

教材編號：U2956c  
微軟 MCTS/MCPD 認證系列  
.NET 核心程式設計  
(含 Visual Basic 及 C# 程式碼)



恒逸資訊教育訓練中心  
台北市復興北路 99 號 16F / 14 F / 12F  
TEL:02-25149191 FAX:02-25149292  
新竹市光復路二段 295 號 3 樓  
TEL:03-5723322 FAX:03-5726566  
台中市中港路一段 201 號 2 樓  
TEL:04-23297722 FAX:04-23102000  
高雄市中山二路 2 號 25 樓  
TEL:07-5361199 FAX:07-5366161  
<http://edu.uuu.com.tw>

【版權所有，未經恒逸書面授權請勿翻印或轉載】

微軟 MCTS/MCPD 認證系列  
.NET 核心程式設計

---

**SYSTEX**  
making it happen 精誠資訊

**U'COM** 恒逸資訊  
教育訓練中心  
Information Technology Education Center

恒逸資訊教育訓練中心

作者／許薰尹/羅慧真/趙敏翔/許嘉仁/張書源/鄭淑芬/高光弘

改版作者／羅慧真

初版日期／2006/12/31、改版日期／2009/1/31

地址／台北市復興北路 99 號 14F  
電話／(02)25149191



# .NET 核心程式設計

微軟 MCTS/MCPD 認證系列

教材編號：U2956c

# 教材大綱

第一章：使用集合管理物件	
Object-Base 的集合.....	4
處理集合所需之相關的介面 .....	5
使用 ArrayList 集合 .....	7
使用 Hashtable .....	9
使用 SortedList 集合 .....	10
使用 HybridDictionary 集合 .....	12
使用集合物件的考量 .....	14
練習 1.1：選擇適合的集合類別 .....	16
泛型集合 (Generic Collection) .....	18
處理泛型集合所需之相關的介面 .....	20
使用堆疊 (Stack).....	23
使用佇列 (Queue) .....	24
練習 1.2：選用適當的泛型集合 .....	26
Dictionary 相關的泛型集合 .....	29
練習 1.3：選用適當的泛型集合 .....	33
StringCollection 類別 .....	37
StringDictionary 類別.....	38
NameValuePairCollection 類別.....	40
泛型介面 (Generic Interface).....	42
練習 1.4：建立具有比較功能的自訂類別 .....	44
總結 .....	47

## 第二章：管理與存取檔案系統

檔案系統 .....	4
存取磁碟資訊 .....	5
資料夾的管理 .....	7
檔案管理 .....	9
檔案讀寫 .....	11
Path 類別 .....	13
練習 2.1：使用 File 讀寫檔案 .....	15
監控檔案目錄 .....	17
練習 2.2：檔案系統及檔案監視 .....	18
資料流的讀寫 .....	22
認識資料流 .....	23
讀寫檔案串流 .....	25
讀寫文字到資料流 .....	27
練習 2.3：資料流的讀寫 .....	30
讀寫二進位資料 .....	34
暫存於記憶體 .....	37
為資料流加上緩衝器 .....	38
練習 2.4：使用 BufferedStream .....	39
壓縮資料概念 .....	43
練習 2.5：使用 DeflateStream 壓縮與解壓縮 .....	47
隔離儲存區的運作 .....	52
什麼是隔離儲存區 .....	53

IsolatedStorageFile 類別 .....	56
IsolatedStorageFileStream 類別 .....	58
練習 2.6：讀寫隔離儲存區的檔案 .....	59
總結 .....	63

### 第三章：文字處理

字串處理 .....	4
StringBuilder 類別 .....	6
練習 3.1：使用 StringBuilder .....	7
規則運算式 .....	9
認識規則運算式 .....	10
規則運算式的格式 .....	11
.NET Framework 的規則運算式 .....	13
文字格式比對 .....	15
擷取特定規則的文字 .....	17
練習 3.2：使用 Regex 擷取標題 .....	19
認識文字編碼 .....	22
使用 Encoding 類別 .....	23
練習 3.3：使用 UTF8 讀取文件 .....	25
總結 .....	28

### 第四章：全球化程式設計

文化特性的運作 .....	4
認識全球化應用程式 .....	5
存取文化資訊 .....	6
取得使用者的文化特性 .....	8
指定特定語系 .....	9
擷取所有的 CultureInfo .....	11
查看地區的細節資訊 .....	12
練習 4.1：列出所有的文化資訊 .....	13
根據文化特性進行… .....	17
數字的格式化 .....	18
日期時間的格式化 .....	20
文字比較 .....	22
練習 4.2：列出所有的文化資訊 .....	24
自訂文化特性 .....	27
設計步驟 .....	28
練習 4.3：建立自訂文化特性 .....	30
總結 .....	34

### 第五章：序列化處理

序列化的處理概念 .....	4
序列化的格式 .....	6
Binary 序列化 .....	8
SOAP 序列化 .....	12
練習 5.1：使用 Binary 序列化方法保存物件 .....	15
XML 序列化 .....	19
練習 5.2：將 XML 字串序列化/反序列化 .....	22

---

使用 Attribute 控制 XML 序列化 .....	26
練習 5.3：處理集合及物件繼承的序列化問題.....	32
自訂序列化 .....	37
使用 ISerializable 介面 .....	38
使用 OnXXX Attribute 處理自訂序列化 .....	41
練習 5.4：使用 OnXXXAttribute .....	42

## 第六章：程式繪圖

Point 及 Size 結構.....	4
Graphics 物件.....	6
筆, 筆刷及字型物件 .....	7
直線 .....	9
矩形 .....	11
筆刷的變化 .....	12
練習 6.1：繪製一個傳統型體溫計 .....	15
圓形 .....	18
Pie 形 .....	20
文字處理 .....	22
練習 6.2：以 Pie 形呈現市佔率 .....	27
影像處理 .....	30
練習 6.3：網頁動態影像處理 .....	34

## 第七章：安裝組件與設定組態檔

組件配置及版本控制 .....	4
認識組件 .....	5
私有與共享組件 .....	7
共享組件 .....	8
認識全域組件快取 .....	9
組件版本控制 .....	10
開發階段共享組件的方式 .....	12
.NET Framework 執行階段的版本控制 .....	14
練習 7.1：開發階段共享組件的方式 .....	16
編修組態檔 .....	19
認識組態檔 .....	20
一般常用的設定資訊 .....	21
專案的 Settings 組態 .....	23
存取 Application 範圍的組態項目 .....	25
練習 7.2：存取 Settings 組態檔 .....	27
使用程式管理組態檔 .....	31
如何開啟組態檔 .....	33
Application Scope 的組態項目結構 .....	38
保護設定檔資料 .....	40
練習 7.3：修改應用程式組態檔 .....	43
認識 Window Installer .....	48
建立 Setup 專案 .....	49
自訂安裝程式 .....	50
認識 Installer 類別 .....	51
Setup 專案 .....	55

## 第八章：監控及偵錯

認識 Process 類別.....	5
取得 Process 的資訊.....	6
ProcessStartInfo 類別.....	8
作業系統的事件檢視.....	11
認識作業系統的事件記錄.....	12
建立事件記錄.....	15
寫入事件訊息到記錄檔.....	16
讀取事件訊息.....	18
刪除事件記錄檔.....	19
練習 8.1：列出事件記錄.....	20
認識偵錯與追蹤.....	24
練習 8.2：使用 Debug 類別偵錯.....	26
追蹤應用程式.....	32
管理應用程式效能.....	37
效能監視.....	38
建立自訂效能計數器.....	40
效能計數器.....	42
練習 8.3：建立自訂效能計數器.....	45
Windows 管理規範 - WMI.....	48
如何查詢目前暫停中的服務.....	51
如何監控新啓動的應用程式(同步).....	53
如何監控被暫停的服務(非同步).....	55
練習 8.4：監控新建立帳號(非同步).....	57

## 第九章：建立多執行緒應用程式

執行緒基本觀念.....	4
建立執行緒.....	5
Thread 物件常用成員.....	8
執行緒生命週期.....	9
中止執行緒執行.....	10
練習 9.1：建立多執行緒應用程式.....	11
執行緒集區(ThreadPool).....	14
執行緒執行環境(Execution Context).....	16
練習 9.2：使用 ThreadPool 物件.....	17
Timer 物件類別.....	19
多執行緒中的共用資源.....	20
Synchronization Lock.....	21
Monitor 類別.....	22
Mutex 類別.....	23
Semaphore 類別.....	24
ReaderWriterLock 類別.....	25
非同步程式開發模型 (Asynchronous Programming Model).....	26
判斷背景執行緒是否完成.....	27
練習 9.3：使用 APM 建立多執行緒應用程式.....	28

## 第十章：Windows 服務與 AppDomain

認識 Windows 服務(Service) .....	4
服務管理員 .....	6
建立 Windows Service .....	8
認識狀態與方法之間的關聯 .....	10
練習 10.1：建立 Windows 服務 .....	11
認識 Installer 類別 .....	15
服務的安全性內容 .....	16
安裝 Windows 服務 .....	18
使用安裝專案 .....	19
練習 10.2：建立服務的安裝類別 .....	22
管理及控制服務 .....	26
使用 ServiceController 類別 .....	27
練習 10.3：管理及控制服務 .....	29
認識應用程式定義域(Application Domain) .....	35
認識 AppDomain 類別 .....	37
如何建立應用程式定義域 .....	38
如何取得目前的應用程式定義域 .....	40
練習 10.4：載入組件到應用程式定義域 .....	41
總結 .....	43

## 第十一章：程式碼安全性

認識程式碼存取安全 (Code Access Security) .....	4
什麼是 Evidence? .....	5
什麼是安全原則 (Security Policy)? .....	7
什麼是程式群組 (Code Group)? .....	9
什麼是安全原則層級 (Security Policy Level)? .....	11
符合多個原則層級 (Policy Levels) 的處理 .....	12
使用.NET Framework 組態工具 .....	13
保護組件使用宣告式安全 (Declarative) .....	16
認識宣告式與命令式安全性 .....	17
使用 CAS 組件宣告的原因 .....	19
宣告式 Attribute 的屬性 .....	20
如何建立組件宣告 .....	21
練習 11.1：組件宣告式安全 .....	23
練習 11.2：設定安全原則 .....	27
方法的權限安全種類 .....	30
Demand 與 LinkDemand 的相異性 .....	32
練習 11.3：宣告式與命令式安全性 .....	34
總結 .....	37

## 第十二章：驗證與授權

驗證及授權 .....	4
認識驗證及授權 .....	5
使用 Role-Based Security .....	7
Windows 驗證使用 RBS .....	9
練習 12.1：Windows 驗證使用 RBS .....	12
自行驗證使用 RBS .....	15
執行安全檢查 .....	20

---

練習 12.2：自行驗證使用 RBS.....	22
使用 ACL 保護檔案系統 .....	27
認識是 DACL .....	28
認識是 SACL.....	30
使用.NET Framework 的存取控制類別 .....	33
存取 SD 資訊.....	35
變更存取控制項目 .....	37
練習 12.3：設定檔案保留繼承 .....	39

### 第十三章：資料加密及雜湊處理

加/解密及雜湊演算法的運作概念 .....	4
了解資料加密 .....	5
對稱式金鑰 .....	6
非對稱式金鑰 .....	7
了解雜湊演算法 .....	9
資料加/解密的程式 .....	11
.NET Framework 支援的加/解密技術 .....	12
使用對稱式金鑰加密 .....	13
使用對稱式金鑰解密 .....	16
練習 13.1：使用對稱式金鑰加/解密 .....	18
非對稱式金鑰加/解密 .....	25
練習 13.2：使用非對稱式金鑰加/解密 .....	27
資料的雜湊處理 .....	30
練習 13.3：使用雜湊式演算法 .....	34

### 第十四章：與非.NET 程式的互通

認識 Win32 API.....	4
Platform Invocation Service .....	6
DllImportAttribute.....	9
處理複雜的型別 .....	10
練習 14.1：呼叫 Win32 API .....	13
認識 Runtime Callable Wrapper .....	19
Wrapper class .....	20
建立 Interop Assembly .....	21
如何存取 COM 物件 .....	23
練習 14.2：在.NET 中呼叫 COM 元件 .....	26
認識 COM Callable Wrapper .....	30
設計可讓 COM 呼叫的.NET 元件 .....	32
建立.NET 元件給 COM 用戶端使用 .....	34
部署給 COM 使用的.NET 元件 .....	37
練習 14.3：建立.NET 組件供 COM 用戶端使用 .....	39

## 第十五章：電子郵件程式設計

.NET Framework 的 Mail 機制.....	4
建立及傳送 E-mail 訊息的程序.....	5
建立 Mail 訊息.....	6
使用 SmtpClient 傳送 Mail.....	8
練習 15.1：使用 SmtpClient 傳送 Mail.....	10
在組態檔設定 SmtpClient .....	14
指定多個收件者 .....	16
在電子郵件中附加檔案 .....	18
練習 15.2：在傳送的 Mail 中附加檔案.....	20
建立 HTML E-Mail.....	23
利用非同步傳送 Mail.....	25
非同步傳送 Mail 設計步驟.....	27
練習 15.3：使用非同步方式傳送 Mail.....	29

## 關於本課程...

本課程針對.NET Framework 應用程式的進階開發技術，讓您瞭解如何透過.NET 平台所提供的功能撰寫程式，幫助你建立企業級的應用程式。

對於企業級應用程式，常需要像是檔案的壓縮/解壓縮技術來減少資料傳遞的大小或儲存空間的大小。偵測使用者是否正確的使用可以進行「稽核與錯誤追蹤」。某些系統狀況需要軟體可以自動使用信件（E-mail）通知管理人員。軟體是否可以應付企業的「效能」需求，這要如何進行偵測、如何為你的軟體加上多層的防護，必免軟體被冒用，被竊取…等安全技術議題。這些技術你將從這個課程學習到。同時這個課程也是 MCTS、MCPD 先修課程，以協助取得相關認證。

### 對象

- 需了解基本的程式設計概念以及.NET 平台開發者
- 使用過 Visual Basic 或 C# 程式開發程式者
- 對.NET 進階相關技術有興趣的專業程式開發人員
- 欲取得 MCPD、MCTS 認證的程式開發人員

### 先修課程

已完成以下課程所具備技術能力

- U9995 Visual Basic 2008 程式語言概論
- 或
- U9996 Visual C# 2008 程式語言概論

### 背景知識

本課程學員應具備以下基本技術：

- 需熟悉 Visual Basic 或 C# 程式語言

### 參考課程

完成本課程之後，可參考以下相關課程：

- U2546VB : .NET Windows 程式設計(Visual Basic 2005)
- U2546C# : .NET Windows 程式設計(Visual C# 2005)
- U70529VB : .NET 分散式應用程式開發(Visual Basic 2005)
- U70529C# : .NET 分散式應用程式開發(Visual C# 2005)
- 6460 : Windows Presentation Foundation 設計與開發(Visual Basic 2008 & Visual C# 2008)
- 6461 : Windows Communication Foundation 設計與開發(Visual Basic 2008 & Visual C# 2008)
- 6462 : Windows Workflow Foundation 設計與開發(Visual Basic 2008 & Visual C# 2008)

## 課程目標

上完本課程您將具備：

- 學習如何使用集合及泛型集合管理資料
- 如何使用程式管理檔案系統
- 如何讀寫檔案串流及使用程式進行串流壓縮及解壓縮的技術
- 利用文字規則的特性使用 Regular Expression 進行文字搜尋技巧
- 了解如何設計多國語言--符合當地文化資訊（像是日期格式、數字格式）的應用程式
- 如何使用.NET Framework 的繪圖類別設計一個影像處理的程式
- 如何使用序列化與反序列化進行物件資訊的保存
- 讀寫事件檢視器以稽核應用程式
- 管理應用程式的效能使用效能計數器
- 應用程式的偵錯與追蹤
- 了解如何設定應用程式的組態檔並使用程式讀寫組態檔
- 了解如何使用 Installer 類別進行應用程式的自訂安裝
- 了解如何進行應用程式組件的版本控制
- 如何開發一個可以多工（多執行緒）的應用程式
- 如何實作系統常駐程式的 Windows 服務以及如何安裝 Windows 服務
- 如何利用加解密技術進行機密資料的保密
- 如何以程式存取安全（Code Access Security）控管應用程式的存取權限
- 如何從使用者角色進行安全控管及利用 ACL 進行檔案存取權限
- 了解.NET 的程式如何與 Visual Basic 6 所開發的元件（COM 物件）合作
- 如何在程式中進行信件（Mail）訊息的傳送

# 課程軟體安裝須知

## 適用版本

本教材適用於以下.NET Framework 版本：

1. .NET Framework 2.0
2. .NET Framework 3.0
3. .NET Framework 3.5

## 作業系統

1. Windows Server 2003 英文版。

## 軟體

1. Visual Studio 2008 Professional 英文版
2. Visual Studio 2008 Professional Service Pack 1 英文版
3. SQL Server 2005 英文版
4. SQL Server 2005 Express 英文版
5. Office 2003 中文版 或 Office 2007

# 第一章：使用集合管理 物件

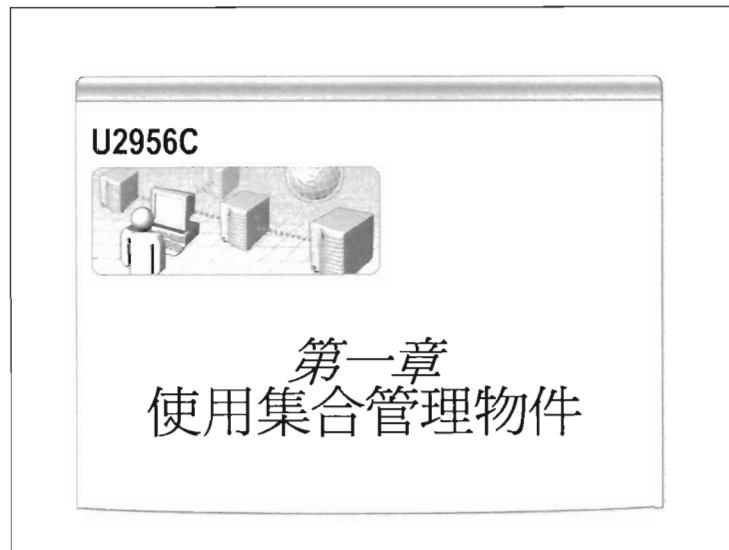
## 本章大綱

Object-Base 的集合.....	4
處理集合所需之相關的介面.....	5
使用 ArrayList 集合.....	7
使用 Hashtable .....	9
使用 SortedList 集合.....	10
使用 HybridDictionary 集合 .....	12
使用集合物件的考量.....	14
練習 1.1 : 選擇適合的集合類別.....	16
泛型集合 (Generic Collection).....	18
處理泛型集合所需之相關的介面.....	20
使用堆疊 (Stack).....	23
使用佇列 (Queue).....	24
練習 1.2 : 選用適當的泛型集合.....	26
Dictionary 相關的泛型集合.....	29
練習 1.3 : 選用適當的泛型集合.....	33
StringCollection 類別.....	37
StringDictionary 類別 .....	38
NameValueCollection 類別 .....	40
泛型介面 (Generic Interface).....	42
練習 1.4 : 建立具有比較功能的自訂類別.....	44
總結.....	47

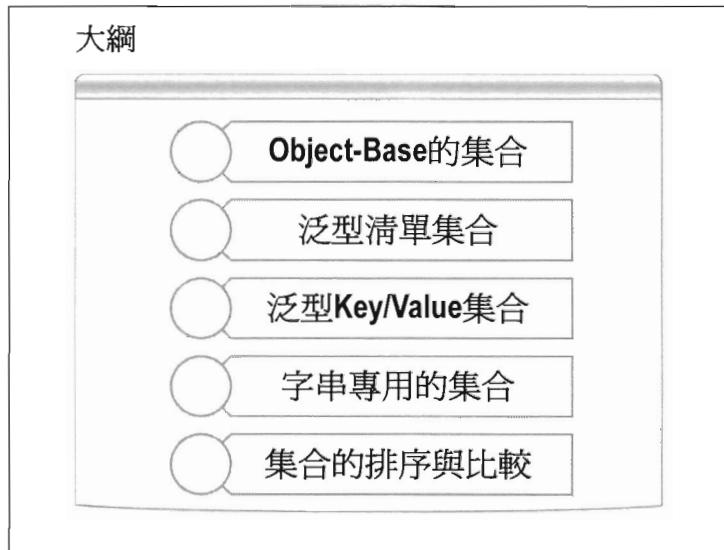
作者：

羅慧真





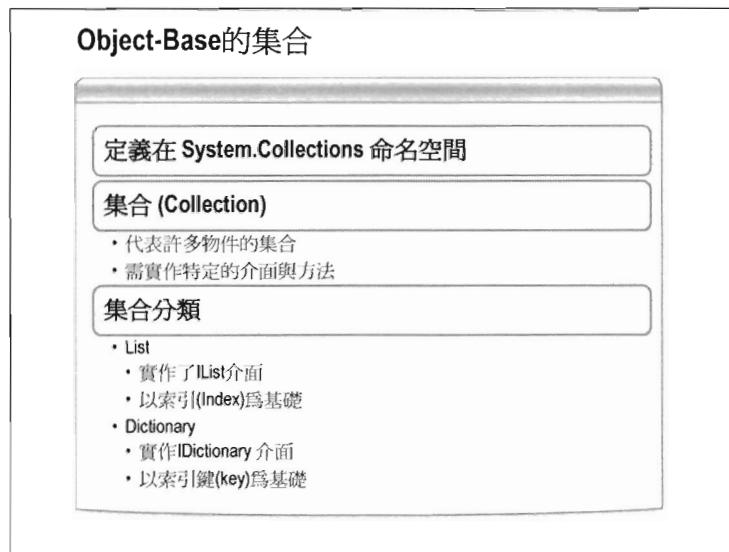
---



處理大量類似性質的資料可以使用陣列或者是集合，要具有可新增、移除項目的彈性，那麼就屬集合具有此彈性。.NET Framework 提供了許多的集合類別，在這個章節中將介紹.NET Framework 所提供的集合與泛型集合建立能動態新增項目的資料儲存結構。您應該根據集合的特色，適當地選擇集合類型來存放資料。

本章介紹以下主題：

- Object-Base 的集合
- 泛型清單集合
- 泛型 Key/Value 集合
- 字串專用的集合
- 集合的排序與比較



## Object-Base 的集合

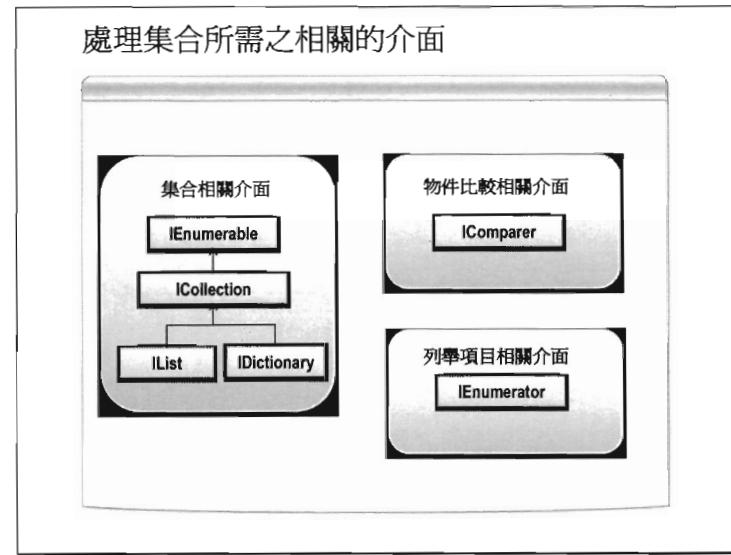
Object-Base 的集合就是一般傳統的集合類型，它們都被定義在 System.Collections 命名空間中，儲存的元素為 Object 型別，在處理時必須進行型別轉換。

.NET 在 System.Collections 命名空間之下定義許多不同物件的集合，包含 List、佇列 (Queue)、ArrayList、Hashtable、Dictionary 等，這些物件都各有其用途。不過所有在此命名空間下的類別都實作了 ICollection 介面，而 ICollection 介面則實作了 IEnumerable 介面。

在這一節中將介紹如何使用集合 (Collection)，並介紹常用的集合物件，以及這些特殊介面的用途。

### 了解集合

集合通常分為兩大類：List 與 Dictionary。List 實作了 IList 介面，以索引 (Index) 為基礎；Dictionary 實作了 IDictionary 介面，以索引鍵 (key) 為基礎，代表資料儲存時，都是一個 key 值配一個 value 方式存放。



## 處理集合所需之相關的介面

一般處理集合時需要對集合內容進行新增、插入、查詢元素所屬之索引位置，進行排序以及項目的列舉。於是根據其功能定義了幾個介面：

- **IEnumerable**，定義了列舉集合項目的功能，有實作這個介面的集合才可以使用 For Each(或 foreach)語法列與集合項目。
- **ICollection**，擴充了 **IEnumerable**，多了像是 **Count** 屬性、**CopyTo** 方法等功能。
- **IList**，定義了以整數索引存取物件的集合類型，除了擴充 **IEnumerable**、**ICollection** 之外，還新增了 **Add**、**Insert**、**Remove...** 等功能定義。
- **IDictionary**，定義了以 Key/Value 存取物件的集合類型，除了擴充 **IEnumerable**、**ICollection** 之外，還新增了 **Add**、**Insert**、**Remove...** 等功能定義。
- **IComparer**，定義兩個物件比較的功能。
- **IEnumerator**，定義簡單反覆運算列舉值的功能。

## 了解 IList 介面

IList 介面主要用來管理許多物件之間的順序，也就是說 IList 介面代表著一些按次序排序的物件所成的集合，可經由索引值存取其內含的元素(也稱項目)。.NET 類別函式庫的 System.Collections.IList 介面定義一個標準的 List 物件提供的方法和屬性。常用的方法與屬性：

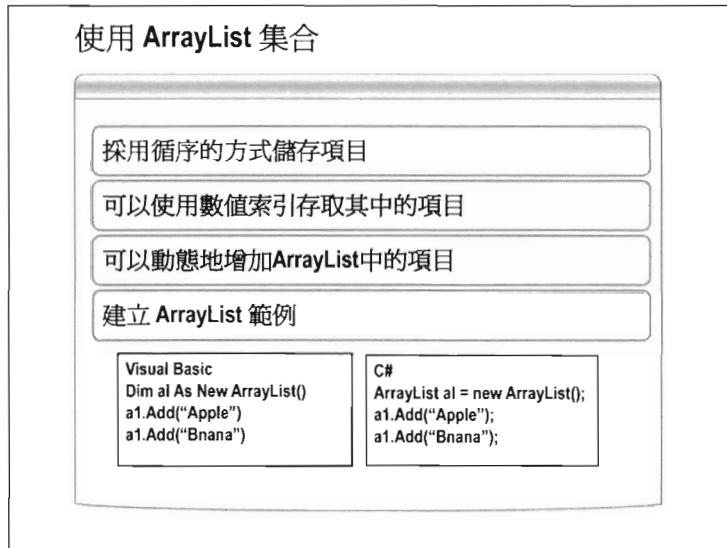
方法與屬性	說明
Add 方法	新增一個項目到 IList，並回傳新項目所在的索引值
Clear 方法	清除 IList 中所有的項目
IndexOf 方法	判斷 IList 中特定項目的索引值，若有包含此項目則回傳此項目的索引值；否則回傳-1
Insert 方法	新增一個項目到 IList 特定的位置
Remove 方法	移除 IList 中特定的項目

## 了解 IDictionary 介面

.NET 類別函式庫的 System.Collections.IDictionary 介面定義一個標準的 Dictionary 物件該有的方法和屬性。並在許多的 Collection 類型的類別中實作它們，如.NET 中的 HashTable、SortedList 類別都實作了 IDictionary 介面。

IDictionary 介面常用的方法與屬性：

方法與屬性	說明
Add 方法	新增一個項目到 Dictionary，每個項目需由 key/value 物件配對出現。
Clear 方法	清除 Dictionary 中所有的項目
Remove 方法	移除 Dictionary 中的特定項目
Keys 屬性	由 Dictionary 中所有項目的 key 所組成的集合
Values 屬性	由 Dictionary 中所有項目的 value 所組成的集合



## 使用 ArrayList 集合

ArrayList 定義在 System.Collections 命名空間之下，它和陣列非常地類似，同樣地也實作了 IList 介面。ArrayList 物件中的每一個項目，乃是採用循序的方式儲存，並提供一個數值的索引值來存取其中的項目。比陣列有彈性的地方在於它允許你動態地增加 ArrayList 中的項目，陣列則否。

ArrayList 也是參考型別的一種，意味著在使用之前需先建立物件的實體，在 C#或 Visual Basic 中建立物件的標準方式便是使用 new 保留字。ArrayList 常用方法：

- Add 方法：若要新增項目到 ArrayList 可以使用 Add 方法。

你也可以使用 foreach 語法瀏覽 ArrayList 物件的內容，如欲讀取 ArrayList al 的內容，範例如下：

```
Visual Basic
Dim al As New ArrayList()
al.Add("Anita")
al(1) = "Andy"
al.Insert(0, "Mickey")
For Each o As Object In al
    Console.WriteLine(o)
Next
```

C#

```
ArrayList al = new ArrayList();
al.Add("Anita");
al.Add("Andy");
al.Insert(0, "Mickey");
Console.WriteLine("第0項是{0}", al[0]);
foreach (object o in al) {
    Console.WriteLine(o);
}
```

注意，若是自訂的集合物件，若要透過 foreach 方法讀取，你必需為集合類別實作 `IEnumerator` 與 `IEnumerable` 兩個介面，撰寫列舉資料的邏輯。

## 使用 `ArrayList` 的考量

`ArrayList` 適合用在動態新增項目的物件存取上，特別是在新增、刪除集合項目的動作很頻繁的時候。儘可能在 `ArrayList` 中的項目不再變動的情況下，呼叫 `TrimToSize` 方法，固定它的大小。這樣可以最佳化記憶體的使用，不過使用時要小心，若你已經利用 `TrimToSize` 固定它的大小，後續又新增項目到其中，那麼它的效能會變差，因為 `ArrayList` 需動態增加大小。

再者，若放到 `ArrayList` 已經事先排序過，那麼要搜尋其中的項目可以使用 `ArrayList.BinarySearch` 方法會比使用 `Contains` 方法達到較好的效率。最後，請不要使用 `ArrayList` 來存放字串，`StringCollection` 是比較好的選擇。



Hashtable 及 SortedList 集合類別都是 Key/Value 成對出現的資料結構，兩者略有不同，前者會按內部的演算法排序項目，後者則是按照 Key 進行排列。

## 使用 **Hashtable**

Hashtable 是一種索引鍵/值組(Key/Value) 成對出現的資料結構，透過 key 值，以找到對應的內容，它是一種集合型別的物件。

Hashtable 物件中的每個項目存放的順序，和加入到 Hashtable 的先後次序無關。通常 Hashtable 會使用自己的演算法來排序內部的項目，以達最佳執行效能。Hashtable 的各別項目型別是 DictionaryEntry 項目，Key 與 Value 可以取得索引鍵(key)與值。範例如下：

### Visual Basic

```

Dim ht As New Hashtable()
ht.Add(1, "Number 1")
ht.Add("A001", New Employee(101, "Anita"))
ht.Add(DateTime.Now, "Lisa's birthday")

For Each entry In ht
    Console.WriteLine("hashValue:{0}, key:{1}, value:{2}", _
        entry.GetHashCode(), entry.Key, entry.Value)
Next

```

C#

```

Hashtable ht = new Hashtable();
ht.Add(1, "Number 1");
ht.Add("A001", new Employee(101, "Anita"));
ht.Add(DateTime.Now, "Lisa's birthday");

foreach (DictionaryEntry entry in ht)
{
    Console.WriteLine("hashValue:{0}, key:{1}, value:{2}"
    ,
    entry.GetHashCode (), entry.Key, entry.Value);
}

```

結果：

```

hashValue:1782878348, key:2008/11/14 上午12:08:21, value:Lisa'
s birthday
hashValue:710044785, key:A001, value:101,Anita
hashValue:1608268553, key:1, value:Number 1

```

### 使用 **Hashtable** 的考量

若集合中的項目少於 10 個，可改用 ListDictionary 來存放 Key/Value 配對所成的集合，可得較佳效能。

## 使用 **SortedList** 集合

顧名思義，SortedList 代表其中的項目可以根據 key 值進行排序的動作，此外它不允許 key 值重複出現。也因為 SortedList 可以進行一些較複雜的排序動作，所以其執行效率較 Hashtable 慢一些。但因可以以 key 或索引值找尋對應的值，所以使用上較 Hashtable 有彈性。與 Hashtable 一樣各別項目型別是 DictionaryEntry 項目，Key 與 Value 可以取得索引鍵(key)與值。範例如下：

Visual Basic

```

Dim list As New SortedList()
list.Add("1", "Number 1")
list.Add("A001", New Employee(101, "Anita"))
list.Add(DateTime.Now.Day.ToString(), "Lisa's birthday")

Dim keyList As IList = list.GetKeyList()
Dim valueList As IList = list.GetValueList()
Console.WriteLine("用索引取Key, Value. Key:{0}, Value:{1}", _
keyList(2), valueList(2))

```

C#

```
SortedList list = new SortedList();
```

```
list.Add("1", "Number 1");
list.Add("A001", new Employee(101, "Anita"));
list.Add(DateTime.Now.Day.ToString(), "Lisa's birthday");

IList keyList = list.GetKeyList();
IList valueList = list.GetValueList();
Console.WriteLine("用索引取Key, Value. Key:{0}, Value:{1}",
    keyList[2], valueList[2]);
```

結果：

```
用索引取Key, Value. Key:A001, Value:101,Anita
```

### 使用 **SortedList** 物件的考量

儘量不要在以下情況下使用 SortedList：

- 集合中的項目大量變動時，因為新增項目之後需排序，需花費很久時間。若變動性大，不如改用 ArrayList，它也提供排序的功能，但花費的時間比 SortedList 少。
- 不要使用 SortedList 排序字串，會額外花費轉型的負擔，若有需要排序字串，可以改用 StringCollection 集合。



## 使用 HybridDictionary 集合

HybridDictionary 和 Hashtable 類似，用在同時應用在少量資料，或量大資料的物件所成的集合。在集合中項目少的時候，它內部會建立 ListDictionary 物件來存放資料項目。ListDictionary 的特色是使用單向連結串列來實作 IDictionary 介面。HybridDictionary 類別位於 System.Collections.Specialized 命名空間之下，當無法預期集合中的資料量，集合中的項目需要 key/value 配對出現時，你可以使用 HybridDictionary 物件。

若資料項目資料很多時，HybridDictionary 物件內部會切換成利用 Hashtable 來儲存資料。不過千萬不要使用 HybridDictionary 進行排序，它的效果並不是很好。

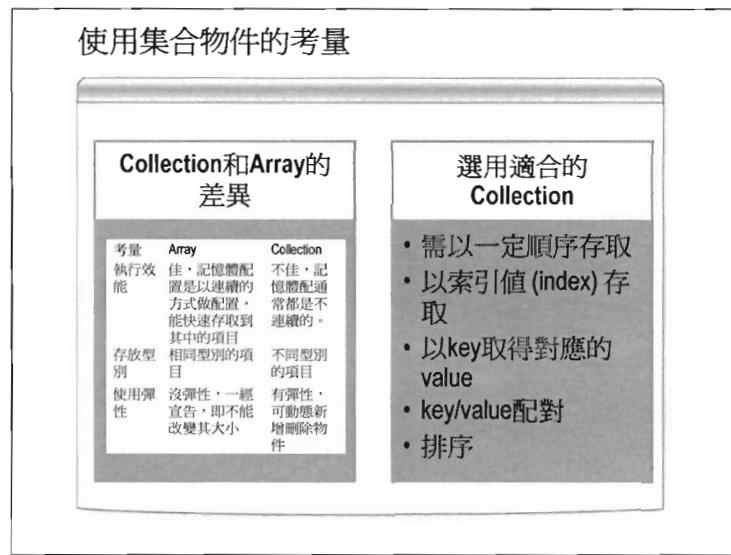
使用時先引用 System.Collections 與 System.Collections.Specialized 命名空間：

```
Visual Basic
Imports System.Collections
Imports System.Collections.Specialized
```

```
C#
using System.Collections;
```

```
using System.Collections.Specialized;
```

新增資料到 HybridDictionary 時要注意，索引鍵不能是 Null (C# 在 Visual Basic 中不可以為 Nothing)，不過值可以是 Null 或 Nothing。



## 使用集合物件的考量

.NET 對不同的需求提供了許多不同的 Collection 型別供程式設計師使用。除本章提及的 Collection 之外，尚包含許多特殊的 Collection，若要得到這方面更詳細的資訊，可以參閱 MSDN 文件。

### Collection 和 Array 的差異

Collection 和 Array 非常類似，大部份的 Collection 可以使用類似存取陣列的索引值來存取內部的項目，在選用時可以考慮以下狀況：

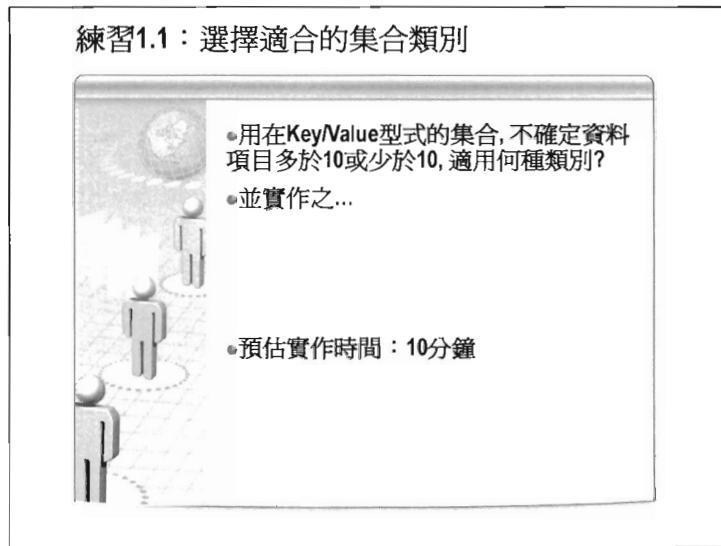
考量	Array	Collection
執行效能	佳，記憶體配置是以連續的方式做配置，能快速存取到其中的項目	不佳，記憶體配通常都是不連續的。
存放型別	相同型別的項目	不同型別的項目
使用彈性	沒彈性，一經宣告，即不能改變其大小	有彈性，可動態新增刪除物件

### 選用適合的 Collection

Collection 的總類繁多，可按下列需求選用最適用的：

項目	說明
需以一定順序存取	若要以先進先出的順序存取內部的項目，可以採用 Queue，或要以後進先出的順序存取內部的項目，可以採用 Stack。

類似透過陣列的方式存取—以索引值(index)存取	若要使用索引值存取內部項目，可以使用 ArrayList、StringCollection、NameValuePairCollection，它們都擁有一個以 0 為基底的索引。
以 key 取得對應的 value	<p>若其中的 key/value 項目可能含多種型別，可以使用 Hashtable。若要以 key 值存取 value 可以使用 Hashtable、SortedList、StringDictionary。</p> <ul style="list-style-type: none"> <li>• Hashtable：適用於在隨機情況下以 key 搜尋陣列。</li> <li>• StringDictionary：適合隨機搜尋字串。</li> <li>• ListDictionary：當集合中的項目少於 10 個時，最適合。</li> </ul>
key/value 配對	若要一個 key 值對應一個 value，可以使用任何支援 IDictionary 介面的集合。若要一個 key 值對應多個 value，可以使用 KeyValuePairCollection
排序	<p>若有排序的考量，可以使用三種 Collection：</p> <ul style="list-style-type: none"> <li>• SortedList：可以根據 IComparer 實作的排序方式進行排序。SortedList 在建構集合的過程中，會事先排序其中的項目。因此，建立 SortedList 物件的過程中，要花費的時間較久。不過，若你後續修改 SortedList 中已存的資料，或新增一些項目到其中，它會自動重新排序，而且效率相當好。簡單的說，SortedList 適合用在資料幾乎固定，且異動幅度較小的情況下。</li> <li>• Hashtable：根據 key 的雜湊碼(Hash code)排序。</li> <li>• ArrayList：使用 Sort 方法排序。若你要顯示的資料是唯讀的，建議使用 ArrayList，效能上會比使用 SortedList 好。</li> <li>• KeyValuePairCollection：適用在排序的字串。</li> </ul>



### 練習 1.1：選擇適合的集合類別

目的：

了解集合之間的特性，選擇適合的集合類別。

功能描述：

在這個練習中，要使用 Key/Value 型式的集合，不確定資料項目多於 10 或少於 10，適用何種類別？\_\_\_\_\_

請在 Windows 應用程式實作這個類別。

預估實作時間：10 分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod01\_1。
3. 在 Form 放上 Button 及 ListBox 控制項。
4. 在 Form1 程式檔案最上方，加入以下命名空間：

```
Visual Basic
Imports System.Collections
Imports System.Collections.Specialized
```

```
C#
using System.Collections;
using System.Collections.Specialized;
```

5. 在 Button1 的 Click 事件加入以下程式：

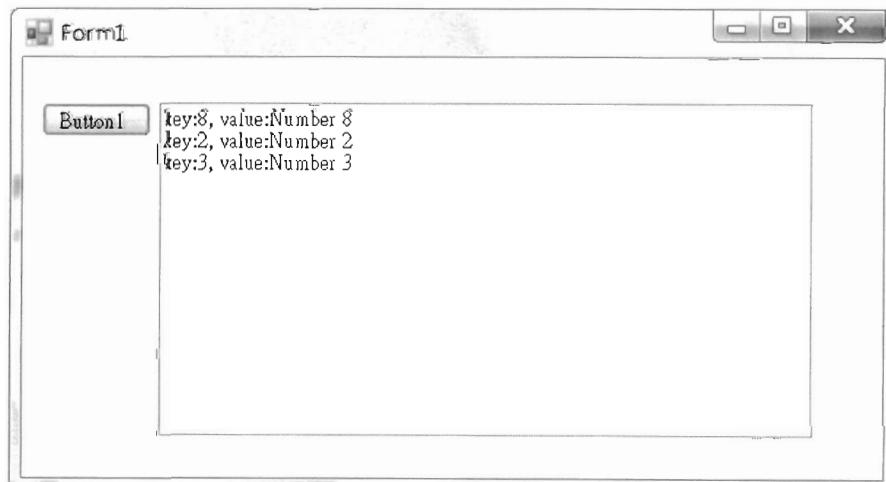
```
Visual Basic
Dim hyper As New HybridDictionary
hyper.Add(8, "Number 8")
hyper.Add(2, "Number 2")
hyper.Add(3, "Number 3")

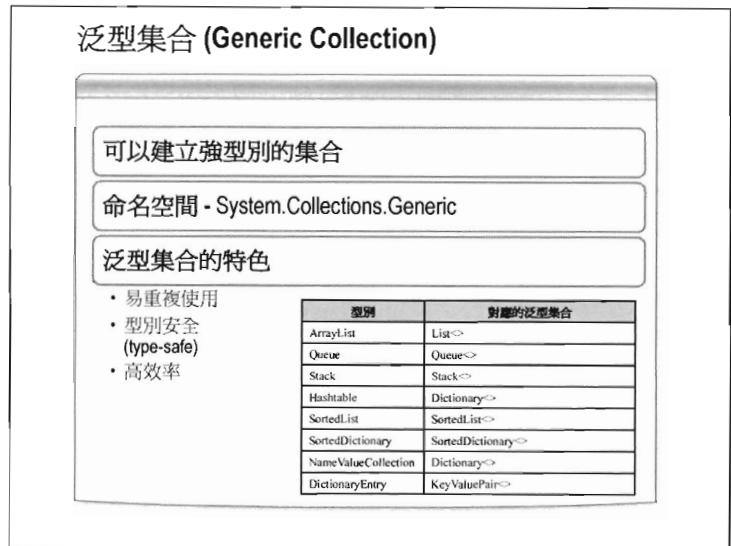
For Each entry In hyper
    ListBox1.Items.Add(String.Format(
        "key:{0}, value:{1}", entry.Key, entry.Value))
Next
```

```
C#
HybridDictionary hyper = new HybridDictionary();
hyper.Add(8, "Number 8");
hyper.Add(2, "Number 2");
hyper.Add(3, "Number 3");

foreach (DictionaryEntry entry in hyper){
    ListBox1.Items.Add(String.Format(
        "key:{0}, value:{1}", entry.Key, entry.Value));
}
```

6. 執行應用程式。





## 泛型集合 (Generic Collection)

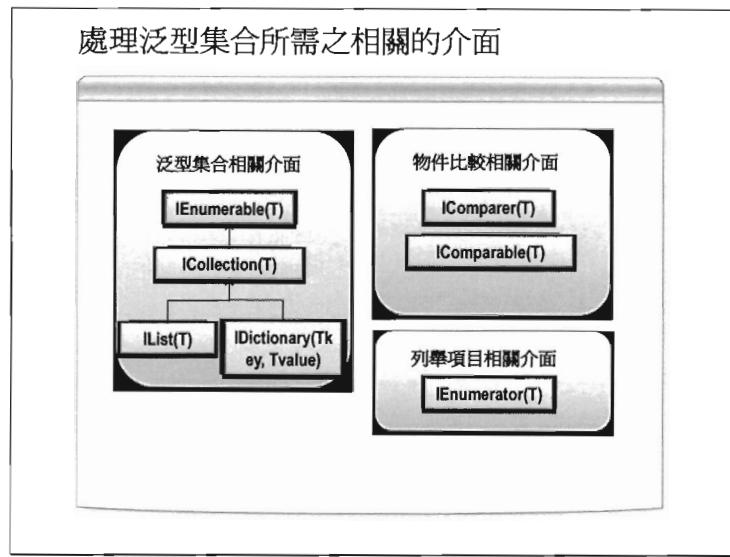
在前面的小節中介紹的集合，其中的每個項目都只能夠以 Object 型別來存取。你需要將它們手動轉換成適當的型別，才可以操作之。.NET 2.0 時中引進 Generic Collection (泛型集合) 來解決這個問題。System.Collections.Generic 與 System.Collections.ObjectModel 命名空間下包含許多定義泛型集合 (Generic Collection) 的介面和類別，讓你建立強型別的集合。和非泛型的集合物件相形之下，可以得到更佳的效能。

泛型集合的特色是易重複使用性、具備型別安全 (type-safe)，和高效率。因此微軟的 MSDN 文件上建議，所有使用.NET 2.0 版以後的應用程式，都應使用新的泛型集合類別，而不要使用舊版的非泛型對應類別，例如使用 List<>，不要使用 ArrayList。

下表是.NET 集合型別與對應的泛型集合之列表：

型別	對應的泛型集合
ArrayList	List<>
Queue	Queue<>
Stack	Stack<>
Hashtable	Dictionary<>
SortedList	SortedList<>
ListDictionary	Dictionary<>
HybridDictionary	Dictionary<>

型別	對應的泛型集合
SortedDictionary	SortedDictionary<>
NameValueCollection	Dictionary<>
DictionaryEntry	KeyValuePair<>
StringCollection	List<String>
StringDictionary	Dictionary<String>



## 處理泛型集合所需之相關的介面

泛型集合與 Object-Based 集合所需的功能是類似的，都一樣需要對集合內容進行新增、插入、查詢元素所屬之索引位置，進行排序以及項目的列舉。所以也據此定義了一組泛型專用的集合介面，它們皆屬於 System.Collections.Generic 命名空間，說明如下：

- **IEnumerable(T)**，定義專為泛型集合所需的列舉集合項目的功能，有實作這個介面的集合才可以使用 For Each(或 foreach) 語法列舉集合項目。
- **ICollection(T)**，擴充了 **IEnumerable(T)** 泛型，多了像是 Count 屬性、CopyTo 方法、Clear 方法清除項目內容等功能。
- **IList(T)**，定義了以整數索引存取物件的泛型集合類型，除了擴充 **IEnumerable**、**ICollection** 之外，還新增了 Add、Insert、Remove... 等功能定義。
- **IDictionary(TKey, TValue)**，定義了以 Key/Value 存取物件的泛型集合類型，除了擴充 **IEnumerable**、**ICollection** 之外，還新增了 Add、Insert、Remove... 等功能定義。
- **IComparer(T)**，定義兩個泛型物件比較的功能。
- **IEnumerator(T)**，定義簡單反覆運算列舉值的功能。



## List 泛型類別

在 System.Collections.Generic 命名空間下的 List 類別是對應到 ArrayList 的泛型類別，使用大小可動態增加的陣列來實作 IList 泛型介面。你可以使用數值索引來存取這個集合中的項目，索引是以零開始。

List 泛型類別通常使用來建立簡單且具型別安全 (Type-Safe) 的物件所成的集合，支援 IList 介面，也可以使用 foreach 語法瀏覽其中的項目。

List 泛型類別常用方法：

- Add 方法：將物件加入至 List 的結尾。List 中可以接受 Null 參照 (即 Visual Basic 中的 Nothing)。項目新增時，項目的型別必需符合 List 的泛用型別參數。
- ForEach 方法：指定個別項目要執行的方法。

以下為 List 泛型類別範例，使用 Add 方法加入紀念日的日期，使用 ForEach 方法逐項列出：

```
Visual Basic
Imports System
Imports System.Collections.Generic
```

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim anniversaries As New List(Of DateTime)
    anniversaries.Add(New DateTime(1911, 10, 10))
    anniversaries.Add(New DateTime(1945, 10, 25))
    anniversaries.Add(New DateTime(1931, 4, 4))
    anniversaries.ForEach(AddressOf PrintAnniversaries)
End Sub

Sub PrintAnniversaries(ByVal day As DateTime)
    Console.WriteLine(day)
End Sub

```

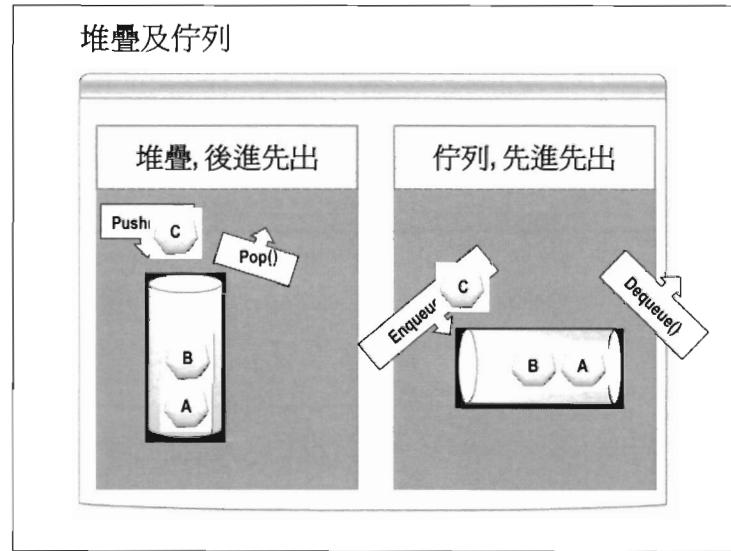
```

C#
using System;
using System.Collections.Generic;
private void button1_Click(object sender, EventArgs e)
{
    List<DateTime> anniversaries = new List<DateTime>();
    anniversaries.Add(new DateTime(1911, 10, 10));
    anniversaries.Add(new DateTime(1945, 10, 25));
    anniversaries.Add(new DateTime(1931, 4, 4));

    anniversaries.ForEach(delegate(DateTime day) {
        Console.WriteLine(day);
    });
}

```

此外，當集合中的項目為實值型別 (Value-Typed) 時，泛型集合型別的效能通常比對應的非泛型集合更佳，因為泛型不需要將集合項目進行 Boxing 處理。



堆疊及佇列兩種集合類型是循序存取集合項目的集合方式。差別在這兩者的進出順序不同，堆疊為後進先出，佇列為先進先出。

## 使用堆疊 (Stack)

堆疊類別提供循序存取其中項目的能力，運作時採後進先出 (LIFO) 的策略。堆疊和 ArrayList 都屬於一種集合，和 ArrayList 較不同的基本操作方法：

- Push 方法：新增一個項目到堆疊頂端。
- Pop 方法：從堆疊頂端取出一個項目。

存取 Stack 的範例如下：

Visual Basic

```
Dim st As New Stack(Of String)
st.Push("A")
st.Push("B")
st.Push("C")

Console.WriteLine(st.Pop())
Console.WriteLine(st.Pop())
Console.WriteLine(st.Pop())
```

C#

```
Stack<String> st = new Stack<String>();
st.Push("A");
st.Push("B");
st.Push("C");

Console.WriteLine(st.Pop());
Console.WriteLine(st.Pop());
Console.WriteLine(st.Pop());
```

列出結果是：

```
C  
B  
A
```

## 使用堆疊的考量

堆疊是一個簡單的後進先出資料結構，在以下情況可以使用之：

- 若需要按後進先出順序處理資料時
- 當你使用完資料之後就可丟棄此資料項目時
- 建立時最好設定初始的容量 (Capacity)，避免無意義的浪費記憶體。

## 使用佇列 (Queue)

Queue 類別和 Stack 類別一樣提供循序存取的能力，但其中 Queue 類別是按內部項目新增的先後順序來存取其中的項目。也就是說佇列是先進先出 (First In , First Out , FIFO) 類型的資料結構，第一個被新增的項目，將會第一個被刪除；同理，第 n 個被新增的項目，將會第 n 個被刪除。.NET 的提供佇列類別，支援上述這種 FIFO 的架構。常見的方法和屬性如下：

- Enqueue 方法：若要新增一個項目到佇列的後面，可以用 Enqueue 方法。
- Dequeue 方法：若要從佇列前方讀取、或移除一個項目，可以用 Dequeue 方法。Dequeue 方法不僅僅可以從佇列中移除一個項目，還會回傳目前被移除的這個項目。若試著從一個空的佇列中移除一個項目，將會觸發 InvalidOperationException 例外。你可以使用錯誤處理常式來捕捉這個錯誤。

**Visual Basic**

```

Dim q As New Queue(Of String)
q.Enqueue("A")
q.Enqueue("B")
q.Enqueue("C")

Console.WriteLine(q.Dequeue())
Console.WriteLine(q.Dequeue())
Console.WriteLine(q.Dequeue())

```

**C#**

```

Queue<String> q = new Queue<String>();
q.Enqueue("A");
q.Enqueue("B");
q.Enqueue("C");

Console.WriteLine(q.Dequeue());
Console.WriteLine(q.Dequeue());
Console.WriteLine(q.Dequeue());

```

列出結果是：

A

B

C

### 使用 **Queue** 物件的考量

**Queue** 是先進先出的資料結構，當你需要循序使用資料時，它是一個很好的選擇。不過，若你想要以 **String** 的方式使用資料，改用 **NameValueCollection** 可能較為適當。

佇列資料結構採用先進先出的方式存放資料，和堆疊 (Stack) 恰恰相反。若要新增一個項目到佇列的後面，可以叫用 **Enqueue** 方法；若要從佇列前方讀取、或移除一個項目，可以叫用 **Dequeue** 方法。

### 練習1.2：選用適當的泛型集合

- 暫存性的資料，先進先出，選用何種類別？
- 如何加入新項目，取出項目，清除所有項目？
- 預估實作時間：10分鐘

### 練習 1.2：選用適當的泛型集合

目的：

這個練習要你了解泛型集合類別的特性，並選出適合的泛型集合類別，以了學會使用適當的方法操作集合項目。

功能描述：

在這個練習中，請找出適合儲存銀行客戶臨櫃排隊號碼牌的泛型集合？\_\_\_\_\_

如何新增排隊代碼，移除項目，清除所有項目？\_\_\_\_\_

預估實作時間：10分鐘

實作步驟：

1. 從「Practices\VB 或 CS\Mod01\_2\Starter\Mod01\_2」開啓 Mod01\_2.sln 檔案。
2. 開啓 Form1 程式碼檢視，宣告類別層級變數如下：

Visual Basic

```
Dim nextNum As Integer = 1
Dim PaiDui As New Queue(Of Integer)
```

C#

```
int nextNum = 1;
Queue<int> PaiDui = new Queue<int>();
```

3. 找到 ShowCount 副程式，在 Label1 上顯示目前狀況：

Visual Basic

```
label1.Text = string.Format("目前尚有{0}人在等待中，下個號碼是{1}", PaiDui.Count, nextNum);
```

C#

```
label1.Text = string.Format("目前尚有{0}人在等待中，下個號碼是{1}", PaiDui.Count, nextNum);
```

4. 在 button1 的 Click 事件實作「抽號碼牌」的程序：

Visual Basic

```
PaiDui.Enqueue(nextNum)
nextNum += 1
ShowCount()
```

C#

```
PaiDui.Enqueue(nextNum);
nextNum += 1;
ShowCount();
```

5. 在 button2 的 Click 事件實作「第 n 號櫃台」的程序：

Visual Basic

```
If PaiDui.Count = 0 Then
    MessageBox.Show("沒人排隊")
    Return
End If
Dim btn As Button = CType(sender, Button)
Dim num As Integer = PaiDui.Dequeue()
ShowCount()
label2.Text = String.Format("{0}請到{1}", num, btn.Text)
```

C#

```
if (PaiDui.Count == 0)
{
    MessageBox.Show("沒人排隊");
    return;
}
Button btn = (Button)sender;
int num = PaiDui.Dequeue();
ShowCount();
```

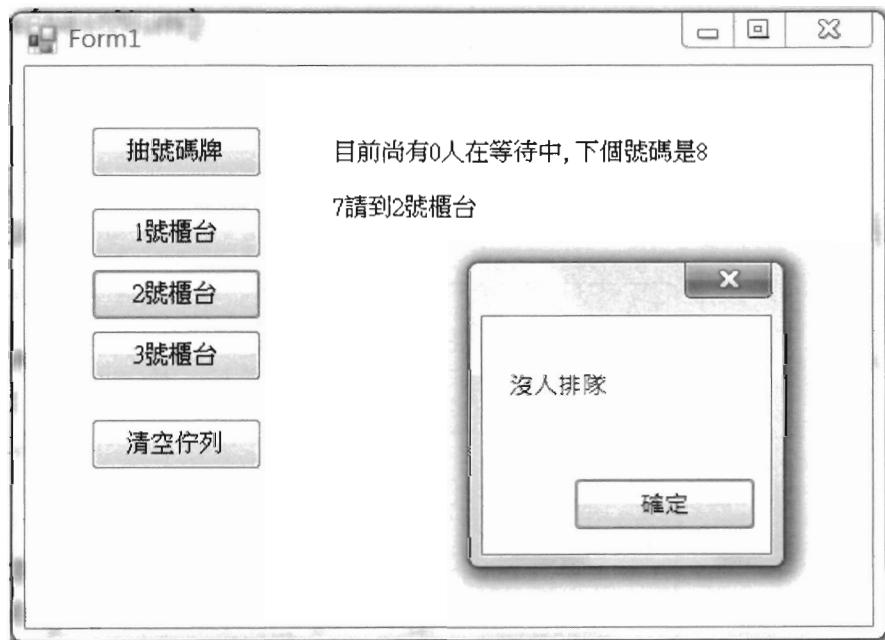
```
label2.Text = string.Format ("{0}請到{1}", num, btn.Text  
);
```

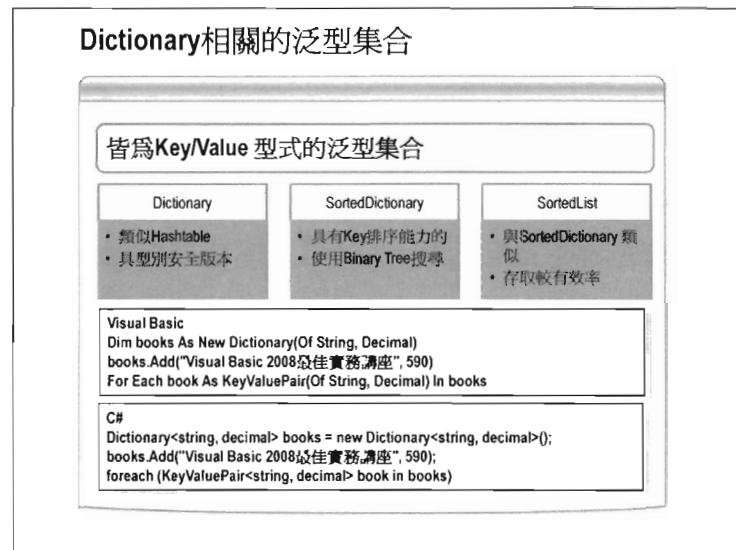
6. 在 button3 的 Click 事件實作「清空佇列」的程序：

Visaul Basic  
PaiDui.Clear()  
ShowCount()

C#  
PaiDui.Clear();  
ShowCount();

7. 執行結果：





## Dictionary 相關的泛型集合

泛型集合有三個 Key/Value 型式的類別，分別是 Dictionary、SortedDictionary、SortList。它們皆可使用泛型型別的索引鍵存取泛型型別的值。

Dictionary 泛型類別中的每個索引鍵都必須是唯一的。而其中的每一個項目都是由一個值及其索引鍵組成的 KeyValuePair 結構。使用 KeyValuePair 類別時，需要傳兩個參數，代表索引鍵 (Key) 與值 (Value) 的型別。除了使用 KeyValuePair 結構存取項目，也可以使用 Keys 與 Values 屬性。

### Dictionary 泛型類別

Dictionary 泛型類別是 Dictionary 泛型的型別安全版本 (type-safe)。它提供索引鍵與值的對應，也就是說加入 Dictionary 泛型中的每一個項目都是由值及其關聯索引鍵所組成。Dictionary 泛型類別的特色是使用索引鍵來取值的速度非常快。以下範例使用 Add 加入集合內容，For Each 語法取得集合內的資訊：

**Visual Basic**

```
Dim books As New Dictionary(Of String, Decimal)
books.Add("Visual Basic 2008最佳實務講座", 590)
books.Add("ADO.NET 3.5精研講座", 850)
```

```

books.Add("ASP.NET 3.5最佳實務講座 Using Visual C#", 680)
books.Add("Visual C# 2008精研講座", 650)

For Each book As KeyValuePair(Of String, Decimal) In books
    Console.WriteLine("書名:{0}, 售價:{1}", book.Key, book.Value)
Next

```

```

C#
Dictionary<string, decimal> books = new Dictionary<string, decimal>();
books.Add("Visual Basic 2008最佳實務講座", 590);
books.Add("ADO.NET 3.5精研講座", 850);
books.Add("ASP.NET 3.5最佳實務講座 Using Visual C#", 680);
books.Add("Visual C# 2008精研講座", 650);

foreach (KeyValuePair<string, decimal> book in books) {
    Console.WriteLine("書名:{0}, 售價:{1}",
                      book.Key, book.Value);
}

```

## SortedDictionary 泛型類別

SortedDictionary 泛型類別與 Dictionary 都提供了類似的屬性及方法，但它的內部結構使用二進位搜尋樹狀目錄。項目根據索引鍵排序，索引鍵必須唯一不可重複。除了使用 KeyValuePair 結構存取項目，也可以使用 Keys 與 Values 屬性存取。以下範例使用 Keys 屬性及索引鍵的方式取值：

```

Visual Basic
Dim books As New SortedDictionary(Of String, Decimal)
books.Add("Visual Basic 2008最佳實務講座", 590)
books.Add("ADO.NET 3.5精研講座", 850)
books.Add("ASP.NET 3.5最佳實務講座 Using Visual C#", 680)
books.Add("Visual C# 2008精研講座", 650)
For Each key As String In books.Keys
    Console.WriteLine("書名:{0}, 售價:{1}", key, books(key))
Next

```

```

C#
SortedDictionary<string, decimal> books = new SortedDictionary<string, decimal>();
books.Add("Visual Basic 2008最佳實務講座", 590);
books.Add("ADO.NET 3.5精研講座", 850);
books.Add("ASP.NET 3.5最佳實務講座 Using Visual C#", 680);

```

```

books.Add("Visual C# 2008精研講座", 650);
foreach (string key in books.Keys) {
    Console.WriteLine("書名:{0}, 售價:{1}", key, books[key]);
}

```

## SortedList 泛型類別

SortedList 泛型類別擁有與 Dictionary 及 SortedDictionary 類似的屬性與方法，不過 SortedList 多了 IndexOfKey、IndexOfValue、RemoveAt 等方法。以下範例是將既有的 books (Dictionary 泛型類別) 在 SortedList 的建構函式填入：

Visual Basic

```

Dim copy As New SortedList(Of String, Decimal)(books)
For Each book As KeyValuePair(Of String, Decimal) In copy
    Console.WriteLine("書名:{0}, 售價:{1}", book.Key, book.Value)
Next

```

C#

```

SortedList<string, decimal> copy = new SortedList<string, decimal>(books);
foreach (KeyValuePair<string, decimal> book in copy)
{
    Console.WriteLine("書名:{0}, 售價:{1}", book.Key, book.Value);
}

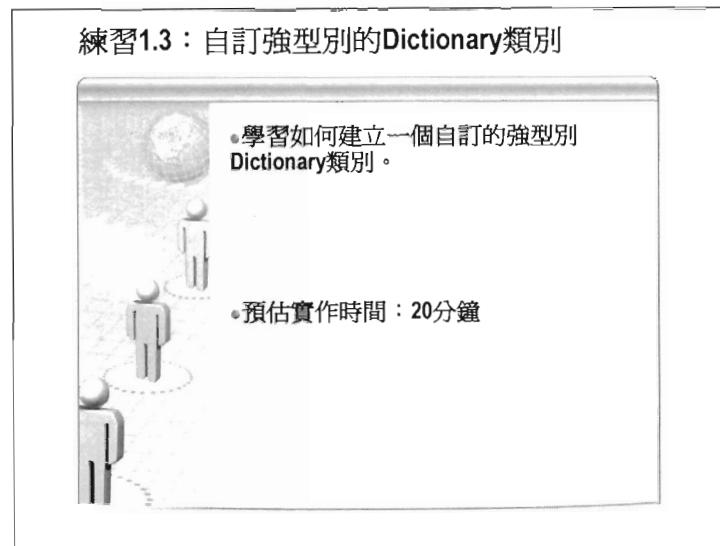
```

## 使用 Dictionary 的考量

此三個類別功能都相當類似，使用時請考量以下情況：

- 是否需要排序，若不需要請使用 Dictionary 類別，否則請考慮另外兩者。
- SortedList 比 SortedDictionary 佔用較少的記憶體空間。
- SortedList 內容維護兩組(Keys/Values)陣列，因此若需用整數索引值存取 Keys 或 Values 時會比 SortedDictionary 有效率。
- 如果將所有項目一次填入集合使用 SortedList 會比 SortedDictionary 快。

- 如果是逐項插入未排序資料，使用 SortedDictionary 會比 SortedList 快。



### 練習 1.3：選用適當的泛型集合

目的：

這個練習要學習如何建立一個自訂的強型別 Dictionary 類別。

功能描述：

這個練習需要建立一個自訂的具有 Key/Value 的 type safe 集合類別，請問應該繼承或實作哪個類別？

這個練習中已建立一個 Employee 類別，但還需要再建立一個 Employee 物件專用的 type safe 集合類別。除此之外，可以利用網頁資料繫結的技術與集合物件進行資料繫結。

預估實作時間：20 分鐘

實作步驟：

1. 從『Start』→『Program』→『Microsoft Visual Studio 2008』→『Microsoft Visual Studio 2008』，啓動 Visual Studio 開發環境。
2. 開啓位於『\U2956\Practices\VB 或 CS\Mod01\_3\Starter\Mod01\_3』的 Web Site。

3. 開啓在 App\_Code 裡的 EmployeesCollection.vb(for Visual Basic) 或 EmployeesCollection.cs(for C#)。
4. 在「TODO: 在這裡建立 EmployeesCollection 類別」的下一行加上 EmployeesCollection 類別，並繼承 Dictionary 類別，程式如下：

```
Visual Basic
Public Class EmployeesCollection
    Inherits Dictionary(Of Integer, Employee)

End Class
```

```
C#
public class EmployeesCollection: Dictionary<int, Employee>
{
```

}

5. 在類別裡撰寫一個 Add 方法只有 Employee 型別的 Value 參數。

```
Visual Basic
Public Overloads Sub Add(ByVal Value As Employee)
    Add(Value.ID, Value)
End Sub
```

```
C#
public void Add(Employee Value) {
    Add(Value.ID, Value);
}
```

6. 開啓 Default.aspx 的程式碼檢視，在 Page 的類別層級建立一個 EmployeesCollection 型別的物件。

```
Visual Basic
Dim employees As New EmployeesCollection()
```

```
C#
EmployeesCollection employees = new EmployeesCollection();
```

7. 在 Page\_Load 事件中為員工集合加入員工項目，並與 ListBox 及 GridView 做資料繫結，程式如下：

## Visual Basic

```

employees.Add(101, New Employee(101, "Anita"))
employees.Add(New Employee(102, "Andy"))
employees.Add(New Employee(103, "Mary"))
If Not Page.IsPostBack Then
    GridView1.DataSource = employees.Values
    GridView1.DataBind()
    ListBox1.DataSource = employees
    ListBox1.DataTextField = "Key"
    ListBox1.DataBind()
End If

```

## C#

```

employees.Add(101, new Employee(101, "Anita"));
employees.Add(new Employee(102, "Andy"));
employees.Add(new Employee(103, "Mary"));
if (!Page.IsPostBack) {
    GridView1.DataSource = employees.Values;
    GridView1.DataBind();
    ListBox1.DataSource = employees ;
    ListBox1.DataTextField = "Key";
    ListBox1.DataBind();
}

```

8. 在 ListBox 的 SelectedIndexChanged 事件顯示使用者所選的項目在 Label1 控制項上。

## Visual Basic

```

If ListBox1.SelectedIndex < 0 Then Return

Dim key As Integer = Int32.Parse(ListBox1.SelectedValue)
Label1.Text = key & "," & employees(key).Name

```

## C#

```

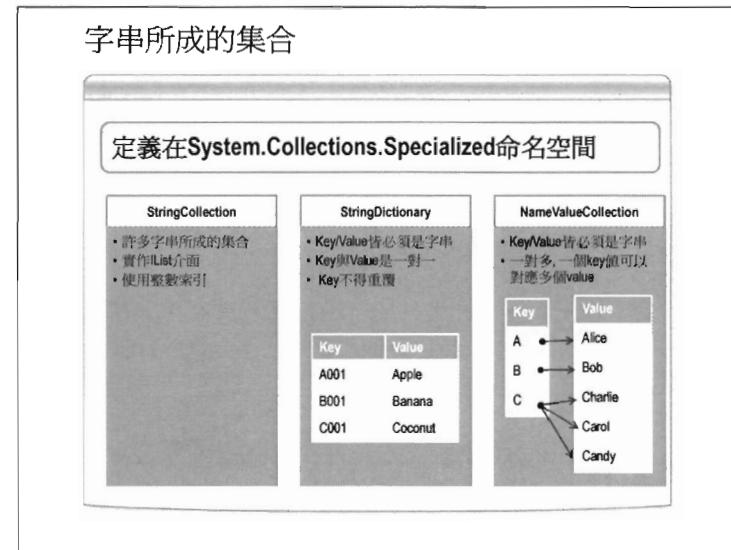
if (ListBox1.SelectedIndex<0)
    return ;

int key = int.Parse (ListBox1.SelectedValue);
Label1.Text = key + "," + employees[key].Name ;

```

9. 執行結果如下：





在 System.Collections.Specialized 命名空間之下有一些特別的集合，其中包含 String 專用的集合：

- **StringCollection**：類似 ArrayList 只不過它只能存 String 型別。
- **StringDictionary**：類似 HashTable，只不過它的 Key/Value 都必須是 String 型別。
- **NameValueCollection**：也是 Key/Value 的集合，但它允許一個 Key 對多個 Value。

## StringCollection 類別

和 ArrayList 相同的是它們都實作 IList 介面，因此 StringCollection 類別也有一個 Count 屬性，代表其中包含的項目總數，不過 StringCollection 並沒有 Capacity 屬性，它的記憶體配置和 ArrayList 有些不同。

StringCollection 代表許多字串所成的集合，值得注意的是這個類別的靜態成員 (Static Member) 實作時加入了多執行緒同時叫用的同步考量，因此能夠在多條執行緒同時使用到靜態成員時能確保資料同步，保障執行緒安全。不過，對於非靜態成員則不能確保執行緒安全，當

一條執行緒在瀏覽集合的內容時，若有另一條執行緒企圖修改集合中的項目，將觸發例外。

#### Visual Basic

```
Dim fruit As New StringCollection()
fruit.Add("Apple")
fruit.Add("Pineapple")
fruit.Add("Banana")
fruit.Insert(0, "Coconut")
Console.WriteLine("第一個是" + fruit(0))
Console.WriteLine("最後一個是" + fruit(fruit.Count - 1))
```

#### C#

```
StringCollection fruit = new StringCollection();
fruit.Add("Apple");
fruit.Add("Pineapple");
fruit.Add("Banana");
fruit.Insert(0, "Coconut");
Console.WriteLine("第一個是" + fruit[0]);
Console.WriteLine("最後一個是" + fruit[fruit.Count - 1]);
```

### 使用 **StringCollection** 注意事項

**StringCollection** 是由字串組成的集合，你可以利用它來儲存與操作字串資料，由其是在集合中的字串項目變動頻繁的情況下。此外，記得不要利用 **StringCollection** 排序字串。

### StringDictionary 類別

**StringDictionary** 內部是一個 **Hashtable**，其中項目的 **key/value** 型別都是 **string**。

**StringDictionary** 類別內部實作一個 **Hashtable** 資料結構，換句話說，**StringDictionary** 內的 **DictionaryEntry** 項目是一個 **key/value** 配對出現的項目，但 **key** 值的型別皆為字串，對應的 **value** 之型別也為字串。你可以使用 **key** 值來取得對應的 **value**。

Key	Value
A001	Apple
B001	Banana
C001	Coconut

**Visual Basic**

```

Dim fruit As New StringDictionary()
fruit.Add("A", "Apple")
fruit.Add("B", "Banana")
fruit.Add("C", "Coconut")
If fruit.ContainsKey("A") Then
    Console.WriteLine("A:" + fruit("A"))
Else
    Console.WriteLine("沒有Key A")
End If

If fruit.ContainsValue("Pineapple") Then
    Console.WriteLine("有Pineapple")
Else
    Console.WriteLine("沒有Pineapple")
End If

```

**C#**

```

StringDictionary fruit = new StringDictionary();
fruit.Add("A", "Apple");
fruit.Add("B", "Banana");
fruit.Add("C", "Coconut");
if (fruit.ContainsKey("A"))
    Console.WriteLine("A:" + fruit["A"]);
else
    Console.WriteLine("沒有Key A");

if (fruit.ContainsValue("Pineapple"))
    Console.WriteLine("有Pineapple");
else
    Console.WriteLine("沒有Pineapple");

```

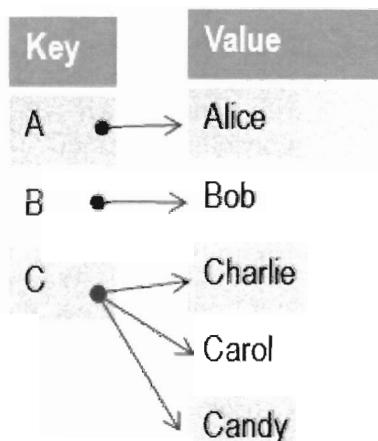
**使用 StringDictionary 注意事項**

StringDictionary 的 key 值是由強型別的字串組成，而不是 object 型別。若集合中的項目變動頻率不大時，可以使用 StringDictionary。由於 StringDictionary 底層是一個 Hashtable，因此將根據 key 值進行排

序。這也代表一件事，若你的集合中所存的資料項目都是字串，最好選用 StringDictionary，不要使用 Hashtable，效能會較好。

## NameValueCollection 類別

NameValueCollection 類別中的項目也是 key/value 成對出現，但較特殊的一個 key 值可以對應多個 value。換言之，一個 string 型別的 key 可以對應多個字串，有點類似一對多的概念。



NameValueCollection 中一個 key 可以有多個對應的字串值。你可以針對每個 key 值叫用 GetValues 方法，傳入 key 值(也可傳入索引位置)來讀取相對的 value，如以下範例傳入 key 值讀取對應的 value：

```

Visual Basic
Dim member As New NameValueCollection()
member.Add("A", "Anita")
member.Add("A", "Alice")
member.Add("B", "Bob")
member.Add("C", "Candy")
member.Add("C", "Charlie")
member.Add("C", "Carol")

For Each k As String In member.Keys
    Console.WriteLine("Key:{0}", k)
    Console.WriteLine(vbTab & "Values:" & _
        String.Join(", ", member.GetValues(k)))
Next

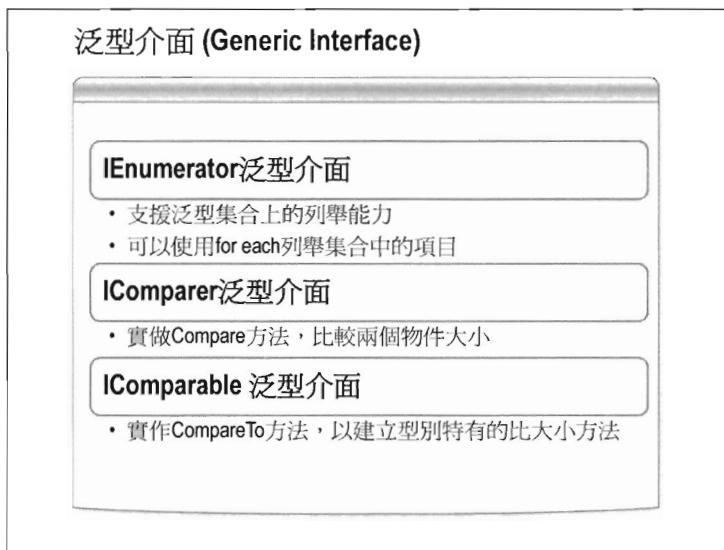
```

```
NameValueCollection member = new NameValueCollection();
member.Add("A", "Anita");
member.Add("A", "Alice");
member.Add("B", "Bob");
member.Add("C", "Candy");
member.Add("C", "Charlie");
member.Add("C", "Carol");

foreach (string k in member.Keys)
{
    Console.WriteLine("Key:{0}", k);
    Console.WriteLine ("\tValues:" +
        string.Join(", ", member.GetValues(k)));
}
```

## 使用 **NameValueCollection** 物件的考量

**NameValueCollection** 是一個依據字串型別的 key 與 Value 值排序好的集合，你可以利用 key 或索引值來存取其中的項目。而且 key 值是可以重複的。它適合使用在經常性變動的資料上，尤其是常常需要新增或刪除其中的項目時。



## 泛型介面 (Generic Interface)

泛型集合類型物件實做了一些特殊的介面，如 `ICollection`，`IEnumerable`、`IEnumerable`、`ICollection`、`IList` 等等。你可以利用這些介面存取集合中的項目，而兼顧型別安全的特色。

### IEnumerator 泛型介面

`IEnumerator` 泛型介面支援泛型集合上的列舉能力，實作此介面的泛型集合能夠讓 C# 或 Visual Basic 語言透過 `for each` 的語法列舉集合中的項目。

### IComparer 泛型介面

`IComparer` 泛型介面常和 `System.Collections.Generic.List.Sort` 與 `System.Collections.Generic.List.BinarySearch` 方法搭配使用，以自訂集合內項目的排序方式，像 `Generic SortedDictionary` 與 `Generic SortedList` 就實作了這個介面。

這個介面中包含一個 `Compare` 方法，比較兩個物件大小，並利用回傳值指出一個物件是等於、小於或大於另一個物件，以支援排序能力。也就是說，若 `Compare` 方法傳回 0 時，表示兩個物件排序相同。若 `Compare` 方法傳回小於零的值時，第一個物件小於第二個物

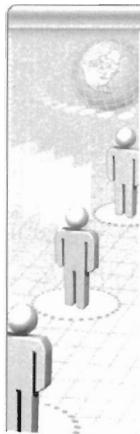
件。若 Compare 方法傳回大於零的值時，第一個物件大於第二個物件。

### **IComparable 泛型介面**

IComparable 泛型介面定義一個通用的比較方式，以比對集合中項目的大小。你可以實作此介面，來撰寫比較邏輯，以方便排序。

實值型別 (value type)或類別 (class) 會實作 IComparable 介面的 CompareTo 方法，以建立型別特有的比較方法，對集合中的項目進行排序。

### 練習1.4：建立具有比較功能的自訂類別



- 這個練習要建立一個員工類別，在集合中員工必須按照年資排列，較資深的員工必須排在前面。
- 請問員工類別必須實作什麼介面，才可以達到這個目的？
- 預估實作時間：20分鐘

### 練習 1.4：建立具有比較功能的自訂類別

**目的：**

學會辨識泛型介面的用途，並套用在適當的情境。這個練習會在自訂的類別中提供具有比較功能方法，以便能夠在集合中進行排序。

**功能描述：**

這個練習要建立一個員工類別，在集合中員工必須按照年資排列，較資深的員工必須排在前面。

請問員工類別必須實作什麼介面，才可以達到這個目的？

**預估實作時間：20分鐘**

**實作步驟：**

1. 從『U2956\Practices\{VB 或 CS\}\Mod01\_4\Starter\Mod01\_4』開啓 Mod01\_4.sln 檔案。
2. 開啓 Employee.vb(for Visual Basic)，在 Employee 類別加上實作 IComparable 泛型介面，

<pre>Visual Basic Public Class Employee     Implements IComparable(Of Employee)</pre>
---

打完實作介面之後按「Enter」鍵，會自動在類別最後出現 CompareTo 方法，如下：

```
Public Function CompareTo(ByVal other As Employee) As Integer
Implements System.IComparable(Of Employee).CompareTo
End Function
```

在方法裡寫上如下的程式：

```
Return Me.HireDate.CompareTo(other.HireDate)
```

3. 開啓 Employee.cs(for C#)，在 Employee 類別加上實作 IComparable 泛型介面。

```
C#
class Employee:IComparable<Employee>
{
```

按一下介面名稱下方的智慧標籤，會自動在類別最後出現 CompareTo 方法，如下：

```
#region IComparable<Employee> 成員
public int CompareTo(Employee other)
{
    throw new NotImplementedException();
}
#endregion
```

將 throw new ....那行程式碼註解，並寫上如下的程式：

```
return this.HireDate.CompareTo(other.HireDate);
```

4. 開啓 Form1 的程式碼視窗，在 button1\_Click 的程式區塊中寫上以下程式碼：

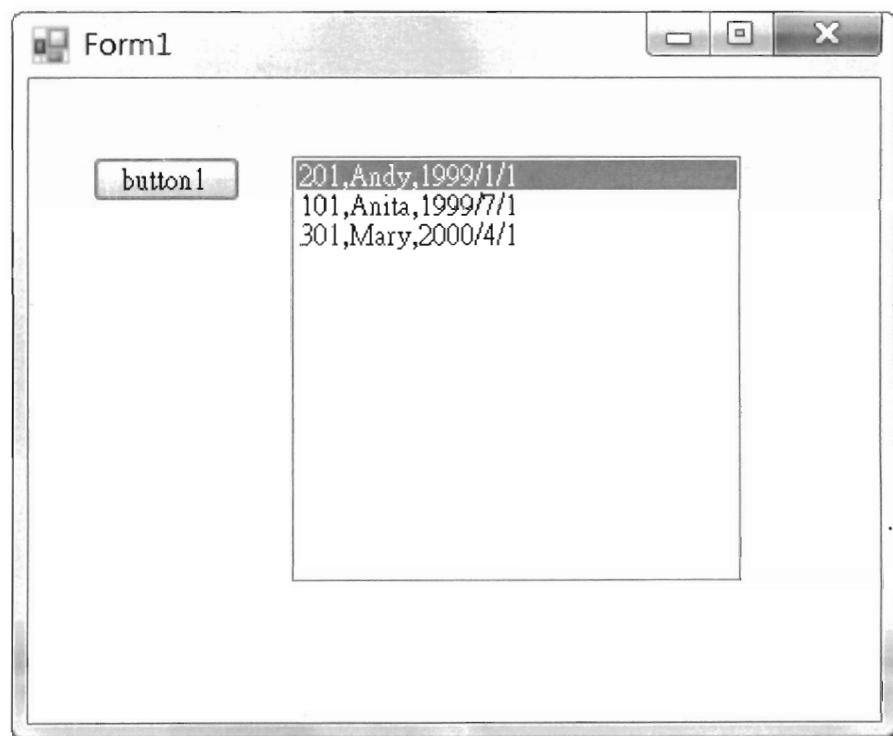
```
Visual Basic
Dim employees As New List(Of Employee)
employees.Add(New Employee(101, "Anita", #7/1/1999#))
employees.Add(New Employee(201, "Andy", #1/1/1999#))
employees.Add(New Employee(301, "Mary", #4/1/2000#))
employees.Sort()

ListBox1.DataSource = employees
ListBox1.DisplayMember = "ToString()"
```

```
C#
List<Employee> employees = new List<Employee>();
employees.Add(new Employee(101, "Anita", new DateTime(1999,
7,1)));
employees.Add(new Employee(201, "Andy", new DateTime(1999,
1,1)));
employees.Add(new Employee(301, "Mary", new DateTime (2000,
4,1)));
employees.Sort();

listBox1.DataSource = employees;
listBox1.DisplayMember = "ToString()";
```

5. 執行結果如下：





## 總結

集合通常分為兩大類：List 與 Dictionary。List 實作了 IList 介面，以索引為基礎；Dictionary 實作了 IDictionary 介面，以 key 為基礎。

Collection 代表許多物件的集合，你可以透過本章介紹的 ICollection 或 IList 等介面自行設計集合物件。設計的方式可以從無到有，建立一個類別，然後實作這些介面的方法，以設計一個具型別安全 (type-safe) 的集合。

當你把實值型別資料放到集合之中，會有 boxing 的負擔，因此為了提升效能應該儘量使用泛型集合。

泛型集合的特色是易重複使用性、具備型別安全 (type-safe)，和高效率。

## 第二章：管理與存取檔案系統

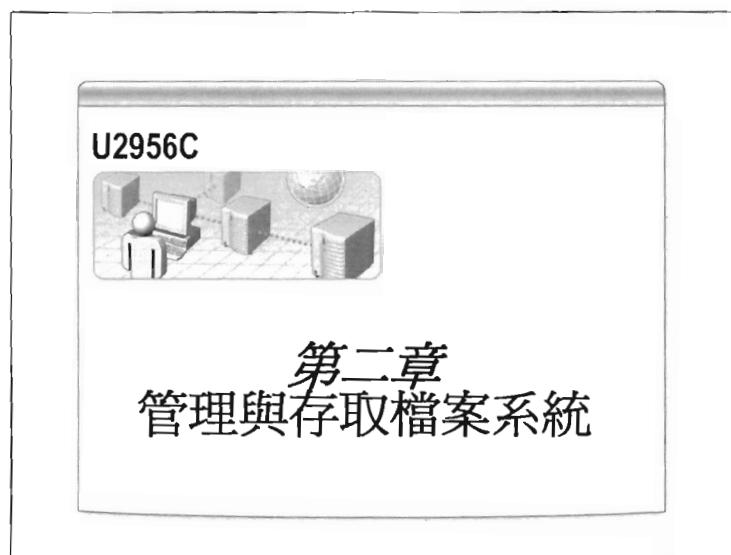
### 本章大綱

檔案系統 .....	4
存取磁碟資訊 .....	5
資料夾的管理 .....	7
檔案管理 .....	9
檔案讀寫 .....	11
Path 類別 .....	13
練習 2.1：使用 File 讀寫檔案 .....	15
監控檔案目錄 .....	17
練習 2.2：檔案系統及檔案監視 .....	18
資料流的讀寫 .....	22
認識資料流 .....	23
讀寫檔案串流 .....	25
讀寫文字到資料流 .....	27
練習 2.3：資料流的讀寫 .....	30
讀寫二進位資料 .....	34
暫存於記憶體 .....	37
為資料流加上緩衝器 .....	38
練習 2.4：使用 BufferedStream .....	39
壓縮資料概念 .....	43
練習 2.5：使用 DeflateStream 壓縮與解壓縮 .....	47
隔離儲存區的運作 .....	52
什麼是隔離儲存區 .....	53
IsolatedStorageFile 類別 .....	56
IsolatedStorageFileStream 類別 .....	58
練習 2.6：讀寫隔離儲存區的檔案 .....	59
總結 .....	63

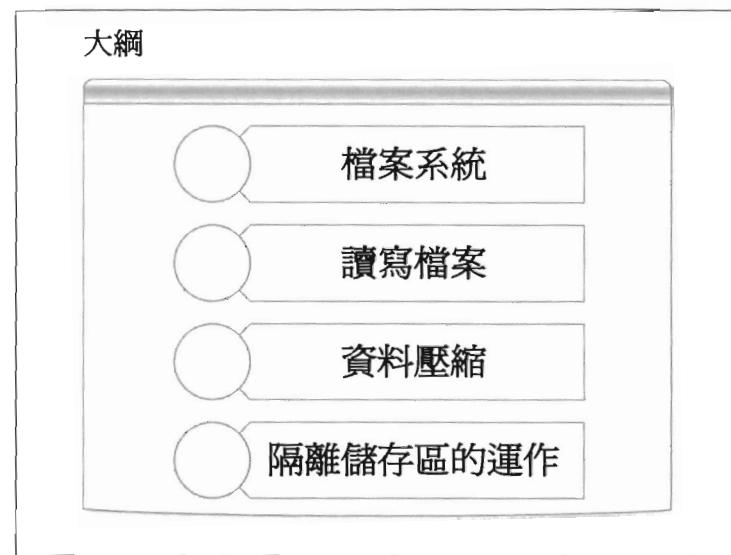
作者：

羅慧真





---



在這個章節中將介紹如何利用.NET Framework 提供的檔案系統功能管理檔案系統以及讀寫檔案。在.NET 2.0 之後的版本也多了資料流的壓縮功能，在這章中也可以看到壓縮解壓縮的功能。為了安全的因素或許系統並沒有授權給應用程式太多存取硬碟檔案的權限，不過，通常都會開放一個特定的隔離儲存區供應用程式使用，這章最後一節你也會學到相關功能。

本章介紹以下主題：

- 檔案系統
- 讀寫檔案
- 資料壓縮
- 隔離儲存區的運作

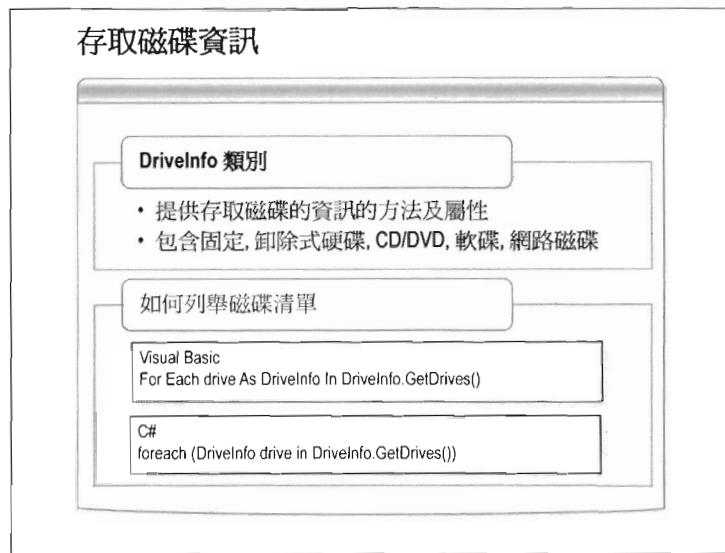


## 檔案系統

.NET Framework 所提供的檔案系統相關功能屬於命名空間 System.IO。當中提供 DriveInfo、File、FileInfo、Directory、 DirectoryInfo 及 Path 類別，讓程式人員可以以簡單的方式管理檔案系統。

除此之外，還提供一個特別的功能 FileSystemWatcher 類別可以進行檔案目錄監控異動的情況並以事件通知。

- 存取磁碟資訊
- 檔案系統的類別
- 監控檔案目錄



## 存取磁碟資訊

DriveInfo 類別提供存取磁碟的資訊。可以使用 DriveInfo 讀取磁碟類型，判斷哪些磁碟可以使用以及查詢容量剩餘空間。

以下是常用屬性列表：

- VolumeLabel，磁碟的標籤名稱。
- DriveFormat，取得磁碟格式化類型(NTFS 或 FAT...)。
- TotalSize，取得磁碟的總量。

以下是常用方法列表：

- GetDrives，取得電腦上的所有邏輯磁碟的磁碟資訊。

## 如何列舉磁碟清單

讀取磁碟清單資訊的步驟如下：

1. 呼叫 DriveInfo 的靜態方法（以 Visual Basic 而言是共享方法）GetDrives。
2. 透過迴圈程式取得 GetDrives 所回傳的 DriveInfo 物件。

如下範例：

Visual Basic

```
Dim drives As DriveInfo() = DriveInfo.GetDrives()
For Each drive As DriveInfo In drives
    listBox1.Items.Add("磁碟:" + drive.Name)
    listBox1.Items.Add(" 種類:" + drive.DriveType.ToString())
    If (drive.IsReady) Then
        listBox1.Items.Add(" 標籤名稱:" + drive.VolumeLabel)
        listBox1.Items.Add(" 檔案系統:" + drive.DriveFormat)
        listBox1.Items.Add(" 容量:" & drive.TotalSize)
    Else
        listBox1.Items.Add(" 目前無法使用.")
    End If
Next
```

C#

```
DriveInfo[] drives = DriveInfo.GetDrives();
foreach (DriveInfo drive in drives)
{
    listBox1.Items.Add("磁碟:" + drive.Name);
    listBox1.Items.Add(" 種類:" + drive.DriveType.ToString());
    if (drive.IsReady)
    {
        listBox1.Items.Add(" 標籤名稱:" + drive.VolumeLabel);
        listBox1.Items.Add(" 檔案系統:" + drive.DriveFormat);
        listBox1.Items.Add(" 容量:" + drive.TotalSize);
    }
    else {
        listBox1.Items.Add(" 目前無法使用.");
    }
}
```

## 資料夾的管理

提供大部份檔案總管中資料夾的操作功能

- 查詢子資料夾及檔案集的功能
- 建立、刪除、移動資料夾
- 查詢及修改資料夾的屬性

### Directory 類別

- 提供靜態(共享)方法
- 方法的參數指定要處理的資料夾路徑
- 適用只處理一次

### DirectoryInfo 類別

- 必須先建立物件實體
- 在建構函式的參數指定要處理的資料夾路徑
- 適用處理多次

## 資料夾的管理

.NET Framework 的命名空間 System.IO 提供了兩個管理資料夾的類別，一個是 Directory，一個是 DirectoryInfo。這兩個類別都有提供取得子資料夾清單及資料夾下的檔案集，還可以建立資料夾、刪除、複製...等功能。

### Directory 類別

Directory 類別提供靜態方法(for C#, Visual Basic 是共享方法)，不需建立物件實體直接呼叫靜態(共享)方法並傳入要動作的資料夾路徑即可。它有以下常用方法：

- Delete，刪除指定的資料夾。
- Exists，判斷指定路徑是否參考磁碟上的現有目錄。
- GetLogicalDrives，取得電腦上的所有邏輯磁碟名稱，以 String 型別回傳 "<drive letter>:\\" 的格式。

```
VisualBasic
Directory.Delete("C:\MyFolder")
```

```
C#
```

```
Directory.Delete(@"C:\MyFolder");
```

## DirectoryInfo 類別

提供與 Directory 類別類似的功能，不過這個類別必須先建立執行個體 (Instance) 方法，並於建構函式傳入要處理的資料夾路徑。以下是常用屬性列表：

- Parent，取得指定子資料夾的父層 (Parent) 目錄。
- Root，取得路徑的根資料夾 (Root) 部分。

以下是常用方法列表：

- Create，建立資料夾。
- GetDirectories，傳回目前資料夾的子資料夾。
- GetFiles，從目前的資料夾傳回檔案清單。

### Visual Basic

```
Dim path As String = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
If Not Directory.Exists(path) Then Return

Dim root As New DirectoryInfo(path)
For Each di As DirectoryInfo In root.GetDirectories()
    ListBox1.Items.Add(
        String.Format("{0}, Attributes:{1}.", di.Name, di.Attributes))
Next
```

### C#

```
String path = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
if (! Directory.Exists(path)) return;

DirectoryInfo root = new DirectoryInfo(path);
foreach (DirectoryInfo di in root.GetDirectories())
{
    ListBox1.Items.Add(
        String.Format("{0}, Attributes:{1}.", di.Name, di.Attributes));
}
```



## 檔案管理

.NET Framework 的命名空間 System.IO 提供了兩個管理檔案的類別，一個是 File，一個是 FileInfo。這兩個類別都有提供取得建立、刪除、複製檔案...等功能。

### File 類別

File 類別提供靜態方法(for C#, Visual Basic 是共享方法)，不需建立物件實體直接呼叫靜態(共享)方法並傳入要動作的資料夾路徑即可。可以複製檔案、刪除檔案、開啟檔案。常用方法如下：

- Copy，複製檔案。
- Delete，刪除檔案。
- Encrypt，檔案加密。

Visual Basic  
File.Copy(TextBox1.Text, TextBox2.Text)

C#  
File.Copy(TextBox1.Text, TextBox2.Text);

## FileInfo 類別

提供與 FileInfo 類別類似的功能，不過這個類別必須先建立執行個體 (Instance) 方法，並於建構函式傳入要處理的檔名路徑。以下是常用方法列表：

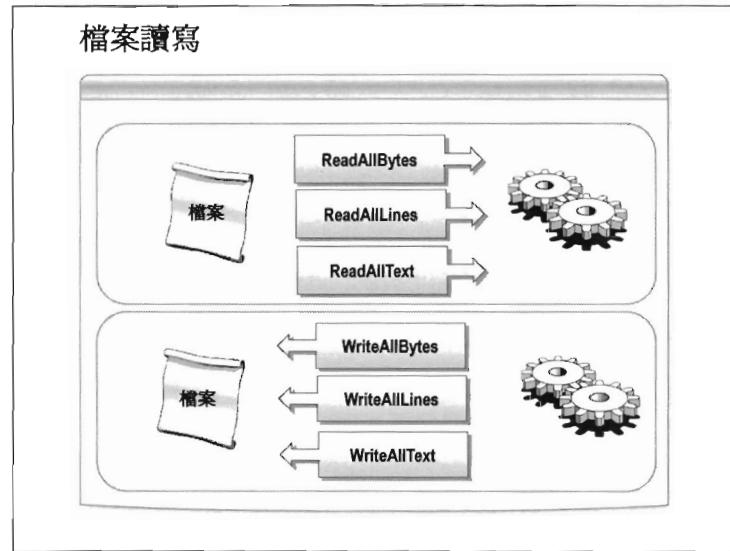
- CopyTo，複製現有的檔案到新的檔案。
- MoveTo，搬移檔案。
- Decrypt，檔案解密。

### Visual Basic

```
Dim fi As New FileInfo(TextBox1.Text)
If fi.Exists Then
    Dim oldName As String = fi.FullName
    fi.MoveTo(TextBox2.Text)
    MessageBox.Show(oldName + "->" + fi.FullName)
End If
```

### C#

```
FileInfo fi = new FileInfo(TextBox1.Text);
{
    string oldName = fi.FullName;
    fi.MoveTo(TextBox2.Text);
    MessageBox.Show(oldName + "->" + fi.FullName);
}
```



## 檔案讀寫

File 類別提供一些讀寫檔案內容的方法。以下範例是讀寫檔案的方法：

- ReadAllBytes，開啓並讀取 Binary 檔，回傳 Byte 陣列，然後關閉。
- WriteAllBytes，將 Byte 陣列寫入指定的檔名然後關閉。檔案若不存在會建立新檔案，檔案若存在會蓋掉同名檔案。
- ReadAllLines，讀取文字檔以行為單位回傳 String 陣列，然後關閉。
- WriteAllLines，將 String 陣列寫入指定的檔案，然後關閉。
- ReadAllText，讀取文字檔以回傳 String 型別，然後關閉。
- WriteAllText，將 String 寫入指定的檔案，然後關閉。

這個範例是將文字方塊的內容轉成 Byte 陣列，然後利用 WriteAllBytes 寫入檔案。

```
Visual Basic
Dim data() As Byte = Encoding.UTF8.GetBytes(textBox1.Text)
File.WriteAllBytes(txtFileName.Text, data)
```

System.Text

```
C#
byte[] data = Encoding.UTF8.GetBytes(textBox1.Text);
File.WriteAllBytes(txtFileName.Text, data);
```

這個範例是將使用 ReadAllBytes 讀取檔案，並利用 Encoding 將傳回的 Byte 陣列轉成字串。

```
Visual Basic
Dim data() As Byte
data = File.ReadAllBytes(txtFileName.Text)
textBox1.Text = Encoding.UTF8.GetString(data)
```

```
C#
byte[] data = null;
data = File.ReadAllBytes(txtFileName.Text);
textBox1.Text = Encoding.UTF8.GetString(data);
```

這個範例是使用 WriteAllText 讀取檔案內容：

```
Visual Basic
File.WriteAllText(txtFileName.Text, textBox1.Text)
```

```
C#
File.WriteAllText(txtFileName.Text, textBox1.Text);
```

這個範例是使用 ReadAllText 讀取檔案內容：

```
Visual Basic
textBox1.Text = File.ReadAllText(txtFileName.Text)
```

```
C#
textBox1.Text= File.ReadAllText(txtFileName.Text);
```



## Path 類別

用來做為路徑、檔名、副檔名的字串處理功能，像是將路徑與檔名結合的 Combine 方法，以及取出副檔名的 GetExtension 等等。

以下是常用方法列表：

- GetRandomFileName，取得隨機檔名。
- GetExtension，傳回指定檔名中副檔名的部份。
- ChangeExtension，變更檔名中副檔名的部份。

## 如何利用 Path 修改副檔名

可以利用 Path 提供的各種方法處理檔名的字串，但要注意它只處理字串，並不會真正修改實際在硬碟中的檔名哦。範例如下所示：

```
Visual Basic
Dim oldName As String = "c:\u2956\mydoc.txt"
ListBox1.Items.Add("原始名稱:" & oldName)
ListBox1.Items.Add("副檔名:" & Path.GetExtension(oldName))
ListBox1.Items.Add("修改檔名:" & Path.ChangeExtension( _
oldName, "bak"))
```

C#

```
String oldName = @"c:\u2956\mydoc.txt";
ListBox1.Items.Add("原始名稱:" + oldName);
ListBox1.Items.Add("副檔名:" + Path.GetExtension(oldName));
ListBox1.Items.Add("修改檔名:" +
    Path.ChangeExtension(oldName, "bak"));
```

如果要真實的修改硬碟中的檔名，可以使用 FileInfo 的 MoveTo。

### 練習2.1：使用File讀寫檔案

• 瞭解如何使用File的靜態方法讀寫多行文字到檔案。

• 預估實作時間：5分鐘

### 練習 2.1：使用 File 讀寫檔案

#### 目的：

學會使用 File 類別所提供的讀寫多行文字的方法。

#### 功能描述：

在這個練習中會用到 File 提供的靜態(共享)方法 ReadAllLines 及 WriteAllLines 將多行的文字讀及寫到檔案。

#### 預估實作時間：5分鐘

#### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod02\_1。
3. 在 Form 放上兩個 TextBox 及兩個 Button。第二個 TextBox 設定 MultiLine 為 True。一個 Button 的 Text 設為 WriteAllLines，另一個的 Text 設為 ReadAllLines。

4. 在 WriteAllLines 按鈕的 Click 事件實作寫入 TextBox2 的內容到 TextBox1 的檔名。

Visual Basic

```
Dim data() As String = textBox1.Lines
File.WriteAllLines(txtFileName.Text, data)
```

C#

```
string[] data = textBox1.Lines;
File.WriteAllLines(txtFileName.Text, data);
```

5. 在 ReadAllLines 按鈕的 Click 事件實作讀取 TextBox1 指定的檔名，並將內容顯示到 TextBox2

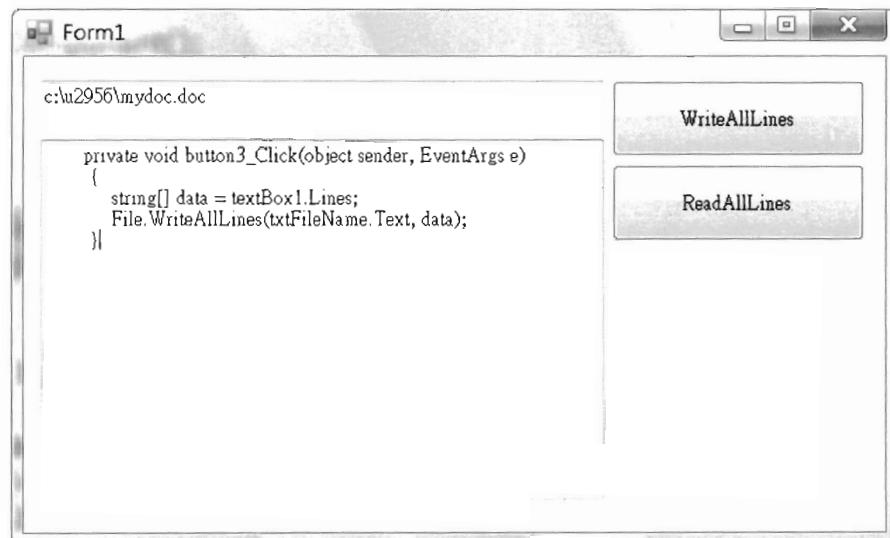
Visual Basic

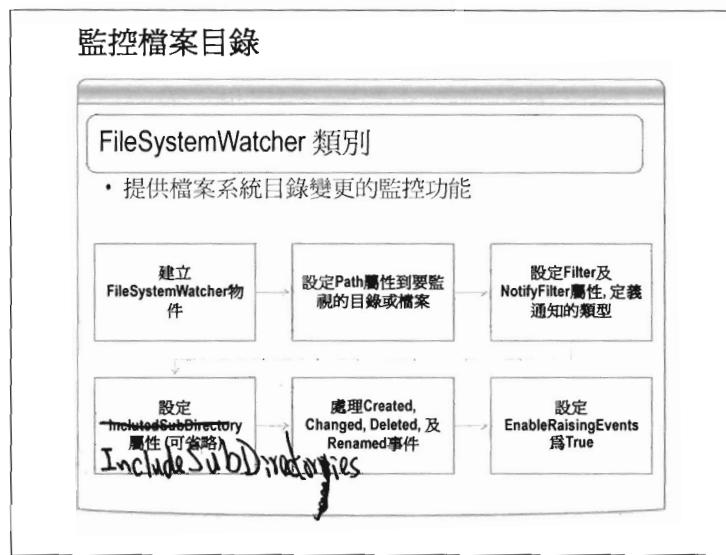
```
Dim data() As String
data = File.ReadAllLines(txtFileName.Text)
textBox1.Lines = data
```

C#

```
string[] data = null;
data = File.ReadAllLines(txtFileName.Text);
textBox1.Lines = data;
```

6. 執行並測試。





## 監控檔案目錄

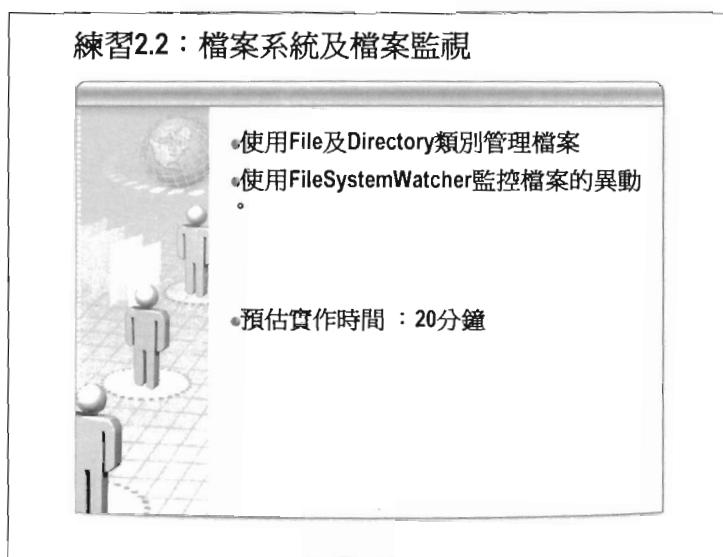
`FileSystemWatcher` 類別提供檔案系統目錄是否變更的監控功能。可以用來監控某個特定的檔案或是目錄是否新增、刪除，或者更新，並在 `EnableRaisingEvents` 屬性設為 `True` 啓用事件通知，在指定事件程序後便會在建立、修改檔案時觸發事件。

以下是常用屬性列表：

- `Filter`，監控事件使用的過濾條件。
- `NotifyFilter`，取得或設定要監看的變更類型。
- `EnableRaisingEvents`，預設為 `False`，設為 `True` 才能啓用事件的觸發。

以下是事件列表：

- `Created`，監控的 `Path` 之內，建立檔案或資料夾觸發的事件。
- `Renamed`，監控的 `Path` 之內，重新命名的檔案或資料夾觸發的事件。
- `Changed`，監控的 `Path` 之內，變更的檔案內容時觸發的事件。
- `Deleted`，監控的 `Path` 之內，刪除檔案或目錄時觸發的事件。



## 練習 2.2：檔案系統及檔案監視

### 目的：

學會使用 File 類別及 Directory 類別變更檔案，以及使用 FileSystemWatcher 監控檔案的異動。

### 功能描述：

在這個練習中會用到 File 類別建立、刪除、搬移檔案，Directory 類別建立及刪除子目錄。還有 FileSystemWatcher 監控檔案的異動狀況並以事件通知。

### 預估實作時間：20 分鐘

### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod02\_2。
3. 在 Form1 放上三個 Button。

4. 第一個 Button 的 Text 屬性指定為「File.AppendAllText」，並在 Click 事件使用 File.AppendAllText，建立或在既有檔案加入文字。

Visual Basic

```
File.AppendAllText("c:\u2956\mytext.txt", "new line")
```

C#

```
File.AppendAllText("c:\u2956\mytext.txt", "new line");
```

5. 第二個 Button 的 Text 屬性指定為「File.Move」，並在 Click 事件使用 File 類別檢查檔案是否存在，然後將檔案移到目標位置。

Visual Basic

```
Dim sourceFileName As String = "c:\u2956\mytext.txt"
Dim destFileName As String = "c:\u2956\mydoc.txt"
If Not File.Exists(sourceFileName) Then
    MessageBox.Show(sourceFileName + "不存在")
    Return
End If
If File.Exists(destFileName) Then
    File.Delete(destFileName)
End If

File.Move(sourceFileName, destFileName)
```

C#

```
string sourceFileName = @"c:\u2956\mytext.txt";
string destFileName = @"c:\u2956\mydoc.txt";
if (!File.Exists (sourceFileName))
{
    MessageBox.Show (sourceFileName + "不存在");
    return;
}
if (File.Exists(destFileName))
    File.Delete(destFileName);

File.Move(sourceFileName, destFileName);
```

6. 第三個 Button 的 Text 屬性指定為「Directory.CreateDirectory」，並在 Click 事件使用 Directory 類別檢查子目錄是否存在，並建立子目錄。

Visual Basic

```
Dim newFolder As String = "c:\u2956\subdir"
If Directory.Exists(newFolder) Then
```

```

        Directory.Delete(newFolder)
End If
Directory.CreateDirectory(newFolder)

```

```

C#
string newFolder = @"c:\u2956\subdir";
if (Directory.Exists(newFolder))
    Directory.Delete(newFolder);
Directory.CreateDirectory(newFolder);

```

7. 測試以上三個功能是否能正常運作。

8. 在 Form\_Load 事件加上檔案監控功能，使用  
FileSystemWatcher

```

Visual Basic
Dim watcher As New FileSystemWatcher()
watcher.Path = "c:\u2956"
watcher.IncludeSubdirectories = True
watcher.Filter = "*.*"
watcher.NotifyFilter = NotifyFilters.LastAccess Or _
    NotifyFilters.LastWrite Or NotifyFilters.FileName Or _
    NotifyFilters.Size Or NotifyFilters.DirectoryName
AddHandler watcher.Created, New FileSystemEventHandler(AddressOf watcher_Changed)
AddHandler watcher.Changed, New FileSystemEventHandler(AddressOf watcher_Changed)
AddHandler watcher.Renamed, New RenamedEventHandler(AddressOf watcher_Renamed)
watcher.EnableRaisingEvents = True

```

```

C#
FileSystemWatcher watcher = new FileSystemWatcher();
watcher.Path = @"c:\u2956";
watcher.IncludeSubdirectories = true;
watcher.Filter = "*.*";
watcher.NotifyFilter = NotifyFilters.LastAccess | NotifyFilters.LastWrite |
    NotifyFilters.FileName | NotifyFilters.Size | NotifyFilters.DirectoryName ;
watcher.Created += new FileSystemEventHandler(watcher_Changed);
watcher.Changed += new FileSystemEventHandler(watcher_Changed);
watcher.Renamed += new RenamedEventHandler(watcher_Renamed);
watcher.EnableRaisingEvents = true;

```

9. 加上 watcher\_Changed 事件，並在接收到檔案變更時印出訊息。

```
Visual Basic
Sub watcher_Changed(ByVal sender As Object, ByVal e As FileSystemEventArgs)
    Console.WriteLine("{0}:{1}", e.ChangeType, e.FullPath)
End Sub
```

```
C#
void watcher_Changed(object sender, FileSystemEventArgs e)
{
    string msg=e.ChangeType + ":" + e.FullPath;
    Console.WriteLine(msg);
}
```

10. 加上 watcher\_Renamed 事件，並在接收到檔案變更時印出訊息。

```
Visual Basic
Sub watcher_Renamed(ByVal sender As Object, ByVal e As RenamedEventArgs)
    Console.WriteLine("{0}: {1}-{2}", e.ChangeType, e.OldFullPath, e.FullPath)
End Sub
```

```
C#
void watcher_Renamed(object sender, RenamedEventArgs e)
{
    string msg = e.ChangeType + ":" + e.OldFullPath + "->" + e.FullPath;
    Console.WriteLine(msg);
}
```

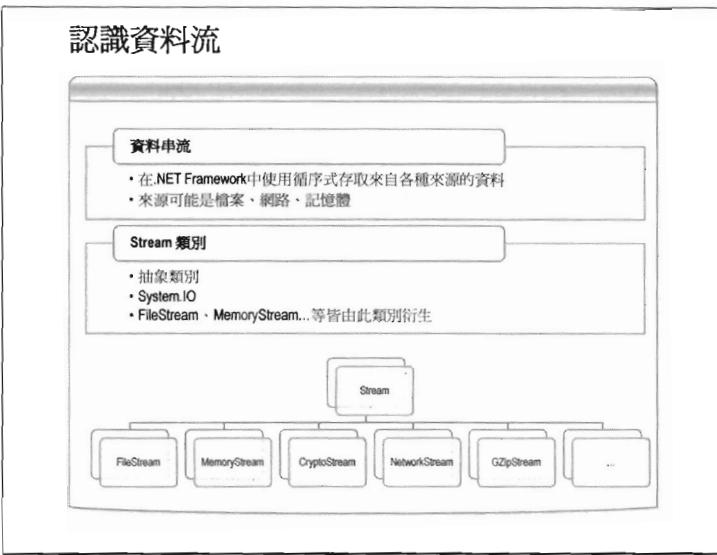
11. F5 執行，按三個 Button 試試看 Output Windows 輸出什麼訊息。



## 資料流的讀寫

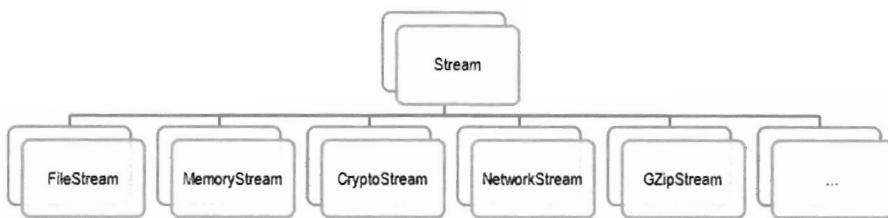
在這一節我們將介紹什麼是 Stream(資料流)，並且學習如何利用 FileStream 讀寫檔案，以及如何處理文字、Binary 格式的資料。另外，存取資料流時不一定要將資料存檔，也可以使用 Memory 做為暫存空間。為了處理資料流更有效率，例如 NetworkStream，可以為它加上緩衝空間以提昇存取時的效率。

- 認識資料流
- 檔案串流的讀寫
- 讀寫不同的格式
- 文字資料
- 二進位資料
- 特殊處理
- 資料流暫存於記憶體
- 為資料流加上緩衝器



## 認識資料流

資料流指的是在.NET Framework 中使用循序式或隨機式存取來自各種來源的資料，這些來源可能是檔案、網路、記憶體等。Stream 類別是資料串流處理的抽象類別，不管是檔案、網路、記憶體等來源的資料串流都是繼承自 Stream 類別，具有 Stream 類別的基本功能。.NET Framework 中所有與資料串流有關的功能像是 FileStream(System.IO)、MemoryStream(System.IO)、CryptoStream(System.Security)、NetworkStream(System.Net)、GzipStream(System.Compression)皆是繼承自 Stream 類別，如下圖。

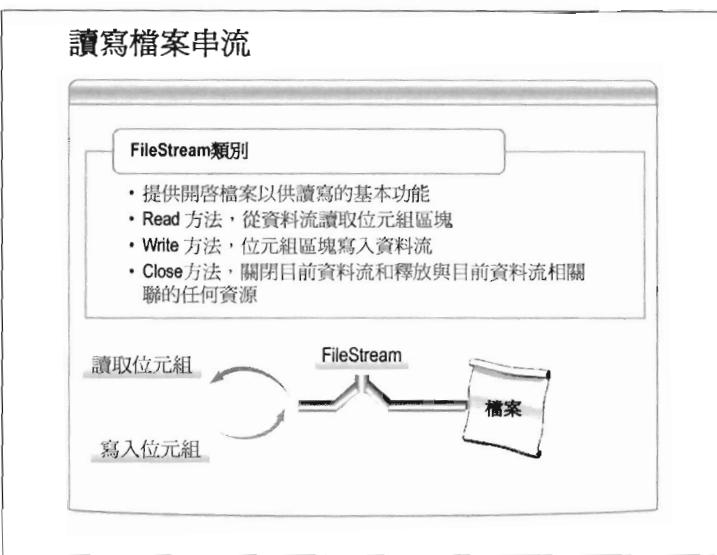


它的常用屬性列表如下：

- CanRead，CanWrite，查詢資料流是否支援讀取或寫入
- Length，資料流的長度 (以 Byte 為單位)。
- Position，資料流目前的位置。

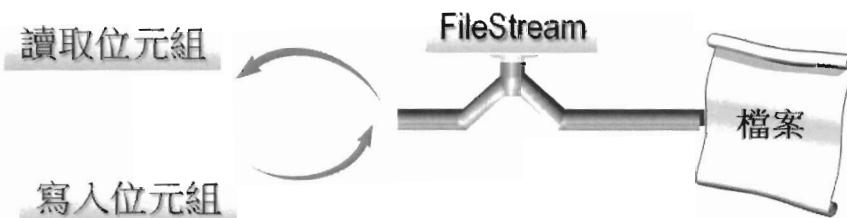
它的常用方法列表如下：

- Read，讀取下一個 byte 到指定的 byte 陣列。
- Write，將 byte 陣列寫入到資料流。
- Flush，清出所有緩衝區的資料，並寫到資料流對應的來源，  
並造成任何緩衝資料都寫入基礎裝置。
- Close，關閉資料流，同時將清出所有緩衝區的資料。



## 讀寫檔案串流

FileStream 類別提供開啓檔案以供讀寫的基本功能。可以用來讀寫檔案內容，讀寫時是以位元組(bytes)為單位讀取，如需處理為文字，必須使用 Encoding 類別將位元組轉為文字，或是使用 StreamReader、StreamWriter 類別處理。



以下是常用屬性列表：

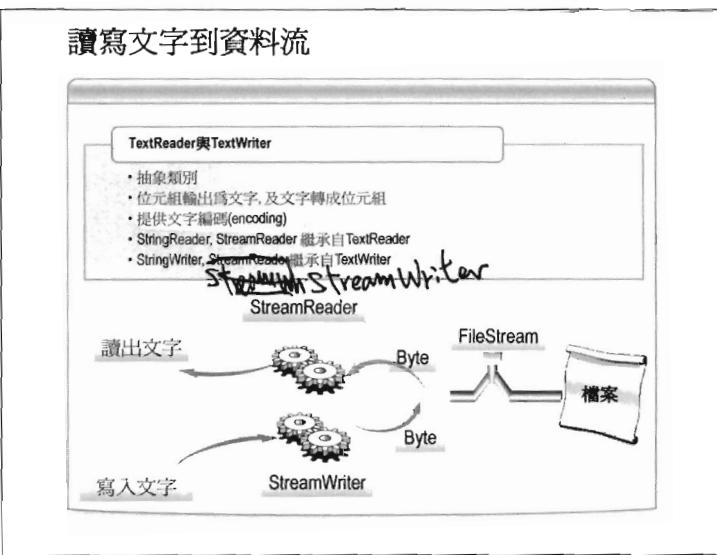
- Length，取得資料流的位元組長度。

### FileMode 列舉型別

指定作業系統應該如何開啓檔案。成員列表，Append、Create、CreateNew、Open、OpenOrCreate、Truncate。

## FileAccess 列舉型別

定義檔案的讀取、寫入，或讀/寫存取權限的常數。這個列舉型別的 FlagsAttribute 屬性允許將其成員值以位元組合的方式來使用。以下是成員列表，Read、ReadWrite、Write。



## 讀寫文字到資料流

Stream 類別所延伸的 FileStream 等類別讀寫內容都是 Byte 陣列為單位。若要將這些資料流處理成文字的話，必須使用 System.Text 命名空間下的 Encoding 類別進行編碼。.NET Framework 提供了 StreamReader、TextWriter 抽象類別，提供文字處理讀寫的能力。

StringReader 以及 StreamReader，就是繼承自 TextReader，以下是 TextReader 常用方法列表：

- ReadLine，讀取一行的字串。
- ReadToEnd，讀取字串從目前位置到最後一個字元。
- Peek，取得下一個字元，但並不會移動位置。
- Close，關閉及清出所有緩衝區資料。

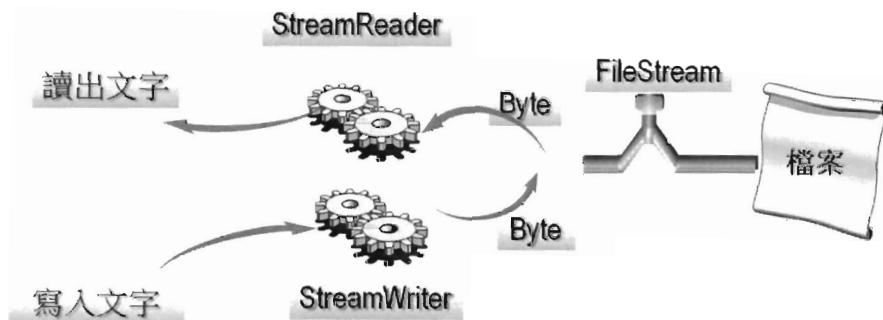
StringWriter 與 StreamWriter 是繼承自 StreamWriter。以下是常用成員列表：

- AutoFlush 屬性，設為 True 時會在呼叫寫入方法時自動清出緩衝區。
- WriteLine 方法，在寫入的字串之後後加上換行符號。

## StreamReader , StreamWriter 類別

StreamReader 類別會將 Stream 所讀取的 Byte 資料透過指定的 Encoding (預設是 UTF8) 轉成字串輸出。

StreamWriter 類別則是字串透過指定的 Encoding (預設是 UTF8) 轉成 Byte 寫到 Stream。



## StringReader , StringWriter 類別

提供讀寫字串內容的功能。

以下範例是利用 StreamReader 逐行讀取 TextBox 的內容，並顯示在 ListBox 控制項上。

### Visual Basic

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim sr As New StringReader(TextBox1.Text)
    Do While sr.Peek() <> -1
        Dim aline As String = sr.ReadLine
        ProccessText(aline)
    Loop
    sr.Close()
End Sub

Sub ProccessText(ByVal aLine As String)
    ListBox1.Items.Add(aLine)
End Sub

```

### C#

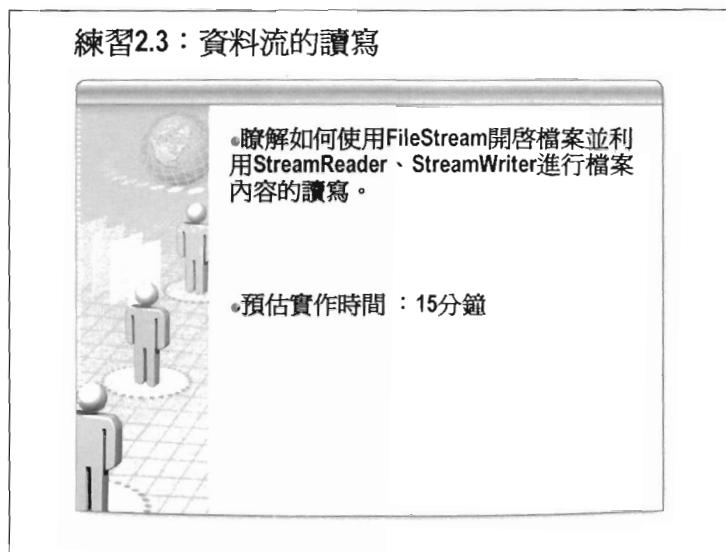
```

private void Button1_Click(object sender, EventArgs e)
{
    StringReader sr = new StringReader(TextBox1.Text);
    while (sr.Peek() != -1)
    {

```

```
String aLine = sr.ReadLine();
ProcessText(aLine);
}
sr.Close();
}
void ProcessText(String aLine )
{
    ListBox1.Items.Add(aLine);
}
```

Dad  
CodeGen



## 練習 2.3：資料流的讀寫

### 目的：

瞭解如何使用 FileStream 開啓檔案並利用 StreamReader、  
StreamWriter 進行檔案內容的讀寫。

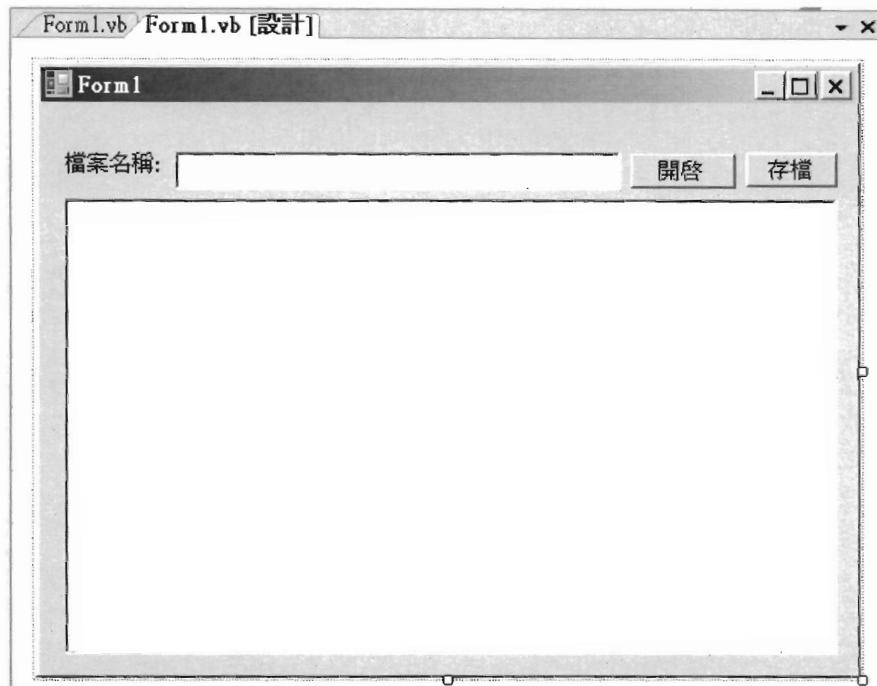
### 功能描述：

在這個練習中在一個 Windows 的介面下進行檔案的開啓，並且讀取檔案到畫面上，以及將更新的資料寫回檔案。

### 預估實作時間：15分鐘

### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod02\_3。
3. 從「Solution Explorer」開啓 Form1，設計畫面如下圖。



4. 進入程式碼檢視視窗，匯入必要的命名空間：

```
Visual Basic
Imports System.IO
Imports System.Text
```

```
C#
using System.IO;
```

5. 進入 Button1\_Click 事件程序。撰寫開啓檔案（使用者要開啓的檔名放在 TextBox1.Text），並顯示檔案內容於 TextBox2。

```
Visual Basic
Dim fileName As String = TextBox1.Text
If Not File.Exists(fileName) Then
    MessageBox.Show("檔案不存")
    Exit Sub
End If
Using fs As New FileStream(fileName, FileMode.Open,
    FileAccess.Read)
    Dim sr As New StreamReader(fs, Encoding.GetEncoding("big5"))
    TextBox2.Text = sr.ReadToEnd()
    sr.Close()
End Using
```

```
C#
```

```

String fileName = TextBox1.Text;
if (!File.Exists(fileName))
{
    MessageBox.Show("檔案不存");
    return;
}

using (FileStream fs = new FileStream(fileName, FileMode.Open,
    FileAccess.Read))
{
    StreamReader sr = new StreamReader(fs,
        Encoding.GetEncoding("big5"));
    TextBox2.Text = sr.ReadToEnd();
    sr.Close();
}

```

6. 進入 Button2\_Click 事件程序。撰寫修改的內容寫回檔案。

Visual Basic

```

Dim fileName As String = TextBox1.Text
Using fs As New FileStream(fileName, FileMode.Create, FileAccess.
    Write)
    Dim sw As New StreamWriter(fs, _
        Encoding.GetEncoding("big5"))
    sw.WriteLine(TextBox2.Text)
    sw.Close()
End Using

```

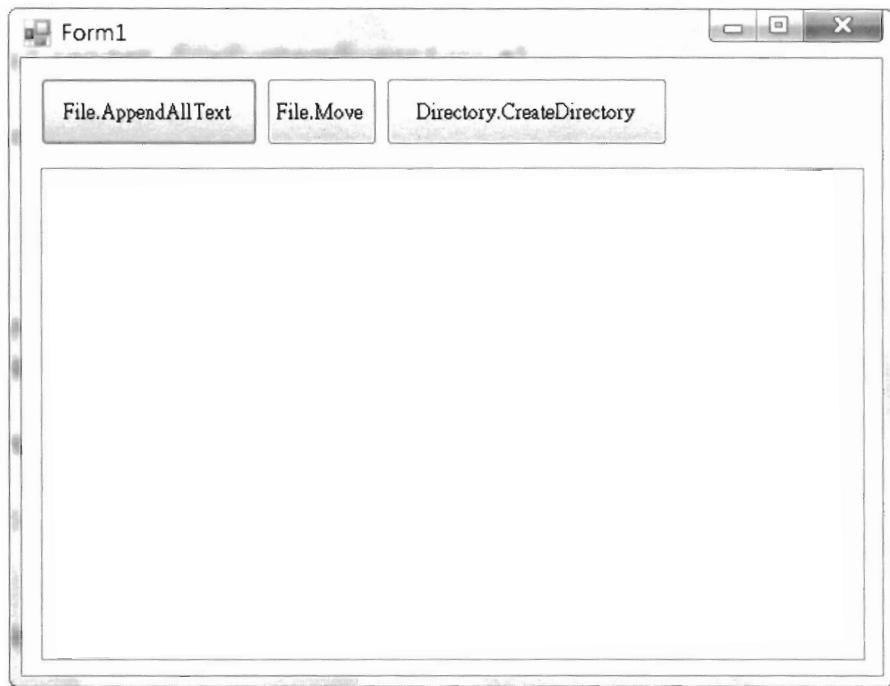
C#

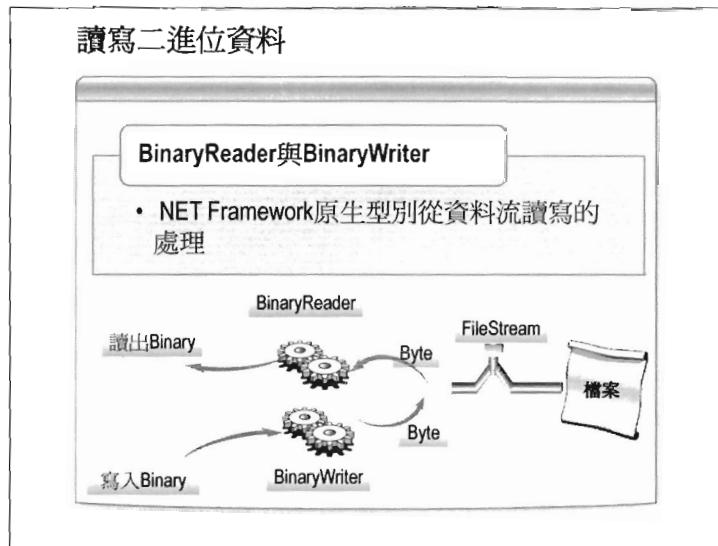
```

String fileName = TextBox1.Text ;
using (FileStream fs =
    new FileStream(fileName, FileMode.Create, FileAccess.Write))
{
    StreamWriter sw =
        new StreamWriter(fs, Encoding.GetEncoding("big5"));
    sw.WriteLine(TextBox2.Text);
    sw.Close();
}

```

7. 執行並測試專案。

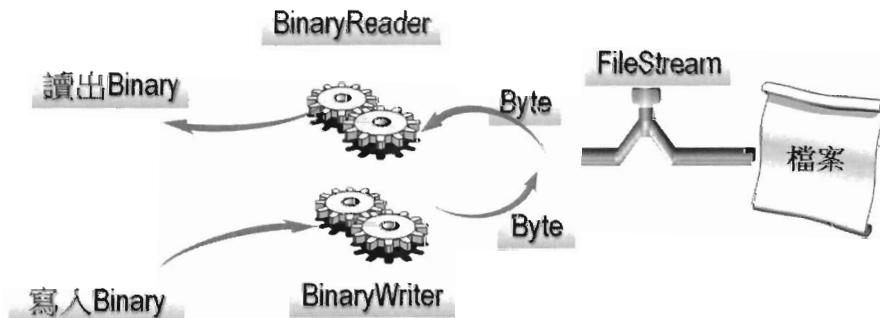




## 讀寫二進位資料

BinaryReader 與 StreamReader 有點類似，都是可以讀取 Stream 類別的來源，差別是 BinaryReader 是將讀取的 Byte 資料轉成.NET Framework 的原生型別。

BinaryWriter 則是將.NET Framework 的原生型別轉成 Byte 資料寫到 Stream。



以下範例是使用 BinaryWriter 將 Student 的資料寫到檔案：

```

Visual Basic
Dim students(1) As Student
students(0) = New Student() With {.ID = 1, .Name = "Lisa", .Avg
Score = 88}
students(1) = New Student() With {.ID = 2, .Name = "Mickey", .A

```

```

    vgScore = 90}
Using fs As FileStream = File.Create("c:\u2956\students.bin")
    Dim writer As New BinaryWriter(fs)
    For Each st In students
        writer.Write(st.ID)
        writer.Write(st.Name)
        writer.Write(st.AvgScore)
    Next
End Using

```

```

C#
Student[] students = new Student[]
    {new Student{ ID = 1, Name = "Lisa", AvgScore = 88 },
     new Student { ID = 2, Name = "Mickey", AvgScore = 90 }};
using (FileStream fs = File.Create (@"c:\u2956\students.bin"))
{
    BinaryWriter writer = new BinaryWriter(fs);
    foreach (Student st in students)
    {
        writer.Write(st.ID);
        writer.Write(st.Name);
        writer.Write(st.AvgScore);
    }
}

```

以下範例是從檔案讀取資料，並填回 Student 的欄位：

```

Visual Basic
Using fs As FileStream = File.Open("c:\u2956\students.bin", FileMode.Open)
    Dim reader As New BinaryReader(fs)
    Do While reader.PeekChar() <> -1
        Dim st As New Student With _
            {.ID = reader.ReadInt32(), _
             .Name = reader.ReadString(), _
             .AvgScore = reader.ReadSingle()}
        listBox1.Items.Add(st.ID & ", " & st.Name & ", " & st.AvgScore)
    Loop
End Using

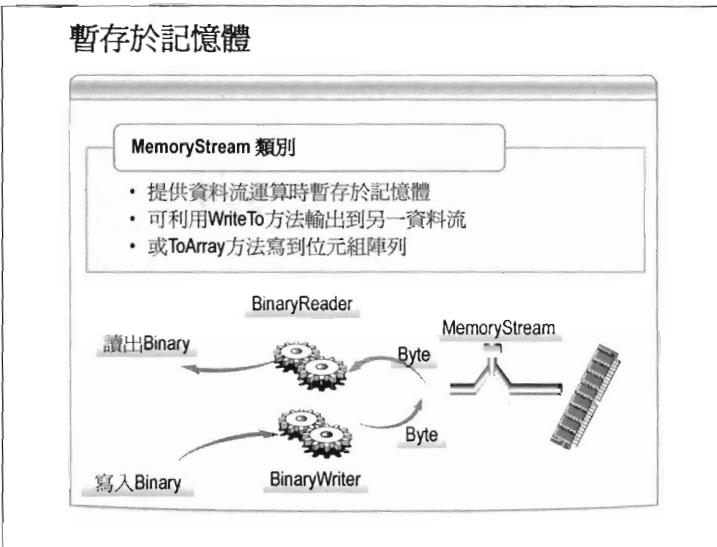
```

```

C#
Student[] students = new Student[]
    {new Student{ ID = 1, Name = "Lisa", AvgScore = 88 },
     new Student { ID = 2, Name = "Mickey", AvgScore = 90 }};
using (FileStream fs = File.Create (@"c:\u2956\students.bin"))
{
    BinaryWriter writer = new BinaryWriter(fs);
    foreach (Student st in students)
    {
}

```

```
        writer.Write(st.ID);
        writer.Write(st.Name);
        writer.Write(st.AvgScore);
    }
}
```

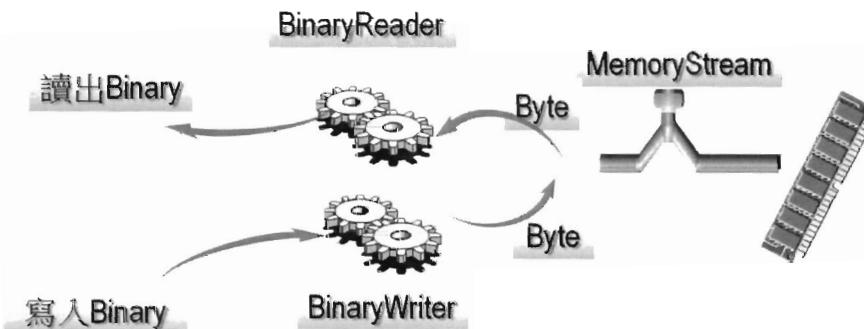


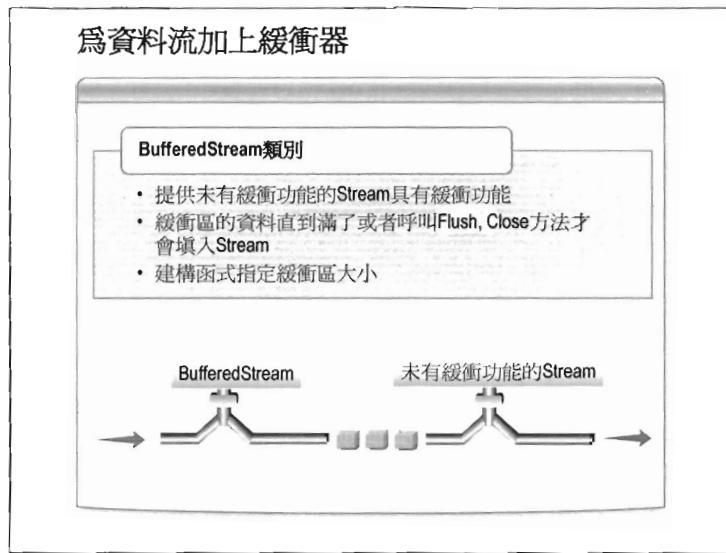
## 暫存於記憶體

當程式在進行某些特別的運算時，必須有暫存性的資料流，除了使用暫存檔之外，另一個更有效率的選擇就是使用 `MemoryStream`，將資料流暫存於記憶體，減少實體 IO 的運作，以增進運算效能。

以下是常用方法列表：

- `WriteTo`，可將 `MemoryStream` 直接輸出到另一個 Stream
- `ToArray`，將整個資料流的內容寫到 Byte 的陣列中。





## 為資料流加上緩衝器

大部份的 Stream 像是 FileStream 內部都有實作緩衝器的功能，所以當寫入資料到 FileStream 時並不會馬上輸出到檔案，必須等到緩衝器溢出時或者明確的呼叫 Flush 或 Close 方法才會將緩衝區資料清出到檔案。

並不是全部的 Stream 都有實作緩衝器的功能，像是 NetworkStream 就沒有緩衝區，若要利用緩衝器提昇存取效能，那就得用到 BufferedStream 類別。



```
Visual Basic
Dim buffer As New BufferedStream(netStream, 8192)
buffer.Write(data)
```

```
C#
BufferedStream buffer = new BufferedStream(newStream, 8192)
buffer.Writer(data);
```

### 練習2.4：使用BufferedStream

•瞭解如何使用BufferedStream增進讀取資料的效能。

•預估實作時間：20分鐘

### 練習 2.4：使用 BufferedStream

#### 目的：

學會使用 BufferedStream 類別增加讀取資料的效能。

#### 功能描述：

在這個練習中會用到 WebRequest 及WebResponse 類別存取網路的檔案做到上傳及下載的功能。為了增進存取的效能，在處理資料流時使用 BufferedStream 類別。

#### 預估實作時間：20 分鐘

#### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod02\_4。

3. 在 Form1 放入一個 TextBox，取名為 txtURL，放入兩個 Button，及另一個 TextBox，並設定此 TextBox 的 MultiLine 為 True。
4. 第一個 Button 的 Text 設定為「Upload」，第二個 Button 的 Text 設定為「Download」。
5. 進入程式碼檢視，先匯入必要的命名空間：

```
Visual Basic
Imports System.Net
Imports System.IO
Imports System.Text
```

```
C#
using System.IO;
using System.Net;
```

6. 在 Upload 按鈕的 Click 事件實作檔案上傳的程式：

```
Visual Basic
Dim content() As Byte
content = Encoding.UTF8.GetBytes(TextBox1.Text)
Dim request As WebRequest = WebRequest.Create(txtURL.Text)
request.Method = "POST"
Dim webStream As Stream = request.GetRequestStream()
Dim buffer As New BufferedStream(webStream, 1024)
buffer.Write(content, 0, content.Length)
buffer.Close()
Try
    request.GetResponse()
    MessageBox.Show("完成")
Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try
```

```
C#
Byte[] content= Encoding.UTF8.GetBytes(TextBox1.Text);
WebRequest request = WebRequest.Create(txtURL.Text);
request.Method = "POST";
Stream webStream = request.GetRequestStream();
BufferedStream buffer=new BufferedStream(webStream, 1024);
buffer.Write(content, 0, content.Length);
buffer.Close();
try
{
    request.GetResponse();
```

```

        MessageBox.Show("完成");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

7. 在 Download 按鈕的 Click 事件實作下載檔案的程式：

#### Visual Basic

```

Dim request As WebRequest = WebRequest.Create(txtURL.Text)
request.Method = "POST"
Dim response As WebResponse = request.GetResponse()
Dim webStream As Stream = response.GetResponseStream()
Dim buffer As New BufferedStream(webStream, 1024)
Dim content(1023) As Byte
Dim length As Integer
Dim sb As New StringBuilder
Do
    length = buffer.Read(content, 0, 1024)
    sb.Append(Encoding.UTF8.GetString(content).Trim())
    Array.Clear(content, 0, 1024)
Loop While length = 1024
buffer.Close()
TextBox1.Text = sb.ToString()

```

#### C#

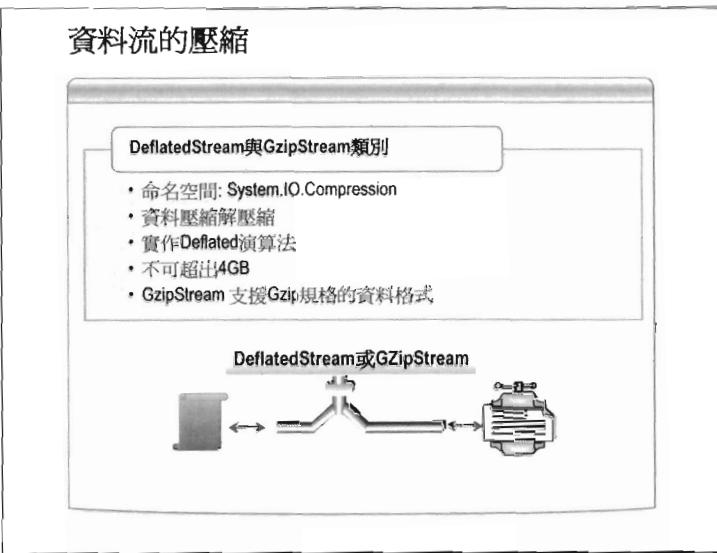
```

WebRequest request = WebRequest.Create(txtURL.Text);
request.Method = "POST";
WebResponse response = request.GetResponse();
Stream webStream = response.GetResponseStream();
BufferedStream buffer=new BufferedStream(webStream, 1024);
Byte[] content=new byte[1024];
int length =0;
StringBuilder sb =new StringBuilder(1024);
do
{
    length = buffer.Read(content, 0, 1024);
    sb.Append(Encoding.UTF8.GetString(content).Trim());
    Array.Clear(content, 0, 1024);
} while (length == 1024);
buffer.Close();
TextBox1.Text = sb.ToString();

```

8. 執行並測試。





## 壓縮資料概念

當資料量太大時而又必須用於網路傳輸時，可以進行檔案壓縮。.NET Framework 提供兩種壓縮資料流的方法，分別是 GZIP 與 Deflated，這兩種方式都是無失真檔案壓縮程解壓縮的工業標準演算法。而它們的壓縮解壓縮的大小限制是不能大於 4GB，同時若是針對已經壓縮的資料流進行壓縮可能會變的更大。這兩個功能的類別實作在 System.IO.Compression 的命名空間之下。

### DeflateStream 類別

使用結寶(Deflate)演算法進行壓縮和解壓縮資料流。因為這個類別也是繼承自 Stream 類別，因此使用方式與前述的 Stream 類似。

### GZipStream 類別

提供資料流的壓縮和解壓縮功能，因為這個類別也是繼承自 Stream 類別，因此使用方式與前述的 Stream 類似。

### 使用 GZipStream 類別進行壓縮與解壓縮檔案

這個範例是使用 GzipStream 類別進行檔案壓縮作業。

- 先判斷檔案是否存在。

2. 使用 FileStream 類別開啓指定檔案，然後使用 FileStream 物件的 Read 方法讀取檔案內的 Byte 陣列資訊。
3. 建立一個 FileStream 物件作業壓縮後的輸出檔案。
4. 建立 GZipStream 物件並指定進行壓縮 (CompressionMode.Compress) 及輸出到檔案。
5. 呼叫 GZipStream 物件的 Write 方法進行壓縮的輸出。

Visual Basic

```

Dim fileName As String = TextBox1.Text
If Not File.Exists(fileName) Then
    MessageBox.Show("檔案不存在")
    Exit Sub
End If

Using fs As New _
    FileStream(fileName, FileMode.Open, FileAccess.Read)
    Dim fileLength As Integer = fs.Length
    Dim originalData(fileLength - 1) As Byte
    fs.Read(originalData, 0, fileLength)

    Dim outputFile As New FileStream(fileName + ".gzip", _
        FileMode.Create, FileAccess.Write)

    Dim gzip As New GZipStream(outputFile, _
        CompressionMode.Compress)
    gzip.Write(originalData, 0, fileLength)
    gzip.Close()
    outputFile.Close()
End Using

```

C#

```

String fileName = TextBox1.Text;
if (!File.Exists(fileName))
{
    MessageBox.Show("檔案不存在");
    return;
}

using (FileStream fs = new
    FileStream(fileName, FileMode.Open, FileAccess.Read))
{
    int fileLength = (int)fs.Length;
    Byte[] originalData = new Byte[fileLength];
    fs.Read(originalData, 0, fileLength);

    FileStream outputFile = new FileStream(fileName + ".gzip",
        FileMode.Create, FileAccess.Write);

    GZipStream gzip = new GZipStream(outputFile,

```

```

        CompressionMode.Compress);
gzip.Write(originalData, 0, fileLength);
gzip.Close();
outputFile.Close();
}

```

這個範例是使用 GZipStream 類別進行檔案的解壓縮作業。

1. 首先判斷檔案是否存在。
2. 使用 FileStream 物件開啟要解壓縮的檔案。
3. 建立 GZipStream 物件並指定為進行解壓縮檔案  
(CompressionMode.Decompress)。
4. 建立 FileStream 物件為準備輸出解壓縮結果的檔案。
5. 在迴圈中使用 GZipStream 物件的 Read 方法進行檔案的解壓縮讀取，並寫入到檔案。

```

Visual Basic
Dim fileName As String = TextBox1.Text
If Not File.Exists(fileName) Then
    MessageBox.Show("檔案不存在")
    Exit Sub
End If

Using fs As New FileStream(fileName, _
    FileMode.Open, FileAccess.Read)
    Dim gzip As New GZipStream(fs, _
        CompressionMode.Decompress)

    Dim outputFN As String = _
        Path.GetDirectoryName(fileName) + "\\" + _
        Path.GetFileNameWithoutExtension(fileName) + ".org"
    Dim outputFS As New FileStream(outputFN, _
        FileMode.Create, FileAccess.Write)
    Dim buffer(1023) As Byte
    Dim readLength As Integer
    Do
        readLength = gzip.Read(buffer, 0, 1024)
        outputFS.Write(buffer, 0, readLength)
        Array.Clear(buffer, 0, 1024)
    Loop While readLength = 1024
    outputFS.Close()
End Using

```

```

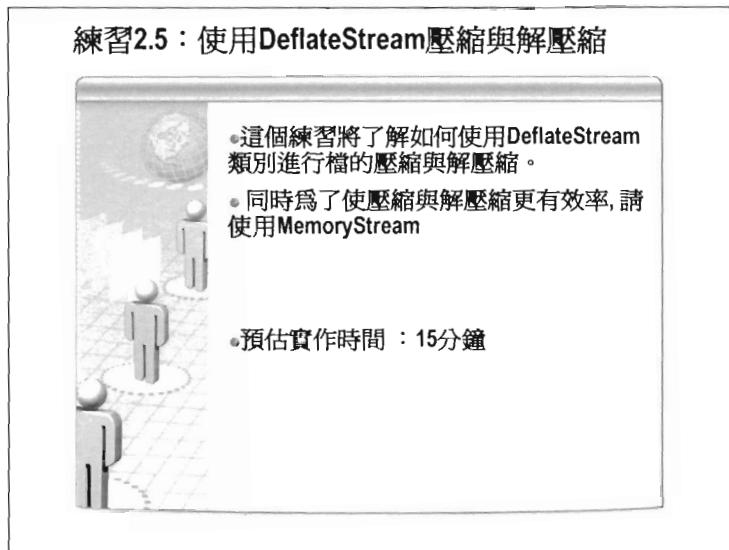
C#
String fileName = TextBox1.Text;

```

```
if (!File.Exists(fileName) )
{
    MessageBox.Show("檔案不存在");
    return;
}

using (FileStream fs = new
    FileStream(fileName, FileMode.Open, FileAccess.Read))
{
    GZipStream gzip = new GZipStream(fs,
        CompressionMode.Decompress);

    String outputFN = Path.GetDirectoryName(fileName) + "\\"
        + Path.GetFileNameWithoutExtension(fileName) + ".org";
    FileStream outputFS = new FileStream(outputFN,
        FileMode.Create, FileAccess.Write);
    Byte[] buffer = new Byte[1024];
    int readLength ;
    do
    {
        readLength = gzip.Read(buffer, 0, 1024);
        outputFS.Write(buffer, 0, readLength);
        Array.Clear(buffer, 0, 1024);
    } while (readLength == 1024);
    outputFS.Close();
}
```



## 練習 2.5 : 使用 DeflateStream 壓縮與解壓縮

### 目的：

這個練習將了解如何使用 DeflateStream 類別進行檔的壓縮與解壓縮。

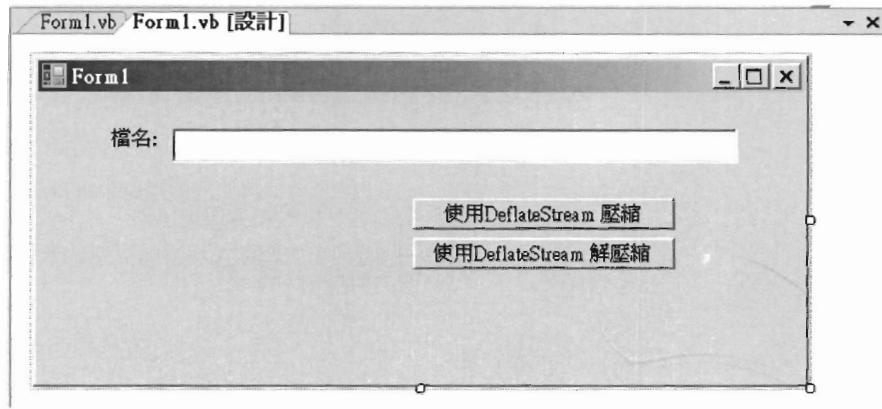
### 功能描述：

在這個練習中在一個 Windows 的介面指定要進行壓縮與解壓縮的檔名，然後將指定檔名進行壓縮與解壓縮。同時為了使壓縮與解壓縮更有效率，請使用 MemoryStream。

### 預估實作時間 : 15 分鐘

### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod02\_5。
3. 從「Solution Explorer」開啟 Form1，設計畫面如下：



4. 進入程式碼檢視視窗，匯入必要的命名空間：

```
Visual Basic
Imports System.IO
Imports System.IO.Compression
```

```
C#
using System.IO;
using System.IO.Compression;
```

5. 寫一個函式可以將傳入的 Byte 陣列進行壓縮，並將壓縮的結果以 Byte 陣列傳回。

```
Visual Basic
Function CompressBytes(ByVal doc As Byte()) As Byte()
    Using inmemory As New MemoryStream()
        Dim def As New DeflateStream(inmemory, CompressionMode.Compress)
        def.Write(doc, 0, doc.Length)
        def.Flush()
        Return inmemory.ToArray()
    End Using
End Function
```

```
C#
byte[] CompressBytes(byte[] doc) {
    using (MemoryStream inmemory = new MemoryStream())
    {
        DeflateStream def = new DeflateStream(inmemory, CompressionMode.Compress);
        def.Write(doc, 0, doc.Length);
        def.Flush();
        return inmemory.ToArray();
    }
}
```

6. 寫一個函式可以將傳入的 Stream 進行解壓縮，並將解壓縮的使用 MemoryStream 的 WriteTo 輸出到另一個 Stream 型別的參數。

```
Visual Basic
Sub DecompressStream(ByVal input As Stream, ByVal output As Stream)
    Using inmemory As New MemoryStream()
        Dim def As New DeflateStream(input, CompressionMode.Decompress)
        Dim buffer(1023) As Byte
        Dim readLength As Integer
        Do
            readLength = def.Read(buffer, 0, 1024)
            inmemory.Write(buffer, 0, readLength)
            Array.Clear(buffer, 0, 1024)
        Loop While readLength = 1024
        inmemory.Flush()
        inmemory.WriteTo(output)
    End Using
End Sub
```

```
C#
void DecompressStream(Stream input, Stream output)
{
    using (MemoryStream inmemory = new MemoryStream())
    {
        DeflateStream def = new DeflateStream(input, CompressionMode.Decompress);
        Byte[] buffer = new Byte[1024];
        int readLength;
        do
        {
            readLength = def.Read(buffer, 0, 1024);
            inmemory.Write(buffer, 0, readLength);
            Array.Clear(buffer, 0, 1024);
        } while (readLength == 1024);
        inmemory.Flush();
        inmemory.WriteTo(output);
    }
}
```

7. 進入 Button1\_Click 事件程序。撰寫壓縮檔的程式碼。  
先判斷檔案是否存在。

```
Visual Basic
Dim fileName As String = TextBox1.Text
If Not File.Exists(fileName) Then
    MessageBox.Show("檔案不存在")
Exit Sub
```

```

End If

Using fs As New FileStream(fileName, FileMode.Open, FileAccess.
Read)
    Dim fileLength As Integer = fs.Length
    Dim originalData(fileLength - 1) As Byte
    fs.Read(originalData, 0, fileLength)
    fs.Close()

    Dim output As Byte() = CompressBytes(originalData)
    Dim outputFile As New FileStream(fileName + ".def", FileMode.
Create, FileAccess.Write)
        outputFile.Write(output, 0, output.Length)
        outputFile.Close()
End Using

```

```

C#
String fileName = TextBox1.Text;
if (!File.Exists(fileName))
{
    MessageBox.Show("檔案不存在");
    return;
}
using (FileStream fs = new FileStream(fileName, FileMode.Open,
FileAccess.Read))
{
    int fileLength = (int)fs.Length;
    Byte[] originalData = new Byte[fileLength];
    fs.Read(originalData, 0, fileLength);
    fs.Close();
    byte[] output = CompressBytes(originalData);
    FileStream outputFile = new FileStream(fileName + ".def", Fil
eMode.Create, FileAccess.Write);
        outputFile.Write(output, 0, output.Length);
        outputFile.Close();
}

```

#### 8. 進入 Button2\_Click 事件程序。撰寫解壓縮程式。

```

Visual Basic
Dim fileName As String = TextBox1.Text
If Not File.Exists(fileName) Then
    MessageBox.Show("檔案不存在")
    Exit Sub
End If
Using fs As New FileStream(fileName, FileMode.Open, FileAccess.
Read)
    Dim outputFN As String =
        Path.GetDirectoryName(fileName) + "\" +
        Path.GetFileNameWithoutExtension(fileName) + ".org"
    Dim outputFS As New FileStream(outputFN, FileMode.Create,
FileAccess.Write)

```

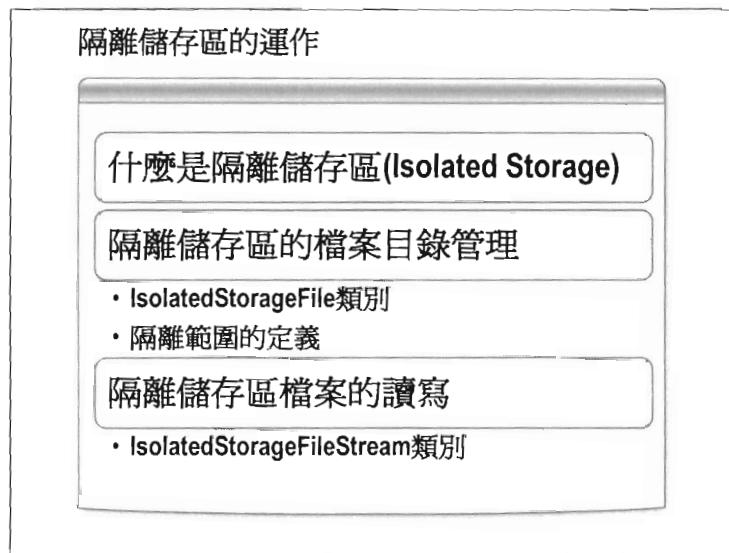
```
DecompressStream(fs, outputFS)
outputFS.Close()
End Using
```

```
C#
String fileName = TextBox1.Text;
if (!File.Exists(fileName) )
{
    MessageBox.Show("檔案不存在");
    return;
}

using (FileStream fs = new FileStream(fileName, FileMode.Open,
FileAccess.Read))
{
    String outputFN = Path.GetDirectoryName(fileName) + "\\"
        + Path.GetFileNameWithoutExtension(fileName) + ".org";
    FileStream outputFS = new FileStream(outputFN, FileMode.Create,
    FileAccess.Write);
    DecompressStream(fs, outputFS);
    outputFS.Close();
}
```

9. 執行並測試專案。

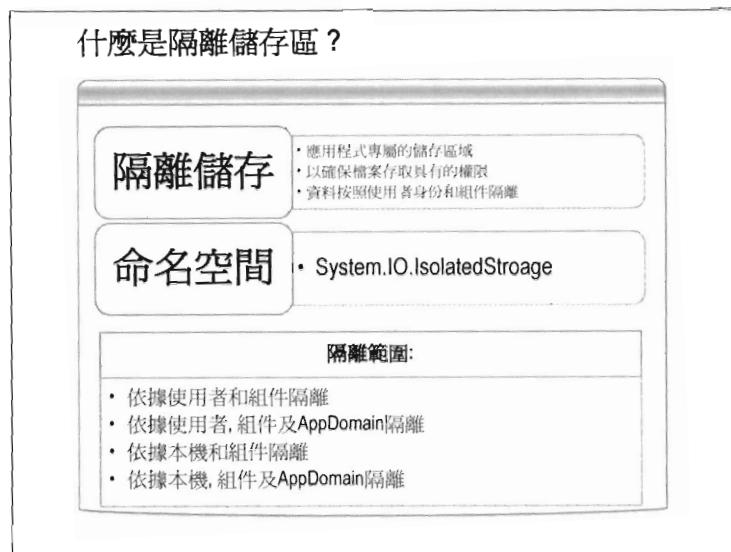




## 隔離儲存區的運作

這一節我們將介紹隔離儲存區 (Isolated Storage)，資料隔離存放的運作及目的，這節內容如下：

- 什麼是隔離儲存區 (Isolated Storage)
- 隔離儲存區的檔案目錄管理
- IsolatedStorageFile 類別
- 隔離範圍的定義
- 隔離儲存區檔案的讀寫
- IsolatedStorageFileStream 類別



## 什麼是隔離儲存區

一般檔案是放置在檔案系統中，也就是磁碟的任何目錄裡，只要有權限的使用者便可開啟檔案。不過，為了安全的原由，使用者登入的身份或者應用程式的來源可能是最低的權限，可是應用程式又有一些運算所需的暫存檔或記錄檔對檔案系統需要有讀寫的權限，安全原則與應用程式的需求似乎有所抵觸。

.NET Framework 提供一個很好的作法，就是畫分一個專區給特定的程式使用，系統安全人員大可將其他資料夾的安全性設成最低權限，應用程式若需要存取檔案系統直接在所屬專區存取，開發人員不用擔心部署後的環境沒有磁碟權限可以供應用程式使用。我們將這個專區稱為隔離儲存區。

隔離範圍包含：

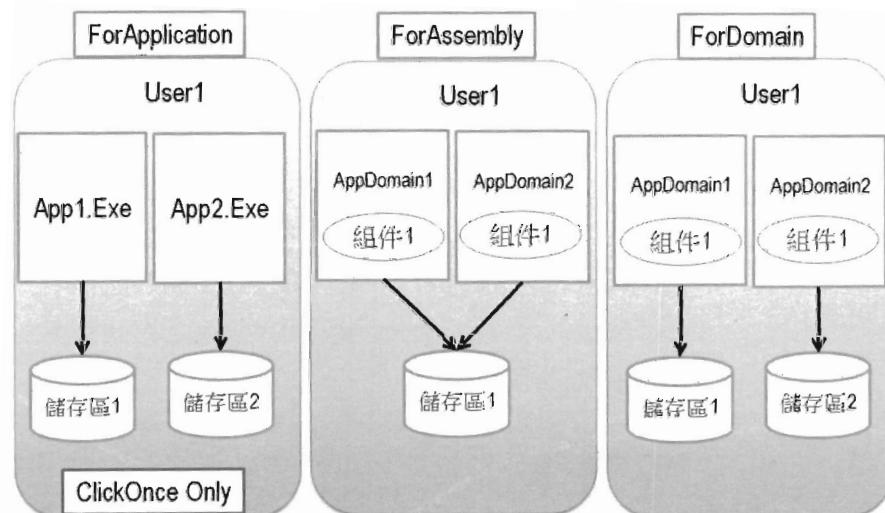
- 依據使用者和應用程式 (限 ClickOnce)
- 依據使用者和組件隔離
- 依據使用者、組件及 AppDomain 隔離
- 依據本機和應用程式 (限 ClickOnce)
- 依據本機和組件隔離
- 依據本機、組件及 AppDomain 隔離

依據使用者和隔離方式，存放在以下位置：

作業系統	位置
Windows Server 2003 及 Windows XP	<SYSTEMDRIVE>\Documents and Settings\<user>\Local Settings\Application Data\IsolatedStorage
Windows Server 2008 及 Vista	<SYSTEMDRIVE>\Users\<user>\AppData\Local\IsolatedStorage

以上路徑是目前登入使用者專屬目錄，接下會根據以下的識別方式建立的子目錄做為區隔：

- ForApplication，根據應用程式的識別碼區隔出儲存區，不同的應用程式存放不同的位置，此項限 ClickOnce 的應用程式使用。
- ForAssembly，根據組件的識別碼區隔，相同組件使用相同的儲存區。
- ForDomain，根據應用程式定義域 (AppDomain) 及組件的識別碼區隔，因此相同組件執行在不同的應用程式定義域會儲存在不同的儲存區。

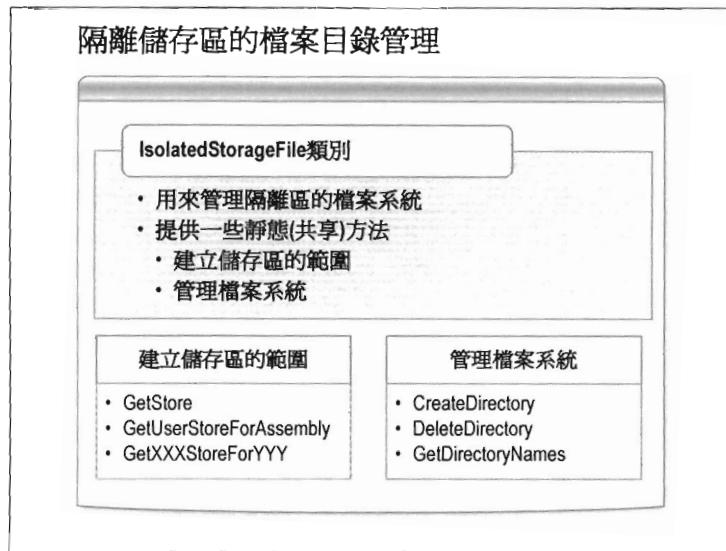


依據本機和隔離方式，是存放在以下位置：

作業系統	位置
Windows Server 2003 及 Windows XP	<SYSTEMDRIVE>\Documents and Settings\All Users\Local Settings\Application Data\IsolatedStorage

Windows Server 2008 及 Vista	<SYSTEMDRIVE> \ProgramData\IsolatedStorage
--------------------------------	---

以上路徑是本機使用者應用程式設定的共用路徑，依據本機和隔離方式(是這三種 ForApplication、ForAssembly、ForDomain)，與依據使用者和隔離方式一樣，差別在於本機和隔離方式是與所有使用者共用儲存區。



## IsolatedStorageFile 類別

用來取得應用程式存放檔案的虛擬檔案系統。為了建立及取得可以使用的存放區，IsolatedStorageFile 提供三種靜態方法，分別用來取得不同隔離儲存區的方法：

- GetMachineStoreForApplication，根據本機及應用程式的識別碼區隔出儲存區，不同的應用程式存放不同的位置，此項限 ClickOnce 的應用程式使用。
- GetMachineStoreForAssembly，根據本機及組件的識別碼區隔，相同組件使用相同的儲存區。
- GetMachineStoreForDomain，根據本機、應用程式定義域 (AppDomain) 及組件的識別碼區隔，因此相同組件執行在不同的應用程式定義域會儲存在不同的儲存區。
- GetUserStoreForApplication，根據使用者及應用程式的識別碼區隔出儲存區，不同的應用程式存放不同的位置，此項限 ClickOnce 的應用程式使用。
- GetUserStoreForAssembly，根據使用者及組件的識別碼區隔，相同組件使用相同的儲存區。

- GetUserStoreForDomain，根據使用者、應用程式定義域 (AppDomain) 及組件的識別碼區隔，因此相同組件執行在不同的應用程式定義域會儲存在不同的儲存區。

隔離儲存區有可能會被系統管理人員限制存取空間，因此可以利用以下兩個 IsolatedStorageFile 物件實體的屬性取得空間資訊：

- CurrentSize，可以取得隔離儲存區目前的大小。
- MaximumSize，可以取得隔離儲存區配額限制內最大空間。

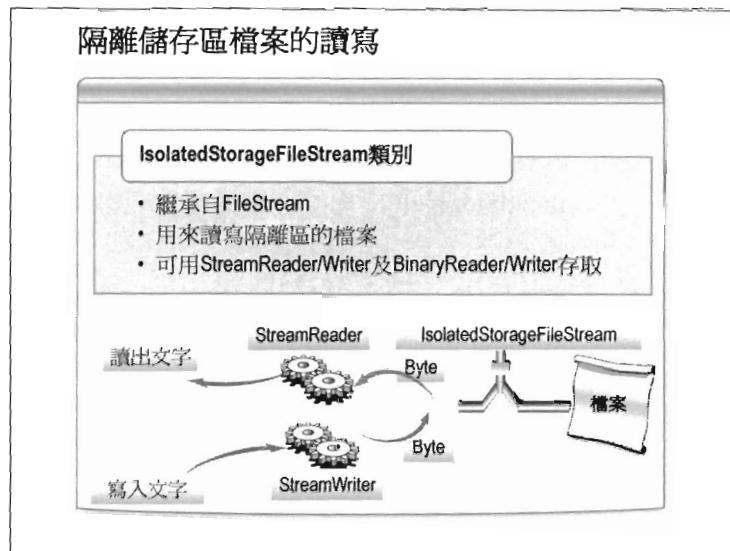
以下範例是在隔離區儲存使用者個人喜好，隔離區要根據使用者與組件做為區隔，並建立子目錄 newFolder。

Visual Basic

```
Dim isoFile As IsolatedStorageFile  
isoFile = IsolatedStorageFile.GetUserStoreForAssembly()  
isoFile.CreateFile("newFolder")
```

C#

```
IsolatedStorageFile isoFile;  
isoFile = IsolatedStorageFile.GetMachineStoreForDomain();  
isoFile.CreateFile("newFolder");
```



## IsolatedStorageFileStream 類別

自 `FileStream` 類別繼承而來，功能與 `FileStream` 幾乎一樣，唯一不同的是將檔案存放在指定的隔離儲存區。

建立 `IsolatedStorageFileStream` 物件呼叫其建構函式時，在最後一個參數傳入隔離區 (`IsolatedStorageFile` 型別) 的物件即可。

### Visual Basic

```
Dim fs As New IsolatedStorageFileStream(fileName, _
    FileMode.Open, FileAccess.Read, isoFile)
Dim sr As New StreamReader(fs,
    Encoding.GetEncoding("big5"))
TextBox2.Text = sr.ReadToEnd()
sr.Close()
fs.Close()
```

### C#

```
FileStream fs = new IsolatedStorageFileStream(fileName,
    FileMode.Open, FileAccess.Read, isoFile)
StreamReader sr = new StreamReader(fs,
    Encoding.GetEncoding("big5"));
TextBox2.Text = sr.ReadToEnd();
sr.Close();
```

### 練習2.6：讀寫隔離儲存區的檔案

•這個練習使用IsolatedStorageFile建立以本機及組件為隔離的儲存區，並使用IsolatedStorageFileStream建立檔案資料流以進行讀寫作業。

•預估實作時間：15分鐘

### 練習 2.6：讀寫隔離儲存區的檔案

#### 目的：

這個練習使用 IsolatedStorageFile 建立以本機及組件為隔離的儲存區，並使用 IsolatedStorageFileStream 建立檔案資料流以進行讀寫作業。

#### 功能描述：

在這個練習要完成一個可以記錄公司名稱在本機和組件的範圍裡，檔名是 CompanySettings.dat

*CompanySettings*

預估實作時間：15 分鐘

#### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod02\_6。

3. 從「Solution Explorer」開啓 Form1，要有一個 Label，一個 TextBox，一個 Button。Label 的 Text 設為「公司名稱」，Button 的 Text 設為「Save」。

4. 進入程式碼檢視視窗，匯入必要的命名空間：

```
Visual Basic
Imports System.IO
Imports System.IO.IsolatedStorage
Imports System.Text
```

```
C#
using System.IO;
using System.IO.IsolatedStorage;
```

5. 宣告表單層級的私有變數 isoFile 為 IsolatedStorageFile 型別。

```
Visual Basic
Dim isoFile As IsolatedStorageFile
```

```
C#
IsolatedStorageFile isoFile;
```

6. 在表單載入時(Form1\_Load 事件)初始化 isoFile 變數為本機與組件為隔離的儲存區：

```
Visual Basic
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    isoFile = IsolatedStorageFile.GetMachineStoreForAssembly()
    If isoFile.GetFileNames("CompanySettings.dat").Length = 0 Then
        Return
    End If

    Dim isoStream As New IsolatedStorageFileStream("CompanySettings.dat", FileMode.Open, isoFile)
    Dim result As String = (New StreamReader(isoStream)).ReadToEnd()
    isoStream.Close()
    TextBox1.Text = result
End Sub
```

```
C#
private void Form1_Load(object sender, EventArgs e)
{
```

```

isoFile = IsolatedStorageFile.GetMachineStoreForAssembly();
if (isoFile.GetFileNames("CompanySettings.dat").Length == 0)
    return;

IsolatedStorageFileStream isoStream =
    new IsolatedStorageFileStream("CompanySettings.dat", FileMode.Open, isoFile);

String result = (new StreamReader(isoStream)).ReadToEnd();
isoStream.Close();
TextBox1.Text = result;
}

```

7. 進入 Button1\_Click 事件程序。撰寫開啓檔案及寫檔的程式碼：

Visual Basic

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim isoStream As New IsolatedStorageFileStream("CompanySettings.dat", FileMode.Create, isoFile)
    Dim sw As New StreamWriter(isoStream, Encoding.UTF8)
    sw.WriteLine(TextBox1.Text)
    sw.Close()
End Sub

```

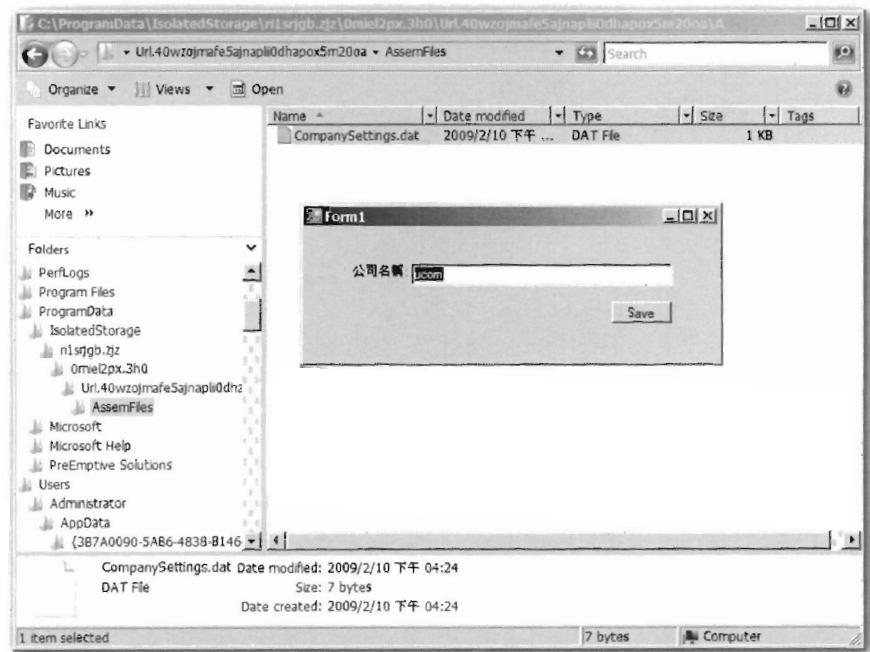
C#

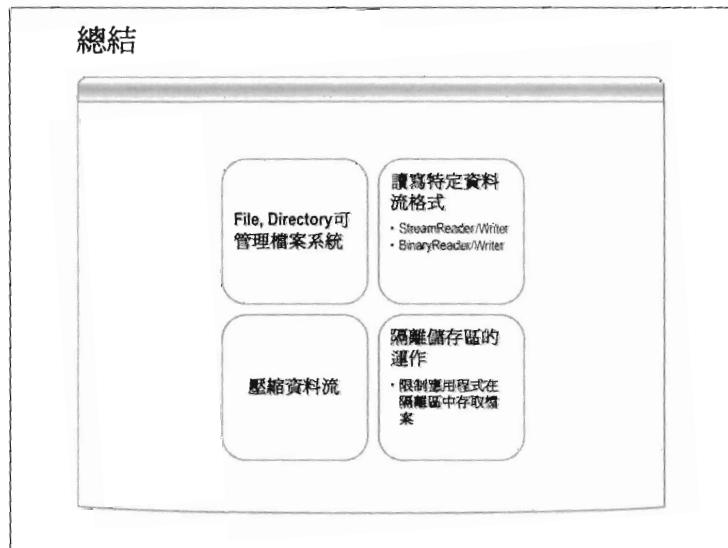
```

private void Button1_Click(object sender, EventArgs e)
{
    IsolatedStorageFileStream isoStream =
        new IsolatedStorageFileStream("CompanySettings.dat", FileMode.Create, isoFile);
    StreamWriter sw=new StreamWriter(isoStream, Encoding.UTF8);
    sw.WriteLine(TextBox1.Text);
    sw.Close();
}

```

8. 執行結果如下：





## 總結

這個章節中，了解如何從程式中進行檔案及目錄的列舉、刪除以及搬移，接著學習如何從資料串流中進行檔案的讀與寫功能，其中可以透過 StreamReader、StreamWriter 進行文字檔案的讀寫動作。

了解如何透過程式進行檔案的壓縮、解壓縮作業。並且學習如何將應用程式專屬的資料檔存放在安全的隔離儲存區，及其運作方式。

## 第三章：文字處理

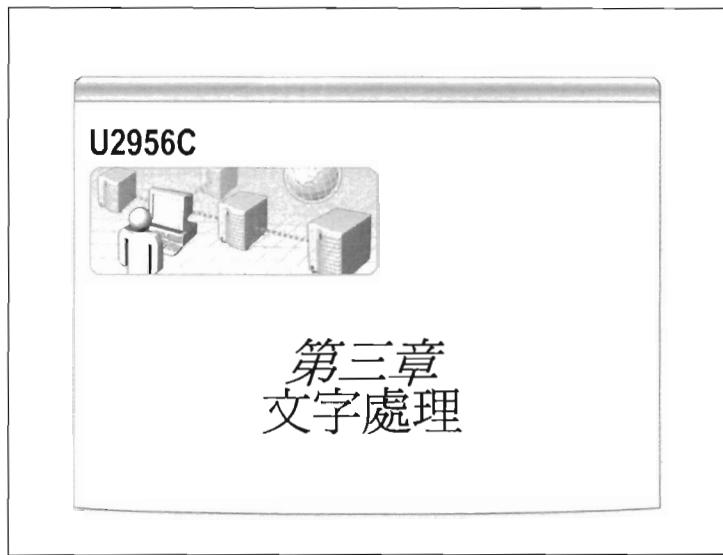
### 本章大綱

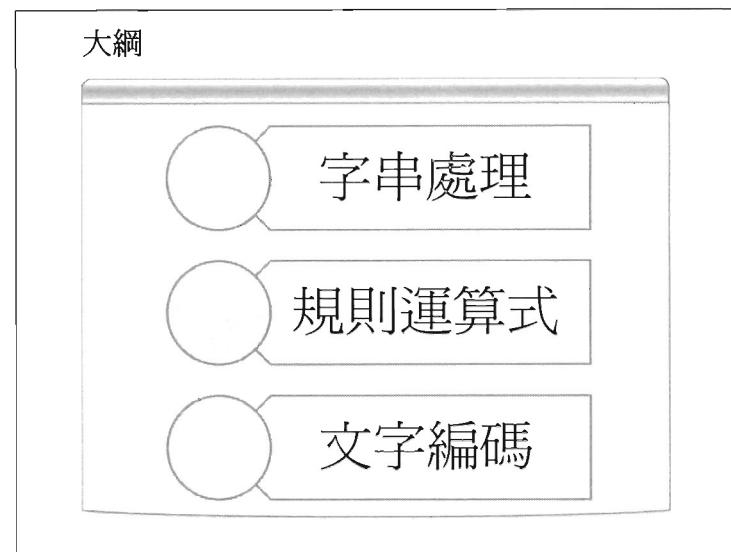
字串處理.....	4
StringBuilder 類別 .....	6
練習 3.1：使用 StringBuilder .....	7
規則運算式.....	9
認識規則運算式.....	10
規則運算式的格式 .....	11
.NET Framework 的規則運算式.....	13
文字格式比對 .....	15
擷取特定規則的文字.....	17
練習 3.2：使用 Regex 擷取標題.....	19
認識文字編碼.....	22
使用 Encoding 類別.....	24
練習 3.3：使用 UTF8 讀取文件 .....	26
總結.....	29

作者：

羅慧真





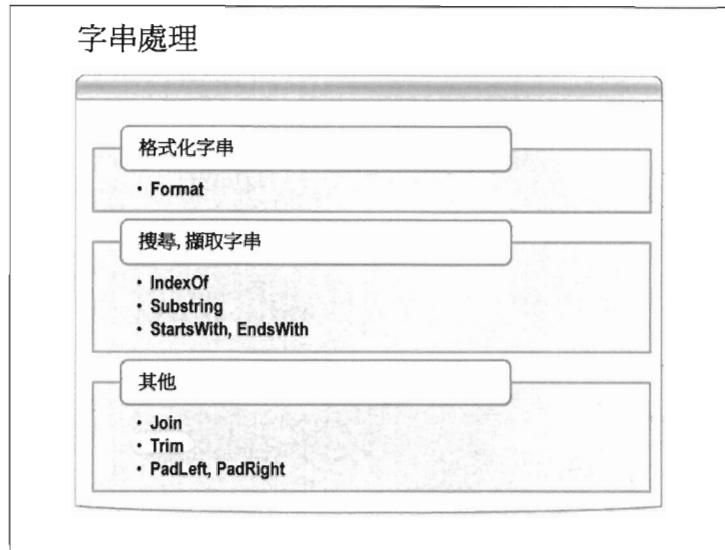


---

在這個章節中將介紹 String 類別的特性以及如何讓經常變更的字串處理更有效率。除了字串本身之外，對於文字你需要更多的檢查格式、搜尋、擷取等功能，這裡也會介紹如何利用規則運算式進行文字的處理。另外，一般的文字都必須經過編碼輸入輸出到資料來源 (檔案或資料庫...)。

本章介紹以下主題：

- 字串處理
- 規則運算式
- 文字編碼



## 字串處理

String 類別提供了字串常用的處理功能，包含格式化字串、搜尋、擷取字串.....等多種功能

### 格式化字串

Format 方法提供格式化字串的處理，以下範例印出日期資訊。

```

Visual Basic
Dim msg As String = "今天是{0}月{1}日, 星期{2}"
MessageBox.Show(String.Format(msg,
    DateTime.Now.Month, DateTime.Now.Day,
    DateTime.Now.DayOfWeek.GetHashCode()))
  
```

```

C#
string msg = "今天是{0}月{1}日, 星期{2}";
MessageBox.Show(string.Format(msg,
    DateTime.Now.Month, DateTime.Now.Day,
    DateTime.Now.DayOfWeek.GetHashCode()));
  
```

### 搜尋、擷取字串

IndexOf 方法用來查詢某字串的索引位置，Substring 方法用來擷取字串，StartsWith 與 EndsWith 方法用來判斷字串的開頭或結束是否為指定的字串。

**Visual Basic**

```

Dim msg As String = "String 類別提供了字串常用的處理功能"
If msg.StartsWith("String") Then
    MessageBox.Show("擷取前六個字:" + msg.Substring(0, 6))
End If

If msg.EndsWith("功能") Then
    Dim idx As Integer = msg.LastIndexOf("功能")
    MessageBox.Show("擷取\"功能\"兩個字:" & _
        msg.Substring(idx, 2))
End If

```

**C#**

```

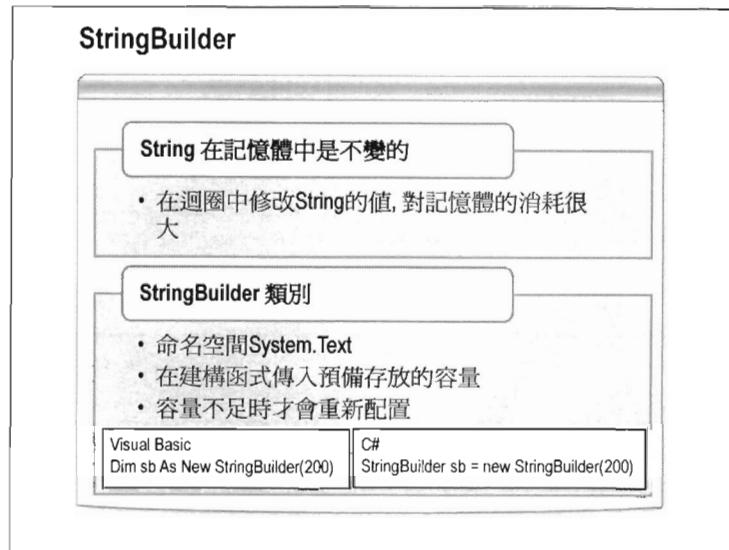
string msg = "String 類別提供了字串常用的處理功能";
if (msg.StartsWith("String"))
{
    MessageBox.Show("擷取前六個字:" + msg.Substring(0, 6));
}

if (msg.EndsWith("功能"))
{
    int idx = msg.LastIndexOf("功能");
    MessageBox.Show("擷取\"功能\"兩個字:" + 
        msg.Substring(idx, 2));
}

```

**其他**

Join 方法可以將字串陣列串成一個字串。Trim 方法用來將字串前後的空間去除。PadLeft、PadRight 方法則是將文字左或右方補上特定字元。



String 在記憶體中是不變的，字串被修改時，會放棄原有儲放位置，將修改字串放在新的位置，所以程序或迴圈中若需要一直修改字串，將對記憶體產生很大的消耗。雖然 GC 會去清理不用的記憶體，但仍難免耗費系統資源。

## StringBuilder 類別

.NET Framework 在命名空間 System.Text 提供一個 StringBuilder 類別可改善這個問題。它可以在建立物件時會先預備一個存放容量，當字串改變時直接在原位置修改、加入、插入，直到預備容量不足時才會重新配置更大的容量。以下範例建立存放容量為 200 的 StringBuilder 物件：

Visual Basic Dim sb As New StringBuilder(200)
--

C# StringBuilder sb =new StringBuilder(200);
---

### 練習3.1：使用StringBuilder

•這個練習要顯示磁碟資訊於畫面，收集資訊時為了避免迴圈使用String會浪費系統資源，使用StringBuilder收集資訊。

•預估實作時間：5分鐘

### 練習 3.1：使用 StringBuilder

#### 目的：

這個練習要顯示磁碟資訊於畫面，收集資訊時為了避免迴圈使用 String 會浪費系統資源，使用 StringBuilder 收集資訊。

#### 功能描述：

在這個練習中會用到 DriveInfo 取得磁碟資訊，並在 For Each 迴圈中逐項取得資訊於 StringBuilder，最後呼叫 StringBuilder 的 ToString()方法顯示於 Label 上。

#### 預估實作時間：5 分鐘

#### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啟動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod03\_1。

3. 在 Form 放上一個 Button 及一個 Label。
4. Button 的 Click 事件，寫下列舉電腦所有磁碟資訊功能的程式，使用 StringBuilder 記錄，最後呼叫 ToString() 顯示在 Label1 上，如以下程式碼：

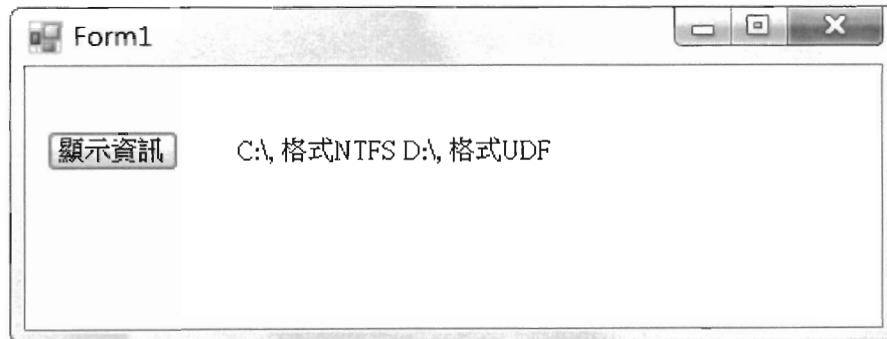
Visual Basic

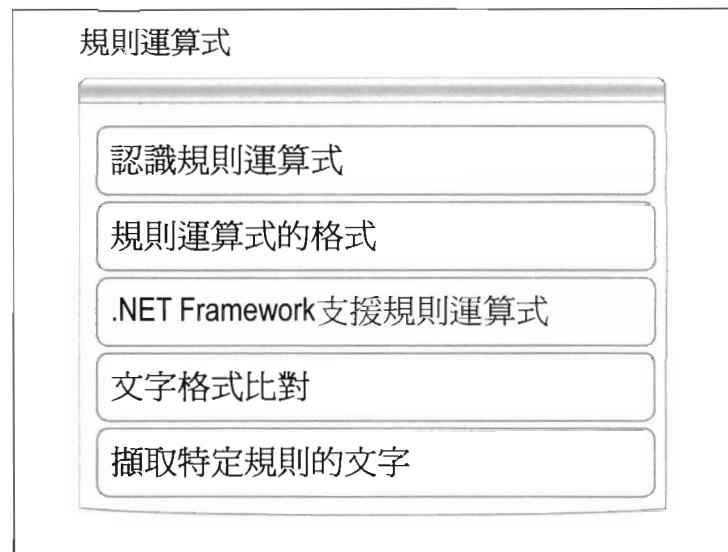
```
Dim sb As New StringBuilder(200)
For Each info As DriveInfo In DriveInfo.GetDrives
    sb.AppendFormat("{0}, 格式{1} ", info.Name, info.DriveFormat)
Next
Label1.Text = sb.ToString()
```

C#

```
StringBuilder sb = new StringBuilder(200);
foreach(DriveInfo info in DriveInfo.GetDrives())
{
    if(info.IsReady)
        sb.AppendFormat("{0}, 格式{1} ", info.Name, info.DriveFormat);
}
Label1.Text = sb.ToString();
```

5. 執行並測試。



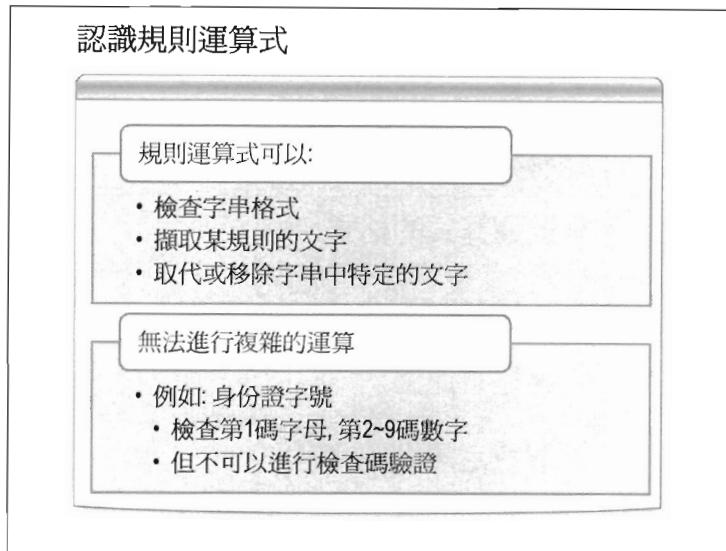


---

## 規則運算式

這一節將了解什麼是規則運算式，.NET Framework 提供的規則運算式功能，如何利用規則運算式進行輸入文字的格式檢查，或者擷取特定的文字。

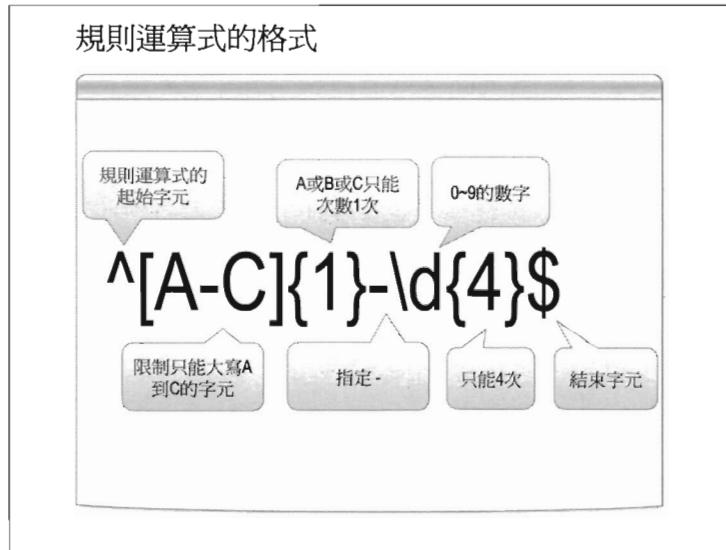
- 認識規則運算式
- 規則運算式的格式
- .NET Framework 的規則運算式
- 文字格式比對
- 擷取特定規則的文字



## 認識規則運算式

規則運算式是.NET Framework 提供一個強大而且效率佳的一種文字處理方式。你可以利用規則運算式進行輸入字串的格式檢查，或者從輸入字串中擷取某個特定規則的文字，也可以藉由規則運算式的類別取代或移除字串中特定的文字。

規則運算式進行文字處理很有效率，不過它並不能夠用來進行複雜的計算。例如，你可以用規則運算式檢查使用者輸入的身份證字號格式是否正確 (第 1 碼必須是字母，第 2-9 碼必須是數字)，但並不能用規則運算式進行身份證字號檢查碼的驗證 (檢查碼用來檢查身份證字號是否合法)。



## 規則運算式的格式

規則運算式使用一些符號來代表某些特定的規則，以

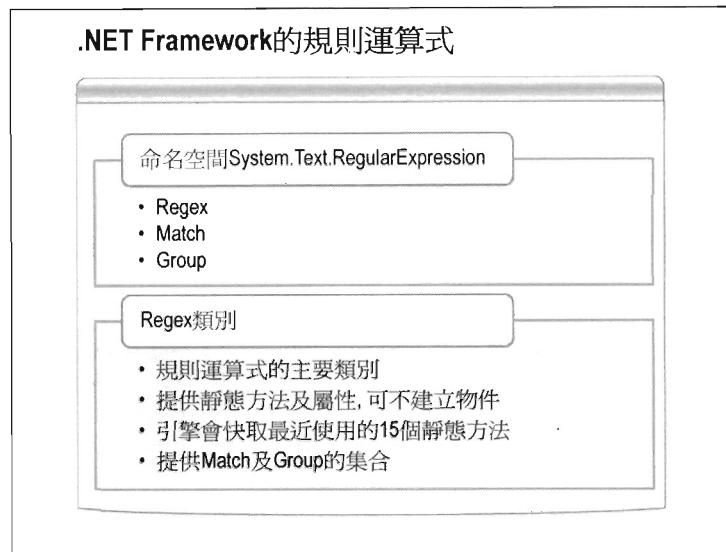
`^[A-C]{1}-\d{4}\$`

為例，「`^`」代表比對從字串的開始位置，「`\$`」代表比對到字串的結束，另外它有以下幾個特別的字元語法：

- `[A-Z]`，代表允許大寫 A-Z 的文字。使用中括號`[]`輸入允許的字元。
- `[^A-Z]`，代表不允許大寫 A-Z 的文字。使用中括號`[^]`輸入允許的字元。
- `\w`，如同`[A-Za-z_0-9]`，允許大小寫字母、數字及底線。
- `\W`，與`\w`相反。不允許大小寫字母、數字及底線。
- `\d`，如同`[0-9]`，允許 0 到 9 的數字。
- `\D`，與`\d`相反，不允許 0 到 9 的數字。
- `\s`，允許空白字元。
- `\S`，與`\s`相反，不允許空白字元。

以下是數量語法，代表某規則可以出現的次數：

- $\{n,m\}$ ，用大括號{}代表規則次數，n是最少次數，m是最多次數。 $\{3\}$ ，代表三次。 $\{3,\}$ 代表至少三次， $\{3,5\}$ 代表最少三次最多五次。
- \*，如同 $\{0,\}$ ，代表零到多次。
- ?，如同 $\{0,1\}$ ，代表零或一次。
- +，如同 $\{1,\}$ ，代表至少一次。



## .NET Framework 的規則運算式

.NET Framework 的規則運算式功能放在命名空間 System.Text.RegularExpressions，包含以下類別做為處理規則運算式的功能。

- Regex 類別，用來表示規則運算式。
- Match 類別，可以透過 Regex 的實體或靜態方法取得輸入文字符合規則的部份。
- Group 類別，用來擷取規則運算式中指定的群組。

Regex 類別是規則運算式中主要的類別，可以透過建立物件實體的方式（從建構函式傳入規則）指定要檢查或擷取的規則，或者使用靜態方法及屬性進行文字處理。.NET Framework 2.0 之後會快取最近使用過的 15 個靜態規則運算式以提升處理效率。

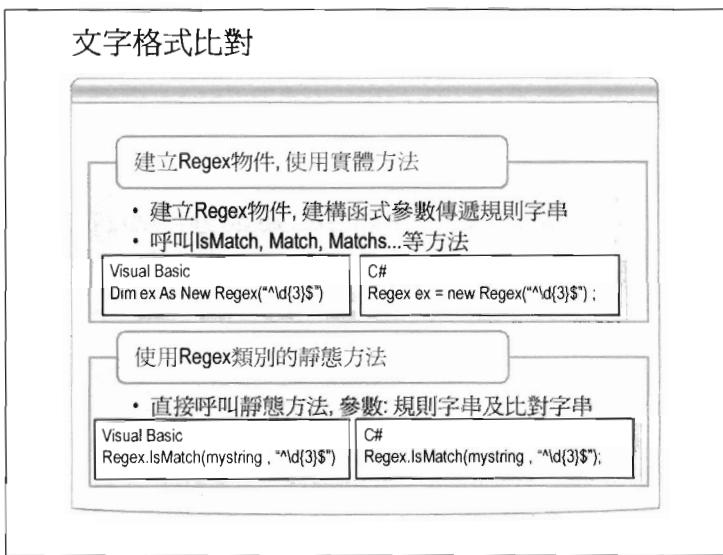
RegEx 類別的常用方法如下：

- Match()，用來搜尋字串中指定的規則，回傳 Match 類別。
- IsMatch()，用來比對字串中是否符合指定的規則，回傳 Boolean 值。

- Replace()，用來取代字串中所有符合指定規則的文字為其他文字，回傳 String 值為取代後的結果。

Match 類別，是一個單一規則運算式比對後的結果。它的常用方法如下：

- Index，從原始字串找到符合規則的第一個字元位置。
- Length，符合規則的字元數。
- Success，代表比對成功與否，回傳 Boolean。



## 文字格式比對

你可以在建立 Regex 物件的建構函式傳入規則字串，然後進行文字規則的比對。

以上範例是使用 Regex 物件的 `IsMatch` 方法檢查輸入文字是不是 3+2 郵遞區號：

```
Visual Basic
Function CheckZipCode(ByVal InputText As String) As Boolean
    Dim ex As New Regex("^\\d{3}(\\d{2})?$")
    Return ex.IsMatch(InputText)
End Function
```

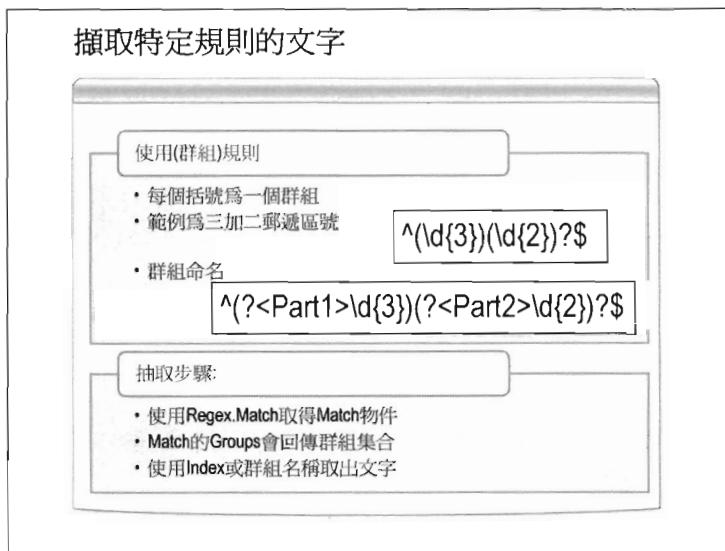
```
C#
bool CheckZipCode(string InputText )
{
    Regex ex = new Regex(@"^\\d{3}(\\d{2})?");
    return ex.IsMatch(InputText);
}
```

或者使用 `IsMatch` 的靜態 (共享 for Visual Basic) 方法：

```
Visual Basic
```

```
If Not Regex.IsMatch(TextBox2.Text, "^[A-Z]{1}\d{9}$") Then  
    MessageBox.Show("格式錯誤")  
End If
```

```
C#  
if (!Regex.IsMatch(TextBox2.Text, @"^[A-Z]{1}\d{9}$"))  
    MessageBox.Show("格式錯誤");
```



## 擷取特定規則的文字

規則運算式有群組功能，例如要定義三加二郵遞區號：

**^(?<Part1>\d{3})(?<Part2>\d{2})?\$/**

第一個括號中`\d{3}`代表前三碼的數字，第二個括號`(\d{2})?`是後兩碼，後兩碼可有可無，就是 0 或 1 的意思使用問號`(?)`。

如果在程式中擷取特定的部份也可以用群組命名，在括號內使用`?<群組名稱>`指定群組名稱，例如：第一組取名為 Part1，第二組取名為 Part2。

**^(?<Part1>\d{3})(?<Part2>\d{2})?\$/**

## 擷取步驟

以下是擷取特定文字的步驟：

1. 使用`Regex.Match`取得`Match`物件。
2. `Match`的`Groups`會回傳群組集合。
3. 使用`Index`或群組名稱取出文字。

以下範例是取出三加二郵遞區號，區域與路段的部份：

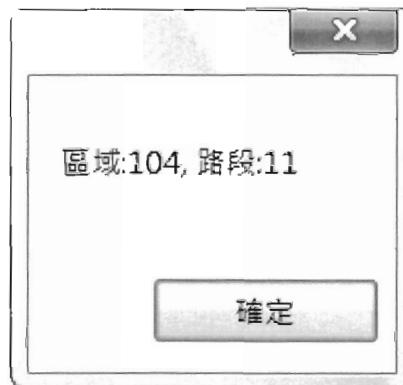
Visual Basic

```
Dim m As Match = Regex.Match(TextBox4.Text, "^(<Part1>\d{3})(?<Part2>\d{2})?$")
If m.Success Then
    MessageBox.Show(String.Format("區域:{0}, 路段:{1}",
        m.Groups("Part1").Value, m.Groups("Part2").Value))
End If
```

C#

```
Match m = Regex.Match(textBox4.Text, @"^(<Part1>\d{3})(?<Part2>\d{2})?$");
if (m.Success)
    MessageBox.Show(String.Format("區域:{0}, 路段:{1}",
        m.Groups["Part1"].Value, m.Groups["Part2"].Value));
```

假設輸入的是 10411，執行結果如下：



### 練習3.2：使用Regex 摳取標題

•這個練習要使用Regex 摳取HTML 文件中使用<h1>...</h1>或<h?>...</h?> 取出標題。

•預估實作時間：10分鐘

### 練習 3.2: 使用 Regex 摳取標題

#### 目的：

這個練習要使用 Regex 摳取 HTML 文件中使用<h1>…</h1>或<h?>…</h?> 取出標題。

#### 功能描述：

在這個練習會使用 Regex 摳取 HTML 文件中使用<h1>…</h1>、h2、h3、h4.....的標題並顯示於 ListBox 上。

**預估實作時間：10 分鐘**

#### 實作步驟：

HTML 要中擳取<Hn>之中</Hn>的文字，例如，以下字串：

```
<h1>titl1</h1><h2>titl2</h2><h3>title3</h3>
```

要取出以下的結果顯示在 ListBox 上。

```
1:titl1  
2:titl2  
3:title3
```

規則運算式要怎麼寫？

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod03\_2。
3. 在 Form 放上一個 Text、Button、及一個 ListBox。
4. 在 Button 的 Click 事件寫下以下程式，使用群組名稱指定 <hn> 將 n 取為 Level，將 <hn> 中間的文字取名為 title：

**Visual Basic**

```

Dim ex As New Regex("\<h(?<Level>\d{1})\>(?'title'\w*)\</h\d{1}\>")

For Each m As Match In ex.Matches(TextBox3.Text)
    For Each c As Capture In m.Captures
        Console.WriteLine(c.Value)
    Next
    ListBox1.Items.Add(m.Groups("Level").Value & ";" & m.Groups("title").Value)
Next

```

**C#**

```

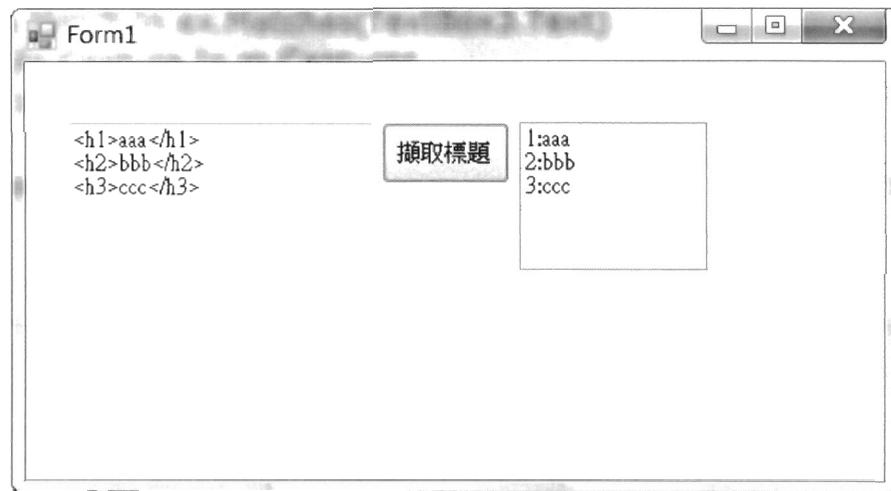
Regex ex = new Regex(@"\<h(?<Level>\d{1})\>(?'title'\w*)\</h\d{1}\>");

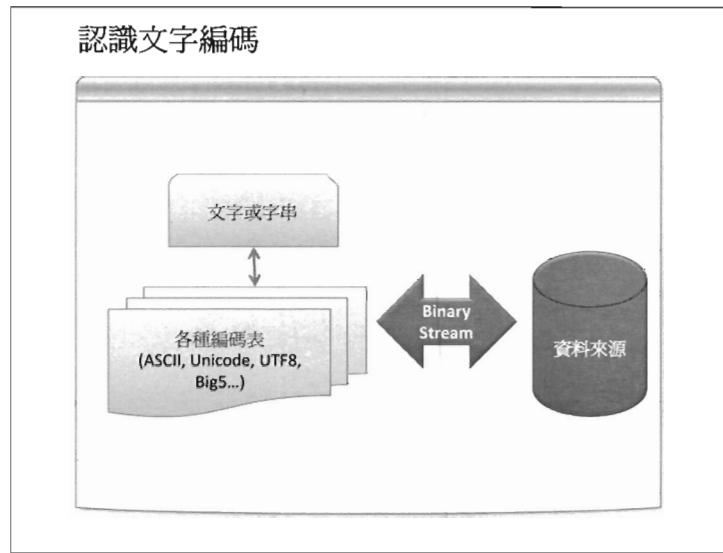
foreach (Match m in ex.Matches(TextBox3.Text))
{
    foreach (Capture c in m.Captures)
        Console.WriteLine(c.Value);

    listBox1.Items.Add(m.Groups["Level"].Value + ":" + m.Groups["title"].Value);
}

```

5. 執行結果如下：





## 認識文字編碼

基本上每個文字檔都是 Byte 序列，如果要讓這些 Byte 序列的資料正確的以文字顯示，就必須經過應用程式處理編碼 (encoding)，相同的文字要寫回檔案也必須經過編碼 (encoding) 成 Byte 陣列才可寫回檔案。

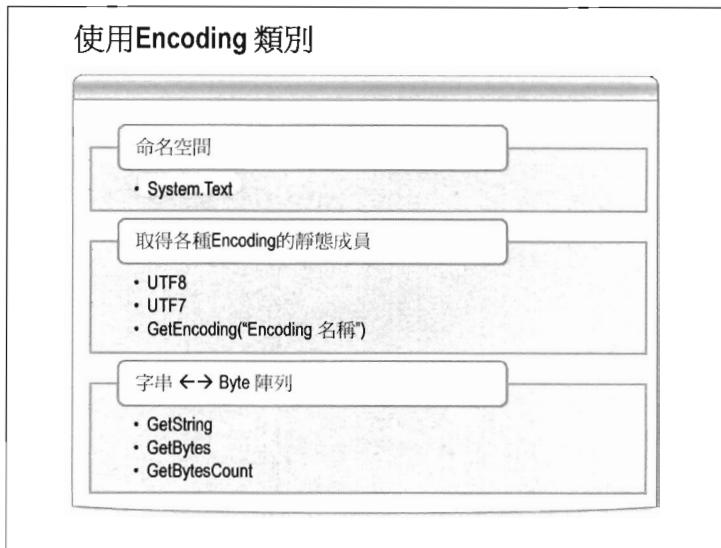
編碼處理會將一個字元轉換成一個 Byte 或兩個 Bytes (單一位元或雙位元)，字元對應幾個 Bytes 必需要看編碼格式的類型。早期電腦只有 ASCII (American Standard Code for Information Interchange)，它定義了 128 個字元，包含大小寫的英文字母、阿拉伯數字、標點符號…等適用的語言為英語。

近年來，為了可以發展多國語言的應用程式，Unicode (標準萬國碼) 以漸漸取代 ANSI (American National Standards Institute) 及 ASCII。

### 什麼是 Unicode？

Unicode 是指每個字元有一個唯一的識別碼，不論何種作業平台、程式模型、語系。是一個包含上千個字元，涵蓋各國語言的文字、字母、方言的編碼格式。Unicode 並不是唯一一種編碼標準，有數種 Unicode 標準，像是 UTF-8、UTF-16、UTF-32...等。而.NET

Framework 支援這幾種 UTF (Unicode Transformation Format) 編碼格式。



## 使用 `Encoding` 類別

.NET Framework 將文字處理的相關功能都放在 `System.Text` 的命名空間之中。`Encoding` 類別提供了許多文字編碼的功能，包含以下靜態成員可取得 `Encoding` 物件：

- `UTF8` 屬性，傳回 `UTF8` 字元集的 `Encoding` 物件。
- `UTF7` 屬性，傳回 `UTF7` 字元集的 `Encoding` 物件。
- `GetEncoding` 方法屬性，傳回指定名稱 (例如，`Big5`、`GB2312...`) 字元集的 `Encoding` 物件。

除此之外，也提供字串與 `Byte` 陣列互轉的方法：

- `GetString` 方法，傳入 `Byte` 陣列經過編碼回傳字串。
- `GetBytes` 方法，傳入 `String` 經過編碼回傳 `Byte` 陣列。
- `GetBytesCount` 方法，計算傳入的 `String` 經過編碼會有多少個 `Bytes`。

以下範例是使用 `big5` 編碼，將輸入字串取出編碼後的 `byte` 陣列並列出：

Visual Basic <code>Dim big5 As Encoding = Encoding.GetEncoding("big5")</code>
--

```
Dim buffer() As Byte = big5.GetBytes(textBox1.Text)
listBox1.Items.Add("字串：" + textBox1.Text)
listBox1.Items.Add("編碼：")
For Each b As Byte In buffer
    listBox1.Items.Add(vbTab + b.ToString())
Next
```

```
C#
Encoding big5 = Encoding.GetEncoding("big5");
byte[] buffer = big5.GetBytes(textBox1.Text);
listBox1.Items.Add("字串：" + textBox1.Text);
listBox1.Items.Add("編碼：" );
foreach (byte b in buffer) {
    listBox1.Items.Add("\t" + b.ToString());
}
```



### 練習 3.3：使用 UTF8 讀取文件

目的：

判斷文件是否使用 UTF8 編碼，如果是，就使用 UTF8 Encoding 方法讀取文件。

功能描述：

這個練習使用 Encoding 的 GetPreamble() 方法取得文件編碼方式的位元組序列，判斷如果是用 UTF8 編碼，便使用 UTF8 Encoding 方法讀取文件。

預估實作時間：5分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod03\_3。

3. 在表單中放入兩個 TextBox 及一個 Button。
4. 第二個 TextBox 的 MultiLine 設為 True。
5. 寫一個副程式用來判斷是否為指定編碼。

**Visual Basic**

```
Function IsUnicode(ByVal en As Encoding, ByVal buffer() As Byte)
As Boolean
    Dim preamble() As Byte = en.GetPreamble()
    IsUnicode = False
    For i = 0 To preamble.Length - 1
        If preamble(i) <> buffer(i) Then
            IsUnicode = False
            Exit For
        Else
            IsUnicode = True
        End If
    Next
    Return IsUnicode
End Function
```

**C#**

```
bool IsUnicode(Encoding en, byte[] buffer) {
    byte[] preamble = en.GetPreamble();
    bool IsUnicode = false;
    for (int i = 0; i < preamble.Length; i++) {
        if (preamble[i] != buffer[i])
        {
            IsUnicode = false;
            break;
        }
        else {
            IsUnicode = true;
        }
    }
    return IsUnicode;
}
```

6. 在 Button 的 Click 事件判斷是否為 UTF8 的文件，如果是使用 UTF8 編碼顯示文件於 TextBox2 上：

**Visual Basic**

```
Dim fs As New FileStream(textBox1.Text, FileMode.Open, FileAccess.Read)
Dim buffer(fs.Length - 1) As Byte
fs.Read(buffer, 0, fs.Length)
fs.Close()

If (IsUnicode(Encoding.UTF8, buffer)) Then
    textBox2.Text = Encoding.UTF8.GetString(buffer)
```

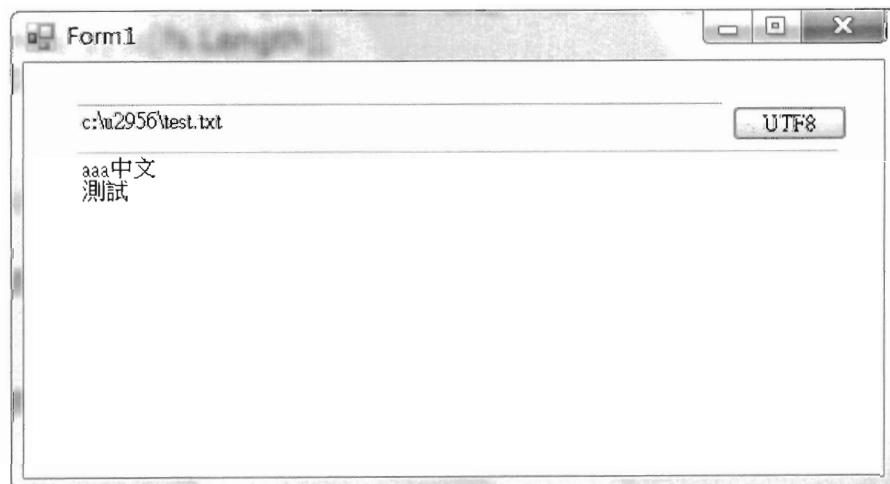
```
Else
    textBox2.Text = "未知的編碼格式"
End If
```

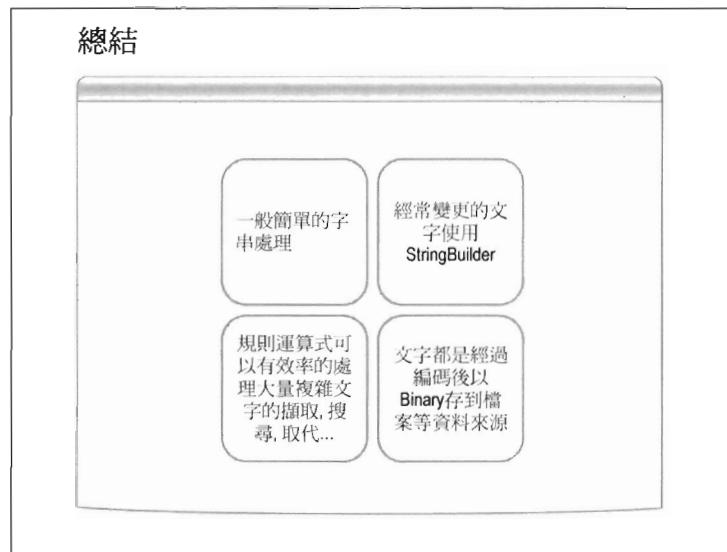
```
C#
FileStream fs = new FileStream(textBox1.Text, FileMode.Open , FileAccess.Read);
byte[] buffer = new byte[fs.Length];
fs.Read(buffer, 0,(int)fs.Length);
fs.Close();

if (IsUnicode(Encoding.UTF8, buffer))
{
    textBox2.Text = Encoding.UTF8.GetString(buffer);
}
else {
    textBox2.Text = "未知的編碼格式";
}
```

7. 使用 Notepad 編輯文件，存檔為 UTF8 的格式。

8. 執行程式並開啟該檔案。





## 總結

這一章介紹如何處理文字，包含簡單的字串處理，使用 String 類別所提供的多種方法，以及 String 在記憶體是不變的，若要經常變更使用 StringBuilder。

應用程式中輸入的字串經常需要是特定的格式，可以使用 Regex 類別處理規則運算式。

程式中的文字類型資料都是根據程式中的編碼格式 (encoding) 宣告處理過之後，再以 Binary 格式輸入或是輸出。

# 第四章：全球化程式設計

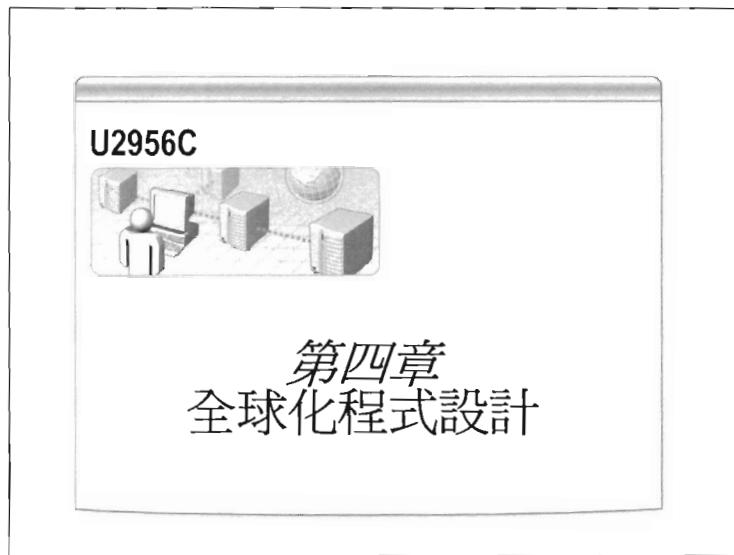
## 本章大綱

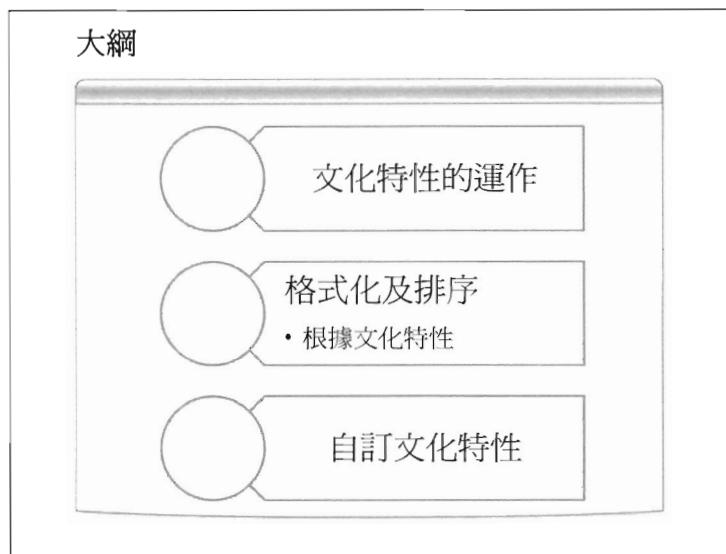
文化特性的運作.....	4
認識全球化應用程式.....	5
存取文化資訊.....	6
取得使用者的文化特性.....	8
指定特定語系.....	9
擷取所有的 CultureInfo.....	11
查看地區的細節資訊.....	12
練習 4.1：列出所有的文化資訊.....	13
根據文化特性進行.....	17
數字的格式化.....	18
日期時間的格式化.....	20
文字比較.....	22
練習 4.2：列出所有的文化資訊.....	24
自訂文化特性.....	27
設計步驟.....	28
練習 4.3：建立自訂文化特性.....	30
總結.....	34

作者：

羅慧真





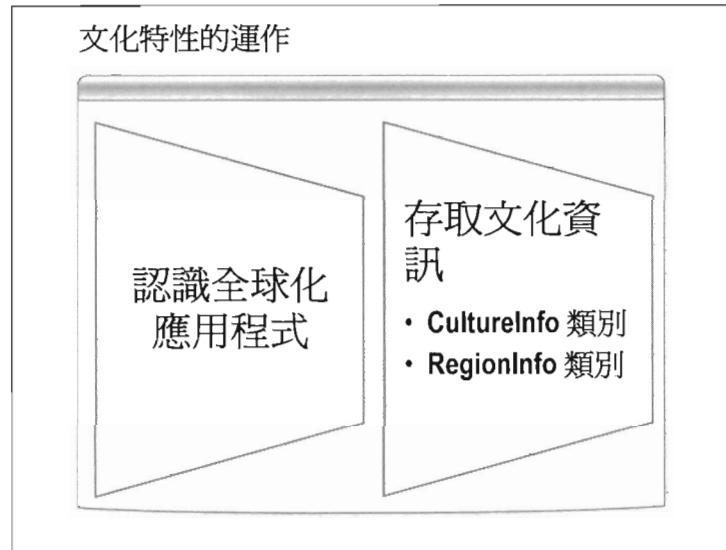


像 Microsoft Office 這類產品可以發行全球，除了它本身功能強大之外，最重要的是產品本身已經全球化，什麼是全球化？什麼是本地化？本章將會做一個介紹。

如何開發一個具全球化的應用程式？這是本章的重點，.NET Framework 提供了許多類別可以讓開發者快速的建置多國語系的系統以便讓應用程式更廣泛的被應用。善用.NET Framework 所提供開發世界性的應用程式所需的各種支援，這個章節將提供資訊，協助開發者設計和開發世界性的應用程式。

本章介紹以下主題：

- 文化特性的運作
- 根據文化特性格式化及排序
- 自訂文化特性

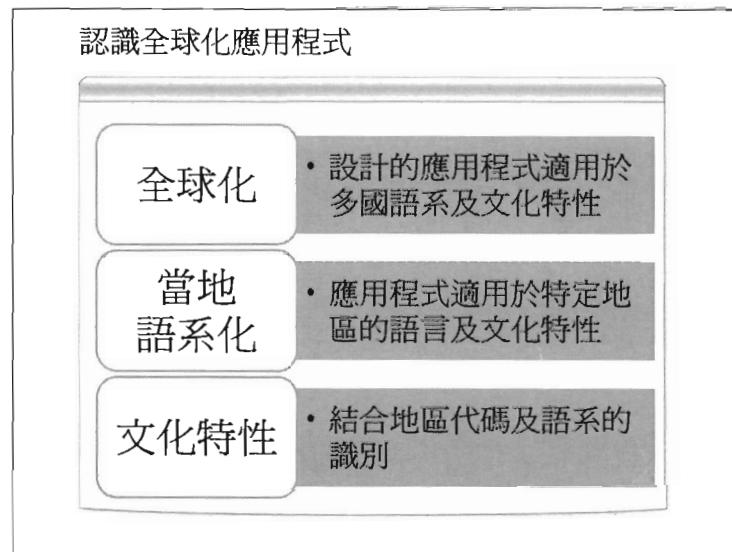


---

## 文化特性的運作

在這個一節中將介紹全球化、文化特性應用程式的概念，並了解如果在程式中存取文化資訊。

- 認識全球化應用程式
- 存取文化資訊
  - CultureInfo 類別
  - RegionInfo 類別



## 認識全球化應用程式

什麼是全球化 (Globalization) ?

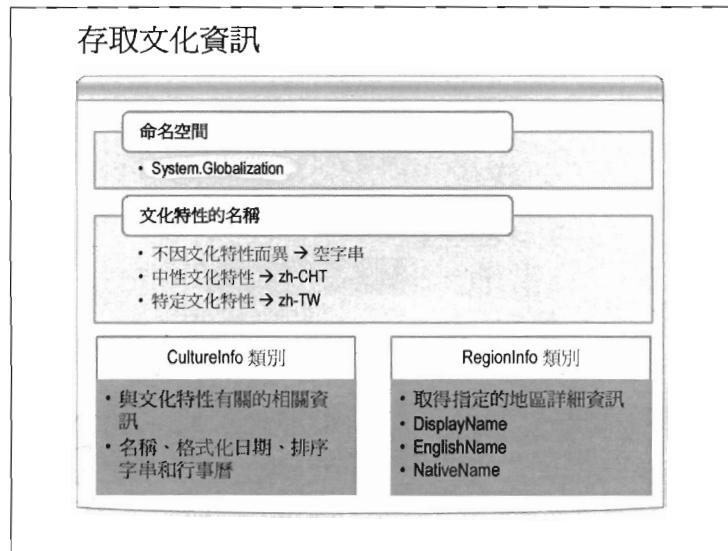
設計一個適用於多國語系及文化特性的應用程式，這個過程就叫應用程式全球化。其中包含支援使用者介面的語系當地化以及日期時間、數字的格式以及文字排序都必須符合當地的文化特性。

什麼是當地語系化 (Localization) ?

當地語系化可以轉換針對應用程式支援的所有文化特性，並且使用應用程式的資源 (如 Resource 檔中的資料) 來當作當地語系化版本的一個步驟。

什麼是文化特性 (Culture) ?

文化特性包含了日期時間、數字、貨幣...等格式的展現，還有文字排序、月曆。這些在不同的地區/國別的人有不同的使用習慣，文化特性將這些 (格式、排序、月曆) 與地區代碼及語系的識別結合。



## 存取文化資訊

開發全球化系統時，在.NET Framework 中所使用的相關類別全部放在 System.Globalization 命名空間 (Namespace)裡，其中包含定義文化特性相關資訊的類別，包括語言、國家/地區、使用中的曆法、日期、貨幣和數字格式模式，以及字串的排序順序。這些類別 (Class) 對撰寫全球化（國際化）的應用程式很有用。

例如：CultureInfo、StringInfo 和 TextInfo 之類的類別即提供進階的全球化功能，使用這些類別來簡化多國語言應用程式的開發程序。開發者可以將目前文化特性的 CultureInfo 物件取出使用或者是傳遞給自訂的 CultureInfo 物件來套用其特性定的規則或資料格式等。

### 文化特性的名稱

設計全球化系統主要要找到適當的文化特性來顯示適合的資料或是畫面，然而文化特性通常分為三組：

- 不因文化特性而異

不因文化特性而異就表示不區分文化特性。可以使用空字串 ("") 的名稱指定「不因文化特性而異」。

- 中性文化特性

中性文化特性是與語言相關聯的文化特性，但不與國別/區域相關聯。例如，"zh-CHT"(繁體中文)是中性文化特性

- 特定文化特性

特定文化特性是與語言和國別/區域相關聯的文化特性。例如，"ja" 是中性文化特性，而 "ja-JP" 是特定文化特性，另外要注意的是"zh-CHT" (繁體中文) 是中性文化特性。

## CultureInfo 類別

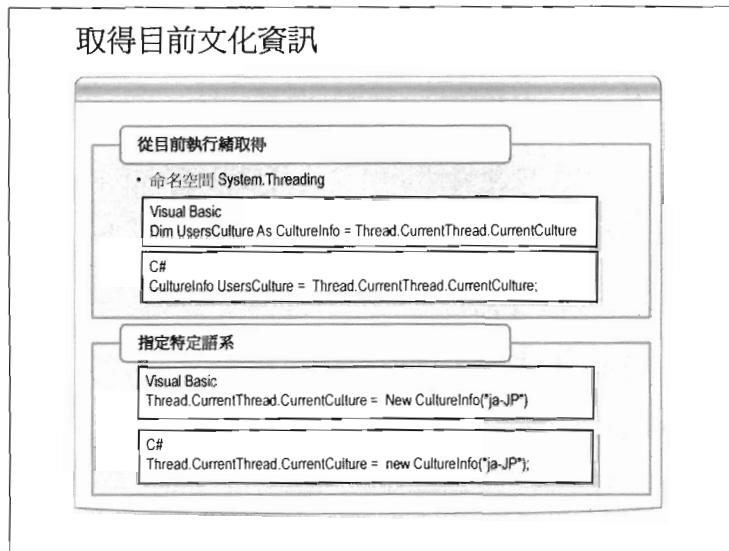
CultureInfo 類別具有特定文化特性的資訊，例如相關的語言、次語言 (Sublanguage)、國家/地區、曆法和文化等。

這個類別也提供一些屬性針對「特定文化特性」相關的類別存取，例如：DateTimeFormatInfo、NumberFormatInfo、CompareInfo 或 TextInfo 等執行個體存取。而這些物件則包含了設計應用程式時所會用到的特定文化特性資訊，例如：

- 大小寫的指定
- 日期和數字的格式化
- 字串的比較

## RegionInfo 類別

雖然 CultureInfo 提供了大部分的文化特性資料，但是一些有關區域性的資料，CultureInfo 並沒有直接提供，這時候就可以使用 RegionInfo 來提供有關區域性更詳細的資料。而 CultureInfo 類別有兩個屬性都可以建立 RegionInfo 物件，其中一個是使用一般名稱來建立，如 Name，另一個是使用 LCID 數值來識別。



## 取得使用者的文化特性

如果想要取得或設定目前執行緒的文化特性 (Culture) 的話，可以使用 System.Threading 命名空間下的 Thread 物件，Thread 物件中有一個 CurrentThread 屬性會收回目前執行緒的執行個體，接著再使用此執行緒的 CurrentCulture 屬性收回相對應的 CultureInfo 執行個體。

### 使用的命名空間

使用程式碼取得文化特性必須引用以下兩個命名空間：

- System.Globalization

包含多種全球化程式所需的類別，其中常見的 CultureInfo 類別便是設定某文化資訊的類別。

- System.Threading

System.Threading 命名空間則是包含一些關於執行緒等類別像是 Thread 類別。

下列範例程式碼使用 Thread 物件去取得目前執行緒的文化特性，並將其文化特性的名稱列出，撰寫程式前要先匯入 System.Globalization 和 System.Threading 這兩個命名空間：

```

Visual Basic
Dim UsersCulture As CultureInfo
UsersCulture = Thread.CurrentThread.CurrentCulture
Console.WriteLine("目前的文化特性：" + UsersCulture.Name)
Console.WriteLine("文化特性日期格式：" + DateTime.Now.ToString()
())
Console.WriteLine("文化特性貨幣格式：" + 1000.ToString("c"))

```

```

C#
CultureInfo UsersCulture =
    Thread.CurrentThread.CurrentCulture;
Console.WriteLine("目前的文化特性：" + UsersCulture.Name);
Console.WriteLine("文化特性日期格式：" + DateTime.Now.ToString()
());
Console.WriteLine("文化特性貨幣格式：" + 1000.ToString("c"));

```

## 指定特定語系

應用程式設計時可以同時表現多種語系的資訊，例如：銀行的貨幣兌換表格。那麼如果想要應用程式執行展示特定的語系資料，就不能使用自動設定的方式，此時可以透過建立 `CultureInfo` 物件，在其建構函式中指定某個特定語系的「特定文化特性」名稱。

指定的「特定文化特性」名稱，可以透過 `CultureInfo` 類別定義特定文化資訊，下表是關於 `CultureInfo` 類別的名稱、代碼、語系-國家/地區對照表(更多資訊請參考 MSDN Library [http://msdn2.microsoft.com/en-us/library/kx54z3k7\(en-US,VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/kx54z3k7(en-US,VS.80).aspx))：

名稱	代碼	語系-國家/地區
zh-TW	0x0404	Chinese - Taiwan
zh-CHS	0x0004	Chinese (Simplified)
en-US	0x0409	English - United States
ja-JP	0x0411	Japanese - Japan
ko-KR	0x0412	Korean - Korea

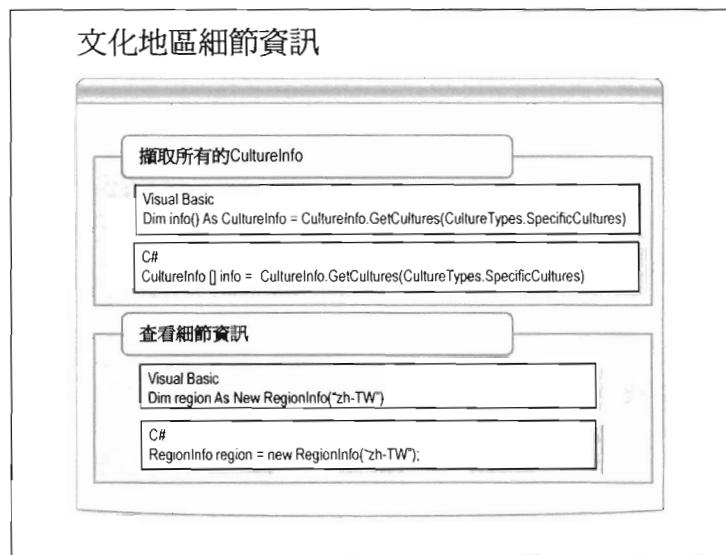
在程式中設定其他語系的方式其實不難，只要將目前執行緒的文化資訊指定為某地方語系即可。此範例將套用日本地區文化慣例，程式如下：

Visual Basic

```
Thread.CurrentThread.CurrentCulture = New CultureInfo("ja-JP")
```

C#

```
Thread.CurrentThread.CurrentCulture = new CultureInfo("ja-JP");
```



## 擷取所有的 CultureInfo

若要列出所有系統所支援的文化特性，則可使用 `CultureInfo` 物件的 `GetCultures` 方法來擷取文化特性 (Culture) 類型清單。在叫用 `GetCultures` 方法時，如果設定的參數不同將會回傳不同的資料：

- `types` 參數設定為 `SpecificCultures` 值

傳回所有特定文化特性。

- `types` 為 `NeutralCultures` 值

傳回所有中性文化特性和不變文化特性。

- `types` 設定為 `AllCultures` 值

傳回所有中性和特定文化特性、Windows 系統中安裝的文化特性，以及使用者建立的自訂文化特性。

`SpecificCultures` 是指具體的國家/地區專用的文化特性 (Culture)，例如，"zh-TW" (中文 - 台灣) 是一個特定文化特性。這些文化特性 (Culture) 型別值是由 `CultureInfo.CultureTypes` 屬性傳回，同時也充當篩選條件，限制 `CultureInfo.GetCultures` 方法所傳回的文化特性。

下列程式範例是設定參數為 SpecificCultures 值以取得傳回所有特定的文化特性。

#### Visual Basic

```
For Each UsersCulture As CultureInfo In _
    CultureInfo.GetCultures(CultureTypes.SpecificCultures)
    Console.WriteLine("Culture: " & UsersCulture.Name)
Next
```

#### C#

```
foreach (CultureInfo UsersCulture in
    CultureInfo.GetCultures(CultureTypes.SpecificCultures))
{
    Console.WriteLine("Culture: " + UsersCulture.Name);
}
```

## 查看地區的細節資訊

若要查看某地區的詳細名稱使用格式名稱等細節資訊，可以使用 RegionInfo 類別，它提供許多名稱的屬性，像是 DisplayName、EnglishName、CurrencyEnglishName... 等。

以下範例是取得目前的文化資訊，並使用 RegionInfo 取得詳細資訊：

#### Visual Basic

```
Dim UsersCulture As CultureInfo = Thread.CurrentThread.CurrentCulture
Dim region As New RegionInfo(UsersCulture.Name)
Console.WriteLine(region.DisplayName)
Console.WriteLine(region.EnglishName)
Console.WriteLine(region.NativeName)
Console.WriteLine(region.CurrencyEnglishName)
Console.WriteLine(region.CurrencyNativeName)
```

#### C#

```
CultureInfo UsersCulture = Thread.CurrentThread.CurrentCulture;
RegionInfo region = new RegionInfo(UsersCulture.Name);
Console.WriteLine(region.DisplayName);
Console.WriteLine(region.EnglishName);
Console.WriteLine(region.NativeName);
Console.WriteLine(region.CurrencyEnglishName );
Console.WriteLine(region.CurrencyNativeName);
```

### 練習4.1：列出所有的文化資訊

• 練習如何在Windows 應用程式列出所有的文化資訊，並可以查詢文化資訊的名稱與細節資訊。

• 預估實作時間：15分鐘

### 練習 4.1：列出所有的文化資訊

#### 目的：

練習如何在 Windows 應用程式列出所有的文化資訊，並可以查詢文化資訊的名稱與細節資訊。

#### 功能描述：

在這個練習中會在 Windows 應用程式的 ListBox 顯示所有的文化資訊，並在使用者選取項目時在 TextBox 顯示選取的地區名稱及細節資訊。

#### 預估實作時間：15分鐘

#### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod04\_1。
3. 在 Form 放上一個 ListBox 及一個 TextBox。

4. 先必要的匯入命名空間：

```
Visual Basic
Imports System.Globalization
Imports System.Text
```

```
C#
using System.Globalization;
```

5. 在表單載入時將所有地區名稱列於 ListBox 控制清單中：

```
Visual Basic
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    For Each info In CultureInfo.GetCultures(CultureTypes.SpecificCultures)
        ListBox1.Items.Add(info)
    Next
End Sub
```

```
C#
private void Form1_Load(object sender, EventArgs e)
{
    foreach (CultureInfo info in CultureInfo.GetCultures(CultureTypes.SpecificCultures))
    {
        ListBox1.Items.Add(info);
    }
}
```

6. 在使用者選取項目時，在 TextBox 顯示選取的地區名稱及細節資訊：

```
Visual Basic
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged
    Dim selectedCulture As CultureInfo
    selectedCulture = CType(ListBox1.SelectedItem, CultureInfo)
    Dim region As New RegionInfo(selectedCulture.Name)

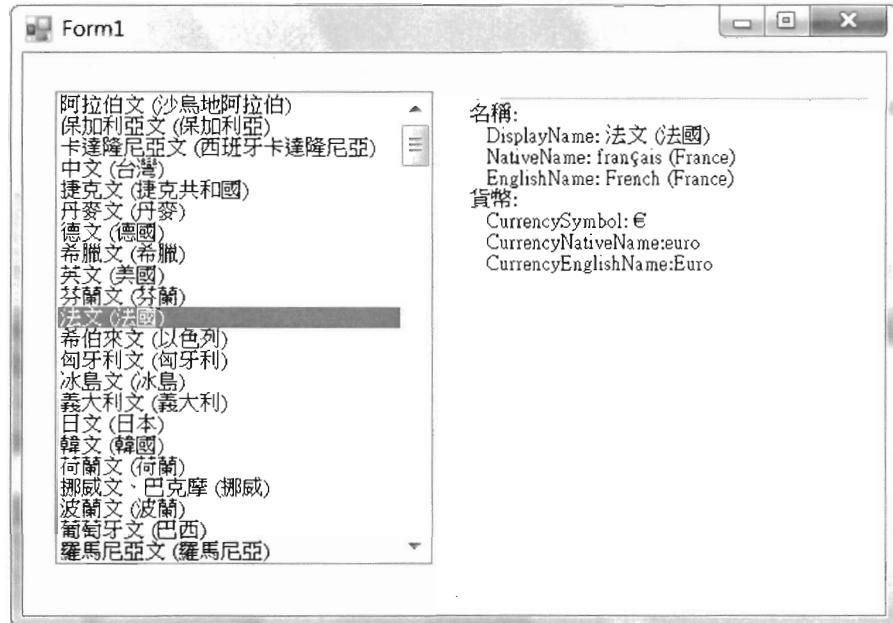
    Dim sb As New StringBuilder
    sb.AppendLine("名稱:")
    sb.AppendLine(" DisplayName: " & selectedCulture.DisplayName)
    sb.AppendLine(" NativeName: " & selectedCulture.NativeName)
    sb.AppendLine(" EnglishName: " & selectedCulture.EnglishName)
    sb.AppendLine(" 貨幣:")
```

```
sb.AppendLine(" CurrencySymbol:" & region.CurrencySymbol)
sb.AppendLine(" CurrencyNativeName:" & region.CurrencyNativeName)
sb.AppendLine(" CurrencyEnglishName:" & region.CurrencyEnglishName)
TextBox1.Text = sb.ToString()
End Sub
```

```
C#
private void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    CultureInfo selectedCulture ;
    selectedCulture = ListBox1.SelectedItem as CultureInfo;
    RegionInfo region = new RegionInfo(selectedCulture.Name);

    StringBuilder sb = new StringBuilder();
    sb.AppendLine("名稱:");
    sb.AppendLine(" DisplayName: " + selectedCulture.DisplayName);
    sb.AppendLine(" NativeName: " + selectedCulture.NativeName);
    sb.AppendLine(" EnglishName: " + selectedCulture.EnglishName);
    sb.AppendLine("貨幣:");
    sb.AppendLine(" CurrencySymbol:" + region.CurrencySymbol);
    sb.AppendLine(" CurrencyNativeName:" + region.CurrencyNativeName);
    sb.AppendLine(" CurrencyEnglishName:" + region.CurrencyEnglishName);
    TextBox1.Text = sb.ToString();
}
```

7. 執行結果如下：



根據文化特性進行...

數字的格式化

日期時間的格式化

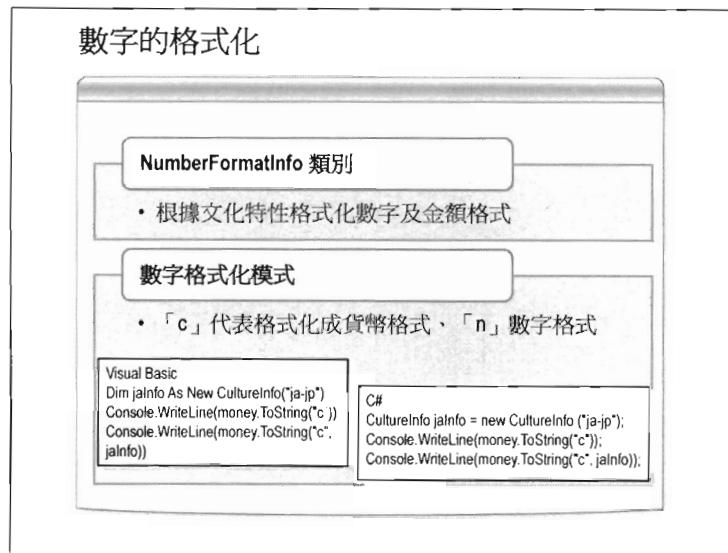
文字比較

## 根據文化特性進行...

CultureInfo 類別除了提供國家/地區名稱的識別之外，最主要它還包含了 NumberFormat、DateTimeFormat，CompareInfo 等屬性。這些屬性掌管著這個文化特性的數字、貨幣、日期、時間的顯示格式以及字串比較和排序的方式。

這些格式及比較方式並不是固定的，如果有需要的話，可以在程式中去修改它們。這一節將介紹如何去處理這些變更的程序。

- 數字的格式化
- 日期時間的格式化
- 文字比較



## 數字的格式化

你可以透過 `ToString` 方法傳入格式化的模式，像是「`c`」代表格式化成貨幣格式、「`e`」科學符號、「`n`」數字格式。範例：

```

Visual Basic
Console.WriteLine("金額:" + money.ToString("c"))
Console.WriteLine("數字:" + money.ToString("n"))

```

```

C#
Console.WriteLine ("金額:" + money.ToString("c"));
Console.WriteLine("數字:" + money.ToString("n"));

```

數值轉字串使用 `ToString` 方法，相反的字串轉數值，則是使用數值型別 (`Int32`、`Decimal`、`Double`...等) `Parse` 方法或者 `TryParse` 方法，這兩個方法預設都只能轉標準的數字字串，如果字串中包含貨幣符號或者千位符號將無法順利轉型成功。可以利用 `NumberStyles` 列舉常數指定數字的格式，如以下範例：

```

Visual Basic
Dim strmoney As String = "NT$109874.3753"
Dim money As Decimal
If Not Decimal.TryParse(strmoney, NumberStyles.Currency, Nothing)

```

```

g, money) Then
    money = 52389.1053
End If

```

```

C#
string strmoney = "NT$109874.3753";
decimal money;
if (!decimal.TryParse(strmoney, NumberStyles.Currency,null, out
money)) {
    money = 52389.1053M;
}

```

## 使用 **NumberFormatInfo** 類別

預設數字及貨幣的格式會根據目前「控制台」的「地區語言選項」設定顯示，如果想修改的話，除了使用格式化字串之外，也可以修改 **CultureInfo** 的 **NumberFormat**。如下範例：

```

Visual Basic
Dim jaInfo As New CultureInfo("ja-jp")
Console.WriteLine("金額:" + money.ToString("c", jaInfo));
Console.WriteLine("數字:" + money.ToString("n", jaInfo));

jaInfo.NumberFormat.CurrencyDecimalDigits = 4;
jaInfo.NumberFormat.NumberDecimalDigits = 4;
Console.WriteLine("金額:" + money.ToString("c", jaInfo));
Console.WriteLine("數字:" + money.ToString("n", jaInfo));

```

```

C#
CultureInfo jaInfo = new CultureInfo ("ja-jp");
Console.WriteLine("金額:" + money.ToString("c", jaInfo ));
Console.WriteLine("數字:" + money.ToString("n", jaInfo ));

jaInfo.NumberFormat.CurrencyDecimalDigits = 4;
jaInfo.NumberFormat.NumberDecimalDigits = 4;
Console.WriteLine("金額:" + money.ToString("c", jaInfo));
Console.WriteLine("數字:" + money.ToString("n", jaInfo));

```



## 日期時間的格式化

你可以透過 `ToString` 方法傳入格式化的模式，像是「`d`」代表輸出短日期格式、「`D`」輸出長日期格式、「`t`」輸出短時間格式、「`T`」輸出長時間格式、「`f`」輸出完整日期時間的短格式、「`F`」輸出完整日期時間的長格式。範例：

```

Visual Basic
Console.WriteLine(Now.ToString("F"))

```

```

C#
DateTime now = DateTime.Now;
Console.WriteLine(now.ToString("F"));

```

## 使用 `DateTimeFormatInfo`

預設日期時的格式會根據目前「控制台」的「地區語言選項」設定顯示，如果想修改的話，除了使用格式化字串之外，也可以修改 `CultureInfo` 的 `DateTimeFormat` 屬性。如下範例：

```

Visual Basic
Dim ci As New CultureInfo("zh-HK", False)
Console.WriteLine(ci.DateTimeFormat.ShortDatePattern)

```

```
Console.WriteLine(Now.ToString("d", ci))
```

```
C#
CultureInfo ci = new CultureInfo("zh-HK", false);
Console.WriteLine(ci.DateTimeFormat.ShortDatePattern);
Console.WriteLine(now.ToString("d", ci));
Console.WriteLine(now.ToString(ci.DateTimeFormat.ShortDatePattern));
```

以下範例是修改星期的名稱：

```
Visual Basic
For Each d In ci.DateTimeFormat.DayNames
    Console.WriteLine(d)
Next
Console.WriteLine(DateTime.Now.ToString("F", ci))
Console.WriteLine(ci.DateTimeFormat.GetDayName(DateTime.Now.DayOfWeek))

ci.DateTimeFormat.DayNames = New String() {"a", "b", "c", "d", "e", "f", "g"}
Console.WriteLine("星期:" + ci.DateTimeFormat.GetDayName(DateTime.Now.DayOfWeek))
Console.WriteLine(DateTime.Now.ToString("F", ci))
```

```
C#
foreach (string d in ci.DateTimeFormat.DayNames)
{
    Console.WriteLine(d);
}
ci.DateTimeFormat.DayNames = new string[] { "a", "b", "c", "d", "e", "f", "g" };
Console.WriteLine(DateTime.Now.ToString("F", ci));
Console.WriteLine(ci.DateTimeFormat.GetDayName(DateTime.Now.DayOfWeek));

ci.DateTimeFormat.DayNames = new string[] { "a", "b", "c", "d", "e", "f", "g" };

Console.WriteLine("星期:" + ci.DateTimeFormat.GetDayName(DateTime.Now.DayOfWeek));
Console.WriteLine(DateTime.Now.ToString("F", ci));
```



## 文字比較

除了數字、日期時間之外，各地區的文字排序及比較方式也略有不同。這個部份可以透過 CultureInfo 的 CompareInfo 屬性取得 CompareInfo 物件進行文化特性的文字比對。CompareOptions 可以指定比較時是否忽略大寫小寫空白字元寬度...等。

以下範例是比較數字 123 及全型數字 1 2 3，預設這兩個比較是不相等，經過指定 CompareOptions 指定為 IgnoreWidth 忽略字元寬度，這兩個字串的比較變成相等。

```

Visual Basic
Dim ci As New CultureInfo("zh-TW")
Dim s1 As String = "123"
Dim s2 As String = "1 2 3"
Dim compStr As CompareInfo = ci.CompareInfo
Console.WriteLine("{0} {1} {2}", s1,
If(s1.CompareTo(s2) = 0, "=", "!="), s2)

Console.WriteLine("{0} {1} {2}", s1,
If(compStr.Compare(s1, s2, CompareOptions.IgnoreWidth) =
0, "=", "!="), s2)

```

C#

```
CultureInfo ci = new CultureInfo("zh-TW");

string s1 = "123";
string s2 = "1 2 3 ";
CompareInfo compStr = ci.CompareInfo;
Console.WriteLine("{0} {1} {2}", s1,
    (s1.CompareTo(s2)==0?"=":"!="), s2);

Console.WriteLine("{0} {1} {2}", s1,
    (compStr.Compare(s1, s2, CompareOptions.IgnoreWidth)==0?
    "=":"!="),s2);
```

### 練習4.2：根據文化特性格式化

•這個練習要顯示台灣地區的數字及日期時間格式。

•預估實作時間：10分鐘

### 練習 4.2：列出所有的文化資訊

目的：

練習如何這個練習要顯示台灣地區的數字及日期時間格式。

功能描述：

在這個練習中會在 Windows 應用程式的 ListBox 顯示台灣地區的數字及日期時間。

預估實作時間：10 分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod04\_2。
3. 在 Form 放上一個 ListBox 及兩個 Button。
4. 先匯入必要的命名空間：

```
Visual Basic
Imports System.Globalization
```

```
C#
using System.Globalization;
```

5. 在 Button1 處理日期：

```
Visual Basic
Dim price As Decimal = -32800
Dim numInfo As NumberFormatInfo = New CultureInfo("zh-TW").NumberFormat
numInfo.CurrencyNegativePattern = 1
listBox1.Items.Add(price)
```

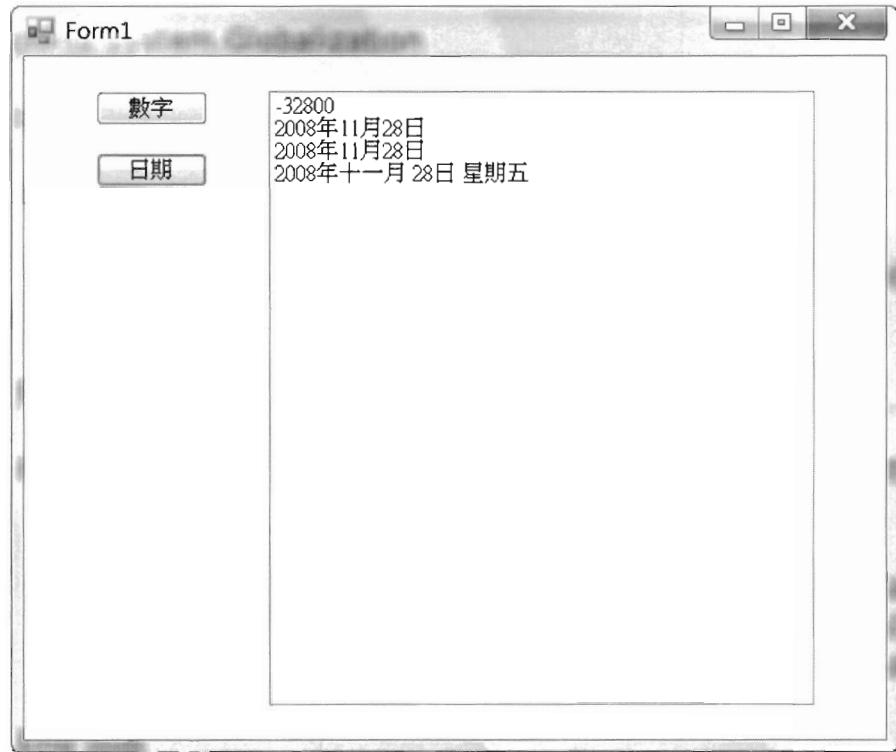
```
C#
decimal price = -32800;
NumberFormatInfo numInfo = new CultureInfo("zh-TW").NumberFormat;
numInfo.CurrencyNegativePattern = 1; → - $n.
listBox1.Items.Add(price); → listBox1.Items.Add(price) ← a.ToString("c", numInfo);
```

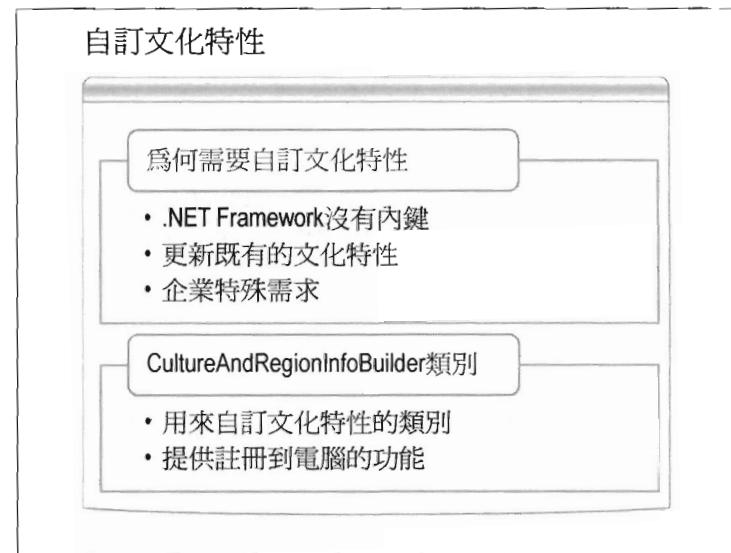
6. 在 Button2 處理日期：

```
Visual Basic
Dim ci As New CultureInfo("zh-TW", False)
listBox1.Items.Add(Now.ToString("D", ci))
listBox1.Items.Add(Now.ToString(ci.DateTimeFormat.LongDatePattern))
ci.DateTimeFormat.LongDatePattern = "yyyy年MM月dd日 dddd"
listBox1.Items.Add(Now.ToString(ci.DateTimeFormat.LongDatePattern))
```

```
C#
DateTime now = DateTime.Now;
CultureInfo ci = new CultureInfo("zh-TW", false);
listBox1.Items.Add(now.ToString("D", ci));
listBox1.Items.Add(now.ToString(ci.DateTimeFormat.LongDatePattern));
ci.DateTimeFormat.LongDatePattern = "yyyy年MM月dd日 dddd";
listBox1.Items.Add(now.ToString(ci.DateTimeFormat.LongDatePattern));
```

7. 執行結果：

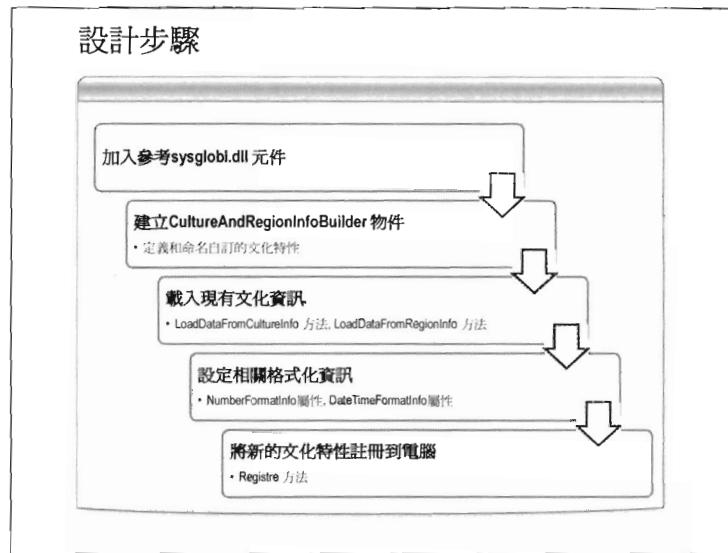




## 自訂文化特性

由.NET Framework 和 Windows 作業系統提供的預先定義文化特性，提供了一般應用程式常用的資料，如國家/區域使用的語言，以及用來格式化和比較字串、日期及數字的文字特性。不過，如果預先定義的文化特性不能提供應用程式所需要的資訊，此時開發者就可以建立自訂的文化特性。

然而，應用程式只可使用在 CultureInfo 類別中預先定義的文化特性，所以如果要使用自訂的文化特性，就必須先使用 CultureAndRegionInfoBuilder 類別建立新的自訂文化特性或覆寫預先定義的文化特性。



## 設計步驟

首先，定義和建立自訂文化特性的第一個步驟為：

使用 CultureAndRegionInfoBuilder 定義和命名自訂的文化特性，程式碼如下：

```

Visual Basic
Dim DemoBuilder As New _
CultureAndRegionInfoBuilder("zh-TW-uuu", _
CultureAndRegionModifiers.None)
  
```

```

C#
CultureAndRegionInfoBuilder DemoBuilder =
new CultureAndRegionInfoBuilder("zh-TW-uuu",
CultureAndRegionModifiers.None);
  
```

## 載入現有的 **CultureInfo** 物件和 **RegionInfo** 物件

定義和建立自訂文化特性的第二個步驟為載入開發者想使用的 CultureInfo 物件和 RegionInfo 物件，要達到此目標可以：：使用 CultureAndRegionInfoBuilder 物件的方法：

- 使用 LoadDataFromCultureInfo 方法

載入現有的 CultureInfo 物件。

- 使用 LoadDataFromRegionInfo 方法

載入現有的 RegionInfo 物件。

下列程式碼的功能為載入台灣文化特性的 CultureInfo 物件和 RegionInfo 物件。

Visual Basic

```
Dim TWCulture As New CultureInfo("zh-TW")
Dim TWRegion As New RegionInfo("zh-TW")
DemoBuilder.LoadDataFromCultureInfo(TWCulture)
DemoBuilder.LoadDataFromRegionInfo(TWRegion)
```

C#

```
CultureInfo TWCulture = new CultureInfo("zh-TW");
RegionInfo TWRegion = new RegionInfo("zh-TW ");
DemoBuilder.LoadDataFromCultureInfo(TWCulture);
DemoBuilder.LoadDataFromRegionInfo(TWRegion);
```



### 練習 4.3：建立自訂文化特性

**目的：**

練習如何在應用程式中建立自訂的文化特性，並把自訂的文化特性註冊到系統上，並且顯示自訂的字串、日期及數字等格式。

**功能描述：**

透過客製化的方式自訂一個文化特性叫做「zh-TW-uuu」，並將此文化特性註冊到機器上，讓此文化特性擁有自己的貨幣顯示格式，以供延伸設計使用。

**預估實作時間：20 分鐘**

**實作步驟：**

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 從「File」→「New Project」→選擇「Console Application」→在 Name 的位置輸入「Mod04\_3」，選用適當的程式語言 (Language)，Visual Basic 或 C#。

3. 首先，在 Mod04\_3 專案按右鍵，選加入參考→選 sysglobl.dll，按下確定。
4. VB 在 Module1.vb 類別的最上方(C#在 Program.cs)加上必要的 System.Globalization 和 System.Threading 命名空間。

```
Visual Basic
Imports System.Globalization
Imports System.Threading
```

```
C#
using System.Globalization;
using System.Threading;
```

5. 在程式進入點 Main 方法中使用 CultureAndRegionInfoBuilder 物件搭配 CultureInfo 和 RegionInfo 物件建立一個自訂的文化特性，並且分別使用 LoadDataFromCultureInfo 和 LoadDataFromRegionInfo 來載入 CultureInfo 和 RegionInfo 物件，程式碼如下。

```
Visual Basic
Sub Main()
    Dim DemoBuilder As New CultureAndRegionInfoBuilder("zh-TW-uuu", CultureAndRegionModifiers.None)

    Dim TWCulture As New CultureInfo("zh-TW")
    Dim TWRegion As New RegionInfo("zh-TW")
    DemoBuilder.LoadDataFromCultureInfo(TWCulture)
    DemoBuilder.LoadDataFromRegionInfo(TWRegion)
End Sub
```

```
C#
static void Main(string[] args)
{
    CultureAndRegionInfoBuilder DemoBuilder =
        new CultureAndRegionInfoBuilder("zh-TW-uuu", CultureAndRegionModifiers.None);

    CultureInfo TWCulture = new CultureInfo("zh-TW");
    RegionInfo TWRegion = new RegionInfo("zh-TW");
    DemoBuilder.LoadDataFromCultureInfo(TWCulture);
    DemoBuilder.LoadDataFromRegionInfo(TWRegion);
}
```

6. 在 Main 中繼續建立 NumberFormatInfo 物件以設定文化特性的數值顯示方式，程式碼如下。

Visual Basic

```
DemoBuilder.NumberFormat.CurrencySymbol = "@"
DemoBuilder.NumberFormat.CurrencyDecimalDigits = 4
```

C#

```
DemoBuilder.NumberFormat.CurrencySymbol = "@";
DemoBuilder.NumberFormat.CurrencyDecimalDigits = 4;
```

7. 建立 `DateTimeFormatInfo` 物件以設定文化特性的日期顯示方式，程式碼如下。

Visual Basic

```
DemoBuilder.GregorianDateFormat.DateSeparator = "."
```

C#

```
DemoBuilder.GregorianDateFormat.DateSeparator = ".";
```

8. 將自訂的文化註冊到電腦上使用，透過叫用 `CultureAndRegionInfoBuilder` 物件的 `Register` 方法來註冊。

緊接著透過程式將目前執行緒中的文化特性指定為先前所自訂的文化特性，並且將自定的日期以及貨幣的格式顯示出來，程式碼如下。

Visual Basic

Try

'將自訂的文化註冊到電腦上使用

`DemoBuilder.Register()`

```
Thread.CurrentThread.CurrentCulture = New CultureInfo("zh-TW-uuu")
```

```
Console.WriteLine("目前的文化特性:" + Thread.CurrentThread.CurrentCulture.Name)
```

```
Console.WriteLine("文化特性日期格式:" + DateTime.Now.ToString())
```

```
Console.WriteLine("文化特性貨幣格式:" + 1000.ToString("c"))
```

Catch

Finally

```
CultureAndRegionInfoBuilder.Unregister("zh-TW-uuu")
```

End Try

Console.Read()

C#

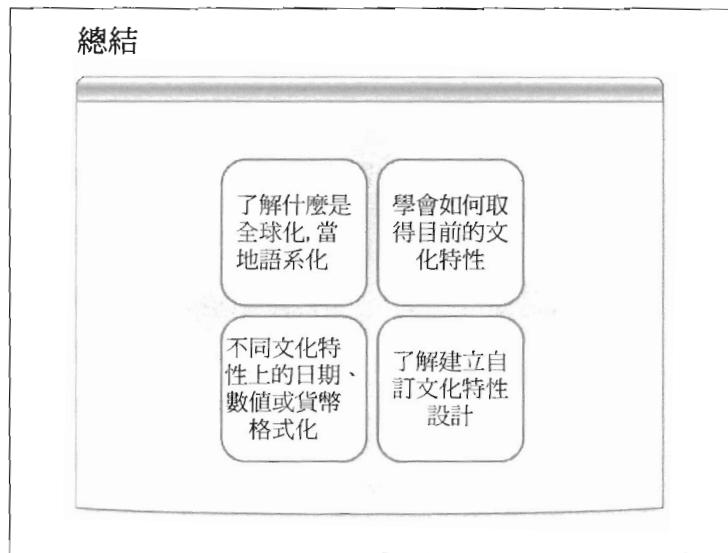
try

```
{  
    //將自訂的文化註冊到電腦上使用  
    DemoBuilder.Register();  
    //透過程式指定語系  
    Thread.CurrentThread.CurrentCulture = new CultureInfo("zh-TW-uuu");  
    Console.WriteLine("目前的文化特性：" + Thread.CurrentThread.CurrentCulture.Name);  
    Console.WriteLine("文化特性日期格式：" + DateTime.Now.ToString());  
    Console.WriteLine("文化特性貨幣格式：" + 1000.ToString("c"));  
}  
catch { }  
finally {  
    CultureAndRegionInfoBuilder.Unregister("zh-TW-uuu");  
}  
Console.Read();
```

9. 建置專案後直接執行其 Exe 執行檔或是按下 F5 執行，將會發現其文化特性以期日期格式和貨幣格式都改變了。



10. 結束程式執行。



## 總結

在一般的應用程式中如果要設計全球化的系統，可以使用.NET Framework所提供的相關類別，讓開發者在設計系統或應用程式時可以節省更多的成本，達到更強大的功能。

.NET Framework 包含了許多定義文化特性和關資訊的類別，包括語言、國家/地區、使用中的曆法、日期、貨幣和數字格式模式，以及字串的排序順序。這些類別 (Class) 對撰寫全球化 (國際化) 的應用程式很有用。

在這個章節所學到的設計技巧很多都是.NET Framework 針對多國語言系統所強化或新增的設計，透過這些設計技巧相信能讓我們的系統擁有更大的彈性設計。

# 第五章：序列化處理

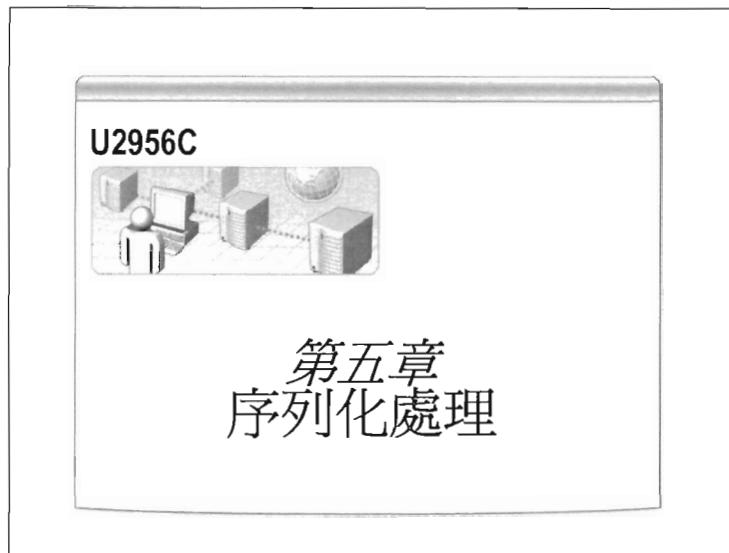
## 本章大綱

序列化的處理概念 .....	4
序列化的格式 .....	6
Binary 序列化 .....	8
SOAP 序列化 .....	12
練習 5.1：使用 Binary 序列化方法保存物件 .....	15
XML 序列化 .....	19
練習 5.2：將 XML 字串序列化/反序列化 .....	22
使用 Attribute 控制 XML 序列化 .....	26
練習 5.3：處理集合及物件繼承的序列化問題 .....	32
自訂序列化 .....	37
使用 ISerializable 介面 .....	38
使用 OnXXX Attribute 處理自訂序列化 .....	40
練習 5.4：使用 OnXXXAttribute .....	41

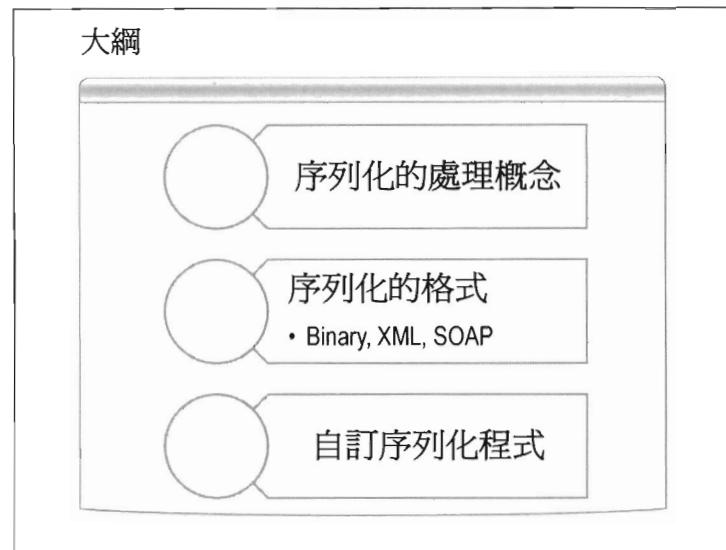


作者：

羅慧真



---

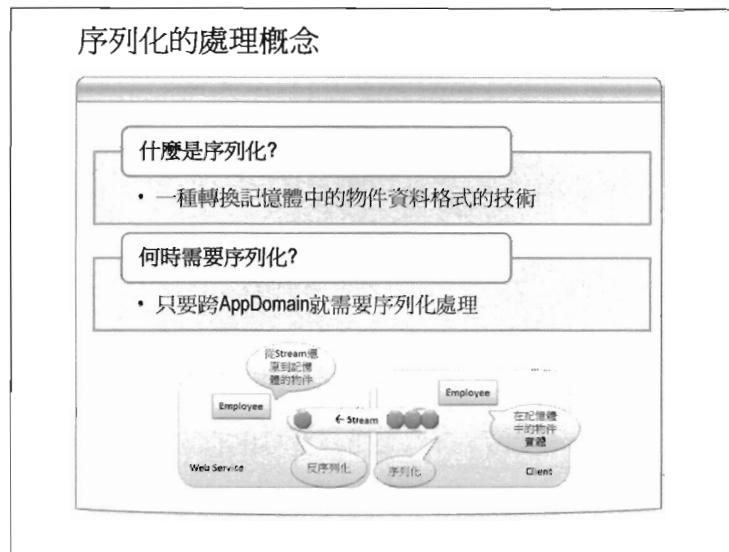


---

在這個章節中將介紹如何利用.NET Framework 提供的序列化技術，你將了解為何需要進行序列化處理，如何進行序列化...以及如何自訂序列化的程式。

本章介紹以下主題：

- 序列化的處理概念
- 序列化的格式
  - Binary
  - SOAP
  - XML
- 自訂序列化程式



## 序列化的處理概念

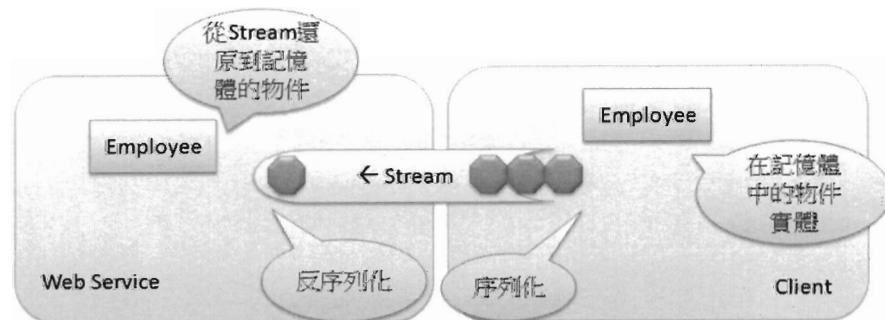
序列化是一種轉換記憶體中的物件資料格式的技術，目的在於將物件資料轉換成可以保存於檔案或其他儲存位置的技術，方便未來將物件資料保存或傳送。

### 何時需要序列化？

並不是所有的程式都需要進行序列化處理。

一般的程式呼叫方法傳遞物件因為都是在同一個記憶空間，所以直接傳遞物件是不需要進行轉換記憶體的動作，也就是序列化。

但是當程式跨越不同記憶體空間，像是呼叫 Web Service 的方法，將本地物件傳到 Web Service 時就必須進行序列化的動作。



不同記憶體空間最小單位是 AppDomain，如果程式只要是跨越 AppDomain 溝通，就需要序列化的動作。

### 何謂序列化與反序列化

序列化技術同時也包含反序列化方法，當物件資料序列化成資料串流 (Stream) 格式，藉由網路傳送到其他位置時，需使用反序列化的方法將之還原成物件方能繼續使用物件。

序列化的格式		
<b>.NET內建的三種序列化格式</b>		
序列化種類	儲存格式	特性
Binary	二進位格式	保存物件的完整型別
SOAP	XML 格式, SOAP 標準	保存物件的完整型別
XML	XML 格式	只保存物件當中 Public 的成員資料與成員屬性

## 序列化的格式

.NET 支援三種序列化技術，Binary Serialization 能將物件資料轉換化成二進位格式，可完整保留物件狀態；XML Serialization 只保存物件當中 Public 的成員資料與成員屬性，並轉換成 XML 格式。SOAP Serialization，也是 XMP 格式，不過它是 SOAP 標準，而且可以完整保留物件狀態。

### Binary Serialization 技術

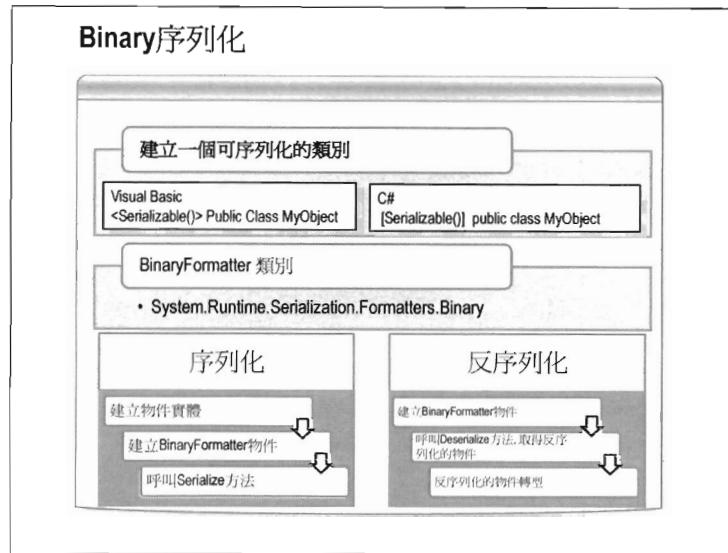
Binary Serialization 技術提供物件的完整型別保存，通常用在跨不同應用程式之間的物件交換，例如：像剪貼簿一樣的功能，可將物件序列化之後放進剪貼簿，就算原應用程式關閉，還是能夠從剪貼簿反序列化到另外一個應用程式使用。

將物件序列化之後可以轉換到資料串流、硬碟或記憶體，透過網路便能輕鬆地在各個應用程式之間交換資料。像 Web Service、Remoting 技術就是依靠序列化技術完成遠端呼叫的資料傳遞。

## XML 與 SOAP Serialization 技術

XML Serialization 只保存物件當中 Public 的成員資料與成員屬性，不保證型別完整性，通常應用在單純的資料保存及傳遞，且由於序列化後的資料格式為 XML，而 XML 是一個開放的標準，所以極為適合用於現今的網際網路中。

但是卻因為 XML 格式完全開放自行定義，會導致不相容於所有應用程式的情況發生，而 SOAP 標準就成為應用程式之間 XML 資料格式的標準了，讓物件資料更具透通性。



## Binary 序列化

Binary 序列化保存物件 Public 與 Private 的成員資料，並轉換成資料串流，也就是一連串的 Byte，可以將之保存於記憶體中 (MemoryStream)、檔案中(FileStream)或藉由網路傳送(NetworkStream) 等等。

而經由反序化動作也能從這些不同類型的資料串流，讀取並重新建立還原物件的原貌，這感覺其實很像星際大戰電影中的人物傳送，先將人物序列化為分子，然後在異地把分子重組還原人物。

### 建立一個可序列化的類別

欲使用 Binary 序列化技術的類別，必須先加上 Serializable Attribute，表示此類別可被序列化，編譯時編譯器便會協助產生查詢此類別相關介面的程式，範例如下：

#### Visual Basic

```
<Serializable()>
Public Class MyObject
    Private strName As String = "Jerry"
    Public intAge As Integer = 10
```

```

<NonSerialized()> _
Public dateBirthday As DateTime = DateTime.Now

Public Function GetName() As String
    Return strName
End Function
End Class

```

C#

```

[Serializable()]
public class MyObject
{
    private string strName = "Jerry";
    public int intAge = 10;
    [NonSerialized()]
    public DateTime dateBirthday = DateTime.Now;

    public string GetName(){
        return strName;
    }
}

```

範例中 NonSerialized Attribute 可以指定某個成員資料不列入序列化與反序列化動作中。

## 序列化物件

設計好可序列化的類別之後，在主程式中先引用相關名稱空間，如下：

```

Visual Basic
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.IO

```

```

C#
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

```

接下來建立實體物件並使用，然後使用建立 BinaryFormatter 物件並呼叫 Serialize 方法將實體物件序列化成檔案，範例如下：

**Visual Basic**

```
'1, 建立及使用物件
Dim obj As New MyObject
bj.intAge = 18
'2, 將物件序列化成Binary檔案
Dim formatter As New BinaryFormatter()
Dim fs As New FileStream(Application.StartupPath & "\MyObject.bi
n", FileMode.Create)
formatter.Serialize(fs, obj)
fs.Close()
```

**C#**

```
//1, 建立及使用物件
MyObject obj = new MyObject();
obj.intAge = 18;
//2, 將物件序列化成Binary檔案
BinaryFormatter formatter = new BinaryFormatter();
FileStream fs = new FileStream(Application.StartupPath + "\\MyO
bject.bin", FileMode.Create);
formatter.Serialize(fs, obj);
fs.Close();
```

執行完畢後可以檢視 MyObject.bin 檔案是否產生。

## 反序列化物件

反過來，從檔案還原物件可以透過 Deserialize 方法完成，然後取出物件資料以證明還原成功，範例如下：

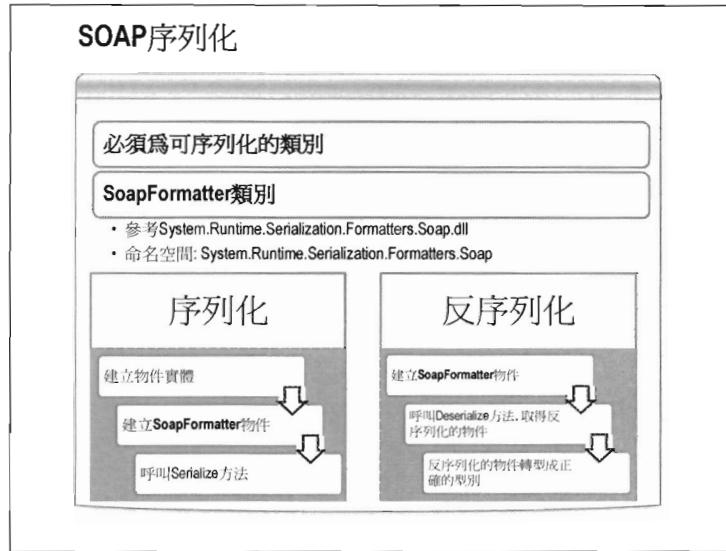
**Visual Basic**

```
'從Binary檔案反序列化回原物件
Dim formatter As New BinaryFormatter()
Dim fs As New FileStream(Application.StartupPath & "\MyObject.bi
n", FileMode.Open)
Dim obj As MyObject = CType(formatter.Deserialize(fs), MyObjec
t)
fs.Close()
MessageBox.Show("Deserialize OK...Private strName = " & obj.Get
Name())
```

**C#**

```
//從Binary檔案反序列化回原物件
BinaryFormatter formatter = new BinaryFormatter();
FileStream fs = new FileStream(Application.StartupPath + "\\MyO
bject.bin", FileMode.Open);
MyObject obj = (MyObject)formatter.Deserialize(fs);
```

```
fs.Close();
MessageBox.Show("Deserialize OK...Private strName = " + obj.Ge
tName());
```



## SOAP 序列化

SOAP(Simple Object Access Protocol)標準定義了基本物件型別的 XML 描述方法，也定義了遠端程序呼叫的標準。

### 建立一個可序列化的類別

欲使用 SOAP 序列化技術的類別，必須先加上 Serializable Attribute，表示此類別可被序列化，編譯時編譯器便會協助產生查詢此類別相關介面的程式。

### SOAP 序列化物件

SOAP 序列化綜合 XML 序列化與 Binary 序列化的兩個重要特性，首先格式為 SOAP 標準 XML，具備跨網際網路的特性，對於 Private 資料也能夠保存；另外，類別也必須加入 Serializable Attribute，標示為可序列化類別。

實作測試程式時必須在工具中參考 System.Runtime.Serialization.Formatters.Soap.dll，然後在主程式中匯入相關命名空間，如下：

```
Visual Basic
Imports System.Runtime.Serialization.Formatters.Soap
```

```
Imports System.IO
```

C#

```
using System.Runtime.Serialization.Formatters.Soap;
using System.IO;
```

接下來建立實體物件並使用，然後使用建立 SoapFormatter 物件並呼叫 Serialize 方法將實體物件序列化成檔案，與前面介紹的 Binary 序列化方法類似，範例如下：

Visual Basic

```
'1, 建立及使用物件
Dim obj As New MyObject
obj.intAge = 18
'2, 將物件序列化成Soap Xml檔案
Dim formatter As New SoapFormatter()
Dim fs As New FileStream(Application.StartupPath & "\MyObject_Soap.xml", FileMode.Create)
formatter.Serialize(fs, obj)
fs.Close()
```

C#

```
//1, 建立及使用物件
MyObject obj = new MyObject();
obj.intAge = 18;
//2, 將物件序列化成Soap Xml檔案
SoapFormatter formatter = new SoapFormatter();
FileStream fs = new FileStream(Application.StartupPath + "\\MyObject_Soap.xml", FileMode.Create);
formatter.Serialize(fs, obj);
fs.Close();
```

執行完畢後可以檢視 MyObject\_Soap.xml 檔案是否產生，內容應該如下：

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```

<SOAP-ENV:Body>
<a1:MyObject id="ref-1" xmlns:a1="http://schemas.microsoft.co
m/clr/nsassem/U2956Demo/U2956Demo%2C%20Version%3D1.0.
0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull"
>
<strName id="ref-3">Jerry</strName>
<intAge>18</intAge>
</a1:MyObject>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

## 反序列化物件

反過來，從檔案還原物件也是透過 Deserialize 方法完成，然後取出物件資料以證明還原成功，範例如下：

### Visual Basic

```

'從Soap Xml檔案檔案反序列化回原物件
Dim formatter As New SoapFormatter()
Dim fs As New FileStream(Application.StartupPath & "\MyObject_S
oap.xml", FileMode.Open)
Dim obj As MyObject = CType(formatter.Deserialize(fs), MyObjec
t)
fs.Close()
MessageBox.Show("Deserialize OK...Private strName = " & obj.Get
Name)

```

### C#

```

//從Soap Xml檔案檔案反序列化回原物件
SoapFormatter formatter = new SoapFormatter();
FileStream fs = new FileStream(Application.StartupPath + "\\MyO
bject_Soap.xml", FileMode.Open);
MyObject obj = (MyObject)formatter.Deserialize(fs);
fs.Close();
MessageBox.Show("Deserialize OK...Private strName = " + obj.Ge
tName);

```

### 練習5.1：使用Binary序列化方法保存物件

•練習如何使用Binary序列化方法，將應用程式中使用的物件保存於檔案系統，並於下次使用時能夠載入物件繼續使用。

•預估實作時間：15分鐘

### 練習 5.1：使用 Binary 序列化方法保存物件

#### 目的：

練習如何使用 Binary 序列化方法，將應用程式中使用的物件保存於檔案系統，並於下次使用時能夠載入物件繼續使用。

#### 功能描述：

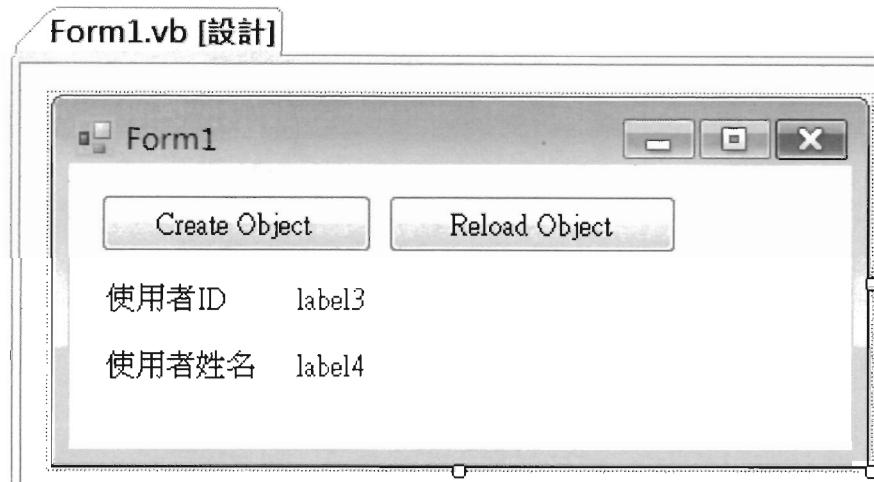
在這個練習中，你將使用 Visual Studio 開發工具建立一個類別，然後於應用程式中建立使用，並使用 Binary 序列化方法將物件保存於檔案系統，之後再讀回應用程式中使用。

#### 預估實作時間：15 分鐘

#### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod05\_1。

3. 在 Form1 的設計畫面中放置兩個 Button 控制項及兩個 Label 控制項，如下圖：



4. 在 Form1 的程式中匯入命名空間，如下：

```
Visual Basic
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.IO
```

```
C#
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
```

5. 在 Mod05\_1 專案點選滑鼠右鍵新增一個類別，命名為 Users.vb (Visual Basic 專案) 或 Users.cs (C#專案)，類別程式內容如下：

```
Visual Basic
<Serializable()> _
Public Class Users
    Public UserID As Integer
    Public UserName As String
End Class
```

```
C#
using System;
using System.Collections.Generic;
using System.Text;

namespace Mod04_2
{
    [Serializable()]
    public class Users
```

```
{
    public int UserID;
    public string UserName;
}
}
```

6. 在[Create Object]按鈕的 Click 事件加入以下程式，建立物件  
並將物件序列化成 Users.bin 檔案：

Visual Basic

```
'1, 建立及使用物件
Dim obj As New Users
obj.UserID = 1
obj.UserName = "Jerry"
'2, 將物件序列化成Binary檔案
Dim formatter As New BinaryFormatter()
Dim fs As New FileStream(Application.StartupPath & "..\..\..\Users.bin", FileMode.Create)
formatter.Serialize(fs, obj)
fs.Close()
MessageBox.Show("Serialize OK...")
```

C#

```
//1, 建立及使用物件
Users obj = new Users();
obj.UserID = 1;
obj.UserName = "Jerry";
//2, 將物件序列化成Binary檔案
BinaryFormatter formatter = new BinaryFormatter();
FileStream fs = new FileStream(Application.StartupPath + "..\..\..\..\Users.bin", FileMode.Create);
formatter.Serialize(fs, obj);
fs.Close();
MessageBox.Show("Serialize OK...");
```

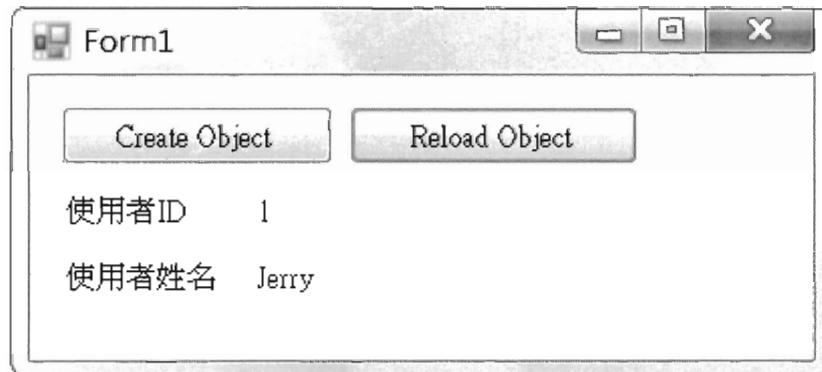
7. 在[Reload Object]按鈕的 Click 事件加入以下程式，將  
Users.bin 檔案中的資料還原成物件並顯示物件內容：

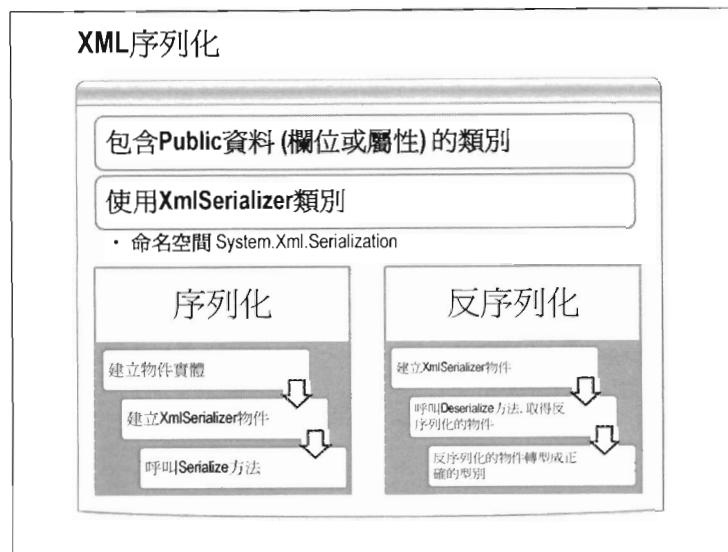
Visual Basic

```
'1, 從Binary檔案反序列化回原物件
Dim formatter As New BinaryFormatter()
Dim fs As New FileStream(Application.StartupPath & "..\..\..\Users.bin", FileMode.Open)
Dim obj As Users = CType(formatter.Deserialize(fs), Users)
fs.Close()
'2, 顯示資料
label3.Text = obj.UserID
label4.Text = obj.UserName
```

```
C#
//1, 從Binary檔案反序列化回原物件
BinaryFormatter formatter = new BinaryFormatter();
FileStream fs = new FileStream(Application.StartupPath + "..\\..\\
..\\"Users.bin", FileMode.Open);
Users obj = ((Users)(formatter.Deserialize(fs)));
fs.Close();
//2, 顯示資料
label3.Text = obj.UserID.ToString();
label4.Text = obj.UserName;
```

8. 測試時先點選[Create Object]按鈕將物件序列化成 Users.bin 檔案。
9. 點選[Reload Object]按鈕顯示載入 Users.bin 後的物件資料，結果如下圖：





## XML 序列化

XML 序列化針對物件的 Public 成員資料，轉換成 XML 格式資料串流，由於 XML 是開放的標準，所以好處就是能讓所有的應用程式都能接受。就像 Web Service 技術也是使用 XML 序列化的 `XmlSerializer` 類別來建立 XML 資料串流的。

### 建立一個類別包含 **Public** 的資料

XML 序列化技術只能針對 Public 資料做轉換的動作，所以設計類別的時候只設計 Public 成員資料，範例如下：

#### Visual Basic

```
Public Class MyBook
    Public BookID As Integer
    Public BookName As String
End Class
```

#### C#

```
public class MyBook {
    public int BookID;
```

```
    public string BookName;
}
```

## XML 序列化物件

設計好包含 Public 成員的類別之後，在主程式中先引用相關命名空間，如下：

Visual Basic Imports System.IO Imports System.Xml.Serialization
---

C# using System.IO; using System.Xml.Serialization;
---

接下來建立實體物件並使用，然後使用建立 `XmlSerializer` 物件並呼叫 `Serialize` 方法將實體物件序列化成檔案，與前面介紹的 `Binary` 序列化方法相當類似，範例如下：

Visual Basic '1, 建立及使用物件 Dim obj As New MyBook obj.BookID = 10 obj.BookName = "ASP.NET完全探索" '2, 將物件序列化成XML檔案 Dim ser As New XmlSerializer(GetType(MyBook)) Dim fs As New FileStream(Application.StartupPath & "\MyBook.xml", FileMode.Create) ser.Serialize(fs, obj) fs.Close()
--

C# //1, 建立及使用物件 MyBook obj = new MyBook(); obj.BookID = 10; obj.BookName = "ASP.NET完全探索"; //2, 將物件序列化成XML檔案 XmlSerializer ser = new XmlSerializer(typeof(MyBook)); Dim fs As New FileStream(Application.StartupPath + "\\MyBook.xml", FileMode.Create); ser.Serialize(fs, obj);
---

```
fs.Close();
```

執行完畢後可以檢視 MyBook.xml 檔案是否產生，內容應該如下：

```
<?xml version="1.0"?>
<MyBook xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <BookID>10</BookID>
  <BookName>ASP.NET完全探索</BookName>
</MyBook>
```

## 反序列化物件

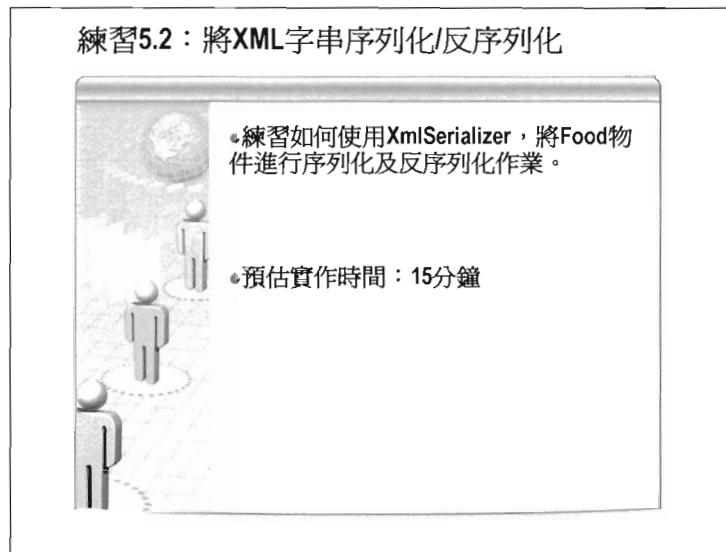
反過來，從檔案還原物件也是透過 Deserialize 方法完成，然後取出物件資料以證明還原成功，範例如下：

### Visual Basic

```
'從XML檔案反序列化回原物件
Dim ser As New XmlSerializer(GetType(MyBook))
Dim fs As New FileStream(Application.StartupPath & "\MyBook.xml",
", FileMode.Open)
Dim obj As MyBook = CType(ser.Deserialize(fs), MyBook)
fs.Close()
MessageBox.Show("Deserialize OK...Public BookName = " & obj.BookName)
```

### C#

```
//從XML檔案反序列化回原物件
XmlSerializer ser = new XmlSerializer(typeof(MyBook));
FileStream fs = new FileStream(Application.StartupPath + "\\MyBook.xml", FileMode.Open);
MyBook obj = (MyBook)ser.Deserialize(fs);
fs.Close();
MessageBox.Show("Deserialize OK...Public BookName = " + obj.BookName);
```



## 練習 5.2：將 XML 字串序列化/反序列化

目的：

練習如何使用 XmlSerializer，將 Food 物件進行序列化及反序列化作業。

功能描述：

在這個練習中自訂一個 Food 類別，並使用 XmlSerializer 進行 Food 物件的序列化將 XML 字串顯示於文字方塊，並可從字串反序列化回物件。

預估實作時間：15 分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod05\_2。

3. 加入一個類別取名為 Food，並加入兩個屬性 Name 與 Price，  
程式碼如下：

```
Visual Basic
Public Class Food

    Private _name As String
    Private _price As Decimal
    Public Property Name() As String
        Get
            Return _name
        End Get
        Set(ByVal value As String)
            _name = value
        End Set
    End Property

    Public Property Price() As Decimal
        Get
            Return _price
        End Get
        Set(ByVal value As Decimal)
            _price = value
        End Set
    End Property
End Class
```

```
C#
public class Food
{
    public string Name { get; set; }
    public decimal Price { get; set; }
}
```

4. 在 Form 放兩個 Button 與一個 TextBox。

5. 先匯入必要的命名空間：

```
Visual Basic
Imports System.Xml
Imports System.IO
Imports System.Xml.Serialization
```

```
C#
using System.Xml;
using System.Xml.Serialization;
using System.IO;
```

6. 在第一個 Button 的 Click 事件進行 Food 物件序列化的程序。

**Visual Basic**

```
Dim f As New Food() With {.Name = "雞腿便當", .Price = 80}
Dim sw As New StringWriter()
Dim ser As New XmlSerializer(GetType(Food))
ser.Serialize(sw, f)
sw.Flush()
textBox1.Text = sw.ToString()
```

**C#**

```
Food f = new Food() { Name = "雞腿便當", Price = 80 };
StringWriter sw = new StringWriter();
XmlSerializer ser = new XmlSerializer(typeof(Food));
ser.Serialize(sw, f);
sw.Flush();
textBox1.Text = sw.ToString();
```

7. 在第二個 Button 的 Click 事件進行 Food 物件反序列化的程序。

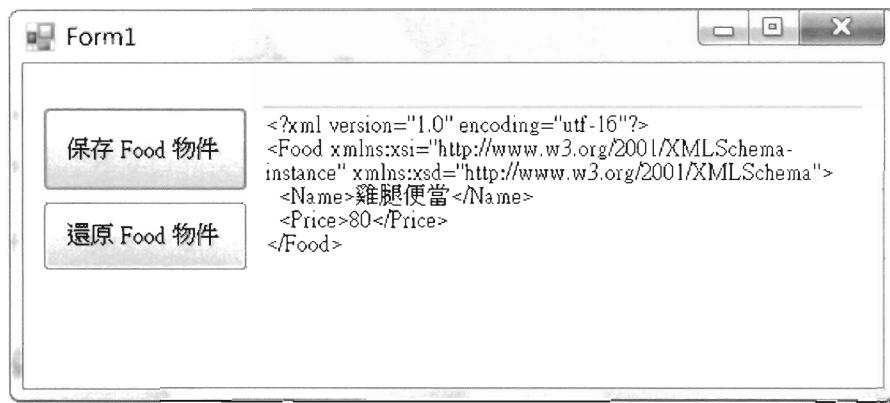
**Visual Basic**

```
Dim ser As New XmlSerializer(GetType(Food))
Dim sr As New StringReader(textBox1.Text)
Dim f As Food = DirectCast(ser.Deserialize(sr), Food)
textBox1.Clear()
If f IsNot Nothing Then
    textBox1.Text = f.Name & vbTab & f.Price
Else
    textBox1.Text = "不是吃的!!"
End If
```

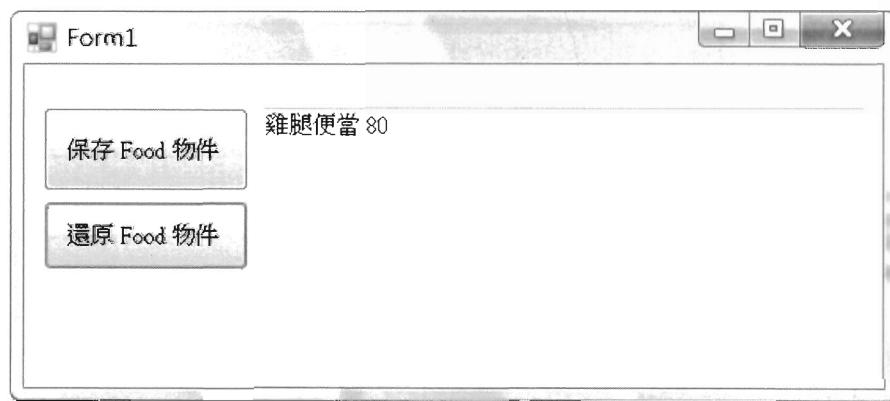
**C#**

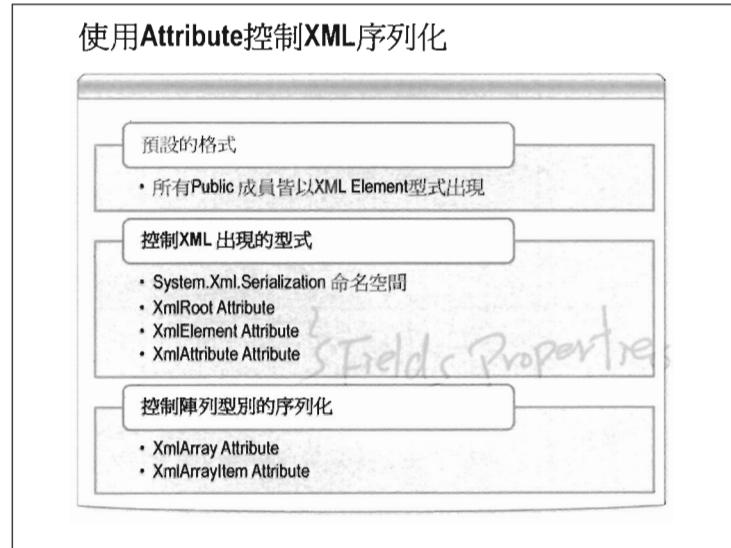
```
XmlSerializer ser = new XmlSerializer(typeof(Food));
StringReader sr = new StringReader(textBox1.Text );
Food f = ser.Deserialize(sr) as Food;
textBox1.Clear();
if (f != null)
    textBox1.Text = f.Name + "\t" + f.Price;
else
    textBox1.Text = "不是吃的!!";
```

8. 執行序列化的結果：



9. 執行反序列化的結果：





## 使用 Attribute 控制 XML 序列化

XML 序列化後的 XML 資料有預設的格式，標籤名稱都是使用類別名稱以及類別成員名稱，而且都是使用 XML Element 描述，藉由適當的 Attribute 可以控制序列化所產生的 XML 文件格式。

### 建立一個類別包含 Public 的資料

XML 序列化技術只能針對 Public 資料做轉換的動作，所以設計類別的時候只設計 Public 成員資料，範例如下：

Visual Basic

```
Public Class Product
    Public ProductID As Integer
    Public ProductName As String
End Class
```

C#

```
public class Product {
    public int ProductID;
    public string ProductName;
```

```
}
```

## XML 序列化及反序列化物件

使用 `XmlSerializer` 物件作序列化動作後，預設會使用類別名稱當作根節點標籤，而類別成員 `ProductID` 也會成為子節點標籤，產生如下的 XML 格式資料：

```
<?xml version="1.0"?>
<Product xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <ProductID>50</ProductID>
  <ProductName>MCSD.NET課程</ProductName>
</Product>
```

## 如何使用 `Attribute` 控制 XML 序列化

使用相關控制 XML 序列化 `Attribute`，需在類別中引用 `System.Xml.Serialization` 命名空間，接下來可使用 `XmlRoot`、`XmlElement` 及 `XmlAttribute` 來控制 XML 序列化後的 XML 格式，如下：

### Visual Basic

```
Imports System.Xml.Serialization

<XmlRoot("產品")> _
Public Class Product
  <XmlElement("產品編號")> _
  Public ProductID As Integer
  <XmlAttribute("產品名稱")> _
  Public ProductName As String
End Class
```

### C#

```
using System.Xml.Serialization;

[XmlRoot("產品")]
public class Product {
  [XmlElement("產品編號")]
  public int ProductID;
  [XmlAttribute("產品名稱")]
```

```
    public string ProductName;
}
```

經過 XmlSerializer 物件序列化之後會產生以下 XML 格式資料，根節點標籤由[Product]改成了[產品]，[ProductID]也修改成[產品編號]，而原本的[ProductName]標籤變成了[產品名稱]屬性，結果如下：

```
<?xml version="1.0"?>
<產品 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema" 產品名稱="MC
SD.NET 課程">
  <產品編號>50</產品編號>
</產品>
```

## 控制陣列型別的序列化

當使用 XML 序列化的類別如果是陣列型別時，如下列類別的設計：

### Visual Basic

```
Public Class Group
  Public Employees(1) As Employee
End Class
Public Class Employee
  Public Name As String
End Class
```

### C#

```
public class Group{
  public Employee[] Employees;
}
public class Employee{
  public string Name;
}
```

經過 XmlSerializer 物件序列化的程式範例如下：

### Visual Basic

```
'1, 建立及使用物件
Dim obj As New Group
```

```

Dim emp1, emp2 As New Employee
emp1.Name = "Jerry"
emp2.Name = "John"
obj.Employees(0) = emp1
obj.Employees(1) = emp2

'2, 將物件序列化成XML檔案
Dim ser As New XmlSerializer(GetType(Group))
Dim fs As New FileStream(Application.StartupPath & "\Group.xml",
 FileMode.Create)
ser.Serialize(fs, obj)
fs.Close()

```

## C#

```

//1, 建立及使用物件
Group obj = new Group();
Employee emp1, emp2 = new Employee();
emp1.Name = "Jerry";
emp2.Name = "John";
obj.Employees(0) = emp1;
obj.Employees(1) = emp2;

//2, 將物件序列化成XML檔案
XmlSerializer ser = new XmlSerializer(typeof(Group));
FileStream fs = new FileStream(Application.StartupPath + "\\Group.xml",
 FileMode.Create);
ser.Serialize(fs, obj);
fs.Close();

```

序列化之後會產生以下 XML 格式資料：

```

<?xml version="1.0"?>
<Group xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Employees>
    <Employee>
      <Name>Jerry</Name>
    </Employee>
    <Employee>
      <Name>John</Name>
    </Employee>
  </Employees>
</Group>

```

使用相關控制 XML 序列化 Attribute，需在類別中引用 System.Xml.Serialization 名稱空間，接下來可使用 XmlArray 及 XmlArrayItem 來控制 XML 序列化後的 XML 格式，如下：

## Visual Basic

```
Imports System.Xml.Serialization

Public Class Group
    <XmlArray("團隊成員")> _
    <XmlArrayItem("成員名稱")> _
    Public Employees(1) As Employee
End Class

Public Class Employee
    Public Name as String
End Class
```

## C#

```
using System.Xml.Serialization;

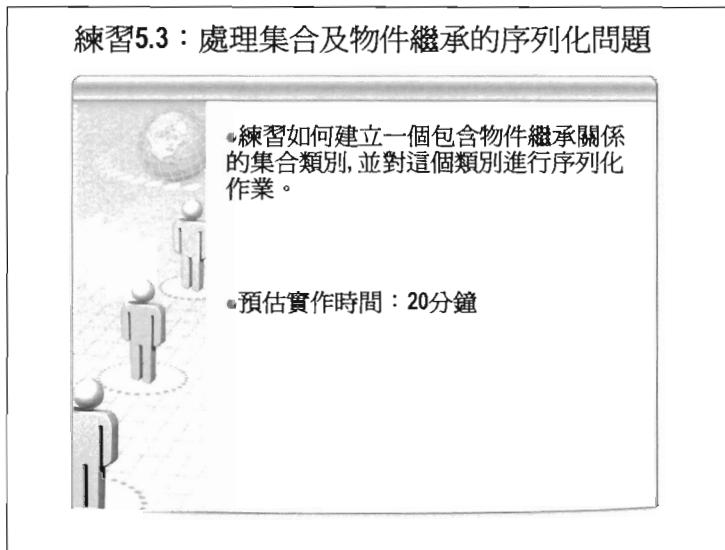
public class Group{
    [XmlArray("團隊成員")]
    [XmlArrayItem("成員名稱")]
    public Employee[] Employees;
}

public class Employee{
    public string Name;
}
```

經過 XmlSerializer 物件序列化之後會產生以下 XML 格式資料，陣列標籤由[Employees]改成了[團隊成員]，[Employee]也修改成[成員名稱]，結果如下：

```
<?xml version="1.0"?>
<Group xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <團隊成員>
        <成員名稱>
            <Name>Jerry</Name>
        </成員名稱>
        <成員名稱>
            <Name>John</Name>
        </成員名稱>
    </團隊成員>
```

</Group>



## 練習 5.3：處理集合及物件繼承的序列化問題

目的：

練習如何建立一個包含物件繼承關係的集合類別，並對這個類別進行序列化作業。

功能描述：

繼續前一個練習，這個練習會建立一個 SeaFood 類別繼承自 Food 類別及 Group 類別裡面包含 Food 集合。程式會對 Group 類別進行序列化作業，必須讓 Group 類別可以正確無誤的進行序列化作業。

預估實作時間：20 分鐘

實作步驟：

- 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
- 開啓前一個專案，或從 Practices\VB(或 CS)\Mod05\_3\Starter\Mod05\_2 開啓 Mod05\_2.sln。
- 開啓 Food.vb (或.cs)，在 Food 類別裡覆寫 ToString()方法。

```

Visual Basic
Public Overrides Function ToString() As String
    Return Name & vbTab & Price
End Function

```

```

C#
public override string ToString()
{
    return Name + "\t" + Price ;
}

```

4. 加入一個新的類別名為 SeaFood，繼承自 Food，並多個 Type 屬性，屬性序列化成 XML 時必須是 Attribute，還要覆寫 ToString()。

```

Visual Basic
Public Class SeaFood : Inherits Food
    Private _type As String
    <XmlAttribute("Type")> _
    Public Property Type() As String
        Get
            Return _type
        End Get
        Set(ByVal value As String)
            _type = value
        End Set
    End Property

    Public Overrides Function ToString() As String
        Return Type & vbTab & MyBase.ToString()
    End Function
End Class

```

```

C#
public class SeaFood : Food {
    [XmlAttribute ("Type")]
    public string Type { get; set; }
    public override string ToString()
    {
        return Type + "\t" + base.ToString ();
    }
}

```

5. 加入一個類別名為 Group，及公開屬性 Foods，型別 List。

```

Visual Basic
Public Class Group
    Public Foods As New List(Of Food)
End Class

```

```
C#
public class Group
{
    public List<Food> Foods= new List<Food>();
}
```

6. 開啓 Form1 放兩個 Button。

7. 先匯入必要的命名空間：

```
Visual Basic
Imports System.Text
```

8. 在 Button3 的按鈕建立 Group 物件，並且序列化：

```
Visual Basic
Dim g As New Group()
g.Foods.Add(New Food() With {.Name = "雞腿便當", .Price = 80})
g.Foods.Add(New SeaFood() With {.Name = "大閘蟹", .Price = 300, .Type = "蝦蟹類"})
Dim sw As New StringWriter()
Dim ser As New XmlSerializer(GetType(Group))
ser.Serialize(sw, g)
sw.Flush()
textBox1.Text = sw.ToString()
```

```
C#
Group g = new Group();
g.Foods.Add (new Food() { Name = "雞腿便當", Price = 80 });
g.Foods.Add(new SeaFood() { Name = "大閘蟹", Price = 300, Type = "蝦蟹類" });
StringWriter sw = new StringWriter();
XmlSerializer ser = new XmlSerializer(typeof(Group));
ser.Serialize(sw, g);
sw.Flush();
textBox1.Text = sw.ToString();
```

9. 在 Button4 的按鈕進行反序列化作業：

```
Visual Basic
Dim ser As New XmlSerializer(GetType(Group))
Dim sr As New StringReader(textBox1.Text)
Dim g As Group = DirectCast(ser.Deserialize(sr), Group)
textBox1.Clear()
If g IsNot Nothing Then
    Dim sb As New StringBuilder()
    For Each f As Food In g.Foods
        sb.AppendLine(f.ToString())
```

```

Next
    textBox1.Text = sb.ToString()
Else
    textBox1.Text = "不是吃的!!"
End If

```

```

C#
XmlSerializer ser = new XmlSerializer(typeof(Group));
StringReader sr = new StringReader(textBox1.Text);
Group g = ser.Deserialize(sr) as Group;
textBox1.Clear();
if (g != null)
{
    StringBuilder sb = new StringBuilder();
    foreach (Food f in g.Foods)
    {
        sb.AppendLine ( f.ToString () );
    }
    textBox1.Text = sb.ToString();
}
else
    textBox1.Text = "不是吃的!!";

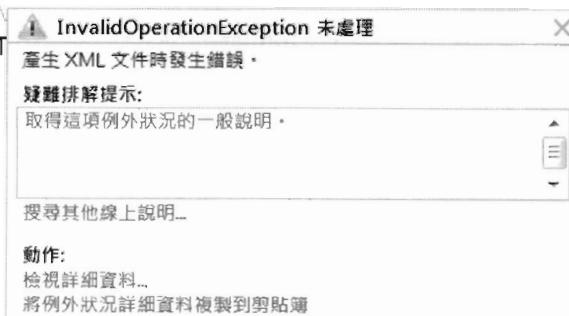
```

10. 執行，並按下 Button3，將會發生錯誤：

```

StringWriter sw = new StringWriter();
XmlSerializer ser = new XmlSerializer(typeof(Group));
ser.Serialize(sw, g);
sw.Flush();
textBox1.Text = sw.T

```



11. 請校正錯誤，在 Group 類別的 Foods 成員加上：

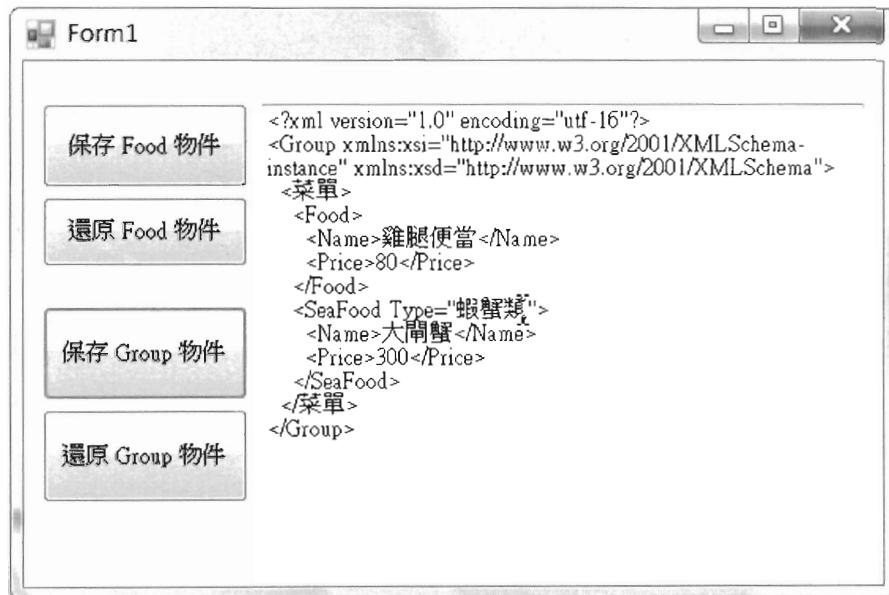
```

Visual Basic
Public Class Group
<XmlAttribute("菜單")> _
<XmlAttribute(GetType(Food))> _
<XmlAttribute(GetType(SeaFood))> _
Public Foods As New List(Of Food)
End Class

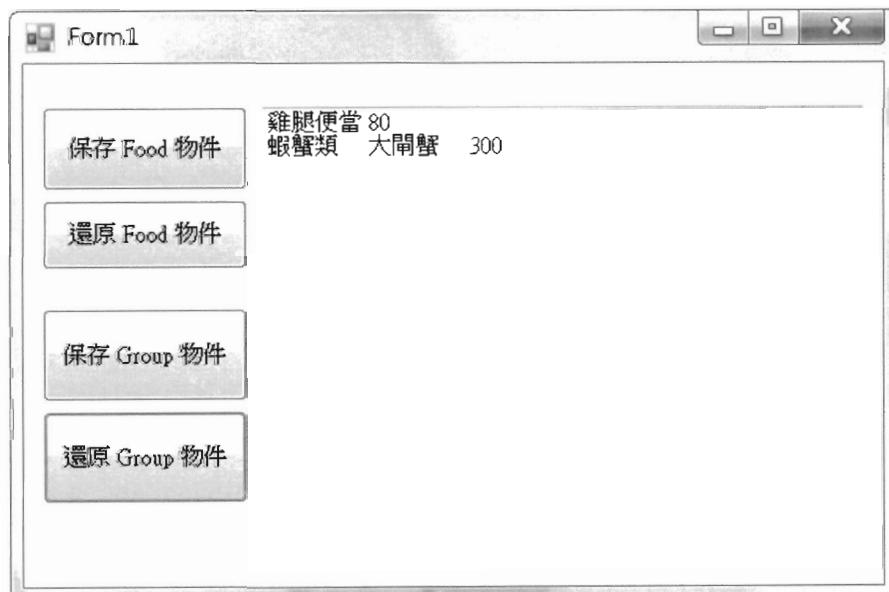
```

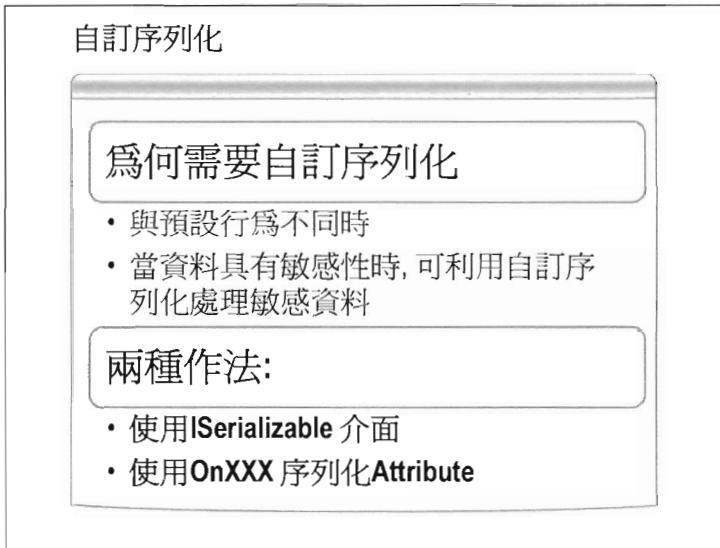
```
C#
public class Group
{
    [XmlArray("菜單")]
    [XmlArrayItem(typeof(Food))]
    [XmlArrayItem(typeof(SeaFood))]
    public List<Food> Foods = new List<Food>();
}
```

12. 再執行一次程式，Button3 的執行結果：



13. Button4 的執行結果：





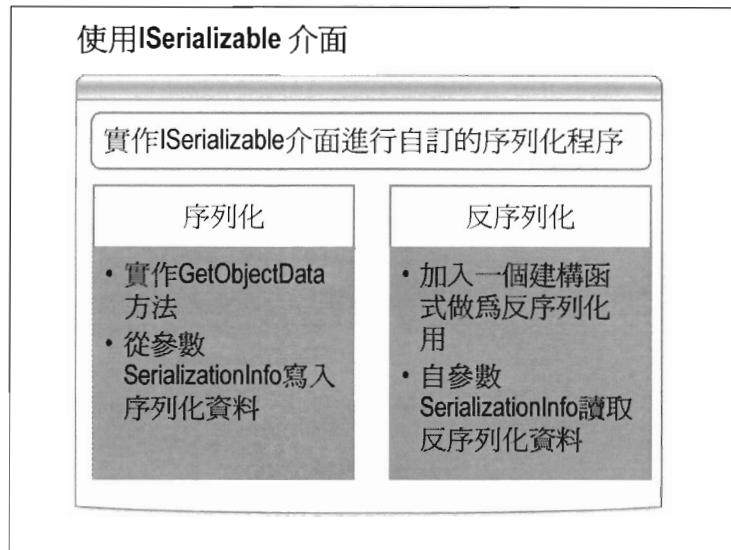
## 自訂序列化

看起來序列化似乎很簡單，需要 Binary 或 SOAP 的序列化時，在類別之前加上 `Serializable Attribute` 即可。可如果資料是具有敏感性的，不太適合在未經編碼的情況下被傳遞或者保存下來...等各種原因，這時就必須自訂序。

自訂序列化也就是自行撰寫序列化及反序列化的程序，有兩種方式：

- 使用 `ISerializable` 介面
- 使用 `OnXXX` 序列化 `Attribute`

這一節將分別介紹。



## 使用 ISerializable 介面

ISerializable 介面，屬於 System.Runtime.Serialization 命名空間。實作這個介面的類別實作一個 GetObjectData 方法及建構函式。

序列化程序是寫在 GetObjectData 方法，它有兩個參數：

- SerializationInfo，寫入要保存的資料。
- StreamingContent，序列化作業輸出的目的端資料流。

反序列化程序必須寫在一個非公開的建構函式，一樣要有兩個參數，這兩個參數與 GetObjectData 方法相同。

以下範例是 Employee 類別 Salary 屬性可能是敏感性的資料，當 Level 值小於 3 時 Salary 不可被保存或傳遞，反之則可以。

```

Visual Basic
Imports System.Runtime.Serialization
<Serializable()> Public Class Employee
    Implements ISerializable

    Private _name As String
    Private _level As Short
    <NonSerialized()> Private _salary As Decimal

    Public Property Name() As String

```

```

Get
    Return _name
End Get
Set(ByVal value As String)
    _name = value
End Set
End Property

Public Property Level() As Short
Get
    Return _level
End Get
Set(ByVal value As Short)
    _level = value
End Set
End Property

Public Property Salary() As Decimal
Get
    Return _salary
End Get
Set(ByVal value As Decimal)
    _salary = value
End Set
End Property
Sub New()

End Sub

Private Sub New(ByVal info As SerializationInfo, ByVal context
As StreamingContext)
    Name = info.GetString("Name")
    Level = info.GetInt16("Level")
    Salary = info.GetDecimal("Salary")
End Sub

Public Sub GetObjectData(ByVal info As System.Runtime.Serialization.SerializationInfo, ByVal context As System.Runtime.Serialization.StreamingContext) Implements System.Runtime.Serialization.ISerializable.GetObjectData
    info.AddValue("Name", Name)
    info.AddValue("Level", Level)
    info.AddValue("Salary", If(Level < 3, 0, Salary))
End Sub
End Class

```

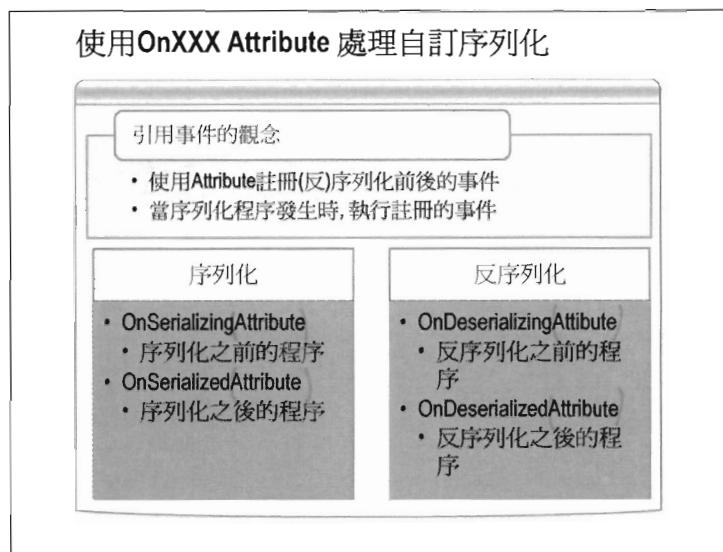
```

C#
using System.Runtime.Serialization;
namespace Demo02
{

```

```
[Serializable]
class Employee:ISerializable
{
    [NonSerialized] private decimal _salary;
    public String Name { get; set; }
    public short Level { get; set; }
    public decimal Salary {
        get{return _salary ;}
        set { _salary = value; }
    }
    public Employee() { }
    #region ISerializable 成員
    private Employee(SerializationInfo info, StreamingContext context)
    {
        Name = info.GetString ("Name");
        Level = info.GetInt16("Level");
        Salary = info.GetDecimal ("Salary");
    }

    public void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        //throw new NotImplementedException();
        info.AddValue("Name", Name);
        info.AddValue("Level", Level);
        info.AddValue("Salary", (Level<3 ? 0:Salary));
    }
    #endregion
}
```

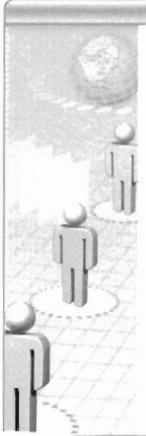


## 使用 OnXXX Attribute 處理自訂序列化

.NET 2.0 版之後序列化新增了序列化過程的事件處理程序，以簡化自訂序列化的程序，它是以 Attribute 註冊(反)序列化前後的事件，當序列化程序發生時，執行註冊的程序。

序列化時會執行 OnSerializingAttribute 及 OnSerializedAttribute。OnSerializingAttribute 是在進行序列化之前會執行的程序，OnSerializedAttribute 則是在完成序列化之後會執行的程序。

反序列化時會執行 OnDeserializingAttribute 及 OnDeserializedAttribute。OnDeserializingAttribute 是在進行反序列化之前會執行的程序，OnDeserializedAttribute 則是在完成反序列化之後會執行的程序。

**練習5.4：使用OnXXXAttribute**

- 練習如何 OnXXXAttribute 的方式在序列化及反序列化時處理敏感性資料。

- 預估實作時間：15分鐘

**練習 5.4：使用 OnXXXAttribute****目的：**

練習如何 OnXXXAttribute 的方式在序列化及反序列化時處理敏感性資料。

**功能描述：**

在這個練習要處理 Employee 類別的薪水欄位，當 Level 小於 3 時，Salary 的值不可直接被序列化。這個練習要用 OnXXXAttribute 的方式自訂序列化程序。

**預估實作時間：15 分鐘****實作步驟：**

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod05\_4。
3. 匯入命名空間：

```
Visual Basic
Imports System.Runtime.Serialization
```

```
C#
using System.Runtime.Serialization;
```

4. 加入一個 Employee 類別，從 Practices\VB(或  
CS)\Mod05\_3\Starter\Mod05\_2 開啓 Employee.txt 文字檔，複  
製貼以下類別程式：

```
Visual Basic
<Serializable()> Public Class Employee

    Private _name As String
    Private _level As Short
    <NonSerialized()> Private _salary As Decimal
    Private sensData As Decimal
    Private _updateDate As DateTime

    Public Property Name() As String
        Get
            Return _name
        End Get
        Set(ByVal value As String)
            _name = value
        End Set
    End Property

    Public Property Level() As Short
        Get
            Return _level
        End Get
        Set(ByVal value As Short)
            _level = value
        End Set
    End Property

    Public Property Salary() As Decimal
        Get
            Return _salary
        End Get
        Set(ByVal value As Decimal)
            _salary = value
        End Set
    End Property

    Public Property UpdateTime() As DateTime
        Get
            Return _updateDate
        End Get
        Private Set(ByVal value As DateTime)
```

```

    _updateDate = value
End Set
End Property
End Class

```

```

C#
[Serializable] class Employee
{
    [NonSerialized] private decimal _salary;
    private decimal sensdata;
    public String Name { get; set; }
    public short Level { get; set; }
    public decimal Salary {
        get{return _salary ;}
        set { _salary = value; }
    }
    public DateTime UpdateTime { get; private set; }
}

```

5. 加上序列化之後的程序。

```

Visual Basic
<OnSerializing()> _
Public Sub Serializing(ByVal content As StreamingContext)
    sensdata = If(Level < 3, 0, Salary)
End Sub

```

```

C#
[OnSerializing]
public void Serializing(StreamingContext content)
{
    sensdata = (Level < 3 ? 0 : Salary);
}

```

6. 加上反序列化之前的程序。

```

Visual Basic C#
[OnDeserializing]
public void Deserializing(StreamingContext content)
{
    Console.WriteLine("反序列化之前執行的程序");
    Console.WriteLine("Name:{0}, Level:{1}, sensdata:{2}",
Name, Level, sensdata );
    UpdateTime = DateTime.Now;
}

```

```

Visual Basic C#
<OnDeserializing()> Public Sub Deserializing(ByVal content As

```

```

StreamingContext)
    Console.WriteLine("反序列化之前執行的程序")
    Console.WriteLine("Name:{0}, Level:{1}, sensdata:{2}", Na
me, Level, sensData)
    UpdateTime = DateTime.Now
End Sub

```

7. 加上反序列化之後的程序。

```

Visual Basic
<OnDeserialized()> Public Sub Deserialized(ByVal content As S
treamingContext)
    Salary = If(Level < 3, 0, sensData)
End Sub

```

```

C#
[OnDeserialized]
public void Deserialized(StreamingContext content)
{
    Salary = (Level < 3 ? 0 : sensdata);
}

```

8. 在 Form 放兩個 Button 與一個 Label。

9. 決定命名空間：

```

Visual Basic
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary

```

```

C#
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

```

10. 在 Button1 進行序列化作業：

```

Visual Basic
Dim emp As New Employee() With {.Name = "Anita", .Level
= 10, .Salary = 30000}
Using fs As New FileStream("emp.bin", FileMode.Create, File
Access.Write)
    Dim formatter As New BinaryFormatter()
    formatter.Serialize(fs, emp)
End Using

```

```

C#
Employee emp = new Employee() { Name = "Anita", Level = 10,

```

```

        Salary = 30000 };
using (FileStream fs = new FileStream("emp.bin", FileMode.Create,
e, FileAccess.Write))
{
    BinaryFormatter formatter = new BinaryFormatter();
    formatter.Serialize(fs, emp);
}

```

11. 在 Button2 進行反序列化作業：

```

Visual Basic
Dim emp As Employee
Using fs As New FileStream("emp.bin", FileMode.Open, FileAccess.
Read)
    Dim formatter As New BinaryFormatter()
    emp = DirectCast(formatter.Deserialize(fs), Employee)
End Using
label1.Text = emp.Name & vbCrLf & emp.Level & vbCrLf & emp.S
alary

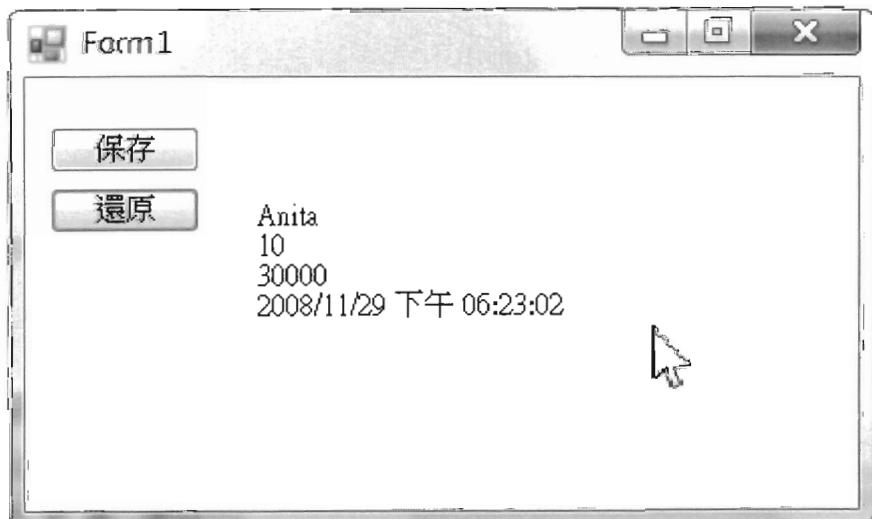
```

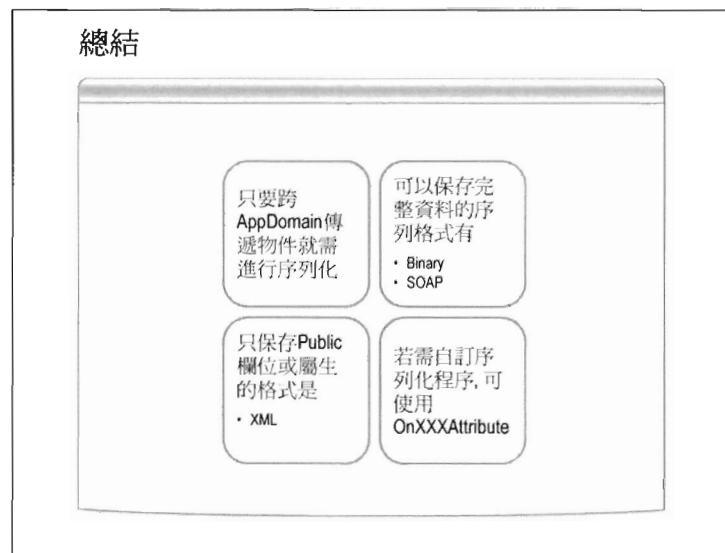
```

C#
Employee emp;
using (FileStream fs = new FileStream("emp.bin", FileMode.Open,
FileAccess.Read))
{
    BinaryFormatter formatter = new BinaryFormatter();
    emp = (Employee)formatter.Deserialize(fs);
}
label1.Text = emp.Name + "\n" + emp.Level + "\n"
+ emp.Salary +"\n" + emp.UpdateTime ;

```

12. 執行結果如下：





---

這一章介紹序列化程序包含四大重點：

- 只要跨 AppDomain 傳遞物件就需進行序列化。
- 可以保存完整資料的序列格式有 Binary、SOAP。
- 只保存 Public 欄位或屬生的格式是 XML。
- 若需自訂序列化程序, 可使用 OnXXXAttribute。

# 第六章：程式繪圖

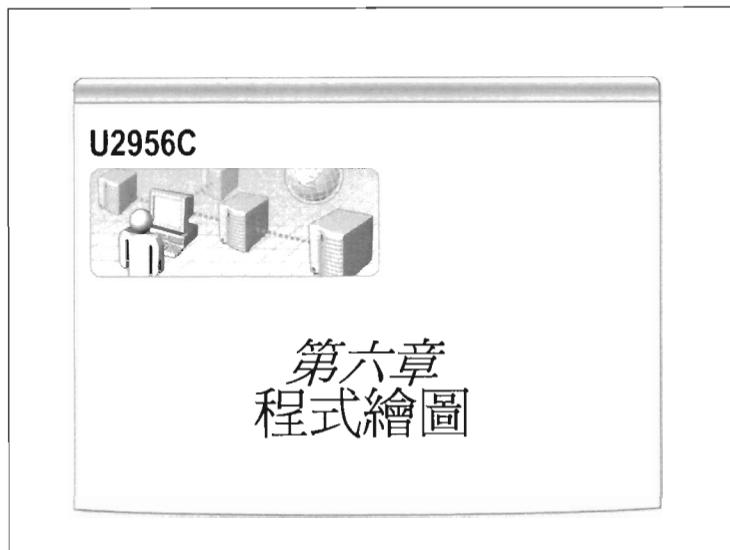
## 本章大綱

Point 及 Size 結構.....	4
Graphics 物件 .....	6
筆, 筆刷及字型物件 .....	7
直線.....	9
矩形.....	11
筆刷的變化.....	12
練習 6.1 : 繪製一個傳統型體溫計.....	15
圓形.....	18
Pie 形 .....	20
文字處理.....	22
練習 6.2 : 以 Pie 形呈現市佔率 .....	27
影像處理.....	30
練習 6.3 : 網頁動態影像處理.....	34

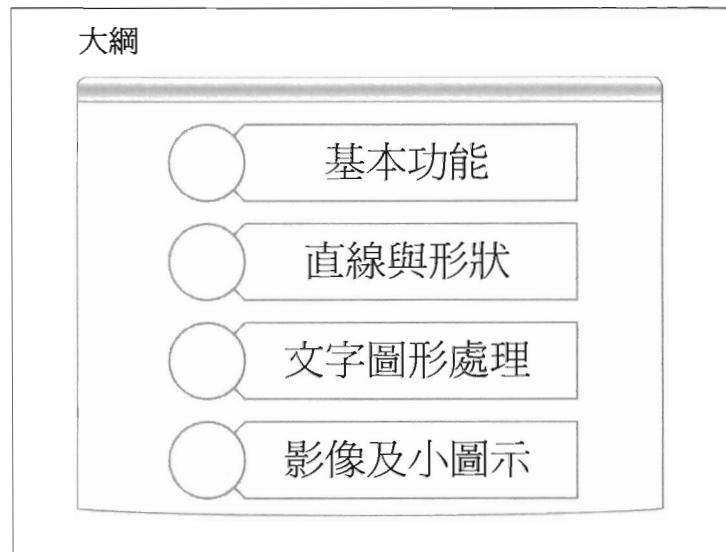
作者：

羅慧真





---



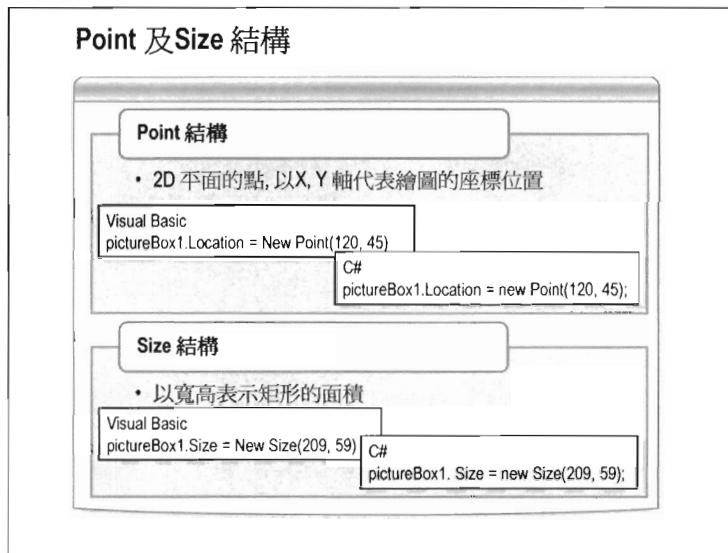
.NET Framework 提供較之前版本 GDI 技術更強大的 GDI+技術，讓開發者可以使用繪圖相關的類別庫，建立圖像、繪製文字、處理圖片等工作。

在視窗應用程式裡，所有的 Windows Form 及 Windows Control 都是靠 GDI+物件繪製出使用者介面的；雖然網站應用程式並非依靠繪圖來產生使用者介面，但是也能搭配 Image 伺服器控制項指定網頁程式動態產生圖片。

在.NET Framework 中提供的繪圖相關的類別庫在 System.Drawing 命名空間。在這個章節中你將學會如何繪製直線、形狀與、圖片…等。

本章介紹以下主題：

- 基本功能
- 直線與形狀
- 文字圖形處理
- 影像及小圖示



## Point 及 Size 結構

開始進行繪圖作業之前，你一定要先認識這兩個結構，一個是 Point 一個是 Size。

### Point 結構

Point 結構是決定繪圖物件的所在位置，它是 2D 平面的點，以 X 及 Y 軸代表繪圖的座標位置。你可以在 Windows Form 的控制項看到它的蹤跡，下面這個範例是將 PictureBox 控制項定位在表單 X 及 Y 軸的 120、45 位置：

Visual Basic pictureBox1.Location = New Point(120, 45)
---

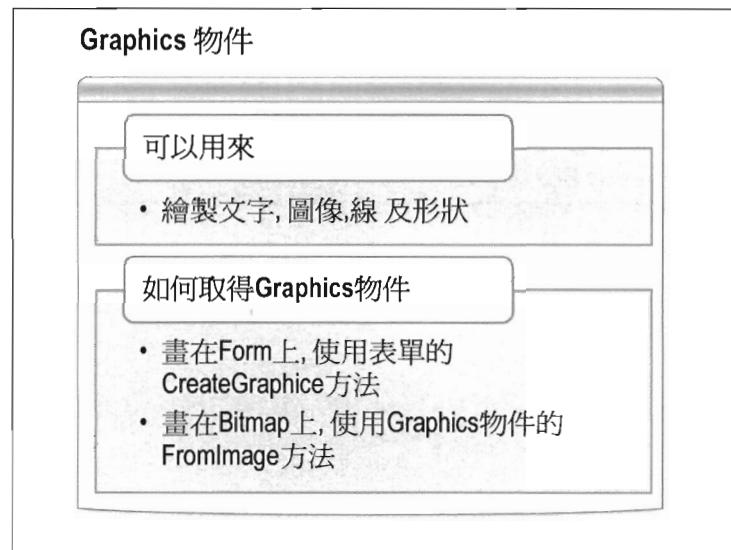
C# pictureBox1.Location = new Point(120, 45);
--

### Size 結構

Size 結構是決定繪圖物件的大小，它是以寬高表示矩形的面積。例如設定 PictureBox 寬為 209，高為 59 的大小。

```
Visual Basic  
pictureBox1. Size = new Size(209, 59);
```

```
C#  
pictureBox1. Size = new Size(209, 59);
```



## Graphics 物件

一個全新的 Graphics 物件就像一張空白的圖畫紙，藉由 Graphics 物件內建的一些方法繪製內容，也像一個全新的小畫家工具，開啓時是空白的畫面，但是利用工具列所提供的工具就能繪製內容。只不過在使用 GDI+技術時只能依靠程式，並沒有工具列可以使用。也可以這麼說，小畫家工具的功能都能使用 GDI+的相關物件來完成，在熟悉 GDI+技術後，開發者也能夠撰寫一個像小畫家一般的工具程式。

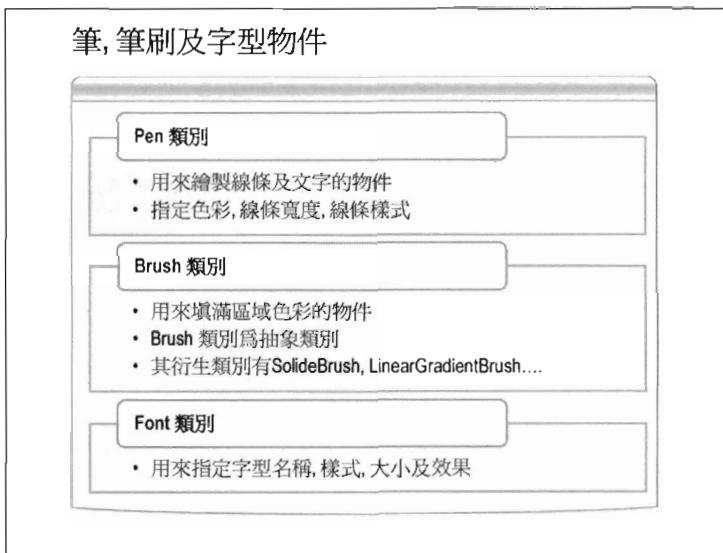
建構圖像內容必須先取得 Graphics 物件，在視窗程式中可以使用下列範例程式取得：

Visual Basic

```
Dim g As Graphics = Me.CreateGraphics
```

C#

```
Graphics g = this.CreateGraphics();
```



## 筆, 筆刷及字型物件

繪圖最常用的當然是筆和筆刷，而對應的物件就 Pen 與 SolidBrush 類別，而繪製文字時也需要設定字型及大小，此時就必須建立 Font 類別物件。

### 建立筆物件

建立筆物件時只需指定需要的顏色即可，範例如下：

```
Visual Basic
Dim p As New Pen(Color.Blue)
```

```
C#
Pen p = new Pen(Color.Blue);
```

### 建立筆刷物件

建立筆刷物件時也是只需指定需要的顏色即可，範例如下：

```
Visual Basic
```

```
Dim b As New SolidBrush(Color.Green)
```

C#

```
SolidBrush b = new SolidBrush(Color.Green);
```

### 建立字型物件

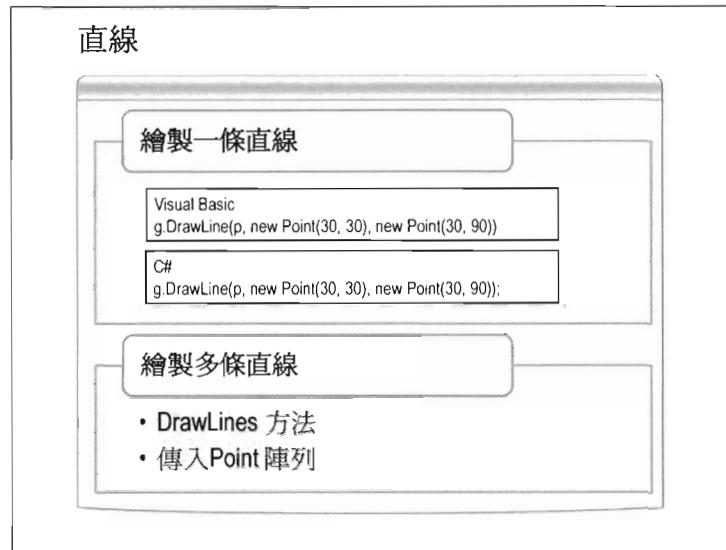
建立字型物件時需指定字型與大小，範例如下：

Visual Basic

```
Dim f As New Font("新細明體", 20)
```

C#

```
Font f = new Font("新細明體", 20);
```



## 直線

繪製直線時需要一個筆物件，並可指定顏色、樣式及寬度等等屬性，稍加變化，直線的起點座標固定，然後使用亂數來決定直線的終點座標，範例程式如下：

### Visual Basic

```
Dim rnd As New Random
Dim g As Graphics = Me.CreateGraphics
Dim p As New Pen(Color.Blue)
p.Width = 2
p.DashStyle = Drawing2D.DashStyle.DashDotDot
g.DrawLine(p, 150, 150, rnd.Next(50, 250), rnd.Next(50, 250))
```

### C#

```
Random rnd = new Random();
Graphics g = this.CreateGraphics();
Pen p = new Pen(Color.Blue);
p.Width = 2;
p.DashStyle = Drawing2D.DashStyle.DashDotDot;
g.DrawLine(p, 150, 150, rnd.Next(50, 250), rnd.Next(50, 250));
```

505 <--> 250

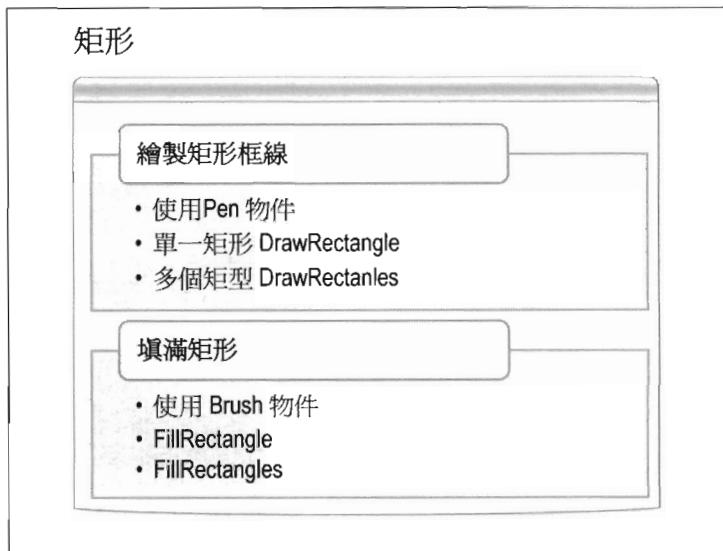
畫單一直線使用 DrawLine 方法，多條直線就使用 DrawLines 方法，有點不同的是傳入的是 Point 陣列。

Visual Basic

```
g.DrawLines(p,  
New Point() {New Point(150, 150), New Point(rnd.Next(50, 250),  
rnd.Next(50, 250)), New Point(rnd.Next(100, 300)), New Point(rn  
d.Next(100, 300))})
```

C#

```
g.DrawLines(p, new Point[] {new Point(150, 150), new Point(rnd.  
Next(50, 250), rnd.Next(50, 250)), new Point (rnd.Next (100,30  
0)), new Point(rnd.Next (100,300))});
```



## 矩形

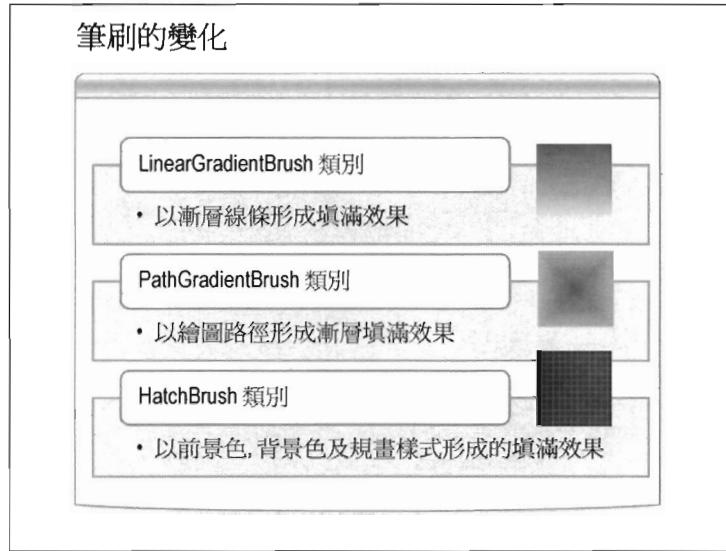
繪製矩形時前面的例子是使用筆物件並呼叫 Graphics 物件的 DrawRectangle 方法；這邊的範例改用筆刷物件並呼叫 Graphics 物件的 FillRectangle 方法，可以繪製出實心的方形，範例程式如下：

### Visual Basic

```
Dim g As Graphics = Me.CreateGraphics
Dim rnd As New Random
Dim b1 As New SolidBrush(Color.Green)
g.FillRectangle(b1, 50, 50, 150, 20)
g.FillRectangle(Brushes.Green, 50, 50, rnd.Next(10, 150), 20)
```

### C#

```
Graphics g = this.CreateGraphics();
Random rnd = new Random();
SolidBrush b1 = new SolidBrush(Color.Green);
g.FillRectangle(b1, 50, 50, 150, 20);
g.FillRectangle(Brushes.Green, 50, 50, rnd.Next(10, 150), 20);
```



## 筆刷的變化

填色需要用到筆刷物件，.NET 所有的筆刷類型都是繼承自 Brush 這個類別，一般比較常用純色(SolidBrush 類別)來填滿色彩。除了純色之外，.NET 還提供多種填色的筆刷，像是漸層...等。

### LinearGradientBrush 類別

LinearGradientBrush 類別提供以漸層線條形成填滿效果。以下範例會以漸層線條填滿效果。

#### Visual Basic

```
Dim g As Graphics = Me.CreateGraphics
g.Clear(Me.BackColor)
Dim r1 As New Rectangle(50, 50, 100, 100)
Dim b1 As New LinearGradientBrush(r1, Color.Green, Color.Yellow,
w, LinearGradientMode.Vertical)
g.FillRectangle(b1, r1)
```

#### C#

```
Graphics g = this.CreateGraphics();
g.Clear(this.BackColor);
Rectangle r1 = new Rectangle(50, 50, 100, 100);
LinearGradientBrush b1 = new LinearGradientBrush(r1, Color.Gree
```

```
n, Color.Yellow, LinearGradientMode.Vertical);
g.FillRectangle(b1, r1);
```

## PathGradientBrush 類別

PathGradientBrush 類別以繪圖路徑形成漸層填滿效果。以下範例方形路徑填滿漸層色。

### Visual Basic

```
Dim g As Graphics = Me.CreateGraphics
g.Clear(Me.BackColor)
Dim r1 As New Rectangle(50, 50, 100, 100)
Dim gp As New GraphicsPath()
gp.AddRectangle(r1)
Dim b1 As New PathGradientBrush(gp)
b1.CenterColor = Color.Red
b1.SurroundColors = New Color() {Color.Aqua}
g.FillRectangle(b1, r1)
```

### C#

```
Graphics g = this.CreateGraphics();
g.Clear(this.BackColor);
Rectangle r1 = new Rectangle(50, 50, 100, 100);
GraphicsPath gp = new GraphicsPath ();
gp.AddRectangle (r1);
PathGradientBrush b1 = new PathGradientBrush(gp);
b1.CenterColor = Color.Red;
b1.SurroundColors = new Color[] { Color.Aqua };
g.FillRectangle(b1, r1);
```

## HatchBrush 類別

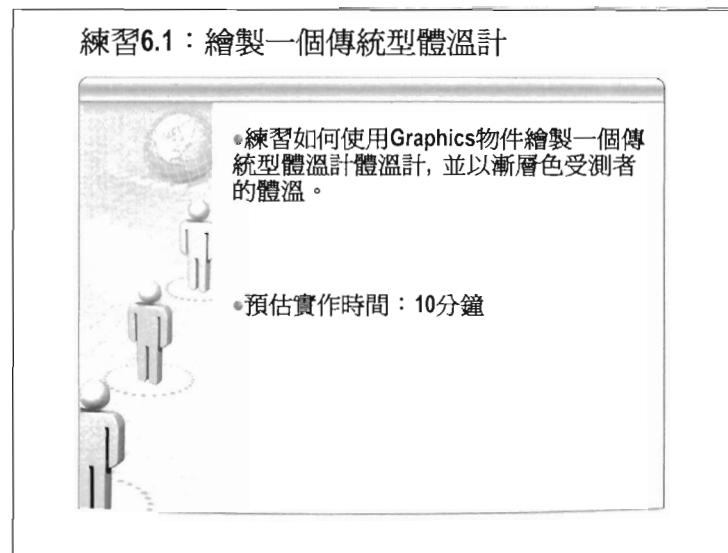
HatchBrush 類別以前景色、背景色及規畫樣式形成的填滿效果。以下範例以紅色為前景藍色為背景並以交叉線條填滿。

### Visual Basic

```
Dim g As Graphics = Me.CreateGraphics
g.Clear(Me.BackColor)
Dim r1 As New Rectangle(50, 50, 100, 100)
Dim b1 As New HatchBrush(HatchStyle.Cross, Color.Red, Color.Blue)
g.FillRectangle(b1, r1)
```

### C#

```
Graphics g = this.CreateGraphics();
g.Clear(this.BackColor);
Rectangle r1 = new Rectangle(50, 50, 100, 100);
HatchBrush b1 = new HatchBrush(HatchStyle.Cross, Color.Red, Co
lor.Blue);
g.FillRectangle(b1, r1);
```



## 練習 6.1：繪製一個傳統型體溫計

### 目的：

練習如何使用 Graphics 物件繪製一個傳統型體溫計，並以漸層色呈現受測者的體溫。

### 功能描述：

在這個練習中，你將使用 Graphics 物件繪製一個傳統型體溫計，包含體溫計的外框並以純色做為底色，以線性漸層做為目前溫度的呈現。

### 預估實作時間：15 分鐘

### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod06\_1。
3. 在 Form1 的設計畫面中放置一個 Button 控制項。

4. 匯入必要命名空間：

```
Visual Basic
Imports System.Drawing.Drawing2D
```

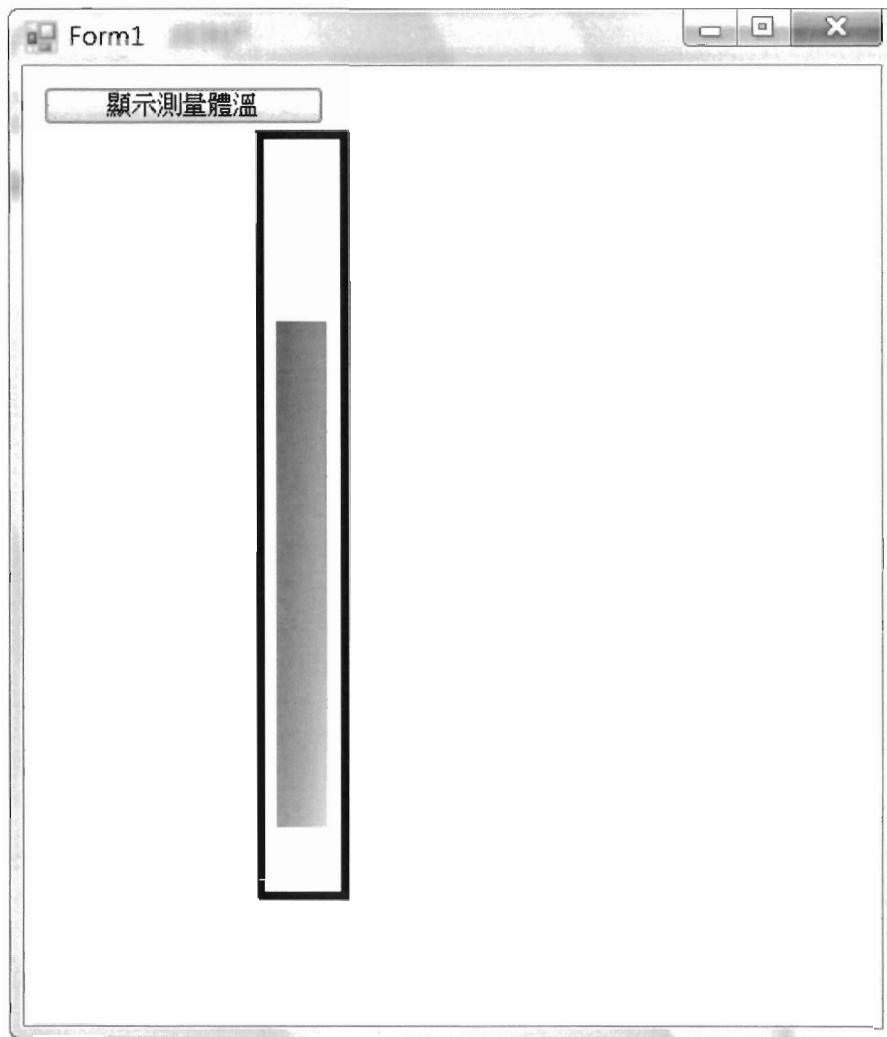
```
C#
using System.Drawing.Drawing2D;
```

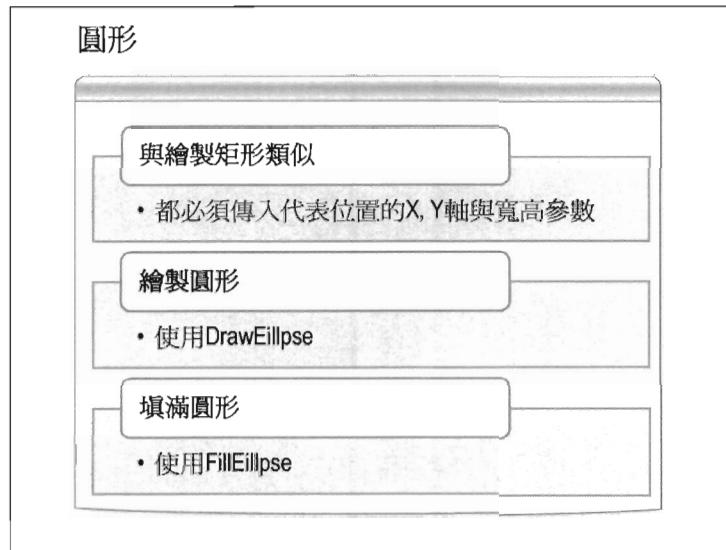
5. 在 Button 的 Click 事件撰寫體溫計的功能，使用 Honeydew 做為背景色，Red 到 Pink 的線性漸層色作為填滿受測者的溫度：

```
Visual Basic
Dim backBrush As New SolidBrush(Color.Honeydew)
Dim borderPen As New Pen(Color.Black, 5)
Dim borderRec As New Rectangle(140, 40, 50, 450)
Dim tempRec As New Rectangle(150, 150, 30, 300)
Dim tempBrush As New LinearGradientBrush(
    tempRec, Color.Red, Color.Pink,
    LinearGradientMode.ForwardDiagonal)
Dim g = Me.CreateGraphics()
g.Clear(Me.BackColor)
g.FillRectangle(backBrush, borderRec)
g.DrawRectangle(borderPen, borderRec)
g.FillRectangle(tempBrush, tempRec)
```

```
C#
SolidBrush backBrush = new SolidBrush(Color.Honeydew);
Pen borderPen = new Pen(Color.Black, 5);
Rectangle borderRec = new Rectangle(140, 40, 50, 450);
Rectangle tempRec = new Rectangle(150, 150, 30, 300);
LinearGradientBrush tempBrush = new LinearGradientBrush(
    tempRec, Color.Red, Color.Pink,
    LinearGradientMode.ForwardDiagonal);
Graphics g = this.CreateGraphics();
g.Clear(this.BackColor);
g.FillRectangle(backBrush, borderRec);
g.DrawRectangle(borderPen, borderRec);
g.FillRectangle(tempBrush, tempRec);
```

6. 執行結果如下：





## 圓形

繪製圓形也包括橢圓形，繪製時的特性是先建立方形範圍，圓形的大小就由方形範圍決定，如以下圖解：



繪製圓形與繪製方形非常類似，只要給定左上角座標以及寬度高度即可，範例程式如下：

### Visual Basic

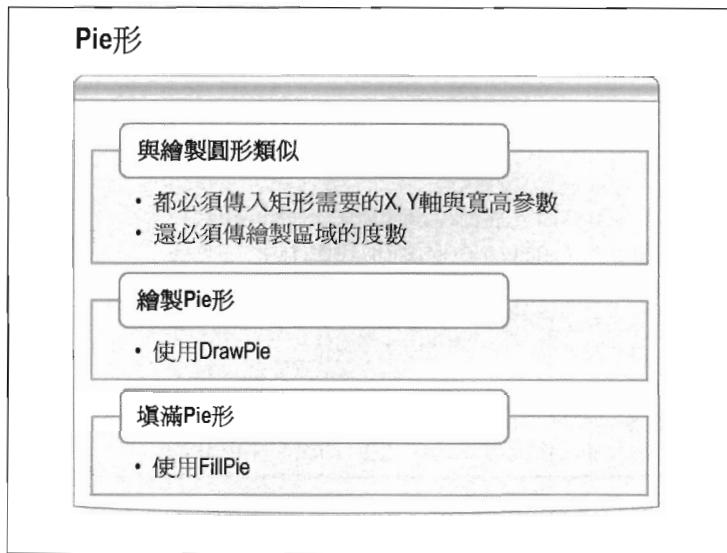
```

Dim rnd As New Random
Dim g As Graphics = Me.CreateGraphics
'清除繪製的圖形
g.Clear(Me.BackColor)
Dim p As New Pen(Color.Blue)
Dim width As Integer = rnd.Next(10, 150)
Dim height As Integer = rnd.Next(10, 150)
'繪製方形寬高範圍，對照冊
g.DrawRectangle(p, 50, 50, width, height)
'繪製圓形
g.DrawEllipse(p, 50, 50, width, height)

```

C#

```
Random rnd = new Random();
Graphics g = this.CreateGraphics();
//清除繪製的圖形
g.Clear(this.BackColor);
Pen p = new Pen(Color.Blue);
int width = rnd.Next(10, 150);
int height = rnd.Next(10, 150);
//繪製方形寬高範圍, 對照用
g.DrawRectangle(p, 50, 50, width, height);
//繪製圓形
g.DrawEllipse(p, 50, 50, width, height);
```



## Pie 形

繪製 Pie 形的方法與繪製圓形類似，都必須傳入矩形需要的 X, Y 軸與寬高參數，及傳入繪製區域的度數。

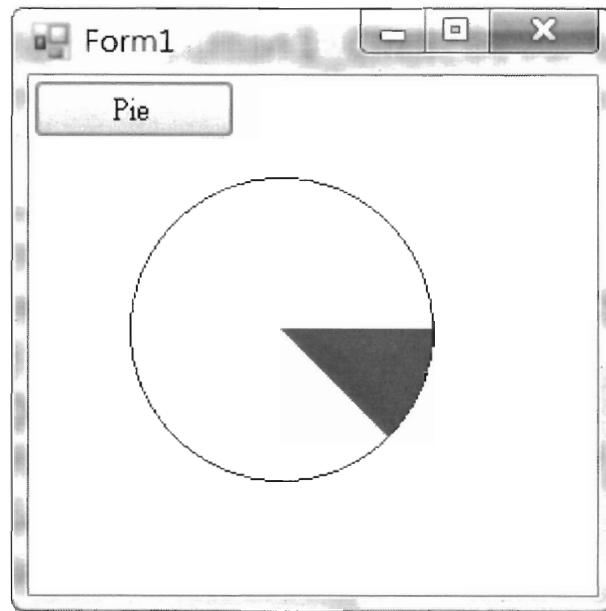
下面範例在圓形中填入一塊 Pie 形，FillPie 方法的第三個參數是 Pie 的起始度數，第四個參數是 Pie 的第 2 個邊角度數。

```
Visual Basic
Dim g As Graphics = Me.CreateGraphics
'清除繪製的圖形
g.Clear(Me.BackColor)
Dim p As New Pen(Color.Blue)
Dim rec As New Rectangle(50, 50, 150, 150)
'繪製圓形
g.DrawEllipse(p, rec)
g.FillPie(Brushes.Red, rec, 0.0F, 45.0F)
```

```
C#
Graphics g = this.CreateGraphics();
//清除繪製的圖形
g.Clear(this.BackColor);
Pen p = new Pen(Color.Blue);
Rectangle rec = new Rectangle (50,50, 150, 150);
//繪製圓形
g.DrawEllipse(p, rec);
```

```
g.FillPie(Brushes.Red, rec, 0.0f, 45f);
```

呈現結果如下圖：





## 文字處理

文字圖形處理包括繪製文字圖形，需建立筆物件並設定字型、大小及顏色，再設定繪製的座標等等，繪製的文字資料也需估算所佔的寬高，以避免與其他繪製的內容重疊或超出繪製範圍。

### 在方形範圍內繪製文字

繪製文字時可以限制文字僅在某個方形區塊中，文字從左自右印出，超過範圍的部份會自動折行，但是底下超過的就看不到了。範例程式如下：

#### Visual Basic

```
Dim g As Graphics = Me.CreateGraphics
g.Clear(Me.BackColor)
Dim r As New Rectangle(50, 50, 100, 100)
Dim s As String = "繪製文字時可以限制文字僅在某個方形區塊中，  
文字從左自右印出，超過範圍的部份會自動折行，但是底下超過的就  
看不到了。"
Dim f As New Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point)
g.DrawString(s, f, Brushes.Blue, r)
g.DrawRectangle(Pens.Red, r)
```

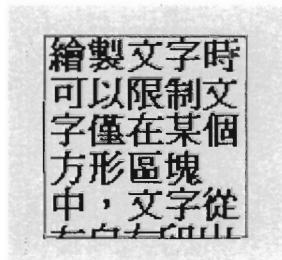
#### C#

```

Graphics g = this.CreateGraphics();
g.Clear(this.BackColor);
Rectangle r = new Rectangle(50, 50, 100, 100);
String s = "繪製文字時可以限制文字僅在某個方形區塊中，文字從左  
自右印出，超過範圍的部份會自動折行，但是底下超過的就看不到了。";
Font f = new Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point);
g.DrawString(s, f, Brushes.Blue, r);
g.DrawRectangle(Pens.Red, r);

```

繪製的結果如下圖：



### 繪製文字對齊

繪製文字時可以使用 `StringFormat` 物件設定對齊方式。範例程式如下：

#### Visual Basic

```

Dim g As Graphics = Me.CreateGraphics
g.Clear(Me.BackColor)
Dim r As New Rectangle(50, 50, 100, 100)
Dim s As String = "文字"
Dim f As New Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point)
Dim sf As New StringFormat()
'sf.Alignment = StringAlignment.Far
'sf.Alignment = StringAlignment.Center
sf.Alignment = StringAlignment.Far
'sf.LineAlignment = StringAlignment.Top
sf.LineAlignment = StringAlignment.Near
g.DrawString(s, f, Brushes.Blue, r, sf)
g.DrawRectangle(Pens.Red, r)

```

#### C#

```

Graphics g = this.CreateGraphics();
g.Clear(this.BackColor);
Rectangle r = new Rectangle(50, 50, 100, 100);

```

```

String s = "文字";
Font f = new Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point);
StringFormat sf = new StringFormat();
//水平對齊, Far=Right,Near=Left,Center=Center
sf.Alignment = StringAlignment.Far;
//垂直對齊, Far=Bottom,Near=Top,Center=Middle
sf.LineAlignment = StringAlignment.Near;
g.DrawString(s, f, Brushes.Blue, r, sf);
g.DrawRectangle(Pens.Red, r);

```

繪製的結果如下圖：



### 繪製文字方向

繪製文字時也可以使用 StringFormat 物件設定 FormatFlags 屬性改變文字方向，修改上一個範例並加入下列範例程式：

#### Visual Basic

```

'文字方向
sf.FormatFlags = StringFormatFlags.DirectionVertical

```

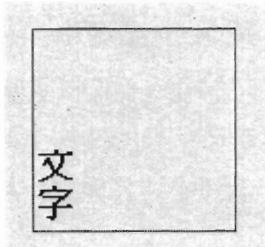
#### C#

```

'文字方向
sf.FormatFlags = StringFormatFlags.DirectionVertical;

```

繪製的結果如下圖：



## 去鋸齒

繪製文字時會很容易看出鋸齒狀的邊線，感覺會比較生硬，如下圖：

原本文字

設定 Graphics 物件的 TextRenderingHint 屬性為 TextRenderingHint.AntiAlias，可以做到除去鋸齒，將字型邊緣填補色點，使其看起來較為平順，範例程式如下：

### Visual Basic

```
Dim g As Graphics = Me.CreateGraphics
g.Clear(Me.BackColor)
g.TextRenderingHint = TextRenderingHint.AntiAlias
g.DrawString("去鋸齒文字", New Font("Arial", 20), Brushes.Black,
50, 100)
```

### C#

```
Graphics g = this.CreateGraphics();
Pen p = new Pen(Color.Blue);
g.TextRenderingHint = TextRenderingHint.AntiAlias;
g.DrawString("去鋸齒文字", new Font("Arial", 20), Brushes.Black,
50, 100);
```

繪製出的文字會較為平順美觀，如下圖：

去鋸齒文字

## 估計繪製文字圖形的寬高

繪製文字時需要在有限的範圍繪製，而因為字型的種類與大小都不相同，且需避免重疊的情況，就需要精密的計算繪製出的文字會佔多大的寬高，以利繪製其他文字或圖形。

計算繪製文字的寬高會使用到 `Graphics` 物件的 `MeasureString` 的方法，參數除了要傳入欲繪製的文字字串與字型物件外，第三個參數需給定繪製區域的範圍寬高，範例程式如下：

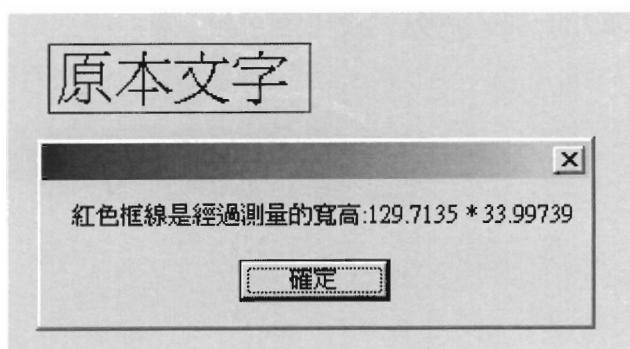
### Visual Basic

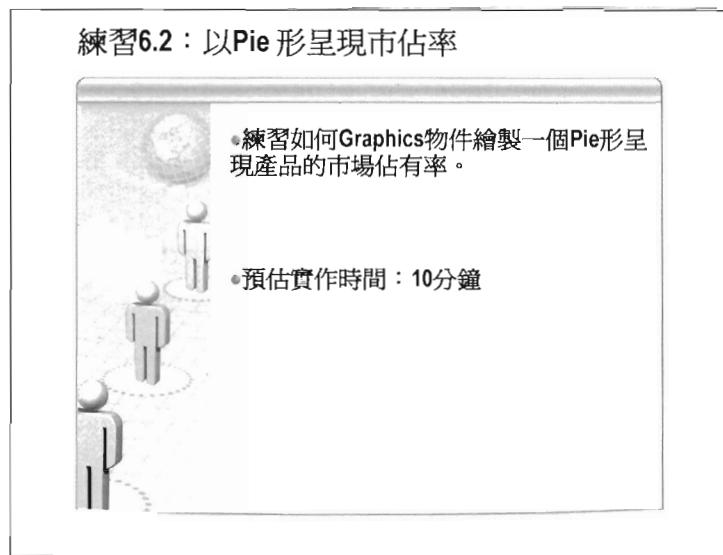
```
Dim g As Graphics = Me.CreateGraphics
g.Clear(Me.BackColor)
Dim f As New Font("Arial", 20)
Dim w As Single = g.MeasureString("原本文字", f, Me.Size).Width
Dim h As Single = g.MeasureString("原本文字", f, Me.Size).Height
g.DrawString("原本文字", f, Brushes.Black, 50, 50)
g.DrawRectangle(Pens.Red, 50, 50, w, h)
MessageBox.Show("紅色框線是經過測量的寬高:" & w.ToString() &
" * " & h.ToString())
```

### C#

```
Graphics g = this.CreateGraphics();
Pen p = new Pen(Color.Blue);
Font f = new Font("Arial", 20);
Single w = g.MeasureString("原本文字", f, this.Size).Width;
Single h = g.MeasureString("原本文字", f, this.Size).Height;
g.DrawString("原本文字", f, Brushes.Black, 50, 50);
g.DrawRectangle(Pens.Red, 50, 50, w, h);
MessageBox.Show("紅色框線是經過測量的寬高:" + w.ToString() +
" * " + h.ToString());
```

繪製的結果如下圖：





## 練習 6.2：以 Pie 形呈現市佔率

**目的：**

練習如何 Graphics 物件繪製一個 Pie 形呈現產品的市場佔有率。

**功能描述：**

在這個練習中，你將使用 Graphics 物件繪製一個 Pie 形呈現產品的市場佔有率。以藍色 Pie 形表示其他公司產品佔有率，以紅格線表示本公司產品佔有率，並且必須凸顯本公司產品佔有率。

**預估實作時間：10 分鐘**

**實作步驟：**

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod06\_2。
3. 在 Form1 的設計畫面中放置一個 Button 按制項。

## 4. 匯入必要命名空間：

```
Visual Basic
Imports System.Drawing.Drawing2D
```

```
C#
using System.Drawing.Drawing2D;
```

## 5. 在 Button 的 Click 事件撰寫產品市場佔有率的 Pie 形圖：

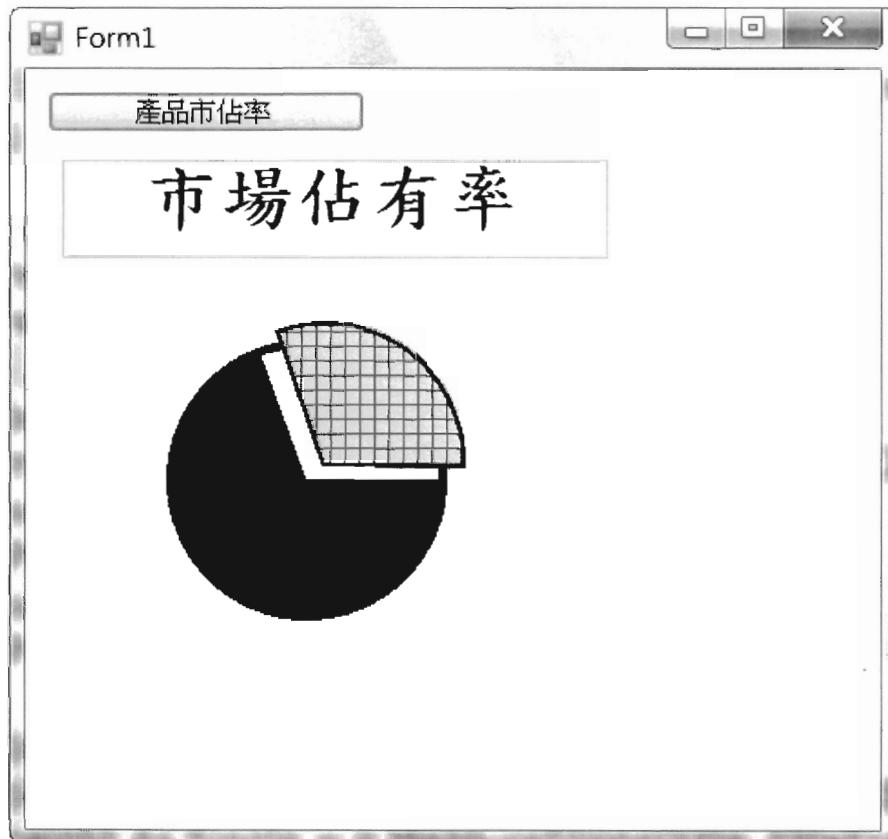
```
Visual Basic
Dim g As Graphics = Me.CreateGraphics()
'清除繪製的圖形
g.Clear(Me.BackColor)
Dim rec1 As New Rectangle(50, 50, 150, 150)
Dim rec2 As New Rectangle(60, 40, 150, 150)
'繪製圓形
Dim borderPen As New Pen(Color.Black, 5)
Dim myprod As New HatchBrush(HatchStyle.Cross, Color.Red, Color.Pink)
g.DrawEllipse(borderPen, rec1)
g.FillPie(Brushes.Blue, rec1, 0.0F, 250.0F)
g.DrawPie(borderPen, rec2, 251.0F, 110.0F)
g.FillPie(myprod, rec2, 251.0F, 110.0F)

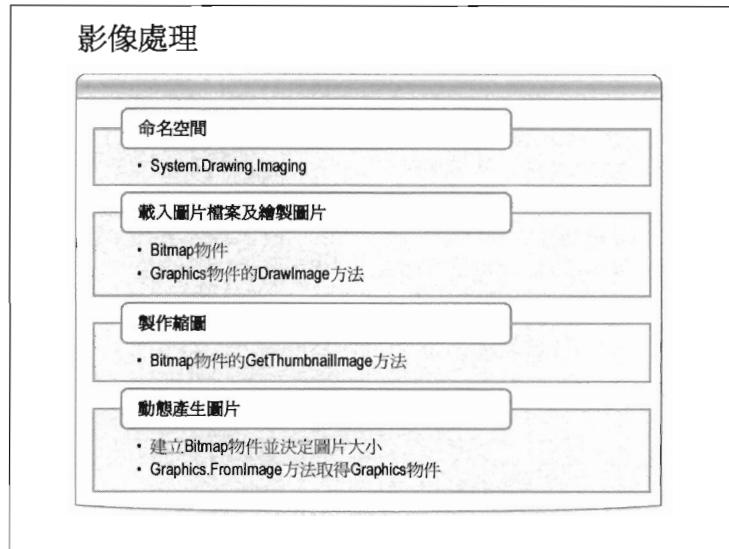
Dim title As String = "市場佔有率"
Dim titleFont As New Font("標楷體", 24, FontStyle.Bold)
Dim height As Single = g.MeasureString(title, titleFont).Height
Dim titleRec As New Rectangle(20, 50, 300, height)
Dim sf As New StringFormat()
sf.Alignment = StringAlignment.Center
sf.LineAlignment = StringAlignment.Center
g.DrawString(title, titleFont, Brushes.Blue, titleRec, sf)
g.DrawRectangle(Pens.SkyBlue, titleRec)
```

```
C#
Graphics g = this.CreateGraphics();
//清除繪製的圖形
g.Clear(this.BackColor);
Pen p = new Pen(Color.Blue);      // 80 %
Rectangle rec1 = new Rectangle(50, 50, 150, 150);
Rectangle rec2 = new Rectangle(60, 40, 150, 150);
//繪製圓形
Pen borderPen = new Pen(Color.Black, 5);
g.DrawEllipse(borderPen, rec1);
g.FillPie(Brushes.Blue, rec1, 0.0f, 250.0f);
g.DrawPie(borderPen, rec2, 251.0f, 110.0f);
HatchBrush myprod = new HatchBrush(HatchStyle.Cross, Color.Red, Color.Pink);
g.FillPie(myprod, rec2, 251.0f, 110.0f);
```

```
string title = "市場佔有率";
Font titleFont = new Font("標楷體", 24, FontStyle.Bold );
float height = g.MeasureString(title, titleFont).Height;
Rectangle titleRec = new Rectangle(20, 50, 300, (int)height);
StringFormat sf = new StringFormat ();
sf.Alignment = StringAlignment.Center;
sf.LineAlignment = StringAlignment.Center;
g.DrawString(title, titleFont, Brushes.Blue, titleRec, sf);
g.DrawRectangle(Pens.SkyBlue, titleRec);
```

6. 執行結果如下：





## 影像處理

影像就是一般我們所說的圖檔或者圖片檔案，它所屬的命名空間是 System.Drawing.Imaging。

### 載入圖片檔案及繪製圖片

文字與圖形的繪製還不夠讓畫面豐富，如果能再結合一些圖片，能讓使用者介面更為親切方便。範例程式如下：

#### Visual Basic

```

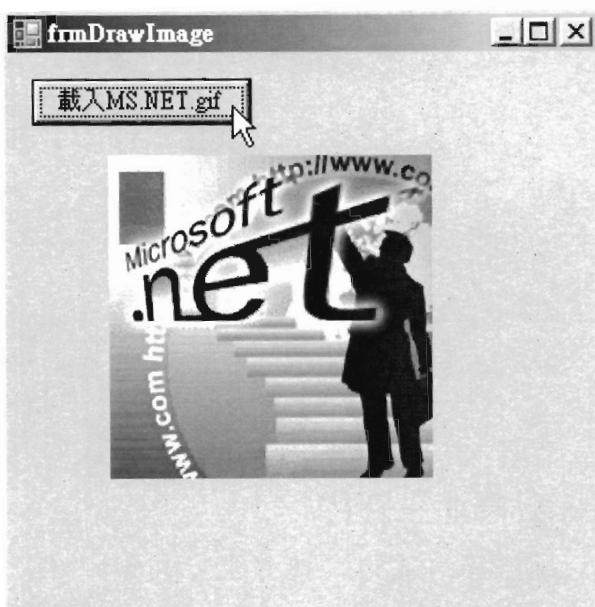
Dim g As Graphics = Me.CreateGraphics
g.Clear(Me.BackColor)
Dim bmp As New Bitmap("MS.NET.gif")
g.DrawImage(bmp, 50, 50)
  
```

#### C#

```

Graphics g = this.CreateGraphics();
g.Clear(this.BackColor);
Bitmap bmp = new Bitmap("MS.NET.gif");
g.DrawImage(bmp, 50, 50);
  
```

繪製的結果如下圖：



## 製作縮圖

當圖片檔案量較多時，通常會做圖片預覽列表的功能，一般稱之為縮圖，程式會使用到 Bitmap 物件的 GetThumbnailImage 方法。範例程式如下：

### Visual Basic

```
Dim g As Graphics = Me.CreateGraphics
g.Clear(Me.BackColor)
Dim bmp As New Bitmap("heroSpot_top.jpg")
Dim pThumbnail As Image = bmp.GetThumbnailImage(100, 100,
Nothing, New IntPtr())
g.DrawImage(pThumbnail, 50, 50, pThumbnail.Width, pThumbnail.
I.Height)
```

### C#

```
Graphics g = this.CreateGraphics();
g.Clear(this.BackColor);
Bitmap bmp = new Bitmap("heroSpot_top.jpg");
Image pThumbnail = bmp.GetThumbnailImage(100, 100, null, ne
w IntPtr());
g.DrawImage(pThumbnail, 50, 50, pThumbnail.Width, pThumbnail.
I.Height);
```

原本的圖片寬高為 787\*379，經過縮圖之後寬高變成 100\*100，所以會有些變形的情況，繪製的結果如下圖：



### 動態產生圖片

動態產生圖片除了要先建立 Bitmap 物件並決定圖片大小以外，最重要的是要使用 Graphics.FromImage 方法取得 Graphics 物件，這樣才能做一些繪製的動作。

繪製完畢之後，建立一個 PictureBox 物件用來展現圖片，另外也可以呼叫 Bitmap 物件的 Save 方法將圖片存檔，範例程式如下：

#### Visual Basic

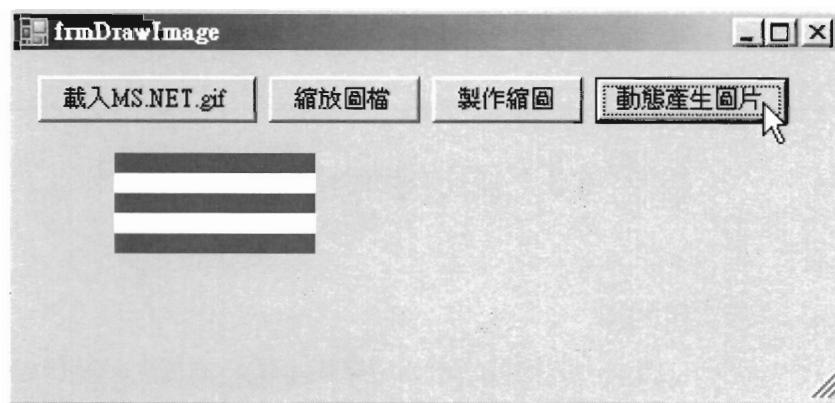
```
'建立圖片物件(準備畫出旗子圖形)
Dim flag As New Bitmap(200, 100)
Dim flagGraphics As Graphics = Graphics.FromImage(flag)
flagGraphics.FillRectangle(Brushes.Red, 0, 0, 200, 10)
flagGraphics.FillRectangle(Brushes.White, 0, 10, 200, 10)
flagGraphics.FillRectangle(Brushes.Red, 0, 20, 200, 10)
flagGraphics.FillRectangle(Brushes.White, 0, 30, 200, 10)
flagGraphics.FillRectangle(Brushes.Red, 0, 40, 200, 10)
flagGraphics.FillRectangle(Brushes.White, 0, 50, 200, 10)
'使用PictureBox控制項展現圖片
Dim pictureBox1 As New PictureBox()
pictureBox1.Location = New Point(50, 50)
Me.Controls.Add(pictureBox1)
pictureBox1.Image = flag
'將繪製出的圖片存檔
flag.Save("flag.jpg", ImageFormat.Jpeg)
```

#### C#

```
//建立圖片物件(準備畫出旗子圖形)
Bitmap flag = new Bitmap(200, 100);
Graphics flagGraphics = Graphics.FromImage(flag);
flagGraphics.FillRectangle(Brushes.Red, 0, 0, 200, 10);
flagGraphics.FillRectangle(Brushes.White, 0, 10, 200, 10);
```

```
flagGraphics.FillRectangle(Brushes.Red, 0, 20, 200, 10);
flagGraphics.FillRectangle(Brushes.White, 0, 30, 200, 10);
flagGraphics.FillRectangle(Brushes.Red, 0, 40, 200, 10);
flagGraphics.FillRectangle(Brushes.White, 0, 50, 200, 10);
//使用PictureBox控制項展現圖片
PictureBox pictureBox1 = new PictureBox();
pictureBox1.Location = new Point(50, 50);
this.Controls.Add(pictureBox1);
//將繪製出的圖片存檔
pictureBox1.Image = flag;
//將繪製出的圖片存檔
flag.Save("flag.jpg", ImageFormat.Jpeg);
```

PictureBox 控制項展現的結果如下圖：



### 練習6.3：網頁動態影像處理

- 將上一個練習的市佔率結果呈現於網頁上。
- 練習如何使用 Bitmap 的 Save 方法將動態圖輸出到網頁

• 預估實作時間：15分鐘

## 練習 6.3：網頁動態影像處理

### 目的：

將上一個練習的市佔率結果呈現於網頁上。練習如何使用 Bitmap 的 Save 方法將動態圖輸出到網頁。

### 功能描述：

在這個練習中，會將上個練習的市佔率結果改由網頁呈現，以便使用者從網頁查看。你將使用 Bitmap 的 Save 方法及 Response 的 OutputStream 將動態圖輸出到網頁，

預估實作時間：10 分鐘

### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Web Site，取名為 Mod06\_3。
3. 加入新項目→Generic Handler→取名為 MarketShare.ashx，記得選取你使用的語言。

4. 汇入必要命名空間：

```
Visual Basic
Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.Drawing.Imaging
```

```
C#
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
```

5. 清除 ProcessRequest 程序裡的程式。

6. 在 ProcessRequest 的程序裡加上以下程式：(建立 Bitmap 物件，並且宣告 Graphics 物件)

```
Visual Basic
Dim bmp As New Bitmap(400, 400)
Dim g As Graphics
g = Graphics.FromImage(bmp)
g.Clear(Color.White)
```

```
C#
Bitmap bmp = new Bitmap(400, 400);
Graphics g = Graphics.FromImage(bmp);
g.Clear(Color.White);
```

7. 從「Practices\ (VB 或 CS)\」開啓 Starter.txt 檔案，將程式碼複製到第 5 步驟程式的下方：

```
Visual Basic
Dim rec1 As New Rectangle(50, 50, 150, 150)
Dim rec2 As New Rectangle(60, 40, 150, 150)
'繪製圓形
Dim borderPen As New Pen(Color.Black, 5)
Dim myprod As New HatchBrush(HatchStyle.Cross, Color.Red, Color.Pink)
g.DrawEllipse(borderPen, rec1)
g.FillPie(Brushes.Blue, rec1, 0.0F, 250.0F)
g.DrawPie(borderPen, rec2, 251.0F, 110.0F)
g.FillPie(myprod, rec2, 251.0F, 110.0F)

Dim title As String = "市場佔有率"
Dim titleFont As New Font("標楷體", 24, FontStyle.Bold)
Dim height As Single = g.MeasureString(title, titleFont).Height
```

```

Dim titleRec As New Rectangle(20, 50, 300, height)
Dim sf As New StringFormat()
sf.Alignment = StringAlignment.Center
sf.LineAlignment = StringAlignment.Center
g.DrawString(title, titleFont, Brushes.Blue, titleRec, sf)
g.DrawRectangle(Pens.SkyBlue, titleRec)

```

C#

```

Pen p = new Pen(Color.Blue);      86 / 10
Rectangle rec1 = new Rectangle(50, 50, 150, 150);
Rectangle rec2 = new Rectangle(60, 40, 150, 150);      90 / 40
//繪製圓形
Pen borderPen = new Pen(Color.Black, 5);
g.DrawEllipse(borderPen, rec1);
g.FillPie(Brushes.Blue, rec1, 0.0f, 250.0f);
g.DrawPie(borderPen, rec2, 251.0f, 110.0f);
HatchBrush myprod = new HatchBrush(HatchStyle.Cross, Color.Red, Color.Pink);
g.FillPie(myprod, rec2, 251.0f, 110.0f);

string title = "市場佔有率";
Font titleFont = new Font("標楷體", 24, FontStyle.Bold );
float height = g.MeasureString(title, titleFont).Height;
Rectangle titleRec = new Rectangle(20, 50, 300, (int)height);
StringFormat sf = new StringFormat ();
sf.Alignment = StringAlignment.Center;
sf.LineAlignment = StringAlignment.Center;
g.DrawString(title, titleFont, Brushes.Blue, titleRec, sf);
g.DrawRectangle(Pens.SkyBlue, titleRec);

```

- 在最後一行設定輸出的文件格式為 image/jpeg，並且使用 Bitmap 的 Save 方法圖案輸出到網頁。

Visual Basic

```

context.Response.ContentType = "image/jpeg"
bmp.Save(context.Response.OutputStream, ImageFormat.Jpeg)

```

C#

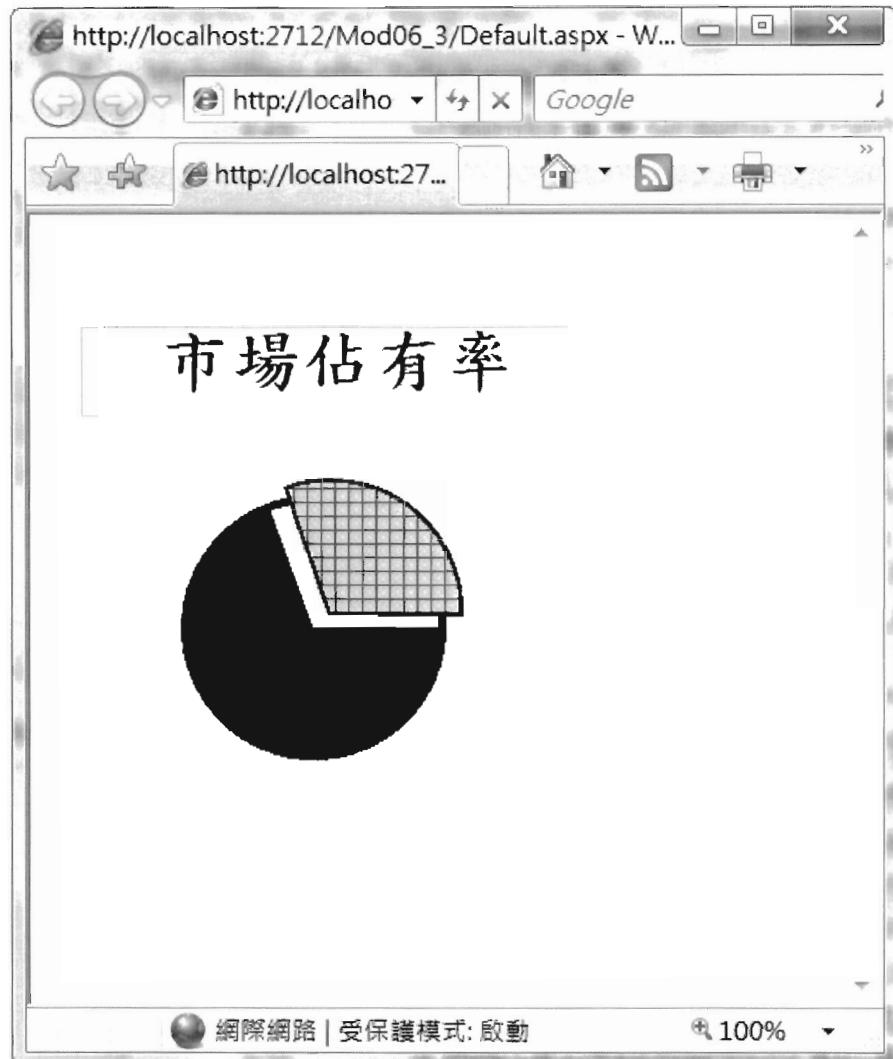
```

context.Response.ContentType = "image/jpeg";
bmp.Save(context.Response.OutputStream, ImageFormat.Jpeg);

```

- 開啓 Default.aspx，然後從 Toolbox 拖拉 Image 物件到設計視窗，設定 ImageUrl 為 MarketShare.ashx。

- 執行網頁。



## 總結

藉由**Graphics**  
物件內建的  
一些方法繪  
製內容

筆刷可以變  
化出多種填  
滿效果

**StringFormat**  
類別提供豐  
富的文字顯  
示格式

**Bitmap**類別  
可以產出多  
種圖像

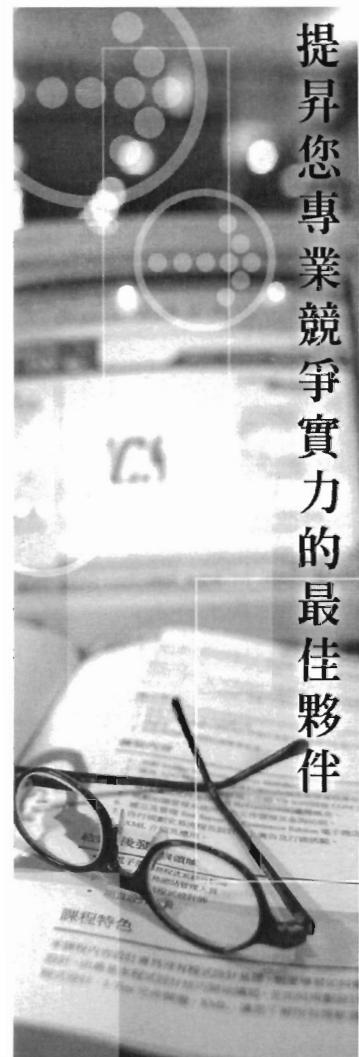
# 第七章：安裝組件與設定組態檔

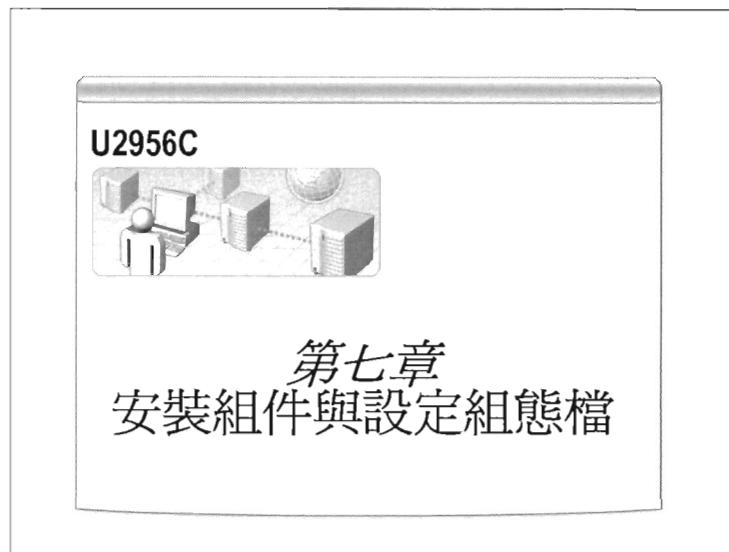
## 本章大綱

組件配置及版本控制.....	4
認識組件.....	5
私有與共享組件.....	7
共享組件.....	8
認識全域組件快取.....	9
組件版本控制.....	10
開發階段共享組件的方式.....	12
.NET Framework 執行階段的版本控制.....	14
練習 7.1：開發階段共享組件的方式.....	16
編修組態檔.....	19
認識組態檔.....	20
一般常用的設定資訊.....	21
專案的 Settings 組態.....	23
存取 Application 範圍的組態項目.....	25
練習 7.2：存取 Settings 組態檔.....	27
使用程式管理組態檔.....	31
如何開啓組態檔.....	33
Application Scope 的組態項目結構.....	38
保護設定檔資料.....	40
練習 7.3：修改應用程式組態檔.....	43
認識 Window Installer.....	48
建立 Setup 專案.....	49
自訂安裝程式.....	50
認識 Installer 類別.....	51
Setup 專案.....	55

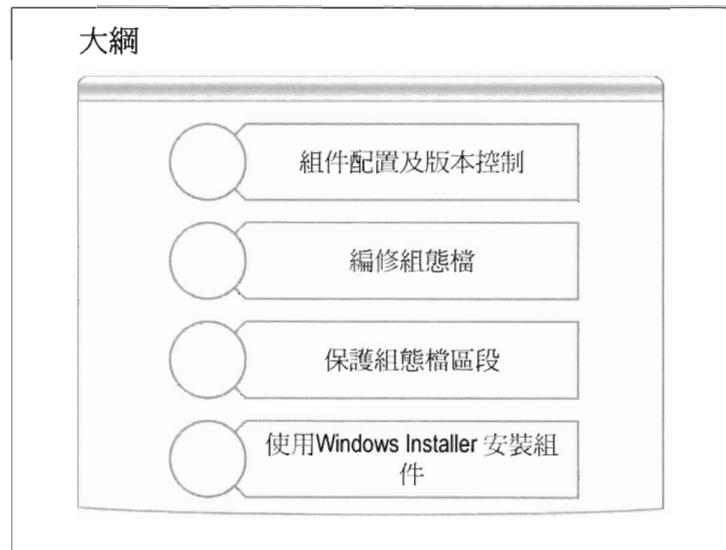
作者：

羅慧真





---

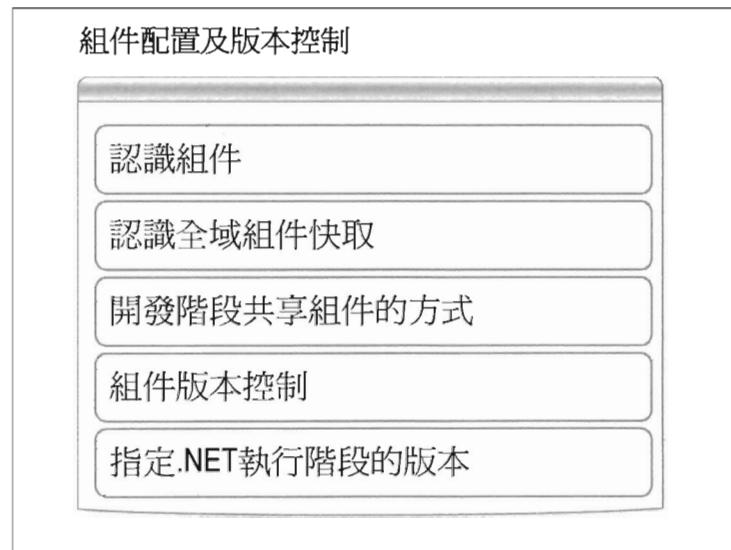


---

在這個章節中將認識什麼是組件，以及如何部署共享組件，同時也將學到如何在程式碼中編修組態檔以及保護組態檔區段，最後你將學習如何使用 Windows Installer 安裝組件以及自訂安裝程序。

本章介紹以下主題：

- 組件配置及版本控制
- 編修組態檔
- 保護組態檔區段
- 使用 Windows Installer 安裝組件

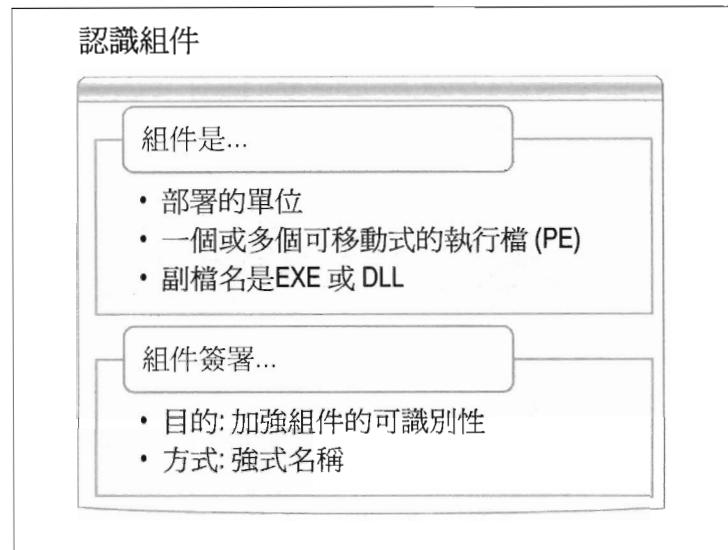


---

## 組件配置及版本控制

這一節中將介紹什麼是組件，如何部署共享組件到全域組件快取 (GAC)，以及開發階段共享組件的方式，共有一下幾個主題：

- 認識組件
- 認識全域組件快取
- 開發階段共享組件的方式
- 組件版本控制
- 指定.NET 執行階段的版本



## 認識組件

組件是.NET Framework 應用程式的一個建置單位，你可以使用 Visual Studio 建立任何類型的專案，一個專案可以產出一個組件，它的副檔名是.EXE 或是.DLL。你可以將產出的 EXE 或 DLL 複製到任何有.NET Framework 的機器上即可執行，它是一個或多個可移動式的檔案 (PE: Portable executable)，亦可將它當作一個部署單位。

一般組件在建置時除了你賦予的組件名稱之外，內部也會有一個簡易的雜湊碼做為識別，可是當組件必須成為共享組件或者需要有特別的安全設定時，為了加強可識別性，必須為組件設定強式名稱。

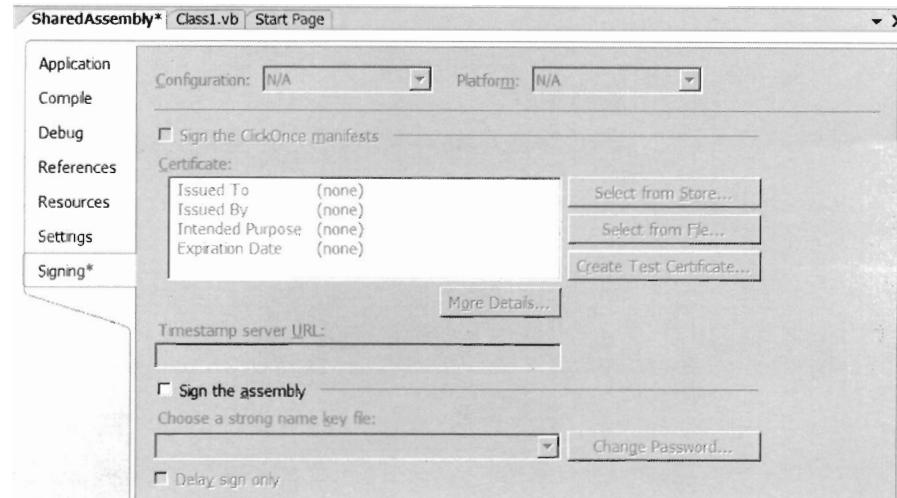
## 如何為組件設定強式名稱

強式名稱組件的設定如下：

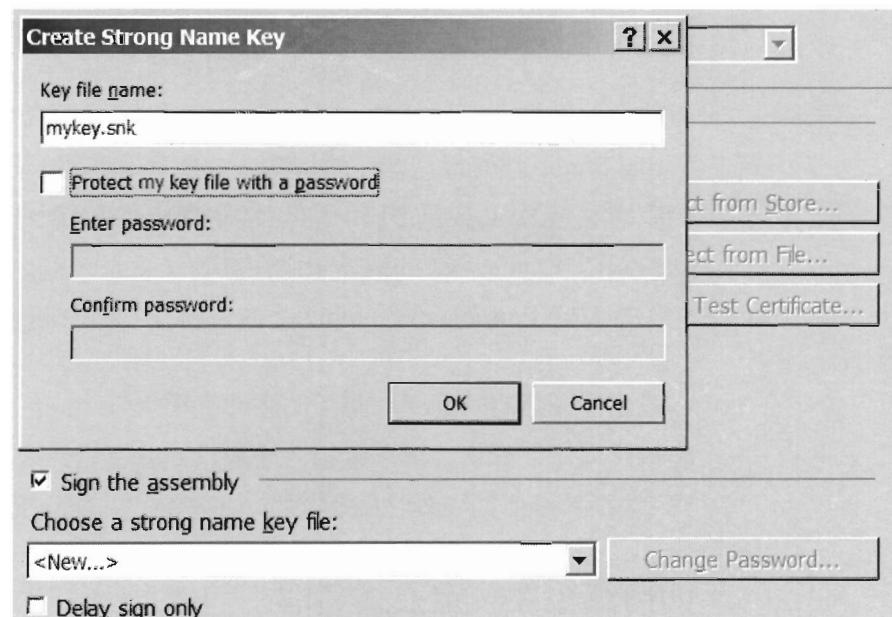
1. Solution Explorer 上選取 My Project 按滑鼠右鍵選取

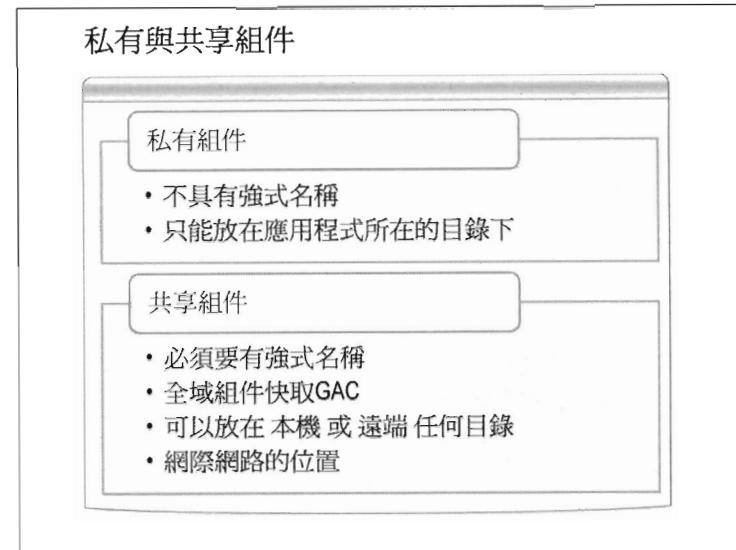
「Open」。

2. 在專案屬性頁上選取(左邊)「Signing」，並勾選「Sign the assembly」。



3. 在「Choose a strong name key file:」下拉項目選取<New...>，接著會出現「Create Strong Name Key」視窗，在此視窗的「Key file name:」上輸入強式名稱金鑰的檔名。





## 私有與共享組件

.NET 所製作的組件預設是私有組件(Private Assembly)，必須與應用程式的組件放在同一層目錄下。而共享組件(Shared Assembly)，則沒有這個限制，通常會將共享組件放在全域組件快取(GAC: Global Assembly Cache 在「系統磁碟:\WINDOWS\assembly」位置下)，不過要成為共享組件必須先將組件製作成強式名稱的組件 (Strong-Name Assembly)。

### 私有組件

私有組件不具有強式名稱，只能放在應用程式所在目錄下，例如，應用程式放在 C:\App1 之下，那麼其參考的私有組件也只能放在 C:\App1 之下，若需要放在之下的子目錄，例如，C:\App1\Bin，則必須寫組態檔讓主程式由組態檔得知到那裡找參考組件，設定如下：

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="bin"/>
    </assemblyBinding>
  </runtime>
</configuration>
```

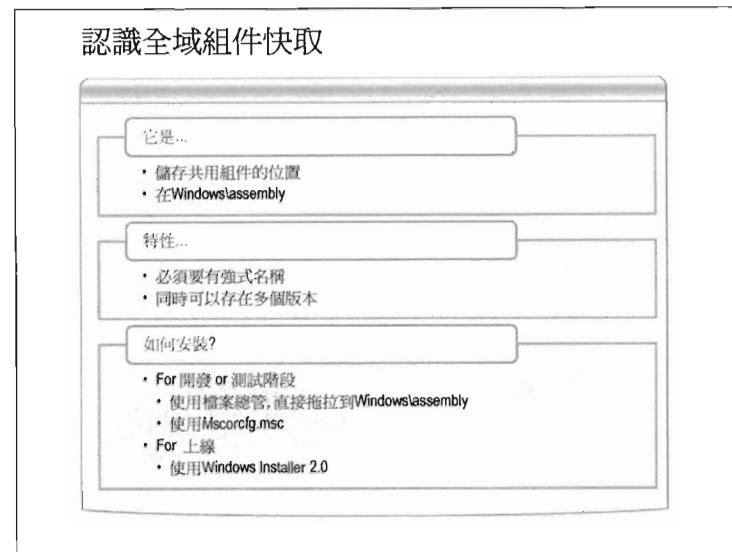
## 共享組件

共享組件首要條件就是要有強式名稱，同時具有強式名稱的組件所參考的也只能是強式名稱的組件。共享組件顧名思義就是它可以供多個應用程式共享，所以可以不限定放在應用程式相同目錄，而是可以放在任何位置，像是：

- 全域組件快取 GAC
- 可以放在本機任何目錄
- 可以放在遠端任何目錄
- 網際網路的位置

應用程式在啓用組件時，不同到天涯海角到處找，所以除了放在 GAC 之外的位置，皆需要寫組態檔讓.NET 的 Runtime 知道到那裡載入組件，例如指定 assembly1.dll 組件從 <http://edu.uuu.com.tw> 的位置下載，組態設定如下：

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="assembly1"
          publicKeyToken="22cc4ba45e0a89b2"
          culture="neutral" />
        <codeBase version="2.0.0.0"
          href="http://edu.uuu.com.tw/assembly1.dll"/>
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```



## 認識全域組件快取

全域組件快取它的簡式名稱是 GAC 完整的英文是 Global Assembly Cache 它放在「系統磁碟:\WINDOWS\assembly」位置下，是本機儲存共用組件的位置。

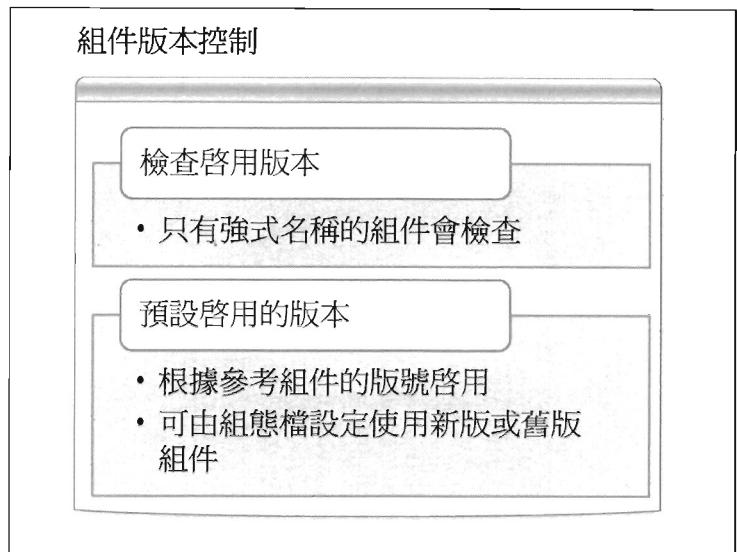
要放在此處的組件一定要有強式名稱，同時它具有 Side-by-Side 的特性，Side-by-Side 是指可以將相同組件多個版本共存於 GAC 下。

事實上，當應用程式參考到強式名稱的組件，它在載入時不僅會檢查組件名稱、強式名稱，還會檢查版本，所以當 GAC 下存有相同組件多個版本時，並不會造成載入混淆的問題，它會載入正確的版本。

如何將組件安裝到 GAC 呢？方式有很多種，在開發或者測試階段可以使用：

- 檔案總管拖拉到系統磁碟:\WINDOWS\assembly 下
- 使用 GACUTIL.exe
- 使用 MMC 工具指到 Mscorcfg.msc

若是要安裝到使用者的平台，最好的選擇是使用 Windows Installer。



## 組件版本控制

當 App1.exe 有參考到 A.dll 組件時，A.dll 組件是強式名稱，App1.exe 會記錄參考時 A.dll 的強式名稱及版本...等資訊。當 A.dll 被要求載入時.NET 的 Runtime 會根據參考組件的強式名稱及版版做載入，所以 App1.exe 當初如果是參考 A.dll 的 1.0.0.0 版，那麼載入的就是 1.0.0.0 版。

當 A.dll 有新版時又不想動新建置 App1.exe 時，可否讓 App1.exe 也參考到新版的 A.dll 呢？

答案是，可以的，不過必須寫組態檔指定重新導向的版本編號。組態設定如下：

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="a"
          publicKeyToken="38cd4ab45e0a69a8"
          culture="neutral" />
        <bindingRedirect oldVersion="1.0.0.0"
          newVersion="2.0.0.0"/>
      </dependentAssembly>
```

```
</assemblyBinding>
</runtime>
</configuration>
```



## 開發階段共享組件的方式

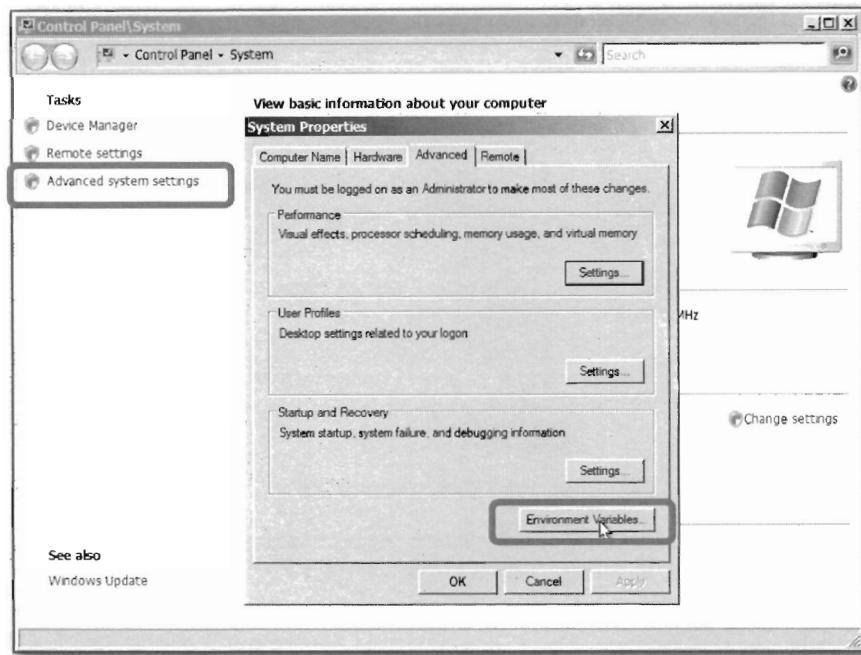
共享組件放在「全域組件快取」是應用程式進行部署時的最後目標，不過在開發階段組件會在修改、建置、測試之間循環，測試之前必須將最新版的組件放到「全域組件快取」，這是一件相當麻煩的事。

在開發階段可以建立 DEVPATH 環境變數取代「全域組件快取」，將 DEVPATH 的環境變數指向共享組件的輸出位置，並且設定以下組態，以告知.NET 的 Runtime 到 DEVPATH 的位置找出組件。

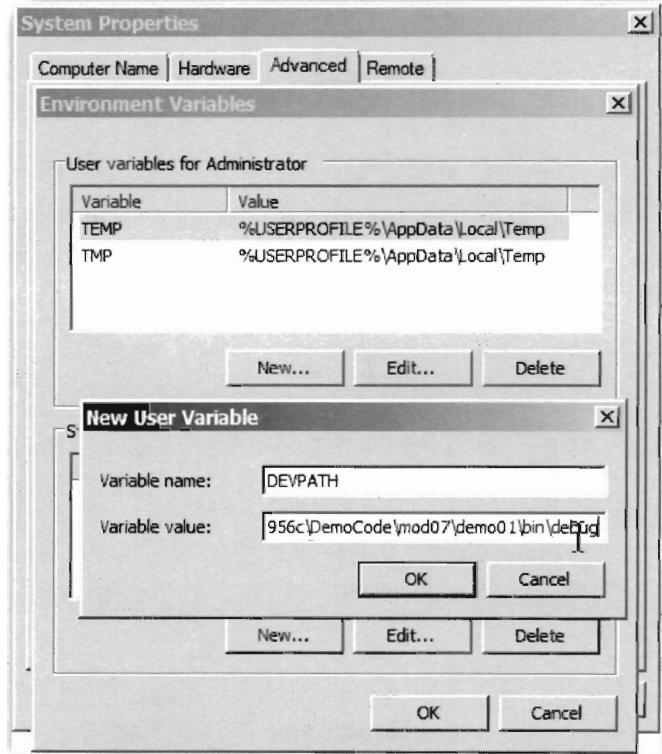
```
<configuration>
<runtime>
  <developmentMode developerInstallation="true"/>
</runtime>
</configuration>
```

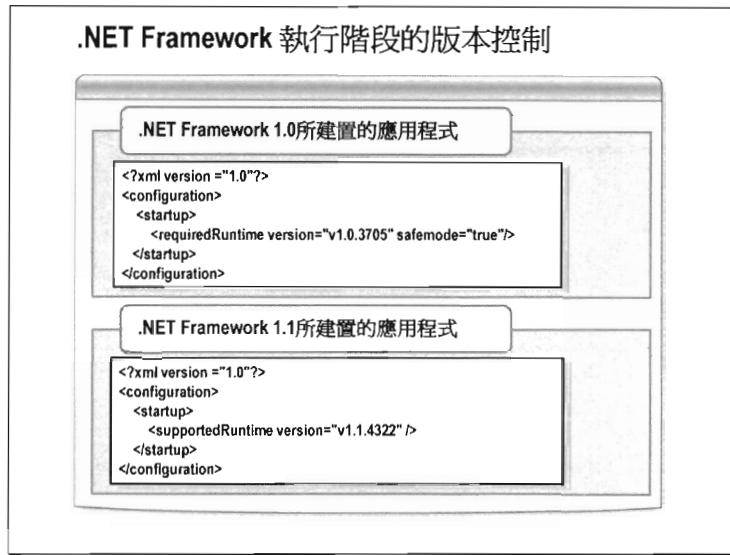
## 如何新增環境變數

1. 開啓「System Properties」視窗，點選「Advanced」頁籤，按「Environment Variables」按鈕。



2. 在「Environment Variables」視窗上選取「New」。
3. 在「New User Variable」視窗輸入變數名稱與變數值即可。





## .NET Framework 執行階段的版本控制

若是使用.NET Framework 較新的版本所建置的應用程式只能在有該版本的環境上執行，原因是程式中可能大量引用新的類別及指令，在舊版的.NET Runtime 中可能不支援。

反過來，使用.NET Framework 1.0 版所建置的應用程式通常可以在.NET Framework 1.0、1.1、2.0 或更新的.NET 環境上執行。相同的.NET Framework 1.1 版所建置的應用程式也通常可以在.NET Framework 1.1、2.0 上執行。這裡說的是「通常可以」，必須經過實際測試才能知道。

當你實際測得應用程式只能在較新版本的.NET Runtime 上執行，就可以在組態檔上限制應用程式的.NET Runtime 版本。.NET Framework 1.0 所建置的應用程式與.NET Framework 1.1 以後所建置的應用程式組態檔並不相同。請看以下說明：

- .NET Framework 1.0 所建置的應用程式，必須使用 requiredRuntime 指定應用程式所支援的.NET Framework 版本，設定如下：

```
<?xml version ="1.0"?>
<configuration>
  <startup>
```

```
<requiredRuntime version=" v1.0.3705" safemode="true"/>
</startup>
</configuration>
```

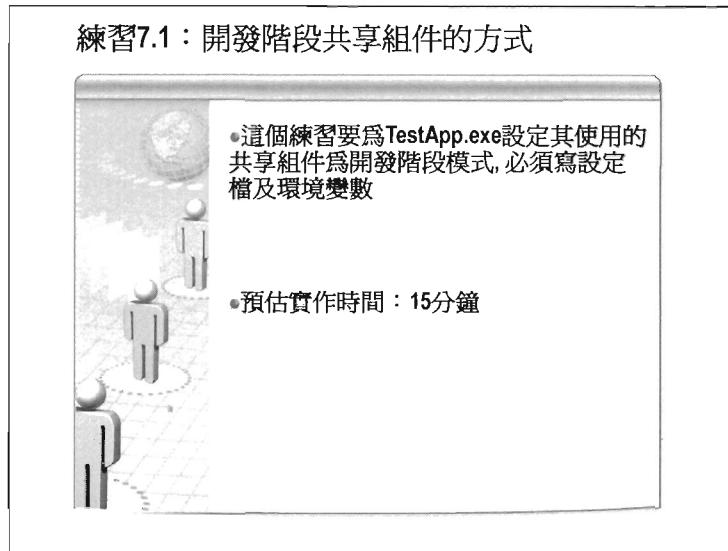
requiredRuntime 的 version 屬性是指定版本，safemode 是指定程式碼是否搜尋註冊機碼以決定 Runtime 版本，預設為 false(尋找註冊機碼)，設為 true 時代表不尋找註冊機碼。

- .NET Framework 1.1(含)以後建置的應用程式，必須使用 supportedRuntime 指定應用程式所支援的 Runtime 版本。

```
<?xml version ="1.0"?>
<configuration>
<startup>
    <supportedRuntime version="vn.n.nnnn" />
</startup>
</configuration>
```

例如，使用.NET Framework 1.1 建置的應用程式，如你經過實際測試，並不相容於.NET Framework 2.0 的環境，這時要指定應用程式只支援.NET Framework 1.1 時，組態如下：

```
<?xml version ="1.0"?>
<configuration>
<startup>
    <supportedRuntime version="v1.1.4322" />
</startup>
</configuration>
```



## 練習 7.1：開發階段共享組件的方式

目的：

這個練習要 TestApp.exe 設定其使用的共享組件為開發階段模式，必須寫設定檔及環境變數。

功能描述：

MySharedAssembly.dll 為共用組件，它還在開發中，仍常常會修改程式，為免每次修改組件都要將新版程式移到 GAC，請設定 DEVPath 及相關組態。

預估實作時間：15 分鐘

實作步驟：

1. 開啓檔案總管，並到「C:\U2956C\Practices\VB 或 CS\Mod07\_1\Starter」目錄，執行 TestApp.exe。
2. 按下表單中的按鈕，結果無法啓動 MySharedAssembly.dll 組件。
3. 在 TestApp.exe 的目錄下，新增一個文字檔檔名為 TestApp.exe.config，並編輯內容：

```
<configuration>
  <runtime>
    <developmentMode developerInstallation="true"/>
  </runtime>
</configuration>
```

4. 開啓「System Properties」視窗，點選「Advanced」頁籤，按「Environment Variables」按鈕。
5. 在「Environment Variables」視窗上選取「New」。
6. 在「New User Variable」視窗在 Variable name 輸入 DEVPATH，在 Variable value 輸入「C:\U2956C\Practices\[VB 或 CS]\Mod07\_1\Starter\Mod07\_1\MySharedLibrary\bin\Debug」。
7. 再執行一次 TestApp.exe，按下表單中的按鈕，結果如下：



8. 結束 TestApp.exe。
9. 開啓 Starter\Mod07\_1\MySharedLibrary\目錄下的專案，並修改 Class1.vb 或 Class1.cs 程式碼如下：

```
Visual Basic
Public Class Class1
  Public Shared Function SayHello() As String
    Return "你好"
  End Function
End Class
```

```
C#
public class Class1
{
    public static string SayHello()
    {
        return "你好";
    }
}
```

10. 存檔及建置 MySharedLibrary 專案。
11. 再執行一次 TestApp.exe，按下表單中的按鈕，結果如下：

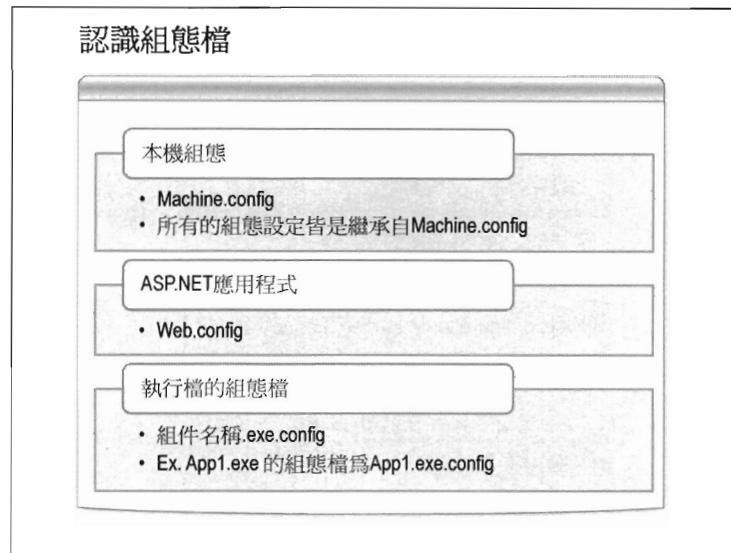




## 編修組態檔

這一節你將認識什麼是組態檔，一般應用程式的資訊例如連線字串、Web 服務的連結位置...等資訊要儲存在組態的何處？以及如何使何使用 ConfigurationManager 存取組態檔，這一節包含以下單元：

- 認識組態檔
- 一般常用的設定資訊
- 專案的 Settings 組態
  - 存取 Application/User 範圍的組態項目
- 使用 ConfigurationManager 管理組態檔
  - 如何開啓組態檔
  - 如何存取組態資訊
  - Application Scope 的組態項目結構
- 保護組態檔區段



## 認識組態檔

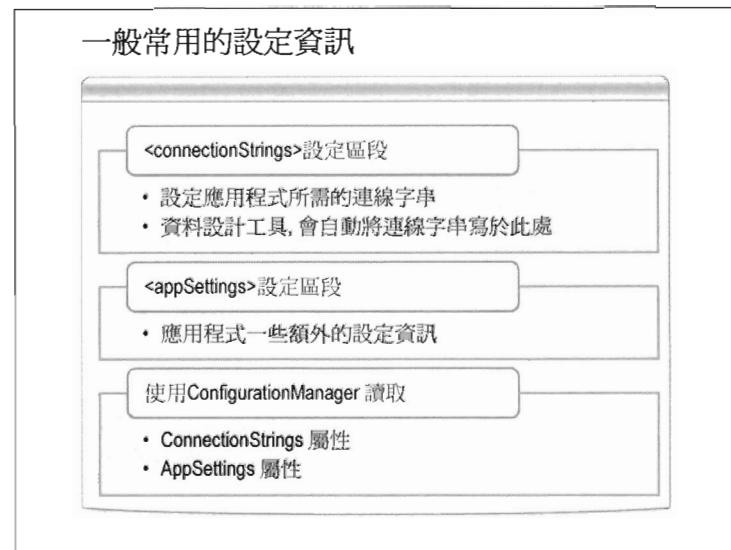
組態檔是一個 XML 格式的文件，利用 XML 標籤設定應用程式所需的配置資訊，像是連線字串、網站的安全驗證模式、授權...等資訊。

所有類型的應用程式都會套用本機層級的組態檔，它的檔名是 Machine.config，它存在

[C:\Windows\Microsoft.NET\Framework\v2.0.50727\Config]的位置，修改此檔時請僅慎因為它將會影響到本機的所有應用程式。

ASP.NET 的組態檔名為 Web.Config 可以存在網站、應用程式目錄、子目錄之下。

一般執行檔的組態檔則必須存在執行檔所在的目錄，而且檔名必須與執行檔名相同加副檔名.config。例如，執行檔名為 App1.exe 那麼組態檔名必須存在相同目錄檔名為 App1.exe.config。



## 一般常用的設定資訊

應用程式中常用的設定組態包含連線字串及一些像是 WebService 位置，總公司名稱、伺服器位置...等資訊。連線字串的組態資訊要放在 `<connectionStrings>` 區段中，其他的則是放在 `<appSettings>` 區段中。

### connectionStrings 區段

連線字串放在組態檔的 `connectionStrings` 區段，Visual Studio 中與資料庫連線有關的設計工具多數會自動將連線字串存在此區段中，開發人員也可以自行設定，組態如下：

```
<connectionStrings>
    <add name="NorthwindConnectionString" connectionString=
        ="Data Source=.\sqlexpress;Initial Catalog=Northwind;Integrated
        Security=True"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

利用程式讀取連線字串：

```
Visual Basic
ConfigurationManager.ConnectionStrings("NorthwindConnectionString").ConnectionString
```

```
C#
ConfigurationManager.ConnectionStrings["NorthwindConnectionString"].ConnectionString;
```

## appSettings 區段

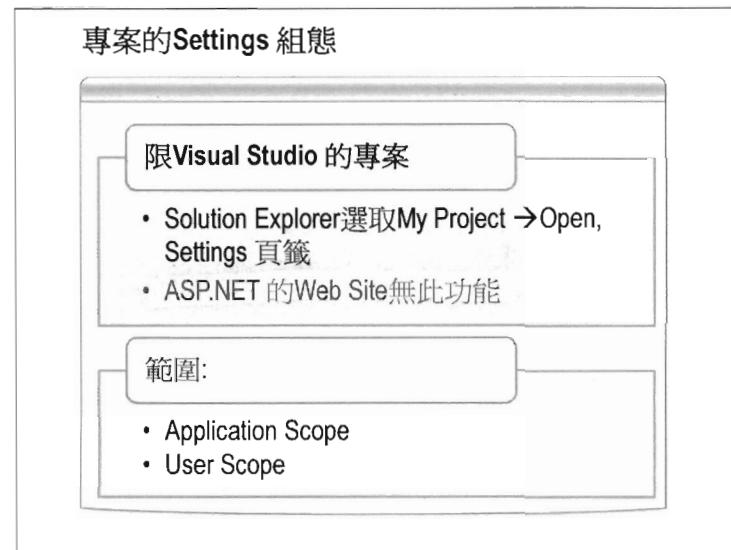
針對常用的應用程式設定，開發者可以將之存放在組態檔中的 appSettings 區段，以 Key-Value 的方式存放：

```
<appSettings>
<add key="DocPath" value="c:\Documents\ " />
<add key="AdSite" value="http://www.netmag.com.tw/" />
</appSettings>
```

利用程式取出應用程式設定，範例如下：

```
Visual Basic
ConfigurationManager.AppSettings("DocPath")
```

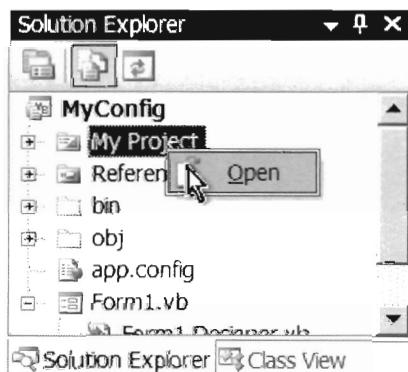
```
C#
ConfigurationManager.AppSettings["DocPath"];
```



## 專案的 **Settings** 組態

Settings 組態僅限於 Visual Studio 的專案類型，像是 Windows 應用程式專案、WPF 應用程式專案，不過 ASP.NET 的 Web Site 沒有 Settings 組態。

Settings 組態包含連線字串、應用程式組態、使用者組態檔等。開啓編輯器的方式是在 Solution Explorer 中選取 My Project 項目，然後按下滑鼠右鍵點選「Open」。



開啓專案的屬性視窗之後，選取「Settings」頁籤，在右方的格子中可以編輯組態檔。組態項目具有強型別的能力，在定義組態項目時必須定義它的型別。

## Scope

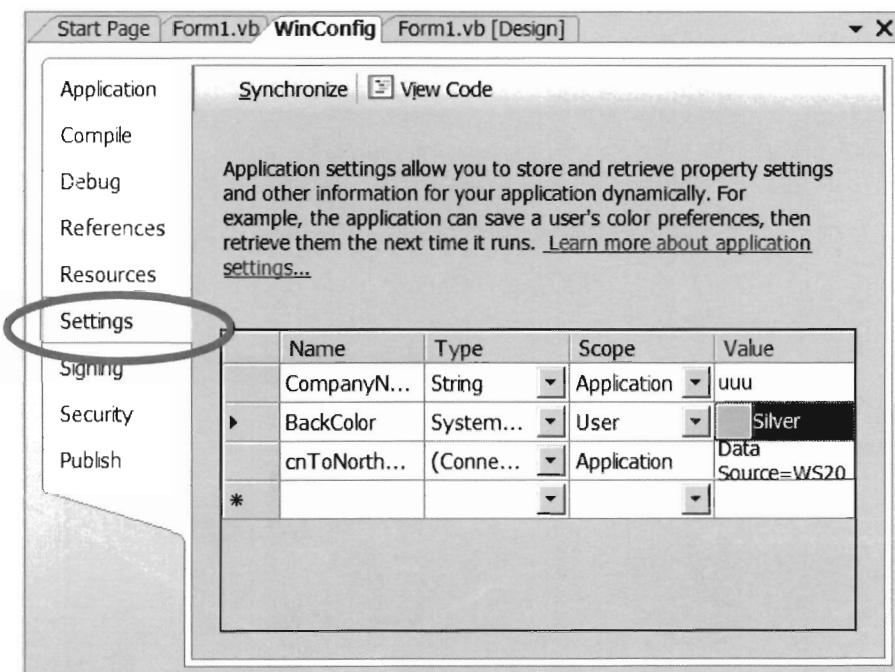
Windows 應用程式的組態項目分成二個存取範圍，一個是 Application，一個是 User。

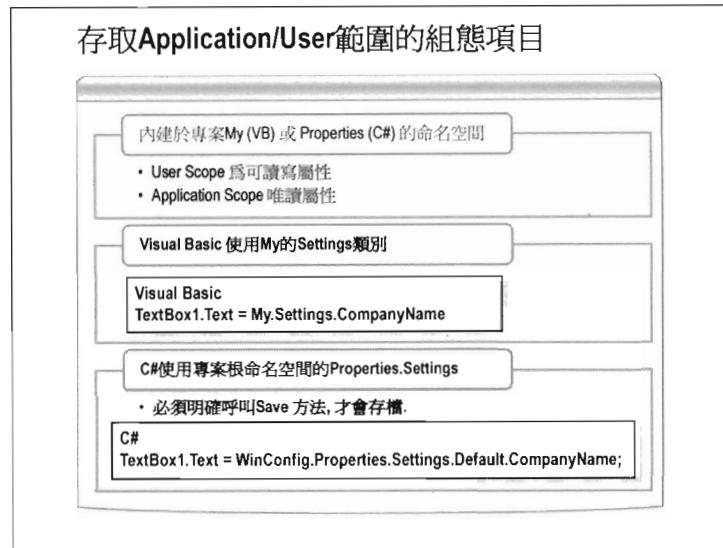
- Application

是指應用程式所有的使用者共享的組態資訊，像是連線字串，應用程式專屬的環境變數。

- User

是指應用程式登入使用者的個別組態資訊，每個使用者可以有自己的組態項目值，並且存在系統的使用者組態檔中  
 「C:\Documents and Settings\[使用者]\Local  
 Settings\Application Data\[公司名稱]\[組件].exe.Url\_[GUID]\版本\user.config」





## 存取 Application 範圍的組態項目

Application 範圍的組態項目是唯讀項目。Visual Basic 與 C# 的存取語法有很大的分別。Visual Basic 在建立組態檔時會在 My 的命名空間下產生 Settings 類別，所以呼叫 My.Settings 即可存取組態項目。

```
Visual Basic
TextBox1.Text = My.Settings.CompanyName
```

C# 在建立組態檔時會在專案的根命名空間下產生一個 Properties 命名空間的 Settings 類別，使用 Settings 的 Default 屬性就可以取得要組態項目。

```
C#
TextBox1.Text =
WinConfig.Properties.Settings.Default.CompanyName;
```

## 存取 User 範圍的組態項目

User 範圍的組態項目是允許讀寫的項目，Visual Basic 使用 My 的命名空間的 Settings 類別，下列範例是讀取設定的值：

Visual Basic

```
Me.BackColor = My.Settings.BackColor  
TextBox1.Text = My.Settings.BackColor.Name
```

下列範例是寫入設定值：

Visual Basic

```
My.Settings.BackColor = Color.FromName(TextBox1.Text)
```

C#使用根命名空間下的 Properties 命名空間的 Settings 類別，下列範例是讀取設定值：

C#

```
this.BackColor = WinConfig.Properties.Settings.Default.BackColor;  
TextBox1.Text =  
    WinConfig.Properties.Settings.Default.BackColor.Name;
```

下列範例是寫入設定值：

C#

```
WinConfig.Properties.Settings.Default.BackColor =  
    Color.FromName(TextBox1.Text);
```

C# 必須呼叫 Save 方法才會存檔。

### 練習7.2：存取Settings組態檔

•瞭解如何使用存取Windows專案的  
Settings組態檔

VB使用【My.Settings】

C#使用【專案名.Properties】

•預估實作時間：15分鐘

## 練習 7.2：存取 Settings 組態檔

### 目的：

瞭解如何使用存取 Windows 專案的組態檔。

### 功能描述：

在這個練習中將學會如何在專案設定組態檔，以及在程式中存取組態檔的設定項目。

**預估實作時間：20 分鐘**

### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod07\_2。
3. 在「Solution Explorer」選取專案名稱，按滑鼠右鍵選取「Properties」進入專案屬性視窗。

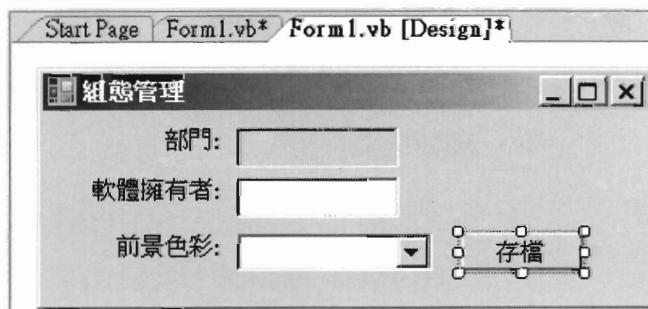
4. 在 Application 頁籤中，找到右方的「Assembly Information...」並按下。
5. 在 Application 頁籤中，找到右方的「Assembly Information...」並按下。進入「Assembly Information」視窗，修改「Company」為 UUU 並按下「OK」。
6. 接著點選「Settings」，設定以下組態項目：

Name	Type	Scope	Value
Department	String	Application	軟體開發部
Owner	String	User	
ForeColor	System.Drawing.Color	User	

7. 開啓 Form1 的設計視窗，從「Toolbox」拖拉控制項到表單，並設定以下屬性：

控制項	屬性	值
Label	ID	Label1
	Text	部門:
TextBox	ID	depTextBox
	ReadOnly	True
Label	ID	Label2
	Text	軟體擁有者:
TextBox	ID	ownerTextBox
Label	ID	Label2
	Text	前景色彩:
ComboBox	ID	foreColorList
	Item	Red Blue Black
Button	ID	Button1
	Text	存檔

畫面安排如下：



8. 寫一個副程式套用所有控制項的前景色彩為組態項目  
ForeColor。

```
Visual Basic
Private Sub ApplyForeColor()
    For Each ctl As Control In Me.Controls
        ctl.ForeColor = My.Settings.ForeColor
    Next
End Sub
```

```
C#
void ApplyForeColor()
{
    foreach (Control ctl in this.Controls)
    {
        ctl.ForeColor = Properties.Settings.Default.ForeColor;
    }
}
```

9. 進入 Form\_Load 事件程序，載入組態項目。

讀取所有組態項目到 TextBox 及 ComboBox 控制項上，並且  
呼叫 ApplyForeColor 方法。

```
Visual Basic
depTextBox.Text = My.Settings.Department
ownerTextBox.Text = My.Settings.Owner
foreColorList.SelectedItem = My.Settings.ForeColor.Name
ApplyForeColor()
```

```
C#
depTextBox.Text = Properties.Settings.Default.Department;
ownerTextBox.Text = Properties.Settings.Default.Owner ;
foreColorList.SelectedItem =
Properties.Settings.Default.ForeColor.Name;
ApplyForeColor();
```

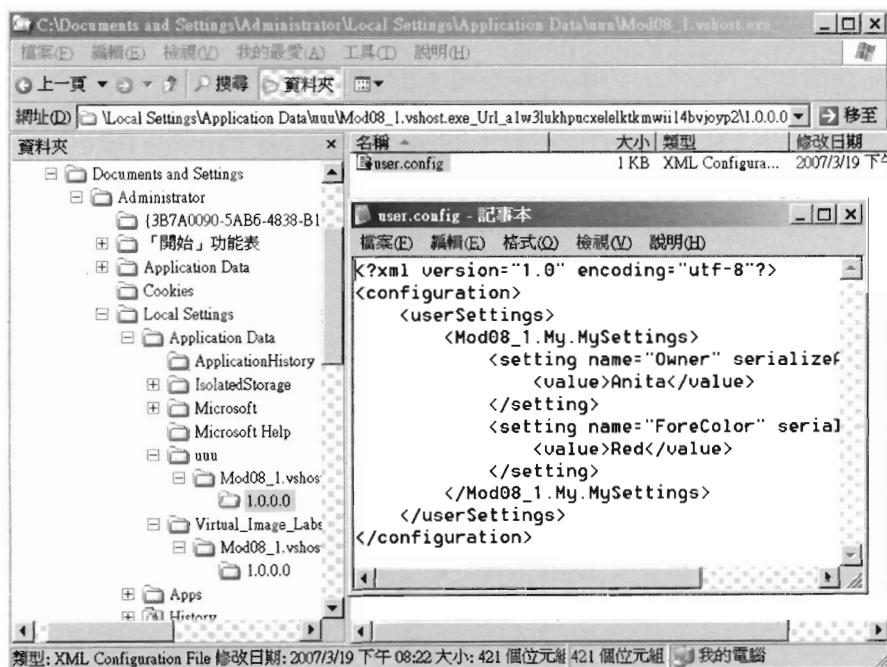
10. 進入 Button1\_Click 事件程序，修改組態項目。

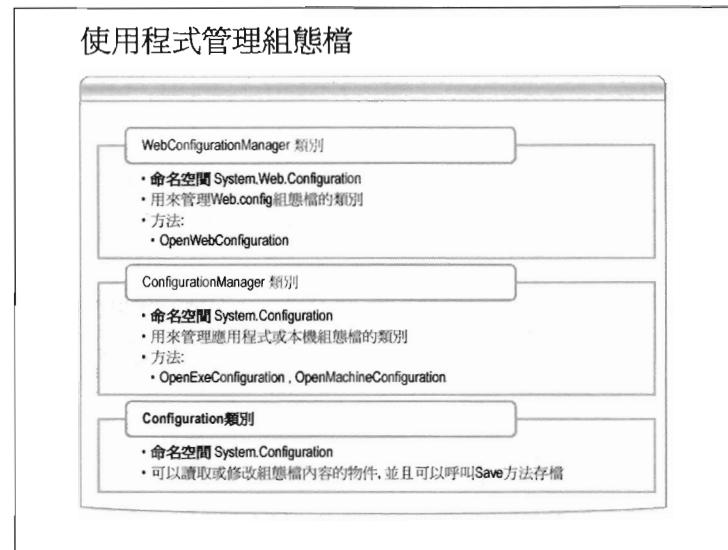
將修改的組態項目存回組態檔。

```
Visual Basic
My.Settings.Owner = ownerTextBox.Text
My.Settings.ForeColor = Color.FromName(foreColorList.Text)
ApplyForeColor()
```

```
C#
Properties.Settings.Default.Owner = ownerTextBox.Text;
Properties.Settings.Default.ForeColor =
    Color.FromName(foreColorList.Text);
Properties.Settings.Default.Save(); // C# 必須明確呼叫Save方法
ApplyForeColor();
```

11. 執行應用程式，應該會看到組態檔初始的狀態，試著修改組態項目，並按下「存檔」按鈕，並結束應用程式(組態項目若有修改會在結束應用程式時存回檔案。C# 必須明確呼叫 Save 方法)
12. 開啓檔案總管，移到「C:\Documents and Settings\Administrator\Local Settings\Application Data\uuu\...」(Windows Server 2003)或「C:\Users\Administrator\AppData\Local\...\...」(Windows Server 2008)之下的子資料夾，直到找到 user.config 檔為止。
13. 使用 Notepad 開啓 user.config 檔，應該要看到修改組態項目的結果。





## 使用程式管理組態檔

組態檔除了可以使用文字編輯器手動編輯之外，也可以利用程式碼進行控制。.NET Framework 提供  `ConfigurationManager` 類別，它是屬於 `System.Configuration` 命名空間，專案必須參考 `System.Configuration.dll`。 `ConfigurationManager` 類別可以讓程式存取本機、應用程式、使用者的組態資訊。

`ConfigurationManager` 類別提供以下幾種常用的方法：

- `OpenExeConfiguration`，開啓執行中應用程式的組態檔。
- `OpenMachineConfiguration`，開啓本機電腦 `Machine.config` 的組態檔。
- `AppSettings`，取得目前應用程式的設定資訊。
- `ConnectionStrings`，取得目前應用程式的連線字串資訊。

在開啓應用程式時可以指定使用者層級，它的列舉常數名稱是 `ConfigurationUserLevel`，包含以下列列舉值：

- `None`，取得套用給全部使用者的組態資訊。
- `PerUserRoaming`，取得套用給目前使用者的漫遊(Roaming)組態資訊。

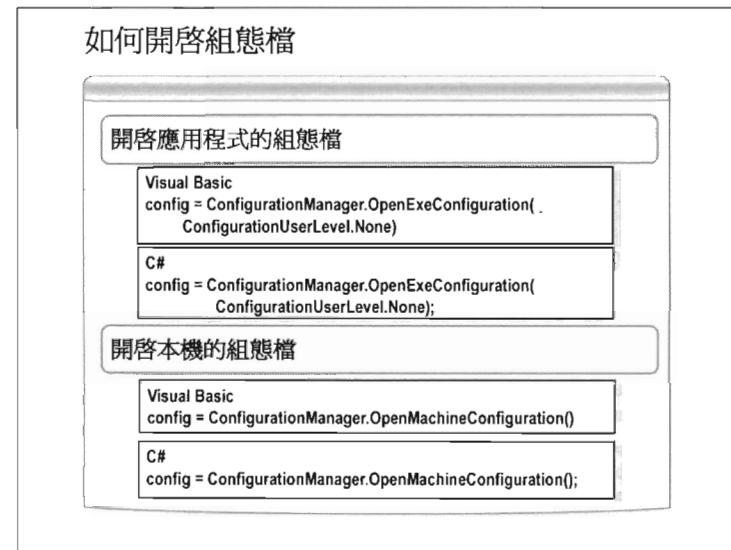
- PerUserRoamingAndLocal，取得套用給目前使用者的本機組態資訊。

註：漫遊(Roamining)在網域(Domain)環境中，使用者允許漫遊於網域中其他電腦，而使用者組態檔會隨著使用者的登入而更新到登入的那台電腦。

另外用來存取 Web.config 的 WebConfigurationManager 類別，它是屬於 System.Configuration 命名空間。可以透過 OpenWebConfiguration 開啓 Web.config 檔。

## Configuration 類別

由 WebConfigurationManager 的 OpenWebConfiguration 方法及 ConfigurationManager 的 OpenExeConfiguration 或 OpenMachineConfiguration 方法回傳的物件。可以利用 Configuration 類別進行組態檔內容的搜尋、讀取、修改以及存檔。



## 如何開啓組態檔

在必要時可以從程式中開啓組態檔(.config)，動態的修改組態檔的內容或者為組態檔中的區段進行加密，保護組態資訊。

開啓應用程式組態檔的步驟如下：

1. 專案須參考 System.Configuration 組件。
2. 匯入 System.Configuration 命名空間。
3. 宣告 Configuration 變數。
4. 呼叫 ConfigurationManager 類別所提供的靜態方法 OpenExeConfiguration。

程式碼如下：

```

Visual Basic
Dim config As Configuration
config = ConfigurationManager.OpenExeConfiguration(
    ConfigurationUserLevel.None)
Label1.Text = config.FilePath

```

```

C#
Configuration config =
    ConfigurationManager.OpenExeConfiguration(

```

```
ConfigurationUserLevel.None);
Label1.Text = config.FilePath;
```

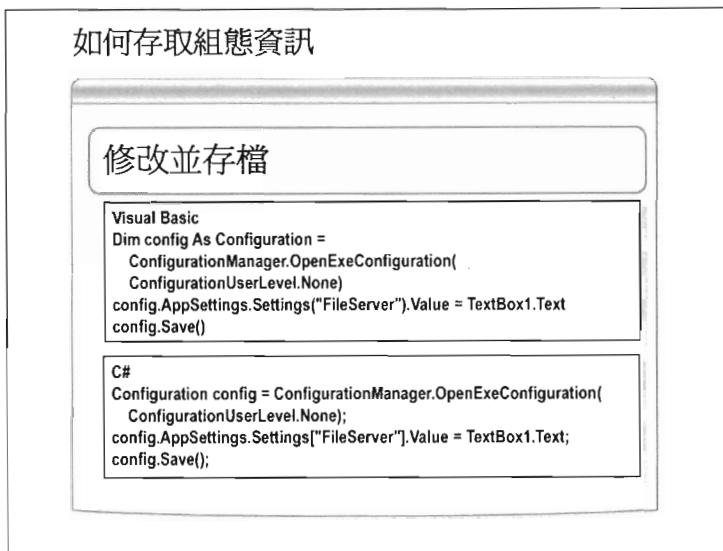
開啟本機的組態檔步驟如下：

1. 專案須參考 System.Configuration 組件。
2. 匯入 System.Configuration 命名空間。
3. 宣告 Configuration 變數。
4. 呼叫 ConfigurationManager 類別所提供的靜態方法  
OpenMachineConfiguration。

程式碼如下：

```
Visual Basic
Dim config As Configuration
config = ConfigurationManager.OpenMachineConfiguration()
Label1.Text = config.FilePath
```

```
C#
Configuration config =
    ConfigurationManager.OpenMachineConfiguration();
Label1.Text = config.FilePath;
```



## AppSettings 區段的組態存取

appSettings 區段在 Windows 應用程式的專案必須自行開啓 app.Config 檔案編輯。用來定義應用程式的環境變數，例如以下區段是定義應用程式的檔案伺服器位置：

```

<appSettings>
    <add key="FileServer" value="instructor"/>
</appSettings>

```

若要讀取 appSettings 區段的值，可以使用 ConfigurationManager 類別的 AppSettings 屬性。

```

Visual Basic
TextBox1.Text = ConfigurationManager.AppSettings("FileServer")

```

```

C#
TextBox1.Text = ConfigurationManager.AppSettings["FileServer"];

```

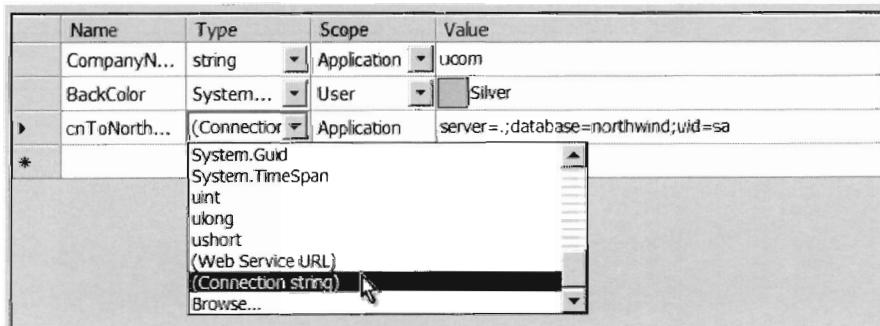
若要修改 appSettings 區段中的值，則必須開啓組態檔，並在修改值之後使用 ConfigurationManager 類別的 Save 方法進行存檔。

```
Visual Basic
Dim config As Configuration
config =
    ConfigurationManager.OpenExeConfiguration(
        ConfigurationUserLevel.None)
config.AppSettings.Settings("FileServer").Value = TextBox1.Text
config.Save()
MessageBox.Show("存檔完成")
```

```
C#
Configuration config ;
config = ConfigurationManager.OpenExeConfiguration(
    ConfigurationUserLevel.None);
config.AppSettings.Settings["FileServer"].Value = TextBox1.Text;
config.Save();
MessageBox.Show("存檔完成");
```

### **connectionStrings** 區段的組態存取

連線字串組態項目通常會在使用與資料庫有關的控制項時產生。如在 Windows 應用程式專案中要編輯連線字串的組態項目，必須在 Settings 頁籤中的 Type 欄位選取(Connection string)，如下圖：



連線字串的組態定義如下：

```
<connectionStrings>
    <add name="MyConfig.Properties.Settings.cnToNorthwind"
        connectionString="server=.;database=northwind;uid=sa" />
</connectionStrings>
```

若要讀取 connectionStrings 區段的值，可以使用 ConfigurationManager 類別的 ConnectionStrings 屬性。

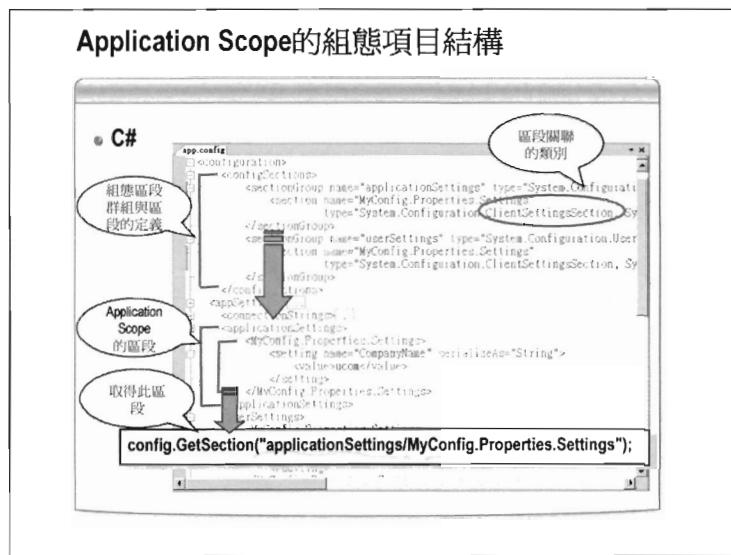
```
Visual Basic  
TextBox1.Text = ConfigurationManager.ConnectionStrings(_  
    "MyConfig.My.MySettings.cnToNorthwind").ConnectionString
```

```
C#  
TextBox1.Text = ConfigurationManager.ConnectionStrings(  
    "MyConfig.My.MySettings.cnToNorthwind").ConnectionString;
```

若要修改 connectionStrings 區段中的值，則必須開啓組態檔，並在修改值之後使用 Configuration 類別的 Save 方法進行存檔。

```
Visual Basic  
Dim config As Configuration  
config = ConfigurationManager.OpenExeConfiguration( _  
    ConfigurationUserLevel.None)  
config.ConnectionStrings.ConnectionStrings("MyConfig.My.MySetti  
ngs.cnToNorthwind").ConnectionString = TextBox1.Text  
config.Save()  
MessageBox.Show("存檔完成")
```

```
C#  
Configuration config ;  
config = ConfigurationManager.OpenExeConfiguration(  
    ConfigurationUserLevel.None);  
config.ConnectionStrings.ConnectionStrings["MyConfig.My.MySetti  
ngs.cnToNorthwind"].ConnectionString = TextBox1.Text;  
config.Save();  
MessageBox.Show("存檔完成");
```



## Application Scope 的組態項目結構

這是 Visual Basic 專案的組態，與 Slide 上的組態稍有不同，不過這不是我們的重點。這裡要說明的是如何修改 Application Scope 的組態項目。



一個新的組態群組或是區段在被使用之前皆必須經過定義過程。在 <configuration>下的<configSection>區段便是在定義

<applicationSettings>及<userSettings>群組區段，以及內在區段的關聯類別。取得此關聯類別可以協助開發人員取得 Application Scope 的組態值，並且可以修改其組態值。

下面這段程式，可以取得 Application Scope，也就是<MyConfig.My.MySettings>區段內容。

**Visual Basic**

```
Dim section As ClientSettingsSection
section = CType(config.GetSection("applicationSettings/MyConfig.
My.MySettings"), ClientSettingsSection)
```

**C#**

```
ClientSettingsSection section;
section = (ClientSettingsSection)config.GetSection("applicationSet
tings/MyConfig.Properties.Settings");
```

若要修改區段內容，透過 Settings 的 Get 方法指定要取得的組態名稱，並使用 Value 屬性取得<value></value>這個區段（SettingValueElement 型別），然後從 ValueXml 寫入要修改的值。

**Visual Basic**

```
Dim value As SettingValueElement = section.Settings.Get("Compa
nyName").Value;
value.ValueXml.InnerText = TextBox1.Text
```

**C#**

```
SettingValueElement value = section.Settings.Get("CompanyName
").Value;
value.ValueXml.InnerText = textBox1.Text;
```

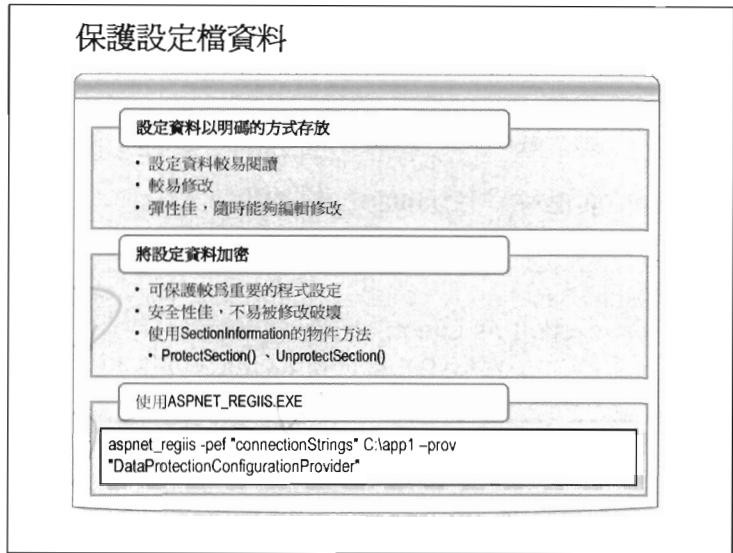
再將修改後的屬性寫回。

**Visual Basic**

```
section.Settings.Get("CompanyName").Value = value
```

**C#**

```
section.Settings.Get("CompanyName").Value = value;
```



## 保護設定檔資料

在設定檔中所有的設定資料都以明碼的方式儲存，尤其像帳號密碼這類資料相當敏感，萬一不小心在開發過程被其他不相干的人看到，又或者有其他因素被竊取，都會造成極大的傷害。

所以有時候開發者會自行撰寫加解密程式，將加密過的資料放入設定檔中，到取用的時候再做解密的動作。

.NET 提供了一個類似開關的方式，可以使用程式動態改變設定檔中某個區段設定的資料變成加密資料，當然也能動態改成不加密。可使用 `SectionInformation` 屬性的 `ProtectSection` 方法加密，使用 `UnprotectSection` 方法解密。

```
Visual Basic
Dim config As Configuration

config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)
Dim info As SectionInformation
info = config.ConnectionStrings.SectionInformation
If info.IsProtected Then
    info.UnprotectSection()
    Button2.Text = "保護"
Else
    info.ProtectSection("DataProtectionConfigurationProvider")
```

```

        Button2.Text = "解除保護"
End If
config.Save()

```

```

C#
Configuration config ;
config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
SectionInformation info ;
info = config.ConnectionStrings.SectionInformation;
if (info.IsProtected )
{
    info.UnprotectSection();
    Button2.Text = "保護";
}
else{
    info.ProtectSection("DataProtectionConfigurationProvider");
    Button2.Text = "解除保護";
}
config.Save();

```

加密前的<connectionStrings>區段如下圖：

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
        </configSections>
    <connectionStrings>
        <add name="Demo04.My.MySettings.cnString"
            connectionString="Data Source=.\sqlexpress;Initial Catalog=Northwind;Integrated Security=True;providerName="System.Data.SqlClient" />
    </connectionStrings>
<system.diagnostics>

```

加密後的<connectionStrings>區段如下圖：

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
        </configSections>
    <connectionStrings configProtectionProvider="DataProtectionConfigurationProvider">
        <EncryptedData>
            <CipherData>
                <CipherValue>AQAAANCmnd8BFdERjHoAwE/Cl+sBAAAAoaK8+ODvaUap3avTi7RC...
            </CipherValue>
        </CipherData>
    </EncryptedData>
    </connectionStrings>
<system.diagnostics>

```

## 呼叫加密服務

可呼叫 ASP.NET 的 Protected configuration 服務，將 Web.config 的部分內容進行加密處理。先進入「Visual Studio 2008 命令提示字元」，

如果以 Visual Studio 2008 的 Web 伺服器而非 IIS，則用「-pef」，並於其後加上 Web.config 設定檔所在的路徑：

```
aspnet_regiis.exe -pef "connectionStrings" C:\Projects\DemoWeb  
-prov "DataProtectionConfigurationProvider"
```

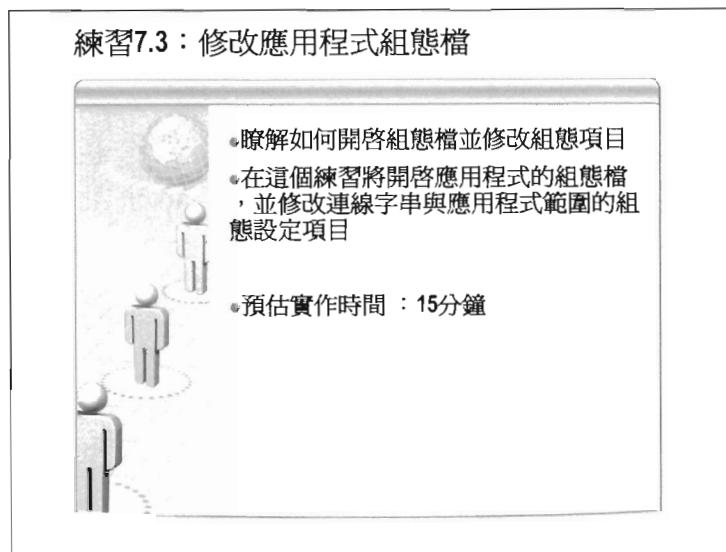
以下說明這兩個參數的差別：

- -pef

指定要加密的區段及 Web.config 設定檔所在的路徑。

- -prov

指定 provider。.NET Framework 有兩種 Provider：  
RSAProtectedConfigurationProvider、  
DataProtectionConfigurationProvider。



## 練習 7.3：修改應用程式組態檔

目的：

瞭解如何開啓組態檔並修改組態項目。

功能描述：

在這個練習將開啓應用程式的組態檔並修改連線字串與應用程式範圍的組態設定項目。

預估實作時間：20 分鐘

實作步驟：

1. 從『Start』→『Program』→『Microsoft Visual Studio 2008』→『Microsoft Visual Studio 2008』，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod07\_3。
3. 在『Solution Explorer』選取專案名稱，按滑鼠右鍵選取『Properties』進入專案屬性視窗。
4. 在 Application 頁籤中，按下右方的『Assembly Information...』。

5. 進入「Assembly Information」視窗，修改「Company」為  
UUU 並按下「OK」。
6. 接著點選「Settings」，設定以下組態項目：

Name	Type	Scope	Value
cnToPubs	(Connection string)	Application	server=.;database=Pubs;uid=sa
InvoiceNo	String	Application	

7. 開啓 Form1 的設計視窗，從「Toolbox」拖拉控制項到表單，並設定以下屬性：

控制項	屬性	值
Label	ID	Label1
	Text	統一發票:
TextBox	ID	InvoiceTextBox
Label	ID	Label2
	Text	連線字串:
TextBox	ID	ConnStringTextBox
Button	ID	Button1
	Text	存檔

8. 在「Solution Explorer」專案名稱的位置按滑鼠右鍵選取「Add Reference」，選取 System.Configuration，按「OK」。
9. 進入 Form1 程式碼視窗，匯入命名空間：

```
Visual Basic
Imports System.Configuration
```

```
C#
using System.Configuration;
```

10. 進入 Form\_Load 事件程序，載入組態項目。讀取所有組態項目到 TextBox。

```
Visual Basic
InvoiceTextBox.Text = My.Settings.InvoiceNo
ConnStringTextBox.Text = My.Settings.cnToPubs
```

```
C#
InvoiceTextBox.Text = Properties.Settings.Default.InvoiceNo;
ConnStringTextBox.Text = Properties.Settings.Default.cnToPubs;
```

11. 進入 Button1\_Click 事件程序，將修改的組態項目存回組態檔。

#### Visual Basic

- 使用 ConfigurationManager 的 OpenExeConfiguration 開啓應用程式組態檔。
- 連線字串"Mod07\_3.My.MySettings.cnToPub"。

```
Dim config As Configuration
config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)
config.ConnectionStrings.ConnectionStrings("Mod07_3.My.MySettings.cnToPubs").ConnectionString = ConnStringTextBox.Text
```

- 取得 Application Scope 區段內的組態值並修改其值。

```
Dim section As ClientSettingsSection
section = CType(config.GetSection("applicationSettings/Mod07_3.My.MySettings"), ClientSettingsSection)

Dim value As SettingValueElement
value = section.Settings.Get("InvoiceNo").Value
value.ValueXml.InnerText = InvoiceTextBox.Text
section.Settings.Get("InvoiceNo").Value = value
```

- 存回組態檔。

```
config.Save()
```

#### C#

- 使用 ConfigurationManager 的 OpenExeConfiguration 開啓應用程式組態檔。
- 連線字串"Mod07\_3.Properties.Settings.cnToPubs"。

```
Configuration config
    = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
config.ConnectionStrings.ConnectionStrings["Mod07_3.Properties.Settings.cnToPubs"].ConnectionString = ConnStringTextBox.Text;
```

- 取得 Application Scope 區段內的組態值並修改其值。

```
ClientSettingsSection section
= (ClientSettingsSection)config.GetSection(@"applicationSettings/Mod07_3.Properties.Settings");
```

```
SettingValueElement value = section.Settings.Get("InvoiceNo").Value;
value.ValueXml.InnerText = InvoiceTextBox.Text;
section.Settings.Get("InvoiceNo").Value = value;
```

- 存回組態檔。

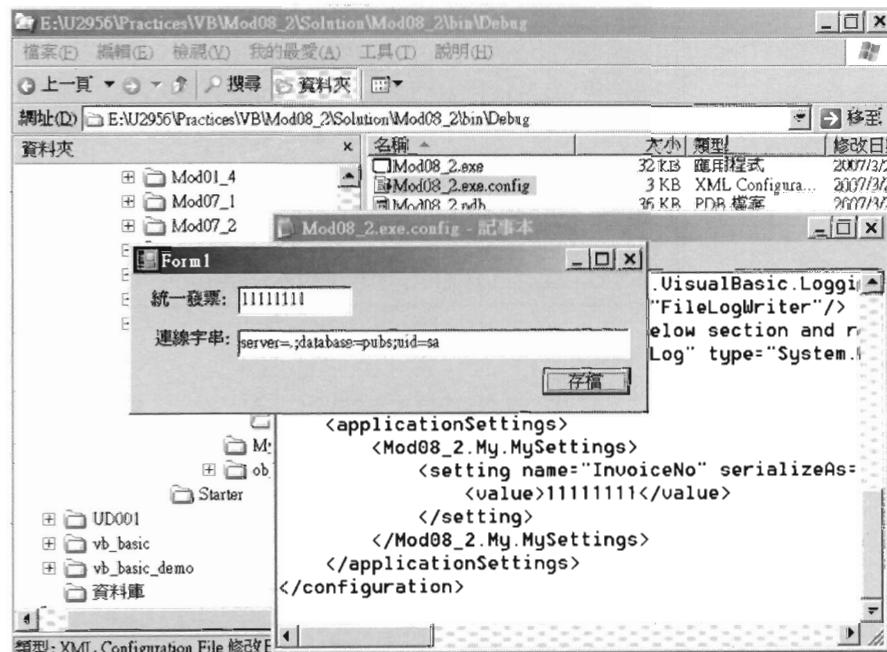
```
config.Save();
```

12. 建置專案。

13. 開啓檔案總管移到「Practices\VB 或

CS\Mod07\_3\Solution\Mod07\_3\bin\Debug」目錄，執行  
Mod07\_3.exe，修改「統一發票」及「連線字串」，然後按  
「存檔」。

14. 使用 Notepad 開啓 Mod07\_3.exe.config 檔。

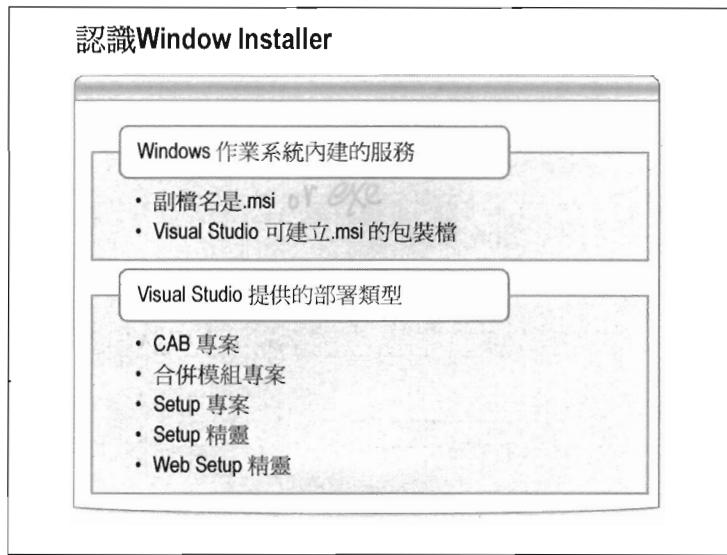




---

這一節將介紹如何使用 Windows Installer 包裝組件。Visual Studio 提供 Setup 專案類型用來提供包裝應用程式功能，如果包裝程式有另外的自訂程序，它也容許程式人員自訂安裝程序。這一節包含以下內容：

- 認識 Window Installer
- 建立 Setup 專案
- 建立自訂安裝程式

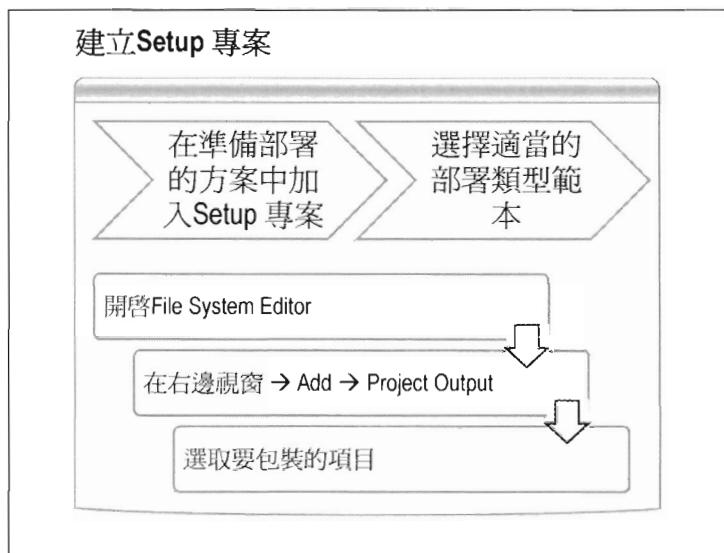


## 認識 Window Installer

Windows Installer 是 Windows 2000、XP 之後版本(含)所內建的服務，可以與 AD 服務整合做軟體派送的功能，它的副檔名是.msi。Visual Studio 也有提供 Windows Installer 包裝應用程式功能，可以產生 Setup.exe 及.msi 檔。

Visual Studio 的安裝專案類型包含：

- Setup 專案，包裝一般應用程式。
- Setup 精靈，以精靈步驟包裝一般應用程式。
- Web Setup 專案，包裝 Web 應用程式。
- Web Setup 精靈，以精靈步驟包裝 Web 應用程式。
- CAB 專案，將應用程式包裝成一個 CAB 壓縮檔，以便於網頁上傳送。
- 合併模組專案，將多個元件包裝成一個合併模組，合併模組為.msm 副檔，以便於合併到一般的 Setup 專案。

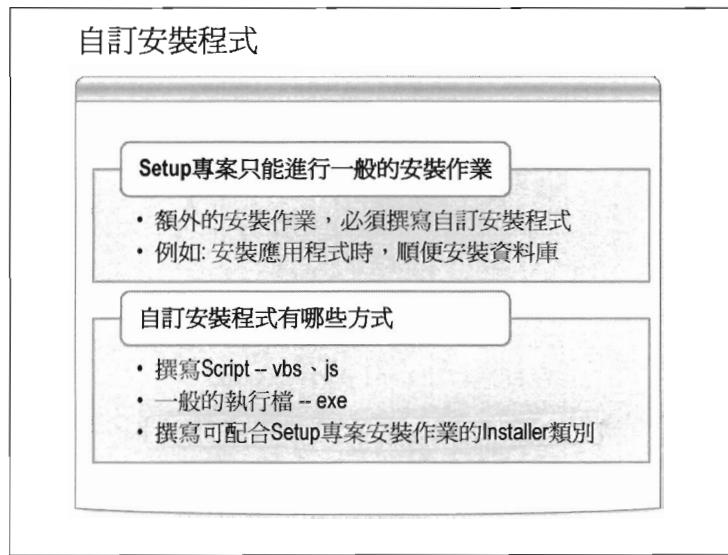


## 建立 Setup 專案

首先先開啓準備部署的方案，然後在即有的方案中加入 Setup 專案，在 Setup 專案中選擇適當的部署類型範本。

建立好 Setup 專案之後：

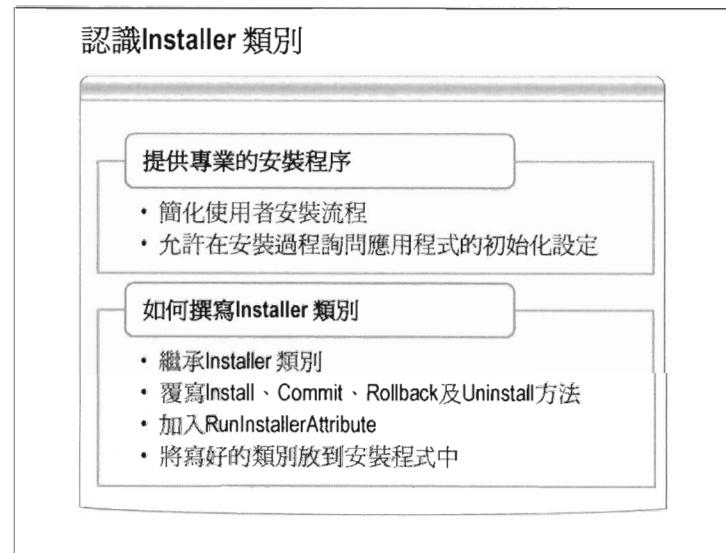
1. 在 File System Editor 視窗開啓 Application Folder
2. 在其右方視窗按滑鼠右鍵選取 Add → Project Output
3. 在 Project Output 視窗選取要包裝的項目。



## 自訂安裝程式

Setup 專案可以進行一般應用程式、Web 應用程式...等安裝程式的包裝作業，但常常除了將應用程式複製到目的端之外，常有些額外的安裝作業，這些額外安裝的作業程序，必須由開發人員自行撰寫 Installer 類別。像是很多應用程式還需順便建立資料庫，進行資料庫設定等程序。

一般開發人員會將這些自訂安裝程序使用 Script(vbs 或 js)、或是執行檔(Exe)、動態連結檔(Dll)。通常較複雜的程序撰寫類別並繼承 Installer 類別來搭配 Setup 專案進行安裝程序。



## 認識 Installer 類別

Installer 類別是自訂安裝程式的基底類別，隨著自訂安裝的需求可以覆寫以下類別。

- Install，在安裝程序時進行的程式。
- Commit，在完成安裝程序並確認安裝成功時進行的程式。
- Rollback，在安裝失敗後將環境復原到原狀時所需進行的程式。
- Unintall，在移除程式時進行的程式。

Installer 的延伸類別可以透過安裝程式工具 Installutil.exe 及 Setup 專案上執行。

## RunInstallerAttribute

這個 Attribute 是用來指定 Installer 的延伸類別是否透過 Installutil.exe 及 Setup 專案執行安裝程式。如果 RunInstallerAttribute 設為 true 時，安裝程式可以讓 Installutil.exe 及 Setup 專案進行安裝作業，預設是 false。

## 如何撰寫 Installer 類別

1. 建立 Class Library 專案。
2. 加入參考(Add Reference)項目 System.Configuration.Install
3. 加入類別
4. 繼承 Installer

```
Visual Basic
Imports System.Configuration.Install
Imports System.ComponentModel
Public Class AppInstaller
    Inherits Installer
End Class
```

```
C#
using System.Configuration.Install;
using System.ComponentModel;
public class AppInstaller:Installer
{
}
```

*System.Windows.Forms  
Optional*

5. 在類別上加上 RunInstallerAttribute

```
Visual Basic
<RunInstaller(True)> Public Class AppInstaller
    '程式碼省略
End Class
```

```
C#
[RunInstaller(true)]public class AppInstaller:Installer
{
    //程式碼省略
}
```

6. 覆寫 Install 方法。可以使用 Context 物件的 Parameters 集合取得 Setup 專案 Custom Actions 的 CustomActionData 屬性中的值。

```
Visual Basic
Public Overrides Sub Install(ByVal stateSaver As System.Collections.IDictionary)
    MyBase.Install(stateSaver)
    MessageBox.Show("進行Install程序" & vbCrLf &_
        "擁有者:" & Me.Context.Parameters.Item("owner"))
End Sub
```

```
C#
    public override void Install(System.Collections.IDictionary state
Saver)
    {
        base.Install(stateSaver);
        MessageBox.Show("進行Install程序\r\n" +
            "擁有者:" + this.Context.Parameters ["owner"]);
    }
```

7. 覆寫 Commit 方法。可以使用 Context 物件的 Parameters 集合取得 Setup 專案 Custom Actions 的 CustomActionData 屬性中的值。

```
Visual Basic
    Public Overrides Sub Commit(ByVal savedState As System.Collections.IDictionary)
        MyBase.Commit(savedState)
        MessageBox.Show("進行Commit程序" )
    End Sub
```

```
C#
    public override void Commit(System.Collections.IDictionary savedState)
    {
        base.Commit(savedState);
        MessageBox.Show("進行Commit程序");
    }
```

8. 覆寫 Rollback 方法。可以使用 Context 物件的 Parameters 集合取得 Setup 專案 Custom Actions 的 CustomActionData 屬性中的值。

```
Visual Basic
    Public Overrides Sub Rollback(ByVal savedState As System.Collections.IDictionary)
        MyBase.Rollback(savedState)
        MessageBox.Show("進行Rollback程序" )
    End Sub
```

```
C#
    public override void Rollback(System.Collections.IDictionary savedState)
    {
        base.Rollback(savedState);
    }
```

```
    MessageBox.Show("進行Rollback程序");
}
```

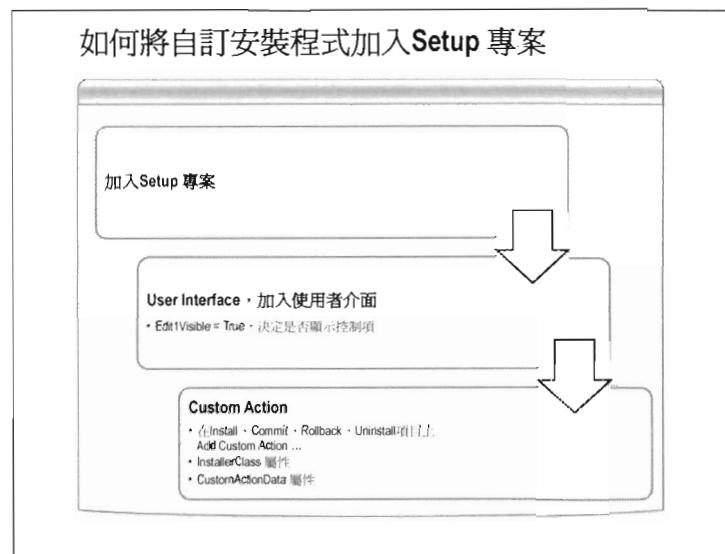
9. 覆寫 Uninstall 方法。可以使用 Context 物件的 Parameters 集合取得 Setup 專案 Custom Actions 的 CustomActionData 屬性中的值。

**Visual Basic**

```
Public Overrides Sub Uninstall(ByVal savedState As System.Collections.IDictionary)
    MyBase.Uninstall(savedState)
    MessageBox.Show("進行Uninstall程序" )
End Sub
```

**C#**

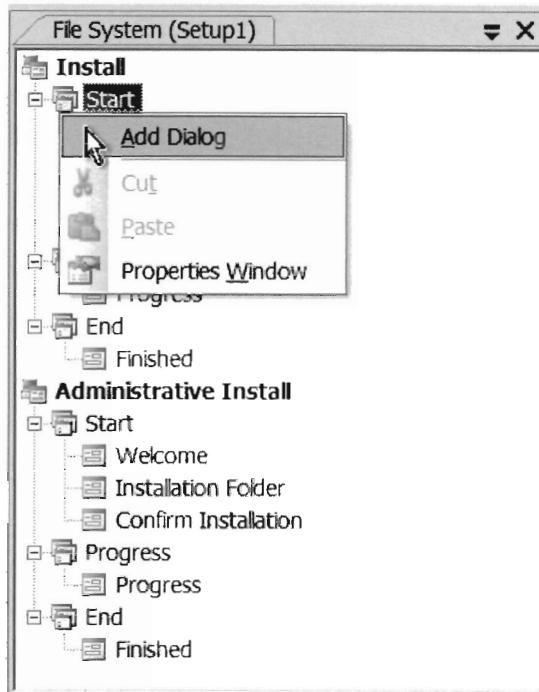
```
public override void Uninstall(System.Collections.IDictionary sa
vedState)
{
    base.Uninstall(savedState);
    MessageBox.Show("進行Uninstall程序");
}
```



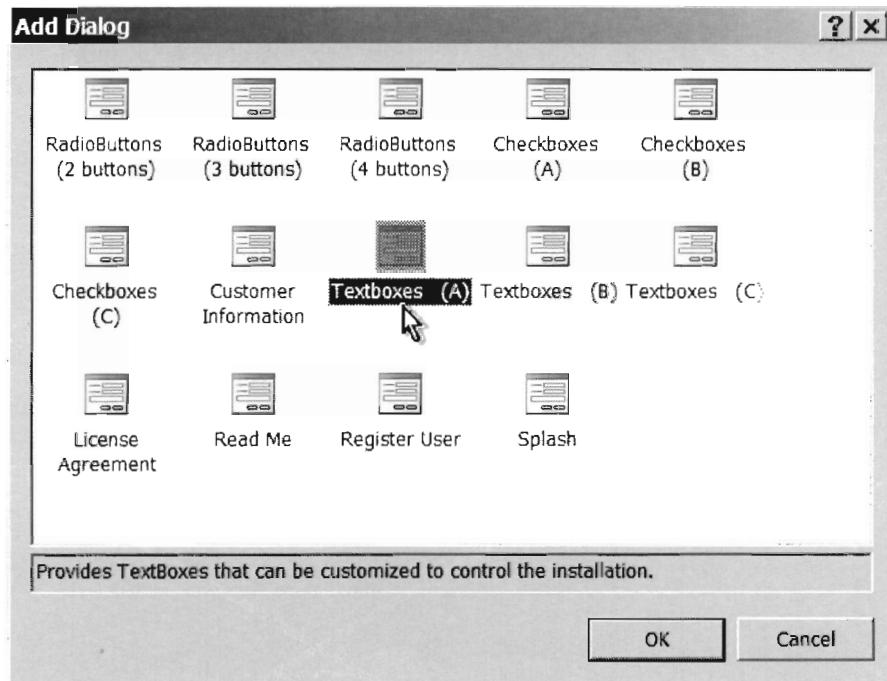
## Setup 專案

使用 Setup 專案進行專案的封裝：

1. 在選單選取「File」→「Add New Project」，在「Add New Project」視窗選取 Setup 專案。
2. 在「File System」視窗選取「Application Folder」並按滑鼠右鍵選取「Add」→「Project Output」，然後在「Add Project Output Group」視窗「Project」選取要輸出的專案及「Primary output」。
3. 從選單選取「View」→「Edit」→「User Interface」，展開「Install」選取「Start」按滑鼠右鍵點選「Add Dialog」。



4. 在「Add Dialog」視窗選取「Textboxes (A)」項目，按「OK」。



5. 回到「User Interface」視窗，設定 Textboxes (A)項目的屬性如下：

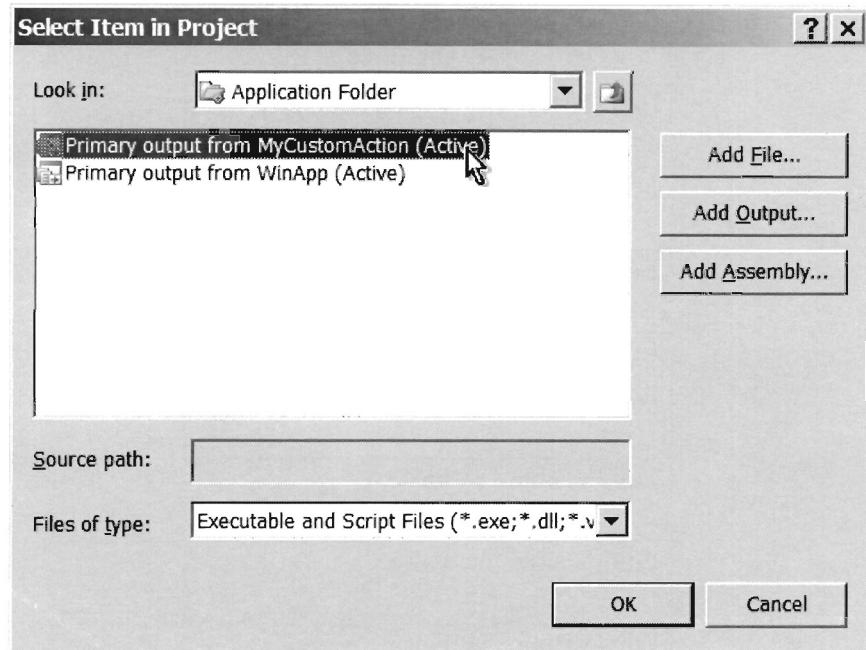
屬性	設定值	描述
----	-----	----

BannerText	Custom User Interface	自訂視窗的主標題(橫幅文字)。
BodyText	Type your name	自訂視窗的用途說明文字。
Edit1Label	Owner Name:	第一個文字方塊的標籤文字
Edit1Property	PRODUCTOWNER	用來這個文字方塊的屬性名稱。
Edit1Value		第一個文字方塊中的值。
Edit1Visible	True	第一個文字方塊是否顯示。
Edit2Visible	False	第二個文字方塊是否顯示。
Edit3Visible	False	第三個文字方塊是否顯示。
Edit4Visible	False	第四個文字方塊是否顯示。

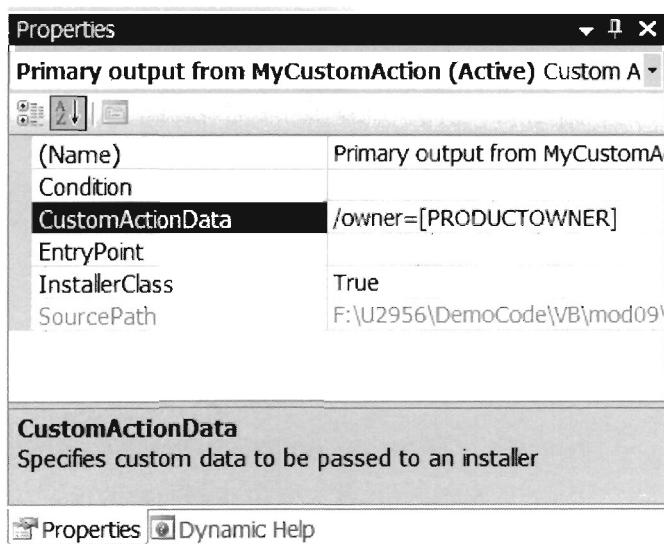
6. 從選單選取「View」→「Edit」→「Custom Actions」。
7. 在「Custom Actions」視窗選取「Install」節點按滑鼠右鍵選取「Add Custom Action...」。



8. 接著在「Select Item in Project」視窗找到放置 Installer 延伸類別所寫的專案位置並選取該項目，按下「OK」。



9. 選取加入的項目，然後設定屬性 CustomActionData 為「/owner=[PRODUCTOWNER]」，owner 會成為 Installer 延伸類別的 Context 物件的參數集合(Parameters)。



10. 完成設定之後便可進行建置專案與安裝專案了。

## 總結

了解如何使  
用組態檔設  
定.NET 執行  
的版本

了解如何使  
用DEVPATH  
找出共用組  
件

了解如何使  
用Configuration  
API 存取組態  
檔

了解如何自  
訂安裝程序

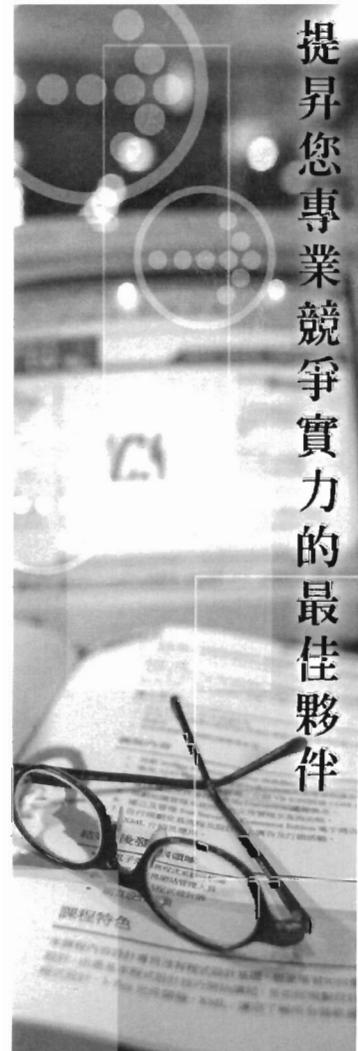
# 第八章：監控及偵錯

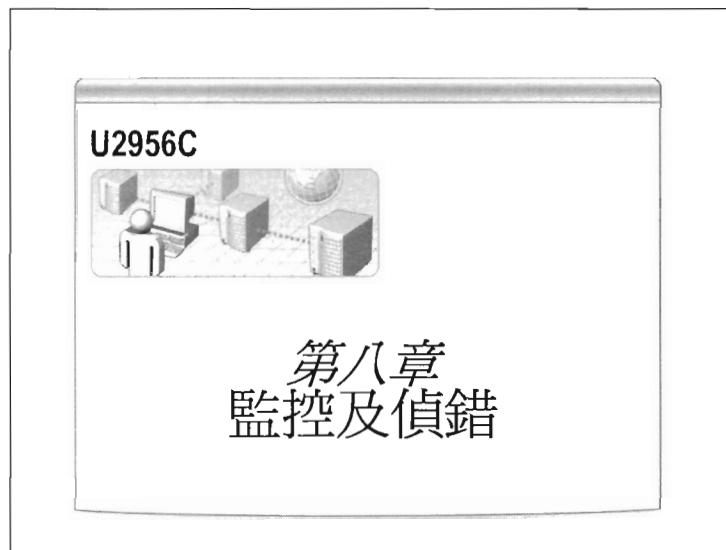
## 本章大綱

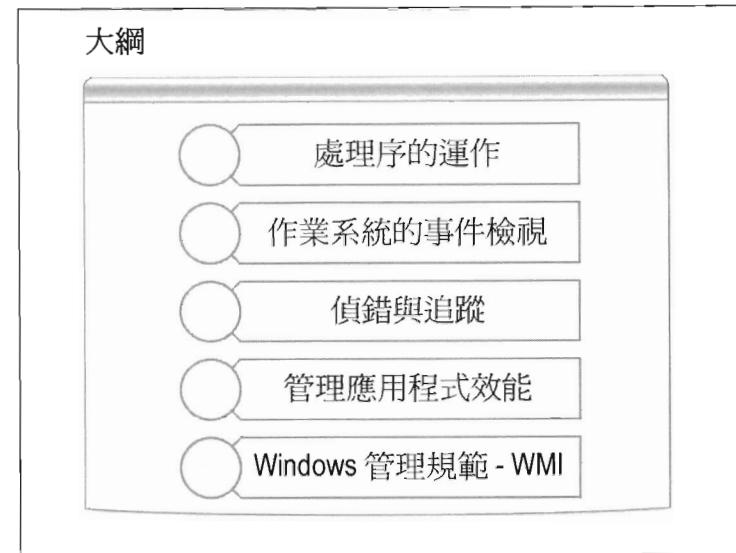
認識 Process 類別.....	5
取得 Process 的資訊.....	6
ProcessStartInfo 類別.....	8
作業系統的事件檢視.....	11
認識作業系統的事件記錄.....	12
建立事件記錄.....	15
寫入事件訊息到記錄檔.....	16
讀取事件訊息.....	18
刪除事件記錄檔.....	19
練習 8.1：列出事件記錄.....	20
認識偵錯與追蹤.....	24
練習 8.2：使用 Debug 類別偵錯.....	26
追蹤應用程式.....	32
管理應用程式效能.....	37
效能監視.....	38
建立自訂效能計數器.....	40
效能計數器.....	42
練習 8.3：建立自訂效能計數器.....	45
Windows 管理規範 - WMI.....	49
如何查詢目前暫停中的服務.....	52
如何監控新啓動的應用程式(同步).....	54
如何監控被暫停的服務(非同步).....	56
練習 8.4：監控並建立帳號(非同步).....	58

作者：

羅慧貞



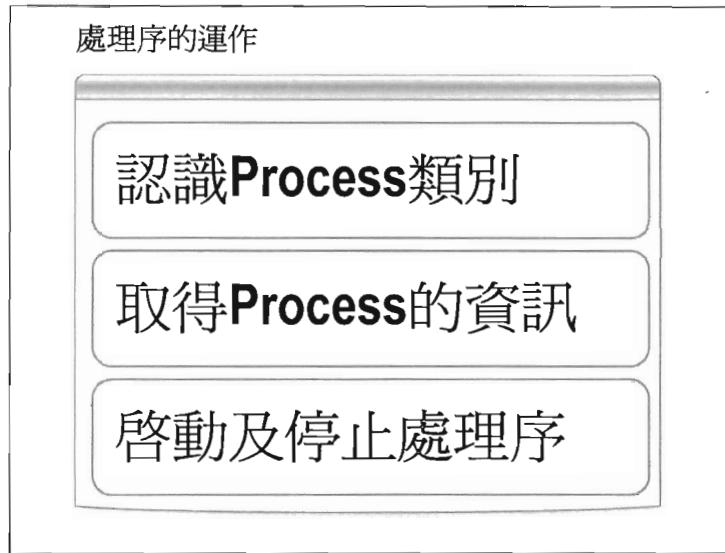




在這個章節中將介紹如何在程式中控制處理序的啓動、關閉及列表，如何在程式中將事件記錄在作業系統的事件檢視記錄檔，以及如何利用.NET 提供的 Debug 及 Trace 進行應用程式的除錯及追蹤功能。作業系統提供了一個效能檢視器的功能，它除了可以監控作業系統及周邊效能計數器的數據也容許你的應用程式自訂效能計數器。如何監控效能數據、如何自訂應用程式專屬的效能計數器，這章也將介紹之。

本章介紹以下主題：

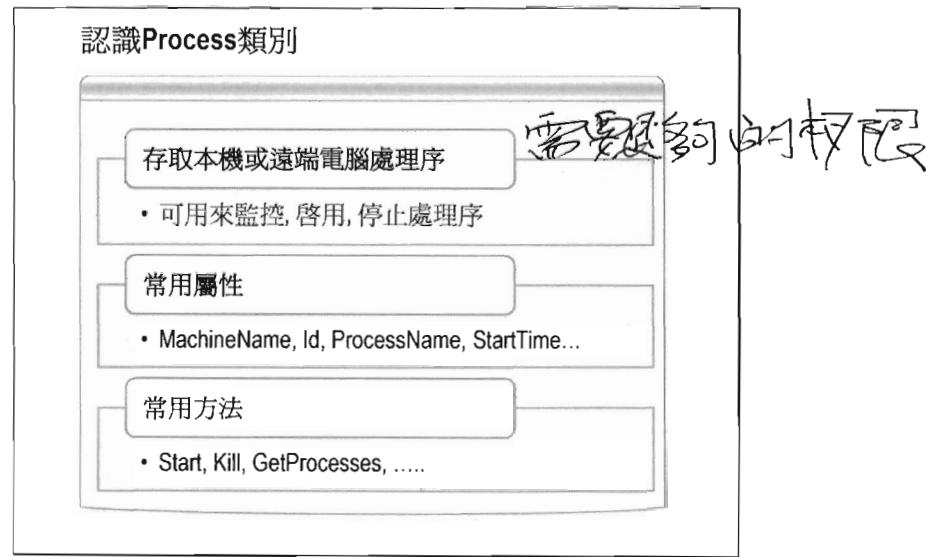
- 處理序的運作
- 作業系統的事件檢視
- 偵錯與追蹤
- 管理應用程式效能
- Windows 管理規範 - WMI



---

在這一節中將介紹 Process 類別的用途，內容包含：

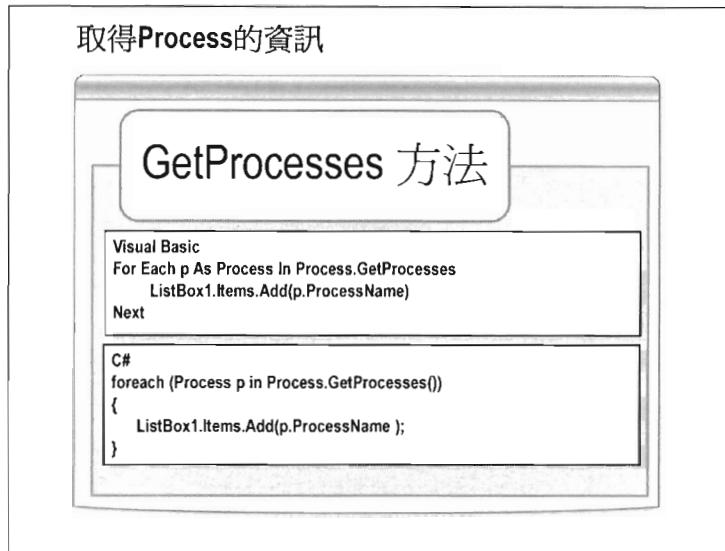
- 認識 Process 類別
- 取得 Process 的資訊
- 啓動及停止處理序



## 認識 Process 類別

Process 類別可以用來存取本機或是遠端電腦的處理序。這個類別可以用來啓動、停止、控制和監視應用程式。這個類別提供以下常用的成員：

- GetProcesses()，取得指定電腦執行中的處理程序。
- GetProcessesByName()，取得指定名稱的處理程序。
- Start()，啓動處理程序。
- Kill()，立即停止處理程序。
- CloseMainWindow，關閉有視窗介面的 Process。
- ID 屬性，處理程序的識別碼。
- ProcessName 屬性，處理程序的名稱。
- Responding 屬性，檢查處理程序的使用者介面是否有回應。
- Modules 屬性，取得 Process 物件所載入的模組。



## 取得 Process 的資訊

使用 GetProcesses 方法可以取得電腦執行中的處理序清單，它會回傳 Process 陣列，可以使用 For Each 指令(Visual Basic，C#是 foreach)列舉其內容。

以下範例是使用 Process 類別的 GetProcesses 靜態方法 (於 Visual Basic 是共享方法) 取得目前電腦中執行中的處理程序，並將處理程序的名稱列印在 ListBox 控制項中。

```

Visual Basic
ListBox1.Items.Clear()
For Each p As Process In Process.GetProcesses
    ListBox1.Items.Add(p.ProcessName)
Next

```

```

C#
ListBox1.Items.Clear();
foreach (Process p in Process.GetProcesses())
{
    ListBox1.Items.Add(p.ProcessName );
}

```

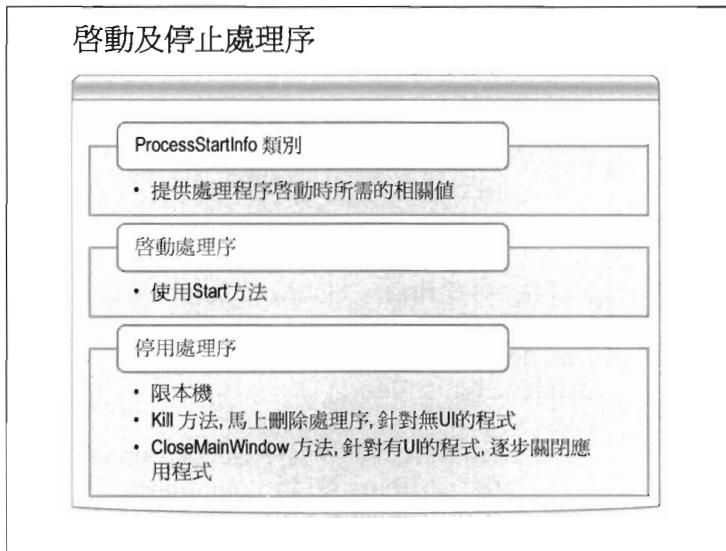
以下範例是使用者在第一個 ListBox 選取處理程序的名稱之後，接著在第二個 ListBox 控制項顯示被選取處理程序的已載入模組名稱。使用 Process 類別的 GetProcessesByName 靜態方法取得指定的處理程序物件，然後再從處理程序物件的 Modules 屬性取得該處理程序已載入模組名稱。

Visual Basic

```
Dim pName As String = ListBox1.SelectedItem.ToString()
Dim procs As Process() = Process.GetProcessesByName(pName)
If procs.Length > 0 Then
    ListBox2.Items.Clear()
    Try
        For Each m As ProcessModule In procs(0).Modules
            ListBox2.Items.Add(m.ModuleName)
        Next
    Catch ex As Exception
    End Try
End If
```

C#

```
String pName = ListBox1.SelectedItem.ToString();
Process[] p = Process.GetProcessesByName(pName);
if (p.Length > 0)
{
    ListBox1.Items.Clear();
    try
    {
        foreach (ProcessModule m in p[0].Modules)
        {
            ListBox2.Items.Add(m.ModuleName);
        }
    }
    catch
    { }
}
```



## ProcessStartInfo 類別

提供處理程序啟動時所需的相關值，像是應用程式的檔名、命令列引數、啟動應用程式的執行帳號與密碼...等。完成處理程序啟動時所需的相關屬性值定義後，可以使用 Process 類別的 Start 靜態方法 (Visual Basic 為共享方法) 傳 ProcessStartInfo 物件來啟動這個應用程式。它的常用成員如下：

- FileName，要啟動應用程式的路徑與檔名。
- Arguments，啟動應用程式所需的命令列引數。
- UseShellExecute，是否使用作業系統的 Shell 啓動處理程序。預設為 True。如果有指定 UserName 與 Password 時，必須設定為 False。
- UserName，啟動應用程式時使用的帳號。
- Password，啟動應用程式時使用的使用者密碼。型別是 System.Security.SecureString，SecureString 物件類似 String 物件，不過是經過加密的字串。

下面這個範例是使用 ProcessStartInfo 建立物件，並指定要啟動的應用程式名稱為 MyNotepad.exe，這是一個類似記事本的文字編輯器，可以命令列引數指定要開啟的檔案。使用 Arguments 屬性指定要開啟的檔案名稱，指定程式執行的帳號與密碼，另外，若有指定程式執行的

帳號與密碼 UseShellExecute 屬性就必須設為 False。接著使用 Process 的 Start 方法啟動 ProcessStartInfo 物件。

#### Visual Basic

```
Dim info As New ProcessStartInfo("MyNotepad.exe")
info.Arguments = "c:\2956.txt"
If CheckBox1.Checked Then
    '請建立帳號User1與密碼1
    info.UseShellExecute = False
    info.UserName = "User1"
    info.Password = New System.Security.SecureString()
    info.Password.AppendChar(Convert.ToChar("1"))
End If
Process.Start(info)
```

#### C#

```
ProcessStartInfo info = new ProcessStartInfo("MyNotepad.exe");
info.Arguments = @"c:\2956.txt";
if (CheckBox1.Checked )
{//請建立帳號User1與密碼1
    info.UseShellExecute = false;
    info.UserName = "User1";
    info.Password = new System.Security.SecureString();
    info.Password.AppendChar(Convert.ToChar("1"));
}
Process.Start(info);
```

## 停用處理序

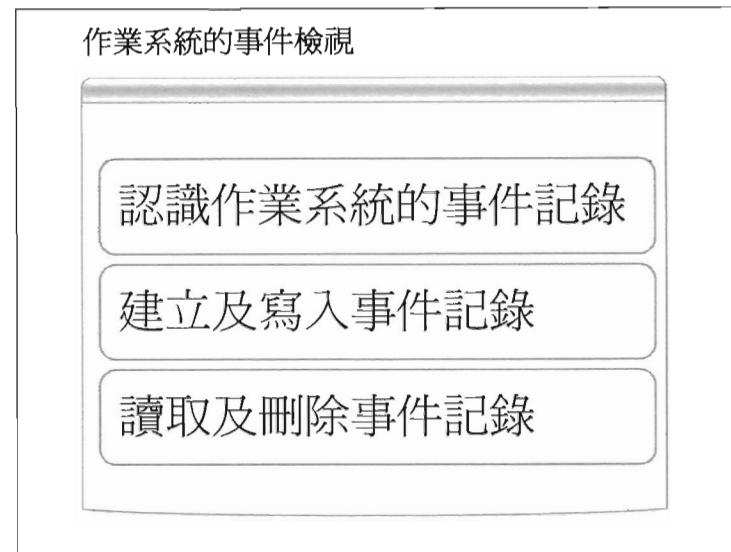
可以使用 Kill 或是 CloseMainWindow 方法，CloseMainWindow 方法會逐步將視窗關閉之後結束應用程式，除非必要避免使用 Kill 方法。以下範例是將使用者所選取的 Process 項目刪除。

#### Visual Basic

```
If ListBox1.SelectedIndex < 0 Then
    Return
End If
Dim proc As Process
proc = CType(ListBox1.SelectedItem, Process)
Try
    If Not proc.HasExited Then
        proc.CloseMainWindow()
        proc.WaitForExit()
        proc.Close()
    Else
        MessageBox.Show("已結束")
    End If
```

```
Catch  
    MessageBox.Show("無法結束")  
End Try  
Button1.PerformClick()
```

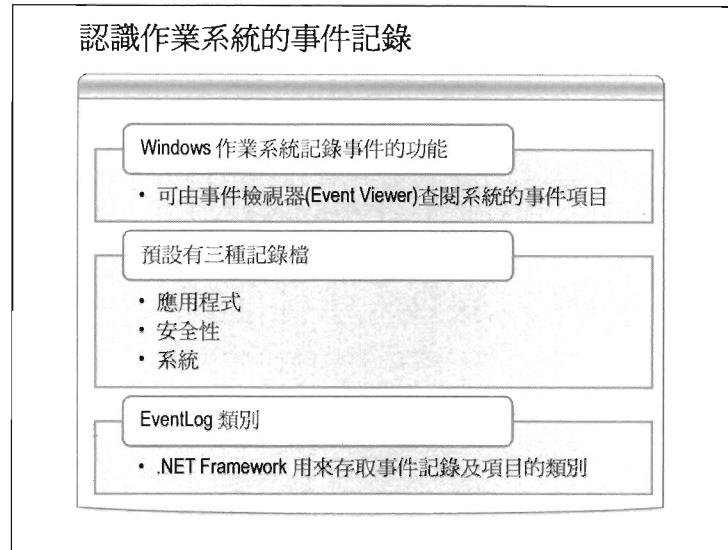
```
C#  
if (ListBox1.SelectedIndex < 0)  
    return;  
  
Process proc = (Process)ListBox1.SelectedItem;  
  
try  
{  
    if (!proc.HasExited)  
    {  
        proc.CloseMainWindow();  
        proc.WaitForExit();  
        proc.Close();  
    }  
    else  
        MessageBox.Show("已結束");  
}  
catch  
{  
    MessageBox.Show("無法結束!!");  
}  
Button1.PerformClick();
```



## 作業系統的事件檢視

在這一節中將介紹如何在.NET 應用程式中寫入記錄到作業系統的事件記錄，並且了解 EventLog 類別的用法，包含以下項目：

- 認識作業系統的事件記錄
- 建立及寫入事件記錄
- 讀取及刪除事件記錄



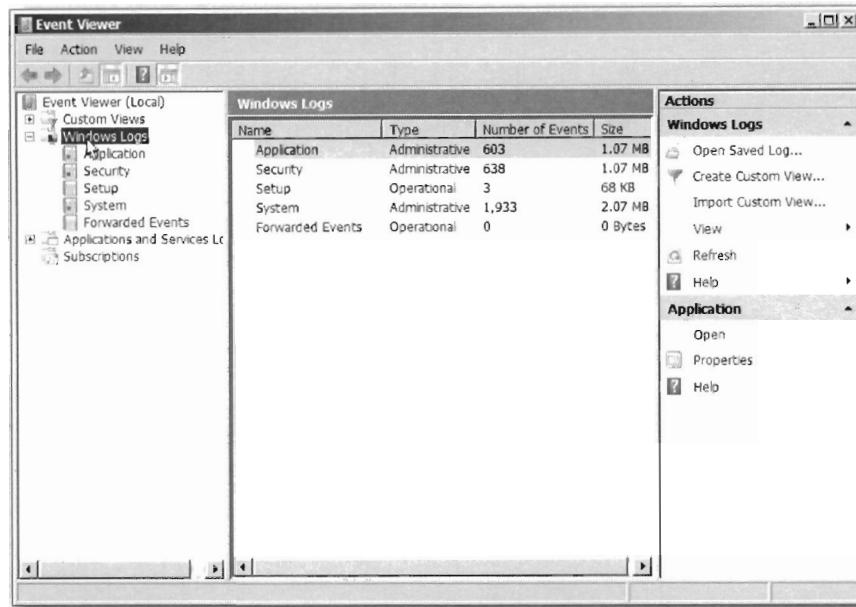
## 認識作業系統的事件記錄

Windows 作業系統在運作時會將作業系統、服務、硬體...等所發生的錯誤訊息、警告訊息、一般訊息...等資訊收集在事件記錄檔中，讓管理人員可以利用事件檢視器去查看或追蹤系統的狀態。

### 事件記錄檔(Log)

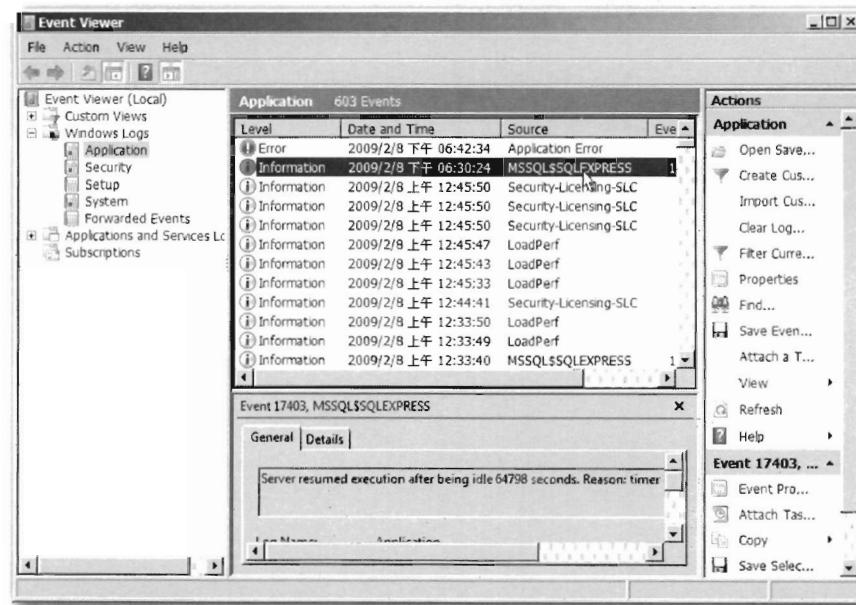
預設作業系統會有三個事件記錄檔，分別是應用程式 (Application)、安全性 (Security)、系統 (System)。

- 應用程式 (Application)，用來記錄各種應用程式的事件，像是 SQL Server 如果服務啟動失敗，會將啟動失敗的原因或訊息寫在應用程式日誌檔。
- 安全性 (Security)，用來記錄系統的安全稽核事項，例如，啓用使用者登入成功或失敗的稽核功能，當使用者嘗試登入成功或失敗時，會將原因或訊息寫在安全性日誌檔。
- 系統 (System)，用來記錄系統元件的事件。例如，啟動時某個驅動程式或系統元件無法載入時，會將原因或訊息寫在安全性日誌檔



## 事件來源(Source)

事件訊息的來源，通常是指應用程式的名稱。來源一定屬於一個記錄檔，例如 MSSQLSERVER 來源屬於應用程式記錄檔。



## 訊息類型(EventLogEntryType)

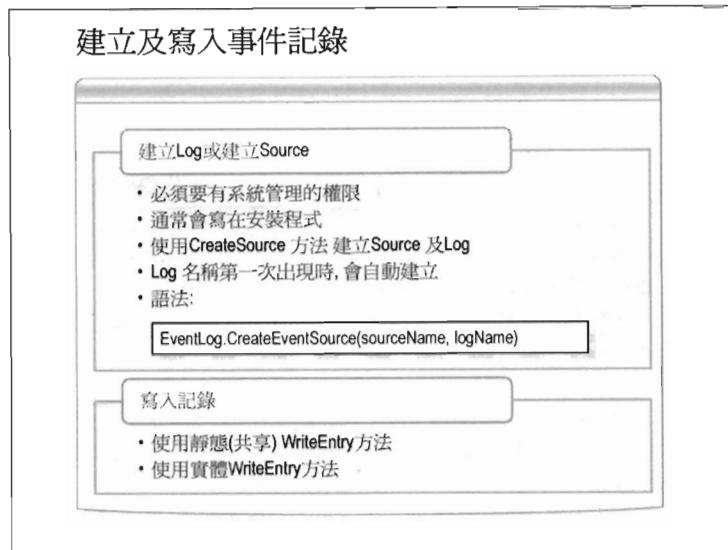
指的是訊息的種類，包含一般訊息、警告訊息...等，在.NET 的程式中有個 EventLogEntryType 的列舉常數，包含了事件記錄檔的訊息類型列表：

- Error，錯誤的事件訊息。
- FailureAudit，稽核的對象嘗試發生失敗時的事件訊息。
- Information，一般的事件訊息。
- SuccessAudit，稽核的對象嘗試成功時的事件訊息。
- Warning，警告的事件訊息。

## EventLog 類別

EventLog 類別是.NET Framework 提供來建立、刪除存取事件記錄檔的一組程式。它屬於 System.Diagnostics 命名空間。它提供的常用方法如下：

- Exists，用來判斷某個指定的記錄檔(Log)是否存在。
- SourceExists，用來判斷某個指定的事件來源是否存在。
- CreateEventSource，用來建立事件的來源(Source)，來源在建立時，必須指定所屬的記錄檔，同時在建立時會進行寫註冊機碼記錄來源所屬性的記錄檔。必須擁有系統管理權限的使用者才能執行這個方法，通常會寫在安裝程式中。
- Delete，用來刪除指定的記錄檔(Log)。
- WriteEntry，用來將訊息寫入記錄檔中，如果指定的事件來源是未曾建立，會自動建立及註冊事件來源。



## 建立事件記錄

EventLog 類別並沒有提供建立記錄檔(log)的方法，不過記錄檔在這台電腦上第一次出現時會自動建立。那麼什麼方式可以建立記錄檔呢？可以透過 `CreateEventSource` 或 `WriteEntry` 方法指定來源(Source)或訊息所屬的記錄檔，當記錄檔不存在於系統時，系統會自動建立一個新的記錄檔。

不管是建立記錄檔或來源名稱，都會對系統進行註冊的動作，這個行為必須要有「系統管理」的權限才可以進行。因此建議建立記錄檔或來源名稱的程式寫在安裝程式中(因為安裝程式必須是由系統管理者進行)，以確保執行時有權限可以成功建立記錄檔或來源名稱。

以下程式碼是使用 `SourceExists` 方法判斷來源名稱是否存在，如果不存在使用 `CreateEventSource` 方法建立來源名稱。

```
Visual Basic
Dim logName As String = "MyLog"
Dim sourceName As String = Application.ProductName
Dim message As String
If EventLog.SourceExists(sourceName) Then
    message = String.Format("日誌名稱:{0}, 來源名稱:{1}已存在",
                           logName, sourceName)
Else
    EventLog.CreateEventSource(sourceName, logName)
```

```

    message = String.Format("日誌名稱:{0}, 來源名稱:{1}, 建立日期:{2}",
                           logName, sourceName, Date.Now)
End If
EventLog.WriteEntry(sourceName, message, EventLogEntryType.Information)

```

```

C#
string logName = "MyLog";
string sourceName = Application.ProductName;
String message ;
if (EventLog.SourceExists(sourceName))
    message = String.Format("日誌名稱:{0}, 來源名稱:{1}已存在",
                           logName, sourceName);
else {
    EventLog.CreateEventSource(sourceName, logName);
    message = String.Format("日誌名稱:{0}, 來源名稱:{1}, 建立日期:{2}",
                           logName, sourceName, DateTime.Now);
}
EventLog.WriteEntry(sourceName, message,
                    EventLogEntryType.Information);
MessageBox.Show(message);

```

## 寫入事件訊息到記錄檔

寫入事件訊息到記錄檔的方法有二種：

- 使用 EventLog 的共享方法 WriteEntry。這是用在訊息只需寫入一次時，必須在寫 WriteEntry 時指定訊息所屬的記錄檔 (Log)、來源(Source)。

```

Visual Basic
EventLog.WriteEntry(sourceName, message, _
EventLogEntryType.Information)

```

```

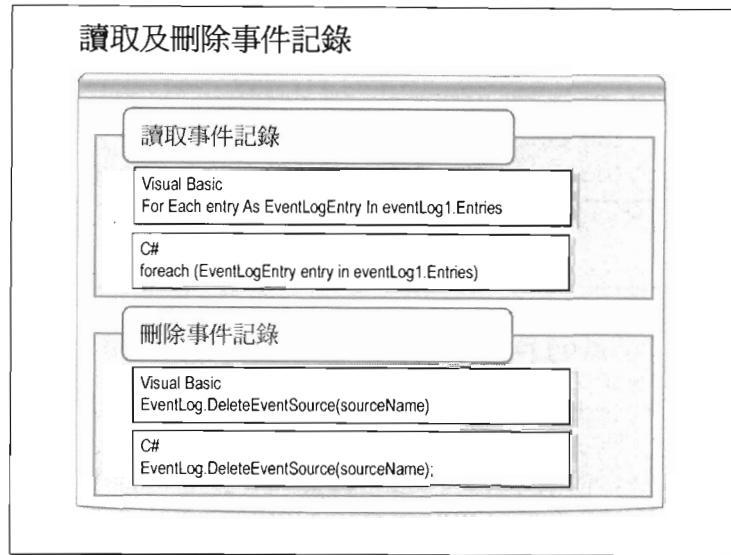
C#
EventLog.WriteEntry(sourceName, message,
                    EventLogEntryType.Information);

```

- 建立 EventLog 物件，使用物件的 WriteEntry 方法。這是用在訊息要寫入多次時，EventLog 物件指定 Log 與 Source 屬性，之後使用 WriteEntry 方法，只需傳遞寫入訊息與訊息類型。

```
Visual Basic
myLog As New EventLog("MyLog")
myLog.Source = Application.ProductName
myLog.WriteEntry("這是一般訊息", _
    EventLogEntryType.Information)
myLog.WriteEntry("這是警告訴息", EventLogEntryType.Warning)
myLog.WriteEntry("這是錯誤訊息", EventLogEntryType.Error)
```

```
C#
EventLog myLog = new EventLog("MyLog");
myLog.Source = Application.ProductName;
myLog.WriteEntry("這是一般訊息", _
    EventLogEntryType.Information);
myLog.WriteEntry("這是警告訴息", EventLogEntryType.Warning);
myLog.WriteEntry("這是錯誤訊息", EventLogEntryType.Error);
```



## 讀取事件訊息

EventLog 物件提供 Entries 屬性集合，可以使用 For 迴圈讀取事件訊息、事件類型，與事件寫入日期。每個事件訊息的類別是 EventLogEntry ~

EventLogEntry 類別屬性說明如下：

- Message，取得事件訊息。
- EntryType，取得事件的類型。
- TimeWritten，取得事件的寫入時間。

```

Visual Basic
Dim myLog As New EventLog(logName)
myLog.Source = sourceName
For Each entry As EventLogEntry In myLog.Entries
    ListBox1.Items.Add( _
        String.Format("訊息:{0}, 類型:{1}, 日期:{2}", _
            entry.Message, _
            entry.EntryType.ToString(), _
            entry.TimeWritten))

```

Next

```
C#
EventLog myLog=new EventLog(logName);
```

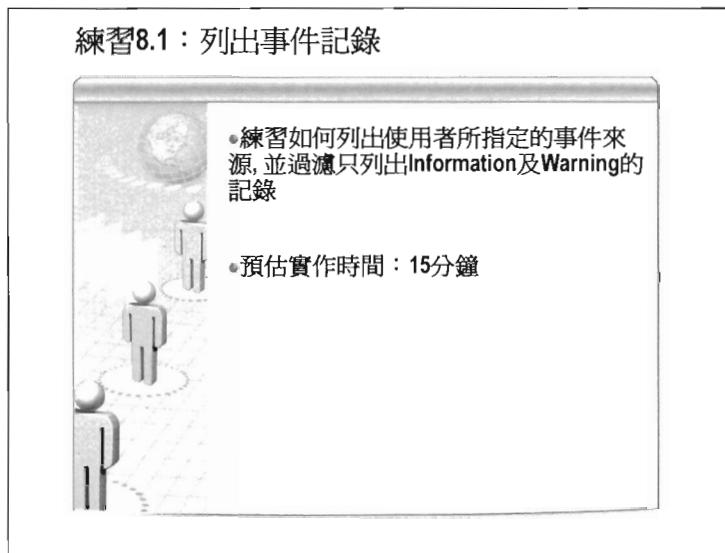
```
myLog.Source = sourceName;
foreach (EventLogEntry entry in myLog.Entries)
{
    ListBox1.Items.Add(
        String.Format("訊息:{0}, 類型:{1}, 日期:{2}",
        entry.Message, entry.EntryType.ToString(),
        entry.TimeWritten));
}
```

## 刪除事件記錄檔

如果要將記錄檔(log)從電腦移除，可以使用 Exists 方法判斷記錄是否存在，如果存在使用 Delete 方法刪除記錄檔。

```
Visual Basic
If EventLog.Exists(logName) Then
    EventLog.Delete(logName)
    MessageBox.Show("刪除日誌")
Else
    MessageBox.Show("日誌不存在")
End If
```

```
C#
if (EventLog.Exists(logName))
{
    EventLog.Delete(logName);
    MessageBox.Show("刪除日誌");
}
else
    MessageBox.Show("日誌不存在");
```



## 練習 8.1：列出事件記錄

目的：

這個練習如何列出使用者所指定的事件來源，並過濾只列出 Information 及 Warning 的記錄。

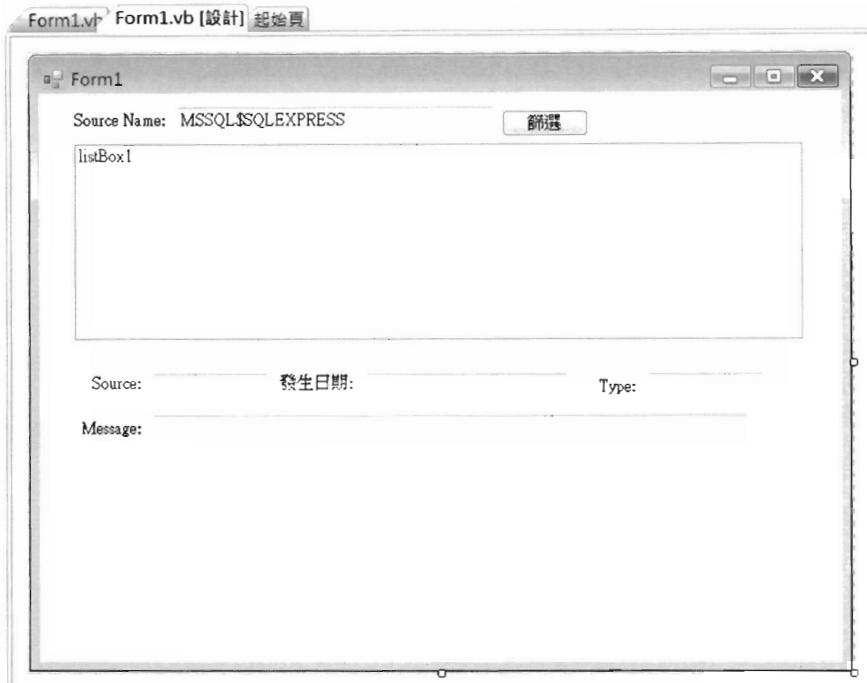
功能描述：

這個練習會在畫面上讓使用者輸入事件來源名稱，然後程式會篩選出 Information 及 Warning 的事件記錄列於清單之中。

預估實作時間：15 分鐘

實作步驟：

1. 到「\Practices\VB 或 CS\Mod08\_1\Starter」目錄，開啟 Mod08\_1.sln 方案，表單畫面如下：



2. 若開啟的是 Visual Basic 專案請跳過此步驟。切到程式碼視窗，C# 程式必須先匯入命名空間：

```
C#
using System.Diagnostics;
```

3. 宣告一個 List 泛型集合變數，做為儲存篩選記錄的容器：

```
Visual Basic
Dim entryList As New List(Of EventLogEntry)
```

```
C#
List<EventLogEntry> entryList = new List<EventLogEntry>();
```

4. 在 Button1 按鈕的 Click 事件，要將事件來源名稱的事件記錄列表於 listBox1 清單中，程式碼如下：

```
Visual Basic
Dim myLog As New EventLog("Application")
myLog.Source = textBox1.Text
For Each entry As EventLogEntry In myLog.Entries
    If entry.Source = textBox1.Text Then
        If entry.EntryType = EventLogEntryType.Information Or _
            entry.EntryType = EventLogEntryType.Warning Then
            entryList.Add(entry)
        End If
    End If
Next
```

```
listBox1.DataSource = entryList
listBox1.DisplayMember = "Message"
```

```
C#
EventLog myLog = new EventLog("Application");
myLog.Source = textBox1.Text ;
foreach (EventLogEntry entry in myLog.Entries)
{
    if (entry.Source == textBox1.Text)
    {
        if (entry.EntryType == EventLogEntryType.Information || 
            entry.EntryType == EventLogEntryType.Warning)
        {
            entryList.Add(entry);
        }
    }
}

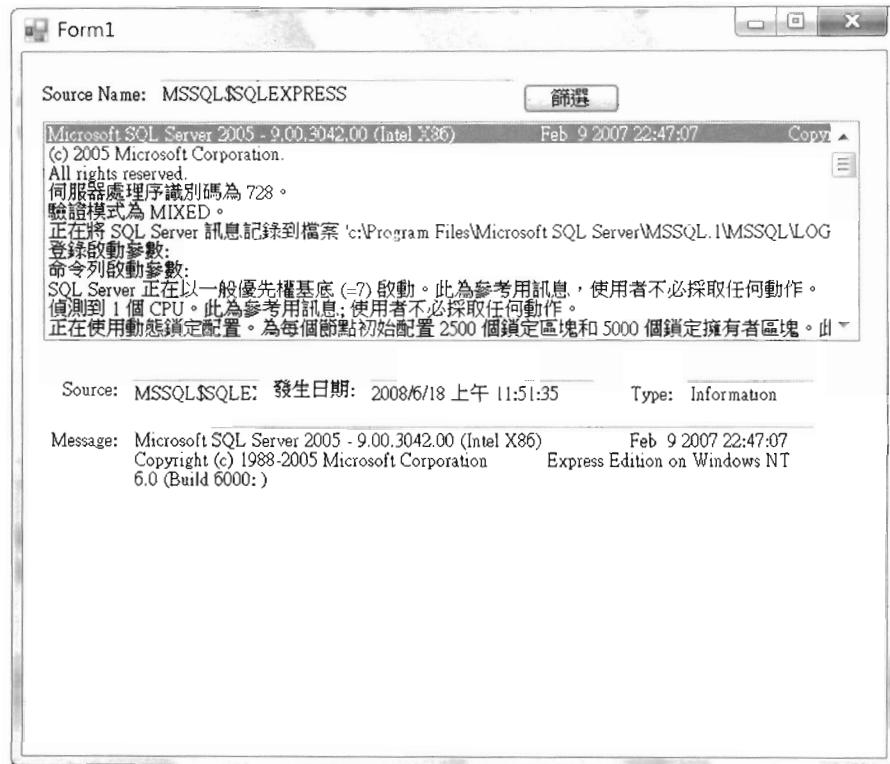
listBox1.DataSource = entryList;
listBox1.DisplayMember = "Message";
```

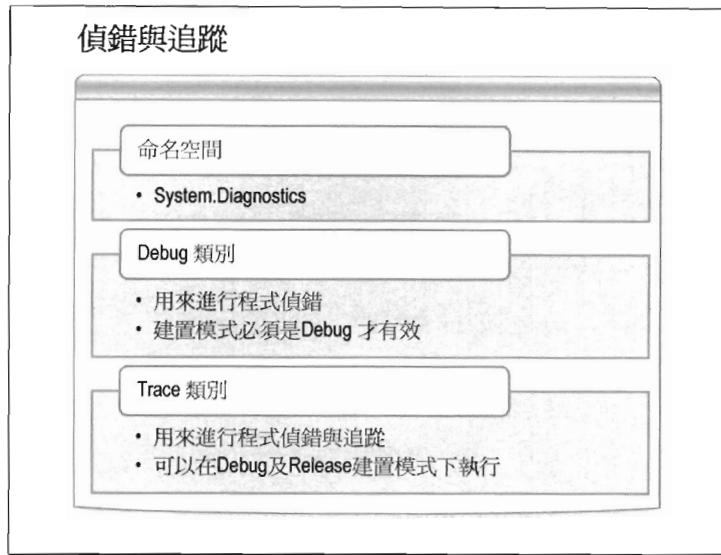
5. 在 listBox1 選取的 SelectedIndexChanged 事件中，將事件記錄的細節顯示於視窗的下方，程式碼如下：

```
Visual Basic
Dim entry As EventLogEntry
entry = CType(listBox1.SelectedItem, EventLogEntry)
textBox2.Text = entry.Source
textBox3.Text = entry.Message
textBox4.Text = entry.TimeWritten.ToString()
textBox5.Text = entry.EntryType.ToString()
```

```
C#
EventLogEntry entry =(EventLogEntry) listBox1.SelectedItem ;
textBox2.Text = entry.Source;
textBox3.Text = entry.Message;
textBox4.Text = entry.TimeWritten.ToString();
textBox5.Text = entry.EntryType.ToString();
```

6. 執行程式，結果如下：





## 認識偵錯與追蹤

.NET Framework 提供用來偵測、配合開發工具的偵錯、系統上線後的追蹤功能...等功能類別都歸類在 `System.Diagnostics` 命名空間之下。其中用來偵錯與追蹤的類別分別是 `Debug` 類別與 `Trace` 類別。

- `Debug` 類別，在程式開發與測試階段的使用，多數功能與 Visual Studio 開發工具綁在一起，用來輸出程式中的變數與執行變化，做為程式偵錯之用。例如，`Write`、`WriteLine`...等方法預設是將結果輸出「Output Window」。`Debug` 類別也只有在建置模式為 `Debug` 模式時有效。
- `Trace` 類別，可以用在程式開發、測試與上線等階段使用，功能與 `Debug` 類似，可以用來輸出程式中的變數或執行變化，做為日後的追蹤之用。建置模式 `Debug` 與 `Release` 模式都有效。

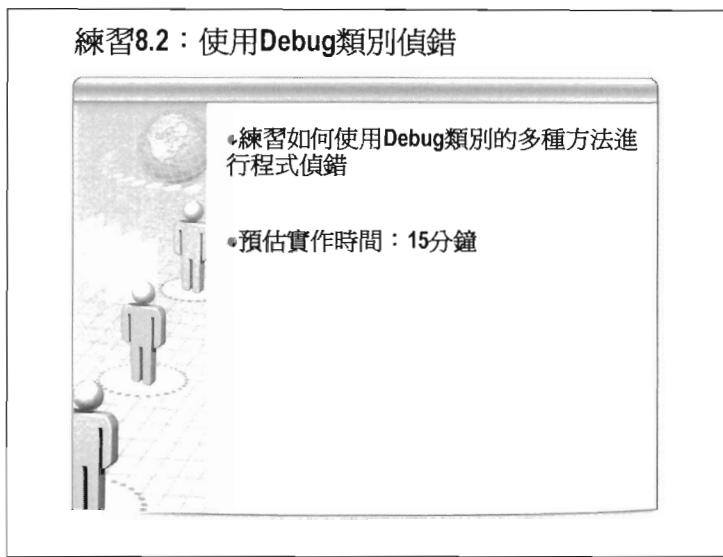
## Debug 類別

`Debug` 類別可以用在應用程式偵錯階段(建置模式為 `Debug` 模式)，它提供了一些方法可以輸出偵錯的訊息、判斷條件並中止程式顯示訊息，常見方法如下：

- Assert，檢查條件是否為 False，如果為 False，顯示訊息。
- Fail，在程式判斷到有錯誤時，可以呼叫這個方法，此方法會顯示錯誤訊息。
- WriteLine，在 Output 視窗輸出一行錯誤訊息。
- WriteLineIf，檢查條件是否成立(為 True)，如果成立則顯示一行錯誤訊息到 Output 視窗。

## Trace 類別

Trace 類別主要是用來在上線之後進行應用程式的追蹤，並且可以在 Debug 及 Release 的建置模式下執行。它的用法與 Debug 幾乎一樣，差別是當它執行在 Release 模式下並沒有 Output 視窗可供輸出資訊，必須使用 Listeners 屬性指定輸出單元，通常還要搭配 TraceSwitch 類別做為追蹤資訊是否輸出的切換。.NET Framework 2.0 之後的版本新增 TraceSource 類別用來取代.NET 1.0 時的 Trace 類別。



## 練習 8.2：使用 Debug 類別偵錯

目的：

練習如何使用 Debug 類別的多種方法進行程式偵錯。

功能描述：

在這個練習中將練習在程式碼中使用 Debug 類別的多個方法，你將會了解這些方法的使用時機。

預估實作時間：15 分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod08\_2。
3. 設計 Form1 畫面如下：



4. 在「查詢員工電話」雙擊兩下進入 Click 事件。

5. C# 必須先匯入命名空間：

```
C#
using System.Diagnostics;
```

6. 在 Button1\_Click 事件，撰寫以下程式：

- 使用 Assert 檢查使用者如果沒有輸入要查詢的姓名，就顯示訊息提示使用者必須輸入姓名。

```
Visual Basic
Dim cnString As String = ""
Debug.Assert(Not String.IsNullOrEmpty(TextBox1.Text), _
"必須輸入姓名!!")
```

```
C#
String cnString = "";
Debug.Assert(! String.IsNullOrEmpty(TextBox1.Text),
"必須輸入姓名!!");
```

- 然後程式會檢查設定檔是否有設定連線字串，如果沒有設定連線字串，使用 Debug.Fail 方法顯示錯誤訊息。

```
Visual Basic
Dim cnSettings As ConnectionStringSettingsCollection
cnSettings = ConfigurationManager.ConnectionStrings
If cnSettings Is Nothing OrElse _
cnSettings("cnNorthwind") Is Nothing Then
```

```

    Debug.Fail("必須設定連線字串")
Else
    cnString = cnSettings("cnNorthwind").ConnectionString
End If

```

```

C#
ConnectionStringSettingsCollection cnSettings ;
cnSettings = ConfigurationManager.ConnectionStrings;
if (cnSettings == null || cnSettings["cnNorthwind"]==null )
{
    Debug.Fail("必須設定連線字串");
}
else
    cnString = cnSettings["cnNorthwind"].ConnectionString ;

```

- 接著進行連線及查詢作業，並使用 `Debug.WriteLine` 顯示作業的情況。以及使用 `Debug.WriteLineIf` 檢查是否取得查詢結果，如果沒有，就顯示查詢不到該員工。

```

Visual Basic
Dim cn As New SqlConnection(cnString)
Dim com As New SqlCommand("Select HomePhone From Employees Where FirstName=@fname", cn)
com.Parameters.AddWithValue("@fName", TextBox1.Text)
Try
    cn.Open()
    Debug.WriteLine("連線完成")
    Dim result As String
    result = com.ExecuteScalar()
    Debug.WriteLine("查詢完成")
    Debug.WriteLineIf(String.IsNullOrEmpty(result), _
        TextBox1.Text & "不是員工.")
    If String.IsNullOrEmpty(result) Then
        Label2.Text = "查無此人."
    Else
        Label2.Text = result
    End If
Catch ex As Exception
    Debug.WriteLine("Error!" & ex.Message)
End Try

```

```

C#
SqlConnection cn =new SqlConnection(cnString);
SqlCommand com = new SqlCommand("Select HomePhone From Employees Where FirstName=@fname", cn);
com.Parameters.AddWithValue("@fName", TextBox1.Text);
try
{

```

```

cn.Open();
Debug.WriteLine("連線完成");

String result = (string)com.ExecuteScalar();
Debug.WriteLine("查詢完成");
Debug.WriteLineIf(String.IsNullOrEmpty(result),
    TextBox1.Text + "不是員工.");
if (String.IsNullOrEmpty(result))
    Label2.Text = "查無此人。";
else
    Label2.Text = result;
}
catch (Exception ex)
{
    Debug.WriteLine("Error!" + ex.Message);
}

```

7. 執行應用程式，不要輸入 TextBox1，會出現：(按中止)



8. 執行應用程式，任意輸入 TextBox1，會出現：(按中止)



9. 回到 Visual Studio，加入 app.config，設定連線字串如下：

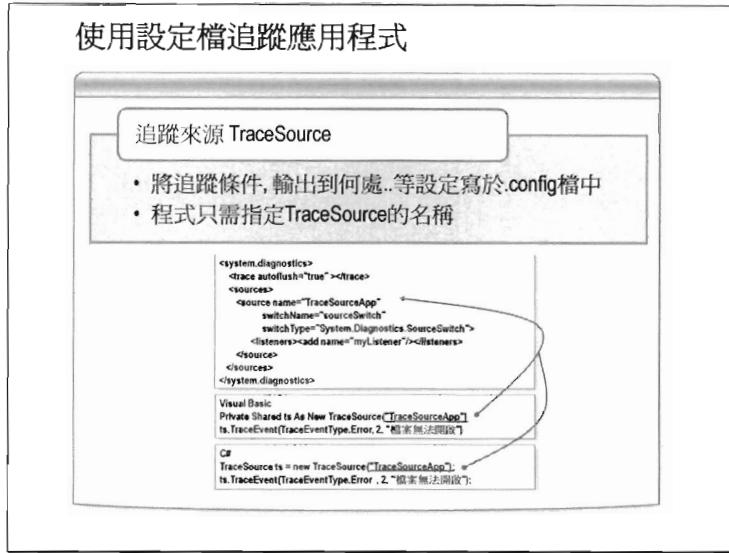
```
<connectionStrings>
    <add name="cnNorthwind"
        connectionString="Data Source=.;Initial Catalog=Northwind;Integrated Security=True"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

10. 執行應用程式，任意輸入 TextBox1，會出現：(請檢查  
「Output」視窗)



11. 輸入「Anne」執行結果如下：



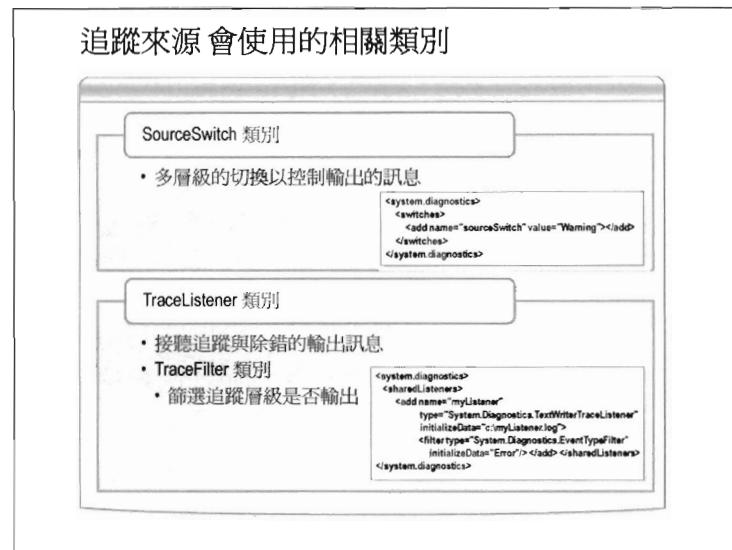


## 追蹤應用程式

TraceSource 類別是用來以應用程式為單位，將追蹤訊息所需的組態寫在設定檔，在建立 TraceSource 時指定設定檔中的 Source 名稱，藉以指定追蹤要用的 TraceSwitch 及 Listener 集合的組態。

以下是 TraceSource 常用的方法：

- **TraceEvent**，用來進行追蹤事件與資訊，並將訊息寫到 Listener 集合中指定輸出單元。此方法將來可以與自動化工具搭配做到通知的功能。
- **TraceInformation**，用來進行追蹤資訊的輸出，並將訊息寫到 Listener 集合中指定輸出單元。主要是提供給使用者閱讀的訊息，並不能與自動化工具搭配使用。
- **Flush**，將要輸出的資訊從暫存區清除並寫入 Listener 集合指中指定輸出單元。



## SourceSwitch 類別

提供多層級的切換以控制輸出的訊息。這是 TraceSource 類別用來指定追蹤訊息輸出的層級(TraceSource 物件的 Switch 屬性)。

SourceSwitch 物件的 Level 屬性是 TraceLevel 列舉型別，包含以下型別：

- Off，不輸出追蹤和偵錯的資訊。
- Error，只輸出錯誤訊息。
- Warning，輸出警告與錯誤訊息。
- Information，輸出一般資訊、警告與錯誤訊息。
- Verbose，輸出所有追蹤和偵錯的資訊。

爲了避免在切換追蹤層級及要修改程式及重新編譯程式，多將切換層級寫在組態檔中，使用<switches>區段定義切換層級的值。

```
<system.diagnostics>
<switches>
<add name="sourceSwitch" value="Warning"></add>
</switches>
</system.diagnostics>
```

## TraceListener 類別

用來接聽追蹤與除錯的輸出訊息，抽象類別。使用 TraceSource 物件的 Listeners 屬性可以取得接聽追蹤訊息的 Listener 集合。從 TraceListener 延伸的類別包含：

- DefaultTraceListener，追蹤與除錯預設的輸出方式。是指 Visual Studio 開發工具的輸出視窗。
- EventLogTraceListener，追蹤與除錯輸出到事件日誌檔。
- TextWriterTraceListener，追蹤與除錯輸出到 TextWriter 或 Stream 類別所指定的來源。
- DelimitedListTraceListener，是從 TextWriterTraceListener 延伸的類別，追蹤與除錯輸出到文字檔等單元，使用 Delimiter 屬性指定文字的分隔符號。
- XmlWriterTraceListener，是從 TextWriterTraceListener 延伸的類別，追蹤與除錯輸出到 XML 格式的文字檔。

接聽程式(Listener)組態多數會寫在組態檔中，使用<sharedListeners>區段指定可用的接聽程式(Listener)及屬性。

```
<system.diagnostics>
  <sharedListeners>
    <add name="myListener"
        traceOutputOptions="ProcessID, DateTime"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="c:\myListener.log">
    </add>
  </sharedListeners>
</system.diagnostics>
```

組態中的 TraceOutputOption 屬性，可以指定要輸出的項目，可以包含以下屬性值(TraceOptions 列舉型別)：

- None，不寫入項目。
- DateTime，寫入追蹤事件發生的日期和時間。
- Timestamp，寫入追蹤事件發生的時間戳記。
- ProcessId，寫入處理程序的識別代碼。
- ThreadId，寫入執行緒的識別代碼。
- Callstack，寫入程序的呼叫堆疊。

## TraceFilter 類別

追蹤接聽程式(Listener)用來篩選追蹤層級是否輸出。這個類別提供追蹤切換程式(TraceSwitch)之外再一次的過濾動作。TraceFilter 類別是一個抽象基底類別。它有二個延伸類別：

- EventTypeFilter，指定接聽程式依據事件類型過濾追蹤訊息。
- SourceFilter，指定接聽程式依據追蹤來源過濾追蹤訊息。

TraceSwitch 決定是否輸出，假設 TraceSwitch 設為 Warning 時，當追蹤的事件類型(TraceEventType)為 Warning、Error 時訊息會輸出，若事件類型為 Verbose、Info 時訊息就不會輸出。除此之外，可以在接聽程式定義過濾事件類型為 Error 時才接聽該訊息。以下範例便是設定 myListener 接聽程式只接聽 Error 的事件類型。

```
<sharedListeners>
  <add name="myListener"
    type="System.Diagnostics.TextWriterTraceListener"
    initializeData="c:\ErrorEvent.log">
    <filter type="System.Diagnostics.EventTypeFilter"
      initializeData="Error"/>
  </add>
</sharedListeners>
```

## TraceSource 的設計步驟：

在開始使用 TraceSource 類別之前，必須先寫好組態檔，組態檔必須定義好 TraceSwitch、TraceListener 等設定，才可開始在程式中指定 TraceSource。

1. 在 app.config 的<system.diagnostics>區段內寫入<switches>及<sharedListeners>的設定，然後在<sources>區段內寫入追蹤來源<source>及 name 屬性、switchName...等屬性，並在區段內指定接聽程式<listeners>區段。

```
<configuration>
  <system.diagnostics>
    <trace autoflush="true" />
    <sources>
      <source name="TraceSourceApp"
        switchName="sourceSwitch"
        switchType="System.Diagnostics.SourceSwitch">
```

```

<listeners>
    <add name="myListener"/>
</listeners>
</source>
</sources>
<switches>
    <add name="sourceSwitch" value="Warning"/>
</switches>
<sharedListeners>
    <add name="myListener"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="c:\myListener.log">
        <filter type="System.Diagnostics.EventTypeFilter"
            initializeData="Error"/>
    </add>
</sharedListeners>
</system.diagnostics>
</configuration>

```

2. 接著在程式中建立 TraceSource 物件，並在建構函式的參數指定<source>的 name 屬性。

#### Visual Basic

```
Private Shared ts As New TraceSource("TraceSourceApp")
```

#### C#

```
static TraceSource ts = new TraceSource("TraceSourceApp");
```

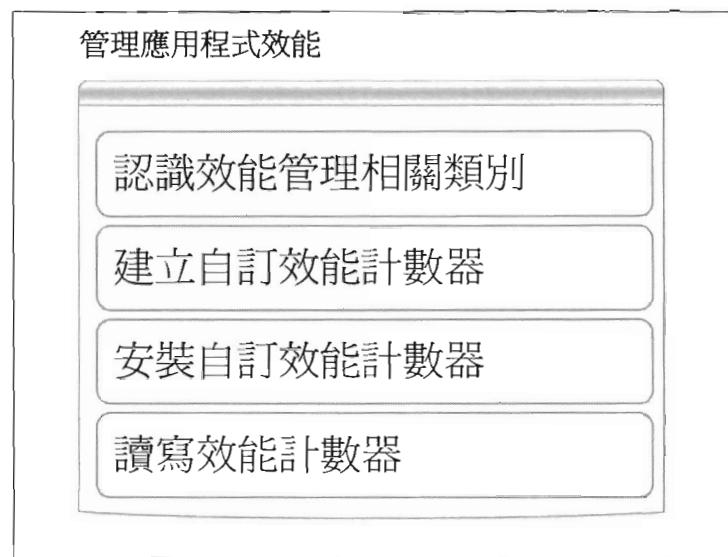
3. 在需要輸出追蹤訊息時呼叫 TraceSource 物件的 TraceEvent 方法，並傳入訊息所屬的類型及訊息代碼，訊息字串。

#### Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    ts.TraceEvent(TraceEventType.Error, 2, "檔案無法開啓")
End Sub
```

#### C#

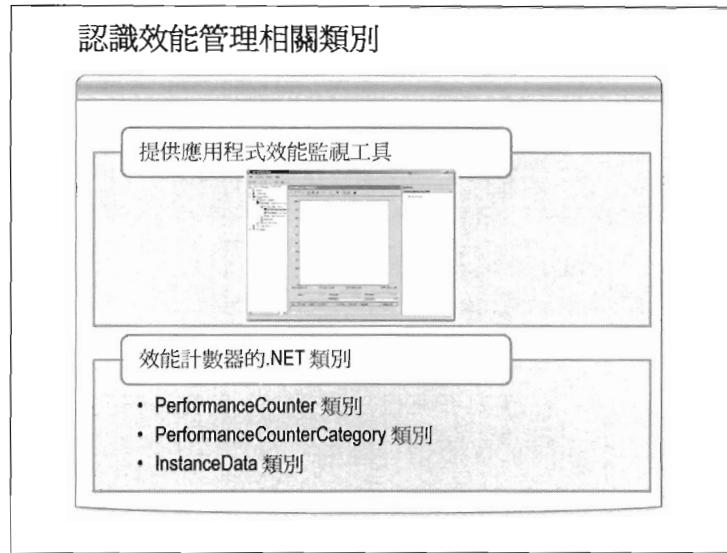
```
private void button1_Click(object sender, EventArgs e)
{
    ts.TraceEvent(TraceEventType.Error, 2, "檔案無法開啓");
}
```



## 管理應用程式效能

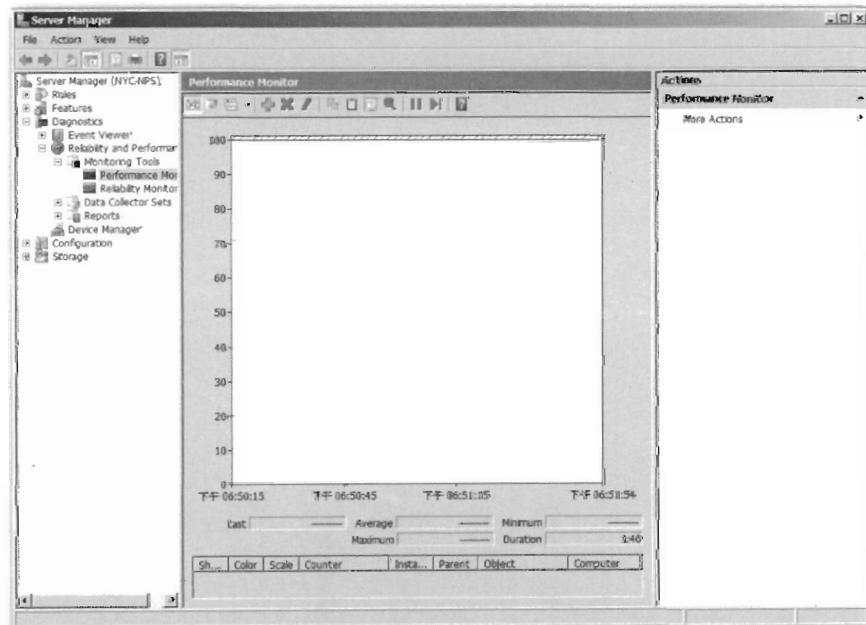
在這一節中你將了解在程式中如何取得效能計數器的數據，以及利用效能計數器的功能記錄應用程式的計數器，這一節包含以下單元：

- 認識效能管理相關類別
- 建立自訂效能計數器
- 安裝自訂效能計數器
- 讀寫效能計數器



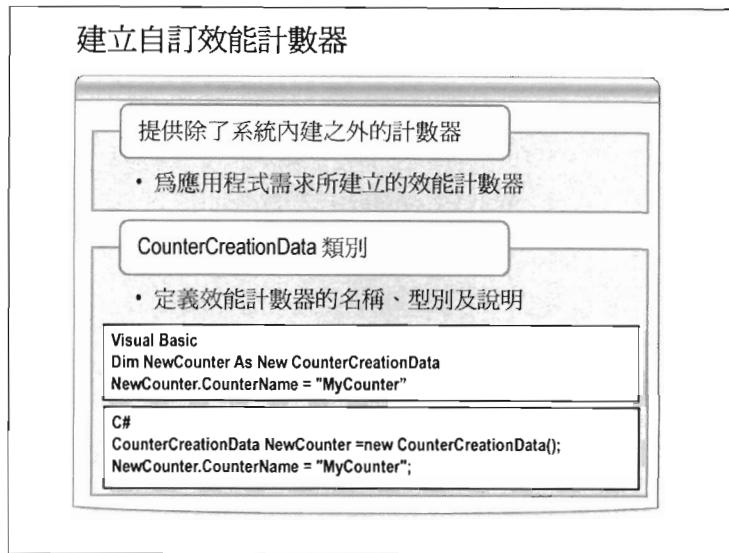
## 效能監視

Windows 作業系統有提供一個「效能」(Performance) 工具，用來對作業系統或是應用程式進行效能監視的工具。效能工具中有許多計數器，每個計數器負責不同事項的監視，例如，實體記憶體的使用量、處理器執行中的時間百分比...等。



.NET Framework 提供一組效能監視與管理的類別，歸屬於 System.Diagnostics 命名空間，包含 PerformanceCounter、PerformanceCounterCategory、InstanceData... 等多項類別。

- PerformanceCounterCategory，計數器的種類。
- PerformanceCounter，計數器項目。
- InstanceData，計數器種類的例項，例如系統有雙處理器，那就會有 0、1 個例項及 Total 例項。



## 建立自訂效能計數器

應用程式也可以將程式執行的效能數據與作業系統的效能工具整合。首先必須先建立效能計數器與種類，然後才在程式中依據狀況增減效能數據值。

CounterCreationData 類別是用來建立自訂計數器，並利用其屬性定義新計數器的名稱、型別以及說明文字。它的常見成員如下：

- CounterName，計數器名稱。
- CounterType，計數器型別。
- CounterHelp，計數器說明文字。

以下範例是自訂效能計數器，並指定其名稱、型別與說明。

```
Visual Basic
Dim NewCounter As New CounterCreationData
NewCounter.CounterName = "MyCounter"
NewCounter.CounterType = PerformanceCounterType.NumberOfItems64
NewCounter.CounterHelp = "這是應用程式自訂的計數器"
```

```
C#
```

```
CounterCreationData NewCounter = new CounterCreationData();
NewCounter.CounterName = "MyCounter";
NewCounter.CounterType = PerformanceCounterType.NumberOfItems64;
NewCounter.CounterHelp = "這是應用程式自訂的計數器";
```

效能計數器必須歸屬於某一個種類下，而通常一個種類下會有多個計數器項目。使用 CounterCreateionData 建立好計數器物件之後，必須將它們加到 CounterCreationDataCollection 物件，CountreCreateionDataCollection 物件是個自訂計數器的集合物件，可以存放多個 CounterCreateionData 物件。

#### Visual Basic

```
Dim col As New CounterCreationDataCollection
col.Add(NewCounter)
```

#### C#

```
CounterCreationDataCollection col =
    new CounterCreationDataCollection();
col.Add(NewCounter);
```

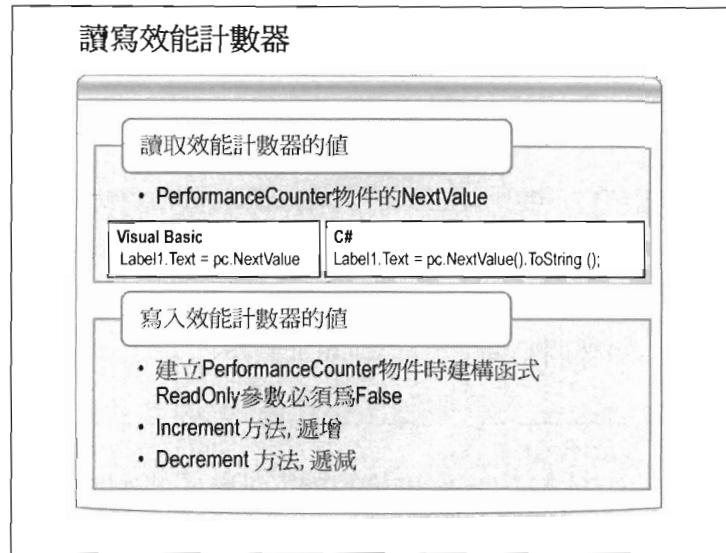
最後要將效能計數器及種類建立在本機系統上，必須使用 PerformanceCounterCategory 類別的 Create 方法。範例程式如下：

#### Visual Basic

```
PerformanceCounterCategory.Create(
    "MyCategory", "自訂計數器種類",
    PerformanceCounterCategoryType.SingleInstance,
    col)
```

#### C#

```
PerformanceCounterCategory.Create(
    "MyCategory", "自訂計數器種類",
    PerformanceCounterCategoryType.SingleInstance,
    col);
```



## 效能計數器

PerformanceCounter 類別是.NET Framework 用來監視效能計數器的類別。提供讀取或寫入效能計數器的功能。它包含以下成員：

- MachineName 屬性，指定或取得效能計數器的電腦名稱。
- CategoryName 屬性，效能計數器的種類，例如 Processor，代表處理器，Member，代表記憶體。每個種類下可含多項計數器。
- InstanceName 屬性，效能計數器種類的例項名稱，像是 Processor (處理器) 一台電腦可以有多顆處理器，指的是哪顆處理器。
- CounterName 屬性，效能計數器的名稱，例如% Processor Time。
- NextValue()，取得下個計數值。

以下範例是在表單類別中建立效能計數器的物件 (PerformanceCounter)，然後在表單載入時設定種類為 Processor (處理器)、計數器為 % Processor Time (處理器執行中的執行緒的時間百分比)、例項名稱為 \_Total (所有的例項)。

```
Dim pc As New PerformanceCounter
Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    pc.CategoryName = "Processor"
    pc.CounterName = "% Processor Time"
    pc.InstanceName = "_Total"
End Sub
```

```
C#
PerformanceCounter pc = new PerformanceCounter();
private void Form2_Load(object sender, EventArgs e)
{
    pc.CategoryName = "Processor";
    pc.CounterName = "% Processor Time";
    pc.InstanceName = "_Total";
}
```

使用 Timer 控制項以每秒計時。在 Timer 控制項的 Tick 事件發生時，在 Label 控制項上顯示新的計數值。

```
Visual Basic
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Timer1.Tick
    Label1.Text = pc.NextValue()
End Sub
```

```
C#
private void Timer1_Tick(object sender, EventArgs e)
{
    Label1.Text = pc.NextValue().ToString();
}
```

## 更新效能計數器數據

在自訂效能計數器之後，應用程式可以依據狀況增減效能計數器數據。只要建立 PerformanceCounter 物件並使用 Increment、IncrementBy、Decrement、DecrementBy 等方法便可為效能計數器數據進行增減量的動作，有個前題，PerformanceCounter 必須建立為可讀寫的狀態方能進行增減量的動作。範例如下：

```
Visual Basic
Dim counter As New PerformanceCounter("MyCategory", _
    "MyCounter", False)
counter.Increment()
```

```
C#
PerformanceCounter counter =new _
    PerformanceCounter("MyCategory", "MyCounter", false);
counter.Increment();
```

### 練習8.3：建立自訂效能計數器

- 瞭解如何建立應用程式專屬的效能計數器。
- 在這個練習中會建立計算訂單建立總數的效能計數器，此計數器歸屬Mod08\_3應用程式。

• 預估實作時間：15分鐘

## 練習 8.3：建立自訂效能計數器

### 目的：

瞭解如何建立應用程式專屬的效能計數器。

### 功能描述：

在這個練習中會建立計算訂單建立總數的效能計數器，此計數器歸屬 Mod08\_3 應用程式。

**預估實作時間：15 分鐘**

### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod08\_3。
3. 開啟 Form1 的設計視窗，從「Toolbox」拖拉二個 Button 到表單上，並設定屬性如下：

控制項	屬性	值
-----	----	---

Button	ID	Button1
	Text	建立效能計數器與種類
Button	ID	Button2
	Text	建立新訂單

4. 若開啟的是 Visual Basic 專案請跳過此步驟。C# 專案必須先匯入命名空間。

```
C#
using System.Diagnostics;
```

5. 進入 Button1\_Click 事件程序，建立效能計數器名稱為「OrderCounter」，將之歸類於 Mod08\_3 的效能種類。

```
Visual Basic
If PerformanceCounterCategory.Exists("Mod08_3") Then
    MessageBox.Show("Mod08_3效能種類已建立")
    Exit Sub
End If

Dim NewCounter As New CounterCreationData
NewCounter.CounterName = "OrderCounter"
NewCounter.CounterType = PerformanceCounterType.NumberOfItems64
NewCounter.CounterHelp = "建立訂單的總數"

Dim col As New CounterCreationDataCollection
col.Add(NewCounter)

PerformanceCounterCategory.Create(
    "Mod08_3",
    "Mod08_3 應用程式",
    PerformanceCounterCategoryType.SingleInstance,
    col)
```

```
C#
if (PerformanceCounterCategory.Exists("Mod08_3"))
{
    MessageBox.Show("Mod08_3效能種類已建立");
    return ;
}

CounterCreationData NewCounter = new CounterCreationData();
NewCounter.CounterName = "OrderCounter";
NewCounter.CounterType = PerformanceCounterType.NumberOfItems64;
NewCounter.CounterHelp = "建立訂單的總數";

CounterCreationDataCollection col = new CounterCreationDataCollection();
```

```
col.Add(NewCounter);

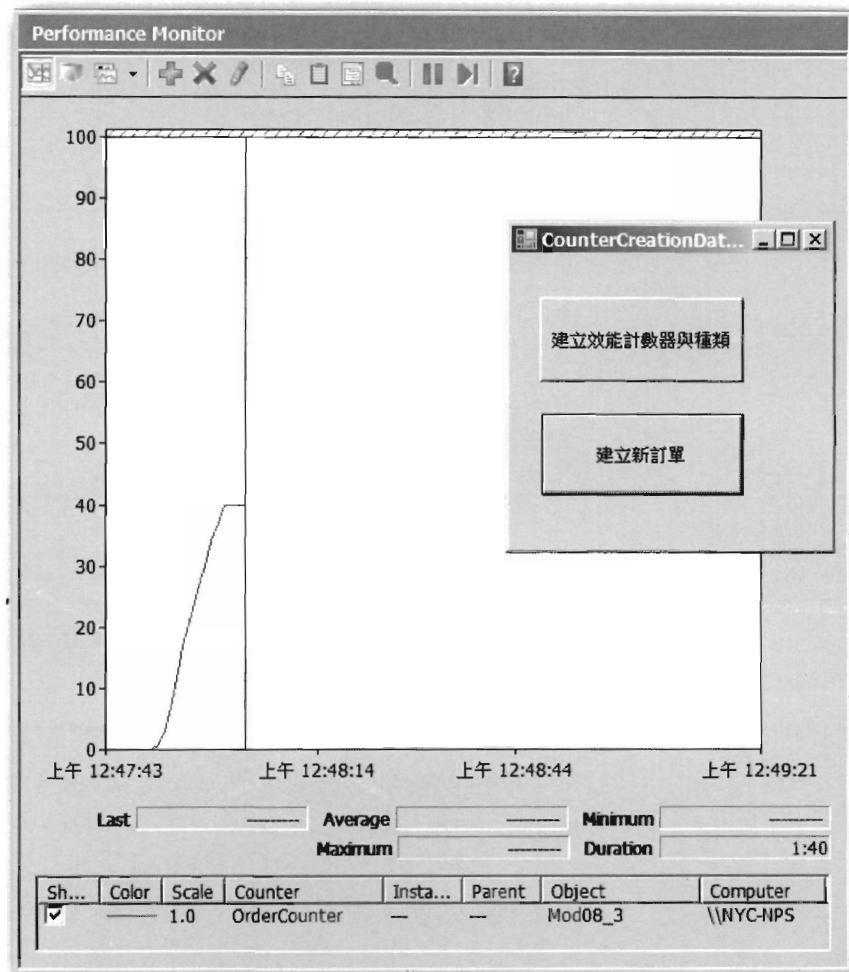
PerformanceCounterCategory.Create(
    "Mod08_3", "Mod08_3 應用程式",
    PerformanceCounterCategoryType.SingleInstance,
    col);
```

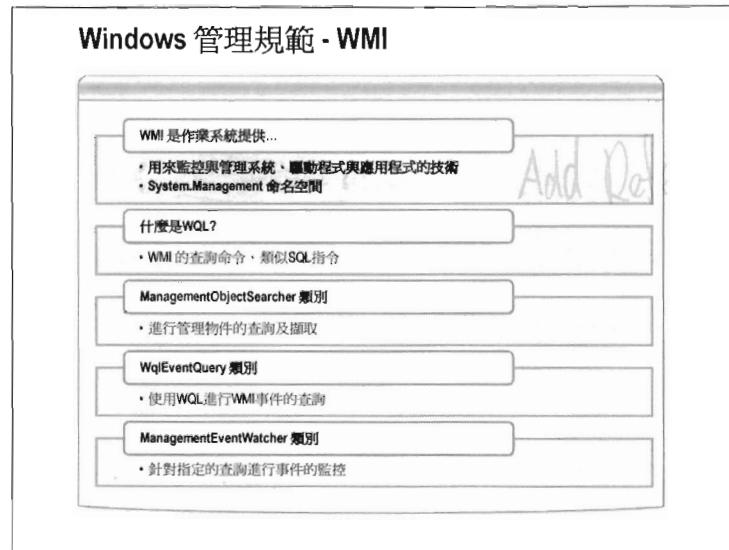
6. 進入 Button2\_Click 事件程序，在建立訂單時，增加 OrderCounter 效能計數器的計數量。

```
Visual Basic
Dim counter As New PerformanceCounter("Mod08_3", "OrderCounter", False)
counter.Increment()
```

```
C#
PerformanceCounter counter = new PerformanceCounter("Mod08_3", "OrderCounter", false);
counter.Increment();
```

7. 按「F5」執行應用程式，按下「建立效能計數器與種類」的按鈕。
8. 開啓效能監看工具。
9. 回到 Mod08\_3.exe 應用程式，多按幾次「建立新訂單」按鈕，並一邊查效能工具，應該會看到 OrderCounter 計數器增量。





## Windows 管理規範 - WMI

什麼是 WMI？WMI 是作業系統提供用來監控與管理系統、驅動程式與應用程式的一種技術。依系統服務、驅動程式、應用程式分成多項類別，每個類別有多種不同的屬性及功能。可以讓開發人員透過 Script 或是.NET 的程式進行系統資訊的查詢、事件監控等管理功能。在存取 WMI 之前，可以先到 MSDN 查詢系統提供的類別。

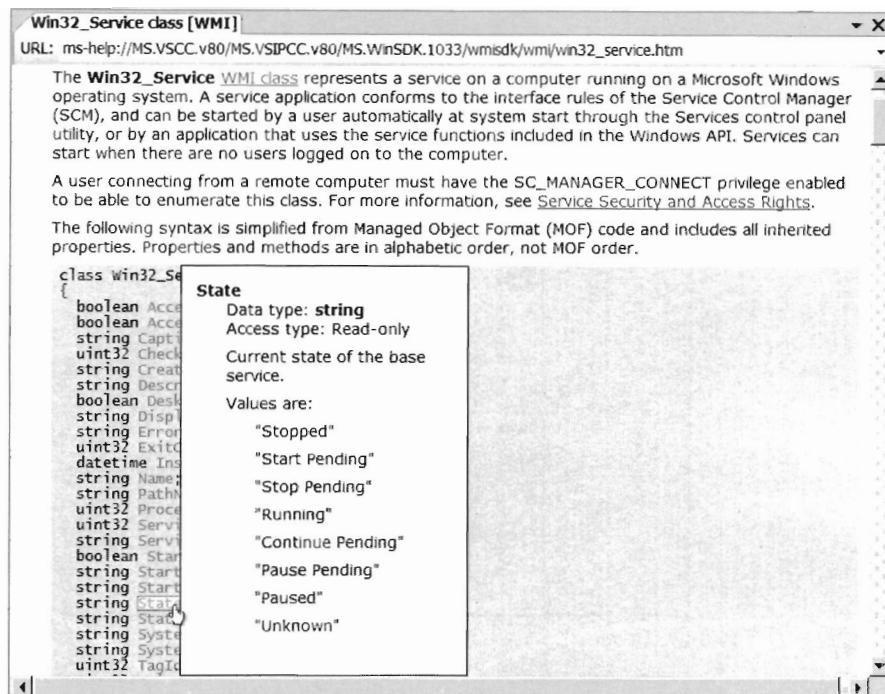
The screenshot shows the MSDN WMI documentation interface. On the left is a navigation pane titled 'Contents' with a 'Filtered by' dropdown set to '(no filter)'. The main pane displays the 'Operating System Classes [WMI]' page. At the top, it shows the URL: ms-help://MS.VSCC.v80/MS.VSPCC.v80/MS.WinSDK.1033/vmsdk/wmi/operating\_sys.htm. Below the URL, a subcategory 'Processes' is selected, with a note: 'The Processes subcategory groups classes that represent system processes and threads.' A table lists three classes under 'Processes':

Class	Description
Win32_Process	Instance class Represents a sequence of events on a Windows system.
Win32_ProcessStartup	Instance class Represents the startup configuration of a Windows process.
Win32_Thread	Instance class Represents a thread of execution.

Below this, another subcategory 'Registry' is shown with a note: 'The class in the Registry subcategory represents the contents of the Windows registry.' A table lists one class under 'Registry':

Class	Description
Win32_Registry	Instance class Represents the system registry on a Windows computer system.

例如，Win32\_Service WMI 類別，提供多種屬性與方法，可以由 MSDN 中查得。



## 什麼是 WQL?

進行 WMI 類別查詢的語言，類似 SQL 查詢指令。例如，要查詢目前的系統執行中的處理序。

```
Select * From Win32_Process
```

或者是查詢目前啓用中的服務。

```
Select * From Win32_Service Where State='Running'
```

### ManagementObjectSearcher 類別

進行管理物件的查詢及擷取功能。可以用來列舉查詢類別，像是處理程序、服務、磁碟機以及網路介面卡...等系統管理物件。使用 WQL 查詢指令及 Get 方法擷取查詢結果。

### WqlEventQuery 類別

使用 WQL 進行 WMI 事件的查詢。與 ManagementEventWatcher 類別配合進行查詢物件的事件監控。

### ManagementEventWatcher 類別

針對指定的查詢進行事件的監控。在 ManagementEventWatcher 類別指定 Query 屬性 (WqlEventQuery 類別) 進行指定系統物件的監控動

作。提供 `WaitForNextEvent` 方法進行同步事件監控。註冊 `EventArrived` 事件及呼叫 `Start` 方法進行非同步事件的監控。

## 如何查詢目前暫停中的服務

使用 ManagementObjectSearcher 類別

- 執行 WQL 查詢
- 使用 Get 方法取得查詢結果

Visual Basic

```
Dim searcher As ManagementObjectSearcher
searcher = New ManagementObjectSearcher("Select * from
Win32_Service where State='Paused'")
For Each pausedService As ManagementObject In searcher.Get
```

C#

```
ManagementObjectSearcher searcher ;
searcher = new ManagementObjectSearcher("Select * from
Win32_Service where State='Paused'");
foreach (ManagementObject pausedService in searcher.Get())
```

## 如何查詢目前暫停中的服務

使用 WQL 及 ManagementObjectSearcher 類別，並使用 Get 方法取得查詢結果。例如，要查詢目前暫停中的服務，系統的服務類別是 Win32\_Service，服務狀態是 State 欄位，它的 WQL 如下：

```
Select * from Win32_Service where State='Paused'
```

在建立 ManagementObjectSearcher 物件時傳入 WQL 查詢，然後透過 ManagementObjectSearcher 物件的 Get 方法，便可列舉目前暫停的服務。

Visual Basic

```
Dim searcher As ManagementObjectSearcher
searcher = New ManagementObjectSearcher("Select * from Win32_
_Service where State='Paused'")
For Each pausedService As ManagementObject In searcher.Get
    ListBox1.Items.Add(pausedService("DisplayName"))
Next
```

C#

```
ManagementObjectSearcher searcher ;
searcher = new ManagementObjectSearcher("Select * from Win32_
```

```
_Service where State='Paused"');
foreach (ManagementObject pausedService in searcher.Get())
{
    ListBox1.Items.Add(pausedService["DisplayName"]);
}
```

### 如何監控新啓動的應用程式(同步)

使用 ManagementEventWatcher 訂閱事件

• 使用 WaitForNextEvent，等待事件發生

Visual Basic

```
Dim query As New WqlEventQuery(" __InstanceCreationEvent",
    New TimeSpan(0, 0, 1), "TargetInstance isa ""Win32_Process""")
Dim watcher As New ManagementEventWatcher(query)
watcher.Options.Timeout = New TimeSpan(0, 0, 10)
Dim mbo As ManagementBaseObject = watcher.WaitForNextEvent()
```

C#

```
WqlEventQuery query = new WqlEventQuery(" __InstanceCreationEvent",
    new TimeSpan(0, 0, 1), "TargetInstance isa \"Win32_Process\"");
ManagementEventWatcher watcher = new ManagementEventWatcher(query);
watcher.Options.Timeout = new TimeSpan(0, 1, 0);
ManagementBaseObject mbo = watcher.WaitForNextEvent();
```

### 如何監控新啓動的應用程式(同步)

使用 Wql 或者建立 WqlEventQuery 物件及 ManagementEventWatcher 可以針對特定的系統物件進行事件的訂閱，以達到監視系統的目的。例如要監視系統新建立的實體，它的類別是

\_\_InstanceCreationEvent，若指定 TargetInstance 欄位為 Win32\_Process 時，代表監視新啓動的處理程序。以下是建立 WqlEventQuery 物件的程式碼：

Visual Basic

```
Dim query As New WqlEventQuery(
    "__InstanceCreationEvent",
    New TimeSpan(0, 0, 1),
    "TargetInstance isa ""Win32_Process""")
```

C#

```
WqlEventQuery query = new WqlEventQuery(
    "__InstanceCreationEvent",
    new TimeSpan(0, 0, 1),
    "TargetInstance isa \"Win32_Process\"");
```

使用 ManagementEventWatcher 物件訂閱指定的查詢，並使用 WaitForNextEvent 方法，等待事件的發生。若有指定 Options.Timeout

屬性，則是設定等待的逾期期限，若逾期訂閱的事件未發生，程式便會產生 Exception。

**Visual Basic**

```
Dim watcher As New ManagementEventWatcher  
watcher.Query = query  
watcher.Options.Timeout = New TimeSpan(0, 0, 10)  
MessageBox.Show("開啓應用程式 (notepad.exe) 以觸發事件。")  
Dim mbo As ManagementBaseObject = _  
    watcher.WaitForNextEvent()  
  
MessageBox.Show(String.Format(  
    "應用程式 {0} 已建立, 路徑是: {1}", _  
    CType(mbo("TargetInstance"), _  
        ManagementBaseObject)("Name"), _  
    CType(mbo("TargetInstance"), _  
        ManagementBaseObject)("ExecutablePath")))  
  
watcher.Stop()
```

**C#**

```
ManagementEventWatcher watcher = new ManagementEventWatcher(query);  
watcher.Options.Timeout = new TimeSpan(0, 1, 0);  
MessageBox.Show("開啓應用程式 (notepad.exe) 以觸發事件。");  
ManagementBaseObject mbo = watcher.WaitForNextEvent();  
  
MessageBox.Show(  
    String.Format("應用程式 {0} 已建立, 路徑是: {1}",  
        ((ManagementBaseObject)(mbo["TargetInstance"]))["Name"],  
        ((ManagementBaseObject)(mbo["TargetInstance"]))["ExecutablePath"]));  
  
watcher.Stop();
```

### 如何監控被暫停的服務(非同步)

註冊 EventArrived 事件及呼叫 Start 方法

```

Visual Basic
Dim equery As New WqlEventQuery("__InstanceModificationEvent",
    New TimeSpan(0, 0, 2), "TargetInstance isa ""Win32_Service"" and
    TargetInstance.State='Paused'")
Dim watcher As New ManagementEventWatcher
AddHandler watcher.EventArrived, AddressOf watcher_EventArrived
watcher.Start()

C#
WqlEventQuery equery=new WqlEventQuery("__InstanceModificationEvent",
    new TimeSpan(0, 0, 2), "TargetInstance isa \\"Win32_Service\\" and
    TargetInstance.State='Paused'");
ManagementEventWatcher watcher = new ManagementEventWatcher(equery);
watcher.EventArrived += new EventArrivedEventHandler(
    watcher_EventArrived);
watcher.Start();

```

### 如何監控被暫停的服務(非同步)

若要監視目前執行中的程式修改狀態的話，類別是 `_InstanceModificationEvent`。監視條件如為服務時則指定 `TargetInstance` 欄位為 `Win32_Service`，並且為暫停狀態指定 `TargetInstance.State` 欄位為'Paused'。建立 `WqlEventQuery` 物件的程式如下。

```

Visual Basic
Dim equery As New WqlEventQuery(
    "__InstanceModificationEvent",
    New TimeSpan(0, 0, 2),
    "TargetInstance isa ""Win32_Service"" and TargetInstance.St
ate='Paused'")

```

```

C#
WqlEventQuery equery=new WqlEventQuery(
    "__InstanceModificationEvent",
    new TimeSpan(0, 0, 2),
    "TargetInstance isa \\"Win32_Service\\" and TargetInstance.St
ate='Paused'");

```

建立 `ManagementEventWatcher` 物件、註冊 `EventArrived` 的事件，然後呼叫 `watch` 物件的 `Start` 方法。

Visual Basic

```
Dim watcher As ManagementEventWatcher
watcher = New ManagementEventWatcher(equery)
AddHandler watcher.EventArrived, AddressOf watcher_EventArrived
watcher.Start()
```

C#

```
ManagementEventWatcher watcher = new ManagementEventWatcher(equery);
watcher.EventArrived += 
    new EventArrivedEventHandler( watcher_EventArrived);
watcher.Start();
```

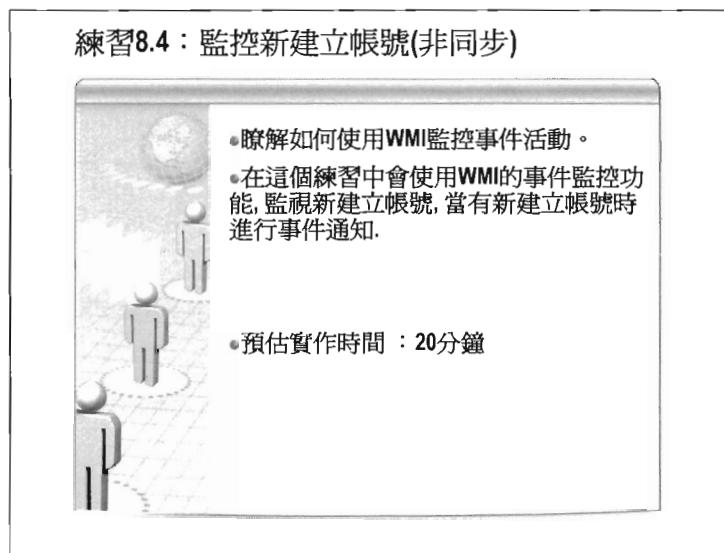
在 EventArrived 事件程序中，可以使用第二個參數 EventArrivedEventArgs 物件的 NewEvent，取得觸發的管理物件，並且取得管理物件中的欄位值，像是 DisplayName 是 Win32\_Service 類別的服務顯示名稱。

Visual Basic

```
Sub watcher_EventArrived(ByVal sender As Object, ByVal e As EventArrivedEventArgs)
    Dim mbo As ManagementBaseObject
    mbo = CType(e.NewEvent, ManagementBaseObject)
    Dim serviceMO As ManagementBaseObject
    serviceMO = CType(mbo("TargetInstance"), ManagementBaseObject)
    MessageBox.Show(serviceMO("DisplayName"))
End Sub
```

C#

```
void watcher_EventArrived(Object sender, EventArrivedEventArgs e )
{
    ManagementBaseObject mbo ;
    mbo = (ManagementBaseObject)e.NewEvent;
    ManagementBaseObject serviceMO ;
    serviceMO = (ManagementBaseObject)(mbo["TargetInstance"]);
    MessageBox.Show("暫停中的服務：" + serviceMO["DisplayNam
e"]);
}
```



## 練習 8.4：監控新建立帳號(非同步)

目的：

瞭解如何使用 WMI 監控事件活動。

功能描述：

在這個練習中會使用 WMI 的事件監控功能，監視新建立帳號，當有新建立帳號時進行事件通知。

預估實作時間：20 分鐘

實作步驟：

1. 開啓 MSDN 線上說明，查詢「Win32\_Account」。
2. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
3. 建立一個新的 Windows Application 專案，將專案命名為 Mod08\_4。
4. 開啓 Form1 的設計視窗，從「Toolbox」拖拉二個 Button 到表單上，並設定屬性如下：

控制項	屬性	值
Label	ID	Label1
	Text	非同步處理, 監控新建立帳號
Button	ID	Button1
	Text	開始監控
Button	ID	Button2
	Text	停止監控

5. 加入參考「System.Management」。

6. 進入 Form1 程式碼檢視，匯入命名空間  
「System.Management」。

```
Visual Basic
Imports System.Management
```

```
C#
using System.Management;
```

7. 嘗試宣告表單層級的變數：

```
Visual Basic
Dim watcher As ManagementEventWatcher
```

```
C#
ManagementEventWatcher watcher;
```

8. 進入 Button1\_Click 事件程序，開始監控建立帳號的事件。

```
Visual Basic
Dim equery As New WqlEventQuery(
    "__InstanceCreationEvent",
    New TimeSpan(0, 0, 2),
    "TargetInstance isa ""Win32_Account""")
watcher = New ManagementEventWatcher(equery)
AddHandler watcher.EventArrived, AddressOf watcher_EventArrived
watcher.Start()
Button1.Enabled = False
```

```
C#
WqlEventQuery equery = new WqlEventQuery(
    "__InstanceCreationEvent",
    new TimeSpan(0, 0, 2),
```

```

    "TargetInstance isa \"Win32_Account\"");
watcher = new ManagementEventWatcher(equery);
watcher.EventArrived += new EventArrivedEventHandler(watcher
_EventArrived);
watcher.Start();
Button1.Enabled = false;
Button2.Enabled = true;

```

9. 編寫 EventArrived 事件程序，處理事件通知時的訊息顯示。

Visual Basic

```

Sub watcher_EventArrived(ByVal sender As Object, ByVal e As Eve
ntArrivedEventArgs)
    Dim mbo As ManagementBaseObject
    mbo = CType(e.NewEvent, ManagementBaseObject)
    Dim serviceMO As ManagementBaseObject
    serviceMO = CType(mbo("TargetInstance"), _
        ManagementBaseObject)
    MessageBox.Show(String.Format(
        "新建立帳號:{0}" & vbCrLf & "SID:{1}",
        serviceMO("Name"), serviceMO("SID")))
End Sub

```

C#

```

void watcher_EventArrived(Object sender, EventArrivedEventArgs
e)
{
    ManagementBaseObject mbo;
    mbo = (ManagementBaseObject)e.NewEvent;
    ManagementBaseObject serviceMO;
    serviceMO = (ManagementBaseObject)(mbo["TargetInstance
"]);
    MessageBox.Show( string.Format (
        "新建立帳號:{0}\r\nSID:{1}",
        serviceMO["Name"] , serviceMO["SID"]));
}

```

10. 進入 Button2\_Click 事件程序，停止監控。

Visual Basic

```

If watcher Is Nothing Then
    Exit Sub
End If

watcher.Stop()
Button1.Enabled = True
Button2.Enabled = False

```

C#

```
if (watcher == null)
```

```
return;  
  
watcher.Stop();  
Button1.Enabled = true;  
Button2.Enabled = false;
```

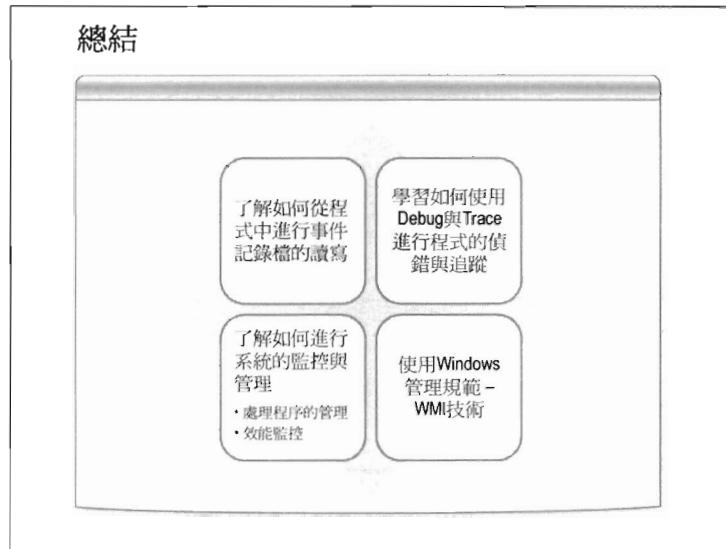
11. 在 Form1\_FormClosed 事件，執行「停止監控」按鈕。

```
Visual Basic  
Button2.PerformClick()
```

```
C#  
Button2.PerformClick();
```

12. 按「F5」執行應用程式，按下「開始監控」按鈕。

13. 建立一個新的使用者帳號，在帳號建立完成之後，應用程式將會顯示訊息方塊，並顯示新帳號及其 SID。



在這個章節你應該要熟悉如何透過 EventLog 物件進行事件記錄檔的寫入應用程式的事件記錄，以及如何建立、刪除事件記錄(Event Log)與事件來源 (Event Source) 的註冊。

另外，必須熟悉使用 Debug 物件進行應用程式的偵錯，以及 TraceSource 物件與組態檔進行應用程式的追蹤作業。

在系統方面，你也應該要熟悉如何使用 Process 類別與物件列舉目前系統正在執行中的處理程序，以及如何建立新的處理程序。還有，應用程式如何使用 PerformanceCounter...等類別與物件進行即有效能計數器的數據收集與建立應用程式專屬的自訂效能計數器。更進一步的，要存取更多作業系統中像是驅動程式、網路卡、服務...等系統物件，就要熟悉如何使用 WMI 所提供的多種類別與物件，以進行系統目前狀態的查詢或是狀態的事件訂閱與監視。

---

在這個章節中將介紹如何在程式中控制處理序的啓動、關閉及列表，如何在程式中將事件記錄在作業系統的事件檢視記錄檔，以及如何利用.NET 提供的 Debug 及 Trace 進行應用程式的除錯及追蹤功能。作業系統提供了一個效能檢視器的功能，它除了可以監控作業系統及週邊效能計數器的數據也容許你的應用程式自訂效能計數器。如何監控效能數據、如何自訂應用程式專屬的效能計數器，這章也將介紹之。

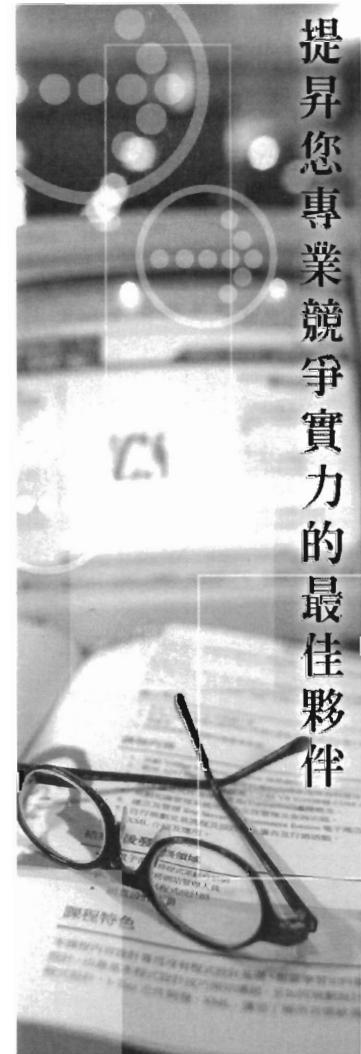
# 第九章：建立多執行緒 應用程式

## 本章大綱

執行緒基本觀念.....	4
建立執行緒.....	5
Thread 物件常用成員.....	8
執行緒生命週期.....	9
中止執行緒執行.....	10
練習 9.1：建立多執行緒應用程式.....	11
執行緒集區(ThreadPool) .....	14
執行緒執行環境(Execution Context).....	16
練習 9.2：使用 ThreadPool 物件.....	17
Timer 物件類別.....	19
多執行緒中的共用資源.....	20
Synchronization Lock .....	21
Monitor 類別.....	22
Mutex 類別.....	23
Semaphore 類別.....	24
ReaderWriterLock 類別.....	25
非同步程式開發模型 (Asynchronous Programming Model).....	26
判斷背景執行緒是否完成.....	27
練習 9.3：使用 APM 建立多執行緒應用程式.....	28

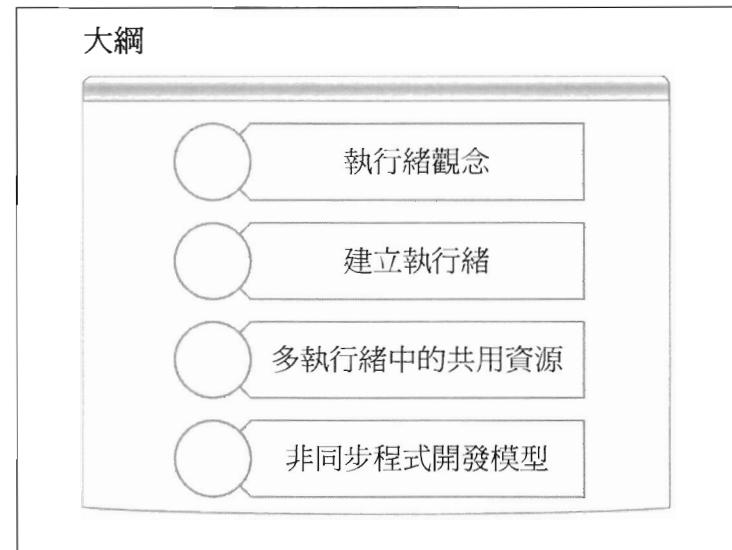
作者：

蘿慧真





---

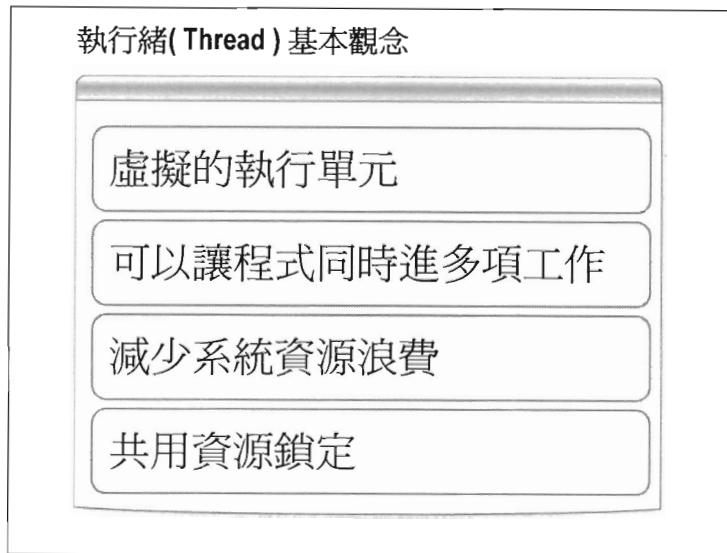


---

在這個章節中將介紹什麼是執行緒，如何撰寫執行緒的程式及在多執行緒的程式中如何共用資源，以及非同步程式開發模型。

本章介紹以下主題：

- 執行緒觀念
- 建立執行緒
- 多執行緒中的共用資源
- 非同步程式開發模型 (Asynchronous Programming Model)

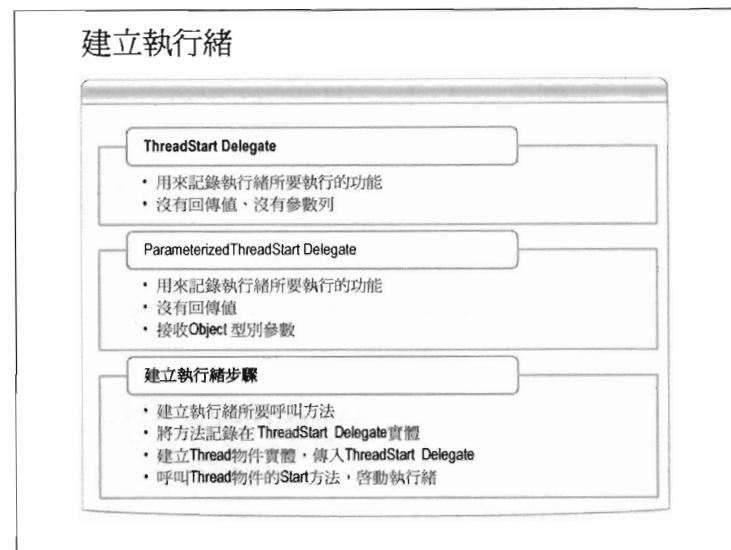


## 執行緒基本觀念

一般的應用程式在開啓時，執行環境會自動建立一個執行緒執行程式中的功能，稱之為主執行緒 (main thread)。因為只有一個執行緒在處理程式，若是程式中有許多需要長時間執行的動作，效能就會因此受到影響。如果想要讓程式可以一次處理多件工作的話，你可以在程式中新增執行緒。

每一個程式中的執行緒都是一個虛擬的執行單元，建立之後就會在程式的背景執行交付的工作。像是程式在開啓網路資源或是開啓資料連線的時候，都會有一些等待的時間，建立多執行緒應用程式最大的好處，就是可以利用這一些系統空閒的時間執行其他的工作，增加系統資源的使用效率。現在有許多新的機器都配備多處理器的硬體環境，如果把多執行緒的程式執行在多處理器的環境中，將可以發揮更強大的功能。

但是多執行緒應用程式也不是沒有缺點，最顯而易見的就是會把程式變得更複雜，同時也不容易除錯。再者，在多執行緒應用程式中需要處理共用資源鎖定(synchronization)，以及避免程式死結(dead lock)等，都是使用時必須要考慮的。



## 建立執行緒

.NET Framework 中將控制執行緒的物件架構定義在 System.Threading 命名空間中。

### ThreadStart Delegate

在.NET 平台的執行緒架構中，每一個執行緒的工作，是由 ThreadStart delegate 物件傳入的，因此在建立執行緒執行工作之前，必須要先將方法的參考記錄在 ThreadStart delegate 之中。ThreadStart delegate 可以參考的方法是沒有回傳值，也沒有接收參數的方法。例如要使用背景執行緒印出現在的時間，可以先建立一個 Work 的方法如下：

```
Visual Basic
Private Shared Sub Work()
    Console.WriteLine(DateTime.Now.ToString())
End Sub
```

```
C#
private static void Work() {
    Console.WriteLine(DateTime.Now.ToString());
}
```

接著在主程式中，將 Work 方法的參考記錄在 ThreadStart delegate 物件中，再傳入 Thread 物件中就可以了：

```
Visual Basic
Shared Sub Main(ByVal args() As String)
    Console.WriteLine("Applicetion Start.")

    Dim doSomething As ThreadStart = New ThreadStart(W
ork)
    Dim worker As Thread = New Thread(doSomething)
    worker.Start()

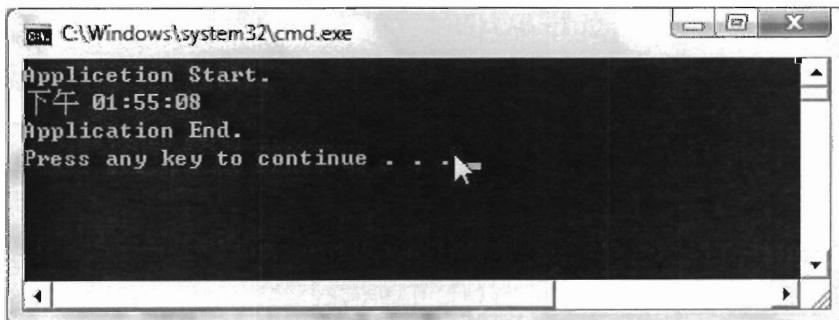
    Console.WriteLine("Application End.")
End Sub
```

```
C#
static void Main(string[] args)
{
    Console.WriteLine("Applicetion Start.");

    ThreadStart doSomething = new ThreadStart(Work);
    Thread worker = new Thread(doSomething);
    worker.Start();

    Console.WriteLine("Application End.");
}
```

呼叫 Thread 物件的 Start 方法就可以啓動執行緒。上面的程式執行結果如下：



### ParameterizedThreadStart Delegate

在預設的 ThreadStart Delegate 定義中，需要 Thread 物件執行的工作必須要寫成沒有接收參數、也沒有回傳值的方法，才可以傳入執行緒中執行。但是實際上當執行緒執行時，我們經常需要從外界傳入資料讓執行緒運算。所以在.NET Framework 2.0 版時增加 ParameterizedThreadStart Delegate 的定義。ParameterizedThreadStart

Delegate 可以參考到接收一個 Object 型別參數、沒有回傳值的方法，因此在執行緒執行時，就可以使用 Start 方法接收外部資料到執行緒中使用。參考下面的例子，首先建立一個接收 Object 型別參數的方法：

Visual Basic

```
Private Shared Sub Work(ByVal name As Object)
    Console.WriteLine("Work with parameter: " + name)
End Sub
```

C#

```
private static void Work(object name) {
    Console.WriteLine("Work with parameter: " + name);
}
```

接著在程式中使用 ParameterizedThreadStart Deleget，就可以傳資料到執行緒中使用了：

Visual Basic

```
Dim doSomething AS ParameterizedThreadStart
    = New ParameterizedThreadStart(Work)
Dim worker As Thread = New Thread(doSomething)
worker.Start("John")
```

C#

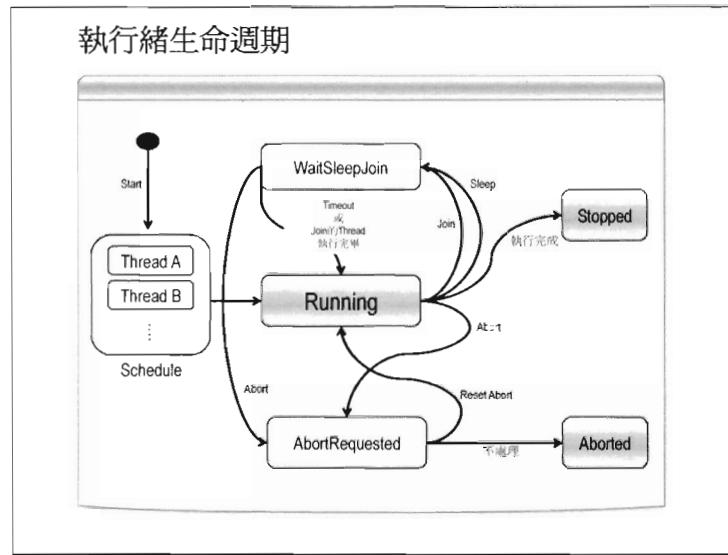
```
ParameterizedThreadStart doSomething
    = new ParameterizedThreadStart(Work);
Thread worker = new Thread(doSomething);
worker.Start("John");
```



## Thread 物件常用成員

建立 Thread 物件實體之後，可以透過 Thread 物件成員取得或設定執行緒狀態，常用的執行緒成員如下：

- Name 屬性：取得或是設定執行緒的名稱。
- Priority 屬性：取得或是設定執行緒的優先權，執行緒按照優先權順序執行。
- ThreadState 屬性：取得目前執行緒的狀態。
- Join 方法：停止目前工作，將執行緒插入執行。
- Thread.Sleep 方法：中止目前執行緒一段時間再繼續執行，中止期間其他的執行緒可以插入執行。



## 執行緒生命週期

程式中所建立的執行緒，實際上是由作業系統的環境執行。一般的機器因為只有一顆處理器(CPU)，因此雖然可以建立多個執行緒，但是並不能同時執行，必須由作業系統排程之後決定執行的順序，一般是優先權高的執行緒會先被執行。Windows 作業系統環境具有分時多工的特性，因此執行緒的管理較為簡單。當程式呼叫 Thread 物件的方法，就可以改變執行緒的狀態。下面是幾個執行緒生命週期中的主要狀態：

### 1. Unstarted :

當 Thread 物件被建立，但是並未進入執行狀態時，即為 Unstarted 狀態。

### 2. Running :

當呼叫 Thread 物件的 start 方法之後，執行緒被執行時，就進入 Running 狀態。還有比較特殊的情況，就是當執行緒從暫停(WaitSleepJoin) 狀態或是呼叫 ResetAbort 方法恢復執行時，也會進入 Running 狀態。

### 3. WaitSleepJoin :

當呼叫 Thread 物件的 Join 方法、Sleep 靜態方法、或是 Monitor.Wait 方法時，都會讓執行緒進入 WaitSleepJoin 狀態。必須要等到暫停的原因解除之後，才會繼續恢復執行。

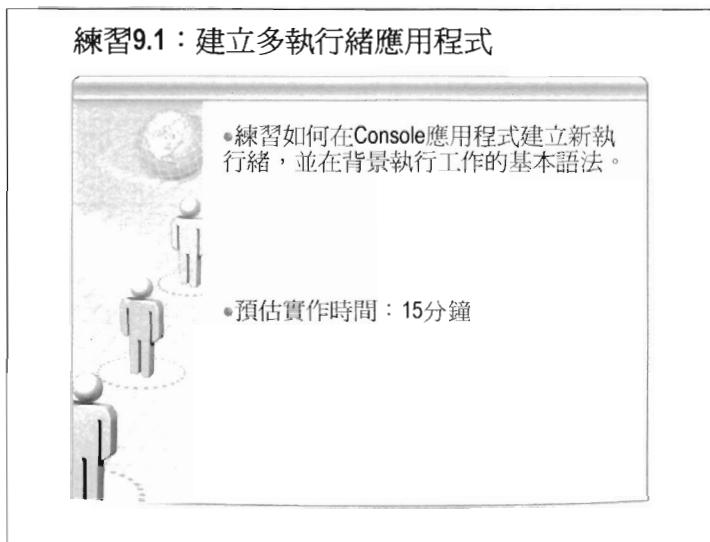
#### 4. AbortRequested：

當呼叫 Thread 物件的 Abort 方法，執行緒就進入 AbortRequested 狀態。如果程式沒有呼叫 ResetAbort 方法，則執行緒就會停止執行(Abort)。

執行緒生命週期的狀態，可以透過 Thread 物件的 ThreadState 屬性取得。但一般執行緒狀態只與偵錯有關。程式碼不應該使用執行緒狀態來同步處理執行緒的活動。

## 中止執行緒執行

除了新增執行緒之外，程式通常也需要中止執行緒執行。從前面 Thread 物件的生命週期介紹中知道，程式可以呼叫 Thread 物件的 Abort 方法中止執行緒。當 Abort 方法被呼叫時，執行環境會在執行緒中產生 ThreadAbortException；除非有使用 Thread.ResetAbort 方法解除 Abort 狀態，要不然執行緒就會中止執行。



## 練習 9.1：建立多執行緒應用程式

目的：

練習如何在 Console 應用程式建立新執行緒，並在背景執行工作的基本語法。

功能描述：

在程式中建立兩個執行緒，並且讓這兩個執行緒分別在背景中執行交互運算動作。

預估實作時間：15 分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Console Application 專案，將專案命名為 Mod09\_1。
3. 在 Module1.vb (C#為 Program.cs)類別檔最前面引用 **System.Threading** 命名空間。

```
Visual Basic
Imports System.Threading
```

```
C#
using System.Threading;
```

4. 宣告無接收參數的 **Print** 方法，執行 20 次迴圈，每次印出自前執行緒的編號以及執行次數，並休息 1 秒：

```
Visual Basic
Sub Print()
    Dim i as Integer
    For i = 1 To 20
        Console.WriteLine("Thread ID:{0}-->Count:{1}", _
            Thread.CurrentThread.ManagedThreadId,i)
        Thread.Sleep(1000)
    Next i
End Sub
```

```
C#
static void Print() {
    for (int i = 1; i <= 20; i++) {
        Console.WriteLine("Thread ID:{0}-->Count:{1}",_
            Thread.CurrentThread.ManagedThreadId,i);
        Thread.Sleep(1000);
    }
}
```

5. 宣告另外一個多載化的 **Print** 方法，接收 **Object** 型別參數，並在方法中將參數值設為迴圈的起始變數，一樣執行迴圈 20 次，其餘動作與步驟 4 所建立的 **Print** 方法相同：

```
Visual Basic
Sub Print(ByVal obj As Object)
    Dim startno As Integer = CType(obj, Integer)
    Dim i as Integer
    For i = startno To (startno + 19)
        Console.WriteLine("Thread ID:{0}-->Count:{1}", _
            Thread.CurrentThread.ManagedThreadId,i)
        Thread.Sleep(1000)
    Next i
End Sub
```

```
C#
static void Print(object obj) {
    int startno = (int)obj;
```

```

        for (int i = startno; i <= startno + 19; i++) {
Console.WriteLine("Thread ID:{0}-->Count:{1}",
                  Thread.CurrentThread.ManagedThreadId,i);
Thread.Sleep(1000);
}
}

```

6. 在程式進入點 Main 方法中建立兩個新執行緒，並利用 ThreadStart 物件與 ParameterizedThreadStart 物件紀錄執行緒執行方法位置，傳入執行緒中：

<b>Visual Basic</b> Dim starter As New ThreadStart(AddressOf Print) Dim paramStart As New ParameterizedThreadStart(AddressOf Print) Dim t1 As New Thread(starter) Dim t2 As New Thread(paramStart)
--

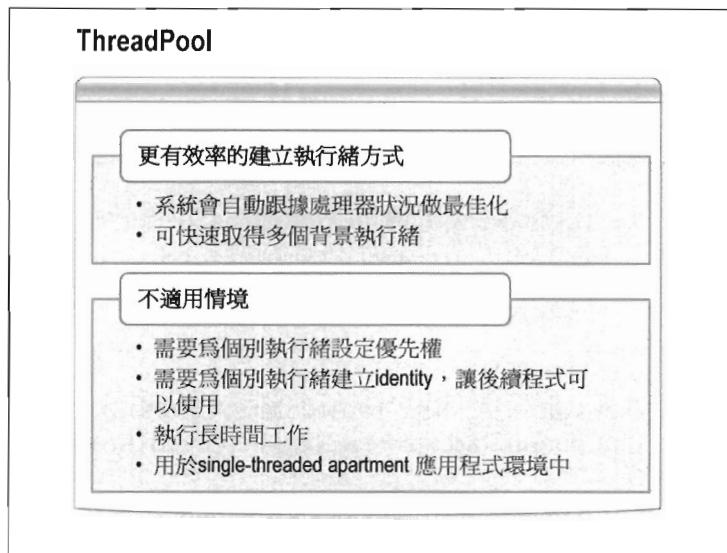
<b>C#</b> ThreadStart starter = new ThreadStart(Print); ParameterizedThreadStart paramStart = new ParameterizedThreadStart(Print); Thread t1 = new Thread(starter); Thread t2 = new Thread(paramStart);
--

7. 接著呼叫兩個執行緒物件的 **Start** 方法，並將接收參數的執行緒的起始數字設為 15，啓動執行緒：

<b>Visual Basic</b> t1.Start() t2.Start(15)
---

<b>C#</b> t1.Start(); t2.Start(15);
---

8. 從功能表選單中，選取「Debug」→「Start Debugging」，執行專案。



## 執行緒集區(ThreadPool)

許多應用程式建立的執行緒把很多的時間花費在休眠狀態，等著事件發生；也有的執行緒進入休眠狀態後只負責定時甦醒，來輪詢變更或更新狀態的資訊。使用執行緒集區可以讓系統更有效率的分配處理器的資源給執行緒，因此若是程式中需要快速在背景執行執行一些簡單的運算動作，就可以直接從集區中取出執行緒執行。執行完的執行緒會先暫存在集區中，下次程式需要執行緒執行工作時，就可以直接從集區中重覆使用執行緒，節省建立新執行緒所需要消耗的系統資源。

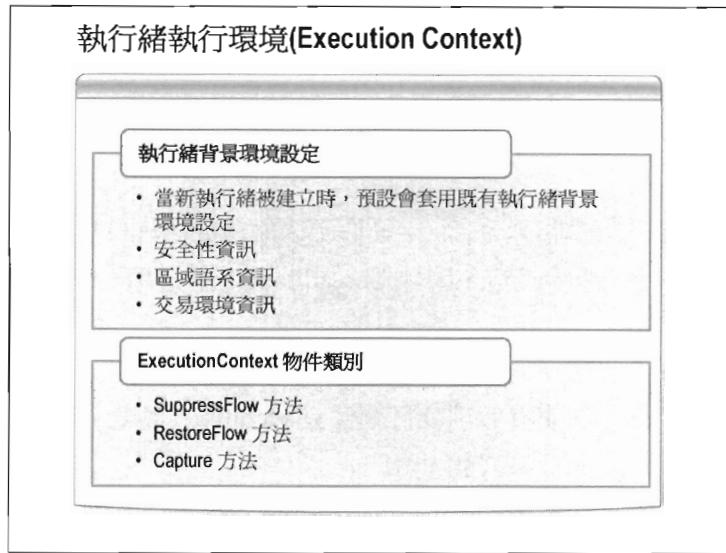
在.NET Framework 中提供了` ThreadPool` 的類別，讓你的程式可以直接存取執行環境中的執行緒集區。利用 `ThreadPool.QueueUserWorkItem` 靜態方法，就可以從目前應用程式集區中取出一個執行緒，並且使用 `WaitCallback Delegate` 傳入工作執行：

```
Visual Basic
Dim work As New WaitCallback(WorkMethod)
If ThreadPool.QueueUserWorkItem(work) then
    Console.WriteLine("OK!")
End If
```

```
C#
WaitCallback work = new WaitCallback(WorkMethod);
If (ThreadPool. QueueUserWorkItem(work)) {
    Console.WriteLine("OK!");
}
```

但是因為執行緒是從集區取出來使用的，因此程式對於該執行緒就無法做太多的設定，因此 ThreadPool 不適合用在下面情境：

1. 需要為個別執行緒設定優先權。
2. 需要為個別執行緒建立 identity，讓後續程式可以使用。
3. 執行長時間工作。
4. 用於 single-threaded apartment 應用程式環境中。

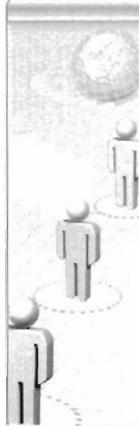


## 執行緒執行環境(Execution Context)

當程式建立新執行緒時，既有執行緒的執行環境(Execution Context)的設定，像是安全性的資訊、區域語系的資訊、以及交易環境的資訊等，都會傳遞到新執行緒中套用。但是如果將這一些資料，傳遞給不需要使用這一些執行環境資訊的執行緒，將浪費一些不必要的系統資源，因此.NET Framework 2.0 之後的版本提供了 ExecutionContext 類別，可以讓你管理目前執行緒的執行環境。如果執行環境的設定不需要傳遞給新的執行緒使用，你可以在建立新執行緒之前呼叫 ExecutionContext.SuppressFlow 靜態方法，暫停將執行環境傳遞給新執行緒的動作；要恢復繼續執行的話，再呼叫 ExecutionContext.RestoreFlow 靜態方法，或是 ExecutionContext 實體的 Undo 方法就可以了。

ExecutionContext 類別也提供 Capture 靜態方法，複製目前執行緒的執行環境，然後再使用 ExecutionContext.Run 靜態方法，可以將程式動態執行在指定的執行環境中。

### 練習9.2：使用ThreadPool物件



•練習使用 ThreadPool 物件建立多執行緒應用程式。

•預估實作時間：8分鐘

## 練習 9.2：使用 ThreadPool 物件

### 目的：

練習使用 ThreadPool 物件建立多執行緒應用程式。

### 功能描述：

修改練習 9.1 的程式，換成使用 ThreadPool 建立程式所需要的背景執行緒執行 Print 方法。。

預估實作時間：8 分鐘

### 實作步驟：

1. 從「\Practices\VB 或 CS\Mod09\_2\Starter\Mod09\_2」開啓 Mod09\_1.sln 檔案。
2. 刪除或註解在 Main 方法中，練習 9.1 所加入的程式碼。
3. 在 Main 方法中，建立一個名為 **cb** 的 WaitCallback 的委派物件，並記錄 Print 方法的位置。

Visual Basic

```
Dim cb As New WaitCallback(AddressOf Print)
```

```
C#
WaitCallback cb = new WaitCallback(Print);
```

4. 呼叫 ThreadPool.QueueUserWorkItem 方法，傳入 WaitCallback 物件以及執行緒方法執行時所需要傳入的計數器初始值：

```
Visual Basic
ThreadPool.QueueUserWorkItem(cb, 1)
ThreadPool.QueueUserWorkItem(cb, 15)
```

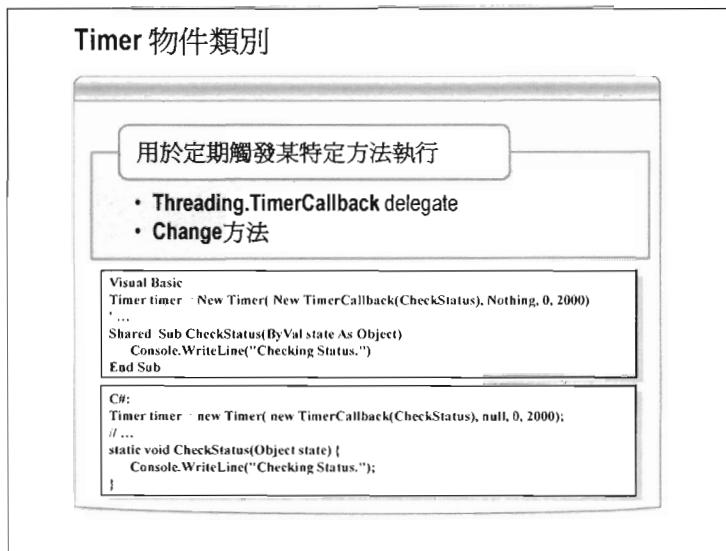
```
C#
ThreadPool.QueueUserWorkItem(cb, 1);
ThreadPool.QueueUserWorkItem(cb, 15);
```

5. 最後呼叫 Console.Read 方法，結束程式編輯。

```
Visual Basic
Console.Read( )
```

```
C#
Console.Read( );
```

6. 按下「F5」鍵，執行專案並檢視執行結果。



## Timer 物件類別

在 System.Threading 命名空間中，定義了一個 Timer 的物件類別，可以設定間隔一段時間觸發某方法。透過 Timer 觸發的方法參考必須要先記錄到 TimerCallback Delegate 物件之後，再傳到 Timer 物件中執行，參考範例如下：

```

Visual Basic
Timer timer = New Timer(
    New TimerCallback(CheckStatus), Nothing, 0, 2000)
' ...
Shared Sub CheckStatus(ByVal state As Object)
    Console.WriteLine("Checking Status.")
End Sub

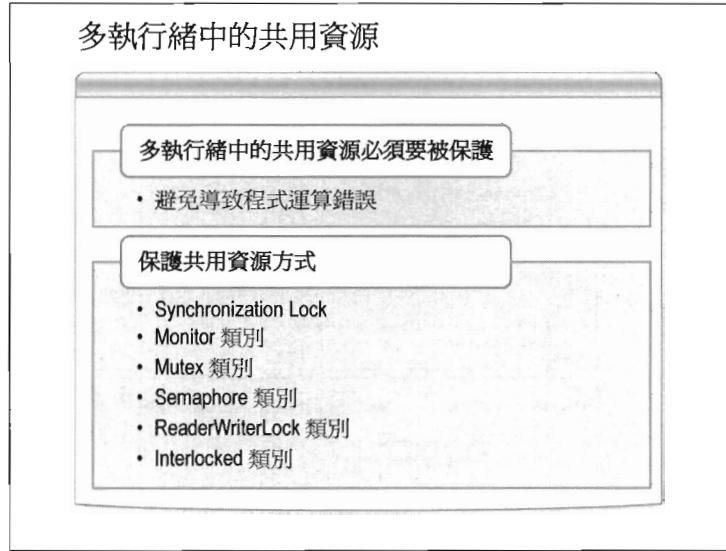
```

```

C#
Timer timer = new Timer(
    new TimerCallback(CheckStatus), null, 0, 2000);
// ...
static void CheckStatus(Object state) {
    Console.WriteLine("Checking Status.");
}

```

如果需要調整觸發方法的間隔，可以呼叫 Timer 物件的 Change 方法；而 Dispose 方法則是用來釋放 Timer 物件執行時所參考的外部資源。



## 多執行緒中的共用資源

多執行緒的執行環境中，執行緒輪流使用處理器的資源做運算，當然也輪流休息。如果在 A 執行緒休息的期間，有其他的執行緒修改了 A 執行緒所使用的物件狀態；等 A 執行緒下次執行時，就會產生錯誤的結果。因此在多執行緒中，我們需要使用一些機制，來保護多執行緒中共用資源的狀態，這就稱之為同步化(Synchronization)。程式中有許多同步化保護共用資源的方法如下：

- Synchronization Lock
- Monitor 類別
- Mutex 類別
- Semaphore 類別
- ReaderWriterLock 類別
- Interlocked 類別

每一種保護方式都有不一樣的效果，接下來就來了解一下這些方法的使用方式。

### Synchronization Lock

確保程式碼區塊執行到完畢，不受其他執行緒打斷

- C# – 使用 lock 程式碼區塊
- VB – 使用 SyncLock 程式碼區塊

在方法宣告時使用 **MethodImpl** 標籤

- 設定 MethodImplOptions.Synchronized
- 鎮定範圍為整個方法

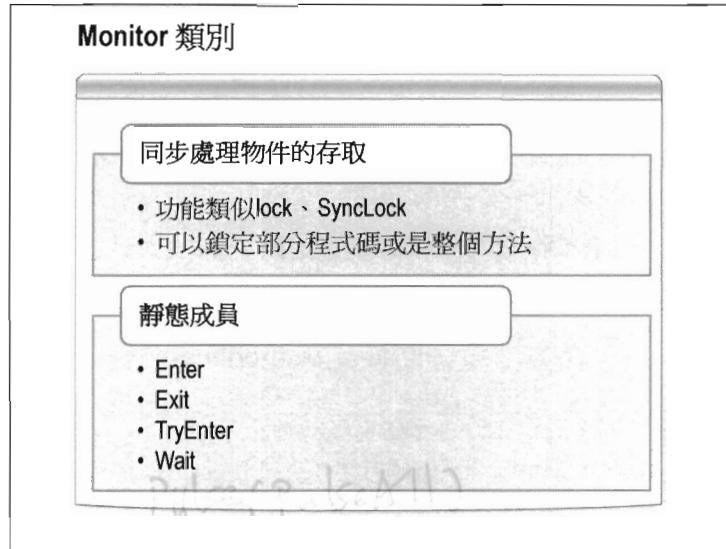
## Synchronization Lock

保護共用資源的最簡單的方式，就是直接鎖定程式，讓程式一次只能有一個執行緒執行，稱之為同步化鎖定(Synchronization Lock)。在 C# 跟 Visual Basic 的程式語言中，都有可以宣告同步化鎖定的語法：在 C# 中使用 lock 程式碼區塊；在 Visual Basic 則是用 SyncLock 程式碼區塊：

```
Visual Basic
SyncLock Me
  ' Do Something
End SyncLock
```

```
C#
lock(this){
  // Do Something
}
```

此外，在宣告方法時使用 **MethodImpl** 的標籤也可以有鎖定方法的效果。跟 SyncLock 或是 lock 與法不一樣的地方是，MethodImpl 標籤鎖定必須要等到整個方法執行完畢之後才會釋放，因此鎖定的時間較長。



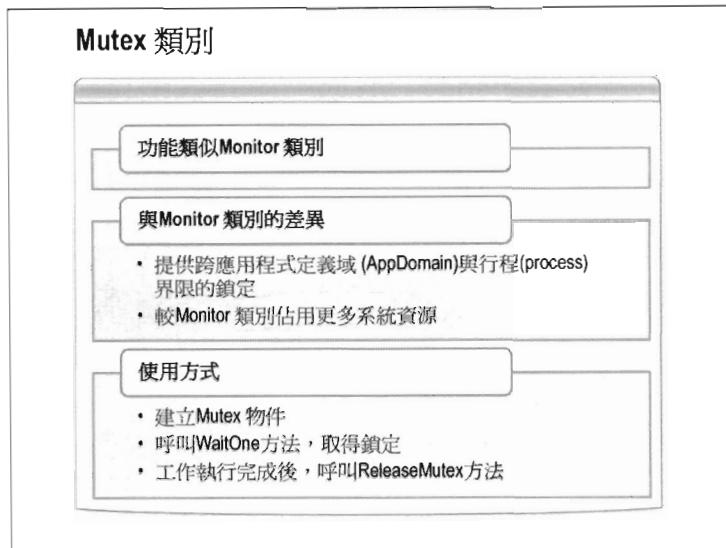
## Monitor 類別

Monitor 類別的用途跟 lock 與 SyncLock 相近，就是用來鎖定程式，讓程式一次只能有一個執行緒使用。Monitor 類別可以用來鎖定部分程式碼區塊或是整個方法：使用 Monitor.Enter 靜態方法開始鎖定；而使用 Monitor.Exit 方法則是釋放鎖定。一般使用的 pattern 如下：

```
Visual Basic :
Monitor.Enter(Me)
Try
    ' Do something....
Finally
    Monitor.Exit(Me)
End Try
```

```
C# :
Monitor.Enter(this);
try{
    // Do something....
}finally{
    Monitor.Exit(this);
}
```

另外，也可以使用 TryEnter 靜態方法指定取得鎖定的等待時間；而 Wait 靜態方法則是會釋出物件的鎖定，以容許其他執行緒鎖定和存取物件，而目前執行緒在另一執行緒存取物件的期間等候。

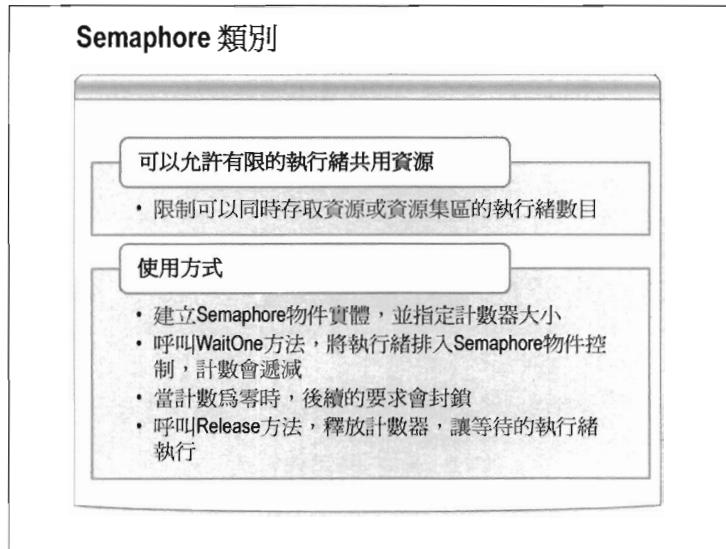


## Mutex 類別

另一個跟 Monitor 類別功能相似的就是 Mutex 類別，但不一樣的地方在於 Mutex 物件主要用於進行提供跨應用程式定義域 (AppDomain)與行程(process)界限的鎖定，因此較 Monitor 物件使用時佔用更多系統資源。Mutex 物件的使用樣板如下：

```
Visual Basic
Dim m As Mutex = New Mutex()
If m.WaitOne(1000,False) Then
    Try
        ' Do something.....
    Finally
        m.ReleaseMutex()
    End Try
End If
```

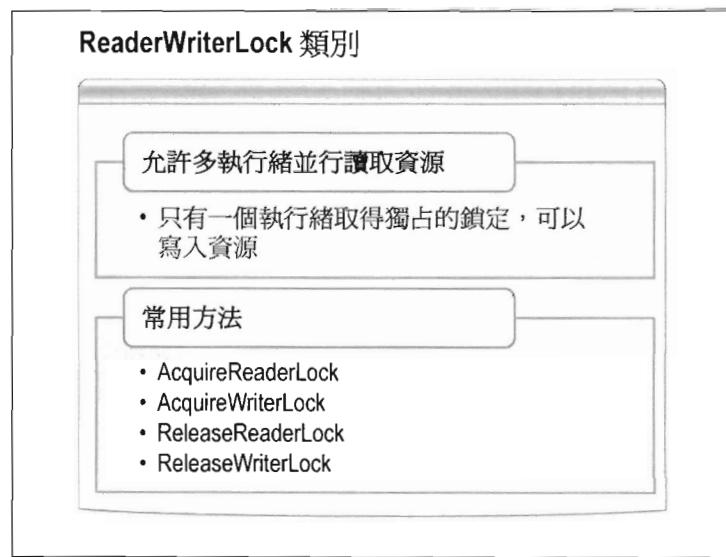
```
C#
Mutex m = new Mutex();
if(m.WaitOne(1000,false)){
    try{
        // Do something.....
    }finally{
        m.ReleaseMutex();
    }
}
```



## Semaphore 類別

前面介紹的一般鎖定的機制，都是鎖定一次只有一個執行緒可以存取資源。若是想要控制允許數個執行緒可以同時存取資源，就必須要使用 Semaphore 物件。Semaphore 物件本身就是一個計數器，可以用來控制同時執行執行緒的數目，在建立時必須要設定最大計數值。

程式中使用 Semaphore 物件的 WaitOne 指令，將執行緒加入 Semaphore 物件控制。每多一個新的執行緒加入控制，Semaphore 計數器的值就會減一；一旦達到零，則後續的執行緒必須要等到 Semaphore 物件呼叫 Release 方法釋放計數器之後，才可以加入排程存取指定資源。

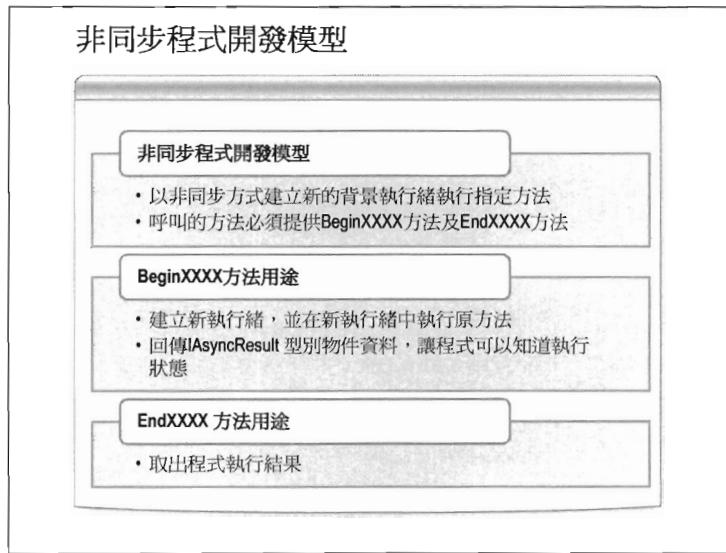


## ReaderWriterLock 類別

還有一種比較特殊的情況，就是程式希望將資源的鎖定減到最小：允許多執行緒並行讀取資源，但是一次只有一個執行緒取得獨佔的鎖定可以寫入資源，此時就要使用 **ReaderWriterLock** 物件。

**ReaderWriterLock** 物件提供了下面的方法，供程式取得/釋放 Reader Lock 或是 Writer Lock：

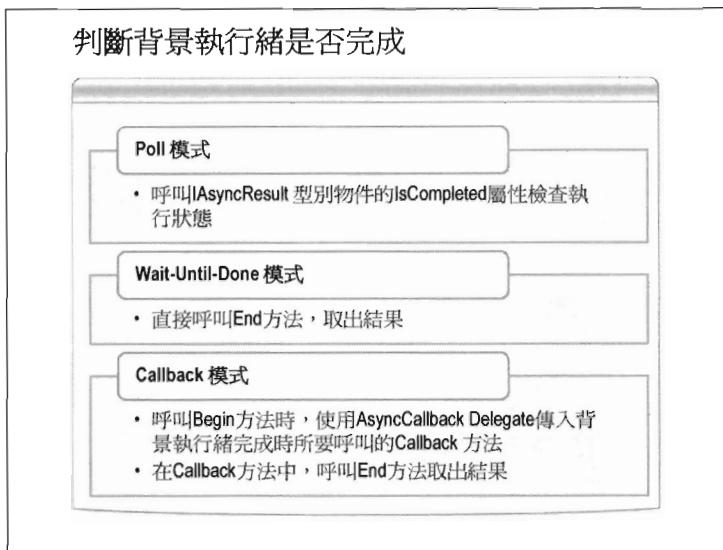
- AcquireReaderLock 方法 / ReleaseReaderLock 方法：  
• 取得/釋放程式的 ReaderLock。
- AcquireWriterLock 方法 / ReleaseWriterLock 方法：  
• 取得/釋放程式的 WriterLock。



## 非同步程式開發模型 (Asynchronous Programming Model)

在.NET Framework 中，針對非同步執行的應用程式，定義了一種特別的非同步程式開發模型(Asynchronous Programming Model)。若是你所呼叫的方法支援這種開發模型，則除了方法本身之外，還會另外再提供 BeginXXXX 方法及 EndXXXX 方法，讓開發人員可以不需要自己寫程式，就可以建立新執行緒執行。像是在 System.IO 、 System.Net 等命名空間中，負責存取外部資源的 FileStream 、 NetworkStream 等物件，就都有支援這種架構。

在非同步程式開發模型中，呼叫 BeginXXXX 方法會建立新的背景執行緒，執行原本方法的內容；而 EndXXXX 方法則是用來取出執行結果的。



## 判斷背景執行緒是否完成

在非同步程式開發模型中，我們一般可以使用下面幾種方式判斷背景執行緒是否完成：

1. Poll 模式：

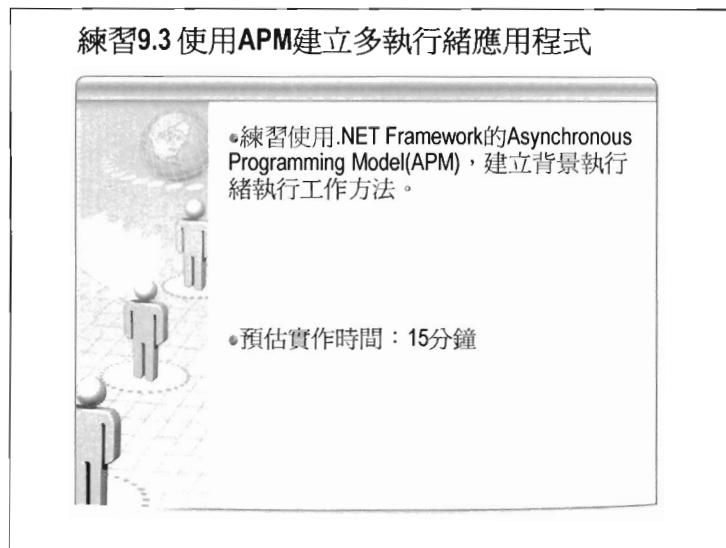
BeginXXX 方法執行時，會回傳 IAsyncResult 型別的物件，你可以透過迴圈不斷檢查 IAsyncResult 型別物件的 IsComplete 屬性是否為 true。

2. Wait-Until-Done 模式：

除此之外，也可以直接在呼叫完 BeginXXX 方法之後，直接呼叫 EndXXX 方法。此時執行環境會暫停目前執行工作，等背景工作完成之後再回傳結果

3. Callback 模式：

在呼叫 Begin 方法時，使用 AsyncCallback Delegate 傳入背景執行緒完成時所要呼叫的 Callback 方法。然後在 Callback 方法中，呼叫 End 方法取出結果。



## 練習 9.3：使用 APM 建立多執行緒應用程式

目的：

練習使用.NET Framework 的 Asynchronous Programming Model(APM)，建立背景執行緒執行工作方法。

功能描述：

透過自訂 Delegate 物件的 BeginInvoke 方法，可以將工作方法執行在背景執行緒中，再透過 Callback 方法的程式呼叫 EndInvoke 方法，就可以取出執行結果。

預估實作時間：15 分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Console Application 專案，將專案命名為 Mod09\_3。

3. 在 Module1.vb (C#為 Program.cs) 類別檔最前面引用  
**System.Threading** 命名空間。

```
Visual Basic
Imports System.Threading
```

```
C#
using System.Threading;
```

4. 在 Module1 模組(C#為 Program 類別)中，宣告計算整數和的方法 **Counter**，程式如下：

```
Visual Basic
Function Counter(ByVal startno As Integer, ByVal stopno As Integer) As Integer
    Dim i, data As Integer
    For i = startno To stopno
        data += i
        Thread.Sleep(50)
    Next i
    Return data
End Function
```

```
C#
static int Counter(int startno, int stopno) {
    int data = 0;
    for (; startno <= stopno; startno++) {
        data += startno;
        Thread.Sleep(50);
    }
    return data;
}
```

5. 接著根據 Counter 方法的參數列與回傳值型別，宣告名為 **MyDelegate** 的委派，程式如下：

```
Visual Basic
Delegate Function MyDelegate(ByVal startno As Integer, _
    ByVal stopno As Integer) As Integer
```

```
C#
delegate int MyDelegate(int startno, int stopno);
```

6. 建立一個 MyDelegate 委派的實體參考到 Counter 方法，並存放在 worker 的靜態變數中。

Visual Basic

```
Dim worker As New MyDelegate(AddressOf Counter)
```

C#

```
static MyDelegate worker = new MyDelegate(Counter);
```

7. 接下來建立 Callback 的靜態函式，取名為 **CallbackMethod**。

在 CallbackMethod 方法中，呼叫 **worker.EndInvoke** 取出背景執行緒工作結果，並透過 Console 物件輸出：

Visual Basic

```
Sub CallbackMethod(ByVal ar As IAsyncResult)
    Dim result As Integer = worker.EndInvoke(ar)
    Console.WriteLine("Result = " + result)
End Sub
```

C#

```
static void CallbackMethod(IAsyncResult ar) {
    int result = worker.EndInvoke(ar);
    Console.WriteLine("Result = " + result);
}
```

8. 在 Main 方法之中建立 **AsyncCallback** 委派實體，參考到 CallbackMethod 方法。接著呼叫 **worker** 靜態變數的 **BeginInvoke** 方法，在參數列中分別傳入起始數字、結束數字、 AsyncCallback 委派及 null，啓動背景執行緒執行工作方法：

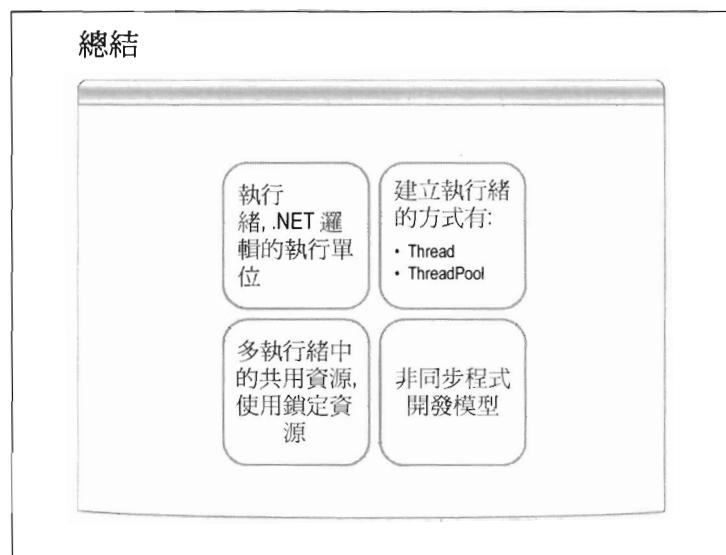
Visual Basic

```
'add code here
Dim asynccb As New AsyncCallback(AddressOf CallbackMethod)
worker.BeginInvoke(1, 100, asynccb, Nothing)
Console.Read ()
```

C#

```
//add code here.
AsyncCallback asynccb = new AsyncCallback(CallbackMethod);
worker.BeginInvoke(1,100,asynccb, null);
Console.Read ();
```

9. 按下「F5」鍵，執行專案並檢視執行結果。



---

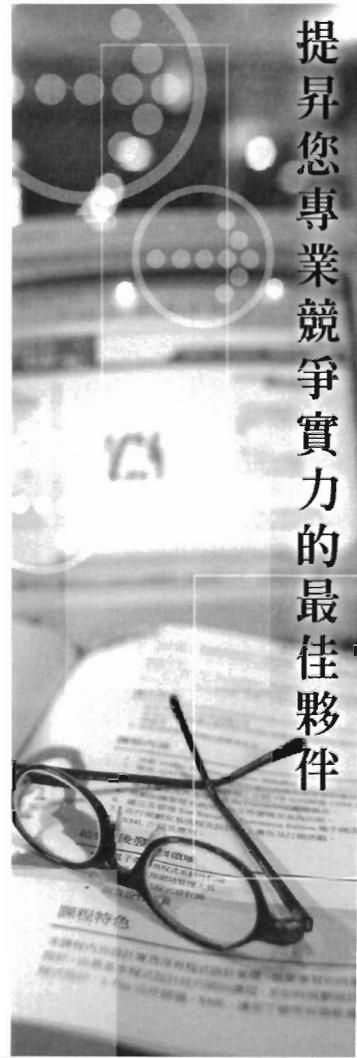
# 第十章：Windows 服務 與 AppDomain

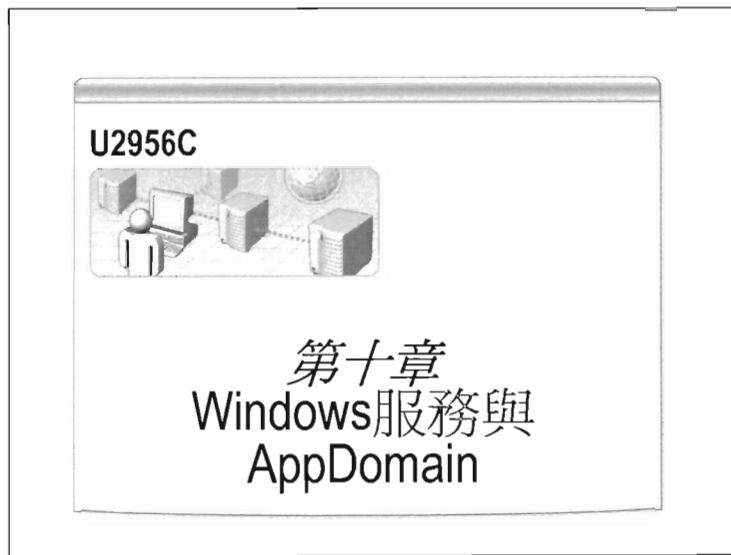
## 本章大綱

認識 Windows 服務(Service).....	4
服務管理員.....	6
建立 Windows Service.....	8
認識狀態與方法之間的關聯.....	10
練習 10.1：建立 Windows 服務.....	11
認識 Installer 類別.....	15
服務的安全性內容.....	16
安裝 Windows 服務.....	18
使用安裝專案.....	19
練習 10.2：建立服務的安裝類別.....	22
管理及控制服務.....	26
使用 ServiceController 類別.....	27
練習 10.3：管理及控制服務.....	29
認識應用程式定義域(Application Domain) .....	35
認識 AppDomain 類別.....	37
如何建立應用程式定義域.....	38
如何取得目前的應用程式定義域.....	40
練習 10.4：載入組件到應用程式定義域.....	41
總結.....	43

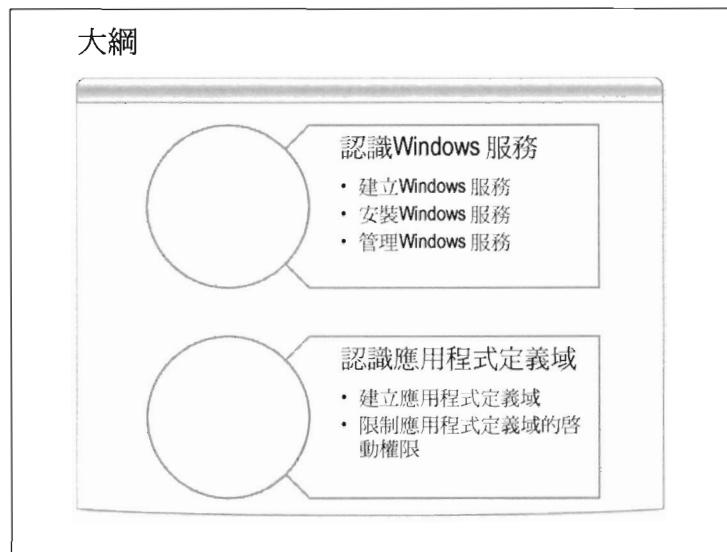
作者：

羅慧真





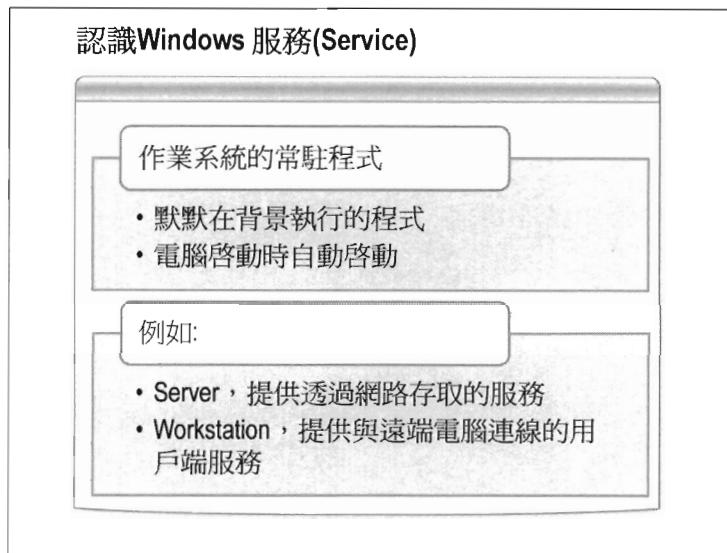
---



在這個章節中你將認識什麼是 Windows 服務，以及如何利用 Visual Studio 開發工具建立 Windows 服務及管理 Windows 服務的應用程式。同時這個章節也介紹什麼是應用程式定義域，如何建立應用程式定義域及權限設定。

本章介紹以下主題：

- 認識 Windows 服務
  - 建立 Windows 服務
  - 安裝 Windows 服務
  - 管理 Windows 服務
- 認識應用程式定義域
  - 建立應用程式定義域
  - 限制應用程式定義域的啓動權限

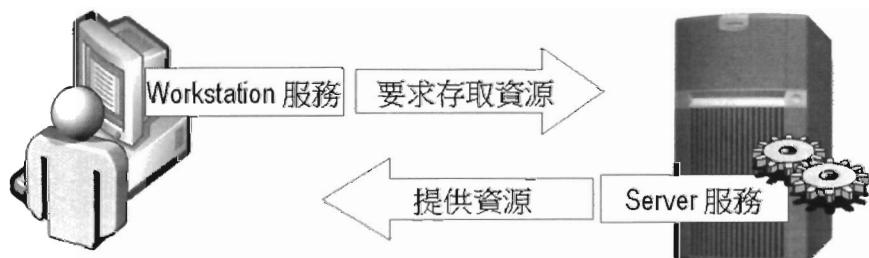


## 認識 Windows 服務 (Service)

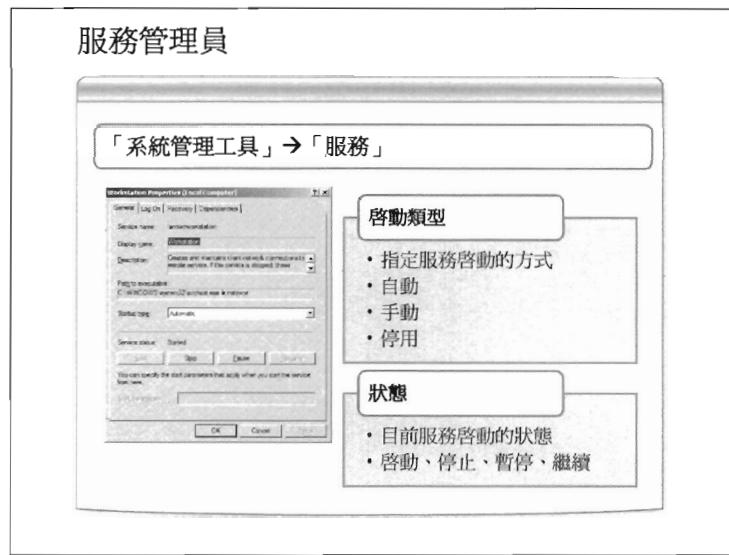
什麼是 Windows 服務 (Service)？它是作業系統的常駐程式，在系統開機之後，不需經由使用者登入及啓動，服務就會自動啓動並且在背景默默的執行服務程式。

有哪些程式是屬於 Windows 服務呢？其實相當的多，例如，大家都有透過網路存取別人電腦中資料的經驗，被存取的電腦如何提供資源給其他電腦使用呢？又電腦如何能夠提供要求存取資源的能力呢？

那是因為每台電腦都有一個叫做 **Server** 的服務，在電腦開機時就啓動並等候網路使用者的存取。相對的，每台電腦能夠連到別人電腦並存取其資源，也是因為每台電腦中有一個叫做 **Workstation** 的服務，才能夠存取遠端電腦的資源。如下圖：

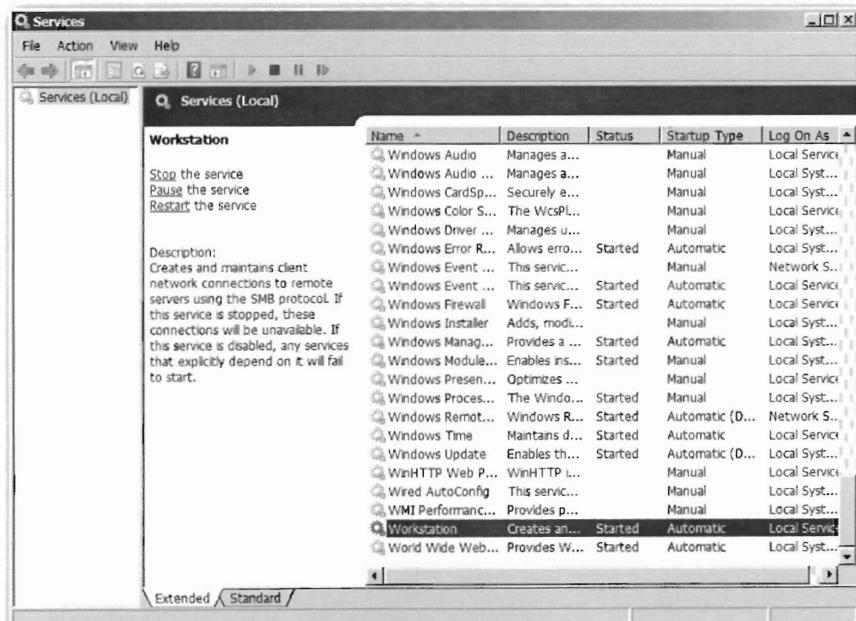


這張圖是為了方便表示這二個服務的功能，所以各自只顯示一種服務。實際上電腦會啓動 **Server** 與 **Workstation** 的服務，因為大部份的情況是電腦可能會提供資源給其他電腦使用，同時也可能去存取其他電腦的資源。



## 服務管理員

服務是否自動啓動，啓動時使用哪個帳號登入系統，這些功能的管理是在「服務管理員」(SCM : Service Control Manager)，它的 MMC 介面從「開始」(Start)→「系統管理工具」(Administrative Tools)→「服務」(Service)進入「服務管理員」介面工具。

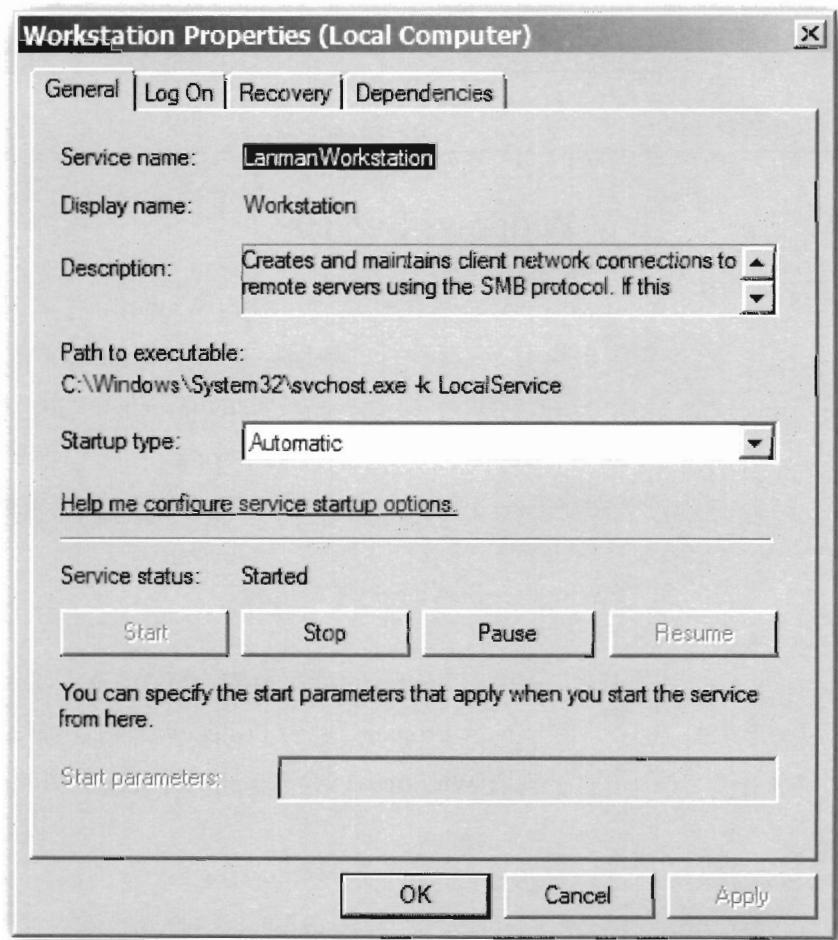


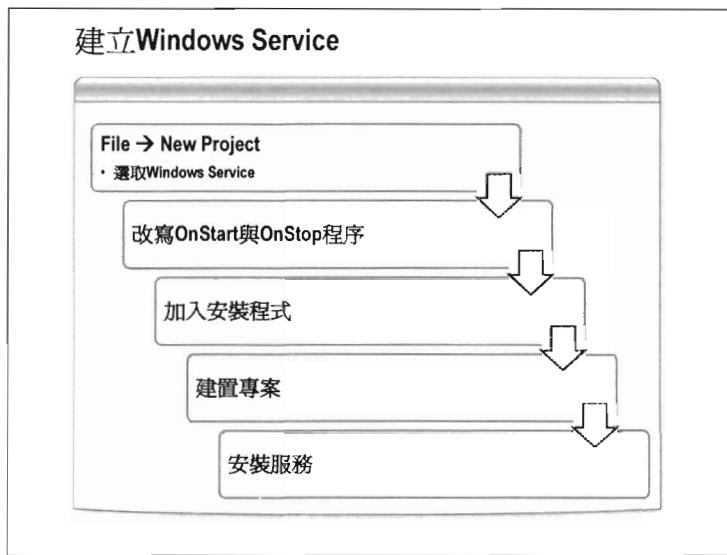
啓動類型(Startup Type)是指服務的啓動方式：

- 「自動」(Automatic)，服務在開機後自動啓動。
- 「手動」(Manual)，服務必須經由人為或程式啓動才會執行。
- 「停用」(Disabled)，服務不可啓動。

狀態(Status)是指服務目前是否執行中，包含以下幾種狀態：

- 「啟動」(Start)，代表服務正在執行中。
- 「停止」(Stop)，代表服務尚未執行。
- 「暫停」(Pause)，代表將執行中的服務進入中止狀態。
- 「繼續」(Resume)，代表將中止狀態的服務轉為繼續執行。



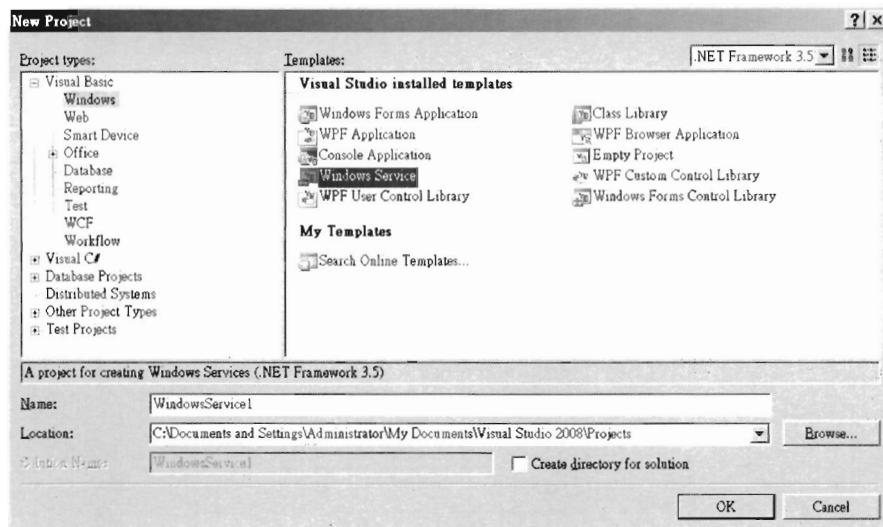


## 建立 Windows Service

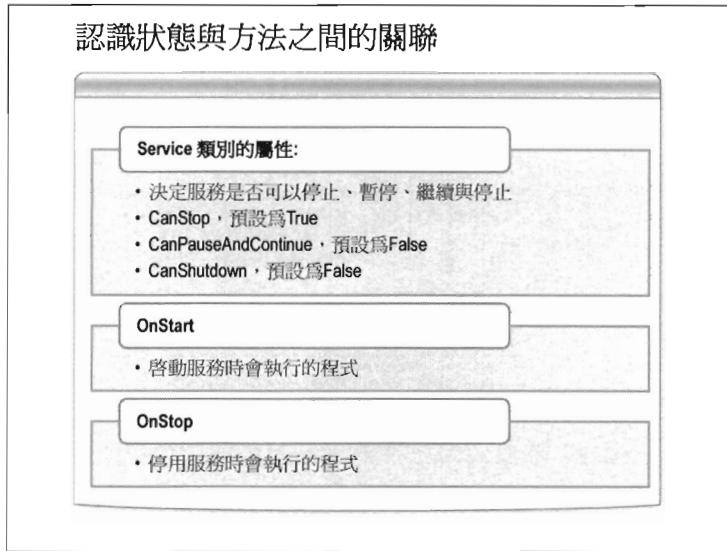
Windows Service 專案有別於一般的 Windows 專案，Windows 應用程式是一種可以直接執行的應用程式，由 Windows 專案所建置的.exe 程式可以直接執行，或者可在 Visual Studio 的環境中按下 F5 執行。Windows Service 專案也是建置為.exe 檔，但此程式並不能直接執行，而是必須先安裝然後再經由「服務管理員」啟動。

建立 Windows Service 的專案步驟如下：

1. 在 Visual Studio 的選單選取「File」→「New Project」。接著在「New Project」視窗 Project types：展開 Visual Basic 或 C# 後選取 Windows，在 Templates：選取 Windows Service。



2. 改寫 Service 類別的 OnStart 與 OnStop 程序。
3. 加入安裝程式，並設定安裝屬性。
4. 建置專案。
5. 安裝服務。
6. 從「服務管理員」啟動服務。



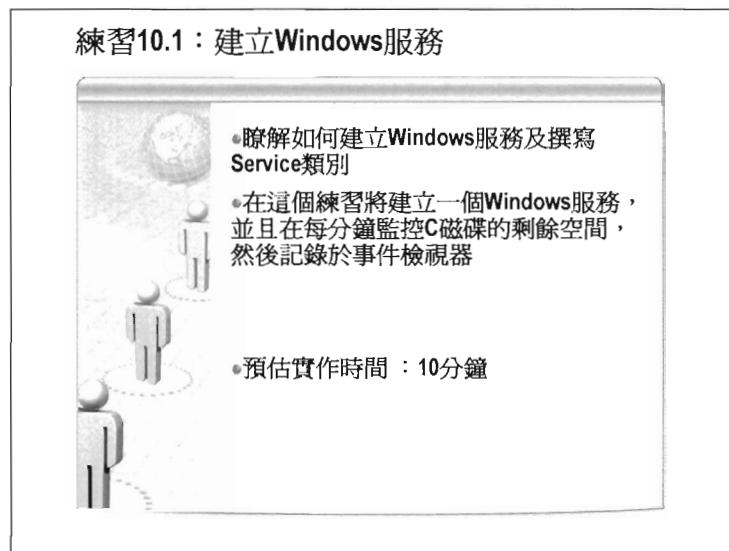
## 認識狀態與方法之間的關聯

建立專案之後 Solution Explorer 會自動加入一個 Service1.vb 或 Service1.cs 檔，此檔案是繼承自 System.ServiceProcess.Service 類別，提供以下幾個屬性：

- CanStop，是否允許停止服務，預設為 True。
- CanPauseAndContinue，是否允許暫停及繼續服務，預設為 False。
- CanShutdown，服務是否接收系統關機的通知，預設為 False。
- AutoLog，是否在自動寫入服務的記事項目到事件記錄檔，預設為 True。

以及以下幾個方法：

- OnStart，啓動服務時會執行的程式。
- OnStop，停用服務時會執行的程式。



## 練習 10.1：建立 Windows 服務

目的：

瞭解如何建立 Windows 服務以及撰寫 Service 類別。

功能描述：

在這個練習將建立一個 Windows 服務，並且在每分鐘監控 C 磁碟的剩餘空間，然後記錄於事件檢視器。

預估實作時間：20 分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 WatchHDFreeSpace。
3. 建立 Windows Service 專案之後，範本會自動建立 Service 類別，開啟 Service1 後會看到：

```
Visual Basic
Public Class Service1
    Protected Overrides Sub OnStart(ByVal args() As String)
        End Sub

    Protected Overrides Sub OnStop()
        End Sub
    End Class
```

```
C#
public partial class Service1 : ServiceBase
{
    protected override void OnStart(string[] args)
    {
    }

    protected override void OnStop()
    {
    }
}
```

4. 匯入命名空間 System.IO。
5. 在類別中加入類別層級的變數，型別為 System.Timers.Timer。

```
Visual Basic
Dim WithEvents serviceTimer As System.Timers.Timer
```

```
C#
System.Timers.Timer serviceTimer;
```

6. 在類別建構函式中初始化 System.Timers.Timer 物件，並指定計時器的計時頻率。

```
Visual Basic
Public Sub New()
    InitializeComponent()

    serviceTimer = New System.Timers.Timer
    serviceTimer.Interval = 60000
End Sub
```

```
C#
public Service1()
{
    InitializeComponent();
    serviceTimer = new System.Timers.Timer();
    serviceTimer.Elapsed += new
        System.Timers.ElapsedEventHandler(serviceTimer_Elapsed);
    serviceTimer.Interval = 60000;
}
```

7. 在 OnStart 及 OnStop 的區段中，分別啓動及停止計時器。

```
Visual Basic
Protected Overrides Sub OnStart(ByVal args() As String)
    serviceTimer.Start()
End Sub

Protected Overrides Sub OnStop()
    serviceTimer.Stop()
End Sub
```

```
C#
protected override void OnStart(string[] args)
{
    serviceTimer.Start();
}

protected override void OnStop()
{
    serviceTimer.Stop();
}
```

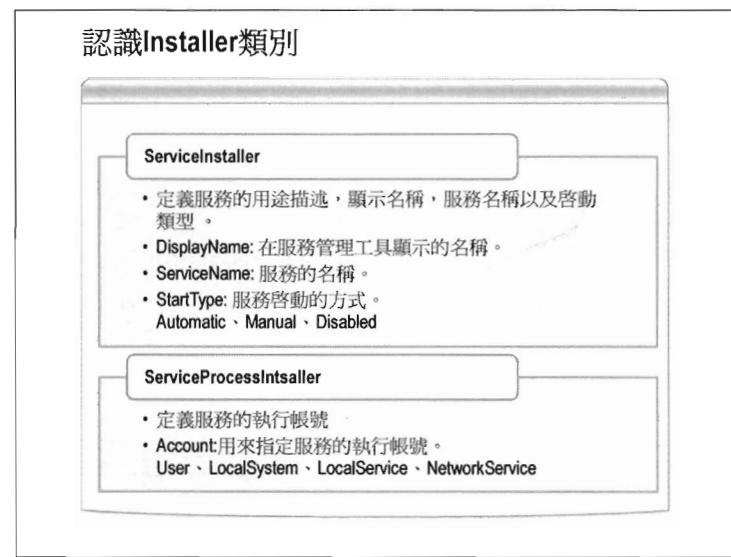
8. 在 Timer 的 Elapsed 事件中，撰寫服務要執行的工作。

記錄 C 磁碟的剩餘空間，使用 DriveInfo 的物件取得剩餘空間資訊，並透過 EventLog 的 WriteEntry 將剩餘空間寫到事件記錄檔中。

```
Visual Basic
Private Sub serviceTimer_Elapsed(ByVal sender As Object, ByVal e
As System.Timers.ElapsedEventArgs) Handles serviceTimer.Elaps
ed
    Dim cDrive As New DriveInfo("C:\")
    EventLog.WriteEntry("目前C磁碟剩餘空間:" & _
        cDrive.TotalFreeSpace)
End Sub
```

```
C#
void serviceTimer_Elapsed(object sender, System.Timers.ElapsedE
ventArgs e)
{
    DriveInfo cDrive = new DriveInfo(@"C:\");
    EventLog.WriteEntry("目前C磁碟剩餘空間：" +
                        cDrive.TotalFreeSpace);
}
```

9. 編譯此專案確認程式無誤即可。



## 認識 Installer 類別

Windows 服務(Service)有別於其他類型的應用程式，並不能直接執行，而且也不能直接從 Visual Studio 執行及偵錯，而是必須經過事先安裝的程序。.NET Framework 提供 ServiceInstaller 及 ServiceProcessInstaller 類別進行服務的安裝。

- ServiceInstaller 用來安裝專案中的服務程式(繼承自 ServiceBase 的服務)及定義服務的用途描述，顯示名稱，服務名稱以及啟動類型。它的屬性如下：
  - Description，描述服務的用途。
  - DisplayName，在服務管理員顯示的名稱。
  - ServiceName，服務的名稱。
  - StartType，服務的啟動方式，選項有 Automatic、Manual 與 Disabled。
- ServiceProcessInstaller 用來安裝專案中的服務程式(繼承自 ServiceBase 的服務)及定義服務的執行帳號。它的屬性如下：
  - Account，指定服務的執行帳號。選項有 User、LocalSystem、LocalService、NetworkService。如果指為 User，必須指定 Username 與 Password。
  - Username，指定服務的執行帳號。
  - Password，指定帳號的密碼。



## 服務的安全性內容

前面說明過 Windows 服務(Service)並不一定需要使用者登入系統才會啓動，多數的情況下服務是在開機之後便自動啓動，所以它的執行帳號並不是目前登入的使用者帳號，在安裝或從服務管理員設定服務屬性時可以指定服務執行的帳號。

下圖是由服務管理員的 MMC 畫面選取 Workstation 服務的帳號設定畫面，Workstation 服務的執行帳號是 Local System account。

管理者或開發人員可以依據服務所需的權限決定服務要使用的帳號，帳號類型包含以下四種：

- User，服務使用一般的使用者帳號，如果 ServiceProcessInstaller 物件沒有指定 Username 及 Password，在服務進行安裝時，安裝程式會跳出訊息視窗詢問服務的帳號或密碼。
- LocalService，本機電腦上未授權的使用者，若需存取遠端伺服器時，提供匿名身份進行驗證。
- LocalSystem，服務控制管理員所使用的帳號，在本機電腦上具有較高的權限，並允許存取網路上其他的電腦。

- NetworkService，提供較高的本機權限，並以此身份提供遠端伺服器進行驗證。



## 安裝 Windows 服務

安裝 Windows 服務的方式有二：

- 使用 InstallUtil.exe
- 使用 Windows Installer

### 使用 InstallUtil.exe

使用 InstallUtil.exe 進行服務安裝是一種便利而簡單的方式，通常是開發者在開發環境中進行測試或偵錯時會使用。

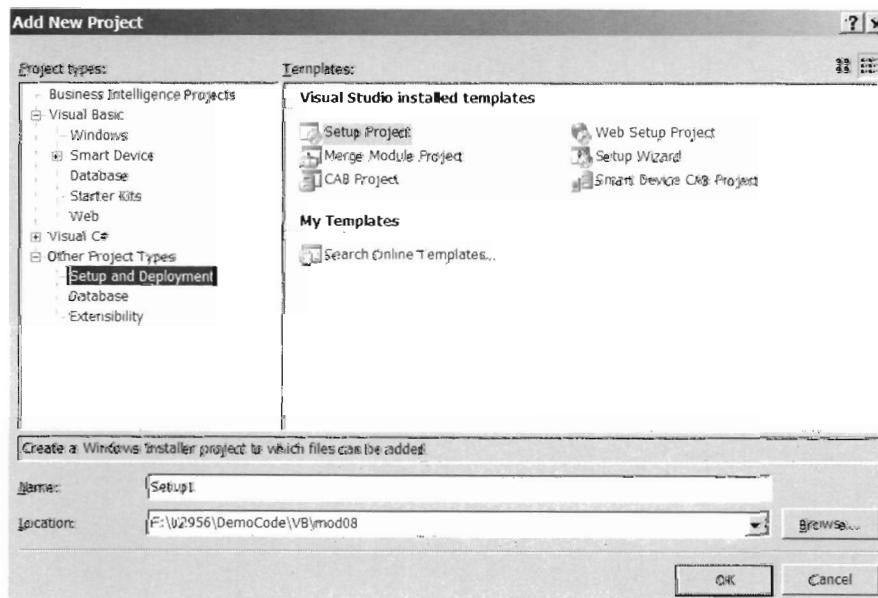
InstallUtil.exe 安裝可以用來安裝及移除.NET 程式的工具，要使用 InstallUtil.exe 安裝的程式必須配合 System.Configuration.Install 命名空間中的類別一起使用。



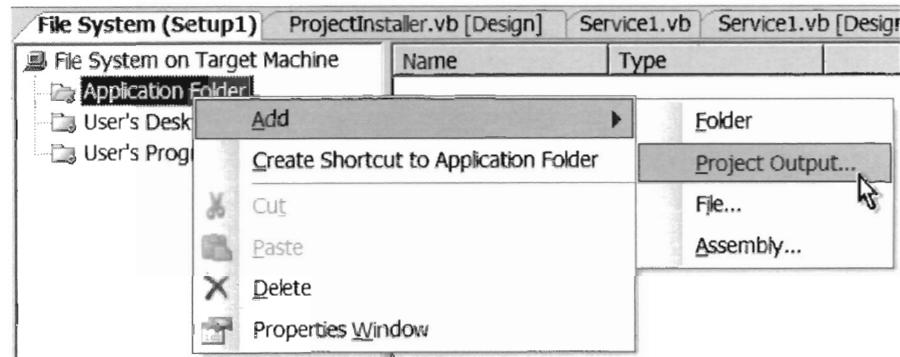
## 使用安裝專案

當服務應用程式完成設計之後，準備要部署到使用者的電腦中時，通常使用安裝程式會比下命令來的方便。它的步驟如下：

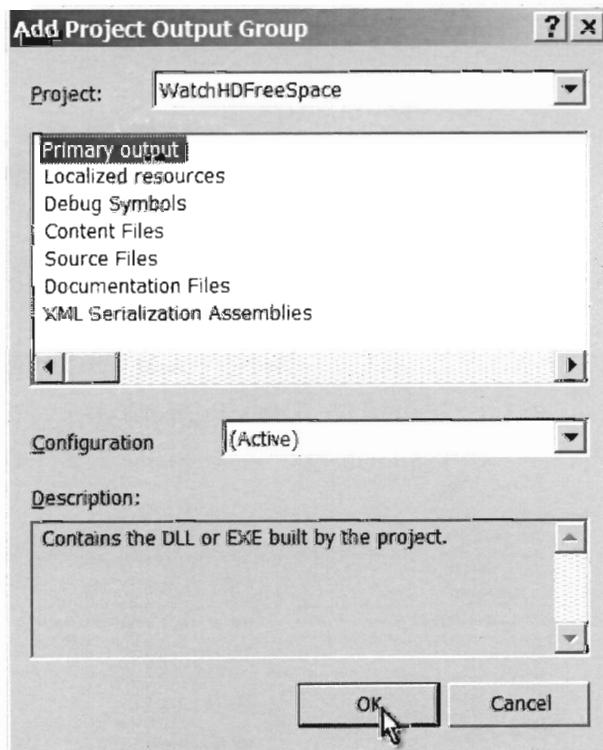
1. 在開啓服務專案的狀態下，選取「File」→「Add」→「New Project」，然後在「Add New Project」視窗下，Project types項目選取「Other Project Types」→「Setup and Deployment」，Templates選取「Setup Project」。



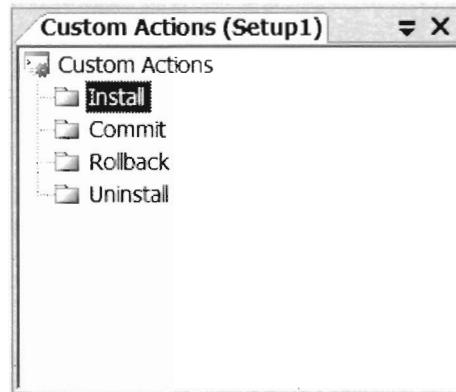
2. 開啓 Setup 專案之後，在 File System 視窗中選取「Application Folder」按滑鼠右鍵選取「Add」→「Project Output...」。



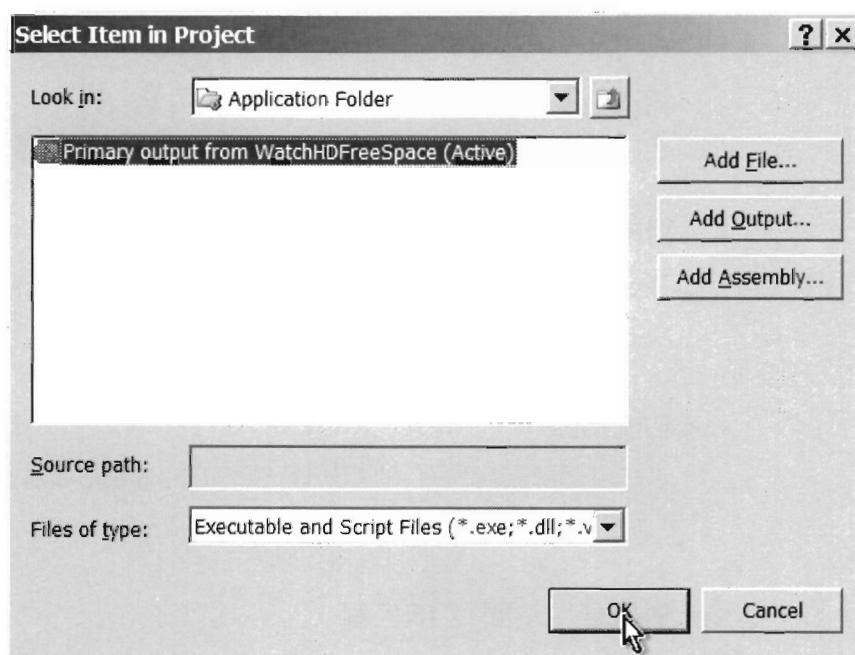
3. 然後在「Add Project Output Group」視窗選取「Primary output」，並按下「OK」。



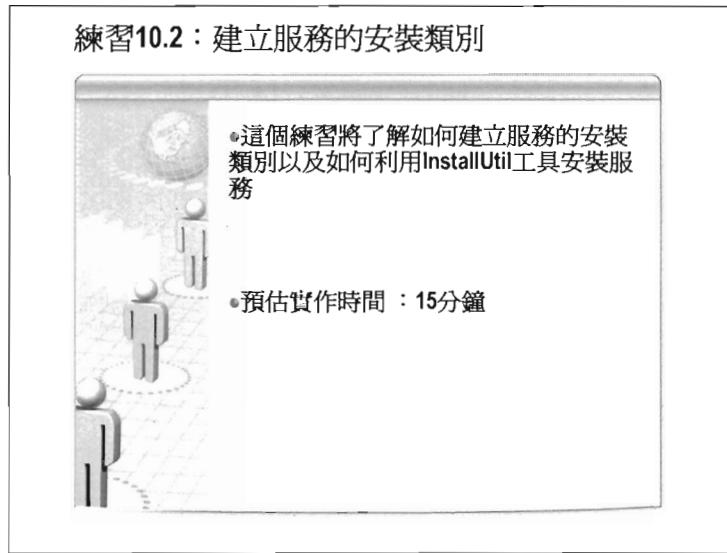
4. 在「Solution Explorer」選取「Setup1」按滑鼠右鍵選取「View」→「Custom Action」，進入 Custom Action 視窗。



5. 分別在 Install、Commit、Rollback、Uninstall 上按滑鼠右鍵選取「Add Custom Action」，在 Select Item in Project 項目上雙擊「Application Folder」，進入如圖畫面，然後點選 Primary output from WatchHDFreeSpace(Active)，並按下「OK」。



建置 Setup1 專案，建置完成之後進行安裝即可。



## 練習 10.2：建立服務的安裝類別

**目的：**

這個練習將了解如何建立服務的安裝類別以及如何利用 InstallUtil 工具安裝服務。

**功能描述：**

在這個練習延續前一個練習，為前一個練習所完成的服務建立安裝類別，並且使用 InstallUtil 工具安裝服務。

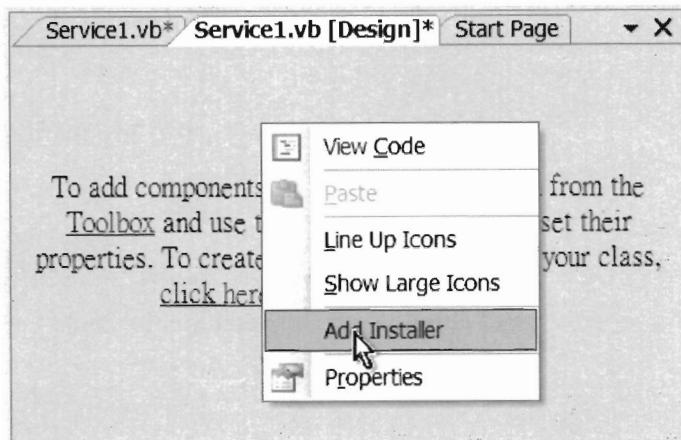
**預估實作時間：15分鐘**

**實作步驟：**

並不需要特別撰寫 ServiceInstaller 及 ServiceProcessInstaller 類別的程式，因為 Visual Studio 會自動產生程式碼。建立的步驟如下：

1. 開啓前一個練習專案，或從「Practices\VB 或 CS\Mod10\_2\Starter\WatchHDFreeSpace」開啓 WatchHDFreeSpace.sln 檔案。
2. 開啓 Service1 類別的設計視窗，修改 ServiceName 屬性為「WatchHDService」。

3. 然後在設計視窗上按滑鼠右鍵選取「Add Installer」，如下圖：



接著專案會新增一個 ProjectInstaller 類別。在 ProjectInstaller 的設計視窗中自動會新增 ServiceInstaller1 與 ServiceProcessInstaller1 二個項目，如下圖：



4. 設定 ServiceInstaller1 的屬性如下：

屬性	設定值
Description	監控 C 磁碟的剩餘空間
DisplayName	監控磁碟空間
ServiceName	WatchHDService
StartType	Automatic

5. 設定 ServiceProcessInstaller1 的屬性如下：

屬性	設定值

Account	LocalSystem
---------	-------------

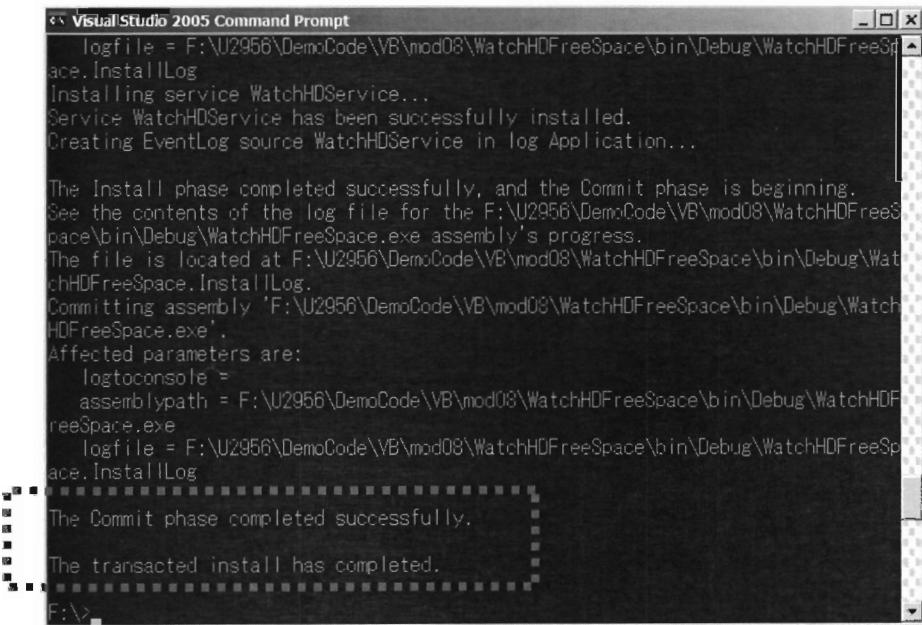
6. 建置專案。

完成安裝類別之後，就可以使用 InstallUtil 進行服務的安裝，步驟如下：

7. 從開始工作列點取程式集，然後選取「Microsoft Visual Studio」→「Visual Studio Tools」→「.NET Framework Command Prompt」
8. 在命令提示列上輸入：(請移到目前程式所在位置)

InstallUtil WatchHDFreeSpace.exe
----------------------------------

9. 按下「Enter」建之後，便開始進行安裝程序。安裝完成之後你會在命令提示列視窗中看到如下訊息便代表安裝完成：



```

C:\> Visual Studio 2005 Command Prompt
logfile = F:\U2956\DemoCode\VB\mod08\WatchHDFreeSpace\bin\Debug\WatchHDFreeSpace.Install\Log
Installing service WatchHDFreeSpace...
Service WatchHDFreeSpace has been successfully installed.
Creating EventLog source WatchHDFreeSpace in log Application...

The Install phase completed successfully, and the Commit phase is beginning.
See the contents of the log file for the F:\U2956\DemoCode\VB\mod08\WatchHDFreeSpace\bin\Debug\WatchHDFreeSpace.exe assembly's progress.
The file is located at F:\U2956\DemoCode\VB\mod08\WatchHDFreeSpace\bin\Debug\WatchHDFreeSpace.Install\Log.
Committing assembly 'F:\U2956\DemoCode\VB\mod08\WatchHDFreeSpace\bin\Debug\WatchHDFreeSpace.exe'.
Affected parameters are:
logtoconsole =
assemblypath = F:\U2956\DemoCode\VB\mod08\WatchHDFreeSpace\bin\Debug\WatchHDFreeSpace.exe
logfile = F:\U2956\DemoCode\VB\mod08\WatchHDFreeSpace\bin\Debug\WatchHDFreeSpace.Install\Log

The Commit phase completed successfully.

The transacted install has completed.

F:\>

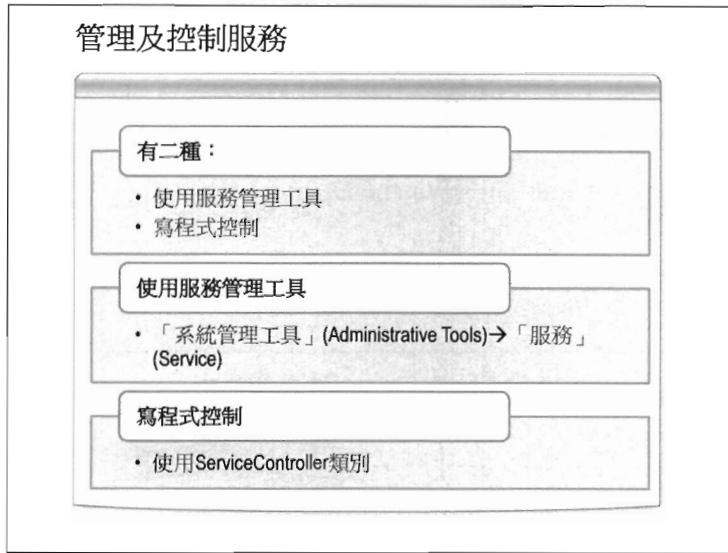
```

10. 安裝完程之後可以開啓「系統管理工具」(Administrative Tools)→「服務」(Service)，在服務的管理視窗中，你看到安裝完成的服務項目。

11. 在「監控磁碟空間」的服務項目上，按滑鼠右鍵選取「啓動」。
12. 啓動完成之後，開啓「系統管理工具」(Administrative Tools)→「事件檢視器」(Event Viewer)，應該會看到來源(Source)為 WatchHDService 的項目。

如果要移除則使用相同的命令，加上”-u”的參數便可以移除服務。

```
InstallUtil WatchHDFreeSpace.exe -u
```

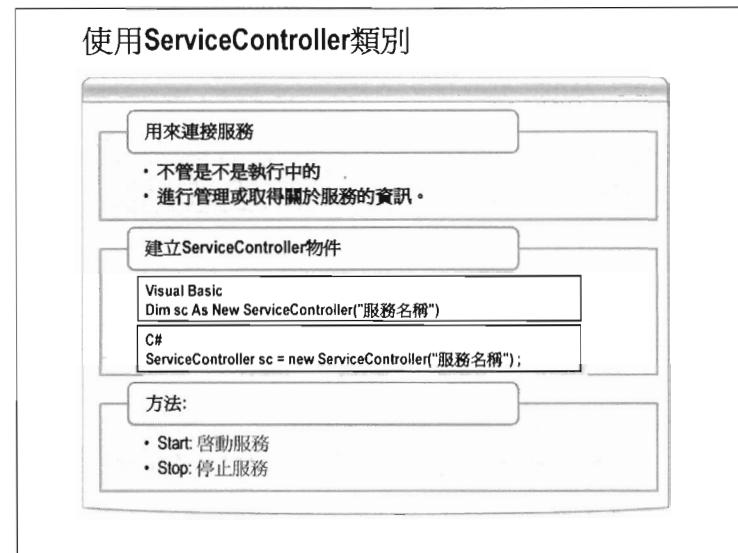


## 管理及控制服務

管理及控制服務的方式有二種，一個是使用服務管理工具，一個是寫程式控制。

### 使用服務管理工具

從「開始」(Start)→「系統管理工具」(Administrative Tools)→「服務」(Service)。在服務管理工具中可以看到服務的清單列表及服務的說明。



## 使用 ServiceController 類別

ServiceController 類別是.NET Framework 提供用來連接到不管是執行中的服務，並進行管理或取得關於服務的資訊。這個類別屬於 System.ServiceProcess 命名空間，開發人員可以利用它所提供的方法進行的啓動或停止服務。它的常用方法說明如下：

- Start，啓動服務。
- Stop，停止服務。同時會停止與此服務相關的其他服務。
- WaitForStatus，等待某個狀態。在呼叫 Start 或 Stop 之後，ServiceController 會開始進行啓動或停止服務的作業，但並不代表啓動或停止作業完成。如果要確認已經完成可以呼叫 WaitForStatus 方法等待完成。

它的常用屬性列表如下：

- DisplayName，服務的顯示名稱。
- Status，服務目前的狀態。

以下範例是啓動服務：

```
Visual Basic
Dim sc As New ServiceController("WatchHDService")
```

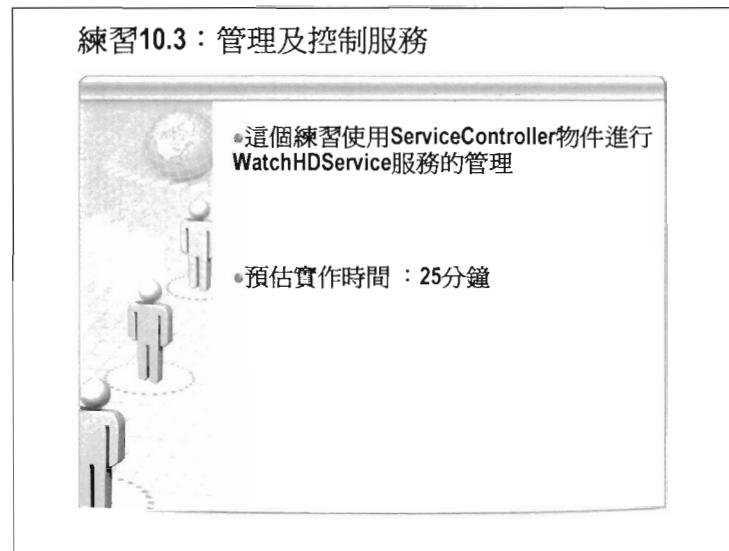
```
sc.Start()
```

```
C#
ServiceController sc = new ServiceController("WatchHDService");
sc.Start();
```

如果服務已啓動又呼叫 Start 方法，程式將會發生 InvalidOperationException 例外狀況，為了避免這個問題，通常會先使用 Status 判斷服務目前的狀態，再來啓動服務。範例如下：

```
Visual Basic
If sc.Status = ServiceControllerStatus.Stopped Then
    sc.Start()
    sc.WaitForStatus(ServiceControllerStatus.Running)
    Label1.Text = sc.DisplayName & " 啓動完成."
End If
```

```
C#
if (sc.Status == ServiceControllerStatus.Stopped)
{
    sc.Start();
    sc.WaitForStatus(ServiceControllerStatus.Running);
    Label1.Text = sc.DisplayName + " 啓動完成.";
}
```



## 練習 10.3：管理及控制服務

目的：

這個練習使用 ServiceController 物件進行 WatchHDService 服務的管理。

功能描述：

這個練習延續前一個練習，將建立一個新的 Windows 專案，做為控制 WatchHDService 服務的管理工具。最後使用 Setup 工具安裝服務及管理工具。

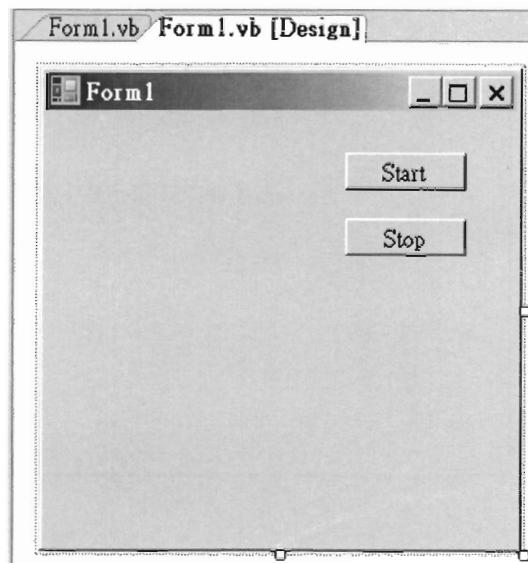
預估實作時間：25 分鐘

實作步驟：

使用 ServiceController 物件管理服務的啓動與停用。

1. 開啓前一個練習專案，或從「Practices\VB 或 CS\Mod010\_3\Starter\WatchHDFreeSpace」開啓 WatchHDFreeSpace.sln 檔案。
2. 選單選取「File」→「Add」→「New Project ...」，建立 Windows Application，取名為「ControlService」。

3. 在 Form1 從「Toolbox」拖拉二個 Button 控制項到表單上，並分別設定其 Text 屬性為 Start 及 Stop。



4. 再從「Toolbox」拖拉 Label 控制項到表單上。  
 5. 加入參考 System.ServiceProcess.dll。  
 6. 進入程式碼視窗，匯入命名空間 System.ServiceProcess：

```
Visual Basic
Imports System.ServiceProcess
```

```
C#
using System.ServiceProcess;
```

7. 在表單類別內建立 ServiceController 類別的物件實體：

```
Visual Basic
Dim sc As New ServiceController("WatchHDService")
```

```
C#
ServiceController sc = new ServiceController("WatchHDService");
```

8. 啓動服務，進入 Button1\_Click 事件程序。

避免重複啓動服務發生 InvalidOperationException 例外狀況，  
使用 Status 判斷服務目前的狀態，才啓動服務：

```
Visual Basic
If sc.Status = ServiceControllerStatus.Stopped Then
    sc.Start()
    sc.WaitForStatus(ServiceControllerStatus.Running)
    Label1.Text = sc.DisplayName & " 啓動完成."
End If
```

```
C#
if (sc.Status == ServiceControllerStatus.Stopped)
{
    sc.Start();
    sc.WaitForStatus(ServiceControllerStatus.Running);
    label1.Text = sc.DisplayName + " 啓動完成.";
}
```

9. 停用服務，進入 Button2\_Click 事件程序。使用 Status 判斷目前服務狀態，若為執行階段呼叫 Stop 方法，停用服務。

```
Visual Basic
If sc.Status = ServiceControllerStatus.Running Then
    sc.Stop()
    sc.WaitForStatus(ServiceControllerStatus.Stopped)
    Label1.Text = sc.DisplayName & " 停止."
End If
```

```
C#
if (sc.Status == ServiceControllerStatus.Running)
{
    sc.Stop();
    sc.WaitForStatus(ServiceControllerStatus.Stopped );
    label1.Text = sc.DisplayName + " 停止.";
}
```

10. 確定目前服務是安裝的狀態。
11. 執行「ControlService」專案。按「Start」按鈕，並使用系統的「服務」確認目前服務的啓動狀態。
12. 也試試「Stop」按鈕，確認功能無誤。

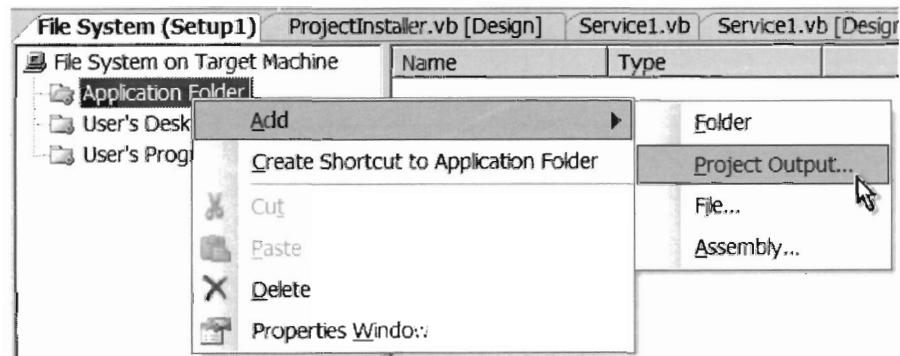
使用安裝專案包裝服務與管理程式。

13. 先移除 WatchHDFreeSpace 服務，使用 InstallUtil 工具。

```
InstallUtil WatchHDFreeSpace.exe -u
```

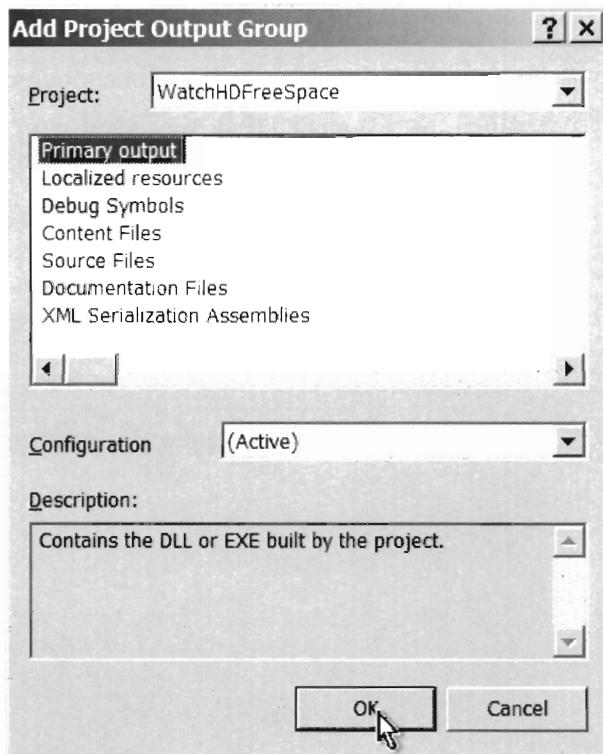
14. 選取「File」→「Add」→「New Project」，然後在「Add New Project」視窗下，Project types 項目選取「Other Project Types」→「Setup and Deployment」，Templates 選取「Setup Project」。

15. 開啟 Setup 專案之後，在 File System 視窗中選取「Application Folder」按滑鼠右鍵選取「Add」→「Project Output...」。



在「Add Project Output Group」視窗，Project 下拉項目選取「ControlService」，清單選取「Primary output」，並按下「OK」。

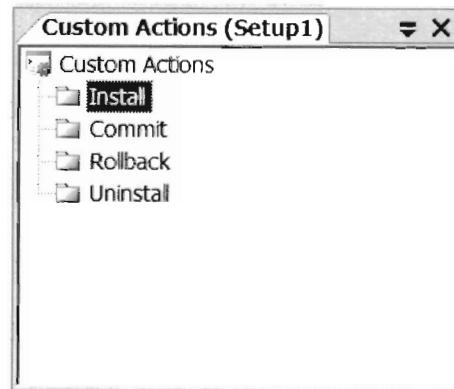
16. 加入專案輸出，選取「WatchHDFreeSpace」專案，及「Primary output」，並按下「OK」。



17. 建立服務管理應用程式的程式捷徑。

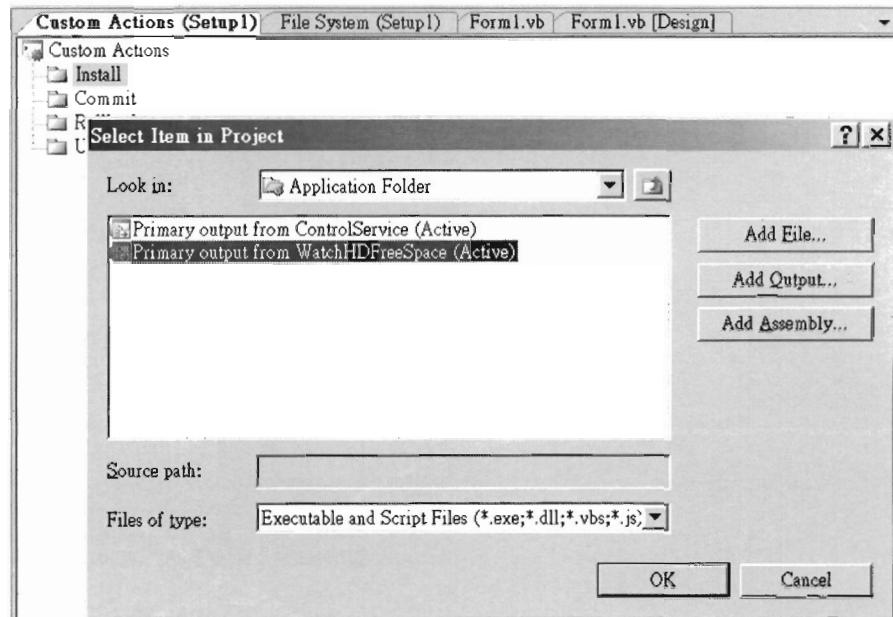
選取「Primary output from ControlService (Active)」按滑鼠右鍵，選取「Create Shortcut to Primary output from ControlService (Active)」，捷徑名稱取名為「ControlService」，然後移到「User's Desktop」。

18. 在「Solution Explorer」選取「Setup1」按滑鼠右鍵選取「View」→「Custom Action」，進入 Custom Action 視窗。

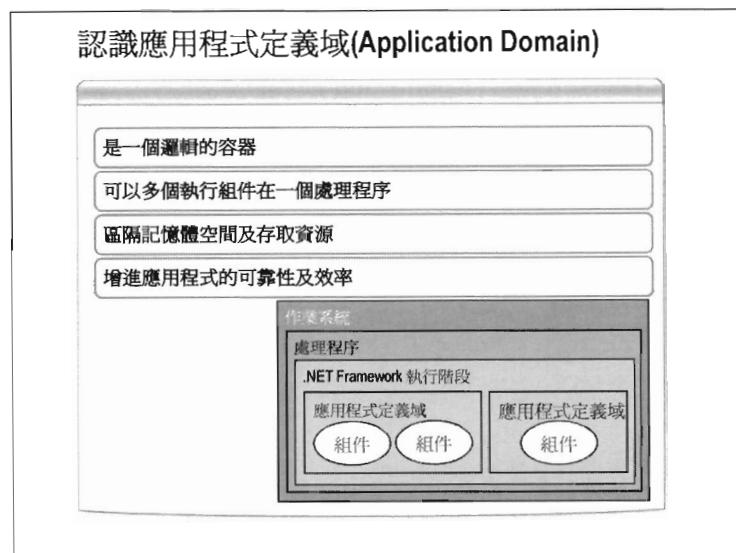


19. 在 Install 項目上按滑鼠右鍵選取「Add Custom Action」。

在 Select Item in Project 項目上雙擊「Application Folder」，點選 Primary output from WatchHDFreeSpace(Active)，並按下「OK」。



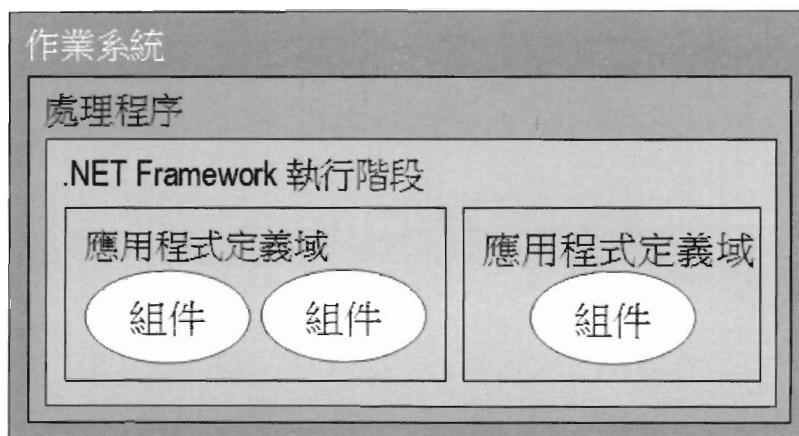
20. 分別在 Commit、Rollback、Uninstall 上進行相同動作。
21. 建置 Setup1 專案，建置完成之後進行安裝。
22. 安裝完成可在桌面看到 ControlService 的程式捷徑，可使用此程式管理 WatchHDFreeSpace 服務。



## 認識應用程式定義域(Application Domain)

應用程式定義域(Application Domain)的設計目的類似 Windows 作業系統的處理程序，處理程序的目的是用來隔離應用程式與應用程式之間的程式碼，避免在執行過程中受到其他應用程式不穩定或安全性的影響，以提高每一個單一應用程式的穩定度與安全性。

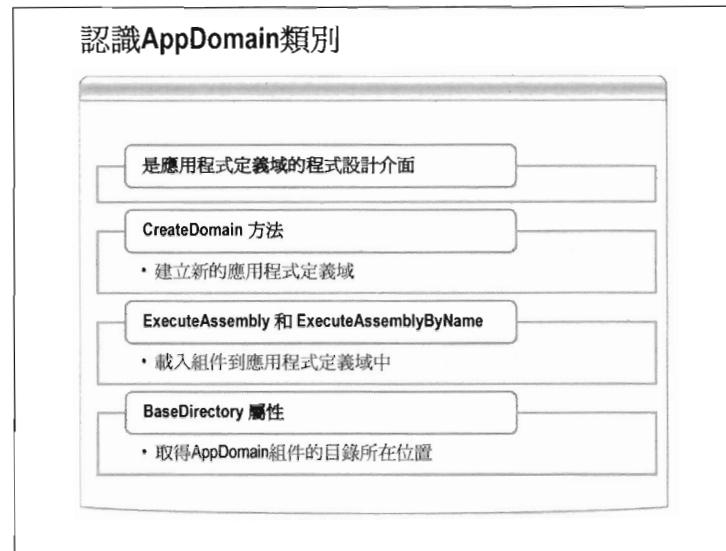
應用程式定義域則是在處理程序中，再進行一層的隔間，每一個隔間可以載入一到多個組件(Assembly)，如下圖：



應用程式定義域像是一個邏輯上的容器，開發者可以依據組件的版本控制、安全性，可靠性等考量進行組件執行環境的隔離。每一個應用

程式定義域獨立控管專屬的記憶體空間，及存取資源，避免遭受其他不相干組件的不良影響，進而達到組件的可靠性及執行效能。

應用程式定義域(Application Domain)通常用在多人共享的服務類型應用程式，像是 Web 應用程式，每個獨立連進來的使用者，需要獨立的執行空間，避免被其他使用者不同的執行階段所影響。以 IIS 5.0 的 ASP.NET 工作處理程序 Aspnet\_wp.exe 為例，假設有 5 個人同時瀏覽到 ASP.NET 的網站，ASP.NET 會為每個使用者建立獨立的應用程式定義域，並將 ASP.NET 的組件(Assembly)執行在各別的應用程式定義域。



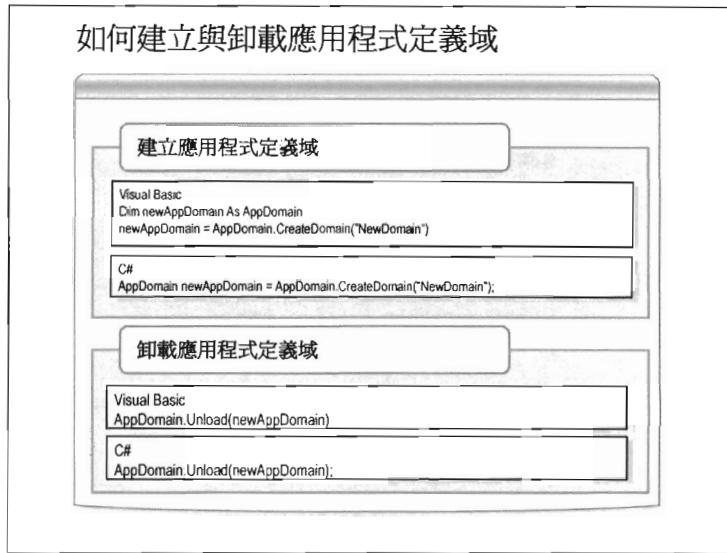
## 認識 AppDomain 類別

AppDomain 類別是應用程式定義域(Application Domain)的程式設計介面。它包含一些可以用來建立定義域及載入組件(Assembly)到定義域的方法，方法列表如下：

- CreateDomain：建立新的應用程式定義域，靜態方法。
- ExecuteAssmby：載入指定路徑檔名的組件到應用程式定義域執行。
- ExecueAssemblyByName：載入指定的組件名稱到應用程式定義域執行。
- Unload：卸載應用程式定義域。

它的屬性包含：

- CurrentDomain，取得目前執行中的應用程式定義域 (Application Domain) ~
- BaseDirectory，取得目前 AppDomain 組件的目錄所在位置。



## 如何建立應用程式定義域

建立應用程式定義域(Application Domain)是呼叫 AppDomain 類別所提供的靜態方法 CreateDomain，並在參數中傳入新的應用程式定義域名稱。如下範例，建立新的應用程式定義域及印出新的定義域名稱。

```

Visual Basic
Dim newAppDomain As AppDomain
newAppDomain = AppDomain.CreateDomain("NewDomain")
Label1.Text &= "新的定義域名稱:" & newAppDomain.FriendlyName

```

```

C#
AppDomain newAppDomain=
    AppDomain.CreateDomain("NewDomain");
Label1.Text += "新的定義域名稱:" +
    newAppDomain.FriendlyName;

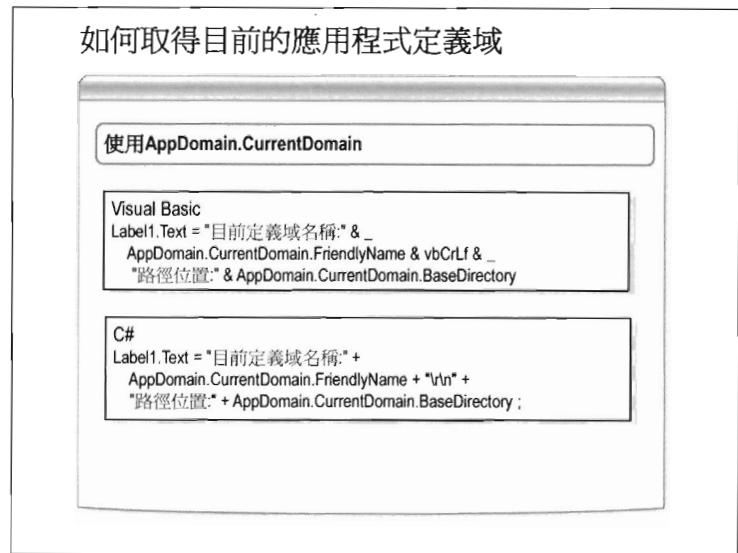
```

## 如何卸載應用程式定義域

卸載應用程式定義域是呼叫 AppDomain 類別所提供的靜態方法 Unload 方法，並在參數中傳入要卸載的應用程式定義域物件。

```
Visual Basic  
AppDomain.Unload(newAppDomain)
```

```
C#  
AppDomain.Unload(newAppDomain);
```



## 如何取得目前的應用程式定義域

取得目前的應用程式定義域(Application Domain)，使用 AppDomain 的 CurrentDomain。下面範例說明，取得目前定義域名稱以及執行程式的組件位置。

Visual Basic

```
Label1.Text = "目前定義域名稱:" & _  
    AppDomain.CurrentDomain.FriendlyName & vbCrLf &_  
    "路徑位置:" & AppDomain.CurrentDomain.BaseDirectory
```

C#

```
Label1.Text = "目前定義域名稱:" +  
    AppDomain.CurrentDomain.FriendlyName + "\r\n" +  
    "路徑位置:" + AppDomain.CurrentDomain.BaseDirectory ;
```

### 練習10.4：載入組件到應用程式定義域

•瞭解如何使用AppDomain的ExecuteAssembly方法載入組件到應用程式定義域(Application Domain)。

•預估實作時間：5分鐘

### 練習 10.4：載入組件到應用程式定義域

#### 目的：

瞭解如何使用 AppDomain 的 ExecuteAssembly 方法載入組件到應用程式定義域(Application Domain)。

#### 功能描述：

在這個練習中將學會將 OtherApp.exe 應用程式載入一個名為「NewDomain」的應用程式定義域。

#### 預估實作時間：5分鐘

#### 實作步驟：

1. 打開 Visual Studio 開發工具，建立專案存放於「\Practices\VB 或 CS\Mod010\_4\Starter\Mod010\_4」，專案名稱為「WinAppDomain」。
2. 從「Solution Explorer」開啟 Form1.cs
3. 從「Toolbox」拖拉 Button 到表單，並設定 Text 屬性「載入組件 AppDomain」。

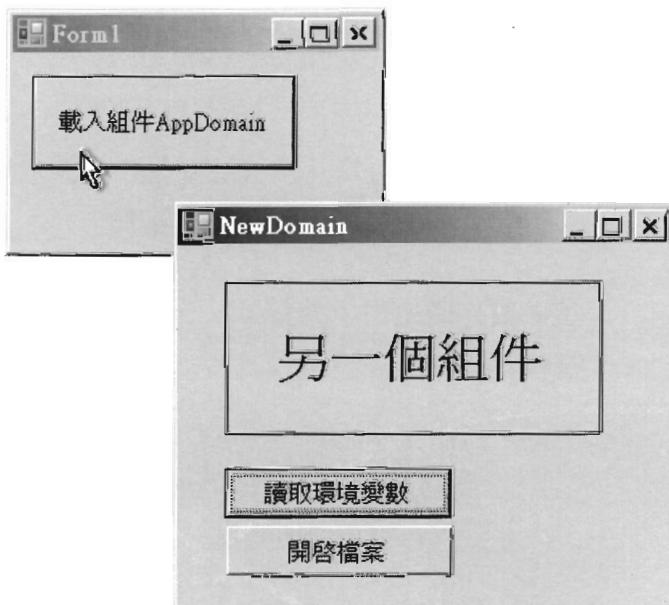
4. 進入 Button1\_Click 事件程序。

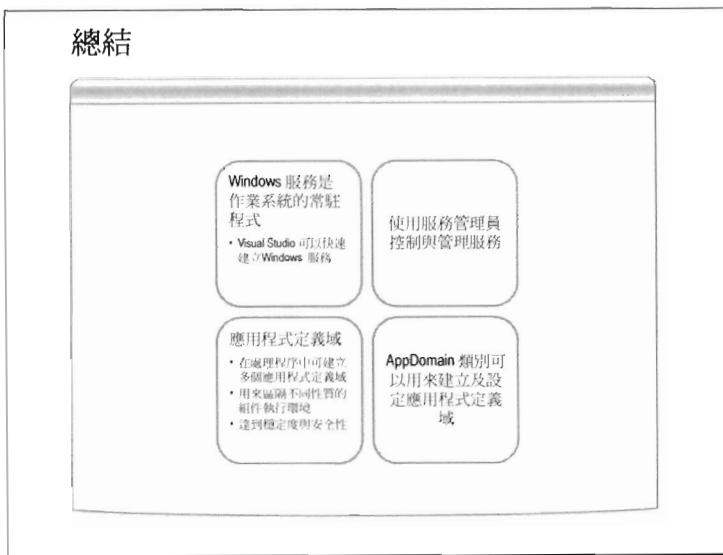
- 使用 AppDomain 的 CreateDomain 方法名為「NewDomain」的應用程式定義域。
- 呼叫 ExecuteAssembly 方法載入 OtherApp.exe。

```
Visual Basic
Dim newAppDomain As AppDomain
newAppDomain = AppDomain.CreateDomain("NewDomain")
newAppDomain.ExecuteAssembly("..\\..\\OtherApp.exe")
```

```
C#
AppDomain newAppDomain ;
newAppDomain = AppDomain.CreateDomain("NewDomain");
newAppDomain.ExecuteAssembly(@"..\..\OtherApp.exe");
```

5. 從「\Practices\VB 或 CS\Mod07\_1\Starter\」複製 OtherApp.exe 到 WinAppDomain 專案目錄中。
6. 按「F5」執行應用程式，並按下「載入組件 AppDomain」按鈕，執行結果如圖：





## 總結

Windows 服務(Service)是作業系統的常駐程式，像是提供電腦基本的網路存取功能就是依賴 Windows 服務--Server 與 Workstation。可以使用服務管理員的工具控制服務的啓動與停用，以及服務執行時使用的帳號。開發 Windows 服務可以使用 Visual Studio，同時別忘了建立 Windows 服務的安裝類別，因為 Windows 服務並非一般 Windows 應用程式可以直接執行.exe 的執行檔，而是必須經過安裝程序註冊的作業系統上，才能成為作業系統的常駐程式也就是 Windows 服務。

在這個章節認識了什麼是應用程式定義域(Application Domain)，是用來區隔不同性質的組件執行環境，同時在一個處理程序中可以依據需求建立多個應用程式定義域，以達到應用程式的穩定度與安全性。

建立、卸載應用程式定義域以及載入應用程式定義域是使用 AppDomain 類別所提供的功能來實作的。

# 第十一章：程式碼安全 性

## 本章大綱

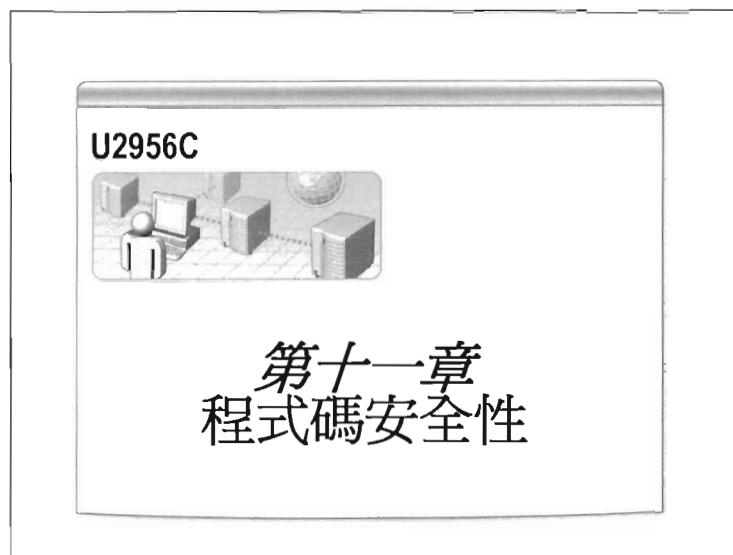
認識程式碼存取安全 (Code Access Security) .....	4
什麼是 Evidence? .....	5
什麼是安全原則 (Security Policy)?.....	7
什麼是程式群組 (Code Group)?.....	9
什麼是安全原則層級 (Security Policy Level)? .....	11
符合多個原則層級 (Policy Levels) 的處理.....	12
使用.NET Framework 組態工具.....	13
保護組件使用宣告式安全 (Declarative).....	16
認識宣告式與命令式安全性.....	17
使用 CAS 組件宣告的原因 .....	19
宣告式 Attribute 的屬性.....	20
如何建立組件宣告 .....	21
練習 11.1：組件宣告式安全 .....	23
練習 11.2：設定安全原則 .....	27
方法的權限安全種類.....	30
Demand 與 LinkDemand 的相異性.....	32
練習 11.3：宣告式與命令式安全性 .....	34
總結 .....	37

提升您專業競爭實力的最佳夥伴

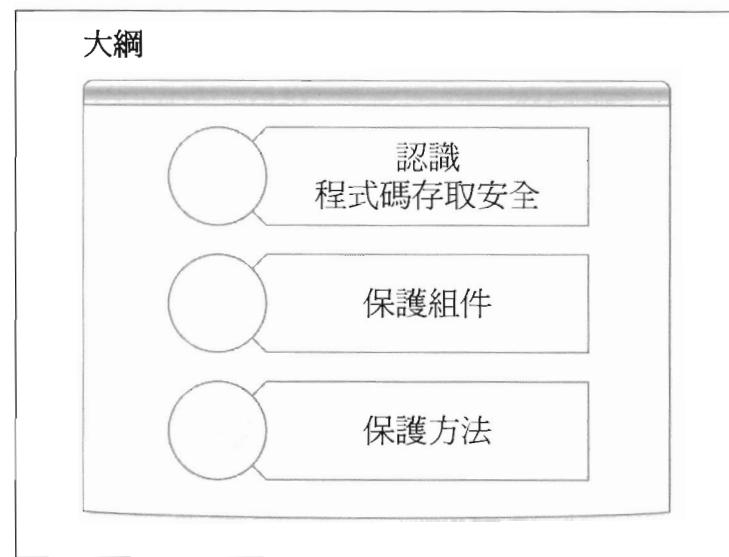


作者：

羅慧真



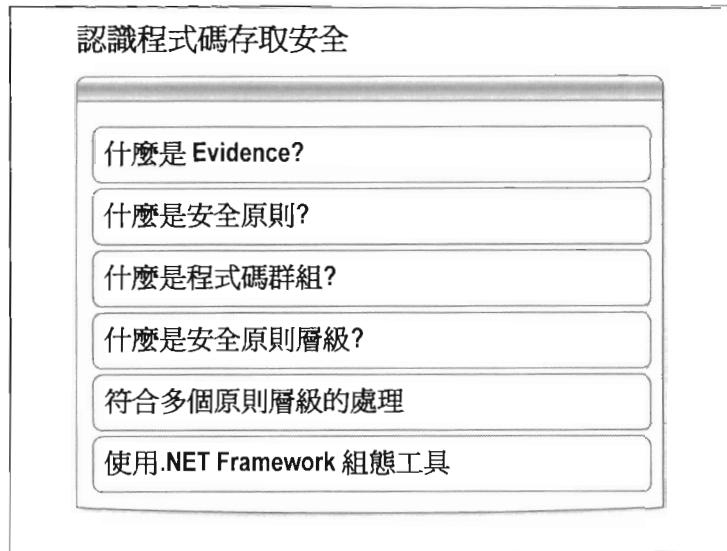
---



在這個章節中將介紹什麼是程式碼存取安全 (Code Access Security)，在這個議題中包含，Evidence、權限 (Permission)、權限集 (Permission Set)、安全原則 (Security Policy)、程式群組 (Code Group) 等名詞解釋與安全性的運作方式。另外也會介紹如何保護組件以及如何保護方法。

本章介紹以下主題：

- 認識程式碼存取安全 (Code Access Security)
  - Evidence
  - 權限
  - 安全原則
- 保護組件
  - 使用宣告式安全
- 保護方法
  - 使用宣告式安全
  - 使用命令式安全

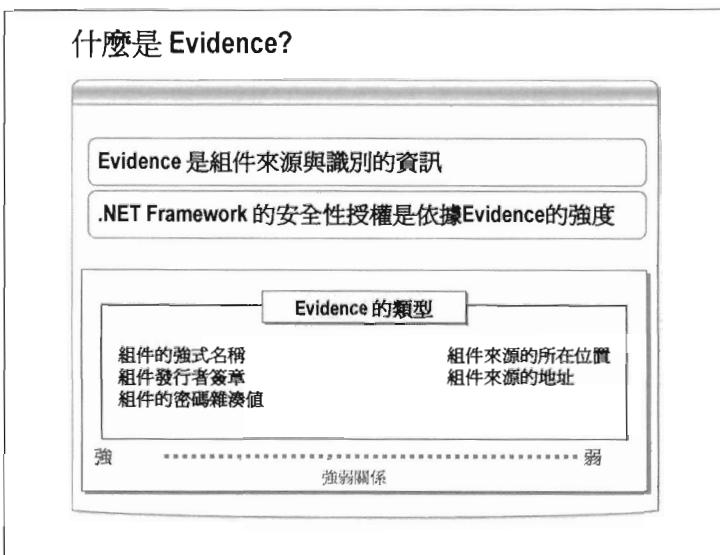


## 認識程式碼存取安全 (Code Access Security)

程式碼存取安全 (Code Access Security, 簡稱為 CAS) 是一種讓管理人員及開發人員控制應用程式授權的安全性系統。有別於以人或角色為授權對象的角色安全性 (Role Base Security, 簡稱為 RBS)，程式碼存取安全是以組件的來源、識別資訊 (Evidence) 做為授權對象，避免不明的程式來源危害系統的安全性。

這一節將認識程式碼存取安全的相關名詞與其運作原理，本節介紹以下主題：

- 什麼是 Evidence?
- 什麼是安全原則 (Security Policy)?
- 什麼是程式碼群組 (Code Group)?
- 什麼是安全原則層級 (Security Policy Level)?
- 符合多個原則層級 (Policy Levels) 的處理
- 使用.NET Framework 組態工具



## 什麼是 Evidence?

Evidence 是組件的來源與識別的相關資訊。

以組件來源而言包含以下幾種：

- 組件來源的所在位置 (Zone) ，像是本機電腦 (local Computer) 、網際網路 (Internet Zone) 、內部網路 (Intranet Zone) 。
- 組件來源的地址，使用 URL (Uniform Resource Locator) 、 UNC (Universal Naming Convention) 或本機磁碟路徑表示。

以組件識別資訊而言包含以下幾種：

- 組件的強式名稱 (Strong Name)，強式名稱是由唯一識別的公開金鑰、組件的簡式名稱 (simple name) ，以及版本所組合而成。
- 組件發行者簽章，組件包含發行者的數位簽章。
- 組件的密碼雜湊值 (Cryptographic hash) 。

Evidence 的類型可以由開發者自行依據組件的需要而自訂其類型，只要在執行階段自訂的 Evidence 類型符合安全原則組態便可執行該組件。

## 執行階段如何使用 Evidence

在組件被載入執行階段時，它會使用執行階段的主程式 (host) 目前組件的 Evidence 做為 .NET Framework 的安全系統。目前的 Evidence，.NET Framework 有三種不同的主程式 (host) 執行階段，Shell 主程式、Microsoft ASP.NET 主程式、Microsoft Internet Explorer 主程式。使用哪種主程式做為執行階段的主程式，這要看程式是如何執行的，例如，應用程式的啓動是由命令提示列或檔案總管啓動的，則使用 Shell 主程式做為執行階段。

至於安全系統如何決定組件的權限層則是依據組件 Evidence 的強度來決定。

## Evidence 的強度

Evidence 的強弱關係與組件識別及來源的 Evidence 類型有關。組件識別資訊提供強度高的 Evidence，程式較不容易遭受到欺騙行為。例如，組件的強式名稱提供了可靠的 Evidence，因為若要竄改組件的強式名稱，除非能夠取得程式發行者的私密金鑰，否則將是一件非常困難的工程。組件發行者簽章也屬於強度高的 Evidence 類型。

相形之下，組件來源的 Evidence 則為較弱的 Evidence。像是，組件來自於 Zone 或 URL 的識別方式很容易遭到竄改與欺騙行為。如果給予這類組件較高的執行權限則程式遭到安全風險相對的提高。

## 什麼是安全原則 (Security Policy)?

安全原則負責控制程式碼存取支援的權限

程式碼的識別及來源決定存取的結果

程式碼的識別及來源即是Evidence

安全原則會對照到符合的Evidence的權限集

- Nothing
- Execution
- Internet
- LocalIntranet
- Everything
- FullTrust
- Custom-defined

## 什麼是安全原則 (Security Policy)?

程式存取安全原則 (Code Access Security Policy) 是.NET Framework 的安全系統設計特性之一，可以預防只有以使用者做為授權對象的軟體所帶來的安全性弱點。

程式存取安全原則可以讓使用者或管理者控制以組件的識別資訊或來源為基礎給予權限組件的層級。

### .NET Framework 內建的權限類別

.NET Framework 提供許多內建的程式存取權限類別，這些類別是為了保護系統資源的存取。內建的權限類別如下清單：

- FileIOPermission，檔案系統中的檔案及目錄權限。
- IsolatedStorageFilePermission，隔離儲存的權限。
- RegistryPermission，存取註冊機碼的權限。
- EventLogPermission，存取事件檢視器的權限。
- EnvironmentPermission，存取環境變數的權限。
- DirectoryServicesPermission，存取作業系統目錄服務的權限。
- PerformanceCounterPermission，存取效能計數器的權限。
- PrintingPermission，使用印表功能的權限。

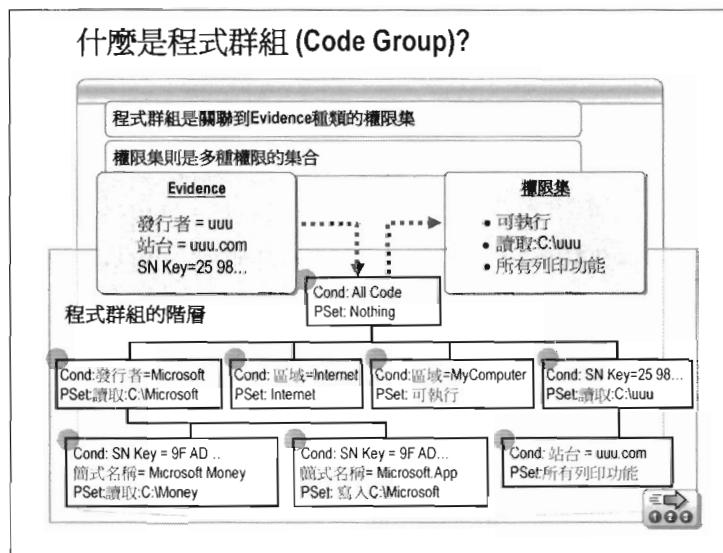
- `SqlClientPermission`，存取資料提供者 Microsoft SQL Server 的權限。

其他相關權限類別請查詢 MSDN 線上說明。如需內建之外的其他權限類別，開發人員可以實作自訂類別來擴充權限類別。

安全性原則會比對 Evidence 所設定的權限集合來決定應用程式的執行權限，以下項目是預設的權限集合。

- `Nothing`，沒有權限，程式將無法執行。
- `Execution`，有執行的權限，但無法使用受保護的資源。
- `Internet`，有執行的權限，並具有建立上層視窗、檔案對話方法的能力，以及存取隔離儲存區 (Isolated Storage) 限制的配額空間。從網際網路來的組件預設使用此權限集。
- `LocalIntranet`，有執行的權限，可以無限制的建立使用者介面、使用存取隔離儲存區 (Isolated Storage) 沒有配額限制、使用 DNS 服務，及讀取 USERNAME、TEMP 及 TMP 環境變數等權限。來自於內部網路的組件預設享用此權限集。
- `Everything`，所有內建的權限，除了 Skip Verification 權限外。
- `FullTrust`，完全享有所有資源的存取。

這些是系統預設的權限集，如有需要依據程式特殊需求可自訂權限集。



## 什麼是程式群組 (Code Group)?

程式群組 (Code Group) 是由一些符合特定條件成員所組合而成的程式邏輯群組。任何程式只要符合這些成員條件 (Membership condition) 便屬於該群組。這些成員條件是使用 Evidence 定義。

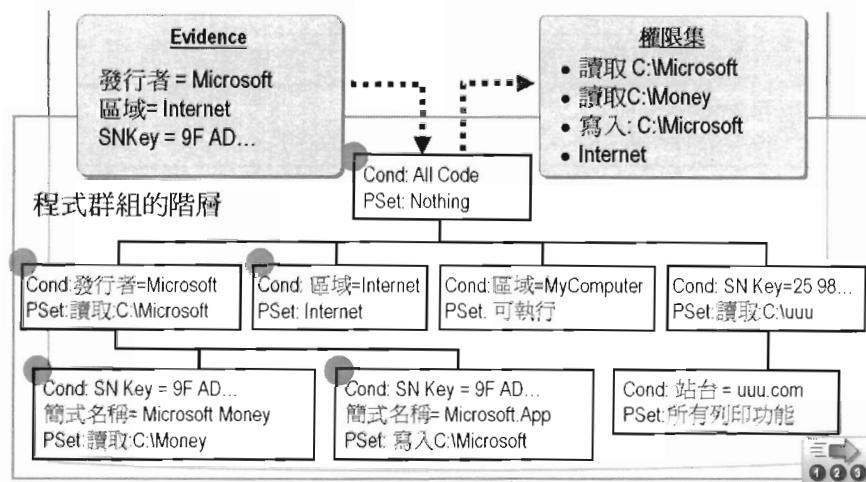
程式群組是由成員條件及權限集合 (Permissions Set) 組成的。應該這樣說，符合成員條件的程式允許執行某些權限集合這樣的組合便是程式群組。當組件在載入時，系統會檢查組件的 Evidence 是否符合某個程式群組中定義的成員條件，如果符合，便具有程式群組所付予的權限集合。

系統可以設定的成員條件 (Membership condition)有以下幾項：

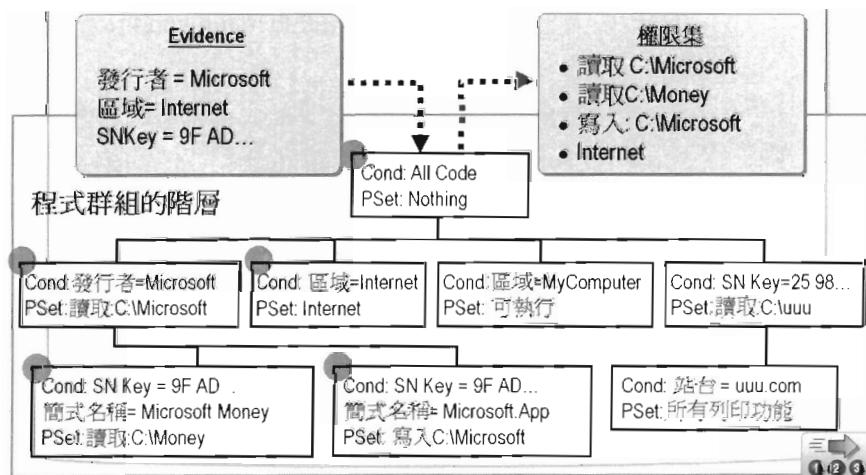
- Application directory，應用程式的所在目錄。
- Cryptographic hash，組件由 MD5 或 SHA1 的雜湊演算法所產生的雜湊值。
- Software publisher，組件中數位簽章的公開金鑰。
- Strong Name，強式名稱組件，包含公開金鑰、簡式名稱及版本。
- URL，組件來源，像是 [http://www.microsoft.com/\\*](http://www.microsoft.com/*) 或 [//Server1/AppFolder](http://Server1/AppFolder)。

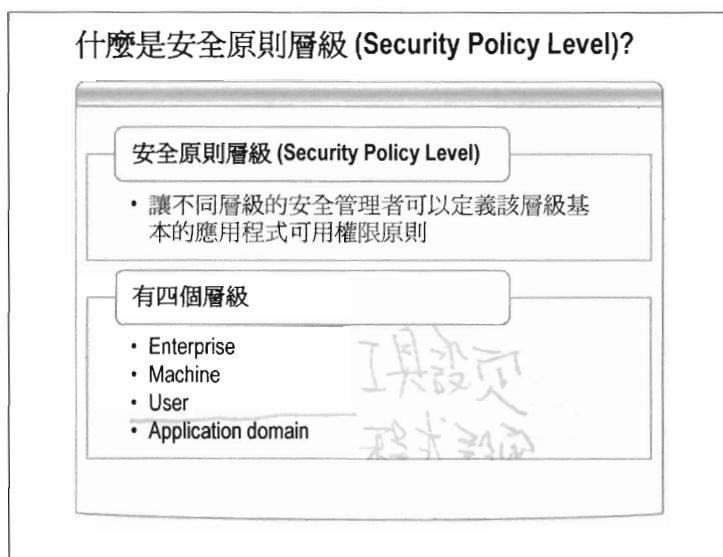
- Web Site，網站的組件來源，像是\*.microsoft.com 或 [www.microsoft.com](http://www.microsoft.com)。
- Zone，安全區域的組件來源，像是 MyComputer、Internet... 等。

例如，應用程式的發行者是 Microsoft，來自於 Internet，強式名稱金鑰是 9F AD...，它所符合的程式群組有 All Code、Microsoft、Internet，及強式名稱的鍵值是 9F AD...五個程式集合。這些權限取聯集結果為，可讀寫 C:\Microsoft 目錄、唯讀 C:\Money 目錄、Internet 的權限集。



另一個組件的發行者是 uuu 公司，來自於 uuu.com 網站，強式名稱金鑰是 25 98 ...，它符合的程式群組是 All Code、MyComputer、強式名稱金鑰是 25 98 ...，及網站 uuu.com 等四個程式群組。允許的權限有 Execution、讀取 C:\uuu 目錄，所以印表機的存取權。



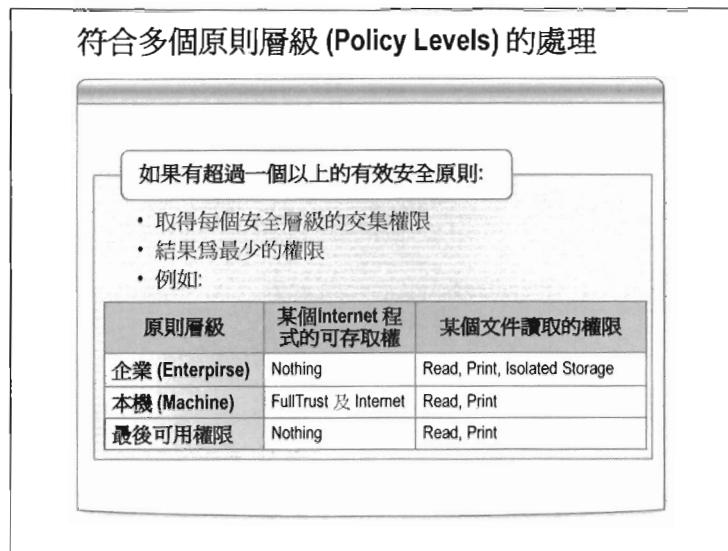


## 什麼是安全原則層級 (Security Policy Level)?

為了企業整體的安全性，企業網域 (Enterprise Domain) 的系統管理者對於可信任的軟體來源或發行者可能與一般使用者有不同的想法。安全原則層級 (Security Policy Level) 讓不同層級的安全管理者可以定義該層級基本的應用程式可用權限原則。

安全原則層級有四個層級。每個原則層級包含階層式的程式群組。

- Enterprise policy，由企業管理者 (Enterprise Administrators) 定義企業網域 (Enterprise Domain) 的安全原則。包含階層式的程式群組並套用到整個網域所有應用程式。
- Machine policy，由電腦管理者 (Administrators) 定義電腦的安全原則。包含階層式的程式群組並套用到整台電腦的所有應用程式。
- User policy，由使用者定義登入帳號的安全原則。
- Application domain policy，由執行階段的程式定義。



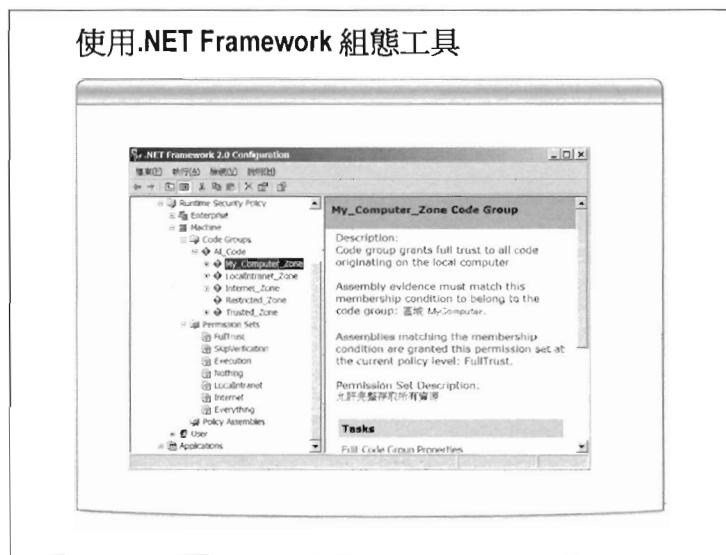
## 符合多個原則層級 (Policy Levels) 的處理

在有多層安全原則層級下，應用程式載入時必然需套用多個層級的程式群組 (Code Group)，最後會被套用到應用程式或組件的存取權限是什麼？

套用原則是同一層的程式群組 (Code Group) 採用聯集，聯集後的結果再與其他層級使用交集的方式取得最後的存取權限。

案例 1，某個 Internet 的應用程式在企業層 (Enterprise Level) 為 Nothing，在本機層 (Machine Level) 符合二個程式群組獲得 FullTrust 及 Internet，聯集後的結果為 FullTrust。企業層 (Enterprise Level) 與本機層 (Machine Level) 的權限以交集計算最後權限為 Nothing。

案例 2，某個讀取文件的應用程式，在企業層 (Enterprise Level) 符合二個程式群組獲得 Read、Print、Isolated Storage，聯集後的結果為 Read、Print、Isolated Storage，在本機層 (Machine Level) 符合二個程式群組獲得 Read 及 Print，聯集後的結果為 Read 及 Print。企業層 (Enterprise Level) 與本機層 (Machine Level) 的權限以交集計算最後權限為 Read 及 Print。

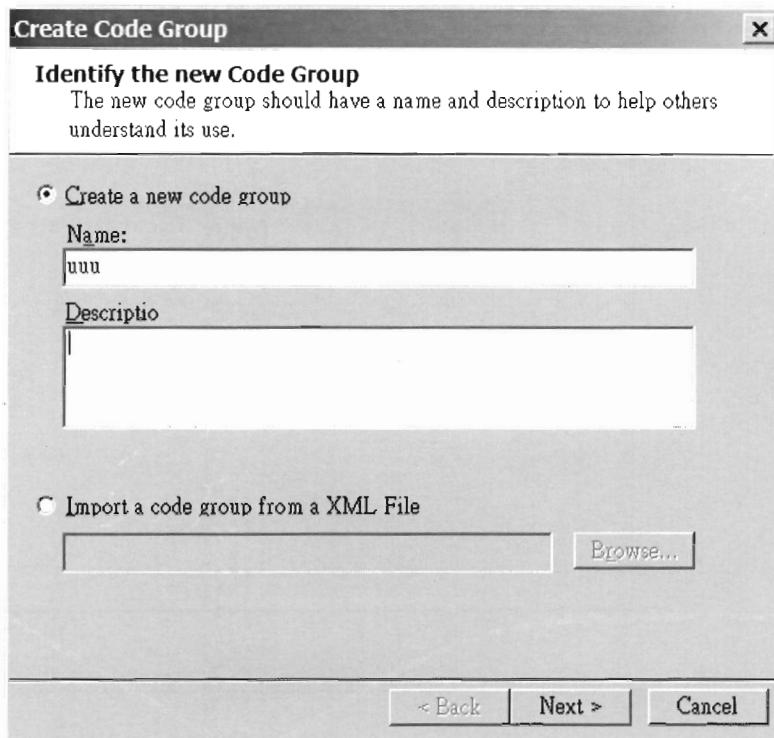


## 使用.NET Framework 組態工具

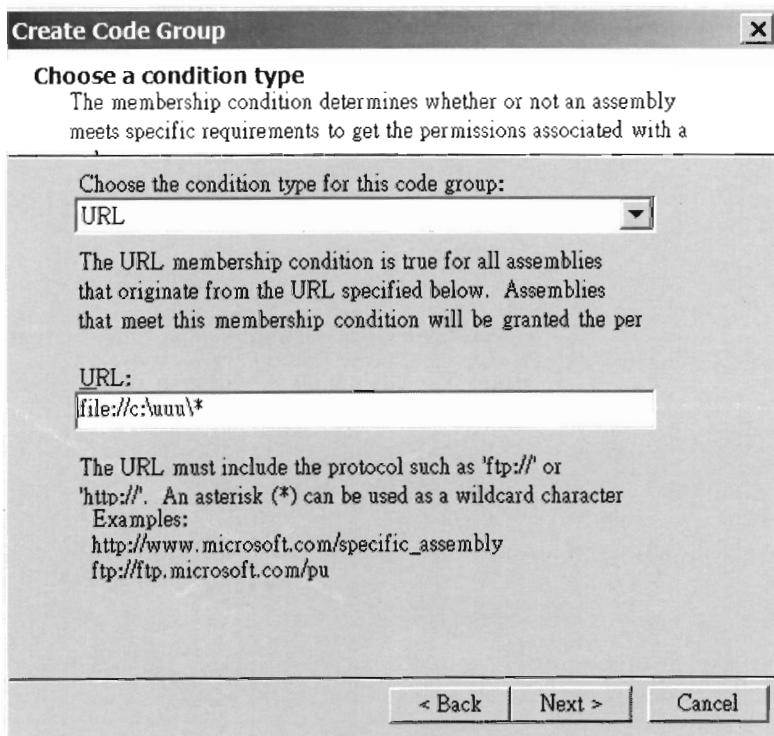
.NET Framework 的組態工具，在「Administrative Tools」→「Microsoft .NET Framework Configuration」項目。

以下是在本機層 (Machine Level) 建立新的程式群組 (Code Group) 的步驟：

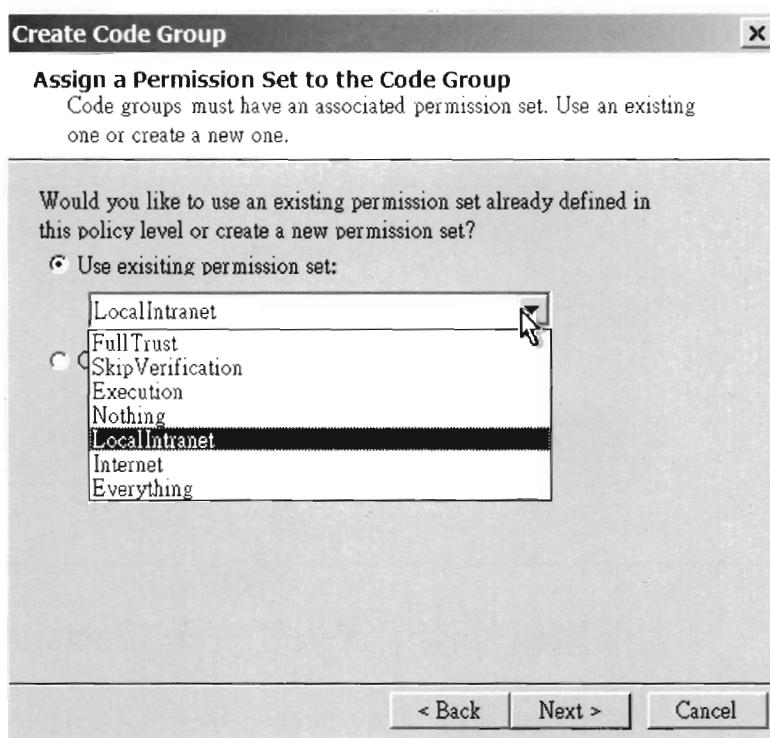
1. 展開「Runtime Security Policy」、「Code Groups」、「All\_Code」。點選「All\_Code」項目然後按滑鼠右鍵選取「New...」。
2. 在「Create Code Group」視窗選取「Create a new code group」，並為其取名。例如，uuu。然後按「Next」按鈕。



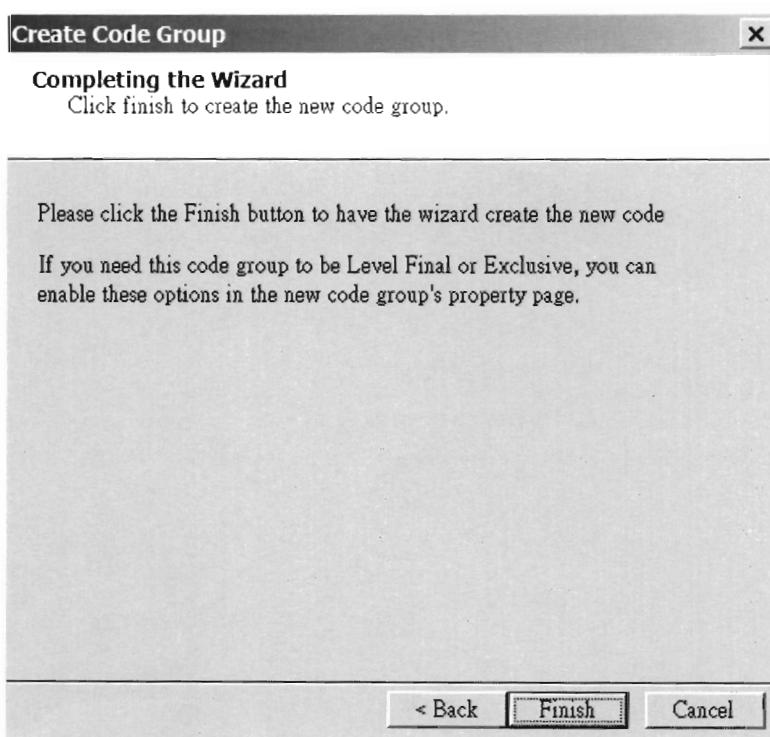
3. 在「Choose the condition type for this code group:」選取程式群組的成員條件，例如：URL。然後設定條件，例如：[file:///c:/uuu/\\*](file:///c:/uuu/*)。接著按「Next」。

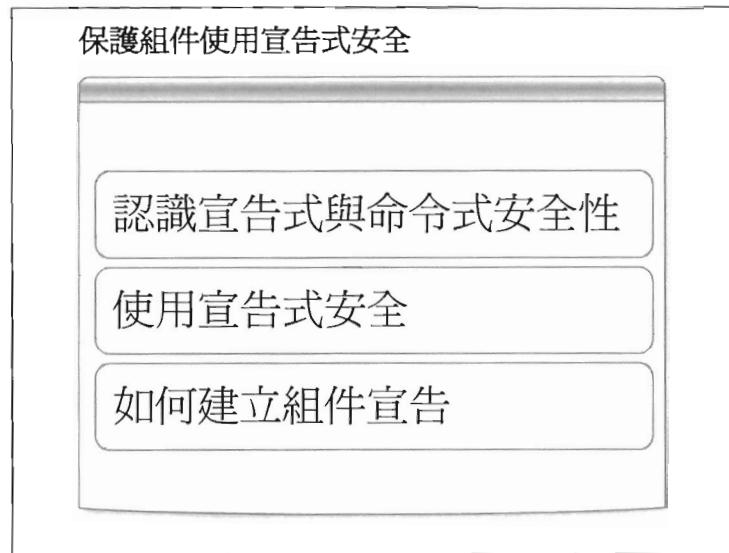


4. 在「Use existing permission set:」下拉選項選取適合的權限集。例如，LocalIntranet。接著按「Next」。



5. 按「Finish」完成建立程式群組的精靈步驟。



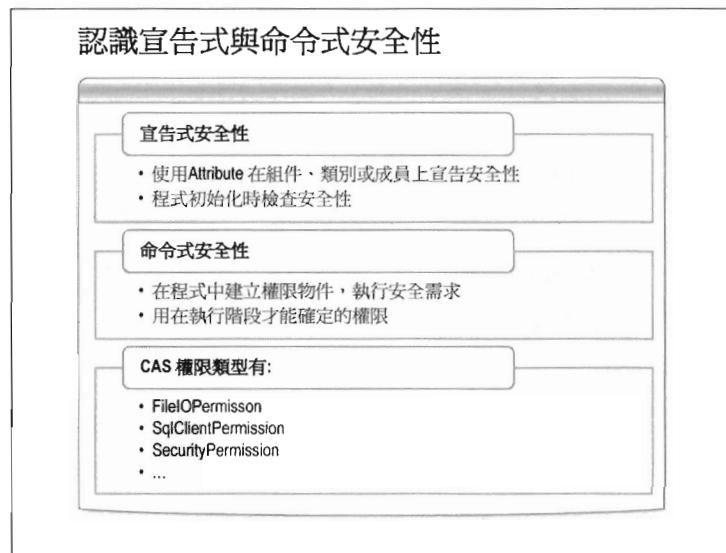


## 保護組件使用宣告式安全 (Declarative)

前一節你瞭解了什麼是程式碼存取安全 (以下簡稱 CAS) 及使用 CAS 限制應用程式可以執行的權限。在這一節中我們將從應用程式的角度去要求執行權限，並限制應用程式執行在部分信任 (Partially Trusted) 的安全環境中。

從應用程式角度要求或檢查程式碼存取安全有二種方式一種是宣告式 (Declarative) 、一種是命令式 (Imperative)。本節將介紹什麼是宣告式安全性以及什麼是命令式安全性，為何要使用宣告式的原因，以及相關的實作方式。本節將介紹以下主題：

- 認識宣告式與命令式安全性
- 使用宣告式安全
- 如何建立組件宣告



## 認識宣告式與命令式安全性

CAS 在程式中的安全寫作方式有二種，一個是宣告式安全性、一個是命令式安全性：

- 宣告式安全性 (Declarative Security)

使用 Attribute 在組件、類別或成員上進行安全性的宣告，這會在程式初始化時檢查安全性。

- 命令式安全性 (Imperative Security)

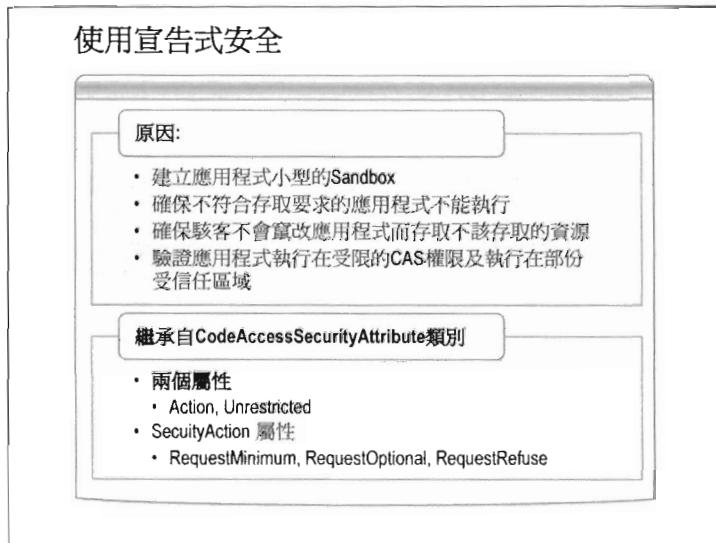
在程式中建立權限物件，執行安全需求。用在執行階段才能確定的權限。

## CAS 權限類型

CAS 可以限制存取多種不同類型的資源，包含檔案、目錄、印表機，網路資源...等多項類型。每種資源類型在.NET Framework 都有提供相對的類別，下列為其部份常見的類型：

- FileIOPermission，附加內容、讀寫於檔案或目錄。

- `DnsPermission`，存取 Domain Name System (DNS)。
- `SqlClientPermission`，存取 SQL 資料庫。
- `SecurityPermission`，執行程式、建立和操作 AppDomain 的能力，略過這個組件中程式碼驗證的能力 (Skip Verification)。
- `UIPermission`，存取使用者操作界面的功能。
- `StrongNameIdentityPermission`，定義強式名稱 (Strong Name) 的識別使用權限。用來定義存取某個型別的 Public 成員的強式名稱需求。



## 使用 CAS 組件宣告的原因

使用程式碼存取安全 (簡稱: CAS) 的宣告式安全有三個之要的原因：

- 確保不符合安全存取要求的應用程式無法啓動  
如果不想應用程式在執行階段出現安全性的例外狀況，可以在組件宣告 CAS 必要的權限，使用 SecurityAction.RequestMinimum 定義應用程式至少需滿足的需求。如果使用者電腦環境上並不允許應用程式所定義的最少需求，而使用者又啓動該應用程式時，應用程式將會在啓動階段就出現安全性錯誤通知，並終止應用程式的啓動。
- 為應用程式建立小型的 Sandbox，確保駭客不會竄改應用程式而存取不該存取的資源。  
應用程式或組件限制存取資源最少的權限，只允許應用程式或組件在有限的安全環境之下執行及存取資源，這樣可以避免駭客竄改組件或記憶體中的程式堆疊，企圖非法存取不該存取的系統資源。
- 驗證應用程式執行在受限的 CAS 權限及執行在部分受信任區域。  
如果應用程式中有某些不是太主要的功能，但那些功能會存取到權限受限的資源，那麼你可以將那些資源的權限要求為 SecurityAction.RequestOptional，當執行環境並未授權存取

SecurityAction.RequestOptional 指定的資源，應用程式仍會繼續執行，直到程式存取到那些資源時，系統將會發生 System.Security.Policy.PolicyException。

## 宣告式 Attribute 的屬性

CAS 的資源權限類別的宣告式 Attribute 皆繼承自 CodeAccessSecurityAttribute 類別。所以每個資源權限類別都會有二個屬性，如下：

- **Action**，指定資源權限的行動。型別是 SecurityAction 列舉常數。
- **Unrestricted**，一個 Boolean 值。如果設定為 True，代表同意無限制的存取該資源。

## SecurityAction 選項

用來定 Action 屬性的選項。當使用宣告式安全性 (Declarative Security) 時有三個選項可以使用：

- **SecurityAction.RequestMinimum**，指定組件必要的執行權限。如果組件執行的安全環境不符合此權限，在載入時將會發生 System.Security.Policy.PolicyException。
- **SecurityAction.RequestOptional**，使用 RequestOptional 與 RequestMinimum 列舉僅需的權限相關於間接的拒絕其他的權限。執行環境若不符合 RequestOptional 所要求的權限時，組件仍會被載入執行，直到程式執行到沒有權限的資源時，才會發生 SecurityException 例外錯誤。
- **SecurityAction.RequestRefuse**，拒絕可能被濫用的資源權限。當組件的 Evidence 在使用者的電腦上符合某些執行權限，組件中若有設定 RequestRefuse 可拒絕使用該權限。這個拒絕的動作只有在組件範圍內有效。應用狀況可以是使用 RequestOptional 要求一個權限集，但權限集中有某幾項是組件不會用到的權限，必免遭濫用，可以使用 RequestRefuse 拒絕那些不用的權限。



## 如何建立組件宣告

程式必須匯入 System.Security.Permission 的命名空間，然後依據組件所需的必要權限進行宣告。

例如，這個程式只需使用到視窗使用者介面，以及需要讀取 C:\Windows 目錄下的檔案，於是使用 UIPermission 及 FileIOPermission 宣告權限，並使用 SecurityPermission 指定 Unrestricted 為 True，代表這個組件無限制的存取 SecurityPermission 的資源。

```

Imports System.Security.Permissions
Imports System.IO

<Assembly: UIPermission(SecurityAction.RequestMinimum, Windo
w:=UIPermissionWindow.SafeTopLevelWindows)>
<Assembly: FileIOPermission(SecurityAction.RequestOptional, Vie
wAndModify:="C:\Windows")>
<Assembly: SecurityPermission(SecurityAction.RequestOptional, U
nrestricted:=True)>

Public Class Form1

    Private Sub button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles button1.Click
        Dim mydir As New DirectoryInfo("C:\Windows")
        Dim fileList As FileInfo() = mydir.GetFiles()
    End Sub

```

```
    MessageBox.Show("C:\Windows 下共有" + fileList.Length.ToString() + "檔案.")
End Sub
End Class
```

```
C#
using System.Security.Permissions ;
using System.IO;

[assembly: UIPermission(SecurityAction.RequestMinimum, Window
=UIPermissionWindow.SafeTopLevelWindows)]
[assembly: FileIOPermission(SecurityAction.RequestOptional, View
AndModify=@"C:\Windows")]
[assembly: SecurityPermission(SecurityAction.RequestOptional, Un
restricted = true )]

namespace CASDeclare
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            DirectoryInfo mydir = new DirectoryInfo (@"C:\Windows
");
            FileInfo [] fileList = mydir.GetFiles ();
            MessageBox.Show (@"C:\Windows 下共有" + fileList.Length.ToString () + "檔案.");
        }
    }
}
```

### 練習11.1：組件宣告式安全

- 瞭解如何建立組件宣告式安全。
- 在這個練習應用程式會使用到SQL連線，並限制只能連到某台伺服器，若連到其他伺服器將會發生權限不足。

•預估實作時間：15分鐘

### 練習 11.1：組件宣告式安全

#### 目的：

瞭解如何建立組件宣告式安全。

#### 功能描述：

在這個練習應用程式會使用到 SQL 連線，並限制只能連到某台伺服器，若連到其他伺服器將會發生權限不足。

#### 預估實作時間：15 分鐘

#### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod11\_1。
3. 開啟 Form1 的設計視窗，從「Toolbox」拖拉控制項表單並設定屬性如下：

控制項	屬性	值
Label1	ID	Label1
	Text	連線字串:
TextBox1	ID	TextBox1
	Text	建立連線

4. 進入 Form1 的程式碼視窗，匯入必要的命名空間：

```
Visual Basic
Imports System.Security.Permissions
Imports System.Data.SqlClient
```

```
C#
using System.Security.Permissions;
using System.Data.SqlClient ;
```

5. 在 Form1 程式碼視窗 Namespace 之下的位置宣告組件的安全性，如下：

```
Visual Basic
<Assembly: UIPermission(SecurityAction.RequestMinimum, Window:=UIPermissionWindow.SafeTopLevelWindows)>
<Assembly: SqlClientPermission(SecurityAction.RequestMinimum, ConnectionString:="server=.;database=northwind;integrated security=true")>
<Assembly: SecurityPermission(SecurityAction.RequestOptional, Unrestricted:=True)>
```

```
C#
[assembly: UIPermission(SecurityAction.RequestMinimum, Window=UIPermissionWindow.SafeTopLevelWindows)]
[assembly: SqlClientPermission(SecurityAction.RequestMinimum, ConnectionString="server=.;database=northwind;integrate d security=true")]
[assembly: SecurityPermission(SecurityAction.RequestOptional, Un restricted=true)]
```

6. 進入 Button1\_Click 事件程序，建立連線，連線字串為 TextBox1 中輸入的值。

```
Visual Basic
Dim cn As New SqlConnection(TextBox1.Text)
Try
```

```

cn.Open()
MessageBox.Show("連線成功")
Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try

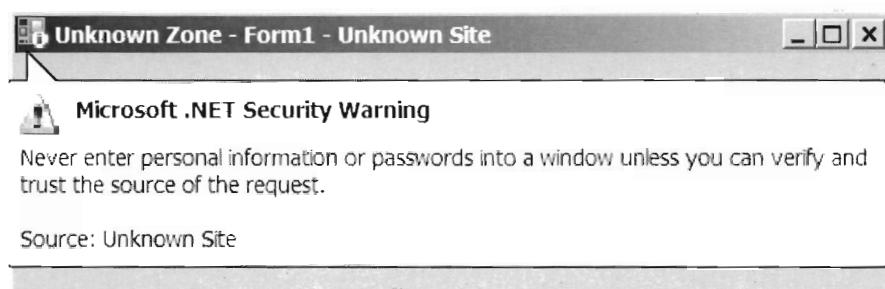
```

```

C#
SqlConnection cn = new SqlConnection (textBox1.Text );
try {
    cn.Open ();
    MessageBox.Show ("連線成功.");
}
catch (Exception ex) {
    MessageBox.Show (ex.Message );
}

```

7. 按「F5」執行應用程式。在 Form1 的上角會出現安全提示。  
點一下安全提示即可開始使用。



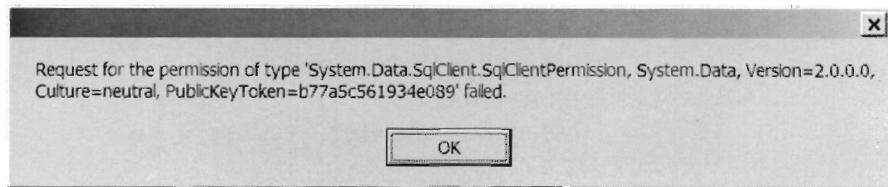
8. 在連線字串欄位輸入「server=.;database=northwind;integrated security=true」，按「建立連線」，結果如下：



9. 修改連線字串的參數，例如伺服器名稱、資料庫或安全登入的資訊，按「建立連線」，例如：

```
server=.\sqlexpress;database=northwind;uid=sa
```

10. 結果如下：



### 練習11.2：設定安全原則

- 瞭解如何設定安全原則。
- 延續前一個練習，設定 C:\Test 資料夾的安全原則，並將 Mod10\_1.exe 部署到 C:\Test。

• 預估實作時間：10分鐘

### 練習 11.2：設定安全原則

#### 目的：

瞭解如何設定安全原則。

#### 功能描述：

延續前一個練習，設定 C:\Test 資料夾的安全原則，並將 Mod10\_1.exe 部署到 C:\Test。

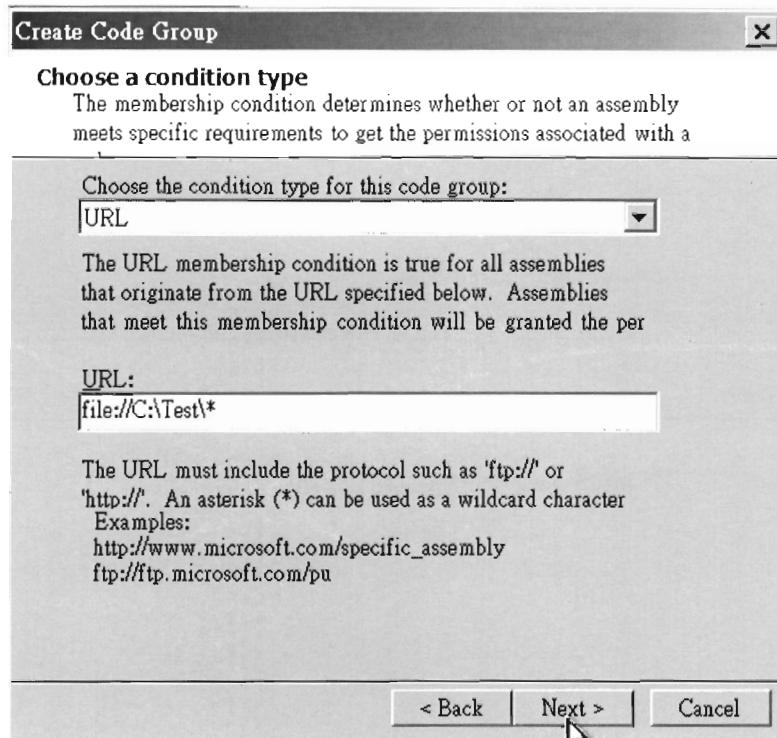
預估實作時間：10 分鐘

#### 實作步驟：

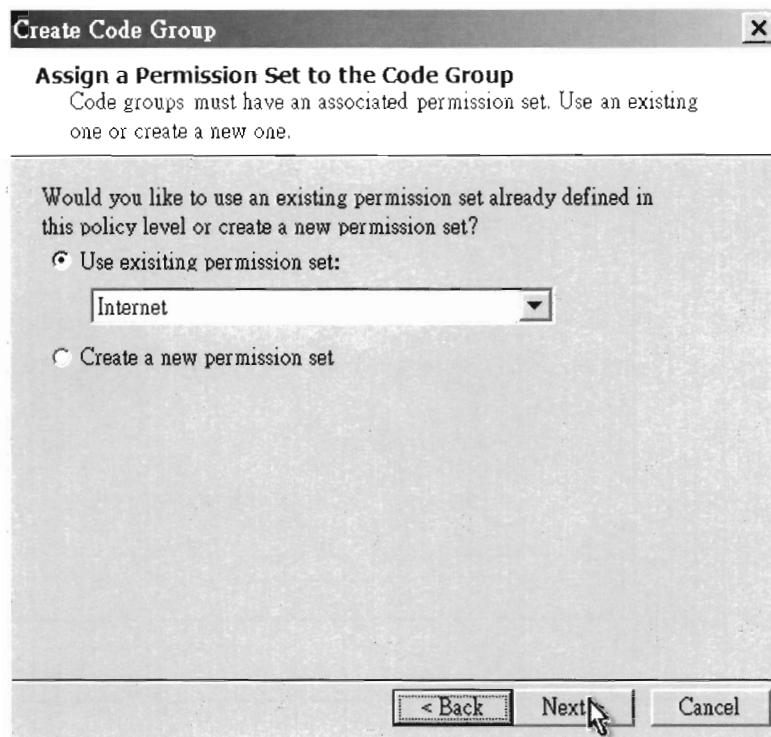
1. 打開檔案總管在 C 磁碟的根路徑建立資料夾「Test」。
2. 上個練習所產生的 Mod11\_1.exe 複製到 C:\Test 目錄。
3. 執行應用程式，在連線字串欄位輸入  
`「server=.;database=northwind;integrated security=true」`。結果應該與練習 11.1 相同。
4. 從「Administration Tools」→「.NET Framework 2.0 Configuration」，展開「Runtime Security Policy」、「Code Group」、「All\_Code」。

5. 在「All\_Code」項目上按滑鼠右鍵選取「New...」新增一個 Code Group 名稱為「Mod11\_2」，按「Next」。

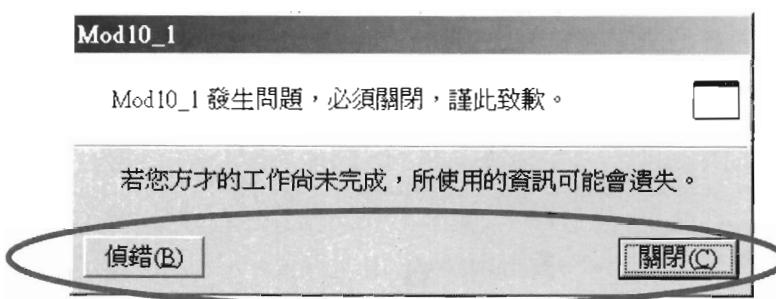
6. 在「Choose the condition type for this code group:」下拉選取 URL，在「URL:」欄位輸入「file:///c:\test\\*」，按「Next」。

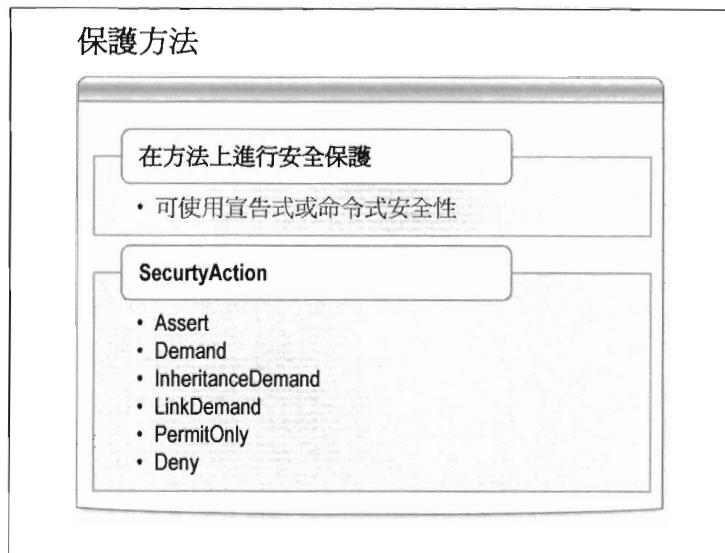


7. 選取「Use existing permission set:」為「Internet」，按「Next」。



8. 接著在下個畫面按「Finish」按鈕。
9. 在建立好的 Code Group 項目「Copy of mod11\_2」上按滑鼠右鍵選取「Properties」，在內容的「General」頁下方勾選「This policy level will only have the permissions form the permission set associated with this code group」。
10. 回到檔案總管 C:\Test，重新執行 Mod11\_1.exe，程式將因權限不足而無法執行，並得到如下結果：





## 方法的權限安全種類

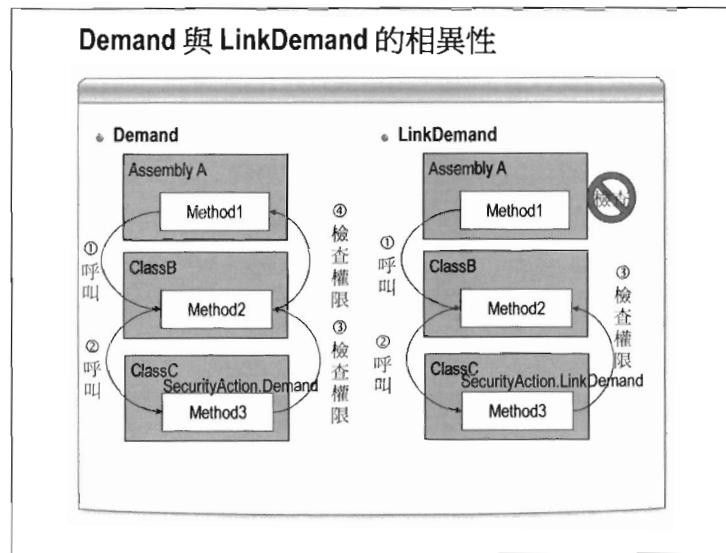
SecurityAction 除了前面所認識的那三種 (RequestMinimum、RequestOptional、RequestRefuse) 僅限於使用在組件宣告式安全性 (Declarative Security) 之外，還有六種項目這些項目皆可用在宣告式安全性與命令式安全性 (Imperative Security)。包含以下六項：

- Assert，即使堆疊中較高層的呼叫端尚未授與存取資源的使用權限，進行呼叫的程式碼仍可以存取由目前的使用權限物件所識別的資源。
- Demand，所有在呼叫堆疊中較高層的呼叫端，必須已經被授與由目前使用權限物件所指定的使用權限。
- InheritanceDemand，繼承類別或覆寫方法的衍生類別 (Derived Class) 必須已經授與指定的使用權限。
- LinkDemand，呼叫端必須已經授與指定的使用權限。
- PermitOnly，即使程式碼已經授與存取其他資源的使用權限，仍只能存取由這個使用權限物件所指定的資源。
- Deny，即使已經被授與使用權限，仍會拒絕呼叫端存取由目前使用權限物件所指定資源的功能。

## 使用方法權限的原則

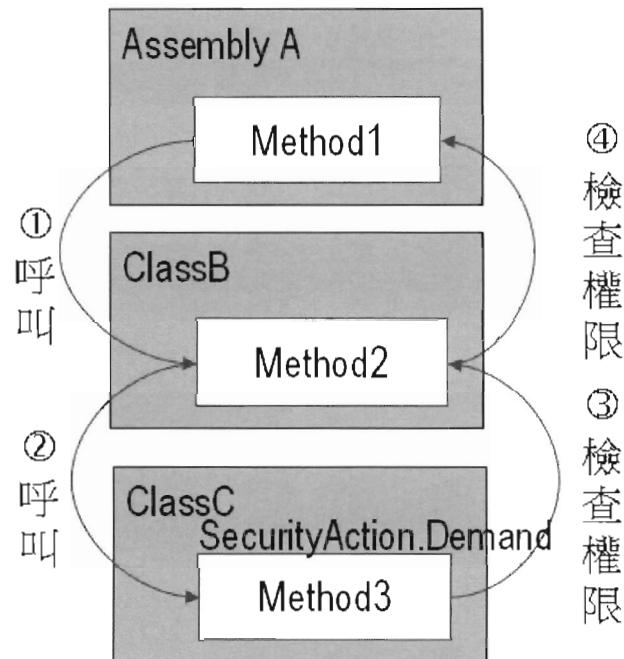
對於開發人員來說，方法權限有非常多的選擇，開發人員應當視應用程式的需求進行適當的選擇，以下有幾點建議：

- 使用 `SecurityAction.PermitOnly` 宣告限制方法可用的權限，使用 `PermitOnly` 逐一列表方法中所需的權限。
- 使用 `SecurityAction.Deny` 進一步的宣告防止方法存取拒絕使用的權限。
- 在方法區段中需要較少的權限時，使用命令式的 `CodeAccessPermission.PermitOnly` 以減少可用的權限，之後再使用 `CodeAccessPermission.RevertPermitOnly` 恢復原來的權限。
- 當你想允許程式呼叫部份信任的程式 (Partially Trusted Code) 的方法時需要某些權限，但確可能缺少該權限，可以使用 `CodeAccessPermission.Assert` 取得該權限的資源。不過使用此功能要小心，因為有可能讓程式出現潛在的弱點。在使用完該資源之後可以呼叫 `CodeAccessPermission.RevertAssert` 恢復原來的權限。
- 當組件中呼叫一些非.NET Framework 中的功能，像是 Unmanaged Code (COM 物件或 Win32 API) 可以使用 `CodeAccessPermission.Demand`。

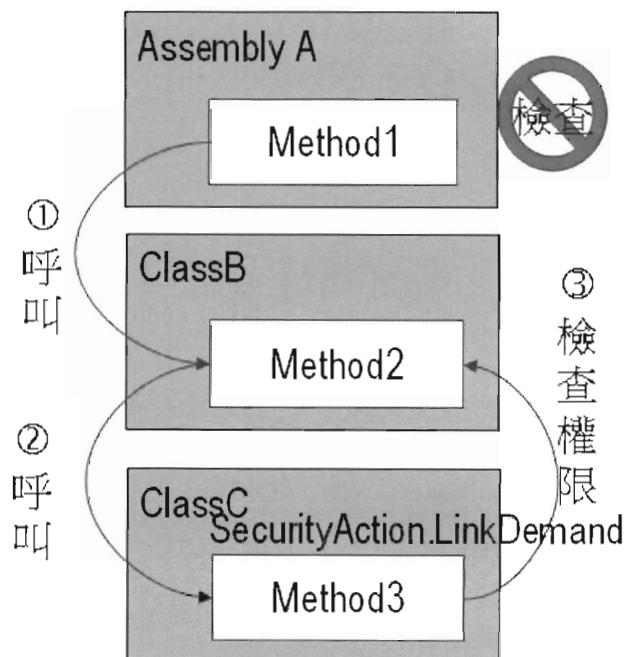


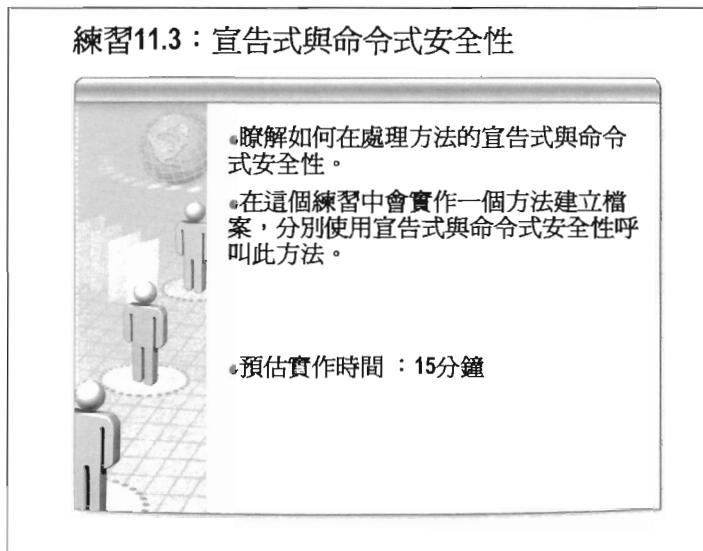
## Demand 與 LinkDemand 的相異性

Demand 與 LinkDemand 二者在檢查的過程上是有所不同，Demand 會去驗證所有堆疊中的呼叫端的權限，LinkDemand 則是只會驗證直接呼叫端的權限。如下圖，AssemblyA 的 Method1 呼叫 ClassB 的 Method2，ClassB 的 Method2 呼叫 ClassC 的 Method3，Method3 設定為 SecurityAction.Demand，在執行 Method3 時會去檢查 ClassB 的 Method2 及 AssemblyA 的 Method1 的權限。



相同的情況，如果 ClassC 的 Method3 設為  
SecurityAction.LinkDemand，則只會檢查 ClassB 的 Method2 的權限，  
而不會去檢查 AssemblyA 的 Method1 的權限。





### 練習 11.3：宣告式與命令式安全性

#### 目的：

瞭解如何在處理方法的宣告式與命令式安全性。

#### 功能描述：

在這個練習中會實作一個方法建立檔案，分別使用宣告式與命令式安全性呼叫此方法。

#### 預估實作時間：15 分鐘

#### 實作步驟：

- 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
- 建立一個新的 Windows Application 專案，將專案命名為 Mod11\_3。
- 開啟 Form1 的設計視窗，從「Toolbox」拖拉控制項表單並設定屬性如下：

控制項	屬性	值
-----	----	---

Button	ID	Button1
	Text	使用宣告式
Button	ID	Button2
	Text	使用命令式

4. 進入 Form1 的程式碼視窗，加入必要的命名空間。

```
Visual Basic
Imports System.Security.Permissions
Imports System.IO
```

```
C#
using System.Security.Permissions;
using System.IO;
```

5. 撰寫方法 CreateFile 在 Form1 類別，並限定此方法的 SecurityAction 為 Demand。

```
Visual Basic
<FileIOPermission(SecurityAction.Demand)> _
Sub CreateFile ()
    Using sw As StreamWriter = File.CreateText("c:\test\b.txt")

    End Using
    MessageBox.Show("建立c:\test\b.txt ")
End Sub
```

```
C#
[FileIOPermission(SecurityAction.Demand)]
void CreateFile ()
{
    using (StreamWriter sw = File.CreateText(@"c:\test\b.txt"))
    {
    }
    MessageBox.Show(@"建立c:\test\b.txt ");
}
```

6. 在 Button1\_Click 事件，使用宣告式允許寫入 C:\Test 目錄，呼叫 CreateFile 方法。

```
Visual Basic
<FileIOPermission(SecurityAction.Demand, Write:="C:\Test")> _
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
```

```
CreateFile()
End Sub
```

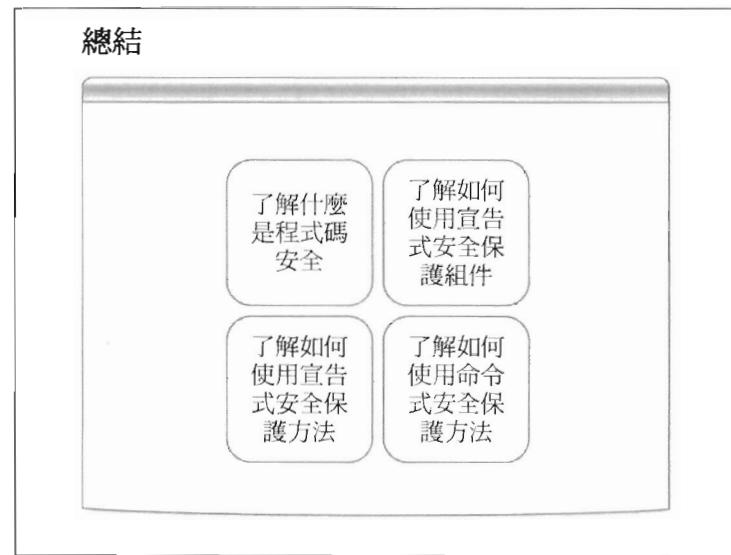
```
C#
[FileIOPermission(SecurityAction.Demand, Write=@"C:\Test")]
private void Button1_Click(object sender, EventArgs e)
{
    CreateFile();
}
```

7. 在 Button2\_Click 事件，使用命令式允許寫入 C:\Test 目錄，呼叫 CreateFile 方法。

```
Visual Basic
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    Dim fp As New FileIOPermission(FileIOPermissionAccess.Write, "c:\test")
    fp.Demand()
    CreateFile()
End Sub
```

```
C#
private void Button2_Click(object sender, EventArgs e)
{
    FileIOPermission fp =
        new FileIOPermission(FileIOPermissionAccess.Write, @"c:\test");
    fp.Demand();
    CreateFile();
}
```

8. 執行應用程式，按下「Button1」及「Button2」應該都可以順利建立檔案。
9. 將 SecurityAction 改成 Deny，則執行結果都會出現 SecurityException，因為拒絕存取 c:\Test 目錄。



## 總結

在這個章節你應該要熟悉什麼是程式碼存取安全，是由多項安全要素組合而成 Evidence、權限、權限集、安全原則及程式群組。同時你必須知道如何使用.NET Framework 組態工具定義程式碼安全原則。

另一方面，程式碼中有宣告式與命令式安全性，如何利用宣告式安全進行組件的保護作業，以及如何使用宣告式與命令式安全性進行方法 (Method) 的保護作業，都是本章必須學會的內容。

# 第十二章：驗證與授權

## 本章大綱

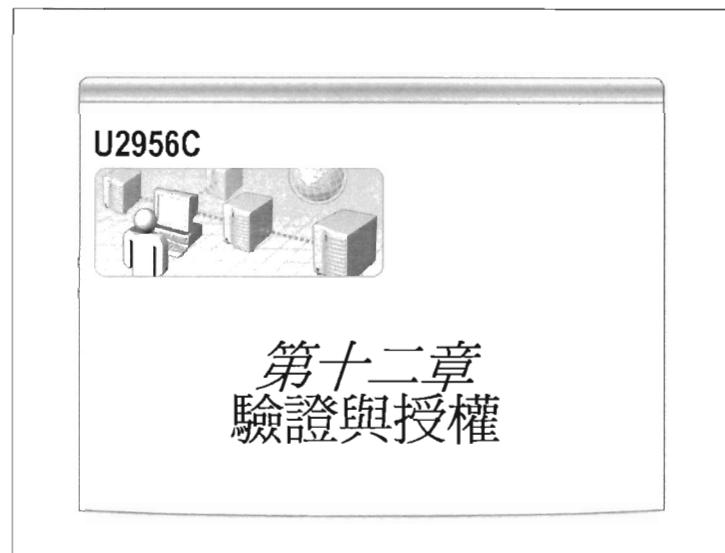
驗證及授權.....	4
認識驗證及授權.....	5
使用 Role-Based Security.....	7
Windows 驗證使用 RBS.....	9
練習 12.1：Windows 驗證使用 RBS .....	12
自行驗證使用 RBS.....	15
執行安全檢查 .....	20
練習 12.2：自行驗證使用 RBS .....	22
使用 ACL 保護檔案系統.....	27
認識是 DACL.....	28
認識是 SACL.....	30
使用.NET Framework 的存取控制類別.....	33
存取 SID 資訊.....	35
變更存取控制項目 .....	37
練習 12.3：設定檔案保留繼承 .....	39

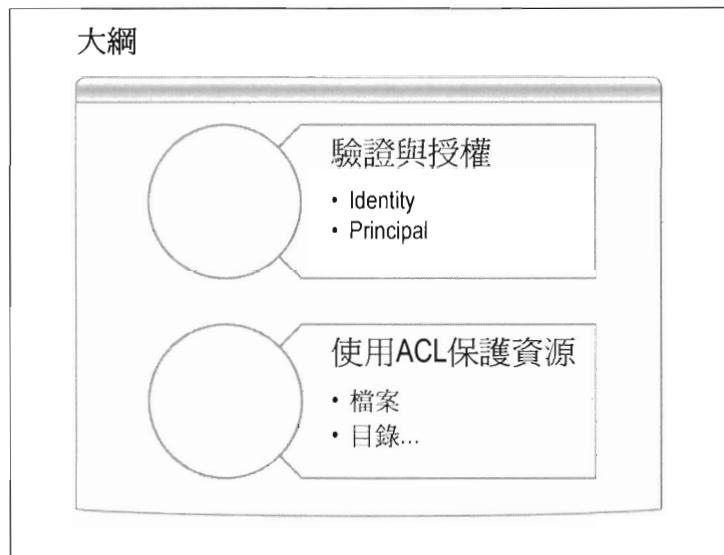
提昇您專業競爭實力的最佳夥伴

作者：

羅慧真





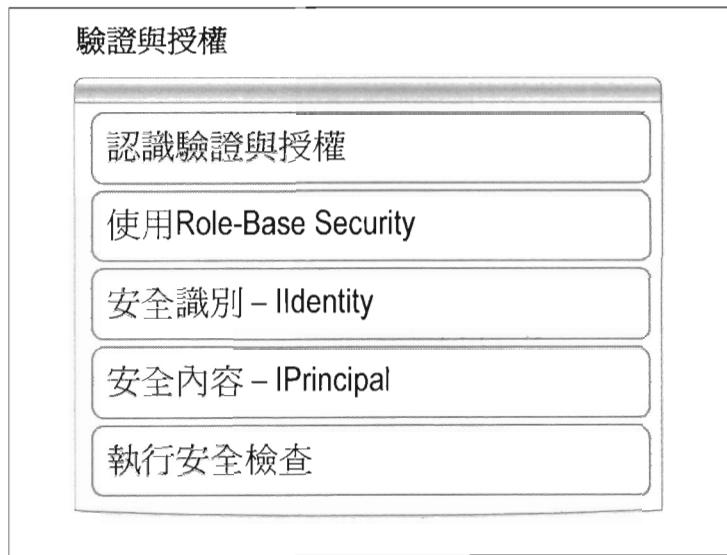


---

在這個章節中將介紹如何在.NET 應用程式中進行角色授權的程式設計，另外你也將學會如何使用.NET 所提供的 ACL 相關類別來進行保護資源，這些資源像是檔案、目錄、註冊機碼...等。

本章介紹以下主題：

- 驗證與授權
  - Identity
  - Principal
- 使用 ACL 保護資源
  - 檔案
  - 目錄...

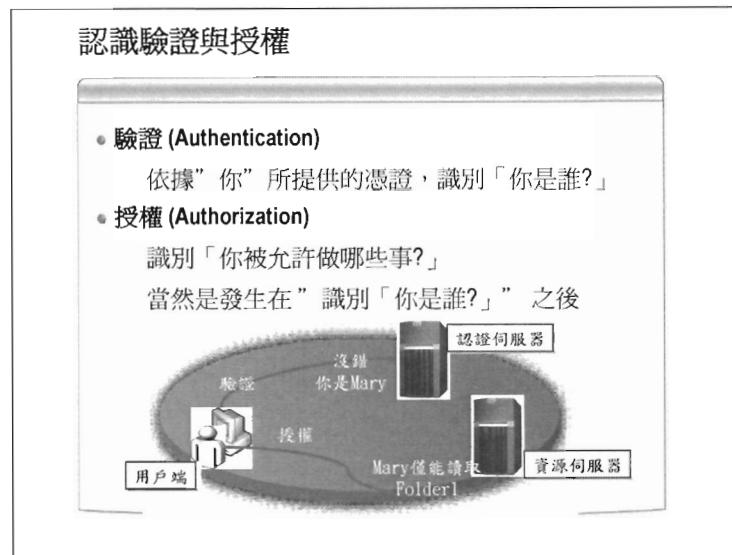


---

## 驗證及授權

這一節我們將介紹什麼是驗證以及什麼是授權的概念，並且了解如何利用.NET Framework 使用角色安全管理(Role-Based Security:簡稱 RBS)。

- 認識驗證與授權
- 使用 Role-Based Security
- 安全識別 – IIdentity
- 安全內容 – IPrincipal
- 執行安全檢查



## 認識驗證及授權

什麼是驗證 (Authentication)?

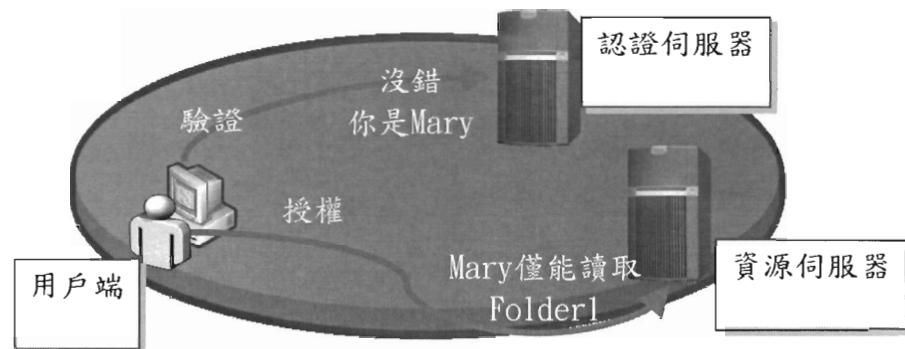
驗證就是向應用程式提出使用者憑證讓應用程式進行身份確認的作業。在存取系統資源之前，應用程式或系統都必須先辨識使用者對象方能決定是否允許存取，而這個辨識使用者的動作就稱之為驗證。

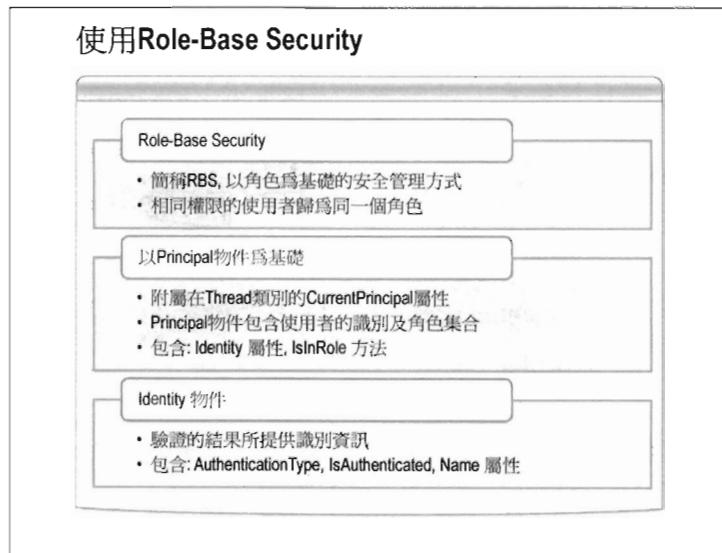
什麼是授權 (Authorization)?

授權必須發生在驗證之後，確認使用者身份之後，使用者要進行資源的存取時，系統或應用程式會進行檢查確認使用者有存取資源的權限之後，才授予進行資源存取的權力，這個過程稱之為授權。

例如，下面這張圖是使用者 Mary 在進入系統之前先使用帳號與密碼向網域的認證伺服器進行驗證程序，完成驗證程序之後，使用者的電腦會取得存取票證 (Access Token)，存取票證 (Access Token) 包含使用者的身份識別及所屬群組。

接著使用者開始存取網路上的資源，資源在被存取前會進行授權檢查，使用者的電腦會提出使用者的存取票證 (Access Token)，被要求的資源伺服器會進行存取票證 (Access Token) 的查驗確認使用者有權力存取該資源才提供資源。





## 使用 Role-Base Security

Role-Base Security 簡稱 RBS，是一種以角色為基礎的安全管理方式。有別於 CAS (Code Access Security) 是以呼叫堆疊為檢查安全啓動的作法。

RBS 的程序是使用者經過驗證之後，取得使用者識別(Identity)及使用者所屬角色(Roles)，在進行存取資源時檢查使用者所屬的角色進行權限的授與。

而角色則是，將使用者做群組管理，將有相同權限的使用者歸為同一個角色，以簡化程式授權的程序。

### Principal 物件

RBS 的作業方式是以 Principal 物件為基礎，Principal 物件包含使用者識別資訊及所屬角色的集合，並且附屬於 Thread 類別之下，當程式需要檢查 RBS 時，可經由 Thread 類別的 CurrentPrincipal 取得 Principal 物件，以進行角色安全檢查。Principal 物件是實作自 IPrincipal 介面，包含兩個成員：

- Identity 屬性，使用者的身份資訊。

- `IsInRole` 方法，傳入群組名稱，回傳 Boolean 值，代表使用者是否屬於該群組。

## Identity 物件

Identity 物件是通過驗證之後所取得識別資訊，實作自 `IIdentity` 介面，包含以下三個屬性：

- `AuthenticationType`，驗證類型。
- `IsAuthenticated`，是否通過驗證。
- `Name`，識別身份者的名字。



## Windows 驗證使用 RBS

應用程式的驗證方式使用 Windows 驗證時，可以使用 WindowsIdentity 及 WindowsPrincipale 兩個類別進行以角色為基礎的安全性<sup>4</sup>

### WindowsIdentity

提供 Windows 登入的帳號、驗證類別以及存取票證 (Access Token)。屬於 System.Security.Principal 命名空間，包含方法如下：

- GetAnonymous()，傳回匿名的 Windows 帳號 (型別 WindowsIdentity)
- GetCurrent()，傳回目前的 Windows 帳號 (型別 WindowsIdentity)
- Impersonate()，讓程式模擬不同的 Windows 帳號。
- Group 屬性，取得使用者所屬的群組。

### 呼叫 WindowsIdentity 的 GetCurrent 方法

如果 WindowsPrincipal 只使用一次，可以使用這種方式建立 WindowsPrincipal 物件。建立物件時在函式傳入目前登入者的身份識別 (WindowsIdentity)。

使用 IsInRole 方法時若為作業系統內建的群組，可以使用 WindowsBuiltInRole 列舉常數或”BUILTIN\內建群組名稱”。一般自訂群組則是”電腦名或網域名稱\群組名稱”。

#### Visual Basic

```
Dim iden As WindowsIdentity = WindowsIdentity.GetCurrent()
Dim prin As New WindowsPrincipal(iden)

listBox1.Items.Add("使用者: " + iden.Name)
listBox1.Items.Add("AuthenticationType:" + iden.AuthenticationType)
listBox1.Items.Add("Group: ")
For Each group As IdentityReference In iden.Groups
    Dim ntGroup As NTAccount = CType(group.Translate(GetType(NTAccount)), NTAccount)
    listBox1.Items.Add(vbTab + ntGroup.Value)
Next
listBox1.Items.Add("管理群組?" & prin.IsInRole(WindowsBuiltInRole Administrator))
```

#### C#

```
WindowsIdentity iden = WindowsIdentity.GetCurrent();
WindowsPrincipal prin = new WindowsPrincipal(iden);

listBox1.Items.Add("使用者: " + iden.Name);
listBox1.Items.Add("AuthenticationType:" + iden.AuthenticationType);
listBox1.Items.Add("Group: ");
foreach (IdentityReference group in iden.Groups)
{
    NTAccount ntGroup = (NTAccount)group.Translate(typeof(NTAccount));
    listBox1.Items.Add("\t" + ntGroup.Value );
}
listBox1.Items.Add("管理群組?" + prin.IsInRole(WindowsBuiltInRole Administrator));
```

### 設定應用程式定義域的 Principal 原則為 PrincipalPolicy.WindowsPrincipal

若是在應用程式使用期間會使用到多次，可以將目前的應用程式定義域 Principal 原則設定為 PrincipalPolicy.WindowsPrincipal，之後可以從 Thread 的 CurrentPrincipal 取得 Principal。

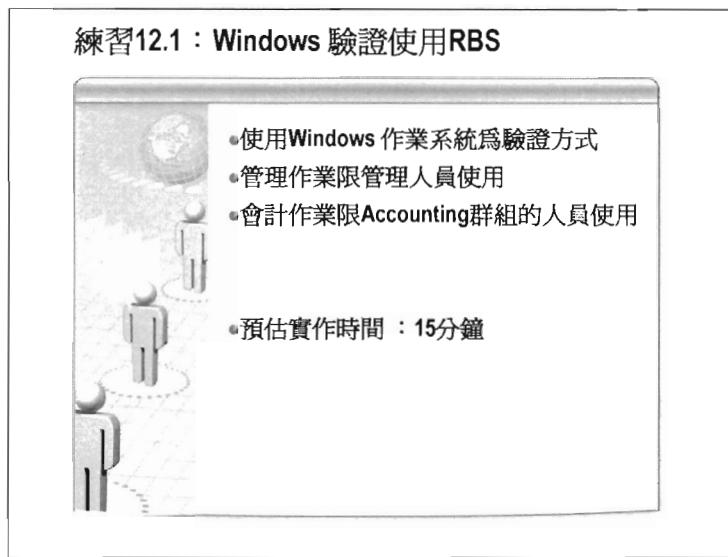
#### Visual Basic

```
AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.Wind
```

```
    owsPrincipal)

Dim prin As WindowsPrincipal
prin = CType(Thread.CurrentPrincipal, WindowsPrincipal)
TextBox5.Text = IIf(prin.IsInRole("BUILTIN\Administrators"), "是",
", "否")
TextBox6.Text = IIf(prin.IsInRole("BUILTIN\PowerUser"), "是",
"否")
TextBox7.Text = IIf(prin.IsInRole("WS2003\SD"), "是", "否")
```

```
C#
AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.Wind
owsPrincipal);
WindowsPrincipal prin = (WindowsPrincipal)Thread.CurrentPrincip
al;
TextBox5.Text = prin.IsInRole(@"BUILTIN\Administrators")? "是"
: "否";
TextBox6.Text = prin.IsInRole(@"BUILTIN\PowerUser")? "是" :
否";
TextBox7.Text = prin.IsInRole(@"WS2003\SD")? "是" : "否";
```



## 練習 12.1：Windows 驗證使用 RBS

目的：

瞭解如何使用 Windows 作業系統為驗證方式進行角色為基礎的安全(RBS)檢查。

功能描述：

在這個練習應用程式必須限制「管理作業」限管理人員使用，「會計作業」限 Accounting 群組的人員使用。

預估實作時間：20 分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod12\_1。
3. 設定應用程式驗證方式為 Windows。

- Visual Basic 請在選單選取「Project」→「Mod12\_1屬性」，在「Application」頁籤中找到「Authentication Mode」確認是否為「Windows」(預設即是)。
- C# 者請開啟 Program.cs 在「Main」的程式中加上以下程式碼：

```
AppDomain.CurrentDomain.SetPrincipalPolicy(System.Security.Principal.PrincipalPolicy.WindowsPrincipal);
```

4. 開啓 Form1 加入兩個 Button 控制項，Text 屬性分別為「管理作業」及「會計作業」。
5. 匯入命名空間：

Visual Basic  
Imports System.Threading

C#  
using System.Threading;

6. 在 Form\_Load 事件顯示目前使用者名稱：

Visual Basic  
Me.Text = Thread.CurrentPrincipal.Identity.Name

C#  
this.Text = Thread.CurrentPrincipal.Identity.Name;

7. 在「管理作業」的按鈕檢查使用者是否為作業系統內建的管理群組。

Visual Basic  
If Thread.CurrentPrincipal.IsInRole("BUILTIN\Administrators") Then  
 MessageBox.Show("進行管理作業")  
Else  
 MessageBox.Show("非管理者請勿嘗試執行此功能")  
End If

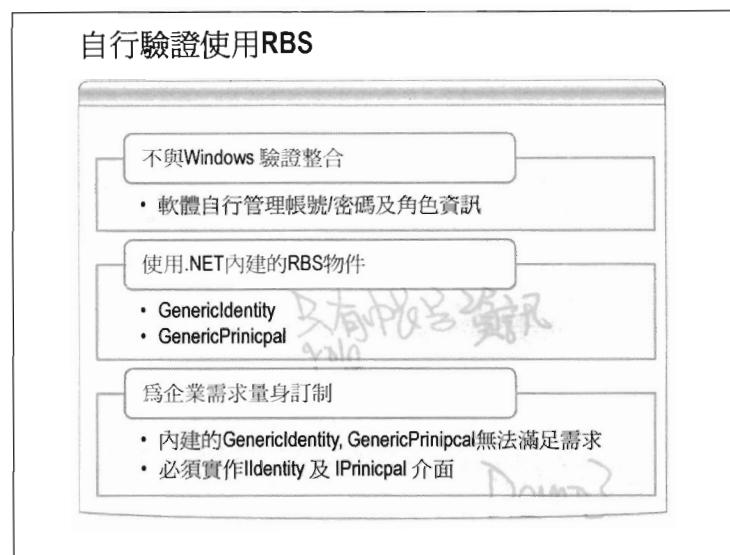
```
C#
if (Thread.CurrentPrincipal.IsInRole(@"BUILTIN\Administrators"))
    MessageBox.Show("進行管理作業");
else
    MessageBox.Show("非管理者請勿嘗試執行此功能");
```

8. 在「會計作業」的按鈕檢查使用者是否為 Accounting 群組。

```
Visual Basic
Dim roleName As String
roleName = Environment.GetEnvironmentVariable("USERDOMAIN")
"") & "\Accounting"
If Thread.CurrentPrincipal.IsInRole(roleName) Then
    MessageBox.Show("進行會計作業")
End If
```

```
C#
String roleName;
roleName = Environment.GetEnvironmentVariable("USERDOMAIN")
") + "\\Accounting";
if (Thread.CurrentPrincipal.IsInRole(roleName) )
    MessageBox.Show("進行會計作業");
else
    MessageBox.Show("必須是會計組人員才能使用此功能");
```

9. 在系統建立一個「Accounting」群組及「User1」帳號，並將「User1」帳號加入「Accounting」群組。
10. 測試程式分別執行「管理作業」及「會計作業」。如果使用管理群組帳號，那麼將可進行「管理作業」，如果使用 Accounting 群組的帳號，那麼可進行「會計作業」。



## 自行驗證使用 RBS

應用程式不一定會與 Windows 作業系統的驗證方式整合，而是自行撰寫驗證程式，並且將使用者及所屬角色等資訊存於其他儲存體，例如資料庫。

### 使用.NET 內建的 RBS 物件

程式人員必須在軟體中撰寫驗證程式，並且取得使用者的識別建立 GenericIdentity 與 GenericPrincipal。

```
Visual Basic
dim iden as new GenericIdentity(userName)
dim prin as new GenericPrincipal(iden, roles)
Thread.CurrentPrincipal = prin
```

```
C#
GenericIdentity iden = new GenericIdentity(userName);
GenericPrincipal prin = new GenericPrincipal(iden, roles);
Thread.CurrentPrincipal = prin;
```

## 自訂 Identity 類別

.NET Framework 提供的身份識別物件有 WindowsIdentity 及 GenericIdentity 二者。

- WindowsIdentity 為作業系統登入的身份識別物件。
- GenericIdentity 使用在非作業系統登入的身份識別物件。 GenericIdentity 類別只提供一般性的身份識別屬性，像是 AuthenticationType、IsAuthenticated、Name。

企業需要更多的屬性來做為身份識別用，那麼 GenericIdentity 類別將不夠使用，此時可以實作 IIdentity 介面自訂 Identity 類別。IIdentity 介面定義必須實作三個屬性：

- AuthenticationType，驗證類型。
- IsAuthenticated，是否通過驗證。
- Name，識別身份者的名字。

以下範例是實作 IIdentity 介面及其必要實作的屬性。

```

Imports System.Security.Principal

Public Class MyIdentity
    Implements IIdentity

    Public ReadOnly Property AuthenticationType() As String Implements System.Security.Principal.IIdentity.AuthenticationType
        Get
            Return Me._authenticationType
        End Get
    End Property

    Public ReadOnly Property IsAuthenticated() As Boolean Implements System.Security.Principal.IIdentity.IsAuthenticated
        Get
            Return Me._isAuthenticated
        End Get
    End Property

    Public ReadOnly Property Name() As String Implements System.Security.Principal.IIdentity.Name
        Get
            Return Me._name
        End Get
    End Property
End Class

```

```
C#
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Principal;
namespace CustomRBS
{
    class MyIdentity : IIdentity
    {
        #region IIdentity Members

        public string AuthenticationType
        {
            get {
                return this._authenticationType;
            }
        }

        public bool IsAuthenticated
        {
            get {
                return this._isAuthenticated;
            }
        }

        public string Name
        {
            get {
                return this.Name;
            }
        }

        #endregion
    }
}
```

## 自訂 Principal 類別

.NET Framework 提供的 Principal 物件有 WindowsPrincipal 及 GenericPrincipal 二者。

- WindowsPrincipal 為作業系統登入的安全性內容包含識別身份以是否屬於某角色成員。
- GenericPrincipal 使用在非作業系統登入的安全性內容。GenericPrincipal 類別只提供一般性的安全性內容，像是 IsInRole 方法及 Identity 屬性。

企業需要更多的屬性來做為安全性內容，那麼 GenericPrincipal 類別將不夠使用，此時可以實作 IPrincipal 介面自訂 Principal 類別。IPrincipal 介面定義必須實作兩個成員：

- Identity 屬性，使用者的識別資訊。
- IsInRole 方法，提供檢查使用者是否屬於某個角色的功能。

以下範例是實作 IPrincipal 及其必要實作的成員：

```
Visual Basic
Imports System.Security.Principal
Public Class MyPrincipal
    Implements IPrincipal
    Public ReadOnly Property Identity() As System.Security.Principal.IIdentity Implements System.Security.Principal.IPrincipal.Identity
        Get
            Return _identity
        End Get
    End Property

    Public Function IsInRole(ByVal role As String) As Boolean Implements System.Security.Principal.IPrincipal.IsInRole
        If Array.BinarySearch(_roles, role) >= 0 Then
            Return True
        Else
            Return False
        End If
    End Function
```

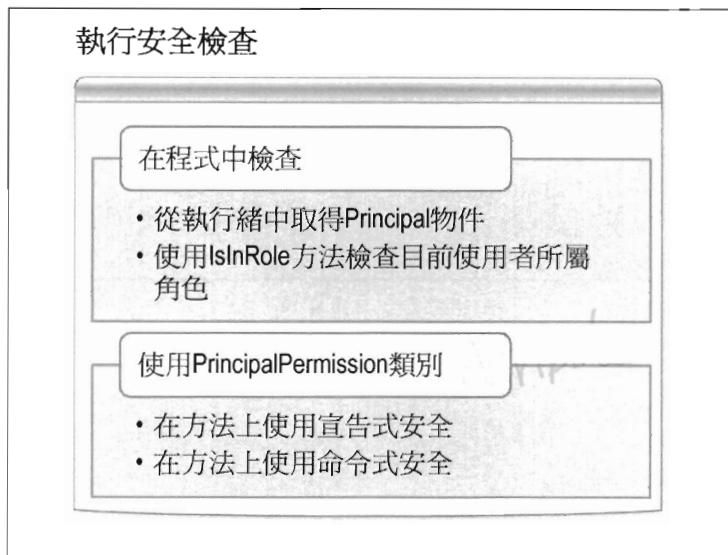
```
C#
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Principal;
namespace CustomRBS
{
    class MyPrincipal : IPrincipal
    {
        #region IPrincipal Members

        public IIdentity Identity
        {
            get {
                return _identity;
            }
        }

        public bool IsInRole(string role)
```

```
{   if (Array.BinarySearch(_roles, role) >= 0 )
    return true;
else
    return false;
}

#endregion
}
```



## 執行安全檢查

PrincipalPermission 可以用來對方法進行宣告式角色安全性角色安全性。它所屬的命名空間是 System.Security.Permissions。

在方法之前使用 PrincipalPermission Attribute 並指定允許存取的角色。以下範例是允許管理者存取 AdminsOnly 方法。

```
Visual Basic
<PrincipalPermission(SecurityAction.Demand, ROLE:="BUILTIN\Administrators")>
Sub AdminsOnly()
    MessageBox.Show("管理者專有的方法")
End Sub
```

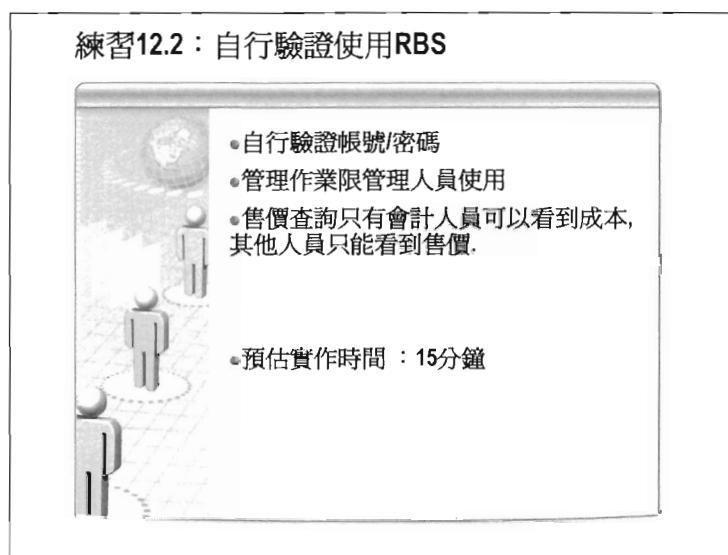
```
C#
[PrincipalPermission(SecurityAction.Demand, Role=@"BUILTIN\Administrators")]
void AdminsOnly()
{
    MessageBox.Show("管理者專有的方法");
}
```

以下範例允許所有驗證過的使用者存取 AnyLogonUser 方法 ~

```
Visual Basic
```

```
<PrincipalPermission(SecurityAction.Demand, authenticated:=True)>
Sub AnyLogonUser()
    MessageBox.Show("登入使用者可用的方法")
End Sub
```

```
C#
[PrincipalPermission(SecurityAction.Demand, Authenticated=true)]
void AnyLogonUser()
{
    MessageBox.Show("登入使用者可用的方法");
}
```



## 練習 12.2：自行驗證使用 RBS

目的：

瞭解如何在自行進行驗證程序的狀況下使用以角色為基礎的安全性 (RBS)。

功能描述：

在這個練習包含一個登入表單，在登入表單處理驗證程序。這個應用程式只允許通過驗證的人員使用，其中「管理作業」限管理人員使用，「售價查詢」只有會計人員可以看到售價及成本，其他人員只能看到售價。

預估實作時間：20 分鐘

實作步驟：

1. 從『Practices\VB 或 CS\Mod12\_2\Starter\Mod12\_2』開啟 Mod12\_2.sln 檔案。

2. Visual Basic 設定專案的驗證模式：

Visual Basic 請在選單選取『Project』→『Mod12\_1 屬性』，在『Application』頁籤中找到『Authentication Mode』確認是否為『Application-defined』。

3. 開啓登入表單 LoginForm 的程式碼視窗，找到「TODO: 指定 Thread.CurrentPrincipal」，在此指定 Thread 的 CurrentPrincipal。

Visual Basic

```
'TODO: 指定Thread.CurrentPrincipal
Dim iden As New GenericIdentity(UsernameTextBox.Text)
Thread.CurrentPrincipal = New GenericPrincipal(iden, roles)
```

C#

```
//TODO: 指定Thread.CurrentPrincipal
GenericIdentity iden = new GenericIdentity(UsernameTextBox.Text);
Thread.CurrentPrincipal = new GenericPrincipal(iden, roles);
```

4. 開啓 Form1，在 Form\_Load 事件呼叫 LoginForm，並檢查使用者是否通過驗證。如果未通過驗證，結束程式。如果通過驗證在表單的標題列顯示使用者的帳號。

Visual Basic

```
LoginForm.ShowDialog()
If Not Thread.CurrentPrincipal.Identity.IsAuthenticated Then
    MessageBox.Show("請先登入!!")
    Me.Close()
End If

Me.Text = "使用者:" + Thread.CurrentPrincipal.Identity.Name
```

C#

```
using (LoginForm login = new LoginForm ())
{
    login.ShowDialog();
}
if (!Thread.CurrentPrincipal.Identity.IsAuthenticated )
{
    MessageBox.Show("請先登入!!");
    this.Close();
}

this.Text = "使用者:" + Thread.CurrentPrincipal.Identity.Name;
```

5. 找到 AdminOnly 方法，這個方法只限 Administrators 角色的人員使用。

Visual Basic

```
<PrincipalPermission(SecurityAction.Demand, ROLE:="Administrators")> _
```

```
Sub AdminsOnly()
```

```
C#
[PrincipalPermission(SecurityAction.Demand, Role = "Administrators")]
void AdminsOnly()
```

6. 在 Button2 的 Click 事件，找到「TODO: 限會計人員可以查詢成本」，在此顯示成本\$150，但成本只限會計人員查看。

Visual Basic

```
'TODO: 限會計人員可以查詢成本
If Thread.CurrentPrincipal.IsInRole("Accounting") Then
    msg &= vbTab & "成本$150"
End If
```

C#

```
//TODO: 限會計人員可以查詢成本
if ( Thread.CurrentPrincipal.IsInRole("Accounting") )
    msg += "\t成本$150";
```

7. 執行應用程式，使用帳號「Admin」，結果如下：





8. 執行應用程式，使用帳號「User1」，結果如下：





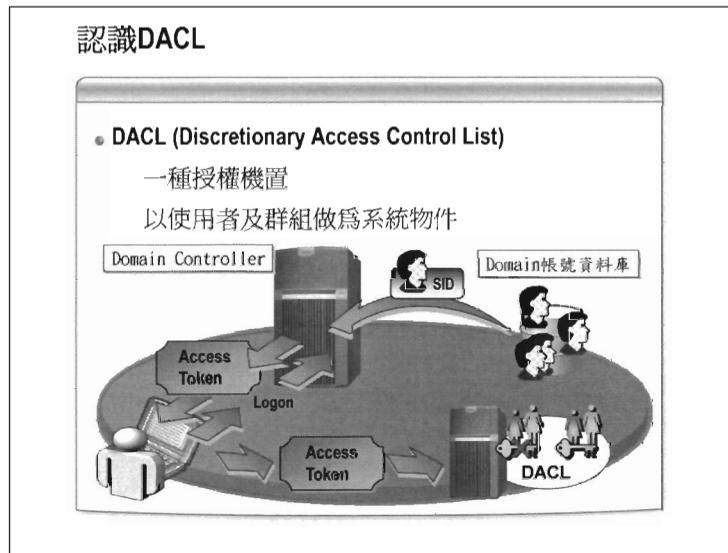


## 使用 ACL 保護檔案系統

這一節我們將介紹什麼是 ACL、DACL、SACL，以及在程式之中如何設定或者讀取檔案、目錄的 ACL。

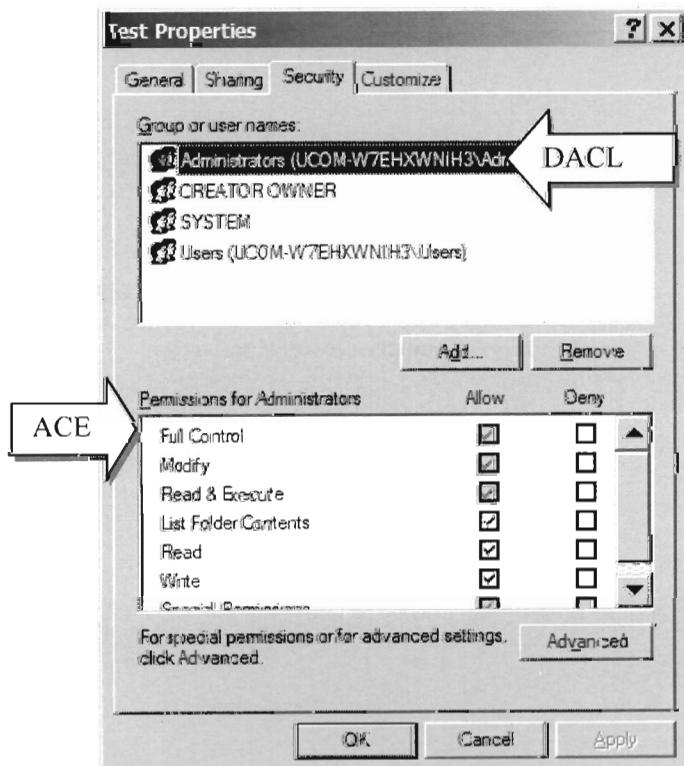
這一節的主題如下：

- 認識 DACL
- 認識 SACL
- 使用.NET Framework 的存取控制類別
- 使用 Security Descriptor
- 讀取存取控制資訊
- 變更存取控制項目



## 認識是 DACL

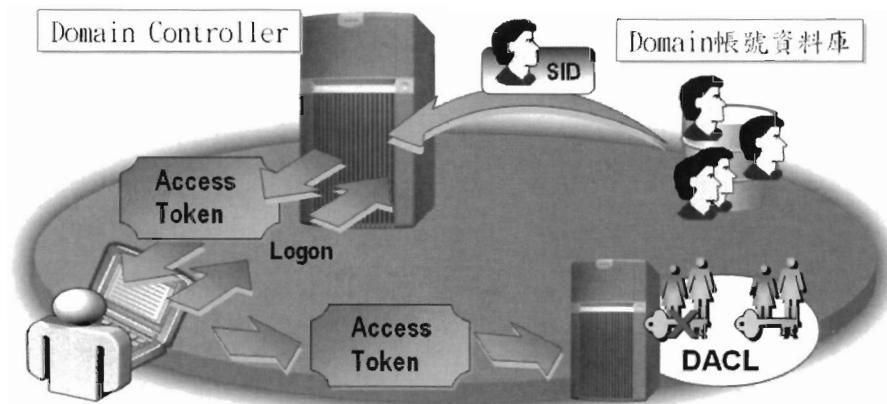
說到 DACL (Discretionary Access Control List) 可能大家都不太知道是什麼，不過看到下面這個畫面你可能會覺得相當的熟悉。



DACL (Discretionary Access Control List) 是一種授權 (Authorization) 機制，將系統物件 (例如，檔案、目錄) 授予使用者或群組可否存取的能力。

ACE (Access Control Entry) 是安全物件的控制存取或監控存取。例如，檔案是否「允許」進行「修改」，就是一個 ACE。DACL (Discretionary Access Control List) 是由多個 ACE 組合而成。例如 Administrators 群組具有允許完全控制、修改、讀取及執行...等權利。

使用者要使用系統資源時，必須先進行登入到電腦或網域的動作，如果使用者身份無誤，系統會給使用者一份存取票證 (Access Token)，裡面包含帳號的唯一識別碼、所屬群組的識別碼。接著使用者進行網路資源存取，這份存取票證會被送到資源所在的那台電腦進行授權檢查，授權檢查是將存取票證與 DACL 進行比對，以確認使用者是否具有存取資源的權利。





## 認識是 SACL

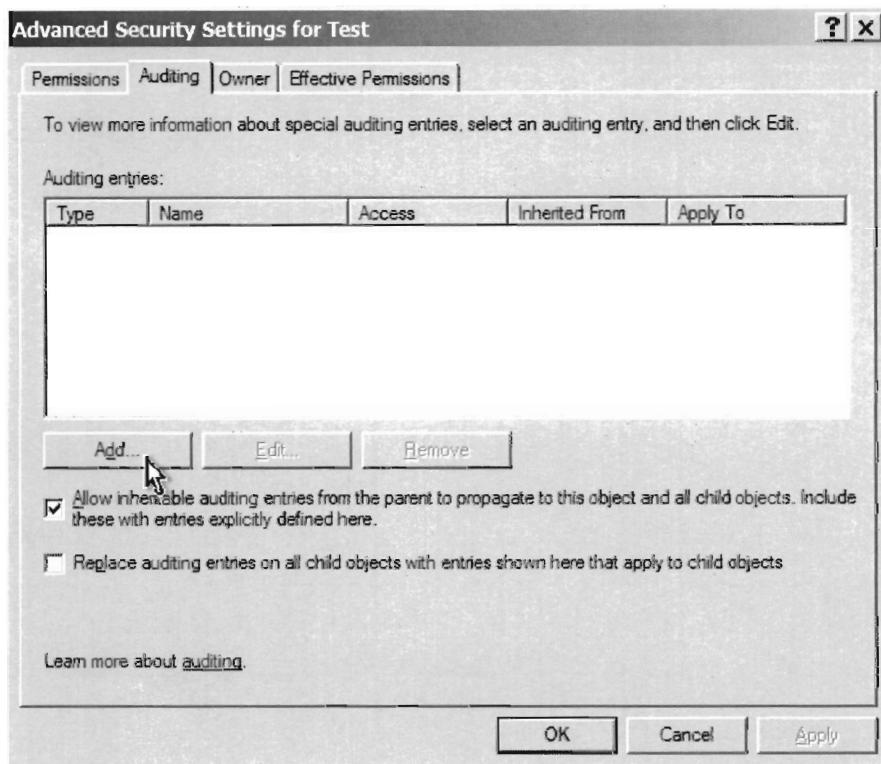
SACL (Security Access Control List) 與 DACL (Discretionary Access Control List) 類似，都是一種清單，也是由 ACE (Access Control Entry) 組合而成。差別是，它是一種事件記錄的機置，用來稽核人員對於資源物件(例如，檔案及目錄)存取狀況。

## 如何進行目錄的存取稽核

以下是進行目錄的存取稽核操作步驟：

1. 在目錄上按滑鼠右鍵選取「Properties」。
2. 在目錄的內容視窗中點選「Security」頁籤。點選「Advanced...」。
3. 在「Advanced」視窗中點選「Audit」頁籤。

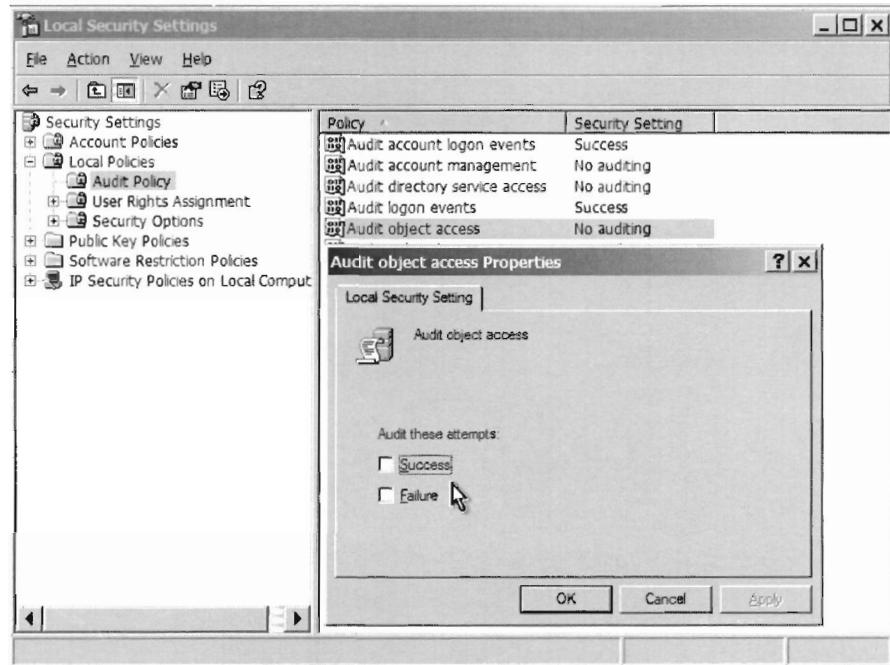
這樣便可以開始進行稽核項目的編修了。

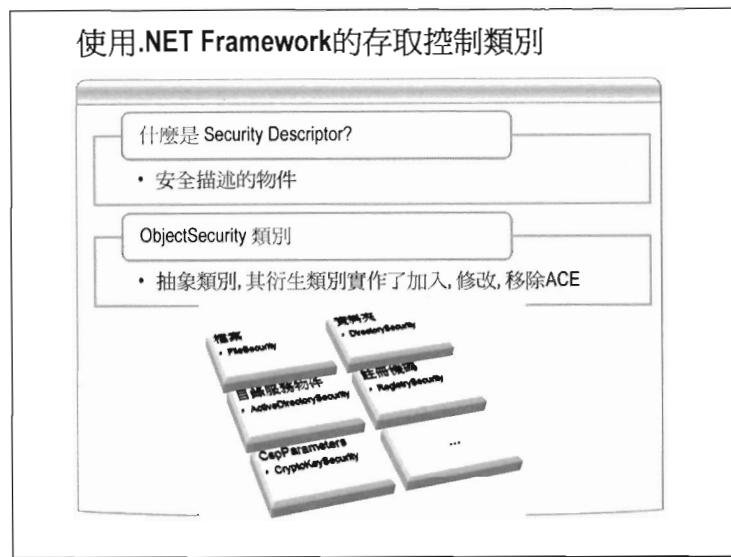


## 啓用稽核記錄

預設作業系統並沒有啓動稽核記錄，必須從「本機安全原則」項目進行啓動作業，步驟如下：

1. 選取『Administrative Tools』 → 『Local Security Policy』。
2. 在 Local Security Policy 工具的左邊區域展開『Local Policies』項目，點選『Audit Policy』。
3. 在 Local Security Policy 工具的右邊區域雙擊『Audit Object Access』項目，並啓用『Failure』或『Success』。





## 使用.NET Framework 的存取控制類別

Security Descriptor 簡稱 SD，是 Windows 作業系統對於資源物件的安全描述。例如，以檔案而言，每個檔案的安全頁籤中的 ACL 就是描述檔案物件的安全內容。

### ObjectSecurity 類別

.NET Framework 2.0 將 SD 的行為設計了 ObjectSecurity 抽象類別，它的衍生類別實作了加入、修改、移除 ACE 項目的功能。下表是資源類別與安全描述類別的對照，每個資源類別都有 GetAccessControl 及 SetAccessControl 可以取得或設定安全描述類別。這些安全描述類別都是繼承自 ObjectSecurity。

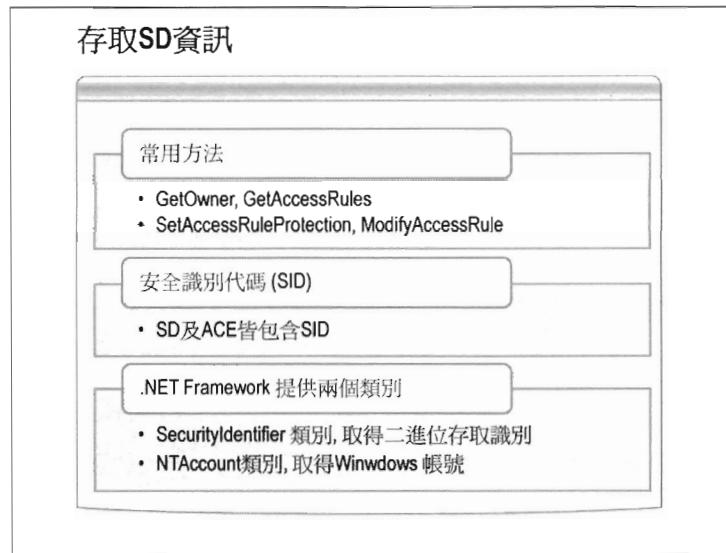
說明	資源類別	安全描述的類別 (SD)
包裝 Win32 檔案	File、FileInfo、 FileStream	FileSecurity
包裝 Win32 目錄	Directory、 DirectoryInfo	DirectorySecurity
包裝網域目錄服務節點	DirectoryEntry	ActiveDirectorySecurity
包裝註冊機碼鍵值	RegistryKey	RegistrySecurity

包裝加密服務提供者 (CSP: Cryptographic Service Provider)	CspParameters	CryptoKeySecurity
包裝 Win32 進行同步處理的物件	Mutex	MutexSecurity

## AccessRule 類別

AccessRule 類別也是抽象類別它的衍生類別提供物件的存取權限類型，例如 FileSystemAccessRule 類別提供指定的帳號或群組的讀取或寫入的權限。

安全描述的類別 (SD)	存取權限類型
FileSecurity, DirectorySecurity	FileSystemAccessRule
RegistrySecurity	RegistryAccessRule
CryptoKeySecurity	CryptoKeyAccessRule
MutexSecurity	MutexAccessRule



## 存取 SD 資訊

存取 SD 內容可以透過 ObjectSecurity 的衍生類別的以下幾個方法進行存取：

- GetOwner，取得物件的擁有者。
- GetGroup，取得物件擁有者所屬的群組資訊。
- GetAccessRules，取得物件的存取規則。
- SetAccessRuleProtection，設定存取規則的保護方式。
- ModifyAccessRule，修改存取規則。

在作業系統裡，所有安全管理的物件都有安全識別代碼(SID)，這些安全識別代碼可以用二進位的方式存取，它的類別是 SecurityIdentifier。如果它是一個 Windows 帳號則可以使用 NTAccount 類別，這兩個類別都有一個 Value 屬性，NTAccount 類別的 Value 會傳回 Windows 帳號名稱。

GetAccessRules 方法的三個參數分別是：

- includeExplicit，是否包含物件明確的存取規則。
- includeInherited，是否包含物件繼承的存取規則。
- targetType，擷取存取規則的安全識別項。

SetAccessRuleProtection 方法有兩個參數：

- isProtected，決定是否保護這個檔相關的存取規則及是否繼承。設為 True 代表要保護並避免繼承。
- preserveInheritance，決定是否保留繼承的存取規則。設為 True 代表要保留繼承的存取規則。

## 使用 GetAccessControl 取得 DACL

以下範例是使用 GetAccessControl 取得 c:\Test 目錄下 U2956.txt 檔案的 DACL。GetAccessControl 方法會回傳 FileSecurity 物件。呼叫 FileSecurity 物件的 GetAccessRules 方法可以取得安全物件的存取規則集合 (型別 AuthorizationRuleCollection)。

**Visual Basic**

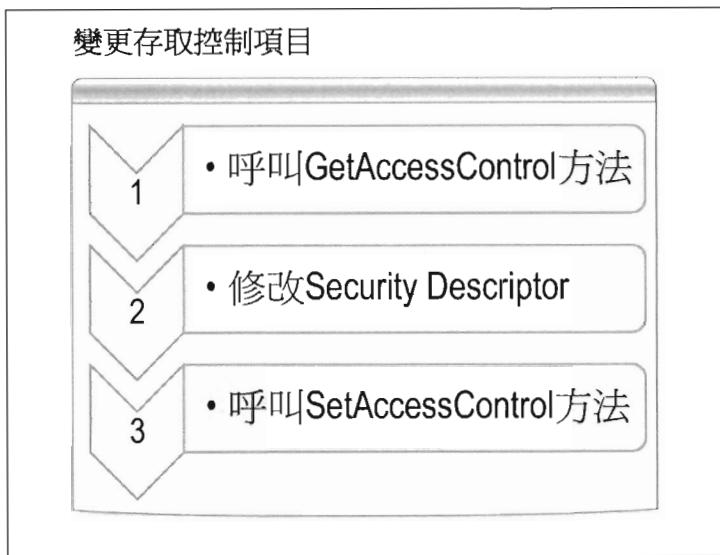
```
Dim sd As FileSecurity
sd = File.GetAccessControl(fileName)
ListBox1.Items.Clear()
ListBox1.Items.Add("Owner:" & sd.GetOwner(GetType(NTAccount)).Value)
ListBox1.Items.Add("Group:" & sd.GetGroup(GetType(NTAccount)).Value)

For Each ar As FileSystemAccessRule In sd.GetAccessRules(True,
True, GetType(NTAccount))
    ListBox1.Items.Add(ar.IdentityReference.Value & vbTab & ar.Fi-
leSystemRights.ToString() & vbTab & ar.AccessControlType.ToString())
Next
```

**C#**

```
FileSecurity sd = File.GetAccessControl(fileName);
ListBox1.Items.Clear();
ListBox1.Items.Add("Owner:" + sd.GetOwner(typeof(NTAccount)).Value);
ListBox1.Items.Add("Group:" + sd.GetGroup(typeof(NTAccount)).Value);

foreach (FileSystemAccessRule ar in sd.GetAccessRules(true, true,
typeof(NTAccount)))
{
    ListBox1.Items.Add(ar.IdentityReference.Value + "\t" +
ar.FileSystemRights.ToString() + "\t" +
ar.AccessControlType.ToString());
}
```



## 變更存取控制項目

若要修改物件的存取控制項目，它的步驟如下：

1. 呼叫 GetAccessControl 方法，先取得 ObjectSecurity 物件。
2. 使用 AccessRule 物件設定存取規則，呼叫 ObjectSecurity 物件的 ModifyAccessRule 方法修改 Security Descriptor。
3. 呼叫 SetAccessControl 方法存入設定。

以下範例是設定 User1 具有允許讀取檔案的權限：

```
Visual Basic
Dim sd As FileSecurity
sd = File.GetAccessControl(fileName)

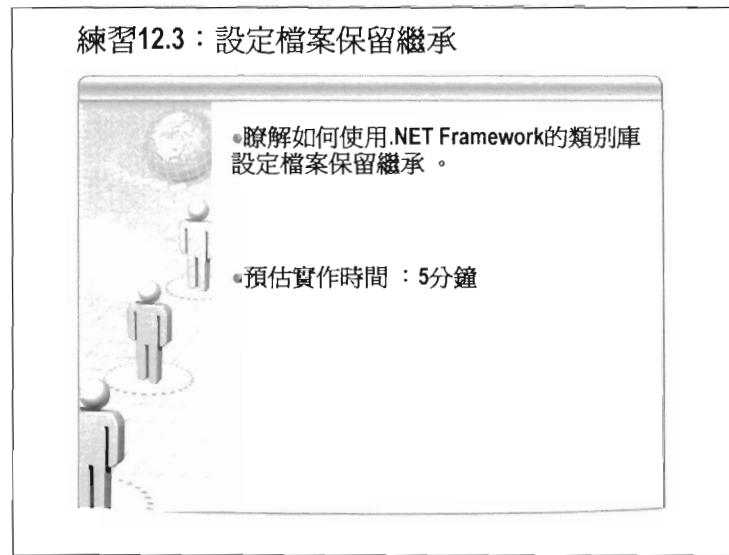
Dim account As String
account = Environment.GetEnvironmentVariable("USERDOMAIN")
& "\User1"
Dim ar As New FileSystemAccessRule(account, FileSystemRights.R
ead, AccessControlType.Allow)
Dim modify As Boolean
sd.ModifyAccessRule(AccessControlModification.Add, ar, modify)

File.SetAccessControl(fileName, sd)
MessageBox.Show("修改結果:" + modify)
```

```
C#
FileSecurity sd = File.GetAccessControl(fileName);

String account ;
account = Environment.GetEnvironmentVariable("USERDOMAIN")
+ @"\User1";
FileSystemAccessRule ar = new FileSystemAccessRule(account, Fil
eSystemRights.Read, AccessControlType.Allow);
Boolean modify ;
sd.ModifyAccessRule(AccessControlModification.Add, ar,out modif
y);

File.SetAccessControl(fileName, sd);
MessageBox.Show("修改結果:" + modify);
```



## 練習 12.3：設定檔案保留繼承

目的：

瞭解如何使用.NET Framework 的類別庫設定檔案保留繼承，這個練習會用到 FileSecurity 及 File 兩個類別。

功能描述：

在這個練習中會使用 Windows 表單讓使用者輸入要做檔案保留繼承的檔名，然後按下「設定」，設定指定檔案為保留繼承。

預估實作時間：5 分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod12\_3。
3. 在 Form1 放入 Label、TextBox 及 Button 控制項。
4. Label 控制項的 Text 設定為「檔案位置:」。

5. Button 控制項的 Text 設定為「設定:」。

6. 匯入必要命名空間：

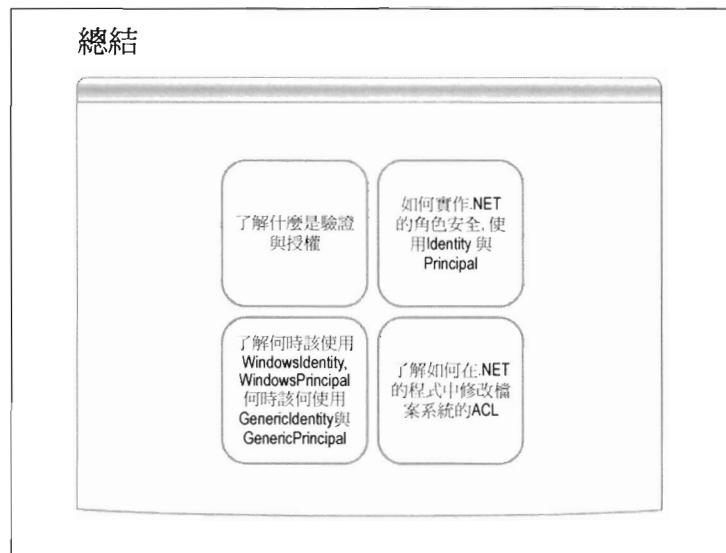
```
Visual Basic
Imports System.Security.AccessControl
Imports System.IO
```

```
C#
using System.Security.AccessControl;
using System.IO;
```

7. Button 控制項的 Click 事件設定 TextBox1 的 Text 屬性所指定的檔名為保留繼承：

```
Visual Basic
Dim path As String = textBox1.Text
Dim fs As New FileSecurity(path, AccessControlSections.All)
fs.SetAccessRuleProtection(True, True)
File.SetAccessControl(path, fs)
MessageBox.Show("ok")
```

```
C#
string path = textBox1.Text;
FileSecurity fs = new FileSecurity(path, AccessControlSections.All);
fs.SetAccessRuleProtection(true, true);
File.SetAccessControl(path, fs);
MessageBox.Show("ok");
```



# 第十三章：資料加密及 雜湊處理

## 本章大綱

加解密及雜湊演算法的運作概念.....	4
了解資料加密.....	5
對稱式金鑰.....	6
非對稱式金鑰.....	7
了解雜湊演算法.....	9
資料加解密的程式.....	11
.NET Framework 支援的加解密技術.....	12
使用對稱式金鑰加密.....	13
使用對稱式金鑰解密.....	16
練習 13.1：使用對稱式金鑰加/解密.....	18
非對稱式金鑰加/解密.....	25
練習 13.2：使用非對稱式金鑰加/解密.....	27
資料的雜湊處理.....	30
練習 13.3：使用雜湊式演算法.....	34

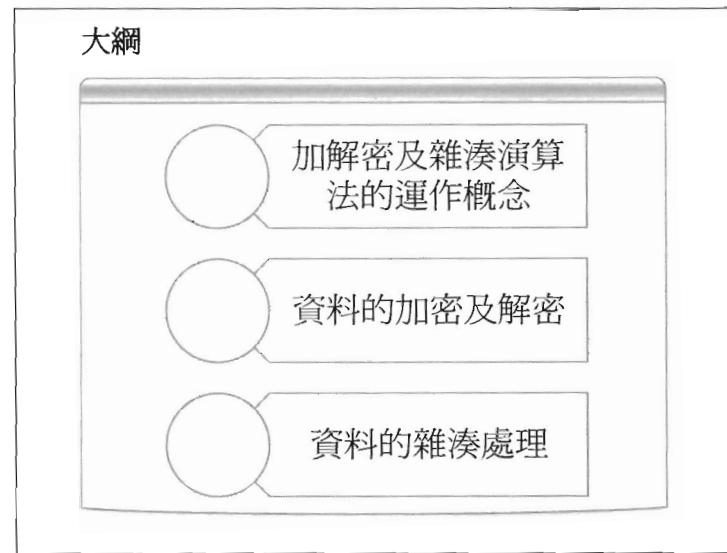
提升您專業競爭實力的最佳夥伴

作者：

羅慧真





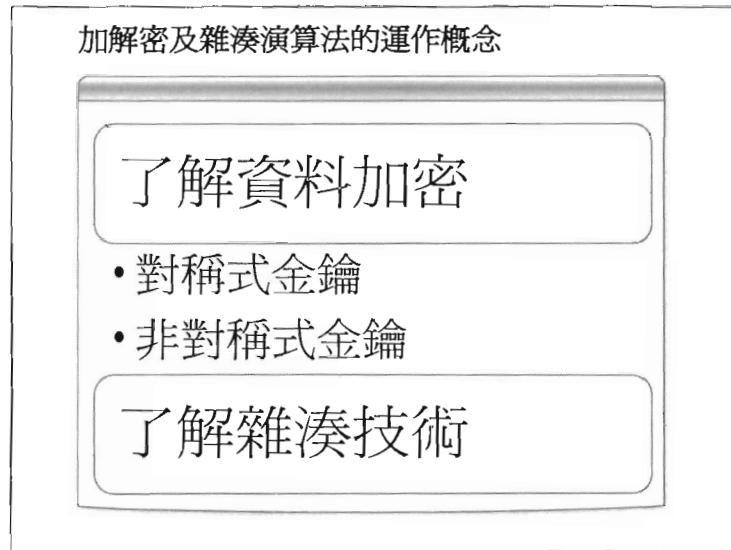


---

資料在網路上的傳遞上可能遭到截取、竄改、重送，如何使用加解密技術進行資料的保護。在這個章節中將介紹如何利用.NET Framework 的加密技術進行資料保護。

本章介紹以下主題：

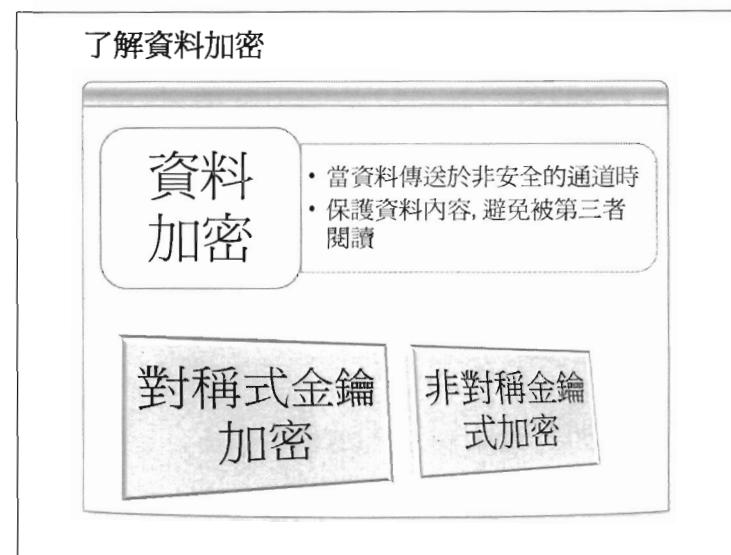
- 加解密及雜湊演算法的運作概念
- 資料的加密及解密
- 資料的雜湊處理



## 加解密及雜湊演算法的運作概念

這一節將介紹資料加密技術的方式，包含對稱式、非對稱式的加密方式的運作原理及特性。另外也將介紹雜湊技術，它是用來確保資料的完整性，這裡也將介紹它的運作原理及特性。

- 了解資料加密
  - 對稱式金鑰
  - 非對稱式金鑰
- 了解雜湊技術

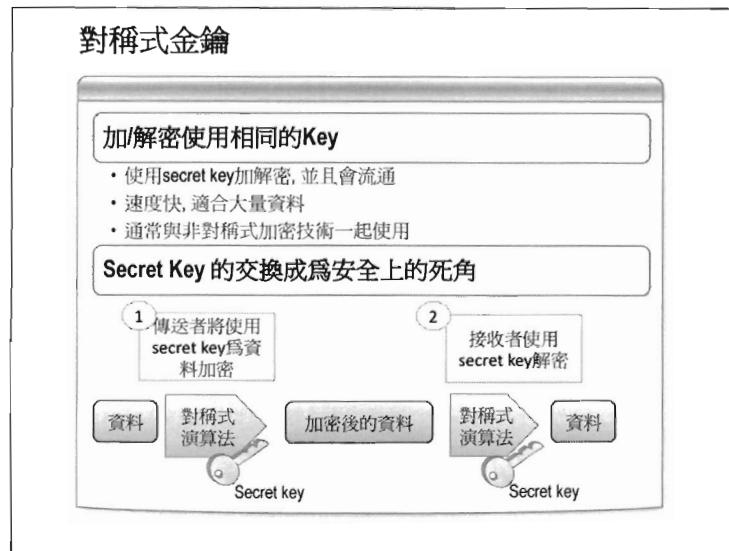


## 了解資料加密

資料加密的目的是資料傳送於非安全的通道，保護資料的內容，避免被第三者閱讀。作法是將原始文件的內容變成一堆利用加/解密的演算法及加密金鑰將資料變成密文，而這些密文必須使用原來的演算法及金鑰才能還原成原始文件。

加/解密的演算法以是否使用相同解密的金鑰分為兩種：

- 對稱式金鑰
- 非對稱式金鑰

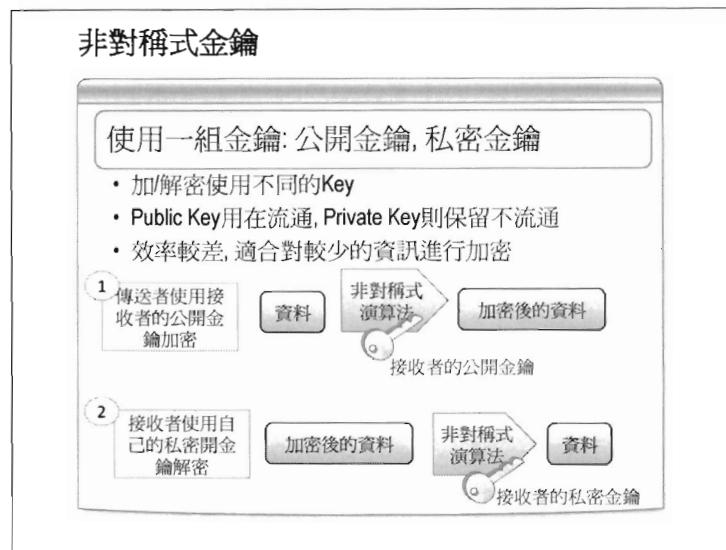


## 對稱式金鑰

對稱式金鑰 (Key) 是非常普遍使用來保密資料的方式，它的方式是加密與解密時使用相同的金鑰，這個金鑰通常被稱為 Secret Key。因為使用相同的金鑰進行加/解密，所以它的演算法通常速度較快，也比較適用於對大量的資料進行加/解密的處理。

不過，加/解密使用相同的金鑰(Secret Key)，也帶來了一個缺點，收到密文的一方也必須拿到 Secret Key 才能還原資料，如果 Secret Key 透過網路傳遞，不只是接收者可以取得 Secret Key，駭客也有機會取得 Secret Key，成爲安全上的死角，因此通常會搭配非對稱式金鑰一起使用。





## 非對稱式金鑰

與對稱式金鑰加解密技術不同，進行加密之前先取得一組金鑰，一個稱為公開金鑰 (Public Key)，一個稱為私密金鑰 (Private Key)。公開金鑰可以允許公開於網路或傳遞給其他人，而私密金鑰則是必須保存好，不會在網路流通及傳遞。進行加密時使用其中一個金鑰，解密就得使用另一個金鑰。依據不同用途，可能使用公開金鑰進行加密，可能使用私密金鑰加密。

加/解密使用不同的金鑰處理，相對的處理的效率也會較對稱式來的差，因此通常用在較短小的資訊上使用非對稱式。



## 資料保密

**情境：**資料由使用者 A (Alice) 發佈給使用者 B (Bob)，Alice 必須確保資料只有 Bob 可以讀取，其他使用者無法解開攔截下來的密文資訊。

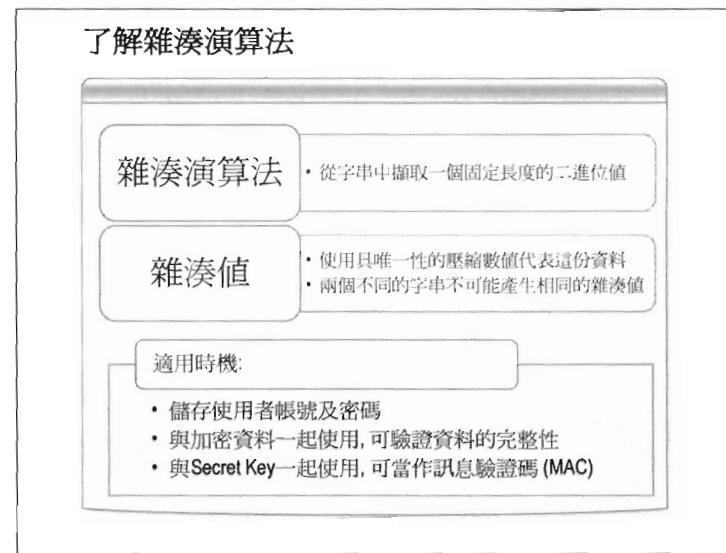
**處理方式：**Alice 使用 Bob 的公開金鑰為資料進行加密。Bob 收到資料使用自己的私密金鑰進行解密。因為資料是使用 Bob 的公開金鑰加密的，所以只有 Bob 可以解開，其他的使用者攔截到密文資訊，因沒有 Bob 的私密金鑰所以無法多解開密文。

**效能改善：**通常要傳遞的資料量是比較大的，使用非對稱式加解密技術會影響到處理效能。改善措施是，使用對稱式為資料進行加密，然後將對稱式的金鑰使用非對稱式加密。

## 確保資料來源

**情境：**資料由使用者 A (Alice) 發佈給使用者 B (Bob)，Bob 必須確認資料是源自於 Alice。

**處理方式：**Alice 使用私密金鑰為資料進行加密(或者說簽名)。Bob 收到資料後可以使用 Alice 的公開金鑰進行解密(可以說成是比對簽名)，如果成功的解開即可確認資料是由 Alice 發佈的。



## 了解雜湊演算法

雜湊演算法是用來驗證資料的完整性。資訊在網路上傳遞有可能遭到攔截、竄改、重送，為了確認資料如同當初傳遞出來的完整未經竄改，可以使用雜湊式演算法。

雜湊演算法的適用時機：

- 儲存使用者帳號及密碼。
- 與加密資料一起使用，可驗證資料的完整性。
- 與 Secret Key 一起使用，可當作訊息驗證碼 (MAC)。

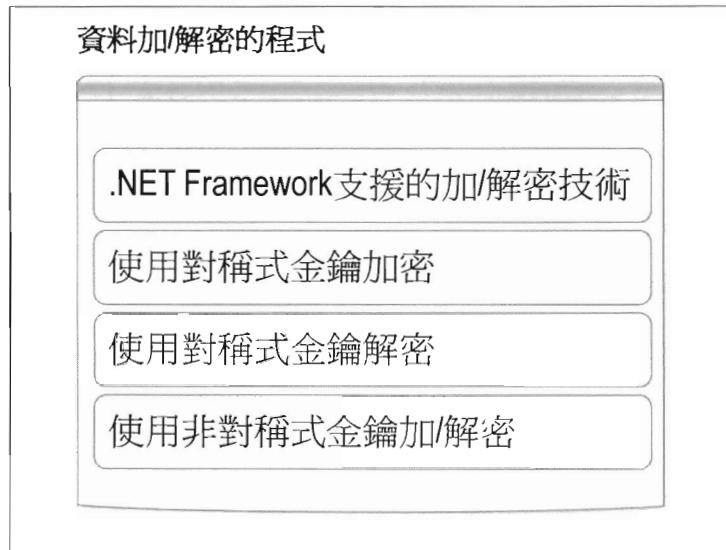
雜湊式演算法會從字串中擷取一個固定長度的二進位雜湊值，是一種單向式的加密技術，也就是資料經過雜湊式演算法產生出來的雜湊值無法經過解密還原為原值。雜湊值使用具唯一性的壓縮數值代表這份資料，兩個不同的字串不可能產生相同的雜湊值。

**情境：**資料由使用者 A (Alice) 發佈給使用者 B (Bob)，Bob 必須確認資料的完整性 (未經竄改)。

**處理方式：**Alice 雜湊式演算法為資料進行處理，取得一份雜湊值，將資料與雜湊值進行傳遞給 Bob，Bob 取得之後使用相同的雜湊式演

算法產生一份新的雜湊值，接著比對二份雜湊是否相同，如果相同代表資料未經竄改，如果不相同代表資料已遭竄改。

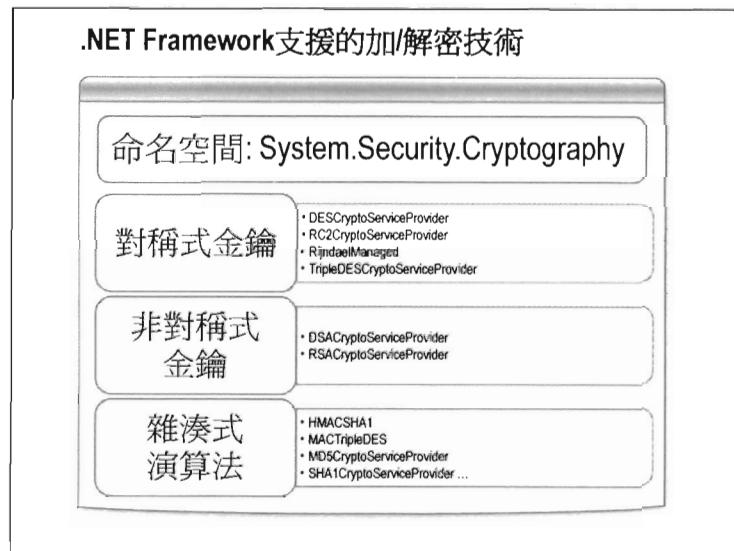
**加強資料保密：**可與加解密演算法搭配使用，加強資訊的保密性。



## 資料加/解密的程式

這一節將介紹如何使用.NET 的程式進行資料加解密的處理，這一節包含以下內容：

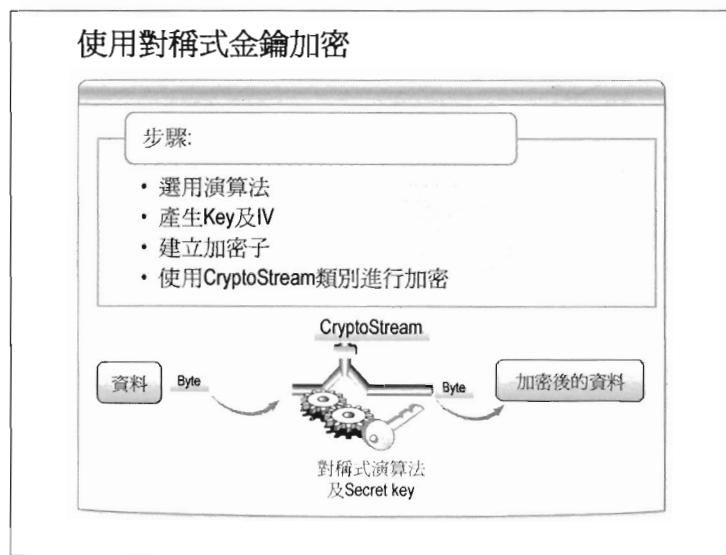
- .NET Framework 支援的加/解密技術
- 使用對稱式金鑰加密
- 使用對稱式金鑰解密
- 使用非對稱式金鑰加/解密



## .NET Framework 支援的加/解密技術

.NET Framework 提供多種加/解密技術的演算法相關類別皆屬於 System.Security.Cryptography 命名空間，相關類別及演算法種類請參考下表：

演算法種類	類別名稱
對稱式金鑰	<ul style="list-style-type: none"> <li>DESCryptoServiceProvider</li> <li>RC2CryptoServiceProvider</li> <li>RijndaelManaged</li> <li>TripleDESCryptoServiceProvider</li> </ul>
非對稱式金鑰	<ul style="list-style-type: none"> <li>DSACryptoServiceProvider</li> <li>RSACryptoServiceProvider</li> </ul>
雜湊式演算法	<ul style="list-style-type: none"> <li>HMACSHA1</li> <li>MACTripleDES</li> <li>MD5CryptoServiceProvider</li> <li>SHA1CryptoServiceProvider ...</li> </ul>



## 使用對稱式金鑰加密

使用對稱式金鑰加密的步驟如下：

- 選用演算法
- 產生 Key 及 IV
- 建立加密子
- 使用 CryptoStream 類別進行加密

### **DESCryptoServiceProvider** 類別

DESCryptoServiceProvider 與 TripleDESCryptoServiceProvider 是非常常用的加解密演算法，DES 是一種歷史悠久的演算法，而 TripleDES 則提供比 DES 演算法更安全可靠的一種演算法 (進行三次的 DES 計算)。它們的成員如下：

- Key，取得或設定演算法的金鑰值。
- IV，取得或設定演算法的起始值。
- GenerateKey()，隨機產生一組 Byte 陣列的金鑰值。
- GenerateIV()，隨機產生一組 Byte 陣列的起始值。
- CreateEncryptor()，傳入 Key 與 IV 產生型別為 ICryptoTransform 介面的加密物件。

- CreateDecryptor()，傳入 Key 與 IV 產生型別為 ICryptoTransform 介面的解密物件。

## CryptoStream 類別

負責對資訊串流 (Stream) 進行加解密作業的類別，在建立此類別時必須傳入資訊串流及使用的演算法 (包含金鑰)，還有讀寫方向。



以下範例是使用 DESCryptoServiceProvider 進行資料的加密。

```

Visual Basic
Dim key As Byte()
Dim iv() As Byte
Dim cipherMessage() As Byte
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
'建立加解密的驗算法
    Dim objDES As New DESCryptoServiceProvider()
'產生加解密的Key與IV
    objDES.GenerateKey()
    objDES.GenerateIV()
    key = objDES.Key
    iv = objDES.IV
'建立加密的物件
    Dim objCrypto As ICryptoTransform = objDES.CreateEncryptor(key, iv)
    Dim ms As New MemoryStream
'進行加密，並將密文寫入串流
    Dim cs As New CryptoStream(ms, objCrypto, CryptoStreamMode.Write)
    Dim sw As New StreamWriter(cs)
    sw.WriteLine(TextBox1.Text)
    sw.Flush()
    cs.FlushFinalBlock()

    ms.Position = 0
    ReDim cipherMessage(ms.Length - 1)
    ms.Read(cipherMessage, 0, ms.Length)
'密文以字串顯示
    TextBox2.Text = BitConverter.ToString(cipherMessage)

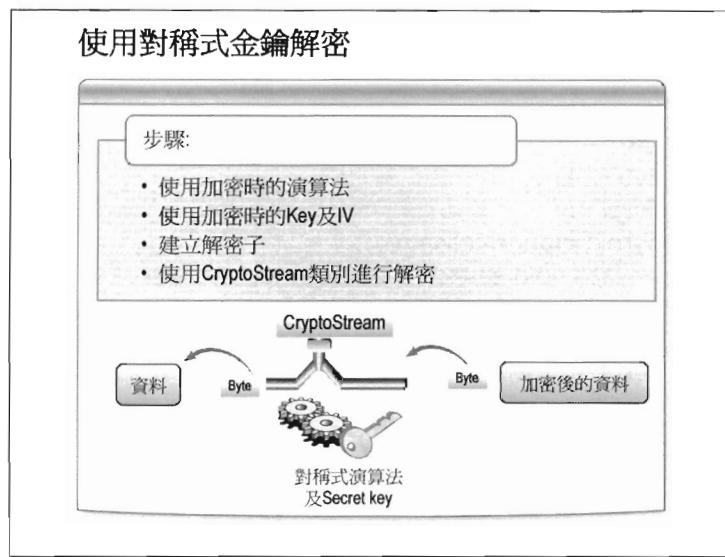
```

```
    cs.Close()
End Sub
```

```
C#
Byte[] key;
Byte[] iv ;
Byte[] cipherMessage;
private void Button1_Click(object sender, EventArgs e)
{
    DESCryptoServiceProvider objDES =new DESCryptoServiceProvider();
    objDES.GenerateKey();
    objDES.GenerateIV();
    key = objDES.Key;
    iv = objDES.IV;

    ICryptoTransform objCrypto = objDES.CreateEncryptor(key, iv);
    MemoryStream ms =new MemoryStream();
    CryptoStream cs =new CryptoStream(ms, objCrypto, CryptoStreamMode.Write);
    StreamWriter sw =new StreamWriter(cs);
    sw.WriteLine(TextBox1.Text);
    sw.Flush();
    cs.FlushFinalBlock();

    ms.Position = 0;
    cipherMessage = new Byte[ms.Length];
    ms.Read(cipherMessage, 0, (int)ms.Length);
    TextBox2.Text = BitConverter.ToString(cipherMessage);
    cs.Close();
}
```



## 使用對稱式金鑰解密

使用對稱式金鑰解密的步驟如下：

- 使用加密時的演算法
- 使用加密時的 Key 及 IV
- 建立解密子
- 使用 CryptoStream 類別進行解密



以下範例是使用 DESCryptoServiceProvider 進行資料的解密。

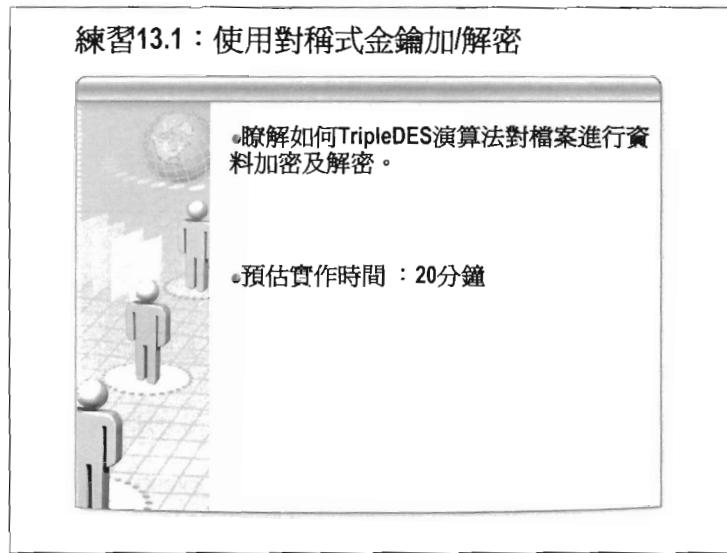
### Visual Basic

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    Dim objDES As New DESCryptoServiceProvider
    Dim objCrypto As ICryptoTransform = objDES.CreateDecryptor
```

```
(key, iv)
    Dim ms As New MemoryStream(cipherMessage)
    Dim cs As New CryptoStream(ms, objCrypto, CryptoStreamMode.Read)
        Dim sr As New StreamReader(cs)
        TextBox3.Text = sr.ReadToEnd()
        sr.Close()
End Sub
```

```
C#
private void Button2_Click(object sender, EventArgs e)
{
    DESCryptoServiceProvider objDES = new DESCryptoServiceProvider();
    ICryptoTransform objCrypto = objDES.CreateDecryptor(key, iv);
    MemoryStream ms = new MemoryStream(cipherMessage);
    CryptoStream cs = new CryptoStream(ms, objCrypto, CryptoStreamMode.Read);

    StreamReader sr = new StreamReader(cs);
    TextBox3.Text = sr.ReadToEnd();
    sr.Close();
}
```



## 練習 13.1：使用對稱式金鑰加/解密

目的：

瞭解如何使用對稱式金鑰的 TripleDES 演算法對檔案進行資料加密及解密。

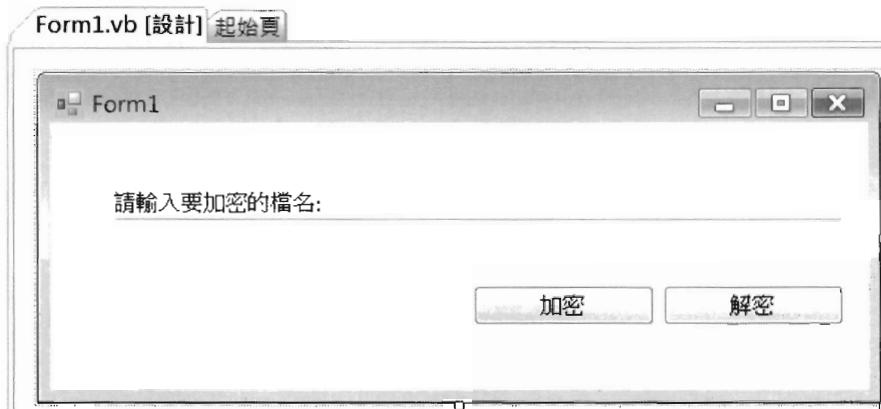
功能描述：

在這個練習會針對使用者所指定的檔案進行加密及解密，並且使用對稱式金鑰的 TripleDES 演算法。

預估實作時間：20分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod13\_1。
3. 在 Form1 設計畫面如下圖：



4. 汇入以下命名空間：

```
Visual Basic
Imports System.Security.Cryptography
Imports System.IO
Imports System.Text
```

```
C#
using System.Security.Cryptography;
using System.IO;
```

5. 撰寫加密的副程式，名稱是 ProcessCrypto：

- 產生加密時所需的 Key 及 IV。
- 將 Key 及 IV 存入 MemoryStream。
- 使用 TripleDESCryptoServiceProvider 及 CryptoStream 類別進行加密作業，並將加密內容寫到 MemoryStream。
- 呼叫 MemoryStream 的 ToArray 方法並傳回其結果。

```
Visual Basic
Function ProcessCrypto(ByVal sourceStream As Stream) As Byte()
    Dim key(), iv() As Byte
    Dim objDES As New TripleDESCryptoServiceProvider()
    objDES.GenerateKey()
    objDES.GenerateIV()
    key = objDES.Key
    iv = objDES.IV

    Dim objCrypto As ICryptoTransform = objDES.CreateEncryptor(key, iv)
    Dim ms As New MemoryStream
```

```

Dim writer As New BinaryWriter(ms)
writer.Write(key)
writer.Write(iv)

Using cs As New CryptoStream(ms, objCrypto, CryptoStream
Mode.Write)
    Dim buffer(1023) As Byte
    Dim readCount As Integer
    Do
        readCount = sourceStream.Read(buffer, 0, 1024)
        cs.Write(buffer, 0, readCount)
        Array.Clear(buffer, 0, 1024)
    Loop While readCount = 1023
    cs.FlushFinalBlock()

    Return ms.ToArray()
End Using
End Function

```

```

C#
byte[] ProcessEncrypto(Stream sourceStream )
{
    Byte[] key, iv ;
    TripleDESCryptoServiceProvider objDES = new TripleDESCry
ptoServiceProvider();
    objDES.GenerateKey();
    objDES.GenerateIV();
    key = objDES.Key;
    iv = objDES.IV;
    ICryptoTransform objCrypto = objDES.CreateEncryptor(key,
iv);

    MemoryStream ms = new MemoryStream();
    BinaryWriter writer = new BinaryWriter(ms);
    writer.Write(key);
    writer.Write(iv);

    using (CryptoStream cs = new CryptoStream(ms, objCrypto,
CryptoStreamMode.Write))
    {
        Byte [] buffer= new Byte[1024];
        int readCount ;
        do{
            readCount = sourceStream.Read(buffer, 0, 1024);
            cs.Write(buffer, 0, readCount);
            Array.Clear(buffer, 0, 1024);
        } while (readCount == 1023);
        cs.FlushFinalBlock();    {s>4

        return ms.ToArray();
    }
}

```

6. 撰寫解密的副程式，名稱是 ProcessDecrypto：

- 讀取檔頭中的 Key 及 IV。
- 讀取加密資料並存入 MemoryStream，MemoryStream 物件名稱為 encMS。
- 使用 TripleDESCryptoServiceProvider 及 CryptoStream 類別進行解密作業，並將解密內容寫到 MemoryStream，MemoryStream 物件名稱為 decMS。
- 呼叫 MemoryStream(decMS)的 ToArray 方法並傳回其結果。

#### Visual Basic

```

Function ProcessDecrypto(ByVal sourceStream As Stream) As Byte()
    Dim key, iv, data As Byte()
    Dim reader As New BinaryReader(sourceStream)
    key = reader.ReadBytes(24)
    iv = reader.ReadBytes(8)
    data = reader.ReadBytes(sourceStream.Length - 30)
    Dim encMS As New MemoryStream(data)
    Dim decMS As New MemoryStream
    Dim objDES As New TripleDESCryptoServiceProvider()
    Dim objCrypto As ICryptoTransform = objDES.CreateDecryptor(key, iv)

    Using cs As New CryptoStream(encMS, objCrypto, CryptoStreamMode.Read)
        Dim buffer(1023) As Byte
        Dim readCount As Integer
        Do
            readCount = cs.Read(buffer, 0, 1024)
            decMS.Write(buffer, 0, readCount)
            Array.Clear(buffer, 0, 1024)
        Loop While readCount = 1023
        decMS.Flush()

        Return decMS.ToArray
    End Using
End Function

```

#### C#

```

byte[] ProcessDecrypto(Stream sourceStream )
{
    byte[] key, iv, data;
    BinaryReader reader = new BinaryReader(sourceStream);
    key = reader.ReadBytes(24);
    iv = reader.ReadBytes(8);
}

```

```

        data = reader.ReadBytes((int)sourceStream.Length);
        MemoryStream encMS = new MemoryStream(data);
        MemoryStream decMS = new MemoryStream();
        TripleDESCryptoServiceProvider objDES = new TripleDESCryptoServiceProvider();
        ICryptoTransform objCrypto = objDES.CreateDecryptor(key, iv);

        using (CryptoStream cs = new CryptoStream(encMS, objCrypto, CryptoStreamMode.Read))
        {
            Byte[] buffer = new byte[1024];
            int readCount;
            do
            {
                readCount = cs.Read(buffer, 0, 1024);
                decMS.Write(buffer, 0, readCount);
                Array.Clear(buffer, 0, 1024);
            } while (readCount == 1024);
            decMS.Flush();

            return decMS.ToArray();
        }
    }
}

```

7. 在「加密」按鈕的 Click 事件進行檔案加密的處理：

- 如果使用者輸入的檔名不存在就不處理。
- 使用 FileStream 開啓檔案，並呼叫 ProcessEncrypto 方法傳入 FileStream 物件。
- 確認接收到 ProcessEncrypto 方法所回傳的 Byte 陣列有資料後將 Byte 陣列(加密後的內容)存入檔案。

Visual Basic

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim path As String = TextBox1.Text
    If Not File.Exists(path) Then
        Return
    End If
    Using fs As New FileStream(path, FileMode.Open)
        Dim encrypto() As Byte = ProcessEncrypto(fs)
        If encrypto.Length = 0 Then
            MessageBox.Show("沒有處理!!")
            Return
        End If
        Dim outputFS As New FileStream(path & ".enc", FileMode.Create)
        outputFS.Write(encrypto, 0, encrypto.Length)
        outputFS.Close()
    End Using
End Sub

```

```

End Using
MessageBox.Show("加密完成")
End Sub

```

```

C#
private void Button1_Click(object sender, EventArgs e)
{
    String path = TextBox1.Text;
    if (!File.Exists(path) )
        return;
    using (FileStream fs = new FileStream(path, FileMode.Open))
    {
        Byte[] encrypto = ProcessEncrypto(fs);
        if ( encrypto.Length == 0)
        {
            MessageBox.Show("沒有處理!!!");
            return;
        }
        FileStream outputFS = new FileStream(path + ".enc", FileMode.Create);
        outputFS.Write(encrypto, 0, encrypto.Length );
        outputFS.Close();
    }
    MessageBox.Show("加密完成");
}

```

8. 在「解密」按鈕的 Click 事件進行檔案解密的處理：

- 如果使用者輸入的檔名不存在就不處理。
- 使用 FileStream 開啓檔案，並呼叫 ProcessDecrypto 方法傳入 FileStream 物件。
- 確認接收到 ProcessDecrypto 方法所回傳的 Byte 陣列有資料後將 Byte 陣列(解密後的內容)存入檔案。

```

Visual Basic
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    Dim path As String = TextBox1.Text
    If Not File.Exists(path) Then
        Return
    End If
    Using fs As New FileStream(path, FileMode.Open)
        Dim decrypto() As Byte = ProcessDecrypto(fs)
        If decrypto.Length = 0 Then
            MessageBox.Show("沒有處理!!")
        Return
    End If

```

```

End If
Dim decFileName As String = System.IO.Path.ChangeExtension(path, ".dec")
Dim outputFS As New FileStream(decFileName, FileMode.Create)
outputFS.Write(decrypto, 0, decrypto.Length)
outputFS.Close()
End Using
MessageBox.Show("解密完成")
End Sub

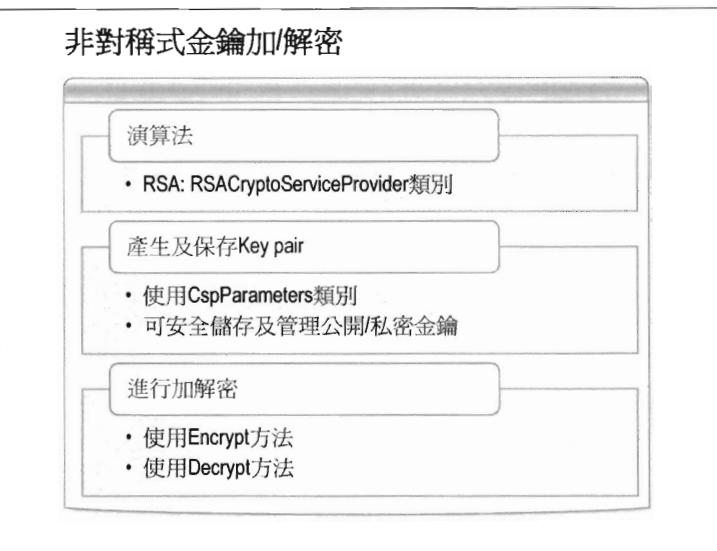
```

```

C#
private void Button2_Click(object sender, EventArgs e)
{
    String path = TextBox1.Text;
    if (!File.Exists(path)) return;
    using (FileStream fs = new FileStream(path, FileMode.Open))
    {
        Byte[] decrypto = ProcessDecrypto(fs);
        if (decrypto.Length == 0)
        {
            MessageBox.Show("沒有處理!!!");
            return;
        }
        String decFileName = System.IO.Path.ChangeExtension(path, ".dec");
        FileStream outputFS = new FileStream(decFileName, FileMode.Create);
        outputFS.Write(decrypto, 0, decrypto.Length);
        outputFS.Close();
    }
    MessageBox.Show("解密完成");
}

```

9. 分別測試檔案的加密及解密，並確認是否正確進行加解密作業。



## 非對稱式金鑰加/解密

非對稱式金鑰演算法是 RSA，.NET 提供的類別是 RSACryptoServiceProvider。進行非對稱式金鑰進行加密的步驟是：

1. 產生 Key pair，使用 CspParameters 類別可以產生及安全的儲存及管理公開/私密金鑰。
2. 建立 RSACryptoServiceProvider 物件，並在呼叫建構函式是傳入 CspParameters 物件。
3. 呼叫 RSACryptoServiceProvider 的 Encrypt 方法進行加密。

以下範例是使用 RSACryptoServiceProvider 類別進行加密處理：

```
Visual Basic
Dim param As New CspParameters()
param.KeyContainerName = "MyKeyContainer"
param.Flags = CspProviderFlags.UseMachineKeyStore
Dim RSA As New RSACryptoServiceProvider(param)

arEncrypted = RSA.Encrypt(key, False)
```

C#

```
CspParameters param = new CspParameters();
param.KeyContainerName = "MyKeyContainer";
param.Flags = CspProviderFlags.UseMachineKeyStore;
RSACryptoServiceProvider RSA = new RSACryptoServiceProvider(param);

arEncrypted = RSA.Encrypt(key, false);
```

進行非對稱式金鑰進行解密的步驟是：

1. 取得的 Key pair，使用 CspParameters 類別可以產生及安全的儲存及管理公開/私密金鑰。
2. 建立 RSACryptoServiceProvider 物件，並在呼叫建構函式是傳入 CspParameters 物件。
3. 呼叫 RSACryptoServiceProvider 的 Decrypt 方法進行加密。

以下範例是使用 RSACryptoServiceProvider 類別進行解密處理：

**Visual Basic**

```
Dim param As New CspParameters()
param.KeyContainerName = "MyKeyContainer"
param.Flags = CspProviderFlags.UseMachineKeyStore
Dim RSA As New RSACryptoServiceProvider(param)

Dim arDecrypted() As Byte
arDecrypted = RSA.Decrypt(arEncrypted, False)
```

**C#**

```
CspParameters param = new CspParameters();
param.KeyContainerName = "MyKeyContainer";
param.Flags = CspProviderFlags.UseMachineKeyStore;
RSACryptoServiceProvider RSA = new RSACryptoServiceProvider(param);

Byte[] arDecrypted = RSA.Decrypt(arEncrypted, false);
```

### 練習13.2：使用非對稱式金鑰加/解密

- 上個練習對稱式的Key附加在檔案中，無法真的做到保護檔案內容的作用。
- 這個練習使用非對稱式金鑰加強對 Secret Key 的保護。
- 瞭解如何RSA非對稱式演算法為Secret Key進行加密及解密。

•預估實作時間：15分鐘

## 練習 13.2：使用非對稱式金鑰加/解密

### 目的：

瞭解如何 RSA 非對稱式演算法為 Secret Key 進行加密及解密。

### 功能描述：

上個練習將 Secret Key 附加在檔案中，無法真的做到保護檔案內容的作用。這個練習使用非對稱式金鑰加強對 Secret Key 的保護。

預估實作時間：20 分鐘

### 實作步驟：

1. 開啓上一個練習 Mod13\_1 專案。
2. 開啓 Form1 的程式碼，加上 ProtectKey 副程式，使用非對稱式金鑰保護 Secret Key：

Visual Basic

```
Function ProtectKey(ByVal key As Byte()) As Byte()
    Dim param As New CspParameters()
    param.KeyContainerName = "MyKeyContainer"
    param.Flags = CspProviderFlags.UseMachineKeyStore
    Dim RSA As New RSACryptoServiceProvider(param)
```

```

    Return RSA.Encrypt(key, False)
End Function

```

```

C#
byte[] ProtectKey(Byte[] key)
{
    CspParameters param = new CspParameters();
    param.KeyContainerName = "MyKeyContainer";
    param.Flags = CspProviderFlags.UseMachineKeyStore;
    RSACryptoServiceProvider RSA = new RSACryptoServiceProvider(param);

    return RSA.Encrypt(key, false);
}

```

3. 開啓 Form1 的程式碼，加上 UnprotectKey 副程式，使用非對稱式金鑰解除 Secret Key 的保護：

```

Visual Basic
Function UnprotectKey(ByVal key As Byte()) As Byte()
    Dim param As New CspParameters()
    param.KeyContainerName = "MyKeyContainer"
    param.Flags = CspProviderFlags.UseMachineKeyStore
    Dim RSA As New RSACryptoServiceProvider(param)

    Return RSA.Decrypt(key, False)
End Function

```

```

C#
byte[] UnprotectKey(byte[] key)
{
    CspParameters param = new CspParameters();
    param.KeyContainerName = "MyKeyContainer";
    param.Flags = CspProviderFlags.UseMachineKeyStore;
    RSACryptoServiceProvider RSA = new RSACryptoServiceProvider(param);

    return RSA.Decrypt(key, false);
}

```

4. 找到 ProcessEncrypt 方法的程式區塊，將之前直接寫入 Key 及 IV 到檔案的程式碼修改成呼叫 ProtectKey 方法後再寫入檔案：

Visual Basic

```
Dim writer As New BinaryWriter(ms)
writer.Write(ProtectKey(key))
writer.Write(ProtectKey(iv))
```

C#

```
BinaryWriter writer = new BinaryWriter(ms);
writer.Write(ProtectKey(key));
writer.Write(ProtectKey(iv));
```

5. 找到 ProcessDecrypto 方法的程式區塊，將之前直接讀取檔案的 Key 及 IV 程式碼修改成呼叫 UnprotectKey 方法：

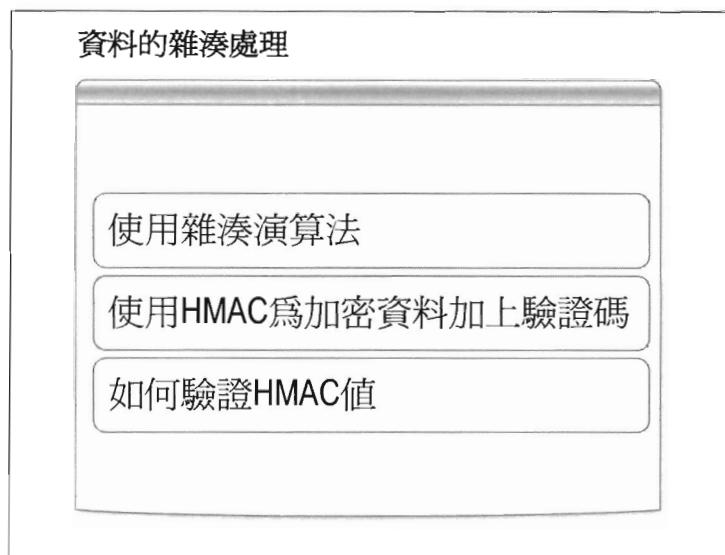
Visual Basic

```
Dim reader As New BinaryReader(sourceStream)
key = UnprotectKey(reader.ReadBytes(128))
iv = UnprotectKey(reader.ReadBytes(128))
data = reader.ReadBytes((int)sourceStream.Length - 256);
```

C#

```
BinaryReader reader = new BinaryReader(sourceStream);
key = UnprotectKey(reader.ReadBytes(128));
iv = UnprotectKey(reader.ReadBytes(128));
data = reader.ReadBytes((int)sourceStream.Length - 256);
```

6. 分別測試檔案的加密及解密，並確認是否正確進行加解密作業。



## 資料的雜湊處理

這一節將介紹如何在.NET 的應用程式中使用雜湊演算的技術驗證資料的完整性，這一節包含以下內容：

- 使用雜湊演算法
- 使用 HMAC 為加密資料加上驗證碼
- 如何驗證 HMAC 值



.NET Framework 提供的雜湊式演算法的類別有：

- HMACSHA1
- MACTripleDES
- MD5CryptoServiceProvider
- SHA1CryptoServiceProvider
- HMACSHA1
- MACTripleDES
- MD5CryptoServiceProvider
- SHA1CryptoServiceProvider

你可以選擇適當的類別進行資料完整性的驗證，作業方式是在建置雜湊值時，根據內容產生雜湊值。在需要驗證完整性時，根據內容產生雜湊值，然後與建置時所產生的雜湊值比對。

以下範例是使用 SHA1CryptoServiceProvider 進行雜湊式演算比對作業。

1. 原始資料為 TextBox1 中輸入的值，使用 SHA1CryptoServiceProvider 演算法取得雜湊值，並將雜湊值顯示於 TextBox2 控制項中。

```

Dim Hash1() As Byte
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim Message As Byte() = _
        Encoding.Unicode.GetBytes(TextBox1.Text)

    Dim SHhash As New SHA1CryptoServiceProvider()
    Dim HashValue As Byte()

    HashValue = SHhash.ComputeHash(Message)
    Hash1 = HashValue

    TextBox2.Text = BitConverter.ToString(HashValue)
End Sub

```

```

C#
Byte[] Hash1;
private void Button1_Click(object sender, EventArgs e)
{
    Byte[] Message = Encoding.Unicode.GetBytes(textBox1.Text);

    SHA1CryptoServiceProvider SHhash =
        new SHA1CryptoServiceProvider();
    Byte[] HashValue = SHhash.ComputeHash(Message);
    Hash1 = HashValue;

    textBox2.Text = BitConverter.ToString(HashValue);
}

```

2. 在 TextBox3 控制項輸入一個新值，並使用 SHA1CryptoServiceProvider 產生新的雜湊值。接著與第一份雜湊值進行比對，如果比對正確，代表密碼正確。

```

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    Dim Message As Byte() = _
        Encoding.Unicode.GetBytes(textBox3.Text)

    Dim SHhash As New SHA1CryptoServiceProvider()
    Dim Hash2 As Byte()

    Hash2 = SHhash.ComputeHash(Message)

    Dim bSame As Boolean = True
    For i As Integer = 0 To Hash1.Length - 1
        If Hash1(i) <> Hash2(i) Then
            bSame = False
            Exit For
    Next
    If bSame Then
        MessageBox.Show("密碼正確")
    Else
        MessageBox.Show("密碼錯誤")
    End If
End Sub

```

```
End If
Next

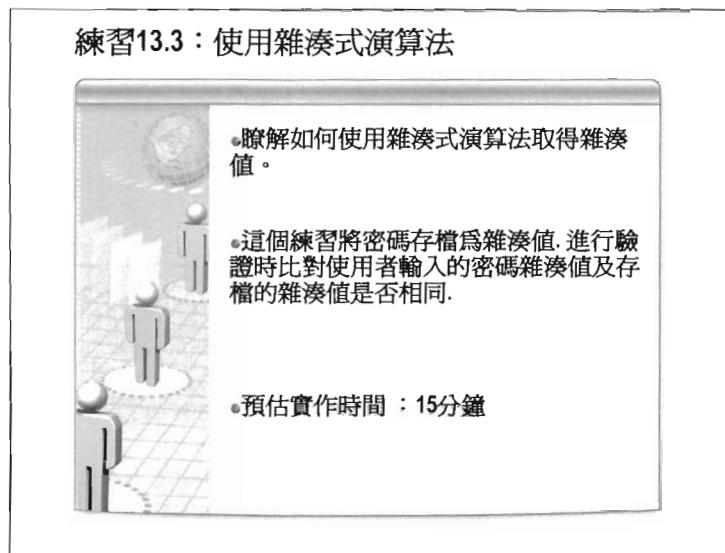
If bSame Then
    TextBox4.Text = "比對正確"
Else
    TextBox4.Text = "比對不正確"
End If
End Sub
```

```
C#
private void Button2_Click(object sender, EventArgs e)
{
    Byte[] Message = Encoding.Unicode.GetBytes(TextBox3.Text);
    SHA1CryptoServiceProvider SHhash =
        new SHA1CryptoServiceProvider();
    Byte[] Hash2 ;

    Hash2 = SHhash.ComputeHash(Message);

    bool bSame = true;
    for (int i = 0 ; i < Hash1.Length ; i++)
    {
        if (Hash1[i] != Hash2[i])
        {
            bSame = false;
            break;
        }
    }

    if (bSame)
        TextBox4.Text = "比對正確";
    else
        TextBox4.Text = "比對不正確";
}
```



### 練習 13.3：使用雜湊式演算法

#### 目的：

瞭解如何應用雜湊式演算法進行帳號密碼的驗證處理。

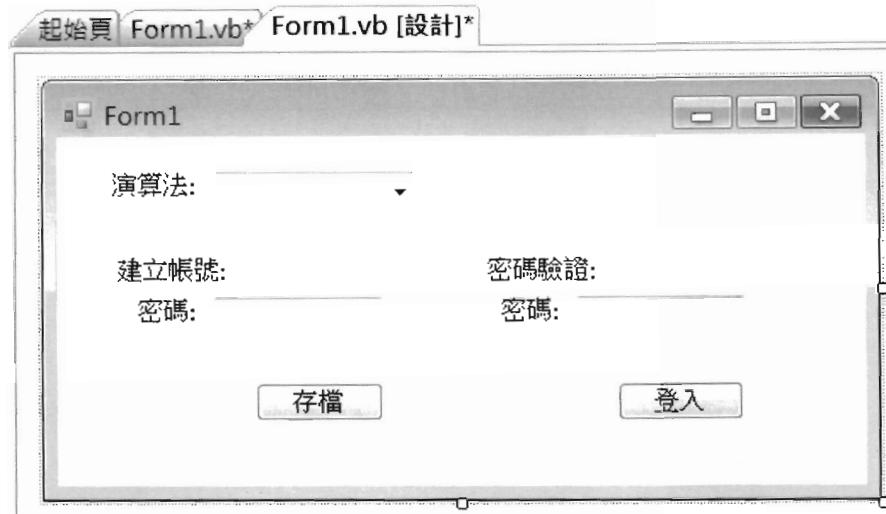
#### 功能描述：

在這個練習會在建立帳號時將密碼存檔為雜湊值。在進行密碼驗證時，比對使用者輸入的密碼所產生的雜湊值及存檔的雜湊值是否相同。

#### 預估實作時間：15分鐘

#### 實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod13\_3。
3. 在 Form1 設計畫面如下圖：



4. 匯入必要的命名空間：

```
Visual Basic
Imports System.IO
Imports System.Text
Imports System.Security.Cryptography
```

```
C#
using System.IO;
using System.Security.Cryptography;
```

5. 在 Form1 加入一個副程式名稱是 GetHashCode，用來產生雜湊值：

```
Visual Basic
Function GetHashCode(ByVal hashName As String, ByVal data As String) As Byte()
    Dim hashAlog As HashAlgorithm
    hashAlog = HashAlgorithm.Create(hashName)

    Dim buffer() As Byte = Encoding.UTF8.GetBytes(data)
    Return hashAlog.ComputeHash(buffer)
End Function
```

```
C#
byte[] GetHashCode(String hashName , String data )
{
    HashAlgorithm hashAlog = HashAlgorithm.Create(hashName)
```

```

e);

Byte[] buffer = Encoding.UTF8.GetBytes(data);
return hashAlog.ComputeHash(buffer);
}

```

6. 加入另一個副程式名稱 VerifyHashCode，用來比對兩個雜湊值是否相同：

**Visual Basic**

```

Function VerifyHashCode(ByVal hash1() As Byte, ByVal hash2()
() As Byte) As Boolean
    For i As Integer = 0 To hash1.Length - 1
        If hash1(i) <> hash2(i) Then
            Return False
        End If
    Next
    Return True
End Function

```

**C#**

```

bool VerifyHashCode(Byte[] hash1, Byte[] hash2)
{
    for (int i = 0; i < hash1.Length; i++)
    {
        if (hash1[i] != hash2[i])
            return false;
    }
    return true;
}

```

7. 在「存檔」按鈕撰寫以下程式，以將使用者所輸入的密碼經過指定的(ComboBox1 控制項) 雜湊演算法產生雜湊值並存檔：

**Visual Basic**

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    If String.IsNullOrEmpty(ComboBox1.Text) OrElse String.IsNullOrEmpty(TextBox1.Text) Then
        Return
    End If
    Dim hashName As String = ComboBox1.Text
    Dim hashValue() As Byte = GetHashCode(hashName, TextBox1.Text)

    Dim saveToFile As New FileStream("mypassword.dat", FileMode.Create)

```

```

ode.Create)
    saveToFile.Write(hashValue, 0, hashValue.Length)
    saveToFile.Close()
    MessageBox.Show("存檔成功")
End Sub

```

C#

```

private void Button1_Click(object sender, EventArgs e)
{
    if (String.IsNullOrEmpty(ComboBox1.Text) || String.IsNullOrEmpty(textBox1.Text))
        return;

    String hashName = ComboBox1.Text;
    Byte[] hashValue = GetHashValue(hashName, textBox1.Text);

    FileStream saveToFile=new FileStream("mypassword.dat",
", FileMode.Create);
    saveToFile.Write(hashValue, 0, hashValue.Length);
    saveToFile.Close();
    MessageBox.Show("存檔成功");
}

```

- 在「登入」按鈕撰寫以下程式，以將使用者所輸入的密碼經過指定的(ComboBox1 控制項) 雜湊演算法產生雜湊值，並與先前存檔的雜湊值進行比對：

Visual Basic

```

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    If String.IsNullOrEmpty(ComboBox1.Text) OrElse String.IsNullOrEmpty(textBox2.Text) Then
        Return
    End If
    Dim hashName As String = ComboBox1.Text
    Dim hashValue2() As Byte = GetHashValue(hashName, textBox2.Text)

    Dim saveToFile As New FileStream("mypassword.dat", FileMode.Open)
    Dim hashValue(saveToFile.Length - 1) As Byte
    saveToFile.Read(hashValue, 0, saveToFile.Length)
    saveToFile.Close()

    If VerifyHashValue(hashValue, hashValue2) Then
        MessageBox.Show("密碼正確")
    Else
        MessageBox.Show("密碼不正確")
    End If
End Sub

```

```
End If  
End Sub
```

C#

```
private void Button2_Click(object sender, EventArgs e)  
{  
    if (String.IsNullOrEmpty(ComboBox1.Text) || String.IsNullOrEmpty(textBox2.Text))  
        return;  
  
    string hashName = ComboBox1.Text;  
    byte[] hashValue2 = GetHashValue(hashName, textBox2.Text);  
  
    FileStream saveToFile = new FileStream("mypassword.dat", FileMode.Open);  
    byte[] hashValue = new byte[saveToFile.Length];  
    saveToFile.Read(hashValue, 0, (int)saveToFile.Length);  
    saveToFile.Close();  
  
    if (VerifyHashValue(hashValue, hashValue2))  
        MessageBox.Show("密碼正確");  
    else  
        MessageBox.Show("密碼不正確");  
}
```

9. 執行程式並測試。如果第一次輸入的密碼與第二次輸入的密碼相符合，得到的結果就會是「密碼正確」，否則就會「密碼不正確」。

## 總結

了解加解密技術

了解如何使用對稱式演算法

了解如何使用非對稱式演算法

了解如何使用雜湊式演算法

# 第十四章：與非.NET 程式的互通

## 本章大綱

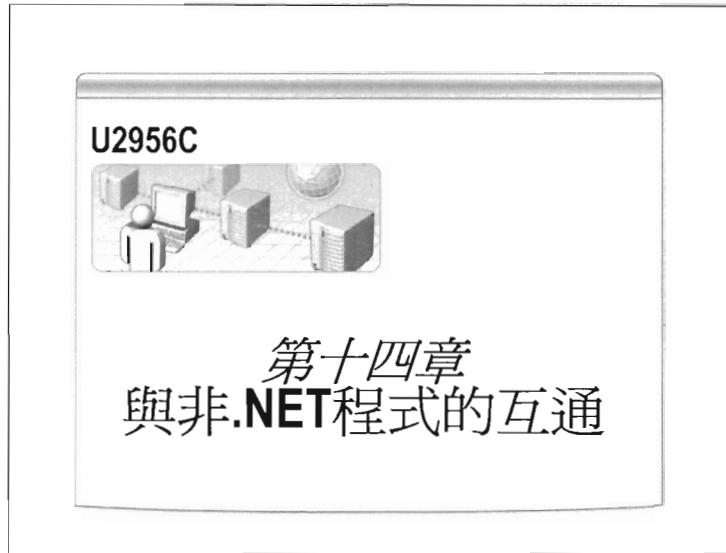
認識 Win32 API.....	4
Platform Invocation Service.....	6
DllImportAttribute.....	9
處理複雜的型別.....	10
練習 14.1：呼叫 Win32 API .....	13
認識 Runtime Callable Wrapper .....	19
Wrapper class.....	20
建立 Interop Assembly.....	21
如何存取 COM 物件.....	23
練習 14.2：在.NET 中呼叫 COM 元件 .....	26
認識 COM Callable Wrapper .....	30
設計可讓 COM 呼叫的.NET 元件.....	32
建立.NET 元件給 COM 用戶端使用 .....	34
部署給 COM 使用的.NET 元件.....	37
練習 14.3：建立.NET 組件供 COM 用戶端使用.....	39

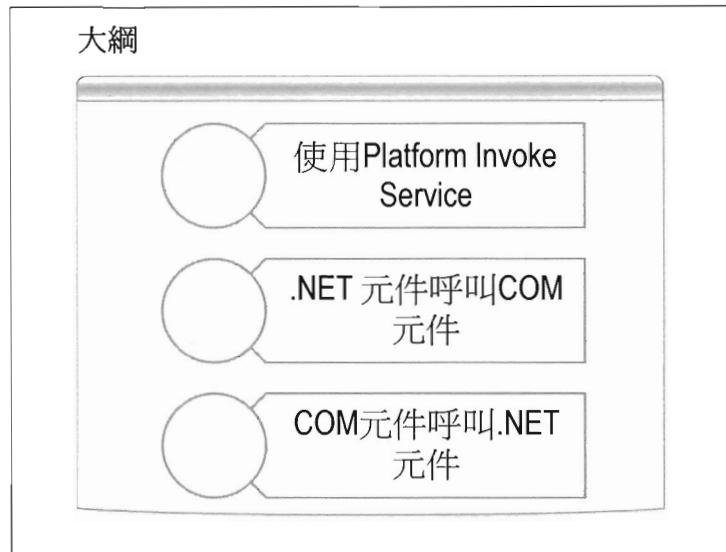
提昇您專業競爭實力的最佳夥伴

作者：

羅慧真



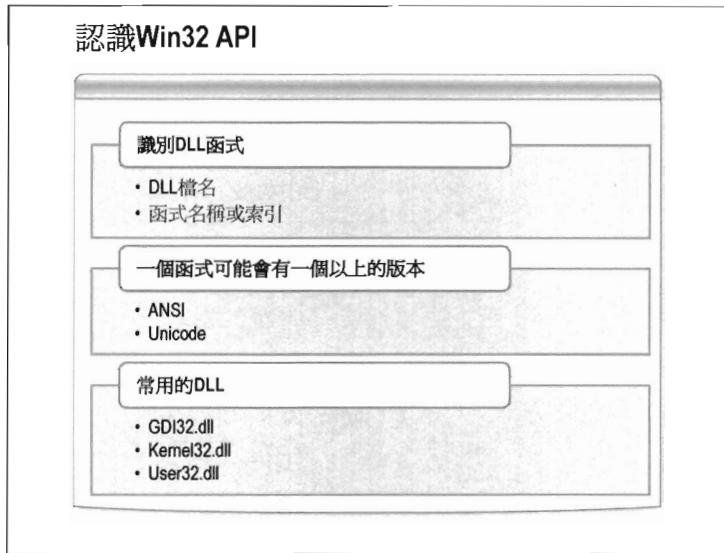




.NET Framework 可以與 API 及 COM 元件程式互動，這一章將介紹如何讓.NET 程式與 API 及 COM 元件程式互動。

本章介紹以下主題：

- 使用 Platform Invoke Service
- .NET 元件呼叫 COM 元件
- COM 元件呼叫.NET 元件



## 認識 Win32 API

每一個 DLL 函式可由以下兩項資訊唯一識別：

- 函式名稱或索引序號
- 包含實作程式的 DLL 檔名。

舉例而言，以 User32.dll 裡的 MessageBox 函式為例，可用函式名稱 MessageBox 加上 DLL 檔名：User32.dll、User32 或 user32 來唯一識別。

每一個 Win32 API (Windows application programming interface) 的函式可有兩個版本：ANSI 及 Unicode。在不指定情況下，預設為 ANSI。某些函式甚至有兩個以上的版本。以 MessageBox 函式為例，MessageBoxA 是 ANSI 版、MessageBoxW 是 Unicode 版。

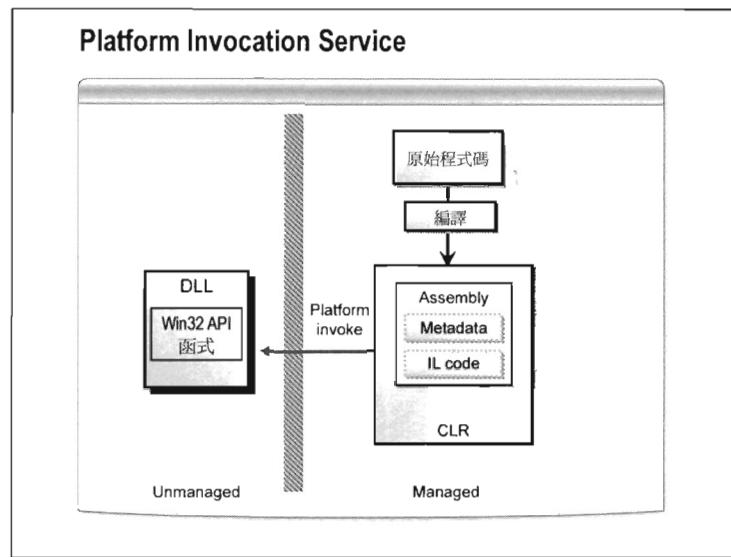
### Win32 API 中常用的 DLL

以下為 Win32 API 中常用的 DLL：

- GDI32.dll：提供與 Graphics Device Interface (GDI) 相關函式，以便在裝置上進行圖形化的輸出，像是繪圖或是寫字。

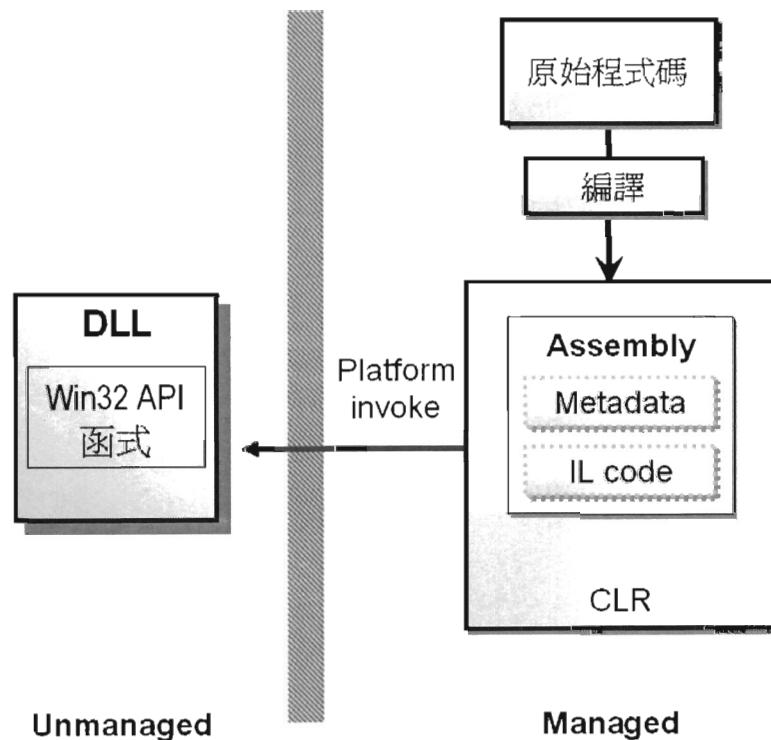
- Kernel32.dll：提供低階的作業系統函式，以管理記憶體及系統資源。
- User32.dll：提供 Windows 作業系統管理函式，像是訊息處理、計時器、選單及行程間的通訊。

~~InterMi~~



## Platform Invocation Service

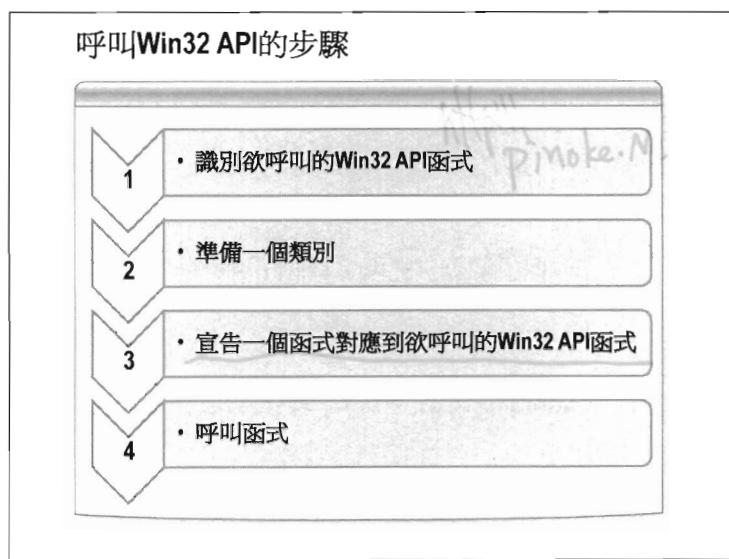
如下圖所示，執行時，Platform invocation service 透過 metadata 得知如何找到匯出的函式及處理引數。



當 Platform invocation service 呼叫一個 Win32 API 時，會依序執行以下的動作：

1. 找到包含此 Win32 API 函式的 DLL。
2. 將 DLL 載入記憶體。
3. 找到記憶體中此函式所在的位址，並將其相關的引數放入堆疊。
4. 將執行權交給 Win32 API 函式。
5. 若發生錯誤，Platform invocation service 會丟出例外。

注意：步驟一及步驟二只在第一次呼叫時做一次。



透過 Platform invocation Service，.NET 程式可以呼叫隱身於 DLL 中的函式。以下為實作的相關步驟：

#### 1. 識別欲呼叫的 Win32 API 函式。

開發人員應知道欲呼叫的 Win32 API 函式儲存在那個 DLL 檔、函式名稱又是什麼。

#### 2. 準備一個類別。

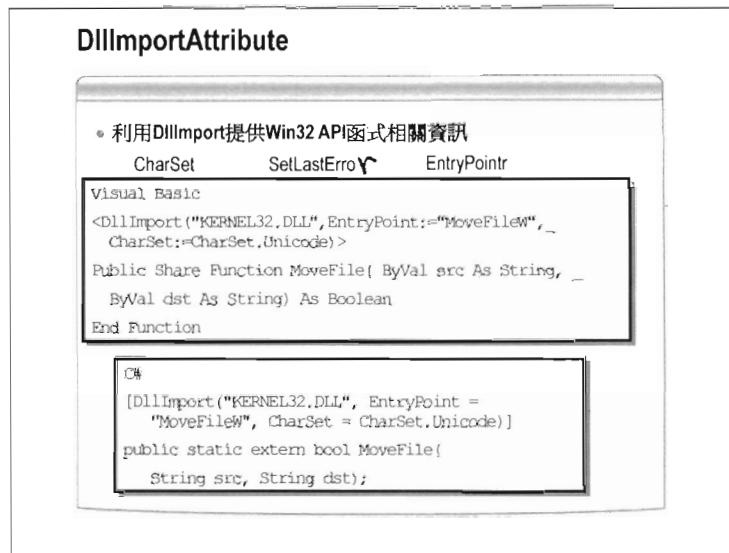
開發人員可使用既有的類別，或是額外建立的類別。以便稍後可在類別中加入一個函式宣告，透過這個函式宣告對應到欲呼叫的 Win32 API 函式。

#### 3. 宣告一個函式對應到欲呼叫的 Win32 API 函式。

在類別中宣告一個函式，並在函式中以 `DllImportAttribute` 指定欲對應的 Win32 API 函式。每一個函式只能對應到一個 Win32 API 函式。函式中不用撰寫任何程式，只要宣告即可。

若程式語言為 Visual Basic 還可用 `Declare` 故述式及 `Lib` 關鍵字，取代 `DllImportAttribute` 指定欲對應的 Win32 API 函式。

#### 4. 呼叫函式。



NameSpace?  
System.Runtime.InteropServices

## DllImportAttribute

在 Visual Basic 程式語言中，欲對應到 Win32 API 的函式必須是個 Shared 的函式，如下所示：

```

Visual Basic
<DllImport("KERNEL32.DLL", EntryPoint := "MoveFileW",
    SetLastError := True, CharSet := CharSet.Unicode)> _
Public Shared Function MoveFile(src As String, dst As String) As Boolean
    '函式中不用撰寫任何程式，只要宣告即可。
End Function

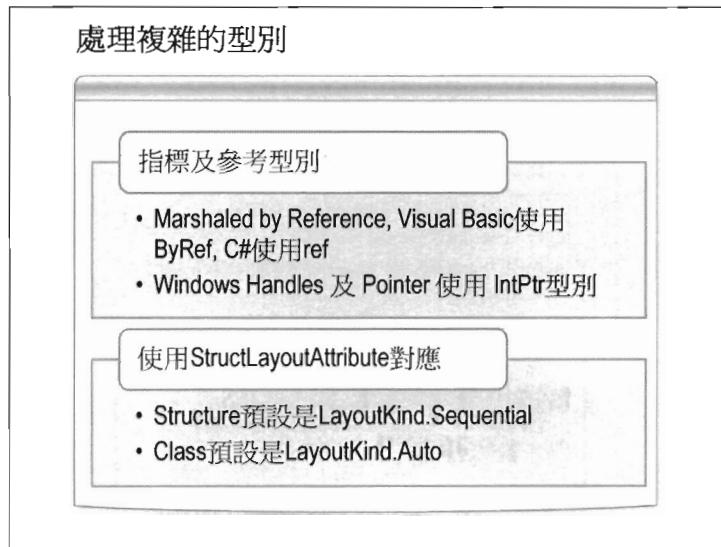
```

在 C# 程式語言中，欲對應到 Win32 API 的函式必須是個 static 且為 extern 的函式（在 C# 中 extern 代表由外部實作的意思），如下所示：

```

C#
[DllImport("KERNEL32.DLL", EntryPoint="MoveFileW", SetLastError=true,
    CharSet=CharSet.Unicode)]
//函式中不用撰寫任何程式，只要宣告即可。
public static extern bool MoveFile(String src, String dst);

```



## 處理複雜的型別

如果型別沒有處理正確將使記憶體配置發生錯誤而使得要執行的函式無法執行。下表是 Windows 型別與.NET 型別的對照表：

Windows 型別	.NET 型別
HANDLE, HWND	System.IntPtr
BYTE	System.Byte
SHORT	System.Int16
WORD	System.UInt16
INT	System.Int32
LONG	System.Int32
BOOL	System.Int32
DWORD	System.UInt32
INT64	System.Int64
CHAR	System.Char
WCHAR	System.Char

在參數傳遞時若是 Marshaled by Reference，那麼 Visual Basic 必須使用 ByRef、C# 使用 ref 關鍵字來傳遞參數。當遇到 Windows Handles 及 Pointer 使用 IntPtr 型別時則使用 System.IntPtr。

## 使用 StructLayoutAttribute 對應

若是型別是 Structure 及 Class 那麼就必須使用 StructLayoutAttribute 來對應到 Managed Code 了，StructLayout 必須傳一個參數 LayoutKind 列舉常數以指定物件的記憶體配置方式，它有三個成員：

- Sequential，按照順序匯出到 Unmanaged 的記憶體。
- Explicit，明確的指定匯出到 Unmanaged 記憶體的配置方式。
- Auto，自動選擇匯出到 Unmanaged 記憶體的配置方式。

Structure 預設是使用 Sequential，而 Class 則是使用 Auto。

以下範例是 C++ 的 Structure，用來指定系統時間、日期：

```
C++
typedef struct _SYSTEMTIME {
    WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME
```

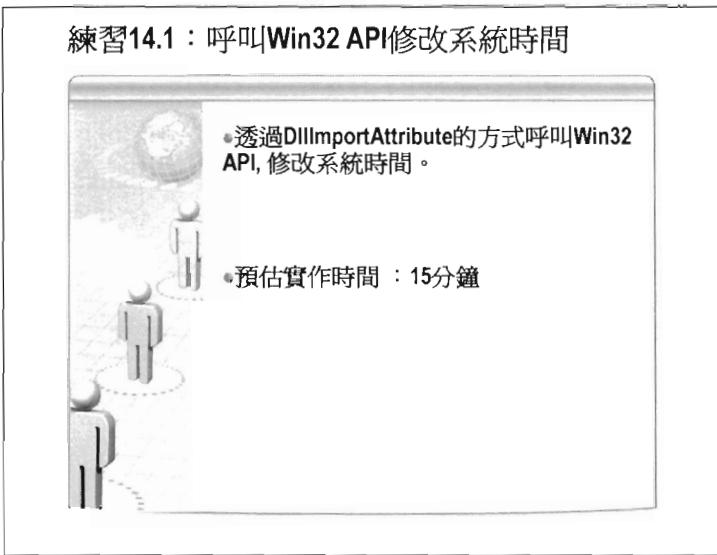
在 Visual Basic 中必須對應成：

```
Visual Basic
<StructLayout(LayoutKind.Sequential)> _
Public Structure SystemTime
    Public wYear As UShort
    Public wMonth As UShort
    Public wDayOfWeek As UShort
    Public wDay As UShort
    Public wHour As UShort
    Public wMinute As UShort
    Public wSecond As UShort
    Public wMilliseconds As UShort
End Structure
```

在 C# 中必須對應成：

```
C#
```

```
[StructLayout(LayoutKind.Sequential)]
public struct SystemTime
{
    public ushort wYear ;
    public ushort wMonth;
    public ushort wDayOfWeek;
    public ushort wDay;
    public ushort wHour;
    public ushort wMinute;
    public ushort wSecond;
    public ushort wMilliseconds;
}
```



## 練習 14.1：呼叫 Win32 API

目的：

透過 DllImportAttribute 的方式呼叫 Win32 API，修改系統時間。

功能描述：

在這個練習中將會透過 DllImportAttribute 的方式呼叫 Kernel32.dll 中的 SetLocalTime。

預估實作時間：15 分鐘

實作步驟：

1. 打開 MSDN 說明文件，使用 Index 查詢「SetLocalTime」及「SYSTEMTIME」。
2. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。請使用系統管理身份執行。
3. 建立一個新的 Windows Application 專案，將專案命名為 Mod14\_1。

4. 在專案中加入一個新的類別，取名為「SystemUtil」。
5. 汇入必要的命名空間。

```
Visual Basic
Imports System.Runtime.InteropServices
```

```
C#
using System.Runtime.InteropServices;
```

6. 在 SystemUtil 類別宣告一個結構 Structure 以對應到 SYSTEMTIME。

```
Visual Basic
<StructLayout(LayoutKind.Sequential)> _
Public Structure SystemTime
    Public wYear As UShort
    Public wMonth As UShort
    Public wDayOfWeek As UShort
    Public wDay As UShort
    Public wHour As UShort
    Public wMinute As UShort
    Public wSecond As UShort
    Public wMilliseconds As UShort
End Structure
```

```
C#
[StructLayout(LayoutKind.Sequential)]
public struct SystemTime
{
    public ushort wYear ;
    public ushort wMonth;
    public ushort wDayOfWeek;
    public ushort wDay;
    public ushort wHour;
    public ushort wMinute;
    public ushort wSecond;
    public ushort wMilliseconds;
}
```

7. 在 SystemUtil 類別中宣告一個函式，並在其前方加上屬性說明要呼叫的 Win32 API：

```
Visual Basic
<DllImport("Kernel32.dll")> _
```

```
Public Shared Function SetLocalTime(ByRef lpSystemTime As SystemTime) As Boolean
End Function
```

```
C#
[DllImport("Kernel32.dll")]
public extern static bool SetLocalTime(ref SystemTime lpSystemTime);
```

8. 開啓 Form1 的設計介面，設計畫面如下圖：



9. 在「設定」的 Click 事件中撰寫以下程式：

- 先判斷輸入的日期是否符合日期格式。
- 將輸入的日期設定到 SystemTime 結構。
- 呼叫 SetLocalTime 設定新的日期。

```
Visual Basic
Dim newDate As DateTime
If Not DateTime.TryParse(TextBox1.Text, newDate) Then
    MessageBox.Show("必須是日期時間的格式")
    Return
End If
Dim time As SystemUtil.SystemTime
```

```

time.wYear = newDate.Year
time.wMonth = newDate.Month
time.wDay = newDate.Day
time.wDayOfWeek = newDate.DayOfWeek
time.wHour = newDate.Hour
time.wMinute = newDate.Minute
time.wSecond = newDate.Second
time.wMilliseconds = newDate.Millisecond

If SystemUtil.SetLocalTime(time) Then
    MessageBox.Show("設定成功")
Else
    MessageBox.Show("失敗")
End If

```

```

C#
DateTime newDate;
if (!DateTime.TryParse (TextBox1.Text, out newDate ))
{
    MessageBox.Show("必須是日期時間的格式");
    return;
}
SystemUtil.SystemTime time ;
time.wYear = (ushort)newDate.Year ;
time.wMonth = (ushort)newDate.Month;
time.wDay = (ushort)newDate.Day;
time.wDayOfWeek = (ushort)newDate.DayOfWeek;
time.wHour = (ushort)newDate.Hour;
time.wMinute = (ushort)newDate.Minute;
time.wSecond = (ushort)newDate.Second;
time.wMilliseconds = (ushort)newDate.Millisecond;

if (SystemUtil.SetLocalTime(ref time))
    MessageBox.Show("設定成功");
else
    MessageBox.Show("失敗");

```

10. 執行結果如下：



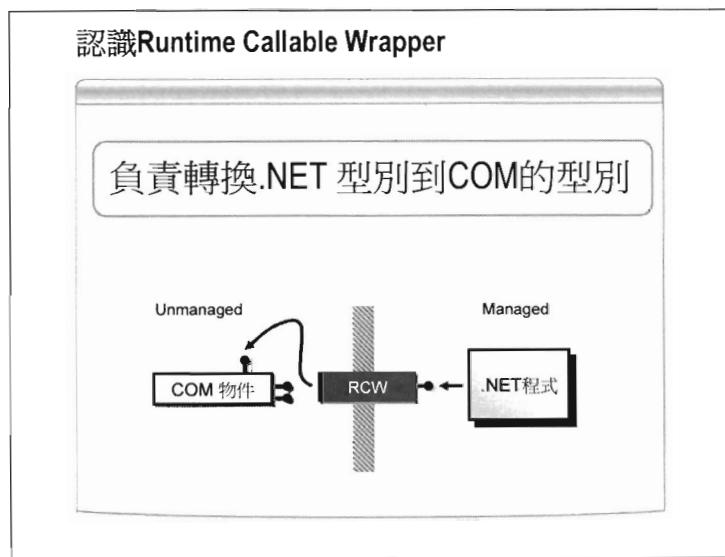


---

COM (Component Object Model) 技術可讓物件將本身的功能提供給其他元件使用，而且早在.NET Framework 出現之前，COM 技術在 Windows 環境中也會風靡一時，因此在一些情況下需要將.NET 及 COM 元件進行整合，以下內容便將介紹.NET 與 COM 元件的整合方式及作法。

這一節包含以下內容：

- 認識 COM Callable Wrapper
- 如何設計.NET 元件給 COM 使用
- 如何建立.NET 元件給 COM 使用
- 如何部署給 COM 使用的.NET 元件



## 認識 Runtime Callable Wrapper

什麼是 Runtime Callable Wrapper ? 在認識 Runtime Callable Wrapper 之前你必須先了解 COM 元件與.NET 元件的差異性。

### COM 元件與.NET 元件的差異

COM 與.NET Framework 的物件模型有幾個不同處：

使用 COM 物件的程式必須負起管理這些 COM 物件生命週期的責任；在.NET 中物件生命週期的管理是由 CLR 執行環境中的 GC 統一負責，一般程式不用處理。

使用 COM 物件若要知道物件是否提供某項功能必須向提供此功能的介面進行查詢，若物件提供此功能便會傳回一個介面指標否則便不傳回；在.NET 中若要知道物件是否具備某項功能可利用 Reflection 方式查詢。

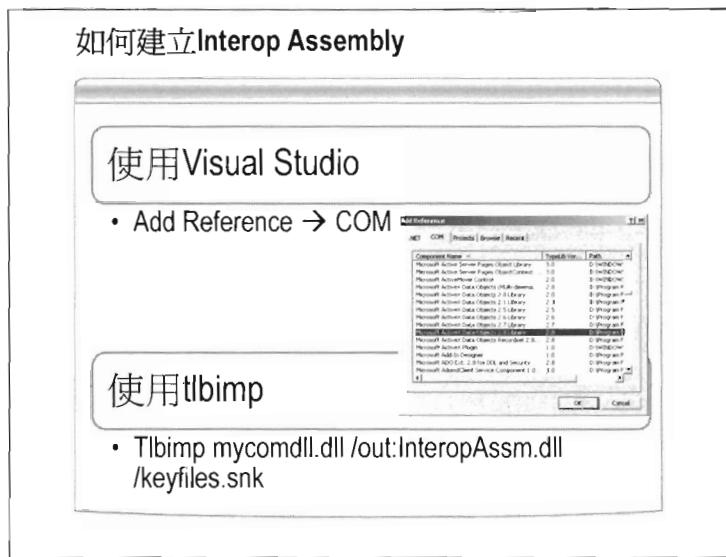
所有.NET 物件都是由.NET Framework 執行環境統一管理（Managed code），執行環境可根據執行效能的考量，重新規劃.NET 物件在記憶體中的位置，將物件遷移到新位址後再更新物件的參考位址。然而使用 COM 物件的程式（相對於.NET 的 Managed Code），就是

Unmanaged code) 會將物件的指標位址記住，因而與物件所在的位址產生相依性，COM 物件不可以隨意遷移。

若要整合.NET 與 COM 元件，勢必得想辦法解決兩者的差異。

## Wrapper class

.NET 執行環境提供的解決方法是透過 wrapper class 將兩邊的元件加以包裝，讓 managed 及 unmanaged 程式把元件看成是與本身同類的程式。當.NET 程式呼叫 COM 物件時，.NET 執行環境會建立一個 runtime callable wrapper (RCW)，由 RCW 負責整合 managed 與 unmanaged 環境的不同處。當 COM 程式呼叫.NET 物件時，.NET 執行環境會產生一個 COM callable wrapper (CCW) 進行反向操作，讓 COM 程式以為所呼叫的.NET 物件其實是 COM 物件。



## 建立 Interop Assembly

大部分的 COM 元件其 Type Library 就包含在其.dll 檔案中，在 Visual Studio 開發環境參考 COM 元件的方法和以前在 Visual Studio 6.0 的步驟差不多，但在私底下 Visual Studio 會呼叫 TlbImp.exe 工具程式為指定的 COM 元件建立對應的 interop assembly。.NET 程式對 COM 物件的呼叫其實是先被引導到 interop assembly 再交給實際的 COM 物件；而 COM 物件回應的結果也是先被引導到 interop assembly 再交給.NET 程式。

參考 COM 物件

以下為於 Visual Studio 專案中參考 COM 物件的步驟：

1. 點選 Visual Studio 主選單中的「Project」，再點選其下的「Add Reference」，於跳出的對應話窗中切換到「COM」頁籤。
  2. 自清單中選取欲參考的 COM 元件。

若是透過 Visual Studio 參考 COM 元件，開發工具會自動在私底下呼叫 Tlbimp.exe 工具程式為你建立 interop assemblies。

## 使用 Tlbimp.exe 工具程式建立 **interop assembly**

不透過開發工具也可自行在命令提示字元模式下利用 Tlbimp.exe 工具  
程式手動產生 interop assemblies。

以如下指令為例，DemoCOMTypeLib.dll 中必須包含 COM 元件的  
Type Library（COM 元件的實際程式內容與其 Type Library 可以放在  
不同的檔案中），/out 指定產生的 interop assembly 檔名為  
DemoInterop.dll：

```
Tlbimp DemoCOMTypeLib.dll /out:DemoInterop.dll
```

DemoCOMTypeLib.dll 對應的 COM 元件必須先以 Regsvr32.exe 向作  
業系統進行註冊，程式執行時才能成功建立及使用 COM 物件。

## 如何存取 COM 物件

建立 COM 物件，然後呼叫物件成員

- 使用 new 關鍵字

```
Visual Basic
Dim Player As New WindowsMediaPlayer
Player.openPlayer(fileName)
```

```
C#
WindowsMediaPlayer Player = new WindowsMediaPlayer();
Player.openPlayer(fileName);
```

釋放 COM 物件

- 使用 Marshal 類別的 ReleaseComObject 方法

## 如何存取 COM 物件

建立 Interop Assembly 之後，接下來必須匯 COM 類別的命名空間，以下這個範例會使用到 Windows Word 的 Interop 物件，它的命名空間是：

```
Visual Basic
Imports Word = Microsoft.Office.Interop.Word
```

```
C#
using Word = Microsoft.Office.Interop.Word;
```

匯入 COM 類別的命名空間之後，必須使用 New 關鍵字建立物件實體，接著呼叫它的相關成員，這個範例會建立 Word 應用程式物件，使用 Visible 屬性讓 Word 應用程式呈現在螢幕上，使用 Documents 屬性的 Open 方法開啟指定的文件。

```
Visual Basic
Dim wordAP As Word.Application
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    If wordAP Is Nothing Then
        wordAP = New Word.Application
```

```

End If
wordAP.Visible = True
wordAP.Documents.Open(Application.StartupPath & "\mydoc.docx")
End Sub

```

**C#**

```

Word.Application wordAP ;
object missing = Type.Missing ;
private void button2_Click(object sender, EventArgs e)
{
    if (wordAP == null)
        wordAP = new Word.ApplicationClass();

    object fileName = Application.StartupPath + @"\mydoc.docx";
    wordAP.Visible = true;
    wordAP.Documents.Open(ref fileName,
        ref missing, ref missing, ref missing, ref missing,
        ref missing, ref missing, ref missing, ref missing,
        ref missing, ref missing, ref missing, ref missing);
}

```

如果 COM 物件是用 Visual Basic 語法寫的像是 Microsoft Office 系列，當使用 C#語呼叫時遇到 Visual Basic 的可省略參數語法，C#就必須特別處理。以此例而言 Open 方法的可省略參數都必須處理成 Type.Missing (其他案例則可能要處理成 null 值)，故先宣告一個物件型別的 missing 變數指定為 Type.Missing。

## 釋放 COM 物件

將 COM 物件的變數指向 Nothing(for Visual Basic) 或 null (for C#)只能讓 Managed Code 記憶體部份去處理釋放行為，若要馬上釋放 Unmanaged Code 的 COM 物件必須透過 Marshal 類別的(命名空間：System.Runtime.InteropServices) ReleaseComObject 方法。

以下範例是釋放 Word Application 的 COM 物件：

```

Visual Basic
Private Sub CloseWordApp()
    Try
        wordAP.Quit()
    Catch
    End Try
    Marshal.ReleaseComObject(wordAP)
    wordAP = Nothing
End Sub

```

```
C#
private void CloseWordApp()
{
    try
    {
        wordAP.Quit(ref missing, ref missing, ref missing);
    }
    catch
    {
    }
    Marshal.ReleaseComObject(wordAP);
    wordAP = null;
}
```

Quit 方法是結束 Word Application 的方法，使用者可能先行關閉，故使用 Try...Catch 語法處理可能發生的例外狀況。



## 練習 14.2：在.NET 中呼叫 COM 元件

目的：

在 Visual Studio 開發環境中參考並使用 COM 元件。

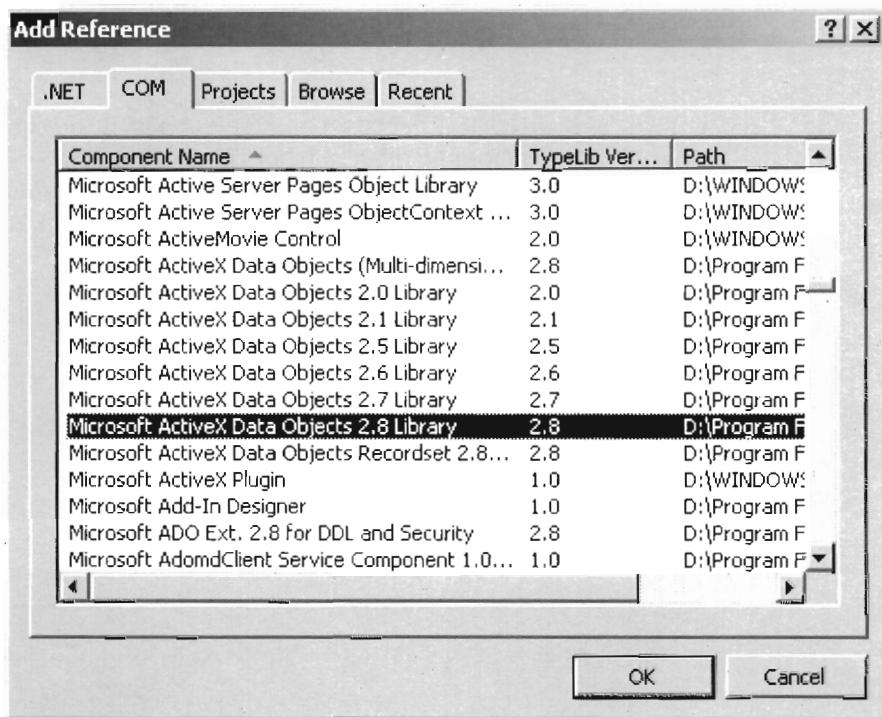
功能描述：

在這個練習中你將學會如何在 Visual Studio 開發環境中參考並使用 ADODB 的 COM 元件。

預估實作時間：10 分鐘

實作步驟：

1. 打開 Visual Studio 開發工具，建立 Console 專案存放於「磁碟:\U2956\Practices\VB 或 CS」，專案名稱為「Mod14\_2」。
2. 在「Solution Explorer」視窗中的「Mod14\_2」上按下滑鼠右鍵，選擇「Add Reference...」。
3. 在「Add Reference」視窗中切換到「COM」頁籤，點選其下的「Microsoft ActiveX Data Objects 2.8 Library」，按下「OK」。



4. 打開「Module1.vb」找到 Sub Main(Visual Basic)，或  
 「Program.cs」找到「static void Main」，於其中輸入以下程  
 式：(登入方式請自行設定)

```

Visual Basic
Sub Main
    Dim cn As New ADODB.Connection
    cn.Open("Provider=SQLOLEDB;Data Source=localhost;" &
            "Initial Catalog=pubs;user id=sa;password=123;")

    Dim rs As New ADODB.Recordset
    rs.Open("select * from titles", cn)

    Do While Not rs.EOF
        Console.WriteLine(rs.Fields(1).Value)
        rs.MoveNext()
    Loop

    rs.Close()
    cn.Close()
    Marshal.ReleaseComObject(rs)
    Marshal.ReleaseComObject(cn)
End Sub

```

```

C#
static void Main(string[] args)
{
    ADODB.Connection cn = new ADODB.Connection();
    cn.Open("Provider=SQLOLEDB;Data Source=localhost;");
}

```

```

        + "Initial Catalog=pubs;user id=sa;password=123;",
        "", "", -1);

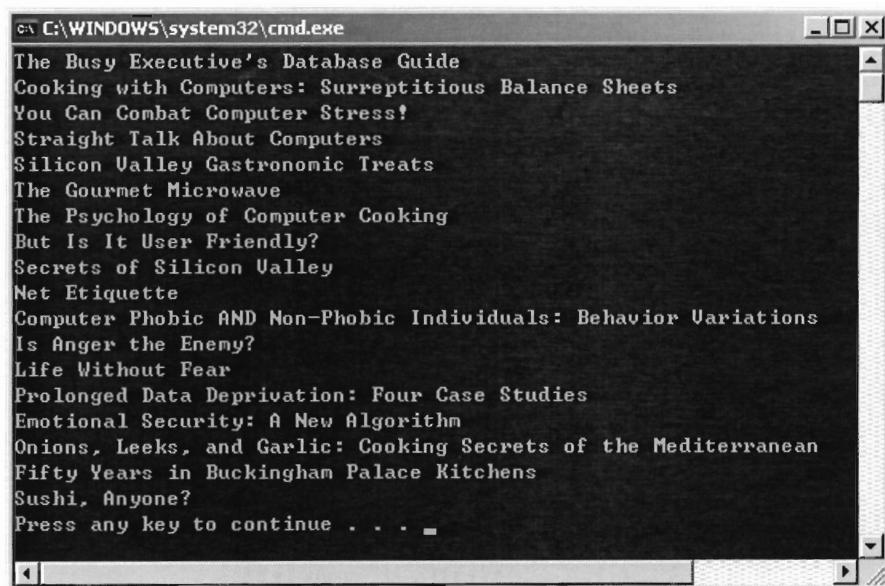
ADODB.Recordset rs =new ADODB.Recordset();
rs.Open("select * from titles",
        cn,ADODB.CursorTypeEnum.adOpenStatic,
        ADODB.LockTypeEnum.adLockReadOnly,-1 );

while (! rs.EOF){
    Console.WriteLine(rs.Fields[1].Value);
    rs.MoveNext();
};

rs.Close();
cn.Close();
Marshal.ReleaseComObject(rs);
Marshal.ReleaseComObject(cn);
}

```

5. 點選主選單「Debug」下的「Start Without Debugging」，確認命令提示字元視窗中是否出現如下的訊息視窗。



### COM元件呼叫.NET 元件

認識 COM Callable Wrapper

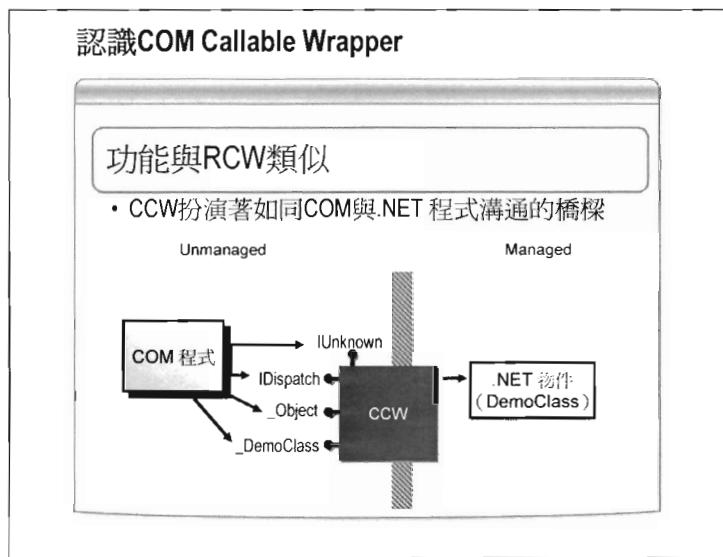
如何設計.NET 元件給 COM 使用

如何建立.NET 元件給 COM 使用

如何部署給 COM 使用的.NET 元件

若希望.NET 元件可在 COM 環境中使用，在設計時必須注意一些事項。以下課程將說明如何設計可供 COM 呼叫的.NET 元件，內容包含：

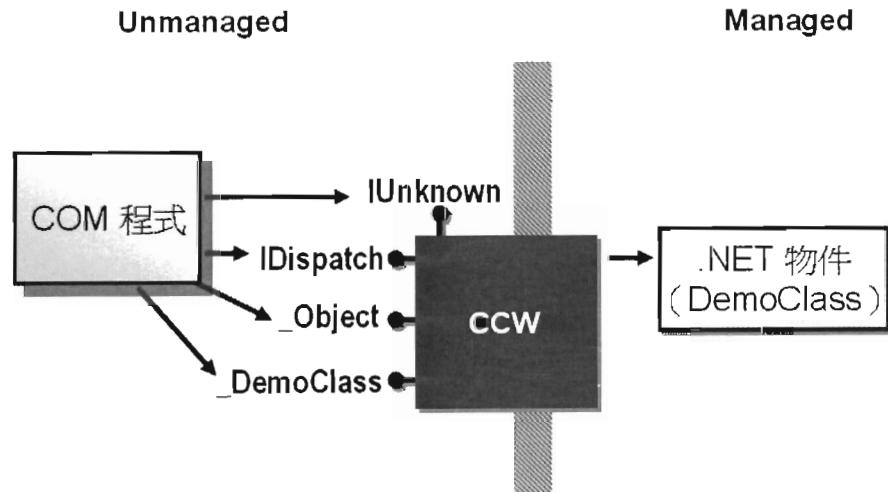
- 認識 COM Callable Wrapper
- 如何設計.NET 元件給 COM 使用
- 如何建立.NET 元件給 COM 使用
- 如何部署給 COM 使用的.NET 元件



## 認識 COM Callable Wrapper

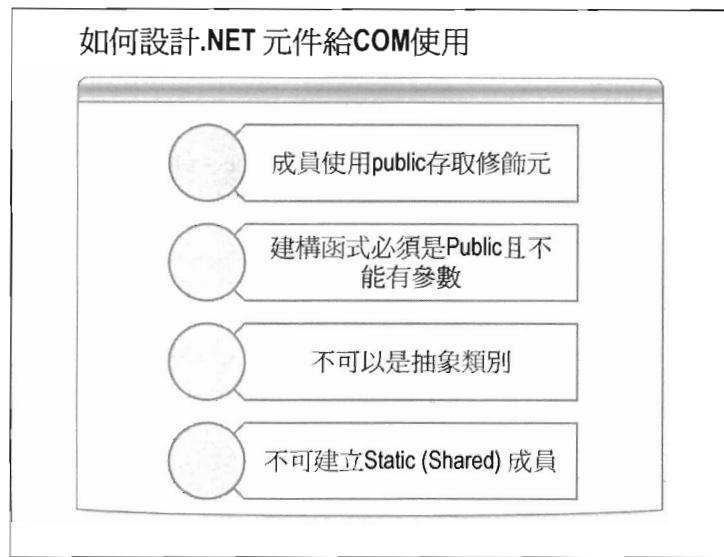
CCW ( COM callable wrapper ) 會提供所有 public 、 COM 可以看得到的介面、資料型別及傳回值給 COM 程式，就像所有的 COM 物件一樣。在 COM 的世界中，物件間的互動主要是透過介面，為讓.NET 物件符合這個原則，CCW 便要為.NET 物件模擬出 COM 介面都會有的 IUnknown 及 IDispatch 。

如下圖所示，CCW 對其包裝的.NET 物件只會保持一個參考值，所有 COM 程式對此.NET 物件的存取一律透過 CCW :



由於.NET 類別並不強制一定得實作什麼介面，因此透過 Tlbexp.exe 建立 Type library 時，會自動為.NET 類別產生一些介面稱之為「Class Interface」。每個 Class Interface 都只包含 public 的方法、屬性、資料欄位及事件。預設 Class Interface 的名稱為類別名稱前加上一個底線，以類別 DemoClass 為例，其 Class Interface 即為 \_DemoClass。

在物件導向的作用下，子類別涵蓋其父類別的功能；因此以 DemoClass 為例，若其父類別為 DemoBaseClass、DemoBaseClass 的父類別為 System.Object，則在 Tlbexp.exe 為 DemoClass 類別產生的 Type library 中將包含三個 Class Interface：\_DemoClass、\_DemoSupperClass 及 \_Object。



## 設計可讓 COM 呼叫的.NET 元件

若要開放.NET 元件給 COM 使用，請在設計.NET 元件時注意以下事項，以確保 COM 程式可以成功地建立呼叫：

- 成員使用 public 存取修飾詞

所有要給 COM 使用的型別、方法、屬性及欄位都必須是 public，因為匯出成 Type Library 時只會記錄 public 的成員及類別，所以 COM 程式只能看到 public 的成員。Static 靜態成員及常數欄位不會匯出。

另外即使是 public 成員也可利用 ComVisibleAttribute，限制某些成員是否可在 COM 中看到。

- 建構函式必須是 Public 且不能有參數

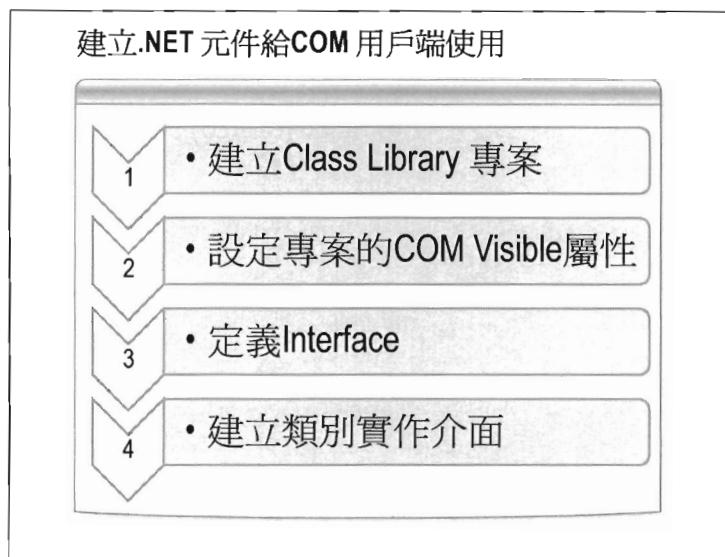
即使是 public 的類別，但若沒有可供 COM 呼叫的建構函式，仍舊無法直接使用（可透過間接的方式建立）。每個要給 COM 使用的類別都要有一個 public、沒有參數的建構函式，因為這是 COM 程式唯一可以呼叫的建構函式。

- 不可以是抽象類別

無論在 COM 或是在.NET 中都無法為抽象類別建立物件。

- 不可建立 Static (Shared) 成員

COM 不支援使用 Static (Shared) 成員。



## 建立.NET 元件給 COM 用戶端使用

建立.NET 元件給 COM 用戶端在 Visual Studio 2008 的開發工具上是一件非常容易的事，只要以下幾個步驟即可：

1. 建立 Class Library 專案。
2. 勾選專案的 Make assembly COM-Visible 屬性。
3. 定義 Interface。

```

Visual Basic
Interface IStudent
    Property FullName() As String
    Function GetScore() As Integer
End Interface

```

```

C#
interface IStudent
{
    string FullName
    {
        get;
        set;
    }
    int GetScore();
}

```

```
}
```

4. 建立類別實作介面。

```
Visual Basic
Public Class Student
    Implements IStudent

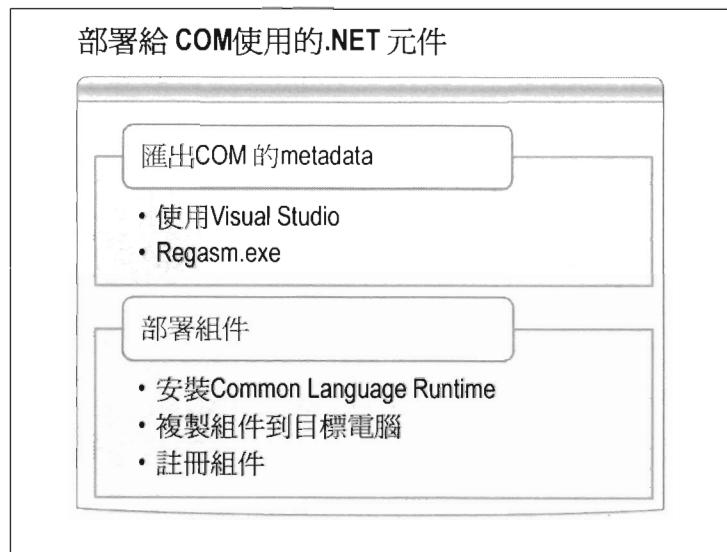
    Dim _fullName As String
    Public Property FullName() As String Implements IStudent.FullName
        Get
            Return _fullName
        End Get
        Set(ByVal value As String)
            _fullName = value
        End Set
    End Property

    Public Function GetScore() As Integer Implements IStudent.GetScore
        Dim rnd As New Random
        Dim score As Integer
        score = rnd.Next(60, 100)
        Return score
    End Function
End Class
```

```
C#
public class Student : IStudent
{
    #region IStudent 成員
    string fullName;
    public string FullName
    {
        get
        {
            return fullName;
        }
        set
        {
            fullName = value;
        }
    }

    public int GetScore()
    {
        Random rnd = new Random();
        int score = rnd.Next(60, 100);
        return score;
    }
}
```

```
    }  
    #endregion  
}
```



## 部署給 COM 使用的 .NET 元件

COM 用戶端呼叫元件是會先透過註冊機碼尋找元件的相關資訊，像是 metadata 及元件檔的所在位置，因此.NET 的元件要給部署時，必須先進行註冊。一般的 COM 元件可使用 Regsvr32.exe 註冊，.NET 的組件就不能使用 Regsvr32.exe，它有兩種註冊方式：

- 使用 Visual Studio

如果是在開發階段，可以使用 Visual Studio 專案的 Register for COM Interop 選項 (Visual Basic 是在專案屬性視窗的 Compiler 頁籤，C# 則是在 Build 頁籤)。勾選 Register for COM Interop 項目是會在建置專案時自動將組件註冊成 COM 元件。

- 使用 Regasm.exe

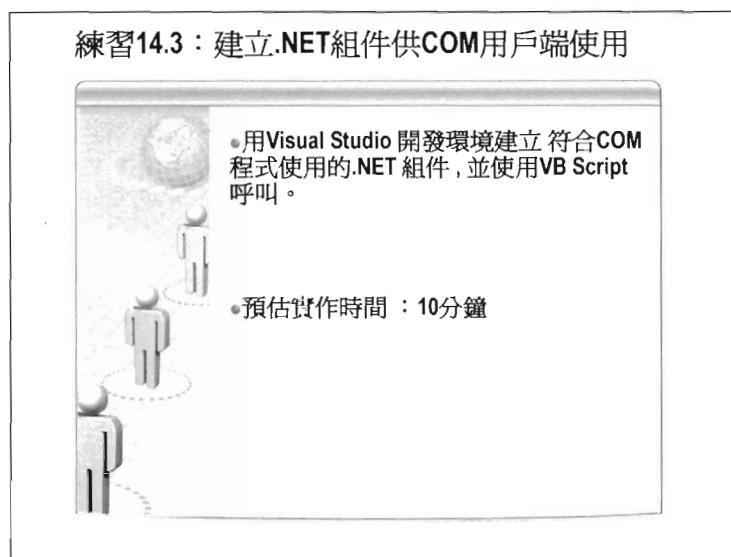
如果是在沒有安裝 Visual Studio 的電腦上，例如用戶端的電腦，那麼則是使用命令提示列指令 – Regasm.exe，這個工具附屬在.NET SDK 中，可以直接複製到用戶端的電腦，它的命令參數如下：

```
Regasm MyLibrary.dll /codebase
```

## 部署組件

部署.NET 所寫的 COM 元件時必須進行以下步驟：

1. 在用戶端的電腦必須要安裝 CLR (Common Language Runtime)。
2. 複製組件到目標電腦。
3. 註冊組件。



### 練習 14.3：建立.NET 組件供 COM 用戶端使用

目的：

用 Visual Studio 開發環境建立符合 COM 用戶端使用的.NET 組件，  
並使用 VB Script 呼叫。

功能描述：

在這個練習中將會使用 Visual Studio 開發類別庫，並依循 COM 可呼  
叫的規定建立組件，然後使用 VB Script 呼叫這個組件。

預估實作時間：15 分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」  
→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發  
環境。請使用系統管理身份執行。
2. 建立一個新的 Class Library 專案，將專案命名為 Mod14\_3。
3. 打開 Class1.vb 或 Class1.cs 的檔案。

4. 在程式檔裡加入 Interface，包含 CompanyName 屬性及 GetStockPrice 方法：

```
Visual Basic
Public Interface ICustomer
    Property CompanyName() As String
    Function GetStockPrice() As Double
End Interface
```

```
C#
interface ICustomer
{
    string CompanyName { get; set; }
    double GetStockPrice();
}
```

5. 實作 ICustomer 介面，類別名稱是 Customer：

```
Visual Basic
Public Class Customer
    Implements ICustomer
    Dim _companyName As String
    Public Property CompanyName() As String Implements ICustomer.CompanyName
        Get
            Return _companyName
        End Get
        Set(ByVal value As String)
            _companyName = value
        End Set
    End Property

    Public Function GetStockPrice() As Double Implements ICustomer.GetStockPrice
        Dim rnd As New Random
        Dim stockPrice As Double
        stockPrice = rnd.Next(1, 1000)
        Return stockPrice
    End Function
End Class
```

```
C#
public class Customer : ICustomer
{
    #region ICustomer 成員
    string companyName;
```

```

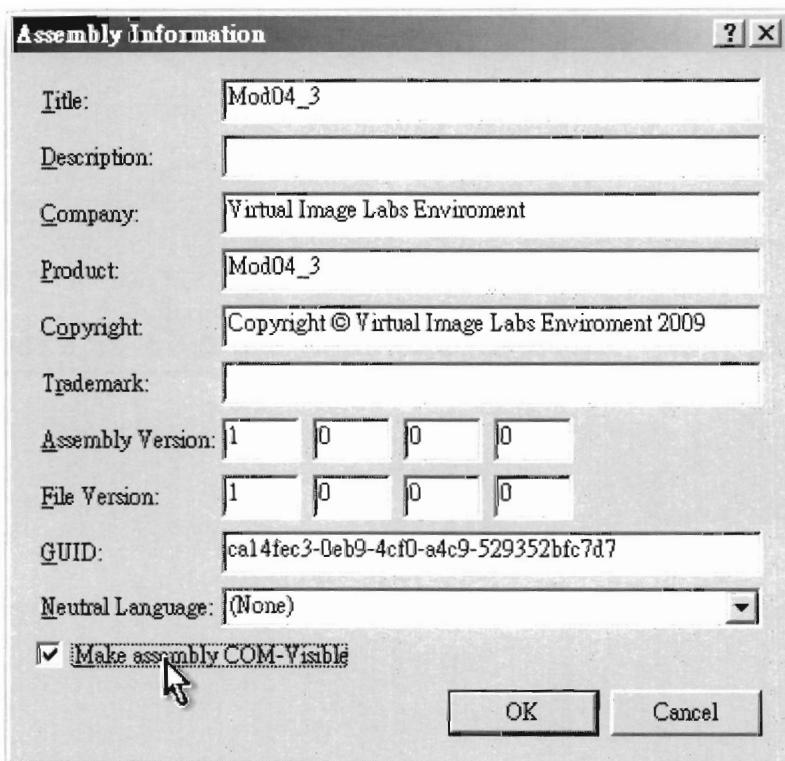
public string CompanyName
{
    get
    {
        return companyName;
    }
    set
    {
        companyName = value;
    }
}

public double GetStockPrice()
{
    Random rnd = new Random();
    double stockPrice = rnd.Next(1, 1000);
    return stockPrice;
}

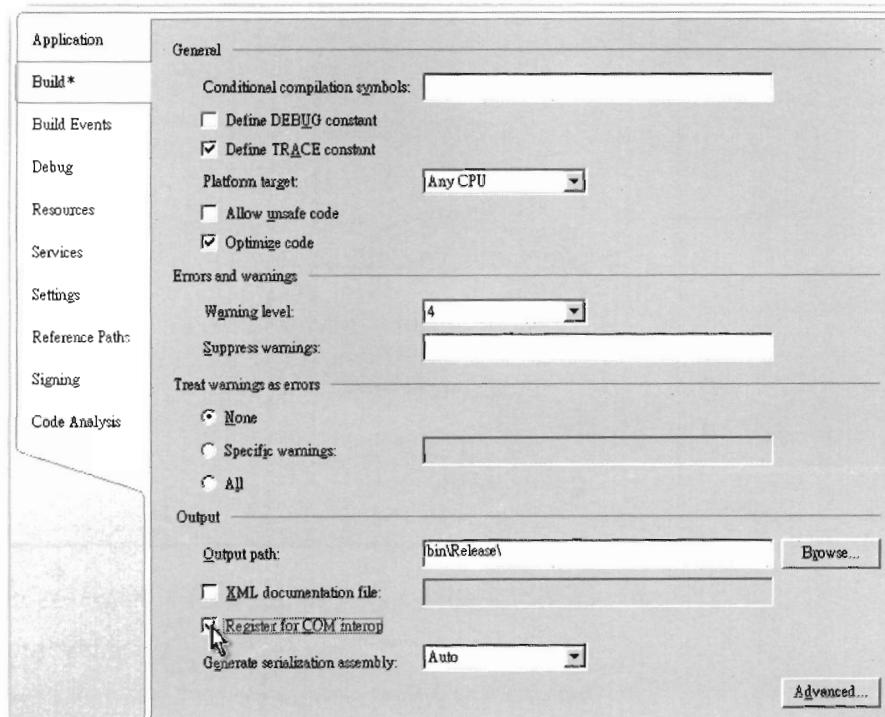
#endregion
}

```

6. 開啓專案屬性，在「Application」頁按「Assembly Information ...」按鈕，接著在「Assembly Information ...」畫面找到「Make assembly COM-Visible」項目並勾選之。



7. 切到「Compiler」(for Visual Basic)或「Build」(for C#)，  
勾選「Register for COM interop」項目。

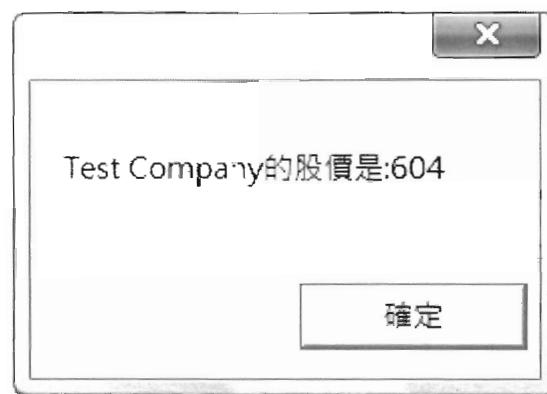


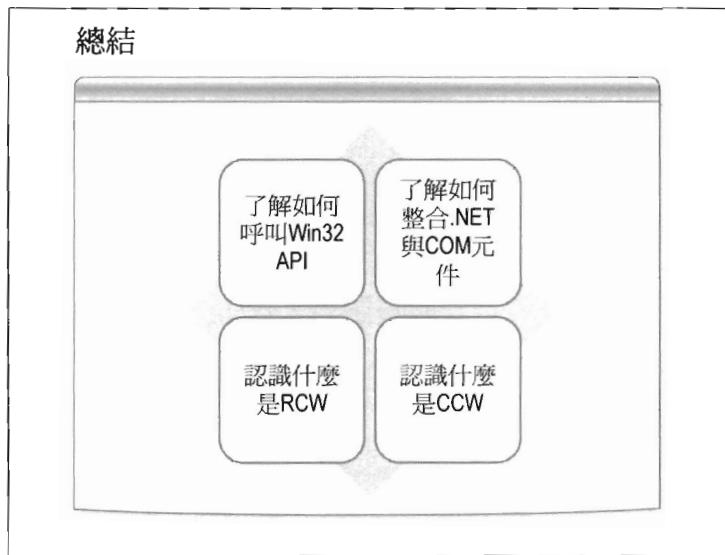
8. 建置專案。(會在建置時自動註冊)

9. 使用「Notepad」編寫 VB Script 呼叫 Mod14\_3.Customer 元件，程式碼如下：

```
Dim obj
Set obj = CreateObject("Mod14_3.Customer")
obj.CompanyName = "Test Company"
MsgBox obj.CompanyName & "的股價是:" & obj.GetStockPrice()
```

10. 接著存檔為 CallNet.vbs，然後從檔案總管執行 CallNet.vbs，  
執行結果如下：





---

# 第十五章：電子郵件程 式設計

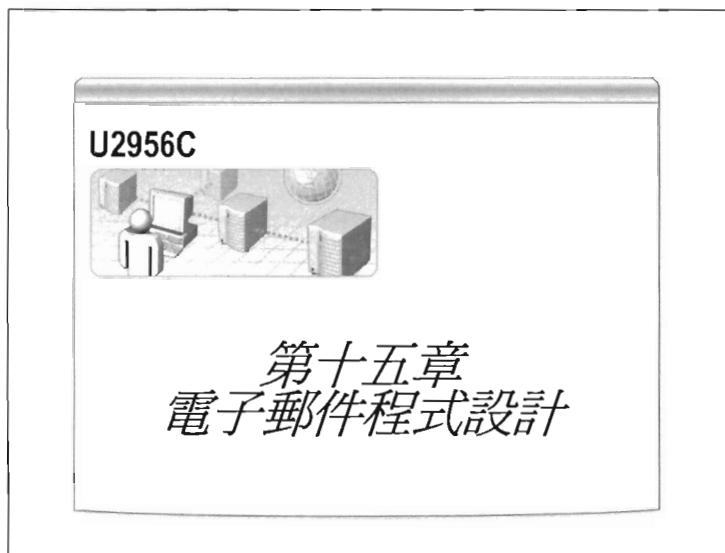
## 本章大綱

.NET Framework 的 Mail 機制.....	4
建立及傳送 E-mail 訊息的程序.....	5
建立 Mail 訊息.....	6
使用 SmtpClient 傳送 Mail.....	8
練習 15.1：使用 SmtpClient 傳送 Mail .....	10
在組態檔設定 SmtpClient .....	14
指定多個收件者.....	16
在電子郵件中附加檔案.....	17
練習 15.2：在傳送的 Mail 中附加檔案.....	18
建立 HTML E-Mail.....	21
利用非同步傳送 Mail.....	22
非同步傳送 Mail 設計步驟 .....	23
練習 15.3：使用非同步方式傳送 Mail.....	25

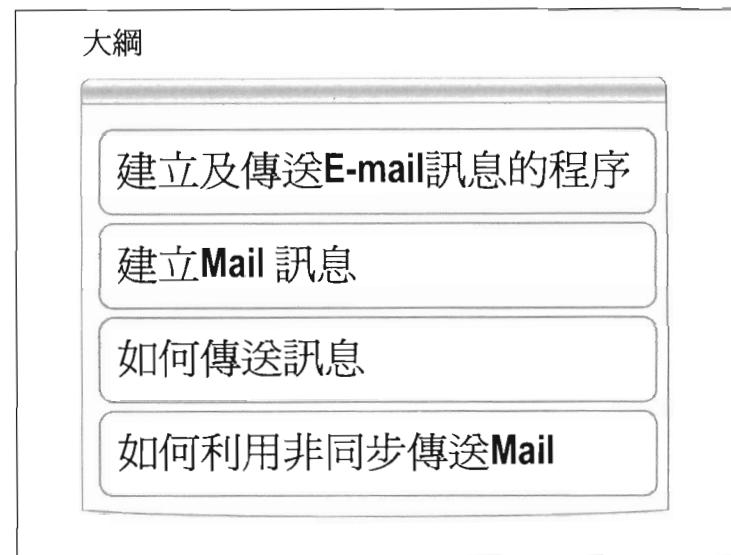
作者：

羅慧真





---

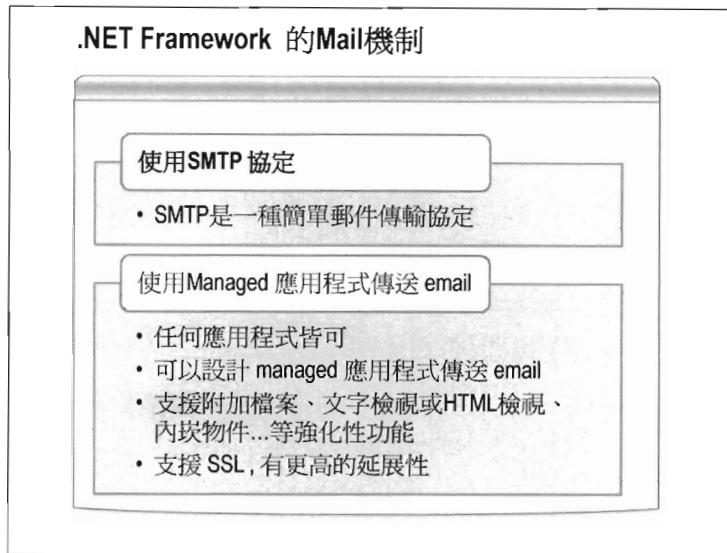


---

利用程式傳送電子郵件可以設計的很簡單，也可以設計的很複雜。譬如說：簡單的傳送電子郵件機制只要單純的設定收件人、標題、內容、送件人就可以傳送一封文字類型的電子郵件，但是如果要傳送的電子郵件還要考量其編碼、附加檔案、文件格式等等，就得多方面的考量和設計。

在這個章節中將學習到：

- 建立及傳送 E-mail 訊息的程序
- 建立 Mail 訊息
- 如何傳送訊息
- 如何利用非同步傳送 Mail



## .NET Framework 的 Mail 機制

若要從應用程式傳送電子郵件，則必須在伺服器上安裝及設定網際網路資訊服務 (IIS) 的 Simple Mail Transfer Protocol (SMTP) 服務。IIS SMTP 服務是一個簡單的元件，用來將電子郵件轉寄到 SMTP 伺服器進行傳遞。

### 應用程式如何發送 Mail ？

可以使用.NET 程式語言如 Visual Basic 或是 C# 來開發應用程式，透過 System.Net.Mail 命名空間下的相關物件來設計，即可達到傳送 Mail 的功能，且在.NET Framework 2.0 以後的版本中傳送 Mail 不再僅限用於網站應用程式的設計。

### 新增的強化功能

另外，針對設計電子郵件的功能面設計也做了許多強化，例如可以內嵌物件到欲傳送的 Mail 中或是附加多個檔案、使用不同的檢視模式 (HTML 或是一般文字模式) 等等...。

除了以上的基本功能強化外，更可以設定透過 SSL 來加密連線，讓傳送的 Mail 更安全以增加應用程式設計的延展性。



## 建立及傳送 E-mail 訊息的程序

在開發應用程式要執行傳送郵件功能時，有以下幾個程序設計：

- 首先建立一個 MailMessage 物件

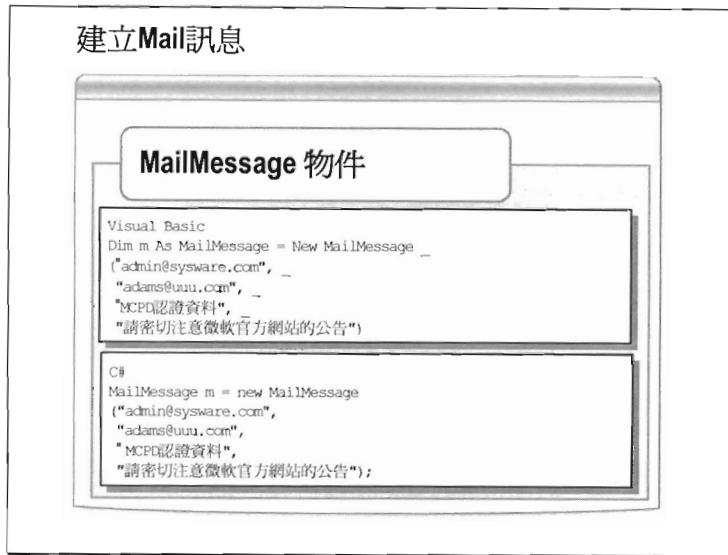
MailMessage 是 System.Net.Mail 命名空間下的物件，主要用來表示電子郵件訊息的內容。有需要的話再加上附件檔案，可以使用 Attachment 類別建立郵件附件。

- 建立 SmtpClient 物件並指定 SMTP 伺服器

SmtpClient 類別會將電子郵件傳輸至所指定之郵件傳遞用途的 SMTP 主機。

透過 SmtpClient 物件在 SMTP 伺服器上認證，如果 SMTP 伺服器需要安全驗證的話，則必須要將合法的使用者身份透過 SmtpClient 物件傳送給 SMTP 伺服器。

使用 SmtpClient 的 Send 方法傳送 Mail，使用 SmtpClient 物件來傳送電子郵件有兩種傳送方式，分別為同步與非同步，若要使用同步方式傳送郵件則叫用 Send 方法，反之若要使用非同步方式傳送郵件則叫用 SendAsync 方法。



## 建立 Mail 訊息

使用 MailMessage 物件設計欲傳送的電子郵件訊息相關內容，而這些訊息會使用 SmtpClient 類別傳送到 SMTP 伺服器進行傳送，因此若要指定電子郵件訊息的寄件者、收件者和內容等資料，可設計 MailMessage 類別的關聯屬性。

下列表格中為常用的郵件訊息資料設定：

電子郵件部份	屬性
寄件者	From
收件者	To
主旨	Subject
訊息主體	Body
副本 (CC)	CC
密件副本 (BCC)	Bcc
附件	Attachments

MailMessage 為 System.Net.Mail 命名空間下的類別，因此在撰寫程式建立執行個體前可以先在應用程式中匯入 System.Net.Mail 命名空間，接著建立物件將其相關的參數如：寄件者、收件者和內容等在建構函式中傳入或是透過屬性指定皆可。

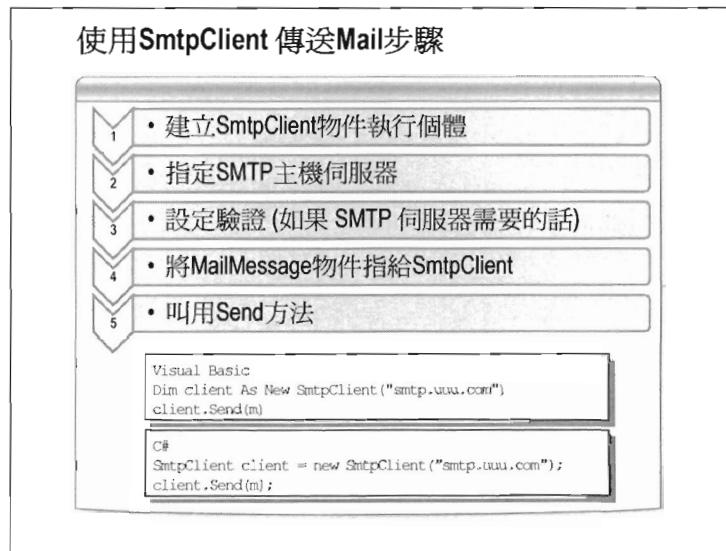
下列程式碼為建立 MailMessage 物件並傳入建構函式參數的建立方式，第一個參數為送件者地址、第二個參數為收件者地址、第三個參數為郵件標題、第四個參數為郵件內容。

Visual Basic

```
Dim m As MailMessage = New MailMessage _  
("admin@sysware.com", _  
"adams@uuu.com", _  
"MCPD認證資料", _  
"請密切注意微軟官方網站的公告")
```

C#

```
MailMessage m = new MailMessage  
("admin@sysware.com",  
"adams@uuu.com",  
"MCPD認證資料",  
"請密切注意微軟官方網站的公告");
```



## 使用 SmtpClient 傳送 Mail

SmtpClient 類別是用來傳送電子郵件，下列幾個類別可以用來設定使用 SmtpClient 傳送的電子郵件內容：

- Attachment 類別：表示檔案附件，這個類別可在電子郵件訊息中附加檔案、資料流或文字。
- MailAddress 類別：表示寄件者和收件者的電子郵件地址。
- MailMessage 類別：表示電子郵件訊息。

在 SmtpClient 的執行個體中指定的以上這幾個類別所建立的相關資料後，就可以將電子郵件傳送出去。若要使用 SmtpClient 傳送 Mail，可透過以下步驟來執行：

1. 建立 SmtpClient 物件執行個體
2. 指定 SMTP 主機伺服器
3. 設定驗證 (如果 SMTP 伺服器需要的話)
4. 將 MailMessage 物件指給 SmtpClient
5. 叫用 Send 方法

下列程式碼為建立 MailMessage 物件後，透過 SmtpClient 物件執行個體指定好 SMTP 主機伺服器後，叫用 Send 方法執行傳送。

Visual Basic

```
Dim m As MailMessage = New MailMessage _  
("admin@sysware.com", _  
"adams@uuu.com", _  
"MCPD認證資料", _  
"請密切注意微軟官方網站的公告")  
Dim client As New SmtpClient("smtp.uuu.com")  
client.Send(m)
```

C#

```
MailMessage m = new MailMessage  
("admin@sysware.com",  
"adams@uuu.com",  
"MCPD認證資料",  
"請密切注意微軟官方網站的公告");  
SmtpClient client = new SmtpClient("smtp.uuu.com");  
client.Send(m);
```



## 練習 15.1 : 使用 SmtpClient 傳送 Mail

目的：

練習如何在 Winform 應用程式中使用 SmtpClient 物件傳送 Mail。

功能描述：

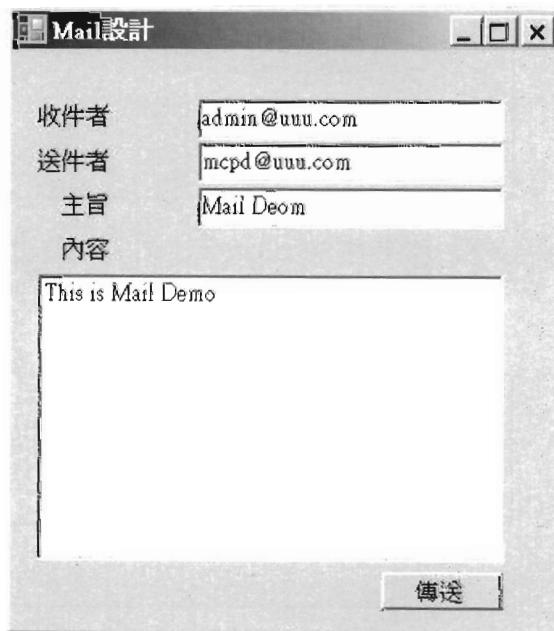
設計一個 Windows 應用程式，在畫面上設計號傳送郵件的功能，接著再使用 SmtpClient 物件來傳送 Mail，並且設定 Server 的 SMTP 讓 MAIL 可以順利傳送。

預估實作時間：15 分鐘

實作步驟：

1. 從「Start」→「Program」→「Microsoft Visual Studio 2008」→「Microsoft Visual Studio 2008」，啓動 Visual Studio 開發環境。
2. 建立一個新的 Windows Application 專案，將專案命名為 Mod15\_1。
3. 在 Form1 表單上設計畫面，如下表：

控制項	name	Text
Label1		收件者
Label2		寄件者
Label3		主旨
Label4		內容
TextBox1	To TextBox	admin@uuu.com
TextBox2	From TextBox	mcpd@uuu.com
TextBox3	Subject TextBox	Mail Demo
TextBox4	Body TextBox	This is Mail Demo
Button1	SendButton	傳送



4. 點選 Form1 表單 → 按右鍵選「View Code」→ 在 Form1.vb (VB)  
類別的最上方(C#在 Form1.cs)加上必要的 System.Net.Mail 命  
名空間。

```
Visual Basic
Imports System.Net.Mail
```

```
C#
using System.Net.Mail;
```

5. 回到 Form1 設計畫面，雙擊 SendButton，在 SendButton 的 click 事件中撰寫程式使用 MailMessage 物件組織要傳送的 mail 訊息，程式碼如下。

Visual Basic

```
Private Sub SendButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SendButton.Click
    Dim m As New MailMessage(FromTextBox.Text,
                           ToTextBox.Text, SubjectTextBox.Text, BodyTextBox.Text)
End Sub
```

C#

```
private void SendButton_Click(object sender, EventArgs e)
{
    MailMessage m = new MailMessage(FromTextBox.Text,
                                   ToTextBox.Text,
                                   SubjectTextBox.Text,
                                   BodyTextBox.Text);
}
```

6. 接著在 SendButton\_Click 中繼續建立 SmtpClient 物件設定 SMTP Server 並且叫用 Send 方法來傳送 Mail，程式碼如下：

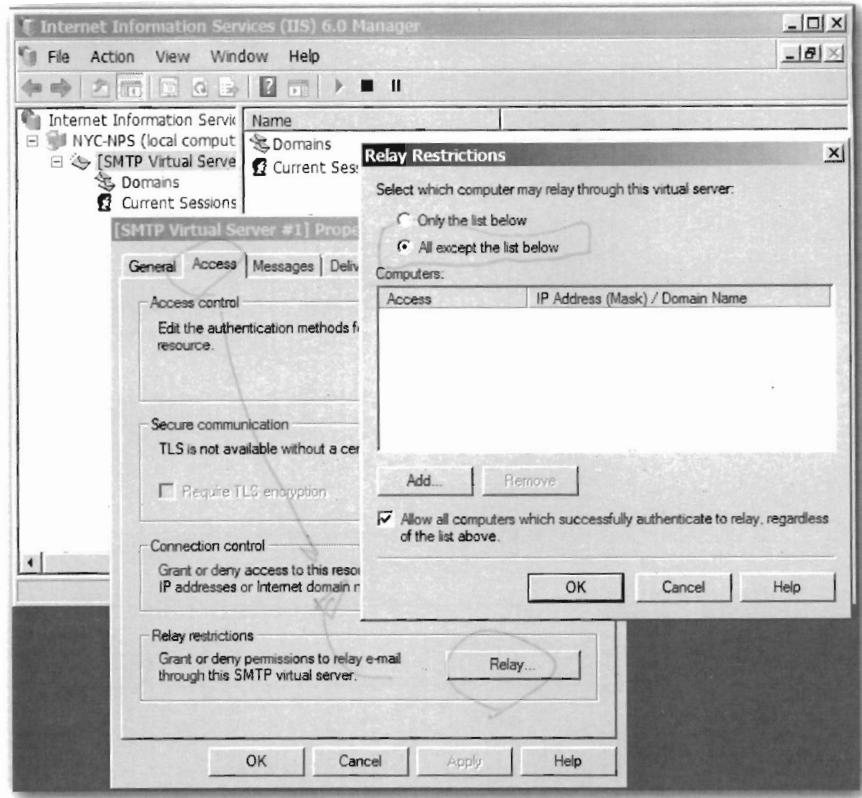
Visual Basic

```
Dim client As New SmtpClient("localhost")
client.Send(m)
MessageBox.Show("Mail傳送成功!")
```

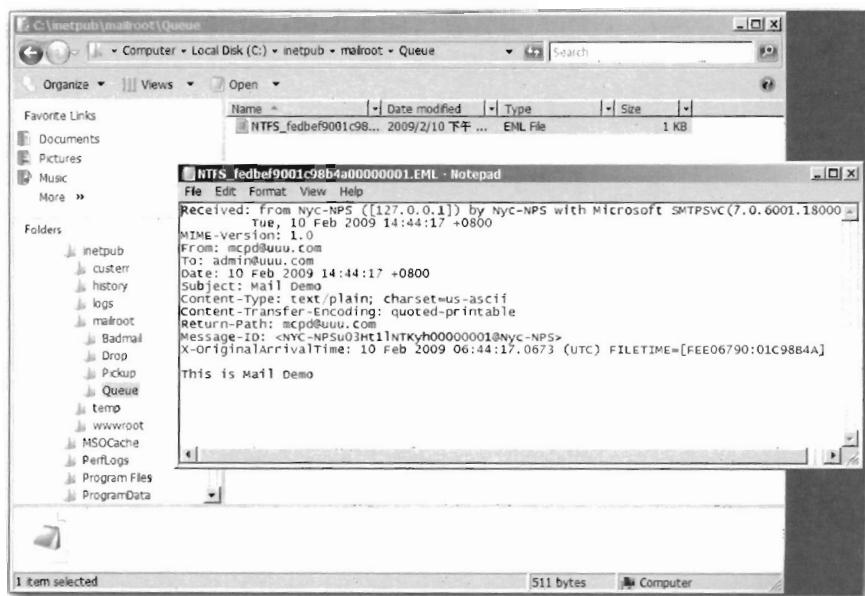
C#

```
SmtpClient client = new SmtpClient("localhost");
client.Send(m);
MessageBox.Show("Mail傳送成功!");
```

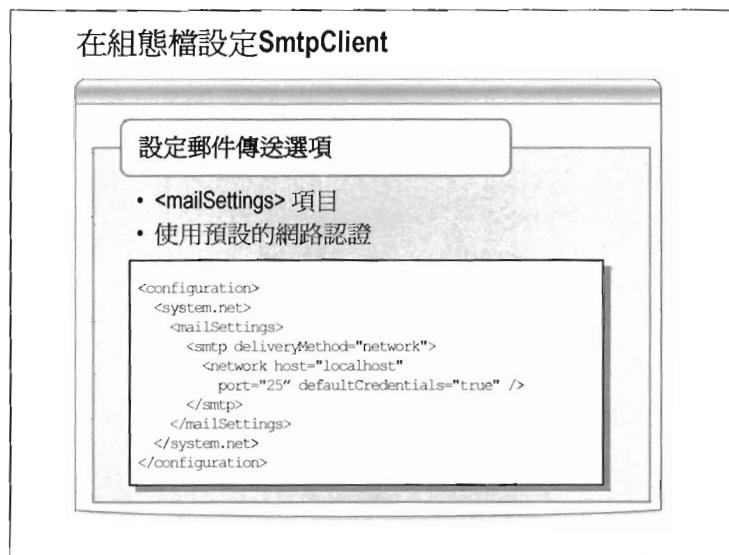
7. 撰寫好程式之後，接著設定 SMTP 啟服器，先在「控制台」的「新增移除程式」→「Windows 元件中」將 SMTP Server 安裝起來。
8. 打開 SMTP 啟服器，設定 SMTP 中的「Relay Restrictions(轉接限制)」功能，點選第二個選項「All except the list below(除了下列之外所有的)」，請參考下圖所示：



9. 按下 F5 執行應用程式測試，按下傳送按鈕。
10. 打開檔案總管檢視 C:\Inetpub\mailroot\Queue，可看到一封透過程式傳送的信件，打開後畫面如下(可以使用 Outlook Express、Outlook 或是 Notepad 等軟體查看內容)：



11. 執行完畢後，關閉應用程式。



## 在組態檔設定 SmtpClient

除了直接在應用程式中撰寫程式去建立 SmtpClient 物件並設定相關的屬性資料，如 SMTP 伺服器或身分驗證等資料，為了加強開發程式的延展性，.NET Framework 也提供透過組態檔的設定方式來指定 SMTP 伺服器等相關參數。

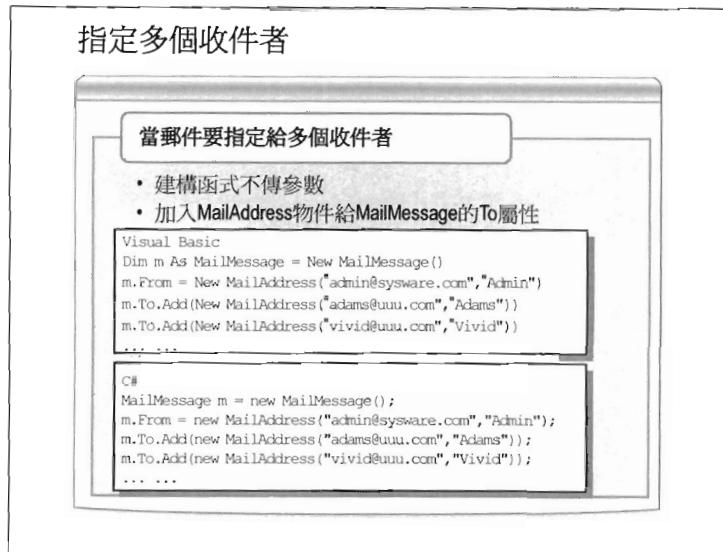
在此使用 ASP.NET 網站應用程式來示範，因為 ASP.NET 定義郵件功能的方式是寫在 Web.config 上使用<mailSettings>項目來設定，以便於網站開發者容易修改。

以下這段組態檔設定會指定適當的 SMTP 參數，使用預設的網路認證傳送電子郵件。Web.config 的設定如下：

```
<configuration>
<system.net>
<mailSettings>
<smtp deliveryMethod="network">
<network
host="localhost"
port="25"
defaultCredentials="true" />
</smtp>
</mailSettings>
</system.net>
</configuration>
```

包含在< system.net >的< mailSettings >有以下 Attribute：

- `Smtp`：設定 SMTP 選項。
- `host`：SMTP 伺服器名稱。
- `port`：SMTP 伺服器所使用的通訊埠。
- `defaultCredentials`：是否使用預設的驗證身份。



## 指定多個收件者

某些情況下使用者會想將電子郵件一次傳送給很多人，此時可以透過建立多個 MailAddress 物件，在每一個 MailAddress 中指定收件人地址以及收件人的名字，接著將 MailAddress 物件指派給 MailMessage 物件的 To 屬性即可。

以下程式碼為將一封電子郵件傳送給多個收件者：

**Visual Basic**

```
Dim m As MailMessage = New MailMessage()
m.From = New MailAddress("admin@sysware.com", "Admin")
m.To.Add(New MailAddress("adams@uuu.com", "Adams"))
m.To.Add(New MailAddress("vivid@uuu.com", "Vivid"))
...
m.Subject = "MCPD認證相關資料"
m.Body = "請密切注意微軟官方網站的公告!"
```

**C#**

```
MailMessage m = new MailMessage();
m.From = New MailAddress("admin@sysware.com", "Admin");
m.To.Add(New MailAddress("adams@uuu.com", "Adams"));
m.To.Add(New MailAddress("vivid@uuu.com", "Vivid"));
...
m.Subject = "MCPD認證相關資料";
m.Body = "請密切注意微軟官方網站的公告!";
```

### 在Mail中附加檔案

#### 使用Attachment物件

- 將檔案加入AttachmentCollection
- MailMessage.Attachments 使用Add方法加入

```
Visual Basic
Dim m As New MailMessage()
Dim AF As New Attachment("C:\MCPD.txt")
m.Attachments.Add(AF)
```

```
C#
MailMessage m = new MailMessage();
Attachment AF = new Attachment("C:\MCPD.txt");
m.Attachments.Add(AF);
```

### 在電子郵件中附加檔案

若要在電子郵件訊息中加入附件，可使用 Attachment 類別建立附件，然後使用 MailMessage 的 Attachments 屬性將 Attachment 類別所建立的物件執行個體加入訊息中，附件內容可以是 String、Stream 或檔案名稱。

然而某些情況下根據收件者所用的電子郵件讀取器 (Reader) 和附件的檔案類型而定，某些收件者可能無法讀取附件。對於無法以原始形式顯示附件的用戶端，可以搭配使用 MailMessage 的 AlternateViews 屬性為其指定替代檢視。

下列程式碼為一簡單附件設計功能，其功能為將 C:\下的 MCPD.txt 文字檔加入郵件中當做電子郵件的附件檔：

```
Visual Basic
Dim m As New MailMessage()
Dim AF As New Attachment("C:\MCPD.txt")
m.Attachments.Add(AF)
```

```
C#
MailMessage m = new MailMessage();
Attachment AF = new Attachment("C:\\MCPD.txt");
m.Attachments.Add(AF);
```

### 練習15.2：在傳送的Mail中附加檔案



### 練習 15.2：在傳送的 Mail 中附加檔案

目的：

練習如何在 Winform 應用程式中傳送 Mail 時附加檔案。

功能描述：

設計一個 Windows 應用程式，使用 SmtpClient 物件來傳送 Mail，並且在傳送郵件的同時將會附加所指的檔案，且將 mail 傳送給多個收件者。

預估實作時間：10 分鐘

實作步驟：

1. 打開檔案總管，將路徑指到「\U2956\Practices\VB 或 CS\Mod13\_2\Starter\ Mod13\_1」，雙擊 Mod13\_1.sln 開啓 Mod13\_1 專案。
2. 點選 Form1 表單 → 按右鍵選「View Code」。
3. 使用 MailMessage 物件建立兩個郵件收件者以及一個寄件者。

4. 找到程式中的註解「TODO:設定多個收件者」，在這段註解下撰寫程式，程式碼如下。

Visual Basic

```
'TODO:設定多個收件者
Dim m As New MailMessage()
m.From = New MailAddress("admin@uuu.com", "Admin")
m.To.Add(New MailAddress("adams@uuu.com", "Adams"))
m.To.Add(New MailAddress("vivid@uuu.com", "Vivid"))
m.Subject = SubjectTextBox.Text
m.Body = BodyTextBox.Text
```

C#

```
//TODO:設定多個收件者
MailMessage m = new MailMessage();
m.From = new MailAddress("admin@uuu.com", "Admin");
m.To.Add(new MailAddress("adams@uuu.com", "Adams"));
m.To.Add(new MailAddress("vivid@uuu.com", "Vivid"));
m.Subject = SubjectTextBox.Text;
m.Body = BodyTextBox.Text;
```

5. 設計 Mail 附加檔案，。建立 Attachment 物件，把此物件使用 MailMessage 的 Attachments 屬性加入在郵件中附加檔案。

6. 找到程式中的註解「TODO:設定附加檔案」，在這段註解下撰寫程式，程式碼如下。

Visual Basic

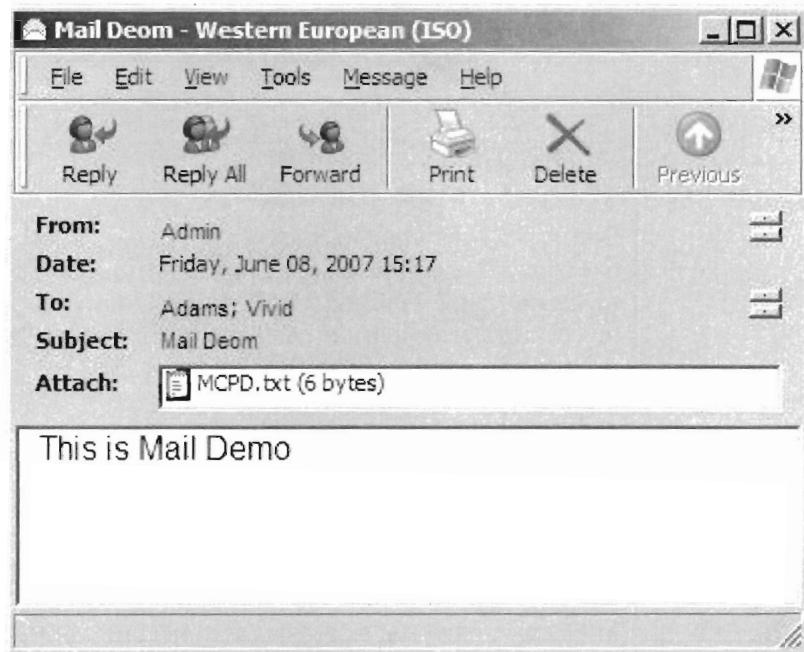
```
'TODO:設定附加檔案
Dim AF As New Attachment("C:\MCPD.txt")
m.Attachments.Add(AF)
```

C#

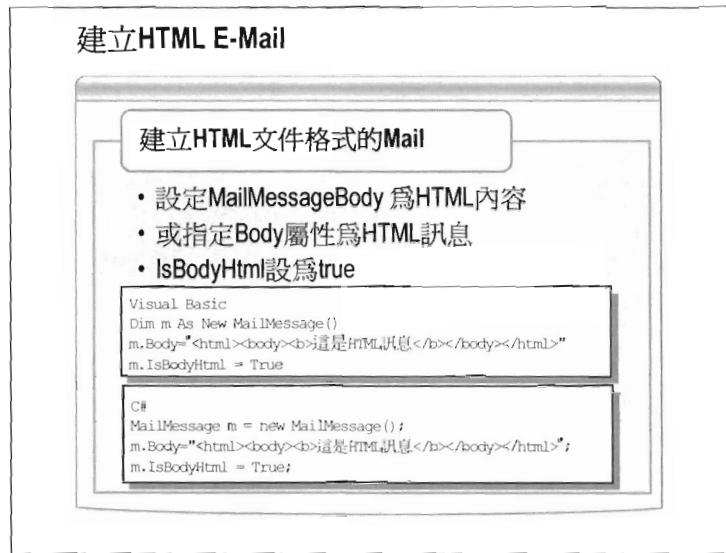
```
//TODO:設定附加檔案
Attachment AF = new Attachment(@"C:\MCPD.txt");
m.Attachments.Add(AF);
```

7. 按下 F5 執行應用程式測試，按下傳送按鈕。

8. 打開檔案總管檢視 C:\inetpub\mailroot\Queue，可看到一封透過程式傳送的信件，打開後畫面如下：



9. 執行完畢後，關閉應用程式。



## 建立 HTML E-Mail

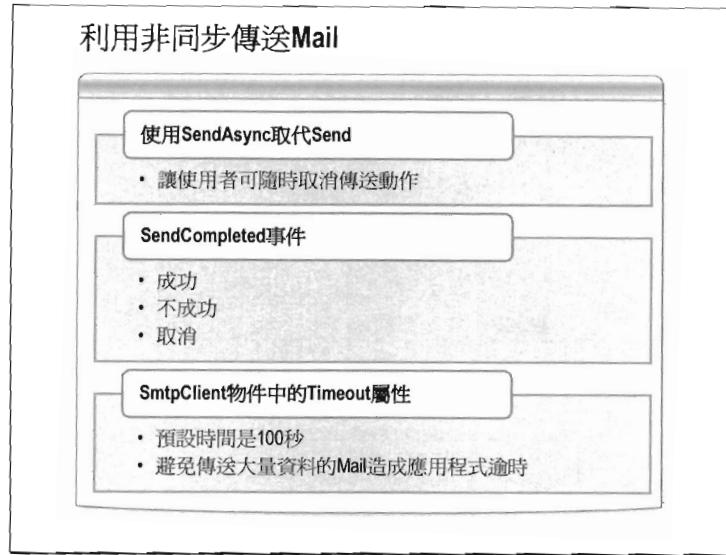
如果要傳送的電子郵件想要以 HTML 的格式傳送出去，那麼最重要的兩個屬性設定分別為：

- 設定 MailMessage 物件的 IsBodyHtml 為 true，IsBodyHtml 屬性是用來設定郵件訊息主體是否採用 Html 格式。
- 設定 MailMessageBody 為 HTML 內容
- 或是直接指定 MailMessage 物件的 Body 屬性為 HTML 格式的訊息。

下列程式碼將 MailMessage 物件的 IsBodyHtml 屬性設定為 true 並且設定 Body 屬性為 HTML 格式的訊息：

```
Visual Basic
Dim m As New MailMessage()
m.Body = "<html><body><b>這是HTML格式的訊息</b></body></html>"
m.IsBodyHtml = True
```

```
C#
MailMessage m = new MailMessage();
m.Body = "<html><body><b>這是HTML格式的訊息</b></body></html>";
m.IsBodyHtml = True;
```



## 利用非同步傳送 Mail

基本上傳送一封並不會花費多大的時間，所以大部分都在短間之內就已經傳送完成，但是有時候因為信件的內容過多或是附檔的資料過大時，常常會造成傳送時間過久，甚至造成傳送逾時的情況發生，在.NET Framework 的 SmtpClient 物件中的 Timeout 屬性預設時間是 100 秒，當 SMTP 伺服器超過 100 秒沒有回應就會造成 Timeout。

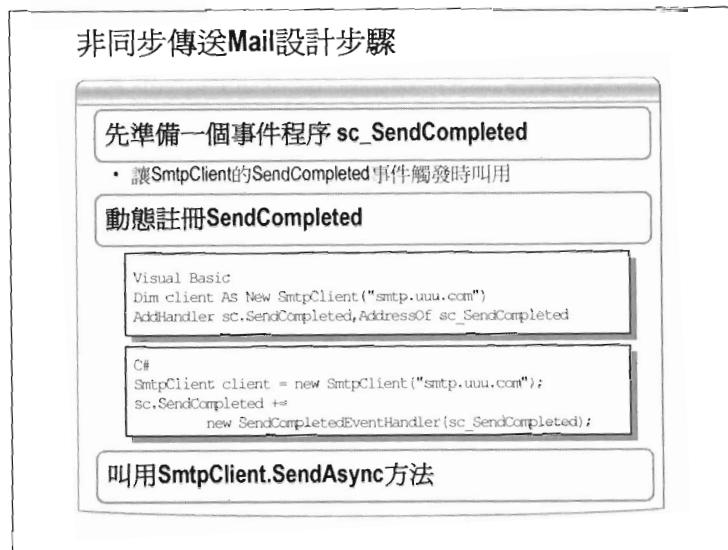
因此如果將 Timeout 屬性設定為正整數，而當 SmtpClient 物件叫用 Send 方法執行後，可能因為資料量過大或是因為些不明的因素以導致無法在指定的時間內完成，則 SmtpClient 類別會丟回 SmtpException 例外狀況。

所以比較好的設計方式是讓使用者可以在傳送 Mail 過程中隨時可以取消傳送 Mail 的動作，如果要達到這樣的功能可以使用 SmtpClient 物件的非同步傳送設計。

每次電子郵件訊息以非同步方式傳送時，會在傳送作業完成時引發 SendCompleted 事件，SendCompleted 事件的委派以及事件程序中的事件處理常式分別為：

SendCompletedEventHandler 是 SendCompleted 的委派。

- AsyncCompletedEventArgs 類別提供事件資料給事件處理常式。



## 非同步傳送 Mail 設計步驟

當了解基本的非同步傳送電子郵件運作方式之後，接著介紹非同步傳送電子郵件的幾個步驟：

1. 準備一個事件程序讓 `SmtpClient` 物件的 `SendCompleted` 事件觸發時叫用。

Visual Basic

```
Sub sc_SendCompleted(ByVal sender As Object, ByVal e As AsyncCompletedEventArgs)
If e.Cancelled Then
    Response.Write ("郵件取消")
Else
    If Not (e.Error Is Nothing) Then
        Response.Write ("Error: " + e.Error.ToString())
    Else
        Response.Write ("郵件已傳遞")
    End If
End If
End Sub
```

C#

```
void sc_SendCompleted(object sender, AsyncCompletedEventArgs e)
{
```

```

if (e.Cancelled)
    Response.Write ("郵件取消");
else if (e.Error != null)
    Response.Write ("Error: " + e.Error.ToString());
else
    Response.Write ("郵件已傳送");
}

```

2. 透過委派(Delegate)動態註冊 SmtpClient.SendCompleted 去執行定義好的事件程序。

Visual Basic
Dim client As New SmtpClient("smtp.uuu.com") AddHandler sc.SendCompleted,AddressOf sc_SendCompleted
C#
SmtpClient client = new SmtpClient("smtp.uuu.com"); sc.SendCompleted += new SendCompletedEventHandler(sc_SendCompleted);

3. 叫用 SmtpClient.SendAsync 方法
4. 也可以使用 SmtpClient.SendAsyncCancel 方法讓使用者取消傳送 Mail

Visual Basic
'非同步傳送郵件 sc.SendAsync(m, Nothing)
'取消傳送 sc.SendAsyncCancel()

C#
//非同步傳送郵件 sc.SendAsync(m, null);
//取消傳送 sc.SendAsyncCancel();



### 練習 15.3：使用非同步方式傳送 Mail

目的：

練習如何在 Winform 應用程式中使用非同步方式傳送 Mail，讓使用者可隨時取消傳送動作。

功能描述：

SMTP 伺服器設定為匿名存取，在 Winform 應用程式中設計非同步方式傳送 Mail，當使用者傳送郵件時，可以隨時按下取消的按鈕來中斷傳送中的郵件。

預估實作時間：20 分鐘

實作步驟：

1. 打開檔案總管，將路徑指到「\U2956\Practices\VB 或 CS\Mod15\_3\Starter\Mod15\_3」，雙擊 Mod15\_1.sln 開啓 Mod15\_1 專案。
2. 點選 Form1 表單 → 按右鍵選「View Code」。
3. 如果是 VB 專案，則先匯入 System.ComponentModel 命名空間，C# 專案則不用。

Visual Basic  
Imports System.ComponentModel

4. 要設計非同步傳送的第一步驟是設計一個給 SmtpClient 物件的 SendCompleted 事件要執行的事件程序。
5. 找到程式碼中的註解「TODO:撰寫 SendCompleted 事件程序」，在其下一行撰寫程式，並使用 AsyncCompletedEventArgs 去判斷執行狀況，程式碼如下：

```
Visual Basic
'TODO:撰寫SendCompleted事件程序
Sub sc_SendCompleted(ByVal sender As Object, ByVal e As AsyncCompletedEventArgs)
    If e.Cancelled Then
        MessageBox.Show("郵件取消。", "取消",
                       MessageBoxButtons.OK, MessageBoxIcon.Error)
    Else
        If Not (e.Error Is Nothing) Then
            MessageBox.Show("錯誤: " + e.Error.ToString(), "錯誤",
                           MessageBoxButtons.OK, MessageBoxIcon.Error)
        Else
            MessageBox.Show("郵件傳送成功", "成功",
                           MessageBoxButtons.OK, MessageBoxIcon.Information)
        End If
    End If
    SendButton.Text = "傳送"
End Sub
```

```
C#
//TODO:撰寫SendCompleted事件程序
void sc_SendCompleted(object sender, AsyncCompletedEventArgs e)
{
    if (e.Cancelled)
        MessageBox.Show("郵件取消。", "取消",
                       MessageBoxButtons.OK, MessageBoxIcon.Error);
    else if (e.Error != null)
        MessageBox.Show("錯誤: " + e.Error.ToString(),
                       "錯誤", MessageBoxButtons.OK, MessageBoxIcon.Error);
    else
        MessageBox.Show("郵件傳送成功", "成功",
                       MessageBoxButtons.OK, MessageBoxIcon.Information);
    SendButton.Text = "傳送";
}
```

6. 動態註冊 SmtpClient 物件的 SendCompleted 事件，以執行 sc\_SendCompleted 事件程序，並且叫用 SmtpClient 物件的 SendAsync 方法來執行非同步傳送郵件。
7. 找到程式碼中的註解「TODO:動態註冊 SendCompleted 事件」，在其下一行撰寫程式，程式碼如下：

Visual Basic

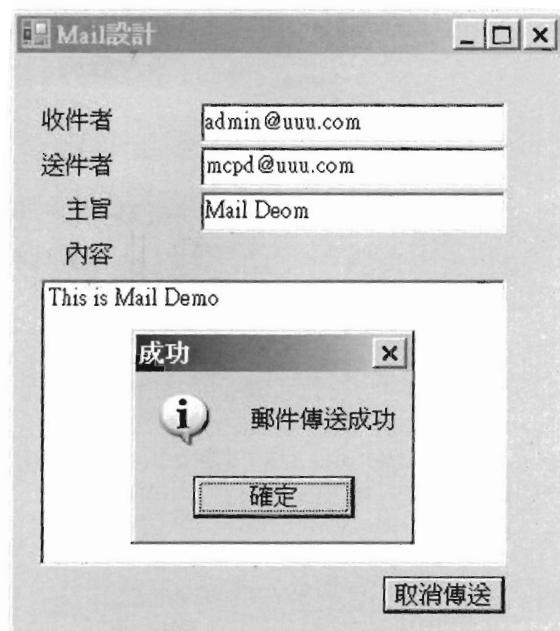
```
'TODO:動態註冊SendCompleted事件
If SendButton.Text = "傳遞" Then
    AddHandler client.SendCompleted, AddressOf sc_SendCompleted
    client.SendAsync(m, Nothing)
    SendButton.Text = "取消傳送"
Else
    client.SendAsyncCancel()
End If
```

C#

```
//TODO:動態註冊SendCompleted事件
if (SendButton.Text == "傳遞")
{
    client.SendCompleted += new SendCompletedEventHandler(sc_SendCompleted);
    client.SendAsync(m, null);
    SendButton.Text = "取消傳送";
}
else
{
    client.SendAsyncCancel();
}
```

8. 按下 F5 開啓應用程式，點選「傳遞」。

- 如果 SendAsync 被取消，則顯示取消訊息。
- 如果產生錯誤，則顯示錯誤訊息。
- 如果傳送成功，則顯示成功訊息。



9. 關閉執行應用程式。

