

# iOS程式設計

讓應用程式更生動：進階元件

王昱景 Brian Wang

[brian.wang.frontline@gmail.com](mailto:brian.wang.frontline@gmail.com)

# ImageView 元件

- ImageView 介面元件的主要功能是顯示圖形
- 有了 ImageView 加持應用程式將更加美觀及生動

# UIImageView 元件常用屬性

Image	設定元件要顯示的圖形
Highlighted	設定需強調元件時顯示的圖形
Mode	設定圖形顯示模式。預設值為 Scale To Fill，作用是會自動縮放圖形以填滿 UIImageView 元件

- 圖形顯示模式
  - 應用程式中要顯示的原始圖形可能與 UIImageView 元件設定的顯示區域大小不同
  - 系統建立了 13 種圖形顯示模式 (Mode 屬性)
  - 開發者可依據需要使用適當的模式
  - Scale To Fill、Aspect Fit、Aspect Fill、Redraw、Center、Top、Bottom、Left、Right、Top Left、Top Right、Bottom Left、Bottom Right



- 圖形檔加入專案的方式
  - 應用程式的圖形檔最好加入專案成為專案的一部份，未來應用程式上架時就能包含在應用程式內
  - 第一種方式在 Finder 中選取要加入的圖形檔，再將這些檔案拖曳到 Assets.scassets 內
  - 第二種方式是在 Finder 中將要加入的圖形檔都置於同一資料夾，再將此資料拖曳到專案中

- 屬形面板設定圖形來源
  - 將圖形檔加入專案後，這些圖形檔案名稱會在屬性面板 Image 屬性下拉式選單中顯示
  - 開發者點選檔案名稱就可在 UIImageView 元件顯示該圖形

- 如果是複製資料夾方式加入圖形檔，Image 屬性會呈現附加檔名
- 若以 Assets.scassets 方式加入圖形檔，Image 屬性只呈現檔案名稱，不顯示附加檔名

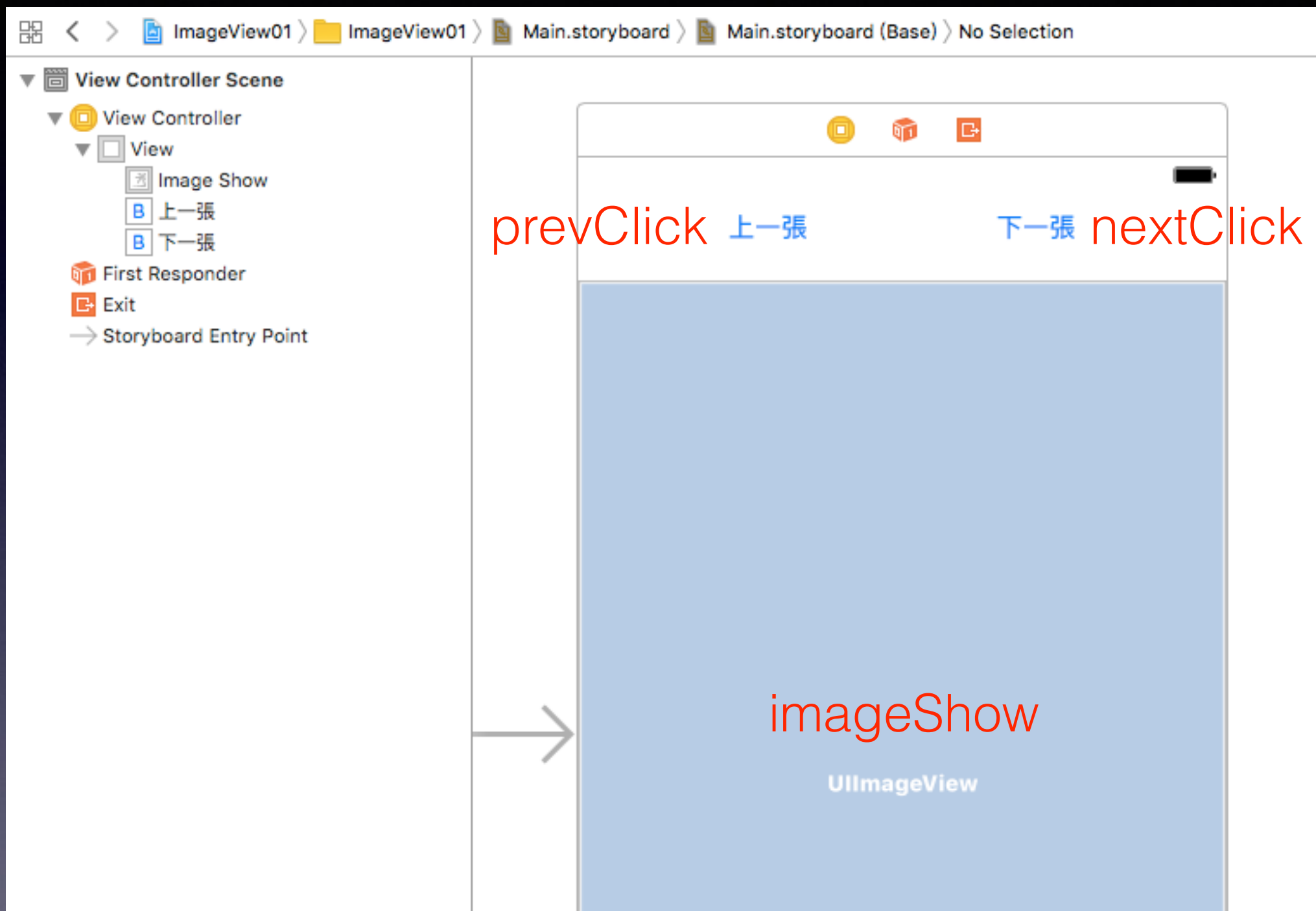
- 程式設定圖形來源
  - 使用屬性面板設定圖形來源固然很方便，卻無法任意改變顯示的圖形
  - 必須以程式設定圖形來源才能視需要改變顯示圖形

變數名稱.image = UIImage(named : 圖形檔案名稱)



# 圖片播放器

- 建立名稱為 ImageView01 的 Single View 專案
- 開始時顯示第一張圖片，按 下一張 鈕會切換到下一張圖片，若已達到最後一張就回到第一張
- 按 上一張 鈕會切換到上一張圖片，若已達到第一張就移到最後一張



```

1 import UIKit
2
3 class ViewController: UIViewController {
4     var arrayImage = ["img01", "img02", "img03", "img04", "img05", "img06"] //儲存圖片陣列
5     var p:Int = 0 //儲存目前圖片
6     var count:Int = 0 //儲存圖片總數量
7
8     @IBOutlet weak var imageShow: UIImageView! //ImageView元件
9
10    //按 上一張 鈕
11    @IBAction func prevClick(_ sender: UIButton) {
12        p -= 1 //目前圖片編號減一
13        if p < 0 { //如果編號小於0就設為最後一張
14            p = count - 1
15        }
16        imageShow.image = UIImage(named: arrayImage[p]) //顯示圖片
17    }
18
19    //按 下一張 鈕
20    @IBAction func nextClick(_ sender: UIButton) {
21        p += 1 //目前圖片編號加一
22        if p == count { //如果編號大於圖片總數就設為第一張
23            p = 0
24        }
25        imageShow.image = UIImage(named: arrayImage[p])
26    }
27
28    override func viewDidLoad() {
29        super.viewDidLoad()
30        imageShow.image=UIImage(named: "img01") //開始時顯示第一張
31        count = arrayImage.count //取得圖片總數
32    }
33
34    override func didReceiveMemoryWarning() {
35        super.didReceiveMemoryWarning()
36        // Dispose of any resources that can be recreated.
37    }
38
39
40 }
41
42

```

# 自動播放圖片

- UIImageView 元件提供數個自動播放圖片的方法及屬性
- 要自動播放圖片必須有圖片來源
- 因有多張圖片，所以圖片來源是陣列



- 先把所有圖片名稱儲存於一個名稱陣列中，再以 for 迴圈將圖形依序儲存於圖片陣列

```
for var i=0; i < 圖片數量; i++ {  
    圖片陣列.append (UIImage (named : 圖片名稱陣列 [i]))  
}
```

- UIImageView 元件儲存圖片來源的屬性是 animationImages 屬性

UIImageView 元件.animationImages = 圖片來源陣列

- 設定播放時間的是 `animationDuration` 屬性
- `animationDuration` 屬性指的是全部播放時間，而不是單張圖片播放時間，所以必須自行計算播放時間

`ImageView 元件.animationDuration = 播放時間`

- 最後是 `animationRepeatCount` 屬性設定播放次數
- 預設值為 0 表示無數次循環

`ImageView 元件.animationRepeatCount = 播放次數`

- 如此就完成自動播放圖片準備工作
- 執行 `startAnimating()` 方法就可開始播放
- 執行 `stopAnimating()` 方法就可停止播放

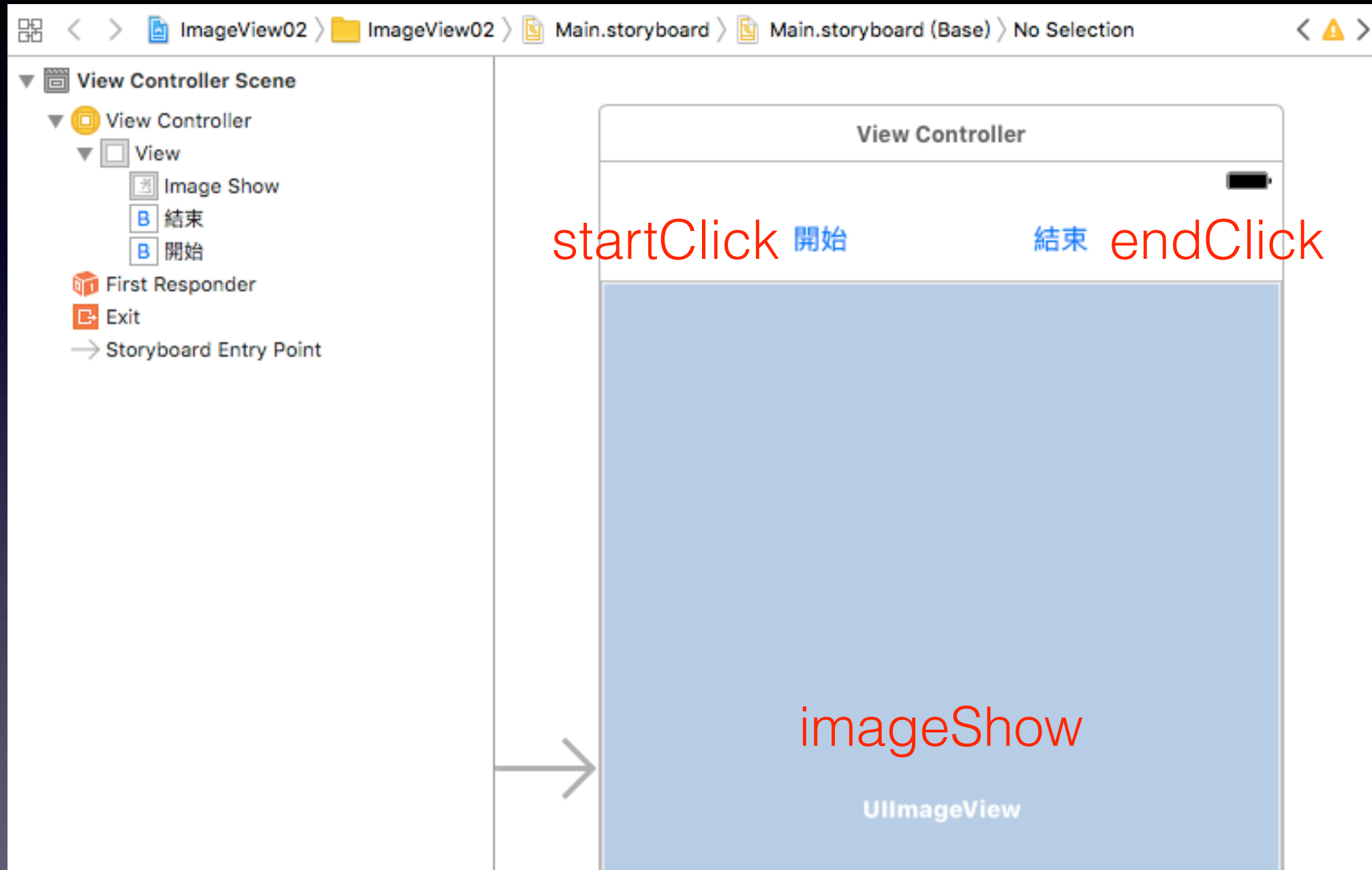
`ImageView 元件.startAnimating()` // 開始播放

`ImageView 元件.stopAnimating()` // 停止播放



# 自動圖片播放器

- 建立名稱為 ImageView02 的 Single View 專案
- 按 開始 鈕每隔 2 秒切換到下一張圖片
- 會循環播放，直到按 結束 鈕就停止播放，同時回到第一張圖片



```

1 import UIKit
2
3 class ViewController: UIViewController {
4     var arrayName = ["img01", "img02", "img03", "img04", "img05", "img06"] //儲存圖片名稱
5     var arrayImage:Array<UIImage> = [] //儲存圖片
6     var count:Int = 0 //儲存圖片總數量
7
8     @IBOutlet weak var imageShow: UIImageView! //ImageView元件
9
10    //開始播放
11    @IBAction func startClick(_ sender: UIButton) {
12        imageShow.startAnimating()
13    }
14
15    //結束播放
16    @IBAction func endClick(_ sender: UIButton) {
17        imageShow.stopAnimating()
18    }
19
20    override func viewDidLoad() {
21        super.viewDidLoad()
22
23        imageShow.image=UIImage(named: "img01") //開始時顯示第一張
24        count = arrayName.count //取得圖片總數
25        for i in 0 ..< count {
26            arrayImage.append(UIImage(named: arrayName[i]))
27        }
28        imageShow.animationImages = arrayImage //圖片來源
29        imageShow.animationDuration = TimeInterval(count * 2) //每2秒播放一張
30        //imageShow.animationRepeatCount = 1 //播放次數
31    }
32
33    override func didReceiveMemoryWarning() {
34        super.didReceiveMemoryWarning()
35        // Dispose of any resources that can be recreated.
36    }
37
38
39 }
40
41

```

# 圖形特效：框線、圓角及陰影

- UIImageView 元件預設範圍是長方形
- 建立 UIImageView 元件時，系統自動將元件置於圖層 (Layer) 中
- 可在圖層中設定屬性製作一些圖形特殊效果
- 為圖形加上框線、將四個直角修為圓角及為圖形加上陰影



- 框線

ImageView 元件.layer.borderColor = 顏色 // 框線顏色

ImageView 元件.layer.borderWidth = 數值 // 框線寬度

- 圓角

- 圓角圖形的原理是以 `cornerRadius` 屬性設定半徑畫弧，將圖層直角改為圓角
- 再使用圖層遮罩顯示圓角

`ImageView 元件.layer.cornerRadius = 數值 // 圓角角度`

`ImageView 元件.layer.masksToBounds = true // 圖層遮罩生效`

- 陰影

- 陰影可讓圖形感覺有立體效果
- 陰影效果可設定的屬性較多，包括顏色、位移、透明度及半徑

ImageView 元件.layer.shadowColor = 顏色 // 陰影顏色

ImageView 元件.layer.shadowOffset = CGSizeMake(x 位移, y 位移) // 陰影位移

ImageView 元件.layer.shadowOpacity = 數值 // 陰影透明度

ImageView 元件.layer.shadowRadius = 數值 // 陰影半徑

# 圖片加框、圓角及陰影

- 建立名稱為 ImageView03 的 Single View 專案
- 按 框線、圓角、陰影 鈕分別顯示對應特效圖片
- 按 正常 鈕則恢復原狀
- 呈 圓角 效果時，陰影 效果無效





```

1 import UIKit
2
3 class ViewController: UIViewController {
4     @IBOutlet weak var imageShow: UIImageView! //ImageView元件
5
6     //按 框線 鈕
7     @IBAction func borderClick(_ sender: UIButton) {
8         imageShow.layer.borderColor = UIColor.red.cgColor //框線顏色
9         imageShow.layer.borderWidth = 5 //框線粗細
10    }
11
12    //按 圓角 鈕
13    @IBAction func roundClick(_ sender: UIButton) {
14        imageShow.layer.cornerRadius = 30 //圓角角度
15        imageShow.layer.masksToBounds = true //圖層遮罩生效
16    }
17
18    //按 陰影 鈕
19    @IBAction func shadowClick(_ sender: UIButton) {
20        imageShow.layer.shadowColor = UIColor.blue.cgColor //陰影顏色
21        imageShow.layer.shadowOffset = CGSize(width: 10, height: 10) //陰影位移
22        imageShow.layer.shadowOpacity = 0.8 //陰影透明度
23        imageShow.layer.shadowRadius = 5 //陰影半徑
24    }
25
26    //按 正常 鈕
27    @IBAction func normalClick(_ sender: UIButton) {
28        imageShow.layer.borderWidth = 0 //移除框線
29        imageShow.layer.cornerRadius = 0 //移除圓角
30        imageShow.layer.masksToBounds = false //移除圖層遮罩
31        imageShow.layer.shadowOpacity = 0 //移除陰影
32    }
33
34    override func viewDidLoad() {
35        super.viewDidLoad()
36
37        imageShow.image=UIImage(named: "img01") //開始時顯示圖片
38    }
39
40    override func didReceiveMemoryWarning() {
41        super.didReceiveMemoryWarning()
42        // Dispose of any resources that can be recreated.
43    }
44
45 }
46
47

```

# DatePicker 元件

- DatePicker 介面元件讓使用者利用滾筒來選取日期及時間
- 滾筒捲動的動畫非常生動，視覺效果很好

# DatePicker 元件常用屬性

Mode	設定日期時間顯示模式。預設值為 Date and Time，表示日期及時間都會顯示
Locale	設定地區。預設值為 Default，是以英文顯示
Interval	設定顯示項目的時間間隔，範圍是 1 分鐘到 30 分鐘。預設值為 1 分鐘
Date	設定選取日期。預設值為目前日期
Constraints / Minimum Date	設定最小選取日期。須先核選此項目，再於下方輸入日期及時間；若未核選此項目，則無法輸入日期及時間
Constraints / Maximum Date	設定最大選取日期。須先核選此項目，再於下方輸入日期及時間；若未核選此項目，則無法輸入日期及時間
Timer	設定倒數計時的時間，單位為秒



- 日期時間顯示模式
  - Mode 屬性設定日期時間顯示模式有 Date and Time、Date、Time 及 Count Down Timer (倒數計時) 四種模式

- 地區

- 日期時間顯示的文字與 Locale 屬性所設定的 地區 有關，預設值為 Default，是以英文顯示日期時間
- 如果要以中文顯示，需修改 Locale 屬性值為 Chinese (Traditional, Taiwan)

- 項目時間間隔
  - Interval 屬性設定兩個項目之間的時間間隔
  - 單位是 分鐘，預設值為 1 minute

# 程式設定 DatePicker 屬性

- 在屬性面板設定屬性固然方便，卻有許多缺點
- 首先是每次拖曳建立元件時，都要一一設定各種屬性，開發者常會遺漏某個屬性設定
- 其次是閱讀他人程式時，不太可能檢視每個元件的每一個屬性，所以降低了程式可讀性



- 有些屬性並未列在屬性面板中，這些屬性必須利用程式才能設定
- 大部分有經驗的開發者，多會使用程式來設定元件各種屬性

- Mode 屬性
  - 模式有四種值：DateAndTime、Date、Time 及 CountdownTimer

DatePicker 元件.datePickerMode = UIDatePickerMode.模式

- Locale 屬性
  - DatePicker 元件顯示的文字會隨 Locale 屬性設定值而異
  - 不同語系其 地區代碼 不同，較常用的是 en 為英文、zh\_TW 為繁體中文、zh\_CN 為簡體中文、ja 為日文

DatePicker 元件.locale = NSLocale(localeIdentifier : "地區代碼")

- Date 及 MinimumDate 屬性
  - Date 屬性是設定目前選取日期，通常會設定程式開始執行的日期 (目前日期) 為 Date 屬性的設定值
  - Swift 中可利用 NSDate() 函數取得目前的日期



- MinimumDate 屬性可設定最小選取日期，許多應用程式不允許使用目前日期之前的日期，要達成此功能只要將 MinimumDate 屬性設定為目前日期即可

```
date1.date = NSDate()  
date1.minimumDate = NSDate()
```

- 日期時間顯示格式
- Swift 以 NSDateFormatter 類別來設定日期及時間的顯示格式
- 指定格式後再以 NSDateFormatter 類別的 StringFromDate 方法將日期型別轉換為字串顯示

- NSDateFormatter 類別設定格式有兩種方式，第一種方式是使用 `dateStyle` 設定日期，`timeStyle` 設定時間格式
- 此種方式是使用系統內建的格式

```
var 變數名稱 = NSDateFormatter()  
變數名稱.dateStyle = NSDateFormatterStyle.格式名稱  
變數名稱.timeStyle = NSDateFormatterStyle.格式名稱  
顯示字串 = 變數名稱.StringFromDate(DatePicker 元件.date)
```

格式	dateStyle	timeStyle
FullStyle	Tuesday, October 9, 2014	4:45:23 PM Taipei Standard Time
LongStyle	Tuesday, October 9, 2014	4:45:23 PM GMT +8
MediumStyle	Oct 9, 2014	4:45:23 PM
ShortStyle	10/9/14	4:45 PM
NoStyle	無	無



- 第二種方式是使用 `dateFormat` 設定日期時間格式
- `dateFormat` 是一個字串，字串以 6 個字母代表西元年、月、日、時、時、分、秒

字母	意義	備註
y	西元年	
M	月	月份是大寫 M
d	日	
h	時	
m	分	分鐘是小寫 m
s	秒	

- 一個字母代表一個數字，若字母不足，仍會顯示完整數值
- 若數字不足，會在數字值前面補 0

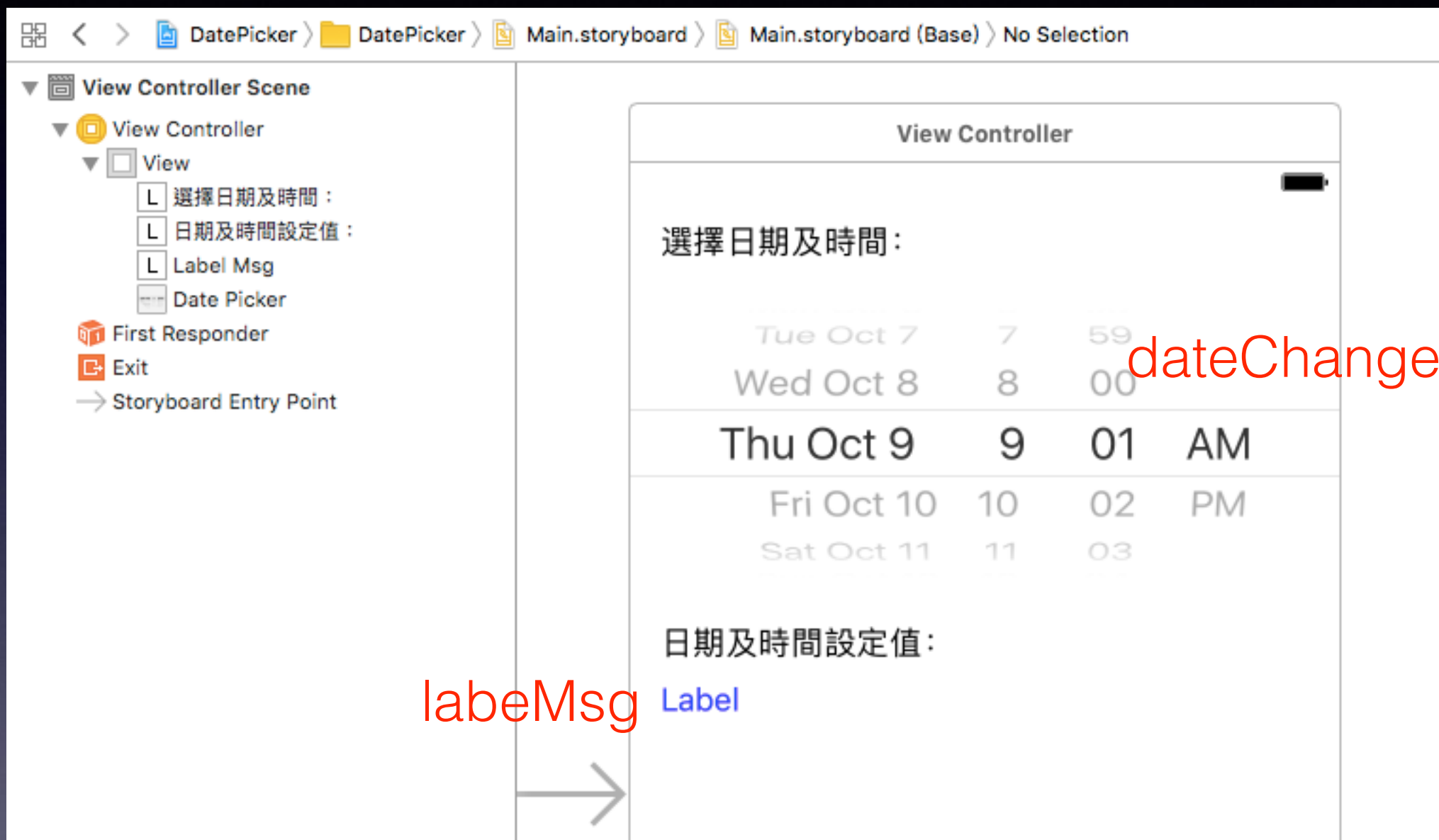
```
var 變數名稱 = NSDateFormatter()  
變數名稱.dateFormat = 格式字串  
顯示字串 = 變數名稱.stringFromDate(DatePicker 元件.date)
```

dateFormat 字串值	顯示
MM-dd-yyyy hh:mm:ss	09-14-2014 08:23:30
y/M/d h:m:s	2014/9/14 8:23:30
西元 y 年 M 月 d 日 hh 點 mm 分 ss 秒	西元 2014 年 9 月 14 日 08 點 23 分 30 秒



# 選取日期時間

- 建立名稱為 DatePicker 的 Single View 專案
- 開始時下方會顯示目前日期及時間
- 使用者在左方點選日期，也可使用捲動方式選取日期
- 點選或捲動右方則可選取時間
- 新日期及時間會立刻於下方更新



```
1 import UIKit
2
3 class ViewController: UIViewController {
4     var dateFormatter = DateFormatter() //宣告日期時間格式變數
5
6     @IBOutlet weak var datePicker: UIDatePicker! //DatePicker元件連結
7     @IBOutlet weak var labelMsg: UILabel! //顯示訊息連結
8
9     //使用者選取日期時間就更新顯示
10    @IBAction func dateChange(_ sender: UIDatePicker) {
11        labelMsg.text = dateFormatter.string(from: datePicker.date)
12    }
13
14    override func viewDidLoad() {
15        super.viewDidLoad()
16
17        datePicker.datePickerMode = UIDatePickerMode.dateAndTime //顯示模式
18        datePicker.locale = Locale(identifier: "zh_TW") //繁體中文
19        datePicker.date = Date() //開始時為現在日期及時間
20        dateFormatter.dateFormat = "西元y年M月d日 hh點mm分ss秒" //顯示格式
21        labelMsg.text = dateFormatter.string(from: datePicker.date)
22    }
23
24    override func didReceiveMemoryWarning() {
25        super.didReceiveMemoryWarning()
26        // Dispose of any resources that can be recreated.
27    }
28
29
30 }
31
32
```

# Stepper 元件

- Stepper 元件是以按鈕方式改變數值的工具
- 有 + 和 - 兩個按鈕
- 按 + 鈕會使 Current 屬性值（程式中稱為 Value 屬性）增加 Step 屬性值，按 - 鈕則使 Current 屬性值減少 Step 屬性值



# Stepper 元件常用屬性

Value / Minimum	元件可設定的最小值，預設值為 0
Value / Maximum	元件可設定的最大值，預設值為 100
Current	元件目前設定值，預設值為 0
Step	每次增加或減少的數值

- 圖型大小無法變更
- 系統預設 Stepper 元件的寬為 94 pixels，高為 29 pixels
- 此元件大小是固定的，不允許使用者變更

- Value 屬性 (Current 屬性)
- Stepper 元件的設定值並不會在介面顯示
- 通常開發者會配合 Label 元件，將 Stepper 元件的設定值在 Label 元件顯示

- 屬性面板中以 Current 設定 Stepper 元件值，  
程式中則是以 Value 屬性設定或讀取 Stepper  
元件值

Stepper 元件.value = 數值  
變數名稱 = Stepper 元件.value



# NSTimer 類別與倒數計時

- DatePicker 元件的 Model 屬性設為 Count Down Timer 時，可做為倒數計時器使用
- 應用程式中常需要定期持續執行一些固定的工作，例如時鐘，每一秒要更新顯示的時間
- Swift 以 NSTimer 類別來實現定期執行工作

- 第一步驟是建立全域變數儲存 NSTimer 物件

`var timer:NSTimer?`

- 第二步驟是建立 NSTimer 物件

`timer = NSTimer.scheduledTimerWithTimeInterval(1, target: self,  
selector: Selector("showTime"), userInfo: nil, repeats: true)`

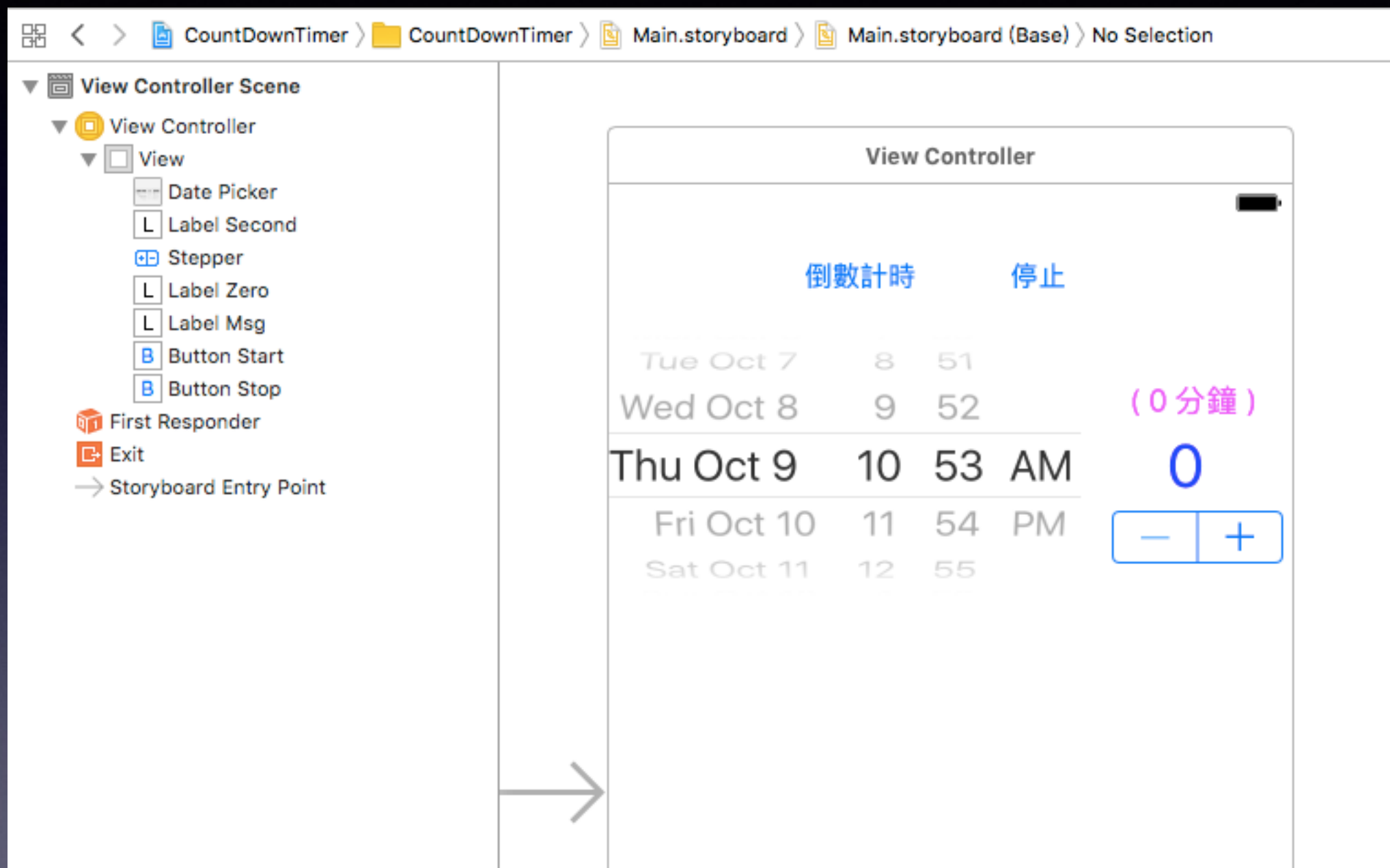
- 第三步驟是建立 NSTimer 物件執行的函數

# 倒數計時器

- 建立名稱為 CountdownTimer 的 Single View 專案
- 開始時預設為 1 分 0 秒
- 捲動或點選 DatePicker 元件可設定小時即分鐘
- 按 + 或 - 紐可增減秒數

- 設定完成後按 倒數計時 紐開始倒數計時
- 按 停止 紐可終止倒數計時，並將時間歸零
- 當時間小於 1 分鐘時，秒數上方會顯示 0 分鐘，告知使用者時間在 1 分鐘內
- 倒數計時結束時，下方會顯示 時間到！訊息





```

import UIKit

class ViewController: UIViewController {
    var timer:Timer? //定時執行變數
    var countTime:Int = 0 //倒數總秒數
    var countSec:Int = 0 //倒數時間的秒數部分

    @IBOutlet weak var buttonStart: UIButton! //開始鈕
    @IBOutlet weak var buttonStop: UIButton! //結束鈕
    @IBOutlet weak var datePicker: UIDatePicker! //DatePicker元件
    @IBOutlet weak var stepper: UIStepper! //Stepper元件
    @IBOutlet weak var labelSecond: UILabel! //顯示秒數
    @IBOutlet weak var labelMsg: UILabel! //顯示訊息
    @IBOutlet weak var labelZero: UILabel! //顯示 0分鐘

    @IBAction func stepperClick(_ sender: UIStepper) {
        labelSecond.text = "\(Int(stepper.value))" //顯示秒數
    }

    //按 倒數計時 鈕
    @IBAction func startClick(_ sender: UIButton) {
        labelZero.isHidden = true //不顯示 0分鐘
        buttonStop.isEnabled = true //停止 鈕有作用
        buttonStart.isEnabled = false //倒數計時 鈕無作用
        labelMsg.text = "" //清除訊息
        datePicker.isEnabled = false //不可選取時間
        stepper.isEnabled = false //不可設定秒數
        countSec = Int(stepper.value) //取得秒數部分
        countTime = Int(datePicker.countDownDuration) + countSec //計算總秒數
        timer = Timer.scheduledTimer(timeInterval: 1, target: self, selector: #selector(ViewController.
            countDown), userInfo: nil, repeats: true) //啟動定時執行
    }
}

```

```

//定時執行函數
func countdown() {
    countSec -= 1 //秒數部分減1
    countTime -= 1 //總秒數減1
    if countSec == -1 { //如果秒數部分需重置
        if countTime <= 0 { //時間到
            labelMsg.text = "時間到!" //顯示訊息
            stop() //停止計時
        } else { //重置秒數
            countSec = 59 //秒數部分設為59
            labelSecond.text = "\(countSec)"
            datePicker.countDownDuration = TimeInterval(countTime) //重置分鐘數
            if countTime < 60 { //如果在1分鐘內
                labelZero.isHidden = false //顯示 0分鐘
            }
        }
    } else {
        labelSecond.text = "\(countSec)"
    }
}

//按 停止 鈕
@IBAction func stopClick(_ sender: UIButton) {
    stop()
}

```

```

//停止計時
func stop() {
    buttonStop.isEnabled = false //停止 鈕無作用
    buttonStart.isEnabled = true //倒數計時 鈕有作用
    labelZero.isHidden = true //不顯示 0分鐘
    datePicker.isEnabled = true //可選取時間
    stepper.isEnabled = true //可設定秒數
    stepper.value = 0 //設定秒數為0
    labelSecond.text = "0" //顯示0秒
    timer!.invalidate() //停止定時執行
    timer = nil
}

override func viewDidLoad() {
    super.viewDidLoad()

    datePicker.datePickerMode = UIDatePickerMode.countDownTimer //倒數計時模式
    stepper.minimumValue = 0 //秒數設定最小值為 0
    stepper.maximumValue = 59 //秒數設定最大值為 59
    stepper.value = 0 //設定秒數為0
    labelSecond.text = "0" //顯示0秒
    buttonStop.isEnabled = false //停止 鈕無作用
    labelZero.isHidden = true //不顯示 0分鐘
}

override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    // Dispose of any resources that can be recreated.
}
}

```



# Switch 元件

- Switch 元件是很像開關的元件
- 可以在介面建立 開 / 關 按鈕
- 功能是用來控制布林 (Boolean) 資料型態

State	元件狀態，有 On 及 Off 兩種，預設值為 On
On Tint	元件狀態為 On 時，左方圓形的顏色，預設為綠色
Thumb Tint	元件狀態為 On 時，右方圓形的顏色，預設為白色

- 與 Stepper 元件相同，Switch 元件大小是固定的，不允許使用者變更
- 系統預設 Switch 元件的寬度為 51 pixels，高為 31 Pixels

# Slider 元件

- Slider 元件是一個像滑動電阻器的元件
- 其功能與 Stepper 元件類似，也是改變數值的工具
- Slider 元件只有一個按鈕，拖曳按鈕就可改變 Current 屬性值（在程式中稱為 Value 屬性）












Value/Minimum	元件可設定的最小值，預設值為 0
Value/Maximum	元件可設定的最大值，預設值為 1
Current	元件目前設定值，預設值為 0.5
Minimum Track Tint	元件按鈕左方的顏色，預設為藍色
Maximum Track Tint	元件按鈕右方的顏色，預設為灰色
Events/Continuous Updates	如果核選此項目，使用者在拖曳按鈕時，元件值會隨時更新；若未核選此項目，使用者在拖曳按鈕過程中元件值不會更新，直到放開按鈕時才更新，預設為核選此項目

# 圖片縮放

- 建立名稱為 Slider 的 Single View 專案
- 開始時圖片以原尺寸（100%）顯示
- 拖曳下方 Slider 元件的按鈕可改變 Slider 設定值，圖片及下方百分率會同步改變
- 最小尺寸為 30%

- 若按上方 Switch 元件會變為 關 狀態，同時 Slider 元件無法拖曳，故圖片無法改變大小
- 再按一次 Switch 原健會變為 開 狀態，同時 Slider 元件可以拖曳，圖片也可以改變大小

- ▼  View Controller Scene
  - ▼  View Controller
    - ▼  View
      -  Switch Photo
      -  Image
      -  Label Msg
      -  Slider Photo
    -  First Responder
    -  Exit
    - Storyboard Entry Point



View Controller









大小: 100%



```

import UIKit

class ViewController: UIViewController {
    var imgWidth:CGFloat = 0 //圖片寬度
    var imgHeight:CGFloat = 0 //圖片高度

    @IBOutlet weak var switchPhoto: UISwitch! //Switch元件
    @IBOutlet weak var image: UIImageView! //ImageView元件
    @IBOutlet weak var sliderPhoto: UISlider! //slider元件
    @IBOutlet weak var labelMsg: UILabel! //顯示圖片大小

    //Switch元件On時才可改變圖形大小
    @IBAction func switchChange(_ sender: UISwitch) {
        if switchPhoto.isOn {
            sliderPhoto.isEnabled = true
        } else {
            sliderPhoto.isEnabled = false
        }
    }

    //拖曳Slider元件改變圖形大小
    @IBAction func sliderChange(_ sender: UISlider) {
        if switchPhoto.isOn {
            image.frame.size.width = imgWidth * CGFloat(sliderPhoto.value)
            image.frame.size.height = imgHeight * CGFloat(sliderPhoto.value)
            labelMsg.text = "大小:\(Int(sliderPhoto.value * 100))%"
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        imgWidth = image.frame.size.width //取得圖形原始寬度
        imgHeight = image.frame.size.height //取得圖形原始高度
        sliderPhoto.frame.size.width = 280 //Slider元件寬度
        sliderPhoto.minimumValue = 0.3 //Slider元件最小值
        sliderPhoto.value = 1 //Slider元件初始值:100%
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}

```

# Progress View 元件

- 應用程式執行一個需花費一段時間的工作時，開發者常會在介面中安排一個 進度條
- 告知使用者目前程式執行到何種程度
- 避免使用者以為程式未在執行而離開應用程式
- 此 進度條 就是以 Progress View 元件完成

- Progress View 元件是兩個顏色的線形元件
- 功能與 Slider 元件類似
- 可以顯示元件值
- 但 Progress View 元件沒有按鈕，無法與使用者互動
- 只能以程式設定元件值









Style	有 Default 及 Bar 兩種，預設值為 Default
Progress	元件目前設定值，預設值為 0.5
Progress Tint	元件左方的顏色，預設為藍色
Track Tint	元件右方的顏色，預設為灰色

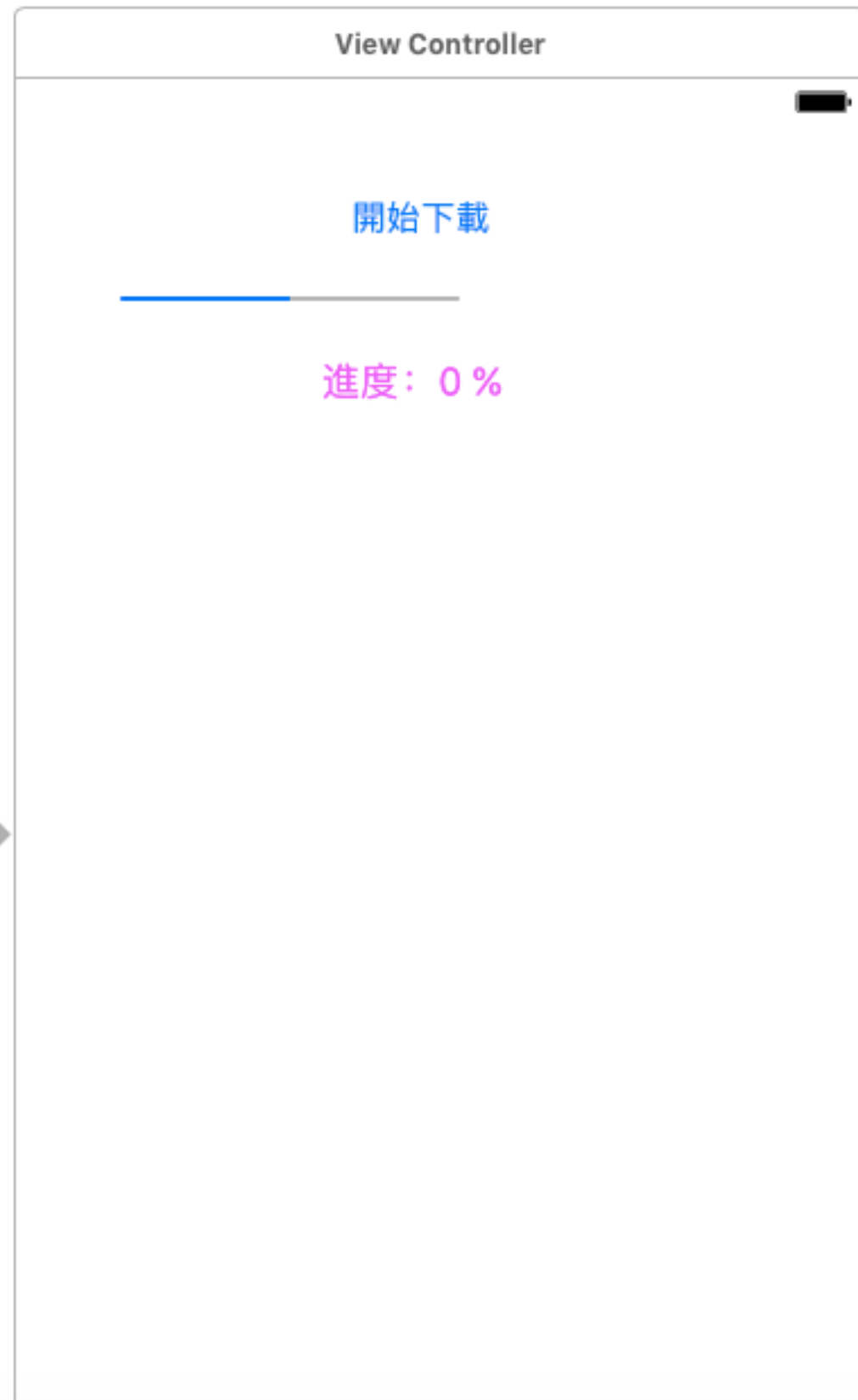


- 系統預設 Progress View 元件的寬為 150 pixels，高為 2 pixels
- Progress View 元件的寬度可以改變，但高度無法變更

# 進度條

- 建立名稱為 Progress 的 Single View 專案
- 開始時下載進度為 0
- 按 開始下載 鈕後每 0.1 秒進度增加 1%
- 同時 開始下載 鈕變為無作用，直到全部下載 (100%) 才恢復有作用

- ▼  View Controller Scene
  - ▼  View Controller
    - ▼  View
      -  Button Start
      -  Label Msg
      -  Progress
    -  First Responder
    -  Exit
    - Storyboard Entry Point



```
import UIKit

class ViewController: UIViewController {
    var timer:Timer? //計時器
    var count:Int = 0 //計數

    @IBOutlet weak var progress: UIProgressView! //ProgressView元件
    @IBOutlet weak var labelMsg: UILabel! //顯示進度
    @IBOutlet weak var buttonStart: UIButton! //開始鈕

    //按 開始 鈕
    @IBAction func startClick(_ sender: UIButton) {
        buttonStart.isEnabled = false //讓 開始 鈕失效
        count = 0 //計數歸零
        timer = Timer.scheduledTimer(timeInterval: 0.1, target: self, selector: #selector(ViewController.
            showProgress), userInfo: nil, repeats: true) //啟動計時器
    }

    //定時執行函數
    func showProgress() {
        progress.progress = Float(count) / 100 //設定進度
        labelMsg.text = "進度: \(count)%" //顯示進度
        count += 1 //進度加1
        if count > 100 { //若進度大於100就結束
            timer!.invalidate() //停止計時器
            timer = nil
            buttonStart.isEnabled = true //讓 開始 鈕有作用
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        progress.frame.size.width = 280 //ProgressView元件寬度
        progress.progressTintColor = UIColor.red //進度顏色
        progress.trackTintColor = UIColor.green
        progress.progress = 0 //起始值為0
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```