ASP.NET 2.0 專案部署問題研究

作者: 李明儒

Visual Studio 2005 推出了嶄新的網站專案模型--Web Site Project，方便的即時編譯特性讓許多開發者耳目一新，但隨著系統成熟，面對部署及持續更新的需求，這個新專案型別的缺點開始一一浮現...

使用 Visual Studio.NET 2003 的開發者接觸 Visual Studio 2005 後，會發現 ASP.NET 1.1 專案被自動轉換成全新的專案格式(Web Site Project)，有別於以往，操作開發起來更爲簡便，例如:

1. 專案檔(.csproj、.vbproj)不見了，直接開啓 Web 所在的資料夾就可以編輯專案，不需要先掛在 IIS 上，也不需要安裝 FPSE(FrontPage Server Extension)。
2. .cs/.vb 檔修改儲存後就直接生效，不需建置(Build)就可執行看結果。
3. 引用 Partial Class 觀念，Server-Side 程式不再有 OnInit、InitializeComponent 等 IDE 自動產生的 WebControl 物件宣告指令碼。
4. 同一 Web 專案中可以 C#與 VB.NET 並存(雖然這不像是個好主意)。

方便之餘，我們也開始注意到一些略爲不便的差異，例如: 與頁面無關的獨立 Class .cs/.vb 檔必須集中放在 App_Code 目錄下、無法將特定的目錄及檔案排除在專案之外...等等。但最嚴重的問題，卻要到專案部署階段才會現身。

如果不想讓原始碼在正式機上露臉，VS 2005 提供了 Publish Web Site 功能可預先建置 DLL 檔後再部署。但每次編譯(Compile)的 DLL 檔名會夾帶隨機產生的雜湊碼，ASPX<@Page>宣告則會配合隨機檔名修改而每次不同；系統上線後持續性的修 Bug、加功能的工作是少不了的，每次建置出來的檔案不同讓這類小幅更新工作頓時變得棘手。當部署與維護的不便逐漸蓋過開發測試時的便利性，開發社群的不滿漸漸壓過原先的喜悅，微軟面對這波聲浪，採取的策略是亡羊補牢，迅速地提供了配套解決方案以及新的專案選擇。

在本文中，我們將簡介 Web Site Project 的特性，探討部署時會遭遇的問題，並提供解決方案建議。

Web Site Project 的特性

Web Site Project 是 Visual Studio 2005 所引進的新一代專案類型，當初的著眼點是要改良 VS.NET 2003 網站專案的一些缺陷。這些改良更動了底層的編譯程序及專案模型，帶來便利性的同時，也產生了一些副作用。以下對幾個主要的特性改變加以說明，談一談它們的優點及可能造成的影響。

1. Project Based -> File Based

   在 ASP.NET 1.1 專案中，每個網站應用專案會有一個專屬的專案檔(.csproj 或.vbproj)，其中表列了該專案所包含的 ASPX、ASCX、CS/VB、HTM、CSS、JPG、GIF… 等等大小檔案。在 Web Site Project 中，則不再需要.csproj 或.vbproj 檔。原來 Web Site Project 取消了透過檔案維護檔案清單的做法，將整個專案目錄下的所有檔案，自動視爲專案的一部分。同時，它也擺脫了對 IIS 及 FPSE 的需求，只要能取得網站的完整目錄，就可以直接開啓編修整個專案。即使開發機器上沒有安裝 IIS，VS 2005 內建的 Web Development Server，也能啓動一個隨機 Port Number 的迷你 Web Server，用來測試及偵錯網站程式。這些特性簡化了新建及接手專案開發工作的複雜度，但也產生一些缺點，例如: 有些檔案基於管理方便才放在網站目錄下，不需納入專案中被編譯，在 ASP.NET 1.1 Project 中可以將它們排除(Exclude)在專案外，Web Site Project 則無從支援。

2. Code-Behind -> Code-Beside

   以往 ASP.NET 1.1 Project 中，ASPX 中需要透過<asp:TextBox Id="Text1" runat="server">的方式宣告 Web Control，而對應的 Code-Behind 檔中則還有 OnInit、InitializeComponent 兩個 IDE 自動產生的 Method，用來宣告 TextBox Text1=new TextBox();，二者性質重複，也就很容易發生 ASPX 與 Code-Behind 程式不一致的情況。因此在 VS.NET 2003 中，InitializeComponent 的 Method 前後會有註解宣告它們由 Web Form Designer 自動產生，請勿隨意更動。使用者在拖拉 Web Control 後，則常需要在 Design/Source 兩個 Tab 間切換，好讓 ASPX 的變動反應到 Code-Behind 端。

   Web Site Project 引進了新的 Code-Beside 模式，利用 Partial Class 的觀念，InitializeComponent 部分的程式碼改成在使用者存取 ASPX 時才自動產生，現場編譯。如此，沒有 ASPX 與 Code-Beside 宣告同步的問題，Code-Beside 程式中不再出現這些例行化的瑣碎程式碼，讓開發人員可以 100%聚焦在商業邏輯上。

   只是這種自動產生與即時編譯的概念，顛覆了過去所有 aspx.cs 會被編輯在同一個 DLL 中的概念。亦即 WebForm1.aspx.cs 與 WebForm2.aspx.cs 會分別獨立編譯，過去從 WebForm1 去參照 WebForm2 中 Public Member 的技巧變得不可行；而 InitializeComponen 的完全自動化，以往由 Code-Behind 程式動態建立、載入 WebControl、UserControl 的進階技巧變得困難重重。

3. 編譯(Compilation)時機的改變

   在 ASP.NET 1.1 專案中，Code-Behind 部分的.cs/.vb 程式碼必須先編譯成一個 DLL 檔，放在 BIN 目錄下。而在 Web Site Project 的預設模式中，.cs/.vb 不需事先編譯，而會在 aspx 被存取時才會動態編譯(稱之爲隨選編譯 On-Demand Compilation，或即時編譯 Just-In-Time Compilation)。這個新設計帶了一些好處，例如:當我們在做 Server-Side 程式碼除錯時，只需修改後存檔，接著用瀏覽器重新載入，馬上就可以看到結果，不像過去必須重新編譯整個網站專案，再重新啓動偵錯程序，方便許多。因此有個觀念的改變值得注意，當 Web Site Project 以偵錯模式執行時，雖會對整個網站專案進行完整的建置(Build)作業，純粹只是爲了檢測程式碼是否能順利完成編譯罷了，並不在於產生 DLL 供後續使用；待瀏覽器開啓要 Debug 的網頁時，還是得重新編譯 Code-Beside 檔案。由於 Web Site Project 編譯的範圍涵蓋了 ASPX Parsing/ASPX Inline Code/Code-Beside Code，建置速度比 ASP.NET 1.1 Web Project 慢上許多，每次 Debug 前耗費大量時間進行非必要性的編譯動作，對大型的專案來說十分不划算。因此我們可以透過專案屬性設定，設定偵錯執行時完全不做編譯或只編譯單一網頁，另外還可指定建置 Solution 時排除 Web Site Project(如圖 1)，如此可大幅減少開發測試期間的無謂等待。

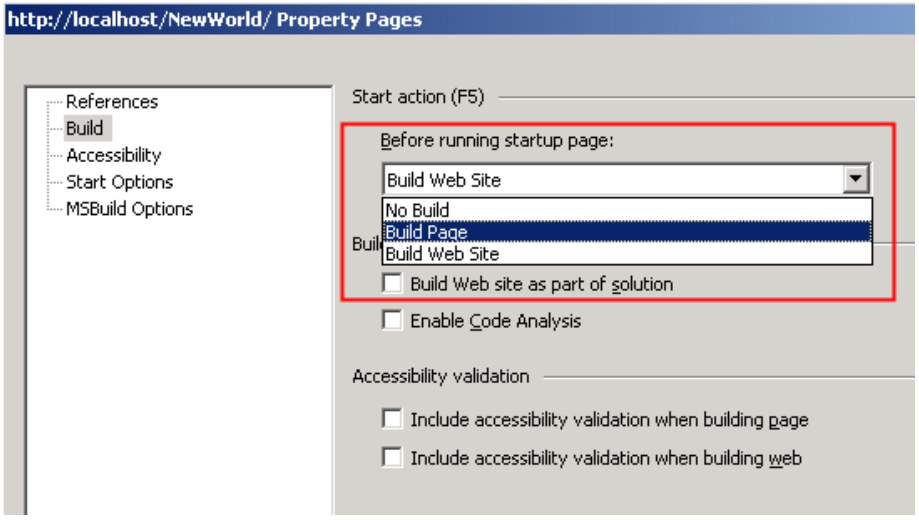

圖 1 透過修改 Web Site Project 屬性改善 Debug 效率

4. App_Code 目錄

   VS 2005 自動升級 ASP.NET 1.1 專案後，你會發現原本散落在專案各角落的獨立 Class 檔案，會被自動集中到一個名爲 App_Code 的目錄下。原因如同先前所提的隨選編譯概念，網頁相關的 Code-Beside 程式會在網頁被存取時才進行編譯，因此跟網頁未直接關聯的程式碼就必須另外處理。放在 App_Code 下的程式碼，會被另外單獨編譯，無論我們如何調整圖 1 的設定，App_Code 裡的程式還是會在建置時，甚至 VS 2005 IDE 操作期間重新編譯(因爲其中可能包含自訂的控制項，設計期間就會用到)。在 App_Code 下的程式可以透過在 web.config 指定 codeSubDirectories 的方式，同時混用 VB.NET 或 C#所寫的類別，只是混用多種語言一般會衍生後續維護上的困擾，能免則免!

   由於 App_Code 編譯時機較爲頻繁，如果放了太多的類別，會影響建置及 VS 2005 操作的速度。此時可考慮將其獨立成一個 Class Library 專案，對整體操作的順暢度將有所幫助。

5. 部署觀念的改變

   誠如先前所討論的，Web Site Project 的建置模式已趨於隨選編譯。但所謂隨選編譯的前題是，Code-Beside 的.cs/.vb 檔案必須存在於 IIS 網站目錄下，才能在使用者存取該網頁時即時進行編譯。將原始程式碼一併部署到正式營運主機，會大幅提高程式邏輯外洩的風險，顯然違背一般的資安原則，正式運作的系統還是應回歸將程式編譯爲 DLL 後再部署的做法。針對這種預先編譯的需求，VS 2005 內建了 Publish Web Site 這項功能，可以對 Web Site Project 進行預先編譯(Precompile)的動作，再一併將 ASPX、DLL 檔案複製到特定目錄下。只是在複製時，我們無法選擇只更新其中某幾個檔案，它會一律將原有目錄清空再全部重新寫入(包含原本的 web.config、額外複製過去的檔案都會遺失)。因此千萬不要用它直接更新正式的網站目錄，以免發生悲劇。比較可行的做法是先 Publish 到本機目錄，再手動將要更新的檔案及 DLL 複製到目的主機上。但另外一個大問題是，Publish Web Site 會爲每個資料夾產生一個檔名包含隨機雜湊碼的 DLL 檔，且每次編譯的雜湊值都不同。當我們只需少量更新時，得一一比對找尋 ASPX 對應的 DLL，是件頭痛的事。Publish Web Site 雖提供了選項，支援"每個網頁擁有固定檔案名的獨立 DLL"(如圖 2)，但也未能徹底解決問題。關於 Web Site Project 部署的一籮筐的問題，稍後再一一細數

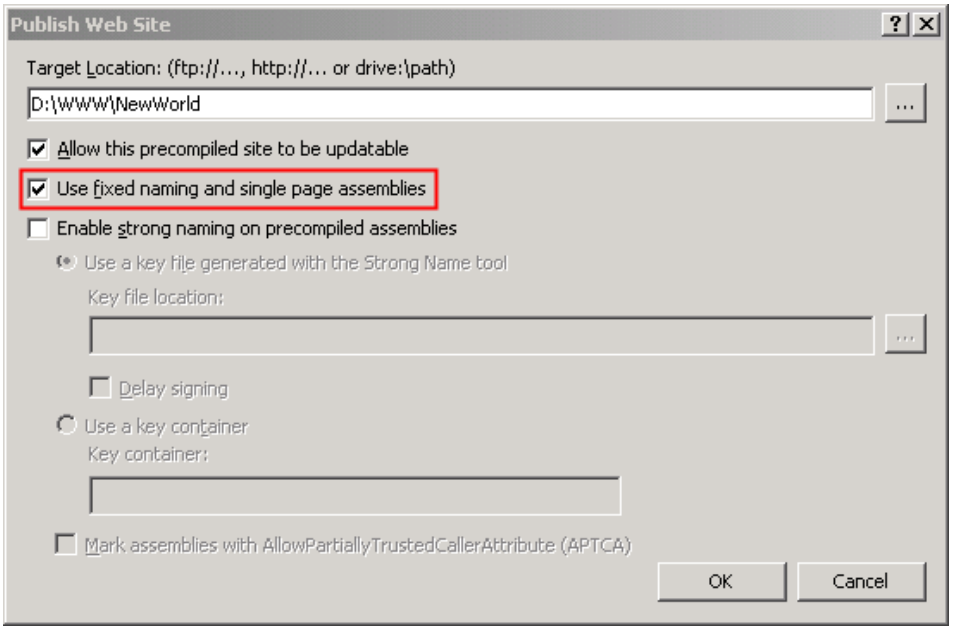

圖 2 Publish Web Site 可指定產生固定的 DLL 名稱

Web Site Project 的部署難題

由以上的說明，Web Site Project 最重要的變革就在於編譯模式的調整，它帶來了一些便利性。例如: Server-Side 程式的更新不需要編輯、免除了 ASPX 與 Server-Side 程式間控制項宣告的同步問題，但這些設計較偏重開發測試階段的改良，並未周詳思考部署時可能遭遇的嚴重問題，VS 2005 內建的部署功能(Publish Web Site)也未直實地切合實務需求。

隨選編譯模式下，直接更改 Server-Side 檔案，不需重新編譯及更換 DLL 檔的想法很酷，但前題是必須將原始程式碼儲存於正式主機上，大大違反了一般公認的資安原則。若要避免隨選編譯模式會曝露原始碼的風險，預先編譯成 DLL 再部署的做法是較合宜的策略。

VS 2005 內建的 Publish Web Site 的功能，可以預先將 Code-Beside 原始碼預先編譯成 DLL，甚至將 ASPX 檔案內容也一併藏入 DLL 中。但是，Publish Web Site 卻有幾項頗爲嚴重的缺點:

1. 編譯完成的檔案會完全覆寫原有的目錄結構，無法指定只更新那些檔案。實務上，開發、測試、正式台的 web.config 多半不同，網站目錄下的子目錄、檔案也可能略有差異。由於 Publish Web Site 會刪除整個目的目錄後全新重建(包含 web.config)，除了第一次全新部署時可以使用，日後的更新就只能先寫到本機目錄，再手工更新檔案。
2. Publish Web Site 功能預設每次編譯出來的 DLL 檔名中都會夾帶隨機產生的雜湊碼，雖然 VS 2005 會自動修改 ASPX，讓<@Page>的 inherits 屬性自動指向新產生的 DLL 檔名(如圖 3)。但如果要手工更新某個 ASPX 的 Code-Beside 程式，就得先查看 ASPX 以找出它對應的 DLL 檔，將該檔案更新至正式機，再將同一目錄下的 ASPX 也一併更新(因爲預設同一資料夾的 Code-Beside 程式會編譯成同一個 DLL 檔)，如果正式機上的舊版 DLL 要一併移除乾淨，還得先由正式版上的原有 ASPX 追查檔名，過程十分繁瑣。

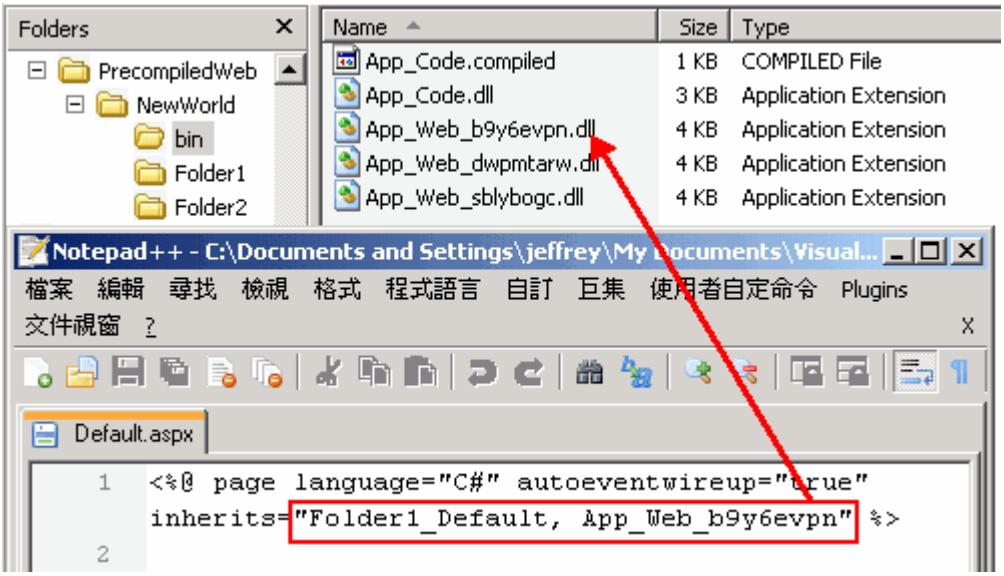

圖 3 VS 2005 會修改 ASPX 內容指向隨機產生的 DLL 檔名

3. 爲了解決前述隨機檔名的問題，Publish Web Site 提供了一個選項，可以設定 DLL 採用固定檔名不再隨機產生，但是該選項會一併將 DLL 拆分成每個 ASPX 網頁專屬一個 DLL 檔。雖然這能減少了更新單一 ASPX 與 DLL 的複雜度，更換 DLL 時，也不會影響到其他的 ASPX；但在大型 Web Application 中，可能包含了數百上千個 ASPX 檔；網站運作過程要涉及這麼多個 DLL 的載入與管理，對效能肯定有害。

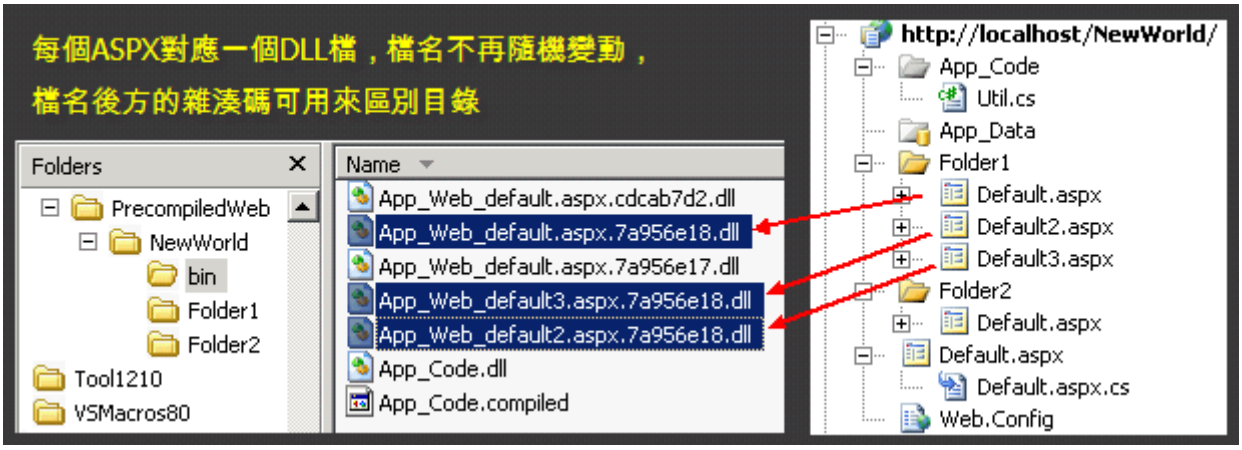

圖 4 固定檔名時爲每個 ASPX 一個 DLL 檔

以上的所提的問題絕非 Web Site Project 的原罪，也並非不能解決。微軟翻修了 VS 2005 的編譯功能，並推出了 MSBuild，這個新一代的建置引擎，功能強大，彈性十足，要克服以上的問題輕而易舉(關於 MSBuild 的介紹與應用說明，讀者可以參考 RUN!PC 146 期第 175 頁沈炳宏先生的文章)。Web Site Project 所遭遇的部署問題，我認爲肇因於制定功能規格時，過於著重開發體驗、測試過程的改良，卻忽視了部署實務的需求。而 Publish Web Site 這個功能，介面上所能提供的選擇太少，辜負了強大的 MSBuild 引擎，迫便開發/部署人員得在少得可憐的幾個選項間左右爲難。

微軟的回應

隨著以 ASP.NET 2.0 開發的專案進入測試、部署階段，Web Site Project 在部署上的缺失也逐漸浮現，並在開發人員社群中引發熱烈的討論(當然不乏怒火中燒的猛烈批評)，微軟傾聽了開發社群的心聲，面對問題，並陸續提出了解決方案。

首先，一個更具彈性的建置選項設定介面以 Add-In 方式推出，以 Web Deployment Project[參 2]的型式讓我們可以對 Web Site Project 編譯與部署的細節進行微調，充分發揮 MSBuild 的彈性與擴充性，以符合各式部署需求。

當然，Web Site Project 架構的改變，影響的並不只部署而已。許多開發者還是習慣 ASP.NET 1.1 時代的專案模式: 以.csproj/.vbproj 控制專案細節的模式、全部的 Code-Bind 程式建置成單一的 DLL；而一些涉及網頁物件繼承、動態控制項建立等進階技巧的網頁，移殖到 Web Site Project 時或多或少需要修改。因此另一種新的專案類別—Web Application Project 也問市了，讓開發者可以依循原先 ASP.NET 1.1 網站專案的概念來開發 ASP.NET 2.0 專案。(Web Application Project 已包含在 VS 2005 SP1 中)

ASP.NET 2.0 部署難題的解決之道

由以上的探討，我們細數了 ASP.NET 2.0 Web Site Project 部署時會遇到的幾個問題，包含了:

1. 隨選編譯(On-Demand Compilation)模式有原始程式碼曝露的風險。
2. Publish Web Site 預設的隨機式 DLL 檔名，增加了後續小規模更新的困難度。
3. Publish Web Site 的固定檔名模式，會細分到每個 ASPX 對應一個 DLL 檔，並不符合大型專案的效能需求。

這些問題讓 Web Site Project 部署及維護起來困難重重，大大抵消了開發、測試期間享受到的便利。微軟從開發社群的反應體察問題，迅速地提出解決對策，一個可以彈性設定編譯/部署細節的 Add-In—Web Deployment Project(WDP)，以及讓開發人員有機會重拾 ASP.NET 1.1 專案模式的新選擇—Web Application Project(WAP)。

Web Deployment Project

WDP 是一個 VS 2005 的 Add-In，可以從微軟的網站免費下載[參 3]。它說穿了就是一個具有 GUI 介面的 MSBuild 設定檔，試圖透過簡便的介面操作以發揮 MSBuild 強大的彈性與擴充能力。WDP 的彈性足以完全克服 Publish Web Site 功能的諸多限制，滿足大部分正式環境的部署需求，再有不足之處，MSBuild 本身提供的擴充性也能提供適當的解決途徑。

以下我們就以實例的方式，一一說明使用 WDP 進行部署工作的優點。假設我們有一個如圖 5 的 Web Site Project，安裝 WDP Add-In 後，在專案的右鍵選單中會多了一個"Add Web Deployment Project…"的新選項，按下後 Solution 中就會自動新增一個 WDP 專案。

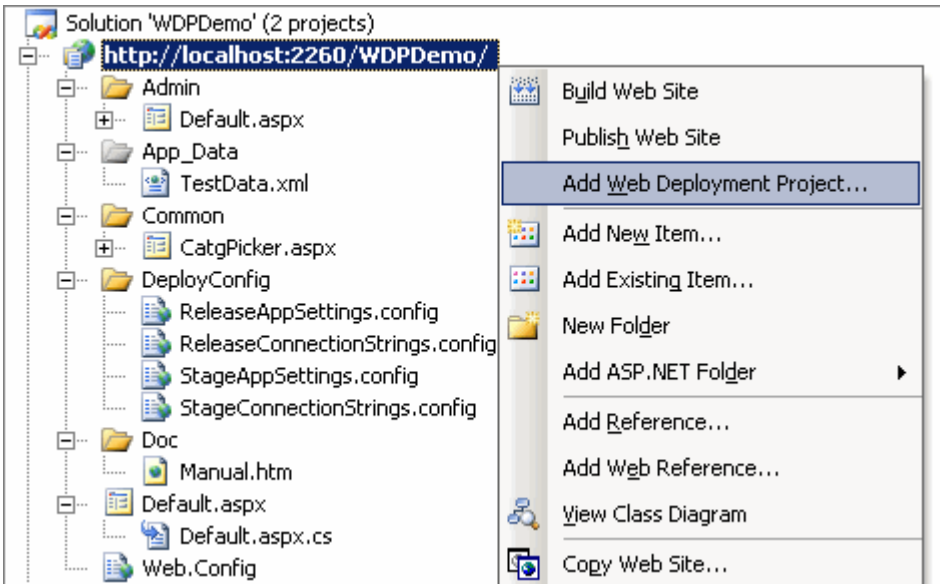

圖 5 爲 Web Site Project 新增 Web Deployment Project

新增的 WDP 專案有兩種編輯方式，可以透過類似專案屬性頁面的 GUI 設定選項及參數，或是透過右鍵選單中的 Open Project File 直接編輯 XML 格式的.wdproj 檔案。有一部分的細節設定，只能透過編輯.wdproj 檔案的方式進行(例如: 指定 AfterBuild 動作)。WDP 可以同時維護多組設定值，例如: 一組針對 Staging、一組針對 Release。每組的設定彼此獨立，不受干擾。

以下就來看看 WDP 設定時可用的選項，並說明 WDP 所提供的便利性。圖 6 爲幾項基本的編譯設定，其中可以決定是否要產生 Debug 用的.pdb 檔、透過 IIS Metabase 資料決定原始程式路徑，以及是否保留 ASPX 檔內容。

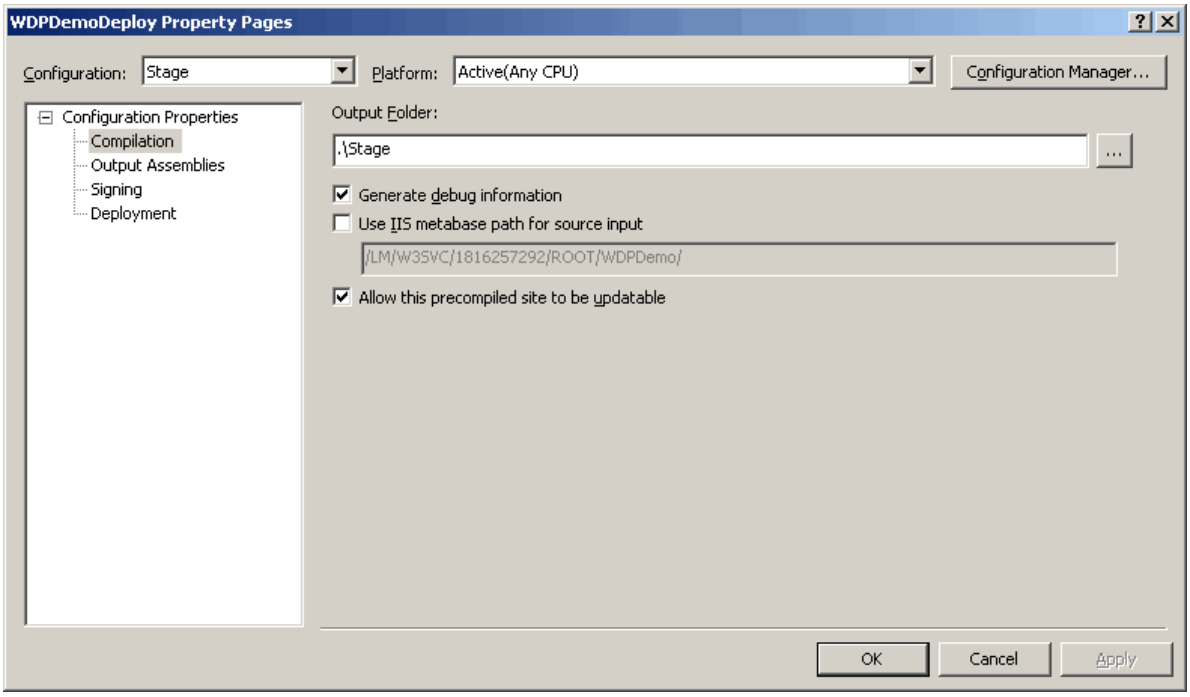

圖 6 WDP 的基本編譯設定

ASP.NET 1.1 Project 在建置時產生的單一 DLL 檔，其實只包含了 Code-Behind 的程式內容，ASPX 內的程式相關部分還是得留待使用者存取網站時才動態編輯。而 Web Site Project 則多了一種選擇，可以連 ASPX 內容也預先編譯(PreComplie)成 DLL，如此正式台上可以不需要儲存 ASPX 檔。

當我們取消"Allow this precompiled site to be updatable."時，所有 ASPX 的內容都會編譯後存入 DLL 中，原本的 ASPX 檔變成只有一列文字的空殼，目的只爲了避免 IIS 傳回找不到檔案的訊息(如圖 7)。連 ASPX 的內容都隱藏，資安當然加分，但相對的缺點是未來就無法透過更新單一 ASPX 檔的方式做小幅修改。另外，由於 ASPX 的內容隱藏在 DLL 中，IIS 得知道要去哪裡找到該 ASPX 的內容，因此 BIN 下會儲存檔名格式如 default.aspx.7a956e18.compiled 的 XML 檔案，用來對應虛擬的 ASPX 路徑及所屬程式類別，檔名中的 7a956e18 爲資料夾路徑所產生的雜湊碼，以區別不同目錄下的同名檔案。IIS 在啓動網站應用程式，會將這些.compiled 檔載入記憶體中，以利稍後使用者存取時能正確找到預先編譯好的程式碼。

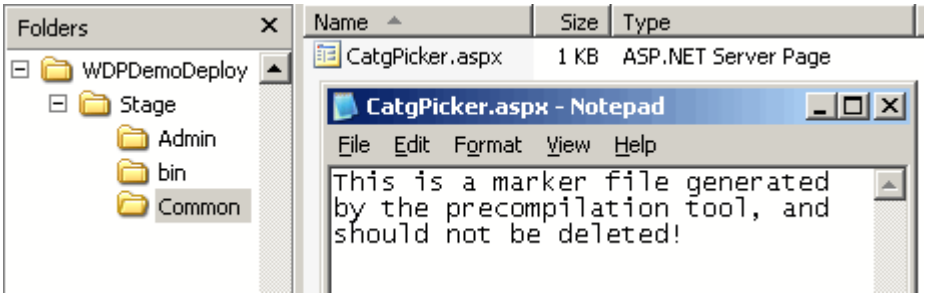

圖 7 ASPX 檔只剩一列註解文字

WDP 最大的改進，即在可以避免 DLL 檔名夾雜隨機雜湊碼的問題，讓後續的管理與更新作業趨於合理。輸出組件檔(Output Assemblies)設定頁(參見圖 8)即爲 WDP 的精華所在，透過這裡的設定，可針對不同的部署需求，決定用單一 DLL 檔、每個資料夾一個 DLL 檔，或是以網頁爲單位分檔。以下說明各種選擇的意義:

1. Merge all outputs to a single assembly

如同 ASP.NET 1.1 專案般，將整個網站專案的所有程式內容編譯成單一的 DLL 檔。

2. Merge each folder output to its own assembly
   每個資料夾的程式編譯成一個 DLL(如圖 9)，可以避免因更新少數 ASPX 網頁而影響到整個網站應用程式，又可避免每個網頁一個 DLL 造成效能問題，是我最常用的選項。
3. Merge all pages and controls to a single assembly
   UI 的部分(ASPX、ASCX… 等等)編譯成一顆 DLL，而與 App_Code、App_WebReferences 等 DLL 獨立，如此可單獨更新 UI 或 App_Code。
4. Create a separate assembly for each page
   爲每一個 ASPX 產生一顆專屬且包含雜湊碼的 DLL，Publish Web Site 功能中也有同樣的選項。雖然可將更新單一網頁的影響範圍縮到最小，但是當專案 ASPX 數量龐大時，會有效能降低與管理不易的副作用。

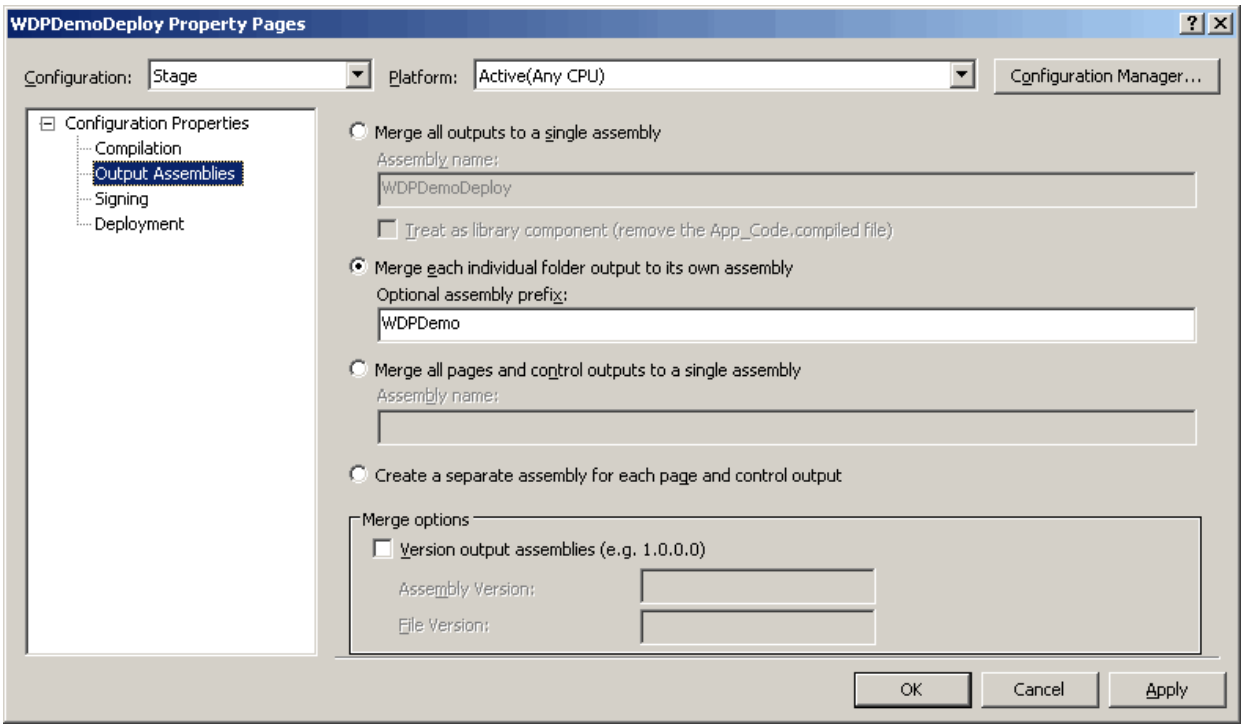

圖 8 輸出組件檔(Output Assemblies)設定頁

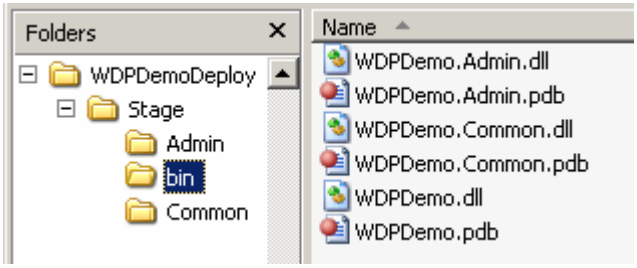

圖 9 每個資料夾下的程式編譯成一個 DLL 檔

web.config 置換功能

WDP 除了可以解決 DLL 編譯問題外，還有不少方便的設計，其中我覺得最有用的是能爲 Debug、Release

等各組態設定不同的 web.config 設定值。在實務上，網站程式在 Debug、Staging、Production 等主機上使用的資料庫連線字串、檔案路徑、郵件伺服器位址等不盡相同，因此要同時維護多分 web.config 檔，更新檔案時要特別留意不要被覆寫。

WDP 支援所謂的 web.config 置換功能，可以爲各組態指定不同的 appSettings、connectionStrings 內容，透過如圖 6 的設定畫面，可以在建置網站時，指定依組態由不同的檔案讀取 appSettings、connectionStrings XML Node，置換原始 web.config 的設定。如此，在不同組態下進行部置，可自動產生適用目的主機的 web.config 檔案，頗爲方便。

不過要注意，指定的配置資料檔案中只能包含如<appSettings><add …/></appSettings>的單一 XmlNode，且 appSettings 與 connectionStrings 必須要分存兩個檔案。

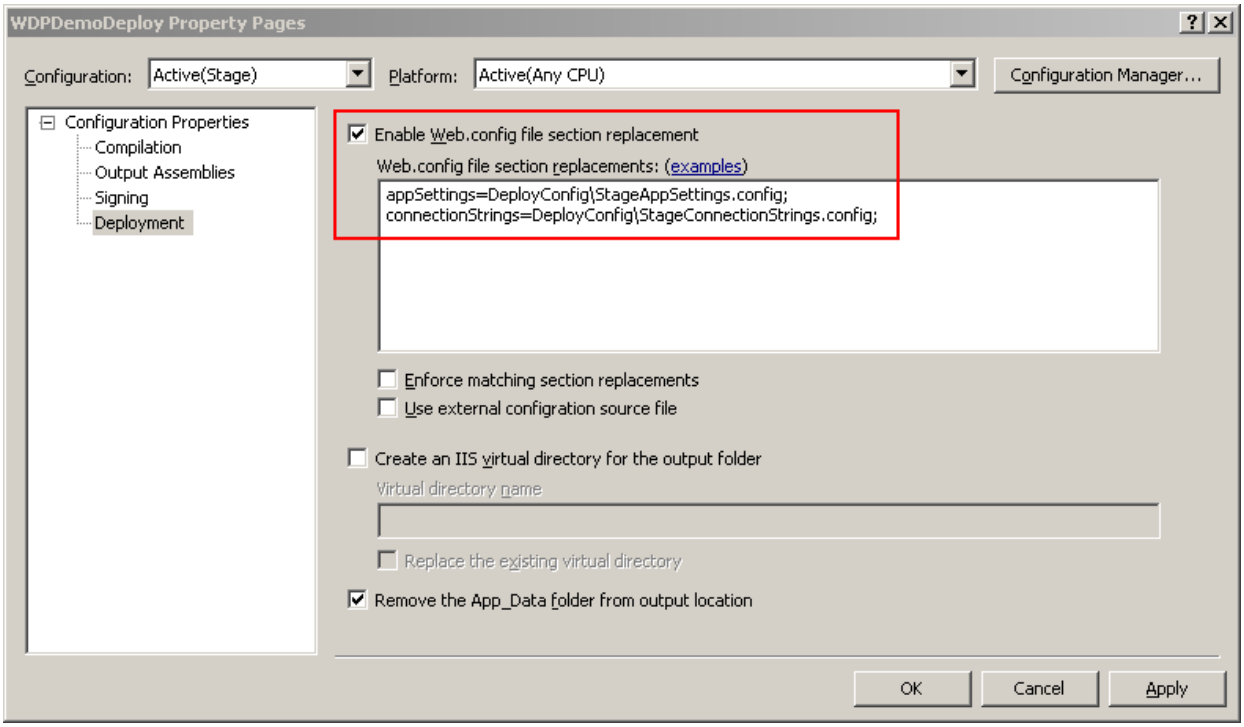

圖 10 web.config 置換設定

客製部署作業

除了編譯之外，部署工作可能還要一些額外動作，例如: 刪除不必要的目錄、額外複製檔案…等等。由於 WDP 的核心就是由 MSBuild、aspnet_compiler.exe、aspnet_merge.exe 架構而成。而 MSBuild 本身是一個具備相當彈性及擴充性的建置平台，要達成前述的檔案刪除、複製工作輕而易舉，若真有不足，還可以透過自己開發 Task 的方式無限擴充。

以下利用一個簡單的例子示範如何客製部署作業: 假設我們有一個如圖 1 的專案，其中 Doc 目錄下爲操作相關文件，不需要部署至正式主機，希望將其排除。另外，配合 Staging、Release 的 web.config，在 DeployConfig 目錄下放了四個設定檔，它們不需要被部署到正式主機，但如被排除，建置過程中會參考不到，所以必須採取事後刪除的做法。

利用 Open Project File 的右鍵選項開啓 WDP 的 wdproj XML 檔案，就可以如圖 11 進行修改。其中，ExcludeFromBuild 可以在建置時忽略 Doc 目錄。由於建置期間必須用到 DeployConfig 目錄下的檔案置換

web.config，不能排除，所以我們可以安排了一個 AfterBuild Task，在其中以 RemoveDir 方式將目的資料夾下的 DeployConfig 目錄全部清除。另外因爲用了 ExcludeFromBuild，建置完成後會遺留下一個 Source 資料夾，也在 AfterBuild Task 中一併清除。

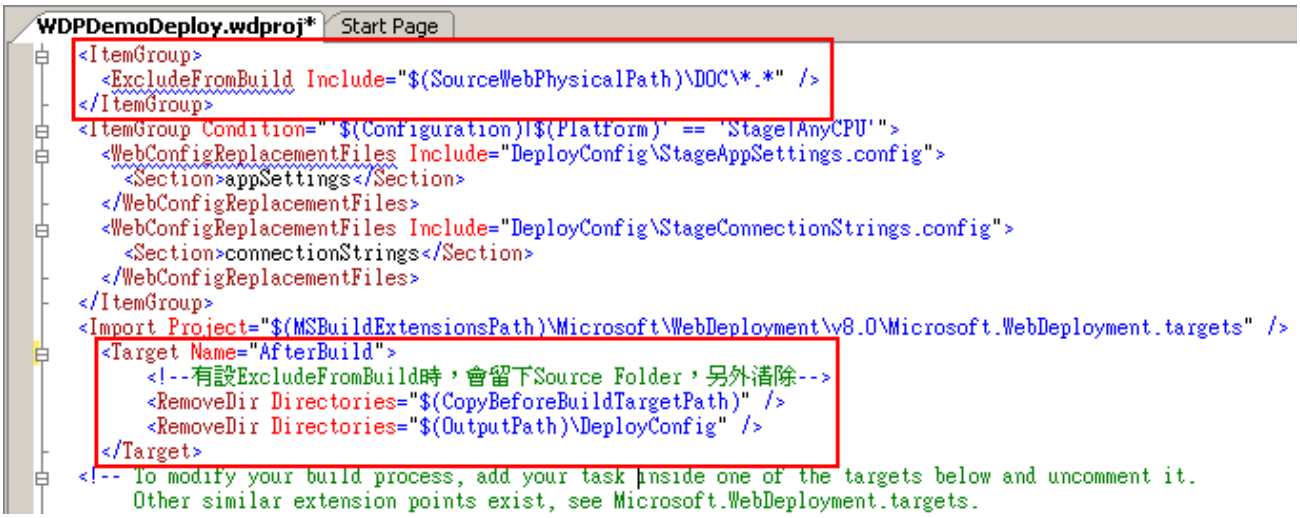

圖 11 修改 wdproj 進行進階客製

關於 WDP 進階客製的說明，可以參見微軟 MSDN 中的說明[參 4]。

Web Application Project

看來 Web Site Project 配合 Web Deployment Project 後，算是解決了部署問題。但對一些長期耕耘 ASP.NET 1.1 專案的朋友來說，Web Site Project 的架構畢竟跟 ASP.NET 1.1 時代有些差距，雖然可以自動升級，但還是可能會遺留一些需要手動修改的地方。例如: MVC 模式下獨立元件需要參考 Page 物件及使用者控制項、Page 物件繼承行爲的改變…等等。

強制由 ASP.NET 1.1 專案升級至 Web Site Project 對一些人來說，得耗費相當的時間調整修改才能順利執行，而摸索熟悉新的模式也需要學習成本，不少人開始懷舊起來。於是微軟推出了 Web Application Project(WAP) Add-In，一種遵循 ASP.NET 1.1 專案概念的專案型式，Code-Behind 編譯模式、單一 DLL 輸出、用.csproj 設定專案、透過.aspx.cs.designer 拿回建立物件的主導權… 等等。在 WAP 出現後，開發人員現在有兩種選擇，保有 ASP.NET 1.1 的專案成果，沿用既有的知識與技巧，或是接受 Web Site Project 的優點，投入 Web Site Project 專案模式的開發。微軟在策略上將對 Web Site Project 與 Web Application Project 提供同等的支援，開發人員可以視自己的需求選用適合或偏好的專案類型。

WAP 初期是以 Add-In 的方式提供免費下載，但目前已融入 VS 2005 SP1。安裝 VS 2005 SP1 後，新增專案時就會多了 ASP.NET Web Application 專案類別可以選擇。Web Site Project 則需要透過簡單的手動程序轉換成爲 WAP，詳細的轉換步驟可以參考 Scott Guthrie 的說明文件。[參 5]

結語

在本文中，我們檢視了 Web Site Project 在部署上面臨的問題，也介紹了能提供完整部署功能的 Web Deployment Project，以及重回 Code-Beside 模式的 Web Application Project，它們補足了 VS 2005 內建 Publish

Web Site 功能的不足，爲 ASP.NET 2.0 提供了可用的部署解決方案，苦於部署無方的 ASP.NET 2.0 開發們可以參考看看。

【參考資料】

參 1 ASP.NET 2.0 的 CodeBeside 模式 http://tinyurl.com/y7plwe

參 2 Web Deployment Project http://msdn2.microsoft.com/en-us/asp.net/aa336619.aspx

參 3. Web Deployment Project 下載 http://msdn2.microsoft.com/en-us/asp.net/aa336619.aspx

參 4 使用 Web Deployment Project http://tinyurl.com/a5jan

參 5 Upgrade Web Site Project to VS 2005 Web Application Project http://tinyurl.com/ymvrfw