C#中自己動手創建一個Web Server（非Socket實現）

2017-01-11

**簡介：**

**目錄**

* 介紹
* Web Server在Web架構系統中的作用
* Web Server與Web網站程式的交互
* HTTPListener與Socket兩種方式的差異
* 附帶Demo源碼概述
* Demo效果截圖
* 總結

**介紹**

本篇文章主要介紹使用HTTPListener類型自己動手創建一個Web Server，創建的Web Server能夠接收來自流覽器端的HTTP請求，並且能夠傳遞給對應的Web網站進行處理，最後將處理結果（Html或者其他格式）返回給流覽器。

博主前面曾經介紹過使用Socket模擬Web Server的運行過程、Socket模擬流覽器發送HTTP請求過程。分別參見：

* [**socket請求Web伺服器過程**](http://www.cnblogs.com/xiaozhi_5638/p/3912668.html?spm=a2c6h.12873639.article-detail.5.3e87474fv2W2u1)
* [**Socket實現簡單Web伺服器**](http://www.cnblogs.com/xiaozhi_5638/p/3917943.html)

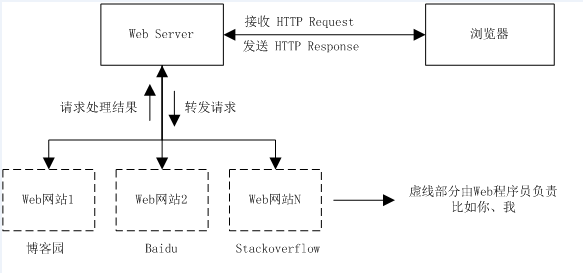
本篇文章並沒有使用Socket去實現，而是使用另外一種封裝程度更高、更抽象的[System.Net.HTTPListener](http://msdn.microsoft.com/zh-cn/library/system.net.httplistener(v=vs.110).aspx" \t "_blank)類型實現。

**Web Server在Web架構系統中的作用**

Web Server在一個B/S架構系統中起到的作用不僅多而且相當重要，Web開發者大部分時候並不需要瞭解它的詳細工作機制。雖然不同的Web Server可能功能並不完全一樣，但是以下三個功能幾乎是所有Web Server必須具備的：

* 接收來自流覽器端的HTTP請求
* 將請求轉發給指定Web網站程式（後者由Web開發者編寫，負責處理請求）
* 向流覽器發送請求處理結果

下圖顯示Web Server在整個Web架構系統中所處的重要位置：

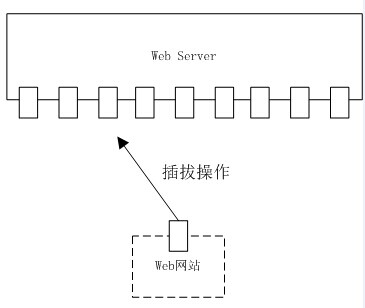


如上圖，Web Server起到了一個“承上啟下”的作用（雖然並沒有“上下”之分），它負責連接用戶和Web網站。

我們可以看到，Web Server預設情況下需要與Web開發者編寫的Web網站程式“通信”（圖中假設三個網站均在一個Web Server上），那麼這裡怎麼處理呢？即時上，任何Web開發者在使用某個平臺開發Web程式時，必須遵守某些“規則”，比如使用到某些框架等。遵守了這些規則，開發出來的網站就可以放到Web Server上，這是不是有點像“程式擴展”的意思？

**Web Server與Web網站程式的交互**

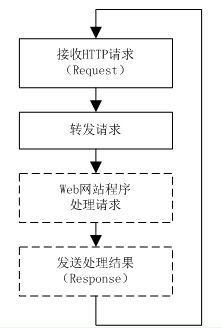
每個網站就像一個個“外掛程式”，只要網站開發過程中遵循了Web Server提出的規則，那麼該網站就可以“插”在Web Server上，我們便可以通過流覽器訪問網站。



理論上講，每個Web Server均是一個宿主，而每個網站均是一個外掛程式（plug-in）。Web Server主要負責通訊等功能，網站程式主要負責資料處理。

至於“宿主”怎樣與“外掛程式”通信，請參見博主前面的一篇文章“[應用程式擴展](http://www.cnblogs.com/xiaozhi_5638/p/3347275.html)”。

由於每個Web Server均能持續接收HTTP請求，因此每個Web Server中均應該存在一個類似下圖所示的迴圈結構：



如上圖，為了提高Web Server接收HTTP請求的效率，圖中虛線框一般採用非同步處理，請求處理過程不會影響整個迴圈。

**HTTPListener與Socket兩種方式的差異**

事實上，HTTP協定是應用層協定。資料在傳輸層上依然是採用TCP進行傳輸的，因此，自己動手採用Socket方式完全能夠類比出Web Server的工作過程（正如文章開頭講到的）。博主前面有一篇博客講述“使用Socket請求Web Server”，其實就是使用Socket來類比流覽器的通訊行為。在.NET中的System.Net命名空間中，包含一些更高層次、更抽象的類型也可以完成對流覽器的模擬，如System.Net.HTTPWebRequest和System.Net.HTTPWebResponse等類型，至於它們和直接使用Socket有什麼區別，請參見下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 分類 | Web Server端 | 流覽器端 | 優點 | 缺點 |
| Socket方式 | **Socket.Accept：**負責接收流覽器端的Socket連接請求  **Socket.Receive：**負責接收流覽器發送的資料  **Socket.Send：**負責向流覽器發送資料 | **Socket.Connect：**負責向Web Server發送連接請求  **Socket.Receive：**負責接收Web Server發來的回復  **Socket.Send：**負責向Web Server發送請求 | 更底層，靈活性更強 | 更底層，需要充分瞭解HTTP協定、TCP/IP協定 |
| System.Net  命名空間中的類型 | **HTTPListener.GetContext：**負責接收流覽器端的HTTP請求  **HTTPListenerRequest：**該類負責接收流覽器端的請求（Request）資料  **HTTPListenerResponse：**該類負責向流覽器發送回復（Response）資料 | **HTTPWebRequest：**該類負責向Web Server發送HTTP請求  **HTTPWebResponse：**該類負責接收來自Web Server發來的回復 | 更高層級別的抽象，不需要過多的瞭解HTTP、TCP等通訊知識 | 更抽象，用法固定（不過需要的都已經包含） |

可以看到，以上兩種方式最終達到的效果其實是一樣的。

**注：請區分HTTPWebRequest與Asp.NET中的HTTPRequest。後者只能用在Asp.NET中，屬於Asp.NET中的核心物件。同理請區分HTTPWebResponse與Asp.NET中的HTTPResponse。它們的命名空間分別為：System.NET和System.Web。**

**附帶Demo源碼概述**

源碼包含三個項目，分別為：

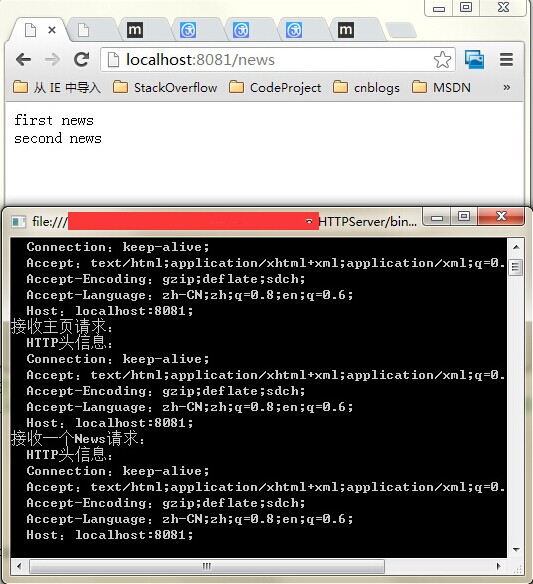
* HTTPServer：模擬的一個Web Server（不足70行代碼）
* HTTPUtility：一個抽象層，專門為了Web Server與網站程式之間的交互。這裡充分應用了“依賴倒置原則（DIP）”，目的就是降低Web Server與網站程式之間的耦合度。
* MyWebsite：一個（類比的）網站程式，需要依賴HTTPUtility。

如果將Demo中的三塊與現實一一類比，那麼HTTPServer便是IIS/Apache，HTTPUtility便是我們開發Web程式時需要使用到的框架/原則，MyWebsite便是我們開發出來的Web網站程式。

將編譯之後的MyWebsite專案DLL檔拷貝到HTTPServer可執行程式同一目錄下的web資料夾中即可（類似一個網站發佈的過程）。打開HTTPServer.exe檔運行，即可在流覽器中訪問MyWebsite網站。

源碼中注釋比較詳細，在此就不多說源碼的事情。

**Demo效果截圖**

****

**總結**

兩種方式實現的過程、代碼結構均類似。主要掌握兩點：

* Web Server中的迴圈結構（泵），負責持續接收請求
* Web Server與網站程式（Plug-in）之間的交互