介紹侏儸紀時代的網站壓力測試工具-ACT

作者：李明儒

時間都來到 2007，眼看 Visual Studio 2008 都要上市了！是哪個死白目又回頭介紹起 Visual Studio.NET 2003 的壓力測試工具?? 大家先別氣到要拔網路線，待我說明原委。

這兩天，同事的系統需要做壓力測試，問我有無建議的工具。想了一下，Visual Studio 2005 的壓力測試工具只剩 Tester Edition 及 Team Suite 才有，有些開發人員裝的是 VS 2005 Developer Edition 未必有得用。新的壓力測試專案雖然換湯不換藥，使用操作方式卻已有所改變，我還沒花時間學(事實上，看來也甭學了，等 VS 2008 好了)，讓侏儸紀時代的阿公級壓力測試工具—Application Center Test 再露個兩手，一樣做出漂漂亮亮的測試報表，有何不可?

我翻出兩年前在雜誌發表過介紹 ACT 的舊文章，讓有需要的人參看看，早就精通 ACT 或對老舊工具過敏的朋友請略過本文。

【注意】這篇文章的時空背景是 2004 年，請大家抱著懷舊的心情閱讀，勿挑剔其中不合時宜之處。

壓力測試簡介

對網站開發者而言，設計與建置一個足以承載每秒上萬使用者同時使用的網站系統，是項困難重重的挑戰，也是項無比的榮耀，而其中蘊含了跨及軟、硬體、網路、作業系統、架構規劃、設計哲學等種種的深奧學問。並非每個開發者所開發出來的網站都有機會上線親臨千軍萬馬的震憾教育(有時是非戰之罪，系統設計得再強韌，行銷與營運未必能建立足夠的人氣)，但在那一天來臨之前，要如何確認自己設計的系統夠強悍，能在嚴苛的考驗下存活? 總不能兵臨城下時再砌磚築牆吧！於是，壓力測試成爲設計者驗證系統負載能力的有效工具。

壓力測試受重視，與網路應用的興起有關。在最早期的軟體應用中，程式常常專爲某一使用者所獨用，因此最重要的問題是程式執行的結果是否正確，系統的回應時間也幾乎由程式的演算法設計決定。在網路開始普及並逐步成爲電腦系統對外的主要管道後，在軟體應用上也起了革命性的變化。

使用者的數量不再受限於實體終端機的數目，一旦伺服器的網卡連上企業內部網路，就可能面臨上百名員工在同一時間發出需求，執行同一軟體。若連接到網際網路，更有可能面臨破萬的使用者蜂踴而入，網站愈是熱門，網站系統人員愈是生活在水深火熱之中。

不知大家之前有沒有留意過一則新聞，在所得稅報稅的最後幾天，接受上傳的伺服器忙線到不支倒地，廠商緊急再新增兩伺服器加入服務才化險爲夷。事情最後雖然是順利解決了，但負責系統維運的相關人員，想必過了好一陣子食不下嚥的難熬日子。

因此，針對預期會有多人使用的系統(網站系統即是典型)，一般除了系統功能是否正確符合設計的驗證型測試，還會進行所謂的壓力測試(Stress Testing)，亦即以大量的負載需求，模擬出上線時會產生的真實狀況，了解系統(其實嚴格來說是指軟硬體的整體)的容量上限。測試結果有二個用途，一則是用來做爲容量設計的參考，例如：對具有負載平衡(Load Balance)的架構來說，可以做爲決定伺服器數目的參考。再者可做爲效能調校的參考，例如：修改演算法或資料庫索引設計後，檢視效能是否有所提升?

這個龐大的測試工程一般會以軟體來模擬出同時間大量的使用者請求，否則要協調 10 萬個使用者同時上線所需的配套作業，幾乎是件不可能的任務。(線上遊戲大概是壓力測試中的異數吧！總會有許多熱心的測試員會迫不急待的要參與計劃，並提供建議)。因此，有不少的壓力測試軟硬體應運而生，亦有提供專業壓力測試服務

的公司存在，全是因應網路伺服器系統在此一方面的殷殷需求。而微軟也沒有忽略掉這一塊，在 Visual Studio .NET Enterprise Architecture/Developer Edition 中提供了 ACT(Application Center Test)這個工具。

ACT 基本概念

ACT，由其名詞即可知道，這又是微軟從 Microsoft Application Center Server 所提取出來整合入 VS.NET 中，由說明文件可知在 Application Center 的版本中，還具備多台 ACT Client 聯合測試、靜態測試等更先進的功能。但就一般的用途而言，VS.NET 中所附的功能即已足用無虞了。

其實在 ASP 時代，微軟亦提供了免費的 WAST(Web Application Stress Tool)，可對網站進行壓力測試，筆者也曾使用過一段時間。相較之下，ACT 在介面上更爲友善，多了了自動繪製統計圖表的功能，而最具震撼力的莫過於所有的測試過程是以 VBScript 的方式儲存，並開放使用者進行修改。這一點爲 ACT 創造了無限的可能，花一些心思，開發者將可測試腳本修改爲由前一傳回結果決定後續 Request 的智慧型測試，也可判讀結果是否符合預期，並與其他機制串連。ACT 搖身一變，可以成爲自動化測試網站功能的工具。試想，若每次改完程式都能對所有功能做一次完整的測試，對軟體品質管控，將有莫大的助益。只是要編修出功能測試 Script，也得花上相當功夫，但若在功能規格未變的前題下，辛苦一次，就能使用自動機制不厭其煩的仔細驗證，相較於人工測試，不必受耐心、毅力與 EQ 等人格特質的干擾，應是划算的投資。

一般而言，ACT 最便捷的使用方式是利用它的錄製功能，只需開啓 IE，進行瀏覽網頁的動作即可。此時，ACT 會記下 IE 與網站間的所有互動，結束錄製後，可以從 Script 視窗看到對網站的每一個 HTTP Request(不管是 ASP、HTML、圖檔、CSS… 每一個由網站取回資料的 Request 都被記錄下來)都轉化爲一個個 VBScript Function Call，甚至使用 HTML Form 以 POST 方式送出的資料也是以字串內容的方式呈現。這意味著所有的 Request 都可以被調整修改以符合我們的測試目標。

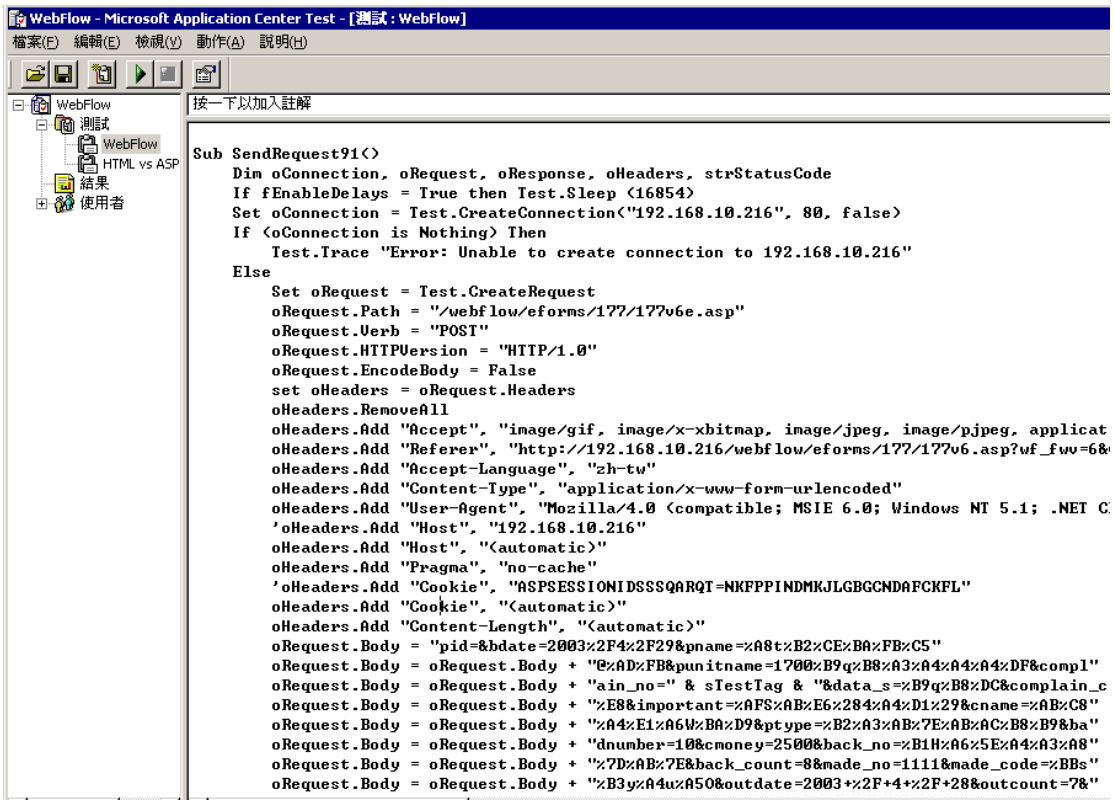

圖 1 利用錄製功能可將網頁存取動作儲存爲 VBScript 形式

ACT 的使用相當直覺而簡單，而且說明文件也頗爲詳細，在此便不多做贅述。而在熟悉 ACT 的操作後，效能調校工作變成一件有趣的工作，由於 ACT 將測試程序變成 Click-And-See 般的單純，因此，開發者可儘情的進行各種修改，並即刻知道其所帶來的改變。

HTML, ASP, ASP.NET 初次交鋒

在熟悉了 ACT 之後，我找了個老掉牙的主題，讓 ASP 與 ASP.NET 對戰一下，進行第一個測試--HTML、ASP、ASP.NET 的效能比較。

HTML 檔案的存取不涉及任何解析及程式邏輯，速度最快應該無庸置疑，而 ASP 與 ASP.NET 對決呢？是否從系統架構來說就即可論斷？根據微軟.NET的文件可知，基於ASP.NET使用的是編譯(Compile)好的程式碼，自然比起 ASP 的解譯式(Interpret)式 Script 快！現在，我們就使用 ACT 來做一個證實。

爲求公平起見，我們準備 HTML、ASP、ASP.NET 檔案各一，做的是相同的事--顯示現在的時間。

HTML 的 Code 如下：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <TITLE></TITLE>
        <META NAME="GENERATOR" Content="Microsoft Visual Studio 7.0">
    </HEAD>
    <BODY>
        <P><FONT face="新細明體">現在時間=
        <script>
        var now   //取得時間的變數
        var geth  //取得時的變數
        var getm  //取得分的變數
        var gets  //取得秒的變數
        var getyear
        var getmon
        var getdate

        now=new Date()  //擷取時間
        geth=now.getHours()  //擷取時
        if(geth<=9) geth="0"+geth  //如果數字小於 2 位數就補上 0
        getm=now.getMinutes()  //擷取分
        if(getm<=9) getm="0"+getm
        gets=now.getSeconds()  //擷取秒
        if(gets<=9) gets="0"+gets
        getyear=now.getYear()
        getmon=now.getMonth()+1
        if(getmon<=9) getmon="0"+getmon
        getdate=now.getDate()
        if(getdate<=9) getdate="0"+getdate
        var Now=getyear+"/"+getmon+"/"+getdate+"
"+geth+":"+getm+":"+gets
        document.write(Now);

        </script>
        </FONT></P>
    </BODY>
```

```
</HTML>
```

ASP 程式碼如下：

```
<% Option Explicit %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 7.0">
<TITLE></TITLE>
</HEAD>
<BODY>
現在時間=<%=Now%>
</BODY>
</HTML>
```

ASP.NET 則在 WebForm 中加了一個 Label Web Control，接著在 Page_OnLoad 的程式碼則如下：

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Put user code to initialize the page here
    Label1.Text="現在時間="+DateTime.Now.ToString("yyyy/MM/dd
hh:mm:ss");
}
```

在正式測試之前，補充一個觀念。在壓力測試時，目的在飆出系統的極限，除了伺服器端要承受很大的負載外，測試端(即 ACT 所在的機器)爲了要模擬出大量的 Request(要求)，CPU 的負擔也不小。所以在 ACT 說明文件上也提到在做壓力測試時，應使伺服器保持在 80%以上的 CPU 使用率，否則很有可能測試的瓶頸來自於測試端或網路頻寬，而非伺服器端，結果就可能失真。

說明一下我所使用測試平台：

- 伺服器端：Celeron 900MHz 512M RAM Windows 2000 Server + SQL Server 2000
- 測試端：Pentium 4 2.4GHz 1G RAM Windows 2000 Server

讀者可能會有點狐疑，是不是寫反了？其實這是因爲我們測試的目的在於比較 ASP 與 ASP.NET 的效能，而不像 PC DIY 裡的硬體效能競技，並不追求數值的突破，而是著重於不同測試值之間的比較。若伺服器的效能太強，可能測試端的 CPU 都已破表了(CPU Load 100%)，伺服器仍遊刃有餘，反而看不出效能的差異。

以此種組合，大約 ACT 模擬五個 Client 端同時存取時，測試端 CPU 維持在 100%，伺服器端則超過 90%，符合我們的要求。

因此，模擬 5 個 Client，分別測試 HTML、ASP 與 ASP.NET，結果如圖 2：

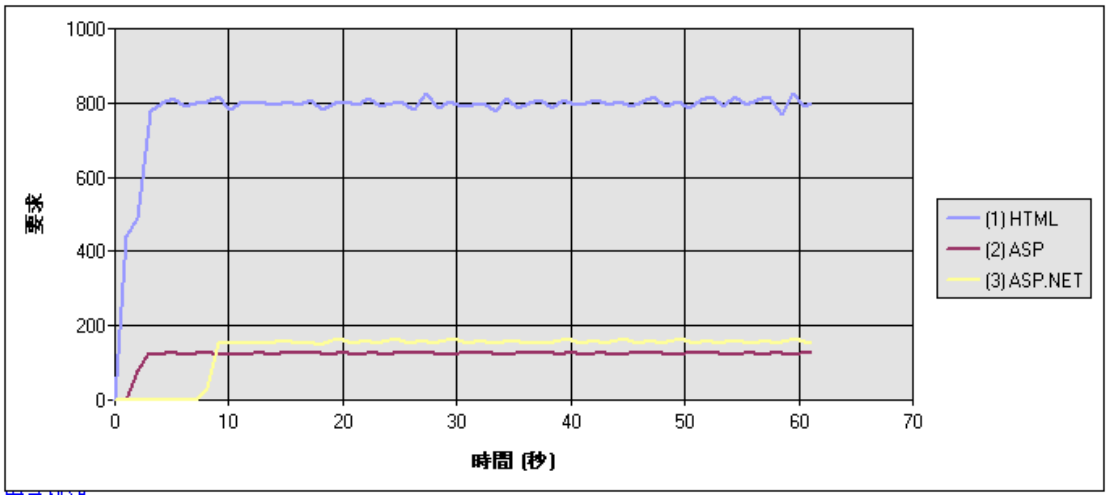

圖 2 分別以 HTML, ASP, ASP.NET 撰寫之時間顯示程式效能比較圖

由圖 2，我們可以明顯的看出，單純的 HTML 比 ASP 或 ASP.NET 都快上許多，因此針對超熱門的網站，將動態內容事先 Cache 成靜態 HTML，在效能上將有絕對的幫忙。第二點值得注意的是 ASP.NET 因爲 Just-In-Time Compile 的設計，第一次啓動時需要較長的時間，但之後即發揮編譯相較於直譯的優勢，以些微的差距勝過 ASP。

ADO, ADO.NET 正面對決

雖然由前面的超簡單程式實驗已得到 ASP.NET 較快的結果，不過話又說回來，諸位使用 ASP/ASP.NET 絕非只是爲了在沒戴手錶時能查查時間罷了。大部分 ASP/ASP.NET 的戰場多在於與資料庫的整合上，做爲資料新增、修改、刪除的 UI 呈現層。因此我們將實驗回歸到現實面，使用 ASP/ADO 與 ASP.NET/ADO.NET 進行資料庫查詢，並以 HTML Table 方式呈現。因只是示範目的，在此就直接使用 SQL Pubs 資料庫中的 Authors 資料表，簡單的將的所有作者的作者 ID、姓名、電話、地址，以表格方式列成清單。

ASP 的程式碼如下：

```
<% Option Explicit %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 7.0">
<TITLE></TITLE>
</HEAD>
<BODY>
<table border=1>
<tr>
<th>作者 ID</th>
<th>姓名</th>
<th>電話</th>
<th>地址</th>
</tr>
<%
    Dim rsX,sTemp
    Set rsX=GetRS("select au_id,au_fname+' '+au_lname as
au_name,phone,address from authors")
    While Not rsX.EOF
        sTemp=sTemp & "<tr><td>" & rsX("au_id") & "</td><td>" &
rsX("au_name") & "</td><td>" & rsX("phone") & "</td><td>" &
rsX("address") & "</td></tr>" & vbCrLf
        rsX.MoveNext
    Wend
    Response.Write sTemp

    Function GetRS(SQL)
        Const SQLCnnStr="Provider=SQLOLEDB;Data Source=127.0.0.1;User
ID=demo;Password=benchmark;Initial Catalog=pubs"
        Dim cnX,rsX
        Set cnX=Server.CreateObject("ADODB.Connection")
        Set rsX=Server.CreateObject("ADODB.Recordset")

        rsX.CursorLocation=ADODB.adUseClient
        cnX.Open SQLCnnStr
```

```
        rsX.Open SQL, cnX, ADODB.adOpenForwardOnly,
ADODB.adLockBatchOptimistic
        Set rsX.ActiveConnection=Nothing
        cnX.Close
        Set cnX=Nothing
        Set GetRS=rsX
    End Function


%>
</table>
</BODY>
</HTML>
```

而改寫爲 ASP.NET 時，一般人應該不只是 ASPX 檔中，照著 ASP 的寫法依樣畫葫蘆，應該要善用 DataGrid 這等好東西，三兩下就將資料庫呈現介面給做出來。因此我們選擇的作法是，在頁面上插入一個 DataGrid 物件，接著在 Page_Load Event 中，以 ADO.NET 的物件連上 SQL 取回 DataSet，餵入 DataGrid Object 即可，連一行 HTML Code 都不沾手，好用到令人動容。

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Put user code to initialize the page here
    SqlConnection myConnection = new SqlConnection(SQLCnnStr);
    SqlCommand myCommand = new SqlCommand("select au_id,au_fname+'
'+au_lname as au_name,phone,address from authors", myConnection);
    SqlDataReader myReader;
    myCommand.CommandType = CommandType.Text;
    myConnection.Open();
    myReader =
myCommand.ExecuteReader(CommandBehavior.CloseConnection);
    DataGrid1.DataSource=myReader;
    DataGrid1.DataBind();
}
```

在我們的測試中，爲了簡便起見，SQL 安裝於 IIS 同一台上。觀察測試期間，ASP 或 ASP.NET 的壓力測試期，伺服器的 CPU Load 都會站上 100%，其中約 70%來自於 ASP 或 ASP.NET 的 Thread，20%來自於 IIS 本身(InetInfo.exe)，SQL 平均保持在 5%-10%，研判不致成爲瓶頸。因此，就採取 SQL 與 IIS 同台的方式，進行後續的測試。

針對這兩隻程式，測試結果如圖 3。由圖中可看出，ASP.NET 再一次的發揮出後來居上的功力，與 ASP 拉出相當的效能差距。以一分鐘內所處理的 Request 數，大約是 3974:2919，快了約 37%。

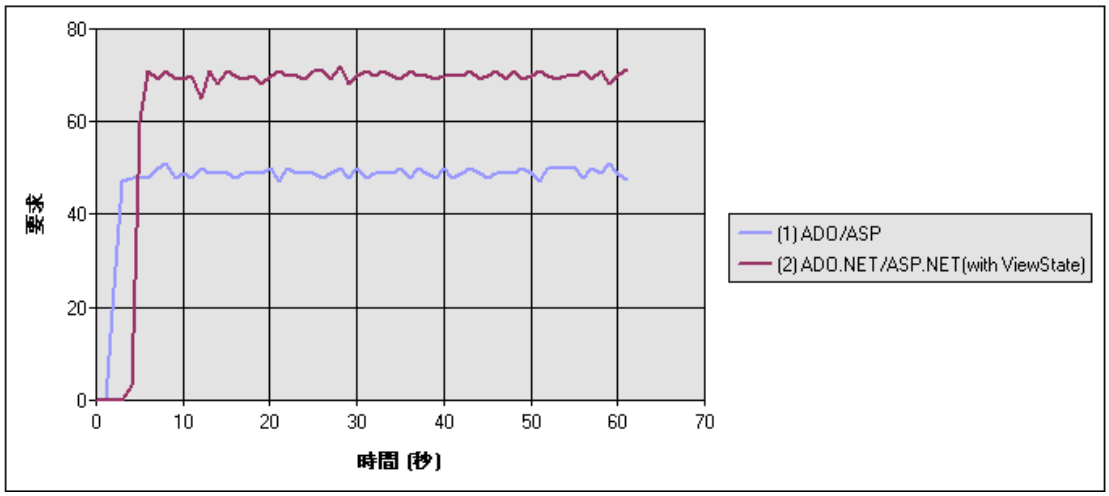

圖 3 ADO 與 ADO.NET 程式效能比較

更上一層樓

得到了 ASP.NET 確實比 ASP 快的結論後，讓我們再深入探討由現在的結果中是否能再找到更上一層樓的空間？細看 ADO/ADO.NET 的測試結果，關於資料的傳輸量比較如圖 4 所示。

		要求	內容長度 (位元組)		等待第一個回應位元組的時間 (毫秒)		等待最後一個回應位元組的時間 (毫秒)	
		總計	平均	標準裝置	平均	標準裝置	平均	標準裝置
GET 192.168.10.161/benchmark/ado.asp								
	(1)	2,912	2,703.00	0.00	100.42	30.26	100.52	30.25
	(2)	-	-	-	-	-	-	-
GET 192.168.10.161/benchmark/adonet.aspx								
	(1)	-	-	-	-	-	-	-
	(2)	3,974	9,455.00	0.00	73.35	87.41	73.49	89.00

圖 4 ADO, ADO.NET 資料內容長度比較

圖 4 是 ACT 所提供的 Request 統計，令人驚奇的發現，ADONET.ASPX 所傳送至客戶端的資料，即使以網頁的角度來看與 ADO.ASP 相去不遠，但網頁的長度卻足足是 ASP 的三倍有餘。再深究其原因，會發現用來保存 Web Control 屬性的 VIEWSTATE Hidden 欄位是最大的元凶，當 Web Control 內保存的資料、狀態屬性愈多，長度愈長。而且因爲其爲 INPUT 欄位之一，這些資料會在每次 Web Control 的操作中，都會在 Client-Server 之間來回傳送，殺死不少頻寬。而 VIEWSTATE 的選項預設是開啓的，但卻非所有的情境中都會用到。因此我們修改 ADONET.ASPX 的內容，在 DataGrid 的 Tag 中加上 EnableViewState="False"

```
<asp:DataGrid id="DataGrid1" style="Z-INDEX: 101; LEFT: 10px; POSITION: absolute; TOP: 12px" runat="server" AutoGenerateColumns="False" EnableViewState="False">
```

再測試一次的結果如圖 5, 6 所示，關閉 DataGrid 的 ViewState 後，整體 HTML 長度縮爲 3271 個位元，與 ADO.ASP 的結果接近，一分鐘所處理的 Request 數，也多了近 900 個，證明了關閉不必要的 Web Control ViewState，在效能上可得到可觀的提升，在我們的測試結果中，比開啓時快了約 26%。

	要求	內容長度 (位元組)		等待第一個回應位元組的時間 (毫秒)		等待最後一個回應位元組的時間 (毫秒)	
	總計	平均	標準裝置	平均	標準裝置	平均	標準裝置
GET 192.168.10.161/benchmark/adonet.aspx							
(1)	3,974	9,455.00	0.00	73.35	87.41	73.49	89.00
(2)	5,027	3,271.00	0.00	57.75	74.36	57.87	75.94

圖 5 ViewState 關閉後的效能統計，(2)爲關閉後的測試結果

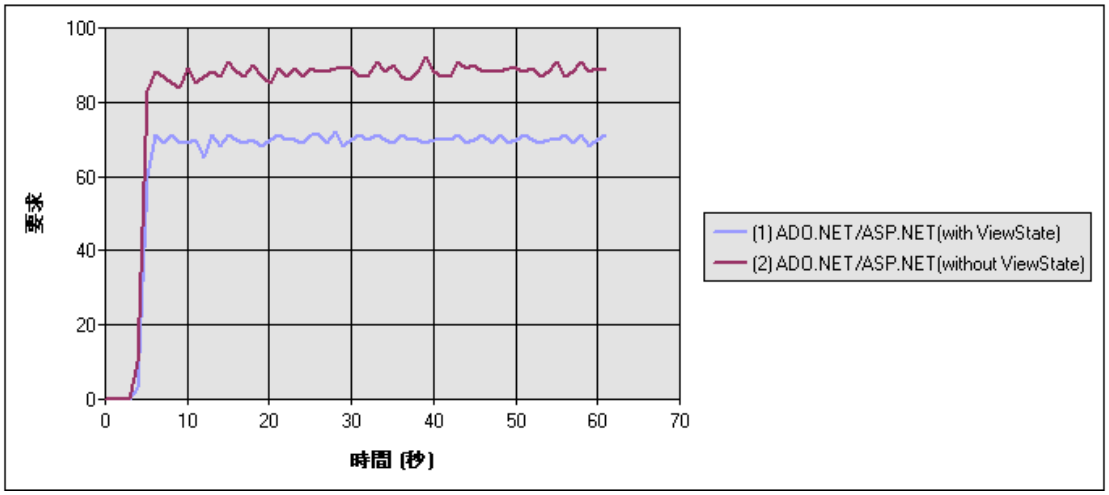

圖 6 DataGrid ViewState 關閉前後的效能比較圖

結論

在本文中，介紹了 VS.NET 所附 ACT 壓力測試工具的使用，使用 ACT 得到 ASP.NET 比 ASP 快 30%的實測結果，透過 ViewState 屬性的調整，小小的體驗了一下對 ASP.NET 做校能調整(Performance Tuning)的樂趣。對於校能調整這門深奧且充滿挑戰性的學問，掌握了壓力測試工具，就如取得開啓大門的鎖鑰，門後的世界有無限的空間可以探索發掘。