被遺忘的文字利器–探索.NET Regular Expression 元件

作者 李明儒

試想以下的情境：

- 公司最近要研發一套類似 Google 的網頁搜索引擎，你負責開發網頁擷取器核心，其中最棘手的部分是要具備由一個網頁再延伸至其所超連結出去的其他網頁…
- 行銷人員拿來一份十萬筆客戶地址檔，請你解析出縣市、郵遞區號、地址三個欄位轉入 CRM 系統，但要命的是原始資料雜亂不已，有的郵遞區號在前，有的在中間…
- 你加入了檢索引擎過濾器(Filter)的開發團隊，目標是將各式資料檔中的純文字部份提取出來，你的第一件任務是簡單的 HTML 檔案解析，但是，HTML 原始碼千變萬化…

以上的挑戰當然不是每個程式開發者都有機會遭遇，但應該沒有人會否認能寫出這等程式的傢伙還真有兩把刷子。本文將會介紹.NET 中一群好用但常被遺忘的類別--Regular Expression，如何讓文字處理程式的發展工作單純一點、輕鬆一些，同時也會試著使用.NET Regular Expression 建立一個實例程式的雛型。

文字處理一直是 Web 程式開發時的重點工作，因爲網頁的內容、使用者所輸入的資料，無一不是以文字爲主體。格式嚴謹的文字資料，例如：XML、CSV 等，可使用簡單的函數加以解析。但開發者仍不時得面對自由格式的情境，尤其是不透過自己程式 UI 而得的其他資料來源，就無法在輸入時要求其符合內定的格式需求。面對這種格式鬆散的字串，要如何由其中擷取我們所需的資料，得建立彈性高的複雜演算法來解析(Parse)字串內容，而在.NET 中，也支援此種能模糊(Fuzzy)處理的元件–Regular Expression，本文便將介紹 Regular Expression 在文字處理上的應用。

Regular Expression 的歷史

Regular Expression，有人譯爲字串樣版或規則表達式，不過似乎並無統一的中譯，本文將以原文爲主。Regular Expression 的基本精神，其實可以回溯到大家早已習以爲常的 Wildcard 字元*、?。相信各位肯定有在 DOS、UNIX 使用 Wildcard Character 的經驗，就是 abc*.txt、*.*這種廣義的檔名限定法。這種以特定字符去限定某一群對象的觀念，再加以延伸強化之後，就可以巧妙的組合出恰到好處的 Filter，將符合需求的字串由一長串複雜的字串中提取出來。

可以想見，Regular Expression 當然不會是.NET 所創新發展出來的概念。早在 UNIX 系統中就出現了 sde、awk、grep 等支援 Regular Expression 的工具程式，而其開始大發光芒則是在 Web 初試啼聲的年代。當時的動態網頁程式大多依賴所謂的 CGI(Common Gateway Interface)程式來產生，並且 Web 主機也多以 UNIX 系統爲主。因此在最早期，有人用 C++撰寫 CGI 程式、有人用 Shell 指令檔撰寫，直到 Perl 語言出現在江湖上，以其簡潔易學的語法與強大的文字處理能力(在那個年代，每一行 HTML 語法都得由自己拼湊，可以想見文字處理在 CGI 程式中所佔的分量)，讓許多開發者驚豔，放下難學難寫的 C++與功能簡陋的 Shell Script，改用 Perl 開發出各式各樣的網頁應用程式，而讓 Perl 語言擁有一段風光盛世。

直至今日，即使 ASP、Servlet、JSP、.NET、J2EE 等新一代的動態網頁開發技術百鳥爭鳴，Perl 逐漸失去往日的風光，但它仍不斷演進，各式免費的網路、資料庫應用模組日益齊全，同時也仍保有相當的死忠支持者(或許不久之後就可見到 Perl.NET 的問市)。熟悉 Windows 環境的讀者如在 UNIX 平台上，需要一個易上手

的開發工具時，Perl 也是個不錯的選擇。

回到正題，Perl 之所以能在文字處理上呼風喚雨，主要是因爲其充分發揮了 Regular Expression 的優勢，將 Regular Expression 與語法核心緊密的結合，幾行精簡的指令就完成一件驚天動地的文字處理工程！好東西是不會寂寞的，Regular Expression 的觀念也被廣泛的應用在各式的語言與工具中，Java、JavaScript、UNIX 的工具程式、甚至廣受歡迎的文字工具 UltraEdit 都可以發現其蹤跡。不過看來 VB、VBScript 倒是在這點上交了白卷，坊間介紹 ASP、VB 的書也甚少探討此種進階的文字處理技巧。因此，看到許多 VB 門派的開發者在面對複雜的文字處理，往往雙手一攤，而高段一點的則會埋首苦幹，用 IF、THEN、OR、AND、FOR、NEXT 堆砌出上百行的程式碼，寫出各式可歌可泣的 Parser。事實上，在 VB、VBScript 時代，也不難找到一些免費的 Regular Expression 元件，再一次站在巨人的肩上，享受一群程式專家所精心開發的心血結晶。而在.NET 中，在.NET Framework 中就具備 Regular Expression 相關的工具元件，不必外求，對.NET 開發者來說，的確是個大好消息。只是延續歷史的軌跡，許多人都忽略了.NET Framework 的角落埋藏了這麼一付神兵利器，在面對複雜的文字處理時，還是束手無策，不然就是忙著土法鍊鋼。今天我們就要花點時間，來探索一下這個常被遺忘的文字利器–Regular Expression。

初識 Regular Expression

我們都知道，程式是很死板的東西。IF 判斷，是就是，不是就不是，如果要求彈性，就得加上一段 OR、AND 的複雜組合。如果還想加上限定文字出現數目，就不免得加上計數器、迴圈等機制。先舉一個例子，我們由某處獲得一份姓名與電話的清單，但當初在輸入介面上未做任何限制，開放給使用者自由填寫的結果，出現了各式的組合，例如：02-23939889、(02)23939889、0933123123、0933-123-123 千奇百怪。在資料庫的俚語裡，有個很傳神的形容，通常會稱這份資料很"髒"(Dirty)，要經一番清洗整理後才好使用。如今，這個任務落到了你身上，老板要求你把這些電話資料整理一下，市話一律改成(02)3939889 的標準格式，而行動電話則要識別出來存在另一個欄位，我們來看看這個說難不難，說簡單不簡單的程式要如何來寫。

這裡以 C#爲例，使用 VB 的朋友應可輕易改寫之。

```
        private void WashTelNumList()
        {
            //用陣列儲存要測試的資料
            string[] aryTelNum={"02-23939889","(02)23939889","02 23939889",
"0933123123", "0933-123-123" };
            for (int i=0; i<aryTelNum.Length; i++)
            {
                MessageBox.Show(ParsePhoneNum(aryTelNum[i]));
            }
        }

        //解析電話號碼, 前面加上M表行動, C表市話, U表無法識別
        private string ParsePhoneNum(string PhoneNumber)
        {
            bool bMobil=true, bCityLine=true;
            //09x開頭, 且10碼數字表示爲行動
            //先去除-及任何空白符號
            string sMobilTest=PhoneNumber.Replace("-","");
            sMobilTest=sMobilTest.Replace(" ","");
            //檢查是否爲10碼數字, 且前二碼爲09
            if (sMobilTest.Length!=10) bMobil=false;
            //在Reference中加上Visual Basic .NET Runtime, C#也可以用VB的函數, 夠酷吧!!
            if (bMobil
```

```
&& !Microsoft.VisualBasic.Information.IsNumeric(sMobilTest)) bMobil=false;
            if (sMobilTest.Substring(0,2) != "09") bMobil=false;
            if (bMobil) return "M"+sMobilTest;

            //檢查是否爲含區域號碼的市話號碼?
            string sCLTest=PhoneNumber.Trim(); //去頭尾空白
            string sZonePart=""; //區碼部分
            string sPhonePart=""; //電話號碼部分
            //先排除以空白分隔區碼及電話的情形
            if (sCLTest.IndexOf(" ")==2 || sCLTest.IndexOf(" ")==3)
            {
                int i=sCLTest.IndexOf(" ");
                sZonePart=sCLTest.Substring(0,i);
                sPhonePart=sCLTest.Substring(i+1);
            }
            else
            {
                //檢查是否有)或-?
                if (sCLTest.IndexOf("-")>0 || sCLTest.IndexOf(")")>0)
                {
                    //取出區域碼
                    int i=sCLTest.IndexOf(")");
                    if (i>0)
                    {
                        //取出)之前的部分並去掉(
                        sZonePart=sCLTest.Substring(0,i).Replace("(","");
                        sPhonePart=sCLTest.Substring(i+1);
                    }
                    else //使用-分隔區碼與電話碼的情形
                    {
                        i=sCLTest.IndexOf("-");
                        sZonePart=sCLTest.Substring(0,i);
                        sPhonePart=sCLTest.Substring(i+1);
                    }
                }
            }
            sZonePart=sZonePart.Trim(); //區域去空白
            sPhonePart=sPhonePart.Replace(" ",""); //電話號碼部分去除空白及-
            sPhonePart=sPhonePart.Replace("-","");
            //檢查區碼是否爲2-3碼純數字且第一碼爲0
            if (sZonePart.Length<2 || sZonePart.Length>3 ||
sZonePart.Substring(0,1)!="0"
|| !Microsoft.VisualBasic.Information.IsNumeric(sZonePart)) bCityLine=false;
            //檢查電話號碼部分是否爲8或7碼純數字
            if (sPhonePart.Length<7 || sPhonePart.Length>8
|| !Microsoft.VisualBasic.Information.IsNumeric(sPhonePart)) bCityLine=false;
            if (bCityLine) return "C("+sZonePart+")"+sPhonePart;
            return "U"+sCLTest;

        }
```

註：上述的程式碼中表演了一招密技--在 C#中使用了 VB.NET 的 IsNumeric 函數。方法是新增一個參照(Reference)到 Microsoft Visual Basic.NET Runtime，你就可以使用 Microsoft.VisualBasic 一系列命名空間的函數。於是在 C#中也可與 IsDate、IsNumeric、Split 等熟悉的好用 VB 函數喜相逢，對由 VB 轉戰 C#的朋友來說應是一個不錯的選擇。

這裡只舉了五種不同的格式變化，當要因應更多的格式變化時，程式演算法會更複雜、程式碼也會更多。

筆者過去在處理文字上有些經驗，對文字處理常用的函數與技巧也算熟悉，所以程式寫來還算簡短。對新手來說，有可能卡在某個環節而無以爲繼，再不然就是用較瑣碎的演算法處理，程式碼又可能再長上一大截。

的確，每次針對不同的格式，就得重新量身打造解析文字內容的程式碼，實在是件苦差事；而電話號碼分析還算是比較單純的例子，有些彈性要求更高的案例中，解析程式碼甚至會多達數百上千行。於是，就有了一個構想，何妨將常用的解析技巧提取出來，將這些程式碼做成一個引擎，解析時要用的規則則簡化成一些特殊的語法。則日後，每次要解析不同的文字格式時，只要重新編寫規則語句就好了，於是，Regular Expression 就誕生了!!

在詳細介紹 Regular Expression 之前，讓我們先見識一下它的威力，來產生我們立志學好它的原動力。現在，我們改用.NET 的 Regex 元件重寫上面的程式範例。

```
        private void WashTelNumList()
        {
            //用陣列儲存要測試的資料
            string[] aryTelNum={"02-23939889","(02)23939889","02 23939889",
"0933123123", "0933-123-123"};
            for (int i=0; i<aryTelNum.Length; i++)
            {
                MessageBox.Show(ParsePhoneNumRegEx(aryTelNum[i]));
            }
        }

        //解析電話號碼, 前面加上M表行動, C表市話, U表無法識別
        private string ParsePhoneNumRegEx(string PhoneNumber)
        {
            //使用Regex前要記得using System.Text.RegularExpressions
            if (Regex.Match(PhoneNumber, @"09\d\d-?\d{3}-?\d{3}").Success)
            {
                if (PhoneNumber.Replace("-","").Length != 10) return
"U"+PhoneNumber;
                return Regex.Replace(PhoneNumber,
@"09(?<p1>\d\d)-?(?<p2>\d{3})-?(?<p3>\d{3})", "M09${p1}${p2}${p3}");
            }
            if (Regex.Match(PhoneNumber,
@"[(]*\d{2,3}[-)\s]\d{2,4}-?\d{4}").Success)
            {
                string sTemp=Regex.Replace(PhoneNumber,
@"[(]*(?<zone>\d{2,3})[-)\s](?<p1>\d{2,4})-?(?<p2>\d{4})",
"(${zone})${p1}${p2}");
                if (!Regex.Match(sTemp, @"[(]\d{2,3}[)]\d{7,8}").Success)
                    return "U"+sTemp;
                else
                    return "C"+sTemp;
            }
            return "U"+PhoneNumber;
        }
```

同樣的程式結果，大約十行就搞定，酷吧!! 然而其中出現了一堆類似天書般的文字，就是剛才提到每次自定比對原則的特殊語法，乍看之下不知所云，其實並不難學。接下來，我們就要來學寫這種威力強大的天書。

.NET Regular Expression Pattern

不同語言裡的 Regular Expression 會有些許不同，不過基本的幾個元素都相近就是了。這裡介紹幾個常用的元素，其餘的語法讀者可在.NET Framework SDK Reference 中找到詳細的介紹(在 Regular

Expression Language Elements 一節)：

1. 比對字串中如有用到．$ ^ { [(|) * + ? \等字串，要記得用方括號夾起來[]。例如：要比對 NT$，要寫成 NT[$]。
2. 比較常用的特殊字元有\r、\n、\t(Tab)等。
3. 最主要的比對符號如下：

字元符號範例	代表意義
.	\n 換行符號以外的所有字元，若設爲 Singleline 模式，則代表任何字元。
[aeiou]	表示方括號中的任一個字元均可以，此例表示 a、e、i、o 或 u 的任一個字元均可。
[^aeiou]	表示這五個字元以外的任何字元均可。
[0-9a-fA-F]	若是相連的字元，可使用-連字線來表示，0-9 就表示 0,1,2,3,4,5,6,7,8,9 均可。
\w	所有的文數字，英文字母、中文字及數字再加上底線字元。
\W	和\w 剛好相反，所有不是文數字的字元。
\s	所有的空白(white-space)字元，相當於空白、\f、\n、\r、\t、\v 等不會顯現的字元。
\S	與\s 相反，不屬於\s 的所有字元。
\d	所有的數字(0-9)。
\D	所有的非數字。

4. 比對時也可指定出現的位置，

位置符號	說明
^	出現在字串的開頭。(受 Multiline 選項影響)
$	出現在字串的結尾。(受 Multiline 選項影響)
\A	出現在整個字串的開頭。(不受 Multiline 選項影響)
\Z	出現在整個字串的結尾。(後方仍可接一個\n，不受 Multiline 選項影響)
\z	出現在整個字串的結尾。(後方不可接任何字元，不受 Multiline 選項影響)

註：所謂 Multiline 選項，主要在字串中有換行符號時會發生差別，可在呼叫函數時設定參數，而筆者則較常直接加在比對字串的前方(即所謂的 inline character)。例如：

```
string sML="Item1 is a dog.\nItem2 is a cat.\nItem3 is a mouse.\n";
    MessageBox.Show(Regex.IsMatch(sML,"^Item2").ToString());
MessageBox.Show(Regex.IsMatch(sML,"(?m)^Item2").ToString());
```

其中 Inline Character (?m)就是啓用了 Multiline 選項，所以第一個測試傳回 False，第二個則傳回 True，這個例子應該就可以清楚的解釋 Multiline 選項的影響。同時這裡再補充兩個常用的選項，i=Ignore Case 不分大小寫，s=Single Line 則決定.符號是否包含\n。

5. 比對符號可以指定出現的次數，如以下的說明：

次數修飾元	說明
*	出現 0 次或多次，相當於{0,}。
+	出現 1 次以上，相當於{1,}。

?	出現 0 次或 1 次，相當於{0,1}。
{n}	出現正好 n 次。例如：(pizza){2} -> "pizzapizza"就符合比對。
{n,}	最少出現 n 次，例如：(abc){2,}.
{n,m}	最少出現 n 次，但不超過 m 次。
*? +? ?? {n}? {n,}? {n,m}?	當有多種符合的可能時，以最少數目的爲準，例如：比對"123ABC"字串，如果用"\d*"可得 123，若用"\d*?"則得到 1，其餘的意義可類推。英文稱爲 Lazy *, Lazy +, Lazy ?...

6. 另外一個常用的是分群功能，針對將資料拆解成一個一個欄位的場合，非常有用。例如：一個在地址清單中，最前面爲縣市，接著是郵遞區號，再來才是地址，若我們要分別將縣市、郵遞區號、地址擷取出來，則分群功能就很有效，尤其是還可爲每個群組取名字。下面的例子就是利用分組功能將地址中的縣市、郵遞區號、地址等資訊給分別顯示出來。

```
        private void GroupSample()
        {
            string sSource="台北市110東興路59號6F";
            //利用(?<groupname>pattern)的語法, 將比對結果分組
            Match
match=Regex.Match(sSource,@"(?<city>\D+?)(?<zipcode>\d{3})(?<addr>.+)");
            if (match.Success)
            {
                MessageBox.Show("縣市
="+match.Groups["city"].ToString());
                MessageBox.Show("郵遞區號
="+match.Groups["zipcode"].ToString());
                MessageBox.Show("地址
="+match.Groups["addr"].ToString());
            }
        }
```

常見的應用

在筆者的使用經驗中，Regular Expression 常見於以下幾種應用方式：

1. 比對找出特定的資料是否存在

   就是指定特定資料中的格式 Pattern，交由 Regex.IsMatch Method 傳回是否在字串中發現符合格式的字元組合。例如：前一陣子引發討論的 SQL Injection 問題，除了使用將單引號置換成兩個雙引號的消極防衛，使用 Regular Expression 比對，還可以比對找出可能具備攻擊意圖的輸入，採取較主動的警告措施。不過這種單純比對而不處理的應用情境並不算多。

2. 判定輸入資料的格式是否符合規定

   這算是十分普遍而實用的應用，而 ASP.NET 中，甚至也有 RegularExpressionValidator Web Control，完全不需應用到 Regex 物件，就可以直接輸入比對 Regular Expression Pattern 作爲參數，極爲方便。若要自行開發簡單的 Script Engine 時，Regular Expression 也可作爲語法檢查(Syntax Checking)程式碼的核心。

3. 由字串中擷取特定的欄位

   這也是極爲常見的應用方式，基本的語法是用圓括號將要分群命名的部分夾起來，同時以?<groupname>加

以命名。例如：一個格式稍亂的字串中含有姓名，生日與電話號碼，利用分群比對，可以逐一將姓名、生日與電話號碼取出來，參考以下的範例：

```
        //分群擷取功能測試
        private void GroupTest()
        {
            string sSource=" Jeffrey, 1997/4/1, 0800956956  Sharon, 2001/11/4,
0933123123  Ryan 1999/12/31 (02)87682688";
            foreach (Match match in
Regex.Matches(sSource,@"(?<name>[^ ,]+)[ ,]+(?<date>[0-9/]+)[ ,]+(?<tel>[0-9(
)]+)"))
            {
                //將結果輸出在Debug的輸出視窗中
                System.Diagnostics.Debug.WriteLine("Name=" +
match.Groups["name"].ToString() + " Date="+match.Groups["date"].ToString() + "
TelNo=" + match.Groups["tel"].ToString());
            }
        }
```

另外一個非常有用的地方是用來解析網頁內容，從中取出有用的部分。大致的方法，就是找出你所要內容前後 TAG 的特徵，例如：Regex.Match(sHTMLCode,"(?is)<title>(?<title>.+?)</title>") 就可以輕易取出 HTML 文件的 TITLE，而且<TITLE> Tag 的大小寫不受限(i 選項)、寫成兩行也能被接受(s 選項)。

4. 內容更換

String.Replace Method 可以用來置換特定的字元，而 Regex.Replace 則更上一層樓，可使用更彈性的比對，更厲害的是要更換的新字串中還可加入比對出來分群組果。舉個更實際點的例子，在一篇文章中，我們想將所有出現的電子郵件地址，在前後加上與的 HTML Tag，使用以下這行指令即可，夠簡單吧!!

```
sArticle=Regex.Replace(sArticle,"(?<mail>[a-zA-z0-9_.]+@[a-zA-Z0-9.-]+)","<a
href=\"mailto:${mail}\">${mail}</a>");
```

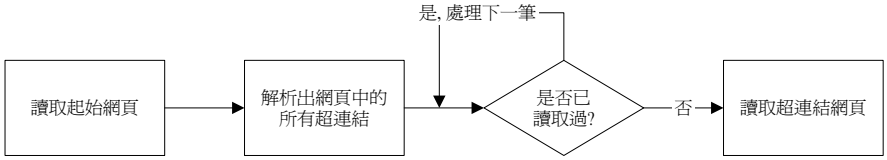
另外，置換字串中的${groupname}可以將比對的結果再代入要更換成的新字串中，應用起來就千變萬化了！

應用實例—網頁連鎖擷取

在了解 Regular Expression 的妙用之後，想必各位一定已經迫不及待想動手解決過去懸置已久的文字處理難題了吧!! 而這裡我們就以前言中所提及的網頁擷取需求爲實例，讓各位進一步體會 Regular Expression 的威力。

簡單來說，網頁連鎖擷取的重點就是要由一個網頁找出其超連結出去的其他相關網頁內容，最基本的作法就是由 HTML Code 中找出<a href="…">的 Tag。而在此順道介紹另一個技術，就是由程式去讀取網頁。過去應該不少人是使用 MSXML 中的 XMLHttpRequest 物件，而在.NET 的基礎類別中，可以找到 System.Net 下有個 HttpWebRequest 類別具有相同的能力。

我們先歸納出程式的處理邏輯，如以下的流程圖：



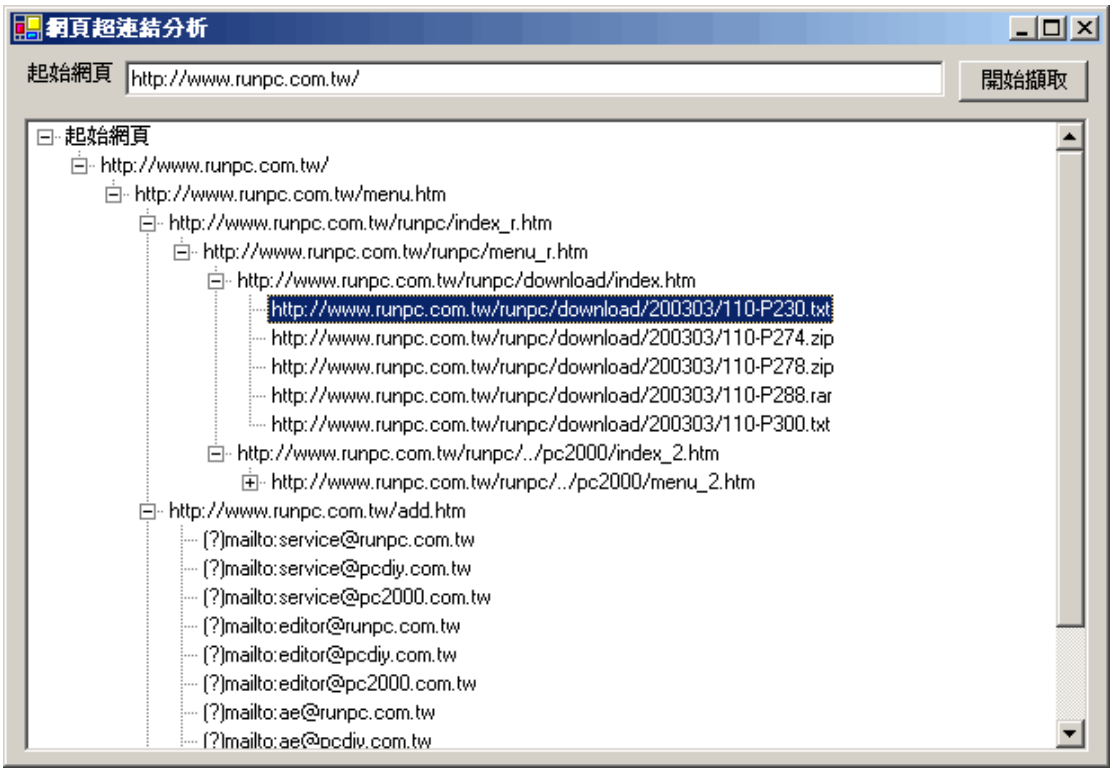
在流程中，讀取超連結出去的網頁後，又需對其中的超連結再作解析，而我們使用 Recursive 的概念讓程式碼簡潔一點。至於防止重覆讀取的機制，則使用 ArrayList 來模擬即可，如要提升效能與適用規模，則可改用資料庫來管理。

以下的程式碼，示範如何呼叫一個函數，以某個 URL 作爲起始點，追蹤其所超連結出去的諸多網頁。爲了展現網頁間的關聯，這裡用了 TreeView Control 來概略表現網頁間的連結關係。

```
        private ArrayList aryLinks=new ArrayList();
        private string BaseURL="";

                        … Windows Forms固定程式碼部分省略 …

        private void btnOK_Click(object sender, System.EventArgs e)
        {
            //取出http://.../ 的部分作爲BaseURL
            try
            {
                BaseURL=Regex.Match(txtURL.Text,
"(?i)(?<baseurl>http://[^/]+?/)").Groups["baseurl"].ToString();
            }
            catch
            {
                MessageBox.Show("起始網頁之URL格式應爲http://www.xxx.com/!");
                return;
            }
            const string ROOT="起始網頁";
            treeView1.Nodes.Clear();
            TreeNode RootNode=new TreeNode(ROOT);
            treeView1.Nodes.Add(RootNode);
            AnalyzeHyperlink(txtURL.Text,RootNode);
        }

        //此一函數可被遞迴呼叫
        private void AnalyzeHyperlink(string URL,TreeNode ParentNode)
        {
            if (URL.Substring(0,1)=="/") //絕對路徑由BaseURL起算
            {
                URL=BaseURL+URL;
            }
            else if (URL.IndexOf(":")<0) //排除mailto: http: ftp:
javascript: ..., 其餘採相對路徑, 由Parent的所在位址起算
            {
                URL=Regex.Match(ParentNode.Text,
"(?i)(?<baseurl>http://.+/)").Groups["baseurl"].ToString()+URL;
            }
            string HTMLCode="";
            if (URL.Substring(0,5).ToLower()=="http:")
            {
                HTMLCode=SketchWebPage(URL);
                if (HTMLCode=="FAILED") URL="(X)"+URL; //若無法取回網頁則在Text
上標註(X)
            }
            else //非HTTP連結標註(?)
            {
                URL="(?)"+URL;
            }
            TreeNode NowNode=new TreeNode(URL);
            ParentNode.Nodes.Add(NowNode);
```

```
            foreach (Match match in Regex.Matches(HTMLCode,
"(?i)[<](?<tag>\\w+)\\s[^>]*?(?<name>src|href)=['\"](?<url>.+?)[\"'].*?[>]"))
            {
                string HTMLTag=match.Groups["tag"].ToString().ToUpper();
                if (!(HTMLTag=="A" || HTMLTag=="FRAME")) continue; //只限定FRAME
與A Tag, 可視需要增刪
                string Hyperlink=match.Groups["url"].ToString();
                if (Regex.IsMatch(Hyperlink,"(?i)http://")) continue; //超連至
其他網站者不處理
                if (aryLinks.Contains(Hyperlink)) continue; //已處理過的就跳過
                aryLinks.Add(Hyperlink);
                AnalyzeHyperlink(Hyperlink,NowNode);
                treeView1.Update();
            }

        }

        //不錯用的函數, 給URL就傳回網頁內容
        private string SketchWebPage(string URL)
        {
            HttpWebRequest reqPage=(HttpWebRequest) WebRequest.Create(URL);
            try
            {
                HttpWebResponse rspPage=(HttpWebResponse)
reqPage.GetResponse();
                Stream stmPage=rspPage.GetResponseStream();
                StreamReader srPage=new
StreamReader(stmPage,System.Text.Encoding.GetEncoding("big5"));
                string sTemp=srPage.ReadToEnd();
                rspPage.Close();
                stmPage.Close();
                srPage.Close();
                return sTemp;
            }
            catch
            {
                return "FAILED";
            }

        }
```

程式執行結果如下圖所示:



有些讀者或許已想到了這個程式的進一步應用。沒錯，它可以改寫成將網站的內容複製到本機硬碟上，達到離線瀏覽的目的。但上述的程式碼只提供了基本的功能，要做到離線瀏覽複本的擷取，還會涉及圖檔與附件檔的存檔與重新命名、SRC、HREF 連結的修改等等進一步的議題。而這些工作的基本原理都大致相同，不外乎由 HTML Tag 中找出連結的圖片、檔案與網頁，將其取回另存新檔後，再將原有的 HTML Tag 更改爲連結至新檔名。依此來看，Regular Expression 還是可以提供極大的幫助。

好用的 Regular Expression 測試器

Regular Expression 的 Pattern 語法規則簡單，應用起來卻千變萬化，巧妙各有不同，但要組合出精確而簡潔的 Pattern，需要經驗，有時更像場智力測驗。依筆者經驗，在撰寫規則 Pattern 時，甚少能一次 OK。例如：原始字串中目視有 10 個符合處，第一次可能因規則太過寬鬆而錯抓爲 12 處，修改之後又因爲太嚴格而只得到 8 個。就像這樣，常常要反覆修改多次，才能剛好切中所有要鎖定的對象。且在往來處理各式資料的過程中，也常會發現原先未預期到的資料格式，這時就需要重新調整比對規則。

此一過程有點像打靶時的表尺校正(或稱爲歸零射擊，好個讓人懷念的名詞!)，往往不外乎加一個?號抓到 12 個，將.改成\w 會變成 8 個之類的細部調整測試。因此，若能有一個方便的操作介面，能讓你即刻看到調整後的結果，而不需歷經"修改->Compile->測試"的冗長步驟，肯定能大大縮短在調校測試上所花的時間。

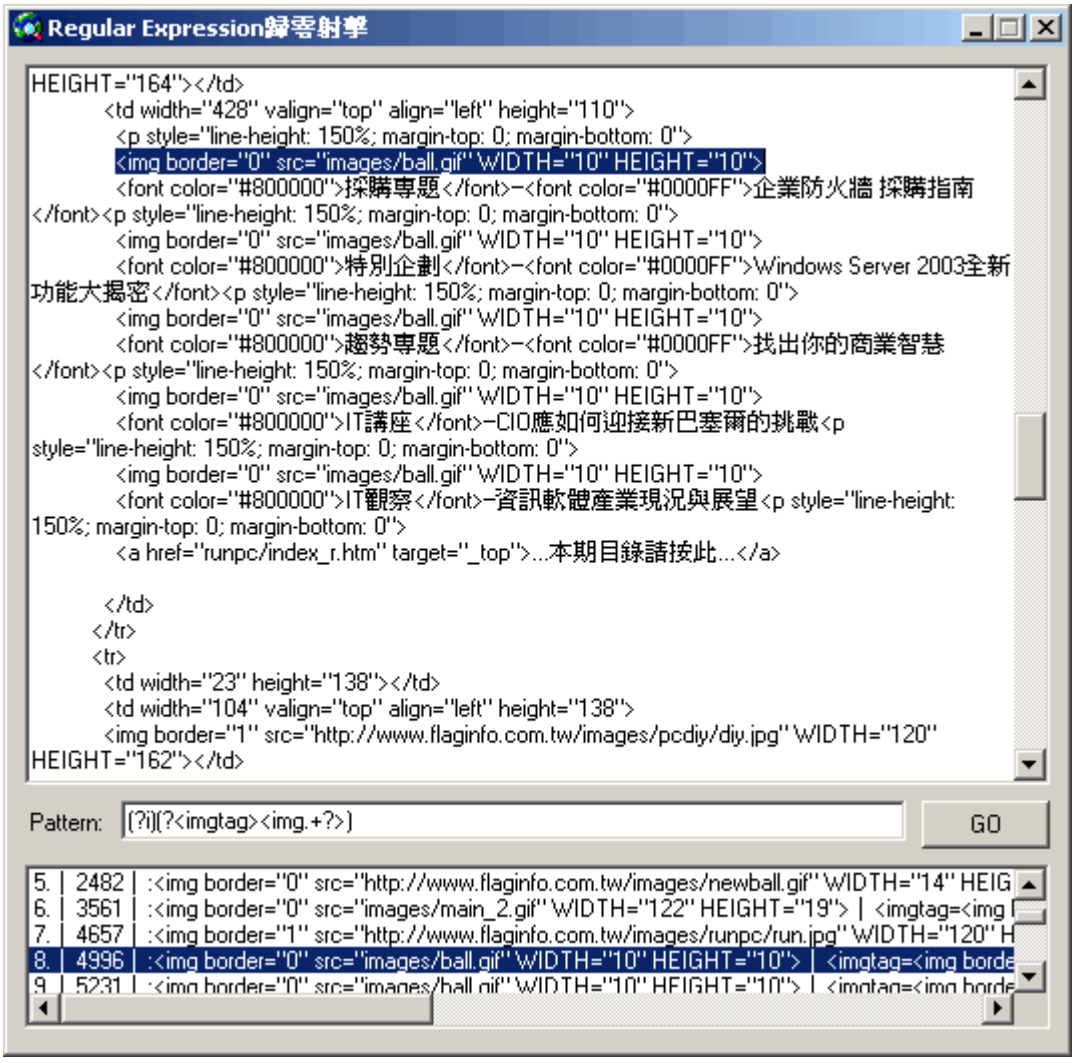
需要爲發明之母，因而筆者寫了一個 Regular Expression"歸零射擊"測試器，構造非常簡單，由一個輸入原始字串的多行 TextBox、一個輸入 Pattern 的單行 TextBox、一個顯示符合結果的 ListBox，加上一顆執行鈕就 OK 了。而程式結構也十分單純，按下執行鈕時，用 Regex.Matches 將原始字串以使用者輸作的 Pattern 作比對，結果再逐一輸出到 ListBox 就好了。爲了讓它更好用些，當使用者點選某一行符合結果時，會自動將原始字串中標出其所在位置。千萬別覺得它長相不起眼，程式碼亦不過短短幾行。靠著這個簡陋卻實用的工具，筆者在學習與應用.NET Regular Expression 的過程中，倒是省卻了可觀的摸索測試時間呢!

```
using System;
using System.Drawing;
```

```
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Text.RegularExpressions;

namespace RegexTester
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.TextBox txtInput;
        private System.Windows.Forms.TextBox txtPattern;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.ListBox lstResult;
        private System.Windows.Forms.Label label1;

                            …Windows Form之通用程式碼部分省略…

        private void button1_Click(object sender, System.EventArgs e)
        {
            try
            {
                //先使用Regex解析Pattern是否有定義Group
                ArrayList aList=new ArrayList();
                foreach (Match m in Regex.Matches(txtPattern.Text,
"[(][?][<](?<groupname>.+?)[>]"))
                {
                    aList.Add(m.Groups["groupname"].Value); //將出現的Group名稱放入
ArrayList中
                }
                lstResult.Items.Clear();
                int i=0;
                foreach (Match m in Regex.Matches(txtInput.Text,txtPattern.Text))
                {
                    i++;
                    string
sResult=i.ToString()+". |"+m.Index.ToString()+" |:"+m.Groups[0].Value;
                    if (m.Groups.Count>1)
                    {
                        for (int j=0; j<aList.Count; j++)
                        {

    sResult+=" |<"+aList[j].ToString()+"="+m.Groups[aList[j].ToString()].Value+">
";
                        }
                    }
                    lstResult.Items.Add(sResult);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("發生錯誤: "+ex.Message);
            }
        }

        //爲加強互動性，使用者點選某筆結果時，可映對出在原文中的位置，並形成反白
```

```
        private void lstResult_SelectedIndexChanged(object sender,
System.EventArgs e)
        {
            string sTemp=lstResult.Items[lstResult.SelectedIndex].ToString();
            string[] aAnalysis=Regex.Split(sTemp," |");
            int StartPos=int.Parse(aAnalysis[1]);
            txtInput.Focus();
            txtInput.Select(StartPos,aAnalysis[2].Length-1);
            txtInput.ScrollToCaret();
        }

    }
}
```

程式執行畫面如下圖所示：



結語

在本文中，我們介紹了.NET Regular Expression 的概念與語法，並舉了一些應用的範例。同時，我們也以 Regular Expression 爲核心完成了一個能順著超連結延伸到相關網頁的網頁連鎖擷取器。最後，筆者也分享了在學習應用 Regular Expression 的過程中，一個簡單實用的 Regular Expression 語法測試工具。希

望透過此番介紹，讀者對.NET 的 Regular Expression 功能與應用能有初步的認識，未來在遭遇進階的文字處理時，能善用它的力量，解決掉各式複雜的文字迷宮挑戰，讓這個常被遺忘的文字利器不再蒙塵!