12

CHAPTER

ASP.NET 程式設計實務

ASP.NET 進階議題

12-1 Code-Behind

舊版 ASP 網頁中混和了 HTML 標籤以及程式敘述，這樣會讓程式不容易閱讀及除錯。 ASP.NET 提供了可以將 HTML 標籤以及程式敘述分離的技術，讓顯示使用者介面的 HTML 標籤獨立成爲一個檔案，而程式邏輯則分離成另外一個檔案，這樣的技術稱爲「Code-Behind」。使用 Code-Behind 技術的好處是：

- 使用者介面以及程式邏輯可以同時由不同的開發人員同時開發，只需要知道網頁上控制項的種類以及 ID 即可。
- 負責使用者介面的網頁開發人員只需要負責控制項的配置及美化，不需要理會實作網頁程式邏輯的語言及內容
- 程式開發人員只需要負責撰寫程式邏輯，不用擔心使用者介面的配置及美化等問題。
- 網頁開發人員可以選擇其它的 HTML 編輯工具來編輯使用者介面。
- 網頁開發人員可以將程式編譯成 DLL 檔，這樣一來在佈署網頁時就可以隱藏所撰寫的原始程式碼。

Code-Behind 網頁的組成

Code-Behind 網頁是由兩個分開的檔案所組成，一個爲負責使用者介面定義及配置的「.aspx」檔，而另一個爲負責網頁程式邏輯的**.vb**(VB.NET) 或**.cs**(C#.NET) 等程式撰寫的類別檔，將視使用語言的不同而有所不同。要撰寫 Code-Behind 的程式邏輯必需先宣告一個繼承自 System.Web.UI.Page 類別的衍生類別，並在衍生類別中將網頁的程式碼實作。下圖爲 Code-Behind 網頁示意圖：

使用範例

下列範例實作了網頁 EX01.aspx 的 Code-Behind 程式檔, 並在類別中實作了 Page_Load 事件程序：

```
Imports System
Imports System.Web

Public Class EX01
  Inherits System.Web.UI.Page

  Public Sub Page_Load(sender As Object, e As EventArgs)
    Response.Write("這是 Code-Behind 的程式碼.")
  End Sub

End Class
```

程式 EX01.aspx.vb

由於上述的 Code-Behind 檔案是使用者介面檔 EX01.aspx 的程式邏輯檔案。爲了容易管理, 一般會將類別的名稱和 .aspx 網頁指定一樣的名稱, 並且在存檔時指定檔案名稱爲 .aspx 使用者介面網頁網頁檔名再加上程式語言的副檔名。故上述程式存成 EX01.aspx.vb, 即容易分辨該 Code-Behind 檔案爲 EX01.aspx 網頁的 Code-Behind 檔案。而使用 Code-Behind 的使用者介面網頁, 要在頁首加上 @Page 指令：

```
<%@Page  Inherits="類別名稱"  Src="程式邏輯檔案名稱"%>
```

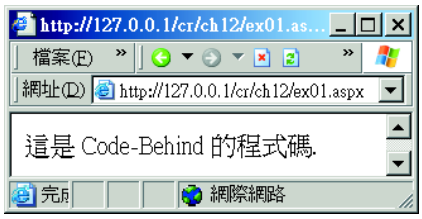
由於 Code-Behind 的所有程式邏輯必需宣告在一個類別中，所以使用 Code-Behind 的 .aspx 網頁必需指明程式的來源要繼承的是那個類別。另外如果要保護原始碼的機密，則可以將程式邏輯檔編輯成 DLL 檔並佈署到 bin 目錄後，移除 Src 屬性即可。

使用範例

下列範例爲 EX01.aspx.vb 的使用者介面檔 EX01.aspx：

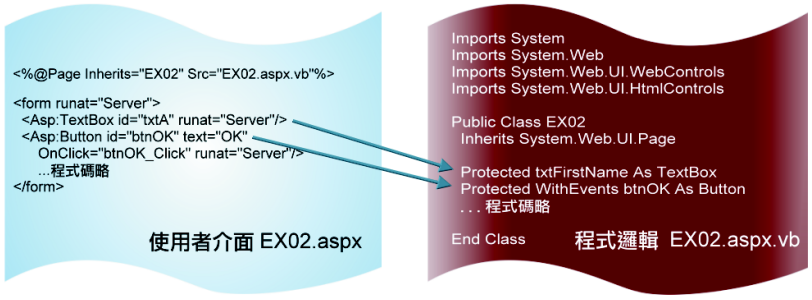
```
<%@Page  Inheruts="EX01"  Src="EX01.aspx.vb"%>
程式 EX01.aspx
```

由於上述範例並沒有放置任何視覺元素，所以在載入時只會顯示 Code-Behind 檔案內 Page_Load 程序的執行結果：



存取控制項

要在 Code-Behind 程式中存取 .aspx 網頁上有指明「id="ID" runat="Server"」屬性的控制項，則在 Code-Behind 檔案中必需宣告一個 Protected 範圍、相同類別、名稱和控制項 ID 屬性一樣的變數。若要處理控制項的事件，則要必需再加上 WithEvents 的宣告後，再撰寫事件處理程序。



使用範例

下列範例的 Code-Behind 檔案除了可以存取 .aspx 網頁上的控制項外, 還可以接收按鈕 btnOK 的 OnClick 事件：

```
Imports System
Imports System.Web
'要直接使用 Web 控制項所匯入的命名空間
Imports System.Web.UI.WebControls
'要直接使用 HTML 控制項所匯入的命名空間
Imports System.Web.UI.HtmlControls


Public Class EX02
  Inherits System.Web.UI.Page   ' 繼承自於System.Web.UI.Page

  '要接收 btnOK 的事件, 所以要加上 WithEvents
  '變數的名稱要和aspx網頁上的控制項ID一樣'
  Protected WithEvents btnOk As Button
  Protected txtFirstName As TextBox
  Protected txtLastName As TextBox
  Protected lblMsg As Label

Public Sub btnOK_Click(sender As Object, _
                       e As EventArgs) Handles btnOk.Click
  lblMsg.Text = txtLastName.Text & _
                txtFirstName.Text & ", 你好!"
End Sub
End Class
```

程式 EX02.aspx.vb

由於 EX02.aspx.vb 這個 Code-Behind 程式內的變數要被 EX02.aspx 所存取, 所以不能宣告爲 Private 或使用 Dim 宣告。由於要處理 btnOK 控制項的 OnClick 事件, 所以在宣告 btnOK 時要加上 WithEvent 外, 在撰寫事件處理程序時也要在程序宣告的面加上「Handles btnOK.OnClick」, 表示該程序要處理 btnOK 的 OnClick 事件。下列爲 EX02.aspx 網頁檔的內容：

```
<%@Page Inherits="EX02" Src="EX02.aspx.vb"%>

<form runat="Server">
  姓: <Asp:TextBox id="txtLastName" runat="Server"/><br>
  名: <Asp:TextBox id="txtFirstName" runat="Server"/>
  <Asp:Button id="btnOK" text="確定" OnClick="btnOK_Click"
      runat="Server"/><p>
  <Asp:Label id="lblMSG" runat="Server"/>
  名: <Asp:TextBox id="txtA" runat="Server"/>
</form>
```

程式 EX02.aspx

上述網頁 EX02.aspx 除了控制項的 id 屬性要和 Code-Behind 上的物件變數一致外，要執行的事件程序也必需明確宣告才會正確動作。網頁 EX02.aspx 執行的結果如右圖所示：

Code-Behind 除了可以讓程式容易閱讀及管理外，也可以讓開發小組同時開發網頁使用者介面以及程式邏輯。另外 Visual Studio.NET 開發平台中的 ASP.NET 專案，全部都是使用 Code-Behind 的方式來開發。

使用者自訂控制項

撰寫 ASP.NET 網頁解決方案時經常會用到一些重複的功能，如使用者身份查驗等。在每個不同的網頁中撰寫這些重複的程式碼不但浪費時間，而且所撰寫出來的程式碼不易維護，也容易造成錯誤。此時最好將這些經常重複使用的功能製作成可重複再利用的元件，倘若該功能不需要使用者介面，則可以製作成沒有使用者介面的類別。若該功能需要使用固定的使用者介面，則可以設計成使用者自訂控制項，來滿足不同的解決方案並提升程式開發的效率。

使用者自訂控制項並不是預先編譯好的元件，它是在被別的網頁使用時才會立即被編譯。使用者自訂控制項也可以稱爲「Pagelet」，其副檔名爲「.ascx」，是一種只能在其它 ASP.NET 網頁中存在及執行的 ASP.NET 網頁。使用者自訂控制項的優點有：

- 使用者自訂控制項由於存放在 .ascx 網頁中，所以可以由其它不同的語言所寫成。
- 使用者自訂控制項包含自我執行時所需要的程式碼，並不會和 ASP.NET 網頁上的變數或程序產生混淆。
- 使用者自訂控制項可以重複在不同的 ASP.NET 網頁中使用，提升程式開發的效率。

製作使用者自訂控制項

使用者控制項的副檔名爲「.ascx」，這個副檔名可以用來確定該網頁不能單獨執行，其使用方式和一般控制項相同，都是在一般 ASP.NET 中使用特定的標籤來配置使用者控制項。使用者控制項是由 HTML 標籤、HTML 控制項以及 Web 控制項所組成，但是不可以包含<head>、<body>以及<form>標籤，這是因爲使用者控制項在 ASP.NET 網頁中被使用時，這些標籤已經在原來的 ASP.NET 網頁中存在。使用者自訂控制項可以提供屬性及方法、處理自己的事件(如 Page_Load)，並且在宣告的時候要加上「<%@Control Language=" 所使用的程式語言 "%>」，而且一個 .ascx 檔案內只能夠有一個「@Control」指令。另外使用者控制項上面的任何物件，並不能在 ASP.NET 網頁中直接存取，必需在使用者控制項的 .ascx 檔案中以宣告屬性程序的方式來存取才可以。

使用範例

下列範例宣告了一個可以輸入使用者姓名的 UserName 使用者自訂控制項，並提供了 FirstName 、LastName 、FullName 屬性以及 Reset 方法：

```
<%@Control Language="VB" %>

  姓: <Asp:TextBox id="txtLastName" runat="Server"/><br>
  名: <Asp:TextBox id="txtFirstName" runat="Server"/>

<script language="vb" runat="Server">
Private Sub Page_Load(sender As Object, e As EventArgs)
  If Page.IsPostBack = False Then
    txtFirstName.Text = "請輸入名"
    txtLastName.Text = "請輸入姓"
  End If
End Sub

Public Property FirstName() As String
  Get
    Return txtFirstName.Text
  End Get
  Set(Value As String)
    txtFirstName.Text = Value
  End Set
End Property

Public Property LastName() As String
  Get
    Return txtLastName.Text
  End Get
  Set(Value As String)
    txtLastName.Text = Value
  End Set
End Property

Public ReadOnly Property FullName() As String
  Get
    Return txtFirstName.Text & " " & txtLastName.Text
  End Get
End Property

Public Sub Reset()
  txtFirstName.Text = "請輸入名"
  txtLastName.Text = "請輸入姓"
End Sub
</script>
```

程式 UserName.ascx

上述範例在控制項第一次被載入的時候, 將 txtFirstName 以及 txtLastName 的 Text 屬性分別填入「請輸入名」以及「請輸入姓」的字串。而控制項的 FirstName 屬性對應到 txtFirstName 控制項的 Text 屬性, LastName 屬性對應到 txtLastName 控制項的 Text 屬性, FullName 屬性爲唯讀屬性, 將 txtFirstName 以及 txtLastName 的 Text 屬性傳回, 最後的 Reset 方法則是將 txtFirstName 以及 txtLastName 的 Text 屬性還原爲控制項第一次被載入時的「請輸入名」以及「請輸入姓」的字串。

使用者自訂控制項的使用

要在 ASP.NET 網頁中使用使用者自訂控制項, 必須在網頁的前面使用「@Register」指令, 其語法如下所示：

```
<%@Register TagPrefix="自訂命名空間"  TagName="控制項標籤名稱"
            Src="ascx檔案位置"%>
```

其中「TagPrefix」屬性是用來指定控制項的命名空間, 而「TagName」屬性是用來指定使用者控制項的標籤名稱。例如 TagPrefix 屬性設定爲「Charles」而 TagName 屬性設定爲「UserName」, 則在 ASP.NET 網頁中要使用該使用者自訂控制項的語法如下所示：

```
<Charles:UserName id="unA" runat="Server"/>
<Charles:UserName id="unB" runat="Server"/>
```

上述標籤除了指定 ID 屬性外, 也要指定 runat 屬性爲「Server」。另外由於該控制項包含可傳回資料的 TextBox, 所以在使用的時候也必需被 form 標籤包圍。

使用範例

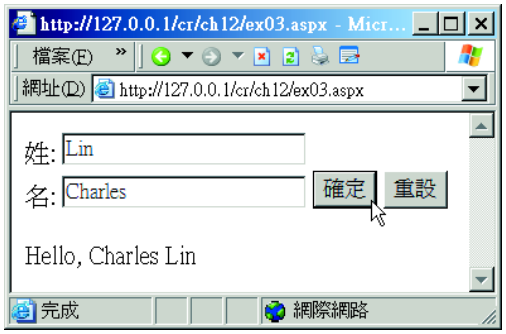
下列範例顯示如何使用 UserName.ascx 使用者自訂控制項：

```
<%@Register TagPrefix="Charles" TagName="UserName"
            Src="UserName.ascx"%>

<form runat="Server">
  <Charles:UserName id="unA" runat="Server"/>
  <Asp:Button id="btnOK" text="確定" OnClick="btnOK_Click"
       runat="Server"/>
  <Asp:Button id="btnReset" text="重設"
       OnClick="btnReset_Click" runat="Server"/><p>
  <Asp:Label id="lblMSG" runat="Server"/>
</form>

<script language="vb" runat="Server">
Private Sub btnOK_Click(sender As Object, e As EventArgs)
  lblMsg.Text = "Hello, " & unA.FullName
End Sub

Private Sub btnReset_Click(sender As Object, e As EventArgs)
  unA.Reset()
End Sub
</script>
```

程式 EX03.aspx

網頁開始執行時，控制項會執行自己的 Page_Load 事件將控制項 txtFirstName 以及 txtLastName 的 Text 屬性改成「請輸入名」以及「請輸入姓」的字串：

在 txtFirstName 以及 txtLastName 中輸入姓名後，按下確定按鈕：



按下重置按鈕呼叫 UserControl 的 Reset 方法重置 TextBox 的內容：

善用 UserControl 不但可以增加開發網頁的效率，而且可以減少 ASP.NET 網頁上的程式碼，讓程式易於閱讀及除錯。

「@Control」指令和「@Page」指令有相同的屬性，不過 Trace 和 AspCompat 屬性例外。倘若要追蹤使用者自訂控制項的資料，只需要開啓 ASP.NET 網頁上的 @Page 指令，將其 Trace 屬性設爲 True 即可。

使用者自訂控制項的 Code-Behind 技術

使用者自訂控制項也可以使用 Code-Behind 的方式來撰寫，讓使用者自訂的開發及管理享受到 Code-Behind 的好處。基本上將使用者控制項以 Code-Behind 的方式撰寫和 Code-Behind 的網頁撰寫方式差不多，只不過 Code-Behind 程式的類別繼承要由「System.Web.UI.Page」改成繼承自「System.Web.UI.UserControl」。

使用範例

下列範例將 UserName 使用者自訂控制項改成以 Code-Behind 的方式寫成，下列程式碼爲 Code-Behind 程式碼的撰寫方式：

```
Imports System
Imports System.Web
Imports System.Web.UI.WebControls
Imports System.Web.UI.HtmlControls

Public Class UserNameCB
  '繼承自於System.Web.UI.UserControl
  Inherits System.Web.UI.UserControl

  Protected txtFirstName As TextBox
  Protected txtLastName As TextBox

  Private Sub Page_Load(sender As Object, e As EventArgs)
    If Page.IsPostBack = False Then
      txtFirstName.Text = "請輸入名"
      txtLastName.Text = "請輸入姓"
    End If
  End Sub

  Public Property FirstName() As String
    Get
      Return txtFirstName.Text
    End Get
    Set(Value As String)
      txtFirstName.Text = Value
    End Set
  End Property

  Public Property LastName() As String
    Get
      Return txtLastName.Text
    End Get
    Set(Value As String)
      txtLastName.Text = Value
    End Set
  End Property

  Public ReadOnly Property FullName() As String
    Get
      Return txtFirstName.Text & " " & txtLastName.Text
    End Get
  End Property
```

```
  Public Sub Reset()
    txtFirstName.Text = "請輸入名"
    txtLastName.Text = "請輸入姓"
  End Sub
End Class
```

程式 UserNameCB.ascx.vb

上述範例除了繼承的類別有所不同外, 其餘皆和製作 Code-Behind 網頁的方式相同。下列則爲使用者自訂控制項的使用者介面部分的宣告：

```
<%@Control Language="VB" Inherits="UserNameCB"
           Src="USerNameCB.ascx.vb"%>

  姓: <Asp:TextBox id="txtLastName" runat="Server"/><br>
  名: <Asp:TextBox id="txtFirstName" runat="Server"/>
```

程式 UserNameCB.ascx

上述 Code-Behind 使用者自訂控制項的使用方式, 和 EX03.aspx 網頁的方式一樣, 只需要將第一行的 @Resgister 指令所參用的檔案名稱改成 UserNameCB.ascx 即可, 如下列程式碼片段所示：

```
<%@Register TagPrefix="Charles" TagName="UserName"
            Src="UserNameCB.ascx"%>
```

12-2 自訂 Web 控制項

使用者自訂控制項儲存於副檔名 .ascx 的檔案中, 每一個使用使用者自訂控制項的 ASP.NET 網頁都必需在同一個目錄內複製一份 .ascx 檔案, 並且在執行時才即時編譯 .ascx 檔。若要將使用者自訂控制項製作成已經編譯過的 Assembly, 並放置在 GAC 中供所有的 ASP.NET 網頁使用, 則必須將使用者自訂控制項製作成「自訂 Web 控制項」(或稱爲自訂 Server 控制項)。

使用範例

下列範例顯示如何製作自訂 Web 控制項, 該控制項提供 Text 方法可以設定控制項所要顯示的文字, 並提供了一個 FontSize 屬性可以設定顯示文字的字型大小：

```
Imports System
Imports System.Web.UI

Namespace Charles
  Public Class Msg
    Inherits Control   ' 繼承自於 System.Web.UI.Control

    Public Text As String
    Private m_FontSize As Byte = 3

    Public Property FontSize()As Byte
      Get
        Return m_FontSize
      End Get
      Set(ByVal Value As Byte)
        '限制只接受 0-7 的值
        If Value >= 0 AndAlso Value <=7 Then
          m_FontSize = Value
        End If

      End Set
    End Property

    '覆寫控制項輸出視覺介面的Render事件
    Protected Overrides Sub Render(Output As HtmlTextWriter)
    '下列敘述輸出字串自使用者瀏覽器
      Output.Write("<font size=" & m_FontSize.ToString() & _
                   ">" & Text & "</font>")
    End Sub


  End Class
End Namespace
```

程式 Msg.vb

自訂 Web 控制項要繼承自於 System.Web.UI.Control 類別, 並且要覆寫產生使用者介面時會引發的 Render 事件所執行的事件程序, 並在 Render 事件程序中利用 HttpTextWriter 物件的 Write 方法, 將產生使用者介面的 HTML 等資訊送至瀏覽器。由於該類別參用到 System.Web.dll 以及 System.dll 兩個 Assembly, 所以編譯的語法如下所示：

```
vbc /t:Library /r:System.Web.dll,System.dll /out:Msg.dll Msg.vb
```

編譯完成後, 由於尚未將 Msg.dll 這個 Assembly 部署至 GAC 中, 所以還是一樣在 IIS 先建立一個虛擬目錄, 再複製到下一層的 bin 目錄中即可(參閱前一章)。在 ASP.NET 網頁中使用自訂 Web 控制項要先加入 @Register 指令, 其使用語法如下所示：

```
<%@Register  TagPrefix="標籤前置字"  Namespace="類別所在命名空間"
             Assembly="Assembly名稱"%>
```

使用範例

下列範例顯示如何使用 Msg 自訂 Web 控制項：

```
<%@Register TagPrefix="Charles" Namespace="Charles"
            Assembly="Msg"%>

<Charles:Msg id="msgA" runat="Server"/>

<script language="VB" runat="Server">
Private Sub Page_Load(sender As Object, e As EventArgs)
  msgA.FontSize = 5
  msgA.Text = "自訂 Web 控制項"
End Sub
</script>
```

程式 EX04.aspx

事件的觸發

要讓自訂 Web 控制項觸發自訂的事件, 必需實作「IPostBackDataHandler」介面及「IPostBackEventHandler」介面, 這兩個介面所宣告的成員及說明如下表所示：

- IPostBackDataHandler介面

成 員	說 明
LoadPostData 方法	實作此方法可以取得並處理從自訂 Web 控制項所傳回來的資料
RaisePostDataChangedEvent	實作此方法後, 若自訂 Web 控制項的狀態有任何變更, 則會通知 ASP.NET 網頁

- IPostBackEventHandler介面

成 員	說 明
RaisePostBackEvent	實作此方法後, 可以讓自訂 Web 控制項觸發事件至 ASP.NET 網頁

使用範例

下列範例製作了使用者身份驗證自訂 Web 控制項, 提供 UserID、UserPWD 屬性, 並提供一個 Result 自訂事件來顯示驗證結果, 最後將此控制項佈署至 GAC 中：

```
Imports System
Imports System.Web
Imports System.Web.UI
Imports System.Collections.Specialized
Imports System.Reflection

<Assembly: AssemblyKeyFile("Auth.snk")>
<Assembly: AssemblyVersion("1.0.0.0")>

Namespace Charles

  Public Class AuthEventArgs
    Inherits EventArgs
    Public IsValid As Boolean = False
    Public Sub New(ByVal blnValid As Boolean)
      MyBase.New()
      IsValid = blnValid
    End Sub
  End Class

  Public Class Auth
    Inherits Control
    Implements IPostBackDataHandler, IPostBackEventHandler

    Private m_UserID As String = ""
    Private m_UserPWD As String = ""
    Public Event Result(ByVal sender As Object, _
                        e As AuthEventArgs)

    Public Property UserID() As String
      Get
        Return m_UserID
      End Get
      Set(ByVal Value As String)
        Value = Value.Replace("'", "_")  '不允許輸入
        Value = Value.Replace(";", "_")  '單引號, 分
        Value = Value.Replace(" ", "_")  '號或是底線
        m_UserID = Value.ToLower()  '帳號不分大小寫
      End Set
    End Property

    Public Property UserPWD() As String
      Get
```

```
      Return m_UserPWD
    End Get
    Set(ByVal Value As String)
      Value = Value.Replace("'", "_")   '不允許輸入
      Value = Value.Replace(";", "_")   '單引號, 分
      Value = Value.Replace(" ", "_")   '號或是底線
      m_UserPWD = Value.ToLower()   '密碼不分大小寫
    End Set
  End Property

  '產生使用者介面的程序
  Protected Overrides Sub Render( _
                          ByVal Output As HtmlTextWriter)

    Output.Write("帳號: <input name=" & Me.UniqueID & _
       " id=UserID type=text value=" & Me.m_UserID & "><br>")
    Output.Write("密碼: <input name=" & Me.UniqueID & _
       " id=UserPWD type=password><br>")
    Output.Write("<input type=Reset value=重置>  ")

  '取得將控制項在 Client 端的ID, 並製作 PostBack 的 JavaScript
    Output.Write("<input type=button value=確定 " & _
       "OnClick=""JavaScript:" & _
       Page.GetPostBackEventReference(Me) & """>")
  End Sub

  '下列程序用來載入傳回的資料,  並且決定要不要執行
  'RaisePostBackEvent 程序
  Public Overridable Function LoadPostData( _
    ByVal PostDataKey As String, _
    ByVal Values As NameValueCollection) As Boolean _
    Implements IPostBackDataHandler.LoadPostData

    'PostDataKey 參數存放傳回資料的控制項在 Client 端的 ID
    'Values 參數則存放了所有控制項所存放的資料
    '假設使用者輸入帳號爲 charles 密碼爲 pass ,則
    '在兩個 Text 中 Box 中所輸入的資料會以 charles,pass 的方式傳回

    '取得控制項的傳回參數
    Dim strValues As String = Values(PostDataKey)
    '取得逗號所在位置
    Dim intIndex As Integer = strValues.IndexOf(",")
```

```
      '下列兩行敘述分別取得使用者所輸入的帳號及密碼
      m_UserID = strValues.SubString(0 , intIndex)
      m_UserPWD = strValues.SubString( _
        intIndex + 1, strValues.Length - intIndex - 1)
      Return True  '執行 RaisePostBackEvent 程序
    End Function

    '當控制項的資料有所改變的時候, 用來通知 ASP.NET
    Public Overridable Sub RaisePostDataChangedEvent() _
    Implements IPostBackDataHandler.RaisePostDataChangedEvent
      '由於實作了IpostBackDataHandler介面,
      '就算沒有實作的程式碼也必需宣告
    End Sub

    '用來引發事件的程序
    Public Overridable Sub RaisePostBackEvent( _
      ByVal EventArgument As String) _
      Implements IPostBackEventHandler.RaisePostBackEvent

      '此段區塊可以放置檢查資料源是否有該使用者資料
      If m_UserID="charles" AndAlso m_UserPWD="pass" Then
        RaiseEvent Result(Me, New AuthEventArgs(True))
      Else
        RaiseEvent Result(Me, New AuthEventArgs(False))
      End If
    End Sub
  End Class
End Namespace
```

程式 Auth.vb

由於要將 Auth.vb 所編譯完成的組件部署至 GAC 中, 所以先利用工具程式 SN.EXE 產生檔案名稱爲 Auth.snk 的一對公私鑰：

```
sn /k Auth.snk
```

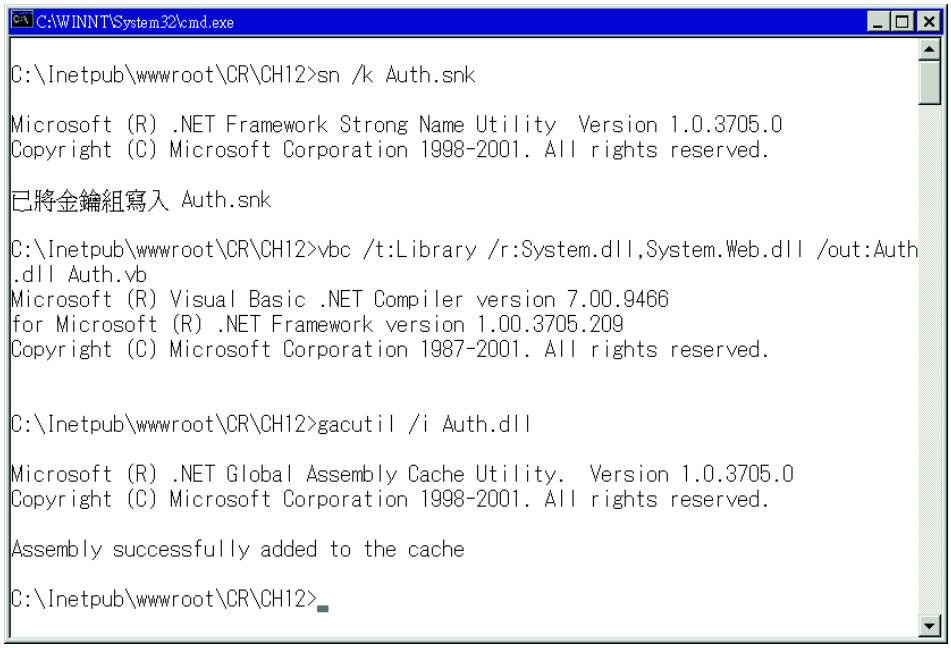
另外上述範例參用了 System.Web.dll, 所以在編譯時必需要加入：

```
vbc /t:library /r:System.dll,System.Web.dll /out:Auth.dll Auth.vb
```

編譯完成後, 就可以將 Auth.dll 部署至 GAC 中供所有的應用程式使用：

```
gacutil /i Auth.dll
```

所有編譯的步驟如下圖所示：



將 Assembly 部署至 GAC 後, 還要設定該目錄的 web.config 檔才可以使用：

```
<!--Web.Config Configuration File-->

<configuration>
   <system.web>
     <customErrors mode="Off"/>
     <compilation>
       <assemblies>
         <add assembly=
           "Auth, Version=1.0.0.0, Culture=neutral,
           PublicKeyToken=f1a28c978afa1999"/>
```

```
      </assemblies>
    </compilation>
    <globalization requestEncoding="big5"
                   responseEncoding="big5"/>
  </system.web>
</configuration>
```

注意上述的<add>標籤的 assembly 屬性設定，由於版面編排的關係 assembly 的設定值無法印在書上的同一行。 由於屬性設定值是字串，所以實際在輸入的時候必須要在同一行。

使用範例

下列範例顯示如何在 ASP.NET 網頁中使用 Auth.dll 所提供的自訂 Web 控制項：

```
<%@Register TagPrefix="Charles" Namespace="Charles"
 Assembly="Auth"%>

<form runat="Server">
  <Charles:Auth id="authA" OnResult="authA_Result"
          runat="Server"/>
  <Asp:Label id="lblMSG" Text="請輸入帳號及密碼"
       runat="Server"/>
</form>

<script language="VB" runat="Server">
Private Sub authA_Result(sender As Object, e As AuthEventArgs)
  If e.IsValid Then
    lblMsg.Text="驗證成功"
  Else
    lblMsg.Text="驗證失敗"
  End If
End Sub
</script>
```

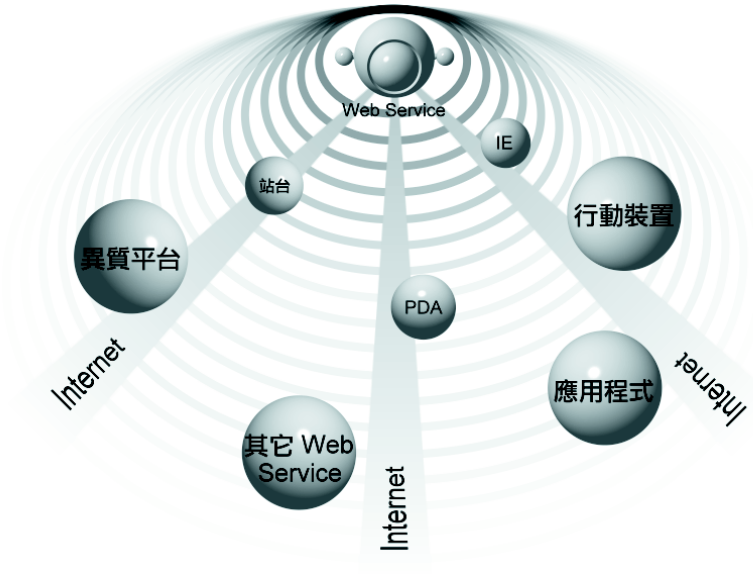
程式 EX05.aspx

爲了分辨屬性及事件處理程序，在宣告控制項的事件程序時規定一律在指定的事件名稱前加上字首「On」。故在標籤中以「OnResult="authA_Result"」的設定方式，宣告了當物件觸發 Result 事件時，要執行 authA_Result 這個事件處理程序。右圖爲輸入正確使用者帳號及密碼的執行結果：

自訂 Web 控制項和前一章所提到的中層元件的使用方式不太一樣，中層元件不需要建立虛擬目錄即可使用，而自訂 Web 控制項必需建立虛擬目錄才可以被使用，另外自訂 Web 控制項可以透過 ViewState 進行自我狀態的維護。

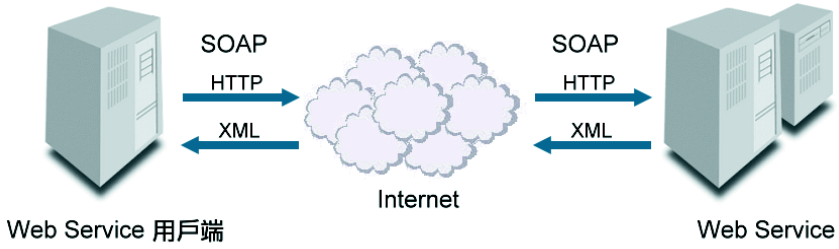
12-3 Web Services

網際網路的出現提升了企業之間的溝通及資料交換的便利。在 Web Services 出現以前，每個企業的資料規格、使用協定以及實作技術皆不相同，沒有一個公開的標準，要整合企業之間的資料不是一件容易的事。開發人員經常要針對某個特定的團體開發一個特定的解決方案，對開發人員是一個很重的負擔。Web Services 是一種全新的技術，最主要的目的是讓程式使用網際網路上的標準，透過 Internet 彼此交換資料、分享企業邏輯，整合分散在不同地方的應用程式。



Web Services 簡介

Web Service 提供了一個公開的標準將程式簡單快速地繫結在一起，不用在意該程式的開發語言以及執行平台。 Web Services 使用 Internet 上的公開標準－「SOAP」(Simple Object Access Protocol, 簡單物件存取協定)來傳送資料, SOAP 是一種架構在 HTTP 以及 XML 上的協定, 使用 HTTP 協定的好處是可以穿越防火牆，而且不需要開啓其它的通訊埠，可以存取網際網路就可以使用 Web Services 。



建立 Web Service

Web Services 可以預先編譯成 Assembly, 也可以原始的「.asmx」檔存在, 兩種方式都要預先撰寫定義 Web Servicses 的 .asmx 檔案。要產生 Web Service, 首先要加入「@WebService」指令, 其語法如下所示：

```
<@WebService Language="VB" Class="Class名稱"%>
```

Class 屬性表示要提供 Web Service 的類別。若 Class 被群組到命名空間, 或被編譯到 Assembly 時, 則其語法如下所示：

```
<@WebService Language="VB" Class="命名空間.Class名稱, Assembly名稱"%>
```

宣告完畢後, 接著匯入 System.Web.Services 命名空間, 並建立提供 Web Service 的 Class：

```
<@WebService Language="VB" Class="Class名稱"%>
Imports System.Web.Services

Public Class Class名稱
  Inherits WebService

  <WebMethod()>Public Function 可透過Web被呼叫的程序() As 型別
    …程式碼略
  End Function
End Class
```

Web Services 的類別要宣告繼承自「System.Web.Services.WebService」類別, 另外「<WebMethod()>」要和程序寫在同一行, 表示該程序可以在 Web 端被呼叫使用, 也就是用戶端可以呼叫標記爲<WebMethod()>的程序。 Web Services 就是以標記爲<WebMethod()>的方法提供用戶端叫用。

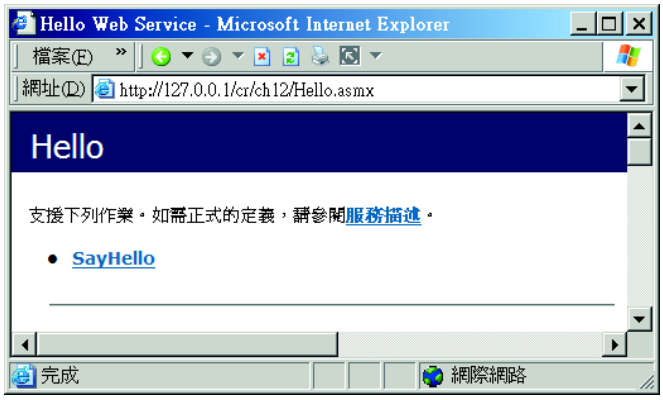
使用範例

下列範例宣告了一個名爲 Hello 的 Web Service, 並提供一個傳回字串 Hello Web Service的方法 SayHello：

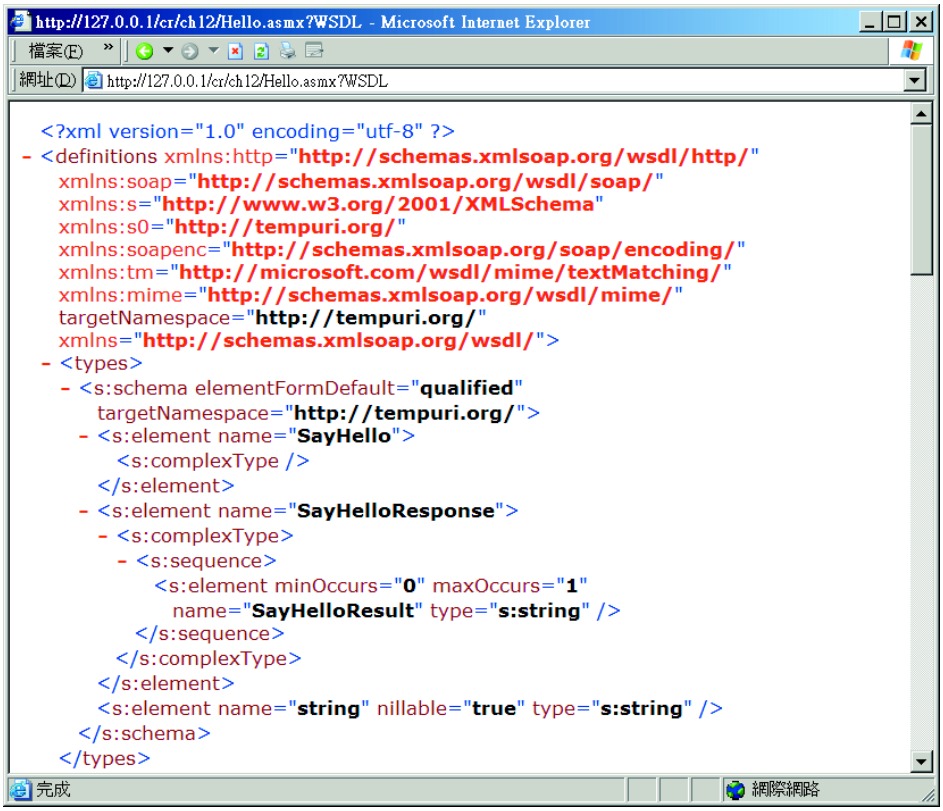
```
<%@WebService Language="VB" Class="Hello"%>

Imports System.Web.Services

Public Class Hello
  Inherits WebService

  <WebMethod()>Public Function SayHello() As String
    Return "Hello Web Service!"
  End Function

End Class
```

程式 Hello.asmx

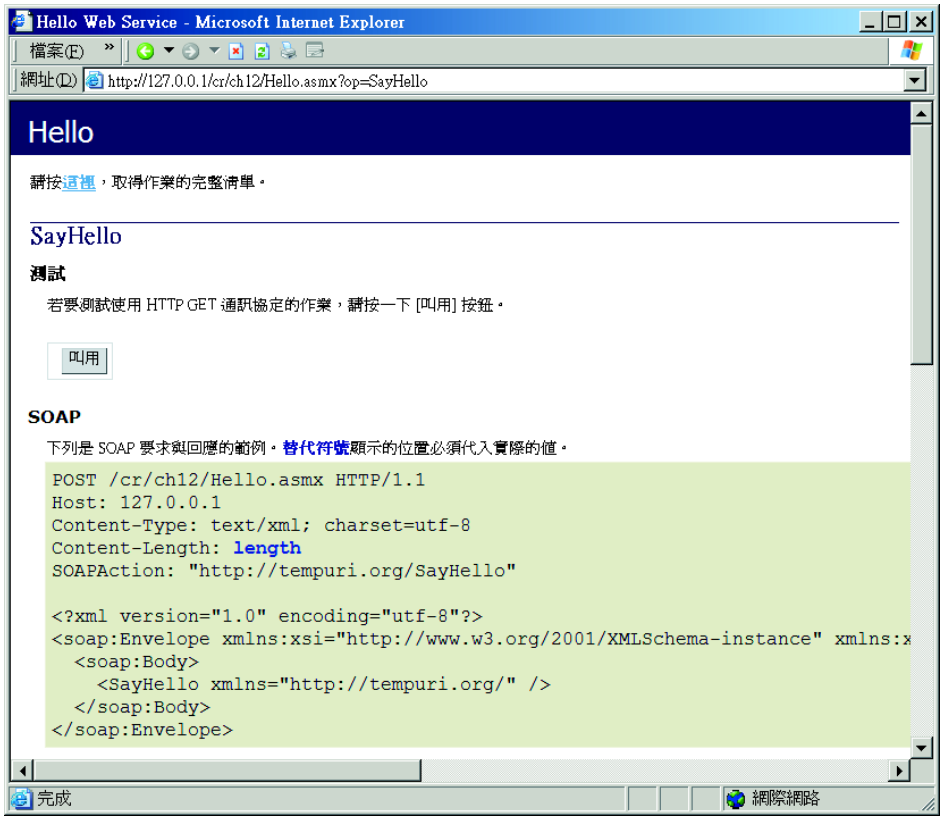
程式撰寫完畢並存成 .asmx 檔後, 就可以立即使用瀏覽器進行測試。下圖爲使用瀏覽器, 輸入 Hello.asmx 這個 Web Service 所在的位址後的執行畫面：



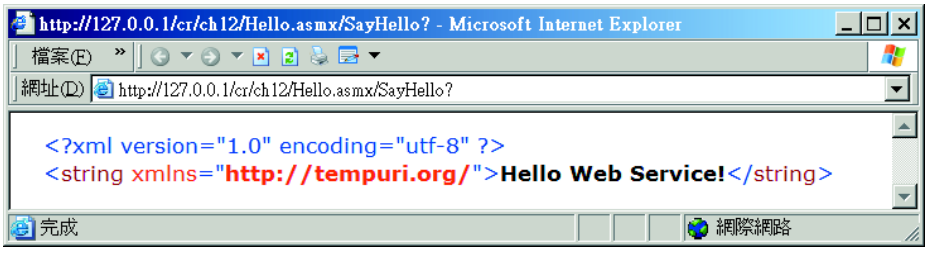
點選「服務描述」(Service Description)超連結後, 可以觀察這個 Web Service 的「Web Service描述語言」(WSDL , Web Service Description Language)內容：



WSDL 和 Assembly 的中介資料很像, 是用來描述 Web Service 提供的公用成員、方法所接收的參數、傳回值的資料型別等資訊。由於 WSDL 是以標準 XML 格式所撰寫, 所以任何可以讀取 XML 的用戶端都可以讀取 WSDL 的資料, 進而使用 Web Service 所提供的服務。接著回到上一個網頁的內容並點選「SayHello」超連結, ASP.NET 會顯示使用 SOAP、HTTP Get 以及 HTTP Post 協定來存取 Web Service 的語法：



若要使用 HTTP Get 協定來測試 Web Service, 則可以按下「叫用」(Invoke)按鈕來引發 Web Service, 即可以得到 Web Service 的執行結果：



上述畫面的執行結果是以 XML 的格式來傳回, 其中第一行的標籤宣告了使用的 XML 版本爲 1.0, 而編碼格式爲「utf-8」。而第二行爲執行的結果, 其傳回值的資料型別爲 String, 由於沒有宣告 Web Service 的命名空間, 故該 XML 標籤使用預設的命名空間「http://tempuri.org」。若想要讓 Web Services 在網際網路上正確的被使用, 則最好建立自己的命名空間。這個命名空間可以用來辨識 Web Services 的執行結果, 以避免和其它 Web Services 的執行結果混淆。要在網際網路上擁有獨一無二的名稱, 最簡單直接的方法就是申請一個網址, 並將網址作爲自訂 Web Service 的命名空間, 自訂命名空間的宣告語法如下所示：

```
<WebService(Namespace="http://網址")>Public Class 類別名稱
  Inherits WebService
  <WebMethod()>Public Function 可透過Web被呼叫的程序() As 型別
     …程式碼略
  End Function
End Class
```

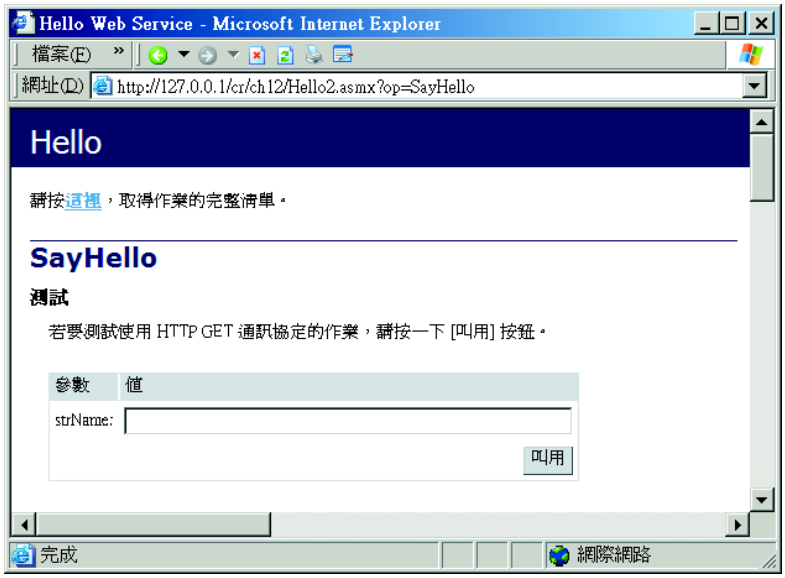
使用範例

下列範例將 Web Service 群組至自訂的命名空間內, 以避免和其它 Web Services 的執行結果造成混淆, 並將 SayHello 方法改成接收代表姓名的字串, 並傳回問候語：

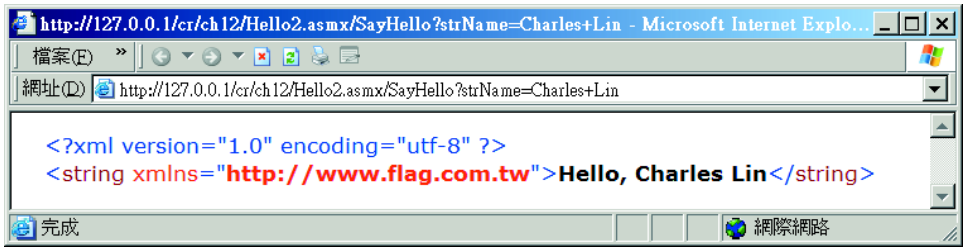
```
<%@WebService Language="VB" Class="Hello"%>
Imports System.Web.Services
<WebService(Namespace:="http://www.flag.com.tw")> _
Public Class Hello
  Inherits WebService

  <WebMethod()>Public Function SayHello( _
                      ByVal strName As String) As String
    Return "Hello, " & strName
  End Function

End Class
```

程式 Hello2.asmx

利用測試網頁SayHello方法時, 測試網頁會自動產生文字輸入盒讓你輸入測試參數：



上述範例執行結果如下圖所示：



將 Web Services 編譯成 Assembly

Web Services 可以編譯成 DLL, 將 Web Services 編譯成 DLL 有兩個優點：

- 本機端要使用編譯成 DLL 的 Web Services 時, 不需要透過 SOAP 以及 HTTP 的呼叫, 可以直接像使用其它 Assembly 一樣使用, 以提升執行效能。
- 製成 DLL 的 Web Services 可以使用許多 Assembly 的功能, 例如加入版本控管等資訊。

要將 Web Services 以 DLL 的方式存在, 其實就是將 Web Services 以 Code-Behind 的方式來撰寫。Web Services 的 Code Behind 寫法一樣包含兩個檔案, 第一個檔案是只有一行 WebService 指令的 .asmx 檔, 第二個檔案就是將 Web Services 的 Class 分開存放在副檔名爲 .vb 的檔案中, 接著將 .vb 檔案編譯成 DLL 之後再部署。

使用範例

下列範例以 Code-Behind 的方式製作了一個名爲 Math 的 Web Service, 並提供了一個 Square 方法可以傳回使用者傳入數值的平方。

.asmx 檔案內容：

```
<%@WebService Language="VB" Class="Charles.Math, Math"%>
```

程式 Math.asmx

上述 .asmx 檔案只有一行, 其中 Class 屬性的第一個參數爲類別所在命名空間以及類別名稱, 第二個參數爲 Assembly 名稱。

Code-Behind 檔案內容：

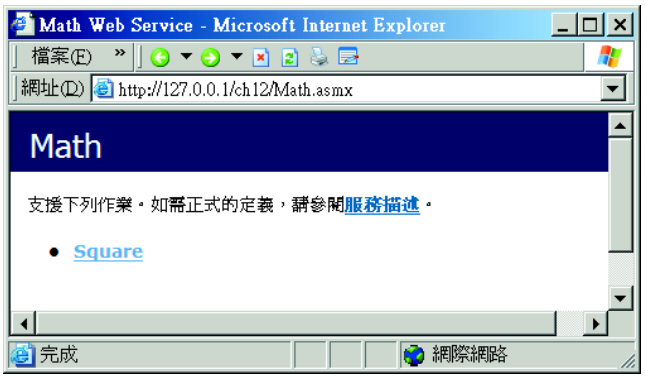
```
Imports System
Imports System.Web.Services

Namespace Charles

  <WebService(Namespace:="http://www.flag.com.tw")> _
  Public Class Math
    Inherits WebService

    <WebMethod()>Public Function Square(ByVal intA As Integer) _
                                                 As Integer
       Return intA * intA
    End Function

  End Class

End Namespace
```

程式 Math.asmx.vb

上述 Math.asmx.vb 的 Code-Behind 程式撰寫完畢後就可以使用下列語法編譯：

```
vbc /t:Library /r:System.dll,System.Web.Services.dll
/out:Math.dll Math.asmx.vb
```

編譯完畢後, 接著將 Assembly 檔案 Math.dll 部署至 bin 子目錄或 GAC 中才可使用。本範例將 Math.dll 部署至下一層子目錄中, 故注意右列畫面瀏覽的網址爲虛擬目錄：



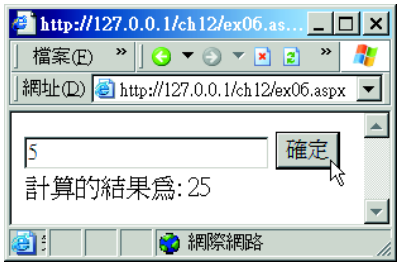
使用範例

下列範例示範 ASP.NET 網頁如何使用本機端編譯成 DLL 的 Web Service：

```
<%@Import  Namespace="chMath=Charles.Math"%>

<form runat="Server">
  <Asp:TextBox id="txtA" runat="Server"/>
  <Asp:Button id="btnOK" text="確定" OnClick="btnOK_Click"
      runat="Server"/><br>
  <Asp:Label  id="lblMSG"  Text="請輸入預計算平方的數"
      runat="Server"/>
</form>

<script language="vb" runat="Server">
Private Sub BtnOK_Click(sender As Object, e As EventArgs)
  Dim wsMath As New chMath()
  lblMsg.Text = "計算的結果爲: " & _
                wsMath.Square(CInt(txtA.Text)).ToString()
End Sub
</script>
```

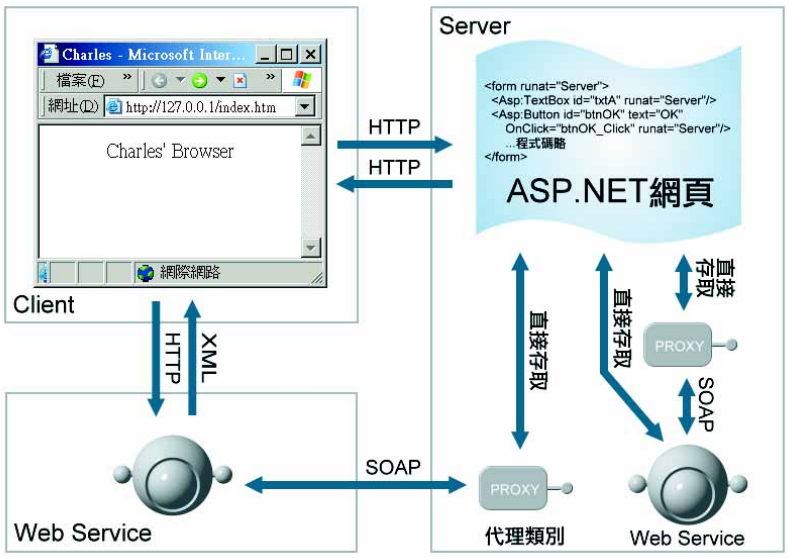
程式 EX06.aspx

上述範例爲了避免和 System.Math 類別造成名稱衝突, 所以將自訂的 Math 類別改成別名 chMath, 右圖爲執行結果：



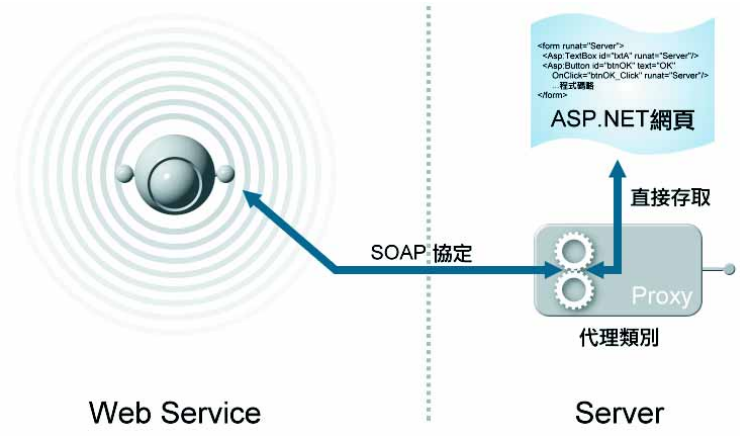
使用 Web Services

倘若 Web Services 存在於本機端並編譯成 DLL, 那麼直接叫用是最簡單的使用方法。倘若 Web Services 在 Internet 上的某個位址, 那麼產生一個「代理」(Proxy)類別會比較方便。 Web Services 一般都是由 ASP.NET 網頁或其它程式叫用, 以 ASP.NET 網頁而言是將結果透過 HTTP 協定回傳給使用者, 通常使用者不會直接去叫用 Web Services 。 Web Services 的執行模型如下圖所示：



代理類別

代理類別是一種包含轉送邏輯以及整理資料的程式，本身並不包含實作 Web Services 內容的程式碼。代理程式接受本機端程式的委託，和提供 Web Services 的一方透過 SOAP 協定連線並叫用 Web Method。待 Web Method 將執行完畢所產生的結果透過 SOAP 協定傳送回來後，再將以 XML 格式所傳回的結果轉換成實際的資料型別及內容，最後才將資料傳回給叫用的程式。下圖爲代理類別與本機端程式以及 Web Service 的互動示意圖：



使用 WSDL 工具產生代理類別

.NET Framework SDK 提供了一個可以自動產生定義代理類別原始碼的工具「WSDL.EXE」，位於「C:\Program Files\Microsoft.NET\FrameworkSDK\Bin」目錄中。 WSDL.EXE 會自動分析 Web Service 描述語言(WSDL)所定義的 Web Service，並自動產生可以存取 Web Service 的代理類別原始碼，而不用自己撰寫程式。 WSDL.EXE 的使用語法如下所示：

```
WSDL [功能] {URL | path}
```

其功能參數爲非必要性參數, 定義了產生原始碼的使用語言、輸出檔案名稱等設定, URL 或 path 參數爲必要參數, 用來指定 .wsdl 、.xsd 或 .disco 文件所在的位址。 WSDL.EXE 的常用功能如下表所示：

功 能	說 明
/d[omain]	當所連結的 Server 端需要驗證時所指定的 Domain 名稱
/l[anguage]	產生代理類別的原始碼所要使用的語言, 可選擇 CS(VC#預設值)、JS (JScript.NET)或是 VB(VB.NET)
/n[amespace]	代理類別要被安置的命名空間, 預設爲最外層(Global)
/o[ut]	所產生的原始碼檔名
/p[assword]	當所連結的 Server 端需要驗證時所指定的密碼
/protocol	產生代理類別所要使用的通訊協定, 可選擇 SOAP(預設值)、HttpGet、HttpPost 或在設定檔中所指定的自訂的協定
/proxy	要建立連線所使用的代理伺服器位址, 預設使用系統的設定
/pd	(或/proxydomain)代理伺服器所在的 Domain
/pp	(或/proxypassword)使用代理伺服器的密碼
/pu	(或/proxydomain)使用代理伺服器的使用者帳號
/u[sername]	當所連結的 Server 端需要驗證時所指定的使用者帳號

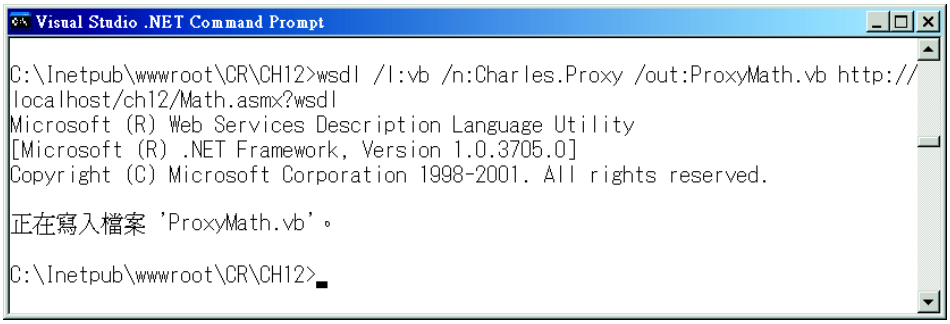
使用範例

下列範例產生存取 Math 這個 Web Service 的代理類別：

首先使用 WSDL.EXE 產生定義該類別的原始碼, 使用指令如下所示：

```
wsdl /l:vb /n:Charles.Proxy /out:ProxyMath.vb
http://localhost/ch12/Math.asmx?wsdl
```

上述指令指定使用 VB.NET 語言來產生代理類別的原始碼, 將所產生的類別群組到 Charles.Proxy 命名空間內, 並且將輸出的原始碼檔案命名爲 ProxyMath.vb, 最後指定代理類別所要參考的 wsdl 所在的位址。該指令的執行結果如下圖所示：



執行完畢後會產生以 VB.NET 所撰寫的代理類別原始檔 ProxyMath.vb, WSDL 工具會將 Web Service 的名稱當成類別名稱, 故本範例會產生一個 Math 類別。最後使用編譯器將原始碼編譯成 DLL, 其使用指令如下所示：

```
vbc /out:ProxyMath.dll /t:Library /r:System.dll,System.Web.
Services.dll,System.XML.dll  roxyMath.vb
```

注意上述指令爲同一行。編譯完畢後即可將該 Assembly 部署並使用, 本範例將 Assembly 部署至 bin 子目錄中。

使用範例

下列範例顯示如何透過代理類別存取 Web Services：

```
<%@Import  Namespace="chMath=Charles.Proxy.Math"%>

<form runat="Server">
  <Asp:TextBox id="txtA" runat="Server"/>
  <Asp:Button id="btnOK" text="確定" OnClick="btnOK_Click"
      runat="Server"/><br>
  <Asp:Label  id="lblMSG"  Text="請輸入預計算平方的數"
      runat="Server"/>
</form>

<script language="vb" runat="Server">
```

```
Private Sub BtnOK_Click(sender As Object, e As EventArgs)
  Dim wsMath As New chMath()
  lblMsg.Text = "計算的結果爲: " & _
                wsMath.Square(CInt(txtA.Text)).ToString()
End Sub
</script>
```

程式 EX07.aspx

上述程式碼除了類別所在的命名空間不一樣之外，其它程式和範例 EX06.aspx 使用本機端的 DLL 完全一樣, 表示使用代理類別可以讓叫用遠端的 Web Services 和使用本機端的類別一樣容易。

散佈 Web Services 的服務訊息

當你產生一個 Web Service 後, 這個 Web Service 就可以被其它的用戶端使用了。要推廣你的 Web Service 有兩個方式：第一個方式是透過散佈「Discovery 文件」, 第二個方式是使用「UDDI」。

製作 Discovery 文件

Discovery 文件以 XML 的格式儲存在副檔名「.disco」檔中, 其中包含 Web Services 的 WSDL 所在位址等資訊。要使用該 Web Service 的使用者將副檔名爲 .disco 的 Discovery 檔案下載後, 即可使用 WSDL.EXE 自動產生存取 Web Service 的代理類別。.NET Framework SDK 提供了一個可以自動產生 Discovery 文件的工具程式 DISCO.EXE, 位於「C:\Program Files\Microsoft.NET\Framework-SDK\Bin」目錄中。 DISCO.EXE 的使用語法如下所示：

```
DISCO [功能] URL
```

其功能參數爲非必要性參數, 定義了要如何產生 Discovery 檔案的設定, 而 URL 爲必要參數, 用來指定 Web Service 實際所在的位址。DISCO.EXE 的常用功能如下表所示：

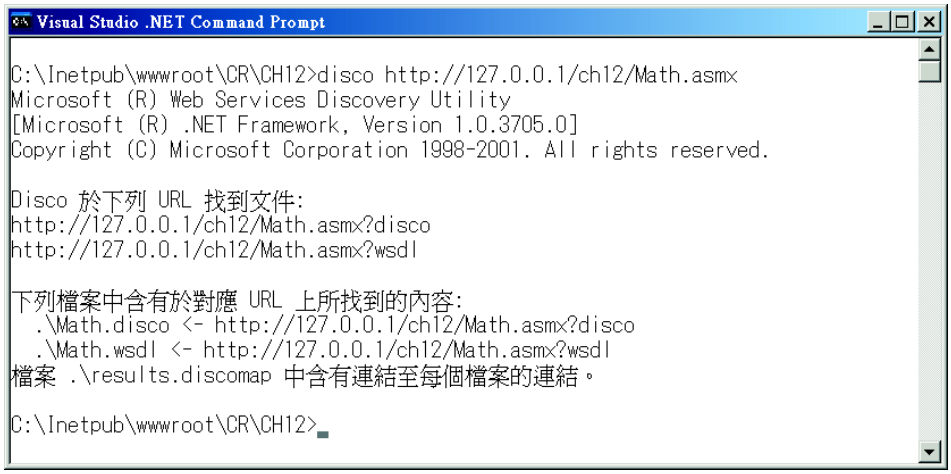
功 能	說 明
/d[omain]	當所連結的 Server 端需要驗證時所指定的 Domain 名稱
/nosave	只偵測是否有 Web Service 所提供的 WSDL 與 Discovery 資料存在, 並不存檔。
/o[ut]	所產生的檔案所要安置的目錄名稱
/p[assword]	當所連結的 Server 端需要驗證時所指定的密碼
/proxy	要建立連線所使用的代理伺服器位址, 預設使用系統的設定
/pd	(或/proxydomain)代理伺服器所在的 Domain
/pp	(或/proxypassword)使用代理伺服器的密碼
/pu	(或/proxydomain)使用代理伺服器的使用者帳號
/u[sername]	當所連結的 Server 端需要驗證時所指定的使用者帳號

使用範例

下列範例使用 DISCO.EXE 產生了 Math Web Service 的 WSDL 以及 Discovery 文件：

```
disco http://127.0.0.1/ch12/Math.asmx
```

注意, 實際在使用的時候要將位址「127.0.0.1」改成實際申請的 IP 位址。上述範例的執行畫面如下圖所示：



DISCO.EXE 自動產生了三個檔案, 其中 math.disco 爲 Discovery 文件, math.wsdl 爲 WSDL 文件, 而最後的 results.discomap 則記載了 DISCO.EXE 產生了哪些檔案。下列 Discovery 文件即是由工具程式 DISCO.EXE 所產生：

```
<?xml version="1.0" encoding="utf-8"?>
<discovery  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://schemas.xmlsoap.org/disco/">
  <contractRef  ref="http://127.0.0.1/ch12/math.asmx?wsdl"
         docRef="http://127.0.0.1/ch12/math.asmx"
         xmlns="http://schemas.xmlsoap.org/disco/scl/"  />
  <soap  address="http://127.0.0.1/ch12/math.asmx"
        xmlns:q1="http://www.flag.com.tw"
        binding="q1:MathSoap"
        xmlns="http://schemas.xmlsoap.org/disco/soap/"  />
</discovery>
```

程式 文件 math.disco

這兩個檔案製作完畢後, 就可以透過公開的網站將 Web Service 散播出去。

UDDI

要散播自訂的 Web Services 服務資訊或搜尋網際網路上已經存在的 Web Services, 可以使用「UDDI」(Universal Description, Discovery and Integration)。UDDI 是由 Ariba 、 IBM 、微軟三大廠商於 2000 年 9 月所提出的一個計畫, 後來加入了 Sun 、 Oracle 、 Compaq 、 HP 、 Intel 、 SAP 與其他三百多家公司參與, UDDI 使用 XML 技術來描述及整合網際網路上的 Web Services, 並可以讓欲使用 Web Services 的組織可以很容易就透過搜尋引擎找到所欲使用的資源。 UDDI 是針對全球 Web Services 的一個「描述、探索與整合」的公開標準, 可以登錄自行開發或搜尋他人所提供的商業服務或企業資訊。 UDDI 是一種不依賴任何平台的標準, 它提供了一個網際網路上的公開架構, 可以讓任何單位描述所提供的服務、讓需要服務的單位搜尋商業服務, 並整合這些商業服務 (UDDI 組織的網址 http://www.uddi.org)。

目前微軟以及 IBM 都有提供 UDDI 服務網站，這兩個網站的網址分別爲http://uddi.microsoft.com (微軟) 以及 http://uddi.ibm.com (IBM)，日後將會有 HP 等其他公司加入。

Web Services 的登錄及搜尋

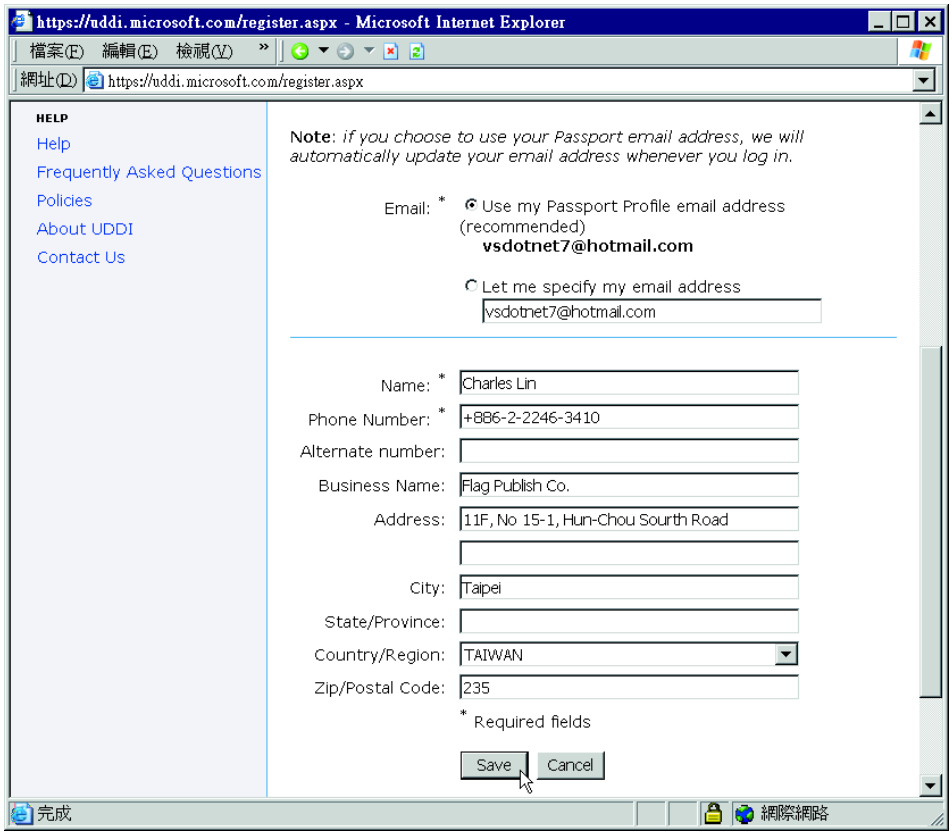
透過 UDDI 公用網站來散佈 Web Services 服務資訊是件很容易的事，接下來要透過微軟 UDDI 公用網站來散佈組織基本資訊以及所提供的 Web Services。首先瀏覽微軟 UDDI 公用網站，位於 http://uddi.microsoft.com：

微軟 UDDI 公用網站

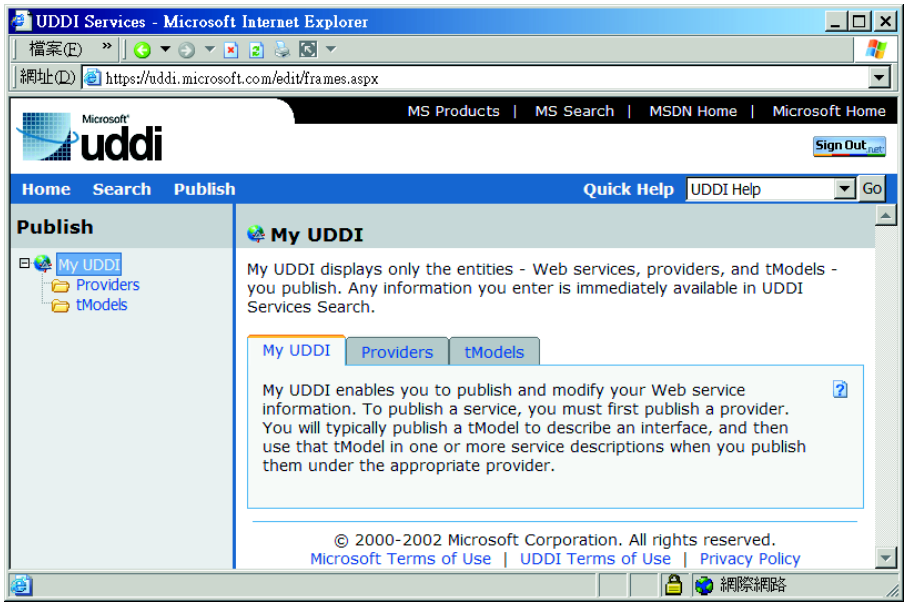
登錄 Web Services

在微軟 UDDI 公用網站的左側選擇「Register」(註册)選項登錄組織及 Web Services 資訊：

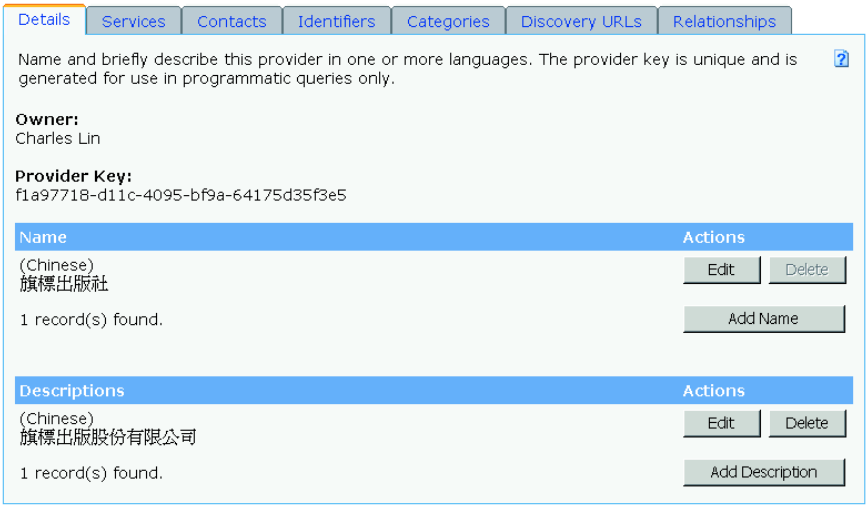
要在微軟 UDDI 上登錄組織及 Web Services 資訊，必需使用「.NET Passport」作爲驗證依據，若沒有 .NET Passport 請先申請。按下「Sign In」(簽入)使用 .NET Passport 登入後，則出現下列填表網頁：



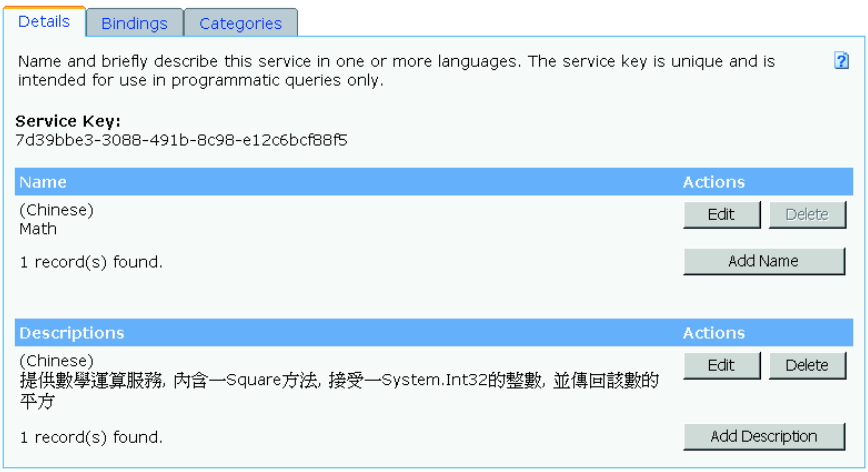
填入必填資料後按下「Save」儲存，若沒問題則會出現使用條款網頁，若同意條款內容則按下 Accept 同意後，即可開始登錄組織基本資料以及所提供的 Web Services 訊息：



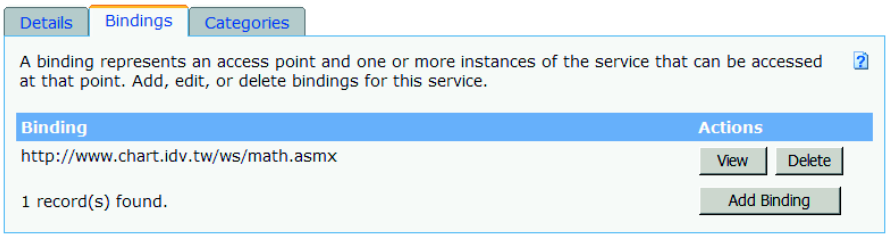
要登錄所提供的 Web Services 之前，首先要登錄提供者的資料。提供者可以是個人、團體、企業等任何提供 Web Services 的單位。提供者資料登錄單位資訊、所提供的服務、聯絡人等資訊：



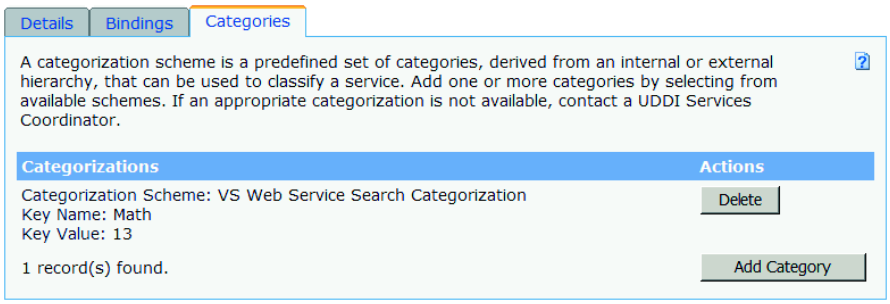
Services 頁籤用來登錄所提供的 Web Services, 下列畫面爲選擇 Service 頁籤後, 按下 Add Service 按鈕新增一個 Math 服務後的結果：



Bindings 頁籤爲指定 Math 這個 Web Service 的實際位置, 下圖爲指定後的畫面：

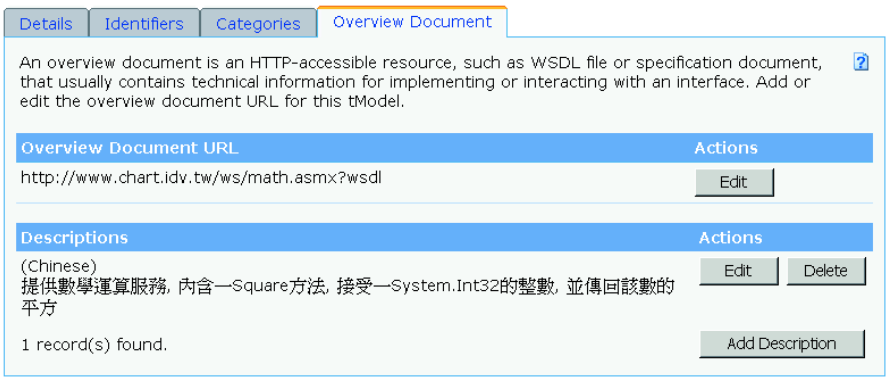


Categories 頁籤則是用來對 Web Service 進行分類, 讓需要服務的使用者可以快速的依分類找到。下圖將 Math 這個 Web Service 分類成 VS Web Service Search Categorization 目錄中的 Math 項目：

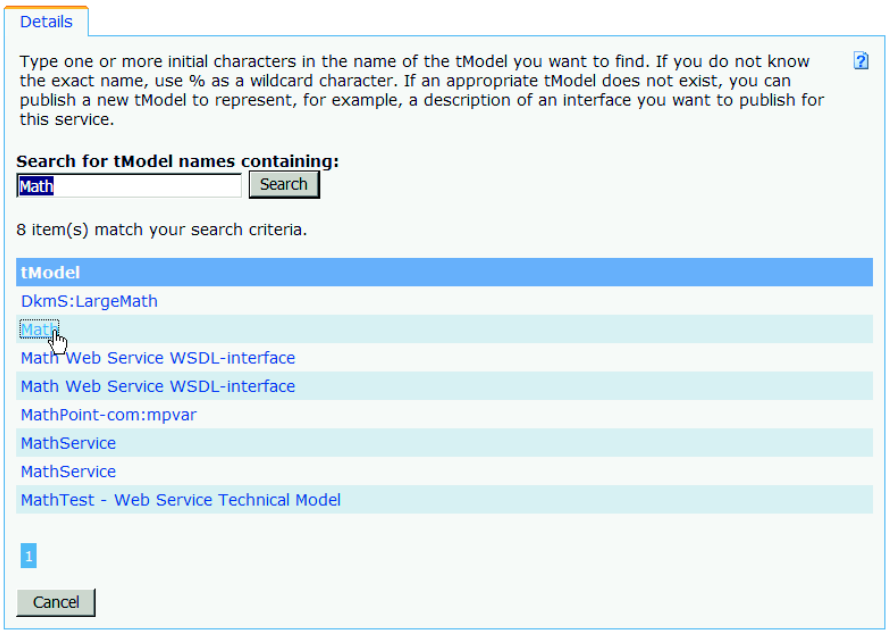


登錄 tModel

tModel (Technology Model) 技術模型是用來描述 Web Services 的一種抽象型別, 是一個類別模型。對 Web Services 而言, tModels 相當於 WSDL 檔。故些接下來將 WSDL 文件所在的位置加入後, Web Service 的登錄就成功了。按下 tModel 頁籤並新增一個 tModel 後, 一樣可以編輯 tModel 的許多資訊, 下圖爲選擇「Overview Document」頁籤並指定 WSDL 文件所在位置：

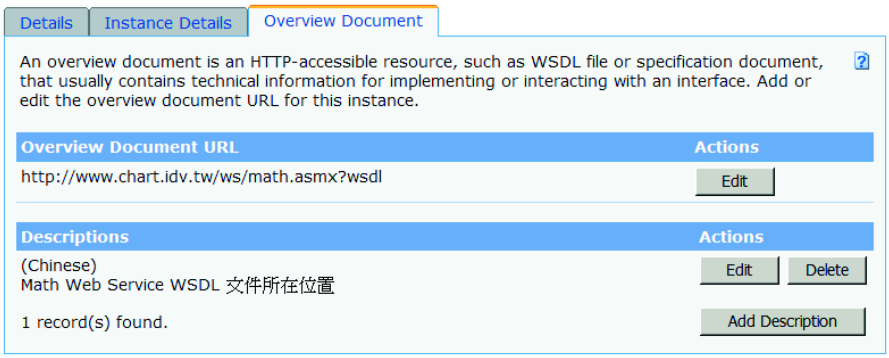


最後將 Instance Info (實體資訊)指向剛才所設定的 tModel 後即可完成登錄。Instance Info 用來結合 Web Services 和技術模型文件(即 WSDL 檔案)。使用者在找到 Web Services 的描述時, 即可得知 Web Services WSDL 文件從何方取得。 Instance Info 的設定需要輸入一個 tModel 的名稱, 按下 Search 按鈕後會出現所有符合的 tModel, 這邊當然是選擇剛才所建立的 tModel：

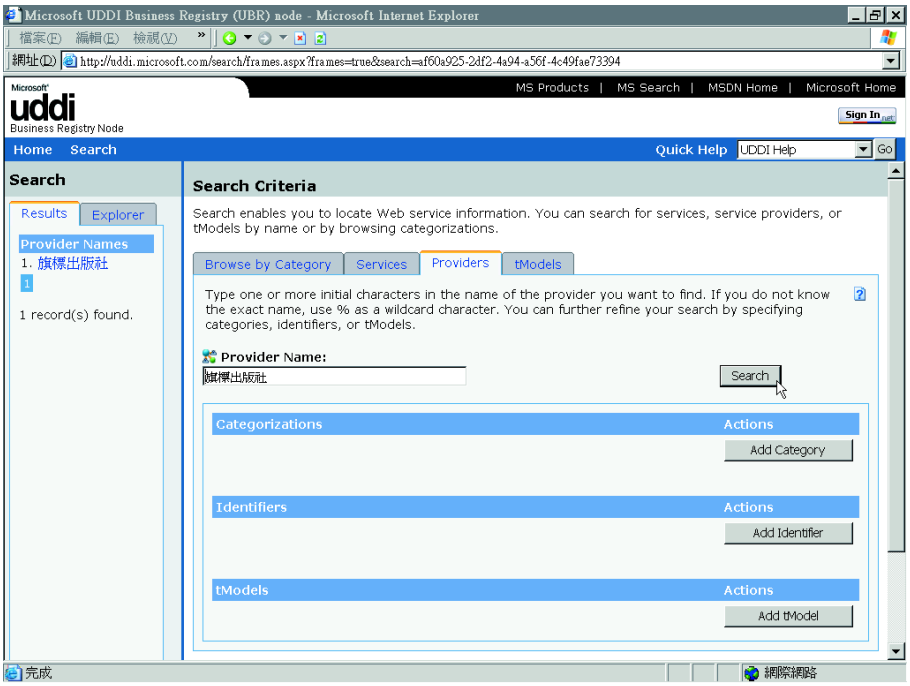


指定完成後接下來切換到 Instance Details 頁籤, 這裡可以輸入 Web Service 所需要的參數說明, 或是直接輸入含有該 Web Service 使用說明的網址：

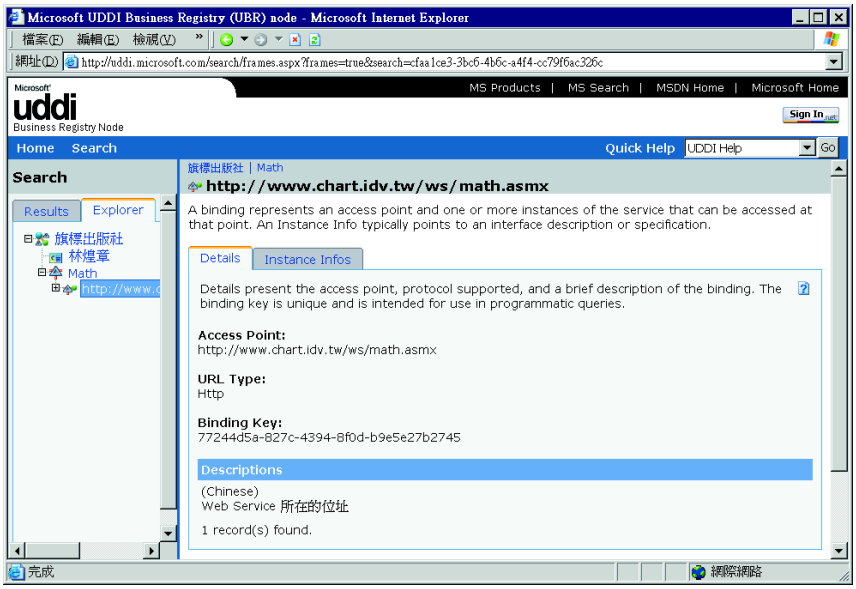
最後切換到 Overview Document 頁籤, 這裡輸入 Web Service 的 WSDL 文件所在位址即可：



登錄完成後，即可在微軟 UDDI 網站上找到自己所登錄的資料。下圖瀏覽微軟 UDDI 網頁的首頁後，點選 Search 項目搜尋 Providers 的結果：

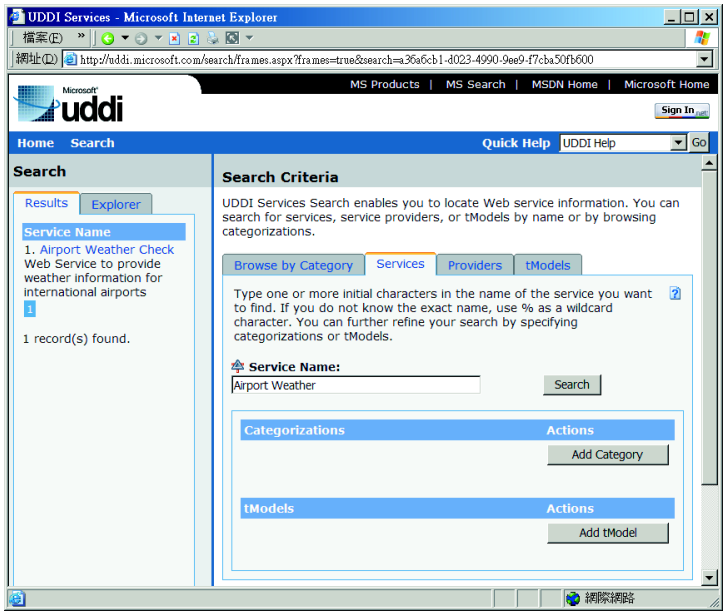


搜尋的結果顯示在左側頁框，按下搜尋結果後即可瀏覽所登錄的資訊：

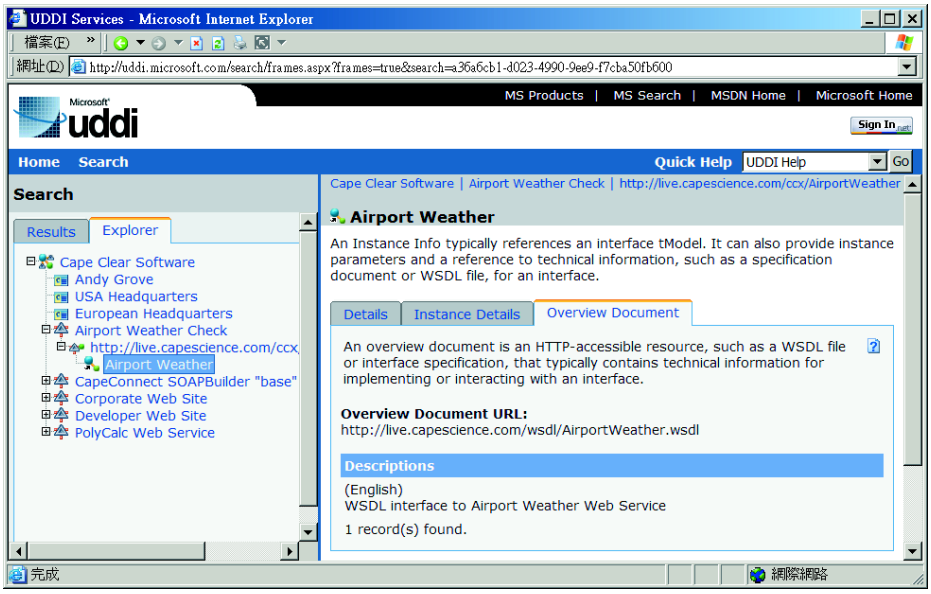


搜尋及使用 Web Services

了解如何登錄及搜尋 Web Services 後，接下來要實際搜尋及使用網際網路上現有的 Web Services。若要搜尋查詢國際機場天氣的 Web Services，則點選 Services 頁籤後並在 Service Name 文字輸入盒中輸入 Airport Weather，搜尋有關機場天氣的 Web Services。右圖爲搜尋的結果：



搜尋的結果顯示有一個「Airport Weather Check」符合搜尋的條件，點選該項目並選擇所需要的 Airport Weather Check 項目後，即出現下列畫面：



提供 Web Services 的公司爲「Cape Clear Software」，該公司有三位聯絡人外，並提供了五個 Web Services。查到 Web Service "Airport Weather Check" 的 WSDL 文件檔案位置 http://live.capescience.com/wsdl/AirportWeather.wsdl 後，即可使用 WSDL.EXE 工具來產生代理類別。首先使用 WSDL.EXE 工具來產生代理類別的原始檔：

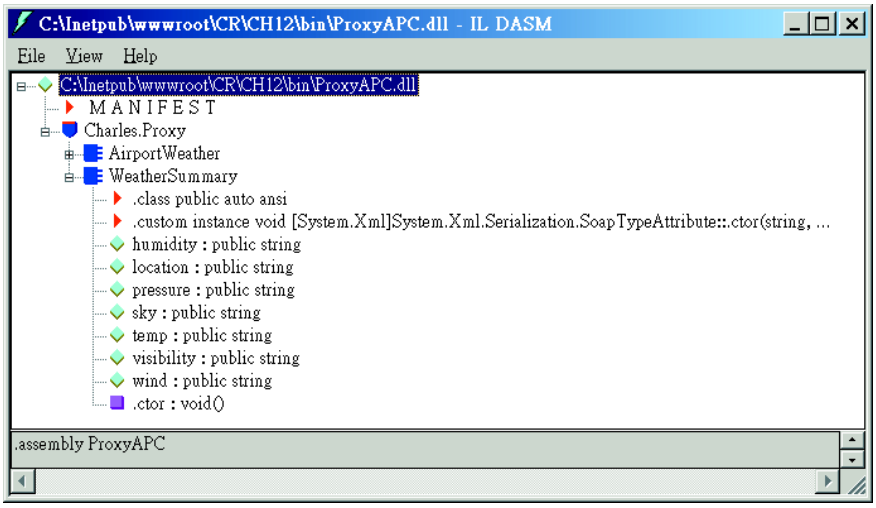
```
wsdl /l:vb /n:Charles.Proxy /out:ProxyAPC.vb
     http://live.capescience.com/wsdl/AirportWeather.wsdl
```

以上敘述在同一行，會產生原始檔 ProxyAPC.vb，並將類別群組至 Charles.Proxy 命名空間中。接下來使用 VB 編譯器將原始碼 ProxyAPC.vb 編譯成 Assembly：

```
vbc /out:ProxyAPC.dll /t:Library /r:System.dll,
    System.Web.Services.dll,System.XML.dll ProxyAPC.vb
```

以上敘述在同一行, 會產生 Assembly 檔 ProxyAPC.dll, 並將 ProxyAPC.dll 移至下一層的 bin 目錄中即可。「AirportWeather」這個 Web Service 的使用方法可以在網址 http://www.capescience.com/webservices/airportweather 中查詢：

AirportWeather Web Service 提供了許多方法, 這些方法都接受以機場代號爲參數, 例如可以取得風速的 getWind 方法以及取得氣溫的 getTemperature 方法。除了可以取回單項資料的 get 方法外，還提供傳回比較詳細資料的 getSummary 方法, 該方法傳回 WeatherSummary 型別的物件, 該物件有諸如濕度、壓力、氣候、能見度以及風速等屬性。關於這些類別的資訊，可以透過 ILDASM.EXE 來檢視 ProxyAPC.dll 這個 Assembly：



下表列出 AirportWeather Web Service 所提供的常用 Web Method：

Web Method	說 明
getHumidity	取得機場濕度
getPressure	取得機場壓力
getSky	取得機場氣候
getSummary	取得詳細資料
getTemperature	取得機場溫度
getVisibility	取得機場能見度
getWind	取得機場風速

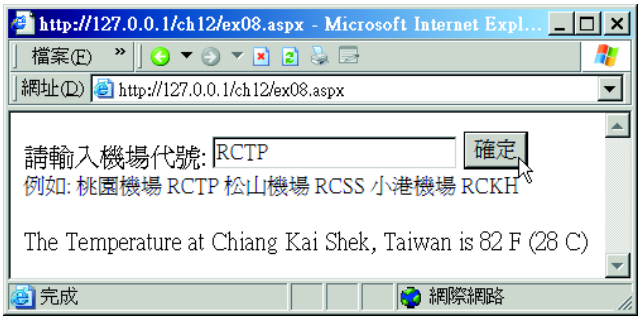
其實透過 Web Method 的名稱即可知道所提供的服務爲何，這就是命名法則的重要性。這些方法都接受國際機場代號爲參數，台灣的國際機場代號爲：桃園機場爲 RCTP 、松山機場爲 RCSS 、小港機場則爲 RCKH, 其它國際機場的代號可至 http://www.ar-group.com/icaoiata.htm 查詢。

使用範例

下列範例示範如何透過代理類別使用 AirportWeather 這個 Web Service 所提供的國際機場天氣查詢服務：

```
<%@Import Namespace="Charles.Proxy"%>

<form runat="Server">
  請輸入機場代號: <Asp:TextBox id="txtAP" runat="Server"/>
  <Asp:Button id="btnOK" text="確定" OnClick="btnOK_Click"
      runat="Server"/><br>
  <font size=2>
  例如: 桃園機場 RCTP 松山機場 RCSS 小港機場 RCKH
  </font><p>
  <Asp:Label id="lblMSG" runat="Server"/>
</form>

<script language="vb" runat="Server">
Private Sub BtnOK_Click(sender As Object, e As EventArgs)
  Dim wsAW As New AirportWeather()
  lblMsg.Text = wsAW.getTemperature(txtAP.Text)
End Sub
</script>
```

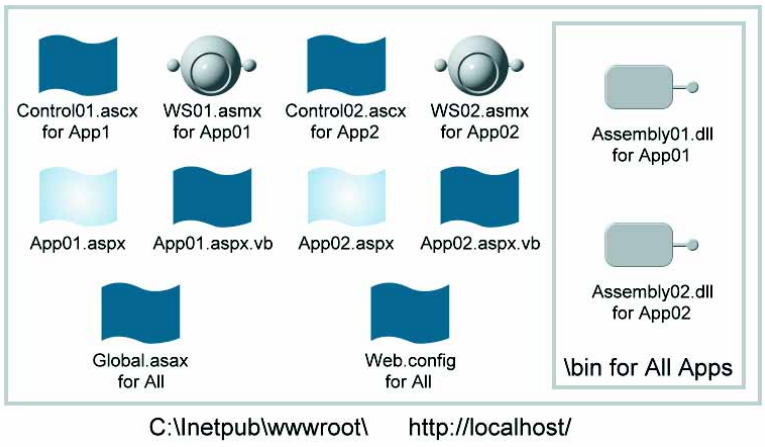
程式 EX08.aspx



12-4 Web 應用程式

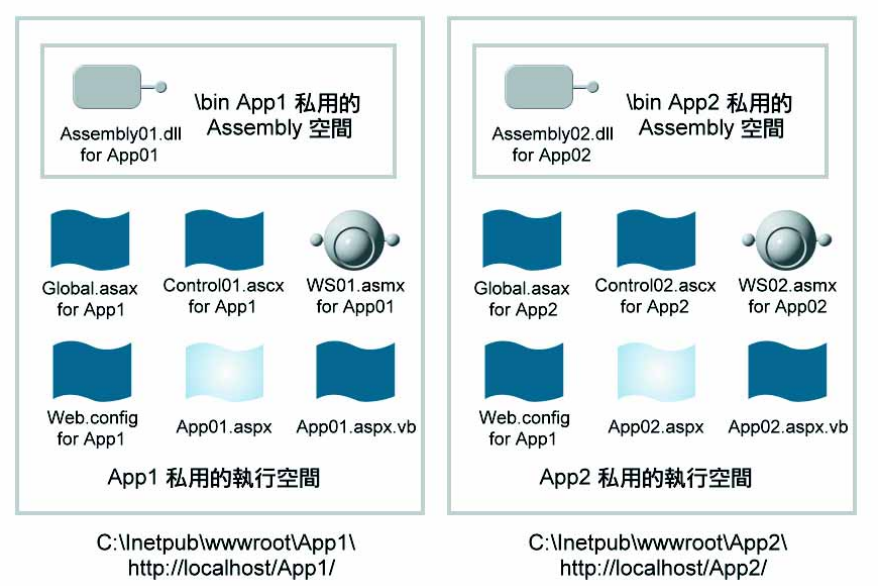
Web 應用程式的優點

一個網站可能提供許多不同用途的網頁，這些網頁又可能組成一些 Web 應用程式，而這些 Web 應用程式在執行時又需要許多資源。例如 ASP .NET Web 應用程式是由許多 ASP.NET 網頁、Web Services、自訂控制項、Assemblies 以及設定檔等所組成。若將這些 ASP.NET Web 應用程式全部存放在同一個地方，那麼這些 ASP.NET Web 應用程式將會發生許多問題。例如無法針對特定 ASP.NET Web 應用程式指定特定層級的安全策略、 ASP .NET 應用程式不易部署及設定、檔案雜亂不易管理等問題。



不易設定、部署、管理及維護

這些問題的最佳解決方案，就是爲每一個 Web 應用程式指定一個特定的執行空間，在這個空間內所使用環境都是針對該 Web 應用程式所設計，例如特定的設定檔、統一的安全模型、應用程式私有的資源等，這樣一來 Web 應用程式不但容易部署及設定，也容易管理及維護。



容易設定、部署、管理及維護

要讓每個 Web 應用程式擁有自己的執行空間, 最簡單的方法就是使用 IIS 服務管理員建立虛擬目錄(前面章節使用過), 或是將路徑 C:\Inetpub\wwwroot\ 之下任意階層的子目錄之屬性設定爲應用程式。這樣一來每個 Web 應用程式皆有自己的執行空間, 例如本書第十一章所做的示範, 將目錄 C:\Inetpub\wwwroot\cr\ch11 指定爲虛擬目錄並給予別名 ch11 。這樣一來 ch11 即變成一個 Web 應用程式, 實際執行時 Web 應用程式 ch11 的位址即爲 http://127.0.0.1/ch11/, 並擁有自己的私有執行空間。例如在下一層 bin 目錄中可存放私有的 Assemblies 、可以擁有自己的 Web.config 設定檔以及 Global.asax 全域事件檔, 並且可以擁有自己的安全層級設定等。

設定 Web 應用程式

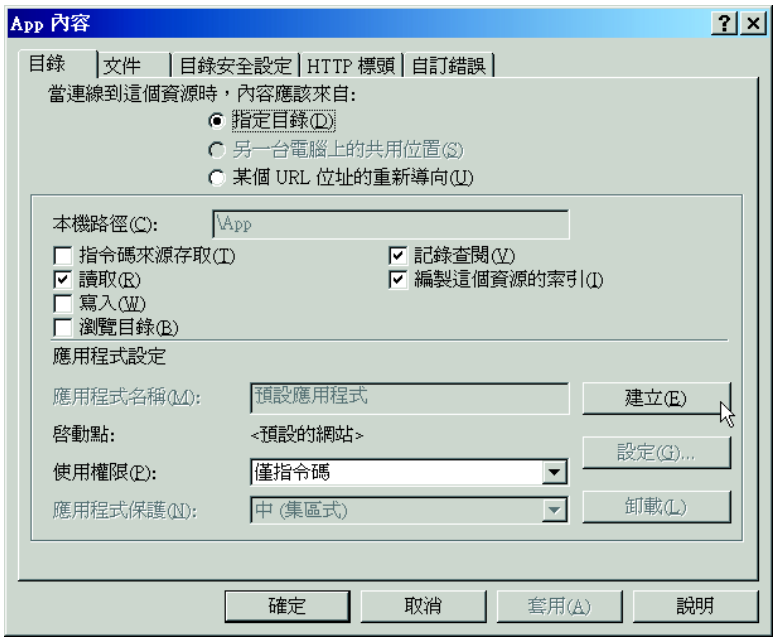
Web 應用程式根目錄在安裝 IIS 之後預設爲 http://localhost/ 或 http://機器名稱 / 或 http://127.0.0.1/ , 在網際網路上的實際位址爲 http:// 實際網址 / , 而實際的路徑則對應到 C:\Inetpub\wwwroot\ 。 Web 應用程式的根目錄包含了許多檔案及設定, 例如 Web.config 以及存放 Assemblies 的子目錄 bin 。

要自訂 Web 應用程式, 可以使用前面所使用建立虛擬目錄的方法, 或直接將位於 C:\Inetpub\wwwroot\ 目錄之下的子目錄設定爲 Web 應用程式。以指定虛擬目錄的方式設定 Web 應用程式, Web 應用程式的目錄不需要在路徑 C:\Inetpub\wwwroot\ 而可以在任何地方, 只要設定虛擬目錄時指定該目錄的的 別名 , 即會對應到 http://Inetpub/wwwroot/ 別名 / 的位址。

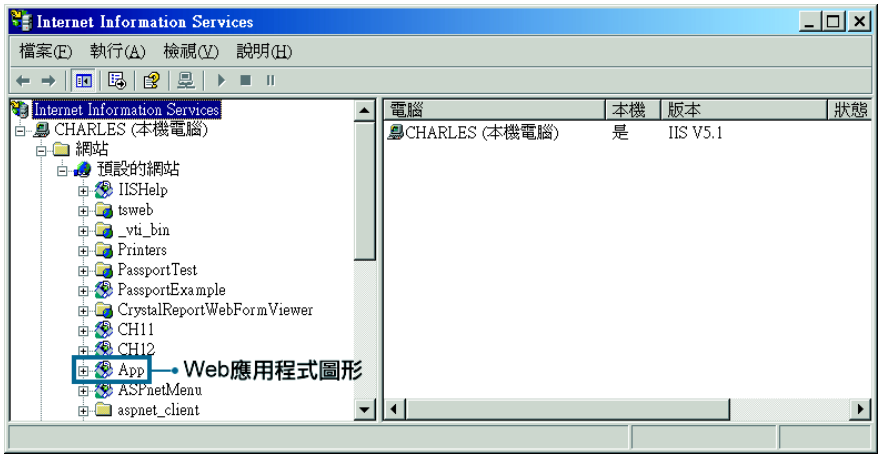
使用範例

下列範例示範如何以設定目錄屬性的方式建立 Web 應用程式：

首先在 C:\Inetpub\wwwroot\ 目錄之下建立一個子目錄 App, 接著開啓 IIS 服務管理員並展開「網站」->「預設的網站」後即可發現目錄 App 。此時在 App 目錄上按滑鼠右鍵並選擇「內容」, 即出現下列視窗：



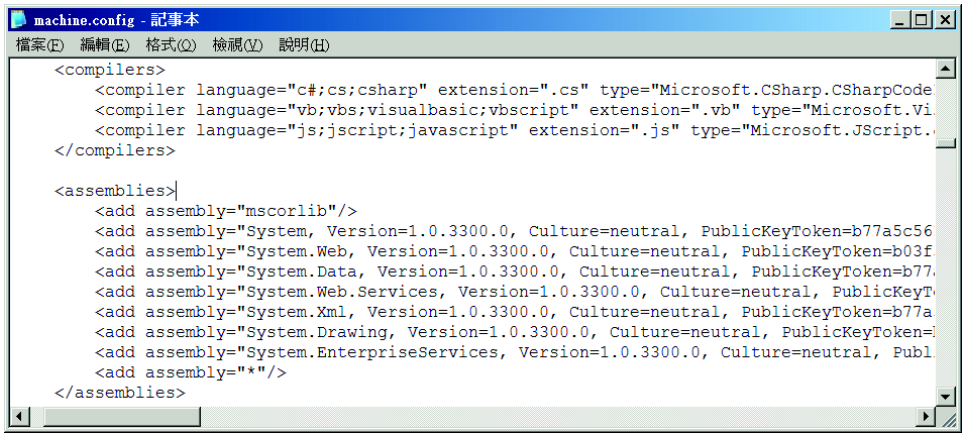
按下「建立」按鈕後, 該目錄即成爲一個 Web 應用程式：



若要移除 Web 應用程式, 只要在內容視窗中按下「移除」按鈕即可。

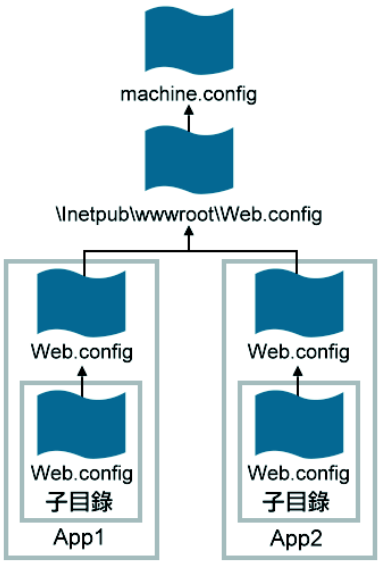
ASP.NET 網頁設定

ASP.NET 網頁使用 XML 來作爲設定格式, 所以大小寫有分別, 並使用許多元素來標明每個設定的區段。每一台 Server 都有一個名爲「machine.config」的主設定檔, 存放於「C:\Win 目錄\Microsoft.NET\Framework\版本編號\CONFIG\」目錄之下, 這個設定檔中存放了所有 ASP.NET 應用程式的設定值, 修改這個主要的設定檔會引影響所有的應用程式, 例如編譯時預設載入的 Assemblies 等：



machine.config 預設在編譯時加入的 Assemblies

除了主要的設定檔 machine.config 之外, 倘若在 Web 伺服器的根目錄中(一般位於 C:\Inetpub\wwwroot\) 已經有一個 Web.config 設定檔, 那麼所有的 Web 應用程式將會被這個設定檔所影響。 Web 伺服器根目錄及 Web 應用程式資料夾內的 Web.config 檔是選擇性的, 除了 Web 伺服器根目錄中的 Web.config 設定之外, 每個 ASP.NET 應用程式也可以在自己的 Web 應用程式執行空間中擁有自己的 Web.config 檔, 而 Web 應用程式內的子目錄也可以擁有自己的 Web.config 檔。ASP.NET 網頁執行時, 若所執行的子目錄中有 Web.config 設定檔, 則子目錄的設定檔所指定的設定值將會覆寫父目錄設定檔內的設定值, 其餘設定將會繼承父目錄的設定。



Web.config 檔案的繼承示意圖

Web.config 檔案的基本結構如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <元素名稱1>
      <子元素名稱
        屬性名稱1=設定值
        屬性名稱N=設定值  />
    </元素名稱1>
    <元素名稱2
        屬性名稱1=設定值
        屬性名稱N=設定值  />
    <元素名稱N
        屬性名稱1=設定值
        屬性名稱N=設定值  />
  </system.web>
</configuration>
```

Web.config 中經常使用的設定元素, 其說明如下表所示：

元素	說明
<appSettings>	可以加入自訂的設定, 並使用 ConfigurationSettings 類別的 AppSettings 方法讀入
<authentication>	設定 ASP.NET 應用程式的驗證模式
<authorization>	設定 ASP.NET 應用程式的授權
<compilation>	ASP.NET 的編譯設定, 例如<compilation>內的<assemblies>子標籤, 可以在編譯時加入指定的 Assemblies
<customErrors>	當 ASP.NET 網頁發生錯誤時, 可以自訂所要顯示的錯誤訊息
<globalization>	用來控制區域設定
<httpModules>	設定要加入的 HTTP 模組, 讓 ASP.NET 應用程式可以直接叫用模組中的 Shared 成員。例如 machine.config 預設已經加入了輸出快取、Session 狀態管理、驗證及授權用的模組
<identity>	設定匿名使用者是否要使用特定的 Windows 使用者帳號, 讓程式可以在自訂的安全層級執行, 本功能預設爲關閉
<sessionState>	設定應用程式的 Session 狀態
<trace>	設定 ASP.NET 的 trace 服務

使用範例

下列範例示範如何在 Web.config 檔案中使用自訂的設定, 並在 ASP.NET 網頁讀取使用：

```
<!--Web.Config Configuration File-->
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <appSettings>
     <add key="FirstName" value="Charles" />
     <add key="LastName" value="Lin" />
   </appSettings>
   <system.web>
      <customErrors mode="Off"/>
      <globalization requestEncoding="big5"
                     responseEncoding="big5"/>
   </system.web>
</configuration>
```

上述 Web.config 檔案只要新增一個「<appSettings>」區段即可, 另外特別注意「<appSettings>」區段所在的位置位於「<configuration>」標籤的下一個階層。若加入的資料只有一個, 則可以使用「<appSettings key=" 鍵值 " value=" 值 "/>」的方式寫成一行。

```
<script language="vb" runat="Server">
Private Sub Page_Load(sender As Object, e As EventArgs)
  Dim  strFN As String = _
       ConfigurationSettings.AppSettings("FirstName")
  Dim  strLN As String = _
       ConfigurationSettings.AppSettings("LastName")
  Response.Write("Your name is: " & strFN & " " & strLN)
End Sub
</script>
```

程式 EX09.aspx

上述範例執行結果如右圖所示：

使用範例

下列範例示範如何使用自訂的錯誤顯示網頁：

```
<!--Web.Config Configuration File-->
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.web>

     <customErrors
  defaultRedirect="Http://www.chart.idv.tw/err/ErrMsg.aspx"
       mode="On"/>
   <globalization requestEncoding="big5"
                  responseEncoding="big5"/>
   </system.web>
</configuration>
```

上述設定檔將自訂錯誤模式開啓，並指明發生錯誤時將網址重新導向至Http://www.chart.idv.tw/err/ErrMsg.aspx 外, 還會傳遞一參數 aspxerrorpath 用來指明發生錯誤的路徑及網頁。

```
<head>
  <title>Error</title>
</head>

<body>
<h2>錯誤</h2>
對不起, 您所瀏覽的網頁<Asp:Label id="lblMsg" runat="server"/>
發生錯誤!

<script language="vb" runat="Server">
Private Sub Page_Load(sender As Object, e As EventArgs)
  lblMsg.Text = Request("aspxerrorpath")
End Sub
</script>
</body>
```

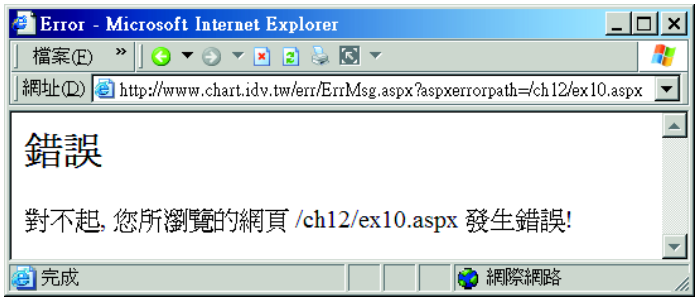
程式 ErrMsg.aspx

引發錯誤的網頁 EX10.aspx：

```
<form runat="Server">
  <Asp:Button id="btnErr" text="引發錯誤"
       OnClick="btnErr_Click" runat="Server"/>
</form>

<script language="vb" runat="Server">
Private Sub btnErr_Click(sender As Object, e As EventArgs)
  Throw New Exception()
End Sub
</script>
```

程式 EX10.aspx

下列爲按下引發錯誤按鈕後，被重新導向的畫面：



除了指定通用的錯誤所重新導向的網頁外，還可以指定發生特定錯誤所要重新導向的網頁，只要加上「<error/>」子元素的宣告即可。下列<custom-Errors>元素設定當發生狀態碼爲 404，所要求的資源不存在於伺服器時重新導向至指定的位址：

```
<customErrors
defaultRedirect="Http://www.chart.idv.tw/err/ErrMsg.aspx"
    mode="On">
    <error
      redirect="指定的URL">
      statusCode="404"/>
<customErrors/>
```

關於其它 HTTP 狀態碼，可以查詢 .NET Framework 文件「HttpStatusCode」列舉型別。

身份驗證及授權

要讓每個使用者可以依據系統授與的權限瀏覽合適的網頁，防止使用者瀏覽非經授權的網頁，可以使用 ASP.NET 所提供的「驗證」(Authentication)及「授權」(Authorization)功能。驗證是確認使用者身分的一個過程，當使用者出示系統所支援的證件(如帳號密碼或 Passport)時，系統即對該證件進行查核的動作，並依據使用者的身分給予一個適當的使用權限。

ASP.NET 驗證模式

ASP.NET 支援三種驗證模式, 分別爲 Windows 驗證、 Passport 驗證以及 Forms 驗證模式。第一種 Windows 驗證結合了 Windows 的驗證機制, 透過這種驗證方式幾乎不用寫什麼程式, 不過由於驗證的資料不是被 ASP.NET 應用程式所管理, 使用這種驗證方法在部署 ASP.NET 應用程式時需要額外的處理。第二種 Passport 驗證則使用微軟所提供的驗證服務, 微軟對於使用 Passport 服務的會員網站提供一個集中的驗證服務, 事實上這個服務就是一個 Web Service 。若要使用 Passport 驗證服務則要下載 Passport SDK, 可至「http://msdn.microsoft.com/downloads/」的「Software Development Kit」項目找到。第三種 Forms 驗證模式則可以使用自訂的資料庫或 Web.config 內所存放的資料進行使用者身分驗證, 這種驗證方式要先指定一個登入網頁, 只要有任何未通過驗證的 Request 就會被重新導向到登入網頁, 直到通過驗證才可以繼續瀏覽。指定驗證模式的 Web.config 設定語法如下所示：

```
<!--Web.Config Configuration File-->
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <authentication  mode="驗證模式"/>
  </system.web>
</configuration>
```

其中驗證模式的設定值爲「None」(無)、「Windows」(Windows 驗證)、「Forms」(Forms 驗證)或「Passport」(Passport 驗證), 並注意大小寫有分別。

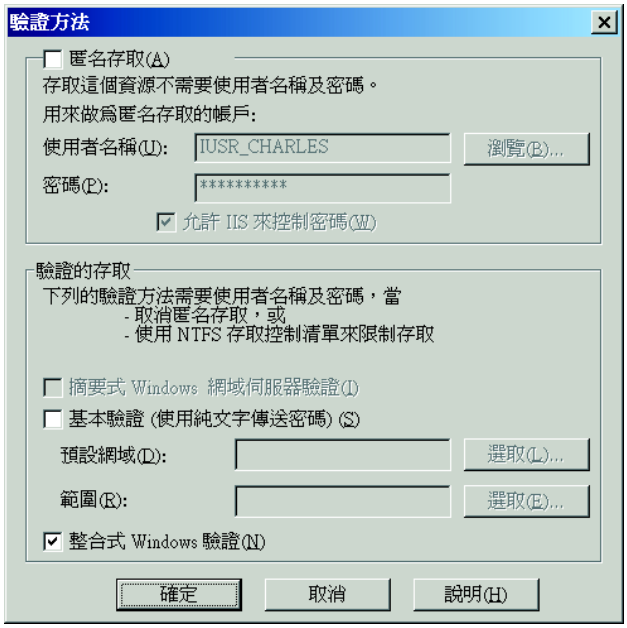
Windows 驗證模式

Windows 驗證模式主要是搭配 Windows 所建立的帳號, 並透過 IIS 的設定來完成驗證。要支援 Windows 驗證模式, 要使用 IIS 服務管理員除去 Web 應用程式的匿名存取的選項, 並且建立 Windows 的使用者帳號供驗證使用, 最後使用檔案總管設定應用程式所屬資料夾的權限即可。

使用範例

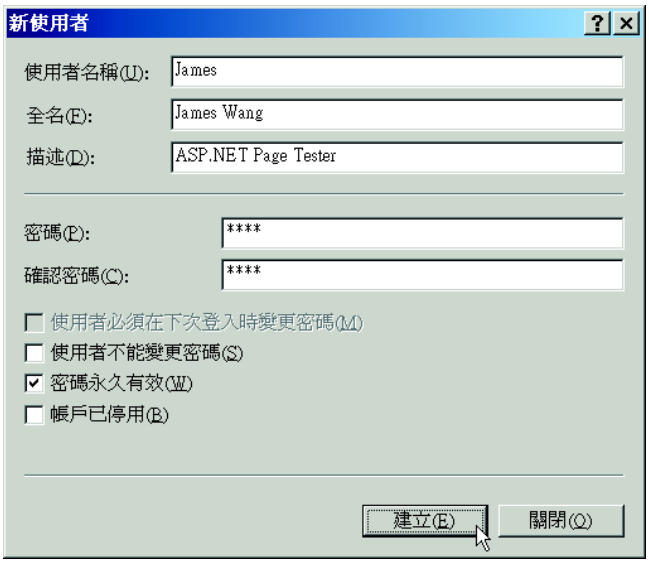
下列範例示範如何使用 Windows 驗證模式來保全 Web 應用程式 App 內的網頁：

首先建立一 Web 應用程式 App, 並使用 IIS 服務管理員顯示 App 的內容後, 選擇「目錄安全設定」頁籤, 點選「編輯」按鈕出現「驗證方法」視窗後, 將「匿名存取」的勾選取消：



接開啓「控制台」→「系統管理工具」→「點腦管理」項目後，並在「系統工具」→「本機使用者和群組」→「使用者」項目中按右鍵，並點選「新使用者」項目新增使用者：

輸入使用者帳號及密碼等資訊，並點選「密碼永久有效」項目後，按下「建立」按鈕：



接著將測試網頁 AuthWindows.aspx 複製到 APP 目錄之下, 測試網頁的原始碼如下所示：

```
<script language="vb" runat="Server">
Private Sub Page_Load(sender As Object, e As EventArgs)
  Dim strLogonUser As String = _
    Request.ServerVariables("Logon_User")
  Dim strAuthUser As String = _
    Request.ServerVariables("Auth_User")
  Response.Write("登入使用者爲: " & strLogonUser)
  Response.Write("<br>驗證的身份爲: " & strAuthUser)
End Sub
</script>
```

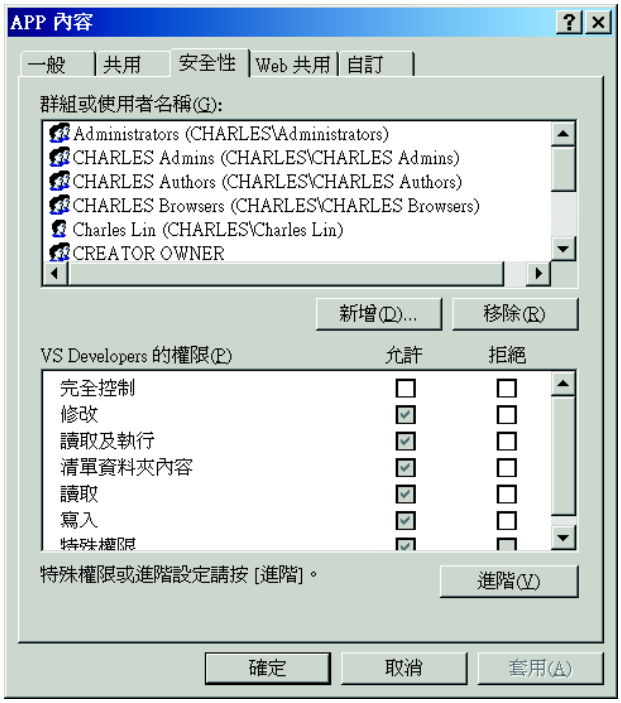
程式 AuthWindows.aspx

倘若使用者通過驗證, 則會顯示使用者的登入身份以及驗證後的身份資訊。下列爲本範例 web.config 檔的設定內容：

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Web.Config Configuration File-->
<configuration>
  <system.web>
    <authentication mode="Windows"/>
    <globalization requestEncoding="big5"
                   responseEncoding="big5"/>
  </system.web>
</configuration>
```

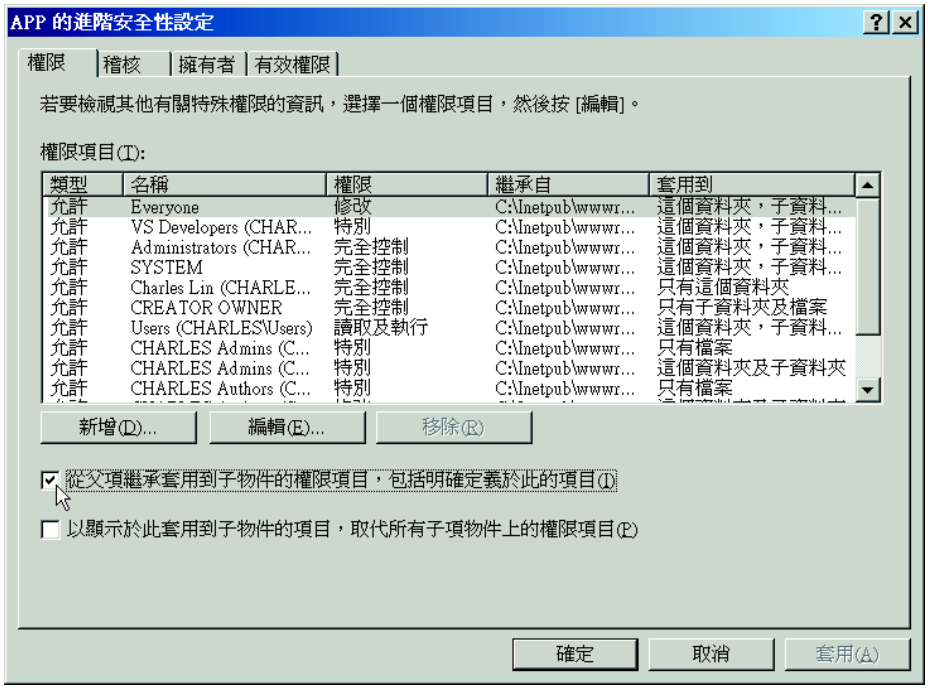
程式 Web.config

建立完畢後接著要指定 App 目錄的存取權限。開啓檔案總管後瀏覽 App 目錄所在的位址「C:\Inetpub\wwwroot\App」, 接著在目錄 App 上按下滑鼠右鍵並選擇「內容」後, 在出現的「APP 內容」對話盒中切換到「安全性」頁籤：

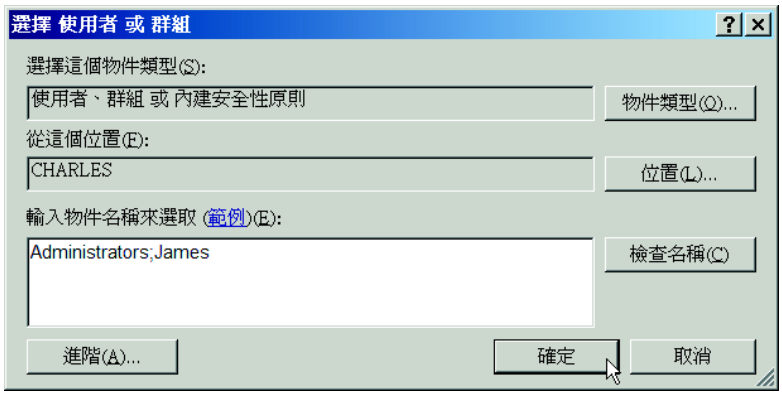


Windows XP 必需修改檔案總的設定才會出現安全性頁籤。首先開啓檔案總管並選擇「工具」->「資料夾選項」, 出現「資料夾選項」對話盒後切換到「檢視」頁籤, 並將「使用簡易檔案共用(建議使用)」的項目取消勾選即可。

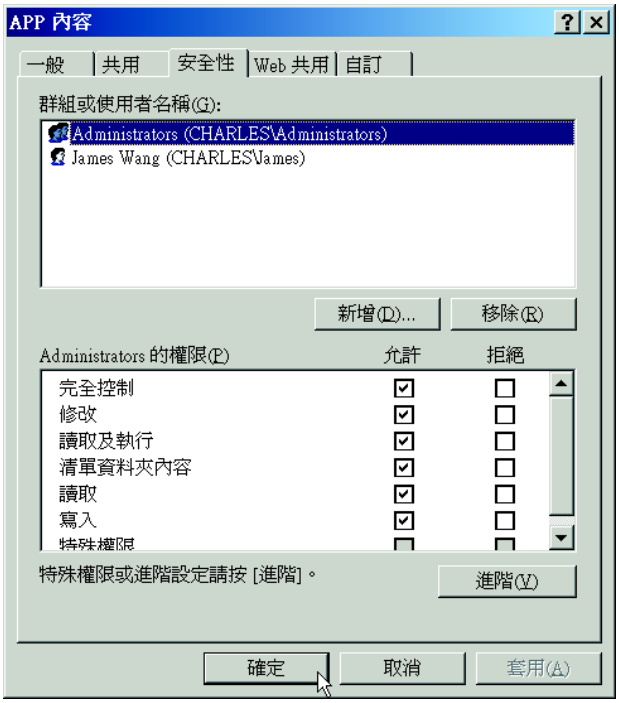
安全性頁籤列出了可以存取該目錄的使用者以及其權限, 按下「進階」按鈕出現「APP 的進階安全設定」對話盒後, 將「從父項繼承套用到子物件的權限項目, 包括明確定義於此的項目」選項取消勾選後, 即可移除所有該目錄的預設權限：



移除所有的使用者後，接下來必需加入兩個使用者。按下「APP 內容」對話盒內的新增按鈕，在出現的「選擇使用者或群組」對話盒中的「輸入物件的名稱來選取」文字輸入盒內輸入「Administrators;James」後按下「確定」：

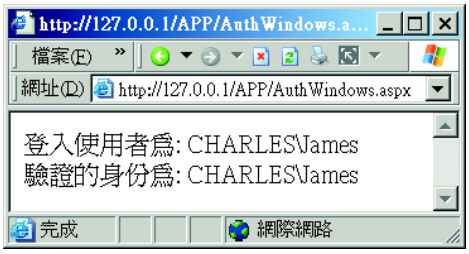


帳號建立完畢後，將「Administrators」群組的權限全部打開，以確保執行 ASP.NET 的帳號「ASPNET」可以在該目錄內執行程式，而帳號「James」不需要作任何更動：



最後開啓瀏覽器，在網址列輸入 Http://127.0.0.1/APP/AuthWindows.aspx 後，即出現詢問帳號及密碼的對話盒：

輸入正確的密碼及帳號後即可瀏覽網頁，否則出現「您沒有檢視此網頁的授權」訊息。右圖爲輸入正確帳號及密碼的結果：



安裝了 .NET Framework SDK 後，安裝程式會在系統中建立一個名爲「ASPNET」的帳號，該帳號屬於 Windows 的 Administrators 群組，ASP.NET 在執行程式時就是透過這個帳號來和 Windows 互動。若想要自訂 ASP.NET 執行時的 Windows 權限，則可以使用「<identity>」標籤。其宣告語法如下所示：

```
<system.web>
  <identity impersonate="true"
    userName="欲使用的 Windows 帳號"
    password="密碼"/>
</system.web>
```

設定完畢後就可以使用自訂的 Windows 安全層級來執行 ASP.NET 應用程式。

Forms 驗證模式

當 ASP.NET 應用程式設定爲 Forms 驗證模式時，用戶端送來的 Request 還是必須先透過 IIS，所以 IIS 的目錄安全設定要設定爲允許匿名存取。Forms 驗證模式的驗證流程如下圖所示：

用戶端首先發出 Request 瀏覽被保護的 ASP.NET 網頁, 由於 IIS 的驗證設定爲允許匿名存取, 所以此時 IIS 將用戶端的 Request 傳遞給 ASP.NET 應用程式處理, 並未使用到 IIS 的驗證。 ASP.NET 應用程式收到用戶端的 Request 後, 先檢查是否有附加任何通過驗證的 Cookie；如果沒有任何通過驗證的 Cookie, 則重新導向到 Web.config 中所設定的登入網頁。使用者在登入網頁中出示合法的證件並通過驗證後, ASP.NET 會產生一個通過驗證的 Cookie 。

通過驗證的使用者要瀏覽受保護的網頁時, ASP.NET 會執行存取權限比對。網頁的存取權限於 Web.config 檔案中設定, ASP.NET 在進行權限比對時會將 Cookie 所記載的使用者名稱和 Web.config 內所設定的權限進行比對。若使用者擁有存取該網頁的權限, ASP.NET 將依程式的設計回到使用者原來 Request 的網頁或其他網頁；若使用者沒有該網頁的存取權限, 則顯示拒絕存取的訊息。

Forms 驗證模式的設定

要讓 ASP.NET 支援 Forms 驗證模式, 要設定「<authentication >」的 mode 屬性爲「Forms」並指定登入的網頁是哪一個, 「<authentication >」部分的設定語法如下所示：

```
<system.web>
  <authentication mode="Forms">
    <forms name="cookie名稱"
           loginUrl="登入網頁"
           timeout="cookie有效時間"/>
  </authentication>
</system.web>
```

其中「<authentication >」部分內多一個「<forms>」標籤, 用來設定 Forms 驗證模式的相關設定。其中「name」屬性用來指定存放通過驗證的 Cookie 名稱, 預設的名稱爲「.ASPXAUTH」。如果同一台 Web 伺服器執行多個 Web 應用程式, 而且每個 Web 應用程式都需要使用 Cookie 來存放驗證資訊, 此時就必需爲每個不同的 Web 應用程式在 Web.config 檔中指定不同的 Cookie 名稱。而「loginUrl」屬性則用來指明登入網頁的位址, 「timeout」屬性則用來設定記載驗證資訊的 Cookie 其有效期限, 預設值爲 30 分鐘。

使用範例

下列 Web.config 設定檔指定使用 Forms 驗證模式, 驗證後的資訊存放於名爲「.ASPXAUTH」的 Cookie 中, 登入的網頁爲「Login.aspx」, Cookie 超過 10 分鐘後即失效：

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Web.Config Configuration File-->
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms  name=".ASPXAUTH"
              loginUrl="Login.aspx"
              timeout="10"/>
    </authentication>
    <globalization requestEncoding="big5"
                   responseEncoding="big5"/>
  </system.web>
</configuration>
```

程式 Web.config

存取權限的設定

驗證模式設定完畢後, 可以使用「<authorization>」部分來設定使用者對應用程式存取權限, <authorization> 部分的設定語法如下所示：

```
<system.web>
  <authorization>
    <allow roles="使用者群組1,使用者群組2,使用者群組N"
           users="使用者帳號1,使用者帳號2,使用者帳號N"/>
    <deny roles="使用者群組1,使用者群組2,使用者群組N"
          users="使用者帳號1,使用者帳號2,使用者帳號N"/>
  </authorization>
</system.web>
```

users 屬性值若爲 *, 則表示所有使用者的意思；若爲 ？則爲匿名使用者。 roles 屬性則是用來指定 Windows 的使用者群組, 倘若使用 Forms 驗證模式則不需要指定。

下列 Web.config 設定檔片段指定 Web 應用程式不允許匿名存取的使用者：

```
<system.web>
  <authorization>
    <deny users="?"/>
  </authorization>
</system.web>
```

若只要指定 Web 應用程式只允許 Charlie 以及 John 存取時, 可以使用下列 Web.config 的片段設定：

```
<system.web>
  <authorization>
    <allow users="Charlie,John"/>
    <deny users="*"/>
  </authorization>
</system.web>
```

上述設定也包括了拒絕匿名使用者的存取, 注意 <allow> 標籤必需在 <deny> 標籤之上才會正確工作, 若 <deny> 標籤在 <allow> 標籤上, 則所有的使用者都會被拒絕存取。另外若把 * 改成 ?, 則只拒絕匿名存取的使用者存取, 包括 Charlie 以及 John 之外的使用者都可以存取。

設定特定 ASP.NET 網頁的權限

<location> 標籤可以指定特定區域的權限, 這個區域可以是單一 ASP.NET 網頁或整個目錄的 ASP.NET 網頁權限, 其使用語法如下所示：

```
<location  path="設定權限的位置">
<system.web>
  <authorization>
    <allow roles="使用者群組1,使用者群組2,使用者群組N"
           users="使用者帳號1,使用者帳號2,使用者帳號N"/>
    <deny roles="使用者群組1,使用者群組2,使用者群組N"
          users="使用者帳號1,使用者帳號2,使用者帳號N"/>
  </authorization>
</system.web>
</location>
```

使用範例

下列 Web.config 設定檔指定 John 子目錄以及 John.aspx 網頁的權限, 只有使用者 John 才可以存取：

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Web.Config Configuration File-->
<configuration>
  <location path="John.aspx">
    <system.web>
      <authorization>
        <allow users="John"/>
        <deny users="*"/>
      </authorization>
    </system.web>
  </location>
  <location path="John">
    <system.web>
      <authorization>
        <allow users="John"/>
        <deny users="*"/>
      </authorization>
    </system.web>
  </location>
  <system.web>
    <authentication mode="Forms">
    <forms name=".ASPXAUTH" loginUrl="Login.aspx"
           timeout="10"/>
    </authentication>
    <authorization>
      <allow users="Charlie"/>
      <deny users="*"/>
    </authorization>
    <globalization requestEncoding="big5"
                   responseEncoding="big5"/>
  </system.web>
</configuration>
```

程式 Web.config

<location>標籤只針對副檔名爲 .aspx 的 ASP.NET 網頁有效, 至於其它副檔名的檔案(如 .html 檔)則無效。

核發通過驗證的證明

使用者出示如帳號及密碼等合法證件後, 即可執行證件的查驗。驗證使用者身份的資料可以存放在資料庫、檔案, 或直接存放在 Web.config 檔案中(稍後將有介紹)。倘若使用者所出示的身份證件通過程式的驗證, ASP .NET 則必需依使用者的帳號產生一個證明通過驗證的 Cookie, 並將這個驗證身份的 Cookie 核發給使用者, 讓使用者在存取網頁的時候可以將這個 Cookie 當作通行證。

FormsAuthentication 類別

要核發證明通過驗證的 Cookie, 可以直接使用「FormsAuthentication」類別所提供「SetAuthCookie」或「RedirectFromLoginPage」的 Shared 方法來核發 Cookie。FormsAuthentication 類別支援許多用來執行驗證相關工作的方法, 如下表所示：

方法	說明
Authenticate	使用 Web.config 檔案中所存放的使用者驗證資料來執行驗證的工作
GetAuthCookie	使用給予的使用者帳號產生一證明通過驗證的 Cookie。本方法不會將 Cookie 發出, 所以應用程式可以在如何發出 Cookie 上有更多的控制
GetRedirectUrl	取得要重新導向的URL, 該 URL 是使用者原先所要求瀏覽的網頁
RedirectFromLoginPage	核發通過驗證的 Cookie, 並將通過驗證後的使用者重新導向至原先所要求瀏覽的網頁
SetAuthCookie	核發通過驗證的 Cookie, 不過不重新導向
SignOut	清除通過驗證的 Cookie

SetAuthCookie 與 RedirectFromLoginPage 方法

SetAuthCookie 方法會核發通過驗證的 Cookie, 而 RedirectFromLoginPage 方法除了核發 Cookie 外, 還會自動重新導向至使用者所要求瀏覽的網頁。由於 machine.config 設定檔預設已經加入了 FormsAuthentication 類別所屬的模組, 所以可以在 ASP.NET 網頁中直接叫用 FormsAuthentication 類別的所有 Shared 方法。 SetAuthCookie 與 RedirectFromLoginPage 兩個方法所接受的參數都一樣, 其使用語法如下所示：

```
RedirectFromLoginPage|SetAuthCookie(UserID As String,
                                    IsDurable As Boolean)

RedirectFromLoginPage|SetAuthCookie(UserID As String,
                                    IsDurable As Boolean
                                    CookieDir As String)
```

兩個方法的參數說明如下表所示：

參數	說明
UserID	通過驗證的使用者帳號
IsDurable	指明 Cookie 是否要持續存放於用戶端。若設定值爲 False, 則 Cookie 在使用者離線後便消失；若設定值爲 True, 則 Cookie 會存放在用戶端的電腦中, 下次瀏覽相同的 Web 應用程式時不需要重新驗證, 可直接使用
CookieDir	指明 Cookie 所存放的路徑, 預設爲「\」

使用範例

下列範例示範如何使用 Forms 驗證模式, 並核發使用者證明通過驗證的 Cookie, 接著將網址重新導向使用者所要求瀏覽的網頁：

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Web.Config Configuration File-->
<configuration>
  <system.web>
    <authentication mode="Forms">
```

```
      <forms name=".ASPXAUTH" loginUrl="Login.aspx"
             timeout="10"/>
    </authentication>
    <authorization>
      <allow users="Charlie"/>
      <deny users="*"/>
    </authorization>
    <globalization requestEncoding="big5"
                   responseEncoding="big5"/>
  </system.web>
</configuration>
```

程式 Web.config

上述設定指明整個 Web 應用程式只允許使用者 Charlie 存取, 其它使用者以及匿名使用者將無法瀏覽除登入網頁 Login.aspx 之外的網頁。

```
<form runat="Server">
  帳號: <Asp:TextBox id="txtUser" runat="Server"/><br>
  密碼: <Asp:TextBox id="txtPWD" TextMode="Password"
              runat="Server"/>
  <Asp:Button id="btnOK" text="確定" OnClick="btnOK_Click"
       runat="Server"/><p>
  <Asp:CheckBox id="chkSaveCookie" text="記住驗證 Cookie"
       runat="Server"/><br>
  <Asp:Label id="lblMSG" runat="Server"/>
</form>

<script language="vb" runat="Server">
Private Sub btnOK_Click(sender As Object, e As EventArgs)
  If Check(txtUser.Text, txtPWD.Text) Then
    FormsAuthentication.RedirectFromLoginPage(txtUser.Text, _
      chkSaveCookie.Checked)
  Else
    lblMSG.Text="驗證失敗, 請重新輸入驗證資訊!"
  End IF
End Sub

Private Function Check(ByVal UID As String, _
                       ByVal UPWD As String) As Boolean
  If UID = "Charlie" AndAlso UPWD="pass" Then
    Return True
  ElseIf UID = "John" AndAlso UPWD="pass" Then
```

```
    Return True
  Else
    Return False
  End If
End Function
</script>
```

程式 Login.aspx

上述範例宣告了一個驗證使用者的程序 Check，倘若使用者輸入的帳號及密碼正確則會傳回 True 反之則傳回 False，實際應用則可以連結至資料庫或透過建立在 Web.config 中的使用者資料進行身份驗證的工作。使用者所出示的證件驗證成功後，程式即呼叫 FormsAuthentication 類別的 RedirectFromLoginPage 方法，利用使用者的帳號產生一個通過驗證的 Cookie 。驗證 Cookie 可以依據使用者的選擇來決定要不要存放在用戶端，若使用者勾選「記住驗證 Cookie」選項，則通過驗證的 Cookie 在使用者離線後將留存在用戶端的電腦中，下次瀏覽相同 Web 應用程式時便可透過此一驗證 Cookie 直接查驗存取網頁的權限。

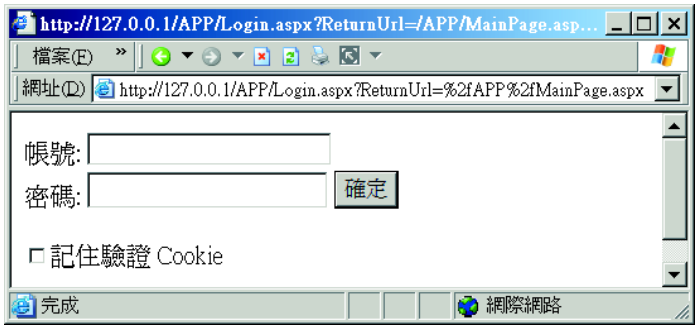
驗證 Cookie 核發完畢並決定是否留存後，接著即檢查使用者對原來所要求瀏覽的網頁是否擁有存取權限。倘若使用者擁有存取權限，則重新導向至使用者原始所要求瀏覽的網頁，反之則停留在登入網頁。若想要自訂驗證後的執行流程，則可以使用只發出驗證 Cookie，不作任何重新導向的工作的 SetAuthCookie 方法。下列 ASP.NET 網頁爲使用者欲瀏覽的網頁：

```
<form runat="Server">
  <Asp:Label id="lblMSG" runat="Server"/>
  <Asp:Button id="btnSignOut" text="登出"
      OnClick="btnSignOut_Click" runat="Server"/>
</form>

<script language="vb" runat="Server">
Private Sub Page_Load(sender As Object, e As EventArgs)
  Dim strAuthUser As String = User.Identity.Name
  Dim strAuthType As String = User.Identity.AuthenticationType
  Response.Write("登入使用者爲: " & strAuthUser)
  Response.Write("<br>驗證的模式爲: " & strAuthType)
  Response.Write("<br>是否通過驗證: " & _
```

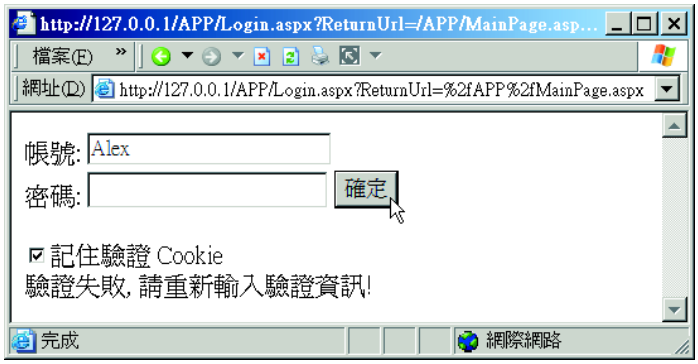
```
                User.Identity.IsAuthenticated)
End Sub

Private Sub btnSignOut_Click(sender As Object, e As EventArgs)
  FormsAuthentication.SignOut()
  lblMSG.Text="已經登出!"
  btnSignOut.Text="登入"
End Sub
</script>
```

程式 MainPage.aspx

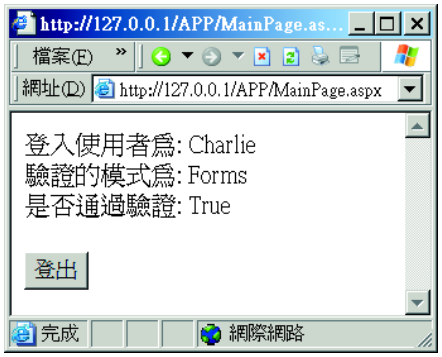
上述範例使用了實作 FormsIdentity 類別的 User 物件, 該物件可以取得使用者帳號、驗證模式以及是否通過驗證等資訊。若使用者沒有驗證 Cookie 便要直接存取網頁 Mainpage.aspx, ASP .NET 則會自動導向至登入網頁 Login.aspx：



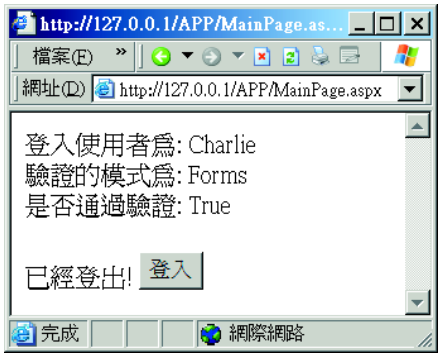
觀察上圖畫面中的網址列, 發現 ASP.NET 一併將所要求瀏覽的原始網頁製成參數 ReturnUrl, 以方便通過驗證後取得原來的網址。若輸入錯誤的使用者帳號或密碼, 則會停留在登入網頁並顯示錯誤訊息：



若輸入使用者 John 正確的帳號及密碼, 由於 John 沒有存取 Web 應用程式的權限, 所以會停留在登入網頁, 不會重新導向也不會顯示任何訊息。而擁有整個 Web 應用程式存取權限的使用者 Charlie 登入後, 才會自動重新導向原始要求的網頁 MainPage.aspx 並顯示使用者資訊：



若使用者取得驗證的 Cookie 後想要將 Cookie 取消, 則可以按下「登出」按鈕使用 FormsAuthentication 類別的 SignOut 方法, SignOut 方法可以立即清除通過驗證的 Cookie：



由於 Cookie 在按下登出按鈕後已經被清除, 此時若再按下顯示爲「登入」的同一個按鈕, 由於 ASP.NET 找不到通過驗證的 Cookie, 所以不需要撰寫任何程式即由 ASP.NET 自動重新導向到登入網頁。

使用 Web.config 檔存放驗證資料

如果你不想另建資料庫或檔案來存放使用者驗證資料, 則可以使用 Web.config 檔來存放使用者的驗證資訊, 並且使用 FormsAuthentication 類別的 Authenticate 方法來進行驗證的工作。要在 Web.config 檔案中存放使用者驗證資料, 必需要在 <forms> 標籤中加入 <credentials> 子標籤, <credentials> 子標籤的設定語法如下所示：

```
<system.web>
  <authentication mode="Forms">
    <forms name="Cookie 名稱" loginUrl="登入網頁"
      timeout="Cookie 有效時間">
      <credentials passwordFormat="密碼編碼格式">
        <user name="帳號1" password="密碼1"/>
        <user name="帳號2" password="密碼2"/>
        <user name="帳號N" password="密碼N"/>
      </credentials>
    </forms>
  </authentication>
</system.web>
```

<credentials> 區段爲存放使用者驗證資料的地方, passwordFormat 屬性則表示密碼的存放格式, 其設定值有 Clear 、 MD5 以及 SHA1 三種。屬性值爲 Clear 表示不進行編碼, MD5 則表示使用 Message Digest 5 演算法的編碼方式來儲存, 而 SHA1 則表示使用 SHA1 演算法的編碼方式來儲存, 這兩種演算法可以使用 FormsAuthentication 類別的 HashPasswordForStoringInConfigFile 方法自動執行編碼的工作。最後 <credentials> 區段內的 <user> 子標籤則是用來存放使用者的帳號及密碼。

使用範例

下列範例使用 Web.config 設定檔來存放使用者資訊, 並使用 FormsAuthentication 類別的 Authenticate 方法來進行驗證的工作：

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Web.Config Configuration File-->
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms name=".ASPXAUTH" loginUrl="LoginCFGFile.aspx"
             timeout="10">
        <credentials passwordFormat="Clear">
          <user name="Charlie" password="pass"/>
          <user name="John" password="pass"/>
        </credentials>
      </forms>
```

```
    </authentication>
    <authorization>
      <allow users="Charlie"/>
      <deny users="*"/>
    </authorization>
    <globalization requestEncoding="big5"
                   responseEncoding="big5"/>
  </system.web>
</configuration>
```

程式 Web.config

```
<form runat="Server">
  帳號: <Asp:TextBox id="txtUser" runat="Server"/><br>
  密碼: <Asp:TextBox id="txtPWD" TextMode="Password"
             runat="Server"/>
  <Asp:Button id="btnOK" text="確定" OnClick="btnOK_Click"
       runat="Server"/><p>
  <Asp:CheckBox id="chkSaveCookie" text="記住驗證 Cookie"
       runat="Server"/><br>
  <Asp:Label id="lblMSG" runat="Server"/>
</form>

<script language="vb" runat="Server">
Private Sub btnOK_Click(sender As Object, e As EventArgs)
  If FormsAuthentication.Authenticate(txtUser.Text, _
                                      txtPWD.Text) Then
    FormsAuthentication.RedirectFromLoginPage(txtUser.Text, _
                                     chkSaveCookie.Checked)
  Else
    lblMSG.Text="驗證失敗,  請重新輸入驗證資訊!"
  End IF
End Sub
</script>
```

程式 LoginCFGFile.aspx

由於使用 Forms 驗證可以簡化使用者驗證及授權所需要撰寫的程式，而且最後的部署不需要太多處理，所以建議使用 Forms 驗證模式。另外使用 Forms 驗證模式時所傳遞的資料並未予以加密保護，所以實際應用時應配合一些如 SSL 協定來保護敏感的資料。

資料傳遞的保密

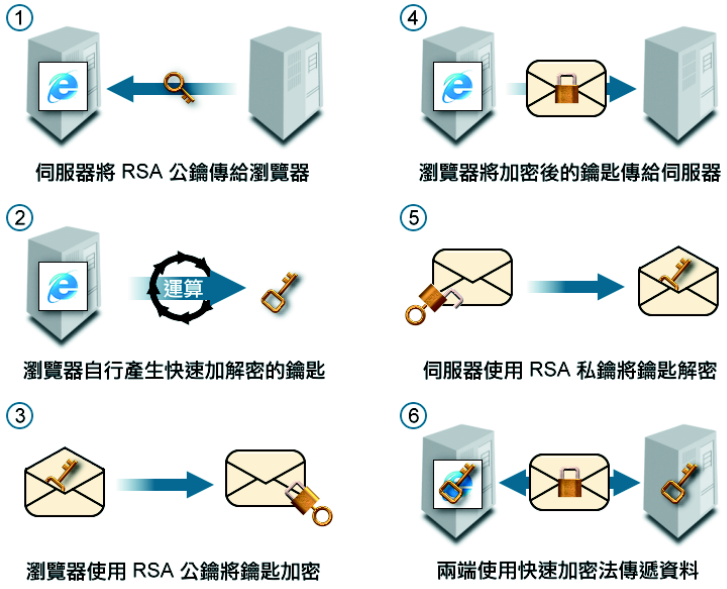
在網際網路傳送任何資料時，必需要注意安全性的問題。使用 HTTP 協定所傳送的任何資料，預設都是直接以未經編碼的文字資料(Clear Text)來傳送，這樣在傳遞一些如密碼、信用卡資料、個人資料等敏感資料時，就有可能讓有心人士有機會擷取所傳遞的資訊，進而造成安全上的漏洞。爲了避免這個問題，在傳送這些資料的時候應予以編碼加密，接收到資料後再予以解碼。

.NET Framework 支援許多種密碼方法，其中最被廣爲使用的方法爲 RSA，RSA 是由麻省理工學院的三位教授 Rivest 、 Shamir，及 Adelman 在 1977 年所提出的一種演算法。 RSA 使用「公 / 私鑰 」(Public/Private Key)來進行資料的編解碼工作，將訊息以公鑰加以編碼加密後傳送，接著透過相對應的私鑰才能將編碼後的資料解碼。由於編碼和解碼所使用的鑰匙不同，所以是一種非對稱的編碼方式。目前 RSA 較爲廣泛被使用的實作爲 SSL (Secure Sockets Layer)及 PGP (Pretty Good Privacy)，關於 RSA 的相關應用資訊，可以至 RSA 資訊安全公司網站 http://www.rsasecurity.com 查詢。

SSL

SSL 協定可以將資料進行編碼保密後再予以傳遞。使用 SSL 協定之前必需要先向「憑證機構 」(Certification Authority, CA)申請一個公鑰，憑證機構主要用來確認公鑰的實際所有單位。用戶端在使用 SSL 時並不需要擁有公鑰，而是當用戶端瀏覽使用 SSL 的網站時，該網站會提供經憑證管理機構驗證過的公鑰給用戶端瀏覽器，讓瀏覽器在傳送資料的時候可以使用公鑰來進行資料加密。不過由於使用 RSA 演算法需要較長的時間來進行資料的編解碼，無法滿足網際網路上用戶端和 Web 伺服端之間頻繁且即時的互動。

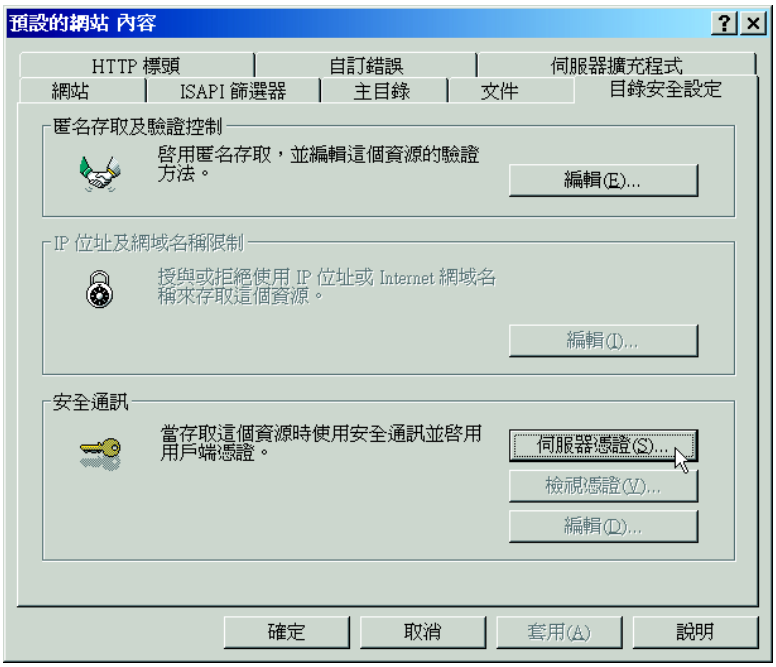
爲了讓網際網路上資料傳遞的速度不因執行 RSA 演算法而下降，SSL 在傳遞實際的資料時使用了較快速的 DES 、 RC2 或 RC4 等對稱式的快速演算法，然後再透過較安全的 RSA 演算法將快速演算法所要使用的鑰匙加密。對稱式的編解碼表示編碼與解碼所使用的鑰匙都一樣，相異於 RSA 演算法在編碼時使用公鑰、而解碼時使用私鑰的不對稱。瀏覽器收到 Web 伺服器的 RSA 公鑰後，會將自己所產生的鑰匙透過 RSA 公鑰加密，並且將加密後的鑰匙傳送給 Web 伺服器。接著 Web 伺服器再使用 RSA 私鑰執行鑰匙的解碼，這樣一來瀏覽器和 Web 伺服器都擁有一副相同的編解碼鑰匙，接下來資料傳遞的工作即可使用這副鑰匙進行較快速的編解碼工作。



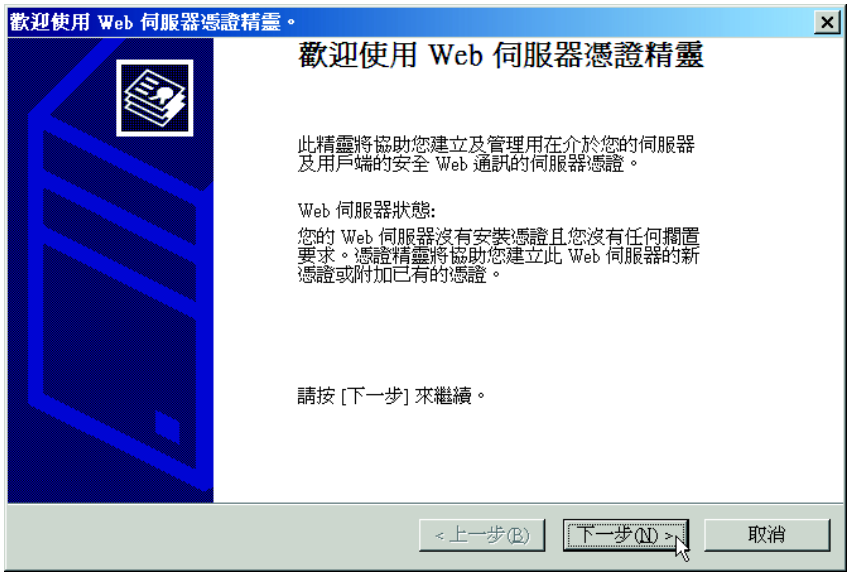
SSL 協定執行步驟

申請憑證

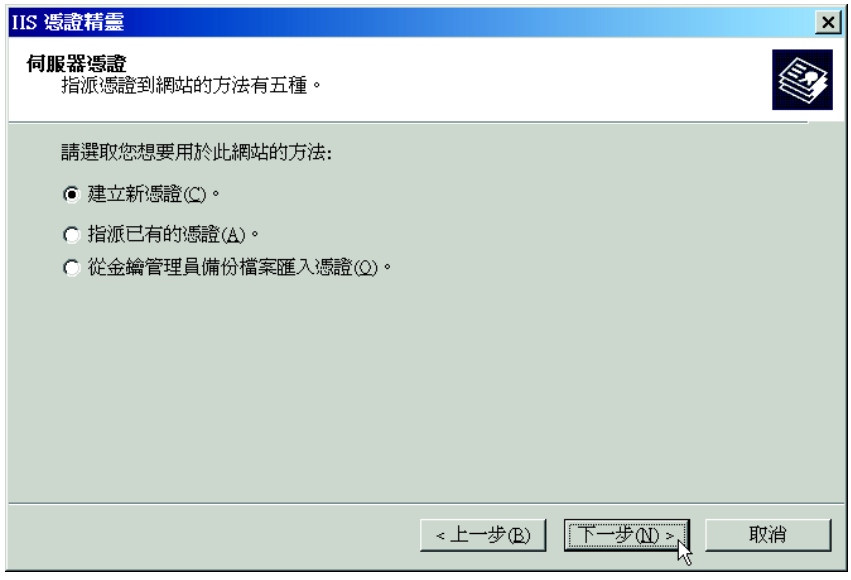
要使用 SSL 協定要先申請憑證，憑證內含有註冊組織的識別名稱、公開金鑰、公開金鑰有效期限、憑證機構的註冊識別名稱與電子簽章等資訊，用來確認擁有公開金鑰的組織。憑證要向如 Verisign(http://www.verisign.com) 與 Thawte (http://www.thawte.com) 等憑證機構申請，國內也有許多憑證機構接受憑證的申請。不過憑證並不是免費的，而且申請之後不能修改，所以在申請的時候必需小心填寫相關資料。要向憑證機構申請憑證，首先要先使用 Web 伺服器憑證精靈製作憑證需求檔，然後再使用憑證需求檔向憑證機構申請憑證。首先開啓 IIS 服務管理員，在「預設的網站」按滑鼠右鍵後選擇「內容」項目，並在「預設網站內容」對話盒出現後，切換到「目錄安全設定」頁籤：



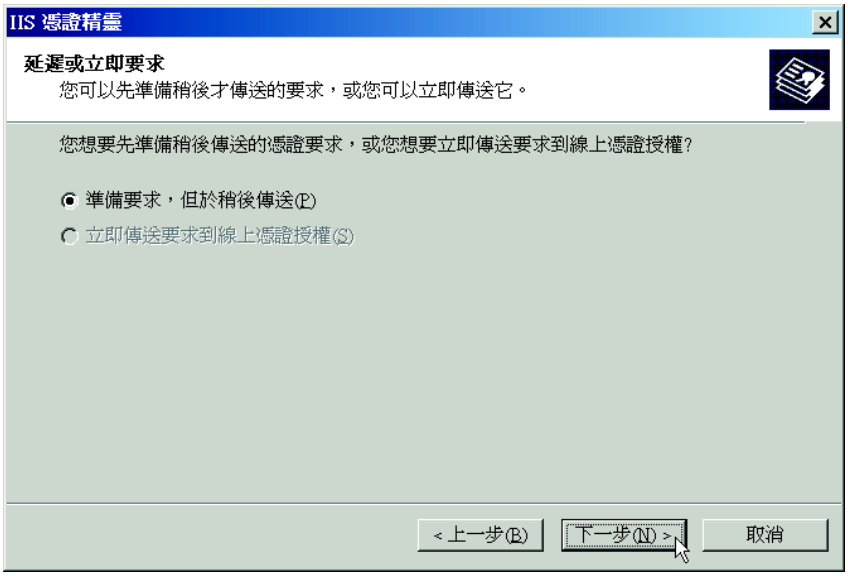
按下「伺服器憑證」項目後即出現 Web 伺服器憑證精靈：



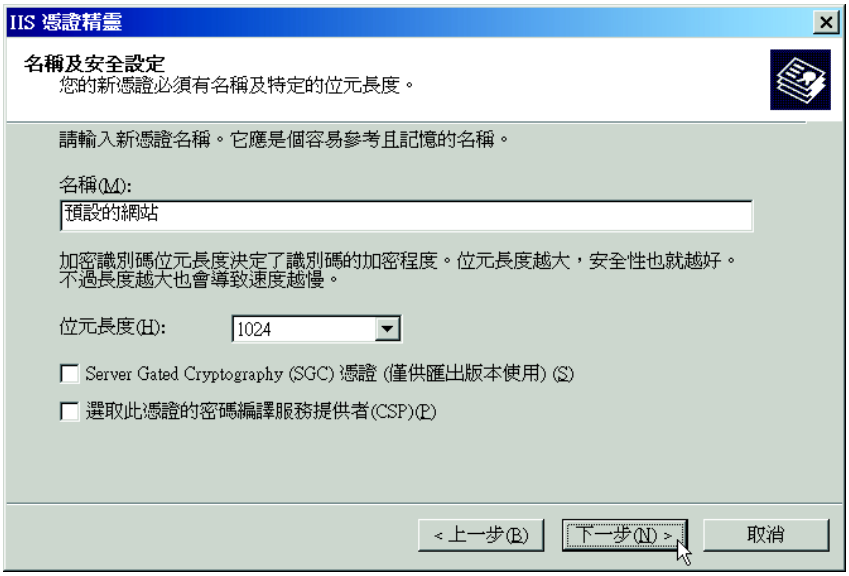
按下「下一步」按鈕後點選「建立新憑證」項目：



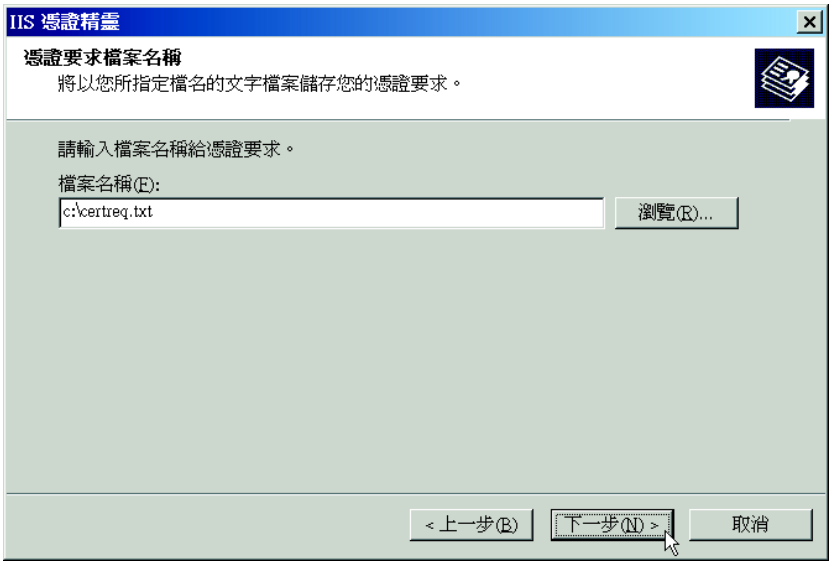
按下「下一步」按鈕後點選「準備要求，但於稍後傳送」項目：



按下「下一步」按鈕後，選擇加密識別碼的位元長度：

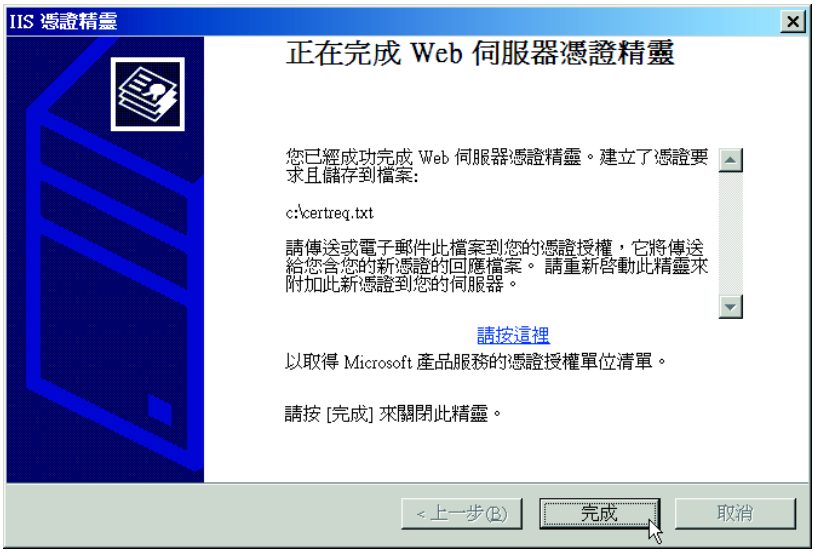


按下「下一步」按鈕後，輸入公司名稱、單位、地理資訊等相關訊息後，接著輸入憑證要求檔的輸出檔名及位置：

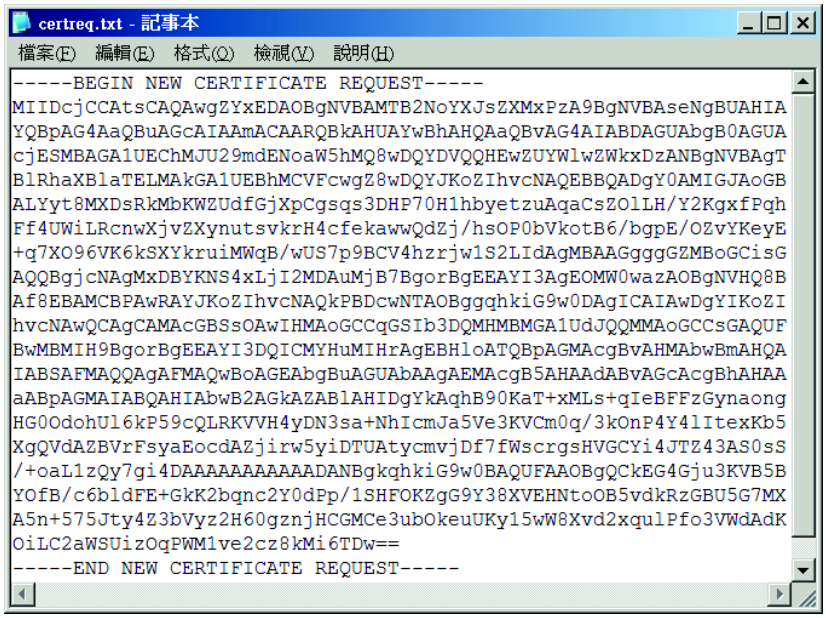


按下「下一步」按鈕後，會再列出一次所輸入的憑證資訊以求慎重。由於稍後要向 Verisign 憑證中心申請憑證，所以公司資訊必須全部使用英文填寫：

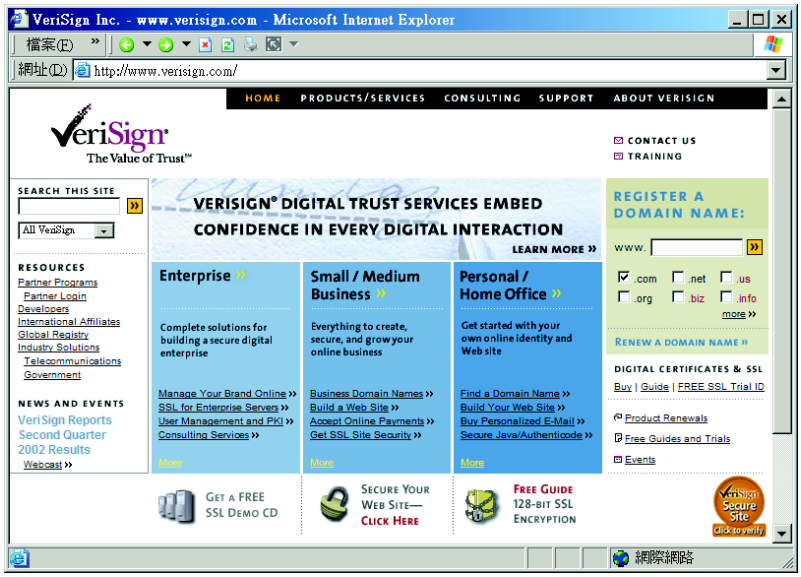
按下「下一步」按鈕後，即順利產生憑證要求檔：



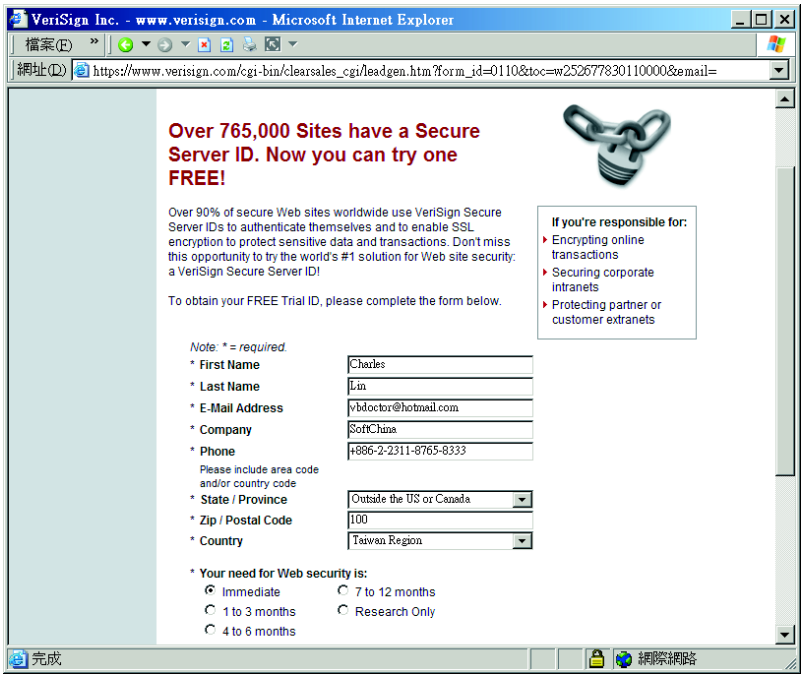
憑證要求檔將憑證要求的資訊編碼後以文字模式儲存：



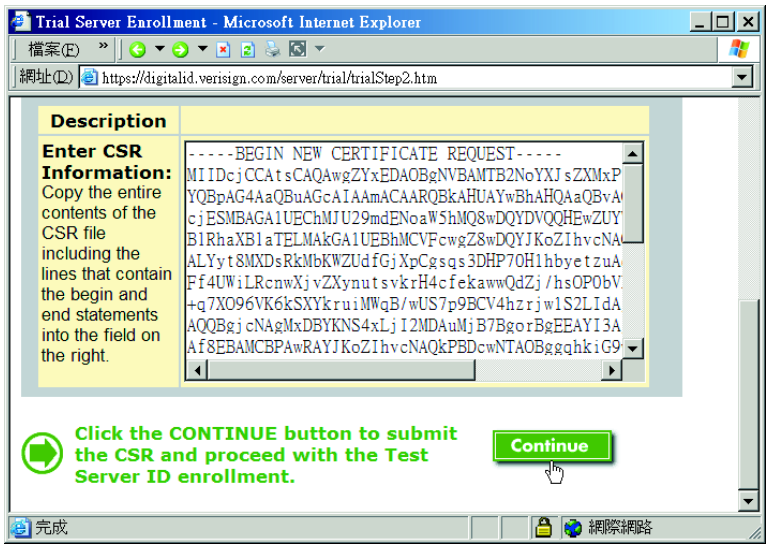
接著選擇一家憑證機構，本範例以Verisign(http://www.verisign.com)認證中心爲例：



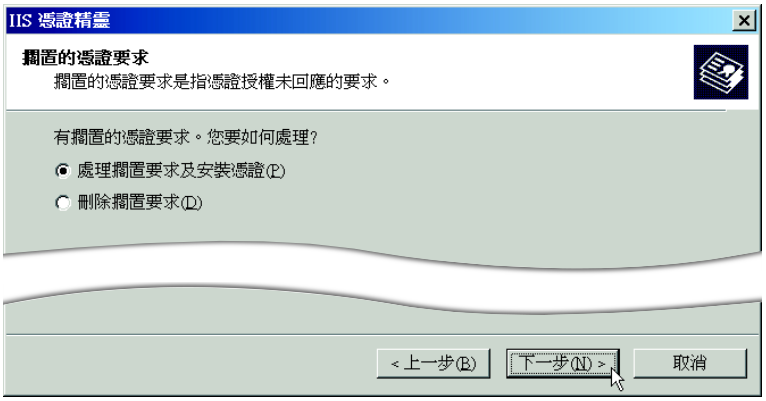
點選「Secure Your Web Site」項目後, 即可輸入申請憑證所需要的資料：



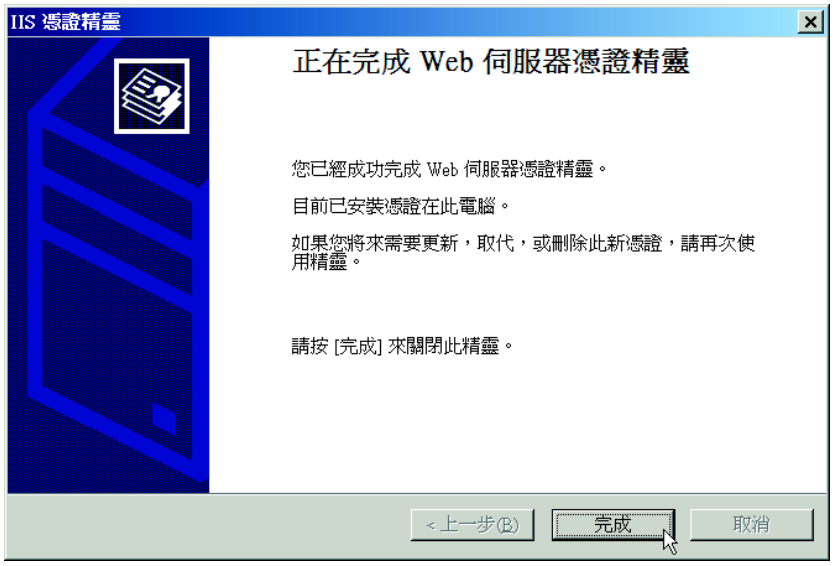
基本資料輸入完畢後，將會出現需要輸入 CSR 資訊的文字輸入盒，將剛才所產生憑證要求檔的內容複製後貼上，即可完成憑證的申請：



憑證申請完畢後，不需多久即可收到憑證中心所寄發的電子郵件，電子郵件中包含憑證中心所核發的憑證，憑證的內容和憑證需求檔很類似。將憑證的部分複製到文字檔中，並存成副檔名爲「.cer」的文字檔後，即可使用 Web 伺服器憑證精靈安裝憑證。憑證檔很重要，最好複製一份妥善保管。重新開啓 Web 伺服器憑證精靈後，所出現的畫面如下所示：



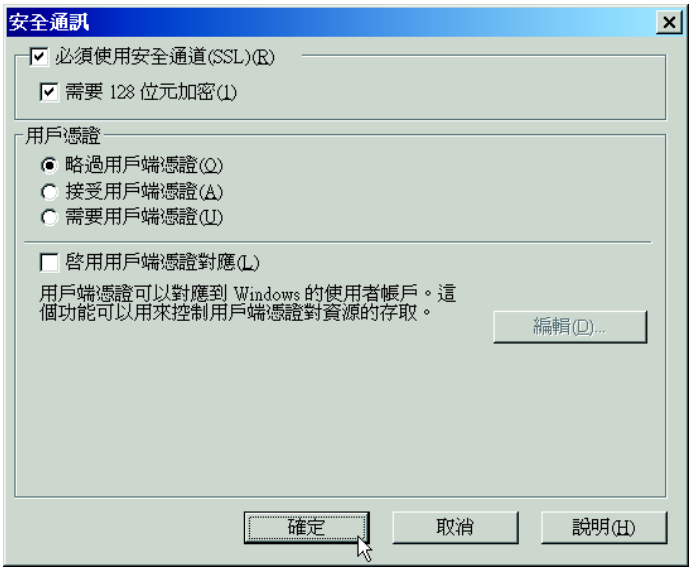
點選「處理擱置要求及安裝憑證」項目後，即可將所儲存的「.cer」憑證檔安裝到 Web 伺服器中：



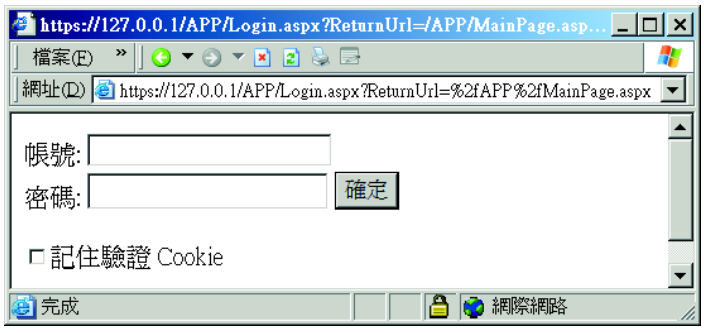
SSL 的設定

使用 IIS 服務管理員可以指定要使用 SSL 安全通訊的整個 Web 應用程式、特定目錄或 ASP.NET 網頁。本範例以之前所建立的 APP 應用程式爲例，開啓 APP 應用程式內容對話盒後，點選「目錄安全設定」頁籤：

在「安全通訊」群組中按下編輯「按鈕」，即出現「安全通訊」對話盒：



勾選「必需使用安全通道」以及「需要 128 位元加密」後，即可完成整個 APP 應用的安全通訊設定。由於使用了 SSL 協定，所以要將原來使用的 HTTP 協定改成 HTTPS，下列爲輸入網址「https://127.0.0.1/APP/MainPage.aspx」後的瀏覽器畫面：



由於使用 SSL 協定，所以瀏覽器的狀態列出現了黃色的鎖，表示資料的傳輸已受到保護。

Global.asax

每一個 ASP.NET 應用程式(包括 Web Services)都支援一個 Global.asax 檔, 提供整個應用程式發生某些事件或狀態改變時執行一些因應的程式碼。Global.asax 可以用來執行的工作有：

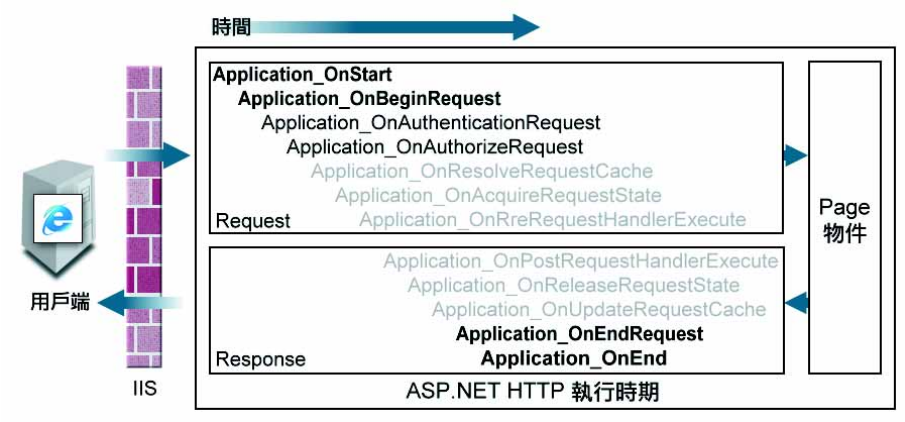
- 用來回應特定的 Application 物件與 Session 物件的事件。
- 使用 @Assembly 命令註册應用程式所會使用的組件。
- 使用 @Import 命令匯入命名空間, 這樣一來就可以不用在每個 Page 中逐一執行匯入的工作。
- 使用<object runat="Server">語法產生應用程式階層物件的實體。

這些事件的宣方法如下所示：

```
Public Sub 事件名稱(sender As Object, e As EventArgs)
  …程式碼略
End Sub
```

Global.asax 支援 18 個事件, 下表列出常用的的事件：

事件	說明
Application_OnAuthenticateRequest	當 Request 準備好要被驗證時引發
Application_OnAuthorizeRequest	當 Request 準備好要被授權時引發
Application_OnBeginRequest	當收到每一個 Request 時引發
Application_OnEndRequest	當每一個 Request 處理結束時引發
Application_OnEnd	當整個 Application 物件結束時引發
Application_OnError	當應用程式發生未處理的錯誤時引發
Application_OnStart	當整個 Application 物件啓始時引發
Session_OnEnd	當每個 Session 物件結束時引發
Session_OnStart	當每個 Session 物件啓始時引發



倘若沒有將目錄設定爲 Web 應用程式, 則 Global.asax 不會引發任何事件。

Application_OnBeginRequest 、Application_OnEndRequest 事件

每一次用戶端瀏覽網頁時就會引發 Application_OnBeginRequest 事件, 這個事件可以用來在每個網頁執行之前預先做些準備的動作；而每個網頁執行完畢後都會引發 Application_OnEndRequest 事件, 可以用來執行每一個網頁的清理作業或是爲每個網頁加上頁腳。

使用範例

下列範例在 Global.asax 檔案中使用 Application_OnBeginRequest 事件動態並隨機從 XML 檔案中載入歡迎詞, 並在 Application_OnEndRequest 事件中爲每一頁網頁加入頁腳：

- XML 檔案內容：

```
<?xml version="1.0" encoding="utf-8" ?>
<NewDataSet>
  <WebData>
    <Message>你好</Message>
  </WebData>
  <WebData>
    <Message>Hello</Message>
  </WebData>
  <WebData>
    <Message>Welcome</Message>
  </WebData>
  <WebData>
    <Message>歡迎</Message>
  </WebData>
</NewDataSet>
```

程式 檔案 XMLFile1.xml

- Global.asax檔案內容：

```
<script language="vb" runat="Server">
Public Sub Application_OnBeginRequest(sender As Object, _
                                      e As EventArgs)
  Dim xmlDoc As New System.Xml.XmlDocument()
  Dim intIdx As Integer
  xmlDoc.Load(Server.MapPath("XMLFile1.xml"))
  intIdx = Rnd() * _
    (xmlDoc.SelectNodes("/NewDataSet/WebData").Count - 1)

  Dim strA As String = xmlDoc.SelectNodes( _
              "/NewDataSet/WebData").Item(intIdx).InnerText
  Response.Write(strA & "<hr size=1>")
End Sub

Public Sub Application_OnEndRequest(sender As Object, _
                                    e As EventArgs)
   Response.Write("<hr size=1>Powered by ASP.NET")
End Sub
</script>
```

程式 Global.asax

上述兩個檔案皆儲存在目錄「C:\Inetpub\wwwroot\cr\ch12」中, 將影響該目錄中所有 ASP.NET 網頁的執行結果, 自動爲 ASP.NET 應用程式 ch12 的每個網頁加入頁首及頁腳, 下圖爲瀏覽網頁 Login.aspx 的變化：

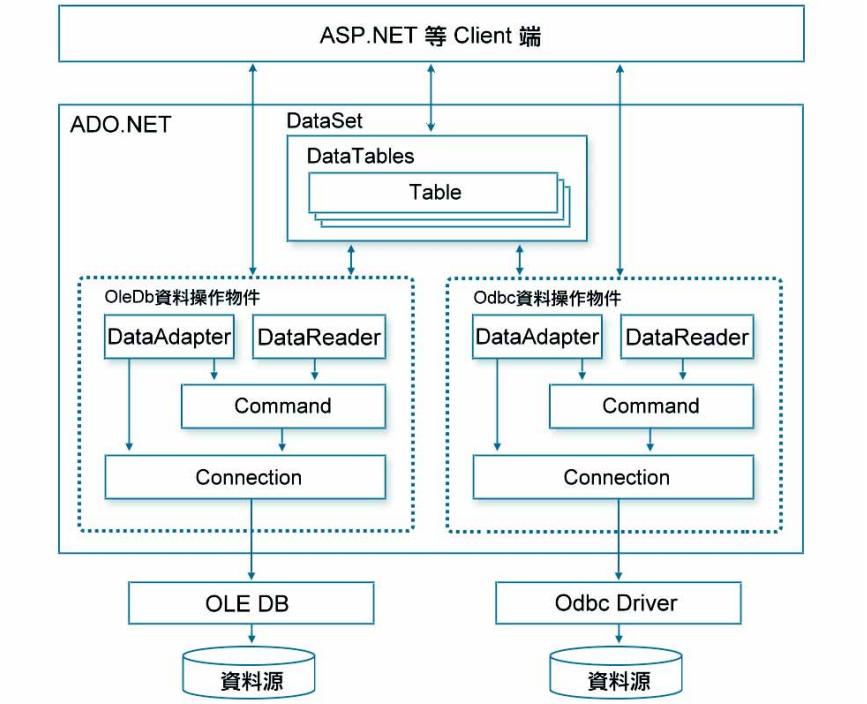
上述範例使用 Rnd() 函式隨機取得 0 至 XML 文件中 Message 元素數量減 1 的數值, 以決定所要顯示的歡迎詞。 System.XML.XMLDocument 類別提供了許多操作 XML 文件的能力, 例如 Load 方法可以讀取整個 XML 文件、SelectNodes 方法可以執行 XML 文件節點的操作, 而路徑「NewDataSet/WebData」中「Message」元素的數量, 可以使用「SelectNodes("XPath").Count」來取得。透過 SelectNodes 方法的 Items 屬性, 可以對 XML 元素執行特定的操作, 例如 InnerText 屬性可以傳回 XML 元素的內容。關於 XML 文件以及 XMLDocument 類別的詳細用法已經超出本書所要討論的範圍, 有興趣的讀者可以參閱線上文件的相關說明。

特定事件的引發時機

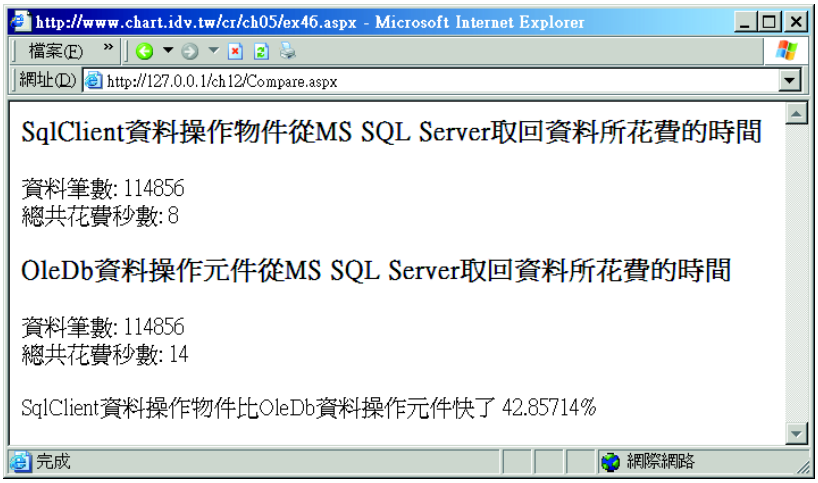
Application_OnStart 在 ASP.NET 應用程式第一次啓動時引發, 透過本事件可以用來執行一些準備執行 ASP.NET 應用程式前的工作, 例如將資料源中應用程式的共用資料取回、設定應用程式共用的變數及物件。而 Application_OnEnd 事件則和 Application_OnStart 事件相反, 在整個 ASP.NET 應用程式結束執行時才會引發, 透過 Application_OnEnd 事件可以執行如關閉資料源連結、清除或重設共用變數等工作。 ASP.NET 應用程式只要啓動後, 後續所接收的其他 Request 則將不再引發 Application_OnStart 事件, 直到整個 ASP.NET 應用程式因爲結束而引發 Application_OnEnd 事件後, 重新啓動 ASP.NET 應用程式才會再引發。 ASP.NET 應用程式啓動期間使用者瀏覽任何網頁, 並不會發生 Application_OnStart 以及 Application_OnEnd 事件。而 Session_OnStart 事件在每一個用戶端第一次 Request 網頁時會觸發一次, 爾後只要 Session 沒有失效即不再觸發 Session_OnStart 事件。 Session_OnStart 事件可以用來針對每個特定的使用者執行一些程式, 而在每個 Session 結束時則可以使用 Session_OnEnd 事件進行清除, 或是加入將 Session 儲存至資料庫的程式碼。

12-5 資料庫進階議題

想要和 MS SQL Server 互動, 要透過 SqlClient 資料操作物件。SqlClient 資料操作物件最主要是針對 MS SQL Server 來進行資料操作, 由於它直接呼叫 MS SQL Server 中的 API 而不透過 OleDb, 所以效能比較好。SqlClient 資料操作物件與 ODBC 資料操作物件的物件模型以及使用方法都和 OleDb 一樣, 如下圖所示：



爲了證明使用 SqlClient 資料操作物件和 MS SQL Server 連線比使用 OleDb 資料操作物件的效能好，我們實際寫了一個測試程式。這個程式從 MS SQL Server 中取回十一萬筆資料，並計算使用 SqlClient 資料操作物件比 OleDb 資料操作物件的效能快多少(測試程式爲 Compare.aspx)：



執行結果顯示 SqlClient 資料操作物件比 OleDb 資料操作物件的執行效能佳, 故針對 MS SQL Server 的資料操作當然使用 SqlClient 資料操作物件。

宣告SqlClient資料操作物件的命名空間

在使用 SqlClient 資料操作物件的時候, 除了要宣告 System.Data 的命名空間外, 還要宣告 System.Data.SqlClient 命名空間；如下程式碼片段所示：

```
<%@Import Namespace="System.Data"%>
<%@Import Namespace="System.Data.SqlClient"%>
```

宣告SqlClient資料操作物件

宣告 Connection 物件、Command 物件、DataAdapter 物件及 DataReader 物件時, 記得加上字首 Sql 。如下列範例所示：

```
Dim cnA As SqlConnection
Dim cmA As SqlCommand
Dim dscA As SqlDataAdapter
Dim drA As SqlDataReader
```

從 MS SQL Server 取回資料

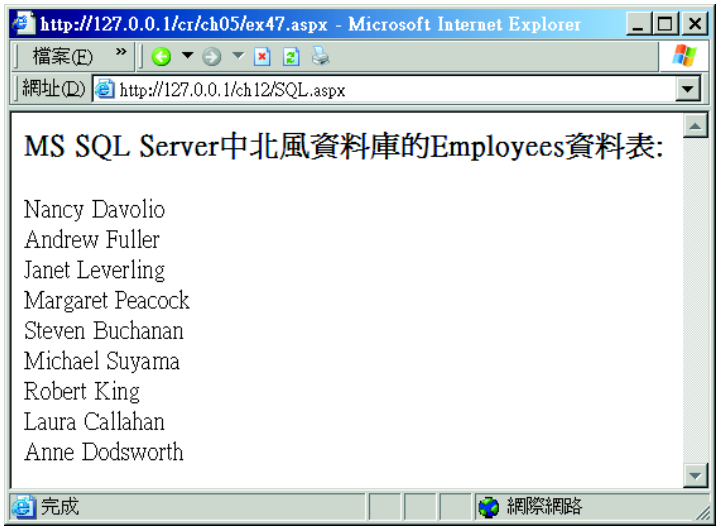
另外在建立 Connection 物件的時候, 由於已經知道要和 MS SQL Server 連線, 所以不需要指定 Connection 物件的 Provider 屬性。另外 DataTable 以及 DataSet 物件不是資料操作物件, 不負責執行資料源的資料操作, 故 DataTable 以及 DataSet 的宣告及使用方法不變。

使用範例

下列範例利用 SqlClient 資料控制元件將 MS SQL Server 中北風資料庫的 Employees 資料表取回：

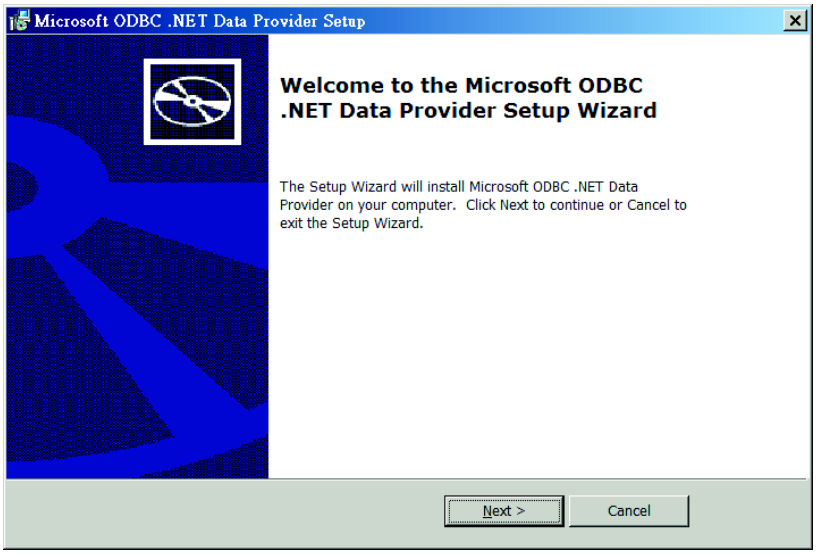
```
<%@Import Namespace=System.Data%>
<%@Import Namespace=System.Data.SqlClient%>

<script language="VB" runat="Server">
Private Sub Page_Load(sender As Object, e As EventArgs)
  Dim daA As New SqlDataAdapter("Select * From Employees", _
                "Data Source=Chart;" & _
                "Initial Catalog=Northwind;" & _
                "User Id=sa;Password=")
  Dim dtTable As New DataTable()
  daA.Fill(dtTable)

  Response.Write( _
    "<h3>MS SQL Server中北風資料庫的Employees資料表:</h3>")

  Dim shtR As Short
  For shtR=0 To dtTable.Rows.Count-1
    Response.Write( dtTable.Rows(shtR)("FirstName") & " " & _
                    dtTable.Rows(shtR)("LastName") & "<br>")
  Next
End Sub
</script>
```

程式 SQL.aspx

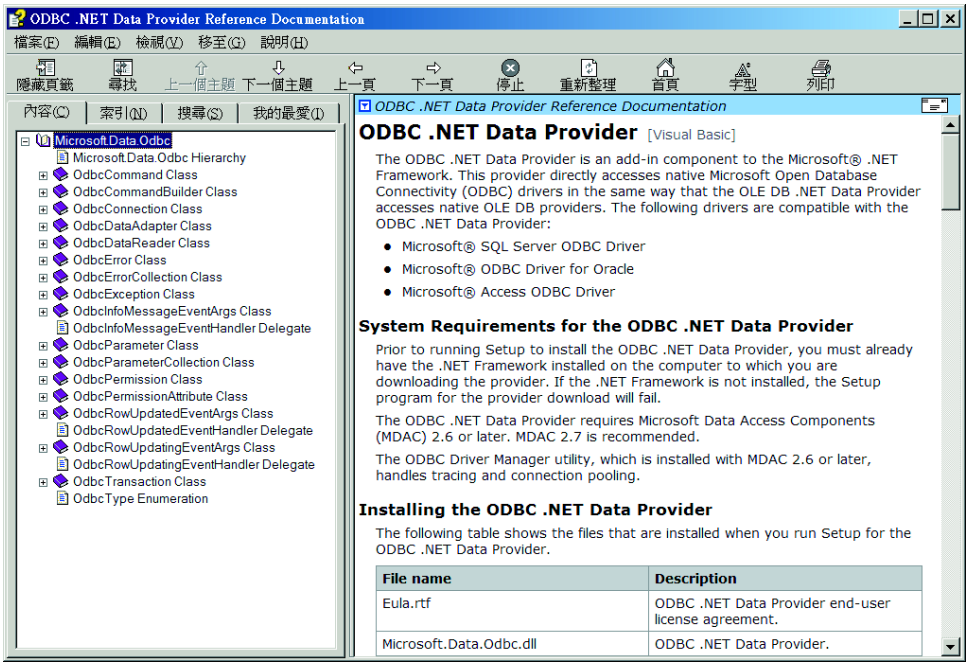


利用 ODBC 資料操作物件操作資料源

由於 ODBC 不是 ADO.NET 內建的資料操作物件, 所以若要透過 ODBC 和資料源互動, 則必須另外安裝 ODBC 資料操作物件才可以。 ODBC 資料操作物件可以在微軟MSDN網站http://msdn.microsoft.com/downloads/中的Software Development Kit 項目中找到, 下載安裝封包 odbc_net.msi 後, 直接點選即可安裝：



安裝完畢後, 在「開始」→「程式集」→「ODBC .NET Data Provider」項目內就可以找到「ODBC .NET Data Provider Documentation」, 點選後即可查詢 ODBC 資料操作物件的所有成員：

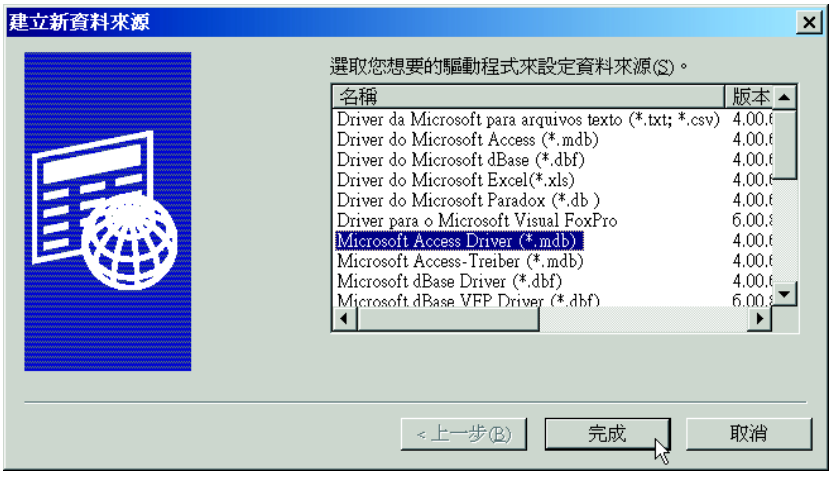


資料來源名稱的設定

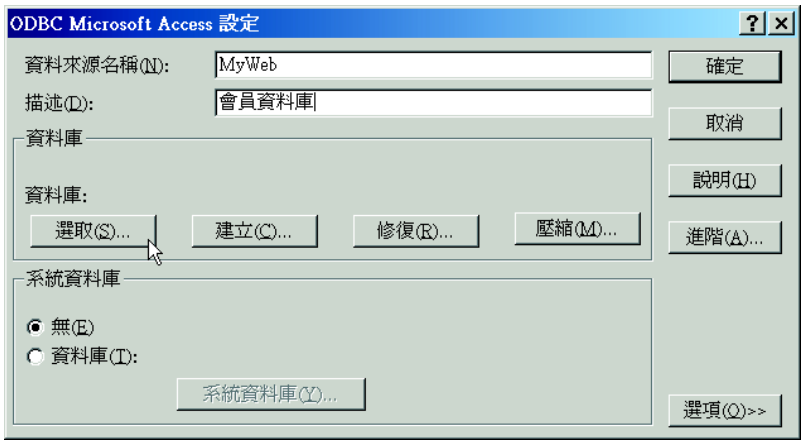
爲了使用上的便利, 可以先在系統上設定 ODBC「資料來源名稱」(DSN, Data Source Name)。首先開啓「開始」 → 「設定」 → 「控制台」, 並選擇「系統管理工具」後, 出現下列圖示：

開啓「資料來源(ODBC)」項目，並點選「系統資料來源名稱」：

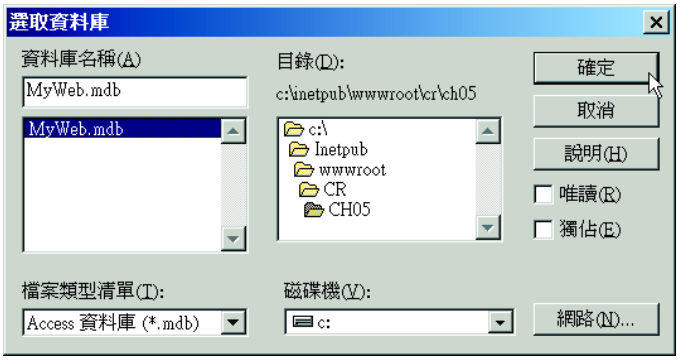
點選「新增」按鈕，並選擇所要連結的資料源種類：



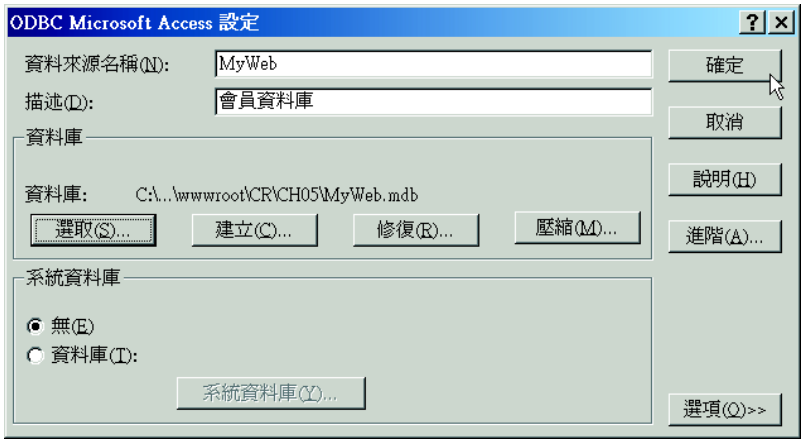
這裡我們以透過 ODBC 連結至「C:\InetPub\wwwroot\cr\ch05\MyWeb.mdb」爲例, 選擇「Microsoft Access Driver (*.mdb)」點選完成後, 出現下列視窗：



輸入資料來源名稱以及描述後，點選「選取」按鈕選擇資料庫檔案所在位址：



選擇完畢後，按下確定即回到上個畫面：



再按下確定後，即看到成功的新增了「MyWeb」這個系統資料來源：

使用範例

下列範例透過 ODBC 資料操作物件, 將資料來源名稱爲 MyWeb 的 Members 資料表全部列出：

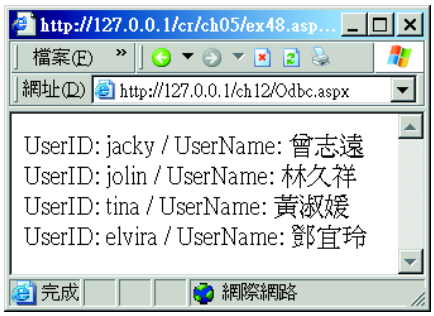
```
<%@Import Namespace="System.Data"%>
<%@Import Namespace="Microsoft.Data.Odbc"%>

<script language="VB" runat="Server">
Private Sub Page_Load(sender As Object, e As EventArgs)
  Dim cnA As New OdbcConnection("DSN=MyWeb")
  Dim cmA As New OdbcCommand( _
                "Select UserId,UserName From Members",cnA)
  cnA.Open()
  Dim drA As OdbcDataReader = _
         cmA.ExecuteReader(CommandBehavior.CloseConnection)

  Do while drA.Read()
   Response.Write("UserID: " & drA.Item("UserId"))
   Response.Write(" / UserName: " & _
                 drA.Item("UserName") & "<br>")
  Loop

  drA.Close()
End Sub
</script>
```

程式 Odbc.aspx

程式在編譯時, 預設會到應用程式目錄下的「\bin」目錄中找尋所指定的組件。本範例沒有將 Microsoft.Data.Odbc.dll 布署到 GAC 的狀況下, 會到 Ch12 的應用程式目錄下的 bin 子目錄尋找「Microsoft.Data.Odbc.dll」這個組件, 故必需將位於「C:\Program Files\Microsoft.NET\Odbc.Net」目錄下的「Microsoft.Data.Odbc.dll」這個組件複製到應用程式下的 bin 子目錄下才可以使用。 ODBC 資料操作物件的命名空間爲「Microsoft.Data.Odbc」, 所以要宣告下列的命名空間：

```
<%@Import Namespace="Microsoft.Data.Odbc"%>
```

故執行結果如下圖所示：



利用程式指定資料來源

使用 ODBC 資料操作元件也可以利用 ConnectionString 來指定資料來源，而不用預先建立好資料來源名稱。下表列出 OdbcConnection 物件的 ConnectionString 所接受的參數：

參數	說明
Driver	ODBC驅動程式名稱, 如： Access 設定值爲{ Microsoft Access Driver (*.mdb)} Excel 設定值爲{ Microsoft Excel Driver (*.xls)} MS SQL Server 設定值爲{SQL Server} Oracle 設定值爲{Microsoft ODBC for Oracle} 文字檔設定值爲{ Microsoft Text Driver (*.txt; *.csv)}
Server	資料庫伺服器名稱
UID	使用者帳號
PWD	使用者密碼
Database	欲使用之資料庫名稱
DBQ	資料庫檔案位址
DSN	資料來源名稱

其中可能的 ConnectionString 設定值如下所示：

- SQL Server:

  Driver={SQL Server};Server=伺服器名稱;UID=帳號;PWD=密碼;Database=Northwind

- Oracle:

  Driver={Microsoft ODBC for Oracle};Server=伺服器名稱;UID=帳號;PWD=密碼;DataBase=Northwind

- Access:

  Driver={Microsoft Access Driver (*.mdb)};DBQ=c:\InetPub\wwwroot\cr\ch05\MyWeb.mdb

- Excel:

  Driver={Microsoft Excel Driver (*.xls)};DBQ=c:\My Documents\book1.xls

- 文字檔:

  Driver={Microsoft Text Driver (*.txt; *.csv)};DBQ=c:\MyDocuments

- DSN:

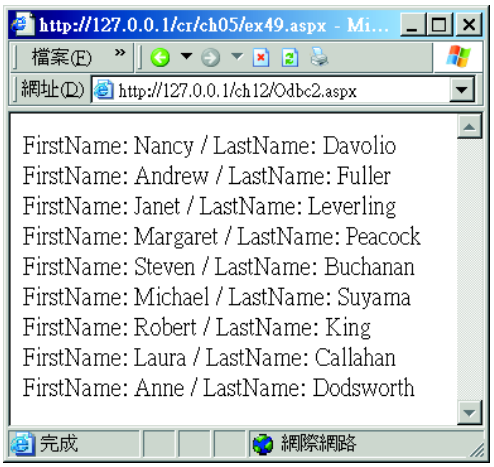
  DSN= 資料來源名稱

使用範例

下列範例透過 ODBC 資料操作物件使用 ConnectionString 和名爲 Chart 的 MS SQL Server 連線, 並取回 Northwind 資料庫的 Employees 資料表中的 FirstName 欄位以及 LastName 欄位：

```
<%@Import Namespace="System.Data"%>
<%@Import Namespace="Microsoft.Data.Odbc"%>

<script language="VB" runat="Server">
Private Sub Page_Load(sender As Object, e As EventArgs)
  Dim daA As New OdbcDataAdapter( _
             "Select FirstName,LastName From Employees", _
             "Driver={SQL Server};Server=Chart;" & _
             "Database=Northwind;UID=sa;PWD=")
  Dim dtTable As New DataTable()
  daA.Fill(dtTable)
  Dim intI As Integer

  For intI=0 To dtTable.Rows.Count-1
    Response.Write("FirstName: " & _
                   dtTable.Rows(intI).Item("FirstName"))
    Response.Write(" / LastName: " & _
              dtTable.Rows(intI).Item("LastName") & "<br>")
  Next
End Sub
</script>
```

程式 Odbc2.aspx



交易物件

在對資料源執行多個 SQL 敘述時若要確保每個敘述都確實執行成功，而不會因爲其中某個敘述失敗而造成不可回復的錯誤，可以使用「交易」(Transaction)。以使用提款機轉帳爲例，假設某人要從 A 帳戶轉 100 元至 B 帳戶，要先從 A 帳戶扣除 100 元後再將 100 元轉至 B 帳戶。若在執行此一動作時網路發生問題，那麼可能發生 A 帳戶的錢已經扣除了，可是錢卻沒有轉到 B 帳戶的狀況。要確保所有的工作都順利執行完成，若有失敗則將資料庫回復原狀，則可以使用交易物件。交易物件必須配合資料操作物件使用，就是說使用 SqlClient 資料操作物件必須配合對應的 SqlTransaction 物件。下表爲 Transaction 物件常用的屬性及方法：

成員	說明
Connection 屬性	傳回交易所使用的Connection物件，若交易失效則傳回Nothing
IsolationLevel 屬性	指定交易所使用的隔離層級
Commit 方法	確認交易，確認交易過程中所執行的所有SQL指令
RollBack 方法	取消交易，將資料庫回復至交易進行前的狀態
Save方法	儲存階段性的交易，可以用來回復部分交易

交易物件是由 Connection 物件的 BeginTransaction 方法所產生，呼叫後除了產生交易物件外之也表示交易開始進行。

使用範例

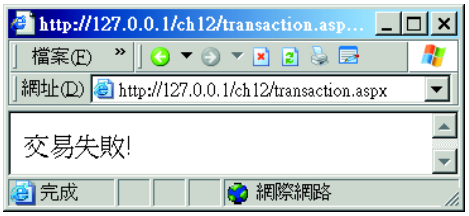
下列範例使用交易新增兩筆會員資料，其中只要有任何一筆新增失敗，資料庫將回復至交易前的狀態：

```
<%@Import Namespace=System.Data%>
<%@Import Namespace=System.Data.OleDb%>

<script language="VB" runat="Server">
Private Sub Page_Load(sender As Object, e As EventArgs)
  Dim cnA As New OleDbConnection( _
    "Provider=Microsoft.Jet.OleDb.4.0;" & _
```

```
    "Data Source=C:\Inetpub\wwwroot\cr\ch05\MyWeb.mdb")
  cnA.Open()
  Dim cmA As New OleDbCommand("",cnA)

  '交易開始, 並使用變數 tsTrans 接收交易物件的參照
  Dim tsTrans As OleDbTransaction = cnA.BeginTransaction()

  cmA.Transaction = tsTrans
  cmA.CommandText = "Insert into Members " & _
    "Values('john','1234','黃約翰','0987654321'," & _
    "'台北縣中和市','johnhuang@hotmail.com')"
  Try
    cmA.ExecuteNonQuery()
    cmA.CommandText = "Insert into Members " & _
      "Values('john','5678','李約翰','0912777222'," & _
      "'台北縣永和市','johnlee@hotmail.com')"
    cmA.ExecuteNonQuery()
    tsTrans.Commit()
    Response.Write("交易成功!")
  Catch ex As Exception
    tsTrans.RollBack()
    Response.Write("交易失敗!")
  Finally
    cnA.Close()
  End Try
End Sub
</script>
```

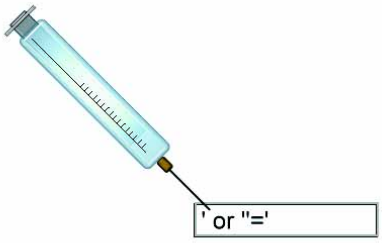
程式 Transaction.aspx

上述範例故意在交易中新增兩筆 UserID 欄位都是 john 的使用者。由於 UserID 欄位爲主鍵故執行第二個 SQL 敘述時即發生例外, 即由交易物件呼叫 RollBack 方法取消交易並將資料庫回復至交易進行前的狀態, 所以這兩筆資料永遠都不會被加入資料庫中。



資料隱碼的安全問題

所謂「資料隱碼」(SQL Injection)爲使用者在輸入資料時，在文字輸入盒輸入一些可以造成網頁安全漏洞或破壞資料庫的 SQL 敘述。資料隱碼的問題在每一個支援 SQL 敘述的資料庫都有，這個問題不是資料庫本身的安全漏洞，而是程式開發人員利用使用者輸入的資料和資料庫互動時無意所產生的漏洞。



輸入資料時植入 SQL 敘述

假設我們有一個使用者登入的網頁，分別使用 txtUserID 以及 txtUserPWD 兩個文字輸入盒讓使用者輸入帳號及密碼，並且使用任何支援 SQL 敘述的資料庫儲存使用者驗證資料(本範例以由 Access 資料庫所建立的 MyWeb.mdb 會員資料庫爲例)：

```
<%@Import NameSpace="System.Data"%>
<%@Import NameSpace="System.Data.OleDb"%>

<form runat="Server">
  帳號: <Asp:TextBox id="txtUserID" runat="Server"/><br>
  密碼: <Asp:TextBox id="txtUserPWD"
            TextMode="Password" runat="Server"/>
  <Asp:Button id="btnOK" text="確定" OnClick="btnOK_Click"
       runat="Server"/><p>
  <Asp:Label id="lblMSG" runat="Server"/>
</form>

<script language="vb" runat="Server">
Private Sub btnOK_Click(sender As Object, e As EventArgs)
  Dim cnA As New OleDbConnection( _
```

```
    "Provider=Microsoft.Jet.OleDb.4.0;" & _
    "Data Source=C:\Inetpub\wwwroot\cr\ch05\MyWeb.mdb")
  cnA.Open()
  Dim strSQL As String = _
    "Select UserName From Members " & _
    "Where UserId = '" & txtUserID.Text & _
    "' And UserPWD = '" & txtUserPWD.Text & "'"

  Dim cmA As New OleDbCommand(strSQL, cnA)
  Dim drA As OleDbDataReader = cmA.ExecuteReader()
  If drA.Read() Then  '如果有讀到資料
    lblMsg.Text = drA.Item(0) & " 您好"
  Else
    lblMsg.Text = "驗證失敗"
  End IF
  drA.Close()
  cnA.Close()
End Sub
</script>
```

程式 SQLInjection.aspx

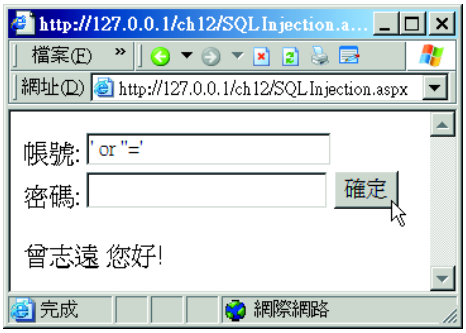
注意這下列 SQL 敘述：

```
  Dim strSQL As String = _
    "Select UserName From Members " & _
    "Where UserId = '" & txtUserID.Text & _
    "' And UserPWD = '" & txtUserPWD.Text & "'"
```

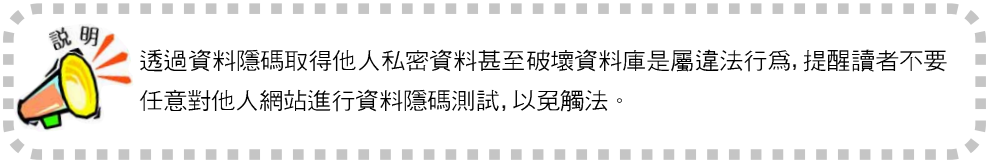
如果此時使用者在這兩個文字輸入盒中輸入「' or "='」, 則 SQL 敘述代入使用者所輸入的資料後則變成：

```
Select UserName From Members
Where UserId = '' or ''='' And UserPWD = '' or '' = ''
```

「User = "」敘述由於沒有該使用者故爲 False, 不過因爲「"="」是恆成立的, 所以接著使用「or "="」會讓整個條件判斷的結果爲 True, 這樣一來 Where 子句即喪失過濾資料的能力, 這個敘述可以讓任何人通過身份驗證：



使用者也可以透過資料隱碼下達刪除資料庫等危害資料庫安全及完整性的其他 SQL 敘述，所以資料隱碼是一個非常重要的安全性議題，在撰寫這方面的網頁時一定要避免。



資料隱碼等安全漏洞的避免

為了有效避免資料隱碼等造成安全漏洞的問題，下列有幾個注意事項：

- 移去使用者所輸入的單引號「'」、分號「;」以及空白。
- 不要使用資料庫的最高權限(以 MS SQL Server 來說是 sa)來連結資料庫，應該針對每個資料庫使用適當的權限。
- 使用預存程序來執行資料庫操作稍後將有詳細說明。
- 由於範例程式及範例資料庫並沒有考慮周詳，所以在實際的伺服器上應予以移除(如 SQL Server 中的 Northwind 以及 Pubs 範例資料庫)。
- 隨時注意是否有新的修補程式(Service Pack)。

參數物件

「參數」(Parameter)物件最主要可以配合「預存程序」(Stored Procedure), 或是搭配 SQL 敘述使用, 以簡化寫程式的繁瑣步驟並可避免資料隱碼問題。以下爲參數物件的建構子：

```
Public Sub New()

Public Sub New(參數名稱As String, _
               參數值 As Object)

Public Sub New(參數名稱 As String, _
               資料型別 As OleDbType)

Public Sub New(參數名稱 As String, _
               資料型別 As OleDbType, _
               參數大小 As Integer)

Public Sub New(參數名稱 As String, _
               資料型別 As OleDbType, _
               參數大小 As Integer, _
               來源欄位 As String)

Public Sub New(參數名稱 As String, _
               資料型別 As OleDbType, _
               參數大小 As Integer, _
               參數方向 As ParameterDirection, _
               是否接受Null值 As Boolean, _
               所要保留的數值位數 As Byte, _
               所要保留的小數位數 As Byte, _
               來源欄位 As String, _
               資料來源版本 As DataRowVersion, _
               參數值 As Object)
```

下表爲 OleDbParameter 物件常用的屬性：

屬性	說明
DbType	傳回或設定參數的 DbType
Direction	傳回或設定參數的方向, 其設定值有 Input、InputOutput、Output 以及 ReturnValue 四種, 預設為 Input
IsNullable	傳回或設定參數是否接受 Null 值
OleDbType	傳回或設定參數的 OleDbType
ParameterName	傳回或設定參數名稱
Precision	傳回或設定 Value 屬性的數字其最大數目
Scale	傳回或設定 Value 屬性的小數位數
Size	以 Byte 為單位傳回或設定資料的大小
SourceColumn	傳回或設定參數對應至 DataTable 中的欄位名稱
SourceVersion	傳回或設定參數在取得 Value 時所要使用的 DataRowVersion
Value	傳回或設定參數的值

其中 .NET Framework 共通型別、DbType、OleDbType 以及 SQLDbType 的對照如下表所示：

.NET Framewok 共通型別	OleDbType	SqlDbType	DbType
bool	Boolean	Bit	Boolean
byte	UnasignedTinyInt	TinyInt	Byte
byte[]	VarBinary	VarBinary	Binary
char	Char	不支援	不支援
DateTime	DBTimeStamp	DateTime	DateTime
Decimal	Decimal	Decimal	Decimal
double	Double	Float	Double
float	Single	Real	Single
Guid	Guid	UniqueIdentifier	Guid
int Integer	Int	不支援	
Int16	SmallInt	SmallInt	Int16
Int32	Int	Int	Int32
Int64	BigInt	BigInt	Int64
long	BigInt	BigInt	不支援
object	Variant	Variant	Object
short	SmallInt	SmallInt	不支援

string	VarWChar	NVarChar	String
UInt16	Unsigned Small Int	不支援	UInt16
UInt32	UnsignedInt	不支援	UInt32
UInt64	UnsignedBogInt	不支援	UInt64
不支援	VarChar	VarChar	AnsiStrubg
不支援	Currency	Money	Currency
不支援	DBDate	DateTime	Date
不支援	TinyInt	TinyInt	SByte
不支援	DBTime	DateTime	Time
不支援	VarNumeric	不支援	VarNumeric

參數物件可以將參數的內容自動代入 SQL 敘述中, 只要在 SQL 敘述中安插問號「?」或使用相對應的參數名稱即可。

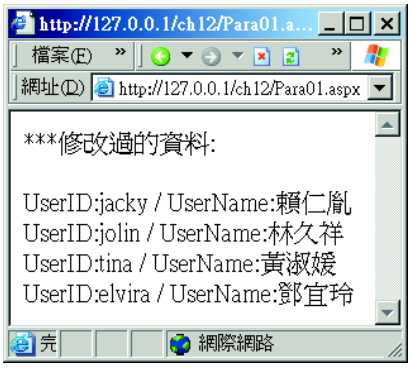
使用範例

下列範例明確產生 UpdateCommand, 並使用 OleDbParameter 物件搭配 SQL 敘述產生更新回資料源的敘述：

```
<%@Import Namespace=System.Data%>
<%@Import Namespace=System.Data.OleDb%>

<script language="VB" runat="Server">
Private Sub Page_Load(sender As Object, e As EventArgs)
  Dim cnA As New OleDbConnection( _
    "Provider=Microsoft.Jet.OleDb.4.0;" & _
    "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb")
  Dim daA As New OleDbDataAdapter( _
    "Select UserId,UserName From Members", cnA)

  Dim cmUpdate As New OleDbCommand( _
    "Update Members Set UserName=? Where UserID=?",cnA)

  Dim prmA As New OleDbParameter( _
    "UserName",OleDbType.Char, 10, "UserName")
  cmUpdate.Parameters.Add(prmA)   '將參數加入參數集合中

  prmA=New OleDbParameter( _
```

```
    "UserID",OleDbType.Char, 10, "UserID")
  cmUpdate.Parameters.Add(prmA)   '將參數加入參數集合中

  daA.UpdateCommand = cmUpdate

  Dim dtTable As New DataTable()
  daA.Fill(dtTable)   '  將使用者資料填入DataTable物件中

  dtTable.Rows(0)("UserName") = "賴仁胤"
  daA.Update(dtTable)

  dtTable.Clear()     '  清除DataTable
  daA.Fill(dtTable)   '  將使用者資料填入DataTable物件中

  Dim tmpRow As DataRow
  Response.Write("***修改過的資料:<p>")

  For Each tmpRow In dtTable.Rows
    Response.Write( "UserID:" & tmpRow.Item("UserID") & " / ")
    Response.Write("UserName:" & _
                   tmpRow.Item("UserName") & "<br>")
  Next
End Sub
</script>
```

程式 Para01.aspx



上述範例指定了資料更新敘述「"Update Members Set UserName=? Where UserId=?"」，其中兩個問號表示將由 Parameter 物件的值代入，產生 Parameter 物件時，使用下列敘述：

```
Dim prmA As New OleDbParameter( _
  "UserName",OleDbType.Char,10,"UserName")
```

指定了參數名稱爲「UserName」、資料型別爲「OleDbType.Char」、資料的長度爲 10, 所對應的欄位爲「UserName」, 接著將此參數加入 UpdateCommand 物件的 Parameters 集合中管理：

```
cmUpdate.Parameters.Add(prmA)
```

接著實作另外一個新的 Parameter, 並指定參數名稱爲「UserId」、資料型別爲「OleDbType.Char」、資料的長度爲 10, 所對應的欄位爲「UserId」後, 再將此參數加入 UpdateCommand 物件的 Parameters 集合中管理：

```
prmA=New OleDbParameter( _
  "UserId",OleDbType.Char,10,"UserId")
cmUpdate.Parameters.Add(prmA)    '將參數加入參數集合中
```

故程式將第一筆資料的 UserName 欄位修改後而呼叫 DataAdapter 物件的 Update 方法進行更新時, DataAdapter 會檢查 DataTable 內每一個 DataRow 物件的 RowState 屬性。倘若是內容爲 DataRowState.Modified, 則將欄位 UserId 以及 UserName 的內容取出後, 代入 UpdateCommand 的 SQL 敘述中。故原來的敘述則變成：

```
Update Members Set UserName='賴仁胤' Where UserId='Jacky'
```

上列的敘述傳送至資料源後，就達到資料更新的目的了。

具名參數

具名參數是以「@」開頭來命名參數, 例如「@UserName」。具名參數可以讓 SQL 敘述更容易閱讀, 特別是使用 MS SQL Server 的預存程序時只能使用具名參數。以下爲 SQL 敘述修改成具名參數後的內容：

```
Update Members Set UserName=@UserName Where UserId=@UserId
```

而 Parameter 的宣告也要改成：

```
Dim prmA As New OleDbParameter( _
  "@UserName",OleDbType.Char,10,"UserName")
cmUpdate.Parameters.Add(prmA)   ' 將參數加入參數集合中

prmA=New OleDbParameter( _
  "@UserId",OleDbType.Char,10,"UserId")
cmUpdate.Parameters.Add(prmA)   ' 將參數加入參數集合中
```

詳細程式可參考 Para02.aspx

使用範例

下列範例使用 Parameter 物件建立所有的 Command 物件：

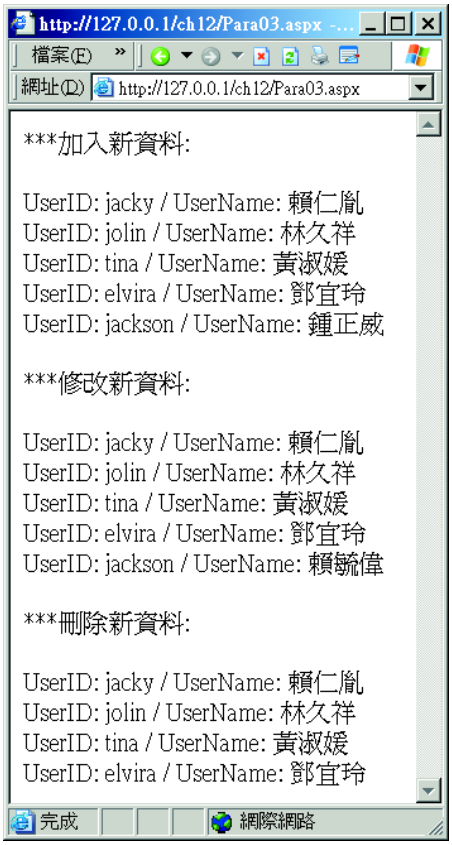
```
<%@Import Namespace=System.Data%>
<%@Import Namespace=System.Data.OleDb%>

<script language="VB" runat="Server">
Private Sub Page_Load(sender As Object, e As EventArgs)
  Dim cnA As New OleDbConnection( _
    "Provider=Microsoft.Jet.OleDb.4.0;" & _
    "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb")
  Dim daA As New OleDbDataAdapter( _
    "Select * From Members", cnA)

  Dim cmDelete As New OleDbCommand( _
    "Delete From Members Where UserId=@UserId", cnA)
  Dim cmInsert As New OleDbCommand( _
    "Insert Into Members Values(" & _
    "@UserId,@UserPwd,@UserName," & _
    "@UserTel,@UserAdd,@UserEmail)", cnA)
  Dim cmUpdate As New OleDbCommand( _
    "Update Members Set UserId=@UserId," & _
    "UserPwd=@UserPwd,UserName=@UserName," & _
    "UserTel=@UserTel,UserAdd=@UserAdd," & _
```

```
    "UserEmail=@UserEmail Where UserId=@UserId", cnA)

  cmDelete.Parameters.Add( _
   New OleDbParameter( _
     "@UserId", OleDbType.Char, 10, "UserId"))
  cmDelete.Parameters("@UserId").SourceVersion = _
   DataRowVersion.Original
  cmInsert.Parameters.Add( _
   New OleDbParameter( _
     "@UserId", OleDbType.Char, 10, "UserId"))
  cmInsert.Parameters.Add( _
   New OleDbParameter( _
     "@UserPwd", OleDbType.Char, 10, "UserPwd"))
  cmInsert.Parameters.Add( _
   New OleDbParameter( _
     "@UserName", OleDbType.Char, 10, "UserName"))
  cmInsert.Parameters.Add( _
   New OleDbParameter( _
     "@UserTel", OleDbType.Char, 10, "UserTel"))
  cmInsert.Parameters.Add( _
   New OleDbParameter( _
     "@UserAdd", OleDbType.Char, 50, "UserAdd"))
  cmInsert.Parameters.Add( _
   New OleDbParameter( _
     "@UserEmail", OleDbType.Char, 50, "UserEmail"))
  cmUpdate.Parameters.Add( _
   New OleDbParameter( _
     "@UserId", OleDbType.Char, 10, "UserId"))
  cmUpdate.Parameters.Add( _
   New OleDbParameter( _
     "@UserPwd", OleDbType.Char, 10, "UserPwd"))
  cmUpdate.Parameters.Add( _
   New OleDbParameter( _
     "@UserName", OleDbType.Char, 10, "UserName"))
  cmUpdate.Parameters.Add( _
   New OleDbParameter( _
     "@UserTel", OleDbType.Char, 10, "UserTel"))
  cmUpdate.Parameters.Add( _
   New OleDbParameter( _
     "@UserAdd", OleDbType.Char, 50, "UserAdd"))
  cmUpdate.Parameters.Add( _
   New OleDbParameter( _
     "@UserEmail", OleDbType.Char, 50, "UserEmail"))
```

```
daA.DeleteCommand = cmDelete
daA.InsertCommand = cmInsert
daA.UpdateCommand = cmUpdate

Dim dtTable As New DataTable()
daA.Fill(dtTable)

Response.Write("***加入新資料:<p>")
Dim tmpRow As DataRow = dtTable.NewRow()
tmpRow.Item("UserId") = "jackson"
tmpRow.Item("UserPwd") = "bear"
tmpRow.Item("UserName") = "鍾正威"
tmpRow.Item("UserTel") = "0915135000"
tmpRow.Item("UserAdd") = "台中市中正區"
tmpRow.Item("UserEmail") = "jackson@hotmail.com"

dtTable.Rows.Add(tmpRow)

daA.Update(dtTable)
dtTable.Clear()
daA.Fill(dtTable)
For Each tmpRow In dtTable.Rows
 Response.Write("UserID: " & tmpRow.Item("UserID") & " / ")
 Response.Write("UserName: " & _
      tmpRow.Item("UserName") & "<br>")
Next

Response.Write("<p>***修改新資料:<p>")
Dim arRow() As DataRow
arRow = dtTable.Select("UserId='jackson'")

If arRow.GetUpperBound(0) >= 0 Then _
 arRow(0).Item("UserName") = "賴毓偉"

daA.Update(dtTable)
dtTable.Clear()
daA.Fill(dtTable)
For Each tmpRow In dtTable.Rows
 Response.Write( "UserID: " & tmpRow.Item("UserID") & " / ")
 Response.Write("UserName: " & _
                tmpRow.Item("UserName") & "<br>")
Next
```

```
  Response.Write("<p>***刪除新資料:<p>")
  arRow = dtTable.Select("UserId='jackson'")
  If arRow.GetUpperBound(0) >= 0 Then arRow(0).Delete()

  daA.Update(dtTable)
  dtTable.Clear()
  daA.Fill(dtTable)
  For Each tmpRow In dtTable.Rows
    Response.Write( "UserID: " & tmpRow.Item("UserID") & " / ")
    Response.Write("UserName: " & _
                   tmpRow.Item("UserName") & "<br>")
  Next
End Sub
</script>
```

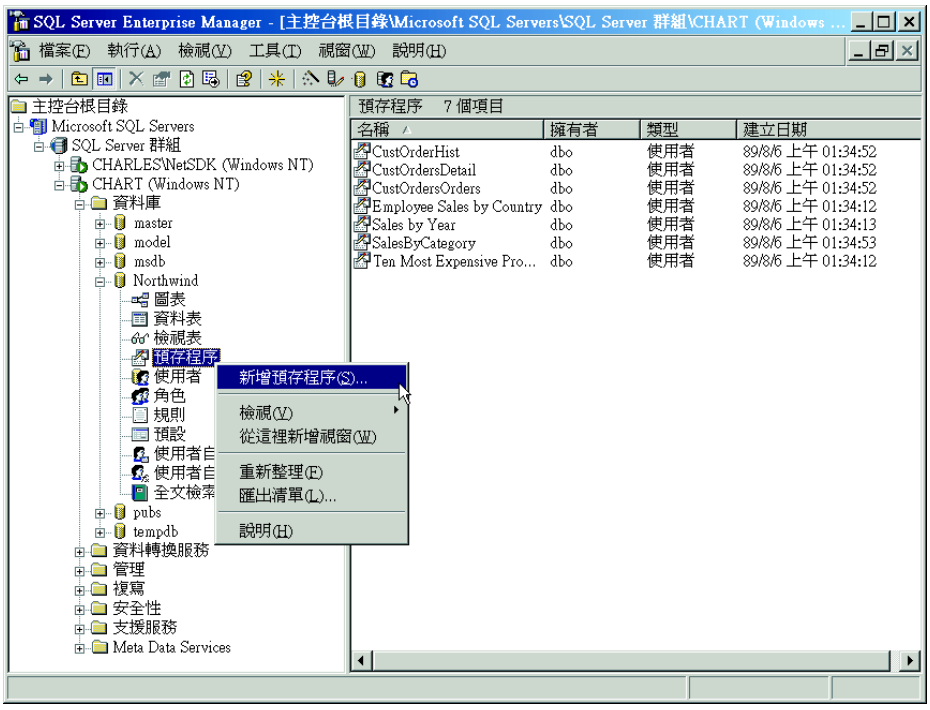
程式 Para03.aspx



Parameter 物件可以搭配 MS SQL Server「預存程序」(Stored Procdure)的使用。預存程序是一種存放在資料庫伺服器內預先寫好的 SQL 敘述，由於預存程序已經被資料庫伺服器經過分析及調整，不但讓執行效能較佳、可供衆多不同應用程式使用、易於管理及維護外，還可以避免資料隱碼的問題。

預存程序的製作

本範例以 MS SQL Server 爲例, 安裝完 MS SQL Server 後可以在「程式集」→「Microsoft SQL Server」群組中點選 SQL Server 的管理工具「Enterprise Manager」, 開啓之後依序展開資料庫伺服器、「資料庫」以及「Northwind」範例資料庫項目，接著在「預存程序」項目上按右鍵後並點選「新增預存程序」：



出現「預存程序屬性」的對話盒後，在對話盒內輸入下列所要使用的預存程序：

```
CREATE PROCEDURE QEmpName
@EID int,@EFN nvarchar(10) output,@ELN nvarchar(20) output
AS
Select @EFN=FirstName,@ELN=LastName
From Employees Where EmployeeID = @EID
GO
```

預存程式 QEmpName

該序存程序名稱爲 QEmpName 並接受三個參數，其中第一個參數爲整數型別的 @EID，負責用來接收所要查詢的員工編號，而 @EFN 以及 @ELN 是 nvarchar 型別的參數，刮號內爲參數所接收的資料長度；這兩個參數最後宣告爲 output 的用意則是指明以傳址的方式接收參數，最後並將查到的使用者姓名存入這兩個參數內，這樣一來使用預存程式的開發人員就可以接收到執行結果。

使用範例

下列範例示範如何使用存放在 MS SQL Server 內的預存程序 QEmpName：

```
<%@Import NameSpace="System.Data"%>
<%@Import NameSpace="System.Data.SqlClient"%>

<form runat="Server">
  員工編號: <Asp:TextBox id="txtEID" runat="Server"/>
  <Asp:Button id="btnOK" text="確定" OnClick="btnOK_Click"
      runat="Server"/><p>
  <Asp:Label id="lblMSG" runat="Server"/>
</form>

<script language="vb" runat="Server">
Private Sub btnOK_Click(sender As Object, e As EventArgs)
  Dim cnA As New SqlConnection( _
   "Data Source=Chart;Initial Catalog=Northwind;" & _
   "User ID=sa;Password=pass")
  Dim cmA As New SqlCommand("QEmpName", cnA)
  cmA.CommandType = CommandType.StoredProcedure
  cmA.Parameters.Add(New System.Data.SqlClient.SqlParameter( _
   "@EID", System.Data.SqlDbType.Int, 4))

  cmA.Parameters.Add(New System.Data.SqlClient.SqlParameter( _
   "@EFN", System.Data.SqlDbType.NVarChar, 10))

  cmA.Parameters.Add(New System.Data.SqlClient.SqlParameter( _
   "@ELN", System.Data.SqlDbType.NVarChar, 20))

  cmA.Parameters("@EID").Value = txtEID.Text
  cmA.Parameters("@EFN").Direction = ParameterDirection.Output
  cmA.Parameters("@ELN").Direction = ParameterDirection.Output
  cnA.Open()
  cmA.ExecuteNonQuery()

  If IsDbNull(cmA.Parameters("@EFN").Value) Then
   lblMsg.Text = "找不到所輸入的員工編號"
  Else
```

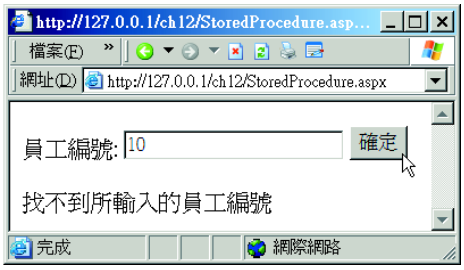
```
    lblMsg.Text = "您所查詢的員工姓名為: " & _
                  cmA.Parameters("@EFN").Value & " " & _
                  cmA.Parameters("@ELN").Value
  End If
  cnA.Close()
End Sub
</script>
```

程式 StoredProcedure.aspx

由於 ParameterDirection.Input 爲參數方向的預設值, 所以 @EID 參數不需要特別指定其 Direction 屬性, 不過由於參數 @EFN 以及 @ELN 要接收預存程序的輸出, 所以必須指定其 Direction 屬性爲「ParameterDirection.Output」：

```
cmA.Parameters("@EFN").Direction = ParameterDirection.Output
cmA.Parameters("@ELN").Direction = ParameterDirection.Output
```

若沒有查詢到員工姓名則參數會收到 DBNull 值, 此時要透過 IsDbNull 進行判斷的工作以免發生錯誤。程式的執行結果爲：



習 題

1. 解釋何謂 Code-Behind 以及其優點。

2. 試利用 Code-Behind 技術製作一驗證使用者帳號及密碼的登入網頁。

3. 試作一個 Pagelet 使用者自訂控制項, 可以執行華攝氏溫度互換的功能。

4. 試作一個自訂 Web 控制項, 該控制項由 Label 、 TextBox 以及 Button 組成, 該控制項提供了 Label 屬性可以存取 Label 所顯示的文字、 Text 屬性可以存取 TextBox 控制項的 Text 屬性、 Message 屬性可以用來存取 Button 控制項的 Text 屬性, 並且在使用者按下按鈕後可以觸發 OnClick 事件。

5. 解釋何謂 Web Services 、 WSDL 、 DISCO 以及 UDDI 。

6. 試製作一英制單位轉公制單位的 Web Service, 使用者輸入英尺英吋後, 可以透過該 Web Service 轉換成公分。

7. 試利用 Forms 驗證模式作爲身份驗證的依據, 並且利用自訂資料庫查驗使用者的身份。

8. 解釋何謂 SSL 以及其工作原理。

9. 解釋何謂資料隱碼、交易以及預存程序。

10. 試將第五章所使用的會員資料表匯入 MS SQL Server 中, 並製作一接受使用者帳號爲參數且輸出使用者姓名的預存程序 QUserName, 並透過參數物件使用該預存程序。