
第九周 Swift UI

Danny Shen

本節的主題

Swift UI 是什麼？

必要條件與限制

一些簡單的案例

與 Story Board 交互應用

為什麼要用 Swift UI

Swift UI 是什麼

WWDC 2019 發佈，它是一種有效率的頁面建構方式

Swift 是一個新的程式庫 (對，它不是一個UI設計方法，它是程式庫)，它的位階類似 UIKit，是最上層的東西，所以 View 元件和 UIKit 不完全重覆

它的設計理念也和 UIKit 完全不同，基本上不是以座標系統為主，而是以容器 (Container) 系統為主，它比較像是 Flutter 或 React Native 的設計理念

沒有分為 UI 檔和程式檔，但仍可以預覽結果檔案

大量的使用 Closure 來設計你的 code，在大多數的狀況下，可以減少許多 UI 設計與拉頁面的程式碼

基本條件

限用於 iOS 13~ macOS 10.15~ (舊版不能用, 無法包版),

和同一期的 tvOS 13, watchOS 6

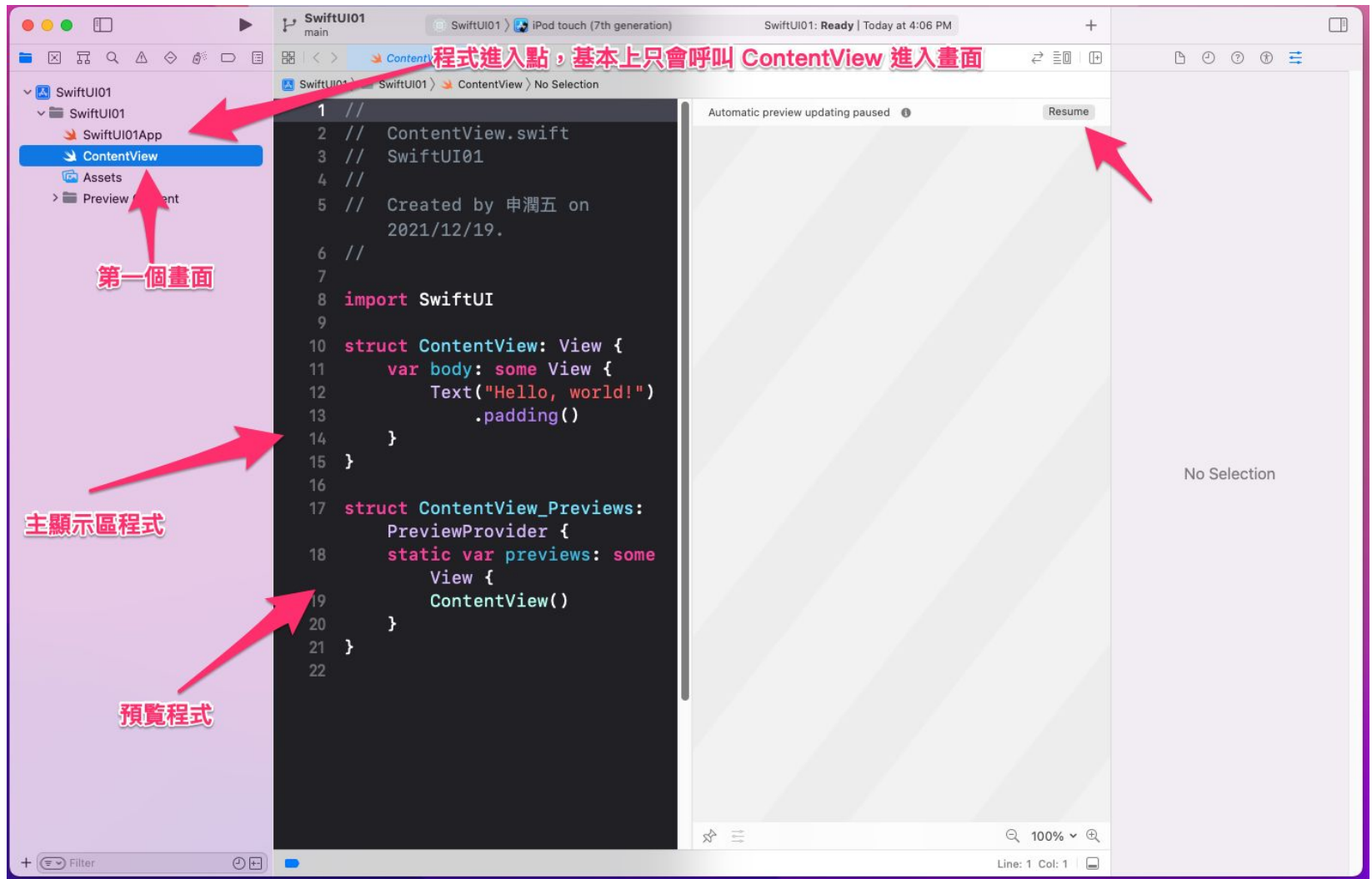
開發工具 Xcode 11 以上

開發環境最好是 macOS 10.15 以上, 之前的版本不完整支援

只能使用 Swift 不支援 Objective-C

來, 開始我們的 Swift UI 吧

建立一個新的 Swift 專案



APP 程式的開始

進入點不再是一個 ViewController

而是一個 View (注意, 是 View, 不是 UIView)

預設進入點名稱叫 ContentView

它必需有一個名為 body 屬性, 是一個 some View 類的實體

所以初學的重點就在如何寫這個叫 body 的 some View

some View 指可能是任何一個與 View 相同動作的物件, 列如 Text, Image 等

ContentView_Previews 與 previews 是類似的存在, 但只用在預覽, 所以在最終執行時, ContentView_Previews 是無關緊要的

畫面其實是一種 Container

Container 是一種放置 UI 元件的容器，有各種不同的 Container，基本上把內容 (content) 放到 container 中，元件就會自動排列了

每個 View 只能有一個主要的 Container，但 Container 中，可以有其他的 Container 作巢狀排列

```
VStack {  
    Text("Hello, World!")  
    Text("This is my first SwiftUI")  
}
```

為什麼是 some View 不是 View

- 因為.....
 - View 是一個 struct 不是一個 class 不具繼承功能
- 所以 是回應 some View 而不是 View
 - 一個 UIButton class 繼承了 UIView class, 所以程式要求一個 UIView 時, 你可以回應一個 UIButton
 - 一個 Button struct 並沒有繼承 View struct, 所以程式要求一個 View 時, 不可以回應一個 Button
 - 所以程式要求 some View 的意思是, 要求一個和 View 行為模式相同的物件, 只要能生成畫面即可, 不一定要繼承 View, 因為實務上, 也無法繼承

試試自訂格式

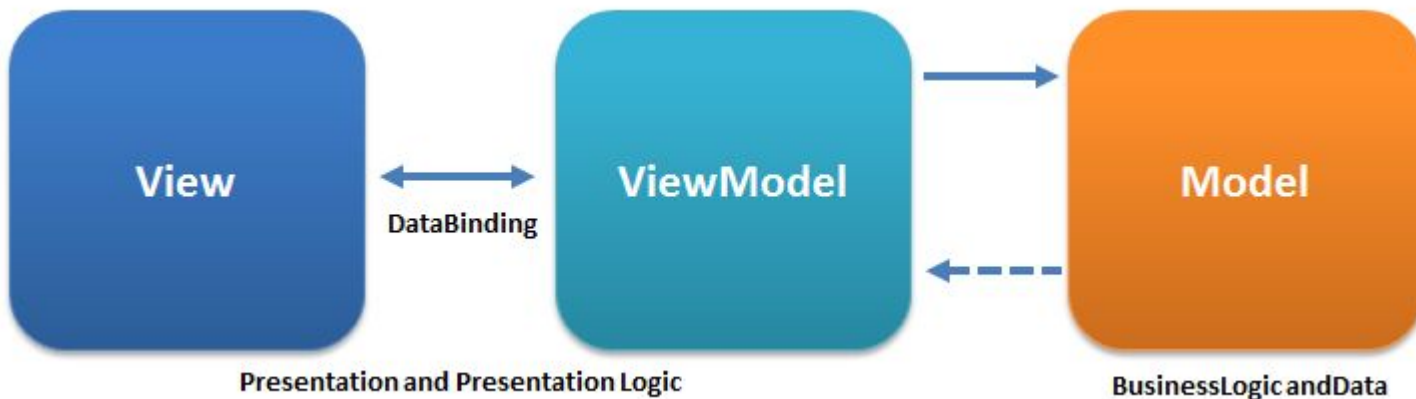
```
VStack {  
  Text("AAA").font(.largeTitle)  
  Text("BBB")  
  HStack {  
    Text("AAA")  
    Text("BBB")  
  }  
  ZStack {  
    Text("AAA")  
    Text("BBB")  
  }  
}
```

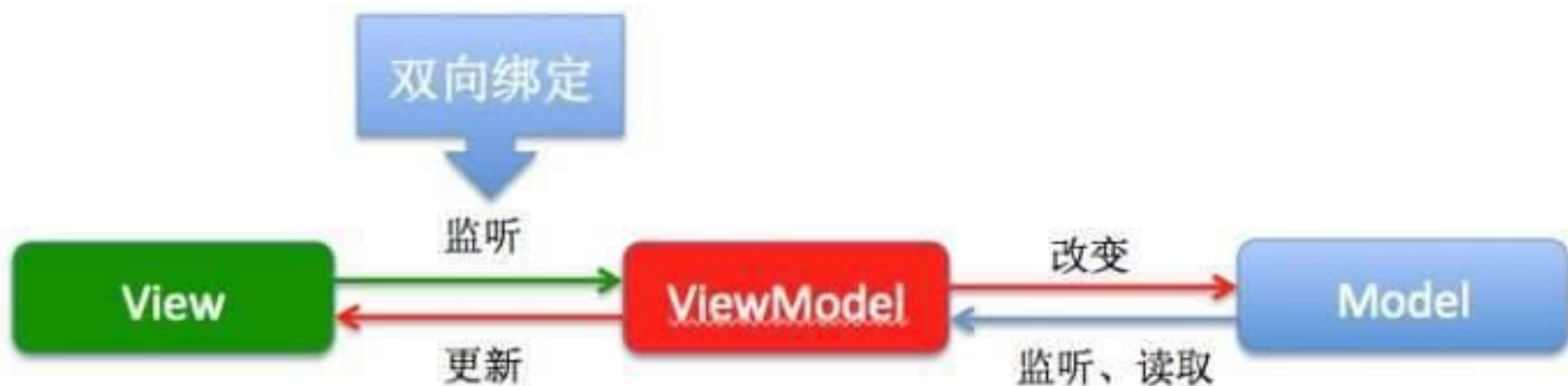
MVVM 架構

<https://zh.wikipedia.org/wiki/MVVM>

不同於 UIKit 使用 MVC 架構, SwiftUI 使用的是 MVVM 架構
(Model-view-viewmodel)

<https://zh.wikipedia.org/wiki/MVVM#/media/File:MVVMPattern.png>



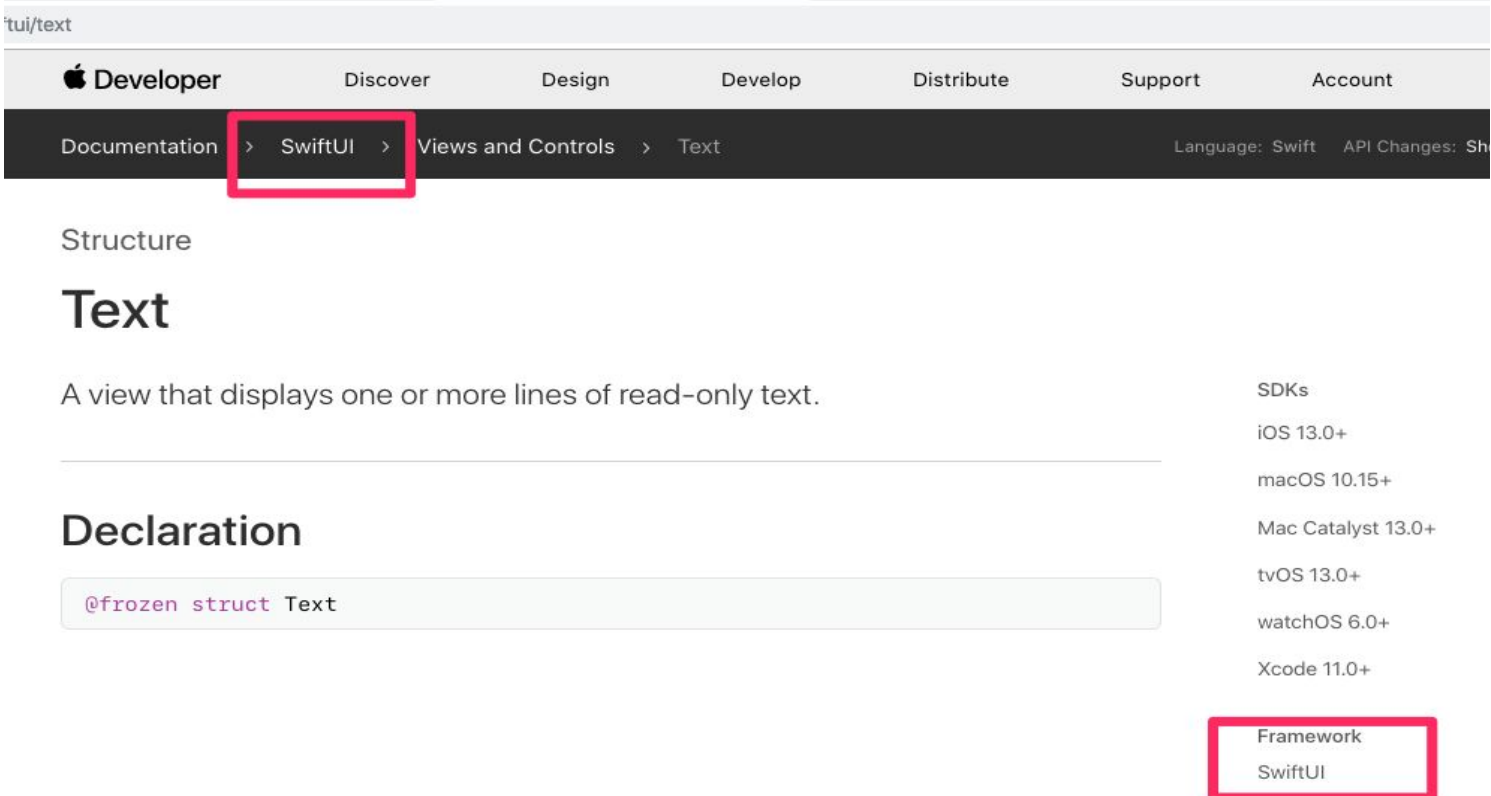


Model-View-ViewModel

搜尋官方文件

google 語法

【Text SwiftUI inurl:apple.com】



The screenshot shows the Apple Developer website's documentation for SwiftUI's Text view. The top navigation bar includes links for Developer, Discover, Design, Develop, Distribute, Support, and Account. Below this, a breadcrumb trail shows the path: Documentation > SwiftUI > Views and Controls > Text. The 'SwiftUI' link in the breadcrumb is highlighted with a red box. The main content area is titled 'Text' and describes it as a view for displaying read-only text. Below the description is a 'Declaration' section with a code snippet: `@frozen struct Text`. On the right side, there is a list of SDKs supported by the framework, including iOS 13.0+, macOS 10.15+, Mac Catalyst 13.0+, tvOS 13.0+, watchOS 6.0+, and Xcode 11.0+. At the bottom right, a red box highlights the 'Framework' section, which lists 'SwiftUI'.

tu/text

Developer Discover Design Develop Distribute Support Account

Documentation > SwiftUI > Views and Controls > Text Language: Swift API Changes: Sh

Structure

Text

A view that displays one or more lines of read-only text.

Declaration

```
@frozen struct Text
```

SDKs

- iOS 13.0+
- macOS 10.15+
- Mac Catalyst 13.0+
- tvOS 13.0+
- watchOS 6.0+
- Xcode 11.0+

Framework

SwiftUI

UIKit 的 UIViewController vs. SwiftUI 的 View

是一種 class

通常是 UIViewController 的 屬性

由事件驅動更新

由上層繼承下來層層的物件

頁面切換由多個 ViewController 組成

由Storyboard Xib 預覽

是一種 struct

通常由 SceneDelegate 來啟動

由資料驅動更新

由回傳值套用some View不是固定物件

頁面切換由不同 View 與資料切換

編譯 _Preview 實現 Canvas 預覽

TabView & 資料驅動 (binding)

實作 TabBar View

@State 為狀態設定的變數，該變數直接加上 \$ 然後放在取值的地方，要顯示地方就值接放變數，當變數有變化時，其值就會影響到畫面重新繪製

加上 \$ 的操作，稱之為【綁定】(binding)

```
@State var activeTab: Int = 0
var body: some View {
    TabView(selection: $activeTab) {
        Text("in page \$(activeTab)")
            .tabItem {
                Image(systemName: "list.bullet")
            }.tag(1)
        Text("in page \$(activeTab)")
            .tabItem {
                Image(systemName: "list.bullet")
            }.tag(2)
    }
}
```

SwiftUI 的畫面更新

更新畫面時機為【binding 的 資料變動時】

相當於目前畫面的 some View 中的任何變數, 都在 setter 中, 加入更新畫面的程式

每一次數值改變, 都會刷新整個可視範圍的 some View

思考的改變:

UIKit:

執行那個動作時, 畫面應該更新

SwiftUI:

1. 資料最後如何出現在畫面上
2. 如何更新資料

有參數的 View

如果一個 `@State` 的屬性，沒有變預設值，建構器就會只留下必需輸入屬性的建構器，我們就可以在建立時，輸入該項目的建構器

```
@State var displayString:String
var body: some View {
    VStack{
        Text(displayString)
        HStack{
            Text("請輸入：")
            TextField("", text: $displayString)
        }
    }
}
```


製作一個調色盤

使用 Slider

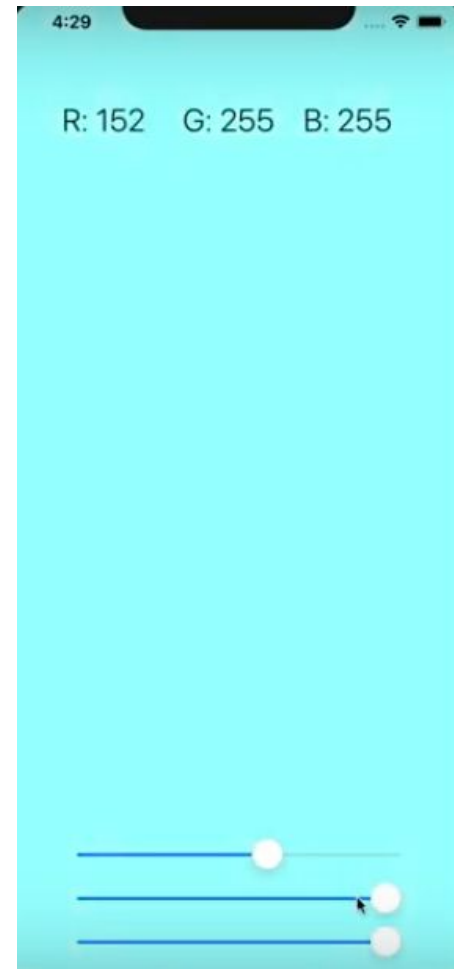
使用 Color

學習使用各種排版方式, 對齊等

Zstack, Spacer

自訂子 View

使用 binding 來改變顏色



製作更複雜的畫面

CS193P

使丹佛大學每年會開一門課，通常在春季(4~7月)，使用前一年秋季發版的最新技術，YouTube 上的 Stanford 頻道上即可找到它 (即俗稱白鬍子老頭Paul Hegarty 的課程)。

這個課程雖然是面對 iOS 新手，但並不是給完全無經驗的人開的，必需最好有 OOP 經驗，並自習過 Swift 的基本語法，最好有其他平台的程式開發經驗。

2021 年的課程，第一範例就是一個俗稱【神經衰弱】的翻牌配對遊戲。

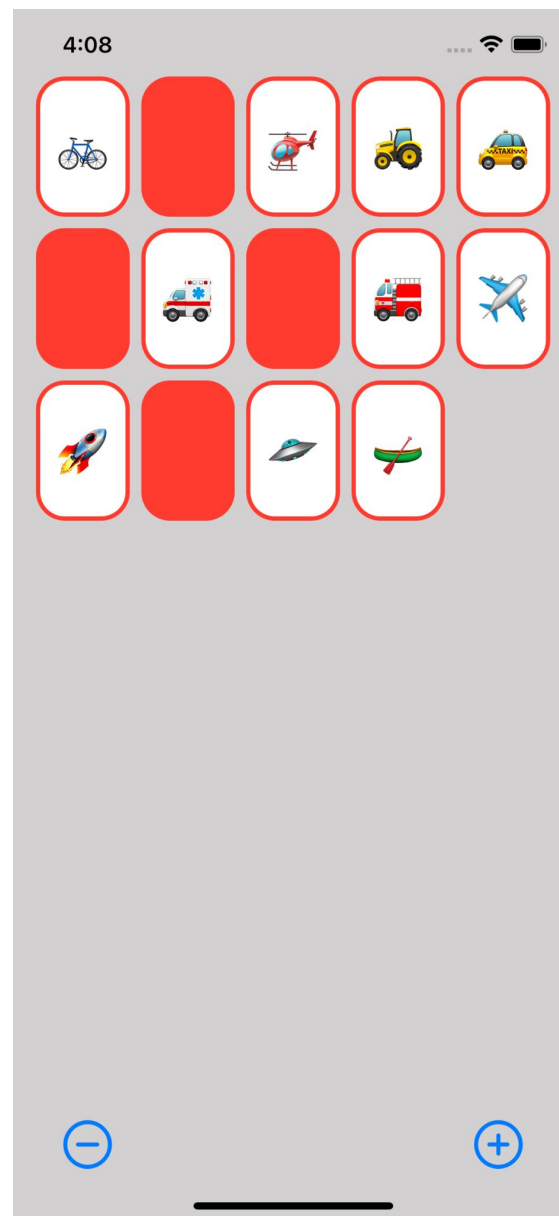
這個程式佔了 CS193P 課程的 $\frac{1}{4}$ 我們就來實作這個課程的一部份



實作目標

CSP193 前兩節的課程

1. 可新增減少卡片的按鈕
2. 按下卡片可以翻面



主 Content View 與 參數

```
var emojis = ["🚲", "🚂", "🚁", "🚜", "🚕", "🚗", "🚑", "🚒", "🚓", "🚚", "✈️", "🚀", "🚢", "🚤", "🛥️",  
"🚊", "🏍️", "🚒", "🚏", "🛵", "🚗", "🚚", "🚂", "🚗", "🚊"]
```

```
@State var emojiCount = 10
```

```
var body: some View {
```

```
    VStack {
```

```
        ScrollView {
```

```
            LazyVGrid(columns: [GridItem(.adaptive(minimum: 65))]) {
```

```
                ForEach(emojis[0..
```

```
                    CardView(content: emoji)
```

```
                        .aspectRatio(2/3, contentMode: .fit)
```

```
                }
```

```
            }
```

```
        }
```

```
        .foregroundColor(.red)
```

```
        Spacer()
```

```
        HStack {
```

```
            remove
```

```
            Spacer()
```

```
            add
```

```
        }
```

```
        .font(.largeTitle)
```

```
        .padding(.horizontal)
```

```
    }
```

```
    .padding(.horizontal)
```

```
}
```

ForEach

```
ForEach(emojis[0..<emojiCount], id: \.self) { emoji in  
    CardView(content: emoji)  
        .aspectRatio(2/3, contentMode: .fit)  
}
```

ForEach 有三個參數

- 內容集合
- id: 可使用 `\.self` 把自己當成 id
- 產生 some View 的 Closures

LazyVGrid

```
LazyVGrid(columns: [GridItem(.adaptive(minimum: 65))]) {  
    ForEach(emojis[0..        CardView(content: emoji)  
            .aspectRatio(2/3, contentMode: .fit)  
    }  
}
```

一個容器，將 some View 排列在垂直增長的網格中，僅根據需要創建項目(看不到的不建)

參數 columns: 每一行應有的 some View 格式

新增/刪除 按鈕 View

```
var remove: some View {  
    Button {  
        if emojiCount > 1 {  
            emojiCount -= 1  
        }  
    } label: {  
        Image(systemName:  
"minus.circle")  
    }  
}
```

```
var add: some View {  
    Button {  
        if emojiCount <  
emojis.count {  
            emojiCount += 1  
        }  
    } label: {  
        Image(systemName:  
"plus.circle")  
    }  
}
```

卡片 CardView

```
struct CardView: View {
    var content: String
    @State var isFaceUp = true

    var body: some View {
        ZStack {
            let shape = RoundedRectangle(cornerRadius: 20)
            if isFaceUp {
                shape.fill().foregroundColor(.white)
                shape.strokeBorder(lineWidth: 3)
                Text(content)
                    .font(.largeTitle)
            } else {
                shape.fill()
            }
        }
        .onTapGesture {
            isFaceUp = !isFaceUp
        }
    }
}
```


接下來的動作

<https://www.youtube.com/playlist?list=PLpGHT1n4-mAsxuRxVPv7kj4-dQYoC3VVu>

<https://cs193p.sites.stanford.edu/>

或搜尋 “CS193P”

SwiftUI + UIKit

— 先用 SwiftUI 還是 UIKit？ —

為什麼要混用？

- 很多東西 SwiftUI 還不完美，還不是無所不能
- 第三方 ViewController 的功能，無法直接加入
- 有些東西，你已經會寫 UIKit 了，SwiftUI 可能也可以做出來，但需花時間研發
- 總之，可以解決很多相容性的問題

在 SwiftUI 中使用 UIView

- How ? (建立一個中介 struct)
 - 建立一個 struct 添加 UIViewRepresentable Protocol
 - 實作 `func makeUIView(context: Context) -> some UIViewController {}`
 - 實作 `func updateUIView(_ uiView: TheView, context: Context) {}`
 - 把該 struct 當成 SomeView 使用
- 要注意的事
 - frame, bound 不會產生作用
 - AutoLayout 會產生作用

TheViewForSwiftUI

```
import SwiftUI
struct TheViewForSwiftUI: UIViewRepresentable{
    typealias UIViewType = TheView
    func makeUIView(context: Context) -> TheView {
        return TheView()
    }
    func updateUIView(_ uiView: TheView, context: Context) {

    }
}
```

在 SwiftUI 中使用 UIViewController

- How ? (建立一個中介 struct)
 - 建立一個 struct 添加 UIViewControllerRepresentable Protocol
 - 實作 func makeUIViewController(context: Context) -> some UIViewController{}
 - 實作 func updateUIViewController(_ uiViewController: UIViewControllerType, context: Context) {}
 - 把該 struct 當成 SomeView 使用
- 要注意的事
 - push present tab 等等的效果, 可能和你想像的不盡相同, 請多做測試

TheVCforSwiftUI

```
import SwiftUI
struct TheVCforSwiftUI: UIViewControllerRepresentable{
    func makeUIViewController(context: Context) -> some UIViewController {
        let nav = UINavigationController()
        nav.addChild(TheViewController())
        nav.popToRootViewController(animated: true)
        return nav
    }

    func updateUIViewController(_ uiViewController: UIViewControllerType, context: Context) {

    }
    typealias UIViewType = TheViewController
}
```

在 UIKit 中, 使用 SwiftUI

使用 UIHostingController 來包裝 SwiftUI 的 View

```
import UIKit
import SwiftUI

class ViewController: UIViewController {
    @IBAction func theAction(_ sender: Any) {
        let vc = UIHostingController(rootView: TheSwiftUIView())
        vc.modalPresentationStyle = .fullScreen
        self.present(vc, animated: true)
    }
}
```


使用環境參數回到原畫面

@Environment

Environment 是一些 View 的屬性，預設連自己都無法存取
要自己寫一個新的參數放回去，dismiss 就是其中一個

```
import SwiftUI
struct TheSwiftUIView: View {
    @Environment(\.dismiss) private var dismiss
    var body: some View {
        Text("Back")
            .onTapGesture {
                self.dismiss()
            }
    }
}
```

AppDelegate 不見了？

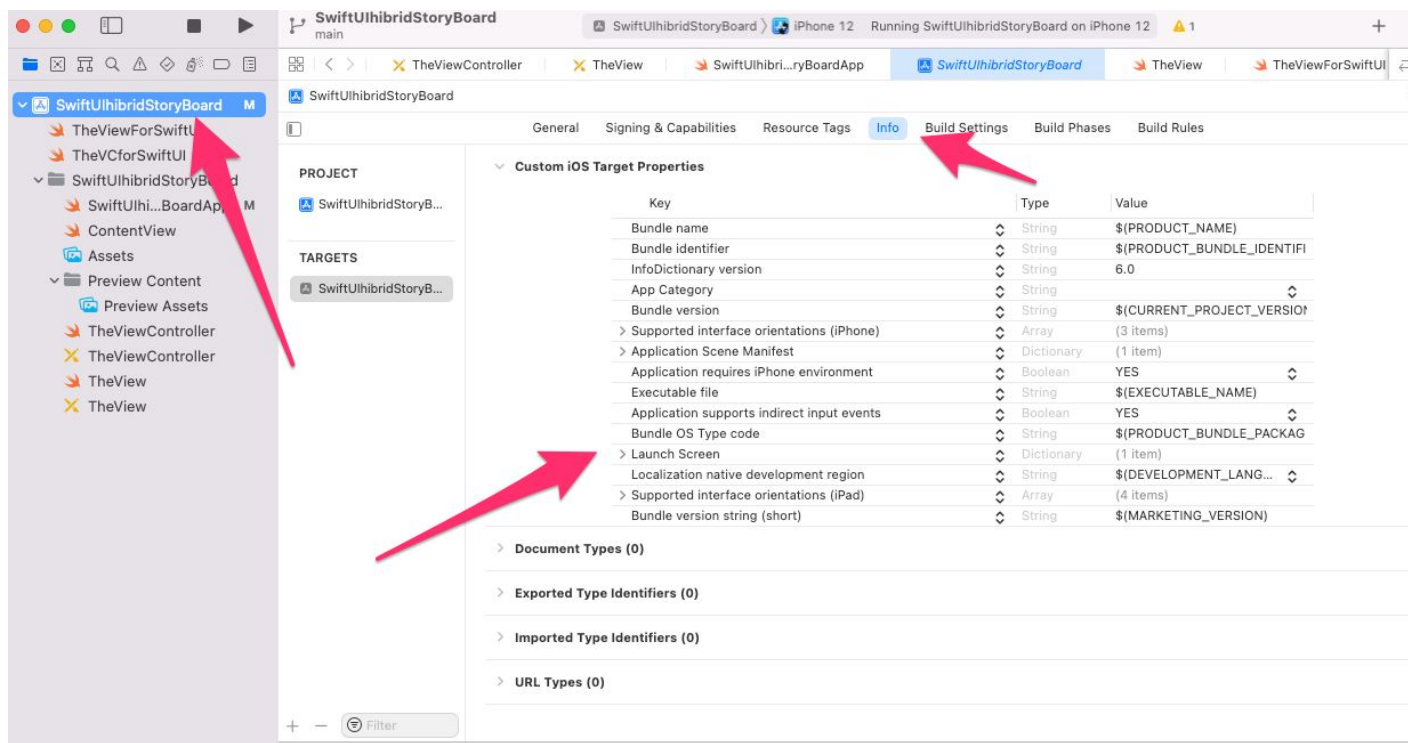
現在有一個 [APP名稱].swift 有看到嗎？雖然方法不太一樣，但你可以自己加回去

```
import SwiftUI
class AppDelegate: NSObject, UIApplicationDelegate {
    func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey : Any]? = nil) -> Bool
{
    print("log-didFinishLaunching")
    return true
}
}
```

```
@main
struct SwiftUIHybridStoryboardApp: App {
    @UIApplicationDelegateAdaptor(AppDelegate.self)
    appDelegate
    @Environment(\.scenePhase) var scenePhase
    var body: some Scene {
        WindowGroup {
            ContentView()
        }.onChange(of: scenePhase) { newScenePhase
            switch newScenePhase {
            case .active:
                print("APP 啟動了")
            case .inactive:
                print("APP 休眠了")
            case .background:
                print("APP 進入背景了")
            @unknown default:
                print("APP ...進入了一個不明狀態")
            }
        }
    }
}
```

Info.plist 也不見了？

請直接加到
Target 中



討論議題

應不應該, 想不想要用 SwiftUI

在那些情況下, SwiftUI 是優先選擇?

在那些情況下, 不應該使用 SwiftUI

什麼時候/什麼條件下, SwiftUI 會變成主流?

UIKit 會消失嗎? 還是會和 SwiftUI 並存?