

Xamarin 行動開發 iOS 課程

Steven Chang
講師 張朝銘



Agenda

- 開發工具及環境介紹
- 基礎架構及專案結構
- 基本控制項介紹
- 多頁面巡覽控制
- TableView使用
- 客製化TableView Cell
- 跨平台架構設計
- Sqlite.Net-With Share File
- 使用WebService With PCL
- 使用WebAPI+Json.Net With PCL
- 共享更多程式碼-MVVM模型

Lab下載

<https://github.com/stevenchang0529/XamainClass2015iOS>



開發工具安裝及環境介紹

Xamarin 安裝-iOS

■ Windows

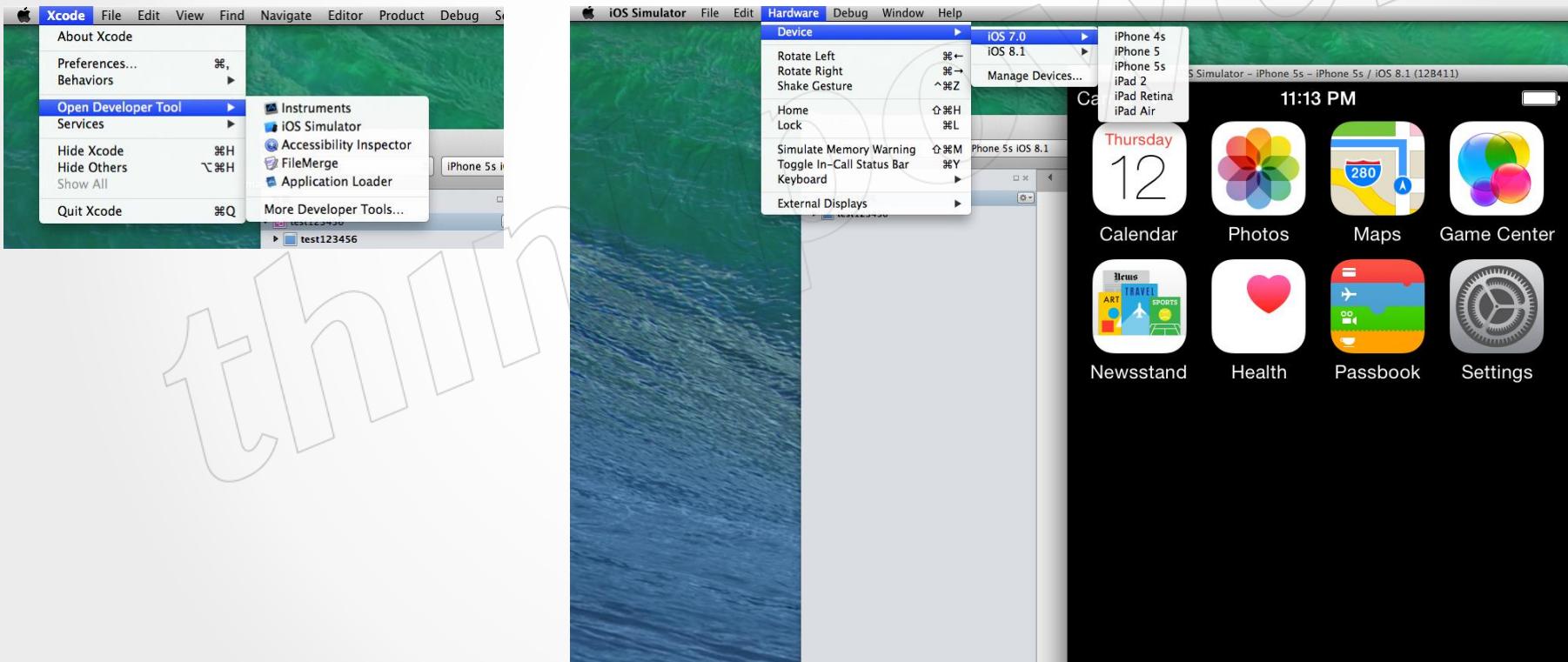
- Windows7或更新的作業系統
- Xamarin.iOS
- Xamarin Plug-in for Visual Studio(2010以上)
- Xamarin Studio

■ MAC

- OS X Lion 或更新的作業系統
- Xcode與iOS SDK
- Xamarin.iOS
- Xamarin Studio for MAC

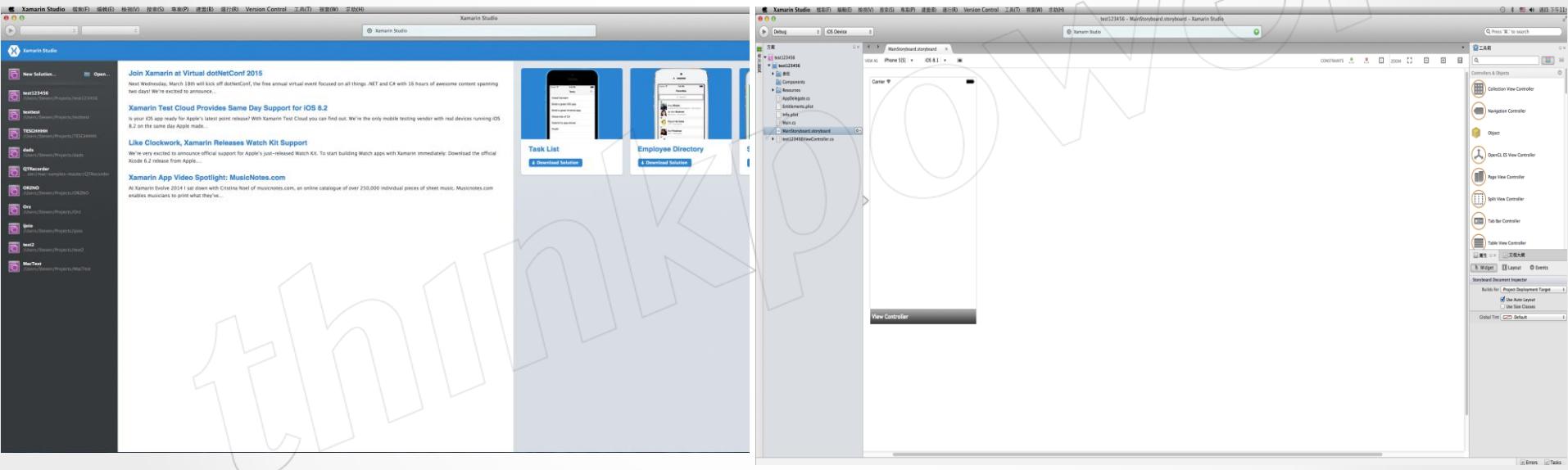
使用模擬器

- 安裝Xcode後，會附帶模擬器
- 可選擇各版本的iOS以及手機模擬



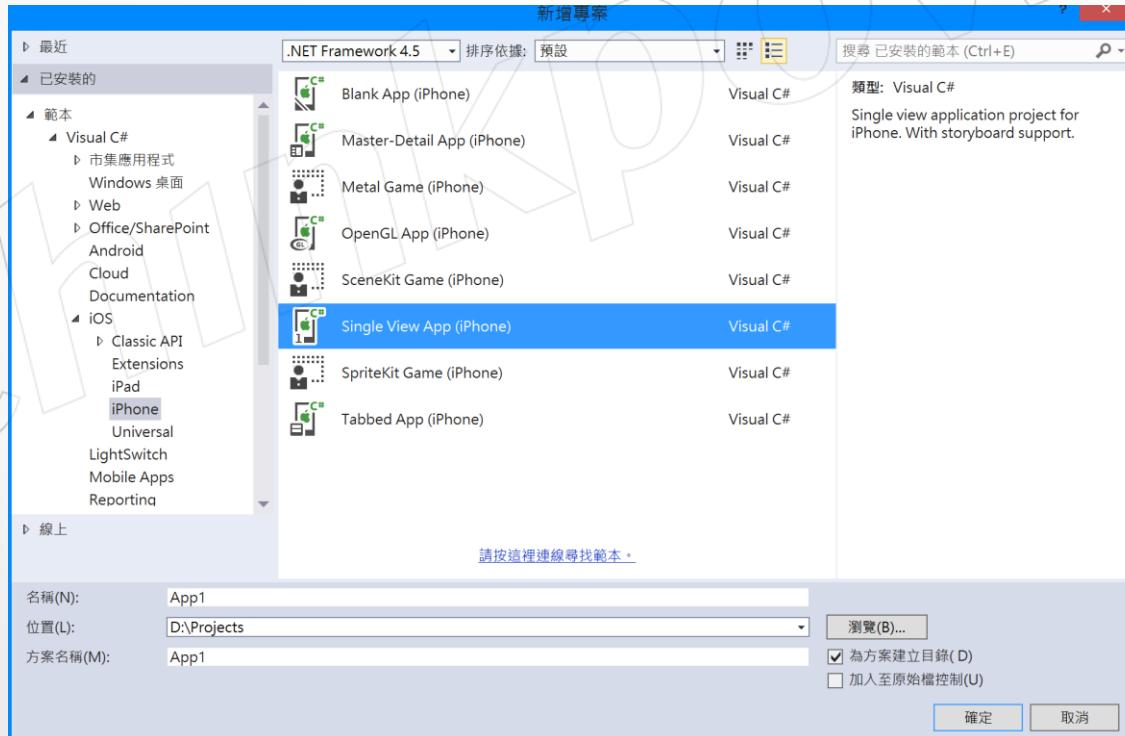
使用Xamarin Studio

- iOS專案僅支援在MAC環境下
- 介面與Visual Studio類似，並可與Git整合



使用Visual Studio

- 支援2010以上(最新到2015)
- 可整合TFS、Git
- 需登入試用帳號或付費帳號才能使用
- 開發iOS專案依然需要有MAC環境，並開啟Build Host

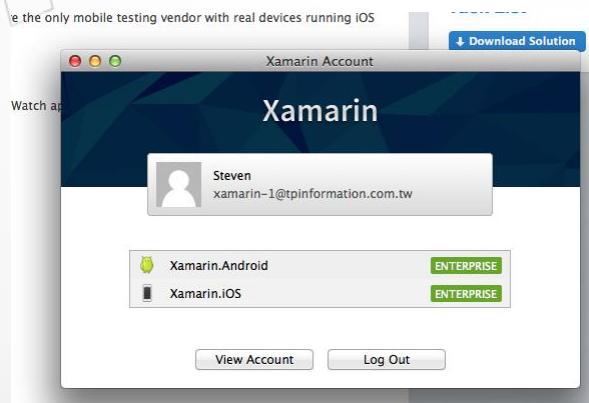


註冊試用帳號

- 於以下網址註冊Xamarin帳號
<https://store.xamarin.com/account/register>
- Email不會進行認證可隨意填寫
- 試用帳號於APP開啟時會顯示試用提示畫面
- 試用所編譯出來的APP在安裝後僅能使用24小時
- 試用帳號可使用30天
- iOS與Android需分別啟動試用

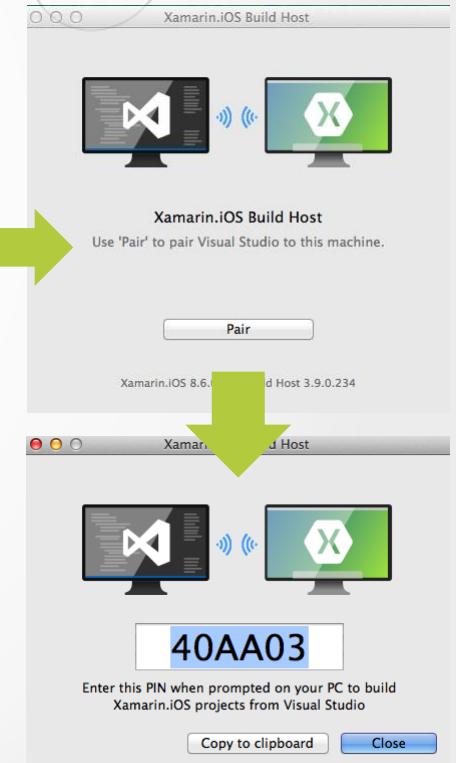
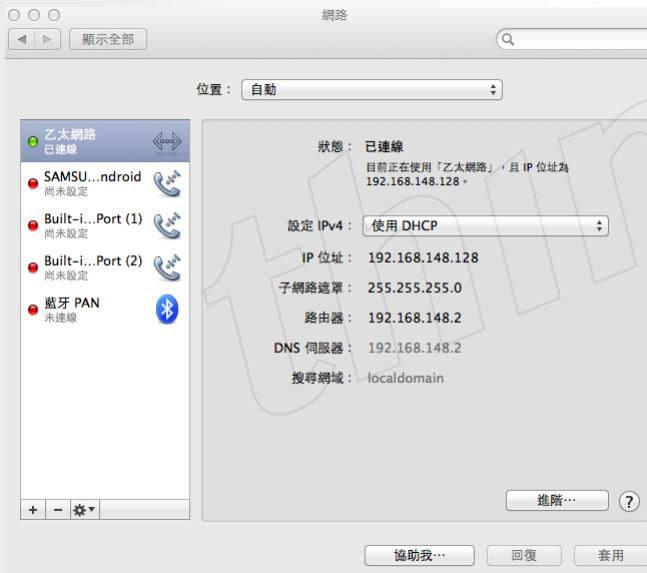
下載及試用

- 於Xamarin官網中可下載一鍵安裝檔
<http://xamarin.com/platform>
- 安裝程式會自動偵測下載所有需要下載的檔案
- 使用Build Host需有一組Xamarin付費或試用帳號
- Xamarin Studio中選擇 Account 進行登出、登入及檢視帳號資訊



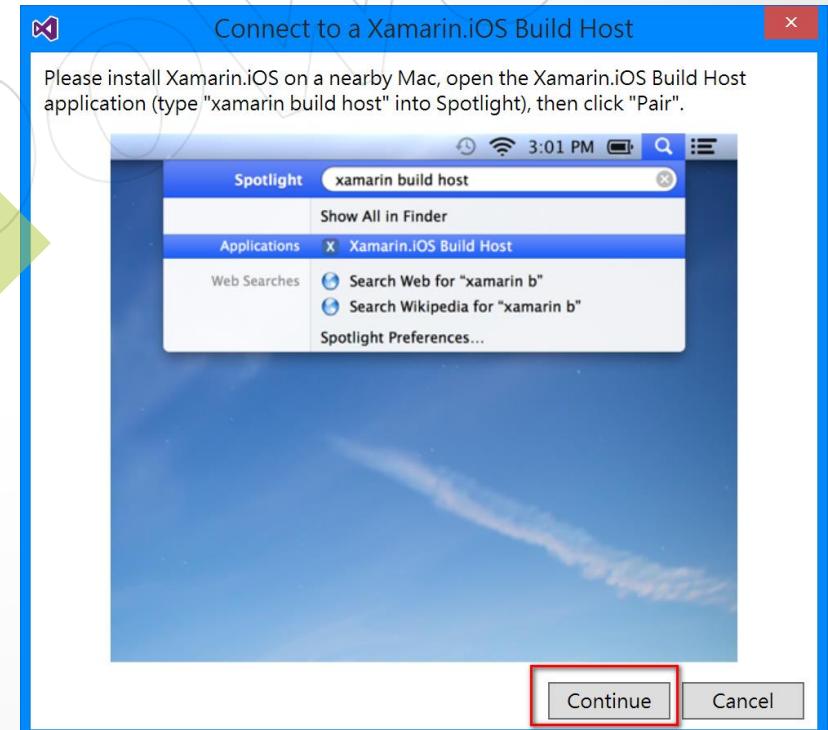
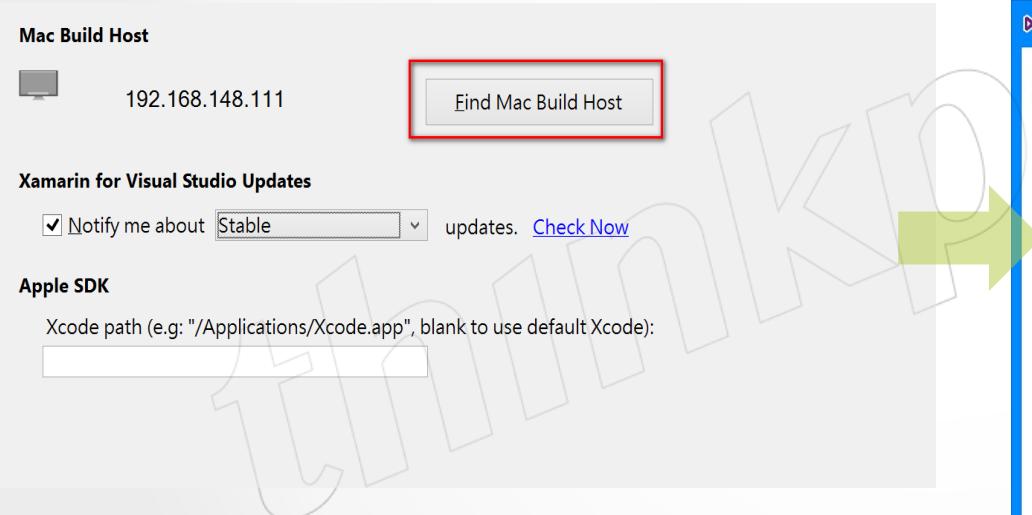
開啟Build Host

- 使用Visual Studio開發需開啟MAC環境上的Build Host
- 選擇Mac圖示→系統偏好設定→網路 記下目前IP
- 於OS中選擇 前往→應用程式→Xamarin.iOS Build Host
- 按下Pair，記下PIN碼



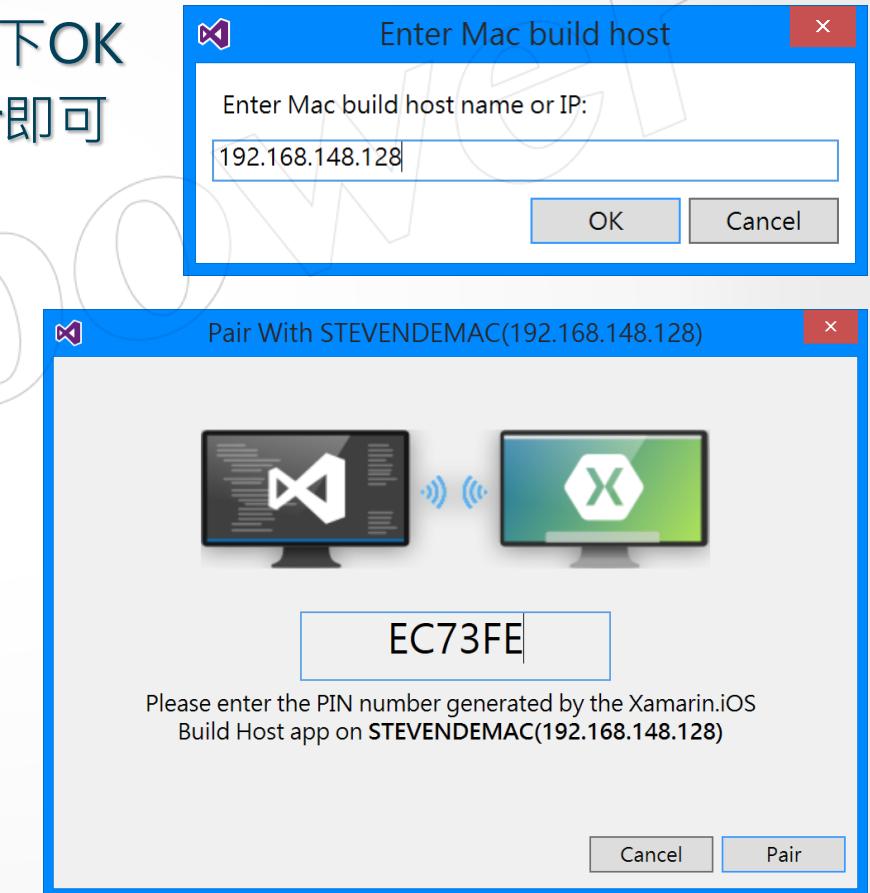
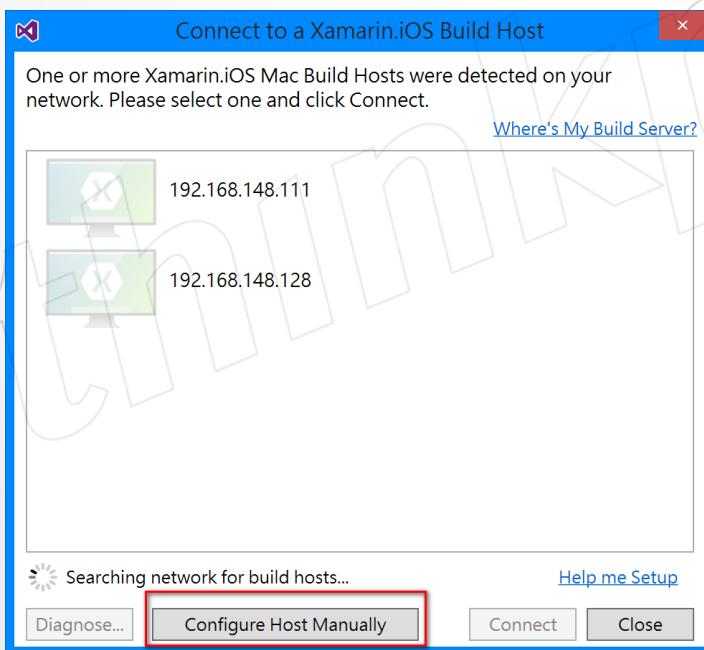
連接Build Host

- 開啟Visual Studio→工具→選項→Xamarin →iOS Setting
- 點選Find Mac Bulid Host →按下Continue



連接Build Host

- 選擇Configure Host Manually
- 輸入於OS環境上看到的IP位置按下OK
- 輸入剛剛記下的PIN碼後按下Pair即可

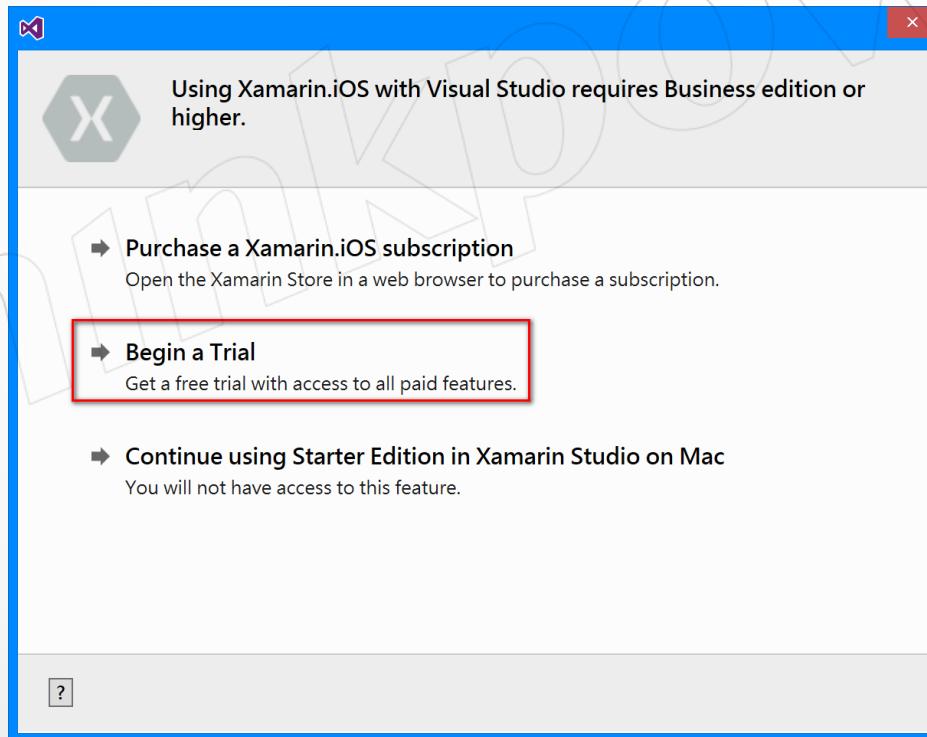




基礎架構及專案結構

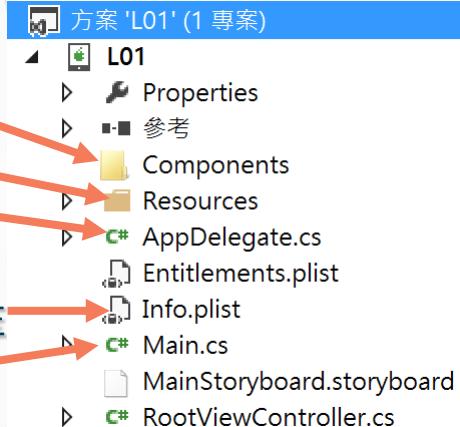
新增iOS專案

- 開啟Visual Studio並新增專案，選擇iOS→iPhone→Single View App
- 開啟專案後會跳出驗證畫面，選擇啟用試用帳號



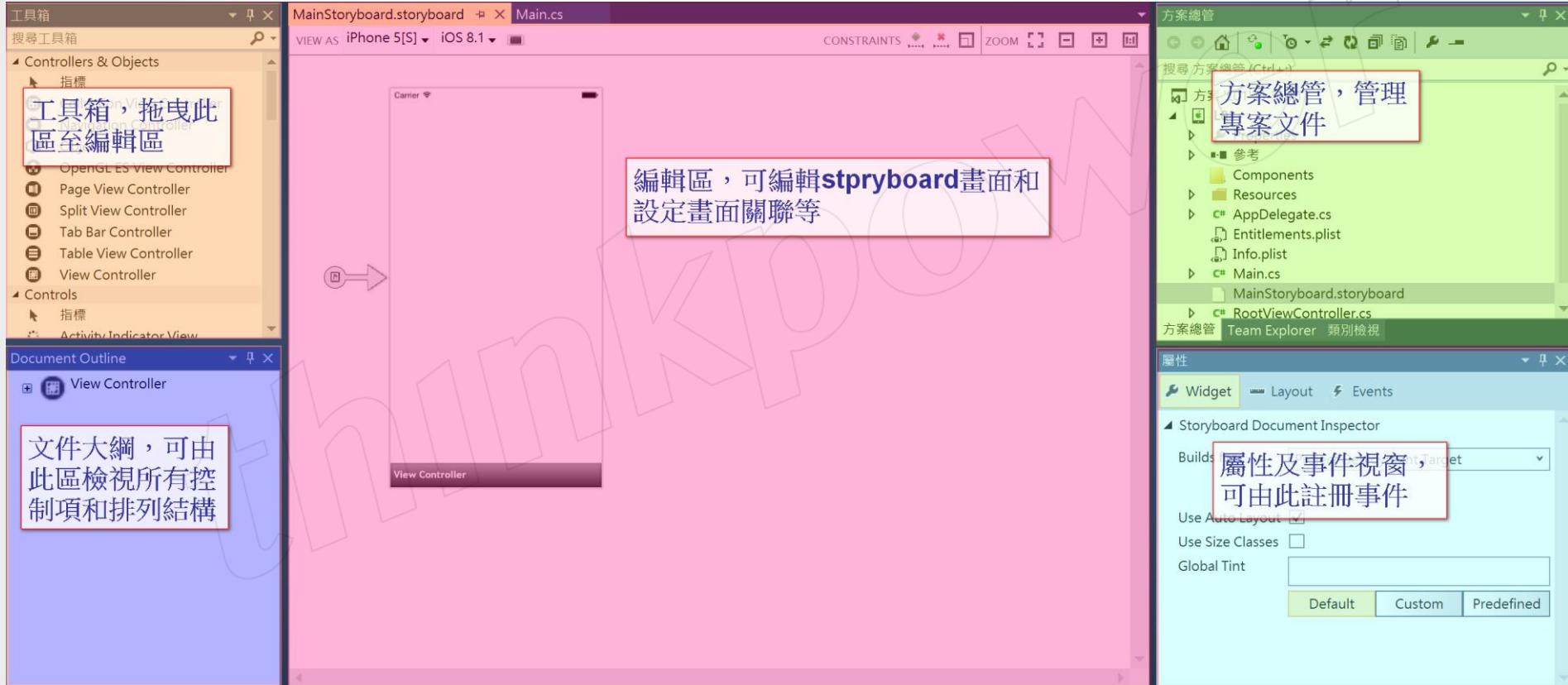
專案結構

- Components:
放置下載的Xamarin元件
- Resources:
放置圖片，多媒體檔案等
- AppDelegate.cs
負責處理來自系統的事件的主要應用程式
- Info.plist:
設置應用程式的icon，版本，佈署裝置和啟動圖片等
- Main.cs
應用程式起始點進入點



可視化編輯畫面

17



iOS架構

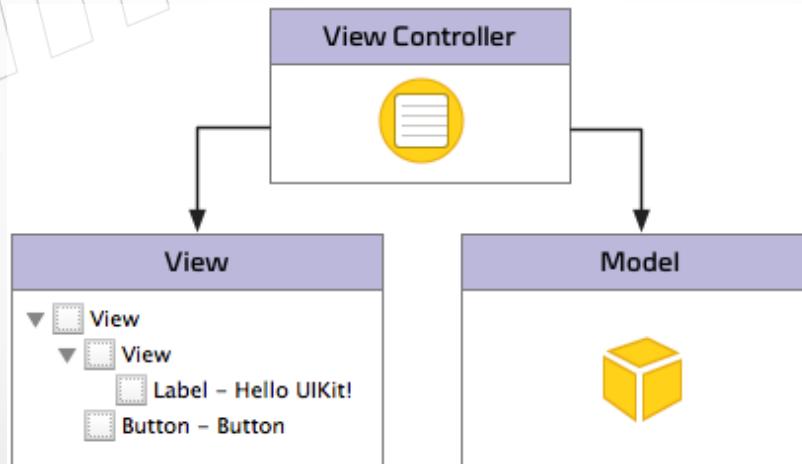
採用了MVC框架:

■ View

檢視層，通常為一個xib檔，或者是storyboard中的一個View，泛指所有使用者看的到的畫面，檔案實際上也是由XML組成的

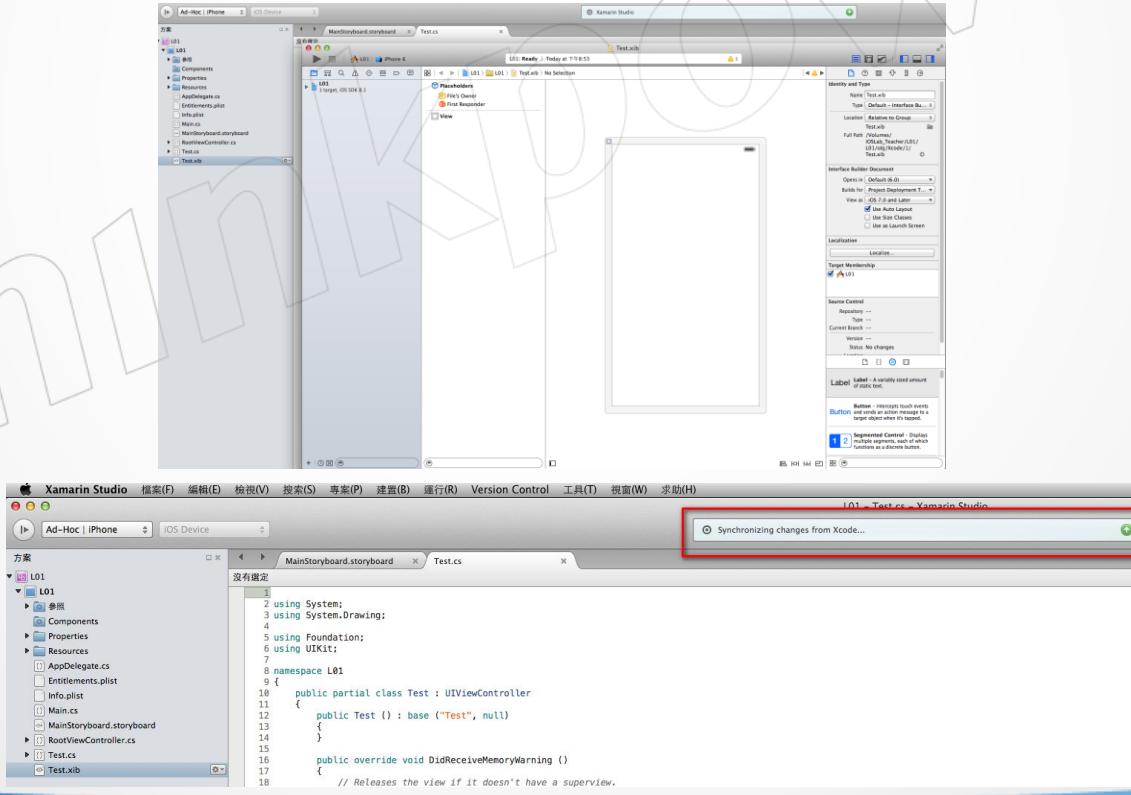
■ Controller

控制器層，在iOS中通常指的是UIViewController，每一個View可指定對應到一個UIViewController。



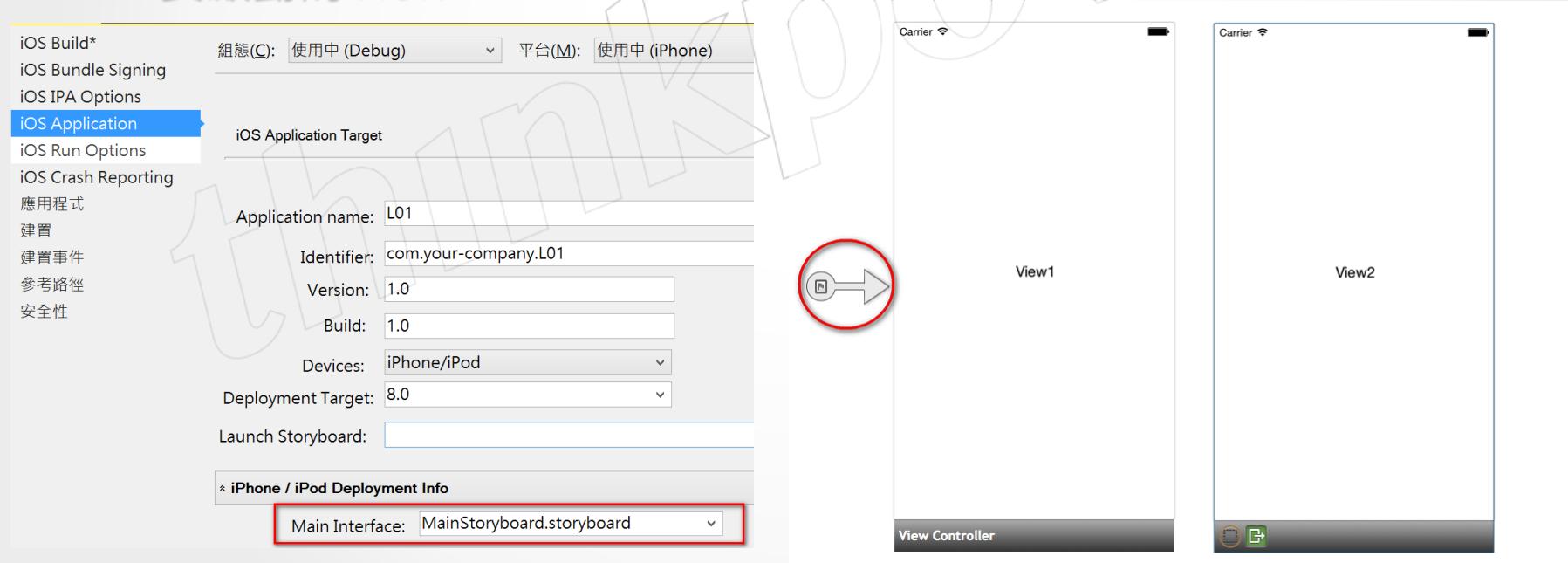
xib檔使用

- Xamarin不支援編輯xib檔案，需由Xcode編輯，因此在Visual Studio下無法編輯xib檔。
- 於Xamarin專案中點選xib檔會自動開啟Xcode，關閉Xcode後會自動將畫面資訊同步回Xamarin Studio



指定起始畫面

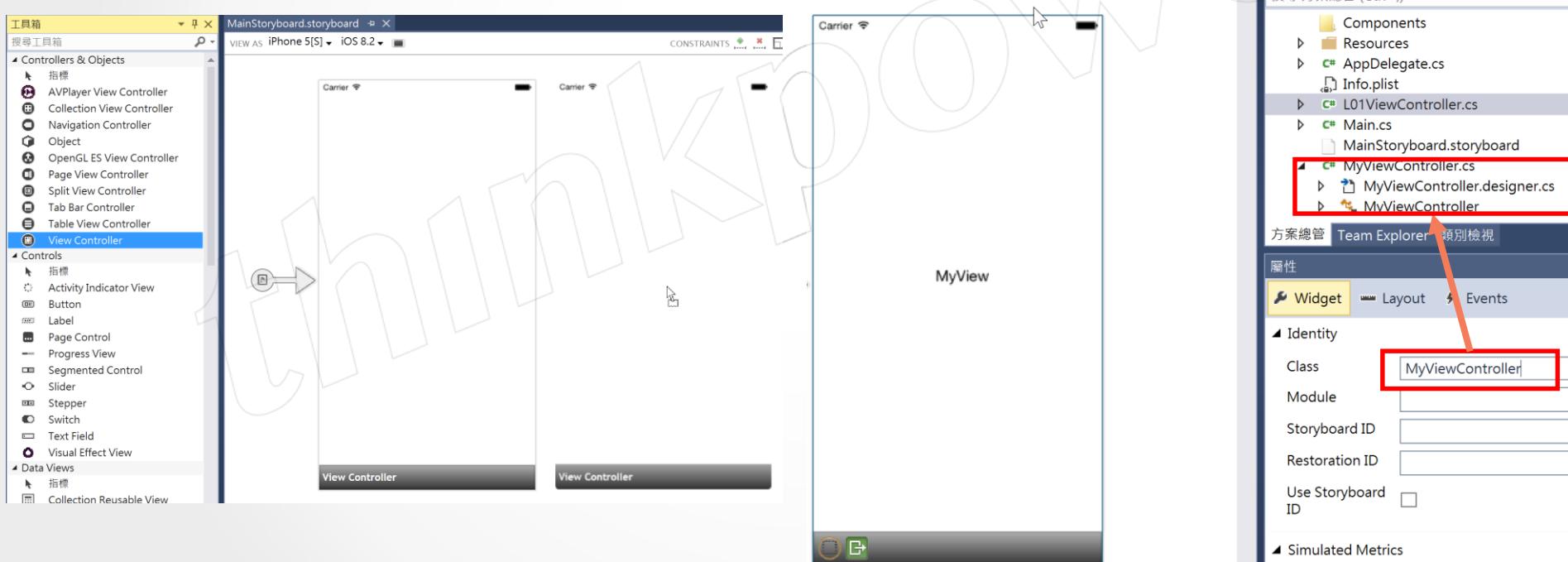
- 開啟專案屬性，選擇iOS Application，調整Main Interface可指定主要啟動的Storyboard
- 開啟Storyboard後，移動如圖箭頭的部分，可以指定Storyboard中要啟動的View



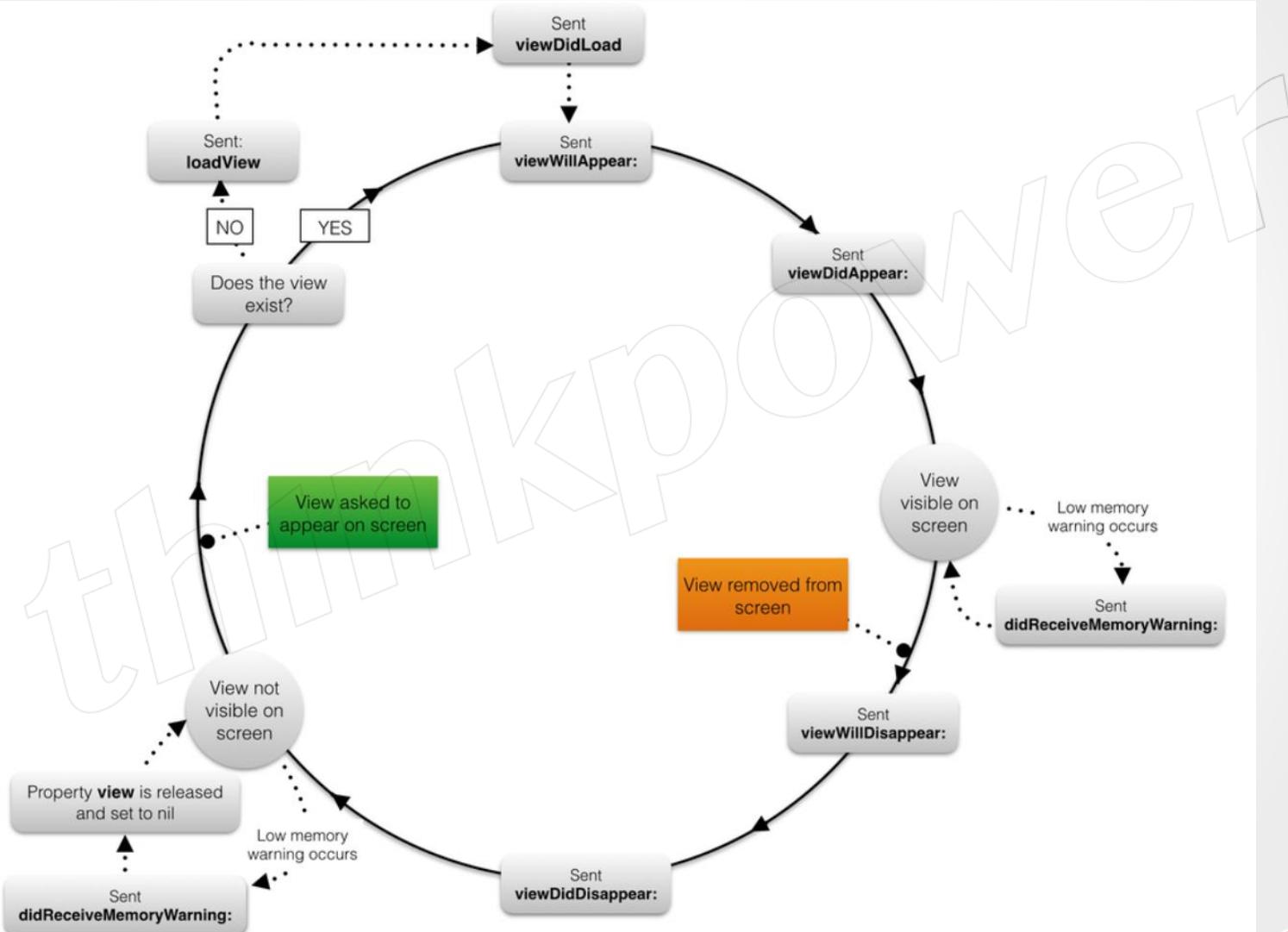
建立View對應的ViewController

21

- 在工具箱內拖曳ViewController可新增一個View
- 點選新增的View，在屬性視窗中的Class填入自訂的Controller名稱
- 定義完Class名稱後按下Enter，會自動產生對應的類別檔

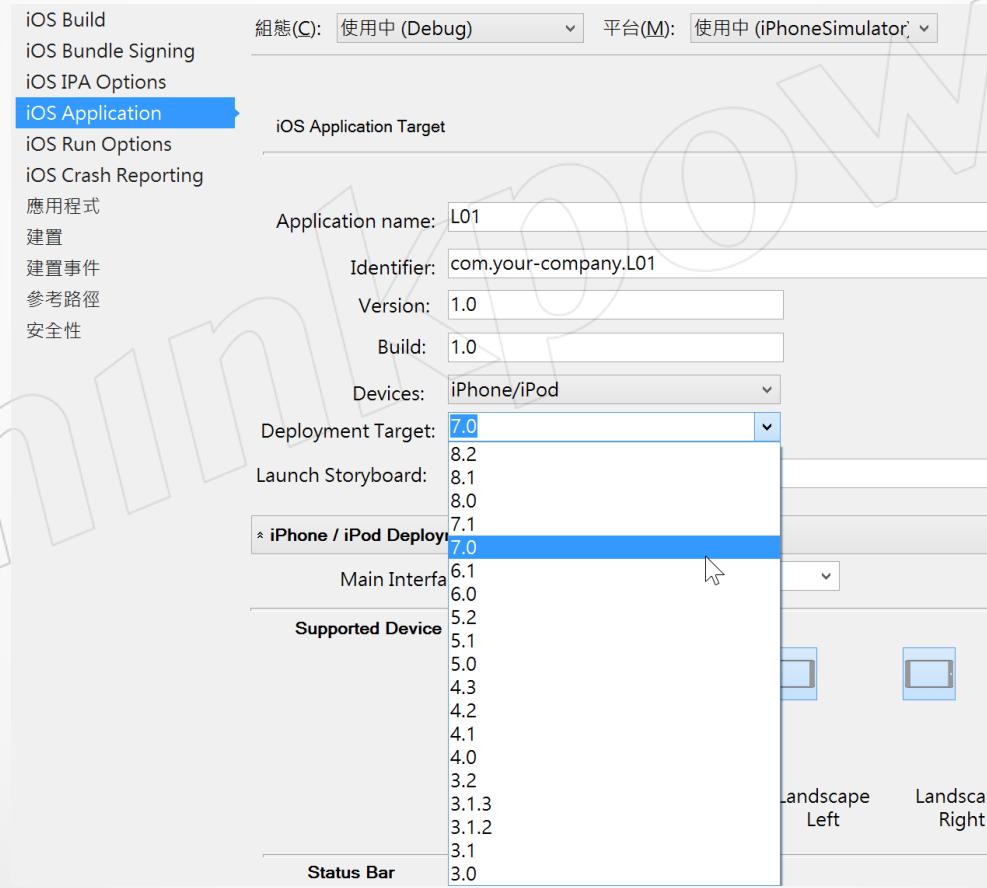


ViewController生命週期



設定版本

於專案屬性中選擇iOS Application，調整Deployment Target可變更目標裝置版本

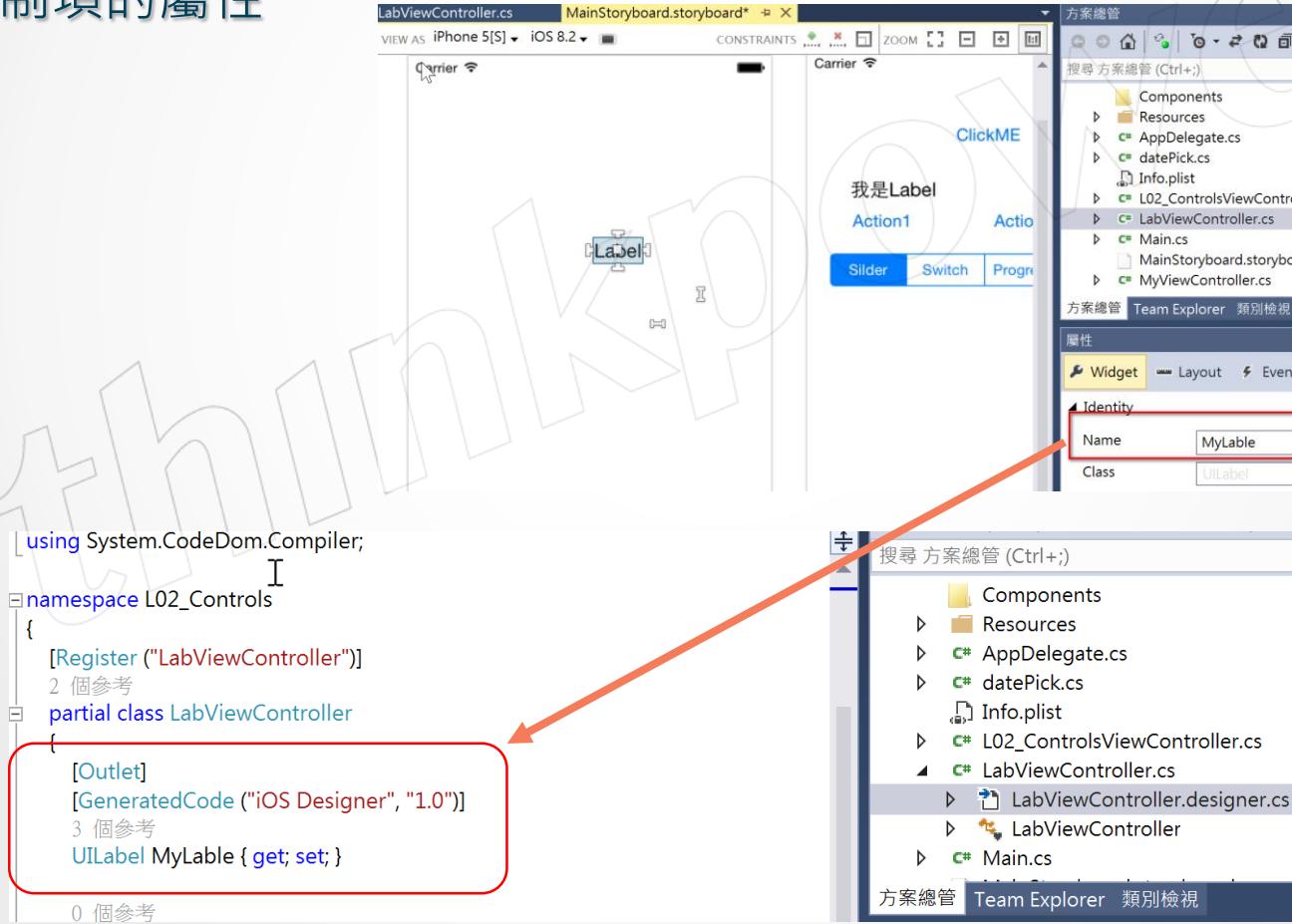




基本控制項介紹

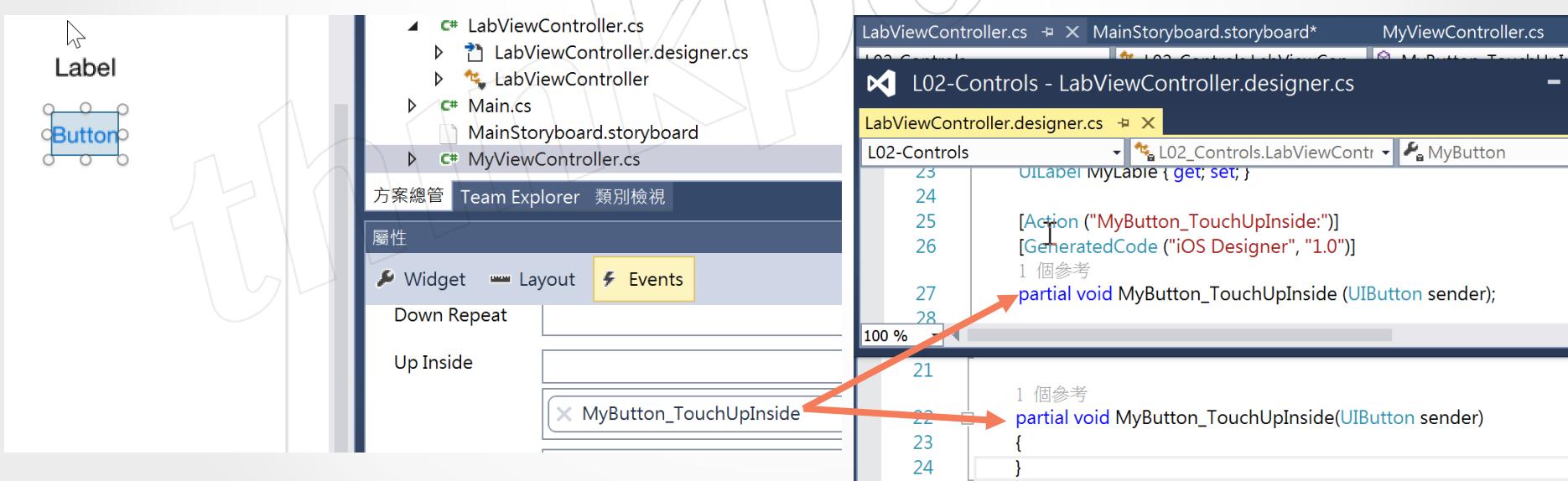
控制項命名

- 拖曳控制項到View後，可在屬性中選擇Name為控制項命名
- 命名完成後，會自動在對應的controller.designer.cs檔案中，產生該控制項的屬性



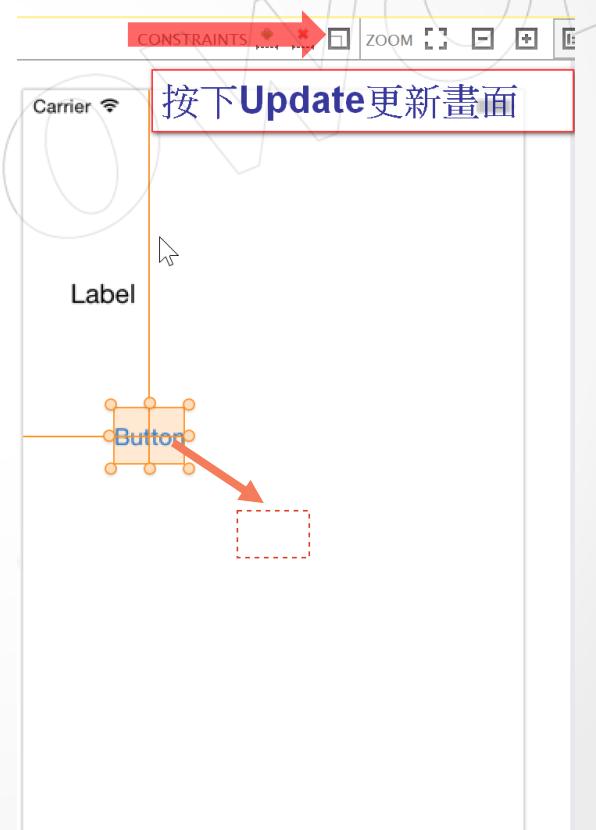
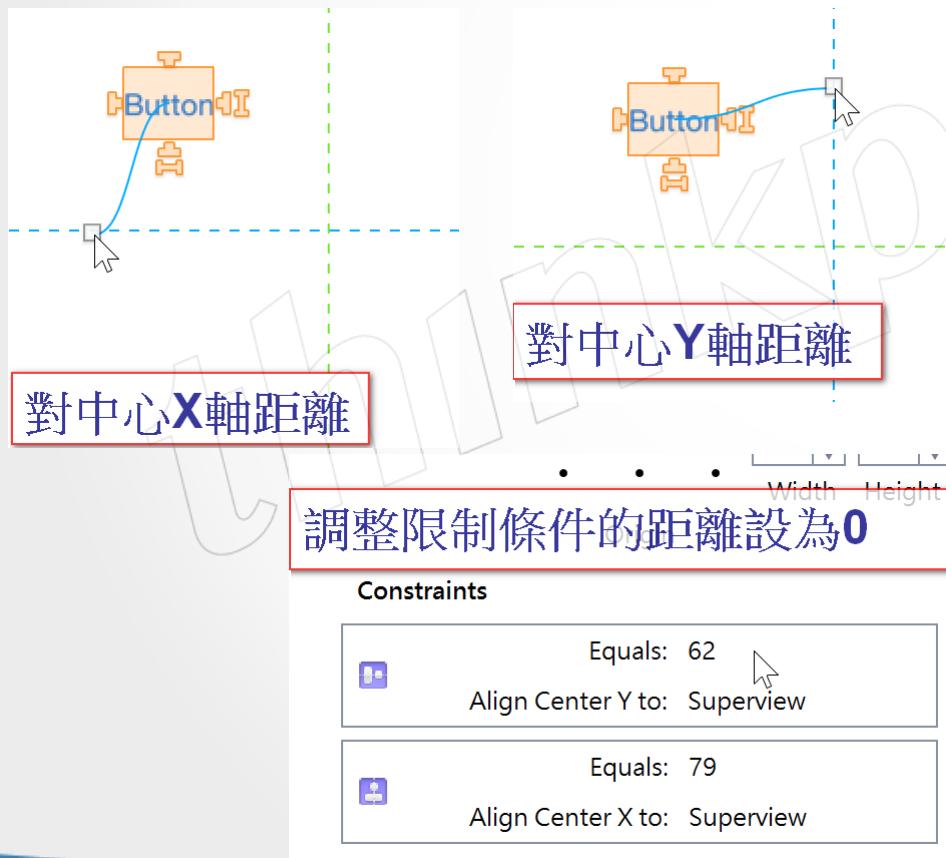
註冊事件

- 拖曳控制項到View後，可使用程式碼的方式註冊事件
- 控制項的初值賦予與事件註冊應在ViewDidLoad()進行
- 以Button為例，可直接滑鼠點Button兩下註冊Button的Action (TouchUpInside) · 或直接在程式碼註冊Button的TouchUpInside事件
- Action產生的為部分方法(partial) · 事件則是委派方法



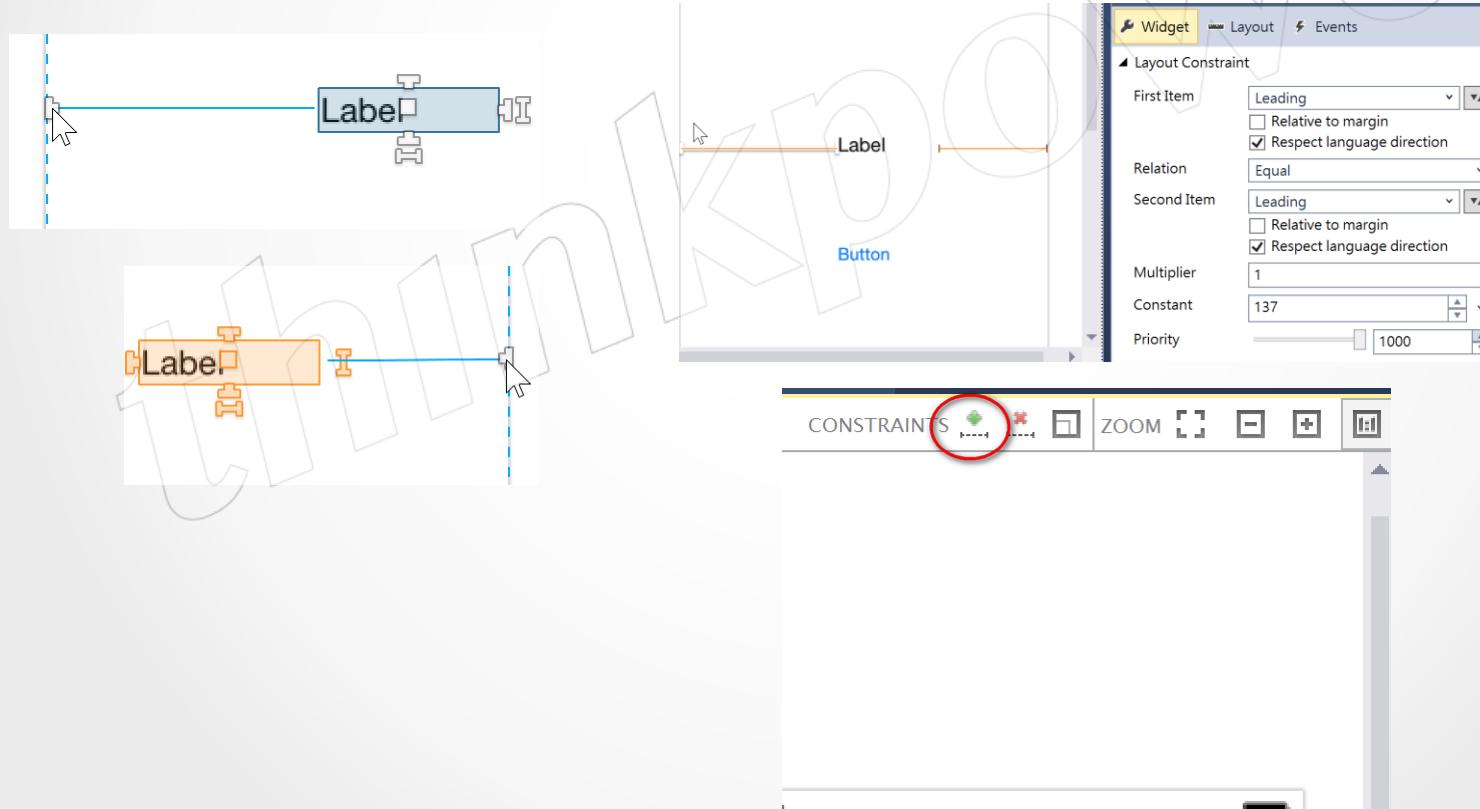
限制條件-Constraints 置中

除了使用滑鼠拖曳及自行設定長寬外，還可以為控制項加入限制條件，左鍵點兩下控制項，會出現下圖顯示，例如將控制項置於目前所在View的中心。



限制條件-Constraints 拉寬

- 增加控制項距離左側和右側的寬度之限制條件
- 調整限制條件數值
- 也可以直接點上方的+號增加基本限制條件



基本控制項

■ UILabel

- 文字行數:lines



■ UITextField

- 提示:PlaceHolder
- 文字:Text



■ UIButton

- 點擊事件:TouchUpInside



基本控制項

■ UISegmentedControl

- 切換事件:ValueChanged

Silder Switch Progress Image

■ UISlider

- 最大最小值:MaxValue、MinValue



■ UISwitch

- 開關狀態:On(bool)



■ UIProgressView

- 進度值:Progress(flot:0~1)



基本控制項

■ UIImageView

- 圖片縮放模式:Mode
- 圖片來源:Image



■ UIDatePicker

- 選擇顯示時間或選擇日期:Mode
- 選擇最大與最小可選時間日期:MaximumDate、MinimumDate



基本控制項

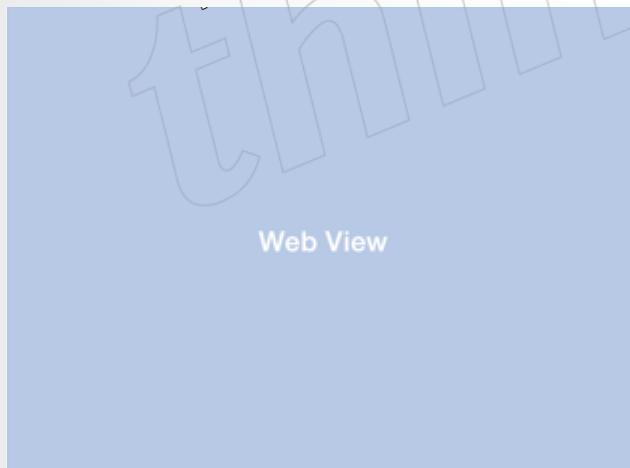
■ UIAlertView

- 顯示視窗>Show()
- 按鈕事件:Click



■ UIWebView

- 載入網址:LoadRequest()



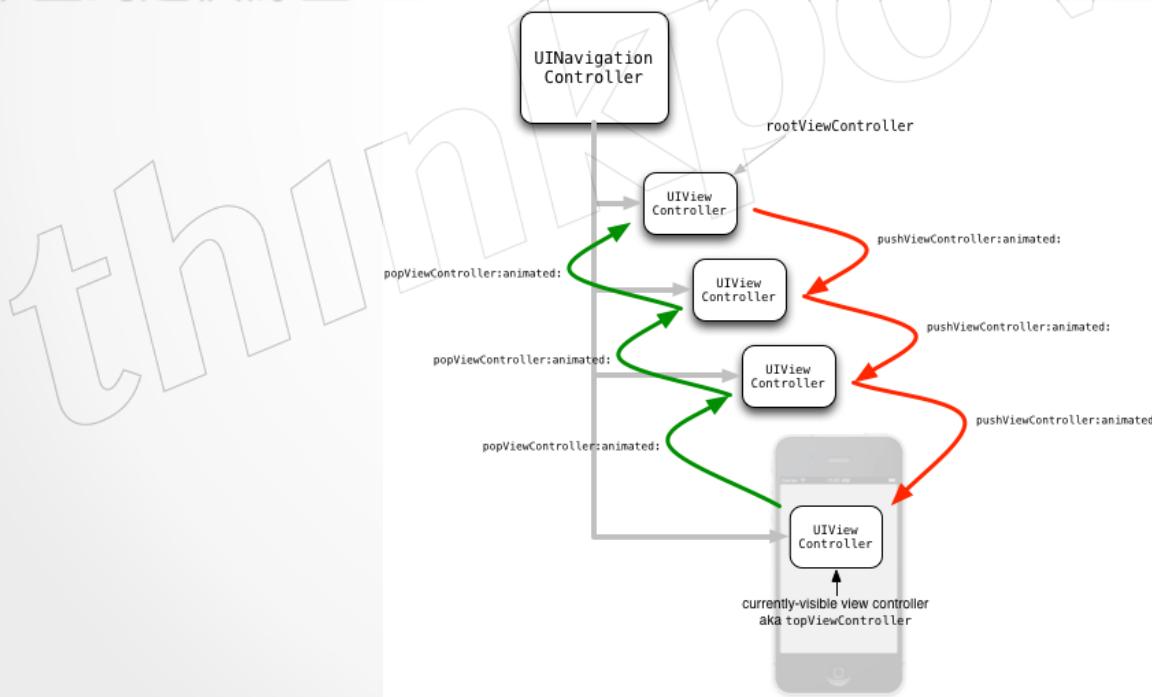
```
LoadRequest(new NSURLRequest(new NSURL("http://www.thinkpower.com.tw")));
```



多頁面巡覽控制

使用Navigation Controller

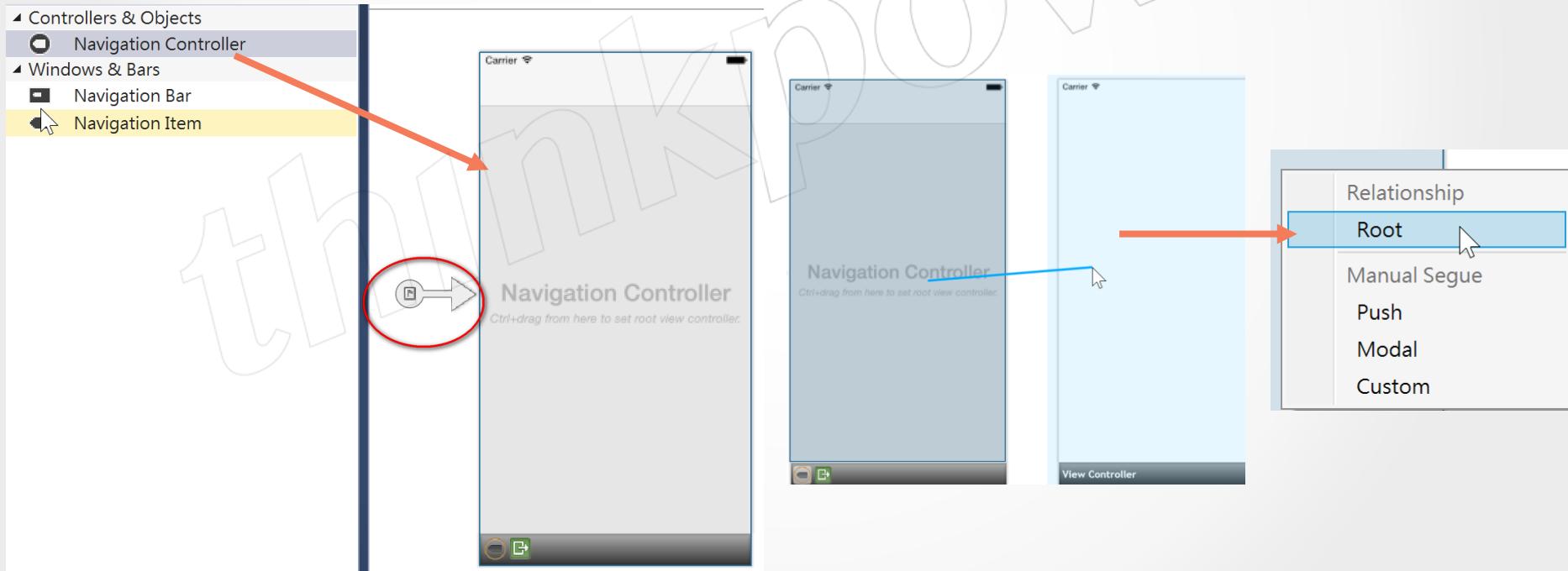
當我們要使用多頁面瀏覽時，最常使用的就是`NavigationController`，可以譯作導航控制器，可以將他理解成一條線，這個線中可以加入多個`ViewController`，而建立這條線時會加入第一個`ViewController`，稱為`Root ViewController`，而當切換頁面時就會加入新的`ViewController`依序堆疊到這個線上。



加入Navigation Controller

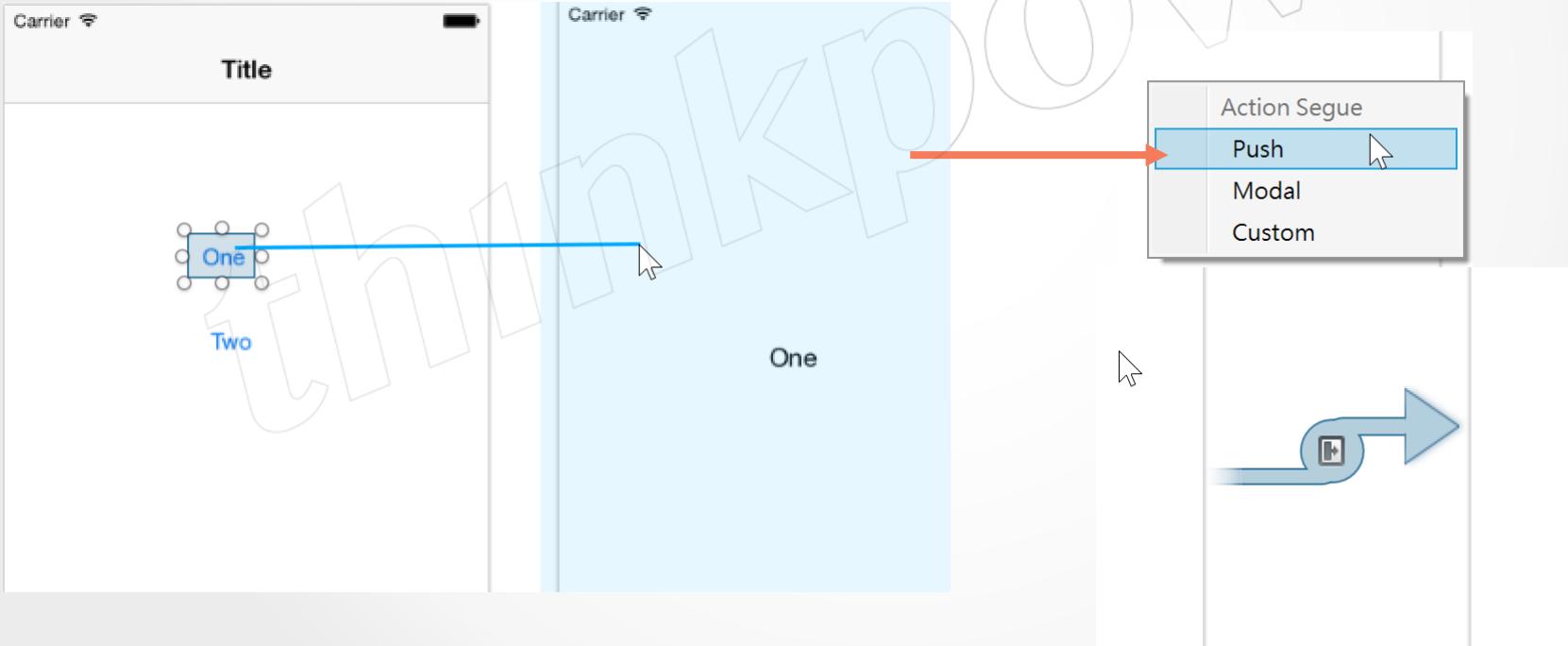
在Storyboard中可以很快速的加入Navigation Controller

- 從工具箱中拖曳Navigation Controller並設為進入點
- 設定Root View(滑鼠左鍵+Ctrl拖曳至要加入的View)



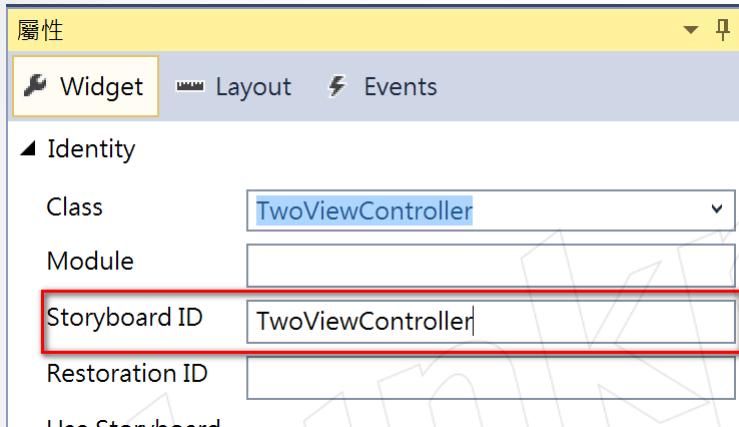
使用拖曳方式顯示下一個View

- 在View中對要換到下一页的控制項使用滑鼠左鍵+Ctrl拖曳到下一個View
- 在下一個View選擇Push，兩個View之間會出現連線(



使用程式碼顯示下一個View

- 需將View給予StoryboardID屬性



- 使用程式碼建出TwoViewController，並使用PushViewController()

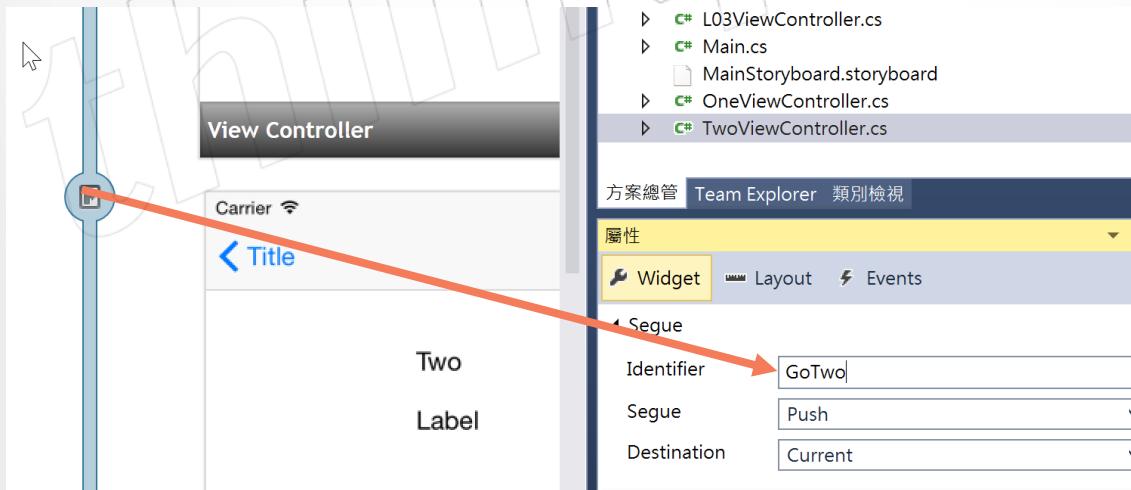
```
//lab:使用程式碼的方式按下Two By Code讓Root轉換至Two
var controller = this.Storyboard.InstantiateViewController("Test") as TwoViewController;
this.NavigationController.PushViewController(controller, true);
```

傳遞參數給下一個View

- 如透過拉線的方式想要傳遞參數給下一個View，需覆寫 PrepareForSegue方法

```
public override void PrepareForSegue(UIStoryboardSegue segue, NSObject sender)
{
    base.PrepareForSegue(segue, sender);
}
```

- 在兩個View相連的線稱為Segue，可以設定屬性Identifier來判斷 PrepareForSegue是要進行哪一個Segue



其他導航控制方法

以下還列出了其他常用的導航控制方法

- **PopViewControllerAnimated()**
回到上一個ViewController
- **PopToRootViewController()**
回到Root ViewController
- **PopToViewController()**
回到目前堆疊中某一個ViewController

Navigation Item

使用了Navigation Controller後，除了Root View之外的View在畫面上方會增加一個Navigation Item，並在左邊附上一個Back Button，Button的文字預設會等於上一個View的Title



自訂左邊和右邊的按鈕

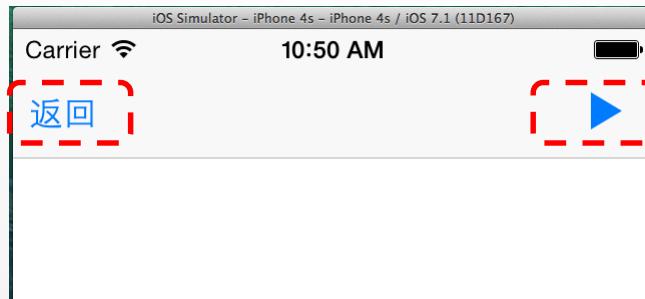
使用程式碼我們可以自訂左邊與新增右邊的按鈕。

■ 變更左方的按鈕自訂文字 SetLeftBarButtonItem()

```
this.NavigationItem.SetLeftBarButtonItem(new UIBarButtonItem("返回", UIBarButtonItemStyle.Plain, (sender, e) =>
{
}),
), true);
```

■ 變更右方的按鈕為系統圖示 SetRightBarButtonItem()

```
this.NavigationItem.SetRightBarButtonItem(new UIBarButtonItem(UIBarButtonItemSystemItem.Play,(sender,e)=>
{
}),
), true);
```



調整Nav Bar樣式

42

■ 調整Title文字

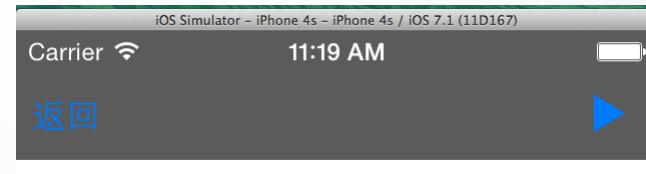
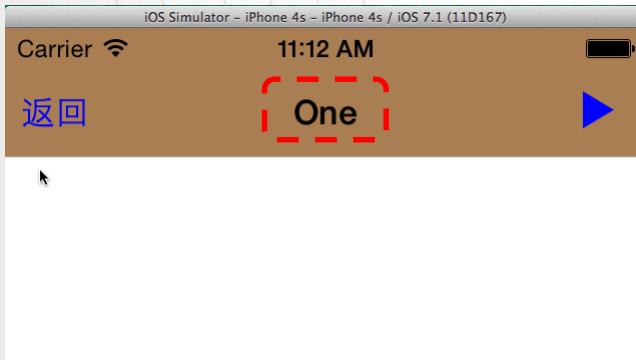
```
this.Title = "One";
```

■ 設定文字和背景顏色

```
this.NavigationController.NavigationBar.TintColor = UIColor.Blue;  
this.NavigationController.NavigationBar.BarTintColor = UIColor.Brown;
```

■ 設定Style樣式

```
NavigationController.NavigationBar.BarStyle = UIBarStyle.Black;
```





TableView使用

TableView使用

TableView相當於Android中的ListView，也是以列表方式呈現資料，在APP中也很常使用到。

- 使用UITableViewController，在UITableViewController處理事件、決定分組，以及處理每一個Cell。
- TableView中的每一列稱為Cell，類別為UITableViewCell，在UITableViewController中要決定每一個Cell的樣式，iOS提供了幾種預設的Cell可使用：
http://developer.xamarin.com/guides/ios/user_interface/tables/part_3_-_customizing_a_table's_appearance/
- 將UITableViewController指定給TableView上的Source屬性。

實作UITableViewSource

UITableViewSource為抽象類別，要實作兩個抽象方法

- **GetCell()**
決定每一個Cell的資料與樣式
- **RowsInSection()**
Section為Table分組(Group)，預設值為1，即不分組，不分組時這裡只要回傳資料來源的數量

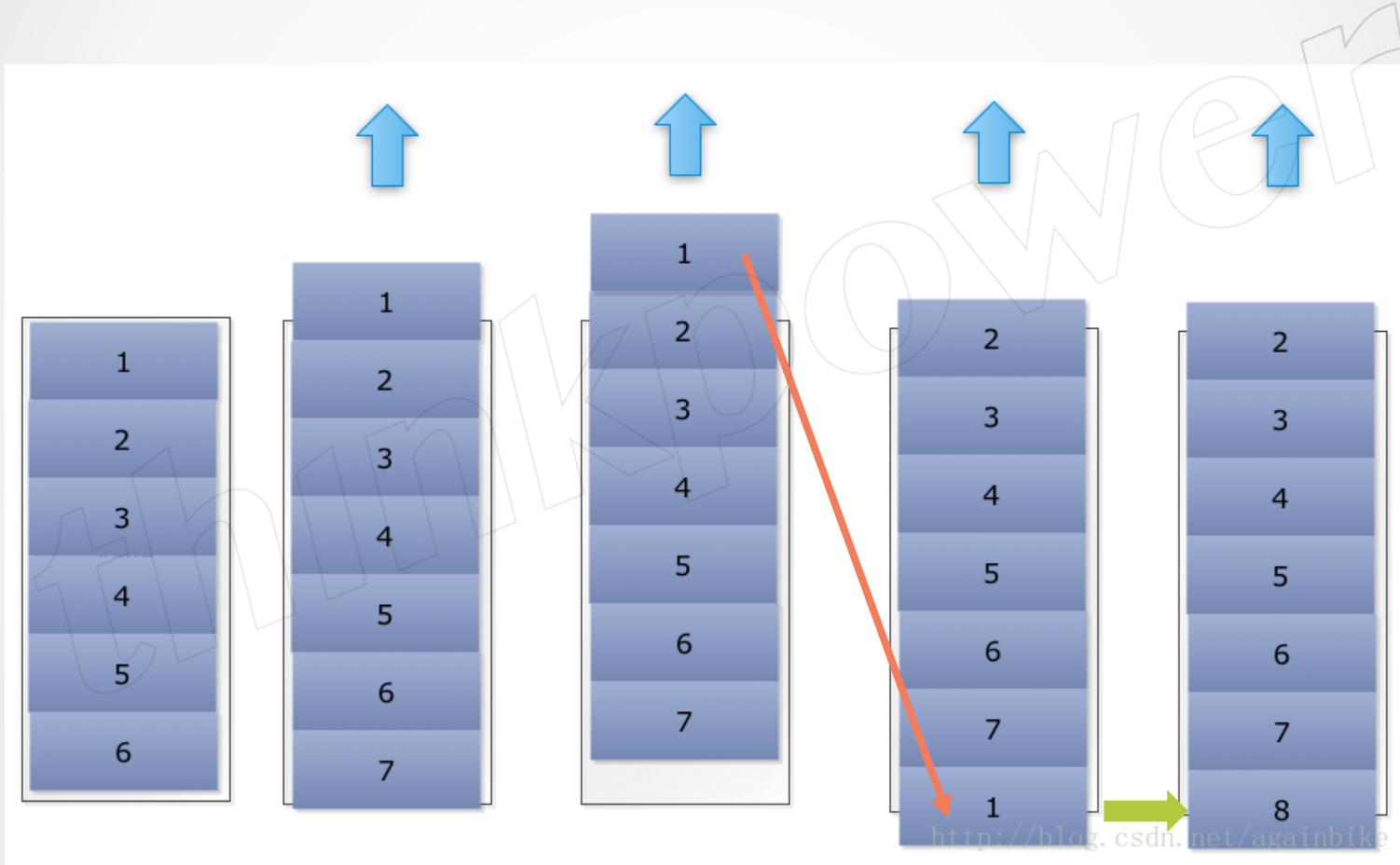
```
public class MySource : UITableViewSource
{
    0 個參考
    public override UITableViewCell GetCell(UITableView tableView, NSIndexPath indexPath)
    {
        throw new NotImplementedException();
    }

    0 個參考
    public override int RowsInSection(UITableView tableview, int section)
    {
        throw new NotImplementedException();
    }
}
```

TableView Cell重用原理

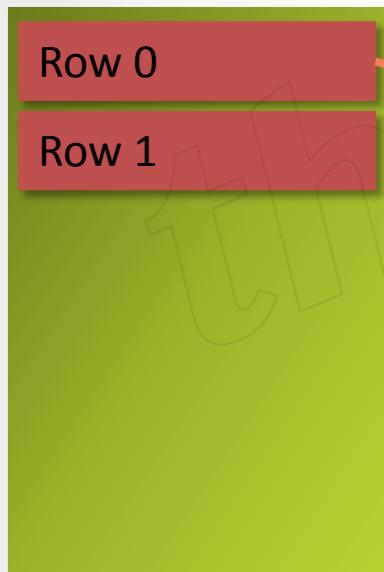
- TableView的每一個Row都是一個Cell，並且會有一個識別ID
- 假設資料100筆，當目前畫面只顯示10筆資料(10筆Cell)，則在記憶體中也只會建立10筆Cell
- 當TableView往下或上滑動時，進到視窗外的Row的Cell會被放置進TableView內的回收區域
- 每當新的Cell要顯示在畫面上時，都會到UITableViewSource的GetCell()中取要Cell，在此方法中可以透過識別ID向TableView要可重用的Cell

TableView Cell重用原理



實作GetCell()

在GetCell()中，會利用TableView的DequeueReusableCell()方法，給予Cell的識別ID後找回是否有可重複使用的Cell，並用方法傳入的參數NSIndexPath找到Cell於Table的索引值，NSIndexPath中提供了Section(群組索引)與Row(於群組中的Cell索引)，當DequeueReusableCell()傳回NULL代表無可重複使用的Cell。



```
public override UITableViewCell GetCell(tableView, indexPath)
```

```
{  
    var cell = tableView.DequeueReusableCell("ID");  
    if (cell == null)  
    {  
    }  
    return cell;  
}
```

透過ID要可重用的Cell

重用區

建立Cell

iOS預設提供了4種可用的Cell，提供三種欄位，屬性如下：

TextLabel
DetailTextLabel
ImageView



```
if (cell == null)
{
    cell = new UITableViewCell(UITableViewCellStyle.Default, _cellID);
    //cell = new UITableViewCell (UITableViewCellStyle.Subtitle, _cellID);
    //cell = new UITableViewCell (UITableViewCellStyle.Value1, _cellID);
    //cell = new UITableViewCell (UITableViewCellStyle.Value2, _cellID);
}
```

加入Accessory

我們還可以為Cell加入Accessory，並且撰寫當這個Accessory被點擊時要執行的程式碼。(Accessory會因iOS版本不同而變更)



```
cell.Accessory = UITableViewCellAccessory.Checkmark;
//cell.Accessory = UITableViewCellAccessory.DisclosureIndicator;
//cell.Accessory = UITableViewCellAccessory.DetailDisclosureButton;
```

```
public override void AccessoryButtonTapped(UITableView tableView, NSIndexPath indexPath)
```

滑動Cell刪除資料

於UITableViewSource中覆寫CanEditRow()與CommitEditingStyle()可以實作滑動出現刪除按鈕的效果，刪除按鈕預設文字為Delete，覆寫TitleForDeleteConfirmation()可變更按鈕文字。

```
public override bool CanEditRow(UITableView tableView, NSIndexPath indexPath)
{
    return true;
}

public override void CommitEditingStyle(UITableView tableView, UITableViewCellEditingStyle editingStyle)
{
    switch (editingStyle)
    {
        case UITableViewCellEditingStyle.Delete:
            Items.RemoveAt(indexPath.Row);
            tableView.DeleteRows(new NSIndexPath[] { indexPath }, UITableViewRowAnimation.Fade);
    }
}

public override string TitleForDeleteConfirmation(UITableView tableView, NSIndexPath indexPath)
{
    return "刪除" + Items[indexPath.Row];
}
```



覆寫 UITableView Cell 選取方法

52

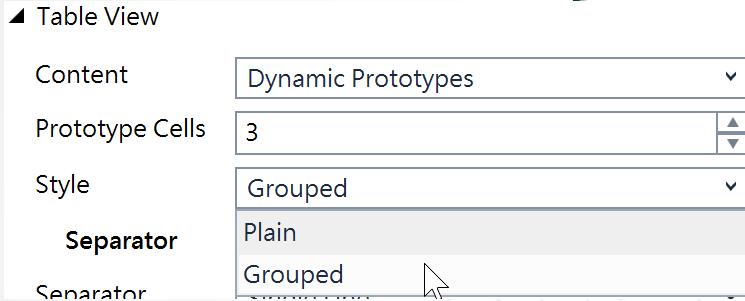
- UITableViewSource 提供了當Cell被選取時會觸發的方法，需自行覆寫
- 事件參數中會提供NSIndexPath，透過它可找到對應的群組和物件索引

```
public override void RowSelected(UITableView tableView, NSIndexPath indexPath)
{
    base.RowSelected(tableView, indexPath);
}
```

製作分組的TableView

當資料需要分組展現在畫面上時，也可讓TableView進行資料的分組

■ 更改TableView的屬性Style為Grouped



■ 於UITableViewSource覆寫NumberOfSections()，傳回群組數

```
public override int NumberOfSections(UITableView tableView)
{
    return Datas.Count;
}
```

製作分組的TableView

■ 改寫RowsInSection()，傳回指定群組的子數量

```
public override int RowsInSection(UITableView tableView, int section)
{
    //return Items.Count;
    return Datas[section].Name.Count(); //for group
}
```

■ 於UITableViewSource覆寫TitleForHeader()，群組的標頭

```
public override string TitleForHeader(UITableView tableView, int section)
{
    return Datas[section].Type;
}
```





客製化TableView Cell

客製化TableView Cell

除了使用預設的Cell外，還可以使用以下的方式自訂Cell

- 繼承UITableViewCell，在Cell中**手刻**所有控制項，並且指定X、Y軸的位置以及設定寬度、高度等...

```
public override void LayoutSubviews ()  
{  
    base.LayoutSubviews ();  
    AccName.Frame = this.isAlltotal ? new System.Drawing.RectangleF (20, 16, 102, 20) : new  
    System.Drawing.RectangleF (20, 20, 200, 20);  
    AccNo.Frame = new System.Drawing.RectangleF (20, 20, 200, 20);  
    AccTotal.Frame = new System.Drawing.RectangleF (160, 16, 102, 20);  
  
    btn.Frame = new System.Drawing.RectangleF (267, 10, 29, 31);
```

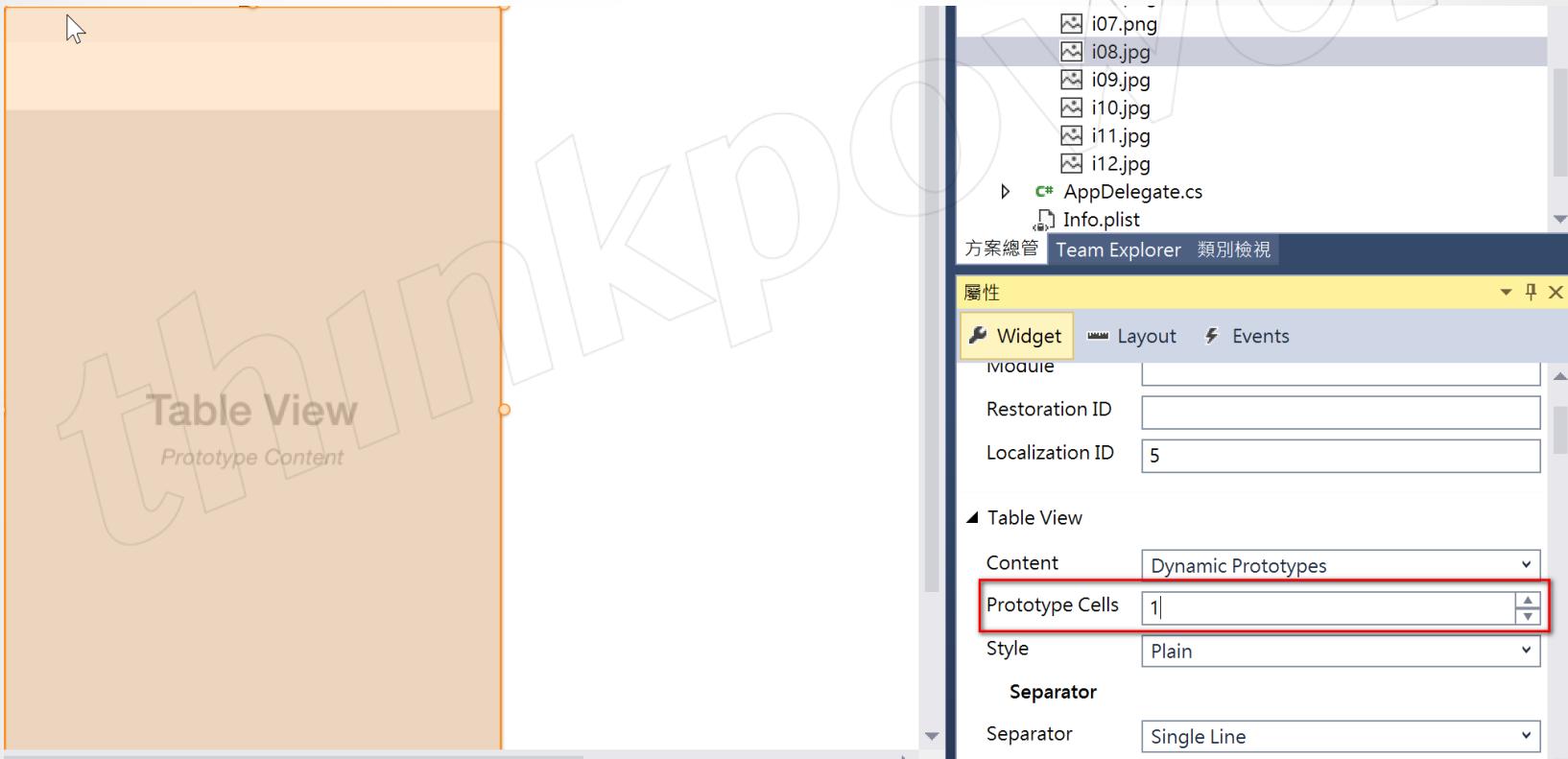
- 使用xib檔建立Cell，並註冊Cell給TableView

```
MyTable.RegisterNibForCellReuse(UINib.FromName("MyCell", NSBundle.MainBundle), "ID");
```

- 使用Storyboard建立客製化Cell

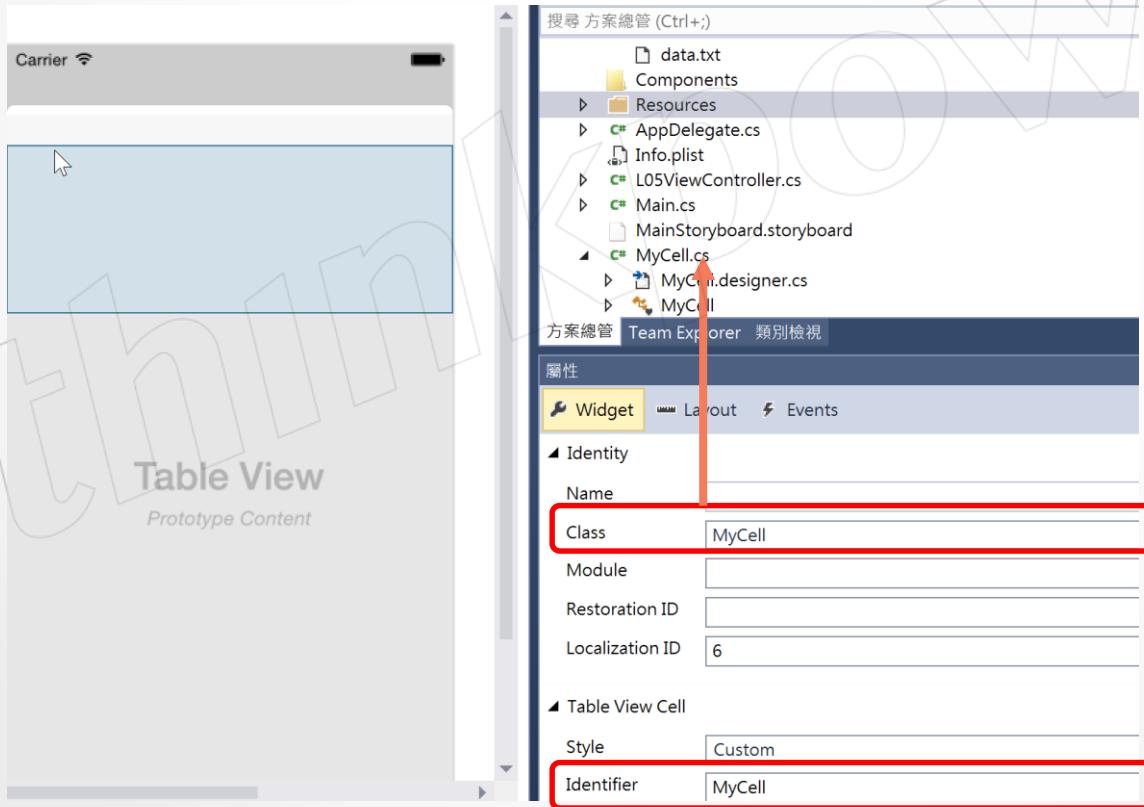
設定客製化Cell數量

在Storyboard中拉的TableView預設會有3個可客製的Cell，若只需要客製化1個Cell可將TableView的Prototype設為1



為Cell建立ID和類別

接著選到Cell上，因重複使用是靠Cell的ID做辨識，所以在此要設定屬性Identifier，並在屬性Class的地方為這個Cell建立一個類別，就可開始在這個Cell上進行客製化。



自訂的Cell類別

- 當我們為Cell上的控制項命名時，會自動產生對應的控制項物件到自訂的Cell類別內

```
partial class MyCell
{
    [Outlet]
    [GeneratedCode ("iOS Designer", "1.0")]
    3 個參考
    UIImageView img { get; set; }

    [Outlet]
    [GeneratedCode ("iOS Designer", "1.0")]
    3 個參考
    UILabel lbl { get; set; }
```

- 由於在Cell中自動產生的控制項都是非公開的(private)，可在Cell內建立一個方法讓資料傳入

```
partial class MyCell : UITableViewCell
{
    0 個參考
    public MyCell (IntPtr handle) : base (handle)
    {
    }

    0 個參考
    public void SetData(Data data)
    {
        this.lbl.... I
    }
}
```

在GetCell中取得自訂Cell

在GetCell()方法內同樣還是使用DequeueReusableCell()，而ID則使用前步驟在畫面上屬性Identifier的設定。於先前不同的是，這裡不需要再考慮是否重用的問題，因為系統會自動判斷，所以取出的Cell一定不會Null。

```

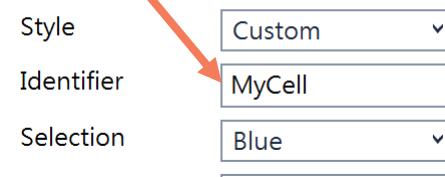
public override UITableViewCell GetCell(UITableView tableView, NSIndexPath indexPath)
{
    var cell = tableView.DequeueReusableCell("MyCell") as MyCell;
    cell.SetData(this.Items[indexPath.Row]);
    return cell;
}

partial class MyCell : UITableViewCell
{
    0 個參考
    public MyCell (IntPtr handle) : base (handle)
    {
    }

    0 個參考
    public void SetData(Data data)
    {
        this.lbl.... I
    }
}

```

▲ Table View Cell





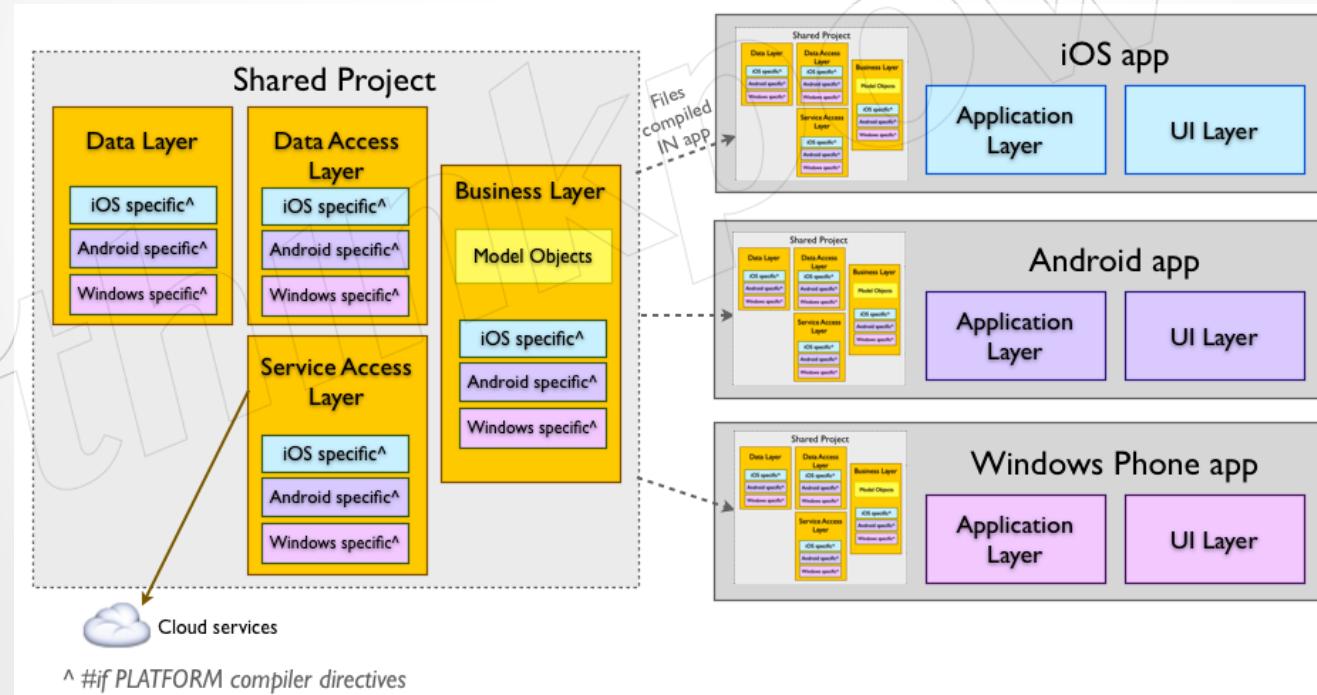
跨平台架構設計

在Xamarin中我們可以使用兩種方式讓程式碼共享至各個平台，再加上良好的程式架構設計，可以共享最多約70%左右的程式碼。

- 使用Shared Projects(意即共享檔案方式)來共享程式碼，並且可搭配條件式編譯來選擇每個平台要編譯的程式碼區塊(如有需要)。
- 使用可攜式類別庫(Portable Class Libraries)，可選擇多種平台共享程式碼，並可搭配介面(Interface)來實作各平台上不同程式碼但功能相同的部分(IoC-控制反轉)

Shared Projects

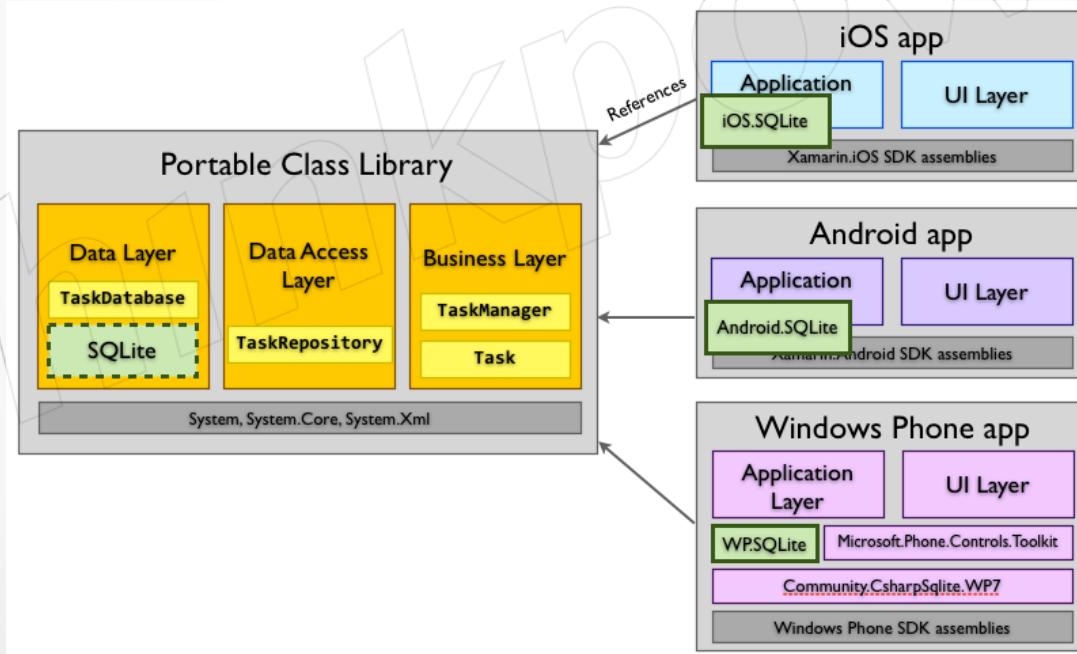
使用Shared Projects可以共享在多種專案類型中，並且可用條件式編譯的方式讓每個平台編譯出不同程式碼以編譯不同的framework，缺點是共享的檔案會被當成專案本身的程式碼編譯，無法單獨編譯出dll檔。



Portable Class Libraries

64

Portable Class Libraries簡稱為PCL，同樣可以在多種類型的專案中進行程式碼共享，但會有平台的限制，而PCL專案會單獨編譯出dll檔案，最大的缺點是它使用的framework會依照所選擇的平台有所限制，選擇越多則限制越多。



總結

65

在兩種方式中使用可攜式類別庫(PCL)是最簡單使用的方法，但他有一些缺點，部分常使用到的類別在PCL中是無法使用的(如File、WebClient等)，而有些類別可透過Nuget上下載安裝取得。兩種不同的方式將會影響到你專案整體的架構與寫法，應選擇最適合你專案中有效的方法。

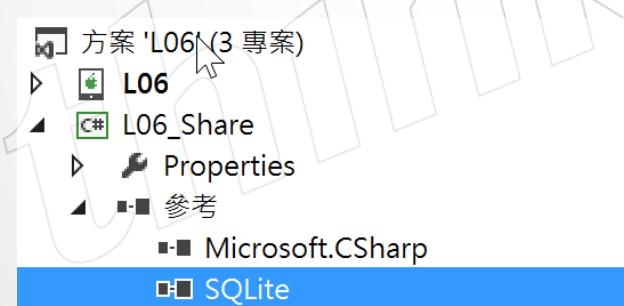


Sqlite.Net- With Share File

建立一般類別庫

因Sqlite在Android或iOS上的名稱空間及所有物件方法皆是相同的，所以我們可以使用Share File的方式將對Sqlite的操作都封裝在一個Class檔案內。

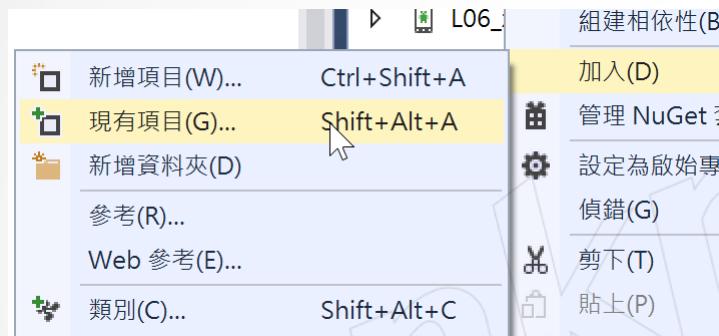
- 在iOS專案於Components Store下載Sqlite.Net
- 在方案中加入一個一般類別庫專案，並將剛剛下載的Sqlite.Net加入至參考



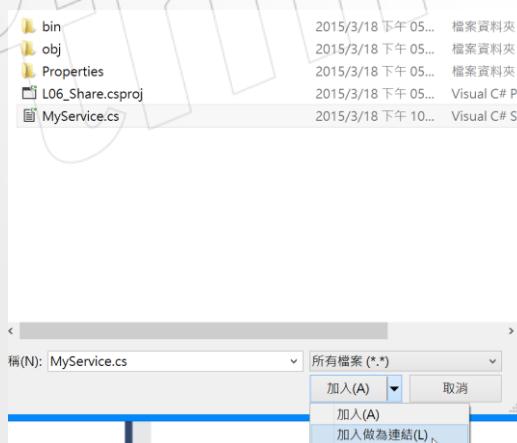
- 在一般類別庫專案中加入一個類別檔，於該類別內撰寫對Sqlite的操作

加入Share File

- 在iOS專案中加入現有項目，選擇於一般類別專案中剛剛建立的類別檔案。



- 加入檔案時選擇使用 加入做為連結即完成加入Share File



條件式編譯

- 加入的Share File可以直接把它當成專案內的類別檔，可直接建立該類別使用
- 使用條件式編譯可以讓不同的平台編譯出不同的程式碼
 Android: _ANDROID_
 iOS: _IOS_

```
#if XXX
    程式碼區塊
#endif
#if _ANDROID_
    dbName="droid.db";
#endif
#if _IOS_
    dbName="ios.db";
#endif
```

- 專案屬性內可自訂義條件式編譯的符號

一般

條件式編譯的符號(Y): _ANDROID1_

定義 DEBUG 常數(U)

定義 TRACE 常數(T)



使用WebService With PCL

建立可攜式類別庫

在PCL的專案中，一樣可以透過加入服務參考的方式將WebService參考進專案中。

■ 在Visual Studio中新增可攜式類別庫專案

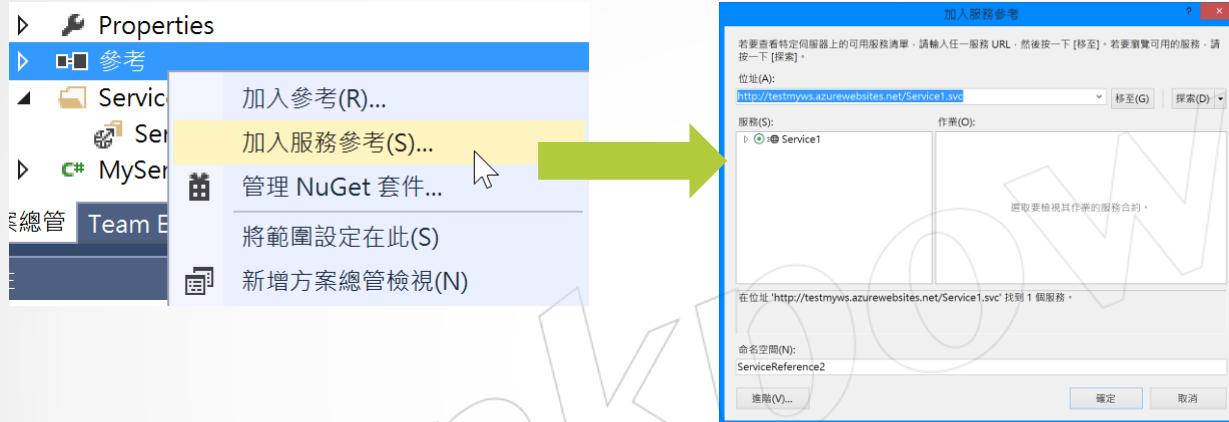


■ 選擇PCL要共用的專案平台，一定要搭配一個Windows平台

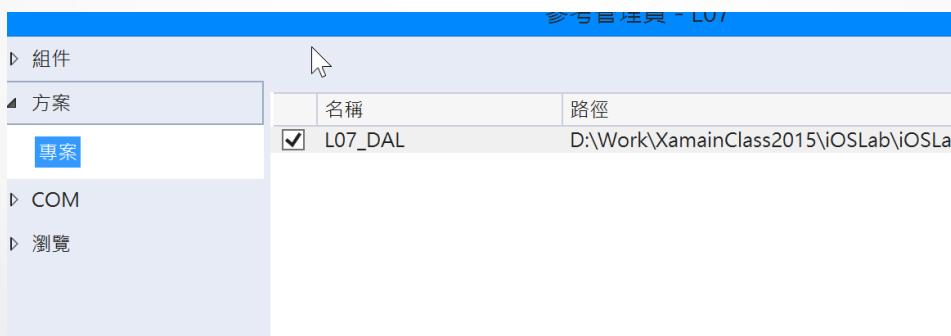


加入Web服務參考

■ 於PCL專案中加入服務參考，PCL使用的新版的Web服務參考



■ 於其他需要呼叫WebService的專案中，使用加入參考的方式將PCL專案加入專案參考



撰寫服務類別

- 接著就可以開始在PCL的專案中，新增一個類別將與WebService相關的部分封裝進去，我們同樣是使用由工具幫我們產生的proxy class幫我們與WebService聯繫

```
■ {} L07 DAL.ServiceReference1
  ▷ GetDataCompletedEventArgs
  ▷ •• IService1
  ▷ •• IService1Channel
  ▷ Service1Client
  ▷ Service1Client.EndpointConfiguration
  ▷ Service1Client.Service1ClientChannel
```

- 與以往做法不同的是，在PCL中並不會產生Web.config檔案，且 Xamarin也不支援system.configuration.*的類別，因此我們要自行建立HttpBinding與EndpointAddress設定

```
Service1Client service = new Service1Client(
    new BasicHttpBinding(BasicHttpSecurityMode.None),
    new EndpointAddress("http://testmyws.azurewebsites.net/Service1.svc"));
```

封裝方法

在PCL中加入的Web服務，只會產生非同步EAP模型的方法，為了讓外部呼叫這個類別時可以方便使用，因此通常會將它封裝為TAP的非同步模型，與在Android中的EAP方式相同都是使用TaskCompletionSource<T>

- ⌚ L07 DAL.ServiceReference1.Service1Client.GetDataAsync(int)
- ⌚ L07 DAL.ServiceReference1.Service1Client.GetDataAsync(int, object)
- ⚡ L07 DAL.ServiceReference1.Service1Client.GetDataCompleted

```
public Task<string> CallService(int value)
{
    TaskCompletionSource<string> task=new TaskCompletionSource<string>();
    service.GetDataCompleted += (sender, e) =>
    {
        task.SetResult(e.Result);
    };
    service.GetDataAsync(value);
    return task.Task;
}
```



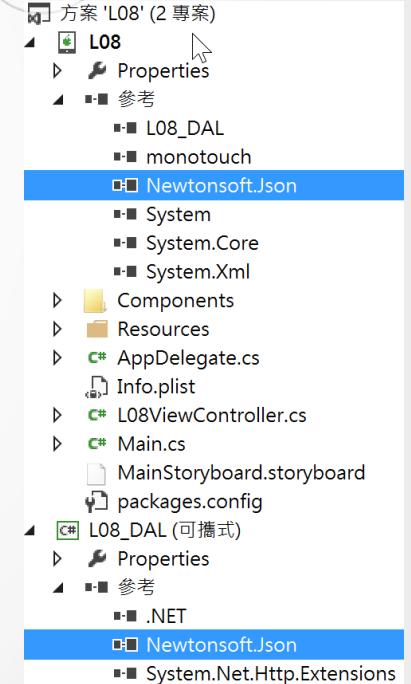
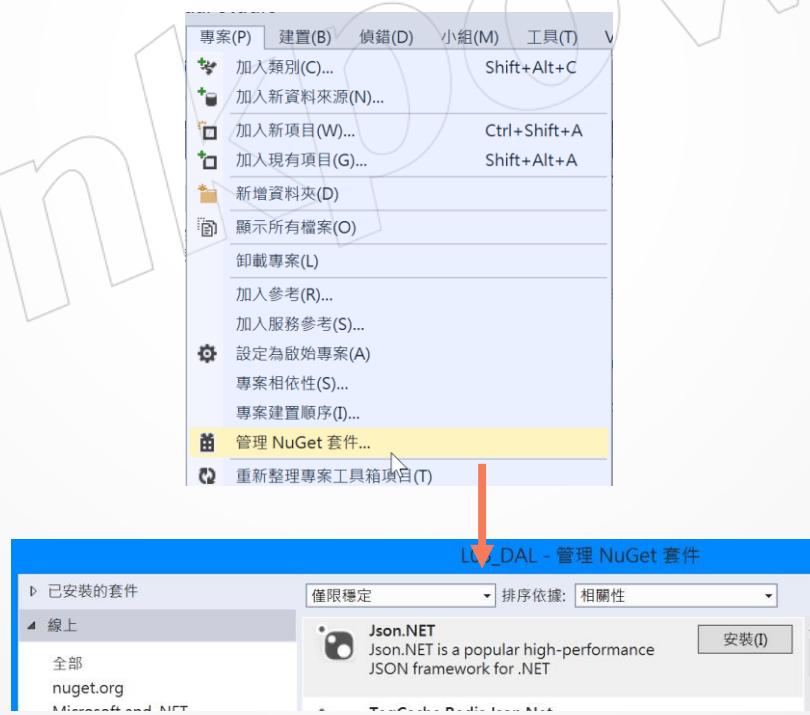
III 使用WebAPI+Json.Net With PCL

新增PCL專案

使用WebAPI+Json.NET我們同樣可以將與WebAPI溝通的部分封裝成一個方法，和Json的轉換與Json轉換後的Class都搬移到PCL專案內，在PCL中可用NuGet取得Json.NET或者直接參考iOS專案中取得的Json.NET元件，iOS專案也同樣需加入Json.NET。

```
public class APIService
{
    1 個參考
    public async Task<Info> CallAPI()
    {
        //呼叫WebAPI
    }
}

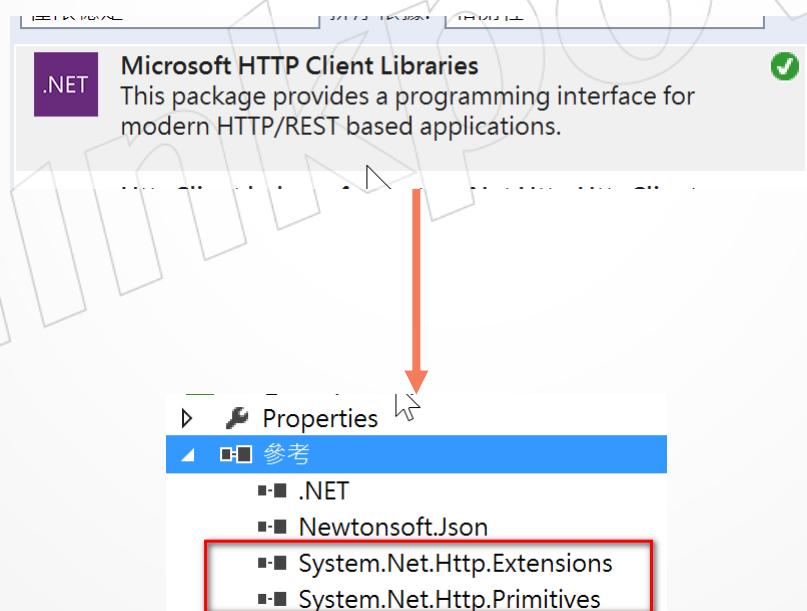
1 個參考
public class Info
{
    0 個參考
    public string Json { get; set; }
    0 個參考
    public string Name { get; set; }
    0 個參考
    public int Age { get; set; }
    0 個參考
    public string Work { get; set; }
}
```



取得HttpClient

77

在PCL專案由於會共享到其他Windows平台的專案(win8APP、Silverlight) · 所以會無法使用WebClient的類別 · 也沒有HttpClient可用 · 但這兩種類別在Xamarin.iOS下是可使用的 · 因此我們可在NuGet中下載HttpClient套件 ·



封裝為方法

使用HttpClient提供的非同步方法GetStringAsync()，就可以做到如 WebClient相同的效果，我們可以將呼叫API，將Json轉成物件的邏輯都封裝到一個方法內供外部呼叫使用。

```
public async Task<Info> CallAPI()
{
    HttpClient request = new HttpClient();
    var result = await request.GetStringAsync("http://testmyapi.azurewebsites.net/api/values/");
    var info=Newtonsoft.Json.JsonConvert.DeserializeObject<Info>(result);
    info.Json = result;
    return info;
}
```

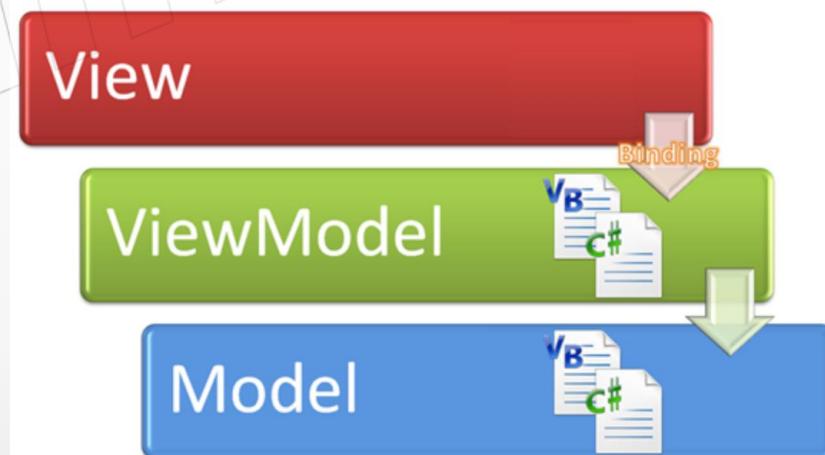


共享更多程式碼 - MVVM 模型

什麼是MVVM

MVVM的完整名稱是 **Model-View-ViewModel**，使用MVVM模型可以讓程式碼的共享率往上提升。

- View:依然指的是使用者介面，一般可用Binding的方式與ViewModel溝通。
- ViewModel:就是View所使用的Model，透過公開的屬性與View進行Binding，並與Model層溝通
- Model:同MVC的Model

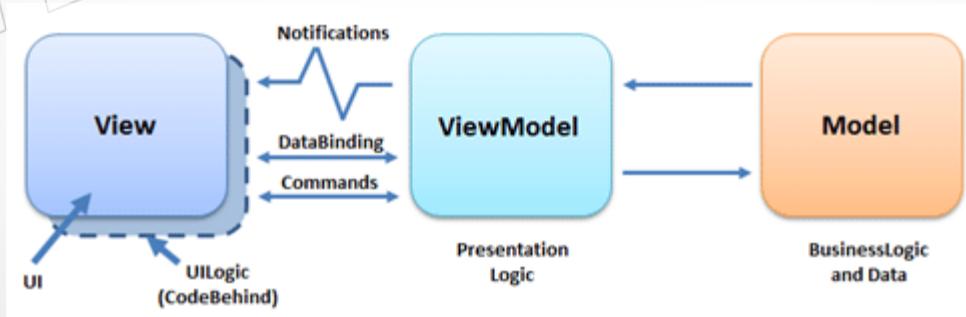


MVVM架構

MVVM基本上一個View會對應到一個ViewModel，而ViewModel內的公開屬性會透過Binding的方式與View做單向或雙向的溝通，舉例來說假設ViewModel有個屬性名為Name，在View中就會有一個Label與這個屬性做Binding。

當ViewModel需要發送通知給View時，可以使用事件或者是實作介面的方式通知View。

而在View上的控制向事件，如Button的Click等，則會用Command的方式告知ViewModel按鈕被Click了。



在Xamarin中套用MVVM

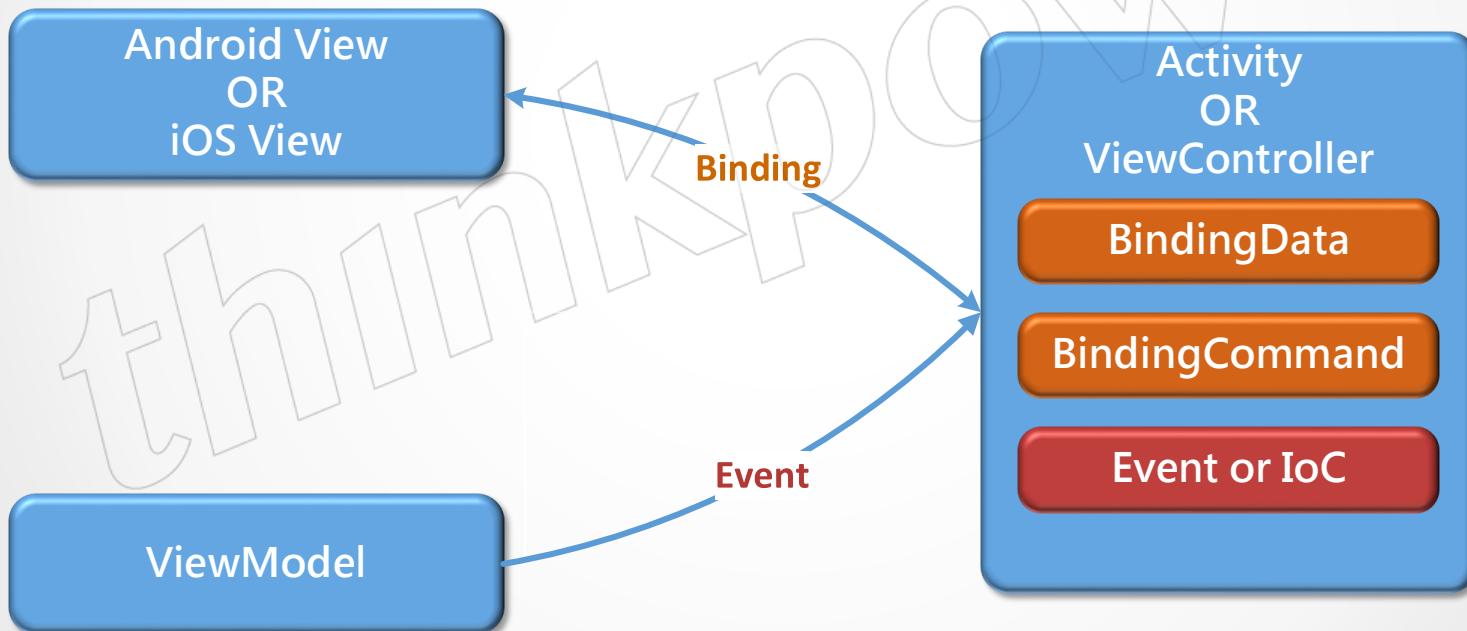
- 在Windows專案中，如WPF在設定Binding時，是在View的xaml檔的XML內直接撰寫Binding的動作。

```
<Grid>
    <Label Content="{Binding PostsTitle}" Height="28" HorizontalAlignment="Left"/>
    <Button Content="Button" Height="23" HorizontalAlignment="Left"
</Grid>
```

- 在Xamarin.iOS或Android專案內，我們無法直接對View去撰寫自行定義的xml(Android可以)，但可以在Controller內撰寫Binding的程式碼與註冊ViewModel的事件，也就是在Activity或者ViewController中只需關注於Binding程式碼與事件通知的部分。

在Xamarin中套用MVVM

當我們套用了MVVM架構後，基本上就是把Controller端用來註冊 ViewModel的事件，和BindindData與BindingCommand的動作，而其它如部分的畫面邏輯，畫面上的數值驗證等都可以在ViewModel中進行。



Xamarin可用的MVVM元件

目前在Xamarin中常見到MVVM元件有兩種：

- MVVC Cross

設定與使用上較為複雜，但他可直接在Android的xml內定義Binding
可參考：

<http://www.dotblogs.com.tw/toysboy21/archive/2014/05/21/145196.aspx>

- MVVM Light

使用程式碼在Controller端撰寫Binding的動作，但因該元件利用了物件反射的方式註冊事件與取得、設定屬性，因此程式碼最佳化選項無法使用，導致編譯出的APP過大。

若了解的MVVM的原理與架構，其實是可自行撰寫屬於自己的MVVM元件。

實作ViewModel

以Android的空白專案Hello World為例，首先我們可以得知第一個屬性就是按鈕上的文字，因此在ViewModel中就會有一個對應的屬性，並在set與get中撰寫MVVM元件提供的方法。

```
public class MainViewModel : ViewModelBase
{
    2 個參考
    public MainViewModel()
    {
        this.ButtonText = "Hello! MVVM, Click Me!";
    }
    4 個參考
    public string ButtonText
    {
        get { return this.GetProperty(() => this.ButtonText, _buttonText); }
        set { this SetProperty(() => this.ButtonText, value, ref _buttonText); }
    }
}
```

實作ViewModel

而按鈕按下的Click事件同樣也會Binding到ViewModel中，也就是BindingCommand，在ViewModel內建立一個由元件提供的Command類別物件，並撰寫Command被觸發後要執行的程式，在按鈕按下後會把一個計數的int+1並顯示到Button的文字上，所以可以直接對剛剛所建立的ButtonText屬性做修改，它就會自動Binding更新到畫面上。

```
public BindingCommand DoButtonClick
{
    get
    {
        if (_doButtonClick == null)
            _doButtonClick = new BindingCommand(() =>
        {
            this.ButtonText = string.Format("你按下了{0}次", _count++);
        });
        return _doButtonClick;
    }
}
```

撰寫Binding程式碼

以Android為例，在Activity中就會分別把按鈕的文字與Click事件Binding到ViewModel上。

```
MainViewModel model = new MainViewModel();
button.Click += CommandGenerate.BindingCommand(model.DoButtonClick);
model.AddBinding(c => c.ButtonText, () => button.Text, c => button.Text=c);
```

在iOS中的ViewController中也是做同樣的動作，但因為Button文字的設定、取得與事件不同，會有一些小差異

```
MainViewModel model = new MainViewModel();
button.TouchUpInside += CommandGenerate.BindingCommand(model.DoButtonClick);
model.AddBinding(c => c.ButtonText, () => button.TitleLabel.Text, c => buttonSetTitle( c, UIControlState.Normal));
```

MVVM帶來的好處

使用MVVM的設計模式，可以帶來很多好處

- 程式碼階層分割得更乾淨簡潔，依賴性也會降低
- 更好的關注點分離，設計介面的工程師只需專注於畫面於Binding之處理，而撰寫ViewModel的工程師可專注於UI邏輯的處理和與Model溝通的部分。
- 在ViewModel中共享更多的程式碼