Chapter 3

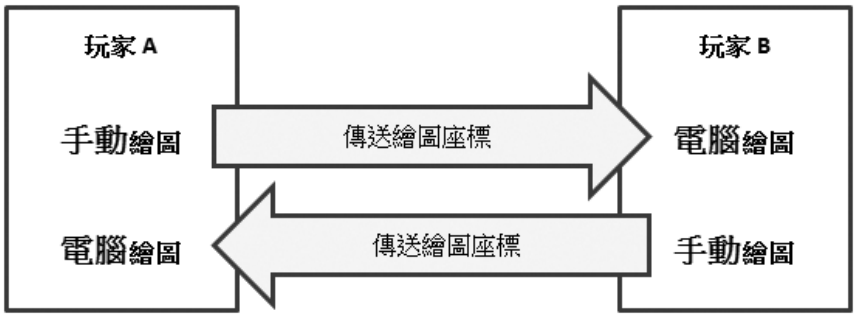
UDP 塗鴉牆

CHECK POINT!

3-1 網路互動繪圖的通訊流程

本章將介紹一個可以經過UDP網路通訊模式互動塗鴉的程式，讀者在表單上所畫的任何圖案都會顯示於另一位玩家的視窗，且對方的塗鴨也會出現在讀者的表單上，有如共同創作一個畫面。單機版的塗鴉程式(如小畫家)，主要的技巧就是使用MouseDown事件開始繪圖，然後在持續的MouseMove事件之中繪製連續的小線段來形成曲線，最後在MouseUp事件時結束一個筆畫。

要讓程式能將近端的繪圖結果顯示在遠端的程式，當然可以互相傳遞已經繪製完成的影像物件，但是每畫一筆劃就要用網路傳整張圖片實在缺乏效率。因此，合理的程序是將己方繪圖的動作資訊(座標)經過網路傳到遠端，再由遠端的程式接收後以程式碼解析，並重繪一樣的筆畫。這樣雖然程式會比較複雜，但是需要的網路通訊流量卻可以大大減少。簡單的通訊流程如下圖所示：



在前面的兩章中我們實際傳遞的訊息分別是：即時通對話的文字，與猜數字遊戲的數字，本章我們將繼續使用類似的機制傳遞繪圖的座標資訊。雖然本範例不是一個完整的遊戲，但它可以透過網路來傳遞使用

者對滑鼠所做的操作；在多數的連線對戰遊戲中，這種方式是玩家之間通訊的重點。

3-2 表單設計

本章想實現的功能是讓玩家直接在表單上自由繪圖，每一筆畫不僅可以在自身表單上看到結果，完成之後也可以正確地顯示在遠端玩家的表單上。我們所需要的介面，只有設定網路通訊所需的幾個物件而已；包含三個TextBox物件，依序名為TextBox1~TextBox3，「連線」鈕相當於上一章的「啟動接收」鈕，名稱應該是Button1。表單實際大小可以自訂，不會影響後續程式的執行。

3-3 沿用的程式

本範例依舊可以沿用前兩章的部分程式碼，包括：

(1)表單匯入的命名空間(必須位於程式碼起始處眾多using項目之後)。

```
using System.Net;          //匯入網路通訊協定相關函數
using System.Net.Sockets; //匯入網路插座功能函數
using System.Threading;    //匯入多執行緒功能函數
```

以及(2)監聽物件與執行緒的宣告，(3)啟動監聽的按鈕，(4) Function MyIP副程序，(5)顯示本身IP於標題列的Form_Load事件副程序，(6)關閉監聽的Form_FormClosing事件副程序。此外，自訂的Listen監聽副程序也是必須的，但內容將有大幅改變，複製後請先將框架內的程式碼清空，再撰寫如下的程式碼：

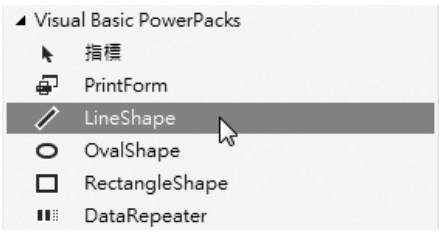
```
UdpClient U; //宣告UDP通訊物件
Thread Th;   //宣告監聽用執行緒
//啟動監聽按鍵程序
private void Button1_Click(object sender, EventArgs e)
{
    Control.CheckForIllegalCrossThreadCalls = false; //忽略跨執行緒操作的錯誤
    Th = new Thread(Listen); //建立監聽執行緒，目標副程序→Listen
    Th.Start();              //啟動監聽執行緒
    Button1.Enabled = false; //按鍵失效，不能(也不需要)重複開啟監聽
}
//監聽副程序
private void Listen()
{
}
//找出本機IP
private string MyIP()
{
    string hn = Dns.GetHostName();                    //取得本機電腦名稱
    IPAddress[] ip = Dns.GetHostEntry(hn).AddressList; //取得本機IP陣列(可能有多個)
    foreach (IPAddress it in ip)                      //列舉各個IP
    {
        if (it.AddressFamily == AddressFamily.InterNetwork) //如果是IPv4格式
        {
            return it.ToString();                          //回傳此IP字串
        }
    }
    return "";                                        //找不到合格IP，回傳空字串
}
//表單載入
private void Form1_Load(object sender, EventArgs e)
{
    this.Text += " " + MyIP();                        //顯示本機IP於標題列
}
//關閉接聽執行續(如果有的話)
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
```

```
        try
        {
            Th.Abort(); //關閉監聽執行緒
            U.Close();  //關閉監聽器
        }
        catch
        {
            //忽略錯誤，程式繼續執行！
        }
    }
```

以上程式碼的意涵請參考前面UDP即時通程式單元說明，在此不再重複。還有Listen自訂副程序框架必須先寫好，但是內容稍後再介紹及撰寫。

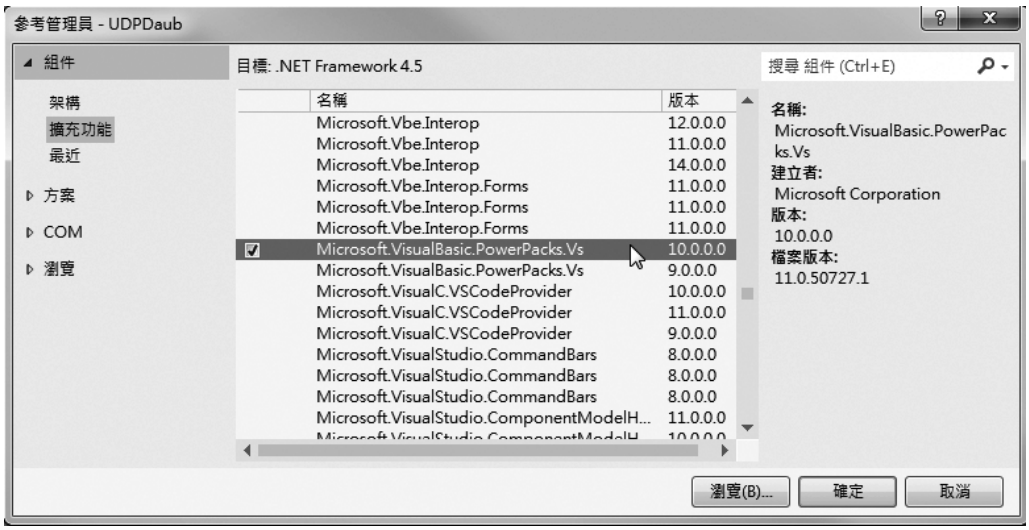
3-4 使用Shape物件繪圖的準備

雖然我們可以使用影像操作模式進行繪圖，就是宣告Bitmap影像物件與相搭配的Graphics繪圖物件，然後以DrawLine()函數執行繪製線段的動作。但是這樣繪圖時，每一個小線段的繪製都需要進行一次圖像轉移，速度上會比較慢。如果使用工具箱中所提供的LineShape物件(如下圖)來進行向量式的繪圖，效能會提高許多。採用這種繪圖方式的缺點相對於影像繪圖是不容易直接將影像儲存下來，如有必要儲存時就必須另外撰寫較多的程式碼，來製作成Bitmap影像物件。但是以網路互動的目的來說，速度快還是首要的考量！



比較遺憾的是目前的Visual Studio對於Shape物件的支援還不完整，簡單說Shape物件並不是功能完整（如Label）的獨立控制項，因此在撰寫程式動態產生Shape物件時並不能直接將它們加入表單之中，而是必須間接的先加入到一個ShapeContainer（畫布）物件，再將此畫布物件加入為表單物件，此情況有如要在表單上先貼上一張透明片，才能在此透明片上繪製Shape圖案。

但ShapeContainer並不在【工具箱】中，且包含它的命名空間(Microsoft.VisualBasic.PowerPacks.Vs) 預設也是未加入專案的！所以必須作以下「加入參考」的動作。請到功能表的「專案」項目選擇「加入參考」，出現如下視窗後選擇：Microsoft.VisualBasic.PowerPacks.Vs (版本10較佳)。



加入以上參考後，還必須匯入命名空間，在程式碼最前面，眾多using之後加入一行：

```
using Microsoft.VisualBasic.PowerPacks;//匯入VB向量繪圖功能
```

接下來我們就可以為後續由滑鼠動態產生的LineShape物件們先準備好一個家了！請雙按表單空白處進入Form_Load事件副程序，並在此事件副程序框架的外層與內部分別加入如下的程式碼：

```
//繪圖相關變數宣告
ShapeContainer C;              //畫布物件(本機繪圖用)
ShapeContainer D;              //畫布物件(遠端繪圖用)
Point stP;                     //繪圖起點
string p;                      //筆畫座標字串
//表單載入
private void Form1_Load(object sender, EventArgs e)
{
    C = new ShapeContainer(); //建立畫布(本機繪圖用)
    this.Controls.Add(C);     //加入畫布C到表單
    D = new ShapeContainer(); //建立畫布(遠端繪圖用)
    this.Controls.Add(D);     //加入畫布D到表單
}
```

在此宣告了兩個全域變數等級的ShapeContainer物件C與D，分別供本機及遠端玩家使用，原因是自己在繪圖時，有可能遠端也有訊息同步傳來要畫圖，此時如果共用一個畫布會產生搶畫布的跨執行緒衝突。還好畫布物件預設是透明的，我們在Form_Load事件副程序中宣告建置C與D的實體（new ShapeContainer()），並將它們都加入表單之中(this.Controls.Add)，兩張畫布上的繪圖結果最後看起來會是完全重疊在一起的！等於是兩個人各在一張透明片上繪圖，然後將結果疊在一起呈現。請注意到：在互猜數字或下棋類的遊戲中我們可以(也必須)控制輪到誰出招，但是塗鴉牆則不能有這種限制，如此設計是有必要的！

至於stP是用來記錄線段起點的變數，Point是一種包含X與Y座標的資料型別，使用Point可以避免每一個座標點都要使用X與Y兩個變數

來處理，程式碼會比較簡短。至於p則是用來記錄繪圖過程座標的字串，繪圖中必須完整記錄畫過了哪些點，最後才能正確地將筆畫傳給對手。這兩個變數都必須在多個副程序之間共用，所以必須宣告為公用的全域變數。

3-5 使用MouseDown與MouseMove繪圖

經過前一節的準備，我們已經有了可以繪圖的畫布，接著就是使用Form1的MouseDown與MouseMove事件來動態連續將LineShape物件加入畫布中，產生繪圖效果！程式碼如下：

```
//本機端開始繪圖
private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    stP = e.Location;                                  //起點
    p = stP.X.ToString() + "," + stP.Y.ToString(); //起點座標紀錄
}
```

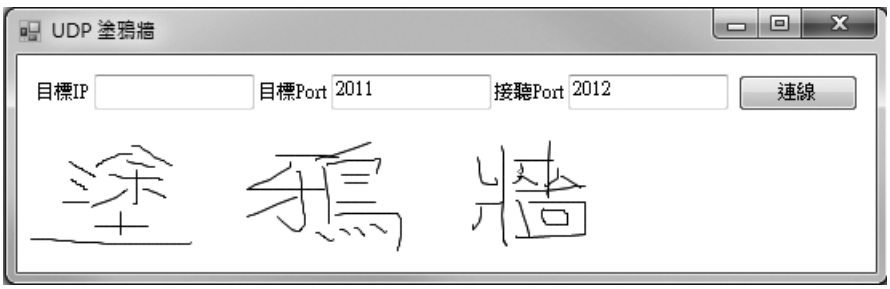
首先在MouseDown事件副程序中會將滑鼠位置紀錄為起點stP，此處可以看到使用Point資料型別的方便之處，即一行程式就同時將X與Y座標的資訊記錄到變數中了！同時間，也必須開始記錄整個筆畫經過的座標，就是以 "X, Y" 的字串格式先記錄stP的內容。此處如果p字串內有前一筆畫的資料將會被覆蓋掉，所以每一筆畫一定是從MouseDown，也就是使用者按下滑鼠時開始的。

```
//本機端繪圖中
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    if (e.Button == System.Windows.Forms.MouseButtons.Left)
    {
```

```
            LineShape L = new LineShape(); //建立線段物件
            L.StartPoint = stP;            //線段起點
            L.EndPoint = e.Location;       //線段終點
            L.Parent = C;                  //線段加入畫布C
            stP = e.Location;              //終點變起點
            p += "/" + stP.X.ToString() + "," + stP.Y.ToString();//持續紀錄座標
        }
    }
```

接著進入MouseMove事件副程序，首先須確認滑鼠左鍵是否處於按下狀態（e.Button == Windows.Forms.MouseButtons.**Left**），如果是，就開始繪圖了！先是宣告建置一個LineShape物件的實體(L)，接著定義起點（StartPoint）是stP，終點（EndPoint）是目前滑鼠的位置(e.Location)，然後將此物件實體的父物件（Parent）設定為C，這就是將線段(L)加入到畫布(C)上。因為畫布物件C已經在Form_Load時放在表單上，所以我們就可以看到線段顯示在表單上了！

此時的程式已經可以看到一個線段的繪圖結果。接著必須將新的線段起點（stP）設定為目前的滑鼠位置，這樣下一個線段才會接續在前一線段的末端。同時間，也必須繼續記錄經過的座標，在此為了區隔每個點的資料，我們是以 "/" 作為分隔字元，方便後續的接收端可以正確地區隔每個點的座標。在此可以先執行程式，測試本機塗鴉的效果了！畫面如下：



這裡用到了一個比較進階的「動態產生物件」的程式技巧。事實上不僅是字串或數值資料可以在程式進行中宣告產生，【工具箱】中的物件也是可以臨時用程式碼宣告建置，甚至刪除的！只是必須注意：非文字或數值的資料型別(或稱物件)，宣告建置時必須加上new關鍵字才會產生物件的實體，否則會發生錯誤！但即使是建置了實體，如果沒有以程式碼表明加入表單或表單上的某個物件時，使用者還是看不見這個物件的！

3-6 使用MouseUp傳出繪圖訊息

如果每次畫完一筆畫還要按一個按鈕送出筆畫座標，無疑是個很愚蠢的設計！我們合理的假設使用者畫的每一筆畫都是要傳送出去的，所以傳送資料的程式可以寫在放開滑鼠的MouseUp事件副程序裡，程式碼如下：

```
//送出繪圖動作
private void Form1_MouseUp(object sender, MouseEventArgs e)
{
    int Port = int.Parse(TextBox2.Text);              //設定發送的目標Port
    UdpClient S = new UdpClient(TextBox1.Text, Port);//建立UDP物件
    byte[] B = Encoding.Default.GetBytes(p);          //翻譯p字串為B陣列
    S.Send(B, B.Length);                              //發送資料
    S.Close();                                        //關閉UDP物件
}
```

這部分的程式與前面範例中的送出資料程式非常相似，就是將p字串翻譯成Byte陣列（B)，然後依據使用者設定的目標IP與Port建置一個臨時的UdpClient通訊物件S，並將資料送出。

3-7 接收資料自動繪圖的程式

遠端資料送到本機電腦時，當然要由Listen副程序來接收處理，收訊後的主要工作就是解析資料為座標點陣列，再用程式重繪出遠端使用者的繪圖動作，程式碼如下：

```
//監聽副程序
private void Listen()
{
    int Port = int.Parse(TextBox3.Text); //設定監聽用的通訊埠
    U = new UdpClient(Port);             //建立UDP監聽器
    //建立本機端點資訊
    IPEndPoint EP = new IPEndPoint(IPAddress.Parse("127.0.0.1"), Port);
    while (true) //持續監聽的無限迴圈→有訊息(True)就處理，無訊息就等待！
    {
        byte[] B = U.Receive(ref EP);            //訊息到達時讀取資訊到B陣列
        string A = Encoding.Default.GetString(B);//翻譯B陣列為字串A
        string[] Q = Z[1].Split('/');            //切割座標點資訊
        Point[] R = new Point[Q.Length];         //宣告座標點陣列
        for (int i = 0; i < Q.Length; i++)
        {
            string[] K = Q[i].Split(',');        //切割X與Y座標
            R[i].X = int.Parse(K[0]);            //定義第i點X座標
            R[i].Y = int.Parse(K[1]);            //定義第i點Y座標
        }
        for (int i = 0; i < Q.Length - 1; i++)
        {
            LineShape L = new LineShape();       //建立線段物件
            L.StartPoint = R[i];                 //線段起點
            L.EndPoint = R[i + 1];               //線段終點
            L.Parent = D; //線段L加入畫布D(遠端使用者繪圖)
        }
    }
}
```

上述程式碼除了粗體字部分，其實是與之前章節都一樣，在此不再重複說明。收到的資料轉譯回字串A之後，我們先用 "/" 間隔字元將字串切割、分解為Q字串陣列。想像一下：A的內容可能是這個樣子：

"0,0/1,1/2,3/…"。拆解之後的結果會是：**Q(0)="0,0"**，**Q(1)="1,1"**，**Q(2)="2,3"…**。A字串包含幾個座標點應該就會有幾個Q的成員，所以接著宣告一個名為R的Point陣列，個數與Q陣列相等。進一步用 "," 分隔字元拆解每個Q成員為K陣列，其中**K(0)**就是**X**座標，**K(1)**就是**Y**座標。使用一個迴圈一一處理，就可以將代表繪圖座標的Point陣列R之內容給解析出來！

接著我們再使用一個迴圈繪製連續的線段，此時先想一個問題：如果是5個點應該可以畫成幾個線段呢？答案是5 – 1 = 4（植樹問題），所以繪圖的迴圈終點（Q.Length-2）比之前的迴圈終點(Q.Length-1)少1。繪製線段的程式基本上與之前的MouseMove繪圖類似，但是前面有提到過，如果將遠端使用者的塗鴉也畫在C畫布上，又剛好碰到本機使用者也在畫圖時，會產生搶畫布物件(C)的衝突，所以應該將此地產生的線段畫在D畫布上。試試看，此時已經可以讓兩個程式實體進行遠端互動塗鴉繪圖了！

3-8 同時傳遞筆色的程式

只是傳遞筆畫座標或許太簡單了，如果希望同時可以選擇畫筆的顏色或畫筆的粗細，並且在遠端正確複製，應該如何進行呢？在此就以選擇畫筆的顏色做個示範，首先請在表單上加入四個RadioButton物件，並修改Text屬性為：Red，Green，Blue與Black，並將Black按鈕的Checked屬性設為True，畫面如下：

接著請到MouseMove事件副程序中加入如下粗體字的程式碼：

```
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    if (e.Button == System.Windows.Forms.MouseButtons.Left)
    {
        LineShape L = new LineShape();                                  //建立線段物件
        L.StartPoint = stP;                                             //線段起點
        L.EndPoint = e.Location;                                        //線段終點
        if (RadioButton1.Checked) { L.BorderColor = Color.Red; }    //紅筆
        if (RadioButton2.Checked) { L.BorderColor = Color.Lime; }   //亮綠色筆
        if (RadioButton3.Checked) { L.BorderColor = Color.Blue; }   //藍筆
        if (RadioButton4.Checked) { L.BorderColor = Color.Black; } //黑筆
        L.Parent = C;                                                   //線段加入畫布C
        stP = e.Location;                                               //終點變起點
        p += "/" + stP.X.ToString() + "," + stP.Y.ToString();       //持續紀錄座標
    }
}
```

上面程式的意義是當建立LineShape物件時，依據使用者選擇的顏色 (RadioButton) 改變BorderColor，因為線段物件的邊界 (Border) 就是本體，所以BorderColor(邊界顏色)就是改變畫筆的顏色了！但是光這樣修改，接收訊息的對方不會知道你用的筆色，因此在送出p字串時還必須加入一個顏色代號，在此就用1~4代表上面的四個顏色！表單的MouseUp事件副程序需加入四行程式(粗體字部分)變成這樣：

```
private void Form1_MouseUp(object sender, MouseEventArgs e)
{
    int Port = int.Parse(TextBox2.Text);              //設定發送的目標Port
    UdpClient S = new UdpClient(TextBox1.Text, Port); //建立UDP物件
    if (RadioButton1.Checked) { p = "1_" + p; }       //紅筆
    if (RadioButton2.Checked) { p = "2_" + p; }       //亮綠色筆
    if (RadioButton3.Checked) { p = "3_" + p; }       //藍筆
    if (RadioButton4.Checked) { p = "4_" + p; }       //黑筆
    byte[] B = Encoding.Default.GetBytes(p);          //翻譯p字串為B陣列
    S.Send(B, B.Length);                              //發送資料
    S.Close();                                        //關閉UDP物件
}
```

當然，傳送的訊息結構變了，監聽用的Listen程式處理方式也必須同步修改！修改部分也是用粗體字標示，程式碼如下：

```
        private void Listen()
        {
         int Port = int.Parse(TextBox3.Text); //設定監聽用的通訊埠
            U = new UdpClient(Port);          //建立UDP監聽器
            //建立本機端點資訊
            IPEndPoint EP = new IPEndPoint(IPAddress.Parse("127.0.0.1"), Port);
            while (true) //持續監聽的無限迴圈→有訊息(True)就處理，無訊息就等待！
            {
                byte[] B = U.Receive(ref EP);            //訊息到達時讀取資訊到B陣列
                string A = Encoding.Default.GetString(B);//翻譯B陣列為字串A
                string[] Z = A.Split('_');               //切割顏色與座標資訊
            string[] Q = Z[1].Split('/');                //切割座標點資訊
            Point[] R = new Point[Q.Length];             //宣告座標點陣列
                for (int i = 0; i < Q.Length; i++)
                {
                    string[] K = Q[i].Split(',');  //切割X與Y座標
                    R[i].X = int.Parse(K[0]);      //定義第i點X座標
                    R[i].Y = int.Parse(K[1]);      //定義第i點Y座標
                }
                for (int i = 0; i < Q.Length - 1; i++)
                {
                    LineShape L = new LineShape(); //建立線段物件
                    L.StartPoint = R[i];           //線段起點
                    L.EndPoint = R[i + 1];         //線段終點
                    switch (Z[0])                  //筆色
                    {
```

```
                    case "1":                   //紅筆
                        L.BorderColor = Color.Red;
                        break;
                    case "2":                   //亮綠色筆
                        L.BorderColor = Color.Lime;
                        break;
                    case "3":                   //藍筆
                        L.BorderColor = Color.Blue;
                        break;
                    case "4":                   //黑筆
                        L.BorderColor = Color.Black;
                        break;
                }
                L.Parent = D;                   //線段L加入畫布D(遠端使用者繪圖)
            }
        }
    }
```

如同之前切割座標點與X，Y的方式相似，此處先用一個陣列Z以底線 ("_") 字元切割顏色與座標訊息為Z(0)與Z(1)。在繪製線段時，再依據Z(0)決定使用的筆色即可！一個不變的原則是同一個筆畫的相關資訊應該在同一次資料傳送中完成，如果要同時傳遞筆的粗細應該可以很容易發揮你的想像力完成。

在此英文好的讀者或許會問為何綠色的畫筆不是Green？而是Lime？這是實驗得知的結果，使用Green其實會出現比較暗的綠色，不是我們預期的純綠色，Lime才是RGB組合為 (0,255,0) 的那個亮亮的綠色。測試看看此時程式應該已經可以正確地傳遞筆畫的顏色了！

課後討論

有關Shape物件的使用

本例使用了動態產生LineShape的方式繪圖，如同匯入參考時所見，其實這些都是Visual Basic語言的功能函數，現在從C#專案也可以輕鬆引用，真是太棒了！事實上在VB6的時代，這些Shape物件就已經非常好用，而且也常被使用到！尤其是可以設為物件陣列的特性，對於常常需要設計很多小物件之遊戲程式的設計者來說簡直是太棒了！但是很遺憾的，這一部分的物件程式似乎沒有很順利地升級到.NET世代，雖然目前【工具箱】中有看到這些物件，但是預設的匯入程式庫項目仍然不完整，所以才會需要在3-4節中介紹的那些麻煩的設定與程式碼。

以遊戲設計者的角度來說當然非常希望.NET也可以快點承襲VB6時代既有的長處，讓Shape物件的使用更加方便，最好所有的物件也都能以陣列方式操作。如果暫時還是如此不便的話，讀者也可以沾沾自喜，在此多學到了一種高效能的繪圖程式寫法。資訊技術如浪潮一般的持續進步，如果總是等到設計環境都很方便時再學習某種技術，就沒有領先的優勢了！學習較新且較難的技術，雖然等到環境進步時會有點失落，因為大家都可以簡單地完成一樣的事情，之前困難的技術操作都白學了！但是這個短暫領先的時刻就是讀者暫時所擁有的優勢，所以還是值得爭取與珍惜的。

範例檢視 Example Review

[專案設計頁面]

Form1：

[完整程式碼]

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;                          //匯入網路通訊協定相關函數
using System.Net.Sockets;                  //匯入網路插座功能函數
using System.Threading;                    //匯入多執行緒功能函數
using Microsoft.VisualBasic.PowerPacks; //匯入VB向量繪圖功能

namespace UDPDaub
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }

    UdpClient U; //宣告UDP通訊物件
    Thread Th;   //宣告監聽用執行緒
    //啟動監聽按鍵程序
    private void Button1_Click(object sender, EventArgs e)
    {
      Control.CheckForIllegalCrossThreadCalls = false; //忽略跨執行緒操作的錯誤
      Th = new Thread(Listen); //建立監聽執行緒，目標副程序→Listen
      Th.Start();              //啟動監聽執行緒
      Button1.Enabled = false; //按鍵失效，不能(也不需要)重複開啟監聽
    }
```

```
//找出本機IP
private string MyIP()
{
  string hn = Dns.GetHostName();                        //取得本機電腦名稱
  IPAddress[] ip = Dns.GetHostEntry(hn).AddressList; //取得本機IP陣列(可能有多個)
  foreach (IPAddress it in ip)                          //列舉各個IP
  {
    if (it.AddressFamily == AddressFamily.InterNetwork)//如果是IPv4格式
    {
      return it.ToString();                              //回傳此IP字串
    }
  }
  return "";      //找不到合格IP，回傳空字串
}
//繪圖相關變數宣告
ShapeContainer C; //畫布物件(本機繪圖用)
ShapeContainer D; //畫布物件(遠端繪圖用)
Point stP;        //繪圖起點
string p;         //筆畫座標字串

//表單載入
private void Form1_Load(object sender, EventArgs e)
{
  C = new ShapeContainer(); //建立畫布(本機繪圖用)
  this.Controls.Add(C);     //加入畫布C到表單
  D = new ShapeContainer(); //建立畫布(遠端繪圖用)
  this.Controls.Add(D);     //加入畫布D到表單
  this.Text += " " + MyIP();//顯示本機IP於表單標題列
}

//關閉接聽執行續(如果有的話)
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
  try
  {
    Th.Abort(); //關閉監聽執行緒
    U.Close();  //關閉監聽器
  }
  catch
  {
    //如果監聽執行緒沒開會出現錯誤，程式跳到此處執行，
    //此處不寫程式就是忽略錯誤，程式繼續執行的意思！
  }
}

//監聽副程序
private void Listen()
{
  int Port = int.Parse(TextBox3.Text); //設定監聽用的通訊埠
  U = new UdpClient(Port);             //建立UDP監聽器
```

```
  //建立本機端點資訊
  IPEndPoint EP = new IPEndPoint(IPAddress.Parse("127.0.0.1"), Port);
  while (true) //持續監聽的無限迴圈→有訊息(True)就處理，無訊息就等待！
  {
    byte[] B = U.Receive(ref EP);             //訊息到達時讀取資訊到B陣列
    string A = Encoding.Default.GetString(B);//翻譯B陣列為字串A
    string[] Z = A.Split('_');                //切割顏色與座標資訊
    string[] Q = Z[1].Split('/');             //切割座標點資訊
    Point[] R = new Point[Q.Length];          //宣告座標點陣列
    for (int i = 0; i < Q.Length; i++)
    {
      string[] K = Q[i].Split(',');           //切割X與Y座標
      R[i].X = int.Parse(K[0]);               //定義第i點X座標
      R[i].Y = int.Parse(K[1]);               //定義第i點Y座標
    }
    for (int i = 0; i < Q.Length - 1; i++)
    {
      LineShape L = new LineShape();          //建立線段物件
      L.StartPoint = R[i];                    //線段起點
      L.EndPoint = R[i + 1];                  //線段終點
      switch (Z[0]) //筆色
      {
        case "1":   //紅筆
          L.BorderColor = Color.Red;
          break;
        case "2":   //亮綠色筆
          L.BorderColor = Color.Lime;
          break;
        case "3":   //藍筆
          L.BorderColor = Color.Blue;
          break;
        case "4":   //黑筆
          L.BorderColor = Color.Black;
          break;
      }
      L.Parent = D; //線段L加入畫布D(遠端使用者繪圖)
    }
  }
}

//本機端開始繪圖
private void Form1_MouseDown(object sender, MouseEventArgs e)
{
  stP = e.Location;                                  //起點
  p = stP.X.ToString() + "," + stP.Y.ToString(); //起點座標紀錄
}
```

```
    //本機端繪圖中
    private void Form1_MouseMove(object sender, MouseEventArgs e)
    {
      if (e.Button == System.Windows.Forms.MouseButtons.Left)
      {
        LineShape L = new LineShape(); //建立線段物件
        L.StartPoint = stP;            //線段起點
        L.EndPoint = e.Location;       //線段終點
        if (RadioButton1.Checked) { L.BorderColor = Color.Red; }  //紅筆
        if (RadioButton2.Checked) { L.BorderColor = Color.Lime; } //亮綠色筆
        if (RadioButton3.Checked) { L.BorderColor = Color.Blue; } //藍筆
        if (RadioButton4.Checked) { L.BorderColor = Color.Black; }//黑筆
        L.Parent = C;                                             //線段加入畫布C
        stP = e.Location;                                         //終點變起點
        p += "/" + stP.X.ToString() + "," + stP.Y.ToString();     //持續紀錄座標
      }
    }

    //送出繪圖動作
    private void Form1_MouseUp(object sender, MouseEventArgs e)
    {
      int Port = int.Parse(TextBox2.Text);              //設定發送的目標Port
      UdpClient S = new UdpClient(TextBox1.Text, Port); //建立UDP物件
      if (RadioButton1.Checked) { p = "1_" + p; }       //紅筆
      if (RadioButton2.Checked) { p = "2_" + p; }       //亮綠色筆
      if (RadioButton3.Checked) { p = "3_" + p; }       //藍筆
      if (RadioButton4.Checked) { p = "4_" + p; }       //黑筆
      byte[] B = Encoding.Default.GetBytes(p);          //翻譯p字串為B陣列
      S.Send(B, B.Length);                              //發送資料
      S.Close();                                        //關閉UDP物件
    }
  }
}
```