏡。

若要將著色器濾鏡套用到物件，您要先建立一個 Shader 實體，代表您使用的 Pixel Bender 著色器。如需有關建立 Shader 實體之程序以及如何指定輸入影像和參數值的詳細資訊，請參閱第 253 頁「使用 Pixel Bender 著色器」。

將著色器當做濾鏡使用時，請記住三個重要事項：

- 著色器必須定義為至少接受一個輸入影像。
- 使用濾鏡的物件 (套用濾鏡的顯示物件或 BitmapData 物件) 會被傳遞到著色器，做為第一個輸入影像值。因此，請勿手動指定第一個影像輸入的值。
- 如果著色器定義一個以上的輸入影像，則必須手動指定其它輸入 (亦即，藉由設定屬於 Shader 實體之任何 ShaderInput 實體的 input 屬性)。

一旦您的著色器有一個 Shader 物件之後，您就要建立 ShaderFilter 實體。這是您實際使用的濾鏡物件，使用起來就像其它任何濾鏡一樣。若要建立使用 Shader 物件的 ShaderFilter，請呼叫 ShaderFilter() 建構函式，然後將 Shader 物件當做引數傳遞，如列出的程式碼所示：

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
```

如需著色器濾鏡的完整使用範例，請參閱第 268 頁 「使用著色器做為濾鏡」。

濾鏡處理顯示物件範例：濾鏡工作台

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

「濾鏡工作台」所提供的使用者介面，可將不同的濾鏡套用至影像與其它視覺內容，並檢視產生的程式碼，以便在 ActionScript 中產生相同的效果。除了提供試驗濾鏡的工具之外，此應用程式也示範下列技術：

- 建立各種濾鏡的實體
- 將多個濾鏡套用至顯示物件

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。您可以在 Samples/FilterWorkbench 資料夾中找到 「濾鏡工作台」應用程式檔案。此應用程式是由下列檔案組成：

檔案	說明
com/example/programmingas3/filterWorkbench/FilterWorkbenchController.as	提供應用程式主功能的類別，包括切換到套用濾鏡的內容，以及套用濾鏡到內容。
com/example/programmingas3/filterWorkbench/IFilterFactory.as	此介面定義由每個 filter factory 類別所實作的常用方法，並定義 FilterWorkbenchController 類別用來與個別的 filter factory 類別互動的常見功能。
在 com/example/programmingas3/filterWorkbench/ 資料夾中： BevelFactory.as BlurFactory.as ColorMatrixFactory.as ConvolutionFactory.as DropShadowFactory.as GlowFactory.as GradientBevelFactory.as GradientGlowFactory.as	類別，用於實作 IFilterFactory 介面。這些類別都提供建立及設定濾鏡的單一類型值之功能。應用程式內的濾鏡屬性面板會使用這些 factory 類別來建立其特定的濾鏡實體，這些濾鏡實體會由 FilterWorkbenchController 類別進行擷取並套用至影像內容。
com/example/programmingas3/filterWorkbench/IFilterPanel.as	定義常用方法的介面，這些方法是由定義使用者介面面板 (用來操作應用程式中的濾鏡值) 的類別所實作。
com/example/programmingas3/filterWorkbench/ColorStringFormatter.as	Utility 類別，內含將數字顏色值轉換為十六進位字串格式的方法
com/example/programmingas3/filterWorkbench/GradientColor.as	當做 value 物件的類別，可將與 GradientBevelFilter 及 GradientGlowFilter 內的每個顏色關聯的三個值 (color、alpha 和 ratio) 結合成單一物件
使用者介面 (Flex)	
FilterWorkbench.mxml	定義應用程式使用者介面的主要檔案。
flexapp/FilterWorkbench.as	用來提供主應用程式的使用者介面功能之類別；此類別會在應用程式的 MXML 檔案當做程式碼後置類別使用。

檔案	說明
在 flexapp/filterPanels 資料夾中： BevelPanel.mxml BlurPanel.mxml ColorMatrixPanel.mxml ConvolutionPanel.mxml DropShadowPanel.mxml GlowPanel.mxml GradientBevelPanel.mxml GradientGlowPanel.mxml	一組 MXML 組件，提供每個面板的功能，這些面板可用於設定單一濾鏡的選項。
flexapp/ImageContainer.as	在螢幕上做為已載入影像之容器的顯示物件
flexapp/controls/BGColorCellRenderer.as	用於變更 DataGrid 組件中儲存格背景顏色的自訂儲存格輸出器
flexapp/controls/QualityComboBox.as	定義可在數個濾鏡面板中，用於 「品質」設定之下拉式清單方塊的自訂控制項。
flexapp/controls/TypeComboBox.as	定義可在數個濾鏡面板中，用於 「類型」設定之下拉式清單方塊的自訂控制項。
使用者介面 (Flash)	
FilterWorkbench.fla	定義應用程式使用者介面的主要檔案。
flashapp/FilterWorkbench.as	用來提供主應用程式的使用者介面功能之類別；此類別會在應用程式的 FLA 檔當做文件類別使用。
在 flashapp/filterPanels 資料夾中： BevelPanel.as BlurPanel.as ColorMatrixPanel.as ConvolutionPanel.as DropShadowPanel.as GlowPanel.as GradientBevelPanel.as GradientGlowPanel.as	類別，提供每個面板的功能，這些面板可用於設定單一濾鏡的選項。 每個類別中也內含一個關聯的 MovieClip 元件，該元件位於主應用程式 FLA 檔的元件庫中，其名稱與類別的名稱相符 (例如，元件 "BlurPanel" 會連結至在 BlurPanel.as 中定義的類別)。構成使用者介面的組件將會定位並在元件內命名。
flashapp/ImageContainer.as	在螢幕上做為已載入影像之容器的顯示物件
flashapp/BGColorCellRenderer.as	用於變更 DataGrid 組件中儲存格背景顏色的自訂儲存格輸出器
flashapp/ButtonCellRenderer.as	自訂儲存格輸出器，用來在 DataGrid 組件內的某個儲存格中嵌入 button 組件。

檔案	說明
已套用濾鏡的影像內容	
com/example/programmingas3/filterWorkbench/ImageType.as	此類別可做為 value 物件使用，內含單一影像檔的類型和 URL，應用程式可以在該影像檔上載入並套用濾鏡。此類別也包含一組常數，用於表示實際可用的影像檔。
images/sampleAnimation.swf、 images/sampleImage1.jpg、 images/sampleImage2.jpg	在應用程式中已套用濾鏡的影像和其它的視覺內容。

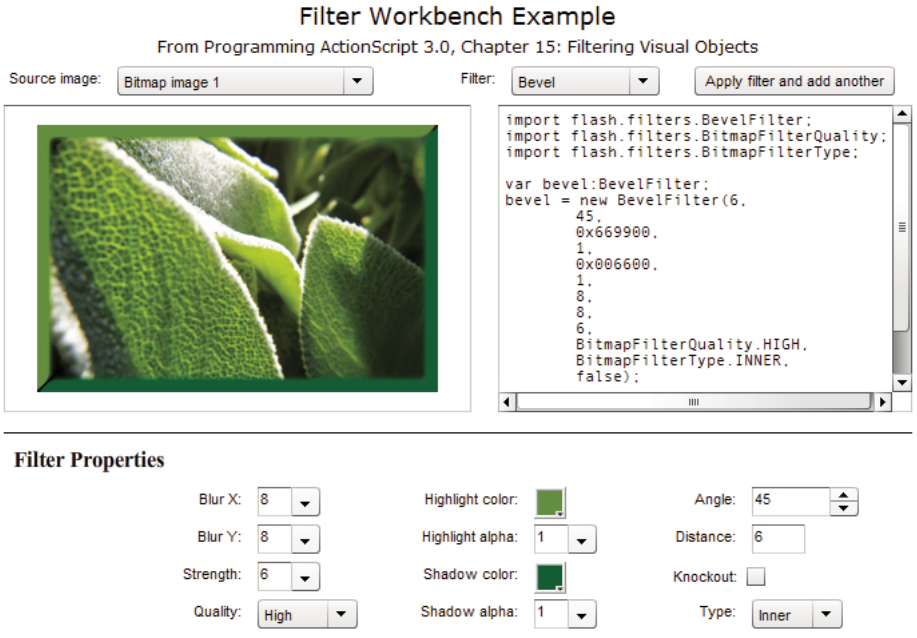
以 ActionScript 濾鏡進行試驗

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

「濾鏡工作台」應用程式可用來協助您試驗各種濾鏡效果，並為該效果產生相關 ActionScript 程式碼。此應用程式可讓您從三種內含視覺內容的不同檔案中進行選取，包括點陣圖影像和以 Flash 建立的動畫，以及對選取的影像套用八種不同的 ActionScript 濾鏡，您可以個別套用或與其它濾鏡組合套用。此應用程式包含下列濾鏡：

- 斜角 (flash.filters.BevelFilter)
- 模糊 (flash.filters.BlurFilter)
- 顏色矩陣 (flash.filters.ColorMatrixFilter)
- 迴旋 (flash.filters.ConvolutionFilter)
- 投影 (flash.filters.DropShadowFilter)
- 光暈 (flash.filters.GlowFilter)
- 漸層斜角 (flash.filters.GradientBevelFilter)
- 漸層光暈 (flash.filters.GradientGlowFilter)

只要使用者選取了某個影像和要套用至該影像的濾鏡，應用程式就會顯示一個面板，內含設定所選濾鏡的特定屬性之控制項。例如，下列影像顯示了已選取 「斜角」濾鏡的應用程式：



隨著使用者調整濾鏡的屬性，預覽內容便會即時更新。使用者也可以藉由自訂某個濾鏡、按一下 「套用」按鈕、自訂另一個濾鏡、按一下 「套用」按鈕等步驟，套用多個濾鏡。

應用程式的濾鏡面板有幾項功能和限制：

- 顏色矩陣濾鏡內含一組控制項，可直接操作一般影像的屬性，包括亮度、對比、飽和度和色相。此外，也可以指定自訂的顏色矩陣值。
- 使用 ActionScript 時才提供的迴旋濾鏡包含了一組常用的迴旋矩陣值，或者可以指定自訂值。不過，雖然 ConvolutionFilter 類別可接受任何大小的矩陣，「濾鏡工作台」應用程式使用的是固定的 3 x 3 矩陣，這也是最常用的濾鏡大小。
- 置換對應濾鏡和著色器濾鏡只可以在 ActionScript 內使用，無法在 「濾鏡工作台」應用程式內使用。

建立濾鏡實體

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

「濾鏡工作台」應用程式內含一組類別，每個可用的濾鏡都有一個類別，可由個別的面板用來建立濾鏡。當使用者選取濾鏡時，與濾鏡面板關聯的 ActionScript 程式碼會建立適當 filter factory 類別的實體 (這些類別稱為「factory 類別」，由於其目的是要建立其它物件的實體，因此就像是在現實世界中製造個別產品的工廠)。

每當使用者變更面板上的屬性值時，面板的程式碼便會呼叫 factory 類別內適當的方法。每個 factory 類別都內含特定的方法，可讓面板用來建立適當的濾鏡實體。例如，如果使用者選取了 「模糊」濾鏡，應用程式便會建立一個 BlurFactory 實體。BlurFactory 類別包含一個可接受三個參數的 modifyFilter() 方法：blurX、blurY 和 quality，這三個參數都會用來建立所需的 BlurFilter 實體：

```
private var _filter:BlurFilter;

public function modifyFilter(blurX:Number = 4, blurY:Number = 4, quality:int = 1):void
{
    _filter = new BlurFilter(blurX, blurY, quality);
    dispatchEvent(new Event(Event.CHANGE));
}
```

另一方面，如果使用者選取了「迴旋」濾鏡，由於該濾鏡提供了更多彈性，便可控制較多屬性組。在 ConvolutionFactory 類別中，下列程式碼會在使用者於濾鏡面板上選取不同值時呼叫：

```
private var _filter:ConvolutionFilter;

public function modifyFilter(matrixX:Number = 0,
                                                matrixY:Number = 0,
                                                matrix:Array = null,
                                                divisor:Number = 1.0,
                                                bias:Number = 0.0,
                                                preserveAlpha:Boolean = true,
                                                clamp:Boolean = true,
                                                color:uint = 0,
                                                alpha:Number = 0.0):void
{
    _filter = new ConvolutionFilter(matrixX, matrixY, matrix, divisor, bias, preserveAlpha, clamp, color,
alpha);
    dispatchEvent(new Event(Event.CHANGE));
}
```

請注意，在各種情況下，變更濾鏡值時，factory 物件便會傳送一個 Event.CHANGE 事件，以通知偵聽程式該濾鏡的值已變更。FilterWorkbenchController 類別會實際執行將濾鏡套用至已套用濾鏡之內容的工作，並偵聽該事件，以查明何時需要擷取新的濾鏡副本，並重新將濾鏡套用至已套用濾鏡的內容上。

FilterWorkbenchController 類別不需要知道每個 filter factory 類別的特殊細節；它只需要知道濾鏡已變更，並且能夠存取濾鏡的副本即可。為了支援此動作，應用程式包含了一個介面 (即 IFilterFactory)，該介面會定義 filter factory 類別需要提供的行為，因此應用程式的 FilterWorkbenchController 實體便可以執行其工作。IFilterFactory 定義了 getFilter() 方法，可在 FilterWorkbenchController 類別內使用：

```
function getFilter():BitmapFilter;
```

請注意，getFilter() 介面方法定義會指定其傳回 BitmapFilter 實體，而非特定的濾鏡類型。BitmapFilter 類別不會定義特定的濾鏡類型。因此，BitmapFilter 是建立所有濾鏡類別的基底類別。每個 filter factory 類別都會定義一個特定的 getFilter() 方法實作，將所建立的濾鏡物件參考傳回。例如，以下是 ConvolutionFactory 類別的縮寫版原始碼：

```
public class ConvolutionFactory extends EventDispatcher implements IFilterFactory
{
    // ------- Private vars -------
    private var _filter:ConvolutionFilter;
    ...
    // ------- IFilterFactory implementation -------
    public function getFilter():BitmapFilter
    {
        return _filter;
    }
    ...
}
```

ConvolutionFactory 類別實作 getFilter() 方法時，會傳回一個 ConvolutionFilter 實體，不過任何呼叫 getFilter() 的物件都不需要知道這件事 (依照 ConvolutionFactory 所遵守的 getFilter() 方法之定義來看，它必須傳回任何 BitmapFilter 實體，該實體可以是 ActionScript 的任何濾鏡類別)。

將濾鏡套用至顯示物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如先前所述，「濾鏡工作台」應用程式會使用 FilterWorkbenchController 類別的實體 (此後稱為 「控制器實體」)，該實體會實際執行將濾鏡套用至所選視覺物件的工作。在控制器實體可以套用濾鏡之前，必須先得知應該將濾鏡套用到哪個影像或視覺內容。使用者選取某個影像時，應用程式會呼叫 FilterWorkbenchController 類別內的 setFilterTarget() 方法，並傳遞在 ImageType 類別內定義的其中一個常數至該方法：

```
public function setFilterTarget(targetType:ImageType):void
{
    ...
    _loader = new Loader();
    ...
    _loader.contentLoaderInfo.addEventListener(Event.COMPLETE, targetLoadComplete);
    ...
}
```

控制器實體會使用該資訊來載入指定的檔案，並在載入後將其儲存在名為 _currentTarget 的實體變數中：

```
private var _currentTarget:DisplayObject;

private function targetLoadComplete(event:Event):void
{
    ...
    _currentTarget = _loader.content;
    ...
}
```

當使用者選取了某個濾鏡時，應用程式會呼叫控制器實體的 setFilter() 方法，為控制器提供相關 filter factory 物件的參考，並將之儲存在一個名為 _filterFactory 的實體變數中。

```
private var _filterFactory:IFilterFactory;

public function setFilter(factory:IFilterFactory):void
{
    ...

    _filterFactory = factory;
    _filterFactory.addEventListener(Event.CHANGE, filterChange);
}
```

請注意，如先前的內容所提及，控制器實體不知道所提供之 filter factory 實體的特定資料類型：它只知道該物件實作了 IFilterFactory 實體，表示該物件擁有一個 getFilter() 方法，並且會在濾鏡變更時傳送 change (Event.CHANGE) 事件。

當使用者在濾鏡的面板中變更濾鏡的屬性時，控制器實體會發現濾鏡已經透過 filter factory 的 change 事件變更，而且這個事件會呼叫控制器實體的 filterChange() 方法。接著，該方法就會呼叫 applyTemporaryFilter() 方法：

```
private function filterChange(event:Event):void
{
    applyTemporaryFilter();
}

private function applyTemporaryFilter():void
{
    var currentFilter:BitmapFilter = _filterFactory.getFilter();

    // Add the current filter to the set temporarily
    _currentFilters.push(currentFilter);

    // Refresh the filter set of the filter target
    _currentTarget.filters = _currentFilters;

    // Remove the current filter from the set
    // (This doesn't remove it from the filter target, since
    // the target uses a copy of the filters array internally.)
    _currentFilters.pop();
}
```

將濾鏡套用至顯示物件的工作會在 applyTemporaryFilter() 方法內發生。首先，控制器會藉由呼叫 filter factory 的 getFilter() 方法來擷取對濾鏡物件的參考。

```
var currentFilter:BitmapFilter = _filterFactory.getFilter();
```

控制器實體擁有一個名為 _currentFilters 的 Array 實體變數，當中會儲存所有已套用至顯示物件的濾鏡。下一個步驟是要將最近更新的濾鏡加入該陣列：

```
_currentFilters.push(currentFilter);
```

接下來，程式碼會將濾鏡的陣列指定給顯示物件的 filters 屬性，以便將濾鏡實際套用到影像：

```
_currentTarget.filters = _currentFilters;
```

最後，由於這個最近加入的濾鏡仍是 「作用中」的濾鏡，不該將其永久套用至顯示物件，因此應該要將該濾鏡從 _currentFilters 陣列中移除：

```
_currentFilters.pop();
```

將這個濾鏡從陣列中移除並不會影響已套用濾鏡的顯示物件，因為顯示物件已在將濾鏡陣列指定給 filters 屬性時對其製作副本，並且會使用該內部陣列，而不是原始的陣列。基於這個原因，除非您再次將陣列指定給顯示物件的 filters 屬性，否則對濾鏡陣列進行的任何變更都不會影響顯示物件。

第 15 章 使用 Pixel Bender 著色器

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

Adobe Pixel Bender Toolkit 可以讓開發人員撰寫著色器，以創造圖形效果並執行其他影像和資料處理。Pixel Bender 位元組碼可以在 ActionScript 中執行，將效果套用於影像資料或視覺內容。在 ActionScript 中使用 Pixel Bender 著色器可讓您創造自訂的視覺效果，並執行超出 ActionScript 中內建功能之外的資料處理。

備註：自 Flash Player 10 與 Adobe AIR 1.5 開始便可支援 Pixel Bender。在 GPU 顯示下不支援 Pixel Bender 混合、濾鏡和填色。

更多說明主題

Adobe Pixel Bender 技術中心

Pixel Bender 開發人員指南

Pixel Bender 參考

flash.display.Shader

flash.filters.ShaderFilter

Flash 的 Pixel Bender 基本概念

Flex 的 Pixel Bender 基本概念

Pixel Bender 著色的基本概念

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

Adobe Pixel Bender 是用來建立或操作影像內容的程式設計語言。您可以使用 Pixel Bender 建立核心 (亦稱為著色器)。著色器可以定義針對影像的每個像素獨立執行的函數。每一次呼叫函數的結果就是影像中該像素座標上的輸出色彩。您可以指定輸入影像和參數值來自訂作業。在單一著色器執行作業期間，輸入與參數值都是常數。唯一不同的地方就是像素座標 (其顏色為函數呼叫的結果)。

您可以同時針對多個輸出像素座標，呼叫同一個著色器函數。如此可以改善著色器的效能，大幅提高處理效率。

在 ActionScript 中，可使用著色器輕鬆建立三種特效：

- 繪圖填色
- 混合模式
- 濾鏡

在獨立模式下，您也可以建立著色器。使用獨立模式時，會直接存取著色器的結果，而非預先指定其使用目的。存取結果時，可以當做影像資料、二進位或數字資料來存取。資料不一定要是影像資料。利用這種方式時，您可以為著色器提供一組資料做為輸入。著色器會處理該組資料，然後您即可存取著色器傳回的結果資料。

從 Flash Player 10 和 Adobe AIR 1.5 開始，可以使用 Pixel Bender 支援。在 GPU 顯示下不支援 Pixel Bender 混合、濾鏡和填色。在行動裝置上，Pixel Bender 著色器確實會在 CPU 顯示下執行。不過，效能並非與桌上型電腦同等級。許多著色器程式每秒可能只能執行少許影格。

重要概念與術語

下列參考清單包含建立與使用 Pixel Bender 著色器時會遇到的重要術語：

核心 對 Pixel Bender 而言，核心就和著色器一樣。使用 Pixel Bender 時，程式碼會定義核心，而這個核心會定義針對影像的每個像素獨立執行的函數。

Pixel Bender 位元組碼 編譯 Pixel Bender 核心時，核心會轉換成 Pixel Bender 位元組碼。位元組碼會在執行階段存取和執行。

Pixel Bender 語言 用來建立 Pixel Bender 核心的程式設計語言。

Pixel Bender 工具箱 用來從 Pixel Bender 原始碼建立 Pixel Bender 位元組碼檔案的應用程式。這套工具組可讓您撰寫、測試和編譯 Pixel Bender 原始碼。

Shader 就本文件來說，著色器是一組以 Pixel Bender 語言撰寫的功能。著色器的程式碼會建立視覺特效或執行計算。無論是哪一種情況，著色器都會傳回一組資料 (通常是影像的像素)。著色器會對每個資料點執行相同的作業，唯一不同之處，就是輸出像素的座標。著色器並非使用 ActionScript 所撰寫。而是使用 Pixel Bender 語言來撰寫，並編譯為 Pixel Bender 位元組碼。您可以在編譯階段將著色器內嵌於 SWF 檔中，或者在執行階段將著色器當做外部檔案來載入。無論是哪一種情況，您都可以藉由建立 Shader 物件並將其連結至著色器位元組碼，透過 ActionScript 來存取著色器。

著色器輸入 提供給著色器執行計算的複雜輸入 (通常是點陣圖影像資料)。在著色器執行計算期間，會使用單一值 (也就是單一影像或一組二進位資料) 做為著色器中每個已定義之輸入變數的值。

著色器參數 提供給著色器執行計算的單一值 (或有限的一組值)。每個參數值都是針對一次著色器計算作業所定義，並且在執行期間，都會使用相同的值。

逐步執行程式碼範例

您可測試列出的範例程式碼。測試程式碼的步驟包括執行程式碼，以及檢視所建立 SWF 的結果。所有範例都會使用繪圖 API (著色器特效會使用或修改此繪圖 API) 來建立內容。

大部分列出的範例程式碼都包含兩個部分：其中一個部分是範例中使用之著色器的 Pixel Bender 原始碼。您必須先使用 Pixel Bender Toolkit 將原始碼編譯為 Pixel Bender 位元組碼檔案。請遵循下列步驟，建立 Pixel Bender 位元組碼檔案：

1. 開啟 Adobe Pixel Bender Toolkit。如有必要，請從 「建置」選單選擇 「開啟 Flash Player 警告和錯誤」。
2. 複製列出的 Pixel Bender 程式碼，然後貼至 Pixel Bender Toolkit 的程式碼編輯器窗格。
3. 從 「檔案」選單選擇 「匯出核心濾鏡供 Flash Player 使用」。
4. 將 Pixel Bender 的位元組碼檔案儲存在與 Flash 文件相同的目錄中。檔案的名稱必須與範例描述中指定的名稱相符。

每個範例的 ActionScript 部分都是撰寫為類別檔案。在 Flash Professional 中測試範例：

1. 建立空白的 Flash 文件，並儲存在電腦上。
2. 建立新的 ActionScript 檔案，並將它儲存在此 Flash 文件所在的相同目錄中。檔案的名稱必須和列出的程式碼中的類別名稱相符。例如，如果列出的程式碼定義名為 MyApplication 的類別，就會使用 MyApplication.as 這個名稱來儲存 ActionScript 檔案。
3. 將列出的程式碼複製到 ActionScript 檔案中，並儲存該檔案。
4. 在 Flash 文件中按一下 「舞台」或工作區的空白部分，以啟動文件的 「屬性」檢測器。
5. 在 「屬性」檢測器的 「文件類別」欄位中，輸入從文字中所複製 ActionScript 類別的名稱。
6. 使用 「控制 > 測試影片」執行程式

   您將會在預覽視窗中看到範例的結果。

第 945 頁 「如何使用 ActionScript 範例」詳細說明這些用於測試範例程式碼清單的技術。

載入或嵌入著色器

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

在 ActionScript 中使用 Pixel Bender 著色器的第一個步驟，就是在您的 ActionScript 程式碼中存取著色器。由於著色器是使用 Adobe Pixel Bender Toolkit 建立，並以 Pixel Bender 語言撰寫而成，因此無法在 ActionScript 中直接存取，而必須建立代表 Pixel Bender 著色器的 Shader 類別實體至 ActionScript。Shader 物件可以讓您查明有關著色器的資訊，例如，著色器應該會預期得到參數或輸入影像值。您需要將 Shader 物件傳遞給其他物件，才能真正使用著色器。例如，若要使用著色器做為濾鏡，必須指定 Shader 物件給 ShaderFilter 物件的 shader 屬性。或者，若要使用著色器做為繪圖填色，必須以 Shader 物件做為引數，傳遞給 Graphics.beginShaderFill() 方法。

您的 ActionScript 程式碼有兩種方式可以存取由 Adobe Pixel Bender Toolkit 建立的著色器 (.pbj 檔)：

- 在執行階段載入：著色器檔案可以使用 URLLoader 物件，載入做為外部資源。這項技巧就像載入文字檔之類的外部資源。下列範例示範在執行階段載入著色器位元組碼檔案，並將其與 Shader 實體連結：

```
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.BINARY;
loader.addEventListener(Event.COMPLETE, onLoadComplete);
loader.load(new URLRequest("myShader.pbj"));

var shader:Shader;

function onLoadComplete(event:Event):void {
    // Create a new shader and set the loaded data as its bytecode
    shader = new Shader();
    shader.byteCode = loader.data;

    // You can also pass the bytecode to the Shader() constructor like this:
    // shader = new Shader(loader.data);

     // do something with the shader
}
```

- 嵌入 SWF 檔中：在編譯階段，可以使用 [Embed] 中繼資料標籤將著色器檔案嵌入 SWF 檔中。只有使用 Flex SDK 來編譯 SWF 檔，才能使用 [Embed] 中繼資料標籤。[Embed] 標籤的 source 參數會指向著色器檔案，而其 mimeType 參數則是 "application/octet-stream"，如下列範例所示：

```
[Embed(source="myShader.pbj", mimeType="application/octet-stream")]
var MyShaderClass:Class;

// ...

// create a shader and set the embedded shader as its bytecode
var shader:Shader = new Shader();
shader.byteCode = new MyShaderClass();

// You can also pass the bytecode to the Shader() constructor like this:
// var shader:Shader = new Shader(new MyShaderClass());

// do something with the shader
```

不管用哪種方式，您都可以將原始著色器位元組碼 (URLLoader.data 屬性或是 [Embed] 資料類別的實體) 連結至 Shader 實體。如上述範例所示範，您有兩種方式可以指定位元組碼給 Shader 實體。您可以將著色器位元組碼當成引數傳遞給 Shader() 建構函數。或者，您可以將它設為 Shader 實體的 byteCode 屬性。

建立 Pixel Bender 著色器並連結至 Shader 物件之後，您就可以使用著色器，以多種方式來創造效果。您可以用它來做為濾鏡、混合模式、點陣圖填色，或者獨立處理點陣圖或其他資料。您也可以使用 Shader 物件的 data 屬性，存取著色器的中繼資料、指定輸入影像，以及設定參數值。

存取著色器中繼資料

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

建立 Pixel Bender 著色器核心時，作者也可以在 Pixel Bender 原始碼中指定有關著色器的中繼資料。在 ActionScript 中使用著色器時，您也可以檢查著色器並擷取其中繼資料。

當您建立 Shader 實體並連結至 Pixel Bender 著色器時，就會建立包含有關著色器資料的 ShaderData 物件，並儲存在 Shader 物件的 data 屬性中。ShaderData 類別不會定義本身的任何屬性。但是在執行階段，則會為著色器原始碼中所定義的每個中繼資料值，以動態方式將屬性加入至 ShaderData 物件。每個屬性的名稱都與中繼資料內所指定的名稱相同。例如，假設 Pixel Bender 著色器的原始碼包含下列中繼資料定義：

```
namespace : "Adobe::Example";
vendor : "Bob Jones";
version : 1;
description : "Creates a version of the specified image with the specified brightness.";
```

那麼，便會以下列屬性和值為該著色器建立 ShaderData 物件：

- namespace (String): "Adobe::Example"
- vendor (String): "Bob Jones"
- version (String): "1"
- description (String): "Creates a version of the specified image with the specified brightness"

由於中繼資料屬性是以動態方式加入至 ShaderData 物件，您可以使用 for..in 迴圈來檢查 ShaderData 物件。您可以使用這項技巧，查明著色器是否有任何中繼資料以及中繼資料的值。除了中繼資料屬性以外，ShaderData 物件也可以具有代表在著色器中所定義輸入和參數的屬性。當您使用 for..in 迴圈來檢查 ShaderData 物件時，請檢查每個屬性的資料類型，以判斷屬性是輸入 (ShaderInput 實體)、參數 (ShaderParameter 實體) 或中繼資料值 (String 實體)。下列範例說明如何使用 for..in 迴圈來檢查著色器 data 屬性的動態屬性。每一個中繼資料值都會加入至 Vector 實體 metadata。請注意，這個範例是假設 Shader 實體 myShader 已經建立：

```
var shaderData:ShaderData = myShader.data;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (!(shaderData[prop] is ShaderInput) && !(shaderData[prop] is ShaderParameter))
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the metadata
```

此範例還有另一種版本，可同時擷取著色器輸入和參數，請參閱第 257 頁 「識別著色器輸入和參數」。如需有關輸入和參數屬性的詳細資訊，請參閱第 256 頁 「指定著色器輸入和參數值」。

指定著色器輸入和參數值

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

許多 Pixel Bender 著色器已定義要運用在著色器處理中所使用的一個或多個輸入影像。例如，著色器接受來源影像，然後套用特殊效果之後再輸出，是很常見的做法。依著色器的使用方式而定，輸入值可能會自動指定，也可能必須明確地提供。同樣地，許多著色器會指定參數，以便用來自訂著色器的輸出。您也必須明確地為每一個參數設定值，才能使用著色器。

請使用 Shader 物件的 data 屬性來設定著色器輸入和參數，並判斷一個特定的著色器應該預期收到輸入或參數。data 屬性是一個 ShaderData 實體。

識別著色器輸入和參數

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

指定著色器輸入和參數值的第一個步驟，就是查明您要使用的特定著色器應該會預期收到任何輸入影像或參數。每個 Shader 實體都具有包含 ShaderData 物件的 data 屬性。如果著色器定義任何輸入或參數，就會做為該 ShaderData 物件的屬性進行存取。屬性的名稱會與著色器原始碼中的輸入和參數指定的名稱相符。例如，如果著色器定義輸入 src，ShaderData 物件就會有代表該輸入的 src 屬性。每個代表輸入的屬性都是 ShaderInput 實體，而每個代表參數的屬性都是 ShaderParameter 實體。

在理想狀況下，著色器的作者會提供著色器的文件記錄，指出著色器應該會接到的輸入值和參數、所代表的項目、適當的值等資訊。

但是，如果著色器並沒記錄文件 (而您又沒有其原始碼)，您可以檢視著色器資料以識別輸入和參數。代表輸入和參數的屬性是以動態方式加入至 ShaderData 物件。因此，您可以使用 for..in 迴圈來檢視 ShaderData 物件，以查明其關聯的著色器是否定義任何輸入或參數。如上文第 256 頁「存取著色器中繼資料」一節中所述，為著色器所定義的任何中繼資料值也都可以做為加入至 Shader.data 屬性的動態屬性來存取。當您使用這項技巧來識別著色器輸入和參數時，請檢查動態屬性的資料類型。如果屬性是 ShaderInput 實體，就代表是輸入；如果是 ShaderParameter 實體，就代表是參數；否則，就是中繼資料值。下列範例說明如何使用 for..in 迴圈來檢查著色器 data 屬性的動態屬性。每項輸入 (ShaderInput 物件) 都會加入至 Vector 實體 inputs。每個參數 (ShaderParameter 物件) 都會加入至 Vector 實體 parameters。最後，任何中繼資料都會加入至 Vector 實體 metadata。請注意，這個範例是假設 Shader 實體 myShader 已經建立：

```
var shaderData:ShaderData = myShader.data;
var inputs:Vector.<ShaderInput> = new Vector.<ShaderInput>();
var parameters:Vector.<ShaderParameter> = new Vector.<ShaderParameter>();
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (shaderData[prop] is ShaderInput)
    {
        inputs[inputs.length] = shaderData[prop];
    }
    else if (shaderData[prop] is ShaderParameter)
    {
        parameters[parameters.length] = shaderData[prop];
    }
    else
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the inputs or properties
```

指定著色器輸入值

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

許多著色器應該都會接到一個或多個輸入影像，以用於著色器處理中。但是在許多情況下，輸入會在使用 Shader 物件時自動指定。例如，假設著色器需要一項輸入，而且該著色器會用來做為濾鏡。當套用篩選至顯示物件或 BitmapData 物件時，該物件會自動設定為輸入。在此情況下，您不要明確設定輸入值。

但是在某些情況下，尤其是在著色器定義多項輸入時，一定要明確設定輸入的值。著色器中定義的每項輸入都是在 ActionScript 中以 ShaderInput 物件來代表。ShaderInput 物件是 Shader 物件之 data 屬性中 ShaderData 實體的屬性，如第 257 頁「識別著色器輸入和參數」一節中所述。例如，假設著色器定義了輸入 src，而該著色器連結至 Shader 物件 myShader。在此情況下，您要使用下列識別名稱，存取對應於 src 輸入的 ShaderInput 物件：

```
myShader.data.src
```

每個 ShaderInput 物件都有 input 屬性，可用來設定該輸入的值。您已將 input 屬性設定為 BitmapData 實體以指定影像資料。您也可以將 input 屬性設定為 BitmapData 或 Vector.<Number> 實體，以指定二進位或數字資料。如需有關使用 BitmapData 或 Vector.<Number> 實體當作輸入的詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中所列的 ShaderInput.input。

除了 input 屬性以外，ShaderInput 物件也有屬性可用來判斷輸入應該預期收到的影像類型。這些屬性包括 width、height 和 channels 屬性。每個 ShaderInput 物件也都有 index 屬性，可用來判斷是否必須為輸入提供明確的值。如果著色器應該會接到比自動設定數目更多的輸入，就要設定這些輸入的值。如需著色器不同使用方式以及是否自動設定輸入值的詳細資訊，請參閱第 261 頁「使用著色器」。

指定著色器參數值

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

有些著色器會定義參數值，用於建立其結果。例如，改變影像亮度的著色器可能會指定亮度參數，用來判斷作業影響亮度的程度。在著色器中定義的單一參數依著色器中的參數定義而定，會預期接到單一值或多個值。著色器中定義的每個參數都是在 ActionScript 中以 ShaderParameter 物件來代表。ShaderParameter 物件是 Shader 物件之 data 屬性中 ShaderData 實體的屬性，如第 257 頁「識別著色器輸入和參數」一節中所述。例如，假設著色器定義了參數 brightness，而該著色器連結至 Shader 物件 myShader。在此情況下，您要使用下列識別名稱，存取對應於 brightness 參數的 ShaderParameter 物件：

```
myShader.data.brightness
```

若要設定參數的一個或多個值，請建立包含一個或多個值的 ActionScript 陣列，並指定該陣列至 ShaderParameter 物件的 value 屬性。value 屬性是定義為 Array 實體，因為單一著色器參數有可能需要多個值。即使著色器參數只預期單一值，您也必須將該值包覆在 Array 物件中，指定至 ShaderParameter.value 屬性。下面列出的程式碼示範，將單一值設定為 value 屬性：

```
myShader.data.brightness.value = [75];
```

如果著色器的 Pixel Bender 原始碼為參數定義了預設值，就會建立包含一個或多個預設值的陣列，並在建立 Shader 物件時指定至 ShaderParameter 物件的 value 屬性。將陣列指定至 value 屬性 (包括指定它是否為預設值) 之後，可以透過變更陣列元素的值來變更參數值。您不必建立新陣列，並指定至 value 屬性。

下列範例會示範如何在 ActionScript 中設定著色器的參數值。在此範例中，著色器會定義參數 color。color 參數是在 Pixel Bender 原始碼中宣告為 float4 變數，也就是說，它是具有四個浮點數的陣列。在範例中，color 參數值是持續不斷地改變，而且每次變更，就使用著色器在螢幕上繪製彩色矩形。所得到的結果就是動畫式的顏色變更。

備註：這個範例的程式碼是由 Ryan Taylor 所撰寫的。感謝 Ryan 與我們分享這個範例。您可以查看 Ryan 的作品並閱讀他的文章，網址為 www.boostworthy.com/。

這個 ActionScript 程式碼是以三個方法為基礎所寫成：

- init()：在 init() 方法中，程式碼會載入包含著色器的 Pixel Bender 位元組碼檔案。載入這個檔案時，會呼叫 onLoadComplete() 方法。
- onLoadComplete()：在 onLoadComplete() 方法中，程式碼會建立名為 shader 的 Shader 物件，以及名為 texture 的 Sprite 實體。在 renderShader() 方法中，程式碼會將著色器結果繪製到 texture 內 (每個影格繪製一次)。
- onEnterFrame()：每個影格呼叫 onEnterFrame() 方法一次，製作出動畫效果。在此方法中，程式碼將著色器參數值設定為新顏色，然後呼叫 renderShader() 方法將著色器結果繪製成矩形。

- renderShader()：在 renderShader() 方法中，程式碼會呼叫 Graphics.beginShaderFill() 方法來指定著色器填色。然後，程式碼會繪製一個矩形，其填色是由著色器輸出 (所產生的顏色) 所定義。如需有關以這種方式使用著色器的詳細資訊，請參閱第 261 頁「使用著色器做為繪圖填色」。

下面列出這個範例的 ActionScript 程式碼。範例中會使用這個類別做為主應用程式類別，供 Flash Builder 中的 ActionScript 專案使用，或者做為文件類別以供 Flash Professional 中的 FLA 檔使用：

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ColorFilterExample extends Sprite
    {
        private const DELTA_OFFSET:Number = Math.PI * 0.5;
        private var loader:URLLoader;
        private var shader:Shader;
        private var texture:Sprite;
        private var delta:Number = 0;

        public function ColorFilterExample()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ColorFilter.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            texture = new Sprite();

            addChild(texture);

            addEventListener(Event.ENTER_FRAME, onEnterFrame);
        }
        private function onEnterFrame(event:Event):void
        {
```

```
            shader.data.color.value[0] = 0.5 + Math.cos(delta - DELTA_OFFSET) * 0.5;
            shader.data.color.value[1] = 0.5 + Math.cos(delta) * 0.5;
            shader.data.color.value[2] = 0.5 + Math.cos(delta + DELTA_OFFSET) * 0.5;
            // The alpha channel value (index 3) is set to 1 by the kernel's default
            // value. This value doesn't need to change.

            delta += 0.1;

            renderShader();
        }

        private function renderShader():void
        {
            texture:graphics.clear();
            texture.graphics.beginShaderFill(shader);
            texture.graphics.drawRect(0, 0, stage.stageWidth, stage.stageHeight);
            texture.graphics.endFill();
        }
    }
}
```

下面列出 ColorFilter 著色器核心的原始碼，可用來建立 ColorFilter.pbj Pixel Bender 位元組碼檔案：

```
<languageVersion : 1.0;>
kernel ColorFilter
<
    namespace : "boostworthy::Example";
    vendor : "Ryan Taylor";
    version : 1;
    description : "Creates an image where every pixel has the specified color value.";
>
{
    output pixel4 result;

    parameter float4 color
    <
        minValue:float4(0, 0, 0, 0);
        maxValue:float4(1, 1, 1, 1);
        defaultValue:float4(0, 0, 0, 1);
    >;

    void evaluatePixel()
    {
        result = color;
    }
}
```

如果是使用沒有文件記錄的著色器，您可以檢視 ShaderParameter 物件的 type 屬性，了解陣列中必須包含何種類型的多少元素。type 屬性指出著色器本身中所定義參數的資料類型。如需一份每個參數類型應該會接到的元素類型和數目清單，請參閱「Adobe® Flash® Professional CS5 的 ActionScript® 3.0 參考」中 ShaderParameter.value 屬性所列的項目。

每個 ShaderParameter 物件也都具有 index 屬性，指出該參數在著色器的參數順序中所應佔據的位置。除了這些屬性之外，ShaderParameter 物件也可以具有包含由著色器作者所提供中繼資料值的屬性。例如，作者可以為參數指定下限、上限和預設值等中繼資料值。作者所指定的任何中繼資料值都會加入至 ShaderParameter 物件做為動態屬性。若要檢視這些屬性，請使用 for..in 迴圈，重複執行 ShaderParameter 物件的動態屬性，以識別其中繼資料。下列範例說明如何使用 for..in 迴圈來識別 ShaderParameter 物件的中繼資料。每一個中繼資料值都會加入至 Vector 實體 metadata。請注意，此範例假設 Shader 實體 myShader 已經建立，而且該實體已知具有參數 brightness：

```
var brightness:ShaderParameter = myShader.data.brightness;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in brightness)
{
    if (brightness[prop] is String)
    {
        metadata[metadata.length] = brightness[prop];
    }
}

// do something with the metadata
```

使用著色器

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

Pixel Bender 著色器可在 ActionScript 中做為 Shader 物件使用以後，就可透過多種方式加以運用：

- 著色器繪圖填色：著色器定義使用繪圖 API 所繪製的填色部分
- 混合模式：著色器定義兩個重疊顯示物件之間的混合情形
- 濾鏡：著色器會定義修改視覺內容外觀的濾鏡
- 獨立著色器處理：在未指定輸出的預定用途下執行的著色器處理。您也可以選擇讓著色器在背景中執行，當處理完成之後就可使用結果。這項技巧可用來產生點陣圖資料，也可用來處理非視覺資料。

使用著色器做為繪圖填色

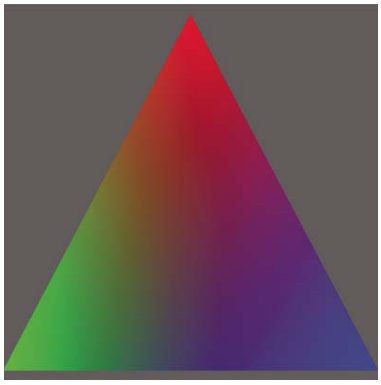
Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

當您使用著色器建立繪圖填色時，必須使用繪圖 API 方法建立向量形狀。著色器的輸出是用來對形狀填色，其方式與使用繪圖 API 將任何點陣圖影像用來做為點陣圖填色的方式相同。若要建立著色器填色，請在您要開始繪製形狀的那段程式碼中呼叫 Graphics 物件的 beginShaderFill() 方法。您可以傳遞 Shader 物件做為 beginShaderFill() 方法的第一個引數，如下列程式碼所示：

```
var canvas:Sprite = new Sprite();
canvas.graphics.beginShaderFill(myShader);
canvas.graphics.drawRect(10, 10, 150, 150);
canvas.graphics.endFill();
// add canvas to the display list to see the result
```

當您使用著色器做為繪圖填色時，必須設定著色器所需的任何輸入影像值與參數值。

下列範例中會示範如何使用著色器做為繪圖填色。在這個範例中，著色器會建立三個點的漸層。這個漸層具有三種顏色，每種顏色都在三角形的三個頂點，三種顏色互相漸層混合。此外，這些顏色還會旋轉，以建立像是動畫的旋轉顏色特效。



備註：這個範例的程式碼是由 Petri Leskinen 所撰寫的。感謝 Petri 與我們分享這個範例。您可以查看更多 Petri 撰寫的範例和教學課程，網址為 http://pixelero.wordpress.com/。

這個 ActionScript 程式碼是以三個方法為基礎所寫成：

- init()：載入應用程式時，會呼叫 init() 方法。在這個方法中，程式碼會設定 Point 物件的初始值，這些物件表示三角形的三個頂點。此外，程式碼還會建立名為 canvas 的 Sprite 實體。之後，在 updateShaderFill() 中，程式碼會將著色器結果繪製到 canvas 內 (每個影格繪製一次)。最後，程式碼會載入著色器位元組碼檔案。
- onLoadComplete()：在 onLoadComplete() 方法中，程式碼會建立名為 shader 的 Shader 物件，以及設定初始參數值。最後，程式碼會加入 updateShaderFill() 方法做為 enterFrame 事件的偵聽程式，這表示每個影格都會呼叫一次這個方法，以建立動畫特效。
- updateShaderFill()：每個影格都會呼叫一次 updateShaderFill() 方法，以建立動畫特效。在這個方法中，程式碼會計算與設定著色器參數的值。然後，程式碼會呼叫 beginShaderFill() 方法以建立著色器填色，並且呼叫其它繪圖 API 方法，在三角形中繪製著色器結果。

下面列出這個範例的 ActionScript 程式碼。範例中會使用這個類別做為主應用程式類別，供 Flash Builder 中的 ActionScript 專案使用，或者做為文件類別以供 Flash Professional 中的 FLA 檔使用：

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Point;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ThreePointGradient extends Sprite
    {
        private var canvas:Sprite;
        private var shader:Shader;
        private var loader:URLLoader;

        private var topMiddle:Point;
        private var bottomLeft:Point;
        private var bottomRight:Point;

        private var colorAngle:Number = 0.0;
        private const d120:Number = 120 / 180 * Math.PI; // 120 degrees in radians
```

```
        public function ThreePointGradient()
        {
            init();
        }

        private function init():void
        {
            canvas = new Sprite();
            addChild(canvas);

            var size:int = 400;
            topMiddle = new Point(size / 2, 10);
            bottomLeft = new Point(0, size - 10);
            bottomRight = new Point(size, size - 10);

            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ThreePointGradient.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            shader.data.point1.value = [topMiddle.x, topMiddle.y];
            shader.data.point2.value = [bottomLeft.x, bottomLeft.y];
            shader.data.point3.value = [bottomRight.x, bottomRight.y];

            addEventListener(Event.ENTER_FRAME, updateShaderFill);
        }

        private function updateShaderFill(event:Event):void
        {
            colorAngle += .06;

            var c1:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle);
            var c2:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle + d120);
            var c3:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle - d120);

            shader.data.color1.value = [c1, c2, c3, 1.0];
            shader.data.color2.value = [c3, c1, c2, 1.0];
            shader.data.color3.value = [c2, c3, c1, 1.0];

            canvas.graphics.clear();
            canvas.graphics.beginShaderFill(shader);

            canvas.graphics.moveTo(topMiddle.x, topMiddle.y);
            canvas.graphics.lineTo(bottomLeft.x, bottomLeft.y);
            canvas.graphics.lineTo(bottomRight.x, bottomLeft.y);

            canvas.graphics.endFill();
        }
    }
}
```

下面列出 ThreePointGradient 著色器核心的原始碼，可用來建立 ThreePointGradient.pbj Pixel Bender 位元組碼檔案：

```
<languageVersion : 1.0;>
kernel ThreePointGradient
<
    namespace : "Petri Leskinen::Example";
    vendor : "Petri Leskinen";
    version : 1;
    description : "Creates a gradient fill using three specified points and colors.";
>
{
    parameter float2 point1 // coordinates of the first point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 0);
    >;

    parameter float4 color1 // color at the first point, opaque red by default
    <
        defaultValue:float4(1.0, 0.0, 0.0, 1.0);
    >;

    parameter float2 point2 // coordinates of the second point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 500);
    >;

    parameter float4 color2 // color at the second point, opaque green by default
    <
        defaultValue:float4(0.0, 1.0, 0.0, 1.0);
    >;

    parameter float2 point3 // coordinates of the third point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 500);
    >;

    parameter float4 color3 // color at the third point, opaque blue by default
    <
        defaultValue:float4(0.0, 0.0, 1.0, 1.0);
    >;
```

```
    output pixel4 dst;

    void evaluatePixel()
    {
        float2 d2 = point2 - point1;
        float2 d3 = point3 - point1;

        // transformation to a new coordinate system
        // transforms point 1 to origin, point2 to (1, 0), and point3 to (0, 1)
        float2x2 mtrx = float2x2(d3.y, -d2.y, -d3.x, d2.x) / (d2.x * d3.y - d3.x * d2.y);
        float2 pNew = mtrx * (outCoord() - point1);

        // repeat the edge colors on the outside
        pNew.xy = clamp(pNew.xy, 0.0, 1.0); // set the range to 0.0 ... 1.0

        // interpolating the output color or alpha value
        dst = mix(mix(color1, color2, pNew.x), color3, pNew.y);
    }
}
```

備註：如果在圖形處理單位 (GPU) 下顯示時使用著色器填色，填色的區域將是青色。

如需有關使用繪圖 API 繪製形狀的詳細資訊，請參閱第 187 頁「使用繪圖 API」。

使用著色器做為混合模式

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

使用著色器做為混合模式就像使用其它混合模式一樣。著色器會定義兩個顯示物件混合在一起所產生的視覺外觀。若要使用著色器做為混合模式，請將 Shader 物件指定給前景顯示物件的 blendShader 屬性。將 null 以外的值指定給 blendShader 屬性時，會自動將顯示物件的 blendMode 屬性設定為 BlendMode.SHADER。下列程式碼會示範如何使用著色器做為混合模式。請注意，這個範例會假設顯示清單上有個名為 foreground 的顯示物件包含在與其它顯示內容相同的父輩中，而且 foreground 與其它內容重疊：

```
foreground.blendShader = myShader;
```

當您使用著色器做為混合模式時，必須使用至少兩個輸入來定義著色器。如上述範例所示，您不需在程式碼中設定輸入值，程式碼會自動使用兩個混合的影像做為著色器輸入。前景影像會設定為第二個影像 (它是已套用混合模式的顯示物件)。背景影像是透過採用合成前景影像的範圍框後方的所有像素所建立。背景影像會設定為第一個輸入影像。如果您將使用兩個以上的輸入來定義著色器，則必須為前兩個輸入之後的任何輸入提供值。

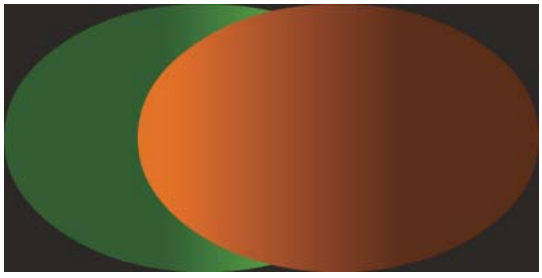
下列範例會示範如何使用著色器做為混合模式。這個範例會使用以明度為基礎的變亮混合模式。混合的結果，就是這兩個混合物件的最亮像素值會變成所顯示的像素。

備註：這個範例的程式碼是由 Mario Klingemann 所撰寫的。感謝 Mario 與我們分享這個範例。您可以查看更多 Mario 的作品並閱讀他的文章，網址為 www.quasimondo.com/。

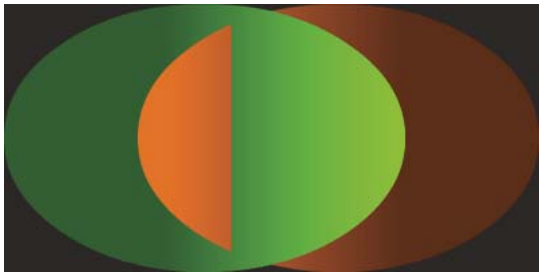
這個重要的 ActionScript 程式碼是以下列兩個方法為基礎所撰寫：

- init()：載入應用程式時，會呼叫 init() 方法。在這個方法中，程式碼會載入著色器位元組碼檔案。
- onLoadComplete()：在 onLoadComplete() 方法中，程式碼會建立名為 shader 的 Shader 物件，再繪製三個物件。第一個物件 backdrop 是暗灰色的背景，位於混合物件後方。第二個物件 backgroundShape 是具有綠色漸層的橢圓。第三個物件 foregroundShape 是具有橘色漸層的橢圓。

foregroundShape 橢圓是混合的前景物件，混合的背景影像則是由 foregroundShape 物件的範圍框所重疊之部分 backdrop 和部分 backgroundShape 所組成。foregroundShape 物件是顯示清單中最前面的物件。這個物件會與 backgroundShape 部分重疊，並且與 backdrop 完全重疊。由於有這樣的重疊情形，因此在未套用混合模式的情況下，橘色橢圓 (foregroundShape) 會完整顯示，而綠色橢圓 (backgroundShape) 則會因為遭橘色橢圓遮蓋而隱藏：



不過，在套用混合模式的情況下，綠色橢圓較亮的部分則會 「露出來」，因為這個部分比 foregroundShape 與其重疊的部分還亮：



下面列出這個範例的 ActionScript 程式碼。範例中會使用這個類別做為主應用程式類別，供 Flash Builder 中的 ActionScript 專案使用，或者做為文件類別以供 Flash Professional 中的 FLA 檔使用：

```
package
{
    import flash.display.BlendMode;
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class LumaLighten extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function LumaLighten()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
```

```
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("LumaLighten.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            var backdrop:Shape = new Shape();
            var g0:Graphics = backdrop.graphics;
            g0.beginFill(0x303030);
            g0.drawRect(0, 0, 400, 200);
            g0.endFill();
            addChild(backdrop);

            var backgroundShape:Shape = new Shape();
            var g1:Graphics = backgroundShape.graphics;
            var c1:Array = [0x336600, 0x80ff00];
            var a1:Array = [255, 255];
            var r1:Array = [100, 255];
            var m1:Matrix = new Matrix();
            m1.createGradientBox(300, 200);
            g1.beginGradientFill(GradientType.LINEAR, c1, a1, r1, m1);
            g1.drawEllipse(0, 0, 300, 200);
            g1.endFill();
            addChild(backgroundShape);

            var foregroundShape:Shape = new Shape();
            var g2:Graphics = foregroundShape.graphics;
            var c2:Array = [0xff8000, 0x663300];
            var a2:Array = [255, 255];
            var r2:Array = [100, 255];
            var m2:Matrix = new Matrix();
            m2.createGradientBox(300, 200);
            g2.beginGradientFill(GradientType.LINEAR, c2, a2, r2, m2);
            g2.drawEllipse(100, 0, 300, 200);
            g2.endFill();
            addChild(foregroundShape);

            foregroundShape.blendShader = shader;
            foregroundShape.blendMode = BlendMode.SHADER;
        }
    }
}
```

下面列出 LumaLighten 著色器核心的原始碼，可用來建立「LumaLighten.pbj」 Pixel Bender 位元組碼檔案：

```
<languageVersion : 1.0;>
kernel LumaLighten
<
    namespace : "com.quasimondo.blendModes";
    vendor : "Quasimondo.com";
    version : 1;
    description : "Luminance based lighten blend mode";
>
{
    input image4 background;
    input image4 foreground;

    output pixel4 dst;

    const float3 LUMA = float3(0.212671, 0.715160, 0.072169);

    void evaluatePixel()
    {
        float4 a = sampleNearest(foreground, outCoord());
        float4 b = sampleNearest(background, outCoord());
        float luma_a = a.r * LUMA.r + a.g * LUMA.g + a.b * LUMA.b;
        float luma_b = b.r * LUMA.r + b.g * LUMA.g + b.b * LUMA.b;

        dst = luma_a > luma_b ? a : b;
    }
}
```

如需有關使用混合模式的詳細資訊，請參閱第 156 頁「套用混合模式」。

備註：當 Pixel Bender 著色器程式在 Flash Player 或 AIR 中以混合模式執行時，取樣與 outCoord() 函數的行為會與在其他內容中的行為不同。在混合模式下，取樣函數永遠會傳回由著色器所評估的目前像素。例如，您無法藉由將偏移量新增至 outCoord() 來取樣相鄰的像素。同樣地，如果您在取樣函數之外使用 outCoord() 函數，其座標會永遠評估為 0。例如，您無法使用像素的位置來影響混合影像的結合方式。

使用著色器做為濾鏡

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

使用著色器做為濾鏡就像在 ActionScript 中使用其它任何濾鏡一樣。當您使用著色器做為濾鏡時，已套用濾鏡的影像 (顯示物件或 BitmapData 物件) 會傳遞至著色器。著色器會使用輸入影像建立濾鏡輸出，這個輸出通常是原始影像的修改版本。如果已套用濾鏡的物件是顯示物件，著色器的輸出就會顯示在螢幕上，取代已套用濾鏡的顯示物件。如果已套用濾鏡的物件是 BitmapData 物件，著色器的輸出就會是已呼叫其 applyFilter() 方法之 BitmapData 物件的內容。

若要使用著色器做為濾鏡，您必須先建立 Shader 物件，如第 255 頁「載入或嵌入著色器」所述。然後，再建立連結至 Shader 物件的 ShaderFilter 物件。ShaderFilter 物件是套用至已套用濾鏡之物件的濾鏡。如同套用任何濾鏡一般，您能以相同方式將這個濾鏡套用至物件。接著，再將這個物件傳遞至顯示物件的 filters 屬性，或者對 BitmapData 物件呼叫 applyFilter() 方法。例如，下列程式碼會建立 ShaderFilter 物件，並對名為 homeButton 的顯示物件套用濾鏡。

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
homeButton.filters = [myFilter];
```

當您使用著色器做為濾鏡時，必須使用至少一個輸入來定義著色器。如上述範例所示，您不需在程式碼中設定輸入值，程式碼會將已套用濾鏡的顯示物件或 BitmapData 物件設定做為輸入影像。如果您將使用一個以上的輸入來定義著色器，則必須為第一個輸入之後的任何輸入提供值。

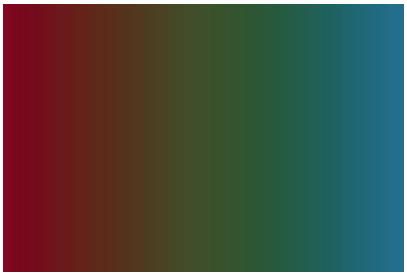
在某些情況下，濾鏡會變更原始影像的尺寸。例如，典型的陰影特效會加入額外的像素，其中包含已加入至影像的陰影。當您使用會變更影像尺寸的著色器時，請設定 leftExtension、rightExtension、topExtension 和 bottomExtension 屬性，指出希望的影像大小變更量。

在下列範例中，會示範如何使用著色器做為濾鏡。在這個範例中，濾鏡會將影像的紅色、綠色和藍色色版值變成相反值，以產生具有 「負片」特效的影像。

備註：這個範例使用的著色器是隨附於 Pixel Bender Toolkit 的 invertRGB.pbk Pixel Bender 核心。您可以從 Pixel Bender Toolkit 的安裝目錄載入這個核心的原始碼。請編譯這個原始碼，並將位元組碼檔案儲存在與原始碼相同的目錄中。

這個重要的 ActionScript 程式碼是以下列兩個方法為基礎所撰寫：

- init()：載入應用程式時，會呼叫 init() 方法。在這個方法中，程式碼會載入著色器位元組碼檔案。
- onLoadComplete()：在 onLoadComplete() 方法中，程式碼會建立名為 shader 的 Shader 物件，再建立及繪製名為 target 之物件的內容。target 物件是以線性漸層顏色填滿的矩形，漸層顏色在左邊為紅色，中間為黃綠色，右邊為淡藍色。未套用濾鏡的物件看起來如下：



套用濾鏡之後，矩形會呈現相反的顏色，看起來如下：

這個範例使用的著色器是隨附於 Pixel Bender Toolkit 的 「invertRGB.pbk」樣本 Pixel Bender 核心。您可以在 Pixel Bender Toolkit 安裝目錄的 「invertRGB.pbk」檔案中找到原始碼。請編譯這個原始碼，並將位元組碼檔案 (命名為 「invertRGB.pbj」) 儲存在與 ActionScript 原始碼相同的目錄中。

下面列出這個範例的 ActionScript 程式碼。範例中會使用這個類別做為主應用程式類別，供 Flash Builder 中的 ActionScript 專案使用，或者做為文件類別以供 Flash Professional 中的 FLA 檔使用：

```
package
{
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.filters.ShaderFilter;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class InvertRGB extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function InvertRGB()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("invertRGB.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            var target:Shape = new Shape();
            addChild(target);

            var g:Graphics = target.graphics;
            var c:Array = [0x990000, 0x445500, 0x007799];
            var a:Array = [255, 255, 255];
            var r:Array = [0, 127, 255];
            var m:Matrix = new Matrix();
            m.createGradientBox(w, h);
            g.beginGradientFill(GradientType.LINEAR, c, a, r, m);
            g.drawRect(10, 10, w, h);
            g.endFill();

            var invertFilter:ShaderFilter = new ShaderFilter(shader);
            target.filters = [invertFilter];
        }
    }
}
```

如需有關套用濾鏡的詳細資訊，請參閱第 226 頁「建立與套用濾鏡」。

在獨立模式中使用著色器

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

當您在獨立模式中使用著色器時，著色器程序的執行方式與輸出的使用方式無關。您必須指定要執行的著色器、設定輸入和參數值，然後指定要在其中放置結果資料的物件。基於下列兩個原因，您可以在獨立模式中使用著色器：

- 處理非影像的資料：在獨立模式中，您可以選擇將任意二進位或數字資料 (而非點陣圖影像資料) 傳遞至著色器。除了點陣圖影像資料以外，您還可以選擇要以二進位資料或數字資料傳回著色器結果。
- 背景處理：當您在獨立模式中執行著色器時，著色器預設會以非同步方式執行。這表示，在應用程式繼續執行期間，著色器會於背景執行，而在著色器處理完成時，您的程式碼會收到通知。您可以使用需要長時間執行的著色器，在執行期間，著色器並不會導致應用程式使用者介面或其它程序發生無法回應的情況。

您可以使用 ShaderJob 物件，在獨立模式中執行著色器。首先，您必須建立 ShaderJob 物件，並將其連結至表示要執行之著色器的 Shader 物件。

```
var job:ShaderJob = new ShaderJob(myShader);
```

接著，再設定著色器所需的任何輸入或參數值。如果您要在背景執行著色器，請為 ShaderJob 物件的 complete 事件註冊偵聽程式。當著色器完成工作時，便會呼叫您註冊的偵聽程式：

```
function completeHandler(event:ShaderEvent):void
{
    // do something with the shader result
}

job.addEventListener(ShaderEvent.COMPLETE, completeHandler);
```

接下來，請建立在著色器作業完成時，要在其中寫入作業結果的物件。然後，再將該物件指定給 ShaderJob 物件的 target 屬性：

```
var jobResult:BitmapData = new BitmapData(100, 75);
job.target = jobResult;
```

如果您要使用 ShaderJob 執行影像處理，請將 BitmapData 實體指定給 target 屬性。如果您要處理二進位或數字資料，請將 ByteArray 物件或 Vector.<Number> 實體指定給 target 屬性。在這種情況下，您必須設定 ShaderJob 物件的 width 和 height 屬性，指定要輸出至 target 物件的資料量。

備註：您可以傳遞引數至 ShaderJob() 建構函式，一次設定 ShaderJob 物件的 shader、target、width 和 height 屬性，如下所示：var job:ShaderJob = new ShaderJob(myShader, myTarget, myWidth, myHeight);

當您要開始執行著色器時，請呼叫 ShaderJob 物件的 start() 方法：

```
job.start();
```

根據預設，呼叫 start() 可以讓 ShaderJob 以非同步方式執行。在這種情況下，程式會立刻從下一行程式碼繼續執行，而不需要等待著色器作業完成。當著色器作業完成時，ShaderJob 物件會呼叫其 complete 事件偵聽程式，告知這些偵聽程式作業已經完成。在這個時候 (也就是在 complete 事件偵聽程式的主體中)，target 物件會包含著色器作業結果。

備註：您可以直接從傳遞至偵聽程式方法的事件物件擷取著色器結果，而不必使用 target 屬性。這個事件物件是 ShaderEvent 實體。視您設定為 target 屬性之物件的資料類型，ShaderEvent 物件具有可用來存取結果的三個屬性：ShaderEvent.bitmapData、ShaderEvent.byteArray 和 ShaderEvent.vector。

此外，您也可以傳遞 true 引數至 start() 方法。在這種情況下，著色器作業會以同步方式執行。如此一來，所有程式碼 (包括與使用者介面的互動及其它事件) 都會暫停，直到著色器作業完成為止。當著色器作業完成時，target 物件會包含著色器結果，然後程式會從下一行程式碼繼續執行。

```
job.start(true);
```

第 16 章 使用影片片段

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 Adobe® Flash® 開發環境中，MovieClip 類別是建立動畫與影片片段元件的核心類別。它可以顯示物件的所有行為與功能，而且還可以控制其時間軸的其它屬性和方法。

影片片段的基本觀念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

對於使用 Flash 編寫工具建立動畫內容，以及要透過 ActionScript 控制該內容的人而言，影片片段是非常重要的元素。每當您在 Flash 中建立影片片段元件時，Flash 會將元件加入 Flash 文件的元件庫中。根據預設，這個元件會變成 MovieClip 類別的實體，也因此會有 MovieClip 類別的屬性與方法。

將影片片段元件的實體放置在「舞台」上時，除非其播放作業是使用 ActionScript 來變更，否則影片片段會自動依其時間軸 (如果它具有一個以上的時間軸) 進行。這就是時間軸與 MovieClip 類別之間的區別。前者可以透過 Flash 編寫工具，讓您利用移動或形狀補間來建立動畫。相較之下，透過屬於 Sprite 類別之實體的顯示物件的話，您只能以程式設計的方式變更物件的值來建立動畫。

在舊版的 ActionScript 中，MovieClip 類別是「舞台」上所有實體的基底類別。在 ActionScript 3.0 中，影片片段只是可以出現在螢幕上的許多顯示物件當中的一種。如果顯示物件的功能不需要用到時間軸，使用 Shape 類別或 Sprite 類別來替代 MovieClip 類別也許可以改善顯示的效能。如需有關針對工作選擇適當顯示物件的詳細資訊，請參閱第 145 頁「選擇 DisplayObject 子類別」。

重要概念與術語

下列參考清單包含與影片片段相關的重要術語：

AVM1 SWF 使用 ActionScript 1.0 或 ActionScript 2.0 建立的 SWF 檔，通常適用於 Flash Player 8 或更早版本。

AVM2 SWF 使用 ActionScript 3.0 建立的 SWF 檔，適用於 Adobe Flash Player 9 或更新版本或者 Adobe AIR。

外部 SWF 與專案 SWF 分開建立的 SWF 檔，用以載入至 SWF 專案檔並在該 SWF 檔中播放。

影格 時間軸上最小的時間片段。與電影膠片一樣，每個影格就像動畫的即時快照，而且當連續快速播放影格時，就會建立動畫的效果。

時間軸 連續影格的比喻表示法，構成影片片段的連續動畫。MovieClip 物件的時間軸相當於 Flash 編寫工具中的時間軸。

播放磁頭 在時間軸中用來識別位置 (影格) 的標記，它會顯示在指定時刻。

使用 MovieClip 物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您發佈 SWF 檔時，Flash 會將「舞台」上所有的影片片段元件實體都轉換為 MovieClip 物件。您可以在「屬性」檢測器的「實體名稱」欄位中，指定影片片段元件的實體名稱，讓 ActionScript 可以使用該元件。建立 SWF 檔時，Flash 會產生一段程式碼，這段程式碼會在「舞台」上建立 MovieClip 實體，並使用該實體名稱來宣告變數。如果在其它已命名的影片片段

中有巢狀結構的已命名影片片段，則這些子影片片段即視為父影片片段的屬性，也就是說，您可以使用點語法來存取子影片片段。例如，如果實體名稱為 childClip 的影片片段是以巢狀方式存在於另一個實體名稱為 parentClip 的片段中，您便可以呼叫此程式碼，播放子片段的時間軸動畫：

```
parentClip.childClip.play();
```

備註：放在 Flash 編寫工具的 「舞台」上的子實體，無法從父實體的建構函式內以程式碼存取，因為在程式碼執行時這些實體尚未建立。存取子系之前，父輩必須改為利用程式碼建立子實體，或是延遲回呼函數的存取，該函數會偵聽子系，以傳送其 Event.ADDED_TO_STAGE 事件。

在 ActionScript 2.0 的 MovieClip 類別中只有一些舊版的方法和屬性仍然維持不變，但是其它方法和屬性都已經變更。字首為底線的所有屬性都已經重新命名。例如，_width 和 _height 屬性現在已經改為 width 和 height，而 _xscale 和 _yscale 則是改為 scaleX 和 scaleY。如需 MovieClip 類別的屬性和方法完整清單，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。

控制影片片段播放作業

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash 是使用內含時間軸的方式來播放動畫或變更播放時的狀態。使用時間軸的任何視覺元素都必須是屬於 MovieClip 物件或是擴充自 MovieClip 類別的物件。雖然 ActionScript 可以控制任何影片片段停止、播放或前往時間軸上的其它點，但不能用來以動態方式建立時間軸或在特定影格增加內容。這項作業只能由 Flash 編寫工具來執行。

在播放 MovieClip 時，它會沿著時間軸並依據 SWF 檔的影格速率所指定的速度播放。或者，您也可以設定 ActionScript 的 Stage.frameRate 屬性，覆寫這項設定。

播放和停止影片片段

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

play() 和 stop() 方法可以跨越其時間軸對影片片段執行基本的控制。例如，假設在 「舞台」上有影片片段元件，其中包含腳踏車移過螢幕的動畫，而其實體名稱設定為 bicycle。如果下列程式碼附加到主要時間軸上的關鍵影格，

```
bicycle.stop();
```

腳踏車將不會移動 (其動畫將不會播放)。透過其它某些使用者互動，便可以開始移動腳踏車。例如，如果您有名為 startButton 的按鈕，在主要時間軸上的關鍵影格之下列程式碼將可以達到按一下按鈕使動畫播放的目的：

```
// This function will be called when the button is clicked. It causes the
// bicycle animation to play.
function playAnimation(event:MouseEvent):void
{
    bicycle.play();
}
// Register the function as a listener with the button.
startButton.addEventListener(MouseEvent.CLICK, playAnimation);
```

向前快轉與倒帶

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

play() 和 stop() 方法並不是控制影片片段中播放的唯一方式。您也可以使用 nextFrame() 和 prevFrame() 方法，手動將播放磁頭沿著時間軸向前或向後移動。呼叫上述任一種方法時，都會先停止播放，然後分別向前或向後將播放磁頭移動一個影格。

每當觸發影片片段物件的 enterFrame 事件時，使用 play() 方法便與呼叫 nextFrame() 類似。您可以為 enterFrame 事件建立事件偵聽程式，並通知 bicycle 在偵聽程式函數中前往它的前一個影格，讓 bicycle 影片片段向後播放，如下列程式碼所示：

```
// This function is called when the enterFrame event is triggered, meaning
// it's called once per frame.
function everyFrame(event:Event):void
{
    if (bicycle.currentFrame == 1)
    {
        bicycle.gotoAndStop(bicycle.totalFrames);
    }
    else
    {
        bicycle.prevFrame();
    }
}
bicycle.addEventListener(Event.ENTER_FRAME, everyFrame);
```

在正常播放作業中，如果影片片段不只有單一影格，在播放時則將不斷地循環。也就是說，當播放到最後的影格後，它會回到「影格 1」繼續循環。當您使用 prevFrame() 或 nextFrame() 時，此行為不會自動發生 (當播放磁頭在 「影格 1」時呼叫 prevFrame()，不會將播放磁頭移到最後一個影格)。在上述範例中的 if 條件會檢查播放磁頭是否已經向後回到第一個影格，然後將播放磁頭設定為前進到最後的影格，以便有效地建立一個繼續向後播放的影片片段迴圈。

跳至不同的影格與使用影格標籤

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

將影片片段傳送至新的影格是一項簡單的工作。只要呼叫 gotoAndPlay() 或 gotoAndStop() 其中一種，就可以讓影片片段跳至指定為參數的影格編號。另一種方式是，您可以傳遞一個與影格標籤名稱相符的字串。您可以為時間軸上的任何影格指定標籤。如果要執行這項作業，請在時間軸上選取一個影格，然後在 「屬性」檢測器的 「影格標籤」欄位中輸入名稱。

使用影格標籤取代影格編號的優點，在建立複雜的影片片段時特別明顯。當動畫中的影格、圖層和補間的數量愈來愈多時，您可以考慮為重要的影格加上標籤，用來詳細描述影片片段行為的變化 (例如，「靜止」、「走動」或 「跑動」)。這種方式可提高程式碼的可讀性，同時也提供了更大的彈性，因為 ActionScript 呼叫而前往已加上標籤之影格的是指向單一參考 (標籤) 的指標，而不是特定的影格編號。如果您稍後決定將動畫的特定部分移到不同的影格，那麼只要在新位置中為這些影格保留相同的標籤，就不需要變更 ActionScript 程式碼。

為了在程式碼中顯示影格標籤，ActionScript 3.0 還包括了 FrameLabel 類別。此類別的每個實體都代表單一影格標籤，而且具有代表影格標籤名稱 (在 「屬性」檢測器中指定) 的 name 屬性，另外還有代表影格數目的 frame 屬性，該影格的標籤是放置在時間軸上。

為了存取與影片片段實體關聯的 FrameLabel 實體，MovieClip 類別包括會直接傳回 FrameLabel 物件的兩個屬性。currentLabels 屬性會傳回陣列，其中含有橫跨整個影片片段時間軸的所有 FrameLabel 物件。currentLabel 屬性則會傳回字串，其中包含最近在時間軸上所遇到的影格標籤名稱。

假設您已建立影片片段 robot 並且為它的一些動畫狀態加上標籤，您可以設定一個條件來檢查 currentLabel 屬性，以存取目前 robot 的狀態，如下列程式碼所示：

```
if (robot.currentLabel == "walking")
{
    // do something
}
```

Flash Player 11.3 和 AIR 3.3 為 FrameLabel 類別新增了 frameLabel 事件。您可以將事件處理常式指定給代表影格標籤的 FrameLabel 實體。當播放磁頭進入影格時，會傳送事件。

下列範例會在 MovieClip 的影格標籤陣列中，為第二個影格標籤建立 FrameLabel 實體。然後，為 frameLabel 事件註冊事件偵聽程式：

```
var myFrameLabel:FrameLabel = robot.currentLabels[1];
myFrameLabel.addEventListener(Event.FRAME_LABEL, onFrameLabel);

function onFrameLabel(e:Event):void {
    //do something
}
```

使用場景

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 Flash 編寫環境中，您可以使用場景來區分 SWF 檔將要處理的各個時間軸。您可以使用 gotoAndPlay() 或 gotoAndStop() 方法的第二個參數，指定播放磁頭要移動的目的地場景。所有 FLA 檔都是由一個起始場景開始，但是您可以建立新場景。

使用場景不一定是最佳的做法，因為場景會有一些缺點。Flash 文件中如果含有多個場景，維護作業會變得很困難，特別是在具有多位製作人員的環境中。使用多個場景也會讓頻寬變得不具效率，因為發佈程序會將所有場景合併成單一時間軸。這種情況會造成以漸進方式下載所有場景，即使從未播放過這些場景也是如此。基於這些原因，除非是為了組織多個以時間軸為基礎的長篇動畫，否則，通常不建議您使用多個場景。

MovieClip 類別的 scenes 屬性會傳回 Scene 物件的陣列，代表 SWF 檔中的所有場景。currentScene 屬性會傳回 Scene 物件，代表目前播放的場景。

Scene 類別具有數個提供場景資訊的屬性。labels 屬性會傳回 FrameLabel 物件的陣列，代表該場景中的影格標籤。name 屬性會以字串形式傳回場景的名稱。numFrames 屬性會傳回 int，代表場景中的影格總數。

使用 ActionScript 建立 MovieClip 物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 Flash 中將內容加入螢幕的其中一種方法，就是將資源從元件庫拖曳到 「舞台」上，但這並不是唯一一項工作流程。對於複雜的專案，資深的開發人員通常較喜歡採用程式設計的方式來建立影片片段。使用這個方法有幾種好處：重複使用程式碼比較容易、可以加快編譯階段的速度，以及只有在 ActionScript 中才可以使用的細部修改作業。

ActionScript 3.0 的顯示清單 API 能讓以動態方式建立 MovieClip 物件的程序更有效率。能夠直接初始化 MovieClip 實體的功能，可將它加入顯示清單的程序分開，以提供彈性與簡單性，而不會犧牲操控性。

在 ActionScript 3.0 中，當您以程式設計方式建立影片片段 (或任何其它顯示物件) 實體時，在針對顯示物件容器呼叫 addChild() 或 addChildAt() 方法並將它加入顯示清單之前，它不會顯示在螢幕上。這種方式可以讓您建立影片片段、設定其屬性，甚至在螢幕上顯示該影片片段之前呼叫方法。如需有關使用顯示清單的詳細資訊，請參閱第 135 頁 「使用顯示物件容器」。

匯出元件庫元件給 ActionScript 使用

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

依據預設，在 Flash 文件之元件庫中的影片片段元件實體無法以動態方式建立 (亦即，僅能使用 ActionScript 來建立)。這是因為要匯出在 ActionScript 中使用的每一個元件都會增加 SWF 檔的大小，而且已知有些元件可能不會用於舞台上。因此，為了讓某個元件可在 ActionScript 中使用，您必須指定應該將該元件匯出給 ActionScript 使用。

匯出元件給 **ActionScript** 使用：

1. 在 「元件庫」面板中選取元件，然後開啟其 「元件屬性」對話方塊。
2. 若有需要，請啟動 「進階」設定。

3 在「連結」區段中，啟動「匯出給 ActionScript 使用」核取方塊。

這麼做將會啟動「類別」和「基底類別」欄位。

根據預設，會將移除空格的元件名稱填入「類別」欄位 (例如，名稱為「Tree House」的元件會變成「TreeHouse」)。若要指定元件應該為其行為使用自訂類別，請輸入類別的完整名稱，包括它在這個欄位中的套件。如果您想要在 ActionScript 中能夠建立元件的實體，但是不需要加入任何其它行為，則可以保留原來的類別名稱。

「基底類別」欄位的值預設為 flash.display.MovieClip。如果您希望元件能擴充另一個客戶類別的功能，只要該類別會擴充 Sprite (或 MovieClip) 類別，您就可以改指定該類別的名稱。

4 按「確定」按鈕儲存變更。

此時，如果 Flash 找不到連結的 SWC 檔案或具有指定類別之定義的外部 ActionScript 檔案 (例如，如果您不需要為元件加入其他行為)，就會顯示警告：

「類別路徑中找不到這個類別的定義，將在匯出時自動產生至 SWF 檔」。

如果您的元件庫元件需要的功能不會超出 MovieClip 類別提供的範圍，就可以忽略這個警告。

如果您沒有為元件提供類別，Flash 將會為元件建立一個與此類別相等的類別：

```
package
{
    import flash.display.MovieClip;

    public class ExampleMovieClip extends MovieClip
    {
        public function ExampleMovieClip()
        {
        }
    }
}
```

如果您希望在元件中加入額外的 ActionScript 功能，請在以下的程式碼結構內加入適當的屬性和方法。例如，假設您的影片片段元件包含寬度及高度都為 50 個像素的圓形，並且將元件指定為匯出給 ActionScript 並搭配名稱為 Circle 的類別使用。置入 Circle.as 檔案之後，下列程式碼就可以擴充 MovieClip 類別，並將額外的方法 getArea() 和 getCircumference() 提供給元件：

```
package
{
    import flash.display.MovieClip;

    public class Circle extends MovieClip
    {
        public function Circle()
        {
        }

        public function getArea():Number
        {
        // The formula is Pi times the radius squared.
        return Math.PI * Math.pow((width / 2), 2);
        }

        public function getCircumference():Number
        {
        // The formula is Pi times the diameter.
        return Math.PI * width;
        }
    }
}
```

下列程式碼 (放置在 Flash 文件之「影格 1」的關鍵影格上) 將會建立元件實體，並將它顯示在螢幕上：

```
var c:Circle = new Circle();
addChild(c);
trace(c.width);
trace(c.height);
trace(c.getArea());
trace(c.getCircumference());
```

這段程式碼會示範以 ActionScript 為架構的實體化，做為將個別資源拖曳到「舞台」的替代方案。它建立的圓形將具有影片片段的所有屬性，而且也有在 Circle 類別中定義的自訂方法。這是一個很基本的範例，您的元件庫元件可以在它的類別中指定任何數目的屬性和方法。

以 ActionScript 為架構的實體化具有強大的功能，因為它允許您以動態方式建立大量的實體，如果是以手動方式建立這些實體，則將是非常冗長而單調的作業。它同樣具備了很大的彈性，因為您可以在建立階段自訂每一個實體的屬性。當您使用迴圈以動態方式建立數個 Circle 實體時，就會明顯感覺到它帶來的好處。使用先前在 Flash 文件之元件庫中所描述的 Circle 元件與類別，將下列程式碼放在「影格 1」的關鍵影格上：

```
import flash.geom.ColorTransform;

var totalCircles:uint = 10;
var i:uint;
for (i = 0; i < totalCircles; i++)
{
    // Create a new Circle instance.
    var c:Circle = new Circle();
    // Place the new Circle at an x coordinate that will space the circles
    // evenly across the Stage.
    c.x = (stage.stageWidth / totalCircles) * i;
    // Place the Circle instance at the vertical center of the Stage.
    c.y = stage.stageHeight / 2;
    // Change the Circle instance to a random color
    c.transform.colorTransform = getRandomColor();
    // Add the Circle instance to the current timeline.
    addChild(c);
}

function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

這為您示範了如何使用程式碼快速建立和自訂某個元件的多個實體。每一個實體都會根據迴圈內目前的計數重新定位，並且設定其 transform 屬性 (Circle 會藉由擴充 MovieClip 類別的方式來繼承) 以賦予每一個實體一種隨機顏色。

載入外部 SWF 檔

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 ActionScript 3.0 中，SWF 檔是透過 Loader 類別載入。若要載入外部 SWF 檔，ActionScript 需要執行四項工作：

1 使用檔案的 URL 建立新的 URLRequest 物件。

2 建立新的 Loader 物件。

3 呼叫 Loader 物件的 load() 方法，將 URLRequest 實體當做參數傳遞。

4 針對顯示物件容器 (例如 Flash 文件的主要時間軸) 呼叫 addChild() 方法，以便將 Loader 實體加入顯示清單。

最後，完成的程式碼看起來像這樣：

```
var request:URLRequest = new URLRequest("http://www.[yourdomain].com/externalSwf.swf");
var loader:Loader = new Loader()
loader.load(request);
addChild(loader);
```

這個相同的程式碼透過指定影像檔案的 URL (而不是 SWF 檔的 URL)，可用來載入如 JPEG、GIF 或 PNG 影像等外部影像檔案。SWF 檔和影像檔案的不同點是它可以包含 ActionScript。因此，雖然載入 SWF 檔也許和載入影像的程序相同，但是如果 Flash Player 或 AIR 正在播放 SWF，而且您想要使用 ActionScript 以任何方式與外部 SWF 檔進行通訊，那麼載入外部 SWF 檔時，載入端 SWF 檔與被載入端 SWF 檔都必須位於相同的安全執行程序中。此外，如果外部 SWF 檔中的類別和載入端 SWF 檔中的類別共享相同的命名空間時，您可能需要為被載入端 SWF 檔建立新的應用程式網域，如此才能避免命名空間衝突。如需有關安全及應用程式網域考量的詳細資訊，請參閱第 125 頁 「使用應用程式網域」及第 914 頁 「載入內容」。

成功載入外部 SWF 檔後，您就可以透過 Loader.content 屬性來存取這個檔案。如果此外部 SWF 檔已經發佈為 ActionScript 3.0 的格式，根據它所擴充的類別型態，物件可以是一個影片片段或 Sprite。

與其他平台相比，在 Adobe AIR for iOS 中載入 SWF 檔案有些不同。如需詳細資訊，請參閱第 169 頁 「在 AIR for iOS 中載入 SWF 檔案」。

載入舊版 SWF 檔的考量事項

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如果外部 SWF 檔已經使用舊版的 ActionScript 發佈，就必須考慮一些重要的限制事項。和執行於 AVM2 (ActionScript Virtual Machine 2) 中的 ActionScript 3.0 SWF 檔不同的是，使用 ActionScript 1.0 或 2.0 發佈的 SWF 檔是在 AVM1 (ActionScript Virtual Machine 1) 中執行。

將 ActionScript 1.0 或 2.0 SWF 檔載入 ActionScript 3.0 SWF 檔時，(比起載入 ActionScript 3.0 SWF 檔) 有很大的差異。Flash Player 也對先前發佈的內容提供完整的舊版相容性。能在舊版 Flash Player 中執行的任何內容都可以在支援 ActionScript 3.0 的 Flash Player 中執行。但有下列限制：

- ActionScript 3.0 程式碼可載入 ActionScript 1.0 或 2.0 中所撰寫的 SWF 檔。當 ActionScript 1.0 或 2.0 SWF 檔成功載入時，載入的物件 (Loader.content 屬性) 會是 AVM1Movie 物件。AVM1Movie 實體與 MovieClip 實體不同。而是顯示物件，但和影片片段不同的是，它不包括與時間軸相關的方法或屬性。父輩 AVM2 SWF 檔將無法存取所載入 AVM1Movie 物件的屬性、方法或物件。
- 以 ActionScript 1.0 或 2.0 撰寫的 SWF 檔無法載入以 ActionScript 3.0 撰寫的 SWF 檔。這表示，以 Flash 8、Flex Builder 1.5 或更早版本所編寫的 SWF 檔無法載入 ActionScript 3.0 SWF 檔。

  唯一的例外是，只要 ActionScript 2.0 SWF 檔先前未將任何內容載入其任何階層，就可以用 ActionScript 3.0 SWF 檔取代本身。ActionScript 2.0 SWF 檔可以呼叫 loadMovieNum()，然後將值 0 傳遞至 level 參數，藉以達到此目的。
- 一般來說，以 ActionScript 1.0 或 2.0 撰寫的 SWF 檔必須進行移轉，才能和 ActionScript 3.0 撰寫的 SWF 檔一起使用。例如，假設您以 ActionScript 2.0 建立了媒體播放程式。這個媒體播放程式會載入同樣以 ActionScript 2.0 所建立的各種內容。但是，您不能以 ActionScript 3.0 建立新內容，並在該媒體播放程式中載入此內容。您必須將這個媒體播放程式移轉為 ActionScript 3.0。

  但是，如果您使用 ActionScript 3.0 建立媒體播放程式，這個媒體播放程式就可以直接載入 ActionScript 2.0 內容。

下表摘要說明舊版 Flash Player 載入新內容和執行指令碼編寫的限制，以及不同 ActionScript 版本所撰寫的 SWF 檔在跨指令碼處理方面有哪些限制。

支援的功能	Flash Player 7	Flash Player 8	Flash Player 9 和 10
可以載入哪些 SWF 版本	7 和更早版本	8 和更早版本	9 (或 10) 和更早版本
包含的 AVM	AVM1	AVM1	AVM1 和 AVM2
可執行哪些 ActionScript 版本撰寫的 SWF	1.0 和 2.0	1.0 和 2.0	1.0、2.0 和 3.0

在下表中，「支援的功能」表示在 Flash Player 9 或更新版本中執行的內容。在 Flash Player 8 或更早版本中執行的內容只能載入、顯示、執行和跨指令碼處理 ActionScript 1.0 和 2.0。

支援的功能	使用 ActionScript 1.0 and 2.0 建立的內容	使用 ActionScript 3.0 建立的內容
可以載入的內容和執行程式碼的內容版本	僅 ActionScript 1.0 和 2.0	ActionScript 1.0 和 2.0 以及 ActionScript 3.0
可以跨指令碼處理的內容版本	僅 ActionScript 1.0 和 2.0 (透過本機連線的 ActionScript 3.0)	透過本機連線的 ActionScript 1.0 和 2.0。 ActionScript 3.0

移動影片片段範例：RuntimeAssetsExplorer

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

對用於一個以上專案的元件庫而言，「匯出給 ActionScript 使用」功能特別有用。如果 Flash Player 或 AIR 執行 SWF 檔，匯出至 ActionScript 的元件可供將它載入的同一個安全執行程序內的任何 SWF 檔使用。使用這種方式，單一 Flash 文件就可以產生一個 SWF 檔，並指定它只能用來存放圖像資源。這項技巧特別適用於大型專案，使用視覺資源的設計人員可以和以後會建立「包裝函式」SWF 檔並在執行階段載入圖像資源 SWF 檔的開發人員一起協同作業。您可以使用這個方法來維護一系列的檔案版本，讓圖像資源獨立於程式設計開發的進度之外。

RuntimeAssetsExplorer 應用程式會載入本身是 RuntimeAsset 之子類別的任何 SWF 檔，可以讓您瀏覽該 SWF 檔的可用資源。此範例將說明下列各項：

- 使用 Loader.load() 載入外部 SWF 檔
- 以動態方式建立匯出給 ActionScript 使用的元件庫元件
- 控制 MovieClip 播放的 ActionScript 控制項

在開始之前，請注意在 Flash Player 中執行的每一個 SWF 檔都必須位於相同的安全執行程序中。如需詳細資訊，請參閱第 901 頁 「安全執行程序」。

若要取得此樣本的應用程式檔案，請下載 Flash Professional 樣本。您可以在 Samples/RuntimeAssetsExplorer 檔案夾中找到 RuntimeAssetsExplorer 應用程式檔案。此應用程式是由下列檔案組成：

檔案	說明
RuntimeAssetsExample.mxml 或 RuntimeAssetsExample.fla	應用程式使用者介面，在 Flash 中為 FLA，在 Flex 中為 MXML。
RuntimeAssetsExample.as	Flash (FLA) 應用程式的文件類別
GeometricAssets.as	實作 RuntimeAsset 介面的類別範例。
GeometricAssets.fla	一個連結到 GeometricAssets 類別 (FLA 的文件類別) 的 FLA 檔，該類別包含匯出供 ActionScript 使用的元件。

檔案	說明
com/example/programmingas3/runtimeassetexplorer/RuntimeLibrary.as	介面，定義將載入至 Explorer 容器之所有執行階段資源 SWF 檔所需的方法。
com/example/programmingas3/runtimeassetexplorer/AnimatingBox.as	在旋轉方塊的圖形中之元件庫元件類別。
com/example/programmingas3/runtimeassetexplorer/AnimatingStar.as	在旋轉星形的形狀中之元件庫元件類別。

建立執行階段元件庫介面

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

為使 Explorer 可以與 SWF 元件庫適當地互動，必須先將執行階段資源元件庫的結構形式化。為完成這項作業，我們將先建立一個介面，它與類別類似，因為它是用來界定預期結構的方法藍圖，但與類別不同的是，它不包括方法主體。此介面可提供一種方法，讓執行階段元件庫與 Explorer 互相通訊。載入至瀏覽器的每一個執行階段資源 SWF 都將實作這個介面。如需有關介面以及它們如何能更有用的詳細資訊，請參閱 「學習 ActionScript 3.0」中的 「介面」。

RuntimeLibrary 介面會很簡單，只需要使用一個函數，就可以將要匯出並可在執行階段元件庫中使用的元件類別路徑陣列提供給結構檢視器。在最後的結果中，這個介面將具有一個方法：getAssets()。

```
package com.example.programmingas3.runtimeassetexplorer
{
    public interface RuntimeLibrary
    {
        function getAssets():Array;
    }
}
```

建立資源元件庫 SWF 檔

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

藉由定義 RuntimeLibrary 介面，便可以建立多個能夠載入至其它 SWF 檔的資源元件庫 SWF 檔。建立 SWF 資源元件庫包括四項工作：

- 建立資源元件庫 SWF 檔的類別
- 為元件庫中包含的個別資源建立類別
- 建立實際的圖像資源
- 將圖像元素與類別建立關聯，然後發佈這個元件庫 SWF

建立類別以實作 **RuntimeLibrary** 介面

接著，我們將建立 GeometricAssets 類別來實作 RuntimeLibrary 介面。這將變成 FLA 的文件類別。此類別的程式碼與 RuntimeLibrary 介面非常類似，它們之間的差異是在類別定義中 getAssets() 方法具有方法主體。

```
package
{
    import flash.display.Sprite;
    import com.example.programmingas3.runtimeassetexplorer.RuntimeLibrary;

    public class GeometricAssets extends Sprite implements RuntimeLibrary
    {
        public function GeometricAssets() {

        }
        public function getAssets():Array {
            return [ "com.example.programmingas3.runtimeassetexplorer.AnimatingBox",
                     "com.example.programmingas3.runtimeassetexplorer.AnimatingStar" ];
        }
    }
}
```

如果我們先建立第二個執行階段元件庫，就可以依據其它可以提供其 getAssets() 實作的類別 (例如，AnimationAssets) 來建立另一個 FLA。

建立每一個 **MovieClip** 資源的類別

在這個範例中，我們將只擴充 MovieClip 類別，而不將任何功能加入自訂資源。下列 AnimatingStar 的程式碼與 AnimatingBox 的程式碼很類似：

```
package com.example.programmingas3.runtimeassetexplorer
{
    import flash.display.MovieClip;

    public class AnimatingStar extends MovieClip
    {
        public function AnimatingStar() {
        }
    }
}
```

發佈元件庫

現在我們藉由建立新的 FLA，並在 「屬性」檢測器的 「文件類別」欄位中輸入 GeometricAssets，將 MovieClip 架構的資源連接至新的類別。為了達到這個範例的目標，我們將建立兩個很基本的形狀，這兩個形狀會使用時間軸補間動畫對 360 個影格進行逆時針方向旋轉。將 animatingBox 和 animatingStar 元件設為 「匯出給 ActionScript 使用」，並將 「類別」欄位設為 getAssets() 實作中指定的各個類別路徑。在我們將標準 MovieClip 方法設為子類別時，flash.display.MovieClip 的預設基底類別仍然維持不變。

設定元件的匯出設定之後，發佈 FLA。您現在已經擁有您的第一個執行階段元件庫。這個 SWF 檔可以載入至另一個 AVM2 SWF 檔，而且新的 SWF 檔就可以使用 AnimatingBox 和 AnimatingStar 元件。

將元件庫載入至另一個 SWF 檔

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

最後要處理的函數部分是資源檢視器的使用者介面。在此範例中，執行階段元件庫的路徑是以硬式編碼的方式編碼為 ASSETS_PATH 變數。或者，您也可以使用 FileReference 類別。例如，建立介面來瀏覽硬碟中特定的 SWF 檔。

成功載入執行階段元件庫後，Flash Player 就會呼叫 runtimeAssetsLoadComplete() 方法：

```
private function runtimeAssetsLoadComplete(event:Event):void
{
    var rl:* = event.target.content;
    var assetList:Array = rl.getAssets();
    populateDropdown(assetList);
    stage.frameRate = 60;
}
```

在此方法中，變數 rl 代表被載入端 SWF 檔。程式碼會呼叫被載入端 SWF 檔的 getAssets() 方法，以取得可用的資源清單，並呼叫 populateDropDown() 方法，將清單中的可用資源填入 ComboBox 組件。該方法接著會儲存每個資源的完整類別路徑。按一下使用者介面上的 「增加」按鈕會觸發 addAsset() 方法：

```
private function addAsset():void
{
    var className:String = assetNameCbo.selectedItem.data;
    var AssetClass:Class = getDefinitionByName(className) as Class;
    var mc:MovieClip = new AssetClass();
    ...
}
```

這會取得 ComboBox (assetNameCbo.selectedItem.data) 中目前選取之資源的類別路徑，並且為了建立該資源的新實體，還會使用 getDefinitionByName() 函數 (來自於 flash.utils 套件) 取得該資源之類別的實際參考。

第 17 章 使用移動補間動畫

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本，需要 **Flash CS3** 或更新的版本

第 162 頁 「將物件製成動畫」會說明如何在 ActionScript 中實作以指令碼編寫的動畫。

這個主題將說明如何使用不同的技巧來建立動畫：移動補間動畫。這項技巧可讓您利用 Adobe® Flash® Professional，在文件中藉由互動方式設定動作來產生移動的效果。接著，您就可以在執行階段，於動態 ActionScript 動畫中使用該動作。

Flash Professional 會自動產生 ActionScript，它可以實作移動補間動畫，讓您可以進行複製並重複使用。

若要建立移動補間動畫，您必須取得 Flash Professional 的授權。

更多說明主題

fl.motion 套件

移動補間動畫的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本，需要 **Flash CS3** 或更新的版本

移動補間動畫可讓您以簡單的方式建立動畫。

移動補間動畫會針對每個影格逐一修改顯示物件的屬性 (例如位置或旋轉)。藉由對顯示物件套用各種濾鏡和其它屬性，移動補間動畫也可以在顯示物件移動期間變更物件的外觀。您只需使用 Flash Professional 以互動方式建立移動補間動畫即可，Flash Professional 會自動為移動補間動畫建立 ActionScript。首先，請在 Flash 中，使用 「將移動複製為 ActionScript 3.0」命令，複製建立移動補間動畫的 ActionScript。接著，您就可以在執行階段，於自己的動態動畫中重複使用此 ActionScript 來建立移動的效果。

如需有關建立移動補間動畫的詳細資訊，請參閱 「使用 Flash Professional」中的 「移動補間動畫」一節。

重要概念與術語

下列是與此功能相關的重要術語：

移動補間動畫 一種建構，可以產生顯示物件的中間影格 (在不同時間會有不同狀態)，使得從第一個狀態進入第二個狀態期間呈現流暢的感覺。使用移動補間動畫，即可移動顯示物件使其橫越舞台，也可以使其隨著時間經過而放大、縮小、旋轉、淡化或變色。

在 Flash 中複製移動補間動畫指令碼

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本，需要 **Flash CS3** 或更新的版本

補間動畫可產生中間影格，以呈現顯示物件在時間軸上兩個不同影格中不同的狀態。如此一來，即可讓人感覺第一個影格中的影像流暢地變成第二個影格中的影像。在移動補間動畫中，外觀的變更通常涉及顯示物件位置的變更，因而產生移動的效果。除了變更顯物件的位置以外，移動補間動畫還能對顯示物件執行旋轉、傾斜、調整大小或套用濾鏡。

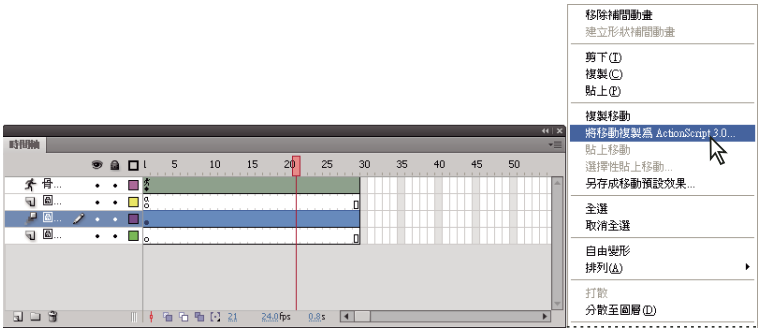
首先，請在 Flash 中，將顯示物件沿著時間軸在不同關鍵影格之間移動，建立移動補間動畫。接著，Flash 便會自動產生描述該補間動畫的 ActionScript 程式碼，如此一來，您便能夠複製程式碼並儲存在檔案中。如需有關建立移動補間動畫的詳細資訊，請參閱 「使用 Flash Professional」中的 「移動補間動畫」一節。

在 Flash 中，您可以利用兩種方式存取「將移動複製為 ActionScript 3.0」命令。第一種方式是透過舞台上的補間動畫快顯選單來存取：

1 選取舞台上的移動補間動畫。

2 按一下滑鼠右鍵 (Windows) 或按 Control+ 按一下 (Macintosh)。

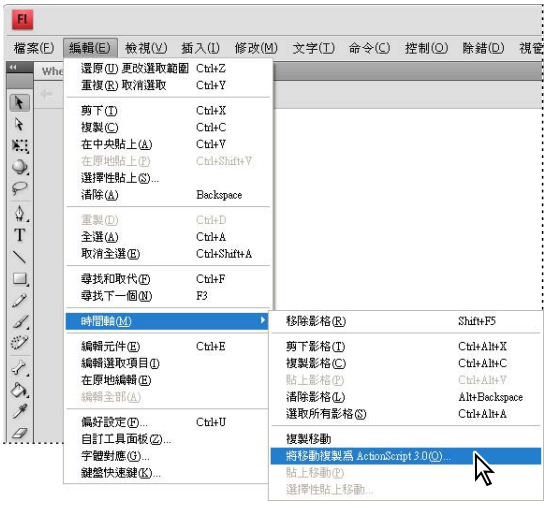
3 選擇「將移動複製為 ActionScript 3.0」。



第二種方式則是從 Flash 的「編輯」選單直接選擇該命令：

1 選取舞台上的移動補間動畫。

2 選取「編輯 > 時間軸 > 將移動複製為 ActionScript 3.0」。



複製指令碼之後，請將它貼在檔案中並加以儲存。

建立移動補間動畫以及複製並儲存指令碼之後，您即可在自己的動態 ActionScript 動畫中原封不動地重複使用，也可以進行修改。

整合移動補間動畫指令碼

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本，需要 **Flash CS3** 或更新的版本

從 Flash 複製的 ActionScript 程式碼檔頭會列出支援移動補間動畫所需的全部模組。

移動補間動畫類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本，需要 **Flash CS3** 或更新的版本

重要的移動補間動畫類別有 AnimatorFactory、MotionBase 和 Motion 類別，您可以從 fl.motion 套件取得這些類別。視移動補間動畫所操作的屬性而定，您可能還需要其它類別。例如，若要移動補間動畫對顯示物件執行變形或旋轉，請匯入適當的 flash.geom 類別。若要讓它套用濾鏡，請匯入 flash.filter 類別。在 ActionScript 中，移動補間動畫是 Motion 類別的實體。Motion 類別會儲存可套用至視覺物件的關鍵影格連續動畫。動畫資料包括位置、縮放、旋轉、傾斜、顏色、濾鏡以及加 / 減速。

下列 ActionScript 是從移動補間動畫複製而來，此移動補間動畫是在 Flash 中建立，以便將實體名稱為 Symbol1_2 的顯示物件製成動畫。在程式碼中，會為名為 __motion_Symbol1_2 的 MotionBase 物件宣告變數。MotionBase 是 Motion 類別的父類別。

```
var __motion_Symbol1_2:MotionBase;
```

接著，Script 會建立 Motion 物件：

```
__motion_Symbol1_2 = new Motion();
```

Motion 物件名稱

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本，需要 **Flash CS3** 或更新的版本

在之前的案例中，Flash 會自動將 Motion 物件命名為 __motion_Symbol1_2。它會將 __motion_ 前置詞附加至顯示物件的名稱。因此，自動產生的名稱便會以 Flash 中移動補間動畫之目標物件的實體名稱做為依據。Motion 物件的 duration 屬性表示移動補間動畫中的影格總數：

```
__motion_Symbol1_2.duration = 200;
```

對於您要複製其補間動畫的顯示物件實體，如果實體名稱不存在，Flash 預設會自動為該實體命名。

若您在自己的動畫中重複使用 Flash 建立的 ActionScript，可保持 Flash 自動產生的名稱，或替代為不同名稱。如果您變更補間動畫的名稱，請務必在整個指令碼中進行變更。

或者，您也可以在 Flash 中，將自己選擇的名稱指定給移動補間動畫的目標物件。然後，再建立移動補間動畫並複製指令碼。無論使用哪一種命名方式，請務必確認 ActionScript 程式碼中的 Motion 物件具有唯一的名稱。

說明動畫

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本，需要 **Flash CS3** 或更新的版本

MotionBase 類別的 addPropertyArray() 方法可用於加入值的陣列，以說明每個補間屬性。

對於移動補間動畫中的每個關鍵影格，這個陣列可能包含一個陣列項目。這些陣列的其中幾個陣列所包含的項目往往少於移動補間動畫中關鍵影格的總數。當陣列中的最後一個值不會針對剩餘的影格進行變更時，就會發生這種情形。

如果陣列引數的長度大於 Motion 物件的 duration 屬性時，addPropertyArray() 就會根據陣列引數的長度調整 duration 屬性的值，而不會針對之前已加入的屬性增加關鍵影格。對於多出來的動畫影格，新加入的關鍵影格則會繼續存在。

執行動畫時，Motion 物件的 x 和 y 屬性會描述補間動畫物件所變更的位置。如果在每個關鍵影格中，顯示物件的位置都有變更，那麼這些座標就是每個關鍵影格中都會變更的值。您可以使用 addPropertyArray() 方法，加入其它移動屬性。例如，如果要重新調整補間動畫物件的大小，請加入 scaleX 和 scaleY 屬性並設定其值；如果要傾斜補間動畫物件，請加入 skewX 和 skewY 屬性並設定其值；如果要旋轉補間動畫物件，請加入 rotationConcat 屬性並設定其值。

您可以使用 addPropertyArray() 方法，定義下列補間動畫屬性：

x	在父輩座標空間中，物件變形點的水平位置
y	在父輩座標空間中，物件變形點的垂直位置
z	在父輩座標空間中，物件變形點的深度 (z 軸) 位置
scaleX	從變形點套用的物件水平縮放 (百分比)
scaleY	從變形點套用的物件垂直縮放 (百分比)
skewX	從變形點套用的物件水平傾斜角度，以度數為單位
skewY	從變形點套用的物件垂直傾斜角度，以度數為單位
rotationX	物件從其原始方向沿著 x 軸的旋轉
rotationY	物件從其原始方向沿著 y 軸的旋轉
rotationConcat	從變形點套用時，物件相對於之前方向移動的旋轉 (z 軸) 值
useRotationConcat	如果設定這個屬性，當 addPropertyArray() 提供移動所需的資料時，目標物件便會旋轉
blendMode	BlendMode 類別值，指定物件與其下方圖形的顏色混合方式
matrix3D	如果關鍵影格具有 matrix3D 屬性，則這個屬性會用於 3D 補間動畫。使用時，會忽略之前所有變形屬性
rotationZ	物件從其原始方向相對於 3D 父輩容器的 z 軸旋轉 (以度數為單位)。這個屬性是用於 3D 補間動畫而非 rotationConcat

視您在 Flash 中指定給移動補間動畫哪些屬性，加入至自動產生之指令碼的屬性將有所不同。自訂專屬的指令碼時，您可以新增、移除或修改上述某些屬性。

下列程式碼會將幾個值指定給名為 __motion_Wheel 之移動補間動畫的屬性。在這個範例中，補間動畫顯示物件的位置不會改變，但會在補間動畫的整個 29 個影格中原地打轉。指定給 rotationConcat 陣列的多個值會定義旋轉的方式，這個移動補間動畫的其它屬性值則不會改變。

```
__motion_Wheel = new Motion();
__motion_Wheel.duration = 29;
__motion_Wheel.addPropertyArray("x", [0]);
__motion_Wheel.addPropertyArray("y", [0]);
__motion_Wheel.addPropertyArray("scaleX", [1.00]);
__motion_Wheel.addPropertyArray("scaleY", [1.00]);
__motion_Wheel.addPropertyArray("skewX", [0]);
__motion_Wheel.addPropertyArray("skewY", [0]);
__motion_Wheel.addPropertyArray("rotationConcat",
    [
        0,-13.2143,-26.4285,-39.6428,-52.8571,-66.0714,-79.2857,-92.4999,-105.714,
        -118.929,-132.143,-145.357,-158.571,-171.786,-185,-198.214,-211.429,-224.643,
        -237.857,-251.071,-264.286,-277.5,-290.714,-303.929,-317.143,-330.357,
        -343.571,-356.786,-370
    ]
);
__motion_Wheel.addPropertyArray("blendMode", ["normal"]);
```

在下個範例中，名為 Leaf_1 的顯示物件會移動橫越舞台。針對動畫所包含之 100 個影格的其中每一個影格，這個顯示物件的 x 和 y 屬性陣列都包含不同的值。此外，當物件移動橫越舞台時，會在其 z 軸上旋轉。rotationZ 屬性陣列中的多個項目會決定轉動的方式。

```
__motion_Leaf_1 = new MotionBase();
__motion_Leaf_1.duration = 100;
__motion_Symbol1_4.addPropertyArray("y",
    [
        0,5.91999,11.84,17.76,23.68,29.6,35.52,41.44,47.36,53.28,59.2,65.12,71.04,
        76.96,82.88,88.8,94.72,100.64,106.56,112.48,118.4,124.32,130.24,136.16,142.08,
        148,150.455,152.909,155.364,157.818,160.273,162.727,165.182,167.636,170.091,
        172.545,175,177.455,179.909,182.364,184.818,187.273,189.727,192.182,194.636,
        197.091,199.545,202,207.433,212.865,218.298,223.73,229.163,234.596,240.028,
        245.461,250.893,256.326,261.759,267.191,272.624,278.057,283.489,
        288.922,294.354,299.787,305.22,310.652,316.085,321.517,326.95,330.475,334,
        337.525,341.05,344.575,348.1,351.625,355.15,358.675,362.2,365.725,369.25,
        372.775,376.3,379.825,383.35,386.875,390.4,393.925,397.45,400.975,404.5,
        407.5,410.5,413.5,416.5,419.5,422.5,425.5
    ]
);
__motion_Symbol1_4.addPropertyArray("scaleX", [1.00]);
__motion_Symbol1_4.addPropertyArray("scaleY", [1.00]);
__motion_Symbol1_4.addPropertyArray("skewX", [0]);
__motion_Symbol1_4.addPropertyArray("skewY", [0]);
__motion_Symbol1_4.addPropertyArray("z",
    [
        0,0,0,
```

初始化濾鏡陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本，需要 **Flash CS3** 或更新的版本

initFilters() 方法可用於初始化濾鏡。這個方法的第一個引數是一個陣列，其中包含套用至顯示物件之所有濾鏡的完整類別名稱。這個濾鏡名稱陣列是從 Flash 中移動補間動畫的濾鏡清單產生。在指令碼副本中，您可以對這個陣列新增或移除 flash.filters 套件中的任何濾鏡。下列呼叫會初始化目標顯示物件的濾鏡清單。在呼叫中，會將 DropShadowFilter、GlowFilter、BevelFilter 和清單副本套用至 Motion 物件內的每個關鍵影格。

```
__motion_Box.initFilters(["flash.filters.DropShadowFilter", "flash.filters.GlowFilter",
"flash.filters.BevelFilter"], [0, 0, 0]);
```

新增濾鏡

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本，需要 **Flash CS3** 或更新的版本

addFilterPropertyArray() 方法會搭配下列引數，描述已初始化之濾鏡的屬性：

1. 第一個引數會按照索引來識別濾鏡。這個索引就是之前呼叫 initFilters() 時，所傳遞之濾鏡類別名稱陣列內濾鏡名稱的位置。
2. 第二個引數是每個關鍵影格中，用來儲存濾鏡的濾鏡屬性。
3. 第三個引數是指定之濾鏡屬性的值。

在已經呼叫了 initFilters() 的情況下，使用下列程式碼呼叫 addFilterPropertyArray() 時，會將 5 這個值指定給 DropShadowFilter 的 blurX 和 blurY 屬性。DropShadowFilter 是已初始化之濾鏡陣列中的第一個 (索引 0) 項目：

```
__motion_Box.addFilterPropertyArray(0, "blurX", [5]);
__motion_Box.addFilterPropertyArray(0, "blurY", [5]);
```

下列三個呼叫會分別將值指定給 GlowFilter (已初始化之濾鏡陣列中的第二個 (索引 1) 項目) 的 quality、alpha 和 color 屬性：

```
__motion_Box.addFilterPropertyArray(1, "quality", [BitmapFilterQuality.LOW]);
__motion_Box.addFilterPropertyArray(1, "alpha", [1.00]);
__motion_Box.addFilterPropertyArray(1, "color", [0xff0000]);
```

下列四個呼叫會分別將值指定給 BevelFilter (已初始化之濾鏡陣列中的第三個 (索引 2) 項目) 的 shadowAlpha、shadowColor、highlightAlpha 和 highlightColor：

```
__motion_Box.addFilterPropertyArray(2, "shadowAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "shadowColor", [0x000000]);
__motion_Box.addFilterPropertyArray(2, "highlightAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "highlightColor", [0xffffff]);
```

使用 ColorMatrixFilter 調整顏色

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本，需要 **Flash CS3** 或更新的版本

當您初始化 ColorMatrixFilter 之後，即可設定適當的 AdjustColor 屬性，以調整補間動畫顯示物件的亮度、對比、飽和度和色相。一般而言，在 Flash 中建立移動補間動畫時，會套用 AdjustColor 濾鏡；您可在自己的 ActionScript 中進行微調。在下列範例中，當顯示物件移動時，色相和飽和度即會改變。

```
__motion_Leaf_1.initFilters(["flash.filters.ColorMatrix"], [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorBrightness", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorContrast", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorSaturation",
    [
        0,-0.589039,1.17808,-1.76712,-2.35616,-2.9452,-3.53424,-4.12328,
        -4.71232,-5.30136,-5.89041, 6.47945,-7.06849,-7.65753,-8.24657,
        -8.83561,-9.42465,-10.0137,-10.6027,-11.1918,11.7808,-12.3699,
        -12.9589,-13.5479,-14.137,-14.726,-15.3151,-15.9041,-16.4931,
        17.0822,-17.6712,-18.2603,-18.8493,-19.4383,-20.0274,-20.6164,
        -21.2055,-21.7945,22.3836,-22.9726,-23.5616,-24.1507,-24.7397,
        -25.3288,-25.9178,-26.5068,-27.0959,27.6849,-28.274,-28.863,-29.452,
        -30.0411,-30.6301,-31.2192,-31.8082,-32.3973,32.9863,-33.5753,
        -34.1644,-34.7534,-35.3425,-35.9315,-36.5205,-37.1096,-37.6986,
        38.2877,-38.8767,-39.4657,-40.0548,-40.6438,-41.2329,-41.8219,
        -42.411,-43
    ],
    -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorHue",
    [
        0,0.677418,1.35484,2.03226,2.70967,3.38709,4.06451,4.74193,5.41935,
        6.09677,6.77419,7.45161,8.12903,8.80645,9.48387,10.1613,10.8387,11.5161,
        12.1935,12.871,13.5484,14.2258,14.9032,15.5806,16.2581,16.9355,17.6129,
        18.2903,18.9677,19.6452,20.3226,21,22.4286,23.8571,25.2857,26.7143,28.1429,
        29.5714,31,32.4286,33.8571,35.2857,36.7143,38.1429,39.5714,41,42.4286,43.8571,
        45.2857,46.7143,48.1429,49.5714,51,54,57,60,63,66,69,72,75,78,81,84,87,
        90,93,96,99,102,105,108,111,114
    ],
    -1, -1);
```

將移動補間動畫與其顯示物件建立關聯

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本，需要 **Flash CS3** 或更新的版本

最後一項工作，便是將移動補間動畫與其操作的一個或多個顯示物件建立關聯。

AnimatorFactory 類別可管理移動補間動畫與其目標顯示物件之間的關聯。AnimatorFactory 建構函式的引數是 Motion 物件：

```
var __animFactory_Wheel:AnimatorFactory = new AnimatorFactory(__motion_Wheel);
```

使用 AnimatorFactory 類別的 addTarget() 方法，即可將目標顯示物件與其移動補間動畫建立關聯。從 Flash 複製而來的 ActionScript 會將 addTarget() 這一行標示為註解，並且不會指定實體名稱：

```
// __animFactory_Wheel.addTarget(<instance name goes here>, 0);
```

在副本中，指定要與移動補間動畫建立關聯的顯示物件。在下列範例中，會將目標指定為 greenWheel 和 redWheel：

```
__animFactory_Wheel.AnimatorFactory.addTarget(greenWheel, 0);
__animFactory_Wheel.AnimationFactory.addTarget(redWheel, 0);
```

您可以多次呼叫 addTarget()，將多個顯示物件與同一個移動補間動畫建立關聯。

第 18 章 使用反向運動

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本，需要 **Flash CS4** 或更新的版本

如果要創造自然的動作，反向運動 (IK) 是很有用的技巧。

IK 可讓您在名為 IK 骨架的一連串連接部分中建立座標式移動，讓各部分能以真實的方式一起移動。骨架的各個部分是由骨塊與關節所組成。指定骨架的端點時，IK 便會計算到達端點所需的關節角度。

若要手動計算這些角度，可能會花費您許多時間。這項功能的好處是，您可以使用 Adobe® Flash® Professional 以互動方式建立骨架。然後，再使用 ActionScript 將骨架製成動畫。隨附於 Flash Professional 的 IK 引擎能執行計算，以描述骨架的移動方式。您可以在 ActionScript 程式碼中透過特定參數，限制移動的方式。

IK 的 Flash Professional CS5 版本新概念是骨塊彈起，通常用於高品質的動畫應用程式。將此功能與新的動態 Physics 引擎搭配使用，可讓您設定逼真的移動方式。而且，可在執行階段和編寫期間看到此效果。

若要建立反向運動骨架，您必須取得 Flash Professional 的授權。

更多說明主題
fl.ik 套件

反向運動的基本概念

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本，需要 **Flash CS4** 或更新的版本

反向運動 (IK) 可讓您藉由連結各個部分，使其以真實的方式互相進行相對運動，以建立逼真的動畫。

例如，使用 IK 時，若要將一條腿移至特定位置，只要將到達該位置所需的腿關節移動加以連接即可。IK 會使用 IK 骨架結構中鍊結在一起的骨塊架構。fl.ik 套件可以協助您建立類似自然動作的動畫。它可以讓您將多個 IK 骨架順暢地製成動畫，就算不甚瞭解 IK 演算法背後的物理觀念也沒關係。

首先，使用 Flash Professional 建立 IK 骨架與其附加的骨塊和關節，然後，再藉由存取 IK 類別，在執行階段將它們製成動畫。

如需有關如何建立 IK 骨架的詳細指示，請參閱 「使用 Flash Professional」中的 「使用反向運動」一節。

重要概念與術語
下列參考清單包含了與此功能相關的重要術語：

骨架 由骨塊和關節組成的運動鏈 (在電腦動畫中用來模擬自然的動作)

骨塊 就像動物骨骼中的骨頭一樣，是骨架中固定的部分。

反向運動 (IK) 彈性連接之物件 (稱為運動鏈或骨架) 的參數決定程序。

關節 就像動物的關節一樣，位於兩個骨塊連接的位置，連接之後，骨塊便可以移動。

Physics 引擎 一整套的物理相關演算法，用於提供動畫類生物的動作。

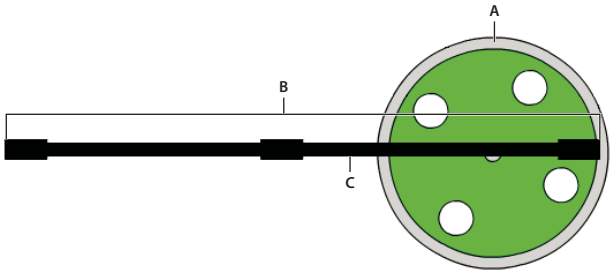
彈起 骨塊移動時、父骨塊移動後，骨塊的反應以及一段時間後逐漸消退的特性。

將 IK 骨架製成動畫概觀

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本，需要 **Flash CS4** 或更新的版本

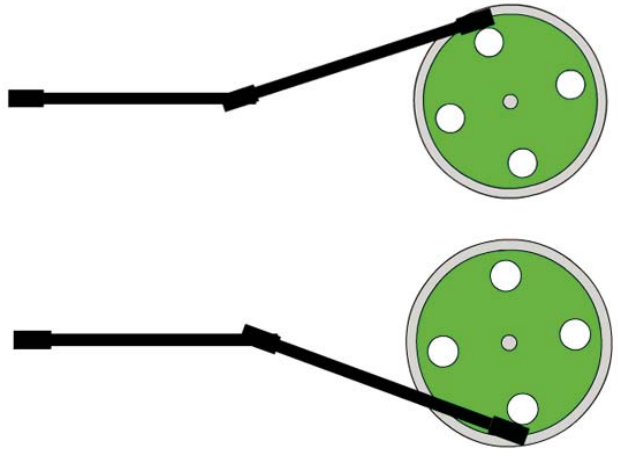
在 Flash Professional 建立 IK 骨架之後，請使用 fl.ik 類別限制其移動方式、追蹤其事件，以及在執行階段將其製成動畫。

下圖顯示名為 Wheel 的影片片段。其中轉軸是名為 Axle 之 IKArmature 的實體。IKMover 類別會隨著滾輪旋轉同步移動骨架。在骨架中，IKBone (ikBone2) 的尾端關節會附加至滾輪。



A. 滾輪 **B.** 轉軸 **C.** ikBone2

在執行階段，滾輪會根據第 285 頁 「說明動畫」所述的 __motion_Wheel 移動補間動畫來旋轉。IKMover 物件會初始化轉軸並控制轉軸的移動方式。下圖顯示兩張附加至旋轉中滾輪在不同影格時的轉軸骨架快照。



在執行階段，下列 ActionScript 會執行以下動作：

- 取得骨架及其組件的相關資訊
- 實體化 IKMover 物件
- 隨著滾輪旋轉一併移動轉軸

```
import fl.ik.*

var tree:IKArmature = IKManager.getArmatureByName("Axle");
var bone:IKBone = tree.getBoneByName("ikBone2");
var endEffector:IKJoint = bone.tailJoint;
var pos:Point = endEffector.position;

var ik:IKMover = new IKMover(endEffector, pos);
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;

Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90, 0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

用來移動轉軸的 IK 類別如下：

- IKArmature：描述骨架 (由骨塊和關節組成的樹狀結構)，而骨架必須使用 Flash Professional 建立
- IKManager：文件中 IK 骨架的所有容器類別，而骨架必須使用 Flash Professional 建立
- IKBone：IK 骨架的組成部分
- IKJoint：兩個 IK 骨塊之間的連接點
- IKMover：初始化 IK 骨架和控制其移動方式

如需這些類別的完整詳細說明，請參閱 ik 套件。

取得 IK 骨架的相關資訊

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本，需要 **Flash CS4** 或更新的版本

首先，針對組成要移動之部位的骨架、骨塊和關節宣告變數

下列程式碼會使用 IKManager 類別的 getArmatureByName() 方法，將 Axle 骨架的值指定給 IKArmature 變數 tree。其中，Axle 骨架是之前利用 Flash Professional 所建立。

```
var tree:IKArmature = IKManager.getArmatureByName("Axle");
```

同樣地，下列程式碼會使用 IKArmature 類別的 getBoneByName() 方法，將 ikBone2 骨塊的值指定給 IKBone 變數。

```
var bone:IKBone = tree.getBoneByName("ikBone2");
```

ikBone2 骨塊尾端的關節是骨架的一部分，會附加至旋轉中的滾輪。

下列程式碼會宣告 endEffector 變數，並將它指定給 ikBone2 骨塊的 tailjoint 屬性：

```
var endEffector:IKJoint = home.tailjoint;
```

pos 變數是一個點，其中儲存了 endEffector 關節的目前位置。

```
var pos:Point = endEffector.position;
```

在這個範例中，pos 是轉軸尾端與滾輪連接的關節位置。這個變數的原始值是從 IKJoint 的 position 屬性取得。

初始化 IKMover 和限制其移動方式

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本，需要 **Flash CS4** 或更新的版本

IKMover 類別的實體會移動轉軸。

下列程式碼會初始化 IKMover 物件 ik，並將要移動的元素以及移動的起點傳遞至其建構函式：

```
var ik:IKMover = new IKMover(endEffector, pos);
```

IKMover 類別的各項屬性可以讓您限制骨架的移動方式。您可以根據移動的距離、循環和時間來限制移動方式。

下列屬性配對可以強制套用這些限制。這些配對是由 Boolean 值和整數組成 (分別代表示移動是否受限以及指定限制內容)：

Boolean 屬性	整數屬性	限制設定
limitByDistance:Boolean	distanceLimit:int	設定 IK 引擎每次循環移動的距離上限 (以像素為單位)。
limitByIteration:Boolean	iterationLimit:int	設定 IK 引擎對每次移動所執行的循環次數上限。
limitByTime:Boolean	timeLimit:int	設定要分派給 IK 引擎執行移動的時間上限 (以毫秒為單位)。

根據預設，所有 Boolean 值都會設定為 false，因此除非您明確將 Boolean 值設定為 true，否則移動方式並不會受到限制。將適當的屬性設定為 true 即可強制套用限制。如果您設定了某項限制內容的值，但是卻沒有設定相對應的 Boolean 屬性，該項限制便會遭忽略。在這種情況下，IK 引擎會持續移動物件，直到到達其它限制或 IKMover 的目標位置為止。

在下列範例中，會將每次循環後骨架移動的距離上限設定為 0.1，並將每次移動的循環次數上限設定為十次。

```
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;
```

移動 IK 骨架

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本，需要 **Flash CS4** 或更新的版本

IKMover 會在滾輪的事件偵聽程式內移動轉軸。針對滾輪的每個 enterFrame 事件，骨架的新目標位置都會經過計算。利用骨架的 moveTo() 方法，IKMover 即可將尾端關節移至其目標位置或其 limitByDistance、limitByIteration 和 limitByTime 屬性所設定的上限。

```
Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90,0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

使用彈起

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本，需要 **Flash CS5** 或更新的版本

Flash Professional CS5 的反向運動支援骨塊彈起。骨塊彈起可以在編寫期間設定，而且骨塊彈起特質可以在執行階段新增或修改。彈起是骨塊及其關節的一種屬性。它有兩種特質：IKJoint.springStrength 會設定彈起的幅度，而 IKJoint.springDamping 會增加強度值的阻力以及變更彈起的衰退速率。

彈起強度是百分比值，從預設值 0 (完全堅硬) 到 100 (十分鬆散，由物理性質控制)。骨塊加上彈起會反應其關節的移動。若未啟用其他轉移 (旋轉、x 或 y)，則彈起設定將沒有作用。

彈起阻尼值是百分比值，從預設值 0 (無阻力) 到 100 (高阻尼)。阻尼會變更骨塊的初始移動及其回到休息位置之間的時間長度。

您可以按一下 IKArmature 物件的 IKArmature.springsEnabled 屬性，便可以檢查和瞭解是否為它啟用了彈起功能。其他彈起屬性和方法屬於個別的 IKJoint 物件。可以啟用關節進行角度旋轉和沿著 x 與 y 軸轉移。您可以使用 IKJoint.setSpringAngle 來定位旋轉關節的彈起角度，並使用 IKJoint.setSpringPt 來定位轉移關節的彈起位置。

此範例會按名稱選取骨塊，並識別其 tailJoint。程式碼會測試父骨架，瞭解是否已經啟用彈起，然後為關節設定彈起屬性。

```
var arm:IKArmature = IKManager.getArmatureAt(0);
var bone:IKBone = arm.getBoneByName("c");
var joint:IKJoint = bone.tailJoint;
if (arm.springsEnabled) {
    joint.springStrength = 50; //medium spring strength
    joint.springDamping = 10; //light damping resistance
    if (joint.hasSpringAngle) {
        joint.setSpringAngle(30); //set angle for rotational spring
    }
}
```

使用 IK 事件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本，需要 **Flash CS4** 或更新的版本

IKEvent 類別可讓您建立包含 「IK 事件」相關資訊的事件物件。IKEvent 資訊會描述因為超過指定的時間、距離或循環限制而已經終止的運動。

下列程式碼會顯示用於追蹤時間限制事件的事件偵聽程式和處理常式。這個事件處理常式會回報當超過 IKMover 的時間限制時，所引發之事件的時間、距離、循環次數和關節屬性。

```
var ikmover:IKMover = new IKMover(endjoint, pos);
ikMover.limitByTime = true;
ikMover.timeLimit = 1000;

ikmover.addEventListener(IKEvent.TIME_LIMIT, timeLimitFunction);

function timeLimitFunction(evt:IKEvent):void
{
    trace("timeLimit hit");
    trace("time is " + evt.time);
    trace("distance is " + evt.distance);
    trace("iterationCount is " + evt.iterationCount);
    trace("IKJoint is " + evt.joint.name);
}
```

第 19 章 在三維環境 (3D) 中工作

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

Flash Player 和 AIR 執行階段透過兩種方法支援 3D 圖形。您可以使用 Flash 顯示清單上的三維顯示物件。這個方法適用於在 Flash 內容中加入 3D 特效，同時適用於低多邊形數的物件。在 Flash Player 11 和 AIR 3 或更新的版本中，您可以使用 Stage3D API 來顯示複雜的 3D 場景。

Stage3D 檢視區域不是顯示物件。相反地，3D 圖形會顯示至 Flash 顯示清單底下 (以及任何 StageVideo 檢視區域平面上方) 所顯示的檢視區域。與其使用 Flash DisplayObject 類別建立場景，請使用可由程式控制的 3D 管道 (類似 OpenGL 和 Direct3D)。這個管道接受三角形資料和紋理做為輸入，並且使用您提供的著色器程式來顯示場景。當用戶端電腦上有相容的圖像處理單元 (GPU) 以及支援的驅動程式可以使用時，就會用到硬體加速。

Stage3D 提供極低階的 API。在應用程式中，建議您使用支援 Stage3D 的 3D 架構。您可以建立自己的架構，也可以使用其中一個商業廣告，開啟已可供使用的來源架構。

如需有關使用 Stage3D 開發 3D 應用程式以及可用 3D 架構的詳細資訊，請造訪 Flash Player 開發人員中心：Stage 3D。

3D 顯示物件的基本概念

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

對於投影在二維畫面上的二維 (2D) 物件與三維 (3D) 物件而言，兩者之間的主要差異在於物件多了一個維度。第三個維度可讓物件移近或遠離使用者的視點。

當您明確將顯示物件的 z 屬性設定為數值時，物件便會自動建立 3D 變形矩陣。您可以變更這個矩陣，以修改物件的 3D 變形設定。

此外，3D 旋轉也與 2D 旋轉不同。在 2D 中，旋轉軸一律與 x/y 平面垂直，換句話說，旋轉軸是在 z 軸上；而在 3D 中，旋轉軸則可以在 x、y 或 z 軸周圍。只要設定顯示物件的旋轉和縮放屬性，即可讓物件在 3D 空間中移動。

重要概念與術語

下列參考清單包含程式設計三維圖形時會碰到的重要術語：

透視 在 2D 平面中，平行線往遠處延伸的兩端看起來會在消失點處交會，如此可讓人產生深度和距離的錯覺。

投影 具有兩個維度以上的物件在 2D 影像上的呈現。例如，3D 投影會將 3D 座標點對應在 2D 平面上。

旋轉 藉由繞圈方式移動物件包含的每一點，以變更物件的方向 (通常也會變更位置)。

變形 藉由轉移、旋轉、縮放、傾斜或其中各種動作組合，改變 3D 座標點或數組座標點。

轉移 藉由將物件包含的每一點往相同的位置等量移動，變更物件的位置。

消失點 以線性透視表示時，向後延伸的平行線看似交會的點。

Vector 3D 向量會使用迪卡兒座標 x、y 和 z 來表示三維空間中的某個點或位置。

頂點 轉折點

具有紋理的網格 定義 3D 空間中物件的任一點。

UV 對應 對 3D 表面套用紋理或點陣圖的方式。UV 對應會將各個值指定給影像的座標，做為水平 (U) 軸和垂直 (V) 軸的百分比

T 值 當 3D 物件移近或遠離目前視點時，用於決定 3D 物件大小縮放係數。

剔除 顯示 (或隱藏) 具有特定迂迴的表面。使用剔除，即可隱藏目前視點看不見的表面。

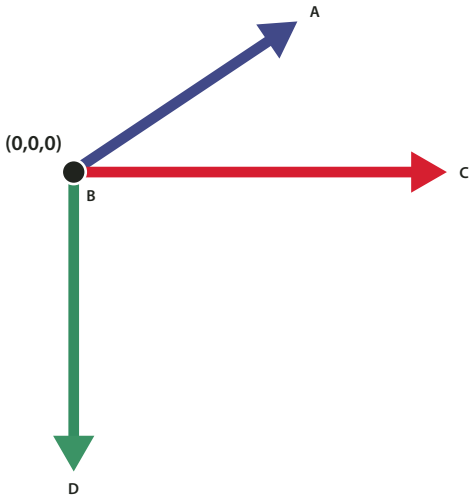
瞭解 Flash Player 和 AIR 執行階段中的 3D 顯示物件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

在 Flash Player 10 之前的 Flash Player 版本以及 Adobe AIR 1.5 之前的 Adobe AIR 版本中，顯示物件具有 x 和 y 兩個屬性，這兩個屬性會決定物件在 2D 平面上的位置。從 Flash Player 10 和 Adobe AIR 1.5 開始，每個 ActionScript 顯示物件都有 z 屬性，可讓您決定物件在 z 軸 (通常可用來表示深度或距離) 上的位置。

Flash Player 10 和 Adobe AIR 1.5 新增了 3D 效果支援。不過，顯示物件本質上都是平面的。每個顯示物件 (例如 MovieClip 或 Sprite 物件) 最終都會在單一平面上以二維方式顯示。3D 功能可讓您對這些平面物件，以三維方式進行放置、移動、旋轉和其它變形動作。此外，也能讓您管理 3D 座標點，並將其轉換成 2D 的 x、y 座標，如此即可將 3D 物件投影在 2D 檢視上。您可以使用這些功能模擬多種 3D 體驗。

ActionScript 使用的 3D 座標系統不同於其它系統。在 ActionScript 中使用 2D 座標時，x 值會隨著您沿著 x 軸向右移動而增加，而 y 值會隨著您沿著 y 軸向下移動而增加。3D 座標系統保留了這些慣例，並且新增了 z 軸，其值會隨著您遠離視點而增加。



ActionScript 3D 座標系統中 x、y 和 z 軸的正數方向。
A. +Z 軸 **B.** 原點 **C.** +X 軸 **D.** +Y 軸

備註：請注意，Flash Player 和 AIR 一律會以圖層來表示 3D。這表示，如果在顯示清單中，物件 A 位於物件 B 之前，那麼無論這兩個物件的 z 軸值為何，Flash Player 或 AIR 一律會將 A 顯示在 B 前面。若要解決顯示清單順序和 z 軸順序之間的這種衝突，請使用 transform.getRelativeMatrix3D() 方法儲存 3D 顯示物件的圖層後，再重新排列其順序。如需詳細資訊，請參閱 第 306 頁「使用 Matrix3D 物件以重新排列顯示物件」。

下列 ActionScript 類別可支援新的 3D 相關功能：

1. flash.display.DisplayObject 類別包含 z 屬性和新的旋轉及縮放屬性，可讓您在 3D 空間中操作顯示物件。DisplayObject.local3DToGlobal() 方法可提供一種簡單的方式，在 2D 平面上投影 3D 幾何。
2. flash.geom.Vector3D 類別可以當做資料結構使用，讓您管理 3D 座標點。此外，這個類別也支援向量數學。
3. flash.geom.Matrix3D 類別支援複雜的 3D 幾何變形，例如旋轉、縮放和轉移。
4. flash.geom.PerspectiveProjection 可以控制在 2D 檢視上對應 3D 幾何時所搭配的參數。

您可以使用兩種不同方式，在 ActionScript 中模擬 3D 影像：

1 在 3D 空間中排列平面物件並製成動畫。這個方式包含使用顯示物件的 x、y 和 z 屬性，或使用 DisplayObject 類別設定旋轉和縮放屬性，將顯示物件製成動畫。若使用 DisplayObject.transform.matrix3D 物件，還可以產生更複雜的動作。DisplayObject.transform.perspectiveProjection 物件可讓您自訂以 3D 透視來繪製顯示物件的方式。對於主要以平面組成的 3D 物件，當您希望將該物件製成動畫時，請使用這個方式。這個方式的範例包含 3D 影像展示或在 3D 空間中排列的 2D 動畫物件。

2 透過 3D 幾何產生 2D 三角形，然後使用紋理顯示這些三角形。若要使用這個方式，首先您必須定義和管理 3D 物件的相關資料，然後再將該資料轉換成 2D 三角形以顯示在螢幕上。您可以將點陣圖紋理對應至這些三角形，然後再使用 Graphics.drawTriangles() 方法，將這些三角形繪製成為圖形物件。這個方式的範例包含從檔案載入 3D 模型資料，並於螢幕上顯示該模型，或者產生及繪製 3D 範圍做為三角形網格。

建立和移動 3D 顯示物件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

若要將 2D 顯示物件轉換成 3D 顯示物件，您可以明確將物件的 z 屬性設定為數值。當您為 z 屬性指定值時，便會為顯示物件建立新的 Transform 物件。設定 DisplayObject.rotationX 或 DisplayObject.rotationY 屬性也同樣能建立新的 Transform 物件。Transform 物件包含 Matrix3D 屬性，可控制顯示物件在 3D 空間中的呈現方式。

下列程式碼會為名為「leaf」的顯示物件設定座標：

```
leaf.x = 100; leaf.y = 50; leaf.z = -30;
```

您可以在「leaf」之 Transform 物件的 matrix3D 屬性中，看到這些值以及所衍生的屬性：

```
var leafMatrix:Matrix3D  = leaf.transform.matrix3D;

trace(leafMatrix.position.x);
trace(leafMatrix.position.y);
trace(leafMatrix.position.z);
trace(leafMatrix.position.length);
trace(leafMatrix.position.lengthSquared);
```

如需 Transform 物件的屬性相關資訊，請參閱 Transform 類別。如需 Matrix3D 物件的屬性相關資訊，請參閱 Matrix3D 類別。

在 3D 空間中移動物件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

您可以變更物件的 x、y 或 z 屬性值，在 3D 空間中移動物件。當您變更 z 屬性的值時，檢視者會覺得物件看起來靠近或遠離自己。

在下列程式碼中，會變更兩個橢圓的 z 屬性來回應事件，以便沿著其 z 軸前後移動。ellipse2 的移動速度比 ellipse1 快，因為其 z 屬性針對每個 Frame 事件是以 20 為倍數增加，而 ellipse1 的 z 屬性則是以 10 為倍數增加：

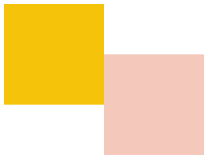
```
var depth:int = 1000;

function ellipse1FrameHandler(e:Event):void
{
    ellipse1Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 10;
}
function ellipse2FrameHandler(e:Event):void
{
    ellipse2Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 20;
}
function setDepth(e:Event, d:int):int
{
    if(e.currentTarget.z > depth)
    {
        e.currentTarget.z = depth;
        d = -1;
    }
    else if (e.currentTarget.z <  0)
    {
        e.currentTarget.z = 0;
        d = 1;
    }
}
```

在 3D 空間中旋轉物件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

您可以根據物件的下列旋轉屬性設定，以三種方式旋轉物件：rotationX、rotationY 和 rotationZ。

下圖顯示未旋轉的兩個正方形：

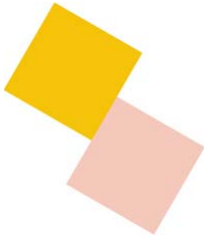


下圖顯示當您遞增正方形之容器的 rotationY 屬性，讓它們在 y 軸上旋轉時的模樣。旋轉兩個正方形的容器 (或父輩顯示物件) 時，兩個正方形本身也會旋轉：

```
container.rotationY += 10;
```

下圖顯示當您設定正方形之容器的 rotationX 屬性時，正方形的模樣。這麼做會讓它們在 x 軸上旋轉。

下圖顯示當您遞增正方形之容器的 rotationZ 屬性時，正方形的模樣。這麼做會讓它們在 z 軸上旋轉。



一個顯示物件可以同時在 3D 空間中移動和旋轉。

在 2D 檢視上投影 3D 物件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

flash.geom 套件中的 PerspectiveProjection 類別提供一種簡易方式，讓您在透過 3D 空間移動顯示物件時套用基本透視。

如果不明確為 3D 空間建立透視投影，3D 引擎便會使用預設的 PerspectiveProjection 物件 (位於根層級，並且會套用至其所有子系)。

定義 PerspectiveProjection 物件如何顯示 3D 空間的三個屬性如下：

- fieldOfView
- projectionCenter
- focalLength

修改 fieldOfView 的值時，會自動修改 focalLength 的值 (反之亦然)，因為兩者具有相互依存的關係。

用於計算 focalLength (已指定 fieldOfView 值) 的公式為：

```
focalLength = stageWidth/2 * (cos(fieldOfView/2) / sin(fieldOfView/2)
```

通常，您會明確修改 fieldOfView 屬性。

檢視的欄位

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

藉由操作 PerspectiveProjection 類別的 fieldOfView 屬性，您可以讓靠近檢視者的 3D 顯示物件看起來較大，並且讓遠離檢視者的物件看起來較小。

fieldOfView 屬性會指定介於 0 和 180 度之間的角度，以決定透視投影的強度。指定的值愈大，對於沿著其 z 軸移動的顯示物件而言，所套用的扭曲程度就愈大。指定的 fieldOfView 值較小時，物件的縮放程度也較小，看起來只像是在空間中向後移動一點點距離。指定的 fieldOfView 值較大時，物件的扭曲程度也較大，看起來就像是移動了很遠的距離。最大值 179.9999... 度會導致極大的魚眼攝像機鏡頭效果。fieldOfView 的最大值是 179.9999...，而最小值是 0.00001...。剛好為 0 和 180 都是不合法的值。

投影中心

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

projectionCenter 屬性表示透視投影中的消失點。這個屬性的作用會當做位移，套用至舞台左上角的註冊點 (0,0)。

當物件看起來離檢視者愈來愈遠時，它會向消失點傾斜，最終便會消失。請想像有個無限長的走廊。當您向走廊前方看去，牆壁的邊緣會在走廊遠端的消失點交會。

如果消失點位於舞台的中央，走廊的盡頭便會在中央的某一點消失不見。projectionCenter 屬性的預設值為舞台的中央。例如，假設您希望元素出現在舞台的左邊，3D 區域出現在舞台的右邊，請將 projectionCenter 設定為指向舞台的右邊，讓該處成為 3D 檢視區域的消失點。

焦距

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

focalLength 屬性表示視點的原點 (0,0,0) 和顯示物件在其 z 軸的位置之間的距離。

長焦距就像望遠鏡頭一樣，可視範圍較狹隘，且物件之間的距離都經過壓縮。短焦距就像廣角鏡頭一樣，可視範圍較寬廣，但是會產生大幅度的扭曲。而中等焦距所呈現的大概就是肉眼可以看見的模樣。

通常在透視變形期間，focalLength 會隨著顯示物件移動而經過動態重新計算，但是您也能明確加以設定。

預設的透視投影值

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

在根層級建立的預設 PerspectiveProjection 物件具有下列值：

- fieldOfView：55
- perspectiveCenter：stagewidth/2,stageHeight/2
- focalLength：stageWidth/ 2 * (cos(fieldOfView/2) / sin(fieldOfView/2))

如果您未建立專屬的 PerspectiveProjection 物件，上述即為預設使用的值。

您可以實體化已自行修改 projectionCenter 和 fieldOfView 的專屬 PerspectiveProjection 物件。在這種情況下，新建立物件的預設值 (以預設舞台大小 500 x 500 為基礎) 如下所示：

- fieldOfView：55
- perspectiveCenter：250,250
- focalLength：480.24554443359375

範例：透視投影

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

在下列範例中，會示範如何使用透視投影建立 3D 空間。範例中會說明如何透過 projectionCenter 屬性，修改消失點及變更空間的透視投影。這項變更會強迫 focalLength 和 fieldOfView 根據伴隨而來的 3D 空間扭曲執行重新計算。

在本範例中，會執行下列步驟：

1 建立名為 center 的 Sprite，當做具有十字準線的圓圈

2 將 center Sprite 的座標指定給 perspectiveProjection 屬性 (屬於根層級的 transform 屬性) 的 projectionCenter 屬性

3 為滑鼠事件新增事件偵聽程式，以呼叫可修改 projectionCenter 的處理常式，讓此屬性可以隨著 center 物件的位置而改變

4 建立四個具有 Accordion 樣式的方塊，以形成透視空間的四周圍

當您測試 ProjectionDragger.swf 這個範例時，請將圓圈拖曳至不同的位置。消失點會隨著圓圈移動，並落在您放開圓圈的位置。當您移動投影中心使其遠離舞台中央時，請留意包圍住空間的四個方塊會延伸並開始扭曲。

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。您可以在 Samples/ProjectionDragger 資料夾中找到 ProjectionDragger 應用程式檔案。

```
package
{
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.geom.Point;
    import flash.events.*;
    public class ProjectionDragger extends Sprite
    {
        private var center : Sprite;
        private var boxPanel:Shape;
        private var inDrag:Boolean = false;

        public function ProjectionDragger():void
        {
            createBoxes();
            createCenter();
        }
        public function createCenter():void
        {
            var  centerRadius:int = 20;

            center = new Sprite();

            // circle
            center.graphics.lineStyle(1, 0x000099);
            center.graphics.beginFill(0xCCCCCC, 0.5);
            center.graphics.drawCircle(0, 0, centerRadius);
            center.graphics.endFill();
            // cross hairs
            center.graphics.moveTo(0, centerRadius);
            center.graphics.lineTo(0, -centerRadius);
            center.graphics.moveTo(centerRadius, 0);
            center.graphics.lineTo(-centerRadius, 0);
            center.x = 175;
            center.y = 175;
            center.z = 0;
            this.addChild(center);

            center.addEventListener(MouseEvent.MOUSE_DOWN, startDragProjectionCenter);
            center.addEventListener(MouseEvent.MOUSE_UP, stopDragProjectionCenter);
            center.addEventListener( MouseEvent.MOUSE_MOVE, doDragProjectionCenter);
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x, center.y);
        }
        public function createBoxes():void
        {
            // createBoxPanel();
            var boxWidth:int = 50;
            var boxHeight:int = 50;
            var numLayers:int = 12;
            var depthPerLayer:int = 50;
```

```
            // var boxVec:Vector.<Shape> = new Vector.<Shape>(numLayers);
            for (var i:int = 0; i < numLayers; i++)
            {
                this.addChild(createBox(150, 50, (numLayers - i) * depthPerLayer, boxWidth, boxHeight,
0xCCCCFF));
                this.addChild(createBox(50, 150, (numLayers - i) * depthPerLayer, boxWidth, boxHeight,
0xFFCCCC));
                this.addChild(createBox(250, 150, (numLayers - i) * depthPerLayer, boxWidth, boxHeight,
0xCCFFCC));
                this.addChild(createBox(150, 250, (numLayers - i) * depthPerLayer, boxWidth, boxHeight,
0xDDDDDD));
            }
        }

        public function createBox(xPos:int = 0, yPos:int = 0, zPos:int = 100, w:int = 50, h:int = 50,
color:int = 0xDDDDDD):Shape
        {
            var box:Shape = new Shape();
            box.graphics.lineStyle(2, 0x666666);
            box.graphics.beginFill(color, 1.0);
            box.graphics.drawRect(0, 0, w, h);
            box.graphics.endFill();
            box.x = xPos;
            box.y = yPos;
            box.z = zPos;
            return box;
        }
        public function startDragProjectionCenter(e:Event)
        {
            center.startDrag();
            inDrag = true;
        }

        public function doDragProjectionCenter(e:Event)
        {
            if (inDrag)
            {
                root.transform.perspectiveProjection.projectionCenter = new Point(center.x, center.y);
            }
        }

        public function stopDragProjectionCenter(e:Event)
        {
            center.stopDrag();
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x, center.y);
            inDrag = false;
        }
    }
}
```

若要執行更複雜的透視投影，請使用 Matrix3D 類別。

執行複雜的 3D 變形

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

Matrix3D 類別可讓您在座標空間中對 3D 座標點執行變形，或者將 3D 座標點從某一座標空間對應至另一座標空間。

即使不懂矩陣數學，您也一樣可以使用 Matrix3D 類別。使用這個類別的各個方法，即可處理大部分常見的變形作業。您不需要擔心要明確設定或計算矩陣中每個元素的值。

將顯示物件的 z 屬性設定為數值之後，即可使用顯示物件之 Transform 物件的 Matrix3D 屬性，擷取顯示物件的變形矩陣。

```
var leafMatrix:Matrix3D = this.transform.matrix3D;
```

您可以使用 Matrix3D 物件的各個方法，對顯示物件執行轉移、旋轉、縮放和透視投影。

使用 Vector3D 類別搭配其 x、y 和 z 屬性，即可管理 3D 座標點。這個類別也可以表示物理學上的空間向量，也就是具有方向和量值的向量。Vector3D 類別的各個方法都能讓您搭配空間向量執行常見的計算工作，例如加法、內積和外積計算。

備註：Vector3D 類別與 ActionScript 的 Vector 類別無關。Vector3D 類別包含可用於定義和操作 3D 座標點的屬性與方法，而 Vector 類別則支援具有類型之物件的陣列。

建立 Matrix3D 物件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

建立或擷取 Matrix3D 物件的主要方式有三種：

- 使用 Matrix3D() 建構函式方法，實體化新的矩陣。Matrix3D() 建構函式會採用 Vector 物件 (包含 16 個數值)，並且將每個值放在矩陣的儲存格中。例如：

  ```
  var rotateMatrix:Matrix3D = new Matrix3D(1,0,0,1, 0,1,0,1, 0,0,1,1, 0,0,0,1);
  ```

- 設定顯示物件的 z 屬性值，再從該物件的 transform.matrix3D 屬性擷取變形矩陣。
- 透過在根顯示物件上呼叫 perspectiveProjection.toMatrix3D() 方法，擷取控制 3D 物件在舞台上顯示方式的 Matrix3D 物件。

套用多種 3D 變形

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

您可以使用 Matrix3D 物件，一次套用許多 3D 變形。例如，如果您要對某一立方體依序執行旋轉、縮放和移動，可以將這三種各自分開的變形套用至立方體的每一個點。不過，另一種方式更有效率，就是在一個 Matrix3D 物件中預先計算多種變形，然後再對每一個點執行一種矩陣變形。

備註：矩陣變形的套用順序很重要，因為矩陣計算並不具有互換性。例如，對於旋轉和轉移作業，套用的順序如果相反，所得到的結果也會不同。

在下列範例中，會示範兩個執行多種 3D 變形的方式。

```
package {
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;
    import flash.geom.*;

public class Matrix3DTransformsExample extends Sprite
    {
        private var rect1:Shape;
        private var rect2:Shape;

public function Matrix3DTransformsExample():void
        {
            var pp:PerspectiveProjection = this.transform.perspectiveProjection;
            pp.projectionCenter = new Point(275,200);
            this.transform.perspectiveProjection = pp;

            rect1 = new Shape();
            rect1.x = -70;
            rect1.y = -40;
            rect1.z = 0;
            rect1.graphics.beginFill(0xFF8800);
            rect1.graphics.drawRect(0,0,50,80);
            rect1.graphics.endFill();
            addChild(rect1);

            rect2 = new Shape();
            rect2.x = 20;
            rect2.y = -40;
            rect2.z = 0;
            rect2.graphics.beginFill(0xFF0088);
            rect2.graphics.drawRect(0,0,50,80);
            rect2.graphics.endFill();
            addChild(rect2);

            doTransforms();
        }

        private function doTransforms():void
        {
            rect1.rotationX = 15;
            rect1.scaleX = 1.2;
            rect1.x += 100;
            rect1.y += 50;
            rect1.rotationZ = 10;

            var matrix:Matrix3D = rect2.transform.matrix3D;
            matrix.appendRotation(15, Vector3D.X_AXIS);
            matrix.appendScale(1.2, 1, 1);
            matrix.appendTranslation(100, 50, 0);
            matrix.appendRotation(10, Vector3D.Z_AXIS);
            rect2.transform.matrix3D = matrix;
        }
    }
}
```

在 doTransforms() 方法中，第一個程式碼區塊會使用 DisplayObject 的屬性，變更矩形形狀的旋轉、縮放、和位置設定。第二個程式碼區塊則會使用 Matrix3D 類別，執行相同的變形。

使用 Matrix3D 方法的主要優點在於，所有的計算作業都是先在矩陣中執行。然後，當顯示物件的 transform.matrix3D 屬性已設定完成時，再將計算的結果套用至顯示物件 (僅套用一次)。設定 DisplayObject 的屬性可以讓原始碼更容易讓人閱讀。不過，每次設定旋轉或縮放屬性時，這麼做就必須執行多個計算作業，並且變更多個顯示物件屬性。

如果程式碼將對多個顯示物件套用多次相同的複雜變形，請將 Matrix3D 物件儲存為變數，然後再重複執行套用的動作。

使用 Matrix3D 物件以重新排列顯示物件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

如之前所提及，顯示清單中顯示物件的圖層順序會決定顯示的圖層順序，而與顯示物件的相對 z 軸無關。如果您的動畫將顯示物件的屬性改變為與顯示清單順序不同的順序，則檢視者看到的顯示物件圖層順序可能不會對應至 x 軸圖層的順序。如此一來，看起來應該遠離檢視者的物件可能會出現在較靠近檢視者的物件前方。

若要確保 3D 顯示物件的圖層順序能對應至這些物件的相對深度，請使用如下所示的方法：

1. 使用 Transform 物件的 getRelativeMatrix3D() 方法，取得子系 3D 顯示物件的相對 z 軸。
2. 使用 removeChild() 方法，從顯示清單移除這些物件。
3. 根據這些顯示物件的 z 軸值加以排列。
4. 使用 addChild() 方法，依照相反的順序，將這些子系重新加入至顯示清單中。

上述重新排列順序的步驟，可以確保這些物件都能依照其相對 z 軸順序顯示。

下列程式碼會強制修正某一 3D 方塊的六個顯示表面。在對方塊套用旋轉之後，程式碼會重新排列方塊的表面：

```
public var faces:Array; . . .

public function ReorderChildren()
{
    for(var ind:uint = 0; ind < 6; ind++)
    {
        faces[ind].z = faces[ind].child.transform.getRelativeMatrix3D(root).position.z;
        this.removeChild(faces[ind].child);
    }
    faces.sortOn("z", Array.NUMERIC | Array.DESCENDING);
    for (ind = 0; ind < 6; ind++)
    {
        this.addChild(faces[ind].child);
    }
}
```

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。您可以在 Samples/ReorderByZ 資料夾中找到這些應用程式檔案。

使用三角形產生 3D 特效

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

在 ActionScript 中，您必須使用 Graphics.drawTriangles() 方法執行點陣圖變形，因為 3D 模型都是以空間中的一組三角形表示 (不過，由於 Flash Player 和 AIR 都不支援深度緩衝區，因此顯示物件本質上仍是以平面 (或 2D) 呈現。您可以在第 297 頁「瞭解 Flash Player 和 AIR 執行階段中的 3D 顯示物件」中找到相關說明)。Graphics.drawTriangles() 方法與 Graphics.drawPath() 方法類似，前者會使用一組座標來繪製三角形路徑。

若要熟悉 Graphics.drawPath() 的用法，請參閱第 199 頁 「繪製路徑」。

Graphics.drawTriangles() 方法會使用 Vector.<Number>，指定三角形路徑的座標點位置：

```
drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null, uvtData:Vector.<Number> = null,
culling:String = "none"):void
```

drawTriangles() 的第一個參數 vertices 是唯一必要的參數。這個參數是數字的向量，可定義繪製三角形時所需的各個座標。每三組座標 (共六個數字) 即可表示一個三角形路徑。如果不使用 indices 參數，向量的長度就一定要是六的因數，因為每個三角形都需要三組座標 (三組 x/y 值)。例如：

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([
        10,10,  100,10,  10,100,
        110,10, 110,100, 20,100]));
```

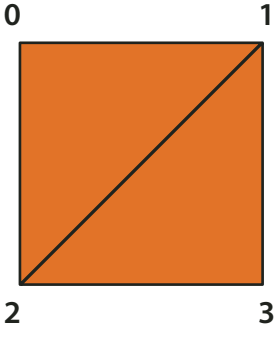
上述三角形都不會共用任何座標點，但是如果有的話，drawTriangles() 的第二個參數 indices 就可以針對多個三角形，用來重複使用 vertices 向量中的值。

使用 indices 參數時，請注意 indices 值都是座標點索引，而非 vertices 陣列元素的直接相對索引。換句話說，如 indices 所定義，vertices 向量中的索引實際上就是真實的索引除以 2。例如，對 vertices 向量的第三個座標點而言，即使該點的第一個數值是從向量索引 4 開始，您仍然是使用 2 做為 indices 的值。

例如，以使用 indices 參數合併兩個三角形來共用對角斜邊為例：

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([10,10, 100,10, 10,100, 100,100]),
    Vector.<int>([0,1,2, 1,3,2]));
```

請注意，雖然已使用兩個三角形繪製了一個正方形，但是 vertices 向量中僅指定了四個座標點。使用 indices 時，每個三角形都會重複用到這兩個三角形所共用的兩個點。如此即可將所有頂點數目從 6 個 (12 個數字) 減少為 4 個 (8 個數字)：



使用 vertices 參數合併兩個三角形所繪製的正方形

如果是較大的三角形網格，而且其中有多個三角形都共用大多數座標點時，這項技巧就很有用。

您可以對三角形套用所有填色。這些填色能套用至所產生的三角形網格，就像套用至其它任何形狀一樣。

對點陣圖執行變形

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

點陣圖變形可讓人產生透視的錯覺，或對三維物件產生具有「紋理」的錯覺。具體來說，您可以使點陣圖朝消失點的方向扭曲，讓影像朝著消失點移動時，看起來就像是縮小了一樣。或者，您也可以使用二維點陣圖為三維物件建立表面，讓人產生三維物件具有紋理或「折合」的錯覺。

使用消失點的二維表面以及使用點陣圖折合的三維物件

UV 對應

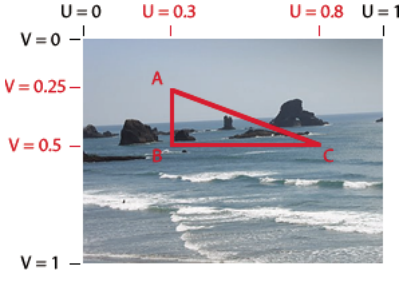
Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

開始使用材質之後，您會希望使用 drawTriangles() 的 uvtData 參數。這個參數可以讓您設定用於點陣圖填色的 UV 對應。

UV 對應是一種為物件加上材質的方法，所仰賴的兩個值為 U 水平 (x) 值和 V 垂直 (y) 值 (以百分比為基礎，而非像素值)。0 U 和 0 V 是影像的左上角，而 1 U 和 1 V 則是影像的右下角：

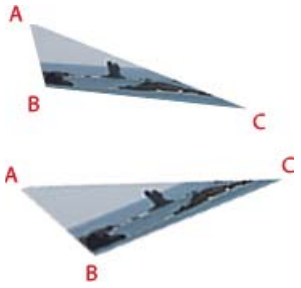
點陣圖上 UV 0 和 1 的位置

您可以指定 UV 座標當做三角形的向量，將這些向量與影像上對應的位置產生關聯：



點陣圖影像上三角區域的 UV 座標

三角形各點的 UV 值都會保持一致：



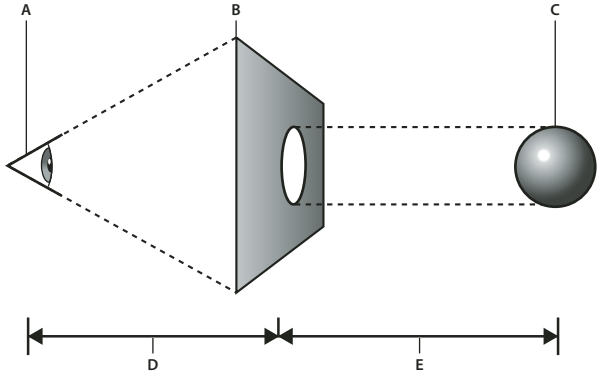
三角形的各頂點會移動且點陣圖會扭曲，如此各點的 UV 值都保持不變

將 ActionScript 3D 變形套用至與點陣圖有關聯的三角形時，會根據 UV 值將點陣圖影像套用至三角形。因此，您不需要使用矩陣計算，只要設定或調整 UV 值，即可建立三維特效。

Graphics.drawTriangles() 方法也能接受三維變形的部分選擇性資訊：T 值。uvtData 中的 T 值表示 3D 透視，更具體來說，就是相關聯頂點的縮放比例係數。UVT 對應會將透視更正加入至 UV 對應。例如，如果將物件放在 3D 空間中遠離檢視者的位置，讓它看起來是原始大小的 50%，那麼，因為繪製三角形的目的是要表示 3D 空間中的物件，所以物件在 z 軸上的位置就會決定物件的 T 值。也就是說，物件的 T 值就是 0.5。決定 T 值的方程式如下：

```
T = focalLength/(focalLength + z);
```

在這個方程式中，focalLength 表示焦距或經過計算的 「螢幕」位置，這個位置會規定檢視中所提供的透視量。



焦距和 z 值
A. 視點 **B.** 螢幕 **C.** 3D 物件 **D.** focalLength 值 **E.** z 值

T 的值是用於縮放基本的形狀，讓它們看起來似乎在遠方，離自己有一段距離。這個值通常用於將 3D 座標點轉換成 2D 座標點。如果是 UVT 資料，則通常會用於縮放具有透視特效之三角形內座標點間的點陣圖。

當您定義 UVT 值時，T 值會直接接在針對頂點所定義的 UV 值後面。得出 T 值之後，uvtData 參數中的每三個值 (U、V 和 T) 都會符合 vertices 參數中的每兩個值 (x 和 y)。只有 UV 值時，uvtData.length == vertices.length。得出 T 值之後，uvtData.length = 1.5*vertices.length。

下列範例會顯示使用 UVT 資料在 3D 空間中旋轉的平面。在範例中，會使用 「helper」類別 ImageLoader 載入名為 ocean.jpg 的影像，以便將影像指定給 BitmapData 物件。

以下為 ImageLoader 類別的原始碼 (請儲存下列程式碼，並命名為 ImageLoader.as)：

```
package {
    import flash.display.*
    import flash.events.*;
    import flash.net.URLRequest;
    public class ImageLoader extends Sprite {
        public var url:String;
        public var bitmap:Bitmap;
    public function ImageLoader(loc:String = null) {
        if (loc != null){
            url = loc;
            loadImage();
        }
    }
    public function loadImage():void{
        if (url != null){
            var loader:Loader = new Loader();
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onComplete);
            loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR, onIoError);

                var req:URLRequest = new URLRequest(url);
                loader.load(req);
            }
        }

    private function onComplete(event:Event):void {
            var loader:Loader = Loader(event.target.loader);
            var info:LoaderInfo = LoaderInfo(loader.contentLoaderInfo);
            this.bitmap = info.content as Bitmap;
            this.dispatchEvent(new Event(Event.COMPLETE));
    }

    private function onIoError(event:IOErrorEvent):void {
            trace("onIoError: " + event);
    }
    }
}
```

以下為 ActionScript，其中使用了三角形、UV 對應和 T 值，使影像看起來就像一邊旋轉並且朝著消失點縮小。請儲存下列程式碼，並命名為 Spinning3dOcean.as：

```
package {
    import flash.display.*
    import flash.events.*;
    import flash.utils.getTimer;

    public class Spinning3dOcean extends Sprite {
        // plane vertex coordinates (and t values)
        var x1:Number = -100,y1:Number = -100,z1:Number = 0,t1:Number = 0;
        var x2:Number = 100,y2:Number = -100,z2:Number = 0,t2:Number = 0;
        var x3:Number = 100,y3:Number = 100,z3:Number = 0,t3:Number = 0;
        var x4:Number = -100,y4:Number = 100,z4:Number = 0,t4:Number = 0;
        var focalLength:Number = 200;
        // 2 triangles for 1 plane, indices will always be the same
        var indices:Vector.<int>;

        var container:Sprite;

        var bitmapData:BitmapData; // texture
        var imageLoader:ImageLoader;
        public function Spinning3dOcean():void {
            indices =  new Vector.<int>();
            indices.push(0,1,3, 1,2,3);
```

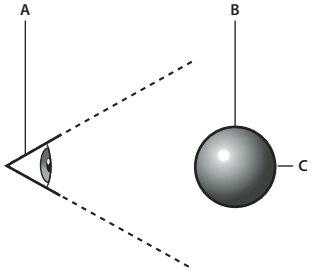
```
            container = new Sprite(); // container to draw triangles in
            container.x = 200;
            container.y = 200;
            addChild(container);

            imageLoader = new ImageLoader("ocean.jpg");
            imageLoader.addEventListener(Event.COMPLETE, onImageLoaded);
        }
        function onImageLoaded(event:Event):void {
            bitmapData = imageLoader.bitmap.bitmapData;
            // animate every frame
            addEventListener(Event.ENTER_FRAME, rotatePlane);
        }
        function rotatePlane(event:Event):void {
            // rotate vertices over time
            var ticker = getTimer()/400;
            z2 = z3 = -(z1 = z4 = 100*Math.sin(ticker));
            x2 = x3 = -(x1 = x4 = 100*Math.cos(ticker));

            // calculate t values
            t1 = focalLength/(focalLength + z1);
            t2 = focalLength/(focalLength + z2);
            t3 = focalLength/(focalLength + z3);
            t4 = focalLength/(focalLength + z4);

            // determine triangle vertices based on t values
            var vertices:Vector.<Number> = new Vector.<Number>();
            vertices.push(x1*t1,y1*t1, x2*t2,y2*t2, x3*t3,y3*t3, x4*t4,y4*t4);
            // set T values allowing perspective to change
            // as each vertex moves around in z space
            var uvtData:Vector.<Number> = new Vector.<Number>();
            uvtData.push(0,0,t1, 1,0,t2, 1,1,t3, 0,1,t4);

            // draw
            container.graphics.clear();
            container.graphics.beginBitmapFill(bitmapData);
            container.graphics.drawTriangles(vertices, indices, uvtData);
        }
    }
}
```

若要測試這個範例，請將這兩個類別檔案儲存於「ocean.jpg」影像所在的目錄。您可以看到原始點陣圖的變形方式，看起來就像在 3D 空間中一邊旋轉，一邊朝著遠方消失。

剔除

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

剔除是一項程序，可決定描繪器不應該顯示三維物件的哪些表面，因為從目前的視點無法看到這些隱藏的表面。在 3D 空間中，從視點看不到三維物件「背後」的表面：



從視點無法看到 3D 物件的背面。
A. 視點 **B.** 3D 物件 **C.** 三維物件的背面

所有三角形在本質上都一定會顯示，無論其大小、形狀或位置為何都一樣。剔除可確保 Flash Player 或 AIR 能正確顯示您的 3D 物件。此外，為了在顯示週期期間節省效能，有時您會希望描繪器略過某些三角形。以在空間中旋轉的立方體為例。無論在任何時候，您都不會看到立方體呈現三個以上的表面，因為看不到的那些表面都在看得到的表面的相反方向。由於會看不到這些表面，因此描繪器就不需要加以繪製。如果不使用剔除，Flash Player 或 AIR 便會一併顯示前後表面。

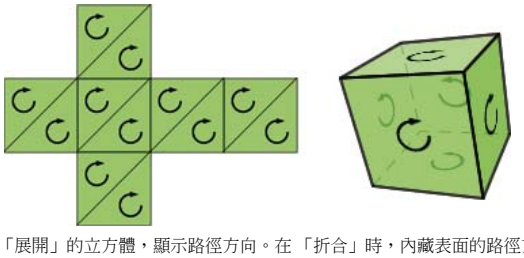
立方體，其中幾個表面是從目前視點看不到的

因此，Graphics.drawTriangles() 方法便具有第四個參數，以建立剔除值。

```
public function drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null,
uvtData:Vector.<Number> = null, culling:String = "none"):void
```

culling 參數具有衍生自 TriangleCulling 列舉類別的值：TriangleCulling.NONE、TriangleCulling.POSITIVE 和 TriangleCulling.NEGATIVE。這些值與定義物件表面之三角形路徑的方向有關。用於決定剔除的 ActionScript API 會假設 3D 形狀的所有外顯三角形都應以相同的路徑方向繪製。一旦某個三角形表面翻轉之後，路徑方向也會變更。此時，就能剔除 (不再顯示) 該三角形。

如此一來，POSITIVE 的 TriangleCulling 值便會移除具有正向路徑方向 (順時針) 的三角形；而 NEGATIVE 的 TriangleCulling 值則會移除具有負向路徑方向 (逆時針) 的三角形。如果是立方體，那麼外顯的表面便具有正向路徑方向，而內藏的表面則具有負向路徑方向：



「展開」的立方體，顯示路徑方向。在「折合」時，內藏表面的路徑方向則相反。

若要瞭解剔除的運作方式，請從先前第 308 頁「UV 對應」的範例開始，將 drawTriangles() 方法的 culling 參數設定為 TriangleCulling.NEGATIVE：

```
container.graphics.drawTriangles(vertices, indices, uvtData, TriangleCulling.NEGATIVE);
```

請注意，當物件旋轉時，不會顯示影像的「背面」。

第 20 章 使用文字的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要在 Adobe® Flash® Player 或 Adobe® AIR™ 中顯示螢幕上的文字，請使用 TextField 類別的實體，或使用 Flash 文字引擎類別。這些類別可讓您建立、顯示以及格式化文字。您也可以使用 Text Layout Framework (TLF)，這是以 Flash 文字引擎類別為基礎的元件庫，但其設計更加方便使用。在行動裝置上，您可以將 StageText 類別用於文字輸入。

您可以建立文字欄位的特定內容，或是指定文字的來源，然後設定該文字的外觀。您也可以在使用者輸入文字或按一下超文字連結時，回應使用者事件。

TextField 類別和 Flash Text Engine 類別二者都可以讓您在 Flash Player 和 AIR 中顯示以及管理文字。您可以使用 TextField 類別來建立文字物件，以便用於顯示和輸入。TextField 類別為其他文字型組件 (例如 TextArea 和 TextInput) 提供基礎。您可以使用 TextFormat 類別來設定 TextField 物件的字元和段落格式，也可以使用 Textfield.styleSheet 屬性和 StyleSheet 類別來套用 「階層式樣式表 (CSS)」。您可以將 HTML 格式的文字 (可以包含內嵌媒體，例如影片片段、SWF 檔、GIF 檔、PNG 檔和 JPEG 檔) 直接指定給文字欄位。

自 Flash Player 10 和 Adobe AIR 1.5 開始提供的 Flash 文字引擎可提供精確控制文字規格、格式和雙向文字的低階支援。它也提供改善的文字流排和增強的語言支援。雖然您可以使用 Flash Text Engine 來建立以及管理文字元素，不過它主要的設計是做為建立文字處理元件的基礎，而且需要較多的程式設計專業。Text Layout Framework 包含以 Flash Text Engine 為基礎的文字處理元件，可以提供更便利的方法來使用新文字引擎的進階功能。Text Layout Framework 是完全以 ActionScript 3.0 建立的可延伸元件庫。您可以使用現有的 TLF 元件，或是使用此架構來建立您自己的文字元件。

從 AIR 3 開始提供使用的 StageText 類別，提供原生文字輸入欄位。因為這個欄位是由裝置作業系統所提供，所以會隨著裝置最熟悉的使用者而提供使用體驗。StageText 實體不是顯示物件。與其將它加入顯示清單，請為實體指定一個舞台，並在那個稱為 「檢視區域」的舞台上指定顯示區域。StageText 實體會顯示在任何顯示物件的前方。

如需有關這些主題的詳細資訊，請參閱：

- 第 315 頁 「使用 TextField 類別」
- 第 336 頁 「使用 Flash 文字引擎」
- 第 362 頁 「使用 Text Layout Framework」
- 使用 StageText 進行原生文字輸入

重要概念與術語

下列參考清單包含文字處理所涉及的重要術語：

階層式樣式表 標準語法，用以指定透過 XML (或 HTML) 格式建構之內容的樣式和格式。

裝置字體 安裝在使用者電腦上的字體。

動態文字欄位 ActionScript 可以變更其內容但不能透過使用者輸入變更其內容的文字欄位。

內嵌字體 已將其文字外框資料儲存在應用程式 SWF 檔中的字體。

HTML 文字 使用 ActionScript (內含 HTML 格式標籤以及實際文字內容) 輸入至文字欄位的文字內容。

輸入文字欄位 可以透過使用者輸入或 ActionScript 變更其內容的文字欄位。

特殊字距 字元組之間的間距調整，用以讓單字的間距成等比，讓文字更容易閱讀。

靜態文字欄位 在編寫工具中建立的文字欄位，但執行 SWF 檔時不能變更此文字欄位的內容。

文字行規格 文字欄位中各文字內容部分大小的測量結果，例如文字的基線、字元頂端的高度、下緣大小 (小寫字母延伸到基線下方的部分) 等。

字距調整 調整一組字母或文字區塊之間的間距以增加或減少密度，讓文字更容易閱讀。

第 21 章 使用 TextField 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 TextField 類別的實體來顯示文字，或是在 Adobe® Flash® Player 或 Adobe® AIR™ 中建立螢幕的文字輸入欄位。TextField 類別是其他文字型組件 (例如 TextArea 組件或 TextInput 組件) 的基礎。

文字欄位內容可以預先在 SWF 檔中指定、從文字檔案或資料庫中載入，或由使用者與您的應用程式互動來輸入。在文字欄位內，文字可以出現為顯示的 HTML 內容，並在顯示的 HTML 中內嵌影像。在您建立文字欄位實體之後，可以使用 flash.text 類別 (例如 TextFormat 和 StyleSheet) 來控制文字的外觀。flash.text 套件幾乎包含了所有與在 ActionScript 中建立、管理和格式化文字相關的類別。

您可以藉由使用 TextFormat 物件定義格式並將該物件指定給文字欄位，來格式化文字。如果您的文字欄位包含 HTML 文字，則可以將 StyleSheet 物件套用至文字欄位，藉此將樣式指定給文字欄位內容的特定部分。TextFormat 物件或 StyleSheet 物件包含定義文字外觀的屬性 (例如顏色、大小和寬度)。TextFormat 物件會將屬性指定給文字欄位內的所有內容，或是指定給某範圍的文字。例如，在相同的文字欄位內，某個句子可以是紅色粗體文字，而下一句可以是藍色斜體文字。

除了 flash.text 套件中的類別之外，您還可以使用 flash.events.TextEvent 類別來回應與文字相關的使用者動作。

更多說明主題

第 321 頁 「指定文字格式」

第 316 頁 「顯示 HTML 文字」

第 322 頁 「套用階層式樣式表」

顯示文字

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

雖然編寫工具 (例如 Adobe Flash Builder 和 Flash Professional) 提供數個顯示文字的選項 (包含文字相關組件或文字工具)，但是以程式設計方式顯示文字的最簡單方式還是透過文字欄位。

文字類型

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

文字欄位內的文字類型依其來源來區分特性：

- 動態文字

  動態文字包含從外部來源 (例如文字檔案、XML 檔，甚至是遠端網路服務) 中載入的內容。

- 輸入文字

  輸入文字是使用者輸入的任何文字，或是使用者可以編輯的動態文字。您可以設定樣式表來格式化輸入文字，或使用 flash.text.TextFormat 類別，將屬性指定給輸入內容的文字欄位。如需詳細資訊，請參閱第 319 頁 「擷取文字輸入」。

- 靜態文字

靜態文字只能透過 Flash Professional 建立。您無法使用 ActionScript 3.0 建立靜態文字實體。不過，您可以使用 ActionScript 類別 (例如 StaticText 和 TextSnapshot) 來處理現有靜態文字實體。如需詳細資訊，請參閱第 326 頁 「使用靜態文字」。

修改文字欄位內容

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以將字串指定給 flash.text.TextField.text 屬性，以定義動態文字。您可以將字串直接指定給屬性，如下所示：

```
myTextField.text = "Hello World";
```

您也可以透過指令碼中定義的變數，將值指定給 text 屬性 (如下例所示)：

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class TextWithImage extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myText:String = "Hello World";

        public function TextWithImage()
        {
            addChild(myTextBox);
            myTextBox.text = myText;
        }
    }
}
```

您也可以透過遠端變數，將值指定給 text 屬性。有三個選項可讓您從遠端來源載入文字值：

- flash.net.URLLoader 和 flash.net.URLRequest 類別會從本機或遠端位置載入文字的變數。
- FlashVars 特質內嵌於裝載 SWF 檔的 HTML 頁面中，而且可以包含文字變數的值。
- flash.net.SharedObject 類別會管理值的永續儲存區。如需詳細資訊，請參閱第 604 頁 「儲存本機資料」。

顯示 HTML 文字

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

flash.text.TextField 類別具有 htmlText 屬性，該屬性可將文字字串識別為包含格式化內容之 HTML 標籤的文字字串。與下列範例相同，您必須將字串值指定給 htmlText 屬性 (不是 text 屬性)，讓 Flash Player 或 AIR 將文字顯示為 HTML：

```
var myText:String = "<p>This is <b>some</b> content to <i>render</i> as <u>HTML</u> text.</p>";
myTextBox.htmlText = myText;
```

Flash Player 和 AIR 支援 htmlText 屬性的一部分 HTML 標籤和實體。「Adobe® Flash® Professional CS5 的 ActionScript® 3.0 參考」中的 flash.text.TextField.htmlText 屬性說明提供支援的 HTML 標籤和實體詳細資訊。

使用 htmlText 屬性指定內容之後，就可以使用樣式表或 textformat 標籤來管理內容的格式。如需詳細資訊，請參閱第 321 頁 「格式化文字」。

在文字欄位中使用影像

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

另一個將內容顯示為 HTML 文字的優點，是您可以將影像包含在文字欄位中。您可以使用 img 標籤來參考本機或遠端影像，或是讓它出現在相關的文字欄位內。

下列範例會建立名為 myTextBox 的文字欄位，並在顯示的文字內包含眼睛形狀的 JPG 影像 (儲存在與 SWF 檔相同的目錄中)：

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class TextWithImage extends Sprite
    {
        private var myTextBox:TextField;
        private var myText:String = "<p>This is <b>some</b> content to <i>test</i> and <i>see</i></p><p><img
src='eye.jpg' width='20' height='20'></p><p>what can be rendered.</p><p>You should see an eye image and some
<u>HTML</u> text.</p>";

        public function TextWithImage()
        {
            myTextBox.width = 200;
            myTextBox.height = 200;
            myTextBox.multiline = true;
            myTextBox.wordWrap = true;
            myTextBox.border = true;

            addChild(myTextBox);
            myTextBox.htmlText = myText;
        }
    }
}
```

img 標籤支援 JPEG、GIF、PNG 和 SWF 檔。

在文字欄位中捲動文字

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在許多情況下，您的文字會比顯示文字的文字欄位還要長。或是，您的輸入欄位可讓使用者輸入多於一次可以顯示的文字。您可以使用 flash.text.TextField 類別的捲動相關屬性來垂直或水平管理較長的內容。

捲動相關屬性包含 TextField.scrollV、TextField.scrollH、maxScrollV 和 maxScrollH。您可以使用這些屬性來回應事件 (例如滑鼠按鍵次數或按下按鍵)。

下列範例會建立固定大小的文字欄位，且文字欄位包含多於欄位一次可顯示的文字。使用者按一下文字欄位時，文字會垂直捲動。

```
package
{
    import flash.display.Sprite;
    import flash.text.*;
    import flash.events.MouseEvent;

    public class TextScrollExample extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myText:String = "Hello world and welcome to the show. It's really nice to meet you.
Take your coat off and stay a while. OK, show is over. Hope you had fun. You can go home now. Don't forget
to tip your waiter. There are mints in the bowl by the door. Thank you. Please come again.";

        public function TextScrollExample()
        {
            myTextBox.text = myText;
            myTextBox.width = 200;
            myTextBox.height = 50;
            myTextBox.multiline = true;
            myTextBox.wordWrap = true;
            myTextBox.background = true;
            myTextBox.border = true;

            var format:TextFormat = new TextFormat();
            format.font = "Verdana";
            format.color = 0xFF0000;
            format.size = 10;

            myTextBox.defaultTextFormat = format;
            addChild(myTextBox);
            myTextBox.addEventListener(MouseEvent.MOUSE_DOWN, mouseDownScroll);
        }

        public function mouseDownScroll(event:MouseEvent):void
        {
            myTextBox.scrollV++;
        }
    }
}
```

選取和處理文字

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以選取動態或輸入文字。因為 TextField 類別的文字選取屬性和方法會使用索引位置來設定要處理的文字範圍，所以您可以透過程式設計方式選取動態或輸入文字，即使您不知道內容也是一樣。

備註：在 Flash Professional 中，如果您選擇靜態文字欄位上的 「可選取」選項，則匯出放在顯示清單上的文字欄位會是一般動態文字欄位。

選取文字

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

flash.text.TextField.selectable 屬性預設是 true，而且您可以使用 setSelection() 方法，透過程式設計方式選取文字。

例如，您可以設定當使用者按一下文字欄位時，會選取文字欄位內的特定文字：

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL CAPS is selected.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, selectText);

function selectText(event:MouseEvent):void
{
    myTextField.setSelection(49, 65);
}
```

同樣地，如果在初次顯示文字時，您想要選取文字欄位內的文字，請建立將文字欄位新增至顯示清單時所呼叫的事件處理常式函數。

擷取使用者選取的文字

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

TextField selectionBeginIndex 和 selectionEndIndex 是 「唯讀」屬性，因此無法設定為透過程式設計方式選取文字，但可以用來擷取使用者目前已選取的項目。此外，輸入文字欄位可以使用 caretIndex 屬性。

例如，下列程式碼會追蹤使用者所選文字的索引值。

```
var myTextField:TextField = new TextField();
myTextField.text = "Please select the TEXT IN ALL CAPS to see the index values for the first and last
letters.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.MOUSE_UP, selectText);

function selectText(event:MouseEvent):void
{
    trace("First letter index position: " + myTextField.selectionBeginIndex);
    trace("Last letter index position: " + myTextField.selectionEndIndex);
}
```

您可以將 TextFormat 物件屬性的集合套用至選取範圍，以變更文字外觀。如需有關將 TextFormat 屬性集合套用至所選取文字的詳細資訊，請參閱第 324 頁 「格式化文字欄位內的文字範圍」。

擷取文字輸入

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

文字欄位的 type 屬性預設為 dynamic。如果您使用 TextFieldType 類別將 type 屬性設為 input，則您可以收集使用者輸入，並儲存值以用於應用程式的其他部分。輸入文字欄位適用於表單和任何應用程式，而使用者可使用此表單和應用程式來定義文字值以便在程式的其他位置使用。

例如，下列程式碼會建立名為 myTextBox 的輸入文字欄位。使用者在欄位中輸入文字時，會觸發 textInput 事件。事件處理常式 textInputCapture 會擷取輸入的文字字串，並將變數指定給它。Flash Player 或 AIR 會在另一個文字欄位 (myOutputBox) 中顯示新文字。

```
package
{
    import flash.display.Sprite;
    import flash.display.Stage;
    import flash.text.*;
    import flash.events.*;

    public class CaptureUserInput extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myOutputBox:TextField = new TextField();
        private var myText:String = "Type your text here.";

        public function CaptureUserInput()
        {
            captureText();
        }

        public function captureText():void
        {
            myTextBox.type = TextFieldType.INPUT;
            myTextBox.background = true;
            addChild(myTextBox);
            myTextBox.text = myText;
            myTextBox.addEventListener(TextEvent.TEXT_INPUT, textInputCapture);
        }

        public function textInputCapture(event:TextEvent):void
        {
            var str:String = myTextBox.text;
            createOutputBox(str);
        }

        public function createOutputBox(str:String):void
        {
            myOutputBox.background = true;
            myOutputBox.x = 200;
            addChild(myOutputBox);
            myOutputBox.text = str;
        }

    }
}
```

限制文字輸入

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

因為輸入文字欄位通常用於應用程式的表單或對話方塊中，所以您可能想要限制使用者可以在文字欄位中輸入的字元類型，或甚至隱藏文字 (例如，針對密碼)。flash.text.TextField 類別具有 displayAsPassword 屬性和 restrict 屬性，您可以利用這些屬性進行設定以控制使用者輸入。

displayAsPassword 屬性會在使用者輸入文字時隱藏文字 (將它顯示為一連串的星號)。displayAsPassword 設為 true 時，「剪下」和「複製」指令以及其對應的鍵盤快捷鍵將失去作用。如下例所示，指定 displayAsPassword 屬性的方式就和指定其他屬性 (例如背景和顏色) 一樣：

```
myTextBox.type = TextFieldType.INPUT;
myTextBox.background = true;
myTextBox.displayAsPassword = true;
addChild(myTextBox);
```

restrict 屬性則較為複雜，因為您需要指定使用者可以在輸入文字欄位中鍵入的字元，您可以允許特定字母、數字或是某範圍的字母、數字和字元。下列程式碼允許使用者只在文字欄位中輸入大寫字母 (而不是數字或特殊字元)：

```
myTextBox.restrict = "A-Z";
```

ActionScript 3.0 使用連字符號來定義範圍，並使用跳脫字元來定義排除字元。如需有關定義輸入文字欄位限制的詳細資訊，請參閱 「Adobe® Flash® Professional CS5 的 ActionScript® 3.0 參考」中的 flash.text.TextField.restrict 屬性項目。

備註：如果您使用 flash.text.TextField.restrict 屬性，執行階段就會自動將限制的字母轉換成允許的大小寫。如果您使用 fl.text.TLFTextField.restrict 屬性 (也就是，使用 TLF 文字欄位)，則執行階段會忽略限制的字母。

格式化文字

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

有數個選項可讓您透過程式設計方式來格式化文字的顯示方式。您可以直接在 TextField 實體上設定屬性 (例如 TextFIeld.thickness、TextField.textColor 和 TextField.textHeight 屬性)。也可以使用 htmlText 屬性來指定文字欄位內容，以及使用支援的 HTML 標籤 (例如 b、i 和 u)。但您也可以將 TextFormat 物件套用至含有純文字的文字欄位，或是將 StyleSheet 物件套用至含有 htmlText 屬性的文字欄位。使用 TextFormat 和 StyleSheet 物件可讓您更能控制應用程式中的文字外觀和其一致性。您可以定義 TextFormat 或 StyleSheet 物件，並將它套用至應用程式中的多個或所有文字欄位。

指定文字格式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 TextFormat 類別設定數種不同的文字顯示屬性，並將它們套用至 TextField 物件的整個內容，或是套用至文字範圍。

下列範例會將一個 TextFormat 物件套用至整個 TextField 物件，而將第二個 TextFormat 物件套用至該 TextField 物件內的文字範圍：

```
var tf:TextField = new TextField();
tf.text = "Hello Hello";

var format1:TextFormat = new TextFormat();
format1.color = 0xFF0000;

var format2:TextFormat = new TextFormat();
format2.font = "Courier";

tf.setTextFormat(format1);
var startRange:uint = 6;
tf.setTextFormat(format2, startRange);

addChild(tf);
```

TextField.setTextFormat() 方法只會影響已在文字欄位中顯示的文字。如果變更 TextField 中的內容，則應用程式可能需要再呼叫一次 TextField.setTextFormat() 方法以重新套用格式。您也可以設定 TextField defaultTextFormat 屬性，以指定要用於使用者輸入文字的格式。

套用階層式樣式表

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

文字欄位可以包含純文字或 HTML 格式文字。純文字儲存於實體的 text 屬性中，而 HTML 文字儲存於 htmlText 屬性中。

您可以使用 CSS 樣式宣告，來定義可以套用至多個不同文字欄位的文字樣式。CSS 樣式宣告可以透過您的應用程式碼來建立，或是在執行時期從外部 CSS 檔載入。

flash.text.StyleSheet 類別可處理 CSS 樣式。StyleSheet 類別可辨識的 CSS 屬性集合有限。如需 StyleSheet 類別所支援樣式屬性的詳細清單，請參閱 「Adobe® Flash® Professional CS5 的 ActionScript® 3.0 參考」中的 flash.textStylesheet 項目。

如下例所示，您可以透過您的程式碼來建立 CSS，並使用 StyleSheet 物件將這些樣式套用至 HTML 文字：

```
var style:StyleSheet = new StyleSheet();

var styleObj:Object = new Object();
styleObj.fontSize = "bold";
styleObj.color = "#FF0000";
style.setStyle(".darkRed", styleObj);

var tf:TextField = new TextField();
tf.styleSheet = style;
tf.htmlText = "<span class = 'darkRed'>Red</span> apple";

addChild(tf);
```

建立 StyleSheet 物件之後，此範例程式碼會建立簡單物件來保留一組樣式宣告屬性。然後，它會呼叫 StyleSheet.setStyle() 方法，將新樣式新增至名稱為 「.darkred」的樣式表。接著，它會將 StyleSheet 物件指定給 TextField styleSheet 屬性，以套用樣式表格式。

若要讓 CSS 樣式生效，您必須先將樣式表套用至 TextField 物件，然後才能設定 htmlText 屬性。

依據設計，您無法編輯含有 ⊠ 式表的文字欄位。如果您擁有輸入文字欄位，並在其中指定 ⊠ 式表，則文字欄位會顯示 ⊠ 式表的屬性，但是文字欄位不允許使用者在其中輸入新的文字。而且，您無法在含有指定 ⊠ 式表的文字欄位中使用下列 ActionScript API：

- TextField.replaceText() 方法
- TextField.replaceSelectedText() 方法
- TextField.defaultTextFormat 屬性
- TextField.setTextFormat() 方法

如果文字欄位已指定樣式表，但之後將 TextField.styleSheet 屬性設為 null，則 TextField.text 和 TextField.htmlText 屬性的內容會將標籤和特質新增至其內容，以併入先前指定的樣式表格式。若要保留原始 htmlText 屬性，請先將它儲存至變數，然後再將樣式表設為 null。

載入外部 CSS 檔

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您可以在執行時期從外部檔案載入 CSS 資訊時，CSS 方法在格式化方面的功能會更加強大。當 CSS 資料位在應用程式的外部時，您可以變更應用程式中的文字視覺樣式，不需要變更 ActionScript 3.0 原始碼。部署您的應用程式之後，就可以變更外部 CSS 檔以變更應用程式的外觀，不需要重新部署應用程式 SWF 檔。

StyleSheet.parseCSS() 方法會將含有 CSS 資料的字串轉換為 StyleSheet 物件中的樣式宣告。下列範例會顯示如何讀取外部 CSS 檔，以及如何將其樣式宣告套用至 TextField 物件。

首先，以下是要載入之 CSS 檔 (名稱為 example.css) 的內容：

```
p {
    font-family: Times New Roman, Times, _serif;
    font-size: 14;
}

h1 {
    font-family: Arial, Helvetica, _sans;
    font-size: 20;
    font-weight: bold;
}

.bluetext {
    color: #0000CC;
}
```

接著是載入 example.css 檔並將樣式套用至 TextField 內容之類別的 ActionScript 程式碼：

```
package
{
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import flash.text.StyleSheet;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;

    public class CSSFormattingExample extends Sprite
    {
        var loader:URLLoader;
        var field:TextField;
        var exampleText:String = "<h1>This is a headline</h1>" +
            "<p>This is a line of text. <span class='bluetext'>" +
            "This line of text is colored blue.</span></p>";

        public function CSSFormattingExample():void
        {
            field = new TextField();
            field.width = 300;
            field.autoSize = TextFieldAutoSize.LEFT;
            field.wordWrap = true;
            addChild(field);

            var req:URLRequest = new URLRequest("example.css");

            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            var sheet:StyleSheet = new StyleSheet();
            sheet.parseCSS(loader.data);
            field.styleSheet = sheet;
            field.htmlText = exampleText;
        }
    }
}
```

載入 CSS 資料時，onCSSFileLoaded() 方法會執行並呼叫 StyleSheet.parseCSS() 方法，以將樣式宣告傳輸到 StyleSheet 物件。

格式化文字欄位內的文字範圍

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

flash.text.TextField 類別中非常實用的方法是 setTextFormat() 方法。使用 setTextFormat()，您可以將特定屬性指定給文字欄位某部分的內容，以回應使用者輸入 (例如提醒使用者必須填寫特定項目的表單)，或是在使用者選取部分文字時，變更文字欄位內所強調的文字子區段。

下列範例會在字元範圍上使用 TextField.setTextFormat()，讓使用者按一下文字欄位時變更 myTextField 部分內容的外觀：

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL CAPS changes format.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, changeText);

var myformat:TextFormat = new TextFormat();
myformat.color = 0xFF0000;
myformat.size = 18;
myformat.underline = true;

function changeText(event:MouseEvent):void
{
    myTextField.setTextFormat(myformat, 49, 65);
}
```

進階文字顯示

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 在 flash.text 套件中提供了多種可控制顯示文字屬性的類別 (包含內嵌字體、消除鋸齒設定、Alpha 色版控制和其他特定設定)。「Adobe® Flash® Professional CS5 的 ActionScript® 3.0 參考」提供了這些類別和屬性 (包含 CSMSettings、Font 和 TextRenderer 類別) 的詳細說明。

使用內嵌字體

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您在應用程式中指定 TextField 的特定字體時，Flash Player 或 AIR 會尋找名稱相同的裝置字體 (位在使用者電腦上的字體)。如果在系統上找不到該字體，或使用者的同名字體版本稍有不同，則顯示的文字與您所預期的會有極大的差異。根據預設，文字會以 Times Roman 字體顯示。

若要確定使用者看到的是正確的字體，您可以將該字體內嵌至應用程式 SWF 檔。內嵌字體具有許多優點：

- 內嵌字體字元會消除鋸齒，它們的邊緣平滑，特別是針對較大的文字。
- 您可以旋轉使用內嵌字體的文字。
- 內嵌字體文字可以設定為透明或半透明。
- 您可以搭配使用 kerning CSS 樣式與內嵌字體。

使用內嵌字體的最大限制是會增加應用程式的檔案大小或下載大小。

將字體檔內嵌至應用程式 SWF 檔的正確方法會隨著開發環境而有所不同。

內嵌字體之後，您就可以確定 TextField 使用的是正確的內嵌字體：

- 將 TextField 的 embedFonts 屬性設為 true。
- 建立 TextFormat 物件，然後將其 fontFamily 屬性設為內嵌字體的名稱，並將 TextFormat 物件套用至 TextField。指定內嵌字體時，fontFamily 屬性只能包含單一名稱；它無法使用以逗號分隔的多個字體名稱清單。
- 如果使用 CSS 樣式設定 TextField 或元件的字體，請將 font-family CSS 屬性設為內嵌字體的名稱。如果您想要指定內嵌字體，則 font-family 屬性必須包含單一名稱，而不是名稱清單。

在 **Flash** 中內嵌字體

Flash Professional 可讓您內嵌幾乎系統上所安裝的所有字體，包含 TrueType 字體和 Type 1 Postscript 字體。

您可以透過多種方式將字體內嵌至應用程式，包含：

- 在「舞台」上設定 TextField 的字體和樣式屬性，以及按一下「內嵌字體」核取方塊
- 建立和參考字體元件
- 建立和使用含有內嵌字體元件的執行時期共享元件庫

如需有關如何將字體內嵌至應用程式的詳細資訊，請參閱「使用 Flash」中的「動態或輸入文字欄位的內嵌字體」。

在 **Flex** 中內嵌字體

您可以透過多種方式將字體內嵌至 Flex 應用程式，包含：

- 在指令碼中使用 [Embed] 中繼資料標籤
- 使用 @font-face 樣式宣告
- 為字型建立一個類別，並使用 [Embed] 標籤內嵌該類別。

您只能在 Flex 應用程式中直接內嵌 TrueType 字體。其他格式的字體 (例如 Type 1 Postscript 字體) 可以使用 Flash Professional 先內嵌至 SWF 檔，然後再於 Flex 應用程式中使用該 SWF 檔。如需有關在 Flex 中使用來自 SWF 檔內嵌字體的詳細資訊，請參閱 Using Flex 4 中的 “Embedding fonts from SWF files”。

更多說明主題

一致文字外觀的內嵌字體

Peter deHaan：內嵌字體

Divillysausages.com：AS3 字體內嵌主類別

控制銳利度、粗細和消除鋸齒

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 或 AIR 預設時便會決定文字顯示控制的設定 (例如調整文字大小、變更顏色，或顯示於各種背景時的銳利度、粗細和消除鋸齒)。在某些情況下 (例如，如果文字太小或太大，或文字位在不同的背景上)，您可能會想要保有控制這些設定的權限。您可以使用 flash.text.TextRenderer 類別和其相關類別 (例如 CSMSettings 類別)，來覆寫 Flash Player 或 AIR 設定。這些類別可讓您精確控制內嵌文字的顯示品質。如需有關內嵌字體的詳細資訊，請參閱第 324 頁「使用內嵌字體」。

備註：flash.text.TextField.antiAliasType 屬性必須有值 AntiAliasType.ADVANCED，才能讓您設定銳利度、粗細或 gridFitType 屬性，或使用 TextRenderer.setAdvancedAntiAliasingTable() 方法。

下列範例會使用內嵌字體 myFont，將自訂連續筆劃調整 (CSM) 屬性和格式套用至顯示的文字。使用者按一下顯示的文字時，Flash Player 或 Adobe AIR 會套用自訂設定：

```
var format:TextFormat = new TextFormat();
format.color = 0x336699;
format.size = 48;
format.font = "myFont";

var myText:TextField = new TextField();
myText.embedFonts = true;
myText.autoSize = TextFieldAutoSize.LEFT;
myText.antiAliasType = AntiAliasType.ADVANCED;
myText.defaultTextFormat = format;
myText.selectable = false;
myText.mouseEnabled = true;
myText.text = "Hello World";
addChild(myText);
myText.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:Event):void
{
    var myAntiAliasSettings = new CSMSettings(48, 0.8, -0.8);
    var myAliasTable:Array = new Array(myAntiAliasSettings);
    TextRenderer.setAdvancedAntiAliasingTable("myFont", FontStyle.ITALIC, TextColorType.DARK_COLOR,
myAliasTable);
}
```

使用靜態文字

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

靜態文字只能在 Flash Professional 中建立。您無法使用 ActionScript 透過程式設計方式實體化靜態文字。靜態文字適用於較短且不會變更的文字 (動態文字會變更)。您可將靜態文字視為類似 Flash Professional 中 「舞台」所繪製的圓形或方形圖形元素。雖然靜態文字的限制多於動態文字，但是 ActionScript 3.0 可讓您使用 StaticText 類別來讀取靜態文字的屬性值。您也可以使用 TextSnapshot 類別來讀取靜態文字的值。

使用 StaticText 類別存取靜態文字欄位

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

一般而言，您會在 Flash Professional 的「動作」面板中，使用 flash.text.StaticText 類別與放在「舞台」的靜態文字實體進行互動。您也可以使用 ActionScript 檔來與含有靜態文字的 SWF 檔互動。在任一種情況下，您都無法透過程式設計方式來實體化靜態文字實體。靜態文字在 Flash Professional 中建立。

若要建立現有靜態文字欄位的參考，請重複執行顯示清單中的項目，並指定變數。例如：

```
for (var i = 0; i < this.numChildren; i++) {
var displayitem:DisplayObject = this.getChildAt(i);
if (displayitem instanceof StaticText) {
trace("a static text field is item " + i + " on the display list");
        var myFieldLabel:StaticText = StaticText(displayitem);
        trace("and contains the text: " + myFieldLabel.text);
}
}
```

參考靜態文字欄位之後，您就可以在 ActionScript 3.0 中使用該欄位的屬性。下列程式碼會附加至 「時間軸」中的影格，並假設變數 myFieldLabel 已指定給靜態文字參考。動態文字欄位 myField 位於與 myFieldLabel 的 x 和 y 值相對的位置，並會再次顯示 myFieldLabel 的值。

```
var myField:TextField = new TextField();
addChild(myField);
myField.x = myFieldLabel.x;
myField.y = myFieldLabel.y + 20;
myField.autoSize = TextFieldAutoSize.LEFT;
myField.text = "and " + myFieldLabel.text
```

使用 TextSnapshot 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如果您想要透過程式設計方式使用現有靜態文字實體，則可以使用 flash.text.TextSnapshot 類別來搭配 flash.display.DisplayObjectContainer 的 textSnapshot 屬性。換句話說，您將透過 DisplayObjectContainer.textSnapshot 屬性來建立 TextSnapshot 實體。之後，您可以將方法套用至該實體以擷取值，或選取部分的靜態文字。

例如，將含有文字 「TextSnapshot Example」的靜態文字欄位放在 「舞台」上。請將下列 ActionScript 新增至 「時間軸」的影格 1：

```
var mySnap:TextSnapshot = this.textSnapshot;
var count:Number = mySnap.charCount;
mySnap.setSelected(0, 4, true);
mySnap.setSelected(1, 2, false);
var myText:String = mySnap.getSelectedText(false);
trace(myText);
```

如果您想要使用已載入 SWF 檔之靜態文字欄位中的文字做為應用程式另一個部分的值，則 TextSnapshot 類別會非常實用。

TextField 範例：報紙樣式文字格式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

新聞版面範例會格式化文字，使其外觀看起來就像報紙上的新聞報導一樣。輸入文字可以包含報導的標題、副標題和內文。指定顯示寬度和高度之後，此新聞版面範例就會格式化標題和副標題，以取得顯示區域的完整寬度。新聞報導的內容文字會分佈在兩欄或更多欄中。

此範例會說明下列 ActionScript 程式設計的技巧：

- 擴充 TextField 類別
- 載入和套用外部 CSS 檔
- 將 CSS 樣式轉換為 TextFormat 物件
- 使用 TextLineMetrics 類別取得文字顯示大小的資訊

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。您可以在 Samples/NewsLayout 的資料夾中找到新聞版面應用程式檔。此應用程式是由下列檔案組成：

檔案	說明
NewsLayout.mxml 或 NewsLayout.fla	應用程式使用者介面，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/newslayout/StoryLayoutComponent.as	放置 StoryLayout 實體的 Flex UIComponent 類別。
com/example/programmingas3/newslayout/StoryLayout.as	主要 ActionScript 類別，可排列新聞報導的所有元件以進行顯示。
com/example/programmingas3/newslayout/FormattedTextField.as	TextField 類別的子類別，用以管理自己的 TextFormat 物件。
com/example/programmingas3/newslayout/HeadlineTextField.as	FormattedTextField 類別的子類別，用以調整字體大小使其符合所需的寬度。
com/example/programmingas3/newslayout/MultiColumnTextField.as	ActionScript 類別，用以將文字分為兩欄或更多欄。
story.css	定義版面文字樣式的 CSS 檔。

讀取外部 CSS 檔

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

新聞版面應用程式會先讀取本機 XML 檔中的報導文字。然後再讀取外部 CSS 檔，該檔案會提供標題、副標題和主要文字內容的格式資訊。

CSS 檔案定義了三種樣式，即報導的標準段落樣式、標題的 h1 樣式及副標題的 h2 樣式。

```
p {
    font-family: Georgia, "Times New Roman", Times, _serif;
    font-size: 12;
    leading: 2;
    text-align: justify;
    indent: 24;
}

h1 {
    font-family: Verdana, Arial, Helvetica, _sans;
    font-size: 20;
    font-weight: bold;
    color: #000099;
    text-align: left;
}

h2 {
    font-family: Verdana, Arial, Helvetica, _sans;
    font-size: 16;
    font-weight: normal;
    text-align: left;
}
```

用來讀取外部 CSS 檔的技術，與第 322 頁「載入外部 CSS 檔」中所述的技術相同。載入 CSS 檔之後，應用程式會執行 onCSSFileLoaded() 方法 (如下所示)。

```
public function onCSSFileLoaded(event:Event):void
{
    this.sheet = new StyleSheet();
    this.sheet.parseCSS(loader.data);

    h1Format = getTextStyle("h1", this.sheet);
    if (h1Format == null)
    {
        h1Format = getDefaultHeadFormat();
    }
    h2Format = getTextStyle("h2", this.sheet);
    if (h2Format == null)
    {
        h2Format = getDefaultHeadFormat();
        h2Format.size = 16;
    }
    pFormat = getTextStyle("p", this.sheet);
    if (pFormat == null)
    {
        pFormat = getDefaultTextFormat();
        pFormat.size = 12;
    }
    displayText();
}
```

onCSSFileLoaded() 方法會建立 StyleSheet 物件，並讓該物件剖析輸入 CSS 資料。報導的主要文字內容會顯示在 MultiColumnTextField 物件中，而此物件可直接使用 StyleSheet 物件。不過，標題欄位會使用 HeadlineTextField 類別，而此類別會使用 TextFormat 物件做為其格式。

onCSSFileLoaded() 方法會呼叫 getTextStyle() 方法兩次，以便將 CSS 樣式宣告轉換為 TextFormat 物件，進而分別與這兩個 HeadlineTextField 物件搭配使用。

```
public function getTextStyle(styleName:String, ss:StyleSheet):TextFormat
{
    var format:TextFormat = null;

    var style:Object = ss.getStyle(styleName);
    if (style != null)
    {
        var colorStr:String = style.color;
        if (colorStr != null && colorStr.indexOf("#") == 0)
        {
            style.color = colorStr.substr(1);
        }
        format = new TextFormat(style.fontFamily,
                        style.fontSize,
                        style.color,
                        (style.fontWeight == "bold"),
                        (style.fontStyle == "italic"),
                        (style.textDecoration == "underline"),
                        style.url,
                        style.target,
                        style.textAlign,
                        style.marginLeft,
                        style.marginRight,
                        style.indent,
                        style.leading);

        if (style.hasOwnProperty("letterSpacing"))
        {
            format.letterSpacing = style.letterSpacing;
        }
    }
    return format;
}
```

屬性名稱與屬性值的意義在 CSS 樣式宣告與 TextFormat 物件之間會有所不同。getTextStyle() 方法會將 CSS 屬性值轉換為 TextFormat 物件預期的值。

在頁面上排列報導元素

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

StoryLayout 類別會格式化標題、副標題和主要文字內容欄位，並將它們排列為報紙的樣式。displayText() 方法一開始會建立並放置各種欄位。

```
public function displayText():void
{
    headlineTxt = new HeadlineTextField(h1Format);
    headlineTxt.wordWrap = true;
    headlineTxt.x = this.paddingLeft;
    headlineTxt.y = this.paddingTop;
    headlineTxt.width = this.preferredWidth;
    this.addChild(headlineTxt);

    headlineTxt.fitText(this.headline, 1, true);

    subtitleTxt = new HeadlineTextField(h2Format);
    subtitleTxt.wordWrap = true;
    subtitleTxt.x = this.paddingLeft;
    subtitleTxt.y = headlineTxt.y + headlineTxt.height;
    subtitleTxt.width = this.preferredWidth;
    this.addChild(subtitleTxt);

    subtitleTxt.fitText(this.subtitle, 2, false);

    storyTxt = new MultiColumnText(this.numColumns, 20,
                        this.preferredWidth, 400, true, this.pFormat);
    storyTxt.x = this.paddingLeft;
    storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
    this.addChild(storyTxt);

    storyTxt.text = this.content;
...
```

藉由將每個欄位的 y 屬性設定為等於前一個欄位的 y 屬性再加上高度，即可讓欄位放置於前一個欄位的下方。而因為 HeadlineTextField 物件和 MultiColumnTextField 物件可以變更其高度以放入內容，所以需要此動態置入方式計算。

改變字體大小以符合欄位大小

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

指定寬度 (像素) 以及要顯示的行數上限之後，HeadlineTextField 會改變字體大小，讓文字符合欄位的大小。如果文字很短，字體就會變大，而建立出 Tabloid 樣式標題。如果文字很長，字體就會變小。

下面顯示的 HeadlineTextField.fitText() 方法會進行字體大小的調整工作：

```
public function fitText(msg:String, maxLines:uint = 1, toUpper:Boolean = false, targetWidth:Number = -
1):uint
{
    this.text = toUpper ? msg.toUpperCase() : msg;

    if (targetWidth == -1)
    {
        targetWidth = this.width;
    }

    var pixelsPerChar:Number = targetWidth / msg.length;

    var pointSize:Number = Math.min(MAX_POINT_SIZE, Math.round(pixelsPerChar * 1.8 * maxLines));

    if (pointSize < 6)
    {
        // the point size is too small
        return pointSize;
    }
```

```
    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(--pointSize, maxLines);
    }
    else
    {
        return growText(pointSize, maxLines);
    }
}

public function growText(pointSize:Number, maxLines:uint = 1):Number
{
    if (pointSize >= MAX_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize + 1);

    if (this.numLines > maxLines)
    {
        // set it back to the last size
        this.changeSize(pointSize);
        return pointSize;
    }
    else
    {
        return growText(pointSize + 1, maxLines);
    }
}

public function shrinkText(pointSize:Number, maxLines:uint=1):Number
{
    if (pointSize <= MIN_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(pointSize - 1, maxLines);
    }
    else
    {
        return pointSize;
    }
}
```

HeadlineTextField.fitText() 方法會使用簡單的遞歸技術來調整字體的大小。它會先猜測文字中每個字元的平均像素數，並根據此數值來計算起點大小。然後會變更字體大小，並檢查文字是否因進行文字繞排，而導致建立的行數超過文字行上限。如果行數太多，則它會呼叫 shrinkText() 方法縮小字體，然後再試一次。如果行數不多，它會呼叫 growText() 方法放大字體，然後再試一次。當再放大 1 點的字體會建立太多的文字行時，此程序就會停止。

讓文字分佈在多欄

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

MultiColumnTextField 類別會將文字分散到多個 TextField 物件，而這些物件會排列成類似報紙欄位的版面。

MultiColumnTextField() 建構函式會先建立 TextField 物件陣列 (一個直欄一個物件)，如下所示：

```
for (var i:int = 0; i < cols; i++)
{
    var field:TextField = new TextField();
    field.multiline = true;
    field.autoSize = TextFieldAutoSize.NONE;
    field.wordWrap = true;
    field.width = this.colWidth;
    field.setTextFormat(this.format);
    this.fieldArray.push(field);
    this.addChild(field);
}
```

使用 addChild() 方法將每個 TextField 物件新增至陣列，並新增至顯示清單。

只要 StoryLayout text 屬性或 styleSheet 屬性一變更，就會呼叫 layoutColumns() 方法來重新顯示文字。layoutColumns() 方法會呼叫 getOptimalHeight() 方法，找出可將所有文字放入指定版面寬度所需的正確像素高度。

```
public function getOptimalHeight(str:String):int
{
    if (field.text == "" || field.text == null)
    {
        return this.preferredHeight;
    }
    else
    {
        this.linesPerCol = Math.ceil(field.numLines / this.numColumns);

        var metrics:TextLineMetrics = field.getLineMetrics(0);
        this.lineHeight = metrics.height;
        var prefHeight:int = linesPerCol * this.lineHeight;

        return prefHeight + 4;
    }
}
```

首先，getOptimalHeight() 方法會計算每個直欄的寬度。然後再設定陣列中第一個 TextField 物件的寬度和 htmlText 屬性。getOptimalHeight() 方法會使用第一個 TextField 物件來找出文字中單字繞排的總行數，並據此識別每個直欄中的行數。接著會呼叫 TextField.getLineMetrics() 方法來擷取 TextLineMetrics 物件，而此物件包含第一行中文字大小的詳細資訊。TextLineMetrics.height 屬性表示文字行的完整高度 (像素)，包含上緣、下緣和行距。因此，MultiColumnTextField 物件的最佳高度為行高乘以每直欄行數，再加上 4 (這是 TextField 物件頂端和底端粗細各為 2 像素的邊框)。

以下是完整 layoutColumns() 方法的程式碼：

```
public function layoutColumns():void
{
    if (this._text == "" || this._text == null)
    {
        return;
    }

    var field:TextField = fieldArray[0] as TextField;
    field.text = this._text;
    field.setTextFormat(this.format);

    this.preferredHeight = this.getOptimalHeight(field);

    var remainder:String = this._text;
    var fieldText:String = "";
    var lastLineEndedPara:Boolean = true;

    var indent:Number = this.format.indent as Number;

    for (var i:int = 0; i < fieldArray.length; i++)
    {
        field = this.fieldArray[i] as TextField;

        field.height = this.preferredHeight;
        field.text = remainder;

        field.setTextFormat(this.format);

        var lineLen:int;
        if (indent > 0 && !lastLineEndedPara && field.numLines > 0)
        {
            lineLen = field.getLineLength(0);
            if (lineLen > 0)
            {
                field.setTextFormat(this.firstLineFormat, 0, lineLen);
                }
            }

        field.x = i * (colWidth + gutter);
        field.y = 0;

        remainder = "";
        fieldText = "";

        var linesRemaining:int = field.numLines;
        var linesVisible:int = Math.min(this.linesPerCol, linesRemaining);

        for (var j:int = 0; j < linesRemaining; j++)
        {
            if (j < linesVisible)
            {
                fieldText += field.getLineText(j);
            }
            else
            {
                remainder +=field.getLineText(j);
            }
        }

        field.text = fieldText;

        field.setTextFormat(this.format);
```

```
        if (indent > 0 && !lastLineEndedPara)
        {
            lineLen = field.getLineLength(0);
            if (lineLen > 0)
            {
                field.setTextFormat(this.firstLineFormat, 0, lineLen);
            }
        }

        var lastLine:String = field.getLineText(field.numLines - 1);
        var lastCharCode:Number = lastLine.charCodeAt(lastLine.length - 1);

        if (lastCharCode == 10 || lastCharCode == 13)
        {
        lastLineEndedPara = true;
        }
        else
        {
        lastLineEndedPara = false;
        }

        if ((this.format.align == TextFormatAlign.JUSTIFY) &&
                (i < fieldArray.length - 1))
        {
        if (!lastLineEndedPara)
        {
            justifyLastLine(field, lastLine);
        }
    }
    }
}
```

呼叫 getOptimalHeight() 方法以設定 preferredHeight 屬性之後，layoutColumns() 方法會重複執行 TextField 物件，將每個物件的高度都設為 preferredHeight 值。layoutColumns() 方法接著會將剛好的文字行數分佈至每個欄位，這樣就不需要捲動任何個別欄位，而且每個連續欄位中文字的開始位置就是前一個欄位中文字的結束位置。如果文字對齊樣式已設為「齊行」，則會呼叫 justifyLastLine() 方法來對齊欄位中的最後一行文字。否則，最後一行會視為段落結尾，而不會對齊字行。

第 22 章 使用 Flash 文字引擎

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

自 Flash Player 10 和 Adobe® AIR™1.5 開始提供的 Adobe® Flash® 文字引擎 (FTE)，可精確控制文字規格、格式化和雙向文字的低階支援。它可以改善文字排列方式，並增強語言支援。雖然 FTE 可以用來建立及管理簡單的文字元素，但是它的主要目的是做為開發人員建立文字處理組件的基礎。因此，Flash 文字引擎採用了更進階的程式設計開發技術。若要顯示簡單文字元素，請參閱第 315 頁「使用 TextField 類別」。

Text Layout Framework 內含以 FTE 為基礎的文字處理組件，可讓您更輕易地使用其進階功能。Text Layout Framework 是完全以 ActionScript 3.0 建立的可延伸元件庫。您可以使用現有的 TLF 元件，或是使用此架構來建立您自己的文字元件。如需詳細資訊，請參閱第 362 頁「使用 Text Layout Framework」。

更多說明主題

flash.text.engine 套件

建立和顯示文字

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

構成 Flash 文字引擎的類別可讓您建立、格式化和控制文字。下列類別是使用 Flash 文字引擎建立和顯示文字的基本建構區塊：

- TextElement/GraphicElement/GroupElement - 包含 TextBlock 實體的內容
- ElementFormat - 指定 TextBlock 實體內容的格式特質
- TextBlock - 建立文字段落的 Factory
- TextLine - 透過 TextBlock 建立的文字行

若要顯示文字，請從 String 建立 TextElement 物件，並使用 ElementFormat 物件來指定格式特性。指定 TextElement 給 TextBlock 物件的 content 屬性。呼叫 TextBlock.createTextLine() 方法來建立要顯示的文字行。createTextLine() 方法會傳回一個 TextLine 物件，內含指定寬度可容納的最長字串。重複呼叫此方法，直到整個字串格式化為文字行。當不再需要建立文字行時，TextBlock 物件的 textLineCreationResult 屬性會指定值：TextLineCreationResult.COMPLETE。若要顯示文字行，請將它們新增到顯示清單 (並使用適當的 x 和 y 位置值)。

例如，下列程式碼使用這些 FTE 類別來顯示 "Hello World! This is Flash Text Engine!" (使用預設格式和字體值)。在此簡單的範例中，只會建立一行文字。

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class HelloWorldExample extends Sprite
    {
        public function HelloWorldExample()
        {
            var str = "Hello World! This is Flash Text Engine!";
            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;

            var textLine1:TextLine = textBlock.createTextLine(null, 300);
            addChild(textLine1);
            textLine1.x = 30;
            textLine1.y = 30;
        }
    }
}
```

createTextLine() 的參數會指定新的字行要從哪一個字行開始，以及字行的寬度 (像素)。開始新行的字行通常是前一行，但如果是第一行，則為 null。

新增 GraphicElement 和 GroupElement 物件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

您可以將 GraphicElement 物件指定給 TextBlock 物件，以顯示影像或圖形元素。從圖形或影像建立 GraphicElement 類別實體，並將該實體指定給 TextBlock.content 屬性。接著以一般方式呼叫 TextBlock.createTextline() 以建立文字行。下列範例會建立兩行文字行，其中一行含有 GraphicElement 物件，另一行含有 TextElement 物件。

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GraphicElementExample extends Sprite
    {
        public function GraphicElementExample()
        {
            var str:String = "Beware of Dog!";

            var triangle:Shape = new Shape();
            triangle.graphics.beginFill(0xFF0000, 1);
            triangle.graphics.lineStyle(3);
            triangle.graphics.moveTo(30, 0);
            triangle.graphics.lineTo(60, 50);
            triangle.graphics.lineTo(0, 50);
            triangle.graphics.lineTo(30, 0);
            triangle.graphics.endFill();

            var format:ElementFormat = new ElementFormat();
```

```
            format.fontSize = 20;

            var graphicElement:GraphicElement = new GraphicElement(triangle, triangle.width,
triangle.height, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = graphicElement;
            var textLine1:TextLine = textBlock.createTextLine(null, triangle.width);
            textLine1.x = 50;
            textLine1.y = 110;
            addChild(textLine1);

            var textElement:TextElement = new TextElement(str, format);
            textBlock.content = textElement;
            var textLine2 = textBlock.createTextLine(null, 300);
            addChild(textLine2);
            textLine2.x = textLine1.x - 30;
            textLine2.y = textLine1.y + 15;
        }
    }
}
```

您可以建立 GroupElement 物件來建立一組 TextElement、GraphicElement 和其他 GroupElement 物件。GroupElement 可指定給 TextBlock 物件的 content 屬性。GroupElement() 建構函式的參數是 Vector，可指向組成群組的文字、圖形和群組元素。下列範例會將兩個圖形元素和一個文字元素組成群組，並將它們指定為文字區塊的一個單位。

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GroupElementExample extends Sprite
    {
        public function GroupElementExample()
        {
            var str:String = "Beware of Alligators!";

            var triangle1:Shape = new Shape();
            triangle1.graphics.beginFill(0xFF0000, 1);
            triangle1.graphics.lineStyle(3);
            triangle1.graphics.moveTo(30, 0);
            triangle1.graphics.lineTo(60, 50);
            triangle1.graphics.lineTo(0, 50);
            triangle1.graphics.lineTo(30, 0);
            triangle1.graphics.endFill();

            var triangle2:Shape = new Shape();
            triangle2.graphics.beginFill(0xFF0000, 1);
            triangle2.graphics.lineStyle(3);
            triangle2.graphics.moveTo(30, 0);
            triangle2.graphics.lineTo(60, 50);
            triangle2.graphics.lineTo(0, 50);
            triangle2.graphics.lineTo(30, 0);
            triangle2.graphics.endFill();
```

```
            var format:ElementFormat = new ElementFormat();
            format.fontSize = 20;
            var graphicElement1:GraphicElement = new GraphicElement(triangle1, triangle1.width,
triangle1.height, format);
            var textElement:TextElement = new TextElement(str, format);
            var graphicElement2:GraphicElement = new GraphicElement(triangle2, triangle2.width,
triangle2.height, format);
            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>();
            groupVector.push(graphicElement1, textElement, graphicElement2);
            var groupElement = new GroupElement(groupVector);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = groupElement;
            var textLine:TextLine = textBlock.createTextLine(null, 800);
            addChild(textLine);
            textLine.x = 100;
            textLine.y = 200;
        }
    }
}
```

取代文字

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

呼叫 TextElement.replaceText() 以取代在 TextElement 中指定給 TextBlock.content 屬性的文字，即可取代 TextBlock 實體中的文字。

下列範例使用 replaceText()，先將文字插入行首，接著將文字附加至行尾，最後取代行中的文字。

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class ReplaceTextExample extends Sprite
    {
        public function ReplaceTextExample()
        {

            var str:String = "Lorem ipsum dolor sit amet";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription);
            format.fontSize = 14;
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;
            createLine(textBlock, 10);
            textElement.replaceText(0, 0, "A text fragment: ");
            createLine(textBlock, 30);
            textElement.replaceText(43, 43, "...");
            createLine(textBlock, 50);
            textElement.replaceText(23, 28, "(ipsum)");
            createLine(textBlock, 70);
        }

        function createLine(textBlock:TextBlock, y:Number):void {
            var textLine:TextLine = textBlock.createTextLine(null, 300);
            textLine.x = 10;
            textLine.y = y;
            addChild(textLine);
        }
    }
}
```

replaceText() 方法會以 newText 參數所指定的文字來取代 beginIndex 和 endIndex 參數所指定的文字。如果 beginIndex 和 endIndex 參數的值相同，則 replaceText() 會將指定的文字插入該位置。否則，它會以新文字取代 beginIndex 和 endIndex 所指定的字元

處理 FTE 中的事件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

您可以將事件偵聽程式新增至 TextLine 實體，就如同處理其他顯示物件一樣。例如，您可以偵測使用者何時將滑鼠指向文字行上方，或使用者何時按一下該行。下列範例將會偵測這兩個事件。當您將滑鼠指向該行上方時，游標會變成按鈕游標，而當您按一下該行時，游標會改變顏色。

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventHandlerExample extends Sprite
    {
        var textBlock:TextBlock = new TextBlock();

        public function EventHandlerExample():void
        {
            var str:String = "I'll change color if you click me.";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription, 18);
            var textElement = new TextElement(str, format);
            textBlock.content = textElement;
            createLine(textBlock);
        }

        private function createLine(textBlock:TextBlock):void
        {
            var textLine:TextLine = textBlock.createTextLine(null, 500);
            textLine.x = 30;
            textLine.y = 30;
            addChild(textLine);
            textLine.addEventListener("mouseOut", mouseOutHandler);
            textLine.addEventListener("mouseOver", mouseOverHandler);
            textLine.addEventListener("click", clickHandler);
        }

        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
            Mouse.cursor = "arrow";
        }

        function clickHandler(event:MouseEvent):void {
            if(textBlock.firstLine)
                removeChild(textBlock.firstLine);
            var newFormat:ElementFormat = textBlock.content.elementFormat.clone();
```

```
            switch(newFormat.color)
            {
                case 0x000000:
                    newFormat.color = 0xFF0000;
                    break;
                case 0xFF0000:
                    newFormat.color = 0x00FF00;
                    break;
                case 0x00FF00:
                    newFormat.color = 0x0000FF;
                    break;
                case 0x0000FF:
                    newFormat.color = 0x000000;
                    break;
            }
            textBlock.content.elementFormat = newFormat;
            createLine(textBlock);
        }
    }
}
```

鏡像事件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

您也可以在將事件發送器上，建立文字區塊或一部分文字區塊的事件鏡像。首先，建立 EventDispatcher 實體，然後將它指定給 TextElement 實體的 eventMirror 屬性。如果文字區塊包含單一文字元素，則此文字引擎會鏡像整個文字區塊的事件。如果文字區塊包含多個文字元素，則此文字引擎只會鏡像已設定 eventMirror 屬性的 TextElement 實體事件。下列範例中的文字包含三個元素：單字 "Click"、單字 "here" 和字串 "to see me in italic"。此範例會將事件發送器指定給第二個文字元素「here」，並新增事件偵聽程式 (clickHandler() 方法)。clickHandler() 方法會將文字變更為斜體。它也會以「Click here to see me in normal font!」取代第三個文字元素的內容。

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventMirrorExample extends Sprite
    {
        var fontDescription:FontDescription = new FontDescription("Helvetica", "bold");
        var format:ElementFormat = new ElementFormat(fontDescription, 18);
        var textElement1 = new TextElement("Click ", format);
        var textElement2 = new TextElement("here ", format);
        var textElement3 = new TextElement("to see me in italic! ", format);
        var textBlock:TextBlock = new TextBlock();

        public function EventMirrorExample()
        {
            var myEvent:EventDispatcher = new EventDispatcher();

            myEvent.addEventListener("click", clickHandler);
            myEvent.addEventListener("mouseOut", mouseOutHandler);
            myEvent.addEventListener("mouseOver", mouseOverHandler);

            textElement2.eventMirror=myEvent;

            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>;
```

```
            groupVector.push(textElement1, textElement2, textElement3);
            var groupElement:GroupElement = new GroupElement(groupVector);

            textBlock.content = groupElement;
            createLines(textBlock);
        }

        private function clickHandler(event:MouseEvent):void
        {
            var newFont:FontDescription = new FontDescription();
            newFont.fontWeight = "bold";

            var newFormat:ElementFormat = new ElementFormat();
            newFormat.fontSize = 18;
            if(textElement3.text == "to see me in italic! ") {
                newFont.fontPosture = FontPosture.ITALIC;
                textElement3.replaceText(0,21, "to see me in normal font! ");
            }
            else {
                newFont.fontPosture = FontPosture.NORMAL;
                textElement3.replaceText(0, 26, "to see me in italic! ");
            }
            newFormat.fontDescription = newFont;
            textElement1.elementFormat = newFormat;
            textElement2.elementFormat = newFormat;
            textElement3.elementFormat = newFormat;
            createLines(textBlock);
        }

        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
                Mouse.cursor = "arrow";
        }

        private function createLines(textBlock:TextBlock):void
        {
            if(textBlock.firstLine)
                removeChild (textBlock.firstLine);
            var textLine:TextLine = textBlock.createTextLine (null, 300);
            textLine.x = 15;
            textLine.y = 20;
            addChild (textLine);
        }
    }
}
```

mouseOverHandler() 和 mouseOutHandler() 函數會在游標停留在「here」這個字上方時將游標設為按鈕游標，而在離開時設回箭號。

格式化文字

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

TextBlock 物件是建立文字行的 Factory。您可以透過 TextElement 物件指定 TextBlock 的內容。ElementFormat 物件會處理文字的格式化作業。ElementFormat 類別定義的屬性包括：基線對齊、特殊字距、字距調整、文字旋轉以及字體大小、顏色和大小寫。這個類別也包含 FontDescription (會在第 347 頁「使用字體」中詳細說明)。

使用 ElementFormat 物件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

ElementFormat 物件的建構函式可以使用一長串選擇性參數清單中的任何一個 (包含 FontDescription)。您也可以在建構函式外部設定這些屬性。下列範例會顯示用來定義和顯示簡單文字行時，各種物件之間的關係：

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function ElementFormatExample()
        {
            fd.fontName = "Garamond";
            ef = new ElementFormat(fd);
            ef.fontSize = 30;
            ef.color = 0xFF0000;
            str = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

字體顏色和透明度 (Alpha)

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

ElementFormat 物件的 color 屬性可設定字體顏色。此值是整數，代表顏色的 RGB 元件；例如，0xFF0000 表示紅色，而 0x00FF00 表示綠色。預設值是黑色 (0x000000)。

alpha 屬性可設定元素 (TextElement 和 GraphicElement) 的 Alpha 透明度值。值的範圍可以從 0 (完全透明) 到 1 (完全不透明，此為預設值)。您將看不見 alpha 為 0 的元素，但該元素仍在作用中。此值會乘以任何繼承的 Alpha 值，因此會讓元素更加透明。

```
var ef:ElementFormat = new ElementFormat();
ef.alpha = 0.8;
ef.color = 0x999999;
```

基線對齊和位移

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

文字行中最大文字的字體和大小會決定文字行的主基線。您可以設定 TextBlock.baselineFontDescription 和 TextBlock.baselineFontSize 來覆寫這些值。您可以將主基線對齊文字內的其中一條基線。這包含上緣基線和下緣基線，或是表意字上基線、表意字中央基線或表意字下基線。

A. 上緣 **B.** 基線 **C.** 下緣 **D.** x- 高度

在 ElementFormat 物件中，三個屬性可決定基線和對齊特性。alignmentBaseline 屬性會設定 TextElement 或 GraphicElement 的主要基線。此基線是元素的 「對齊」線，而且所有文字的主基線都會對齊這條基線的位置。

dominantBaseline 屬性可指定要使用元素的哪一條基線，藉此決定該文字行上元素的垂直位置。預設值是 TextBaseline.ROMAN，但也可以設定讓 IDEOGRAPHIC_TOP 或 IDEOGRAPHIC_BOTTOM 基線成為主基線。

baselineShift 屬性會將基線依指定的像素數沿著 y 軸移動。在一般 (非旋轉) 文字中，正值會將基線往下移動，而負值會將基線往上移動。

印刷大小寫

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

ElementFormat 的 TypographicCase 屬性指定文字大小寫 (例如大寫、小寫 或小型大寫字)。

```
var ef_Upper:ElementFormat = new ElementFormat();
ef_Upper.typographicCase = TypographicCase.UPPERCASE;

var ef_SmallCaps:ElementFormat = new ElementFormat();
ef_SmallCaps.typographicCase = TypographicCase.SMALL_CAPS;
```

旋轉文字

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

您可以用 90 度的倍數，來旋轉文字區塊或文字區段內的字符。TextRotation 類別會定義下列用來設定文字區塊和字符旋轉的常數：

常數	值	說明
AUTO	"auto"	指定 90 度逆時針旋轉。通常與垂直的亞洲文字搭配使用，只會旋轉需要旋轉的字符。
ROTATE_0	"rotate_0"	指定不進行任何旋轉。
ROTATE_180	"rotate_180"	指定 180 度旋轉。
ROTATE_270	"rotate_270"	指定 270 度旋轉。
ROTATE_90	"rotate_90"	指定 90 度順時針旋轉。

若要旋轉文字區塊中的文字行，則請先設定 TextBlock.lineRotation 屬性，然後再呼叫 TextBlock.createTextLine() 方法來建立文字行。

若要旋轉文字區塊或區段內的字符，請將 ElementFormat.textRotation 屬性設為您希望字符旋轉的度數。字符是組成字元的形狀，或包含多個字符之字元的一部分。例如，字母「a」和「i」上的點都是字符。

旋轉字符適用於部分亞洲語言，在這些語言中，您往往需要將文字行旋轉為垂直方向，但並不要旋轉文字行內的字元。如需有關旋轉亞洲文字的詳細資訊，請參閱第 350 頁「對齊東亞文字」。

此下是旋轉文字區塊和其內字符的範例，也就是旋轉亞洲文字的方法。此範例也會使用日文字體：

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class RotationExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function RotationExample()
        {
            fd.fontName = "MS Mincho";
            ef = new ElementFormat(fd);
            ef.textRotation = TextRotation.AUTO;
            str = "This is rotated Japanese text";
            te = new TextElement(str, ef);
            tb.lineRotation = TextRotation.ROTATE_90;
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

鎖定和仿製 ElementFormat

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

將 ElementFormat 物件指定給任何類型的 ContentElement 時，其 locked 屬性會自動設為 true。嘗試修改鎖定的 ElementFormat 物件時，會擲出 IllegalOperationError。最佳作法是先完整定義這類物件，然後再將它指定給 TextElement 實體。

如果您想要修改現有 ElementFormat 實體，請先檢查其 locked 屬性。如果設為 true，則使用 clone() 方法來建立物件的解除鎖定複本。此解除鎖定物件的屬性可以進行變更，您可以稍後將它指定給 TextElement 實體。任何透過它建立的新字行都具有新的格式。先前透過此相同物件建立且使用舊格式的字行會保留不變。

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd:FontDescription = new FontDescription();

        public function ElementFormatCloneExample()
        {
            fd.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            ef1.fontSize = 24;
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontSize = 32;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

使用字體

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

FontDescription 物件會與 ElementFormat 搭配使用，藉此識別字體並定義部分特性。這些特性包含字體名稱、粗細、型態、顯示方式，以及如何找到字體 (裝置或內嵌)。

備註： FTE 不支援 Type 1 字體或點陣圖字體 (例如 Type 3、ATC、sfnt-wrapped CID 或 Naked CID)。

定義字體特性 (FontDescription 物件)

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

FontDescription 物件的 fontName 屬性可以是單一名稱或以逗號分隔的名稱清單。例如，在 「Arial, Helvetica, _sans」清單中，文字引擎會先尋找 「Arial」，接著尋找 「Helvetica」，最後再尋找 「_sans」 (如果找不到前兩個字體)。這組字體名稱包含三個一般裝置字體名稱：“_sans”、“_serif” 和 “_typewriter”。它們會對應至特定裝置字體 (視播放系統而定)。最佳作法是指定預設名稱 (例如所有字體說明中使用裝置字體的名稱)。如果未指定 fontName，則預設會使用 “_serif”。

fontPosture 屬性可以設為預設值 (FontPosture.NORMAL) 或斜體 (FontPosture.ITALIC)。fontWeight 屬性可以設為預設值 (FontWeight.NORMAL) 或粗體 (FontWeight.BOLD)。

```
var fd1:FontDescription = new FontDescription();
fd1.fontName = "Arial, Helvetica, _sans";
fd1.fontPosture = FontPosture.NORMAL;
fd1.fontWeight = FontWeight.BOLD;
```

內嵌與裝置字體的比較

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

FontDescription 物件的 fontLookup 屬性會指定文字引擎要尋找裝置字體或內嵌字體來顯示文字。如果指定裝置字體 (FontLookup.DEVICE)，則執行階段會尋找播放系統上的字體。而指定內嵌字體 (FontLookup.EMBEDDED_CFF) 會讓執行階段尋找在 SWF 檔中指定名稱的內嵌字體。只有內嵌 CFF (壓縮字體格式) 字體才能使用此設定。如果找不到指定的字體，則會使用備用裝置字體。

裝置字體產生較小的 SWF 檔。內嵌字體可讓您各平台之間保持字體的真實度。

```
var fd1:FontDescription = new FontDescription();
fd1.fontLookup = FontLookup.EMBEDDED_CFF;
fd1.fontName = "Garamond, _serif";
```

顯示模式和微調

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

自 Flash Player 10 和 Adobe AIR 1.5 開始提供了 CFF (壓縮字體格式) 顯示。此類型的字體顯示會讓文字更容易辨識，並讓小型字體具備更高的顯示品質。此設定只適用於內嵌字體。FontDescription 會將 renderingMode 屬性預設為此設定 (RenderingMode.CFF)。您可以將此屬性設為 RenderingMode.NORMAL，以符合 Flash Player 7 或舊版本所使用的顯示類型。

選取 CFF 顯示時，第二個屬性 cffHinting 可控制字體的水平詞幹放入子像素格線的方式。預設值 CFFHinting.HORIZONTAL_STEM 會使用 CFF 微調。如果將此屬性設為 CFFHinting.NONE ，則會移除微調 (這適用於動畫或大型字體)。

```
var fd1:FontDescription = new FontDescription();
fd1.renderingMode = RenderingMode.CFF;
fd1.cffHinting = CFFHinting.HORIZONTAL_STEM;
```

鎖定和仿製 FontDescription

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

將 FontDescription 物件指定給 ElementFormat 時，它的 locked 屬性會自動設為 true。嘗試修改鎖定的 FontDescription 物件時，會擲出 IllegalOperationError。最佳作法是先完整定義這類物件，然後再將它指定給 ElementFormat。

如果您想要修改現有 FontDescription，請先檢查它的 locked 屬性。如果設為 true，則使用 clone() 方法來建立物件的解除鎖定複本。此解除鎖定物件的屬性可以進行變更，可以稍後將它指定給 ElementFormat。任何透過此 TextElement 建立的新字行都具有新的格式。先前透過此相同物件建立的字行會保留不變。

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class FontDescriptionCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd1:FontDescription = new FontDescription();
        private var fd2:FontDescription;

        public function FontDescriptionCloneExample()
        {
            fd1.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            fd2 = (fd1.locked) ? fd1.clone() : fd1;
            fd2.fontName = "Arial";
            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontDescription = fd2;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

控制文字

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

FTE 提供一組新的文字格式控制項，以處理對齊和字元間距 (特殊字距和字距調整)。也有屬性可控制斷行方式，以及在字行內設定定位停駐點。

對齊文字

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

對齊文字時，會調整單字 (有時是字母) 之間的間距，讓段落中的所有字行都有相同的長度。此功能的作用是對齊兩側的文字，而單字和字母之間的間距將會有所不同。報紙和雜誌中的文字欄經常會是對齊的。

SpaceJustifier 類別中的 lineJustfication 屬性可讓您控制文字區塊中的字行對齊。LineJustification 類別會定義用來指定對齊選項的常數：ALL_BUT_LAST 對齊所有文字行，但不含最後一行；ALL_INCLUDING_LAST 對齊所有文字，包含最後一行；UNJUSTIFIED 不對齊文字 (預設值)。

若要對齊文字，請將 lineJustification 屬性設為 SpaceJustifier 類別實體，並將該實體指定給 TextBlock 實體的 textJustifier 屬性。下列範例會建立一個段落，而該段落除了最後一行之外，其他所有的文字行都是對齊的。

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class JustifyExample extends Sprite
    {
        public function JustifyExample()
        {
            var str:String = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, " +
            "sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut " +
            "enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut " +
            "aliquip ex ea commodo consequat.";

            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement=new TextElement(str,format);
            var spaceJustifier:SpaceJustifier=new SpaceJustifier("en",LineJustification.ALL_BUT_LAST);

            var textBlock:TextBlock = new TextBlock();
            textBlock.content=textElement;
            textBlock.textJustifier=spaceJustifier;
            createLines(textBlock);
        }

        private function createLines(textBlock:TextBlock):void {
            var yPos=20;
            var textLine:TextLine=textBlock.createTextLine(null,150);

            while (textLine) {
                addChild(textLine);
                textLine.x=15;
                yPos+=textLine.textHeight+2;
                textLine.y=yPos;
                textLine=textBlock.createTextLine(textLine,150);
            }
        }
    }
}
```

若要讓字母以及單字之間的間距不同，請將 SpaceJustifier.letterspacing 屬性設為 true。開啟字母間距可以減少單字之間出現不美觀之間隔的機率，這種情況有時會在使用簡單對齊時發生。

對齊東亞文字

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

若要對齊東亞文字，必須進行額外考量。這些文字可以從上到下書寫，且部分字元 (稱為避頭尾字元) 不能出現在行首或行尾。JustificationStyle 類別會定義下列常數以指定處理這些字元的選項。PRIORITIZE_LEAST_ADJUSTMENT 依據展開或壓縮字行 (視哪種可以產生最想要的結果而定) 來進行對齊。PUSH_IN_KINSOKU 依據在行尾壓縮避頭尾字元來對齊，或當沒有避頭尾字元或該空間不足時，展開字行來對齊。

PUSH_OUT_ONLY 依據展開字行來進行對齊。若要建立垂直亞洲文字區塊，請將 TextBlock.lineRotation 屬性設為 TextRotation.ROTATE_90，並將 ElementFormat.textRotation 屬性設為 TextRotation.AUTO (預設值)。將 textRotation 屬性設為 AUTO 後，文字中的字符會保持垂直，而不會在旋轉字行一起旋轉。AUTO 設定只會針對全形字和全字符逆時針旋轉 90 度 (由字符的 Unicode 屬性所決定)。下列範例會顯示垂直的日文文字區塊，並使用 PUSH_IN_KINSOKU 選項來進行對齊。

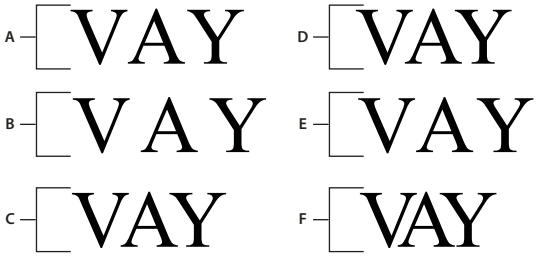
```
package
{
    import flash.text.engine.*;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.system.Capabilities;

    public class EastAsianJustifyExample extends Sprite
    {
        public function EastAsianJustifyExample()
        {
            var Japanese_txt:String = String.fromCharCode(
            0x5185, 0x95A3, 0x5E9C, 0x304C, 0x300C, 0x653F, 0x5E9C, 0x30A4,
            0x30F3, 0x30BF, 0x30FC, 0x30CD, 0x30C3, 0x30C8, 0x30C6, 0x30EC,
            0x30D3, 0x300D, 0x306E, 0x52D5, 0x753B, 0x914D, 0x4FE1, 0x5411,
            0x3051, 0x306B, 0x30A2, 0x30C9, 0x30D3, 0x30B7, 0x30B9, 0x30C6,
            0x30E0, 0x30BA, 0x793E, 0x306E)
            var textBlock:TextBlock = new TextBlock();
            var font:FontDescription = new FontDescription();
            var format:ElementFormat = new ElementFormat();
            format.fontSize = 12;
            format.color = 0xCC0000;
            format.textRotation = TextRotation.AUTO;
            textBlock.baselineZero = TextBaseline.IDEOGRAPHIC_CENTER;
            var eastAsianJustifier:EastAsianJustifier = new EastAsianJustifier("ja",
LineJustification.ALL_BUT_LAST);
            eastAsianJustifier.justificationStyle = JustificationStyle.PUSH_IN_KINSOKU;
            textBlock.textJustifier = eastAsianJustifier;
            textBlock.lineRotation = TextRotation.ROTATE_90;
            var linePosition:Number = this.stage.stageWidth - 75;
            if (Capabilities.os.search("Mac OS") > -1)
                // set fontName: Kozuka Mincho Pro R
                font.fontName = String.fromCharCode(0x5C0F, 0x585A, 0x660E, 0x671D) + " Pro R";
            else
                font.fontName = "Kozuka Mincho Pro R";
            textBlock.content = new TextElement(Japanese_txt, format);
            var previousLine:TextLine = null;

            while (true)
            {
                var textLine:TextLine = textBlock.createTextLine(previousLine, 200);
                if (textLine == null)
                    break;
                textLine.y = 20;
                textLine.x = linePosition;
                linePosition -= 25;
                addChild(textLine);
                previousLine = textLine;
            }
        }
    }
}
```

特殊字距和字距調整

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

特殊字距和字距調整會影響文字區塊中相鄰字元組之間的距離。特殊字距可控制字元組如何相互「搭配」(例如「WA」或「Va」)。特殊字距設定於 ElementFormat 物件中。預設為啟用 (Kerning.ON)，而且可以設為 OFF 或 AUTO，在啟用的情況下，如果不是漢字、平假名或片假名，則特殊字距只會套用至字元之間。

字距調整會在文字區塊中的所有字元之間加上或減去固定的像素數，而且也會設定於 ElementFormat 物件中。它可用於內嵌和裝置字體。FTE 支援兩個字距調整屬性：trackingLeft 會從字元左邊加上 / 減去像素，而 trackingRight 會從右邊加上 / 減去像素。如果使用特殊字距，則會在每個字元組的特殊字距值加上或減去字距調整值。



A. Kerning.OFF **B.** TrackingRight=5、Kerning.OFF **C.** TrackingRight=-5、Kerning.OFF **D.** Kerning.ON **E.** TrackingRight=-5、Kerning.ON **F.** TrackingRight=-5、Kerning.ON

```
var ef1:ElementFormat = new ElementFormat();
ef1.kerning = Kerning.OFF;

var ef2:ElementFormat = new ElementFormat();
ef2.kerning = Kerning.ON;
ef2.trackingLeft = 0.8;
ef2.trackingRight = 0.8;

var ef3:ElementFormat = new ElementFormat();
ef3.trackingRight = -0.2;
```

繞排文字的斷行

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

ElementFormat 物件的 breakOpportunity 屬性會決定當繞排文字分成多行時，哪些字元可用於斷行。預設值 BreakOpportunity.AUTO 使用的是標準 Unicode 屬性 (例如在單字之間斷行以及連字號處斷行)。使用 BreakOpportunity.ALL 可將任何字元視為斷行機會，而這適用於在路徑中將文字斷行的效果。

```
var ef:ElementFormat = new ElementFormat();
ef.breakOpportunity = BreakOpportunity.ALL;
```

定位停駐點

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

若要在文字區塊中設定定位停駐點，請建立 TabStop 類別實體以定義定位停駐點。TabStop() 建構函式的參數會指定如何對齊文字與定位停駐點。這些參數會指定定位停駐點的位置以及要對齊的值 (針對小數點對齊，並以字串表示)。此值一般是小數點，但也可以是逗點、貨幣符號、日元符號或歐元符號 (舉例來說)。下行的程式碼會建立定位停駐點 tab1。

```
var tab1:TabStop = new TabStop(TabAlignment.DECIMAL, 50, ".");
```

一旦建立文字區塊的定位停駐點之後，請將它們指定給 TextBlock 實體的 tabStops 屬性。因為 tabStops 屬性需要 Vector，所以請先建立 Vector 並在其中新增定位停駐點。Vector 可讓您將一組定位停駐點指定給文字區塊。下列範例會建立 Vector<TabStop> 實體，並在其中新增一組 TabStop 物件。然後，它會將定位停駐點指定給 TextBlock 實體的 tabStops 屬性。

```
var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
tabStops.push(tab1, tab2, tab3, tab4);
textBlock.tabStops = tabStops
```

如需有關 Vector 的詳細資訊，請參閱第 21 頁 「使用陣列」。

下列範例會顯示每個 TabStop 對齊選項的效果。

```
package {

    import flash.text.engine.*;
    import flash.display.Sprite;

    public class TabStopExample extends Sprite
    {
        public function TabStopExample()
        {
            var format:ElementFormat = new ElementFormat();
            format.fontDescription = new FontDescription("Arial");
            format.fontSize = 16;

            var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
            tabStops.push(
                new TabStop(TabAlignment.START, 20),
                new TabStop(TabAlignment.CENTER, 140),
                new TabStop(TabAlignment.DECIMAL, 260, "."),
                new TabStop(TabAlignment.END, 380));
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = new TextElement(
                "\tt1\tt2\tt3\tt4\n" +
                "\tThis line aligns on 1st tab\n" +
                "\t\t\t\tThis is the end\n" +
                "\tThe following fragment centers on the 2nd tab:\t\t\n" +
                "\t\tit's on me\t\t\n" +
                "\tThe following amounts align on the decimal point:\n" +
                "\t\t\t45.00\t\n" +
                "\t\t\t75,320.00\t\n" +
                "\t\t\t6,950.00\t\n" +
                "\t\t\t7.01\t\n", format);

            textBlock.tabStops = tabStops;
            var yPosition:Number = 60;
            var previousTextLine:TextLine = null;
            var textLine:TextLine;
            var i:int;
            for (i = 0; i < 10; i++) {
                textLine = textBlock.createTextLine(previousTextLine, 1000, 0);
                textLine.x = 20;
                textLine.y = yPosition;
                addChild(textLine);
                yPosition += 25;
                previousTextLine = textLine;
            }
        }
    }
}
```

Flash Text Engine 範例：新聞版面

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

此程式設計範例會示範如何使用 「Flash 文字引擎」設計簡單的新聞頁面。此頁面包含一個大型標題、一個副標題和一個多欄的內文區段。

首先建立 FLA 檔，並將下列程式碼附加至預設圖層的影格 #2：

```
import com.example.programmingas3.newslayout.StoryLayout ;
// frame sc ript - create  a 3-columned arti cle layout
var story:StoryLayout = new StoryLayout(720, 500, 3, 10);
story.x = 20;
story.y = 80;
addChild(story);
stop();
```

StoryLayout.as 是此範例的控制器指令碼。它會設定內容、從外部樣式表讀取樣式資訊，以及將那些樣式指定給 ElementFormat 物件。然後，它會建立標題、副標題和多欄文字元素。

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.StyleSheet;
    import flash.text.engine.*;

    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.net.URLLoader;
    import flash.display.Sprite;
    import flash.display.Graphics;

    public class StoryLayout extends Sprite
    {
        public var headlineTxt:HeadlineTextField;
        public var subtitleTxt:HeadlineTextField;
        public var storyTxt:MultiColumnText;
        public var sheet:StyleSheet;
        public var h1_ElFormat:ElementFormat;
        public var h2_ElFormat:ElementFormat;
        public var p_ElFormat:ElementFormat;

        private var loader:URLLoader;

        public var paddingLeft:Number;
        public var paddingRight:Number;
        public var paddingTop:Number;
        public var paddingBottom:Number;

        public var preferredWidth:Number;
        public var preferredHeight:Number;

        public var numColumns:int;

        public var bgColor:Number = 0xFFFFFF;

        public var headline:String = "News Layout Example";
        public var subtitle:String = "This example formats text like a newspaper page using the Flash Text
Engine API. ";

        public var rawTestData:String =
        "From the part Mr. Burke took in the American Revolution, it was natural that I should consider him
a friend to mankind; and as our acquaintance commenced on that ground, it would have been more agreeable to
me to have had cause to continue in that opinion than to change it. " +
        "At the time Mr. Burke made his violent speech last winter in the English Parliament against the
French Revolution and the National Assembly, I was in Paris, and had written to him but a short time before
to inform him how prosperously matters were going on. Soon after this I saw his advertisement of the Pamphlet
he intended to publish: As the attack was to be made in a language but little studied, and less understood
in France, and as everything suffers by translation, I promised some of the friends of the Revolution in
that country that whenever Mr. Burke's Pamphlet came forth, I would answer it. This appeared to me the more
```

```
necessary to be done, when I saw the flagrant misrepresentations which Mr. Burke's Pamphlet contains; and
that while it is an outrageous abuse on the French Revolution, and the principles of Liberty, it is an
imposition on the rest of the world. " +
        "I am the more astonished and disappointed at this conduct in Mr. Burke, as (from the circumstances
I am going to mention) I had formed other expectations. " +
        "I had seen enough of the miseries of war, to wish it might never more have existence in the world,
and that some other mode might be found out to settle the differences that should occasionally arise in the
neighbourhood of nations. This certainly might be done if Courts were disposed to set honesty about it, or
if countries were enlightened enough not to be made the dupes of Courts. The people of America had been bred
up in the same prejudices against France, which at that time characterised the people of England; but
experience and an acquaintance with the French Nation have most effectually shown to the Americans the
falsehood of those prejudices; and I do not believe that a more cordial and confidential intercourse exists
between any two countries than between America and France. ";

        public function StoryLayout(w:int = 400, h:int = 200, cols:int = 3, padding:int = 10):void
        {
            this.preferredWidth = w;
            this.preferredHeight = h;

            this.numColumns = cols;

            this.paddingLeft = padding;
            this.paddingRight = padding;
            this.paddingTop = padding;
            this.paddingBottom = padding;

            var req:URLRequest = new URLRequest("story.css");
            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            this.sheet = new StyleSheet();
            this.sheet.parseCSS(loader.data);

            // convert headline styles to ElementFormat objects
            h1_ElFormat = getElFormat("h1", this.sheet);
            h1_ElFormat.typographicCase = TypographicCase.UPPERCASE;
            h2_ElFormat = getElFormat("h2", this.sheet);
            p_ElFormat = getElFormat("p", this.sheet);
            displayText();
        }

        public function drawBackground():void
        {
            var h:Number = this.storyTxt.y + this.storyTxt.height +
                            this.paddingTop + this.paddingBottom;
            var g:Graphics = this.graphics;
            g.beginFill(this.bgColor);
            g.drawRect(0, 0, this.width + this.paddingRight + this.paddingLeft, h);
            g.endFill();
        }

        /**
         * Reads a set of style properties for a named style and then creates
         * a TextFormat object that uses the same properties.
         */
        public function getElFormat(styleName:String, ss:StyleSheet):ElementFormat
        {
        var style:Object = ss.getStyle(styleName);
```

```
if (style != null)
{
var colorStr:String = style.color;
if (colorStr != null && colorStr.indexOf("#") == 0)
{
    style.color = colorStr.substr(1);
}
        var fd:FontDescription = new FontDescription(
                              style.fontFamily,
                              style.fontWeight,
                              FontPosture.NORMAL,
                              FontLookup.DEVICE,
                              RenderingMode.NORMAL,
                              CFFHinting.NONE);
var format:ElementFormat = new ElementFormat(fd,
                      style.fontSize,
                      style.color,
                      1,
                      TextRotation.AUTO,
                      TextBaseline.ROMAN,
                      TextBaseline.USE_DOMINANT_BASELINE,
                      0.0,
                      Kerning.ON,
                      0.0,
                      0.0,
                      "en",
                              BreakOpportunity.AUTO,
                              DigitCase.DEFAULT,
                              DigitWidth.DEFAULT,
                      LigatureLevel.NONE,
                              TypographicCase.DEFAULT);

if (style.hasOwnProperty("letterSpacing"))
{
    format.trackingRight = style.letterSpacing;
}
}
return format;
}

public function displayText():void
{
headlineTxt = new HeadlineTextField(h1_ElFormat,headline,this.preferredWidth);
headlineTxt.x = this.paddingLeft;
```

```
        headlineTxt.y = 40 + this.paddingTop;
            headlineTxt.fitText(1);
            this.addChild(headlineTxt);

        subtitleTxt = new HeadlineTextField(h2_ElFormat,subtitle,this.preferredWidth);
        subtitleTxt.x = this.paddingLeft;
        subtitleTxt.y = headlineTxt.y + headlineTxt.height;
            subtitleTxt.fitText(2);
        this.addChild(subtitleTxt);

        storyTxt = new MultiColumnText(rawTestData, this.numColumns,
                        20, this.preferredWidth, this.preferredHeight, p_ElFormat);
        storyTxt.x = this.paddingLeft;
        storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
        this.addChild(storyTxt);

            drawBackground();
        }
    }
}
```

FormattedTextBlock.as 將做為建立文字區塊的基底類別。它也會包含用來變更字體大小和大小寫的公用程式函數。

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class FormattedTextBlock extends Sprite
    {
        public var tb:TextBlock;
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var textWidth:int;
        public var totalTextLines:int;
        public var blockText:String;
        public var leading:Number = 1.25;
        public var preferredWidth:Number = 720;
        public var preferredHeight:Number = 100;

        public function FormattedTextBlock(ef:ElementFormat,txt:String, colW:int = 0)
        {
            this.textWidth = (colW==0) ? preferredWidth : colW;
            blockText = txt;
            ef1 = ef;
            tb = new TextBlock();
            tb.textJustifier = new SpaceJustifier("en",LineJustification.UNJUSTIFIED,false);
            te = new TextElement(blockText,this.ef1);
            tb.content = te;
            this.breakLines();
        }

        private function breakLines()
        {
            var textLine:TextLine = null;
            var y:Number = 0;
            var lineNum:int = 0;
            while (textLine = tb.createTextLine(textLine,this.textWidth,0,true))
            {
                textLine.x = 0;
                textLine.y = y;
                y += this.leading*textLine.height;
                this.addChild(textLine);
```

```
            }
            for (var i:int = 0; i < this.numChildren; i++)
            {
                TextLine(this.getChildAt(i)).validity = TextLineValidity.STATIC;
            }
            this.totalTextLines = this.numChildren;
        }

        private function rebreakLines()
        {
            this.clearLines();
            this.breakLines();
        }

        private function clearLines()
        {
            while(this.numChildren)
            {
                this.removeChildAt(0);
            }
        }

        public function changeSize(size:uint=12):void
        {
            if (size > 5)
            {
                var ef2:ElementFormat = ef1.clone();
                ef2.fontSize = size;
                te.elementFormat = ef2;
                this.rebreakLines();
            }
        }

        public function changeCase(newCase:String = "default"):void
        {
            var ef2:ElementFormat = ef1.clone();
            ef2.typographicCase = newCase;
            te.elementFormat = ef2;
        }
    }
}
```

HeadlineTextBlock.as 會延伸 FormattedTextBlock 類別，並用於建立標題。它會包含將文字放入頁面上已定義空間的函數。

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    public class HeadlineTextField extends FormattedTextBlock
    {

        public static var MIN_POINT_SIZE:uint = 6;
        public static var MAX_POINT_SIZE:uint = 128;

        public function HeadlineTextField(te:ElementFormat,txt:String,colW:int = 0)
        {
            super(te,txt);
        }

        public function fitText(maxLines:uint = 1, targetWidth:Number = -1):uint
        {
            if (targetWidth == -1)
            {
                targetWidth = this.width;
            }

            var pixelsPerChar:Number = targetWidth / this.blockText.length;
            var pointSize:Number = Math.min(MAX_POINT_SIZE,
                            Math.round(pixelsPerChar * 1.8 * maxLines));

            if (pointSize < 6)
            {
                // the point size is too small
                return pointSize;
            }

            this.changeSize(pointSize);
            if (this.totalTextLines > maxLines)
            {
                return shrinkText(--pointSize, maxLines);
            }
            else
            {
                return growText(pointSize, maxLines);
            }
        }

        public function growText(pointSize:Number, maxLines:uint = 1):Number
        {
            if (pointSize >= MAX_POINT_SIZE)
            {
                return pointSize;
            }

            this.changeSize(pointSize + 1);
            if (this.totalTextLines > maxLines)
            {
                // set it back to the last size
                this.changeSize(pointSize);
                return pointSize;
            }
            else
            {
                return growText(pointSize + 1, maxLines);
            }
        }
```

```
        public function shrinkText(pointSize:Number, maxLines:uint=1):Number
        {
            if (pointSize <= MIN_POINT_SIZE)
            {
                return pointSize;
            }
            this.changeSize(pointSize);

            if (this.totalTextLines > maxLines)
            {
                return shrinkText(pointSize - 1, maxLines);
            }
            else
            {
                return pointSize;
            }
        }
    }
}
```

MultiColumnText.as 會處理多欄設計中的文字格式化。它會示範如何彈性使用 TextBlock 物件做為 Factory 的方式，並藉此建立、格式化和放置文字行。

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.engine.*;

    public class MultiColumnText extends Sprite
    {
        private var tb:TextBlock;
        private var te:TextElement;
        private var numColumns:uint = 2;
        private var gutter:uint = 10;
        private var leading:Number = 1.25;
        private var preferredWidth:Number = 400;
        private var preferredHeight:Number = 100;
        private var colWidth:int = 200;

        public function MultiColumnText(txt:String = "",cols:uint = 2,
            gutter:uint = 10, w:Number = 400, h:Number = 100,
            ef:ElementFormat = null):void
        {
            this.numColumns = Math.max(1, cols);
            this.gutter = Math.max(1, gutter);

            this.preferredWidth = w;
            this.preferredHeight = h;

            this.setColumnWidth();

            var field:FormattedTextBlock = new FormattedTextBlock(ef,txt,this.colWidth);
            var totLines:int = field.totalTextLines;
            field = null;
            var linesPerCol:int = Math.ceil(totLines/cols);

            tb = new TextBlock();
            te = new TextElement(txt,ef);
            tb.content = te;
            var textLine:TextLine = null;
            var x:Number = 0;
            var y:Number = 0;
```

```
            var i:int = 0;
            var j:int = 0;
            while (textLine = tb.createTextLine(textLine,this.colWidth,0,true))
            {
                textLine.x = Math.floor(i/(linesPerCol+1))*(this.colWidth+this.gutter);
                textLine.y = y;
                y += this.leading*textLine.height;
                j++;
                if(j>linesPerCol)
                {
                    y = 0;
                    j = 0;
                }
                i++;

                this.addChild(textLine);
            }
        }

        private function setColumnWidth():void
        {
        this.colWidth = Math.floor( (this.preferredWidth -
            ((this.numColumns - 1) * this.gutter)) / this.numColumns);
        }

    }
}
```

第 23 章 使用 Text Layout Framework

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

Text Layout Framework 的概觀

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

Text Layout Framework (TLF) 是可延伸的 ActionScript 元件庫。TLF 是在 Adobe® Flash® Player 10 與 Adobe® AIR® 1.5 的文字引擎上所建立。TLF 會在網站上提供創新印刷樣式的進階印刷和文字版面功能。此架構可與 Adobe® Flex® 或 Adobe® Flash® Professional 搭配使用。開發人員可以使用或擴充現有的組件，或者他們可以使用架構來建立自己的文字組件。

TLF 包括下列功能：

- 雙向文字、垂直文字以及超過 30 種書寫文字，包括阿拉伯文、希伯來文、中文、日文、韓文、泰文、寮文、越南文等等
- 在多欄與連結的容器之間選取、編輯和排列文字。
- 東亞印刷樣式的垂直文字、直排內橫排 (垂直文字中的水平) 以及對齊
- 豐富的印刷控制項，包括字距微調、連字、印刷大小寫、數字大小寫、數字 ⊠ 度以及選擇性連字號
- 剪下、複製、貼上、還原和標準鍵盤與滑鼠手勢，以進行編輯
- 豐富的開發人員 API，以操作文字內容、版面及標記，並建立自訂文字組件
- 可靠清單支援包括自訂標記與編號格式
- 內嵌影像與定位規則

TLF 是一個 ActionScript 3.0 元件庫，架構於 Flash Player 10 引進的 Flash 文字引擎 (FTE) 之上。FTE 可以透過 flash.text.engine 套件加以存取，而此套件屬於 Flash Player 10 應用程式設計介面 (API) 的一部分。

不過，Flash Player API 只提供對文字引擎的低階存取，這表示部分工作可能需要相當大量的程式碼。TLF 會將低階程式碼封裝至較簡單的 API。TLF 也提供概念性的架構，以便將 FTE 定義的基本建構區塊組織到較易於使用的系統中。

與 FTE 不同的是，Flash Player 並未內建 TLF。而是完全以 ActionScript 3.0 寫成的獨立元件庫。因為這是可擴充的架構，因此可針對特定環境加以自訂。Flash Professional 和 Flex SDK 都包括根據 TLF 架構的組件。

更多說明主題

"Flow" TLF 標記應用程式

支援複雜文字

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

TLF 提供複雜文字的支援。複雜文字的支援包括顯示和編輯由右到左文字的功能。TLF 也提供顯示和編輯由左到右和由右到左文字的功能，如阿拉伯文與希伯來文。此架構不僅支援中文、日文和韓文的垂直文字版面，還支援直排內橫排 (TCY 元素)。TCY 元素是內嵌至垂直連續文字內的水平文字區塊。下列是支援的文字：

- 拉丁文 (英文、西班牙文、法文、越南文等)
- 希臘文、斯拉夫文、亞美尼亞文、喬治亞文及衣索比亞文

- 阿拉伯文與希伯來文
- 漢字表意字和假名 (中文、日文和韓文) 以及韓文 Johab (韓文)
- 泰文、寮文和高棉文
- 梵文、孟加拉文、果魯穆奇文、馬來亞拉姆文、特拉古文、坦米爾文、古吉拉特文、歐利亞文、坎那達文和藏文
- 提非納文、爨文、卻洛奇文、加拿大原住民統一語音節、德瑟列特文、蕭伯納文、范文、塔加拉族文、哈努諾文、布錫語、塔班瓦語

在 Flash Professional 與 Flex 中使用 Text Layout Framework

您可以直接使用 TLF 類別在 Flash 中建立自訂組件。此外，Flash Professional CS5 提供一個新的類別 fl.text.TLFTextField，以封裝 TLF 功能。在 ActionScript 中，請使用 TLFTextField 類別來建立使用 TLF 進階文字顯示功能的文字欄位。建立 TLFTextField 物件的方法與您使用 TextField 類別來建立文字欄位的方法相同。接著使用 textFlow 屬性，從 TLF 類別指定進階格式。

您也可以在 Flash Professional 中使用文字工具，在舞台上建立 TLFTextField 實體。然後就可以使用 ActionScript 的 TLF 類別來控制文字欄位內容的格式與版面。如需詳細資訊，請參閱 「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考」中的 TLFTextField。

如果您是使用 Flex，請使用 TLF 類別。如需詳細資訊，請參閱第 363 頁 「使用 Text Layout Framework」。

使用 Text Layout Framework

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

如果您是使用 Flex 或是建立自訂文字組件，請使用 TLF 類別。TLF 是一個 ActionScript 3.0 元件庫，完全包含在 textLayout.swc 元件庫中。TLF 元件庫大約包含 100 個 ActionScript 3.0 類別和介面，並組織成 10 個套件。這些套件是 flashx.textLayout 套件的子套件。

Text Layout Framework 類別

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

TLF 類別可以區分為三個種類：

- 資料結構與格式化類別
- 顯示類別
- 使用者互動類別

資料結構與格式化類別

下列套件包含 TLF 的資料結構與格式化類別：

- flashx.textLayout.elements
- flashx.textLayout.formats
- flashx.textLayout.conversion

TLF 的主要資料結構是文字流排階層，這是在元素套件中所定義。在此結構內，您可以使用格式套件將樣式和特質指定給連續文字。也可以使用轉換套件，控制文字如何匯入和匯出資料結構。

顯示類別

下列套件包含 TLF 的顯示類別：

- flashx.textLayout.factory
- flashx.textLayout.container
- flashx.textLayout.compose

在這些套件中的類別可協助 Flash Player 顯示文字。factory 套件提供簡單的方式來顯示靜態文字。容器套件包含的類別和介面可定義動態文字的顯示容器。撰寫套件會定義用來在容器中定位和顯示動態文字的技術。

使用者互動類別

下列套件包含 TLF 的使用者互動類別：

- flashx.textLayout.edit
- flashx.textLayout.operations
- flashx.textLayout.events

您可以使用編輯和作業套件定義的類別，來編輯資料結構中儲存的文字。事件套件包含事件處理類別。

使用 Text Layout Framework 建立文字的一般步驟

下列步驟說明使用「文字版面」格式建立文字的一般程序：

1. 將格式化的文字匯入 TLF 資料結構。如需詳細資訊，請參閱第 367 頁「使用 TLF 組織文字」與第 371 頁「使用 TLF 格式化文字」。
2. 為文字建立一或多個連結的顯示物件容器。如需詳細資訊，請參閱第 372 頁「使用 TLF 管理文字容器」。
3. 將資料結構中的文字與容器關聯，並設定編輯和捲動選項。如需詳細資訊，請參閱第 373 頁「使用 TLF 啟用文字選取範圍、編輯和還原」。
4. 建立事件處理常式以重新排列文字，來回應調整大小 (或其他) 事件。如需詳細資訊，請參閱第 374 頁「使用 TLF 處理事件」。

Text Layout Framework 範例：新聞版面

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

下列範例示範如何使用 TLF 來配置簡單的新聞頁面。此頁面包含一個大型標題、一個副標題和一個多欄的內文區段：

```
package
{
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.Event;
    import flash.geom.Rectangle;

    import flashx.textLayout.compose.StandardFlowComposer;
    import flashx.textLayout.container.ContainerController;
    import flashx.textLayout.container.ScrollPolicy;
    import flashx.textLayout.conversion.TextConverter;
    import flashx.textLayout.elements.TextFlow;
    import flashx.textLayout.formats.TextLayoutFormat;

    public class TLFNewsLayout extends Sprite
    {
        private var hTextFlow:TextFlow;
        private var headContainer:Sprite;
        private var headlineController:ContainerController;
        private var hContainerFormat:TextLayoutFormat;

        private var bTextFlow:TextFlow;
        private var bodyTextContainer:Sprite;
        private var bodyController:ContainerController;
        private var bodyTextContainerFormat:TextLayoutFormat;

        private const headlineMarkup:String = "<flow:TextFlow
xmlns:flow='http://ns.adobe.com/textLayout/2008'><flow:p textAlign='center'><flow:span
fontFamily='Helvetica' fontSize='18'>TLF News Layout Example</flow:span><flow:br/><flow:span
fontFamily='Helvetica' fontSize='14'>This example formats text like a newspaper page with a headline, a
subtitle, and multiple columns</flow:span></flow:p></flow:TextFlow>";

        private const bodyMarkup:String = "<flow:TextFlow xmlns:flow='http://ns.adobe.com/textLayout/2008'
fontSize='12' textIndent='10' marginBottom='15' paddingTop='4' paddingLeft='4'><flow:p
marginBottom='inherit'><flow:span>There are many </flow:span><flow:span
fontStyle='italic'>such</flow:span><flow:span> lime-kilns in that tract of country, for the purpose of
burning the white marble which composes a large part of the substance of the hills. Some of them, built
years ago, and long deserted, with weeds growing in the vacant round of the interior, which is open to the
sky, and grass and wild-flowers rooting themselves into the chinks of the stones, look already like relics
of antiquity, and may yet be overspread with the lichens of centuries to come. Others, where the lime-burner
still feeds his daily and nightlong fire, afford points of interest to the wanderer among the hills, who
seats himself on a log of wood or a fragment of marble, to hold a chat with the solitary man. It is a
lonesome, and, when the character is inclined to thought, may be an intensely thoughtful occupation; as it
proved in the case of Ethan Brand, who had mused to such strange purpose, in days gone by, while the fire
in this very kiln was burning.</flow:span></flow:p><flow:p marginBottom='inherit'><flow:span>The man who
now watched the fire was of a different order, and troubled himself with no thoughts save the very few that
were requisite to his business. At frequent intervals, he flung back the clashing weight of the iron door,
and, turning his face from the insufferable glare, thrust in huge logs of oak, or stirred the immense brands
with a long pole. Within the furnace were seen the curling and riotous flames, and the burning marble, almost
molten with the intensity of heat; while without, the reflection of the fire quivered on the dark intricacy
of the surrounding forest, and showed in the foreground a bright and ruddy little picture of the hut, the
spring beside its door, the athletic and coal-begrimed figure of the lime-burner, and the half-frightened
child, shrinking into the protection of his father's shadow. And when again the iron door was closed, then
reappeared the tender light of the half-full moon, which vainly strove to trace out the indistinct shapes
of the neighboring mountains; and, in the upper sky, there was a flitting congregation of clouds, still
faintly tinged with the rosy sunset, though thus far down into the valley the sunshine had vanished long
and long ago.</flow:span></flow:p></flow:TextFlow>";

        public function TLFNewsLayout()
        {
            //wait for stage to exist
```

```
            addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
        }

        private function onAddedToStage(evtObj:Event):void
        {
            removeEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
            stage.scaleMode = StageScaleMode.NO_SCALE;
            stage.align = StageAlign.TOP_LEFT;

            // Headline text flow and flow composer
            hTextFlow = TextConverter.importToFlow(headlineMarkup, TextConverter.TEXT_LAYOUT_FORMAT);

            // initialize the headline container and controller objects
            headContainer = new Sprite();
            headlineController = new ContainerController(headContainer);
            headlineController.verticalScrollPolicy = ScrollPolicy.OFF;
            hContainerFormat = new TextLayoutFormat();
            hContainerFormat.paddingTop = 4;
            hContainerFormat.paddingRight = 4;
            hContainerFormat.paddingBottom = 4;
            hContainerFormat.paddingLeft = 4;

            headlineController.format = hContainerFormat;
            hTextFlow.flowComposer.addController(headlineController);
            addChild(headContainer);
            stage.addEventListener(flash.events.Event.RESIZE, resizeHandler);

            // Body text TextFlow and flow composer
            bTextFlow = TextConverter.importToFlow(bodyMarkup, TextConverter.TEXT_LAYOUT_FORMAT);

            // The body text container is below, and has three columns
            bodyTextContainer = new Sprite();
            bodyController = new ContainerController(bodyTextContainer);
            bodyTextContainerFormat = new TextLayoutFormat();
            bodyTextContainerFormat.columnCount = 3;
            bodyTextContainerFormat.columnGap = 30;

            bodyController.format = bodyTextContainerFormat;
            bTextFlow.flowComposer.addController(bodyController);
            addChild(bodyTextContainer);
            resizeHandler(null);
        }

        private function resizeHandler(event:Event):void
        {
            const verticalGap:Number = 25;
            const stagePadding:Number = 16;
            var stageWidth:Number = stage.stageWidth - stagePadding;
            var stageHeight:Number = stage.stageHeight - stagePadding;
            var headlineWidth:Number = stageWidth;
            var headlineContainerHeight:Number = stageHeight;

            // Initial compose to get height of headline after resize
            headlineController.setCompositionSize(headlineWidth,
headlineContainerHeight);
            hTextFlow.flowComposer.compose();
```

```
            var rect:Rectangle = headlineController.getContentBounds();
            headlineContainerHeight = rect.height;

            // Resize and place headline text container
            // Call setCompositionSize() again with updated headline height
            headlineController.setCompositionSize(headlineWidth, headlineContainerHeight );
            headlineController.container.x = stagePadding / 2;
            headlineController.container.y = stagePadding / 2;
            hTextFlow.flowComposer.updateAllControllers();

            // Resize and place body text container
            var bodyContainerHeight:Number = (stageHeight - verticalGap - headlineContainerHeight);
            bodyController.format = bodyTextContainerFormat;
            bodyController.setCompositionSize(stageWidth, bodyContainerHeight );
            bodyController.container.x = (stagePadding/2);
            bodyController.container.y = (stagePadding/2) + headlineContainerHeight + verticalGap;
            bTextFlow.flowComposer.updateAllControllers();
        }
    }
}
```

TLFNewsLayout 類別使用兩個文字容器。一個容器顯示標題與副標題，而另一個則顯示三欄內文。為了簡潔起見，文字當作 TLF 標記文字，以硬式編碼的方式寫入範例。headlineMarkup 變數包含標題和副標題，bodyMarkup 變數則包含內文。如需 TLF 標記的詳細資訊，請參閱第 367 頁「使用 TLF 組織文字」。

在經過一些初始化後，onAddedToStage() 函數會將標題文字匯入 TextFlow 物件，這是 TLF 的主要資料結構：

```
hTextFlow = TextConverter.importToFlow(headlineMarkup, TextConverter.TEXT_LAYOUT_FORMAT);
```

接下來，會為容器建立 Sprite 物件，並且會建立控制器並與容器相關聯：

```
headContainer = new Sprite();
headlineController = new ContainerController(headContainer);
```

會將控制器初始化以設定格式化、捲動及其他選項。控制器包含定義文字排入之容器邊界的幾何。TextLayoutFormat 物件包含格式化選項：

```
hContainerFormat = new TextLayoutFormat();
```

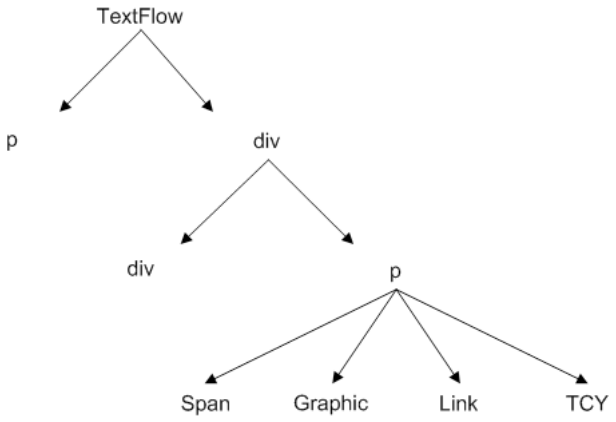
控制器會指定給流排撰寫器，而且函數會將容器新增至顯示清單。容器實際的撰寫和顯示則委託給 resizeHandler() 方法。會執行相同的步驟順序以初始化內文 TextFlow 物件。

resizeHandler() 方法會測量可供顯示容器的空間，然後依此調整容器的大小。初始呼叫 compose() 方法以計算標題容器的適當高度。然後，resizeHandler() 方法就可以使用 updateAllControllers() 方法來置入和顯示標題容器。最後，resizeHandler() 方法會使用標題容器的大小來決定內文容器的位置。

使用 TLF 組織文字

TLF 會使用階層式樹狀結構來代表文字。樹狀結構中的每個節點都是元素套件中所定義的類別實體。例如，樹狀結構的根節點永遠是 TextFlow 類別實體。TextFlow 類別代表整個文字內文。內文是文字與其他元素的集合，視為一個單位或流排。單一內文可能需要一個以上的直欄或文字容器來顯示。

與根節點不同的是，其餘的元素並不是嚴格的 XHTML 元素。下圖顯示架構的階層：



TextFlow 階層

Text Layout Framework 標記

瞭解 TLF 的結構也有助於處理 TLF 標記。TLF 標記是文字的 XML 表示法，這是 TLF 的一部分。雖然此架構也支援其他 XML 格式，但是 TLF 標記的獨特之處在於特別以 TextFlow 階層的結構為基礎。如果您使用此標記格式從 TextFlow 匯出 XML，則 XML 匯出時會具有此完整的階層。

「TLF 標記」在 TextFlow 階層中提供文字最高真實度的呈現。標記語言為每個 TextFlow 階層的基本元素提供標記，而且也為 TextLayoutFormat 類別中的所有可用格式化屬性提供特質。

下表包含可用於 TLF 標記中的標籤。

元素	說明	子系	類別
textflow	標記的根元素。	div、p	TextFlow
div	TextFlow 內的區段。可包含一組段落。	div, list, p	DivElement
p	段落。	a, tcy, span, img, tab, br, g	ParagraphElement
a	連結。	tcy, span, img, tab, br, g	LinkElement
tcy	水平連續文字 (用於垂直 TextFlow 中)。	a, span, img, tab, br, g	TCYElement
span	段落內的連續文字。		SpanElement
img	段落中的影像。		InlineGraphicElement
tab	定位字元。		TabElement
br	換行字元。用於在段落中結束一行；文字會延續到下一行，但保持在同一個段落。		BreakElement
linkNormalFormat	定義正常狀態下的連結格式屬性。	TextLayoutFormat	TextLayoutFormat
linkActiveFormat	當滑鼠落在連結上時，定義用於作用中狀態下的連結格式屬性。	TextLayoutFormat	TextLayoutFormat
linkHoverFormat	當滑鼠在連結的邊緣之內 (或上方) 時，定義用於懸浮狀態下的連結格式屬性。	TextLayoutFormat	TextLayoutFormat

元素	說明	子系	類別
li	清單項目元素。必須在清單元素內。	div, li, list, p	ListItemElement
list	清單。清單可以是巢狀，或與另一個清單相鄰。清單項目可套用不同的標籤或編號配置。	div, li, list, p	ListElement
g	群組項目。用於將項目群組在一個段落中。可讓您將元素巢狀在段落層級之下。	a, tcy, span, img, tab, br, g	SubParagraphGroup Element

更多說明主題

TLF 2.0 清單標記

TLF 2.0 SubParagraphGroupElement 與 typeName

使用編號和項目清單

您可使用 ListElement 與 ListItemElement 類別，將項目清單新增至您的文字控制項。項目清單可以是巢狀的，也可以使用不同的項目符號 (或標記)、自動編號和大綱編號自訂項目清單。

若要在文字流排中建立清單，請使用 <list> 標記。然後針對清單中的每個清單項目，在 <list> 標記中使用 <li> 標記。您可以使用 ListMarkerFormat 類別來自訂項目符號的外觀。

下列範例會建立簡易的清單：

```
<flow:list paddingRight="24" paddingLeft="24">
    <flow:li>Item 1</flow:li>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

您可在其他清單中巢狀清單，如下列範例所示：

```
<flow:list paddingRight="24" paddingLeft="24">
    <flow:li>Item 1</flow:li>
    <flow:list paddingRight="24" paddingLeft="24">
        <flow:li>Item 1a</flow:li>
        <flow:li>Item 1b</flow:li>
        <flow:li>Item 1c</flow:li>
    </flow:list>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

若要自訂清單中的標記類型，請使用 ListElement 的 listStyleType 屬性。此屬性可以是 ListStyleType 類別定義的任何值 (例如 check、circle、decimal 以及 box)。下列範例會建立含有多種標記類型與自訂計數器遞增量的清單：

```
<flow:list paddingRight="24" paddingLeft="24" listStyleType="upperAlpha"> <flow:li>upperAlpha
item</flow:li> <flow:li>another</flow:li> </flow:list> <flow:list paddingRight="24" paddingLeft="24"
listStyleType="lowerAlpha"> <flow:li>lowerAlpha item</flow:li> <flow:li>another</flow:li> </flow:list>
<flow:list paddingRight="24" paddingLeft="24" listStyleType="upperRoman"> <flow:li>upperRoman
item</flow:li> <flow:li>another</flow:li> </flow:list> <flow:list paddingRight="24" paddingLeft="24"
listStyleType="lowerRoman"> <flow:listMarkerFormat> <!-- Increments the list by 2s rather than 1s. -->
<flow:ListMarkerFormat counterIncrement="ordered 2"/> </flow:listMarkerFormat> <flow:li>lowerRoman
item</flow:li> <flow:li>another</flow:li> </flow:list>
```

您可使用 ListMarkerFormat 類別來定義計數器。除定義計數器的遞增量之外，您也可使用 counterReset 屬性重設計數器加以自訂。

您可使用 ListMarkerFormat 的 beforeContent 與 afterContent 屬性，進一步自訂清單中的標記外觀。這些屬性會套用至顯示於標記內容前後的內容。

下列範例會在標記之前新增 “XX” 字串，並在標記之後新增 “YY” 字串：

```
<flow:list listStyleType="upperRoman" paddingLeft="36" paddingRight="24">
    <flow:listMarkerFormat>
        <flow:ListMarkerFormat fontSize="16"
            beforeContent="XX"
            afterContent="YY"
            counterIncrement="ordered -1"/>
        </flow:listMarkerFormat>
    <flow:li>Item 1</flow:li>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

content 屬性本身可定義進一步的標記格式自訂內容。下列範例顯示依順序排列的大寫羅馬數字標記：

```
<flow:list listStyleType="disc"  paddingLeft="96" paddingRight="24">
    <flow:listMarkerFormat>
        <flow:ListMarkerFormat fontSize="16"
            beforeContent="Section "
            content="counters(ordered,&quot;*&quot;,upperRoman)"
            afterContent=": "/>
    </flow:listMarkerFormat>
    <flow:li>Item 1</li>
    <flow:li>Item 2</li>
    <flow:li>Item 3</li>
</flow:list>
```

如先前的範例所示，content 屬性也可插入字尾 (顯示於標記之後、afterContent 之前的字串)。若要在提供流排的 XML 內容時插入此字串，請將字串括在 &quote; HTML 實體中，而不是使用引號 ("<***string***>")。

更多說明主題

TLF 2.0 清單標記

在 TLF 中使用邊框距離

每個 FlowElement 均支援邊框距離屬性，使用此屬性可控制每個元素內容區域的位置，以及內容區域之間的空間。

元素的總寬度是其內容寬度加上 paddingLeft 與 paddingRight 屬性的和。元素的總高度是其內容高度加上 paddingTop 與 paddingBottom 屬性的和。

邊框距離是邊框與內容之間的空間。邊框距離屬性為 paddingBottom、paddingTop、paddingLeft 以及 paddingRight。可套用邊框距離到 TextFlow 物件以及以下子元素：

- div
- img
- li
- list
- p

邊框距離屬性不可套用至範圍元素。

下列範例可設定 TextFlow 上的邊框距離屬性：

```
<flow:TextFlow version="2.0.0" xmlns:flow="http://ns.adobe.com/textLayout/2008" fontSize="14"
textIndent="15" paddingTop="4" paddingLeft="4" fontFamily="Times New Roman">
```

邊框距離屬性的有效值為數字 (以像素為單位)、「auto」或 「inherit」。預設值為 “auto”，顧名思義為一個自動計算的值，除 ListElement 以外的所有元素都將設為 0。對於 ListElement 而言，除了使用 listAutoPadding 屬性值的清單起始端之外，“auto” 均為 0。listAutoPadding 的預設值為 40，提供清單預設的縮排值。

預設不會繼承邊框距離屬性。“auto” 與 “inherit” 值均為 FormatValue 類別定義的常數。

邊框距離屬性可以是負值。

更多說明主題

TLF 2.0 中的邊框距離變更

使用 TLF 格式化文字

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

flashx.textLayout.formats 套件包含的介面和類別，可讓您指定格式給文字流排階層樹狀結構中的任一個 FlowElement。套用格式的方法有兩種。您可以個別指定特定格式，或使用一個特殊格式物件來同時指定一組格式。

ITextLayoutFormat 介面包含可以套用至 FlowElement 的所有格式。部分格式會套用至整個文字容器或文字段落，但不會以此類推地套用至個別字元。例如，格式如對齊和定位停駐點會套用至整個段落，但不會套用至個別字元。

使用屬性指定 FlowElement 的格式

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

您可以透過屬性指定，設定任何 FlowElement 的格式。FlowElement 類別實作 ITextLayoutFormat 介面，因此 FlowElement 類別的所有子類別都必須也實作該介面。

例如，在下列程式碼中，會說明如何指定個別格式給 ParagraphElement 實體。

```
var p:ParagraphElement = new ParagraphElement();
p.fontSize = 18;
p.fontFamily = "Arial";
```

使用 TextLayoutFormat 類別指定 FlowElement 的格式

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

您可以使用 TextLayoutFormat 類別，將格式套用至 FlowElement。這個類別可用來建立含有您想要之所有格式值的特殊格式物件。之後，就可以將該物件指定給任何 FlowElement 物件的 format 屬性。TextLayoutFormat 和 FlowElement 都實作 ITextLayoutFormat 介面。這種安排會確保這兩個類別都包含相同的格式屬性。

如需詳細資訊，請參閱 「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考」中的 TextLayoutFormat。

格式繼承

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

透過文字流程階層繼承格式。如果您將 TextLayoutFormat 實體指定給具有子系的 FlowElement 實體，則該架構會起始「重疊顯示」程序。在重疊顯示期間，架構會遞迴地檢查階層中每個繼承該 FlowElement 的節點。然後決定是否將繼承值指定給每個格式屬性。在重疊顯示期間，會套用下列規則：

1 屬性值只能繼承自直系祖系 (有時稱為父系)。
2 只有在屬性還沒有值 (也就是值為 undefined) 時，才會繼承屬性值 .
3 某些特質在未定義時並不會繼承值，除非該特質的值設為 「inherit」或常數 flashx.textLayout.formats.FormatValue.INHERIT。

例如，如果您在 TextFlow 層級設定 fontSize 值，則該設定會套用至 TextFlow 中的所有元素。換句話說，值會往下重疊顯示文字流排階層。不過，您可以直接指定新的值給指定元素，即可覆寫該元素中的值。另舉一個反例，如果您在 TextFlow 層級設定 backgroundColor 值，則 TextFlow 的子系並不會繼承該值。backgroundColor 屬性在重疊顯示期間並不會從父系繼承。您可以將每個子系的 backgroundColor 屬性設為 flashx.textLayout.formats.FormatValue.INHERIT，就可以覆寫此行為 .

如需詳細資訊，請參閱 「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考」中的 TextLayoutFormat。

使用 TLF 匯入和匯出文字

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

flashx.textLayout.conversion 中的 TextConverter 類別。* 套件可讓您在 TLF 匯入或匯出文字。如果計劃在執行階段載入文字，而不是將文字編譯至 SWF 檔案，請使用這個類別。您也可以使用這個類別，將儲存在 TextFlow 實體中的文字匯出至 String 或 XML 物件。

匯入和匯出都是十分簡單的程序。您可以呼叫 export() 方法或 importToFlow() 方法，這兩種方法都是 TextConverter 類別的一部分。兩種方法都是靜態的，這表示您是呼叫 TextConverter 類別的方法，而不是呼叫 TextConverter 類別實體的方法。

在 flashx.textLayout.conversion 套件中的類別，為您選擇儲存文字的位置提供相當大的彈性。例如，如果您將文字儲存至資料庫中，就可以將文字匯出至架構來顯示文字。您可以使用 flashx.textLayout.edit 套件中的類別來變更文字，並將變更的文字匯回您的資料庫。

如需詳細資訊，請參閱 「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考」中的 flashx.textLayout.conversion。

使用 TLF 管理文字容器

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

將文字儲存在 TLF 資料結構後，Flash Player 即可顯示它。儲存在流排階層中的文字必須轉換為 Flash Player 可以顯示的格式。TLF 提供兩種方式，讓您從流排建立顯示物件。第一個較簡單的方式適用於顯示靜態文字。第二個較複雜的方式可讓您建立可以選取和編輯的動態文字。不論是哪一種方式，文字最終都會轉換為 TextLine 類別的實體，而該類別是 Flash Player 10 中 flash.text.engine.* 套件的一部分。

建立靜態文字

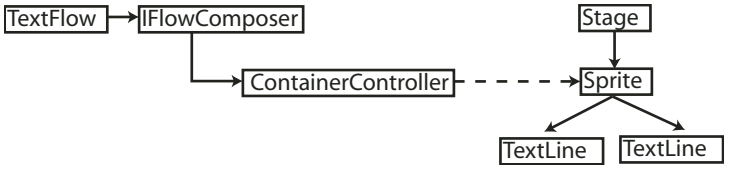
簡單方式使用的是 TextFlowTextLineFactory 類別 (位於 flashx.textLayout.factory 套件)。這個方式除了簡單之外，其優點是記憶體使用量少於 FlowComposer 方式。這種方式適合用於使用者不需要編輯、選取或捲動的靜態文字。

如需詳細資訊，請參閱 「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考」中的 TextFlowTextLineFactory。

建立動態文字與容器

如果想對於文字的顯示擁有比 TextFlowTextLineFactory 提供的控制更多，可以使用流排撰寫器。例如，使用流排撰寫器時，使用者可以選取和編輯文字。如需詳細資訊，請參閱第 373 頁 「使用 TLF 啟用文字選取範圍、編輯和還原」。

流排撰寫器是 flashx.textLayout.compose 套件中 StandardFlowComposer 類別的實體。流排撰寫器可管理如何將 TextFlow 轉換為 TextLine 實體，以及管理如何將那些 TextLine 實體放入一或多個容器。



IFlowComposer 可能沒有任何 ContainerController，也可能有多個 ContainerController。

每個 TextFlow 實體都會有實作 IFlowComposer 介面的對應物件。此 IFlowComposer 物件可透過 TextFlow.flowComposer 屬性加以存取。您可以透過此屬性呼叫由 IFlowComposer 介面定義的方法。這些方法可讓您將此文字與一或多個容器關聯，並準備要在容器中顯示的文字。

容器是 Sprite 類別的實體，而該類別是 DisplayObjectContainer 類別的子類別。這兩個類別都是 Flash Player 顯示清單 API 的一部分。容器是用於 TextLineFactory 類別中矩形邊界的進階型態。與矩形邊界類似，容器會定義 TextLine 實體顯示的區域。與矩形邊界不同的是，容器有對應的 「控制器」物件。控制器會管理一個容器或一組容器的捲動、撰寫、連結、格式化和事件處理。每個容器都有對應的控制器物件，該物件是 flashx.textLayout.container 套件中 ContainerController 類別的實體。

若要顯示文字，請建立控制器物件以管理容器，並將它與流排撰寫器關聯。讓容器產生關聯之後，撰寫文字以供顯示。所以，容器有兩個狀態：撰寫和顯示。撰寫是將文字從文字流排階層轉換為 TextLine 實體，並計算這些實體是否放得進容器的程序。顯示則是更新 Flash Player 顯示清單的程序。

如需詳細資訊，請參閱 「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考」中的 IFlowComposer、StandardFlowComposer 與 ContainerController。

使用 TLF 啟用文字選取範圍、編輯和還原

Flash Player 9.0 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

選取或編輯文字的功能是控制於文字流排層級。TextFlow 類別的每個實體都有一個相關聯的互動管理員。您可以透過 TextFlow 物件的 TextFlow.interactionManager 屬性來存取物件的互動管理員。若要啟用文字選取，請將 SelectionManager 類別的實體指定給 interactionManager 屬性。若要同時啟用文字選取和編輯，請指定 EditManager 類別的實體，而非指定 SelectionManager 類別的實體。若要啟用還原操作，請建立 UndoManager 類別的實體，然後在呼叫 EditManager 的建構函式時傳遞為引數。UndoManager 類別會維護使用者最近編輯活動的操作記錄，並讓使用者還原或重做特定編輯。這三個類別都是編輯套件的一部分。

如需詳細資訊，請參閱「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考」中的 SelectionManager、EditManager 和 UndoManager。

使用 TLF 處理事件

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

TextFlow 物件會在許多情況下傳送事件，包括：

- 當文字或版面變更時
- 作業開始之前和作業完成之後
- FlowElement 物件的狀態變更時
- 撰寫作業完成時

如需詳細資訊，請參閱「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考」中的 flashx.textLayout.events。

更多說明主題

TLF FlowElement 與 LinkElement 事件及 EventMirror

將影像定位在文字內部

若要將 InlineGraphicElement 定位在文字內部，可以使用下列屬性：

- InlineGraphicElement 類別的 float 屬性
- FlowElement 的 clearFloats 屬性

float 屬性可控制圖形與圖形周圍文字的位置。clearFloats 屬性可控制段落元素相對於 float 的位置。

若要將影像位置控制在文字元素內部，可使用 float 屬性。下例範例會新增影像到段落，並將該影像靠左對齊，使文字圍繞在右側：

```
<flow:p paragraphSpaceAfter="15" >Images in a flow are a good thing. For example, here is a float. It should
show on the left: <flow:img float="left" height="50" width="19" source="../assets/bulldog.jpg"></flow:img>
Don't you agree? Another sentence here. Another sentence here. Another sentence here. Another sentence here.
Another sentence here. Another sentence here. Another sentence here. Another sentence here.</flow:p>
```

float 屬性的有效值為 “left”、“right”、“start”、“end” 及 “none”。Float 類別可定義這些常數。預設值為 “none”。

當您想調整一般會圍繞在影像周圍之後續段落的開始位置時，clearFloats 屬性就很有用。例如，假設您有一個比第一個段落大的影像。為確保第二個段落會在影像後面開始，請設定 clearFloats 屬性。

下列範例使用較第一個段落文字高的影像。為使第二個段落在文字區塊中的影像之後開始，此範例將第二個段落的 clearFloats 屬性設為 “end”。

```
<flow:p paragraphSpaceAfter="15" >Here is another float, it should show up on the right: <flow:img
float="right" height="50" elementHeight="200" width="19" source="../assets/bulldog.jpg"></flow:img>We'll
add another paragraph that should clear past it.</flow:p><flow:p clearFloats="end" >This should appear after
the previous float on the right.</flow:p>
```

clearFloats 屬性的有效值為 “left”、“right”、“end”、“start”、“none” 及 “both”。ClearFloats 類別可定義這些常數。您也可將 clearFloats 屬性設為 “inherit”，它是 FormatValue 類別所定義的常數。預設值為 “none”。

更多說明主題

TLF 浮點

第 24 章 使用聲音

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 是為融入式、互動式應用程式所設計，而強大的融入式應用程式往往會忽略聲音這個元素。您可以將音效加入視訊遊戲、將音效回應加入應用程式使用者介面，或甚至建立以聲音做為核心的應用程式，分析透過網際網路載入的 MP3 檔案。

您可載入外部音效檔案，並使用 SWF 內嵌的音效檔。您可控制音效，以建立聲音資訊的視覺呈現形式，並從使用者的麥克風擷取聲音。

更多說明主題

flash.media 套件

flash.events.SampleDataEvent

使用聲音的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

電腦可以捕捉數位音效 (聲音資訊的電腦形式) 並加以編碼，也可以儲存數位音效並加以擷取，透過喇叭進行播放。您可以使用 Adobe® Flash® Player 或 Adobe® AIR™ 和 ActionScript 播放聲音。

當聲音資料轉換為數位形式時，會具有各種不同的特性，例如聲音的音量，以及為立體聲或單聲道。當您在 ActionScript 中播放聲音時，也可以調整這些特性，例如，讓聲音更大聲，或是讓它聽起來好像來自某個方向。

您必須先將聲音資訊載入 Flash Player 或 AIR 中，才能在 ActionScript 中控制聲音。有五種方式可以將音效資料載入 Flash Player 或 AIR 中，讓您透過 ActionScript 加以使用。

- 將 mp3 檔案等外部聲音檔案載入 SWF。
- 在建立 SWF 檔案時，將聲音資訊直接內嵌至 SWF 檔案。
- 從使用者電腦連接的麥克風捕捉音效。
- 從伺服器串流音效。
- 動態產生和播放音效

當您從外部聲音檔案載入聲音資料時，就可以開始播放聲音檔案的開頭，同時繼續載入其餘的聲音資料。

雖然有多種聲音檔案格式可用來編碼數位音效，但是 ActionScript 3.0、Flash Player 和 AIR 都支援儲存格式為 MP3 的聲音檔案。它們無法直接載入或播放具有 WAV 或 AIFF 等其它格式的聲音檔案。

當您使用 ActionScript 中的聲音時，有可能會用到 flash.media 套件的數項類別。Sound 類別是載入聲音檔案或指定函數給對聲音進行取樣的事件然後開始播放時，用以存取音效資訊的類別。一旦開始播放聲音，Flash Player 和 AIR 就會讓您存取 SoundChannel 物件。由於已載入的音效檔案有可能只是您在使用者的電腦上播放的其中一個聲音，因此每個正在播放的聲音都會使用自己的 SoundChannel 物件；而所有 SoundChannel 物件在混合後的結合輸出，便是實際在電腦喇叭上所播放的聲音。您可以使用此 SoundChannel 實體，控制聲音的屬性並停止其播放。最後，如果您想要控制結合的音效，SoundMixer 類別可讓您控制混合的輸出。

當您在 ActionScript 中使用聲音時，也可以使用其它數個類別以執行更細部的工作。如需有關所有與聲音相關之類別的詳細資訊，請參閱第 376 頁「瞭解聲音架構」。

重要概念與術語

下列參考清單包含您可能會遇到的重要術語：

振幅 在聲音波形上的一點到零線或平衡線的距離。

位元速率 針對聲音檔案的每秒編碼或串流資料量。對於 MP3 檔案而言，位元速率通常是以每秒多少千位元 (kbps) 來表示。較高的位元速率通常是指較高品質的聲波。

緩衝 在播放之前接收和儲存聲音資料的作業。

mp3 MPEG-1 Audio Layer 3 (或 MP3)，是常用的聲音壓縮格式。

左右相位 在立體聲的音場中，左右聲道之間的音效訊號位置。

波峰 聲音波形中的最高點。

取樣頻率 定義從類比音效訊號取得的每秒取樣數目以製作數位訊號。標準光碟音效的取樣頻率是每秒 44.1 kHz 或 44,100 個樣本。

串流 在播放聲音檔案或視訊檔案的前面部分時，同時仍然繼續從伺服器載入該檔案之後面部分的程序。

音量 聲音的大小。

波形 經過一段時間後，聲音訊號變化的振幅圖像形狀。

瞭解聲音架構

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

應用程式可以從五個主要來源載入聲音資料：

- 在執行階段載入的外部聲音檔案
- 內嵌於應用程式 SWF 檔的聲音資源
- 來自麥克風 (連接到使用者的系統) 的聲音資料
- 從遠端媒體伺服器 (例如 Flash Media Server) 串流的聲音資料
- 透過使用 sampleData 事件處理常式動態產生的聲音資料

聲音資料可以在播放之前完全載入，也可以藉由串流方式載入 (也就是在播放的同時仍然會繼續載入作業)。

ActionScript 3.0 聲音類別支援儲存格式為 MP3 的聲音檔案。它們無法直接載入或播放具有 WAV 或 AIFF 等其它格式的聲音檔案。不過，從 Flash Player 9.0.115.0 開始，您可以使用 NetStream 類別載入和播放 AAC 音效檔案。這項技巧與用來載入和播放視訊內容的技巧相同。如需有關這項技巧的詳細資訊，請參閱第 403 頁 「使用視訊」。

使用 Adobe Flash Professional 時，您可以匯入 WAV 或 AIFF 聲音檔案，然後以 MP3 格式將它們嵌入應用程式的 SWF 檔中。Flash 編寫工具也能讓您壓縮嵌入的聲音檔案，以縮減檔案大小，但是縮減大小之後，聲音品質便會降低。如需詳細資訊，請參閱 「使用 Flash」中的 「匯入聲音」。

ActionScript 3.0 聲音架構會利用 flash.media 套件中的下列類別。

類別	說明
flash.media.Sound	Sound 類別會處理聲音的載入作業、管理基本的聲音屬性，以及開始播放聲音。
flash.media.SoundChannel	當應用程式播放 Sound 物件時，會建立新的 SoundChannel 物件以控制播放作業。SoundChannel 物件會控制聲音左右播放聲道的音量。每個聲音都會播放屬於自己的 SoundChannel 物件。
flash.media.SoundLoaderContext	SoundLoaderContext 類別可指定在載入聲音時要使用的緩衝秒數，以及指定在載入檔案時，Flash Player 或 AIR 是否要從伺服器尋找原則檔案。SoundLoaderContext 物件是用來當做 Sound.load() 方法的參數。

類別	說明
flash.media.SoundMixer	SoundMixer 類別可控制與應用程式中所有聲音都相關的播放和安全性屬性。事實上，多個聲道會透過共通的 SoundMixer 物件混合在一起，因此在 SoundMixer 物件中的屬性值將會影響所有目前正在播放的 SoundChannel 物件。
flash.media.SoundTransform	SoundTransform 類別含有控制音量和左右相位的值。此外，SoundTransform 物件還可以套用至個別的 SoundChannel 物件、全域 SoundMixer 物件或是 Microphone 物件。
flash.media.ID3Info	ID3Info 物件包含代表 ID3 中繼資料資訊的屬性，該資訊通常儲存在 MP3 聲音檔案中。
flash.media.Microphone	Microphone 類別代表連接到使用者電腦的麥克風或其它聲音輸入裝置。麥克風的音效輸入可以遞送至本機喇叭或傳送至遠端伺服器。Microphone 物件會控制屬於自己的聲音串流之增量、取樣頻率和其它特性。
flash.media.AudioPlaybackMode	AudioPlaybackMode 類別會定義 SoundMixer 類別之 audioPlaybackMode 屬性的常數。

每個載入和播放的聲音都需要屬於自己的 Sound 類別實體和 SoundChannel 類別實體。接著，在播放期間，多個 SoundChannel 實體的輸出就會由全域 SoundMixer 類別加以混合。

Sound、SoundChannel 和 SoundMixer 類別不會用於從麥克風或從像是 Flash Media Server 等串流媒體伺服器取得的聲音資料。

載入外部聲音檔案

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

每個 Sound 類別的實體都是用來載入特定聲音資源並觸發播放動作。應用程式無法重複使用 Sound 物件來載入多個聲音。如果想要載入新的聲音資源，就必須建立新的 Sound 物件。

如果您正在載入小型聲音檔案，例如要附加至按鈕的按一下聲音，應用程式可以建立新的 Sound，並讓它自動載入聲音檔案，如下所示：

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
```

Sound() 建構函式會接受 URLRequest 物件做為它的第一個參數。當提供 URLRequest 參數的值時，新的 Sound 物件便會開始自動載入指定的聲音資源。

除了最簡單的例子以外，應用程式都應該注意聲音的載入進度，並在載入期間注意是否發生錯誤。例如，假設按一下聲音所佔的大小相當大，則可能在使用者按一下觸發該聲音的按鈕時，該聲音仍尚未完全載入。嘗試播放未載入的聲音可能會造成執行階段錯誤，因此，最好能等聲音完全載入，再讓使用者採取可能開始播放聲音的行動。

Sound 物件會在聲音載入程序期間傳送一些不同的事件。您的應用程式可以偵聽這些事件以追蹤載入進度，並確定在播放之前聲音已完全載入。下表列出 Sound 物件可以傳送的事件。

事件	說明
open (Event.OPEN)	在聲音載入作業開始之前立即傳送。
progress (ProgressEvent.PROGRESS)	從檔案或串流接收資料時，在聲音載入過程中定期傳送。
id3 (Event.ID3)	當 MP3 聲音有 ID3 資料可用時傳送。
complete (Event.COMPLETE)	已載入所有聲音資源的資料時傳送。
ioError (IOErrorEvent.IO_ERROR)	找不到聲音檔案，或是在接收所有聲音資料之前中斷載入程序時傳送。

下列程式碼說明聲音完成載入之後的播放方式：

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(Event.COMPLETE, onSoundLoaded);
var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onSoundLoaded(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
```

首先，此程式碼樣本會建立新的 Sound 物件，並且不會為 URLRequest 參數指定初始值。接著，它會偵聽 Sound 物件的 Event.COMPLETE 事件，該事件會在所有聲音資料都已載入時執行 onSoundLoaded() 方法。然後，它會呼叫 Sound.load() 方法，並搭配該聲音檔案的新 URLRequest 值。

當聲音載入完成時，就會執行 onSoundLoaded() 方法。Event 物件的 target 屬性是 Sound 物件的參考。呼叫 Sound 物件的 play() 方法，然後開始播放聲音。

監視聲音載入的過程

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

聲音檔案有可能很大，而且需要很長的時間才能載入。當 Flash Player 和 AIR 應用程式甚至在未完全載入聲音之前就已開始播放，您可能會想要提示使用者，指出已載入的聲音資料進度，以及已播放聲音的進度。

Sound 類別會傳送下列兩個事件，進而讓聲音的載入進度顯示變得相當容易：ProgressEvent.PROGRESS 和 Event.COMPLETE。下列範例將說明如何使用這些事件，顯示關於載入聲音的進度資訊：

```
import flash.events.Event;
import flash.events.ProgressEvent;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
s.addEventListener(Event.COMPLETE, onLoadComplete);
s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);

var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onLoadProgress(event:ProgressEvent):void
{
    var loadedPct:uint = Math.round(100 * (event.bytesLoaded / event.bytesTotal));
    trace("The sound is " + loadedPct + "% loaded.");
}

function onLoadComplete(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
function onIOError(event:IOErrorEvent)
{
    trace("The sound could not be loaded: " + event.text);
}
```

此程式碼首先會為 ProgressEvent.PROGRESS 和 Event.COMPLETE 事件建立 Sound 物件，然後將偵聽程式加入該物件。在呼叫 Sound.load() 方法並且從聲音檔案接收第一份資料之後，就會發生 ProgressEvent.PROGRESS 事件，並觸發 onSoundLoadProgress() 方法。

已載入的聲音資料之百分比等於 ProgressEvent 物件的 bytesLoaded 屬性值除以 bytesTotal 屬性值。在 Sound 物件上也可以使用相同的 bytesLoaded 與 bytesTotal 屬性。上述範例只是顯示聲音載入進度的訊息，但是您可以輕鬆地使用 bytesLoaded 與 bytesTotal 值以更新進度列組件，例如 Adobe Flex 架構或 Adobe Flash 編寫工具隨附的組件。

此範例也會說明在載入聲音檔案時，應用程式如何辨識和回應錯誤。例如，如果找不到指定檔案名稱的聲音檔案，Sound 物件就會傳送 Event.IO_ERROR 事件。在上述程式碼中，發生錯誤時，就會執行 onIOError() 方法，並顯示簡短的錯誤訊息。

使用內嵌聲音

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

對於應用程式的使用者介面中，做為指示器的小型聲音 (例如，按一下按鈕時播放的聲音) 而言，使用內嵌聲音 (而不是從外部檔案載入聲音) 是最有用的。

當您在應用程式中嵌入聲音檔案時，所產生的 SWF 檔大小會隨著聲音檔案的大小而增加。換句話說，在應用程式中嵌入大型聲音檔案可能會使 SWF 檔的大小增加到無法預期的大小。

將聲音檔案嵌入應用程式 SWF 檔的正確方法會隨著開發環境而異。

在 Flash 中使用內嵌聲音檔案

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash 編寫工具可讓您以多種聲音格式匯入聲音，並將它們儲存為 「元件庫」中的元件。您接著便可以將它們指定給時間軸中的影格或某個按鈕狀態的影格，也可以將它們搭配 「行為」使用，或直接在 ActionScript 程式碼中使用。本節會說明如何透過 Flash 編寫工具，在 ActionScript 程式碼中使用內嵌聲音。如需有關在 Flash 中使用內嵌聲音之其它方式的詳細資訊，請參閱 「使用 Flash」中的 「匯入聲音」。

使用 **Flash** 編寫工具嵌入聲音檔案：

1. 選取 「檔案 > 匯入 > 匯入至元件庫」，然後選取一個聲音檔案，將它匯入。
2. 在 「元件庫」面板中的匯入檔案名稱上按一下右鍵，再選取 「屬性」。按一下 「匯出給 ActionScript 使用」核取方塊。
3. 在 「類別」欄位中，輸入要在 ActionScript 中參考此內嵌聲音時使用的名稱。根據預設，此欄位中會使用聲音檔案的名稱。如果檔案名稱包含句點，例如 「DrumSound.mp3」這個名稱中的句點，您必須將它變更成像 「DrumSound」這樣，因為 ActionScript 不允許類別名稱中含有句點字元。「基底類別」欄位應該仍然會顯示 flash.media.Sound。
4. 按一下 「確定」。您可能會看到一個對話方塊，表示在類別路徑中找不到這個類別的定義。按一下 「確定」並繼續。如果您輸入的類別名稱與應用程式類別路徑中任何類別名稱都不相符，就會自動產生繼承自 flash.media.Sound 類別的新類別。
5. 若要使用內嵌聲音，必須在 ActionScript 中參考該聲音的類別名稱。例如，下列程式碼一開始會建立自動產生之 DrumSound 類別的新實體：

   ```
   var drum:DrumSound = new DrumSound();
   var channel:SoundChannel = drum.play();
   ```

   DrumSound 是 flash.media.Sound 類別的子類別，因此會繼承 Sound 類別的方法和屬性，包括如上所示的 play() 方法。

在 Flex 中使用內嵌聲音檔案

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

有許多方式可以將字體嵌入 Flex 應用程式中，包括：

- 在指令碼中使用 [Embed] 中繼資料標籤
- 在 MXML 中使用 @Embed 指令，以指定內嵌資源做為組件 (如 Button 或 a SoundEffect) 的屬性。
- 在 CSS 檔案中使用 @Embed 指令

本節內容說明第一種做法：如何使用 [Embed] 中繼資料標籤，將聲音嵌入 Flex 應用程內的 ActionScript 程式碼中。

若要將資源嵌入 ActionScript 程式碼中，請使用 [Embed] 中繼資料標籤。

將聲音檔案放在主來源資料夾或專案組建路徑中的其他資料夾。當 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。

下列程式碼嵌入稱為 smallSound.mp3 的聲音，並使用 soundClass 變數，將參考儲存至與該聲音相關聯的內嵌資源類別。然後，程式碼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.Sound;
    import flash.media.SoundChannel;

    public class EmbeddedSoundExample extends Sprite
    {
        [Embed(source="smallSound.mp3")]
        public var soundClass:Class;

        public function EmbeddedSoundExample()
        {
            var smallSound:Sound = new soundClass() as Sound;
            smallSound.play();
        }
    }
}
```

若要使用內嵌聲音來設定 Flex 組件的屬性，則應該轉換為 mx.core.SoundAsset 類別的實體，而不是 Sound 類別的實體。如需使用 SoundAsset 類別的類似範例，請參閱 「學習 ActionScript 3.0」中的 「內嵌資源類別」。

使用串流聲音檔案

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當聲音或視訊檔案在其資料仍然繼續載入期間就已播放，即稱為 「串流」。從遠端伺服器載入的外部聲音檔案通常會進行串流。如此一來，使用者就不必等到所有聲音資料載入之後才能聽到聲音。

SoundMixer.bufferTime 屬性代表 Flash Player 或 AIR 在播放聲音之前，應該取得的聲音資料毫秒數。換句話說，如果 bufferTime 屬性設定為 5000，Flash Player 或 AIR 就會從聲音檔案載入至少 5000 毫秒的資料，才開始播放聲音。預設的 SoundMixer.bufferTime 值是 1000。

您可以在載入聲音時明確地指定新的 bufferTime 值，讓應用程式覆寫個別聲音的全域 SoundMixer.bufferTime 值。若要覆寫預設的緩衝時間，請先建立 SoundLoaderContext 類別的新實體、設定其 bufferTime 屬性，然後將它當做參數傳遞給 Sound.load() 方法，如下所示：

```
import flash.media.Sound;
import flash.media.SoundLoaderContext;
import flash.net.URLRequest;

var s:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
var context:SoundLoaderContext = new SoundLoaderContext(8000, true);
s.load(req, context);
s.play();
```

當播放繼續時，Flash Player 和 AIR 會嘗試將聲音緩衝維持在相同的大小或更大。如果聲音資料載入的速度比播放速度還快，播放將會繼續而不會中斷。不過，如果資料載入速率因為網路限制而慢下來，則播放磁頭可能會達到聲音緩衝區的結尾。如果發生此情況，播放就會暫停，但是一旦有更多聲音資料載入，它就會自動繼續。

若要得知播放是否因為 Flash Player 或 AIR 在等待資料載入而暫停，請使用 Sound.isBuffering 屬性。

使用動態產生的音效

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

備註：從 Flash Player 10 和 Adobe AIR 1.5 開始引進了使用動態產生音訊的功能

您不要載入或串流已有的聲音，也可以動態產生音效資料。當您為 Sound 物件的 sampleData 事件指定事件偵聽程式時，可以產生音效資料 (sampleData 會定義於 flash.events 套件的 SampleDataEvent 類別中)。在這個環境中，Sound 物件並不會從檔案載入聲音資料，而是做為透過指定給此事件的函數而串流給它之聲音資料的通訊端。

當您將 sampleData 事件偵聽程式加入至 Sound 物件時，這個物件會定期要求資料以加入至聲音緩衝。這項緩衝包含 Sound 物件要播放的資料。當您呼叫 Sound 物件的 play() 方法時，這個方法會在要求新的聲音資料時傳送 sampleData 事件 (只有在 Sound 物件尚未從檔案載入 MP3 資料時才適用)。

SampleDataEvent 物件包含 data 屬性。您可以在自己的事件偵聽程式中，將 data 物件設定為 ByteArray 物件。您對這個物件設定的位元組陣列會加入 Sound 物件播放的緩衝聲音資料。緩衝區中的位元組陣列是浮點值從 -1 到 1 的串流。每個浮點值都表示聲音樣本某一聲道 (左或右) 的振幅。對聲音進行取樣時，取樣頻率是每秒 44,100 個樣本。每個樣本都包含左右聲道，並插入做為位元組陣列中的浮點資料。

您可以在自己的處理常式函數中，使用 ByteArray.writeFloat() 方法將 data 屬性設定為 sampleData 事件。例如，下列程式碼會產生正弦波：

```
var mySound:Sound = new Sound();
mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, sineWaveGenerator);
mySound.play();
function sineWaveGenerator(event:SampleDataEvent):void
{
    for (var i:int = 0; i < 8192; i++)
    {
        var n:Number = Math.sin((i + event.position) / Math.PI / 4);
        event.data.writeFloat(n);
        event.data.writeFloat(n);
    }
}
```

呼叫 Sound.play() 時，應用程式會開始呼叫您的事件處理常式，要求聲音樣本資料。當聲音播放時，應用程式會持續傳送事件，直到您停止提供資料或呼叫 SoundChannel.stop() 為止。

事件的延遲因平台而異，而且在未來發行的 Flash Player 和 AIR 中可能會變更。請勿依賴特定的延遲，建議您計算所需的延遲。若要計算延遲，請使用下列公式：

```
(SampleDataEvent.position / 44.1) - SoundChannelObject.position
```

每次呼叫事件偵聽程式時，提供給 SampleDataEvent 物件之 data 屬性的樣本數範圍可以從 2048 到 8192。為了達到最佳效能，請盡可能提供多個樣本 (最高可達 8192 個樣本)。您提供越少的樣本，播放時越可能發生喀嚓聲和爆音。這個行為可能因平台而異，而且會發生於各種情況，例如調整瀏覽器大小時。您撰寫的程式碼在只提供 2048 個樣本時可能適用於某個平台，但該程式碼在不同平台上執行時可能會不盡理想。如果您需要最低的延遲，請考慮讓使用者可選取資料量。

如果在每次呼叫 sampleData 事件偵聽程式時，您提供的樣本數少於 2048，應用程式就會在播放剩餘的樣本之後停止，SoundChannel 物件接著會傳送 SoundComplete 事件。

修改 MP3 資料的聲音

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

您可以使用 Sound.extract() 方法從 Sound 物件擷取資料，然後再使用 (及修改) 該資料，以寫入至其它 Sound 物件的動態串流供播放之用。例如，下列程式碼會使用已載入之 MP3 檔案的位元組，並透過濾鏡函數 upOctave() 加以傳送：

```
var mySound:Sound = new Sound();
var sourceSnd:Sound = new Sound();
var urlReq:URLRequest = new URLRequest("test.mp3");
sourceSnd.load(urlReq);
sourceSnd.addEventListener(Event.COMPLETE, loaded);
function loaded(event:Event):void
{
    mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, processSound);
    mySound.play();
}
function processSound(event:SampleDataEvent):void
{
        var bytes:ByteArray = new ByteArray();
        sourceSnd.extract(bytes, 8192);
        event.data.writeBytes(upOctave(bytes));
}
function upOctave(bytes:ByteArray):ByteArray
{
    var returnBytes:ByteArray = new ByteArray();
    bytes.position = 0;
    while(bytes.bytesAvailable > 0)
    {
        returnBytes.writeFloat(bytes.readFloat());
        returnBytes.writeFloat(bytes.readFloat());
        if (bytes.bytesAvailable > 0)
        {
            bytes.position += 8;
        }
    }
    return returnBytes;
}
```

所產生之聲音的限制

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

當您搭配 Sound 物件使用 sampleData 事件偵聽程式時，所啟用的其它 Sound 方法只有 Sound.extract() 和 Sound.play()。呼叫任何其它方法或屬性都會產生例外。SoundChannel 物件的所有方法和屬性仍然會啟用。

播放聲音

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

播放載入的聲音就像呼叫 Sound 物件的 Sound.play() 方法一樣簡單，如下所示：

```
var snd:Sound = new Sound(new URLRequest("smallSound.mp3"));
snd.play();
```

使用 ActionScript 3.0 播放聲音時，您可以執行下列作業：

- 從特定的開始位置播放聲音
- 先暫停聲音，之後再從相同的位置繼續播放
- 確實得知聲音完成播放的時間
- 追蹤聲音的播放進度
- 在聲音播放期間變更聲音或左右相位

若要在播放期間執行這些作業，請使用 SoundChannel、SoundMixer 和 SoundTransform 類別。

SoundChannel 類別可控制單一聲音的播放作業。您可以將 SoundChannel.position 屬性視為播放磁頭，指出正在播放的聲音資料中目前的點。

當應用程式呼叫 Sound.play() 方法時，就會建立 SoundChannel 類別的新實體以控制播放作業。

您可以將位置 (以毫秒為單位) 當做 Sound.play() 方法的 startTime 參數來傳遞，讓應用程式從特定的開始位置播放聲音。透過在 Sound.play() 方法的 loops 參數中傳遞數值，也可以指定固定次數以快速連續的方式重複播放聲音。

呼叫 Sound.play() 方法並搭配 startTime 與 loops 參數時，每次都會從相同的起點重複播放聲音，如下列程式碼所示：

```
var snd:Sound = new Sound(new URLRequest("repeatingSound.mp3"));
snd.play(1000, 3);
```

在此範例中，於聲音開始播放之後，就會從某個點播放聲音一秒鐘，並連續播放三次。

暫停和繼續聲音

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如果應用程式播放較長的聲音，例如歌曲或 Podcast，您可能要讓使用者可暫停及繼續播放那些聲音。若是使用 ActionScript，則在播放期間無法以確實地暫停聲音，而只能停止聲音。不過，您可以從任何點開始播放聲音。因此，您可以在聲音停止時記錄它的位置，並在稍後從該位置開始重播聲音。

例如，假設下列程式碼會載入和播放聲音檔案：

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var channel:SoundChannel = snd.play();
```

在播放聲音期間，SoundChannel.position 屬性會指出聲音檔案中目前正在播放的那一點。您的應用程式可以在停止播放聲音之前儲存該位置值，如下所示：

```
var pausePosition:int = channel.position;
channel.stop();
```

若要繼續播放聲音，請傳遞之前儲存的位置值，以便從之前停止的相同點重新啟動聲音。

```
channel = snd.play(pausePosition);
```

監視播放作業

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您的應用程式可能會想知道聲音何時停止播放，這樣它才可以開始播放另一個聲音，或是清除在先前播放期間所使用的某些資源。當 SoundChannel 類別的聲音完成播放時，此類別就會傳送 Event.SOUND_COMPLETE 事件。您的應用程式可以偵聽此事件並採取適當的行動，如下所示：

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("smallSound.mp3");
snd.load(req);

var channel:SoundChannel = snd.play();
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

public function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
}
```

SoundChannel 類別在播放期間不會傳送進度事件。若要報告播放進度，應用程式可以設定自己的時間機制，並追蹤聲音播放磁頭的位置。

若要計算已播放的聲音百分比，您可以將 SoundChannel.position 屬性的值除以正在播放的聲音資料長度：

```
var playbackPercent:uint = 100 * (channel.position / snd.length);
```

不過，只有聲音資料在播放開始之前就已完全載入的情況下，此程式碼才會報告精確的播放百分比。Sound.length 屬性會顯示目前載入的聲音資料大小，而不是整個聲音檔案的最終大小。若要追蹤仍然繼續在載入的串流聲音之播放進度，您的應用程式應該估計完整聲音檔案的最終大小，並在上述計算方式中使用該值。您可以使用 Sound 物件的 bytesLoaded 與 bytesTotal 屬性，估算聲音資料的最終長度，如下所示：

```
var estimatedLength:int =
    Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
var playbackPercent:uint = 100 * (channel.position / estimatedLength);
```

下列程式碼會載入較大的聲音檔案，然後使用 Event.ENTER_FRAME 事件做為其計時機制，以顯示播放進度。它會定期報告播放百分比，而計算方式是以目前的位置值除以聲音資料的總長度：

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new
    URLRequest("http://av.adobe.com/podcast/csbu_dev_podcast_epi_2.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

function onEnterFrame(event:Event):void
{
    var estimatedLength:int =
        Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
    var playbackPercent:uint =
        Math.round(100 * (channel.position / estimatedLength));
    trace("Sound playback is " + playbackPercent + "% complete.");
}

function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

在聲音資料開始載入之後，此程式碼會呼叫 snd.play() 方法，並將產生的 SoundChannel 物件儲存在 channel 變數中。接著，它會將事件偵聽程式加入 Event.ENTER_FRAME 事件的主要應用程式，並將另一個事件偵聽程式加入在播放完成時所發生之 Event.SOUND_COMPLETE 事件的 SoundChannel 物件。

每次應用程式到達其動畫中的新影格時，就會呼叫 onEnterFrame() 方法。onEnterFrame() 方法會根據已經載入的資料量，估算聲音檔案的總長度，然後計算並顯示目前的播放百分比。

當整段聲音都已經播放時，就會執行 onPlaybackComplete() 方法，以移除 Event.ENTER_FRAME 事件的事件偵聽程式，這樣它才不會在播放完之後嘗試顯示進度更新。

每秒可以傳送許多次 Event.ENTER_FRAME 事件。在某些情況下，您不會想要這麼頻繁地顯示播放進度。在這些情況下，您的應用程式就可以使用 flash.util.Timer 類別來設定自己的計時機制。請參閱第 1 頁 「使用日期與時間」。

停止串流聲音

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在串流聲音 (也就是在播放聲音時，其資料仍然繼續載入) 的播放程序中，有一個特別的執行邏輯，就是：當您的應用程式針對正在播放串流聲音的 SoundChannel 實體呼叫 SoundChannel.stop() 方法時，聲音播放會停止一個影格，然後在下一個影格上，它會從聲音的開頭重新開始。會發生此情況是因為聲音載入程序還在進行中。若要停止串流聲音的載入和播放，請呼叫 Sound.close() 方法。

載入和播放聲音時的安全性考量

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您應用程式存取聲音資料的功能有可能因為 Flash Player 或 AIR 的安全性模型而受到限制。每個聲音都受限於兩個不同安全執行程序的限制：內容本身的安全執行程序 (內容安全執行程序) 以及載入和播放聲音的應用程式或物件之安全執行程序 (擁有者安全執行程序)。對於應用程式安全執行程序中的 AIR 應用程式內容來說，所有聲音 (包括從其他網域載入的聲音) 都可供應用程式安全執行程序中的內容存取。但是，其他安全執行程序中的內容則與在 Flash Player 中執行之內容遵守相同的規則。如需有關一般 Flash Player 安全性模型以及安全執行程序之定義的詳細資訊，請參閱第 900 頁「安全性」。

內容安全執行程序可控制是否能使用 id3 屬性或 SoundMixer.computeSpectrum() 方法，從聲音擷取詳細的聲音資料。它並不會限制聲音檔案本身的載入或播放。

聲音檔案的原始網域會定義內容安全執行程序的安全性限制。一般而言，如果聲音檔案與載入該檔案的應用程式或物件之 SWF 檔位於相同的網域或資料夾，應用程式或物件便具有該聲音檔案的完整存取權限。如果聲音來自與應用程式不同的網域，則仍然可以使用原則檔案，將它加入內容安全執行程序。

您的應用程式可以將具有 checkPolicyFile 屬性的 SoundLoaderContext 物件當做參數傳遞至 Sound.load() 方法。將 checkPolicyFile 屬性設定為 true，可以告知 Flash Player 或 AIR 從載入聲音的伺服器中尋找原則檔案。如果原則檔案存在，而且它授與載入 SWF 檔的網域存取權限，SWF 檔就可以載入聲音檔案、存取 Sound 物件的 id3 屬性，並且為載入的聲音呼叫 SoundMixer.computeSpectrum() 方法。

擁有者安全執行程序可以控制聲音的本機播放作業。開始播放聲音的應用程式或物件會定義擁有者安全執行程序。

只要正在播放的所有 SoundChannel 物件都符合下列準則，SoundMixer.stopAll() 方法便會讓它們全部靜音：

- 這些聲音都是由相同擁有者安全執行程序中的物件開始播放。
- 這些聲音都是來自具有原則檔案的來源，該檔案會將存取權限授與呼叫 SoundMixer.stopAll() 方法的應用程式或物件之網域。

但在 AIR 應用程式中，應用程式安全執行程序中的內容 (使用 AIR 應用程式所安裝的內容) 沒有這些安全性限制。

若要得知 SoundMixer.stopAll() 方法是否真的會停止所有播放中的聲音，您的應用程式可以呼叫 SoundMixer.areSoundsInaccessible() 方法。如果該方法傳回 true 值，則表示有些播放的聲音是在目前擁有者安全執行程序的控制之外，而且將不會由 SoundMixer.stopAll() 方法停止。

對於從外部檔案載入的所有聲音，SoundMixer.stopAll() 方法也會使播放磁頭不再繼續播放。不過，如果動畫移到新的影格，則內嵌於 FLA 檔中的聲音，以及使用 Flash 編寫工具附加到時間軸中影格的聲音，都可能會再次開始播放。

控制聲音音量和左右相位

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

個別的 SoundChannel 物件會控制聲音的左右立體聲道。如果 MP3 聲音是單聲道的聲音，則 SoundChannel 物件的左右立體聲道將包含相同的聲音波形。

您可以使用 SoundChannel 物件的 leftPeak 與 rightPeak 屬性，得知正在播放聲音之每個立體聲道的振幅。這些屬性可以顯示聲音波形本身的波峰振幅，但是並不代表實際的播放音量。實際的播放音量是聲音波形的振幅函數，而且音量值會在 SoundChannel 物件與 SoundMixer 類別中設定。

您可以使用 SoundChannel 物件的 pan 屬性，在播放期間為每個左聲道和右聲道指定不同的音量。pan 屬性的值範圍包括從 -1 到 1，其中 -1 是指左聲道以最大音量播放，右聲道則為靜音；而 1 則表示右聲道以最大音量播放，左聲道則為靜音。介於 -1 與 1 之間的數值會為左右聲道值設定等比例值，而值為 0 則表示兩個聲道都會以對稱且中等的音量來播放。

下列程式碼範例會以 0.6 做為音量值，並以 -1 做為平衡值 (最大的左聲道音量，無右聲道音量) 來建立 SoundTransform 物件。它會將 SoundTransform 物件當做參數傳遞至 play() 方法，這會將該 SoundTransform 物件套用到用來控制播放作業所建立的新 SoundChannel 物件。

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var trans:SoundTransform = new SoundTransform(0.6, -1);
var channel:SoundChannel = snd.play(0, 1, trans);
```

您可以設定 SoundTransform 物件的 pan 或 volume 屬性，然後將該物件套用為 SoundChannel 物件的 soundTransform 屬性，藉以在聲音播放期間改變音量與左右相位。

您也可以使用 SoundMixer 類別的 soundTransform 屬性，一次設定所有聲音的全域音量與平衡值，如下列範例所示：

```
SoundMixer.soundTransform = new SoundTransform(1, -1);
```

您也可以使用 SoundTransform 物件來設定 Microphone 物件 (請參閱第 391 頁「擷取聲音輸入」)、Sprite 物件及 SimpleButton 物件的音量和平衡值。

下列範例會在播放聲音期間，將聲音的左右相位從左聲道到右聲道以及從右聲道到左聲道輪流交換。

```
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var panCounter:Number = 0;

var trans:SoundTransform;
trans = new SoundTransform(1, 0);
var channel:SoundChannel = snd.play(0, 1, trans);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

addEventListener(Event.ENTER_FRAME, onEnterFrame);

function onEnterFrame(event:Event):void
{
    trans.pan = Math.sin(panCounter);
    channel.soundTransform = trans; // or SoundMixer.soundTransform = trans;
    panCounter += 0.05;
}

function onPlaybackComplete(event:Event):void
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

此程式碼首先會載入聲音檔案，然後建立新的 SoundTransform 物件，並將 volume 屬性設定為 1 (最大音量)，以及將 pan 屬性設定為 0 (左右聲道平衡)。然後，再呼叫 snd.play() 方法，將 SoundTransform 物件當做參數傳遞。

在聲音播放期間，onEnterFrame() 方法會重複執行。onEnterFrame() 方法會使用 Math.sin() 函數來產生介於 -1 與 1 之間的值，這個範圍與 SoundTransform.pan 屬性可接受的值一致。SoundTransform 物件的 pan 屬性會設定為新的值，而聲道的 soundTransform 屬性則設定為使用改變後的 SoundTransform 物件。

若要執行這個範例，請以本機 MP3 檔案的名稱取代 bigSound.mp3 這個檔案名稱。接著，請執行範例。當右聲道音量變小時，您應該會聽到左聲道的音量變大，反之亦然。

在此範例中，設定 SoundMixer 類別的 soundTransform 屬性也可以達到相同的效果。不過，這將會影響目前播放的所有聲音之左右相位，並不只是此 SoundChannel 物件所播放的單一聲音。

使用聲音中繼資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用 MP3 格式的聲音檔案可以包含有關 ID3 標籤格式聲音的其它資料。

不是每個 MP3 檔案都包含 ID3 中繼資料。當 Sound 物件載入 MP3 聲音檔案時，如果該聲音檔案包含 ID3 中繼資料，此物件就會傳送 Event.ID3 事件。若要避免執行階段錯誤，您的應用程式應該等收到 Event.ID3 事件，才存取已載入聲音的 Sound.id3 屬性。

下列程式碼將說明如何辨識已載入聲音檔案的 ID3 中繼資料：

```
import flash.events.Event;
import flash.media.ID3Info;
import flash.media.Sound;

var s:Sound = new Sound();
s.addEventListener(Event.ID3, onID3InfoReceived);
s.load("mySound.mp3");

function onID3InfoReceived(event:Event)
{
    var id3:ID3Info = event.target.id3;

    trace("Received ID3 Info:");
    for (var propName:String in id3)
    {
        trace(propName + " = " + id3[propName]);
    }
}
```

在此程式碼中，會先建立 Sound 物件，並告訴它偵聽 Event.ID3 事件。載入聲音檔案的 ID3 中繼資料時，就會呼叫 onID3InfoReceived() 方法。傳遞至 onID3InfoReceived() 方法之 Event 物件的目標是原始的 Sound 物件，因此該方法接著會取得 Sound 物件的 id3 屬性，然後重複執行所有其具名屬性以追蹤其值。

存取原始聲音資料

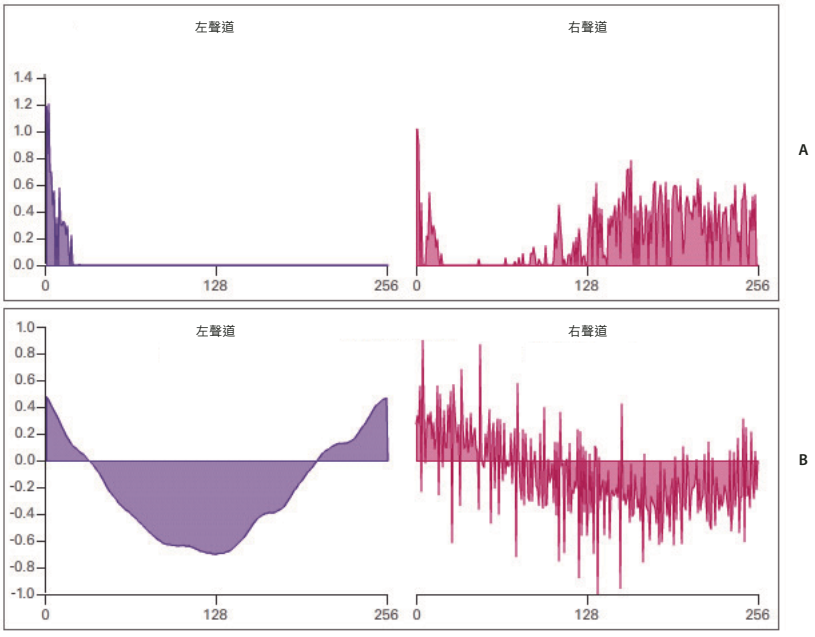
Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

SoundMixer.computeSpectrum() 方法可讓應用程式讀取目前正在播放的聲音波形之原始聲音資料。如果目前有一個以上的 SoundChannel 物件正在播放，SoundMixer.computeSpectrum() 方法就會顯示每個混合在一起的 SoundChannel 物件所結合而成的聲音資料。

聲音資料將以包含 512 位元組資料的 ByteArray 物件傳回，每個物件都包含介於 -1 到 1 的浮點值。這些值都代表正在播放的聲音波形中的點振幅。這些值會以兩個 256 群組傳送，第一個群組是供左立體聲道使用，而第二個群組則是供右立體聲道使用。

如果 FFTMode 參數設定為 true，SoundMixer.computeSpectrum() 方法便會傳回頻譜資料，而不是波形資料。頻譜會從最低頻率到最高頻率，顯示依聲音頻率排列的振幅。快速傅利葉轉換 (Fast Fourier Transform，FFT) 可用來將聲音波形資料轉換為頻譜資料。產生的頻譜值範圍從 0 到大約 1.414 (2 的平方根)。

下圖會比較當 FFTMode 參數設定為 true 以及設定為 false 時，computeSpectrum() 方法所傳回的資料。其資料用於此圖的聲音包含左聲道中大聲的低音以及右聲道中的鼓擊聲。



由 SoundMixer.computeSpectrum() 方法傳回的值
A. fftMode=true **B.** fftMode=false

computeSpectrum() 方法也可以傳回以較低的位元速率重新取樣的資料。一般而言，這雖然無法得到詳細資料，但可獲得較平滑的聲音波形資料或頻率資料。stretchFactor 參數會控制取樣 computeSpectrum() 方法資料的速率。當 stretchFactor 參數設定為預設值 0 時，聲音資料的取樣速率為 44.1 kHz。stretchFactor 參數的後繼值可依次將速率減半，因此值為 1 會指定 22.05 kHz 的速率，而值為 2 則會指定 11.025 kHz 的速率，依此類推。使用較高的 stretchFactor 值時，computeSpectrum() 方法仍然會傳回每個立體聲道 256 個位元組。

SoundMixer.computeSpectrum() 方法具有某些限制：

- 因為來自麥克風或 RTMP 串流的資料無法穿越全域 SoundMixer 物件，所以 SoundMixer.computeSpectrum() 方法將不會從這些來源傳回資料。
- 如果正在播放的一或多個聲音是來自目前內容安全執行程序之外的來源，則安全性限制將會導致 SoundMixer.computeSpectrum() 方法擲回錯誤。如需有關 SoundMixer.computeSpectrum() 方法之安全性限制的詳細資訊，請參閱第 386 頁 「載入和播放聲音時的安全性考量」與第 919 頁 「將載入的媒體當做資料加以存取」。

但在 AIR 應用程式中，應用程式安全執行程序中的內容 (使用 AIR 應用程式所安裝的內容) 沒有這些安全性限制。

建立簡單的聲音視覺化檢視器

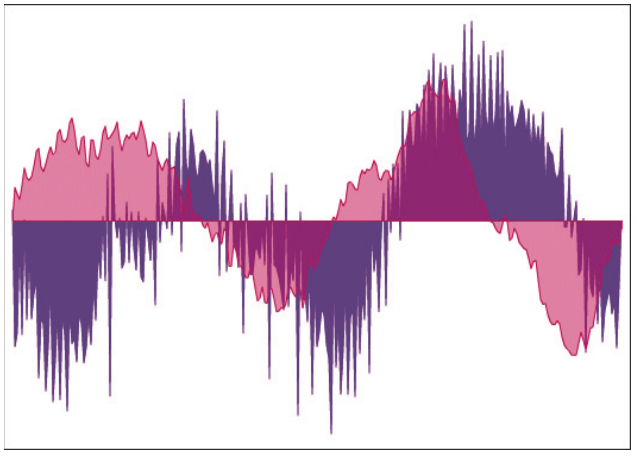
Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列範例會使用 SoundMixer.computeSpectrum() 方法，顯示隨著每個影格變化的聲音波形圖表：

```
import flash.display.Graphics;
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

const PLOT_HEIGHT:int = 200;
const CHANNEL_LENGTH:int = 256;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
snd.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

var bytes:ByteArray = new ByteArray();

function onEnterFrame(event:Event):void
{
    SoundMixer.computeSpectrum(bytes, false, 0);

    var g:Graphics = this.graphics;

    g.clear();
    g.lineStyle(0, 0x6600CC);
    g.beginFill(0x6600CC);
    g.moveTo(0, PLOT_HEIGHT);

    var n:Number = 0;

    // left channel
    for (var i:int = 0; i < CHANNEL_LENGTH; i++)
    {
        n = (bytes.readFloat() * PLOT_HEIGHT);
        g.lineTo(i * 2, PLOT_HEIGHT - n);
    }
    g.lineTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);
    g.endFill();

    // right channel
    g.lineStyle(0, 0xCC0066);
    g.beginFill(0xCC0066, 0.5);
    g.moveTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);

    for (i = CHANNEL_LENGTH; i > 0; i--)
    {
        n = (bytes.readFloat() * PLOT_HEIGHT);
        g.lineTo(i * 2, PLOT_HEIGHT - n);
    }
    g.lineTo(0, PLOT_HEIGHT);
    g.endFill();
}

function onPlaybackComplete(event:Event)
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

此範例會先載入和播放聲音檔案，然後偵聽 Event.ENTER_FRAME 事件，此事件將會在聲音播放時觸發 onEnterFrame() 方法。onEnterFrame() 方法會從呼叫 SoundMixer.computeSpectrum() 方法開始，它會將聲音波形資料儲存在 bytes ByteArray 物件中。

聲音波形會使用向量繪圖 API 進行標繪。for 迴圈會循環第一個 256 資料值，以代表左立體聲道，並使用 Graphics.lineTo() 方法從每個點繪製線條到下一個點。第二個 for 迴圈會循環下一組的 256 值，這一次會以反轉順序，並且從右至左進行標繪。產生的聲音波形標繪可能會出現有趣的鏡像效果，如下列影像所示。



擷取聲音輸入

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Microphone 類別可讓您的應用程式連接到使用者系統上的麥克風或其它聲音輸入裝置，並將輸入音效廣播到該系統的喇叭，或將音效資料傳送到遠端伺服器 (例如 Flash Media Server)。您可以從麥克風和記錄存取原始音效資料或是處理它；也可以直接傳送音效給系統的喇叭，或將壓縮的音效資料傳送至遠端伺服器。對於傳送至遠端伺服器的資料，您可以使用 Speex 或 Nellymoser 轉碼器 (Flash Player 10 和 Adobe AIR 1.5 已可支援 Speex 轉碼器)。

更多說明主題

Michael Chaize：AIR、Android 及麥克風

Christophe Coenraets：Android 的語音備註

存取麥克風

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Microphone 類別並沒有建構函式方法。但是，您可以使用靜態的 Microphone.getMicrophone() 方法來取得新的 Microphone 實體，如下所示：

```
var mic:Microphone = Microphone.getMicrophone();
```

呼叫 Microphone.getMicrophone() 方法並且不使用參數，將會傳回在使用者系統上發現的第一個聲音輸入裝置。

系統可以連接一個以上的聲音輸入裝置。您的應用程式可以使用 Microphone.names 屬性，取得所有可用聲音輸入裝置的名稱陣列。然後，可以呼叫 Microphone.getMicrophone() 方法並搭配 index 參數，該參數符合陣列中裝置名稱的索引值。

系統可能沒有連接麥克風或其它聲音輸入裝置。您可以使用 Microphone.names 屬性或 Microphone.getMicrophone() 方法，檢查使用者是否安裝了聲音輸入裝置。如果使用者沒有安裝聲音輸入裝置，則 names 陣列的長度為零，而且 getMicrophone() 方法會傳回 null 值。

當您的應用程式呼叫 Microphone.getMicrophone() 方法時，Flash Player 會顯示「Flash Player 設定」對話方塊，提示使用者允許或拒絕 Flash Player 存取系統上的攝影機與麥克風。當使用者按一下此對話方塊中的「允許」或「拒絕」按鈕之後，就會傳送 StatusEvent。StatusEvent 實體的 code 屬性會指出允許或拒絕存取麥克風，如此範例所示：

```
import flash.media.Microphone;

var mic:Microphone = Microphone.getMicrophone();
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

function onMicStatus(event:StatusEvent):void
{
    if (event.code == "Microphone.Unmuted")
    {
        trace("Microphone access was allowed.");
    }
    else if (event.code == "Microphone.Muted")
    {
        trace("Microphone access was denied.");
    }
}
```

如果允許存取，StatusEvent.code 屬性將包含「Microphone.Unmuted」，如果拒絕存取，則包含「Microphone.Muted」。

當使用者允許或拒絕存取麥克風時，Microphone.muted 屬性將會分別設定為 true 或 false。不過，在傳送 StatusEvent 之前，並不會設定 Microphone 實體的 muted 屬性，因此您的應用程式也應該等到傳送 StatusEvent.STATUS 事件之後，再檢查 Microphone.muted 屬性。

為了讓 Flash Player 顯示設定對話方塊，應用程式視窗必須夠大才能顯示它 (至少 215 * 138 像素)。否則，會自動拒絕存取。

在 AIR 應用程式安全執行程序中執行的內容，不需要使用者權限，即可存取麥克風。因此，永遠不會傳送麥克風的靜音和取消靜音的狀態事件。在應用程式安全執行程序之外執行的 AIR 內容，確實需要使用者的權限，因此可能會傳送這些狀態事件。

將麥克風音效遞送至本機喇叭

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

藉由呼叫 Microphone.setLoopback() 方法並搭配值為 true 的參數，可以將來自麥克風的音效輸入遞送至本機系統喇叭。

當來自本機麥克風的聲音遞送至本機喇叭時，建立音效回應迴圈時將具有風險，因為這可能會造成大聲且長而尖銳的聲音，而且有可能損毀聲音硬體。呼叫 Microphone.setUseEchoSuppression() 方法並搭配值為 true 的參數會減少 (但是不會完全消除) 發生音訊回授的風險。除非您確定使用者使用耳機或喇叭以外的播放裝置來播放聲音，否則 Adobe 建議在呼叫 Microphone.setLoopback(true) 之前，一定要呼叫 Microphone.setUseEchoSuppression(true)。

下列程式碼將說明如何將音效從本機麥克風遞送至本機系統喇叭：

```
var mic:Microphone = Microphone.getMicrophone();
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

改變麥克風音效

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您的應用程式可以透過兩種方式，修改來自麥克風的音效資料。首先，它可以變更輸入聲音的增量，這將可有效地將輸入值乘以指定的量，以建立較大聲或較小聲的聲音。Microphone.gain 屬性可以接受介於 0 與 100 (包含兩者) 之間的數值。值為 50 就像乘以 1 倍一樣，會指定正常的音量。值為零就像乘以零一樣，可有效地將輸入音效化為無聲。50 以上的值會指定大於正常的音量。

您的應用程式也可以變更輸入音訊的取樣頻率。較高的取樣頻率可提升聲音品質，但是也會建立更密集的資料串流，這些串流會使用更多的資源來傳輸和儲存資料。Microphone.rate 屬性代表單位為 kHz 的音效取樣頻率，預設的取樣頻率是 8 kHz。如果您的麥克風支援較高的頻率，就可以將 Microphone.rate 屬性設定為高於 8 kHz 的值。例如，將 Microphone.rate 屬性值設定為 11，就表示將取樣頻率設定為 11 kHz，而設定為 22 則表示將取樣頻率設定為 22 kHz，依此類推。可用的取樣頻率取決於所選取的轉碼器而有所不同。使用 Nellymoser 轉碼器時，您可以指定 5、8、11、16、22 和 44 kHz 做為取樣頻率；使用 Speex 轉碼器 (從 Flash Player 10 和 Adobe AIR 1.5 開始提供) 時，您只能使用 16 kHz。

偵測麥克風活動

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

為了保留頻寬及處理資源，Flash Player 會在麥克風未傳輸聲音時嘗試進行偵測。當麥克風的活動等級停留在靜音等級臨界值以下一段時間後，Flash Player 會停止傳輸音效輸入，並改傳送簡單的 ActivityEvent。如果您使用 Speex 轉碼器 (Flash Player 10 或更新版本以及 Adobe AIR 1.5 或更新版本中所提供)，請將靜音等級設定為 0，確保應用程式會持續傳輸音效資料。Speex 語音活動偵測會自動降低頻寬。

備註：當應用程式監視麥克風時，Microphone 物件只會傳送 Activity 事件。因此，如果您未呼叫 setLoopBack(true)、未新增範本資料事件的偵聽程式，或是將麥克風連接到 NetStream 物件，則不會傳送任何活動事件。

Microphone 類別的三個屬性會監視和控制活動的偵測作業：

- 唯讀的 activityLevel 屬性會指出麥克風正在偵測的聲音量，從 0 到 100 的等級。
- silenceLevel 屬性會指定要啟動麥克風所需的聲音量，並傳送 ActivityEvent.ACTIVITY 事件。silenceLevel 屬性也會使用 0 到 100 的等級，而預設值則為 10。
- silenceTimeout 屬性會描述活動等級必須停留在靜音等級以下的毫秒數，直到傳送 ActivityEvent.ACTIVITY 事件以指出麥克風目前為靜音為止。silenceTimeout 的預設值為 2000。

Microphone.silenceLevel 與 Microphone.silenceTimeout 都是唯讀屬性，但是可以使用 Microphone.setSilenceLevel() 方法來變更其值。

在某些情況下，偵測到新活動時啟動的麥克風程序可能會造成短暫的延遲。將麥克風永遠保持作用中則可以避免這種啟動延遲狀況。您的應用程式可以呼叫 Microphone.setSilenceLevel() 方法並將 silenceLevel 參數設定為零，告訴 Flash Player 將麥克風保持在作用中，並持續收集音效資料，即使沒有偵測到聲音也是一樣。反之，將 silenceLevel 參數設定為 100，可防止麥克風一直處於啟動的狀態。

下列範例會顯示麥克風的資訊，並報告 Microphone 物件傳送的活動事件與狀態事件：

```
import flash.events.ActivityEvent;
import flash.events.StatusEvent;
import flash.media.Microphone;

var deviceArray:Array = Microphone.names;
trace("Available sound input devices:");
for (var i:int = 0; i < deviceArray.length; i++)
{
    trace(" " + deviceArray[i]);
}

var mic:Microphone = Microphone.getMicrophone();
mic.gain = 60;
mic.rate = 11;
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
mic.setSilenceLevel(5, 1000);

mic.addEventListener(ActivityEvent.ACTIVITY, this.onMicActivity);
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

var micDetails:String = "Sound input device name: " + mic.name + '\n';
micDetails += "Gain: " + mic.gain + '\n';
micDetails += "Rate: " + mic.rate + " kHz" + '\n';
micDetails += "Muted: " + mic.muted + '\n';
micDetails += "Silence level: " + mic.silenceLevel + '\n';
micDetails += "Silence timeout: " + mic.silenceTimeout + '\n';
micDetails += "Echo suppression: " + mic.useEchoSuppression + '\n';
trace(micDetails);

function onMicActivity(event:ActivityEvent):void
{
    trace("activating=" + event.activating + ", activityLevel=" +
        mic.activityLevel);
}

function onMicStatus(event:StatusEvent):void
{
    trace("status: level=" + event.level + ", code=" + event.code);
}
```

當您執行上述範例時，請對著系統麥克風說話或製造一些聲音，並監控在主控台或除錯視窗中產生的 trace 陳述式。

將音效傳入媒體伺服器或自媒體伺服器送出

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用 ActionScript 並搭配 Flash Media Server 等串流媒體伺服器時，可以提供其它音效功能。

特別是，您的應用程式可以將 Microphone 物件附加至 NetStream 物件，並從使用者的麥克風將資料直接傳輸到伺服器。音效資料也可以從伺服器串流至應用程式，並當做 MovieClip 的一部分來播放或使用 Video 物件來播放。

從 Flash Player 10 和 Adobe AIR 1.5 開始引進了 Speex 轉碼器。若要設定用於傳送至媒體伺服器的壓縮音效，請設定 Microphone 物件的 codec 屬性。這個屬性具有兩個值，這兩個值都是 SoundCodec 類別中的列舉值。如果將 codec 屬性設定為 SoundCodec.SPEEX，表示選取 Speex 轉碼器來壓縮音效；而設定為 SoundCodec.NELLYMOSER (預設值) 時，則表示選取 Nellymoser 轉碼器來壓縮音效。

如需詳細資訊，請參閱 Flash Media Server 線上說明文件，網址為 www.adobe.com/go/learn_fms_docs_tw。

捕捉麥克風聲音資料

Flash Player 10.1 以及更新的版本，**Adobe AIR 2** 以及更新的版本

在 Flash Player 10.1. 及 AIR 2 或更新的版本中，您可以從麥克風資料捕捉資料，成為浮點值的位元組陣列。每一個值代表一個單聲道音效資料的取樣。

若要取得麥克風資料，請為 Microphone 物件的 sampleData 事件設定事件偵聽程式。當麥克風緩衝區充滿了聲音取樣時，Microphone 物件會定期傳送 sampleData 事件。SampleDataEvent 物件具有 data 屬性，它是聲音取樣的位元組陣列。每一個浮點值代表一個取樣，每個都是一個單聲道聲音取樣。

以下程式碼會捕捉麥克風聲音資料，然後放到名稱為 soundBytes 的 ByteArray 物件中：

```
var mic:Microphone = Microphone.getMicrophone();
mic.setSilenceLevel(0, DELAY_LENGTH);
mic.addEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);
function micSampleDataHandler(event:SampleDataEvent):void {
    while(event.data.bytesAvailable) {
        var sample:Number = event.data.readFloat();
        soundBytes.writeFloat(sample);
    }
}
```

您可以重複使用取樣位元組來播放 Sound 物件的音效。如果您這樣做，就應該將 Microphone 物件的 rate 屬性設定成 44，這是 Sound 物件使用的取樣頻率 (您也可以將較低頻率捕捉到麥克風取樣轉換成 Sound 物件需要的 44 kHz 頻率)。另外請記住一點，Microphone 物件捕捉的是單聲道取樣，但 Sound 物件使用立體聲，所以您應該將 Microphone 物件捕捉到的每一個位元組寫入 Sound 物件兩次。以下範例會捕捉 4 秒的麥克風資料，然後使用 Sound 物件來播放：

```
const DELAY_LENGTH:int = 4000;
var mic:Microphone = Microphone.getMicrophone();
mic.setSilenceLevel(0, DELAY_LENGTH);
mic.gain = 100;
mic.rate = 44;
mic.addEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);

var timer:Timer = new Timer(DELAY_LENGTH);
timer.addEventListener(TimerEvent.TIMER, timerHandler);
timer.start();

function micSampleDataHandler(event:SampleDataEvent):void
{
    while(event.data.bytesAvailable)
    {
        var sample:Number = event.data.readFloat();
        soundBytes.writeFloat(sample);
    }
}
var sound:Sound = new Sound();
var channel:SoundChannel;
function timerHandler(event:TimerEvent):void
{
    mic.removeEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);
    timer.stop();
    soundBytes.position = 0;
```

```
    sound.addEventListener(SampleDataEvent.SAMPLE_DATA, playbackSampleHandler);
    channel.addEventListener( Event.SOUND_COMPLETE, playbackComplete );
    channel = sound.play();
}

function playbackSampleHandler(event:SampleDataEvent):void
{
    for (var i:int = 0; i < 8192 && soundBytes.bytesAvailable > 0; i++)
    {
        trace(sample);
        var sample:Number = soundBytes.readFloat();
        event.data.writeFloat(sample);
        event.data.writeFloat(sample);
    }
}

function playbackComplete( event:Event ):void
{
    trace( "Playback finished.");
}
```

如需有關從聲音取樣資料播放聲音的詳細資訊，請參閱第 381 頁「使用動態產生的音效」。

聲音範例：Podcast Player

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Podcast 是在網際網路上以點播或訂閱方式散佈的聲音檔案。Podcast 通常會發佈做為一系列中的某一部分，而該系列又稱為 Podcast 頻道。因為 Podcast 的廣播節目可以在任何地方持續一分鐘到許多小時，所以在播放時通常會以串流的方式進行。Podcast 廣播節目 (也稱為項目) 通常是以 MP3 檔案格式傳送。雖然視訊 Podcast 也很流行，但是此樣本應用程式只會播放使用 MP3 檔案的音效 Podcast。

此範例不是具有完整功能的 Podcast Aggregator 應用程式。例如，它並不會管理特定 Podcast 的訂閱，也不會在下一次應用程式執行時，記住使用者已聽過哪些 Podcast。它可以做為功能更加完整之 Podcast Aggregator 的起點。

Podcast Player 範例將說明下列 ActionScript 程式設計技術：

- 讀取外部 RSS Feed 並解析其 XML 內容
- 建立 SoundFacade 類別以簡化聲音檔案的載入和播放作業
- 顯示聲音播放進度
- 暫停和繼續聲音播放

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。您可以在 Samples/PodcastPlayer 資料夾中找到 Podcast Player 應用程式檔案，此應用程式是由下列檔案組成：

檔案	說明
PodcastPlayer.mxml 或 PodcastPlayer.fla	應用程式使用者介面，在 Flash 中為 FLA，在 Flex 中為 MXML。
comp/example/programmingas3/podcastplayer/PodcastPlayer.as	文件類別，包含 Podcast Player 的使用者介面邏輯 (僅適用於 Flash)。
SoundPlayer.mxml	可顯示播放按鈕和進度列，並控制聲音播放的 MXML 組件，僅適用於 Flex。
main.css	應用程式使用者介面的樣式 (僅適用於 Flex)。
images/	製作按鈕樣式的圖示 (僅適用於 Flex)。
comp/example/programmingas3/podcastplayer/SoundPlayer.as	SoundPlayer 影片片段元件的類別，包含聲音播放程式的使用者介面邏輯 (僅適用於 Flash)。
comp/example/programmingas3/podcastplayer/PlayButtonRenderer.as	在資料格儲存格中顯示播放按鈕的自訂儲存格輸出器 (僅適用於 Flash)。
com/example/programmingas3/podcastplayer/RSSBase.as	基底類別，為 RSSChannel 類別與 RSSItem 類別提供一般的屬性與方法。
com/example/programmingas3/podcastplayer/RSSChannel.as	ActionScript 類別，用以儲存 RSS 頻道的資料。
com/example/programmingas3/podcastplayer/RSSItem.as	ActionScript 類別，用以儲存 RSS 項目的資料。
com/example/programmingas3/podcastplayer/SoundFacade.as	應用程式的主要 ActionScript 類別。它會封裝 Sound 類別與 SoundChannel 類別的方法與事件，並增加對暫停和繼續播放的支援。
com/example/programmingas3/podcastplayer/URLService.as	從遠端 URL 擷取資料的 ActionScript 類別。
playerconfig.xml	包含 RSS Feed 之清單的 XML 檔案，這些 RSS Feed 都代表 Podcast 頻道。
comp/example/programmingas3/utils/DateUtil.as	用來簡單設定日期格式的類別 (僅適用於 Flash)。

讀取 Podcast 頻道的 RSS 資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Podcast Player 應用程式會從讀取一些 Podcast 頻道及其廣播節目的資訊開始。

1. 首先，應用程式會讀取 XML 設定檔 (此檔案包含 Podcast 頻道的清單)，並向使用者顯示頻道清單。
2. 當使用者選取其中一個 Podcast 頻道時，它會讀取該頻道的 RSS Feed，並顯示頻道節目的清單。

此範例會使用 URLLoader 公用程式類別，從遠端位置或本機檔案擷取文字資料。Podcast Player 首先會建立 URLLoader 物件，以從 playerconfig.xml 檔案取得 XML 格式的 RSS Feed 清單。接著，當使用者從清單選取特定的內容時，就會建立新的 URLLoader 物件，以便從該 Feed 的 URL 讀取 RSS 資料。

使用 SoundFacade 類別簡化聲音載入和播放作業

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 的聲音架構非常強大，但卻也複雜。只需要基本聲音載入和播放功能的應用程式可以使用一個類別，藉由提供一組較簡單的方法呼叫與事件，隱藏某些複雜度。在軟體設計模式的世界中，這種類別就稱為「外觀」。

SoundFacade 類別會提供單一介面，以執行下列工作：

- 使用 Sound 物件、SoundLoaderContext 物件以及 SoundMixer 類別載入聲音檔案。
- 使用 Sound 物件和 SoundChannel 物件播放聲音檔案
- 傳送播放進度事件
- 使用 Sound 物件和 SoundChannel 物件來暫停和繼續播放聲音。

SoundFacade 類別會嘗試以較低的複雜度，提供 ActionScript 聲音類別的大部分功能。

下列程式碼會顯示類別宣告、類別屬性以及 SoundFacade() 建構函式方法：

```
public class SoundFacade extends EventDispatcher
{
    public var s:Sound;
    public var sc:SoundChannel;
    public var url:String;
    public var bufferTime:int = 1000;

    public var isLoaded:Boolean = false;
    public var isReadyToPlay:Boolean = false;
    public var isPlaying:Boolean = false;
    public var isStreaming:Boolean = true;
    public var autoLoad:Boolean = true;
    public var autoPlay:Boolean = true;

    public var pausePosition:int = 0;

    public static const PLAY_PROGRESS:String = "playProgress";
    public var progressInterval:int = 1000;
    public var playTimer:Timer;

    public function SoundFacade(soundUrl:String, autoLoad:Boolean = true,
                                autoPlay:Boolean = true, streaming:Boolean = true,
                                bufferTime:int = -1):void
    {
        this.url = soundUrl;
```

```
        // Sets Boolean values that determine the behavior of this object
        this.autoLoad = autoLoad;
        this.autoPlay = autoPlay;
        this.isStreaming = streaming;

        // Defaults to the global bufferTime value
        if (bufferTime < 0)
        {
            bufferTime = SoundMixer.bufferTime;
        }

        // Keeps buffer time reasonable, between 0 and 30 seconds
        this.bufferTime = Math.min(Math.max(0, bufferTime), 30000);

        if (autoLoad)
        {
            load();
        }
    }
```

SoundFacade 類別會擴充 EventDispatcher 類別，讓它可以傳送屬於自己的事件。該類別程式碼首先會宣告 Sound 物件與 SoundChannel 物件的屬性。此類別也會儲存聲音檔案的 URL 值，以及串流聲音時要使用的 bufferTime 屬性值。除此之外，它還會接受一些會影響載入和播放行為的 Boolean 參數值：

- autoLoad 參數會告訴物件，只要一建立物件就開始載入聲音。
- autoPlay 參數會指出只要載入足夠的聲音資料就開始播放聲音。如果這是串流聲音，則只要已載入 bufferTime 屬性所指定的足夠資料就會開始播放。
- streaming 參數會指出在完成載入之前，聲音檔案可以開始播放。

bufferTime 參數預設值為 -1。如果建構函式方法在 bufferTime 參數中偵測到負值，它會將 bufferTime 屬性設定為 SoundMixer.bufferTime 的值。這可以讓應用程式將預設值設定為所需的全域 SoundMixer.bufferTime 值。

如果 autoLoad 參數已設定為 true，則建構函式方法會立即呼叫下列 load() 方法，開始載入聲音檔案：

```
public function load():void
{
    if (this.isPlaying)
    {
        this.stop();
        this.s.close();
    }
    this.isLoaded = false;

    this.s = new Sound();

    this.s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
    this.s.addEventListener(Event.OPEN, onLoadOpen);
    this.s.addEventListener(Event.COMPLETE, onLoadComplete);
    this.s.addEventListener(Event.ID3, onID3);
    this.s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
    this.s.addEventListener(SecurityErrorEvent.SECURITY_ERROR, onIOError);

    var req:URLRequest = new URLRequest(this.url);

    var context:SoundLoaderContext = new SoundLoaderContext(this.bufferTime, true);
    this.s.load(req, context);
}
```

load() 方法會建立新的 Sound 物件，然後為所有重要的聲音事件增加偵聽程式。接著，它會告訴 Sound 物件使用 SoundLoaderContext 物件傳入 bufferTime 值，以載入聲音檔案。

由於您可以變更 url 屬性，因此可以使用 SoundFacade 實體來連續播放不同的聲音檔案：只需變更 url 屬性，再呼叫 load() 方法，就會載入新的聲音檔案。

下列三個事件偵聽程式方法將說明 SoundFacade 物件如何追蹤載入的進度，並決定何時開始播放聲音：

```
public function onLoadOpen(event:Event):void
{
    if (this.isStreaming)
    {
        this.isReadyToPlay = true;
        if (autoPlay)
        {
            this.play();
        }
    }
    this.dispatchEvent(event.clone());
}

public function onLoadProgress(event:ProgressEvent):void
{
    this.dispatchEvent(event.clone());
}

public function onLoadComplete(event:Event):void
{
    this.isReadyToPlay = true;
    this.isLoaded = true;
    this.dispatchEvent(evt.clone());

    if (autoPlay && !isPlaying)
    {
        play();
    }
}
```

onLoadOpen() 方法會在聲音載入作業開始時執行。如果可以在串流模式中播放聲音，則 onLoadComplete() 方法會立即將 isReadyToPlay 旗標設定為 true。isReadyToPlay 旗標會決定應用程式是否可以開始聲音播放，也許回應如按一下「播放」按鈕等使用者動作。由於 SoundChannel 類別會管理聲音資料的緩衝，因此在呼叫 play() 方法之前，不需要明確檢查是否已載入足夠的資料。

onLoadProgress() 方法會在載入程序期間定期執行。它會直接傳送其 ProgressEvent 物件之副本，供使用此 SoundFacade 物件的程式碼使用。

已完全載入聲音資料後，就會執行 onLoadComplete() 方法，並視需要為非串流聲音呼叫 play() 方法。play() 方法本身如下所示。

```
public function play(pos:int = 0):void
{
    if (!this.isPlaying)
    {
        if (this.isReadyToPlay)
        {
            this.sc = this.s.play(pos);
            this.sc.addEventListener(Event.SOUND_COMPLETE, onPlayComplete);
            this.isPlaying = true;

            this.playTimer = new Timer(this.progressInterval);
            this.playTimer.addEventListener(TimerEvent.TIMER, onPlayTimer);
            this.playTimer.start();
        }
    }
}
```

play() 方法會在聲音可以播放時呼叫 Sound.play() 方法。產生的 SoundChannel 物件是儲存在 sc 屬性中。play() 方法接著會建立 Timer 物件，該物件將用以定期傳送播放進度事件。

顯示播放進度

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

建立 Timer 物件以驅動播放監視功能是一項複雜的作業，因此您最好只撰寫一次程式碼就好。將此 Timer 邏輯封裝在如 SoundFacade 等可重複使用的類別中，可讓應用程式在聲音載入和播放時，偵聽相同種類的進度事件。

SoundFacade.play() 方法所建立的 Timer 物件每秒都會傳送 TimerEvent 實體。每當有新的 TimerEvent 到達時，下列 onPlayTimer() 方法就會執行：

```
public function onPlayTimer(event:TimerEvent):void
{
    var estimatedLength:int =
        Math.ceil(this.s.length / (this.s.bytesLoaded / this.s.bytesTotal));
    var progEvent:ProgressEvent =
        new ProgressEvent(PLAY_PROGRESS, false, false, this.sc.position, estimatedLength);
    this.dispatchEvent(progEvent);
}
```

onPlayTimer() 方法會實作在第 384 頁「監視播放作業」一節中描述的大小估算技術。接著，它會建立具有 SoundFacade.PLAY_PROGRESS 事件類型的新 ProgressEvent 實體，其中 bytesLoaded 屬性會設定為 SoundChannel 物件的目前位置，而且 bytesTotal 屬性會設定為聲音資料的估算長度。

暫停和繼續播放

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

之前顯示的 SoundFacade.play() 方法可接受對應至聲音資料中開始位置的 pos 參數。如果 pos 值為零，就會從一開始播放聲音。

SoundFacade.stop() 方法也會接受 pos 參數，如下所示：

```
public function stop(pos:int = 0):void
{
    if (this.isPlaying)
    {
        this.pausePosition = pos;
        this.sc.stop();
        this.playTimer.stop();
        this.isPlaying = false;
    }
}
```

每當呼叫 SoundFacade.stop() 方法時，它就會設定 pausePosition 屬性，這樣一來，如果使用者想要繼續播放相同聲音時，應用程式就會知道要放置播放磁頭的位置。

下面所顯示的 SoundFacade.pause() 與 SoundFacade.resume() 方法會分別叫用 SoundFacade.stop() 與 SoundFacade.play() 方法，每一次都會傳遞 pos 參數值。

```
public function pause():void
{
    stop(this.sc.position);
}

public function resume():void
{
    play(this.pausePosition);
}
```

pause() 方法會將目前的 SoundChannel.position 值傳遞給 play() 方法，它會將該值儲存在 pausePosition 屬性中。resume() 方法會使用 pausePosition 值做為起點，再次開始播放相同的聲音。

擴充 Podcast Player 範例

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

此範例顯示極為精簡的 Podcast Player，以展示可重複使用的 SoundFacade 類別的用法。您可以增加其它功能以增強此應用程式的有用性，其中包括下列項目：

- 將每個節目的 Feed 清單與使用資訊儲存在 SharedObject 實體中，以利下次使用者執行應用程式時使用。
- 讓使用者將自己的 RSS Feed 加入 Podcast 頻道的清單。
- 當使用者停止或離開節目時，記住播放磁頭的位置。這樣一來，下次使用者執行應用程式時就可以從該點重新啟動該節目。
- 下載節目的 MP3 檔案，當使用者未連線到網際網路時便可以離線聆賞。
- 增加訂閱功能，定期檢查在 Podcast 頻道中的新節目並自動更新節目清單。
- 使用如 Odeo.com 等提供 Podcast 廣播服務的網站所提供的 API，增加 Podcast 搜尋和瀏覽功能。

第 25 章 使用視訊

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash 視訊是網際網路上所使用的其中一項傑出技術。不過，傳統視訊的表現方式 (在矩形螢幕中，包含進度列及底下的控制按鈕) 是唯一可能的視訊用法。透過 ActionScript，您可以對視訊載入、顯示和播放方式做精密的調整並加以控制。

視訊的基本觀念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Adobe® Flash® Player 和 Adobe® AIR™ 的其中一項重要功能，就是能夠以操作其他視訊內容 (如影像、動畫、文字等) 的相同方式來使用 ActionScript 顯示和處理視訊資訊。當您在 Adobe Flash CS4 Professional 中建立 Flash 視訊 (FLV) 檔案時，可以選取包含常用播放控制項的外觀元素。不過，您不需要將自己侷限在可用的選項。透過 ActionScript，您可以對視訊載入、顯示和播放方式做精密的調整並加以控制。這表示您可以建立自己的視訊播放程式外觀元素，或是以任何您想要與傳統稍有不同的方式來使用視訊。在 ActionScript 中使用視訊涉及處理數個類別的組合：

- Video 類別：在「舞台」上的典型視訊內容方塊是 Video 類別的實體。Video 類別是一種顯示物件，因此可以使用套用至其它顯示物件的相同技術來操作它，例如定位、套用變形、套用濾鏡與混合模式等等。
- StageVideo 類別：Video 類別通常會使用軟體解碼與顯示。裝置上可以使用 GPU 硬體加速時，您的應用程式可以切換至 StageVideo 類別，善加利用硬體加速呈現方式。StageVideo API 包括一組事件，用於告訴您的程式碼在 StageVideo 與 Video 物件之間切換的時機。舞台視訊會對視訊播放強制實行一些次要限制。若您的應用程式接受那些限制，請實作 StageVideo API。請參閱第 437 頁「原則與限制」。
- NetStream 類別：當您載入由 ActionScript 控制的視訊檔案時，NetStream 實體代表視訊內容的來源，在此例中，則為視訊資料的串流。使用 NetStream 實體也牽涉到使用 NetConnection 物件，該物件會連線到視訊檔案，就像是提供視訊資料的通道。
- Camera 類別：當您從連接到使用者電腦的攝影機處理視訊資料時，Camera 實體代表視訊內容的來源，也就是使用者的攝影機與它所提供的視訊資料。這是 Flash Player 11.4 和 AIR 3.4 中的新功能，您可以使用攝影機來輸入 StageVideo。

要載入外部視訊時，您可以從標準網站伺服器載入檔案以進行漸進式下載，也可以使用如 Adobe 的 Flash® Media Server 等專用伺服器所傳遞的串流視訊。

重要概念與術語

提示點 可以放置在視訊檔案中某個特定時間點的標記。例如，做為書籤以供尋找該時間點，或是提供與該時間點相關的其它資料。

編碼 以某個格式取得視訊資料並將它轉換為另一個視訊資料格式的程序。例如，取得高解析度的來源視訊，並將它轉換為適用於網際網路傳遞的格式。

影格 單一片段的視訊資訊。每個影格就像靜態影像一樣，顯示某個時間點的快照。透過以高速連續播放影格的方式，就會產生動態的感覺。

關鍵影格 包含影格之完整資訊的視訊影格。在關鍵影格後面的其它影格只包含它們與關鍵影格有何不同的資訊，並不包含整個影格的資訊。

中繼資料 內嵌在視訊檔案中，並且在視訊載入完成時擷取的視訊檔案資訊。

漸進式下載 從標準網站伺服器傳遞視訊檔案時，會使用漸進式下載的方式載入視訊資料，這表示載入視訊資訊的作業會以連續的方式進行。其好處在於可以在下載整個檔案之前，就開始播放視訊。不過，您無法往前跳到尚未載入的視訊部分。

串流 它是漸進式下載的替代方案，可以使用特殊的視訊伺服器，透過名為「串流」的技術 (有時稱為「真實串流」) 在網際網路上傳遞視訊。透過串流方式，檢視者的電腦永遠不會一次下載整個視訊。為了加速下載時間，不論何時，電腦都只需要總視訊資訊的其中一部分。由於特殊的伺服器會控制視訊內容的傳遞作業，因此隨時都可以存取視訊的任何部分，而不必等到先下載後才能加以存取。

瞭解視訊格式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

除了 Adobe FLV 視訊格式以外，Flash Player 和 Adobe AIR 也都支援視訊編碼為 H.264 以及音效編碼為 HE-AAC 的 MPEG-4 標準檔案格式。這些格式會在較低的位元速率提供高品質視訊串流。開發人員可以善用業界的標準工具 (包括 Adobe Premiere Pro 和 Adobe After Effects) 建立及提供令人驚豔的視訊內容。

類型	格式	容器
視訊	H.264	MPEG-4：MP4、M4V、F4V、3GPP
視訊	Sorenson Spark	FLV 檔
視訊	ON2 VP6	FLV 檔
音訊	AAC+ / HE-AAC / AAC v1 / AAC v2	MPEG-4：MP4、M4V、F4V、3GPP
音訊	Mp3	Mp3
音訊	Nellymoser	FLV 檔
音訊	Speex	FLV 檔

更多說明主題

Flash Media Server：支援的轉碼器

Adobe HTTP 動態串流

行動裝置的編碼視訊

在 Android 上的 AIR 可以解碼範圍廣泛的 H.264 視訊。不過，只有一小部分的 H.264 視訊子集能夠流暢地在行動電話上播放。這是因為有許多行動電話都受到處理能力的限制。適用於行動的 Adobe Flash Player 可以使用內建硬體加速來解碼 H.264 視訊。此解碼可確保在較不耗電的狀態下獲得較佳的品質。

H.264 標準支援一些編碼技術。只有高規格的裝置可流暢地播放具有複雜描述檔與色階的視訊。然而，大多數的裝置都可以播放以基線描述檔編碼的視訊。在行動裝置上，硬體加速可供部分技術使用。描述檔與層級參數會定義編碼器所使用的此編碼技術與設定子集。對於開發人員而言，它會轉譯成以所選解析度編碼的視訊，這在大部分的裝置上都能順暢播放。

雖然從硬體加速獲益的解析度會隨不同裝置而異，但是大部分的裝置都支援下列標準解析度。

外觀比例	建議的解析度		
4:3	640 × 480	512 × 384	480 × 360
16:9	640 × 360	512 x 288	480 × 272

備註：Flash Player 支援 H.264 標準的每個層級與描述檔。遵守這些建議可確保大部分裝置可進行硬體加速，並獲得較佳的使用者經驗。這些建議不具有強制性。

如需在 Adobe Media Encoder CS5 中的詳細討論與編碼設定，請參閱為行動裝置上 Flash Player 10.1 編碼 H.264 視訊的建議。

備註：在 iOS 上，使用 Video 類別僅可播放使用 Sorenson Spark and On2 VP6 編碼的視訊。您可使用 flash.net.navigateToURL() 功能來啟動視訊的 URL，以便在裝置視訊播放器播放 H.264 編碼視訊。您也可使用在 StageWebView 物件中顯示 HTML 頁面的 <video> 標籤來播放 H.264 視訊。

Flash Player 和 AIR 與已編碼之視訊檔案的相容性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 7 支援使用 Sorenson™ Spark™ 視訊編碼的 FLV 檔。Flash Player 8 支援在 Flash Professional 8 中，以 Sorenson Spark 和 On2 VP6 編碼器進行編碼的 FLV 檔 (其中 On2 VP6 視訊轉碼器能支援 Alpha 色版)。

Flash Player 9.0.115.0 和更新版本都支援衍生自標準 MPEG-4 容器格式的檔案，其中包括 F4V、MP4、M4A、MOV、MP4V、3GP 和 3G2 (如果這些檔案包含 H.264 編碼視訊或 HE-AAC v2 編碼音效或兩者的話)。相較於 Sorenson 或 On2 中的相同編碼描述檔，H.264 會在較低位元速率提供較高品質視訊。HE-AAC v2 是 AAC 的延伸，採用了 MPEG-4 視訊標準所定義的標準音訊格式。HE-AAC v2 會使用光譜波段複製 (SBR) 和參數立體聲 (PS) 技術以增加低位元速率的編碼效率。

下表列出支援的轉碼器，也一併列出對應的 SWF 檔格式及播放這些檔案時所需的 Flash Player 和 AIR 版本：

轉碼器	SWF 檔案格式版本 (支援的最舊發佈版本)	Flash Player 和 AIR (播放所需的最舊版本)
Sorenson Spark	6	Flash Player 6、Flash Lite 3
On2 VP6	6	Flash Player 8、Flash Lite 3 只有 Flash Player 8 及更新版本支援 On2 VP6 視訊的發佈和播放。
H.264 (MPEG-4 Part 10)	9	Flash Player 9 更新 3、AIR 1.0
ADPCM	6	Flash Player 6、Flash Lite 3
Mp3	6	Flash Player 6、Flash Lite 3
AAC (MPEG-4 Part 3)	9	Flash Player 9 更新 3、AIR 1.0
Speex (audio)	10	Flash Player 10、AIR 1.5
Nellymoser	6	Flash Player 6

瞭解 Adobe F4V 和 FLV 視訊檔案格式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Adobe 提供了 F4V 和 FLV 視訊檔案格式，可用來為 Flash Player 和 AIR 提供串流內容。如需這些視訊檔案格式的完整說明，請前往 www.adobe.com/go/video_file_format_tw。

F4V 視訊檔案格式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

從 Flash Player 更新 3 (9.0.115.0) 和 AIR 1.0 開始，Flash Player 和 AIR 都支援 Adobe F4V 視訊格式 (以 ISO MP4 格式為基礎)，這個格式的子集可支援不同的功能。Flash Player 需要有效的 F4V 檔，才能以下列其中一個最上層方塊開始：

- ftyp

  ftyp 方塊可以識別某一程式要播放特定檔案格式所需的功能。

- moov

  moov 方塊實際上就是 F4V 檔的標頭。它包含其它一個或多個方塊，而這些方塊則又包含其它定義 F4V 資料結構的方塊。F4V 檔必須只包含一個 moov 方塊。

- mdat

  mdat 方塊包含 F4V 檔的資料負荷。FV 檔只包含一個 mdat 方塊。moov 方塊也必須出現在檔案中，因為 mdat 方塊無法進行自我辨識。

F4V 檔支援位元組順序為 Big-Endian 的多位元組整數，其中最高位元組在前，儲存在最低位址。

FLV 視訊檔案格式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Adobe FLV 檔格式包含了經過編碼的音訊和視訊資料，可以使用 Flash Player 播放。您可以使用編碼器 (例如 Adobe Media Encoder 或 Sorenson™ Squeeze) 將 QuickTime 或 Windows Media 視訊檔轉換成 FLV 檔。

備註：若要建立 FLV 檔，您可以將視訊匯入 Flash 中，並將其匯出成 FLV 檔。您可以使用 FLV Export 外掛程式，透過支援的視訊編輯應用程式匯出 FLV 檔。若要從網站伺服器載入 FLV 檔，請向網站伺服器註冊檔案的副檔名和 MIME 類型。請查看網站伺服器的說明文件。FLV 檔的 MIME 類型是 video/x-flv。如需詳細資訊，請參閱第 430 頁 「關於設定能在伺服器上使用的 FLV 檔」。

如需有關 FLV 檔的詳細資訊，請參閱第 430 頁 「視訊檔案的進階主題」。

外部與內嵌視訊的比較

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用外部視訊檔案時，可以提供某些無法在使用匯入的視訊時利用的功能：

- 您可以在應用程式中使用較長的視訊片段，而不會造成播放遲緩的現象。外部視訊檔案會使用快取的記憶體，這表示大型檔案會分成數個小部分儲存，並透過動態方式存取。因此，和內嵌視訊檔案相比，F4V 和 FLV 檔所需的記憶體更少。
- 外部視訊檔案的影格速率不一定要和播放它的 SWF 檔相同。例如，您可以將 SWF 檔的影格速度設定為 30 每秒影格數 (fps)，同時將視訊的影格速度設定為 21 fps。比起內嵌視訊，這項設定能讓您更容易控制這個視訊，確保能夠順暢地播放。這項設定也能讓您以不同的影格速率播放視訊檔案，而不需要改變現有的 SWF 檔內容。
- 如果是外部視訊檔案，在載入視訊期間播放 SWF 內容便不會中斷。匯入的視訊檔案有時會中斷文件播放，以便執行某些功能 (例如，存取 CD-ROM 光碟機)。視訊檔案可以獨立執行 SWF 內容的功能，而不需中斷播放。
- 使用外部 FLV 檔時，因為您可以使用事件處理常式存取視訊的中繼資料，所以為視訊內容加上註解的工作也變得較為容易。

瞭解 Video 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Video 類別可讓您在應用程式中顯示即時串流的視訊，而不需要將它內嵌在 SWF 檔中。您可以使用 Camera.getCamera() 方法，捕捉並播放即時視訊。此外，您也可以使用 Video 類別，透過 HTTP 或從本機檔案系統播放視訊檔案。在專案中使用 Video 有數種方式：

- 使用 NetConnection 與 NetStream 類別，以動態方式載入視訊檔案，並在 Video 物件中顯示視訊。
- 從使用者的攝影機擷取輸入。如需詳細資訊，請參閱第 443 頁 「使用攝影機」。
- 使用 FLVPlayback 組件。
- 使用 VideoDisplay 控制項。

備註：在 「舞台」上之 Video 物件的實體是 Video 類別的實體。

即使 Video 類別在 flash.media 套件中，它仍然是繼承自 flash.display.DisplayObject 類別，因此所有的顯示物件功能 (如矩陣變形和濾鏡) 也會套用至 Video 實體。

如需詳細資訊，請參閱第 146 頁 「操作顯示物件」、第 177 頁 「處理幾何」和第 226 頁 「以濾鏡處理顯示物件」。

載入視訊檔案

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用 NetStream 與 NetConnection 類別載入視訊需要多個步驟。就最佳作法而言，應以指定的順序將 Video 物件加入顯示清單，接著將 NetStream 物件附加到 Video 實體，然後呼叫 NetStream 物件的 play() 方法：

1. 建立 NetConnection 物件。如果您要連接至本機視訊檔案或不使用伺服器 (例如 Adobe's Flash Media Server 2) 的視訊檔案，請將 null 傳遞至 connect() 方法，從 HTTP 位址或本機磁碟播放視訊檔案。如果要連線到伺服器，請將這個參數設定為包含伺服器上之視訊檔案的應用程式 URI。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
```

2. 建立新的 Video 物件以顯示視訊並將該物件加入舞台顯示清單，如下列程式碼片段所示：

```
var vid:Video = new Video();
addChild(vid);
```

3. 建立 NetStream 物件，以傳遞 NetConnection 物件做為建構函式的引數。下列程式碼片段會將 NetStream 物件連線至 NetConnection 實體，並且為串流設定事件處理常式：

```
var ns:NetStream = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS,netStatusHandler);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);

function netStatusHandler(event:NetStatusEvent):void
{
    // handle netStatus events, described later
}

function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}
```

4 使用 Video 物件的 attachNetStream() 方法，將 NetStream 物件附加至 Video 物件，如下列程式碼片段所示：

```
vid.attachNetStream(ns);
```

5 將視訊檔案 URL 當做引數使用，呼叫 NetStream 物件的 play() 方法，以開始播放視訊。下列程式碼片段會在與 SWF 檔案相同的目錄中載入並播放名為「video.mp4」的視訊檔案：

```
ns.play("video.mp4");
```

更多說明主題

Flex：Spark VideoPlayer 控制項

spark.components.VideoDisplay

控制視訊播放

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

NetStream 類別可提供控制視訊播放作業的四個主要方法：

pause()：暫停播放視訊串流。如果已經暫停視訊，呼叫這個方法就不會有任何動作。

resume()：繼續播放被暫停的視訊串流。如果視訊已經在播放，呼叫這個方法就不會有任何動作。

seek()：尋找離指定位置 (與串流開頭之間的位移，以秒為單位) 最近的關鍵影格。

togglePause()：暫停或繼續播放串流。

備註：這個類別沒有 stop() 方法。若要停止串流，您必須暫停播放並搜尋至視訊串流的開頭。

備註：play() 方法並不會繼續播放，它會用於載入視訊檔案。

下列範例將示範如何使用幾個不同的按鈕來控制視訊。若要執行下列範例，請建立新的文件，並在工作區中增加四個按鈕實體 (pauseBtn、playBtn、stopBtn 以及 togglePauseBtn)：

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

pauseBtn.addEventListener(MouseEvent.CLICK, pauseHandler);
playBtn.addEventListener(MouseEvent.CLICK, playHandler);
stopBtn.addEventListener(MouseEvent.CLICK, stopHandler);
togglePauseBtn.addEventListener(MouseEvent.CLICK, togglePauseHandler);

function pauseHandler(event:MouseEvent):void
{
    ns.pause();
}
function playHandler(event:MouseEvent):void
{
    ns.resume();
}
function stopHandler(event:MouseEvent):void
{
    // Pause the stream and move the playhead back to
    // the beginning of the stream.
    ns.pause();
    ns.seek(0);
}
function togglePauseHandler(event:MouseEvent):void
{
    ns.togglePause();
}
```

視訊正在播放時，按一下 pauseBtn 按鈕實體會讓視訊檔案暫停播放。如果已經暫停視訊，按一下這個按鈕就不會有任何作用。如果之前已經暫停播放，按一下 playBtn 按鈕實體就會使視訊繼續播放；否則如果視訊已經播放，此按鈕就不會有任何作用。

偵測視訊串流的結尾

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

為了偵聽視訊串流的開頭與結尾，您必須將事件偵聽程式加入 NetStream 實體以偵聽 netStatus 事件。下列程式碼將示範如何偵聽整個視訊播放作業的各種程式碼：

```
ns.addEventListener(NetStatusEvent.NET_STATUS, statusHandler);
function statusHandler(event:NetStatusEvent):void
{
    trace(event.info.code)
}
```

上述程式碼會產生下列輸出：

```
NetStream.Play.Start
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Flush
NetStream.Play.Stop
NetStream.Buffer.Empty
NetStream.Buffer.Flush
```

您要特別偵聽的兩個程式碼是 「NetStream.Play.Start」與 「NetStream.Play.Stop」，分別是指視訊播放的開頭與結尾。下列程式碼片段會使用 switch 陳述式篩選兩個程式碼並追蹤訊息：

```
function statusHandler(event:NetStatusEvent):void
{
    switch (event.info.code)
    {
        case "NetStream.Play.Start":
            trace("Start [" + ns.time.toFixed(3) + " seconds]");
            break;
        case "NetStream.Play.Stop":
            trace("Stop [" + ns.time.toFixed(3) + " seconds]");
            break;
    }
}
```

透過偵聽 netStatus 事件 (NetStatusEvent.NET_STATUS)，您可以建立視訊播放程式，以便在目前的視訊完成播放時，立即載入播放清單中的下一個視訊。

以全螢幕模式播放視訊

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 和 AIR 可以讓您為視訊播放建立全螢幕應用程式，並支援將視訊放大為全螢幕。

對於在桌上型電腦全螢幕模式下執行的 AIR 內容，系統的螢幕保護程式和省電選項會在播放期間停用，直到視訊輸入停止或使用者結束全螢幕模式為止。

如需有關使用全螢幕模式的完整詳細資訊，請參閱第 141 頁 「使用全螢幕模式」。

為瀏覽器中的 **Flash Player** 啟用全螢幕模式

您必須先透過 「發佈」範本，為應用程式啟用全螢幕模式，瀏覽器中的 Flash Player 才能使用全螢幕模式。允許全螢幕的範本包括內含 allowFullScreen 參數的 <object> 和 <embed> 標籤。下列範例會顯示 <embed> 標籤中的 allowFullScreen 參數。

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    id="fullScreen" width="100%" height="100%"
    codebase="http://fpdownload.macromedia.com/get/flashplayer/current/swflash.cab">
    ...
    <param name="allowFullScreen" value="true" />
    <embed src="fullScreen.swf" allowFullScreen="true" quality="high" bgcolor="#869ca7"
        width="100%" height="100%" name="fullScreen" align="middle"
        play="true"
        loop="false"
        quality="high"
        allowScriptAccess="sameDomain"
        type="application/x-shockwave-flash"
        pluginspage="http://www.adobe.com/go/getflashplayer">
    </embed>
    ...
</object>
```

在 Flash 中選取 「檔案 > 發佈設定」，然後在 「發佈設定」對話方塊的 「HTML」索引標籤上，選取 「僅限 Flash - 允許全螢幕」範本。

在 Flex 中，確認 HTML 範本包含支援全螢幕的 <object> 和 <embed> 標籤。

起始全螢幕模式

對於在瀏覽器中執行的 Flash Player 內容，按下滑鼠或按鍵都可以為視訊啟動全螢幕模式。例如，您可以在使用者按一下標示為 「全螢幕」的按鈕時，或者從快顯選單選取 「全螢幕」命令時，起始全螢幕模式。若要回應使用者，請將事件偵聽程式加入至發生動作的對象物件。在下列程式碼中，會將事件偵聽程式加入至按鈕，使用者按一下按鈕即可進入全螢幕模式：

```
var fullScreenButton:Button = new Button();
fullScreenButton.label = "Full Screen";
addChild(fullScreenButton);
fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);

function fullScreenButtonHandler(event:MouseEvent)
{
    stage.displayState = StageDisplayState.FULL_SCREEN;

}
```

程式碼會將 Stage.displayState 屬性設定為 StageDisplayState.FULL_SCREEN，以啟動全螢幕模式。其中程式碼會將整個舞台放大為全螢幕，並且調整視訊的縮放比例，使其佔據整個舞台。

fullScreenSourceRect 屬性可讓您指定要放大為全螢幕的特定舞台區域。首先，請定義您要放大為全螢幕的矩形區域。然後，再將它指定給 Stage.fullScreenSourceRect 屬性。fullScreenButtonHandler() 功能的這個版本隨即會加入額外兩行程式碼，這兩行程式碼只會將視訊放大為全螢幕。

```
private function fullScreenButtonHandler(event:MouseEvent)
{
    var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width, video.height);
    stage.fullScreenSourceRect = screenRectangle;
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

雖然這個範例會叫用事件處理常式以回應按滑鼠的動作，但 Flash Player 和 AIR 進入全螢幕的技巧是相同的。首先，定義您要縮放的矩形區域，然後設定 Stage.displayState 屬性。如需詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。

以下為完整的範例，當中加入了為視訊建立連線和 NetStream 物件並開始播放視訊的程式碼。

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import fl.controls.Button;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.FullScreenEvent;
    import flash.geom.Rectangle;

    public class FullScreenVideoExample extends Sprite
    {
        var fullScreenButton:Button = new Button();
        var video:Video = new Video();

        public function FullScreenVideoExample()
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("http://www.helpexamples.com/flash/video/water.flv");

            fullScreenButton.x = 100;
            fullScreenButton.y = 270;
            fullScreenButton.label = "Full Screen";
            addChild(fullScreenButton);
            fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);
        }

        private function fullScreenButtonHandler(event:MouseEvent)
        {
            var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width, video.height);
            stage.fullScreenSourceRect = screenRectangle;
            stage.displayState = StageDisplayState.FULL_SCREEN;
        }

        public function onMetaData(infoObject:Object):void
        {
            // stub for callback function
        }
    }
}
```

onMetaData() 是用於處理視訊中繼資料 (如果有的話) 的回呼函數。所謂回呼函數，就是執行階段為了回應某些發生的事件所呼叫的函數。在上述範例中，onMetaData() 函數為虛設常式，可以滿足提供函數時所需的必要條件。如需詳細資訊，請參閱第 414 頁「為中繼資料和提示點撰寫回呼方法」。

離開全螢幕模式

使用者輸入一個鍵盤快速鍵 (如 Escape 鍵)，就可以離開全螢幕模式。您可以在 ActionScript 中將 Stage.displayState 屬性設定為 StageDisplayState.NORMAL，以結束全螢幕模式。下列範例中的程式碼會在 NetStream.Play.Stop netStatus 事件發生時，結束全螢幕模式。

```
videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);

private function netStatusHandler(event:NetStatusEvent)
{
    if(event.info.code == "NetStream.Play.Stop")
        stage.displayState = StageDisplayState.NORMAL;
}
```

全螢幕硬體加速

當您將舞台的矩形區域放大為全螢幕時，Flash Player 或 AIR 會使用硬體加速 (視可用情形和是否已啟用而定)。執行階段會使用電腦中的視訊卡，加快將視訊或部分舞台放大為全螢幕大小的速度。在這些情況下，從 Video 類別 (或 Camera 類別：Flash Player 11.4/AIR 3.4 及更新版本) 切換成 StageVideo 類別，通常對 Flash Player 應用程式效能有利。

如需有關在全螢幕模式中進行硬體加速的詳細資訊，請參閱第 141 頁 「使用全螢幕模式」。如需有關 StageVideo 的詳細資訊，請參閱第 435 頁 「使用硬體加速呈現方式的 StageVideo 類別」。

串流視訊檔案

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要從 Flash Media Server 串流檔案，您可以使用 NetConnection 與 NetStream 類別以連接至遠端伺服器實體並播放指定的串流。若要指定「即時訊息通訊協定」(Real-Time Messaging Protocol，RTMP) 伺服器，您可以將所需的 RTMP URL (例如 「rtmp://localhost/appName/appInstance」) 傳遞給 NetConnection.connect() 方法，而不要傳遞 null。若要從指定的 Flash Media Server 播放即時或錄製的特定串流，您可以為 NetStream.publish() 發佈的即時資料傳遞辨識名稱，或是將要播放的錄製檔案名稱傳遞給 NetStream.play() 方法。

將視訊傳送至伺服器

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如果您要建立更複雜的、涉及視訊或攝影機物件的應用程式，Flash Media Server 可結合串流媒體功能與開發環境，以供建立媒體應用程式並將其傳遞給廣大的觀眾。這種結合讓開發人員能夠建立一些特別的應用程式，如隨選視訊、即時網路事件廣播和 Mp3 串流，以及視訊日誌撰寫、視訊傳訊和多媒體交談環境等。如需詳細資訊，請參閱 Flash Media Server 線上說明文件，網址為 www.adobe.com/go/learn_fms_docs_tw。

瞭解提示點

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 Adobe F4V 或 FLV 視訊檔案的編碼期間，您可以在檔案中內嵌提示點。以往，內嵌提示點的目的，是要為電影放映員提供視覺信號，指出電影膠卷已經快結束。在 Adobe F4V 和 FLV 視訊格式中，當視訊串流內的提示點發生時，可以讓您在應用程式中觸發其它一個或多個動作。

您可以搭配 Flash 視訊使用數種不同的提示點。建立視訊檔案時，您可以使用 ActionScript，與內嵌在視訊檔案中的提示點互動。

- 瀏覽提示點：針對視訊檔案進行編碼時，您可以將瀏覽提示點內嵌在視訊串流和中繼資料封包內。使用瀏覽提示點可以讓使用者搜尋檔案的指定部分。

- 事件提示點：針對視訊檔案進行編碼時，您可以將事件提示點內嵌在視訊串流和中繼資料封包內。您可以編寫程式碼，處理在視訊播放期間，於指定點遭到觸發的事件。
- ActionScript 提示點：只有 Flash FLVPlayback 組件才能使用 ActionScript 提示點。ActionScript 提示點是您使用 ActionScript 程式碼建立及存取的外部提示點。您可以編寫程式碼，觸發關於這個視訊播放的提示點。由於視訊播放程式會分別追蹤這些外部提示點，所以精確度比起內嵌的提示點會較差 (差異可達到十分之一秒)。如果您打算建立使用者會瀏覽至提示點的應用程式，便應該在對檔案進行編碼時，就直接建立並內嵌提示點，而不是使用 ActionScript 提示點。您應該將提示點內嵌在 FLV 檔中，如此才能達到更佳的精確度。

瀏覽提示點會在指定的提示點位置建立關鍵影格，因此您便可以使用程式碼將視訊播放程式的播放磁頭移至該處。您可以在視訊檔案中設定某些特定點，讓使用者可以進行搜尋。例如，這個視訊可能包含了數個章節或段落，而您便可以藉由在視訊中內嵌瀏覽提示點的方式，控制這個視訊。

如需有關搭配提示點對 Adobe 視訊檔案進行編碼的詳細資訊，請參閱 「使用 Flash」中的 「內嵌提示點」。

您可以藉由編寫 ActionScript 的方式，存取提示點參數。提示點參數是回呼處理常式所接收的事件物件之一部分。

若要在 FLV 檔達到特定提示點時觸發程式碼中的某些動作，請使用 NetStream.onCuePoint 事件處理常式。

若要針對 F4V 視訊檔案中的提示點同步化動作，您必須從 onMetaData() 或 onXMPData() 回呼函數擷取該提示點資料，並使用 ActionScript 3.0 中的 Timer 類別觸發該提示點。如需有關 F4V 提示點的詳細資訊，請參閱第 424 頁 「使用 onXMPData()」。

如需有關處理提示點和中繼資料的詳細資訊，請參閱第 414 頁 「為中繼資料和提示點撰寫回呼方法」。

為中繼資料和提示點撰寫回呼方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以在到達特定提示點或是播放程式收到特定中繼資料時，在應用程式中觸發動作。當這些事件發生時，您必須使用特定的回呼方法做為事件處理常式。NetStream 類別會指定可能在播放期間發生的下列中繼資料事件：onCuePoint (僅限 FLV 檔)、onImageData、onMetaData、onPlayStatus、onTextData 和 onXMPData。

您必須為這些處理常式撰寫回呼方法，否則 Flash 執行階段可能會擲出錯誤。例如，下列程式碼會播放與 SWF 檔位於相同資料夾中的 video.flv 檔：

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    trace(event.text);
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

上述程式碼會載入名為 video.flv 的本機視訊檔案，並偵聽要傳送的 asyncError (AsyncErrorEvent.ASYNC_ERROR)。當原生非同步程式碼擲出例外時，會傳送此事件。在這種情況下，事件會在視訊檔案包含中繼資料或是提示點資訊時傳送，而且尚未定義適當的偵聽程式。如果您對於視訊檔案的中繼資料或是提示點資訊沒興趣，則上述程式碼會處理 asyncError 事件並忽略錯誤。如果您的 FLV 含有中繼資料與數個提示點，trace() 函數會顯示下列錯誤訊息：

```
Error #2095: flash.net.NetStream was unable to invoke callback onMetaData.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
```

因為 NetStream 物件無法找到 onMetaData 或 onCuePoint 回呼方法，所以會發生這個錯誤。在應用程式中定義這些回呼方法有多種方式。

更多說明主題

Flash Media Server：處理串流中的中繼資料

將 NetStream 物件的 client 屬性設定為 Object

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

透過將 client 屬性設定為 Object 或 NetStream 的子類別，您可以重新遞送 onMetaData 與 onCuePoint 回呼方法，或是完全忽略它們。下列範例將示範您如何使用空的 Object 以忽略回呼方法，而不需偵聽 asyncError 事件：

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var customClient:Object = new Object();

var ns:NetStream = new NetStream(nc);
ns.client = customClient;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

如果您想要偵聽 onMetaData 或 onCuePoint 回呼方法，就必須定義方法以處理這些回呼方法，如下列程式碼片段所示：

```
var customClient:Object = new Object();
customClient.onMetaData = metaDataHandler;
function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
```

上述程式碼會偵聽 onMetaData 回呼方法，並呼叫 metaDataHandler() 方法，以追蹤字串。如果 Flash 執行階段遇到提示點，即使沒有定義任何 onCuePoint 回呼方法，仍然不會產生錯誤。

建立自訂類別並定義方法以處理回呼方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列程式碼會將 NetStream 物件的 client 屬性設定為自訂類別 CustomClient，它可以為回呼方法定義處理常式：

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = new CustomClient();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

CustomClient 類別如下所示：

```
package
{
    public class CustomClient
    {
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
    }
}
```

CustomClient 類別會為 onMetaData 回呼處理常式定義處理常式。如果遇到提示點，並且呼叫 onCuePoint 回呼處理常式，將會傳送 asyncError 事件 (AsyncErrorEvent.ASYNC_ERROR)，說明錯誤為「flash.net.NetStream 無法叫用回呼 onCuePoint」。若要避免此錯誤，您就必須在 CustomClient 類別中定義 onCuePoint 回呼方法，或是為 asyncError 事件定義事件處理常式。

擴充 NetStream 類別並增加方法以處理回呼方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列程式碼會建立 CustomNetStream 類別的實體，在稍後列出的程式碼中將會定義這個類別：

```
var ns:CustomNetStream = new CustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

下面列出的程式碼會定義 CustomNetStream 類別，該類別會擴充 NetStream 類別、處理所需 NetConnection 物件的建立作業，並處理 onMetaData 與 onCuePoint 回呼處理常式方法：

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public class CustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public function CustomNetStream()
        {
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
        public function onCuePoint(infoObject:Object):void
        {
            trace("cue point");
        }
    }
}
```

如果您要在 CustomNetStream 類別中重新命名 onMetaData() 與 onCuePoint() 方法，可以使用下列程式碼：

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public class CustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public var onMetaData:Function;
        public var onCuePoint:Function;
        public function CustomNetStream()
        {
            onMetaData = metaDataHandler;
            onCuePoint = cuePointHandler;
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
        private function metaDataHandler(infoObject:Object):void
        {
            trace("metadata");
        }
        private function cuePointHandler(infoObject:Object):void
        {
            trace("cue point");
        }
    }
}
```

擴充 NetStream 類別並使其成為動態類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以擴充 NetStream 類別並使子類別成為動態類別，這樣就能以動態方式增加 onCuePoint 與 onMetaData 回呼處理常式。這會在下列程式碼中說明：

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

DynamicCustomNetStream 類別如下所示：

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public dynamic class DynamicCustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public function DynamicCustomNetStream()
        {
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
    }
}
```

因為 DynamicCustomNetStream 是動態類別，所以即使 onMetaData 與 onCuePoint 回呼處理常式沒有處理常式，也不會擲出錯誤。如果您要為 onMetaData 與 onCuePoint 回呼處理常式定義方法，可以使用下列程式碼：

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.onMetaData = metaDataHandler;
ns.onCuePoint = cuePointHandler;
ns.play("http://www.helpexamples.com/flash/video/cuepoints.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
function cuePointHandler(infoObject:Object):void
{
    trace("cue point");
}
```

將 NetStream 物件的 client 屬性設定成 this

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

透過將 client 屬性設定成 this，應用程式會在目前範圍中搜尋 onMetaData() 與 onCuePoint() 方法。這會在下列範例中說明：

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

如果呼叫 onMetaData 或 onCuePoint 回呼處理常式，但是沒有方法可處理回呼，則不會產生任何錯誤。若要處理這些回呼處理常式，請在程式碼中建立 onMetaData() 與 onCuePoint() 方法，如下列程式碼片段所示：

```
function onMetaData(infoObject:Object):void
{
    trace("metadata");
}
function onCuePoint(infoObject:Object):void
{
    trace("cue point");
}
```

使用提示點和中繼資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用 NetStream 回呼方法，即可在視訊播放時，捕捉及處理提示點和中繼資料事件。

使用提示點

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下表說明您可以用來在 Flash Player 和 AIR 中捕捉 F4V 和 FLV 提示點的回呼方法。

執行階段	F4V	FLV
Flash Player 9/ AIR1.0		OnCuePoint
		OnMetaData
Flash Player 10		OnCuePoint
	OnMetaData	OnMetaData
	OnXMPData	OnXMPData

下列範例會使用簡單的 for..in 迴圈，重複執行 onCuePoint() 函數所收到之 infoObject 參數中的每一個屬性。它會在收到提示點資料時，呼叫 trace() 函數並顯示訊息。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onCuePoint(infoObject:Object):void
{
    var key:String;
    for (key in infoObject)
    {
        trace(key + ": " + infoObject[key]);
    }
}
```

執行範例後，會顯示下列輸出：

```
parameters:
name: point1
time: 0.418
type: navigation
```

這個程式碼會使用前述技術中的其中一項，設定回呼方法所執行的目標物件。您可以使用其它技術。如需詳細資訊，請參閱第 414 頁「為中繼資料和提示點撰寫回呼方法」。

使用視訊中繼資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 OnMetaData() 和 OnXMPData() 函數，存取視訊檔案內的中繼資料資訊 (包括提示點)。

使用 OnMetaData()

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

中繼資料包含視訊檔案的相關資訊，例如時間長度、寬度、高度和影格速率。視您用來編碼視訊檔案的軟體而定，加入至視訊檔案的中繼資料資訊將有所不同。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onMetaData(infoObject:Object):void
{
    var key:String;
    for (key in infoObject)
    {
        trace(key + ": " + infoObject[key]);
    }
}
```

上述程式碼會產生下列輸出：

```
width: 320
audiodelay: 0.038
canSeekToEnd: true
height: 213
cuePoints: ,,
audiodatarate: 96
duration: 16.334
videodatarate: 400
framerate: 15
videocodecid: 4
audiocodecid: 2
```

如果您的視訊檔案不含音效，則與音效相關的中繼資料資訊 (例如 audiodatarate) 便會傳回 undefined，因為在編碼過程中，並未將音效資訊加入中繼資料內。

在上述程式碼中，提示點資訊並未顯示。為了顯示提示點中繼資料，您可以使用下列函數，它會以遞迴方式顯示 Object 中的項目：

```
function traceObject(obj:Object, indent:uint = 0):void
{
    var indentString:String = "";
    var i:uint;
    var prop:String;
    var val:*;
    for (i = 0; i < indent; i++)
    {
        indentString += "\t";
    }
    for (prop in obj)
    {
        val = obj[prop];
        if (typeof(val) == "object")
        {
            trace(indentString + " " + prop + ": [Object]");
            traceObject(val, indent + 1);
        }
        else
        {
            trace(indentString + " " + prop + ": " + val);
        }
    }
}
```

使用上述程式碼片段以追蹤 onMetaData() 方法中的 infoObject 參數，將會建立下列輸出：

```
width: 320
audiodatarate: 96
audiocodecid: 2
videocodecid: 4
videodatarate: 400
canSeekToEnd: true
duration: 16.334
audiodelay: 0.038
height: 213
framerate: 15
cuePoints: [Object]
    0: [Object]
        parameters: [Object]
            lights: beginning
        name: point1
        time: 0.418
        type: navigation
    1: [Object]
        parameters: [Object]
            lights: middle
        name: point2
        time: 7.748
        type: navigation
    2: [Object]
        parameters: [Object]
            lights: end
        name: point3
        time: 16.02
        type: navigation
```

下列範例會顯示 MP4 視訊的中繼資料。它會假設有個名為 metaDataOut 的 TextArea 物件，以便在其中寫入中繼資料。

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.events.NetStatusEvent;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.MouseEvent;

    public class onMetaDataExample extends Sprite
    {
        var video:Video = new Video();

        public function onMetaDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);
            video.x = 185;
            video.y = 5;

            video.attachNetStream(videoStream);

            videoStream.play("video.mp4");

            videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
        }

        public function onMetaData(infoObject:Object):void
        {
            for(var propName:String in infoObject)
            {
                metaDataOut.appendText(propName + "=" + infoObject[propName] + "\n");
            }
        }

        private function netStatusHandler(event:NetStatusEvent):void
        {
            if(event.info.code == "NetStream.Play.Stop")
                stage.displayState = StageDisplayState.NORMAL;
        }
    }
}
```

onMetaData() 函數為此視訊所產生的輸出如下：

```
moovposition=731965
height=352
avclevel=21
videocodecid=avc1
duration=2.36
width=704
videoframerate=25
avcprofile=88
trackinfo=[object Object]
```

使用資訊物件

下表列出視訊中繼資料的可能值，這些值會傳遞至所收到之 Object 中的 onMetaData() 回呼函數：

參數	說明
aacaot	AAC 音效物件類型，支援 0、1 或 2。
avclevel	AVC IDC 階層編號，例如 10、11、20、21 等。
avcprofile	AVC 描述檔編號，例如 55、77、100 等。
audiocodecid	指定所使用之音效轉碼器 (編碼 / 解碼技術) 的字串，例如 「.Mp3」或 「mp4a」。
audiodatarate	指出編碼音效時使用的速率數字，即每秒的 KB 數。
audiodelay	數字，指出原始 FLV 檔的 FLV 檔 「time 0」存在的時間。視訊內容必須經過些許延遲，才能正確地同步化音訊。
canSeekToEnd	Boolean 值，當 FLV 檔是以最後一個影格上的關鍵影格編碼，以允許搜尋漸進式下載視訊檔案的結尾時，則為 true。如果 FLV 檔不是以最後一個影格上的關鍵影格編碼，則為 false。
cuePoints	物件的陣列，內嵌於 FLV 檔的其中一個提示點。如果 FLV 檔沒有任何提示點，值即為 undefined。每個物件都有下列屬性： • type：字串，用以指定做為 「navigation」或 「event」的提示點類型。 • name：代表提示點名稱的字串。 • time：數字，具有三位小數位數 (毫秒) 的提示點時間，以秒為單位。 • parameters：具有使用者在建立提示點時所指定之名稱值配對的選擇性物件。
duration	數字，指定視訊檔案的持續時間，以秒為單位。
framerate	FLV 檔的影格速率。
height	數字，表示 FLV 檔的高度，以像素為單位。
seekpoints	陣列，將可用的關鍵影格列出做為以秒為單位的時間戳記。選擇性。
tags	索引鍵值配對的陣列，表示 「ilst」 Atom (MP4 檔之 ID3 標籤的對等項目) 中的資訊。iTunes 會使用這些標籤。如果可以的話，也能用於顯示圖案。
trackinfo	提供 MP4 檔中所有音軌之資訊 (包括其取樣描述 ID) 的物件。
videocodecid	字串，代表用來編碼視訊的編碼器版本。例如，「avc1」或 「VP6F」。
videodatarate	FLV 檔的視訊資料速率。
videoframerate	MP4 視訊的影格速率。
width	數字，表示 FLV 檔的寬度，以像素為單位。

下表列出 videocodecid 參數的可能值：

videocodecid	轉碼器名稱
2	Sorenson H.263
3	螢幕視訊 (僅 SWF 第 7 版和更新版本)
4	VP6 (僅 SWF 第 8 版和更新版本)
5	具有 Alpha 色版的 VP6 視訊 (僅 SWF 第 8 版和更新版本)

下表列出 audiocodecid 參數的可能值：

audiocodecid	轉碼器名稱
0	未壓縮
1	ADPCM
2	Mp3
4	Nellymoser 16 kHz 單聲道
5	Nellymoser 8kHz 單聲道
6	Nellymoser
10	AAC
11	Speex

使用 onXMPData()

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

onXMPData() 回呼函數會接收內嵌於 Adobe F4V 或 FLV 視訊檔案中的 Adobe 可延伸中繼資料平台 (XMP) 專屬資訊。XMP 中繼資料包含提示點以及其它視訊中繼資料。XMP 中繼資料支援是 Flash Player 10 和 Adobe AIR 1.5 新增的功能，後續版本都將持續提供這項支援。

下列範例會處理 XMP 中繼資料內的提示點資料：

```
package
{
    import flash.display.*;
    import flash.net.*;
    import flash.events.NetStatusEvent;
    import flash.media.Video;

    public class onXMPDataExample extends Sprite
    {
        public function onXMPDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;
            var video:Video = new Video();

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("video.f4v");
        }

        public function onMetaData(info:Object):void {
            trace("onMetaData fired");
        }

        public function onXMPData(infoObject:Object):void
        {
            trace("onXMPData Fired\n");
            //trace("raw XMP =\n");
            //trace(infoObject.data);
```

```
            var cuePoints:Array = new Array();
            var cuePoint:Object;
            var strFrameRate:String;
            var nTracksFrameRate:Number;
            var strTracks:String = "";
            var onXMPXML = new XML(infoObject.data);
            // Set up namespaces to make referencing easier
            var xmpDM:Namespace = new Namespace("http://ns.adobe.com/xmp/1.0/DynamicMedia/");
            var rdf:Namespace = new Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#");
            for each (var it:XML in onXMPXML..xmpDM::Tracks)
            {
                 var strTrackName:String = it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::trackName;
                 var strFrameRateXML:String = it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::frameRate;
                 strFrameRate = strFrameRateXML.substr(1,strFrameRateXML.length);

                 nTracksFrameRate = Number(strFrameRate);

                 strTracks += it;
            }
            var onXMPTracksXML:XML = new XML(strTracks);
            var strCuepoints:String = "";
            for each (var item:XML in onXMPTracksXML..xmpDM::markers)
            {
                strCuepoints += item;
            }
            trace(strCuepoints);
        }
    }
}
```

對於名為 startrekintro.f4v 的簡短視訊檔案，上述範例會產生下列追蹤程式碼行。這些程式碼行會顯示用於瀏覽的提示點資料以及 XMP 中繼資料內的事件提示點：

```
onMetaData fired
onXMPData Fired

<xmpDM:markers xmlns:xmp="http://ns.adobe.com/xap/1.0/"
xmlns:xmpDM="http://ns.adobe.com/xmp/1.0/DynamicMedia/"
xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#" xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"
xmlns:stEvt="http://ns.adobe.com/xap/1.0/sType/ResourceEvent#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:x="adobe:ns:meta/">
  <rdf:Seq>
    <rdf:li>
      <rdf:Description xmpDM:startTime="7695905817600" xmpDM:name="Title1" xmpDM:type="FLVCuePoint"
xmpDM:cuePointType="Navigation">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="Title" xmpDM:value="Star Trek"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
    <rdf:li>
      <rdf:Description xmpDM:startTime="10289459980800" xmpDM:name="Title2" xmpDM:type="FLVCuePoint"
xmpDM:cuePointType="Event">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="William Shatner" xmpDM:value="First Star"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Light Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
  </rdf:Seq>
</xmpDM:markers>
onMetaData fired
```

備註： 在 XMP 資料中，時間的儲存單位為 「DVA 刻度」，而非秒。若要計算提示點時間，請將開始時間除以影格速率。例如，開始時間 7695905817600 除以影格速率 254016000000 等於 30:30。

若要查看完整的原始 XMP 中繼資料 (包括影格速率)，請移除 onXMPData() 函數開頭的第二和第三個 trace() 陳述式之前的註解識別項 (//'s)。

如需有關 XMP 的詳細資訊，請前往：

- partners.adobe.com/public/developer/xmp/topic.html
- www.adobe.com/devnet/xmp/

使用影像中繼資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

onImageData 事件會透過 AMF0 資料通道，傳送影像資料做為位元組陣列。資料的格式可以是 JPEG、PNG 或 GIF。以您為 onCuePoint 和 onMetaData 定義回呼方法的相同方式來定義 onImageData() 回呼方法，即可處理此資訊。下列範例會使用 onImageData() 回呼方法，存取及顯示影像資料：

```
public function onImageData(imageData:Object):void
{
    // display track number
    trace(imageData.trackid);
    var loader:Loader = new Loader();
    //imageData.data is a ByteArray object
    loader.loadBytes(imageData.data);
    addChild(loader);
}
```

使用文字中繼資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

onTextData 事件會透過 AMF0 資料通道傳送文字資料。此文字資料採用 UTF-8 格式，而且包含根據 3GP 定時文字規格進行格式化的其它相關資訊。此規格會定義標準化的副標題格式。以您為 onCuePoint 或 onMetaData 定義回呼方法的相同方式，定義 onTextData() 回呼方法即可處理此資訊。在下列範例中，onTextData() 方法會顯示曲目 ID 編號以及對應的曲目文字。

```
public function onTextData(textData:Object):void
{
    // display the track number
    trace(textData.trackid);
    // displays the text, which can be a null string, indicating old text
    // that should be erased
    trace(textData.text);
}
```

監視 NetStream 活動

Flash Player 10.3 以及更新的版本，**Adobe AIR 2.7** 以及更新的版本

您可以監視 NetStream 活動，收集支援媒體使用分析和報告的必要資訊。本章節討論的監視功能可讓您建立媒體度量庫，以便在沒有特定視訊播放程式 (顯示媒體) 的關閉聯合的情況下收集資料。這能讓您的用戶端開發人員在使用元件庫時，選擇他們喜愛的視訊播放程式。使用 NetMonitor 類別可在應用程式中監視 NetStream 物件的建立與活動。NetMonitor 類別提供任何指定時間所存在的作用中 NetStreams 清單，並在每次建立 NetStream 物件時同時傳送事件。

NetStream 物件會依正在播放的媒體類型，傳送下表所列的事件：

事件	漸進式下載	RTMP 串流	HTTP 串流
NetStream.Play.Start	是	是	否
NetStream.Play.Stop	是	是	否
NetStream.Play.Complete	是	是	否
NetStream.SeekStart.Notify	是	是	是
NetStream.Seek.Notify	是	是	是
NetStream.Unpause.Notify	是	是	是
NetStream.Unpause.Notify	是	是	是
NetStream.Play.Transition	不適用	是	不適用
NetStream.Play.TransitionComplete	不適用	是	不適用

事件	漸進式下載	RTMP 串流	HTTP 串流
NetStream.Buffer.Full	是	是	是
NetStream.Buffer.Flush	是	是	是
NetStream.Buffer.Empty	是	是	是

NetStream 範例相關的 NetStreamInfo 物件也會儲存在媒體中發現的最後一個中繼資料和 XMP 資料物件。

透過 HTTP 串流播放媒體時，不會傳送 NetStream.Play.Start、NetStream.Play.Stop 和 NetStream.Play.Complete，因為應用程式已完全控制媒體串流。視訊播放程式應針對 HTTP 串流合成與傳送這些事件。

同樣地，程式也不會針對漸進方式下載或 HTTP 媒體傳送 NetStream.Play.Transition 和 NetStream.Play.TransitionComplete。動態位元傳輸速率切換是 RTMP 功能。如果使用 HTTP 串流的視訊播放程式支援類似的功能，播放程式便可以合成並傳送轉換事件。

更多說明主題

Adobe 開發人員連線：在 Flash 中度量視訊耗用

監視 NetStream 事件

兩種事件提供有用的使用資料：netStatus 和 mediaTypeData。此外，計時器還可用來定期記錄 NetStream 播放磁頭的位置。

netStatus 事件提供資訊，可讓您用來判斷使用者已檢視的串流量。緩衝區和 RTMFP 串流轉換事件也會產生 netStatus 事件。

mediaTypeData 事件提供中繼和 XMP 資料資訊。Netstream.Play.Complete 事件會當做 mediaTypeData 事件來傳送。您也可以透過 mediaTypeData 事件使用串流所內嵌的其他資料，包括提示點、文字和影像。

下列範例說明如何在應用程式中，建立類別來監視任何作用中 NetStreams 的狀態和資料事件。一般而言，這種類別會將有意用來分析的資料上傳到伺服器，以進行收集。

```
package com.adobe.example
{
    import flash.events.NetDataEvent;
    import flash.events.NetMonitorEvent;
    import flash.events.NetStatusEvent;
    import flash.net.NetMonitor;
    import flash.net.NetStream;

    public class NetStreamEventMonitor
    {
        private var netmon:NetMonitor;
        private var heartbeat:Timer = new Timer( 5000 );

        public function NetStreamEventMonitor()
        {
            //Create NetMonitor object
            netmon = new NetMonitor();
            netmon.addEventListener( NetMonitorEvent.NET_STREAM_CREATE, newNetStream );

            //Start the heartbeat timer
            heartbeat.addEventListener( TimerEvent.TIMER, onHeartbeat );
            heartbeat.start();
        }

        //On new NetStream
        private function newNetStream( event:NetMonitorEvent ):void
        {
            trace( "New Netstream object");
```

```
            var stream:NetStream = event.netStream;
            stream.addEventListener(NetDataEvent.MEDIA_TYPE_DATA, onStreamData);
            stream.addEventListener(NetStatusEvent.NET_STATUS, onStatus);
        }

        //On data events from a NetStream object
        private function onStreamData( event:NetDataEvent ):void
        {

            var netStream:NetStream = event.target as NetStream;
            trace( "Data event from " + netStream.info.uri + " at " + event.timestamp );
            switch( event.info.handler )
            {
                case "onMetaData":
                    //handle metadata;
                    break;
                case "onXMPData":
                    //handle XMP;
                    break;
                case "onPlayStatus":
                    //handle NetStream.Play.Complete
                case "onImageData":
                    //handle image
                    break;
                case "onTextData":
                    //handle text
                    break;
                default:
                    //handle other events

            }
        }

        //On status events from a NetStream object
        private function onStatus( event:NetStatusEvent ):void
        {
            trace( "Status event from " + event.target.info.uri + " at " + event.target.time );
            //handle status events
        }
        //On heartbeat timer
        private function onHeartbeat( event:TimerEvent ):void
        {
            var streams:Vector.<NetStream> = netmon.listStreams();
            for( var i:int = 0; i < streams.length; i++ )
            {
                trace( "Heartbeat on " + streams[i].info.uri + " at " + streams[i].time );
                //handle heartbeat event
            }
        }

    }
}
```

偵測播放程式網域

使用者正在檢視媒體的網頁之 URL 和網域，不一定永遠可輕易存取。如果主控網站允許，您可以使用 ExternalInterface 類別來取得確切的 URL。不過，有些允許協力廠商視訊播放程式的網站並不允許使用 ExternalInterface。在這種情況下，您可以從 Security 類別的 pageDomain 屬性中取得目前網頁的網域。基於使用者的安全性和隱私權考量，不會洩露完整的 URL。

頁面網域可以從 Security 類別的靜態 pageDomain 屬性取得：

```
var domain:String = Security.pageDomain;
```

視訊檔案的進階主題

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列主題將討論一些使用 FLV 檔的特殊問題。

關於設定能在伺服器上使用的 FLV 檔

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在使用 FLV 檔時，您也必須設定伺服器，才能在伺服器上使用 FLV 檔格式。「多用途網際網路信件擴充標準」(Multipurpose Internet Mail Extensions，MIME) 為一項標準化的資料規格，讓您能夠透過網際網路連線傳送非 ASCII 格式的檔案。由於網頁瀏覽器和電子郵件用戶端會設定用來解譯為數眾多的 「MIME 類型」，所以能夠收發視訊、音訊、圖形和格式化文字。若要從網站伺服器載入 FLV 檔，您可能需要向網站伺服器註冊副檔名與 MIME 類型；請查閱網站伺服器說明文件。FLV 檔的 MIME 類型是 video/x-flv。FLV 檔類型的所有資訊如下：

- Mime 類型：video/x-flv
- 副檔名：.flv
- 必要參數：無
- 選擇性參數：無
- 編碼注意事項：FLV 檔是二進位檔案，而某些應用程式可能還會要求設定 application/octet-stream 子類型
- 安全性問題：無
- 已發佈的規格：www.adobe.com/go/video_file_format_tw

Microsoft 在 Microsoft Internet Information Services (IIS) 6.0 網站伺服器中，改變了舊版處理串流資訊的方式。舊版的 IIS 並不需要針對串流 Flash 視訊進行任何修改，在 IIS 6.0 (Windows 2003 的預設瀏覽器) 中，伺服器便需要 MIME 類型，才能辨識 FLV 檔是否為串流媒體。

將串流外部 FLV 檔的 SWF 檔放置在 Microsoft Windows Server® 2003 上，並以瀏覽器來檢視時，SWF 檔會正確地播放，但 FLV 視訊卻不會進行串流。這個問題會影響所有放在 Windows Server 2003 上的 FLV 檔，包括以舊版 Flash 編寫工具和 Adobe 的 Macromedia Flash Video Kit for Dreamweaver MX 2004 製作的檔案。但是，如果在其它作業系統中進行測試，這些檔案則可以正確地播放。

如需有關設定 Microsoft Windows 2003 和 Microsoft IIS Server 6.0，以便串流 FLV 視訊的詳細資訊，請參閱 www.adobe.com/go/tn_19439_tw。

關於在 Macintosh 上指向本機 FLV 檔的位置

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如果您嘗試使用以斜線 (/) 表示的相對路徑，在 Apple® Macintosh® 電腦的非系統磁碟機上播放本機 FLV 檔，則該視訊將不會播放。「非系統磁碟機」包括 (但不限於) CD-ROM、已分割的硬碟、抽取式儲存媒體，以及連接的儲存裝置。

備註：只有作業系統的限制才會導致這項失敗，Flash Player 或 AIR 中的限制則不會。

若要在 Macintosh 電腦的非系統磁碟機上播放 FLV 檔，請以使用冒號 (:) 標記法的絕對路徑指向這個檔案，而不是斜線 (/) 標記法。以下為兩種標記法不同之處：

- 斜線標記法：myDrive/myFolder/myFLV.flv
- 冒號標記法：(Mac OS®) myDrive:myFolder:myFLV.flv

視訊範例：視訊點唱機

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列範例會建立簡單的視訊點唱機，它能動態地載入要依序播放的視訊清單。這讓您建立的應用程式可讓使用者瀏覽一系列的視訊教學課程，或者指定在傳遞使用者要求的視訊之前，應該播放哪些廣告。這個範例將示範下列 ActionScript 3.0 的功能：

- 根據視訊檔案的播放進度來更新播放磁頭
- 偵聽和剖析視訊檔案的中繼資料
- 以網路串流處理特定的程式碼
- 載入、播放、暫停和停止動態載入的 FLV
- 根據網路串流的中繼資料，調整顯示清單上視訊物件的大小

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。您可以在 Samples/VideoJukebox 資料夾中找到「視訊點唱機」應用程式檔案，此應用程式是由下列檔案組成：

檔案	說明
VideoJukebox.fla 或 VideoJukebox.mxml	主應用程式檔案，在 Flex 中為 MXML，在 Flash 中為 FLA。
VideoJukebox.as	提供應用程式主要功能的類別。
playlist.xml	列出哪些視訊檔案將載入視訊點唱機的檔案。

載入外部視訊播放清單檔案

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

外部 playlist.xml 檔案會指定要載入哪些視訊，以及要播放它們的順序。為了載入 XML 檔案，您必須使用 URLLoader 物件以及 URLRequest 物件，如下列程式碼所示：

```
uldr = new URLLoader();
uldr.addEventListener(Event.COMPLETE, xmlCompleteHandler);
uldr.load(new URLRequest(PLAYLIST_XML_URL));
```

此程式碼是置於 VideoJukebox 類別的建構函式中，因此在執行其它程式碼之前會先載入檔案。一旦 XML 檔案載入完成，就會呼叫 xmlCompleteHandler() 方法，以便將外部檔案剖析為 XML 物件，如下列程式碼所示：

```
private function xmlCompleteHandler(event:Event):void
{
    playlist = XML(event.target.data);
    videosXML = playlist.video;
    main();
}
```

playlist XML 物件包含來自外部檔案的原始 XML，而視訊 XML 則是一種 XMLList 物件，只包含視訊節點。在下列程式碼片段中可以看到樣本 playlist.xml 檔案：

```
<videos>
    <video url="video/caption_video.flv" />
    <video url="video/cuepoints.flv" />
    <video url="video/water.flv" />
</videos>
```

最後，xmlCompleteHandler() 方法會呼叫 main() 方法，它會在顯示清單上設定各種組件實體，並設定用來載入外部 FLV 檔的 NetConnection 與 NetStream 物件。

建立使用者介面

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要建立使用者介面，您必須拖曳五個 Button 實體到顯示清單上，並為它們指定下列實體名稱：playButton、pauseButton、stopButton、backButton 和 forwardButton。

對於每一個 Button 實體，您都必須指定其 click 事件的處理常式，如下列程式碼片段所示：

```
playButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
pauseButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
stopButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
backButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
forwardButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
```

buttonClickHandler() 方法會使用 switch 陳述式判斷按下的是哪個按鈕實體，如下列程式碼所示：

```
private function buttonClickHandler(event:MouseEvent):void
{
    switch (event.currentTarget)
    {
        case playButton:
            ns.resume();
            break;
        case pauseButton:
            ns.togglePause();
            break;
        case stopButton:
            ns.pause();
            ns.seek(0);
            break;
        case backButton:
            playPreviousVideo();
            break;
        case forwardButton:
            playNextVideo();
            break;
    }
}
```

接下來，將 Slider 實體加入至顯示清單，並為它指定實體名稱為 volumeSlider。下列程式碼會將滑動軸實體的 liveDragging 屬性設定為 true，並為滑動軸實體的 change 事件定義事件偵聽程式：

```
volumeSlider.value = volumeTransform.volume;
volumeSlider.minimum = 0;
volumeSlider.maximum = 1;
volumeSlider.snapInterval = 0.1;
volumeSlider.tickInterval = volumeSlider.snapInterval;
volumeSlider.liveDragging = true;
volumeSlider.addEventListener(SliderEvent.CHANGE, volumeChangeHandler);
```

將 ProgressBar 實體加入至顯示清單，並為它指定實體名稱為 positionBar。將此實體的 mode 屬性設定為手動，如下列程式碼片段所示：

```
positionBar.mode = ProgressBarMode.MANUAL;
```

最後，將 Label 實體加入至顯示清單，並為它指定實體名稱為 positionLabel。計時器實體將會設定這個 Label 實體的值。

偵聽視訊物件的中繼資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當 Flash Player 遇到每個載入視訊的中繼資料時，就會針對 NetStream 物件的 client 屬性呼叫 onMetaData() 回呼處理常式。下列程式碼會初始化 Object，並設定指定的回呼處理常式：

```
client = new Object();
client.onMetaData = metadataHandler;
```

metadataHandler() 方法會將其資料複製到程式碼前面部分所定義的 meta 屬性。這可讓您在整個應用程式中隨時存取目前視訊的中繼資料。接下來，會調整 「舞台」上視訊物件的大小，以符合從中繼資料傳回的尺寸。最後，會移動 positionBar 進度列實體，並根據目前播放的視訊大小調整大小。下列程式碼包含整個 metadataHandler() 方法：

```
private function metadataHandler(metadataObj:Object):void
{
    meta = metadataObj;
    vid.width = meta.width;
    vid.height = meta.height;
    positionBar.move(vid.x, vid.y + vid.height);
    positionBar.width = vid.width;
}
```

動態載入 視訊

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要動態載入各個視訊，應用程式必須使用 NetConnection 與 NetStream 物件。下列程式碼會建立 NetConnection 物件，並將 null 傳遞給 connect() 方法。透過指定 null，Flash Player 會連接至本機伺服器上的視訊，而不會連接至伺服器，如 Flash Media Server。

下列程式碼會建立 NetConnection 與 NetStream 實體、為 netStatus 事件定義事件偵聽程式，並將 client Object 指定給 client 屬性：

```
nc = new NetConnection();
nc.connect(null);

ns = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
ns.client = client;
```

每次變更視訊狀態時，就會呼叫 netStatusHandler() 方法。這包括當視訊開始或停止播放、正在緩衝或是如果找不到視訊串流時。下列程式碼會列出 netStatusHandler() 事件：

```
private function netStatusHandler(event:NetStatusEvent):void
{
    try
    {
        switch (event.info.code)
        {
            case "NetStream.Play.Start":
                t.start();
                break;
            case "NetStream.Play.StreamNotFound":
            case "NetStream.Play.Stop":
                t.stop();
                playNextVideo();
                break;
        }
    }
    catch (error:TypeError)
    {
        // Ignore any errors.
    }
}
```

上述程式碼會評估 info 物件的程式碼屬性，並將程式碼篩選為 「NetStream.Play.Start」、「NetStream.Play.StreamNotFound」或 「NetStream.Play.Stop」。其它所有程式碼將會遭忽略。如果網路串流已開始，程式碼就會啟動 Timer 實體並更新播放磁頭。如果找不到或無法停止網路串流，就會停止 Timer 實體，而且應用程式會嘗試播放在播放清單中的下一個視訊。

每次 Timer 執行時，positionBar 進度列實體會呼叫 ProgressBar 類別的 setProgress() 方法以更新其目前的位置，而 positionLabel Label 實體則會以目前視訊的已經過時間及總時間來更新。

```
private function timerHandler(event:TimerEvent):void
{
    try
    {
        positionBar.setProgress(ns.time, meta.duration);
        positionLabel.text = ns.time.toFixed(1) + " of " meta.duration.toFixed(1) + " seconds";
    }
    catch (error:Error)
    {
        // Ignore this error.
    }
}
```

控制視訊的音量

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以設定 NetStream 物件的 soundTransform 屬性，針對動態載入的視訊控制音量。視訊點唱機應用程式可讓您變更 volumeSlider Slider 實體的值以修改音量。下列程式碼將說明如何變更音量，方法是將 Slider 組件的值指定給 SoundTransform 物件，而該物件會設定為 NetStream 物件的 soundTransform 屬性：

```
private function volumeChangeHandler(event:SliderEvent):void
{
    volumeTransform.volume = event.value;
    ns.soundTransform = volumeTransform;
}
```

控制視訊播放

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當視訊到達視訊串流的結尾或是使用者跳到上一個或下一個視訊時，應用程式的其餘部分就會控制視訊播放。

下列方法會從目前選取的索引之 XMLList 擷取視訊 URL：

```
private function getVideo():String
{
    return videosXML[idx].@url;
}
```

playVideo() 方法會呼叫 NetStream 物件的 play() 方法以載入目前選取的視訊：

```
private function playVideo():void
{
    var url:String = getVideo();
    ns.play(url);
}
```

playPreviousVideo() 方法會遞減目前的視訊索引、呼叫 playVideo() 方法以載入新視訊檔案，並將進度列設定成可以看見：

```
private function playPreviousVideo():void
{
    if (idx > 0)
    {
        idx--;
        playVideo();
        positionBar.visible = true;
    }
}
```

最後一個方法 playNextVideo() 會遞增視訊索引並呼叫 playVideo() 方法。如果目前的視訊是播放清單中的最後一個視訊，就會呼叫 Video 物件的 clear() 方法，而進度列實體的 visible 屬性會設定為 false：

```
private function playNextVideo():void
{
    if (idx < (videosXML.length() - 1))
    {
        idx++;
        playVideo();
        positionBar.visible = true;
    }
    else
    {
        idx++;
        vid.clear();
        positionBar.visible = false;
    }
}
```

使用硬體加速呈現方式的 **StageVideo** 類別

Flash Player 10.2 以及更新的版本

Flash Player 可透過使用 H.264 解碼的硬體加速使視訊效能最佳化。您可使用 StageVideo API 進一步提高效能。舞台視訊可讓您的應用程式充分利用硬體加速呈現方式。

支援 StageVideo API 的執行階段包括：

- Flash Player 10.2 以及更新的版本

備註：在 Flash Player 11.4/AIR 3.4 及更新版本中，您可以使用 StageVideo 進行攝影機輸入。

舞台視訊功能的可下載原始碼與其他詳細資訊可在開始使用舞台視訊中找到。

如需 StageVideo 快速入門課程，請參閱使用舞台視訊。

關於使用 StageVideo 的硬體加速

硬體加速呈現方式包括視訊縮放、色彩轉換和位圖傳送，可增強硬體加速解碼的效能。在提供 GPU (硬體) 加速的裝置上，您可以使用 flash.media.StageVideo 物件，直接在裝置硬體上處理視訊。直接處理可在 GPU 處理視訊時釋放 CPU 以執行其他工作。另一方面，舊版 Video 類別通常使用軟體呈現方式。軟體呈現方式在 CPU 上進行，可能耗用大量的系統資源。

目前僅有少數裝置提供完整的 GPU 加速功能。不過，舞台視訊可讓應用程式充分利用可用的硬體加速。

StageVideo 類別的存在，並不代表 Video 類別已經過時。這兩個類別結合起來在任何指定時間，提供裝置資源允許的最佳視訊顯示經驗。您的應用程式可以偵聽適當的事件，並視需要在 StageVideo 與 Video 之間切換，以充分利用硬體加速。

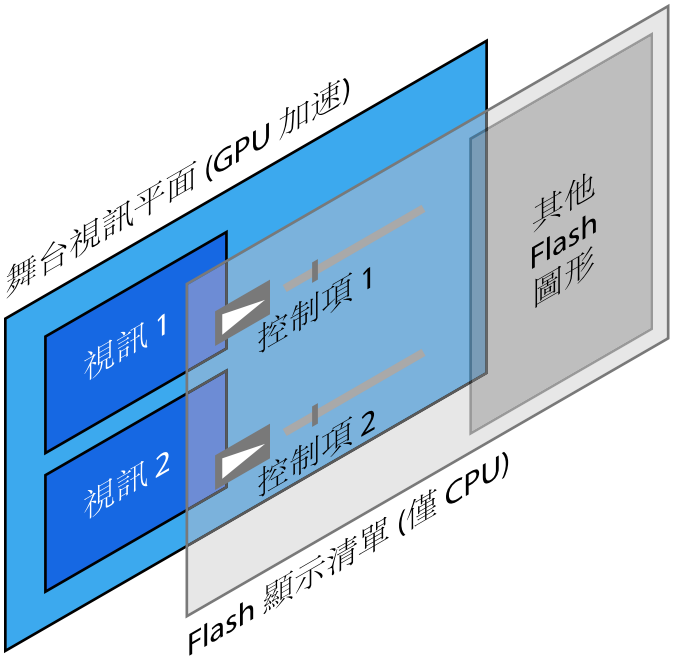
StageVideo 類別會在視訊用途上強制實行某些限制。在實作 StageVideo 之前，請檢視原則，並確定您的應用程式接受那些限制。若您接受那些限制，請在 Flash Player 偵測到可用的硬體加速呈現方式時使用 StageVideo 類別。請參閱第 437 頁「原則與限制」。

平行平面：舞台視訊與 Flash 顯示清單

使用舞台視訊模型時，Flash Player 可從顯示清單中分離視訊。Flash Player 會在兩個 Z 順序堆疊的平面之間分割複合顯示：

舞台視訊平面 舞台視訊平面位於背景。它只會顯示硬體加速視訊。因為這個設計，如果在裝置上不支援硬體加速或是未提供，則無法使用此平面。在 ActionScript 中，StageVideo 物件會處理在舞台視訊平面上播放的視訊。

Flash 顯示清單平面 Flash 顯示清單實體會在舞台視訊平面前面的平台上複合。顯示清單實體包括執行階段顯示的任何內容，包括播放控制項。未提供硬體加速時，只能使用 Video 類別物件在此平面上播放視訊。舞台視訊永遠會顯示在 Flash 顯示清單圖形的後面。



視訊顯示平面

StageVideo 物件會以螢幕中一個非旋轉且與視窗對齊的矩形區域顯示。您不能將物件放置在舞台視訊平面之後。不過，您可以使用 Flash 顯示清單平面，將其他圖形放置在舞台視訊平面之上。舞台視訊會與 顯示清單同時執行。因此，您可以同時使用兩種機制，在兩個不同平面建立統一的視覺效果。例如，您可以使用播放控制項的前面平面，以便操作背景執行的舞台視訊。

舞台視訊與 H.264 轉碼器

在 Flash Player 應用程式中，實作視訊硬體加速包含兩個步驟：

1 將視訊編碼為 H.264

2 實作 StageVideo API

為獲得最佳結果，請執行這兩個步驟。H.264 轉碼器可讓您發揮硬體加速的最大功能，包括視訊解碼與呈現方式。

舞台視訊可消除 GPU 對 CPU 回讀。換言之，GPU 不會再將解碼的影格傳送回 CPU 來複合顯示清單物件。而是直接將解碼與顯示的影格傳送到螢幕的顯示清單物件後面。此技術可減少 CPU 與記憶體使用量，並可提供較佳的像素真實度。

更多說明主題

第 404 頁 「瞭解視訊格式」

原則與限制

當視訊以全螢幕模式執行時，若裝置支援硬體加速，則永遠可使用舞台視訊。但 Flash Player 也會在瀏覽器中執行。在瀏覽器內容中，wmode 設定會影響舞台視訊的可用性。如果您想要使用舞台視訊，請一律嘗試使用 wmode="direct"。舞台視訊不是全螢幕模式時，則不相容於其他 wmode 設定。這個限制表示在執行階段舞台視訊可能會在可用與無法使用的狀態之間游移不定。例如，如果使用者在舞台視訊執行時結束全螢幕模式，則視訊內容會回到瀏覽器。如果將瀏覽器 wmode 參數設為 "direct"，舞台視訊可能會突然變成無法使用。Flash Player 可透過一組事件向應用程式溝通播放內容變更。若您實作 StageVideo API，請保留 Video 物件做為備份，以便在無法使用舞台視訊時使用。

因為舞台視訊與硬體的直接關係，它會限制某些視訊功能。舞台視訊會強制下列限制：

- 對於每個 SWF 檔案，Flash Player 會將可以同時顯示視訊的 StageVideo 物件數目限制為四個。不過，須視裝置硬體資源而定，實際限制有可能更少。
- 視訊時間未與執行階段顯示的 內容時間同步。
- 視訊顯示區域僅能為矩形。您無法使用較為進階的顯示區域，像是橢圓形或不規則形狀。
- 您無法旋轉視訊。
- 您無法以點陣圖快取視訊，也無法使用 BitmapData 物件存取它。
- 您無法對視訊套用濾鏡。
- 您無法對視訊套用顏色變化。
- 您無法對視訊套用 Alpha 值。
- 您套用至顯示清單平面物件的混合模式，不會套用至舞台視訊。
- 您只能將視訊置於完整像素邊界。
- 雖然 GPU 顯示是供指定裝置硬體使用的最佳方案，但是它在不同裝置並不是 100% 的「像素相同」。由於驅動程式與平台的不同，會有些微的差異。
- 有一些裝置不支援所有需要的顏色空間。例如，有些裝置不支援 BT.709 (H.264 標準)。在這種情況下，您可以使用 BT.601 進行快速顯示。
- 您無法以 WMODE 設定 (例如，一般、不透明或是透明) 使用舞台視訊。舞台視訊不是全螢幕模式時，僅支援 WMODE=direct。WMODE 對 Safari 4 或更新版本、IE 9 或更新版本沒有作用。

在大部分情況下，這些限制不會影響視訊播放程式應用程式。如果您可以接受這些限制，請盡可能使用舞台視訊。

更多說明主題

第 141 頁「使用全螢幕模式」

使用 StageVideo API

舞台視訊是執行階段中的一種機制，可增強視訊播放與裝置效能。執行階段會建立和維護此機制：身為開發人員，您的角色就是要將應用程式設定成利用它。

若要使用舞台視訊，您可以實作事件處理常式的架構，以偵測舞台視訊可使用和無法使用的時機。當您收到舞台視訊無法使用的通知時，您會收到來自 Stage.stageVideos 屬性的 StageVideo 物件。執行階段會以一或多個 StageVideo 物件填入此 Vector 物件。然後，您可以使用其中一個提供的 StageVideo 物件 (不是 Video 物件) 顯示串流視訊。

在 Flash Player 中，當您收到舞台視訊無法再使用的通知時，請將視訊串流切換回 Video 物件。

備註：您無法建立 StageVideo 物件。

Stage.stageVideos 屬性

Stage.stageVideos 屬性是可以讓您存取 StageVideo 實體的 Vector 物件。此向量最多可包含四個 StageVideo 物件，須視硬體與系統資源而定。行動裝置可能限制為一個或無。

當舞台視訊無法使用時，此向量不會包含任何物件。若要避免執行階段錯誤，請確定只在最新的 StageVideoAvailability 事件指出舞台視訊為可用時，才存取此向量的成員。

StageVideo 事件

StageVideo API 架構提供下列事件：

StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 當 Stage.stageVideos 屬性變更時傳送。StageVideoAvailabilityEvent.availability 屬性會指出 AVAILABLE 或 UNAVAILABLE。使用此事件以判斷 stageVideos 屬性是否包含任何 StageVideo 物件，而不是直接檢查 Stage.stageVideos 向量的長度。

StageVideoEvent.RENDER_STATE 已在將 NetStream 或 Camera 物件附加至 StageVideo 物件並播放時傳送。指出目前使用的解碼類型：硬體、軟體或無法使用 (未顯示任何項目)。事件目標包含 videoWidth 與 videoHeight 屬性，可用於調整視訊檢視區域大小。

重要事項：從 StageVideo 目標物件取得的座標會使用 「舞台」座標，因為它們不是標準顯示清單的一部分。

VideoEvent.RENDER_STATE 使用 Video 物件時傳送。指出軟體或硬體加速的解碼是否正在使用。如果此事件表示硬體加速的解碼，若有可能，請切換到 StageVideo 物件。Video 事件目標包含 videoWidth 與 videoHeight 屬性，可用於調整視訊檢視區域大小。

實作 StageVideo 功能的工作流程

請依循這些進階步驟來實作 StageVideo 功能：

1. 偵聽 StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 事件，瞭解變更 Stage.stageVideos 向量的時間。請參閱第 440 頁 「使用 StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 事件」。
2. 如果 StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 事件報告可使用舞台視訊，請使用事件處理常式中的 Stage.stageVideos Vector 物件來存取 StageVideo 物件。
3. 使用 StageVideo.attachNetStream() 來附加 NetStream 物件，或者使用 StageVideo.attachCamera() 來附加 Camera 物件。
4. 使用 NetStream.play() 來播放視訊。
5. 偵聽 StageVideo 物件上的 StageVideoEvent.RENDER_STATE 事件，以判斷播放視訊的狀態。收到這個事件也表示此視訊的 ⊠ 度與高度屬性已經初始化或變更。請參閱第 441 頁 「使用 StageVideoEvent.RENDER_STATE 與 VideoEvent.RENDER_STATE 事件」。
6. 偵聽 Video 物件上的 VideoEvent.RENDER_STATE 事件。此事件與 StageVideoEvent.RENDER_STATE 提供的狀態相同，因此您可以用它來判斷 GPU 加速是否可用。收到這個事件也表示此視訊的 ⊠ 度與高度屬性已經初始化或變更。請參閱第 441 頁 「使用 StageVideoEvent.RENDER_STATE 與 VideoEvent.RENDER_STATE 事件」。

初始化 StageVideo 事件偵聽程式

在應用程式初始化期間，設定您的 StageVideoAvailabilityEvent 與 VideoEvent 偵聽程式。例如，您可以在 flash.events.Event.ADDED_TO_STAGE 事件處理常式中初始化這些偵聽程式。此事件可保證您的應用程式在舞台上是可見的：

```
public class SimpleStageVideo extends Sprite
    private var nc:NetConnection;
    private var ns:NetStream;

    public function SimpleStageVideo()
    {
        // Constructor for SimpleStageVideo class
        // Make sure the app is visible and stage available
        addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
    }

    private function onAddedToStage(event:Event):void
    {
        //...
        // Connections
        nc = new NetConnection();
        nc.connect(null);
        ns = new NetStream(nc);
        ns.addEventListener(NetStatusEvent.NET_STATUS, onNetStatus);
        ns.client = this;

        // Screen
        video = new Video();
        video.smoothing = true;

        // Video Events
        // the StageVideoEvent.STAGE_VIDEO_STATE informs you whether
        // StageVideo is available
        stage.addEventListener(StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY,
            onStageVideoState);
        // in case of fallback to Video, listen to the VideoEvent.RENDER_STATE
        // event to handle resize properly and know about the acceleration mode running
        video.addEventListener(VideoEvent.RENDER_STATE, videoStateChange);
        //...
    }
```

使用 StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 事件

在 StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 處理常式中，決定是否根據 StageVideo 的可用性來使用 Video 或 StageVideo· 物件。如果將 StageVideoAvailabilityEvent.availability 屬性設定為 StageVideoAvailability.AVAILABLE，請使用 StageVideo。在此情況下，您可以依賴 Stage.stageVideos 向量，以包含一或多個 StageVideo 物件。從 Stage.stageVideos 屬性取得 StageVideo 物件，並附加 NetStream 物件。因為 StageVideo 物件永遠都會顯示在背景，因此請移除任何現有的 Video 物件 (永遠都在前景)。您也可以利用此事件處理常式來新增 StageVideoEvent.RENDER_STATE 事件的偵聽程式。

如果將 StageVideoAvailabilityEvent.availability 屬性設定為 StageVideoAvailability.UNAVAILABLE，請勿使用 StageVideo 或存取 Stage.stageVideos 向量。在此狀況下，將 NetStream 物件附加至 Video 物件。最後，將 StageVideo 或 Video 物件新增至舞台，並呼叫 NetStream.play()。

下列程式碼顯示如何處理 StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 事件：

```
private var sv:StageVideo;
private var video:Video;

private function onStageVideoState(event:StageVideoAvailabilityEvent):void
{
    // Detect if StageVideo is available and decide what to do in toggleStageVideo
    toggleStageVideo(event.availability == StageVideoAvailability.AVAILABLE);
}

private function toggleStageVideo(on:Boolean):void
{
    // To choose StageVideo attach the NetStream to StageVideo
    if (on)
    {
        stageVideoInUse = true;
        if ( sv == null )
        {
            sv = stage.stageVideos[0];
            sv.addEventListener(StageVideoEvent.RENDER_STATE, stageVideoStateChange);
                sv.attachNetStream(ns);
        }

        if (classicVideoInUse)
        {
            // If you use StageVideo, remove from the display list the
            // Video object to avoid covering the StageVideo object
            // (which is always in the background)
            stage.removeChild ( video );
            classicVideoInUse = false;
        }
    } else
    {
        // Otherwise attach it to a Video object
        if (stageVideoInUse)
            stageVideoInUse = false;
        classicVideoInUse = true;
        video.attachNetStream(ns);
        stage.addChildAt(video, 0);
    }

    if ( !played )
    {
        played = true;
        ns.play(FILE_NAME);
    }
}
```

重要事項：應用程式第一次存取位於 Stage.stageVideos[0] 的 Vector 元素，預設矩形設定為 0,0,0,0, 且平移與縮放屬性均使用預設值。請一律將這些值重設為您偏好的設定。您可使用 StageVideoEvent.RENDER_STATE 或 VideoEvent.RENDER_STATE 事件目標的 videoWidth 與 videoHeight 屬性來計算視訊檢視區域大小。

請在開始使用舞台視訊，下載此範本應用程式的完整原始碼。

使用 StageVideoEvent.RENDER_STATE 與 VideoEvent.RENDER_STATE 事件

StageVideo 與 Video 物件會在顯示環境變更時傳送通知應用程式的事件。這些事件包括 StageVideoEvent.RENDER_STATE 與 VideoEvent.RENDER_STATE。

StageVideo 或 Video 物件會在附加與開始播放 NetStream 物件時傳送顯示狀態事件。當顯示環境變更時，也會傳送此事件；例如調整視訊檢視區域大小時。使用這些通知可將您的檢視區域重設為事件目標物件目前的 videoHeight 與 videoWidth 值。

報告的顯示狀態包括：

- RENDER_STATUS_UNAVAILABLE
- RENDER_STATUS_SOFTWARE
- RENDER_STATUS_ACCELERATED

無論目前播放視訊的類別為何，顯示狀態都會指出硬體加速解碼的使用情形。查看 StageVideoEvent.status 屬性可瞭解必要的解碼是否可用。若此屬性設定為 “unavailable”，StageVideo 物件將無法播放視訊。若是此狀態，您必須立即將 NetStream 物件重新附加至 Video 物件。其他狀態則是通知您的應用程式關於目前的顯示狀況。

下表描述 Flash Player 中的 StageVideoEvent 與 VideoEvent 物件的所有顯示狀態值的含意：

	VideoStatus.ACCELERATED	VideoStatus.SOFTWARE	VideoStatus.UNAVAILABLE
StageVideoEvent	解碼和呈現都在硬體中進行 (最佳效能)。	在硬體中呈現，在軟體中解碼 (可接受的效能)。	GPU 資源不足，無法處理視訊，未顯示任何項目。回復為 **Video** 物件。
VideoEvent	在軟體中呈現，在硬體中解碼。(只有在較新的桌上型電腦系統上，才能提供可接受的效能。效果較差的全螢幕效能。)	在軟體中呈現，在軟體中解碼。(最差的效能。效果較差的全螢幕效能。)	(不適用)

顏色空間

舞台視訊可使用基礎硬體功能，以支援顏色空間。SWF 內容可以提供指出其慣用顏色空間的中繼資料。不過，裝置圖形硬體會決定是否可以使用顏色空間。某個裝置可能支援數個顏色空間，而其他裝置則不支援。如果硬體不支援要求的顏色空間，Flash Player 會嘗試在支援的顏色空間尋找最接近相符的顏色空間。

若要查詢硬體支援哪些顏色空間，請使用 StageVideo.colorSpaces 屬性。此屬性會以 String 向量傳回支援的顏色空間清單。

```
var colorSpace:Vector.<String> = stageVideo.colorSpaces();
```

若要瞭解目前播放的視訊正在使用哪些顏色空間，請檢查 StageVideoEvent.colorSpace 屬性。請在 StageVideoEvent.RENDER_STATE 事件的事件處理常式中檢查此屬性：

```
var currColorSpace:String;

//StageVideoEvent.RENDER_STATE event handler
private function stageVideoRenderState(event:Object):void
{
    //...
    currColorSpace = (event as StageVideoEvent).colorSpace;
    //...
}
```

如果 Flash Player 找不到不支援的顏色空間的替代方案，舞台視訊會使用預設的顏色空間 BT.601。例如，具有 H.264 編碼的視訊串流通常會使用 BT.709 顏色空間。如果裝置硬體不支援 BT.709，則 colorSpace 屬性會傳回 "BT601"。"unknown" 的 StageVideoEvent.colorSpace 值指出硬體不提供查詢顏色空間的方法。

如果您的應用程式將目前的顏色空間視為無法接受，則可以選擇從 StageVideo 物件切換至 Video 物件。Video 類別支援透過軟體複合的所有顏色空間。

第 26 章 使用攝影機

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

連接至使用者電腦的攝影機可以做為使用 ActionScript 來顯示和操作視訊資料的來源。Camera 類別是內建在 ActionScript 的機制，可用於電腦或裝置攝影機。

在行動裝置上，您也可以使用 CameraUI 類別。CameraUI 類別會啟動個別攝影機應用程式，以允許使用者擷取靜態影像或視訊。當使用者完成時，您的應用程式可透過 MediaPromise 物件存取影像或視訊。

更多說明主題

Christian Cantrell：如何以跨平台方式使用 CameraUI

Michaël Chaize：Android、AIR 及攝影機

瞭解 Camera 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Camera 物件可讓您連接使用者的本機攝影機，並將視訊廣播回到本機使用者，或廣播至遠端伺服器 (例如 Flash Media Server)。

使用 Camera 類別，您可以存取下列各種關於使用者攝影機的資訊：

- 使用者電腦或裝置上安裝的攝影機都是可供使用的
- 是否有安裝攝影機
- 允許或拒絕 Flash Player 存取使用者的攝影機
- 哪個是目前為作用中的攝影機
- 所擷取視訊的寬度與高度

Camera 類別包括幾個有用的方法與屬性，讓您可以搭配攝影機物件使用。例如，靜態的 Camera.names 屬性包含目前安裝在使用者電腦上的攝影機名稱陣列。您也可以使用 name 屬性，顯示目前作用中攝影機的名稱。

備註：跨網路串流攝影機視訊時，您應該永遠處理網路中斷。網路中斷發生的原因有很多種，特別是在行動裝置上。

在螢幕上顯示攝影機內容

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

連接至攝影機所需的程式碼會比使用 NetConnection 與 NetStream 類別以載入視訊來得少。Camera 類別也可能很快地變得難以處理，因為您必須先讓使用者允許 Flash Player 連接至他們的攝影機，才可以存取這些攝影機。

下列程式碼將示範如何使用 Camera 類別，連接至使用者的本機攝影機：

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

備註：Camera 類別並沒有建構函式方法。為了建立新的 Camera 實體，您可以使用靜態的 Camera.getCamera() 方法。

設計攝影機應用程式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您撰寫應用程式以連接使用者的攝影機時，必須在程式碼中說明下列項目：

- 檢查使用者目前是否已安裝攝影機。處理沒有攝影機的情況。
- (僅適用於 Flash Player) 檢查使用者是否已明確允許存取攝影機。基於安全理由，播放程式會顯示 「Flash Player 設定」對話方塊，讓使用者允許或拒絕存取其攝影機。這將可避免 Flash Player 在未取得使用者的允許下，連接至他們的攝影機並廣播視訊串流。如果使用者按一下允許，您的應用程式就可以連接至使用者的攝影機。如果使用者按一下拒絕，您的應用程式將無法存取使用者的攝影機。您的應用程式一定要妥善處理這兩種情形。
- 僅適用於 AIR，請檢查應用程式支援的裝置描述檔是否支援 Camera 類別。
- 在行動瀏覽器中不支援 Camera 類別。
- 在使用 GPU 顯示模式的行動 AIR 應用程式中不支援 Camera 類別。
- 在行動裝置上，一次只能有一個攝影機為作用中。

更多說明主題

Christophe Coenraets：多人視訊井字遊戲

Mark Doherty：Android 雷達圖應用程式 (來源)

連接至使用者的攝影機

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

連接至使用者攝影機的第一步就是建立新的攝影機實體，作法是建立 Camera 類型的變數，並將它初始化以傳回靜態 Camera.getCamera() 方法的值。

下一步就是建立新的視訊物件，並將 Camera 物件附加至該視訊物件。

第三步就是將該視訊物件加入顯示清單。您需要執行步驟 2 與步驟 3，因為 Camera 類別並不會擴充 DisplayObject 類別，所以無法將它直接加入至顯示清單。若要顯示攝影機所擷取的視訊，您可以建立新的視訊物件並呼叫 attachCamera() 方法。

下列程式碼會顯示這三個步驟：

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

請注意，如果使用者沒有安裝攝影機，應用程式將不會顯示任何項目。

針對實際的情況，您還必須為應用程式執行額外的步驟。如需詳細資訊，請參閱第 445 頁 「確定已安裝攝影機」和第 446 頁「偵測存取攝影機的權限」。

更多說明主題

Lee Brimelow：如何存取 Android 裝置上的攝影機

確定已安裝攝影機

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在您嘗試針對攝影機實體使用任何方法或屬性之前，必須確定使用者是否已安裝攝影機。有兩個方法可以檢查使用者是否已安裝攝影機：

- 檢查靜態的 Camera.names 屬性，它包含可用攝影機名稱的陣列。一般而言，這個陣列將會有一個或較少的字串，因為大部分的使用者不太可能會同時安裝一個以上的攝影機。下列程式碼將說明您如何檢查 Camera.names 屬性，以查看使用者是否有任何可用的攝影機：

```
if (Camera.names.length > 0)
{
    trace("User has at least one camera installed.");
    var cam:Camera = Camera.getCamera(); // Get default camera.
}
else
{
    trace("User has no cameras installed.");
}
```

- 檢查靜態 Camera.getCamera() 方法的傳回值。如果沒有可使用的攝影機或是未安裝攝影機，此方法會傳回 null，否則它會傳回 Camera 物件的參考。下列程式碼將說明您如何檢查 Camera.getCamera() 方法，以查看使用者是否有任何可用的攝影機：

```
var cam:Camera = Camera.getCamera();
if (cam == null)
{
    trace("User has no cameras installed.");
}
else
{
    trace("User has at least 1 camera installed.");
}
```

由於 Camera 類別不會擴充 DisplayObject 類別，因此無法利用 addChild() 方法直接加入至顯示清單。為了顯示攝影機所擷取的視訊，您必須建立新的 Video 物件並針對 Video 實體呼叫 attachCamera() 方法。

此程式碼片段會顯示如果有可用的攝影機，那麼該如何連接該部攝影機；如果沒有，應用程式就不會顯示任何項目：

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
```

行動裝置攝影機

行動瀏覽器的 Flash Player 執行階段不支援 Camera 類別。

在行動裝置的 AIR 應用程式上，您可存取攝影機或裝置上的攝影機。在行動裝置上，您可以同時使用前後攝影機，但是任何時候只會顯示一台攝影機 (連接第二台攝影機將會中斷第一台的連接)。前方攝影機是水平鏡像在 iOS，但在 Android 則非如此。

偵測存取攝影機的權限

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 AIR 應用程式安全執行程序中，應用程式可以存取任何攝影機，而不需取得使用者的權限。不過，在 Android 上，應用程式必須在應用程式描述器中指定 Android CAMERA 權限。

使用者必須明確允許 Flash Player 存取攝影機，Flash Player 才能顯示攝影機的輸出內容。呼叫 attachCamera() 方法時，Flash Player 會顯示「Flash Player 設定」對話方塊，提示使用者允許或拒絕 Flash Player 存取攝影機與麥克風。如果使用者按一下「允許」按鈕，Flash Player 就會在「舞台」上的 Video 實體中顯示攝影機的輸出內容。如果使用者按一下「拒絕」按鈕，Flash Player 就無法連接至攝影機，而且 Video 物件也不會顯示任何項目。

如果您要偵測使用者是否允許 Flash Player 存取攝影機，可以偵聽攝影機的 status 事件 (StatusEvent.STATUS)，如下列程式碼所示：

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
function statusHandler(event:StatusEvent):void
{
    // This event gets dispatched when the user clicks the "Allow" or "Deny"
    // button in the Flash Player Settings dialog box.
    trace(event.code); // "Camera.Muted" or "Camera.Unmuted"
}
```

只要使用者按一下「允許」或「拒絕」，就會呼叫 statusHandler() 函數。您可以使用下列兩個方法中的其中一個來偵測使用者按下了哪一個按鈕：

- statusHandler() 函數的 event 參數包含 code 屬性，這個屬性含有「Camera.Muted」或「Camera.Unmuted」字串。如果值為「Camera.Muted」，表示使用者按下了「拒絕」按鈕，此時 Flash Player 將無法存取攝影機。在下列程式碼片段中，您會看到這種範例：

  ```
  function statusHandler(event:StatusEvent):void
  {
      switch (event.code)
      {
          case "Camera.Muted":
              trace("User clicked Deny.");
              break;
          case "Camera.Unmuted":
              trace("User clicked Accept.");
              break;
      }
  }
  ```

- Camera 類別包含名為 muted 的唯讀屬性，它會指定使用者在 Flash Player「私用」面板中已拒絕存取攝影機 (true) 或允許存取攝影機 (false)。在下列程式碼片段中，您會看到這種範例：

```
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("User clicked Deny.");
    }
    else
    {
        trace("User clicked Accept.");
    }
}
```

透過檢查要傳送的狀態事件，您可以撰寫程式碼以處理使用者接受或拒絕存取攝影機，並適當進行清理。例如，如果使用者按一下 「拒絕」按鈕，您可以對使用者顯示訊息，說明如果他們要參與視訊交談，就必須按一下 「允許」；或者您可以改為確認顯示清單上的 Video 物件是否已刪除，以釋放系統資源。

在 AIR 中，Camera 物件不會傳送狀態事件，因為使用攝影機的權限不是動態的。

最佳化攝影機視訊品質

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

根據預設，Video 類別的新實體是 320 像素寬 x 240 像素高。為了最佳化視訊品質，您應該永遠確定視訊物件的尺寸與攝影機物件傳回的視訊尺寸相同。您可以使用 Camera 類別的 width 和 height 屬性取得攝影機物件的寬度與高度，接著再設定視訊物件的 width 和 height 屬性以符合攝影機物件的尺寸，或是您可以將攝影機的寬度與高度傳遞至 Video 類別的建構函式方法，如下列程式碼片段所示：

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video(cam.width, cam.height);
    vid.attachCamera(cam);
    addChild(vid);
}
```

既然 getCamera() 方法會傳回攝影機物件的參考 (或是如果沒有可用的攝影機，就會傳回 null)，您就可以存取攝影機的方法與屬性，即使使用者拒絕存取其攝影機也一樣。這可讓您使用攝影機原始的高度與寬度來設定視訊實體的大小。

```
var vid:Video;
var cam:Camera = Camera.getCamera();

if (cam == null)
{
    trace("Unable to locate available cameras.");
}
else
{
    trace("Found camera: " + cam.name);
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    vid = new Video();
    vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("Unable to connect to active camera.");
    }
    else
    {
        // Resize Video object to match camera settings and
        // add the video to the display list.
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
    }
    // Remove the status event listener.
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}
```

如需有關全螢幕模式的詳細資訊，請參閱第 139 頁 「設定 Stage 的屬性」中的 「全螢幕模式」一節。

監控攝影機狀態

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

攝影機類別包含數個屬性，可讓您監視 Camera 物件的目前狀態。例如，下列程式碼會使用 Timer 物件以及顯示清單上的文字欄位實體，顯示攝影機的數個屬性：

```
var vid:Video;
var cam:Camera = Camera.getCamera();
var tf:TextField = new TextField();
tf.x = 300;
tf.autoSize = TextFieldAutoSize.LEFT;
addChild(tf);

if (cam != null)
{
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    vid = new Video();
    vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
    if (!cam.muted)
    {
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
        t.start();
    }
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}

var t:Timer = new Timer(100);
t.addEventListener(TimerEvent.TIMER, timerHandler);
function timerHandler(event:TimerEvent):void
{
    tf.text = "";
    tf.appendText("activityLevel: " + cam.activityLevel + "\n");
    tf.appendText("bandwidth: " + cam.bandwidth + "\n");
    tf.appendText("currentFPS: " + cam.currentFPS + "\n");
    tf.appendText("fps: " + cam.fps + "\n");
    tf.appendText("keyFrameInterval: " + cam.keyFrameInterval + "\n");
    tf.appendText("loopback: " + cam.loopback + "\n");
    tf.appendText("motionLevel: " + cam.motionLevel + "\n");
    tf.appendText("motionTimeout: " + cam.motionTimeout + "\n");
    tf.appendText("quality: " + cam.quality + "\n");
}
```

每 1/10 秒 (100 毫秒) 就會傳送 Timer 物件的 timer 事件，而且 timerHandler() 函數會更新顯示清單上的文字欄位。

第 27 章 使用數位版權管理

Flash Player 10.1 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

Adobe® Access™ 是內容保護解決方案，可透過順暢地存取優質內容，讓內容擁有者、發行者和廣告商瞭解新收益來源。發行者可使用 Adobe Access 來加密內容、建立原則及核發授權。Adobe Flash Player 和 Adobe AIR 併入 DRM 元件庫 (即 Adobe Access 模組)。這個模組可讓執行階段與 Adobe Access 授權伺服器進行通訊，以及播放保護的內容。因此，執行階段會完成受 Adobe Access 保護之內容及 Flash Media Server 所散佈內容的效期。

內容提供者可以使用 Adobe Access 提供免費內容和優質內容。例如，消費者想要看沒有廣告的電視節目。則消費者可以註冊並付費給內容發行者，然後輸入其使用者憑證以存取和播放沒有廣告的節目。

另一個範例是，消費者想要在出差時，在無法存取網際網路的情況下離線檢視內容。AIR 應用程式支援這樣的離線工作流程。在註冊並付費給發行者取得內容之後，使用者就可以從發行者網站存取並下載內容及相關聯的 AIR 應用程式。使用 AIR 應用程式，使用者可以在允許的期間內離線檢視內容。該內容也可以使用網域來和位於相同裝置群組內的其它裝置共享 (**3.0** 新增功能)。

Adobe Access 也支援匿名存取，不需要進行使用者驗證。例如，發行者可以使用匿名存取來散佈廣告支援的內容。發行者也可以使用匿名存取功能，允許目前的內容在指定天數內供人自由存取。內容提供者也可以指定與限制其內容所需的播放程式類型和版本。

使用者嘗試在 Flash Player 或 Adobe AIR 中播放保護的內容時，您的應用程式必須呼叫 DRM API。DRM API 會起始播放受保護內容的工作流程。執行階段會透過 Adobe Access 模組來聯絡授權伺服器。授權伺服器可視需要驗證使用者，並核發授權，以允許播放保護的內容。執行階段會收到授權，並將內容解密予以播放。

這裡會說明如何讓應用程式播放受 Adobe Access 保護的內容。您並不需要瞭解如何使用 Adobe Access 來加密內容或維護原則。不過，這是假設您與 Adobe Access 授權伺服器進行通訊，來驗證使用者並擷取授權。也假設您特別設計應用程式，來播放 Adobe Access 保護的內容。

如需有關 Adobe Access 的概觀 (包括建立原則)，請參閱 Adobe Access 隨附的文件。

更多說明主題

flash.net.drm 套件

flash.net.NetConnection

flash.net.NetStream

瞭解保護的內容工作流程

Flash Player 10.1 以及更新的版本，**Adobe AIR 2.0** 以及更新的版本

重要：Flash Player 11.5 及更新版本會整合 Adobe Access 模組，因此不需要執行更新步驟 (呼叫 SystemUpdater.update(SystemUpdaterType.DRM))。這包括下列瀏覽器和平台：

- Flash Player 11.5 ActiveX 控制項，適用於所有平台，但使用 Intel 處理器的 Windows 8 的 Internet Explorer 除外
- Flash Player 11.5 外掛程式，適用於所有瀏覽器
- Adobe AIR (桌上型電腦及行動裝置)

這表示，在下列情況中 「還是需要」更新步驟：

- 使用 Intel 處理器的 Windows 8 的 Internet Explorer
- Flash Player 11.4 及以前版本，但 Google Chrome 22 及更新版本 (所有平台) 或 21 及更新版本 (Windows) 除外

備註：在具有 Flash Player 11.5 或更新版本的系統上，您還是可以安全地呼叫 SystemUpdater.update(SystemUpdaterType.DRM)，但不會下載任何項目。

下列高階工作流程顯示應用程式如何擷取和播放保護的內容。這個工作流程假設應用程式特別設計成播放 Adobe Access 保護的內容：

1. 取得內容中繼資料。
2. 如有需要，處理 Flash Player 的更新。
3. 確認本機是否有授權。如果有，則載入它，並前往步驟 7。否則，請前往步驟 4。
4. 確認是否需要驗證。如果否，可以前往步驟 7。
5. 如果需要驗證，則向使用者取得驗證憑證，並將它們傳遞給授權伺服器。
6. 如果需要註冊網域，請加入網域 (AIR 3.0 或更新版本)。
7. 驗證成功之後，從伺服器下載授權。
8. 播放內容。

如果未發生錯誤，而且使用者已獲授權檢視內容，NetStream 物件就會傳送 DRMStatusEvent 物件。應用程式就會開始播放。DRMStatusEvent 物件保存了相關的憑證資訊，以識別使用者的原則和權限。例如，它保留的資訊是關於內容是否可以離線使用或授權何時到期。應用程式可以利用這份資料，告知使用者其原則的狀態。例如，應用程式可在狀態列上顯示使用者仍能檢視內容的剩餘天數。

如果允許使用者離線存取，便會快取憑證，並將加密的內容下載到使用者的電腦。在授權快取持續時間定義的時間內，都可以存取該內容。事件的 detail 屬性會包含 "DRM.voucherObtained"。應用程式會決定要在本機何處儲存內容以供離線檢視。您也可以使用 DRMManager 類別預先載入憑證。

備註：AIR 與 Flash Player 都支援快取和預先載入憑證。不過，只有 AIR 才支援下載和儲存加密的內容。

應用程式必須負責明確處理錯誤事件。其中包括使用者已輸入有效的憑證 (Credential)，但保護加密內容的憑證 (Voucher) 卻限制其存取內容。例如，如果尚未付費以取得權限，則驗證的使用者無法存取內容。當同一發行者的兩位註冊會員，嘗試共用只有一位會員付費的內容時，也會發生同樣的情形。應用程式必須通知使用者發生的錯誤，並提供解決方法建議。典型的解決方法建議是提供如何註冊及付費取得檢視權限的指示。

詳細的 API 工作流程

Flash Player 10.1 以及更新的版本，**AIR 2.0** 以及更新的版本

這個工作流程提供受保護內容工作流程的較詳細檢視。這個工作流程說明播放 Adobe Access 保護之內容的特定 API。

1. 使用 URLLoader 物件，載入受保護內容之中繼資料檔案的位元組。將這個物件設為變數 (例如 metadata_bytes)。

   所有受 Adobe Access 控制的內容都具有 Adobe Access 中繼資料。封裝內容時，這個中繼資料可以與內容一起儲存為不同的中繼資料檔案 (.metadata)。如需詳細資訊，請參閱 Adobe Access 文件。

2. 建立 DRMContentData 實體。將這個程式碼放入 try-catch 區塊：

   new DRMContentData(***metadata_bytes***)

   其中 metadata_bytes 是在步驟 1 中取得的 URLLoader 物件。

3. (僅限 Flash Player) 執行階段會檢查 Adobe Access 模組。若找不到，便會擲回 DRMErrorEvent 錯誤碼 3344 或 DRMErrorEvent 錯誤碼 3343 的 IllegalOperationError。

若要處理此錯誤，請使用 SystemUpdater API 下載 Adobe Access 模組。下載此模組之後，SystemUpdater 物件會傳送 COMPLETE 事件。請納入這個事件的事件偵聽程式，以便傳送這個事件時回到步驟 2。下列程式碼會示範這些步驟：

```
flash.system.SystemUpdater.addEventListener(Event.COMPLETE, updateCompleteHandler);
flash.system.SystemUpdater.update(flash.system.SystemUpdaterType.DRM)
```

```
private function updateCompleteHandler (event:Event):void {
    /*redo step 2*/
    drmContentData = new DRMContentData(metadata_bytes);
}
```

如果必須更新播放程式本身，則會傳送狀態事件。如需有關處理這個事件的詳細資訊，請參閱第 465 頁 「更新事件的偵聽」。

備註：在 AIR 應用程式中，AIR 安裝程式會處理 Adobe Access 模組的更新以及必要的執行階段更新。

4 建立偵聽程式，以偵聽從 DRMManager 物件傳送的 DRMStatusEvent 和 DRMErrorEvent：

```
DRMManager.addEventListener(DRMStatusEvent.DRM_STATUS, onDRMStatus);
DRMManager.addEventListener(DRMErrorEvent.DRM_ERROR, onDRMError);
```

在 DRMStatusEvent 偵聽程式中，檢查憑證是否有效 (非 null)。在 DRMErrorEvent 偵聽程式中，處理 DRMErrorEvent。請參閱第 458 頁 「使用 DRMStatusEvent 類別」和第 462 頁 「使用 DRMErrorEvent 類別」。

5 載入內容播放所需的憑證 (授權)。

首先，嘗試載入本機儲存的授權，以播放內容：

```
DRMManager.loadvoucher(drmContentData, LoadVoucherSetting.LOCAL_ONLY)
```

載入完成之後，DRMManager 物件會傳送 DRMStatusEvent.DRM_Status。

6 如果 DRMVoucher 物件不是 null，則為有效憑證。跳至步驟 13。

7 如果 DRMVoucher 物件是 null，請檢查此內容原則要求的驗證方法。使用 DRMContentData.authenticationMethod 屬性。

8 如果驗證方法是 ANONYMOUS，請前往步驟 13。

9 如果驗證方法是 USERNAME_AND_PASSWORD，則您的應用程式必須提供機制，讓使用者輸入憑證。將這些憑證傳遞給授權伺服器，以驗證使用者：

```
DRMManager.authenticate(metadata.serverURL, metadata.domain, username, password)
```

如果驗證失敗，DRMManager 會傳送 DRMAuthenticationErrorEvent，如果驗證成功，則會傳送 DRMAuthenticationCompleteEvent。建立這些事件的偵聽程式。

10 如果驗證方法是 UNKNOWN，便必須使用自訂驗證方法。在此情況下，內容提供者已安排透過不使用 ActionScript 3.0 API 的頻外方法來完成驗證。自訂驗證程序必須產生可傳遞給 DRMManager.setAuthenticationToken() 方法的驗證權杖。

11 如果驗證失敗，您的應用程式必須回到步驟 9。確認應用程式具有機制可以處理和限制重複的驗證失敗。例如，嘗試三次之後，會向使用者顯示一則訊息，指出驗證失敗，因此無法播放內容。

12 若要使用儲存的權杖，而非提示使用者輸入憑證，則請使用 DRMManager.setAuthenticationToken() 方法設定權杖。然後，從授權伺服器下載授權，並以步驟 8 的方式播放內容。

13 (選擇性) 如果驗證成功，則可以擷取驗證權杖 (其為記憶體中快取的位元組陣列)。使用 DRMAuthenticationCompleteEvent.token 屬性，取得這個權杖。您可以儲存並使用驗證權杖，讓使用者不需要重複輸入這個內容的憑證。授權伺服器會判斷驗證權杖的有效期間。

14 如果驗證成功，則從授權伺服器下載授權：

```
DRMManager.loadvoucher(drmContentData, LoadVoucherSetting.FORCE_REFRESH)
```

載入完成之後，DRMManager 物件會傳送 DRMStatusEvent.DRM_STATUS。偵聽這個事件，您可以在傳送這個事件後播放內容。

15 建立 NetStream 物件並呼叫它的 play() 方法來播放視訊：

```
stream = new NetStream(connection);
stream.addEventListener(DRMStatusEvent.DRM _STATUS, drmStatusHandler);
stream.addEventListener(DRMErrorEvent.DRM_ERROR, drmErrorHandler);
stream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
stream.client = new CustomClient();
video.attachNetStream(stream);
stream.play(videoURL);
```

DRMContentData 與工作階段物件

DRMContentData 建立後，便會做為參考 Flash Player DRM 模組的工作階段物件使用。收到此 DRMContentData 的所有 DRMManager API 都將使用特定的 DRM 模組。不過，有 2 個 DRMManager API 不使用 DRMContentData。這些規則包括：

1 authenticate()

2 setAuthenticationToken()

Since 因為沒有相關聯的 DRMContentData，所以叫用這些 DRMManager API 會使用磁碟中最新的 DRM 模組。如果在應用程式的 DRM 工作流程進行期間更新 DRM 模組，便可能會發生問題。請考量下列情況：

1 應用程式建立使用 AdobeCP1 做為 DRM 模組的 DRMContentData 物件 contentData1。

2 應用程式叫用 DRMManager.authenticate(contentData1.serverURL,...) 方法。

3 應用程式叫用 DRMManager.loadVoucher(contentData1, ...) 方法。

如果在應用程式能夠進行步驟 2 之前便更新 DRM 模組，DRMManager.authenticate() 方法便會結束使用 AdobeCP2 做為 DRM 模組來進行驗證。因為步驟 3 的 loadVoucher() 方法仍然使用 AdobeCP1 做為 DRM 模組，所以會失敗。更新可能是由於另一個應用程式叫用 DRM 模組更新所致。在應用程式啟動時叫用 DRM 模組更新，便可避免這種情況。

DRM 相關事件

應用程式嘗試播放保護的內容時，執行階段會傳送多個事件：

- DRMDeviceGroupErrorEvent (僅限 AIR)，由 DRMManager 所傳送
- DRMAuthenticateEvent (僅限 AIR)，由 NetStream 所傳送
- DRMAuthenticationCompleteEvent，由 DRMManager 所傳送
- DRMAuthenticationErrorEvent，由 DRMManager 所傳送
- DRMErrorEvent，由 NetStream 和 DRMManager 所傳送
- DRMStatusEvent，由 NetStream 和 DRMManager 所傳送
- StatusEvent
- NetStatusEvent 請參閱第 465 頁 「更新事件的偵聽」。

若要支援 Adobe Access 保護的內容，請新增事件偵聽程式以處理 DRM 事件。

預先載入憑證以供離線播放

Adobe AIR 1.5 以及更新的版本

您可以預先載入播放受 Adobe Access 保護之內容所需的憑證 (授權)。不論是否有作用中的網際網路連線，預先載入憑證便可讓使用者檢視內容。(預先載入程序本身需要網際網路連線)。您可以使用 NetStream 類別 preloadEmbeddedMetadata() 方法和 DRMManager 類別來預先載入憑證。在 AIR 2.0 以及更新的版本中，您可以使用 DRMContentData 物件直接預先載入憑證。因為這個技術可讓您更新 DRMContentData 物件 (不論其內容為何)，所以偏好使用這個技術。(preloadEmbeddedData() 方法會從內容中取出 DRMContentData。)

使用 DRMContentData

Adobe AIR 2.0 以及更新的版本

下列步驟描述為使用 DRMContentData 物件的受保護媒體檔案預先載入憑證的工作流程。

1. 取得所封裝內容的二進位中繼資料。如果使用 Adobe Access Java Reference Packager，則會以副檔名 .metadata 自動產生這個中繼資料檔案。例如，您可以使用 URLLoader 類別來下載此中繼資料。
2. 建立 DRMContentData 物件，將中繼資料傳遞給建構函式：

```
var drmData:DRMContentData = new DRMContentData( metadata );
```

3. 其餘步驟則與第 450 頁 「瞭解保護的內容工作流程」所述的工作流程相同。

使用 preloadEmbeddedMetadata()

Adobe AIR 1.5 以及更新的版本

下列步驟說明如何使用 preloadEmbeddedMetadata() 預先載入受 DRM 保護媒體檔之憑證的工作流程：

1. 下載並儲存媒體檔 (DRM 中繼資料只能從本機儲存的檔案中預先載入)。
2. 建立 NetConnection 和 NetStream 物件，並實作 NetStream 用戶端物件的 onDRMContentData() 和 onPlayStatus() 回呼函數。
3. 建立一個 NetStreamPlayOptions 物件，並將 stream 屬性設定為本機媒體檔的 URL。
4. 呼叫 NetStream 的 preloadEmbeddedMetadata()，傳遞 NetStreamPlayOptions 物件來找出要剖析的媒體檔。
5. 如果媒體檔含有 DRM 中繼資料，便會叫用 onDRMContentData() 回呼函數。中繼資料會做為 DRMContentData 物件傳遞至此函數。
6. 使用 DRMManager 的 loadVoucher() 方法，透過 DRMContentData 物件來取得憑證。

   如果 DRMContentData 物件的 authenticationMethod 屬性值是 flash.net.drm.AuthenticationMethod.USERNAME_AND_PASSWORD，則請先在媒體權限伺服器上驗證使用者，再載入憑證。DRMContentData 物件的 serverURL 和 domain 屬性可以隨著使用者的憑證一同傳遞至 DRMManager 的 authenticate() 方法。
7. 檔案剖析完畢時便會叫用 onPlayStatus() 回呼函數。如果沒有呼叫 onDRMContentData() 函數，檔案不會包含取得憑證所需的中繼資料。缺少這個呼叫也可能表示 Adobe Access 未保護此檔案。

AIR 的下列程式碼範例說明如何預先載入本機媒體檔的憑證：

```
package
{
import flash.display.Sprite;
import flash.events.DRMAuthenticationCompleteEvent;
import flash.events.DRMAuthenticationErrorEvent;
import flash.events.DRMErrorEvent;
import flash.ev ents.DRMStatusEvent;
import flash.events.NetStatusEvent;
import flash.net.NetConnection;
import flash.net.NetStream;
import flash.net.NetStreamPlayOptions;
import flash.net.drm.AuthenticationMethod;
import flash.net.drm.DRMContentData;
import flash.net.drm.DRMManager;
import flash.net.drm.LoadVoucherSetting;
public class DRMPreloader extends Sprite
{
    private var videoURL:String = "app-storage:/video.flv";
    private var userName:String = "user";
    private var password:String = "password";
    private var preloadConnection:NetConnection;
    private var preloadStream:NetStream;
    private var drmManager:DRMManager = DRMManager.getDRMManager();
    private var drmContentData:DRMContentData;
    public function DRMPreloader():void {
        drmManager.addEventListener(
            DRMAuthenticationCompleteEvent.AUTHENTICATION_COMPLETE,
            onAuthenticationComplete);
        drmManager.addEventListener(DRMAuthenticationErrorEvent.AUTHENTICATION_ERROR,
            onAuthenticationError);
        drmManager.addEventListener(DRMStatusEvent.DRM_STATUS, onDRMStatus);
        drmManager.addEventListener(DRMErrorEvent.DRM_ERROR, onDRMError);
        preloadConnection = new NetConnection();
        preloadConnection.addEventListener(NetStatusEvent.NET_STATUS, onConnect);
        preloadConnection.connect(null);
    }

    private function onConnect( event:NetStatusEvent ):void
    {
        preloadMetadata();
    }
    private function preloadMetadata():void
    {
        preloadStream = new NetStream( preloadConnection );
        preloadStream.client = this;
        var options:NetStreamPlayOptions = new NetStreamPlayOptions();
        options.streamName = videoURL;
        preloadStream.preloadEmbeddedData( options );
    }
    public function onDRMContentData( drmMetadata:DRMContentData ):void
    {
        drmContentData = drmMetadata;
        if ( drmMetadata.authenticationMethod == AuthenticationMethod.USERNAME_AND_PASSWORD )
        {
            authenticateUser();
        }
        else
            {
                getVoucher();
            }
    }
    private function getVoucher():void
```

```
    {
        drmManager.loadVoucher( drmContentData, LoadVoucherSetting.ALLOW_SERVER );
    }

    private function authenticateUser():void
    {
        drmManager.authenticate( drmContentData.serverURL, drmContentData.domain, userName, password );
    }
    private function onAuthenticationError( event:DRMAuthenticationErrorEvent ):void
    {
        trace( "Authentication error: " + event.errorID + ", " + event.subErrorID );
    }

    private function onAuthenticationComplete( event:DRMAuthenticationCompleteEvent ):void
    {
        trace( "Authenticated to: " + event.serverURL + ", domain: " + event.domain );
        getVoucher();
    }
    private function onDRMStatus( event:DRMStatusEvent ):void
    {
        trace( "DRM Status: " + event.detail);
        trace("--Voucher allows offline playback = " + event.isAvailableOffline );
        trace("--Voucher already cached          = " + event.isLocal );
        trace("--Voucher required authentication = " + !event.isAnonymous );
    }
    private function onDRMError( event:DRMErrorEvent ):void
    {
        trace( "DRM error event: " + event.errorID + ", " + event.subErrorID + ", " + event.text );
    }
    public function onPlayStatus( info:Object ):void
    {
        preloadStream.close();
    }
}
}
```

NetStream 類別的 DRM 相關成員和事件

Flash Player 10.1 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

NetStream 類別將在 Flash Player 或 AIR 應用程式與 Flash Media Server 或本機檔案系統之間提供單向串流連線 (NetStream 類別也支援漸進式下載)。NetStream 物件是 NetConnection 物件內的通道。NetStream 類別會傳送四個 DRM 相關事件：

事件	說明
drmAuthenticate (僅限 AIR)	定義於 DRMAuthenticateEvent 類別。如果 NetStream 物件嘗試播放的受保護內容，需要在播放之前進行使用者憑證驗證，則會傳送這個事件。 此事件的屬性包括 header、usernamePrompt、passwordPrompt 和 urlPrompt 屬性，可用來取得與設定使用者的憑證。此事件將反覆發生，直到 NetStream 物件收到有效的使用者憑證為止。
drmError	定義於 DRMErrorEvent 類別，而且會在 NetStream 物件嘗試播放保護的內容以及發生 DRM 相關錯誤時傳送。例如，一旦使用者授權失敗，便會傳送 DRM 錯誤事件物件。使用者尚未購買檢視內容的權限，也會發生此錯誤。如果內容提供者不支援檢視應用程式，也會發生此錯誤。
drmStatus	定義於 DRMStatusEvent 類別。開始播放保護的內容時 (使用者通過驗證並授權播放內容時)，會傳送這個事件。DRMStatusEvent 物件包含與憑證相關的資訊。憑證資訊包括是否可以離線檢視內容，或者憑證何時到期且屆滿後再也無法檢視內容。
status	定義於 events.StatusEvent，只有在應用程式透過叫用 NetStream.play() 方法，嘗試播放保護的內容時才會傳送。狀態碼屬性的值將是 "DRM.encryptedFLV"。

NetStream 類別包括下列 DRM 特有方法 (僅用於 AIR)：

方法	說明
resetDRMVouchers()	刪除所有本機快取的數位版權管理 (DRM) 憑證資料。應用程式必須重新下載憑證，使用者才能存取加密內容。 例如，下列程式碼會從快取移除所有憑證： NetStream.resetDRMVouchers();
setDRMAuthenticationCredentials()	將一組驗證憑證 (包括使用者名稱、密碼和驗證類型) 傳遞給 NetStream 物件以進行驗證。有效的驗證類型為 "drm" 和 "proxy"。使用 "drm" 驗證類型時，提供的憑證將交由 Adobe Access 進行驗證。使用 "proxy" 驗證類型時，憑證則是由 Proxy 伺服器進行驗證，而且必須符合 Proxy 伺服器所需的憑證。例如，企業可以要求 Proxy 伺服器先驗證應用程式，然後使用者才可以存取網際網路。Proxy 選項允許此一驗證類型。除非使用匿名驗證，否則使用者在 Proxy 驗證後還是必須由 Adobe Access 進行驗證，才能取得憑證並播放內容。您可以再次使用 setDRMAuthenticationCredentials() 搭配 “drm" 選項，由 Adobe Access 進行驗證。
preloadEmbeddedMetadata()	剖析本機媒體檔，以取得內嵌的中繼資料。當找到 DRM 的相關中繼資料時，AIR 會呼叫 onDRMContentData() 回呼函數。

此外，在 AIR 中，於呼叫 preloadEmbeddedMetaData() 方法之後，NetStream 物件會呼叫 onDRMContentData() 和 onPlayStatus() 回呼函數。在媒體檔中找到 DRM 中繼資料時，便呼叫 onDRMContentData() 函數。剖析檔案之後，會呼叫 onPlayStatus() 函數。onDRMContentData() 和 onPlayStatus() 函數必須在指定給 NetStream 實體的 client 物件上定義。如果您使用相同的 NetStream 物件來預先載入憑證和播放內容，請先等待 preloadEmbeddedMetaData() 所產生的 onPlayStatus() 呼叫，再開始播放。

在 AIR 的下列程式碼中，會設定使用者名稱 ("administrator")、密碼 ("password") 和 "drm" 驗證類型，以進行使用者驗證。setDRMAuthenticationCredentials() 方法提供的憑證必須符合內容提供者已知且接受的憑證。這些憑證與允許使用者檢視內容的使用者憑證相同。此範例並未包含播放視訊及確認視訊串流連線成功的程式碼。

```
var connection:NetConnection = new NetConnection();
connection.connect(null);

var videoStream:NetStream = new NetStream(connection);

videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                             drmAuthenticateEventHandler)

private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
{
    videoStream.setDRMAuthenticationCredentials("administrator", "password", "drm");
}
```

使用 DRMStatusEvent 類別

Flash Player 10.1、**Adobe AIR 1.0** 以及更新的版本

開始順利播放受 Adobe Access 保護的內容時，NetStream 物件會傳送 DRMStatusEvent 物件 (成功表示已驗證授權，而且使用者已獲驗證並授權檢視內容)。如果允許匿名使用者存取，也將傳送 DRMStatusEvent。在此情況下會檢查授權，以確認是否允許不需要驗證的匿名使用者存取與播放內容。匿名使用者可能會因各種原因而遭到拒絕存取。例如，匿名使用者無法在授權到期時存取內容。

DRMStatusEvent 物件包含與授權相關的資訊。這類資訊包括是否可以離線提供授權，或者憑證何時到期且屆滿後再也無法提供授權。應用程式可以利用這份資料，告知使用者其原則和權限的狀態。

DRMStatusEvent 屬性

Flash Player 10.1、**Adobe AIR 1.0** 以及更新的版本

DRMStatusEvent 類別包括下列屬性。在 1.0 以後的 AIR 版本中，可以使用部分屬性。如需完整版本資訊，請參閱「Adobe® Flash® Platform 的 ActionScript® 3.0 參考」。

針對 Flash Player 10.1 不支援的屬性，DRMVoucher 類別會提供類似的 Flash Player 屬性。

屬性	說明
contentData	DRMContentData 物件，包含內嵌於內容中的 DRM 中繼資料。
detail (僅限 AIR)	用來說明狀態事件內容的字串。在 DRM 1.0 中，唯一有效的值為 DRM.voucherObtained。
isAnonymous (僅限 AIR)	指出使用者是否不需要提供驗證憑證就能存取受 Adobe Access 保護的內容 (true = 是，false = 否)。false 值表示使用者提供的使用者名稱和密碼，必須與內容提供者已知且預期的憑證相符。
isAvailableOffline (僅限 AIR)	指出使用者是否可以離線使用受 Adobe Access 保護的內容 (true = 是，false = 否)。如果要讓數位保護的內容可供離線檢視，其憑證必須快取至使用者的本機電腦。
isLocal	指出播放內容所需的憑證是否快取在本機中。
offlineLeasePeriod (僅限 AIR)	內容可供離線檢視的剩餘天數。

屬性	說明
policies (僅限 AIR)	包含自訂 DRM 屬性的自訂物件。
voucher	DRMVoucher。
voucherEndDate (僅限 AIR)	憑證到期的絕對日期，屆滿後即再也無法檢視內容。

建立 DRMStatusEvent 處理常式

Flash Player 10.1、**Adobe AIR 1.0** 以及更新的版本

下列範例將為引發事件的 NetStream 物件建立事件處理常式，以輸出其 DRM 內容狀態資訊。這個事件處理常式會新增至指向受保護內容的 NetStream 物件。

```
function drmStatusEventHandler(event:DRMStatusEvent):void
{
    trace(event);
}
function drmStatusEventHandler(event:DRMStatusEvent):void
{
    trace(event);
}
```

使用 DRMAuthenticateEvent 類別

Adobe AIR 1.0 以及更新的版本

如果 NetStream 物件嘗試播放的受保護內容，需要在播放之前才進行使用者憑證驗證，則會傳送 DRMAuthenticateEvent 物件。

DRMAuthenticateEvent 處理常式負責收集所需的憑證 (使用者名稱、密碼和類型) 並將這些值傳遞給 NetStream.setDRMAuthenticationCredentials() 方法以供驗證。每個 AIR 應用程式都必須提供某些機制以取得使用者憑證。例如，應用程式可能會提供簡單的使用者介面，讓使用者輸入使用者名稱和密碼值。同時提供機制來處理和限制重複的驗證嘗試。

DRMAuthenticateEvent 屬性

Adobe AIR 1.0 以及更新的版本

DRMAuthenticateEvent 類別包含下列屬性：

屬性	說明
authenticationType	指出提供的憑證是使用 Adobe Access (「drm」) 或 Proxy 伺服器 (「proxy」) 進行驗證。例如，必要時，「proxy」選項可讓應用程式由 Proxy 伺服器進行驗證，使用者才能存取網際網路。除非使用匿名驗證，否則使用者在 Proxy 驗證後還是必須由 Adobe Access 進行驗證，才能取得憑證並播放內容。您可以再次使用 setDRMAuthenticationcredentials() 搭配「drm」選項，由 Adobe Access 進行驗證。
header	伺服器所提供的加密內容檔案檔頭，當中含有加密內容的情境相關資訊。 此檔頭字串可以傳遞給 Flash 應用程式，讓應用程式建構使用者名稱 - 密碼對話方塊。檔頭字串可用來做為對話方塊的指示。例如，檔頭可以是「請輸入您的使用者名稱與密碼」。
netstream	起始此事件的 NetStream 物件。
passwordPrompt	由伺服器提供，要求提供密碼憑證的提示。此字串可包含所需之密碼類型的說明。
urlPrompt	由伺服器提供，要求提供 URL 字串的提示。此字串可提供使用者名稱和密碼將傳送的目標位置。
usernamePrompt	由伺服器提供，要求提供使用者名稱憑證的提示。此字串可包含所需之使用者名稱類型的說明。例如，內容提供者可要求以電子郵件地址做為使用者名稱。

先前提及的字串只能由 FMRMS 伺服器提供。Adobe Access 伺服器不使用這些字串。

建立 DRMAuthenticateEvent 處理常式

Adobe AIR 1.0 以及更新的版本

下列範例所建立的事件處理常式會將一組硬式編碼的驗證憑證傳遞給引發事件的 NetStream 物件 (範例中並未包含播放視訊及確認視訊串流連線成功的程式碼)。

```
var connection:NetConnection = new NetConnection();
connection.connect(null);

var videoStream:NetStream = new NetStream(connection);

videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                             drmAuthenticateEventHandler)

private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
{
    videoStream.setDRMAuthenticationCredentials("administrator", "password", "drm");
}
```

建立介面以擷取使用者憑證

Adobe AIR 1.0 以及更新的版本

如果保護的內容需要使用者驗證，AIR 應用程式通常必須透過使用者介面來擷取使用者的驗證憑證。

下列 Flex 範例建立了簡易的使用者介面，可用來擷取使用者憑證。此介面中的面板物件包含兩個 TextInput 物件，分別用於輸入使用者名稱和密碼憑證。面板上還有一個按鈕，將會叫用 credentials() 方法。

```
<mx:Panel x="236.5" y="113" width="325" height="204" layout="absolute" title="Login">
    <mx:TextInput x="110" y="46" id="uName"/>
    <mx:TextInput x="110" y="76" id="pWord" displayAsPassword="true"/>
    <mx:Text x="35" y="48" text="Username:"/>
    <mx:Text x="35" y="78" text="Password:"/>
    <mx:Button x="120" y="115" label="Login" click="credentials()"/>
</mx:Panel>
```

credentials() 方法是由使用者定義的方法，會將使用者名稱和密碼值傳遞給 setDRMAuthenticationCredentials() 方法。傳遞這些值之後，credentials() 方法隨即重設 TextInput 物件的值。

```
<mx:Script>
    <![CDATA[
        public function credentials():void
        {
            videoStream.setDRMAuthenticationCredentials(uName, pWord, "drm");
            uName.text = "";
            pWord.text = "";
        }
    ]]>
</mx:Script>
```

實作此類型簡易介面的方法之一，是包含面板做為新狀態的一部分。新狀態是根據擲出 DRMAuthenticateEvent 物件時的基礎狀態。下列範例包含 VideoDisplay 物件，而其 source 特質指向保護的 FLV 檔案。在這個例子中，credentials() 方法經過修改，所以它也會讓應用程式回到基礎狀態。在傳遞使用者憑證並重設 TextInput 物件值後，此方法也會這麼做。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    width="800"
    height="500"
    title="DRM FLV Player"
    creationComplete="initApp()" >

    <mx:states>
        <mx:State name="LOGIN">
            <mx:AddChild position="lastChild">
                    <mx:Panel x="236.5" y="113" width="325" height="204" layout="absolute"
                            title="Login">
                    <mx:TextInput x="110" y="46" id="uName"/>
                    <mx:TextInput x="110" y="76" id="pWord" displayAsPassword="true"/>
                    <mx:Text x="35" y="48" text="Username:"/>
                    <mx:Text x="35" y="78" text="Password:"/>
                    <mx:Button x="120" y="115" label="Login" click="credentials()"/>
                    <mx:Button x="193" y="115" label="Reset" click="uName.text='';
                            pWord.text='';"/>
                </mx:Panel>
            </mx:AddChild>
        </mx:State>
    </mx:states>

    <mx:Script>
        <![CDATA[
                import flash.events.DRMAuthenticateEvent;
            private function initApp():void
            {
                videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                                             drmAuthenticateEventHandler);
            }
```

```
            public function credentials():void
            {
                videoStream.setDRMAuthenticationCredentials(uName, pWord, "drm");
                uName.text = "";
                pWord.text = "";
                currentState='';
            }

            private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
            {
                currentState='LOGIN';
            }
        ]]>
    </mx:Script>

    <mx:VideoDisplay id="video" x="50" y="25" width="700" height="350"
        autoPlay="true"
        bufferTime="10.0"
        source="http://www.example.com/flv/Video.flv" />
</mx:WindowedApplication>
```

使用 DRMErrorEvent 類別

Flash Player 10.1 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

嘗試播放受保護內容的 NetStream 物件發生 DRM 相關錯誤時，Adobe Flash Player 和 Adobe AIR 會傳送 DRMErrorEvent 物件。如果使用者憑證在 AIR 應用程式中無效，則在使用者輸入有效的憑證或應用程式拒絕未來的嘗試之前，DRMAuthenticateEvent 物件會重複傳送。應用程式負責偵聽其它所有的 DRM 錯誤事件，以便偵測、識別及處理 DRM 相關錯誤。

縱使具有有效的使用者憑證，內容憑證的規定仍可能不讓使用者檢視加密的內容。例如，使用者嘗試以未獲授權的應用程式檢視內容時，就會遭到拒絕。所謂未獲授權的應用程式，是指未經加密內容發行者驗證的應用程式。在此情況下，便會傳送 DRMErrorEvent 物件。

如果內容已損毀，或者應用程式的版本與憑證所指定的版本不符，也有可能觸發錯誤事件。應用程式必須提供適當機制來處理錯誤。

DRMErrorEvent 屬性

Flash Player 10.1 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如需完整的錯誤清單，請參閱 「Adobe® Flash® Professional CS5 的 ActionScript® 3.0 參考」中的執行階段錯誤碼。DRM 相關錯誤是從錯誤 3300 開始。

建立 DRMErrorEvent 處理常式

Flash Player 10.1 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列範例將為引發事件的 NetStream 物件建立事件處理常式。當 NetStream 嘗試播放保護的內容而發生錯誤時，便會呼叫此處理常式。一般而言，當應用程式遇到錯誤時，它會執行大量的清理工作。接著會通知發生錯誤的使用者，並提供解決問題的選項。

```
private function drmErrorEventHandler(event:DRMErrorEvent):void
{
    trace(event.toString());
}
```

使用 DRMManager 類別

Flash Player 10.1 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

使用 DRMManager 類別，管理應用程式中的憑證和媒體權限伺服器工作階段。

憑證管理 (僅限 **AIR)**

只要使用者播放保護的內容，執行階段就會取得和快取檢視內容所需的授權。如果應用程式在本機儲存檔案，且授權允許離線播放，使用者就可以在 AIR 應用程式中檢視內容。像這樣的本機離線播放，就算在無法連線至媒體權限伺服器時也能夠成功播放。使用 DRMManager 和 NetStream preloadEmbeddedMetadata() 方法，您可以預先快取憑證。應用程式不需要取得檢視內容所需的授權。例如，您的應用程式可以在使用者仍處於線上時下載媒體檔並取得憑證。

若要預先載入憑證，請使用 NetStream 的 preloadEmbeddedMetadata() 方法來取得 DRMContentData 物件。DRMContentData 物件含有媒體權限伺服器的 URL 和網域，該伺服器可提供授權並說明是否需要驗證使用者。有了這項資訊以後，您便可以呼叫 DRMManager 的 loadVoucher() 方法來擷取並快取憑證。如需有關預先載入憑證的工作流程的詳細資訊，請參閱第 454 頁 「預先載入憑證以供離線播放」。

工作階段管理

您也可以使用 DRMManager 來向媒體權限伺服器驗證使用者的權限，並可以管理持續性的工作階段。

您可以呼叫 DRMManager 的 authenticate() 方法來建立媒體權限伺服器的工作階段。當成功完成驗證以後，DRMManager 就會傳送 DRMAuthenticationCompleteEvent 物件。此物件含有一個工作階段權杖。您可以儲存此權杖來建立未來的工作階段，使用者便不再需要提供他們的帳戶憑證。請將此權杖傳遞至 setAuthenticationToken() 方法，以建立一個通過驗證的新工作階段。(產生權杖之伺服器的設定決定權杖到期時間及其他特質。AIR 應用程式程式碼不應解譯權杖資料結構，因為未來的 AIR 更新可能會變更結構)。

您可以將驗證權杖傳輸給其它電腦。若要保護權杖，您可以將它們儲存在 AIR 加密本機儲存區中。請參閱第 612 頁 「加密本機儲存」以取得詳細資訊。

DRMStatus 事件

Flash Player 10.1 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

順利呼叫 loadVoucher() 方法之後，DRMManager 會傳送 DRMStatusEvent 物件。

如果已取得憑證，則事件物件之 detail 屬性 (僅限 AIR) 的值會是 "DRM.voucherObtained"，而且 voucher 屬性會包含 DRMVoucher 物件。

如果未取得憑證，則 detail 屬性 (僅限 AIR) 的值還是 "DRM.voucherObtained"；不過，voucher 屬性會是 null。例如，如果您使用 localOnly 的 LoadVoucherSetting，而且沒有本機快取的憑證，則無法取得憑證。

如果 loadVoucher() 呼叫未順利完成 (也許是因為發生驗證或通訊錯誤)，則 DRMManager 會改為傳送 DRMErrorEvent 或 DRMAuthenticationErrorEvent 物件。

DRMAuthenticationComplete 事件

Flash Player 10.1 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

當透過呼叫 authenticate() 方法而成功驗證使用者時，DRMManager 就會傳送 DRMAuthenticationCompleteEvent 物件。

DRMAuthenticationCompleteEvent 物件含有一個可重複使用的權杖，該權杖可讓使用者驗證的效力在所有應用程式工作階段之間持續存在。將此權杖傳遞給 DRMManager setAuthenticationToken() 方法，以重新建立工作階段 (權杖特質 (例如到期時間) 是由權杖建立者所設定。Adobe 不提供檢查權杖特質的 API)。

DRMAuthenticationError 事件

Flash Player 10.1 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

當透過呼叫 authenticate() 或 setAuthenticationToken() 方法並無法成功驗證使用者時，DRMManager 就會傳送 DRMAuthenticationErrorEvent 物件。

使用 DRMContentData 類別

Flash Player 10.1 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

DRMContentData 物件包含受 Adobe Access 保護之內容的中繼資料屬性。DRMContentData 屬性含有必要的資訊，可取得檢視內容所需的授權憑證。您可以使用 DRMContentData 類別取得與內容相關聯的中繼資料檔案 (如第 451 頁 「詳細的 API 工作流程」所述)。

如需詳細資訊，請參閱 「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考」中的 DRMContentData 類別。

更新 Flash Player 以支援 Adobe Access

Flash Player 10.1 和更新的版本

重要：Flash Player 11.5 及更新版本會整合 Adobe Access 模組，因此不需要執行更新步驟 (呼叫 SystemUpdater.update(SystemUpdaterType.DRM))。這包括下列瀏覽器和平台：

- Flash Player 11.5 ActiveX 控制項，適用於所有平台，但 Windows 8 上的 Internet Explorer 除外
- Flash Player 11.5 外掛程式，適用於所有瀏覽器
- Adobe AIR (桌上型電腦及行動裝置)

這表示，在下列情況中 「還是需要」更新步驟：

- Windows 8 上的 Internet Explorer
- Flash Player 11.4 及以前版本，但 Google Chrome 22 及更新版本 (所有平台) 或 21 及更新版本 (Windows) 除外

備註：在具有 Flash Player 11.5 或更新版本的系統上，您還是可以安全地呼叫 SystemUpdater.update(SystemUpdaterType.DRM)，但不會下載任何項目。

若要支援 Adobe Access，Flash Player 需要 Adobe Access 模組。Flash Player 嘗試播放保護的內容時，執行階段會指出是否必須下載模組或新版本的 Flash Player。因此，Flash Player 允許 SWF 開發人員依需要選擇不要進行更新。

在大部分情況下，若要播放保護的內容，SWF 開發人員會更新至必要的 Adobe Access 模組或播放程式。若要更新，您可以使用 SystemUpdater API 取得最新版本的 Adobe Access 模組或 Flash Player。

SystemUpdater API 一次只允許一個更新。錯誤碼 2202 指出目前的執行階段實體或另一個實體正在進行更新。例如，如果 Internet Explorer 的 Flash Player 實體正在進行更新，則 Firefox 中執行的 Flash Player 實體無法進行更新。

只有桌面平台才支援 SystemUpdater API。

備註：如果是 10.1 之前的 Flash Player 版本，請使用舊版播放程式中支援的更新機制 (從 www.adobe.com 或 ExpressInstall 手動下載並安裝)。此外，AIR 安裝程式會處理 Adobe Access 的必要更新，但不支援 SystemUpdater API。

更新事件的偵聽

Flash Player 10.1 和更新的版本

需要更新 Adobe Access 模組時，NetStream 物件會傳送代碼值為 DRM.UpdateNeeded 的 NetStatusEvent。這個值指出 NetStream 物件無法使用任何目前安裝的 Adobe Access 模組，來播放保護的串流。偵聽這個事件，並呼叫下列程式碼：

```
SystemUpdater.update(flash.system.SystemUpdaterType.DRM)
```

這個程式碼會更新播放程式中安裝的 Adobe Access 模組。不需要使用者同意就可以更新這個模組。

如果找不到 Adobe Access 模組，則會擲回錯誤。請參閱第 451 頁 「詳細的 API 工作流程」的步驟 3。

備註：如果在 10.1 版本之前的播放程式加密串流上呼叫 play()，則會傳送代碼值為 NetStream.Play.StreamNotFound 的 NetStatusEvent。在舊版播放程式中，使用那些播放程式支援的更新機制 (從 www.adobe.com 或 ExpressInstall 手動下載並安裝)。

需要更新播放程式本身時，SystemUpdater 物件會傳送代碼值為 DRM.UpdateNeededButIncompatible 的 StatusEvent。若要更新播放程式，則需要使用者同意。在您的應用程式中提供介面，讓使用者同意並起始播放程式的更新。偵聽 StatusEvent 事件，並呼叫下列程式碼：

```
SystemUpdater.update(flash.system.SystemUpdaterType.SYSTEM);
```

這個程式碼會起始播放程式的更新。

SystemUpdater 類別的其他事件記載於適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中。

播放程式更新完成之後，會將使用者重新導向至更新的開始頁面。下載 Adobe Access 模組，而且可以開始播放串流。

頻外授權

Flash Player 11 以及更新的版本，**Adobe AIR 3.0** 以及更新的版本

透過使用 storeVoucher 方法在磁碟和在記憶體儲存憑證 (授權)，無需聯絡 Adobe Access 授權伺服器，也可以頻外取得授權。

若要在 Flash Player 與 AIR 播放加密的視訊，對應的執行階段必須取得該視訊的 DRM 憑證。DRM 憑證包含視訊的解密金鑰且是由使用者部署的 Adobe Access 授權伺服器產生。

Flash Player/AIR 執行階段取得此憑證的方法是，將憑證要求傳送至視訊的 DRM 中繼資料 (DRMContentData 類別) 中指示的 Adobe Access 授權伺服器。Flash/AIR 應用程式可透過呼叫 DRMManager.loadVoucher() 方法來觸發此授權要求。或者，如果在磁碟或記憶體都沒有該內容的授權時，Flash Player/AIR 執行階段就會在開始播放加密的視訊時，自動要求憑證。任一種情況都會與 Adobe Access 授權伺服器通訊，而使 Flash/AIR 應用程式的效能受到影響。

DRMManager.storeVoucher() 允許 Flash/AIR 應用程式將頻外取得的 DRM 憑證傳送至 Flash Player/AIR 執行階段。執行階段便可略過授權要求程序，並使用轉送的憑證來播放加密的視訊。DRM 憑證仍必須先由 Adobe Access 授權伺服器產生後，才能頻外加以取得。不過，您可以選擇將憑證裝載在任何 HTTP 伺服器，而不是公開的 Adobe Access 授權伺服器。

DRMManager.storeVoucher() 也可以用來在多部裝置之間支援 DRM 憑證共享功能。在 Adobe Access 3.0 中，這是指 「網域支援」功能。如果您的部署環境支援此使用案例，您可以使用 DRMManager.addToDeviceGroup() 方法向裝置群組註冊多部電腦。如果某部電腦上有指定內容與網域繫結的有效憑證，AIR 應用程式便可使用 DRMVoucher.toByteArray() 方法來擷取序列化的 DRM 憑證，而在其它電腦則可以使用 DRMManager.storeVoucher() 方法來匯入憑證。

裝置註冊

DRM 憑證是繫結至使用者的電腦。因此，Flash /AIR 應用程式需要使用者電腦的唯一 ID，以參考正確的序列化 DRM 憑證物件。下列案例說明裝置註冊程序：

假設您已執行下列工作：

- 您已設定 Adobe Access 伺服器 SDK。
- 您已設定 HTTP 伺服器，用於取得預先產生的授權。
- 您已建立一個要檢視受保護內容的 Flash 應用程式。

裝置註冊階段包含下列動作：

1. Flash 應用程式建立隨機產生的 ID。
2. Flash 應用程式叫用 DRMManager.authenticate() 方法。應用程式必須在驗證要求中納入隨機產生的 ID。例如，在使用者名稱欄位納入 ID。
3. 步驟 2 所提及的動作會導致 Adobe Access 將驗證要求傳送給客戶的伺服器。此要求包含裝置憑證。
   a. 伺服器會從要求中擷取裝置憑證與產生的 ID 並加以儲存。
   b. 客戶子系統為此裝置憑證預先產生授權，加以儲存並透過將其與產生的 ID 建立關聯來授與存取權。
4. 伺服器以一則 「成功」訊息回應該要求。
5. Flash 應用程式在本機的本機共享物件 (LSO) 儲存產生的 ID。

在裝置註冊之後，Flash 應用程式會以在先前的配置使用裝置 ID 的相同方式，來使用產生的 ID。

1. Flash 應用程式將嘗試在 LSO 尋找產生的 ID。
2. 如果找到產生的 ID，Flash 應用程式便會在下載預先產生的授權時使用產生的 ID。Flash 應用程式會使用 DRMManager.storeVoucher() 方法，將授權傳送至 Adobe Access 用戶端以供耗用。
3. 如果找不到產生的 ID，Flash 應用程式將會進行裝置註冊程序。

原廠預設值

當裝置的使用者叫用原廠預設值選項時，便會清除裝置憑證。若要繼續播放保護的內容，Flash 必須再次進行裝置註冊程序。如果 Flash 應用程式傳送過時的預先產生授權，因為該加密授權僅供舊的裝置 ID 使用，所以 Adobe Access 用戶端會加以拒絕。

網域支援

Flash Player 11 以及更新的版本，**Adobe AIR 3.0** 以及更新的版本

如果內容中繼資料指定需要網域註冊，AIR 應用程式便會叫用 API 以加入裝置群組。此動作可觸發將網域註冊要求傳送至網域伺服器。在向裝置群組權核發授權後，便可將該授權匯出並與其它加入裝置群組的裝置共享。

然後，可在 DRMContentData 的 VoucherAccessInfo 物件使用裝置群組資訊，該物件是用來提供必要資訊以成功擷取和耗用憑證。

使用網域支援功能播放加密的內容

若要使用 Adobe Access 播放加密的內容，請執行下列步驟：

1 使用 VoucherAccessInfo.deviceGroup 以檢查是否需要註冊裝置群組。

2 是否需要驗證：

   a 使用 DeviceGroupInfo.authenticationMethod 屬性以檢查是否需要驗證。

   b 如果需要驗證，請透過執行下列其中一個步驟來驗證使用者：

      - 取得使用者的使用者名稱與密碼。叫用 DRMManager.authenticate(deviceGroup.serverURL, deviceGroup.domain, username, password)。

      - 取得快取 / 預先產生的驗證權杖並叫用 DRMManager.setAuthenticationToken()。

   c 叫用 DRMManager.addToDeviceGroup()。

3 透過執行下列其中一項工作來取得內容的憑證：

   a 使用 DRMManager.loadVoucher() 方法。

   b 從在相同裝置群組註冊的不同裝置中取得憑證。透過 DRMManager.storeVoucher() 方法，將憑證提供給 DRMManager。

4 使用 NetStream.play() 方法播放加密的內容。

從 Adobe Access 授權伺服器取得授權之後，任何裝置都能使用 DRMVoucher.toByteArray() 方法提供授權的原始位元組，來匯出內容的授權。內容提供者通常會限制裝置群組中的裝置數目。如果達到限制，在註冊目前的裝置之前，您必須在未使用的裝置上呼叫 DRMManager.removeFromDeviceGroup() 方法。

授權預覽

Flash 應用程式可傳送授權預覽要求，其表示在要求使用者購買內容，以決定使用者的電腦實際上是否符合所有必要的播放準則之前，應用程式能夠執行預覽作業。授權預覽是指用戶端預覽授權 (查看該授權允許哪些權限) 的功能，與預覽內容 (在決定購買之前，先檢視小部分內容) 不同。對於每部電腦而言是唯一的一些參數：可用的輸出與其保護狀態、可用的執行階段 /DRM 版本以及 DRM 用戶端安全性層級。授權預覽模式允許執行階段 /DRM 用戶端測試授權伺服器商業邏輯，以及向使用者提供資訊，以便做出旁徵博引的決定。因此用戶端可知有效授權看起來如何，但實際上不會收到可將內容解密的金鑰。您可以選擇性支援授權預覽，而且只有當您實作使用此功能的自訂應用程式才有必要支援。

傳遞內容

因為 Flash Player 將網路層精簡，而只是單純將保護的內容提供給 Adobe Access 子系統，使得 Adobe Access 無從驗證內容的傳遞機制。因此，您可以透過 HTTP、HTTP 動態串流、RTMP 或 RTMPE 傳遞內容。

不過，因為在 Adobe Access 能夠擷取授權來解密內容之前，受保護內容的中繼資料不可或缺 (通常是以 '.metadata' 檔案的形式)，所以可能會發生一些問題。特別是在使用 RTMP/RTMPE 通訊協定時，只有 FLV 與 F4V 資料能夠透過 Flash Media Server (FMS) 傳遞至用戶端。因此，用戶端必須利用其它方式來擷取中繼資料 Blob。解決此問題的一項方案是在 HTTP 伺服器裝載中繼資料，然後實作用戶端視訊播放程式，根據所播放的內容來擷取適當的中繼資料。

```
private function getMetadata():void{

    extrapolated-path-to-metadata = "http://metadatas.mywebserver.com/" + videoname;
    var urlRequest : URLRequest = new URLRequest(extrapolated-path-to-the-metadata + ".metadata");
    var urlStream : URLStream = new URLStream();
    urlStream.addEventListener(Event.COMPLETE, handleMetadata);
    urlStream.addEventListener(IOErrorEvent.NETWORK_ERROR, handleIOError);
    urlStream.addEventListener(IOErrorEvent.IO_ERROR, handleIOError);
    urlStream.addEventListener(IOErrorEvent.VERIFY_ERROR, handleIOError);
    try{
        urlStream.load(urlRequest);
     }catch(se:SecurityError){
        videoLog.text += se.toString() + "\n";
     }catch(e:Error){
        videoLog.text += e.toString() + "\n";
     }
}
```

開放原始碼媒體架構

開放原始碼媒體架構 (OSMF) 是 ActionScript 類型的架構，讓您在建立自己的多媒體時能有極大彈性並完全掌控。如需有關 OSMF 的詳細資訊，請造訪 OSMF 開發人員網站。

播放受保護內容的工作流程

1 建立 MediaPlayer 實體。

```
player = new MediaPlayer();
```

2 向播放程式註冊 MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE 事件。如果內容受 DRM 保護，便會傳送此事件。

```
player.addEventListener(MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE, onDRMCapabilityChange);
```

3 在事件處理常式中，取得 DRMTrait 實體。您可以透過 DRMTrait 介面來叫用 DRM 相關方法，例如 authenticate()。載入 DRM 保護的內容時，OSMF 會執行 DRM 驗證動作並傳送狀態事件。將 DRMEvent.DRM_STATE_CHANGE 事件處理常式新增至 DRMTrait。

```
private function onDRMCapabilityChange
    (event :MediaPlayerCapabilityChangeEvent) :void
        {
            if (event.type == MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE
                && event.enabled)
            {
                drmTrait = player.media.getTrait(MediaTraitType.DRM) as DRMTrait;
                drmTrait.addEventListener
                    (DRMEvent.DRM_STATE_CHANGE, onDRMStateChange);
            }
        }
```

4 以 onDRMStateChange() 方法處理 DRM 事件。

```
private function onDRMStateChange(event :DRMEvent) :void
{
    trace ( "DRMState: ",event.drmState);
    switch(event.drmState)
    {
        case DRMState.AUTHENTICATION_NEEDED:
            // Identity-based content
            var authPopup :AuthWindow = AuthWindow.create(_parentWin);
            authPopup.serverURL = event.serverURL;
            authPopup.addEventListener("dismiss", function () :void {
                trace ("Authentication dismissed");
                if(_drmTrait != null)
                {
                    //Ignore authentication. Just
                    //try to acquire a license.
                    _drmTrait.authenticate(null, null);
                }
            });
            authPopup.addEventListener("authenticate",
                            function (event :AuthWindowEvent) :void {
                if(_drmTrait != null)
                {
                    _drmTrait.authenticate(event.username, event.password);
                }
            });
            authPopup.show();
            break;
        case DRMState.AUTHENTICATING:
            //Display any authentication message.
            trace("Authenticating...");
            break;
        case DRMState.AUTHENTICATION_COMPLETE:
            // Start to retrieve voucher and playback.
            // You can display the voucher information at this point.
            if(event.token)
            // You just received the authentication token.
            {
                trace("Authentication success. Token: \n", event.token);
            }
            else
            // You have got the voucher.
            {
                trace("DRM License:");
                trace("Playback window period: ",
                    !isNaN(event.period) ? event.period == 0 ?
                    "<unlimited>" : event.period : "<none>");
```

```
                trace("Playback window end date: ",
                    event.endDate != null ? event.endDate : "<none>");
                trace("Playback window start date: ",
                    event.startDate != null ? event.startDate : "<none>");
            }
            break;
        case DRMState.AUTHENTICATION_ERROR:
            trace ("DRM Error:", event.mediaError.errorID +
                "[" + DRMErrorEventRef.getDRMErrorMnemonic
                (event.mediaError.errorID) + "]");
            //Stop everything.
            player.media = null;
            break;
        case DRMState.DRM_SYSTEM_UPDATING:
            Logger.log("Downloading DRM module...");
            break;
        case DRMState.UNINITIALIZED:
            break;
    }
}
```

第 28 章 在 AIR 中新增 PDF 內容

Adobe AIR 1.0 以及更新的版本

在 Adobe® AIR® 中執行的應用程式不僅可以顯示 SWF 和 HTML 內容，還可以顯示 PDF 內容。AIR 應用程式會使用 HTMLLoader 類別、WebKit 引擎和 Adobe® Reader® 瀏覽器外掛程式來顯示 PDF 內容。在 AIR 應用程式中，PDF 內容可以延伸超過應用程式的完整高度和寬度，或者做為介面的一部分。Adobe Reader 瀏覽器外掛程式會控制 AIR 應用程式的 PDF 檔案顯示方式。對 Reader 工具列介面所做的修改 (例如位置、錨點和可見性的控制項)，會一直存在於後續在 AIR 應用程式和瀏覽器中檢視的 PDF 檔案。

重要事項：若要在 AIR 中顯示 PDF 內容，使用者必須安裝 Adobe Reader 或 Adobe® Acrobat® 8.1 或更新版本。

偵測 PDF 功能

Adobe AIR 1.0 以及更新的版本

如果使用者沒有 Adobe Reader 或 Adobe Acrobat 8.1 或更新版本，就無法在 AIR 應用程式中顯示 PDF 內容。若要偵測使用者是否能顯示 PDF 內容，請先查看 HTMLLoader.pdfCapability 屬性。這個屬性必須設定為 HTMLPDFCapability 類別的下列其中一個常數：

常數	說明
HTMLPDFCapability.STATUS_OK	偵測到所需的 Adobe Reader 版本 (8.1 或更新版本)，而且可將 PDF 內容載入至 HTMLLoader 物件。
HTMLPDFCapability.ERROR_INSTALLED_READER_NOT_FOUND	未偵測到任何 Adobe Reader 版本。HTMLLoader 物件無法顯示 PDF 內容。
HTMLPDFCapability.ERROR_INSTALLED_READER_TOO_OLD	偵測到 Adobe Reader，但版本太舊。HTMLLoader 物件無法顯示 PDF 內容。
HTMLPDFCapability.ERROR_PREFERRED_READER_TOO_OLD	偵測到所需的 Adobe Reader 版本 (8.1 或更新版)，但負責處理 PDF 內容的 Adobe Reader 版本比 Reader 8.1 還舊。HTMLLoader 物件無法顯示 PDF 內容。

在 Windows 中，如果在使用者系統上執行的是 Adobe Acrobat 或 Adobe Reader 7.x 或更新版本，則即使安裝可支援載入 PDF 的更新版本，還是會使用舊版本。在這種情況下，如果 pdfCapability 屬性的值為 HTMLPDFCapability.STATUS_OK，則當 AIR 應用程式嘗試載入 PDF 內容時，該舊版 Acrobat 或 Reader 便會顯示警告 (而且不會在 AIR 應用程式中擲回例外)。如果這是使用者可能會發生的狀況，您可以考慮提供他們指示，在執行應用程式時關閉 Acrobat。如果 PDF 內容未在可接受的時間範圍內載入，您可以顯示這些指示。

在 Linux 中，AIR 會在使用者匯出的 PATH (如果其中含有 acroread 命令) 以及 /opt/Adobe/Reader 目錄中尋找 Adobe Reader。

以下程式碼會偵測使用者是否可以在 AIR 應用程式中顯示 PDF 內容。如果使用者無法顯示 PDF，程式碼就會追蹤與 HTMLPDFCapability 錯誤物件相對應的錯誤碼：

```
if(HTMLLoader.pdfCapability == HTMLPDFCapability.STATUS_OK)
{
    trace("PDF content can be displayed");
}
else
{
    trace("PDF cannot be displayed. Error code:", HTMLLoader.pdfCapability);
}
```

載入 PDF 內容

Adobe AIR 1.0 以及更新的版本

您可以建立 HTMLLoader 實體、設定其尺寸以及載入 PDF 的路徑，以將該 PDF 加入至 AIR 應用程式。

下列範例會從外部網站載入 PDF。請以可用之外部 PDF 的路徑取代 URLRequest。

```
var request:URLRequest = new URLRequest("http://www.example.com/test.pdf");
pdf = new HTMLLoader();
pdf.height = 800;
pdf.width = 600;
pdf.load(request);
container.addChild(pdf);
```

您也可以從檔案 URL 和 AIR 專屬 URL 配置 (例如 app 和 app-storage) 載入內容。例如，下列程式碼會載入應用程式目錄之 PDFs 子目錄中的 test.pdf 檔案：

app:/js_api_reference.pdf

如需有關 AIR URL 配置的詳細資訊，請參閱 第 702 頁 「URI 配置」。

為 PDF 內容編寫指令碼

Adobe AIR 1.0 以及更新的版本

您可以使用 JavaScript 控制 PDF 內容，就像在瀏覽器的網頁中控制 PDF 內容一樣。

適用於 Acrobat 的 JavaScript 擴充功能可以提供許多功能，包括：

- 控制網頁瀏覽和顯示比例
- 處理文件中的表單
- 控制多媒體事件

如需有關適用於 Adobe Acrobat 之 JavaScript 擴充功能的詳細資訊，請造訪 Adobe Acrobat Developer Connection，網址為：http://www.adobe.com/devnet/acrobat/javascript.html。

HTML 與 PDF 之間通訊的基本概念

Adobe AIR 1.0 以及更新的版本

HTML 網頁中的 JavaScript 可以呼叫表示 PDF 內容之 DOM 物件的 postMessage() 方法，將訊息傳送給 PDF 內容中的 JavaScript。以下列內嵌式 PDF 內容為例：

```
<object id="PDFObj" data="test.pdf" type="application/pdf" width="100%" height="100%"/>
```

下面這個在內含之 HTML 內容中的 JavaScript 程式碼會將訊息傳送給 PDF 檔的 JavaScript：

```
pdfObject = document.getElementById("PDFObj");
pdfObject.postMessage(["testMsg", "hello"]);
```

PDF 檔可以包含 JavaScript 以便接收這個訊息。在某些內容 (其中包括文件、資料夾、網頁、欄位和批次層級) 中，您可以將 JavaScript 程式碼加入至 PDF 檔。在此僅會針對文件層級內容來討論，這個層級的內容會定義當開啟 PDF 文件時，所評估的預設指令碼。

PDF 檔可以將 messageHandler 屬性加入至 hostContainer 物件。messageHandler 屬性為一個物件，會定義預設處理常式函數以回應訊息。例如，下列範例會定義函數，以處理 PDF 檔從裝載容器 (即內嵌在 PDF 檔的 HTML 內容) 收到的訊息。

```
this.hostContainer.messageHandler = {onMessage: myOnMessage};

function myOnMessage(aMessage)
{
    if(aMessage[0] == "testMsg")
    {
        app.alert("Test message: " + aMessage[1]);
    }
    else
    {
        app.alert("Error");
    }
}
```

HTML 網頁中的 JavaScript 程式碼可以呼叫包含在網頁中之 PDF 物件的 postMessage() 方法。呼叫這個方法時，便會將訊息 ("Hello from HTML") 傳送給 PDF 檔中的文件層級 JavaScript。

```
<html>
    <head>
    <title>PDF Test</title>
    <script>
        function init()
        {
            pdfObject = document.getElementById("PDFObj");
            try {
                 pdfObject.postMessage(["alert", "Hello from HTML"]);
            }
            catch (e)
            {
                alert( "Error: \n name = " + e.name + "\n message = " + e.message );
            }
        }
    </script>
    </head>
    <body onload='init()'>
        <object
            id="PDFObj"
            data="test.pdf"
            type="application/pdf"
            width="100%" height="100%"/>
    </body>
</html>
```

如需進階範例，以及使用 Acrobat 8 在 PDF 檔案中新增 JavaScript 的詳細資訊，請參閱在 Adobe AIR 中對 PDF 內容進行跨指令碼處理。

透過 ActionScript 為 PDF 內容編寫指令碼

Adobe AIR 1.0 以及更新的版本

ActionScript 程式碼 (在 SWF 內容中) 無法直接與 PDF 內容中的 JavaScript 進行通訊。不過，對於載入 PDF 內容之 HTMLLoader 物件中載入的 HTML 網頁，ActionScript 可以與網頁中的 JavaScript 進行通訊，而且該 JavaScript 程式碼可以與所載入 PDF 檔中的 JavaScript 進行通訊。如需詳細資訊，請參閱第 845 頁「在 AIR 中撰寫 HTML 和 JavaScript 程式」。

AIR 中 PDF 內容的已知限制

Adobe AIR 1.0 以及更新的版本

Adobe AIR 中的 PDF 內容具有下列限制：

- PDF 不會在透明 (transparent 屬性設定為 true) 的視窗 (NativeWindow 物件) 中顯示。
- PDF 檔的顯示順序在運作時，與 AIR 應用程式中的其它顯示物件不同。雖然 PDF 內容會根據 HTML 顯示順序正確進行修剪，不過一律都在 AIR 應用程式的顯示順序中的內容上方。
- 對於含有 PDF 文件的 HTMLLoader 物件，物件中的某些視覺屬性如果發生變更，該 PDF 文件就會變為隱藏。這些屬性包括 filters、alpha、rotation 和 scaling 屬性。變更這些屬性會隱藏 PDF 內容，直到屬性重設為止。如果您變更包含 HTMLLoader 物件之顯示物件容器的上述屬性，也會隱藏 PDF 內容。
- 對於含有 PDF 內容的 NativeWindow 物件，只有將其 Stage 物件的 scaleMode 屬性設為 StageScaleMode.NO_SCALE，PDF 內容才會是可見的。若設定為其它的值，就會隱藏 PDF 內容。
- 如果按一下 PDF 檔中內容的連結，便會更新 PDF 內容的捲動位置。如果按一下 PDF 檔以外的內容，便會重新導向包含 PDF 的 HTMLLoader 物件 (即使連結的目標是新視窗也一樣)。
- PDF 註解工作流程在 AIR 中無法運作。

第 29 章 使用者互動的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您的應用程式可使用 ActionScript 3.0 來回應使用者活動，以便建立互動性。請注意，本節假定您已經熟悉 ActionScript 3.0 事件模型的使用。如需詳細資訊，請參閱第 108 頁 「處理事件」。

擷取使用者輸入

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

不管是透過鍵盤、滑鼠、相機或是這些裝置的組合，使用者互動都是一切互動的基礎。在 ActionScript 3.0 中識別並回應使用者互動主要還是與偵聽事件有關。

InteractiveObject 類別 (DisplayObject 類別的子類別) 可提供處理使用者互動時所需的一般事件與功能結構。您無須直接建立 InteractiveObject 類別實體。相反地，諸如 SimpleButton、Sprite、TextField 的顯示物件，以及各種 Flash 編寫工具與 Flex 組件都會從此類別繼承其使用者互動模式，進而共用其結構。也就是說，您所學習到的技巧，以及針對來自 InteractiveObject 之物件撰寫來處理其使用者互動的程式碼，都同樣適用其它所有物件。

重要概念與術語
在進行之前，務必先熟悉下列重要的使用者互動術語：

字元碼 代表目前字元集中某個字元的數值碼 (與鍵盤上按下的某個按鍵相關)。例如，儘管 「D」和 「d」都是由美式英文鍵盤上相同按鍵所建立，卻各具有不同的字元碼。

快顯選單 當使用者按一下滑鼠右鍵，或是使用特定的鍵盤 / 滑鼠組合時，所出現的選單。一般來說，快顯選單命令會直接套用到所按下的按鍵。例如，某個影像的快顯選單可能包含一個可在個別視窗顯示影像以及一個下載該影像的命令。

焦點 表示選取的元素目前為作用中，而且成為鍵盤或滑鼠互動的目標。

按鍵碼 對應至鍵盤上實際按鍵的數值碼。

更多說明主題
InteractiveObject

Keyboard

Mouse

ContextMenu

管理焦點

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

互動物件可以透過程式設計或是使用者動作來接收焦點。此外，如果將 tabEnabled 屬性設為 true，當使用者按下 Tab 鍵後，焦點會從舊的物件轉移到新的物件上。請注意，除了下列情況以外，tabEnabled 的值預設都是 false：

- 若是 SimpleButton 物件，這個值會是 true。

- 若是輸入文字欄位，這個值會是 true。
- 若是 buttonMode 設定為 true 的 Sprite 物件或 MovieClip 物件，這個值會是 true。

針對上述每一種情況，您都可以為 FocusEvent.FOCUS_IN 或 FocusEvent.FOCUS_OUT 加入偵聽程式，以提供焦點變更時的其它行為。這種方式特別適用於文字欄位和表單，但也可以用於本身是 InteractiveObject 類別之子類別的 Sprite、影片片段或任何物件。下列範例將說明如何透過 Tab 鍵來啟用焦點循環，以及如何回應後續的焦點事件。在此情況中，每個正方形都會隨著所收到的焦點而變更顏色。

備註：Flash Professional 運用了鍵盤快速鍵來管理焦點；因此，為了適當地模擬焦點管理，您應該在瀏覽器或 AIR 中 (不要使用 Flash) 測試 SWF 檔。

```
var rows:uint = 10;
var cols:uint = 10;
var rowSpacing:uint = 25;
var colSpacing:uint = 25;
var i:uint;
var j:uint;
for (i = 0; i < rows; i++)
{
    for (j = 0; j < cols; j++)
    {
        createSquare(j * colSpacing, i * rowSpacing, (i * cols) + j);
    }
}

function createSquare(startX:Number, startY:Number, tabNumber:uint):void
{
    var square:Sprite = new Sprite();
    square.graphics.beginFill(0x000000);
    square.graphics.drawRect(0, 0, colSpacing, rowSpacing);
    square.graphics.endFill();
    square.x = startX;
    square.y = startY;
    square.tabEnabled = true;
    square.tabIndex = tabNumber;
    square.addEventListener(FocusEvent.FOCUS_IN, changeColor);
    addChild(square);
}
function changeColor(event:FocusEvent):void
{
    event.target.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

探索輸入類型

Flash Player 10.1 以及更新的版本，**Adobe AIR 2** 以及更新的版本

Flash Player 10.1 和 Adobe AIR 2 版本已引進功能，測試執行階段環境是否支援特定輸入類型。您可以使用 ActionScript 測試裝置上目前部署的執行階段是否具有下列功能：

- 支援觸控筆或手指輸入 (或根本沒有觸控輸入)。
- 具有虛擬或實體鍵盤供使用者使用 (或根本沒有鍵盤)。
- 顯示游標 (若否，則根據游標是否停留在物件的功能不會作用)。

輸入探索 ActionScript API 包括：

- flash.system.Capabilities.touchscreenType 屬性：執行階段提供的值，指出目前環境支援的輸入類型。
- flash.system.TouchscreenType 類別：Capabilities.touchscreenType 屬性之列舉值常數的類別。
- flash.ui.Mouse.supportsCursor 屬性：執行階段提供的值，指出永久性游標是否可用。
- flash.ui.Keyboard.physicalKeyboardType 屬性：執行階段提供的值，指出可以使用完整實體鍵盤、只能使用數字鍵台，或根本沒有鍵盤。
- flash.ui.KeyboardType 類別：flash.ui.Keyboard.physicalKeyboardType 屬性之列舉值常數的類別。
- flash.ui.Keyboard.hasVirtualKeyboard 屬性：執行階段提供的值，指出是否提供虛擬鍵盤讓使用者使用 (取代實體鍵盤，或是也有實體鍵盤)。

輸入探索 API 可讓您利用使用者的裝置功能，或在那些功能未出現時提供替代方式。這些 API 特別適用於開發行動應用程式和已啟用觸控的應用程式。例如，如果您的介面是針對具有觸控筆之小型按鈕的行動裝置，則可以為使用手指觸控進行輸入的使用者提供具有大型按鈕的替代介面。下列程式碼是針對具有 createStylusUI() 函數的應用程式，而此函數指定適用於進行觸控筆互動的一組使用者介面元素。另一個函數 createTouchUI() 指定另一組使用者介面元素，適用於進行手指互動：

```
if(Capabilities.touchscreenType == TouchscreenType.STYLUS ){
    //Construct the user interface using small buttons for a stylus
    //and allow more screen space for other visual content
    createStylusUI();
} else if(Capabilities.touchscreenType = TouchscreenType.FINGER){
    //Construct the user interface using larger buttons
    //to capture a larger point of contact with the device
    createTouchUI();
}
```

開發不同輸入環境的應用程式時，請考慮下列相容性圖表：

環境	supportsCursor	touchscreenType == FINGER	touchscreenType == STYLUS	touchscreenType == NONE
傳統桌面	true	False	False	true

環境	supportsCursor	touchscreenType == FINGER	touchscreenType == STYLUS	touchscreenType == NONE
電容式觸控螢幕裝置 (可偵測細微人體觸控的數位板、PDA 和手機，例如 Apple iPhone 或 Palm Pre)	False	true	False	False
電阻式觸控螢幕裝置 (可偵測精確高壓接觸的數位板、PDA 和手機，例如 HTC Fuze)	False	False	true	False
非觸控螢幕裝置 (可執行應用程式但沒有偵測接觸之螢幕的功能手機和裝置)	False	False	False	true

備註： 不同的裝置平台可以支援多種輸入類型的組合。此圖表可當成一般指南使用。

第 30 章 鍵盤輸入

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您的應用程式可擷取與反應鍵盤輸入，並操作 IME，讓使用者以多位元組的語言輸入非 ASCII 文字字元。請注意，本節假定您已經熟悉 ActionScript 3.0 事件模型的使用。如需詳細資訊，請參閱第 108 頁 「處理事件」。

如需探索執行階段期間可用鍵盤支援種類 (例如實體、虛擬、英數字母或 12 個按鈕數值) 的詳細資訊，請參閱第 477 頁 「探索輸入類型」。

輸入法編輯器 (IME) 可讓使用者透過標準鍵盤輸入複雜字元和符號。您可以使用 IME 類別，讓使用者在您的應用程式中使用他們的系統 IME。

更多說明主題

flash.events.KeyboardEvent

flash.system.IME

擷取鍵盤輸入

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

顯示物件如果從 InteractiveObject 類別繼承了物件的互動模式，則可以透過事件偵聽程式回應鍵盤事件。例如，您可以將事件偵聽程式放置在 「舞台」上，以偵聽並回應鍵盤輸入。在下列程式碼中，事件偵聽程式會擷取按鍵按下動作，並顯示按鍵名稱與按鍵碼屬性：

```
function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (character code: " + event.charCode +
")");
}
stage.addEventListener(KeyboardEvent.KEY_DOWN, reportKeyDown);
```

某些按鍵 (例如 Ctrl 鍵) 就算按下之後不會出現文字，也會產生事件。

在上一個程式碼範例中，鍵盤事件偵聽程式擷取了整個 「舞台」的鍵盤輸入。您也可以針對 「舞台」上的特定顯示物件撰寫事件偵聽程式；一旦物件成為焦點就可觸發這個事件偵聽程式。

在下列範例中，只有當使用者在 TextField 實體中輸入時，「輸出」面板才會反映按鍵動作。按住 Shift 鍵則會暫時將 TextField 的邊框顏色變為紅色。

這個程式碼會假定 「舞台」上有名為 tf 的 TextField 實體。

```
tf.border = true;
tf.type = "input";
tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
tf.addEventListener(KeyboardEvent.KEY_UP,reportKeyUp);

function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (key code: " + event.keyCode + "
character code: " + event.charCode + ")");
    if (event.keyCode == Keyboard.SHIFT) tf.borderColor = 0xFF0000;
}

function reportKeyUp(event:KeyboardEvent):void
{
    trace("Key Released: " + String.fromCharCode(event.charCode) + " (key code: " + event.keyCode + "
character code: " + event.charCode + ")");
    if (event.keyCode == Keyboard.SHIFT)
    {
        tf.borderColor = 0x000000;
    }
}
```

這個 TextField 類別也會回報 textInput 事件，您可以在使用者輸入文字時偵聽這個事件。如需詳細資訊，請參閱第 319 頁「擷取文字輸入」。

備註：在 AIR 執行階段，可以取消鍵盤事件。在 Flash Player 執行階段，則無法取消鍵盤事件。

按鍵碼與字元碼

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以存取鍵盤事件的 keyCode 和 charCode 屬性，以判斷所按下的按鍵並隨後觸發其它動作。keyCode 屬性是一個數值，會對應至鍵盤上某個按鍵的值。charCode 屬性是目前字元集中該按鍵的數值 (預設字元集為支援 ASCII 的 UTF-8)。

按鍵碼值與字元碼值之間的主要不同之處在於，按鍵碼值表示鍵盤上的特定按鍵 (數字鍵台上的 1 與鍵盤上面那一整排數字中的 1 不同，但是產生「1」與「!」的按鍵卻是同一個)，而字元值則表示特定字元 (大寫 R 和小寫 r 字元是不一樣的)。

備註：如需按鍵與 ASCII 字元碼值之間的對應關係，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中的 flash.ui.Keyboard 類別。

按鍵與其按鍵碼之間的對應關係取決於裝置與作業系統。因此，請勿透過按鍵對應關係來觸發動作。相反地，請透過 Keyboard 類別所提供之預先定義的常數值，參考適當的 keyCode 屬性。例如，不要使用 Shift 鍵的按鍵對應，而改用 Keyboard.SHIFT 常數 (如上一個程式碼樣本所示)。

KeyboardEvent 優先順序

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如同其它事件一樣，鍵盤事件順序是由顯示物件階層架構所決定，而不是由程式碼中 addEventListener() 方法的指定順序所決定。

例如，假定您將名為 tf 的文字欄位置入名為 container 影片片段中，然後為按鍵事件將事件偵聽程式同時加入這兩個實體，如下列範例所示：

```
container.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
container.tf.border = true;
container.tf.type = "input";
container.tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);

function reportKeyDown(event:KeyboardEvent):void
{
    trace(event.currentTarget.name + " hears key press: " + String.fromCharCode(event.charCode) + " (key
code: " + event.keyCode + " character code: " + event.charCode + ")");
}
```

因為文字欄位及其父容器上都有一個偵聽程式，所以每次在 TextField 中按下按鍵時，就會呼叫兩次 reportKeyDown() 函數。請注意，每次按下一個按鍵，文字欄位就會在 container 影片片段傳送事件之前，先傳送一個事件。

作業系統以及網頁瀏覽器會搶先在 Adobe Flash Player 或 AIR 之前處理鍵盤事件。例如，使用 Microsoft Internet Explorer 時，請在任何包含物件的 SWF 檔傳送鍵盤事件時，先按下 Ctrl+W 鍵，關閉瀏覽器視窗。

使用 IME 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

IME 類別可讓您在 Flash Player 或 Adobe AIR 中操作作業系統的 IME。

您可以使用 ActionScript 判斷下列資訊：

- 使用者的電腦中是否已安裝 IME (Capabilities.hasIME)
- 使用者的電腦目前啟用或停用 IME (IME.enabled)
- 目前 IME 所使用的轉換模式 (IME.conversionMode)

您可以將輸入文字欄位與特定的 IME 內容產生關聯。當您在多個輸入欄位之間進行切換時，也可以將 IME 切換為平假名 (日文)、全形數字、半形數字和直接輸入等等。

IME 可讓使用者以多位元組的語言 (例如中文、日文及韓文) 輸入非 ASCII 的文字字元。

如需有關使用 IME 的詳細資訊，請參閱您為其開發應用程式之作業系統的說明文件。如需其它資源，請參閱下列網站：

- http://www.msdn.microsoft.com/goglobal/
- http://developer.apple.com/library/mac/navigation/
- http://www.java.sun.com/

備註：如果 IME 未在使用者的電腦上啟用，呼叫 Capabilities.hasIME 之外的 IME 方法或屬性便會失敗。您以手動方式啟用 IME 之後，對於 IME 方法和屬性的後續 ActionScript 呼叫都會如預期地發揮作用。例如，假設您使用的是日文 IME，則必須在呼叫任何 IME 方法或屬性之前啟用該 IME。

檢查是否已安裝並啟用 IME

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在您呼叫任何 IME 方法或屬性之前，請務必檢查目前使用者的電腦中是否已安裝並啟用該 IME。下列程式碼說明如何在呼叫任何方法之前，檢查使用者是否已安裝並啟動 IME：

```
if (Capabilities.hasIME)
{
    if (IME.enabled)
    {
        trace("IME is installed and enabled.");
    }
    else
    {
        trace("IME is installed but not enabled. Please enable your IME and try again.");
    }
}
else
{
    trace("IME is not installed. Please install an IME and try again.");
}
```

上述程式碼會先檢查使用者是否已使用 Capabilities.hasIME 屬性，安裝了 IME。如果將此屬性設定為 true，程式碼接著就會使用 IME.enabled 屬性，檢查使用者目前是否已啟用該 IME。

判斷目前所啟用的 IME 轉換模式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

建置多國語言應用程式時，您可能需要判斷使用者目前已啟動的轉換模式。下列程式碼說明如何檢查使用者是否已安裝 IME，以及如果已經安裝，那麼目前啟動的 IME 轉換模式為何：

```
if (Capabilities.hasIME)
{
    switch (IME.conversionMode)
    {
        case IMEConversionMode.ALPHANUMERIC_FULL:
            tf.text = "Current conversion mode is alphanumeric (full-width).";
            break;
        case IMEConversionMode.ALPHANUMERIC_HALF:
            tf.text = "Current conversion mode is alphanumeric (half-width).";
            break;
        case IMEConversionMode.CHINESE:
            tf.text = "Current conversion mode is Chinese.";
            break;
        case IMEConversionMode.JAPANESE_HIRAGANA:
            tf.text = "Current conversion mode is Japananese Hiragana.";
            break;
        case IMEConversionMode.JAPANESE_KATAKANA_FULL:
            tf.text = "Current conversion mode is Japanese Katakana (full-width).";
            break;
        case IMEConversionMode.JAPANESE_KATAKANA_HALF:
            tf.text = "Current conversion mode is Japanese Katakana (half-width).";
            break;
        case IMEConversionMode.KOREAN:
            tf.text = "Current conversion mode is Korean.";
            break;
        default:
            tf.text = "Current conversion mode is " + IME.conversionMode + ".";
            break;
    }
}
else
{
    tf.text = "Please install an IME and try again.";
}
```

上述程式碼會先檢查使用者是否已安裝 IME，然後再藉由比較 IME.conversionMode 屬性與 IMEConversionMode 類別中的每一個常數，檢查目前使用的 IME 轉換模式為何。

設定 IME 轉換模式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您變更使用者的 IME 轉換模式時，必須確保程式碼是位於 try..catch 區塊中，因為如果 IME 無法設定轉換模式時，使用 conversionMode 屬性來設定轉換模式便會擲回錯誤。下列程式碼會示範如何使用 try..catch 區段來設定 IME.conversionMode 屬性：

```
var statusText:TextField = new TextField;
statusText.autoSize = TextFieldAutoSize.LEFT;
addChild(statusText);
if (Capabilities.hasIME)
{
    try
    {
        IME.enabled = true;
        IME.conversionMode = IMEConversionMode.KOREAN;
        statusText.text = "Conversion mode is " + IME.conversionMode + ".";
    }
    catch (error:Error)
    {
        statusText.text = "Unable to set conversion mode.\n" + error.message;
    }
}
```

上述程式碼會先建立文字欄位，以便用來向使用者顯示狀態訊息。接著，如果已安裝 IME，則此程式碼會啟用 IME，並將轉換模式設定為韓文。如果使用者的電腦中沒有安裝韓文 IME，Flash Player 或 AIR 就會擲回錯誤，而且 try..catch 區塊便會捕捉到此錯誤。try..catch 區塊會在先前建立的文字欄位中顯示該錯誤訊息。

停用特定文字欄位的 IME

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在某些情況下，您可能要在使用者輸入字元期間，停用使用者的 IME。例如，假設某個文字欄位只接受數值輸入，您可能就不希望 IME 出現並拖慢資料輸入的速度。

下列範例會示範如何偵聽 FocusEvent.FOCUS_IN 和 FocusEvent.FOCUS_OUT 事件，並加以停用使用者的 IME：

```
var phoneTxt:TextField = new TextField();
var nameTxt:TextField = new TextField();

phoneTxt.type = TextFieldType.INPUT;
phoneTxt.addEventListener(FocusEvent.FOCUS_IN, focusInHandler);
phoneTxt.addEventListener(FocusEvent.FOCUS_OUT, focusOutHandler);
phoneTxt.restrict = "0-9";
phoneTxt.width = 100;
phoneTxt.height = 18;
phoneTxt.background = true;
phoneTxt.border = true;
addChild(phoneTxt);

nameField.type = TextFieldType.INPUT;
nameField.x = 120;
nameField.width = 100;
nameField.height = 18;
nameField.background = true;
nameField.border = true;
addChild(nameField);

function focusInHandler(event:FocusEvent):void
{
    if (Capabilities.hasIME)
    {
        IME.enabled = false;
    }
}
function focusOutHandler(event:FocusEvent):void
{
    if (Capabilities.hasIME)
    {
        IME.enabled = true;
    }
}
```

此範例會先建立 phoneTxt 和 nameTxt 兩個輸入文字欄位，然後再將兩個事件偵聽程式加入至 phoneTxt 文字欄位。當使用者將焦點設定為 phoneTxt 文字欄位時，就會傳送 FocusEvent.FOCUS_IN 事件並停用 IME。當 phoneTxt 文字欄位失去焦點時，則會傳送 FocusEvent.FOCUS_OUT 事件並重新啟用 IME。

偵聽 IME 組成事件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當設定組成字串時，就會傳送 IME 組成事件。例如，假設使用者已啟用並啟動 IME，然後以日文輸入字串，則一旦使用者選取該組成字串，就會傳送 IMEEvent.IME_COMPOSITION 事件。為了偵聽 IMEEvent.IME_COMPOSITION 事件，您必須將事件偵聽程式加入至 System 類別中的靜態 ime 屬性 (flash.system.System.ime.addEventListener(...))，如下所示：

```
var inputTxt:TextField;
var outputTxt:TextField;

inputTxt = new TextField();
inputTxt.type = TextFieldType.INPUT;
inputTxt.width = 200;
inputTxt.height = 18;
inputTxt.border = true;
inputTxt.background = true;
addChild(inputTxt);

outputTxt = new TextField();
outputTxt.autoSize = TextFieldAutoSize.LEFT;
outputTxt.y = 20;
addChild(outputTxt);

if (Capabilities.hasIME)
{
    IME.enabled = true;
    try
    {
        IME.conversionMode = IMEConversionMode.JAPANESE_HIRAGANA;
    }
    catch (error:Error)
    {
        outputTxt.text = "Unable to change IME.";
    }
    System.ime.addEventListener(IMEEvent.IME_COMPOSITION, imeCompositionHandler);
}
else
{
    outputTxt.text = "Please install IME and try again.";
}

function imeCompositionHandler(event:IMEEvent):void
{
    outputTxt.text = "you typed: " + event.text;
}
```

上述程式碼會建立兩個文字欄位，並加入顯示清單中。第一個文字欄位 inputTxt 是可讓使用者輸入日文文字的輸入文字欄位，第二個文字欄位 outputTxt 是動態文字欄位，可讓使用者看到錯誤訊息，或回應使用者在 inputTxt 文字欄位中輸入的日文字串。

虛擬鍵盤

Flash Player 10.2 以及更新的版本，**AIR 2.6** 以及更新的版本

如手機和平板電腦的行動裝置通常會提供虛擬的軟鍵盤，而非實體鍵盤。Flash API 中的類別允許您：

- 偵測虛擬鍵盤何時出現及關閉。
- 防止鍵盤出現。
- 決定虛擬鍵盤要遮住的舞台區域。
- 建立指向鍵盤時顯示鍵盤的互動物件 (iOS 上的 AIR 應用程式不支援)。
- (僅適用於 AIR) 停用自動左右相位行為，讓您的應用程式可以修改其顯示以容納鍵盤。

控制虛擬鍵盤行為

當使用者在文字欄位或特別設定的互動式物件內輕點時，執行階段會自動開啟虛擬鍵盤。當鍵盤開啟後，執行階段會依循原生平台平移與調整應用程式內容大小的慣例，讓使用者在輸入時可以看到文字。

鍵盤開啟後，成為焦點的物件會依序傳送下列事件：

softKeyboardActivating 事件 — 開始於舞台上顯示鍵盤之前立即傳送。若您對傳送的事件物件呼叫 preventDefault() 方法，則不會開啟虛擬鍵盤。

softKeyboardActivate 事件 — 在 softKeyboardActivating 事件處理完成後傳送。成為焦點的物件傳送此事件時，Stage 物件的 softKeyboardRect 屬性已經更新，以反映虛擬鍵盤遮住的舞台區域。此事件無法取消。

備註：若鍵盤大小變更，例如使用者變更鍵盤類型時，成為焦點的物件會傳送第二個 softKeyboardActivate 事件。

softKeyboardDeactivate 事件 — 在虛擬鍵盤因任何原因關閉時傳送。此事件無法取消。

下列範例會在舞台上新增兩個 TextField 物件。上方的 TextField 可避免當您輕點欄位時顯示鍵盤，若已顯示鍵盤，則會將它關閉。下方的 TextField 則示範預設行為。範例會報告這兩個文字欄位所傳送的軟體鍵盤事件。

```
package
{
import flash.display.Sprite;
import flash.text.TextField;
import flash.text.TextFieldType;
import flash.events.SoftKeyboardEvent;
public class SoftKeyboardEventExample extends Sprite
{
    private var tf1:TextField = new TextField();
    private var tf2:TextField = new TextField();

    public function SoftKeyboardEventExample()
    {
        tf1.width = this.stage.stageWidth;
        tf1.type = TextFieldType.INPUT;
        tf1.border = true;
        this.addChild( tf1 );

        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATING, preventSoftKe    yboard );
        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, preventSoftKe    yboard );
        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, preventSoftKeyboard );

        tf2.border = true;
        tf2.type = TextFieldType.INPUT;
        tf2.width = this.stage.stageWidth;
```

```
        tf2.y = tf1.y + tf1.height + 30;
        this.addChild( tf2 );

        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATING, allowSoftKeyboard );
        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, allowSoftKeyboard );
        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, allowSoftKeyboard);
    }

    private function preventSoftKeyboard( event:SoftKeyboardEvent ):void
    {
            event.preventDefault();
            this.stage.focus = null; //close the keyboard, if raised
            trace( "tf1 dispatched: " + event.type + " -- " + event.triggerType );
    }
    private function allowSoftKeyboard( event:SoftKeyboardEvent )    :void
    {
            trace( "tf2 dispatched: " + event.type + " -- " + event.triggerType );
    }
}
}
```

新增互動式物件的虛擬鍵盤支援

Flash Player 10.2 和更新的版本，**AIR 2.6** 和更新的版本 **(** 但在 **iOS** 上不支援 **)**

一般而言，只有輕點 TextField 物件時，才會開啟虛擬鍵盤。您可以設定 InteractiveObject 類別的實體，以便在接收焦點時開啟虛擬鍵盤。

若要設定 InteractiveObject 實體以開啟軟鍵盤，請將 needsSoftKeyboard 屬性設定為 true。指定物件給舞台顯示屬性時，軟鍵盤會自動開啟。此外，您可以呼叫 InteractiveObject 的 requestSoftKeyboard() 方法以顯示鍵盤。

以下範例說明您如何編寫 InteractiveObject 程式碼以作為文字輸入欄位。此範例顯示的 TextInput 類別會設定 needsSoftKeyboard 屬性，以便在需要時顯示鍵盤。然後物件會偵聽 keyDown 事件並將輸入的字元插入到欄位。

範例使用 Flash 文字引擎以附加並顯示任何輸入的文字並處理重要的事件。為了方便起見，範例不會實作完整功能的文字欄位。

```
package  {
    import flash.geom.Rectangle;
    import flash.display.Sprite;
    import flash.text.engine.TextElement;
    import flash.text.engine.TextBlock;
    import flash.events.MouseEvent;
    import flash.events.FocusEvent;
    import flash.events.KeyboardEvent;
    import flash.text.engine.TextLine;
    import flash.text.engine.ElementFormat;
    import flash.events.Event;

    public class TextInput extends Sprite
    {

        public var text:String = " ";
        public  var textSize:Number = 24;
        public var textColor:uint = 0x000000;
        private var _bounds:Rectangle = new Rectangle( 0, 0, 100, textSize );
        private var textElement: TextElement;
        private var textBlock:TextBlock = new  TextBlock();

        public function TextInput( text:String = "" )
        {
```

```
            this.text = text;
            this.scrollRect = _bounds;
            this.focusRect= false;

            //Enable keyboard support
            this.needsSoftKeyboard = true;
            this.addEventListener(MouseEvent.MOUSE_DOWN, onSelect);
            this.addEventListener(FocusEvent.FOCUS_IN, onFocusIn);
            this.addEventListener(FocusEvent.FOCUS_OUT, onFocusOut);

            //Setup text engine
            textElement = new TextElement( text, new ElementFormat( null, textSize, textColor ) );
            textBlock.content = textElement;
            var firstLine:TextLine = textBlock.createTextLine( null, _bounds.width - 8 );
            firstLine.x = 4;
            firstLine.y = 4 + firstLine.totalHeight;
            this.addChild( firstLine );

        }

        private function onSelect( event:MouseEvent ):void
        {
            stage.focus = this;
        }
        private function onFocusIn( event:FocusEvent ):void
        {
            this.addEventListener( KeyboardEvent.KEY_DOWN, onKey );
        }

        private function onFocusOut( event:FocusEvent ):void
        {
            this.removeEventListener( KeyboardEvent.KEY_UP, onKey );
        }

        private function onKey( event:KeyboardEvent ):void
        {
            textElement.replaceText( textElement.text.length, textElement.text.length,
String.fromCharCode( event.charCode ) );
            updateText();
        }
        public function set bounds( newBounds:Rectangle ):void
        {
            _bounds = newBounds.clone();
            drawBackground();
            updateText();
            this.scrollRect = _bounds;

            //force update to focus rect, if needed
            if( this.stage!= null && this.focusRect && this.stage.focus == this )
                this.stage.focus = this;
        }

        private function updateText():void
        {
            //clear text lines
            while( this.numChildren > 0 ) this.removeChildAt( 0 );

            //and recreate them
            var textLine:TextLine = textBlock.createTextLine( null, _bounds.width - 8);
            while ( textLine)
            {
                textLine.x = 4;
```

```
                if( textLine.previousLine != null )
                {
                    textLine.y = textLine.previousLine.y +
                                 textLine.previousLine.totalHeight + 2;
                }
                            else
                {
                    textLine.y = 4 + textLine.totalHeight;
                }
                this.addChild(textLine);
                textLine = textBlock.createTextLine(textLine, _bounds.width - 8 );
            }
        }

        private function drawBackground():void
        {
            //draw background and border for the field
            this.graphics.clear();
            this.graphics.beginFill( 0xededed );
            this.graphics.lineStyle( 1, 0x000000 );
            this.graphics.drawRect( _bounds.x + 2, _bounds.y + 2, _bounds.width - 4, _bounds.height - 4);
            this.graphics.endFill();
        }
    }
}
```

以下主要應用程式類別說明如何使用 TextInput 類別並在顯示鍵盤或裝置方向更改時管理應用程式配置。主要類別會建立 TextInput 物件並設定邊界以將舞台填滿。當軟鍵盤顯示或舞台大小更改時，類別會調整 TextInput 物件的大小。類別會偵聽 TextInput 物件的軟鍵盤事件並從舞台調整事件的大小。不管事件的成因，應用程式會決定舞台的可見區域並調整輸入控制大小以進行填滿。所以，不言可喻，您在實際的應用程式需要更為精密的配置演算法。

```
package  {

    import flash.display.MovieClip;
    import flash.events.SoftKeyboardEvent;
    import flash.geom.Rectangle;
    import flash.events.Event;
    import flash.display.StageScaleMode;
    import flash.display.StageAlign;

    public class CustomTextField extends MovieClip {

        private var customField:TextInput = new TextInput("Input text: ");

        public function CustomTextField() {
            this.stage.scaleMode = StageScaleMode.NO_SCALE;
            this.stage.align = StageAlign.TOP_LEFT;
            this.addChild( customField );
            customField.bounds = new Rectangle( 0, 0, this.stage.stageWidth, this.stage.stageHeight );

            //track soft keyboard and stage resize events
            customField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, onDisplayAreaChange );
            customField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, onDisplayAreaChange
);
            this.stage.addEventListener( Event.RESIZE, onDisplayAreaChange );
        }

        private function onDisplayAreaChange( event:Event ):void
        {
            //Fill the stage if possible, but avoid the area covered by a keyboard
            var desiredBounds = new Rectangle( 0, 0, this.stage.stageWidth, this.stage.stageHeight );
            if( this.stage.stageHeight - this.stage.softKeyboardRect.height < desiredBounds.height )
                desiredBounds.height = this.stage.stageHeight - this.stage.softKeyboardRect.height;

            customField.bounds = desiredBounds;
        }
    }
}
```

備註：舞台僅會傳送調整大小事件，以因應 scaleMode 屬性設定為 noScale 時的方向更改。在其他模式，舞台尺寸不會更改；而會調整內容大小以適合舞台。

處理應用程式顯示變更

AIR 2.6 和更新版本

在 AIR，您可以將應用程式描述器中的 softKeyboardBehavior 元素設定為 none，藉此關閉與顯示軟鍵盤相關的預設左右相位和調整大小行為。

```
<softKeyboardBehavior>none</softKeyboardBehavior>
```

當您關閉自動行為時，您的應用程式會負責對應用程式顯示進行所需的調整。當鍵盤開啟時，會傳送 softKeyboardActivate 事件。當傳送 softKeyboardActivate 事件時，舞台的 softKeyboardRect 屬性會包含開啟的鍵盤所遮住之區域的尺寸。使用這些尺寸來移動或調整內容大小，以便在開啟鍵盤和在使用者輸入時適當地顯示內容 (當鍵盤關閉時，softKeyboardRect 矩形的尺寸會全部為零)。

當鍵盤關閉時，會傳送 softKeyboardDeactivate 事件，而您可使應用程式顯示回復為正常狀態。

```
package  {
    import flash.display.MovieClip;
    import flash.events.SoftKeyboardEvent;
    import flash.events.Event;
    import flash.display.StageScaleMode;
    import flash.display.StageAlign;
    import flash.display.InteractiveObject;
    import flash.text.TextFieldType;
    import flash.text.TextField;

    public class PanningExample extends MovieClip {

        private var textField:TextField = new TextField();

        public function PanningExample() {
            this.stage.scaleMode = StageScaleMode.NO_SCALE;
            this.stage.align = StageAlign.TOP_LEFT;

            textField.y = this.stage.stageHeight - 201;
            textField.width = this.stage.stageWidth;
            textField.height = 200;
            textField.type = TextFieldType.INPUT;
            textField.border = true;
            textField.wordWrap = true;
            textField.multiline = true;

            this.addChild( textField );

            //track soft keyboard and stage resize events
            textField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, onKeyboardChange );
            textField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, onKeyboardChange );
            this.stage.addEventListener( Event.RESIZE, onDisplayAreaChange );
        }

        private function onDisplayAreaChange( event:Event ):void
        {
            textField.y = this.stage.stageHeight - 201;
            textField.width = this.stage.stageWidth;
        }

        private function onKeyboardChange( event:SoftKeyboardEvent ):void
        {
            var field:InteractiveObject = textField;
            var offset:int = 0;

            //if the softkeyboard is open and the field is at least partially covered
            if( (this.stage.softKeyboardRect.y != 0) && (field.y + field.height >
this.stage.softKeyboardRect.y) )
                offset = field.y + field.height - this.stage.softKeyboardRect.y;

            //but don't push the top of the field above the top of the screen
            if( field.y - offset < 0 ) offset += field.y - offset;

            this.y = -offset;
        }
    }
}
```

備註：在 Android 上，在這些情況下，包括全螢幕模式，可能無法從作業系統使用完全符合的鍵盤尺寸。在這些情況下，會估計大小。同時，在橫向方向時，會使用原生全螢幕 IME 鍵盤來輸入所有文字。IME 鍵盤是內建的文字輸入欄位，會遮住整個舞台。您無法顯示沒有填滿螢幕的橫向鍵盤。

第 31 章 滑鼠輸入

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您的應用程式可擷取與回應滑鼠輸入以建立互動性。請注意，本節假定您已經熟悉 ActionScript 3.0 事件模型的使用。如需詳細資訊，請參閱第 108 頁 「處理事件」。

如需探索執行階段期間可用滑鼠支援種類 (例如永久性游標、觸控筆或觸控輸入) 的詳細資訊，請參閱第 477 頁 「探索輸入類型」。

更多說明主題

flash.ui.Mouse

flash.events.MouseEvent

第 498 頁 「觸控、多點觸控和手勢輸入」

擷取滑鼠輸入

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

滑鼠的按一下動作會建立滑鼠事件，這些事件可以用來觸發互動性功能。您可以將事件偵聽程式加入 「舞台」上，以便偵聽在 SWF 檔中任何位置所發生的滑鼠事件。您也可以將事件偵聽程式加入「舞台」上屬於 InteractiveObject 子物件的物件中 (例如，Sprite 或 MovieClip)：按一下物件時，就會觸發這些偵聽程式。

如同鍵盤事件一樣，滑鼠事件也會反昇。在下列範例中，因為 square 是 Stage 的子系，所以當按一下正方形後，將同時從 Sprite 的 square 以及 Stage 物件傳送事件：

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0xFF0000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.addEventListener(MouseEvent.CLICK, reportClick);
square.x =
square.y = 50;
addChild(square);

stage.addEventListener(MouseEvent.CLICK, reportClick);

function reportClick(event:MouseEvent):void
{
    trace(event.currentTarget.toString() + " dispatches MouseEvent. Local coords [" + event.localX + "," +
event.localY + "] Stage coords [" + event.stageX + "," + event.stageY + "]");
}
```

在上一個範例中，請注意滑鼠事件包含了有關按一下事件的位置資訊。localX 和 localY 屬性包含發生按一下事件所在的位置 (在顯示鏈中最下層的子系)。例如，按一下 square 的左上角會報告區域座標 [0,0]，因為該位置為 square 的註冊點。或者，stageX 和 stageY 屬性也會參考到 「舞台」上按一下事件的全域座標。針對這些座標的相同按一下事件也會報告座標 [50,50]，因為 square 已經移到這些座標上。不管您希望如何回應使用者互動，這兩個座標配對都會很有幫助。

備註：在全螢幕模式中，您可以設定應用程式以使用滑鼠鎖定。滑鼠鎖定會停用游標並啟用未繫結的滑鼠移動方式。如需詳細資訊，請參閱第 141 頁 「使用全螢幕模式」。

MouseEvent 物件也包含 altKey、ctrlKey 和 shiftKey 這些 Boolean 屬性。您可以使用這些屬性來檢查發生滑鼠按一下事件時，是否同時按下了 Alt、Ctrl，或 Shift 鍵。

在舞臺上四處拖曳 Sprite

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以利用 Sprite 類別的 startDrag() 方法，允許使用者在舞臺上四處拖曳 Sprite 物件。下列程式碼將顯示此類範例：

```
import flash.display.Sprite;
import flash.events.MouseEvent;

var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFFCC00);
circle.graphics.drawCircle(0, 0, 40);

var target1:Sprite = new Sprite();
target1.graphics.beginFill(0xCCFF00);
target1.graphics.drawRect(0, 0, 100, 100);
target1.name = "target1";

var target2:Sprite = new Sprite();
target2.graphics.beginFill(0xCCFF00);
target2.graphics.drawRect(0, 200, 100, 100);
target2.name = "target2";

addChild(target1);
addChild(target2);
addChild(circle);

circle.addEventListener(MouseEvent.MOUSE_DOWN, mouseDown)

function mouseDown(event:MouseEvent):void
{
    circle.startDrag();
}
circle.addEventListener(MouseEvent.MOUSE_UP, mouseReleased);

function mouseReleased(event:MouseEvent):void
{
    circle.stopDrag();
    trace(circle.dropTarget.name);
}
```

如需詳細資訊，請參閱第 146 頁「變更位置」中建立滑鼠拖曳互動一節的內容。

在 AIR 中拖放

在 Adobe AIR 中，您可以啟用拖放支援，允許使用者將資料拖曳至應用程式或從應用程式拖出資料。如需詳細資訊，請參閱第 521 頁「在 AIR 中拖放」。

自訂滑鼠游標

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以隱藏 「舞台」上任何顯示物件的滑鼠游標 (滑鼠指標) 或是將它的滑鼠按鍵對換。若要隱藏滑鼠游標，請呼叫 Mouse.hide() 方法。呼叫 Mouse.hide()、偵聽 「舞台」的 MouseEvent.MOUSE_MOVE 事件，以及將顯示物件 (您的自訂游標) 的座標設定為該事件的 stageX 和 stageY 屬性，即可自訂游標。下列範例將說明這項工作的基本執行方式：

```
var cursor:Sprite = new Sprite();
cursor.graphics.beginFill(0x000000);
cursor.graphics.drawCircle(0,0,20);
cursor.graphics.endFill();
addChild(cursor);

stage.addEventListener(MouseEvent.MOUSE_MOVE,redrawCursor);
Mouse.hide();

function redrawCursor(event:MouseEvent):void
{
    cursor.x = event.stageX;
    cursor.y = event.stageY;
}
```

滑鼠輸入範例：WordSearch

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

此範例將藉由處理滑鼠事件來示範使用者互動。使用者儘可能地在包含隨機排列字母的格點狀拼字遊戲表中建立許多單字，過程中需透過水平或垂直移動表中的字母來完成拼字，但是絕對不可重複使用相同的字母。這個範例可示範下列的 ActionScript 3.0 功能：

- 以動態方式建立組件的格點
- 回應滑鼠事件
- 根據使用者互動維護得分記錄

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。您可以在 Samples/WordSearch 檔案夾中找到 WordSearch 應用程式檔案，此應用程式是由下列檔案組成：

檔案	說明
WordSearch.as	提供應用程式主要功能的類別。
WordSearch.fla 或 WordSearch.mxml	主應用程式檔案，在 Flex 中為 MXML，在 Flash 中為 FLA。
dictionary.txt	用來判斷拼字是否可得分以及是否正確的檔案。

載入字典

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在建立用來尋找單字的遊戲時，會需要用到字典。這個範例中包括名為 dictionary.txt 的文字檔案，其中包含一份以換行符號分隔的單字清單。建立名為 words 的陣列之後，loadDictionary() 函數便需要這個檔案，而在成功載入檔案後，這個檔案就會變成一個很長的字串。您可以使用 split() 從每一個出現歸位字元 (字元碼 10) 或換行符號 (字元碼 13) 實體的位置處斷開，將這個字串剖析成單字陣列。這個剖析動作會在 dictionaryLoaded() 函數中發生：

```
words = dictionaryText.split(String.fromCharCode(13, 10));
```

建立使用者介面

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在儲存單字之後，就可以開始設定使用者介面。建立兩個 Button 實體：一個用來送出單字，另一個用來清除目前正在拼湊的單字。在每個情況中，您必須偵聽該按鈕所廣播的 MouseEvent.CLICK 事件，並接著呼叫函數以回應使用者輸入。在 setupUI() 函數中，下列程式碼會在這兩個按鈕上分別建立偵聽程式：

```
submitWordButton.addEventListener(MouseEvent.CLICK,submitWord);
clearWordButton.addEventListener(MouseEvent.CLICK,clearWord);
```

產生遊戲表

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

遊戲表就是一個包含一些隨機排列字母的格點表。在 generateBoard() 函數中，您可以將一個迴圈嵌入另一個迴圈，來建立二維格點。第一個迴圈可增加列數，而第二個迴圈則是增加每列的總欄數。由這些列與欄所交錯組合而成的儲存格，每一個都包含用來代表表中字母的按鈕。

```
private function generateBoard(startX:Number, startY:Number, totalRows:Number, totalCols:Number,
buttonSize:Number):void
{
    buttons = new Array();
    var colCounter:uint;
    var rowCounter:uint;
    for (rowCounter = 0; rowCounter < totalRows; rowCounter++)
    {
        for (colCounter = 0; colCounter < totalCols; colCounter++)
        {
            var b:Button = new Button();
            b.x = startX + (colCounter*buttonSize);
            b.y = startY + (rowCounter*buttonSize);
            b.addEventListener(MouseEvent.CLICK, letterClicked);
            b.label = getRandomLetter().toUpperCase();
            b.setSize(buttonSize,buttonSize);
            b.name = "buttonRow"+rowCounter+"Col"+colCounter;
            addChild(b);

            buttons.push(b);
        }
    }
}
```

雖然每一行程式碼中只能為 MouseEvent.CLICK 事件新增一個偵聽程式，由於它將用於 for 迴圈，所以將會指定給每個 Button 實體。同時，每個按鈕也會指定一個來自所屬列與欄位置的名稱，這樣稍後要在程式碼中參考每個按鈕的列與欄值就會比較容易。

透過使用者輸入來建立單字

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以透過選取水平或垂直相鄰的字母來拼出單字，但是絕對不可重複使用同一個字母。每次按一下都會產生一個滑鼠事件，並在當下檢查使用者所拼的單字，以確定所按的字母能夠接續先前所按下的字母。如果沒有接續正確，就會移除上一個單字並開始新的拼字階段。這項檢查會在 isLegalContinuation() 方法中發生。

```
private function isLegalContinuation(prevButton:Button, currButton:Button):Boolean
{
    var currButtonRow:Number = Number(currButton.name.charAt(currButton.name. indexOf("Row") + 3));
    var currButtonCol:Number = Number(currButton.name.charAt(currButton.name.indexOf("Col") + 3));
    var prevButtonRow:Number = Number(prevButton.name.charAt(prevButton.name.indexOf("Row") + 3));
    var prevButtonCol:Number = Number(prevButton.name.charAt(prevButton.name.indexOf("Col") + 3));

    return ((prevButtonCol == currButtonCol && Math.abs(prevButtonRow - currButtonRow) <= 1) ||
             (prevButtonRow == currButtonRow && Math.abs(prevButtonCol - currButtonCol) <= 1));
}
```

String 類別的 charAt() 和 indexOf() 方法都會從目前已按下的按鈕與之前已按下的按鈕中取得適當的列和欄。此時，如果列或欄沒有變更，或是已經變更的列或欄是在上一次變更以來的單一增量範圍內，isLegalContinuation() 方法就會傳回 true。如果您想要變更遊戲規則，並允許對角拼字，則可以移除未變更的列或欄檢查，而且最後一行的程式碼將如下所示：

```
return (Math.abs(prevButtonRow - currButtonRow) <= 1) && Math.abs(prevButtonCol - currButtonCol) <= 1));
```

檢查送出的單字

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要完成遊戲程式碼，需要建立檢查送出的單字以及計算分數的機制。searchForWord() 方法同時包含：

```
private function searchForWord(str:String):Number
{
    if (words && str)
    {
        var i:uint = 0
        for (i = 0; i < words.length; i++)
        {
            var thisWord:String = words[i];
            if (str == words[i])
            {
                return i;
            }
        }
        return -1;
    }
    else
    {
        trace("WARNING: cannot find words, or string supplied is null");
    }
    return -1;
}
```

這個函數會循序處理字典中的所有單字。如果使用者所拼的單字與字典中的單字相符，就會傳回該單字在字典中的位置。如果是有效的位置，submitWord() 方法會接著檢查回應並更新得分記錄。

自訂作業

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

類別的一開始是好幾個常數。您可以藉由修改這些變數來修改此遊戲。例如，您可以增加 TOTAL_TIME 變數來改變可以使用的遊戲時間。您也可以稍微增加 PERCENT_VOWELS 變數，提高找到單字的可能性。

第 32 章 觸控、多點觸控和手勢輸入

Flash Player 10.1 以及更新的版本，**Adobe AIR 2** 以及更新的版本

Flash Platform 的觸控事件處理功能包括來自觸控裝置上單一接觸點或多個接觸點的輸入。而且，Flash 執行階段會處理合併多個觸控點與移動的事件，以建立手勢。換句話說，Flash 執行階段會解譯兩種類型的輸入：

觸控 使用單點裝置 (例如手指、觸控筆或觸控裝置上的其他工具) 的輸入。部分裝置支援使用手指或觸控筆同時接觸多個點。

多點觸控 多個同時接觸點的輸入。

手勢 裝置或作業系統回應一或多個觸控事件，所解譯的輸入。例如，使用者同時旋轉兩根手指，而裝置或作業系統將該觸控輸入解譯為旋轉手勢。部分手勢是透過一根手指或觸控點來執行，部分手勢則需要多個觸控點。裝置或作業系統會建立要指定給輸入的手勢類型。

根據使用者的裝置，觸控和手勢輸入可以進行多點觸控輸入。ActionScript 提供 API 來處理觸控事件、手勢事件，以及個別追蹤且進行多點觸控輸入的觸控事件。

備註：根據處理運算裝置與作業系統而定，偵聽觸控與筆勢事件會耗用大量資源 (相當於每秒顯示數個影格)。當您不是真的需要經由觸控或手勢提供的額外功能時，通常最好使用滑鼠事件。當您確實使用觸控或手勢事件時，請考量降低會發生圖形變更的數量，特別是在快速傳送這類事件時，例如，在平移、旋轉或縮放作業期間。例如，當使用者使用縮放手勢來調整組件大小時，您可以停止該組件內的動畫。

更多說明主題

flash.ui.Multitouch

flash.events.TouchEvent

flash.events.GestureEvent

flash.events.TransformGestureEvent

flash.events.GesturePhase

flash.events.PressAndTapGestureEvent

Paul Trani：行動裝置上的觸控事件與手勢

Mike Jones：虛擬遊戲控制器 (英文)

觸控輸入的基本概念

Flash Player 10.1 以及更新的版本，**Adobe AIR 2** 以及更新的版本

Flash Platform 在支援觸控輸入的環境中執行時，InteractiveObject 實體可以偵聽觸控事件，並呼叫處理常式。通常，您會像處理 ActionScript 中的其他事件一樣，處理觸控、多點觸控和手勢事件 (如需使用 ActionScript 處理事件的基本資訊，請參閱第 108 頁「處理事件」)。

不過，為了讓 Flash 執行階段解譯觸控或手勢，執行階段必須在支援觸控或多點觸控輸入的硬體和軟體環境中執行。如需比較不同觸控螢幕類型的圖表，請參閱第 477 頁「探索輸入類型」。此外，如果執行階段是在容器應用程式 (例如瀏覽器) 內執行，則該容器會將輸入傳遞給執行階段。在部分情況下，目前的硬體和作業系統環境支援多點觸控，但是含有 Flash 執行階段的瀏覽器只會解譯輸入，並不會將它傳遞給執行階段。或者，就直接忽略所有輸入。

下圖顯示從使用者到執行階段的輸入流程：

從使用者到 Flash Platform 執行階段的輸入流程

幸運的是，開發觸控應用程式的 ActionScript API 包括類別、方法和屬性，以判斷執行階段環境中的觸控或多點觸控輸入支援。您用來判斷觸控輸入支援的 API 是進行觸控事件處理的「探索 API」。

重要概念與術語

下列參考清單包含撰寫觸控事件處理應用程式的重要術語：

探索 API 用來測試執行階段環境是否支援觸控事件和不同輸入模式的方法和屬性。

觸控事件 使用單一接觸點，在觸控裝置上執行的輸入動作。

觸控點 單一觸控事件的接觸點。即使裝置不支援手勢輸入，還是可能支援多個同時觸控點。

觸控順序 代表單一觸控之生命期的一連串事件。這些事件包括一個開始、零個以上的移動，以及一個結束。

多點觸控事件 使用數個接觸點 (例如多根手指) 在觸控裝置上執行的輸入動作。

手勢事件 追蹤某個複雜移動而在觸控裝置上執行的輸入動作。例如，一個手勢是用兩根手指觸控螢幕，並將它們同時沿著抽象圓形 (具長寬屬性) 的周圍移動，以表示旋轉。

階段 事件流程中的不同時間點 (例如開始和結束)。

觸控筆 與觸控螢幕互動的工具。觸控筆的精確度高於人的手指。部分裝置只能辨識來自特定類型觸控筆的輸入。可以辨識觸控筆輸入的裝置，可能無法辨識多個同時接觸點或手指接觸。

按下並輕點 特定類型的多點觸控輸入手勢，其中，使用者用手指按下觸控裝置，然後使用另一根手指或另一個指標裝置進行輕點。在多點觸控應用程式中，這個手勢通常是用來模擬按一下滑鼠右鍵。

觸控輸入 API 結構

ActionScript 觸控輸入 API 的設計，是要解決觸控輸入處理取決於 Flash 執行階段之硬體和軟體環境的問題。觸控輸入 API 主要解決觸控應用程式開發的三個需求：探索、事件和階段。協調這些 API，以針對使用者產生可預測且具回應的體驗；即使在您開發應用程式時目標裝置不明仍然可進行開發。

探索

探索 API 提供執行階段測試硬體和軟體環境的能力。執行階段所填入的值，可決定 Flash 執行階段在其目前環境中可用的觸控輸入。此外，可使用探索屬性和方法的集合，來設定應用程式回應滑鼠事件 (非觸控事件，以防環境不支援部分觸控輸入時)。如需詳細資訊，請參閱第 500 頁「觸控支援探索」。

事件

ActionScript 使用事件偵聽程式和事件處理常式來管理觸控輸入事件，就和管理其他事件一樣。不過，觸控輸入事件處理也必須考量下列各項：

- 裝置或作業系統可能會以多種方式解譯觸控，可能是一連串的觸控或統稱為手勢。
- 觸控裝置的單一觸控 (透過手指、觸控筆或指標裝置) 一律也會傳送滑鼠事件。您可以使用 MouseEvent 類別中的事件類型來處理滑鼠事件，或是設計應用程式只回應觸控事件。您也可以設計應用程式同時回應兩者。
- 應用程式可以回應多個同時觸控事件，以及分別處理每個事件。

一般會使用探索 API 視條件處理應用程式所處理的事件及其處理方式。應用程式辨識執行階段環境之後，就可以呼叫適當的處理常式，或在使用者與應用程式互動時建立正確的事件物件。應用程式也可以指出無法在目前環境中處理特定輸入，以及提供使用者替代方式或資訊。如需詳細資訊，請參閱第 501 頁 「觸控事件處理」和第 504 頁 「手勢事件處理」。

階段

如果是觸控和多點觸控應用程式，則觸控事件物件包含的屬性，可追蹤使用者互動的階段。撰寫 ActionScript 來處理階段 (例如使用者輸入的開始、更新或結束階段)，以回應使用者。回應事件階段後，視覺物件會隨著使用者觸控和移動螢幕上的觸控點而變更。或是使用階段來追蹤手勢在變換時的特定屬性。

如果是觸控點事件，則會追蹤使用者停留在特定互動式物件上的時間長短。應用程式可以個別追蹤多個同時觸控點的階段，並據此處理各階段。

如果是手勢，則會解譯進行手勢轉換的特定資訊。追蹤單一或多個接觸點在螢幕上移動時的座標。

觸控支援探索

Flash Player 10.1 以及更新的版本，**Adobe AIR 2** 以及更新的版本

使用 Multitouch 類別屬性，設定應用程式所處理的觸控輸入範圍。然後測試環境，確定支援 ActionScript 所處理的事件。更明確地說，要先建立應用程式的觸控輸入類型。選項為：觸控點、手勢或無 (將所有觸控輸入解譯為按一下滑鼠，並只使用滑鼠事件處理常式)。然後，使用 Multitouch 類別的屬性和方法，確定執行階段處於支援您應用程式所需的觸控輸入環境。測試觸控輸入類型支援的執行階段環境 (例如是否可以解譯手勢)，並據此回應。

備註：Multitouch 類別屬性是靜態屬性，而且不屬於任何類別的實體。請以 Multitouch.property 語法來使用它們，例如：

```
var touchSupport:Boolean = Multitouch.supportsTouchEvents;
```

設定輸入類型

因為觸控事件可能會有多個元素或階段，所以 Flash 執行階段必須辨識要解譯的觸控輸入類型。如果手指只是碰觸觸控螢幕，那麼執行階段會傳送觸控事件？等待手勢？還是執行階段會將觸控追蹤為按下滑鼠事件？支援觸控輸入的應用程式必須建立觸控事件類型，應用程式會在 Flash 執行階段處理這些觸控事件類型。使用 Multitouch.inputMode 屬性建立執行階段的觸控輸入類型。輸入模式可以是三個選項的其中一項：

無 不特別處理觸控事件。設定 Multitouch.inputMode=MultitouchInputMode.NONE，並使用 MouseEvent 類別來處理輸入。

單一觸控點 個別解譯所有觸控輸入，而且可以追蹤和處理所有觸控點。設定 Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT，並使用 TouchEvent 類別來處理輸入。

手勢輸入 裝置或作業系統會將輸入解譯為手指在螢幕上移動的複雜形式。裝置或作業系統會統一將移動指定給單一手勢輸入事件。設定 Multitouch.inputMode=MultitouchInputMode.GESTURE，並使用 TransformGestureEvent、PressAndTapGestureEvent 或 GestureEvent 類別來處理輸入。

如需使用 Multitouch.inputMode 屬性在處理觸控事件之前設定輸入類型的範例，請參閱第 501 頁 「觸控事件處理」。

觸控輸入支援的測試

Multitouch 類別的其他屬性所提供的值，可用來將應用程式微調為目前環境的觸控支援。Flash 執行階段會針對允許的同時觸控點數目或可用的手勢來填入值。如果執行階段所處的環境，不支援應用程式需要處理的觸控事件，請提供替代方式讓使用者使用。例如，提供滑鼠事件處理，或目前環境可用或不可用功能的資訊。

您也可以使用 API 進行鍵盤、觸控和滑鼠支援，請參閱第 477 頁 「探索輸入類型」。

如需有關相容性測試的詳細資訊，請參閱第 508 頁 「疑難排解」。

觸控事件處理

Flash Player 10.1 以及更新的版本，**Adobe AIR 2** 以及更新的版本

基本觸控事件的處理方式，與 ActionScript 處理其他事件的方式相同 (例如滑鼠事件)。您可以偵聽 TouchEvent 類別中之事件類型常數所定義的一連串觸控事件。

備註：針對多個觸控點輸入 (例如使用多根手指觸控裝置)，第一個接觸點會傳送滑鼠事件和觸控事件。

處理基本觸控事件：

1 將 flash.ui.Multitouch.inputMode 屬性設為 MultitouchInputMode.TOUCH_POINT，以設定應用程式來處理觸控事件。
2 將事件偵聽程式連接至類別實體，而此類別是繼承 InteractiveObject 類別 (例如 Sprite 或 TextField) 的屬性。
3 指定要處理的觸控事件類型。
4 呼叫事件處理常式函數，以執行回應事件的作業。

例如，下列程式碼會在觸控螢幕內 mySprite 中繪製的正方形被點選時顯示訊息：

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taphandler);

function taphandler(evt:TouchEvent): void {
    myTextField.text = "I've been tapped";
    myTextField.y = 50;
    addChild(myTextField);
}
```

觸控事件屬性

發生事件時，會建立事件物件。TouchEvent 物件包含的資訊，是有關觸控事件的位置和條件。您可以使用事件物件的屬性來擷取該資訊。

例如，下列程式碼會建立 TouchEvent 物件 evt，然後在文字欄位中顯示事件物件的 stageX 屬性 (「舞台」空間中觸控發生點的 x 座標)：

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taphandler);

function taphandler(evt:TouchEvent): void {
myTextField.text = evt.stageX.toString;
myTextField.y = 50;
addChild(myTextField);
}
```

請參閱 TouchEvent 類別，取得事件物件可用屬性的詳細資訊。

備註：在所有執行階段環境中，並非支援所有 TouchEvent 屬性。例如，並非所有觸控裝置都可以偵測使用者施予觸控螢幕的壓力。因此，無法偵測壓力的裝置就不支援 TouchEvent.pressure 屬性。嘗試測試特定屬性支援，以確保應用程式可以運作，並參閱第 508 頁 「疑難排解」，以取得詳細資訊。

觸控事件階段

在 InteractiveObject 上和外部，透過各種舞台來追蹤觸控事件，就像處理滑鼠事件一樣。而且，透過觸控互動的開始、中間和結束，來追蹤觸控事件。TouchEvent 類別提供的值可用來處理 touchBegin、touchMove 和 touchEnd 事件。

例如，您可以使用 touchBegin、touchMove 和 touchEnd 事件，在使用者觸碰和移動顯示物件時提供視覺回應：

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);
var myTextField:TextField = new TextField();
myTextField.width = 200;
myTextField.height = 20;
addChild(myTextField);

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
stage.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
stage.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);
function onTouchBegin(event:TouchEvent) {
    myTextField.text = "touch begin" + event.touchPointID;
}
function onTouchMove(event:TouchEvent) {
    myTextField.text = "touch move" + event.touchPointID;
}
function onTouchEnd(event:TouchEvent) {
    myTextField.text = "touch end" + event.touchPointID;
}
```

備註：初始觸控偵聽程式會連接至 mySprite，但是移動和結束觸控事件的偵聽程式則否。如果使用者的手指或指標裝置移動到顯示物件的前面，則 「舞台」還是會繼續偵聽觸控事件。

觸控點 ID

TouchEvent.touchPointID 屬性是撰寫回應觸控輸入之應用程式的必要部分。Flash 執行階段會將唯一的 touchPointID 值指定給每個觸控點。只要應用程式回應觸控輸入的階段或移動，就會先檢查 touchPointID，再處理事件。Sprite 類別的觸控輸入拖曳方法，使用 touchPointID 屬性做為參數，來處理正確的輸入實體。touchPointID 屬性可確保事件處理常式回應正確的觸控點。否則，事件處理常式會回應裝置上觸控事件類型的所有實體 (例如所有 touchMove 事件)，進而產生無法預測的行為。在使用者拖曳物件時，這個屬性特別重要。

touchPointID 屬性可用來管理整個觸控序列。觸控序列具有一個 touchBegin 事件、零個或以上的 touchMove 事件，以及一個具有相同 touchPointID 值的 touchEnd 事件。

下列範例會先建立 touchMoveID 變數以測試正確的 touchPointID 值，再回應觸控移動事件。否則，其他觸控輸入也會觸發事件處理常式。請注意，適用於移動和結束階段的偵聽程式位於舞台上，而不是顯示物件。如果使用者的觸控移動超出顯示物件的界限，則舞台會偵聽移動或結束階段。

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);
var myTextField:TextField = new TextField();
addChild(myTextField);
myTextField.width = 200;
myTextField.height = 20;
var touchMoveID:int = 0;

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
function onTouchBegin(event:TouchEvent) {
    if(touchMoveID != 0) {
        myTextField.text = "already moving. ignoring new touch";
        return;
    }
    touchMoveID = event.touchPointID;

    myTextField.text = "touch begin" + event.touchPointID;
    stage.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
    stage.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);
}
function onTouchMove(event:TouchEvent) {
    if(event.touchPointID != touchMoveID) {
        myTextField.text = "ignoring unrelated touch";
        return;
    }
    mySprite.x = event.stageX;
    mySprite.y = event.stageY;
    myTextField.text = "touch move" + event.touchPointID;
}
function onTouchEnd(event:TouchEvent) {
    if(event.touchPointID != touchMoveID) {
        myTextField.text = "ignoring unrelated touch end";
        return;
    }
    touchMoveID = 0;
    stage.removeEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
    stage.removeEventListener(TouchEvent.TOUCH_END, onTouchEnd);
    myTextField.text = "touch end" + event.touchPointID;
}
```

觸控和拖曳

Flash Player 10.1 以及更新的版本，**Adobe AIR 2** 以及更新的版本

在 Sprite 類別中新增兩種方法，以提供支援觸控點輸入之觸控應用程式的其他支援：Sprite.startTouchDrag() 和 Sprite.stopTouchDrag()。這些方法的行為與滑鼠事件的 Sprite.startDrag() 和 Sprite.stopDrag() 的行為相同。不過，請注意，Sprite.startTouchDrag() 和 Sprite.stopTouchDrag() 方法都是採用 touchPointID 值做為參數。

執行階段會將 touchPointID 值指定給觸控事件的事件物件。如果環境支援同時間的多個觸控點，則這個值可用來回應特定觸控點 (即使不處理手勢也是一樣)。如需 touchPointID 屬性有關的詳細資訊，請參閱第 502 頁 「觸控點 ID」。

下列程式碼顯示簡單的觸控事件開始拖曳事件處理常式和停止拖曳事件處理常式。bg 變數是含有 mySprite 的顯示物件：

```
mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
mySprite.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);

function onTouchBegin(e:TouchEvent) {
    e.target.startTouchDrag(e.touchPointID, false, bg.getRect(this));
    trace("touch begin");
}

function onTouchEnd(e:TouchEvent) {
    e.target.stopTouchDrag(e.touchPointID);
    trace("touch end");
}
```

而下列顯示更進階的範例，即合併拖曳與觸控事件階段：

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
mySprite.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
mySprite.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);

function onTouchBegin(evt:TouchEvent) {
    evt.target.startTouchDrag(evt.touchPointID);
    evt.target.scaleX *= 1.5;
    evt.target.scaleY *= 1.5;
}

function onTouchMove(evt:TouchEvent) {
    evt.target.alpha = 0.5;
}

function onTouchEnd(evt:TouchEvent) {
    evt.target.stopTouchDrag(evt.touchPointID);
    evt.target.width = 40;
    evt.target.height = 40;
    evt.target.alpha = 1;
}
```

手勢事件處理

Flash Player 10.1 以及更新的版本，**Adobe AIR 2** 以及更新的版本

使用與基本觸控事件相同的方式，處理手勢事件。您可以偵聽 TransformGestureEvent 類別、GestureEvent 類別和 PressAndTapGestureEvent 類別中事件類型常數所定義的一連串手勢事件。

若要處理手勢觸控事件：

1. 將 flash.ui.Multitouch.inputMode 屬性設為 MultitouchInputMode.GESTURE，以設定應用程式來處理手勢輸入。
2. 將事件偵聽程式連接至類別實體，而此類別是繼承 InteractiveObject 類別 (例如 Sprite 或 TextField) 的屬性。
3. 指定要處理的手勢事件類型。
4. 呼叫事件處理常式函數，以執行回應事件的作業。

例如，下列程式碼會在揮動觸控螢幕內 mySprite 中繪製的正方形時顯示訊息：

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TransformGestureEvent.GESTURE_SWIPE, swipehandler);

function swipehandler(evt:TransformGestureEvent): void {
myTextField.text = "I've been swiped";
myTextField.y = 50;
addChild(myTextField);
}
```

兩根手指點選事件的處理方式相同，但是使用 GestureEvent 類別：

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(GestureEvent.GESTURE_TWO_FINGER_TAP, taphandler);

function taphandler(evt:GestureEvent): void {
myTextField.text = "I've been two-finger tapped";
myTextField.y = 50;
addChild(myTextField);
}
```

按下並輕點事件的處理方式也相同，但是使用 PressAndTapGestureEvent 類別：

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(PressAndTapGestureEvent.GESTURE_PRESS_AND_TAP, taphandler);

function taphandler(evt:PressAndTapGestureEvent): void {
myTextField.text = "I've been press-and-tapped";
myTextField.y = 50;
addChild(myTextField);
}
```

備註：在所有執行階段環境中，並非支援所有 GestureEvent、TransformGestureEvent 和 PressAndTapGestureEvent 事件類型。例如，並非所有觸控裝置都可以偵測多根手指揮動。因此，無法偵測多根手指揮動的裝置就不支援 InteractiveObject gestureSwipe 事件。嘗試測試特定事件支援，以確保應用程式可以運作，並參閱第 508 頁「疑難排解」，以取得詳細資訊。

手勢事件屬性

手勢事件的事件屬性範圍小於基本觸控事件。而其存取方式一樣是透過事件處理常式函數中的事件物件。

例如，使用者在 mySprite 上執行旋轉手勢時，下列程式碼會將它旋轉。這個文字欄位會顯示最後一個手勢之後的旋轉量 (測試這個程式碼時，將它旋轉數次以查看值變更)：

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var mySpriteCon:Sprite = new Sprite();
var myTextField:TextField = new TextField();
myTextField.y = 50;
addChild(myTextField);

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(-20,-20,40,40);
mySpriteCon.addChild(mySprite);
mySprite.x = 20;
mySprite.y = 20;
addChild(mySpriteCon);

mySprite.addEventListener(TransformGestureEvent.GESTURE_ROTATE, rothandler);

function rothandler(evt:TransformGestureEvent): void {
evt.target.parent.rotationZ += evt.target.rotation;
myTextField.text = evt.target.parent.rotation.toString();
}
```

備註：在所有執行階段環境中，並非支援所有 TransformGestureEvent 屬性。例如，並非所有觸控裝置都可以偵測螢幕上的手勢旋轉。因此，無法偵測手勢旋轉的裝置就不支援 TransformGestureEvent.rotation 屬性。嘗試測試特定屬性支援，以確保應用程式可以運作，並參閱第 508 頁 「疑難排解」，以取得詳細資訊。

手勢階段

此外，還可以透過階段來追蹤手勢事件，因此您可以在執行手勢時追蹤屬性。例如，您可以追蹤在使用揮動手勢移動物件時的 x 座標。使用這些值，可以在揮動完成之後，透過路徑中的所有點來繪製線條。或是，在螢幕中使用平移手勢拖曳顯示物件時，以視覺方式變更該顯示物件。請在平移手勢完成之後再次變更物件。

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;
var mySprite = new Sprite();
mySprite.addEventListener(TransformGestureEvent.GESTURE_PAN , onPan);
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0, 0, 40, 40);
var myTextField = new TextField();
myTextField.y = 200;
addChild(mySprite);
addChild(myTextField);

function onPan(evt:TransformGestureEvent):void {

    evt.target.localX++;

if (evt.phase==GesturePhase.BEGIN) {
    myTextField.text = "Begin";
    evt.target.scaleX *= 1.5;
    evt.target.scaleY *= 1.5;
}
if (evt.phase==GesturePhase.UPDATE) {
    myTextField.text = "Update";
    evt.target.alpha = 0.5;
}
if (evt.phase==GesturePhase.END) {
    myTextField.text = "End";
    evt.target.width = 40;
    evt.target.height = 40;
    evt.target.alpha = 1;
}
}
```

備註：更新階段的頻率取決於執行階段的環境。部分的作業系統和硬體組合則根本不會轉送更新。

簡單手勢事件的手勢階段是「all」

部分的手勢事件物件不會追蹤手勢事件的個別階段，而是改為以 all 值填入事件物件的 phase 屬性。簡單手勢會揮動，而兩根手指點選則不會透過多個階段來追蹤事件。針對偵聽 gestureSwipe 或 gestureTwoFingerTap 事件，其 InteractiveObject 事件物件的 phase 屬性在傳送事件之後一律會是 all：

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;
var mySprite = new Sprite();
mySprite.addEventListener(TransformGestureEvent.GESTURE_SWIPE, onSwipe);
mySprite.addEventListener(GestureEvent.GESTURE_TWO_FINGER_TAP, onTwoTap);
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0, 0, 40, 40);
var myTextField = new TextField();
myTextField.y = 200;
addChild(mySprite);
addChild(myTextField);

function onSwipe(swipeEvt:TransformGestureEvent):void {
    myTextField.text = swipeEvt.phase // Output is "all"
}
function onTwoTap(tapEvt:GestureEvent):void {
    myTextField.text = tapEvt.phase // Output is "all"
}
```

疑難排解

Flash Player 10.1 以及更新的版本，**Adobe AIR 2** 以及更新的版本

觸控輸入之硬體和軟體支援的變更速度極為快速。這項參考不會列出支援多點觸控之作業系統和軟體組合的每個裝置清單。不過，它提供使用探索 API 的準則，來判斷您的應用程式是否部署於支援多點觸控的裝置，以及提供 ActionScript 程式碼的疑難排解提示。

Flash 執行階段會根據裝置、作業系統或包含軟體 (例如瀏覽器) 傳遞給執行階段的資訊，來回應觸控事件。軟體環境上的這個依存關係，讓製作多點觸控相容性的說明文件更形複雜。部分裝置解譯手勢或觸控動作的方式會與其他裝置不同。旋轉是否透過同時旋轉兩根手指來定義？旋轉一根手指是否在螢幕上繪製圓形？根據硬體和軟體環境，旋轉手勢可能是上述情況之一，但有時則完全不同。因此，裝置會將使用者輸入告知作業系統，然後作業系統將該資訊傳遞給執行階段。如果執行階段位於瀏覽器，則瀏覽器軟體有時會解譯手勢或觸控事件，而且不會將輸入傳遞給執行階段。這種行為與「快速鍵」的行為類似：您會嘗試使用特定的按鍵組合讓 Flash Player 在瀏覽器內執行某個作業，而瀏覽器反而會一直開啟某個功能表。

個別的 API 和類別若與特定作業系統不相容，則會特別說明。您可以在此瀏覽個別的 API 項目，從 Multitouch 類別開始：http://help.adobe.com/zh_TW/FlashPlatform/reference/actionscript/3/flash/ui/Multitouch.html。

此下是一些常見的手勢和觸控說明：

平移 從左到右或從右到左移動手指。部分裝置需要兩根手指才能進行平移。

旋轉 用兩根手指觸控，然後將它們以圓形移動 (就像它們同時追蹤表面上的假想圓形)。而樞紐點會設定於兩根手指觸控點之間的中點。

揮動 快速從左到右或從右到左、從上到下或從下到上移動三根手指。

縮放 用兩根手指觸控，然後將它們移離彼此以放大顯示，移近彼此以縮小顯示。

按下並輕點 移動或按下一根手指，然後使用另一根手指輕點表面。

每個裝置都有該裝置所支援之手勢的文件，以及如何在該裝置上執行每個手勢的文件。一般而言，使用者必須在手勢之間，讓所有手指離開接觸的裝置 (視作業系統而定)。

如果您發現應用程式未回應觸控事件或手勢，請測試下列項目：

1. 觸控或手勢事件連接至繼承自 InteractiveObject 類別的物件類別，而您是否為該事件設定事件偵聽程式？只有 InteractiveObject 實體才能偵聽觸控和手勢事件。
2. 您是否在 Flash Professional CS5 內測試應用程式？如果是這樣，因為 Flash Professional 可以攔截互動，請嘗試發佈和測試應用程式。
3. 先從簡單的開始，並查看哪一項會發生作用 (下列程式碼範例是來自 Multitouch.inputMode 的 API 項目)：

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField()

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taplistener);

function taplistener(e:TouchEvent): void {
    myTextField.text = "I've been tapped";
    myTextField.y = 50;
    addChild(myTextField);
}
```

點選矩形。如果這個範例可行，便可以知道環境支援簡單點選。然後，您可以嘗試較複雜的處理。

手勢支援的測試較為複雜。個別的裝置或作業系統支援任何手勢輸入組合，或不支援任何輸入。

以下是縮放手勢的簡單測試：

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;

stage.addEventListener(TransformGestureEvent.GESTURE_ZOOM , onZoom);
var myTextField = new TextField();
myTextField.y = 200;
myTextField.text = "Perform a zoom gesture";
addChild(myTextField);

function onZoom(evt:TransformGestureEvent):void {
    myTextField.text = "Zoom is supported";
}
```

在裝置上執行縮放手勢，並查看文字欄位是否填入 Zoom is supported 訊息。事件偵聽程式會新增至舞台，所以您可以在測試應用程式的任何部分執行手勢。

以下是平移手勢的簡單測試：

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;

stage.addEventListener(TransformGestureEvent.GESTURE_PAN , onPan);
var myTextField = new TextField();
myTextField.y = 200;
myTextField.text = "Perform a pan gesture";
addChild(myTextField);

function onPan(evt:TransformGestureEvent):void {
    myTextField.text = "Pan is supported";
}
```

在裝置上執行平移手勢，並查看文字欄位是否填入 Pan is supported 訊息。事件偵聽程式會新增至舞台，所以您可以在測試應用程式的任何部分執行手勢。

有些作業系統和裝置組合支援這兩種手勢，有些只支援其中一種，有些兩者都不支援。測試應用程式的部署環境，加以確定。

已知問題

下列是觸控輸入的已知問題：

1 Windows Mobile 作業系統上的行動 Internet Explorer 會自動縮放 SWF 檔案內容：

將下列項目新增至裝載 SWF 檔案的 HTML 網頁，可覆寫這種 Internet Explorer 縮放行為：

```
<head>
<meta name="viewport" content="width=device-width, height=device-height, initial-scale=1.0">
</head>
```

2 Windows 7 (或其他作業系統)，使用者必須在手勢之間讓指標裝置 (或手指) 離開螢幕。例如，旋轉和縮放影像：

- 執行旋轉手勢。
- 讓手指離開螢幕。
- 將手指放回螢幕，並執行縮放手勢。

3 Windows 7 (或其他作業系統)，如果使用者的手勢非常快，旋轉和縮放手勢不一定會產生「更新」階段。

4 Windows 7 Starter Edition 不支援多點觸控。如需詳細資訊，請參閱 AIR Labs Forum：
http://forums.adobe.com/thread/579180?tstart=0

5 如果是 Mac OS 10.5.3 以及更新的版本，則 Multitouch.supportsGestureEvents 值一律會是 true，即使硬體不支援手勢事件也是一樣。

第 33 章 複製和貼上

Flash Player 10 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用剪貼簿 API 中的類別，將資訊複製到系統剪貼簿及從中複製資訊。在 Adobe® Flash® Player 或 Adobe® AIR® 中執行的應用程式支援或可以轉換的資料格式包括：

- 文字
- HTML 格式化文字
- RTF 格式資料
- 經過序列化的物件
- 物件參考 (只有在原始應用程式內才有效)
- 點陣圖 (僅限 AIR)
- 檔案 (僅限 AIR)
- URL 字串 (僅限 AIR)

複製及貼上基本概念

Flash Player 10 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

複製和貼上 API 包含下列類別：

套件	類別
flash.desktop	• Clipboard • ClipboardFormats • ClipboardTransferMode

靜態屬性 Clipboard.generalClipboard 代表作業系統剪貼簿。Clipboard 類別提供了可在剪貼簿物件上讀寫資料的方法。

針對一般複製和貼上鍵盤快速鍵，HTMLLoader 類別 (在 AIR 中) 和 TextField 類別都會實作預設行為。自訂組件若要實作複製和貼上方面的快速鍵行為，可以直接偵聽這些按鍵動作。您也可以使用原生選單命令搭配各按鍵的對等用法，間接回應按鍵動作。

若將同一份資訊的不同表示形式填入單一 Clipboard 物件，即可使其它應用程式便於理解與取用該份資料。例如，填入的影像可以同時包括影像資料、經過序列化的 Bitmap 物件和檔案等形式。任一格式的資料皆可延遲顯示，直到要讀取該種格式的資料時再實際建立所需格式。

讀取與寫入系統剪貼簿

Flash Player 10 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要讀取作業系統剪貼簿，請呼叫 Clipboard.generalClipboard 物件的 getData() 方法，並且傳入所要讀取的格式名稱：

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

if(Clipboard.generalClipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
    var text:String = Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT);
}
```

備註：在 Flash Player 或 AIR 的非應用程式安全執行程序中執行的內容，只能在 paste 事件的事件處理常式內呼叫 getData() 方法。也就是說，只有在 AIR 應用程式安全執行程序中的程式碼，才能在 paste 事件處理常式以外的地方呼叫 getData() 方法。

若要寫入剪貼簿，請將一種或多種格式的資料加入至 Clipboard.generalClipboard 物件。任何現有資料若為相同的格式，都將自動遭到覆寫。不過，在寫入新的資料前最好也清除系統剪貼簿，以確保一併刪除任何不相干的其它格式資料。

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

var textToCopy:String = "Copy to clipboard.";
Clipboard.generalClipboard.clear();
Clipboard.generalClipboard.setData(ClipboardFormats.TEXT_FORMAT, textToCopy, false);
```

備註：在 Flash Player 中或 AIR 的非應用程式安全執行程序中執行的內容，只能在使用者事件 (例如鍵盤或滑鼠事件，或者 copy 或 cut 事件) 的事件處理常式內呼叫 setData() 方法。也就是說，只有在 AIR 應用程式安全執行程序中的程式碼，才能在使用者事件處理常式以外的地方呼叫 setData() 方法。

在 AIR 中 HTML 的複製和貼上

Adobe AIR 1.0 以及更新的版本

Adobe AIR 的 HTML 環境本身提供了另一組複製和貼上事件和預設行為。唯有執行於應用程式安全執行程序內的程式碼，才能透過 AIR Clipboard.generalClipboard 物件直接存取系統剪貼簿。位於非應用程式安全執行程序內的 JavaScript 程式碼如需存取剪貼簿，可在 HTML 文件中的元素傳送複製或貼上事件時，透過其傳送的 event 物件來進行。

複製和貼上事件包括：copy、cut 和 paste。發生這些事件時所傳送的物件提供了 clipboardData 屬性，可以用來存取剪貼簿。

預設行為

Adobe AIR 1.0 以及更新的版本

根據預設，每當經由鍵盤快速鍵或快顯選單產生複製命令，AIR 便會複製選取的項目加以回應。而在可編輯的區域內，AIR 則會剪下文字以回應剪下命令，或將文字貼至游標位置或選取範圍以回應貼上命令。

若要避免此預設行為，事件處理常式可以呼叫所傳送之 event 物件的 preventDefault() 方法。

使用 event 物件的 clipboardData 屬性

Adobe AIR 1.0 以及更新的版本

由於發生複製或貼上事件而傳送的 event 物件，其 clipboardData 屬性可讓您用來讀取與寫入剪貼簿資料。

若要在處理複製或剪下事件時寫入剪貼簿，請使用 clipboardData 物件的 setData() 方法，並且傳入所要複製的資料和 MIME 類型：

```
function customCopy(event){
    event.clipboardData.setData("text/plain", "A copied string.");
}
```

若要存取貼上的資料，您可以使用 clipboardData 物件的 getData() 方法，並且傳入該資料格式的 MIME 類型。可用的格式會透過 types 屬性取得。

```
function customPaste(event){
    var pastedData = event.clipboardData("text/plain");
}
```

唯有透過 paste 事件所傳送的 event 物件，才能存取 getData() 方法和 types 屬性。

下列範例會說明如何覆寫 HTML 網頁預設的複製和貼上行為。copy 事件處理常式會先將複製的文字設定為斜體，然後再複製到剪貼簿成為 HTML 文字。cut 事件處理常式則先將選取的資料複製到剪貼簿，再將該份資料從文件中移除。paste 處理常式會插入 HTML 格式的剪貼簿內容，並將插入的內容設定為粗體文字樣式。

```
<html>
<head>
    <title>Copy and Paste</title>
    <script language="javascript" type="text/javascript">
        function onCopy(event){
            var selection = window.getSelection();
            event.clipboardData.setData("text/html","<i>" + selection + "</i>");
            event.preventDefault();
        }

        function onCut(event){
             var selection = window.getSelection();
             event.clipboardData.setData("text/html","<i>" + selection + "</i>");
             var range = selection.getRangeAt(0);
             range.extractContents();

            event.preventDefault();
        }

        function onPaste(event){
            var insertion = document.createElement("b");
            insertion.innerHTML = event.clipboardData.getData("text/html");
             var selection = window.getSelection();
             var range = selection.getRangeAt(0);
             range.insertNode(insertion);
            event.preventDefault();
        }
    </script>
</head>
<body onCopy="onCopy(event)"
     onPaste="onPaste(event)"
     onCut="onCut(event)">
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore
veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam
voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur
magni dolores eos qui ratione voluptatem sequi nesciunt.</p>
</body>
</html>
```

Clipboard 資料格式

Flash Player 10 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Clipboard 格式描述了放置在 Clipboard 物件中的資料。Flash Player 或 AIR 會自動在 ActionScript 資料類型和系統剪貼簿格式之間轉譯標準資料格式。此外，您也可以利用應用程式定義的格式，在各個以 ActionScript 撰寫的應用程式之間轉換應用程式物件。

Clipboard 物件可以包含同一份資訊的各種不同格式表示形式。例如，表示 Sprite 的 Clipboard 物件可以包含參考格式 (可在相同應用程式中使用)、經過序列化的格式 (可供 Flash Player 或 AIR 中執行的其它應用程式使用)、點陣圖格式 (可供影像編輯程式使用) 以及可能具有延遲顯示功能以編碼 PNG 檔的檔案清單格式 (用於將此 Sprite 的表示形式複製或拖曳到檔案系統)。

標準資料格式

Flash Player 10 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ClipboardFormats 類別提供了定義標準格式名稱的常數：

常數	說明
TEXT_FORMAT	文字格式資料會和 ActionScript String 類別進行雙向轉譯。
HTML_FORMAT	帶有 HTML 標記的文字。
RICH_TEXT_FORMAT	RTF 格式資料會和 ActionScript ByteArray 類別進行雙向轉譯，但不會解譯或轉譯 RTF 標記。
BITMAP_FORMAT	(僅限 AIR) 具有點陣圖格式的資料會轉譯至 / 自 ActionScript 的 BitmapData 類別。
FILE_LIST_FORMAT	(僅限 AIR) 具有檔案清單格式的資料會轉譯至 / 自 ActionScript 的 File 物件陣列。
URL_FORMAT	(僅限 AIR) 具有 URL 格式的資料會轉譯至 / 自 ActionScript 的 String 類別。

在 AIR 應用程式載入的 HTML 內容中，基於回應 copy、cut 或 paste 事件而複製和貼上資料時，必須改用 MIME 類型來取代 ClipboardFormat 字串。有效的資料 MIME 類型如下：

MIME 類型	說明
文字	"text/plain"
URL	"text/uri-list"
點陣圖	"image/x-vnd.adobe.air.bitmap"
檔案清單	"application/x-vnd.adobe.air.file-list"

備註：HTML 內容中的 RTF 格式資料，無法透過 paste 事件發生時傳送之 event 物件的 clipboardData 屬性取得。

自訂資料格式

Flash Player 10 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用應用程式定義的自訂格式來傳輸物件的參考或是已序列化的副本。參考只有在相同應用程式中才有效。經過序列化的物件則可在各個應用程式之間傳輸，但只能與序列化或還原序列化期間仍有效的物件搭配使用。物件的屬性若為簡單類型或是可以序列化的物件，通常即表示該物件可以序列化。

如果要將經過序列化的物件加入至 Clipboard 物件，請呼叫 Clipboard.setData() 方法，並將 serializable 參數設定為 true。格式名稱可以是某種標準格式，或是由應用程式定義的任意字串。

傳輸模式

Flash Player 10 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用自訂資料格式將物件寫入至剪貼簿時，可以從剪貼簿將物件資料當做參考來讀取，或當做原始物件的序列化副本來讀取。有四種傳輸模式，供您判斷物件傳輸形式是一項參考或已序列化的副本：

傳輸模式	說明
ClipboardTransferModes.ORIGINAL_ONLY	只傳回參考。若無法取得參考，則傳回 null 值。
ClipboardTransferModes.ORIGINAL_PREFFERED	若能取得參考便將其傳回，反之則傳回已序列化的副本。
ClipboardTransferModes.CLONE_ONLY	只傳回已序列化的副本。若無法取得已序列化的副本，則傳回 null 值。
ClipboardTransferModes.CLONE_PREFFERED	若能取得已序列化的副本便將其傳回。反之則傳回參考。

讀取與寫入自訂資料格式

Flash Player 10 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在剪貼簿中寫入物件時，除了在 format 參數開頭保留當做前置詞使用的 air: 或 flash: 以外，您可以使用任何字串。使用代表該格式的相同字串即可讀取物件。下列範例說明如何在剪貼簿中讀取與寫入物件：

```
public function createClipboardObject(object:Object):Clipboard{
    var transfer:Clipboard = Clipboard.generalClipboard;
    transfer.setData("object", object, true);
}
```

若要從剪貼簿物件中擷取已序列化的物件 (在執行放入或貼上作業後)，請使用相同的格式名稱以及 CLONE_ONLY 或 CLONE_PREFFERED 轉換模式。

```
var transfer:Object = clipboard.getData("object", ClipboardTransferMode.CLONE_ONLY);
```

參考一律會加入至 Clipboard 物件。若要 (在拖放或貼上作業之後) 從剪貼簿物件擷取參考，請使用 ORIGINAL_ONLY 或 ORIGINAL_PREFFERED 轉換模式，而不要使用經過序列化的副本：

```
var transferredObject:Object =
    clipboard.getData("object", ClipboardTransferMode.ORIGINAL_ONLY);
```

物件參考只有在 Clipboard 物件源自目前的應用程式時才有效。使用 ORIGINAL_PREFFERED 轉換模式，即可在參考可用時存取參考；而無法使用參考時，則存取經過序列化的副本。

延遲顯示

Flash Player 10 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如果資料格式的建立較耗費運算資源，您可以利用延遲顯示函數視需要提供資料。當放入或貼上作業的接收者要求取得延遲的格式資料時，才會呼叫該函數。

請使用 setDataHandler() 方法，將延遲顯示函數加入 Clipboard 物件。函數必須傳回正確格式的資料。例如，若您呼叫了 setDataHandler(ClipboardFormat.TEXT_FORMAT, writeText)，則 writeText() 函數必須傳回字串。

如果是使用 setData() 方法將相同類型的資料格式加入至 Clipboard 物件，則該資料的優先順序會大於延遲版本 (如此將永遠不會呼叫顯示函數)。即使二度存取同一份剪貼簿資料，也未必會再次呼叫延遲顯示函數。

備註：在 Mac OS X 中，延遲顯示功能僅適用於自訂資料格式。使用標準資料格式時，則會立即呼叫顯示函數。

使用延遲顯示函數貼上文字

Flash Player 10 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列範例會說明如何實作延遲顯示函數。

當使用者按下 「複製」按鈕時，應用程式會清除系統剪貼簿中的資料，確保不會留下任何之前剪貼簿作業所產生的資料。接著，setDataHandler() 方法會將 renderData() 函數設定為剪貼簿輸出器。

當使用者從目的地文字欄位的快顯選單選取 「貼上」命令時，應用程式會存取剪貼簿並設定目的地文字。由於已經使用函數 (而非字串) 設定了剪貼簿中的文字資料格式，因此剪貼簿會呼叫 renderData() 函數。renderData() 函數傳回來源文字的內容，然後再指定給目標文字。

請注意，如果您編輯了來源文字後按 「貼上」按鈕，貼上的文字就會反映編輯結果，即使您是在按下 「複製」按鈕後才進行編輯也一樣。這是因為延遲顯示函數會等到您按 「貼上」按鈕才複製來源文字 (若在實用性質的應用程式中使用延遲顯示函數，您可能需要設法儲存或保護來源文字以免發生此問題)。

Flash 範例

```
package {
    import flash.desktop.Clipboard;
    import flash.desktop.ClipboardFormats;
    import flash.desktop.ClipboardTransferMode;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.text.TextFieldType;
    import flash.events.MouseEvent;
    import flash.events.Event;
    public class DeferredRenderingExample extends Sprite
    {
        private var sourceTextField:TextField;
        private var destination:TextField;
        private var copyText:TextField;
        public function DeferredRenderingExample():void
        {
            sourceTextField = createTextField(10, 10, 380, 90);
            sourceTextField.text = "Neque porro quisquam est qui dolorem "
                + "ipsum quia dolor sit amet, consectetur, adipisci velit.";

            copyText = createTextField(10, 110, 35, 20);
            copyText.htmlText = "<a href='#'>Copy</a>";
            copyText.addEventListener(MouseEvent.CLICK, onCopy);

            destination = createTextField(10, 145, 380, 90);
            destination.addEventListener(Event.PASTE, onPaste);
        }
        private function createTextField(x:Number, y:Number, width:Number,
                        height:Number):TextField
        {
            var newTxt:TextField = new TextField();
            newTxt.x = x;
            newTxt.y = y;
            newTxt.height = height;
            newTxt.width = width;
            newTxt.border = true;
            newTxt.multiline = true;
            newTxt.wordWrap = true;
            newTxt.type = TextFieldType.INPUT;
```

```
            addChild(newTxt);
            return newTxt;
        }
        public function onCopy(event:MouseEvent):void
        {
            Clipboard.generalClipboard.clear();
            Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT,
                            renderData);
        }
        public function onPaste(event:Event):void
        {
            sourceTextField.text =
            Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
        }
        public function renderData():String
        {
            trace("Rendering data");
            var sourceStr:String = sourceTextField.text;
            if (sourceTextField.selectionEndIndex >
                     sourceTextField.selectionBeginIndex)
            {
                return sourceStr.substring(sourceTextField.selectionBeginIndex,
                                    sourceTextField.selectionEndIndex);
            }
            else
            {
                return sourceStr;
            }
        }
    }
}
```

Flex 範例

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" width="326" height="330"
applicationComplete="init()">
    <mx:Script>
    <![CDATA[
    import flash.desktop.Clipboard;
    import flash.desktop.ClipboardFormats;

    public function init():void
    {
        destination.addEventListener("paste", doPaste);
    }

    public function doCopy():void
    {
        Clipboard.generalClipboard.clear();
        Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT, renderData);
    }
    public function doPaste(event:Event):void
    {
        destination.text = Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
    }

    public function renderData():String{
        trace("Rendering data");
        return source.text;
    }
    ]]>
    </mx:Script>
    <mx:Label x="10" y="10" text="Source"/>
    <mx:TextArea id="source" x="10" y="36" width="300" height="100">
        <mx:text>Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci
velit.</mx:text>
    </mx:TextArea>
    <mx:Label x="10" y="181" text="Destination"/>
    <mx:TextArea id="destination"  x="12" y="207" width="300" height="100"/>
    <mx:Button click="doCopy();" x="91" y="156" label="Copy"/>
</mx:Application>
```

第 34 章 加速計輸入

Flash Player 10.1 以及更新的版本，**Adobe AIR 2** 以及更新的版本

Accelerometer 類別會根據裝置的移動感應器偵測到的活動傳送事件。這個資料代表裝置的位置或是沿著三維軸的移動。當裝置移動時，感應器會偵測此移動並回傳裝置的加速座標。Accelerometer 類別提供查詢是否支援加速計的方法，同時也設定加速事件傳送的速率。

加速計軸會標準化成顯示方向，而非裝置的實體方向。當裝置重新調整顯示的方向時，加速計軸也會重新調整方向。因此當使用者以正常直立的檢視位置握住電話時，Y 軸永遠大致垂直，不論將電話旋轉成哪個方向都一樣。例如，如果自動方向為關閉，當瀏覽器中的 SWF 內容為全螢幕模式時，則加速計軸在旋轉裝置時不會調整方向。

更多說明主題

flash.sensors.Accelerometer

flash.events.AccelerometerEvent

檢查加速計支援

使用 Accelerometer.isSupported 屬性測試執行階段環境是否可以使用這個功能：

```
if (Accelerometer.isSupported)
{
    // Set up Accelerometer event listeners and code.
}
```

每個 API 項目列出的執行階段版本，都可以存取 Accelerometer 類別及其成員。不過，執行階段的目前環境會判斷這個功能的可用性。例如，在 Flash Player 10.1 中，您可以使用 Accelerometer 類型屬性來編譯程式碼，但是需要使用 Accelerometer.isSupported 屬性來測試 Accelerometer 功能在使用者裝置上的可用性。如果 Accelerometer.isSupported 在執行階段是 true，則表示 Accelerometer 支援目前存在。

偵測加速計變更

若要使用加速計感應器，請初始化 Accelerometer 物件，並註冊它所傳送的 update 事件。update 事件是 Accelerometer 事件物件。這個事件有四個屬性，而且每個屬性都是數字：

- accelerationX：沿著 x 軸加速，單位為 G。當裝置處於直立狀態時，X 軸的方向是從裝置的左邊到右邊 (裝置的頂端面向上方時，裝置就是處於直立狀態)。如果裝置移向右方，則加速為正值。
- accelerationY：沿著 y 軸加速，單位為 G。當裝置處於直立狀態時，Y 軸的方向是從裝置的底部到上方 (裝置的頂端面向上方時，裝置就是處於直立狀態)。如果裝置向上移到這個軸的相對位置，加速會是正值。
- accelerationZ：沿著 z 軸加速，單位為 G。Z 軸延伸方向與裝置表面垂直。如果您移動裝置，讓裝置的表面指向上方，加速會是正值。如果裝置的表面朝面地面，則加速會是負值。
- timestamp：從執行階段初始化之後，事件時間的毫秒數。

1 g 是標準重力加速，大約 9.8 米 / 秒 ^{2}。

以下是在文字欄位中顯示加速計資料的基本範例：

```
var accl:Accelerometer;
if (Accelerometer.isSupported)
{
    accl = new Accelerometer();
    accl.addEventListener(AccelerometerEvent.UPDATE, updateHandler);
}
else
{
    accTextField.text = "Accelerometer feature not supported";
}
function updateHandler(evt:AccelerometerEvent):void
{
    accTextField.text = "acceleration X: " + evt.accelerationX.toString() + "\n"
            + "acceleration Y: " + evt.accelerationY.toString() + "\n"
            + "acceleration Z: " + evt.accelerationZ.toString()
}
```

若要使用這個範例，請確認建立 accTextField 文字欄位，並先將它新增至顯示清單，再使用這個程式碼。

您可以呼叫 Accelerometer 物件的 setRequestedUpdateInterval() 方法，調整加速計事件的預定時間間隔。這個方法接受一個參數 (interval)，而這個參數是要求的更新間隔 (毫秒)：

```
var accl:Accelerometer;
accl = new Accelerometer();
accl.setRequestedUpdateInterval(1000);
```

加速計更新之間的實際時間可能大於或小於這個值。在更新間隔中所做的任何變更會影響所有註冊的偵聽程式。如果您未呼叫 setRequestedUpdateInterval() 方法，則應用程式會根據裝置的預設間隔來接收更新。

加速計資料會有某種程度的不準確度。您可以使用最近資料的移動平均值讓資料更平順。例如，下列範例納入最近加速計讀數與目前讀數，以取得四捨五入的結果。

```
var accl:Accelerometer;
var rollingX:Number = 0;
var rollingY:Number = 0;
var rollingZ:Number = 0;
const FACTOR:Number = 0.25;

if (Accelerometer.isSupported)
{
    accl = new Accelerometer();
    accl.setRequestedUpdateInterval(200);
    accl.addEventListener(AccelerometerEvent.UPDATE, updateHandler);
}
else
{
    accTextField.text = "Accelerometer feature not supported";
}
function updateHandler(event:AccelerometerEvent):void
{
    accelRollingAvg(event);
    accTextField.text = rollingX + "\n" +  rollingY + "\n" + rollingZ + "\n";
}

function accelRollingAvg(event:AccelerometerEvent):void
{
    rollingX = (event.accelerationX * FACTOR) + (rollingX * (1 - FACTOR));
    rollingY = (event.accelerationY * FACTOR) + (rollingY * (1 - FACTOR));
    rollingZ = (event.accelerationZ * FACTOR) + (rollingZ * (1 - FACTOR));
}
```

不過，只有在加速計更新很小時，才適合使用這個移動平均值。

第 35 章 在 AIR 中拖放

Adobe AIR 1.0 以及更新的版本

您可使用 Adobe® AIR™ 拖放 API 中的類別，支援使用者介面拖放手勢。此處所謂的 「手勢」是指由作業系統和應用程式居間傳達，使用者有意執行複製、移動或連結資訊的動作。「拖出」手勢發生於使用者拖曳物件以移出組件或應用程式時。「拖入」手勢則發生於使用者從組件或應用程式外部拖曳物件以移入其中時。

使用拖放 API，您可以讓使用者在各應用程式之間，以及單一應用程式中的各組件之間拖曳資料。支援的傳輸格式包括：

- 點陣圖
- 檔案
- HTML 格式化文字
- 文字
- RTF 格式資料
- URL
- 檔案承諾
- 經過序列化的物件
- 物件參考 (只有在來源應用程式內才有效)

在 AIR 中拖放的基本概念

Adobe AIR 1.0 以及更新的版本

如需在 AIR 應用程式中使用拖放功能的快速說明與程式碼範例，請到 Adobe Developer Connection 參閱下列快速入門文章：

- 支援拖放以及複製和貼上 (Flex)
- 支援拖放以及複製和貼上 (Flash)

拖放 API 包含下列類別。

套件	類別
flash.desktop	• NativeDragManager • NativeDragOptions • Clipboard • URLFilePromise • IFilePromise 下列類別定義了與拖放 API 搭配使用的常數： • NativeDragActions • ClipboardFormat • ClipboardTransferModes
flash.events	NativeDragEvent

拖放手勢階段

拖放手勢包括三個階段：

起始 使用者按住滑鼠按鈕，同時從組件或組件中的項目進行拖曳，以起始拖放作業。拖曳的項目其來源組件通常是指定做為拖曳初始程式，而且會傳送 nativeDragStart 和 nativeDragComplete 事件。Adobe AIR 應用程式基於回應 mouseDown 或 mouseMove 事件而呼叫 NativeDragManager.doDrag() 方法，以開始拖曳作業。

如果拖曳作業是從 AIR 應用程式外部起始，就不會有傳送 nativeDragStart 或 nativeDragComplete 事件的初始程式物件。

拖曳 使用者按住滑鼠按鈕的同時，將滑鼠游標移至另一個組件、應用程式或桌面上。只要拖曳動作還在進行，初始程式物件就會一直傳送 nativeDragUpdate 事件。(但在 Linux 版 AIR 中不會傳送此事件。) 當使用者將滑鼠移至 AIR 應用程式中可能的放下目標上方時，放下目標就會傳送 nativeDragEnter 事件。事件處理常式可以檢查事件物件，判斷拖曳的資料是否為該目標所接受的格式；如果是，則呼叫 NativeDragManager.acceptDragDrop() 方法而讓使用者能在目標上放下資料。

只要拖曳手勢仍停留在互動式物件上方，該物件將一直傳送 nativeDragOver 事件。當拖曳手勢離開互動式物件時，物件則會傳送 nativeDragExit 事件。

放下 使用者在可行的放下目標上方放開滑鼠。如果此目標是 AIR 應用程式或組件，該目標物件就會傳送 nativeDragDrop 事件。事件處理常式可以透過事件物件來存取傳輸的資料。如果目標位於 AIR 外部，則交由作業系統或另一個應用程式處理放下動作。在這兩種情況下，初始物件會傳送 nativeDragComplete 事件 (若拖曳動作是從 AIR 內部開始)。

拖入手勢和拖出手勢都是由 NativeDragManager 類別所控制。NativeDragManager 類別的所有成員均為靜態，因而請勿建立此類別的實體。

Clipboard 物件

經由拖曳而進出應用程式或組件的資料是包含在 Clipboard 物件內。單一 Clipboard 物件可以提供同一份資訊的不同表示形式，從而使其它應用程式便於理解與取用該份資料。例如，提供的影像可同時包括影像資料、經過序列化的 Bitmap 物件和檔案等形式。等到要讀取特定格式的資料時再呼叫顯示函數，便能達到延遲顯示資料的目的。

一旦拖曳手勢開始，要存取 Clipboard 物件就只能從 nativeDragEnter、nativeDragOver 和 nativeDragDrop 事件的事件處理常式中著手。在拖曳手勢結束後，即無法讀取或重複使用 Clipboard 物件。

應用程式物件能以參考形式及經過序列化的物件形式傳輸。參考形式只有在來源應用程式內才有效。經過序列化的物件則可在各個 AIR 應用程式之間傳輸，但只能與序列化或還原序列化期間仍為有效的物件搭配使用。經過序列化的物件會轉換成以字串為基礎的資料傳輸格式，亦即 Action Message Format for ActionScript 3 (AMF3)。

使用 **Flex** 架構

在大部分的情況下，建置 Flex 應用程式時最好使用 Adobe® Flex™ 拖放 API。Flex 架構針對執行於 AIR 中的 Flex 應用程式提供了相應的功能集 (此架構內部即是使用 AIR NativeDragManager)。Flex 也保有一套限制較多的功能集，以因應在更嚴謹的瀏覽器環境下執行應用程式或組件之需要。您無法在執行於 AIR 執行階段環境外的組件或應用程式中使用 AIR 類別。

支援拖出手勢

Adobe AIR 1.0 以及更新的版本

為了支援拖出手勢，您必須建立 Clipboard 物件以回應 mouseDown 事件，再將該物件傳送給 NativeDragManager.doDrag() 方法。接著，您的應用程式即可偵聽初始物件的 nativeDragComplete 事件，以決定一旦使用者完成或放棄手勢時要採取何種行動。

準備即將傳輸的資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

為了準備所要拖曳的資料或物件，請建立 Clipboard 物件並在其中加入資訊 (包括一種或多種即將傳輸的格式)。使用標準資料格式便能傳遞任何可自動轉譯成原生剪貼簿格式的資料，使用應用程式定義的格式則可傳遞物件。

如果將資訊轉換成以特定格式傳輸較耗費運算資源，您可以提供執行轉換的處理常式函數名稱。唯有當接收端組件或應用程式讀取相關聯的格式時，才會呼叫該函數。

如需有關剪貼簿格式的詳細資訊，請參閱第 514 頁「Clipboard 資料格式」。

下列範例說明如何建立 Clipboard 物件以包含單一點陣圖的多種格式：Bitmap 物件、原生點陣圖格式，以及列有點陣圖載入來源檔案的檔案清單格式：

```
import flash.desktop.Clipboard;
import flash.display.Bitmap;
import flash.filesystem.File;
public function createClipboard(image:Bitmap, sourceFile:File):Clipboard{
    var transfer:Clipboard = new Clipboard();
    transfer.setData("CUSTOM_BITMAP", image, true); //Flash object by value and by reference
    transfer.setData(ClipboardFormats.BITMAP_FORMAT, image.bitmapData, false);
    transfer.setData(ClipboardFormats.FILE_LIST_FORMAT, new Array(sourceFile), false);
    return transfer;
}
```

開始拖出作業

Adobe AIR 1.0 以及更新的版本

若要開始拖曳作業，請呼叫 NativeDragManager.doDrag() 方法以回應滑鼠按下事件。doDrag() 方法是靜態方法，會用到下列參數：

參數	說明
initiator	拖曳動作的起源物件，此物件會傳送 dragStart 和 dragComplete 事件。初始程式 initiator 必須是互動式物件。
clipboard	包含要傳輸之資料的 Clipboard 物件。Clipboard 物件是在拖放序列期間所傳送的 NativeDragEvent 物件中參考。

參數	說明
dragImage	(選擇性) 在拖曳期間顯示的 BitmapData 物件。此影像可指定 alpha 值 (注意：Microsoft Windows 一律會對拖曳影像套用固定 Alpha 值淡化效果)。
offset	(選擇性) 指定拖曳影像與滑鼠感應區之間偏移值的 Point 物件。使用負值座標可將拖曳影像往左上方移至相對於滑鼠游標的偏移位置。若未提供 offset 參數，則會將拖曳影像左上角置於滑鼠感應區。
actionsAllowed	(選擇性) 指定拖曳作業可行之動作 (複製、移動或連結) 的 NativeDragOptions 物件。若未提供此引數，則允許所有動作。DragOptions 物件是在 NativeDragEvent 物件中參考，以使可能的拖曳目標能夠檢查所允許的動作是否相容於目標組件的用途。例如，「垃圾桶」組件大致只接受允許移動動作的拖曳手勢。

下列範例說明如何對載入自檔案的點陣圖物件開始拖曳作業。此範例會載入影像，並於 mouseDown 事件發生時開始拖曳作業。

```
 package
{
import flash.desktop.NativeDragManager;
import mx.core.UIComponent;
import flash.display.Sprite;
import flash.display.Loader;
import flash.system.LoaderContext;
import flash.net.URLRequest;
import flash.geom.Point;
import flash.desktop.Clipboard;
import flash.display.Bitmap;
import flash.filesystem.File;
import flash.events.Event;
import flash.events.MouseEvent;

public class DragOutExample extends UIComponent Sprite {
    protected var fileURL:String = "app:/image.jpg";
    protected var display:Bitmap;

    private function init():void {
        loadImage();
    }
    private function onMouseDown(event:MouseEvent):void {
        var bitmapFile:File = new File(fileURL);
        var transferObject:Clipboard = createClipboard(display, bitmapFile);
        NativeDragManager.doDrag(this,
                            transferObject,
                            display.bitmapData,
                            new Point(-mouseX,-mouseY));
    }
    public function createClipboard(image:Bitmap, sourceFile:File):Clipboard {
        var transfer:Clipboard = new Clipboard();
        transfer.setData("bitmap",
                            image,
                            true);
                            // ActionScript 3 Bitmap object by value and by reference
        transfer.setData(ClipboardFormats.BITMAP_FORMAT,
                            image.bitmapData,
                            false);
                            // Standard BitmapData format
        transfer.setData(ClipboardFormats.FILE_LIST_FORMAT,
```

```
                            new Array(sourceFile),
                            false);
                            // Standard file list format
        return transfer;
    }
    private function loadImage():void {
        var url:URLRequest = new URLRequest(fileURL);
        var loader:Loader = new Loader();
        loader.load(url,new LoaderContext());
        loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onLoadComplete);
    }
    private function onLoadComplete(event:Event):void {
        display = event.target.loader.content;
        var flexWrapper:UIComponent = new UIComponent();
        flexWrapper.addChild(event.target.loader.content);
        addChild(flexWrapper);
        flexWrapper.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
    }
}
}
```

完成拖出傳輸

Adobe AIR 1.0 以及更新的版本

當使用者放開滑鼠而將拖曳的項目放下時，初始程式物件便會傳送 nativeDragComplete 事件。您可以檢查此事件物件的 dropAction 屬性，然後採取適當的行動。例如，若其動作為 NativeDragAction.MOVE，您即可將來源項目從原始位置移除。使用者如果要放棄拖曳手勢，可以將游標移至可行的放下目標外面再放開滑鼠按鈕。對於已放棄的手勢，拖曳管理員會將其 dropAction 屬性設定為 NativeDragAction.NONE。

支援拖入手勢

Adobe AIR 1.0 以及更新的版本

為了支援拖入手勢，您的應用程式 (或者，大體上是指應用程式的視覺化組件) 必須回應 nativeDragEnter 或 nativeDragOver 事件。

一般放下作業的步驟

Adobe AIR 1.0 以及更新的版本

放下作業通常會發生下列一連串事件：

1. 使用者將剪貼簿物件拖曳到某個組件上方。
2. 該組件傳送 nativeDragEnter 事件。
3. nativeDragEnter 事件處理常式檢查事件物件，據以判定適用的資料格式和允許的動作。如果組件能夠處置放下動作，便會呼叫 NativeDragManager.acceptDragDrop()。
4. NativeDragManager 將滑鼠游標變更為可放下物件的形狀。
5. 使用者在組件上方放開物件。
6. 接收端組件傳送 nativeDragDrop 事件。

7 接收端組件從事件物件內的 Clipboard 物件中讀取其所需格式的資料。

8 如果拖曳手勢源自 AIR 應用程式內部，初始的互動式物件即傳送 nativeDragComplete 事件。如果手勢源自 AIR 外部，就不會傳送任何回應。

傳達拖入手勢

Adobe AIR 1.0 以及更新的版本

當使用者將剪貼簿項目拖曳至視覺化組件的邊界內時，該組件會傳送 nativeDragEnter 和 nativeDragOver 事件。這些事件的處理常式則可檢查事件物件的 clipboard 和 allowedActions 屬性，以判斷組件能否接受剪貼簿項目。為了傳達訊號表示組件能夠接受放下動作，事件處理常式必須呼叫 NativeDragManager.acceptDragDrop() 方法，同時傳遞接收端組件的參考。若有多個已登錄的事件偵聽程式呼叫了 acceptDragDrop() 方法，則以清單中所列的最後一個處理常式為最優先。只要滑鼠還沒移出接收端物件的邊界，因而未觸發 nativeDragExit 事件，acceptDragDrop() 呼叫就仍然有效。

如果傳遞給 doDrag() 的 allowedActions 參數允許一種以上的動作，使用者可以藉由按住輔助按鍵，指定想要執行哪一種允許的動作。拖曳管理員就會變更游標影像，告訴使用者一旦完成放下動作時將發生何種動作。意圖執行的動作是由 NativeDragEvent 物件的 dropAction 屬性所回報。拖曳手勢的動作設定僅為提供建議。參與傳輸的組件必須實作適當的行為。例如，為了完成移動動作，拖曳初始程式可能會移除拖曳的項目，而放下目標則加入該項目。

拖曳目標可藉由設定 NativeDragManager 類別的 dropAction 屬性，將放下動作限定為三種可能的動作之其中一種。如果使用者利用鍵盤試圖選擇另一種動作，NativeDragManager 便會顯示「不適用」游標形狀。請同時在 nativeDragEnter 和 nativeDragOver 事件的處理常式中設定 dropAction 屬性。

下列範例說明如何實作 nativeDragEnter 或 nativeDragOver 事件的事件處理常式。唯有當拖曳的剪貼簿物件包含文字格式資料時，此處理常式才接受拖入手勢。

```
import flash.desktop.NativeDragManager;
import flash.events.NativeDragEvent;

public function onDragIn(event:NativeDragEvent):void{
    NativeDragManager.dropAction = NativeDragActions.MOVE;
    if(event.clipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
        NativeDragManager.acceptDragDrop(this); //'this' is the receiving component
    }
}
```

完成放下動作

Adobe AIR 1.0 以及更新的版本

當使用者在已接受手勢的互動式物件上放下拖曳的項目時，該互動式物件便會傳送 nativeDragDrop 事件。此事件的處理常式可以從事件物件的 clipboard 屬性擷取資料。

如果剪貼簿包含應用程式定義的格式，則拖曳管理員將傳回物件的參考還是已序列化的版本，取決於傳遞給 Clipboard 物件 getData() 方法的 transferMode 參數。

下列範例說明如何實作 nativeDragDrop 事件的事件處理常式：

```
 import flash.desktop.Clipboard;
import flash.events.NativeDragEvent;

public function onDrop(event:NativeDragEvent):void {
    if (event.clipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)) {
    var text:String =
        String(event.clipboard.getData(ClipboardFormats.TEXT_FORMAT,
                                        ClipboardTransferMode.ORIGINAL_PREFERRED));
}
```

一旦結束事件處理常式，Clipboard 物件將不再有效。任何嘗試存取該物件或其資料的行為都會產生錯誤。

更新組件的視覺外觀

Adobe AIR 1.0 以及更新的版本

組件可以根據 NativeDragEvent 事件來更新視覺外觀。下表說明一般組件基於回應不同的事件所做的變更類型：

事件	說明
nativeDragStart	初始的互動式物件可以利用 nativeDragStart 事件，提供視覺回應指出拖曳手勢係源自該互動式物件。
nativeDragUpdate	初始的互動式物件可以利用 nativeDragUpdate 事件，在手勢期間更新其狀態。(Linux 版 AIR 沒有此事件。)
nativeDragEnter	可能的接收端互動式物件可以利用此事件取得焦點，或以視覺化方式指出其本身能否接受放下動作。
nativeDragOver	可能的接收端互動式物件可以利用此事件，回應滑鼠在該互動式物件內部移動的狀況。例如，指出滑鼠已進入街道地圖畫面之類複雜組件的「熱點」感應區。
nativeDragExit	可能的接收端互動式物件可以利用此事件，在拖曳手勢移出其邊界時還原狀態。
nativeDragComplete	初始的互動式物件可以利用此事件，更新其相關聯的資料模型。例如，從清單中移除項目，同時還原其視覺狀態。

在拖入手勢期間追蹤滑鼠位置

Adobe AIR 1.0 以及更新的版本

當拖曳手勢停留在某個組件上方時，該組件會傳送 nativeDragOver 事件。這些事件每隔幾毫秒將傳送一次，而且也會隨著滑鼠移動而傳送。nativeDragOver 事件物件可以用來判斷滑鼠目前在組件上的位置。若接收端組件相當複雜，但並非由子組件所構成，則存取滑鼠位置或許會有幫助。例如，假設您的應用程式顯示了含有街道地圖的點陣圖，而您想要反白標示使用者已將資訊拖曳至地圖上的哪些區域，即可利用 nativeDragOver 事件所回報的滑鼠座標，追蹤滑鼠在地圖內的位置。

在 HTML 中拖放

Adobe AIR 1.0 以及更新的版本

若要拖曳資料進出 HTML 型應用程式 (或是進出由 HTMLLoader 所顯示的 HTML)，您可以使用 HTML 拖放事件。HTML 拖放 API 可讓您拖曳項目進出 HTML 內容中的 DOM 元素。

備註：您也可以對含有 HTML 內容的 HTMLLoader 物件，透過偵聽其事件的方式來使用 AIR NativeDragEvent 和 NativeDragManager API。不過，HTML API 與 HTML DOM 之間的整合更加完善，讓您能夠控制預設的行為。

預設的拖放行為

Adobe AIR 1.0 以及更新的版本

HTML 環境針對涉及文字、影像和 URL 的拖放手勢提供了預設的行為。使用預設的行為，您隨時都能將這些類型的資料從元素中拖曳出來。不過，您只能將文字拖曳到元素中，而且僅限於頁面上可編輯區域內的元素。當您在頁面的可編輯區域之間或內部拖曳文字時，預設的行為便是執行移動動作。如果您從不可編輯的區域或應用程式外部將文字拖曳到可編輯區域，預設的行為則是執行複製動作。

您可藉由自行處理拖放事件以便覆寫預設的行為。若要取消預設的行為，必須對發生拖放事件時所傳送的物件呼叫 preventDefault() 方法。接著，您即可將資料插入至放下目標，並視需要從拖曳來源移除資料，藉以執行所選的動作。

根據預設，使用者能夠選取並拖曳任何文字，以及拖曳影像和連結。您可以使用 WebKit CSS 屬性 -webkit-user-select 來控制任何 HTML 元素可供選取的方式。例如，當 -webkit-user-select 設定為 none 時，元素內容即為無法選取的狀態，因此也就無法拖曳。您也可以使用 CSS 屬性 -webkit-user-drag 來控制能否拖曳一整個元素。不過，元素的內容視為各自獨立。使用者仍可拖曳已選取的部分文字。如需詳細資訊，請參閱 第 840 頁 「AIR 中的 CSS」。

HTML 中的拖放事件

Adobe AIR 1.0 以及更新的版本

拖曳動作的起源初始程式元素會傳送下列事件：

事件	說明
dragstart	當使用者開始拖曳手勢時即傳送。此事件的處理常式可於必要時呼叫事件物件的 preventDefault() 方法，以制止拖曳動作。若要控制是否允許複製、連結或移動拖曳的資料，請設定 effectAllowed 屬性。預設的行為會將選取的文字、影像和連結放入剪貼簿，但您亦可使用事件物件的 dataTransfer 屬性，為拖曳手勢設定不同的資料。
drag	在拖曳手勢期間不斷傳送。
dragend	當使用者放開滑鼠按鈕以致結束拖曳手勢時即傳送。

拖曳目標會傳送下列事件：

事件	說明
dragover	當拖曳手勢停留在元素邊界內期間不斷傳送。此事件的處理常式應該設定 dataTransfer.dropEffect 屬性，指出一旦使用者放開滑鼠後，放下動作將會造成複製、移動或連結動作。
dragenter	當拖曳手勢進入元素的邊界時即傳送。 如果您在 dragenter 事件處理常式中變更了 dataTransfer 物件的任何屬性，這些變更很快就會被接下來的 dragover 事件所覆寫。另一方面，dragenter 與初次發生的 dragover 事件之間會有短暫的延遲，如果兩者設定了不同的屬性則可能導致游標閃爍。在許多情況下，這兩種事件可以共用同一個事件處理常式。
dragleave	當拖曳手勢移出元素的邊界時即傳送。
drop	當使用者在元素上放下資料時即傳送。唯有透過此事件的處理常式，才能存取到拖曳的資料。

基於回應上述事件而傳送的事件物件與滑鼠事件非常相似。您可以使用滑鼠事件屬性如 (clientX, clientY) 和 (screenX, screenY) 來判斷滑鼠位置。

拖曳事件物件最重要的屬性莫過於 dataTransfer，其中包含了所拖曳的資料。dataTransfer 物件本身具有下列屬性和方法：

屬性或方法	說明
effectAllowed	拖曳動作來源所允許的效果。此值通常是在 dragstart 事件的處理常式中設定。請參閱：第 529 頁 「HTML 中的拖曳效果」。
dropEffect	由目標或使用者所選擇的效果。如果您在 dragover 或 dragenter 事件處理常式中設定 dropEffect，AIR 便會更新滑鼠游標以指出當使用者放開滑鼠時將發生何種效果。如果 dropEffect 設定值不符合任何一種允許的效果，則會顯示 「不適用」游標形狀且不允許放下項目。若您並未設定 dropEffect 以回應最近發生的 dragover 或 dragenter 事件，使用者即可使用作業系統標準輔助按鍵，從各種允許的效果中選擇。 最終效果是由 dragend 發生時所傳送之物件的 dropEffect 屬性所回報。如果使用者在可行的目標外面放開滑鼠表示放棄放下項目，dropEffect 即設定為 none。
types	包含 MIME 類型字串的陣列，各字串代表存在於 dataTransfer 物件中的每一種資料格式。
getData(mimeType)	取得由 mimeType 參數所指定之格式的資料。 只有在回應 drop 事件時，才能呼叫 getData() 方法。

屬性或方法	說明
setData(mimeType)	將 mimeType 參數所指定之格式的資料加入 dataTransfer 中。您可以呼叫 setData() 並指定每一種 MIME 類型，藉此加入多種格式的資料。經由預設的拖曳行為放入 dataTransfer 物件中的任何資料都將遭到清除。 只有在回應 dragstart 事件時，才能呼叫 setData() 方法。
clearData(mimeType)	清除由 mimeType 參數所指定之格式的任何資料。
setDragImage(image, offsetX, offsetY)	設定自訂的拖曳影像。唯有回應 dragstart 事件，以及因為 -webkit-user-drag CSS 樣式設定為 element 而拖曳整個 HTML 元素時，才會呼叫 setDragImage() 方法。image 參數可以是 JavaScript Element 或 Image 物件。

HTML 拖放的 MIME 類型

Adobe AIR 1.0 以及更新的版本

與 HTML 拖放事件的 dataTransfer 物件搭配使用的 MIME 類型包括：

資料格式	MIME 類型
文字	"text/plain"
HTML	"text/html"
URL	"text/uri-list"
點陣圖	"image/x-vnd.adobe.air.bitmap"
檔案清單	"application/x-vnd.adobe.air.file-list"

您也可以使用其它的 MIME 字串，包括應用程式定義的字串在內。不過，其它應用程式可能無法辨識或使用以這類格式傳輸的資料。您必須負責將預期的格式資料加入 dataTransfer 物件中。

重要事項：唯有執行於應用程式安全執行程序內的程式碼才能存取放下的檔案。在非應用程式安全執行程序內，嘗試讀取或設定 File 物件的任何屬性都將產生安全性錯誤。如需詳細資訊，請參閱第 533 頁「在非應用程式的 HTML 安全執行程序內處理檔案放下」。

HTML 中的拖曳效果

Adobe AIR 1.0 以及更新的版本

拖曳手勢的初始程式可以在 dragstart 事件的處理常式中設定 dataTransfer.effectAllowed 屬性，藉此限制所允許的拖曳效果。可用的字串值如下所示：

字串值	說明
"none"	不允許拖曳作業。
"copy"	資料將會複製到目的端，而原始資料則仍留在原位。
"link"	透過連結回原始資料的方式，讓放下目標與之共用該份資料。
"move"	資料將會複製到目的地，並且從原始位置移除。
"copyLink"	資料可供複製或連結。

字串值	說明
"copyMove"	資料可供複製或移動。
"linkMove"	資料可供連結或移動。
"all"	資料可供複製、移動或連結。若已制止預設的行為，預設效果就是 All。

拖曳手勢的目標可以設定 dataTransfer.dropEffect 屬性，指出將在使用者完成放下動作時採取何種動作。如果放下效果屬於任何一種允許的動作，系統就會顯示相應的複製、移動或連結游標形狀。否則，系統將顯示 「不適用」游標形狀。如果該目標並未設定任何放下效果，使用者即可利用輔助按鍵從各種允許的效果中選擇。

請一併在 dragover 和 dragenter 事件的處理常式中設定 dropEffect 值：

```
function doDragStart(event) {
    event.dataTransfer.setData("text/plain","Text to drag");
    event.dataTransfer.effectAllowed = "copyMove";
}

function doDragOver(event) {
    event.dataTransfer.dropEffect = "copy";
}

function doDragEnter(event) {
    event.dataTransfer.dropEffect = "copy";
}
```

備註：儘管只有 dragenter 的處理常式始終需要設定 dropEffect 屬性，但要留意的是，接著發生的 dragover 事件會將這個屬性重設為預設值。因此，回應這兩種事件時都該設定 dropEffect。

將 HTML 元素中的資料拖曳至目標

Adobe AIR 1.0 以及更新的版本

預設的行為讓您能夠經由拖曳方式，複製 HTML 網頁中大部分的內容。您可以使用 CSS 屬性 -webkit-user-select 和 -webkit-user-drag 來控制允許拖曳的內容。

請在 dragstart 事件的處理常式中覆寫預設的拖出行為。只要對事件物件的 dataTransfer 屬性呼叫 setData() 方法，即可將您自己的資料放入拖曳手勢。

在您並未依賴預設行為的情況下，如需指定來源物件支援的拖曳效果，請對 dragstart 事件發生時所傳送的事件物件設定 dataTransfer.effectAllowed 屬性。您可以選擇各種效果的任意組合。例如，若來源元素同時支援 「複製」和 「連結」效果，請將該屬性設定為 "copyLink"。

設定拖曳的資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 dragstart 事件的處理常式中，使用 dataTransfer 屬性來加入拖曳手勢所傳輸的資料。使用 dataTransfer.setData() 方法並且傳入 MIME 類型以及要傳輸的資料，將資料放入剪貼簿。

例如，假設您的應用程式中有一個 ID 為 imageOfGeorge 的影像元素，就可以使用下列 dragstart 事件處理常式。此範例加入了 George 照片的多種資料格式表示形式，從而使其它應用程式便於取用拖曳的資料。

```
function dragStartHandler(event){
    event.dataTransfer.effectAllowed = "copy";

    var dragImage = document.getElementById("imageOfGeorge");
    var dragFile = new air.File(dragImage.src);
    event.dataTransfer.setData("text/plain","A picture of George");
    event.dataTransfer.setData("image/x-vnd.adobe.air.bitmap", dragImage);
    event.dataTransfer.setData("application/x-vnd.adobe.air.file-list",
                                new Array(dragFile));
}
```

備註：當您呼叫 dataTransfer 物件的 setData() 方法時，預設的拖放行為不會加入任何資料。

將資料拖曳到 HTML 元素中

Adobe AIR 1.0 以及更新的版本

預設的行為只允許將文字拖曳到頁面的可編輯區域。您可藉由在元素的開頭標籤中加入 contenteditable 特質，以指定讓該元素及其子元素成為可編輯的項目。您也可以將文件物件的 designMode 屬性設定為 "on"，使整份文件成為可編輯。

針對任何能夠接受拖曳的資料之頁面元素，您可以處理其 dragenter、dragover 和 drop 事件，如此便能支援替代的拖入行為。

啟用拖入

Adobe AIR 1.0 以及更新的版本

若要處理拖入手勢，您必須先取消預設的行為。對於任何您想要用來當做放下目標的 HTML 元素，偵聽其 dragenter 和 dragover 事件。在這些事件的處理常式中，呼叫所傳送之事件物件的 preventDefault() 方法。取消預設的行為後，不可編輯的區域就能接受放下動作。

取得放下的資料

Adobe AIR 1.0 以及更新的版本

您可以在 ondrop 事件的處理常式中存取放下的資料：

```
function doDrop(event){
    droppedText = event.dataTransfer.getData("text/plain");
}
```

使用 dataTransfer.getData() 方法並且傳入所要讀取資料格式的 MIME 類型，將資料讀入剪貼簿。您可以使用 dataTransfer 物件的 types 屬性，找出可用的資料格式為何。types 陣列中包含每一種可用格式的 MIME 類型字串。

一旦您在 dragenter 或 dragover 事件中取消了預設的行為，就必須自行將任何放下的資料插入文件中適當的位置。目前並無任何 API 可將元素內的滑鼠位置轉換成插入點。這項限制導致插入類型的拖曳手勢變得較難實作。

範例：覆寫預設的 HTML 拖入行為

Adobe AIR 1.0 以及更新的版本

此範例所實作的放下目標會顯示一份表格，以列出放下的項目所提供的每一種資料格式。

預設的行為原本是允許在應用程式內拖曳文字、連結和影像。此範例會覆寫預設的拖入行為，覆寫對象則會做為放下目標的 div 元素。為了讓不可編輯的內容能夠接受拖入手勢，主要步驟在於同時對 dragenter 和 dragover 事件發生時所傳送的事件物件呼叫 preventDefault() 方法。回應 drop 事件時，處理常式會將傳輸的資料轉換成 HTML 列元素，然後再將該列插入表格中以供顯示。

```
<html>
<head>
<title>Drag-and-drop</title>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
<script language="javascript">
    function init(){
        var target = document.getElementById('target');
        target.addEventListener("dragenter", dragEnterOverHandler);
        target.addEventListener("dragover", dragEnterOverHandler);
        target.addEventListener("drop", dropHandler);

        var source = document.getElementById('source');
        source.addEventListener("dragstart", dragStartHandler);
        source.addEventListener("dragend", dragEndHandler);

        emptyRow = document.getElementById("emptyTargetRow");
    }

    function dragStartHandler(event){
        event.dataTransfer.effectAllowed = "copy";
    }

    function dragEndHandler(event){
        air.trace(event.type + ": " + event.dataTransfer.dropEffect);
    }

    function dragEnterOverHandler(event){
        event.preventDefault();
    }

    var emptyRow;
    function dropHandler(event){
        for(var prop in event){
            air.trace(prop + " = " + event[prop]);
        }
        var row = document.createElement('tr');
        row.innerHTML = "<td>" + event.dataTransfer.getData("text/plain") + "</td>" +
            "<td>" + event.dataTransfer.getData("text/html") + "</td>" +
            "<td>" + event.dataTransfer.getData("text/uri-list") + "</td>" +
            "<td>" + event.dataTransfer.getData("application/x-vnd.adobe.air.file-list") +
            "</td>";

        var imageCell = document.createElement('td');
        if((event.dataTransfer.types.toString()).search("image/x-vnd.adobe.air.bitmap") > -1){
            imageCell.appendChild(event.dataTransfer.getData("image/x-vnd.adobe.air.bitmap"));
        }
        row.appendChild(imageCell);
        var parent = emptyRow.parentNode;
        parent.insertBefore(row, emptyRow);
    }
</script>
</head>
<body onLoad="init()" style="padding:5px">
<div>
    <h1>Source</h1>
       <p>Items to drag:</p>
    <ul id="source">
```

```
        <li>Plain text.</li>
           <li>HTML <b>formatted</b> text.</li>
           <li>A <a href="http://www.adobe.com">URL.</a></li>
           <li><img src="icons/AIRApp_16.png" alt="An image"/></li>
           <li style="-webkit-user-drag:none;">
        Uses "-webkit-user-drag:none" style.
           </li>
           <li style="-webkit-user-select:none;">
        Uses "-webkit-user-select:none" style.
           </li>

       </ul>
</div>
<div id="target" style="border-style:dashed;">
    <h1 >Target</h1>
       <p>Drag items from the source list (or elsewhere).</p>
       <table id="displayTable" border="1">
        <tr><th>Plain text</th><th>Html text</th><th>URL</th><th>File list</th><th>Bitmap Data</th></tr>
           <tr
id="emptyTargetRow"><td> </td><td> </td><td> </td><td> </td><td> </td></tr>
       </table>
       </div>
</div>
</body>
</html>
```

在非應用程式的 HTML 安全執行程序內處理檔案放下

Adobe AIR 1.0 以及更新的版本

非應用程式的內容無法存取因拖曳檔案到 AIR 應用程式中而產生的 File 物件，也無法透過安全執行程序橋接將這些 File 物件中的任何一個傳遞至應用程式內容 (存取物件屬性必須在序列化期間進行)。不過，藉由偵聽 HTMLLoader 物件的 AIR nativeDragDrop 事件，您還是可以在應用程式中放下檔案。

一般而言，如果使用者在裝載非應用程式內容的頁框中放下檔案，子系並不會將 drop 事件傳給父系。不過，由於 HTMLLoader (此為 AIR 應用程式中所有 HTML 內容的容器) 傳送的事件不屬於 HTML 事件流程的一部分，您仍然可以在應用程式內容中接收 drop 事件。

為了接收檔案放下事件，父文件可使用 window.htmlLoader 所提供的參考，將事件偵聽程式加入 HTMLLoader 物件：

```
window.htmlLoader.addEventListener("nativeDragDrop",function(event){
    var filelist = event.clipboard.getData(air.ClipboardFormats.FILE_LIST_FORMAT);
    air.trace(filelist[0].url);
});
```

下列範例使用父文件將子頁面載入至遠端安全執行程序 (http://localhost/)。父文件會偵聽 HTMLLoader 物件的 nativeDragDrop 事件以追蹤檔案 URL。

```
<html>
<head>
<title>Drag-and-drop in a remote sandbox</title>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
<script language="javascript">
    window.htmlLoader.addEventListener("nativeDragDrop",function(event){
        var filelist = event.clipboard.getData(air.ClipboardFormats.FILE_LIST_FORMAT);
        air.trace(filelist[0].url);
    });
</script>
</head>
<body>
       <iframe src="child.html"
               sandboxRoot="http://localhost/"
               documentRoot="app:/"
               frameBorder="0"  width="100%" height="100%">
       </iframe>
</body>
</html>
```

子文件必須成為有效的放下目標，因此應在 HTML dragenter 和 dragover 事件處理常式中呼叫 Event 物件 preventDefault() 方法。否則不會發生放下事件。

```
<html>
<head>
    <title>Drag and drop target</title>
    <script language="javascript" type="text/javascript">
        function preventDefault(event){
            event.preventDefault();
        }
    </script>
</head>
<body ondragenter="preventDefault(event)" ondragover="preventDefault(event)">
<div>
<h1>Drop Files Here</h1>
</div>
</body>
</html>
```

放下檔案承諾

Adobe AIR 2 以及更新的版本

檔案承諾是一種拖放剪貼簿格式，可以讓使用者將尚不存在的檔案拖離 AIR 應用程式。例如，只要利用檔案承諾，您的應用程式就可以允許使用者將 proxy 圖示拖曳至桌面資料夾。proxy 圖示代表已知即將可從 URL 取得使用的檔案或某些資料。使用者拖曳圖示後，執行階段會下載資料並將檔案寫入拖放的位置。

您可以在 AIR 應用程式中使用 URLFilePromise 類別，拖放在 URL 可以存取的檔案。在 aircore 程式庫中提供 URLFilePromise 實作，是 AIR 2 SDK 的一部分。請使用 SDK frameworks/libs/air 目錄中的 aircore.swc 或 aircore.swf 檔案。

您也可以使用 IFilePromise 介面 (定義於執行階段 flash.desktop 套件) 實作自己的檔案承諾邏輯。

檔案承諾的概念類似於在剪貼簿使用資料處理常式函數來延遲顯示。拖放檔案時請使用檔案承諾而不是延遲顯示。產生或下載資料時，這種延遲顯示技術會在拖曳手勢造成不想要的停頓。請將延遲顯示用於複製及貼上作業 (因應不支援檔案承諾的狀態)。

檔案承諾使用限制

與在拖放剪貼簿中可以放入的其他資料格式相比，檔案承諾存在以下的限制：

- 檔案承諾只能拖離 AIR 應用程式；它們無法拖曳至 AIR 應用程式中。
- 並非所有作業系統皆支援檔案承諾。請使用 Clipboard.supportsFilePromise 屬性測試主機系統是否支援檔案承諾。在不支援檔案承諾的系統上，您應該提供一種替代機制，以便用於下載或產生檔案資料。
- 檔案承諾不可以用於複製及貼上剪貼簿 (Clipboard.generalClipboard)。

更多說明主題

flash.desktop.IFilePromise

air.desktop.URLFilePromise

投放遠端檔案

Adobe AIR 2 以及更新的版本

使用 URLFilePromise 類別來建立檔案承諾物件，用於代表可以在 URL 取得的檔案或資料。使用 FILE_PROMISE_LIST 剪貼簿格式，將一或多個檔案承諾新增至剪貼簿。在以下範例中，會到 http://www.example.com/foo.txt 下載一個檔案，然後儲存至投放位置，變成 bar.txt。(遠端和本機檔案名稱不必完全一樣)。

```
if( Clipboard.supportsFilePromise )
{
    var filePromise:URLFilePromise = new URLFilePromise();
    filePromise.request = new URLRequest("http://example.com/foo.txt");
    filePromise.relativePath = "bar.txt";

    var fileList:Array = new Array( filePromise );
    var clipboard:Clipboard = new Clipboard();
    clipboard.setData( ClipboardFormats.FILE_PROMISE_LIST_FORMAT, fileList );
    NativeDragManager.doDrag( dragSource, clipboard );
}
```

將更多的檔案承諾物件新增至指定給剪貼簿的陣列，如此便可以允許使用者一次拖曳一個以上的檔案。您也可以在 relativePath 屬性中指定子目錄，這樣作業中的部分或所有檔案便會被移至與投放位置相對的子目錄中。

以下範例說明如何進行一項包含多個檔案承諾的拖曳作業。在這個範例中，article.html 這個 html 網頁會以檔案承諾放在剪貼簿中，包含兩個連結的影像檔。這兩個影像檔會複製到 images 子資料夾中，這樣便可以保有相對的連結。

```
if( Clipboard.supportsFilePromise )
{   //Create the promise objects
    var filePromise:URLFilePromise = new URLFilePromise();
    filePromise.request = new URLRequest("http://example.com/article.html");
    filePromise.relativePath = "article.html";

    var image1Promise:URLFilePromise = new URLFilePromise();
    image1Promise.request = new URLRequest("http://example.com/images/img_1.jpg");
    image1Promise.relativePath = "images/img_1.html";
    var image2Promise:URLFilePromise = new URLFilePromise();
    image2Promise.request = new URLRequest("http://example.com/images/img_2.jpg");
    image2Promise.relativePath = "images/img_2.jpg";


    //Put the promise objects onto the clipboard inside an array
    var fileList:Array = new Array( filePromise, image1Promise, image2Promise );
    var clipboard:Clipboard = new Clipboard();
    clipboard.setData( ClipboardFormats.FILE_PROMISE_LIST_FORMAT, fileList );
    //Start the drag operation
    NativeDragManager.doDrag( dragSource, clipboard );
}
```

實作 IFilePromise 介面

Adobe AIR 2 以及更新的版本

若要為無法使用 URLFilePromise 物件存取的資源提供檔案承諾，您可以在自訂類別中實作 IFilePromise 介面。IFilePromise 介面會定義 AIR 執行階段使用的方法和屬性，用於存取一旦檔案承諾投入之後，會被寫入檔案中的資料。

IFilePromise 實作會將另一個物件傳送至可以提供資料給檔案承諾的 AIR 執行階段。這個物件必須實作 IDataInput 介面，AIR 執行階段會透過這個介面去讀取資料。例如，URLFilePromise 類別 (它會實作 IFilePromise) 會使用 URLStream 物件當做資料提供者。

AIR 可以同步或非同步讀取資料。IFilePromise 實作會在 isAsync 屬性傳回適當的值，報告支援的存取模式。如果提供非同步資料存取，資料提供者物件必須實作 IEventDispatcher 介面以及傳送必要的事件，例如 open、progress 以及 complete。

您可以使用自訂類別，或者以下其中一個內建的類別，當做檔案承諾的資料提供者。

- ByteArray (同步)
- FileStream (同步或非同步)
- Socket (非同步)
- URLStream (非同步)

若要實作 IFilePromise 介面，您必須為以下的函數和屬性提供程式碼：

- open():IDataInput — 傳回資料提供者物件 (也就是從此處讀取檔案承諾所需的資料)。物件必須實作 IDataInput 介面。若非同步提供資料，物件也必須實作 IEventDispatcher 介面以及傳送所需的事件 (請參閱第 538 頁「在檔案承諾中使用非同步資料提供者」)。
- get relativePath():String — 提供路徑，其中包括建立之檔案的檔案名稱。路徑會以拖放作業中使用者所選擇的投放位置為基準進行解析。若要確定路徑會針對主機作業系統使用適當的分隔符號字元，請在指定內含目錄的路徑時，使用 File.separator 常數。您可以新增 setter 函數或者使用建構函式參數，允許在執行階段設定路徑。
- get isAsync():Boolean — 將資料提供者物件是否同步或非同步提供它的資料這件事，告知 AIR 執行階段。
- close():void — 當資料完全讀取時由執行階段呼叫 (或者發生錯誤禁止進一步讀取)。您可以將這個函數用於清除資源。
- reportError(e:ErrorEvent):void — 當讀取資料發生錯誤時，由執行階段呼叫。

在涉及檔案承諾的拖放作業時，所有的 IFilePromise 方法都是由執行階段呼叫的。一般情況下，您的應用程式邏輯不應該直接呼叫任何這些方法。

在檔案承諾中使用同步資料提供者

Adobe AIR 2 以及更新的版本

實作 IFilePromise 介面最簡單的方法就是使用同步資料提供者物件，例如 ByteArray 或同步的 FileStream。在以下範例中，會建立 ByteArray 物件、填入資料並在呼叫 open() 方法時傳回。

```
package
{
    import flash.desktop.IFilePromise;
    import flash.events.ErrorEvent;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class SynchronousFilePromise implements IFilePromise
    {
        private const fileSize:int = 5000; //size of file data
        private var filePath:String = "SynchronousFile.txt";

        public function get relativePath():String
        {
            return filePath;
        }

        public function get isAsync():Boolean
        {
            return false;
        }

        public function open():IDataInput
        {
            var fileContents:ByteArray = new ByteArray();

            //Create some arbitrary data for the file
            for( var i:int = 0; i < fileSize; i++ )
            {
                fileContents.writeUTFBytes( 'S' );
            }

            //Important: the ByteArray is read from the current position
            fileContents.position = 0;
            return fileContents;
        }

        public function close():void
        {
            //Nothing needs to be closed in this case.
        }

        public function reportError(e:ErrorEvent):void
        {
            trace("Something went wrong: " + e.errorID + " - " + e.type + ", " + e.text );
        }
    }
}
```

實際應用方面，同步檔案承諾應用面不廣。若資料量很少，只要在暫存目錄中建立檔案，然後將一般檔案清單陣列新增至拖放剪貼簿之中即可。在另一方面，若資料量很大或者產生資料時需要進行大量的計算，則需要長時間執行同步程序。長時間執行同步程序會封鎖 UI 更新而且封鎖時間相當長，讓您的應用程式看起來似乎沒有反應。若要避免這種問題，您可以建立一個由計時器驅動的非同步資料提供者。

在檔案承諾中使用非同步資料提供者

Adobe AIR 2 以及更新的版本

當您使用非同步資料提供者物件時，IFilePromise isAsync 屬性必須是 true 而且 open() 方法傳回的物件必須實作 IEventDispatcher 介面。執行階段會偵聽多個替代事件，如此不同的內部物件就可以當做是資料提供者。例如，FileStream 和 URLStream 物件會傳送 progress 事件，而 Socket 物件會傳送 socketData 事件。執行階段會從這些所有的物件偵聽正確的事件。

以下事件會產生從資料提供者物件讀取資料的程序：

- Event.OPEN — 通知執行階段資料來源已經就緒。
- ProgressEvent.PROGRESS — 通知執行階段資料可以取得。執行階段會從資料提供者物件讀取可用的資料量。
- ProgressEvent.SOCKET_DATA — 通知執行階段資料可以取得。socketData 事件是由通訊端式物件傳送的。至於其他物件類型，您應該傳送一個 progress 事件。(執行階段會偵聽這兩個事件，以便偵測何時可以讀取資料)。
- Event.COMPLETE — 通知執行階段資料已經全部讀取了。
- Event.CLOSE — 通知執行階段資料已經全部讀取了。(執行階段會為了這個目的來偵聽 close 和 complete)。
- IOErrorEvent.IOERROR — 通知執行階段讀取資料時發生錯誤。執行階段會中止建立檔案，然後呼叫 IFilePromise close() 方法。
- SecurityErrorEvent.SECURITY_ERROR — 通知執行階段發生安全性錯誤。執行階段會終止建立檔案，然後呼叫 IFilePromise close() 方法。
- HTTPStatusEvent.HTTP_STATUS — 執行階段會將它與 httpResponseStatus 一起使用，以便確定可用的資料就是想要的內容，而不是錯誤訊息 (例如 404 網頁)。以 HTTP 通訊協定為基礎的物件應傳送這個事件。
- HTTPStatusEvent.HTTP_RESPONSE_STATUS — 執行階段會將它與 httpStatus 一起使用，以便確定可用的資料就是想要的內容。以 HTTP 通訊協定為基礎的物件應傳送這個事件。

資料提供者應該按照以下順序傳送這些事件：

1. open 事件
2. progress 或 socketData 事件
3. complete 或 close 事件

備註：內建的物件、FileStream、Socket 以及 URLStream，會自動傳送正確的事件。

以下範例會使用自訂的非同步資料提供者來建立一個檔案承諾。這種資料提供者類別會擴充 ByteArray (為了支援 IDataInput) 以及實作 IEventDispatcher 介面。在每一個計時器事件中，這個物件會產生一堆資料，然後傳送一個 progress 事件來通知執行階段資料可以取得。產生足夠的資料後，這個物件會傳送一個 complete 事件。

```
package
{
import flash.events.Event;
import flash.events.EventDispatcher;
import flash.events.IEventDispatcher;
import flash.events.ProgressEvent;
import flash.events.TimerEvent;
import flash.utils.ByteArray;
import flash.utils.Timer;

[Event(name="open", type="flash.events.Event.OPEN")]
[Event(name="complete",  type="flash.events.Event.COMPLETE")]
[Event(name="progress", type="flash.events.ProgressEvent")]
[Event(name="ioError", type="flash.events.IOErrorEvent")]
[Event(name="securityError", type="flash.events.SecurityErrorEvent")]
public class AsyncDataProvider extends ByteArray implements IEventDispatcher
{
    private var dispatcher:EventDispatcher = new EventDispatcher();
    public var fileSize:int = 0; //The number of characters in the file
    private const chunkSize:int = 1000; //Amount of data written per event
    private var dispatchDataTimer:Timer = new Timer( 100 );
    private var opened:Boolean = false;

    public function AsyncDataProvider()
    {
        super();
        dispatchDataTimer.addEventListener( TimerEvent.TIMER, generateData );
    }

    public function begin():void{
        dispatchDataTimer.start();
    }

    public function end():void
    {
        dispatchDataTimer.stop();
    }
    private function generateData( event:Event ):void
    {
        if( !opened )
        {
            var open:Event = new Event( Event.OPEN );
            dispatchEvent( open );
            opened = true;
        }
        else if( position + chunkSize < fileSize )
        {
            for( var i:int = 0; i <= chunkSize; i++ )
            {
                writeUTFBytes( 'A' );
            }
            //Set position back to the start of the new data
            this.position -= chunkSize;
            var progress:ProgressEvent =
                new ProgressEvent( ProgressEvent.PROGRESS, false, false, bytesAvailable, bytesAvailable +
chunkSize);
            dispatchEvent( progress )
        }
        else
        {
            var complete:Event = new Event( Event.COMPLETE );
            dispatchEvent( complete );
```

```
        }
    }
     //IEventDispatcher implementation
    public function addEventListener(type:String, listener:Function, useCapture:Boolean=false,
priority:int=0, useWeakReference:Boolean=false):void
    {
        dispatcher.addEventListener( type, listener, useCapture, priority, useWeakReference );
    }

    public function removeEventListener(type:String, listener:Function, useCapture:Boolean=false):void
    {
        dispatcher.removeEventListener( type, listener, useCapture );
    }

    public function dispatchEvent(event:Event):Boolean
    {
        return dispatcher.dispatchEvent( event );
    }

    public function hasEventListener(type:String):Boolean
    {
        return dispatcher.hasEventListener( type );
    }

    public function willTrigger(type:String):Boolean
    {
        return dispatcher.willTrigger( type );
    }
}
}
```

備註：因為本範例中的 AsyncDataProvider 類別擴充 ByteArray，所以它無法也擴充 EventDispatcher。為了實作 IEventDispatcher 介面，這個類別會使用內部的 EventDispatcher 物件，然後將 IEventDispatcher 方法呼叫轉送至該內部物件。您也可以擴充 EventDispatcher 以及實作 IDataInput (或實作這兩個介面)。

非同步 IFilePromise 實作幾乎與同步實作完全相同。主要的差別在於 isAsync 會傳回 true 而 open() 方法會傳回非同步資料物件：

```
package
{
    import flash.desktop.IFilePromise;
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.utils.IDataInput;

    public class AsynchronousFilePromise extends EventDispatcher implements IFilePromise
    {
        private var fileGenerator:AsyncDataProvider;
        private const fileSize:int = 5000; //size of file data
        private var filePath:String = "AsynchronousFile.txt";

        public function get relativePath():String
        {
            return filePath;
        }

        public function get isAsync():Boolean
        {
            return true;
        }
```

```
        public function open():IDataInput
        {
            fileGenerator = new AsyncDataProvider();
            fileGenerator.fileSize = fileSize;
            fileGenerator.begin();
            return fileGenerator;
        }

        public function close():void
        {
            fileGenerator.end();
        }

        public function reportError(e:ErrorEvent):void
        {
            trace("Something went wrong: " + e.errorID + " - " + e.type + ", " + e.text );
        }
    }
}
```

第 36 章 使用選單

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用快顯選單 API 中的類別，修改網頁式 Flex 和 Flash Player 應用程式的快顯選單。

您可以使用原生選單 API 中的類別，定義 Adobe® AIR® 中的應用程式、視窗、快顯選單和彈出式選單。

選單的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如需在 AIR 應用程式中建立原生功能表的快速說明與程式碼範例，請到 Adobe Developer Connection 參閱下列快速入門文章：

- 將原生功能表新增至 AIR 應用程式 (Flex)
- 將原生功能表新增至 AIR 應用程式 (Flash)

原生選單類別可以讓您存取執行應用程式之作業系統的原生選單功能。NativeMenu 物件可以用於應用程式選單 (可在 Mac OS X 中使用)、視窗選單 (可在 Windows 和 Linux 中使用)、快顯選單和彈出式選單。

在 AIR 外部，您可使用快顯選單類別，修改您應用程式中使用者對某物件按下右鍵或 Command+ 按一下時，Flash Player 要自動顯示的快顯選單。(AIR 應用程式不會顯示自動快顯選單。)

選單類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

選單類別包括：

套件	類別
flash.display	• NativeMenu • NativeMenuItem
flash.ui	• ContextMenu • ContextMenuItem
flash.events	• Event • ContextMenuEvent

各種選單

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

AIR 支援下列選單類型：

快顯選單 當使用者以滑鼠右鍵按一下或按 Command+ 按一下 SWF 內容中的互動式物件或 HTML 內容中的 document 元素時，會開啟快顯選單做為回應。

在 Flash Player 執行階段中，選單類別會自動顯示。您可使用 ContextMenu 和 ContextMenuItem 類別，將您自己的命令加入選單。亦可移除部份 (但並非全部) 內建命令。

在 AIR 執行階段中，您可以使用 NativeMenu 或 ContextMenu 類別建立快顯選單。在 AIR 的 HTML 內容中，您可以使用 Webkit HTML 和 JavaScript API，將快顯選單加入 HTML 元素。

應用程式選單 (僅限 AIR) 應用程式選單屬於通用選單，適用於整個應用程式。Mac OS X 支援應用程式選單，Windows 和 Linux 則不支援。在 Mac OS X 中，作業系統會自動建立應用程式選單。您可以使用 AIR 選單 API，將項目和子選單加入至標準選單。您可以加入偵聽程式，以處理現有的選單命令或移除現有的項目。

視窗選單 (僅限 AIR) 視窗選單與單一視窗相關聯，並且會顯示在標題列下方。藉由建立 NativeMenu 物件並將其指定給 NativeWindow 物件的 menu 屬性，即可將選單加入至視窗。Windows 和 Linux 作業系統支援視窗選單，Mac OS X 則不支援。原生選單只能搭配具有系統顏色的視窗使用。

停駐列和系統匣圖示選單 (僅限 AIR) 這些圖示選單與快顯選單類似，並且會指定給 Mac OS X 停駐列或 Windows 和 Linux 工作列通知區域中的應用程式圖示。停駐列和系統匣圖示選單都會使用 NativeMenu 類別。在 Mac OS X 中，加入選單中的項目時，這些項目會位於標準作業系統項目上方；在 Windows 和 Linux 中則沒有標準選單。

彈出式選單 (僅限 AIR) AIR 的彈出式選單與快顯選單類似，但是不一定要和特定應用程式物件或組件相關聯。藉由呼叫 NativeMenu 物件的 display() 方法，即可在視窗中的任意位置顯示彈出式選單。

自訂選單 原生選單全部都由作業系統繪製，因此可以存在於 Flash 和 HTML 顯示模型之外。您一律可使用 MXML、ActionScript 或 JavaScript (於 AIR 中) 建立自訂的非原生選單，不一定要使用原生選單。這類選單一定要完整地顯示在應用程式內容之中。

Flex 選單 Adobe® Flex™ 架構可提供一組 Flex 選單組件。Flex 選單是由執行階段 (而非作業系統) 所繪製，而且不是 「原生」選單。Flex 選單組件可以用於 Flex 視窗，並且沒有系統顏色。使用 Flex 選單組件的另一個優點，就是您可以透過宣告的方式，將選單指定為 MXML 格式。如果您使用 Flex 架構，請使用 Flex 選單類別來建立視窗選單，而不要使用原生選單。

預設選單 (僅限 **AIR)**

下列預設選單會由作業系統或內建的 AIR 類別提供：

- Mac OS X 中的應用程式選單
- Mac OS X 中的停駐列圖示選單
- HTML 內容中所選取之文字和影像的快顯選單
- TextField 物件 (或擴充此 TextField 的物件) 中所選取文字的快顯選單

關於快顯選單

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 SWF 內容中，對於繼承自 InteractiveObject 的任何物件，您都可以藉由將選單物件指定給所繼承物件的 contextMenu 屬性，為所繼承的物件指定快顯選單。選單預設會包含好幾個命令，包括 「快轉」、「後退」、「列印」、「品質」和 「縮放」。在 AIR 執行階段中，指定給 contextMenu 之選單物件的類型，可以是 NativeMenu 或 ContextMenu。在 Flash Player 執行階段中，只能使用 ContextMenu 類別。

當您使用 ContextMenu 和 ContextMenuItem 類別時，就可以偵聽原生選單事件或快顯選單事件；這兩個事件都會傳送。ContextMenuEvent 物件屬性的其中一個優點是，contextMenuOwner 可以指出選單附加至哪一個物件，而 mouseTarget 可指出按一下哪個物件可開啟選單。NativeMenuEvent 物件並不提供此項資訊。

下列範例會建立 Sprite，並加入簡單的編輯快顯選單：

```
 var sprite:Sprite = new Sprite();
sprite.contextMenu = createContextMenu()
private function createContextMenu():ContextMenu{
    var editContextMenu:ContextMenu = new ContextMenu();
    var cutItem:ContextMenuItem = new ContextMenuItem("Cut")
    cutItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doCutCommand);
    editContextMenu.customItems.push(cutItem);

    var copyItem:ContextMenuItem = new ContextMenuItem("Copy")
    copyItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doCopyCommand);
    editContextMenu.customItems.push(copyItem);

    var pasteItem:ContextMenuItem = new ContextMenuItem("Paste")
    pasteItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doPasteCommand);
    editContextMenu.customItems.push(pasteItem);

    return editContextMenu
}
private function doCutCommand(event:ContextMenuEvent):void{trace("cut");}
private function doCopyCommand(event:ContextMenuEvent):void{trace("copy");}
private function doPasteCommand(event:ContextMenuEvent):void{trace("paste");}
```

備註：與在瀏覽器環境中顯示的 SWF 內容不同的是，AIR 中的快顯選單沒有任何內建命令。

自訂 **Flash Player** 快顯選單

在瀏覽器或放映檔中，SWF 內容中的快顯選單都會有內建項目。除了 「設定」和 「關於」命令以外，您可以移除選單中所有預設的命令。將 Stage 屬性 showDefaultContextMenu 設定為 false 可將這些命令從快顯選單中移除。

若要為特定顯示物件建立自訂的快顯選單，請先建立一個新的 ContextMenu 類別實體、呼叫 hideBuiltInItems() 方法，並將該實體指定給該 DisplayObject 實體的 contextMenu 屬性。下列範例將以動態方式繪製一個正方形，並使用快顯選單命令隨機變更它的顏色：

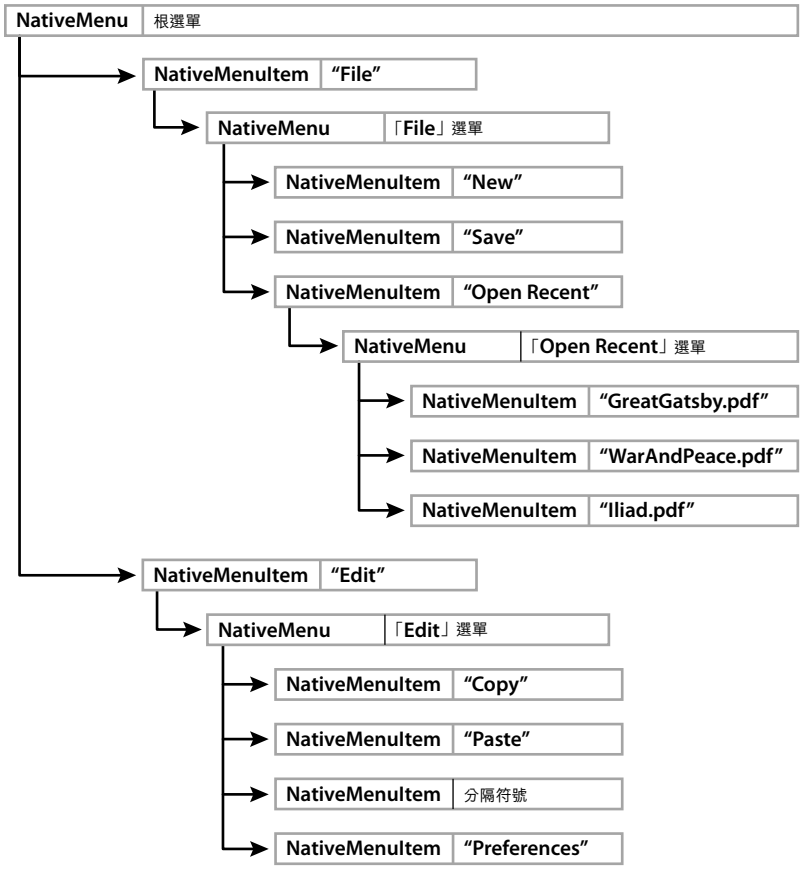
```
var square:Sprite = new Sprite();
square.graphics.beginFill(0x000000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.x =
square.y = 10;
addChild(square);

var menuItem:ContextMenuItem = new ContextMenuItem("Change Color");
menuItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT,changeColor);
var customContextMenu:ContextMenu = new ContextMenu();
customContextMenu.hideBuiltInItems();
customContextMenu.customItems.push(menuItem);
square.contextMenu = customContextMenu;

function changeColor(event:ContextMenuEvent):void
{
    square.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    return new ColorTransform(Math.random(), Math.random(), Math.random(),1,(Math.random() * 512) - 255,
(Math.random() * 512) -255, (Math.random() * 512) - 255, 0);
}
```

原生選單結構 (AIR)

Adobe AIR 1.0 以及更新的版本

原生選單本質上都具有階層。NativeMenu 物件包含子 NativeMenuItem 物件。NativeMenuItem 物件表示子選單，因此可以包含 NativeMenu 物件。結構中的最上層或根層級物件表示應用程式和視窗選單的選單列 (內容、圖示和彈出式選單沒有選單列)。

下列圖表將說明典型選單的結構。根選單表示選單列，其中包含參考 File 與 Edit 子選單的兩個選單項目。結構中的 File 子選單包含兩個命令項目和一個參考 「Open Recent」選單子選單的項目，而這個子選單則包含三個項目。Edit 子選單包含三個命令和一個分隔符號。



定義子選單時，需要 NativeMenu 和 NativeMenuItem 物件。NativeMenuItem 物件會定義上層選單中顯示的標籤，並且可以讓使用者開啟子選單。NativeMenu 物件會當做子選單中項目的容器來使用。NativeMenuItem 物件會透過 NativeMenuItem 的 submenu 屬性參考 NativeMenu 物件。

如需建立這個選單的程式碼範例，請參閱第 553 頁 「原生選單範例：視窗與應用程式選單 (AIR)」。

選單事件

Adobe AIR 1.0 以及更新的版本

NativeMenu 與 NativeMenuItem 物件兩者都會傳送 preparing、displaying 和 select 事件：

Preparing：每當物件即將開始使用者互動時，選單及其選單項目會將 preparing 事件傳送至任何已註冊的偵聽程式。互動包括使用鍵盤快速鍵開啟選單或是選取項目。

備註：preparing 事件只有在 Adobe AIR 2.6 和更新的版本中才能使用。

displaying： 在選單顯示之前，選單和其選單項目都會立即將 displaying 事件傳送至任何已註冊的偵聽程式。

preparing 與 displaying 事件能讓您在使用者看到選單之前，更新選單內容或項目的外觀。例如，在 「Open Recent」選單之 displaying 事件的偵聽程式中，您可以變更選單項目以反映最近檢視過的文件清單。

如果您移除選單項目，其鍵盤快速鍵會觸發 preparing 事件，則會有效地取消選單互動，而且不會傳送 select 事件。

事件的 target 與 currentTarget 屬性都是已註冊偵聽程式的物件：選單本身或是選單的其中一個項目。

preparing 事件會在 displaying 事件之前傳送。您通常會偵聽一個事件或另一個事件，而不會同時偵聽兩者。

select： 當使用者選擇某個命令項目時，這個項目便會將 select 事件傳送至任何已註冊的偵聽程式。由於無法選取子選單和分隔符號，因此這兩個項目永遠都不會傳送 select 事件。

select 事件會從某個選單項目反昇至該項目所屬的選單，並且一直向反昇至根選單。您可以直接對某個項目偵聽 select 事件，而且可以在選單結構中偵聽更高層級的項目。當您對某個選單偵聽 select 事件時，可以使用事件的 target 屬性識別所選取的項目。由於事件會經由選單階層反昇，因此事件物件的 currentTarget 屬性會識別目前的選單物件。

備註：ContextMenu 和 ContextMenuItem 物件都會傳送 menuItemSelect 和 menuSelect 事件以及 select、preparing 和 displaying 事件。

原生選單命令的對等按鍵 (AIR)

Adobe AIR 1.0 以及更新的版本

您可以為選單命令指定對等按鍵 (有時稱為快速鍵)。當按下按鍵或按鍵組合時，選單命令會將 select 事件傳送至任何已註冊的偵聽程式。包含項目的選單必須是應用程式或使用中視窗之選單的一部分，才能叫用命令。

對等按鍵包含兩個部分：表示主要按鍵的字串，以及必須按下之輔助按鍵的陣列。若要指定主要按鍵，請將選單項目的 keyEquivalent 屬性設定為表示該按鍵的單一字元字串。如果您使用大寫字母，則輔助按鍵陣列中會自動加入 Shift 鍵。

在 Mac OS X 中，預設輔助按鍵為 Command 鍵 (Keyboard.COMMAND)；在 Windows 和 Linux 中，則為 Control 鍵 (Keyboard.CONTROL)。這些預設按鍵都會自動加入至輔助按鍵陣列中。若要指定不同的輔助按鍵，請將包含所需按鍵碼的新陣列指定給 keyEquivalentModifiers 屬性，而預設的陣列便會遭到覆寫。不管您使用預設輔助按鍵，還是指定專屬輔助按鍵陣列，如果指定給 keyEquivalent 屬性的字串為大寫字母，陣列中都會加入 Shift 鍵。代表用來當做輔助按鍵之按鍵碼的常數都已經在 Keyboard 類別中加以定義。

所指定的對等按鍵字串會自動在選單項目名稱旁顯示，其格式則視使用者的作業系統和系統偏好設定而異。

備註：如果您將 Keyboard.COMMAND 值指定給 Windows 作業系統中的輔助按鍵陣列，則選單中不會顯示任何對等按鍵。不過，Control 鍵必須用來啟動選單命令。

下列範例會指定 Ctrl+Shift+G 做為選單項目的對等按鍵：

```
var item:NativeMenuItem = new NativeMenuItem("Ungroup");
item.keyEquivalent = "G";
```

下列範例會藉由直接設定輔助按鍵陣列，指定 Ctrl+Shift+G 做為對等按鍵：

```
var item:NativeMenuItem = new NativeMenuItem("Ungroup");
item.keyEquivalent = "G";
item.keyEquivalentModifiers = [Keyboard.CONTROL];
```

備註：您只能針對應用程式和視窗選單觸發對等按鍵。如果您將對等按鍵加入至快顯選單或彈出式選單，雖然對等按鍵會在選單標籤中顯示，但是卻永遠不會叫用相關聯的選單命令。

記憶鍵 (AIR)

Adobe AIR 1.0 以及更新的版本

記憶鍵是作業系統中選單之鍵盤介面的一部分。Linux、Mac OS X 和 Windows 都能讓使用者透過鍵盤開啟選單及選取命令，但是三者之間有細微的差別。

在 Mac OS X 中，使用者會輸入選單或命令的第一個或前兩個字母，然後按 Return 鍵。mnemonicIndex 屬性會遭到忽略。

在 Windows 中，只需輸入一個代表性字母即可。這個代表性字母預設是標籤中的第一個字元，但是，如果您將記憶鍵指定給選單項目，此字元就會成為指定的字母。如果選單中有兩個項目具有相同的代表性字元 (無論是否已指定記憶鍵)，則使用者鍵盤與選單的互動便會有些微的變更。使用者必須依需要按下字母多次以反白顯示所需的項目，然後按下 Enter 鍵才能完成選取動作，而不是單按一次字母就能選取選單或命令。若要保持一致的行為，就應該針對視窗選單，將唯一的記憶鍵指定給選單中的每個項目。

在 Linux 上不會提供預設的記憶鍵。您必須為選單項目指定 mnemonicIndex 屬性的 ⊠，以提供記憶鍵。

您可以指定記憶鍵字元做為標籤字串中的索引。標籤中第一個字元的索引為 0。因此，若要使用 「r」當做標籤為 「Format」之選單項目的記憶鍵，請將 mnemonicIndex 屬性設定為等於 2。

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.mnemonicIndex = 2;
```

選單項目狀態

Adobe AIR 1.0 以及更新的版本

選單項目有 checked 和 enabled 兩個狀態屬性：

checked 設定為 true 即可在項目標籤旁顯示核取標記。

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.checked = true;
```

enabled 切換 true 和 false 值即可控制是否啟用命令。已停用的項目會呈現 「灰色」，並且不會傳送 select 事件。

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.enabled = false;
```

將物件附加至選單項目

Adobe AIR 1.0 以及更新的版本

NativeMenuItem 類別的 data 屬性可以讓您參考每個項目中的任意物件。例如，在 「Open Recent」選單中，您可以將每份文件的 File 物件指定給每個選單項目。

```
var file:File = File.applicationStorageDirectory.resolvePath("GreatGatsby.pdf")
var menuItem:NativeMenuItem = docMenu.addItem(new NativeMenuItem(file.name));
menuItem.data = file;
```

建立原生選單 (AIR)

Adobe AIR 1.0 以及更新的版本

本主題將說明如何建立 AIR 支援的各種原生選單類型。

建立根選單物件

Adobe AIR 1.0 以及更新的版本

若要建立 NativeMenu 物件做為選單的根層級，請使用 NativeMenu 建構函式：

```
var root:NativeMenu = new NativeMenu();
```

對於應用程式和視窗選單而言，根選單表示選單列，並且只能包含可開啟子選單的項目。由於快顯選單與彈出式選單都沒有選單列，因此根選單可以包含命令、分隔線和子選單。

建立選單之後，您可以加入選單項目。除非您使用選單物件的 addItemAt() 方法以特定索引方式加入項目，不然項目在選單中的顯示順序即為加入時的順序。

請指定選單做為應用程式、視窗、圖示或快顯選單，或者將選單顯示為彈出式選單，如下列各節所示：

設定應用程式選單或視窗選單

您的程式碼要同時容納應用程式選單或 (Mac 作業系統支援) 以及視窗選單 (其他作業系統支援)，這一點很重要。

```
var root:NativeMenu = new NativeMenu();
if (NativeApplication.supportsMenu)
{
    NativeApplication.nativeApplication.menu = root;
}
else if (NativeWindow.supportsMenu)
{
    nativeWindow.menu = root;
}
```

備註：Mac OS 會針對每個應用程式，定義包含標準項目的選單。如果將新的 NativeMenu 物件指定給 NativeApplication 物件的 menu 屬性，即可取代標準選單。您也可以使用標準選單而非將其取代。

Adobe Flex 提供 FlexNativeMenu 類別，可以用來輕輕鬆鬆建立各平台都有效的選單。如果您使用 Flex 架構，就可以使用 FlexNativeMenu 類別來取代 NativeMenu 類別。

在互動式物件上設定快顯選單

```
interactiveObject.contextMenu = root;
```

設定停駐列圖示選單或系統匣圖示選單

您的程式碼要同時容納應用程式選單或 (Mac 作業系統支援) 以及視窗選單 (其他作業系統支援)，這一點很重要。

```
if (NativeApplication.supportsSystemTrayIcon)
{
    SystemTrayIcon(NativeApplication.nativeApplication.icon).menu = root;
}
else if (NativeApplication.supportsDockIcon)
{
    DockIcon(NativeApplication.nativeApplication.icon).menu = root;
}
```

備註：Mac OS X 會針對應用程式停駐列圖示，定義標準選單。當您將新的 NativeMenu 指定給 DockIcon 物件的 menu 屬性時，該選單中的所有項目都會顯示在標準項目的上方。您無法移除、存取或修改標準選單項目。

以彈出方式顯示選單

```
root.display(stage, x, y);
```

更多說明主題

開發跨平台 AIR 應用程式

建立子選單

Adobe AIR 1.0 以及更新的版本

若要建立子選單，請將 NativeMenuItem 物件加入至上層選單，然後將定義子選單的 NativeMenu 物件指定給該項目的 submenu 屬性。AIR 可提供兩種方式，讓您建立子選單項目及其相關聯的選單物件：

您可以使用 addSubmenu() 方法，一次建立選單項目及其相關的選單物件：

```
var editMenuItem:NativeMenuItem = root.addSubmenu(new NativeMenu(), "Edit");
```

您也可以先建立選單項目，然後再將選單物件指定給該選單項目的 submenu 屬性：

```
var editMenuItem:NativeMenuItem = root.addItem("Edit", false);
editMenuItem.submenu = new NativeMenu();
```

建立選單命令

Adobe AIR 1.0 以及更新的版本

若要建立選單命令，請將 NativeMenuItem 物件加入至選單，然後再加入參考實作選單命令之函數的事件偵聽程式：

```
var copy:NativeMenuItem = new NativeMenuItem("Copy", false);
copy.addEventListener(Event.SELECT, onCopyCommand);
editMenu.addItem(copy);
```

您可以對命令項目本身偵聽 select 事件 (如下列範例所示)，或者對上層選單物件偵聽 select 事件。

備註：表示子選單和分隔線的選單項目不會傳送 select 事件，因此無法當做命令使用。

建立選單分隔線

Adobe AIR 1.0 以及更新的版本

若要建立分隔線，請建立 NativeMenuItem 並在建構函式中將 isSeparator 參數設定為 true。接著，再將分隔線加入至選單中的正確位置：

```
var separatorA:NativeMenuItem = new NativeMenuItem("A", true);
editMenu.addItem(separatorA);
```

為該分隔線指定的標籤 (如果有的話) 並不會顯示。

關於 HTML 中的快顯選單 (AIR)

Adobe AIR 1.0 以及更新的版本

在使用 HTMLLoader 物件顯示的 HTML 內容中，contextmenu 事件可用來顯示快顯選單。當使用者對所選取的文字叫用 contextmenu 事件 (藉由以滑鼠右鍵按一下或按 Command+ 按一下文字的方式) 時，快顯選單預設會自動顯示。若要避免開啟預設選單，請偵聽 contextmenu 事件，並呼叫事件物件的 preventDefault() 方法：

```
function showContextMenu(event){
    event.preventDefault();
}
```

接著，您可以使用 DHTML 技巧顯示自訂的快顯選單，或者顯示 AIR 原生快顯選單。在下列範例中，會呼叫選單的 display() 方法以回應 HTML 的 contextmenu 事件，藉此顯示原生選單：

```
<html>
<head>
<script src="AIRAliases.js" language="JavaScript" type="text/javascript"></script>
<script language="javascript" type="text/javascript">

function showContextMenu(event){
    event.preventDefault();
    contextMenu.display(window.nativeWindow.stage, event.clientX, event.clientY);
}

function createContextMenu(){
    var menu = new air.NativeMenu();
    var command = menu.addItem(new air.NativeMenuItem("Custom command"));
    command.addEventListener(air.Event.SELECT, onCommand);
    return menu;
}

function onCommand(){
    air.trace("Context command invoked.");
}

var contextMenu = createContextMenu();
</script>
</head>
<body>
<p oncontextmenu="showContextMenu(event)" style="-khtml-user-select:auto;">Custom context menu.</p>
</body>
</html>
```

顯示彈出式原生選單 (AIR)

Adobe AIR 1.0 以及更新的版本

您可以藉由呼叫選單的 display() 方法，隨時在視窗上方的任意位置顯示任何 NativeMenu 物件。這個方法必須參考舞台，因此只有應用程式安全執行程序中的內容能以彈出方式顯示選單。

下列方法會顯示名為 popupMenu 之 NativeMenu 物件所定義的選單，以回應按一下滑鼠的動作：

```
private function onMouseClick(event:MouseEvent):void {
    popupMenu.display(event.target.stage, event.stageX, event.stageY);
}
```

備註：這個選單不一定要為了回應事件而顯示。任何方法都可以呼叫 display() 函數。

處理選單事件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當使用者選取選單或選單項目時，選單便會傳送事件。

選單類別的事件摘要

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以將事件偵聽程式加入至選單或個別項目，以處理選單事件。

Object	傳送的事件
NativeMenu (AIR)	Event.PREPARING (Adobe AIR 2.6 及更新的版本) Event.DISPLAYING Event.SELECT (從子項目和子選單傳遞)
NativeMenuItem (AIR)	Event.PREPARING (Adobe AIR 2.6 及更新的版本) Event.SELECT Event.DISPLAYING (從上層選單傳遞)
ContextMenu	ContextMenuEvent.MENU_SELECT
ContextMenuItem	ContextMenuEvent.MENU_ITEM_SELECT Event.SELECT (AIR)

選取選單事件

Adobe AIR 1.0 以及更新的版本

若要處理按一下選單項目的動作，請將 select 事件的事件偵聽程式加入至 NativeMenuItem 物件：

```
var menuCommandX:NativeMenuItem = new NativeMenuItem("Command X");
menuCommandX.addEventListener(Event.SELECT, doCommandX)
```

由於 select 事件都會反昇至項目所屬的選單，因此您也可以對上層選單偵聽 select 事件。在選單層級進行偵聽時，您可以使用事件物件的 target 屬性，判斷所選取的選單命令。下列範例會追蹤所選取命令的標籤：

```
var colorMenuItem:NativeMenuItem = new NativeMenuItem("Choose a color");
var colorMenu:NativeMenu = new NativeMenu();
colorMenuItem.submenu = colorMenu;

var red:NativeMenuItem = new NativeMenuItem("Red");
var green:NativeMenuItem = new NativeMenuItem("Green");
var blue:NativeMenuItem = new NativeMenuItem("Blue");
colorMenu.addItem(red);
colorMenu.addItem(green);
colorMenu.addItem(blue);

if(NativeApplication.supportsMenu){
    NativeApplication.nativeApplication.menu.addItem(colorMenuItem);
    NativeApplication.nativeApplication.menu.addEventListener(Event.SELECT, colorChoice);
} else if (NativeWindow.supportsMenu){
    var windowMenu:NativeMenu = new NativeMenu();
    this.stage.nativeWindow.menu = windowMenu;
    windowMenu.addItem(colorMenuItem);
    windowMenu.addEventListener(Event.SELECT, colorChoice);
}

function colorChoice(event:Event):void {
    var menuItem:NativeMenuItem = event.target as NativeMenuItem;
    trace(menuItem.label + " has been selected");
}
```

如果您使用 ContextMenuItem 類別，便可以偵聽 select 或 menuItemSelect 事件。menuItemSelect 事件可提供您關於擁有快顯選單之物件的額外資訊，但是不會反昇至項目所屬的選單。

displaying 選單事件

Adobe AIR 1.0 以及更新的版本

若要處理開啟選單的動作，您可以加入 displaying 事件的偵聽程式，這個事件會在選單顯示之前傳送。您可以使用 displaying 事件更新選單 (例如，透過加入或移除項目，或透過更新個別項目的 enabled 或 checked 狀態)。您亦可從 ContextMenu 物件偵聽 menuSelect 事件。

在 AIR 2.6 及更新的版本中，您可以使用 preparing 事件來更新選單，以回應顯示選單或使用鍵盤快速鍵選取項目。

原生選單範例：視窗與應用程式選單 (AIR)

Adobe AIR 1.0 以及更新的版本

下列範例會建立第 545 頁 「原生選單結構 (AIR)」中所示的選單。

這個選單是針對在 Windows (只支援視窗選單) 和 Mac OS X (只支援應用程式選單) 中運作而設計。為了辨識作業系統，MenuExample 類別建構函式會檢查 NativeWindow 和 NativeApplication 類別的靜態 supportsMenu 屬性。如果 NativeWindow.supportsMenu 為 true，建構函式便會為視窗建立 NativeMenu 物件，然後建立並加入 File 和 Edit 子選單。如果 NativeApplication.supportsMenu 為 true，建構函式便會建立 File 和 Edit 選單，並將它們加入由 Mac OS X 作業系統所提供的現有選單。

這個範例也會示範選單事件的處理方式。處理 select 事件時，會在項目層級和選單層級處理。這個鏈結中的每個選單 (從包含所選取項目的選單到根選單) 都會回應 select 事件。displaying 事件會搭配「Open Recent」選單使用。就在選單開啟之前，選單中的項目都會根據 recentDocuments 陣列重新整理 (在此範例中，這個陣列實際上並不會變更)。您也可以對個別項目偵聽 displaying 事件，不過此範例不會這麼做。

```
package {
    import flash.display.NativeMenu;
    import flash.display.NativeMenuItem;
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.filesystem.File;
    import flash.desktop.NativeApplication;

    public class MenuExample extends Sprite
    {
        private var recentDocuments:Array =
            new Array(new File("app-storage:/GreatGatsby.pdf"),
                      new File("app-storage:/WarAndPeace.pdf"),
                      new File("app-storage:/Iliad.pdf"));

        public function MenuExample()
        {
            var fileMenu:NativeMenuItem;
            var editMenu:NativeMenuItem;

            if (NativeWindow.supportsMenu){
                stage.nativeWindow.menu = new NativeMenu();
                stage.nativeWindow.menu.addEventListener(Event.SELECT, selectCommandMenu);
                fileMenu = stage.nativeWindow.menu.addItem(new NativeMenuItem("File"));
```

```
                fileMenu.submenu = createFileMenu();
                editMenu = stage.nativeWindow.menu.addItem(new NativeMenuItem("Edit"));
                editMenu.submenu = createEditMenu();
            }

            if (NativeApplication.supportsMenu){
                NativeApplication.nativeApplication.menu.addEventListener(Event.SELECT,
selectCommandMenu);
                fileMenu = NativeApplication.nativeApplication.menu.addItem(new NativeMenuItem("File"));
                fileMenu.submenu = createFileMenu();
                editMenu = NativeApplication.nativeApplication.menu.addItem(new NativeMenuItem("Edit"));
                editMenu.submenu = createEditMenu();
            }
        }

        public function createFileMenu():NativeMenu {
            var fileMenu:NativeMenu = new NativeMenu();
            fileMenu.addEventListener(Event.SELECT, selectCommandMenu);

            var newCommand:NativeMenuItem = fileMenu.addItem(new NativeMenuItem("New"));
            newCommand.addEventListener(Event.SELECT, selectCommand);
            var saveCommand:NativeMenuItem = fileMenu.addItem(new NativeMenuItem("Save"));
            saveCommand.addEventListener(Event.SELECT, selectCommand);
            var openRecentMenu:NativeMenuItem =
                    fileMenu.addItem(new NativeMenuItem("Open Recent"));
            openRecentMenu.submenu = new NativeMenu();
            openRecentMenu.submenu.addEventListener(Event.DISPLAYING,
                                            updateRecentDocumentMenu);
            openRecentMenu.submenu.addEventListener(Event.SELECT, selectCommandMenu);

            return fileMenu;
        }

        public function createEditMenu():NativeMenu {
            var editMenu:NativeMenu = new NativeMenu();
            editMenu.addEventListener(Event.SELECT, selectCommandMenu);

            var copyCommand:NativeMenuItem = editMenu.addItem(new NativeMenuItem("Copy"));
            copyCommand.addEventListener(Event.SELECT, selectCommand);
            copyCommand.keyEquivalent = "c";
            var pasteCommand:NativeMenuItem =
                    editMenu.addItem(new NativeMenuItem("Paste"));
            pasteCommand.addEventListener(Event.SELECT, selectCommand);
            pasteCommand.keyEquivalent = "v";
            editMenu.addItem(new NativeMenuItem("", true));
            var preferencesCommand:NativeMenuItem =
                    editMenu.addItem(new NativeMenuItem("Preferences"));
            preferencesCommand.addEventListener(Event.SELECT, selectCommand);

            return editMenu;
        }

        private function updateRecentDocumentMenu(event:Event):void {
            trace("Updating recent document menu.");
            var docMenu:NativeMenu = NativeMenu(event.target);

            for each (var item:NativeMenuItem in docMenu.items) {
                docMenu.removeItem(item);
            }

            for each (var file:File in recentDocuments) {
                var menuItem:NativeMenuItem =
```

```
                    docMenu.addItem(new NativeMenuItem(file.name));
            menuItem.data = file;
            menuItem.addEventListener(Event.SELECT, selectRecentDocument);
        }
    }

    private function selectRecentDocument(event:Event):void {
        trace("Selected recent document: " + event.target.data.name);
    }

    private function selectCommand(event:Event):void {
        trace("Selected command: " + event.target.label);
    }

    private function selectCommandMenu(event:Event):void {
        if (event.currentTarget.parent != null) {
            var menuItem:NativeMenuItem =
                    findItemForMenu(NativeMenu(event.currentTarget));
            if (menuItem != null) {
                trace("Select event for \"" +
                        event.target.label +
                        "\" command handled by menu: " +
                        menuItem.label);
            }
        } else {
            trace("Select event for \"" +
                    event.target.label +
                    "\" command handled by root menu.");
        }
    }

    private function findItemForMenu(menu:NativeMenu):NativeMenuItem {
        for each (var item:NativeMenuItem in menu.parent.items) {
            if (item != null) {
                if (item.submenu == menu) {
                    return item;
                }
            }
        }
        return null;
    }
  }
}
```

第 37 章 AIR 中的工作列圖示

Adobe AIR 1.0 以及更新的版本

許多作業系統 (例如 Mac OS X 停駐列) 都提供工作列，可包含代表應用程式的圖示。Adobe® AIR® 提供了介面，可透過 NativeApplication.nativeApplication.icon 屬性來與應用程式工作列圖示互動。

- 使用系統匣圖示和停駐圖示 (Flex)
- 使用系統匣圖示和停駐圖示 (Flash)

更多說明主題

flash.desktop.NativeApplication

flash.desktop.DockIcon

flash.desktop.SystemTrayIcon

關於工作列圖示

Adobe AIR 1.0 以及更新的版本

AIR 會自動建立 NativeApplication.nativeApplication.icon 物件。依作業系統而定，此物件類型會是 DockIcon 或 SystemTrayIcon。您可以使用 NativeApplication.supportsDockIcon 和 NativeApplication.supportsSystemTrayIcon 屬性，判斷 AIR 在目前的作業系統上所支援的 InteractiveIcon 子類別。InteractiveIcon 基底類別提供 width、height 和 bitmaps 等屬性，您可以使用這些屬性變更用來做為圖示的影像，但是，在不適當的作業系統上存取特屬 DockIcon 或 SystemTrayIcon 的屬性會產生執行階段錯誤。

若要設定或變更用來做為圖示的影像，請建立包含一個或多個影像的陣列，然後指定給 NativeApplication.nativeApplication.icon.bitmaps 屬性。工作列圖示的大小可能會因不同作業系統而有差異。若要避免因縮放而降低影像品質，您可以將多種大小的影像加入 bitmaps 陣列。如果您提供多個影像，AIR 會選取最接近目前顯示大小的工作列圖示，只有在必要時才進行縮放。下列範例會使用兩種影像來設定工作列圖示的影像：

```
NativeApplication.nativeApplication.icon.bitmaps =
            [bmp16x16.bitmapData, bmp128x128.bitmapData];
```

若要變更圖示影像，請將包含一個或多個新影像指定給 bitmaps 屬性。您可以變更影像來回應 enterFrame 或 timer 事件，以產生動畫圖示。

若要從 Windows 和 Linux 上的通知區域移除圖示，或是在 Mac OS X 上還原預設圖示外觀，請將 bitmaps 設定為空陣列：

```
NativeApplication.nativeApplication.icon.bitmaps = [];
```

停駐圖示

Adobe AIR 1.0 以及更新的版本

當 NativeApplication.supportsDockIcon 為 true 時，AIR 支援停駐圖示。NativeApplication.nativeApplication.icon 屬性代表停駐列上的應用程式圖示 (而不是視窗停駐圖示)。

備註：AIR 不支援在 Mac OS X 之下的停駐列上變更視窗圖示，而且應用程式停駐圖示的變更只會在應用程式執行時套用，該圖示會在應用程式終止時回復其正常外觀。

停駐圖示選單

Adobe AIR 1.0 以及更新的版本

您可以建立包含命令的 NativeMenu 物件，並指定給 NativeApplication.nativeApplication.icon.menu 屬性，將命令加入至標準停駐選單。選單中的項目是顯示於標準停駐圖示選單項目的上方。

讓停駐圖示跳躍

Adobe AIR 1.0 以及更新的版本

您可以呼叫 NativeApplication.nativeApplication.icon.bounce() 方法，讓停駐圖示跳躍。如果將 bounce() priority 參數設定為 informational，則圖示會跳躍一次；如果將它設定為 critical，則圖示會一直跳躍，直到使用者啟動應用程式為止。priority 參數的常數會定義於 NotificationType 類別中。

備註：如果應用程式已在作用中，則圖示不會跳躍。

停駐圖示事件

Adobe AIR 1.0 以及更新的版本

按一下停駐圖示時，NativeApplication 物件會傳送 invoke 事件。如果應用程式不在執行中，系統會啟動該應用程式；否則，會將 invoke 事件遞送給執行中的應用程式實體。

系統匣圖示

Adobe AIR 1.0 以及更新的版本

當 NativeApplication.supportsSystemTrayIcon 為 true 時 (目前這是 Windows 和大部分 Linux 版本才有的情況)，AIR 會支援系統匣圖示。在 Windows 和 Linux 中，系統匣圖示會顯示在工作列的通知區域。根據預設，不會顯示任何圖示。若要顯示圖示，請將包含 BitmapData 物件的陣列指定給圖示 bitmaps 屬性。若要變更圖示影像，請將包含新影像的陣列指定給 bitmaps。若要移除圖示，請將 bitmaps 設定為 null。

系統匣圖示選單

Adobe AIR 1.0 以及更新的版本

您可以建立 NativeMenu 物件並將該物件指定給 NativeApplication.nativeApplication.icon.menu 屬性，將選單加入系統匣圖示 (作業系統不提供預設選單)。用滑鼠右鍵按一下圖示，即可存取系統匣圖示選單。

系統匣圖示工具提示

Adobe AIR 1.0 以及更新的版本

設定工具提示屬性，即可將工具提示加入至圖示：

```
NativeApplication.nativeApplication.icon.tooltip = "Application name";
```

系統匣圖示事件

Adobe AIR 1.0 以及更新的版本

NativeApplication.nativeApplication.icon 屬性參考的 SystemTrayIcon 物件會為 click、mouseDown、mouseUp、rightClick、rightMouseDown 和 rightMouseUp 事件傳送 ScreenMouseEvent。您可以搭配圖示選單使用這些事件，讓使用者能夠在沒有可見視窗時，與您的應用程式互動。

範例：建立沒有視窗的應用程式

Adobe AIR 1.0 以及更新的版本

下列範例會建立具有系統匣圖示，但沒有可見視窗的 AIR 應用程式。(應用程式的 visible 屬性在應用程式描述器中不可設為 true，否則應用程式啟動時會看到視窗。

```
package
{
    import flash.display.Loader;
    import flash.display.NativeMenu;
    import flash.display.NativeMenuItem;
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.desktop.DockIcon;
    import flash.desktop.SystemTrayIcon;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.desktop.NativeApplication;

    public class SysTrayApp extends Sprite
    {
        public function SysTrayApp():void{
            NativeApplication.nativeApplication.autoExit = false;
            var icon:Loader = new Loader();
            var iconMenu:NativeMenu = new NativeMenu();
            var exitCommand:NativeMenuItem = iconMenu.addItem(new NativeMenuItem("Exit"));
                exitCommand.addEventListener(Event.SELECT, function(event:Event):void {
                    NativeApplication.nativeApplication.icon.bitmaps = [];
                    NativeApplication.nativeApplication.exit();
                });

            if (NativeApplication.supportsSystemTrayIcon) {
                NativeApplication.nativeApplication.autoExit = false;
                icon.contentLoaderInfo.addEventListener(Event.COMPLETE, iconLoadComplete);
                icon.load(new URLRequest("icons/AIRApp_16.png"));

                var systray:SystemTrayIcon =
```

```
                NativeApplication.nativeApplication.icon as SystemTrayIcon;
            systray.tooltip = "AIR application";
            systray.menu = iconMenu;
        }

        if (NativeApplication.supportsDockIcon){
            icon.contentLoaderInfo.addEventListener(Event.COMPLETE,iconLoadComplete);
            icon.load(new URLRequest("icons/AIRApp_128.png"));
            var dock:DockIcon = NativeApplication.nativeApplication.icon as DockIcon;
            dock.menu = iconMenu;
        }
    }

    private function iconLoadComplete(event:Event):void
    {
        NativeApplication.nativeApplication.icon.bitmaps =
            [event.target.content.bitmapData];
    }
  }
}
```

備註：使用 Flex WindowedApplication 組件時，您必須將 WindowedApplication 標籤的 visible 特質設為 false。此特質會取代應用程式描述器中的設定。

備註：此範例假定在應用程式的 icons 子目錄中有名為 AIRApp_16.png 和 AIRApp_128.png 的影像檔案 (AIR SDK 中包含樣本圖示檔案，您可以將此檔案複製至專案資料夾中)。

視窗工作列圖示和按鈕

Adobe AIR 1.0 以及更新的版本

圖示化的視窗一般會顯示於工作列或停駐列的視窗區域，以便讓使用者輕鬆存取背景或將視窗最小化。Mac OS X 停駐列會顯示您應用程式的圖示，並顯示每個最小化視窗的圖示。Microsoft Windows 和 Linux 工作列會顯示按鈕，其中包含程式圖示和應用程式在每個一般類型視窗中的標題。

反白標示工作列視窗按鈕

Adobe AIR 1.0 以及更新的版本

當視窗位於背景中時，您可以通知使用者與視窗相關的關注事件已發生。在 Mac OS X 中，您可以透過應用程式停駐圖示跳躍來通知使用者 (如第 557 頁「讓停駐圖示跳躍」一節中所述)。在 Windows 和 Linux 上，您可以呼叫 NativeWindow 實體的 notifyUser() 方法，反白標示視窗工作列按鈕。傳遞至該方法的 type 參數會判斷通知的急迫性：

- NotificationType.CRITICAL：視窗圖示會閃爍，直到使用者將視窗移到前景為止。
- NotificationType.INFORMATIONAL：透過變更顏色來反白標示視窗圖示。

  備註：在 Linux 上只會支援資訊類型的通知。不論傳遞哪一種類型的 ⊠給 notifyUser() 函數，都會製造相同的效果。

  下列陳述式會反白標示視窗的工作列按鈕：

  ```
  stage.nativeWindow.notifyUser(NotificationType.CRITICAL);
  ```

  在不支援視窗層級通知的作業系統上呼叫 NativeWindow.notifyUser() 方法不會有任何作用。請使用 NativeWindow.supportsNotification 屬性來判斷是否支援視窗通知。

建立沒有工作列按鈕或圖示的視窗

Adobe AIR 1.0 以及更新的版本

在 Windows 作業系統上，用 utility 或 lightweight 類型建立的視窗不會出現在工作列上。不可見的視窗也不會顯示在工作列上。

由於起始視窗必須為 normal 類型，若要建立不會在工作列中出現任何視窗的應用程式，必須關閉起始視窗或讓它保持為不可見。若要關閉應用程式中所有視窗而不終止應用程式，請將 NativeApplication 物件的 autoExit 屬性設定為 false，再關閉最後一個視窗。若要完全避免起始視窗變成可見，請將 <visible>false</visible> 加入應用程式描述器檔案的 <initalWindow> 元素 (而不要將 visible 屬性設定為 true，或呼叫視窗的 activate() 方法)。

在應用程式所開啟的新視窗中，將傳遞給視窗建構函式之 NativeWindowInitOption 物件的 type 屬性設定為 NativeWindowType.UTILITY 或 NativeWindowType.LIGHTWEIGHT。

在 Mac OS X 中，最小化的視窗會顯示於停駐工作列上。您可以隱藏視窗而不要將它最小化，以避免顯示最小化圖示。下列範例會偵聽 nativeWindowDisplayState 變更事件，並在視窗正進行最小化時，取消該事件；而處理常式會將視窗 visible 屬性設定為 false：

```
 private function preventMinimize(event:NativeWindowDisplayStateEvent):void{
    if(event.afterDisplayState == NativeWindowDisplayState.MINIMIZED){
        event.preventDefault();
        event.target.visible = false;
    }
}
```

當您將 visible 屬性設定為 false 時，如果視窗已在 Mac OS X 停駐列上最小化，則不會移除停駐圖示。使用者仍然可以按一下圖示，讓視窗重新出現。

第 38 章 使用檔案系統

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash® Player 透過 FileReference 類別，提供基本的檔案讀寫功能。基於安全的原因，在 Flash Player 中執行時，使用者一律必須授予權限，您才能讀寫檔案。

Adobe® AIR® 提供的主機電腦檔案系統存取權，比 Flash Player 所提供的更加完整。使用 AIR 檔案系統 API，您便可以存取和管理目錄及檔案、建立目錄及檔案，以及將資料寫入檔案等等。

更多說明主題

flash.net.FileReference

flash.net.FileReferenceList

flash.filesystem.File

flash.filesystem.FileStream

使用 FileReference 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

FileReference 物件表示用戶端或伺服器電腦中的資料檔案。FileReference 類別的方法可以讓您的應用程式在本機載入和儲存資料檔案，以及與遠端伺服器之間來回傳輸檔案資料。

FileReference 類別現在提供兩種不同的方式供您載入、傳輸及儲存資料檔案。導入 FileReference 類別後，本類別就包括 browse() 方法、upload() 方法以及 download() 方法。使用 browse() 方法，允許使用者選取檔案。使用 upload() 方法，將檔案資料傳輸至遠端伺服器。使用 download() 方法，從伺服器擷取該資料後儲存至本機檔案。從 Flash Player 10 和 Adobe AIR 1.5 開始，FileReference 類別還包括 load() 和 save() 方法。load() 和 save() 方法也可以讓您直接存取以及儲存本機檔案。使用這些方法類似於在 URLLoader 和 Loader 類別中以對應方式命名的方法。

備註：延伸出 FileReference 類別的 File 類別，以及 FileStream 類別，提供了使用檔案和本機檔案系統的其他函數。僅 AIR 支援 File 和 FileStream 類別，Flash Player 並不支援。

FileReference 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

每個 FileReference 物件都代表本機電腦中的一個資料檔案。FileReference 類別的屬性包含以下相關資訊：檔案的大小、類型、檔名、副檔名、建立者、建立日期和修改日期。

備註：只有 Mac OS 支援 creator 屬性，所有其它的平台則會傳回 null。

備註：只有 Adobe AIR 才支援 extension 屬性。

您可以用下列兩種方法建立 FileReference 類別的實體：

- 使用 new 運算子，如下列程式碼所示：

```
import flash.net.FileReference;
var fileRef:FileReference = new FileReference();
```

- 呼叫 FileReferenceList.browse() 方法，這個方法會開啟對話方塊，提示使用者選取一個或多個要上傳的檔案。接著，如果使用者成功選取一個或多個檔案，方法便會建立 FileReference 物件的陣列。

一旦建立 FileReference 物件之後，您就可以執行下列工作：

- 呼叫 FileReference.browse() 方法，這個方法會開啟對話方塊，提示使用者從本機檔案系統選取一個檔案。這通常會在下列呼叫之前完成，即呼叫 FileReference.upload() 方法或 FileReference.load() 方法。呼叫 FileReference.upload() 方法，將檔案上傳至遠端伺服器。呼叫 FileReference.load() 方法，開啟本機檔案。
- 呼叫 FileReference.download() 方法。download() 方法會開啟對話方塊，讓使用者選取新檔案的儲存位置。接著，方法會從伺服器下載資料，並儲存在新檔案中。
- 呼叫 FileReference.load() 方法。這個方法會使用 browse() 方法，開始從先前選取的檔案載入資料。必須先完成 browse() 作業 (使用者選取檔案)，才能呼叫 load() 方法。
- 呼叫 FileReference.save() 方法。這個方法會開啟對話方塊，提示使用者在本機檔案系統上選擇單一檔案位置。然後，再將資料儲存至指定的位置。

備註：由於一次只能開啟一個對話方塊，因此一次只能執行一個 browse()、download() 或 save() 動作。

直到下列其中一種情況發生之前，不會定義 FileReference 物件屬性，例如 name、size 或 modificationDate：

- 已經呼叫 FileReference.browse() 方法或 FileReferenceList.browse() 方法，而且使用者已經從對話方塊中選取檔案。
- 已經呼叫 FileReference.download() 方法，而且使用者已經從對話方塊中指定新的檔案位置。

備註：執行下載動作時，在下載作業完成之前，只能填入 FileReference.name 屬性。檔案下載完成後，就可以使用所有的屬性。

在呼叫 FileReference.browse()、FileReferenceList.browse()、FileReference.download()、FileReference.load() 或 FileReference.save() 方法執行時，大部分播放程式會繼續進行 SWF 檔的播放，包括傳送事件和執行程式碼。

在上傳和下載作業中，SWF 檔只能存取其本身網域 (包含原則檔所指定的任何網域) 內的檔案。如果伺服器與啟動上傳或下載的 SWF 檔不在相同的網域中，您必須將原則檔存放在包含所要存取檔案的伺服器上。

請參閱 FileReference。

從檔案載入資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

FileReference.load() 方法可讓您將資料從本機檔案載入記憶體。

備註：若要載入記憶體，您的程式碼必須先呼叫 FileReference.browse() 方法，讓使用者選取要載入的檔案。此限制不適用於應用程式安全執行程序的 Adobe AIR 中執行的內容

FileReference.load() 方法會在被呼叫後立即傳回，但並不是立即就可使用正在上傳的資料。FileReference 物件會傳送事件，在載入程序的每個步驟都叫用偵聽程式方法。

在載入程序期間，FileReference 物件會傳送下列事件：

- open 事件 (Event.OPEN)：在載入作業開始時傳送。
- progress 事件 (ProgressEvent.PROGRESS)：在從檔案讀取資料時以資料位元組定期傳送。
- complete 事件 (Event.COMPLETE)：在載入作業順利完成時傳送。
- ioError 事件 (IOErrorEvent.IO_ERROR)：如果載入程序因為在開啟或讀取檔案資料時發生輸入 / 輸出錯誤而失敗時，就會傳送。

當 FileReference 物件傳送 complete 事件時，載入的資料可以做為 FileReference 物件的 data 屬性中之 ByteArray 存取。

下列範例會示範如何提示使用者選取檔案，然後將資料從檔案載入記憶體：

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample1 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample1()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onComplete(evt:Event):void
        {
            trace("File was successfully loaded.");
            trace(fileRef.data);
        }

        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

範例程式碼會先建立名為 fileRef 的 FileReference 物件，然後再呼叫其 browse() 方法。browse() 方法會開啟對話方塊，提示使用者選取檔案。選取檔案後，程式碼會叫用 onFileSelected() 方法。這個方法會為 progress 和 complete 事件加入偵聽程式，然後呼叫 FileReference 物件的 load() 方法。範例中的其它處理常式方法只會輸出訊息，以報告載入作業的進度。載入完成時，應用程式便會使用 trace() 方法，顯示所載入檔案的內容。

在 Adobe AIR 中，FileStream 類別提供其他函數，以便從本機檔案讀取資料。請參閱第 593 頁 「讀取和寫入檔案」。

將資料儲存至本機檔案

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

FileReference.save() 方法可讓您將資料儲存至本機檔案。首先，方法會開啟對話方塊，讓使用者輸入檔案名稱以及檔案的儲存位置。當使用者選取檔案名稱和儲存位置之後，資料隨即寫入新檔案。當檔案儲存成功時，本機檔案的屬性就會填入 FileReference 物件的屬性。

備註：您的程式碼只能呼叫 FileReference.save() 方法來回應使用者事件 (例如，按下滑鼠或按下按鍵事件的使用者啟動的事件)。否則就會擲出錯誤。此限制不適用於應用程式安全執行程序的 Adobe AIR 中執行的內容。

FileReference.save() 方法在遭到呼叫之後會立即傳回。接著，FileReference 物件會傳送事件，在檔案儲存程序的每個步驟都呼叫偵聽程式方法。

在檔案儲存程序期間，FileReference 物件會傳送下列事件：

- select 事件 (Event.SELECT)：在使用者指定所要儲存新檔案的檔案名稱和位置時傳送。
- cancel 事件 (Event.CANCEL)：在使用者按一下對話方塊中的 「取消」按鈕時傳送。
- open 事件 (Event.OPEN)：在儲存作業開始時傳送。
- progress 事件 (ProgressEvent.PROGRESS)：在資料位元組儲存至檔案時定期傳送。
- complete 事件 (Event.COMPLETE)：在儲存作業順利完成時傳送。
- ioError 事件 (IOErrorEvent.IO_ERROR)：如果儲存程序因為嘗試將資料儲存至檔案時發生輸入 / 輸出錯誤而失敗時，就會傳送。

傳入 FileReference.save() 方法之 data 參數中的物件類型將決定資料寫入檔案的方式：

- 如果是 String 值，則會使用 UTF-8 編碼，將這個值儲存為文字檔案。
- 如果是 XML 物件，則會以 XML 格式寫入至檔案，並保留所有格式設定。
- 如果是 ByteArray 物件，則其內容會直接寫入至檔案而不經過轉換。
- 如果是其它物件，FileReference.save() 方法會呼叫該物件的 toString() 方法，然後將產生的 String 值儲存為 UTF-8 文字檔案。如果無法呼叫物件的 toString() 方法，則會擲回錯誤。

如果 data 參數的值為 null，則會擲回錯誤。

下列程式碼會擴充上述 FileReference.load() 方法的範例。從檔案讀取資料之後，這個範例會提示使用者輸入檔案名稱，然後將資料儲存在新檔案中：

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample2 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample2()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }
        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
        }
        public function onComplete(evt:Event):void
        {
            trace("File was successfully loaded.");
            fileRef.removeEventListener(Event.SELECT, onFileSelected);
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.removeEventListener(Event.COMPLETE, onComplete);
            fileRef.removeEventListener(Event.CANCEL, onCancel);
            saveFile();
        }
        public function saveFile():void
        {
            fileRef.addEventListener(Event.SELECT, onSaveFileSelected);
            fileRef.save(fileRef.data,"NewFileName.txt");
        }

        public function onSaveFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.addEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.addEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveProgress(evt:ProgressEvent):void
```

```
        {
            trace("Saved " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onSaveComplete(evt:Event):void
        {
            trace("File saved.");
            fileRef.removeEventListener(Event.SELECT, onSaveFileSelected);
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.removeEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.removeEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveCancel(evt:Event):void
        {
            trace("The save request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

從檔案載入所有資料後，程式碼會呼叫 onComplete() 方法。onComplete() 方法會移除載入事件的偵聽程式，然後呼叫 saveFile() 方法。saveFile() 方法會呼叫 FileReference.save() 方法。FileReference.save() 方法會開啟新的對話方塊，讓使用者輸入新的檔案名稱和儲存檔案的位置。其餘事件偵聽程式方法都會追蹤檔案儲存程序的進度，直到程序完成為止。

在 Adobe AIR 中，FileStream 類別提供其他函數，以便將資料寫入本機檔案中。請參閱第 593 頁「讀取和寫入檔案」。

將檔案上傳到伺服器

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如果要將檔案上傳到伺服器，請先呼叫 browse() 方法，讓使用者可以選取一個或多個要上傳的檔案。然後，呼叫 FileReference.upload() 方法時，就會將選取的檔案傳輸到伺服器。如果使用者使用 FileReferenceList.browse() 方法選取多個檔案，Flash Player 就會建立包含所選取檔案的陣列，並命名為 FileReferenceList.fileList。然後，您就可以使用 FileReference.upload() 方法分別上傳每一個檔案。

備註：FileReference.browse() 方法只能讓您上傳單一檔案。若要讓使用者可以上傳多個檔案，請使用 FileReferenceList.browse() 方法。

根據預設，系統檔案挑選器對話方塊可以讓使用者從本機電腦挑選任何類型的檔案。開發人員可以使用 FileFilter 類別，並將檔案篩選實體的陣列傳遞給 browse() 方法，以指定一或多個自訂檔案類型篩選：

```
var imageTypes:FileFilter = new FileFilter("Images (*.jpg, *.jpeg, *.gif, *.png)", "*.jpg; *.jpeg; *.gif;
*.png");
var textTypes:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt; *.rtf");
var allTypes:Array = new Array(imageTypes, textTypes);
var fileRef:FileReference = new FileReference();
fileRef.browse(allTypes);
```

當使用者選取檔案，然後按一下系統檔案挑選器中的「開啟」按鈕後，就會送出 Event.SELECT 事件。如果已經使用 FileReference.browse() 方法選取要上傳的檔案，下列程式碼會將檔案傳送至網站伺服器：

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
try
{
    var success:Boolean = fileRef.browse();
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/fileUploadScript.cfm")
    try
    {
        fileRef.upload(request);
    }
    catch (error:Error)
    {
        trace("Unable to upload file.");
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

使用 FileReference.upload() 方法將資料傳送到伺服器時，您可以利用 URLRequest.method 和 URLRequest.data 屬性，以便使用 POST 或 GET 方法來傳送變數。

當您嘗試使用 FileReference.upload() 方法上傳檔案時，會傳送下列事件：

- open 事件 (Event.OPEN)：在上傳作業開始時傳送。
- progress 事件 (ProgressEvent.PROGRESS)：在上傳檔案的資料位元組時定期傳送。
- complete 事件 (Event.COMPLETE)：在上傳作業順利完成時傳送。
- httpStatus 事件 (HTTPStatusEvent.HTTP_STATUS)：在上傳程序因為 HTTP 錯誤而失敗時傳送。
- httpResponseStatus 事件 (HTTPStatusEvent.HTTP_RESPONSE_STATUS)：當 upload() 或 uploadUnencoded() 方法呼叫嘗試透過 HTTP 存取資料，而且 Adobe AIR 能夠偵測並傳回要求的狀態碼時傳送。
- securityError 事件 (SecurityErrorEvent.SECURITY_ERROR)：在上傳作業因為安全性違規而失敗時傳送。
- uploadCompleteData 事件 (DataEvent.UPLOAD_COMPLETE_DATA)：在已經順利上傳並從伺服器接收資料以後傳送。
- ioError 事件 (IOErrorEvent.IO_ERROR)：在上傳程序因為下列任一原因而失敗時傳送：
  - Flash Player 讀取、寫入或傳送檔案時發生輸入 / 輸出錯誤。
  - SWF 嘗試將檔案上傳至需要驗證 (如使用者名稱和密碼) 的伺服器。上傳期間，Flash Player 沒有提供讓使用者輸入密碼的工具。
  - url 參數的值包含無效的通訊協定。FileReference.upload() 方法必須使用 HTTP 或 HTTPS。

Flash Player 並不完全支援需要驗證的伺服器。只有在使用瀏覽器外掛程式或 Microsoft ActiveX® 控制項之瀏覽器中執行的 SWF 檔，才能提供對話方塊，以提示使用者輸入使用者名稱和密碼進行驗證，而且只能下載檔案。如果使用外掛程式或 ActiveX 控制項上傳檔案，或是使用獨立或外部播放程式進行上傳 / 下載作業，則檔案傳輸會失敗。

若要在 ColdFusion 中建立伺服器指令碼，以接受透過 Flash Player 上傳的檔案，您可以參考下列程式碼：

```
<cffile action="upload" filefield="Filedata" destination="#ExpandPath('./')#" nameconflict="OVERWRITE" />
```

這段 ColdFusion 程式碼會上傳由 Flash Player 送出的檔案，並將它儲存在 ColdFusion 範本所在的相同目錄中，而且會覆寫任何具有相同名稱的檔案。上一段程式碼使用的程式碼行數是接受檔案上傳所需的最少行數，因此這段指令碼不可以用於正式的執行環境。最理想的方式是加入資料驗證功能，以確保使用者只能上傳可接受的檔案類型 (如影像檔案)，而不使用具有潛在危險的伺服器端指令碼。

下列程式碼將示範如何使用 PHP 進行檔案上傳作業，其中也包括資料驗證。這段指令碼會將上傳目錄中可上傳的檔案數量限制為 10，並確保檔案大小少於 200 KB，而且限制只能上傳 JPEG、GIF 或 PNG 檔案並儲存到檔案系統。

```
<?php
$MAXIMUM_FILESIZE = 1024 * 200; // 200KB
$MAXIMUM_FILE_COUNT = 10; // keep maximum 10 files on server
echo exif_imagetype($_FILES['Filedata']);
if ($_FILES['Filedata']['size'] <= $MAXIMUM_FILESIZE)
{
    move_uploaded_file($_FILES['Filedata']['tmp_name'], "./temporary/".$_FILES['Filedata']['name']);
    $type = exif_imagetype("./temporary/".$_FILES['Filedata']['name']);
    if ($type == 1 || $type == 2 || $type == 3)
    {
        rename("./temporary/".$_FILES['Filedata']['name'], "./images/".$_FILES['Filedata']['name']);
    }
    else
    {
        unlink("./temporary/".$_FILES['Filedata']['name']);
    }
}
$directory = opendir('./images/');
$files = array();
while ($file = readdir($directory))
{
    array_push($files, array('./images/'.$file, filectime('./images/'.$file)));
}
usort($files, sorter);
if (count($files) > $MAXIMUM_FILE_COUNT)
{
    $files_to_delete = array_splice($files, 0, count($files) - $MAXIMUM_FILE_COUNT);
    for ($i = 0; $i < count($files_to_delete); $i++)
    {
        unlink($files_to_delete[$i][0]);
    }
}
print_r($files);
closedir($directory);

function sorter($a, $b)
{
    if ($a[1] == $b[1])
    {
        return 0;
    }
    else
    {
        return ($a[1] < $b[1]) ? -1 : 1;
    }
}
?>
```

您可以使用 POST 或 GET 要求方法，將其它變數傳遞給上傳指令碼。若要將其它 POST 變數傳遞給上傳指令碼，請使用下列程式碼：

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();
function selectHandler(event:Event):void
{
    var params:URLVariables = new URLVariables();
    params.date = new Date();
    params.ssid = "94103-1394-2345";
    var request:URLRequest = new
URLRequest("http://www.yourdomain.com/FileReferenceUpload/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    request.data = params;
    fileRef.upload(request, "Custom1");
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

上述範例會建立 URLVariables 物件，可傳遞給遠端伺服器端指令碼。在舊版 ActionScript 中，您可以將值傳入查詢字串，以便將變數傳遞給伺服器上傳指令碼。ActionScript 3.0 則允許您使用 URLRequest 物件將變數傳遞給遠端指令碼，可讓您在傳遞資料時使用 POST 或 GET 方法，因而使得傳遞大型資料集的作業更容易、更簡潔。為了指定變數使用 GET 或 POST 要求方法來傳遞，您可以分別將 URLRequest.method 屬性設為 URLRequestMethod.GET 或 URLRequestMethod.POST。

ActionScript 3.0 也可以讓您覆寫預設的 Filedata 上傳檔案欄位名稱，方式是：在使用 upload() 方法時加上第二個參數，如上一個範例中所示 (以 Custom1 取代預設值 Filedata)。

根據預設，Flash Player 並不會傳送測試性上傳作業，不過您可以傳遞 true 值做為 upload() 方法的第三個參數來覆寫這項預設值。測試上傳作業的目的是要檢查是否可以完成實際的檔案上傳作業，必要時，也可以一併檢查伺服器驗證作業是否成功。

備註：目前，測試上傳作業僅適用於 Windows 平台上的 Flash Player。

處理檔案上傳的伺服器指令碼應該要有包含下列元素的 HTTP POST 要求：

- 值為 multipart/form-data 的 Content-Type。
- name 特質設為 "Filedata"，而且 filename 特質設為原始檔案名稱的 Content-Disposition。您可以傳遞一個值給 FileReference.upload() 方法中的 uploadDataFieldName 參數，指定自訂的 name 特質。
- 檔案的二進位內容。

以下是 HTTP POST 要求樣本：

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filedata"; filename="sushi.jpg"
Content-Type: application/octet-stream

Test File
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
(actual file data,,,)
```

下列 HTTP POST 要求樣本會傳送 api_sig、api_key 和 auth_token 等三個 POST 變數，並使用 "photo" 這個自訂上傳資料欄位名稱值：

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_sig"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_key"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="auth_token"

XXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="photo"; filename="sushi.jpg"
Content-Type: application/octet-stream

(actual file data,,,)
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7--
```

從伺服器下載檔案

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以讓使用者利用 FileReference.download() 方法從伺服器下載檔案，這個方法需要用到 request 和 defaultFileName 兩個參數。第一個參數為 URLRequest 物件，其中包含要下載之檔案的 URL。第二個參數是選擇性參數，它可以讓您指定顯示在「下載檔案」對話方塊中的預設檔案名稱。如果您省略第二個參數 defaultFileName，就會採用指定的 URL 檔名。

下列程式碼將從 SWF 檔案所在的相同目錄中下載 index.xml 檔案：

```
var request:URLRequest = new URLRequest("index.xml");
var fileRef:FileReference = new FileReference();
fileRef.download(request);
```

如果不想使用 index.xml，而想以 currentnews.xml 做為預設名稱，請指定 defaultFileName 參數，如下列程式碼片段所示：

```
var request:URLRequest = new URLRequest("index.xml");
var fileToDownload:FileReference = new FileReference();
fileToDownload.download(request, "currentnews.xml");
```

當伺服器檔案名稱不具意義或是由伺服器自行產生時，重新命名檔案是不錯的做法。而且，在不直接下載檔案而使用伺服器端指令碼下載檔案的情況下，明確指定 defaultFileName 參數也是很好的處理方式。例如，當伺服器端指令碼是依據傳遞給它的 URL 變數下載特定的檔案時，您就必須指定 defaultFileName 參數。否則，已下載檔案的預設名稱將是您伺服器端指令碼的名稱。

您也可以將參數附加到 URL，藉由使用 download() 方法，將資料傳送給伺服器，以供伺服器指令碼進行剖析。下列 ActionScript 3.0 程式碼片段將會依據傳遞給 ColdFusion 指令碼的參數下載文件：

```
package
{
    import flash.display.Sprite;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.net.URLRequestMethod;
    import flash.net.URLVariables;

    public class DownloadFileExample extends Sprite
    {
        private var fileToDownload:FileReference;
        public function DownloadFileExample()
        {
            var request:URLRequest = new URLRequest();
            request.url = "http://www.[yourdomain].com/downloadfile.cfm";
            request.method = URLRequestMethod.GET;
            request.data = new URLVariables("id=2");
            fileToDownload = new FileReference();
            try
            {
                fileToDownload.download(request, "file2.txt");
            }
            catch (error:Error)
            {
                trace("Unable to download file.");
            }
        }
    }
}
```

下列程式碼將示範 ColdFusion 指令碼的 download.cfm 如何根據 URL 變數的值，從伺服器的兩個檔案中下載其中一個檔案：

```
<cfparam name="URL.id" default="1" />
<cfswitch expression="#URL.id#">
    <cfcase value="2">
        <cfcontent type="text/plain" file="#ExpandPath('two.txt')#" deletefile="No" />
    </cfcase>
    <cfdefaultcase>
        <cfcontent type="text/plain" file="#ExpandPath('one.txt')#" deletefile="No" />
    </cfdefaultcase>
</cfswitch>
```

FileReferenceList 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

FileReferenceList 類別可以讓使用者選取一個或多個檔案以上傳至伺服器端指令碼。檔案上傳作業是由 FileReference.upload() 方法處理，使用者所選取的每一個檔案都必須呼叫這個方法。

下列程式碼將建立兩個 FileFilter 物件 (imageFilter 和 textFilter)，並將它們以陣列形式傳遞給 FileReferenceList.browse() 方法。作業系統檔案對話方塊中就會顯示兩種類型的檔案篩選。

```
var imageFilter:FileFilter = new FileFilter("Image Files (*.jpg, *.jpeg, *.gif, *.png)", "*.jpg; *.jpeg;
*.gif; *.png");
var textFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt; *.rtf");
var fileRefList:FileReferenceList = new FileReferenceList();
try
{
    var success:Boolean = fileRefList.browse(new Array(imageFilter, textFilter));
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
```

雖然 FileReferenceList 類別允許您選取一個以上的檔案，但是使用 FileReferenceList 類別允許使用者選取並上傳一個或多個檔案，與使用 FileReference.browse() 選取檔案的方式並無不同。在上傳多個檔案時，您必須使用 FileReference.upload() 來上傳每一個選取的檔案，如下列程式碼所示：

```
var fileRefList:FileReferenceList = new FileReferenceList();
fileRefList.addEventListener(Event.SELECT, selectHandler);
fileRefList.browse();

function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/fileUploadScript.cfm");
    var file:FileReference;
    var files:FileReferenceList = FileReferenceList(event.target);
    var selectedFileArray:Array = files.fileList;
    for (var i:uint = 0; i < selectedFileArray.length; i++)
    {
        file = FileReference(selectedFileArray[i]);
        file.addEventListener(Event.COMPLETE, completeHandler);
        try
        {
            file.upload(request);
        }
        catch (error:Error)
        {
            trace("Unable to upload files.");
        }
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

由於已經將 Event.COMPLETE 事件加入陣列中的每一個 FileReference 物件，因此 Flash Player 會在完成每一項檔案上傳作業後呼叫 completeHandler() 方法。

使用 AIR 檔案系統 API

Adobe AIR 1.0 以及更新的版本

Adobe AIR 檔案系統 API 包含下列類別：

- 檔案
- FileMode
- FileStream

檔案系統 API 可讓您執行下列動作 (及其他更多)：

- 複製、建立、刪除，以及移動檔案和目錄
- 取得檔案與目錄的相關資訊
- 讀取和寫入檔案

AIR 檔的基本概念

Adobe AIR 1.0 以及更新的版本

如需瞭解在 AIR 中使用檔案系統的快速說明與程式碼範例，請參閱 Adobe Developer Connection 中的快速入門文章：

- 建置文字檔案編輯器 (Flash)
- 建置文字檔案編輯器 (Flex)
- 建置目錄搜尋應用程式 (Flex)
- 從 XML 偏好設定檔案讀取和寫入資料 (Flex)
- 壓縮檔案和資料 (Flex)

Adobe AIR 可提供您用來存取、建立和管理檔案及資料夾的類別。這些類別都包含在 flash.filesystem 套件中，使用方式如下所述：

File 類別	說明
File	File 物件代表檔案或目錄的路徑。您可以使用 File 物件，建立指向檔案或資料夾的指標，起始與該檔案或資料夾之間的互動。
FileMode	FileMode 類別會定義在 FileStream 類別 open() 和 openAsync() 方法的 fileMode 參數中使用的字串常數。這些方法的 fileMode 參數會決定當檔案開啟時 FileStream 物件的可用功能，其中包括寫入、讀取、附加及更新功能。
FileStream	FileStream 物件會用來開啟檔案以供讀取及寫入。一旦您建立 File 物件並指向新的或現有檔案，就會將該指標傳送至 FileStream 物件，如此一來，您就可以開啟檔案，並且讀取資料或寫入資料。

File 類別中的某些方法具有同步與非同步版本兩者：

- File.copyTo() 和 File.copyToAsync()
- File.deleteDirectory() 和 File.deleteDirectoryAsync()
- File.deleteFile() 和 File.deleteFileAsync()
- File.getDirectoryListing() 和 File.getDirectoryListingAsync()
- File.moveTo() 和 File.moveToAsync()
- File.moveToTrash() 和 File.moveToTrashAsync()

此外，FileStream 作業會根據下列 FileStream 物件開啟檔案的方式，以同步或非同步方式運作：藉由呼叫 open() 方法或呼叫 openAsync() 方法來開啟。

非同步版本可以讓您初始化在背景執行的程序，並且在完成時 (或者錯誤事件發生時) 傳送事件。在這些非同步背景程序執行期間，可以執行其它程式碼。執行非同步版本作業時，您必須設定事件偵聽程式函數，方法是使用呼叫該函數的 File 或 FileStream 物件的 addEventListener() 方法。

同步版本可以讓您撰寫更簡單的程式碼，而不需設定事件偵聽程式。不過，由於無法在同步方法執行期間執行其它程式碼，因此會延遲重要的程序 (例如呈現顯示物件和製作動畫)。

在 AIR 中使用 File 物件

Adobe AIR 1.0 以及更新的版本

File 物件是指向檔案系統中檔案或資料夾的指標。

File 類別會擴充 FileReference 類別。FileReference 類別 (在 Adobe® Flash® Player 和 AIR 中都可以使用) 代表檔案的指標。File 類別會加入 Flash Player (在瀏覽器中執行的 SWF 檔) 基於安全考量而未公開的屬性和方法。

關於 File 類別

Adobe AIR 1.0 以及更新的版本

您可以使用 File 類別執行下列工作：

- 取得特殊目錄 (包括使用者目錄、使用者的文件目錄、應用程式的啟動目錄，以及應用程式目錄) 的路徑
- 複製檔案及目錄
- 移動檔案及目錄
- 刪除檔案及目錄 (或者將它們移至垃圾桶)
- 列出某個目錄中包含的檔案及目錄
- 建立暫存檔案及資料夾

一旦 File 物件指向某個檔案路徑，您就可以使用 FileStream 類別，經由這個路徑來讀取和寫入檔案資料。

File 物件可以指向還不存在的檔案或目錄路徑。因此，您可以使用這類 File 物件建立檔案或目錄。

File 物件的路徑

Adobe AIR 1.0 以及更新的版本

每一個 File 物件都有兩個屬性，每一個屬性都會定義自己的路徑：

屬性	說明
nativePath	指定檔案的平台專屬路徑。例如，在 Windows 中的路徑會是「c:\Sample directory\test.txt」；而在 Mac OS 中則是「/Sample directory/test.txt」。在 Windows 中，nativePath 屬性會使用反斜線 (\) 字元做為目錄分隔符號字元；而在 Mac OS 和 Linux 中，則是使用正斜線 (/) 字元。
url	這個屬性會使用 file URL 配置來指向檔案。例如，在 Windows 中的路徑會是「file:///c:/Sample%20directory/test.txt」；而在 Mac OS 中則是「file:///Sample%20directory/test.txt」。除了 file 以外，執行階段還包括其它特殊的 URL 配置。如需詳細資訊，請參閱第 583 頁「支援的 AIR URL 配置」。

File 類別包含可在 Mac OS、Windows 和 Linux 中指向標準目錄的靜態屬性。這些屬性包括：

- File.applicationStorageDirectory — 每個已安裝 AIR 應用程式唯一的儲存目錄。此目錄是儲存動態應用程式資源和使用者偏好設定的適當位置。請考慮在其他位置儲存大量的資料。

  在 Android 和 iOS 上，當應用程式解除安裝或是使用者選擇清除應用程式資料時，會移除應用程式儲存目錄，但是在其他平台上則不是這樣。

- File.applicationDirectory— 安裝應用程式 (與任何其他已安裝的資產) 的目錄在某些作業系統上，會將應用程式儲存在單一套件檔案中，而不是實體目錄中。在此情況下，可能無法使用原生路徑來存取內容。應用程式目錄是唯讀。
- File.desktopDirectory — 使用者桌面目錄。如果平台未定義桌面目錄，則會使用檔案系統上的其他位置。
- File.documentsDirectory — 使用者文件目錄。如果平台未定義文件目錄，則會使用檔案系統上的其他位置。

- File.userDirectory — 使用者目錄。如果平台未定義使用者目錄，則會使用檔案系統上的其他位置。

備註：當平台沒有為桌面、文件或使用者目錄定義標準位置時，File.documentsDirectory、File.desktopDirectory 和 File.userDirectory 可以參照相同的目錄。

這些屬性在不同作業系統上各有不同的值。例如，Mac 和 Windows 對於使用者的桌面目錄各有不同的原生路徑。不過，File.desktopDirectory 屬性會指向每個平台上適當的目錄路徑。若要撰寫可跨平台使用的應用程式，請使用此屬性做為基礎，讓應用程式參照到所用的其他目錄和檔案。然後使用 resolvePath() 方法來修改路徑。例如，此程式碼會指向應用程式儲存目錄中的 preferences.xml 檔案：

```
var prefsFile:File = File.applicationStorageDirectory;
prefsFile = prefsFile.resolvePath("preferences.xml");
```

雖然 File 類別可讓您指向特定檔案路徑，但這麼做會讓應用程式無法跨平台使用。例如，C:\Documents and Settings\joe\ 這個路徑僅適用於 Windows。因此，最好使用 File 類別的靜態屬性，如 File.documentsDirectory。

一般目錄位置

平台	目錄類型	一般檔案系統位置
Android	應用程式	/data/data/
	應用程式儲存區	/data/data/air.applicationID/filename/Local Store
	快取	/data/data/applicationID/cache
	桌上型	/mnt/sdcard
	文件	/mnt/sdcard
	暫存	/data/data/applicationID/cache/FlashTmp.randomString
	使用者	/mnt/sdcard
iOS	應用程式	/var/mobile/Applications/uid/filename.app
	應用程式儲存區	/var/mobile/Applications/uid/Library/Application Support/applicationID/Local Store
	快取	/var/mobile/Applications/uid/Library/Caches
	桌上型	無法存取
	文件	/var/mobile/Applications/uid/Documents
	暫存	/private/var/mobile/Applications/uid/tmp/FlashTmpNNN
	使用者	無法存取
Linux	應用程式	/opt/filename/share
	應用程式儲存區	/home/userName/.appdata/applicationID/Local Store
	桌上型	/home/userName/Desktop
	文件	/home/userName/Documents
	暫存	/tmp/FlashTmp.randomString
	使用者	/home/userName
Mac	應用程式	/Applications/filename.app/Contents/Resources
	應用程式儲存區	/Users/*userName*/Library/Preferences/*applicationid*/Local Store (AIR 3.2 和更早版本) *path*/Library/Application Support/*applicationid*/Local Store (AIR 3.3 和更新版本)，其中 path 是 /Users/*userName*/Library/Containers/*bundle-id*/Data (安全執行程序的環境) 或 /Users/userName (若在安全執行程序環境外執行)
	快取	/Users/userName/Library/Caches
	桌面	/Users/*userName*/Desktop
	文件	/Users/*userName*/Documents
	暫存	/private/var/folders/JY/randomString/TemporaryItems/FlashTmp
	使用者	/Users/*userName*

平台	目錄類型	一般檔案系統位置
Windows	應用程式	C:\Program Files\filename
	應用程式儲存區	C:\Documents and settings\userName\ApplicationData\applicationID\Local Store
	快取	C:\Documents and settings\userName\Local Settings\Temp
	桌上型	C:\Documents and settings\userName\Desktop
	文件	C:\Documents and settings\userName\My Documents
	暫存	C:\Documents and settings\userName\Local Settings\Temp\randomString.tmp
	使用者	C:\Documents and settings\userName

這些目錄的實際原生路徑會隨作業系統與電腦組態而異。在這個表格顯示的路徑為一般範例。您應該總是使用適當的靜態 File 類別屬性以參照這些目錄，這樣您的應用程式即可在任何平台上正常運作。在實際的 AIR 應用程式中，會從應用程式描述器取得表格中顯示的 applicationID 與 filename 值。如果您在應用程式描述器中指定發行者 ID，那麼發行者 ID 會附加至這些路徑中的應用程式 ID。userName 的值是安裝使用者的帳戶名稱。

將 File 物件指向目錄

Adobe AIR 1.0 以及更新的版本

有多種方式可以將 File 物件設定為指向某個目錄。

指向使用者的主目錄

Adobe AIR 1.0 以及更新的版本

您可以將 File 物件指向使用者的主目錄。下列程式碼會將 File 物件設定為指向主目錄的 AIR Test 子目錄：

```
var file:File = File.userDirectory.resolvePath("AIR Test");
```

指向使用者的文件目錄

Adobe AIR 1.0 以及更新的版本

您可以將 File 物件指向使用者的文件目錄。下列程式碼會將 File 物件設定為指向文件目錄的 AIR Test 子目錄：

```
var file:File = File.documentsDirectory.resolvePath("AIR Test");
```

指向桌面目錄

Adobe AIR 1.0 以及更新的版本

您可以將 File 物件指向桌面。下列程式碼會將 File 物件設定為指向桌面的 AIR Test 子目錄：

```
var file:File = File.desktopDirectory.resolvePath("AIR Test");
```

指向應用程式儲存目錄

Adobe AIR 1.0 以及更新的版本

您可以將 File 物件指向應用程式儲存目錄。對於每一個 AIR 應用程式而言，都有一個可定義應用程式儲存目錄的唯一相關聯路徑。這個目錄對於每一個應用程式與使用者而言都具有唯一性。您可以使用這個目錄，儲存使用者和應用程式的專屬資料 (例如使用者資料或偏好設定檔案)。例如，下列程式碼會將 File 物件指向 prefs.xml 這個偏好設定檔案，這個檔案位於應用程式儲存目錄中：

```
var file:File = File.applicationStorageDirectory;
file = file.resolvePath("prefs.xml");
```

這個應用程式儲存目錄的位置通常是根據使用者名稱和應用程式 ID 而定。此處所提供的下列檔案系統位置是用以協助您進行應用程式除錯。您永遠都應該使用 File.applicationStorage 屬性或 app-storage: URI 配置，來解析此目錄中的檔案：

- 在 Mac OS 上 — 依 AIR 版本而不同：

  AIR 3.2 和更早版本：/Users/user name/Library/Preferences/***applicationID***/Local Store/

  AIR 3.3 和更新版本：***path***/Library/Application Support/***applicationID***/Local Store，其中 ***path*** 是 /Users/***username***/Library/Containers/***bundle-id***/Data (安全執行程序的環境) 或 /Users/***username*** (若在安全執行程序環境外執行)

  例如 (AIR 3.2)：

  ```
  /Users/babbage/Library/Preferences/com.example.TestApp/Local Store
  ```

- 在 Windows 中，則是位於 Documents and Settings 目錄內，位置為：

  C:\Documents and Settings\user name\Application Data\applicationID\Local Store\

  例如：

  ```
  C:\Documents and Settings\babbage\Application Data\com.example.TestApp\Local Store
  ```

- Linux— 位於：

  /home/ 使用者名稱 /.appdata/applicationID/Local Store/

  例如：

  ```
  /home/babbage/.appdata/com.example.TestApp/Local Store
  ```

- Android— 位於：

  /data/data/androidPackageID/applicationID/Local Store

  例如：

  ```
  /data/data/air.com.example.TestApp/com.example.TestApp/Local Store
  ```

備註：如果應用程式具有發行者 ID，則發行者 ID 也會做為應用程式儲存目錄路徑的一部分。

使用 File.applicationStorageDirectory 建立之 File 物件的 URL (以及 url 屬性) 會使用 app-storage URL 配置 (請參閱第 583 頁「支援的 AIR URL 配置」)，如下所示：

```
var dir:File = File.applicationStorageDirectory;
dir = dir.resolvePath("preferences");
trace(dir.url); // app-storage:/preferences
```

指向應用程式目錄

Adobe AIR 1.0 以及更新的版本

您可以將 File 物件指向應用程式的安裝目錄 (也就是應用程式目錄)。您可以使用 File.applicationDirectory 屬性參考這個目錄。您可以使用這個目錄，檢查與應用程式一併安裝的應用程式描述器檔案或其它資源。例如，下列程式碼會將 File 物件指向應用程式目錄中名為 images 的目錄：

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("images");
```

使用 File.applicationDirectory 建立之 File 物件的 URL (以及 url 屬性) 會使用 app URL 配置 (請參閱第 583 頁「支援的 AIR URL 配置」)，如下所示：

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("images");
trace(dir.url); // app:/images
```

備註：在 Android 上，應用程式套件中的檔案無法透過 nativePath 來存取。nativePath 屬性是空的字串。永遠使用 URL 存取應用程式目錄中的檔案，而非使用原生路徑。

指向快取目錄

Adobe AIR 3.6 以及更新的版本

您可以使用 File.cacheDirectory 屬性，將 File 物件指向作業系統的暫存或快取目錄。這個目錄包含的暫存檔案並非應用程式執行所需的檔案，而且若遭到刪除，也不會對使用者造成任何問題或資料遺失。

在大多數的作業系統中，快取目錄屬於暫存目錄。在 iOS 上，快取目錄相當於應用程式元件庫的 Caches 目錄。這個目錄中的檔案不會備份到線上儲存空間，並且若裝置的可用儲存空間不足，很可能會遭到作業系統刪除。如需詳細資訊，請參閱第 583 頁「控制檔案備份和快取」。

指向檔案系統根目錄

Adobe AIR 1.0 以及更新的版本

File.getRootDirectories() 方法會列出 Windows 電腦中的所有根磁碟，例如 C: 和掛接式磁碟。在 Mac OS 和 Linux 中，這個方法一定會傳回電腦的唯一根目錄 (即「/」目錄)。StorageVolumeInfo.getStorageVolumes() 方法提供掛接式儲存體磁碟區的詳細資訊 (請參閱第 592 頁「使用儲存空間」)。

備註：在 Android 上無法讀取檔案系統的根目錄。會傳回參照具有原生路徑「/」之目錄的 File 物件，但是該物件的屬性沒有精確的值。例如，spaceAvailable 永遠是 0。

指向明確的目錄

Adobe AIR 1.0 以及更新的版本

您可以設定 File 物件的 nativePath 屬性，將 File 物件指向明確的目錄，如下列範例所示 (適用於 Windows)：

```
var file:File = new File();
file.nativePath = "C:\\AIR Test";
```

重要事項：以這種方式指向明確路徑，將使程式碼無法跨平台使用。例如，上一個範例只能在 Windows 上運作。您可使用 File 物件的靜態屬性，如 File.applicationStorageDirectory，以找出可跨平台運作的目錄。再使用 resolvePath() 方法 (請參閱下一節) 瀏覽至相對路徑。

瀏覽至相對路徑

Adobe AIR 1.0 以及更新的版本

您可以使用 resolvePath() 方法，取得相對於其它指定之路徑的路徑。例如，下列程式碼會將 File 物件設定為指向使用者之主目錄的「AIR Test」子目錄：

```
var file:File = File.userDirectory;
file = file.resolvePath("AIR Test");
```

您也可以使用 File 物件的 url 屬性，將物件指向 URL 字串架構的目錄，如下所示：

```
var urlStr:String = "file:///C:/AIR Test/";
var file:File = new File()
file.url = urlStr;
```

如需詳細資訊，請參閱第 583 頁「修改 File 的路徑」。

讓使用者瀏覽以選取目錄

Adobe AIR 1.0 以及更新的版本

File 包含 browseForDirectory() 方法，這個方法可以產生系統對話方塊，讓使用者在其中選取目錄指定給該物件。browseForDirectory() 是非同步方法。如果使用者選取目錄並按一下「開啟」按鈕，File 物件便會傳送 select 事件；而如果使用者按一下「取消」按鈕，則會傳送 cancel 事件。

例如，下列程式碼會讓使用者選取某個目錄，並且在選取之後立即輸出該目錄的路徑：

```
var file:File = new File();
file.addEventListener(Event.SELECT, dirSelected);
file.browseForDirectory("Select a directory");
function dirSelected(e:Event):void {
    trace(file.nativePath);
}
```

備註：在 Android 上，不支援 browseForDirectory() 方法。呼叫此方法不會發生作用；會立即傳送取消事件。要讓使用者選取目錄，請改用自訂且應用程式定義的對話方塊。

指向應用程式的叫用目錄

Adobe AIR 1.0 以及更新的版本

您可以檢查在叫用應用程式時傳送之 InvokeEvent 物件的 currentDirectory 屬性，取得應用程式的叫用目錄位置。如需詳細資訊，請參閱第 757 頁「擷取命令列引數」。

將 File 物件指向檔案

Adobe AIR 1.0 以及更新的版本

有多種方式可以設定 File 物件指向的目標檔案。

指向明確檔案路徑

Adobe AIR 1.0 以及更新的版本

重要事項：指向明確路徑，將使程式碼無法跨平台使用。例如，C:/foo.txt 這個路徑僅適用於 Windows。您可使用 File 物件的靜態屬性，如 File.applicationStorageDirectory，以找出可跨平台運作的目錄。再使用 resolvePath() 方法 (請參閱第 583 頁「修改 File 的路徑」) 瀏覽至相對路徑。

您可以使用 File 物件的 url 屬性，將物件指向 URL 字串架構的目錄或檔案，如下所示：

```
var urlStr:String = "file:///C:/AIR Test/test.txt";
var file:File = new File()
file.url = urlStr;
```

您也可以將 URL 傳遞給 File() 建構函數，如下所示：

```
var urlStr:String = "file:///C:/AIR Test/test.txt";
var file:File = new File(urlStr);
```

url 屬性一定會傳回經過 URI 編碼的 URL (例如，以 "%20 取代空格)：

```
file.url = "file:///c:/AIR Test";
trace(file.url); // file:///c:/AIR%20Test
```

您也可以使用 File 物件的 nativePath 屬性來設定明確的路徑。例如，下列程式碼在 Windows 電腦中執行時，會將 File 物件設定為指向 C: 磁碟之 AIR Test 子目錄中的 test.txt 檔案：

```
var file:File = new File();
file.nativePath = "C:/AIR Test/test.txt";
```

您也可以將這個路徑傳遞給 File() 建構函數，如下所示：

```
var file:File = new File("C:/AIR Test/test.txt");
```

使用正斜線 (/) 字元做為 nativePath 屬性的路徑分隔符號。在 Windows 上，您亦可使用反斜線 (\) 字元，但這樣會導致應用程式無法跨平台使用。

如需詳細資訊，請參閱第 583 頁 「修改 File 的路徑」。

列舉目錄中的檔案

Adobe AIR 1.0 以及更新的版本

您可以使用 File 物件的 getDirectoryListing() 方法，取得指向根層級目錄中檔案和子目錄的 File 物件陣列。如需詳細資訊，請參閱第 588 頁 「列舉目錄」。

讓使用者瀏覽以選取檔案

Adobe AIR 1.0 以及更新的版本

File 類別包含下列方法，這些方法可以產生系統對話方塊，讓使用者在其中選取檔案指定給該物件：

- browseForOpen()
- browseForSave()
- browseForOpenMultiple()

這些方法都是非同步方法。當使用者選取檔案 (對於 browseForSave()，則是目標路徑) 時，browseForOpen() 和 browseForSave() 方法便會傳送 select 事件。使用 browseForOpen() 和 browseForSave() 方法時，一旦選取項目之後，目標 File 物件就會立即指向選取的檔案。當使用者選取多個檔案時，browseForOpenMultiple() 方法便會傳送 selectMultiple 事件。selectMultiple 事件的類型為 FileListEvent，這個事件具有 files 屬性，該屬性是 File 物件陣列 (指向選取的檔案)。

例如，下列程式碼會將 「開啟」對話方塊呈現在使用者面前，讓使用者可以在其中選取檔案：

```
var fileToOpen:File = File.documentsDirectory;
selectTextFile(fileToOpen);

function selectTextFile(root:File):void
{
    var txtFilter:FileFilter = new FileFilter("Text", "*.as;*.css;*.html;*.txt;*.xml");
    root.browseForOpen("Open", [txtFilter]);
    root.addEventListener(Event.SELECT, fileSelected);
}

function fileSelected(event:Event):void
{
    trace(fileToOpen.nativePath);
}
```

當您呼叫瀏覽方法時，如果應用程式已經開啟其它瀏覽器對話方塊，執行階段就會擲回 Error 例外。

備註：在 Android 上，只能使用 browseForOpen() 與 browseForOpenMultiple() 方法來選取音效檔。雖然使用者可以輸入任意的檔案名稱，browseForSave() 對話方塊仍然只會顯示媒體檔。若要開啟和儲存非媒體檔案，您應該考慮使用自訂對話方塊以取代這些方法。

修改 File 的路徑

Adobe AIR 1.0 以及更新的版本

您也可以呼叫 resolvePath() 方法，或者修改現有 File 物件的 nativePath 或 url 屬性，修改 File 物件的路徑，如下列範例所示 (適用於 Windows)：

```
var file1:File = File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
trace(file1.nativePath); // C:\Documents and Settings\userName\My Documents\AIR Test
var file2:File = File.documentsDirectory;
file2 = file2.resolvePath("..");
trace(file2.nativePath); // C:\Documents and Settings\userName
var file3:File = File.documentsDirectory;
file3.nativePath += "/subdirectory";
trace(file3.nativePath); // C:\Documents and Settings\userName\My Documents\subdirectory
var file4:File = new File();
file4.url = "file:///c:/AIR Test/test.txt";
trace(file4.nativePath); // C:\AIR Test\test.txt
```

使用 nativePath 屬性時，請使用正斜線 (/) 字元，做為路徑分隔字元。在 Windows 上，您亦可使用反斜線 (\) 字元，但不建議這麼做，因為這樣會導致程式碼無法跨平台運作。

支援的 AIR URL 配置

Adobe AIR 1.0 以及更新的版本

在 AIR 中定義 File 物件的 url 屬性時，您可以使用下列任何 URL 配置：

URL 配置	說明
file	用於指定相對於檔案系統根項目的路徑。例如： `file:///c:/AIR Test/test.txt` URL 標準會指定 file URL 必須採用 file://<host>/<path> 的形式。在特殊情況下，<host> 可以是空字串，這時就表示「正在解譯其 URL 的電腦」。因此，file URL 通常都有三條斜線 (///)。
app	用於指定相對於應用程式之安裝根目錄 (包含所安裝應用程式之 application.xml 檔案的目錄) 的路徑。例如，下列路徑會指向應用程式安裝目錄的 images 子目錄： `app:/images`
app-storage	用於指定相對於應用程式儲存目錄的路徑。針對每一個安裝的應用程式，AIR 都會定義唯一的應用程式儲存目錄，可以用來儲存某個應用程式專屬的資料。例如，下列路徑會指向應用程式儲存目錄之 settings 子目錄中的 prefs.xml 檔案： `app-storage:/settings/prefs.xml`

控制檔案備份和快取

Adobe AIR 3.6 及更新的版本、**iOS** 和僅 **OS X**

某些作業系統 (大多數是 iOS 和 Mac OS X) 會提供使用者自動將應用程式檔案備份到遠端儲存空間的功能。此外，在 iOS 上，備份檔案與否以及可在何處儲存不同用途的檔案，均有其限制。

下列摘要說明如何符合 Apple 的檔案備份和儲存原則。如需進一步資訊，請參閱下一章節。

- 若要指定檔案不需要備份且 (僅 iOS) 允許作業系統在裝置儲存空間不足時加以刪除，請將檔案儲存在快取目錄中 (File.cacheDirectory)。這是 iOS 上慣用的儲存位置，應可用於大多數可以重新產生或重新下載的檔案。

- 若要指定檔案不需要備份但不可由作業系統加以刪除，請將檔案儲存在其中一個應用程式元件庫目錄，例如應用程式儲存目錄 (File.applicationStorageDirectory) 或文件目錄 (File.documentsDirectory)。請將 File 物件的 preventBackup 屬性設定為 true。這是為了符合 Apple 對可以重新產生或再次下載的內容的要求，但也是為了讓您的應用程式在離線使用時正常運作的必要動作。

指定檔案進行備份

為了節省備份空間並減少網路頻寬使用，Apple 的 iOS 和 Mac 應用程式原則指定，應僅針對包含使用者所輸入資料或無法以其他方式重新產生或下載的資料的檔案進行備份。

根據預設，應用程式元件庫資料夾中的所有檔案都會備份。在 Mac OS X 上，這是應用程式儲存目錄。在 iOS 上，這包括應用程式儲存目錄、應用程式目錄、桌面目錄、文件目錄和使用者目錄 (因為那些目錄會對應至 iOS 上的應用程式元件庫資料夾)。因此，任何位於那些目錄的檔案都會根據預設備份到伺服器儲存空間。

如果您要將可由應用程式重新建立的檔案儲存在其中一個位置，應該為檔案設定旗標，好讓作業系統知道不要備份該檔案。若要指示檔案不可備份，請將 File 物件的 preventBackup 屬性設定為 true。

請注意，在 iOS 上，對於位於任何應用程式元件庫資料夾中的檔案，即使檔案的 preventBackup 屬性設定為 true，檔案還是會標記為作業系統不可刪除的永續性檔案。

控制檔案快取和刪除

Apple 的 iOS 應用程式原則指定，當裝置上的儲存空間不足時，作業系統應盡量刪除可重新產生的內容。

在 iOS 上，應用程式元件庫資料夾 (例如應用程式儲存目錄或文件目錄) 中的檔案會標記為永久檔案，並且不會遭到作業系統刪除。

對於可由應用程式重新產生並在儲存空間不足時加以刪除的檔案，請儲存在應用程式快取目錄中。您可以使用 File.cacheDirectory 靜態屬性來存取快取目錄。

在 iOS 上，快取目錄相當於應用程式的快取目錄 (<Application Home>/Library/Caches)。在其他作業系統上，這個目錄會對應至可相比的目錄。例如，在 Mac OS X 上，它也會對應至應用程式元件庫的 Caches 目錄。在 Android 上，快取目錄會對應至應用程式的快取目錄。在 Windows 上，快取目錄會對應至作業系統的 temp 目錄。在 Android 和 Windows 上，這是可透過呼叫 File 類別的 createTempDirectory() 和 createTempFile() 方法加以存取的相同目錄。

尋找兩個檔案之間的相對路徑

Adobe AIR 1.0 以及更新的版本

您可以使用 getRelativePath() 方法，尋找兩個檔案之間的相對路徑：

```
var file1:File = File.documentsDirectory.resolvePath("AIR Test");
var file2:File = File.documentsDirectory
file2 = file2.resolvePath("AIR Test/bob/test.txt");

trace(file1.getRelativePath(file2)); // bob/test.txt
```

getRelativePath() 方法的第二個參數 useDotDot 允許在結果中傳回 .. 語法，以指出上層目錄：

```
var file1:File = File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
var file2:File = File.documentsDirectory;
file2 = file2.resolvePath("AIR Test/bob/test.txt");
var file3:File = File.documentsDirectory;
file3 = file3.resolvePath("AIR Test/susan/test.txt");

trace(file2.getRelativePath(file1, true)); // ../..
trace(file3.getRelativePath(file2, true)); // ../../bob/test.txt
```

取得檔名的標準化版本

Adobe AIR 1.0 以及更新的版本

檔案和路徑名稱在 Windows 和 Mac OS 中不會區分大小寫。在下列範例中，兩個 File 物件會指向相同的檔案：

```
File.documentsDirectory.resolvePath("test.txt");
File.documentsDirectory.resolvePath("TeSt.TxT");
```

不過，文件和目錄名稱則包含大小寫。例如，下列範例會假設在 documents 目錄中有名為 AIR Test 的資料夾：

```
var file:File = File.documentsDirectory.resolvePath("AIR test");
trace(file.nativePath); // ... AIR test
file.canonicalize();
trace(file.nativePath); // ... AIR Test
```

canonicalize() 方法會轉換 nativePath 物件，為檔案或目錄名稱使用正確的大小寫。在區分大小寫的檔案系統上 (例如 Linux)，當同時存在多個僅名稱大小寫不同的檔案時，canonicalize() 方法就會調整路徑來對應所找到的第一個檔案 (尋找的順序由檔案系統決定)。

您也可以使用 canonicalize() 方法，將短檔名 (「8.3」名稱) 轉換為長檔名 (適用於 Windows)，如下列範例所示：

```
var path:File = new File();
path.nativePath = "C:\\AIR~1";
path.canonicalize();
trace(path.nativePath); // C:\AIR Test
```

使用套件和符號連結

Adobe AIR 1.0 以及更新的版本

各種作業系統都支援套件檔案和符號連結檔案：

套件 — 在 Mac OS 中，目錄可以指定為套件，並在 Mac OS Finder 中顯示為單一檔案，而非顯示為目錄。

符號連結 —Mac OS、Linux 和 Windows Vista 皆支援符號連結。符號連結可讓檔案指向磁碟上的另一個檔案或目錄。符號連結與別名相似，但卻不同。別名一定會回報為檔案 (而非目錄)，而且讀取或寫入別名或捷徑絕不會影響它所指向的原始檔案或目錄。另一方面，符號連結的行為和它所指向的符號或目錄完全相同。它可以回報為檔案或目錄，而且讀取或寫入符號連結會影響它所指向的原始檔案或目錄，而非符號連結本身。此外，在 Windows 中，對於參考連接點 (在 NTFS 檔案系統中使用) 的 File 物件，isSymbolicLink 屬性是設定為 true。

File 類別包含 isPackage 和 isSymbolicLink 屬性，可用於檢查 File 物件參考的是套件還是符號連結。

下列程式碼會重複執行使用者的桌面目錄，並列出 「不是」套件的子目錄：

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (desktopNodes[i].isDirectory && !desktopNodes[i].isPackage)
    {
        trace(desktopNodes[i].name);
    }
}
```

下列程式碼會重複執行使用者的桌面目錄，並列出 「不是」符號連結的檔案和目錄：

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (!desktopNodes[i].isSymbolicLink)
    {
        trace(desktopNodes[i].name);
    }
}
```

canonicalize() 方法會變更符號連結的路徑，使其指向該連結所參考的檔案或目錄。下列程式碼會重複執行使用者的桌面目錄，並回報本身為符號連結之檔案所參考的路徑：

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (desktopNodes[i].isSymbolicLink)
    {
        var linkNode:File = desktopNodes[i] as File;
        linkNode.canonicalize();
        trace(linkNode.nativePath);
    }
}
```

判斷磁碟的可用空間

Adobe AIR 1.0 以及更新的版本

File 物件的 spaceAvailable 屬性表示在 File 位置的可用空間 (以位元組為單位)。例如，下列程式碼會檢查應用程式儲存目錄中的可用空間：

```
trace(File.applicationStorageDirectory.spaceAvailable);
```

如果 File 物件參考目錄，則 spaceAvailable 屬性會指出目錄中檔案可使用的空間。如果 File 物件參考檔案，則 spaceAvailable 屬性會指出檔案可成長的空間。如果檔案位置不存在，spaceAvailable 屬性會設定為 0。如果 File 物件參考符號連結，則 spaceAvailable 屬性會設定為符號連結所指向位置的可用空間。

目錄或檔案的可用空間通常與包含目錄或檔案之磁碟中的可用空間相同。不過，可用空間可能會將帳戶配額和每個目錄限制納入考量。

將檔案或目錄加入至磁碟，通常需要比檔案實際大小或目錄內容大小更大的空間。例如，作業系統可能需要更多空間來儲存索引資訊。必要的磁區也可能會使用額外的空間。此外，可用空間會動態變更。因此，您無法預期為檔案儲存區配置 「所有」空間。如需有關在檔案系統中寫入資料的詳細資訊，請參閱第 593 頁 「讀取和寫入檔案」。

StorageVolumeInfo.getStorageVolumes() 方法提供掛接式儲存體磁碟區的詳細資訊 (請參閱第 592 頁 「使用儲存空間」)。

使用預設系統應用程式開啟檔案

Adobe AIR 2 以及更新的版本

在 AIR 2 中，您可以使用作業系統註冊的應用程式來開啟檔案。例如，AIR 應用程式可以使用註冊的應用程式來開啟 DOC 檔案。使用 File 物件的 openWithDefaultApplication() 方法開啟檔案。例如，以下程式碼使用 DOC 檔案的預設應用程式來開啟使用者桌面上名稱為 test.doc 檔案：

```
var file:File = File.deskopDirectory;
file = file.resolvePath("test.doc");
file.openWithDefaultApplication();
```

備註：在 Linux 上，檔案的 MIME 類型 (非副檔名) 會決定檔案的預設應用程式。

以下程式碼可以讓使用者瀏覽 mp3 檔案，然後使用 mp3 檔案的預設播放應用程式來開啟它。

```
var file:File = File.documentsDirectory;
var mp3Filter:FileFilter = new FileFilter("MP3 Files", "*.mp3");
file.browseForOpen("Open", [mp3Filter]);
file.addEventListener(Event.SELECT, fileSelected);

function fileSelected(e:Event):void
{
    file.openWithDefaultApplication();
}
```

您不可以將 openWithDefaultApplication() 方法用於應用程式目錄中的檔案。

AIR 會禁止您使用 openWithDefaultApplication() 方法開啟特定的檔案。在 Windows 上，AIR 會禁止開啟特定檔案類型的檔案，例如 EXE 或 BAT。在 Mac OS 和 Linux 上，AIR 會讓您無法開啟以特定應用程式啟動的檔案 (這些包括 Mac OS 上的 Terminal 和 AppletLauncher，以及 Linux 上的 csh、bash 或 ruby)。嘗試使用 openWithDefaultApplication() 方法開啟這些檔案會產生例外。如需禁止的檔案類型詳細清單，請參閱 File.openWithDefaultApplication() 方法的語言參考項目。

備註：此限制不適用於使用原生安裝程式 (延伸桌面應用程式) 安裝的 AIR 應用程式。

取得檔案系統資訊

Adobe AIR 1.0 以及更新的版本

File 類別包含下列靜態屬性，可提供檔案系統的某些實用資訊：

屬性	說明
File.lineEnding	主機作業系統使用的行尾字元順序。在 Mac OS 和 Linux 中，這是換行字元。在 Windows 中，這是歸位字元，後面接著換行字元。
File.separator	主機作業系統的路徑組件分隔字元。在 Mac OS 和 Linux 中，這是正斜線 (/) 字元。在 Windows 中，這是反斜線 (\) 字元。
File.systemCharset	主機作業系統針對檔案使用的預設編碼方式。這個編碼方式與作業系統針對不同語言所使用的字元集有關。

Capabilities 類別也包含有用的系統資訊，使用這些檔案時可以參考這些資訊：

屬性	說明
Capabilities.hasIME	指定播放程式是在有安裝 (true) 或未安裝 (false) 輸入法編輯器 (IME) 的系統上執行。
Capabilities.language	指定執行播放程式之系統的語言碼。
Capabilities.os	指定目前的作業系統。

備註：使用 Capabilities.os 判別系統特性時請小心。若有更特定的屬性能判斷系統特性，請使用該屬性。否則您所撰寫的程式碼有可能無法在所有平台上正確運作。例如，以下列程式碼為例：

```
var separator:String;
if (Capablities.os.indexOf("Mac") > -1)
{
    separator = "/";
}
else
{
    separator = "\\";
}
```

此程式碼在 Linux 上會產生問題。最好僅使用 File.separator 屬性。

使用目錄

Adobe AIR 1.0 以及更新的版本

執行階段所具備的功能可以讓您使用本機檔案系統中的目錄。

如需有關建立指向目錄之 File 物件的詳細資訊，請參閱第 578 頁 「將 File 物件指向目錄」。

建立目錄

Adobe AIR 1.0 以及更新的版本

File.createDirectory() 方法可以讓您建立目錄。例如，下列程式碼會建立名為 AIR Test 的目錄，做為使用者主目錄的子目錄。

```
var dir:File = File.userDirectory.resolvePath("AIR Test");
dir.createDirectory();
```

如果這個目錄已經存在，則 createDirectory() 方法便不會有任何作用。

此外，在某些模式下，FileStream 物件會在開啟檔案時建立目錄。當您初始化 FileStream 實體並將 FileStream() 建構函式的 fileMode 參數設定為 FileMode.APPEND 或 FileMode.WRITE 時，便會建立遺漏的目錄。如需詳細資訊，請參閱第 594 頁 「讀取和寫入檔案的工作流程」。

建立暫存目錄

Adobe AIR 1.0 以及更新的版本

File 類別包含 createTempDirectory() 方法，可以用來在 「系統」的暫存目錄資料夾中建立新目錄，如下列範例所示：

```
var temp:File = File.createTempDirectory();
```

createTempDirectory() 會自動建立唯一的暫存目錄 (可以節省您為了尋找新的唯一位置而要花費的功夫)。

您可以使用暫存目錄，暫時儲存應用程式某個工作階段使用的暫存檔案。請注意，您可以使用 createTempFile() 方法，在 「系統」的暫存目錄中建立新的唯一暫存檔案。

在關閉應用程式之前，您可能會想要刪除這個暫存目錄，因為系統 「不會」自動刪除所有裝置上的該目錄。

列舉目錄

Adobe AIR 1.0 以及更新的版本

您可以使用 File 物件的 getDirectoryListing() 或 getDirectoryListingAsync() 方法，取得指向某個目錄中檔案和子資料夾的 File 物件陣列。

例如，下列程式碼會列出使用者的文件目錄內容 (而不會檢查子目錄)：

```
var directory:File = File.documentsDirectory;
var contents:Array = directory.getDirectoryListing();
for (var i:uint = 0; i < contents.length; i++)
{
    trace(contents[i].name, contents[i].size);
}
```

使用這個方法的非同步版本時，directoryListing 事件物件具有 files 屬性，這個屬性是與目錄有關的 File 物件陣列：

```
var directory:File = File.documentsDirectory;
directory.getDirectoryListingAsync();
directory.addEventListener(FileListEvent.DIRECTORY_LISTING, dirListHandler);

function dirListHandler(event:FileListEvent):void
{
    var contents:Array = event.files;
    for (var i:uint = 0; i < contents.length; i++)
    {
        trace(contents[i].name, contents[i].size);
    }
}
```

複製和移動目錄

Adobe AIR 1.0 以及更新的版本

您可以複製或移動目錄，只要使用與您用來複製或移動檔案的相同方法即可。例如，下列程式碼會以同步方式複製目錄：

```
var sourceDir:File = File.documentsDirectory.resolvePath("AIR Test");
var resultDir:File = File.documentsDirectory.resolvePath("AIR Test Copy");
sourceDir.copyTo(resultDir);
```

當您將 copyTo() 方法的 overwrite 參數設定為 true 時，現有目標目錄中的所有檔案及資料夾都會遭刪除，並且以來源目錄中的檔案及資料夾取代 (即使來源目錄中沒有目標檔案也一樣)。

您為 copyTo() 方法的 newLocation 參數設定的目錄是指定將產生之目錄的路徑，而「不是」指定會包含將產生之目錄的「上層」目錄。

如需詳細資訊，請參閱第 591 頁「複製和移動檔案」。

刪除目錄內容

Adobe AIR 1.0 以及更新的版本

File 類別包含 deleteDirectory() 和 deleteDirectoryAsync() 方法。這兩個方法可用來刪除目錄，前者會以同步方式運作，而後者則是以非同步方式運作 (請參閱：第 574 頁「AIR 檔的基本概念」)。這兩個方法都包含 deleteDirectoryContents 參數 (採用 Boolean 值)，設定為 true (預設值為 false) 時，呼叫這個方法會刪除非空白目錄，否則只會刪除空白目錄。

例如，下列程式碼會以同步方式刪除使用者文件目錄的 AIR Test 子目錄：

```
var directory:File = File.documentsDirectory.resolvePath("AIR Test");
directory.deleteDirectory(true);
```

下列程式碼會以非同步方式刪除使用者文件目錄的 AIR Test 子目錄：

```
var directory:File = File.documentsDirectory.resolvePath("AIR Test");
directory.addEventListener(Event.COMPLETE, completeHandler)
directory.deleteDirectoryAsync(true);

function completeHandler(event:Event):void {
    trace("Deleted.")
}
```

File 類別還包含 moveToTrash() 和 moveToTrashAsync() 方法，可以讓您用來將目錄移至系統的垃圾桶。如需詳細資訊，請參閱第 592 頁「將檔案移至垃圾桶」。

處理檔案

Adobe AIR 1.0 以及更新的版本

您可以利用 AIR 檔案 API，在應用程式中加入基本的檔案互動功能。例如，您可以讀取和寫入檔案、複製和刪除檔案等等。由於應用程式會存取本機檔案系統，因此請先參閱：第 928 頁 「AIR 安全性」。

備註：您可以將某個檔案類型與 AIR 應用程式產生關聯 (如此只要按兩下該檔案，即可開啟 AIR 應用程式)。如需詳細資訊，請參閱第 764 頁 「管理檔案關聯」。

取得檔案資訊

Adobe AIR 1.0 以及更新的版本

File 類別包含下列屬性，這些屬性可提供 File 物件所指向檔案或目錄的相關資訊：

File 屬性	說明
creationDate	本機磁碟中檔案的建立日期。
creator	已過時。請使用 extension 屬性 (這個屬性會回報檔案的 Macintosh 建立者類型，只能在 Mac OS X 之前的 Mac OS 版本中使用)。
downloaded	(AIR 2 以及更新的版本) 指出是否已下載參考的檔案或目錄 (從網際網路)。只有在可以將檔案標記為已下載的作業系統上，本屬性才有意義，如下： • Windows XP Service Pack 2 和更新版本，以及在 Windows Vista 上 • Mac OS 10.5 與更新的版本
exists	參考的檔案或目錄是否存在。
extension	副檔名，也就是最後一個點 (".") 字元後面 (不含 "." 本身) 的名稱部分。如果檔案名稱中沒有點字元，則副檔名為 null。
icon	包含為檔案所定義之圖示的 Icon 物件。
isDirectory	File 物件是否參考某個目錄。
modificationDate	本機磁碟中最後修改之檔案或目錄的日期。
name	本機磁碟中檔案或目錄的名稱 (如果有副檔名，則包含該副檔名)。
nativePath	以主機作業系統形式表示的完整路徑。請參閱：第 575 頁 「File 物件的路徑」。
parent	包含由 File 物件所表示之資料夾或檔案的資料夾。如果 File 物件參考檔案系統根目錄中的檔案或目錄，這個屬性便為 null。
size	本機磁碟中的檔案大小，以位元組為單位。
type	已過時。請使用 extension 屬性 (在 Macintosh 中，這個屬性是四個字元的檔案類型，只能在 Mac OS X 之前的 Mac OS 版本中使用)。
url	檔案或目錄的 URL。請參閱：第 575 頁 「File 物件的路徑」。

如需這些屬性的詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中的 File 類別項目。

複製和移動檔案

Adobe AIR 1.0 以及更新的版本

File 類別包含兩個用於複製檔案或目錄的方法：copyTo() 和 copyToAsync()。File 類別包含兩個用於移動檔案或目錄的方法：moveTo() 和 moveToAsync()。copyTo() 和 moveTo() 方法是以同步方式運作，而 copyToAsync() 和 moveToAsync() 方法則是以非同步方式運作 (請參閱：第 574 頁 「AIR 檔的基本概念」)。

若要複製或移動檔案，請設定兩個 File 物件。其中一個物件指向要複製或移動的檔案，而且它是呼叫複製或移動方法的物件；而另一個物件則指向目的地 (結果) 路徑。

下列程式碼會在使用者文件目錄的 AIR Test 子目錄中，將 test.txt 檔案複製成為相同目錄中名為的 copy.txt 檔案：

```
var original:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var newFile:File = File.resolvePath("AIR Test/copy.txt");
original.copyTo(newFile, true);
```

在上述範例中，copyTo() 方法的 overwrite 參數 (第二個參數) 設定為 true。將 overwrite 設定為 true 時，現有的目標檔案便會遭覆寫。這個參數是選擇性參數。如果您將這個參數設定為 false (預設值)，則當目標檔案存在時，作業會傳送 IOErrorEvent 事件 (並且不會複製檔案)。

「非同步」版本的複製和移動方法都是以非同步方式運作。您可以使用 addEventListener() 方法監視工作順利完成或發生的錯誤狀況，如下列程式碼所示：

```
var original = File.documentsDirectory;
original = original.resolvePath("AIR Test/test.txt");

var destination:File = File.documentsDirectory;
destination =  destination.resolvePath("AIR Test 2/copy.txt");

original.addEventListener(Event.COMPLETE, fileMoveCompleteHandler);
original.addEventListener(IOErrorEvent.IO_ERROR, fileMoveIOErrorEventHandler);
original.moveToAsync(destination);

function fileMoveCompleteHandler(event:Event):void {
    trace(event.target); // [object File]
}
function fileMoveIOErrorEventHandler(event:IOErrorEvent):void {
    trace("I/O Error.");
}
```

File 類別也包含 File.moveToTrash() 和 File.moveToTrashAsync() 方法，可以將檔案或目錄移至系統的垃圾桶。

刪除檔案

Adobe AIR 1.0 以及更新的版本

File 類別包含 deleteFile() 和 deleteFileAsync() 方法。這兩個方法可用來刪除檔案：前者會以同步方式運作，而後者則是以非同步方式運作 (請參閱：第 574 頁 「AIR 檔的基本概念」)。

例如，下列程式碼會以同步方式刪除使用者文件目錄中的 test.txt 檔案：

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.deleteFile();
```

下列程式碼會以非同步方式刪除使用者文件目錄中的 test.txt 檔案：

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.addEventListener(Event.COMPLETE, completeHandler)
file.deleteFileAsync();

function completeHandler(event:Event):void {
    trace("Deleted.")
}
```

File 類別還包含 moveToTrash() 和 moveToTrashAsync() 方法，可以讓您用來將檔案或目錄移至 「系統」的垃圾桶。如需詳細資訊，請參閱第 592 頁 「將檔案移至垃圾桶」。

將檔案移至垃圾桶

Adobe AIR 1.0 以及更新的版本

File 類別包含 moveToTrash() 和 moveToTrashAsync() 方法。這兩個方法可用來將檔案或目錄傳送至 「系統」的垃圾桶：前者會以同步方式運作，而後者則是以非同步方式運作 (請參閱：第 574 頁 「AIR 檔的基本概念」)。

例如，下列程式碼會以同步方式，將使用者文件目錄中的 test.txt 檔案移至 「系統」的垃圾桶：

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.moveToTrash();
```

備註：在不具有可復原資源回收筒功能的作業系統上，將會立即移除檔案。

建立暫存檔案

Adobe AIR 1.0 以及更新的版本

File 類別包含 createTempFile() 方法，可以用來在 「系統」的暫存目錄資料夾中建立新檔案，如下列範例所示：

```
var temp:File = File.createTempFile();
```

createTempFile() 會自動建立唯一的暫存檔案 (可以節省您為了尋找新的唯一位置而要花費的功夫)。

您可以使用暫存檔案，暫時儲存在應用程式某個工作階段使用的資訊。請注意，您也可以使用 createTempDirectory() 方法，在「系統」的暫存目錄中建立新的唯一暫存目錄。

在關閉應用程式之前，您可能會想要刪除這個暫存檔案，因為系統 「不會」自動刪除所有裝置上的該檔案。

使用儲存空間

Adobe AIR 2 以及更新的版本

在 AIR 2 中，當大量儲存體磁碟區掛接或卸下時，您會察覺到。StorageVolumeInfo 類別會定義一個 Singleton storageVolumeInfo 物件。掛接儲存體磁碟區後，StorageVolumeInfo.storageVolumeInfo 物件會傳送一個 storageVolumeMount 事件。然後在卸下之後，就會傳送一個 storageVolumeUnmount 事件。StorageVolumeChangeEvent 類別會定義這些事件。

備註：在較新的 Linux 版本上，StorageVolumeInfo 物件只會針對實體裝置及在特定位置掛接的網路磁碟機傳送 storageVolumeMount 及 storageVolumeUnmount 事件。

StorageVolumeChangeEvent 類別的 storageVolume 屬性是一個 StorageVolume 物件。StorageVolume 類別會定義儲存體磁碟區的基本屬性：

- drive— Windows 上的磁碟機代號 (在其他作業系統上是 null)
- fileSystemType— 儲存體磁碟區上的檔案系統類型 (例如 "FAT"、"NTFS"、"HFS" 或 "UFS")
- isRemoveable— 磁碟區是否可卸除 (true) 或 (false)

- isWritable— 磁碟區是否可寫入 (true) 或 (false)
- name— 磁碟區的名稱
- rootDirectory— 與磁碟區根目錄相對應的檔案物件

StorageVolumeChangeEvent 類別也包括 rootDirectory 屬性。rootDirectory 屬性是參照已掛接或卸下之儲存體磁碟區根目錄的 File 物件。

針對卸下的磁碟區，並未定義 StorageVolumeChangeEvent 物件的 storageVolume 屬性 (null)。不過您可以存取事件的 rootDirectory 屬性。

以下程式碼會在儲存體磁碟區掛接後，輸出它的名稱以及檔案路徑：

```
StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME_MOUNT,
onVolumeMount);
function onVolumeMount(event:StorageVolumeChangeEvent):void
{
    trace(event.storageVolume.name, event.rootDirectory.nativePath);
}
```

以下程式碼會在儲存體磁碟區卸下後，輸出它的檔案路徑：

```
StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME_UNMOUNT,
onVolumeUnmount);
function onVolumeUnmount(event:StorageVolumeChangeEvent):void
{
    trace(event.rootDirectory.nativePath);
}
```

StorageVolumeInfo.storageVolumeInfo 物件包含 getStorageVolumes() 方法。這個方法會根據目前掛接的儲存體磁碟機，傳回 StorageVolume 物件的向量。以下程式碼顯示如何列示所有掛接之儲存體磁碟區的名稱和根目錄：

```
var volumes:Vector.<StorageVolume> = new Vector.<StorageVolume>;
volumes = StorageVolumeInfo.storageVolumeInfo.getStorageVolumes();
for (var i:int = 0; i < volumes.length; i++)
{
    trace(volumes[i].name, volumes[i].rootDirectory.nativePath);
}
```

備註：在較新的 Linux 版本中，getStorageVolumes() 方法會傳回與特定位置掛接之實體裝置和網路磁碟機對應的物件。

File.getRootDirectories() 方法會列示根目錄 (請參閱第 580 頁 「指向檔案系統根目錄」)。不過，StorageVolume 物件 (由 StorageVolumeInfo.getStorageVolumes() 方法列舉) 會提供儲存體磁碟區的詳細資訊。

您可以使用 StorageVolume 物件的 rootDirectory 屬性的 spaceAvailable 屬性來取得儲存體磁碟區的可用空間。(請參閱第 586 頁 「判斷磁碟的可用空間」)。

更多說明主題

StorageVolume

StorageVolumeInfo

讀取和寫入檔案

Adobe AIR 1.0 以及更新的版本

FileStream 類別可讓 AIR 應用程式讀取和寫入檔案系統。

讀取和寫入檔案的工作流程

Adobe AIR 1.0 以及更新的版本

讀取和寫入檔案的工作流程如下：

將指向某個路徑的 **File** 物件初始化。
File 物件代表您要使用檔案 (或者即將建立檔案) 的路徑。

```
var file:File = File.documentsDirectory;
file = file.resolvePath("AIR Test/testFile.txt");
```

上述範例是使用 File 物件的 File.documentsDirectory 屬性和 resolvePath() 方法來初始化 File 物件。不過，您還可以使用其它多種方式，將 File 物件指向某個檔案。如需詳細資訊，請參閱第 581 頁 「將 File 物件指向檔案」。

初始化 **FileStream** 物件。

呼叫 **FileStream** 物件的 **open()** 或 **openAsync()** 方法。
要呼叫的方法視您要開啟檔案來執行同步還是非同步作業而定。請使用 File 物件做為開啟方法的 file 參數。針對 fileMode 參數，請從 FileMode 類別指定常數，這個類別會指定檔案的使用方式。

例如，下列程式碼會初始化用來建立檔案並覆寫所有現有資料的 FileStream 物件：

```
var fileStream:FileStream = new FileStream();
fileStream.open(file, FileMode.WRITE);
```

如需詳細資訊，請參閱第 595 頁 「初始化 FileStream 物件以及開啟和關閉檔案」、第 595 頁 「FileStream 開啟模式」。

如果您以非同步方式開啟檔案 (使用 **openAsync()** 方法)，請為 **FileStream** 物件加入事件偵聽程式並加以設定。
這些事件偵聽程式方法會回應 FileStream 物件在各種情況下傳送的事件。這些情況包括從檔案讀取資料、發生 I/O 錯誤時，或者要寫入的資料已經全部寫入時。

如需詳細資訊，請參閱第 599 頁 「非同步程式設計與以非同步方式開啟之 FileStream 物件所產生的事件」。

視需要納入用於讀取和寫入資料的程式碼。
許多 FileStream 類別的方法都與讀取和寫入資料有關 (這些方法都是以 「read」或 「write」做為開頭)。您選擇用來讀取或寫入資料的方法需視目標檔案的資料格式而定。

舉例來說，如果目標檔案中的資料屬於 UTF 編碼的文字，您就可以使用 readUTFBytes() 和 writeUTFBytes() 方法。如果您要將資料當做位元組陣列來處理，則需使用 readByte()、readBytes()、writeByte() 和 writeBytes() 方法。如需詳細資訊，請參閱第 599 頁 「資料格式，以及選擇要使用的讀取和寫入方法」。

如果您是以非同步方式開啟檔案，請確保在呼叫讀取方法前，有足夠的資料可供使用。如需詳細資訊，請參閱第 597 頁 「讀取緩衝區和 FileStream 物件的 bytesAvailable 屬性」。

要對檔案寫入資料之前，如果您希望查看可用的磁碟空間容量，可以查看 File 物件的 spaceAvailable 屬性。如需詳細資訊，請參閱第 586 頁 「判斷磁碟的可用空間」。

使用完檔案時，請呼叫 **FileStream** 物件的 **close()** 方法。
呼叫 close() 方法可以讓其他應用程式使用該檔案。

如需詳細資訊，請參閱第 595 頁 「初始化 FileStream 物件以及開啟和關閉檔案」。

如需使用 FileStream 類別讀取和寫入檔案的樣本應用程式，請參閱 Adobe AIR Developer Center 中的下列文章：

- 建置文字檔案編輯器

- 建置文字檔案編輯器
- 從 XML 偏好設定檔案讀取和寫入資料

使用 FileStream 物件

Adobe AIR 1.0 以及更新的版本

FileStream 類別會定義用於開啟、讀取和寫入檔案的方法。

FileStream 開啟模式

Adobe AIR 1.0 以及更新的版本

FileStream 物件的 open() 和 openAsync() 方法各自包含一個 fileMode 參數，這個參數會定義檔案串流的某些屬性，其中包括下列項目：

- 能夠從檔案進行讀取
- 能夠在檔案中進行寫入
- (在寫入時) 是否一律從檔案的結尾開始附加資料
- 當檔案不存在時 (而且其上層目錄也不存在時) 的處理方式

以下列出各種檔案模式 (您可以指定這些模式做為 open() 和 openAsync() 方法的 fileMode 參數值)：

檔案模式	說明
FileMode.READ	指定開啟的檔案只供讀取。
FileMode.WRITE	指定開啟的檔案可供寫入。如果檔案不存在，則會在開啟 FileStream 物件時建立檔案。如果檔案已經存在，則其中全部現有資料都會遭刪除。
FileMode.APPEND	指定開啟的檔案可供附加資料。如果檔案不存在，則會建立檔案。如果檔案已經存在，現有的資料不會遭覆寫，而且所有寫入作業都會從檔案的結尾開始。
FileMode.UPDATE	指定開啟的檔案可供讀取和寫入。如果檔案不存在，則會建立檔案。指定這個模式，即可對檔案執行隨機讀取 / 寫入作業。您可以在檔案中的任何位置進行讀取。在檔案中寫入資料時，只有已寫入的位元組會覆寫現有位元組 (其它所有位元組不會改變)。

初始化 FileStream 物件以及開啟和關閉檔案

Adobe AIR 1.0 以及更新的版本

當您開啟 FileStream 物件時，這個物件就可以對檔案執行讀取和寫入作業。您可以將 File 物件傳遞給 FileStream 物件的 open() 或 openAsync() 方法，以開啟這個 FileStream 物件。

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.READ);
```

fileMode 參數 (open() 和 openAsync() 方法的第二個參數) 會指定檔案的開啟模式：可供讀取、寫入、附加或更新。如需詳細資訊，請參閱上一節：第 595 頁 「FileStream 開啟模式」。

如果您是使用 openAsync() 方法開啟檔案以執行非同步檔案作業，請設定事件偵聽程式以處理非同步事件：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completeHandler);
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.addEventListener(IOErrorEvent.IO_Error, errorHandler);
myFileStream.openAsync(myFile, FileMode.READ);

function completeHandler(event:Event):void {
    // ...
}

function progressHandler(event:ProgressEvent):void {
    // ...
}

function errorHandler(event:IOErrorEvent):void {
    // ...
}
```

視您使用 open() 或 openAsync() 方法而定，檔案開啟時可供同步或非同步作業。如需詳細資訊，請參閱：第 574 頁 「AIR 檔的基本概念」。

如果您在 FileStream 物件的開啟方法中，將 fileMode 參數設定為 FileMode.READ 或 FileMode.UPDATE，那麼當您一開啟 FileStream 物件，就會將資料讀取至讀取緩衝區中。如需詳細資訊，請參閱第 597 頁 「讀取緩衝區和 FileStream 物件的 bytesAvailable 屬性」。

您可以呼叫 FileStream 物件的 close() 方法關閉相關聯的檔案，讓其它應用程式可以使用這個檔案。

FileStream 物件的 position 屬性

Adobe AIR 1.0 以及更新的版本

FileStream 物件的 position 屬性可以決定資料在下一個讀取或寫入方法的讀取和寫入位置。

執行讀取或寫入作業之前，請先將 position 屬性設定為檔案中的任何有效位置。

例如，下列程式碼會將 "hello" 字串 (UTF 編碼格式) 寫入檔案中 8 這個位置：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.UPDATE);
myFileStream.position = 8;
myFileStream.writeUTFBytes("hello");
```

當您先開啟 FileStream 物件時，position 屬性會設定為 0。

執行讀取作業之前，position 的值必須大於等於 0，並且小於檔案中的位元組數 (亦即檔案中的現有位置)。

只有在下列情況下，才會修改 position 屬性的值：

- 明確設定 position 屬性時。
- 呼叫讀取方法時。
- 呼叫寫入方法時。

當您呼叫 FileStream 物件的讀取或寫入方法時，position 屬性會立即遞增您讀取或寫入的位元組數。視使用的讀取方法而定，position 會遞增您指定要讀取的位元組數，或遞增可用的位元組數。當您之後再呼叫一次讀取或寫入方法時，就會在新的位置讀取或寫入資料。

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.UPDATE);
myFileStream.position = 4000;
trace(myFileStream.position); // 4000
myFileStream.writeBytes(myByteArray, 0, 200);
trace(myFileStream.position); // 4200
```

不過，當 FileStream 以附加模式開啟時則為例外，因為 position 屬性並不會在呼叫寫入方法之後變更 (在附加模式下，一定會將資料寫入至檔案的結尾，因此與 position 屬性的值無關)。

對於開啟供執行非同步作業的檔案，直到下一行程式碼執行之後，寫入作業才完成。不過您可以循序呼叫多個非同步方法，執行階段會依照順序執行這些方法：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.openAsync(myFile, FileMode.WRITE);
myFileStream.writeUTFBytes("hello");
myFileStream.writeUTFBytes("world");
myFileStream.addEventListener(Event.CLOSE, closeHandler);
myFileStream.close();
trace("started.");

closeHandler(event:Event):void
{
    trace("finished.");
}
```

以下為上述程式碼的 trace 輸出：

```
started.
finished.
```

您「可以」在呼叫讀取或寫入方法之後立即 (或隨時) 指定 position 值，然後下一個讀取或寫入作業便會在該位置開始執行。例如，請注意下列程式碼會在呼叫 writeBytes() 作業之後立即設定 position 屬性，而且 position 會設定為該值 (300)，即使在寫入作業完成之後也一樣：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.openAsync(myFile, FileMode.UPDATE);
myFileStream.position = 4000;
trace(myFileStream.position); // 4000
myFileStream.writeBytes(myByteArray, 0, 200);
myFileStream.position = 300;
trace(myFileStream.position); // 300
```

讀取緩衝區和 **FileStream** 物件的 **bytesAvailable** 屬性

Adobe AIR 1.0 以及更新的版本

開啟具有讀取功能 (open() 或 openAsync() 方法的 fileMode 參數設定為 READ 或 UPDATE) 的 FileStream 物件時，執行階段會將資料儲存在內部緩衝區中。FileStream 物件會在您開啟檔案 (藉由呼叫 FileStream 物件的 open() 或 openAsync() 方法) 時，立即開始將資料讀取至緩衝區。

對於開啟供執行同步作業的檔案 (使用 open() 方法)，您一律可以將 position 指標設定為任何有效的位置 (必須在檔案的範圍內) 並開始讀取任意資料量 (必須在檔案的範圍內)，如下列程式碼 (其中假設檔案至少包含 100 個位元組) 所示：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.READ);
myFileStream.position = 10;
myFileStream.readBytes(myByteArray, 0, 20);
myFileStream.position = 89;
myFileStream.readBytes(myByteArray, 0, 10);
```

無論檔案是開啟供執行同步還是非同步作業，讀取方法一定都會從 「可用」的位元組 (由 bytesAvalable 屬性表示) 來讀取資料。以同步方式讀取時，檔案的所有位元組隨時都可供讀取。以非同步方式讀取時，則只能從 position 屬性指定的位置之後的位元組開始讀取，並且會在每次非同步緩衝區填滿資料時傳送 progress 事件。

對於開啟供執行 「同步」作業的檔案，bytesAvailable 屬性一律設定為表示從 position 屬性到檔案結尾的位元組數 (檔案中的所有位元組都可供讀取)。

對於開啟供執行 「非同步」作業的檔案，您必須確保讀取緩衝區中已經有足夠的資料，才能呼叫讀取方法。對於以非同步方式開啟的檔案，隨著讀取作業的進行，會將檔案的資料 (亦即當讀取作業開始，位於 position 指定之位置的資料) 加入至緩衝區，而且每一次讀取位元組時，bytesAvailable 屬性都會遞增。bytesAvailable 屬性表示從 position 屬性指定的位置開始，到緩衝區結尾的可用位元組數。FileStream 物件會定期傳送 progress 事件。

對於以非同步方式開啟的檔案，當讀取緩衝區中的資料已經可供使用時，FileStream 物件便會定期傳送 progress 事件。例如，下列程式碼會在將資料讀取至緩衝區之後，將這些資料讀取至 ByteArray 物件 bytes 中：

```
var bytes:ByteArray = new ByteArray();
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.openAsync(myFile, FileMode.READ);

function progressHandler(event:ProgressEvent):void
{
    myFileStream.readBytes(bytes, myFileStream.position, myFileStream.bytesAvailable);
}
```

對於以非同步方式開啟的檔案，只有讀取緩衝區中的資料才可供讀取。此外，當您讀取資料時，就會從讀取緩衝區中移除該資料。對於讀取作業，您必須確保讀取緩衝區中有資料，才能呼叫讀取作業。例如，下列程式碼會從檔案中 4000 這個位置，讀取 8000 個位元組資料：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
myFileStream.position = 4000;

var str:String = "";

function progressHandler(event:Event):void
{
    if (myFileStream.bytesAvailable > 8000 )
    {
        str += myFileStream.readMultiByte(8000, "iso-8859-1");
    }
}
```

在寫入作業期間，FileStream 物件不會將資料讀取至讀取緩衝區中。當寫入作業完成時 (亦即，寫入緩衝區中的所有資料都已經寫入至檔案)，FileStream 物件會啟動新的讀取緩衝區 (假設相關聯的 FileStream 物件在開啟時具有讀取功能)，並且從 position 屬性指定的位置，開始將資料讀取至該讀取緩衝區中。position 屬性會是最後一個位元組的寫入位置，但是如果使用者在寫入作業完成後，為 position 物件指定不同的值，則會是不同的位置。

非同步程式設計與以非同步方式開啟之 **FileStream** 物件所產生的事件

Adobe AIR 1.0 以及更新的版本

以非同步方式開啟檔案 (使用 openAsync() 方法) 時，讀取和寫入檔案的作業都會以非同步方式執行。因此，在將資料讀取至讀取緩衝區中，以及正在寫入輸出資料時，都可以執行其它 ActionScript 程式碼。

這表示您必須為以非同步方式開啟之 FileStream 物件所產生的事件登錄。

為 progress 事件註冊之後，每當有新的資料可供讀取時，您就會收到通知，如下列程式碼所示：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";

function progressHandler(event:ProgressEvent):void
{
    str += myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

您可以為 complete 事件登錄以讀取全部資料，如下列程式碼所示：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";
function completeHandler(event:Event):void
{
    str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

就像輸入資料會先儲存在緩衝區之後，才會以非同步方式讀取這些資料一樣，您在非同步串流上寫入的資料也會先儲存在緩衝區，然後再以非同步方式寫入至檔案。當資料已經寫入至檔案時，FileStream 物件會定期傳送 OutputProgressEvent 物件。OutputProgressEvent 物件包含 bytesPending 屬性，這個屬性設定為要寫入的剩餘位元組數。有時為了顯示進度對話方塊，您可以為 outputProgress 事件登錄，以便於緩衝區中的這些資料實際寫入至檔案時收到通知。不過，通常並不需要這麼做。特別是您可以呼叫 close() 方法，而不需考量未寫入的位元組。FileStream 物件將繼續寫入資料，而且在最後一個位元組寫入至檔案以及基礎檔案關閉之後，便會傳送 close 事件。

資料格式，以及選擇要使用的讀取和寫入方法

Adobe AIR 1.0 以及更新的版本

磁碟中的每一個檔案都是一組位元組。在 ActionScript 中，檔案的資料一律都可以表示為 ByteArray。例如，下列程式碼會將檔案的資料讀取至名為 bytes 的 ByteArray 物件中：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completeHandler);
myFileStream.openAsync(myFile, FileMode.READ);
var bytes:ByteArray = new ByteArray();

function completeHandler(event:Event):void
{
    myFileStream.readBytes(bytes, 0, myFileStream.bytesAvailable);
}
```

同樣地，下列程式碼會將資料從名為 bytes 的 ByteArray 寫入至檔案：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.WRITE);
myFileStream.writeBytes(bytes, 0, bytes.length);
```

不過，通常您不會希望將資料儲存在 ActionScript ByteArray 物件中。此外，資料檔案通常具有指定的檔案格式。

例如，檔案中的資料可能具有文字檔案格式，而且您會希望在 String 物件中呈現這類資料。

因此，FileStream 類別包含了讀取和寫入方法，可用於針對 ByteArray 物件以外的類型，執行資料的讀取和寫入作業。例如，readMultiByte() 方法可以讓您讀取檔案的資料，並將它儲存至字串，如下列程式碼所示：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";

function completeHandler(event:Event):void
{
    str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

readMultiByte() 方法的第二個參數會指定 ActionScript 用於解譯資料的文字格式 (在範例中為「iso-8859-1」)。Adobe AIR 支援常見的字元集編碼 (請參閱支援的字元集)。

FileStream 類別也包含 readUTFBytes() 方法，可以將讀取緩衝區的資料讀取至使用 UTF-8 字元集的字串中。由於 UTF-8 字元集中的字元都沒有固定長度，所以請勿在會回應 progress 事件的方法中使用 readUTFBytes()，因為位於讀取緩衝區結尾的資料可能會以不完整的字元來呈現 (使用 readMultiByte() 方法並搭配長度不固定的字元編碼時，也會發生這種情形)。因此，請在 FileStream 物件傳送 complete 事件時，讀取整個資料集。

此外，writeMultiByte() 和 writeUTFBytes() 這兩個相似的寫入方法也可以搭配 String 物件和文字檔案使用。

readUTF() 和 writeUTF() 方法 (請勿與 readUTFBytes() 和 writeUTFBytes() 混淆) 也會對檔案執行文字資料的讀取和寫入作業，但是這兩個方法會假設該文字資料是在指定該文字資料長度的資料之前，這種作法在標準的文字檔案中並不常見。

某些以 UTF 編碼的文字檔案會以「UTF-BOM」(位元組順序標記) 字元做為開頭，這個字元會定義位元組順序和編碼格式 (例如 UTF-16 或 UTF-32)。

如需文字檔案的讀取和寫入範例，請參閱第 601 頁 「範例：將 XML 檔案讀取至 XML 物件中」。

readObject() 和 writeObject() 可以讓您輕易地對 ActionScript 的物件儲存和擷取資料。資料的編碼格式為 AMF (ActionScript Message Format)，Adobe AIR、Flash Player、Flash Media Server 和 Flex Data Services 包含可搭配此格式資料運作的 API。

您可以使用其它讀取和寫入方法 (例如 readDouble() 和 writeDouble())。不過，如果使用這些方法，請確定檔案格式符合這些方法定義之資料的格式。

檔案格式通常比簡單的文字格式更為複雜。例如，MP3 檔案包含經過壓縮的資料，而這些資料只能以 MP3 檔案專用的解壓縮程序和解碼演算法來解譯。MP3 檔案也可包含 ID3 標籤，而這些標籤包含檔案的相關中繼標籤資訊 (例如，歌曲的標題和藝人)。ID3 格式有多種版本，而最簡單的版本 (ID3 第 1 版) 將在第 602 頁 「範例：使用隨機存取權讀取和寫入資料」一節中加以說明。

其它檔案格式 (例如影像、資料庫、應用程式文件等檔案的格式) 都有不同的結構，因此若要在 ActionScript 中使用這些格式的資料，您必須了解資料結構的建立方式。

使用 load() 和 save() 方法

Flash Player 10 以及更新的版本，**Adobe AIR 1.5** 以及更新的版本

Flash Player 10 為 FileReference 類別加入了 load() 和 save() 方法。AIR 1.5 也有這兩個方法，而 File 類別繼承了 FileReference 類別的方法。這些方法的設計是為了讓使用者在 Flash Player 中安全地進行檔案資料的載入和儲存。不過，AIR 應用程式也可以將這些方法做為一個能讓檔案進行非同步載入和儲存的簡易途徑。

例如，以下的程式碼會將一個字串儲存到文字檔中：

```
var file:File = File.applicationStorageDirectory.resolvePath("test.txt");
var str:String = "Hello.";
file.addEventListener(Event.COMPLETE, fileSaved);
file.save(str);
function fileSaved(event:Event):void
{
    trace("Done.");
}
```

save() 方法的 data 參數可以接受 String、XML 或 ByteArray 類型的值。若引數為 String 或 XML 類型的值，這個方法就會將檔案儲存為 UTF-8 編碼的文字檔。

當此程式碼樣本執行時，應用程式會顯示一個對話方塊來讓使用者選取檔案的儲存目標。

以下的程式碼會從 UTF-8 編碼的文字檔中載入一個字串：

```
var file:File = File.applicationStorageDirectory.resolvePath("test.txt");
file.addEventListener(Event.COMPLETE, loaded);
file.load();
var str:String;
function loaded(event:Event):void
{
    var bytes:ByteArray = file.data;
    str = bytes.readUTFBytes(bytes.length);
    trace(str);
}
```

FileStream 類別所提供的功能比 load() 和 save() 方法所提供的功能還多：

- 使用 FileStream 類別時，您對資料的讀取和寫入作業便能透過同步以及非同步的方式進行。
- FileStream 類別可讓您以遞增方式將資料寫入檔案。
- FileStream 類別可讓您以隨機存取的目的開啟檔案 (可對檔案的任何區段進行讀取和寫入)。
- FileStream 類別可讓您指定檔案的存取類型，只要設定 open() 或 openAsync() 方法的 fileMode 參數即可。
- FileStream 類別不需對使用者顯示 「開啟」或 「儲存」對話方塊就可讓您將資料存入檔案。
- 使用 FileStream 類別讀取資料時，您可以直接使用位元組陣列之外的類型。

範例：將 XML 檔案讀取至 XML 物件中

Adobe AIR 1.0 以及更新的版本

在下列範例中，會示範如何對包含 XML 資料的文字檔案執行讀取和寫入作業。

若要讀取檔案中的資料，請先初始化 File 和 FileStream 物件，然後呼叫 FileStream 的 readUTFBytes() 方法，再將取得的字串轉換為 XML 物件：

```
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
var fileStream:FileStream = new FileStream();
fileStream.open(file, FileMode.READ);
var prefsXML:XML = XML(fileStream.readUTFBytes(fileStream.bytesAvailable));
fileStream.close();
```

同樣地，將資料寫入至檔案也很簡單，只要設定適當的 File 和 FileStream 物件，然後呼叫 FileStream 物件的寫入方法即可。請將 XML 資料的字串版本傳遞給寫入方法，如下列程式碼所示：

```
var prefsXML:XML = <prefs><autoSave>true</autoSave></prefs>;
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
fileStream = new FileStream();
fileStream.open(file, FileMode.WRITE);

var outputString:String = '<?xml version="1.0" encoding="utf-8"?>\n';
outputString += prefsXML.toXMLString();

fileStream.writeUTFBytes(outputString);
fileStream.close();
```

上述範例使用 readUTFBytes() 和 writeUTFBytes() 方法，因為這兩個方法會假設檔案格式為 UTF-8。如果檔案格式並非 UTF-8，您就必須使用不同的方法 (請參閱第 599 頁「資料格式，以及選擇要使用的讀取和寫入方法」)。

上述範例使用開啟供執行同步作業的 FileStream 物件。您也可以開啟檔案供執行非同步作業 (如此便必須依賴事件偵聽程式函數來回應事件)。例如，在下列程式碼中，會說明如何以非同步方式讀取 XML 檔案：

```
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
var fileStream:FileStream = new FileStream();
fileStream.addEventListener(Event.COMPLETE, processXMLData);
fileStream.openAsync(file, FileMode.READ);
var prefsXML:XML;

function processXMLData(event:Event):void
{
    prefsXML = XML(fileStream.readUTFBytes(fileStream.bytesAvailable));
    fileStream.close();
}
```

當已將整個檔案都讀取至讀取緩衝區中時 (當 FileStream 物件傳送 complete 事件時)，就會叫用 processXMLData() 方法。這個方法會呼叫 readUTFBytes() 方法以取得讀取資料的字串版本，並且根據該字串建立名為 prefsXML 的 XML 物件。

如需示範這些功能的樣本應用程式，請參閱 從 XML 偏好設定檔案讀取和寫入資料。

範例：使用隨機存取權讀取和寫入資料

Adobe AIR 1.0 以及更新的版本

MP3 檔案可包含 ID3 標籤，而這些標籤位於檔案 (包含可識別歌曲的中繼資料) 開頭或結尾的區段。ID3 標籤格式本身具有不同的版本。在下列範例中，會說明如何使用「檔案資料的隨機存取權」(表示範例可以在檔案中的任意位置執行存取和寫入作業)，對包含最簡單之 ID3 格式 (ID3 1.0 版) 的 MP3 檔案執行讀取和寫入作業。

包含 ID3 標籤第 1 版的 MP3 檔案在檔案結尾處 (最後的 128 位元組中) 包含 ID3 資料。

當存取檔案供隨機讀取 / 寫入存取時，請務必將 open() 或 openAsync() 方法的 fileMode 參數設定為 FileMode.UPDATE：

```
var file:File = File.documentsDirectory.resolvePath("My Music/Sample ID3 v1.mp3");
var fileStr:FileStream = new FileStream();
fileStr.open(file, FileMode.UPDATE);
```

如此，您才可以對檔案執行讀取和寫入作業。

在開啟檔案時，您可以將 position 指標設定為檔案結尾之前 128 個位元組這個位置：

```
fileStr.position = file.size - 128;
```

上述程式碼會將 position 屬性設定為檔案中的這個位置，因為 ID3 v1.0 格式指定 ID3 標籤資料必須儲存在檔案的最後 128 個位元組中。該版本的規格也規定下列項目：

- 標籤的前 3 個位元組必須包含 "TAG" 字串。
- 之後的 30 個字元必須包含 MP3 歌曲的標題 (以字串表示)。
- 之後的 30 個字元必須包含藝人的姓名 (以字串表示)。
- 之後的 30 個字元必須包含專輯的名稱 (以字串表示)。
- 之後的 4 個字元必須包含年份 (以字串表示)。
- 之後的 30 個字元必須包含註解 (以字串表示)。
- 之後 1 個位元組必須包含表示歌曲類型的代碼。
- 所有文字資料的格式都必須是 ISO 8859-1。

id3TagRead() 方法會在資料已讀入之後 (complete 事件發生時) 檢查資料：

```
function id3TagRead():void
{
    if (fileStr.readMultiByte(3, "iso-8859-1").match(/tag/i))
    {
        var id3Title:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Artist:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Album:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Year:String = fileStr.readMultiByte(4, "iso-8859-1");
        var id3Comment:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3GenreCode:String =  fileStr.readByte().toString(10);
    }
}
```

您也可以對檔案執行隨機存取寫入作業。例如，您可以剖析 id3Title 變數，確保它已經透過正確方式改為大寫 (使用 String 類別的方法)，接著再將經過修改的字串 (名為 newTitle) 寫入至檔案，如下所示：

```
fileStr.position = file.length - 125;    // 128 - 3
fileStr.writeMultiByte(newTitle, "iso-8859-1");
```

為了符合 ID3 第 1 版的標準，newTitle 字串的長度必須為 30 個字元，並且在結尾處以字元碼 0 來填補 (String.fromCharCode(0))。

第 39 章 儲存本機資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

SharedObject 類別可用來將少量資料儲存至用戶端電腦。在 Adobe AIR 中，EncryptedLocalStore 類別也可用來在 AIR 應用程式中，將少量的重要隱私權使用者資料儲存至本機電腦。

您亦可在檔案系統中讀取與寫入檔案，並 (在 Adobe AIR 中) 存取本地資料庫檔案。如需詳細資訊，請參閱第 561 頁 「使用檔案系統」和第 615 頁 「在 AIR 中使用本機 SQL 資料庫」。

共享物件時，有幾個安全性相關因素。如需詳細資訊，請參閱 第 900 頁 「安全性」中的第 926 頁 「共享物件」。

共享物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

共享物件有時候又稱為 「Flash cookie」，是一種由您所瀏覽網站在您的電腦上建立的資料檔案。共享物件最常用來增強您的網路瀏覽體驗，例如，讓您自訂經常瀏覽的網站外觀。

關於共享物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

共享物件的作用與瀏覽器 cookie 類似。SharedObject 類別可用來將資料儲存在使用者的本機硬碟上，並在相同的工作階段或後面的工作階段期間呼叫該資料。應用程式只有在相同的網域上執行時，才能存取它們自己的 SharedObject 資料。資料不會傳送至伺服器，且無法由在其他網域中執行的其他應用程式存取，但可供相同網域的應用程式存取。

共享物件與 cookie 的比較

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

cookie 和共享物件十分類似。因為大部分網頁程式設計人員都熟悉 cookie 的運作，所以將本機 SharedObject 與 cookie 做比較可能比較有用。

遵守 RFC 2109 標準的 cookie 一般具有下列屬性：

- 它們會過期，而且預設通常是在工作階段結束時。
- 用戶端可以根據特定網站來停用它們。
- 總限制為 300 個 cookie，而每個網站最多 20 個 cookie。
- 個別的大小通常限制為 4 KB。
- 它們有時被認為是安全性威脅，因此有時會在用戶端上予以停用。
- 它們儲存在用戶端瀏覽器所指定的位置。
- 它們是透過 HTTP 從用戶端傳輸至伺服器。

相較之下，共享物件具有下列屬性：

- 它們預設不會過期。

- 每個的大小預設都限制為 100 KB。
- 它們可以儲存簡單資料類型 (例如 String、Array 和 Date)。
- 它們儲存在應用程式所指定的位置 (在使用者的主目錄中)。
- 它們不會在用戶端與伺服器之間進行傳輸。

關於 SharedObject 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用 SharedObject 類別，您可以建立和刪除共享物件，以及偵測所使用之 SharedObject 物件的目前大小。

建立共享物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要建立 SharedObject 物件，請使用 SharedObject.getLocal() 方法，其語法如下：

```
SharedObject.getLocal("objectName" [, pathname]): SharedObject
```

下列範例會建立稱為 mySO 的共享物件：

```
public var mySO:SharedObject;
mySO = SharedObject.getLocal("preferences");
```

這樣會在用戶端電腦上建立稱為 preferences.sol 的檔案。

「本機」這個詞彙指的是共享物件的位置。在此例中，Adobe® Flash® Player 會在用戶端本機的主目錄中儲存 SharedObject 檔案。

當您建立共享物件時，Flash Player 會在其執行程序內建立應用程式及網域的新目錄。也會建立 *.sol 檔案，以儲存 SharedObject 資料。此檔案的預設位置是使用者主目錄的子目錄。下表顯示此目錄的預設位置：

作業系統	位置
Windows 95/98/ME/2000/XP	c:/Documents and Settings/username/Application Data/Macromedia/Flash Player/#SharedObjects
Windows Vista/Windows 7	c:/Users/username/AppData/Roaming/Macromedia/Flash Player/#SharedObjects
Macintosh OS X	/Users/username/Library/Preferences/Macromedia/Flash Player/#SharedObjects/web_domain/path_to_application/application_name/object_name.sol
Linux/Unix	/home/username/.macromedia/Flash_Player/#SharedObjects/web_domain/path_to_application/application_name/object_name.sol

#SharedObjects 目錄下是隨機命名的目錄。該目錄下是符合主機名稱的目錄，然後是應用程式的路徑，最後是 *.sol 檔案。

例如，如果您在本機主機上要求應用程式 MyApp.swf，而且位在子目錄 /sos 內，則 Flash Player 會將 *.sol 檔案儲存至下列位置 (在 Windows XP 上)：

```
c:/Documents and Settings/fred/Application Data/Macromedia/Flash
Player/#SharedObjects/KROKWXRK/#localhost/sos/MyApp.swf/data.sol
```

備註：如果您未在 SharedObject.getLocal() 方法中提供名稱，則 Flash Player 會將檔案命名為 undefined.sol。

Flash 預設最多可為每個網域在本機儲存 100 KB 的持續性 SharedObject 物件。使用者可以設定此值。應用程式嘗試將資料儲存至會讓它大於 100 KB 的共享物件時，Flash Player 會顯示 「本地儲存區」對話框，讓使用者可以針對要求存取的網域允許或拒絕更多本地儲存區。

指定路徑

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用選用的 pathname 參數指定 SharedObject 檔案的位置。此檔案必須是該網域之 SharedObject 目錄的子目錄。例如，如果您在 localhost 上要求應用程式並指定下列項目：

```
mySO = SharedObject.getLocal("myObjectFile","/");
```

Flash Player 會將 SharedObject 檔案寫入 /#localhost 目錄 (如果應用程式離線，則為 /localhost)。這適用於您希望用戶端上的多個應用程式可以存取相同的共享物件時。在此情況下，用戶端可以執行兩個 Flex 應用程式，這兩個應用程式都會指定共享物件的路徑，即網域的根目錄；用戶端之後就可以從這兩個應用程式存取相同的共享物件。若要在多個應用程式之間共享資料而不加以保存，可以使用 LocalConnection 物件。

如果您指定的目錄不存在，Flash Player 就不會建立 SharedObject 檔案。

將資料新增至共享物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 SharedObject 物件的 data 屬性，將資料新增至 SharedObject 的 *.sol 檔案。若要將新的資料新增至共享物件，請使用下列語法：

```
sharedObject_name.data.variable = value;
```

下列範例會將 userName、itemNumbers 和 adminPrivileges 屬性及其值新增至 SharedObject：

```
public var currentUserName:String = "Reiner";
public var itemsArray:Array = new Array(101,346,483);
public var currentUserIsAdmin:Boolean = true;
mySO.data.userName = currentUserName;
mySO.data.itemNumbers = itemsArray;
mySO.data.adminPrivileges = currentUserIsAdmin;
```

在您將值指定給 data 屬性之後，必須指示 Flash Player 將那些值寫入 SharedObject 的檔案。若要強制 Flash Player 將值寫入 SharedObject 的檔案，請使用 SharedObject.flush() 方法，如下所示：

```
mySO.flush();
```

如果您未呼叫 SharedObject.flush() 方法，則 Flash Player 會在應用程式結束時將值寫入檔案。不過，如果資料超出預設設定，使用者就無法增加 Flash Player 儲存資料的可用空間。因此，最佳做法是呼叫 SharedObject.flush()。

在使用 flush() 方法將共享物件寫入使用者的硬碟時，您必須仔細檢查使用者是否已經使用 「Flash Player 設定管理員」 (www.macromedia.com/support/documentation/tw/flashplayer/help/settings_manager07.html) 明確停用本機儲存功能，如下列範例所示：

```
var so:SharedObject = SharedObject.getLocal("test");
trace("Current SharedObject size is " + so.size + " bytes.");
so.flush();
```

在共享物件中儲存物件

您可以在 SharedObject 的 data 屬性中儲存簡單物件，例如 Arrays 或 Strings。

下列範例是 ActionScript 類別，可定義方法來控制與共享物件的互動。這些方法可讓使用者在共享物件中新增和移除物件。此類別會儲存內含簡單物件的 ArrayCollection。

```
package {
    import mx.collections.ArrayCollection;
    import flash.net.SharedObject;

    public class LSOHandler {

        private var mySO:SharedObject;
        private var ac:ArrayCollection;
        private var lsoType:String;

        // The parameter is "feeds" or "sites".
        public function LSOHandler(s:String) {
            init(s);
        }

        private function init(s:String):void {
            ac = new ArrayCollection();
            lsoType = s;
            mySO = SharedObject.getLocal(lsoType);
            if (getObjects()) {
                ac = getObjects();
            }
        }

        public function getObjects():ArrayCollection {
            return mySO.data[lsoType];
        }

        public function addObject(o:Object):void {
            ac.addItem(o);
            updateSharedObjects();
        }

        private function updateSharedObjects():void {
            mySO.data[lsoType] = ac;
            mySO.flush();
        }
    }

}
```

下列 Flex 應用程式會建立所需的每種共享物件類型的 ActionScript 類別實體。然後會在使用者新增或移除部落格或網站 URL 時，針對該類別呼叫方法。

```
<?xml version="1.0"?>
<!-- lsos/BlogAggregator.mxml -->
<mx:Application
    xmlns:local="*"
    xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="initApp()"
    backgroundColor="#ffffff"
>
    <mx:Script>
        <![CDATA[
        import mx.collections.ArrayCollection;
        import mx.utils.ObjectUtil;
        import flash.net.SharedObject;

        [Bindable]
        public var welcomeMessage:String;

        [Bindable]
        public var localFeeds:ArrayCollection = new ArrayCollection();

        [Bindable]
        public var localSites:ArrayCollection = new ArrayCollection();

        public var lsofeeds:LSOHandler;
        public var lsosites:LSOHandler;

        private function initApp():void {
            lsofeeds = new LSOHandler("feeds");
            lsosites = new LSOHandler("sites");

            if (lsofeeds.getObjects()) {
                localFeeds = lsofeeds.getObjects();
            }
            if (lsosites.getObjects()) {
                localSites = lsosites.getObjects();
            }
        }

        // Adds a new feed to the feeds DataGrid.
        private function addFeed():void {
            // Construct an object you want to store in the
            // LSO. This object can contain any number of fields.
            var o:Object = {name:ti1.text, url:ti2.text, date:new Date()};
            lsofeeds.addObject(o);

            // Because the DataGrid's dataProvider property is
            // bound to the ArrayCollection, Flex updates the
            // DataGrid when you call this method.
            localFeeds = lsofeeds.getObjects();

            // Clear the text fields.
            ti1.text = '';
            ti2.text = '';
        }

        // Removes feeds from the feeds DataGrid.
        private function removeFeed():void {
            // Use a method of ArrayCollection to remove a feed.
            // Because the DataGrid's dataProvider property is
            // bound to the ArrayCollection, Flex updates the
            // DataGrid when you call this method. You do not need
            // to update it manually.
```

```
            if (myFeedsGrid.selectedIndex > -1) {

localFeeds.removeItemAt(myFeedsGrid.selectedIndex);
             }
        }

        private function addSite():void {
            var o:Object = {name:ti3.text, date:new Date()};
            lsosites.addObject(o);
            localSites = lsosites.getObjects();
            ti3.text = '';
        }

        private function removeSite():void {
            if (mySitesGrid.selectedIndex > -1) {

localSites.removeItemAt(mySitesGrid.selectedIndex);
            }
        }

        ]]>
    </mx:Script>

    <mx:Label text="Blog aggregator" fontSize="28"/>

    <mx:Panel title="Blogs">
        <mx:Form id="blogForm">
            <mx:HBox>
                <mx:FormItem label="Name:">
                    <mx:TextInput id="ti1" width="100"/>
                </mx:FormItem>
                <mx:FormItem label="Location:">
                    <mx:TextInput id="ti2" width="300"/>
                </mx:FormItem>
                <mx:Button id="b1" label="Add Feed" click="addFeed()"/>
            </mx:HBox>

            <mx:FormItem label="Existing Feeds:">
                <mx:DataGrid
                    id="myFeedsGrid"
                    dataProvider="{localFeeds}"
                    width="400"
                />
            </mx:FormItem>
            <mx:Button id="b2" label="Remove Feed" click="removeFeed()"/>
        </mx:Form>
    </mx:Panel>

    <mx:Panel title="Sites">
```

```
        <mx:Form id="siteForm">
            <mx:HBox>
                <mx:FormItem label="Site:">
                    <mx:TextInput id="ti3" width="400"/>
                </mx:FormItem>
                <mx:Button id="b3" label="Add Site" click="addSite()"/>
            </mx:HBox>

            <mx:FormItem label="Existing Sites:">
                <mx:DataGrid
                    id="mySitesGrid"
                    dataProvider="{localSites}"
                    width="400"
                />
            </mx:FormItem>
            <mx:Button id="b4" label="Remove Site" click="removeSite()"/>
        </mx:Form>
    </mx:Panel>

</mx:Application>
```

在共享物件中儲存具有類型的物件

您可以在共享物件中儲存具有類型的 ActionScript 實體。作法是呼叫 flash.net.registerClassAlias() 方法來註冊類別。如果您建立類別的實體，並將它儲存於共享物件的資料成員中，之後再讀出該物件，便會取得具有類型的實體。SharedObject objectEncoding 屬性預設支援 AMF3 編碼，並且會從 SharedObject 物件解壓縮儲存的實體；儲存的實體會保留您在呼叫 registerClassAlias() 方法時指定的相同類型。

(僅 iOS) 防止本機共享物件的雲端備份

Adobe AIR 3.7 以及更新的版本、僅 **iOS**

您可以設定 SharedObject.preventBackup 屬性，以控制本機共享物件是否將備份到 iOS 雲端備份服務上。這是為了符合 Apple 對可以重新產生或再次下載的內容的要求，但也是為了讓您的應用程式在離線使用時正常運作的必要動作。

建立多個共享物件

您可以為相同 Flex 應用程式建立多個共享物件。若要執行這項作業，請為它們指定不同的實體名稱 (如下例所示)：

```
public var mySO:SharedObject = SharedObject.getLocal("preferences");
public var mySO2:SharedObject = SharedObject.getLocal("history");
```

這樣會在 Flex 應用程式的本機目錄中建立 preferences.sol 檔案和 history.sol 檔案。

建立安全的 SharedObject

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在使用 getLocal() 或 getRemote() 建立本機或遠端 SharedObject 時，有一個選擇性的參數 secure 可以用來判斷是否僅限於透過 HTTPS 連線傳送的 SWF 檔才能存取此共享物件。如果此參數設定為 true，且 SWF 檔是透過 HTTPS 傳遞時，Flash Player 就會建立新的安全共享物件，或取得現有安全共享物件的參考；這個安全的共享物件就只能透過由 HTTPS 傳遞而呼叫 secure 參數設為 true 的 SharedObject.getLocal() 的 SWF 檔來讀取或寫入。如果此參數設定為 false 而 SWF 檔是透過 HTTPS 傳遞時，Flash Player 就會建立新的共享物件，或取得現有共享物件的參考。

這個共享物件可透過由非 HTTPS 連線傳遞之 SWF 檔來讀取或寫入。如果 SWF 檔是透過非 HTTPS 連線傳遞，而您嘗試將這個參數設為 true，則建立新的共享物件 (或存取之前建立的安全共享物件) 會失敗、擲回錯誤，並將共享物件設為 null。如果嘗試透過非 HTTPS 連線執行下列程式碼片段，SharedObject.getLocal() 方法將擲回錯誤：

```
try
{
    var so:SharedObject = SharedObject.getLocal("contactManager", null, true);
}
catch (error:Error)
{
    trace("Unable to create SharedObject.");
}
```

不管參數的值為何，建立的共享物件數會依網域的允許磁碟空間總數而定。

顯示共享物件的內容

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

共享物件會將值儲存在 data 屬性內。您可以使用 for..in 迴圈在共享物件之內重複執行每一個值，如下列範例所示：

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.hello = "world";
so.data.foo = "bar";
so.data.timezone = new Date().timezoneOffset;
for (var i:String in so.data)
{
    trace(i + ":\t" + so.data[i]);
}
```

刪除共享物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要刪除用戶端上的 SharedObject，請使用 SharedObject.clear() 方法。這樣不會刪除應用程式共享物件預設路徑中的目錄。

下列範例會刪除用戶端中的 SharedObject 檔案：

```
public function destroySharedObject():void {
    mySO.clear();
}
```

SharedObject 範例

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列範例顯示您可以將簡單物件 (例如 Date 物件) 儲存至 SharedObject 物件，而不需要手動序列化和還原序列化那些物件。

下列範例是以歡迎您第一次造訪開始。當您按一下 「Log Out」時，應用程式會將目前日期儲存至共享物件中。下次啟動此應用程式或重新整理頁面時，應用程式會提醒您上次登出的時間，歡迎您再次使用。

若要查看作用中的應用程式，請啟動應用程式，按一下 「Log Out」，然後重新整理頁面。此應用程式會顯示您在前次造訪時按一下 「Log Out」按鈕的日期和時間。您隨時都可以按一下 「Delete LSO」按鈕，以刪除儲存的資訊。

```
<?xml version="1.0"?>
<!-- lsos/WelcomeMessage.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize="initApp()">
  <mx:Script><![CDATA[
  public var mySO:SharedObject;
  [Bindable]
  public var welcomeMessage:String;

  public function initApp():void {
     mySO = SharedObject.getLocal("mydata");
     if (mySO.data.visitDate==null) {
        welcomeMessage = "Hello first-timer!"
     } else {
        welcomeMessage = "Welcome back. You last visited on " +
           getVisitDate();
     }
  }

  private function getVisitDate():Date {
     return mySO.data.visitDate;
  }

  private function storeDate():void {
     mySO.data.visitDate = new Date();
     mySO.flush();
  }

  private function deleteLSO():void {
     // Deletes the SharedObject from the client machine.
     // Next time they log in, they will be a 'first-timer'.
     mySO.clear();
  }

  ]]></mx:Script>
  <mx:Label id="label1" text="{welcomeMessage}"/>
  <mx:Button label="Log Out" click="storeDate()"/>
  <mx:Button label="Delete LSO" click="deleteLSO()"/>
</mx:Application>
```

加密本機儲存

EncryptedLocalStore 類別 (ELS) 提供加密本機儲存機制，您可以將這個機制當做應用程式私有資料的小型快取來使用。ELS 資料無法在應用程式之間共用。ELS 的用途是，讓應用程式可以輕鬆地儲存重新建立的項目 (例如登入憑證) 和其他私有資訊。系統不會將 ELS 資料視為永久資料，如下面的 「加密本機儲存區的限制」和 「最佳作法」所述。

備註：除了加密本機儲存區以外，AIR 也可為 SQL 資料庫的儲存內容加密。如需詳細資訊，請參閱第 652 頁 「為 SQL 資料庫加密」。

您可以使用加密本機儲存區快取機密資訊 (例如，網路服務的登入認證)，確保其安全性。ELS 很適合用來儲存必須對其他使用者保密的資訊。但是，它無法保護在相同使用者帳戶下執行之其他處理程序的資料。因此，不適用於保護機密的應用程式資料，例如 DRM 或加密金鑰。

在桌面平台上，AIR 在 Windows 中會使用 DPAPI (在 Mac OS 與 iOS 中使用 KeyChain，在 Linux 中則使用 KeyRing 或 KWallet) 使加密本機儲存區與每個應用程式及每個使用者產生關聯。加密本機儲存區會使用 AES-CBC 128 位元加密。

在 Android 上，不會加密 EncryptedLocalStorage 類別儲存的資料。資料改由作業系統提供的使用者層級安全功能來保護。Android 作業系統會針對每個應用程式指定個別的使用者 ID。應用程式只能存取它們自己的檔案與在公用位置建立的檔案 (例如抽取式記憶卡)。請注意，在根層級的 Android 裝置以根權限執行的應用程式，能夠存取其它應用程式的檔案。因為根層級的裝置不像非根層級的裝置，會在加密的本機儲存區提供高階的資料保護功能。

只有應用程式安全執行程序中的 AIR 應用程式內容才能存取加密本機儲存區中的資訊。

如果您更新 AIR 應用程式，更新的版本會保留加密本機儲存區中任何現有資料的存取權，除非您執行下列動作：

- 以 stronglyBound 參數設為 true 來加入這些項目。
- 現有版本及更新版本都是在 AIR 1.5.3 之前發佈，而且更新會以移轉簽章來簽署 .

加密本機儲存區的限制

加密本機儲存區中的資料受到使用者作業系統帳戶憑證的保護。其他實體除非以該使用者身份登入，否則無法存取儲存區的資料。不過，對於驗證的使用者所執行之其他應用程式存取的資料，並無法保護其安全。

由於使用者必須獲得驗證，這些攻擊才有作用，因此使用者的私人資料仍受到保護 (除非使用者帳戶本身有所洩漏)。但您應用程式希望保持機密、不受到使用者存取的資料 (例如授權或數位版權管理的金鑰) 並不安全。因此 ELS 並不是儲存這些資訊的適當位置。這裡僅適合儲存使用者的私人資料，如密碼。

ELS 的資料可能會因各種原因遺失。例如，使用者可能會解除安裝應用程式並刪除加密檔案。或者，發行者 ID 可能會在更新時遭到變更。因此，應該將 ELS 視為私人快取，而不是永久的資料儲存區。

不建議使用 stronglyBound 參數，不應將它設為 true。將此參數設為 true 不會對資料提供任何額外的保護。同時，應用程式每次更新時，都會喪失資料的存取權，即使發行者 ID 不變也一樣。

如果儲存的資料超過 10 MB，加密本機儲存區的效能可能變慢。

解除安裝 AIR 應用程式時，解除安裝程式不會刪除加密本機儲存區中的資料。

最佳作法

使用 ELS 的最佳作法包括：

- 使用 ELS 儲存密碼等機密使用者資料 (將 stronglyBound 設為 false)
- 不要使用 ELS 儲存 DRM 金鑰或授權憑證等應用程式機密。
- 可讓應用程式在遺失 ELS 資料時，重新建立儲存在 ELS 的資料。例如，在需要時提示使用者重新輸入帳戶憑證。
- 請勿使用 stronglyBound 參數。
- 如果您將 stronglyBound 設為 true，請勿在更新期間移轉儲存的項目。而是在更新之後重新建立資料。
- 僅儲存相當少量的資料。針對較大量的資料，請使用加密的 AIR SQL 資料庫。

更多說明主題

flash.data.EncryptedLocalStore

將資料加入加密的本機儲存區

使用 EncryptedLocalStore 類別的 setItem() 靜態方法，將資料儲存在本機儲存區。資料將儲存於雜湊表，使用字串做為索引鍵，資料則儲存為位元組陣列。

例如，下列程式碼將某字串儲存於加密本機儲存區：

```
var str:String = "Bob";
var bytes:ByteArray = new ByteArray();
bytes.writeUTFBytes(str);
EncryptedLocalStore.setItem("firstName", bytes);
```

setItem() 方法的第三個參數 stronglyBound 是選擇性參數。將此參數設為 true 時，加密本機儲存區會將儲存的項目繫結至儲存者 AIR 應用程式的數位簽名和位元：

```
var str:String = "Bob";
var bytes:ByteArray = new ByteArray();
bytes.writeUTFBytes(str);
EncryptedLocalStore.setItem("firstName", bytes, false);
```

若以 stronglyBound 設為 true 的方式儲存項目，則後續對 getItem() 的呼叫要能成功，呼叫端 AIR 應用程式必須與儲存者應用程式相同 (假設應用程式目錄中的檔案均未發生資料變更)。如果呼叫端 AIR 應用程式不同於儲存者應用程式，呼叫 getItem() 擷取強式繫結的項目時便會擲回 Error 例外。應用程式一旦更新，將無法讀取先前寫入加密本機儲存區的強式繫結資料。stronglyBound 參數在行動裝置上一律視為 false，所以會忽略將該參數設定為 true。

若 stronglyBound 參數設定成 false (預設值)，只有發行者 ID 需要保持相同，應用程式才能讀取資料。應用程式的位元可能會變更 (且必須由相同的發行者簽署)，但無須與儲存資料的應用程式保持完全相同的位元。擁有原始發行者 ID 的已更新應用程式可繼續存取資料。

備註：實際作業時，將 stronglyBound 設為 true 並不會對資料提供任何額外的保護。「惡意」使用者仍然可以變更應用程式，獲得儲存在 ELS 項目的存取權。此外，資料是從外部進行保護，無論 stronglyBound 設為 true 或 false，非使用者威脅還是很容易入侵。因為上述原因，不建議將 stronglyBound 設為 true。

在加密本機儲存區中存取資料

Adobe AIR 1.0 以及更新的版本

您可使用 EncryptedLocalStore.getItem() 方法，從加密本機儲存區中擷取值，如下範例所示：

```
var storedValue:ByteArray = EncryptedLocalStore.getItem("firstName");
trace(storedValue.readUTFBytes(storedValue.length)); // "Bob"
```

從加密本機儲存區移除資料

Adobe AIR 1.0 以及更新的版本

您可以使用 EncryptedLocalStore.removeItem() 方法，從加密本機儲存區刪除任何值，如下列範例所示：

```
EncryptedLocalStore.removeItem("firstName");
```

您可以呼叫 EncryptedLocalStore.reset() 方法，從加密本機儲存區中清除所有資料，如下列範例所示：

```
EncryptedLocalStore.reset();
```

第 40 章 在 AIR 中使用本機 SQL 資料庫

Adobe AIR 1.0 以及更新的版本

Adobe® AIR® 包含建立及使用本機 SQL 資料庫的功能。執行階段包含 SQL 資料庫引擎，支援許多標準 SQL 功能，並使用開放式來源 SQLite 資料庫系統。本機 SQL 資料庫可用來儲存本機的永續性資料。例如，它可以用於應用程式資料、應用程式使用者設定、文件，或者您要應用程式儲存於本機的任何其它類型資料。

關於本機 SQL 資料庫

Adobe AIR 1.0 以及更新的版本

如需使用 SQL 資料庫的快速說明與程式碼範例，請到 Adobe Developer Connection 參閱下列快速入門文章：

- 以非同步方式使用本機 SQL 資料庫 (Flex)
- 以同步方式使用本機 SQL 資料庫 (Flex)
- 使用加密資料庫 (Flex)
- 以非同步方式使用本機 SQL 資料庫 (Flash)
- 以同步方式使用本機 SQL 資料庫 (Flash)
- 使用加密資料庫 (Flash)

Adobe AIR 包含在執行階段中執行的 SQL 架構關聯式資料庫引擎，其資料儲存於執行 AIR 應用程式之電腦中的本機資料庫 (例如，在電腦硬碟中)。由於資料庫會在本機執行並在本機儲存資料檔案，不管是否連線到網路，資料庫都可以供 AIR 應用程式使用。因此，執行階段的本機 SQL 資料庫引擎提供了用來儲存永續本機應用程式資料的便利機制，如果您具有使用 SQL 和關聯式資料庫經驗，必定能感受到其方便之處。

本機 SQL 資料庫的用法

Adobe AIR 1.0 以及更新的版本

AIR 本機 SQL 資料庫功能可以用於您要在使用者的本機電腦上儲存應用程式資料的任何情況。Adobe AIR 包含多種可以在本機儲存資料的機制，每一種都有不同的優點。下列是 AIR 應用程式中本機 SQL 資料庫一些可能的用法：

- 對於資料導向的應用程式 (例如通訊錄) 來說，資料庫可以用來儲存主應用程式資料。
- 對於文件導向的應用程式來說，使用者可建立文件來儲存，也可以進行共用，每個文件都可以在使用者指定的位置儲存為資料庫檔案 (但請您注意，除非資料庫已經加密，否則任何 AIR 應用程式皆可開啟資料庫檔案。對於可能具有機密性的文件，建議使用加密)。
- 對於能辨識網路的應用程式來說，資料庫可以用來儲存應用程式資料的本機快取，或在無網路連線可用時暫時儲存資料。您可以使用網路資料存放區，建立同步處理本機資料庫的機制。
- 在任何應用程式中，資料庫可以用來儲存使用者的個人應用程式設定，例如視窗大小和位置等使用者選項或應用程式資訊。

更多說明主題

Christophe Coenraets：AIR for Android 上的員工目錄

Raymond Camden: jQuery 與 AIR - 從網頁移至應用程式

關於 AIR 資料庫和資料庫檔案

Adobe AIR 1.0 以及更新的版本

個別 Adobe AIR 本機 SQL 資料庫會在電腦檔案系統中儲存為單一檔案。執行階段包含 SQL 資料庫引擎，可管理資料庫檔案的建立和結構組織，並從資料庫檔案操控及擷取資料。執行階段不會指定資料庫資料儲存在檔案系統中的方式和位置，而會將個別資料庫完全儲存在單一檔案之內。請在檔案系統中指定儲存資料庫檔案的位置。單一 AIR 應用程式可以存取一個或多個個別資料庫 (也就是，不同的資料庫檔案)。由於執行階段會在檔案系統中將每個資料庫儲存為單一檔案，因此，您可以依應用程式的設計及作業系統的檔案存取限制，找出所需的資料庫。每一個使用者都可以使用個別的資料庫來儲存他們的特定資料，或者，應用程式在單一電腦上的所有使用者都可以存取資料庫檔案以取得共用的資料。由於資料會儲存在單一電腦的本機上，因此，不會自動提供不同電腦的使用者進行共用。本機 SQL 資料庫引擎不提供任何針對遠端或伺服器資料庫執行 SQL 陳述式的功能。

關於關聯式資料庫

Adobe AIR 1.0 以及更新的版本

關聯式資料庫是在電腦上儲存 (及擷取) 資料的機制。資料會組織成資料表：列代表記錄或項目，而欄 (有時稱為「欄位」) 則將每筆記錄分成各個值。例如，通訊錄應用程式可以包含 「朋友」資料表。資料表中的每個列都代表儲存在資料庫中的一個朋友。資料表的欄就代表資料，例如名字、姓氏、生日等等。針對資料表中的每個朋友列，資料庫會為每個欄儲存不同的值。

關聯式資料庫是針對儲存複雜資料所設計，當中的每一個項目都與其它類型的項目關聯或相關。在關聯式資料庫中，任何具備一對多關聯性的資料 (也就是，單一記錄可以與不同類型的多筆記錄相關) 都應該分別儲存於不同資料表中。例如，假設您要通訊錄應用程式為每個朋友儲存多個電話號碼，這就是一對多關聯性。此 「朋友」資料表會包含每個朋友的所有個人資訊。另外一個 「電話號碼」資料表則會包含所有朋友的所有電話號碼。

除了儲存有關朋友和電話號碼的資料之外，每個資料表都需要一段資料，來追蹤兩個資料表之間的關聯性，以便比對各個朋友的記錄及其電話號碼。這段資料就是主索引鍵，可以區別資料表中每個列與該資料表中其它列的唯一識別名稱。主索引鍵可以是 「自然索引鍵」，意思是，它是可以自然地區分資料表中每一筆記錄的其中一個資料項目。在 「朋友」資料表中，如果您知道沒有任何一個朋友的生日是相同的，就可以使用生日欄做為 「朋友」資料表的主索引鍵 (自然索引鍵)。如果沒有自然索引鍵，就要另外建立主索引鍵欄，例如 「朋友 ID」，這是任意設定的值，供應用程式用來區分各列。

您可以使用主索引鍵，設定多個資料表之間的關聯性。例如，假設 「朋友」資料表有 「朋友 ID」一欄，其中每個列 (每個朋友) 都包含唯一的號碼。相關的 「電話號碼」資料表可以建立包含兩欄的結構：一個是電話號碼所屬朋友的 「朋友 ID」，一個是實際的電話號碼。這樣，不管一個朋友有多少個電話號碼，全部都可以儲存在 「電話號碼」資料表中，而可以使用 「朋友 ID」主索引鍵連結到相關朋友。當一個資料表的主索引鍵會用於相關資料表中來指定記錄之間的關聯時，相關資料表中的值稱為外部索引鍵。與許多資料庫不同的是，AIR 本機資料庫引擎不允許您建立外部索引鍵限制，自動檢查插入或更新的外部索引鍵值在主索引鍵資料表中是否有對應的列。不過，外部索引鍵關聯性是關聯式資料庫結構的重要部分，而且在資料庫中的資料表之間建立關聯性時，應該要使用外部索引鍵。

關於 SQL

Adobe AIR 1.0 以及更新的版本

結構化查詢語言 (SQL) 可搭配關聯式資料庫使用，以操控及擷取資料。SQL 是描述式語言，而不是程序式語言。SQL 陳述式並不會提供電腦指示，告訴電腦應該如何擷取資料，而是描述您所要的一組資料。資料庫引擎會決定如何擷取該資料。

SQL 語言已經由美國國家標準局 (ANSI) 標準化。Adobe AIR 本機 SQL 資料庫支援大部分 SQL-92 標準。

如需 Adobe AIR 所支援之 SQL 語言的特定說明，請參閱第 952 頁 「本機資料庫內的 SQL 支援」。

關於 SQL 資料庫類別

Adobe AIR 1.0 以及更新的版本

若要在 ActionScript 3.0 中使用本機 SQL 資料庫，請使用 flash.data 套件中下列類別的實體：

類別	說明
flash.data.SQLConnection	提供可建立及開啟資料庫 (資料庫檔案) 的工具，也提供方法可執行資料庫層級作業及控制資料庫交易。
flash.data.SQLStatement	代表在資料庫上執行的單一 SQL 陳述式 (單一查詢或命令)，包括定義陳述式文字及設定參數值。
flash.data.SQLResult	提供取得執行陳述式相關資訊或結果的方式，例如 SELECT 陳述式的結果列、受 UPDATE 或 DELETE 陳述式影響的列數目等等。

若要取得描述資料庫結構的資料結構資訊，請使用 flash.data 套件中的下列類別：

類別	說明
flash.data.SQLSchemaResult	做為呼叫 SQLConnection.loadSchema() 方法所產生的資料庫資料結構結果之容器。
flash.data.SQLTableSchema	提供描述資料庫中單一資料表的資訊。
flash.data.SQLViewSchema	提供描述資料庫中單一檢視的資訊。
flash.data.SQLIndexSchema	提供描述資料庫中資料表或檢視之單一欄的資訊。
flash.data.SQLTriggerSchema	提供描述資料庫中單一觸發程序的資訊。

flash.data 套件中的其它類別會提供搭配 SQLConnection 類別和 SQLColumnSchema 類別所使用的常數：

類別	說明
flash.data.SQLMode	定義一組常數，代表 SQLConnection.open() 和 SQLConnection.openAsync() 方法的 openMode 參數的可能值。
flash.data.SQLColumnNameStyle	定義一組常數，代表 SQLConnection.columnNameStyle 屬性的可能值。
flash.data.SQLTransactionLockType	定義一組常數，代表 SQLConnection.begin() 方法的選項參數之可能值。
flash.data.SQLCollationType	定義一組常數，代表 SQLColumnSchema.defaultCollationType 屬性以及 SQLColumnSchema() 建構函式的 defaultCollationType 參數的可能值。

此外，flash.events 套件中的下列類別會代表您使用的事件 (和支援的常數)：

類別	說明
flash.events.SQLEvent	定義 SQLConnection 或 SQLStatement 實體在其任何作業成功執行後傳送的事件。SQLEvent 類別中定義了每項作業的關聯事件類型常數。
flash.events.SQLErrorEvent	定義 SQLConnection 或 SQLStatement 實體在其任何作業產生錯誤時傳送的事件。
flash.events.SQLUpdateEvent	當連線的其中一個資料庫內資料表資料因為執行 INSERT、UPDATE 或 DELETE SQL 陳述式而變更時，定義 SQLConnection 實體傳送的事件。

最後，flash.errors 套件中的下列類別會提供有關資料庫作業錯誤的資訊：

類別	說明
flash.errors.SQLError	提供有關資料庫作業錯誤的資訊，包括正在嘗試的作業以及失敗的原因。
flash.errors.SQLErrorOperation	定義一組常數，代表 SQLError 類別的 operation 屬性之可能值，指出產生錯誤的資料庫作業。

關於同步和非同步執行模式

Adobe AIR 1.0 以及更新的版本

當您撰寫程式碼來使用本機 SQL 資料庫時，需要從兩個執行模式中指定一個模式以執行資料庫作業：非同步或同步執行模式。本節中的程式碼範例會示範如何以這兩種方式執行每一項作業，讓您能夠選擇最適合您需求的範例。

在非同步執行模式中，您需要對執行階段下達指示，然後執行階段會在您要求的作業完成或失敗時傳送事件。首先，請指示資料庫引擎執行作業。在應用程式繼續執行的同時，資料庫引擎會在背景中進行工作。最後，當作業完成 (或失敗) 時，資料庫引擎會傳送事件。事件會觸發您的程式碼，執行後續作業。這種做法有一項重大的優點：在主應用程式碼繼續執行的同時，執行階段會在背景中執行資料庫作業。如果資料庫作業明顯耗費相當長的時間，應用程式會繼續執行。最重要的是，使用者可以繼續與應用程式互動，而螢幕畫面並不會凍結。不過，非同步作業程式碼可能會比撰寫另一種程式碼更複雜。這種複雜度通常會發生在多個相依作業必須分成各種不同事件偵聽程式方法的情況。

就概念上來說，將作業撰寫成單一步驟順序 (也就是一組同步作業) 的程式碼，而不是分成多個事件偵聽程式方法的一組作業，會比較簡單。除了非同步資料庫作業以外，Adobe AIR 也可以讓您以同步方式執行資料庫作業。在同步執行模式中，作業不會在背景中執行，而會跟其它所有應用程式程式碼在相同的執行順序中執行。請指示資料庫引擎執行作業。接著，程式碼會在資料庫引擎進行工作時暫停。當作業完成之後，就會繼續執行程式碼的下一行。

作業要以非同步方式或同步方式執行，會在 SQLConnection 層級上設定。使用單一資料庫連線，您就無法以同步方式執行一些作業或陳述式，同時又以非同步方式執行其它作業或陳述式。您可以呼叫 SQLConnection 方法來開啟資料庫，指定 SQLConnection 應在同步或非同步執行模式中執行作業。如果您呼叫 SQLConnection.open()，則連線會在同步執行模式中執行作業；如果您呼叫 SQLConnection.openAsync()，則連線作業會在非同步執行模式中執行作業。一旦 SQLConnection 實體使用 open() 或 openAsync() 連線至資料庫之後，就會固定以同步或非同步執行模式作業，除非您關閉並重新開啟資料庫連線。

每一種執行模式都各有優點。雖然每一種模式的大多很類似，但是在各模式中工作時，還是需要注意當中的一些差異。如需有關這些主題的詳細資訊，以及使用各模式的建議，請參閱第 648 頁「使用同步和非同步資料庫作業」。

建立及修改資料庫

Adobe AIR 1.0 以及更新的版本

在您的應用程式能夠新增或擷取資料以前，必須在應用程式中定義您應用程式能夠存取的資料庫。本節描述建立資料庫及在資料庫中建立資料結構的工作。雖然這些工作的使用程度不如資料偵聽和擷取那麼頻繁，但是大部分應用程式都需要執行這些工作。

更多說明主題

注意 Flex：更新現有的 AIR 資料庫

建立資料庫

Adobe AIR 1.0 以及更新的版本

若要建立資料庫檔案，首先要建立 SQLConnection 實體。然後呼叫其 open() 方法，在同步執行模式中開啟，或呼叫其 openAsync() 方法在非同步執行模式中開啟。open() 和 openAsync() 方法都可用來開啟資料庫的連線。如果您傳遞 File 實體來參考沒有 reference 參數 (第一個參數) 的檔案位置，open() 或 openAsync() 方法在會該檔案位置建立資料庫檔案，然後開啟新建立資料庫的連線。

不管是呼叫 open() 方法或 openAsync() 方法來呼叫資料庫，資料庫檔案的名稱都可以是任何有效的檔案名稱加上任何副檔名。如果您呼叫 open() 或 openAsync() 方法，而 reference 參數為 null，就會建立新的記憶體內資料庫，而不是在硬碟上建立資料庫檔案。

下面列出的程式碼會示範使用非同步執行模式來建立資料庫檔案 (新資料庫) 的程序。在此情況下，資料庫檔案會儲存於第 578 頁「指向應用程式儲存目錄」中，檔案名稱是「DBSample.db」：

```
import flash.data.SQLConnection;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

conn.openAsync(dbFile);

function openHandler(event:SQLEvent):void
{
    trace("the database was created successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                conn.openAsync(dbFile);
            }

            private function openHandler(event:SQLEvent):void
            {
                trace("the database was created successfully");
            }

            private function errorHandler(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

備註：雖然 File 類別可讓您指向特定原生檔案路徑，但這麼做會讓應用程式無法跨平台使用。例如，C:\Documents and Settings\joe\test.db 這個路徑僅適用於 Windows。因此，最好使用 File 類別的靜態屬性，如 File.applicationStorageDirectory，和 resolvePath() 方法 (如上一個範例所示)。如需詳細資訊，請參閱 第 575 頁 「File 物件的路徑」。

若要以同步方式執行作業，當您使用 SQLConnection 實體開啟資料庫連線，請呼叫 open() 方法。下列範例會示範如何建立及開啟會以同步方式執行其作業的 SQLConnection 實體：

```
import flash.data.SQLConnection;
import flash.errors.SQLError;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

try
{
    conn.open(dbFile);
    trace("the database was created successfully");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.errors.SQLError;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                try
                {
                    conn.open(dbFile);
                    trace("the database was created successfully");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

建立資料庫資料表

Adobe AIR 1.0 以及更新的版本

在資料庫中建立資料表包含在該資料庫上執行 SQL 陳述式、使用與用來執行 SELECT、INSERT 之類 SQL 陳述式相同的處理等等。若要建立資料表，請使用 CREATE TABLE 陳述式，該陳述式包含了欄的定義和新資料表的限制。如需有關執行 SQL 陳述式的詳細資訊，請參閱第 627 頁「使用 SQL 陳述式」。

下列範例會示範如何使用非同步執行模式，在現有資料庫檔案中建立名為「employees」的資料表。請注意，此程式碼會假設已有一個實體化且連線至資料庫的 SQLConnection 實體 conn。

```
import flash.data.SQLConnection;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

var createStmt:SQLStatement = new SQLStatement();
createStmt.sqlConnection = conn;

var sql:String =
    "CREATE TABLE IF NOT EXISTS employees (" +
    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "   firstName TEXT, " +
    "   lastName TEXT, " +
    "   salary NUMERIC CHECK (salary > 0)" +
    ")";
createStmt.text = sql;

createStmt.addEventListener(SQLEvent.RESULT, createResult);
createStmt.addEventListener(SQLErrorEvent.ERROR, createError);

createStmt.execute();

function createResult(event:SQLEvent):void
{
    trace("Table created");
}

function createError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                var createStmt:SQLStatement = new SQLStatement();
                createStmt.sqlConnection = conn;

                var sql:String =
                    "CREATE TABLE IF NOT EXISTS employees (" +
                    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    "   firstName TEXT, " +
                    "   lastName TEXT, " +
                    "   salary NUMERIC CHECK (salary > 0)" +
                    ")";
                createStmt.text = sql;
```

```
                createStmt.addEventListener(SQLEvent.RESULT, createResult);
                createStmt.addEventListener(SQLErrorEvent.ERROR, createError);

                createStmt.execute();
            }

            private function createResult(event:SQLEvent):void
            {
                trace("Table created");
            }

            private function createError(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

下列範例會示範如何使用同步執行模式，在現有資料庫檔案中建立名為「employees」的資料表。請注意，此程式碼會假設已有一個實體化且連線至資料庫的 SQLConnection 實體 conn。

```
import flash.data.SQLConnection;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

var createStmt:SQLStatement = new SQLStatement();
createStmt.sqlConnection = conn;

var sql:String =
    "CREATE TABLE IF NOT EXISTS employees (" +
    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "   firstName TEXT, " +
    "   lastName TEXT, " +
    "   salary NUMERIC CHECK (salary > 0)" +
    ")";
createStmt.text = sql;

try
{
    createStmt.execute();
    trace("Table created");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                var createStmt:SQLStatement = new SQLStatement();
                createStmt.sqlConnection = conn;

                var sql:String =
                    "CREATE TABLE IF NOT EXISTS employees (" +
                    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    "   firstName TEXT, " +
                    "   lastName TEXT, " +
                    "   salary NUMERIC CHECK (salary > 0)" +
                    ")";
                createStmt.text = sql;

                try
                {
                    createStmt.execute();
                    trace("Table created");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

操控 SQL 資料庫資料

Adobe AIR 1.0 以及更新的版本

當您使用本機 SQL 資料庫時，需要執行一些常見工作。這些工作包括連線至資料庫、在資料庫的資料表中加入及擷取資料。在執行這些工作時，也需要注意一些問題，例如使用資料類型和處理錯誤。

請注意，有些資料庫工作雖然不需要經常進行處理，但是往往必須先行處理才能執行更多常見工作。舉例來說，在連線至資料庫並從資料表擷取資料之前，您必須建立資料庫並在資料庫中建立資料表結構。這些比較不頻繁的起始設定工作會在第 618 頁「建立及修改資料庫」中討論。

您可以選擇以非同步方式執行資料庫作業，也就是，資料庫引擎會在背景中執行，並在作業成功或失敗時，透過傳送事件來通知您。您也可以同步方式來執行這些作業。此時，資料庫作業會一項接著一項執行，而整個應用程式 (包括螢幕的更新) 都會先等待作業完成，再執行其它程式碼。如需有關在非同步或同步執行模式中工作的詳細資訊，請參閱第 648 頁「使用同步和非同步資料庫作業」。

連線到資料庫

Adobe AIR 1.0 以及更新的版本

在您能夠執行任何資料庫作業之前，必須先開啟資料庫檔案的連線。SQLConnection 實體會用來代表一或多個資料庫的連線。使用 SQLConnection 實體進行連線的第一個資料庫稱為「主」資料庫。這個資料庫會使用 open() 方法 (適用於同步執行模式) 或 openAsync() 方法 (適用於非同步執行模式) 進行連線。

如果您使用非同步 openAsync() 作業開啟資料庫，請註冊 SQLConnection 實體的 open 事件，才能知道 openAsync() 作業何時完成。註冊 SQLConnection 實體的 error 事件，以判斷作業是否失敗。

下列範例會示範如何以非同步執行開啟現有的資料庫檔案。資料庫檔案稱為「DBSample.db」，且位於使用者的第 578 頁「指向應用程式儲存目錄」。

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

conn.openAsync(dbFile, SQLMode.UPDATE);

function openHandler(event:SQLEvent):void
{
    trace("the database opened successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLMode;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                conn.openAsync(dbFile, SQLMode.UPDATE);
            }

            private function openHandler(event:SQLEvent):void
            {
                trace("the database opened successfully");
            }

            private function errorHandler(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

下列範例會示範如何以同步執行開啟現有的資料庫檔案。資料庫檔案稱為「DBSample.db」，且位於使用者的第 578 頁「指向應用程式儲存目錄」。

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.errors.SQLError;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

try
{
    conn.open(dbFile, SQLMode.UPDATE);
    trace("the database opened successfully");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLMode;
            import flash.errors.SQLError;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                try
                {
                    conn.open(dbFile, SQLMode.UPDATE);
                    trace("the database opened successfully");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

請注意，非同步範例的 openAsync() 方法呼叫中，以及非同步範例的 open() 方法中，第二個引數是常數 SQLMode.UPDATE。為第二個參數 (openMode) 指定 SQLMode.UPDATE 會在指定的檔案不存在時，導致執行階段傳送錯誤。如果您為 openMode 參數傳遞 SQLMode.CREATE (或如果您讓 openMode 參數保持為關閉)，執行階段會在指定的檔案不存在時，嘗試建立資料庫檔案。不過，檔案如果存在就會開啟，就和使用 SQLMode.Update 的情形一樣。您也可以為 openMode 參數指定 SQLMode.READ，以唯讀模式開啟現有的資料庫。在此情況下，可以從資料庫中擷取資料，但是無法加入、刪除或變更資料。

使用 SQL 陳述式

Adobe AIR 1.0 以及更新的版本

在執行階段中，各個 SQL 陳述式 (查詢或命令) 都會以 SQLStatement 物件來代表。請依照下列步驟，建立並執行 SQL 陳述式：

建立 **SQLStatement** 實體。
SQLStatement 物件代表您應用程式中的 SQL 陳述式。

```
var selectData:SQLStatement = new SQLStatement();
```

指定要針對哪個資料庫執行查詢。
若要執行這項作業，請將 SQLStatement 物件的 sqlConnection 屬性設定為與所想要資料庫連線的 SQLConnection 實體。

```
// A SQLConnection named "conn" has been created previously
selectData.sqlConnection = conn;
```

指定實際的 **SQL** 陳述式。
將陳述式文字建立為 String，並指定給 SQLStatement 實體的 text 屬性。

```
selectData.text = "SELECT col1, col2 FROM my_table WHERE col1 = :param1";
```

定義函數以處理執行作業的結果 (僅適用於非同步執行模式)。
使用 addEventListener() 方法，將函數註冊為 SQLStatement 實體之 result 和 error 事件的偵聽程式。

```
// using listener methods and addEventListener()

selectData.addEventListener(SQLEvent.RESULT, resultHandler);
selectData.addEventListener(SQLErrorEvent.ERROR, errorHandler);

function resultHandler(event:SQLEvent):void
{
    // do something after the statement execution succeeds
}

function errorHandler(event:SQLErrorEvent):void
{
    // do something after the statement execution fails
}
```

另外，您也可以使用 Responder 物件，指定偵聽程式方法。在此情況下，您可以建立 Responder 實體，然後將偵聽程式方法連結至該實體。

```
// using a Responder (flash.net.Responder)

var selectResponder = new Responder(onResult, onError);

function onResult(result:SQLResult):void
{
    // do something after the statement execution succeeds
}

function onError(error:SQLError):void
{
    // do something after the statement execution fails
}
```

如果陳述式文字包括參數定義，請為這些參數指定值。
若要指定參數值，請使用 SQLStatement 實體的 parameters 關聯陣列屬性。

```
selectData.parameters[":param1"] = 25;
```

執行 **SQL** 陳述式。
呼叫 SQLStatement 實體的 execute() 方法。

```
// using synchronous execution mode
// or listener methods in asynchronous execution mode
selectData.execute();
```

此外，如果您是在非同步執行模式中使用 Responder，而不是使用事件偵聽程式，請將 Responder 實體傳遞給 execute() 方法。

```
// using a Responder in asynchronous execution mode
selectData.execute(-1, selectResponder);
```

如需示範這些步驟的特定範例，請參閱下列主題：

第 631 頁 「從資料庫擷取資料」

第 640 頁 「插入資料」

第 645 頁 「變更或刪除資料」

在陳述式中使用參數

Adobe AIR 1.0 以及更新的版本

使用 SQL 陳述式參數可以讓您建立可重複使用的 SQL 陳述式。當您使用陳述式參數時，陳述式之中的值可以變更 (例如正在加入 INSERT 陳述式中的值)，但基本的陳述式文字維持不變。所以，使用參數可提供效能上的優勢，同時也更容易撰寫應用程式的程式碼。

瞭解陳述式參數

Adobe AIR 1.0 以及更新的版本

應用程式經常會在應用程式中多次使用單一 SQL 陳述式，不過會稍做變化。例如，以存貨追蹤應用程式為例，使用者可以將新的存貨項目加入資料庫。將存貨項目加入資料庫的應用程式程式碼會執行 SQL INSERT 陳述式，此陳述式會實際將資料加入至資料庫。但是，每次執行陳述式，陳述式就會有些微的改變。明確地說，也就是，插入資料表中的實際值會不同，因為這些值屬於正在加入的存貨項目。

當您具有會在陳述式中以不同的值多次使用之 SQL 陳述式時，最佳的做法就是使用包含參數的 SQL 陳述式，而不要在 SQL 文字中使用常值。參數是陳述式文字中的預留位置，會在每次執行陳述式時以實際的值取代。若要在 SQL 陳述式中使用參數，請依一般做法建立 SQLStatement 實體。對於實際指定給 text 屬性的 SQL 陳述式，請使用參數預留位置，而不要使用常值。然後，請設定 SQLStatement 實體之 parameters 屬性中元素的值，以定義每個參數的值。parameters 屬性是關聯陣列，因此請使用下列語法來設定特定的值：

```
statement.parameters[parameter_identifier] = value;
```

如果您使用命名的參數，則 parameter_identifier 會是字串；如果您使用未命名的參數，就會是整數索引。

使用命名的參數

Adobe AIR 1.0 以及更新的版本

參數可以是命名的參數。命名的參數具有特定的名稱，可供資料庫用來比對參數值及其陳述式文字中的預留位置。參數名稱會包含 「:」或 「@」字元，後面加上名稱，如下列範例所示：

```
:itemName
@firstName
```

下面列出的程式碼會示範命名的參數用法：

```
var sql:String =
    "INSERT INTO inventoryItems (name, productCode)" +
    "VALUES (:name, :productCode)";

var addItemStmt:SQLStatement = new SQLStatement();
addItemStmt.sqlConnection = conn;
addItemStmt.text = sql;

// set parameter values
addItemStmt.parameters[":name"] = "Item name";
addItemStmt.parameters[":productCode"] = "12345";

addItemStmt.execute();
```

使用未命名的參數

Adobe AIR 1.0 以及更新的版本

除了使用命名的參數之外，您也可以使用未命名的參數。若要使用未命名的參數，請在 SQL 陳述式中使用 「?」字元表示參數。每個參數都會根據陳述中參數的順序指定數字索引，第一個參數從索引 0 開始。下列範例會示範在上一個範例中使用未命名參數的版本：

```
var sql:String =
    "INSERT INTO inventoryItems (name, productCode)" +
    "VALUES (?, ?)";

var addItemStmt:SQLStatement = new SQLStatement();
addItemStmt.sqlConnection = conn;
addItemStmt.text = sql;

// set parameter values
addItemStmt.parameters[0] = "Item name";
addItemStmt.parameters[1] = "12345";

addItemStmt.execute();
```

使用參數的優點

Adobe AIR 1.0 以及更新的版本

在 SQL 陳述式中使用參數有多項優點：

效能更佳 使用參數的 SQLStatement 實體執行起來會比在每次執行時以動態方式建立 SQL 文字的實體更有效率。效能改進的最大原因在於，陳述式只會準備一次，之後就能使用不同的參數值執行多次，而不必重新編譯 SQL 陳述式。

明確的資料輸入 使用參數後，便可以將建構 SQL 陳述式時未知的值取代為所輸入資料。使用參數是唯一可確保傳入資料庫之值使用儲存類別的方式。如果不使用參數，執行階段會嘗試將所有的值根據關聯欄的類型相似性，從文字形式轉換成儲存類別。

如需儲存類別和欄相似性的詳細資訊，請參閱第 969 頁 「資料類型支援」。

更高的安全性 使用參數可協助避免稱為 SQL 插入式攻擊的惡意技巧。在 SQL 插入式攻擊中，使用者會在使用者可存取的位置 (例如，資料輸入欄位) 輸入 SQL 程式碼。如果應用程式程式碼是直接將使用者輸入連接至 SQL 文字中以建構 SQL 陳述式，則使用者所輸入的 SQL 程式碼會針對資料庫來執行。以下列出的程式碼會示範將使用者輸入連接至 SQL 文字中的範例，請勿使用此技巧：

```
// assume the variables "username" and "password"
// contain user-entered data

var sql:String =
    "SELECT userId " +
    "FROM users " +
    "WHERE username = '" + username + "' " +
    "    AND password = '" + password + "'";

var statement:SQLStatement = new SQLStatement();
statement.text = sql;
```

使用陳述式參數來取代將使用者輸入值連接至陳述式的文字，可以防止 SQL 插入式攻擊。由於參數值會視為明確的替代值，而不是陳述式文字常值的一部分，因此不會發生 SQL 插入。以下為上述程式碼範例的建議替代方式：

```
// assume the variables "username" and "password"
// contain user-entered data

var sql:String =
    "SELECT userId " +
    "FROM users " +
    "WHERE username = :username " +
    "    AND password = :password";

var statement:SQLStatement = new SQLStatement();
statement.text = sql;

// set parameter values
statement.parameters[":username"] = username;
statement.parameters[":password"] = password;
```

從資料庫擷取資料

Adobe AIR 1.0 以及更新的版本

從資料庫擷取資料包含兩個步驟。首先，您必須執行 SQL SELECT 陳述式，描述要從資料庫擷取的資料集。然後，您要存取已擷取的資料，並視應用程式所需來顯示或操控。

執行 SELECT 陳述式

Adobe AIR 1.0 以及更新的版本

若要從資料庫擷取現有資料，請使用 SQLStatement 實體。將適當的 SQL SELECT 陳述式指定給實體的 text 屬性，然後呼叫其 execute() 方法。

如需 SELECT 陳述式語法的詳細資訊，請參閱第 952 頁 「本機資料庫內的 SQL 支援」。

下列範例會示範如何使用非同步執行模式，執行 SELECT 陳述式，從名稱為 「products」的資料表中擷取資料：

```
var selectStmt:SQLStatement = new SQLStatement();

// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;

selectStmt.text = "SELECT itemId, itemName, price FROM products";

selectStmt.addEventListener(SQLEvent.RESULT, resultHandler);
selectStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);

selectStmt.execute();

function resultHandler(event:SQLEvent):void
{
    var result:SQLResult = selectStmt.getResult();

    var numResults:int = result.data.length;
    for (var i:int = 0; i < numResults; i++)
    {
        var row:Object = result.data[i];
        var output:String = "itemId: " + row.itemId;
        output += "; itemName: " + row.itemName;
        output += "; price: " + row.price;
        trace(output);
    }
}

function errorHandler(event:SQLErrorEvent):void
{
    // Information about the error is available in the
    // event.error property, which is an instance of
    // the SQLError class.
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                var selectStmt:SQLStatement = new SQLStatement();

                // A SQLConnection named "conn" has been created previously
                selectStmt.sqlConnection = conn;

                selectStmt.text = "SELECT itemId, itemName, price FROM products";

                selectStmt.addEventListener(SQLEvent.RESULT, resultHandler);
                selectStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);

                selectStmt.execute();
            }

            private function resultHandler(event:SQLEvent):void
            {
                var result:SQLResult = selectStmt.getResult();
```

```
                var numResults:int = result.data.length;
                for (var i:int = 0; i < numResults; i++)
                {
                    var row:Object = result.data[i];
                    var output:String = "itemId: " + row.itemId;
                    output += "; itemName: " + row.itemName;
                    output += "; price: " + row.price;
                    trace(output);
                }
            }

            private function errorHandler(event:SQLErrorEvent):void
            {
                // Information about the error is available in the
                // event.error property, which is an instance of
                // the SQLError class.
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

下列範例會示範如何使用同步執行模式，執行 SELECT 陳述式，從名稱為「products」的資料表中擷取資料：

```
var selectStmt:SQLStatement = new SQLStatement();

// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;

selectStmt.text = "SELECT itemId, itemName, price FROM products";

try
{
    selectStmt.execute();

    var result:SQLResult = selectStmt.getResult();

    var numResults:int = result.data.length;
    for (var i:int = 0; i < numResults; i++)
    {
        var row:Object = result.data[i];
        var output:String = "itemId: " + row.itemId;
        output += "; itemName: " + row.itemName;
        output += "; price: " + row.price;
        trace(output);
    }
}
catch (error:SQLError)
{
    // Information about the error is available in the
    // error variable, which is an instance of
    // the SQLError class.
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                var selectStmt:SQLStatement = new SQLStatement();

                // A SQLConnection named "conn" has been created previously
                selectStmt.sqlConnection = conn;

                selectStmt.text = "SELECT itemId, itemName, price FROM products";

                try
                {
                    selectStmt.execute();

                    var result:SQLResult = selectStmt.getResult();

                    var numResults:int = result.data.length;
                    for (var i:int = 0; i < numResults; i++)
                    {
                        var row:Object = result.data[i];
                        var output:String = "itemId: " + row.itemId;
                        output += "; itemName: " + row.itemName;
                        output += "; price: " + row.price;
                        trace(output);
                    }
                }
                catch (error:SQLError)
                {
                    // Information about the error is available in the
                    // error variable, which is an instance of
                    // the SQLError class.
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

在非同步執行模式中，當陳述式完成執行時，SQLStatement 實體會傳送 result 事件 (SQLEvent.RESULT) 指出已成功地執行該陳述式。或者，如果傳遞 Responder 物件做為 execute() 方法的引數，就會呼叫 Responder 物件的結果處理常式函數。在同步執行模式中，執行程序會暫停至 execute() 作業完成後，再繼續執行下一行程式碼。

存取 SELECT 陳述式結果資料

Adobe AIR 1.0 以及更新的版本

在 SELECT 陳述式完成執行後，下一個步驟就是存取已擷取的資料。您可以呼叫 SQL 陳述式的 getResult() 方法來執行 SELECT 陳述式，這樣就可以擷取結果：

```
var result:SQLResult = selectStatement.getResult();
```

getResult() 方法會傳回 SQLResult 物件。SQLResult 物件的 data 屬性是 Array，其中包含 SELECT 陳述式的結果：

```
var numResults:int = result.data.length;
for (var i:int = 0; i < numResults; i++)
{
    // row is an Object representing one row of result data
    var row:Object = result.data[i];
}
```

SELECT 結果集的每個資料列會是 Object 實體 (包含於 data Array)。該物件具備的屬性，其名稱會與結果集的欄名稱相符。屬性中會包含結果集各欄的值。例如，假設 SELECT 陳述式指定有三個欄的結果集，而欄的名稱分別為「itemId」、「itemName」和「price」。針對結果集中的每一個列，會使用名稱 itemId、itemName 和 price 的屬性建立 Object 實體。這些屬性包含其對應欄的值。

下面列出的程式碼會定義文字是 SELECT 陳述式的 SQLStatement 實體。陳述式所擷取的列會包含命名為 employees 的資料表中所有列的 firstName 和 lastName 欄值。此範例會使用非同步執行模式。當執行完成之後，就會呼叫 selectResult() 方法，並使用 SQLStatement.getResult() 存取結果資料列，然後使用 trace() 方法顯示。請注意，此程式碼範例假設已有一個實體化且連線至資料庫的 SQLConnection 實體 conn。它也假設「employees」資料表已建立並填入資料。

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var selectStmt:SQLStatement = new SQLStatement();
selectStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "SELECT firstName, lastName " +
    "FROM employees";
selectStmt.text = sql;

// register listeners for the result and error events
selectStmt.addEventListener(SQLEvent.RESULT, selectResult);
selectStmt.addEventListener(SQLErrorEvent.ERROR, selectError);

// execute the statement
selectStmt.execute();

function selectResult(event:SQLEvent):void
{
```

```
    // access the result data
    var result:SQLResult = selectStmt.getResult();

    var numRows:int = result.data.length;
    for (var i:int = 0; i < numRows; i++)
    {
        var output:String = "";
        for (var columnName:String in result.data[i])
        {
            output += columnName + ": " + result.data[i][columnName] + "; ";
        }
        trace("row[" + i.toString() + "]\t", output);
    }
}

function selectError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var selectStmt:SQLStatement = new SQLStatement();
                selectStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "SELECT firstName, lastName " +
                    "FROM employees";
                selectStmt.text = sql;

                // register listeners for the result and error events
                selectStmt.addEventListener(SQLEvent.RESULT, selectResult);
                selectStmt.addEventListener(SQLErrorEvent.ERROR, selectError);

                // execute the statement
                selectStmt.execute();
            }

            private function selectResult(event:SQLEvent):void
            {
                // access the result data
                var result:SQLResult = selectStmt.getResult();
```

```
                var numRows:int = result.data.length;
                for (var i:int = 0; i < numRows; i++)
                {
                    var output:String = "";
                    for (var columnName:String in result.data[i])
                    {
                        output += columnName + ": " + result.data[i][columnName] + "; ";
                    }
                    trace("row[" + i.toString() + "]\t", output);
                }
            }

            private function selectError(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

下面列出的程式碼會示範與上一個範例相同的技巧，但使用同步執行模式。此範例會定義文字是 SELECT 陳述式的 SQLStatement 實體。陳述式所擷取的列會包含命名為 employees 的資料表中所有列的 firstName 和 lastName 欄值。結果資料列會使用 SQLStatement.getResult() 存取，並使用 trace() 方法顯示。請注意，此程式碼範例假設已有一個實體化且連線至資料庫的 SQLConnection 實體 conn。它也假設「employees」資料表已建立並填入資料。

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var selectStmt:SQLStatement = new SQLStatement();
selectStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "SELECT firstName, lastName " +
    "FROM employees";
selectStmt.text = sql;

try
{
    // execute the statement
    selectStmt.execute();
```

```
    // access the result data
    var result:SQLResult = selectStmt.getResult();

    var numRows:int = result.data.length;
    for (var i:int = 0; i < numRows; i++)
    {
        var output:String = "";
        for (var columnName:String in result.data[i])
        {
            output += columnName + ": " + result.data[i][columnName] + "; ";
        }
        trace("row[" + i.toString() + "]\t", output);
    }
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var selectStmt:SQLStatement = new SQLStatement();
                selectStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "SELECT firstName, lastName " +
                    "FROM employees";
                selectStmt.text = sql;

                try
                {
                    // execute the statement
                    selectStmt.execute();

                    // access the result data
                    var result:SQLResult = selectStmt.getResult();

                    var numRows:int = result.data.length;
```

```
                    for (var i:int = 0; i < numRows; i++)
                    {
                        var output:String = "";
                        for (var columnName:String in result.data[i])
                        {
                            output += columnName + ": ";
                            output += result.data[i][columnName] + "; ";
                        }
                        trace("row[" + i.toString() + "]\t", output);
                    }
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

定義 SELECT 結果資料的資料類型

Adobe AIR 1.0 以及更新的版本

根據預設，由 SELECT 陳述式傳回的每列都會建立為 Object 實體，其屬性名稱會使用結果集之欄名稱來命名，而每個欄的值會做為其關聯陣列的值。但是，在執行 SQL SELECT 陳述式之前，您可以將 SQLStatement 實體的 itemClass 屬性設定為類別。透過設定 itemClass 屬性，由 SELECT 陳述式傳回的每列都會建立為指定類別的實體。執行階段會透過讓 SELECT 結果集中的欄名稱與 itemClass 類別中的屬性名稱相符，將結果欄值指定為屬性值。

指定為 itemClass 屬性值的任何類別都必須具有不需要任何參數的建構函式。此外，對於 SELECT 陳述式傳回的每一欄，此類別還必須包含單一屬性。如果 SELECT 清單中某一欄的屬性名稱與 itemClass 類別中的屬性名稱不符，就會被視為錯誤。

擷取部分 SELECT 結果

Adobe AIR 1.0 以及更新的版本

根據預設，執行 SELECT 陳述式會一次擷取結果集中的所有列。一旦陳述式完成之後，通常會以某種方式處理已擷取的資料，例如建立物件或在螢幕上顯示資料。如果陳述式傳回很多列，一次處理所有資料會對電腦產生極大需求，進而造成使用者介面無法重新繪製。

您可以指示執行階段一次傳回特定數目的結果列，以顯著改進應用程式效能。這種作業方式可讓起始結果資料更快速傳回。同時，也能讓您將結果列分成多組，以便讓使用者在處理完每組列之後更新。請注意，這種技巧只有在非同步執行中才會有實際效用。

若要分批擷取 SELECT 結果，請為 SQLStatement.execute() 方法第一個參數 (prefetch 參數) 指定值。prefetch 參數會指出第一次執行陳述式時要擷取的列數。當您呼叫 SQLStatement 實體的 execute() 方法時，便會視您指定的 prefetch 參數值，來擷取該數目的列：

```
var stmt:SQLStatement = new SQLStatement();
stmt.sqlConnection = conn;

stmt.text = "SELECT ...";

stmt.addEventListener(SQLEvent.RESULT, selectResult);

stmt.execute(20); // only the first 20 rows (or fewer) are returned
```

陳述式會傳送 result 事件，指出可使用第一組結果列。結果 SQLResult 實體的 data 屬性會包含資料列，而且其 complete 屬性會指出還有其它結果列仍待擷取。若要擷取其它結果列，請呼叫 SQLStatement 實體的 next() 方法。就像 execute() 方法一樣，next() 方法的第一個參數會用來指出下一次傳送 result 事件時所要擷取的列數。

```
function selectResult(event:SQLEvent):void
{
    var result:SQLResult = stmt.getResult();
    if (result.data != null)
    {
        // ... loop through the rows or perform other processing ...

        if (!result.complete)
        {
            stmt.next(20); // retrieve the next 20 rows
        }
        else
        {
            stmt.removeEventListener(SQLEvent.RESULT, selectResult);
        }
    }
}
```

每次 next() 方法傳回下一組結果列時，SQLStatement 都會傳送 result 事件。如此一來，相同的偵聽程式就可以繼續處理 (來自 next() 呼叫的) 結果，直到擷取所有列為止。

如需詳細資訊，請參閱有關 SQLStatement.execute() 方法 (prefetch 參數說明) 和 SQLStatement.next() 方法的說明。

插入資料

Adobe AIR 1.0 以及更新的版本

將資料新增至資料庫，涉及執行 SQL INSERT 陳述式。陳述式執行完成之後，如果索引鍵是由資料庫產生的，您就可以存取新插入列的主索引鍵。

執行 INSERT 陳述式

Adobe AIR 1.0 以及更新的版本

若要將資料加入資料庫中的資料表，請建立並執行文字是 SQL INSERT 陳述式的 SQLStatement 實體。

下列範例會使用 SQLStatement 實體，將資料列加入現有的 「employees」資料表。此範例會示範如何使用非同步執行模式插入資料。請注意，此程式碼假設有一個已實體化而且連線至資料庫的 SQLConnection 實體 conn。它也假設「employees」資料表已建立。

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var insertStmt:SQLStatement = new SQLStatement();
insertStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "INSERT INTO employees (firstName, lastName, salary) " +
    "VALUES ('Bob', 'Smith', 8000)";
insertStmt.text = sql;

// register listeners for the result and failure (status) events
insertStmt.addEventListener(SQLEvent.RESULT, insertResult);
insertStmt.addEventListener(SQLErrorEvent.ERROR, insertError);

// execute the statement
insertStmt.execute();

function insertResult(event:SQLEvent):void
{
    trace("INSERT statement succeeded");
}

function insertError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var insertStmt:SQLStatement = new SQLStatement();
                insertStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "INSERT INTO employees (firstName, lastName, salary) " +
                    "VALUES ('Bob', 'Smith', 8000)";
                insertStmt.text = sql;

                // register listeners for the result and failure (status) events
```

```
                insertStmt.addEventListener(SQLEvent.RESULT, insertResult);
                insertStmt.addEventListener(SQLErrorEvent.ERROR, insertError);

                // execute the statement
                insertStmt.execute();
            }

            private function insertResult(event:SQLEvent):void
            {
                trace("INSERT statement succeeded");
            }

            private function insertError(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

下列範例會使用同步執行模式，將資料列加入現有的「employees」資料表。請注意，此程式碼假設有一個已實體化而且連線至資料庫的 SQLConnection 實體 conn。它也假設「employees」資料表已建立。

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var insertStmt:SQLStatement = new SQLStatement();
insertStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "INSERT INTO employees (firstName, lastName, salary) " +
    "VALUES ('Bob', 'Smith', 8000)";
insertStmt.text = sql;

try
{
    // execute the statement
    insertStmt.execute();

    trace("INSERT statement succeeded");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var insertStmt:SQLStatement = new SQLStatement();
                insertStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "INSERT INTO employees (firstName, lastName, salary) " +
                    "VALUES ('Bob', 'Smith', 8000)";
                insertStmt.text = sql;

                try
                {
                    // execute the statement
                    insertStmt.execute();
                    trace("INSERT statement succeeded");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

擷取已插入列的資料庫所產生的主索引鍵

Adobe AIR 1.0 以及更新的版本

將資料列插入資料表之後，您的程式碼必須知道新插入列之資料庫所產生的主索引鍵或列識別名稱值。例如，在一個資料表中插入列後，可能也要將列加入相關的資料表中。在此情況下，請插入主索引鍵值做為相關資料表中的外部索引鍵。新插入列的主索引鍵可以使用與陳述式執行有關的 SQLResult 物件來擷取。這個物件就是在執行 SELECT 陳述式之後，用來存取執行結果資料的物件。與任何 SQL 陳述式相同的是，當 INSERT 陳述式執行完成之後，執行階段會建立 SQLResult 實體。如果使用事件偵聽程式，或使用同步執行模式，請透過呼叫 SQLStatement 物件的 getResult() 方法來存取 SQLResult 實體。另外，如果您使用非同步執行模式，並傳遞 Responder 實體至 execute() 呼叫，則 SQLResult 實體會傳遞並做為結果處理常式函數的引數。不論是哪種情況，如果已執行的 SQL 陳述式是 INSERT 陳述式，則 SQLResult 實體都會具有屬性 lastInsertRowID，其中包含最近插入之列的列識別名稱。

下列範例會示範在非同步執行模式中存取已插入列的主索引鍵：

```
insertStmt.text = "INSERT INTO ...";

insertStmt.addEventListener(SQLEvent.RESULT, resultHandler);

insertStmt.execute();

function resultHandler(event:SQLEvent):void
{
    // get the primary key
    var result:SQLResult = insertStmt.getResult();

    var primaryKey:Number = result.lastInsertRowID;
    // do something with the primary key
}
```

下列範例會示範在同步執行模式中存取已插入列的主索引鍵：

```
insertStmt.text = "INSERT INTO ...";

try
{
    insertStmt.execute();

    // get the primary key
    var result:SQLResult = insertStmt.getResult();

    var primaryKey:Number = result.lastInsertRowID;
    // do something with the primary key
}
catch (error:SQLError)
{
    // respond to the error
}
```

請注意，根據下列規則，列識別名稱可能會也可能不會在資料表定義中指定為主索引鍵欄的欄值：

- 如果資料表是使用其相似性 (欄資料類型) 為 INTEGER 的主索引鍵欄定義，則 lastInsertRowID 屬性會包含已插入該列的值 (或者，如果是 AUTOINCREMENT 欄，則為執行階段所產生的值)。
- 如果資料表是使用多個主索引鍵欄 (複合式索引鍵)，或使用其相似性不是 INTEGER 的單一主索引鍵欄定義，則資料庫會在幕後為該列產生一個整數列識別名稱值。所產生的值是 lastInsertRowID 屬性的值。
- 該值永遠都是最近插入列的列識別名稱。如果 INSERT 陳述式導致觸發程序發生而插入列，則 lastInsertRowID 屬性會包含由觸發程序最後插入之列的列識別名稱，而不是由 INSERT 陳述式所建立之列的列識別名稱。

這些規則的結果就是，如果您要有明確定義的主索引鍵欄，而其值可在執行 INSERT 命令之後透過 SQLResult.lastInsertRowID 屬性取得，則欄必須定義為 INTEGER PRIMARY KEY 欄。即使您的資料表並未包含明確的 INTEGER PRIMARY KEY 欄，在定義與相關資料表的關聯性時，一樣可以接受使用資料庫所產生的列識別名稱做為資料表的主索引鍵。列識別名稱欄值可以透過使用 ROWID、_ROWID_ 或 OID 其中一個特殊欄名稱，於任何 SQL 陳述式中取得。您可以在相關資料表中建立外部索引鍵欄，並使用列識別名稱值做為外部索引鍵欄值，跟使用明確宣告的 INTEGER PRIMARY KEY 欄完全一樣。在此情形下，如果您使用任意主索引鍵而不是自然索引鍵，而且您不在乎執行階段會為您產生主索引鍵值，那麼，不管您使用 INTEGER PRIMARY KEY 欄或系統產生的列識別名稱做為資料表的主索引鍵，以定義兩個資料表之間的外部索引鍵關聯性，都不會有太大的差別。

如需主要索引鍵和產生之列識別名稱的詳細資訊，請參閱第 952 頁「本機資料庫內的 SQL 支援」。

變更或刪除資料

Adobe AIR 1.0 以及更新的版本

執行其它資料操控作業的程序與用來執行 SQL SELECT 或 INSERT 陳述式的程序完全相同，如第 627 頁 「使用 SQL 陳述式」所述。只要在 SQLStatement 實體的 text 屬性中以不同的 SQL 陳述式取代即可。

- 若要變更資料表中現有的資料，請使用 UPDATE 陳述式。
- 若要從資料表刪除一個或多個資料列，請使用 DELETE 陳述式。

如需這些陳述式的說明，請參閱第 952 頁 「本機資料庫內的 SQL 支援」。

使用多個資料庫

Adobe AIR 1.0 以及更新的版本

您可以使用 SQLConnection.attach() 方法，開啟已經具有已開啟資料庫之 SQLConnection 實體上其它資料庫的連線。請在 attach() 方法呼叫中使用名稱參數，將名稱提供給附加的資料庫。撰寫陳述式來操控該資料庫時，就可以在前置詞中使用該名稱 (使用 database-name.table-name 的格式) 來限定 SQL 陳述式中的任何資料表名稱，向執行階段指出該資料表可以在命名的資料庫中找到。

您可以執行單一 SQL 陳述式，其中包括連線至相同 SQLConnection 實體的多個資料庫之資料表。如果交易是在 SQLConnection 實體上建立的，該交易會套用於使用 SQLConnection 實體執行的所有 SQL 陳述式。不管陳述式是執行哪個附加資料庫都是如此。

或者，您也可以在應用程式中建立多個 SQLConnection 實體，其中每一個都連線到一個或多個資料庫。但是，如果您確實使用多個相同資料庫的連線，請務必記住，資料庫交易並不會跨 SQLConnection 實體共用。因此，如果您使用多個 SQLConnection 實體連線至相同的資料庫檔案，就不能指望兩個連線的資料變更會以預期的方式套用。例如，如果兩個 UPDATE 或 DELETE 陳述式都透過不同的 SQLConnection 實體對相同的資料庫執行，而在執行一個作業後應用程式發生錯誤，資料庫資料可能會停留在中間狀態，而無法回復，並可能影響資料庫的完整性 (進而影響到應用程式)。

處理資料庫錯誤

Adobe AIR 1.0 以及更新的版本

大致上，處理資料庫錯誤就像處理其它執行階段錯誤一樣。您應該撰寫程式碼，為可能發生的錯誤做好準備，並回應錯誤而不是留待執行階段來決定處理方式。在一般情形下，可能的資料庫錯誤可以分成三大類：連線錯誤、SQL 語法錯誤，以及限制錯誤。

連線錯誤

Adobe AIR 1.0 以及更新的版本

大部分資料庫錯誤都是連線錯誤，而可能會在任何作業期間發生。雖然有策略可以防止連線錯誤，如果資料庫是應用程式的重要部分，卻很少有簡單的方式，能夠妥善處理而從連線錯誤中恢復。

大部分連線錯誤與執行階段跟作業系統、檔案系統和資料庫檔案之間互動的方式有關。例如，如果使用者沒有權限可以在檔案系統上的特定位置中建立資料庫，就會發生連線錯誤。下列策略可以協助防止連線錯誤：

使用使用者特定的資料庫檔案 請給予每個使用者自己的資料庫檔案，而不是讓所有在單一電腦上使用應用程式的使用者使用單一資料庫檔案。檔案應該位於與使用者帳戶關聯的目錄中。例如，它可以位於應用程式的儲存目錄、使用者的文件資料夾、使用者的桌面等等之中。

考量不同的使用者類型 請使用不同類型的使用者帳戶，在不同的作業系統上測試您的應用程式。請不要假設使用者在電腦上具有系統管理員權限。同時，也不要假設安裝應用程式的個人必定是執行應用程式的使用者。

考慮各種不同的檔案位置 如果您允許使用者指定儲存資料庫檔案的位置或選取要開啟檔案的位置，請考慮使用者會使用的檔案的可能位置。此外，請考慮定義有關使用者可以儲存 (或可從中開啟) 資料庫檔案之位置的限制。例如，您可以只讓使用者開啟在其使用者帳戶之儲存位置內的檔案。

如果發生連線錯誤，最可能發生在初次嘗試建立或開啟資料庫時。這表示，使用者無法在應用程式中執行任何資料庫相關作業。對於特定類型的錯誤，例如唯讀或權限錯誤，其中一個可能的復原技巧是將資料庫檔案複製至不同的位置。應用程式可以將資料庫檔案複製至使用者確實具有權限的不同位置，以建立及寫入至檔案，並改用該位置。

語法錯誤

Adobe AIR 1.0 以及更新的版本

SQL 陳述式格式不正確，而應用程式卻嘗試執行該陳述式時，便會發生語法錯誤。由於本機資料庫 SQL 陳述式會建立為字串，因此無法進行編譯階段 SQL 語法檢查。原因在於，所有 SQL 陳述式都必須先被執行，才能檢查其語法。您可以使用下列策略來防止 SQL 語法錯誤：

徹底測試所有 SQL 陳述式 可能的話，請先在開發應用程式時另行測試 SQL 陳述式，再將陳述式編寫為應用程式程式碼的陳述式文字。此外，您也能使用程式碼測試手法 (例如單位測試)，建立一組測試，執行程式碼中每一項可能的選項和變化。

使用陳述式參數並避免組合 (以動態方式產生的) SQL 請使用參數並避免以動態方式建立 SQL 陳述式，也就是，在每次執行陳述式時使用相同的 SQL 陳述式文字。這能讓您輕鬆測試您的陳述式，並限制可能的變化。如果您必須以動態方式產生 SQL 陳述式，請將陳述式的動態部分維持在最低限度。同時，請仔細驗證任何使用者輸入，以確保不會造成語法錯誤。

若要從語法錯誤中復原，應用程式需要複雜的邏輯，才能夠檢查 SQL 陳述式並修正其語法。若能遵循上述有關防止語法錯誤的準則，您的程式碼便能識別 SQL 語法錯誤 (例如使用者輸入用於陳述式中) 的在執行階段的任何潛在來源。若要從語法錯誤復原，請為使用者提供準則。請指出應該修正哪些錯誤，才能讓陳述式正常執行。

限制錯誤

Adobe AIR 1.0 以及更新的版本

INSERT 或 UPDATE 陳述式嘗試將資料加入至欄時，便會發生限制錯誤。如果新資料違反為資料表或欄定義的其中一項限制，就會發生錯誤。可能的限制組合包括：

唯一限制 指出資料表的所有列中，一個欄不可以有重複的值。或者，在唯一限制中，當多個欄結合時，這些欄的值組合不可以重複。換句話說，就指定的一個或多個唯一欄而言，每個列都必須不同。

主索引鍵限制 就限制允許與不允許的資料而言，主索引鍵限制與唯一限制完全相同。

非 null 限制 指定單一欄無法儲存 NULL 值，因此在每個列中，該欄都必須有值。

檢查限制 讓您可在一個或多個資料表上指定任意限制。其中一種常見的檢查限制為定義該欄的值必須在特定範圍之內的規則 (例如，數值欄的值必須大於 0)。另一種常見檢查限制會指定欄值之間的關係 (例如，欄的值必須與相同列中其它欄的值不同)。

資料類型 (欄相似性) 限制 執行階段會強制套用欄值的資料類型，如果嘗試在欄中儲存不正確類型的值，就會發生錯誤。但是，在許多情況下，值會進行轉換，以便與欄的宣告資料類型相符。如需詳細資訊，請參閱第 647 頁 「使用資料庫資料類型」。

執行階段不會在外部索引鍵值上強制套用限制。換句話說，外部索引鍵值不需要與現有主索引鍵值相符。

除了預先定義的限制類型以外，執行階段 SQL 引擎也支援使用觸發程序。觸發程序與事件處理常式類似。它是預先定義的一組指示，會在發生特定動作時執行。例如，可以定義觸發程序，在資料插入特定資料表或從中刪除時執行。觸發程序的其中一個可能用法，就是檢查資料變更，而且如果未符合指定的條件，就會發生錯誤。因此，觸發程序可以與限制達到相同的目的，防止限制錯誤並從錯誤中復原的策略也適用於觸發程序產生的錯誤。但是，觸發程序所產生錯誤的錯誤 ID 與限制錯誤的錯誤 ID 不同。

適用於特定資料表的這組限制會在您設計應用程式的階段決定。仔細地設計限制，有助於簡化應用程式的設計，也能預防限制錯誤並從限制錯誤中復原。但是，限制錯誤很難以系統化的方式加以預測及避免。難以預測的原因在於，限制錯誤在加入應用程式資料之前不會出現。限制錯誤會與建立資料庫之後加入的資料一起發生。這些錯誤往往是新資料與資料庫中現有資料之間建立關聯所造成的結果。下列策略可以協助您避免許多限制錯誤：

仔細規劃資料庫結構與限制 限制的目的是強制套用應用程式規則，並協助保護資料庫資料的完整性。當您規劃應用程式時，請考慮如何建立您的資料庫結構以支援您的應用程式。該程序的其中一部分是找出資料的規則，例如是否需要特定值、值是否為預設值、是否允許重複的值等等。這些規則可以引導您定義資料庫限制。

明確指定欄名稱 您可以撰寫 INSERT 陳述式且不明確指定要插入欄中的值，但是這種做法會產生不必要的風險。透過明確命名要插入值的欄，您可以允許自動產生值、使用預設值的欄，以及允許 NULL 值的欄。此外，這麼做的話，您就可以確保所有 NOT NULL 欄都已插入明確的值。

使用預設值 當您為欄指定 NOT NULL 限制時，如果可以的話，請在欄定義中指定預設值。應用程式程式碼也可以提供預設值。例如，您的程式碼可以檢查 String 變數是否為 null，並為它指定值，再將它用來設定陳述式參數值。

驗證使用者輸入的資料 請事先檢查使用者輸入的資料，以確保它遵守由限制所指定的限制，尤其是在 NOT NULL 和 CHECK 限制的情況下。當然，UNIQUE 限制較難檢查，因為進行檢查時需要執行 SELECT 查詢，以判斷資料是否為唯一。

使用觸發程序 您可以撰寫驗證 (而且可能取代) 插入資料的觸發程序，或採取其它行動來修正無效的資料。這項驗證和修正可以防止限制錯誤發生。

在很多方面，限制錯誤會比其它類型錯誤更難預防。幸運的是，許多策略可以用來從限制錯誤中復原，而且不會造成應用程式不穩定或無法使用：

使用衝突演算法 當您在欄上定義限制，以及當您建立 INSERT 或 UPDATE 陳述式時，您可以選擇指定衝突演算法。衝突演算法會在發生違反限制的情況時，定義資料庫所採取的行動。資料庫引擎可以採取多項可能的行動。資料庫引擎可以結束單一陳述式或整個交易。它可以忽略錯誤。它甚至可以移除舊資料，然後用程式碼嘗試要儲存的資料取代。

如需詳細資訊，請參閱第 952 頁 「本機資料庫內的 SQL 支援」中的 「ON CONFLICT (衝突演算法)」小節。

提供正確的回應 可能會影響特定 SQL 命令的一組限制是可以事先識別的。因此，您可以預期陳述式可能會造成的限制錯誤。瞭解這點之後，您可以建立應用程式邏輯來回應限制錯誤。例如，假設應用程式包含供輸入新產品的資料輸入表單。如果資料庫中的產品名稱欄是使用 UNIQUE 限制來定義，則將新產品列插入資料庫的動作可能會產生限制錯誤。因此，應用程式會設計用來預期限制錯誤。當錯誤發生時，應用程式會警告使用者，指出指定的產品名稱已經在使用中，並請使用者選擇不同的名稱。另一個可能的回應，則是允許使用者檢視其它名稱相同產品的相關資訊。

使用資料庫資料類型

Adobe AIR 1.0 以及更新的版本

在資料庫中建立資料表時，供建立資料表的 SQL 陳述式會為資料表中各欄定義相似性或資料類型。雖然相似性宣告可以省略，但是在 CREATE TABLE SQL 陳述式中明確地宣告欄相似性會是比較適當的做法。

原則上，您使用 INSERT 陳述式儲存在資料庫中的任何物件，會在執行 SELECT 陳述式時傳回，做為相同資料類型的實體。但是，依儲存值之資料庫欄的相似性而定，已擷取值的資料類型可能會不同。當值儲存在欄中時，如果其資料類型與欄相似性不相符，資料庫會嘗試轉換值以符合欄相似性。例如，如果資料庫欄使用 NUMERIC 相似性宣告，資料庫會嘗試將插入的資料轉換成數值儲存類別 (INTEGER 或 REAL)，再儲存資料。如果無法轉換資料，資料庫會擲回錯誤。根據這項規則，如果將 String 「12345」插入 NUMERIC 欄，資料庫會自動將它轉換成整數值 12345，再將它存入資料庫。當它是以 SELECT 陳述式擷取時，會傳回值，做為數值資料類型 (例如 Number) 的實體，而不是 String 實體。

若要避免不想要的資料類型轉換的最佳方式，請遵循以下兩個規則：首先，使用相似性定義各欄，該相似性要與它預定要儲存的資料類型相符。然後，只插入其資料類型符合已定義相似性的值。遵循這兩個規則提可提供兩項優點：當您插入資料時，它不會以非預期方式進行轉換 (可能會因此遺失其原先的意義)。此外，當您擷取資料時，它會以其原始資料類型傳回。

如需可用欄相似性類型以及在 SQL 陳述式中使用資料類型的詳細資訊，請參閱第 969 頁 「資料類型支援」。

使用同步和非同步資料庫作業

Adobe AIR 1.0 以及更新的版本

上面各節已探討常見的資料庫作業，例如擷取、插入、更新，與刪除資料，以及在資料庫之中建立資料庫檔案和資料表及其它物件。範例也已示範如何以同步和非同步方式執行這些作業。

在此提醒您，在非同步執行模式中，您需要指示資料庫引擎執行作業。然後，在應用程式繼續執行的同時，資料庫引擎會在背景中進行工作。作業完成後，資料庫引擎會傳送事件，告知您該作業已完成。非同步執行的主要優點，在於執行階段會在背景中執行資料庫作業，而主應用程式的程式碼可以繼續執行。在需要耗費相當長時間來執行作業時，這種做法尤其有用。

另一方面，在同步執行模式中，作業不會在背景中執行。請指示資料庫引擎執行作業。程式碼會在資料庫引擎進行工作時暫停。當作業完成之後，就會繼續執行程式碼的下一行。

使用單一資料庫連線，您就無法以同步方式執行一些作業或陳述式，同時又以非同步方式執行其它作業或陳述式。您指定在開啟資料庫連線時，SQLConnection 會以同步或非同步方式作業。如果您呼叫 SQLConnection.open()，則連線會在同步執行模式中執行作業；如果您呼叫 SQLConnection.openAsync()，則連線作業會在非同步執行模式中執行作業。一旦 SQLConnection 實體使用 open() 或 openAsync() 連線到資料庫，就會固定使用同步或非同步執行。

使用同步資料庫作業

Adobe AIR 1.0 以及更新的版本

與非同步執行模式的程式碼相較之下，使用同步執行時，您用來執行及回應作業的實際程式碼會有些微差異。兩種做法之間的主要差異有兩方面：首先，執行的作業需取決於其它作業 (例如，SELECT 結果列或由 INSERT 陳述式所加入列的主索引鍵)，其次，就是處理錯誤方面的差異。

為同步作業撰寫程式碼

Adobe AIR 1.0 以及更新的版本

同步與非同步執行之間的主要差異在於：同步執行模式中必須將程式碼撰寫成單一系列步驟。相反地，在非同步程式碼中，您需要登錄事件偵聽程式，而且經常需要將作業分配給所有偵聽程式方法。當資料庫以同步執行模式連線時，您可以在單一程式碼區塊中，依序執行一系列資料庫作業。下列範例將示範這種技巧：

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, OpenMode.UPDATE);

// start a transaction
conn.begin();

// add the customer record to the database
var insertCustomer:SQLStatement = new SQLStatement();
insertCustomer.sqlConnection = conn;
insertCustomer.text =
    "INSERT INTO customers (firstName, lastName) " +
    "VALUES ('Bob', 'Jones')";
insertCustomer.execute();

var customerId:Number = insertCustomer.getResult().lastInsertRowID;

// add a related phone number record for the customer
var insertPhoneNumber:SQLStatement = new SQLStatement();
insertPhoneNumber.sqlConnection = conn;
insertPhoneNumber.text =
    "INSERT INTO customerPhoneNumbers (customerId, number) " +
    "VALUES (:customerId, '800-555-1234')";
insertPhoneNumber.parameters[":customerId"] = customerId;
insertPhoneNumber.execute();

// commit the transaction
conn.commit();
```

如您所見，不管是使用同步或非同步執行，您都會呼叫相同方法來執行資料庫作業。兩種做法之間的主要差異在於：執行的作業需取決於其它作業，及處理錯誤的方式。

執行的作業需取決於其它作業

Adobe AIR 1.0 以及更新的版本

當您使用同步執行模式時，不必撰寫偵聽事件的程式碼，以判斷作業何時完成；您可以認定，如果一行程式碼中的作業順利完成，就會繼續執行下一行程式碼。因此，若要執行取決於其它作業成功的作業，只要撰寫在所相依作業之後立即執行的相依程式碼即可。例如，若要編寫應用程式以開始交易，請執行 INSERT 陳述式，擷取已插入列的主索引鍵，將該索引鍵插入不同資料表中的其它列，最後認可交易，程式碼可以撰寫為一系列的陳述式。下列範例會示範這些作業：

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, SQLMode.UPDATE);

// start a transaction
conn.begin();

// add the customer record to the database
var insertCustomer:SQLStatement = new SQLStatement();
insertCustomer.sqlConnection = conn;
insertCustomer.text =
    "INSERT INTO customers (firstName, lastName) " +
    "VALUES ('Bob', 'Jones')";
insertCustomer.execute();

var customerId:Number = insertCustomer.getResult().lastInsertRowID;

// add a related phone number record for the customer
var insertPhoneNumber:SQLStatement = new SQLStatement();
insertPhoneNumber.sqlConnection = conn;
insertPhoneNumber.text =
    "INSERT INTO customerPhoneNumbers (customerId, number) " +
    "VALUES (:customerId, '800-555-1234')";
insertPhoneNumber.parameters[":customerId"] = customerId;
insertPhoneNumber.execute();

// commit the transaction
conn.commit();
```

以同步執行處理錯誤

Adobe AIR 1.0 以及更新的版本

在同步執行模式中，您不會偵聽錯誤事件，以判斷作業是否已失敗，而會將任何可以觸發錯誤的程式碼包覆在一組 try..catch..finally 程式碼區塊中。您將擲出錯誤的程式碼包覆在 try 區塊中。撰寫要執行的動作，以便在不同 catch 區塊中，回應各類型的錯誤。請將無論成功或失敗都要執行的任何程式碼 (例如，關閉不再需要的資料庫連線) 放置在 finally 區塊中。下列範例將示範如何使用 try..catch..finally 區塊，以進行錯誤處理。它會以上一個範例為基礎，並加入錯誤處理程式碼：

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, SQLMode.UPDATE);

// start a transaction
conn.begin();

try
{
    // add the customer record to the database
    var insertCustomer:SQLStatement = new SQLStatement();
    insertCustomer.sqlConnection = conn;
    insertCustomer.text =
        "INSERT INTO customers (firstName, lastName)" +
        "VALUES ('Bob', 'Jones')";

    insertCustomer.execute();

    var customerId:Number = insertCustomer.getResult().lastInsertRowID;

    // add a related phone number record for the customer
    var insertPhoneNumber:SQLStatement = new SQLStatement();
    insertPhoneNumber.sqlConnection = conn;
    insertPhoneNumber.text =
        "INSERT INTO customerPhoneNumbers (customerId, number)" +
        "VALUES (:customerId, '800-555-1234')";
    insertPhoneNumber.parameters[":customerId"] = customerId;

    insertPhoneNumber.execute();

    // if we've gotten to this point without errors, commit the transaction
    conn.commit();
}
catch (error:SQLError)
{
    // rollback the transaction
    conn.rollback();
}
```

瞭解非同步執行模式

Adobe AIR 1.0 以及更新的版本

使用非同步執行模式有一項常見的隱憂，就是如果其它 SQLStatement 目前正針對相同的資料庫連線執行，就會推斷您不能開始執行 SQLStatement 實體。事實上，這種推斷並不正確。在 SQLStatement 實體執行時，您不能變更陳述式的 text 屬性，但是，如果您要執行的各個不同 SQL 陳述式都使用不同的 SQLStatement 實體，您可以呼叫 SQLStatement 的 execute() 方法，同一時間，其它 SQLStatement 實體仍然在執行中，並不會導致錯誤。

在內部，當您使用非同步執行模式執行資料庫作業時，每個資料庫連線 (每個 SQLConnection 實體) 本身都具備其佇列或指示要執行的作業清單。執行階段會依序執行各項作業，依加入佇列中的順序執行。當您建立 SQLStatement 實體並呼叫其 execute() 方法時，該陳述式執行作業會加入連線的佇列。如果目前在該 SQLConnection 實體上並沒有執行作業，陳述式會開始在背景中執行。假設在相同程式碼區塊之內，您建立了其它 SQLStatement 實體，而且也呼叫該方法的 execute() 方法。這

第二個陳述式執行作業就會加入佇列中第一個陳述式之後。第一個陳述式完成執行之後，執行階段就會繼續佇列中的下一項作業。即使第一項作業的 result 事件正在主應用程式的程式碼中傳送，佇列中後續作業的處理也會在背景中進行。下列程式碼會示範這項技巧：

```
// Using asynchronous execution mode
var stmt1:SQLStatement = new SQLStatement();
stmt1.sqlConnection = conn;

// ... Set statement text and parameters, and register event listeners ...

stmt1.execute();

// At this point stmt1's execute() operation is added to conn's execution queue.

var stmt2:SQLStatement = new SQLStatement();
stmt2.sqlConnection = conn;

// ... Set statement text and parameters, and register event listeners ...

stmt2.execute();

// At this point stmt2's execute() operation is added to conn's execution queue.
// When stmt1 finishes executing, stmt2 will immediately begin executing
// in the background.
```

資料庫若自動執行佇列中的後續陳述式，會有一項重大的副作用。如果陳述式取決於其它作業的結果，就不能將陳述式加入佇列中 (也就是說，您不能呼叫其 execute() 方法)，必須要等到第一項作業完成後才行。這是因為，一旦您呼叫第二個陳述式的 execute() 方法，就不能變更陳述式的 text 或 parameters 屬性。在此情況下，您必須等候事件指出第一項作業已經完成，才能開始第二項作業。例如，如果您要在交易中執行陳述式，而該陳述式取決於正在開啟的交易作業。呼叫 SQLConnection.begin() 方法開啟交易之後，您必須等候 SQLConnection 實體傳送其 begin 事件。之後，您才能呼叫 SQLStatement 實體的 execute() 方法。在此範例中，組織應用程式以確保作業正確執行的最簡單方式，就是建立註冊為 begin 事件之偵聽程式的方法。呼叫 SQLStatement.execute() 方法的程式碼會放置於該偵聽程式方法之中。

為 SQL 資料庫加密

Adobe AIR 1.5 以及更新的版本

所有的 Adobe AIR 應用程式都使用相同的本機資料庫引擎。因此，所有 AIR 應用程式都可以對未加密的資料庫檔案進行連線、讀取和寫入。從 Adobe AIR 1.5 開始，AIR 便納入了建立和連線未加密資料庫檔案的功能。當您使用加密資料庫時，應用程式如果想連線至資料庫，就必須提供正確的加密金鑰。如果提供的加密金鑰有誤 (或沒有提供金鑰)，應用程式就無法連線至資料庫。因此，應用程式就無法讀取資料庫的資料，也無法將資料寫入或變更。

若要使用加密資料庫，您必須將資料庫建立為加密資料庫。若有現存的加密資料庫，您就可以開啟對該資料庫的連線。您也可以變更加密資料庫的加密金鑰。除了建立和連線的方法不同以外，加密資料庫的使用方法都與非加密資料庫一樣，尤其是 SQL 陳述式的執行方法，不論資料庫是否加密皆相同。

加密資料庫的用法

Adobe AIR 1.5 以及更新的版本

每當您想針對資料庫所儲存的資料進行存取限制時，加密就是一個很有用的方法。Adobe AIR 的資料庫加密功能有多項用途，下列是幾個您可以使用加密資料庫的例子：

- 從伺服器下載應用程式私人資料的唯讀快取。

- 私人資料的本機應用程式儲存區與伺服器進行同步 (資料可傳送至伺服器亦可從伺服器載入)。
- 應用程式所建立和編輯的文件採用加密檔案格式。這些檔案可供單一使用者私人使用，也可以指定讓應用程式的所有使用者共用。
- 本機資料儲存區其它必須讓資料保持私密的用途 (例如第 615 頁 「本機 SQL 資料庫的用法」所描述的幾個用途)，避免所有具電腦或資料庫檔案存取權的人員都來存取資料。

若能瞭解自己為何需要使用加密資料庫，便可輕易判斷應該如何架構應用程式，尤其是因為這會影響應用程式將如何為資料庫建立、取得或儲存加密金鑰。如需關於這些考量事項的詳細資訊，請參閱第 656 頁 「為資料庫加密時的考量事項」。

除了加密資料庫以外，另外還有一種可以讓機密資料保持隱私的機制，就是加密本機儲存區。使用加密本機儲存區時，您將使用 String 金鑰來儲存單一的 ByteArray 值。只有儲存該值的 AIR 應用程式可以加以存取，而且只能在儲存該值的電腦上進行存取。如果使用加密本機儲存區，就不需要建立自己的加密金鑰。基於這些原因，加密本機儲存區最適合儲存單一值或是可在 ByteArray 中輕易完成編碼的單一組值。加密資料庫最適合較大型的資料集，此類資料集需要結構性資料儲存和查詢。如需有關使用加密本機儲存區的詳細資訊，請參閱第 612 頁 「加密本機儲存」。

建立加密資料庫

Adobe AIR 1.5 以及更新的版本

若要使用加密資料庫，建立資料庫檔案時就必須將其加密。如果資料庫在建立時沒有加密，之後就不能再進行加密。同樣地，已加密的資料庫也不能再解密。如果有需要，您可以變更加密資料庫的加密金鑰。如需詳細資訊，請參閱第 655 頁 「變更資料庫的加密金鑰」。如果您的現有資料庫沒有加密，而您想使用資料庫加密，則可以建立一個新的加密資料庫，然後將現有的資料表結構和資料複製到新資料庫中。

加密資料庫與非加密資料庫的建立方法幾乎一樣，如第 619 頁 「建立資料庫」一節所述。您可以先建立一個 SQLConnection 實體來代表對資料庫的連線，然後呼叫 SQLConnection 物件的 open() 方法或 openAsync() 方法為資料庫位置指定一個尚未存在的檔案，藉此方法建立資料庫。建立加密資料庫時的唯一差別就是，您需要提供 encryptionKey 參數的值 (也就是 open() 方法的第五個參數及 openAsync() 方法的第六個參數)。

有效的 encryptionKey 參數值就是一個剛好 16 位元組的 ByteArray 物件。

以下範例示範如何建立加密的資料庫。為求簡化，這些範例中的加密金鑰在應用程式碼中使用硬式編碼。不過，我們非常不建議您使用這項技巧，因為很不安全。

```
var conn:SQLConnection = new SQLConnection();

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

// Create an encrypted database in asynchronous mode
conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);

// Create an encrypted database in synchronous mode
conn.open(dbFile, SQLMode.CREATE, false, 1024, encryptionKey);
```

如需一個以建議方式建立加密金鑰的範例說明，請參閱第 657 頁 「範例：產生並使用加密金鑰」。

連線至加密資料庫

Adobe AIR 1.5 以及更新的版本

就像建立加密資料庫時情況一樣，對加密資料庫進行連線的程序也和連線非加密資料庫的程序類似。第 625 頁 「連線到資料庫」一節對此程序有較為詳細的說明。您可以使用 open() 方法在同步執行模式中開啟連線，或使用 openAsync() 模式在非同步執行模式中開啟連線。開啟加密資料庫時的唯一差別就是，您需要為 encryptionKey 參數指定正確的值 (也就是 open() 方法的第五個參數及 openAsync() 方法的第六個參數)。

如果提供的加密資料不正確，就會發生錯誤。對於 open() 方法，將擲出一個 SQLError 例外。對於 openAsync() 方法，SQLConnection 物件會傳送 SQLErrorEvent，其中的 error 屬性含有一個 SQLError 物件。在這兩種情況下，例外情形所產生之 SQLError 物件的 errorID 屬性值為 3138。該錯誤 ID 所對應的錯誤訊息為「所開啟的檔案並非資料庫檔案」。

以下範例說明如何在非同步執行模式中開啟加密資料庫。為求簡化，此範例中的加密金鑰在應用程式碼中使用硬式編碼。不過，我們非常不建議您使用這項技巧，因為很不安全。

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();
conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);
var dbFile:File = File.applicationStorageDirectory.resolvePath("DBSample.db");

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

conn.openAsync(dbFile, SQLMode.UPDATE, null, false, 1024, encryptionKey);

function openHandler(event:SQLEvent):void
{
    trace("the database opened successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    if (event.error.errorID == 3138)
    {
        trace("Incorrect encryption key");
    }
    else
    {
        trace("Error message:", event.error.message);
        trace("Details:", event.error.details);
    }
}
```

以下範例說明如何在同步執行模式中開啟加密資料庫。為求簡化，此範例中的加密金鑰在應用程式碼中使用硬式編碼。不過，我們非常不建議您使用這項技巧，因為很不安全。

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();
var dbFile:File = File.applicationStorageDirectory.resolvePath("DBSample.db");

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

try
{
    conn.open(dbFile, SQLMode.UPDATE, false, 1024, encryptionKey);
    trace("the database was created successfully");
}
catch (error:SQLError)
{
    if (error.errorID == 3138)
    {
        trace("Incorrect encryption key");
    }
    else
    {
        trace("Error message:", error.message);
        trace("Details:", error.details);
    }
}
```

如需一個以建議方式建立加密金鑰的範例說明，請參閱第 657 頁「範例：產生並使用加密金鑰」。

變更資料庫的加密金鑰

Adobe AIR 1.5 以及更新的版本

當資料庫受到加密時，您可以在稍後變更其加密金鑰。若要變更資料庫的加密金鑰，請先建立一個 SQLConnection 實體並呼叫其 open() 方法或 openAsync() 方法，藉此開啟資料庫的連線。連線至資料庫以後，請呼叫 reencrypt() 方法，將新的加密金鑰做為引數來傳遞。

就像大多數的資料庫作業一樣，reencrypt() 方法的行為會根據資料庫連線是使用同步或非同步執行模式而有所不同。如果您使用 open() 方法來連線資料庫，reencrypt() 作業就會同步執行，而此作業完成後就會繼續執行下一行程式碼：

```
var newKey:ByteArray = new ByteArray();
// ... generate the new key and store it in newKey
conn.reencrypt(newKey);
```

另一方面，如果對於資料庫的連線是以 openAsync() 方法來開啟，則 reencrypt() 作業就會非同步執行。呼叫 reencrypt() 就會開始重新加密的程序。該作業完成時，SQLConnection 物件就會傳送 reencrypt 事件。您可以使用事件偵聽程式來判斷重新加密程序何時結束：

```
var newKey:ByteArray = new ByteArray();
// ... generate the new key and store it in newKey

conn.addEventListener(SQLEvent.REENCRYPT, reencryptHandler);

conn.reencrypt(newKey);

function reencryptHandler(event:SQLEvent):void
{
    // save the fact that the key changed
}
```

reencrypt() 作業會執行其本身的交易工作。如果作業中斷或失敗 (比如說應用程式在作業完成前關閉)，交易就會回復。在這種情況下，原始的加密金鑰仍是資料庫的加密金鑰。

reencrypt() 方法不能用來為資料庫移除加密。如果對 reencrypt() 方法傳遞一個 null 值或是一個非 16 位元組 ByteArray 的加密金鑰，就會發生錯誤。

為資料庫加密時的考量事項

Adobe AIR 1.5 以及更新的版本

第 652 頁 「加密資料庫的用法」一節提出了幾種可使用加密資料庫的情形。不同應用程式的使用情形 (包括這裡提及的以及其它的使用情形) 顯然都會有不同的隱私需求。對於應用程式的加密，您的架構方式對於資料庫隱私程度的控制扮演著相當重要的角色。例如，假設您使用加密資料庫來保持個人資料的隱密性，而且甚至不讓相同電腦上的其他使用者取用該資料，那麼，每一個使用者的資料庫就都需要各自的加密金鑰。為了獲得最高的安全性，您的應用程式可以從使用者輸入的密碼中產生金鑰。若以密碼做為加密金鑰的基礎可以確保即使有人在電腦上假冒使用者的帳戶，仍然不能存取資料。另一種隱密性的考量是，您可能希望資料庫檔案能讓應用程式自己的所有使用者讀取，但不開放給其它應用程式讀取。在這種情況下，每一份安裝的應用程式都必須能存取共用的加密金鑰。

您可以根據自己希望資料擁有的隱密程度來設計應用程式，尤其是該用何種技術來產生加密金鑰。以下根據各種資料隱密程度來列出設計上的建議：

- 若要讓所有能在任何電腦上存取應用程式的使用者存取資料庫，請使用單一金鑰以供應用程式的所有實體使用。例如，當某個應用程式初次執行時，便可以使用 SSL 之類的安全性通訊協定來從伺服器下載共用的加密金鑰，然後就可以將金鑰儲存在加密本機儲存區中，以供日後使用。或者，也可以為電腦上各個使用者的資料進行加密，然後將該資料與某個遠端資料儲存區 (例如伺服器) 同步化，讓資料具有可攜性。
- 若要讓資料庫供任何電腦上的單一使用者存取，請從使用者的機密資料 (例如密碼) 中產生加密金鑰。請特別注意，切勿使用任何與特定電腦有關的值 (例如某個位於加密本機儲存區的值) 來產生金鑰。或者，也可以為電腦上各個使用者的資料進行加密，然後將該資料與某個遠端資料儲存區 (例如伺服器) 同步化，讓資料具有可攜性。
- 若要讓資料庫供單一電腦上的單一使用者存取，請從密碼和產生的 Salt 中產生金鑰。如需此項技術的範例，請參閱第 657 頁 「範例：產生並使用加密金鑰」。

以下是其它的重要安全考量，當您在設計應用程式使用加密資料庫時應將其謹記在心：

- 一個系統的安全程度能有多高，完全在於其最弱環節有多脆弱。如果您透過使用者輸入的密碼來產生加密金鑰，請考慮對密碼的長度和複雜度做出限制。如果是只使用基本字元的短密碼，可能很快就會被猜中。
- AIR 應用程式的來源程式碼會以純文字 (適用於 HTML 內容) 或某種可易於解編的二進位格式 (適用於 SWF 內容) 儲存在使用者的電腦上。因為來源程式碼可供人存取，所以有兩點需要注意：
  - 千萬不要在來源程式碼中對加密金鑰進行硬式編碼
  - 永遠假設您用來產生加密金鑰的技術 (例如隨機字元產生器或特定的雜湊演算法) 都會輕易遭到攻擊者破解
- AIR 資料庫加密會對 CBC-MAC (CCM) 模式的計數器使用「進階加密標準」(AES)。這個加密密碼需要一組由使用者輸入的金鑰來與 Salt 值結合，以達到安全效果。如需此項技術的範例，請參閱第 657 頁 「範例：產生並使用加密金鑰」。
- 當您選擇加密資料庫時，搭配該資料庫的資料庫引擎所使用的所有磁碟檔案都會加密。不過，資料庫引擎會先將某些資料暫存在記憶體內部快取中，藉以改善交易作業的讀取時間效能和寫入時間效能。所有記憶體駐留資料都未受加密。如果某個攻擊者能夠存取 AIR 應用程式所使用的記憶體 (例如透過除錯程式)，則資料庫中目前開啟且未受加密的資料可能就會遭到取用。

範例：產生並使用加密金鑰

Adobe AIR 1.5 以及更新的版本

這個範例應用程式示範了一種能產生加密金鑰的技術，其設計目的就是為使用者的資料提供最高程度的隱密性和安全性。確保私有資料的重要面向之一，就是要求使用者在應用程式每次連線至資料庫時輸入密碼。因此，如這個範例所示，要求這種隱私權層級的應用程式絕對不能直接儲存資料庫加密金鑰。

應用程式是由兩個部分組成：一是產生加密金鑰的 ActionScript 類別 (EncryptionKeyGenerator 類別)，二是示範如何使用該類別的基本使用者介面。如需完整的原始碼，請參閱第 658 頁 「完整範例程式碼：產生並使用加密金鑰」。

使用 EncryptionKeyGenerator 類別取得安全加密金鑰

Adobe AIR 1.5 以及更新的版本

您不需要瞭解 EncryptionKeyGenerator 類別的內部作業方式，就可以在您的應用程式中使用它。若您有興趣瞭解類別產生資料庫加密金鑰的詳細方式，請參閱第 662 頁 「瞭解 EncryptionKeyGenerator 類別」。

請依照這些步驟在應用程式中使用 EncryptionKeyGenerator 類別：

1. 請下載 EncryptionKeyGenerator 類別做為原始碼或編譯過的 SWC。EncryptionKeyGenerator 類別已包含在開放原始碼 ActionScript 3.0 核心元件庫 (as3corelib) 專案中。您可以下載 as3corelib 套件 (包括原始碼和文件)，也可以從專案頁面下載 SWC 或原始碼檔案。
2. 將 EncryptionKeyGenerator 類別的原始碼 (或 as3corelib SWC) 放置在應用程式原始碼可以找到的位置。
3. 在應用程式原始碼中，加入 EncryptionKeyGenerator 類別的 import 陳述式。

   ```
   import com.adobe.air.crypto.EncryptionKeyGenerator;
   ```

4. 在程式碼建立資料庫或開啟與其連線之前，呼叫 EncryptionKeyGenerator() 建構函式來加入建立 EncryptionKeyGenerator 實體的程式碼。

   ```
   var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
   ```

5. 從使用者取得密碼：

   ```
   var password:String = passwordInput.text;

   if (!keyGenerator.validateStrongPassword(password))
   {
       // display an error message
       return;
   }
   ```

   EncryptionKeyGenerator 實體會使用此密碼做為加密金鑰的基礎 (如下一步驟所示)。EncryptionKeyGenerator 實體會針對特定的強式密碼驗證需求來測試密碼。如果驗證失敗，就會發生錯誤。如範例程式碼所示，您可以呼叫 EncryptionKeyGenerator 物件的 validateStrongPassword() 方法，預先檢查密碼。如此一來，您就可以判斷密碼是否符合強式密碼的最低需求，以避免發生錯誤。

6. 從密碼產生加密金鑰：

   ```
   var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);
   ```

   getEncryptionKey() 方法會產生和傳回加密金鑰 (16 位元組 ByteArray)。之後，您可以使用加密金鑰來建立新的加密資料庫或開啟現有的資料庫。

   getEncryptionKey() 方法具有一個必要的參數，也就是在步驟 5 中取得的密碼。

   備註：若要維護最高層級的資料安全性和隱私權，應用程式必須要求使用者在應用程式每次連線至資料庫時輸入密碼。請勿直接儲存使用者的密碼或資料庫加密金鑰。這麼做可能會造成安全性方面的風險。相反地，如本範例中所示，在建立加密資料庫和稍後與其連線時，應用程式都會使用相同的技巧從密碼衍生加密金鑰。

getEncryptionKey() 方法也可接受第二個 (選擇性) 參數，也就是 overrideSaltELSKey 參數。EncryptionKeyGenerator 會建立用來做為加密金鑰一部分的隨機值 (稱為 salt)。為了能夠重新建立加密金鑰，salt 值會儲存在 AIR 應用程式的加密本機儲存區 (ELS) 中。根據預設，EncryptionKeyGenerator 類別會使用特定的 String 做為 ELS 金鑰。雖然可能性不大，但此金鑰仍有可能會與應用程式所使用的其它金鑰衝突。與其使用預設金鑰，您或許會考慮自行指定 ELS 金鑰。在這種情況下，請指定自訂金鑰，方法是將自訂金鑰當做第二個 getEncryptionKey() 參數來傳遞，如下面所示：

```
var customKey:String = "My custom ELS salt key";
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password, customKey);
```

7 建立或開啟資料庫

您的程式碼可以透過 getEncryptionKey() 方法所傳回的加密金鑰來建立新的加密資料庫，或嘗試開啟現有的加密資料庫。這兩種情形會用到 SQLConnection 類別的 open() 或 openAsync() 方法，第 653 頁 「建立加密資料庫」和第 653 頁 「連線至加密資料庫」中包含了相關的說明。

在這個範例中，應用程式的設計是要以非同步執行模式開啟資料庫。程式碼會設定適當的事件偵聽程式並呼叫 openAsync() 方法，將加密金鑰做為最後的引數傳遞：

```
conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, openError);

conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
```

在偵聽程式方法中，程式碼會移除事件偵聽程式的註冊，然後顯示一則狀態訊息來指出資料庫是否已經建立、開啟或發生錯誤。這些事件處理常式最值得注意的部分，是在 openError() 方法中。在該方法中，if 陳述式會檢查資料庫是否存在 (表示程式碼會嘗試連線到現有資料庫)，以及錯誤 ID 是否符合常數 EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID。如果這兩個條件同時成立，就可能代表使用者輸入的密碼有誤 (也可能代表指定的檔案並不是資料庫檔案)。下列為檢查錯誤 ID 的程式碼：

```
if (!createNewDB && event.error.errorID == EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
{
    statusMsg.text = "Incorrect password!";
}
else
{
    statusMsg.text = "Error creating or opening database.";
}
```

如需範例事件偵聽程式的完整程式碼，請參閱第 658 頁 「完整範例程式碼：產生並使用加密金鑰」。

完整範例程式碼：產生並使用加密金鑰

Adobe AIR 1.5 以及更新的版本

以下為範例應用程式 「產生並使用加密金鑰」的完整程式碼。此程式碼由兩部分所組成。

此範例會使用 EncryptionKeyGenerator 類別，從密碼建立加密金鑰。EncryptionKeyGenerator 類別已包含在開放原始碼 ActionScript 3.0 核心元件庫 (as3corelib) 專案中。您可以下載 as3corelib 套件 (包括原始碼和文件)，也可以從專案頁面下載 SWC 或原始碼檔案。

Flex 範例

應用程式 MXML 檔案包含了簡易應用程式的原始碼，該應用程式會建立或開啟與加密資料庫的連線：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="init();">
    <mx:Script>
        <![CDATA[
            import com.adobe.air.crypto.EncryptionKeyGenerator;

            private const dbFileName:String = "encryptedDatabase.db";

            private var dbFile:File;
            private var createNewDB:Boolean = true;
            private var conn:SQLConnection;

            // ------- Event handling -------

            private function init():void
            {
                conn = new SQLConnection();
                dbFile = File.applicationStorageDirectory.resolvePath(dbFileName);
                if (dbFile.exists)
                {
                    createNewDB = false;
                    instructions.text = "Enter your database password to open the encrypted database.";
                    openButton.label = "Open Database";
                }
            }

            private function openConnection():void
            {
                var password:String = passwordInput.text;

                var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();

                if (password == null || password.length <= 0)
                {
                    statusMsg.text = "Please specify a password.";
                    return;
                }

                if (!keyGenerator.validateStrongPassword(password))
                {
                    statusMsg.text = "The password must be 8-32 characters long. It must contain at least
one lowercase letter, at least one uppercase letter, and at least one number or symbol.";
                    return;
                }

                passwordInput.text = "";
                passwordInput.enabled = false;
                openButton.enabled = false;

                var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, openError);

                conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
            }

            private function openHandler(event:SQLEvent):void
            {
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                conn.removeEventListener(SQLErrorEvent.ERROR, openError);
```

```
                statusMsg.setStyle("color", 0x009900);
                if (createNewDB)
                {
                    statusMsg.text = "The encrypted database was created successfully.";
                }
                else
                {
                    statusMsg.text = "The encrypted database was opened successfully.";
                }
            }

            private function openError(event:SQLErrorEvent):void
            {
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                conn.removeEventListener(SQLErrorEvent.ERROR, openError);

                if (!createNewDB && event.error.errorID ==
EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
                {
                    statusMsg.text = "Incorrect password!";
                }
                else
                {
                    statusMsg.text = "Error creating or opening database.";
                }
            }
        ]]>
    </mx:Script>
    <mx:Text id="instructions" text="Enter a password to create an encrypted database. The next time you
open the application, you will need to re-enter the password to open the database again." width="75%"
height="65"/>
    <mx:HBox>
        <mx:TextInput id="passwordInput" displayAsPassword="true"/>
        <mx:Button id="openButton" label="Create Database" click="openConnection();"/>
    </mx:HBox>
    <mx:Text id="statusMsg" color="#990000" width="75%"/>
</mx:WindowedApplication>
```

Flash Professional 範例

應用程式 FLA 檔案包含簡易應用程式的原始碼，該應用程式會建立或開啟與加密資料庫的連線。FLA 檔具有四個放置於舞台的組件：

實體名稱	組件類型	說明
instructions	Label	包含提供給使用者的指示
passwordInput	TextInput	使用者在當中輸入密碼的輸入欄位
openButton	Button	使用者在輸入密碼後所按的按鈕
statusMsg	Label	顯示狀態 (成功或失敗) 訊息

應用程式的程式碼會定義在主時間軸中影格 1 的關鍵影格上。以下為應用程式的程式碼：

```
import com.adobe.air.crypto.EncryptionKeyGenerator;

const dbFileName:String = "encryptedDatabase.db";

var dbFile:File;
var createNewDB:Boolean = true;
var conn:SQLConnection;

init();

// ------- Event handling -------

function init():void
{
    passwordInput.displayAsPassword = true;
    openButton.addEventListener(MouseEvent.CLICK, openConnection);
    statusMsg.setStyle("textFormat", new TextFormat(null, null, 0x990000));

    conn = new SQLConnection();
    dbFile = File.applicationStorageDirectory.resolvePath(dbFileName);

    if (dbFile.exists)
    {
        createNewDB = false;
        instructions.text = "Enter your database password to open the encrypted database.";
        openButton.label = "Open Database";
    }
    else
    {
        instructions.text = "Enter a password to create an encrypted database. The next time you open the
application, you will need to re-enter the password to open the database again.";
        openButton.label = "Create Database";
    }
}

function openConnection(event:MouseEvent):void
{
    var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();

    var password:String = passwordInput.text;

    if (password == null || password.length <= 0)
    {
        statusMsg.text = "Please specify a password.";
        return;
    }

    if (!keyGenerator.validateStrongPassword(password))
    {
        statusMsg.text = "The password must be 8-32 characters long. It must contain at least one lowercase
letter, at least one uppercase letter, and at least one number or symbol.";
        return;
    }

    passwordInput.text = "";
    passwordInput.enabled = false;
    openButton.enabled = false;

    var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);

    conn.addEventListener(SQLEvent.OPEN, openHandler);
    conn.addEventListener(SQLErrorEvent.ERROR, openError);
```

```
    conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
}

function openHandler(event:SQLEvent):void
{
    conn.removeEventListener(SQLEvent.OPEN, openHandler);
    conn.removeEventListener(SQLErrorEvent.ERROR, openError);

    statusMsg.setStyle("textFormat", new TextFormat(null, null, 0x009900));
    if (createNewDB)
    {
        statusMsg.text = "The encrypted database was created successfully.";
    }
    else
    {
        statusMsg.text = "The encrypted database was opened successfully.";
    }
}

function openError(event:SQLErrorEvent):void
{
    conn.removeEventListener(SQLEvent.OPEN, openHandler);
    conn.removeEventListener(SQLErrorEvent.ERROR, openError);

    if (!createNewDB && event.error.errorID == EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
    {
        statusMsg.text = "Incorrect password!";
    }
    else
    {
        statusMsg.text = "Error creating or opening database.";
    }
}
```

瞭解 EncryptionKeyGenerator 類別

Adobe AIR 1.5 以及更新的版本

您並不需要瞭解 EncryptionKeyGenerator 類別的內部作業方式，就可以用它來建立應用程式資料庫的安全加密金鑰。使用此類別的程序會在第 657 頁「使用 EncryptionKeyGenerator 類別取得安全加密金鑰」中說明。不過，您可能會覺得瞭解此類別所使用的技巧很有用。舉例來說，您或許想要針對需要不同資料隱私權層級的情況，調整此類別或納入其中一些技巧。

EncryptionKeyGenerator 類別已包含在開放原始碼 ActionScript 3.0 核心元件庫 (as3corelib) 專案中。您可以下載 as3corelib 套件 (包括原始碼和文件)，也可以檢視專案網站上的原始碼，或者加以下載，以遵照說明操作。

當程式碼建立 EncryptionKeyGenerator 實體並呼叫其 getEncryptionKey() 方法時，會採取數個步驟來確保只有具有正確權限的使用者才可以存取資料。此程序與資料庫建立之前從使用者輸入的密碼產生加密金鑰是相同的，也與重新建立加密金鑰以開啟資料庫的程序相同。

取得並驗證增強式密碼

Adobe AIR 1.5 以及更新的版本

當程式碼呼叫 getEncryptionKey() 方法時，該方法會傳遞密碼做為參數。並該密碼將做為加密金鑰的基礎。因為這項設計使用的是只有使用者才知道的資訊，所以可以確保只有知道密碼的使用者可以存取資料庫裡的資料。就算某個攻擊者存取了使用者電腦上的帳戶，只要不知道密碼就無法進入資料庫。為了獲得最高的安全性，應用程式永遠都不儲存密碼。

應用程式的程式碼會建立 EncryptionKeyGenerator 實體，並呼叫其 getEncryptionKey() 方法，將使用者輸入的密碼當作引數傳遞 (也就是此範例中的 password 變數)：

```
var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);
```

呼叫 getEncryptionKey() 方法時，EncryptionKeyGenerator 類別所採取的第一個步驟是檢查使用者輸入的密碼，以確定其符合密碼強度需求。EncryptionKeyGenerator 類別的密碼必須有 8 - 32 位元的長度。密碼必須混合大小寫字母，且至少有一個數字或符號字元。

負責檢查此樣式的規則運算式會被定義為一個常數，名為 STRONG_PASSWORD_PATTERN：

```
private static const STRONG_PASSWORD_PATTERN:RegExp = /(?=^.{8,32}$)((?=.*\d)|(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$/;
```

檢查密碼的程式碼位於 EncryptionKeyGenerator 類別的 validateStrongPassword() 方法中。此程式碼如下：

```
public function vaidateStrongPassword(password:String):Boolean
{
    if (password == null || password.length <= 0)
    {
        return false;
    }

    return STRONG_PASSWORD_PATTERN.test(password))
}
```

getEncryptionKey() 方法的內部會呼叫 EncryptionKeyGenerator 類別的 validateStrongPassword() 方法，若密碼無效，就會擲出例外。validateStrongPassword() 方法是公用方法，如此一來，應用程式程式碼不必呼叫 getEncryptionKey() 方法便能檢查密碼，可避免造成錯誤。

將密碼擴充為 256 位元

Adobe AIR 1.5 以及更新的版本

此程序稍後會要求密碼要具有 256 位元的長度。程式碼並不會要求各個使用者輸入一個長度剛好 256 位元 (32 個字元) 的密碼，而是藉由重複密碼的字元來建立一個比較長的密碼。

getEncryptionKey() 方法會呼叫 concatenatePassword() 方法來執行建立長密碼的工作。

```
var concatenatedPassword:String = concatenatePassword(password);
```

以下為 concatenatePassword() 方法的程式碼：

```
private function concatenatePassword(pwd:String):String
{
    var len:int = pwd.length;
    var targetLength:int = 32;

    if (len == targetLength)
    {
        return pwd;
    }

    var repetitions:int = Math.floor(targetLength / len);
    var excess:int = targetLength % len;

    var result:String = "";

    for (var i:uint = 0; i < repetitions; i++)
    {
        result += pwd;
    }

    result += pwd.substr(0, excess);

    return result;
}
```

如果密碼少於 256 位元，則程式碼會將密碼自行連接，使其成為 256 位元。如果最後的長度沒有剛好，則最後一次的重複字串會稍微減短，使長度剛好為 256 位元。

產生或擷取 256 位元的 Salt 值

Adobe AIR 1.5 以及更新的版本

下一個步驟就是取得一個 256 位元的 Salt 值，這個值會在稍後的步驟中與密碼結合。salt 是一個隨機值，將附加至使用者輸入的值，或與該值結合，組成一個密碼。對密碼使用 Salt 的做法可以確保就算使用者選擇以文字或一般字詞來做為密碼，系統所使用的 「密碼加 Salt」結合結果仍會是一個隨機值。這樣的隨機性可以協助防禦字典攻擊法，也就是指攻擊者使用一系列的文字來試圖猜出密碼。此外，如果在產生 Salt 值後將其儲存於加密本機儲存區，該值就會在資料庫檔案的所在電腦上與使用者的帳戶產生連結。

如果應用程式是第一次呼叫 getEncryptionKey() 方法，程式碼就會建立隨機的 256 位元 Salt 值，否則，就會從加密本機儲存區中載入 Salt 值。

Salt 將儲存在名為 salt 的變數中。程式碼會嘗試從加密本機儲存區載入 Salt，判斷是否已經建立該值：

```
var salt:ByteArray = EncryptedLocalStore.getItem(saltKey);
if (salt == null)
{
    salt = makeSalt();
    EncryptedLocalStore.setItem(saltKey, salt);
}
```

如果程式碼要建立新的 Salt 值，makeSalt() 方法就會產生 256 位元隨機值。因為這個值最後會儲存在加密本機儲存區中，所以會以 ByteArray 物件的形式產生。這個值會由 makeSalt() 方法使用 Math.random() 方法來隨機產生。Math.random() 方法並不能一次就產生 256 位元，程式碼便改為使用迴圈來呼叫 Math.random() 八次，而每一次都會產生一個隨機的 uint 值，介於 0 和 4294967295 (最大的 uint 值) 之間。使用 uint 值是為了方便，因為 uint 剛好使用 32 位元。只要將八個 uint 值寫入 ByteArray，就能夠產生一個 256 位元的值。以下為 makeSalt() 方法的程式碼：

```
private function makeSalt():ByteArray
{
    var result:ByteArray = new ByteArray;

    for (var i:uint = 0; i < 8; i++)
    {
        result.writeUnsignedInt(Math.round(Math.random() * uint.MAX_VALUE));
    }

    return result;
}
```

當程式碼將 Salt 儲存至加密本機儲存區 (ELS) 或從 ELS 擷取 Salt 時，需要 Salt 下方所儲存的 String 金鑰。若不知道此金鑰，就無法擷取 Salt 值。在這種狀況下，就無法在每次重新開啟資料庫時重新建立加密金鑰。根據預設，EncryptionKeyGenerator 會使用預先定義的 ELS 金鑰，此金鑰會定義於 SALT_ELS_KEY 常數中。除了使用預設金鑰外，應用程式程式碼也可以指定 ELS 金鑰，以用於針對 getEncryptionKey() 方法進行的呼叫。預設金鑰及應用程式指定的 Salt ELS 金鑰都會儲存在名為 saltKey 的變數中。如上所示，對 EncryptedLocalStore.setItem() 和 EncryptedLocalStore.getItem() 的呼叫會使用該變數。

以 XOR 運算子結合 256 位元密碼和 Salt

Adobe AIR 1.5 以及更新的版本

程式碼現在已經有一個 256 位元的密碼與一個 256 位元的 Salt 值。它會使用位元 XOR 運算，將 Salt 和連接的密碼組合成單一的值。事實上，此項技術會從整個可能的字元範圍，建立由字元所組成的 256 位元密碼。即使實際的密碼輸入主要是由英數字元組成，此項原則也不變。隨機性增加的優點是使用者不需要輸入一長串的複雜密碼，便能建立較大的可能密碼組。

XOR 運算的結果會儲存在 unhashedKey 變數中。這兩個值的 XOR 位元運算程序會在 xorBytes() 方法中實現：

```
var unhashedKey:ByteArray = xorBytes(concatenatedPassword, salt);
```

位元 XOR 運算子 (^) 會接受兩個 uint 值，然後傳回一個 uint 值 (uint 值包含 32 位元)。做為引數傳遞至 xorBytes() 方法的輸入值包括 String (密碼) 和 ByteArray (Salt)。之後，程式碼會開始使用迴圈，一次從一筆輸入中擷取 32 位元，進而使用 XOR 運算子來加以連結。

```
private function xorBytes(passwordString:String, salt:ByteArray):ByteArray
{
    var result:ByteArray = new ByteArray();

    for (var i:uint = 0; i < 32; i += 4)
    {
        // ...
    }

    return result;
}
```

在此迴圈中，會從 passwordString 參數中擷取前 32 個位元 (4 位元組)，然後，將執行一個程序來將擷取這些位元，並將其轉換為 uint (o1)：這個程序包含了兩個部分。首先，charCodeAt() 方法會取得各個字元的數值。接著，以位元左移運算子 (<<) 將該值移到 uint 中的適當位置，然後將移位後的值附加至 o1。舉例來說，當使用位元左移運算子 (<<) 往左移 24 位元後再將該值附加給 o1 時，第一個字元 (i) 就會變成第一組的 8 位元。當值往左移 16 位元後將其附加至 o1，第二個字元 (i + 1) 就會變成第二組的 8 位元。第三和第四個字元的值也以相同方式附加。

```
        // ...

        // Extract 4 bytes from the password string and convert to a uint
        var o1:uint = passwordString.charCodeAt(i) << 24;
        o1 += passwordString.charCodeAt(i + 1) << 16;
        o1 += passwordString.charCodeAt(i + 2) << 8;
        o1 += passwordString.charCodeAt(i + 3);

        // ...
```

o1 變數現在包含來自 passwordString 參數的 32 位元。接著，呼叫 salt 參數的 readUnsignedInt() 方法，從參數中擷取出 32 位元。32 位元會儲存在 o2 這個 uint 變數中。

```
        // ...

        salt.position = i;
        var o2:uint = salt.readUnsignedInt();

        // ...
```

最後，這兩個 32 位元 (uint) 值會以 XOR 運算子結合，而結果將寫入一個名為 result 的 ByteArray 中。

```
        // ...

        var xor:uint = o1 ^ o2;
        result.writeUnsignedInt(xor);
        // ...
```

迴圈完成後，便將內含此 XOR 運算結果的 ByteArray 傳回。

```
        // ...
    }

    return result;
}
```

雜湊金鑰

Adobe AIR 1.5 以及更新的版本

一旦將連接的密碼和 Salt 結合後，下一步驟就是使用 SHA-256 雜湊演算法對其進行雜湊，以進一步確保此值的安全性。對值進行雜湊可以使攻擊者更難對其施以反向工程。

程式碼現在有一個名為 unhashedKey 的 ByteArray，其中包含與 Salt 結合的連接密碼。ActionScript 3.0 核心元件庫 (as3corelib) 專案在 com.adobe.crypto 套件中包含了 SHA256 類別。SHA256.hashBytes() 方法可對 ByteArray 執行 SHA-256 雜湊，然後傳回一個 String，其中含有一個十六進位數字形式的 256 位元雜湊結果。EncryptionKeyGenerator 類別會使用 SHA256 類別來雜湊金鑰：

```
var hashedKey:String = SHA256.hashBytes(unhashedKey);
```

從雜湊中擷取加密金鑰

Adobe AIR 1.5 以及更新的版本

加密金鑰必須是一個長度剛好為 16 位元組 (128 位元) 的 ByteArray。SHA-256 雜湊演算法的結果永遠都具有 256 位元的長度。接著，就是要從雜湊的結果中選出 128 位元，將其做為真正的加密金鑰。

在 EncryptionKeyGenerator 類別中，程式碼會呼叫 generateEncryptionKey() 方法，讓金鑰減為 128 位元。然後，再將該方法的結果以 getEncryptionKey() 方法的結果形式傳回：

```
var encryptionKey:ByteArray = generateEncryptionKey(hashedKey);
return encryptionKey;
```

並非一定要使用前 128 個位元來做為加密金鑰。您可以選取以任一起點開始的位元範圍、每隔一個位元進行選取，或者使用其它方法來選取位元。重點在於，程式碼會明確地選出 128 個位元，然後每次都使用相同的 128 個位元。

在此範例中，generateEncryptionKey() 方法會從第 18 個位元組開始取出這個範圍的位元數，將其做為加密金鑰。如前所述，SHA256 類別會傳回一個 String，其中含有一個十六進位數字形式的 256 位元雜湊結果。一個區塊便有 128 位元，因為有太多位元組，所以無法一次新增至 ByteArray。因此，程式碼使用 for 迴圈來從這個十六進位的 String 中擷取字元，然後再將那些字元轉換為真正的數值，再將數值新增至 ByteArray。SHA-256 結果 String 的長度為 64 個字元。128 位元的範圍等於 String 中的 32 個字元，而每個字元代表 4 個位元。可讓您新增至 ByteArray 的最小增量是一個位元組 (8 位元)，等於 hash String 中的兩個字元。因此，迴圈的計數將從 0 至 31 (32 個字元)，每次的增量為 2 個字元。

在迴圈中，程式碼會先找出當前這一對字元的開始位置。因為想挑出的範圍開始於索引位置 17 (也就是第 18 個位元組) 的字元，所以 position 變數的設定就是目前的迴圈指標值 (i) 加 17。程式碼會使用 String 物件的 substr() 方法來從目前的位置擷取出兩個字元。那些字元會儲存在 hex 變數中。然後，程式碼會使用 parseInt() 方法將 hex String 轉換為十進位整數，然後將該值儲存在 byte 這個 int 變數中。最後，程式碼會使用 writeByte() 方法將 byte 中的值加到 result 這個 ByteArray 中。迴圈完成後，result ByteArray 會含有 16 個位元組，如此便已就緒，且可做為資料庫加密金鑰。

```
private function generateEncryptionKey(hash:String):ByteArray
{
    var result:ByteArray = new ByteArray();

    for (var i:uint = 0; i < 32; i += 2)
    {
        var position:uint = i + 17;
        var hex:String = hash.substr(position, 2);
        var byte:int = parseInt(hex, 16);
        result.writeByte(byte);
    }

    return result;
}
```

使用 SQL 資料庫的策略

Adobe AIR 1.0 以及更新的版本

應用程式存取及使用本機 SQL 資料庫的方式有許多種。應用程式設計可能會依應用程式的程式碼組織方式、執行作業的順序和時間等因素而不同。您所選擇的技巧可能對應用程式開發的簡易程度造成影響，也有機會影響在未來更新時，修改應用程式的難易度。此外，這項決定也可能影響到使用者察覺到的應用程式執行順暢度。

發佈預先填入的資料庫

Adobe AIR 1.0 以及更新的版本

當您在應用程式中使用 AIR 本機 SQL 資料庫時，應用程式會預期資料庫應具備一定的資料表、欄等結構。有些應用程式也會預期資料庫檔案中已預先填入資料。確保資料庫有適當結構的其中一個方法，就是在應用程式的程式碼中建立資料庫。當應用程式載入資料庫時，會在特定位置中檢查其資料庫檔案是否存在。如果檔案不存在，應用程式會執行一組命令，以建立資料庫檔案、建立資料庫結構，並用初始資料填入資料表中。

建立資料庫及其資料表的程式碼往往很複雜。這種程式碼通常只會在應用程式安裝週期中使用一次，但仍然會增加應用程式的大小和複雜程度。您可以將預先填入的資料庫與應用程式一起發佈，將此做為透過程式設計方式建立資料庫、結構和資料的替代方式。若要發佈預先定義的資料庫，請在應用程式的 AIR 套件中包含資料庫檔案。

與包含在 AIR 套件中的所有檔案一樣，合併的資料庫檔案會安裝在應用程式目錄中 (以 File.applicationDirectory 屬性代表的目錄)。但是，該目錄中的檔案是唯讀的。請使用來自 AIR 套件的檔案做為 「範本」資料庫。使用者第一次執行應用程式時，會將原始的資料庫檔案複製至使用者的第 578 頁 「指向應用程式儲存目錄」(或其它位置) 中，並在應用程式中使用該資料庫。

使用本機 SQL 資料庫的最佳做法

Adobe AIR 1.0 以及更新的版本

下列清單是一組建議使用的技巧，可在使用本機 SQL 資料庫時，供您用來提升應用程式的效能、安全性，並讓應用程式易於維護。

預先建立的資料庫連線

Adobe AIR 1.0 以及更新的版本

即使您的應用程式在初次載入時不執行任何陳述式，事先將 SQLConnection 物件實體化並呼叫其 open() 或 openAsync() 方法 (例如在初始應用程式啟動之後)，可避免在執行陳述式時發生延遲。請參閱第 625 頁 「連線到資料庫」。

重複使用資料庫連線

Adobe AIR 1.0 以及更新的版本

如果您在應用程式執行時間一直都要存取某個特定資料庫，請保留 SQLConnection 實體的參考，並在整個應用程式執行期間重複使用該參考，而不要不斷關閉又重新開啟連線。請參閱第 625 頁 「連線到資料庫」。

優先使用非同步執行模式

Adobe AIR 1.0 以及更新的版本

撰寫資料存取程式碼時，往往會因為使用同步作業經常只需要較短而較不複雜的程式碼，而想要以同步方式 (而不是以非同步方式) 執行作業。但是，如同第 648 頁 「使用同步和非同步資料庫作業」一節中所述，同步作業可能會有很大的效能衝擊，這一點不僅可讓使用者很明顯地感受到，而且會對他們使用應用程式的體驗造成負面影響。單一作業所耗費的時間需視作業而定，尤其是其中所包含的資料量。例如，只加入單一列至資料庫的 SQL INSERT 陳述式，所耗費的時間就不如擷取數千個資料列的 SELECT 陳述式那麼多。但是，當您使用同步執行來執行多項作業時，作業通常會串聯在一起。即使每項單一作業都很短，應用程式也會凍結到所有同步作業完成為止。結果，多項作業串聯在一起所累積的時間可能會多到讓應用程式動彈不得。

請使用非同步作業做為標準手法，尤其是在包含大量資料列的作業時。有一項技巧可以讓您拆解大量 SELECT 陳述式結果集處理，請參閱第 639 頁 「擷取部分 SELECT 結果」一節的說明。不過，這項技巧只能用於非同步執行模式。請僅在無法使用非同步程式設計達成特定功能時、已考慮過應用程式的使用者將面臨的效能代價後，以及您已測試過應用程式並瞭解應用程式效能所受到的影響後，才使用同步作業。使用非同步執行可能會涉及更複雜的程式編寫。但是，請記住，您只需要撰寫程式碼一次，而應用程式的使用者不管速度快慢都必須重複不斷地使用該程式碼。

在許多情況下，若讓每個要執行的 SQL 陳述式都使用不同的 SQLStatement 實體，多項 SQL 作業可能會一次積累過多，而就程式碼的撰寫方式來說，這使得非同步程式碼就跟同步程式碼一樣。如需詳細資訊，請參閱第 651 頁 「瞭解非同步執行模式」。

使用不同的 SQL 陳述式而不變更 SQLStatement 的 text 屬性

Adobe AIR 1.0 以及更新的版本

對於在應用程式中不只執行一次的任何 SQL 陳述式，請為每個 SQL 陳述式建立不同的 SQLStatement 實體。每次執行該 SQL 命令時，就使用該 SQLStatement 實體。例如，假設您在建置包含四種不同 SQL 作業的應用程式，這些作業都要重複執行多次。在此情況下，請建立四個不同的 SQLStatement 實體，然後呼叫每個陳述式的 execute() 方法來執行該實體。請避免使用另外一種讓所有 SQL 陳述式使用單一 SQLStatement 實體的做法，在每次要執行陳述式之前重新定義其 text 屬性。

使用陳述式參數

Adobe AIR 1.0 以及更新的版本

請使用 SQLStatement 參數，絕對不要將使用者輸入連接到陳述式文字之中。使用參數可讓您的應用程式更加安全，因為這可以降低遭到 SQL 插入式攻擊的可能性，並讓您能夠在查詢中使用物件 (而不只是使用 SQL 常值)。此外，這個方法也能讓陳述式能夠更有效率地執行，因為陳述式可以重複使用而不必每次執行時都要重新進行編譯。如需詳細資訊，請參閱第 629 頁 「在陳述式中使用參數」。

第 41 章 使用位元組陣列

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ByteArray 類別讓您能夠讀取與寫入二進位資料串流，此串流基本上就是位元組陣列。這個類別提供最基礎的資料存取方式。由於電腦資料是由位元組構成 (每組 8 個位元)，能夠讀取以位元組為單位的資料；這意味著您可存取資料，且不需受限於類別和存取方法。ByteArray 類別可讓您剖析位元組層級的任何資料串流，包括點陣圖乃至經網路傳遞的資料串流。

writeObject() 方法讓您能將 Action Message Format (AMF) 序列化格式的物件寫入 ByteArray，readObject() 方法則可讓您從 ByteArray 將序列化物件讀入原始資料類型的變數中。任何物件皆可進行序列化，唯獨顯示物件 (可放置在顯示清單上的物件) 除外。只要自訂類別可供執行階段使用，您也能將序列化物件往回指定給自訂類別實體。當物件轉換成 AMF 之後，您就可以有效率地將其透過網路連線傳輸或儲存至檔案。

本章節所述的 Adobe® AIR® 應用程式樣本會讀取 .zip 檔，以示範如何處理位元組串流、擷取 .zip 檔所包含的檔案清單，再將結果輸出至桌面。

更多說明主題

flash.utils.ByteArray

flash.utils.IExternalizable

Action Message Format 規格

讀取與寫入 ByteArray

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ByteArray 類別是 flash.utils 套件的一份子。若要在 ActionScript 3.0 中建立 ByteArray 物件，請匯入 ByteArray 類別並叫用其建構函式，如下列範例所示：

```
import flash.utils.ByteArray;
var stream:ByteArray = new ByteArray();
```

ByteArray 方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

任何有意義的資料串流都會組織成可供您分析以找出所需資訊的特定格式。例如，一份簡單的員工資料檔可能含有身分證字號、姓名、地址和電話號碼等各項記錄。MP3 音效檔則含有 ID3 標籤，以識別曲名、作者、專輯、出版年份和下載檔案的歌曲類型。這些格式可讓您瞭解各項資料在資料串流中的預期順序，進而能夠明智地讀取位元組串流。

ByteArray 類別包含數種方法，使得資料串流的讀取與寫入更為容易。其中一些方法包括 readBytes() 和 writeBytes()、readInt() 和 writeInt()、readFloat() 和 writeFloat()、readObject() 和 writeObject()，以及 readUTFBytes() 和 writeUTFBytes()。這些方法讓您得以從資料串流將資料讀入特定資料類型的變數中，並可直接將特定資料類型寫入二進位資料串流。

例如，下列程式碼會讀取含有字串和浮點數的簡單陣列，並將每個元素寫入 ByteArray。陣列的組織架構讓程式碼能藉由呼叫適當的 ByteArray 方法 (writeUTFBytes() 和 writeFloat()) 以寫入資料。重複性的資料模式則會使程式碼得以利用迴圈讀取陣列。

```
 // The following example reads a simple Array (groceries), made up of strings
// and floating-point numbers, and writes it to a ByteArray.

import flash.utils.ByteArray;

// define the grocery list Array
var groceries:Array = ["milk", 4.50, "soup", 1.79, "eggs", 3.19, "bread" , 2.35]
// define the ByteArray
var bytes:ByteArray = new ByteArray();
// for each item in the array
for (var i:int = 0; i < groceries.length; i++) {
        bytes.writeUTFBytes(groceries[i++]); //write the string and position to the next item
        bytes.writeFloat(groceries[i]);// write the float
        trace("bytes.position is: " + bytes.position);//display the position in ByteArray
}
trace("bytes length is: " +  bytes.length);// display the length
```

position 屬性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

position 屬性儲存了在讀取或寫入期間，對 ByteArray 編列索引的指標目前位置。position 屬性的初始值為 0 (零)，如下列程式碼所示：

```
var bytes:ByteArray = new ByteArray();
trace("bytes.position is initially: " + bytes.position); // 0
```

當您讀取或寫入 ByteArray 時，所用的方法會更新 position 屬性，以指向最後讀取或寫入的位元組的下一個位置。例如，下列程式碼會將某字串寫入 ByteArray，之後，position 屬性即指向 ByteArray 中該字串的下一個位元組：

```
var bytes:ByteArray = new ByteArray();
trace("bytes.position is initially: " + bytes.position); // 0
bytes.writeUTFBytes("Hello World!");
trace("bytes.position is now: " + bytes.position);// 12
```

同樣地，讀取作業會以讀取的位元組數目來遞增 position 屬性。

```
 var bytes:ByteArray = new ByteArray();

trace("bytes.position is initially: " + bytes.position); // 0
bytes.writeUTFBytes("Hello World!");
trace("bytes.position is now: " + bytes.position);// 12
bytes.position = 0;
trace("The first 6 bytes are: " + (bytes.readUTFBytes(6)));//Hello
trace("And the next 6 bytes are: " + (bytes.readUTFBytes(6)));// World!
```

請注意，您可將 position 屬性設為 ByteArray 中的任何特定位置，以便讀取或寫入該偏移位置。

bytesAvailable 和 length 屬性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

length 屬性會指出 ByteArray 的長度，bytesAvailable 屬性則會指出從目前位置到陣列結尾的剩餘位元組數目。下列範例會說明如何使用這些屬性。此範例會先將文字字串寫入 ByteArray，接著從 ByteArray 一次讀取一個位元組，直到發現字元 "a" 或抵達陣列結尾 (bytesAvailable <= 0) 為止。

```
var bytes:ByteArray = new ByteArray();
var text:String = "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vivamus etc.";

bytes.writeUTFBytes(text); // write the text to the ByteArray
trace("The length of the ByteArray is: " + bytes.length);// 70
bytes.position = 0; // reset position
while (bytes.bytesAvailable > 0 && (bytes.readUTFBytes(1) != 'a')) {
    //read to letter a or end of bytes
}
if (bytes.position < bytes.bytesAvailable) {
    trace("Found the letter a; position is: " + bytes.position); // 23
    trace("and the number of bytes available is: " + bytes.bytesAvailable);// 47
}
```

endian 屬性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

多位元組數字 (亦即需要 1 個位元組以上的記憶體才夠儲存的數字) 儲存在電腦上的方式可能各有不同。例如，整數需要 4 個位元組的記憶體，也就是 32 個位元。某些電腦會將最高位元組數字儲存在前，置於最低記憶體位址空間，有些則是將最低位元組儲存在前。電腦 (或位元組定序) 的這項特質稱為 big endian (最高位元組在前) 或 little endian (最低位元組在前)。例如，數字 0x31323334 將儲存成如下所示的 big endian 和 little endian 位元組順序，其中 a0 代表最低記憶體位址空間的 4 個位元組，而 a3 則代表最高位址空間：

Big Endian	Big Endian	Big Endian	Big Endian
a0	a1	a2	a3
31	32	33	34

Little Endian	Little Endian	Little Endian	Little Endian
a0	a1	a2	a3
34	33	32	31

ByteArray 類別的 endian 屬性可讓您為目前正在處理的多位元組數字標註這種位元組順序。此屬性可接受的值為 "bigEndian" 或 "littleEndian"，而 Endian 類別更定義了 BIG_ENDIAN 和 LITTLE_ENDIAN 常數，便於您將 endian 屬性設為這些字串。

compress() 和 uncompress() 方法

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

compress() 方法可讓您指定壓縮演算法做為參數，根據此參數來壓縮 ByteArray。uncompress() 方法則可讓您指定壓縮演算法，根據此參數來將壓縮過的 ByteArray 進行解壓縮。呼叫 compress() 和 uncompress() 之後，位元組陣列的長度將設定為新長度，而 position 屬性會設定為陣列結尾。

CompressionAlgorithm 類別定義了可供您指定壓縮演算法的常數。ByteArray 類別支援 deflate (僅限 AIR)、zlib 和 lzma 演算法。zlib 壓縮資料格式於 http://www.ietf.org/rfc/rfc1950.txt 中說明。lzma 演算法是 Flash Player 11.4 和 AIR 3.4 新增的演算法，於 http://www.7-zip.org/7z.html 中說明。

許多壓縮格式都使用 deflate 壓縮演算法，如 zlib、gzip 及一些 zip 實作。deflate 壓縮演算法於 http://www.ietf.org/rfc/rfc1951.txt 中說明。

下列範例會使用 lzma 演算法來壓縮名為 bytes 的 ByteArray 陣列：

```
bytes.compress(CompressionAlgorithm.LZMA);
```

下列範例會使用 deflate 演算法，將壓縮過的 ByteArray 進行解壓縮：

```
bytes.uncompress(CompressionAlgorithm.LZMA);
```

讀取與寫入物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

readObject() 方法會從 ByteArray 讀取以 Action Message Format (AMF) 序列化格式進行編碼的物件，而 writeObject() 方法則可寫入此一編碼格式的物件。AMF 是由 Adobe 建立並已取得專利的訊息通訊協定，廣為各種 ActionScript 3.0 類別所採用，包括 Netstream、NetConnection、NetStream、LocalConnection 和共享物件。

單一位元組的類型標記描述了接續在其後的編碼資料類型。AMF 使用下列 13 種資料類型：

```
value-type = undefined-marker | null-marker | false-marker | true-marker | integer-type |
    double-type | string-type | xml-doc-type | date-type | array-type | object-type |
    xml-type | byte-array-type
```

編碼資料始終會接續在類型標記之後，除非標記代表單一可能值如 null、true 或 false，此時只有標記會進行編碼。

AMF 有兩個版本：AMF0 和 AMF3。AMF 0 支援以傳址方式傳送複雜的物件，而且允許端點還原物件關係。AMF 3 不但傳送物件參考，更以傳址方式傳送物件特性和字串，並支援 ActionScript 3.0 新引進的資料類型，藉此改良 AMF 0。ByteArray.objectEcoding 屬性會指定物件資料進行編碼所用的 AMF 版本。flash.net.ObjectEncoding 類別定義了可供您指定 AMF 版本的常數：ObjectEncoding.AMF0 和 ObjectEncoding.AMF3。

下列範例會呼叫 writeObject() 將 XML 物件寫入 ByteArray，接著使用 Deflate 演算法予以壓縮後寫入位於桌面的 order 檔案。此範例會在 AIR 視窗中利用標籤，於作業完成時顯示「Wrote order file to desktop!」訊息。

```
import flash.filesystem.*;
import flash.display.Sprite;
import flash.display.TextField;
import flash.utils.ByteArray;
public class WriteObjectExample extends Sprite
{
    public function WriteObjectExample()
    {
        var bytes:ByteArray = new ByteArray();
        var myLabel:TextField = new TextField();
        myLabel.x = 150;
        myLabel.y = 150;
        myLabel.width = 200;
        addChild(myLabel);

        var myXML:XML =
            <order>
                <item id='1'>
                    <menuName>burger</menuName>
                    <price>3.95</price>
                </item>
                <item id='2'>
                    <menuName>fries</menuName>
                    <price>1.45</price>
                </item>
            </order>;

        // Write XML object to ByteArray
```

```
        bytes.writeObject(myXML);
        bytes.position = 0;//reset position to beginning
        bytes.compress(CompressionAlgorithm.DEFLATE);// compress ByteArray
        writeBytesToFile("order.xml", bytes);
        myLabel.text = "Wrote order file to desktop!";
    }

    private function writeBytesToFile(fileName:String, data:ByteArray):void
    {
        var outFile:File = File.desktopDirectory; // dest folder is desktop
        outFile = outFile.resolvePath(fileName);  // name of file to write
        var outStream:FileStream = new FileStream();
        // open output file stream in WRITE mode
        outStream.open(outFile, FileMode.WRITE);
        // write out the file
        outStream.writeBytes(data, 0, data.length);
        // close it
        outStream.close();
    }
}
```

readObject() 方法會從 ByteArray 讀取 AMF 序列化格式的物件，然後儲存成指定類型的物件。下列範例會先將桌面上的 order 檔案讀入 ByteArray (inBytes) 並予以解壓縮，然後呼叫 readObject() 將之儲存成 XML 物件 orderXML。此範例會使用 for each() 迴圈建構，將每個節點加入至文字區域以供顯示。另外，還會顯示 objectEncoding 屬性的值，以及 order 檔案的內容標題。

```
import flash.filesystem.*;
import flash.display.Sprite;
import flash.display.TextField;
import flash.utils.ByteArray;

public class ReadObjectExample extends Sprite
{
    public function ReadObjectExample()
    {
        var inBytes:ByteArray = new ByteArray();
        // define text area for displaying XML content
        var myTxt:TextField = new TextField();
        myTxt.width = 550;
        myTxt.height = 400;
        addChild(myTxt);
        //display objectEncoding and file heading
        myTxt.text = "Object encoding is: " + inBytes.objectEncoding + "\n\n" + "order file: \n\n";
        readFileIntoByteArray("order", inBytes);

        inBytes.position = 0; // reset position to beginning
        inBytes.uncompress(CompressionAlgorithm.DEFLATE);
        inBytes.position = 0;//reset position to beginning
        // read XML Object
        var orderXML:XML = inBytes.readObject();
```

```
        // for each node in orderXML
        for each (var child:XML in orderXML)
        {
            // append child node to text area
            myTxt.text += child + "\n";
        }
    }

    // read specified file into byte array
    private function readFileIntoByteArray(fileName:String, data:ByteArray):void
    {
        var inFile:File = File.desktopDirectory; // source folder is desktop
        inFile = inFile.resolvePath(fileName);  // name of file to read
        var inStream:FileStream = new FileStream();
        inStream.open(inFile, FileMode.READ);
        inStream.readBytes(data);
        inStream.close();
    }
}
```

ByteArray 範例：讀取 .zip 檔

Adobe AIR 1.0 以及更新的版本

本範例會說明如何讀取含有多個不同類型檔案的單純 .zip 檔。其做法是從每個檔案的中繼資料內擷取相關資料、將每個檔案解壓縮到 ByteArray，再將結果檔案輸出至桌面。

.zip 檔的一般結構是以 PKWARE Inc. 的規格為準，這項規格詳列於 http://www.pkware.com/documents/casestudies/APPNOTE.TXT 中。其開頭為 .zip 封存內第一個檔案的檔案檔頭和檔案資料，且後面接著其它每個檔案的檔案檔頭和檔案資料配對組合 (檔案檔頭的結構將於稍後說明)。接著，.zip 檔可能會包含資料描述項記錄 (通常是在輸出的 zip 檔建立於記憶體而非儲存至磁碟機時)。後續則是幾個其它的選擇性元素：封存解密檔頭、封存額外資料記錄、中央目錄結構、中央目錄記錄的 Zip64 尾端、中央目錄定位器的 Zip64 尾端，以及中央目錄記錄的尾端。

本範例的程式碼僅用來剖析不含資料夾的 zip 檔，同時假設並無資料描述項記錄。位於最終檔案資料後面的各項資訊將全數忽略。

每個檔案的檔案檔頭格式如下：

檔案檔頭簽名	4 個位元組
必要版本	2 個位元組
一般用途位元旗標	2 個位元組
壓縮方法	2 個位元組 (8=DEFLATE；0= 未壓縮)
檔案上次修改時間	2 個位元組
檔案上次修改日期	2 個位元組
crc-32	4 個位元組
壓縮後大小	4 個位元組
解壓縮後大小	4 個位元組
檔案名稱長度	2 個位元組

額外欄位長度	2 個位元組
檔案名稱	變數
額外欄位	變數

接續在檔案檔頭後面的部分就是實際檔案資料，可能已壓縮或未壓縮 (視壓縮方法旗標而定)。旗標為 0 (零) 代表檔案資料未壓縮，8 代表資料使用 DEFLATE 演算法進行壓縮，其它值代表使用其它壓縮演算法。

本範例的使用者介面會包含一個標籤和一個文字區域 (taFiles)。應用程式會將 .zip 檔中所找到每個檔案的下列資訊寫入文字區域：檔案名稱、壓縮後大小與解壓縮後大小。下列 MXML 文件定義 Flex 版本應用程式的使用者介面：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="init();">
    <mx:Script>
        <![CDATA[
            // The application code goes here
        ]]>
    </mx:Script>
    <mx:Form>
        <mx:FormItem label="Output">
            <mx:TextArea id="taFiles" width="320" height="150"/>
        </mx:FormItem>
    </mx:Form>
</mx:WindowedApplication>
```

本程式一開始會執行下列工作：

- 匯入所需的類別

```
import flash.filesystem.*;
import flash.utils.ByteArray;
import flash.events.Event;
```

- 定義使用者介面 (Flash 適用)

```
import fl.controls.*;

//requires TextArea and Label components in the Library
var taFiles = new TextArea();
var output = new Label();
taFiles.setSize(320, 150);
taFiles.move(10, 30);
output.move(10, 10);
output.width = 150;
output.text = "Contents of HelloAir.zip";
addChild(taFiles);
addChild(output);
```

- 定義 bytes ByteArray

```
var bytes:ByteArray = new ByteArray();
```

- 定義變數以儲存檔案檔頭提供的中繼資料

```
// variables for reading fixed portion of file header
var fileName:String = new String();
var flNameLength:uint;
var xfldLength:uint;
var offset:uint;
var compSize:uint;
var uncompSize:uint;
var compMethod:int;
var signature:int;
```

- 定義 File (zfile) 和 FileStream (zStream) 物件以代表 .zip 檔，並指定該 .zip 檔的位置 (即 "HelloAIR.zip" 檔案，位於桌面目錄) 以從中擷取檔案。

```
// File variables for accessing .zip file
var zfile:File = File.desktopDirectory.resolvePath("HelloAIR.zip");
var zStream:FileStream = new FileStream();
```

在 Flex 中，首先執行的程式碼是 init() 方法，此方法會做為 mx:WindowedApplication 根標籤的 creationComplete 處理常式來呼叫。

```
// for Flex
private function init():void
{
```

首先，程式會以 READ 模式開啟 .zip 檔。

```
zStream.open(zfile, FileMode.READ);
```

接著，將 bytes 的 endian 屬性設為 LITTLE_ENDIAN，以指示數值欄位的位元組順序為最低位元組在前。

```
bytes.endian = Endian.LITTLE_ENDIAN;
```

之後，while() 陳述式將起始迴圈反覆執行，直到檔案串流中的目前位置大於或等於檔案大小為止。

```
while (zStream.position < zfile.size)
{
```

迴圈內的第一個陳述式會將檔案串流開頭的 30 個位元組讀入 ByteArray bytes。這 30 個開頭位元組構成了第一個檔案檔頭的固定大小部分。

```
// read fixed metadata portion of local file header
zStream.readBytes(bytes, 0, 30);
```

接著，程式碼自此 30 位元組檔頭的開頭位元組中讀取整數 (signature)。依 ZIP 格式規格定義，每個檔案檔頭的簽名都是十六進位值 0x04034b50；如果簽名並非此值，即表示程式碼執行已超出 .zip 檔的檔案部分，也就是說檔案全都擷取完畢。在此情況下，程式碼會立刻結束 while 迴圈，而不再繼續讀到位元組陣列結尾。

```
bytes.position = 0;
signature = bytes.readInt();
// if no longer reading data files, quit
if (signature != 0x04034b50)
{
    break;
}
```

程式碼的下一段落則指定偏移 8 個單位，以便讀取該位置的檔頭位元組，並將其值儲存在 compMethod 變數中。包含於此位元組的值指出用於壓縮該檔案的壓縮方法。允許的壓縮方法有很多種，但實際上幾乎所有 .zip 檔都會使用 DEFLATE 壓縮演算法。如果目前的檔案使用 DEFLATE 壓縮方法進行壓縮，compMethod 即為 8；若檔案未壓縮，則 compMethod 為 0。

```
bytes.position = 8;
compMethod = bytes.readByte();  // store compression method (8 == Deflate)
```

接續在開頭 30 個位元組後面的是長度不定的檔頭部分，內含檔案名稱，甚至還有額外欄位。這個部分的大小會儲存在 offset 變數中。大小的計算方式則是將檔案名稱長度和額外欄位長度相加 (分別讀取自檔頭偏移位置 26 和 28)。

```
offset = 0;// stores length of variable portion of metadata
bytes.position = 26;  // offset to file name length
flNameLength = bytes.readShort();// store file name
offset += flNameLength; // add length of file name
bytes.position = 28;// offset to extra field length
xfldLength = bytes.readShort();
offset += xfldLength;// add length of extra field
```

之後，程式會讀取長度不定的檔案檔頭部分，而該部分的位元組數目即等於 offset 變數的值。

```
        // read variable length bytes between fixed-length header and compressed file data
        zStream.readBytes(bytes, 30, offset);
```

程式也將從長度不定的檔頭部分讀取檔案名稱，連同檔案壓縮後 (zipped) 與解壓縮後 (原始) 大小一併顯示在文字區域內。

```
// Flash version
        bytes.position = 30;
        fileName = bytes.readUTFBytes(flNameLength); // read file name
        taFiles.appendText(fileName + "\n"); // write file name to text area
        bytes.position = 18;
        compSize = bytes.readUnsignedInt();  // store size of compressed portion
        taFiles.appendText("\tCompressed size is: " + compSize + '\n');
        bytes.position = 22;  // offset to uncompressed size
        uncompSize = bytes.readUnsignedInt();  // store uncompressed size
        taFiles.appendText("\tUncompressed size is: " + uncompSize + '\n');


// Flex version
        bytes.position = 30;
        fileName = bytes.readUTFBytes(flNameLength); // read file name
        taFiles.text += fileName + "\n"; // write file name to text area
        bytes.position = 18;
        compSize = bytes.readUnsignedInt();  // store size of compressed portion
        taFiles.text += "\tCompressed size is: " + compSize + '\n';
        bytes.position = 22;  // offset to uncompressed size
        uncompSize = bytes.readUnsignedInt();  // store uncompressed size
        taFiles.text += "\tUncompressed size is: " + uncompSize + '\n';
```

本範例接著會將目前檔案仍在檔案串流中的其餘部分 (壓縮後大小指定此長度) 讀入 bytes 以覆寫開頭 30 個位元組的檔案檔頭。即使檔案未壓縮，壓縮後大小還是很準確，因為這類檔案在壓縮前後的大小會相等。

```
    // read compressed file to offset 0 of bytes; for uncompressed files
    // the compressed and uncompressed size is the same
    if (compSize == 0) continue;
    zStream.readBytes(bytes, 0, compSize);
```

範例的下一步是將壓縮過的檔案進行解壓縮，然後呼叫 outfile() 函數，將結果檔案寫入輸出檔案串流。傳遞給 outfile() 的參數指定了檔案名稱，以及包含檔案資料的位元組陣列。

```
        if (compMethod == 8) // if file is compressed, uncompress
        {
            bytes.uncompress(CompressionAlgorithm.DEFLATE);
        }
        outFile(fileName, bytes);   // call outFile() to write out the file
```

在上述範例中，bytes.uncompress(CompressionAlgorithm.DEFLATE) 只能在 AIR 應用程式中使用。若要為 AIR 和 Flash Player 解壓縮採用 deflate 壓縮的資料，請叫用 ByteArray 的 inflate() 函數。

右大括弧用來表示 while 迴圈、init() 方法以及 Flex 應用程式程式碼的結尾 (outFile() 方法除外)。當返回 while 迴圈起始處再次執行迴圈時，將繼續處理 .zip 檔中的下一輪位元組；可能是擷取另一個檔案，或因檔案都已處理完畢而停止處理 .zip 檔。

```
    } // end of while loop
} // for Flex version, end of init() method and application
```

outfile() 函數將在桌面上以 WRITE 模式開啟輸出檔，檔案名稱則是由 filename 參數提供。接著，再將 data 參數提供的檔案資料寫入輸出檔案串流 (outStream)，最後關閉該檔案。

```
// Flash version
function outFile(fileName:String, data:ByteArray):void
{
    var outFile:File = File.desktopDirectory; // destination folder is desktop
    outFile = outFile.resolvePath(fileName);  // name of file to write
    var outStream:FileStream = new FileStream();
    // open output file stream in WRITE mode
    outStream.open(outFile, FileMode.WRITE);
    // write out the file
    outStream.writeBytes(data, 0, data.length);
    // close it
    outStream.close();
}

private function outFile(fileName:String, data:ByteArray):void
{
    var outFile:File = File.desktopDirectory; // dest folder is desktop
    outFile = outFile.resolvePath(fileName);  // name of file to write
    var outStream:FileStream = new FileStream();
    // open output file stream in WRITE mode
    outStream.open(outFile, FileMode.WRITE);
    // write out the file
    outStream.writeBytes(data, 0, data.length);
    // close it
    outStream.close();
}
```

第 42 章 網路作業與通訊的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您在 Flash Player 或 AIR 中建置應用程式時，通常需要存取應用程式外部的資源。例如，您可能會將影像的要求傳送至網際網路網站伺服器，接著取得影像資料。您也可能會透過通訊端連線，與應用程式伺服器來回傳送經過序列化的物件。Flash Player 和 AIR API 提供多種類別，讓您的應用程式參與這項交換。這些 API 支援通訊協定 (例如 UDP、TCP、HTTP、RTMP 和 RTMFP) 的 IP 型網路。

下列類別可以用於跨網路傳送和接收資料：

類別	支援的資料格式	通訊協定	說明
Loader	SWF、PNG、JPEG、GIF	HTTP、HTTPS	載入支援的資料類型，以及將資料轉換為顯示物件。 請參閱第 167 頁 「以動態方式載入顯示內容」。
URLLoader	任意 (文字、XML、二進位等)	HTTP、HTTPS	載入任意格式的資料。您的應用程式負責解譯資料。 請參閱第 703 頁 「使用 URLLoader 類別」
FileReference	任意	HTTP	上傳和下載檔案。 請參閱第 561 頁 「使用 FileReference 類別」
NetConnection	視訊、音效、ActionScript 訊息格式 (AMF)	HTTP、HTTPS、RTMP、RTMFP	連接至視訊、音效和遠端物件串流。 請參閱第 403 頁 「使用視訊」。
Sound	音訊	HTTP	載入和播放支援的音效格式。 請參閱第 377 頁 「載入外部聲音檔案」。
XMLSocket	XML	TCP	與 XMLSocket 伺服器交換 XML 訊息。 請參閱 第 692 頁 「XML 通訊端」。
Socket	任意	TCP	連接至 TCP 通訊端伺服器。 請參閱第 687 頁 「二進位用戶端通訊端」。
SecureSocket (AIR)	任意	含有 SSLv3 或 TLSv1 的 TCP	連接至需要 SSL 或 TLS 安全性的 TCP 通訊端伺服器。 請參閱第 688 頁 「安全用戶端通訊端 (AIR)」。
ServerSocket (AIR)	任意	TCP	做為傳入 TCP 通訊端連線的伺服器。 請參閱第 695 頁 「伺服器通訊端」。
DatagramSocket (AIR)	任意	UDP	傳送和接收 UDP 封包。 請參閱 第 697 頁 「UDP 通訊端 (AIR)」

通常，建立網路應用程式時，儲存使用者應用程式狀態的永續性資訊十分有幫助。基於此用途，HTML 網頁和應用程式一般都會使用 Cookie。在 Flash Player 中，您可以基於相同的用途使用 SharedObject 類別。請參閱第 604 頁 「共享物件」 (SharedObject 類別可用於 AIR 應用程式，不過將資料只儲存至一般檔案時的限制較少)。

當您的 Flash Player 或 AIR 應用程式需要與相同電腦上的另一個 Flash Player 或 AIR 應用程式進行通訊時，可以使用 LocalConnection 類別。例如，相同網頁上的兩個 (或以上) SWF 可以彼此通訊。同樣地，在網頁上執行的 SWF 也可以與 AIR 應用程式通訊。請參閱第 716 頁 「與其他 Flash Player 和 AIR 實體進行通訊」。

當您需要與本機電腦上其他非 SWF 程序進行通訊時，可以使用 AIR 2 新增的 NativeProcess 類別。NativeProcess 類別可讓您的 AIR 應用程式啟動其他應用程式，並與之進行通訊。請參閱第 721 頁 「與 AIR 中的原生程序通訊」。

如果您需要執行 AIR 應用程式之電腦的網路環境資訊，請使用下列類別：

- NetworkInfo：提供可用網路介面的資訊 (例如電腦的 IP 位址)。請參閱第 681 頁 「Network 介面」。
- DNSResolver：可讓您查詢 DNS 記錄。請參閱第 684 頁 「網域名稱系統 (DNS) 記錄」。
- ServiceMonitor：可讓您監視伺服器的可用性。請參閱第 683 頁 「服務監視」。
- URLMonitor：可讓您監視特定 URL 資源的可用性。請參閱 第 684 頁 「HTTP 監視」。
- SocketMonitor 和 SecureSocketMonitor：可讓您監視通訊端上資源的可用性。請參閱第 684 頁 「Socket 監視」。

重要概念與術語

下列參考清單包含撰寫網路與通訊程式碼時會遇到的相關重要術語：

外部資料 以某種形式儲存於 應用程式之外的資料，在需要時載入應用程式之中。這種資料可能存放在可以直接載入的檔案中，或是存放在資料庫或是其它可藉由呼叫指令碼或是伺服器上執行的程式來擷取的形式。

URL 編碼變數 URL 編碼格式提供以單一文字字串代表多個變數 (變數名稱 / 數值配對) 的方式。個別的變數則以 name=value 的格式來撰寫。每個變數 (也就是，每個名稱 / 數值配對) 都會以 & 字元來分隔，如下所示：variable1=value1&variable2=value2。這樣一來，所傳送的單一訊息中就可包含無限的變數數量。

MIME 類型 在網際網路通訊中，用來識別特定檔案類型的標準程式碼。任何特定的檔案類型都會有用來識別其類型的特定程式碼。當您傳送檔案或訊息時，電腦 (例如網頁瀏覽器或使用者的 Flash Player 或 AIR 實體) 將指定要傳送的檔案類型。

HTTP 超文字傳輸通訊協定，這是一種標準格式，用來傳遞網頁並透過網際網路傳送各種不同類型的內容。

要求方法 當 AIR 應用程式或網頁瀏覽器之類的應用程式傳送訊息 (稱為 HTTP 要求) 給網站伺服器時，任何傳送的資料都可以用兩種方式之一內嵌到要求中；這兩種 「要求方法」就是 GET 和 POST。在伺服器端，負責接收要求的程式需要在適當的要求區段中找到此資料，因此用來將資料從您的應用程式傳送出去的要求方法，必須與伺服器用來讀取資料的要求方法相符。

通訊端連線 兩個電腦之間通訊的持續連線。

上傳 將檔案傳送至另一部電腦。

下載 從另一部電腦擷取檔案。

Network 介面

Adobe AIR 2 以及更新的版本

您可以使用 NetworkInfo 物件，尋找應用程式可以使用的硬體與軟體網路介面。NetworkInfo 物件是一個 singleton 物件，您不需要建立。請您改用靜態類別屬性 networkInfo，來存取單一 NetworkInfo 物件。當可用的介面之一變更後，NetworkInfo 物件也會傳送 networkChange 事件。

呼叫 findInterfaces() 方法，取得 NetworkInterface 物件清單。清單中的每一個 NetworkInterface 物件會說明可用的介面之一。NetworkInterface 物件提供各種資訊，如 IP 位址、硬體位址、傳輸單位上限以及介面是否作用中。

以下的程式碼範例會追蹤用戶端電腦上每一個介面的 NetworkInterface 屬性：

```
package {
import flash.display.Sprite;
import flash.net.InterfaceAddress;
import flash.net.NetworkInfo;
import flash.net.NetworkInterface;

public class NetworkInformationExample extends Sprite
{
    public function NetworkInformationExample()
    {
        var networkInfo:NetworkInfo = NetworkInfo.networkInfo;
        var interfaces:Vector.<NetworkInterface> = networkInfo.findInterfaces();

        if( interfaces != null )
        {
            trace( "Interface count: " + interfaces.length );
            for each ( var interfaceObj:NetworkInterface in interfaces )
            {
                trace( "\nname: " + interfaceObj.name );
                trace( "display name: " + interfaceObj.displayName );
                trace( "mtu: "   + interfaceObj.mtu );
                trace( "active?: " + interfaceObj.active );
                trace( "parent interface: " + interfaceObj.parent );
                trace( "hardware address: " + interfaceObj.hardwareAddress );
                if( interfaceObj.subInterfaces != null )
                {
                    trace( "# subinterfaces: " + interfaceObj.subInterfaces.length );
                }
                trace("# addresses: " + interfaceObj.addresses.length );
                for each ( var address:InterfaceAddress in interfaceObj.addresses )
                {
                    trace( "  type: "            + address.ipVersion );
                    trace( "  address: " + address.address );
                    trace( "  broadcast: " + address.broadcast );
                    trace( "  prefix length: " + address.prefixLength );
                }
            }
        }
    }
}
}
```

如需詳細資訊，請參閱：

- NetworkInfo
- NetworkInterface
- InterfaceAddress
- Flexpert：使用 Flex 4.5 偵測網路連線類型

網路連線變更

Adobe AIR 1.0 以及更新的版本

AIR 應用程式可以在不穩定且多變的網路連線環境中執行。為了協助應用程式管理與線上資源之間的連線，Adobe AIR 會在每次網路連線的狀態變更為可用或不可用時，傳送網路變更事件。NetworkInfo 物件與應用程式的 NativeApplication 物件會傳送 networkChange 事件。若要回應這個事件，請加入偵聽程式：

```
NetworkInfo.networkInfo.addEventListener(Event.NETWORK_CHANGE, onNetworkChange);
```

並且定義事件處理常式函數：

```
function onNetworkChange(event:Event)
{
    //Check resource availability
}
```

networkChange 事件不會指出所有網路活動中的變更，只會指出已經變更的個別網路連線。AIR 不會嘗試解譯網路變更的意義。已經連接上網路的電腦可能具有許多實體與虛擬連線，因此失去某個連線不一定表示失去資源。另一方面，加入新的連線時，也不一定保證能改善資源的可用性。有時，新連線甚至可能會封鎖先前可用之資源的存取權 (例如，連接至 VPN 時)。

一般而言，應用程式若要判斷自己是否能連接遠端資源，唯一的方法就是嘗試連接該資源。服務監視架構提供以事件為基礎的方式，來回應特定主機的網路連線變更。

備註：服務監視架構會偵測伺服器是否會以可接受的方式回應要求。檢查成功並不保證連線功能全部正常。可調整的網路服務通常會使用快取和載入平衡設備，將流量重新導向至網站伺服器的叢集。在這種情況下，服務提供者只會提供部分網路連線診斷。

服務監視

Adobe AIR 1.0 以及更新的版本

服務監視架構 (與 AIR 架構不同) 位於 aircore.swc 檔案中。若要使用這個架構，您必須將 aircore.swc 檔案包含在建置程序中。

Adobe® Flash® Builder 會自動包含這個元件庫。

ServiceMonitor 類別會實作這個架構以監視網路服務，並且為服務監視器提供基本功能。根據預設，不管網路連線狀態為何，ServiceMonitor 類別的實體都會傳送事件。當建立實體以及執行階段偵測到網路變更時，ServiceMonitor 物件會傳送這些事件。此外，不管一般的網路連線事件為何，您都可以設定 ServiceMonitor 實體的 pollInterval 屬性，根據指定的時間間隔 (以毫秒為單位) 定期檢查連線。直到呼叫 start() 方法之前，ServiceMonitor 物件都不會檢查網路連線。

URLMonitor 是 ServiceMonitor 類別的子類別，可以針對指定的 URLRequest，偵測 HTTP 連線中的變更。

SocketMonitor 也是 ServiceMonitor 類別的子類別，可以偵測透過指定的連接埠，連接至指定之主機的連線中之變更。

備註：在 AIR 2 之前，服務監視架構是發佈於 servicemonitor.swc 元件庫。現在不建議使用這個元件庫。請改用 aircore.swc 元件庫。

Flash CS4 和 CS5 Professional

在 Adobe® Flash® CS4 或 CS5 Professional 中使用這些類別：

1. 選取 「檔案 > 發佈設定」命令。
2. 按一下 ActionScript 3.0 的 「設定」按鈕並選取 「元件庫路徑」。
3. 按一下 「瀏覽至 SWC 檔案」按鈕，並瀏覽至 Flash Professional 安裝資料夾中的 AIK 資料夾。
4. 在這個資料夾內，找到 /frameworks/libs/air/aircore.swc (適用於 AIR 2) 或 /frameworks/libs/air/servicemonitor.swc (適用於 AIR 1.5)。
5. 按一下 「確定」按鈕。
6. 將下列 import 陳述式加入至 ActionScript 3.0 程式碼：

```
import air.net.*;
```

Flash CS3 Professional

若要在 Adobe® Flash® CS3 Professional 中使用這些類別，請將 ServiceMonitorShim 組件從 「組件」面板拖曳至 「元件庫」。然後在 ActionScript 3.0 程式碼中加入下列 import 陳述式：

```
import air.net.*;
```

HTTP 監視

Adobe AIR 1.0 以及更新的版本

URLMonitor 類別會判斷是否能透過連接埠 80 (供 HTTP 通訊使用的典型連接埠)，對指定的位址發出 HTTP 要求。下列程式碼會使用 URLMonitor 類別的實體，偵測 Adobe 網站的連線變更：

```
import air.net.URLMonitor;
import flash.net.URLRequest;
import flash.events.StatusEvent;
var monitor:URLMonitor;
monitor = new URLMonitor(new URLRequest('http://www.example.com'));
monitor.addEventListener(StatusEvent.STATUS, announceStatus);
monitor.start();
function announceStatus(e:StatusEvent):void {
    trace("Status change. Current status: " + monitor.available);
}
```

Socket 監視

Adobe AIR 1.0 以及更新的版本

AIR 應用程式也可以針對推入模型 (Push-Model) 連線，使用通訊端連線。防火牆和網路路由器通常都會基於安全性因素，限制透過未授權的連接埠進行網路通訊。因此，開發人員必須考慮到使用者並無法全天候建立通訊端連線。

下列程式碼會使用 SocketMonitor 類別的實體，偵測通訊端連線的連線變更：監視的連接埠為 6667，IRC 常用的連接埠：

```
import air.net.ServiceMonitor;
import flash.events.StatusEvent;

socketMonitor = new SocketMonitor('www.example.com',6667);
socketMonitor.addEventListener(StatusEvent.STATUS, socketStatusChange);
socketMonitor.start();

function announceStatus(e:StatusEvent):void {
    trace("Status change. Current status: " + socketMonitor.available);
}
```

如果通訊端伺服器需要安全連線，則可以使用 SecureSocketMonitor 類別，來取代 SocketMonitor。

網域名稱系統 (DNS) 記錄

Adobe AIR 2.0 以及更新的版本

您可以使用 DNSResolver 類別來查詢 DNS 資源記錄。DNS 資源記錄提供網域名稱的 IP 位址以及 IP 位址的網域名稱之類的資訊。您可以查詢以下類型的 DNS 資源：

- ARecord— 主機的 IPv4 位址。

- AAAARecord— 主機的 IPv6 位址。
- MXRecord— 主機的郵件交換記錄。
- PTRRecord—IP 位址的主機名稱。
- SRVRecord— 服務的服務記錄。

若要查詢記錄，可以將查詢字串以及代表記錄類型的類別物件，傳送到 DNSResolver 物件的 lookup() 方法。要使用的查詢字串視記錄類型而定：

記錄類別	查詢字串	範例 查詢字串
ARecord	主機名稱	“example.com”
AAAARecord	主機名稱	“example.com”
MXRecord	主機名稱	“example.com”
PTRRecord	IP 位址	“208.77.188.166”
SRVRecord	服務 ID：_service._protocol.host	"_sip._tcp.example.com"

以下程式碼範例會查詢主機 "example.com" 的 IP 位址。

```
package
{
    import flash.display.Sprite;
    import flash.events.DNSResolverEvent;
    import flash.events.ErrorEvent;
    import flash.net.dns.ARecord;
    import flash.net.dns.DNSResolver;

    public class DNSResolverExample extends Sprite
    {

        public function DNSResolverExample()
        {
            var resolver:DNSResolver = new DNSResolver();
            resolver.addEventListener( DNSResolverEvent.LOOKUP, lookupComplete );
            resolver.addEventListener( ErrorEvent.ERROR, lookupError );

            resolver.lookup( "example.com.", ARecord );
        }

        private function lookupComplete( event:DNSResolverEvent ):void
        {
            trace( "Query string: " + event.host );
            trace( "Record count: " + event.resourceRecords.length );
            for each( var record:* in event.resourceRecords )
            {
                if( record is ARecord ) trace( record.address );
            }

        }

        private function lookupError( error:ErrorEvent ):void
        {
            trace("Error: " + error.text );
        }
    }
}
```

如需詳細資訊，請參閱：

- DNSResolver
- DNSResolverEvent
- ARecord
- AAAARecord
- MXRecord
- PTRRecord
- SRVRecord

第 43 章 通訊端

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

通訊端是兩個電腦程序之間所建立的網路連線類型。一般而言，這些程序是在兩部不同的電腦上執行，而這兩部電腦連接至相同的網際網路通訊協定 (IP) 網路。不過，使用特殊 「本機主機」 IP 位址，連接的程序就可以在相同的電腦上執行。

Adobe Flash Player 支援用戶端 「傳輸控制通訊協定 (TCP)」通訊端。Flash Player 應用程式可以連接至另一個做為通訊端伺服器的程序，但是無法接受來自其他程序的傳入連線要求。換句話說，Flash Player 應用程式可以連接至 TCP 伺服器，但是無法做為伺服器。

Flash Player API 也包括 XMLSocket 類別。XMLSocket 類別使用 Flash Player 特定通訊協定，讓您可以與瞭解該通訊協定的伺服器交換 XML 訊息。XMLSocket 類別在 ActionScript 1 時引進，目前仍提供舊版相容性支援。一般而言，除非您連接至為了與 Flash XMLSockets 通訊而特別建立的伺服器，否則新的應用程式應該使用 Socket 類別。

Adobe AIR 新增多種其他類別，以進行通訊端型網路程式設計。AIR 應用程式可以使用 ServerSocket 類別做為 TCP 通訊端伺服器，而且可以使用 SecureSocket 類別連接至需要 SSL 或 TLS 安全性的通訊端伺服器。AIR 應用程式也可以使用 DatagramSocket 類別傳送和接收 「通用資料包通訊協定 (UDP)」訊息。

更多說明主題

flash.net 套件

第 922 頁 「連線至通訊端」

TCP 通訊端

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

「傳輸控制通訊協定」 (TCP) 提供一種方法，可透過持續的網路連線交換訊息。TCP 保證可讓傳送的所有訊息以正確的順序送達 (重大網路問題除外)。TCP 連線需要一個 「用戶端」和一個 「伺服器」。Flash Player 可以建立用戶端通訊端。Adobe AIR 可以額外建立伺服器通訊端。

下列 ActionScript API 可提供 TCP 連線：

- Socket — 允許用戶端應用程式連線到伺服器。Socket 類別無法偵聽傳入的連線。
- SecureSocket (AIR) — 允許用戶端應用程式連線到受信任伺服器，進行加密的通訊。
- ServerSocket (AIR) — 允許應用程式偵聽傳入的連線，並扮演伺服器的角色。
- XMLSocket — 允許用戶端應用程式連線到 XMLSocket 伺服器。

二進位用戶端通訊端

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

二進位通訊端連線與 XML 通訊端連線相當類似，只是用戶端與伺服器並不限於交換 XML 訊息；但這種連線方式會改以二進位資訊來傳輸資料。因此，您可以連線至範圍更廣的服務，包括郵件伺服器 (POP3、SMTP 和 IMAP) 和新聞伺服器 (NNTP)。

Socket 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Socket 類別可讓您進行通訊端連線，以及讀取和寫入原始二進位資料。Socket 類別適用於與使用二進位通訊協定的伺服器交互作業。您可以利用二進位通訊端連線，透過撰寫程式碼，讓不同的網際網路通訊協定 (例如 POP3、SMTP、IMAP 和 NNTP) 之間可以彼此進行互動。然後，這種互動可以讓您的應用程式連線至郵件和新聞伺服器。

Flash Player 可以使用伺服器的二進位通訊協定，直接搭配伺服器運作。有些伺服器會使用 big-endian 位元組順序，有些伺服器會使用 little-endian 位元組順序。由於 「網路位元組順序」為 big-endian，所以網際網路上的多數伺服器都是使用 big-endian 位元組順序。由於 Intel® x86 架構使用的是 little-endian 位元組順序，所以這種順序也廣為使用。對於所要使用的 Endian 位元組順序，應符合傳送或接收資料之伺服器的位元組順序。根據預設，由 IDataInput 和 IDataOutput 介面執行的所有作業，以及實作這些介面的類別 (ByteArray、Socket 和 URLStream) 都是以 big-endian 格式編碼，也就是最高位元組在前；選擇這個預設位元組順序，以符合 Java 和正式網路位元組順序。若要變更所使用的 big-endian 或 little-endian 位元組順序，請將 endian 屬性設為 Endian.BIG_ENDIAN 或 Endian.LITTLE_ENDIAN。

Socket 類別會繼承 IDataInput 和 IDataOutput 介面 (位在 flash.utils 套件中) 定義的所有方法 。您必須使用這些方法，將資料寫入 Socket 或者從中讀取資料。

如需詳細資訊，請參閱：

- Socket
- IDataInput
- IDataOutput
- socketData event

安全用戶端通訊端 (AIR)

Adobe AIR 2 以及更新的版本

您可以使用 SecureSocket 類別來連接至使用 「安全通訊端階層版本 4」 (SSLv4) 或 「傳輸層安全性 (Transport Layer Security，TLS) 版本 1」 (TLSv1) 的通訊端伺服器。安全通訊端提供三個優點：伺服器驗證、資料完整性以及訊息機密性。執行階段會使用伺服器憑證及其與使用者信任儲存區中根或間接憑證授權單位的關係，來驗證伺服器。執行階段需要 SSL 和 TLS 通訊協定實作所使用的密碼演算法，才能提供資料完整性和訊息機密性。

當您使用 SecureSocket 物件連線至伺服器時，執行階段會使用憑證信任儲存區檢查伺服器憑證的有效性。在 Windows 和 Mac 上，作業系統會提供信任儲存區。在 Linux 上，執行階段提供它自己的信任儲存區。

如果伺服器憑證無效或者不被信任，執行階段會傳送 ioError 事件。您可以檢查 SecureSocket 物件的 serverCertificateStatus 屬性，判斷驗證失敗原因。與不具有效和信任憑證的伺服器通訊時，不會提供任何資料。

CertificateStatus 類別會定義代表可能驗證結果的字串常數：

- 過期 — 已經過了憑證到期日。
- 無效 — 憑證無效的原因很多。例如，憑證可能已遭更改、損毀或者憑證的類型錯誤。
- 無效的鍊結 — 一伺服器的憑證鍊結中，一或多個憑證無效。
- 主體不相符 — 伺服器的主機名稱以及憑證一般名稱不相符。換言之，伺服器正在使用錯誤的憑證。
- 已撤銷 — 發照單位已經撤銷憑證。
- 受信任 — 憑證有效而且值得信任。SecureSocket 物件只能連線至使用有效、受信任憑證的伺服器。
- 不明 —SecureSocket 物件尚未驗證憑證。在您呼叫 connect() 之前以及傳送 connect 或 ioError 事件之前，serverCertificateStatus 屬性就是這種狀態值。
- 未受信任的簽署人 — 憑證未 「鍊結」至用戶端電腦信任儲存區中可信任的根憑證。

與 SecureSocket 物件通訊時，伺服器一定要使用安全的通訊協定，而且具有一個有效、受信任的憑證。其他方面，使用 SecureSocket 物件與使用 Socket 物件並無不同。

並不是所有平台都支援 SecureSocket 物件。請使用 SecureSocket 類別的 isSupported 屬性，來測試執行階段是否支援在目前的用戶端電腦上使用 SecureSocket 物件。

如需詳細資訊，請參閱：

- SecureSocket
- CertificateStatus
- IDataInput
- IDataOutput
- socketData event

TCP 通訊端範例：建立 Telnet 用戶端

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Telnet 範例將示範使用 Socket 類別與遠端伺服器建立連線並傳輸資料的技巧。這個範例將示範下列技巧：

- 使用 Socket 類別建立自訂 telnet 用戶端
- 使用 ByteArray 物件將文字傳送至遠端伺服器
- 處理從遠端伺服器接收的資料

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。您可以在 Samples/Telnet 檔案夾中找到 Telnet 應用程式檔案，此應用程式是由下列檔案組成：

檔案	說明
TelnetSocket.fla 或 TelnetSocket.mxml	包含使用者介面的主應用程式檔案，在 Flex 中為 MXML，在 Flash 中為 FLA。
TelnetSocket.as	提供使用者介面邏輯的文件類別 (僅適用於 Flash)。
com/example/programmingas3/Telnet/Telnet.as	提供應用程式的 Telnet 用戶端功能，例如連線至遠端伺服器，以及傳送、接收和顯示資料。

Telnet 通訊端應用程式概觀

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

主 TelnetSocket.mxml 檔案負責建立整個應用程式的使用者介面 (UI)。

除了負責建立 UI 之外，這個檔案還會定義兩個方法 (login() 和 sendCommand())，用來讓使用者連線至指定的伺服器。

下列程式碼將列出主應用程式檔案中的 ActionScript：

```
import com.example.programmingas3.socket.Telnet;

private var telnetClient:Telnet;
private function connect():void
{
    telnetClient = new Telnet(serverName.text, int(portNumber.text), output);
    console.title = "Connecting to " + serverName.text + ":" + portNumber.text;
    console.enabled = true;
}
private function sendCommand():void
{
    var ba:ByteArray = new ByteArray();
    ba.writeMultiByte(command.text + "\n", "UTF-8");
    telnetClient.writeBytesToSocket(ba);
    command.text = "";
}
```

程式碼的第一行會從自訂的 com.example.programmingas.socket 套件中匯入 Telnet 類別。程式碼的第二行會宣告 Telnet 類別的實體 telnetClient，這個實體稍後將由 connect() 方法進行初始化作業。接著，宣告 connect() 方法並對之前宣告的 telnetClient 變數進行初始化作業。這個方法會將使用者指定的 telnet 伺服器名稱、telnet 伺服器連接埠和參考傳遞給顯示清單上的 TextArea 組件，以顯示來自通訊端伺服器的文字回應。connect() 方法的最後兩行則是設定 「面板」的 title 屬性並啟用「面板」組件，讓使用者可以將資料傳送至遠端伺服器。主應用程式檔案中的最後一個方法 sendCommand()，會用來將使用者的命令傳送到遠端伺服器做為 ByteArray 物件。

Telnet 類別概觀

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Telnet 類別負責連線至遠端 Telnet 伺服器以及傳送 / 接收資料。

Telnet 類別會宣告下列私有變數：

```
private var serverURL:String;
private var portNumber:int;
private var socket:Socket;
private var ta:TextArea;
private var state:int = 0;
```

第一個變數 serverURL 包含使用者指定要連線的伺服器位址。

第二個變數 portNumber 是目前正在執行 Telnet 伺服器的埠號。根據預設，Telnet 服務是在埠號 23 上執行。

第三個變數 socket 是 Socket 實體，它的工作是嘗試連線至由 serverURL 和 portNumber 變數定義的伺服器。

第四個變數 ta 是 「舞台」上 TextArea 組件實體的參考。這個組件會用來顯示來自遠端 Telnet 伺服器的回應，或任何可能的錯誤訊息。

最後一個變數 state 是一項數值，用來決定您 Telnet 用戶端可支援的選項。

如前所述，Telnet 類別的建構函式是由主應用程式檔案中的 connect() 方法來呼叫。

Telnet 建構函式會使用三個參數：server、port 和 output。server 和 port 參數可以指定執行 Telnet 服務之伺服器的伺服器名稱和埠號。最後一個參數 output 是 「舞台」上 TextArea 組件實體的參考，會用來將伺服器輸出顯示給使用者。

```
public function Telnet(server:String, port:int, output:TextArea)
{
    serverURL = server;
    portNumber = port;
    ta = output;
    socket = new Socket();
    socket.addEventListener(Event.CONNECT, connectHandler);
    socket.addEventListener(Event.CLOSE, closeHandler);
    socket.addEventListener(ErrorEvent.ERROR, errorHandler);
    socket.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
    socket.addEventListener(ProgressEvent.SOCKET_DATA, dataHandler);
    Security.loadPolicyFile("http://" + serverURL + "/crossdomain.xml");
    try
    {
        msg("Trying to connect to " + serverURL + ":" + portNumber + "\n");
        socket.connect(serverURL, portNumber);
    }
    catch (error:Error)
    {
        msg(error.message + "\n");
        socket.close();
    }
}
```

將資料寫入通訊端

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要將資料寫入通訊端連線，您可以呼叫 Socket 類別中的任何寫入方法。這些寫入方法包括 writeBoolean()、writeByte()、writeBytes()、writeDouble() 以及其他等等。然後，使用 flush() 方法，清除輸出緩衝區的資料。在 Telnet 伺服器中，會使用 writeBytes() 方法將資料寫入通訊端連線，此方法會以位元組陣列做為參數，並將它傳送至輸出緩衝區。writeBytesToSocket() 方法如下所示：

```
public function writeBytesToSocket(ba:ByteArray):void
{
    socket.writeBytes(ba);
    socket.flush();
}
```

主應用程式檔案中的 sendCommand() 方法會呼叫此方法。

顯示來自通訊端伺服器的訊息

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

每當從通訊端伺服器收到訊息或發生事件時，就會呼叫自訂的 msg() 方法。這個方法會將字串附加至 「舞台」上的 TextArea，然後呼叫自訂的 setScroll() 方法，讓 TextArea 組件捲動到底部。下列範例將示範如何使用 msg() 方法：

```
private function msg(value:String):void
{
    ta.text += value;
    setScroll();
}
```

如果沒有設定自動捲動 TextArea 組件的內容，使用者就必須以手動方式將捲軸拖曳到文字區域，才能看到最新的伺服器回應。

捲動 TextArea 組件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

setScroll() 方法中只包含一行 ActionScript，可以垂直捲動 TextArea 組件的內容，以便讓使用者看到最後傳回的文字。下列程式碼片段將說明如何使用 setScroll() 方法：

```
public function setScroll():void
{
    ta.verticalScrollPosition = ta.maxVerticalScrollPosition;
}
```

這個方法會設定 verticalScrollPosition 屬性，也就是目前顯示的最上面一列字元的行號，並將它設為 maxVerticalScrollPosition 屬性的值。

XML 通訊端

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

XML 通訊端可讓您與遠端伺服器建立連線，該連線會保持開啟，直到明確關閉後才會中斷。您可以在伺服器和用戶端之間交換字串資料，例如 XML。使用 XML 通訊端伺服器的好處是，用戶端不需要明確地要求資料。伺服器可以不需等待要求直接傳送資料，並可將資料傳送給每個連線的用戶端。

在 Flash Player 和應用程式安全執行程序外的 Adobe AIR 內容中，目標伺服器上必須有通訊端原則檔才能進行 XML 通訊端連線。如需詳細資訊，請參閱第 907 頁「網站控制 (原則檔)」、第 922 頁「連線至通訊端」。

XMLSocket 類別無法自動通過防火牆，因為它跟 RTMP 通訊協定 (Real-Time Messaging Protocol) 不一樣，XMLSocket 並不具有 HTTP 穿透技術功能。如果您需要使用 HTTP 穿透技術，請考慮改用 Flash Remoting 或 Flash Media Server (可支援 RTMP)。

下列限制適用於 Flash Player 內容或應用程式安全執行程序之外的 AIR 應用程式內容，可以使用 XMLSocket 物件連線至伺服器的方式和位置：

- 對於應用程式安全執行程序之外的內容，XMLSocket.connect() 方法只能連線至大於或等於 1024 的 TCP 連接埠號。這項限制的其中一個影響就是：與 XMLSocket 物件通訊的伺服器常駐程式也必須指定至大於或等於 1024 的連接埠號。埠號在 1024 以下的連接埠通常是供系統服務使用，例如 FTP (21)、Telnet (23)、SMTP (25)、HTTP(80) 和 POP3 (110)，因此基於安全上的考量，禁止 XMLSocket 物件使用這些連接埠。限制連接埠號可防止有人不當存取及濫用這些資源。
- 對於應用程式安全執行程序之外的內容，XMLSocket.connect() 方法只能連線至內容所在相同網域中的電腦 (這項限制與 URLLoader.load() 方法的安全性規則完全相同)。若所要連線的伺服器常駐程式不是在與內容相同的網域中執行，您可以在允許從特定網域存取的伺服器上建立跨網域原則檔。如需有關跨網域原則檔的詳細資訊，請參閱：第 928 頁「AIR 安全性」。

備註：設定伺服器與 XMLSocket 物件通訊可能會很困難。如果應用程式不需要即時的互動性，請使用 URLLoader 類別取代 XMLSocket 類別。

您可以使用 XMLSocket 類別的 XMLSocket.connect() 和 XMLSocket.send() 方法，透過通訊端連線在伺服器之間來回傳輸 XML。XMLSocket.connect() 方法會與網站伺服器的連接埠建立通訊端連線。XMLSocket.send() 方法會將 XML 物件傳遞到通訊端連線中所指定的伺服器。

當您叫用 XMLSocket.connect() 方法時，應用程式會開啟與伺服器之間的 TCP/IP 連線，並將該連線保持在開啟狀態，直到發生下列其中一個事件：

- 呼叫 XMLSocket 類別的 XMLSocket.close() 方法。
- 已沒有 XMLSocket 物件的參考。
- Flash Player 結束。
- 連線中斷 (例如，數據機中斷)。

以 XMLSocket 類別連接到伺服器

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要建立通訊端連線，必須建立伺服器端應用程式來等待通訊端連線要求，並將回應傳送到 Flash Player 或 AIR 應用程式中。這類型的伺服器端應用程式可以使用 AIR 或像是 Java、Python 或 Perl 這類程式語言來編寫。若要使用 XMLSocket 類別，伺服器電腦必須執行一個常駐程式，才能了解 XMLSocket 類別所使用的簡單通訊協定。

- XML 訊息是透過全雙工的 TCP/IP 串流 Socket 連線來傳送。
- 每一個 XML 訊息都是完整的 XML 文件，以零 (0) 位元組作為結尾。
- 透過單一的 XMLSocket 連線，可傳送及接收無限多個 XML 訊息。

建立並連線至 Java XML 通訊端伺服器

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列程式碼將示範如何以 Java 撰寫一個簡單的 XMLSocket 伺服器，它會接受傳入的連線，並在命令提示視窗中顯示收到的訊息。雖然您可以在透過命令列啟動伺服器時，指定不同的埠號，但根據預設，新的伺服器會建立在本機電腦的連接埠 8080 上。

建立新的文字文件，並在其中加入下列程式碼：

```
import java.io.*;
import java.net.*;

class SimpleServer
{
    private static SimpleServer server;
    ServerSocket socket;
    Socket incoming;
    BufferedReader readerIn;
    PrintStream printOut;

    public static void main(String[] args)
    {
        int port = 8080;

        try
        {
            port = Integer.parseInt(args[0]);
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            // Catch exception and keep going.
        }

        server = new SimpleServer(port);
    }

    private SimpleServer(int port)
    {
        System.out.println(">> Starting SimpleServer");
        try
        {
            socket = new ServerSocket(port);
            incoming = socket.accept();
            readerIn = new BufferedReader(new InputStreamReader(incoming.getInputStream()));
            printOut = new PrintStream(incoming.getOutputStream());
            printOut.println("Enter EXIT to exit.\r");
            out("Enter EXIT to exit.\r");
            boolean done = false;
```

```
            while (!done)
            {
                String str = readerIn.readLine();
                if (str == null)
                {
                    done = true;
                }
                else
                {
                    out("Echo: " + str + "\r");
                    if(str.trim().equals("EXIT"))
                    {
                        done = true;
                    }
                }
                incoming.close();
            }
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }

    private void out(String str)
    {
        printOut.println(str);
        System.out.println(str);
    }
}
```

將文件儲存到硬碟並命名為 SimpleServer.java，然後使用 Java 編譯器加以編譯，以建立名為 SimpleServer.class 的 Java 類別檔案。

開啟命令列提示，並輸入 java SimpleServer，就可以啟動 XMLSocket 伺服器。SimpleServer.class 檔案可以位於本機電腦或網路上的任何位置，不一定要放在網站伺服器的根目錄下。

如果因為檔案不在 Java 類別路徑內而無法啟動伺服器，請嘗試使用 java -classpath .SimpleServer 來啟動伺服器。

若要從 應用程式連線至 XMLSocket，必須先建立 XMLSocket 類別的新實體，然後在傳遞主機名稱和埠號期間呼叫 XMLSocket.connect() 方法，如下所示：

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

每當您從伺服器接收資料，就會傳送 data 事件 (flash.events.DataEvent.DATA)：

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
    trace("[" + event.type + "] " + event.data);
}
```

若要將資料傳送至 XMLSocket 伺服器，請使用 XMLSocket.send() 方法並傳遞 XML 物件或字串。Flash Player 會將所提供的參數轉換成 String 物件，並在內容後面加上零 (0) 位元組後傳送到 XMLSocket 伺服器：

```
xmlsock.send(xmlFormattedData);
```

XMLSocket.send() 方法並不會傳回值來指出資料是否已傳輸成功。如果在傳送資料時發生錯誤，就會擲出 IOError 錯誤。

傳送到 XML 通訊端伺服器的每項訊息都必須以換行 (\n) 字元做為結尾。

如需詳細資訊，請參閱 XMLSocket。

伺服器通訊端

Adobe AIR 2 以及更新的版本

使用 ServerSocket 類別，允許其他程序使用「傳輸控制通訊協定」(TCP) 通訊端連線至您的應用程式。連線程序可以在本機電腦或其他連接網路的電腦上執行。當 ServerSocket 物件收到連線要求時，就會傳送 connect 事件。ServerSocketConnectEvent 物件在傳送時，會加上內含 Socket 物件的事件。以後與其他程序通訊時，就可以使用這個 Socket 物件。

偵聽傳入的通訊端連線：

1. 建立一個 ServerSocket 物件，並將它繫結至本機連接埠
2. 為 connect 事件新增事件偵聽程式
3. 呼叫 listen() 方法
4. 回應 connect 事件，它會為每一個傳入的連線提供 Socket 物件

ServerSocket 物件持續偵聽新的連線，直到您呼叫 close() 方法。

以下程式碼範例會說明如何建立通訊端伺服器應用程式。範例會偵聽連接埠 8087 的傳入連線。收到連線之後，該範例會傳送訊息 (" 已連線 " 字串) 至用戶端通訊端。然後，伺服器會將任何收到的訊息回傳給用戶端。

```
package
{
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.ProgressEvent;
    import flash.events.ServerSocketConnectEvent;
    import flash.net.ServerSocket;
    import flash.net.Socket;

    public class ServerSocketExample extends Sprite
    {
        private var serverSocket:ServerSocket;
        private var clientSockets:Array = new Array();

        public function ServerSocketExample()
        {
            try
            {
                // Create the server socket
                serverSocket = new ServerSocket();

                // Add the event listener
                serverSocket.addEventListener( Event.CONNECT, connectHandler );
                serverSocket.addEventListener( Event.CLOSE, onClose );

                // Bind to local port 8087
                serverSocket.bind( 8087, "127.0.0.1" );

                // Listen for connections
                serverSocket.listen();
                trace( "Listening on " + serverSocket.localPort );

            }
            catch(e:SecurityError)
            {
```

```
            trace(e);
        }
    }

    public function connectHandler(event:ServerSocketConnectEvent):void
    {
        //The socket is provided by the event object
        var socket:Socket = event.socket as Socket;
        clientSockets.push( socket );

        socket.addEventListener( ProgressEvent.SOCKET_DATA, socketDataHandler);
        socket.addEventListener( Event.CLOSE, onClientClose );
        socket.addEventListener( IOErrorEvent.IO_ERROR, onIOError );

        //Send a connect message
        socket.writeUTFBytes("Connected.");
        socket.flush();

        trace( "Sending connect message" );
    }

    public function socketDataHandler(event:ProgressEvent):void
    {
        var socket:Socket = event.target as Socket

        //Read the message from the socket
        var message:String = socket.readUTFBytes( socket.bytesAvailable );
        trace( "Received: " + message);

        // Echo the received message back to the sender
        message = "Echo -- " + message;
        socket.writeUTFBytes( message );
        socket.flush();
        trace( "Sending: " + message );
    }

    private function onClientClose( event:Event ):void
    {
        trace( "Connection to client closed." );
        //Should also remove from clientSockets array...
    }

    private function onIOError( errorEvent:IOErrorEvent ):void
    {
        trace( "IOError: " + errorEvent.text );
    }

    private function onClose( event:Event ):void
    {
        trace( "Server socket closed by OS." );
    }
}}
```

如需詳細資訊，請參閱：

- ServerSocket
- ServerSocketConnectEvent
- Socket

UDP 通訊端 (AIR)

Adobe AIR 2 以及更新的版本

「通用資料包通訊協定」 (UDP) 提供一種方法，可透過無狀態網路連線來交換訊息。UDP 不保證會按照順序傳送訊息，或者訊息是否真的已傳送。利用 UDP，作業系統的網路程式碼通常會花較少的時間進行封送、追蹤以及確認訊息。因此，UDP 訊息送達目的地應用程式的時間，通常會比 TCP 訊息的延遲較短。

當您必須傳送即時資訊，例如遊戲中的位置更新，或者語音聊天應用程式的聲音封包時，UDP 通訊端通訊十分有用。在這些應用程式中，部分資料遺失是可接受的，而且低的傳輸延遲比保證送到更為重要。至於絕大部分的其他用途方面，TCP 通訊端是較佳的選擇。

您的 AIR 應用程式可以使用 DatagramSocket 與 DatagramSocketDataEvent 類別來傳送和接收 UDP 訊息。傳送或接收 UDP 訊息：

1 建立 DatagramSocket 物件

2 為 data 事件新增事件偵聽程式

3 使用 bind() 方法，將通訊端繫結至本機 IP 和通訊埠。

4 呼叫 send() 方法、傳入目標電腦的 IP 位址和連接埠，藉以傳送訊息

5 回應 data 事件來接收訊息為此事件傳送的 DatagramSocketDataEvent 物件包含 ByteArray 物件，裡面儲存訊息資料。

以下程式碼範例會說明應用程式如何收發 UDP 訊息。這個範例會將內含 「Hello」這個字串的單一訊息傳送到目標電腦。它也會追蹤任何接收到的訊息內容。

```
package
{
import flash.display.Sprite;
import flash.events.DatagramSocketDataEvent;
import flash.events.Event;
import flash.net.DatagramSocket;
import flash.utils.ByteArray;

public class DatagramSocketExample extends Sprite
{
    private var datagramSocket:DatagramSocket;

    //The IP and port for this computer
    private var localIP:String = "192.168.0.1";
    private var localPort:int = 55555;

    //The IP and port for the target computer
    private var targetIP:String = "192.168.0.2";
    private var targetPort:int = 55555;

    public function DatagramSocketExample()
    {
        //Create the socket
        datagramSocket = new DatagramSocket();
        datagramSocket.addEventListener( DatagramSocketDataEvent.DATA, dataReceived );

        //Bind the socket to the local network interface and port
```

```
        datagramSocket.bind( localPort, localIP );

        //Listen for incoming datagrams
        datagramSocket.receive();

        //Create a message in a ByteArray
        var data:ByteArray = new ByteArray();
        data.writeUTFBytes("Hello.");

        //Send the datagram message
        datagramSocket.send( data, 0, 0, targetIP, targetPort);
    }

    private function dataReceived( event:DatagramSocketDataEvent ):void
    {
        //Read the data from the datagram
        trace("Received from " + event.srcAddress + ":" + event.srcPort + "> " +
            event.data.readUTFBytes( event.data.bytesAvailable ) );
    }
}}
```

使用 UDP 通訊端時，請務必考慮以下方面：

- 資料的單一封包大小，不可以大於網路介面或者傳送者與接收者之間的網路節點的傳輸單位上限 (MTU) 的最小值。ByteArray 物件中傳送至 send() 方法的全部資料，都會以單一封包的形式傳送。(在 TCP 中，大型訊息會被分割成數個封包。)
- 傳送者與目標之間不會交握。若目標不存在，或者指定的連接埠沒有作用中的偵聽程式，則會刪除訊息而且不會發生錯誤。
- 當您使用 connect() 方法時，從其他來源傳送來的訊息會予以略過。UDP 連線只提供方便的封包篩選。這表示目標位址和連接埠不一定有一個有效的偵聽程序。
- UDP 流量會造成網路無法負荷。如果發生網路擁塞現象，則網路管理員可能需要實作服務品質的控制。(TCP 具備內建的流量管理，可以降低網路擁塞的影響。)

如需詳細資訊，請參閱：

- DatagramSocket
- DatagramSocketDataEvent
- ByteArray

IPv6 位址

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 9.0.115.0 和更新版本都支援 IPv6 (網際網路通訊協定第 6 版)。IPv6 為 「網際網路通訊協定」的其中一個版本，可支援 128 位元的位址 (與先前僅支援 32 位元的 IPv4 通訊協定相較之下，此為一大革新)。您可能必須啟動您的網路介面上的 IPv6。如需詳細資訊，請參閱存放資料之作業系統的 「說明」。

如果主機系統上支援 IPv6，您可以在 URL 中指定數字的 IPv6 常值位址，並在前後加上方括弧 ([])，如下所示：

```
[2001:db8:ccc3:ffff:0:444d:555e:666f]
```

Flash Player 會根據下列規則，傳回 IPv6 常值：

- 對於 IPv6 位址，Flash Player 會傳回完整字串形式。
- IP 值沒有以兩個分號表示的縮寫。

- 十六進位值都是小寫形式。
- IPv6 位址被方括弧 ([]) 括住。
- 每個位址 (四個部分為一組) 的輸出都是 0 到 4 個十六進位值，而且會省略開頭的那幾個零。
- 如果四個部分為一組的位址全部都是零，則除了下列註明的例外，否則輸出時都只會有一個零 (而非兩個分號)。

Flash Player 傳回的 IPv6 值具有下列例外：

- 未指定的 IPv6 位址 (全部都是零) 會輸出為 [::]。
- loopback 或 localhost IPv6 位址會輸出為 [::1]。
- IPv4 對應 (已轉換為 IPv6) 位址會輸出為 [::ffff:a.b.c.d]，其中 a.b.c.d 是典型的 IPv4 值 (以點十進位表示法表示)。
- 相容於 IPv4 的位址會輸出為 [::a.b.c.d]，其中 a.b.c.d 是典型的 IPv4 值 (以點十進位表示法表示)。

第 44 章 HTTP 通訊

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Adobe® AIR® 和 Adobe® Flash® Player 應用程式可以與 HTTP 型伺服器進行通訊，以載入資料、影像、視訊，以及交換訊息。

更多說明主題

flash.net.URLLoader

flash.net.URLStream

flash.net.URLRequest

flash.net.URLRequestDefaults

flash.net.URLRequestHeader

flash.net.URLRequestMethod

flash.net.URLVariables

載入外部資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 包含可從外部來源載入資料的機制。這些來源可以提供靜態內容 (例如文字檔) 或網頁指令碼產生的動態內容。資料的格式可以形形色色，而 ActionScript 會提供解碼與存取資料所需的功能。您也可以在擷取資料的過程中，將資料傳送至外部伺服器中。

使用 URLRequest 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

載入外部資料的許多 API 會使用 URLRequest 類別，定義必要網路要求的屬性。

URLRequest 屬性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以在任何安全執行程序中設定 URLRequest 物件的下列屬性：

屬性	說明
contentType	以 URL 要求所傳送任何資料的 MIME 內容類型。如果沒有設定 contentType，則會以 application/x-www-form-urlencoded 送出值。
data	一種物件，其中包含要使用 URL 要求傳輸的資料。
digest	字串，唯一識別儲存到 Adobe® Flash® Player 快取 (或從中擷取) 的已簽署 Adobe 平台組件。

屬性	說明
method	HTTP 要求方法，例如 GET 或 POST。(在 AIR 應用程式安全性網域中執行的內容可以指定 "GET" 或 "POST" 之外的字串做為 method 屬性。任何 HTTP 動詞都允許，而 "GET" 是預設方法。請參閱：第 928 頁「AIR 安全性」)。
requestHeaders	要附加至 HTTP 要求之該 HTTP 要求檔頭的陣列。請注意，在 Flash Player 以及在應用程式安全執行程序外部執行的 AIR 內容中，會限制設定部分檔頭的權限。
url	指定將要求的 URL。

在 AIR 中，您可以設定 URLRequest 類別的其他屬性，而這些屬性只能供應用程式安全執行程序中執行的 AIR 內容使用。應用程式安裝執行程序中的內容也可以使用新的 URL 配置 (除了 file 和 http 等標準配置之外) 來定義 URL。

屬性	說明
followRedirects	指定是 (true，預設值) 否 (false) 要追蹤重新導向。AIR 應用程式安全執行程序才支援此項。
manageCookies	指定 HTTP 通訊協定堆疊是 (true，預設值) 否 (false) 應該為此要求管理 Cookie。在 AIR 應用程式安全執行程序才支援設定此屬性。
authenticate	指定是否應該 (true) 針對此要求，處理驗證要求。在 AIR 應用程式安全執行程序才支援設定此屬性。預設值是驗證要求，所以如果伺服器要求認證，可能會顯示驗證對話方塊。您也可以使用 URLRequestDefaults 類別設定使用者名稱和密碼；請參閱第 701 頁「設定 URLRequest 預設值 (僅限 AIR)」。
cacheResponse	指定是否應該快取這個要求的回應資料。在 AIR 應用程式安全執行程序才支援設定此屬性。預設值是快取回應 (true)。
useCache	指定在此 URLRequest 取回資料之前，是否應該參考本機快取。在 AIR 應用程式安全執行程序才支援設定此屬性。預設值 (true) 是使用本機快取版本 (如果有的話)。
userAgent	指定要用在 HTTP 要求中的使用者代理程式字串。

備註：HTMLLoader 類別有相關的屬性，以供設定與由 HTMLLoader 物件載入之內容相關的設定。如需詳細資訊，請參閱第 845 頁「關於 HTMLLoader 類別」。

設定 URLRequest 預設值 (僅限 AIR)

Adobe AIR 1.0 以及更新的版本

URLRequestDefaults 類別可讓您定義 URLRequest 物件的應用程式特定預設設定。例如，下列程式碼設定 manageCookies 和 useCache 屬性的預設值。所有新的 URLRequest 物件都會使用這些屬性的指定值，而非一般預設值。

```
URLRequestDefaults.manageCookies = false;
URLRequestDefaults.useCache = false;
```

備註：僅可針對在 Adobe AIR 中執行的內容來定義 URLRequestDefaults 類別。Flash Player 不支援。

URLRequestDefaults 類別包含 setLoginCredentialsForHost() 方法，可以讓您指定特定主機的預設使用者名稱和密碼。主機是在方法的主機名稱參數中定義，可以是網域 (如 "www.example.com")，也可以是網域加連接埠號 (如 "www.example.com:80")。請注意，"example.com"、"www.example.com" 和 "sales.example.com" 都會被視為唯一的主機。

這些憑證只有在伺服器需要時才會使用。如果使用者已通過驗證 (例如，使用驗證對話方塊)，則呼叫 setLoginCredentialsForHost() 方法並不會變更驗證過的使用者。

下列程式碼設定傳送至 www.example.com 要求所使用的預設使用者名稱和密碼：

```
URLRequestDefaults.setLoginCredentialsForHost("www.example.com", "Ada", "love1816$X");
```

URLRequestDefaults 設定只會套用至目前應用程式網域，但有一個例外。傳遞給 setLoginCredentialsForHost() 方法的認證，會供 AIR 應用程式內任何應用程式網域中進行的要求使用。

如需詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中的 URLRequestDefaults 類別。

URI 配置

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

從任何安全執行程序進行的要求，都可以使用標準 URI 配置 (如下所示)：

http: 與 https:

上述項目適用於標準網際網路 URL (與網頁瀏覽器使用的方式相同)。

file:

使用 file: 指定位於本機檔案系統的檔案 URL。例如：

```
file:///c:/AIR Test/test.txt
```

在 AIR 中，為應用程式安全執行程序中執行的內容定義 URL 時，您也可以使用下列配置：

app:

使用 app: 來指定相對於已安裝應用程式根目錄的路徑。例如，下列路徑指向已安裝應用程式目錄的資源子目錄：

```
app:/resources
```

使用 AIR Debug Launcher (ADL) 啟動 AIR 應用程式時，應用程式目錄是含有應用程式描述器檔案的目錄。

以 File.applicationDirectory 所建立 File 物件的 URL (和 url 屬性) 會使用 app URI 配置，如下所示：

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("assets");
trace(dir.url); // app:/assets
```

app-storage:

使用 app-storage: 來指定相對於應用程式資料儲存目錄的路徑。針對每一個安裝的應用程式 (和使用者)，AIR 都會建立唯一的應用程式儲存目錄，可以用來儲存該應用程式專屬的資料。例如，下列路徑會指向應用程式儲存目錄之 settings 子目錄中的 prefs.xml 檔案：

```
app-storage:/settings/prefs.xml
```

以 File.applicationStorageDirectory 所建立 File 物件的 URL (和 url 屬性) 會使用 app-storage URI 配置，如下所示：

```
var prefsFile:File = File.applicationStorageDirectory;
prefsFile = prefsFile.resolvePath("prefs.xml");
trace(dir.prefsFile); // app-storage:/prefs.xml
```

mailto:

傳遞至 navigateToURL() 函數的 URLRequest 物件中有 mailto 配置，您可以使用該配置。請參閱第 714 頁「在其他應用程式中開啟 URL」。

您可以使用 URLRequest 物件，它使用所有這些 URI 配置來定義一些不同物件 (例如 FileStream 或 Sound 物件) 的 URL 要求。您也可以在 AIR 中執行的 HTML 內容裡使用這些配置，例如，您可以使用在 img 標籤的 src 特質中。

但是您只能在應用程安全執行程序的內容中使用這些 AIR 特定 URI 配置 (app: 和 app-storage:)。如需詳細資訊，請參閱：第 928 頁「AIR 安全性」。

設定 URL 變數

雖然您可以將變數直接新增至 URL 字串。不過，使用 URLVariables 類別來定義要求所需的任何變數，可以更輕鬆地完成此任務。

您有三種方式，可以將參數新增至 URLVariables 物件：

- 在 URLVariables 建構函式內

- 使用 URLVariables.decode() 方法
- 做為 URLVariables 物件本身的動態屬性

下列範例說明這三種方法，也說明如何將變數指定給 URLRequest 物件：

```
var urlVar:URLVariables = new URLVariables( "one=1&two=2" );
urlVar.decode("amp=" + encodeURIComponent( "&" ) );
urlVar.three = 3;
urlVar.amp2 = "&&";
trace(urlVar.toString()); //amp=%26&amp2=%26%26&one=1&two=2&three=3

var urlRequest:URLRequest = new URLRequest( "http://www.example.com/test.cfm" );
urlRequest.data = urlVar;
```

當您在 URLVariables 建構函式或 URLVariables.decode() 方法內定義變數時，請確定針對 URI 字串中具有特殊意義的字元進行 URL 編碼。例如，當您在參數名稱或值中使用 & 符號時，必須將 & 符號從 & 變更為 %26 (因為 & 符號是做為參數分界的符號)，以編碼 & 符號。最上層 encodeURIComponent() 函數可以用於此用途。

使用 URLLoader 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

URLLoader 類別可讓您將要求傳送至伺服器，以及存取傳回的資訊。您也可以使用 URLLoader 類別，存取位於允許本機檔案存取權限之內容 (例如 Flash Player 具有檔案系統的本機安全執行程序和 AIR 應用程式執行程序) 的本機檔案系統的檔案。URLLoader 類別能以文字、二進位資料或 URL 編碼變數的形式，從 URL 下載資料。URLLoader 類別會傳送事件 (例如 complete、httpStatus、ioError、open、progress 和 securityError)。

ActionScript 3.0 事件處理模型與 ActionScript 2.0 模型明顯不同，後者使用 LoadVars.onData、LoadVars.onHTTPStatus 和 LoadVars.onLoad 事件處理常式。如需有關在 ActionScript 3.0 中處理事件的詳細資訊，請參閱第 108 頁 「處理事件」

下載的資料在下載完成之後才能使用。您可以偵聽傳送的 progress 事件來監視下載的進度 (已載入的位元組和總位元組數目)。不過，如果檔案載入夠快，就可能不會傳送 progress 事件。complete 事件會在檔案下載作業順利完成時傳送。設定 URLLoader dataFormat 屬性，可以接收資料做為文字、原始二進位資料或 URLVariables 物件。

URLLoader.load() 方法 (和選用 URLLoader 類別的建構函式) 所使用的單一參數 request 是一個 URLRequest 物件。URLRequest 物件包含單一 HTTP 要求的所有資訊，例如目標 URL、要求方法 (GET 或 POST)、額外的檔頭資訊和 MIME 類型。

例如，要上傳 XML 封包至伺服器端指令碼時，您可以使用下列程式碼：

```
var secondsUTC:Number = new Date().time;
var dataXML:XML =
    <clock>
        <time>{secondsUTC}</time>
    </clock>;
var request:URLRequest = new URLRequest("http://www.yourdomain.com/time.cfm");
request.contentType = "text/xml";
request.data = dataXML.toXMLString();
request.method = URLRequestMethod.POST;
var loader:URLLoader = new URLLoader();
loader.load(request);
```

上一段程式碼片段會先建立 XML 文件 dataXML，內含要傳送至伺服器的 XML 封包。這個範例會將 URLRequest contentType 屬性設為 "text/xml"，並將 XML 文件指定給 URLRequest data 屬性。最後，這個範例會建立 URLLoader 物件，並使用 load() 方法，將要求傳送至遠端指令碼。

使用 URLStream 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

URLStream 類別提供的存取權限，可以在資料到達時下載資料。URLStream 類別也可以讓您在完成下載前關閉串流。下載的資料是以原始二進位資料提供。

從 URLStream 物件讀取資料時，請使用 bytesAvailable 屬性判斷是否有足夠的資料可以使用，再進行讀取。如果您嘗試讀取的資料多於可用資料，則會擲回 EOFError 例外狀況。

httpResponseStatus 事件 **(AIR)**

在 Adobe AIR 中，除了 httpStatus 事件之外，URLStream 類別還會傳送 httpResponseStatus 事件。httpResponseStatus 事件會在任何回應資料之前進行傳遞。httpResponseStatus 事件 (透過 HTTPStatusEvent 類別代表) 包括 responseURL 屬性 (這是從中傳回回應的 URL)，以及 responseHeaders 屬性 (這是代表傳回回應之回應檔頭的 URLRequestHeader 物件陣列)。

從外部文件載入資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

建立動態應用程式時，從外部檔案或伺服器端指令碼載入資料非常有用。這讓您不需要編輯或重新編譯應用程式，就可以建立動態應用程式。例如，如果要建置 「本日小秘訣」應用程式，您可以撰寫一個伺服器端指令碼，每天從資料庫隨機擷取一個提示，再將它儲存成文字檔案。然後您的 應用程式就可以載入靜態文字檔案的內容，不需要每次都先查詢資料庫。

下列程式碼片段會建立 URLRequest 和 URLLoader 物件，載入 params.txt 這個外部文字檔案的內容：

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

根據預設，若未定義要求的方法，Flash Player 和 Adobe AIR 會使用 HTTP GET 方法載入內容。若要使用 POST 方法來傳送資料，請使用靜態常數 URLRequestMethod.POST 將 request.method 屬性設為 POST，如下列程式碼所示：

```
var request:URLRequest = new URLRequest("sendfeedback.cfm");
request.method = URLRequestMethod.POST;
```

在執行階段載入的 params.txt 外部文件包含下列資料：

```
monthNames=January,February,March,April,May,June,July,August,September,October,November,December&dayNames
=Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
```

此檔案包含兩個參數：monthNames 和 dayNames。每個參數都包含一份以逗號區隔且已剖析為字串的清單。您可以使用 String.split() 方法，將這份清單分割成一個陣列。

請避免使用保留字或語言建構做為外部資料檔案中的變數名稱，因為這樣做會使程式碼的閱讀和除錯工作更為困難。

資料一旦完全載入，就會傳送 complete 事件，而外部文件的內容將提供在 URLLoader 的 data 屬性中，如下列程式碼所示：

```
function completeHandler(event)
{
    var loader2 = event.target;
    air.trace(loader2.data);
}
```

如果遠端文件包含名稱 / 數值配對，您可以先傳入已載入檔案的內容，然後使用 URLVariables 類別來剖析資料，如下所示：

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    var variables:URLVariables = new URLVariables(loader2.data);
    trace(variables.dayNames);
}
```

從外部檔案建立的每對名稱 / 數值配對都會成為 URLVariables 物件中的屬性。在上一段程式碼樣本中，變數物件內的每個屬性都被視為一個字串。如果名稱 / 值配對的值是一份項目清單，您便可以呼叫 String.split() 方法，將字串轉換為陣列，如下所示：

```
var dayNameArray:Array = variables.dayNames.split(",");
```

如果從外部文字檔案載入數值資料，請使用最上層函數 (如 int()、uint() 或 Number()) 將這些值轉換成數值。

您也可以將 URLLoader.dataFormat 屬性設為 URLLoaderDataFormat 類別的其中一個靜態屬性，而不要載入遠端檔案的內容做為字串，然後建立新的 URLVariables 物件。URLLoader.dataFormat 屬性的值可以是下列三個值的其中一個：

- URLLoaderDataFormat.BINARY：URLLoader.data 屬性將包含儲存在 ByteArray 物件中的二進位資料。
- URLLoaderDataFormat.TEXT：URLLoader.data 屬性將包含 String 物件中的文字。
- URLLoaderDataFormat.VARIABLES：URLLoader.data 屬性將包含儲存在 URLVariables 物件中已經過 URL 編碼的變數。

下列程式碼會示範，將 URLLoader.dataFormat 屬性設為 URLLoaderDataFormat.VARIABLES 如何讓您自動將載入的資料解析為 URLVariables 物件：

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class URLLoaderDataFormatExample extends Sprite
    {
        public function URLLoaderDataFormatExample()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/params.txt");
            var variables:URLLoader = new URLLoader();
            variables.dataFormat = URLLoaderDataFormat.VARIABLES;
            variables.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                variables.load(request);
            }
            catch (error:Error)
            {
                trace("Unable to load URL: " + error);
            }
        }
        private function completeHandler(event:Event):void
        {
        var loader:URLLoader = URLLoader(event.target);
        trace(loader.data.dayNames);
        }
    }
}
```

備註：URLLoader.dataFormat 的預設值是 URLLoaderDataFormat.TEXT。

如下列範例所示，從外部檔案載入 XML 與載入 URLVariables 的程序是一樣的。您可以建立 URLRequest 實體和 URLLoader 實體，然後使用它們來下載遠端 XML 文件。檔案下載完成後，就會傳送 Event.COMPLETE 事件，外部檔案的內容也會轉換成 XML 實體，讓您可以使用 XML 方法和屬性來加以解析。

```
package
{
    import flash.display.Sprite;
    import flash.errors.*;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLRequest;

    public class ExternalDocs extends Sprite
    {
        public function ExternalDocs()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/data.xml");
            var loader:URLLoader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                loader.load(request);
            }
            catch (error:ArgumentError)
            {
                trace("An ArgumentError has occurred.");
            }
            catch (error:SecurityError)
            {
                trace("A SecurityError has occurred.");
            }
        }
        private function completeHandler(event:Event):void
        {
            var dataXML:XML = XML(event.target.data);
            trace(dataXML.toXMLString());
        }
    }
}
```

使用外部指令碼進行通訊

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

除了載入外部資料檔案以外，您也可以使用 URLVariables 類別，將變數傳送到伺服器端指令碼，並處理伺服器的回應。例如，如果您需要設計一個遊戲程式，不僅要將使用者的分數傳送到伺服器，以計算是否應該將分數列入最高分記錄，甚至還要將使用者的登入資訊傳送給伺服器進行驗證的話，這就是非常有用的方法。伺服器端指令碼可以處理使用者名稱和密碼、比對資料庫進行驗證，並傳回使用者提供的憑證是否有效的確認訊息。

下列程式碼片段將建立名為 variables 的 URLVariables 物件，它會再建立新的 name 變數。接下來，建立一個 URLRequest 物件，指定要做為傳送變數之目的地伺服器端指令碼的 URL。然後設定 URLRequest 物件的 method 屬性，將變數做為 HTTP POST 要求傳送。若要將 URLVariables 物件加入至 URL 要求，請將 URLRequest 物件的 data 屬性設定為之前建立的 URLVariables 物件。最後會建立 URLLoader 實體，並叫用 URLLoader.load() 方法，啟動要求。

```
var variables:URLVariables = new URLVariables("name=Franklin");
var request:URLRequest = new URLRequest();
request.url = "http://www.[yourdomain].com/greeting.cfm";
request.method = URLRequestMethod.POST;
request.data = variables;
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.VARIABLES;
loader.addEventListener(Event.COMPLETE, completeHandler);
try
{
    loader.load(request);
}
catch (error:Error)
{
    trace("Unable to load URL");
}

function completeHandler(event:Event):void
{
    trace(event.target.data.welcomeMessage);
}
```

下列程式碼包含上一個範例中所使用 Adobe ColdFusion® greeting.cfm 文件的內容：

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
    <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

網路服務要求

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

HTTP 型網路服務有很多種。主要類型包括：

- REST
- XML-RPC
- SOAP

若要在 ActionScript 3 中使用網路服務，請建立 URLRequest 物件、使用 URL 變數或 XML 文件建構網路服務呼叫，以及使用 URLLoader 物件將呼叫傳送至服務。Flex 架構包含多種類別，讓使用網路服務更為簡單；特別適用於存取複雜 SOAP 服務時。從 Flash Professional CS3 開始，您可以在 Flash Professional 開發的應用程式和 Flash Builder 開發的應用程式中使用 Flex 類別。

在 HTML 類型 AIR 應用程式中，您可以使用 URLRequest 和 URLLoader 類別或 JavaScript XMLHttpRequest 類別。如果需要，您也可以建立 SWF 元件庫，向 JavaScript 程式碼公開 Flex 架構的網路服務組件。

當您的應用程式在瀏覽器中執行時，除非裝載網路服務的伺服器也裝載允許從其他網域存取的跨網域原則檔，否則只能在呼叫 SWF 的相同網際網路網域中使用網路服務。透過您自己的伺服器對要求進行 Proxy 處理，是在跨網域原則檔無法使用時經常使用的技術。Adobe Blaze DS 和 Adobe LiveCycle 支援網路服務 Proxy 處理。

在 AIR 應用程式中，當網路服務呼叫源自應用程式安全執行程序時，就不需要跨網域原則檔。AIR 應用程式內容絕不會從遠端網域中提供，因此無法參與跨網域原則所防止的攻擊類型。在 HTML 型 AIR 應用程式中，應用程式安全執行程序中的內容可以使用跨網域 XMLHttpRequest。您可以允許其他安全執行程序中的內容使用跨網域 XMLHttpRequest，而該內容會載入至 iframe。

更多說明主題

第 907 頁 「網站控制 (原則檔)」

Adobe BlazeDS

Adobe LiveCycle ES2

REST 架構

XML-RPC

SOAP 通訊協定

REST 樣式網路服務要求

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

REST 樣式網路服務使用 HTTP 方法動詞來指定基本動作，以及使用 URL 變數指定動作詳細資訊。例如，取得項目資料的要求，可以使用 GET 動詞和 URL 變數來指定方法名稱和項目 ID。產生的 URL 字串可能如下：

```
http://service.example.com/?method=getItem&id=d3452
```

若要使用 ActionScript 存取 REST 樣式網路服務，可以使用 URLRequest、URLVariables 和 URLLoader 類別。在 AIR 應用程式的 JavaScript 程式碼中，也可以使用 XMLHttpRequest。

在 ActionScript 中以程式設計 REST 樣式網路服務呼叫，一般會包含下列步驟：

1. 建立 URLRequest 物件。
2. 在要求物件上設定服務 URL 和 HTTP 方法動詞。
3. 建立 URLVariables 物件。
4. 將服務呼叫參數設為變數物件的動態屬性。
5. 將變數物件指定給要求物件的資料屬性。
6. 使用 URLLoader 物件，將呼叫傳送給服務。
7. 處理 URLLoader 所傳送的 complete 事件，此事件指出服務呼叫已完成。偵聽 URLLoader 物件可能傳送的各種錯誤事件，也十分有用。

例如，假設網路服務公開一個測試方法，可以回應要求者的呼叫參數。下列 ActionScript 程式碼可以用來呼叫服務：

```
import flash.events.Event;
import flash.events.ErrorEvent;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.net.URLRequestMethod;
import flash.net.URLVariables;

private var requestor:URLLoader = new URLLoader();
public function restServiceCall():void
{
    //Create the HTTP request object
    var request:URLRequest = new URLRequest( "http://service.example.com/" );
    request.method = URLRequestMethod.GET;

    //Add the URL variables
    var variables:URLVariables = new URLVariables();
    variables.method = "test.echo";
    variables.api_key = "123456ABC";
    variables.message = "Able was I, ere I saw Elba.";
    request.data = variables;

    //Initiate the transaction
    requestor = new URLLoader();
    requestor.addEventListener( Event.COMPLETE, httpRequestComplete );
    requestor.addEventListener( IOErrorEvent.IOERROR, httpRequestError );
    requestor.addEventListener( SecurityErrorEvent.SECURITY_ERROR, httpRequestError );
    requestor.load( request );
}
private function httpRequestComplete( event:Event ):void
{
    trace( event.target.data );
}

private function httpRequestError( error:ErrorEvent ):void{
    trace( "An error occured: " + error.message );
}
```

在 AIR 應用程式的 JavaScript 中，可以使用 XMLHttpRequest 物件進行相同要求：

```
<html>
<head><title>RESTful web service request</title>
<script type="text/javascript">

function makeRequest()
{
    var requestDisplay = document.getElementById( "request" );
    var resultDisplay  = document.getElementById( "result" );

    //Create a conveninece object to hold the call properties
    var request = {};
    request.URL = "http://service.example.com/";
    request.method = "test.echo";
    request.HTTPmethod = "GET";
    request.parameters = {};
    request.parameters.api_key = "ABCDEF123";
    request.parameters.message = "Able was I ere I saw Elba.";
    var requestURL = makeURL( request );
    xmlhttp = new XMLHttpRequest();
    xmlhttp.open( request.HTTPmethod, requestURL, true);
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
```

```
            resultDisplay.innerHTML = xmlhttp.responseText;
        }
    }
    xmlhttp.send(null);

    requestDisplay.innerHTML = requestURL;
}
//Convert the request object into a properly formatted URL
function makeURL( request )
{
    var url = request.URL + "?method=" + escape( request.method );
    for( var property in request.parameters )
    {
        url += "&" + property + "=" + escape( request.parameters[property] );
    }

    return url;
}
</script>
</head>
<body onload="makeRequest()">
<h1>Request:</h1>
<div id="request"></div>
<h1>Result:</h1>
<div id="result"></div>
</body>
</html>
```

XML-RPC 網路服務要求

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

XML-RPC 網路服務接受的呼叫參數是 XML 文件，而非一組 URL 變數。若要使用 XML-RPC 網路服務來建構交易，請建立適當格式化的 XML 訊息，並使用 HTTP POST 方法將它傳送給網路服務。此外，您應該設定要求的 Content-Type 標頭，讓伺服器將要求資料視為 XML。

下列範例說明如何使用 REST 範例中所顯示的相同網路服務呼叫，但這次是做為 XML-RPC 服務：

```
import flash.events.Event;
import flash.events.ErrorEvent;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.net.URLRequestMethod;
import flash.net.URLVariables;
public function xmlRPCRequest():void
{
    //Create the XML-RPC document
    var xmlRPC:XML = <methodCall>
                                    <methodName></methodName>
                                    <params>
                                        <param>
                                            <value>
                                                <struct/>
                                            </value>
                                        </param>
                                    </params>
                                </methodCall>;

    xmlRPC.methodName = "test.echo";
```

```
    //Add the method parameters
    var parameters:Object = new Object();
    parameters.api_key = "123456ABC";
    parameters.message = "Able was I, ere I saw Elba.";

    for( var propertyName:String in parameters )
    {
        xmlRPC..struct.member[xmlRPC..struct.member.length + 1] =
            <member>
                <name>{propertyName}</name>
                <value>
                    <string>{parameters[propertyName]}</string>
                </value>
            </member>;
    }

    //Create the HTTP request object
    var request:URLRequest = new URLRequest( "http://service.example.com/xml-rpc/" );
    request.method = URLRequestMethod.POST;
    request.cacheResponse = false;
    request.requestHeaders.push(new URLRequestHeader("Content-Type", "application/xml"));
    request.data = xmlRPC;

    //Initiate the request
    requestor = new URLLoader();
    requestor.dataFormat = URLLoaderDataFormat.TEXT;
    requestor.addEventListener( Event.COMPLETE, xmlRPCRequestComplete );
    requestor.addEventListener( IOErrorEvent.IO_ERROR, xmlRPCRequestError );
    requestor.addEventListener( SecurityErrorEvent.SECURITY_ERROR, xmlRPCRequestError );
    requestor.load( request );
}

private function xmlRPCRequestComplete( event:Event ):void
{
    trace( XML(event.target.data).toXMLString() );
}

private function xmlRPCRequestError( error:ErrorEvent ):void
{
    trace( "An error occurred: " + error );
}
```

AIR 中的 WebKit 不支援 E4X 語法，因此上一個範例中用來建立 XML 文件的方法不適用於 JavaScript 程式碼。您必須改用 DOM 方法來建立 XML 文件或以字串來建立文件，並使用 JavaScript DOMParser 類別將字串轉換為 XML。

下列範例使用 DOM 方法建立 XML-RPC 訊息和 XMLHttpRequest，以建構網路服務交易：

```
<html>
<head>
<title>XML-RPC web service request</title>
<script type="text/javascript">

function makeRequest()
{
    var requestDisplay = document.getElementById( "request" );
    var resultDisplay  = document.getElementById( "result" );

    var request = {};
    request.URL = "http://services.example.com/xmlrpc/";
    request.method = "test.echo";
    request.HTTPmethod = "POST";
    request.parameters = {};
    request.parameters.api_key = "123456ABC";
    request.parameters.message = "Able was I ere I saw Elba.";
    var requestMessage = formatXMLRPC( request );

    xmlhttp = new XMLHttpRequest();
    xmlhttp.open( request.HTTPmethod, request.URL, true);
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            resultDisplay.innerText = xmlhttp.responseText;
        }
    }
    xmlhttp.send( requestMessage );

    requestDisplay.innerText = xmlToString( requestMessage.documentElement );
}

//Formats a request as XML-RPC document
function formatXMLRPC( request )
{
    var xmldoc = document.implementation.createDocument( "", "", null );
    var root = xmldoc.createElement( "methodCall" );
    xmldoc.appendChild( root );
    var methodName = xmldoc.createElement( "methodName" );
    var methodString = xmldoc.createTextNode( request.method );
    methodName.appendChild( methodString );

    root.appendChild( methodName );

        var params = xmldoc.createElement( "params" );
        root.appendChild( params );

        var param = xmldoc.createElement( "param" );
        params.appendChild( param );
        var value = xmldoc.createElement( "value" );
        param.appendChild( value );
        var struct = xmldoc.createElement( "struct" );
        value.appendChild( struct );

        for( var property in request.parameters )
        {
            var member = xmldoc.createElement( "member" );
            struct.appendChild( member );

            var name = xmldoc.createElement( "name" );
            var paramName = xmldoc.createTextNode( property );
            name.appendChild( paramName )
            member.appendChild( name );
```

```
            var value = xmldoc.createElement( "value" );
            var type = xmldoc.createElement( "string" );
            value.appendChild( type );
            var paramValue = xmldoc.createTextNode( request.parameters[property] );
            type.appendChild( paramValue )
            member.appendChild( value );
        }
    return xmldoc;
}

//Returns a string representation of an XML node
function xmlToString( rootNode, indent )
{
    if( indent == null ) indent = "";
    var result = indent + "<" + rootNode.tagName + ">\n";
    for( var i = 0; i < rootNode.childNodes.length; i++)
    {
        if(rootNode.childNodes.item( i ).nodeType == Node.TEXT_NODE )
        {
            result += indent + "    " + rootNode.childNodes.item( i ).textContent + "\n";
        }
    }
    if( rootNode.childElementCount > 0 )
    {
        result += xmlToString( rootNode.firstElementChild, indent + "    " );
    }
    if( rootNode.nextElementSibling )
    {
        result += indent + "</" + rootNode.tagName + ">\n";
        result += xmlToString( rootNode.nextElementSibling, indent );
    }
    else
    {
        result += indent +"</" + rootNode.tagName + ">\n";
    }
    return result;
}

</script>
</head>
<body onload="makeRequest()">
<h1>Request:</h1>
<pre id="request"></pre>
<h1>Result:</h1>
<pre id="result"></pre>
</body>
</html>
```

SOAP 網路服務要求

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

SOAP 是根據一般 XML-RPC 網路服務概念所建置，並提供較豐富但較複雜的方法來傳送類型資料。SOAP 網路服務通常會提供網路服務說明語言檔案 (WSDL)，以指定網路服務呼叫、資料類型和服務 URL。雖然 ActionScript 3 不直接支援 SOAP，但是您可以「手動」建構 SOAP XML 訊息，並將它公佈至伺服器，然後剖析結果。不過，針對最簡單 SOAP 網路服務以外的任何項目，使用現有 SOAP 元件庫，還可能節省大量開發時間。

Flex 架構包括用於存取 SOAP 網路服務的元件庫。在 Flash Builder 中，元件庫 (rpc.swc) 會自動併入 Flex 專案中，因為它是 Flex 架構的一部分。在 Flash Professional 中，您可以將 Flex framework.swc 和 rpc.swc 新增至專案的元件庫路徑，然後使用 ActionScript 存取 Flex 類別。

更多說明主題

在 Flash Professional 中使用 Flex 網路服務組件

Cristophe Coenraets：適用於 Android 的即時交易人桌面

在其他應用程式中開啟 URL

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 navigateToURL() 函數，在網頁瀏覽器或其他應用程式中開啟 URL。對於 AIR 中執行的內容，navigateToURL() 函數會在預設系統網頁瀏覽器中開啟頁面。

您傳遞做為此函數的 request 參數之 URLRequest 物件僅會使用 url 屬性。

navigateToURL() 函數的第一個參數是 navigate 參數，它是一個 URLRequest 物件 (請參閱第 700 頁 「使用 URLRequest 類別」)。第二個是選擇性的 window 參數，您可以在其中指定視窗名稱。例如，下列程式碼會開啟 www.adobe.com 網頁：

```
var url:String = "http://www.adobe.com";
var urlReq:URLRequest = new URLRequest(url);
navigateToURL(urlReq);
```

備註：在使用 navigateToURL() 函數時，執行階段會將使用 POST 方法的 URLRequest 物件 (亦即，method 屬性設為 URLRequestMethod.POST 的物件) 視為使用 GET 方法。

使用 navigateToURL() 函數時，是根據呼叫 navigateToURL() 函數的安全執行程序，來允許 URI 配置。

有些 API 允許您在網頁瀏覽器中啟動內容。基於安全考量，在 AIR 中使用這些 API 時會禁止使用一些 URI 配置。禁止使用的配置清單會依使用 API 的程式碼安全執行程序而有所不同。(如需有關安全執行程序的詳細資訊，請參閱：第 928 頁 「AIR 安全性」)。

應用程式安全執行程序 **(僅限 AIR)**

由在 AIR 應用程式安全執行程序中執行的內容所啟動的 URL，可以使用任何 URI 配置。您必須註冊應用程式，才能處理 URI 配置，否則要求不會有任何回應。許多電腦與裝置均支援下列配置：

- http:
- https:
- file:
- mailto: - AIR 會將這些要求導向已登錄的系統郵件應用程式。
- sms: — AIR 會將 sms: 要求導向預設的文字訊息應用程式。URL 格式必須符合執行應用程式的系統慣例。例如，在 Android 上，URI 配置必須是小寫。

  ```
  navigateToURL( new URLRequest( "sms:+15555550101") );
  ```

- tel: — AIR 會將 tel: 要求導向預設的電話撥號應用程式。URL 格式必須符合執行應用程式的系統慣例。例如，在 Android 上，URI 配置必須是小寫。

  ```
  navigateToURL( new URLRequest( "tel:5555555555") );
  ```

- market: — AIR 會將 market: 要求導向 Android 裝置一般支援的 Market 應用程式。

```
navigateToURL( new URLRequest( "market://search?q=Adobe Flash") );
navigateToURL( new URLRequest( "market://search?q=pname:com.adobe.flashplayer") );
```

只要作業系統允許，便可以在該處定義和註冊自訂的 URI 配置。您可以使用配置建立 URL，從 AIR 來啟動應用程式。

遠端安全執行程序
允許下列配置。使用這些配置的方法與在網頁瀏覽器中相同。

- http:
- https:
- mailto: - AIR 會將這些要求導向已登錄的系統郵件應用程式。

所有其它 URI 配置都禁止使用。

具有檔案系統的本機安全執行程序
允許下列配置。使用這些配置的方法與在網頁瀏覽器中相同。

- file:
- mailto: - AIR 會將這些要求導向已登錄的系統郵件應用程式。

所有其它 URI 配置都禁止使用。

具有網路連線的本機安全執行程序
允許下列配置。使用這些配置的方法與在網頁瀏覽器中相同。

- http:
- https:
- mailto: - AIR 會將這些要求導向已登錄的系統郵件應用程式。

所有其它 URI 配置都禁止使用。

本機信任安全執行程序
允許下列配置。使用這些配置的方法與在網頁瀏覽器中相同。

- file:
- http:
- https:
- mailto: - AIR 會將這些要求導向已登錄的系統郵件應用程式。

所有其它 URI 配置都禁止使用。

第 45 章 與其他 Flash Player 和 AIR 實體進行通訊

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

LocalConnection 類別可讓 Adobe® AIR® 應用程式彼此通訊，以及讓瀏覽器中執行的 SWF 內容彼此通訊。您亦可使用 LocalConnection 類別，在 AIR 應用程式與瀏覽器中執行的 SWF 內容之間進行通訊。LocalConnection 可以讓您建立各種可以在多個 Flash Player 和 AIR 實體之間共享資料的多功能應用程式。

關於 LocalConnection 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

LocalConnection 類別可以讓您開發不需要使用 fscommand() 方法或 JavaScript 就可以相互傳送指令的 SWF 檔。LocalConnection 物件只可以讓在相同用戶端電腦上執行的 SWF 檔間彼此通訊，但這些檔案可以在不同的應用程式中執行。例如，在瀏覽器中執行的 SWF 檔與在放映檔中執行的 SWF 檔可以共享資訊，放映檔會維護本機資訊，而在瀏覽器中執行的 SWF 檔則採用遠端連線的方式與其共享資訊 (放映檔是一種 SWF 檔，其儲存格式可允許做為獨立應用程式執行，也就是說，不需要先安裝 Flash Player 就可以執行，因為它已經內嵌於可執行檔內)。

不同的 ActionScript 版本可以使用 LocalConnection 物件在不同的 SWF 檔之間進行通訊：

- ActionScript 3.0 的 LocalConnection 物件可以與 ActionScript 1.0 或 2.0 所建立的 LocalConnection 物件通訊。
- ActionScript 1.0 或 2.0 的 LocalConnection 物件可以與 ActionScript 3.0 所建立的 LocalConnection 物件通訊。

Flash Player 會在不同版本的 LocalConnection 物件之間，自動處理這項通訊。

使用 LocalConnection 物件最簡單的方法就是只允許位於相同網域或相同 AIR 應用程式中的 LocalConnection 物件互相通訊。這樣一來，您就不用擔心安全性問題了。然而，如果您需要讓網域之間彼此通訊，則有幾種方式可用來執行安全性措施。如需詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考中關於 send() 方法的 connectionName 參數討論，以及關於 LocalConnection 類別中的 allowDomain() 和 domain 項目的討論。

您可以使用 LocalConnection 物件傳送及接收單一 SWF 檔中的資料，不過 Adobe 不建議您採用這種方式，請改用共用的物件。

有三種方式可將回呼方法加入 LocalConnection 物件：

- 讓 LocalConnection 類別成為子類別，再加入方法。
- 將 LocalConnection.client 屬性設成實作這些方法的物件。
- 建立會擴充 LocalConnection 的動態類別，再以動態方式加入方法。

第一種加入回呼方法的方式是擴充 LocalConnection 類別。您可以在自訂類別內定義這些方法，而不是以動態方式將它們加入 LocalConnection 實體。下列程式碼將示範這個做法：

```
package
{
    import flash.net.LocalConnection;
    public class CustomLocalConnection extends LocalConnection
    {
        public function CustomLocalConnection(connectionName:String)
        {
            try
            {
                connect(connectionName);
            }
            catch (error:ArgumentError)
            {
                // server already created/connected
            }
        }
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

如果要建立 CustomLocalConnection 類別的新實體，您可以使用下列程式碼：

```
var serverLC:CustomLocalConnection;
serverLC = new CustomLocalConnection("serverName");
```

第二種加入回呼方法的方式是使用 LocalConnection.client 屬性。這種做法會先建立一個自訂類別，並將新的實體指定給 client 屬性，如下列程式碼所示：

```
var lc:LocalConnection = new LocalConnection();
lc.client = new CustomClient();
```

LocalConnection.client 屬性指出應該叫用的物件回呼方法。在上一段程式碼中，client 屬性是設為自訂類別 CustomClient 的新實體。client 屬性的預設值為目前的 LocalConnection 實體。當您有兩個資料處理常式是使用一組相同的方法但卻有不同的運作方式時 (例如，應用程式某個視窗中的一個按鈕會切換第二個視窗中的檢視狀態)，請使用 client 屬性。

若要建立 CustomClient 類別，您可以使用下列程式碼：

```
package
{
    public class CustomClient extends Object
    {
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

第三種加入回呼方法的方式是建立一個動態類別，再以動態方式加入這些方法。這種方式和在舊版 ActionScript 中使用 LocalConnection 類別的方式非常類似，如下列程式碼所示：

```
import flash.net.LocalConnection;
dynamic class DynamicLocalConnection extends LocalConnection {}
```

您可以使用下列程式碼，以動態方式將回呼方法加入這個類別：

```
var connection:DynamicLocalConnection = new DynamicLocalConnection();
connection.onMethod = this.onMethod;
// Add your code here.
public function onMethod(timeString:String):void
{
    trace("onMethod called at: " + timeString);
}
```

上一種加入回呼方法的做法並不是理想的處理方式，因為這種程式碼不容易移植，不建議使用。此外，使用這種建立本機連線的方法同時也會產生效能方面的問題，因為存取動態屬性的速度遠遠不及存取密封屬性的速度。

isPerUser 屬性

isPerUser 屬性已新增至 Flash Player (10.0.32) 和 AIR (1.5.2)，以解決當一個以上的使用者登入 Mac 電腦時所產生的衝突。在其他作業系統上，本機連線一律僅限個別使用者使用，因此會忽略此屬性。isPerUser 屬性應該在新程式碼中設為 true。但是，預設值目前為 false 以提供舊版相容性。預設值可能會在未來發行的執行階段版本中進行變更。

在兩個應用程式之間傳送訊息

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 LocalConnection 類別在不同的 AIR 應用程式之間進行通訊，以及在瀏覽器中執行的不同 Adobe® Flash® Player (SWF) 應用程式之間進行通訊。您亦可使用 LocalConnection 類別，在 AIR 應用程式與瀏覽器中執行的 SWF 應用程式之間進行通訊。

例如，網頁上可以有多個 Flash Player 實體，或是讓某個 Flash Player 實體從彈出式視窗中的 Flash Player 實體擷取資料。

下列程式碼會定義做為伺服器運作的 LocalConnection 物件，並接受從其它應用程式傳入的 LocalConnection 呼叫：

```
package
{
    import flash.net.LocalConnection;
    import flash.display.Sprite;
    public class ServerLC extends Sprite
    {
        public function ServerLC()
        {
            var lc:LocalConnection = new LocalConnection();
            lc.client = new CustomClient1();
            try
            {
                lc.connect("conn1");
            }
            catch (error:Error)
            {
                trace("error:: already connected");
            }
        }
    }
}
```

這段程式碼會先建立名為 lc 的 LocalConnection 物件，然後將 client 屬性設定為物件 clientObject。當其它應用程式呼叫這個 LocalConnection 實體中的方法時，執行階段就會在 clientObject 物件中尋找該方法。

如果已經使用指定的名稱完成連線，便會擲回引數錯誤例外，指出連線嘗試失敗，因為該物件已經連線。

每當 Flash Player 實體連線至這個 SWF 檔，並嘗試為指定的本機連線叫用方法時，這項要求就會傳送至 client 屬性指定的類別 (已設定為 CustomClient1 類別)：

```
package
{
    import flash.events.*;
    import flash.system.fscommand;
    import flash.utils.Timer;
    public class CustomClient1 extends Object
    {
        public function doMessage(value:String = ""):void
        {
            trace(value);
        }
        public function doQuit():void
        {
            trace("quitting in 5 seconds");
            this.close();
            var quitTimer:Timer = new Timer(5000, 1);
            quitTimer.addEventListener(TimerEvent.TIMER, closeHandler);
        }
        public function closeHandler(event:TimerEvent):void
        {
            fscommand("quit");
        }
    }
}
```

若要建立 LocalConnection 伺服器，請呼叫 LocalConnection.connect() 方法並提供唯一的連線名稱。如果已經使用指定的名稱完成連線，便會產生 ArgumentError 錯誤，指出連線嘗試失敗，因為該物件已經連線。

下列程式碼片段將示範如何建立名為 conn1 的 LocalConnection：

```
try
{
    connection.connect("conn1");
}
catch (error:ArgumentError)
{
    trace("Error! Server already exists\n");
}
```

如果要從次要應用程式連線至主要應用程式，您必須先在傳送端 LocalConnection 物件中建立 LocalConnection 物件，然後呼叫 LocalConnection.send() 方法，並指定連線名稱和方法名稱以便執行。例如，若要將 doQuit 方法傳送到您先前建立的 LocalConnection 物件，請使用下列程式碼：

```
sendingConnection.send("conn1", "doQuit");
```

這段程式碼會使用連線名稱 conn1 連線到現有的 LocalConnection 物件，並叫用遠端應用程式中的 doMessage() 方法。如果想將參數傳送至遠端應用程式，請在 send() 方法中的方法名稱之後指定其它引數，如下列程式碼片段所示：

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

連線至不同網域和 AIR 應用程式中的內容

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如果只想允許從特定網域中進行通訊，您可以呼叫 LocalConnection 類別的 allowDomain() 或 allowInsecureDomain() 方法，並傳遞一個或多個可以存取這個 LocalConnection 物件的網域清單，傳遞一個或多個要允許的網域名稱。

在舊版 ActionScript 中，LocalConnection.allowDomain() 和 LocalConnection.allowInsecureDomain() 都是回呼方法，必須由開發人員實作並且必須傳回 Boolean 值；而在 ActionScript 3.0 中，LocalConnection.allowDomain() 和 LocalConnection.allowInsecureDomain() 都是內建的方法，就和 Security.allowDomain() 和 Security.allowInsecureDomain() 一樣，開發人員可以呼叫並傳遞一個或多個要允許的網域名稱。

Flash Player 8 在本機 SWF 檔上新增了安全性限制。允許存取網際網路的 SWF 檔無法同時存取本機檔案系統。如果您指定 localhost，則任何本機 SWF 檔都可以存取這個 SWF 檔。如果 LocalConnection.send() 方法嘗試與安全執行程序中的 SWF 檔進行通訊，但是呼叫端程式碼卻無法存取此安全執行程序，則會傳送 securityError 事件 (SecurityErrorEvent.SECURITY_ERROR)。若要排除這項錯誤，您可以在接收者的 LocalConnection.allowDomain() 方法中指定呼叫者的網域。

有兩個特殊值可以傳遞給 LocalConnection.allowDomain() 和 LocalConnection.allowInsecureDomain() 方法：* 和 localhost。星號值 (*) 允許從所有網域存取。字串 localhost 允許從本機安裝但在應用程式資源目錄之外的內容，呼叫應用程式。

如果 LocalConnection.send() 方法嘗試與安全執行程序中的應用程式進行通訊，但是呼叫端程式碼卻無法存取此安全執行程序，則會傳送 securityError 事件 (SecurityErrorEvent.SECURITY_ERROR)。若要排除這項錯誤，您可以在接收者的 LocalConnection.allowDomain() 方法中指定呼叫者的網域。

如果您只要在相同網域中的內容之間實作通訊，可以指定 connectionName 參數，這個參數不能以底線 (_) 做為開頭，也不指定網域名稱 (例如，myDomain:connectionName)。請在 LocalConnection.connect(connectionName) 命令中使用相同的字串。

如果您要在不同網域的內容之間實作通訊，請指定以底線做為開頭的 connectionName 參數。指定底線可以讓具有接收端 LocalConnection 物件的內容更容易在不同的網域之間使用。此處列出兩種可能的案例：

- 如果 connectionName 字串不以底線做為開頭，執行階段會加上具有父網域名稱和冒號 (例如，myDomain:connectionName) 的前置詞。雖然這樣能確保您的連線不會與來自其它網域的相同名稱連線發生衝突，但是任何傳送端的 LocalConnection 物件仍然必須指定此父網域 (例如，myDomain:connectionName)。如果您將包含接收端 LocalConnection 物件的 HTML 或 SWF 檔移到其它網域，則執行階段會變更前置詞來反映新的父網域 (例如，anotherDomain:connectionName)。您必須手動編輯所有傳送端的 LocalConnection 物件，以指向新的父網域。
- 如果 connectionName 的字串以底線為開頭 (例如，_connectionName)，執行階段就不會將前置詞加入此字串。也就是說，接收端與傳送端的 LocalConnection 物件將使用相同的 connectionName 字串。如果接收端物件使用 LocalConnection.allowDomain() 以指定接受來自任何網域的連線，您可以在不更動任何傳送端 LocalConnection 物件的情況下，將含有接收端 LocalConnection 物件的 SWF 檔移到其它網域。

  在 connectionName 中使用底線的缺點是可能會發生衝突，例如當兩個應用程式同時嘗試使用相同的 connectionName 連線時。第二個的缺點則與安全性相關。使用底線語法的連線名稱不會識別偵聽應用程式的網域。基於上述理由，一般偏好使用完整的網域名稱。

Adobe AIR

若要與 AIR 應用程式安全執行程序 (使用 AIR 應用程式安裝的內容) 中執行的內容進行通訊，必須在連線名稱加上識別 AIR 應用程式的父網域做為前置詞。父網域字串的開頭為 app#，後面接著應用程式 ID 再加上點 (.) 字元，最後緊接發行者 ID (如果有定義的話)。例如，在擁有應用程式 ID com.example.air.MyApp 且無發行者 ID 的應用程式中，其 connectionName 參數所使用的正確父網域為："app#com.example.air.MyApp"。因此，如果基礎連線名稱為 "appConnection"，則 connectionName 參數所使用的完整字串為："app#com.example.air.MyApp:appConnection"。如果應用程式具有發行者 ID，則該 ID 也必須包含在父網域字串中："app#com.example.air.MyApp.B146A943FBD637B68C334022D304CEA226D129B4.1"。

當您允許其它 AIR 應用程式透過本機連線與您的應用程式通訊時，必須呼叫 LocalConnection 物件的 allowDomain()，傳入本機連線網域名稱。對於 AIR 應用程式來說，這個網域名稱是由應用程式和發行者 ID 所產生，與連線字串的方式相同。例如，如果傳送端 AIR 應用程式具有應用程式 ID com.example.air.FriendlyApp 和發行者 ID 214649436BD677B62C33D02233043EA236D13934.1，則您要用來允許這個應用程式連線的網域字串是：app#com.example.air.FriendlyApp.214649436BD677B62C33D02233043EA236D13934.1。(從 AIR 1.5.3 開始，並非所有 AIR 應用程式都有發行者 ID。)

第 46 章 與 AIR 中的原生程序通訊

Adobe AIR 2 以及更新的版本

從 Adobe AIR 2 開始，AIR 應用程式可以透過命令列與其他原生程序一起執行和相互通訊。例如，AIR 應用程式只要透過標準輸入與輸出串流，就可以執行程序並進行通訊。

若要與原生程序進行通訊，請封裝 AIR 應用程式，以透過原生安裝程式進行安裝。原生安裝程式的檔案類型，只適用於建立它的特定作業系統：

- 在 Mac OS 上，是 DMG 檔案。
- 在 Windows 上，是 EXE 檔案。
- 在 Linux 上，是 RPM 或 DEB 套件。

這些應用程式稱為擴充桌面設定檔應用程式。您可以使用 ADT 呼叫 -package 命令並指定 -target native 選項，如此便能建立原生安裝程式。

更多說明主題

flash.filesystem.File.openWithDefaultApplication()

flash.desktop.NativeProcess

原生程序通訊概觀

Adobe AIR 2 以及更新的版本

擴充桌面設定檔中的 AIR 應用程式可以執行檔案，就像是透過命令列叫用一樣。它可以與原生程序的標準串流進行通訊。標準串流包括標準輸入串流 (stdin)、輸出串流 (stdout) 以及標準錯誤串流 (stderr)。

備註：擴充桌面設定檔中的應用程式也可以使用 File.openWithDefaultApplication() 方法來啟動檔案和應用程式。不過，使用這種方法並不無法讓 AIR 應用程式存取標準串流。如需詳細資訊，請參閱第 586 頁 「使用預設系統應用程式開啟檔案」

以下程式碼樣本顯示如何啟動應用程式目錄中的 test.exe 應用程式。應用程式會傳遞 "hello" 引數做為命令列引數，然後將事件偵聽程式加到這個處理的標準輸出串流中：

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
var processArgs:Vector.<String> = new Vector.<String>();
processArgs.push("hello");
nativeProcessStartupInfo.arguments = processArgs;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_OUTPUT_DATA, onOutputData);
process.start(nativeProcessStartupInfo);
public function onOutputData(event:ProgressEvent):void
{
    var stdOut:ByteArray = process.standardOutput;
    var data:String = stdOut.readUTFBytes(process.standardOutput.bytesAvailable);
    trace("Got: ", data);
}
```

啟動以及關閉原生程序

Adobe AIR 2 以及更新的版本

若要啟動原生程序，請設定 NativeProcessInfo 物件執行以下動作：

- 指向您想啟動的檔案
- 儲存當程序啟動時要傳遞的命令列引數 (選擇性)
- 設定程序的工作目錄 (選擇性)

若要啟動原生程序，請傳遞 NativeProcessInfo 物件，讓 NativeProcess 物件的 start() 方法將這個物件當做參數使用。

例如，以下程式碼範例顯示如何啟用應用程式目錄中的 test.exe 應用程式。應用程式會傳遞引數 "hello" 並將使用者的文件目錄設定為工作目錄：

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
var processArgs:Vector.<String> = new Vector.<String>();
processArgs[0] = "hello";
nativeProcessStartupInfo.arguments = processArgs;
nativeProcessStartupInfo.workingDirectory = File.documentsDirectory;
process = new NativeProcess();
process.start(nativeProcessStartupInfo);
```

若要終止程序，請呼叫 NativeProcess 物件的 exit() 方法。

如果您想要檔案可以在安裝的應用程式中執行，請在封裝應用程式時，確定該檔案可以在檔案系統上執行 (您可以在 Mac 和 Linux 上，視需要使用 chmod 設定可執行旗標)。

與原生程序通訊

Adobe AIR 2 以及更新的版本

AIR 應用程式啟動原生程序之後，就可以與程序的標準輸入、標準輸出以及標準錯誤串流通訊。

您可以使用 NativeProcess 物件的下列屬性，從串流讀取資料以及將資料寫入串流：

- standardInput— 包含標準輸入串流資料的存取權。
- standardOutput— 包含標準輸出串流資料的存取權。
- standardError— 包含標準錯誤串流資料的存取權。

寫入標準輸入串流

您可以使用 NativeProcess 物件之 standardInput 屬性的 write 方法，將資料寫入標準輸入串流。當 AIR 應用程式將資料寫入程序時，NativeProcess 物件會傳送 standardInputProgress 事件。

如果在寫入標準輸入串流時發生錯誤，NativeProcess 物件會傳送 ioErrorStandardInput 事件。

呼叫 NativeProcess 物件的 closeInput() 方法，就可以關閉輸入串流。當輸入串流關閉後，NativeProcess 物件會傳送一個 standardInputClose 事件。

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.start(nativeProcessStartupInfo);
process.standardInput.writeUTF("foo");
if(process.running)
{
    process.closeInput();
}
```

從標準輸出串流讀取

使用這個屬性的 read 方法，就可以從標準輸出串流讀取資料。當 AIR 應用程式從程序取得輸出串流資料後，NativeProcess 物件會傳送 standardOutputData 事件。

若寫入標準輸出串流時發生錯誤，NativeProcess 物件會傳送一個 standardOutputError 事件。

程序關閉輸出串流之後，NativeProcess 物件會傳送一個 standardOutputClose 事件。

讀取標準輸入串流中的資料時，請確認在資料產生時就進行讀取。換言之，即新增 standardOutputData 事件的事件偵聽程式。在 standardOutputData 事件偵聽程式中，讀取 NativeProcess 物件之 standardOutput 屬性的資料。而不要只是等待 standardOutputClose 事件或 exit 事件，才讀取所有資料。如果您未在原生程序產生資料時讀取資料，則緩衝區可能會填滿，資料可能會因而遺失。緩衝區填滿後，如果嘗試寫入更多資料，會導致原生程序變得遲緩。不過，如果您未註冊 standardOutputData 事件的事件偵聽程式，則緩衝區不會填滿，程序也不會變得遲緩。在此情況下，您無法存取資料。

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_OUTPUT_DATA, dataHandler);
process.start(nativeProcessStartupInfo);
var bytes:ByteArray = new ByteArray();
function dataHandler(event:ProgressEvent):void
{
    bytes.writeBytes(process.standardOutput.readBytes(process.standardOutput.bytesAvailable);
}
```

從標準錯誤串流讀取

使用這個屬性的 read 方法，就可以從標準錯誤串流讀取資料。當 AIR 應用程式從程序讀取錯誤串流資料後，NativeProcess 物件會傳送 standardErrorData 事件。

若寫入標準錯誤串流時發生錯誤，NativeProcess 物件會傳送一個 standardErrorIoError 事件。

程序關閉錯誤串流後，NativeProcess 物件會傳送 standardErrorClose 事件。

讀取標準錯誤串流中的資料時，請確認在資料產生時就進行讀取。換言之，即新增 standardErrorData 事件的事件偵聽程式。在 standardErrorData 事件偵聽程式中，讀取 NativeProcess 物件之 standardError 屬性的資料。而不要只是等待 standardErrorClose 事件或 exit 事件，才讀取所有資料。如果您未在原生程序產生資料時讀取資料，則緩衝區可能會填滿，資料可能會因而遺失。緩衝區填滿後，如果嘗試寫入更多資料，會導致原生程序變得遲緩。不過，如果您未註冊 standardErrorData 事件的事件偵聽程式，則緩衝區不會填滿，程序也不會變得遲緩。在此情況下，您無法存取資料。

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_ERROR_DATA, errorDataHandler);
process.start(nativeProcessStartupInfo);
var errorBytes:ByteArray = new ByteArray();
function errorDataHandler(event:ProgressEvent):void
{
    bytes.writeBytes(process.standardError.readBytes(process.standardError.bytesAvailable);
}
```

原生程序通訊安全性注意事項

Adobe AIR 2 以及更新的版本

原生處理程序 API 可在使用者系統上執行任何可執行檔。建構和執行命令時要特別小心。如果要執行命令的任何部分源自外部來源，請小心驗證以確保可安全執行此命令。同樣地，您的 AIR 應用程式應該要驗證傳遞到執行中處理程序的任何資料。

然而，驗證輸入可能會發生問題。若要避免這類問題，最好寫入有特定 API 的原生應用程式 (例如 Windows 的 EXE 檔)。這些 API 應該只能處理應用程式明確要求的命令。例如，應用程式只能透過標準輸入串流接受一組有限的指示。

Windows 的 AIR 不允許您直接執行 .bat 檔案。命令解譯器應用程式 (cmd.exe) 會執行 Windows .bat 檔案。當您叫用 .bat 檔案時，此命令應用程式可將傳遞到命令的引數解譯為其他要啟動的應用程式。在引數字串中惡意插入多餘的字元，可能會讓 cmd.exe 執行有害或不安全的應用程式。例如，在無適當資料驗證的情況下，您的 AIR 應用程式可能會呼叫 myBat.bat myArguments c:/evil.exe。命令應用程式除了執行您的批次檔外，還會啟動 evil.exe 應用程式。

第 47 章 使用外部 API

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 外部 API (flash.external.ExternalInterface) 讓 ActionScript 與執行 Adobe Flash Player 的容器應用程式之間可以直接通訊。使用 ExternalInterface API 即可在 SWF 文件與 HTML 頁面的 JavaScript 之間建立互動。

您可以使用外部 API 與容器應用程式互動、在 ActionScript 與 HTML 頁面的 JavaScript 之間傳遞資料。

部分常見外部 API 工作如下：

- 取得容器應用程式的相關資訊
- 使用 ActionScript 呼叫瀏覽器所顯示網頁中的程式碼或 AIR 桌面應用程式
- 從網頁呼叫 ActionScript 程式碼
- 建立 Proxy 以簡化從網頁呼叫 ActionScript 程式碼的程序

備註：外部介面的這項討論只涵蓋下列兩者之間的通訊：SWF 檔案中的 ActionScript 與參考 Flash Player 或載入 SWF 檔案之實體的容器應用程式。其它有關在應用程式內使用 Flash Player 的主題則不在本文件的討論範圍。Flash Player 的設計目的是要當做瀏覽器外掛程式或放映檔 (獨立應用程式) 來使用。如果在其它情況下使用，提供的支援可能很有限。

在 **AIR** 中使用外部 **API**

因為 AIR 應用程式沒有外部容器，所以這個外部介面一般不適用 (通常也不需要)。AIR 應用程式直接載入 SWF 檔案時，應用程式程式碼可以直接與 SWF 中的 ActionScript 程式碼通訊 (受限於安全執行程序限制)。

不過，AIR 應用程式在 HTMLLoader 物件中使用 HTML 網頁載入 SWF 檔案 (或 Flex 中的 HTML 元件) 時，HTMLLoader 物件是做為外部容器。因此，您可以使用外部介面在下列兩者之間進行通訊：已載入 SWF 中的 ActionScript 程式碼與 HTMLLoader 中載入之 HTML 網頁中的 JavaScript 程式碼。

使用外部 API 的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

雖然在某些情況下 SWF 檔可以獨立執行 (例如，若您使用 Adobe® Flash® Professional 建立 SWF 放映檔時)，但是在大部分情況下，SWF 應用程式必須當做另一個應用程式內部的元素來執行。一般來說，內含 SWF 的容器會是一個 HTML 檔；在某些比較少見的情況中，SWF 檔則是當做桌面應用程式的全部或部分使用者介面來使用。

隨著您所使用的應用程式越來越進階，就會越需要在 SWF 檔與容器應用程式之間建立通訊模式。例如，讓 HTML 網頁顯示文字或其它資訊，並加入 SWF 檔以顯示圖表或視訊之類的動態視覺內容，都已經是很普遍的作法。在這種情況下，您可能會希望讓使用者按下網頁上按鈕，就能變更 SWF 檔中的某些部分。ActionScript 包含了一種稱為外部 API 的機制，可在 SWF 檔的 ActionScript 與容器應用程式的其它程式碼之間協助建立起這種通訊類型。

重要概念與術語

下列參考清單包含了與此功能相關的重要術語：

容器應用程式 Flash Player 執行諸如網頁和 HTML 頁面等 SWF 檔案所在的應用程式，其中包含 Flash Player 內容或可在網頁中載入 SWF 的 AIR 應用程式。

放映檔 包含 SWF 內容與 Flash Player 內嵌版本的可執行檔。您可使用 Flash Professional 或獨立 Flash Player 建立放映檔。我們通常會在光碟中加上放映檔來散發 SWF 檔；或是當下載檔案大小不成問題，而 SWF 作者希望確定使用者不管其電腦是否已經安裝了 Flash Player，都能執行 SWF 檔時，就會使用放映檔。

Proxy 媒介應用程式或程式碼，可用來代替某個應用程式 (「呼叫端應用程式」) 呼叫另一個應用程式 (「外部應用程式」) 中的程式碼，而且會將值傳回呼叫端應用程式。下列是您應該使用 Proxy 的幾項理由：

- 藉由在呼叫端應用程式中將原生函數呼叫轉換為外部應用程式能夠理解的格式，簡化執行外部函數呼叫的程序 .
- 用來解決一些導致呼叫者無法與外部應用程式直接通訊的安全性與其它限制問題。

序列化 將物件或資料值轉換為可在兩種程式設計系統 (例如在網際網路中，或單一電腦上執行的兩種不同的應用程式) 的訊息之間傳遞值的特定格式。

逐步執行範例

這裡所提供的許多程式碼範例都是一小段的程式碼，僅供示範之用，並不是檢查數值的完整實用範例或程式碼。由於使用外部 API 需要 (依定義) 撰寫 ActionScript 程式碼以及容器應用程式中的程式碼，因此測試範例的步驟會包括建立容器 (例如，包含 SWF 檔的網頁) 和使用列出的程式碼與容器進行互動。

若要測試 **ActionScript** 與 **JavaScript** 通訊的範例：

1. 使用 Flash Professional 編寫新文件，然後在電腦中儲存此文件。
2. 從主選單中選擇 「檔案 > 發佈設定」。
3. 在 「發佈設定」對話方塊中，確認 「格式」索引標籤上只選取了 「HTML」和 「Flash」核取方塊。
4. 按一下 「發佈」按鈕。這樣便會在同一個檔案夾中產生 SWF 檔和 HTML 檔，它們的名稱和您用來儲存文件的名稱相同。按一下 「確定」，關閉 「發佈設定」對話方塊。
5. 取消選取 HTML 核取方塊。現在既然產生了 HTML 網頁，便可以著手修改它，加入適當的 JavaScript 程式碼。取消選取 HTML 核取方塊可以確保修改 HTML 網頁之後，Flash 不會在發佈 SWF 檔時，以新的 HTML 網頁覆寫您的變更。
6. 按一下 「確定」，關閉 「發佈設定」對話方塊。
7. 使用 HTML 或文字編輯應用程式，開啟 Flash 在發佈 SWF 檔時所建立的 HTML 檔案。在 HTML 原始碼中，加入開頭和結束 script 標籤，然後從列出的程式碼範例中，將 JavaScript 程式碼複製到這兩個標籤之間。

```
<script>
// add the sample JavaScript code here
</script>
```

8. 儲存 HTML 檔並返回 Flash。
9. 在 「時間軸」的 「影格 1」上選取關鍵影格，並開啟 「動作」面板。
10. 將列出的 ActionScript 程式碼複製到 「Script」窗格中。
11. 從主選單中選擇 「檔案 > 發佈」，以您所做的變更來更新 SWF 檔。
12. 使用網頁瀏覽器，開啟您所編輯的 HTML 網頁加以檢視，並測試 ActionScript 與此 HTML 網頁之間的通訊。

若要測試 **ActionScript** 與 **ActiveX** 容器通訊的範例：

1. 使用 Flash Professional 編寫新文件，然後在電腦中儲存此文件。您可能想要將它儲存在容器應用程式預期可找到 SWF 檔的檔案夾中。
2. 從主選單中選擇 「檔案 > 發佈設定」。
3. 在 「發佈設定」對話方塊中，確認 「格式」索引標籤上只選取了 「Flash」核取方塊。
4. 在 「Flash」核取方塊旁的 「檔案」欄位中，按一下檔案夾圖示以選取 SWF 檔的發佈目的地檔案夾。藉由設定 SWF 檔的位置，您可以 (舉例來說) 將文件保留在某個資料夾中，而將已發佈的 SWF 檔放在另一個資料夾，例如包含容器應用程式原始碼的資料夾。

5 在 「時間軸」的 「影格 1」上選取關鍵影格，並開啟 「動作」面板。

6 將範例的 ActionScript 程式碼複製到 「Script」窗格中。

7 從主選單中選擇 「檔案 > 發佈」以重新發佈 SWF 檔。

8 建立並執行容器應用程式，以測試 ActionScript 與容器應用程式之間的通訊。

如需使用外部 API 與 HTML 頁面進行通訊的完整範例，請參閱下列主題：

- 第 731 頁 「外部 API 範例：使用瀏覽器在 ActionScript 和 JavaScript 之間通訊」

這些範例包含完整的程式碼 (包括 ActionScript 和容器錯誤檢查程式碼)，您在使用外部 API 撰寫程式碼時應該使用這些程式碼。如需其它使用外部 API 的完整範例，請參閱 「Adobe® Flash® Professional CS5 的 ActionScript® 3.0 參考」中的 ExternalInterface 類別範例。

外部 API 需求與優點

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

外部 API 屬於 ActionScript 的一部分，可提供一項機制讓 ActionScript 與在 「外部應用程式」中執行的程式碼進行通訊，這個外部應用程式也就是 Flash Player 的容器 (通常是網頁瀏覽器或獨立的放映檔應用程式)。在 ActionScript 3.0 中，是由 ExternalInterface 類別提供外部 API 的功能；而在 Flash Player 8 之前的 Flash Player 版本中，則是由 fscommand() 動作與容器應用程式進行通訊。ExternalInterface 類別取代了 fscommand()。

備註：如果您需要使用舊版的 fscommand() 函數 (例如，為了維持與早期應用程式的相容性，或是與第三方 SWF 容器應用程式或獨立的 Flash Player 維持互動)，您仍然可以在 flash.system 套件中找到這個套件等級函數。

ExternalInterface 類別是一個子系統，可讓您輕鬆地在 ActionScript 和 Flash Player 與 HTML 頁面中的 JavaScript 之間進行通訊。

ExternalInterface 類別僅在下列情況下才能使用：

- Windows 的 Internet Explorer 的所有支援版本 (5.0 和更新版本)
- 所有支援 NPRuntime 介面的瀏覽器，目前這些瀏覽器包含 Firefox 1.0 和更新版本、Mozilla 1.7.5 和更新版本、Netscape 8.0 和更新版本，以及 Safari 1.3 和更新版本。
- SWF 內嵌在由 HTMLLoader 控制項顯示的 HTML 頁面時的 AIR 應用程式。

在其它情況下 (例如在獨立的播放程式中執行時)，ExternalInterface.available 屬性就會傳回 false。

您可以從 ActionScript 呼叫 HTML 網頁上的 JavaScript 函數。和 fscommand() 相比，外部 API 可提供下列更完備的功能：

- 您可以使用任何 JavaScript 函數，而不只是使用能與 fscommand() 函數搭配使用的函數。
- 您可以傳遞任意名稱且任意數目的引數，不必受限於只能傳遞一個命令和單一字串引數。和 fscommand() 相較之下，外部 API 更具彈性。
- 您可以傳遞不同的資料類型 (例如 Boolean、Number 和 String)，而不受限於只能傳遞 String 類型的參數。
- 您可以接收某個呼叫的值，並讓這個值立即回傳到 ActionScript (當您進行該呼叫時)。

重要事項: 如果 HTML 網頁中指定的 Flash Player 實體名稱 (object 標籤的 id 特質) 含有連字符號 (-) 或其它已在 JavaScript 中定義為運算子的字元 (例如 +、*、/、\、. 等字元)，當您在 Internet Explorer 中檢視容器網頁時，使用 ActionScript 進行的 ExternalInterface 呼叫將無法運作。此外，如果定義 Flash Player 實體的 HTML 標籤 (object 和 embed 標籤) 是在 HTML form 標籤的巢狀架構內，使用 ActionScript 進行的 ExternalInterface 呼叫也無法運作。

使用 ExternalInterface 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 與容器應用程式之間的通訊有兩種形式：ActionScript 可以呼叫在容器中定義的程式碼 (如 JavaScript 函數)，或是容器中的程式碼可以呼叫已經指定為可以呼叫的 ActionScript 函數。不論是那一種通訊形式，都可以將資訊送達遭到呼叫的程式碼，而且可以將結果傳回給執行呼叫的程式碼。

ExternalInterface 類別中的兩個靜態屬性和兩個靜態方法可以讓通訊更加便利。這些屬性和方法是用來取得外部介面連線的相關資訊、透過 ActionScript 執行容器中的程式碼，以及讓容器可以呼叫 ActionScript 函數。

取得外部容器的相關資訊

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ExternalInterface.available 屬性會指出目前的 Flash Player 是否在提供外部介面的容器內。如果可使用外部介面，則此屬性為 true；否則為 false。在使用 ExternalInterface 類別提供的其它功能之前，請務必檢查以確定目前的容器可支援外部介面通訊，如下所示：

```
if (ExternalInterface.available)
{
    // Perform ExternalInterface method calls here.
}
```

備註：ExternalInterface.available 屬性會回報目前的容器類型是否支援 ExternalInterface 連線，但不會告訴您目前的瀏覽器是否已啟用 JavaScript。

ExternalInterface.objectID 屬性可讓您判斷 Flash Player 實體的唯一識別名稱 (特別是 Internet Explorer 中 object 標籤的 id 特質，或使用 NPRuntime 介面之瀏覽器中 embed 標籤的 name 特質)。這個唯一 ID 表示瀏覽器中目前的 SWF 文件，而且可以用來建立 SWF 文件的參考，例如，在呼叫容器 HTML 網頁中的 JavaScript 函數時便可以使用。當 Flash Player 容器不是網頁瀏覽器時，此屬性則為 null。

透過 ActionScript 呼叫外部程式碼

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ExternalInterface.call() 方法可以執行容器應用程式內的程式碼。它至少需要使用一個字串參數，此字串包含容器應用程式內所要呼叫的函數名稱。傳遞給 ExternalInterface.call() 方法的任何其它參數也會一併傳遞給容器，做為函數呼叫的參數。

```
// calls the external function "addNumbers"
// passing two parameters, and assigning that function's result
// to the variable "result"
var param1:uint = 3;
var param2:uint = 7;
var result:uint = ExternalInterface.call("addNumbers", param1, param2);
```

如果容器是 HTML 網頁，這個方法就會叫用具有指定名稱的 JavaScript 函數，這個名稱必須已經在此 HTML 容納網頁的 script 元素中加以定義。此 JavaScript 函數的傳回值將回傳給 ActionScript。

```
<script language="JavaScript">
    // adds two numbers, and sends the result back to ActionScript
    function addNumbers(num1, num2)
    {
        return (num1 + num2);
    }
</script>
```

如果此容器是其它 ActiveX 容器，這個方法就會讓 Flash Player 的 ActiveX 控制項傳送它的 FlashCall 事件。指定的函數名稱以及任何參數都將由 Flash Player 序列化成一個 XML 字串。此容器可以存取事件物件之 request 屬性中的資訊，並用它來決定如何執行它自己的程式碼。因為要將值傳回 ActionScript，容器程式碼會呼叫 ActiveX 物件的 SetReturnValue() 方法，傳遞結果 (已序列化成 XML 字串) 做為這個方法的參數。如需有關這類通訊所使用之 XML 格式的詳細資訊，請參閱第 730 頁「外部 API 的 XML 格式」。

不論此容器是網頁瀏覽器或其它 ActiveX 容器，當呼叫失敗或容器方法沒有指定傳回值時，就會傳回 null。如果容納環境屬於呼叫程式碼無法存取的系統安全執行程序，ExternalInterface.call() 方法也會擲回 SecurityError 例外。您只要在容納環境中為 allowScriptAccess 設定適當的值，就可以解決這個問題。例如，若要變更 HTML 網頁中 allowScriptAccess 的值，您可以編輯 object 和 embed 標籤中的適當特質。

透過容器呼叫 ActionScript 程式碼

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

容器只能呼叫函數中的 ActionScript 程式碼，而不能呼叫其它的 ActionScript 程式碼。如果要透過容器應用程式呼叫 ActionScript 函數，您必須執行兩項作業：使用 ExternalInterface 類別註冊函數，然後透過容器的程式碼呼叫它。

首先，您必須註冊 ActionScript 函數以表示它已經可以供容器進行呼叫。使用 ExternalInterface.addCallback() 方法，如下所示：

```
function callMe(name:String):String
{
    return "busy signal";
}
ExternalInterface.addCallback("myFunction", callMe);
```

addCallback() 方法會採用兩個參數。第一個參數是類型為 String 的函數名稱，讓容器知道這個函數的名稱。第二個參數是實際的 ActionScript 函數，會在容器呼叫已定義的函數名稱時執行。因為這些名稱很容易分辨，即使實際的 ActionScript 函數具有不同的名稱，您也可以指定將由容器使用的函數名稱。這個功能在不知道函數名稱時 (例如指定匿名函數，或者要呼叫的函數是在執行階段才會決定) 特別有用。

在使用 ExternalInterface 類別註冊 ActionScript 函數後，容器實際上就可以呼叫這個函數。執行這項作業的方式依容器的類型而定。例如，在網頁瀏覽器的 JavaScript 程式碼中，會將註冊的函數名稱當做是 Flash Player 瀏覽器物件的方法來呼叫 ActionScript 函數 (也就是說，做為 JavaScript 物件的方法，用來表示 object 或 embed 標籤)。換句話說，就像呼叫本機函數一樣地傳遞參數及傳回結果。

```
<script language="JavaScript">
    // callResult gets the value "busy signal"
    var callResult = flashObject.myFunction("my name");
</script>
...
<object id="flashObject"...>
    ...
    <embed name="flashObject".../>
</object>
```

此外，在呼叫執行於桌面應用程式之 SWF 檔中的 ActionScript 函數時，註冊的函數名稱和任何參數也必須序列化成 XML 格式化字串。之後實際的呼叫作業是使用 XML 字串做為參數，由 ActiveX 控制項的 CallFunction() 方法執行呼叫。如需有關這類通訊所使用之 XML 格式的詳細資訊，請參閱第 730 頁「外部 API 的 XML 格式」。

不論是使用那一種方式，ActionScript 函數的傳回值都會回傳給容器程式碼。如果呼叫者是瀏覽器中的 JavaScript 程式碼，就會直接傳回值；如果呼叫者是 ActiveX 容器，則將值序列化成 XML 格式化字串。

外部 API 的 XML 格式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 與裝載 Shockwave Flash ActiveX 控制項的應用程式之間的通訊是使用特殊的 XML 格式將函數呼叫和值進行編碼。外部 API 使用的 XML 格式分為兩個部分。其中一部分的格式是用來表示函數呼叫，另一部分格式則是用來表示個別的值，這個格式是用於函數中的參數以及函數傳回值。這些函數呼叫的 XML 格式是供 ActionScript 所傳送及接收的呼叫使用。對於來自 ActionScript 的函數呼叫，Flash Player 會將 XML 傳遞給容器；而對於來自容器的呼叫，Flash Player 也會預期容器應用程式將會以這種格式來傳遞 XML 字串。下列 XML 片段將顯示經過 XML 格式化的函數呼叫範例：

```
<invoke name="functionName" returntype="xml">
    <arguments>
        ... (individual argument values)
    </arguments>
</invoke>
```

根節點是 invoke 節點。它具有兩個特質：name 表示要呼叫的函數名稱，而 returntype 則永遠為 xml。如果函數呼叫中含有參數，invoke 節點便具有子 arguments 節點，它的子節點會是使用個別值格式 (接下來將會說明) 加以格式化的參數值。

個別值 (包括函數參數和函數傳回值) 會使用包括實際值以外之資料類型資訊的格式化配置。下表將列出 ActionScript 類別以及用來針對該資料類型值進行編碼的 XML 格式：

ActionScript 類別 / 值	**C#** 類別 / 值	格式	註解
null	null	<null/>	
Boolean true	bool true	<true/>	
Boolean false	bool false	<false/>	
String	string	<string> 字串值 </string>	
Number、int、uint	single、double、int、uint	`<number>27.5</number>` `<number>-12</number>`	

Array (可包含各種元素類型)	可包含各種元素類型 (例如 ArrayList 或 object[]) 的集合。	`<array>`
		`<property id="0">`
		`<number>27.5</number>`
		`</property>`
		`<property id="1">`
		`<string>Hello there!</string>`
		`</property>`
		`...`
		`</array>`
			property 節點會定義個別元素，而 id 特質則是從零開始的數值索引。

Object	包含字串索引鍵和物件值的字典，例如包含字串索引鍵的 HashTable	`<object>`
		`<property id="name">`
		`<string>John Doe</string>`
		`</property>`
		`<property id="age">`
		`<string>33</string>`
		`</property>`
		`...`
		`</object>`
			property 節點會定義個別屬性，而 id 特質則是屬性名稱 (字串)。

其它內建或自訂類別		`<null/> or` `<object></object>`	ActionScript 會將其它物件編碼成 null 或是空白物件。不論是那種方式，都將遺失所有的屬性值。

備註：本表藉由範例說明與 ActionScript 類別具備同等功能的 C# 類別；然而，可用來與其它程式設計語言進行通訊的外部 API 或是支援 ActiveX 控制項的執行階段，卻不限於 C# 應用程式。

當您使用包含 ActiveX 容器應用程式的外部 API 來建立自己的應用程式時，可能會發現直接撰寫一個 Proxy 以便將原生函數呼叫轉換為序列化 XML 格式會很方便。如需以 C# 撰寫的 Proxy 類別範例，請參閱透視 ExternalInterfaceProxy 類別。

外部 API 範例：使用瀏覽器在 ActionScript 和 JavaScript 之間通訊

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

這個樣本應用程式將示範在允許與他人交談的即時傳訊應用程式環境中 (應用程式的名稱是 Introvert IM)，於網頁瀏覽器內的 ActionScript 與 JavaScript 之間進行通訊的正確技巧。所有訊息都是使用外部 API，在網頁的 HTML 表單與 SWF 介面之間傳送。這個範例將示範的技巧包括下列各項：

- 在設定通訊之前先確認瀏覽器已經準備開始通訊，以便正確地起始通訊
- 檢查容器是否支援外部 API
- 透過 ActionScript 呼叫 JavaScript 函數、傳遞參數以及接收回應值
- 讓 JavaScript 可以呼叫 ActionScript 方法，並執行這些呼叫

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。您可以在 Samples/IntrovertIM_HTML 檔案夾中找到 Introvert IM 應用程式檔案，此應用程式是由下列檔案組成：

檔案	說明
IntrovertIMApp.fla 或 IntrovertIMApp.mxml	主應用程式檔案，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/introvertIM/IMManager.as	類別，建立與管理 ActionScript 和容器之間的通訊。
com/example/programmingas3/introvertIM/IMMessageEvent.as	自訂的事件類型，從容器接收訊息後，由 IMManager 類別將它送出。
com/example/programmingas3/introvertIM/IMStatus.as	列舉項目，它的值表示可以在應用程式中選取的各種 「可用性」狀態值。
html-flash/IntrovertIMApp.html 或 html-template/index.template.html	Flash 應用程式的 HTML 網頁 (html-flash/IntrovertIMApp.html) 或是用來建立 Adobe Flex 應用程式的容器 HTML 網頁的範本 (html-template/index.template.html)。這個檔案包含所有構成應用程式容器部分的 JavaScript 函數。

準備開始在 ActionScript 與瀏覽器之間進行通訊

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

外部 API 最常見的用途之一，便是允許 ActionScript 應用程式與網頁瀏覽器進行通訊。利用外部 API，ActionScript 方法可以呼叫以 JavaScript 撰寫的程式碼，反之亦然。由於瀏覽器的複雜性以及在內部呈現頁面的不同方式，因此在執行 HTML 網頁的第一個 JavaScript 之前，並無法保證 SWF 文件可以註冊它的回呼。所以，透過 JavaScript 呼叫 SWF 文件中的函數之前，您的 SWF 文件永遠都要先呼叫 HTML 網頁，通知它 SWF 文件已經準備開始接收連線。

例如，透過由 IMManager 類別執行的一系列步驟，Introvert IM 就可以判斷瀏覽器是否已經準備開始進行通訊並準備通訊用的 SWF 檔。第一個步驟是判斷瀏覽器準備開始進行通訊的時間，此步驟會在 IMManager 建構函式中執行，如下所示：

```
public function IMManager(initialStatus:IMStatus)
{
    _status = initialStatus;

    // Check if the container is able to use the external API.
    if (ExternalInterface.available)
    {
        try
        {
            // This calls the isContainerReady() method, which in turn calls
            // the container to see if Flash Player has loaded and the container
            // is ready to receive calls from the SWF.
            var containerReady:Boolean = isContainerReady();
            if (containerReady)
            {
                // If the container is ready, register the SWF's functions.
                setupCallbacks();
            }
            else
            {
                // If the container is not ready, set up a Timer to call the
                // container at 100ms intervals. Once the container responds that
                // it's ready, the timer will be stopped.
                var readyTimer:Timer = new Timer(100);
                readyTimer.addEventListener(TimerEvent.TIMER, timerHandler);
                readyTimer.start();
            }
        }
        ...
    }
    else
    {
        trace("External interface is not available for this container.");
    }
}
```

首先，程式碼會使用 ExternalInterface.available 屬性，檢查目前的容器是否可以使用外部 API。如果可以使用，它就開始進行設定通訊的程序。由於當嘗試與外部應用程式進行通訊時，有可能會發生安全性例外和其它錯誤，因此程式碼會位於 try 區塊 (為保持程式碼簡潔，清單中會省略對應的 catch 區塊) 中。

程式碼接著會呼叫 isContainerReady() 方法，如下所示：

```
private function isContainerReady():Boolean
{
    var result:Boolean = ExternalInterface.call("isReady");
    return result;
}
```

然後 isContainerReady() 會使用 ExternalInterface.call() 方法呼叫 JavaScript 函數 isReady()，如下所示：

```
<script language="JavaScript">
<!--
// ------- Private vars -------
var jsReady = false;
...
// ------- functions called by ActionScript -------
// called to check if the page has initialized and JavaScript is available
function isReady()
{
    return jsReady;
}
...
// called by the onload event of the <body> tag
function pageInit()
{
    // Record that JavaScript is ready to go.
    jsReady = true;
}
...
//-->
</script>
```

isReady() 函數只會傳回 jsReady 變數的值。此變數的起始值是 false，當觸發網頁的 onload 事件後，此變數的值才會變成 true。換句話說，如果 ActionScript 在載入網頁前就呼叫 isReady() 函數，JavaScript 會將 false 傳回給 ExternalInterface.call("isReady")，而使 ActionScript 的 isContainerReady() 方法傳回 false。網頁一經載入後，JavaScript isReady() 函數就會傳回 true，因此 ActionScript isContainerReady() 方法也會傳回 true。

回到 IMManager 建構函式中，根據容器的準備狀態會發生下列兩種狀況的其中一種狀況。如果 isContainerReady() 傳回 true，則程式碼只會呼叫 setupCallbacks() 方法，使用 JavaScript 來完成設定通訊的程序；另一種情況是，如果 isContainerReady() 傳回 false，則會將程序暫停，並建立 Timer 物件，使其每 100 毫秒就呼叫一次 timerHandler() 方法，如下所示：

```
private function timerHandler(event:TimerEvent):void
{
    // Check if the container is now ready.
    var isReady:Boolean = isContainerReady();
    if (isReady)
    {
        // If the container has become ready, we don't need to check anymore,
        // so stop the timer.
        Timer(event.target).stop();
        // Set up the ActionScript methods that will be available to be
        // called by the container.
        setupCallbacks();
    }
}
```

每當呼叫 timerHandler() 方法時，它會再檢查一次 isContainerReady() 方法的結果。一旦容器經過初始化，這個方法就會傳回 true。然後程式碼就會停止執行 Timer 並呼叫 setupCallbacks() 方法，完成瀏覽器的通訊設定程序。

將 ActionScript 方法公開給 JavaScript

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如上一個範例中所示，只要程式碼判斷瀏覽器已經完成準備，就會呼叫 setupCallbacks() 方法。這個方法會開始準備讓 ActionScript to 接收來自 JavaScript 的呼叫，如下所示：

```
private function setupCallbacks():void
{
    // Register the SWF client functions with the container
    ExternalInterface.addCallback("newMessage", newMessage);
    ExternalInterface.addCallback("getStatus", getStatus);
    // Notify the container that the SWF is ready to be called.
    ExternalInterface.call("setSWFIsReady");
}
```

setCallBacks() 方法會呼叫 ExternalInterface.addCallback()，註冊將可以透過 JavaScript 呼叫的兩個方法，以完成與容器通訊的準備工作。在這段程式碼中，第一個參數 (JavaScript ("newMessage" 和 "getStatus") 知道的方法名稱) 和 ActionScript 中的方法會使用相同的名稱 (在這種情況下，使用不同的名稱並沒有任何益處，因此為簡化起見，會重複使用相同的名稱)。最後，再使用 ExternalInterface.call() 方法呼叫 JavaScript 函數 setSWFIsReady()，讓容器知道已經完成 ActionScript 函數的註冊作業。

在 ActionScript 與瀏覽器之間進行通訊

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Introvert IM 應用程式將示範一些如何呼叫容器網頁中 JavaScript 函數的範例。在最簡單的範例 (setupCallbacks() 方法中的範例) 中，呼叫 JavaScript 函數 setSWFIsReady() 時並沒有傳遞任何參數或接收傳回的值：

```
ExternalInterface.call("setSWFIsReady");
```

而在另一個來自 isContainerReady() 方法中的範例，ActionScript 會呼叫 isReady() 函數並接收回應的 Boolean 值：

```
var result:Boolean = ExternalInterface.call("isReady");
```

您也可以使用外部 API，將這些參數傳遞給 JavaScript 函數。例如，您可以考慮使用 IMManager 類別的 sendMessage() 方法，當使用者傳送新的訊息給他的「交談夥伴」時，就會呼叫這個方法：

```
public function sendMessage(message:String):void
{
    ExternalInterface.call("newMessage", message);
}
```

同樣地，會使用 ExternalInterface.call() 呼叫指定的 JavaScript 函數，通知瀏覽器有新的訊息送達。此外，訊息本身會做為額外參數傳遞給 ExternalInterface.call()，因此它也會做為參數傳遞給 JavaScript 函數 newMessage()。

透過 JavaScript 呼叫 ActionScript 程式碼

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

所謂通訊，應該就像一條雙線道一樣，Introvert IM 應用程式也不例外。不僅 Flash Player IM 用戶端會呼叫 JavaScript 來傳送訊息，HTML 表單也會呼叫 JavaScript 程式碼來傳送訊息以及要求來自 SWF 檔的資訊。例如，當 SWF 檔通知容器它已經完成建立聯絡通道且已準備進行通訊時，瀏覽器首先會呼叫 IMManager 類別的 getStatus() 方法，從 SWF IM 用戶端擷取第一個使用者的可用性狀態。這項作業是在網頁內的 updateStatus() 函數中完成，如下所示：

```
<script language="JavaScript">
...
function updateStatus()
{
    if (swfReady)
    {
        var currentStatus = getSWF("IntrovertIMApp").getStatus();
        document.forms["imForm"].status.value = currentStatus;
    }
}
...
</script>
```

程式碼會檢查 swfReady 變數的值，以追蹤 SWF 檔是否已經通知瀏覽器它已經使用 ExternalInterface 類別註冊它的方法。如果 SWF 檔已經準備要接收通訊內容，下一行程式碼 (var currentStatus = ...) 就會實際呼叫 IMManager 類別中的 getStatus() 方法。這一行程式碼將執行三項作業：

- 呼叫 getSWF() JavaScript 函數，傳回代表 SWF 檔之 JavaScript 物件的參考。當 HTML 網頁中有一個以上的 SWF 檔時，傳遞給 getSWF() 的參數會判斷應該傳回哪一個瀏覽器物件。傳遞給這個參數的值必須與用來內嵌 SWF 檔之 object 標籤的 id 特質和 embed 標籤的 name 特質相符。
- 使用 SWF 檔的參考時，呼叫 getStatus() 方法就等於是呼叫 SWF 物件的方法。在這種情況下，會使用函數名稱 getStatus，因為這是使用 ExternalInterface.addCallback() 註冊 ActionScript 函數時所使用的名稱。
- getStatus() ActionScript 方法會傳回值，並且將值指派給 currentStatus 變數，做為 status 文字欄位的內容 (value 屬性)。

備註：看完上述程式碼時，您可能已經注意到，在 updateStatus() 函數的原始碼中，呼叫 getSWF() 函數的程式碼實際上撰寫如下：var currentStatus = getSWF("${application}").getStatus()，其中 ${application} 文字是 HTML 網頁範本中的預留位置。當 Adobe Flash Builder 為該程式產生實際的 HTML 網頁時，預留位置文字會由當做 object 標籤之 id 特質以及 embed 標籤之 name 特質 (在此範例中為 IntrovertIMApp) 的相同文字所取代。這就是 getSWF() 函數需要的值。

sendMessage() JavaScript 函數會示範如何將參數傳遞給 ActionScript 函數 (sendMessage() 是當使用者按下 HTML 網頁上的「傳送」按鈕時就會呼叫的函數)。

```
<script language="JavaScript">
...
function sendMessage(message)
{
    if (swfReady)
    {
        ...
        getSWF("IntrovertIMApp").newMessage(message);
    }
}
...
</script>
```

newMessage() ActionScript 方法需要使用一個參數，讓 JavaScript message 變數可以當做 JavaScript 程式碼之 newMessage() 方法呼叫中的參數，傳遞給 ActionScript。

偵測瀏覽器類型

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

由於瀏覽器存取內容的方式各有不同，因此一律透過 JavaScript 使用視窗或文件物件來偵測使用者執行的瀏覽器類型，以及根據瀏覽器指定的語法來存取影片非常重要，如下列範例中的 getSWF() JavaScript 函數所示：

```
<script language="JavaScript">
...
function getSWF(movieName)
{
    if (navigator.appName.indexOf("Microsoft") != -1)
    {
        return window[movieName];
    }
    else
    {
        return document[movieName];
    }
}
...
</script>
```

如果指令碼無法偵測使用者的瀏覽器類型，當使用者播放 HTML 容器中的 SWF 檔時，便可能會發生無法預期的行為。

第 48 章 AIR 中的 XML 簽名驗證

Adobe AIR 1.5 以及更新的版本

您可以使用 Adobe® AIR® XMLSignatureValidator API 中的類別，驗證數位簽名是否符合 W3C 針對「XML 簽名語法及處理」(XML-Signature Syntax and Processing) 的建議事項子集 (http://www.w3.org/TR/xmldsig-core/)。XML 簽名可協助驗證資料或資訊的完整性以及簽署者的身分。

XML 簽名可用來驗證應用程式所下載的訊息或資源。例如，如果應用程式以訂閱方式來提供服務，則您可以將訂閱條款封裝在簽署的 XML 文件中。若有人嘗試變更訂閱文件，驗證就會失敗。

您也可以使用 XML 簽名來協助驗證應用程式所使用的下載資源，方法是加入包含這些資源之摘要的簽名清單。應用程式可以藉由比較簽名檔案中的摘要與從下載位元組計算而來的摘要，驗證資源是否經過變更。當下載的資源是 SWF 檔，或其它要在應用程式安全執行程序執行的作用中內容時，這種作法便相當實用。

XML 簽名驗證的基本概念

Adobe AIR 1.5 以及更新的版本

如需瞭解驗證 XML 簽章的快速說明與程式碼範例，請到 Adobe Developer Connection 參閱下列快速入門文章：

- 建立和驗證 XML 簽名 (Flex)
- 建立和驗證 XML 簽名 (Flash)

Adobe® AIR® 提供了 XMLSignatureValidator 類別和 IURIDereferencer 介面來驗證 XML 簽名。XMLSignatureValidator 類別所接受的 XML 語法是 W3C 建議事項之「XML 簽章語法及處理」(XML-Signature Syntax and Processing) 的子集 (因為只有一個建議的子集受到支援，所以並非所有合法簽名都可以驗證)。AIR 並不提供 API 來建立 XML 簽名。

XML 簽名驗證類別

Adobe AIR 1.5 以及更新的版本

XML 簽名驗證 API 包括下列類別：

套件	類別
flash.security	• XMLSignatureValidator • IURIDereferencer (介面) XMLSignatureValidator 字串常數是在下列類別中定義： • ReferencesValidationSetting • RevocationCheckSettings • SignatureStatus • SignerTrustSettings
flash.events	• Event • ErrorEvent

使用 XML 簽名驗證類別

Adobe AIR 1.5 以及更新的版本

若要使用 XMLSignatureValidator 類別來驗證 XML 簽名，您必須：

- 建立 XMLSignatureValidator 物件
- 提供 IURIDereferencer 介面的實作。XMLSignatureValidator 物件會呼叫 IURIDereferencer 的 dereference() 方法，將各個參照的 URI 傳至簽名中。dereference() 方法必須解析 URI 並傳回參照的資料 (可能與簽名位於相同的文件裡，也可能在某個外部資源中)。
- 根據應用程式的需要來設定 XMLSignatureValidator 物件的憑證信任、撤銷檢查和參照驗證設定。
- 為 complete 和 error 事件新增偵聽程式。
- 呼叫 verify() 方法，並傳入要驗證的簽名。
- 處理 complete 和 error 事件並對結果進行解譯。

下列範例將實作一個 validate() 函數來驗證 XML 簽名的有效性。以這種方式設定 XMLSignatureValidator 屬性，則簽署憑證必定會位於系統信任儲存區，或鍊結到信任儲存區中的某個憑證。此外，這個範例也假設有個名為 XMLDereferencer 的 IURIDereferencer 類別存在。

```
private function validate( xmlSignature:XML ):void
{
    var verifier:XMLSignatureValidator = new XMLSignatureValidator();
    verifier.addEventListener(Event.COMPLETE, verificationComplete);
    verifier.addEventListener(ErrorEvent.ERROR, verificationError);
    try
    {
        verifier.uriDereferencer = new XMLDereferencer();

        verifier.referencesValidationSetting =
            ReferencesValidationSetting.VALID_IDENTITY;
        verifier.revocationCheckSetting = RevocationCheckSettings.BEST_EFFORT;
        verifier.useSystemTrustStore = true;

        //Verify the signature
        verifier.verify( xmlSignature );
    }
    catch (e:Error)
        {
            trace("Verification error.\n" + e);
        }
}

//Trace verification results
private function verificationComplete(event:Event):void

    var signature:XMLSignatureValidator = event.target as XMLSignatureValidator;
    trace("Signature status: " + signature.validityStatus + "\n");
    trace("  Digest status: " + signature.digestStatus + "\n");
    trace("  Identity status: " + signature.identityStatus + "\n");
    trace("  Reference status: " + signature.referencesStatus + "\n");
}

private function verificationError(event:ErrorEvent):void
{
    trace("Verification error.\n" + event.text);
}
```

XML 簽名驗證程序

Adobe AIR 1.5 以及更新的版本

當您呼叫 XMLSignatureValidator 的 verify() 方法時，AIR 會執行下列步驟：

- 執行階段會使用簽署憑證的公開金鑰對簽名的密碼編譯完整性進行驗證。
- 執行階段會根據 XMLSignatureValidator 物件目前的設定來建立憑證的密碼編譯完整性、身分和可靠性。

  簽署憑證中的信任設定就是驗證程序是否完整的關鍵。簽名驗證是透過一個定義完善的密碼編譯程序來進行，但簽署憑證的可靠與否無法透過演算法來判斷。

  一般來說，有三種方法能判斷憑證是否可靠：

  - 依賴憑證授權單位和作業系統信任儲存區。
  - 直接向簽署者取得另一份憑證來做為憑證的信任錨點，或是取得充足的資訊 (例如公開金鑰) 來確實地識別憑證。
  - 詢問應用程式的使用者是否信任目前的憑證。這樣的詢問對自我簽署的憑證並無用處，因為此類憑證裡的識別資訊並不可靠。
- 執行階段會對簽署資料的密碼編譯完整性進行驗證。

簽署資料可藉助 IURIDereferencer 的實作來進行驗證。針對簽名文件每一個參考，IURIDereferencer 實作的 dereference() 方法都會受到呼叫。dereference() 方法所傳回的資料可用來算出參考的摘要。系統會將這個摘要值拿來與簽名文件記錄的摘要進行比較。如果摘要相符，就表示資料自簽署後未曾遭到竄改。

仰賴 XML 簽名結果時有一個重要考量，就是只有受到簽署的內容才是安全的。讓我們以一份列出套件中所含檔案的簽署清單為例。當 XMLSignatureValidator 為簽名進行驗證時，只會檢查清單本身是否未曾遭到竄改。檔案中的資料並沒有受到簽署，所以即使這份清單所參照的檔案已遭變更或刪除，簽名仍可通過驗證。

備註：若要對此類清單中的檔案進行驗證，您可以計算檔案資料的摘要 (與清單使用相同的雜湊演算法計算)，然後將結果拿來與簽署清單中的摘要進行比較。在某些情況下，您也應檢查是否有任何額外的檔案。

解譯驗證結果

Adobe AIR 1.5 以及更新的版本

驗證的結果是由 XMLSignatureValidator 物件的狀態屬性負責回報，而這些屬性可以在驗證工具物件傳送 complete 事件後讀取。共有四種狀態屬性，包括：validityStatus、digestStatus、identityStatus 和 referencesStatus。

validityStatus 屬性

Adobe AIR 1.5 以及更新的版本

validityStatus 屬性負責回報簽名的整體有效性。validityStatus 的值根據其它三種狀態屬性而定，可能為：

- valid — 如果 digestStatus、identityStatus 和 referencesStatus 均為 valid。
- invalid — 如果有一或多個個別狀態屬性為 invalid。
- unknown - 如果有一或多個個別狀態屬性為 unknown 且沒有個別狀態為 invalid。

digestStatus 屬性

Adobe AIR 1.5 以及更新的版本

digestStatus 屬性負責回報訊息摘要的密碼編譯驗證結果。digestStatus 屬性的值可能為：

- valid — 如果簽名文件本身自簽署後未曾遭到竄改。
- invalid - 如果簽名文件已變更或格式不正確。
- unknown — 如果 verify() 方法完成時並非沒有發生錯誤。

identityStatus 屬性

Adobe AIR 1.5 以及更新的版本

identityStatus 屬性負責回報簽署憑證的狀態。這個屬性的值根據幾個因素而定，包括：

- 憑證的密碼編譯完整性
- 憑證是否已過期或撤銷
- 憑證是否在目前的電腦上受到信任
- XMLSignatureValidator 物件的狀態 (例如：是否有新增其它憑證來建立信任鏈結、那些憑證是否受到信任，以及 useSystemTrustStore 和 revocationCheckSettings 屬性的值)

identityStatus 屬性可能具有下列值：

- valid — 簽署憑證必須符合下列條件才視為有效：
  - 簽署憑證必須從未遭到竄改。

  - 除非簽名具有有效的時間戳記，否則簽署憑證不得是已過期或已撤銷。如果簽名具有時間戳記，則只要憑證在文件簽署時為有效，即會被視為有效 (時間戳記服務用來簽署時間戳記的憑證，必須鏈結到使用者電腦上受信任的根憑證)。
  - 簽署憑證受到信任。如果某個憑證位於系統信任儲存區中，或鍊結至系統信任儲存區中的其它憑證，而且 useSystemTrustStore 屬性設為 true，該憑證即受到信任。您也可以使用 XMLSignatureValidator 物件的 addCertificate() 方法來指定信任某個憑證。
  - 而實際上，這個憑證就是簽署憑證。
- invalid — 憑證已過期或撤銷 (且無任何時間戳記可證明簽署時的有效性)，或憑證已遭竄改。
- unknown — 憑證並非無效，但也並未受到信任。例如，自我簽署的憑證就會被回報為 unknown (除非有受到明確的信任)。如果 verify() 方法完成時並非沒有發生錯誤，或是因為簽名摘要無效而沒有檢查身分，identityStatus 同樣也會回報為 unknown。

referencesStatus 屬性
Adobe AIR 1.5 以及更新的版本

referencesStatus 屬性會針對簽名中 SignedData 元素的各個參照來回報其密碼編譯完整性。

- valid — 如果針對簽名中各個參照所算出的摘要完全符合 XML 簽名中所對應的摘要。valid 的狀態表示簽署資料未曾遭到竄改。
- invalid — 如果有任何算出的摘要不符合簽名的對應摘要。
- unknown — 如果尚未檢查參照摘要。如果整體的簽名摘要為 invalid 或簽署憑證為無效時，便不檢查參照。如果 identityStatus 為 unknown，則參照只有在 referencesValidationSetting 為 validOrUnknown 時才會受到檢查。

關於 XML 簽名

Adobe AIR 1.5 以及更新的版本

XML 簽名是以 XML 語法表示的數位簽名。XML 簽名中的資料可用來驗證資訊是否自簽署後未曾遭到竄改。此外，如果簽署憑證是由受信任的憑證授權單位所核發，則簽署者的身分可透過公開金鑰基礎結構進行驗證。

XML 簽名可套用至任何類型的數位資料 (二進位或 XML 格式皆可)。XML 簽名的用途一般為：

- 檢查外部資源或已下載資源是否遭到修改
- 驗證訊息是否來自已知的來源
- 驗證應用程式的授權或訂閱權限

支援的 XML 簽名語法

Adobe AIR 1.5 以及更新的版本

AIR 可支援下列來自 W3C 建議事項之「XML 簽名語法及處理」(XML Signature Syntax and Processing) 的元素：

- 所有的核心簽名語法元素 (W3C 建議事項文件的第 4 節)— 除了 KeyInfo 元素未完全受到支援以外
- KeyInfo 元素必須只包含 X509Data 元素
- X509Data 元素必須只包含 X509Certificate 元素
- SHA256 摘要方法
- RSA-SHA1 (PKCS1) 簽署演算法

- 「不含註解、規範化的 XML」規範化方法與轉換
- 封內簽名轉換
- 時間戳記

下列文件說明何謂典型的 XML 簽名 (為了簡化範例，其中的大部分密碼編譯簽名資料皆已移除)：

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <Reference URI="URI_to_signed_data">
            <Transforms>
                <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
</Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <DigestValue>uoo...vY=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue>Ked...w==</SignatureValue>
    <KeyInfo>
        <X509Data>
            <X509Certificate>i7d...w==</X509Certificate>
        </X509Data>
    </KeyInfo>
</Signature>
```

構成簽名的關鍵元素為：

- SignedInfo — 內有簽署資料的參照與簽署時所算出的摘要值。簽署資料本身可能與 XML 簽名位於同一份文件內，也可能位於外部。
- SignatureValue — 其中含有 SignedInfo 元素的摘要，此摘要以簽署者的私密金鑰進行加密。
- KeyInfo — 其中含有簽署憑證，以及用以建立信任鍊結所需的所有額外憑證。請注意，雖然 KeyInfo 元素屬於選擇性元素，但如果未將其包含在內，AIR 便無法驗證簽名。

XML 簽名有三種一般類型：

- 封內簽名 — 簽名被包在所簽署的 XML 資料中。
- 封外簽名 — 簽署的 XML 資料包含在 Signature 元素中的 Object 元素之內。
- 分離簽名 — 簽署資料與 XML 簽名是分離的。可能位於某個外部檔案內。或者，簽署資料也可能與簽名位於同一份 XML 文件內，但不是位於 Signature 元素的父元素或子元素內。

XML 簽名會使用 URI 來參照簽署資料。負責簽署與驗證的應用程式都必須使用相同的慣例來解析這些 URI。當您使用 XMLSignatureValidator 類別時，必須提供 IURIDereferencer 介面的實作。這份實作應負責解析 URI 並將簽署資料以 ByteArray 物件傳回。接著會以當初在簽名內用來產生摘要的演算法，為所傳回的 ByteArray 物件產生摘要。

憑證和信任

Adobe AIR 1.5 以及更新的版本

憑證的組成包括一個公開金鑰、識別資訊，並可能包括一或多個屬於核發商憑證授權單位的憑證。

有兩種方法可以在憑證中建立信任。若要建立信任，您可以直接向簽署者取得一份憑證，例如從實體媒體或透過安全的數位傳輸 (例如 SSL 交易) 取得。您也可以依賴憑證授權單位來判斷簽署憑證是否可靠。

若要依賴憑證授權單位，則核發簽署憑證的授權單位必須受到當初對簽名進行驗證的電腦所信任。大多數的作業系統製造商會將某些憑證授權單位的根憑證放到作業系統信任儲存區中。使用者也可以在該儲存區中進行憑證的新增和移除。

即使憑證是由受信任的憑證授權單位所核發，您也必須判斷那是不是自己信任的核發單位。在許多使用案例中，這項判斷是由使用者決定。例如，安裝 AIR 應用程式時，AIR 安裝程式會顯示發行者憑證中的識別資訊，並要求使用者確認是否要安裝此應用程式。在其它情況下，您可能必須將公開金鑰或其它憑證資訊拿來與一份可接受的金鑰清單進行比對 (這份清單必須安全無虞，可以透過本身的簽名加以保護，或是將清單儲存在 AIR 加密本機儲存區中，以免遭到竄改)。

備註：您雖然可以選擇信任未經獨立驗證的簽署憑證 (例如當簽名屬於 「自我簽署」的性質時)，但驗證這樣的簽名並不能讓您安心多少。如果不知道簽名的建立者是誰，即使確定簽名未曾遭到竄改也沒有太大意義。簽名可能是一份簽署有效的偽造簽名。

憑證到期和撤銷

Adobe AIR 1.5 以及更新的版本

所有憑證都會過期。憑證也可能受到核發商憑證授權單位的撤銷，例如當憑證的私密金鑰遭到竄改或竊取時。如果簽名是某個已過期或已撤銷的憑證所簽署，該簽名就會回報為無效，除非簽名中含有時間戳記。如果有時間戳記，則只要憑證在當初簽署時為有效，XMLSignatureValidator 類別就會驗證簽名。

時間戳記是來自時間戳記服務的經過簽署的數位訊息，該項服務可證明某資料在某個日期和時間確實受到簽署。時間戳記是由時間戳記授權單位所核發，且由該單位本身的憑證所簽署。時間戳記所內嵌之時間戳記授權單位憑證必須受目前電腦所信任，時間戳記才可視為有效。XMLSignatureValidator 並不提供 API 來指定使用其它憑證來驗證時間戳記。

實作 IURIDereferencer 介面

Adobe AIR 1.5 以及更新的版本

若要驗證 XML 簽名，您必須提供 IURIDereferencer 介面的實作。這份實作應負責對 XML 簽名文件中 Reference 元素的 URI 進行解析，然後傳回資料以便計算摘要。算出的摘要會拿來與簽名中的摘要進行比較，以判斷參照的資料自簽名建立後是否曾遭到竄改。

備註：HTML 類型的 AIR 應用程式必須匯入一個含有 ActionScript 實作的 SWF 元件庫，以驗證 XML 簽名。IURIDereferencer 介面無法以 JavaScript 進行實作。

IURIDerefencer 介面只具有一個方法，就是 dereference(uri:String)，而此方法必須實作。XMLSignatureValidator 物件會針對簽名中的各個參照來呼叫這個方法。此方法必須以 ByteArray 物件傳回資料。

在大部分的情況下，您也必須新增某些屬性或方法，讓您的提領者 (dereferencer) 物件可以找到參照的資料。例如，如果簽署的資料與簽名位於同一份文件內，您就可以新增一個成員變數來參照該 XML 文件。dereference() 方法接著便可使用這個變數搭配 URI 來找到參照的資料。同樣地，如果簽署的資料位於本機檔案系統中的某個目錄，dereference() 方法可能就需要一個屬性來提供該目錄的路徑，這樣才能夠解析參照的檔案。

XMLSignatureValidator 完全依賴提領者來解譯 URI 字串。URI 的標準提領規則在 W3C 建議事項的「XML 簽章語法及處理」 (XML Signature Syntax and Processing) 的第 4.3.3 節有說明。

提領封內簽名中的 URI

Adobe AIR 1.5 以及更新的版本

在產生封內 XML 簽名時，簽名元素會插入簽署資料中。例如，如果您使用封內簽名結構來簽署下列訊息：

```
<message>
    <data>...</data>
</message>
```

最後的簽署文件如下所示：

```
<message>
    <data>...</data>
     <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <Reference URI="">
                <Transforms>
                    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <DigestValue>yv6...Z0Y=</DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>cCY...LQ==</SignatureValue>
        <KeyInfo>
            <X509Data>
                <X509Certificate>MII...4e</X509Certificate>
            </X509Data>
        </KeyInfo>
    </Signature>
</message>
```

請注意，簽名中僅有一個 Reference 元素，元素中以一個空字串做為其 URI。此內容中的空字串參照至文件根。

也請您注意，轉換演算法指出封內簽名轉換已經套用。封內簽名轉換一經套用，XMLSignatureValidator 就會自動將簽名從文件中移除，然後才計算摘要。這表示提領者在傳回資料時並不需要移除 Signature 元素。

以下提供一個封內簽名提領者的例子：

```
package
{
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class EnvelopedDereferencer
        extends EventDispatcher implements IURIDereferencer
    {
        private var signedMessage:XML;

        public function EnvelopedDereferencer( signedMessage:XML )
        {
            this.signedMessage = signedMessage;
        }

        public function dereference( uri:String ):IDataInput
        {
            try
            {
                if( uri.length != 0 )
                {
                    throw( new Error("Unsupported signature type.") );
```

```
                }
                var data:ByteArray = new ByteArray();
                data.writeUTFBytes( signedMessage.toXMLString() );
                data.position = 0;
            }
            catch (e:Error)
                {
                var error:ErrorEvent =
                    new ErrorEvent("Ref error " + uri + " ", false, false, e.message);
                this.dispatchEvent(error);
                data = null;
                throw new Error("Reference not resolvable: " + uri + ", " + e.message);
            }
            finally
            {
                return data;
            }
        }
    }
}
```

這個提領者類別使用了一個建構函式，此函數具有一個 signedMessage 參數，因而將封內簽名文件提供給 dereference() 方法使用。因為封內簽名中的參照一定是指向簽署資料的根，所以 dereferencer() 方法會將此文件寫入一個位元組陣列，然後將其傳回。

提領封外簽名和分離簽名中的 URI

Adobe AIR 1.5 以及更新的版本

當簽署的資料與簽名本身位於同一份文件中，參照中的 URI 通常會使用 XPath 或 XPointer 語法來敘述已簽署的元素。W3C 建議事項的「XML 簽章語法及處理」(XML-Signature Syntax and Processing) 只建議使用此語法，所以您應該根據將處理的簽名來進行實作 (並且增加充分的錯誤檢查作業，以從容地處理未受支援的語法)。

AIR 應用程式的簽名就是封外簽名的一個例子。應用程式中的檔案列於 Manifest 元素內。Manifest 元素是在 Reference URI 特質中以「#PackageContents」字串描述，而此字串參照 Manifest 元素的 ID：

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#" Id="PackageSignature">
    <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <SignatureMethod Algorithm="http://www.w3.org/TR/xmldsig-core#rsa-sha1"/>
        <Reference URI="#PackageContents">
            <Transforms>
                <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <DigestValue>ZMGqQdaRKQc1HirIRsDpeBDlaElS+pPotdziIAyAYDk=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue Id="PackageSignatureValue">cQK...7Zg==</SignatureValue>
    <KeyInfo>
        <X509Data>
            <X509Certificate>MII...T4e</X509Certificate>
        </X509Data>
    </KeyInfo>
    <Object>
    <Manifest Id="PackageContents">
        <Reference URI="mimetype">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>0/oCb84THKMagtI0Dy0KogEu92TegdesqRr/clXct1c=</DigestValue>
        </Reference>
        <Reference URI="META-INF/AIR/application.xml">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>P9MqtqSdqcqnFgeoHCJysLQu4PmbUW2JdAnc1WLq8h4=</DigestValue>
        </Reference>
        <Reference URI="XMLSignatureValidation.swf">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>OliRHRAgc9qt3Dk0m0Bi53Ur5ur3fAweIFwju74rFgE=</DigestValue>
        </Reference>
    </Manifest>
</Object>
</Signature>
```

驗證此簽名的提領者必須將 Reference 元素中含有 #PackageContents 的 URI 字串取出，然後以 ByteArray 物件傳回 Manifest 元素。「#」符號是指元素的 ID 特質值。

以下範例是提領者驗證 AIR 應用程式簽名的實作。為了將實作簡化，僅以已知的 AIR 簽名結構進行。一般用途的提領者可能會更為複雜。

```
package
{
    import flash.events.ErrorEvent;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class AIRSignatureDereferencer implements IURIDereferencer {
        private const XML_SIG_NS:Namespace =
            new Namespace( "http://www.w3.org/2000/09/xmldsig#" );
        private var airSignature:XML;

        public function AIRSignatureDereferencer( airSignature:XML ) {
            this.airSignature = airSignature;
        }

        public function dereference( uri:String ):IDataInput {
            var data:ByteArray = null;
            try
            {
                if( uri != "#PackageContents" )
                {
                    throw( new Error("Unsupported signature type.") );
                }
                var manifest:XMLList =
                    airSignature.XML_SIG_NS::Object.XML_SIG_NS::Manifest;
                data = new ByteArray();
                data.writeUTFBytes( manifest.toXMLString());
                data.position = 0;
            }
            catch (e:Error)
            {
                data = null;
                throw new Error("Reference not resolvable: " + uri + ", " + e.message);
            }
            finally
            {
                return data;
            }

        }
    }
}
```

當您在驗證此類簽名時，只有 Manifest 元素中的資料會受到驗證。套件中的實際檔案完全沒有受到檢查。若要檢查套件檔案是否遭到竄改，您必須讀取檔案、計算 SHA256 摘要，然後將計算結果拿來與清單中的摘要進行比較。XMLSignatureValidator 不會自動檢查這種次要參照。

備註：此範例僅供說明簽名驗證程序之用。由 AIR 應用程式驗證本身的簽名並沒有太大用處。如果應用程式已遭到竄改，竄改者只要將驗證檢查作業移除即可。

計算外部資源的摘要值

Adobe AIR 1.5 以及更新的版本

AIR 並未包含可計算 SHA256 摘要的函數，但 Flex SDK 確實有一個 SHA256 公用程式類別。SDK 也含有一個 Base64 編碼器公用程式類別，可將計算出的摘要拿來與簽名中的摘要進行比較。

下列範例中的函數會讀取並驗證 AIR 套件清單中的檔案：

```
import mx.utils.Base64Encoder;
import mx.utils.SHA256;

private function verifyManifest( sigFile:File, manifest:XML ):Boolean
{
    var result:Boolean = true;
    var message:String = '';
    var nameSpace:Namespace = manifest.namespace();

    if( manifest.nameSpace::Reference.length() <= 0 )
    {
        result = false;
        message = "Nothing to validate.";
    }
    for each (var reference:XML in manifest.nameSpace::Reference)
    {
        var file:File = sigFile.parent.parent.resolvePath( reference.@URI );
        var stream:FileStream = new FileStream();
        stream.open(file, FileMode.READ);
        var fileData:ByteArray = new ByteArray();
        stream.readBytes( fileData, 0, stream.bytesAvailable );

        var digestHex:String = SHA256.computeDigest( fileData );
        //Convert hexidecimal string to byte array
        var digest:ByteArray = new ByteArray();
        for( var c:int = 0; c < digestHex.length; c += 2 ){
            var byteChar:String = digestHex.charAt(c) + digestHex.charAt(c+1);
            digest.writeByte( parseInt( byteChar, 16 ));
        }
        digest.position = 0;

        var base64Encoder:Base64Encoder = new Base64Encoder();
        base64Encoder.insertNewLines = false;
        base64Encoder.encodeBytes( digest, 0, digest.bytesAvailable );
        var digestBase64:String = base64Encoder.toString();
        if( digestBase64 == reference.nameSpace::DigestValue )
        {
            result = result && true;
            message += "   " + reference.@URI + " verified.\n";
        }
        else
        {
            result = false;
            message += " ---- " + reference.@URI + " has been modified!\n";
        }
        base64Encoder.reset();
    }
    trace( message );
    return result;
}
```

這個函數會循序處理 Manifest 元素中的所有參照。針對各個參照，SHA256 摘要會計算出來、以 base64 格式編碼，然後與清單中的摘要進行比較。AIR 套件中的 URI 會指向應用程式目錄的相對路徑。這些路徑會根據簽名檔案的位置接受解析 (簽名檔案的位置一定在應用程式目錄的 META-INF 子目錄中)。請注意，Flex SHA256 類別會將摘要以十六進位字元的字串傳回。這個字串必須轉換為 ByteArray，其中含有以十六進位字串所表示的位元組。

若要在 Flash CS4 中使用 mx.utils.SHA256 和 Base64Encoder 類別，您可以找出這些類別並將它們複製到應用程式開發目錄中，也可以編譯一個文件庫 SWF，其中含有使用 Flex SDK 的類別。

向參照外部資料的分離簽名提領 URI

Adobe AIR 1.5 以及更新的版本

當 URI 參照外部資源時，必須存取資料就並將其載入 ByteArray 物件。如果 URI 含有的是絕對 URL，那麼就只是要讀取檔案或要求 URL 的問題而已。如果 URI 含有的是相對 URL (這可能是比較常見的情形)，您的 IURIDereferencer 實作就必須有一個方法來解析受簽署檔案的路徑。

在下列範例中，當提領者實體是做為負責解析已簽署檔案的基礎時，File 物件便進行初始化。

```
package
{
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.filesystem.File;
    import flash.filesystem.FileMode;
    import flash.filesystem.FileStream;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;
    public class RelativeFileDereferencer
        extends EventDispatcher implements IURIDereferencer
    {
        private var base:File;

        public function RelativeFileDereferencer( base:File )
        {
            this.base = base;
        }

        public function dereference( uri:String ):IDataInput
        {
            var data:ByteArray = null;
            try{
                var referent:File = this.base.resolvePath( uri );
                var refStream:FileStream = new FileStream();
                data = new ByteArray();
                refStream.open( referent, FileMode.READ );

                refStream.readBytes( data, 0, data.bytesAvailable );

            } catch ( e:Error ) {
                data = null;
                throw new Error("Reference not resolvable: " + referent.nativePath + ", " + e.message );
            } finally {
                return data;
            }
        }
    }
}
```

dereference() 函數只會找出參考 URI 所描述的檔案、將檔案內容載入位元組陣列，然後傳回 ByteArray 物件。

備註：在驗證遠端外部參考之前，請考慮應用程式是否可能受到惡意建構之簽名文件所進行的反隱私回報 (phone home) 或類似類型的攻擊。

第 49 章 用戶端系統環境

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

本文說明如何與使用者的系統進行互動。並說明如何判斷支援的功能，以及如何利用使用者所安裝的輸入法編輯器 (IME) (如果有的話) 建置多國語言的應用程式。本章也將說明應用程式網域的典型用法。

更多說明主題

flash.system.System

flash.system.Capabilities

用戶端系統環境的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

隨著所建立的 應用程式比以往更加進階，您將更加需要瞭解關於使用者作業系統的詳細資訊和存取功能。flash.system 套件所含的類別集合可讓您存取下列系統階層功能：

- 判斷程式碼在哪個應用程式及安全性網域中執行
- 判斷使用者的 Flash® Player 或 Adobe® AIR™ 實體的功能，例如螢幕大小 (解析度)，以及是否提供特定的功能，例如 MP3 音效
- 使用 IME 建置多國語言網站
- 與 Flash 執行階段的容器 (可能是 HTML 網頁或是容器應用程式) 進行互動。
- 將資訊儲存至使用者的剪貼簿中

flash.system 套件也包含 IMEConversionMode 和 SecurityPanel 類別，這些類別分別包含了可以搭配 IME 和 Security 類別使用的靜態常數。

重要概念與術語

下列參考清單包含重要術語：

作業系統 電腦上執行的主要程式，例如 Microsoft Windows、Mac OS X 或 Linux® 等，其它所有應用程式都必須在此主要程式之中執行。

Clipboard 用來保存所複製或剪下之文字或項目的作業系統容器，文字或項目再從這個容器貼到應用程式中。

應用程式網域 用來區分不同 SWF 檔中所使用類別的機制，如果 SWF 檔中包含具有相同名稱的不同類別，此機制可確保這些類別不會互相覆寫。

IME (輸入法編輯器) 透過標準鍵盤，用來輸入複雜字元或符號的程式 (或作業系統工具)。

用戶端系統 在程式設計的術語中，「用戶端」是指在個人的電腦上執行的部分應用程式 (有時是整個應用程式)，一般是由單一使用者加以使用。「用戶端系統」是使用者電腦上的基礎作業系統。

使用 System 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

System 類別包含的方法與屬性，可讓您與使用者的作業系統進行互動，並擷取執行階段目前的記憶體使用量。System 類別的方法和屬性可讓您偵聽 imeComposition 事件，並指示 執行階段利用使用者的目前字碼頁載入外部文字檔，或是將這些檔案載入為 Unicode，或者設定使用者的剪貼簿內容。

在執行階段取得使用者系統的相關資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以藉由檢查 System.totalMemory 屬性，判斷 執行階段目前所佔用的記憶體位元組數量。這個屬性可讓您監視所佔用的記憶體，並根據記憶體等級變更的程度，最佳化您的應用程式。例如，假設某個視覺效果造成記憶體使用量大幅增加，您可能會想要修改此效果，或是將其徹底刪除。

System.ime 屬性是目前所安裝 「輸入法編輯器」 (IME) 的參考。這個屬性可以讓您藉由使用 addEventListener() 方法，偵聽 imeComposition 事件 (flash.events.IMEEvent.IME_COMPOSITION)。

System 類別中的第三個屬性為 useCodePage。當 useCodePage 設定為 true 時，執行階段就會使用作業系統的傳統字碼頁來載入外部文字檔。如果您將此屬性設定為 false，就會指示 執行階段將外部檔案解譯為 Unicode。

如果您將 System.useCodePage 設定為 true，則必須記住，作業系統的傳統字碼頁必須包含外部文字檔中使用的字元，如此才能顯示該檔案中的文字。例如，如果您載入包含中文字元的外部文字檔案，這些中文字元將不會顯示在使用英文版 Windows 字碼頁的系統上，因為該字碼頁並不包含中文字元。

若要確保所有平台上的使用者都可以檢視應用程式中的外部文字檔案，則應該將所有外部文字檔案都編碼為 Unicode，而且保留 System.useCodePage 的 false 預設設定。這樣，執行階段就會將文字解譯為 Unicode。

將文字儲存到剪貼簿

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

System 類別包含名為 setClipboard() 的方法，可讓 Flash 執行階段以指定的字串來設定使用者的剪貼簿內容。基於安全性考量，該類別中並沒有 Security.getClipboard() 方法，因為這種方法具有潛在性風險，可能會讓惡意網站存取最後複製到使用者「剪貼簿」中的資料。

下列程式碼說明發生安全性錯誤時，錯誤訊息如何複製到使用者的剪貼簿中。如果使用者要透過應用程式回報潛在性錯誤，則此錯誤訊息會非常有用。

```
private function securityErrorHandler(event:SecurityErrorEvent):void
{
    var errorString:String = "[" + event.type + "] " + event.text;
    trace(errorString);
    System.setClipboard(errorString);
}
```

Flash Player 10 和 **AIR 1.0**

您可以使用 Clipboard 類別來讀取以及寫入剪貼簿資料，以便回應使用者事件。在 AIR 中，於應用程式安全執行程序中執行的程式碼存取剪貼簿時，不需要使用者事件。

使用 Capabilities 類別

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Capabilities 類別可讓開發人員判斷應用程式的執行環境。您可以使用 Capabilities 類別的各種不同屬性，針對使用者的系統是否支援輔助功能軟體、使用者作業系統的語言，以及目前所安裝 Flash 執行階段的版本等，找出解決方案。

您可以藉由檢查 Capabilities 類別中的屬性自訂應用程式，而以最佳方式搭配特定使用者的系統來運作。例如，您可以藉由檢查 Capabilities.screenResolutionX 和 Capabilities.screenResolutionY 屬性，判斷使用者系統的顯示解析度，並決定最適當的視訊尺寸。或者，您也可以在嘗試載入外部 MP3 檔案之前，先檢查 Capabilities.hasMP3 屬性，了解使用者的系統是否支援 MP3 播放功能。

下列程式碼會使用規則運算式，剖析用戶端所使用的 Flash 執行階段版本：

```
var versionString:String = Capabilities.version;
var pattern:RegExp = /^(\w*) (\d*),(\d*),(\d*)$/;
var result:Object = pattern.exec(versionString);
if (result != null)
{
    trace("input: " + result.input);
    trace("platform: " + result[1]);
    trace("majorVersion: " + result[2]);
    trace("minorVersion: " + result[3]);
    trace("buildNumber: " + result[4]);
    trace("internalBuildNumber: " + result[5]);
}
else
{
    trace("Unable to match RegExp.");
}
```

如果您要將使用者的系統功能傳送到伺服器端指令碼，以便讓相關的資訊儲存在資料庫中，則可以使用下列 ActionScript 程式碼：

```
var url:String = "log_visitor.cfm";
var request:URLRequest = new URLRequest(url);
request.method = URLRequestMethod.POST;
request.data = new URLVariables(Capabilities.serverString);
var loader:URLLoader = new URLLoader(request);
```

功能範例：偵測系統功能

Flash Player 9 和更新的版本

CapabilitiesExplorer 範例會示範如何使用 flash.system.Capabilities 類別，以判斷使用者的 Flash 執行階段版本所支援的功能。此範例將教導您下列技巧：

- 使用 Capabilities 類別，偵測使用者的 Flash 執行階段版本所支援的功能
- 使用 ExternalInterface 類別，偵測使用者的瀏覽器所支援的瀏覽器設定

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。您可以在 Samples/CapabilitiesExplorer 檔案夾中找到 CapabilitiesExplorer 應用程式檔案，它是由下列檔案組成：

檔案	說明
CapabilitiesExplorer.fla 或 CapabilitiesExplorer.mxml	主應用程式檔案，在 Flash 中為 FLA，在 Flex 中為 MXML。
com/example/programmingas3/capabilities/CapabilitiesGrabber.as	提供應用程式之主要功能的類別，這些功能包含將系統 Capabilities 加入陣列中、為項目排序，以即使用 ExternalInterface 類別擷取瀏覽器功能。
capabilities.html	包含必要之 JavaScript 的 HTML 容器，以便與外部 API 進行通訊。

CapabilitiesExplorer 概觀

Flash Player 9 和更新的版本

CapabilitiesExplorer.mxml 檔案的作用是負責設定 CapabilitiesExplorer 應用程式的使用者介面。使用者的 Flash 執行階段版本的功能，會顯示在 「舞台」上的 DataGrid 組件實體中。如果使用者是透過 HTML 容器執行應用程式，並且能使用外部 API 的話，則他們的瀏覽器功能也會顯示出來。

傳送主應用程式檔案的 creationComplete 事件時，便會叫用 initApp() 方法。initApp() 方法會從 com.example.programmingas3.capabilities.CapabilitiesGrabber 類別中呼叫 getCapabilities() 方法。initApp() 方法的程式碼如下：

```
private function initApp():void
{
    var dp:Array = CapabilitiesGrabber.getCapabilities();
    capabilitiesGrid.dataProvider = dp;
}
```

CapabilitiesGrabber.getCapabilities() 方法會傳回經過排序的 Flash 執行階段及瀏覽器功能陣列，然後這個陣列再設定給 「舞台」上 capabilitiesGrid DataGrid 組件實體的 dataProvider 屬性。

CapabilitiesGrabber 類別概觀

Flash Player 9 和更新的版本

CapabilitiesGrabber 類別的靜態 getCapabilities() 方法會將來自 flash.system.Capabilities 類別的每個屬性都加入陣列中 (capDP)。接著，它會呼叫 CapabilitiesGrabber 類別中的靜態 getBrowserObjects() 方法。getBrowserObjects() 方法會使用外部 API，以循環執行瀏覽器的 navigator 物件，此物件中包含瀏覽器的功能。getCapabilities() 方法如下所示：

```
public static function getCapabilities():Array
{
    var capDP:Array = new Array();
    capDP.push({name:"Capabilities.avHardwareDisable", value:Capabilities.avHardwareDisable});
    capDP.push({name:"Capabilities.hasAccessibility", value:Capabilities.hasAccessibility});
    capDP.push({name:"Capabilities.hasAudio", value:Capabilities.hasAudio});
    ...
    capDP.push({name:"Capabilities.version", value:Capabilities.version});
    var navArr:Array = CapabilitiesGrabber.getBrowserObjects();
    if (navArr.length > 0)
    {
        capDP = capDP.concat(navArr);
    }
    capDP.sortOn("name", Array.CASEINSENSITIVE);
    return capDP;
}
```

getBrowserObjects() 方法會傳回瀏覽器之 navigator 物件中每一個屬性的陣列。如果這個陣列的長度為一個或多個項目，則瀏覽器功能陣列 (navArr) 就會附加至 Flash Player 功能陣列 (capDP)，然後整個陣列再依照字母進行排序。最後，經過排序的陣列會傳回至主應用程式檔案，再將這個檔案填入資料格點。getBrowserObjects() 方法的程式碼如下：

```
private static function getBrowserObjects():Array
{
    var itemArr:Array = new Array();
    var itemVars:URLVariables;
    if (ExternalInterface.available)
    {
        try
        {
            var tempStr:String = ExternalInterface.call("JS_getBrowserObjects");
            itemVars = new URLVariables(tempStr);
            for (var i:String in itemVars)
            {
                itemArr.push({name:i, value:itemVars[i]});
            }
        }
        catch (error:SecurityError)
        {
            // ignore
        }
    }
    return itemArr;
}
```

如果目前的使用者環境中可以使用外部 API，Flash 執行階段便會呼叫 JavaScript 的 JS_getBrowserObjects() 方法，循環執行瀏覽器的 navigator 物件，並將經過 URL 編碼的字串值傳回至 ActionScript。然後這個字串便會轉換為 URLVariables 物件 (itemVars)，並加入 itemArr 陣列中，而此陣列會傳回至呼叫端指令碼。

與 JavaScript 進行通訊

Flash Player 9 和更新的版本

建立 CapabilitiesExplorer 應用程式的最後一部分是撰寫必要的 JavaScript，以循環執行瀏覽器之 navigator 物件中的每一個項目，並將名稱 / 值配對附加到暫存陣列中。container.html 檔案中之 JavaScript JS_getBrowserObjects() 方法的程式碼如下所示：

```
<script language="JavaScript">
    function JS_getBrowserObjects()
    {
        // Create an array to hold each of the browser's items.
        var tempArr = new Array();

        // Loop over each item in the browser's navigator object.
        for (var name in navigator)
        {
            var value = navigator[name];

            // If the current value is a string or Boolean object, add it to the
            // array, otherwise ignore the item.
            switch (typeof(value))
            {
                case "string":
                case "boolean":

                    // Create a temporary string which will be added to the array.
                    // Make sure that we URL-encode the values using JavaScript's
                    // escape() function.
```

```
                var tempStr = "navigator." + name + "=" + escape(value);
                // Push the URL-encoded name/value pair onto the array.
                tempArr.push(tempStr);
                break;
            }
        }
        // Loop over each item in the browser's screen object.
        for (var name in screen)
        {
            var value = screen[name];

            // If the current value is a number, add it to the array, otherwise
            // ignore the item.
            switch (typeof(value))
            {
                case "number":
                    var tempStr = "screen." + name + "=" + escape(value);
                    tempArr.push(tempStr);
                    break;
            }
        }
        // Return the array as a URL-encoded string of name-value pairs.
        return tempArr.join("&");
    }
</script>
```

程式碼是從建立暫存陣列開始，該陣列會保存 navigator 物件中的所有名稱 / 值配對。接著再使用 for..in 迴圈循環執行 navigator 物件，然後評估目前值的資料類型，以篩選出不想要的值。在此應用程式中，我們只要類型為 String 或 Boolean 的值，而其它類型 (例如函數或陣列) 則加以忽略。將 navigator 物件中的每一個 String 或 Boolean 值都附加到 tempArr 陣列，然後再使用 for..in 迴圈循環執行瀏覽器的 screen 物件，並將每一個數值加入至 tempArr 陣列。最後會使用 Array.join() 方法，將該暫存陣列轉換為字串。這個陣列會使用 & (and 符號) 做為分隔符號，讓 ActionScript 能夠使用 URLVariables 類別輕易剖析資料。

第 50 章 AIR 應用程式叫用與終止

Adobe AIR 1.0 以及更新的版本

本節討論叫用已安裝 Adobe® AIR® 應用程式的方式，以及關閉執行中應用程式的選項與考量。

備註：只有在 AIR 應用程式安全執行程序中執行的 SWF 內容，才能使用 NativeApplication、InvokeEvent 和 BrowserInvokeEvent 物件。在 Flash Player 執行階段中、瀏覽器或獨立播放程式 (放映檔) 內、或在應用程式安全執行程序外的 AIR 應用程式中執行的 SWF 內容，無法存取這些類別。

如需叫用以及終止 AIR 應用程式的快速說明與程式碼範例，請到 Adobe Developer Connection 參閱下列快速入門文章：

- 啟動選項

更多說明主題

flash.desktop.NativeApplication

flash.events.InvokeEvent

flash.events.BrowserInvokeEvent

叫用應用程式

Adobe AIR 1.0 以及更新的版本

當使用者 (或作業系統) 執行下列操作時，便會叫用 AIR 應用程式：

- 從桌面殼層啟動應用程式。
- 從命令列殼層輸入命令來使用應用程式。
- 開啟某一類型的檔案，而應用程式正好是該類型的預設應用程式。
- (Mac OS X) 按一下停駐工作列上的應用程式圖示 (不論應用程式目前是否正在執行)。
- 選擇從安裝程式啟動應用程式 (在全新安裝過程的結束步驟時，或是按兩下已安裝的應用程式 AIR 檔之後)。
- 當已安裝的版本通知要自行處理應用程式更新時 (由於應用程式描述器檔案中包含 <customUpdateUI>true</customUpdateUI> 宣告)，開始更新 AIR 應用程式。
- (iOS) 擷取 Apple 推播通知服務 (APNs) 的通知。
- 透過 URL 叫用應用程式。
- 造訪裝載 Flash 標誌或應用程式的網頁，以致指定 AIR 應用程式的識別資訊來呼叫 com.adobe.air.AIR launchApplication() 方法 (應用程式描述器檔案中也必須包含 <allowBrowserInvocation>true</allowBrowserInvocation> 宣告，瀏覽器叫用才會成功)。

每當叫用 AIR 應用程式時，AIR 就會透過 Singleton NativeApplication 物件來傳送 invoke 類型的 InvokeEvent 物件。為了讓應用程式能適時自行初始化及註冊事件偵聽程式，invoke 事件都會排入佇列，而不是遭到捨棄。只要已註冊偵聽程式，隨即傳送佇列中的所有事件。

備註：透過瀏覽器叫用功能叫用應用程式時，如果應用程式尚未執行，NativeApplication 物件只會傳送 invoke 事件。

若要接收 invoke 事件，請呼叫 NativeApplication 物件 (NativeApplication.nativeApplication) 的 addEventListener() 方法。為 invoke 事件註冊事件偵聽程式時，此偵聽程式也將接收註冊前發生的所有 invoke 事件。在呼叫 addEventListener() 而有傳回動作之後，就會在短時間間隔內一次傳送一個佇列中的 invoke 事件。若此期間發生了新的 invoke 事件，則可能擱置佇列中的一個或多個事件，先傳送該事件。這樣的事件佇列方式讓您得以處理初始化程式碼執行之前發生的任何 invoke 事件。請牢記，如果您稍後在執行時 (初始化應用程式之後) 加入事件偵聽程式，此偵聽程式仍將會接收應用程式啟動之後發生的所有 invoke 事件。

每個 AIR 應用程式只會啟動一個實體。如果再次叫用現已執行的應用程式，AIR 將傳送新的 invoke 事件給執行中的實體。AIR 應用程式必須負責回應 invoke 事件並採取適當的動作 (例如開啟新的文件視窗)。

InvokeEvent 物件含有傳遞給應用程式的任何引數，以及從中叫用應用程式的目錄位置。若是由於檔案類型關聯而叫用應用程式，命令列引數便會包括檔案的完整路徑。同樣地，如果叫用應用程式是為了進行應用程式更新，則會提供更新版 AIR 檔的完整路徑。

在 Mac OS X 上以單一作業中開啟多個檔案時，會傳送單一的 InvokeEvent 物件。每個檔案都會包含在 arguments 陣列中。在 Windows 和 Linux 上，則會針對每個檔案傳送個別的 InvokeEvent 物件。

應用程式可使用 NativeApplication 物件來註冊偵聽程式，以便處理 invoke 事件：

```
NativeApplication.nativeApplication.addEventListener(InvokeEvent.INVOKE, onInvokeEvent);
```

接下來會定義事件偵聽程式：

```
 var arguments:Array;
var currentDir:File;
public function onInvokeEvent(invocation:InvokeEvent):void {
    arguments = invocation.arguments;
    currentDir = invocation.currentDirectory;
}
```

擷取命令列引數

Adobe AIR 1.0 以及更新的版本

與叫用 AIR 應用程式有關的命令列引數，會以 NativeApplication 物件所傳送的 InvokeEvent 物件來傳遞。InvokeEvent 的 arguments 屬性包含作業系統叫用 AIR 應用程式時傳遞的引數陣列。如果引數含有檔案相對路徑，您通常可以使用 currentDirectory 屬性來解析路徑。

傳遞給 AIR 程式的引數均會視為以空白字元分隔的字串，除非引數包括在雙引號內：

引數	Array
tick tock	{tick,tock}
tick "tick tock"	{tick,tick tock}
"tick" "tock"	{tick,tock}
\"tick\" \"tock\"	{"tick","tock"}

InvokeEvent 物件的 currentDirectory 屬性包含 File 物件，代表從中啟動應用程式的目錄位置。

若是開啟應用程式已註冊的檔案類型導致叫用應用程式，命令列引數即包括該檔案的原生路徑字串 (您的應用程式會負責開啟檔案，或對檔案執行預期的作業)。同樣地，如果應用程式設計成會更新其本身 (而非依賴標準 AIR 更新使用者介面)，則當使用者按兩下具有相符應用程式 ID 的應用程式所在的 AIR 檔時，就會包括該 AIR 檔的原生路徑。

您可以使用 currentDirectory File 物件的 resolve() 方法來存取檔案：

```
if((invokeEvent.currentDirectory != null)&&(invokeEvent.arguments.length > 0)){
    dir = invokeEvent.currentDirectory;
    fileToOpen = dir.resolvePath(invokeEvent.arguments[0]);
}
```

您應同時驗證其中的引數是否確實為檔案路徑。

範例：叫用事件記錄

Adobe AIR 1.0 以及更新的版本

下列範例會說明如何為 invoke 事件註冊偵聽程式以及處理該事件。此範例會記錄所有已接收的叫用事件，並顯示目前的目錄和命令列引數。

ActionScript 範例

```
package
{
    import flash.display.Sprite;
    import flash.events.InvokeEvent;
    import flash.desktop.NativeApplication;
    import flash.text.TextField;

    public class InvokeEventLogExample extends Sprite
    {
        public var log:TextField;

        public function InvokeEventLogExample()
        {
            log = new TextField();
            log.x = 15;
            log.y = 15;
            log.width = 520;
            log.height = 370;
            log.background = true;

            addChild(log);

            NativeApplication.nativeApplication.addEventListener(InvokeEvent.INVOKE, onInvoke);
        }

        public function onInvoke(invokeEvent:InvokeEvent):void
        {
            var now:String = new Date().toTimeString();
            logEvent("Invoke event received: " + now);

            if (invokeEvent.currentDirectory != null)
            {
                logEvent("Current directory=" + invokeEvent.currentDirectory.nativePath);
            }
            else
            {
```

```
                logEvent("--no directory information available--");
            }

            if (invokeEvent.arguments.length > 0)
            {
                logEvent("Arguments: " + invokeEvent.arguments.toString());
            }
            else
            {
                logEvent("--no arguments--");
            }
        }

        public function logEvent(entry:String):void
        {
            log.appendText(entry + "\n");
            trace(entry);
        }
    }
}
```

Flex 範例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
    invoke="onInvoke(event)" title="Invocation Event Log">
    <mx:Script>
    <![CDATA[
    import flash.events.InvokeEvent;
    import flash.desktop.NativeApplication;

    public function onInvoke(invokeEvent:InvokeEvent):void {
        var now:String = new Date().toTimeString();
        logEvent("Invoke event received: " + now);

        if (invokeEvent.currentDirectory != null){
            logEvent("Current directory=" + invokeEvent.currentDirectory.nativePath);
        } else {
            logEvent("--no directory information available--");
        }

        if (invokeEvent.arguments.length > 0){
            logEvent("Arguments: " + invokeEvent.arguments.toString());
        } else {
            logEvent("--no arguments--");
        }
    }

    public function logEvent(entry:String):void {
        log.text += entry + "\n";
        trace(entry);
    }
    ]]>
    </mx:Script>
    <mx:TextArea id="log" width="100%" height="100%" editable="false"
        valueCommit="log.verticalScrollPosition=log.textHeight;"/>
</mx:WindowedApplication>
```

於使用者登入時叫用 AIR 應用程式

Adobe AIR 1.0 以及更新的版本

AIR 應用程式可藉由將 NativeApplication 的 startAtLogin 屬性設定為 true，設定成目前的使用者登入時自動啟動。一旦完成設定，每當使用者登入便會自動啟動應用程式。應用程式將持續在登入時啟動，直到此設定變更為 false、使用者透過作業系統手動變更設定，或是解除安裝應用程式為止。登入時啟動是執行階段的設定。此設定僅會套用至目前的使用者。應用程式必須已安裝完成，才能成功地將 startAtLogin 屬性設定為 true。如果尚未安裝應用程式 (例如，應用程式是由 ADL 啟動) 卻設定了該屬性，便會擲回錯誤。

備註：應用程式不會在電腦系統開機時啟動，而是會隨著使用者登入而啟動。

若要判斷應用程式是自動啟動還是由使用者的動作而啟動，可檢查 InvokeEvent 物件的 reason 屬性。若屬性等於 InvokeEventReason.LOGIN，表示應用程式為自動啟動。對於其他叫用路徑，reason 屬性是以下列方式設定：

- InvokeEventReason.NOTIFICATION (僅 iOS) - 應用程式是透過 APNs 叫用。如需 APNs 的詳細資訊，請參閱使用推送通知。
- InvokeEventReason.OPEN_URL - 應用程式是由另一個應用程式或由系統叫用。
- InvokeEventReason.Standard - 所有其他情況。

若要存取 reason 屬性，則應用程式的目標必須是 AIR 1.5.1 或更新版本 (請於應用程式描述器檔案中設定正確的命名空間值)。

下列簡化的應用程式使用 InvokeEvent 的 reason 屬性，決定發生叫用事件時的反應。若 reason 屬性為「login」，則應用程式會保持在背景。否則，主應用程式會是可見狀態。使用此模式的應用程式一般會於登入時啟動，以完成背景處理或事件監控，並針對使用者觸發的叫用事件開啟視窗。

```
package {
    import flash.desktop.InvokeEventReason;
    import flash.desktop.NativeApplication;
    import flash.display.Sprite;
    import flash.events.InvokeEvent;

    public class StartAtLogin extends Sprite
    {
        public function StartAtLogin()
        {
            try
            {
                NativeApplication.nativeApplication.startAtLogin = true;
            }
            catch ( e:Error )
            {
                trace( "Cannot set startAtLogin:" + e.message );
            }

            NativeApplication.nativeApplication.addEventListener( InvokeEvent.INVOKE, onInvoke );
        }

        private function onInvoke( event:InvokeEvent ):void
        {
            if( event.reason == InvokeEventReason.LOGIN )
            {
                //do background processing...
                trace( "Running in background..." );
            }
            else
            {
                this.stage.nativeWindow.activate();
            }
        }
    }
}
```

備註：若要檢視行為的差異，請封裝並安裝應用程式。startAtLogin 屬性只能在已安裝的應用程式中設定。

從瀏覽器叫用 AIR 應用程式

Adobe AIR 1.0 以及更新的版本

網站可以利用瀏覽器叫用功能，從瀏覽器啟動已安裝的 AIR 應用程式。唯有當應用程式描述器檔案將 allowBrowserInvocation 設定為 true 時，才允許使用瀏覽器叫用功能：

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

透過瀏覽器叫用應用程式時，應用程式的 NativeApplication 物件會傳送 BrowserInvokeEvent 事件。

若要接收 BrowserInvokeEvent 事件，請在 AIR 應用程式中呼叫 NativeApplication 物件 (NativeApplication.nativeApplication) 的 addEventListener() 方法。為 BrowserInvokeEvent 事件註冊事件偵聽程式時，此偵聽程式也將接收註冊前發生的所有 BrowserInvokeEvent 事件。這些事件會在呼叫 addEventListener() 而有傳回動作之後傳送，但未必會比註冊之後可能接收的其它 BrowserInvokeEvent 事件先傳送。這讓您得以處理任何在初始化程式碼執行 (例如，一開始從瀏覽器叫用應用程式時) 之前就已發生的 BrowserInvokeEvent 事件。請牢記，如果您稍後在執行時 (初始化應用程式之後) 加入事件偵聽程式，此偵聽程式仍將會接收應用程式啟動之後發生的所有 BrowserInvokeEvent 事件。

BrowserInvokeEvent 物件包含下列屬性：

屬性	說明
arguments	要傳遞至應用程式的引數 (字串) 陣列。
isHTTPS	瀏覽器中的內容是使用 https URL 配置 (true) 或不使用 (false)。
isUserEvent	瀏覽器叫用是否因使用者事件 (例如按下滑鼠) 所造成。在 AIR 1.0 中，此值始終設定為 true，因為 AIR 瀏覽器叫用功能需要透過使用者事件。
sandboxType	瀏覽器中內容的安全執行程序類型。有效的定義值與 Security.sandboxType 屬性所能使用的值相同，可為下列其中一項： • Security.APPLICATION — 內容位於應用程式安全執行程序中。 • Security.LOCAL_TRUSTED — 內容位於 「具有檔案系統的本機」安全執行程序中。 • Security.LOCAL_WITH_FILE — 內容在 「具有檔案系統的本機」安全執行程序中。 • Security.LOCAL_WITH_NETWORK — 內容位於 「具有網路連線的本機」安全執行程序中。 • Security.REMOTE — 內容位於遠端 (網路) 網域中。
securityDomain	瀏覽器中內容的安全性網域，例如 "www.adobe.com" 或 "www.example.org"。這項屬性設定僅適用於遠端安全執行程序中的內容 (來自網路網域的內容)，而不適用於本機或應用程式安全執行程序中的內容。

如果您要使用瀏覽器叫用功能，請務必考量安全性方面的隱憂。當網站啟動 AIR 應用程式時，可以透過 BrowserInvokeEvent 物件的 arguments 屬性來傳送資料。任何機密性的作業 (例如呼叫 API 載入檔案或程式碼) 均應謹慎使用這份資料。風險程度取決於應用程式處理資料的方式。如果您只希望讓特定網站能夠叫用應用程式，應用程式就該檢查 BrowserInvokeEvent 物件的 securityDomain 屬性。您也可以檢查 BrowserInvokeEvent 物件的 isHTTPS 屬性，要求叫用應用程式的網站必須使用 HTTPS。

應用程式應該驗證所有傳入的資料。例如，應用程式若預期將傳入特定網站的 URL，即應查驗這些 URL 確實指向該網域。這可防止攻擊者誘騙應用程式代為傳送機密資料。

應用程式所使用的 BrowserInvokeEvent 引數絕不該指向本機資源。例如，應用程式切勿根據由瀏覽器傳入的路徑來建立 File 物件。若必須從瀏覽器傳入遠端路徑，應用程式就該確認這些路徑都已採用遠端通訊協定而非 file:// 通訊協定。

終止應用程式

Adobe AIR 1.0 以及更新的版本

終止應用程式最快速的方式，就是呼叫 NativeApplication 的 exit() 方法。若您的應用程式沒有要儲存的資料，或要清除的外部資源，此方法的運作情況會非常良好。呼叫 exit() 將會關閉所有視窗，接著終止應用程式。不過，若要讓應用程式的視窗或其它組件能夠阻斷終止程序，以便儲存重要資料或執行清理工作，請於呼叫 exit() 之前傳送適當的警告事件。

妥善關閉應用程式的另一項考量是：不論關閉程序如何啟動，均提供單一執行路徑。使用者 (或作業系統) 可以透過下列方式，促使應用程式終止：

- 當 NativeApplication.nativeApplication.autoExit 為 true 時，關閉最後一個應用程式視窗。
- 從作業系統中選取應用程式結束命令；例如，當使用者從預設選單選擇結束應用程式命令時。(只適用於 Mac OS；Windows 和 Linux 的系統顏色配置並未提供應用程式結束命令。)
- 將電腦關機。

透過上列任一途徑由作業系統傳達結束命令時，NativeApplication 會傳送 exiting 事件。如果沒有任何偵聽程式取消 exiting 事件，則會關閉所有開啟的視窗。每個視窗都會傳送 closing 事件，再接著傳送 close 事件。若有任何視窗取消了 closing 事件，關閉程序便會停止。

如果您的應用程式注重視窗關閉順序，請自行偵聽來自 NativeApplication 的 exiting 事件，並依正確順序關閉各視窗。若您的文件視窗有工具面板 (舉例說明)，可能必須這麼做。萬一在系統關閉面板之後，使用者又決定儲存某些資料而要取消結束命令，這將導致不便或更糟的情況。在 Windows 上，會等到最後一個視窗關閉後才發生 exiting 事件 (當 NativeApplication 物件的 autoExit 屬性是設定為 true 時)。

為了能在所有平台上提供一致的行為，不論結束序列的起始是透過作業系統顏色配置、選單命令或應用程式邏輯，都請遵循下列最佳做法來結束應用程式：

1 應用程式的程式碼一律會先透過 NativeApplication 物件傳送 exiting 事件，並且確認應用程式的其它組件均未取消該事件後，再呼叫 exit() 方法。

```
public function applicationExit():void {
    var exitingEvent:Event = new Event(Event.EXITING, false, true);
    NativeApplication.nativeApplication.dispatchEvent(exitingEvent);
    if (!exitingEvent.isDefaultPrevented()) {
        NativeApplication.nativeApplication.exit();
    }
}
```

2 偵聽來自 NativeApplication.nativeApplication 物件的 exiting 應用程式事件，並且在處理常式中關閉任何視窗 (先傳送 closing 事件)。當所有視窗都已關閉後，執行任何必要的清理工作，例如儲存應用程式資料或刪除暫存檔。清理過程中，請一律使用同步方法，以確保在應用程式結束前即清理完成。

如果視窗關閉順序並不重要，您就可以循序處理 NativeApplication.nativeApplication.openedWindows 陣列，並逐一關閉每個視窗。假使您 「必須」顧及順序，則請提供適當方式，依正確順序關閉各視窗。

```
private function onExiting(exitingEvent:Event):void {
    var winClosingEvent:Event;
    for each (var win:NativeWindow in NativeApplication.nativeApplication.openedWindows) {
        winClosingEvent = new Event(Event.CLOSING,false,true);
        win.dispatchEvent(winClosingEvent);
        if (!winClosingEvent.isDefaultPrevented()) {
            win.close();
        } else {
            exitingEvent.preventDefault();
        }
    }

    if (!exitingEvent.isDefaultPrevented()) {
        //perform cleanup
    }
}
```

3 各視窗應該負責偵聽自己的 closing 事件，藉以處理各自的清理工作。

4 每個應用程式中僅能使用一個 exiting 偵聽程式，因為稍早呼叫的處理常式無從得知後續的處理常式是否會取消 exiting 事件 (依照執行順序進行並不可靠)。

第 51 章 使用 AIR 執行階段和作業系統資訊

Adobe AIR 1.0 以及更新的版本

本章節將討論 AIR 應用程式管理作業系統檔案關聯、偵測使用者活動，以及取得 Adobe® AIR® 執行階段相關資訊的方法。

更多說明主題
flash.desktop.NativeApplication

管理檔案關聯

Adobe AIR 1.0 以及更新的版本

您必須在應用程式描述器中宣告應用程式和檔案類型之間的關聯。在安裝期間，AIR 應用程式的安裝程式會使 AIR 應用程式與每一個宣告的檔案類型產生關聯，除非有其它應用程式為預設的開啟應用程式，否則會讓 AIR 應用程式做為預設的開啟應用程式。AIR 應用程式安裝程序不會覆寫現有的檔案類型關聯。若要從其它應用程式接管該項關聯，請在執行階段呼叫 NativeApplication.setAsDefaultApplication() 方法。

確認所需的檔案關聯在您的應用程式啟動時都會就定位，是一種較好的作法。因為 AIR 應用程式的安裝程式不會覆寫現有的檔案關聯，而且使用者系統中的檔案關聯隨時都可能改變。當其它應用程式具有目前的檔案關聯時，事先詢問使用者後再接管現有的關聯也是有禮貌的作法。

下列 NativeApplication 類別的方法可以讓應用程式管理檔案關聯，每一個方法都會採用五種類型的副檔名做為參數：

方法	說明
isSetAsDefaultApplication()	如果 AIR 應用程式目前與指定的檔案類型有關聯，則傳回 true。
setAsDefaultApplication()	建立 AIR 應用程式和檔案類型的開啟動作之間的關聯。
removeAsDefaultApplication()	移除 AIR 應用程式和檔案類型之間的關聯。
getDefaultApplication()	報告目前與檔案類型有關聯之應用程式的路徑。

AIR 只能管理原本在應用程式描述器中宣告的檔案類型關聯。您無法取得未經宣告的檔案類型關聯，就算使用者已經手動建立檔案類型和應用程式之間的關聯也一樣。如果呼叫任何檔案類型管理方法時，提供的副檔名屬於應用程式描述器中未宣告的檔案類型，應用程式就會擲出執行階段例外。

取得執行階段版本的修補程式層級

Adobe AIR 1.0 以及更新的版本

NativeApplication 物件具有 runtimeVersion 屬性，這個屬性是應用程式在其中執行的執行階段版本 (字串，例如 "1.0.5")。NativeApplication 物件也具有 runtimePatchLevel 屬性，這個屬性是執行階段的修補程式層級 (數字，例如 2960)。下列程式碼會使用這些屬性：

```
trace(NativeApplication.nativeApplication.runtimeVersion);
trace(NativeApplication.nativeApplication.runtimePatchLevel);
```

偵測 AIR 功能

Adobe AIR 1.0 以及更新的版本

對於已合併至 Adobe AIR 應用程式的檔案，Security.sandboxType 屬性會設定為 Security.APPLICATION 常數所定義的值。您可以根據 Adobe AIR 安全執行程序中是否有檔案載入內容 (此內容不一定包含 AIR 專屬 API)，如下列程式碼所示範：

```
if (Security.sandboxType == Security.APPLICATION)
{
    // Load SWF that contains AIR APIs
}
else
{
    // Load SWF that does not contain AIR APIs
}
```

未與 AIR 應用程式一併安裝的所有資源都會指定給與 Adobe® Flash® Player 在網頁瀏覽器中所指定的相同安全執行程序。遠端資源會根據其來源網域放置於不同的安全執行程序中；而本機資源則會放置於 「具有網路連線的本機」、「具有檔案系統的本機」或 「本機信任」的安全執行程序中。

您可以檢查 Capabilities.playerType 是否設定為 "Desktop"，以得知是否要在執行階段中 (而且不是在於瀏覽器中執行的 Flash Player 內) 執行內容。

如需詳細資訊，請參閱：第 928 頁 「AIR 安全性」。

追蹤使用者狀態

Adobe AIR 1.0 以及更新的版本

NativeApplication 物件會傳送兩個事件，協助您偵測使用者是否正在使用電腦。在 NativeApplication.idleThreshold 屬性設定的時間間隔內，如果沒有偵測到任何滑鼠或鍵盤活動，NativeApplication 就會傳送 userIdle 事件。當下次發生鍵盤或滑鼠輸入活動時，NativeApplication 物件就會傳送 userPresent 事件。idleThreshold 時間間隔是以秒為單位來量測，預設值為 300 (5 分鐘)。您也可以透過 NativeApplication.nativeApplication.lastUserInput 屬性，取得上次使用者輸入活動的秒數。

下列幾行程式碼會將閒置臨界值設定為 2 分鐘，並且偵聽 userIdle 和 userPresent 事件：

```
NativeApplication.nativeApplication.idleThreshold = 120;
NativeApplication.nativeApplication.addEventListener(Event.USER_IDLE, function(event:Event) {
    trace("Idle");
});
NativeApplication.nativeApplication.addEventListener(Event.USER_PRESENT, function(event:Event) {
    trace("Present");
});
```

備註：在任兩個 userPresent 事件之間，只會傳送一個 userIdle 事件。

第 52 章 使用 AIR 原生視窗

Adobe AIR 1.0 以及更新的版本

您可以使用由 Adobe® AIR® 原生視窗 API 所提供的類別來建立及管理桌面視窗。

AIR 中的原生視窗基本概念

Adobe AIR 1.0 以及更新的版本

如需瞭解在 AIR 中使用原生視窗的快速說明與程式碼範例，請參閱 Adobe Developer Connection 中的快速入門文章：

- 建立透明視窗應用程式 (Flex)
- 與視窗互動 (Flex)
- 自訂原生視窗的外觀與感覺 (Flex)
- 啟動視窗 (Flex)
- 建立土司樣式的視窗 (Flex)
- 控制視窗的顯示順序 (Flex)
- 建立可以調整大小、非矩形的視窗 (Flex)
- 與視窗互動 (Flash)
- 自訂原生視窗的外觀與感覺 (Flash)
- 建立土司樣式的視窗 (Flash)
- 控制視窗的顯示順序 (Flash)
- 建立可以調整大小、非矩形的視窗 (Flash)

AIR 提供容易使用的跨平台視窗 API，可供使用 Flash®、Flex™ 和 HTML 程式設計技巧，建立原生作業系統視窗。

有了 AIR，在開發應用程式的外觀時，您可具備寬廣的自由發揮空間。您可以讓所建立的視窗可以在外觀上接近標準的桌面應用程式：在 Mac 上執行時與 Apple 的樣式相符、在 Windows 上執行時與 Microsoft 傳統慣例一致，在 Linux 上又能與視窗管理員產生協調。而且，您完全不需要針對平台來插入專用的程式碼便能夠達成此目的。或者，您也可以使用由 Flex 架構所提供的可設定外觀與可擴充顏色來自行建立樣式，無論應用程式是在哪種系統上執行都沒問題。甚至，您還可以使用向量和點陣圖圖案，自行繪製視窗的顏色，並完整支援桌面的透明度與 Alpha 混合效果。看膩了矩形視窗嗎？何不使用圓角視窗呢？

AIR 中的視窗

Adobe AIR 1.0 以及更新的版本

AIR 支援三種視窗專用的 API：

- ActionScript 導向的 NativeWindow 類別可提供最低層級的視窗 API。NativeWindows 可用於以 ActionScript 和 Flash Professional 編寫的應用程式中。請考慮擴充 NativeWindow 類別，對應用程式中使用的視窗進行特製。
- 在 HTML 環境中，您可以使用 JavaScript Window 類別，使用方法與您在瀏覽器類型的網頁應用程式中一樣。對 JavaScript Window 方法進行的呼叫會轉送至基礎的原生視窗物件。

- Flex 架構的 mx:WindowedApplication 和 mx:Window 類別為 NativeWindow 類別提供了一個 Flex「包裝函式」。當您以 Flex 建立 AIR 應用程式時，WindowedApplication 組件會取代 Application 組件，並且會一直做為 Flex 應用程式中的初始視窗。

ActionScript 視窗

當您使用 NativeWindow 類別建立視窗時，請直接使用 Flash Player 舞台和顯示清單。若要將視覺物件加入 NativeWindow，請將物件加入視窗舞台的顯示清單或舞台上的另一個顯示物件容器。

HTML 視窗

當您建立 HTML 視窗時，會使用 HTML、CSS 和 JavaScript 來顯示內容。若要將視覺物件加入 HTML 視窗，可以將該內容加入 HTML DOM。HTML 視窗是特殊類別的 NativeWindow。AIR 主機會在 HTML 視窗中定義 nativeWindow 屬性，以便存取基礎 NativeWindow 實體。您可以使用這個屬性來存取本節說明的 NativeWindow 屬性、方法和事件。

備註：JavaScript Window 物件也有方法可用來編寫所包含視窗的指令碼，例如 moveTo() 和 close()。如果有不同方法可供使用時，您可以使用最方便的方法。

Flex 架構視窗

當您使用 Flex 架構建立視窗時，一般會使用 MXML 組件來填滿視窗。若要加入 Flex 組件至視窗，請將組件元素加入視窗 MXML 定義中。您也可以使用 ActionScript，以動態方式加入內容。mx:WindowedApplication 和 mx:Window 組件是設定為 Flex 容器，因此能夠直接接受 Flex 組件，而 NativeWindow 物件則不行。必要時，可以使用 nativeWindow 屬性，透過 WindowedApplication 和 Window 物件，存取 NativeWindow 屬性和方法。

初始應用程式視窗

應用程式的第一個視窗是由 AIR 為您自動建立的。AIR 會使用應用程式描述器檔案的 initialWindow 元素中所指定參數，設定視窗的屬性和內容。

如果根內容是 SWF 檔，AIR 會建立 NativeWindow 實體，載入 SWF 檔，然後將其加入至視窗舞台。如果根內容是 HTML 檔，AIR 會建立 HTML 視窗並載入 HTML。

原生視窗類別

Adobe AIR 1.0 以及更新的版本

原生視窗 API 包含下列類別：

套件	類別
flash.display	• NativeWindow • NativeWindowInitOptions • NativeWindowDisplayState • NativeWindowResize • NativeWindowSystemChrome • NativeWindowType
flash.events	• NativeWindowBoundsEvent • NativeWindowDisplayStateEvent

原生視窗事件流程

Adobe AIR 1.0 以及更新的版本

原生視窗會傳送事件，以通知需要知道的組件，將發生或已發生重要變更。許多視窗相關事件是成對傳送。第一個事件會警告即將發生變更。第二個事件會宣佈已發生變更。您可以取消警告事件，但不能取消通知事件。下面依序列出當使用者按一下視窗的最大化按鈕時所發生的事件流程：

1. NativeWindow 物件傳送 displayStateChanging 事件。
2. 如果沒有已登錄偵聽程式取消事件，視窗就會最大化。
3. NativeWindow 物件傳送 displayStateChanging 事件。

   此外，NativeWindow 物件也會傳送視窗大小和位置的相關變更事件。視窗不會傳送這些相關變更的警告事件。相關事件包括：

   a. 如果視窗左上角因最大化作業而移動，就會傳送 move 事件。
   b. 如果視窗大小因為最大化作業而變更，就會傳送 resize 事件。

   NativeWindow 物件會在視窗最小化、還原、關閉、移動和調整大小時，傳送類似的事件序列。

   警告事件只有在透過視窗顏色或其它作業系統控制機制啟動變更時才會傳送。當您呼叫視窗方法來變更視窗大小、位置，或顯示狀態時，視窗只會傳送事件來宣佈變更。想要的話，您可以使用 dispatchEvent() 方法傳送警告事件，以檢查您的警告事件是否已被取消，再處理變更。

   如需視窗 API 類別、方法、內容以及事件的詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。

控制原生視窗樣式和行為方式的屬性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列屬性可控制視窗的基本外觀和行為：

- type
- systemChrome
- transparent
- owner

當您建立視窗時，要在傳遞給視窗建構函式的 NativeWindowInitOptions 物件上設定這些屬性。AIR 會從應用程式描述器讀取初始應用程式視窗的屬性 (但 type 屬性除外，此屬性無法在應用程式描述器中設定，而且永遠設定為 normal)。視窗建立後無法變更這些屬性。

這些屬性中有些設定是彼此互不相容的：如果 transparent 是 true 或 type 是 lightweight，則 systemChrome 無法設定為 standard。

視窗類型

Adobe AIR 1.0 以及更新的版本

AIR 視窗類型是結合原生作業系統的顏色和可見性特質，建立三種功能性視窗類型。請使用定義於 NativeWindowType 類別中的常數，參考程式碼中的類型名稱。AIR 提供下列視窗類型：

類型	說明
一般	典型的視窗。一般視窗會使用完整尺寸樣式的色調，並顯示在 Windows 工作列和 Mac OS X 視窗選單中。
公用程式	工具面板。公用程式視窗會使用精簡版的系統顏色，而且不會顯示在 Windows 工作列和 Mac OS X 視窗選單中。
輕量型	輕量型視窗沒有顏色，而且不會顯示在 Windows 工作列和 Mac OS X 視窗選單中。此外，在 Windows 中，輕量型視窗也沒有「系統」(Alt+ 空白鍵) 選單。輕量型視窗適合當做通知對話方塊和控制項使用，例如開啟短暫顯示區域的下拉式清單方塊。使用輕量型 type 時，必須將 systemChrome 設定為 none。

視窗顏色

Adobe AIR 1.0 以及更新的版本

視窗顏色是一組控制項，可以讓使用者在桌面環境中操控視窗。顏色元素包括標題列、標題列按鈕、邊框，和調整大小移駐夾。

系統顏色

您可以將 systemChrome 屬性設定為 standard 或 none。選擇 standard 系統顏色可以提供您的視窗一組由使用者的作業系統所建立並訂定樣式的標準控制項。選擇 none 可將您的自訂顏色提供給視窗。請使用定義於 NativeWindowSystemChrome 類別中的常數，參考程式碼中設定的系統顏色設定。

系統顏色是由系統進行管理。您的應用程式無法直接存取控制項本身，但是可以在使用控制項時回應已傳送的事件。當您使用標準的視窗顏色時，transparent 屬性必須設定為 false，而且 type 屬性必須為 normal 或 utility。

Flex 顏色

當您使用 Flex WindowedApplication 或 Window 組件時，視窗可以使用系統顏色或由 Flex 架構提供的顏色。若要使用 Flex 顏色，請將用來建立視窗的 systemChrome 屬性設定為 none。使用 Flex 4 Spark 組件，而不使用 MX 組件時，您必須指定外觀類別，才能使用 Flex 顏色。您可以使用內建外觀或提供自己的外觀。下列範例示範如何使用內建 Spark WindowedApplication 外觀類別來提供視窗顏色：

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx">
<fx:Style>
@namespace "library://ns.adobe.com/flex/spark";
WindowedApplication
{
    skinClass:ClassReference("spark.skins.spark.SparkChromeWindowedApplicationSkin");
}
</fx:Style>
</s:WindowedApplication>
```

如需詳細資訊，請參閱 Using Flex 4: About the AIR window containers: Controlling window chrome

自訂顏色

當您沒有使用系統顏色建立視窗時，就必須加入自訂的顏色控制項來處理使用者與視窗之間的互動作業。您也可以自由地建立透明、非矩形的視窗。

若要使用自訂顏色來搭配 mx:WindowedApplication 或 mx:Window 組件，則必須將 showFlexChrome 樣式設為 false。否則，Flex 會將自己的顏色加入視窗。

視窗透明度

Adobe AIR 1.0 以及更新的版本

若要允許使用桌面或其它視窗的視窗 Alpha 混合，請將視窗 transparent 屬性設定為 true。您必須先設定 transparent 屬性，才能建立視窗，而該屬性建立後無法再變更。

透明視窗沒有預設背景。應用程式所繪製且未包含物件的任何視窗區域是看不見的。如果顯示的物件具有小於一的 Alpha 設定值，則物件之下的任何項目 (包括相同視窗、其它視窗以及桌面中的其它顯示物件) 都會穿透顯示出來。

如果您所要建立的應用程式，其邊框的形狀不規則或 「淡化」或看起來是隱形的，那麼透明視窗就很有用。不過，顯示大型 Alpha 混合區域的速度可能會很慢，因此請謹慎使用這種效果。

重要事項：在 Linux 上，完全透明的像素不會傳遞滑鼠事件。您應該避免建立具有大塊完全透明區域的視窗，因為可能會阻礙使用者對其它視窗或桌面項目的存取。在 Mac OS X 和 Windows 上，完全透明的像素則可傳遞滑鼠事件。

透明度不能用於具有系統顏色的視窗。此外，HTML 中的 SWF 和 PDF 內容可能不會顯示於透明視窗。如需詳細資訊，請參閱第 866 頁 「在 HTML 網頁中載入 SWF 或 PDF 內容時必須考量的事項」。

靜態 NativeWindow.supportsTransparency 屬性會報告視窗透明度是否可供使用。透明度不受支援時，應用程式就會與黑色背景混合。在這些情況下，應用程式中任何的透明區域會顯示為不透明的黑色。最好在此屬性的測試結果為 false 的情況下提供補救措施。例如，您可以向使用者顯示警告對話方塊，或顯示不透明的矩形使用者介面。

請注意，Mac 和 Windows 作業系統永遠都支援透明度。Linux 作業系統上的支援需要複合式視窗管理員，但即使當複合式視窗管理員在作用中，也可能因為使用者顯示選項或硬體組態的關係而無法產生透明度。

MXML 應用程式視窗中的透明度

Adobe AIR 1.0 以及更新的版本

MXML 視窗的背景是預設為不透明的，即使您將視窗建立為 transparent 也是如此 (請注意視窗角落的透明度效果)。若要呈現視窗的透明背景，請在樣式表或應用程式 MXML 檔內所包含的 <mx:Style> 元素中設定背景顏色和 Alpha 值。例如，下列樣式宣告會讓背景呈現稍帶透明的綠色：

```
WindowedApplication
{
    background-alpha:".8";
    background-color:"0x448234";
}
```

HTML 應用程式視窗中的透明度

Adobe AIR 1.0 以及更新的版本

根據預設，顯示於 HTML 視窗和 HTMLLoader 物件中的 HTML 內容背景是不透明的，即使包含的視窗是透明的也是如此。若要關閉為 HTML 內容顯示的預設背景，請將 paintsDefaultBackground 屬性設定為 false。下列範例會建立 HTMLLoader，並關閉預設背景：

```
var htmlView:HTMLLoader = new HTMLLoader();
htmlView.paintsDefaultBackground = false;
```

此範例會使用 JavaScript 來關閉 HTML 視窗的預設背景：

```
window.htmlLoader.paintsDefaultBackground = false;
```

如果 HTML 文件中的元素會設定背景顏色，該元素的背景就不是透明的。不支援設定部分透明度 (或不透明度) 值。但是，您可以使用透明 PNG 格式圖形做為頁面或頁面元素的背景，以達到類似的視覺效果。

視窗所有權

一個視窗可以「擁有」一或多個其他視窗。這些被擁有的視窗永遠都會顯示在主視窗前面，會隨主視窗一起縮小和還原，並且會在主視窗關閉時一起關閉。視窗所有權無法轉移至其他視窗或予以移除。一個視窗只能由一個主視窗擁有，但是可以擁有任何數目的其他視窗。

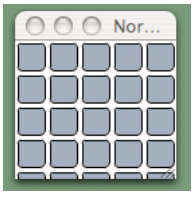
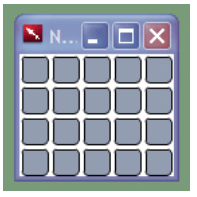
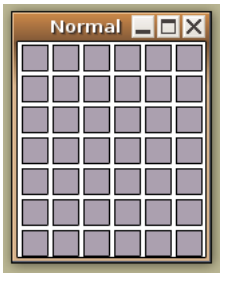
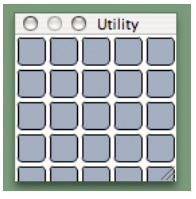
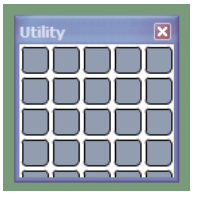
您可以使用視窗擁有權，讓管理工具色盤與對話方塊所使用的視窗使更輕鬆。例如，如果您顯示與文件視窗關聯的「儲存」對話方塊，使文件視窗擁有對話方塊，可讓對話方塊自動保持在文件視窗的前面。

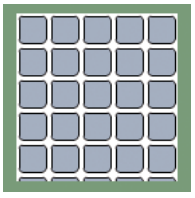
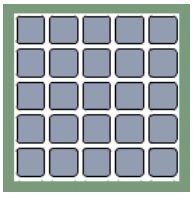
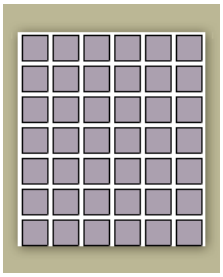
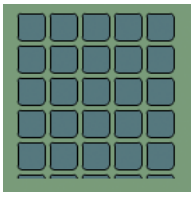
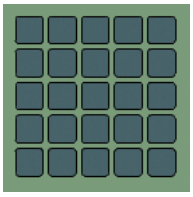
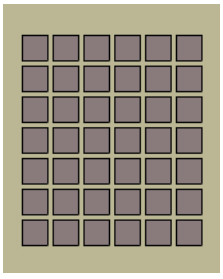
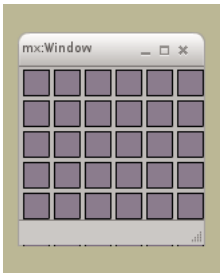
- NativeWindow.owner
- Christian Cantrell：AIR 2.6 中擁有的視窗 (英文)

視覺視窗類別目錄

Adobe AIR 1.0 以及更新的版本

下表說明了在 Mac OS X、Windows 和 Linux 作業系統上不同視窗屬性設定組合的視覺效果：

視窗設定	Mac OS X	Microsoft Windows	Linux*
Type: normal SystemChrome: standard Transparent: false			
Type: utility SystemChrome: standard Transparent: false			

視窗設定	Mac OS X	Microsoft Windows	Linux*
Type: 任何值 SystemChrome: none Transparent: false			
Type: 任何值 SystemChrome: none Transparent: true			
mxWindowedApplication 或 mx:Window Type: 任何值 SystemChrome: none Transparent: true			

*Ubuntu 和 Compiz 視窗管理員

備註：AIR 不支援下列系統顏色元素：Mac OS X 工具列、Mac OS X Proxy 圖示、Windows 標題列圖示，以及替代系統顏色。

建立視窗

Adobe AIR 1.0 以及更新的版本

AIR 會自動為應用程式建立第一個視窗，但是您可以建立所需的其它任何視窗。若要建立原生視窗，請使用 NativeWindow 建構函式方法。

若要建立 HTML 視窗，請使用 HTMLLoader createRootWindow() 方法，或是從 HTML 文件呼叫 JavaScript window.open() 方法。建立的視窗是 NativeWindow 物件，其顯示清單會包含 HTMLLoader 物件。HTMLLoader 物件會解譯並顯示視窗的 HTML 和 JavaScript 內容。您可以使用 window.nativeWindow 屬性，從 JavaScript 存取基礎 NativeWindow 物件的屬性。(在 AIR 應用程式安全執行程序中執行的程式碼才能存取此屬性)。

當您初始化視窗 (包括初始應用程式視窗) 時，應該考慮建立狀態為不可見的視窗，並載入內容或執行任何圖形更新，然後讓視窗可見。這個順序可以防止使用者看到任何不舒服的視覺變更。您可以在應用程式描述器中指定 <visible>false</visible> 標籤 (或不要使用此標籤，因為 false 是預設值)，指定以不可見狀態建立應用程式的初始視窗。預設會看不到新的 NativeWindow。當您使用 HTMLLoader createRootWindow() 方法建立 HTML 視窗時，可以將 visible 引數設為 false。呼叫 NativeWindow activate() 方法或將 visible 屬性設為 true，讓視窗可見。

指定視窗初始化屬性

Adobe AIR 1.0 以及更新的版本

原生視窗的初始化屬性在桌面視窗建立之後就無法再變更。這些永遠不變的屬性及其預設值包括：

屬性	預設值
systemChrome	standard
type	normal
transparent	False
owner	null
maximizable	true
minimizable	true
resizable	true

請在應用程式描述器檔案中，為 AIR 所建立的初始視窗設定屬性。AIR 應用程式的主視窗的 type 屬性永遠都是 normal (其它視窗屬性可以在描述器檔案中指定，例如 visible、width 和 height，但這些屬性可以隨時變更)。

請使用 NativeWindowInitOptions 類別，為您應用程式所建立的其它原生和 HTML 視窗設定屬性。當您建立視窗時，必須傳遞 NativeWindowInitOptions 物件，將視窗屬性指定為 NativeWindow 建構函數或 HTMLLoader createRootWindow() 方法。

下列程式碼會為公用程式視窗建立 NativeWindowInitOptions 物件。

```
var options:NativeWindowInitOptions = new NativeWindowInitOptions();
options.systemChrome = NativeWindowSystemChrome.STANDARD;
options.type = NativeWindowType.UTILITY
options.transparent = false;
options.resizable = false;
options.maximizable = false;
```

無法支援在 transparent 為 true 或 type 為 lightweight 時，將 systemChrome 設定為 standard。

備註：您無法設定使用 JavaScript window.open() 函數所建立視窗的初始化屬性。但是，您可以透過自行實作 HTMLHost 類別來覆寫這些視窗的建立方式。如需詳細資訊，請參閱第 875 頁「處理 window.open() 的 JavaScript 呼叫」。

當您使用 Flex mx:Window 類別建立視窗時，請在 window 物件本身上 (在視窗的 MXML 宣告或建立視窗的程式碼中) 指定初始化屬性。直到呼叫 open() 方法之後，才會建立基礎 NativeWindow 物件。一旦開啟視窗之後，這些初始化屬性就無法再變更。

建立初始應用程式視窗

Adobe AIR 1.0 以及更新的版本

AIR 會根據應用程式描述器中指定的屬性，建立初始應用程式視窗，並載入內容元素中參考的檔案。內容元素必須參考 SWF 或 HTML 檔。

起始視窗可以是您應用程式的主視窗，也可以只用來啟動一個或多個其它視窗。您不一定需要讓它顯示。

使用 ActionScript 建立起始視窗

Adobe AIR 1.0 以及更新的版本

當您使用 ActionScript 來建立 AIR 應用程式時，應用程式的主要類別必須擴充 Sprite 類別 (或 Sprite 的子類別)。此類別會做為應用程式的主進入點。

當您的應用程式啟動時，AIR 會建立視窗、建立主類別的實體，然後將實體加入至視窗舞台。若要存取視窗，您可以偵聽 addedToStage 事件，然後使用 Stage 物件的 nativeWindow 屬性，以取得 NativeWindow 物件的參考。

下列範例說明使用 ActionScript 所建立之 AIR 應用程式，其主類別的基本骨架：

```
package {
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.events.Event;

    public class MainClass extends Sprite
    {
        private var mainWindow:NativeWindow;
        public function MainClass(){
            this.addEventListener(Event.ADDED_TO_STAGE, initialize);
        }

        private function initialize(event:Event):void{
            mainWindow = this.stage.nativeWindow;
            //perform initialization...
            mainWindow.activate(); //show the window
        }
    }
}
```

備註：技術上而言，您「可以」在主要類別的建構函式中存取 nativeWindow 屬性。不過，這是特殊情況，僅適用於初始應用程式視窗。

在 Flash Professional 中建立應用程式時，如果沒有在個別的 ActionScript 檔案中建立您自己的主文件類別，就會自動建立主文件類別。您可以使用舞台的 nativeWindow 屬性，存取初始視窗的 NativeWindow 物件。例如，下列程式碼會啟動主視窗為最大化狀態 (從時間軸)：

```
import flash.display.NativeWindow;

var mainWindow:NativeWindow = this.stage.nativeWindow;
mainWindow.maximize();
mainWindow.activate();
```

使用 Flex 建立起始視窗

Adobe AIR 1.0 以及更新的版本

使用 Flex 架構建立 AIR 應用程式時，請使用 mx:WindowedApplication 做為主 MXML 檔的根元素 (您可以使用 mx:Application 組件，但它並不支援 AIR 中提供的所有功能)。WindowedApplication 組件可做為應用程式的起始進入點。

當您啟動應用程式時，AIR 會建立原生視窗，初始化 Flex 架構，然後將 WindowedApplication 物件加入至視窗舞台。啟動序列完成後，WindowedApplication 會傳送 applicationComplete 事件。請使用 WindowedApplication 實體的 nativeWindow 屬性，存取桌面視窗物件。

下列範例會建立簡單的 WindowedApplication 組件，設定其 x 和 y 座標：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    applicationComplete="placeWindow()">
    <mx:Script>
        <![CDATA[
            private function placeWindow():void{
                this.nativeWindow.x = 300;
                this.nativeWindow.y = 300;
            }
        ]]>
    </mx:Script>
    <mx:Label text="Hello World" horizontalCenter="0" verticalCenter="0"/>
</mx:WindowedApplication>
```

建立 NativeWindow

Adobe AIR 1.0 以及更新的版本

若要建立 NativeWindow，請傳遞 NativeWindowInitOptions 物件給 NativeWindow 建構函式：

```
var options:NativeWindowInitOptions = new NativeWindowInitOptions();
options.systemChrome = NativeWindowSystemChrome.STANDARD;
options.transparent = false;
var newWindow:NativeWindow = new NativeWindow(options);
```

視窗要直到您將 visible 屬性設定為 true，或呼叫 activate() 方法，才會顯示。

建立視窗之後，您可以初始化其屬性，並使用 stage 屬性和 Flash 顯示清單技巧，將內容載入視窗之中。

幾乎在所有案例中，都應該將新原生視窗的 stage scaleMode 屬性設定為 noScale (使用 StageScaleMode.NO_SCALE 常數)。Flash 縮放模式是針對應用程式作者無法得知應用程式顯示空間比例的情形所設計的。縮放模式可以讓作者選擇缺點最少的妥協方案：裁剪內容、延伸或擠壓內容，或使用空白空間填塞。由於您是在 AIR (視窗框架) 中控制顯示空間，因此可以依內容調整視窗大小，或依視窗大小調整內容，而不必妥協。

Flex 和 HTML 視窗的縮放模式會自動設定為 noScale。

備註：若要決定在目前作業系統上允許的最大和最小視窗大小，請使用下列靜態 NativeWindow 屬性：

```
var maxOSSize:Point = NativeWindow.systemMaxSize;
var minOSSize:Point = NativeWindow.systemMinSize;
```

建立 HTML 視窗

Adobe AIR 1.0 以及更新的版本

若要建立 HTML 視窗，您可以呼叫 JavaScript Window.open() 方法，或呼叫 AIR HTMLLoader 類別 createRootWindow() 方法。

任何安全執行程序中的 HTML 內容都可以使用標準的 JavaScript Window.open() 方法。如果內容是在應用程式安全執行程序之外執行，則只能呼叫 open() 方法來回應使用者互動，例如按一下滑鼠或鍵盤。呼叫 open() 時，會建立具有系統顏色的視窗，以便在指定的 URL 顯示內容。例如：

```
newWindow = window.open("xmpl.html", "logWindow", "height=600, width=400, top=10, left=10");
```

備註：您可以擴充 ActionScript 中的 HTMLHost 類別，以自訂使用 JavaScript window.open() 函數建立的視窗。請參閱第 870 頁「關於擴充 HTMLHost 類別」。

應用程式安全執行程序中的內容可以存取功能更強大的視窗建立方法 HTMLLoader.createRootWindow()。透過這個方法，您就可以指定新視窗的所有建立選項。例如，下列 JavaScript 程式碼會建立輕量型視窗，且不使用大小為 300x400 像素的系統顏色。

```
var options = new air.NativeWindowInitOptions();
options.systemChrome = "none";
options.type = "lightweight";

var windowBounds = new air.Rectangle(200,250,300,400);
newHTMLLoader = air.HTMLLoader.createRootWindow(true, options, true, windowBounds);
newHTMLLoader.load(new air.URLRequest("xmpl.html"));
```

備註：如果由新視窗載入的內容位於應用程式安全執行程序之外，window 物件就不會具備以下 AIR 屬性：runtime、nativeWindow 或 htmlLoader。

如果建立透明視窗，則不會永遠顯示內嵌於載入該視窗 HTML 中的 SWF 內容。您必須將用來參考 SWF 檔案之 object 或 embed 標籤的 wmode 參數設定為 opaque 或 transparent。wmode 的預設值為 window，所以依照預設，SWF 內容不會顯示在透明視窗中。無論設定哪一個 wmode 值，PDF 內容都無法顯示在透明視窗中。(AIR 1.5.2 之前的版本，SWF 內容也無法顯示在透明視窗中。)

使用 createRootWindow() 方法建立的視窗依舊會獨立於開啟視窗之外。JavaScript Window 物件的 parent 和 opener 屬性會是 null。開啟的視窗可以使用由 createRootWindow() 函數所傳回的 HTMLLoader 參考，存取新視窗的 Window 物件。在上一個範例的情況下，陳述式 newHTMLLoader.window 會參考已建立視窗的 JavaScript Window 物件。

備註：從 JavaScript 和 ActionScript 都可以呼叫 createRootWindow() 函數。

建立 mx:Window

Adobe AIR 1.0 以及更新的版本

若要建立 mx:Window，您可以使用 mx:Window 做為根標籤來建立 MXML 檔，或直接呼叫 Window 類別建構函式。

下列範例會呼叫 Window 建構函式，建立並顯示 mx:Window：

```
var newWindow:Window = new Window();
newWindow.systemChrome = NativeWindowSystemChrome.NONE;
newWindow.transparent = true;
newWindow.title = "New Window";
newWindow.width = 200;
newWindow.height = 200;
newWindow.open(true);
```

加入內容至視窗

Adobe AIR 1.0 以及更新的版本

將內容加入至 AIR 視窗的方式取決於視窗的類型。例如，MXML 和 HTML 可以讓您以宣告方式來定義視窗的基本內容。您可以將資源嵌入應用程式 SWF 檔中，也可以從不同的應用程式檔案中載入資源。Flex、Flash 和 HTML 內容全都可以透過此方式建立，並以動態方式加入至視窗中。

當您載入 SWF 內容，或載入包含 JavaScript 的 HTML 內容時，必須將 AIR 安全性模型納入考量。應用程式安全執行程序中的任何內容 (也就是，與您的應用程式一起安裝的且可使用 app: URL 配置載入的內容) 都具備可存取所有 AIR API 的完整權限。從這個安全執行程序之外載入的任何內容都不能存取 AIR API。在應用程式安全執行程序之外的 JavaScript 內容不能夠使用 JavaScript Window 物件的 runtime、nativeWindow 或 htmlLoader 屬性。

若要允許安全的跨指令碼處理，您可以使用安全執行程序橋接，在應用程式內容與非應用程式內容之間提供有限的介面。在 HTML 內容中，您也可以將應用程式的頁面對應至非應用程式安全執行程序中，以允許頁面上的程式碼對外部內容進行跨指令碼處理。請參閱：第 928 頁「AIR 安全性」。

載入 SWF 檔或影像

您可以使用 flash.display.Loader 類別，將 Flash SWF 檔或影像載入原生視窗的顯示清單之中：

```
package {
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.display.Loader;

    public class LoadedSWF extends Sprite
    {
        public function LoadedSWF(){
            var loader:Loader = new Loader();
            loader.load(new URLRequest("visual.swf"));
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE,loadFlash);
        }

        private function loadFlash(event:Event):void{
            addChild(event.target.loader);
        }
    }
}
```

備註：使用 ActionScript 1 或 2 建立的舊版 SWF 檔案可共享全域狀態，例如類別定義、Singleton，如果這些檔案載入相同視窗，也會共享全域變數。如果這種 SWF 檔案要依賴未修改的全域狀態才能正確運作，它就不能重複載入相同視窗，也不能使用重疊的類別定義和變數，以另一個 SWF 檔的身分載入相同的視窗。此內容可以載入不同的視窗。

將 **HTML** 內容載入 **NativeWindow**

若要將 HTML 內容載入 NativeWindow，您可以將 HTMLLoader 物件加入視窗舞台，然後將 HTML 內容載入 HTMLLoader，或使用 HTMLLoader.createRootWindow() 方法建立已包含 HTMLLoader 物件的視窗。下列範例會將 HTML 內容顯示在原生視窗的舞台上，300 x 500 像素的顯示區域之中：

```
//newWindow is a NativeWindow instance
var htmlView:HTMLLoader = new HTMLLoader();
htmlView.width = 300;
htmlView.height = 500;

//set the stage so display objects are added to the top-left and not scaled
newWindow.stage.align = "TL";
newWindow.stage.scaleMode = "noScale";
newWindow.stage.addChild( htmlView );

//urlString is the URL of the HTML page to load
htmlView.load( new URLRequest(urlString) );
```

若要將 HTML 頁面載入 Flex 應用程式，您可以使用 Flex HTML 組件。

如果視窗使用透明度 (即視窗的 transparent 屬性為 true)，不會顯示 HTML 檔案中的 SWF 內容，除非用來參考 SWF 檔案之 object 或 embed 標籤的 wmode 參數設定為 opaque 或 transparent。因為 wmode 預設值為 window，依照預設，SWF 內容不會顯示在透明視窗中。無論使用哪一個 wmode 值，PDF 內容都不會顯示在透明視窗中。

同時，如果縮放、旋轉 HTMLLoader 控制項，或 HTMLLoader alpha 屬性設為 1.0 以外的值，SWF 和 PDF 內容都不會顯示。

加入 **SWF** 內容做為 **HTML** 視窗上的重疊部分

由於 HTML 視窗包含於 NativeWindow 實體之中，因此您可以將 Flash 顯示物件加入至顯示清單中的 HTML 圖層上方和下方。

若要將顯示物件加入至 HTML 圖層上方，請使用 window.nativeWindow.stage 屬性的 addChild() 方法。addChild() 方法會將內容加入視窗中任何現有內容的上方圖層中。

若要將顯示物件加入至 HTML 圖層下方，請使用 window.nativeWindow.stage 屬性的 addChildAt() 方法，傳入零值以供 index 參數使用。將物件放在零索引上，會將現有內容 (包括 HTML 顯示) 移至上一層，並將新的內容安插在底部。若要顯示位於 HTML 頁面下方圖層的內容，必須將 HTMLlLoader 物件的 paintsDefaultBackground 屬性設定為 false。此外，設定背景顏色的任何頁面元素都不會是透明的。例如，如果您為頁面主體元素設定背景顏色，則頁面上任何部分都不會是透明的。

下列範例說明如何將 Flash 顯示物件加入做為 HTML 頁面的上層重疊或下層重疊。此範例會建立兩個形狀簡單的物件，並在 HTML 內容的上方和下方各加入一個物件。此範例也會根據 enterFrame 事件，更新形狀位置。

```
 <html>
<head>
<title>Bouncers</title>
<script src="AIRAliases.js" type="text/javascript"></script>
<script language="JavaScript" type="text/javascript">
air.Shape = window.runtime.flash.display.Shape;

function Bouncer(radius, color){
    this.radius = radius;
    this.color = color;

    //velocity
    this.vX = -1.3;
    this.vY = -1;

    //Create a Shape object and draw a circle with its graphics property
    this.shape = new air.Shape();
    this.shape.graphics.lineStyle(1,0);
    this.shape.graphics.beginFill(this.color,.9);
    this.shape.graphics.drawCircle(0,0,this.radius);
    this.shape.graphics.endFill();

    //Set the starting position
    this.shape.x = 100;
    this.shape.y = 100;


    //Moves the sprite by adding (vX,vY) to the current position
    this.update = function(){
        this.shape.x += this.vX;
        this.shape.y += this.vY;

        //Keep the sprite within the window
        if( this.shape.x - this.radius < 0){
            this.vX = -this.vX;
        }
        if( this.shape.y - this.radius < 0){
            this.vY = -this.vY;
        }
        if( this.shape.x  + this.radius > window.nativeWindow.stage.stageWidth){
            this.vX = -this.vX;
        }
        if( this.shape.y  + this.radius > window.nativeWindow.stage.stageHeight){
            this.vY = -this.vY;
        }

    };
}

function init(){
    //turn off the default HTML background
    window.htmlLoader.paintsDefaultBackground = false;
    var bottom = new Bouncer(60,0xff2233);
    var top = new Bouncer(30,0x2441ff);
```

```
    //listen for the enterFrame event
    window.htmlLoader.addEventListener("enterFrame",function(evt){
        bottom.update();
        top.update();
    });

    //add the bouncing shapes to the window stage
    window.nativeWindow.stage.addChildAt(bottom.shape,0);
    window.nativeWindow.stage.addChild(top.shape);
}
</script>
<body onload="init();">
<h1>de Finibus Bonorum et Malorum</h1>
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis
et quasi architecto beatae vitae dicta sunt explicabo.</p>
<p style="background-color:#FFFF00; color:#660000;">This paragraph has a background color.</p>
<p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis
praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias
excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui
officia deserunt mollitia animi, id est laborum et dolorum fuga.</p>
</body>
</html>
```

此範例提供一些進階技巧的基本簡介，這些技巧可在 AIR 中廣泛地應用於 JavaScript 與 ActionScript 上。如果您不熟悉使用 ActionScript 顯示物件，請參閱 《ActionScript 3.0 開發人員指南》中的第 128 頁 「顯示程式設計」。

範例：建立原生視窗

Adobe AIR 1.0 以及更新的版本

下列範例說明如何建立原生視窗：

```
public function createNativeWindow():void {
    //create the init options
    var options:NativeWindowInitOptions = new NativeWindowInitOptions();
    options.transparent = false;
    options.systemChrome = NativeWindowSystemChrome.STANDARD;
    options.type = NativeWindowType.NORMAL;

    //create the window
    var newWindow:NativeWindow = new NativeWindow(options);
    newWindow.title = "A title";
    newWindow.width = 600;
    newWindow.height = 400;

    newWindow.stage.align = StageAlign.TOP_LEFT;
    newWindow.stage.scaleMode = StageScaleMode.NO_SCALE;

    //activate and show the new window
    newWindow.activate();
}
```

管理視窗

Adobe AIR 1.0 以及更新的版本

您可以使用 NativeWindow 類別的屬性和方法來管理桌面視窗的外觀、行為方式和效期。

備註：使用 Flex 架構時，通常最好使用架構類別來管理視窗行為。大部分的 NativeWindow 屬性和方法都可以透過 mx:WindowedApplication 和 mx:Window 類別存取。

取得 NativeWindow 實體

Adobe AIR 1.0 以及更新的版本

操控視窗之前，必須先取得視窗實體。您可以從下列其中一個位置取得視窗實體：

- 用來建立視窗的原生視窗建構函式：

```
var win:NativeWindow = new NativeWindow(initOptions);
```

- 視窗舞台的 nativeWindow 屬性：

```
var win:NativeWindow = stage.nativeWindow;
```

- 視窗中顯示物件的 stage 屬性：

```
var win:NativeWindow = displayObject.stage.nativeWindow;
```

- 視窗所傳送之原生視窗事件的 target 屬性：

```
private function onNativeWindowEvent(event:NativeWindowBoundsEvent):void
{
    var win:NativeWindow = event.target as NativeWindow;
}
```

- 視窗中顯示之 HTML 頁面的 nativeWindow 屬性：

```
var win:NativeWindow = htmlLoader.window.nativeWindow;
```

- NativeApplication 物件的 activeWindow 和 openedWindows 屬性：

```
var nativeWin:NativeWindow = NativeApplication.nativeApplication.activeWindow;
var firstWindow:NativeWindow = NativeApplication.nativeApplication.openedWindows[0];
```

  NativeApplication.nativeApplication.activeWindow 會參考應用程式的作用中視窗 (但如果作用中視窗不是此 AIR 應用程式的視窗，則會傳回 null)。NativeApplication.nativeApplication.openedWindows 陣列會包含尚未關閉之 AIR 應用程式中的所有視窗。

由於 Flex 的 mx:WindowedApplication 和 mx:Window 物件都是顯示物件，您可以使用 stage 屬性，輕鬆地參照 XML 檔案中的應用程式視窗，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" applicationComplete="init();">
    <mx:Script>
        <![CDATA[
            import flash.display.NativeWindow;

            public function init():void{
                var appWindow:NativeWindow = this.stage.nativeWindow;
                //set window properties
                appWindow.visible = true;
            }
        ]]>
    </mx:Script>
</WindowedApplication
```

備註：直到 Flex 架構將 WindowedApplication 或 Window 組件加入視窗舞台之前，組件的 stage 屬性都會是 null。這種行為方式與 Flex Application 組件一致，但並不表示您無法在 WindowedApplication 和 Window 組件的初始化週期中稍早發生之事件 (例如 creationComplete) 的偵聽程式中，存取 stage 或 NativeWindow 實體。傳送 applicationComplete 事件時，可以安全地存取 stage 和 NativeWindow 實體。

啟用、顯示及隱藏視窗

Adobe AIR 1.0 以及更新的版本

若要啟用視窗，請呼叫 NativeWindow activate() 方法。啟用視窗會將視窗移到前面，給予它鍵盤和滑鼠焦點，而且必要時，也會還原視窗或將 visible 屬性設定為 true，以顯示視窗。啟用視窗不會變更應用程式中其它視窗的順序。呼叫 activate() 方法會導致視窗傳送 activate 事件。

若要顯示隱藏的視窗而不加以啟用，請將 visible 屬性設定為 true。這會將視窗移到前面，但不會指定焦點給該視窗。

若要隱藏視窗不讓人看見，請將其 visible 屬性設定為 false。隱藏視窗會同時抑制視窗的顯示以及任何相關的工作列圖示，而在 Mac OS X 上，則會抑制 Windows 選單項目的顯示。

當您變更視窗的可見性時，該視窗擁有的任何視窗可見性也會隨之改變。例如，如果您隱藏視窗，也會同時隱藏其擁有的所有視窗。

備註：在 Mac OS X 中，無法完全隱藏在視窗停駐位置具有圖示的最小化視窗。如果最小化視窗上的 visible 屬性設定為 false，那麼視窗的停駐圖示仍然會顯示。如果使用者按一下圖示，視窗會還原為可見的狀態並顯示。

變更視窗顯示順序

Adobe AIR 1.0 以及更新的版本

AIR 提供多個方法，可用來直接變更視窗的顯示順序。您可以在顯示順序中將視窗向前或向後移動；也可以將視窗移到另一個視窗的上方或後方。同時，使用者也可以透過啟用視窗，重新排列視窗的順序。

您可以將視窗的 alwaysInFront 屬性設定為 true，讓視窗保持在其它視窗前面。如果有多個視窗具備此設定，則這些視窗的顯示順序為彼此間的排序順序，但是它們永遠都會排列在將 alwaysInFront 設定為 false 的視窗上方。

在最上端群組中的視窗也會顯示在其它應用程式中視窗的上方，即使 AIR 應用程式不在作用中也是如此。因為這項行為方式可能會對使用者造成困擾，請只在需要及適當時，將 alwaysInFront 設定為 true。適合使用的範例包括：

- 控制項的暫時性彈出式視窗，例如工具提示、彈出式清單、自訂選單，或下拉式清單方塊。由於這些視窗會在失去焦點時關閉，因此可以避免妨礙使用者檢視另一個視窗的擾人情況。
- 極端緊急的錯誤訊息和警告。當使用者未適時回應而可能會發生無法挽回的變更時，就應該將警告視窗推到最前面。但是，多半錯誤和警告都可以依一般視窗顯示順序處理。

- 短效期的 Toast 樣式視窗。

備註：AIR 不會強制在適當的狀況下使用 alwaysInFront 屬性。但是，如果您的應用程式中斷了使用者的工作流程，很可能會寄存在該使用者的垃圾筒裡。

如果某個視窗擁有其他視窗，則這些視窗永遠都會排序在它前面。如果您在擁有其他視窗的視窗上，呼叫 orderToFront() 或將 alwaysInFront 設為 true，則會將擁有的視窗連同擁有者視窗一起重新排序在其他視窗的前面，但是擁有的視窗仍然會顯示在擁有者的前面。

在擁有的視窗上呼叫視窗排序方法，通常可在相同視窗所擁有的視窗之間正常運作，但是相較於該群組外面的視窗，也可能會變更擁有的視窗整個群組的順序。例如，如果您在擁有的視窗上呼叫 orderToFront()，則會將該視窗、其擁有者及相同擁有者擁有的任何其他視窗都移到視窗顯示順序的前面。

NativeWindow 類別會提供下列屬性和方法，以供設定視窗相對於其它視窗的顯示順序：

成員	說明
alwaysInFront 屬性	指定視窗是否顯示於最上層的視窗群組。 幾乎在所有情況下，false 會是最好的設定。將值從 false 變更為 true，會將視窗移到所有視窗的前面 (但不會加以啟用)。將值從 true 變更為 false，會將該視窗排列在最上層群組中其餘視窗的後面，但仍然在其它視窗的前面。將視窗的屬性設定為其目前的值並不會變更該視窗的顯示順序。 alwaysInFront 設定對於其他視窗擁有的視窗沒有影響。
orderToFront()	將視窗顯示在最前面。
orderInFrontOf()	將此視窗直接移到特定視窗的前面。
orderToBack()	將視窗傳送到其它視窗的後面。
orderBehind()	將視窗直接傳送到特定視窗的後面。
activate()	將視窗移到前面 (同時也讓視窗顯示為可見，並指定焦點)。

備註：如果視窗是隱藏的 (visible 為 false) 或最小化，則呼叫顯示順序的方法沒有任何作用。

在 Linux 作業系統中，不同的視窗管理員會針對視窗的顯示順序強制執行不同的規則：

- 在部分視窗管理員中，公用程式視窗永遠會顯示在一般視窗的前面。
- 在部分視窗管理員中，將 alwaysInFront 設為 true 的全螢幕視窗永遠會顯示在其它同樣將 alwaysInFront 設為 true 的視窗前面。

關閉視窗

Adobe AIR 1.0 以及更新的版本

若要關閉視窗，請使用 NativeWindow.close() 方法。

關閉視窗會取消載入視窗內容，但如果其它物件參考此內容，就不會刪除此內容物件。NativeWindow.close() 方法會以非同步的方式執行，在關閉處理中，視窗所包含的應用程式會繼續執行。close 方法會在關閉作業完成時傳送 close 事件。就技術層面來看，NativeWindow 物件仍然有效，但在已關閉視窗上存取大部分屬性和方法，會產生 IllegalOperationError。您不能重新開啟已關閉的視窗。請檢查視窗的 closed 屬性，以測試視窗是否已關閉。若只是要隱藏視窗不讓人看見，請將 NativeWindow.visible 屬性設定為 false。

如果 Nativeapplication.autoExit 屬性是 true，這是預設值，則在最後一個視窗關閉時，應用程式就會結束。

任何具有擁有者的視窗都會在擁有者關閉時同時關閉。擁有的視窗不會傳送 closing 事件，因此無法防止關閉。會傳送 close 事件。

允許取消視窗作業

Adobe AIR 1.0 以及更新的版本

當視窗使用系統顏色時，使用者與視窗之間的互動可以透過偵聽並取消應用程式事件的預設行為方式加以取消。例如，當使用者按一下系統顏色關閉按鈕時，就會傳送 closing 事件。如果任何已登錄偵聽程式呼叫該事件的 preventDefault() 方法，則視窗就不會關閉。

當視窗不使用系統顏色時，就不會在變更進行之前，自動傳送預期變更的通知事件。因此，如果您呼叫關閉視窗的方法、變更視窗狀態，或設定任何視窗邊界屬性，都無法取消變更。若要在視窗進行變更之前通知應用程式中的組件，您的應用程式邏輯可以使用視窗的 dispatchEvent() 方法，傳送相關的通知事件。

例如，下列邏輯會實作視窗關閉按鈕的可取消事件處理常式：

```
public function onCloseCommand(event:MouseEvent):void{
    var closingEvent:Event = new Event(Event.CLOSING,true,true);
    dispatchEvent(closing);
    if(!closingEvent.isDefaultPrevented()){
        win.close();
    }
}
```

如果偵聽程式呼叫事件 preventDefault() 方法，則 dispatchEvent() 方法會傳回 false。但是，它也可能會在其它原因之下傳回 false，所以最好是明確地使用 isDefaultPrevented() 方法來測試是否應該取消變更。

最大化、最小化和還原視窗

Adobe AIR 1.0 以及更新的版本

若要將視窗最大化，請使用 NativeWindow maximize() 方法。

```
myWindow.maximize();
```

若要將視窗最小化，請使用 NativeWindow minimize() 方法。

```
myWindow.minimize();
```

若要還原視窗 (也就是，回復到最小化或最大化之前的大小)，請使用 NativeWindow restore() 方法。

```
myWindow.restore();
```

擁有視窗最小化或還原時，也會最小化和還原具有擁有者的視窗。擁有的視窗因為其擁有者最小化而最小化時，擁有的視窗不會傳送任何事件。

備註：最小化 AIR 視窗產生的行為與 Mac OS X 的標準行為不同。AIR 視窗不會在應用程式定義的 「標準」大小與使用者所設定最後大小之間切換，而會在應用程式或使用者最後設定的大小與螢幕的完整可使用區域之間切換。

在 Linux 作業系統中，不同的視窗管理員會針對視窗顯示狀態設定強制執行不同的規則：

- 在部分視窗管理員中，公用程式視窗無法最大化。
- 若視窗已設定最大大小時，部分視窗會不允許最大化視窗。也有部分視窗管理員會將顯示狀態設定為最大化，但不會調整視窗大小。這兩種情況都不會傳送顯示狀態變更事件。
- 部分視窗管理員不會理會視窗的 maximizable 或 minimizable 設定。

備註：在 Linux 中，視窗屬性會以非同步的方式變更。如果您在程式的某一行變更了顯示狀態，然後在下一行中讀取該值，所讀取的值仍然會反映舊的設定。不論在哪個平台上，NativeWindow 物件在顯示狀態變更時都會傳送 displayStateChange 事件。如果您需要根據視窗的新狀態來採取某些動作，請務必在 displayStateChange 事件處理常式中執行。請參閱第 788 頁 「偵聽視窗事件」。

範例：最大化、最小化、還原與關閉視窗

Adobe AIR 1.0 以及更新的版本

下列簡短的 MXML 應用程式會示範 Window maximize()、minimize()、restore() 和 close() 方法：

```
<?xml version="1.0" encoding="utf-8"?>

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">


    <mx:Script>
    <![CDATA[
    public function minimizeWindow():void
    {
        this.stage.nativeWindow.minimize();
    }

    public function maximizeWindow():void
    {
        this.stage.nativeWindow.maximize();
    }

    public function restoreWindow():void
    {
        this.stage.nativeWindow.restore();
    }

    public function closeWindow():void
    {
        this.stage.nativeWindow.close();
    }
    ]]>
    </mx:Script>

    <mx:VBox>
        <mx:Button label="Minimize" click="minimizeWindow()"/>
        <mx:Button label="Restore" click="restoreWindow()"/>
        <mx:Button label="Maximize" click="maximizeWindow()"/>
        <mx:Button label="Close" click="closeWindow()"/>
    </mx:VBox>

</mx:WindowedApplication>
```

下列 Flash 的 ActionScript 範例會建立四個可點選的文字欄位，這些欄位會觸發 NativeWindow minimize()、maximize()、restore() 和 close() 方法：

```
package
{
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.text.TextField;

    public class MinimizeExample extends Sprite
    {
        public function MinimizeExample():void
        {
            var minTextBtn:TextField = new TextField();
            minTextBtn.x = 10;
            minTextBtn.y = 10;
            minTextBtn.text = "Minimize";
            minTextBtn.background = true;
            minTextBtn.border = true;
            minTextBtn.selectable = false;
            addChild(minTextBtn);
            minTextBtn.addEventListener(MouseEvent.CLICK, onMinimize);

            var maxTextBtn:TextField = new TextField();
            maxTextBtn.x = 120;
            maxTextBtn.y = 10;
            maxTextBtn.text = "Maximize";
            maxTextBtn.background = true;
            maxTextBtn.border = true;
            maxTextBtn.selectable = false;
            addChild(maxTextBtn);
            maxTextBtn.addEventListener(MouseEvent.CLICK, onMaximize);

            var restoreTextBtn:TextField = new TextField();
            restoreTextBtn.x = 230;
            restoreTextBtn.y = 10;
            restoreTextBtn.text = "Restore";
            restoreTextBtn.background = true;
            restoreTextBtn.border = true;
            restoreTextBtn.selectable = false;
            addChild(restoreTextBtn);
            restoreTextBtn.addEventListener(MouseEvent.CLICK, onRestore);

            var closeTextBtn:TextField = new TextField();
            closeTextBtn.x = 340;
            closeTextBtn.y = 10;
            closeTextBtn.text = "Close Window";
            closeTextBtn.background = true;
            closeTextBtn.border = true;
            closeTextBtn.selectable = false;
            addChild(closeTextBtn);
```

```
            closeTextBtn.addEventListener(MouseEvent.CLICK, onCloseWindow);
        }
        function onMinimize(event:MouseEvent):void
        {
            this.stage.nativeWindow.minimize();
        }
        function onMaximize(event:MouseEvent):void
        {
            this.stage.nativeWindow.maximize();
        }
        function onRestore(event:MouseEvent):void
        {
            this.stage.nativeWindow.restore();
        }
        function onCloseWindow(event:MouseEvent):void
        {
            this.stage.nativeWindow.close();
        }
    }
}
```

調整視窗大小及移動視窗

Adobe AIR 1.0 以及更新的版本

當視窗使用系統顏色時，該設定會提供拖曳控制項，以供調整視窗大小，並在桌面上四處移動視窗。如果視窗不使用系統顏色，則必須自行加入控制項，以便讓使用者調整視窗大小及移動視窗。

備註：若要調整視窗大小或移動視窗，必須先取得 NativeWindow 實體的參考。如需有關如何取得視窗參考的詳細資訊，請參閱第 780 頁「取得 NativeWindow 實體」。

調整視窗大小

若要讓使用者以互動方式調整視窗大小，請使用 NativeWindow 的 startResize() 方法。從 mouseDown 事件呼叫此方法時，調整大小作業會由滑鼠驅動，並在作業系統接到 mouseUp 事件時完成。呼叫 startResize() 時，會傳入引數，指定要開始調整視窗大小的邊緣或角落。

若要以程式設計的方式設定視窗大小，請將視窗的 width、height 或 bounds 屬性設定為所需的尺寸。當您設定邊界時，視窗的大小和位置都可以同時變更。不過，無法保證會依照何種順序變更。部分 Linux 視窗管理員不允許視窗延伸到桌面螢幕的邊界之外。在這種情況下，最終的視窗大小可能會因為屬性的設定順序而遭到限制，即便這些變更的最終效果可能因為順序不同而讓視窗合法。例如，如果您為接近畫面底部的視窗同時變更高度和 y 座標，則如果高度是在 y 座標進行變更之前套用，對於高度所做的完整變更可能就不會生效。

備註：在 Linux 中，視窗屬性會以非同步的方式變更。如果您在程式的某一行調整視窗大小，然後在下一行中讀取其尺寸，則尺寸仍然會反映舊的設定。不論在哪個平台上，NativeWindow 物件在視窗調整大小時都會傳送 resize 事件。如果您需要根據視窗新的大小或狀態來採取某些動作 (例如在視窗中配置控制項)，請務必在 resize 事件處理常式中執行。請參閱第 788 頁「偵聽視窗事件」。

舞台的縮放模式會決定在調整視窗大小時，視窗舞台及其內容的行為。請記住，舞台縮放模式是針對情況 (如網頁瀏覽器) 而設計，而不是由應用程式控制其顯示空間的大小和比例。一般來說，您可以將舞台 scaleMode 屬性設定為 StageScaleMode.NO_SCALE，取得最佳效果。如果您要縮放視窗的內容，仍然可以設定內容的 scaleX 和 scaleY 參數，以因應視窗邊界的變更。

移動視窗

若要移動視窗而不調整其大小，請使用 NativeWindow startMove() 方法。就像 startResize() 方法一樣，從 mouseDown 事件呼叫 startMove() 方法時，移動程序是由滑鼠驅動的，而當作業系統接到 mouseUp 事件時，就已完成程序。

如需有關 startResize() 和 startMove() 方法的詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。

若要以程式設計的方式移動視窗，請將視窗的 x、y 或 bounds 屬性設為所需的位置。設定邊界時，視窗的大小和位置可以同時變更。

備註：在 Linux 中，視窗屬性會以非同步的方式變更。如果您在程式的某一行移動了視窗，然後在下一行中讀取其位置，所讀取的值仍然會反映舊的設定。不論在哪個平台上，NativeWindow 物件在位置變更時都會傳送 move 事件。如果您需要根據視窗的新位置來採取某些動作，請務必在 move 事件處理常式中執行。請參閱第 788 頁「偵聽視窗事件」。

範例：調整視窗大小及移動視窗

Adobe AIR 1.0 以及更新的版本

下列範例會說明如何在視窗上開始進行調整大小及移動作業：

```
package
{
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.display.NativeWindowResize;

    public class NativeWindowResizeExample extends Sprite
    {
        public function NativeWindowResizeExample():void
        {
            // Fills a background area.
            this.graphics.beginFill(0xFFFFFF);
            this.graphics.drawRect(0, 0, 400, 300);
            this.graphics.endFill();

            // Creates a square area where a mouse down will start the resize.
            var resizeHandle:Sprite =
                createSprite(0xCCCCCC, 20, this.width - 20, this.height - 20);
            resizeHandle.addEventListener(MouseEvent.MOUSE_DOWN, onStartResize);

            // Creates a square area where a mouse down will start the move.
            var moveHandle:Sprite = createSprite(0xCCCCCC, 20, this.width - 20, 0);
            moveHandle.addEventListener(MouseEvent.MOUSE_DOWN, onStartMove);
        }

        public function createSprite(color:int, size:int, x:int, y:int):Sprite
        {
            var s:Sprite = new Sprite();
```

```
            s.graphics.beginFill(color);
            s.graphics.drawRect(0, 0, size, size);
            s.graphics.endFill();
            s.x = x;
            s.y = y;
            this.addChild(s);
            return s;
        }

        public function onStartResize(event:MouseEvent):void
        {
            this.stage.nativeWindow.startResize(NativeWindowResize.BOTTOM_RIGHT);
        }

        public function onStartMove(event:MouseEvent):void
        {
            this.stage.nativeWindow.startMove();
        }
    }
}
```

偵聽視窗事件

Adobe AIR 1.0 以及更新的版本

若要偵聽視窗所傳送的事件，請用視窗實體登錄偵聽程式。例如，若要偵聽 closing 事件，請如下所示，以視窗登錄偵聽程式：

```
myWindow.addEventListener(Event.CLOSING, onClosingEvent);
```

傳送事件時，target 屬性會參考視窗傳送該事件。

大部分視窗事件有兩則相關訊息。第一則訊息標示視窗變更即將發生 (且可以取消)，而第二則訊息標示已發生變更。例如，當使用者按一下視窗的 「關閉」按鈕時，就會傳送 closing 事件訊息。如果沒有偵聽程式取消該事件，視窗就會關閉，並傳送 close 事件給任何偵聽程式。

一般來說，警告事件 (如 closing) 只有在使用系統顏色來觸發事件時才會傳送。例如，呼叫 window close() 方法不會自動傳送 closing 事件，而只傳送 close 事件。但是，您可以建構 closing 事件物件，並使用 window dispatchEvent() 方法加以傳送。

傳送 Event 物件的視窗事件如下：

事件	說明
activate	在視窗收到焦點時傳送。
deactivate	在視窗失去焦點時傳送。
closing	在視窗即將關閉時傳送。這只會在按下系統顏色 「關閉」按鈕時自動發生，而在 Mac OS X 中，則是在叫用 Quit 命令時發生。
close	視窗已關閉時傳送。

傳送 NativeWindowBoundsEvent 物件的視窗事件如下：

事件	說明
moving	在視窗左上角變更位置之前，做為移動、調整大小或變更視窗顯示狀態的結果傳送。
move	在左上角已經變更位置之後傳送。
resizing	在視窗寬度或高度變更之前，做為調整大小或顯示狀態變更的結果傳送。
resize	在視窗已經變更大小之後傳送。

對於 NativeWindowBoundsEvent 事件，您可以使用 beforeBounds 和 afterBounds 屬性，判斷即將完成或已完成變更前後的視窗邊界。

傳送 NativeWindowDisplayStateEvent 物件的視窗事件如下：

事件	說明
displayStateChanging	在視窗顯示狀態變更之前傳送。
displayStateChange	在視窗顯示狀態已變更之後傳送。

對於 NativeWindowDisplayStateEvent 事件，您可以使用 beforeDisplayState 和 afterDisplayState 屬性，判斷即將變更或已完成變更前後的視窗顯示狀態。

在部分 Linux 視窗管理員中，最大化已設定最大大小的視窗時，並不會傳送顯示狀態變更事件 (視窗已設定為最大的顯示狀態，但未調整大小)。

顯示全螢幕視窗

Adobe AIR 1.0 以及更新的版本

將 Stage 的 displayState 屬性設定為 StageDisplayState.FULL_SCREEN_INTERACTIVE，會讓視窗進入全螢幕模式，在此模式下會 「允許」鍵盤輸入。(在瀏覽器執行中的 SWF 內容則不允許鍵盤輸入)。若要跳出全螢幕模式，使用者可以按下 Esc 鍵。

備註：部分 Linux 視窗管理員並不會在視窗具有最大大小設定時變更視窗尺寸以符合螢幕大小 (但是會移除視窗系統的顏色)。

例如，下列 Flex 程式碼會定義簡單的 AIR 應用程式，以設定簡單的全螢幕終端機：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    applicationComplete="init()" backgroundColor="0x003030" focusRect="false">
    <mx:Script>
        <![CDATA[
            private function init():void
            {
                stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
                focusManager.setFocus(terminal);
                terminal.text = "Welcome to the dumb terminal app. Press the ESC key to exit..\n";
                terminal.selectionBeginIndex = terminal.text.length;
                terminal.selectionEndIndex = terminal.text.length;
            }
        ]]>
    </mx:Script>
    <mx:TextArea
        id="terminal"
        height="100%" width="100%"
        scroll="false"
        backgroundColor="0x003030"
        color="0xCCFF00"
        fontFamily="Lucida Console"
        fontSize="44"/>
</mx:WindowedApplication>
```

下列 Flash 的 ActionScript 範例則會模擬簡單的全螢幕文字終端機：

```
import flash.display.Sprite;
import flash.display.StageDisplayState;
import flash.text.TextField;
import flash.text.TextFormat;

public class FullScreenTerminalExample extends Sprite
{
    public function FullScreenTerminalExample():void
    {
        var terminal:TextField = new TextField();
        terminal.multiline = true;
        terminal.wordWrap = true;
        terminal.selectable = true;
        terminal.background = true;
        terminal.backgroundColor = 0x00333333;

        this.stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;

        addChild(terminal);
        terminal.width = 550;
        terminal.height = 400;

        terminal.text = "Welcome to the dumb terminal application. Press the ESC key to exit.\n_";

        var tf:TextFormat = new TextFormat();
        tf.font = "Courier New";
        tf.color = 0x00CCFF00;
        tf.size = 12;
        terminal.setTextFormat(tf);

        terminal.setSelection(terminal.text.length - 1, terminal.text.length);
    }
}
```

第 53 章 AIR 的顯示螢幕

Adobe AIR 1.0 以及更新的版本

您可以使用 Adobe® AIR® Screen 類別來存取連接至電腦或裝置的顯示螢幕相關資訊。

更多說明主題
flash.display.Screen

AIR 顯示螢幕基本概念

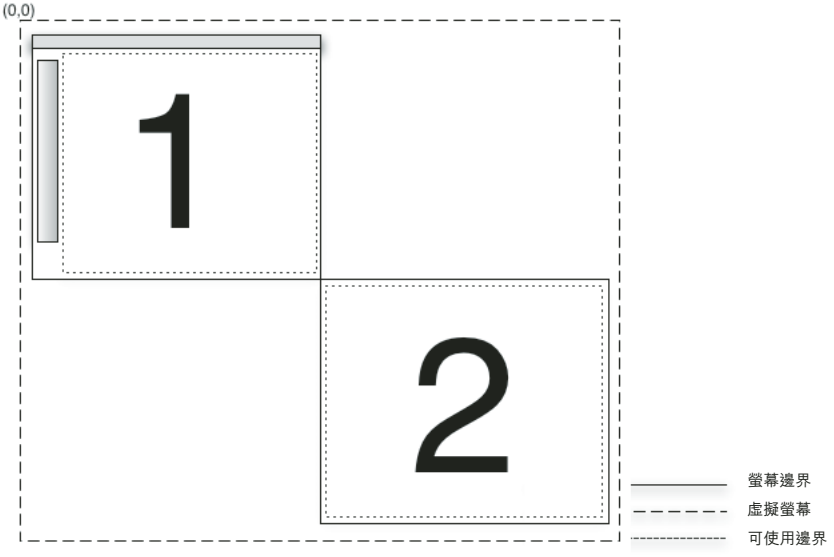
Adobe AIR 1.0 以及更新的版本

- 測量虛擬桌面 (Flex)
- 測量虛擬桌面 (Flash)

螢幕 API 只包含一個 Screen 類別，可提供靜態成員來取得系統螢幕資訊，並供實體成員來描述特定螢幕。

一個電腦系統可以連接多個監視器或顯示螢幕，對應於多個排列於虛擬空間中的桌面螢幕。AIR Screen 類別提供有關螢幕、螢幕相對排列方式，以及螢幕可用空間等資訊。如果有多個監視器對應至相同的螢幕，則只會有一個螢幕。如果螢幕的大小大於監視器的顯示區，則沒有辦法判斷目前可見區域所屬的螢幕部分為何。

螢幕代表獨立的桌面顯示區。螢幕通常是虛擬桌面之中的矩形區。主要顯示畫面的螢幕左上角指定為虛擬桌面座標系統的起點。用來描述螢幕的所有值都會以像素為單位。



在這種螢幕排列法之下，兩個螢幕會同時出現在虛擬桌面上。主螢幕 (#1) 左上角的座標永遠都是 (0,0)。如果螢幕排列法變更為將螢幕 #2 指定為主螢幕，則螢幕 #1 的座標就會變成負值。回報螢幕的可用邊界時，會排除選單列、工作列和停駐列。

如需螢幕 API 類別、方法、屬性和事件的詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。

列舉螢幕

Adobe AIR 1.0 以及更新的版本

您可以使用下列 Screen 方法和屬性，列舉虛擬桌面的螢幕：

方法或屬性	說明
Screen.screens	提供用來描述可用螢幕的 Screen 物件。陣列的順序並不重要。
Screen.mainScreen	為主螢幕提供 Screen 物件。在 Mac OS X 上，主螢幕是顯示選單列的螢幕。在 Windows 上，主螢幕是系統指定的主要螢幕。
Screen.getScreensForRectangle()	提供用來描述與指定矩形相交之螢幕的 Screen 物件陣列。傳遞給這個方法的矩形為虛擬桌面上的像素座標。如果沒有螢幕與矩形相交，陣列就是空的。您可以使用這個方法，找出顯示視窗的螢幕。

請勿儲存由 Screen 類別方法和屬性所傳回的值。因為，使用者或作業系統隨時都可以變更可用的螢幕及其排列方式。

下列範例會使用螢幕 API，在多個螢幕之間移動視窗，以回應按下方向鍵的動作。為了將視窗移至下一個螢幕，此範例會取得 screens 陣列，並依水平或垂直方向排序 (取決於所按的方向鍵)。然後，程式碼會逐步執行已排序的陣列，將每個螢幕的座標與目前螢幕的座標進行比較。為了識別視窗的目前螢幕，此範例會呼叫 Screen.getScreensForRectangle()，傳入視窗邊界。

```
package {
    import flash.display.Sprite;
    import flash.display.Screen;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;

    public class ScreenExample extends Sprite
    {
        public function ScreenExample()
        {
                stage.align = StageAlign.TOP_LEFT;
                stage.scaleMode = StageScaleMode.NO_SCALE;

                stage.addEventListener(KeyboardEvent.KEY_DOWN,onKey);
        }

        private function onKey(event:KeyboardEvent):void{
            if(Screen.screens.length > 1){
                switch(event.keyCode){
                    case Keyboard.LEFT :
                        moveLeft();
                        break;
                    case Keyboard.RIGHT :
                        moveRight();
                        break;
                    case Keyboard.UP :
                        moveUp();
                        break;
                    case Keyboard.DOWN :
                        moveDown();
                        break;
                }
            }
        }

        private function moveLeft():void{
            var currentScreen = getCurrentScreen();
```

```
        var left:Array = Screen.screens;
        left.sort(sortHorizontal);
        for(var i:int = 0; i < left.length - 1; i++){
            if(left[i].bounds.left < stage.nativeWindow.bounds.left){
                stage.nativeWindow.x +=
                    left[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;
            }
        }
    }

    private function moveRight():void{
        var currentScreen:Screen = getCurrentScreen();
        var left:Array = Screen.screens;
        left.sort(sortHorizontal);
        for(var i:int = left.length - 1; i > 0; i--){
            if(left[i].bounds.left > stage.nativeWindow.bounds.left){
                stage.nativeWindow.x +=
                    left[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;
            }
        }
    }

    private function moveUp():void{
        var currentScreen:Screen = getCurrentScreen();
        var top:Array = Screen.screens;
        top.sort(sortVertical);
        for(var i:int = 0; i < top.length - 1; i++){
            if(top[i].bounds.top < stage.nativeWindow.bounds.top){
                stage.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;
                break;
            }
        }
    }

    private function moveDown():void{
        var currentScreen:Screen = getCurrentScreen();

        var top:Array = Screen.screens;
        top.sort(sortVertical);
        for(var i:int = top.length - 1; i > 0; i--){
            if(top[i].bounds.top > stage.nativeWindow.bounds.top){
                stage.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;
                break;
            }
        }
    }

    private function sortHorizontal(a:Screen,b:Screen):int{
        if (a.bounds.left > b.bounds.left){
            return 1;
        } else if (a.bounds.left < b.bounds.left){
```

```
                return -1;
            } else {return 0;}
        }

        private function sortVertical(a:Screen,b:Screen):int{
            if (a.bounds.top > b.bounds.top){
                return 1;
            } else if (a.bounds.top < b.bounds.top){
                return -1;
            } else {return 0;}
        }

        private function getCurrentScreen():Screen{
            var current:Screen;
            var screens:Array = Screen.getScreensForRectangle(stage.nativeWindow.bounds);
            (screens.length > 0) ? current = screens[0] : current = Screen.mainScreen;
            return current;
        }
    }
}
```

第 54 章 列印

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash 執行階段 (如 Adobe® Flash® Player 和 Adobe® AIR™) 可以與作業系統的列印介面通訊，以便讓您將頁面傳遞至列印多工緩衝處理區域。傳送到多工緩衝處理區域的每一頁，都可以包含使用者可見的、動態的或螢幕外的內容，其中還包括資料庫值和動態文字。此外，flash.printing.PrintJob 類別的屬性包含以使用者印表機設定為基礎的值，讓您可以適當地設定頁面格式。

以下內容將說明如何善用 flash.printing.PrintJob 類別的方法與屬性，建立列印工作、讀取使用者列印設定，並根據 Flash 執行階段與使用者作業系統，對列印工作進行調整。

列印的基本概念

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 ActionScript 3.0 中，您可以使用 PrintJob 類別來建立顯示物件的快照，以便在列印輸出中將內容轉換為墨水與紙張的綜合呈現。在某些方面來說，設定要列印的內容與設定在螢幕上顯示內容是一樣的意思，您只需設定好元素的位置與大小，就能建立想要的版面。然而，列印擁有一些與單純在螢幕上顯示版面更為獨特的地方。例如，印表機所使用的解析度與電腦螢幕的解析度不一樣；電腦螢幕的內容是動態且可變更，而列印出來的內容本質上則是靜態的；而在規劃列印時，還需要考量到固定頁面大小的限制與多頁列印的可能性。

儘管這些差異很容易察覺，設定 ActionScript 的列印作業時仍舊得處處留意，小心進行。精準的列印取決於您指定的值以及印表機特性，二者的組合。PrintJob 類別包括各種屬性，可以讓您判斷使用者印表機的重要特性。

重要概念與術語

下列參考清單包含與列印相關的重要術語：

多工緩衝處理區域 作業系統或印表機驅動程式軟體的一部分，可用來追蹤等候列印的頁面，並在印表機能夠再度提供列印服務時，將這些頁面傳送至印表機。

頁面方向 關於紙張的內容列印方向，如果不是水平列印 (橫向)，就是垂直列印 (縱向)。

列印工作 產生單一列印結果的單張或頁面組。

列印頁面

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 PrintJob 類別實體來處理列印作業。若要透過 Flash Player 或 AIR 來列印基本頁面，您必須依序使用下列四個陳述式：

- new PrintJob()：以您指定的名稱建立新的列印工作實體。
- PrintJob.start()：起始作業系統的列印程序、為使用者呼叫列印對話方塊，並填入列印工作的唯讀屬性。
- PrintJob.addPage()：包含列印工作內容的詳細資訊，其中包括 Sprite 物件 (和它所包含的任何子系)、列印區域的大小，以及印表機要將影像列印成向量或點陣圖。然後，您就可以呼叫 addPage()，在數個頁面上列印多個 Sprite。

- PrintJob.send()：將頁面傳送到作業系統的印表機。

因此，例如簡易列印工作指令碼如下 (包括編譯時會用到的 package、import 以及 class 陳述式)：

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;

    public class BasicPrintExample extends Sprite
    {
        var myPrintJob:PrintJob = new PrintJob();
        var mySprite:Sprite = new Sprite();

        public function BasicPrintExample()
        {
            myPrintJob.start();
            myPrintJob.addPage(mySprite);
            myPrintJob.send();
        }
    }
}
```

備註：這個範例會說明列印工作指令碼的基本元素，並且不包含任何處理錯誤的作業。若要建立指令碼，針對使用者取消列印工作做出適當的回應，請參閱第 796 頁「處理例外以及傳回」。

若要清除 PrintJob 物件的屬性，請將 PrintJob 變數設為 null (如 myPrintJob = null)。

Flash 執行階段工作與系統列印

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

由於 Flash 執行階段會將頁面傳送至作業系統的列印介面，因此您應該瞭解 Flash 執行階段所管理的工作，以及作業系統本身列印介面所管理的工作。Flash 執行階段可以起始列印工作、讀取印表機的某些頁面設定、將列印工作內容傳遞給作業系統，以及確認使用者或系統是否已經取消列印工作。至於其它的程序 (例如顯示印表機專屬對話方塊、取消多工緩衝處理區域中的列印工作，或回報印表機的狀態) 則全部由作業系統來處理。當起始或格式化列印工作發生問題時，Flash 執行階段也能送出回應，但只能透過作業系統的列印介面送出特定屬性或條件的問題報告。身為一位開發人員，您撰寫的程式碼必須回應這些屬性或條件。

處理例外以及傳回

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

執行 addPage() 和 send() 呼叫之前，請先檢查 PrintJob.start() 方法是否傳回 true，以免使用者已經取消了列印工作。有個簡單的方式，可以在執行前先檢查是否已經取消這些方法，就是將它們包含在 if 陳述式中，如下列範例所示：

```
if (myPrintJob.start())
{
    // addPage() and send() statements here
}
```

如果 PrintJob.start() 為 true，表示使用者已經選取「列印」(或 Flash 執行階段，例如 Flash Player 或 AIR，已經起始「列印」命令)。所以，可以呼叫 addPage() 和 send() 方法。

同時，為了協助管理列印程序，Flash 執行階段會擲回 PrintJob.addPage() 方法的例外，讓您能夠捕捉錯誤，並將資訊和選項提供給使用者。如果 PrintJob.addPage() 方法失敗，您也可以呼叫其它函數或停止目前的列印工作。您可以透過將 addPage() 呼叫嵌入 try..catch 陳述式中來捕捉這些例外，如下列範例所示。在這個範例中，[params] 是參數的預留位置，用來指定實際要列印的內容：

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error,
    }
    myPrintJob.send();
}
```

列印工作開始執行之後，您就可以使用 PrintJob.addPage() 加入內容，然後查看此內容是否會產生例外 (例如，使用者已經取消了列印工作)。如果產生例外，您就可以在 catch 陳述式中加入邏輯，將資訊或選項提供給使用者 (或 Flash 執行階段)，或是停止目前的列印工作。如果成功加入頁面，接著便可以使用 PrintJob.send() 將這些頁面傳送到印表機。

如果 Flash 執行階段將列印工作傳送到印表機時發生問題 (例如，印表機為離線狀態)，您也可以捕捉這個例外，並將資訊或更多選項 (例如，顯示訊息文字或使用動畫提供警告) 提供給使用者 (或 Flash 執行階段)。例如，您可以在 if..else 陳述式中指定新文字給文字欄位，如下列程式碼所示：

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error.
    }
    myPrintJob.send();
}
else
{
    myAlert.text = "Print job canceled";
}
```

如需實際的範例說明，請參閱第 804 頁 「列印範例：縮放、裁切和回應」。

使用頁面屬性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用者按一下 「列印」對話方塊中的 「確定」，而且 PrintJob.start() 傳回 true 之後，您就可以存取印表機設定所定義的屬性。這些設定包括紙張寬度、紙張高度 (pageHeight 和 pageWidth) 以及紙張上內容的列印方向。由於這些屬性都是印表機的設定，並非由 Flash 執行階段控制，因此您無法變更這些設定；但是您可以利用這些屬性來對齊傳送到印表機的內容，以符合目前的設定。如需詳細資訊，請參閱第 798 頁 「設定大小、縮放與列印方向」。

設定向量或點陣圖顯示

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以手動設定列印工作，將每一個頁面當做向量圖像或點陣圖影像排入多工緩衝處理區域。在某些情況下，向量列印可以產生較小的佇列檔案，影像品質與點陣列印比起來較好。但是，當您要列印的內容中含有點陣圖影像，並且想保留任何的 Alpha 透明度或顏色效果時，請將頁面列印為點陣圖影像。而且，非 PostScript 印表機也會自動將任何向量圖像轉換為點陣圖影像。

將 PrintJobOptions 物件傳遞為 PrintJob.addPage() 的第三個參數，就可以指定點陣列印。

如果是 Flash Player 和 AIR 2 之前的 AIR，請將 PrintJobOptions 物件的 printAsBitmap 參數設為 true，如下所示：

```
var options:PrintJobOptions = new PrintJobOptions();
options.printAsBitmap = true;
myPrintJob.addPage(mySprite, null, options);
```

如果您沒有指定第三個參數的值，則列印工作會採用預設的參數值，也就是使用向量列印。

如果是 AIR 2 以及更新的版本，請使用 PrintJobOptions 物件的 printMethod 屬性來指定列印方法。這個屬性接受三種值，而這些值定義為 PrintMethod 類別中的常數：

- PrintMethod.AUTO：根據列印內容，自動選擇最佳列印方法。例如，如果頁面是由文字組成，則會選擇向量列印方法。不過，如果含有 Alpha 透明度的浮水印影像與文字重疊，則會選擇點陣列印，以保留透明度。
- PrintMethod.BITMAP：強制點陣列印 (不論內容為何)
- PrintMethod.VECTOR：強制向量列印 (不論內容為何)

排定列印工作陳述式時限

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

ActionScript 3.0 沒有將 PrintJob 物件限制在單一影格 (如舊版 ActionScript 作法)。不過，由於作業系統會在使用者按一下 「列印」對話方塊中的 「確定」按鈕之後，對使用者顯示列印狀態資訊，所以請儘快呼叫 PrintJob.addPage() 和 PrintJob.send()，以便將頁面傳送到多工緩衝處理區域。延遲抵達含有 PrintJob.send() 呼叫的影格，也會延遲列印程序。

ActionScript 3.0 將指令碼逾時限制規定為 15 秒鐘。因此，在列印工作序列中的每一個主要陳述式之間的時間間隔也不能超過 15 秒。換句話說，15 秒的指令碼逾時限制也將套用至下列間隔：

- PrintJob.start() 與第一個 PrintJob.addPage() 之間的間隔
- PrintJob.addPage() 與下一個 PrintJob.addPage() 之間的間隔
- 最後一個 PrintJob.addPage() 與 PrintJob.send() 之間的間隔

如果以上任何這些間隔超過 15 秒，下一次針對 PrintJob 實體之 PrintJob.start() 的呼叫就會傳回 false，而且下一個 PrintJob 實體的 PrintJob.addPage() 也會導致 Flash Player 或 AIR 擲回執行階段例外。

設定大小、縮放與列印方向

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在第 795 頁「列印頁面」一節中將詳細說明基本列印工作的步驟，步驟中的輸出將直接反映出列印後的對等螢幕大小和指定之 Sprite 的位置。但是，因為印表機會使用不同的列印解析度，而且不同的設定也會影響列印後 Sprite 的外觀。

Flash 執行階段可以讀取作業系統的列印設定，但必須注意的是這些屬性是唯讀性質：雖然您仍然可以回應它們的值，但不能設定它們的值。因此，舉例來說，您可以依據印表機的頁面大小設定來調整內容以符合大小。您也可以決定印表機的邊界設定和頁面方向。若要回應印表機設定，請指定列印區域、調整螢幕解析度與印表機點數度量之間的差異，或轉換內容以符合使用者印表機的大小或列印方向設定。

使用矩形做為列印區域

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

PrintJob.addPage() 方法可讓您指定要列印之 Sprite 的區域。第二個參數 printArea 的形式為 Rectangle 物件。有三種方式可用來提供這個參數的值：

- 建立具有特定屬性的 Rectangle 物件，然後在 addPage() 呼叫中使用這個矩形，如下列範例所示：

```
private var rect1:Rectangle = new Rectangle(0, 0, 400, 200);
myPrintJob.addPage(sheet, rect1);
```

- 如果尚未指定 Rectangle 物件，您可以在該呼叫本身中指定這個物件，如下列範例所示：

```
myPrintJob.addPage(sheet, new Rectangle(0, 0, 100, 100));
```

- 若要提供 addPage() 呼叫中第三個參數的值，但是不想指定矩形時，可以使用 null 做為第二個參數的值，如下所示。

```
myPrintJob.addPage(sheet, null, options);
```

比較點與像素

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

矩形的寬度與高度都是像素值。印表機會使用點做為列印度量單位。點是固定的實體大小 (1/72 英吋)，不過像素在螢幕上的大小則取決於特定螢幕的解析度。因此，像素與點之間的轉換率，必須根據印表機設定及是否已縮放 Sprite 來決定。未縮放並具有 72 像素寬的 Sprite，將會列印為一英吋的寬度，無論螢幕解析度為何，每一點都會等於一個像素。

您可以使用下列等式，將英吋或公分轉換成崔普 (twip) 或點 (1 崔普相當於 1/20 的點)：

- 1 點 = 1/72 英吋 = 20 崔普
- 1 英吋 = 72 點 = 1440 崔普
- 1 公分 = 567 崔普

如果省略 printArea 參數，或是它並未正確傳遞，則會列印 Sprite 的整個區域。

縮放

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如果您想要在列印之前縮放 Sprite 物件，請在呼叫 PrintJob.addPage() 方法之前設定縮放屬性 (請參閱第 151 頁「調整大小及縮放物件」)，然後在列印後將它們設回原來的值。Sprite 物件的縮放與 printArea 屬性無關。換句話說，如果您指定要列印大小為 50 x 50 像素的區域，就會印出面積為 2500 像素的區域。如果縮放 Sprite 物件，依然會列印相同的 2500 像素，不過會是以縮放的大小列印 Sprite 物件。

如需範例，請參閱第 804 頁「列印範例：縮放、裁切和回應」。

橫向或縱向列印

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

由於 Flash Player 和 AIR 可以偵測列印方向設定，因此您可以在 ActionScript 內建立邏輯，調整內容大小及旋轉以符合印表機設定，如下列範例所示：

```
if (myPrintJob.orientation == PrintJobOrientation.LANDSCAPE)
{
    mySprite.rotation = 90;
}
```

備註：如果您要讀取紙張內容列印方向的系統設定，請務必匯入 PrintJobOrientation 類別。PrintJobOrientation 類別可以提供常數值，定義頁面的內容列印方向。您可以使用下列陳述式匯入這個類別：

```
import flash.printing.PrintJobOrientation;
```

回應頁面高度和寬度

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以利用與處理印表機列印方向設定類似的方式，讀取頁面高度和寬度設定，並在 if 陳述式中嵌入一些邏輯來回應這些設定。下列程式碼為您示範這種範例：

```
if (mySprite.height > myPrintJob.pageHeight)
{
    mySprite.scaleY = .75;
}
```

此外，藉由比較頁面與紙張的尺寸也可以設定頁面的邊界，如下列範例所示：

```
margin_height = (myPrintJob.paperHeight - myPrintJob.pageHeight) / 2;
margin_width = (myPrintJob.paperWidth - myPrintJob.pageWidth) / 2;
```

進階列印技術

Adobe AIR 2 以及更新的版本

從 Adobe AIR 2 開始，PrintJob 類別具有其他屬性和方法，而且支援三個額外的類別：PrintUIOptions、PaperSize 和 PrintMethod。這些變更允許其他的印表機工作流程，並提供作者較佳的列印程序控制能力。變更包含：

- 頁面設定對話方塊：可以顯示標準和自訂頁面設定對話方塊。使用者可以在列印前設定頁面範圍、紙張大小、方向和縮放。
- 列印檢視：可建立檢視模式，正確顯示紙張大小、邊界以及內容在頁面上的位置。
- 限制的列印：作者可以限制列印選項，例如可列印頁面的範圍。
- 品質選項：作者可以調整文件的列印品質，並讓使用者選取解析度和顏色選項。
- 多個列印工作階段：現在可以使用單一 PrintJob 實體來進行多個列印工作階段。應用程式可在每次顯示頁面設定和列印對話方塊時提供一致的設定。

列印工作流程變更

新的列印工作流程包含下列步驟：

- new PrintJob()：建立 PrintJob 實體 (或重複使用現有實體)。您可以在列印工作開始前或列印期間使用許多新的 PrintJob 屬性和方法 (例如 selectPaperSize())。

- PrintJob.showPageSetupDialog()：(選擇性) 顯示頁面設定對話方塊，而不啟動列印工作。
- PrintJob.start() 或 PrintJob.start2()：除了 start() 方法之外，還可以使用 start2() 方法來起始列印多工緩衝程序。start2() 方法可讓您選擇是否顯示 「列印」對話方塊，以及自訂對話方塊 (如果顯示)。
- PrintJob.addPage()：將內容新增至列印工作。與現有程序無異。
- PrintJob.send() 或 PrintJob.terminate()：將頁面傳送至選取的印表機，或者終止列印工作而不傳送。會在發生錯誤時終止列印工作。如果終止 PrintJob 實體，您還是可以重複使用它。不論是否將列印工作傳送至印表機或終止列印工作，在您重複使用 PrintJob 實體時，都會保留目前列印設定。

頁面設定對話方塊

如果目前環境支援，showPageSetupDialog() 方法可顯示作業系統的 「頁面設定」對話方塊。呼叫此方法前，請務必檢查 supportsPageSetupDialog 屬性。下列是簡單的範例：

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
//check for static property supportsPageSetupDialog of PrintJob class
if (PrintJob.supportsPageSetupDialog) {
    myPrintJob.showPageSetupDialog();
}
```

您可以使用 PrintUIOptions 類別選擇性地呼叫方法，以控制要在 「頁面設定」對話方塊中顯示的選項。可以設定最小和最大頁數。下列範例限制只列印前三頁：

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
if (PrintJob.supportsPageSetupDialog) {
    var uiOpt:PrintUIOptions = new PrintUIOptions();
    uiOpt.minPage = 1;
    uiOpt.maxPage = 3;
    myPrintJob.showPageSetupDialog(uiOpt);
}
```

變更列印設定

PrintJob 實體的設定建構完成後，可隨時變更。這包括在 addPage() 呼叫之間以及列印工作傳送或終止之後變更設定。部分設定 (例如 printer 屬性) 會套用至整個列印工作，而非個別頁面。必須先設定那些設定，才能呼叫 start() 或 start2()。

呼叫 selectPaperSize() 方法可設定 「頁面設定」和 「列印」對話方塊中的預設紙張大小。也可在列印工作期間呼叫該方法，設定頁面範圍的紙張大小。它使用 PaperSize 類別中定義的常數進行呼叫，在此範例中，選取 10 號信封大小：

```
import flash.printing.PrintJob;
import flash.printing.PaperSize;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.selectPaperSize(PaperSize.ENV_10);
```

使用 printer 屬性取得或設定目前列印工作的印表機名稱。它會預設為預設印表機的名稱。如果沒有可用的印表機或系統不支援列印，則 printer 屬性是 null。若要變更印表機，則請先使用 printers 屬性來取得可用印表機清單。該屬性是 Vector，而其 String 元素是可用的印表機名稱。將 printer 屬性設為其中一個 String 值，讓該印表機成為作用中印表機。作用中列印工作的 printer 屬性無法變更。在成功呼叫 start() 或 start2() 後，以及在傳送或終止工作前，若嘗試進行變更，會造成失敗。下列是設定此屬性的範例：

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.printer = "HP_LaserJet_1";
myPrintJob.start();
```

copies 屬性會取得作業系統 「列印」對話方塊中設定的複本數量值。firstPage 和 lastPage 屬性則會取得頁面範圍。orientation 屬性則會取得紙張方向設定。這些屬性可以設定以覆寫 「列印」對話方塊中的值。下列範例會設定這些屬性：

```
import flash.printing.PrintJob;
import flash.printing.PrintJobOrientation;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.copies = 3;
myPrintJob.firstPage = 1;
myPrintJob.lastPage = 3;
myPrintJob.orientation = PrintJobOrientation.LANDSCAPE;
```

下列與 PrintJob 有關的唯讀設定，可提供目前印表機設定的有用資訊：

- paperArea：印表機紙張的矩形範圍 (以點為單位)。
- printableArea：可列印區域的矩形範圍 (以點為單位)。
- maxPixelsPerInch：目前印表機的實體解析度 (以每英寸像素為單位)。
- isColor：目前印表機列印顏色的能力 (如果目前印表機可列印彩色，則會傳回 true)。

請參閱第 805 頁 「列印範例：頁面設定和列印選項」。

列印範例：多重頁面列印

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當列印的內容超過一頁時，您可以分別將每一個頁面與不同的 Sprite 建立關聯 (在本範例中，則為 sheet1 和 sheet2)，然後為每一個 Sprite 使用 PrintJob.addPage()。下列程式碼將說明如何使用這項技巧：

```
package
{
    import flash.display.MovieClip;
    import flash.printing.PrintJob;
    import flash.printing.PrintJobOrientation;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.geom.Rectangle;

    public class PrintMultiplePages extends MovieClip
    {
        private var sheet1:Sprite;
        private var sheet2:Sprite;

        public function PrintMultiplePages():void
        {
            init();
            printPages();
        }

        private function init():void
        {
            sheet1 = new Sprite();
```

```
        createSheet(sheet1, "Once upon a time...", {x:10, y:50, width:80, height:130});
        sheet2 = new Sprite();
        createSheet(sheet2, "There was a great story to tell, and it ended quickly.\n\nThe end.", null);
    }

    private function createSheet(sheet:Sprite, str:String, imgValue:Object):void
    {
        sheet.graphics.beginFill(0xEEEEEE);
        sheet.graphics.lineStyle(1, 0x000000);
        sheet.graphics.drawRect(0, 0, 100, 200);
        sheet.graphics.endFill();

        var txt:TextField = new TextField();
        txt.height = 200;
        txt.width = 100;
        txt.wordWrap = true;
        txt.text = str;

        if (imgValue != null)
        {
            var img:Sprite = new Sprite();
            img.graphics.beginFill(0xFFFFFF);
            img.graphics.drawRect(imgValue.x, imgValue.y, imgValue.width, imgValue.height);
            img.graphics.endFill();
            sheet.addChild(img);
        }
        sheet.addChild(txt);
    }

    private function printPages():void
    {
        var pj:PrintJob = new PrintJob();
        var pagesToPrint:uint = 0;
        if (pj.start())
        {
            if (pj.orientation == PrintJobOrientation.LANDSCAPE)
            {
                throw new Error("Page is not set to an orientation of portrait.");
            }

            sheet1.height = pj.pageHeight;
            sheet1.width = pj.pageWidth;
            sheet2.height = pj.pageHeight;
            sheet2.width = pj.pageWidth;

            try
            {
                pj.addPage(sheet1);
                pagesToPrint++;
            }
            catch (error:Error)
            {
            // Respond to error.
```

```
                }

                try
                {
                    pj.addPage(sheet2);
                    pagesToPrint++;
                }
                catch (error:Error)
                {
                    // Respond to error.
                }

                if (pagesToPrint > 0)
                {
                    pj.send();
                }
            }
        }
    }
}
```

列印範例：縮放、裁切和回應

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

這某些情況下，您可能會想要在列印時調整顯示物件的大小 (或其他屬性)，以便就顯示物件在螢幕上顯示與列印在紙張上之間的差異進行調整。當您先調整顯示物件的屬性再進行列印時 (例如，使用 scaleX 和 scaleY 屬性)，請注意，如果物件在縮放後大於所定義的列印區域矩形，則物件會遭到裁切。您可能也要在列印頁面之後重設這些屬性。

下列程式碼將對 txt 顯示物件 (不包括綠色方塊背景) 的尺寸進行縮放，結果造成文字欄位以指定的矩形尺寸被裁切。在列印之後，文字欄位會還原成在螢幕上顯示的原始大小。如果使用者從作業系統的 「列印」對話方塊取消列印工作，Flash 執行階段中的內容也會隨之變更，以提示使用者列印工作已經取消。

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.display.Stage;
    import flash.geom.Rectangle;

    public class PrintScaleExample extends Sprite
    {
        private var bg:Sprite;
        private var txt:TextField;

        public function PrintScaleExample():void
        {
            init();
            draw();
            printPage();
        }

        private function printPage():void
        {
            var pj:PrintJob = new PrintJob();
            txt.scaleX = 3;
            txt.scaleY = 2;
```

```
            if (pj.start())
            {
                trace(">> pj.orientation: " + pj.orientation);
                trace(">> pj.pageWidth: " + pj.pageWidth);
                trace(">> pj.pageHeight: " + pj.pageHeight);
                trace(">> pj.paperWidth: " + pj.paperWidth);
                trace(">> pj.paperHeight: " + pj.paperHeight);

                try
                {
                    pj.addPage(this, new Rectangle(0, 0, 100, 100));
                }
                catch (error:Error)
                {
                    // Do nothing.
                }
                pj.send();
            }
            else
            {
                txt.text = "Print job canceled";
            }
            // Reset the txt scale properties.
            txt.scaleX = 1;
            txt.scaleY = 1;
        }

        private function init():void
        {
            bg = new Sprite();
            bg.graphics.beginFill(0x00FF00);
            bg.graphics.drawRect(0, 0, 100, 200);
            bg.graphics.endFill();

            txt = new TextField();
            txt.border = true;
            txt.text = "Hello World";
        }

        private function draw():void
        {
            addChild(bg);
            addChild(txt);
            txt.x = 50;
            txt.y = 50;
        }
    }
}
```

列印範例：頁面設定和列印選項

Adobe AIR 2 以及更新的版本

下列範例起始下列 PrintJob 設定：複本數量、紙張大小 (legal) 和頁面方向 (橫向)。它會強制讓 「頁面設定」對話方塊優先顯示，然後再顯示 「列印」對話方塊才開始列印工作。

```
package
{
    import flash.printing.PrintJob;
    import flash.printing.PrintJobOrientation;
    import flash.printing.PaperSize;
    import flash.printing.PrintUIOptions;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.display.Stage;
    import flash.geom.Rectangle;

    public class PrintAdvancedExample extends Sprite
    {
        private var bg:Sprite = new Sprite();
        private var txt:TextField = new TextField();
        private var pj:PrintJob = new PrintJob();
        private var uiOpt:PrintUIOptions = new PrintUIOptions();

        public function PrintAdvancedExample():void
        {
            initPrintJob();
            initContent();
            draw();
            printPage();
        }

        private function printPage():void
        {
            //test for dialog support as a static property of PrintJob class
            if (PrintJob.supportsPageSetupDialog)
            {
                pj.showPageSetupDialog();
            }
            if (pj.start2(uiOpt, true))
            {
                try
                {
                    pj.addPage(this, new Rectangle(0, 0, 100, 100));
                }
                catch (error:Error)
                {
                    // Do nothing.
                }
                pj.send();
            }
            else
            {
                txt.text = "Print job terminated";
                pj.terminate();
            }
        }

        private function initContent():void
        {
            bg.graphics.beginFill(0x00FF00);
            bg.graphics.drawRect(0, 0, 100, 200);
            bg.graphics.endFill();

            txt.border = true;
```

```
            txt.text = "Hello World";
        }

        private function initPrintJob():void
        {
            pj.selectPaperSize(PaperSize.LEGAL);
            pj.orientation = PrintJobOrientation.LANDSCAPE;
            pj.copies = 2;
            pj.jobName = "Flash test print";
        }

        private function draw():void
        {
            addChild(bg);
            addChild(txt);
            txt.x = 50;
            txt.y = 50;
        }
    }
}
```

第 55 章 Geolocation

如果裝置支援地理位置，則可以使用 geolocation API 來取得裝置目前的地理位置。如果裝置支援這個功能，則可以取得地理位置資訊。這項資訊包括位置的最新變更高度、精確度、航向、速度與時間戳記。

Geolocation 類別會傳送 update 事件，以回應裝置的位置感應器。update 事件是 GeolocationEvent 物件。

更多說明主題

flash.sensors.Geolocation

flash.events.GeolocationEvent

偵測地理位置變更

若要使用地理位置感應器，請初始化 Geolocation 物件，並註冊它所傳送的 update 事件。update 事件是 Geolocation 事件物件。這個事件有八個屬性：

- altitude：高度 (以公尺為單位)。
- heading：移動方向 (以正北方作為基準，以角度為單位)。
- horizontalAccuracy：水平精確度 (以公尺為單位)
- latitude：緯度 (以角度為單位)。
- longitude：經度 (以角度為單位)。
- speed：速度 (每秒公尺數)。
- timestamp：從執行階段初始化之後，事件時間的毫秒數。
- verticalAccuracy：垂直精確度 (以公尺為單位)。

timestamp 屬性是 int 物件。其他則是 Number 物件。

以下是在文字欄位中顯示地理位置資料的基本範例：

```
var geo:Geolocation;
if (Geolocation.isSupported)
{
    geo = new Geolocation();
    geo.addEventListener(GeolocationEvent.UPDATE, updateHandler);
}
else
{
    geoTextField.text = "Geolocation feature not supported";
}
function updateHandler(event:GeolocationEvent):void
{
    geoTextField.text = "latitude: " + event.latitude.toString() + "\n"
            + "longitude: " + event.longitude.toString() + "\n"
            + "altitude: " + event.altitude.toString()
            + "speed: " + event.speed.toString()
            + "heading: " + event.heading.toString()
            + "horizontal accuracy: " + event.horizontalAccuracy.toString()
            + "vertical accuracy: " + event.verticalAccuracy.toString()
}
```

若要使用這個範例，請確認建立 geoTextField 文字欄位，並先將它新增至顯示清單，再使用這個程式碼。

您可以呼叫 Geolocation 物件的 setRequestedUpdateInterval() 方法，調整地理位置事件的預定時間間隔。這個方法接受一個參數 (interval)，而這個參數是要求的更新間隔 (毫秒)：

```
var geo:Geolocation = new Geolocation();
geo.setRequestedUpdateInterval(10000);
```

地理位置更新之間的實際時間可能大於或小於這個值。在更新間隔中所做的任何變更會影響所有註冊的偵聽程式。如果您未呼叫 setRequestedUpdateInterval() 方法，則應用程式會根據裝置的預設間隔來接收更新。

使用者可以防止應用程式存取地理位置資料。例如，應用程式嘗試取得地理位置資料時，iPhone 會提示使用者。回應提示時，使用者可以拒絕應用程式存取地理位置資料。使用者不允許存取地理位置資料時，Geolocation 物件會傳送 status 事件。此外，Geolocation 物件具有 muted 屬性。地理位置感應器無法使用時，此屬性會設為 true。Geolocation 物件會在 muted 屬性變更時傳送 status 事件。下列程式碼顯示如何偵測地理位置資料無法使用的時機：

```
package
{
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.GeolocationEvent;
    import flash.events.MouseEvent;
    import flash.events.StatusEvent;
    import flash.sensors.Geolocation;
    import flash.text.TextField;
    import flash.text.TextFormat;

    public class GeolocationTest extends Sprite
    {

        private var geo:Geolocation;
        private var log:TextField;

        public function GeolocationTest()
        {
            super();
            stage.align = StageAlign.TOP_LEFT;
            stage.scaleMode = StageScaleMode.NO_SCALE;
            setUpTextField();

            if (Geolocation.isSupported)
            {
                geo = new Geolocation();
                if (!geo.muted)
                {
                    geo.addEventListener(GeolocationEvent.UPDATE, geoUpdateHandler);
                }
                geo.addEventListener(StatusEvent.STATUS, geoStatusHandler);
            }
            else
            {
                log.text = "Geolocation not supported";
            }
        }

        public function geoUpdateHandler(event:GeolocationEvent):void
        {
            log.text = "latitude : " + event.latitude.toString() + "\n";
            log.appendText("longitude : " + event.longitude.toString() + "\n");
        }
```

```
        public function geoStatusHandler(event:StatusEvent):void
        {
            if (geo.muted)
                geo.removeEventListener(GeolocationEvent.UPDATE, geoUpdateHandler);
            else
                geo.addEventListener(GeolocationEvent.UPDATE, geoStatusHandler);
        }

        private function setUpTextField():void
        {
            log = new TextField();
            var format:TextFormat = new TextFormat("_sans", 24);
            log.defaultTextFormat = format;
            log.border = true;
            log.wordWrap = true;
            log.multiline = true;
            log.x = 10;
            log.y = 10;
            log.height = stage.stageHeight - 20;
            log.width = stage.stageWidth - 20;
            log.addEventListener(MouseEvent.CLICK, clearLog);
            addChild(log);
        }
        private function clearLog(event:MouseEvent):void
        {
            log.text = "";
        }
    }
}
```

備註：第一代 iPhone (不含 GPS 裝置) 只會偶爾傳送 update 事件。在這些裝置上，Geolocation 物件一開始會傳送一個或兩個 update 事件。然後在資訊大幅變更時傳送 update 事件。

檢查地理位置支援

使用 Geolocation.isSupported 屬性測試執行階段環境是否可以使用這個功能：

```
if (Geolocation.isSupported)
{
    // Set up geolocation event listeners and code.
}
```

目前，只有針對 iPhone 的 ActionScript 型應用程式和 Flash Lite 4 支援地理位置。如果 Geolocation.isSupported 在執行階段是 true，則支援地理位置。

部分 iPhone 機型沒有 GPS 裝置。這些機型使用其他方法 (例如行動電話三角定位) 取得地理位置資料。在這些機型中，或任何已停用 GPS 的 iPhone 上，Geolocation 物件可能只會傳送一個或兩個初始 update 事件。

第 56 章 國際化應用程式

Flash Player 10.1 以及更新的版本，**Adobe AIR 2.0** 以及更新的版本

flash.globalization 套件可讓您輕鬆建立國際化軟體，以適應不同語言與地區慣例。

更多說明主題

flash.globalization 套件

第 825 頁 「當地語系化應用程式」

Charles Bihis：想要將 Flex/AIR 應用程式當地語系化嗎？

國際化應用程式的基本概念

「全球化」和 「國際化」這兩個術語有時候可以交互使用。但是這些術語的大部分定義為：「全球化」指商業和工程處理程序的結合，而 「國際化」則僅指工程處理程序。

下列為一些重要術語的定義：

全球化 準備和發行全球化產品與公司活動需要廣泛的工程和商業程序。「全球化」由工程活動 (例如國際化、當地語系化及文化化) 和商業活動 (例如產品管理、財務規劃、行銷及法律工作) 所組成。全球化 (Globalization) 有時會縮寫為 G11n (其代表字母 G，然後依序是 11 個字母和字母 n)。「全球化是企業的終極目標」。

國際化 整合產品的工程處理程序，讓產品不需重新設計或重新編譯，即可處理多個語言、Script 和文化慣例 (包含貨幣、排序規則、數字和日期格式等)。這個程序可以分成兩組活動：啟用和當地語系化。國際化 (Internationalization) 有時候也稱為「全球可用」，有時候則縮寫為 I18n。「工程師是國際化的推手」。

當地化 改編產品或服務以用於特定語言、文化以及當地所需外觀的程序。「當地語系化」有時候會縮寫為 L10n。 「當地語系化是譯者的工作」。

文化化 可針對文化特定需求，開發或改編特定功能的工程處理程序。範例包含 Adobe InDesign 的日文發佈功能，以及 Adobe Acrobat 的印章支援功能。

其他重要的國際化術語定義如下：

字元集 一種語言或一組語言使用的字元。字元集包含國家字元、特殊字元 (例如標點符號和數學符號)、數字位數以及電腦控制字元。

定序 針對指定的地區設定，以適當的順序排序文字。

地區 用來代表地理、政治或文化地區 (在很多情況下代表單一國家 / 地區) 所使用之語言和文化慣例的值。唯一地區設定識別碼 (地區設定 ID) 就代表這個值。地區設定 ID 是用來尋找一組提供地區設定特定支援的地區設定資料。這項支援適用於數字和日期的度量單位、剖析和格式化等。

資源組合包 針對應用程式所使用的地區設定而建立的一組已儲存的地區設定特定元素。資源組合包一般包含應用程式之使用者介面中的所有文字元素。在組合包內，會將這些元素翻譯為指定地區設定適用的語言。同時也可以包含其他設定，用以變更特定地區設定的使用者介面版面或行為。資源組合包可以包含其他地區設定特定的媒體類型，或其他媒體類型的參考。

flash.globalization 套件概觀

Flash Player 10.1 以及更新的版本，**Adobe AIR 2.0** 以及更新的版本

flash.globalization 套件使用基礎作業系統的文化支援功能。讓您輕鬆撰寫應用程式，以符合個別使用者的文化慣例。

套件的主要類別包含下列各項：

- Collator 類別，管理字串的排序和比對
- CurrencyFormatter 類別，將數字格式化為貨幣金額字串，以及將輸入字串剖析為貨幣金額和符號
- DateTimeFormatter 類別，格式化日期值
- LocaleID 類別，擷取特定地區設定的資訊
- NumberFormatter 類別，格式化和剖析數值
- StringTools 類別，處理字串的區分地區設定的大小寫轉換

flash.globalization 套件與資源當地語系化

flash.globalization 套件不會處理資源當地語系化。不過，您可以使用 flash.globalization 地區設定 ID，做為利用其他技術擷取當地語系化資源的索引鍵值。例如，您可以使用 ResourceManager 和 ResourceBundle 類別來當地語系化透過 Flex 所建置的應用程式資源。如需詳細資訊，請參閱當地語系化 Flex 應用程式。

Adobe AIR 1.1 也包含協助當地語系化 AIR 應用程式的一些功能，如第 826 頁「當地語系化 AIR 應用程式」所述)。

國際化應用程式的一般方法

下列步驟說明如何使用 flash.globalization 套件國際化應用程式的高階常見方法：

1. 判斷或設定地區設定。
2. 建立服務類別的實體 (Collator、CurrencyFormatter、DateTimeFormatter、NumberFormatter 或 StringTools)。
3. 使用 lastOperationStatus 屬性檢查錯誤和後援。
4. 使用地區設定特有設定的格式和顯示資訊。

下一個步驟是載入和顯示地區設定特有的字串和使用者介面資源。這個步驟包括以下工作：

- 使用 autolayout 功能來調整 UI 的大小，以符合字串長度
- 選擇正確的字體並支援後援字體
- 使用 FTE 文字引擎以支援其他撰寫系統
- 確定已正確處理輸入法編輯器

檢查錯誤和後援

flash.globalization 服務類別都遵循類似的樣式來識別錯誤。也會共用樣式，從無法使用的要求地區設定，後援為使用者作業系統所支援的地區設定。

下列範例顯示如何在初始化服務類別時檢查錯誤和後援。每種服務類別的 lastOperationStatus 屬性都會指出最新的方法呼叫是否已觸發其錯誤或警告。

```
var nf:NumberFormatter = new NumberFormatter("de-DE");
if(nf.lastOperationStatus != LastOperationStatus.NO_ERROR)
{
    if(nf.lastOperationStatus == LastOperationStatus.USING_FALLBACK_WARNING)
    {
        // perform fallback logic here, if needed
        trace("Warning - Fallback locale ID: " + nf.actualLocaleIDName);
    }
    else
    {
        // perform error handling logic here, if needed
        trace("Error: " + nf.lastOperationStatus);
    }
}
```

這個範例只會追蹤是否使用後援地區設定 ID 或發生錯誤的訊息。您的應用程式可以執行其他錯誤處理邏輯 (需要時)。例如，您可以向使用者顯示訊息，或強制應用程式使用支援的特定地區設定。

判斷地區設定

Flash Player 10.1 以及更新的版本，**Adobe AIR 2.0** 以及更新的版本

地區設定可識別國家或地區之語言和文化慣例的特定組合。

您可以安全地以字串來管理地區設定識別碼。不過，利用 LocaleID 類別，您可以取得與地區設定相關的其他資訊。

您可以如下建立 LocaleID 物件：

```
var locale:LocaleID = new LocaleID("es-MX");
```

建立 LocaleID 物件之後，就可以擷取地區設定 ID 的資料。請使用 getKeysAndValues()、getLanguage()、getRegion()、getScript()、getVariant() 和 isRightToLeft() 方法，以及 name 屬性。

從這些方法和屬性擷取的值，可以反映從地區設定識別碼直接取出地區設定的其他資訊。

應用程式在建立區分地區設定的服務 (例如日期格式化程式) 時，必須指定所要的地區設定。所支援地區設定的清單會依作業系統而不同；因此，要求的地區設定可能會無法使用。

Flash Player 會先嘗試比對所要求之地區設定的語言碼。然後，尋找相符的撰寫系統 (Script) 和地區，以嘗試修改地區設定。例如：

```
var loc:LocaleID = new LocaleID("es");
trace(loc.getLanguage()); // es
trace(loc.getScript()); // Latn
trace(loc.getRegion()); // ES
```

在這個範例中，LocaleID() 建構函式擷取的資料是有關最符合該使用者之語言碼 "es" 的地區設定。

設定地區設定 ID

許多方法都可以設定應用程式的目前地區設定，包括下列各項：

- 將單一地區設定 ID 硬式編碼至應用程式。這是常見的做法，但不支援國際化應用程式。
- 使用使用者之作業系統 (或瀏覽器) 或其他使用者偏好設定的地區設定 ID 偏好設定。這種技術通常會產生使用者的最佳地區設定，但這個地區設定不一定正確。風險是作業系統設定未反映使用者的實際偏好設定。例如，使用者可以使用共用電腦，但無法變更作業系統的偏好地區設定。

- 根據使用者的偏好設定設定地區設定 ID 之後，讓使用者從受支援地區設定的清單中進行選取。如果您的應用程式可以支援多個地區設定，則這個策略通常是最佳選項。

您可以如下實作第三個選項：

1 從使用者設定檔、瀏覽器設定、作業系統設定或 Cookie，擷取使用者偏好地區設定或語言的清單 (您的應用程式需要自行實作這個邏輯。flash.globalization 元件庫不支援直接讀取這類偏好設定)。
2 判斷您的應用程式支援其中哪些地區設定，而且預設會選取最佳地區設定。使用 LocaleID.determinePreferredLocales() 方法，根據使用者的偏好地區設定和作業系統所支援的地區設定，來尋找使用者的最佳地區設定。
3 如果不滿意預設地區設定，則提供方法讓使用者變更預設地區設定。

其他地區設定和語言類別限制

fl.lang.Locale 類別可讓您使用含有字串值的資源組合包，來根據地區設定取代文字字串。不過，這個類別不支援其他國際化功能 (例如數字、貨幣或日期格式化、排序和比對等)。此外，只有 Flash Professional 才提供這個類別。

您也可以使用 flash.system.Capabilities.language 屬性，來擷取作業系統的目前語言碼設定。不過，這個屬性只會擷取由兩個字元組成的 ISO 639-1 語言碼 (非完整地區設定 ID)，而且只支援特定一組的地區設定。

透過 AIR 1.5，您可以使用 flash.system.Capabilities.languages 屬性。這個屬性會提供使用者偏好的使用者介面語言陣列。因此，沒有 Capabilities.language 中的限制。

格式化數字

Flash Player 10.1 以及更新的版本，**Adobe AIR 2.0** 以及更新的版本

數值的顯示格式會因地區而有極大的不同。例如，以下是特定地區設定格式化數字 123456.78 的方式：

地區	數字格式
en-US (英文，美國)	-123,456.78
de-DE (德文，德國)	-123.456,78
fr-FR (法文，法國)	-123 456,78
de-CH (德文，瑞士)	-123'456.78
en-IN (英文，印度)	-1,23,456.78
多個阿拉伯文地區設定	123,456.78-

多種因素會影響數字格式，包括：

- 分隔符號。小數分隔符號會放在數字的整數與小數部分之間。這可能是句點、逗點或其他字元。分組分隔符號或千分隔符號可能是句點、逗點、不分行空格或其他字元。
- 分組模式。小數點左邊每個分組分隔符號之間的位數可能是二位數或三位數或其他值。
- 負數指示器。負數在顯示時會在數字左邊或右邊顯示減號，如果是財務應用程式，則是顯示於小括弧內。例如，負 19 可能顯示為 -19、19- 或 (19)。
- 開頭和結尾零。部分文化慣例會在顯示的數字中增加開頭和結尾零。例如，在其他選項中，值 0.17 可能顯示為 .17、0.17 或 0.170。
- 數字字元集合。多種語言 (包括印地文、阿拉伯文和日文) 使用不同的數字字元集合。flash.globalization 套件支援任何對應至數字 0-9 的數字字元集。

NumberFormatter 類別在格式化數值時會考量所有這些因素。

使用 NumberFormatter 類別

NumberFormatter 類別會根據特定地區設定的慣例來格式化數值 (類型為 int、uint 或 Number 的數值)。

下列範例顯示如何利用使用者之作業系統提供的預設格式屬性，將數字格式化。這是最簡單的方式：

```
var nf:NumberFormatter = new NumberFormatter(LocaleID.DEFAULT);
trace(nf.formatNumber(-123456.789))
```

結果會根據使用者的地區設定和使用者偏好設定而不同。例如，如果使用者的地區設定是 fr-FR，則格式化的值會是：

-123.456,789

如果您只想要格式化特定地區設定的數字，而不管使用者的設定，則請特別設定地區設定名稱。例如：

```
var nf:NumberFormatter = new NumberFormatter("de-CH");
trace(nf.formatNumber(-123456.789));
```

在此情況下，結果為：

-123'456.789

formatNumber() 方法接受 Number 的參數。NumberFormatter 類別也有接收 int 做為輸入的 formatInt() 方法，以及接受 uint 做為輸入的 formatUint() 方法。

您可以設定 NumberFormatter 類別的屬性，明確控制格式邏輯 (如這個範例所示)：

```
var nf:NumberFormatter = new NumberFormatter("de-CH");
nf.negativeNumberFormat = 0;
nf.fractionalDigits = 5;
nf.trailingZeros = true;
nf.decimalSeparator = ",";
nf.useGrouping = false;
trace(nf.formatNumber(-123456.789)); //(123456.78900)
```

這個範例會先建立 NumberFormatter 物件，然後：

- 將負數格式設為使用小括弧 (與財務應用程式相同)；
- 將小數分隔符號後面的位數設為 5；
- 指定使用結尾零，確保具有五位數；
- 將小數分隔符號設為逗點；
- 告訴格式化程式不要使用任何分組分隔符號。

備註：上述部分屬性變更時，產生的數字格式就不再對應所指定地區設定的偏好格式。上述部分屬性只適用於不一定要區分地區設定時、需要更詳細控制格式的單一部分 (例如結尾零的數目) 時，或是使用者直接要求變更時 (例如，透過 Windows「控制台」)。

剖析含有數值的字串

NumberFormatter 類別也可以從符合地區設定特有格式需求的字串中取出數值。NumberFormatter.parseNumber() 方法會從字串取出單一數值。例如：

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String =  "-1,234,567.890"
var parsedNumber:Number = nf.parseNumber(inputNumberString);
trace("Value:" + parsedNumber); // -1234567.89
trace("Status:" + nf.lastOperationStatus); // noError
```

parseNumber() 方法處理的字串只能包含位數和數字格式字元 (例如負號和分隔符號)。如果字串包含其他字元，則會設定錯誤碼：

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String =  "The value is 1,234,567.890"
var parsedNumber:Number = nf.parseNumber(inputNumberString);
trace("Value:" + parsedNumber); // NaN
trace("Status:" + nf.lastOperationStatus); // parseError
```

若要從含有其他字母字元的字串取出數字，請使用 NumberFormatter.parse() 方法：

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String = "The value is 123,456,7.890";
var parseResult:NumberParseResult = nf.parse(inputNumberString);
trace("Value:" + parseResult.value); // 1234567.89
trace("startIndex: " + parseResult.startIndex); // 14
trace("Status:" + nf.lastOperationStatus); // noError
```

parse() 方法傳回的 NumberParseResult 物件，會在其值屬性中包含剖析過的數值。startIndex 屬性指出找到的第一個數值字元的索引。您可以使用 startIndex 和 endIndex 屬性，擷取位數前後的字串部分。

格式化貨幣值

Flash Player 10.1 以及更新的版本，**Adobe AIR 2.0** 以及更新的版本

貨幣值的顯示格式不同，就和數字格式一樣。例如，以下是特定地區設定格式化美元值 $123456.78 的方式：

地區	數字格式
en-US (英文，美國)	$123,456.78
de-DE (德文，德國)	123.456,78 $
en-IN (英文，印度)	$ 1,23,456.78

貨幣格式包含數字格式的所有因素，而且再加上這些其他因素：

- 貨幣 ISO 代碼。實際使用的地區設定 ISO 4217 貨幣代碼，是由三個字母組成 (例如 USD 或 EUR)。
- 貨幣符號。實際使用的地區設定貨幣符號或字串 (例如 $ 或 €)。
- 負值貨幣格式。定義貨幣金額中，貨幣符號與負號或小括弧相對於數字部分的位置。
- 正值貨幣格式。定義貨幣金額中，貨幣符號相對於數字部分的位置。

使用 CurrencyFormatter 類別

CurrencyFormatter 類別會根據特定地區設定的慣例，將數值格式化為含有貨幣字串和格式化數字的字串。

當您初始化新的 CurrencyFormatter 物件時，該物件會將其貨幣設為給定地區設定的預設貨幣。

下列範例顯示使用德文地區設定所建立的 CurrencyFormatter 物件，假設貨幣金額是歐元：

```
var cf:CurrencyFormatter = new CurrencyFormatter( "de-DE" );
trace(cf.format(1234567.89)); // 1.234.567,89 EUR
```

在大部分情況下，請不要依賴地區設定的預設貨幣。如果不支援使用者的預設地區設定，則 CurrencyFormatter 類別會指定後援地區設定。後援地區設定可能會使用不同的預設貨幣。此外，您一般也會想要使用者看到正確的貨幣格式，即使金額的單位並非使用者的當地貨幣。例如，加拿大使用者想要看到德國公司的歐元價格，但是要以加拿大樣式進行格式化。

CurrencyFormatter.setCurrency() 方法指定確切的貨幣字串以及使用的貨幣符號。

下列範例是對加拿大的法區使用者顯示歐元的貨幣金額：

```
var cf:CurrencyFormatter = new CurrencyFormatter( "fr-CA" );
cf.setCurrency("EUR", "€ ");
trace(cf.format(1234567.89)); // 1.234.567,89 EUR
```

setCurrency() 方法也可以用來設定明確的貨幣符號，以減少混淆。例如：

```
cf.setCurrency("USD","US$");
```

format() 方法預設會顯示三個字元的 ISO 4217 貨幣代碼，而非貨幣符號。ISO 4217 貨幣代碼十分明確，因此並不需要當地語系化。不過，許多使用者都偏好看到貨幣符號，而非 ISO 代碼。

CurrencyFormatter 類別可協助您決定格式化的貨幣字串所使用的貨幣符號 (如美元符號或歐元符號)，或使用三個字元的 ISO 貨幣字串 (例如 USD 或 EUR)。例如，針對加拿大使用者，加拿大幣金額會顯示為 $200。不過，針對美國使用者，則會顯示為 CAD 200。使用 formattingWithCurrencySymbolIsSafe() 方法，決定指定使用者地區設定的情況下，金額的貨幣符號為模稜兩或不正確。

下列範例會將歐元的值格式化為 en-US 地區設定的格式。根據使用者的地區設定，輸出字串會使用 ISO 貨幣代碼或貨幣符號。

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-CA");

if (cf.formattingWithCurrencySymbolIsSafe("USD"))
{
    trace(cf.format(1234567.89, true)); // $1,234,567.89
}
else
{
    cf.setCurrency("USD", "$");
    trace(cf.format(1234567.89)); // USD1,234,567.89
}
```

剖析含有貨幣值的字串

CurrencyFormatter 類別也可以從符合地區設定特有格式需求的輸入字串中取出貨幣金額和貨幣字串。CurrencyFormatter.parse() 方法會將剖析過的金額和貨幣字串，儲存於 CurrencyParseResult 物件 (如下所示)：

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-US" );
var inputCurrencyString:String = "(GBP 123,56,7.890)";
var parseResult:CurrencyParseResult = cf.parse(inputCurrencyString);
trace("parsed amount: " + parseResult.value); // -1234567.89
trace("currencyString: " + parseResult.currencyString ); // GBP
```

輸入字串的貨幣字串位置可以包含貨幣符號、貨幣 ISO 代碼和其他文字字元。貨幣字串、負數指示器和數值的位置會符合 negativeCurrencyFormat 和 positiveCurrencyFormat 屬性所指定的格式。例如：

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-US" );
var inputCurrencyString:String = "Total $-123,56,7.890";
var parseResult:CurrencyParseResult = cf.parse(inputCurrencyString);
trace("status: " + cf.lastOperationStatus ); // parseError
trace("parsed amount: " + parseResult.value); // NaN
trace("currencyString: " + parseResult.currencyString ); //
cf.negativeCurrencyFormat = 2;
parseResult = cf.parse(inputCurrencyString);
trace("status: " + cf.lastOperationStatus ); // noError
trace("parsed amount: " + parseResult.value); // -123567.89
trace("currencyString: " + parseResult.currencyString ); // Total $
```

在這個範例中，輸入字串的貨幣字串後面會接著減號和數字。不過，en-US 地區設定的預設 negativeCurrencyFormat 值指定先顯示負值指示器。因此，parse() 方法會產生錯誤，而剖析過的值是 NaN。

在它將 negativeCurrencyFormat 設為 2 (其指定先顯示貨幣字串) 之後，parse() 方法就會成功。

格式化日期和時間

Flash Player 10.1 以及更新的版本，**Adobe AIR 2.0** 以及更新的版本

日期與時間值的顯示格式也會因地區而有極大的不同。例如，以下是以特定地區設定的簡短格式來顯示 1962 年 1 月 2 日 1:01 PM：

地區	日期與時間格式
en-US (英文，美國)	1/2/62 1:01pm
fr-FR (法文，法國)	2/1/62 13:01
ja-JP (日文，日本)	1962/2/1 13:01

使用 DateTimeFormatter 類別

DateTimeFormatter 類別會根據特定地區設定的慣例，將 Date 值格式化為日期與時間字串。

模式字串包含要取代為日期或時間值的一串字母，而格式則遵循這種模式字串。例如，在 「yyyy/MM」模式中，「yyyy」字元會取代為四位數年份，後面依序接著 「/」字元和兩位數月份。

您可以使用 setDateTimePattern() 方法明確設定模式字串。不過，最好是根據使用者的地區設定和作業系統偏好設定，來自動設定模式。這種作法有助於確定結果符合文化。

DateTimeFormatter 可以使用三種標準樣式 (LONG、MEDIUM 和 SHORT) 代表日期和時間，也可以使用 CUSTOM 模式。其中一個樣式可以用於日期，而第二個樣式則用於時間。每個樣式使用的實際模式會因作業系統而有些許的不同。

您可以在建立 DateTimeFormatter 物件時指定樣式。如果未指定 style 參數，則會預設為 DateTimeStyle.LONG。您稍後可以使用 setDateTimeStyles() 方法來變更樣式，如下列範例所示：

```
var date:Date = new Date(2009, 2, 27, 13, 1);
var dtf:DateTimeFormatter = new DateTimeFormatter("en-US",
    DateTimeStyle.LONG, DateTimeStyle.LONG);

var longDate:String = dtf.format(date);
trace(longDate); // March 27, 2009 1:01:00 PM

dtf.setDateTimeStyles(DateTimeStyle.SHORT, DateTimeStyle.SHORT);
var shortDate:String = dtf.format(date);
trace(shortDate); // 3/27/09 1:01 PM
```

當地語系化月份名稱和日期名稱

許多應用程式會以日曆或下拉式清單顯示月份名稱和星期幾名稱。

您可以使用 DateTimeFormatter.getMonthNames() 方法，擷取當地語系化的月份名稱清單。根據作業系統，可能可以使用完整和縮寫格式。傳遞 DateTimeNameStyle.FULL 值，可取得完整長度的月份名稱。傳遞 DateTimeNameStyle.LONG_ABBREVIATION 或 DateTimeNameStyle.SHORT_ABBREVIATION 值，可取得較短的版本。

在部分語言的日期格式中，月份名稱如果放在日期值的旁邊則會變更 (而成為它的所有格格式)。如果您想要單獨使用月份名稱，請將 DateTimeNameContext.STANDALONE 值傳遞給 getMonthNames() 方法。不過，若要在格式化日期中使用月份名稱，請傳遞 DateTimeNameContext.FORMAT 值。

```
var dtf:DateTimeFormatter = new DateTimeFormatter("fr-FR");
var months:Vector.<String> = dtf.getMonthNames(DateTimeNameStyle.FULL,
        DateTimeNameContext.STANDALONE);
trace(months[0]); // janvier
months = dtf.getMonthNames(DateTimeNameStyle.SHORT_ABBREVIATION,
         DateTimeNameContext.STANDALONE);
trace(months[0]); // janv.
```

DateTimeFormatter.getWeekdayNames() 方法提供當地語系化的星期幾名稱清單。getWeekdayNames() 方法接受的 nameStyle 和 context 參數，與 getMonthNames() 方法相同 。

```
var dtf:DateTimeFormatter = new DateTimeFormatter("fr-FR");
var weekdays:Vector.<String> = dtf.getWeekdayNames(DateTimeNameStyle.FULL,
           DateTimeNameContext.STANDALONE);
trace(weekdays[0]); // dimanche
weekdays = dtf.getWeekdayNames(DateTimeNameStyle.LONG_ABBREVIATION,
           DateTimeNameContext.STANDALONE);
trace(weekdays[0]); // dim.
```

此外，getFirstWeekday() 方法會傳回所選取地區設定中傳統上會標示為一週開始之日期的索引值。

排序和比較字串

Flash Player 10.1 以及更新的版本，**Adobe AIR 2.0** 以及更新的版本

定序是依正確順序排列項目的程序。定序規則會因地區設定而有明顯不同。如果您是排序清單或符合類似的項目 (例如在文字搜尋演算法中)，則規則也會不同。

排序時，即使是些許的差異 (例如大寫和小寫字母或讀音符號 (如重音)) 通常也十分重要。例如，字母 ö (含分音符號的 o) 視為最符合法文或英文的一般字母 o。不過，在瑞典文中，相同的字母則接在字母 z 後面。而且，在法文和部分其他語言中，文字中的最後重音差異可決定它在排序清單中的順序。

您通常會想要在搜尋時忽略大小寫或讀音符號差異，以增加找到相關相符項目的機會。例如，在法文文件中搜尋 「cote」字元，可能會傳回下列相符項目：「cote」、「côte」和 「coté」。

使用 Collator 類別

Collator 類別的主要方法是 compare() 方法 (主要用於排序) 和 equals() 方法 (用於比對值)。

下列範例顯示 compare() 和 equals() 方法的不同行為。

```
var words:Array = new  Array("coté", "côte");

var sorter:Collator = new Collator("fr-FR", CollatorMode.SORTING);
words.sort(sorter.compare);
trace(words); // côte,coté

var matcher:Collator = new Collator("fr-FR", CollatorMode.MATCHING);
if (matcher.equals(words[0], words[1]))
{
    trace(words[0] + " = " + words[1]); // côte = coté
}
```

這個範例會先針對 「法文 - 法國」地區設定，建立 SORTING 模式的 Collator 物件。然後排序兩個字，這兩個字只有讀音符號不同。這表示 SORTING 比較可區分重音和非重音字元。

排序的執行，是將 Collator 物件之 sort() 方法的參照當成參數傳遞給 Array.sort() 方法。這種技術是使用 Collator 物件控制排序順序的一種最有效方法。

這個範例接著會建立 MATCHING 模式的 Collator 物件。該 Collator 物件比較這兩個字時，會將它們視為相等。這表示 MATCHING 比較將重音和非重音字元視為相同。

自訂 Collator 類別的行為

Collator 類別預設會根據地區設定和使用者的系統偏好設定，使用取自作業系統的字串比較規則。您可以明確設定各種屬性，來自訂 compare() 和 equals() 方法的行為。下表列出屬性以及它們在比較時的效果：

Collator 屬性	特效
numericComparison	控制將數字字元視為數字或文字。
ignoreCase	控制是否忽略大小寫差異。
ignoreCharacterWidth	控制是否將部分中文和日文字元的全形和半形格式視為相同。
ignoreDiacritics	控制是否將使用相同基礎字元但重音不同或讀音符號不同的字串視為相同。
ignoreKanaType	控制是否將只有假名字元類型不同的字串視為相同。
ignoreSymbols	控制是否忽略符號字元 (例如空格、貨幣符號、數學符號和其他項目)。

下列程式碼顯示將 ignoreDiacritics 屬性設為 true，會變更法文文字清單的排序順序：

```
var words:Array = new  Array("COTE", "coté", "côte", "Coté","cote");
var sorter:Collator = new Collator("fr-CA", CollatorMode.SORTING);
words.sort(sorter.compare);
trace(words); // cote,COTE,côte,coté,Coté

sorter.ignoreDiacritics = true;
words.sort(sorter.compare);
trace(words); // côte,coté,cote,Coté,COTE
```

大小寫轉換

Flash Player 10.1 以及更新的版本，**Adobe AIR 2.0** 以及更新的版本

語言的轉換字母大寫字形式 (大寫) 和小寫字形式 (小寫) 規則也不相同。

例如，在大部分使用拉丁字母的語言中，大寫字 「I」的小寫字形式是 「i」。不過，在部分語言 (例如土耳其文和阿澤里文) 中，還有另一個沒有點的字母 "ı"。因此，在那些語言中，小寫沒有點的 「ı」會轉換為大寫 「I」。小寫 「i」則轉換為有點的大寫 「İ」。

StringTools 類別提供方法，以使用語言特有規則來執行這類轉換。

使用 StringTools 類別

StringTools 類別提供兩種方法來執行大小寫轉換：toLowerCase() 和 toUpperCase()。您可以呼叫具有地區設定 ID 的建構函式，來建立 StringTools 物件。StringTools 類別會從作業系統擷取該地區設定 (或後援地區設定) 的大小寫轉換規則。您無法對大小寫轉換演算法做其他自訂動作。

下列範例使用 toUpperCase() 和 toLowerCase() 方法來轉換含有字母 "ß" (升半音 S) 的德文片語。

```
var phrase:String = "Schloß Neuschwanstein";
var converter:StringTools = new StringTools("de-DE");

var upperPhrase:String = converter.toUpperCase(phrase);
trace(upperPhrase); // SCHLOSS NEUSCHWANSTEIN

var lowerPhrase:String = converter.toLowerCase(upperPhrase);
trace(lowerPhrase);// schloss neuschwanstein
```

toUpperCase() 方法會將小寫字母「ß」轉換為大寫字母「SS」。這項轉換只會單向進行。將字母「SS」轉換回小寫字時，結果會是「ss」，而不是「ß」。

範例：國際化股票行情應用程式

Flash Player 10.1 以及更新的版本，**Adobe AIR 2.0** 以及更新的版本

Global Stock Ticker 應用程式擷取和顯示三個不同股票市場 (美國、日本和歐洲) 的虛構股票資料。其中會依據各個地區設定的慣例，將資料格式化。

這個範例說明 flash.globalization 套件的下列功能：

- 區分地區設定的數字格式
- 區分地區設定的貨幣格式
- 設定貨幣 ISO 代碼和貨幣符號
- 區分地區設定的日期格式
- 擷取和顯示適當的月份名稱縮寫

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。您可以在 Samples/GlobalStockTicker 資料夾中找到 Global Stock Ticker 應用程式檔案。此應用程式是由下列檔案組成：

檔案	說明
GlobalStockTicker.mxml 或 GlobalStockTicker.fla	應用程式使用者介面，在 Flash 中為 FLA，在 Flex 中為 MXML。
styles.css	應用程式使用者介面的樣式 (僅適用於 Flex)。
com/example/programmingas3/stockticker/flex/FinGraph.mxml	顯示模擬股票資料圖表的 MXML 元件 (僅限 Flex)。
com/example/programmingas3/stockticker/flash/GlobalStockTicker.as	含有應用程式之使用者介面邏輯的文件類別 (僅限 Flash)。
comp/example/programmingas3/stockticker/flash/RightAlignedColumn.as	Flash DataGrid 元件 的自訂儲存格輸出器 (僅限 Flash)。

檔案	說明
com/example/programmingas3/stockticker/FinancialGraph.as	繪製模擬股票資料圖表的 ActionScript 類別。
com/example/programmingas3/stockticker/Localizer.as	ActionScript 類別，用以管理地區設定和貨幣，以及處理數字、貨幣金額和日期的當地語系化格式。
com/example/programmingas3/stockticker/StockDataModel.as	ActionScript 類別，用以保留 Global Stock Ticker 範例的所有範例資料。

瞭解使用者介面和範例資料

應用程式的主要使用者介面元素如下：

- 選取地區設定的下拉式清單方塊
- 選取市場的下拉式清單方塊
- DataGrid，顯示每個市場中六家公司的資料
- 圖表，顯示所選取公司之股票的模擬歷程資料

這個應用程式會將它的所有地區設定、市場和公司股票範例資料儲存至 StockDataModel 類別。實際的應用程式會從伺服器擷取資料，然後將它儲存至 StockDataModel 之類的類別。在這個範例中，所有資料都是硬式編碼至 StockDataModel 類別。

備註：財務圖表中顯示的資料不一定符合 DataGrid 控制項中顯示的資料。每次選取不同的公司時，就會隨機重新繪製此圖表。這僅供說明之用。

設定地區設定

在部分初始設定工作之後，應用程式會呼叫 Localizer.setLocale() 方法以建立預設地區設定的格式化程式物件。每次使用者從「地區設定」下拉式清單方塊中選取新值時，也會呼叫 setLocale() 方法。

```
public function setLocale(newLocale:String):void
{
    locale  = new LocaleID(newLocale);

    nf = new NumberFormatter(locale.name);
    traceError(nf.lastOperationStatus, "NumberFormatter", nf.actualLocaleIDName);

    cf = new CurrencyFormatter(locale.name);
    traceError(cf.lastOperationStatus, "CurrencyFormatter", cf.actualLocaleIDName);
    symbolIsSafe = cf.formattingWithCurrencySymbolIsSafe(currentCurrency);
    cf.setCurrency(currentCurrency, currentSymbol);
    cf.fractionalDigits = currentFraction;

    df = new DateTimeFormatter(locale.name, DateTimeStyle.LONG, DateTimeStyle.SHORT);
    traceError(df.lastOperationStatus, "DateTimeFormatter", df.actualLocaleIDName);
    monthNames = df.getMonthNames(DateTimeNameStyle.LONG_ABBREVIATION);
}

public function traceError(status:String, serviceName:String, localeID:String) :void
{
    if(status != LastOperationStatus.NO_ERROR)
    {
        if(status == LastOperationStatus.USING_FALLBACK_WARNING)
```

```
        {
            trace("Warning - Fallback locale ID used by "
                    + serviceName + ": " + localeID);
        }
        else if (status == LastOperationStatus.UNSUPPORTED_ERROR)
        {
            trace("Error in " + serviceName + ": " + status);
            //abort application
            throw(new Error("Fatal error", 0));
        }
        else
        {
            trace("Error in " + serviceName + ": " + status);
        }
    }
    else
    {
        trace(serviceName + " created for locale ID: " + localeID);
    }
}
```

setLocale() 方法會先建立 LocaleID 物件。這個物件可簡化稍後取得實際地區設定之詳細資料的作業 (必要時)。

接下來，它會為地區設定建立新的 NumberFormatter、CurrencyFormatter 和 DateTimeFormatter 物件。它會在建立每個格式化程式物件之後呼叫 traceError() 方法。如果要求的地區設定發生問題，則這個方法會在主控台中顯示錯誤和警告訊息 (實際應用程式應該根據這類錯誤採取動作，而不只是追蹤它們)。

建立 CurrencyFormatter 物件之後，setLocale() 方法會將格式化程式的貨幣 ISO 代碼、貨幣符號和 fractionalDigits 屬性設為先前判定的值 (每次使用者從 Markets 下拉式清單方塊中選取新市場時，都會設定那些值)。

建立 DateTimeFormatter 物件之後，setLocale() 方法也會擷取當地語系化月份名稱縮寫的陣列。

格式化資料

格式化的股票資料是以 DataGrid 控制項呈現。DataGrid 欄都會呼叫 label 函數，以使用適當的格式化程式物件來格式化欄值。

例如，在 Flash 版本中，下列程式碼會設定 DataGrid 欄：

```
var col1:DataGridColumn = new DataGridColumn("ticker");
col1.headerText = "Company";
col1.sortOptions = Array.NUMERIC;
col1.width = 200;

var col2:DataGridColumn = new DataGridColumn("volume");
col2.headerText = "Volume";
col2.width = 120;
col2.cellRenderer = RightAlignedCell;
col2.labelFunction = displayVolume;

var col3:DataGridColumn = new DataGridColumn("price");
col3.headerText = "Price";
col3.width = 70;
col3.cellRenderer = RightAlignedCell;
col3.labelFunction = displayPrice;

var col4:DataGridColumn = new DataGridColumn("change");
col4.headerText = "Change";
col4.width = 120;
col4.cellRenderer = RightAlignedCell;
col4.labelFunction = displayPercent;
```

這個範例的 Flex 版本會以 MXML 宣告它的 DataGrid，也會為每欄定義類似的 label 函數。

labelFunction 屬性會參照下列函數，而它們會呼叫 Localizer 類別的格式方法：

```
private function displayVolume(item:Object):String
{
    return localizer.formatNumber(item.volume, 0);
}

private function displayPercent(item:Object):String
{
    return localizer.formatPercent(item.change )  ;
}

  private function displayPrice(item:Object):String
{
    return localizer.formatCurrency(item.price);
}
```

而 Localizer 方法接著會設定並呼叫適當的格式化程式：

```
public function formatNumber(value:Number, fractionalDigits:int = 2):String
{
    nf.fractionalDigits = fractionalDigits;
    return nf.formatNumber(value);
}

public function formatPercent(value:Number, fractionalDigits:int = 2):String
{
    // HACK WARNING: The position of the percent sign, and whether a space belongs
    // between it and the number, are locale-sensitive decisions. For example,
    // in Turkish the positive format is %12 and the negative format is -%12.
    // Like most operating systems, flash.globalization classes do not currently
    // provide an API for percentage formatting.
    nf.fractionalDigits = fractionalDigits;
    return nf.formatNumber(value) + "%";
}

public function formatCurrency(value:Number):String
{
    return cf.format(value, symbolIsSafe);
}

public function formatDate(dateValue:Date):String
{
    return df.format(dateValue);
}
|
```

第 57 章 當地語系化應用程式

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

所謂當地語系化，是指加入資源以支援多個地區的程序。所謂地區，就是語言和國家 / 地區碼的組合。例如，en_US 表示美國使用的英語，而 fr_FR 則表示法國使用的法語。若要針對這些地區來當地語系化應用程式，您必須提供兩組資源：一組 en_US 地區的資源，以及一組 fr_FR 地區的資源。

多個地區可以共用多種語言。例如，en_US 和 en_GB (英國) 為不同的區域。這兩個地區都使用英語，但是國家 / 地區碼指出它們是不同的區域，因此可能會使用不同的資源。例如，使用 en_US 地區的應用程式對於 「顏色」的拼法是 "color"，而 en_GB 地區則是 "colour"。此外，視地區而定，貨幣單位會以美金或英鎊表示，而且日期與時間的格式也有所不同。

您也可以為某種語言提供一組資源，而不指定國家 / 地區碼。例如，您可以為英語提供 en 資源，並且為 en_US 地區提供其它美式英文專屬的資源。

當地語系化一詞的涵義，不僅是翻譯應用程式中所用到的字串而已。它還包括了所有的資源類型，例如音訊檔案、影像和視訊。

選擇地區

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要判斷您的內容或應用程式所使用的地區設定，您可以使用下列其中一種方法：

- flash.globalization 套件 — 在 flash.globalization 套件中使用區分地區設定的類別，以根據作業系統和使用者偏好設定來擷取使用者的預設地區設定。在 Flash Player 10.1 或更新版本或是 AIR 2.0 或更新版本的執行階段上執行的應用程式，建議使用此方式。如需詳細資訊，請參閱第 813 頁 「判斷地區設定」。
- 使用者提示 — 您可以使用某個預設地區啟動應用程式，然後再詢問使用者，讓他們選擇所需的地區。
- (僅限 AIR) Capabilities.languages — Capabilities.languages 屬性會列出使用者偏好語言 (如同作業系統所設定) 的可用語言陣列。這些字串包含 RFC4646 (http://www.ietf.org/rfc/rfc4646.txt) 所定義的語言標籤 (如果適用的話，還包含指令碼和區域資訊)。這些字串會使用連字符號做為分隔符號 (例如 "en-US" 或 "ja-JP")。所傳回陣列中的第一個項目具有與 language 屬性相同的主要語言 ID。例如，如果 languages[0] 是設定為 "en-US"，language 屬性就會設定為 "en" 不過，如果 language 屬性設定為 "xu" (指定未知的語言)，languages 陣列中的第一個元素就會不同。
- Capabilities.language — Capabilities.language 屬性可提供作業系統的使用者介面語言碼。不過，這個屬性的上限為 20 個已知語言。在英文系統中，這個屬性只會傳回語言碼，而不是國家 / 地區碼。基於這些原因，建議您使用 Capabilities.languages 陣列中的第一個元素。

當地語系化 Flex 內容

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Adobe Flex 包含用於當地語系化 Flex 內容的架構。這個架構包含 Locale、ResourceBundle 和 ResourceManagerImpl 類別，以及 IResourceBundle 和 IResourceManagerImpl 介面。

您可以在 Google Code (http://code.google.com/p/as3localelib/) 上找到 Flex 當地語系化元件庫，內含用來排序應用程式地區設定的公用程式類別。

更多說明主題
http://code.google.com/p/as3localelib/

當地語系化 Flash 內容

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Adobe Flash 專業版將 Locale 類別包含在 ActionScript 3.0 組件中。Locale 類別可以讓您控制 SWF 檔顯示多國語言文字的方式。Flash 的「字串」面板允許您在動態文字欄位中使用字串 ID 來替代字串常值。此功能可讓您建立 SWF 檔，顯示從特定語言 XML 檔載入的文字。如需有關使用 Locale 類別的詳細資訊，請參閱適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。

當地語系化 AIR 應用程式

Adobe AIR 1.0 以及更新的版本

AIR SDK 可提供「HTML 當地語系化架構」(包含在 AIRLocalizer.js 檔案中)。這個架構所含的 API 可協助處理 HTML 類型應用程式的多個地區設定。http://code.google.com/p/as3localelib/ 提供用來排序地區設定的 ActionScript 元件庫。

更多說明主題
http://code.google.com/p/as3localelib/

當地語系化日期、時間和貨幣

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

應用程式顯示日期、時間和貨幣的方式會因為不同地區而有極大的差異。例如，美國的日期顯示標準為月 / 日 / 年，而歐洲的日期顯示標準則為日 / 月 / 年。

您可以撰寫程式碼，設定日期、時間和貨幣的格式。例如，下列程式碼會將 Date 物件轉換成月 / 日 / 年或日 / 月 / 年格式。如果將 locale 變數 (表示地區) 設定為 "en_US"，則函數會傳回月 / 日 / 年格式。這個範例會針對其它地區，將 Date 物件轉換成日 / 月 / 年格式。

```
function convertDate(date)
{
    if (locale == "en_US")
    {
        return (date.getMonth() + 1) + "/" + date.getDate() + "/" + date.getFullYear();
    }
    else
    {
        return date.getDate() + "/" + (date.getMonth() + 1) + "/" + date.getFullYear();
    }
}
```

ADOBE FLEX

Flex 架構包含數個控制項，可用來設定日期、時間和貨幣的格式。這些控制項包括 DateFormatter 和 CurrencyFormatter 控制項。

- mx:DateFormatter
- mx:CurrencyFormatter

第 58 章 關於 HTML 環境

Adobe AIR 1.0 以及更新的版本

Adobe® AIR® 使用 WebKit (www.webkit.org) (亦用於 Safari 網頁瀏覽器) 解析、編排與顯示 HTML 和 JavaScript 內容。在 HTML 內容中使用 AIR API 是選擇性選項。針對 HTMLLoader 物件或 HTML 視窗的內容，您可以徹底使用 HTML 和 JavaScript 進行程式設計。大多數現有的 HTML 網頁和應用程式若是使用相容於 WebKit 的 HTML、CSS、DOM 和 JavaScript 功能，通常只需稍加變更就能順利執行。

重要事項：Adobe AIR 執行階段的新版本可能包含 WebKit 的更新版本。新版 AIR 中的 WebKit 更新 「可能」導致已部署的 AIR 產生無法預期的變化。這些變更可能會影響應用程式中 HTML 內容的行為或外觀。例如，WebKit 顯示的改良與修正，可能造成應用程式使用者介面上的元素編排有所變化。因此，強烈建議您在應用程式中提供更新機制。若您因為 AIR 包含的 WebKit 版本而必須更新應用程式，AIR 更新機制可提示使用者安裝您應用程式的新版本。

下表列出使用 WebKit 版本 (相當於 AIR 中使用的版本) 的 Safari 網頁瀏覽器版本：

AIR 版本	Safari 版本
1.0	2.04
1.1	3.04
1.5	4.0 測試版
2.0	4.03
2.5	4.03
2.6	4.03
2.7	4.03
3	5.0.3

您一律可以檢查 HTMLLoader 物件所傳回的預設使用者代理程式字串，來判斷安裝的 WebKit 版本：

```
var htmlLoader:HTMLLoader = new HTMLLoader();
trace( htmlLoader.userAgent );
```

請記住，AIR 使用的 WebKit 版本與開放原始碼版本不同。AIR 不支援部分功能，而且 AIR 版本可能包括對應的 WebKit 版本尚未納入的安全性和程式錯誤修正。請參閱 第 840 頁 「AIR 不支援的 WebKit 功能」。

由於 AIR 應用程式會直接在桌面上執行，並對檔案系統具有完整存取權限，因此 HTML 內容的安全性模型比一般網頁瀏覽器的安全性模型更為嚴謹。在 AIR 中，只有從應用程式安裝目錄載入的內容會放置於 「應用程式安全執行程序」內，而應用程式安全執行程序具有最高層級權限，允許對 AIR API 進行存取。AIR 會根據其它內容來自於何處，將該內容置入隔離的安全執行程序內。從檔案系統載入的檔案將進入本機安全執行程序。使用 http: 或 https: 通訊協定從網路載入的檔案則依遠端伺服器的網域，則會進入各自的安全執行程序。位於這些非應用程式安全執行程序中的內容均禁止存取任何 AIR API，且執行上與在一般網頁瀏覽器中執行的情況並無兩樣。

AIR 中的 HTML 內容若已套用 Alpha、縮放或透明度設定，便不會顯示 SWF 或 PDF 內容。如需詳細資訊，請參閱第 866 頁 「在 HTML 網頁中載入 SWF 或 PDF 內容時必須考量的事項」及第 770 頁 「視窗透明度」。

更多說明主題

Webkit DOM 參考

Safari HTML 參考

Safari CSS 參考

www.webkit.org

HTML 環境概觀

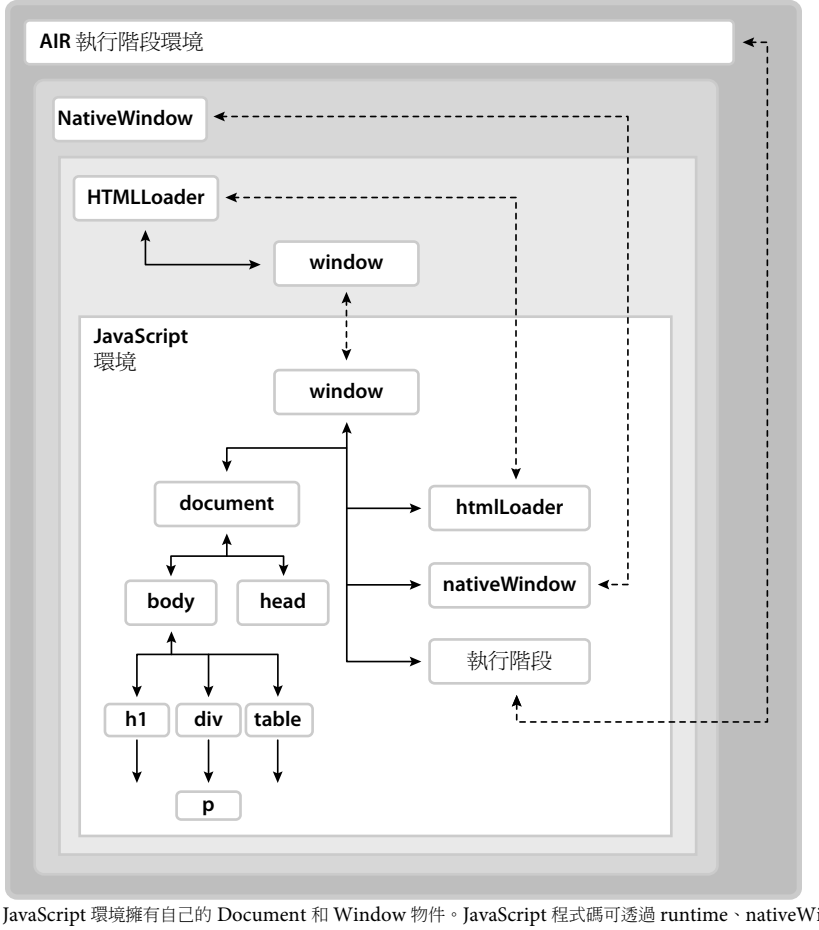
Adobe AIR 1.0 以及更新的版本

Adobe AIR 提供了完全有如瀏覽器般的 JavaScript 環境，附有 HTML 輸出器、文件物件模型和 JavaScript 解譯器。此 JavaScript 環境是以 AIR HTMLLoader 類別表示。在 HTML 視窗中，HTMLLoader 物件包含所有 HTML 內容，而這整個物件則包含在 NativeWindow 物件內。在 SWF 內容中，可將擴充 Sprite 類別的 HTMLLoader 類別加入至舞台的顯示清單，就像任何其它顯示物件一樣。如需 Adobe® ActionScript® 3.0 類別的屬性說明，請參閱第 865 頁「為 AIR HTML 容器編寫指令碼」，以及適用於 Adobe Flash Platform 的 ActionScript 3.0 參考。在 Flex 架構中，AIR HTMLLoader 類別係包覆於 mx:HTML 組件內。mx:HTML 組件則會擴充 UIComponent 類別，因此可直接與其它 Flex 容器搭配使用。除了這點以外，整個 JavaScript 環境在 mx:HTML 組件內的表現並無任何不同。

關於 JavaScript 環境與和 AIR 主機的關係

Adobe AIR 1.0 以及更新的版本

下圖說明了 JavaScript 環境與 AIR 執行階段環境之間的關聯性。雖然圖中只顯示一個原生視窗，AIR 應用程式卻能包含多個視窗 (且單一視窗還能包含多個 HTMLLoader 物件)。



JavaScript 環境擁有自己的 Document 和 Window 物件。JavaScript 程式碼可透過 runtime、nativeWindow 和 htmlLoader 屬性，與 AIR 執行階段環境互動。ActionScript 程式碼則可透過 HTMLLoader 物件的 window 屬性 (此為 JavaScript Window 物件的參考) 與 JavaScript 環境互動。此外，ActionScript 和 JavaScript 物件都能偵聽 AIR 和 JavaScript 物件所傳送的事件。

runtime 屬性可供您存取 AIR API 類別，進而建立新的 AIR 物件以及存取類別 (所謂靜態) 成員。若要存取 AIR API，請在 runtime 屬性後面加上套件名稱和類別名稱。例如，使用下列陳述式便能建立 File 物件：

```
var file = new window.runtime.filesystem.File();
```

備註：AIR SDK 所提供的 JavaScript 檔案 AIRAliases.js 針對最常用的 AIR 類別定義了更方便的別名。匯入該檔案後，您就可以使用簡短格式 air.Class 代替 window.runtime.package.Class。例如，只要使用 new air.File() 便能建立 File 物件。

NativeWindow 物件所提供的屬性可用來控制桌面視窗。在 HTML 網頁中，您可以使用 window.nativeWindow 屬性來存取內含的 NativeWindow 物件。

HTMLLoader 物件所提供的屬性、方法和事件可用來控制內容的載入與顯示方式。在 HTML 網頁中，您可以使用 window.htmlLoader 屬性來存取父系 HTMLLoader 物件。

重要事項：只有已安裝為應用程式一部分且載入為最上層文件的網頁才具有 htmlLoader、nativeWindow 或 runtime 屬性。文件若載入至 frame 或 iframe 中，就不會附加這些屬性 (與父文件位於相同安全執行程序的子文件則可存取父文件的這些屬性。例如，載入至頁框中的文件可以使用 parent.runtime 存取父文件的 runtime 屬性)。

關於安全性

Adobe AIR 1.0 以及更新的版本

AIR 會根據原始網域，在特定的安全執行程序內執行所有程式碼。應用程式內容 (僅限於從應用程式安裝目錄載入的內容) 將放置於 「應用程式」安全執行程序內。只有在此安全執行程序內執行的 HTML 和 JavaScript 能夠存取執行階段環境和 AIR API。同時，當網頁 load 事件的所有處理常式都已傳回後，JavaScript 大部分的動態評估與執行就會遭封鎖於應用程式安全執行程序內。

您可藉由將應用程式頁面載入至 frame 或 iframe 中，並且設定頁框的 AIR 專屬 sandboxRoot 和 documentRoot 特質，將該頁面對應到非應用程式安全執行程序。若將 sandboxRoot 值設定為實際遠端網域，安全執行程序中的內容便能對該網域的內容進行跨指令碼處理。當您要載入遠端內容進行指令碼處理，例如設計 「混合服務型」應用程式時，以此方式對應頁面就很實用。

要讓應用程式和非應用程式的內容能夠彼此進行跨指令碼處理的另一種方式，以及促使非應用程式的內容得以存取 AIR API 的唯一辦法，即是建立 「安全執行程序橋接」。「父系到子系」橋接讓子系 frame、iframe 或視窗的內容得以存取應用程式安全執行程序中定義的選定方法和屬性。相反地，「子系到父系」橋接讓應用程式內容得以存取子系安全執行程序中定義的選定方法和屬性。只要設定 window 物件的 parentSandboxBridge 和 childSandboxBridge 屬性，即可建立安全執行程序橋接。如需詳細資訊，請參閱 第 931 頁 「Adobe AIR 的 HTML 安全性」和 第 837 頁 「HTML 的 frame 和 iframe 元素」。

關於外掛程式和內嵌物件

Adobe AIR 1.0 以及更新的版本

AIR 支援 Adobe® Acrobat® 外掛程式。使用者必須有 Acrobat 或 Adobe® Reader® 8.1 (或更高版本) 才能顯示 PDF 內容。HTMLLoader 物件所提供的屬性可檢查使用者的系統能否顯示 PDF。SWF 檔內容也能在 HTML 環境中顯示，但這是 AIR 內建的功能，而非使用外在的外掛程式。

AIR 不支援其他的 WebKit 外掛程式。

更多說明主題

第 931 頁 「Adobe AIR 的 HTML 安全性」

第 831 頁 「HTML 安全執行程序」

第 837 頁 「HTML 的 frame 和 iframe 元素」

第 835 頁 「JavaScript Window 物件」

第 832 頁 「XMLHttpRequest 物件」

第 471 頁 「在 AIR 中新增 PDF 內容」

AIR 和 WebKit

Adobe AIR 1.0 以及更新的版本

Adobe AIR 使用開放原始碼 WebKit 引擎 (亦用於 Safari 網頁瀏覽器)。AIR 增加許多擴充功能，不但可存取執行階段類別和物件，也提高了安全性。此外，WebKit 本身新增的功能，沒有包含在 HTML、CSS 和 JavaScript 的 W3C 標準中。

其中只包含 AIR 增加部份與最值得注意的 WebKit 擴充功能，如需非標準 HTML、CSS 和 JavaScript 的其他文件，請參閱 www.webkit.org 和 developer.apple.com。如需標準資訊，請參閱 W3C 網站。Mozilla 亦於 HTML、CSS 和 DOM 等主題提供了珍貴的一般參考 (當然，WebKit 和 Mozilla 的引擎是不一樣的)。

AIR 中的 JavaScript

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

AIR 針對 JavaScript 常用物件的典型行為做了多項改變。這些改變多數是為了讓您能更輕鬆地使用 AIR 編寫安全的應用程式。然而，行為上的此等差異也意味著某些常用的 JavaScript 程式碼編寫模式未必能如預期般在 AIR 中執行，包括採用這些模式的現有網路應用程式在內。如需如何修正這類問題的詳細資訊，請參閱第 847 頁 「避免發生與安全性有關的 JavaScript 錯誤」。

HTML 安全執行程序

Adobe AIR 1.0 以及更新的版本

AIR 會根據內容來自於何處，將該內容置入隔離的安全執行程序內。其安全執行程序規則與大多數網頁瀏覽器實作的同源原則以及 Adobe Flash Player 實作安全執行程序的規則相一致。此外，AIR 更提供新的 「應用程式」安全執行程序類型，以容納及保護應用程式內容。請參閱第 901 頁 「安全執行程序」以取得開發 AIR 應用程式時可能面臨的安全執行程序類型。

只有在應用程式安全執行程序內執行的 HTML 和 JavaScript 能夠存取執行階段環境和 AIR API。不過，基於安全考量，JavaScript 多數形式的動態評估與執行也將侷限於應用程式安全執行程序內。即使應用程式確實直接從伺服器載入資訊，這些限制依然存在 (就算是檔案內容、貼上的字串和使用者直接輸入的文字也未必可信任)。

網頁內容的來源決定了其所處的安全執行程序。唯有從應用程式目錄 (由 app: URL 配置所參考的安裝目錄) 載入的內容會放置於應用程式安全執行程序內。從檔案系統載入的內容會位於 「具有檔案系統的本機」或 「本機信任」的安全執行程序，其允許對本機檔案系統內容進行存取與互動，但無法對遠端內容進行這些動作。從網路載入的內容將依其來源網域，放入對應的遠端安全執行程序。

若要讓應用程式頁面能夠任意與遠端安全執行程序中的內容互動，則可將該頁面對應到與遠端內容相同的網域。例如，假設您所編寫的應用程式要顯示網際網路服務提供的地圖資料，就可以將負責載入及顯示服務傳送內容的應用程式頁面對應到該服務網域。將頁面對應到遠端安全執行程序及網域時所用的特質，屬於 HTML frame 和 iframe 元素新增的特質。

為了促使非應用程式安全執行程序中的內容得以安全地使用 AIR 功能，您可以設定父系安全執行程序橋接。若要讓應用程式內容能夠安全地呼叫其它安全執行程序中內容的方法及存取其屬性，則可設定子系安全執行程序橋接。此處所謂 「安全地」是指不能讓遠端內容意外取得未明確公開的物件、屬性或方法的參考。只有簡單資料類型、函數和匿名物件能夠經此橋接進行傳遞。不過，您仍需避免明確公開具有潛在危險的函數。例如，若因公開某介面而讓遠端內容得以在使用者系統上任何地方讀寫檔案，遠端內容可能就會對使用者造成極大損害。

JavaScript eval() 函數

Adobe AIR 1.0 以及更新的版本

一旦頁面載入完成，eval() 函數的使用即侷限於應用程式安全執行程序內。某些用法將獲得許可，以便能夠安然剖析 JSON 格式化資料，但所有產生可執行陳述式的評估都會導致錯誤。第 933 頁「不同安全執行程序中內容的程式碼限制」說明了許可的 eval() 函數用法。

函數建構函式

Adobe AIR 1.0 以及更新的版本

在應用程式安全執行程序中，於頁面載入完成前皆可使用函數建構函式。一旦所有頁面的 load 事件處理常式都已完成，即無法再建立新的函數。

載入外部指令碼

Adobe AIR 1.0 以及更新的版本

位於應用程式安全執行程序中的 HTML 網頁無法使用 script 標籤載入應用程式目錄外的 JavaScript 檔案。應用程式頁面若要載入應用程式目錄外的指令碼，該頁面必須對應到非應用程式安全執行程序。

XMLHttpRequest 物件

Adobe AIR 1.0 以及更新的版本

AIR 所提供的 XMLHttpRequest (XHR) 物件可讓應用程式用來發出資料要求。下列範例會說明簡單的資料要求方式：

```
xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", "http:/www.example.com/file.data", true);
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4) {
        //do something with data...
    }
}
xmlhttp.send(null);
```

與瀏覽器相較之下，AIR 允許執行於應用程式安全執行程序中的內容向任何網域要求資料。包含 JSON 字串的 XHR 結果可評估為資料物件，除非該結果同時包含可執行程式碼。若有任何可執行陳述式存在 XHR 結果中，便會擲回錯誤並造成評估嘗試失敗。

為防止意外插入來自遠端來源的程式碼，若於頁面載入完成前要求資料，同步 XHR 將傳回空的結果。非同步 XHR 一律會於頁面載入後才傳回。

根據預設，AIR 會封鎖非應用程式安全執行程序中的跨網域 XMLHttpRequest。位於應用程式安全執行程序中的父視窗可選擇將其所含 frame 或 iframe 元素的 allowCrossDomainXHR 特質 (此乃 AIR 所新增) 設定為 true，以允許包含非應用程式安全執行程序內容的子頁框進行跨網域要求：

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    allowCrossDomainXHR="true"
</iframe>
```

備註：為了方便起見，也可以使用 AIR URLStream 類別來下載資料。

如果您從含有應用程式內容的 frame 或 iframe 傳送 XMLHttpRequest 到遠端伺服器，而該內容已對應到遠端安全執行程序，則請確認對應的 URL 並未掩蓋 XHR 中所使用的伺服器位址。例如，在下列 iframe 定義中，應用程式內容會對應到 example.com 網域的遠端安全執行程序：

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    documentRoot="app:/sandbox/"
    sandboxRoot="http://www.example.com/"
    allowCrossDomainXHR="true"
</iframe>
```

由於 sandboxRoot 特質會重新對應 www.example.com 位址的根目錄 URL，所有要求將從應用程式目錄載入，而非從遠端伺服器載入。每當進行頁面瀏覽或 XMLHttpRequest 而發出要求時，就會重新對應要求。

為避免意外封鎖對遠端伺服器的資料要求，請將 sandboxRoot 對應到遠端 URL 的子目錄，切勿對應到根目錄。即使該目錄不存在亦無妨。例如，若將上述 iframe 改為如下所示，對 www.example.com 發出要求時便會從遠端伺服器載入，而非從應用程式目錄載入：

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    documentRoot="app:/sandbox/"
    sandboxRoot="http://www.example.com/air/"
    allowCrossDomainXHR="true"
</iframe>
```

在此情況下，本機只會載入位於 air 子目錄中的內容。

如需關於安全執行程序對應的詳細資訊，請參閱 第 837 頁「HTML 的 frame 和 iframe 元素」和 第 931 頁「Adobe AIR 的 HTML 安全性」。

Cookie

Adobe AIR 1.0 以及更新的版本

在 AIR 應用程式中，只有位於遠端安全執行程序的內容 (載入來源為 http: 及 https: 的內容) 能夠使用 Cookie (即 document.cookie 屬性)。在應用程式安全執行程序中，還有其他方式可以儲存永續性資料，包括 EncryptedLocalStore、SharedObject 和 FileStream 類別。

Clipboard 物件

Adobe AIR 1.0 以及更新的版本

WebKit Clipboard API 是由下列事件所驅動：copy、cut 和 paste。傳入這些事件的 event 物件提供了 clipboardData 屬性，可以用來存取剪貼簿。使用 clipboardData 物件的下列方法，即可讀取或寫入剪貼簿資料：

方法	說明
clearData(mimeType)	清除剪貼簿資料。請將 mimeType 參數設定為所要清除資料的 MIME 類型。
getData(mimeType)	取得剪貼簿資料。此方法只能在 paste 事件的處理常式中呼叫。請將 mimeType 參數設定為所要傳回資料的 MIME 類型。
setData(mimeType, data)	複製資料到剪貼簿。請將 mimeType 參數設定為該資料的 MIME 類型。

位於應用程式安全執行程序外的 JavaScript 程式碼只能透過這些事件來存取剪貼簿。然而，位於應用程式安全執行程序中的內容卻可使用 AIR Clipboard 類別直接存取系統剪貼簿。例如，使用下列陳述式即可取得文字格式的剪貼簿資料：

```
var clipping = air.Clipboard.generalClipboard.getData("text/plain",
                                air.ClipboardTransferMode.ORIGINAL_ONLY);
```

有效的資料 MIME 類型如下：

MIME 類型	值
文字	"text/plain"
HTML	"text/html"
URL	"text/uri-list"
點陣圖	"image/x-vnd.adobe.air.bitmap"
檔案清單	"application/x-vnd.adobe.air.file-list"

重要事項：只有位於應用程式安全執行程序中的內容能夠存取剪貼簿上的檔案資料。非應用程式的內容若嘗試存取剪貼簿上的檔案物件，便會擲回安全性錯誤。

如需有關使用剪貼簿的詳細資訊，請參閱第 511 頁「複製和貼上」和「使用 JavaScript 貼上板 (Apple Developer Center)」。

拖放

Adobe AIR 1.0 以及更新的版本

若有拖放手勢進出 HTML，便會產生下列 DOM 事件：dragstart、drag、dragend、dragenter、dragover、dragleave 和 drop。傳入這些事件的 event 物件提供了 dataTransfer 屬性，可以用來存取拖曳的資料。dataTransfer 屬性所參考的物件提供各種方法，名稱如同與剪貼簿事件相關聯的 clipboardData 物件方法。例如，您可以使用下列函數，透過 drop 事件取得文字格式資料：

```
function onDrop(dragEvent){
    return dragEvent.dataTransfer.getData("text/plain",
            air.ClipboardTransferMode.ORIGINAL_ONLY);
}
```

dataTransfer 物件具有下列重要成員：

成員	說明
clearData(mimeType)	清除資料。請將 mimeType 參數設定為所要清除資料形式的 MIME 類型。
getData(mimeType)	取得拖曳的資料。此方法只能在 drop 事件的處理常式中呼叫。請將 mimeType 參數設定為所要取得資料的 MIME 類型。
setData(mimeType, data)	設定所要拖曳的資料。請將 mimeType 參數設定為該資料的 MIME 類型。
types	字串陣列，內含 dataTransfer 物件中目前所有可用資料形式的 MIME 類型。
effectsAllowed	指定是否允許複製、移動或連結拖曳的資料，包括這三種動作的任意組合。請在 dragstart 事件的處理常式中設定 effectsAllowed 屬性。
dropEffect	指定拖曳目標支援哪些允許的放下效果。請在 dragEnter 事件的處理常式中設定 dropEffect 屬性。在拖曳期間，游標會變更形狀以指出當使用者放開滑鼠時將發生何種效果。若未指定 dropEffect，將選用其中一種 effectsAllowed 屬性效果。優先順序為：複製效果 > 移動效果 > 連結效果。使用者可以利用鍵盤修改預設的優先順序。

如需有關 AIR 應用程式如何加入拖放支援的詳細資訊，請參閱第 521 頁「在 AIR 中拖放」和「使用 JavaScript 拖放功能 (Apple Developer Center)」。

innerHTML 與 outerHTML 屬性

Adobe AIR 1.0 以及更新的版本

AIR 針對執行於應用程式安全執行程序中的內容，在 innerHTML 和 outerHTML 屬性的使用上設有安全性限制。於頁面 load 事件發生之前，以及執行任何 load 事件處理常式期間，innerHTML 和 outerHTML 屬性在使用上不會受到限制。不過，縱使頁面已載入，也只能使用 innerHTML 或 outerHTML 屬性為文件加入靜態內容。指定給 innerHTML 或 outerHTML 的字串陳述式若評估為可執行程式碼，均一律予以忽略。例如，如果元素定義中含有事件回呼特質，便會予以忽略而不加入其事件偵聽程式。同樣地，內嵌 <script> 標籤也不會進行評估。如需詳細資訊，請參閱 第 931 頁 「Adobe AIR 的 HTML 安全性」。

Document.write() 與 Document.writeln() 方法

Adobe AIR 1.0 以及更新的版本

於頁面 load 事件發生之前，並非僅限應用程式安全執行程序內才能使用 write() 和 writeln() 方法。不過，一旦頁面已載入，呼叫這兩者之任一方法將不會清空頁面或建立新頁面。而在非應用程式安全執行程序中，若於頁面載入完成後呼叫 document.write() 或 writeln()，便會如同大多數網頁瀏覽器般，清空目前頁面或是開啟新的空白頁面。

Document.designMode 屬性

Adobe AIR 1.0 以及更新的版本

將 document.designMode 屬性設定為 on 值，即可讓文件中所有的元素成為可編輯狀態。內建編輯器支援包括文字編輯、複製、貼上和拖放。將 designMode 設定為 on 相當於將 body 元素的 contentEditable 屬性設定為 true。大部分的 HTML 元素皆可使用 contentEditable 屬性，因此您能藉此定義文件中的可編輯區段。如需詳細資訊，請參閱：第 839 頁 「HTML contentEditable 特質」。

unload 事件 (針對 body 和 frameset 物件)

Adobe AIR 1.0 以及更新的版本

在任何視窗 (包括應用程式主視窗) 的最上層 frameset 或 body 標籤中，請勿使用 unload 事件來回應視窗 (或應用程式) 關閉動作。反之，您應使用 NativeApplication 物件的 exiting 事件 (以偵測應用程式何時關閉)，或者使用 NativeWindow 物件的 closing 事件 (以偵測視窗何時關閉)。例如，下列 JavaScript 程式碼將在使用者關閉應用程式時顯示 「Goodbye.」訊息：

```
var app = air.NativeApplication.nativeApplication;
app.addEventListener(air.Event.EXITING, closeHandler);
function closeHandler(event)
{
    alert("Goodbye.");
}
```

不過，Script 「仍能」成功回應因瀏覽 frame、iframe 或最上層視窗內容而造成的 unload 事件。

備註：未來的 Adobe AIR 版本可能會去除這些限制。

JavaScript Window 物件

Adobe AIR 1.0 以及更新的版本

Window 物件仍是 JavaScript 執行環境中的全域物件。在應用程式安全執行程序中，AIR 為 JavaScript Window 物件增加了一些新屬性，讓您能夠存取 AIR 內建類別以及重要的主控物件。此外，某些方法和屬性會依其是否位於應用程式安全執行程序內而有不同的行為。

Window.runtime 屬性 runtime 屬性讓您能在應用程式安全執行程序內，將內建執行階段類別實體化加以使用。這些類別包含了 AIR 和 Flash Player API (但未提供 Flex 架構等)。例如，下列陳述式會建立 AIR file 物件：

```
var preferencesFile = new window.runtime.flash.filesystem.File();
```

AIR SDK 隨附的 AIRAliases.js 檔案含有各種別名定義，讓您能以簡短格式改寫這類參考。例如，將 AIRAliases.js 匯入頁面後，即可使用下列陳述式建立 File 物件：

```
var preferencesFile = new air.File();
```

window.runtime 屬性僅會針對應用程式安全執行程序中的內容定義，而且只適用於附有 frame 或 iframe 之頁面的父文件。

請參閱第 851 頁 「使用 AIRAliases.js 檔案」。

Window.nativeWindow 屬性 nativeWindow 屬性提供基礎原生視窗物件的參考。利用這個屬性，您可以編寫指令碼來操控視窗功能和屬性如螢幕位置、大小與可見性，以及處理視窗事件如關閉、調整大小和移動等。例如，下列陳述式會關閉視窗：

```
window.nativeWindow.close();
```

備註：NativeWindow 物件所提供的視窗控制功能與 JavaScript Window 物件提供的功能相互重疊。在這兩者之間，您可以選用自己認為最方便的方法。

window.nativeWindow 屬性僅針對應用程式安全執行程序中的內容定義，而且只適用於附有 frame 或 iframe 之頁面的父文件。

Window.htmlLoader 屬性 htmlLoader 屬性提供包含 HTML 內容之 AIR HTMLLoader 物件的參考。利用這個屬性，您可以編寫指令碼來操控 HTML 環境的外觀和行為。例如，您可以使用 htmlLoader.paintsDefaultBackground 屬性決定控制項是否繪製預設的白色背景：

```
window.htmlLoader.paintsDefaultBackground = false;
```

備註：HTMLLoader 物件本身具有 window 屬性，以參考其所含 HTML 內容的 JavaScript Window 物件。利用這個屬性，便能透過 HTMLLoader 容納物件的參考，來存取 JavaScript 環境。

window.htmlLoader 屬性僅會針對應用程式安全執行程序中的內容定義，而且只適用於附有 frame 或 iframe 之頁面的父文件。

Window.parentSandboxBridge 與 Window.childSandboxBridge 屬性 parentSandboxBridge 和 childSandboxBridge 屬性可讓您定義父頁框與子頁框之間的介面。如需詳細資訊，請參閱第 861 頁 「跨指令碼處理不同安全執行程序中內容」。

Window.setTimeout() 與 Window.setInterval() 函數 AIR 針對 setTimeout() 和 setInterval() 函數在應用程式安全執行程序中的使用設有安全性限制。在 setTimeout() 或 setInterval() 呼叫中，即將執行的程式碼不得定義成字串。您必須使用函數參考。如需詳細資訊，請參閱：第 849 頁 「setTimeout() 和 setInterval()」。

Window.open() 函數 如果 open() 方法是由非應用程式安全執行程序中執行的程式碼所呼叫，則只有基於使用者互動 (如按下滑鼠或鍵盤) 而呼叫的情況才會開啟視窗。此外，應用程式標題將附在視窗標題前面 (以防遠端內容開啟的視窗假冒應用程式開啟的視窗)。如需詳細資訊，請參閱第 936 頁 「呼叫 JavaScript window.open() 方法的限制」。

air.NativeApplication 物件

Adobe AIR 1.0 以及更新的版本

NativeApplication 物件會提供應用程式狀態的相關資訊、傳送數種重要的應用程式層級事件，並且提供可控制應用程式行為的實用函數。NativeApplication 物件會自動建立單一實體，透過此類別定義的 NativeApplication.nativeApplication 屬性便能存取該實體。

若要從 JavaScript 程式碼中存取此物件，您可以使用：

```
var app = window.runtime.flash.desktop.NativeApplication.nativeApplication;
```

或者，如果您已匯入 AIRAliases.js 指令碼，也可以使用下列簡短格式：

```
var app = air.NativeApplication.nativeApplication;
```

NativeApplication 物件只能從應用程式安全執行程序內進行存取。如需 NativeApplication 物件的詳細資訊，請參閱第 764 頁 「使用 AIR 執行階段和作業系統資訊」。

JavaScript URL 配置

Adobe AIR 1.0 以及更新的版本

在應用程式安全執行程序內，定義於 JavaScript URL 配置中的程式碼 (例如 href="javascript:alert('Test')") 將遭封鎖而無法執行，但是不會擲回錯誤。

AIR 中的 HTML

Adobe AIR 1.0 以及更新的版本

AIR 和 WebKit 定義一些非標準的 HTML 元素及特質，包括：

第 837 頁 「HTML 的 frame 和 iframe 元素」

第 839 頁 「HTML 元素事件處理常式」

HTML 的 frame 和 iframe 元素

Adobe AIR 1.0 以及更新的版本

AIR 針對應用程式安全執行程序中內容的 frame 和 iframe 元素增加了若干新的特質：

sandboxRoot 特質 sandboxRoot 特質可為 frame src 特質所指定的檔案指定替代的非應用程式來源網域。檔案將會載入到與指定網域相對應的非應用程式安全執行程序中。檔案中的內容和載入自指定網域的內容能夠彼此進行跨指令碼處理。

重要事項：如果您將 sandboxRoot 值設定為網域的基底 URL，所有向該網域發出的內容要求將從應用程式目錄載入，而不是從遠端伺服器載入 (不論要求是因進行頁面瀏覽、XMLHttpRequest 或以任何其它方式載入內容所引起)。

documentRoot 特質 documentRoot 特質指定將從哪個本機目錄載入 URL，以供解析成 sandboxRoot 所指定位置內的檔案。

解析 URL (指定於 frame 的 src 特質或是載入至頁框的內容中) 時，URL 中與 sandboxRoot 指定值相符的部分將以 documentRoot 指定的值取代。因此，如下列頁框標籤所示：

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"/>
```

child.html 將會從應用程式安裝資料夾的 sandbox 子目錄載入，而 child.html 中的相對 URL 則解析成以 sandbox 目錄為基底。請注意，頁框內無法存取遠端伺服器 www.example.com/air 上的任何檔案，因為 AIR 會嘗試從 app:/sandbox/ 目錄載入這類檔案。

allowCrossDomainXHR 特質 在頁框開頭標籤中加入 allowCrossDomainXHR="allowCrossDomainXHR" 可讓頁框中的內容向任何遠端網域發出 XMLHttpRequest。根據預設，非應用程式的內容只能向其本身的來源網域發出這類要求。允許跨網域 XHR 將遭致嚴重的安全隱憂。網頁內的程式碼能與任何網域交換資料。一旦網頁某處插入了惡意內容，目前安全執行程序中的程式碼所能存取的任何資料就可能遭到竄改。請僅在確實有必要跨網域載入資料時，只針對您所建立與掌控的網頁啟用跨網域 XHR。再者，亦應仔細驗證頁面所載入的一切外部資料，以防程式碼插入攻擊或其它形式的攻擊。

重要事項：frame 或 iframe 元素一旦加入 allowCrossDomainXHR 特質，跨網域 XHR 便會啟用 (除非指定值為 "0" 或以字母 "f" 或 "n" 開頭)。例如，將 allowCrossDomainXHR 設定為 "deny" 仍會啟用跨網域 XHR。如果您不想啟用跨網域要求，請勿在元素宣告中加入此特質。

ondominitialize 特質 為頁框的 dominitialize 事件指定事件處理常式。此事件是 AIR 專屬的事件，將於建立頁框的視窗及文件物件後、尚未剖析任何指令碼或建立文件元素之前觸發。

頁框會在載入過程中及早傳送 dominitialize 事件，讓子頁面中的任何指令碼能夠參考由 dominitialize 處理常式加到子文件的物件、變數和函數。父頁面必須與子文件位於相同的安全執行程序，才能直接在子文件中加入或存取任何物件。不過，應用程式安全執行程序中的父系可藉由建立安全執行程序橋接，與非應用程式安全執行程序中的內容進行通訊。

下列範例會說明 iframe 標籤在 AIR 中的用法：

將 child.html 放入遠端安全執行程序，且不要對應到遠端伺服器上的實際網域：

```
<iframe src="http://localhost/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://localhost/air/"/>
```

將 child.html 放入遠端安全執行程序，只允許向 www.example.com 發出 XMLHttpRequest：

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"/>
```

將 child.html 放入遠端安全執行程序，允許向任何遠端網域發出 XMLHttpRequest：

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"
            allowCrossDomainXHR="allowCrossDomainXHR"/>
```

將 child.html 放入本機與檔案系統安全執行程序：

```
<iframe     src="file:///templates/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="app-storage:/templates/"/>
```

將 child.html 放入遠端安全執行程序，使用 dominitialize 事件建立安全執行程序橋接：

```
<html>
<head>
<script>
var bridgeInterface = {};
bridgeInterface.testProperty = "Bridge engaged";
function engageBridge(){
    document.getElementById("sandbox").parentSandboxBridge = bridgeInterface;
}
</script>
</head>
<body>
<iframe id="sandbox"
            src="http://www.example.com/air/child.html"
            documentRoot="app:/"
            sandboxRoot="http://www.example.com/air/"
            ondominitialize="engageBridge()"/>
</body>
</html>
```

下列 child.html 文件會示範子系內容如何存取父輩安全執行程序橋接：

```
<html>
    <head>
        <script>
            document.write(window.parentSandboxBridge.testProperty);
        </script>
    </head>
    <body></body>
</html>
```

如需詳細資訊，請參閱第 861 頁「跨指令碼處理不同安全執行程序中內容」和 第 931 頁「Adobe AIR 的 HTML 安全性」。

HTML 元素事件處理常式

Adobe AIR 1.0 以及更新的版本

AIR 和 WebKit 物件中的 DOM 會傳送未曾見諸於標準 DOM 事件模型中的一些事件。下表會列出相關的事件特質，可供您用來指定這些事件的處理常式：

回呼特質名稱	說明
oncontextmenu	叫用快顯選單 (例如：以滑鼠右鍵按一下或 Command+ 按一下選取的文字) 時呼叫。
oncopy	複製元素內的選取範圍時呼叫。
oncut	剪下元素內的選取範圍時呼叫。
ondominitialize	建立 frame 或 iframe 所載入文件的 DOM 後、尚未建立任何 DOM 元素或剖析指令碼之前呼叫。
ondrag	拖曳元素時呼叫。
ondragend	拖曳動作放開時呼叫。
ondragenter	拖曳手勢進入元素邊界時呼叫。
ondragleave	拖曳手勢移出元素邊界時呼叫。
ondragover	拖曳手勢位於元素邊界內部時持續呼叫。
ondragstart	拖曳手勢開始時呼叫。
ondrop	拖曳手勢在元素上方放開時呼叫。
onerror	載入元素而發生錯誤時呼叫。
oninput	文字輸入表單元素時呼叫。
onpaste	將項目貼入元素時呼叫。
onscroll	可捲動的元素捲動內容時呼叫。
onselectstart	開始進行選取時呼叫。

HTML contentEditable 特質

Adobe AIR 1.0 以及更新的版本

您可為任何 HTML 元素加上 contentEditable 特質，讓使用者能夠編輯該元素的內容。例如，下列 HTML 範例程式碼會將整份文件設定為可編輯，只有第一個 p 元素無法編輯：

```
<html>
<head/>
<body contentEditable="true">
    <h1>de Finibus Bonorum et Malorum</h1>
    <p contentEditable="false">Sed ut perspiciatis unde omnis iste natus error.</p>
    <p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis.</p>
</body>
</html>
```

備註：當 document.designMode 屬性設定為 on 時，文件中的所有元素都將成為可編輯狀態，無論個別元素的 contentEditable 設定為何都一樣。不過，將 designMode 設定為 off 並不會導致 contentEditable 值為 true 的元素無法編輯。如需詳細資訊，請參閱：第 835 頁「Document.designMode 屬性」。

Data: URL

Adobe AIR 2 以及更新的版本

AIR 支援下列元素的 data: URL：

- img
- input type="image"
- CSS 規則允許的影像 (例如背景影像)

Data URL 可讓您將二進位影像資料直接插入至 CSS 或 HTML 文件，做為 base64 編碼字串。下列範例使用 data: URL 做為重複背景：

```
<html>
<head>
<style>
body {
background-
image:url('
ZSBJbWFnZVJlYWR5ccllPAAAAAZQTFRF%2F6cA%2F%2F%2F%2Fgxp3lwAAAAJ0Uk5T%2FwDltzBKAAABF0lEQVR42uzZQQ7CMAxE0e%2F
7X5oNCyRocWzPiJbMBZ6qpIljE%2BnwklgKG7kwUjc2IkIaxkY0CPdEsCCasws6ShXBgmBBmEagpXQQLAgWBAuSY2gaKaWPYEGwIEwg0F
RmECwIFoQeQjJlhJWUEFazjFDJCkI5WYRWMgjtfEGYyQnCXD4jTCdm1zmngFpBFznwVNi5RPSbwbWnpYr%2BBHi%2FtCTfgPLEPL7jBct
AKBRptXJ8M%2BprIuZKu%2BUKcg4YK1PLz7kx4bSqHyPaT4d%2B28OCJJiRBo4FCQsSA0bziT3XubMgYUG6fc5fatmGBQkL0hoJ1IaZMi
QsSFiQ8vRscTjlQOI2iHZwtpHuf%2BJAYiOiJSkj8Z%2FIQ4ABANvXGLd3%2BZMrAAAAAElFTkSuQmCC');
background-repeat:repeat;
}
</style>
</head>
<body>
</body>
</html>
```

使用 data: URL 時，請注意額外空白十分重要。例如，資料字串必須輸入為單一未中斷行。否則，斷行符號會視為資料的一部分，而且影像會無法解碼。

AIR 中的 CSS

Adobe AIR 1.0 以及更新的版本

WebKit 支援數種 CSS 擴充屬性。上述許多擴充功能都使用前置詞：-webkit。請注意，上述部分擴充功能在本質上均屬實驗性質，而且可能會從未來的 WebKit 版本中予以移除。如需有關 Webkit 的 CSS 支援及其 CSS 擴充功能的詳細資訊，請參閱 Safari CSS 參考。

AIR 不支援的 WebKit 功能

Adobe AIR 1.0 以及更新的版本

AIR 不支援 WebKit 或 Safari 4 中的下列功能：

- 透過 window.postMessage 傳送跨網域訊息 (AIR 提供它自己的跨網域通訊 API)
- CSS 變數
- Web 開放字型格式 (WOFF) 與 SVG 字型。
- HTML 視訊和音訊標籤
- 媒體裝置查詢

- 離線應用程式快取
- 列印 (AIR 提供自己的 PrintJob API)
- 拼字和文法檢查程式
- SVG
- WAI-ARIA
- WebSockets (AIR 提供自己的通訊端 API)
- Web 工作者
- WebKit SQL API (AIR 提供它自己的 API)
- WebKit 地理位置 API (AIR 在支援的裝置上提供它自己的地理位置 API)
- WebKit 多檔案上傳 API
- WebKit 觸控事件 (AIR 提供它自己的觸控事件)
- 無線標記語言 (WML)

下列清單包含 AIR 不支援的特定 JavaScript API、HTML 元素，以及 CSS 屬性和值：

不支援的 **JavaScript Window** 物件成員：

- applicationCache()
- console
- openDatabase()
- postMessage()
- document.print()

不支援的 **HTML** 標籤：

- audio
- video

不支援的 **HTML** 特質：

- aria-*
- draggable
- formnovalidate
- list
- novalidate
- onbeforeload
- onhashchange
- onorientationchange
- onpagehide
- onpageshow
- onpopstate
- ontouchstart
- ontouchmove

- ontouchend
- ontouchcancel
- onwebkitbeginfullscreen
- onwebkitendfullscreen
- pattern
- required
- sandbox

不支援的 **JavaScript** 事件：

- beforeload
- hashchange
- orientationchange
- pagehide
- pageshow
- popstate
- touchstart
- touchmove
- touchend
- touchcancel
- webkitbeginfullscreen
- webkitendfullscreen

不支援的 **CSS** 屬性：

- background-clip
- background-origin (使用 -webkit-background-origin)
- background-repeat-x
- background-repeat-y
- background-size (使用 -webkit-background-size)
- border-bottom-left-radius
- border-bottom-right-radius
- border-radius
- border-top-left-radius
- border-top-right-radius
- text-rendering
- -webkit-animation-play-state
- -webkit-background-clip
- -webkit-color-correction
- -webkit-font-smoothing

不支援的 **CSS** 值：

- appearance 屬性值：
  - media-volume-slider-container
  - media-volume-slider
  - media-volume-sliderthumb
  - outer-spin-button
- border-box (background-clip 和 background-origin)
- contain (background-size)
- content-box (background-clip 和 background-origin)
- cover (background-size)
- list 屬性值：
  - afar
  - amharic
  - amharic-abegede
  - cjk-earthly-branch
  - cjk-heavenly-stem
  - ethiopic
  - ethiopic-abegede
  - ethiopic-abegede-am-et
  - ethiopic-abegede-gez
  - ethiopic-abegede-ti-er
  - ethiopic-abegede-ti-et
  - ethiopic-halehame-aa-er
  - ethiopic-halehame-aa-et
  - ethiopic-halehame-am-et
  - ethiopic-halehame-gez
  - ethiopic-halehame-om-et
  - ethiopic-halehame-sid-et
  - ethiopic-halehame-so-et
  - ethiopic-halehame-ti-er
  - ethiopic-halehame-ti-et
  - ethiopic-halehame-tig
  - hangul
  - hangul-consonant
  - lower-norwegian
  - oromo
  - sidama

  - somali
  - tigre
  - tigrinya-er
  - tigrinya-er-abegede
  - tigrinya-et
  - tigrinya-et-abegede
  - upper-greek
  - upper-norwegian
- -wap-marquee (display 屬性)

第 59 章 在 AIR 中撰寫 HTML 和 JavaScript 程式

Adobe AIR 1.0 以及更新的版本

使用 HTML 和 JavaScript 開發 Adobe® AIR® 應用程式時，可以參考許多特有的程式設計主題。無論您是要設計 HTML 類型的 AIR 應用程式，或是設計使用 HTMLLoader 類別 (或 mx:HTML Flex™ 組件) 來執行 HTML 和 JavaScript 之 SWF 類型的 AIR 應用程式，都請特別留意下列資訊。

關於 HTMLLoader 類別

Adobe AIR 1.0 以及更新的版本

Adobe AIR 的 HTMLLoader 類別可定義顯示物件，在 AIR 應用程式中顯示 HTML 內容。SWF 類型的應用程式可以將 HTMLLoader 控制項加入至現有視窗，或者使用 HTMLLoader.createRootWindow() 方法，建立會自動包含 HTMLLoader 物件的 HTML 視窗。從載入的 HTML 網頁中，透過 JavaScript window.htmlLoader 屬性即可存取 HTMLLoader 物件。

從 URL 載入 HTML 內容

Adobe AIR 1.0 以及更新的版本

下列程式碼會將一個 URL 載入 HTMLLoader 物件中 (將 HTMLLoader 新增為舞台的子內容，或新增為其它顯示物件容器，以顯示應用程式的 HTML 內容)：

```
import flash.html.HTMLLoader;

var html:HTMLLoader = new HTMLLoader;
html.width = 400;
html.height = 600;
var urlReq:URLRequest = new URLRequest("http://www.adobe.com/");
html.load(urlReq);
```

根據預設，HTMLLoader 物件的 width 和 height 屬性都設定為 0。將 HTMLLoader 物件加入舞台時，您就必須設定這兩個屬性的值。載入網頁時，HTMLLoader 會傳送數個事件。您可以使用這些事件，判斷與載入網頁的互動是否安全。這些事件都會在第 880 頁 「在 AIR 中處理 HTML 相關事件」中加以說明。

備註：在 Flex 架構中，只有會擴充 UIComponent 類別的類別才能加入做為 Flex Container 組件的子系。如此一來，您就不能直接將 HTMLLoader 加入做為 Flex Container 組件的子系，但是可以使用 Flex mx:HTML 控制項。您可以建置自訂類別，讓這個類別擴充 UIComponent 並包含 HTMLLoader 做為 UIComponent 的子系，也可以加入 HTMLLoader 做為 UIComponent 的子系，並將此 UIComponent 加入 Flex 容器中。

您也可以使用 TextField 類別來呈現 HTML 文字，但是功能會受到限制。The Adobe® Flash® Player 的 TextField 類別支援 HTML 標記的子集，但是基於大小限制，功能也會受到限制 (包含在 Adobe AIR 中的 HTMLLoader 類別無法在 Flash Player 中使用)。

從字串載入 HTML 內容

Adobe AIR 1.0 以及更新的版本

HTMLLoader 物件的 loadString() 方法會將 HTML 內容的字串載入 HTMLLoader 物件中：

```
var html:HTMLLoader = new HTMLLoader();
var htmlStr:String = "<html><body>Hello <b>world</b>.</body></html>";
html.loadString(htmlStr);
```

根據預設，透過 loadString() 方法所載入的內容會放在一個具有下列特性的非應用程式安全執行程序中：

- 具有透過網路載入內容的權限 (非透過檔案系統)。
- 它無法使用 XMLHttpRequest 來載入資料。
- window.location 屬性設為 "about:blank"。
- 其內容無法存取 window.runtime 屬性 (如同其它非應用程式安全執行程序的內容存取一般)。

在 AIR 1.5 中，HTMLLoader 類別包含一個 placeLoadStringContentInApplicationSandbox 屬性。當 HTMLLoader 物件的這個屬性設為 true 時，透過 loadString() 方法所載入的內容就會放在應用程式安全執行程序中 (預設值為 false)。如此，經由 loadString() 方法所載入的內容就可以存取 window.runtime 屬性和所有的 AIR API。如果將這個屬性設定為 true，請確定您在呼叫 loadString() 方法時所用的字串是來自一個受信任的資料來源。當這個屬性設為 true 時，HTML 字串中的程式碼陳述式就會以完整的應用程式權限執行。請只有確定字串中未包含有害的程式碼時，才將這個屬性設為 true。

在以 AIR 1.0 或 AIR 1.1 SDK 編譯的應用程式中，透過 loadString() 方法所載入的內容會放在應用程式安全執行程序中。

在 AIR 應用程式中使用 HTML 時必須注意的安全性規則

Adobe AIR 1.0 以及更新的版本

與 AIR 應用程式一併安裝的檔案都可以存取 AIR API。但是，基於安全性理由，來自其它來源的內容則無法存取 AIR API。舉例來說，這項限制可以避免遠端網域 (例如 http://example.com) 中的內容讀取使用者之桌面目錄中的內容 (或更糟的情況)。

由於有些安全性漏洞，讓惡意使用者可以透過呼叫 eval() 函數 (和相關的 API) 進行攻擊，因此與應用程式一併安裝的內容預設無法使用這些方法。不過，某些 Ajax 架構會呼叫 eval() 函數和相關 API。

因此，若要在 AIR 應用程式中建立正確運作的內容架構，您必須針對來自不同來源的內容，審慎考量規則來制定安全性限制。來自不同來源的內容都會置於不同的安全性分類中，這些分類稱為安全執行程序 (請參閱第 901 頁 「安全執行程序」)。根據預設，與應用程式一併安裝的內容會安裝在名為 「應用程式」安全執行程序的安全執行程序中，這個安全執行程序會將 AIR API 的存取權限授與此內容。應用程式安全執行程序通常是最安全的安全執行程序，其各項限制都是設計用來避免執行不受信任的程式碼。

執行階段可讓您將與應用程式一併安裝的內容載入至應用程式安全執行程序以外的安全執行程序。非應用程式安全執行程序中之內容會在安全性環境中運作，這個環境與典型網頁瀏覽器的安全性環境類似。例如，非應用程式安全執行程序中的程式碼可以使用 eval() 和相關的方法 (但同時也不允許程式碼存取 AIR API)。執行階段包含各種方式，讓不同安全執行程序中的內容在溝通時確保其安全性 (且不會將 AIR API 公開給非應用程式的內容)。如需詳細資訊，請參閱第 861 頁「跨指令碼處理不同安全執行程序中內容」。

如果您呼叫因為安全性因素而禁止在安全執行程序中使用的程式碼，執行階段便會傳送下列 JavaScript 錯誤：「應用程式安全執行程序中的 JavaScript 程式碼造成 Adobe AIR 執行階段安全性違規」。

若要避免發生這個錯誤，請遵循下節第 847 頁 「避免發生與安全性有關的 JavaScript 錯誤」所描述的程式碼做法。

如需詳細資訊，請參閱 第 931 頁 「Adobe AIR 的 HTML 安全性」。

避免發生與安全性有關的 JavaScript 錯誤

Adobe AIR 1.0 以及更新的版本

如果您呼叫因為這些安全性限制而禁止在安全執行程序中使用的程式碼，執行階段便會傳送下列 JavaScript 錯誤：「應用程式安全執行程序中的 JavaScript 程式碼造成 Adobe AIR 執行階段安全性違規」。若要避免發生這個錯誤，請遵循下列程式碼做法。

與安全性有關的 JavaScript 錯誤發生原因

Adobe AIR 1.0 以及更新的版本

一旦引發文件的 load 事件，而且有任何 load 事件處理常式結束時，大部分涉及評估和執行字串的作業就會受到限制，無法在應用程式安全執行程序中執行程式碼。嘗試使用下列類型的 JavaScript 陳述式來評估和執行可能不安全的字串時，就會產生 JavaScript 錯誤：

- eval() 函數
- setTimeout() 和 setInterval()
- Function 建構函式

此外，使用下列類型的 JavaScript 陳述式也會失敗，並且不會產生不安全的 JavaScript 錯誤：

- javascript: URL
- 在 innerHTML 和 outerHTML 陳述式中透過事件之特質指定的事件回呼
- 載入位於應用程式安裝目錄以外的 JavaScript 檔案
- document.write() 和 document.writeln()
- 在 load 事件傳送之前或在 load 事件處理常式期間初始化的同步 XMLHttpRequest
- 動態建立的指令碼元素

備註：在某些受到限制的情況下，可以執行評估字串作業。如需詳細資訊，請參閱第 933 頁「不同安全執行程序中內容的程式碼限制」。

Adobe 已整理出一份支援應用程式安全執行程序的 Ajax 架構清單，請造訪 http://www.adobe.com/go/airappsandboxframeworks_tw 以取得這份清單。

下列幾節將說明如何重新撰寫指令碼來避免這些不安全的 JavaScript 錯誤和不會產生訊息的錯誤，以便在應用程式安全執行程序中執行程式碼。

將應用程式內容對應至不同的安全執行程序

Adobe AIR 1.0 以及更新的版本

在大部分情況下，您都可以重新撰寫應用程式或重新建立其架構，避免發生與安全性有關的 JavaScript 錯誤。不過，在無法重新撰寫或重新建立架構的情況下，您可以使用第 862 頁「將應用程式內容載入至非應用程式安全執行程序」中所述的技巧，將應用程式內容載入至不同的安全執行程序中。如果該項內容也必須存取 AIR API，您可以如第 862 頁「設定安全執行程序橋接介面」中所述，建立新的安全執行程序橋接。

eval() 函數

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在應用程式安全執行程序中，eval() 函數只能在網頁的 load 事件傳送之前，以及在 load 事件處理常式執行期間使用。網頁完成載入之後，呼叫 eval() 將不會執行程式碼。不過，在下列情況中，您可以重新撰寫程式碼以避免使用 eval()。

將屬性指定給物件

Adobe AIR 1.0 以及更新的版本

請不要透過下列所示方式剖析字串來建置屬性存取子：

```
eval("obj." + propName + " = " + val);
```

請改以方括號標記法來存取屬性：

```
obj[propName] = val;
```

使用可以在內容中使用的變數來建立函數

Adobe AIR 1.0 以及更新的版本

請將下列陳述式：

```
function compile(var1, var2){
    eval("var fn = function(){ this."+var1+"(var2) }");
    return fn;
}
```

取代成為：

```
function compile(var1, var2){
    var self = this;
    return function(){ self[var1](var2) };
}
```

使用類別名稱做為字串參數來建立物件

Adobe AIR 1.0 以及更新的版本

假設使用下列程式碼定義假設性的 JavaScript 類別：

```
var CustomClass =
    {
        Utils:
        {
            Parser: function(){ alert('constructor') }
        },
        Data:
        {

        }
    };
var constructorClassName = "CustomClass.Utils.Parser";
```

建立實體的最簡單方式，就是使用 eval()：

```
var myObj;
eval('myObj=new ' + constructorClassName +'()')
```

不過，您可以剖析該類別名稱的其中每一個字元，並使用方括號標記法建置新物件，以避免呼叫 eval()：

```
function getter(str)
{
    var obj = window;
    var names = str.split('.');
    for(var i=0;i<names.length;i++){
        if(typeof obj[names[i]]=='undefined'){
            var undefstring = names[0];
            for(var j=1;j<=i;j++)
                undefstring+="."+names[j];
            throw new Error(undefstring+" is undefined");
        }
        obj = obj[names[i]];
    }
    return obj;
}
```

若要建立實體，請使用：

```
try{
    var Parser = getter(constructorClassName);
    var a = new Parser();
    }catch(e){
        alert(e);
}
```

setTimeout() 和 setInterval()

Adobe AIR 1.0 以及更新的版本

請以函數參考或物件取代傳遞做為處理常式函數的字串。例如，將下列陳述式：

```
setTimeout("alert('Timeout')", 100);
```

取代成為：

```
setTimeout(function(){alert('Timeout')}, 100);
```

或者，當這個函數需要由呼叫者來設定 this 物件時，將下列陳述式：

```
this.appTimer = setInterval("obj.customFunction();", 100);
```

取代成為：

```
var _self = this;
this.appTimer = setInterval(function(){obj.customFunction.apply(_self);}, 100);
```

Function 建構函式

Adobe AIR 1.0 以及更新的版本

對 new Function(param, body) 的呼叫可以取代為行內函數宣告，或者只在處理 load 事件之前使用這個呼叫。

javascript: URL

Adobe AIR 1.0 以及更新的版本

在應用程式安全執行程序中，使用 javascript: URL 配置在連結中定義的程式碼會遭忽略，這不會產生不安全的 JavaScript 錯誤。您可以將如下所示使用 javascript: URL 的連結：

```
<a href="javascript:code()">Click Me</a>
```

取代成為：

```
<a href="#" onclick="code()">Click Me</a>
```

在 innerHTML 和 outerHTML 陳述式中透過事件之特質指定的事件回呼

Adobe AIR 1.0 以及更新的版本

當您使用 innerHTML 或 outerHTML 將元素加入至文件的 DOM 時，所有在陳述式中指定的事件回呼 (例如 onclick 或 onmouseover) 都會遭忽略，這不會產生安全性錯誤。您可以改為將 id 特質指定給這些新元素，並使用 addEventListener() 方法設定這些事件處理常式回呼函數。

舉例來說，您可以在文件中指定某個目標元素，如下所示：

```
<div id="container"></div>
```

將下列陳述式：

```
document.getElementById('container').innerHTML =
    '<a href="#" onclick="code()">Click Me.</a>';
```

取代成為：

```
document.getElementById('container').innerHTML = '<a href="#" id="smith">Click Me.</a>';
document.getElementById('smith').addEventListener("click", function() { code(); });
```

載入位於應用程式安裝目錄以外的 JavaScript 檔案

Adobe AIR 1.0 以及更新的版本

您不可以載入位於應用程式安全執行程序以外的指令碼檔案，這不會產生安全性錯誤。在應用程式安全執行程序中執行的指令碼檔案都必須安裝在應用程式目錄內。若要在網頁中使用外部指令碼，您必須將該網頁對應至不同的安全執行程序。請參閱第 862 頁「將應用程式內容載入至非應用程式安全執行程序」。

document.write() 和 document.writeln()

Adobe AIR 1.0 以及更新的版本

在 load 事件經過處理之後，對 document.write() 或 document.writeln() 的呼叫會遭忽略，這不會產生安全性錯誤。此外，您也可以載入新的檔案，或使用各種 DOM 操作技巧來取代文件主體。

在 load 事件傳送之前或在 load 事件處理常式期間初始化的同步 XMLHttpRequest

Adobe AIR 1.0 以及更新的版本

在網頁的 load 事件傳送之前或在 load 事件處理常式執行期間初始化的同步 XMLHttpRequest 不會傳回任何內容。非同步 XMLHttpRequest 可以初始化，但是只有在傳送 load 事件之後才會傳回內容。在 load 事件經過處理之後，同步 XMLHttpRequest 便會正常運作。

動態建立的指令碼元素

Adobe AIR 1.0 以及更新的版本

動態建立 (例如建立時搭配 innerHTML 或 document.createElement() 方法) 的指令碼元素都會遭忽略。

透過 JavaScript 存取 AIR API 類別

Adobe AIR 1.0 以及更新的版本

除了 Webkit 的標準和擴充元素以外，HTML 和 JavaScript 程式碼也可以存取執行階段提供的主控類別。這些類別可以讓您存取 AIR 提供的各種進階功能，其中包括：

- 存取檔案系統
- 使用本機 SQL 資料庫
- 控制應用程式和視窗選單
- 存取通訊端以進行網路連線
- 使用使用者定義的類別和物件
- 聲音功能

例如，AIR 檔案 API 包括 File 類別，這個類別包含在 flash.filesystem 套件中。因此，您可以在 JavaScript 中建立 File 物件，如下所示：

```
var myFile = new window.runtime.flash.filesystem.File();
```

runtime 物件是特殊的 JavaScript 物件，在應用程式安全執行程序之 AIR 中執行的 HTML 內容可以使用這個物件。這個物件可以讓您透過 JavaScript 存取執行階段類別。runtime 物件的 flash 屬性可以提供 flash 套件的存取權限，而 runtime 物件的 flash.filesystem 屬性可以提供 flash.filesystem 套件 (而且這個套件包含 File 類別) 的存取權限。所謂套件，是指將 ActionScript 中用到的類別組織起來的方式。

備註：runtime 屬性不會自動加入至在 frame 或 iframe 中載入之網頁的視窗物件。不過，只要子文件位於應用程式安全執行程序中，這個子系就可以存取父輩的 runtime 屬性。

由於執行階段類別的套件結構需要開發人員輸入很長的 JavaScript 程式碼字串才能存取每個類別 (例如 window.runtime.flash.desktop.NativeApplication)，而 AIR SDK 包含的 AIRAliases.js 檔案可以讓您更輕易地存取執行階段類別 (舉例來說，只要輸入 air.NativeApplication 即可)。

本指南著重於討論 AIR API 類別，對於 HTML 開發人員可能感興趣之 Flash Player API 的其它類別，則可以在 Adobe AIR API Reference for HTML Developers 中找到相關說明。ActionScript 是在 SWF (Flash Player) 內容中使用的語言。不過，JavaScript 和 ActionScript 的語法很類似 (兩者都是以 ECMAScript 語言版本為基礎)。在 JavaScript (在 HTML 內容中) 和 ActionScript (在 SWF 內容中) 中都可以使用內建類別。

備註：JavaScript 程式碼無法使用 Dictionary、XML 和 XMLList 類別，而這些類別都可以在 ActionScript 中使用。

使用 AIRAliases.js 檔案

Adobe AIR 1.0 以及更新的版本

執行階段類別會組織在套件結構中，如下所示：

- window.runtime.flash.desktop.NativeApplication
- window.runtime.flash.desktop.ClipboardManager

- window.runtime.flash.filesystem.FileStream
- window.runtime.flash.data.SQLDatabase

包含在 AIR SDK 中的 AIRAliases.js 檔案提供了「別名」定義，可以讓您存取執行階段類別且不需要輸入太長的字串。例如，您只要輸入下列字串，就可以存取上述這些類別：

- air.NativeApplication
- air.Clipboard
- air.FileStream
- air.SQLDatabase

上述清單只是 AIRAliases.js 檔案中這些類別的簡短子集。如需完整的類別清單和套件層級函數，請參閱 Adobe AIR API Reference for HTML Developers。

除了常用的執行階段類別以外，AIRAliases.js 檔案還包含下列常用套件層級函數的別名：window.runtime.trace()、window.runtime.flash.net.navigateToURL() 和 window.runtime.flash.net.sendToURL()。與上述函數相對應的別名分別為：air.trace()、air.navigateToURL() 和 air.sendToURL()。

若要使用 AIRAliases.js 檔案，請將下列 script 參考納入 HTML 網頁中：

```
<script src="AIRAliases.js"></script>
```

請視需要調整 src 參考中的路徑。

重要事項：除了有註明的地方以外，本說明文件中的 JavaScript 範例程式碼會假設您已經將 AIRAliases.js 檔案納入 HTML 網頁中。

關於 AIR 中的 URL

Adobe AIR 1.0 以及更新的版本

在 AIR 內執行的 HTML 內容中，您可以使用下表中的任何 URL 配置。使用的位置包含：在用於定義 img、frame、iframe 和 script 標籤的 src 特質中；在 link 標籤的 href 特質中；或者其它您可以提供 URL 的地方。

URL 配置	說明	範例
file	相對於檔案系統根項目的路徑。	`file:///c:/AIR Test/test.txt`
app	相對於所安裝應用程式根目錄的路徑。	`app:/images`
app-storage	相對於應用程式儲存目錄的路徑。針對每一個安裝的應用程式，AIR 都會定義唯一的應用程式儲存目錄，可以用來儲存某個應用程式專屬的資料。	`app-storage:/settings/prefs.xml`
http	標準 HTTP 要求。	`http://www.adobe.com`
https	標準 HTTPS 要求。	`https://secure.example.com`

如需有關在 AIR 中使用 URL 配置的詳細資訊，請參閱第 702 頁「URI 配置」。

許多 AIR API (包括 File、Loader、URLStream 和 Sound 類別) 都會使用 URLRequest 物件，而非使用包含該 URL 的字串。URLRequest 物件本身是使用字串來初始化，因此可以使用任何相同的 URL 配置。例如，下列陳述式會建立可以用來向 Adobe 首頁發出要求的 URLRequest 物件：

```
var urlReq = new air.URLRequest("http://www.adobe.com/");
```

如需 URLRequest 物件的詳細資訊，請參閱：第 700 頁「HTTP 通訊」。

讓 JavaScript 可以使用 ActionScript 物件

Adobe AIR 1.0 以及更新的版本

利用 HTMLLoader 物件所載入之 HTML 網頁的 window.runtime、window.htmlLoader 和 window.nativeWindow 屬性，此 HTML 網頁中的 JavaScript 便可以呼叫 ActionScript 執行內容中定義的類別、物件和函數。您也可以在 JavaScript 執行內容中建立 ActionScript 物件和函數的參考，讓 JavaScript 程式碼可以使用這些物件和函數。

基本範例：透過 ActionScript 存取 JavaScript 物件

Adobe AIR 1.0 以及更新的版本

在下列範例中，會示範如何將參考 ActionScript 物件的屬性加入至 HTML 網頁的全域 window 物件：

```
var html:HTMLLoader = new HTMLLoader();
var foo:String = "Hello from container SWF."
function helloFromJS(message:String):void {
    trace("JavaScript says:", message);
}
var urlReq:URLRequest = new URLRequest("test.html");
html.addEventListener(Event.COMPLETE, loaded);
html.load(urlReq);

function loaded(e:Event):void{
    html.window.foo = foo;
    html.window.helloFromJS = helloFromJS;
}
```

在上一個範例中，HTMLLoader 物件中載入的 HTML 內容 (在名為 test.html 的檔案內) 可以存取父輩 SWF 檔中定義的 foo 屬性和 helloFromJS() 方法：

```
<html>
    <script>
        function alertFoo() {
            alert(foo);
        }
    </script>
    <body>
        <button onClick="alertFoo()">
            What is foo?
        </button>
        <p><button onClick="helloFromJS('Hi.')">
            Call helloFromJS() function.
        </button></p>
    </body>
</html>
```

當存取載入端文件的 JavaScript 內容時，您可以使用 htmlDOMInitialize 事件在網頁建構程序初期建立物件，讓網頁中定義的所有指令碼都可以存取這些物件。如果您要等待 complete 事件，那麼網頁中只有在網頁的 load 事件傳送之後執行的指令碼才可以存取這些新增的物件。

讓 JavaScript 可以使用類別定義

Adobe AIR 1.0 以及更新的版本

若要能在 JavaScript 中使用應用程式的 ActionScript 類別，您可以將載入的 HTML 內容指定給包含這些類別定義的應用程式網域。JavaScript 執行內容的應用程式網域可以透過 HTMLLoader 物件的 runtimeApplicationDomain 屬性來設定。例如，若要將應用程式網域設定為主要應用程式網域，請將 runtimeApplicationDomain 設定為 ApplicationDomain.currentDomain，如下列程式碼所示：

```
html.runtimeApplicationDomain = ApplicationDomain.currentDomain;
```

一旦設定 runtimeApplicationDomain 屬性，JavaScript 內容便會與所指定的網域共用類別定義。若要在 JavaScript 中建立自訂類別的實體，請透過 window.runtime 屬性參考類別定義，並且使用 new 運算子：

```
var customClassObject = new window.runtime.CustomClass();
```

此 HTML 內容必須是來自於相容安全性網域的內容。如果 HTML 內容的來源安全性網域與您指定的應用程式網域不同，網頁便會改用預設的應用程式網域。舉例來說，如果您是從網際網路載入遠端網頁，那麼就無法指定該網頁的應用程式網域做為 ApplicationDomain.currentDomain。

移除事件偵聽程式

Adobe AIR 1.0 以及更新的版本

當您將 JavaScript 事件偵聽程式加入至目前網頁以外的物件 (包括執行階段物件、所載入 SWF 內容中的物件，甚至是在其它網頁中執行之 JavaScript 物件) 時，一定要在取消載入網頁時移除這些事件偵聽程式。否則，這些事件偵聽程式就會將事件傳送至已不存在的處理常式函數。發生這個情況時，您將看到下列錯誤訊息：「應用程式嘗試參考已不存在之 HTML 網頁中的 JavaScript 物件」。移除不再需要的事件偵聽程式時，也會讓 AIR 將相關的記憶體回收。如需詳細資訊，請參閱第 884 頁「移除瀏覽之 HTML 網頁中的事件偵聽程式」。

透過 ActionScript 存取 HTML DOM 和 JavaScript 物件

Adobe AIR 1.0 以及更新的版本

當 HTMLLoader 物件傳送 complete 事件時，您便可以存取網頁之 HTML DOM (文件物件模型) 中的所有物件。可以存取的物件包含顯示元素 (例如網頁中的 div 和 p 物件) 以及 JavaScript 變數和函數。complete 事件對應至 JavaScript 網頁的 load 事件。在傳送 complete 之前，DOM 元素、變數和函數可能尚未經過剖析或尚未建立。請盡可能先等待 complete 事件傳送之後，再存取 HTML DOM。

例如，請考慮下列 HTML 網頁：

```
<html>
    <script>
        foo = 333;
        function test() {
            return "OK.";
        }
    </script>
    <body>
        <p id="p1">Hi.</p>
    </body>
</html>
```

這個簡單的 HTML 網頁會定義名為「foo」的 JavaScript 變數和名為「test()」的 JavaScript 函數。這兩者都是網頁之全域 window 物件的屬性。同時，window.document 物件也包含名為 P 的項目 (ID 為「p1」)，您可以使用 getElementById() 方法來存取這個項目。一旦網頁載入完成 (此時 HTMLLoader 物件會傳送 complete 事件)，您便可以透過 ActionScript 存取這些物件，如下列 ActionScript 程式碼所示：

```
var html:HTMLLoader = new HTMLLoader();
html.width = 300;
html.height = 300;
html.addEventListener(Event.COMPLETE, completeHandler);
var xhtml:XML =
    <html>
        <script>
            foo = 333;
            function test() {
                return "OK.";
            }
        </script>
        <body>
            <p id="p1">Hi.</p>
        </body>
    </html>;
html.loadString(xhtml.toString());

function completeHandler(e:Event):void {
    trace(html.window.foo); // 333
    trace(html.window.document.getElementById("p1").innerHTML); // Hi.
    trace(html.window.test()); // OK.
}
```

若要存取 HTML 元素的內容，請使用 innerHTML 屬性。例如，上一個程式碼範例是使用 html.window.document.getElementById("p1").innerHTML 取得名為「p1」的 HTML 元素。

您也可以透過 ActionScript 設定 HTML 網頁的各項屬性。例如，下列範例會使用做為容器之 HTMLLoader 物件的參考，設定網頁上 p1 元素的內容以及 JavaScript 變數 foo 的值：

```
html.window.document.getElementById("p1").innerHTML = "Goodbye";
html.window.foo = 66;
```

在 HTML 中嵌入 SWF 內容

Adobe AIR 1.0 以及更新的版本

您可以在 AIR 應用程式的 HTML 內容中嵌入 SWF 內容，就像在瀏覽器的 HTML 內容中嵌入 SWF 內容一樣。使用 object 標籤、embed 標籤或兩者，即可嵌入 SWF 內容。

備註：常見的網頁開發作法，是同時使用 object 和 embed 標籤，在 HTML 網頁中顯示 SWF 內容。這種作法在 AIR 中並沒有優點，您可以只在內容中使用符合 W3C 標準的 object 標籤本身，在 AIR 中顯示內容。同時，對於也會在瀏覽器中顯示的 HTML 內容，您也可以繼續使用 object 和 embed 標籤兩者。

如果您已在顯示 HTML 和 SWF 內容的 NativeWindow 物件中啟用透明度，則當用來嵌入內容的視窗模式 (wmode) 值設定為：window 時，AIR 不會顯示 SWF 內容。若要在透明視窗的 HTML 頁面中顯示 SWF 內容，請將 wmode 參數設定為 opaque 或 transparent。window 是 wmode 的預設值，因此如果您沒有指定值，則內容可能無法顯示。

在下列範例中，會示範如何使用 HTML object 標籤，在 HTML 內容中顯示 SWF 檔。wmode 參數已設定為 opaque，即使基礎 NativeWindow 物件為透明，還是可以顯示內容。此 SWF 檔是從應用程式目錄載入，但是您可以使用 AIR 支援的任何 URL 配置 (SWF 檔的載入來源位置會決定 AIR 要在哪個安全執行程序中放置內容)。

```
<object type="application/x-shockwave-flash" width="100%" height="100%">
    <param name="movie" value="app:/SWFFile.swf"></param>
    <param name="wmode" value="opaque"></param>
</object>
```

您也可以使用 Scrip 動態載入內容。下列範例會建立 object 節點，以顯示 urlString 參數中指定的 SWF 檔。在範例中，會加入該節點做為網頁元素的子系，而這個網頁元素的 ID 是由 elementID 參數指定：

```
<script>
function showSWF(urlString, elementID){
    var displayContainer = document.getElementById(elementID);
    var flash = createSWFObject(urlString, 'opaque', 650, 650);
    displayContainer.appendChild(flash);
}
function createSWFObject(urlString, wmodeString, width, height){
    var SWFObject = document.createElement("object");
    SWFObject.setAttribute("type","application/x-shockwave-flash");
    SWFObject.setAttribute("width","100%");
    SWFObject.setAttribute("height","100%");
    var movieParam = document.createElement("param");
    movieParam.setAttribute("name","movie");
    movieParam.setAttribute("value",urlString);
    SWFObject.appendChild(movieParam);
    var wmodeParam = document.createElement("param");
    wmodeParam.setAttribute("name","wmode");
    wmodeParam.setAttribute("value",wmodeString);
    SWFObject.appendChild(wmodeParam);
    return SWFObject;
}
</script>
```

縮放或旋轉 HTMLLoader 物件，或將 alpha 屬性值設為 1.0 以外的值，都無法顯示 SWF 內容。在 AIR 1.5.2 之前的版本中，無論 wmode 的值設定為何，都無法在透明視窗中顯示 SWF 內容。

備註：當內嵌的 SWF 物件嘗試載入如視訊檔的外部資源時，如果 HTML 檔沒有提供視訊檔的絕對路徑，則可能無法適當顯示 SWF 內容。但是，內嵌的 SWF 物件可使用相對路徑載入外部影像檔。

以下範例說明外部資源如何透過內嵌於 HTML 內容的 SWF 物件載入外部資源：

```
var imageLoader;

function showSWF(urlString, elementID){
    var displayContainer = document.getElementById(elementID);
    imageLoader = createSWFObject(urlString,650,650);
    displayContainer.appendChild(imageLoader);
}

function createSWFObject(urlString, width, height){

    var SWFObject = document.createElement("object");
    SWFObject.setAttribute("type","application/x-shockwave-flash");
    SWFObject.setAttribute("width","100%");
    SWFObject.setAttribute("height","100%");

    var movieParam = document.createElement("param");
    movieParam.setAttribute("name","movie");
    movieParam.setAttribute("value",urlString);
    SWFObject.appendChild(movieParam);

    var flashVars = document.createElement("param");
    flashVars.setAttribute("name","FlashVars");

    //Load the asset inside the SWF content.
    flashVars.setAttribute("value","imgPath=air.jpg");
    SWFObject.appendChild(flashVars);

    return SWFObject;
}
function loadImage()
{
  showSWF("ImageLoader.swf", "imageSpot");

}
```

在以下 ActionScript 範例中，會讀取 HTML 檔傳送的影像路徑並在舞台上載入影像：

```
package
{
      import flash.display.Sprite;
      import flash.display.LoaderInfo;
      import flash.display.StageScaleMode;
      import flash.display.StageAlign;
      import flash.display.Loader;
      import flash.net.URLRequest;

      public class ImageLoader extends Sprite
      {
            public function ImageLoader()
            {

                  var flashvars = LoaderInfo(this.loaderInfo).parameters;

                  if(flashvars.imgPath){
                        var imageLoader = new Loader();
                        var image = new URLRequest(flashvars.imgPath);
                        imageLoader.load(image);
                        addChild(imageLoader);
                        imageLoader.x = 0;
                        imageLoader.y = 0;
                        stage.scaleMode=StageScaleMode.NO_SCALE;
                        stage.align=StageAlign.TOP_LEFT;
                  }
            }
      }
}
```

在 HTML 網頁中使用 ActionScript 元件庫

Adobe AIR 1.0 以及更新的版本

AIR 可以擴充 HTML 指令碼元素，如此網頁即可在已編譯的 SWF 檔中匯入 ActionScript 類別。例如，若要匯入名為「myClasses.swf」的元件庫 (位於根應用程式資料夾的 lib 子目錄中)，請在 HTML 檔中納入下列指令碼標籤：

```
<script src="lib/myClasses.swf" type="application/x-shockwave-flash"></script>
```

重要事項：type 特質必須是 type="application/x-shockwave-flash"，這個元件庫才能正確載入。

如果 SWF 內容的編譯結果屬於 Flash Player 10 或 AIR 1.5 SWF，您就必須將應用程式描述器檔案的 XML 命名空間設定為 AIR 1.5 命名空間。

封裝 AIR 檔時，也必須將 lib 目錄和 myClasses.swf 檔案包含其中。

透過 JavaScript Window 物件的 runtime 屬性，存取匯入的類別：

```
var libraryObject = new window.runtime.LibraryClass();
```

如果 SWF 檔中的類別都已經組織為套件，那麼您也必須納入套件名稱。舉例來說，如果 LibraryClass 定義位於名為「utilities」的套件中，您可以使用下列陳述式建立該類別的實體：

```
var libraryObject = new window.runtime.utilities.LibraryClass();
```

備註：若要編譯 ActionScript SWF 元件庫，以便在 AIR 中做為 HTML 網頁的一部分使用，請使用 acompc 編譯器。acompc 公用程式為 Flex SDK 的一部分。如需詳細資訊，請參閱「Flex SDK 說明文件」。

從已匯入的 ActionScript 檔案存取 HTML DOM 和 JavaScript 物件

Adobe AIR 1.0 以及更新的版本

若要在已使用 <script> 標籤匯入至 HTML 網頁的 SWF 檔中，透過 ActionScript 存取網頁內的物件，請將 JavaScript 物件 (例如 window 或 document) 的參考傳遞至 ActionScript 程式碼中定義的函數。您可以在該函數中使用參考來存取 JavaScript 物件 (或其它可以透過傳入的參考來存取的物件)。

例如，請考慮下列 HTML 網頁：

```
<html>
    <script src="ASLibrary.swf" type="application/x-shockwave-flash"></script>
    <script>
        num = 254;
        function getStatus() {
            return "OK.";
        }
        function runASFunction(window){
            var obj = new runtime.ASClass();
            obj.accessDOM(window);
        }
    </script>
    <body onload="runASFunction">
        <p id="p1">Body text.</p>
    </body>
</html>
```

這個簡單的 HTML 網頁具有名為「num」的 JavaScript 變數和名為「getStatus()」的 JavaScript 函數。這兩者都是網頁之 window 物件的屬性。同時，window.document 物件也包含名為 P 的元素 (ID 為「p1」)。

這個網頁會載入 ActionScript 檔案 「ASLibrary.swf」，這個檔案包含 ASClass 類別。ASClass 會定義名為 accessDOM() 的函數，這個函數會追蹤這些 JavaScript 物件的值。accessDOM() 方法會採用 JavaScript Window 物件做為引數。使用這個 Window 參考即可存取網頁中的其它物件 (包括變數、函數和 DOM 元素)，如下列定義所示範：

```
public class ASClass{
    public function accessDOM(window:*):void {
        trace(window.num); // 254
        trace(window.document.getElementById("p1").innerHTML); // Body text..
        trace(window.getStatus()); // OK.
    }
}
```

您可以透過已匯入的 ActionScript 類別，取得及設定 HTML 網頁的屬性。例如，下列函數會設定網頁上 p1 元素的內容，以及設定網頁上 JavaScript 變數 foo 的值。

```
public function modifyDOM(window:*):void {
    window.document.getElementById("p1").innerHTML = "Bye";
    window.foo = 66;
```

轉換 Date 和 RegExp 物件

Adobe AIR 1.0 以及更新的版本

JavaScript 和 ActionScript 語言都會定義 Date 和 RegExp 類別，但是這些類型的物件並不會自動在這兩個執行內容之間轉換。您必須先將 Date 和 RegExp 物件轉換為對等類型，才能在交替執行內容中使用這兩個物件設定屬性或函數參數。

例如，下列 ActionScript 程式碼會將名為 jsDate 的 JavaScript Date 物件轉換為 ActionScript Date 物件：

```
var asDate:Date = new Date(jsDate.getMilliseconds());
```

下列 ActionScript 程式碼會將名為 jsRegExp 的 JavaScript RegExp 物件轉換為 ActionScript RegExp 物件：

```
var flags:String = "";
if (jsRegExp.dotAll) flags += "s";
if (jsRegExp.extended) flags += "x";
if (jsRegExp.global) flags += "g";
if (jsRegExp.ignoreCase) flags += "i";
if (jsRegExp.multiline) flags += "m";
var asRegExp:RegExp = new RegExp(jsRegExp.source, flags);
```

透過 ActionScript 操作 HTML 樣式表

Adobe AIR 1.0 以及更新的版本

一旦 HTMLLoader 物件傳送 complete 事件之後，您便可以檢查和操作網頁中的 CSS 樣式。

例如，請考慮下列簡單的 HTML 文件：

```
<html>
<style>
    .style1A { font-family:Arial; font-size:12px }
    .style1B { font-family:Arial; font-size:24px }
</style>
<style>
    .style2 { font-family:Arial; font-size:12px }
</style>
<body>
    <p class="style1A">
        Style 1A
    </p>
    <p class="style1B">
        Style 1B
    </p>
    <p class="style2">
        Style 2
    </p>
</body>
</html>
```

當 HTMLLoader 物件載入此內容之後，您便可以透過 window.document.styleSheets 陣列的 cssRules 陣列，操作網頁中的 CSS 樣式，如下所示：

```
var html:HTMLLoader = new HTMLLoader( );
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.addEventListener(Event.COMPLETE, completeHandler);
function completeHandler(event:Event):void {
    var styleSheet0:Object = html.window.document.styleSheets[0];
    styleSheet0.cssRules[0].style.fontSize = "32px";
    styleSheet0.cssRules[1].style.color = "#FF0000";
    var styleSheet1:Object = html.window.document.styleSheets[1];
    styleSheet1.cssRules[0].style.color = "blue";
    styleSheet1.cssRules[0].style.font-family = "Monaco";
}
```

上述程式碼會調整 CSS 樣式，讓產生的 HTML 文件外觀看起來如下：

Style 1A

Style 1B

Style 2

請記住，程式碼可以在 HTMLLoader 物件傳送 complete 事件之後，將各種樣式加入至網頁。

跨指令碼處理不同安全執行程序中內容

Adobe AIR 1.0 以及更新的版本

執行階段安全性模型會將程式碼與不同的來源隔離。藉由跨指令碼處理不同安全執行程序中內容，您可以讓某個安全執行程序中的內容存取另一個安全執行程序中選取的屬性和方法。

AIR 安全執行程序和 JavaScript 程式碼

Adobe AIR 1.0 以及更新的版本

AIR 會強制落實相同來源的原則，避免某個網域中的程式碼與另一個網域中的內容互動。所有檔案都會根據其來源，放置在安全執行程序中。一般而言，應用程式安全執行程序中的內容不能違反相同來源原則，也不能針對從應用程式安裝目錄以外載入的內容進行跨指令碼處理。不過，AIR 提供了幾項技巧，可以讓您對非應用程式的內容進行跨指令碼處理。

其中一項技巧是使用 frame 或 iframe，將應用程式內容對應至不同的安全執行程序中。所有自應用程式之安全執行程序區域載入的網頁，其行為就如同這些網頁是自遠端網域載入一樣。例如，將應用程式內容對應至「example.com」網域，此內容即可針對自 example.com 載入的網頁進行跨指令碼處理。

由於這項技巧會將應用程式內容放置在不同的安全執行程序中，所以此內容中的程式碼也將不再受到限制，可以在經過評估的字串中執行程式碼。即使不需要對遠端內容進行跨指令碼處理，您還是可以使用這項安全執行程序對應技巧來減少這些限制。使用許多 JavaScript 架構的其中一種架構，或使用必須依賴評估字串才能執行的現有程式碼時，以這種方式對應內容就特別有用。不過，您必須考量並避免其它風險，也就是當執行應用程式安全執行程序以外的內容時，可能會遭插入或執行不受信任的內容。

同時，對應至其它安全執行程序的應用程式內容便無法存取 AIR API，因此這項安全執行程序對應技巧無法用於將 AIR 功能公開給在應用程式安全執行程序以外執行的程式碼。

另外還有一項跨指令碼處理技巧，可讓您在非應用程式安全執行程序中的內容與應用程式安全執行程序中這個內容的父輩文件之間，建立名為「安全執行程序橋接」的介面。這座橋樑可以讓子系內容存取父輩所定義的屬性和方法、讓父輩存取子系所定義的屬性和方法，或兩者。

最後，您也可以從應用程式安全執行程序和其它安全執行程序 (選擇性) 執行跨網域的 XMLHttpRequest。

如需詳細資訊，請參閱 第 837 頁「HTML 的 frame 和 iframe 元素」、第 931 頁「Adobe AIR 的 HTML 安全性」，以及第 832 頁「XMLHttpRequest 物件」。

將應用程式內容載入至非應用程式安全執行程序

Adobe AIR 1.0 以及更新的版本

若要讓應用程式內容可以安全地針對從應用程式安裝目錄以外載入的內容進行跨指令碼處理，您可以使用 frame 或 iframe 元素，將應用程式內容載入至與該外部內容相同的安全執行程序中。如果您不需要對遠端內容進行跨指令碼處理，但仍希望載入應用程式安全執行程序以外之應用程式的網頁，可以使用 http://localhost/ 或其它不具有風險的值做為來源網域。

AIR 中為 frame 元素加入了新的特質 sandboxRoot 和 documentRoot，可以讓您指定載入至該 frame 的應用程式檔案是否必須對應至非應用程式安全執行程序。剖析為 sandboxRoot URL 下方路徑的檔案會改為從 documentRoot 目錄載入。基於安全性的考量，以這種方式載入的應用程式內容會被視為就像實際是從 sandboxRoot URL 載入一樣。

sandboxRoot 屬性會指定要使用的 URL，以判斷要在哪個安全執行程序或網域中放置 frame 內容。這裡必須使用 file:、http: 或 https: URL 配置。如果您指定相對 URL，內容就會保留在應用程式安全執行程序中。

documentRoot 屬性會指定要從其中載入 frame 內容的來源目錄。這裡必須使用 file:、app: 或 app-storage: URL 配置。

下列範例會對應安裝在應用程式之 sandbox 子目錄中的內容，以便在遠端安全執行程序和 www.example.com 網域中執行：

```
<iframe
    src="http://www.example.com/local/ui.html"
    sandboxRoot="http://www.example.com/local/"
    documentRoot="app:/sandbox/">
</iframe>
```

ui.html 網頁可以使用下列指令碼標籤，從本機 sandbox 資料夾載入 Javascript 檔案：

```
<script src="http://www.example.com/local/ui.js"></script>
```

此外，這個網頁也可以使用如下所示的指令碼標籤，從遠端伺服器中的目錄載入內容：

```
<script src="http://www.example.com/remote/remote.js"></script>
```

sandboxRoot URL 將會遮蔽遠端伺服器中位於相同 URL 的所有內容：在上述範例中，您無法存取位於 www.example.com/local/ (或其中任何子目錄) 中的遠端內容，因為 AIR 會將要求重新對應至本機應用程式目錄。無論這些要求是因為網頁瀏覽、XMLHttpRequest 或其它載入內容的方式而衍生，AIR 都會重新對應這些要求。

設定安全執行程序橋接介面

Adobe AIR 1.0 以及更新的版本

安全執行程序橋接的使用時機如下：當應用程式安全執行程序中的內容必須存取非應用程式安全執行程序中的內容所定義之屬性或方法時，或者當非應用程式的內容必須存取應用程式安全執行程序中的內容所定義之屬性或方法時。使用子系文件之 window 物件的 childSandboxBridge 和 parentSandboxBridge 屬性，即可建立安全執行程序橋接。

建立子系安全執行程序橋接

Adobe AIR 1.0 以及更新的版本

childSandboxBridge 屬性可以讓子系文件將介面公開給父輩文件中的內容。若要公開介面，請將 childSandbox 屬性設定為子系文件中的函數或物件。之後，您便能從父輩文件中的內容存取該物件或函數。下列範例會說明在子系文件中執行的指令碼如何將包含函數和屬性的物件公開給其父輩：

```
var interface = {};
interface.calculatePrice = function(){
    return ".45 cents";
}
interface.storeID = "abc"
window.childSandboxBridge = interface;
```

如果這個子系是載入至 ID 指定為「child」的 iframe 中，您便可以藉由讀取 frame 的 childSandboxBridge 屬性，從父輩內容存取介面：

```
var childInterface = document.getElementById("child").contentWindow.childSandboxBridge;
air.trace(childInterface.calculatePrice()); //traces ".45 cents"
air.trace(childInterface.storeID)); //traces "abc"
```

建立父輩安全執行程序橋接

Adobe AIR 1.0 以及更新的版本

parentSandboxBridge 屬性可以讓父輩文件將介面公開給子系文件中的內容。若要公開介面，父輩文件必須將子系文件的 parentSandbox 屬性設定為父輩文件中定義的函數或物件。之後，您便能子系中的內容存取該物件或函數。在下列範例中，會說明在父輩 frame 中執行的指令碼如何將包含函數和屬性的物件公開給子系文件：

```
var interface = {};
interface.save = function(text){
    var saveFile = air.File("app-storage:/save.txt");
    //write text to file
}
document.getElementById("child").contentWindow.parentSandboxBridge = interface;
```

使用這個介面，子系 frame 中的內容便能將文字儲存至名為 save.txt 的檔案，但是無法存取檔案系統中的其它任何項目。子系內容可以呼叫 save 函數，如下所示：

```
var textToSave = "A string.";
window.parentSandboxBridge.save(textToSave);
```

應用程式內容必須將限制最多的介面公開給其它安全執行程序。非應用程式內容本質上必須視為不值得信任的內容，因為很容易在不小心或惡意情況下，在其中插入程式碼。您必須在適當的位置加上保護措施，避免透過父輩安全執行程序橋接公開的介面遭濫用。

在頁面載入期間存取父輩安全執行程序橋接

Adobe AIR 1.0 以及更新的版本

父輩安全執行程序橋接必須在子系文件中的指令碼執行之前先經過設定，Script 才能存取此安全執行程序橋接。Window、frame 和 iframe 物件都會在新網頁的 DOM 建立之後、尚未剖析任何指令碼或新增 DOM 元素之前傳送 dominitialize 事件。您可以在網頁建構程序初期，使用 dominitialize 事件建立安全執行程序橋接，讓子系文件中的所有指令碼都可以存取此橋接。

在下列範例中，會示範如何建立父輩安全執行程序橋接，以回應從子系 frame 傳送的 dominitialize 事件：

```
<html>
<head>
<script>
var bridgeInterface = {};
bridgeInterface.testProperty = "Bridge engaged";
function engageBridge(){
    document.getElementById("sandbox").contentWindow.parentSandboxBridge = bridgeInterface;
}
</script>
</head>
<body>
<iframe id="sandbox"
            src="http://www.example.com/air/child.html"
            documentRoot="app:/"
            sandboxRoot="http://www.example.com/air/"
            ondominitialize="engageBridge()"/>
</body>
</html>
```

下列 child.html 文件會示範子系內容如何存取父輩安全執行程序橋接：

```
<html>
    <head>
        <script>
            document.write(window.parentSandboxBridge.testProperty);
        </script>
    </head>
    <body></body>
</html>
```

若要針對子系 window (而非 frame) 偵聽 dominitialize 事件，您必須將偵聽程式加入至使用 window.open() 函數建立的新子系 window 物件：

```
var childWindow = window.open();
childWindow.addEventListener("dominitialize", engageBridge());
childWindow.document.location = "http://www.example.com/air/child.html";
```

在此案例中，無法將應用程式內容對應至非應用程式安全執行程序中。這項技巧只有在從應用程式目錄以外載入 child.html 時才有用。您仍然可以將 window 中的應用程式內容對應至非應用程式安全執行程序，但是必須先載入中間網頁。中間網頁本身會使用 frame 載入子系文件，並將文件對應至所需的安全執行程序。

如果您使用 HTMLLoader 類別的 createRootWindow() 函數來建立 window，則此新的 window 就不是文件 (createRootWindow() 的呼叫來源) 的子系。因此，您無法在呼叫端 window 與載入至新 window 的非應用程式內容之間建立安全執行程序橋接，而是必須改為在新的 window 中載入中間網頁，讓網頁本身使用 frame 載入子系文件。之後，您便可以在新 window 的父輩文件與載入至 frame 的子系文件之間建立安全執行程序橋接。

第 60 章 為 AIR HTML 容器編寫指令碼

Adobe AIR 1.0 以及更新的版本

在 Adobe® AIR® 中，HTMLLoader 類別的作用是當做 HTML 內容的容器。這個類別提供了許多衍生自 Sprite 類別的屬性和方法，可控制 ActionScript® 3.0 顯示清單中的物件行為和物件外觀。此外，這個類別還會針對某些工作 (例如載入 HTML 內容及與其互動，以及管理操作記錄) 定義屬性和方法。

HTMLHost 類別會為 HTMLLoader 定義一組預設行為。當您建立 HTMLLoader 物件時，不會提供任何 HTMLHost 實作。因此，當 HTML 內容觸發其中一項預設行為 (例如，變更視窗位置或視窗標題) 時，不會產生任何作用。您可以擴充 HTMLHost 類別，以定義應用程式所需的各項行為。

使用 AIR 建立 HTML 視窗時，會提供 HTMLHost 的預設實作。您可以設定 HTMLLoader 物件的 htmlHost 屬性 (透過將 defaultBehavior 參數設定為 true 的方式以建立新的 HTMLHost 物件的方式)，將預設的 HTMLHost 實作指定給其它 HTMLLoader 物件。

備註：在 Adobe® Flex™ Framework 中，HTMLLoader 物件是由 mx:HTML 組件所包裝。因此，使用 Flex 時，請使用 HTML 組件。

顯示 HTMLLoader 物件的屬性

Adobe AIR 1.0 以及更新的版本

HTMLLoader 物件會繼承 Adobe® Flash® Player Sprite 類別的顯示屬性。舉例來說，您可以調整背景大小、移動背景、隱藏背景，以及變更背景顏色。或者，您也可以套用進階特效，例如濾鏡、遮色片、縮放和旋轉。套用特效時，請考量對易讀性造成的影響。套用某些特效時，載入至 HTML 網頁的 SWF 和 PDF 內容將無法顯示。

HTML 視窗包含可用於呈現 HTML 內容的 HTMLLoader 物件。這個物件受限於視窗區域中，因此變更尺寸、位置、旋轉方式或縮放係數時，不一定都會產生所希望的結果。

基本顯示屬性

Adobe AIR 1.0 以及更新的版本

HTMLLoader 的基本顯示屬性可以讓您將控制項置於其父輩顯示物件中、設定大小，及顯示或隱藏控制項。您不應該變更 HTML 視窗之 HTMLLoader 物件的基本顯示屬性。

這些基本顯示屬性包括：

屬性	備註
x、y	將物件置於其父輩容器中。
width、height	變更顯示區域的尺寸。
visible	控制物件及其包含之內容的可見性。

在 HTML 視窗以外，HTMLLoader 物件的 width 和 height 屬性的預設值為 0。您必須先指定所載入之 HTML 內容的寬度和高度，才能看到其內容。HTML 內容會依照 HTMLLoader 的大小來繪製，並且根據內容中的 HTML 和 CSS 屬性來配置。變更 HTMLLoader 的大小時，內容會經過重新排列。

將內容載入至新的 HTMLLoader 物件 (其中 width 仍然設定為 0) 時，許多人往往會想要使用 contentWidth 和 contentHeight 屬性設定 HTMLLoader 的顯示 width 和 height。這項技巧適用於在根據 HTML 和 CSS 排列規則進行配置時，具有合理最小寬度的網頁。不過，在 HTMLLoader 沒有提供合理寬度的情況下，某些網頁的內容排列會呈現窄長的配置。

備註：當您變更 HTMLLoader 物件的寬度和高度時，scaleX 和 scaleY 值不會變更，就像其它大多數類型的顯示物件一樣。

HTMLLoader 內容的透明度

Adobe AIR 1.0 以及更新的版本

HTMLLoader 物件的 paintsDefaultBackground 屬性 (預設為 true) 可決定 HTMLLoader 物件是否繪製不透明背景。當 paintsDefaultBackground 為 false 時，背景會是透明的。在 HTML 內容之前景元素的後方，可以看見 HTMLLoader 物件下方的顯示物件容器或它顯示物件。

如果 HTML 文件的 body 元素或其它任何元素指定背景顏色 (例如，使用 style="background-color:gray")，則 HTML 的該部分便不透明，並且呈現為指定的背景顏色。如果您設定 HTMLLoader 物件的 opaqueBackground 屬性，而且 paintsDefaultBackground 為 false，則針對 opaqueBackground 所設定的顏色便為可見。

備註：您可以使用透明的 PNG 格式圖像，為 HTML 文件中的元素提供經過 Alpha 混合後的背景。不支援設定 HTML 元素的不透明樣式。

縮放 HTMLLoader 內容

Adobe AIR 1.0 以及更新的版本

請避免使用 1.0 以上的縮放係數來縮放 HTMLLoader 物件。如果放大 HTMLLoader 物件，HTMLLoader 內容會以特定解析度呈現，並呈現像素化。重新調整視窗時，若要避免縮放 HTMLLoader 及其內容，請將 Stage 的 scaleMode 屬性設定為 StageScaleMode.NO_SCALE。

在 HTML 網頁中載入 SWF 或 PDF 內容時必須考量的事項

Adobe AIR 1.0 以及更新的版本

在下列情況中，載入至 HTMLLoader 物件中的 SWF 和 PDF 內容會消失：

- 使用係數為 1.0 以外的值縮放 HTMLLoader 物件。
- 將 HTMLLoader 物件的 alpha 屬性設定為 1.0 以外的值。
- 旋轉 HTMLLoader 內容。

如果您移除違反規定的屬性設定，並移除使用中的濾鏡，內容就會重新顯示。

此外，執行階段無法在透明視窗顯示 PDF 內容。當 object 或 embed 標籤的 wmode 參數設定為 opaque 或 transparent 時，執行階段只顯示內嵌於 HTML 網頁的 SWF 內容。因為 wmode 預設值為 window，所以 SWF 內容不會顯示在透明視窗中，除非您明確設定 wmode 參數。

備註：AIR 1.5.2 之前的版本，無論使用哪一個 wmode 值，都無法顯示內嵌於 HTML 的 SWF。

如需有關在 HTMLLoader 中載入這些媒體類型的詳細資訊，請參閱第 855 頁「在 HTML 中嵌入 SWF 內容」和第 471 頁「在 AIR 中新增 PDF 內容」。

進階顯示屬性

Adobe AIR 1.0 以及更新的版本

HTMLLoader 類別會繼承可供特殊特效使用的數個方法。一般而言，當搭配 HTMLLoader 顯示使用時，這些特效具有幾項限制。但是針對轉場或其它暫時性特性而言，這些特效就很有用。例如，如果您顯示對話方塊以取得使用者輸入，那麼直到使用者關閉對話方塊為止，您都可以讓主要視窗呈現模糊的狀態。同樣地，關閉視窗時，可以用淡出的方式關閉。

進階顯示屬性包括：

屬性	限制
alpha	可能會降低 HTML 內容的可讀性。
filters	在 HTML Window 中，外部特效會遭視窗邊緣裁剪。
graphics	使用圖像命令繪製的形狀 (包含預設背景) 會在 HTML 內容下方顯示。paintsDefaultBackground 屬性必須設定為 false，才能看見所繪製的形狀。
opaqueBackground	不會變更預設背景的顏色。paintsDefaultBackground 屬性必須設定為 false，才能看見這個顏色圖層。
rotation	矩形 HTMLLoader 區域的角落可能會遭視窗邊緣裁剪。在 HTML 內容中載入的 SWF 和 PDF 內容將不會顯示。
scaleX、scaleY	使用大於 1 的縮放係數時，顯示內容可能呈現像素化。在 HTML 內容中載入的 SWF 和 PDF 內容將不會顯示。
transform	可能會降低 HTML 內容的易讀性。HTML 顯示可能會遭視窗邊緣裁剪。如果轉換涉及旋轉、縮放或傾斜，在 HTML 內容中載入的 SWF 和 PDF 內容便不會顯示。

下列範例會示範如何設定 filters 陣列以模糊化整個 HTML 顯示：

```
 var html:HTMLLoader = new HTMLLoader();
var urlReq:URLRequest = new URLRequest("http://www.adobe.com/");
html.load(urlReq);
html.width = 800;
html.height = 600;

var blur:BlurFilter = new BlurFilter(8);
var filters:Array = [blur];
html.filters = filters;
```

捲動 HTML 內容

Adobe AIR 1.0 以及更新的版本

HTMLLoader 類別包含下列屬性，可以讓您控制 HTML 內容的捲動方式：

屬性	說明
contentHeight	HTML 內容的高度，以像素為單位。
contentWidth	HTML 內容的寬度，以像素為單位。
scrollH	HTMLLoader 物件中 HTML 內容的水平捲動位置。
scrollV	HTMLLoader 物件中 HTML 內容的垂直捲動位置。

下列程式碼會設定 scrollV 屬性，將 HTML 內容向下捲動至頁面底部。

```
var html:HTMLLoader = new HTMLLoader();
html.addEventListener(Event.HTML_BOUNDS_CHANGE, scrollHTML);

const SIZE:Number = 600;
html.width = SIZE;
html.height = SIZE;

var urlReq:URLRequest = new URLRequest("http://www.adobe.com");
html.load(urlReq);
this.addChild(html);

function scrollHTML(event:Event):void
{
    html.scrollV = html.contentHeight - SIZE;
}
```

HTMLLoader 不包含水平和垂直捲軸。您可以使用 ActionScript 或 Flex 組件實作捲軸。Flex 的 HTML 組件會自動為 HTML 內容包含捲軸。您也可以使用 HTMLLoader.createRootWindow() 方法，建立包含具有捲軸的 HTMLLoader 物件 (請參閱第 877 頁「建立具有捲動 HTML 內容之功能的視窗」)。

存取 HTML 操作記錄清單

Adobe AIR 1.0 以及更新的版本

在 HTMLLoader 物件中載入新的網頁時，執行階段便會維護這個物件的操作記錄清單。操作記錄清單對應於 HTML 網頁中的 window.history 物件。HTMLLoader 類別包含下列屬性和方法，可以讓您搭配 HTML 操作記錄清單使用：

類別成員	說明
historyLength	操作記錄清單總長度，包括之前與之後項目。
historyPosition	在操作記錄清單中的目前位置。在這個位置之前的操作記錄項目都會以「向後」瀏覽表示，而在這個位置之後的項目則都會以「向前」瀏覽表示。
getHistoryAt()	傳回對應至操作記錄清單中，位於所指定位置之操作記錄項目的 URLRequest 物件。
historyBack()	如果可能，在瀏覽器操作記錄中向後瀏覽。
historyForward()	可能的話，在操作記錄清單中向前瀏覽。
historyGo()	在瀏覽器操作記錄中瀏覽指定的步驟數。如果為正數則向前瀏覽，如果為負數則向後瀏覽。如果瀏覽至零，便會重新整理網頁；如果指定結尾之後的位置，會瀏覽至清單的結尾。

操作記錄清單中的項目是儲存為 HTMLHistoryItem 類型的物件。HTMLHistoryItem 類別具有下列屬性：

屬性	說明
isPost	如果 HTML 網頁包含 POST 資料，請設定為 true。
originalUrl	在任何重新導向之前，HTML 網頁的原始 URL。
title	HTML 網頁的標題。
url	HTML 網頁的 URL。

設定載入 HTML 內容時所用的使用者代理程式

Adobe AIR 1.0 以及更新的版本

HTMLLoader 類別具有 userAgent 屬性，可以讓您設定 HTMLLoader 所用的使用者代理程式字串。請先設定 HTMLLoader 物件的 userAgent 屬性，然後再呼叫 load() 方法。如果您設定 HTMLLoader 實體的這項屬性，之後便「不會」使用傳遞至 load() 方法之 URLRequest 的 userAgent 屬性。

您可以藉由設定 URLRequestDefaults.userAgent 屬性，設定應用程式網域中所有 HTMLLoader 物件使用的預設使用者代理程式字串。對於所有 URLRequest 物件 (而非只有與 HTMLLoader 物件的 load() 方法搭配使用的 URLRequests)，都會套用這些靜態 URLRequestDefaults 屬性做為預設值。如果設定 HTMLLoader 的 userAgent 屬性，便會覆寫預設的 URLRequestDefaults.userAgent 設定。

如果您未針對 HTMLLoader 物件的 userAgent 屬性或 URLRequestDefaults.userAgent 設定使用者代理程式值，則會使用預設的 AIR 使用者代理程式值。這個預設值會根據執行階段作業系統 (如 Mac OS 或 Windows)、執行階段語言和執行階段版本而有所不同，如下列兩個範例所示：

- "Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/420+ (KHTML, like Gecko) AdobeAIR/1.0"
- "Mozilla/5.0 (Windows; U; en) AppleWebKit/420+ (KHTML, like Gecko) AdobeAIR/1.0"

設定字元編碼以供 HTML 內容使用

Adobe AIR 1.0 以及更新的版本

HTML 網頁中可以加入 meta 標籤，以指定網頁使用的字元編碼，如下所示：

```
meta http-equiv="content-type" content="text/html" charset="ISO-8859-1";
```

藉由設定 HTMLLoader 物件的 textEncodingOverride 屬性，即可覆寫網頁設定以確保使用的是特定字元編碼：

```
var html:HTMLLoader = new HTMLLoader();
html.textEncodingOverride = "ISO-8859-1";
```

當 HTML 網頁並未指定具有 HTMLLoader 物件之 textEncodingFallback 屬性的設定時，您可以指定 HTMLLoader 內容要使用的字元編碼：

```
var html:HTMLLoader = new HTMLLoader();
html.textEncodingFallback = "ISO-8859-1";
```

textEncodingOverride 屬性會覆寫 HTML 網頁中的設定，而且 textEncodingOverride 屬性和 HTML 網頁中的設定都會覆寫 textEncodingFallback 屬性。

請先設定 textEncodingOverride 或 textEncodingFallback 屬性，然後再載入 HTML 內容。

為 HTML 內容定義類似瀏覽器的使用者介面

Adobe AIR 1.0 以及更新的版本

JavaScript 提供了數個 API，可以讓您控制顯示 HTML 內容的視窗。在 AIR 中，實作自訂 HTMLHost 類別，就可以覆寫這些 API。

關於擴充 HTMLHost 類別

Adobe AIR 1.0 以及更新的版本

舉例而言，如果應用程式在索引標籤式介面中呈現多個 HTMLLoader 物件，您可能會希望載入之 HTML 網頁所做的標題變更可以變更索引標籤的標籤，而非變更主要視窗的標題。同樣地，藉由重新調整 HTMLLoader 物件在其父輩顯示物件容器中的位置、移動包含 HTMLLoader 物件的視窗、完全不執行任何動作，或者完整執行其它某些動作，程式碼便可以回應 window.moveTo() 呼叫。

AIR 的 HTMLHost 類別可以控制下列 JavaScript 屬性和方法：

- window.status
- window.document.title
- window.location
- window.blur()
- window.close()
- window.focus()
- window.moveBy()
- window.moveTo()
- window.open()
- window.resizeBy()
- window.resizeTo()

當您使用 new HTMLLoader() 建立 HTMLLoader 物件時，並不會啟用上述所列的 JavaScript 屬性或方法。HTMLHost 類別會提供這些 JavaScript API 的預設實作，該實作與瀏覽器類似。您也可以擴充 HTMLHost 類別以自訂行為。若要建立支援預設行為的 HTMLHost 物件，請將 HTMLHost 建構函式中的 defaultBehaviors 參數設定為 true：

```
var defaultHost:HTMLHost = new HTMLHost(true);
```

當您在 AIR 中使用 HTMLLoader 類別 createRootWindow() 方法建立 HTML 視窗時，會自動指定支援預設行為的 HTMLHost 實體。您可以將不同的 HTMLHost 實作指定給 HTMLLoader 的 htmlHost 屬性以變更主控物件行為，或者指定 null 以停用這整個功能。

備註：AIR 會將預設的 HTMLHost 物件指定給針對 HTML 類型之 AIR 應用程式所建立的初始視窗，以及 JavaScript window.open() 方法的預設實作所建立的所有視窗。

範例：擴充 HTMLHost 類別

Adobe AIR 1.0 以及更新的版本

下列範例會說明如何藉由擴充 HTMLHost 類別，自訂 HTMLLoader 物件影響使用者介面的方式：

Flex 範例 **:**

1. 建立可擴充 HTMLHost 類別的類別 (子類別)。
2. 覆寫這個新類別的方法，以處理與使用者介面相關之設定中的變更。例如，下列類別 CustomHost 會針對 window.open() 的呼叫以及 window.document.title 的變更定義行為。對 window.open() 的呼叫會在新視窗中開啟 HTML 網頁，而 window.document.title 的變更 (包含 HTML 網頁的 <title> 元素設定) 會設定該視窗的標題。

```
package
{
    import flash.html.*;
    import flash.display.StageScaleMode;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;

    public class CustomHost extends HTMLHost
    {
        import flash.html.*;
        override public function
            createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                                 windowCreateOptions.y,
                                                 windowCreateOptions.width,
                                                 windowCreateOptions.height);
            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                         windowCreateOptions.scrollBarsVisible, bounds);
            htmlControl.htmlHost = new HTMLHostImplementation();
            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }
            return htmlControl;
        }
        override public function updateTitle(title:String):void
        {
            htmlLoader.stage.nativeWindow.title = title;
        }
    }
}
```

3 在包含 HTMLLoader 的程式碼 (而非 HTMLHost 的新子類別之程式碼) 中，建立新類別的物件。指定這個新物件給 HTMLLoader 的 htmlHost 屬性。下列 Flex 程式碼會使用在上一個步驟中定義的類別 CustomHost：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    applicationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.html.HTMLLoader;
            import CustomHost;
            private function init():void
            {
                var html:HTMLLoader = new HTMLLoader();
                html.width = container.width;
                html.height = container.height;
                var urlReq:URLRequest = new URLRequest("Test.html");
                html.htmlHost = new CustomHost();
                html.load(urlReq);
                container.addChild(html);
            }
        ]]>
    </mx:Script>
    <mx:UIComponent id="container" width="100%" height="100%"/>
</mx:WindowedApplication>
```

若要測試在此所述的程式碼，請將具有下列內容的 HTML 檔包含在應用程式目錄中：

```
<html>
    <head>
        <title>Test</title>
    </head>
    <script>
        function openWindow()
        {
            window.runtime.trace("in");
            document.title = "foo"
            window.open('Test.html');
            window.runtime.trace("out");
        }
    </script>
    <body>
        <a href="#" onclick="openWindow()">window.open('Test.html')</a>
    </body>
</html>
```

Flash Professional 範例 **:**

1 建立供 AIR 使用的 Flash 檔案。將這個檔案的文件類別設定為 CustomHostExample，然後將檔案儲存為 CustomHostExample.fla。

2 建立名為 CustomHost.as 的 ActionScript 檔案，其中包含可擴充 HTMLHost 類別的類別 (子類別)。這個類別會覆寫新類別的特定方法，以處理與使用者介面相關之設定中的變更。例如，下列類別 CustomHost 會針對 window.open() 的呼叫以及 window.document.title 的變更定義行為。對 window.open() 方法的呼叫會在新視窗中開啟 HTML 網頁，而 window.document.title 屬性的變更 (包含 HTML 網頁的 <title> 元素設定) 會設定該視窗的標題。

```
package
{
    import flash.display.StageScaleMode;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;
    import flash.events.Event;
    import flash.events.NativeWindowBoundsEvent;
    import flash.geom.Rectangle;
    import flash.html.HTMLLoader;
    import flash.html.HTMLHost;
    import flash.html.HTMLWindowCreateOptions;
    import flash.text.TextField;

    public class CustomHost extends HTMLHost
    {
        public var statusField:TextField;

        public function CustomHost(defaultBehaviors:Boolean=true)
        {
            super(defaultBehaviors);
        }
        override public function windowClose():void
        {
            htmlLoader.stage.nativeWindow.close();
        }
        override public function createWindow(
                                windowCreateOptions:HTMLWindowCreateOptions ):HTMLLoader
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                        windowCreateOptions.y,
                                        windowCreateOptions.width,
                                        windowCreateOptions.height);
            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
```

```
                                    windowCreateOptions.scrollBarsVisible, bounds);
            htmlControl.htmlHost = new HTMLHostImplementation();
            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }
            return htmlControl;
        }
        override public function updateLocation(locationURL:String):void
        {
            trace(locationURL);
        }
        override public function set windowRect(value:Rectangle):void
        {
            htmlLoader.stage.nativeWindow.bounds = value;
        }
        override public function updateStatus(status:String):void
        {
            statusField.text = status;
            trace(status);
        }
        override public function updateTitle(title:String):void
        {
            htmlLoader.stage.nativeWindow.title = title + "- Example Application";
        }
        override public function windowBlur():void
        {
            htmlLoader.alpha = 0.5;
        }
        override public function windowFocus():void
        {
            htmlLoader.alpha = 1;
        }
    }
}
```

3 建立名為 CustomHostExample.as 的 ActionScript 檔案，以包含應用程式的文件類別。這個類別會建立 HTMLLoader 物件，並將其 host 屬性設定為上一個步驟中定義之 CustomHost 類別的實體：

```
package
{
    import flash.display.Sprite;
    import flash.html.HTMLLoader;
    import flash.net.URLRequest;
    import flash.text.TextField;

    public class CustomHostExample extends Sprite
    {
        function CustomHostExample():void
        {
            var html:HTMLLoader = new HTMLLoader();
            html.width = 550;
            html.height = 380;
            var host:CustomHost = new CustomHost();
            html.htmlHost = host;

            var urlReq:URLRequest = new URLRequest("Test.html");
            html.load(urlReq);

            addChild(html);

            var statusTxt:TextField = new TextField();
            statusTxt.y = 380;
            statusTxt.height = 20;
            statusTxt.width = 550;
            statusTxt.background = true;
            statusTxt.backgroundColor = 0xEEEEEEEE;
            addChild(statusTxt);

            host.statusField = statusTxt;
        }
    }
}
```

若要測試在此所述的程式碼，請將具有下列內容的 HTML 檔包含在應用程式目錄中：

```
<html>
     <head>
     <title>Test</title>
     <script>
     function openWindow()
     {
     document.title = "Test"
     window.open('Test.html');
     }
     </script>
     </head>
     <body bgColor="#EEEEEE">
     <a href="#" onclick="window.open('Test.html')">window.open('Test.html')</a>
     <br/><a href="#" onclick="window.document.location='http://www.adobe.com'">
     window.document.location = 'http://www.adobe.com'</a>
     <br/><a href="#" onclick="window.moveBy(6, 12)">moveBy(6, 12)</a>
     <br/><a href="#" onclick="window.close()">window.close()</a>
     <br/><a href="#" onclick="window.blur()">window.blur()</a>
     <br/><a href="#" onclick="window.focus()">window.focus()</a>
     <br/><a href="#" onclick="window.status = new Date().toString()">window.status=new
Date().toString()</a>
     </body>
</html>
```

處理 window.location 屬性的變更

Adobe AIR 1.0 以及更新的版本

您可以覆寫 locationChange() 方法，處理 HTML 網頁的 URL 變更。當網頁中的 JavaScript 變更 window.location 的值時，就會呼叫 locationChange() 方法。下列範例只會載入所要求的 URL：

```
override public function updateLocation(locationURL:String):void
{
    htmlLoader.load(new URLRequest(locationURL));
}
```

備註：您可以使用 HTMLHost 物件的 htmlLoader 屬性，參考目前的 HTMLLoader 物件。

處理 window.moveBy()、window.moveTo()、window.resizeTo()、window.resizeBy() 的 JavaScript 呼叫

Adobe AIR 1.0 以及更新的版本

您可以覆寫 set windowRect() 方法，處理 HTML 內容邊界中的變更。當網頁中的 JavaScript 呼叫 window.moveBy()、window.moveTo()、window.resizeTo() 或 window.resizeBy() 時，就會呼叫 set windowRect() 方法。下列範例只會更新桌面視窗中的邊界：

```
override public function set windowRect(value:Rectangle):void
{
    htmlLoader.stage.nativeWindow.bounds = value;
}
```

處理 window.open() 的 JavaScript 呼叫

Adobe AIR 1.0 以及更新的版本

您可以覆寫 createWindow() 方法，處理 window.open() 的 JavaScript 呼叫。createWindow() 方法的實作會負責建立新的 HTMLLoader 物件並將其傳回。您通常可以在新的視窗中顯示 HTMLLoader，但是並不需要建立視窗。

下列範例會示範如何使用 HTMLLoader.createRootWindow() 來實作 createWindow() 函數，以建立視窗和 HTMLLoader 物件兩者。您也可以另外建立 NativeWindow 物件，並將 HTMLLoader 加入至視窗舞台。

```
override public function createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader{
    var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
    var bounds:Rectangle = new Rectangle(windowCreateOptions.x, windowCreateOptions.y,
                                 windowCreateOptions.width, windowCreateOptions.height);
    var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                     windowCreateOptions.scrollBarsVisible, bounds);
    htmlControl.htmlHost = new HTMLHostImplementation();
    if(windowCreateOptions.fullscreen){
        htmlControl.stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
    }
    return htmlControl;
}
```

備註：在這個範例中，會將自訂 HTMLHost 實作指定給使用 window.open() 建立的所有新視窗。如有需要，您也可以使用不同的實作，或者將這些新視窗的 htmlHost 屬性設定為 null。

以 createWindow() 方法的參數形式傳遞的物件，就是 HTMLWindowCreateOptions 物件。HTMLWindowCreateOptions 類別包含下列屬性，這些屬性會報告在 window.open() 呼叫之 features 參數字串中設定的值：

HTMLWindowCreateOptions 屬性	在 window.open() 之 JavaScript 呼叫的 features 字串中之對應設定：
fullscreen	fullscreen
height	height
locationBarVisible	location
menuBarVisible	menubar
resizeable	resizable
scrollBarsVisible	scrollbars
statusBarVisible	status
toolBarVisible	toolbar
width	width
x	left 或 screenX
y	top 或 screenY

HTMLLoader 類別不會實作可以在 features 字串中指定的所有功能。因此，您的應用程式必須適時提供捲軸、位置列、選單列、狀態列和工具列。

JavaScript window.open() 方法的其它引數都是由系統來處理。createWindow() 實作不應該在 HTMLLoader 物件中載入內容，或者設定視窗標題。

處理 window.close() 的 JavaScript 呼叫

Adobe AIR 1.0 以及更新的版本

您可以覆寫 windowClose() 以處理 window.close() 方法的 JavaScript 呼叫。下列範例會在呼叫 window.close() 方法時，關閉桌面視窗：

```
override public function windowClose():void
{
    htmlLoader.stage.nativeWindow.close();
}
```

window.close() 的 JavaScript 呼叫不一定要關閉包含內容的視窗。例如，您可以從顯示清單移除 HTMLLoader，讓視窗 (可能包含其它內容) 保持為開啟狀態，如下列程式碼所示：

```
override public function windowClose():void
{
    htmlLoader.parent.removeChild(htmlLoader);
}
```

處理 window.status 屬性的變更

Adobe AIR 1.0 以及更新的版本

您可以覆寫 updateStatus() 方法以處理 window.status 的 JavaScript 變更。下列範例將追蹤 status 值：

```
 override public function updateStatus(status:String):void
{
    trace(status);
}
```

要求的狀態會當做字串傳遞至 updateStatus() 方法。

HTMLLoader 物件不會提供狀態列。

處理 window.document.title 屬性的變更

Adobe AIR 1.0 以及更新的版本

您可以覆寫 updateTitle() 方法以處理 window.document.title 值的 JavaScript 變更。下列範例會變更視窗標題，並且將字串「Sample」附加至標題：

```
 override public function updateTitle(title:String):void
{
    htmlLoader.stage.nativeWindow.title = title + " - Sample";
}
```

設定 HTML 網頁的 document.title 時，所要求的標題就會當做字串傳遞至 updateTitle() 方法。

document.title 的變更不一定要變更包含 HTMLLoader 物件之視窗的標題。舉例而言，您可以變更其它介面元素，例如文字欄位。

處理 window.blur() 和 window.focus() 的 JavaScript 呼叫

Adobe AIR 1.0 以及更新的版本

您可以覆寫 windowBlur() 和 windowFocus() 方法以處理 window.blur() 和 window.focus() 的 JavaScript 呼叫，如下列範例所示：

```
 override public function windowBlur():void
{
    htmlLoader.alpha = 0.5;
}
override public function windowFocus():void
{
    htmlLoader.alpha = 1.0;
    NativeApplication.nativeApplication.activate(htmlLoader.stage.nativeWindow);
}
```

備註：AIR 不會提供用於停用視窗或應用程式的 API。

建立具有捲動 HTML 內容之功能的視窗

Adobe AIR 1.0 以及更新的版本

HTMLLoader 包含 HTMLLoader.createRootWindow() 靜態方法，可以讓您開啟包含 HTMLLoader 物件的新視窗 (由 NativeWindow 物件表示)，並且為該視窗定義某些使用者介面設定。這個方法會採用四個參數，可以讓您定義使用者介面：

參數	說明
visible	Boolean 值，指定一開始就看得見 (true) 或看不見 (false) 視窗。
windowInitOptions	NativeWindowInitOptions 物件。NativeWindowInitOptions 類別會為 NativeWindow 物件定義數個初始化選項，包括下列項目：視窗是否可以最小化、最大化或重新調整大小；視窗具有系統顏色還是自訂顏色；視窗是否為透明 (針對不使用系統顏色的視窗而言)；視窗的類型。
scrollBarsVisible	是 (true) 否 (false) 具有捲軸。
bounds	定義新視窗之位置和大小的 Rectangle 物件。

例如，下列程式碼會使用 HTMLLoader.createRootWindow() 方法，建立具有會使用捲軸之 HTMLLoader 內容的視窗：

```
var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
var bounds:Rectangle = new Rectangle(10, 10, 600, 400);
var html2:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions, true, bounds);
var urlReq2:URLRequest = new URLRequest("http://www.example.com");
html2.load(urlReq2);
html2.stage.nativeWindow.activate();
```

備註：直接在 JavaScript 中呼叫 createRootWindow() 而建立的視窗不會受到開啟的 HTML 視窗影像。舉例來說，JavaScript Window 的 opener 和 parent 等屬性都是 null。不過，如果您覆寫 HTMLHost createWindow() 方法來呼叫 createRootWindow()，藉以直接呼叫 createRootWindow()，則 opener 和 parent 就一定會參考開啟的 HTML 視窗。

建立 HTMLLoader 類別的子類別

Adobe AIR 1.0 以及更新的版本

您可以建立 HTMLLoader 類別的子類別以建立新行為。舉例而言，您可以建立子類別以定義 HTMLLoader 事件 (例如，當呈現 HTML 時或當使用者按一下連結時，所傳送的事件) 的預設事件偵聽程式。

下列範例會擴充 HTMLHost 類別，以便在呼叫 JavaScript window.open() 方法時提供「一般」行為。接著，這個範例會定義 HTMLLoader 的子類別，這個子類別會使用自訂 HTMLHost 實作類別：

```
package
{
       import flash.html.HTMLLoader;
    public class MyHTMLHost extends HTMLHost
       {
           public function MyHTMLHost()
           {
               super(false);
        }
        override public function createWindow(opts:HTMLWindowCreateOptions):void
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(opts.x, opts.y, opts.width, opts.height);
            var html:HTMLLoader = HTMLLoader.createRootWindow(true,
                                                    initOptions,
                                                    opts.scrollBarsVisible,
                                                    bounds);
            html.stage.nativeWindow.orderToFront();
            return html
       }
}
```

下列程式碼會定義 HTMLLoader 類別的子類別，將 MyHTMLHost 物件指定給這個物件的 htmlHost 屬性：

```
package
{
        import flash.html.HTMLLoader;
    import MyHTMLHost;
    import HTMLLoader;
       public class MyHTML extends HTMLLoader
       {
           public function MyHTML()
           {
               super();
               htmlHost = new MyHTMLHost();
           }
       }
}
```

如需有關這個範例中使用之 HTMLHost 類別和 HTMLLoader.createRootWindow() 方法的詳細資訊，請參閱第 869 頁「為 HTML 內容定義類似瀏覽器的使用者介面」。

第 61 章 在 AIR 中處理 HTML 相關事件

Adobe AIR 1.0 以及更新的版本

事件處理系統可讓程式設計人員很方便地針對使用者輸入動作與系統事件做出回應。Adobe® AIR® 事件模型不僅方便好用，而且也符合業界標準。根據「文件物件模型第 3 層事件規格」(Document Object Model (DOM) Level 3 Events Specification，這是遵循業界標準的事件處理架構)，這個事件模型可提供程式設計人員強大且直覺式的事件處理工具。

HTMLLoader 事件

Adobe AIR 1.0 以及更新的版本

HTMLLoader 物件會傳送下列 Adobe® ActionScript® 3.0 事件：

事件	說明
htmlDOMInitialize	當建立 HTML 文件時 (但是會在剖析任何指令碼之前，或者將 DOM 節點加入至網頁之前) 傳送。
complete	在 HTML 網頁中引發 onload 事件之後，立即於建立 HTML DOM 以回應載入作業時傳送。
htmlBoundsChanged	當 contentWidth 和 contentHeight 兩個屬性其中之一或兩者皆已變更時傳送。
locationChange	當 HTMLLoader 的 location 屬性已變更時傳送。
locationChanging	在 HTMLLoader 的位置因為使用者瀏覽、JavaScript 呼叫或重新導向而變更之前傳送。當您呼叫 load()、loadString()、reload()、historyGo()、historyForward() 或 historyBack() 方法時，不會傳送 locationChanging 事件。 呼叫所傳送之事件物件的 preventDefault() 方法會取消瀏覽。 如果連結在系統瀏覽器中開啟，則不會傳送 locationChanging 事件，因為 HTMLLoader 並未變更位置。
scroll	每當 HTML 引擎變更捲動位置時傳送。scroll 事件可以因為瀏覽至網頁中的錨點連結 (# 連結) 而引發，也可以因為呼叫 window.scrollTo() 方法而引發。在文字輸入或文字區域中輸入文字也可以引發 scroll 事件。
uncaughtScriptException	在 HTMLLoader 中發生 JavaScript 例外而 JavaScript 程式碼中未捕捉到該例外時傳送。

您也可以為 JavaScript 事件 (例如 onClick) 註冊 ActionScript 函數。如需詳細資訊，請參閱第 880 頁「使用 ActionScript 處理 DOM 事件」。

使用 ActionScript 處理 DOM 事件

Adobe AIR 1.0 以及更新的版本

您可以註冊 ActionScript 函數來回應 JavaScript 事件。以下列 HTML 內容為例：

```
<html>
<body>
    <a href="#" id="testLink">Click me.</a>
</html>
```

您可以將 ActionScript 函數註冊為處理常式，以處理網頁中的所有事件。例如，下列程式碼會加入 clickHandler() 函數，做為 HTML 網頁中 testLink 元素之 onclick 事件的偵聽程式。

```
var html:HTMLLoader = new HTMLLoader( );
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.addEventListener(Event.COMPLETE, completeHandler);

function completeHandler(event:Event):void {
    html.window.document.getElementById("testLink").onclick = clickHandler;
}

function clickHandler( event:Object ):void {
    trace("Event of type: " + event.type );
}
```

所傳送的事件物件不是 flash.events.Event 類型或 Event 子類別之一。使用 Object 類別，宣告事件處理常式函數的引數類型。

您也可以使用 addEventListener() 方法，為下列事件註冊偵聽程式：例如，您可以使用下列程式碼，取代上述範例中的 completeHandler() 方法：

```
function completeHandler(event:Event):void {
    var testLink:Object = html.window.document.getElementById("testLink");
    testLink.addEventListener("click", clickHandler);
}
```

當某個偵聽程式指向特定 DOM 元素時，您最好先等待父輩 HTMLLoader 傳送 complete 事件，然後才加入事件偵聽程式。因為 HTML 網頁通常都會載入多個檔案，而且直到這些檔案都已經載入並經過剖析之後，才算是完整建置 HTML DOM。當所有元素都已經建立完成，HTMLLoader 便會傳送 complete 事件。

回應未捕捉到的 JavaScript 例外

Adobe AIR 1.0 以及更新的版本

以下列 HTML 為例：

```
<html>
<head>
    <script>
        function throwError() {
            var x = 400 * melbaToast;
        }
    </script>
</head>
<body>
    <a href="#" onclick="throwError()">Click me.</a>
</html>
```

它包含 throwError() 這個 JavaScript 函數，這個函數會參考未知的變數 melbaToast：

```
var x = 400 * melbaToast;
```

當 JavaScript 作業遇到不合規定的作業，而後者未在 JavaScript 中被 try/catch 結構捕捉到時，包含這個網頁的 HTMLLoader 物件便會傳送 HTMLUncaughtScriptExceptionEvent 事件。您可以註冊處理常式來處理這個事件，如下列程式碼所示：

```
var html:HTMLLoader = new HTMLLoader();
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.width = container.width;
html.height = container.height;
container.addChild(html);
html.addEventListener(HTMLUncaughtScriptExceptionEvent.UNCAUGHT_SCRIPT_EXCEPTION,
                          htmlErrorHandler);
function htmlErrorHandler(event:HTMLUncaughtJavaScriptExceptionEvent):void
{
    event.preventDefault();
    trace("exceptionValue:", event.exceptionValue)
    for (var i:int = 0; i < event.stackTrace.length; i++)
    {
        trace("sourceURL:", event.stackTrace[i].sourceURL);
        trace("line:", event.stackTrace[i].line);
        trace("function:", event.stackTrace[i].functionName);
    }
}
```

在 JavaScript 中，您可以使用 window.htmlLoader 屬性來處理相同的事件：

```
<html>
<head>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>

    <script>
        function throwError() {
            var x = 400 * melbaToast;
        }

        function htmlErrorHandler(event) {
            event.preventDefault();
            var message = "exceptionValue:" + event.exceptionValue + "\n";
            for (var i = 0; i < event.stackTrace.length; i++){
                message += "sourceURL:" + event.stackTrace[i].sourceURL +"\n";
                message += "line:" + event.stackTrace[i].line +"\n";
                message += "function:" + event.stackTrace[i].functionName + "\n";
            }
            alert(message);
        }

        window.htmlLoader.addEventListener("uncaughtScriptException", htmlErrorHandler);
    </script>
</head>
<body>
    <a href="#" onclick="throwError()">Click me.</a>
</html>
```

htmlErrorHandler() 事件處理常式會取消這個事件的預設行為 (亦即，將 JavaScript 錯誤訊息傳送至 AIR trace 輸出)，並且產生專屬的輸出訊息。這個事件處理常式會輸出 HTMLUncaughtScriptExceptionEvent 物件之 exceptionValue 的值。它會在 stackTrace 陣列中，輸出每個物件的所有屬性：

```
exceptionValue: ReferenceError: Can't find variable: melbaToast
sourceURL: app:/test.html
line: 5
function: throwError
sourceURL: app:/test.html
line: 10
function: onclick
```

使用 JavaScript 處理執行階段事件

Adobe AIR 1.0 以及更新的版本

執行階段類別支援使用 addEventListener() 方法來加入事件處理常式。若要為某個事件加入處理常式函數，請呼叫傳送該事件之物件的 addEventListener() 方法，提供事件類型和處理常式函數。舉例而言，若要在使用按一下視窗標題列的關閉按鈕來關閉視窗時，偵聽所傳送的 closing 事件，請使用下列陳述式：

```
window.nativeWindow.addEventListener(air.NativeWindow.CLOSING, handleWindowClosing);
```

建立事件處理常式函數

Adobe AIR 1.0 以及更新的版本

下列程式碼會建立簡單的 HTML 檔案，其中顯示主要視窗位置的相關資訊。名為 moveHandler() 的處理常式函數會偵聽主要視窗的 move 事件 (由 NativeWindowBoundsEvent 類別所定義)。

```
<html>
    <script src="AIRAliases.js" />
    <script>
        function init() {
            writeValues();
            window.nativeWindow.addEventListener(air.NativeWindowBoundsEvent.MOVE,
                                                 moveHandler);
        }
        function writeValues() {
            document.getElementById("xText").value = window.nativeWindow.x;
            document.getElementById("yText").value = window.nativeWindow.y;
        }
        function moveHandler(event) {
            air.trace(event.type); // move
            writeValues();
        }
    </script>
    <body onload="init()" />
        <table>
            <tr>
                <td>Window X:</td>
                <td><textarea id="xText"></textarea></td>
            </tr>
            <tr>
                <td>Window Y:</td>
                <td><textarea id="yText"></textarea></td>
            </tr>
        </table>
    </body>
</html>
```

當使用者移動視窗時，textarea 元素會顯示視窗更新後的 X 和 Y 位置。

請注意，該事件物件會當做引數傳遞給 moveHandler() 方法。event 參數可以讓您的處理常式函數檢查事件物件。在此範例中，您會使用該事件物件的 type 屬性，回報該事件為 move 事件。

移除事件偵聽程式

Adobe AIR 1.0 以及更新的版本

您可以使用 removeEventListener() 方法，移除不再需要的事件偵聽程式。移除不再用到的事件偵聽程式是很好的作法。與 addEventListener() 方法所需的參數一樣，使用上述方法必須搭配 eventName 和 listener 參數。

移除瀏覽之 HTML 網頁中的事件偵聽程式

Adobe AIR 1.0 以及更新的版本

當 HTML 內容在瀏覽時，或者當因為包含 HTML 內容的視窗關閉進而捨棄 HTML 內容時，並不會自動移除參考已取消載入網頁之物件的事件偵聽程式。在物件將事件傳送至已取消載入的處理常式時，您會看到下列錯誤訊息：「應用程式嘗試參考已不存在之 HTML 網頁中的 JavaScript 物件」。

若要避免這個錯誤，請在 HTML 網頁關閉之前，先移除網頁中的 JavaScript 事件偵聽程式。如果是在網頁瀏覽的情況下 (在 HTMLLoader 物件中)，請在 window 物件的 unload 事件期間移除事件偵聽程式。

例如，下列 JavaScript 程式碼會移除 uncaughtScriptException 事件的事件偵聽程式：

```
window.onunload = cleanup;
window.htmlLoader.addEventListener('uncaughtScriptException', uncaughtScriptException);
function cleanup()
{
    window.htmlLoader.removeEventListener('uncaughtScriptException',
                            uncaughtScriptExceptionHandler);
}
```

若要避免在關閉包含 HTML 內容的視窗時發生錯誤，請呼叫 cleanup 函數以回應 NativeWindow 物件 (window.nativeWindow) 的 closing 事件。例如，下列 JavaScript 程式碼會移除 uncaughtScriptException 事件的事件偵聽程式：

```
window.nativeWindow.addEventListener(air.Event.CLOSING, cleanup);
function cleanup()
{
    window.htmlLoader.removeEventListener('uncaughtScriptException',
                            uncaughtScriptExceptionHandler);
}
```

亦可在事件偵聽程式一執行時就予以移除 (若此事件只需處理一次)，以避免此錯誤發生。例如，下列 JavaScript 程式碼會呼叫 HTMLLoader 類別的 createRootWindow() 方法以建立 HTML 視窗，並加入 complete 事件的事件偵聽程式。呼叫 complete 事件處理常式時，它會使用 removeEventListener() 函數移除本身的事件偵聽程式：

```
var html = runtime.flash.html.HTMLLoader.createRootWindow(true);
html.addEventListener('complete', htmlCompleteListener);
function htmlCompleteListener()
{
    html.removeEventListener(complete, arguments.callee)
    // handler code..
}
html.load(new runtime.flash.net.URLRequest("second.html"));
```

移除不再需要的事件偵聽程式時，系統垃圾回收器便會針對與這些偵聽程式相關聯的記憶體進行記憶體回收。

檢查現有的事件偵聽程式

Adobe AIR 1.0 以及更新的版本

hasEventListener() 方法可以讓您檢查物件的事件偵聽程式是否存在。

第 62 章 在行動應用程式中顯示 HTML 內容

Adobe AIR 2.5 以及更新的版本

StageWebView 類別會使用行動裝置上的系統瀏覽器控制項以及使用桌上型電腦的標準 Adobe® AIR® HTMLLoader 控制項，以顯示 HTML 內容。請核取 StageWebView.isSupported 屬性以判斷目前的裝置是否支援此類別。不保證行動描述檔中的所有裝置都支援。

在所有描述檔中，StageWebView 類別僅支援 HTML 內容與其餘應用程式之間有限的互動。您可以控制瀏覽，但是不允許資料的跨指令碼處理或直接交換資料。您可以從本機或遠端 URL 載入內容，或是傳遞 HTML 的字串。

StageWebView 物件

StageWebView 物件並非顯示物件，因此無法新增至顯示清單。而是以直接附加至舞台的檢視區域來運作。StageWebView 內容會在任何顯示清單內容上方繪製。您無法控制多個 StageWebView 物件的繪製順序。

若要顯示 StageWebView 物件，您可以將要顯示該物件的舞台指定給 StageWebView 的 stage 屬性。使用 viewPort 屬性可設定顯示的大小。

請在 -8192 與 8191 之間設定 viewPort 屬性的 x 與 y 座標。寬度與高度的最大值為 8191。如果大小超過最大值，會擲出例外。

下列範例會建立 StageWebView 物件，並設定 stage 與 viewPort 屬性，然後顯示 HTML 字串：

```
var webView:StageWebView = new StageWebView();
webView.viewPort = new Rectangle( 0, 0, this.stage.stageWidth, this .stage.stageHeight);
webView.stage = this.stage;
var htmlString:String = "<!DOCTYPE HTML>" +
                        "<html><body>" +
                        "<p>King Philip could order five good steaks.</p>" +
                        "</body></html>";
webView.loadString( htmlString );
```

若要隱藏 StageWebView 物件，請將其 stage 屬性設定為 null。若要刪除整個物件，請呼叫 dispose() 方法。您可以選擇性呼叫 dispose()，但是確實能協助垃圾回收器較快回收物件使用的記憶體。

內容

您可以使用兩種方法，將內容載入 StageWebView 物件：loadURL() 與 loadString()。

loadURL() 方法會在指定的 URL 載入資源。您可以使用系統網頁瀏覽器控制項支援的任何 URI 配置，包括 data:、file:、http:、https: 和 javascript:。不支援 app: 與 app-storage: 配置。AIR 不會驗證 URL 字串。

loadString() 方法會載入包含 HTML 內容的常值字串。以此方法載入的頁面位置會以下列方式表示：

- 桌上型電腦：about:blank
- iOS：htmlString
- Android：編碼過的 htmlString 資料 URI 格式

URI 配置會判斷載入內嵌內容或資料的規則。

URI 配置	載入本機資源	載入遠端資源	本機 XMLHttpRequest	遠端 XMLHttpRequest
data:	否	是	否	否
file:	是	是	是	是
http:、https:	否	是	否	相同網域：
about: (loadString() 方法)	否	是	否	否

備註：如果將舞台的 displayState 屬性設定為 FULL_SCREEN，則在桌上型電腦中，您無法在 StageWebView 中顯示的文字欄位內進行輸入。不過，在 iOS 與 Android 中，即使舞台的 displayState 是 FULL_SCREEN，您仍然可以在 StageWebView 的文字欄位中輸入。

下列範例使用 StageWebView 物件來顯示 Adobe 的網站：

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;

    public class StageWebViewExample extends MovieClip{

        var webView:StageWebView = new StageWebView();

        public function StageWebViewExample() {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadURL( "http://www.adobe.com" );
        }
    }
}
```

在 Android 裝置上，您必須指定 Android INTERNET 權限，應用程式才能順利載入遠端資源。

在 Android 3.0+ 中，應用程式必須在 AIR 應用程式描述器的 Android manifestAdditions 元素中啟用硬體加速，才能顯示 StageWebView 物件中的外掛程式內容。請參閱在 StageWebView 物件中啟用 Flash Player 和其他外掛程式。

JavaScript URI

您可以使用 JavaScript URI，來呼叫 StageWebView 物件載入之 HTML 頁面所定義的函數。使用 JavaScript URI 呼叫的函數會在載入的網頁內容中執行。下列範例使用 StageWebView 物件來呼叫 JavaScript 函數：

```
package {
    import flash.display.*;
    import flash.geom.Rectangle;
    import flash.media.StageWebView;
    public class WebView extends Sprite
    {
        public var webView:StageWebView = new StageWebView();
        public function WebView()
        {
            var htmlString:String = "<!DOCTYPE HTML>" +
            "<html><script type=text/javascript>" +
            "function callURI(){" +
            "alert(\"You clicked me!!\");"+
            "}</script><body>" +
            "<p><a href=javascript:callURI()>Click Me</a></p>" +
            "</body></html>";
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadString( htmlString );
        }
    }
}
```

更多說明主題

Sean Voisen：善加利用 StageWebView

瀏覽事件

當使用者按一下 HTML 中的連結時，StageWebView 物件會傳送 locationChanging 事件。您可以呼叫事件物件的 preventDefault() 方法，以停止瀏覽。否則，StageWebView 物件會導覽至新頁面，並傳送 locationChange 事件。當頁面載入完成時，StageWebView 會傳送 complete 事件。

每個 HTML 重新導向都會傳送 locationChanging 事件。在適當時機會傳送 locationChange 與 complete 事件。

在 iOS 上，除了第一個 loadURL() 或 loadString() 方法以外，會在 locationChange 事件之前傳送 locationChanging 事件。也會透過 iFrame 與 Frame 為導覽變更傳送 locationChange 事件。

下列範例說明您如何防止位置變更，並改在系統瀏覽器中開啟新頁面。

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.events.LocationChangeEvent;
    import flash.geom.Rectangle;
    import flash.net.navigateToURL;
    import flash.net.URLRequest;

    public class StageWebViewNavEvents extends MovieClip{
        var webView:StageWebView = new StageWebView();

        public function StageWebViewNavEvents() {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
        webView.addEventListener( LocationChangeEvent.LOCATION_CHANGING, onLocationChanging );
            webView.loadURL( "http://www.adobe.com" );
        }
        private function onLocationChanging( event:LocationChangeEvent ):void
        {
            event.preventDefault();
            navigateToURL( new URLRequest( event.location ) );
        }
    }
}
```

步驟記錄

使用者按一下 StageWebView 物件所顯示內容的連結時，此控制項會儲存向後與往前的操作記錄堆疊。下列範例說明如何導覽兩個操作記錄堆疊。此範例使用 「後退」和 「搜尋」軟鍵。

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;

    public class StageWebViewExample extends MovieClip{

        var webView:StageWebView = new StageWebView();

        public function StageWebViewExample()
        {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadURL( "http://www.adobe.com" );

            stage.addEventListener( KeyboardEvent.KEY_DOWN, onKey );
        }

        private function onKey( event:KeyboardEvent ):void
        {
            if( event.keyCode == Keyboard.BACK && webView.isHistoryBackEnabled )
            {
                trace("back");
                webView.historyBack();
                event.preventDefault();
            }
            if( event.keyCode == Keyboard.SEARCH && webView.isHistoryForwardEnabled )
            {
                trace("forward");
                webView.historyForward();
            }
        }
    }
}
```

焦點

雖然它不是顯示物件，但是 StageWebView 類別包含的成員可讓您管理此控制項的焦點轉換。

當 StageWebView 物件獲得焦點時，它會傳送 focusIn 事件。若有必要，您可以使用此事件來管理應用程式中的焦點元素。

當 StageWebView 放棄焦點時，它會傳送 focusOut 事件。當使用者使用裝置軌跡球或方向鍵在網頁上 「點擊」，以傳遞第一個或最後一個控制項時，StageWebView 實體可能會放棄焦點。事件物件的 direction 屬性可讓您知道焦點流排是否超過頁面的頂端或頁面的底端。使用此資訊以指定焦點至 StageWebView 上方或下方的適當顯示物件。

在 iOS 上，無法以程式設計方式設定焦點。StageWebView 會傳送 focusIn 與 focusOut 事件，並將 FocusEvent 的 direction 屬性設定為 none。如果使用者在 StageWebView 內部觸控，會傳送 focusIn 事件。如果使用者在 StageWebView 外部觸控，會傳送 focusOut 事件。

下列範例說明如何將焦點從 StageWebView 物件傳遞至 Flash 顯示物件：

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;
    import flash.text.TextField;
    import flash.text.TextFieldType;
    import flash.events.FocusEvent;
    import flash.display.FocusDirection;
    import flash.events.LocationChangeEvent;

    public class StageWebViewFocusEvents extends MovieClip{
        var webView:StageWebView = new StageWebView();
        var topControl:TextField = new TextField();
        var bottomControl:TextField = new TextField();

        public function StageWebViewFocusEvents()
        {
            trace("Starting");
            topControl.type = TextFieldType.INPUT;
            addChild( topControl );
            topControl.height = 60;
            topControl.width = stage.stageWidth;
            topControl.background = true;
            topControl.text = "One control on top.";
            topControl.addEventListener( FocusEvent.FOCUS_IN, flashFocusIn );
            topControl.addEventListener( FocusEvent.FOCUS_OUT, flashFocusOut );

            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 60, stage.stageWidth, stage.stageHeight
- 120 );
            webView.addEventListener( FocusEvent.FOCUS_IN, webFocusIn );
            webView.addEventListener(FocusEvent.FOCUS_OUT, webFocusOut );
            webView.addEventListener(LocationChangeEvent.LOCATION_CHANGING,
                                     function( event:LocationChangeEvent ):void
                                     {
                                        event.preventDefault();
                                     } );
            webView.loadString("<form action='#'><input/><input/><input/></form>");
            webView.assignFocus();

            bottomControl.type = TextFieldType.INPUT;
            addChild( bottomControl );
            bottomControl.y = stage.stageHeight - 60;
            bottomControl.height = 60;
            bottomControl.width = stage.stageWidth;
            bottomControl.background = true;
            bottomControl.text = "One control on the bottom.";
            bottomControl.addEventListener( FocusEvent.FOCUS_IN, flashFocusIn );
            bottomControl.addEventListener( FocusEvent.FOCUS_OUT, flashFocusOut );}

        private function webFocusIn( event:FocusEvent ):void
        {
            trace("Web focus in");
        }

        private function webFocusOut( event:FocusEvent ):void
        {
            trace("Web focus out: " + event.direction);
            if( event.direction == FocusDirection.TOP )
            {
```

```
                stage.focus = topControl;
            }
            else
            {
                stage.focus = bottomControl;
            }
        }

        private function flashFocusIn( event:FocusEvent ):void
        {
            trace("Flash focus in");
            var textfield:TextField = event.target as TextField;
            textfield.backgroundColor = 0xff5566;
        }

        private function flashFocusOut( event:FocusEvent ):void
        {
            trace("Flash focus out");
            var textfield:TextField = event.target as TextField;
            textfield.backgroundColor = 0xffffff;
        }

    }
}
```

點陣圖捕捉

StageWebView 物件會顯示在所有顯示清單內容的上方。您無法在 StageWebView 物件的上方新增內容。例如，您無法在 StageWebView 內容上方展開下拉式清單。若要解決這個問題，請捕捉 StageWebView 的快照。然後，隱藏 StageWebView 並改為新增點陣圖快照。

下列範例說明如何使用 drawViewPortToBitmapData 方法補捉 StageWebView 物件的快照。它透過將舞台設定為 null，以隱藏 StageWebView 物件。在完全載入網頁後，它會呼叫補捉點陣圖的函數並顯示它。當您執行時，程式碼會顯示兩個標籤：Google 與 Facebook。按一下標籤會補捉對應的網頁，並將它顯示為舞台上的快照。

```
package
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.geom.Rectangle;
    import flash.media.StageWebView;
    import flash.net.*;
    import flash.text.TextField;
    public class stagewebview extends Sprite
    {
        public var webView:StageWebView=new StageWebView();
        public var textGoogle:TextField=new TextField();
        public var textFacebook:TextField=new TextField();
        public function stagewebview()
        {
            textGoogle.htmlText="<b>Google</b>";
            textGoogle.x=300;
            textGoogle.y=-80;
            addChild(textGoogle);
            textFacebook.htmlText="<b>Facebook</b>";
            textFacebook.x=0;
            textFacebook.y=-80;
```

```
            addChild(textFacebook);
            textGoogle.addEventListener(MouseEvent.CLICK,goGoogle);
            textFacebook.addEventListener(MouseEvent.CLICK,goFaceBook);
            webView.stage = this.stage;
            webView.viewPort = new Rectangle(0, 0, stage.stageWidth, stage.stageHeight);
        }
    public function goGoogle(e:Event):void
        {
            webView.loadURL("http://www.google.com");
            webView.stage = null;
            webView.addEventListener(Event.COMPLETE,handleLoad);
        }

    public function goFaceBook(e:Event):void
        {
            webView.loadURL("http://www.facebook.com");
            webView.stage = null;
            webView.addEventListener(Event.COMPLETE,handleLoad);
        }
    public function handleLoad(e:Event):void
        {
            var bitmapData:BitmapData = new BitmapData(webView.viewPort.width, webView.viewPort.height);
            webView.drawViewPortToBitmapData(bitmapData);
            var webViewBitmap:Bitmap=new Bitmap(bitmapData);
            addChild(webViewBitmap);
        }
    }
}
```

第 63 章 將 Worker 用於並行

適用於桌面平台的 **Flash Player 11.4** 以及更新的版本，**Adobe AIR 13.4** 以及更新的版本

ActionScript Worker 可讓您同時執行程式碼，換句話說，您可以在背景執行程式碼，而不會中斷主要程式碼的執行。

ActionScript 並行 API 僅在桌面平台的 Flash Player 11.4 及更新版本與 AIR 3.4 及更新版本中提供。行動平台的 AIR 不支援並行。

瞭解 Worker 與並行

適用於桌面平台的 **Flash Player 11.4** 以及更新的版本，**Adobe AIR 13.4** 以及更新的版本

當應用程式未使用 Worker 時，應用程式的程式碼會在單一線性區塊中執行，該區塊執行的步驟稱為 「執行緒」。執行緒會執行開發人員所撰寫的程式碼，也會執行屬於執行階段一部分、大多數的程式碼，特別是會在顯示物件的屬性變更時更新畫面的程式碼。雖然程式碼是在區塊中撰寫為方法和類別，但是在執行階段，程式碼會一次執行一行，就好像它是經過一連串冗長的步驟撰寫而成的。請將這個假設性範例視為應用程式會執行的步驟：

1. 進入影格：執行階段會呼叫任何 enterFrame 事件處理常式並逐一執行它們的程式碼
2. 滑鼠事件：使用者移動滑鼠，並且執行階段會在出現各種變換影像和滑出事件時呼叫任何滑鼠事件處理常式
3. 載入完成事件：從 URL 載入 XML 檔案的要求會隨著載入的檔案資料一起傳回。事件處理常式已呼叫，並且會執行其步驟，讀取 XML 內容並從 XML 資料建立一組物件。
4. 滑鼠事件：滑鼠再次移動，因此，執行階段會呼叫相關的滑鼠事件處理常式
5. 顯示：已沒有事件正在等候，因此，執行階段會根據對顯示物件所做的任何變更來更新畫面
6. 進入影格：再次開始循環

如範例所述，假設性步驟 1-5 會在一個稱為影格的時間區塊內依序執行。由於這些步驟是在單一執行緒中依序執行，因此，執行階段無法中斷程序的一個步驟來執行另一個步驟。以每秒 30 格的影格速率來說，執行階段只有不到 1/30 秒的時間來執行所有這些作業。在許多情況中，那已讓程式碼有足夠的時間來執行，而執行階段在剩餘時間裡也只是等待而已。不過，請假設步驟 3 中載入的 XML 資料是非常大、深度巢狀的 XML 結構。隨著程式碼循環查看 XML 並建立物件，可能會花費 1/30 秒以上的時間來執行該工作。在此情況下，後面的步驟 (回應滑鼠並重繪畫面) 無法如預期般儘快進行。這會導致畫面出現凍結及斷續的情形，因為畫面的重繪速度不夠快，無法回應使用者移動滑鼠的動作。

如果所有程式碼都在相同的執行緒中執行，只有一個方法可以避免偶爾出現斷續及凍結的情形，那就是不要執行需長時間執行的作業，例如循環查看大量資料。ActionScript Worker 提供其他解決方案。使用 Worker，您就可以在個別 Worker 中執行需長時間執行的程式碼。每個 Worker 會以個別執行緒執行，因此，背景 Worker 會在專屬的執行緒中執行需長時間執行的作業。如此，就會釋放主要 Worker 的執行緒，以重繪每個影格的畫面，而不會遭到其他工作封鎖。

這種同時執行多個程式碼作業的能力，又稱為 「並行」。當背景 Worker 完成工作或達到 「進度」時，您就可以傳送主要 Worker 通知和資料。透過這種方式，您可以撰寫用來執行複雜或費時作業的程式碼，而又能避免讓使用者遇到畫面凍結的情形。

Worker 很實用，因為它降低了影格速率因主要顯示執行緒遭到其他程式碼封鎖而丟棄的可能性。不過，Worker 需要額外的系統記憶體和 CPU 使用量，這會讓整體應用程式效能付出不少代價。由於每個 Worker 都會使用專屬的執行階段虛擬機器實體，因此，即使是簡單的 Worker，工作負荷也會很大。在使用 Worker 時，請在所有目標平台間測試您的程式碼，以確保系統上的需求不會太大。Adobe 建議在一般情況下，不要使用超過一個或兩個背景 Worker。

建立及管理 Worker

適用於桌面平台的 **Flash Player 11.4** 以及更新的版本，**Adobe AIR 13.4** 以及更新的版本

將 Worker 做為並行之用的第一步，就是建立背景 Worker。您可以使用兩種物件來建立 Worker。第一種是 Worker 實體，即您所建立的內容。另一種則是 WorkerDomain 物件，此物件會在應用程式中建立 Worker 並管理執行中的 Worker 物件。

當執行階段載入時，就會自動建立 WorkerDomain 物件。此外，執行階段也會自動為應用程式的主要 SWF 建立 Worker。第一個 Worker 又稱為「原始 Worker」。

由於應用程式只有一個 WorkerDomain 物件，因此，您可以使用靜態 WorkerDomain.current 屬性來存取 WorkerDomain 實體。

不管何時，您都可以使用靜態 Worker.current 屬性來存取目前的 Worker 實體 (目前程式碼執行所在的 Worker)。

從 SWF 建立 Worker 物件

就如同主要 SWF 在原始 Worker 內執行一樣，背景 Worker 也會執行單一 SWF 檔案的程式碼。若要使用背景 Worker，您必須將 Worker 的程式碼編寫並編譯為 SWF 檔案。若要建立背景 Worker，父 Worker 需要存取該 SWF 檔案做為 ByteArray 物件的位元組。將該 ByteArray 傳遞至 WorkerDomain 物件的 createWorker() 方法，就會實際建立 Worker。

您可以使用三種主要方式，取得背景 Worker SWF 做為 ByteArray 物件：

內嵌 Worker SWF

使用 [Embed] 中繼標籤，可將 Worker SWF 當做 ByteArray 內嵌於主要 SWF 中：

```
[Embed(source="../swfs/BgWorker.swf", mimeType="application/octet-stream")]
private static var BgWorker_ByteClass:Class;
private function createWorker():void
{
    var workerBytes:ByteArray = new BgWorker_ByteClass();
    var bgWorker:Worker = WorkerDomain.current.createWorker(workerBytes);

    // ... set up worker communication and start the worker
}
```

Worker SWF 會編譯為主要 SWF 當做 ByteArray 子類別，名為 BgWorker_ByteClass。建立該類別的實體會產生已預先填入 Worker SWF 位元組的 ByteArray。

載入外部 Worker SWF

使用 URLLoader 物件可載入外部 SWF 檔案。SWF 檔案必須來自相同的安全性網域，例如，自與主要 SWF 相同的網際網路網域載入的 SWF 檔案，或是 AIR 應用程式套件隨附的 SWF 檔案。

```
var workerLoader:URLLoader = new URLLoader();
workerLoader.dataFormat = URLLoaderDataFormat.BINARY;
workerLoader.addEventListener(Event.COMPLETE, loadComplete);
workerLoader.load(new URLRequest("BgWorker.swf"));

private function loadComplete(event:Event):void
{
    // create the background worker
    var workerBytes:ByteArray = event.target.data as ByteArray;
    var bgWorker:Worker = WorkerDomain.current.createWorker(workerBytes);

    // ... set up worker communication and start the worker
}
```

當 URLLoader 完成載入 SWF 檔案時，SWF 的位元組即可在 URLLoader 物件的 data 屬性 (在這個範例中，即為 event.target.data) 中取得。

將主要 SWF 當做 Worker SWF 使用

您可以將單一 SWF 同時當做主要 SWF 和 Worker SWF 來使用。使用主要顯示類別的 loaderInfo.bytes 屬性可存取 SWF 的位元組。

```
// The primordial worker's main class constructor
public function PrimordialWorkerClass()
{
    init();
}

private function init():void
{
    var swfBytes:ByteArray = this.loaderInfo.bytes;

    // Check to see if this is the primordial worker or the background worker
    if (Worker.current.isPrimordial)
    {
        // create a background worker
        var bgWorker:Worker = WorkerDomain.current.createWorker(swfBytes);

        // ... set up worker communication and start the worker
    }
    else // entry point for the background worker
    {
        // set up communication between workers using getSharedProperty()
        // ... (not shown)

        // start the background work
    }
}
```

如果要使用這個技巧，請使用 if 陳述式，在主要類別的建構函式或所呼叫的方法內分支 SWF 檔案程式碼。若要判斷程式碼是在主要 Worker 或背景 Worker 中執行，請檢查目前 Worker 物件的 isPrimordial 屬性，如範例所示。

開始執行 Worker

一旦建立 Worker，您就能透過呼叫 Worker 物件的 start() 方法，開始執行它的程式碼。start() 作業不會立即進行。若要得知 Worker 何時執行，請為 Worker 物件的 workerState 事件註冊一個偵聽程式。當 Worker 物件切換效期中的狀態時，就會傳送該事件，例如開始執行程式碼的時候。在您的 workerState 事件處理常式中，請確認 Worker 物件的 state 屬性為 WorkerState.RUNNING。此時，Worker 正在執行且其主要類別的建構函式已經執行。下列程式碼清單說明註冊 workerState 事件及呼叫 start() 方法的範例：

```
// listen for worker state changes to know when the worker is running
bgWorker.addEventListener(Event.WORKER_STATE, workerStateHandler);
// set up communication between workers using
// setSharedProperty(), createMessageChannel(), etc.
// ... (not shown)
bgWorker.start();
private function workerStateHandler(event:Event):void
{
    if (bgWorker.state == WorkerState.RUNNING)
    {
        // The worker is running.
        // Send it a message or wait for a response.
    }
}
```

管理 Worker 執行

不管何時，您都可以在應用程式中使用 WorkerDomain 類別的 listWorkers() 方法來存取執行中的 Worker 集。這個方法會傳回 state 屬性為 WorkerState.RUNNING 的 Worker 集，其中包括原始 Worker。對於尚未啟動或執行已遭停止的 Worker，則不會包含在內。

如果不再需要 Worker，您可以呼叫 Worker 物件的 terminate() 方法來關閉 Worker，並且釋放它的記憶體和其他系統資源。

Worker 之間的通訊

適用於桌面平台的 **Flash Player 11.4** 以及更新的版本，**Adobe AIR 13.4** 以及更新的版本

儘管 Worker 在個別執行緒中執行程式碼，但若彼此完全隔離，便無法提供任何好處。歸根究柢，Worker 之間的通訊是指，在 Worker 之間傳遞資料。您可以使用三個主要機制，取得 Worker 之間的資料。

在決定哪些資料共用技巧適用於特定資料傳遞需求時，請將兩個主要方式的差異納入考量。兩者之間的一個差異在於，一個有事件可用來通知接收端，有新資料可供使用，另一個則是接收端 Worker 必須檢查是否有更新。資料共用技巧之間的另一個差異與實際傳遞資料的方式有關。在某些情況中，接收端 Worker 取得的是共用資料的副本，這表示建立更多物件花費了更多記憶體和 CPU 循環。在其他情況中，Worker 存取物件會參考相同的基礎系統記憶體，這表示建立較少物件，整體使用的記憶體也較少。下面概述這些差異：

通訊技巧	接收資料時會傳送事件	共用 Worker 之間的記憶體
Worker 共用屬性	否	否，物件是副本，而非參考
MessageChannel	是	否，物件是副本，而非參考
可共用的 ByteArray	否	是，共用記憶體

使用共用屬性傳遞資料

在 Worker 之間共用資料的最基本方式是，使用共用屬性。每個 Worker 都會維護共用屬性值的內部字典。屬性是以字串索引鍵名稱來儲存，以區分各個屬性。若要儲存 Worker 上的物件做為共用屬性，請使用兩個引數、索引鍵名稱和要儲存的值來呼叫 Worker 物件的 setSharedProperty() 方法：

```
// code running in the parent worker
bgWorker.setSharedProperty("sharedPropertyName", someObject);
```

設定共用屬性之後，透過呼叫 Worker 物件的 getSharedProperty() 方法以傳入索引鍵名稱，即可讀取值：

```
// code running in the background worker
receivedProperty = Worker.current.getSharedProperty("sharedPropertyName");
```

Worker 讀取或設定屬性值沒有任何限制。例如，背景 Worker 中的程式碼可以呼叫它的 setSharedProperty() 方法來儲存值。然後，在父 Worker 中執行的程式碼可以使用 getSharedProperty() 來接收資料。

傳遞至 setSharedProperty() 方法的值幾乎可以是任何類型的物件。當您呼叫 getSharedProperty() 方法時，傳回的物件即為傳遞至 setSharedProperty() 的物件副本，而非相同物件的參數，少數特殊情況除外。如需資料共用方式的特殊之處，請參閱第 899 頁「共用的參考和複製的值」。

使用共用屬性在 Worker 之間傳遞資料的最大優點在於，即使在 Worker 執行之前也可供使用。即使在 Worker 執行之前，您也可以呼叫背景 Worker 物件的 setSharedProperty() 方法來設定共用屬性。當父 Worker 呼叫 Worker 的 start() 方法時，執行階段會呼叫子 Worker 主要類別的建構函式。任何在呼叫 start() 之前設定的共用屬性，都可供子 Worker 中的程式碼讀取。

使用 MessageChannel 傳遞資料

訊息通道在兩個 Worker 之間提供單向的資料傳遞連結。使用 MessageChannel 物件在 Worker 之間傳遞資料具有一個關鍵優勢。當您使用訊息通道傳送訊息 (物件) 時，MessageChannel 物件會傳送 channelMessage 事件。接收端 Worker 中的程式碼可以偵聽該事件，以得知資料何時可供使用。如此一來，接收端 Worker 就不需要不斷地檢查資料更新。

訊息通道只與兩個 Worker 產生關聯，亦即傳送者和接收者。若要建立 MessageChannel 物件，請呼叫傳送端 Worker 物件的 createMessageChannel() 方法，以傳遞接收端 Worker 做為引數：

```
// In the sending worker swf
var sendChannel:MessageChannel;
sendChannel = Worker.current.createMessageChannel(receivingWorker);
```

兩種 Worker 都需要存取 MessageChannel 物件。執行這項作業的最簡單方式就是，使用 setSharedProperty() 方法傳遞 MessageChannel 物件：

```
receivingWorker.setSharedProperty("incomingChannel", sendChannel);
```

在接收端 Worker 中，請為 MessageChannel 物件的 channelMessage 事件註冊偵聽程式。當傳送端 Worker 透過訊息通道傳送資料時，就會傳送這個事件。

```
// In the receiving worker swf
var incomingChannel:MessageChannel;
incomingChannel = Worker.current.getSharedProperty("incomingChannel");
incomingChannel.addEventListener(Event.CHANNEL_MESSAGE, handleIncomingMessage);
```

若要實際傳送資料，請在傳送端 Worker 中呼叫 MessageChannel 物件的 send() 方法：

```
// In the sending worker swf
sendChannel.send("This is a message");
```

在接收端 Worker 中，MessageChannel 會呼叫 channelMessage 事件處理常式。然後，接收端 Worker 可以透過呼叫 MessageChannel 物件的 receive() 方法來取得資料。

```
private function handleIncomingMessage(event:Event):void
{
    var message:String = incomingChannel.receive() as String;
}
```

接收方法所傳回的物件，其資料類型與傳遞至 send() 方法的物件相同。接收的物件是傳送者所傳遞之物件的副本，而不是傳送端 Worker 中的物件參考，除非它是少數資料類型之一，如第 899 頁「共用的參考和複製的值」中所述。

使用可共用的 ByteArray 共用資料

在兩個 Worker 之間傳遞物件時，接收端 Worker 會取得新的物件，該物件是原始物件的副本。兩個物件儲存在系統記憶體的不同位置。因此，所接收之物件的每個副本會增加執行階段所使用的記憶體總數。此外，您對 Worker 中的物件所做的任何變更不會影響其他 Worker 中的副本。如需有關如何複雜資料的詳細資訊，請參閱第 899 頁「共用的參考和複製的值」。

根據預設，ByteArray 物件會使用相同的行為。如果您將 ByteArray 實體傳遞至 Worker 物件的 setSharedProperty() 方法或 MessageChannel 物件的 send() 方法，執行階段就會在電腦的記憶體中建立新的 ByteArray，並且接收端 Worker 會取得參考那個新 ByteArray 的 ByteArray 實體。不過，您可以將 ByteArray 物件的 shareable 屬性設定為 true，藉此變更這個行為。

當可共用的 ByteArray 物件在 Worker 之間傳遞時，接收端 Worker 中的 ByteArray 實體是傳送端 Worker 中 ByteArray 實體所使用之相同基礎作業系統記憶體的參考。當一個 Worker 中的程式碼變更位元組陣列的內容時，那些變更會立即出現在另一個可以存在該共用位元組陣列的 Worker 中。

由於多個 Worker 會同時執行程式碼，因此，可能會有兩個 Worker 同時嘗試存取位元組陣列中的相同位元組。這可能會導致資料遺失或損毀。您可以使用幾個 API 來管理共用資源的存取，並避免發生那些問題。

ByteArray 類別的方法可讓您驗證並變更單一作業中的位元組陣列內容。

- atomicCompareAndSwapIntAt() 方法

- atomicCompareAndSwapLength() 方法

此外，flash.concurrent 套件中的類別會提供可與共用資源搭配使用的存取控制：

- Mutex 類別
- Condition 類別

共用的參考和複製的值

在正常情況下，當您呼叫 Worker.setSharedProperty() 或 MessageChannel.send() 時，傳遞至接收端 Worker 的物件會透過序列化為 AMF 格式的方式來傳遞。這會產生一些後果：

- 呼叫 getSharedProperty() 方法時在接收端 Worker 中建立的物件，會從 AMF 位元組還原序列化。它是原始物件的副本，而非物件的參考。在任一 Worker 中對物件所做的任何變更，都不會在另一個 Worker 中的副本變更。
- 無法使用 Worker.setSharedProperty() 或 MessageChannel.send()，將不能序列化為 AMF 格式的物件 (例如顯示物件) 傳遞至 Worker。
- 為了讓自訂類別可以順利還原序列化，類別定義必須使用 flash.net.registerClassAlias() 函數或 [RemoteClass] 中繼資料來註冊。類別的兩個 Worker 版本必須使用相同的別名。

下列是五個確實在 Worker 之間共用而非複製的特殊物件：

- Worker 物件
- MessageChannel 物件
- 可共用的位元組陣列 (shareable 屬性為 true 的 ByteArray 物件)
- Mutex 物件
- Condition 物件

當您使用 Worker.setSharedProperty() 方法或 MessageChannel.send() 方法傳遞其中一個物件的實體時，每個 Worker 會有相同基礎物件的參考。對一個 Worker 中的實體所做的變更會立即出現在另一個 Worker 中。外，如果您將其中一個物件的相同實體傳遞至 Worker 一次以上，執行階段便無法在接收端 Worker 中建立物件的新副本。反而會重複使用相同參考。

其他資料共用技巧

除了用於傳遞資料的 Worker 特定機制以外，Worker 也可以使用任何支援在兩個 SWF 應用程式之間共用資料的現有 API 來交換資料，如下所示：

- 本機共享物件
- 將資料寫入一個 Worker 中的檔案，並從另一個 Worker 的檔案中讀取
- 將資料儲存至 SQLite 資料庫，並從中讀取資料

當您在兩個以上的 Worker 之間共用資源時，通常必須避免讓多個 Worker 同時存取資源。例如，讓多個 Worker 存取本機檔案系統上的檔案可能會造成資料遺失或損毀，並且不受作業系統支援。

為了避免並行存取問題，請使用 flash.concurrent 套件中的 Mutex 和 Condition 類別，以提供可與共用資源搭配使用的存取控制。

與其他資料共用機制不同，SQLite 資料庫引擎是專為並行存取而設計，並且內建專屬的交易支援。多個 Worker 可以存取 SQLite 資料庫，而不需承擔損毀資料的風險。由於 Worker 使用不同的 SQLConnection 實體，因此，每個 Worker 會存取個別交易中的資料庫。同時進行資料處理作業並不會影響資料的完整性。

請參閱

第 615 頁「在 AIR 中使用本機 SQL 資料庫」

flash.concurrent 套件

第 64 章 安全性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

安全性是 Adobe、使用者、網站擁有者，以及內容開發人員最關切的議題，因此，Adobe® Flash® Player 和 Adobe® AIR™ 中包含一組安全性規則和控制項，為使用者、網站擁有者和內容開發人員確保安全性。此討論涵蓋的安全性模型，是針對 ActionScript 3.0 發行、執行於 Flash Player 9.0.124.0 (或更新版本) 的 SWF 檔案，以及執行於 AIR 1.0 (或更新版本) 中執行的 SWF、HTML 和 JavaScript 檔案 (除非另外註明)。

本討論只提供安全性概觀，而並不是要全面涵蓋說明所有實作細節、使用情況，或使用某些 API 所衍生的後果。如需 Flash Player 安全性概念更深入的探討，請參閱 Flash Player 開發人員中心主題 「安全性」，網址為：www.adobe.com/go/devnet_security_en。

Flash 平台安全性概觀

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 和 AIR 執行階段使用的安全性模型，是以載入之 SWF 檔案、HTML、媒體以及其他資源的來源網域為基礎。來自特定網際網路網域 (如 www.example.com) 的可執行程式碼，永遠都可以存取該網域的所有資料，這些資源是放在相同的安全性群組之中，稱為 「安全執行程序」 (如需詳細資訊，請參閱第 901 頁 「安全執行程序」)。

例如，SWF 檔案中的 ActionScript 程式碼，可以載入 SWF 檔、點陣圖、音效、文字檔，以及來自其本身網域的任何其它資源，而且只要兩個檔案都是使用 ActionScript 3.0 撰寫，永遠都允許來自相同網域的兩個 檔之間的跨指令碼處理。「跨指令碼處理」是指檔案中的程式碼具備另一個檔案的屬性、方法和物件的存取能力。

但是在使用 ActionScript 3.0 撰寫的 SWF 檔與使用舊版 ActionScript 撰寫的 SWF 檔之間並不支援跨指令碼處理，不過這兩種檔案可以使用 LocalConnection 類別進行通訊。而且根據預設，禁止使用 SWF 檔跨指令碼處理來自其他網域的 ActionScript 3.0 SWF 檔以及從其他網域載入資料的功能，但是呼叫被載入 SWF 檔中的 Security.allowDomain() 方法即可授與此存取功能。如需詳細資訊，請參閱第 917 頁 「跨指令碼處理」。

根據預設，下列基本安全性原則永遠都適用：

- 相同安全性執行程序中的資源永遠都可以彼此存取。
- 遠端安全執行程序中的可執行程式碼，永遠都不能存取本機檔案和資料。

Flash Player 和 AIR 執行階段會將以下各項視為個別的網域，然後分別為每個網域設定安全執行程序：

- http://example.com
- http://www.example.com
- http://store.example.com
- https://www.example.com
- http://192.0.34.166

即使具名的網域 (如 http://example.com) 對應於特定 IP 位址 (如 http://192.0.34.166)，執行階段會分別為兩者設置不同的安全執行程序。

開發人員有兩種基本方式可以用來授與 SWF 檔權限，以存取來自與 SWF 檔不同安全執行程序的資源：

- Security.allowDomain() 方法 (請參閱第 910 頁 「作者 (開發人員) 控制」)
- URL 原則檔 (請參閱第 907 頁 「網站控制 (原則檔)」)

在 Flash Player 和 AIR 執行階段安全性模型中，載入內容與擷取或存取資料是有區別的。「內容」是定義為媒體，包括執行階段可以顯示的視覺媒體、音效、視訊或包含顯示媒體的 SWF 檔案或 HTML。「資料」則是定義為只能由 程式碼存取的項目。內容和資料是以不同的方式載入。

- 載入內容 — 您可以使用類別 (例如 Loader、Sound 以及 NetStream 類別) 來載入內容；使用 Flex 時透過 MXML 標籤；或者在 AIR 應用程式中透過 HTML 標籤。
- 擷取資料：您可以使用 Bitmap 物件、BitmapData.draw() 方法和 BitmapData.drawWithQuality() 方法、Sound.id3 屬性，或 SoundMixer.computeSpectrum() 方法，從載入的媒體內容中擷取資料。drawWithQuality 方法可以在 Flash Player 11.3 及更新版本，以及 AIR 3.3 及更新版本中取得。
- 存取資料：您可以使用類別 (例如 URLStream、URLLoader、FileReference、Socket 和 XMLSocket 等類別)，從外部檔案 (如 XML 檔) 載入，直接存取資料。AIR 會產生額外的類別，用於載入資料，例如 FileStream 和 XMLHttpRequest。

Flash Player 安全性模型為載入內容與存取資料定義不同的規則。一般來說，載入內容的限制比存取資料的限制少。

一般而言，您可以從任何位置載入內容 (SWF 檔、點陣圖、MP3 檔和視訊)，但是如果內容是來自與所載入之程式碼或內容不同的網域時，將會置入不同的安全執行程序中進行分割處理。

載入內容作業只包含少數幾道關卡：

- 根據預設，本機 SWF 檔 (由使用者的硬碟之類非網路位址載入的檔案) 會在具有系統檔案的本機安全執行程序中加以分類，這些檔案無法從網路載入內容。如需詳細資訊，請參閱第 902 頁 「本機安全執行程序」。
- 「即時訊息通訊協定」 (Real-Time Messaging Protocol，RTMP) 伺服器可以限制內容的存取權限。如需詳細資訊，請參閱第 916 頁 「使用 RTMP 伺服器所傳遞的內容」。

如果載入的媒體是影像、音效或視訊，其資料 (如像素資料和聲音資料) 只有在媒體原始網域的 URL 原則檔中已經包含了該 SWF 檔的網域，才能由其安全執行程序以外的 SWF 檔來存取。如需詳細資訊，請參閱第 919 頁 「將載入的媒體當做資料加以存取」。

其它形式的載入資料包括文字或 XML 檔，都是用 URLLoader 物件載入。同樣地，在這種情況下，若要從另一個安全執行程序存取任何資料，必須透過原始網域的 URL 原則檔授與權限。如需詳細資訊，請參閱第 921 頁 「使用 URLLoader 和 URLStream」。

備註：至於在 AIR 應用程式安全執行程序中執行的程式碼，永遠不需要原則檔，便能載入遠端內容或資料。

安全執行程序

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

用戶端電腦可以取得個別檔案，可能包含程式碼、內容以及來自一些來源的資料，這些來源像是外部網站、本機檔案系統，或者安裝的 AIR 應用程式。Flash Player 和 AIR 執行階段會在各個程式碼檔案和其它資源 (如共享物件、點陣圖、聲音、視訊和資料檔案) 載入 Flash Player 時，根據其來源分別指定至安全執行程序。下列各節將說明由執行階段強制執行的規則，以控制在指定的安全執行程序中執行的程式碼或內容可以存取的項目。

如需有關 Flash Player 安全性的詳細資訊，請參閱 Flash Player 開發人員中心主題 「安全性」，網址為：www.adobe.com/go/devnet_security_en。

遠端安全執行程序

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 和 AIR 執行階段在與來源網域對應的個別安全執行程序中，會為來自網際網路資源進行分類 (包括 SWF 檔案)。例如，從 example.com 載入的資源會放到不同的安全執行程序中，與 foo.org 載入的資源區隔開來。根據預設，這些檔案都獲得授權，可自本身的伺服器存取任何資源。遠端 SWF 檔可透過明確網站及作者權限 (如 URL 原則檔及 Security.allowDomain() 方法)，允許自其它網域存取額外的資料。如需詳細資訊，請參閱第 907 頁 「網站控制 (原則檔)」、第 910 頁 「作者 (開發人員) 控制」。

遠端 SWF 檔不能載入任何本機檔案或資源。

如需有關 Flash Player 安全性的詳細資訊，請參閱 Flash Player 開發人員中心主題 「安全性」，網址為：www.adobe.com/go/devnet_security_en。

本機安全執行程序

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

「本機檔案」代表任何使用 file: 通訊協定或通用命名慣例 (UNC) 路徑參考的任何檔案。本機 SWF 檔是置入四個本機安全執行程序的其中一個：

- 具有檔案系統的本機安全執行程序 — 基於安全用途，Flash Player 和 AIR 執行階段會將所有本機檔案放到具有檔案系統的本機安全執行程序中 (預設值)。可執行程式碼可以從此安全執行程序中讀取本機檔案 (例如，藉由使用 URLLoader 類別)，但可能無法透過網路進行任何通訊。這麼做可以確保使用者無法將本機資料洩露至網路，或不當地與其他人分享。
- 具有網路連線的本機安全執行程序：編譯 SWF 檔時，您可以指定讓它在以本機檔案方式執行時具有網路存取權限 (請參閱第 904 頁 「設定本機 SWF 檔的安全執行程序類型」)。這些檔案會放在具有網路連線的本機安全執行程序中。指定到 「具有網路連線的本機安全執行程序」的 SWF 檔會喪失存取本機檔案的權限，但這些 SWF 檔卻因而可自網路存取資料。不過，具有網路的本機 SWF 檔還是無法讀取任何網路衍生的資料，除非透過 URL 原則檔或呼叫 Security.allowDomain() 方法，取得執行該動作的權限。若要授與這種權限，則必須使用 <allow-access-from domain="*"/> 或 Security.allowDomain("*")，讓 URL 原則檔授與 「所有」網域權限。如需詳細資訊，請參閱第 907 頁 「網站控制 (原則檔)」、第 910 頁 「作者 (開發人員) 控制」。
- 本機信任的安全執行程序：註冊為受 (使用者或安裝程式應用程式) 信任的本機 SWF 檔會放在本機信任的安全執行程序中。系統管理員和使用者也可以根據安全性考量，重新指定 (移動) 本機 SWF 檔進出本機信任的安全執行程序 (請參閱第 905 頁 「系統管理員控制」和第 906 頁 「使用者控制」)。指定給本機信任的安全執行程序之 SWF 檔可以與任何其它 SWF 檔互動，並從任何一處 (遠端或本機) 載入資料。
- AIR 應用程式執行程序：此執行程序包含以執行中 AIR 應用程式安裝的內容。根據預設，在 AIR 應用程式安全執行程序中執行的程式碼可以從任何網域跨指令碼處理任何檔案。但是，AIR 應用程式安全執行程序之外的檔案，則不允許在應用程式安全執行程序中進行跨指令碼程式碼處理。根據預設，在 AIR 應用程式安全執行程序中的程式碼和內容，可以從任何網域載入內容和資料。

嚴格禁止具有網路的本機與具有檔案系統的本機安全執行程序之間，以及具有檔案系統的本機安全執行程序與遠端安全執行之間的通訊。無論是在 Flash 中執行的應用程式，或使用者或系統管理員執行的應用程式，都無法授與權限允許上述通訊。

以指令碼處理本機 HTML 檔與本機 SWF 檔之間的往返通訊 (例如，使用 ExternalInterface 類別) 需要 HTML 檔和 SWF 檔雙方同時都在本機信任的安全執行程序中。這是因為瀏覽器的本機安全性模型與 Flash Player 本機安全性模型不同的緣故。

位於具有網路的本機安全執行程序中之 SWF 檔無法載入具有檔案系統的本機安全執行程序中之 SWF 檔。位於具有檔案系統的本機安全執行程序中之 SWF 檔無法載入具有網路的本機安全執行程序中之 SWF 檔。

AIR 應用程式執行程序

Adobe AIR 1.0 以及更新的版本

Adobe AIR 執行階段會將額外的應用程式安全執行程序，新增至 Flash Player 安全執行程序模型。安裝後變成 AIR 應用程式一部分的檔案，會載入至應用程式安全執行程序中。應用程式載入的任何其他檔案，它們符合一般 Flash Player 安全性模型指定的安全限制。

安裝應用程式時，AIR 套件中包含的所有檔案都會安裝到使用者電腦的應用程式目錄。開發人員可以透過 app:/ URL 配置，在程式碼中參考此目錄 (請參閱 第 702 頁 「URI 配置」)。應用程式目錄樹狀結構之中的所有檔案會在執行應用程式時，指定給應用程式安全執行程序。應用程式安全執行程序中的內容享有 AIR 應用程式可用的完整權限，包括與本機檔案系統互動。

許多 AIR 應用程式只使用這些安裝於本機的檔案來執行應用程式。但是，AIR 應用程式並不是只限於應用程式目錄中的檔案，而是可以從任何來源載入任何類型的檔案。其中包括使用者電腦的本機檔案，以及可用外部來源的檔案 (例如在區域網路或網際網路上的檔案)。檔案類型對於安全性限制沒有任何影響，從相同來源載入的 HTML 檔案與 SWF 檔案具有相同的安全性權限。

應用程式安全執行程序中的內容可以存取其它安全執行程序中內容不得使用的 AIR API。例如，air.NativeApplication.nativeApplication.applicationDescriptor 屬性會傳回應用程式的應用程式描述器檔案內容，但這個屬性僅供應用程式安全執行程序中的內容使用。另一個有限制 API 的範例是 FileStream 類別，此類別包含讀取及寫入本機檔案系統的方法。

在 「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考」中，僅供應用程式安全執行程序內容使用的 ActionScript API 會以 AIR 標誌標註。在其它安全執行程序中使用這些 API 會導致執行階段擲出 SecurityError 例外。

對於 HTML 內容 (在 HTMLLoader 物件中)，所有 AIR JavaScript API (由 window.runtime 屬性所提供，或是在使用 AIRAliases.js 檔案時由 air 物件所提供) 都可供應用程式安全執行程序內容使用。其它安全執行程序的 HTML 內容不能存取 window.runtime 屬性，所以這項內容不能存取 AIR 或 Flash Player API。

在 AIR 應用程式安全執行程序中執行的內容，存在以下額外的限制：

- 對於應用程式安全執行程序中的 HTML 內容，可在載入程式碼後以動態方式將字串轉換成可執行程式碼的 API 具備使用限制。其目的在於防止應用程式不小心從非應用程式來源 (例如可能不安全的網路網域) 插入 (並執行) 程式碼。其中一個範例是使用 eval() 函數。如需詳細資訊，請參閱第 933 頁 「不同安全執行程序中內容的程式碼限制」。
- 為了防止可能的網路釣魚攻擊，ActionScript TextField 物件中 HTML 內容裡的 img 標籤會在應用程式安全執行程序中的 SWF 內容裡加以忽略。
- 應用程式安全執行程序中的內容無法在 ActionScript 2.0 文字欄位的 HTML 內容中使用 asfunction 通訊協定。
- 應用程式安全執行程序中的 SWF 內容無法使用跨網域快取，這是 Flash Player 9 更新 3 新增的功能。這項功能可以讓 Flash Player 持續地快取 Adobe 平台組件內容，並依需求在載入的 SWF 內容中重複使用 (不需要多次重新載入內容)。

AIR 中 JavaScript 的限制

Adobe AIR 1.0 以及更新的版本

非應用程式安全執行程序中的 JavaScript 內容與應用程式安全執行程序中的內容不同，此 JavaScript 內容 「可以」隨時呼叫 eval() 函數，執行以動態方式產生的程式碼。不過，在 AIR 中，在非應用程式安全執行程序中執行的 JavaScript 會有一些限制。包括：

- 非應用程式安全執行程序中的 JavaScript 程式碼不能存取 window.runtime 物件，因此這種程式碼無法執行 AIR API。
- 根據預設，非應用程式安全執行程序內容不能使用 XMLHttpRequest 呼叫，從呼叫要求的網域之外的其它網域來載入資料。但是，應用程式程式碼可以透過在包含的 frame 或 iframe 中設定 allowCrossdomainXHR 特質，授予非應用程式內容權限來執行此作業。如需詳細資訊，請參閱第 933 頁 「不同安全執行程序中內容的程式碼限制」。

- 呼叫 JavaScript window.open() 方法有其限制。如需詳細資訊，請參閱第 936 頁 「呼叫 JavaScript window.open() 方法的限制」。
- 遠端 (網路) 安全執行程序中的 HTML 內容只能從 (網路 URL 的) 遠端網域載入 CSS、frame、iframe 和 img 內容。
- 具有檔案系統的本機、具有網路連線的本機，或本機信任安全執行程序中的 HTML 內容只能從本機安全執行程序 (而不是從應用程式或網路 URL) 載入 CSS、frame、iframe 和 img 內容。

如需詳細資訊，請參閱第 933 頁 「不同安全執行程序中內容的程式碼限制」。

設定本機 SWF 檔的安全執行程序類型

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

電腦的使用者或系統管理員可以指定本機 SWF 檔為受信任，允許它從所有網域 (包括本機和網路) 載入資料。這是在 「全域 Flash Player Trust」和 「使用者 Flash Player Trust」目錄中指定。如需詳細資訊，請參閱第 905 頁 「系統管理員控制」和第 906 頁 「使用者控制」。

如需有關本機安全執行程序的詳細資訊，請參閱第 902 頁 「本機安全執行程序」。

Adobe Flash Professional

您可以在編寫工具中設定文件的發佈設定，以設定 「具有檔案系統的本機安全執行程序」或 「具有網路連線的本機安全執行程序」的 SWF 檔。

Adobe Flex

您可以在 Adobe Flex 編譯器中設定 use-network 標籤，以設定 「具有檔案系統的本機安全執行程序」或 「具有網路連線的本機安全執行程序」的 SWF 檔。如需詳細資訊，請參閱 「建置及部署 Adobe Flex 3 應用程式」中的 「關於應用程式編譯器選擇」。

Security.sandboxType 屬性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

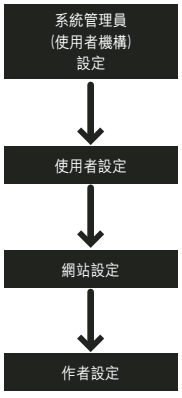
SWF 檔的作者可以使用唯讀的靜態 Security.sandboxType 屬性，判斷 Flash Player 或 AIR 執行階段指定給 SWF 檔的安全執行程序類型。Security 類別包括代表 Security.sandboxType 屬性可能值的常數，如下所示：

- Security.REMOTE：SWF 檔是來自網際網路 URL，而在網域架構的安全執行程序規則下運作。
- Security.LOCAL_WITH_FILE：SWF 檔是本機檔案，但沒有受到使用者信任，而且不是使用網路指派來發佈。該 SWF 檔可以從本機資料來源讀取，但無法與網際網路進行通訊。
- Security.LOCAL_WITH_NETWORK：SWF 檔是本機檔案，沒有受到使用者信任，但是使用網路指派來發佈。SWF 檔可以與網際網路進行通訊，但無法從本機資料來源讀取。
- Security.LOCAL_TRUSTED：SWF 檔是本機檔案，而且使用者透過 「設定管理員」或 Flash Player Trust 組態設定檔，決定加以信任。該 SWF 檔可從本機資料來源讀取，並與網際網路通訊。
- Security.APPLICATION：這個檔案是在 AIR 應用程式中執行，而且它是使用該應用程式的套件 (AIR 檔) 來安裝。根據預設，在 AIR 應用程式安全執行程序中的檔案可以從任何網域跨指令碼處理任何檔案。但是，AIR 應用程式安全執行程序之外的檔案則不允許對 AIR 檔案進行跨指令碼處理。根據預設，在 AIR 應用程式安全執行程序中的檔案可以從任何網域載入內容和資料。

權限控制

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 用戶端執行階段安全性模型是以資源為中心進行設計，這些資源包括 SWF 檔、本機資料和網際網路 URL 等物件。「關係人」包括擁有或使用這些資源的各方人士。關係人可以對本身擁有的資源執行控制 (安全性設定)，而且每一個資源都有四種關係人。Flash Player 會嚴格執行授權階層架構進行這些控制，如下列圖例說明所示：



安全性控制的階層架構

也就是說，如果系統管理員限制存取一項資源，就沒有其他任何關係人能夠覆寫該限制。

至於 AIR 應用程式，這些權限控制項只會套用至 AIR 應用程式安全執行程序外執行的內容。

系統管理員控制

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

電腦的系統管理使用者 (以系統管理權限登入的使用者) 可以套用影響電腦所有使用者的 Flash Player 安全性設定。在非企業環境中 (例如在家用電腦上)，通常只有一個使用者，該使用者同時也具有系統管理存取權限；甚至在企業環境中，也有個人使用者具有電腦的系統管理權限。

系統管理使用者控制總共有兩種類型：

- mms.cfg 檔
- 全域 Flash Player Trust 目錄

mms.cfg 檔

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

mms.cfg 檔是文字檔，可以讓系統管理員啟用或限制存取各種不同的功能。當 Flash Player 啟動時，會從此檔案讀取其安全性設定，然後使用這些設定來限制功能。mms.cfg 檔包含系統管理員用來管理各項功能 (例如隱私權控制、本機檔案安全性、通訊端連線等等) 的設定。

SWF 檔可以存取有關經由呼叫 Capabilities.avHardwareDisable 和 Capabilities.localFileReadDisable 屬性所停用功能的一些資訊，但是 mms.cfg 檔中大部分設定都不能透過 ActionScript 進行查詢。

為了要在電腦上強制執行與獨立於應用程式之外的安全性和隱私權原則，mms.cfg 檔只能由系統管理員進行修改；mms.cfg 檔並不是由應用程式的安裝程式來使用。雖然以系統管理權限執行的安裝程式可以修改 mms.cfg 檔的內容，但是 Adobe 將這種用法視為違反使用者的信任，並強烈要求安裝程式建立者絕對不要修改 mms.cfg 檔。

mms.cfg 檔的儲存位置如下：

- Windows：system\Macromed\Flash\mms.cfg

  (例如，C:\WINDOWS\system32\Macromed\Flash\mms.cfg)

- Mac：app support/Macromedia/mms.cfg

  (例如，/Library/Application Support/Macromedia/mms.cfg)

如需有關 mms.cfg 檔的詳細資訊，請參閱 「Flash Player 管理指南」，網址為：
www.adobe.com/go/flash_player_admin_tw。

全域 Flash Player Trust 目錄

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

系統管理使用者和安裝程式應用程式可以將指定的本機 SWF 檔註冊為受所有使用者信任，這些 SWF 檔是指定給本機信任的安全執行程序。它們可以與其它任何 SWF 檔互動，而且可以從遠端或本機的任何一處載入資料。檔案是在下列位置指定為全域 Flash Player Trust 目錄中受信任的檔案：

- Windows：system\Macromed\Flash\FlashPlayerTrust

  (例如，C:\WINDOWS\system32\Macromed\Flash\FlashPlayerTrust)

- Mac：app support/Macromedia/FlashPlayerTrust

  (例如，/Library/Application Support/Macromedia/FlashPlayerTrust)

Flash Player Trust 目錄可以包含任何數目的文字檔，每一個檔案都列出受信任的路徑，一行列出一個路徑。各個路徑可以是個別的 SWF 檔、HTML 檔或目錄。註解行會以 # 符號做為開頭。例如，Flash Player 信任包含下列文字的設定檔，這些文字會授予指定之目錄及其所有子目錄中全部檔案受信任狀態：

```
# Trust files in the following directories:
C:\Documents and Settings\All Users\Documents\SampleApp
```

在信任設定檔中所列的路徑必須永遠都是本機路徑或 SMB 網路路徑。信任設定檔中的任何 HTTP 路徑都會加以忽略，只有本機檔案可以受信任。

為了避免衝突，請給予每一個信任設定檔對應於安裝應用程式的檔名，並使用 .cfg 做為副檔名。

當開發人員透過安裝程式應用程式散發本機執行的 SWF 檔時，您可以讓安裝程式應用程式將設定檔加入至 「全域 Flash Player Trust」目錄，將完整的權限授與您所要散發的檔案。安裝程式應用程式必須由具有系統管理權限的使用者執行。「全域 Flash Player Trust」目錄與 mms.cfg 檔不同，包含此目錄的目的在於讓安裝程式應用程式授與信任權限。系統管理使用者和安裝程式應用程式都可以使用 「全域 Flash Player Trust」目錄，指定受信任的本機應用程式。

個別使用者也可以使用 Flash Player Trust 目錄 (請參閱第 906 頁 「使用者控制」)。

使用者控制

Flash Player 9 和更新的版本

Flash Player 提供三種不同的使用者層級設定權限機制：「設定 UI」、「設定管理員」和 「使用者 Flash Player Trust」目錄。

設定 UI 和設定管理員

Flash Player 9 和更新的版本

「設定 UI」是為特定網域所設定的快速、互動式機制；「設定管理員」則提供更詳細的介面，並提供功能可進行全域變更，以影響許多或所有網域的權限。此外，當 SWF 檔提出新的權限要求時，需要有關安全性或隱私權的執行階段決策，就會顯示對話方塊，使用者可以在其中調整一些 Flash Player 設定。

「設定管理員」和 「設定 UI」都能提供與安全性有關的選項 (例如攝影機和麥克風設定、共享物件儲存設定，以及與舊版內容相關的設定等等)。AIR 應用程式無法使用 「設定管理員」或 「設定 UI」。

備註：mms.cfg 檔中的任何設定 (請參閱第 905 頁 「系統管理員控制」) 都不會反映在 「設定管理員」中。

如需有關 「設定管理員」的詳細資訊，請參閱 www.adobe.com/go/settingsmanager_tw。

使用者 Flash Player Trust 目錄

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

使用者和安裝程式應用程式可以將指定的本機 SWF 檔註冊為受信任，這些 SWF 檔是指定給本機信任的安全執行程序。它們可以與其它任何 SWF 檔互動，而且可以從遠端或本機的任何一處載入資料。使用者在 「使用者 Flash Player Trust」目錄中指定檔案為受信任，此目錄與共享物件儲存區位於相同目錄中，而位於下列位置 (視目前的使用者而定，會有特定的位置)：

- Windows：app data\Macromedia\Flash Player\#Security\FlashPlayerTrust

  (例如，C:\Documents and Settings\JohnD\Application Data\Macromedia\Flash Player\#Security\FlashPlayerTrust on Windows XP or C:\Users\JohnD\AppData\Roaming\Macromedia\Flash Player\#Security\FlashPlayerTrust on Windows Vista)

  在 Windows 中，會根據預設隱藏 Application Data 資料夾。若要顯示隱藏的資料夾和檔案，請選取「我的電腦」以開啟 Windows 檔案總管，然後選取 「工具 > 資料夾選項」，再選取 「檢視」索引標籤。在 「檢視」索引標籤下，選取 「顯示隱藏的檔案和資料夾」選項按鈕。

- Mac：app data/Macromedia/Flash Player/#Security/FlashPlayerTrust

  (例如，/Users/JohnD/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust)

  這些設定只會影響目前的使用者，而不影響登入電腦的其他使用者。如果沒有系統管理權限的使用者在本身系統內安裝了應用程式，「使用者 Flash Player Trust」目錄會讓安裝程式為該使用者將應用程式註冊為受信任。

  當開發人員透過安裝程式應用程式散發本機執行的 SWF 檔，您可以讓安裝程式應用程式將設定檔加入至 「使用者 Flash Player Trust」目錄，將完整的權限授與您所要散發的檔案。即使在這種狀況下，「使用者 Flash Player Trust」目錄檔案仍然是視為使用者控制項，因為是由使用者的動作 (安裝) 加以啟動。

  另外也有 「全域 Flash Player Trust」目錄，是由系統管理使用者或安裝程式為電腦所有使用者註冊應用程式 (請參閱第 905 頁 「系統管理員控制」)。

網站控制 (原則檔)

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要提供您網站伺服器的資料給另一個網域中的 SWF 檔使用，可以在伺服器中建立原則檔。「原則檔」是放置於您伺服器上特定位置中的 XML 檔。

原則檔會影響一些資源的存取，包括下列項目：

- 點陣圖、聲音和視訊中的資料
- 載入 XML 和文字檔

- 從其它安全網域將 SWF 檔匯入載入端 SWF 檔的安全網域
- 存取通訊端和 XML 通訊端連線

ActionScript 物件會實體化兩種不同的伺服器連線：文件架構伺服器連線和通訊端連線。ActionScript 物件 (如 Loader、Sound、URLLoader 和 URLStream) 會實體化文件架構伺服器連線，這些物件都會從 URL 載入檔案；ActionScript Socket 和 XMLSocket 物件會建立通訊端連線，這是以串流資料而不是以載入的文件運作。

由於 Flash Player 支援兩種伺服器連線，因此有兩種原則檔：URL 原則檔和通訊端原則檔。

- 文件架構連線需要 「URL 原則檔」。這些檔案可以讓伺服器指出其資料和文件可供 SWF 檔使用，從特定網域或所有網域提供服務。
- 通訊端連線需要 「通訊端原則檔」，以便使用 Socket 和 XMLSocket 類別，直接在較低的 TCP 通訊端層級啟用網路作業。

Flash Player 需要使用與嘗試連線所要使用的相同通訊協定來傳輸原則檔。例如，將原則檔置於 HTTP 伺服器中時，來自其它網域的 SWF 檔就允許以它為 HTTP 伺服器載入資料，但是，如果不在相同伺服器上提供通訊端原則檔，就必須禁止來自其它網域的 SWF 檔，在通訊端層級連線至伺服器。換句話說，擷取原則檔的工具必須與連線的工具相符。

原則檔用法和語法會在本節中其餘內容中扼要討論，因為這些原則檔適用於為 Flash Player 10 發佈的 SWF 檔 (原則檔實作與舊版 Flash Player 稍有不同，因為後續的版本已加強了 Flash Player 的安全性)。如需有關原則檔的詳細資訊，請參閱 Flash Player 開發人員中心主題 「Flash Player 9 中的原則檔變更」，網址為：www.adobe.com/go/devnet_security_en。

在 AIR 應用程式安全執行程序裡面執行的程式碼，不需要任何原則檔便可以從 URL 或通訊端存取資料。在 AIR 應用程式中以非應用程式安全執行程序執行的程式碼，則需要原則檔。

主原則檔

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

根據預設，Flash Player (以及 AIR 應用程式安全執行程序以外的 AIR 內容) 會先在伺服器的根目錄尋找名為 crossdomain.xml 的 URL 原則檔，然後再於通訊埠 843 尋找通訊埠原則檔。這兩個位置中的檔案稱為 「主原則檔」 (在通訊端連線的情況下，Flash Player 也會在與主連線相同的連接埠上尋找通訊端原則檔，但是在該連接埠上找到的原則檔並不視為主原則檔)。

除了指定存取權限以外，主原則檔也包含 「中繼原則」陳述式。中繼原則會指定可以包含原則檔的位置。URL 原則檔的預設中繼原則是 「僅主檔」，也就是說，/crossdomain.xml 是伺服器上唯一允許的原則檔。通訊端原則檔的預設中繼原則是 「全部」，也就是說，主機上的任何通訊端都可以提供通訊端原則檔。

備註：在 Flash Player 9 及其舊版中，URL 原則檔的預設中繼原則是 「全部」，也就是說，任何目錄都可以包含原則檔。如果您已部署從預設 /crossdomain.xml 檔以外位置載入原則檔的應用程式，而這些應用程式現在可能是在 Flash Player 10 中執行，您 (或伺服器管理員) 務必要修改主原則檔以允許其他原則檔。如需有關如何指定不同中繼原則的詳細資訊，請參閱 Flash Player 開發人員中心主題 「Flash Player 9 中的原則檔變更」，網址為：www.adobe.com/go/devnet_security_en。

SWF 檔可以呼叫 Security.loadPolicyFile() 方法，檢查不同的原則檔案名稱和不同的目錄位置。但是，如果主原則檔不指定目標位置可提供原則檔，則呼叫 loadPolicyFile() 沒有作用，即使該位置上有原則檔也無效。先呼叫 loadPolicyFile()，再嘗試任何需要原則檔的網路作業。Flash Player 會自動在佇列中將網路要求排在相對應原則檔嘗試之後。因此，舉例來說，在起始網路作業之前立即呼叫 Security.loadPolicyFile() 是可接受的。

檢查主原則檔時，Flash Player 會等候三秒，等候伺服器回應。如果沒有接到回應，Flash Player 就會認定沒有主原則檔。但是，沒有 loadPolicyFile() 呼叫的預設逾時值，Flash Player 會假設被呼叫的檔案是存在的，並且要多久等多久，以便載入。因此，如果您要確定已載入主原則檔，請使用 loadPolicyFile() 明確地呼叫。

即使方法是命名為 Security.loadPolicyFile()，原則檔要等到發出需要原則檔的網路呼叫之後才會載入。呼叫 loadPolicyFile() 只是告知 Flash Player 在需要時到何處尋找原則檔。

您無法接到已起始或已完成原則檔要求的通知，也沒有必要這麼做。Flash Player 會執行非同步原則檢查，並自動等候到已成功查知有原則檔後才起始連線。

下節內容所包含的資訊，僅適用於 URL 原則檔。如需有關通訊端原則檔的詳細資訊，請參閱第 922 頁「連線至通訊端」。

URL 原則檔範圍

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

URL 原則檔僅套用於載入的目錄及其子目錄。根目錄中的原則檔會套用至整個伺服器，但從任意子目錄載入的原則檔則僅套用於該目錄及其子目錄。

原則檔只影響它所在的特定伺服器。例如，位於 https://www.adobe.com:8080/crossdomain.xml 的原則檔案，只適用於在連接埠 8080 上透過 HTTPS 對 www.adobe.com 進行的資料載入呼叫。

在 URL 原則檔中指定存取權限

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

原則檔中包含單獨一個 <cross-domain-policy> 標籤，其中再包含零個或多個 <allow-access-from> 標籤。每一個 <allow-access-from> 標籤中都包含一個特質，即 domain，它指定了確實的 IP 位址、確實的網域，或者萬用字元網域 (任何網域)。萬用字元網域會以下列兩種方式的其中一種指示：

- 使用單一星號 (*)，與所有網域和所有 IP 位址相符
- 使用星號之後再加字尾，只比對找出以指定字尾結束的網域

字尾必須用一點為開頭。然而，有字尾的萬用字元網域可比對找出只含有該字尾而沒有最前端一點的網域。例如，xyz.com 被視為屬於 *.xyz.com 的一部分。在 IP 網域規格中不允許萬用字元。

下面是一個 URL 原則檔的範例，允許存取源自 *.example.com、www.friendOfExample.com 和 192.0.34.166 的 SWF 檔：

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="*.example.com" />
    <allow-access-from domain="www.friendOfExample.com" />
    <allow-access-from domain="192.0.34.166" />
</cross-domain-policy>
```

如果指定了一個 IP 位址，則只允許使用 IP 語法 (例如 http://65.57.83.12/flashmovie.swf) 從該 IP 位址載入的 SWF 檔進行存取，使用網域名稱語法載入的 SWF 檔則無法存取。Flash Player 不會執行 DNS 解析。

您可以允許存取源自任何網域的文件，如下列範例所示：

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
<allow-access-from domain="*" />
</cross-domain-policy>
```

每個 <allow-access-from> 標籤也都有選擇性的 secure 特質，此特質預設為 true。如果原則檔是位於 HTTPS 伺服器上，而您想要讓位於非 HTTP 伺服器上的 SWF 檔可以透過 HTTPS 伺服器載入資料，則您可以將此特質設為 false。

將 secure 特質設為 false 有可能會破壞 HTTPS 所提供的安全性。尤其是此特質設定為 false，將會讓安全性內容門戶洞開而遭受窺探與攻擊。Adobe 強烈建議您不要將 secure 特質設定為 false。

如果要載入的資料位於 HTTPS 伺服器上，但載入端 SWF 檔位於 HTTP 伺服器上，Adobe 建議您將載入端 SWF 檔移到 HTTPS 伺服器上。如此一來，就可以讓您所有的安全資料都在 HTTPS 的保護之下。但是，如果您決定必須將載入端 SWF 檔放在 HTTP 伺服器中，請將 secure="false" 特質加入至 <allow-access-from> 標籤，如下列程式碼所示：

```
<allow-access-from domain="www.example.com" secure="false" />
```

另一個您可以用來准許存取的元素是 allow-http-request-headers-from 標籤。這個元素會授與用戶端權限，從另一個網域裝載內容，以傳送使用者定義的檔頭到您的網域。<allow-access-from> 標籤可以授與其他網域權限從您的網域拉出資料，而 allow-http-request-headers-from 標籤則是授與其他網域權限推入資料到您的網域，以檔頭的形式進行。在下列範例中，任何網域都准許傳送 SOAPAction 檔頭到目前的網域：

```
<cross-domain-policy>
    <allow-http-request-headers-from domain="*" headers="SOAPAction"/>
</cross-domain-policy>
```

如果 allow-http-request-headers-from 陳述式是在主原則檔中，它會套用於主機上的所有目錄。否則，僅套用於包含該陳述式之原則檔的目錄及其子目錄。

預先載入原則檔

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

從伺服器載入資料或連線到通訊端是非同步作業，Flash Player 會等候原則檔完成下載之後，再開始執行主要作業。但是從影像擷取像素資料，或是從聲音擷取樣本資料都是同步作業，必須載入原則檔，才能擷取資料。當您載入媒體時，請指定它檢查原則檔：

- 使用 Loader.load() 方法時，請設定 context 參數的 checkPolicyFile 屬性，此參數是 LoaderContext 物件。
- 使用 <img> 標籤，將影像嵌入文字欄位時，請將 <img> 標籤的 checkPolicyFile 特質設定為 "true"，如下所示：

  ```
  <img checkPolicyFile = "true" src = "example.jpg">
  ```

- 使用 Sound.load() 方法時，請設定 context 參數的 checkPolicyFile 屬性，此參數是 SoundLoaderContext 物件。
- 使用 NetStream 類別時，請設定 NetStream 物件的 checkPolicyFile 屬性。

當您設定上述其中一個參數時，Flash Player 會先檢查已為該網域下載之任何原則檔，然後再於伺服器上的預設位置尋找原則檔，檢查是否有 <allow-access-from> 陳述式和中繼原則存在。最後，它會考慮任何等候執行的 Security.loadPolicyFile() 方法呼叫，以查看是否在範圍中。

作者 (開發人員) 控制

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

用來授與安全性權限的主要 ActionScript API 是 Security.allowDomain() 方法，它會授與所指定網域中的 SWF 檔權限。在下列範例中，SWF 檔會授予來自 www.example.com 網域之 SWF 檔的存取權限：

```
Security.allowDomain("www.example.com")
```

此方法會授與下列項目權限：

- SWF 檔之間的跨指令碼處理 (請參閱第 917 頁 「跨指令碼處理」)
- 顯示清單存取權限 (請參閱第 919 頁 「在顯示清單中移動」)
- 事件偵測 (請參閱第 919 頁 「事件安全性」)
- Stage 物件之屬性和方法的完整存取權限 (請參閱：第 918 頁 「Stage 安全性」)

呼叫 Security.allowDomain() 方法的主要目的，是要授與位於外部網域的 SWF 檔存取權限，以編寫 SWF 檔的指令碼並呼叫 Security.allowDomain() 方法。如需詳細資訊，請參閱第 917 頁 「跨指令碼處理」。

指定 IP 位址為 Security.allowDomain() 方法的參數並不會允許源自指定之 IP 位址的各方進行存取；而是只允許在其 URL 中含有指定之 IP 位址 (不是對應至該 IP 位址的網域名稱) 的一方進行存取。例如，如果網域名稱 www.example.com 對應於 IP 位址 192.0.34.166，則呼叫 Security.allowDomain("192.0.34.166") 並不會授與存取 www.example.com 的權限。

您可以將 "*" 萬用字元傳遞至 Security.allowDomain() 方法，允許所有網域都可以進行存取。因為它會授與 「所有」網域的 SWF 檔權限，以編寫呼叫端 SWF 檔的指令碼，所以請謹慎使用 "*" 萬用字元。

ActionScript 包含第二個權限 API，稱為 Security.allowInsecureDomain()。此方法所執行的作業與 Security.allowDomain() 方法相同，不同的是透過安全 HTTPS 連線提供的 SWF 檔呼叫此方法時，它還能允許透過不安全通訊協定 (如 HTTP) 提供的其它 SWF 檔來存取呼叫端 SWF 檔。但是，在來自安全通訊協定 (HTTPS) 與來自不安全通訊協定 (如 HTTP) 的檔案之間允許指令碼處理並不是很好的安全性做法，這種做法可能會讓安全內容門戶洞開，而遭受窺探與攻擊。這類攻擊是以下列方式運作：由於 Security.allowInsecureDomain() 方法允許透過經由 HTTP 連線提供的 SWF 檔存取您的安全 HTTPS 資料，所以攻擊者只要插入您的 HTTP 伺服器與您的使用者之間，就能以他們自己的 SWF 檔取代您的 HTTP SWF 檔，然後存取您的 HTTPS 資料。

重要事項：在 AIR 應用程式安全執行程序中執行的程式碼，不允許呼叫 Security 類別的 allowDomain() 或 allowInsecureDomain() 方法。

另一個與安全性相關的重要方法是 Security.loadPolicyFile() 方法，它會讓 Flash Player 在非標準位置檢查原則檔。如需詳細資訊，請參閱第 907 頁 「網站控制 (原則檔)」。

限制網路 API

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

網路 API 可以用兩種方式加以限制。為了阻止惡意活動，封鎖了一般已保留連接埠的存取：您不能在程式碼中覆寫這些封鎖作業。若要控制 SWF 檔存取與其他連接埠相關的網路功能，您可以使用 allowNetworking 設定。

已封鎖的連接埠

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 和 Adobe AIR 會限制 HTTP 存取特定連接埠，與瀏覽器相同。在慣常供非 HTTP 類型伺服器使用的特定標準連接埠上不允許執行 HTTP 要求。

任何存取網路 URL 的 API 都會受到這些連接埠封鎖限制拘束。唯一例外的是直接呼叫通訊端的 API，例如 Socket.connect() 和 XMLSocket.connect()，或者呼叫 Security.loadPolicyFile()，會在其中載入通訊端原則檔。通訊端連線是透過使用目標伺服器上的通訊端原則來決定允許或拒絕。

以下列出適用於通訊埠封鎖的 ActionScript 3.0 API：

FileReference.download()、FileReference.upload()、Loader.load()、Loader.loadBytes()、navigateToURL()、NetConnection.call()、NetConnection.connect()、NetStream.play()、Security.loadPolicyFile()、sendToURL()、Sound.load()、URLLoader.load()、URLStream.load()

通訊埠封鎖也適用於 「共享元件庫」匯入作業、在文字欄位中使用 <img> 標籤，以及使用 <object> 和 <embed> 標籤載入 HTML 頁面中的 SWF 檔。

通訊埠封鎖也適用於在文字欄位中使用 <img> 標籤，以及使用 <object> 和 <embed> 標籤載入 HTML 頁面中的 SWF 檔。

下面清單列出已封鎖的連接埠：

HTTP：20 (ftp 資料)、21 (ftp 控制)

HTTP and FTP: 1 (tcpmux), 7 (echo), 9 (discard), 11 (systat), 13 (daytime), 15 (netstat), 17 (qotd), 19 (chargen), 22 (ssh), 23 (telnet), 25 (smtp), 37 (time), 42 (name), 43 (nicname), 53 (domain), 77 (priv-rjs), 79 (finger), 87 (ttylink), 95 (supdup), 101 (hostriame), 102 (iso-tsap), 103 (gppitnp), 104 (acr-nema), 109 (pop2), 110 (pop3), 111 (sunrpc), 113 (auth), 115 (sftp), 117 (uucp-path), 119 (nntp), 123 (ntp), 135 (loc-srv / epmap), 139 (netbios),

143 (imap2), 179 (bgp), 389 (ldap), 465 (smtp+ssl), 512 (print / exec), 513 (login), 514 (shell), 515 (printer), 526 (tempo), 530 (courier), 531 (chat), 532 (netnews), 540 (uucp), 556 (remotefs), 563 (nntp+ssl), 587 (smtp), 601 (syslog), 636 (ldap+ssl), 993 (ldap+ssl), 995 (pop3+ssl), 2049 (nfs), 4045 (lockd), 6000 (x11)

使用 allowNetworking 參數

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以在包含此 SWF 內容的 HTML 頁面中，設定其中 <object> 和 <embed> 標籤內的 allowNetworking 參數，控制 SWF 檔對網路功能的存取權。

allowNetworking 的可能值如下：

- "all" (預設值)：此 SWF 允許的所有網路 API。
- "internal"：此 SWF 檔可能不會呼叫瀏覽器瀏覽或瀏覽器互動 API (如本節下文中所示)，但是可以呼叫其它任何的網路 API。
- "none"：此 SWF 檔不可以呼叫瀏覽器瀏覽或瀏覽器互動 API (如本節下文中所示)，而且也不可以使用任何 SWF-to-SWF 通訊 API (亦如本節下文中所示)。

allowNetworking 參數是主要是針對在 SWF 檔與包圍的 HTML 頁面是來自不同網域時使用而設計的。不建議在所載入 SWF 檔與其包圍的 HTML 頁面位於相同網域時，使用 "internal" 或 "none" 等值，因為您不能確定 SWF 檔永遠都是使用您想用的 HTML 頁面載入。未受信任的各方可以從您的網域載入 SWF 檔而不加上包圍的 HTML，此時 allowNetworking 限制將無法依您原意運作。

呼叫禁止使用的 API 會擲出 SecurityError 例外。

加入 allowNetworking 參數，並在包含 SWF 檔參考的 HTML 頁面的 <object> 和 <embed> 標籤中設定其值，如下列範例所示：

```
<object classic="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    Code base="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,124,0"
    width="600" height="400" ID="test" align="middle">
<param name="allowNetworking" value="none" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowNetworking="none" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML 頁面也可以使用指令碼來產生內嵌 SWF 的標籤。您必須變更程式碼，在程式碼中插入適當的 allowNetworking 設定。Adobe Flash Professional 和 Adobe Flash Builder 產生的 HTML 頁面使用 AC_FL_RunContent() 功能，內嵌 SWF 檔的參考。將 allowNetworking 參數設定新增至指令碼，如下所示：

```
AC_FL_RunContent( ... "allowNetworking", "none", ...)
```

當 allowNetworking 設定成 "internal" 時，下列 API 將會停用：

navigateToURL()、fscommand()、ExternalInterface.call()

除了前述清單上的 API 以外，當 allowNetworking 設定為 "none" 時，也會停止使用下列 API：

sendToURL()、FileReference.download()、FileReference.upload()、Loader.load()、LocalConnection.connect()、LocalConnection.send()、NetConnection.connect()、NetStream.play()、Security.loadPolicyFile()、SharedObject.getLocal()、SharedObject.getRemote()、Socket.connect()、Sound.load()、URLLoader.load()、URLStream.load()、XMLSocket.connect()

即使選定的 allowNetworking 設定允許 SWF 檔使用網路 API，仍可能因安全執行程序限制而有其它限制 (請參閱第 901 頁「安全執行程序」)。

當 allowNetworking 設定成 "none" 時，您無法在 TextField 物件之 htmlText 屬性的 <img> 標籤中參考外部媒體 (會擲回 SecurityError 例外)。

當 allowNetworking 設定成 "none" 時，從匯入的共享元件庫加入 Flash Professional 中 (不是 ActionScript) 的元件會在執行階段遭到封鎖。

全螢幕模式安全性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 9.0.27.0 和更新版本都支援全螢幕模式，在此模式下 Flash Player 中執行的內容可以填滿整個螢幕。若要進入全螢幕模式，請將 Stage 的 displayState 屬性設定為 StageDisplayState.FULL_SCREEN 常數。如需詳細資訊，請參閱第 141 頁「使用全螢幕模式」。

對於在遠端安全執行程序中執行的 SWF 檔案，有一些安全性考量。

若要啟用全螢幕模式，請在包含此 SWF 檔參考的 HTML 網頁中，將 allowFullScreen 參數加入 <object> 和 <embed> 標籤中，並將其值設定為 "true" (預設值為 "false")，如下列範例所示：

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,18,0"
    width="600" height="400" id="test" align="middle">
<param name="allowFullScreen" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML 頁面也可以使用指令碼來產生內嵌 SWF 的標籤。您必須變更指令碼，以便在其中插入適當的 allowFullScreen 設定。Flash Professional 和 Flash Builder 所產生的 HTML 網頁會使用 AC_FL_RunContent() 函數內嵌 SWF 檔的參考，而且您必須加入 allowFullScreen 參數設定，如下所示：

```
AC_FL_RunContent( ... "allowFullScreen", "true", ...)
```

您只能呼叫啟動全螢幕模式的 ActionScript 來回應滑鼠事件或鍵盤事件。如果在其他情況下呼叫，Flash Player 會擲回例外。

當內容進入全螢幕模式時會出現訊息，指示使用者如何離開並返回一般模式。訊息會顯示幾秒鐘，然後淡出。

對於在瀏覽器中執行的內容，在全螢幕模式下使用鍵盤時會受到限制。在 Flash Player 9 中，只支援讓應用程式返回一般模式的鍵盤快速鍵，例如 Escape 鍵。使用者無法在文字欄位中輸入文字或在螢幕周圍瀏覽。在 Flash Player 10 和更新版本中，支援特定非列印用按鍵 (像是，方向鍵、空格鍵和 Tab 鍵)。但是仍然禁止文字輸入。

而在獨立式播放程式或放映檔中則永遠都允許全螢幕模式。此外，在這些環境中也完整支援使用鍵盤 (包括文字輸入)。

呼叫 Stage 物件的 displayState 屬性會擲出例外給跟 Stage 擁有者 (主要 SWF 檔) 位於不同安全執行程序的任何呼叫者。如需詳細資訊，請參閱：第 918 頁「Stage 安全性」。

系統管理員可以在 mms.cfg 檔中設定 FullScreenDisable = 1，為在瀏覽器中執行的 SWF 檔停用全螢幕模式。如需詳細資訊，請參閱第 905 頁「系統管理員控制」。

在瀏覽器中，SWF 檔必須包含於 HTML 網頁中才能允許全螢幕模式：

全螢幕互動模式安全性

Flash Player 11.3 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 11.3 和更新版本都支援全螢幕互動模式，在此模式下 Flash Player 中執行的內容可以填滿整個螢幕，並且「接受文字輸入」。若要進入全螢幕互動模式，請將 Stage 的 displayState 屬性設定為 StageDisplayState.FULL_SCREEN_INTERACTIVE 常數。如需詳細資訊，請參閱第 141 頁「使用全螢幕模式」。

對於在遠端安全執行程序中執行的 SWF 檔案，有一些安全性考量。

若要啟用全螢幕模式，請在包含此 SWF 檔案參考的 HTML 頁面中，將 allowFullScreenInteractive 參數加入 <object> 和 <embed> 標籤中，並將其值設定為 "true" (預設值為 "false")，如下列範例所示：

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,18,0"
    width="600" height="400" id="test" align="middle">
<param name="allowFullScreenInteractive" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML 頁面也可以使用指令碼來產生內嵌 SWF 的標籤。您必須變更指令碼，以便在其中插入適當的 allowFullScreenInteractive 設定。Flash Professional 和 Flash Builder 所產生的 HTML 頁面，會使用 AC_FL_RunContent() 函數內嵌 SWF 檔案的參考，因此您必須加入 allowFullScreenInteractive 參數設定，如下所示：

```
AC_FL_RunContent( ... "allowFullScreenInteractive", "true", ...)
```

您只能呼叫啟動全螢幕互動模式的 ActionScript 來回應滑鼠事件或鍵盤事件。如果在其他情況下呼叫，Flash Player 會擲回例外。

當內容進入全螢幕互動模式時，重疊訊息就會出現。此訊息會顯示全螢幕頁面的網域、結束全螢幕模式的指示，以及 **Allow** 按鈕。在使用者按一下 **Allow** 之前，重疊仍會存在，傳達使用者正處於全螢幕互動模式。

系統管理員可以在 mms.cfg 檔案中設定 FullScreenInteractiveDisable = 1，為在瀏覽器中執行的 SWF 檔案停用全螢幕互動模式。如需詳細資訊，請參閱第 905 頁「系統管理員控制」。

在瀏覽器中，SWF 檔案必須包含於 HTML 頁面中才能允許全螢幕互動模式。

載入內容

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 和 AIR 內容可以載入很多其他類型的內容，其中包括以下幾項：

- SWF 檔
- 影像
- Sound
- 視訊
- HTML 檔案 (僅限 AIR)
- JavaScript (僅限 AIR)

使用 Loader 類別來載入 SWF 檔案和影像

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 Loader 類別來載入 SWF 檔和影像 (JPG、GIF 或 PNG 檔案)。任何 SWF 檔 (在具有檔案系統的本機安全執行程序中之檔案除外) 都可以載入任何網路網域中的 SWF 檔和影像。只有位於本機安全執行程序中的 SWF 檔可以從本機檔案系統載入 SWF 檔和影像，但是位於具有網路的本機安全執行程序中之 SWF 檔則只能載入本機信任的和具有網路的本機安全執行程序中之 SWF 檔。位於具有網路的本機安全執行程序中之 SWF 檔可載入 SWF 檔以外的本機內容 (如影像)，但無法存取載入內容中的資料。

當您從不受信任的來源 (例如與 Loader 物件的根 SWF 檔所屬網域不同的網域) 載入 SWF 檔時，可能會想要定義 Loader 物件的遮色片以避免載入的內容 (Loader 物件的子系) 繪製至該遮色片外面的 「舞台」，如下列程式碼所示：

```
import flash.display.*;
import flash.net.URLRequest;
var rect:Shape = new Shape();
rect.graphics.beginFill(0xFFFFFF);
rect.graphics.drawRect(0, 0, 100, 100);
addChild(rect);
var ldr:Loader = new Loader();
ldr.mask = rect;
var url:String = "http://www.unknown.example.com/content.swf";
var urlReq:URLRequest = new URLRequest(url);
ldr.load(urlReq);
addChild(ldr);
```

當您呼叫 Loader 物件的 load() 方法時，可以指定 context 參數，它是 LoaderContext 物件。LoaderContext 類別包括三個屬性，可以讓您定義如何使用載入內容的環境：

- checkPolicyFile：請只在載入影像檔 (而不是 SWF 檔) 時使用這個屬性。您必須為不是源自包含 Loader 物件之檔案所屬網域的影像檔指定這個方法。如果將這個屬性設定為 true，Loader 會檢查 URL 原則檔的原始伺服器 (請參閱第 907 頁 「網站控制 (原則檔)」)。如果伺服器授與 Loader 網域權限，Loader 網域中 SWF 檔的 ActionScript 就可以存取已載入影像中的資料；換句話說，您可以使用 Loader.content 屬性，取得代表載入影像之 Bitmap 物件的參考，或使用 BitmapData.draw() 或 BitmapData.drawWithQuality() 方法，存取載入影像中的像素。drawWithQuality 方法可以在 Flash Player 11.3 及更新版本，以及 AIR 3.3 及更新版本中取得。
- securityDomain：請只在載入 SWF 檔 (而不是影像) 時使用這個屬性。為不是源自包含 Loader 物件之檔案所屬網域的 SWF 檔指定這個方法。目前 securityDomain 屬性只支援兩個值：null (預設值) 和 SecurityDomain.currentDomain。若指定 SecurityDomain.currentDomain，則會要求將被載入端 SWF 檔 「匯入」載入端 SWF 檔的安全執行程序中，也就是說，會以好像已從載入端 SWF 檔本身之伺服器載入的方式運作。只有被載入端 SWF 檔之伺服器具有 URL 原則檔，而且載入端 SWF 檔的網域允許存取，才能如此運作。如果找到所需的原則檔，則一旦載入作業開始之後，載入器與被載入者都可以自由地互相編寫對方的指令碼，因為它們都位於相同的安全執行程序中。請注意，安全執行程序匯入作業大部分也可以改為執行一般載入作業，然後讓被載入端 SWF 檔呼叫 Security.allowDomain() 方法。後面這種方法可能會比較容易使用，因為被載入端 SWF 檔將會處於本身的自然安全執行程序中，因此能夠存取本身實際伺服器上的資源。
- applicationDomain：請只在載入以 ActionScript 3.0 撰寫的 SWF 檔 (而非影像或以 ActionScript 1.0 或 2.0 撰寫的 SWF 檔) 時使用此屬性。載入檔案時，您可以將檔案置入特定的應用程式網域中，而不是依預設置入載入端 SWF 檔之應用程式網域子系的新應用程式網域中。請注意，應用程式網域是安全網域的子單位，因此只有在所載入的 SWF 檔來自您本身的安全網域 (因為檔案來自您本身的伺服器，或是因為您已使用 securityDomain 屬性，順利地將它匯入您的安全網域) 中時，您才能指定目標應用程式網域。如果指定應用程式網域，但被載入端 SWF 檔屬不同安全網域的一部分，就會忽略您在 applicationDomain 中指定的網域。如需詳細資訊，請參閱第 125 頁 「使用應用程式網域」。

如需詳細資訊，請參閱第 168 頁 「指定載入內容」。

Loader 物件有一個重要屬性，就是 contentLoaderInfo 屬性，它是 LoaderInfo 物件。與一般其它物件不同的是，載入端 SWF 檔與載入的內容會共享 LoaderInfo 物件，而且兩方永遠都可以存取。當載入的內容是 SWF 檔時，就可以透過 DisplayObject.loaderInfo 屬性存取 LoaderInfo 物件。LoaderInfo 物件包含資訊，如載入進度、載入器與被載入者的 URL、載入器與被載入者之間的信任關係，以及其它資訊。如需詳細資訊，請參閱第 167 頁 「監視載入進度」。

載入聲音和視訊

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

任何內容 (具有檔案系統的本機安全執行程序中的內容除外) 都允許使用 Sound.load()、NetConnection.connect() 和 NetStream.play() 方法，從網路來源載入聲音和視訊。

只有具有檔案系統的本機安全執行程序和 AIR 應用程式安全執行程序中的內容，才能從本機檔案系統載入媒體。只有具有檔案系統的本機安全執行程序、AIR 應用程式安全執行程序或本機信任安全執行程序中的內容，才能存取這些載入檔案中的資料。

從載入的媒體中存取資料還有其它限制。如需詳細資訊，請參閱第 919 頁 「將載入的媒體當做資料加以存取」。

使用文字欄位中的 <img> 標籤載入 SWF 檔和影像

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 <img> 標籤，將 SWF 檔和點陣圖載入文字欄位中，如下列程式碼所示：

```
<img src = 'filename.jpg' id = 'instanceName' >
```

您可以使用 TextField 實體的 getImageReference() 方法，存取以這種方式載入的內容，如下列程式碼所示：

```
var loadedObject:DisplayObject = myTextField.getImageReference('instanceName');
```

但是，請注意，以這種方式載入的 SWF 檔和影像是放在對應於其來源的安全執行程序中。

當您使用文字欄位中的 <img> 標籤載入影像檔案時，可以由 URL 原則檔授與權限存取影像中的資料。您可以透過將 checkPolicyFile 特質加入至 <img> 標籤來檢查原則檔，如下列程式碼所示：

```
<img src = 'filename.jpg' checkPolicyFile = 'true' id = 'instanceName' >
```

當您使用文字欄位中的 <img> 標籤載入 SWF 時，可以透過呼叫 Security.allowDomain() 方法，授與存取該 SWF 檔資料的權限。

當您使用文字欄位中的 <img> 標籤載入外部檔案 (而不是使用內嵌於 SWF 中的 Bitmap 類別) 時，會自動建立 Loader 物件做為 TextField 物件的子系，而外部檔案即載入該 Loader 中，與使用 ActionScript 中的 Loader 物件載入檔案完全一樣。在此情形下，getImageReference() 會傳回自動建立的 Loader。存取此 Loader 物件並不需要安全性檢查，因為它是位於與呼叫端程式碼相同的安全執行程序中。

但是，當您參考 Loader 物件的 content 屬性，以存取載入的媒體時，就會套用安全性規則。如果載入的內容是影像，您就必須實作 URL 原則檔；而如果載入的內容是 SWF 檔，則必須讓該 SWF 檔中的程式碼呼叫 allowDomain() 方法。

Adobe AIR

在應用程式安全執行程序中，文字欄位中的 <img> 標記會被略過，防止釣魚攻擊。此外，在應用程式安全執行程序中執行的程式碼，不允許呼叫 Security 的 allowDomain() 方法。

使用 RTMP 伺服器所傳遞的內容

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Media Server 會使用「即時媒體通訊協定」(Real-Time Media Protocol，RTMP) 提供資料、音效和視訊。您可以使用 NetConnection 類別的 connect() 方法載入這個媒體，並傳遞 RTMP URL 做為參數。Flash Media Server 可以根據要求檔案的網域限制連線，並防止內容遭到下載。如需詳細資訊，請參閱 Flash Media Server 線上說明文件，網址為 www.adobe.com/go/learn_fms_docs_tw。

若要使用 BitmapData.draw()、BitmapData.drawWithQuality() 和 SoundMixer.computeSpectrum() 方法擷取 RTMP 串流中的執行階段圖形和聲音資料，您必須允許伺服器上的存取權限。使用伺服器端 ActionScript Client.videoSampleAccess 和 Client.audioSampleAccess 屬性，允許存取 Flash Media Server 上的特定目錄。如需詳細資訊，請參閱伺服器端 ActionScript 語言參考 (drawWithQuality 方法可以在 Flash Player 11.3 及更新版本，以及 AIR 3.3 及更新版本中取得)。

跨指令碼處理

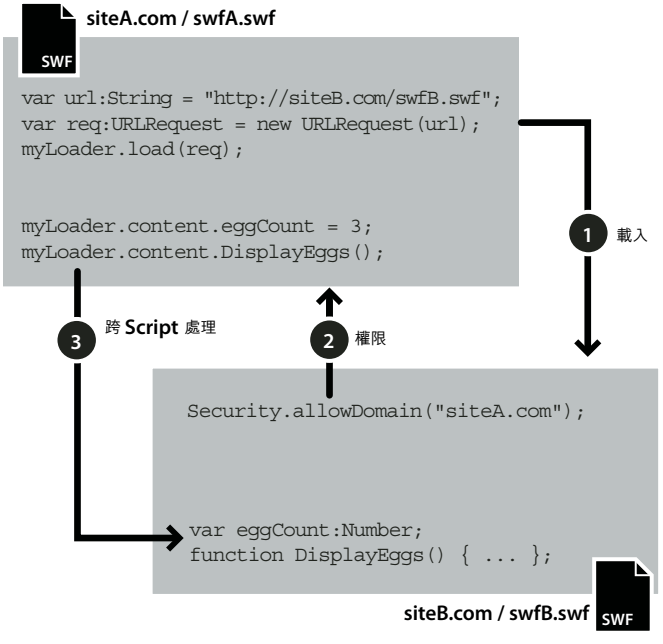
Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如果兩個 SWF 檔都是用 ActionScript 3.0 撰寫，或者在 AIR 中有兩個執行中的 HTML 檔案是同屬一個網域 (例如，一個 SWF 檔的 URL 是 http://www.example.com/swfA.swf，而另一個 SWF 檔的 URL 是 http://www.example.com/swfB.swf)，則一個 SWF 檔中定義的程式碼可以檢查並修改變數、物件、屬性及方法；同理，另一個 SWF 檔也可以反過來執行相同的動作，這就叫做「跨指令碼處理」。

如果而個檔案從不同的網域提供 (例如，http://siteA.com/swfA.swf 和 http://siteB.com/swfB.swf)，則根據預設值，Flash Player 和 AIR 不允許 swfA.swf 改編 swfB.swf，也不允許 swfB.swf 改編 swfA.swf。SWF 檔藉由呼叫 Security.allowDomain()，授與其它網域的 SWF 檔權限。swfB.swf 可透過呼叫 Security.allowDomain("siteA.com")，授與 siteA.com 網域中的 SWF 檔對它進行指令碼處理。

在 AVM1 SWF 檔與 AVM2 SWF 檔之間並不支援跨指令碼處理。AVM1 SWF 檔是以 ActionScript 1.0 或 ActionScript 2.0 所建立的檔案 (AVM1 和 AVM2 是指 ActionScript Virtual Machine)。但是，您可以使用 LocalConnection 類別，在 AVM1 與 AVM2 之間傳送資料。

在任何跨網域狀況下，清楚分辨牽涉的兩方是很重要的。為方便本文進行討論，我們將執行跨指令碼處理的一方稱為「存取的一方」(通常是指要進行存取的 SWF)，另一方則稱為「被存取的一方」(通常是被存取的 SWF)。當 siteA.swf 對 siteB.swf 進行指令碼處理時，siteA.swf 會是存取的一方，而 siteB.swf 則是被存取的一方，如下列圖例說明所示：



使用 Security.allowDomain() 方法建立的跨網域權限屬於非對稱式。在之前的範例中，siteA.swf 可以對 siteB.swf 進行指令碼處理，不過 siteB.swf 無法對 siteA.swf 進行指令碼處理，因為 siteA.swf 並未呼叫 Security.allowDomain() 方法，授與位於 siteB.com 之 SWF 檔的存取權限對其進行指令碼處理。您可以讓兩個 SWF 檔都呼叫 Security.allowDomain() 方法，以設定對稱的權限。

Flash Player 除了可以保護 SWF 檔不受到其它 SWF 檔進行跨網域指令碼處理之外，還可以保護 SWF 檔不受到 HTML 檔的跨網域指令碼處理。HTML-to-SWF 的指令碼編寫可以透過用 ExternalInterface.addCallback() 方法建立的回呼來執行。當 HTML-to-SWF 跨越網域編寫指令碼時，被存取方 SWF 必須呼叫 Security.allowDomain() 方法，而存取方是 SWF 檔時也相同，否則這項作業會失敗。如需詳細資訊，請參閱第 910 頁「作者 (開發人員) 控制」。

同時，Flash Player 也提供 SWF-to-HTML 編寫指令碼的安全性控制。如需詳細資訊，請參閱第 925 頁「控制外傳 URL 存取」。

Stage 安全性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Stage 物件有一些屬性和方法可供顯示清單上的任何 Sprite 或影片片段使用。

但是 Stage 物件有擁有者：就是第一個載入的 SWF 檔。根據預設，下列 Stage 物件的屬性和方法只供與 Stage 擁有者位於相同安全執行程序中的 SWF 檔使用：

屬性	方法
align	addChild()
displayState	addChildAt()
frameRate	addEventListener()
height	dispatchEvent()
mouseChildren	hasEventListener()
numChildren	setChildIndex()
quality	willTrigger()
scaleMode	
showDefaultContextMenu	
stageFocusRect	
stageHeight	
stageWidth	
tabChildren	
textSnapshot	
width	

若要讓不是位於 Stage 擁有者之安全執行程序中的 SWF 檔存取這些屬性和方法，Stage 擁有者 SWF 檔必須呼叫 Security.allowDomain() 方法，將權限授與外部安全執行程序的網域。如需詳細資訊，請參閱第 910 頁「作者 (開發人員) 控制」。

frameRate 屬性則是特殊情況，任何 SWF 檔都可以讀取 frameRate 屬性。但是，只有位於 Stage 擁有者之安全執行程序中的檔案 (或透過呼叫 Security.allowDomain() 方法授與權限的檔案) 才能變更此屬性。

對於 Stage 物件的 removeChildAt() 和 swapChildrenAt() 方法也有限制，但這些限制與其它限制不同。不是要求位於與 Stage 擁有者相同的安全執行程序中，而是若要呼叫這些方法程式碼必須位於與受影響子物件的擁有者相同的網域中，或者子物件也可以呼叫 Security.allowDomain() 方法。

在顯示清單中移動

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

一個 SWF 檔存取從其它安全執行程序中所載入之顯示清單的能力是有限制的。SWF 檔若要存取由不同安全執行程序中另外一個 SWF 檔所建立的顯示清單，則被存取端 SWF 檔必須呼叫 Security.allowDomain() 方法，授與存取端 SWF 檔所在網域存取權限。如需詳細資訊，請參閱第 910 頁「作者 (開發人員) 控制」。

若要存取由 Loader 物件載入的 Bitmap 物件，該影像檔的原始伺服器上必須有 URL 原則檔存在，而且該原則檔必須授與嘗試要存取 Bitmap 物件的 SWF 檔所在網域存取權限 (請參閱第 907 頁「網站控制 (原則檔)」)。

對應於載入的檔案 (及 Loader 物件) 之 LoaderInfo 物件包含下列三個屬性，這些屬性會定義載入的物件與 Loader 物件之間的關係：childAllowsParent、parentAllowsChild 和 sameDomain。

事件安全性

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

與顯示清單相關的事件具有安全存取限制，這些限制會根據傳送事件的顯示清單所在安全執行程序而不同。在顯示清單中的事件有反昇和捕捉階段 (說明於第 108 頁「處理事件」中)。在反昇與捕捉階段期間，事件從來源顯示清單遷移經過顯示清單中的父顯示物件，如果父物件位於與來源顯示清單不同的安全執行程序中，捕捉及反昇階段會在該父物件之下停止，除非在父物件與來源物件的擁有者具有彼此信任關係。這種彼此信任關係可以經由下列作業建立：

1. 擁有父物件的 SWF 檔必須呼叫 Security.allowDomain() 方法，以信任擁有來源物件之 SWF 檔的網域。
2. 擁有來源物件的 SWF 檔必須呼叫 Security.allowDomain() 方法，以信任擁有父物件的 SWF 檔之網域。

對應於載入的檔案 (及 Loader 物件) 之 LoaderInfo 物件包含下列兩個屬性，這些屬性會定義載入的物件與 Loader 物件之間的關係：childAllowsParent 和 parentAllowsChild。

對於從顯示物件以外物件傳送的事件，並沒有安全性檢查及安全性相關含意。

將載入的媒體當做資料加以存取

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

若要存取載入資料，請使用 BitmapData.draw()、BitmapData.drawWithQuality() 和 SoundMixer.computeSpectrum() 方法。透過不同安全執行程序載入的媒體所呈現或播放的圖形或音效物件，預設您無法從中取得像素資料或音效資料。不過，您可以使用下列方法，授與跨安全執行程序界限存取這類資料的權限：

- 在呈現或播放要存取資料的內容中，呼叫 Security.allowDomain() 方法，將資料存取權限授與其他網域中的內容。
- 對於載入的影像、聲音或視訊，請在載入的檔案之伺服器上加入 URL 原則檔。這個原則檔必須將權限授與 SWF 檔案的網域，此 SWF 檔案會嘗試呼叫 BitmapData.draw()、BitmapData.drawWithQuality() 或 SoundMixer.computeSpectrum() 方法，以便從檔案擷取資料。drawWithQuality 方法可以在 Flash Player 11.3 及更新版本，以及 AIR 3.3 及更新版本中取得。

下列各節會提供有關存取點陣圖、聲音和視訊資料的詳細資訊。

存取點陣圖資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

BitmapData 物件的 draw() 和 drawWithQuality() (Flash Player 11.3、AIR 3.3) 方法可以讓您將任何顯示物件的目前顯示像素繪製至 BitmapData 物件。這可以包括 MovieClip 物件、Bitmap 物件或任何顯示物件的像素。這些方法若要繪製像素至 BitmapData 物件，必須符合下列條件：

- 在來源物件不是載入的點陣圖之情況下，來源物件及其 (若是 Sprite 或 MovieClip 物件) 所有子物件都必須與呼叫 draw 方法的物件來自相同網域，或者必須在呼叫 Security.allowDomain() 方法以後呼叫者可存取的 SWF 檔案中。
- 在 Loaded 點陣圖來源物件的情況下，來源物件必須與呼叫 draw 方法的物件來自相同的網域，或者其來源伺服器必須包含 URL 原則檔，其中授與呼叫網域存取權限。

如果不符合這些條件，就會擲回 SecurityError 例外。

當您使用 Loader 類別的 load() 方法載入影像時，可以指定 context 參數 (LoaderContext 物件)。如果您將 LoaderContext 物件的 checkPolicyFile 屬性設定為 true，Flash Player 就會檢查載入聲音的來源伺服器是否有 URL 原則檔。如果有原則檔，而且此檔案授與載入 SWF 檔所在網域的權限，則該檔案就可以存取 Bitmap 物件中的資料，否則就會拒絕存取。

您也可以在經由文字欄位之 <img> 標籤載入的影像中，指定 checkPolicyFile 屬性。如需詳細資訊，請參閱第 916 頁「使用文字欄位中的 <img> 標籤載入 SWF 檔和影像」。

存取聲音資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

下列與聲音相關的 ActionScript 3.0 API 有安全性限制：

- SoundMixer.computeSpectrum() 方法：永遠允許與聲音檔位於相同安全執行程序中執行的程式碼。若為其他安全執行程序中執行的程式碼，就會進行安全性檢查。
- SoundMixer.stopAll() 方法：永遠允許與聲音檔位於相同安全執行程序中執行的程式碼。若為其它安全執行程序中的檔案，就會進行安全性檢查。
- 聲音類別的 id3 屬性：永遠都允許與聲音檔位於相同安全執行程序中的 SWF 檔存取。若為其他安全執行程序中執行的程式碼，就會進行安全性檢查。

每一個聲音檔都有兩種相關聯的安全執行程序，也就是內容安全執行程序和擁有者安全執行程序：

- 聲音的原始網域會決定內容安全執行程序，而且這會決定是否可經由聲音的 id3 屬性和 SoundMixer.computeSpectrum() 方法，從聲音中摘取資料。
- 開始播放聲音的物件則決定擁有者安全執行程序，而這會決定是否可以使用 SoundMixer.stopAll() 方法停止聲音播放。

當您使用 Sound 類別的 load() 方法載入聲音時，可以指定 context 參數 (SoundLoaderContext 物件)。如果您將 SoundLoaderContext 物件的 checkPolicyFile 屬性設為 true，執行階段就會檢查載入聲音的來源伺服器是否有 URL 原則檔。如果具有原則檔，而且檔案允許載入程式碼的網域，則該檔案就可以存取 Sound 物件的 id 屬性，否則就無法存取。此外，設定 checkPolicyFile 屬性也可以為載入的聲音啟用 SoundMixer.computeSpectrum() 方法。

您可以使用 SoundMixer.areSoundsInaccessible() 方法，查出呼叫 SoundMixer.stopAll() 方法是否會停止所有聲音，因為呼叫者無法存取一個或多個聲音擁有者。

呼叫 SoundMixer.stopAll() 方法會停止擁有者與 stopAll() 的呼叫者位於相同安全執行程序中的聲音；也會停止由 (已呼叫 Security.allowDomain() 方法，允許由呼叫 stopAll() 方法的 SWF 檔網域存取之) SWF 檔開始播放的聲音。任何其它聲音都不會停止，只要呼叫 SoundMixer.areSoundsInaccessible() 方法就能顯示這些聲音的存在。

呼叫 computeSpectrum() 方法需要播放的每一個聲音都與呼叫該方法的物件之安全執行程序相同，或是其來源已允許呼叫者的安全執行程序存取，否則就會擲回 SecurityError 例外。若聲音是從內嵌於 SWF 檔中元件庫之聲音載入，則會在被載入端 SWF 檔中呼叫 Security.allowDomain() 方法授與權限。若是從 SWF 檔以外來源載入聲音 (源自載入的 MP3 檔或從視訊檔)，在來源伺服器上的 URL 原則檔便會授與權限，以存取所載入之媒體中的資料。

如需詳細資訊，請參閱第 910 頁 「作者 (開發人員) 控制」和第 907 頁 「網站控制 (原則檔)」。

若要從 RTMP 串流存取聲音資料，您必須允許伺服器上的存取權限。使用伺服器端 ActionScript Client.audioSampleAccess 屬性，允許存取 Flash Media Server 上的特定目錄。如需詳細資訊，請參閱伺服器端 ActionScript 語言參考。

存取視訊資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以使用 BitmapData.draw() 或 BitmapData.drawWithQuality() 方法，擷取目前視訊影格中的像素資料 (drawWithQuality 方法可以在 Flash Player 11.3 及更新版本，以及 AIR 3.3 及更新版本中取得)。

總共有兩種不同的視訊：

- 來自 Flash Media Server 且透過 RTMP 串流的視訊
- 從 FLV 或 F4V 檔案載入的漸進式視訊

若要使用 draw 方法從 RTMP 串流擷取執行階段圖形，您必須允許伺服器上的存取權限。使用伺服器端 ActionScript Client.videoSampleAccess 屬性，允許存取 Flash Media Server 上的特定目錄。如需詳細資訊，請參閱伺服器端 ActionScript 語言參考。

當您以漸進式視訊做為 source 參數呼叫 draw 方法時，方法的呼叫者必須與 FLV 檔案位於相同的安全執行程序，或是 FLV 檔案的伺服器必須具有授與呼叫 SWF 檔案之網域存取權限的原則檔。您可以將 NetStream 物件的 checkPolicyFile 屬性設定為 true，要求下載原則檔。

載入資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 和 AIR 內容可以與伺服器交換資料。載入資料與載入媒體是不同類型的作業，因為載入的資訊會顯示為程式物件，而不是顯示為媒體。一般而言，內容可能會從內容的相同來源網域載入資料。不過，內容通常需要有原則檔，才能載入其他網域中的資料 (請參閱第 907 頁 「網站控制 (原則檔)」)。

備註：在 AIR 應用程式安全執行程序中執行的內容，絕不會從遠端網域提供 (除非開發人員故意將遠端內容匯入至應用程式安全執行程序)，因此無法參與原則檔所防護的攻擊類型。應用程式安全執行程序中的 AIR 內容並未限制透過原則檔載入資料。不過，其他安全執行程序中的 AIR 內容需要遵循這裡說明的限制。

使用 URLLoader 和 URLStream

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

您可以載入如 XML 檔和文字檔這類資料。URLLoader 和 URLStream 類別的 load() 方法是由 URL 原則檔權限加以控制。

如果您使用 load() 方法，從呼叫此方法之程式碼以外的網域載入內容，則執行階段會檢查所載入資源的伺服器上是否有 URL 原則檔。如果具有原則檔，而且此檔案授與載入內容之網域的存取權限，就可以載入資料。

連線至通訊端

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

執行階段預設會尋找從連接埠 843 提供的通訊端原則檔。就像 URL 原則檔一樣，這個檔案稱為 「主原則檔」。

最初在 Flash Player 6 加入原則檔時，並沒有通訊端原則檔的支援。至通訊端伺服器的連線是由原則檔進行授權，該原則檔位於與通訊端伺服器相同主機之埠號 80 上 HTTP 伺服器所含原則檔的預設位置中。Flash Player 9 仍然支援這項功能，但 Flash Player 10 不支援。在 Flash Player 10 中，只有通訊端原則檔可以授權通訊端連線。

通訊端原則檔與 URL 原則檔相同，都支援中繼原則陳述式，指定可提供原則檔的連接埠。但是，通訊端原則檔的預設中繼原則是 「全部」而不是 「僅主檔」。也就是說，除非主原則檔指定限制更多的設定，否則，Flash Player 會假設主機上的任何通訊端都可以提供通訊端原則檔。

根據預設，會停用通訊端和 XML 通訊端連線，即使您連線的通訊端是與 SWF 檔位於相同網域，也一樣停用。您可以透過從下列任何位置提供通訊端原則檔，准許通訊端層級存取：

- 連接埠 843 (主原則檔的位置)
- 與主通訊端連線相同的連接埠
- 與主通訊端連線不同的連接埠

根據預設，Flash Player 會在連接埠 843 上以及與主通訊端連線相同的連接埠上尋找通訊端原則檔。如果您要從不同的連接埠提供通訊端原則檔，SWF 檔必須呼叫 Security.loadPolicyFile()。

通訊端原則檔與 URL 原則檔具有相同的語法，不同的是，通訊端原則檔也必須指定所允許存取的埠號。當原則檔來自低於 1024 的埠號時，就可以授與存取權限給任何埠號；當原則檔來自 1024 或更高的埠號時，則只能授與存取權限給 1024 或更高的埠號。允許存取的連接埠是在 <allow-access-from> 標籤的 to-ports 特質中設定的。單一埠號、埠號範圍以及萬用字元也都是所接受的值。

以下是通訊端原則檔的範例：

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<!-- Policy file for xmlsocket://socks.mysite.com -->
<cross-domain-policy>
    <allow-access-from domain="*" to-ports="507" />
    <allow-access-from domain="*.example.com" to-ports="507,516" />
    <allow-access-from domain="*.example.org" to-ports="516-523" />
    <allow-access-from domain="adobe.com" to-ports="507,516-523" />
    <allow-access-from domain="192.0.34.166" to-ports="*" />
</cross-domain-policy>
```

若要從連接埠或與主通訊端連線相同的連接埠上擷取通訊端原則檔，請呼叫 Socket.connect() 或 XMLSocket.connect() 方法。Flash Player 會先在連接埠 843 上檢查是否有主原則檔。如果找到原則檔，會檢查檔案中是否包含中繼原則陳述式，禁止目標連接埠上的通訊端原則檔。如果未禁止存取，Flash Player 會先在主原則檔中尋找適當的 allow-access-from 陳述式。如果找不到檔案，就會到與主通訊端連線相同的連接埠上尋找通訊端原則檔。

若要在不同位置上擷取通訊端原則檔，請先以特殊的 "xmlsocket" 語法呼叫 Security.loadPolicyFile() 方法，如下所示：

```
Security.loadPolicyFile("xmlsocket://server.com:2525");
```

先呼叫 Security.loadPolicyFile() 方法，再呼叫 Socket.connect() 或 XMLSocket.connect() 方法，然後 Flash Player 會等到已完成您的原則檔要求之後，才決定是否允許您的主連線。但是，如果主原則檔指定目標位置不可提供原則檔，則呼叫 loadPolicyFile() 沒有作用，即使該位置上有原則檔也無效。

如果實作通訊端伺服器，而且必須提供通訊端原則檔，請決定在提供原則檔時，您是要使用接受主連線的連接埠，或是要使用不同的連接埠。不論何種情況，您的伺服器都必須先等候來自您用戶端的傳輸，再傳送回應。

當 Flash Player 要求原則檔時，連線一建立，就一定會立即傳輸下列字串：

```
<policy-file-request/>
```

伺服器接收到此字串以後，就可以傳輸原則檔。來自 Flash Player 的要求永遠都會以 null 位元組結束，而來自伺服器的回應也必須都以 null 位元組結束。

不要期望在原則檔和主連線上重新使用相同的連線，您應該在傳輸原則檔之後關閉連線。若不關閉連線，Flash Player 會先關閉原則檔連線，再重新連線設置主連線。

保護資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

為避免資料在網際網路上傳輸時遭到竊取和竄改，您可以在資料所在的伺服器上使用傳輸層安全性 (TLS) 或通訊端層安全性 (SSL)。接著您便可以使用 HTTPS 通訊協定來連線至伺服器。

在針對 AIR 2 或更新版本建立的應用程式中，您也可以保護 TCP 通訊端通訊。SecureSocket 類別可讓您起始通訊端連線，該連線會連至使用 TLS 第 1 版或 SSL 第 4 版的通訊端伺服器。

傳送資料

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

程式碼將資料傳送至伺服器或資源時，就會傳送資料。一律允許來自網路網域的內容傳送資料。本機 SWF 檔案只有在本機信任安全執行程序、具有網路的本機安全執行程序或 AIR 應用程式安全執行程序中時，才可以將資料傳送至網路位址。如需詳細資訊，請參閱第 902 頁 「本機安全執行程序」。

您可以使用 flash.net.sendToURL() 函數，傳送資料至 URL。其它方法也會傳送要求至 URL，這些包括載入方法 (如 Loader.load() 和 Sound.load()) 以及資料載入方法 (如 URLLoader.load() 和 URLStream.load())。

上傳和下載檔案

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

FileReference.upload() 方法會開始上傳使用者或遠端伺服器所選取的檔案。您必須先呼叫 FileReference.browse() 或 FileReferenceList.browse() 方法，再呼叫 FileReference.upload() 方法。

只有在回應滑鼠事件或鍵盤事件時，才能呼叫起始 FileReference.browse() 或 FileReferenceList.browse() 方法的程式碼。如果在其它情況下呼叫，Flash Player 10 和更新版本會擲回例外。不過，使用者起始的事件並不需要從 AIR 應用程式安全執行程序呼叫這些方法。

呼叫 FileReference.download() 方法會開啟對話方塊，使用者可以在其中從遠端伺服器下載檔案。

備註：如果伺服器需要使用者驗證，則只有在瀏覽器中執行 (也就是使用瀏覽器外掛程式或 ActiveX 控制項) 的 SWF 檔才能提供對話方塊，以提示使用者輸入使用者名稱和密碼以便進行驗證，而且僅供下載。Flash Player 不允許上傳至需要使用者驗證的伺服器。

如果呼叫的 SWF 檔位於具有檔案系統的本機安全執行程序中，則不允許上傳和下載。

根據預設，SWF 檔不可以啟動上傳至或下載自本身以外的伺服器。如果該伺服器提供原則檔，授與叫用 SWF 檔之網域的權限，則 SWF 檔可以針對不同的伺服器進行上傳或下載。

從匯入安全性網域中的 SWF 檔載入內嵌內容

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當您載入 SWF 檔時，可以設定用來載入檔案的 Loader 物件之 load() 方法的 context 參數。這個參數會接受 LoaderContext 物件。當您將此 LoaderContext 物件的 securityDomain 屬性設定為 Security.currentDomain 時，Flash Player 就會檢查被載入端 SWF 檔之伺服器是否有 URL 原則檔。如果有原則檔，而且此檔案允許載入端 SWF 檔的網域存取，就可以將 SWF 檔載入為匯入的媒體。如此一來，載入的檔案就可以存取 SWF 檔元件庫中的物件。

另外一種方式也可以讓 SWF 檔存取從不同安全執行程序之被載入端 SWF 檔中的類別，就是讓被載入端 SWF 檔呼叫 Security.allowDomain() 方法，以授與呼叫的 SWF 檔網域存取權限。您可以將 Security.allowDomain() 方法呼叫加入被載入端 SWF 檔主類別的建構函式方法中，然後讓載入端 SWF 檔加入事件偵聽程式，以回應由 Loader 物件的 contentLoaderInfo 屬性所傳送的 init 事件。傳送此事件時，載入的 SWF 檔已呼叫建構函式方法中的 Security.allowDomain() 方法，而被載入端 SWF 檔中類別可供載入端 SWF 檔使用。載入端 SWF 檔可以呼叫 Loader.contentLoaderInfo.applicationDomain.getDefinition() 或 Loader.contentLoaderInfo.applicationDomain.getQualifiedDefinitionNames()，從被載入端 SWF 檔擷取類別 (Flash Player 11.3 及更新版本，以及 AIR 3.3 及更新版本)。

使用舊版內容

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

在 Flash Player 6 中，用於某些 Flash Player 設定的網域，會依 SWF 檔之網域的結尾部分為準。這些設定包括攝影機和麥克風權限的設定、儲存配額及持續共享物件的儲存。

如果 SWF 檔的網域包含兩個以上的區段 (如 www.example.com)，便會移除該網域的第一個區段 (www)，而使用網域的其餘部分。因此，在 Flash Player 6 中，www.example.com 和 store.example.com 兩者都會使用 example.com 做為這些設定的網域。同樣的，www.example.co.uk 和 store.example.co.uk 兩者都會使用 example.co.uk 做為這些設定的網域。這樣可能會產生問題，使得來自不相關網域的 SWF 檔，如 example1.co.uk 和 example2.co.uk，可以存取相同的共享物件。

在 Flash Player 7 和更新版本中，會根據預設，依 SWF 檔的確實網域選擇播放程式設定。例如：來自 www.example.com 的 SWF 檔會使用 www.example.com 的播放程式設定，來自 store.example.com 的 SWF 檔則會另外使用 store.example.com 的播放程式設定。

在使用 ActionScript 3.0 撰寫的 SWF 檔中，當 Security.exactSettings 設定為 true (預設值) 時，Flash Player 會使用播放程式設定的精確網域；當它設定為 false 時，Flash Player 會使用 Flash Player 6 中所用的網域設定。如果變更 exactSettings 的預設值，則必須在要求 Flash Player 選擇播放程式設定之任何事件執行前進行變更，例如，使用攝影機或麥克風，或是擷取持續共享的物件。

如果您在之前發佈過版本 6 的 SWF 檔，並從其中建立持續的共享物件，若要從使用 ActionScript 3.0 的 SWF 檔中擷取持續共享物件，就必須將 Security.exactSettings 設定為 false，再呼叫 SharedObject.getLocal()。

設定 LocalConnection 連線權限

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

LocalConnection 類別可讓您在不同的 Flash Player 或 AIR 應用程式之間傳送訊息。LocalConnection 物件只能在相同用戶端電腦執行的 Flash Player 或 AIR 內容中進行通訊，但是可以在不同的應用程式中執行；例如，在瀏覽器中執行的 SWF 檔案、在專案中執行的 SWF 檔案以及 AIR 應用程式，都可以使用 LocalConnection 類別進行通訊。

每個 LocalConnection 通訊都會有傳送者和偵聽程式。Flash Player 預設會允許於相同網域執行的程式碼之間的 LocalConnection 通訊。針對不同安全執行程序中執行的程式碼，偵聽程式必須使用 LocalConnection.allowDomain() 方法授與傳送者權限。您傳遞做為 LocalConnection.allowDomain() 方法之引數的字串可以包含下列任何項目：精確的網域名稱、IP 位址，以及 * 萬用字元。

allowDomain() 方法的格式已變更，與 ActionScript 1.0 和 2.0 中的格式不同。在這些舊版本中，allowDomain 是您所實作的回呼方法。在 ActionScript 3.0 中，allowDomain() 是您所呼叫 LocalConnection 類別的內建方法。在這項變更下，allowDomain() 的運作方式與 Security.allowDomain() 非常類似。

SWF 檔可以使用 LocalConnection 類別的 domain 屬性來決定其網域。

控制外傳 URL 存取

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

外傳指令碼編寫和 URL 存取 (透過使用 HTTP URL、mailto: 等等) 是透過使用下列 API 來達成：

- flash.system.fscommand() 函數
- ExternalInterface.call() 方法
- flash.net.navigateToURL() 函數

針對從本機檔案系統載入的內容，只有在程式碼和包含網頁 (如果有的話) 位於本機信任或 AIR 應用程式安全執行程序時，才能成功呼叫這些方法。如果內容在具有網路的本機或具有檔案系統的本機安全執行程序中，呼叫這些方法就會失敗。

針對不在本機載入的內容，所有這些 API 都可以與內嵌的網頁通訊 (取決於如下所述的 AllowScriptAccess 參數的值)。flash.net.navigateToURL() 函數有額外功能可與任何已開啟的瀏覽器視窗或影格通訊，而不只是 SWF 檔所嵌入的網頁。如需有關此函數的詳細資訊，請參閱第 926 頁「使用 navigateToURL() 函數」。

載入 SWF 檔之 HTML 程式碼中的 AllowScriptAccess 參數可控制是否能從 SWF 檔之內執行外傳 URL 存取。在 PARAM 或 EMBED 標籤之內設定此參數。如果沒有為 AllowScriptAccess 設定任何值，SWF 檔和 HTML 網頁只有在都是來自相同的網域時才能進行通訊。

AllowScriptAccess 參數可以有以下其中一個可能值："always"、"sameDomain" 或 "never"。

- 當 AllowScriptAccess 設為 "always" 時，SWF 檔可以與所嵌入的 HTML 網頁進行通訊，即使 SWF 檔與 HTML 網頁位於不同的網域也可以通訊。
- 當 AllowScriptAccess 設為 "sameDomain" 時，SWF 檔只有在 SWF 檔與 HTML 網頁位於相同的網域，SWF 檔才能與嵌入的 HTML 網頁進行通訊。這個值是 AllowScriptAccess 的預設值。使用此設定，或是不要設定 AllowScriptAccess 的值，可防止從一個網域裝載的 SWF 檔存取來自另一個網域之 HTML 網頁中的指令碼。
- 當 AllowScriptAccess 設為 "never" 時，SWF 檔無法與任何 HTML 網頁進行通訊。從 Adobe Flash CS4 Professional 版本開始，已不建議使用此值。如果您不會從自己的網域提供未受信任的 SWF 檔，那麼便不建議使用這個值，而且也沒有必要。如果您的確需要提供不受信任的 SWF 檔，Adobe 建議您特別建立子網域，在此放置所有不受信任的內容。

以下是在 HTML 網頁中設定 AllowScriptAccess 標籤以允許傳出 URL 存取至不同網域的範例：

```
<object id='MyMovie.swf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.adobe.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,0,0' height='100%'
width='100%'>
<param name='AllowScriptAccess' value='always'/>
<param name='src' value=''MyMovie.swf'/>
<embed name='MyMovie.swf' pluginspage='http://www.adobe.com/go/getflashplayer' src='MyMovie.swf'
height='100%' width='100%' AllowScriptAccess='never'/>
</object>
```

使用 navigateToURL() 函數

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

除了上述所討論由 allowScriptAccess 參數指定的安全設定以外，navigateToURL() 函數有第二個選擇性的參數 - target。target 參數可以用來指定 HTML 視窗或影格的名稱，以便傳送 URL 要求。這種要求也套用了其他安全性限制，而其限制則取決於是否已使用 navigateToURL() 做為指令碼或非指令碼陳述式。

若為指令碼陳述式，例如 navigateToURL("javascript: alert('Hello from Flash Player.')")，則套用下列規則。

- 如果 SWF 檔是本機受信任的檔案，要求就會成功。
- 如果目標是 SWF 檔所嵌入的 HTML 網頁，則套用上述的 allowScriptAccess 規則。
- 如果目標保存從與 SWF 檔相同網域載入的內容，則要求會成功。
- 如果目標保存從與 SWF 檔不同網域載入的內容時，則上述兩個條件都不符合，要求會失敗。

若為非指令碼陳述式 (例如 HTTP、HTTPS 和 mailto:，如果套用下列所有條件，則要求會失敗：

- 目標是其中一個特殊的關鍵字 "_top" 或 "_parent"，而且
- SWF 檔是位於從不同網域裝載的網頁中，並且
- SWF 檔是以 allowScriptAccess 為非 "always" 的值嵌入。

如需詳細資訊

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

如需外傳 URL 存取的詳細資訊，請參閱 「適用於 Adobe Flash Platform 的 ActionScript 3.0 參考」中的下列項目：

- flash.system.fscommand() 函數
- ExternalInterface 類別的 call() 方法
- flash.net.navigateToURL() 函數

共享物件

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

Flash Player 提供使用 「共享物件」的功能，這是在 SWF 檔之外持續運作的 ActionScript 物件，可在使用者的檔案系統上本機執行，也能以遠端方式在 RTMP 伺服器中執行。共享物件與 Flash Player 中其它媒體相同，都會進行分割處理並置入安全執行程序中。但是，共享物件的安全執行程序模型有些不同，因為共享物件是不能夠跨越網域界限存取的資源。共享物件是一定要從共享物件存放區中擷取，該存放區是特屬呼叫 SharedObject 類別的方法之各個 SWF 檔所在網域。一般來說，共享物件存放區甚至比 SWF 檔的網域更為特定：根據預設，每個 SWF 檔都會使用特屬其整個原始 URL 的共享物件存放區。如需有關共享物件的詳細資訊，請參閱第 604 頁 「共享物件」。

SWF 檔可以使用 SharedObject.getLocal() 和 SharedObject.getRemote() 方法的 localPath 參數，以便使用僅與其 URL 部分相關聯的共享物件存放區。如此一來，SWF 檔就可以允許與其它 URL 的其它 SWF 檔共享。即使傳遞 '/' 做為 localPath 參數，也還是指定其本身網域專屬的共享物件存放區。

使用者可以使用 「Flash Player 設定」對話方塊或 「設定管理員」，限制存取共享物件。依預設，每個網域可建立的最大共享物件為 100 KB 的資料。系統管理使用者及使用者也可以限制寫入檔案系統的能力。如需詳細資訊，請參閱第 905 頁 「系統管理員控制」和第 906 頁 「使用者控制」。

您可以透過將 SharedObject.getLocal() 方法或 SharedObject.getRemote() 方法的 secure 參數指定為 true，指定共享物件為安全物件。請注意下列 secure 參數的相關事項：

- 如果這個參數設為 true，Flash Player 會建立新的安全共享物件，或取得現有之安全共享物件的參照。而這個安全的共享物件只能透過由 HTTPS 傳遞的 SWF 檔來讀取或寫入，方法是呼叫 secure 參數設為 true 的 SharedObject.getLocal()。
- 如果此參數設定為 false，Flash Player 就會建立新的共享物件，或取得現有共享物件 (可由透過非 HTTPS 連線傳遞的 SWF 檔讀取或寫入) 的參照。

如果呼叫的 SWF 檔不是來自 HTTPS URL，那麼為 SharedObject.getLocal() 方法或 SharedObject.getRemote() 方法的 secure 參數指定 true 的結果就會產生 SecurityError 例外。

共享物件存放區是根據 SWF 檔的原始 URL 進行選擇。即使在匯入載入及動態載入的情況下，SWF 檔不是源自簡單 URL 也是如此。匯入載入是指在將 LoaderContext.securityDomain 屬性設為 SecurityDomain.currentDomain 下載入 SWF 檔的情形。在這種情況下，被載入端 SWF 檔會有虛擬 URL，以其載入端 SWF 檔的網域開頭，然後指定其實際的原始 URL。動態載入是指使用 Loader.loadBytes() 方法載入 SWF 檔的作業。在這種情況下，被載入端 SWF 檔會有虛擬 URL，以其載入端 SWF 檔的網域開頭，然後指定其實際的原始 URL。在匯入載入和動態載入情況下，SWF 檔的虛擬 URL 可以使用 LoaderInfo.url 屬性進行檢查。在選擇共享物件存放區時，虛擬 URL 會視為與真正的 URL 完全一樣。您可以指定使用部分或全部虛擬 URL 的共享物件 localPath 參數。

使用者和系統管理員都可以選擇停用「協力廠商共享物件」。當 SWF 檔的原始 URL 是來自與瀏覽器位址列中所顯示 URL 不同網域時，這是任何在網頁瀏覽器中執行之 SWF 檔的共享物件用法。使用者和系統管理員可以因隱私權的關係，選擇停用協力廠商共享物件用法，以避免跨網域追蹤。為了要避開這項限制，您可能要確保使用共享物件的任何 SWF 檔在載入時，一定在確保該 SWF 檔是來自與瀏覽器位址列中所顯示網域的 HTML 網頁結構之內。當您嘗試使用來自協力廠商的共享物件，而且該協力廠商共享物件已遭停用時，SharedObject.getLocal() 和 SharedObject.getRemote() 方法就會傳回 null。如需詳細資訊，請參閱 www.adobe.com/products/flashplayer/articles/thirdpartylso。

攝影機、麥克風、剪貼簿、滑鼠和鍵盤存取

Flash Player 9 以及更新的版本，**Adobe AIR 1.0** 以及更新的版本

當 SWF 檔嘗試使用 Camera.get() 或 Microphone.get() 方法存取使用者的攝影機或麥克風時，Flash Player 會顯示「私用」對話方塊，讓使用者可以用來允許或拒絕存取他們的攝影機和麥克風。使用者和系統管理使用者也可以透過 mms.cfg 檔、「設定 UI」和「設定管理員」中的控制項，依每個站台或全域方式停用攝影機存取 (請參閱第 905 頁「系統管理員控制」和第 906 頁「使用者控制」)。在使用者限制下，Camera.get() 和 Microphone.get() 方法都各傳回 null 值。您可以使用 Capabilities.avHardwareDisable 屬性，判斷已透過系統管理的方式禁止 (true) 或允許 (false) 使用攝影機和麥克風。

System.setClipboard() 方法允許 SWF 檔以純文字字元字串取代剪貼簿中的內容。這種做法不會有安全性風險。為了要保護密碼及其它機密資料被剪下或複製到剪貼簿所產生的風險，並沒有對應的 getClipboard 方法。

在 Flash Player 中執行的應用程式只能監視在其焦點上發生的鍵盤和滑鼠事件。在 Flash Player 中執行的內容無法偵測其它應用程式中的鍵盤或滑鼠事件。

AIR 安全性

Adobe AIR 1.0 以及更新的版本

AIR 安全性的基本概念

Adobe AIR 1.0 以及更新的版本

AIR 應用程式使用與原生應用程式相同的安全性限制來執行。一般來說，AIR 應用程式就像原生應用程式一樣，允許廣泛存取作業系統功能，例如讀取及寫入檔案、啟動應用程式、在螢幕上繪圖以及與網路通訊。適用於原生應用程式的作業系統限制 (例如特定的使用者權限)，也同樣適用於 AIR 應用程式。

雖然 Adobe® AIR® 安全性模型是由 Adobe® Flash® Player 安全性模型演化而來，其安全性合約與套用於瀏覽器中內容的安全性合約並不相同。這份合約提供開發人員安全性，以供開發提供豐富經驗的廣泛功能，而且擁有在瀏覽器執行之應用程式所不具備的自由。

AIR 應用程式是使用已編譯的位元組碼 (SWF 內容) 或已解譯指令碼 (JavaScript、HTML) 撰寫，因此執行階段會提供記憶體管理。這樣可以將由記憶體管理相關弱點 (如緩衝區溢位和記憶體損毀等) 影響 AIR 應用程式的機會降到最低。這些都是一些最常見弱點，可能會影響以原生程式碼撰寫的桌面應用程式。

安裝與更新

Adobe AIR 1.0 以及更新的版本

AIR 應用程式透過 AIR 安裝程式檔案 (使用 air 副檔名) 或透過原生安裝程式 (使用原生平台的檔案格式與副檔名) 來散佈。例如，Windows 的原生安裝程式格式是 EXE 檔案，而 Android 的原生格式則是 APK 檔案。

安裝 Adobe AIR 並開啟 AIR 安裝程式檔案時，AIR 執行階段會管理安裝程序。使用原生安裝程式時，作業系統會管理安裝程序。

備註：使用 AIR 檔案格式時，開發人員可以指定版本和應用程式名稱，以及發行者來源，但是初始應用程式安裝工作流程本身是無法修改的。這項限制其實對使用者頗有幫助，因為所有 AIR 應用程式都共用由執行階段管理，安全、精簡而一致的安裝程序。如果應用程式必須自訂，可以在初次執行應用程式時提供。

執行階段安裝位置

Adobe AIR 1.0 以及更新的版本

使用 AIR 檔案格式的 AIR 應用程式首先要求在使用者的電腦上安裝執行階段，跟 SWF 檔案必須先安裝 Flash Player 瀏覽器外掛程式一樣。

執行階段安裝於桌上型電腦中的下列位置：

- Mac OS：/Library/Frameworks/
- Windows：C:\Program Files\Common Files\Adobe AIR
- Linux：/opt/Adobe AIR/

若要在 Mac OS 上安裝更新版的應用程式，使用者必須具有適當的系統權限，才能在應用程式目錄內進行安裝。在 Windows 和 Linux 中，使用者必須具有系統管理權限。

備註：在 iOS 上，不會分開安裝 AIR 執行階段，每個 AIR 應用程式都是獨立的應用程式。

執行階段可以透過兩種方式進行安裝：使用隱藏安裝功能 (直接從網頁瀏覽器安裝) 或是以手動方式安裝。此外，封裝為原生安裝程式的 AIR 應用程式也可以將 AIR 執行階段當做一般應用程式安裝程序的一部分，進行安裝 (以這種方式散佈 AIR 執行階段時，需要重新散佈與 Adobe 的合約)。

隱藏安裝 (執行階段和應用程式)

Adobe AIR 1.0 以及更新的版本

隱藏安裝功能提供開發人員精簡的安裝經驗，以供尚未安裝 Adobe AIR 的使用者運用。在隱藏安裝的方式中，開發人員要建立 SWF 檔，讓應用程式可進行安裝。當使用者在 SWF 檔中按一下來安裝應用程式時，SWF 檔會嘗試偵測執行階段。如果無法偵測到已安裝執行階段，就會立即使用開發人員應用程式的安裝程序啟動執行階段。

手動安裝

Adobe AIR 1.0 以及更新的版本

另外，使用者也可以手動方式下載並安裝執行階段，再開啟 AIR 檔。然後，開發人員可以用不同的方式來散佈 AIR 檔 (例如，經由電子郵件或網站上的 HTML 連結)。開啟 AIR 檔之後，執行階段就會開始處理應用程式安裝作業。

應用程式安裝流程

Adobe AIR 1.0 以及更新的版本

AIR 安全性模型允許使用者決定是否安裝 AIR 應用程式。AIR 安裝經驗提供多項原生應用程式安裝技術改進，讓使用者更容易完成這項信任決策：

- 執行階段在所有作業系統上提供一致的安裝經驗，即使從網頁瀏覽器中的連結安裝 AIR 應用程式也完全一致。大部分原生應用程式安裝體驗需要依賴瀏覽器或其它應用程式所提供的安全性資訊 (如果有的話)。
- AIR 應用程式安裝體驗則會識別應用程式來源以及應用程式可用權限的相關資訊 (如果使用者允許進行安裝程序的話)。
- 執行階段會管理 AIR 應用程式的安裝程序。AIR 應用程式無法操控執行階段所使用的安裝程序。

一般來說，使用者不應該安裝來自不信任來源或無法驗證其來源的桌面應用程式。對 AIR 應用程式來說，證明原生應用程式安全性的相關負擔與其它可安裝應用程式完全一樣。

應用程式目的地

Adobe AIR 1.0 以及更新的版本

安裝目錄可以使用下列兩個選項中的任何一個來設定：

1. 使用者在安裝程序期間自訂之目的地。應用程式安裝至使用者指定的任何目錄。
2. 如果使用者不變更安裝目的地，應用程式會安裝於執行階段所指定的預設路徑：
   - Mac OS：~/Applications/
   - Windows XP 及其舊版：C:\Program Files\
   - Windows Vista：~/Apps/
   - Linux：/opt/

   如果開發人員在應用程式描述器檔案中指定 installFolder 設定值，應用程式就會安裝於此目錄的子路徑中。

AIR 檔案系統

Adobe AIR 1.0 以及更新的版本

AIR 應用程式的安裝程序會將開發人員包含於 AIR 安裝程式檔案中的所有檔案複製至使用者的本機電腦。已安裝的應用程式包含下列項目：

- Windows：包含 AIR 安裝程式檔案中所有檔案的目錄。執行階段也會在 AIR 應用程式安裝期間建立 exe 檔案。
- Linux：包含 AIR 安裝程式檔案中所有檔案的目錄。執行階段也會在 AIR 應用程式安裝期間建立 bin 檔案。
- Mac OS：包含 AIR 安裝程式檔案中所有內容的 app 檔案。此檔案可以使用 Finder 中的「顯示套件內容」選項加以檢視。執行階段會建立此 app 檔案，做為 AIR 應用程式安裝程序的一部分。

AIR 應用程式會透過下列方式執行：

- Windows：執行安裝資料夾中的 .exe 檔案，或對應於此檔案的捷徑 (例如 「開始」功能表或桌面上的捷徑)。
- Linux：啟動安裝資料夾中的 .bin 檔案，從 「應用程式」選單選擇應用程式，或者從別名或桌面捷徑執行。
- Mac OS：執行 .app 檔案或指向它的別名。

應用程式檔案系統也包含與應用程式功能相關的子目錄。例如，寫入已加密本機儲存的資訊，會儲存於依應用程式的應用程式識別名稱而命名的目錄名稱下方子目錄中。

AIR 應用程式儲存

Adobe AIR 1.0 以及更新的版本

AIR 應用程式具有寫入使用者硬碟中任何位置的權限，但是，建議開發人員使用 app-storage:/ 路徑，進行與應用程式相關的本機儲存。從應用程式寫入 app-storage:/ 的檔案會位於標準位置 (該位置依使用者的作業系統而定)：

- 在 Mac OS 上：應用程式的儲存目錄會依 AIR 版本而有所不同：
  - **AIR 3.2** 和更早版本 - <appData>/<appId>/Local Store/，其中 <appData> 是使用者的「偏好設定資料夾」，一般來說是 /Users/<user>/Library/Preferences
  - **AIR 3.3** 和更新版本 - <path>/Library/Application Support/<appID>/Local Store，其中 <path> 是 /Users/<user>/Library/Containers/<bundle-id>/Data (安全執行程序的環境) 或 /Users/<user> (若在安全執行程序環境外執行)
- 在 Windows 中：應用程式的儲存目錄是 <appData>\<appId>\Local Store\，其中的 <appData> 是使用者的 CSIDL_APPDATA 「特殊資料夾」，一般來說是 C:\Documents and Settings\<user>\Application Data
- 在 Linux 中：<appData>/<appID>/Local Store/，其中的 <appData> 是 /home/<user>/.appdata

您可以經由 air.File.applicationStorageDirectory 屬性存取應用程式儲存目錄。您可以使用 File 類別的 resolvePath() 方法存取其內容。如需詳細資訊，請參閱第 561 頁 「使用檔案系統」。

更新 Adobe AIR

Adobe AIR 1.0 以及更新的版本

當使用者安裝需要已更新執行階段版本的 AIR 應用程式時，執行階段會自動安裝所需的執行階段更新。

若要更新執行階段，使用者必須具有電腦的系統管理權限。

更新 AIR 應用程式

Adobe AIR 1.0 以及更新的版本

軟體更新的開發和部署是原生程式碼應用程式所面臨的最大安全性挑戰。AIR API 提供了改進這項作業的機制：Updater.update() 方法可以在啟動時叫用，以檢查 AIR 檔所在的遠端位置。如果必須更新，即下載、安裝 AIR 檔，然後重新啟動應用程式。開發人員不僅可以使用這個類別來提供新功能，而且可以回應潛在的安全性弱點。

Updater 類別只能用以更新以 AIR 檔案散佈的應用程式。以原生應用程式散佈的應用程式必須使用原生作業系統的更新設施 (若有任何設施)。

備註：開發人員可以透過設定應用程式描述器檔案的 versionNumber 屬性，指定應用程式的版本。

解除安裝 AIR 應用程式

Adobe AIR 1.0 以及更新的版本

移除 AIR 應用程式會移除應用程式目錄中的所有檔案。但是，不會移除應用程式可能已寫入應用程式目錄之外的所有檔案。移除 AIR 應用程式不會還原 AIR 應用程式對應用程式目錄之外的檔案所做的變更。

Windows 系統管理員的登錄設定

Adobe AIR 1.0 以及更新的版本

在 Windows 上，系統管理員可以設定電腦以阻止 (或允許) AIR 應用程式安裝及執行階段更新。這些設定包含於下列機碼下方的 Windows 登錄中：HKLM\Software\Policies\Adobe\AIR。設定包含下列項目：

登錄設定	說明
AppInstallDisabled	指定是否允許 AIR 應用程式安裝及解除安裝。設定為 0 表示 「允許」，設定為 1 表示 「不允許」。
UntrustedAppInstallDisabled	指定允許安裝未受信任的 AIR 應用程式 (不包含受信任憑證的應用程式)。設定為 0 表示「允許」，設定為 1 表示 「不允許」。
UpdateDisabled	指定是否允許更新執行階段，以做為背景工作或明確安裝的一部分。設定為 0 表示 「允許」，設定為 1 表示「不允許」。

Adobe AIR 的 HTML 安全性

Adobe AIR 1.0 以及更新的版本

本主題說明 AIR HTML 安全性架構以及如何使用 iframes、frames 和安全執行程序橋接，以安裝 HTML 類型的應用程式，並將 HTML 內容整合至 SWF 類型的應用程式。

執行階段會強制執行規則並提供機制，以克服 HTML 和 JavaScript 中可能的安全性弱點。不管應用程式是否主要用 JavaScript 撰寫，或是否將 HTML 和 JavaScript 內容載入 SWF 類型應用程式中，都會強制執行相同的規則。應用程式安全執行程序和非應用程式安全執行程序中的內容都有不同的權限。將內容載入 iframe 或 frame 中時，執行階段會提供安全的「安全執行程序橋接」機制，允許 frame 或 iframe 中的內容安全地與應用程式安全執行程序中的內容進行通訊。

AIR SDK 提供三個類別，可用來顯示 HTML 內容。

HTMLLoader 類別提供 JavaScript 程式碼與 AIR API 之間的緊密整合。

StageWebView 類別是一種 HTML 顯示類別，而且與主機 AIR 應用程式的整合非常有限。StageWebView 類別載入的內容永遠都不會放置在應用程式安全執行程序中，而且無法存取資料或是呼叫在主機 AIR 應用程式中的函數。在桌上型電腦平台上，StageWebView 類別會根據 Webkit (HTMLLoader 類別也會使用它) 使用內建的 AIR HTML 引擎。在行動平台上，StageWebView 類別會使用作業系統提供的 HTML 控制項。因此，在行動平台上，StageWebView 類別具有與系統網頁瀏覽器相同的安全性考量與弱點。

TextField 類別可以顯示 HTML 文字的字串。無法執行 JavaScript，但是文字可以包括連結與外部載入的影像。

如需詳細資訊，請參閱第 847 頁 「避免發生與安全性有關的 JavaScript 錯誤」。

設定 HTML 類型應用程式的概觀

Adobe AIR 1.0 以及更新的版本

Frame 和 iframe 提供便利的結構，可供在 AIR 中組織 HTML 內容。Frame 提供了方法，可維護資料持續性並可安全地配合遠端內容運作。

由於 AIR 中的 HTML 保留其正常的頁面類型組織，如果 HTML 內容的最上端框架 「瀏覽」至不同的頁面，HTML 環境會整個重新整理。您可以使用 frame 和 iframe 來維護 AIR 中的資料持續性，與在瀏覽器中執行之網路應用程式的方式相同。只要您不允許瀏覽至新頁面，就可以在最上端框架中定義您的主應用程式物件。請使用子 frame 或 iframe 載入並顯示應用程式的 transient 部分 (您有各種不同的方式可以維護與框架一起使用或代替框架使用的資料持續性。其中包括 Cookie、本機共享物件、本機檔案儲存、加密檔案存放區，以及本機資料庫儲存)。

因為 AIR 中的 HTML 會保留其可執行程式碼與資料之間正常而模糊的界線，所以 AIR 會將 HTML 環境內最上層頁框的內容放到應用程式安全執行程序中。在頁面 load 事件發生後，AIR 會限制任何可將文字字串轉換成可執行物件的操作 (例如 eval())。即使在應用程式不載入遠端內容時，這項限制也會強制執行。若要允許 HTML 內容執行這些限制的作業，必須使用 frame 或 iframe 將內容放入非應用程式安全執行程序中 (當使用某些依賴 eval() 函數的 JavaScript 應用程式架構時，就可能需要在執行程序子框架中執行內容)。如需有關應用程式安全執行程序中 JavaScript 之限制的完整清單，請參閱第 933 頁 「不同安全執行程序中內容的程式碼限制」。

由於 AIR 中的 HTML 保留其可載入可能不安全之遠端內容的功能，AIR 會強制執行相同來源的原則，以防止一個網域中的內容與其它網域中的內容互動。若要允許應用程式內容與其它網域中的內容彼此互動，可以設定橋接，做為父框架與子框架之間的介面。

設定父子安全執行程序關係

Adobe AIR 1.0 以及更新的版本

AIR 會將 sandboxRoot 和 documentRoot 特質加入 HTML 的 frame 和 iframe 元素。這些特質可以讓您將應用程式內容視為來自其它網域的內容：

特質	說明
sandboxRoot	用來判斷要放置框架內容之安全執行程序和網域的 URL。這裡必須使用 file:、http: 或 https: URL 配置。
documentRoot	從中載入框架內容的 URL。您必須使用 file:、app: 或 app-storage: URL 配置。

下列範例會對應安裝在應用程式之 sandbox 子目錄中的內容，以便在遠端安全執行程序和 www.example.com 網域中執行：

```
<iframe
    src="ui.html"
    sandboxRoot="http://www.example.com/local/"
    documentRoot="app:/sandbox/">
</iframe>
```

設定不同安全執行程序或網域中父和子框架之間的橋接

Adobe AIR 1.0 以及更新的版本

AIR 會將 childSandboxBridge 和 parentSandboxBridge 屬性加入至任何子框架的 window 物件。這些屬性可以讓您定義橋接，以做為父框架與子框架之間的介面。每個橋接都是單向的：

childSandboxBridge：childSandboxBridge 屬性允許子框架向父框架中的內容公開介面。若要公開介面，請將 childSandbox 屬性設定為子框架中的函數或物件，然後，您就能從父框架中的內容存取物件或函數。下列範例會示範在子框架中執行的指令碼如何將包含函數和屬性的物件公開給其父輩：

```
var interface = {};
interface.calculatePrice = function(){
    return .45 + 1.20;
}
interface.storeID = "abc"
window.childSandboxBridge = interface;
```

如果此子內容是在指定 id 為 "child" 的 iframe 中，您可以透過讀取該 frame 的 childSandboxBridge 屬性，從父內容中存取介面：

```
var childInterface = document.getElementById("child").childSandboxBridge;
air.trace(childInterface.calculatePrice()); //traces "1.65"
air.trace(childInterface.storeID)); //traces "abc"
```

parentSandboxBridge：parentSandboxBridge 屬性允許父框架向子框架中內容公開介面。若要公開介面，請將子框架的 parentSandbox 屬性設定為父框架中的函數或物件。然後，您就能從子框架中的內容存取物件或函數。下列範例會示範父輩 frame 中執行的指令碼如何將包含 save 函數的物件公開給子系：

```
var interface = {};
interface.save = function(text){
    var saveFile = air.File("app-storage:/save.txt");
    //write text to file
}
document.getElementById("child").parentSandboxBridge = interface;
```

使用此介面，子框架中的內容可以將文字儲存至 save.txt 檔案中。但是，它對檔案系統沒有任何其它存取權限。一般來說，應用程式內容應該盡可能向其它安全執行程序公開限制最多的介面。子系內容可以呼叫 save 函數，如下所示：

```
var textToSave = "A string.";
window.parentSandboxBridge.save(textToSave);
```

如果子內容嘗試設定 parentSandboxBridge 物件的屬性，執行階段就會擲出 SecurityError 例外。如果父內容嘗試設定 childSandboxBridge 物件的屬性，執行階段就會擲出 SecurityError 例外。

不同安全執行程序中內容的程式碼限制

Adobe AIR 1.0 以及更新的版本

第 931 頁 「Adobe AIR 的 HTML 安全性」主題簡介已說明，執行階段會強制執行規則並提供機制，以克服 HTML 和 JavaScript 中可能有的安全性弱點。本主題會列出這些限制。如果程式碼嘗試呼叫這些受限制的 API，執行階段會擲出錯誤，並顯示訊息 「應用程式安全執行程序中的 JavaScript 程式碼造成 Adobe AIR 執行階段安全性違規」。

如需詳細資訊，請參閱第 847 頁 「避免發生與安全性有關的 JavaScript 錯誤」。

使用 JavaScript eval() 函數及類似技巧的限制

Adobe AIR 1.0 以及更新的版本

對於應用程式安全執行程序中的 HTML 內容，在載入程式碼之後使用可以動態方式將字串轉換成可執行程式碼的 API 有其限制 (在傳送 body 元素的 onload 事件，而且 onload 處理常式函數已完成執行之後)。其目的在於防止應用程式不小心從非應用程式來源 (例如可能不安全的網路網域) 插入 (並執行) 程式碼。

例如，如果您的應用程式使用遠端來源的字串資料來寫入至 DOM 元素的 innerHTML 屬性，則該字串便能包含可執行不安全操作的 (JavaScript) 程式碼。但是，載入內容時，則不會有任何將遠端字串插入 DOM 中的風險。

其中一項限制在於使用 JavaScript eval() 函數。程式碼一旦載入應用程式安全執行程序，且處理了 onload 事件處理常式之後，您就只能以有限的方式使用 eval() 函數。下列規則適用於從應用程式安全執行程序載入程式碼「之後」，使用 eval() 函數的情形。

- 允許使用其中包含常值的運算式。例如：

  ```
  eval("null");
  eval("3 + .14");
  eval("'foo'");
  ```

- 允許物件常值，如下所示：

  ```
  { prop1: val1, prop2: val2 }
  ```

- 「禁止」物件常值 setter/getter，如下所示：

  ```
  { get prop1() { ... }, set prop1(v) { ... } }
  ```

- 允許陣列常值，如下所示：

  ```
  [ val1, val2, val3 ]
  ```

- 「禁止」包含屬性讀取的運算式，如下所示：

  ```
  a.b.c
  ```

- 「禁止」函數叫用。
- 「禁止」函數定義。
- 「禁止」設定任何屬性。
- 「禁止」函數常值。

但是，載入程式碼時，在 onload 事件之前，而在執行 onload 事件處理常式函數期間，這些限制不會套用至應用程式安全執行程序中的內容。

例如，載入程式碼之後，下列程式碼會導致執行階段擲出例外：

```
eval("alert(44)");
eval("myFunction(44)");
eval("NativeApplication.applicationID");
```

如果在應用程式安全執行程序中允許以動態方式產生的程式碼 (例如呼叫 eval() 函數時所建立的程式碼)，則會產生安全性風險。例如，應用程式可能會不小心執行從網路網域中載入的字串，而該字串中可能包含惡意程式碼。舉例來說，這程式碼可能會刪除或改變使用者電腦上的檔案；或者，程式碼可能會向不受信任網路網域回報本機檔案內容。

產生動態程式碼的方式如下列所示：

- 呼叫 eval() 函數。
- 使用 innerHTML 屬性或 DOM 函數，插入指令碼標籤 (載入應用程式目錄之外的指令碼)。
- 使用 innerHTML 屬性或 DOM 函數，插入具有內嵌程式碼的指令碼標籤 (而不是經由 src 特質載入指令碼)。
- 設定 script 標籤的 src 特質，以載入應用程式目錄之外的 JavaScript 檔案。

- 使用 javascript URL 配置 (如 href="javascript:alert('Test')")。
- 使用 setInterval() 或 setTimout() 函數，其中第一個參數 (定義函數以非同步方式執行) 是字串 (要進行評估) 而不是函數名稱 (如 setTimeout('x = 4', 1000))。
- 呼叫 document.write() 或 document.writeln()。

載入內容時，應用程式安全執行程序中的程式碼只能使用這些方法。

這些限制 「不」會阻止搭配 JSON 物件常值使用 eval()。這可以讓應用程式內容配合 JSON JavaScript 元件庫運作。但是，您會受到限制，因此無法使用多載的 JSON 程式碼 (搭配事件處理常式)。

對於其它 Ajax 架構和 JavaScript 程式碼元件庫，請檢查架構或元件庫中的程式碼是否能在以動態方式產生的程式碼上依照這些限制來運作。如果不是，請包含使用非應用程式安全執行程序中架構或元件庫的任何內容。如需詳細資訊，請參閱 第 903 頁「AIR 中 JavaScript 的限制」和第 940 頁 「在應用程式與非應用程式內容之間編寫指令碼」。Adobe 提供一份清單，當中列出可支援應用程式安全執行程序的 Ajax 架構，請參閱 http://www.adobe.com/products/air/develop/ajax/features/。

非應用程式安全執行程序中的 JavaScript 內容與應用程式安全執行程序中的內容不同，此 JavaScript 內容 「可以」隨時呼叫 eval() 函數，執行以動態方式產生的程式碼。

存取 AIR API 的限制 (適用於非應用程式安全執行程序)

Adobe AIR 1.0 以及更新的版本

非應用程式安全執行程序中的 JavaScript 程式碼不能存取 window.runtime 物件，因此這種程式碼無法執行 AIR API。如果非應用程式安全執行程序中內容呼叫下列程式碼，應用程式會擲出 TypeError 例外：

```
try {
    window.runtime.flash.system.NativeApplication.nativeApplication.exit();
}
catch (e)
{
    alert(e);
}
```

例外類型是 TypeError (未定義值)，因為非應用程式安全執行程序中的內容不認得 window.runtime 物件，因此會將其視為未定義值。

您可以使用指令碼橋接，向非應用程式安全執行程序公開執行階段功能。如需詳細資訊，請參閱第 940 頁 「在應用程式與非應用程式內容之間編寫指令碼」。

使用 XMLHttpRequest 呼叫的限制

Adobe AIR 1.0 以及更新的版本

在載入 HTML 內容時及 onLoad 事件期間，應用程式安全執行程序中的 HTML 內容不能使用同步 XMLHttpRequest 方法，從應用程式安全執行程序之外載入資料。

根據預設，非應用程式安全執行程序中的 HTML 內容不允許使用 JavaScript XMLHttpRequest 物件，從呼叫要求的網域之外其它網域載入資料。frame 或 iframe 標籤可以包含 allowcrosscomainxhr 特質。設定此特質至任何非 null 值會允許 frame 或 iframe 中內容使用 Javascript XMLHttpRequest 物件，從呼叫要求的程式碼網域以外的網域載入資料：

```
<iframe id="UI"
    src="http://example.com/ui.html"
    sandboxRoot="http://example.com/"
    allowcrossDomainxhr="true"
    documentRoot="app:/">
</iframe>
```

如需詳細資訊，請參閱第 936 頁 「在不同網域中的內容之間編寫指令碼」。

載入 CSS、frame、iframe 和 img 元素的限制 (適用於非應用程式安全執行程序)

Adobe AIR 1.0 以及更新的版本

遠端 (網路) 安全執行程序中的 HTML 內容只能從 (網路 URL 的) 遠端安全執行程序載入 CSS、frame、iframe 和 img 內容。

具有檔案系統的本機、具有網路連線的本機，或本機信任安全執行程序中的 HTML 內容只能從本機安全執行程序 (而不是從應用程式或遠端安全執行程序) 載入 CSS、frame、iframe 和 img 內容。

呼叫 JavaScript window.open() 方法的限制

Adobe AIR 1.0 以及更新的版本

如果經由呼叫 JavaScript window.open() 方法而建立的視窗顯示來自非應用程式安全執行程序的內容，則該視窗的標題會以主 (啟動) 視窗的標題開頭，後面加上冒號字元。您不能使用程式碼，將視窗的該部分標題移出螢幕之外。

非應用程式安全執行程序中的內容只能順利呼叫 JavaScript window.open() 方法來回應由使用者滑鼠或鍵盤互動所觸發的事件。如此可防止非應用程式內容建立視窗進而可能被人以詐欺手法加以濫用 (例如進行網路釣魚攻擊)。而且，滑鼠或鍵盤事件的事件處理常式不能設定 window.open() 方法，在延遲之後執行 (例如，透過呼叫 setTimeout() 函數執行)。

遠端 (網路) 安全執行程序中的內容只能使用 window.open() 方法，開啟遠端網路安全執行程序中的內容。它不能使用 window.open() 方法，開啟來自應用程式或本機安全執行程序的內容。

具有檔案系統的本機、具有網路連線的本機，或本機信任安全執行程序中的內容 (請參閱第 901 頁 「安全執行程序」) 只能使用 window.open() 方法，開啟本機安全執行程序中的內容。它不能使用 window.open() 方法，開啟來自應用程式或遠端安全執行程序的內容。

呼叫受限制程式碼時發生錯誤

Adobe AIR 1.0 以及更新的版本

如果您呼叫因為這些安全性限制而禁止在安全執行程序中使用的程式碼，執行階段便會傳送下列 JavaScript 錯誤：「應用程式安全執行程序中的 JavaScript 程式碼造成 Adobe AIR 執行階段安全性違規」。

如需詳細資訊，請參閱第 847 頁 「避免發生與安全性有關的 JavaScript 錯誤」。

從字串載入 HTML 內容時的安全執行程序保護

Adobe AIR 1.0 以及更新的版本

HTMLLoader 類別的 loadString() 方法可讓您在執行階段中建立 HTML 內容。不過，如果您用來做為 HTML 內容的資料是從不安全的網際網路來源載入，則資料有可能損毀。基於這個原因，根據預設，以 loadString() 方法建立的 HTML 不會放在應用程式安全執行程序中，而且無法存取 AIR API。不過，您可以將 HTMLLoader 物件的 placeLoadStringContentInApplicationSandbox 屬性設為 true，將透過 loadString() 方法建立的 HTML 放到應用程式安全執行程序中。如需詳細資訊，請參閱第 845 頁 「從字串載入 HTML 內容」。

在不同網域中的內容之間編寫指令碼

Adobe AIR 1.0 以及更新的版本

AIR 應用程式會在安裝時授予特殊權限。有一點很重要，絕對不能讓相同的權限洩漏給其它內容，包括不屬應用程式的遠端檔案和本機檔案。

關於 AIR 安全執行程序橋接

Adobe AIR 1.0 以及更新的版本

在正常情況下，網域的內容無法呼叫位於其它網域的指令碼。為了避免 AIR 應用程式遇到權限資訊或控制意外洩漏的情況，application (應用程式) 安全執行程序中內容 (與應用程式一起安裝的內容) 上設了下列限制：

- 應用程式安全執行程序中的程式碼無法透過呼叫 Security.allowDomain() 方法，在其它安全執行程序中受到許可。從應用程式安全執行程序呼叫這個方法將會擲出錯誤。
- 禁止透過設定 LoaderContext.securityDomain 或 LoaderContext.applicationDomain 屬性，將非應用程式內容匯入至應用程式安全執行程序。

不過，在某些情況下，主 AIR 應用程式需要來自遠端網域的內容，以控制主 AIR 應用程式中指令碼的存取，或反之亦然。為了完成上述作業，執行階段提供 「安全執行程序橋接」機制，可做為兩個安全執行程序之間的閘道。安全執行程序橋接可以提供遠端與應用程式安全執行程序之間的明確互動。

安全執行程序橋接會公開兩個物件，供被載入端和載入端指令碼存取：

- parentSandboxBridge 物件可以讓載入的內容將屬性和函數公開給被載入內容中的指令碼。
- childSandboxBridge 物件可以讓被載入的內容將屬性和函數公開給載入內容中的指令碼。

經由安全執行程序橋接公開的物件是由值傳遞，而非由參考傳遞。所有資料都已序列化。也就是說，由橋接一端公開的物件不能由另一端設定，而被公開的物件都是不具類型的。而且，您也只能公開簡單物件和函數，而不能公開複雜的物件。

如果子內容嘗試將某個物件設定為 parentSandboxBridge 物件，執行階段就會擲出 SecurityError 例外。同樣地，如果父內容嘗試將某個物件設定為 childSandboxBridge 物件，執行階段也會擲出 SecurityError 例外。

安全執行程序橋接範例 (SWF)

Adobe AIR 1.0 以及更新的版本

假設 AIR 音樂商店應用程式要允許遠端 SWF 檔廣播專集的價格，但不希望遠端 SWF 檔洩漏此價格是否為售價。為了執行這項作業，StoreAPI 類別提供了方法，可取得價格，但掩飾售價。然後，此 StoreAPI 類別的實體會指定給載入遠端 SWF 之 Loader 物件的 LoaderInfo 物件之 parentSandboxBridge 屬性。

下列是該 AIR 音樂商店的程式碼：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" title="Music Store"
creationComplete="initApp()">
    <mx:Script>
        import flash.display.Loader;
        import flash.net.URLRequest;

        private var child:Loader;
        private var isSale:Boolean = false;

        private function initApp():void {
            var request:URLRequest =
                    new URLRequest("http://[www.yourdomain.com]/PriceQuoter.swf")

            child = new Loader();
            child.contentLoaderInfo.parentSandboxBridge = new StoreAPI(this);
            child.load(request);
            container.addChild(child);
        }
        public function getRegularAlbumPrice():String {
            return "$11.99";
        }
        public function getSaleAlbumPrice():String {
            return "$9.99";
        }
        public function getAlbumPrice():String {
            if(isSale) {
                return getSaleAlbumPrice();
            }
            else {
                return getRegularAlbumPrice();
            }
        }
    </mx:Script>
    <mx:UIComponent id="container" />
</mx:WindowedApplication>
```

StoreAPI 物件呼叫主應用程式，以擷取一般專集價格，但在呼叫 getSaleAlbumPrice() 方法時，傳回 「無法使用」。下列程式碼會定義 StoreAPI 類別：

```
public class StoreAPI
{
    private static var musicStore:Object;

    public function StoreAPI(musicStore:Object)
    {
        this.musicStore = musicStore;
    }

    public function getRegularAlbumPrice():String {
        return musicStore.getRegularAlbumPrice();
    }

    public function getSaleAlbumPrice():String {
        return "Not available";
    }

    public function getAlbumPrice():String {
        return musicStore.getRegularAlbumPrice();
    }
}
```

下列程式碼代表報告商店價格、但不會報告銷售價格的 PriceQuoter SWF 檔之範例：

```
package
{
    import flash.display.Sprite;
    import flash.system.Security;
    import flash.text.*;

    public class PriceQuoter extends Sprite
    {
        private var storeRequester:Object;

        public function PriceQuoter() {
            trace("Initializing child SWF");
            trace("Child sandbox: " + Security.sandboxType);
            storeRequester = loaderInfo.parentSandboxBridge;

            var tf:TextField = new TextField();
            tf.autoSize = TextFieldAutoSize.LEFT;
            addChild(tf);

            tf.appendText("Store price of album is: " + storeRequester.getAlbumPrice());
            tf.appendText("\n");
            tf.appendText("Sale price of album is: " + storeRequester.getSaleAlbumPrice());
        }
    }
}
```

安全執行程序橋接範例 (HTML)

Adobe AIR 1.0 以及更新的版本

在 HTML 內容中，parentSandboxBridge 和 childSandboxBridge 屬性會加入至子文件的 JavaScript window 物件。如需如何在 HTML 內容中設定橋接函數的範例，請參閱第 862 頁「設定安全執行程序橋接介面」。

限制 API 公開

Adobe AIR 1.0 以及更新的版本

公開安全執行程序橋接時，務必要公開定義可濫用程度的高階 API。請記住，呼叫您橋接實作的內容可能會遭受危害 (例如，經由程式碼插入)。舉例來說，經由橋接公開 readFile(path:String) 方法 (可讀取任意檔案的內容) 便是可能會被濫用的弱點。較為適當的做法是公開不接受路徑且會讀取特定檔案的 readApplicationSetting() API。較具語意的手法會在部分應用程式遭到危害時，限制應用程式可能會造成的損害。

更多說明主題

第 861 頁「跨指令碼處理不同安全執行程序中內容」

第 903 頁「AIR 應用程式執行程序」

寫入磁碟

Adobe AIR 1.0 以及更新的版本

在網頁瀏覽器中執行的應用程式只能與使用者的本機檔案系統進行有限的互動。網頁瀏覽器會實作安全性原則，確保使用者的電腦不會因為載入網頁內容而受到危害。例如，在瀏覽器中透過 Flash Player 執行的 SWF 檔不能與已經在使用者電腦上的檔案直接進行互動。共享的物件和 Cookie 可以寫入至使用者的電腦，供維護使用者偏好設定及其它資料之用，但這是檔案系統互動的極限。由於 AIR 應用程式會以原生方式進行安裝，它們有不同的安全性合約，其中包括跨本機檔案系統讀取和寫入的功能。

這種自由會為開發人員增添許多責任。意外的應用程式安全性違規不但會危及應用程式的功能，更會對使用者的電腦造成傷害。因此，開發人員應該要參閱第 941 頁 「開發人員的最佳安全性做法」。

AIR 開發人員可以使用多種 URL 配置慣例，存取並將檔案寫入本機檔案系統：

URL 配置	說明
app:/	應用程目錄的別名。從此路徑存取的檔案會指定應用程式安全執行程序，並由執行階段授予完整權限。
app-storage:/	本機儲存目錄的別名，由執行階段進行標準化。從此路徑存取的檔案會指定非應用程式安全執行程序。
file:///	代表使用者硬碟的根目錄之別名。如果應用程式目錄中有檔案存在，從此路徑存取的檔案會指定應用程式安全執行程序，否則，會指定非應用程式安全執行程序。

備註：AIR 應用程式不能使用 app: URL 配置修改內容。而且，由於系統管理員設定所致，應用程式目錄只能讀取。

除非使用者電腦有系統管理員限制，否則，AIR 應用程式便有權限，可寫入至使用者硬碟的任何位置。建議開發人員使用 app-storage:/ 路徑來進行與應用程式相關的本機儲存。從應用程式寫入 app-storage:/ 的檔案會放置於標準位置：

- 在 Mac OS 上：應用程式的儲存目錄是 <appData>/<appId>/Local Store/，其中 <appData> 是使用者的偏好設定資料夾。一般是 /Users/< 使用者 >/Library/Preferences
- 在 Windows 上：應用程式的儲存目錄是 <appData>\<appId>\Local Store\，其中 <appData> 是使用者的 CSIDL_APPDATA 特殊資料夾，一般來說是 C:\Documents and Settings\< 使用者名稱 >\Application Data
- 在 Linux 中：<appData>/<appID>/Local Store/，其中的 <appData> 是 /home/<user>/.appdata

如果應用程式是設計用來與使用者檔案系統中的現有檔案互動，請務必參閱第 941 頁 「開發人員的最佳安全性做法」。

安全地使用未受信任內容

Adobe AIR 1.0 以及更新的版本

未指定至應用程式安全執行程序的內容可以為應用程式提供額外的編寫指令碼功能，但僅限於符合執行階段安全性準則的情況。本主題將說明 AIR 與非應用程式內容的安全性合約。

Security.allowDomain()

Adobe AIR 1.0 以及更新的版本

AIR 應用程式會限制非應用程式內容的指令碼存取，比 Flash Player 瀏覽器外掛程式對不受信任內容的指令碼存取限制更為嚴格。例如，在瀏覽器中的 Flash Player，當指定給本機信任的安全執行程序之 SWF 檔呼叫 System.allowDomain() 方法時，從指定網域載入的 SWF 就會獲得指令碼存取權授予。從 AIR 應用程式中 application (應用程式) 內容則不准許類似手法，因為這會授予非應用程式檔案不合理的存取權限，進而得以存取使用者的檔案系統。遠端檔案無法直接存取應用程式安全執行程序，即使呼叫 Security.allowDomain() 方法也不行。

在應用程式與非應用程式內容之間編寫指令碼

Adobe AIR 1.0 以及更新的版本

在應用程式與非應用程式內容之間編寫指令碼的 AIR 應用程式有更為複雜的安全性考量。不在應用程式安全執行程序中的檔案僅能透過使用安全執行程序橋接，存取應用程式安全執行程序中檔案的屬性和方法。安全執行程序橋接是做為應用程式與非應用程式內容之間的閘道，提供兩個檔案之間的明確互動。當正確使用時，安全執行程序橋接會提供一層外加的安全性，限制非應用程式內容存取屬應用程式內容一部分的物件參考。

安全執行程序橋接的優點透過範例最能說明清楚。假設 AIR 音樂商店應用程式要將 API 提供給要自行建立 SWF 檔的廣告商，讓商店應用程式之後可以與此 SWF 檔通訊。商店想要為廣告商提供方法，讓廣告商能從商店尋找演唱者和 CD，但是基於安全性考量，也要隔離一些方法和屬性，使協力廠商 SWF 檔案無法加以存取。

安全執行程序橋接可以提供這項功能。根據預設，在執行階段從外部載入 AIR 應用程式中的內容無法存取主應用程式中的任何方法或屬性。開發人員可以使用自訂安全執行程序橋接實作，在不需公開這些方法和屬性的情況下，將服務提供給遠端內容。請將安全執行程序橋接視為受信任與未受信任內容之間的通道，此通道不需要公開物件參考便可以在載入器與被載入者內容之間提供通訊。

如需有關如何安全使用安全執行程序橋接的詳細資訊，請參閱第 936 頁「在不同網域中的內容之間編寫指令碼」。

防範以動態方式產生的不安全 SWF 內容

Adobe AIR 1.0 以及更新的版本

Loader.loadBytes() 方法能夠讓應用程式從位元組陣列產生 SWF 內容。但是，從遠端來源載入資料中的插入攻擊，會在載入內容時產生嚴重的損害。這在將資料載入應用程式安全執行程序中時更是明顯，所產生的 SWF 內容可在此存取完整的 AIR API。

有些方法可用來合法使用 loadBytes() 方法且不產生可執行的 SWF 程式碼。例如，您可以使用 loadBytes() 方法產生影像資料，控制影像顯示的時間。有些方法則「需要」依賴執行程式碼以進行合法使用，像是動態建立 SWF 內容以進行音效播放。在 AIR 中，根據預設，loadBytes() 方法「不會」讓您載入 SWF 內容，而只允許您載入影像內容。在 AIR 中，loadBytes() 方法的 loaderContext 屬性具有 allowLoadBytesCodeExecution 屬性，您可以將其設定為 true，以明確地允許應用程式使用 loadBytes() 載入可執行的 SWF 內容。下列程式碼會示範如何使用這項功能：

```
var loader:Loader = new Loader();
var loaderContext:LoaderContext = new LoaderContext();
loaderContext.allowLoadBytesCodeExecution = true;
loader.loadBytes(bytes, loaderContext);
```

如果呼叫 loadBytes() 來載入 SWF 內容，而 LoaderContext 物件的 allowLoadBytesCodeExecution 屬性設定為 false (預設值)，則 Loader 物件會擲出 SecurityError 例外。

備註：在未來的 Adobe AIR 版本中，這個 API 可能會改變。發生改變時，您可能需要重新編譯使用 LoaderContext 類別的 allowLoadBytesCodeExecution 屬性之內容。

開發人員的最佳安全性做法

Adobe AIR 1.0 以及更新的版本

雖然 AIR 應用程式使用網頁技術建立，開發人員應該體認到這些應用程式並不會在瀏覽器安全執行程序中運作。也就是說，所建立的 AIR 應用程式，有可能有意或是無意地傷害了本機系統。AIR 試著將此風險減到最低，但還是有可能引進攻擊弱點。本主題涵蓋重要的潛在不安全性。

將檔案匯入應用程式安全執行程序中的風險

Adobe AIR 1.0 以及更新的版本

存在於應用程式目錄中的檔案會指定給應用程式安全執行程序，並具有執行階段的完整權限。建議將寫入本機檔案系統的應用程式寫入 app-storage:/。此目錄會與使用者電腦上的應用程式檔案分別存放，因此檔案並不會指定給應用程式安全執行程序，而降低了安全性風險。建議開發人員考慮下列情形：

- 僅在必要的時候，才在 AIR 檔中 (在安裝的應用程式中) 包含檔案。
- 僅在完全了解並信任 AIR 檔的行為時，才在當中 (在安裝的應用程式中) 包含編寫指令碼的檔案。

- 不要寫入或修改應用程式目錄中的內容。執行階段會透過擲出 SecurityError 例外，阻止應用程式使用 app:/ URL 配置寫入或修改檔案和目錄。
- 不要使用來自網路的資料做為可能會產生程式碼執行的 AIR API 方法之參數。其中包含使用 Loader.loadBytes() 方法和 JavaScript eval() 函數。

使用外部來源決定路徑的風險

Adobe AIR 1.0 以及更新的版本

AIR 應用程式可能會在使用外部資料或內容時遭到竄改。因此，使用來自網路或檔案系統的資料時必須特別注意。信任最終都得靠開發人員及所進行的網路連線來維護，但是載入外來資料本來就會有風險，因此不該用於機密作業的輸入用途。建議開發人員防範下列情形：

- 使用網路來源的資料以決定檔案名稱
- 使用網路來源的資料，以建構應用程式用來傳送私有資訊的 URL

使用、儲存或傳輸不安全憑證的風險

Adobe AIR 1.0 以及更新的版本

在使用者本機檔案系統上儲存使用者憑證需要承擔憑證可能遭到竄改的風險。建議開發人員考慮下列情形：

- 如果必須將憑證儲存在本機上，請在寫入本機檔案系統時對憑證進行加密。執行階段會透過 EncryptedLocalStore 類別，為每個已安裝的應用程式提供唯一的加密儲存。如需詳細資訊，請參閱第 612 頁 「加密本機儲存」。
- 除非來源受到信任且使用 HTTPS: 或傳輸層安全性 (TLS) 通訊協定來傳輸，否則請不要將未加密的使用者憑證傳輸至網路來源。
- 絕對不要在建立憑證時，指定預設的密碼：請讓使用者自行建立密碼。使用者若保留預設值不變，就等於將憑證公開給已知預設密碼的攻擊者一樣。

降級攻擊的風險

Adobe AIR 1.0 以及更新的版本

應用程式安裝期間，執行階段會進行檢查，以確認應用程式版本並不是目前安裝的版本。如果已安裝應用程式，執行階段會比較版本字串與安裝中版本的版本字串。如果字串不同，使用者可以選擇升級安裝。執行階段無法保證新安裝的版本會比舊版更新，只保證當中的內容有所不同。攻擊者可能會散佈舊版給使用者，設法讓使用者落入安全性弱點的圈套中。因此，建議開發人員在執行應用程式時進行版本檢查。讓應用程式檢查網路並尋找是否有必要的更新是很好的做法。如此一來，即使攻擊者讓使用者執行舊版，該舊版也會偵測到目前版本必須進行更新。而且，應用程式若使用清楚的版本編號配置，使用者就不會輕易受騙安裝降級版本。

程式碼簽署

Adobe AIR 1.0 以及更新的版本

所有 AIR 安裝程式檔案都需要進行程式碼簽署。程式碼簽署是一個密碼編譯程序，可確認軟體來源是否正當。AIR 應用程式可以使用外部憑證授權單位 (CA) 發出的憑證或您親自簽署的憑證進行簽署。我們強烈建議您使用知名 CA 提供的商業憑證，這可向使用者提供保證，他們確實是在安裝您的應用程式，而不是偽造的。不過，您可以透過下列方式，建立自我簽署的憑證：透過 SDK 的 adt，或透過 Flash、Flash Builder 或其他使用 adt 產生憑證的應用程式。對於安裝中的應用程式，自行簽署的憑證不保證其是否為真品，而且只能用於應用程式公開發行之前的測試。

Android 裝置上的安全性

Adobe AIR 2.5 以及更新的版本

在 Android 上，就如同所有的電腦裝置一樣，AIR 符合原生安全性模型。同時，AIR 會維護自己的安全性規則，其用意在於使開發人員易於撰寫安全且連接網際網路的應用程式。

因為在 Android 上的 AIR 應用程式使用 Android 套件格式，安裝要符合 Android 安全性模型。不會使用 AIR 應用程式安裝程式。

Android 安全性模型具有三個主要的層面：

- 權限
- 應用程式簽名
- 應用程式使用者 ID

Android 權限

Android 的許多功能是由作業系統權限機制保護。為了使用受保護的功能，AIR 應用程式描述器必須宣告應用程式所需的必要權限。當使用者嘗試安裝應用程式時，Android 作業系統會在安裝繼續之前，對使用者顯示所有必要的權限。

大部分的 AIR 應用程式都需要在應用程式描述器中指定 Android 權限。根據預設，不會包括任何權限。若要透過 AIR 執行階段公開受保護的 Android 功能，需要下列權限：

ACCESS_COARSE_LOCATION 允許應用程式透過 Geolocation 類別存取 WIFI 與行動電話網路位置資料。

ACCESS_FINE_LOCATION 允許應用程式透過 Geolocation 類別存取 GPS 資料。

ACCESS_NETWORK_STATE 與 ACCESS_WIFI_STATE 允許應用程式透過 NetworkInfo 類別存取網路資訊。

CAMERA 允許應用程式存取攝影機。

INTERNET 允許應用程式提出網路要求。也允許遠端除錯。

READ_PHONE_STATE 允許 AIR 執行階段在來電時將音效靜音。

RECORD_AUDIO 允許應用程式存取麥克風。

WAKE_LOCK 與 DISABLE_KEYGUARD 允許應用程式使用 SystemIdleMode 類別設定，避免裝置進入休眠。

WRITE_EXTERNAL_STORAGE 允許應用程式寫入裝置上的外部記憶卡。

應用程式簽名

為 Android 平台建立的所有應用程式套件都必須簽署。因為在 Android 上的 AIR 應用程式是以原生 Android APK 格式封裝，所以它們會根據 Android 慣例來簽署，而不是以 AIR 的慣用方式。雖然 Android 與 AIR 使用程式碼以類似方式進行簽署，但有著相當大的差異：

- 在 Android 上，簽名會確認私密金鑰具有所屬開發人員，並非用來確認開發人員的身分。
- 對於送出至 Android Market 的應用程式，憑證的有效期必須至少 25 年。
- Android 不支援將套件簽名移轉至其他憑證。如果更新是由不同的憑證簽署，則使用者必須解除安裝原始的應用程式，才能安裝更新的應用程式。
- 使用相同憑證簽署的兩個應用程式可以指定共用的 ID，以允許它們存取彼此的快取與資料檔案。(這樣的共用不是透過 AIR 進行。)

應用程式使用者 ID

Android 使用 Linux 核心。系統會為每個已安裝的應用程式指定 Linux 類型的使用者 ID，以便為檔案存取這類的操作決定其權限。在應用程式、應用程式儲存及暫存目錄中的檔案受到保護，要透過檔案系統權限才能存取。在電腦上將 SD 卡掛接成大量儲存裝置時，寫入外部儲存 (也就是 SD 卡) 的檔案可由其他應用程式或使用者讀取、修改和刪除。

以網際網路要求接收的 Cookie 不會在 AIR 應用程式之間共用。

背景影像隱私權

使用者將應用程式切換至背景時，有些 Android 版本會擷取螢幕快照，在目前的應用程式清單以縮圖形式使用。此螢幕快照儲存在裝置記憶體中，攻擊者可以存取來實際控制裝置。

如果您的應用程式會顯示機密資訊，則應防範背景螢幕快照擷取這類資訊。NativeApplication 物件所傳送的 deactivate 事件指出，應用程式即將切換至背景。使用此事件可以清除或隱藏任何機密資訊。

更多說明主題

Android：安全性與權限

Android 上加密的資料

在 Android 上的 AIR 應用程式，可以使用內建 SQL 資料庫可用的加密選項，以儲存加密的資料。為了讓安全性達到最佳狀態，每當應用程式執行時都要以使用者所輸入密碼的對應加密金鑰為基礎。您無法向能夠存取應用程式檔案的攻擊者隱藏在本機儲存的加密金鑰或密碼。如果攻擊者能擷取金鑰，即使加密資料，Android 系統提供的使用者 ID 系統檔案安全功能也無法發揮效用。

EncryptedLocalStore 類別可用來儲存資料，但在 Android 裝置並不會加密該資料。但 Android 安全性模式仰賴應用程式使用者 ID 來保護資料，防止其它應用程式存取。使用共享使用者 ID 且以相同的程式碼簽署憑證來簽署的應用程式，都會使用同一個加密的本機儲存區。

重要事項：在根層級的手機上，以根權限執行的任何應用程式都能存取任何其它應用程式的檔案。因此，使用加密的本機儲存區儲存的資料在根層級裝置上並不安全。

iOS 裝置上的安全性

在 iOS 上，AIR 符合原生安全性模型。同時，AIR 會維護自己的安全性規則，其用意在於使開發人員易於撰寫安全且連接網際網路的應用程式。

因為在 iOS 上的 AIR 應用程式使用 iOS 套件格式，安裝符合 iOS 安全性模型。不會使用 AIR 應用程式安裝程式。此外，在 iOS 裝置上不會使用個別的 AIR 執行階段。所有 AIR 應用程式均包含運作所需的程式碼。

應用程式簽名

為 iOS 平台建立的所有應用程式套件都必須簽署。因為 iOS 上的 AIR 應用程式會以原生 iOS IPA 格式建立套件，所以會根據 iOS 需求而不是 AIR 需求進行簽署。雖然 iOS 與 AIR 使用程式碼以類似方式進行簽署，但有著相當大的差異：

- 在 iOS 上，用來簽署應用程式的憑證必須由 Apple 簽發；無法使用其他憑證機構簽發的憑證。
- 在 iOS 上，Apple 簽發的分配憑證通常有效期為 1 年。

背景影像隱私權

當使用者將應用程式切換到 iOS 上的背景時，作業系統會擷取用來製成轉場動畫的螢幕快照。此螢幕快照儲存在裝置記憶體中，攻擊者可以存取來實際控制裝置。

如果您的應用程式會顯示機密資訊，則應防範背景螢幕快照擷取這類資訊。NativeApplication 物件所傳送的 deactivate 事件指出，應用程式即將切換至背景。使用此事件可以清除或隱藏任何機密資訊。

第 65 章 如何使用 ActionScript 範例

瞭解特定類別和方法運作方式的最佳方法之一，就是執行 ActionScript 3.0 程式碼範例。您可以根據使用的裝置或目標裝置，以不同方式使用範例。

執行 Flash Professional 或 Flash Builder 的電腦 請參閱第 946 頁「在 Flash Professional 執行 ActionScript 3.0 範例」或第 948 頁「在 Flash Builder 執行 ActionScript 3.0 範例」，取得有關如何在這些開發環境執行 ActionScript 3.0 範例的資訊。使用 trace 陳述式以及其他除錯工具，可增加您對於程式碼範例運作方式的瞭解。

行動裝置 您可以在支援 Flash Player 10.1 及更新版本的行動裝置上執行 ActionScript 3.0 程式碼範例。請參閱第 948 頁「在行動裝置執行 ActionScript 3.0 範例」。您也可以在執行 Flash Professional 或 Flash Builder 的電腦上執行這些範例。

電視機 雖然您無法在電視機執行這些範例，但還是可以在電腦執行，以便從範例中學習。請參閱 Adobe Developer Connection 網站上的電視版 Flash Platform，取得有關開發電視機專用應用程式的資訊。

範例類型

ActionScript 3.0 程式碼範例的類型：

- 程式碼片段範例 (可在 ActionScript 3.0 文件集內找到)
- 類別範例 (主要可在 ActionScript 3.0 語言參考中找到)
- 包含多個來源檔案的實際範例 (請從 www.adobe.com/go/learn_programmingAS3samples_flash_tw 下載來源 ZIP 檔案)

程式碼片段範例

程式碼片段看起來如下：

```
var x:int = 5;
trace(x); // 5
```

程式碼片段包含的程式碼，只是用於展示一種想法。它們通常不會包含套件或類別陳述式。

類別範例

很多範例顯示完整 ActionScript 類別的原始碼。類別範例看起來如下：

```
package {
    public class Example1 {
        public function Example1():void {
            var x:int = 5;
            trace(x); //5
        }
    }
}
```

類別範例的程式碼包含一個套件陳述式、類別宣告以及一個建構函式。

包含多個來源檔案的實際範例

《ActionScript 3.0 開發人員指南》中的很多主題會以實際範例做為總結，這些範例顯示如何在實際環境中使用特定的 ActionScript 功能。這些範例通常內含多個檔案，其中包括：

- 一或多個 ActionScript 來源檔案
- Flash Professional 使用的 .FLA 檔案

- 一或多個 Flash Builder 使用的 MXML 檔案
- 範例應用程式使用的資料檔案、影像檔案、來源檔案或者其他資源 (選擇性)。

實際範例一般會以 ZIP 封存檔案提供。

開發人員指南的範例 ZIP 檔案清單

Flash Professional CS5 和 Flex 4 的 ZIP 檔案 (請從 www.adobe.com/go/learn_programmingAS3samples_flash_tw 下載) 包含下列範例：

- AlarmClock (第 120 頁 「事件處理範例：鬧鐘」)
- AlgorithmicVisualGenerator (第 196 頁 「繪製 API 範例：演算法視覺產生器」)
- ASCIIArt (第 17 頁 「字串範例：ASCII 藝術」)
- CapabilitiesExplorer (第 752 頁 「功能範例：偵測系統功能」)
- CustomErrors (第 58 頁 「處理錯誤範例：CustomErrors 應用程式」)
- DisplayObjectTransformer (第 183 頁 「幾何範例：將矩陣變形套用至顯示物件」)
- FilterWorkbench (第 246 頁 「濾鏡處理顯示物件範例：濾鏡工作台」)
- GlobalStockTicker (第 821 頁 「範例：國際化股票行情應用程式」)
- IntrovertIM_HTML (第 731 頁 「外部 API 範例：使用瀏覽器在 ActionScript 和 JavaScript 之間通訊」)
- NewsLayout (第 327 頁 「TextField 範例 : 報紙樣式文字格式」)
- PlayList (第 39 頁 「陣列範例：PlayList」)
- PodcastPlayer (第 396 頁 「聲音範例：Podcast Player」)
- ProjectionDragger (第 301 頁 「範例：透視投影」)
- ReorderByZ (第 306 頁 「使用 Matrix3D 物件以重新排列顯示物件」)
- RSSViewer (第 97 頁 「ActionScript 中的 XML 範例：從網際網路載入 RSS 資料」)
- RuntimeAssetsExplorer (第 279 頁 「移動影片片段範例：RuntimeAssetsExplorer」)
- SimpleClock (第 5 頁 「日期與時間範例：簡易類比時鐘」)
- SpinningMoon (第 215 頁 「點陣圖範例：動畫轉動的月球」)
- SpriteArranger (第 171 頁 「顯示物件範例：SpriteArranger」)
- TelnetSocket (第 689 頁 「TCP 通訊端範例：建立 Telnet 用戶端」)
- VideoJukebox (第 431 頁 「視訊範例：視訊點唱機」)
- WikiEditor (第 77 頁 「一般表示式範例：Wiki 剖析器」)
- WordSearch (第 494 頁 「滑鼠輸入範例：WordSearch」)

在 「Flash 開發人員中心」和 「Flex 開發人員中心」中也可以找到很多與 「快速入門」文章有關的實際範例。

在 Flash Professional 執行 ActionScript 3.0 範例

使用下列其中一個程序 (視範例類型而定) 可執行使用 Flash Professional 的範例。

在 **Flash Professional** 執行程式碼片段範例

在 Flash Professional 執行程式碼片段範例：

1 選取 「檔案 > 新增」。

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

新的 Flash 視窗隨即顯示。

3 按一下 「時間軸」面板中第一個圖層的第一個影格。

4 在 「動作」面板中，輸入或貼上程式碼片段範例。

5 選取 「檔案 > 儲存」。輸入檔案的名稱，然後按一下 「確定」。

6 若要測試範例，請選取 「控制 > 測試影片」。

在 **Flash Professional** 執行類別範例

在 Flash Professional 執行類別範例：

1 選取 「檔案 > 開新檔案」。

2 在 「新增文件」對話方塊中選取 「ActionScript 檔案」，然後按一下 「確定」。新的編輯器視窗隨即顯示。

3 將類別範例程式碼複製後貼到編輯器視窗。

如果此類別是程式的主要文件類別，它必須擴充 MovieClip 類別：

```
import flash.display.MovieClip;
public class Example1 extends MovieClip{
//...
}
```

另外請確定在範例中參照的所有類別都會使用 import 陳述式來宣告。

4 選取 「檔案 > 儲存」。將檔案的名稱指定為範例中相同的類別名稱 (例如 ContextMenuExample.as)。

備註：某些類別範例，例如 flashx.textLayout.container.ContainerController 類別範例，在套件宣告中包含多個階層 (package flashx.textLayout.container.examples {)。對於這些範例，只要將檔案儲存在符合封裝宣告 (flashx/textLayout/container/examples) 的子資料夾中，或移除封裝名稱 (這樣 ActionScript 會只以 package { 開頭)，您就可以從任何位置測試檔案。

5 選取 「檔案 > 開新檔案」。

6 在 「新增文件」對話方塊中選取 「Flash 文件 (ActionScript 3.0)」，然後按一下 「確定」。新的 Flash 視窗隨即顯示。

7 在 「屬性」面板中的 「文件類別」欄位，輸入範例類別的名稱，此名稱就是剛才儲存的 ActionScript 來源檔案名稱 (例如 ContextMenuExample)。

8 選取 「檔案 > 儲存」。將 FLA 檔案的名稱指定為範例中相同的類別名稱 (例如 ContextMenuExample.fla)。

9 若要測試範例，請選取 「控制 > 測試影片」。

在 **Flash Professional** 執行實際範例

實際範例一般會以 ZIP 封存檔案提供。使用 Flash Professional 執行實際範例：

1 將封存檔案解壓縮至選擇的資料夾。

2 在 Flash Professional 中，選取 「檔案 > 開啟」。

3 瀏覽封存檔案要解壓縮的資料夾。選取資料夾中的 FLA 檔案，然後按一下 「開啟」。

4 若要測試範例，請選取 「控制 > 測試影片」。

在 Flash Builder 執行 ActionScript 3.0 範例

使用下列其中一個程序 (視範例類型而定) 可執行使用 Flash Builder 的範例。

在 **Flash Builder** 執行程式碼片段範例

在 Flash Builder 執行程式碼片段範例：

1. 建立新的 Flex 專案 (選取「 檔案 > 新增 > Flex 專案」) 或者在現有的 Flex 專案中建立新的 MXML 應用程式 (選取「檔案 > 新增 > MXML 應用程式」)。為專案或應用程式指定描述性名稱 (例如 ContextMenuExample)。
2. 在產生的 MXML 檔案中，新增一個 <mx:Script> 標籤。
3. 將程式碼片段內容貼到 <mx:Script> 和 </mx:Script> 標籤之間。儲存 MXML 檔案。
4. 若要執行範例，請選取 「執行 > 主要 MXML 檔案的執行功能表選項」 (例如 「執行 > 執行 ContextMenuExample)」。

在 **Flash Builder** 執行類別範例

在 Flash Builder 執行類別範例：

1. 選取 「檔案 > 新增 > ActionScript 專案」。
2. 在 「專案名稱」欄位中輸入主要類別的名稱 (例如 ContextMenuExample)。其他欄位使用預設值 (或者根據您的特殊環境，變更它們)。按一下 「完成」，建立專案以及主要的 ActionScript 檔案。
3. 擦除 ActionScript 檔案中任何產生的內容。將包括套件和匯入陳述式的範例程式碼貼到 ActionScript 檔案中，然後儲存檔案。

   備註：某些類別範例，例如 flashx.textLayout.container.ContainerController 類別範例，在套件宣告中包含多個階層 (package flashx.textLayout.container.examples {)。對於這些範例，只要將檔案儲存在符合封裝宣告 (flashx/textLayout/container/examples) 的子資料夾中，或移除封裝名稱 (這樣 ActionScript 會只以 package { 開頭)，您就可以從任何位置測試檔案。
4. 若要執行範例，請選取 「執行 > 主要 ActionScript 類別名稱的執行功能表選項」 (例如 「執行 > 執行 ContextMenuExample)」。

在 **Flash Builder** 執行實際範例

實際範例一般會以 ZIP 封存檔案提供。使用 Flash Builder 執行實際範例：

1. 將封存檔案解壓縮至選擇的資料夾。為資料夾指定描述性名稱 (例如 ContextMenuExample)。
2. 在 Flash Builder 中，選取 「檔案 > 新增 Flex 專案」。在 「專案位置」部分中，按一下 「瀏覽」，然後選取內含範例檔案的資料夾。在 「專案名稱」欄位中，輸入資料夾名稱 (例如 ContextMenuExample)。其他欄位使用預設值 (或者根據您的特殊環境，變更它們)。按 「下一步」繼續。
3. 在 「輸出」面板中按 「下一步」，接受預設值。
4. 在 「來源路徑」面板中按一下 「主要應用程式檔案」欄位旁邊的 「瀏覽」按鈕。從範例資料夾選取主要 MXML 範例檔案。按一下 「完成」來建立專案檔案。
5. 若要執行範例，請選取 「執行 > 主要 MXML 檔案的執行功能表選項」 (例如 「執行 > 執行 ContextMenuExample)」。

在行動裝置執行 ActionScript 3.0 範例

您可以在支援 Flash Player 10.1 的行動裝置執行 ActionScript 3.0 程式碼範例。不過，一般您可以執行程式碼範例，便能得知特定類別和方法的運作方式。在這種情況下，在桌上型電腦之類的非行動裝置執行範例。在桌上型電腦上，您可以在 Flash Professional 或 Flash Builder 使用 trace 陳述式以及其他除錯工具，增加您對於程式碼範例的瞭解。

若您想在行動裝置上執行範例，可以將檔案複製到裝置或網站伺服器。若要將檔案複製到裝置，然後在瀏覽器中執行範例，請執行以下步驟：

1 按照第 946 頁「在 Flash Professional 執行 ActionScript 3.0 範例」或第 948 頁「在 Flash Builder 執行 ActionScript 3.0 範例」中的指示，執行以下步驟。在 Flash Professional 中，當您選取 「控制 > 測試影片」，就可以建立 SWF 檔案。在 Flash Builder 中，當您執行、除錯或建立自己的 Flash Builder 專案時，就可以建立 SWF 檔案。

2 將 SWF 檔案複製到行動裝置上的目錄。使用裝置提供的軟體來複製檔案。

3 在行動裝置的瀏覽器位址列，輸入 SWF 檔案的 file:// URL。例如，輸入 file:://applications/myExample.swf。

若要將檔案複製到網站伺服器，然後在裝置的瀏覽器中執行範例，請執行以下步驟：

1 建立 SWF 檔案以及 HTML 檔案。首先，按照第 946 頁「在 Flash Professional 執行 ActionScript 3.0 範例」或第 948 頁「在 Flash Builder 執行 ActionScript 3.0 範例」中的指示執行。在 Flash Professional 中，選取「控制 > 測試影片」，只建立 SWF 檔案。若要建立這兩個檔案，請先在 「發佈設定」對話方塊的 「格式」索引標籤中選取 「Flash」和「HTML」。然後，選取「檔案 > 發佈」，建立 HTML 和 SWF 檔案二者。在 Flash Builder 中，當您執行、除錯或建立自己的 Flash Builder 專案時，就可以同時建立 SWF 檔案和 HTML 檔案。

2 將 SWF 檔案和 HTML 檔案複製到網站伺服器的目錄。

3 在行動裝置的瀏覽器位址列，輸入 HTML 檔案的 HTTP 網址。例如，輸入 http://www.myWebServer/examples/myExample.html。

在行動裝置上執行範例之前，請考慮以下每一個問題。

舞台大小

在行動裝置上執行範例時，您使用的舞台大小會比您使用非行動裝置時要小一些。很多範例不需要特殊的 「舞台」大小。建立 SWF 檔案時，請指定裝置適合的 「舞台」大小。例如，指定 176 x 208 個像素。

在《ActionScript 3.0 開發人員指南》中，實際範例的用途是說明不同的 ActionScript 3.0 概念和類別。他們的使用者介面設計就是要在桌上型電腦或筆記型電腦上看來美觀又好用。雖然，這些範例可以在行動裝置上執行，不過 「舞台」大小和使用者介面設計並不適合很小的螢幕。Adobe 建議您在電腦上執行實際的範例，對 ActionScript 有了相當的瞭解之後，再到行動應用程式中使用特定的程式碼片段。

以文字欄位取代 **trace** 陳述式

在行動裝置上執行範例時，您不會看到範例中的 trace 陳述式輸出。若要瞭解輸出，請建立 TextField 類別的實體。然後。將 trace 陳述式的文字附加至文字欄位的 text 屬性。

您可以使用以下的函數，設定一個文字欄位，以便用於追蹤：

```
function createTracingTextField(x:Number, y:Number,
                                width:Number, height:Number):TextField {

    var tracingTF:TextField = new TextField();
    tracingTF.x = x;
    tracingTF.y = y;
    tracingTF.width = width;
    tracingTF.height = height;

    // A border lets you more easily see the area the text field covers.
    tracingTF.border = true;
    // Left justifying means that the right side of the text field is automatically
    // resized if a line of text is wider than the width of the text field.
    // The bottom is also automatically resized if the number of lines of text
    // exceed the length of the text field.
    tracingTF.autoSize = TextFieldAutoSize.LEFT;

    // Use a text size that works well on the device.
    var myFormat:TextFormat = new TextFormat();
    myFormat.size = 18;
    tracingTF.defaultTextFormat = myFormat;

    addChild(tracingTF);
    return tracingTF;
}
```

例如，將此函數新增至文件類別，作為一個私有的函數。然後，在文件的其他方法中，使用類似下列的程式碼來追蹤資料：

```
var traceField:TextField = createTracingTextField(10, 10, 150, 150);
// Use the newline character "\n" to force the text to the next line.
traceField.appendText("data to trace\n");
traceField.appendText("more data to trace\n");
// Use the following line to clear the text field.
traceField.appendText("");
```

appendText() 方法只接受一個值當作是參數。這個值是字串 (字串實體或字串常值)。若要列印非字串變數的值，請先將該值轉換為 String。最簡單的轉換方式就是呼叫物件的 toString() 方法：

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

文字大小

很多範例使用文字欄位，協助說明概念。有時候，調整文字欄位中的文字大小，可以讓使用者更容易看清楚行動裝置的資料。例如，如果範例使用 myTextField 這個文字欄位實體，請使用以下程式碼變更其文字大小：

```
// Use a text size that works well on the device.
var myFormat:TextFormat = new TextFormat();
myFormat.size = 18;
myTextField.defaultTextFormat = myFormat
```

擷取使用者輸入

行動作業系統和瀏覽器會捕捉 SWF 內容不會接收的部分使用者輸入事件。特定行為會視作業系統和瀏覽器而定，但是當您在行動裝置執行範例時，可能會造成無法預期的行為。如需詳細資訊，請參閱 第 480 頁「KeyboardEvent 優先順序」。

而且，很多範例的使用者介面則是針對桌上型電腦和筆記型電腦而設計。例如，《ActionScript 3.0 開發人員指南》中，大部分的實際範例非常適合在桌上型電腦檢視。因此，在行動裝置螢幕上，有時候看不到完整的舞台。瀏覽器內容的平移能力視瀏覽器而定。再者，這些範例的左右移動是針對捕捉以及處理捲動或左右平移事件而設計。因此，部分範例的使用者介面並不適會在較小的螢幕執行。Adobe 建議您在電腦上執行這些範例，對 ActionScript 有了相當的瞭解之後，再到行動應用程式中使用特定的程式碼片段。

如需詳細資訊，請參閱第 150 頁 「左右移動和捲動顯示物件」。

處理焦點

部分範例需要您提供焦點給欄位。提供焦點給欄位後，您就可以輸入文字或選取按鈕。若要提供欄位焦點，請使用行動裝置的指標裝置，例如觸控筆或您的手指。或者，使用行動裝置的導覽鍵，提供欄位焦點。若要選取焦點所在的按鈕，請使用行動裝置的 「選取」鍵，就像是使用電腦上的 Enter 一樣。在部分裝置上，輕點按鈕兩下就可以選取它。

如需有關焦點的詳細資訊，請參閱第 475 頁 「管理焦點」。

處理滑鼠事件

很多範例會偵聽滑鼠事件。在電腦上，例如當使用者使用滑鼠在顯示物件上滾動，或者在顯示物件上按一下滑鼠按鍵後，就會發生這些滑鼠事件。在行動裝置上，觸控筆或手指之類的指標裝置所產生的事件，稱為觸控事件。Flash Player 10.1 會將觸控事件對應至滑鼠事件。這種對應關係可以確保先開發 SWF 內容，再繼續 Flash Player 10.1 的工作。因此，使用指標裝置來選取或拖曳顯示物件時，這些範例仍有效。

效能

行動裝置的處理能力比桌上型裝置差。部分消耗大量 CPU 的範例在行動裝置上的執行速度很慢。例如，第 196 頁「繪製 API 範例：演算法視覺產生器」在進入每一個影格時，會進行大量計算和繪圖。在電腦上執行此範例會說明各種的繪圖 API。不過，由於效能有限，所以此範例不適合部分的行動裝置。

如需行動裝置效能的詳細資訊，請參閱最佳化 Flash 平台的效能。

最佳作法

為行動裝置開發應用程式時，範例並不會將最佳作法納入考量。行動裝置的記憶體與處理能力有限，這一點要特別注意。同樣地，小型螢幕的使用者介面與桌上型電腦顯示器各有不同的需求。如需開發行動裝置應用程式的詳細資訊，請參閱最佳化 Flash 平台的效能。

第 66 章 本機資料庫內的 SQL 支援

Adobe AIR 包含一個 SQL 資料庫引擎，可使用開放程式碼 SQLite 資料庫系統來支援具有許多標準 SQL 功能的本機 SQL 資料庫。執行階段不會指定資料庫資料在檔案系統上的儲存方式或儲存位置。每個資料庫都完整儲存在一個檔案內。開發人員可以指定資料庫檔案在檔案系統中的儲存位置，而且一個 AIR 應用程式可以存取一或多個個別的資料庫 (即個別資料庫檔案)。本文件概述 Adobe AIR 本機 SQL 資料庫支援的 SQL 語法和資料類型。本文件不是完整的 SQL 參考文件，而只是要說明 Adobe AIR 支援之 SQL 方言的具體詳細資料。執行階段支援大部分的 SQL-92 標準 SQL 方言。由於市面上有各種學習 SQL 的參考資料、網站、書籍和訓練教材，因此本文件的目的不在提供完整的 SQL 參考文件或教學資料，而會著重在探討 Apollo AIR 支援的 SQL 語法，以及 SQL-92 與支援的 SQL 方言之間的差異。

SQL 陳述式定義慣例

本文件的陳述式定義中使用下列慣例：

- 文字大小寫
  - UPPER CASE - 常值 SQL 關鍵字以全大寫表示。
  - lower case - 預留位置用詞或子句名稱以全小寫表示。
- 定義字元
  - ::= 表示子句或陳述式定義。
- 群組和替代字元
  - | 管道字元用在替代選項之間，可解釋為 「或」。
  - [] 方括弧中的項目是選擇性的項目，方括弧中可包含單一項目或一組替代項目。
  - () 括在括弧裡的一組替代項目 (以管道字元分隔的項目集合) 用來指定必要的項目群組，也就是這一組項目代表一個必要項目的可能值。
- 數量詞
  - + 加號後面接著括在括弧裡的項目，表示之前的項目可以發生 1 或多次。
  - * 星號字元後面接著括在方括弧裡的項目，表示之前 (括弧裡) 的項目可發生 0 或多次。
- 常值字元
  - * 星號字元用在欄名稱中或函數名稱後的括弧中，表示是常值星號字元，而非 「0 或多次」數量詞。
  - . 句點字元代表常值句點。
  - , 逗號字元代表常值逗號。
  - () 一組括弧括住一個子句或項目，表示括弧是必要的常值括弧字元。
  - 其它字元，除非以其它方式指定，否則其它字元都代表該常值字元。

支援的 SQL 語法

Adobe AIR SQL 資料庫引擎支援下列 SQL 語法清單。這些清單分為幾個部分，分別說明不同陳述式和子句類型、運算式、內建函數和運算子。涵蓋的主題如下：

- 一般 SQL 語法
- 資料操作陳述式 (SELECT、INSERT、UPDATE 和 DELETE)

- 資料定義陳述式 (資料表、索引、檢視和觸發程序的 CREATE、ALTER 和 DROP 陳述式)
- 特殊陳述式和子句
- 內建函數 (彙總、純量和日期 / 時間格式函數)
- 運算子
- 參數
- 不支援的 SQL 功能
- 額外的 SQL 功能

一般 SQL 語法

除了各種陳述式和運算式的特定語法之外，SQL 語法的一般規則如下：

區分大小寫 SQL 陳述式 (包括物件名稱) 不區分大小寫。但 SQL 陳述式中的 SQL 關鍵字通常使用大寫，因此本文件沿用這個慣例。雖然 SQL 語法不區分大小寫，但 SQL 中的文字常值則須區分大小寫，而且比較和排序運算也可能須區分大小寫 (視欄或作業的定序順序定義方式而定)。如需詳細資訊，請參閱 「COLLATE」。

空格 必須使用空白字元 (例如空格、Tab、換行字元等) 分隔 SQL 陳述式中的個別單字。但是文字與符號之間的空白則是選擇性的。SQL 陳述式中空白字元的類型和數量並不重要。您可以使用空白 (例如縮排和斷行符號) 設定 SQL 陳述式的格式，以方便閱讀，不會影響陳述式的意義。

資料操作陳述式

資料操作陳述式是最常用的 SQL 陳述式。這些陳述式會用來擷取、新增、修改和移除資料庫資料表中的資料。支援的資料操作陳述式如下：SELECT、INSERT、UPDATE 和 DELETE。

SELECT

SELECT 陳述式會用來查詢資料庫。SELECT 的結果是零或多個資料列，其中每一列都有固定數目的欄。結果中的欄數由 SELECT 和選擇性的 FROM 關鍵字之間的 result 欄名稱或運算式清單指定。

```
sql-statement   ::=  SELECT [ALL | DISTINCT] result
                     [FROM table-list]
                     [WHERE expr]
                     [GROUP BY expr-list]
                     [HAVING expr]
                     [compound-op select-statement]*
                     [ORDER BY sort-expr-list]
                     [LIMIT integer [( OFFSET | , ) integer]]
result          ::=  result-column [, result-column]*
result-column   ::=  * | table-name . * | expr [[AS] string]
table-list      ::=  table [ join-op table join-args ]*
table           ::=  table-name [AS alias] |
                     ( select ) [AS alias]
join-op         ::=  , | [NATURAL] [LEFT | RIGHT | FULL] [OUTER | INNER | CROSS] JOIN
join-args       ::=  [ON expr] [USING ( id-list )]
compound-op     ::=  UNION | UNION ALL | INTERSECT | EXCEPT
sort-expr-list  ::=  expr [sort-order] [, expr [sort-order]]*
sort-order      ::=  [COLLATE collation-name] [ASC | DESC]
collation-name  ::=  BINARY | NOCASE
```

任何隨意的運算式都可做為結果。如果結果運算式是 *，則那一個運算式就會由所有資料表的所有欄取代。如果運算式是資料表名稱後面接著 .*，則結果便是該資料表的所有欄。

DISTINCT 關鍵字會導致傳回結果列的子集，這些結果列每個各不相同。NULL 值不會被視為與其它值不同。預設行為會傳回所有結果列，這也可用關鍵字 ALL 明確指定。

查詢執行時，會針對 FROM 關鍵字後指定的一或多個資料表進行查詢。如果用逗號分隔多個資料表名稱，查詢會使用各個資料表的交叉聯結。JOIN 語法也可用來指定資料表的聯結方式。唯一支援的一種外部聯結類型是 LEFT OUTER JOIN。join-args 中的 ON 子句運算式必須解析成 Boolean 值。FROM 子句中可以使用括在括弧中的子查詢做為資料表。整個 FROM 子句可以完全省略，如此一來，結果便是由 result 運算式清單的值所構成的單一列。

WHERE 子句會用來限制查詢擷取的列數。WHERE 子句運算式必須解析成 Boolean 值。WHERE 子句篩選是在任何群組動作之前執行，所以 WHERE 子句運算式不得包含彙總函數。

GROUP BY 子句會導致一或多個結果列在輸出中合併成單一列。當結果包含彙總函數時，GROUP BY 子句會特別好用。GROUP BY 子句中的運算式不一定要是出現在 SELECT 運算式清單中的運算式。

HAVING 子句類似 WHERE，可用來限制陳述式傳回的列。但 HAVING 子句是在 GROUP BY 子句指定的任何群組動作之後套用。因此，HAVING 運算式可以參考包含彙總函數的值。HAVING 子句運算式不一定要出現在 SELECT 清單中。HAVING 運算式和 WHERE 運算式一樣，都必須解析成 Boolean 值。

ORDER BY 子句會導致輸出列經過排序。ORDER BY 子句的 sort-expr-list 引數是做為排序索引鍵的運算式清單。在簡單 SELECT 中，這些運算式不一定要是結果的一部分，但在複合 SELECT (使用其中一個 compound-op 運算子的 SELECT) 中，每個排序運算式都必須確實符合其中一個結果欄。每個排序運算式後面可以選擇性的接著一個 sort-order 子句，內含 COLLATE 關鍵字和用於排列文字的定序函數名稱和 (或) 關鍵字 ASC 或 DESC 以指定排序順序 (遞增或遞減)。sort-order 可以省略，這時會使用預設值 (遞增順序)。如需 COLLATE 子句和定序函數的定義，請參閱 「COLLATE」。

LIMIT 子句會設定結果中傳回的列數上限。負數的 LIMIT 表示沒有上限。LIMIT 後面可以選擇性接著 OFFSET，以指定須略過結果集開頭的多少列。在複合 SELECT 查詢中，LIMIT 子句只能出現在最後的 SELECT 陳述式之後，而且限制適用於整個查詢。請注意，如果 LIMIT 子句中使用 OFFSET 關鍵字，則限制是第一個整數，位移是第二個整數。如果用逗號取代 OFFSET 關鍵字，則位移是第一個數字，而限制是第二個數字。這種看起來很矛盾的語法是為了儘可能與舊式 SQL 資料庫系統相容而故意設計的。

複合 SELECT 是由兩個或多個簡單 SELECT 陳述式組成，陳述式之間可使用 UNION、UNION ALL、INTERSECT 或 EXCEPT 其中一個運算子連接。在複合 SELECT 中，所有組成的 SELECT 陳述式都必須指定相同數目的結果欄。只有在最後的 SELECT 陳述式之後 (和唯一的 LIMIT 子句之前，如果有指定的話) 才能有一個 ORDER BY 子句。UNION 和 UNION ALL 運算子會合併前後 SELECT 陳述式的結果，成為單一資料表。差別在於，UNION 的所有結果列各不相同，而 UNION ALL 的所有結果則可以有重複項目。INTERSECT 運算子會取得前後 SELECT 陳述式結果的交集。EXCEPT 會在移除後一個 SELECT 的結果之後取得前一個 SELECT 的結果。如果將三個或更多 SELECT 陳述式複合連接在一起，它們會從第一個到最後一個群組在一起。

如需允許之運算式的定義，請參閱 「運算式」。

從 AIR 2.5 開始，支援 SQL CAST 運算子讀取，以便將 BLOB 資料轉換成 ActionScript ByteArray 物件。例如，下列程式碼會讀取不是以 AMF 格式儲存的原始資料，並將它儲存在 ByteArray 物件中：

```
stmt.text = "SELECT CAST(data AS ByteArray) AS data FROM pictures;";
stmt.execute();
var result:SQLResult = stmt.getResult();
var bytes:ByteArray = result.data[0].data;
```

INSERT

INSERT 陳述式有兩種基本形式，可用來在資料表中填入資料。

```
sql-statement  ::=  INSERT [OR conflict-algorithm] INTO [database-name.] table-name [(column-list)] VALUES
(value-list) |
                    INSERT [OR conflict-algorithm] INTO [database-name.] table-name [(column-list)] select-
statement
                     REPLACE INTO [database-name.] table-name [(column-list)] VALUES (value-list) |
                     REPLACE INTO [database-name.] table-name [(column-list)] select-statement
```

第一種形式 (使用 VALUES 關鍵字) 會在現有資料表中建立一個新列。如果未指定 column-list，則值的數目必須與資料表中的欄數相同。如果指定 column-list，則值的數目必須符合指定的欄數。資料表中的欄如果未出現在欄清單中，則該欄將填入建立資料表時所定義的預設值，或填入 NULL (如果未定義預設值)。

第二種形式的 INSERT 陳述式會從 SELECT 陳述式取得資料。如果未指定 column-list，SELECT 結果中的欄數必須完全符合資料表的欄數，否則就必須符合 column-list 中指定的欄數。SELECT 結果中的每一列都會成為資料表中的一個新項目。SELECT 可以是簡單或複合形式。如需允許的 SELECT 陳述式定義，請參閱 「SELECT」。

選擇性的 conflict-algorithm 可以指定在這一個命令執行期間使用另一個條件約束衝突解決演算法。如需衝突演算法的說明及定義，請參閱第 960 頁 「特殊陳述式和子句」。

兩個 REPLACE INTO 形式的陳述式相當於使用標準的 INSERT [OR conflict-algorithm] 形式加上 REPLACE 衝突演算法 (即 INSERT OR REPLACE... 形式)。

兩個 REPLACE INTO 形式的陳述式相當於使用標準的 INSERT [OR conflict-algorithm] 形式加上 REPLACE 衝突演算法 (即 INSERT OR REPLACE... 形式)。

UPDATE

更新命令會變更表格中現有的記錄。

```
sql-statement  ::=  UPDATE [database-name.] table-name SET column1=value1, column2=value2,... [WHERE expr]
```

命令是由 UPDATE 關鍵字加上您要更新記錄的表格名稱所組成。在 SET 關鍵字後面，請以逗號分隔的清單，提供欄名稱與要變更的欄值。WHERE 子句運算式提供更新記錄的一或多列。

DELETE

刪除命令會用來移除資料表中的記錄。

```
sql-statement  ::=  DELETE FROM [database-name.] table-name [WHERE expr]
```

命令包含 DELETE FROM 關鍵字，後面接著要從中移除記錄的資料表名稱。

如果沒有 WHERE 子句，就會移除資料表的所有列。如果提供 WHERE 子句，只會移除符合運算式的列。WHERE 子句運算式必須解析成 Boolean 值。如需允許之運算式的定義，請參閱 「運算式」。

資料定義陳述式

資料定義陳述式會用來建立、修改和移除資料庫物件，例如資料表、檢視、索引和觸發程序。支援的資料定義陳述式如下：

- 資料表：
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
- 索引：
  - CREATE INDEX
  - DROP INDEX
- 檢視：
  - CREATE VIEWS
  - DROP VIEWS
- 觸發程序：
  - CREATE TRIGGERS
  - DROP TRIGGERS

CREATE TABLE

CREATE TABLE 陳述式包含關鍵字 CREATE TABLE，後面接著新資料表的名稱，之後則是 (括在括弧裡) 欄定義和條件約束的清單。資料表名稱可為識別名稱或字串。

```
sql-statement       ::=  CREATE [TEMP | TEMPORARY] TABLE [IF NOT EXISTS] [database-name.] table-name
                         ( column-def [, column-def]* [, constraint]* )
sql-statement       ::=  CREATE [TEMP | TEMPORARY] TABLE [database-name.] table-name AS select-statement
column-def          ::=  name [type] [[CONSTRAINT name] column-constraint]*
type                ::=  typename | typename ( number ) | typename ( number , number )
column-constraint   ::=  NOT NULL [ conflict-clause ] |
                         PRIMARY KEY [sort-order] [ conflict-clause ] [AUTOINCREMENT] |
                         UNIQUE [conflict-clause] |
                         CHECK ( expr ) |
                         DEFAULT default-value |
                         COLLATE collation-name
constraint          ::=  PRIMARY KEY ( column-list ) [conflict-clause] |
                         UNIQUE ( column-list ) [conflict-clause] |
                         CHECK ( expr )
conflict-clause     ::=  ON CONFLICT conflict-algorithm
conflict-algorithm  ::=  ROLLBACK | ABORT | FAIL | IGNORE | REPLACE
default-value       ::=  NULL | string | number | CURRENT_TIME | CURRENT_DATE | CURRENT_TIMESTAMP
sort-order          ::=  ASC | DESC
collation-name      ::=  BINARY | NOCASE
column-list         ::=  column-name [, column-name]*
```

每個欄定義是欄名稱後面接著該欄的資料類型，再接著一或多個選擇性的欄條件約束。欄的資料類型會限制該欄中可儲存的資料內容。如果試圖在欄中儲存不同資料類型的值，執行階段將會儘可能將值轉換為適當類型，或引發錯誤。如需其它資訊，請參閱 「資料類型支援」一節。

NOT NULL 欄條件約束表示該欄不能包含 NULL 值。

UNIQUE 條件約束會導致對指定的一或多個欄建立索引。這個索引必須包含唯一的索引鍵，也就是針對指定的一或多個欄，任兩列均不得包含重複值或值組合。CREATE TABLE 陳述式可以有多個 UNIQUE 條件約束，包括在欄定義中為多個欄指定一個 UNIQUE 條件約束，和 (或) 多個資料表層級的 UNIQUE 條件約束。

CHECK 條件約束會定義一個要評估的運算式，而且該運算式必須為 True，才會插入或更新列的資料。CHECK 運算式必須解析成 Boolean 值。

欄定義中的 COLLATE 子句會指定比較欄的文字項目時所使用的文字定序函數。預設狀況下會使用 BINARY 定序函數。如需有關 COLLATE 子句和定序函數的詳細資料，請參閱 「COLLATE」。

DEFAULT 條件約束會指定執行 INSERT 時所用的預設值。這個值可為 NULL、字串常數或數字。預設值也可以是 CURRENT_TIME、CURRENT_DATE 或 CURRENT_TIMESTAMP 這幾個不區分大小寫的特殊關鍵字中的一個。如果值是 NULL、字串常數或數字，每當 INSERT 陳述式未針對該欄指定值時，這個值就會逐字插入欄中。如果值是 CURRENT_TIME、CURRENT_DATE 或 CURRENT_TIMESTAMP，則會將目前的 UTC 日期和 (或) 時間插入欄中。CURRENT_TIME 的格式是 HH:MM:SS。CURRENT_DATE 的格式是 YYYY-MM-DD。CURRENT_TIMESTAMP 的格式是 YYYY-MM-DD HH:MM:SS。

指定 PRIMARY KEY 通常只會在對應的欄上建立 UNIQUE 索引。但如果在具有 INTEGER 資料類型 (或其中一個同義詞，例如 int) 的單一欄上指定 PRIMARY KEY 條件約束，資料庫便會使用該欄做為資料表的實際主索引鍵。這表示該欄只能存放唯一的整數值 (請注意，在許多 SQLite 實作中，只有 INTEGER 欄類型會將欄當做內部主索引鍵使用，但是在 INTEGER 的 Adobe AIR 同義詞 (例如 int) 中，也會指定該行為)。

如果資料表沒有 INTEGER PRIMARY KEY 欄，則插入列時會自動產生整數索引鍵。只要使用 ROWID、OID 或 _ROWID_ 幾個特殊名稱的其中一個，便可以隨時存取列的主索引鍵。不論是明確宣告的 INTEGER PRIMARY KEY 或內部產生的值，都可以使用這些名稱。但是，如果資料表具有明確的 INTEGER PRIMARY KEY，結果資料中的欄名稱便是實際的欄名稱，而不是特殊名稱。

INTEGER PRIMARY KEY 欄也可以包含關鍵字 AUTOINCREMENT。使用 AUTOINCREMENT 關鍵字時，資料庫會在執行沒有明確指定欄值的 INSERT 陳述式時，自動在 INTEGER PRIMARY KEY 欄中產生並插入循序遞增的整數索引鍵。

CREATE TABLE 陳述式中只能有一個 PRIMARY KEY 條件約束。它可以是一個欄定義的一部分，也可以是一個單一資料表層級 PRIMARY KEY 條件約束。主索引鍵欄是隱含的 NOT NULL。

接在許多條件約束後面的選擇性 conflict-clause，可以為該條件約束指定另一個預設的條件約束衝突解決演算法。預設值為 ABORT。同一個資料表中的不同條件約束可以使用不同的預設衝突解決演算法。如果 INSERT 或 UPDATE 陳述式指定不同的衝突解決演算法，就會使用該演算法取代 CREATE TABLE 陳述式中指定的演算法。如需詳細資訊，請參閱第 960 頁「特殊陳述式和子句」的「ON CONFLICT (衝突演算法)」小節。

額外的條件約束 (如 FOREIGN KEY 條件約束) 不會造成錯誤，但執行階段會予以忽略。

如果 CREATE 和 TABLE 之間出現 TEMP 或 TEMPORARY 關鍵字，則所建立的資料表只會顯示在同一個資料庫連線 (SQLConnection 實體) 中。關閉資料庫連線時，就會自動刪除該資料表。在暫存資料表上建立的任何索引也都是暫時性的。暫存資料表和索引儲存在與主要資料庫檔案不同的另一個檔案中。

如果指定選擇性的 database-name 前置詞，就會在具名資料庫 (以指定的資料庫名稱呼叫 attach() 方法而連接到 SQLConnection 實體的資料庫) 中建立該資料表。除非 database-name 前置詞是 temp，否則，同時指定 database-name 前置詞和 TEMP 關鍵字是錯誤的。如果未指定資料庫名稱，而且沒有 TEMP 關鍵字，則會在主要資料庫 (使用 open() 或 openAsync() 方法連接到 SQLConnection 實體的資料庫) 中建立該資料表。

欄數或資料表的條件約束數目可隨意自訂且沒有限制，另外，列中的資料數量也可隨意自訂且沒有限制。

CREATE TABLE AS 形式會以查詢結果集的形式定義資料表。資料表欄的名稱是結果中欄的名稱。

如果使用選擇性的 IF NOT EXISTS 子句，而且已經有同名的其它資料表，則資料庫會忽略 CREATE TABLE 命令。

使用 DROP TABLE 陳述式可以移除資料表，而使用 ALTER TABLE 陳述式可以進行有限的變更。

ALTER TABLE

ALTER TABLE 命令可讓使用者將現有資料表重新命名或加入新欄。資料表中的欄無法移除。

```
sql-statement ::= ALTER TABLE [database-name.] table-name alteration
alteration    ::= RENAME TO new-table-name
alteration    ::= ADD [COLUMN] column-def
```

RENAME TO 語法會用來將以 [database-name.] table-name 識別的資料表重新命名成 new-table-name。這個命令無法用來在附加的資料庫之間移動資料表，只能重新命名相同資料庫中的資料表。

如果重新命名的資料表有觸發程序或索引，它們在重新命名後仍會附加到資料表。但如果有任何檢視定義或由觸發程序執行的陳述式參考到重新命名的資料表，它們不會自動修改為使用新資料表名稱。如果重新命名的資料表有關聯的檢視或觸發程序，您必須手動卸除並重新建立觸發程序，或使用新的資料表名稱檢視定義。

ADD [COLUMN] 語法會用來在現有資料表中加入新欄。新欄一定會附加到現有欄清單結束處。column-def 子句可以接受 CREATE TABLE 陳述式中允許的任何形式，但有下列限制：

- 欄不可以有 PRIMARY KEY 或 UNIQUE 條件約束。
- 欄不可以有 CURRENT_TIME、CURRENT_DATE 或 CURRENT_TIMESTAMP 的預設值。
- 如果指定 NOT NULL 條件約束，該欄必須有 NULL 以外的預設值。

ALTER TABLE 陳述式的執行時間不受資料表中的資料量影響。

DROP TABLE

DROP TABLE 陳述式會移除以 CREATE TABLE 陳述式加入的資料表。卸除的資料表是具有指定 table-name 的資料表。它會從資料庫和磁碟檔案中完全移除。資料表無法復原。與資料表相關聯的所有索引也都會刪除。

```
sql-statement  ::=  DROP TABLE [IF EXISTS] [database-name.] table-name
```

根據預設，DROP TABLE 陳述式不會使資料庫檔案變小。資料庫中的空白空間仍會保留，並供之後的 INSERT 作業使用。若要移除資料庫中的可用空間，請使用 SQLConnection.clean() 方法。如果最初建立資料庫時，將 autoClean 參數設定為 true，則會自動釋放空間。

選擇性的 IF EXISTS 子句會抑制資料庫不存在時所發生的錯誤。

CREATE INDEX

CREATE INDEX 命令包含關鍵字 CREATE INDEX，後面接著新索引的名稱、關鍵字 ON、要編製索引的先前已建立資料表名稱，和括在括弧裡的欄名稱清單 (清單中所列是資料表中要做為索引鍵值的欄名稱)。

```
sql-statement  ::=  CREATE [UNIQUE] INDEX [IF NOT EXISTS] [database-name.] index-name
                    ON table-name ( column-name [, column-name]* )
column-name    ::=  name [COLLATE collation-name] [ASC | DESC]
```

每個欄名稱後面可以接著 ASC 或 DESC 關鍵字以指定排序順序，但執行階段會忽略指定的排序順序。排序方式固定採用遞增順序。

每個欄名稱後面的 COLLATE 子句會定義該欄中文字值所用的定序順序。預設的定序順序是 CREATE TABLE 陳述式中為該欄定義的定序順序。如果未指定定序順序，則使用 BINARY 定序順序。如需 COLLATE 子句和定序函數的定義，請參閱「COLLATE」。

單一資料表可附加的索引數目可隨意自訂且沒有限制，一個索引中的欄數也沒有限制。

DROP INDEX

DROP INDEX 陳述式會移除使用 CREATE INDEX 陳述式所加入的索引。指定的索引會從資料庫檔案中完全移除。唯一一個復原索引的方法是重新輸入適當的 CREATE INDEX 命令。

```
sql-statement ::= DROP INDEX [IF EXISTS] [database-name.] index-name
```

根據預設，DROP INDEX 陳述式不會使資料庫檔案變小。資料庫中的空白空間仍會保留，並供之後的 INSERT 作業使用。若要移除資料庫中的可用空間，請使用 SQLConnection.clean() 方法。如果最初建立資料庫時，將 autoClean 參數設定為 true，則會自動釋放空間。

CREATE VIEW

CREATE VIEW 命令會為預先定義的 SELECT 陳述式指定名稱。然後，這個新名稱就可以用在另一個 SELECT 陳述式的 FROM 子句中，取代資料表名稱。檢視常用來簡化查詢，可將複雜 (且常用) 的資料集合併成結構，以供其它作業使用。

```
sql-statement ::= CREATE [TEMP | TEMPORARY] VIEW [IF NOT EXISTS] [database-name.] view-name AS select-
statement
```

如果 CREATE 和 VIEW 之間出現 TEMP 或 TEMPORARY 關鍵字，則所建立的檢視只會顯示在開啟資料庫的 SQLConnection 實體中，而且關閉資料庫時就會自動刪除。

如果指定 [database-name]，系統會在具名資料庫 (使用 attach() 方法以指定的 name 引數連接到 SQLConnection 實體的資料庫) 中建立該檢視。除非 [database-name] 是 temp，否則，同時指定 [database-name] 和 TEMP 關鍵字是錯誤的。如果未指定資料庫名稱，而且沒有 TEMP 關鍵字，則會在主要資料庫 (使用 open() 或 openAsync() 方法連接到 SQLConnection 實體的資料庫) 中建立該檢視。

檢視是唯讀的，所以 DELETE、INSERT 或 UPDATE 陳述式不能用在檢視上，除非至少定義一個相關類型 (INSTEAD OF DELETE、INSTEAD OF INSERT、INSTEAD OF UPDATE) 的觸發程序。如需為檢視建立觸發程序的詳細資訊，請參閱「CREATE TRIGGER」。

使用 DROP VIEW 陳述式可從資料庫中移除檢視。

DROP VIEW

DROP VIEW 陳述式會移除 CREATE VIEW 陳述式所建立的檢視。

```
sql-statement ::= DROP VIEW [IF EXISTS] view-name
```

指定的 view-name 是要卸除的檢視名稱。該檢視會從資料庫移除，但基礎資料表中的任何資料都不會修改。

CREATE TRIGGER

CREATE TRIGGER 陳述式會用來在資料庫結構中加入觸發程序。觸發程序是當指定的資料庫事件 (database-event) 發生時，會自動執行的資料庫作業 (trigger-action)。

```
sql-statement   ::=  CREATE [TEMP | TEMPORARY] TRIGGER [IF NOT EXISTS] [database-name.] trigger-name
                     [BEFORE | AFTER] database-event
                     ON table-name
                     trigger-action
sql-statement   ::=  CREATE [TEMP | TEMPORARY] TRIGGER [IF NOT EXISTS] [database-name.] trigger-name
                     INSTEAD OF database-event
                     ON view-name
                     trigger-action
database-event  ::=  DELETE |
                     INSERT |
                     UPDATE |
                     UPDATE OF column-list
trigger-action  ::=  [FOR EACH ROW] [WHEN expr]
                     BEGIN
                       trigger-step ;
                       [ trigger-step ; ]*
                     END
trigger-step    ::=  update-statement |
                     insert-statement |
                     delete-statement |
                     select-statement
column-list     ::=  column-name [, column-name]*
```

觸發程序是指定為每當特定資料庫資料表發生 DELETE、INSERT 或 UPDATE 時就會引發，或每當資料表的一或多個指定欄的 UPDATE 發生更新時就會引發。除非使用 TEMP 或 TEMPORARY 關鍵字，否則觸發程序是永久性的。在這種情況下，每當 SQLConnection 實體的主要資料庫連線關閉時，就會移除該觸發程序。如果未指定時間 (BEFORE 或 AFTER)，觸發程序會預設為 BEFORE。

由於只支援 FOR EACH ROW 觸發程序，所以 FOR EACH ROW 文字是選擇性的。使用 FOR EACH ROW 觸發程序時，如果 WHEN 子句陳述式評估為 true，就會針對導致引發觸發程序之陳述式所插入、更新或刪除的每個資料庫列執行 trigger-step 陳述式。

如果提供 WHEN 子句，指定為觸發程序步驟的 SQL 陳述式只會針對 WHEN 子句為 True 的列執行。如果未提供 WHEN 子句，則會針對所有列執行這些 SQL 陳述式。

在觸發程序主體 (trigger-action 子句) 內，使用特殊的資料表名稱 OLD 和 NEW，可取得受影響資料表的變更前和變更後值。OLD 和 NEW 資料表的結構符合建立觸發程序之資料表的結構。OLD 資料表包含所觸發的陳述式修改或刪除的任何列，且為執行觸發陳述式作業之前的狀態。NEW 資料表句含所觸發的陳述式修改或建立的任何列，且為執行觸發陳述式作業之後的狀態。WHEN 子句和 trigger-step 陳述式都可以使用 NEW.column-name 和 OLD.column-name 形式的參考從插入、刪除或更新的列中存取值，其中 column-name 是觸發程序相關資料表中的欄名稱。能否使用 OLD 和 NEW 資料表參考需要視觸發程序處理的 database-event 類型而定：

- INSERT – NEW 參考有效
- UPDATE – NEW 和 OLD 參考有效
- DELETE – OLD 參考有效

指定的時間 (BEFORE、AFTER 或 INSTEAD OF) 會決定 trigger-step 陳述式相對於插入、修改或移除相關列的執行時間。在 UPDATE 或 INSERT 陳述式的 trigger-step 中可以指定 ON CONFLICT 子句。但如果在導致引發觸發程序的陳述式中指定 ON CONFLICT 子句，則會改用該衝突處理原則。

除了資料表觸發程序外，您也可以在檢視上建立 INSTEAD OF 觸發程序。如果在檢視上定義一或多個 INSTEAD OF INSERT、INSTEAD OF DELETE 或 INSTEAD OF UPDATE 觸發程序，則在檢視上執行相關類型的陳述式 (INSERT、DELETE 或 UPDATE) 並不是錯誤。在這種情況下，在檢視上執行 INSERT、DELETE 或 UPDATE 會導致引發相關的觸發程序。因為觸發程序是 INSTEAD OF 觸發程序，所以導致引發觸發程序的陳述式不會修改檢視的基礎資料表。但是，您可以使用觸發程序對基礎資料表執行修改作業。

在具有 INTEGER PRIMARY KEY 欄的資料表上建立觸發程序時，必須記住一個重點。如果 BEFORE 觸發程序修改了某列的 INTEGER PRIMARY KEY 欄，但導致引發觸發程序的陳述式原本即將修改該列，則更新不會發生。暫時解決方法是建立具有 PRIMARY KEY 欄的資料表，而非具有 INTEGER PRIMARY KEY 欄的資料表。

使用 DROP TRIGGER 陳述式可以移除觸發程序。卸除資料表或檢視時，與該資料表或檢視相關聯的所有觸發程序也都會自動卸除。

RAISE() 函數

特殊的 SQL 函數 RAISE() 可以使用在觸發程序的 trigger-step 陳述式中。這個函數的語法如下：

```
raise-function  ::=  RAISE ( ABORT, error-message ) |
                     RAISE ( FAIL, error-message ) |
                     RAISE ( ROLLBACK, error-message ) |
                     RAISE ( IGNORE )
```

在執行觸發程序期間，如果呼叫前三種形式之一，系統會執行指定的 ON CONFLICT 處理動作 (ABORT、FAIL 或 ROLLBACK)，並結束目前陳述式的執行。由於 ROLLBACK 表示陳述式執行失敗，所以執行 execute() 方法的 SQLStatement 實體會傳送 error (SQLErrorEvent.ERROR) 事件。在傳送的事件物件中，error 屬性的 SQLError 物件會將其 details 屬性設定為 RAISE() 函數中指定的 error-message。

呼叫 RAISE(IGNORE) 時，目前觸發程序的其餘部分、導致執行觸發程序的陳述式，以及原本接著會執行的任何觸發程序都會放棄。不會回復任何資料庫變更。如果導致執行觸發程序的陳述式本身是某個觸發程序的一部分，則該觸發程序程式會從下一個步驟開頭繼續執行。如需有關衝突解決演算法的詳細資訊，請參閱 「ON CONFLICT (衝突演算法)」一節。

DROP TRIGGER

DROP TRIGGER 陳述式會移除 CREATE TRIGGER 陳述式所建立的觸發程序。

```
sql-statement  ::=  DROP TRIGGER [IF EXISTS] [database-name.] trigger-name
```

觸發程序會從資料庫中移除。請注意，與觸發程序相關的資料表卸除時，該觸發程序就會自動卸除。

特殊陳述式和子句

本節說明執行階段為 SQL 提供的幾個擴充子句，以及可用在許多陳述式、註解和運算式中的兩個語言元素。本節中的元素包括：

COLLATE

COLLATE 子句會用在 SELECT、CREATE TABLE 和 CREATE INDEX 陳述式中，以指定比較和排序值時所用的比較演算法。

```
sql-statement   ::=  COLLATE collation-name
collation-name  ::=  BINARY | NOCASE
```

欄的預設定序類型是 BINARY。將 BINARY 定序用於 TEXT 儲存類別的值時，二進位定序的執行方式是比較記憶體中代表值的位元組，而不管其文字編碼。

NOCASE 定序順序只適用於 TEXT 儲存類別的值。使用時，NOCASE 定序會執行不區分大小寫的比較。

NULL、BLOB、INTEGER 或 REAL 類型的儲存類別不會使用任何定序順序。

若要針對欄使用 BINARY 以外的定序類型，您必須在 CREATE TABLE 陳述式的欄定義中指定 COLLATE 子句。比較兩個 TEXT 值時，會根據下列規則，使用定序順序來決定比較的結果：

- 對於二進位比較運算子，如果任一個運算元是欄，則欄的預設定序類型會決定比較所用的定序順序。如果兩個運算元都是欄，則左側運算元的定序類型會決定所用的定序順序。如果兩個運算元都不是欄，則使用 BINARY 定序順序。
- BETWEEN...AND 運算子相當於使用包含 >= 和 <= 運算子的兩個運算式。例如，運算式 x BETWEEN y AND z 相當於 x >= y AND x <= z。因此，BETWEEN...AND 運算子會遵循上述規則來決定定序順序。
- IN 運算子的行為類似 = 運算子，用途在於決定要使用的定序順序。例如，如果 x 是欄，則運算式 x IN (y, z) 所用的定序順序是 x 的預設定序順序。否則，會使用 BINARY 定序。
- SELECT 陳述式中的 ORDER BY 子句可明確指定排序作業所使用的定序順序。在這種情況下，一定會使用明確的定序順序。否則，如果 ORDER BY 子句所排序的運算式是欄，則會使用欄的預設定序類型來決定排序順序。如果運算式不是欄，則使用 BINARY 定序順序。

EXPLAIN

EXPLAIN 命令修飾詞是 SQL 的非標準擴充。

```
sql-statement  ::=  EXPLAIN sql-statement
```

如果 EXPLAIN 關鍵字出現在其它任何 SQL 陳述式之前，則結果是不實際執行命令，而是報告在沒有 EXPLAIN 關鍵字的情況下，將用來執行命令的虛擬機器指令順序。EXPLAIN 功能屬於進階功能，讓開發人員可以變更 SQL 陳述式文字，以嘗試最佳化效能，或針對似乎無法正常運作的陳述式進行除錯。

ON CONFLICT (衝突演算法)

ON CONFLICT 子句不是單獨的 SQL 命令。它是可能出現在許多其它 SQL 命令中的非標準子句。

```
conflict-clause     ::=  ON CONFLICT conflict-algorithm
conflict-clause     ::=  OR conflict-algorithm
conflict-algorithm  ::=  ROLLBACK |
                         ABORT |
                         FAIL |
                         IGNORE |
                         REPLACE
```

第一種形式的 ON CONFLICT 子句會使用關鍵字 ON CONFLICT，並用在 CREATE TABLE 陳述式中。INSERT 或 UPDATE 陳述式使用第二種形式，並用 OR 取代 ON CONFLICT 使語法看起來比較自然。例如，陳述式不會是 INSERT ON CONFLICT IGNORE，而是 INSERT OR IGNORE。雖然關鍵字不同，但這兩種形式的子句有著相同的意思。

ON CONFLICT 子句會指定用來解析條件約束衝突的演算法。五個演算法分別為 ROLLBACK、ABORT、FAIL、IGNORE 和 REPLACE。預設演算法為 ABORT。五個衝突演算法的說明如下：

ROLLBACK 違反條件約束時，會立即進行 ROLLBACK，結束目前的交易。命令中止，而且 SQLStatement 實體會傳送 error 事件。如果沒有作用中交易 (除了每個命令建立的隱含交易之外)，則這個演算法的作用與 ABORT 一樣。

ABORT 違反條件約束時，命令會撤銷它之前已完成的任何變更，而且 SQLStatement 實體會傳送 error 事件。由於 ROLLBACK 不會執行，因此同一個交易中之前的命令所做的變更仍會保留。ABORT 是預設行為。

FAIL 違反條件約束時，命令會中止，而且 SQLStatement 會傳送 error 事件。但是，陳述式在違反條件約束之前對資料庫所做的變更仍會保留，不會撤銷。例如，如果 UPDATE 陳述式在它嘗試更新的第 100 列發生違反條件約束，則前 99 列的變更仍會保留，但從第 100 列開始的變更都不會發生。

IGNORE 違反條件約束時，不會插入或變更包含違反條件約束的那一列。除了忽略的這一列外，命令會繼續正常執行。包含違反條件約束的列前後的列仍會繼續正常插入或更新。不會傳回錯誤。

REPLACE 違反 UNIQUE 條件約束時，會先移除導致違反條件約束的預先存在列，再插入或更新目前列。因此，插入或更新一定會發生，而且命令會繼續正常執行。不會傳回錯誤。若違反 NOT NULL 條件約束，NULL 值便會由該欄的預設值取代。若欄沒有預設值，便會使用 ABORT 演算法。若違反 CHECK 條件約束，便會使用 IGNORE 演算法。當這個衝突解決策略刪除列以滿足條件約束時，它不會在這些列上叫用刪除觸發程序。

在 INSERT 或 UPDATE 陳述式的 OR 子句中指定的演算法，會覆寫 CREATE TABLE 陳述式中指定的任何演算法。如果 CREATE TABLE 陳述式或執行 INSERT 或 UPDATE 陳述式中沒有指定演算法，便會使用 ABORT 演算法。

REINDEX

REINDEX 命令會用來刪除並重新建立一或多個索引。這個命令在定序順序的定義變更時很好用。

```
sql-statement  ::=  REINDEX collation-name
sql-statement  ::=  REINDEX [database-name .] ( table-name | index-name )
```

使用第一種形式時，使用具名定序順序的所有附加資料庫中的所有索引都會重新建立。使用第二種形式時，若指定 table-name，與該資料表相關的所有索引都會重建。如果提供 index-name，便只會刪除及重新建立指定的索引。

註解

註解不是 SQL 命令，但可以出現在 SQL 查詢中。執行階段會把它們視為空白。可以使用空白的位置都可以開始註解，包括跨越數行的運算式內部。

```
comment             ::=  single-line-comment |
                         block-comment
single-line-comment ::=  -- single-line
block-comment       ::=  /* multiple-lines or block [*/]
```

單行註解會以兩個虛線表示。單行註解的範圍只到目前行結束處。

區塊註解可以跨越任何行數，或內嵌在單一行中。如果沒有終止分隔符號，區塊註解的範圍會延伸至輸入結束處。這種狀況不會被視為錯誤。新的 SQL 陳述式可以在區塊註解結束後的下一行開始。區塊註解可以內嵌在可出現空白的任何位置，包括運算式內部和其它 SQL 陳述式中間。區塊註解無法採用巢狀方式。區塊註解內部的單行註解會遭到忽略。

運算式

運算式是其它 SQL 區塊中的子命令。SQL 陳述式中運算式的有效語法說明如下：

```
expr            ::=  expr binary-op expr |
                     expr [NOT] like-op expr [ESCAPE expr] |
                     unary-op expr |
                     ( expr ) |
                     column-name |
                     table-name.column-name |
                     database-name.table-name.column-name |
                     literal-value |
                     parameter |
                     function-name( expr-list | * ) |
                     expr ISNULL |
                     expr NOTNULL |
                     expr [NOT] BETWEEN expr AND expr |
                     expr [NOT] IN ( value-list ) |
                     expr [NOT] IN ( select-statement ) |
                     expr [NOT] IN [database-name.] table-name |
                     [EXISTS] ( select-statement ) |
                     CASE [expr] ( WHEN expr THEN expr )+ [ELSE expr] END |
                     CAST ( expr AS type ) |
                     expr COLLATE collation-name
like-op         ::=  LIKE | GLOB
binary-op       ::=  see Operators
unary-op        ::=  see Operators
parameter       ::=  :param-name | @param-name | ?
value-list      ::=  literal-value [, literal-value]*
literal-value   ::=  literal-string | literal-number | literal-boolean | literal-blob | literal-null
literal-string  ::=  'string value'
literal-number  ::=  integer | number
literal-boolean  ::=  true | false
literal-blob  ::=  X'string of hexadecimal data'
literal-null  ::=  NULL
```

運算式是值和運算子的任何組合，而且可解析成單一值。運算式一般可分為兩種類型，一種可解析成 Boolean (True 或 False) 值，一種可解析成非 Boolean 值。

在許多常見狀況中 (包括在 WHERE 子句、HAVING 子句、JOIN 子句中的 ON 運算式和 CHECK 運算式)，運算式必須解析成 Boolean 值。符合這種狀況的運算式類型如下：

- ISNULL
- NOTNULL
- IN ()
- EXISTS ()
- LIKE
- GLOB
- 特定函數
- 特定運算子 (尤其是比較運算子)

常值

常值數值會寫成整數或浮點數，且支援科學記號。小數點固定使用 . (句點) 字元。

字串常值會以字串前後加上單引號 ' 表示。若要在字串中包含單引號，須依照範例連續使用兩個單引號：''。

Boolean 常值會以值 true 或 false 表示。常值 Boolean 值用於 Boolean 欄資料類型。

BLOB 常值是一種字串常值，它包含十六進位資料，前面加上一個 x 或 X 字元，例如 X'53514697465'。

常值也可以是字符 NULL。

欄名稱

欄名稱可以是 CREATE TABLE 陳述式中定義的任何名稱，或下列特殊識別名稱：ROWID、OID 或 _ROWID_。這些特殊識別名稱全都描述與每個資料表中每個列相關的唯一隨機整數索引鍵 (「列索引鍵」)。這些特殊識別名稱只有在 CREATE TABLE 陳述式未定義同名的真實欄時才會參考到列索引鍵。列索引鍵的作用類似唯讀欄。凡是可以使用一般欄的位置都可以使用列索引鍵，唯一的例外是您無法在 UPDATE 或 INSERT 陳述式中變更列索引鍵的值。SELECT * FROM table 陳述式不會在結果集中包含列索引鍵。

SELECT 陳述式

SELECT 陳述式可出現在運算式中，做為 IN 運算子右側的運算元、純量 (單一結果值) 或 EXISTS 運算子的運算元。做為純量或 IN 運算子的運算元使用時，SELECT 的結果中只能有一欄，但可以使用複合 SELECT 陳述式 (用 UNION 或 EXCEPT 關鍵字連接)。使用 EXISTS 運算子時，運算式會忽略 SELECT 的結果集中的欄，並在有一或多個欄存在時傳回 TRUE，在結果集空白時傳回 FALSE。如果 SELECT 運算式中沒有條件參考到包含查詢中的值，則運算式會在其它任何處理之前評估一次，並視需要重複使用該結果。如果 SELECT 運算式不包含外部查詢中的變數 (稱為相關子查詢)，則每次需要時都會重新評估 SELECT。

當 SELECT 是 IN 運算子的右側運算元時，如果左側運算元的結果等於 SELECT 陳述式結果集中的任何值時，IN 運算子會傳回 TRUE。IN 運算子前面可以加上 NOT 關鍵字以反轉測試的意義。

當 SELECT 出現在運算式中，但不是 IN 運算子的右側運算元時，SELECT 結果中的第一列會成為運算式中使用的值。如果 SELECT 產生多個結果列，則會忽略第一列之後的所有列。如果 SELECT 產生零列，則 SELECT 的值是 NULL。

CAST 運算式

CAST 運算式會將指定值的資料類型變更成指定的資料類型。指定的類型可以是任何非空白類型名稱，但必須是 CREATE TABLE 陳述式的欄定義中有效的類型。如需詳細資訊，請參閱「資料類型支援」。

額外的運算式元素

下列 SQL 元素也可以用在運算式中：

- 內建函數：彙總函數、純量函數以及日期和時間格式函數
- 運算子
- 參數

內建函數

內建函數分為三個主要類別：

- 彙總函數
- 純量函數
- 日期和時間函數

除了這些函數之外，還有一個特殊函數 RAISE() 會用來在執行觸發程序時提供錯誤通知。這個函數只可以用在 CREATE TRIGGER 陳述式主體內。如需有關 RAISE() 函數的詳細資訊，請參閱「CREATE TRIGGER > RAISE()」。

就像 SQL 中的所有關鍵字一樣，函數名稱不區分大小寫。

彙總函數

彙總函數會對多列的值執行作業。這些函數主要用在 SELECT 陳述式中搭配 GROUP BY 子句一起使用。

AVG(X)	傳回群組中所有非 NULL X 的平均值。字串和看起來不是數字的 BLOB 值都會解譯為 0。AVG() 的結果一定是浮點值，即使所有輸入都是整數。
COUNT(X) COUNT(*)	第一種形式會傳回群組中 X 不是 NULL 的次數。第二種形式 (使用 * 引數) 會傳回群組中的總列數。

MAX(X)	傳回群組中所有值的最大值。最大值依一般排序順序決定。
MIN(X)	傳回群組中所有值的最小非 NULL 值。最小值依一般排序順序決定。如果群組中的所有值都是 NULL，則傳回 NULL。
SUM(X)	傳回群組中所有非 NULL 值的數值總和。如果所有值都是 NULL，SUM() 會傳回 NULL，
TOTAL(X)	TOTAL() 則傳回 0.0。TOTAL() 的結果一定是浮點值。如果所有非 NULL 輸入都是整數，SUM() 的結果是整數值。如果 SUM() 的任何輸入不是整數也不是 NULL，則 SUM() 會傳回浮點值。這個值可能是正確總和的近似值。

在上述使用單一引數的任何彙總函數中，該引數前都可以加上關鍵字 DISTINCT。在這種情況下，會先篩選掉重複項目，再傳給彙總函數。例如，函數呼叫 COUNT(DISTINCT x) 會傳回欄 X 中不同值的數目，而不是欄 x 中非 NULL 值的總數。

純量函數

純量函數會一次處理一列的值。

ABS(X)	傳回引數 X 的絕對值。
COALESCE(X, Y, ...)	傳回第一個非 NULL 引數的複本。如果所有引數都是 NULL，則傳回 NULL。函數必須至少有兩個引數。
GLOB(X, Y)	這個函數會用來實作 X GLOB Y 語法。
IFNULL(X, Y)	傳回第一個非 NULL 引數的複本。如果兩個引數都是 NULL，則傳回 NULL。這個函數的作用與 COALESCE() 相同。
HEX(X)	將引數解譯為 BLOB 儲存類型的值。結果是以十六進位呈現該值的內容。
LAST_INSERT_ROWID()	傳回透過目前 SQLConnection 插入資料庫的最後一列的列識別名稱 (產生的主索引鍵)。這個值與 SQLConnection.lastInsertRowID 屬性傳回的值一樣。
LENGTH(X)	傳回 X 的字串長度，以字元為單位。
LIKE(X, Y [, Z])	這個函數用來實作 SQL 的 X LIKE Y [ESCAPE Z] 語法。如果出現選擇性的 ESCAPE 子句，函數會以三個引數叫用。否則，只會以兩個引數叫用。
LOWER(X)	傳回字串 X 的複本，而且所有字元都轉換為小寫。
LTRIM(X) LTRIM(X, Y)	傳回字串，但移除 X 左側的空白。如果指定 Y 引數，函數會從 X 左側移除 Y 中的任何字元。
MAX(X, Y, ...)	傳回具最大值的引數。引數可以是字串加上數字。最大值由定義的排序順序決定。請注意，當 MAX() 有 2 個或更多引數時，它是簡單函數，但當它只有一個引數時，則是彙總函數。
MIN(X, Y, ...)	傳回具最小值的引數。引數可以是字串加上數字。最小值由定義的排序順序決定。請注意，當 MIN() 有 2 個或更多引數時，它是簡單函數，但當它只有一個引數時，則是彙總函數。
NULLIF(X, Y)	當兩個引數不同時傳回第一個引數，否則傳回 NULL。
QUOTE(X)	這個常式會傳回其引數值的字串，成為適合納入其它 SQL 陳述式的形式。字串前後會加上單引號，並視需要跳脫內部的引號。BLOB 儲存類別會編碼為十六進位常值。當撰寫觸發程序以實作復原 / 重做功能時，這個函數就很好用。
RANDOM(*)	傳回介於 -9223372036854775808 和 9223372036854775807 之間的虛擬亂數整數。這個隨機值不是強式加密。
RANDOMBLOB(N)	傳回包含虛擬亂數位元組的 N 位元組 BLOB。N 應該是正整數。這個隨機值不是強式加密。如果 N 的值是負數，則傳回一個位元組。
ROUND(X) ROUND(X, Y)	將數字 X 捨去到小數點右邊第 Y 位。如果省略 Y 引數，就使用 0。
RTRIM(X) RTRIM(X, Y)	傳回字串，但移除 X 右側的空白。如果指定 Y 引數，函數會從 X 右側移除 Y 中的任何字元。
SUBSTR(X, Y, Z)	傳回輸入字串 X 從第 Y 個字元開始的子字串，長度為 Z 個字元。X 最左邊的字元位於索引位置 1。如果 Y 是負數，子字串的第一個字元由右側算起，而不是由左側算起。
TRIM(X) TRIM(X, Y)	傳回字串，但移除 X 右側的空白。如果指定 Y 引數，函數會從 X 右側移除 Y 中的任何字元。
TYPEOF(X)	傳回運算式 X 的類型。可能的傳回值為 'null'、'integer'、'real'、'text' 和 'blob'。如需有關資料類型的詳細資訊，請參閱「資料類型支援」。
UPPER(X)	傳回輸入字串 X 的複本，而且轉換為全部大寫字母。
ZEROBLOB(N)	傳回包含 N 個位元組 0x00 的 BLOB。

日期和時間格式函數

日期和時間格式函數是一組純量函數，用來建立格式化的日期和時間資料。請注意，這些函數會操作和傳回字串和數值。這些函數無法用於 DATE 資料類型。如果在宣告資料類型為 DATE 的欄資料上使用這些函數，函數將無法依預期方式運作。

DATE(T, ...)	DATE() 函數會傳回字串，內含以下格式的日期：YYYY-MM-DD。第一個參數 (T) 會指定 「時間格式」下方所列格式的時間字串。時間字串後可指定任何數目的修飾詞。修飾詞則列在 「修飾詞」下方。
TIME(T, ...)	TIME() 函數會傳回字串，內含 HH:MM:SS 格式的時間。第一個參數 (T) 會指定 「時間格式」下方所列格式的時間字串。時間字串後可指定任何數目的修飾詞。修飾詞則列在 「修飾詞」下方。
DATETIME(T, ...)	DATETIME() 函數會傳回字串，內含 YYYY-MM-DD HH:MM:SS 格式的日期和時間。第一個參數 (T) 會指定 「時間格式」下方所列格式的時間字串。時間字串後可指定任何數目的修飾詞。修飾詞則列在 「修飾詞」下方。
JULIANDAY(T, ...)	JULIANDAY() 函數會傳回數字，指出從西元前 4714 年 11 月 24 日格林威治時間中午直到提供的日期之間經過的日數。第一個參數 (T) 會指定 「時間格式」下方所列格式的時間字串。時間字串後可指定任何數目的修飾詞。修飾詞則列在 「修飾詞」下方。
STRFTIME(F, T, ...)	STRFTIME() 常式會根據第一個引數 F 指定的格式字串，傳回設定格式的日期。格式字串支援下列替換方式： %d - 月份中的日期 %f - 小數秒 SS.SSS %H - 小時 00-24 %j - 年中的日期 001-366 %J - 凱撒曆日數 %m - 月份 01-12 %M - 分鐘 00-59 %s - 自 1970-01-01 迄今的秒數 %S - 秒 00-59 %w - 一週的日期 0-6 (星期日 = 0) %W - 年中的週數 00-53 %Y - 年份 0000-9999 %% - % 第二個參數 (T) 會指定 「時間格式」下方所列格式的時間字串。時間字串後可指定任何數目的修飾詞。修飾詞則列在 「修飾詞」下方。

時間格式

時間字串可為下列格式中任一項：

YYYY-MM-DD	2007-06-15
YYYY-MM-DD HH:MM	2007-06-15 07:30
YYYY-MM-DD HH:MM:SS	2007-06-15 07:30:59
YYYY-MM-DD HH:MM:SS.SSS	2007-06-15 07:30:59.152
YYYY-MM-DDTHH:MM	2007-06-15T07:30
YYYY-MM-DDTHH:MM:SS	2007-06-15T07:30:59
YYYY-MM-DDTHH:MM:SS.SSS	2007-06-15T07:30:59.152
HH:MM	07:30 (日期是 2000-01-01)
HH:MM:SS	07:30:59 (日期是 2000-01-01)
HH:MM:SS.SSS	07:30:59:152 (日期是 2000-01-01)
now	全球時間 (Universal Coordinated Time) 的目前日期和時間。
DDDD.DDDD	凱撒曆日數，以浮點數表示。

字元 T 在這些格式中是分隔日期和時間的常值字元 "T"。只包含時間的格式會假設日期為 2001-01-01。

修飾詞

時間字串後面可接著零或多個修飾詞，以改變日期或改變日期的解譯方式。可用修飾詞如下：

NNN days	要加至時間的天數。
NNN hours	要加至時間的小時數。
NNN minutes	要加至時間的分鐘數。
NNN.NNNN seconds	要加至時間的秒數和毫秒數。
NNN months	要加至時間的月數。
NNN years	要加至時間的年數。
start of month	將時間向後推到當月月初。
start of year	將時間向後推到該年年初。
start of day	將時間向後推到當天開始時。
weekday N	將時間向前推到指定的星期幾 (0 = 星期日、1 = 星期一，依此類似)。
localtime	將日期轉換為當地時間。
utc	將日期轉換為全球時間。

運算子

SQL 支援大量運算子，包括大部分程式語言都提供的一般運算子，以及數個 SQL 專用的運算子。

一般運算子

SQL 區塊中可使用下列二元運算子，清單中依其優先順序由高而低排列：

```
*    /    %
+    -
<< >> & |
< >=   > >=
=    ==   !=   <> IN
AND
OR
```

支援的一元前置運算子包括：

```
 !    ~    NOT
```

COLLATE 運算子可視為一元前置運算子。COLLATE 運算子的優先順序最高。它的繫結緊密程度一定高於其它任何前置一元運算子或任何二元運算子。

請注意，等於和不等於運算子有兩種變形。等於可以用 = 或 == 表示。不等於運算子可以用 != 或 <> 表示。

|| 運算子是字串連接運算子，可將運算元的兩個字串接在一起。

運算子 % 會輸出其左側運算元除以右側運算元的餘數。

任何二元運算子的結果都是數值，只有 || 連接運算子例外，它的結果是字串。

SQL 運算子

LIKE

LIKE 運算子會執行樣式符合比較。

```
expr     ::=  (column-name | expr) LIKE pattern
pattern  ::=  '[ string | % | _ ]'
```

LIKE 運算子右側的運算元包含樣式，而左側的運算元則包含要符合樣式的字串。樣式中的百分比符號 (%) 是萬用字元，會符合字串中任何順序的零或多個字元。樣式中的底線 (_) 會符合字串中的任何單一字元。其它任何字元會符合其本身或大小寫的同一字元，也就是說，符合會以不區分大小寫的方式執行 (注意：資料庫引擎只了解 7 位元拉丁字元的大小寫。因此，LIKE 運算子對 8 位元 iso8859 字元或 UTF-8 字元而言會區分大小寫。例如，運算式 'a' LIKE 'A' 為 TRUE，但 'æ' LIKE 'Æ' 則為 FALSE)。使用 SQLConnection.caseSensitiveLike 屬性可改變拉丁字元是否需要區分大小寫。

如果使用選擇性的 ESCAPE 子句，則 ESCAPE 關鍵字後面的運算式必須評估為包含單一字元的字串。這個字元可以用在 LIKE 樣式中，以符合百分比或底線字元常值。跳脫字元如果接在百分比符號、底線或其本身後面，就會分別符合字串中的百分比符號、底線或跳脫字元常值。

GLOB

GLOB 運算子類似 LIKE，但使用 Unix 檔案萬用字元語法做為萬用字元。與 LIKE 不同的是，GLOB 會區分大小寫。

IN

IN 運算子會計算其左側運算元是否等於其中一個右側運算元 (括在括弧中的一組數值)。

```
in-expr         ::=  expr [NOT] IN ( value-list ) |
                     expr [NOT] IN ( select-statement ) |
                     expr [NOT] IN [database-name.] table-name
value-list      ::=  literal-value [, literal-value]*
```

右側運算元可以是一組以逗號分隔的常值，或是 SELECT 陳述式的結果。如需有關使用 SELECT 陳述式做為 IN 運算子之右側運算元的說明和限制，請參閱運算式中的 SELECT 陳述式。

BETWEEN...AND

BETWEEN...AND 運算子相當於使用包含 >= 和 <= 運算子的兩個運算式。例如，運算式 x BETWEEN y AND z 相當於 x >= y AND x <= z。

NOT

NOT 運算子是否定運算子。NOT 關鍵字可以放在 GLOB、LIKE 和 IN 運算子前面，以反轉測試的意義 (換句話說，即檢查值是否不符合指示的樣式)。

參數

參數會指定運算式中常值的預留位置，之後再藉由為 SQLStatement.parameters 關聯陣列指定值，而於執行階段填入常值。參數有三種形式：

?	問號表示索引參數。執行階段會根據參數在陳述式中的位置來指定數值 (從零開始) 索引值。
:AAAA	冒號後面接著識別名稱代表名稱為 AAAA 之具名參數的位置。具名參數也會根據它們在 SQL 陳述式中的位置編號。為避免混淆，最好避免混合使用具名和編號參數。
@AAAA	'@' 符號等於冒號。

不支援的 SQL 功能

以下清單列出 Adobe AIR 中不支援的標準 SQL 元素：

FOREIGN KEY 條件約束 FOREIGN KEY 條件約束可以通過剖析但不會強制執行。

觸發程序 不支援 FOR EACH STATEMENT 觸發程序 (所有觸發程序都必須是 FOR EACH ROW)。不支援在資料表上使用 INSTEAD OF 觸發程序 (INSTEAD OF 觸發程序只能用在檢視上)。不支援遞迴觸發程序，即會觸發自身的觸發程序。

ALTER TABLE 只支援 ALTER TABLE 命令的 RENAME TABLE 和 ADD COLUMN 兩種變形。其它種類的 ALTER TABLE 作業 (例如 DROP COLUMN、ALTER COLUMN、ADD CONSTRAINT 等) 都會予以忽略。

巢狀交易 只允許一個作用中交易。

RIGHT 和 FULL OUTER JOIN 不支援 RIGHT OUTER JOIN 或 FULL OUTER JOIN。

可更新的檢視 檢視是唯讀的。您無法在檢視上執行 DELETE、INSERT 或 UPDATE 陳述式。但支援引發 INSTEAD OF 觸發程序以嘗試 DELETE、INSERT 或 UPDATE 檢視，而且可用來更新觸發程序主體中支援的資料表。

GRANT 和 REVOKE 資料庫是普通磁碟檔案，唯一可套用的存取權限就是基礎作業系統的標準檔案存取權限。不會實作主從式 RDBMS 常見的 GRANT 和 REVOKE 命令。

下列 SQL 元素和 SQLite 功能雖然在某些 SQLite 實作中受到支援，但在 Adobe AIR 中不受支援。當中的大部分功能都可以透過 SQLConnection 類別取得：

交易相關的 SQL 元素 (BEGIN、END、COMMIT、ROLLBACK) 這項功能可透過 SQLConnection 類別的交易相關方法取得：SQLConnection.begin()、SQLConnection.commit() 和 SQLConnection.rollback()。

ANALYZE 這項功能可透過 SQLConnection.analyze() 方法取得。

ATTACH 這項功能可透過 SQLConnection.attach() 方法取得。

COPY 不支援這個陳述式。

CREATE VIRTUAL TABLE 不支援這個陳述式。

DETACH 這項功能可透過 SQLConnection.detach() 方法取得。

PRAGMA 不支援這個陳述式。

VACUUM 這項功能可透過 SQLConnection.compact() 方法取得。

不支援系統資料表存取 SQL 陳述式中不提供系統資料表，包括 sqlite_master 和其它使用 "sqlite_" 前置詞的資料表。執行階段包含一組結構 API，可提供物件導向的方式來存取結構資料。如需詳細資訊，請參閱 SQLConnection.loadSchema() 方法。

規則運算式函數 (MATCH() 和 REGEX()) 這些函數無法用在 SQL 陳述式中。

下列功能在許多 SQLite 實作與 Adobe AIR 中並不相同：

索引陳述式參數 在許多實作中，索引陳述式參數會從 1 開始。但在 Adobe AIR 中，索引陳述式參數會從 0 開始 (也就是說，指定的第一個參數是索引 0，指定的第二個參數是索引 1，依此類推)。

INTEGER PRIMARY KEY 欄定義 在許多實作中，只會使用已確實定義為 INTEGER PRIMARY KEY 的欄做為資料表的實際主索引鍵欄。在這些實作中，使用通常為 INTEGER 之同義詞的其他資料類型 (例如 int)，並不會使欄做為內部主索引鍵使用。但是在 Adobe AIR 中，int 資料類型 (和其他 INTEGER 同義詞) 則被視為和 INTEGER 相等。因此，便會使用定義為 int PRIMARY KEY 的欄做為資料表的內部主索引鍵。如需詳細資訊，請參閱 「CREATE TABLE」和 「欄相似性」章節。

額外的 SQL 功能

根據預設，下列欄相似性類型在 SQLite 中不受支援，但在 Adobe AIR 中則受到支援 (請注意，就像 SQL 中的所有關鍵字一樣，這些資料類型名稱不區分大小寫)：

Boolean 對應到 Boolean 類別。

Date 對應到 Date 類別。

int 對應到 int 類別 (等於 INTEGER 欄相似性)。

Number 對應到 Number 類別 (等於 REAL 欄相似性)。

Object 對應到 Object 類別或可使用 AMF3 序列化和還原序列化的子類別 (這包括大部分包含自訂類別的類別，但不包括某些包含顯示物件和以屬性方式包含顯示物件之物件的類別)。

String 對應到 String 類別 (等於 TEXT 欄相似性)。

XML 對應到 ActionScript (E4X) XML 類別。

XMLList 對應到 ActionScript (E4X) XMLList 類別。

根據預設，下列常值在 SQLite 中不受支援，但在 Adobe AIR 中則受到支援：

true 用來代表 Boolean 常值 true，可搭配 BOOLEAN 欄使用。

False 用來代表 Boolean 常值 false，可搭配 BOOLEAN 欄使用。

資料類型支援

不同於大部分 SQL 資料庫，Adobe AIR SQL 資料庫引擎不要求或強制資料表欄必須包含特定類型的值，而是由執行階段使用兩種概念 (儲存類別和欄相似性) 來控制資料類型。本節說明儲存類別和欄相似性，以及在不同情況下如何解決資料類型差異：

- 第 970 頁 「儲存類別」
- 第 970 頁 「欄相似性」

- 第 972 頁「資料類型和比較運算子」
- 第 973 頁「資料類型和數學運算子」
- 第 973 頁「資料類型和排序」
- 第 973 頁「資料類型和群組」
- 第 973 頁「資料類型和複合 SELECT 陳述式」

儲存類別

儲存類別代表用來在資料庫儲存值的實際資料類型。資料庫使用的儲存類別如下：

NULL 值是 NULL 值。

INTEGER 值是具有正負號的整數。

REAL 值是浮點值。

TEXT 值是文字字串 (僅限於 256 MB)。

BLOB 值是二進位大型物件 (Binary Large Object，BLOB)，換句說話，就是原始的二進位資料 (僅限於 256 MB)。

凡是以內嵌在 SQL 陳述式中的常值方式提供給資料庫的值，或是使用參數方式繫結至準備好的 SQL 陳述式的值，在該 SQL 陳述式執行之前，執行階段都會為這些值指定一個儲存類別。

在 SQL 陳述式中的常值如果前後加上單引號或雙引號，執行階段會指定為儲存類別 TEXT，如果該常值是指定為未加引號的數字，而且既沒有小數點也沒有指數，則指定為 INTEGER，如果該常值是未加引號的數字，但是有小數點或指數，則指定為 REAL，如果值是 NULL 則指定為 NULL。儲存類別為 BLOB 的常值會使用 X'ABCD' 記號指定。如需詳細資訊，請參閱「運算式中的常值」。

使用 SQLStatement.parameters 關聯陣列提供為參數的值會被指定為最符合原始繫結資料類型的儲存類別。例如，int 值會繫結為 INTEGER 儲存類別，Number 值會指定為 REAL 儲存類別，String 值會指定為 TEXT 儲存類別，而 ByteArray 物件則會指定為 BLOB 儲存類別。

欄相似性

欄「相似性」是該欄中資料的建議儲存類型。在欄中儲存值時 (透過 INSERT 或 UPDATE 陳述式)，執行階段會嘗試將該值從其資料類型轉換為指定的相似性。例如，如果將 Date 值 (ActionScript 或 JavaScript Date 實體) 插入相似性為 TEXT 的欄，該 Date 值就會轉換為 String 形式 (等於呼叫物件的 toString() 方法)，然後再儲存到資料庫中。如果該值無法轉換為指定的相似性，就會發生錯誤，而且作業不會執行。使用 SELECT 陳述式從資料庫中擷取值時，會傳回對應到相似性之類別的實體，不論該值原先儲存在資料庫時是否由不同資料類型轉換而成。

如果欄接受 NULL 值，就可以使用 ActionScript 或 JavaScript 值 null 做為參數值，以在該欄中儲存 NULL。在 SELECT 陳述式中擷取 NULL 儲存類別值時，該值一定會以 ActionScript 或 JavaScript 值 null 的形式傳回，不論欄的相似性為何。如果欄接受 NULL 值，務必先檢查從該欄擷取的值，以判斷是否為 null，之後才能嘗試將該值轉型成不可為 null 的類型 (例如 Number 或 Boolean)。

執行階段會為資料庫中的每個欄指定下列其中一個類型相似性：

- TEXT (或 String)
- NUMERIC
- INTEGER (或 int)
- REAL (或 Number)
- Boolean
- Date

- XML
- XMLLIST
- Object
- NONE

TEXT (或 String)

具有 TEXT 或 String 相似性的欄會使用儲存類別 NULL、TEXT 或 BLOB 儲存所有資料。如果在 TEXT 相似性的欄中插入數值資料，該資料會在儲存之前轉換為文字。

NUMERIC

具有 NUMERIC 相似性的欄會使用儲存類別 NULL、REAL 或 INTEGER 儲存所有資料。在 NUMERIC 欄中插入文字資料時，執行階段會在儲存之前嘗試將它轉換為整數或實數。如果轉換成功，就會使用 INTEGER 或 REAL 儲存類別儲存該值 (例如，'10.05' 的值會轉換為 REAL 儲存類別，再儲存起來)。如果無法執行轉換，則會發生錯誤。執行階段不會嘗試轉換為 NULL 值。從 NUMERIC 欄中擷取的值會以最符合該值的特定數值類型實體傳回。換句話說，如果值是正整數或 0，就會以 uint 實體傳回。如果值是負整數，則以 int 實體傳回。最後，如果值有浮點組件 (不是整數)，則以 Number 實體傳回。

INTEGER (或 int)

使用 INTEGER 相似性的欄與具有 NUMERIC 相似性的欄行為類似，但有一個例外。如果要儲存的值是沒有浮點組件的實數 (例如 Number 實體)，或該值是可轉換為沒有浮點組件之實數值的文字值，則該值會轉換為整數，並使用 INTEGER 儲存類別儲存。如果嘗試儲存有浮點組件的實數，則會發生錯誤。

REAL (或 Number)

具有 REAL 或 NUMBER 相似性的欄與具有 NUMERIC 相似性的欄行為類似，唯一的例外是它會強制以浮點表示整數值。REAL 欄中的值在從資料庫傳回時，一定會使用 Number 實體。

Boolean

具有 Boolean 相似性的欄會儲存 true 或 false 值。Boolean 欄可接受 ActionScript 或 JavaScript Boolean 實體的值。如果程式碼嘗試儲存 String 值，執行階段會將長度大於零的 String 視為 true，將空白 String 視為 false。如果程式碼嘗試儲存數值資料，任何非零值都儲存為 true，而 0 則儲存為 false。使用 SELECT 陳述式擷取 Boolean 值時，該值會以 Boolean 實體傳回。非 NULL 值會使用 INTEGER 儲存類別儲存 (0 代表 false，1 代表 true)，而且在擷取資料時會轉換為 Boolean 物件。

Date

具有 Date 相似性的欄會儲存日期和時間值。Date 欄的用途是接受 ActionScript 或 JavaScript Date 實體的值。如果嘗試在 Date 欄中儲存 String 值，執行階段會嘗試將它轉換為凱撒曆日期。如果轉換失敗，就會發生錯誤。如果程式碼嘗試儲存 Number、int 或 uint 值，執行階段不會驗證資料，並假設它是有效的凱撒曆日期值。使用 SELECT 陳述式擷取的 Date 值會自動轉換為 Date 實體。Date 值會儲存為使用 REAL 儲存類別的凱撒曆日期，所以排序和比較作業會依照您預期的方式進行。

XML 或 XMLList

使用 XML 或 XMLList 相似性的欄會儲存 XML 結構。當程式碼嘗試使用 SQLStatement 參數在 XML 欄中儲存資料時，執行階段會使用 ActionScript XML() 或 XMLList() 函數嘗試轉換並驗證值。如果值無法轉換為有效的 XML，就會發生錯誤。如果嘗試使用常值 SQL 文字值儲存資料 (例如 INSERT INTO (col1) VALUES ('Invalid XML (no closing tag)'))，驗證不會剖析或驗證該值，並假設它的格式正確。如果儲存無效的值，擷取時會以空白的 XML 物件傳回。XML 和 XMLList 資料會使用 TEXT 儲存類別或 NULL 儲存類別儲存。

Object

具有 Object 相似性的欄會儲存 ActionScript 或 JavaScript 複雜物件，包括 Object 類別實體，以及 Object 子類別的實體 (如 Array 實體和自訂的類別實體)。Object 欄資料會以 AMF3 格式序列化，並使用 BLOB 儲存類別儲存。擷取值時，該值會從 AMF3 還原序列化，並以儲存之類別的實體傳回。請注意，有些 ActionScript 類別 (尤其是顯示物件) 無法還原序列化為原始資料類型的實體，所以在儲存自訂類別實體之前，您必須使用 flash.net.registerClassAlias() 方法為類別註冊別名 (在

Flex 中則是在類別宣告中加入 [RemoteObject] 中繼資料)。此外，在擷取該資料之前，必須為該類別註冊相同的別名。任何無法正確還原序列化的資料 (不論是因為類別本質上無法還原序列化，或因為類別別名遺漏或不相符) 都會以匿名物件 (Object 類別實體) 傳回，其屬性和值則對應到原先儲存的實體。

NONE

具有 NONE 相似性的欄不偏好任一種儲存類別。它在插入資料之前不會嘗試轉換資料。

決定相似性

欄的類型相似性會由 CREATE TABLE 陳述式中該欄的宣告類型決定。決定類型時所套用的規則 (不區分大小寫) 如下：

- 如果欄的資料類型包含 "CHAR"、"CLOB"、"STRI" 或 "TEXT" 中任一個字串，則該欄具有 TEXT/String 相似性。請注意，類型 VARCHAR 包含字串 "CHAR"，所以會指定為 TEXT 相似性。
- 如果欄的資料類型包含 "BLOB" 字串，或如果未指定資料類型，則該欄具有 NONE 相似性。
- 如果欄的資料類型包含 "XMLL" 字串，則該欄具有 XMLList 相似性。
- 如果資料類型是 "XML" 字串，則該欄具有 XML 相似性。
- 如果資料類型是 "OBJE" 字串，則該欄具有 Object 相似性。
- 如果資料類型是 "BOOL" 字串，則該欄具有 Boolean 相似性。
- 如果資料類型是 "DATE" 字串，則該欄具有 Date 相似性。
- 如果資料類型包含 "INT" (包括 "UINT") 字串，則該欄會指定為 INTEGER/int 相似性。
- 如果欄的資料類型包含 "REAL"、"NUMB"、"FLOA" 或 "DOUB" 中任一個字串，則該欄具有 REAL/Number 相似性。
- 否則，相似性會是 NUMERIC。
- 如果資料表是使用 CREATE TABLE t AS SELECT... 陳述式所建立的，則所有欄都未指定資料類型，並指定為 NONE 相似性。

資料類型和比較運算子

執行階段支援二元比較運算子 =、<、<=、>= 和 !=，以及測試集合成員 IN 的作業，和三元比較運算子 BETWEEN。如需有關這些運算子的詳細資訊，請參閱 「運算子」。

比較的結果須視所比較兩個值的儲存類別而定。比較兩個值時適用的規則如下：

- 執行階段會將儲存類別為 NULL 的值視為小於其它任何值 (包括其它儲存類別為 NULL 的值)。
- INTEGER 或 REAL 值小於任何 TEXT 或 BLOB 值。將 INTEGER 或 REAL 與其它 INTEGER 或 REAL 比較時，會執行數字比較。
- TEXT 值小於 BLOB 值。比較兩個 TEXT 值時，會執行二進位比較。
- 比較兩個 BLOB 值時，結果總是使用二進位比較來決定。

三元運算子 BETWEEN 一定會重新轉型為相等的二元運算式。例如，a BETWEEN b AND c 會重新轉型為 a >= b AND a <= c，即使這表示在每一個評估運算式所需的比較中會對 a 套用不同相似性。

類型 a IN (SELECT b ....) 的運算式由之前針對二元運算式列舉的三個規則處理，也就是類似 a = b。例如，如果 b 是欄值而 a 是運算式，則進行任何比較之前會先將 b 的相似性套用到 a。運算式 a IN (x, y, z) 會重新轉型為 a = +x OR a = +y OR a = +z。執行階段會將 IN 運算子右側的值 (在這個範例中是 x、y 和 z 值) 視為運算式，即使它們恰好是欄值。如果 IN 運算子左側的值是欄，就會使用該欄的相似性。如果該值是運算式，則不進行任何轉換。

比較執行方式也會受是否使用 COLLATE 子句影響。如需詳細資訊，請參閱 「COLLATE」。

資料類型和數學運算子

對於每個支援的數學運算子 *、/、%、+ 和 -，評估運算式之前，會對每個運算元套用數值相似性。如果任何運算元無法成功轉換為 NUMERIC 儲存類別，運算式會評估為 NULL。

使用連接運算子 || 時，每個運算元都會在評估運算式之前轉換為 TEXT 儲存類別。如果任何運算元無法轉換為 TEXT 儲存類別，運算式的結果將是 NULL。有兩種情況可能發生這種無法轉換值的狀況：一種是運算元的值是 NULL，另一種是 BLOB 包含非 TEXT 的儲存類別。

資料類型和排序

以 ORDER BY 子句將值排序時，儲存類別為 NULL 的值會排在前面。後面接著 INTEGER 和 REAL 值依數值順序排列，後面接著 TEXT 值依二進位順序或指定的定序順序 (BINARY 或 NOCASE) 排列。最後，則是依二進位順序排列的 BLOB 值。排序之前不會進行儲存類別轉換。

資料類型和群組

使用 GROUP BY 子句將值分組時，不同儲存類別的值會被視為不同值。但 INTEGER 和 REAL 值例外，也就是這兩種值的數值如果相等，就會被視為相等。執行階段不會對 GROUP BY 子句的任何結果值套用相似性。

資料類型和複合 SELECT 陳述式

複合 SELECT 運算子 UNION、INTERSECT 和 EXCEPT 會執行數值之間的隱含比較。執行這些比較之前，每個值可能會先套用相似性。複合 SELECT 結果集之中單一欄可傳回的所有值會套用相同的相似性 (如果有的話)。所套用的相似性是第一個組件 SELECT 陳述式傳回的欄中，在該位置有欄值 (而不是其它種類的運算式) 之欄的相似性。如果某個複合 SELECT 欄的組件 SELECT 陳述式都不會傳回欄值，比較之前就不會對該欄的值套用相似性。

第 67 章 SQL 錯誤詳細訊息、ID 和引數

SQLError 類別代表使用 Adobe AIR 本機 SQL 資料庫時可能發生的各種錯誤。針對任何特定例外，SQLError 實體都有一個包含中文錯誤訊息的 details 屬性。除此之外，每個錯誤訊息各有一個唯一的關聯識別名稱，使用 SQLError 物件的 detailID 屬性即可取得。應用程式可以使用 detailID 屬性來識別特定的 details 錯誤訊息。也可以為使用者提供使用者地區語言的替代文字。detailArguments 陣列中的引數值可放在錯誤訊息字串中的適當位置。當應用程式將錯誤的 details 屬性錯誤訊息向特定地區的使用者直接顯示時，這項功能便十分好用。

下表包含 detailID 值與相關聯中文錯誤訊息文字的清單。訊息中的預留位置文字表示將於執行階段代換成 detailArguments 值的位置。這份清單可做為 SQL 資料庫作業中錯誤訊息當地語系化的來源。

SQLError detailID	中文錯誤詳細訊息和參數
1001	連線已關閉。
1102	必須開啟資料庫後才能執行此作業。
1003	在參數屬性中找到 %s [,| 和 %s] 參數名稱，但在指定的 SQL 中找不到。
1004	參數計數不符。在指定的 SQL 中找到 %d 個，但在參數屬性中卻設定了 %d 個值。需要 %s [,| 和 %s] 的值。
1005	無法開啟自動壓縮功能。
1006	無法設定 pageSize 值。
1007	找不到名稱為 '%s' 且類型為 '%s' 的結構物件 (在資料庫 '%s' 中)。
1008	找不到名稱為 '%s' 的結構物件 (在資料庫 '%s' 中)。
1009	找不到類型為 '%s' 的結構物件 (在資料庫 '%s' 中)。
1010	在資料庫 '%s' 中找不到結構物件。
2001	解析器堆疊溢位。
2002	函數 '%s' 中的引數太多。
2003	在 '%s' 附近：語法錯誤。
2004	已經有使用此名稱的資料表或索引：'%s'。
2005	不允許在 SQL 中用 PRAGMA。
2006	不是可寫入的目錄。
2007	未知或不支援的聯結類型：'%s %s %s'。
2008	目前不支援 RIGHT 和 FULL OUTER JOIN。
2009	NATURAL 聯結不可使用 ON 或 USING 子句。
2010	同一個聯結中不能同時使用 ON 和 USING 子句。
2011	無法使用欄 '%s' 進行聯結，可能不是兩個資料表都有該欄。
2012	運算式中的 SELECT 只能產生單一結果。
2013	沒有這個資料表：'[%s.]%s'。
2014	未指定資料表。
2015	結果集有太多欄 |'%s' 上有太多欄。
2016	%s ORDER|GROUP BY 條件超出範圍 - 應介於 1 到 %d 之間。
2017	ORDER BY 子句中條件太多。
2018	%s ORDER BY 條件超出範圍 - 應介於 1 到 %d 之間。
2019	%r ORDER BY 條件不符合結果集中的任何欄。
2020	ORDER BY 子句應該在 '%s' 之後，而不是之前。
2021	LIMIT 子句應該在 '%s' 之後，而不是之前。
2022	'%s' 左右的 SELECT 所產生的結果欄數不相同。
2023	HAVING 之前需要一個 GROUP BY 子句。
2024	GROUP BY 子句中不允許使用彙總函數。
2025	彙總中的 DISTINCT 後面必須接著運算式。
2026	複合 SELECT 中條件太多。
2027	ORDER|GROUP BY 子句中條件太多。
2028	暫時性觸發程序不能使用限定名稱。
2030	觸發程序 '%s' 已經存在。
2032	無法在檢視上建立 BEFORE|AFTER 觸發程序：'%s'。

2033	無法在資料表上建立 INSTEAD OF 觸發程序：'%s'。
2034	沒有這個觸發程序：'%s'
2035	不支援遞迴觸發程序 ('%s')。
2036	沒有這個欄：%s[.%s[.%s]]
2037	SQL 不允許使用 VACUUM。
2043	資料表 '%s'：索引函數傳回無效的計畫。
2044	聯結中最多只能使用 %d 個資料表。
2046	無法加入 PRIMARY KEY 欄。
2047	無法加入 UNIQUE 欄。
2048	無法加入預設值為 NULL 的 NOT NULL 欄。
2049	無法加入預設值為非常數的欄。
2050	無法將欄加入檢視。
2051	不允許在 SQL 中使用 ANALYZE。
2052	無效的名稱：'%s'。
2053	不允許在 SQL 中使用 ATTACH。
2054	%s '%s' 無法參考資料庫 '%s' 中的物件。
2055	禁止存取 '[%s.]%s.%s'。
2056	未獲授權。
2058	沒有這個檢視：'[%s.]%s'。
2060	暫存資料表名稱必須是非限定名稱。
2061	資料表 '%s' 已經存在。
2062	已經有一個索引的名稱為 '%s'。
2064	重製欄名：'%s'。
2065	資料表 '%s' 有多個主要索引鍵。
2066	不允許在 INTEGER PRIMARY KEY 上使用 AUTOINCREMENT。
2067	沒有這個定序順序：'%s'。
2068	不允許在檢視中使用參數。
2069	檢視 '%s' 已循環定義。
2070	資料表 '%s' 不得卸除。
2071	使用 DROP VIEW 刪除檢視 '%s'。
2072	使用 DROP TABLE 刪除資料表 '%s'。
2073	'%s' 的外部索引鍵只能參考資料表 '%s' 中的一個欄。
2074	外部索引鍵中的欄數不符合被參考資料表中的欄數。
2075	外部索引鍵定義中有未知的欄 '%s'。
2076	資料表 '%s' 不得編製索引。
2077	檢視不得編製索引。
2080	指定的 ON CONFLICT 子句相衝突。
2081	沒有這個索引：'%s'
2082	不得卸除與 UNIQUE 或 PRIMARY KEY 條件約束相關聯的索引。
2083	不允許在 SQL 中使用 BEGIN。
2084	不允許在 SQL 中使用 COMMIT。
2085	不允許在 SQL 中使用 ROLLBACK。
2086	無法開啟暫存資料庫檔案以儲存暫存資料表。
2087	無法識別要重新編製索引的物件。
2088	資料表 '%s' 不得修改。
2089	無法修改 '%s'，因為它是檢視。
2090	變數數目必須介於 ?0 到 ?%d 之間。
2092	誤用別名彙總 '%s'。
2093	模稜兩可的欄名稱：'[%s.[%s.]]%s'。
2094	沒有這個函數：'%s'。
2095	函數 '%s' 的引數數目錯誤。
2096	禁止在 CHECK 條件約束中使用子查詢。
2097	禁止在 CHECK 條件約束中使用參數。
2098	運算式樹狀結構太大 (最大深度 %d)。
2099	RAISE() 只能用在觸發程式中。
2100	資料表 '%s' 有 %d 個欄，但提供的值有 %d 個。
2101	資料庫結構已鎖定：'%s'。

2102	陳述式太長。
2103	因為作用中陳述式所致，而無法刪除 / 修改定序順序。
2104	太多附加的資料庫 - 最多 %d 個。
2105	無法在交易內附加 (ATTACH) 資料庫。
2106	資料庫 '%s' 已在使用中。
2108	附加的資料庫必須與主要資料庫使用相同的文字編碼。
2200	記憶體不足。
2201	無法開啟資料庫。
2202	無法在交易內分離 (DETACH) 資料庫。
2203	無法分離資料庫：'%s'。
2204	資料庫 '%s' 已鎖定。
2205	無法取得資料庫的讀取鎖定。
2206	[欄 | 欄] '%s'[,'%s'] 不是 [唯一 | 唯一]。
2207	資料庫結構格式錯誤。
2208	不支援的檔案格式。
2209	無法辨識的字符：'%s'。
2300	無法將文字值轉換為數值。
2301	無法將字串值轉換為日期。
2302	無法在不遺失資料的情況下將浮點值轉換為整數。
2303	無法回復交易 - SQL 陳述式正在進行中。
2304	無法認可交易 - SQL 陳述式正在進行中。
2305	資料庫資料表已鎖定：'%s'。
2306	唯讀資料表。
2307	字串或 Blob 太大。
2309	無法開啟索引欄以供寫入。
2400	無法開啟類型 %s 的值。
2401	沒有這個 rowid：%s。
2402	物件名稱保留供內部使用：'%s'。
2403	不得修改檢視 '%s'。
2404	欄 '%s' 的預設值不是常數。
2405	未獲授權，無法使用函數 '%s'。
2406	誤用彙總函數 '%s'。
2407	誤用彙總：'%s'。
2408	沒有這個資料庫：'%s'。
2409	資料表 '%s' 沒有名稱為 '%s' 的欄。
2501	沒有這個模組：'%s'。
2508	沒有這個儲存點：'%s'。
2510	無法回復 - 沒有進行中的交易。
2511	無法認可 - 沒有進行中的交易。

第 68 章 Adobe Graphics Assembly Language (AGAL)

Adobe Graphics Assembly Language (AGAL) 是定義頂點和片段顯示程式的著色器語言。AGAL 程式必須上傳到採用本文件所述二進位位元組碼格式的顯示內容。

AGAL 位元組碼格式

AGAL 位元組碼必須使用 Endian.LITTLE_ENDIAN 格式。

位元組碼檔頭

AGAL 位元組碼必須以 7 位元組檔頭做為開頭：

```
A0 01000000 A1 00 -- for a vertex program
A0 01000000 A1 01 -- for a fragment program
```

偏移 (位元組)	大小 (位元組)	名稱	說明
0	1	magic	必須是 0xa0
1	4	版本	必須是 1
5	1	著色器類型 ID	必須是 0xa1
6	1	著色器類型	0 代表頂點程式，1 代表片段程式

字符

標頭後面接著任何數量的字符。每個字符的大小都是 192 位元 (24 位元組) 且格式永遠如下：

[opcode][destination][source1][source2 or sampler]

不是每個 opcode 都會用到所有欄位。未使用的欄位必須設定為 0。

作業程式碼

[opcode] 欄位的大小為 32 位元且可以接受下列其中一個值：

名稱	Opcode	作業	說明
mov	0x00	移動	將資料從 source1 移到 destination，以組件方式
add	0x01	加入	destination = source1 + source2，以組件方式
sub	0x02	減去	destination = source1 - source2，以組件方式
mul	0x03	相乘	destination = source1 * source2，以組件方式
div	0x04	相除	destination = source1 / source2，以組件方式
rcp	0x05	倒數	destination = 1/source1，以組件方式
min	0x06	最小值	destination = minimum(source1,source2)，以組件方式
max	0x07	最大值	destination = maximum(source1,source2)，以組件方式

名稱	Opcode	作業	說明
frc	0x08	分數	destination = source1 - (float)floor(source1)，以組件方式
sqt	0x09	平方根	destination = sqrt(source1)，以組件方式
rsq	0x0a	倒數根	destination = 1/sqrt(source1)，以組件方式
pow	0x0b	次方	destination = pow(source1,source2)，以組件方式
log	0x0c	對數	destination = log_2(source1)，以組件方式
exp	0x0d	指數	destination = 2^source1，以組件方式
nrm	0x0e	標準化	destination = normalize(source1)，以組件方式 (只會產生 3 個組件結果，目標必須遮蓋為 .xyz 以下)
sin	0x0f	正弦	destination = sin(source1)，以組件方式
cos	0x10	餘弦	destination = cos(source1)，以組件方式
crs	0x11	外積	destination.x = source1.y * source2.z - source1.z * source2.y destination.y = source1.z * source2.x - source1.x * source2.z destination.z = source1.x * source2.y - source1.y * source2.x (只會產生 3 個組件結果，目標必須遮蓋為 .xyz 以下)
dp3	0x12	內積	destination = source1.x*source2.x + source1.y*source2.y + source1.z*source2.z
dp4	0x13	內積	destination = source1.x*source2.x + source1.y*source2.y + source1.z*source2.z + source1.w*source2.w
abs	0x14	絕對	destination = abs(source1)，以組件方式
neg	0x15	變為負數	destination = -source1，以組件方式
sat	0x16	飽和	destination = maximum(minimum(source1,1),0)，以組件方式
m33	0x17	乘以矩陣 3x3	destination.x = (source1.x * source2[0].x) + (source1.y * source2[0].y) + (source1.z * source2[0].z) destination.y = (source1.x * source2[1].x) + (source1.y * source2[1].y) + (source1.z * source2[1].z) destination.z = (source1.x * source2[2].x) + (source1.y * source2[2].y) + (source1.z * source2[2].z) (只會產生 3 個組件結果，目標必須遮蓋為 .xyz 以下)
m44	0x18	乘以矩陣 4x4	destination.x = (source1.x * source2[0].x) + (source1.y * source2[0].y) + (source1.z * source2[0].z) + (source1.w * source2[0].w) destination.y = (source1.x * source2[1].x) + (source1.y * source2[1].y) + (source1.z * source2[1].z) + (source1.w * source2[1].w) destination.z = (source1.x * source2[2].x) + (source1.y * source2[2].y) + (source1.z * source2[2].z) + (source1.w * source2[2].w) destination.w = (source1.x * source2[3].x) + (source1.y * source2[3].y) + (source1.z * source2[3].z) + (source1.w * source2[3].w)

名稱	Opcode	作業	說明
m34	0x19	乘以矩陣 3x4	destination.x = (source1.x * source2[0].x) + (source1.y * source2[0].y) + (source1.z * source2[0].z) + (source1.w * source2[0].w) destination.y = (source1.x * source2[1].x) + (source1.y * source2[1].y) + (source1.z * source2[1].z) + (source1.w * source2[1].w) destination.z = (source1.x * source2[2].x) + (source1.y * source2[2].y) + (source1.z * source2[2].z) + (source1.w * source2[2].w) (只會產生 3 個組件結果，目標必須遮蓋為 .xyz 以下)
kil	0x27	刪除 / 捨棄 (僅限片段著色器)	如果單一純量來源組件小於零，則片段會被捨棄且不會繪製至影格緩衝區 (目的地暫存器必須設定為全 0)。
tex	0x28	紋理樣本 (僅限片段著色器)	destination 等於從紋理 source2 載入，座標為 source1。在這個情況中，source2 必須採用 sampler 格式。
sge	0x29	若大於等於則設定	destination = source1 >= source2 ? 1 : 0，以組件方式
slt	0x2a	若小於則設定	destination = source1 < source2 ? 1 : 0，以組件方式
seq	0x2c	若等於則設定	destination = source1 == source2 ? 1 : 0，以組件方式
sne	0x2d	若不等於則設定	destination = source1 != source2 ? 1 : 0，以組件方式

在 AGAL2 中，已經引入下列 opcode：

名稱	Opcode	作業	說明
ddx	0x1a	X 中的部分衍生	將 source1 中 X 的部分衍生載入 destination。
ddy	0x1b	Y 中的部分衍生	將 source1 中 Y 的部分衍生載入 destination。
ife	0x1c	若等於	如果 source1 等於 source2，則跳轉。
ine	0x1d	若不等於	如果 source1 不等於 source2，則跳轉。
ifg	0x1e	若大於	如果 source1 大於或等於 source2，則跳轉。
ifl	0x1f	若小於	如果 source1 小於 source2，則跳轉。
els	0x20	else	Else 區塊
eif	0x21	Endif	關閉 if 或 else 區塊。

目的地欄位格式

[destination] 欄位的大小是 32 位元：

```
31.............................0
----TTTT----MMMMNNNNNNNNNNNNNNNN
```

T = 暫存器類型 (4 位元)

M = 寫入遮色片 (4 位元)

N = 暫存器編號 (16 位元)

- = 未定義，必須是 0

來源欄位格式

[source] 欄位的大小是 64 位元：

```
63.............................................................0
D-------------QQ----IIII----TTTTSSSSSSSSOOOOOOOONNNNNNNNNNNNNNNN
```

D = Direct=0/Indirect=1 代表直接 Q 和 I 會被忽略，1 位元

Q = 索引暫存器組件選取 (2 位元)

I = 索引暫存器類型 (4 位元)

T = 暫存器類型 (4 位元)

S = 拌和 (8 位元，每個組件各 2 位元)

O = 間接偏移 (8 位元)

N = 暫存器編號 (16 位元)

- = 未定義，必須是 0

取樣程式欄位格式

tex opcode 的第二個來源欄位必須採用 [sampler] 格式，其大小為 64 位元：

```
63.............................................................0
FFFFMMMMWWWWSSSSDDDD--------TTTT--------BBBBBBBBNNNNNNNNNNNNNNNN
```

N = 取樣程式暫存器編號 (16 位元)

B = 紋理細緻程度 (LOD) 偏差值，正負號整數，縮放值 8。使用的浮點值為 b/8.0 (8 位元)

T = 暫存器，必須是 5，取樣程式 (4 位元)

F = 濾鏡 (0= 最接近，1= 線性) (4 位元)

M = MIP 映射 (0= 停用，1= 最接近，2= 線性)

W = 換行 (0= 固定，1= 重複)

S = 特殊旗標位元 (必須是 0)

D = 維度 (0=2D，1= 立方體)

程式暫存器

使用的暫存器數目取決於使用的 Context3D 描述檔。下列表格定義暫存器數目及其用法：

名稱 **AGAL2** **AGAL3** 用法	值	**AGAL**	
每個片段程式的數目 每個頂點程式的數目 每個片段程式的數目 每個頂點程式的數目		每個片段程式的數目	每個頂點程式的數目
Context 3D 描述檔支援 標準 超出標準		標準之下	
SWF 版本 25 28 以上		25 之下	
特質 NA 8 NA 16 頂點著色器輸入；從使用 Context3D.setVertexBufferAt() 指定的頂點緩衝區中讀取。	0	NA	8

名稱 **AGAL2** **AGAL3** 用法	值	**AGAL**	
常數 64 250 200 250 著色器輸入；使用函數的 Context3D.setProgramConstants() 系列來設定。	1	28	128
暫時 26 26 26 26 計算用的暫時暫存器；無法從程式外存取。	2	8	8
輸出 1 1 1 1 著色器輸出：在頂點程式中，輸出是片段空間位置；在片段程式中，輸出則是顏色。	3	1	1

名稱 **AGAL2** **AGAL3** 用法	值	**AGAL**	
可變 10 10 10 10 在頂點和片段著色器之間傳輸插補的資料。頂點程式中的可變暫存器會以輸入形式套用到片段程式。值是根據與三角形頂點的距離來插補。	4	8	8

名稱 **AGAL2** **AGAL3** 用法	值	**AGAL**	
取樣程式 16 NA 16 NA 片段著色器輸入：從使用 Context3D.setTextureAt() 指定的紋理讀取。	5	8	NA
片段暫存器 1 NA 1 NA 唯寫且用來重新撰寫已寫在頂點著色器內的 z 值 (或深度值)。	6	NA	NA
驗證權杖 1024 2048		200	

最新的 AGAL Mini Assembler 可以在這裡找到。