swift学习笔记

丁碧云

2016-12-20

CONTENTS

swift介绍

- Swift 在各个方面优于 Objective-C，也不会有那么多复杂的符号和表达式。同时，Swift 更加快速、便利、高效、安全。除此之外，新的 Swift 语言依旧会与 Object-C 相兼容。

权限控制（Access Control）

- Swift语言从Xcode 6 beta 5版本起，加入了对权限控制（Access Control） 的支持。
- private 私有的
  - 在哪里写的，就在哪里用。无论是类、变量、常量还是函数，一旦被标记为私有的，就只能在定义他们的源文件里使用，不能为别的文件所用。
- internal 内部的
  - 标记为internal的代码块，在整个应用（App bundle） 或者框架（framework） 的范围内都是可以访问的。
- public 公开的
  - 标记为public的代码块一般用来建立API，这是最开放的权限，使得任何人只要导入这个模块，都可以访问使用。

注意：swift里面所有代码实体的默认权限，都是最常用的internal。所以当你开发自己的App时，可能完全不用管权限控制的事情。

curl的使用

- swift提供的标准API都是通过curl工具完成的，比如说PUT,GET,POST等等。
- curl基本命令：
  - -H <line> 自定义头信息传递给服务器
  - -i 输出时bao阔protocol头信息
  - -k 允许不使用证书到SSL站点
  - -v 显示详细信息
  - -X<command> 指定命令
  - -d<data> HTTP POST方式传送数据
- 获取token

swift接口使用

- 使用curl操作switf接口
  - 通过获取”token”和”publicURL”我们使用API所用到的权限。
- account操作
  - 查看当前的存储信息
  - 创建container
  - 格式化输出获取的container信息
  - 查看container的元数据、删除container
- object操作
  - 创建、删除object

上传一个object到container中，swift先把这个object传到缓存中然后才传到相应的位置，在缓存中最小单位是object我们是看不到这个东西的，不过传到continer中最小单位就是file了我们是可以看到这个东西的，可以完整的put和get获取其中的数据和元数据。

swift代码解读

family文件夹swift文件

AccessToken.swift

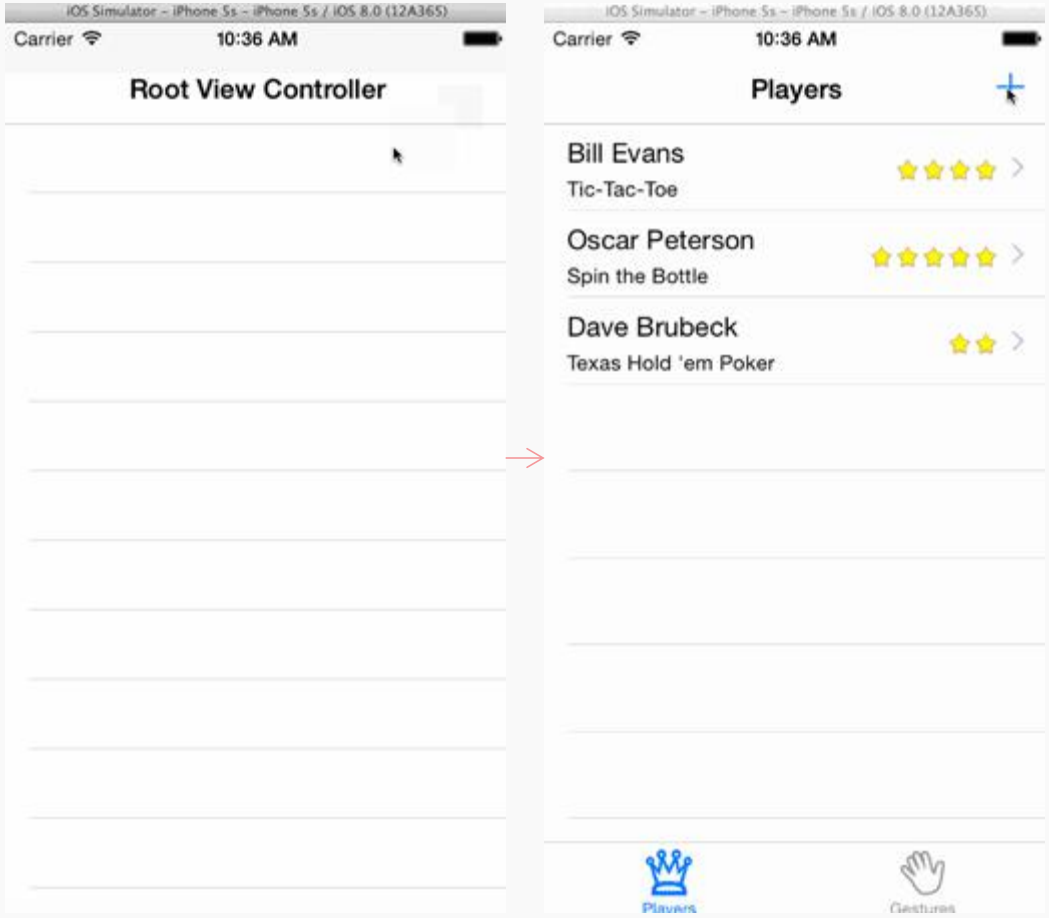
- 获取swift接口的许可

转场

- segue（转场，读作seg-way，源自电影术语，原指两个场景间的过渡衔接），表示一个页面到另一个页面的过渡。此前我们所见的Storyboard连接描述的都是视图控制器的包含关系，而转场是用来切换页面的。转场可以由点击按钮、表项、手势等条件触发。
- 使用转场的好处是，再也不用为呈现新页面写代码了，也不用把按钮连接到IBAction方法上，你只需要在Storyboard中从一个栏按钮项拖到下一个页面就可以创建过渡了。（注：如果你的控件已经绑定了IBAction连接，该连接会被转场屏蔽。）

模态转场

- 右边就是所谓的modal（模态）转场。新页面完全覆盖原页面，在关闭模态页面之前，用户只能在新页面进行交互。后面我们还会看到push（入栈）转场，这种转场会把新页面压入导航控制器的导航栈（navigation stack）。
- 为返回页面，Storyboard提供了unwind（回退）转场。接下来我们要实现返回功能，主要分三个步骤：
  - 1. 创建让用户点选的控件，通常是个按钮。
  - 2. 在你想返回的控制器创建回退方法。
  - 3. 在Storyboard中将控件与回退方法连接。



swift的Storyboard教程

- Storyboards Tutorial in iOS 9: Part 1
  - https://www.raywenderlich.com/113388/storyboards-tutorial-in-ios-9-part-1
  - 代码为：RatingsPt1-iOS8.zip文件
- Swift语言Storyboard教程：第二部
  - http://www.cocoachina.com/swift/20150114/10924.html

Cocoa中常见的设计模式：

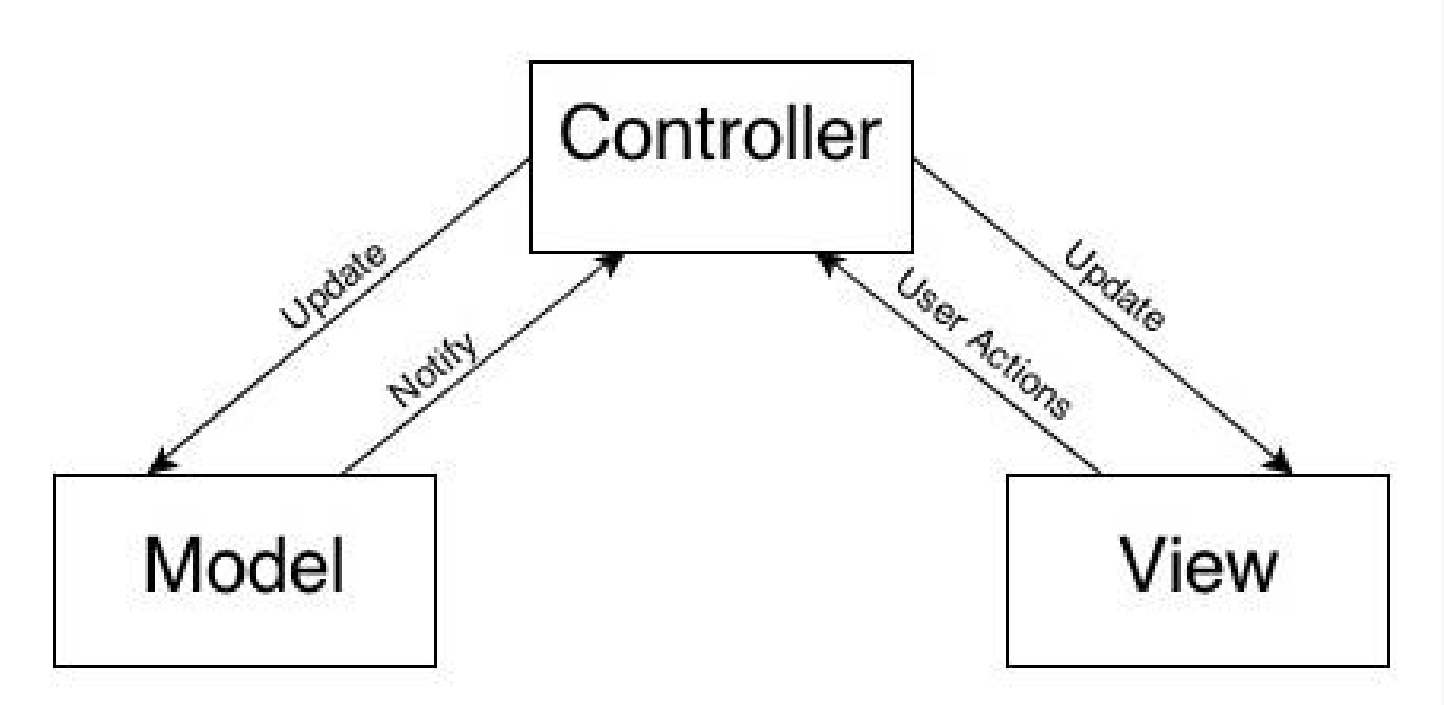
- 创建型（Creational）：
  - 单例模式（Singleton）
- 结构型（Structural）：
  - MVC、装饰者模式 （Decorator）、适配器模式（Adapter）、外观模式 （Facade）
- 行为型（Behavioral）：
  - 观察者模式（Observer）、备忘录模式（Memento）

参考：http://www.csdn.net/article/2015-01-19/2823615-ios-design-patterns-in-swift

MVC——设计模式之王

- Model-View-Controller (缩写 MVC) 是 Cocoa 框架的一部分，并且毋庸置疑是最常用的设计模式之一。它可以帮你把对象根据职责进行划分和归类。
- 作为划分依据的三个基本职责是：
  - 模型层（Model）：存储数据并且定义如何操作这些数据。在我们的例子中，就是Album类。
  - 视图层（View）：负责模型层的可视化展示，并且负责用户的交互，一般来说都是继承自UIView这个基类。在我们的项目中就是AlbumView这个类。
  - 控制器（Controller）：控制器是整个系统的掌控者，它连接了模型层和数据层，并且把数据在视图层展示出来，监听各种事件，负责数据的各种操作。不妨猜猜在我们的项目中哪个是控制器？啊哈猜对了ViewController这个类就是。

三者之间的关系

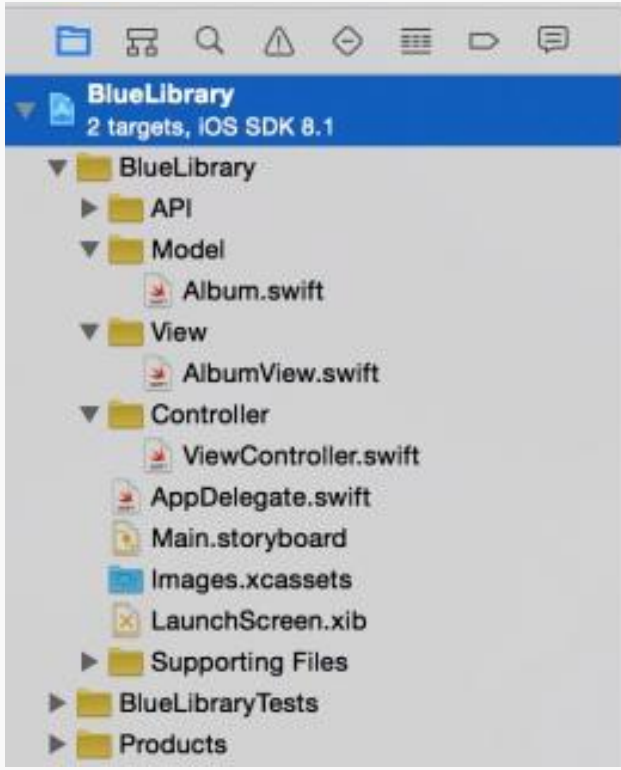


如何使用MVC模式

- 首先，你需要确定你的项目中的每个类都是三大基本类型中的一种：控制器、模型、视图。不要在一个类里糅合多个角色。目前我们创建了Album类和AlbumView类是符合要求的，做得很好。

- 然后，为了确保你遵循这种模式，你最好创建三个项目分组来存放代码，分别是Model、View、Controller，保持每个类型的文件分别独立。

- 接下来把Album.swift拖到Model分组，把AlbumView.swift拖到View分组，然后把ViewController.swift 拖到Controller分组中。



代码的一些示例

swift代码

```
var title : String!
var artist : String!
var genre : String!
var coverUrl : String!
var year : String!
```

这里创建了五个属性，分别对应专辑的标题、作者、流派、封面地址和出版年份。

```
init(title: String, artist: String, genre: String, coverUrl: String, year: String) {
  super.init()
  self.title = title
  self.artist = artist
  self.genre = genre
  self.coverUrl = coverUrl
  self.year = year
}
```

接下来我们添加一个初始化方法

```
func description() -> String {
  return "title: \(title)" +
   "artist: \(artist)" +
   "genre: \(genre)" +
   "coverUrl: \(coverUrl)" +
   "year: \(year)"
}
```

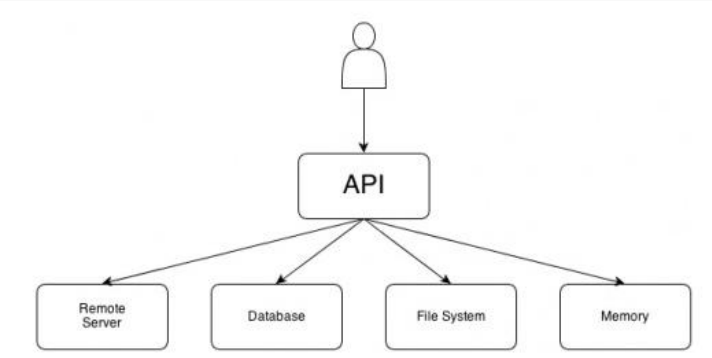
然后再加上下面这个方法

单例模式

- 根据上一页，可知现在你的内容已经组织好了，接下来要做的就是获取专辑的数据。你将会创建一个 API 类来管理数据 - 这里我们会用到下一个设计模式：单例模式。
- 单例模式确保每个指定的类只存在一个实例对象，并且可以全局访问那个实例。一般情况下会使用延时加载的策略，只在第一次需要使用的时候初始化。
- 【注意】在iOS中单例模式很常见，NSUserDefaults.standardUserDefaults()、UIApplication.sharedApplication()、UIScreen.mainScreen()、NSFileManager.defaultManager()这些都是单例模式。
- 单例模式的应用还有另一种情况：你需要一个全局类来处理配置文件。我们很容易通过单例模式实现线程安全的实例访问，而如果有多个类可以同时访问配置文件，那可就复杂多了。

外观模式

- 外观模式在复杂的业务系统上提供了简单的接口。如果直接把业务的所有接口直接暴露给使用者，使用者需要单独面对这一大堆复杂的接口，学习成本很高，而且存在误用的隐患。如果使用外观模式，我们只要暴露必要的 API 就可以了。
- API的使用者完全不知道这内部的业务逻辑有多么复杂。当我们有大量的类并且它们使用起来很复杂而且也很难理解的时候，外观模式是一个十分理想的选择。
- 外观模式把使用和背后的实现逻辑成功解耦，同时也降低了外部代码对内部工作的依赖程度。如果底层的类发生了改变，外观的接口并不需要做修改。
- 举个例子，如果有一天你想换掉所有的后台服务，你只需要修改API内部的代码，外部调用API的代码并不会有改动。



装饰者模式

- 装饰者模式可以动态的给指定的类添加一些行为和职责，而不用对原代码进行任何修改。当你需要使用子类的时候，不妨考虑一下装饰者模式，可以在原始类上面封装一层。

- 在Swift里，有两种方式实现装饰者模式：扩展（Extension）和委托（Delegation）。

委托机制与Core Location

2016-12-21

项目和目标

项目（project）

- 项目是一个文件，包含一系列指向其它文件（源代码、资源、框架和库）的应用，也包含众多和项目有关的设置。项目文件的后缀：.xcodeproj。
- 创建新项目并选择模板后，Xcode会自动创建一个目标，且名字与项目名称相同。

目标（target）

- 项目会有至少一个目标。目标使用项目中的文件构建某个特定的产品（product）。Xcode构建并运行的是目标，不是项目。
- 目标所构建的产品通常就是应用，也可以是编译后的库或是单元检测程序包。

框架（Framwork）

- 框架是一组相关类的集合，可以将其加入目标。Cocoa Touch则是一组框架的集合。通过框架组织Cocoa Touch的好处是只需要为目标加入其需要使用的框架。
- 基本框架：
  - UIKit 框架，包含用于创建iOS用户界面的类；其中的类的前缀是UI。
  - Foundation框架，包含NSString和NSArray等基础类；其中的类的前缀是NS。
  - Core Graphics，提供国兴可的调用接口。
  - Core Location，包含和设备定位相关的类；其中的类的前缀是CL。

应用可以通过使用Core Location框架所包含的类，获得设备的地理位置。

注意，类名前缀是一种约定，以避免名字空间冲突。

THANK YOU