序

+ preface

拜 Apple Watch 於 2015 年 4 月問世之賜，也在《iOS 8 程式設計實戰》要進入再刷之時，我決定把 Apple Watch 的程式設計技術跟著放進書中，成為《iOS 8 + Apple Watch 程式設計實戰》。雖然我相信許多熱血的工程師早就已經在網路上或是透過官方文件把需要的技術摸熟了，但是在全球瘋狂關注 Apple Watch 的時刻，台灣卻沒有一本完整的繁體中文技術資料，總覺得些許遺憾也認為沒這個道理。所以本來應該是在 iOS9 公布之後才要開始進入忙碌的日子，就提前在三、四月展開。

在原本 iOS8 一書中放入 Apple Watch 的決定，其實要謝謝好友游鴻志先生，年後閒聊時希望我盡快把 Apple Watch 的技術出版成冊，雖然最後不是單獨出書，但也剛好趕上 iOS8 再刷的便車。另外要謝謝碁峰出版企劃部，提供再版方式許多重要的建議。

最後提醒讀者，這一本與《iOS8 程式設計實戰－205 個快速上手的開發技巧》的差異，主要就是增加 Apple Watch 單元與一些已經發現的錯誤修正，其他內容並沒有進行大規模的調整。

朱克剛

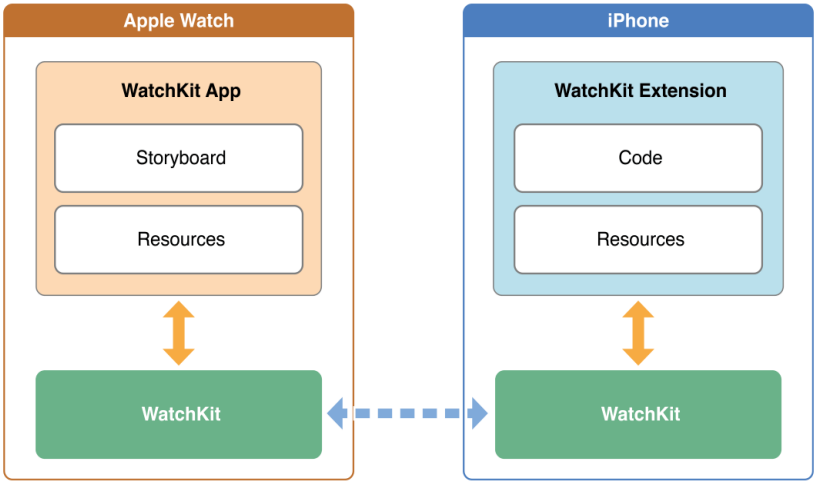
2015 夏初

chapter

22 Apple Watch 與 Extension

Apple Watch

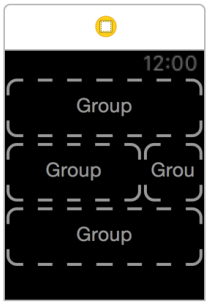
在 Apple Watch 上執行的 App 稱為 Apple Watch App，底層透過 WatchKit framework 提供所有 App 執行時需要的功能，故也稱為 WatchKit App。根據 Apple 公布的 Apple Watch 架構（如下圖所示），擁有 Apple Watch 的使用者想要發揮這支手錶的完整功能， iPhone 必須伴隨在旁。在開發 WatchKit App 時，Xcode 建立的專案會分為兩部分：WatchKit App（圖中左邊部分）與 WatchKit Extension（圖中右邊部分）。前者是在 Apple Watch 上執行，而後者是在 iPhone 上執行。

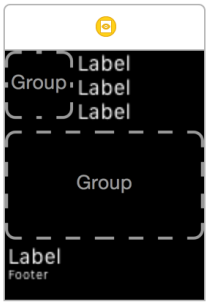


從上圖中亦可知，WatchKit App 部分還包含了 Storyboard 與資源檔案，因此，這部分負責手錶上的使用者介面以及與使用者互動；圖右的 WatchKit Extension 則是撰寫程式碼的地方，例如當使用者按下按鈕後所要執行的程式碼就是寫在 WatchKit Extension 中。這樣的架構表示了 iPhone 負責 WatchKit App 的邏輯運算，也就是當手錶上的 App 要運算 1+1=2 時（舉例而言），iPhone 負責四則運算，然後再將運算結果 2 傳送到手錶上顯示。當然也不是所有的功能都需要 iPhone 在身邊，那些需要 iPhone 在旁邊的 App 都是第三方開發的，大部分內建的 App 比較沒有這個問題。順道一提，目前的 Apple Watch 沒有包含 GPS 硬體，因此所有需要 GPS 定位的 App 都需要 iPhone 在旁才能發揮功能。

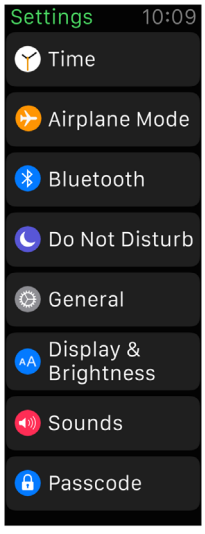
WatchKit framework 讓我們可以撰寫三種不同的手錶應用程式：（一）、標準的 App：可以透過互動元件（例如按鈕）跟使用者互動，並且具備大量且複雜的頁面切換能力；（二）、Glance：一個僅能用來顯示訊息的唯讀畫面，並且在畫面上無法放置互動元件（例如按鈕或 Scroll Bar），因此使用者無法操作它。Glance 的唯一目的就是讓使用者快速讀取訊息而已；（三）、推播：當本地或遠端推播訊息送到 iPhone 時，可以轉送到手錶上來顯示。以上這三項功能在 WatchKit App 的 Storyboard 中都有對應的 Interface Controller 可協助我們快速處理。

WatchKit 所提供的視覺化元件目前還不算太多，其中 Group 元件是專門用來排版的。它可以用來承載其它的元件，當然也包含另外一個 Group 元件。Group 元件的排版模式有兩種：水平排版與垂直排版。水平排版代表所有承載的元件都以水平方向排列，而垂直自然就是所有承載的元件都以垂直方向排列。透過屬性設定，我們還可以決定每個元件所佔的百分比，例如水平方向有兩個元件，左邊的佔 70%寬度，右邊佔 30%寬度。Group 中再放另外一個 Group 可以讓我們設計出更複雜的排版，但請特別留意，手錶的螢幕很小，過度複雜的畫面會讓使用者看得很辛苦。

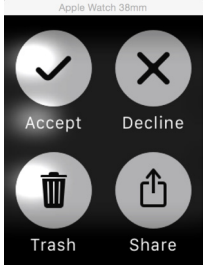




在水平排列上，Apple 建議每一列最好不要超過三個元件，按鈕的話盡量以圖示代替上面顯示的文字，這樣會讓整個版面看起來是舒適的。



使用頻率較低的按鈕，可以盡量以情境選單（Context Menu）來取代傳統的按鈕，這樣就不會佔據寶貴的螢幕空間。情境選單的啟動是藉由 Force Touch 功能觸發的，這是 Apple 最新的技術，也就是螢幕可以感測到使用者是輕點還是重壓，如果是重壓就彈跳出一個最多包含四個按鈕的全螢幕畫面讓使用者可以點選那些按鈕。

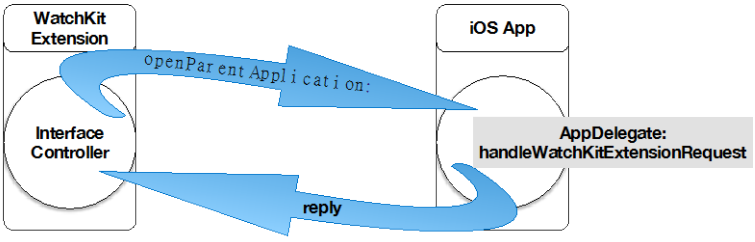


雖然 WatchKit 可以讓開發者加入自行設計的字型，不過 Apple 希望我們用內建的字型，原因是內建的字型已針對 Apple Watch 最佳化，在小螢幕的 Apple Watch 上依然能夠清楚的顯示內容，並且支援動態調整（Dynamic Type），也就是會根據元件的大小動態改變字型大小，永遠保持最佳顯示狀態。

iOS App 與 WatchKit App 之間如果要共享資料，必須設定 iOS App 與 WatchKit Extension 為同一個共享群組 ID 才行。對 iOS 而言，iOS App 與它的 Extension 是在兩個不同的沙盒中執行，所以沙盒與沙盒間的資料是不能互通的，必需要把資料儲存一個共用區兩邊才能互相存取的到。在 Xcode 專案的 Capibilities 頁面的 App Groups 項目中，分別打開 iOS App 與 WatchKit App 的 App Groups，然後設定一個相同的 Group ID，例如 group.com.ckk.watchkittest，之後就可以透過這個 ID 來共享同一份資料。

共用區可以放的資料類型有兩種：一為實體檔案；另一為偏好資料（preferences data），偏好資料的資料型態為 NSUserDefaults，結構為 key-value 形式。

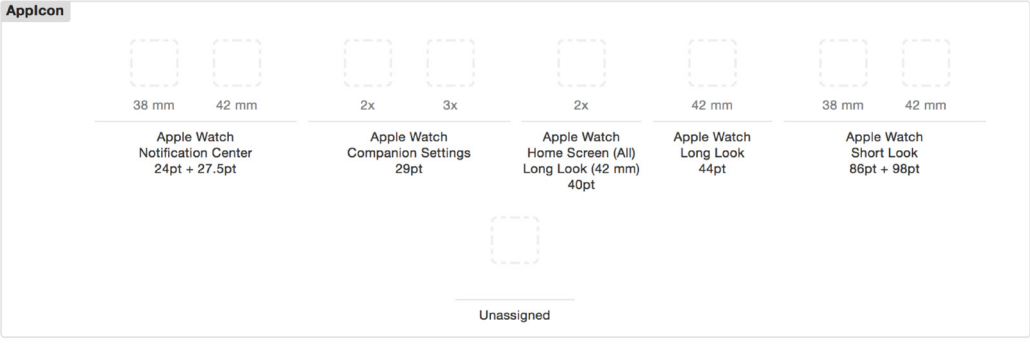
除了共用區之外，WatchKit App 與 iOS App 還可以透過一條特殊通道，將資料以字典（Dictionary）形式互相傳送，這條通道只會在 WatchKit Extension 以及包含它的 iOS App 間出現，啟動方式必須由 WatchKit Extension 主動呼叫 openParentApplication:reply:方法，呼叫後 iOS App 的 AppDelegate 類別中的 application:handleWatchKitExtensionRequest:reply:方法會收到，並且在這個方法中將值回傳給 WatchKit Extension。



現在來說明一下 Apple Watch 所需要的圖示大小，如下表：

圖示種類	Apple Watch (38mm)	Apple Watch (42mm)
Notification Center icon	48 pixels	55 pixels
Long-Look notification icon	80 pixels	88 pixels
Home Screen icon	80 pixels	80 pixels
Short-Look icon	172 pixels	196 pixels

注意上表使用的單位是 pixel，但是在 Xcode 中使用的單位是 point，而 pixel 的數字是 point 的@2x（兩倍）。下圖為 Xcode 中所需要的各式圖檔大小，請讀者自行對照上表比較。



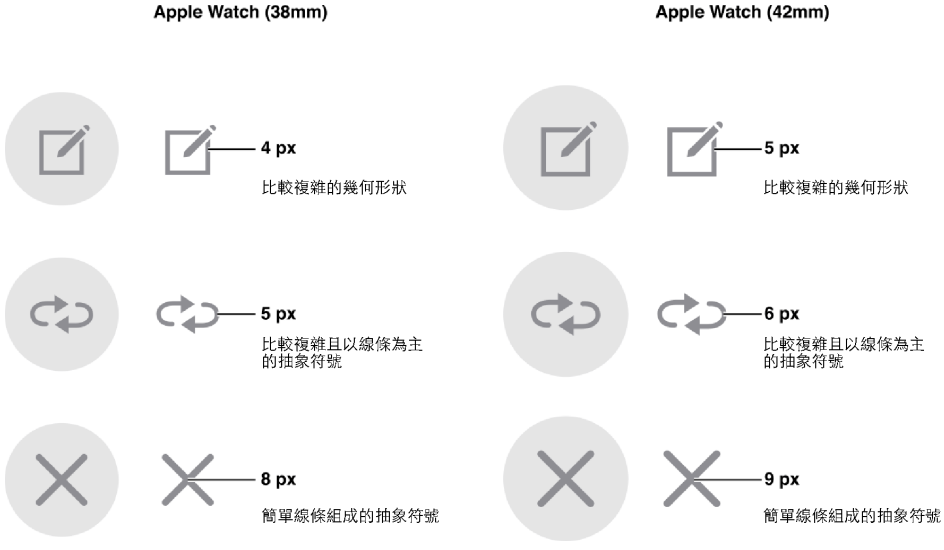
除此之外，在 iPhone 上對應的 Apple Watch App 所需要的圖示規格如下：

Asset	@2x	@3x
App icon	58 pixels	87 pixels

情境選單上的按鈕圖案除了有內建的外，也可以自訂但只能單色。Apple 對自訂圖案的大小與線條粗細有建議值。下表顯示按鈕所在的畫布大小（Canvas size）與按鈕上的圖案大小（Content size）建議值。

Device	Canvas size	Content size
Apple Watch (38mm)	70 pixels	46 pixels
Apple Watch (42mm)	80 pixels	54 pixels

下圖為按鈕上的圖案線條粗細建議值，例如左上角表示 38mm 的蘋果手錶，如果情境選單上的圖案是複雜的幾何形狀，建議線條粗細為 4px。按照這些建議值所畫出來的圖案會讓顯示時是清楚且容易辨識的。另外，圖檔格式請使用 PNG 格式。



當 iPhone 不在身邊時，凡是跟 GPS 或是通訊有關的功能（包含 Siri、email 或遠距推播…等）都無法使用，那 Apple Watch 還能做哪些事情？

1. 顯示時間、查看行事曆或是設定鬧鐘。
2. 播放音樂與照片瀏覽。當然這些音樂與照片是之前透過手機傳過去的。換句話說，所以如果手機不在身邊，那些沒事先傳過去的音樂或照片，手錶自然無法播放或顯示。
3. 追蹤身體健康狀況，例如偵測心跳、計步。數據會先儲存在手錶中，等到與 iPhone 連線時才會將資料儲存到手機的健康 App 中。但因為手錶上沒有 GPS 裝置，所以移動距離與移動軌跡無法知道。
4. 使用 Apple Pay 或是 Passbook。這部分的資料在之前就已經跟手機同步了，所以不用擔心如果 iPhone 不在身邊會無法結帳或是登機。

以上幾點是針對蘋果手錶內建的 App，如果是第三方開發的 App，也就是我們自己寫的 App，因為邏輯運算位於手機端，因此只要 iPhone 不在 Apple Watch 旁，這些第三方的 App 幾乎都無法使用。

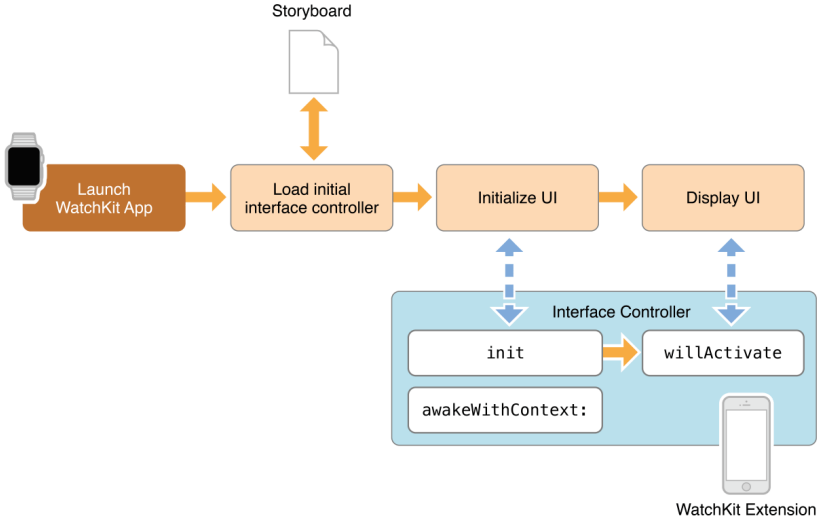
22-1 建立 WatchKit App

難易度 ★★★

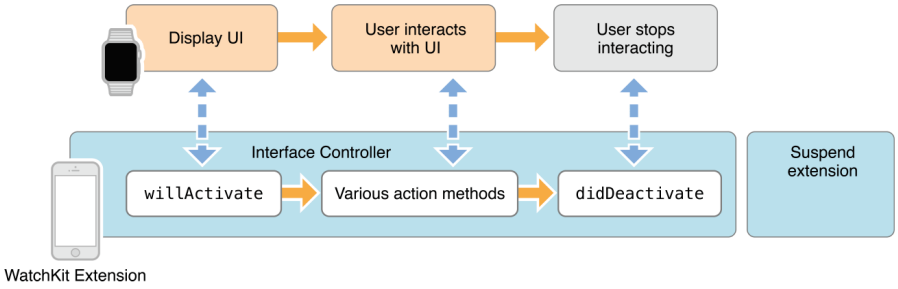
先備知識：無　Framework：無

在蘋果手錶上執行的 App 稱為 WatchKit App。WatchKit App 並不能單獨存在，它必須跟 iOS App 一起搭配使用，也就是在 Xcode 開專案時，必須先建立 iOS App 的專案後再加入 Apple Watch 的 Target，這其實也相當於 WatchKit App 是 iOS App 的 Extension。

WatchKit App 啟動後，InterfaceController 類別中 init、awakeWithContext:與 willActivate 這三個方法會被依序呼叫。其中 init 方法是類別建立時第一個被呼叫的方法，但 Xcode 並沒有幫我們產生，如果需要的話要自己實作。另外兩個方法 Xcode 已經幫我們產生好了。awakeWithContext 是在所有 UI 元件產生前會被呼叫，所以可以在這個方法中初始化或是設定 UI 元件。willActivate 則是 UI 元件都已經設定好，並且即將要顯示在螢幕上時最後會被呼叫，一般來說在這個中不該再去調整 UI 元件，它比較適合拿來播放動畫特效或是開啟另外一個行程之類的工作。



另外一個 Xcode 幫我們產生的方法為 didDeactivate。這個方法是當使用者不再操作這個畫面時會被呼叫。如果整個 App 都不再使用，例如切換到另外一個 App，除了 didDeactivate 會被呼叫外，Watch OS（Apple Watch 上運行的 OS）還會讓這個 App 進入凍結（suspend）狀態。

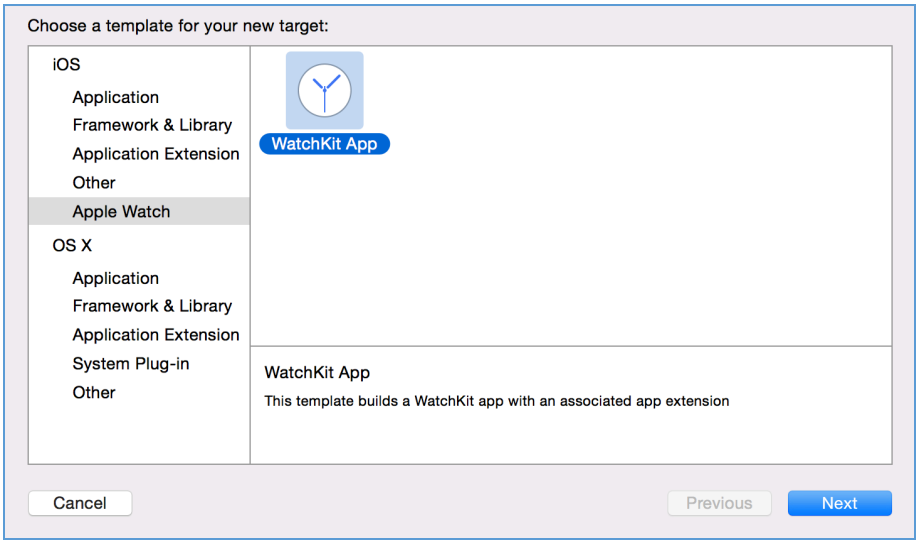


順道一提，如果要測試 willActivate 與 didDeactivate 是否正常運作，點選模擬器選單中的 lock 與 unlock 就可以測試了。

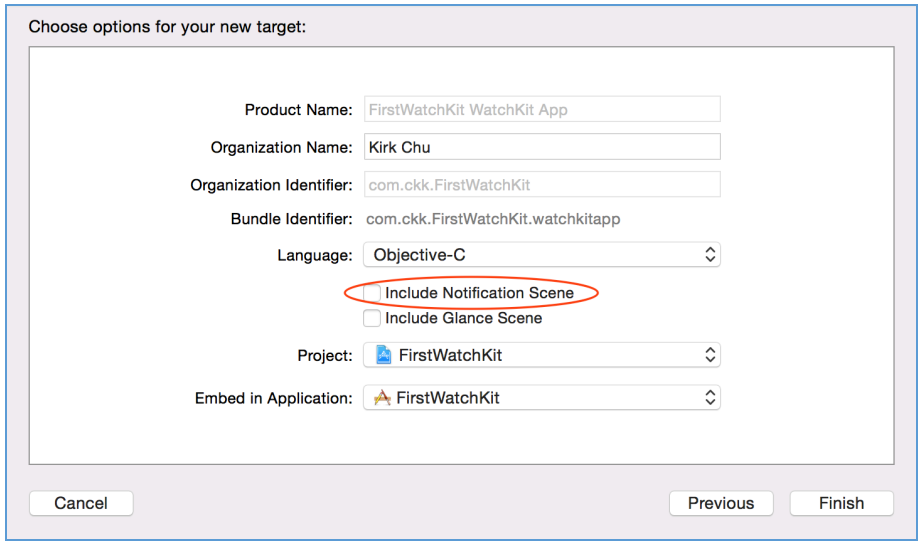
步驟與說明

1 建立 Single View Application 專案。

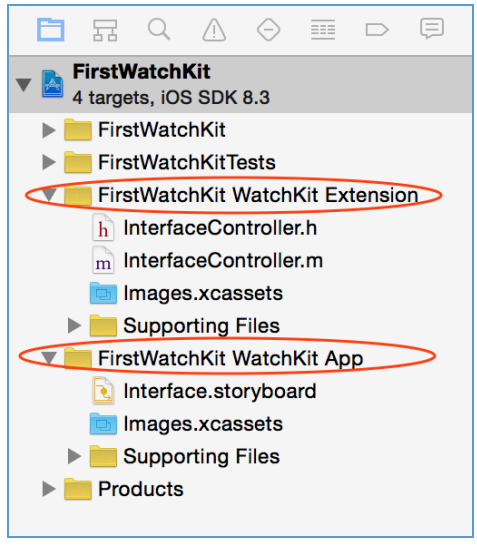
2 在 File 選單中 New 一個 Target，然後選擇 Apple Watch 的 WatchKit App。



3 按下 Next 後，如果不打算使用推播功能，建議可以先把「Include Notification Scene」勾勾拿掉，這樣 Xcode 就不會在 Storyboard 上幫我們產生跟推播通知有關的畫面，可以讓 Storyboard 單純一點。



4 完成後的專案會出現兩個黃色資料夾：一個是 WatchKit Extension；另一個是 WatchKit App。前者負責撰寫程式碼，後者負責畫面安排，所以後者包含了一個 storyboard。



5 檢視 InterfaceController.m，裡面有許多 Xcode 產生的重要方法，這些方法的功能請參考一開始的說明部分。

```
#import "InterfaceController.h"


@interface InterfaceController()

@end


@implementation InterfaceController

- (void)awakeWithContext:(id)context {
    [super awakeWithContext:context];

    // Configure interface objects here.
}

- (void)willActivate {
    // This method is called when watch view controller is about to be
visible to user
    [super willActivate];
}

- (void)didDeactivate {
    // This method is called when watch view controller is no longer visible
    [super didDeactivate];
}

@end
```

6 執行看看。

22-2 介面導覽模式

難易度 ★★★

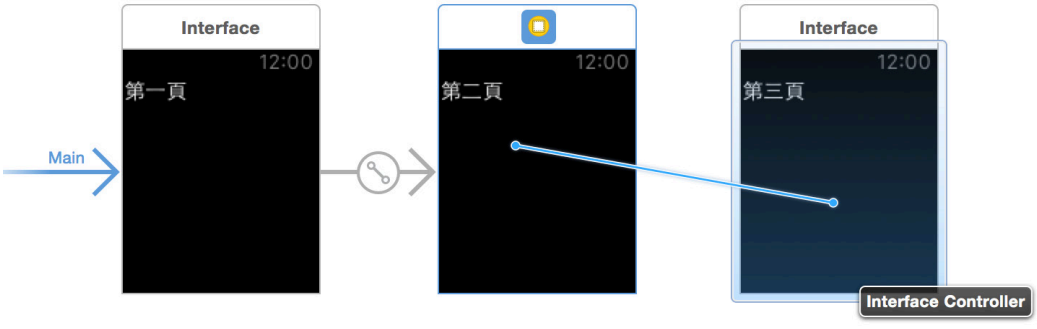
先備知識：無　Framework：無

WatchKit App 的顯示畫面如果超過一個，也就是這個 App 包含了兩個以上的 Interface Controller，那畫面的切換方式或稱之為導覽模式可分為兩種：分頁模式與階層模式。分頁模式是指所有的畫面都位於同一階層，然後用向左或向右揮擊手勢（Swipe）來切換頁面，而且在每一個頁面下方會自動產生小圓圈來告訴使用者目前在第幾頁；階層模式又再分為兩種：Push 與 Modal。Push 是讓下一頁用堆疊的方式疊到目前的畫面上，然後會在畫面最上方出現「<」符號，代表使用者可以透過它回到上一頁；而 Modal 的方式則是下一頁會取代現在的頁面，所以最上方不會出現「<」符號。分頁與階層這兩種模式不能混合使用，在同一個 App 中只能選擇一種。

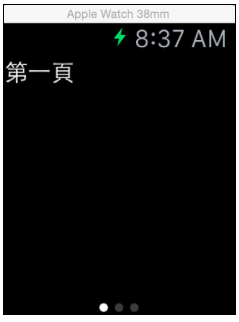
步驟與說明

1 建立 Single View Application 專案，並且加入 WatchKit App 目標。

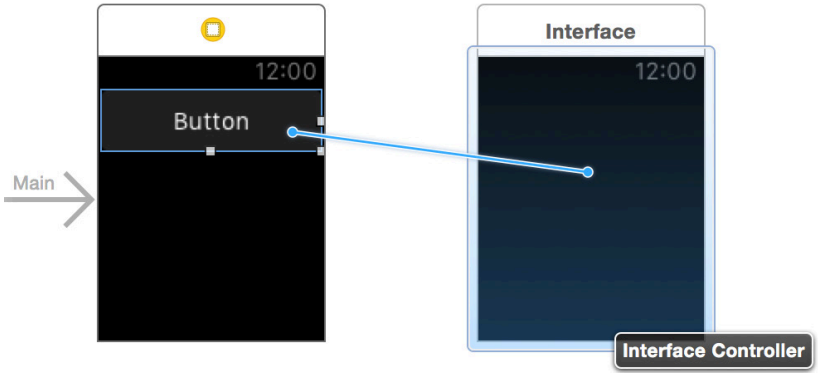
2 分頁模式：開啟 WatchKit App 的 Storyboard，然後在 Storyboard 中拖放需要的 Interface Controller 元件數量，然後從前一個 Interface Controller 上用滑鼠右鍵拉出藍線到下一個 Interface Controller 上，放開滑鼠右鍵後選擇 next page 的 Segue 型態就可以了。



3 執行看看。執行後的畫面就如下圖所示，下方的小圓圈顯示總共有三頁，目前在第一頁。



4 階層模式：設計時，通常在前一個畫面放一個 Button 元件，然後在 Button 上用滑鼠右鍵拉出藍線到下一個畫面，然後選擇 push 或 modal 型態的 Segue，這樣在按鈕按下去後就會啟動到下一個畫面的 Segue。



5 執行時，在第一個畫面按下按鈕，注意看第二個畫面最左上角有一個「<」符號，按下後會回到上一個畫面。如果 Segue 是採用 modal 方式載入下一個畫面的話，就不會出現「<」符號，因為下一個畫面已經取代了上一個畫面。