SwiftUI

Yung Chin, Yen

August 11, 2020

Outline

1 iOS app 的開發界面: UIKit v.s. SwiftUI

1.1 UIKit 與 SwiftUI 的差異性

1. 系統需求 UIKit 是從 Xcode1 就一直存在的 Framework；而 SwiftUI 則是 2019/6 WWDC 所發表的全新用來繪製 UI 的 Framework。因此，SwiftUI 必須搭配 iOS13+ 和 MacOS10.15+。[1]

2. 底層語言 UIKit 底層仍為 Objecitve-C；而 SwiftUI 則是完完全全用 Swift 打造的 Framework。

3. 語法簡潔度 SwiftUI 產生一顯示文字的元件更精簡潔了。

4. 跨平台 跨平台指的非跨 Android(但希望有那麼一天是可以支援的)。跨平台指的是使用 SwiftUI 所開發的專案，可以同時支援 macOS、watchOS、tvOS 等系統。引用一句 WWDC2019 SwiftUI 演講者所說的一句話。

   Learn once, apply everywhere.

5. Automatic Preview 這是此次 SwiftUI 最大的亮點之一，所謂 Automatic Preview，意思指的是即時預覽，即我們一邊調整程式碼的同時，也可以立即看到修改後的結果。

6. 自動支援進階功能 SwiftUI 本身即支援 Dynamic Type、Dark Mode、Localization 等等。這邊特別來講一下 UIKit 和 SwiftUI 在文字設定上有關於 Dark Mode 的差異，UIKit 若是無特別指定文字的顏色 (意即使用 Default 的選項)，在 Light Mode 字體會是白色；相對的在 Dark Mode 即會是白色，這點跟 SwiftUI 沒有特別的差異，但是 SwiftUI 除了 Default 外，還有 Secondary，如果還不喜歡的話，還有第三個選項，就是在 Assets 自行設定 Light Mode 和 Dark Mode 分別要顯示的顏色。

1.2 SwiftUI vs UIKit: Benefits and Drawbacks

1. Drawbacks of SwiftUI [2]
   - It supports only iOS 13 and Xcode 11. By switching to them, you abandon users of older versions of iOS, which is a radical move devoid of concern for the user. But since Apple annually updates its list of supported iOS versions, I think SwiftUI will be used more over the next two years as users install the latest iOS version.

[1] 簡介 SwiftUI & 用其建構一簡單的 APP

[2] SwiftUI vs UIKit: Benefits and Drawbacks

	UIKit	SwiftUI
系統需求	無限制	iOS 13+ MacOS 10.15+
底層語言	Objective-C	Swift
語法簡潔度	敗	勝
跨平台	無	有
Automatic Preview	無	有
自動支援進階功能	無	Dynamic Type Dark Mode Localization

Figure 1: UIKit 與 SwiftUI 的差異性比較圖

- It′ s still very young, so there isn′ t much data on Stack Overflow. This means that you can′ t get much help resolving complicated issues.

- It doesn′ t allow you to examine the view hierarchy in Xcode Previews.

2. Benefits of SwiftUI [2]
   - It′ s easy to learn, and the code is simple and clean.

   - It can be mixed with UIKit using UIHostingController.

   - It allows you to easily manage themes. Developers can easily add dark mode to their apps and set it as the default theme, and users can easily enable dark mode. Besides, it looks awesome.

   - SwiftUI provides mechanisms for reactive programming enthusiasts with BindableObject, ObjectBinding, and the whole Combine framework.

   - It offers Live Preview. This is a very convenient and progressive way to see the results of code execution in real time without having to build. I′ m not sure if it somehow affects the processor. So far, I′ ve noticed a delay provoked by the use of Live Preview, but I think Apple will soon make improvements.

   - SwiftUI no longer needs Interface Builder. It was replaced by Canvas, an interactive interface editor. When writing code, the visual part in Canvas is automatically generated, and when you create visual presentation elements, they automatically appear in the code.

- Your application will no longer crash if you forget to update the @IBOutlet association with the variable.

- There′ s no AutoLayout or related problems. Instead, you use things like HStack, VStack, ZStack, Groups, Lists, and more. Unlike AutoLayout, SwiftUI always produces a valid layout. There′ s no such thing as an ambiguous or unsatisfiable layout. SwiftUI replaces storyboards with code, making it easy to create a reusable view and avoid conflicts related with the simultaneous use of one project by the development team.

2 AppDelegate v.s. SceneDelegate

- SceneDelegate 為 Xcode11 所帶來的變化 (可參考官方文件 WWDC2019)，放在 SwiftUI 提似乎不太合適，但是在接下來在提到 SwiftUI App 的生命週期時會帶到，所以這邊就大概提一下。

- AppDelegate 原來的職責為負責 App 的生命週期和 UI 生命週期，在 Xcode11 後，AppDelegate 將 UI 的生命週期 (Scene Session) 交給 SceneDelegate。原 Xcode10

- 使用 Swift 為 User Interface 的專案 Launch 的生命週期為 AppDelegate ViewController，而使用 SwiftUI 為 User Interface 的專案則變成為 AppDelegate SceneDelegate ContentView，原本應該出現在 AppDelegate 的 applicationWillEnterForeground(_:) 等相關 App 到前、背景等相關的生命週期邏輯也都移至 SceneDelegate 裡了，method 名稱 application 的前綴字也都更改為 scene 了。[3]

2.1 SceneDelegate.swift

```
1 import UIKit
2 import SwiftUI
3
4 class SceneDelegate: UIResponder, UIWindowSceneDelegate {
5
6     var window: UIWindow?
7
```

[3] SwiftUI 初體驗：建構一個簡單 App　讓你了解 SwiftUI 有多強大！

```
8       func scene(_ scene: UIScene, willConnectTo session:
            UISceneSession, options connectionOptions: UIScene.
            ConnectionOptions) {
9           // Use this method to optionally configure and attach the
                UIWindow `window` to the provided UIWindowScene `scene`.
10          // If using a storyboard, the `window` property will
                automatically be initialized and attached to the scene.
11          // This delegate does not imply the connecting scene or
                session are new (see `application:
                configurationForConnectingSceneSession` instead).
12
13          // Create the SwiftUI view that provides the window contents
                .
14          let contentView = ContentView()
15
16          // Use a UIHostingController as window root view controller.
17          if let windowScene = scene as? UIWindowScene {
18              let window = UIWindow(windowScene: windowScene)
19              window.rootViewController = UIHostingController(rootView
                    : contentView)
20              self.window = window
21              window.makeKeyAndVisible()
22          }
23      }
24
25      func sceneDidDisconnect(_ scene: UIScene) {
26          // Called as the scene is being released by the system.
27          // This occurs shortly after the scene enters the background
                , or when its session is discarded.
28          // Release any resources associated with this scene that can
                 be re-created the next time the scene connects.
29          // The scene may re-connect later, as its session was not
                neccessarily discarded (see `application:
                didDiscardSceneSessions` instead).
30      }
31
32      func sceneDidBecomeActive(_ scene: UIScene) {
33          // Called when the scene has moved from an inactive state to
                 an active state.
```

```
34          // Use this method to restart any tasks that were paused (or
                not yet started) when the scene was inactive.
35      }
36
37      func sceneWillResignActive(_ scene: UIScene) {
38          // Called when the scene will move from an active state to
                an inactive state.
39          // This may occur due to temporary interruptions (ex. an
                incoming phone call).
40      }
41
42      func sceneWillEnterForeground(_ scene: UIScene) {
43          // Called as the scene transitions from the background to
                the foreground.
44          // Use this method to undo the changes made on entering the
                background.
45      }
46
47      func sceneDidEnterBackground(_ scene: UIScene) {
48          // Called as the scene transitions from the foreground to
                the background.
49          // Use this method to save data, release shared resources,
                and store enough scene-specific state information
50          // to restore the scene back to its current state.
51      }
52  }
53
54  struct SceneDelegate_Previews: PreviewProvider {
55      static var previews: some View {
56          /*@START_MENU_TOKEN@*/Text("Hello, World!")/*
                @END_MENU_TOKEN@*/
57      }
58  }
```

```
<stdin>:1:8: error: no such module 'UIKit'
import UIKit
       ^
```

3 UIKit

4 SwiftUI

4.1 學習資源

- SwiftUI Tutorials from Apple

4.2 使用 SwiftUI 開啟新專案 [3]

1. 首先，打開 Xcode，並點擊 Create new Xcode project。在 iOS 之下選擇 Single View App，並為專案命名。

2. 然後在下方勾選 Use SwiftUI 的選項，如果沒有勾選該選項的話，Xcode 會自動產生 storyboard 檔案 (UIKit)。

3. Xcode 會自動幫你創建一個名為 ContentView.swif 的檔案，Xcode 會在程式碼的右邊呈現一個即時的預覽視窗 (preview)，點選 resume 鈕生成預覽畫面 (會花一點時間)。

1. ContentView.swift

```
1  import SwiftUI
2
3  struct ContentView: View {
4      var body: some View {
5          Text("文字")
6      }
7  }
8
9  struct ContentView_Previews: PreviewProvider {
10     static var previews: some View {
11         ContentView()
12     }
13 }
```

4.3 Text

1. 改變 Text 的屬性
   - 改變 component 有兩種方式：工具列、code

- Attributes
  - frame
  - foregroundColor
  - background
  - font

(a) SwiftUI Inspector:
   i. on Text object (in preview screen): CMD + click
   ii. select Show SwiftUI Inspector
   iii. change Text, Font, Color
   iv. Monitor the corresponding code changes in code window

`images/inspector-1.gif`

(b) Inspector frame `images/inspector-2.gif`

(c) code **於** Text("...") **後加上屬性** function **或修改其他屬性**
`images/inspector-3.gif`

4.4 VStack

一個以上的物件都要放在 Stack **中，** Stack **與** Stack **可相互包含，加入方式有二:**

1. **由工具列** drag: Xcode **會自動加入相對的** code `images/vstack.gif`

2. coding

```
1 import SwiftUI
2
3 struct ContentView: View {
4     var body: some View {
5         VStack {
```

```
6                 Text("第一行文字")
7                 Text("第二行文字")
8             }
9         }
10    }
11
12    struct ContentView_Previews: PreviewProvider {
13        static var previews: some View {
14            ContentView()
15        }
16    }
```

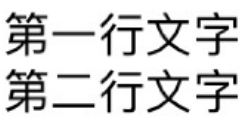

Figure 2: VStack

3. SwiftUI 撰寫原則
   - body 恆為只能 return 一物件。
   - 若有多個物件時，一定得放在 Stack 裡。

4.5 HStack

```
1  import SwiftUI
2
3  struct ContentView: View {
4      var body: some View {
5          HStack {
6              VStack {
7                  Button("請按我") {
8                      print("TEST")
9                  }
10                 .frame(width: 60, height: 30, alignment: .center)
11                 .foregroundColor(.white)
```

```
12                  .background(Color.green)
13                  Button("別亂按") {
14                      print("QQ")
15                  }
16              }
17              VStack {
18                  Text("第一行文字")
19                      .frame(width: 100, height: 30, alignment: .
                            center    )
20                      .foregroundColor(.white)
21                      .background(Color.orange)
22                  Text("第二行文字")
23                      .frame(width: 100, height: 30, alignment: .
                            center)
24                      .foregroundColor(.white)
25                      .background(Color.red)
26              }
27          }
28      }
29  }
30
31  struct ContentView_Previews: PreviewProvider {
32      static var previews: some View {
33          ContentView()
34      }
35  }
```

Figure 3: HStack

4.6 Image

影像來源可以是 System Image **或自行下載/編修的影像** (Customized Image)

1. System Image
   (a) SF Symbols [4]
   (b) **從** iOS 13 **開始，**Apple **介紹了一個名為** SFSymbols **的新功能。**SF Symbols **這功能由** Apple **所設計，當中集合了** 1500 **多個可以在** App **之中使用的符號。**[3]
   (c) Download SF Symbols app
   (d) code

```
1  import SwiftUI
2
3  struct ContentView: View {
4      var body: some View {
5          VStack {
6              Text("System Image")
7                  .font(.headline)
8                  .foregroundColor(.orange)
9              Image(systemName: "icloud")
10             .resizable()
11             .scaledToFit()
12             .frame(width: 100, height: 80, alignment: .
                   center)
13         }
14     }
15 }
16
17 struct ContentView_Previews: PreviewProvider {
18     static var previews: some View {
19         ContentView()
20     }
21 }
```

```
<stdin>:9:13: error: 'init(systemName:)' is unavailable in macOS
            Image(systemName: "icloud")
            ^~~~~
SwiftUI.Image:7:12: note: 'init(systemName:)' has been explicitly ma
```

[4] Find all available images for Image(systemName:) in SwiftUI

```
public init(systemName: String)
       ^
```

(e) Demo

Figure 4: Images-1

2. Customized Image **語法**

(a) Drag image into Project folder Assets.xcassets

(b) Add following code

```
1 Image("ImageName") //file name in Assets.xcassets
2   .resizable()
3   .scaledToFit()
4   .frame(width: 200, height: 160, alignment: .center)
```

```
<stdin>:1:1: error: use of unresolved identifier 'Image'
Image("ImageName") //file name in Assets.xcassets
^~~~~
```

3. Image Attributes

```
1  import SwiftUI
2
3  struct ContentView: View {
4      var body: some View {
5          VStack {
6              Text("Albert␣Camus")
7                  .font(.title)
8                  .foregroundColor(.white)
9                  .background(Color.orange)
10             Image("Albert-Camus")
```

```
11                      .resizable()
12                      .scaledToFill()
13                      .frame(width: 200, height: 200, alignment: .
                            center)
14                      .clipShape(Circle())
15
16              }
17          }
18  }
19
20  struct ContentView_Previews: PreviewProvider {
21          static var previews: some View {
22              ContentView()
23          }
24  }
```

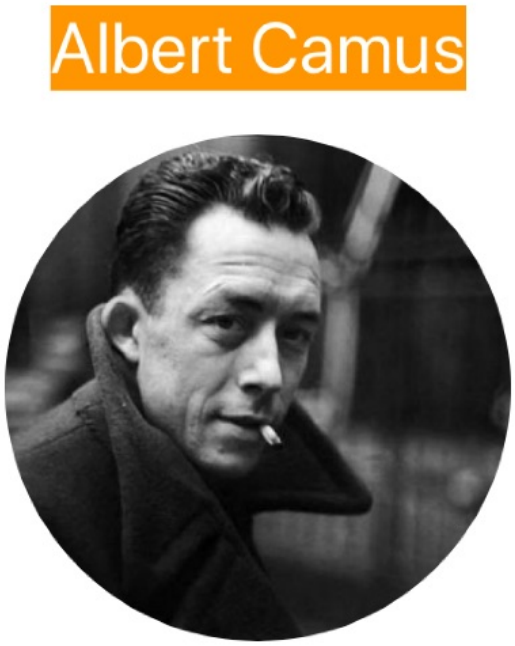

Figure 5: Images-2

4.7 Button

1. **語法**

```
1  //...
2  Button("Title") {
```

```
    //action
}

Button(action: <#T##() -> Void#>, label: <#T##() -> _#>)
//...
```

```
<stdin>:6:16: error: editor placeholder in source file
Button(action: <#T##() -> Void#>, label: <#T##() -> _#>)
               ^
<stdin>:6:42: error: editor placeholder in source file
Button(action: <#T##() -> Void#>, label: <#T##() -> _#>)
                                         ^
<stdin>:6:53: error: expected type for function result
Button(action: <#T##() -> Void#>, label: <#T##() -> _#>)
                                                    ^
<stdin>:2:1: error: use of unresolved identifier 'Button'
Button("Title") {
^~~~~~
<stdin>:6:1: error: use of unresolved identifier 'Button'
Button(action: <#T##() -> Void#>, label: <#T##() -> _#>)
^~~~~~
```

2. **範例: 按下** Button**, 改變** Text title

```
struct ContentView: View {
    @State private var title = "Hello SWiftUI"

    var body: some View {
        VStack {
            Text(verbatim: title)
                .padding(4)
                .foregroundColor(.white)
                .background(Color.gray)
            Button(action: {
                self.title = "Good Day"
            }) {
                Text("請按我1")
                    .foregroundColor(.white)
                    .padding(4)
                    .background(Color.blue)
```

```
17                }
18                Button("請按我2") {
19                    self.title = "Good_night..."
20                }
21            }
22        }
23    }
```

```
<stdin>:2:5: error: unknown attribute 'State'
    @State private var title = "Hello SWiftUI"
    ^
<stdin>:4:20: error: use of undeclared type 'View'
    var body: some View {
                    ^~~~
<stdin>:1:21: error: use of undeclared type 'View'
struct ContentView: View {
                    ^~~~
```

3. Demo

Figure 6: Button

4.8 TextField

1. 語法

```
1 @State private var 變數="值"
2 TextField("提示文字", text: $變數)
```

```
<stdin>:2:33: error: '$' is not an identifier; use backticks to escape
TextField("提示文字", text: $變數)
<stdin>:2:34: error: expected ',' separator
```

```
TextField("提示文字", text: $變數)
<stdin>:1:1: error: unknown attribute 'State'
@State private var 變數="值"
<stdin>:2:1: error: use of unresolved identifier 'TextField'
TextField("提示文字", text: $變數)
<stdin>:2:33: error: use of unresolved identifier '$'
TextField("提示文字", text: $變數)
```

2. **範例: 於 TextField 輸入資料, 顯示於 Text 中**

```
1  import SwiftUI
2
3  struct ContentView: View {
4      @State private var title = ""
5
6      var body: some View {
7          VStack {
8              Text(verbatim: "Hello␣"+title)
9              HStack {
10                 Text("Your␣Name:␣")
11                 TextField("請輸入姓名:", text: $title)
12             }
13         }
14     }
15 }
16
17 struct ContentView_Previews: PreviewProvider {
18     static var previews: some View {
19         ContentView()
20     }
21 }
```

3. Demo

Figure 7: Button

5 Customize UI Components

SwiftUI 提供豐富的 modifier 幫助我們設計客製 UI 元件的樣式，諸如陰影，旋轉等效果皆可透過 modifier 實現，還可以搭配方便的拖曳加入相關程式碼。[5]

5.1 Text

1. Advanced Attributes [6]

```
1  struct ContentView: View {
2      var body: some View {
3          Text("Example")
4              .font(.title)
5              .fontWeight(.bold)
6              .foregroundColor(Color.white)
7              .padding(4)
8              .background(Color.gray)
9              .cornerRadius(14.0)
10             .rotationEffect(Angle(degrees: 15))
11             .rotation3DEffect(Angle(degrees: 30), axis: (x: 10,
                   y: 30, z: 30))
12             .shadow(radius: 20)
13     }
14 }
```

```
<stdin>:2:20: error: use of undeclared type 'View'
    var body: some View {
                   ^~~~
<stdin>:1:21: error: use of undeclared type 'View'
struct ContentView: View {
                    ^~~~
```

2. Demo

[5] 客製 UI 元件樣式的 SwiftUI modifier

[6] 客製 UI 元件樣式的 SwiftUI modifier

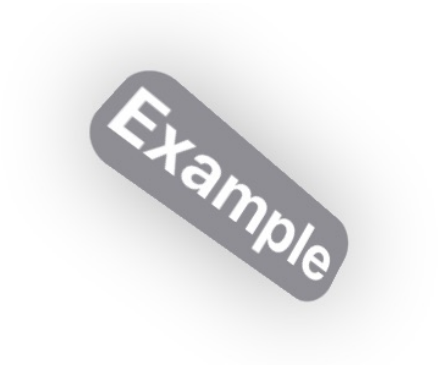

Figure 8: Text Attributes

5.2 Image

1. Advanced Attributes [7]

```
1  import SwiftUI
2
3  struct ContentView: View {
4      var body: some View {
5          VStack {
6              Text("Albert Camus")
7                  .font(.body)
8                  .foregroundColor(.white)
9                  .background(Color.orange)
10             Image("Albert-Camus")
11                 .resizable()
12                 .scaledToFill()
13                 .frame(width: 100, height: 100, alignment: .
                       center)
14                 .clipShape(Circle())
15             Image(systemName: "alarm.fill")
16                 .resizable()
17                 .scaledToFill()
18                 .frame(width: 100, height: 100, alignment: .
                       center)
19             Image("Albert-Camus")
20                 .frame(width: 100, height: 100, alignment: .
                       center)
21                 .mask(Image(systemName: "alarm.fill")
```

[7] SwiftUI 裁切形狀的 clipShape & mask

```
22                      .resizable()
23                      .scaledToFit())
24                  .shadow(radius: 20)
25          }
26      }
27  }
28
29  struct ContentView_Previews: PreviewProvider {
30      static var previews: some View {
31          ContentView()
32      }
33  }
```

```
<stdin>:15:13: error: 'init(systemName:)' is unavailable in macOS
            Image(systemName: "alarm.fill")
            ^~~~~
SwiftUI.Image:7:12: note: 'init(systemName:)' has been explicitly marke
    public init(systemName: String)
           ^
<stdin>:21:23: error: 'init(systemName:)' is unavailable in macOS
                .mask(Image(systemName: "alarm.fill")
                      ^~~~~
SwiftUI.Image:7:12: note: 'init(systemName:)' has been explicitly marke
    public init(systemName: String)
           ^
```

2. Demo

5.3 Button

1. Advanced Attributes [8]

```
1  import SwiftUI
2
3  struct ContentView: View {
4      var body: some View {
5          VStack(spacing: 5.0) {
6              Text("Customized␣Button")
7                  .font(.body)
```

[8] SwiftUI 小技巧：利用 border 修飾符　輕鬆為按鈕或文本繪製邊框

Figure 9: Image Attributes

```
8                       .foregroundColor(.white)
9                       .background(Color.orange)
10                  Button(action: {
11                      print("Hello␣button␣tapped!")
12                  }) {
13                      Text("HI␣HI")
14                          .fontWeight(.bold)
15                          .font(.title)
16                          .foregroundColor(.purple)
17                          .padding()
18                          .border(Color.purple, width: 5)
19                  }
20                  Button(action: {
21                      print("Hello␣button␣tapped!")
22                  }) {
23                      Text("Press␣me")
24                          .fontWeight(.light)
25                          .font(.title)
26                          .foregroundColor(.green)
27                          .padding(5)
28                          .overlay(
29                              Capsule(style: .continuous)
30                                  .stroke(Color.green, style:
                                        StrokeStyle(lineWidth: 3, dash:
                                        [10]))
31                      )
32                  }
33              }
34          }
35  }
```

2. Demo

6 Advanced Commponents

6.1 List

1. **準備單一** cell **格式**

```
1  import SwiftUI
```

Figure 10: Button Attributes

```
2
3  struct ContentView: View {
4      var body: some View {
5          HStack {
6              Image(systemName: "book")
7                  .resizable()
8                  .frame(width: 30, height: 30, alignment: .center
                       )
9              VStack(alignment: .leading) {
10                 Text("Artificial␣Intelligence:␣A␣Modern␣Approach
                       ")
11                     .multilineTextAlignment(.leading)
12                     .foregroundColor(Color.green)
13                 Text("Stuart␣Russell␣and␣Peter␣Norvig")
14                     .multilineTextAlignment(.leading)
15                     .foregroundColor(Color.orange)
16             }
17         }
18     }
19 }
```

```
<stdin>:6:13: error: 'init(systemName:)' is unavailable in macOS
            Image(systemName: "book")
            ^~~~~
SwiftUI.Image:7:12: note: 'init(systemName:)' has been explicitly marke
    public init(systemName: String)
           ^
```

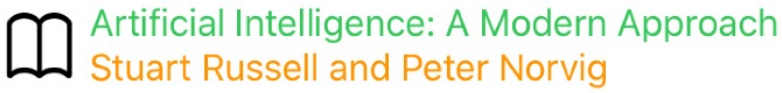

Figure 11: Single cell

2. **轉入 List 格式**

   (a) **將最外層的 VStack 加入 List 中**

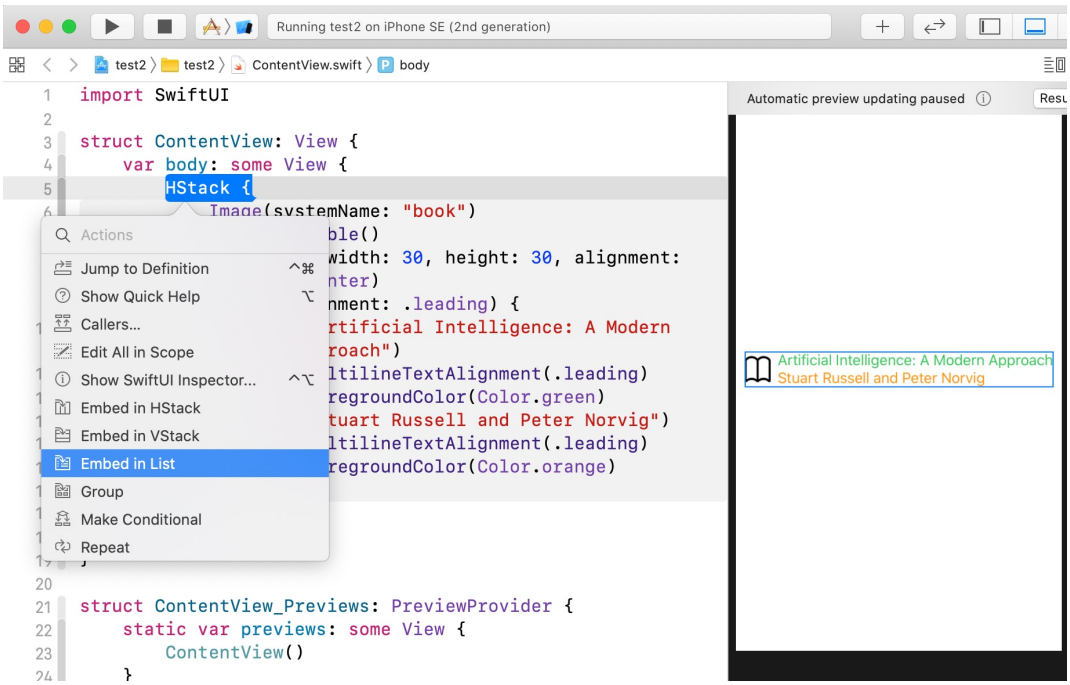

Figure 12: List-1

   (b) list **語法**

```
1  import SwiftUI
2
3  struct ContentView: View {
4      var body: some View {
5          List(0 ..< 5) { item in
6              Image(systemName: "book")
7                  .resizable()
8                  .frame(width: 30, height: 30, alignment: .
                       center)
9              VStack(alignment: .leading) {
10                 Text("Artificial␣Intelligence:␣A␣Modern␣
                       Approach")
11                     .multilineTextAlignment(.leading)
```

```
12                         .foregroundColor(Color.green)
13                     Text("Stuart Russell and Peter Norvig")
14                         .multilineTextAlignment(.leading)
15                         .foregroundColor(Color.orange)
16                 }
17             }
18         }
19 }
```

```
<stdin>:6:13: error: 'init(systemName:)' is unavailable in macOS
            Image(systemName: "book")
            ^~~~~
SwiftUI.Image:7:12: note: 'init(systemName:)' has been explicitly ma
    public init(systemName: String)
           ^
```

(c) 結果

```
1  import SwiftUI
2
3  struct ContentView: View {
4      var body: some View {
5          List(0 ..< 5) { item in
6              Image(systemName: "book")
7                  .resizable()
8                  .frame(width: 30, height: 30, alignment:
                       .center)
9              VStack(alignment: .leading) {
10                 Text("Artificial Intelligence: A Modern
                       Approach")
11                     .multilineTextAlignment(.leading)
12                     .foregroundColor(Color.green)
13                 Text("Stuart Russell and Peter Norvig")
14                     .multilineTextAlignment(.leading)
15                     .foregroundColor(Color.orange)
16             }
17         }
18     }
19 }
```

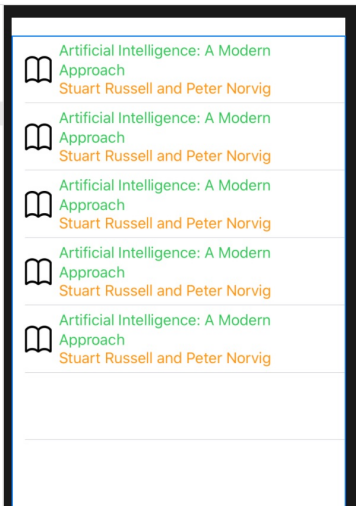

Figure 13: List-2

3. 建立 list 來源數據[3]

In order to handle dynamic items, you must first tell SwiftUI how it can identify which item is which. This is done using the Identifiable protocol, which has only one requirement: some sort of id value that SwiftUI can use to see which item is which. [9]

```
1 import SwiftUI
2
```

[9]How to create a list of dynamic items

```
3  //建立book struct
4  struct Book: Identifiable {
5      var id = UUID()
6      var title: String
7      var author: String
8      var image: String
9  }
10
11 struct ContentView: View {
12     var books = [
13       Book(id: UUID(), title: "地獄藍調", author: "李查德",
             image: "b1"),
14       Book(id: UUID(), title: "至死方休", author: "李查德",
             image: "b2"),
15       Book(id: UUID(), title: "一觸即發", author: "李查德",
             image: "b3"),
16       Book(id: UUID(), title: "索命訪客", author: "李查德",
             image: "b4"),
17       Book(id: UUID(), title: "闇夜回聲␣", author: "李查德",
             image: "b5")]
18
19     //.....
20 }
```

```
<stdin>:11:8: error: type 'ContentView' does not conform to protocol 'V
struct ContentView: View {
       ^
SwiftUI.View:5:20: note: protocol requires nested type 'Body'; do you w
    associatedtype Body : View
                   ^
```

4. 將數據連結到列表中[3]

```
1  import SwiftUI
2
3      //....
4      var body: some View {
5          List(books) { book in
6              Image(book.image)
7                  .resizable()
```

```
8                   .frame(width: 40, height: 40, alignment: .center
                        )
9               VStack(alignment: .leading) {
10                  Text(book.title)
11                      .multilineTextAlignment(.leading)
12                      .foregroundColor(Color.green)
13                  Text(book.author)
14                      .multilineTextAlignment(.leading)
15                      .foregroundColor(Color.orange)
16              }
17          }
18      }
19  }
```

```
<stdin>:19:1: error: extraneous '}' at top level
}
^
<stdin>:5:14: error: use of unresolved identifier 'books'
        List(books) { book in
             ^~~~~
```

5. 結果

Figure 14: List-3

6. 為什麼要加入 id 與 Identifiable
    - Identifiable: 允許 Array 中有重複值
    - id: 明確區分重複值
    - UUID: 自動生成 unique 值
    - 詳細說明如SwiftUI - Dynamic List & Identifiable

6.2 Navigation bar

- 於 body 中最外層的 component 之外加入 NavigationView
- Title: navigationBarTitle()

```
1  import SwiftUI
2    ...
3
4  struct ContentView: View {
5      ....
6      var body: some View {
7          NavigationView {
8              List(books) { book in
9                  ...
10                 }
11             }.navigationBarTitle(Text("書單"))
12         }
13     }
14 }
15 ...
```

```
<stdin>:2:3: error: unary operator cannot be separated from its operand
  ...
  ^

<stdin>:4:1: error: expected expression
struct ContentView: View {
^
```

```
<stdin>:5:5: error: expected 'func' keyword in operator function declaratio
    ....
    ^
    func
<stdin>:5:9: error: expected '(' in argument list of function declaration
    ....
        ^
<stdin>:9:17: error: unary operator cannot be separated from its operand
                ...
                ^


<stdin>:10:17: error: expected expression
                }
                ^
<stdin>:14:1: error: extraneous '}' at top level
}
^
<stdin>:15:1: error: expected expression after unary operator
...
^
<stdin>:15:4: error: expected expression
...
   ^
<stdin>:5:5: error: operator '....' declared in type 'ContentView' must be
    ....
    ^
    static
<stdin>:5:5: error: operators must have one or two arguments
    ....
    ^
<stdin>:5:5: error: expected '{' in body of function declaration
    ....
    ^
<stdin>:5:5: error: member operator '....()' must have at least one argumen
    ....
    ^
<stdin>:8:18: error: use of unresolved identifier 'books'
            List(books) { book in
                 ^~~~~
```

Figure 15: Navigation bar

6.3 Button, Divider

```
1  //
2  //  ContentView.swift
3  //  uitest
4  //
5  //  Created by yen yung chin on 2020/7/29.
6  //  Copyright ℗ 2020 Letranger.tw. All rights reserved.
7  //
8
9  import SwiftUI
10
11 struct ContentView: View {
12     @State private var a = ""
13     @State private var b = ""
14     @State private var c = "Ans:"
15
16     var body: some View {
17         VStack {
```

```
18              VStack {
19                  Divider()
20                  TextField("Number 1: ", text: $b)
21                  Divider()
22                  TextField("Number 2:", text: $a)
23                  Divider()
24                  Button(" ") {
25                      let one = Int(self.a) ?? 0
26                      let two = Int(self.b) ?? 0
27                      self.c = "Ans: " + String(one + two)
28                  }
29                  .frame(width: 40, height: 30, alignment: .center)
30                  .foregroundColor(.white)
31                  .background(Color.green)
32                  .font(.largeTitle)
33                  Divider()
34                  Text(verbatim: c)
35                      .foregroundColor(.gray)
36              }
37              .frame(width: 200, height: 160, alignment: .center)
38
39          }
40      }
41  }
```

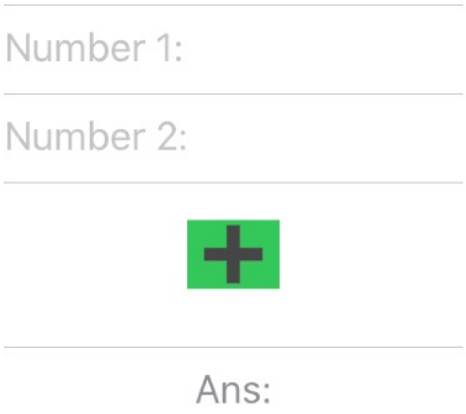

Figure 16: Button

6.4 background, opacity

```
1  //
2  //  ContentView.swift
3  //  uitest
4  //
5  //  Created by yen yung chin on 2020/7/29.
6  //  Copyright ℗ 2020 Letranger.tw. All rights reserved.
7  //
8
9  import SwiftUI
10
11 struct ContentView: View {
12     @State private var a = ""
13     @State private var b = ""
14     @State private var c = "Ans:"
15
16     var body: some View {
17         VStack(alignment: .center) {
18             Text("計算機")
19             Divider()
20             TextField("Number 1: ", text: $b)
21             Divider()
22             TextField("Number 2:", text: $a)
23             Divider()
24             Button(" ") {
25                 let one = Int(self.a) ?? 0
26                 let two = Int(self.b) ?? 0
27                 self.c = "Ans: " + String(one + two)
28             }
29             .frame(width: 40, height: 30, alignment: .center)
30             .foregroundColor(.white)
31             .background(Color.white)
32             .font(.largeTitle)
33             Divider()
34             Text(verbatim: c)
35                 .foregroundColor(.black)
36
37
38         }
39         .padding(60)
```

```
40            .background(Image("background").resizable().scaledToFill())
41            .opacity(0.9)
42        }
43    }
44    struct ContentView_Previews: PreviewProvider {
45        static var previews: some View {
46            ContentView()
47        }
48    }
```

7 Struct, Class, and Enum

struct, class, and enum are fundamentally important concepts for every iOS developer. It′ s difficult to imagine common iOS apps without them. [10]

7.1 Structures

```
1  struct Birthday {
2    var day: Int = 12
3    var month: Int = 12
4    var year: Double = 1999
5
6    func myBirthday() {
7    print("I've born in \(year).\(month).\(day)")
8    }
9  }
10
11 struct Person {
12   var firstName: String = "Abboskhon"
13   var lastName: String = "Shukurullaev"
14   var phoneNumber: String = 123456
15   var emailAddress: String = "abbsh24@gmail.com"
16
17   func myInfo() {
18   print("My name is \(firstName) \(secondName). My phone number is
       \(phoneNumber) and email address is \(emailAddress)")
```

[10] Demystifying Struct, Class, and Enum in Swift 5

Figure 17: Background

```
19    }
20 }
```

```
<stdin>:14:29: error: cannot convert value of type 'Int' to specified type
  var phoneNumber: String = 123456
                            ^~~~~~
<stdin>:18:36: error: use of unresolved identifier 'secondName'
  print("My name is \(firstName) \(secondName). My phone number is \(phoneN
                                   ^~~~~~~~~~
```

1. Mutating methods

   Structs have also got so-called mutating methods that play a role in updating the property values of a structure within an instance method. [10]

```
 1 struct Counter {
 2   var count: Int = 0
 3
 4   mutating func increment() {
 5     count += 1
 6   }
 7   mutating func increment(by amount: Int) {
 8     count += amount
 9   }
10   mutating func reset() {
11     count = 0
12   }
13 }
14 var counter = Counter()    //default is 0
15 counter.increment()        //becomes 1
16 counter.increment(by: 9)   //becomes 10
17 counter.reset()            //reset to 0
```

7.2 Classes

Classes and structures are very similar, and both can be used to define properties and methods. [10]

1. Inheritance & Override methods and properties

   The biggest difference that structs do not have is hierarchical relations. Classes can have parent classes, that are called superclass, and child classes, that are called subclasses. [10]

```
class Animals {
    func animals() {
        print("Animals are mainly of 3 types: Land Animals, Sea
            animals, Air Animals")
    }
}

class LandAnimals: Animals {
    override func animals() {
        print("Land animals are cats, sheeps, horses.")
    }
}

class Cat: LandAnimals {
    override func animals() {
        print("I am a cat and I am a land animal.")
    }
}
```

7.3 結構跟類的比較 (Comparing Structures and Classes) [11]

1. 在 Swift 中的結構與類有許多相同之處，兩者皆能：
   - 定義屬性來儲存值
   - 定義方法來提供功能
   - 定義下標來提供訪問他們用下標語法的值
   - 定義初始化器來設定他們的初始狀態
   - 可被擴展以擴展其功能，超越預設的實現
   - 符合協議以提供某種標準功能
2. 類具有的附加功能，但結構沒有：
   - 繼承使一個類能夠繼承另一個的特性

[11] Day 10: [Swift] 結構和類 (Struct and Class)

- 類型轉換使我們可以在運行時檢查和解釋類實例的類型
- 反初始化器允許類的實例釋放它已分配的任何資源
- 引用計數允許對類實例的多個引用

3. 結構和列舉是值型別 (Structures and Enumerations Are Value Types) 值型別是一種其值在被賦值給變數或常數時被複製，或者在傳遞給函數時被複製。實際上，Swift 的整數、浮點數、布林值、字符串、數組和字典中的所有基本型別都是值型別，並且在幕後實現為結構。所有結構和列舉都是 Swift 中的值型別。這代表著我們創建的任何結構和列舉實例以及它們作為屬性的任何值類型在代碼中傳遞時始終會被複製。[11]

   (a) 範例

```
struct Resolution {
    var width = 0
    var height = 0
}

let hd = Resolution(width: 1920, height: 1080)
var cinema = hd

cinema.width = 2048

print("cinema is now \(cinema.width) pixels wide")
// Prints "cinema is now 2048 pixels wide"

print("hd is still \(hd.width) pixels wide")
// Prints "hd is still 1920 pixels wide"
```

```
cinema is now 2048 pixels wide
hd is still 1920 pixels wide
```

   由上述例子可知，當 cinema 被賦予 hd 當前的值，儲存在 hd 的值被複製到新 cinema 的實例。最後結果兩個擁有相同值但完全不同的實例，所以當修改 cinema.width = 2048 的時候，並不會影響儲存在 hd 中的 width。[11]

4. 類是參考型別 (Classes Are Reference Types) 不同於值型別，當參考型別被指定給一個變數或常數，或是傳遞進一個函數的時候，並不會被複製。參考型別非副本，是使用相同存在的實例。[11]

   (a) 範例

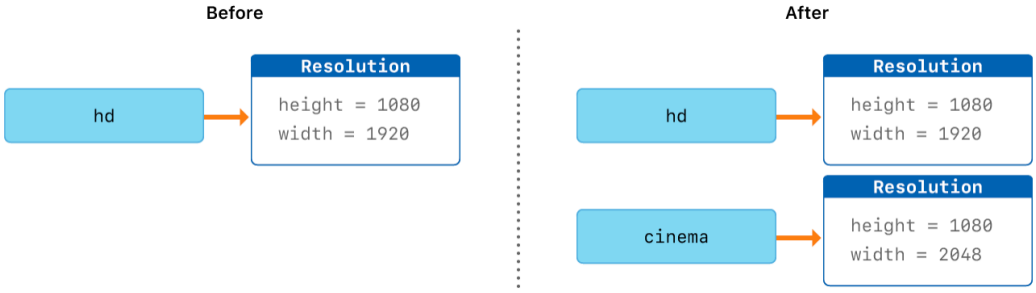

Figure 18: Struct value type

```
1  class VideoMode {
2      var interlaced = false
3      var frameRate = 0.0
4      var name: String?
5  }
6
7  let tenEighty = VideoMode()
8
9  tenEighty.interlaced = true
10 tenEighty.name = "1080i"
11 tenEighty.frameRate = 25.0
12
13 let alsoTenEighty = tenEighty
14 alsoTenEighty.frameRate = 30.0
15
16 print("The frameRate property of tenEighty is now \(
       tenEighty.frameRate)")
17 // Prints "The frameRate property of tenEighty is now 30.0"
```

```
The frameRate property of tenEighty is now 30.0
```

由上述例子可知,當 alsoTenEighty 被指定為 tenEighty,並且修改 alsoTenEighty.frameRate = 30.0 時，也會更動到 tenEighty.frameRate 的值。[11]

7.4 Enumerations

Enumerations are different from structs and classes. Enum is a special Swift type that defines a common type for a group of related values. Then, enums can be interacted with other data types such as a switch, if statements, and others. [10]

從 Swift 3 開始，我們的 enum 內設定的情境字串要以小寫字母開頭。

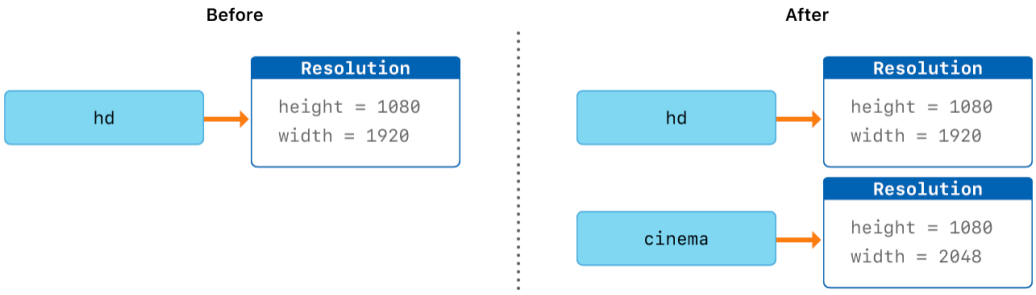

Figure 19: Class Reference type

```swift
//1 representation of enum
enum DownloadStatus {
    case downloading
    case finished
    case failed
    case cancelled
}

var currentStatus = DownloadStatus.downloading

switch currentStatus {
case .downloading:
    print("Downloading...")

case .finished:
    print("Just finished the download...")

case .failed:
    print("Failed to download the file...")

case .cancelled:
    print("The download is cancelled...")
}
```

```
Downloading...
```

此時，你可能會想，為什麼需要使用 enum 來定義多個情況，而不選擇宣告一個包含四個情境項目的 array，如下圖所示:

let downloadStatus = [“downloading” , “finished” , “failed” , “cancelled”]

let currentStatus = downloadStatus[0]

你可以這樣做沒錯,但是如此一來會有兩個缺點,首先,你可能會不知道 downloadStatus[0] 代表什麼，除非你引用 downloadStatus array，若是比較 downloadStatus[0] 與 DownloadStatus.downloading 這兩種表達方式，很明顯的是後者的可讀性比較高。

其次，因為 currentStatus 是 String 類型，變量可以被賦予任何字符串值，無法將它限制為“downloading”, “finished”, “failed”以及“cancelled”，除非執行一些額外的驗證。反之，如果使用 enum，我們可以將 myDirection 限制在.downloading、.finished、.failed 或.cancelled 等四種情境之一，不會出現其他意料之外的情況。[12]

7.5 結構和類之間的選擇 (Choosing Between Structures and Classes)

總的來說，用 class 來定義資料物件的話，就好像是在用雲端共享文件一樣：每個人的螢幕上都會有一份文件可以編輯，但這個文件並沒有存在電腦裡，而是跟雲端的版本連線，所以所有的變動都是直接在雲端版本上更新的。好處是方便，壞處是誰修改了甚麼東西經理不會知道（class 本身沒有帳號功能!)。

用 struct 的話，則是像傳統的離線文件檔案一樣。一開始文件只有經理有，而如果他想要讓手下小美去修改文件的話，他就需要拷貝一份檔案給小美。小美修改完檔案後，必須把它交還給經理，然後經理再決定要不要用修改過的檔案取代原本的文件。[13]

- 在 MVC 架構中，Model 包含了 Data Objects 與 Document，其中 Documents 包含所有的資料管理者元件 (如 Core Data 的 NSManagedObjectContext，或是任何負責下載、上傳資料物件的網路層元件，如自訂的 NetworkManager)。

- Document 最好是用 class 來定義，因為它肩負了許多溝通的工作。Data Objects 則是資料的代表，要用 class 或 struct 來定義都可以。

- 用 class 定義 Data Objects 的話，任何變動只要執行一次就可以了，因為它的實體只會有一個。然而，用 struct 定義的話，則需要將變動手動套用到文件所管理的那份實體，好讓整個 app 都能使用最新的資料。這雖然寫起來較為囉唆，卻讓閱讀與維護更為簡單 (可以確定是在哪裡被變更資料)。[13]

1. 選擇原則
   - Choose Structures by Default

[12]精通 Swift：列舉、閉包、泛型、Protocols 和高階函數

[13]Swift Class vs Struct：設計 Model 時，該用 Struct 還是 Class 呢?

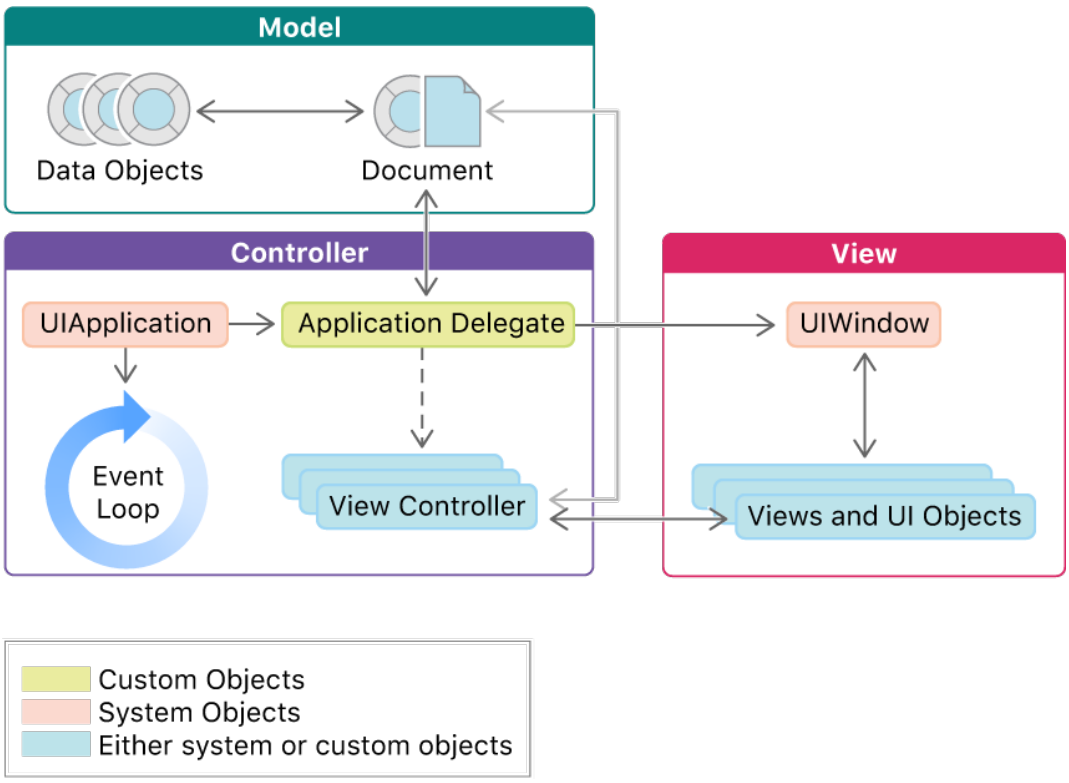

Figure 20: MVC

- Use Classes When You Need Objective-C Interoperability
- Use Classes When You Need to Control Identity
- Use Structures When You Don't Control Identity

8 Protocols

對任何程式開發來說，減少重覆的 code，把權責明確分開，讓 code 維護性變好，是非常重要的課題。而在現今的軟體開發模式中，有許多方法可以做到這點，最為人所知的一個模式，就是利用繼承 (Inheritance)，把會重覆利用的部份放在母類別，讓其它子類別去繼承。另外一種做法，則是利用 Composition Pattern，將功能做成組件分出來，讓需要的模組去組合取用。[14]

> A protocol defines a blueprint of methods, properties, and other requirements that suit a particular task or piece of functionality. The protocol can then be adopted by a class, structure, or enumeration to provide an actual implementation of those requirements. Any type that satisfies the requirements of a

[14] 利用 Protocol Extension 減少重覆的 Code 大大增強 Code 的維護性

protocol is said to conform to that protocol. [15]

協定提供類型可以做的資訊，Classes 和 structs 則提供物件的資訊，協定則提供物件將會執行的動作。[16]

協定是 Swift 一個重要的特性，它會定義出為了完成某項任務或功能所需的方法、屬性，但是本身不會實作這些任務跟功能，而僅僅只是表達出該任務或功能的名稱。協定為方法、屬性、以及其他特定的任務需求或功能定義藍圖。協定可被 class、struct、或 enum 類型採納以提供所需功能的具體實現。滿足了協定中需求的任意類型都叫做遵循了該協定。

除了指定遵循類型必須實現的要求外，你可以擴展一個協定以實現其中的一些需求或實現一個符合類型的可以利用的附加功能。[17]

8.1 範例

1. 版本 1 本例中有兩個 struct: Song, Album 以及一個 class 用來播放 Song 或 Album，原本的 Player 要為不同的 struct 寫不同的 func，而且程式碼大多重複。

```
1  import Cocoa
2  import AVKit
3
4  struct Song {
5      var name: String
6      var album: Album
7      var audioURL: URL
8      var isLiked: Bool
9  }
10
11 struct Album {
12     var name: String
13     var imageURL: URL
14     var audioURL: URL
15     var isLiked: Bool
```

[15] Protocols

[16] Swift 開發指南：Protocols 與 Protocol Extensions 的使用心法

[17] Day-29 Swift 語法 (25) - 協定 Protocol

```
16 }
17
18 class Player {
19     private let avPlayer = AVPlayer()
20
21     func play(_ song: Song) {
22         let item = AVPlayerItem(url: song.audioURL)
23         avPlayer.replaceCurrentItem(with: item)
24         avPlayer.play()
25     }
26
27     func play(_ album: Album) {
28         let item = AVPlayerItem(url: album.audioURL)
29         avPlayer.replaceCurrentItem(with: item)
30         avPlayer.play()
31     }
32 }
```

2. **版本 2 宣告一個** protocol，**定義** audioURL **變數** (read only)，**然後令兩個** struct **皆遵循該** protocol(**方式有二**)，**如此，原本的** Player class **中的** play func **就能只寫一次**。

```
1 import Cocoa
2 import AVKit
3
4 protocol Playable {
5     var audioURL: URL { get }
6 }
7
8 struct Song: Playable {
9     var name: String
10    var album: Album
11    var audioURL: URL
12    var isLiked: Bool
13 }
14
15 struct Album {
16    var name: String
17    var imageURL: URL
18    var audioURL: URL
```

```
19      var isLiked: Bool
20  }
21
22  extension Album: Playable {}
23  class Player {
24      private let avPlayer = AVPlayer()
25
26      func play(_ resource: Playable) {
27          let item = AVPlayerItem(url: resource.audioURL)
28          avPlayer.replaceCurrentItem(with: item)
29          avPlayer.play()
30      }
31  }
```

3. **版本 3 原本** protocol **的真正意思其實只是在確定** audioURL **是否能正確轉換成** Audio**，所以其實將** protocol name **由** Playable **改為** AudioURLConvertable **會更貼近事實。**

```
1   import Cocoa
2   import AVKit
3
4   protocol AudioURLConvertable {
5       var audioURL: URL { get }
6   }
7
8   struct Song: AudioURLConvertable {
9       var name: String
10      var album: Album
11      var audioURL: URL
12      var isLiked: Bool
13  }
14
15  struct Album: AudioURLConvertable {
16      var name: String
17      var imageURL: URL
18      var audioURL: URL
19      var isLiked: Bool
20  }
21
22  class Player {
```

```
23      private let avPlayer = AVPlayer()
24
25      func play(_ resource: AudioURLConvertable) {
26          let item = AVPlayerItem(url: resource.audioURL)
27          avPlayer.replaceCurrentItem(with: item)
28          avPlayer.play()
29      }
30  }
```

8.2 mutating

protocol **除了可以提供傳回值型態的彈性，也可以用來變更** class/struct **中的屬性。如：**

```
1  import Cocoa
2  import AVKit
3
4  protocol Likeable {
5      mutating func markAsLiked()
6  }
7
8  struct Song {
9      var name: String
10     var album: Album
11     var audioURL: URL
12     var isLiked: Bool
13 }
14
15 struct Album {
16     var name: String
17     var imageURL: URL
18     var audioURL: URL
19     var isLiked: Bool
20 }
21
22 extension Song: Likeable {
23     mutating func markAsLiked() {
24         isLiked = true
25     }
26 }
```

可以在不改變原 struct Album 的情況下，藉由 extension 來擴充 Song，使其遵循 Likeable protocol，提供變供屬性 isLiked 的值，* 這在擴充 API 功能時特別有用 *。

8.3 擴充 protocol

除了擴充現有 struct，protocol 也可以用來擴充 protocol，如：

```
1  import Cocoa
2  import AVKit
3
4  protocol Likeable {
5      var isLiked: Bool {get set}
6  }
7
8  extension Likeable {
9      mutating func markAsLiked() {
10         isLiked = true
11     }
12 }
13
14 struct Song {
15     var name: String
16     var album: Album
17     var audioURL: URL
18     var isLiked: Bool
19 }
20
21 struct Album {
22     var name: String
23     var imageURL: URL
24     var audioURL: URL
25     var isLiked: Bool
26 }
27
28 extension Song: Likeable {}
29 extension Album: Likeable {}
```

9 some

Adding the keyword some in front of a return type indicates that the return type is opaque. [18]

9.1 Generics

1. **問題** Generics **允許開發者在不同類型中複用你的程式碼，用來解決下列問題：**

```
1 func swapInts(_ a: inout Int, _ b: inout Int) {
2     let temporaryB = b
3     b = a
4     a = temporaryB
5 }
6
7 var num1 = 10
8 var num2 = 20
9
10 swapInts(&num1, &num2)
11 print(num1)    // 20
12 print(num2)    // 10
```

```
20
10
```

但若想交換字串，則要寫成

```
1 func swapStrings(_ a: inout String, _ b: inout String) {
2     let temporaryB = b
3     b = a
4     a = temporaryB
5 }
```

可以發現除了參數之外，重複的 code **實在太多**

2. **解決方案 將固定型態的參數轉為** Generic type

```
1 import Cocoa
```

[18] What's this "some" in SwiftUI?

```
2
3  func swapTwoValues<T>(_ a: inout T, _ b: inout T) {
4      let temporaryA = a
5      a = b
6      b = temporaryA
7  }
8
9
10 var num1 = 10
11 var num2 = 20
12
13 swapTwoValues(&num1, &num2)
14 print(num1)    // 20
15 print(num2)    // 10
```

```
20
10
```

另一個例子為 Stack 的實作：
原本只能儲存 Int 的 Stack 如下，若要儲存字串則要再另行定義。

```
1  struct IntStack {
2      var items = [Int]()
3      mutating func push(_ item: Int) {
4          items.append(item)
5      }
6      mutating func pop() -> Int {
7          return items.removeLast()
8      }
9  }
```

改為 Generic type 後可動態變更為整數 stack 或字串 stack，如：

```
1  struct Stack<Element> {
2      var items = [Element]()
3      mutating func push(_ item: Element) {
4          items.append(item)
5      }
6      mutating func pop() -> Element {
7          return items.removeLast()
```

```
8       }
9   }
10
11  var stackOfStrings = Stack<String>()
12  stackOfStrings.push("uno")
13  stackOfStrings.push("dos")
14  stackOfStrings.push("tres")
15  stackOfStrings.push("cuatro")
16  // the stack now contains 4 strings
```

9.2 opaque

帶有不透明（opaque）返回類型的函數或方法，將會隱藏其返回值的類型[19]

10 進階主題

- **利用** Protocol Extension **減少重覆的** Code **大大增強** Code **的維護性**
- **精通** Swift**：列舉、閉包、泛型、**Protocols **和高階函數**

[19]Swift **程式語言**—Opaque Types