

## 第七部分 导航

---



授课教师：杨文川



# 导航（2学时）

---

本章目的在于使学生掌握介绍几种导航控制器，包括：标签栏控制器、分页控制器、导航控制器等，并结合上一讲的表格视图，用导航控制器来构建相对复杂的树状导航。

- 标签栏导航
- 分页控制器
- 导航控制器
- 树状导航



# 标签栏导航

---

- 标签栏视图控制器（*UITabBarController*）管理多个互斥显示的可选界面，它根据用户对不同标签的选择，显示对应的子视图控制器的界面。
- 标签栏在界面的底部，由一系列图标和标题对组成。
- 每一组图标和标题构成一个标签，对应一个特定的自定义视图控制器。
- 当用户选择特定的标签时，标签栏控制器将会显示与其对应的视图控制器的视图，取代原来显示的视图。

# 典型应用



iphone系统自带的应用“时钟”就是一个典型的用标签栏来组织页面的例子。



# 构建基于标签栏导航的应用

- 在构建标签栏视图控制器时，需要将每一个标签和对应的视图控制器关联起来。
- 在实际显示的时候，由于标签栏占用了屏幕底部区域，因此标签栏控制器会将每一个视图进行缩放，从而与标签栏中的视图显示尺寸匹配。
- 标签项
  - 标签栏中的每一个标签项也有其对应的视图控制器 *UITabBarItem*，通常需要对其标题title和图标image属性进行配置。
- 委托协议
  - 当用户在操作标签栏时，标签栏控制器将会发送相应通知给它的委托。标签栏控制器的委托可以是任何遵守 *UITabBarControllerDelegate* 协议的对象。
  - 通过委托对象可以阻止特定的标签被选中、或者当某个标签被选中时执行特定的任务。



# 标签栏导航实例

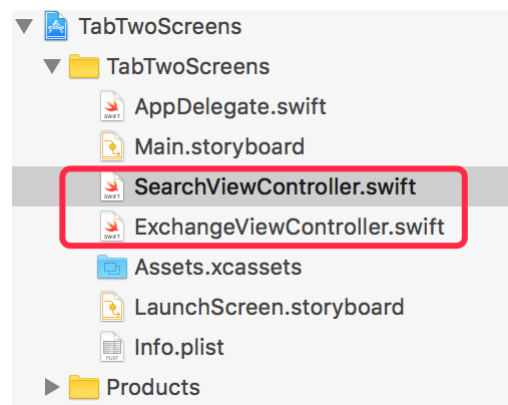
---

- 实例：复合应用（汇兑和检索）
  - 通过UITabBarController来实现该实例。
- 要求
  - 将前面介绍的汇率兑换应用和关键字检索应用合并到一个应用中，通过标签来切换不同应用的界面。

代码参见iosPrj的Chapter06-  
TableTwoScreens

# 创建项目

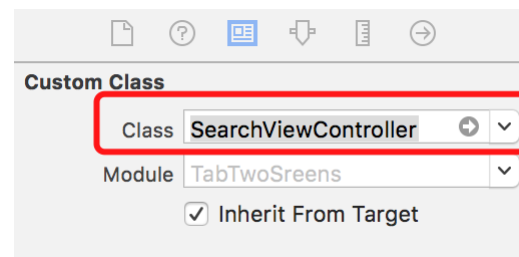
- 通过single view application模板来创建工程TabTwoScreens。模板会缺省建立一个Main.stroyboard故事板文件和一个ViewController.swift文件，删掉ViewController.swift文件。
- 根据项目需求，要将前两个应用整合到TabTwoScreens中，不需要重新编写新的视图控制器代码。
- 向项目中添加应用ExchangeRMBToDollar的文件ExchangeViewController.swift和SearchKeyword应用的文件SearchViewController.swift，



代码参见iosPrj的Chapter06-TableTwoScreens

# 构建用户界面

- 打开故事板文件 Main.storyboard，选中 ViewController，在右侧属性面板中设置自定义类的属性为 SearchViewController，从而为故事板中的视图控制器指定其对应的视图控制器文件。
- 这样便建立了视图控制器文件 SearchViewController.swift 与故事板的关联关系。

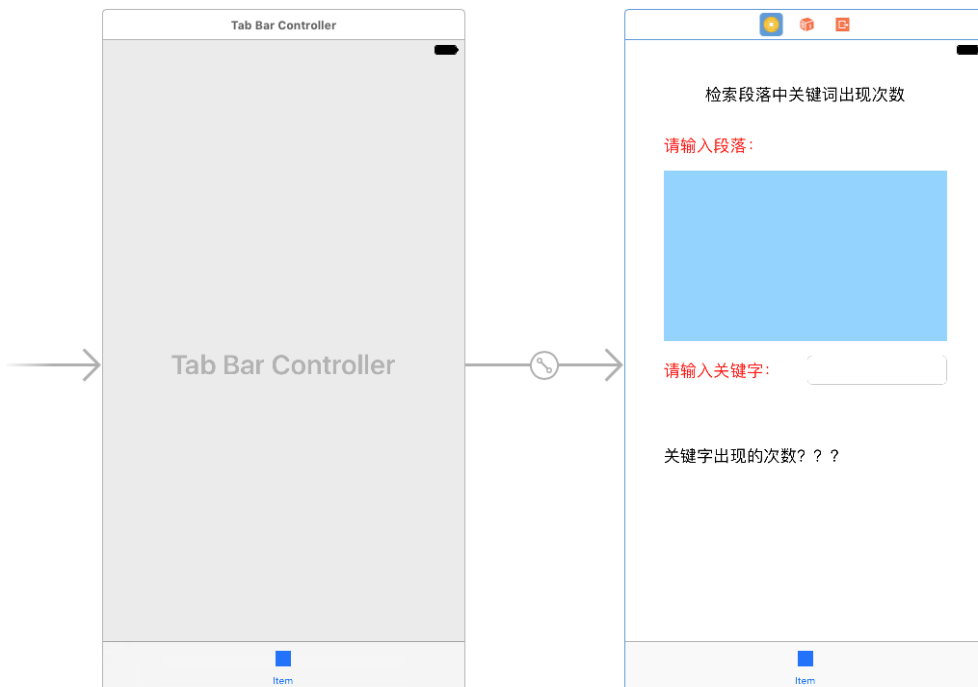




# 构建用户界面

## ■ 添加标签导航

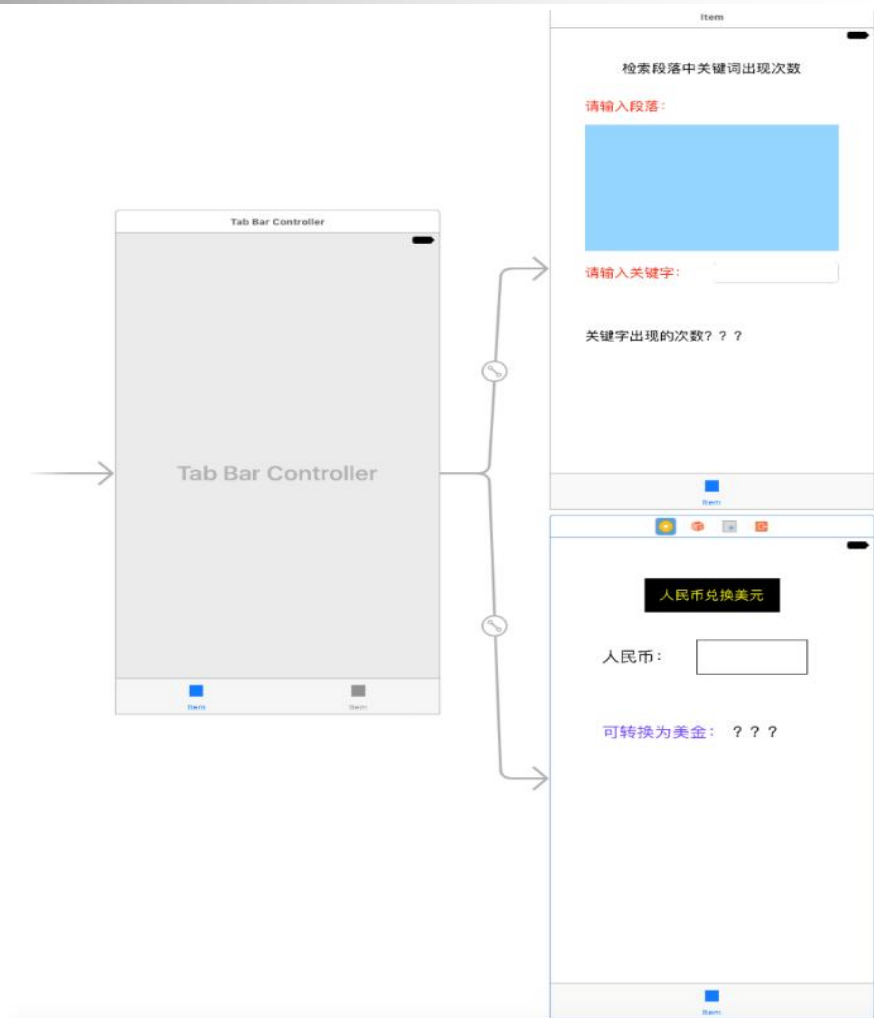
- 向项目中添加标签导航，在菜单中选中“editor”，再子菜单中选择“embed in”，然后选择“Tab Bar Controller”，系统会自动在故事板中创建一个Tab Bar Controller的视图，并建立与视图控制器SearchViewController的连接关系。
- 如图所示，用户界面的最下面出现了一个正方形图标和文字Item，该控件成为Tab Item。当有多个视图界面的时候，就应该有多个Tab Item与其对应。



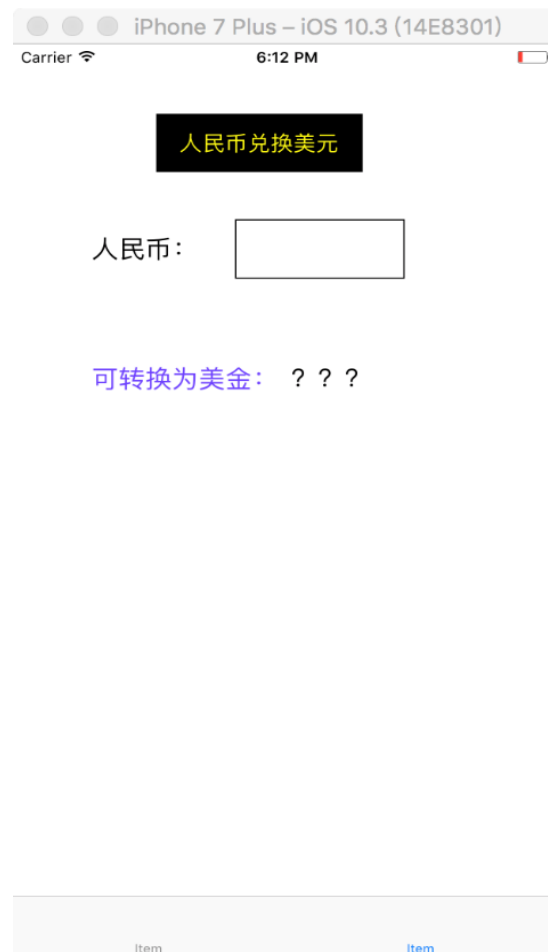
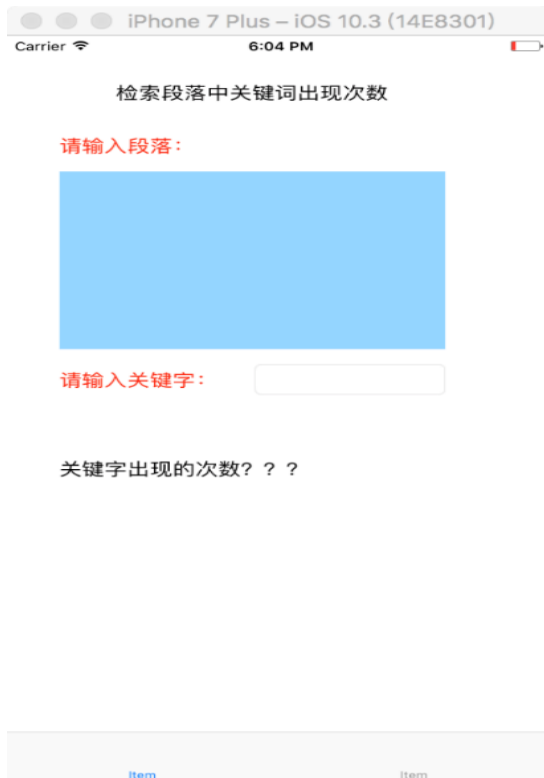
# 构建用户界面

## ■ 向项目中添加另一个应用的界面

- 在控件库中找到View Controller，并将其拖拽到故事板中。
- 该视图控制器是孤立的，与Tab Bar Controller并没有关联。选中Tab Bar Controller，按住shift和鼠标左键，拖拽一条线连接到新加入的视图控制器上，在弹出框中选择“view controller”。
- 在故事板中选中这个视图控制器，根据应用ExchangeRMBToDollar的界面设计向其中添加控件，然后为相关的控件设置委托对象，另外，还需建立相关控件与ExchangeViewController.swift文件的连接关系。

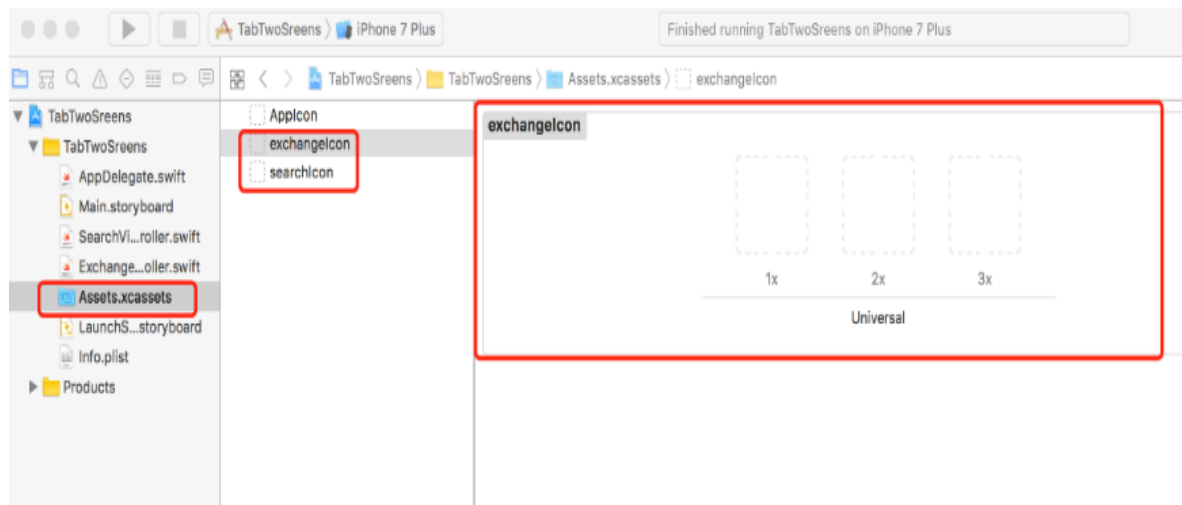


# 运行效果



# 设置标签属性

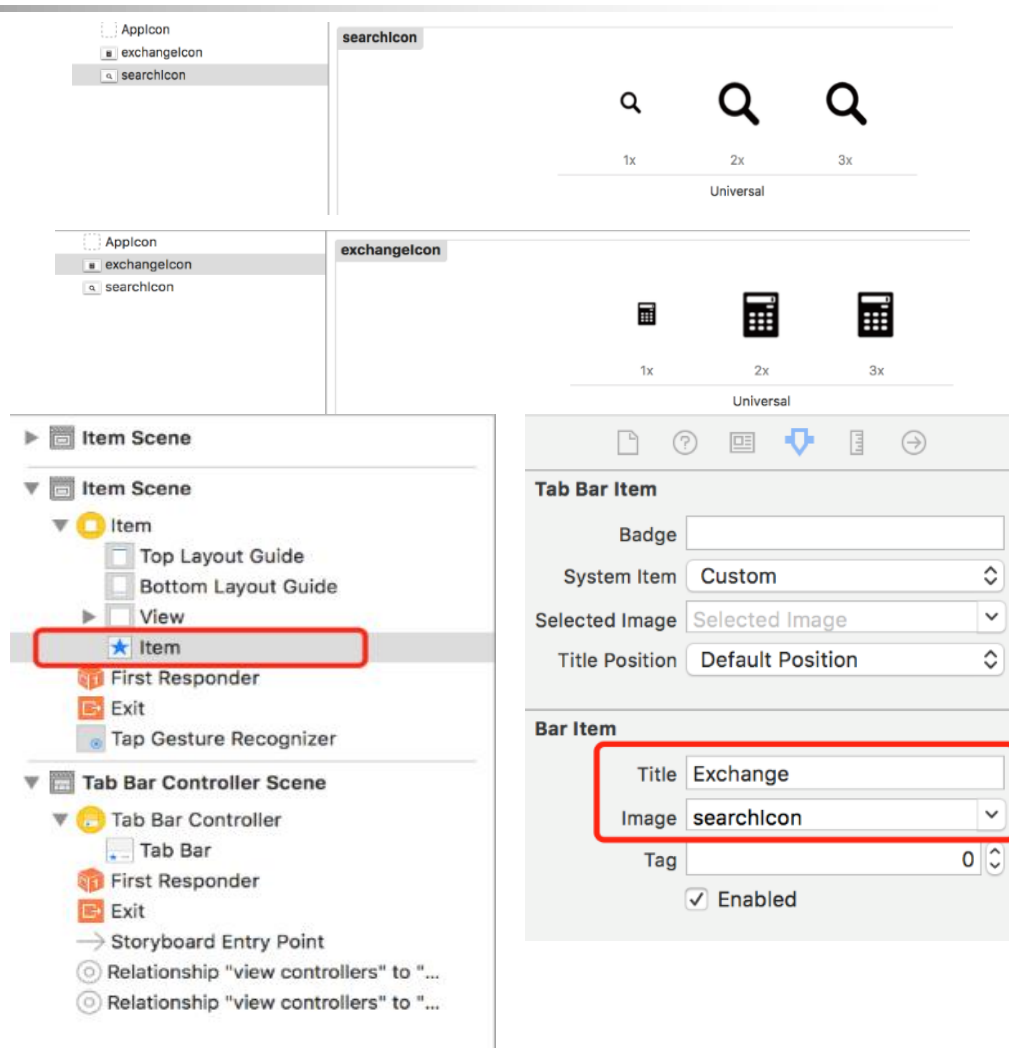
- 项目运行的时候可以看到底部标签栏只显示了文字“Item”，通过设置标签的属性为每一个标签设定不同的标题和图标，大大增加应用的用户友好性。
- 向项目中添加用作图标的图片
  - 标签的图片最好是背景透明的，否则显示效果不佳。图片需要提供三个尺寸（145像素 \* 97像素，97像素 \* 65像素，49像素 \* 33像素），以适用于不同的苹果设备
  - 在左侧的项目文件树中选中资源文件Assets.xcassets，然后点击add添加新的文件夹，并命名为exchangeIcon用来保存人民币兑换美元页面的标签。



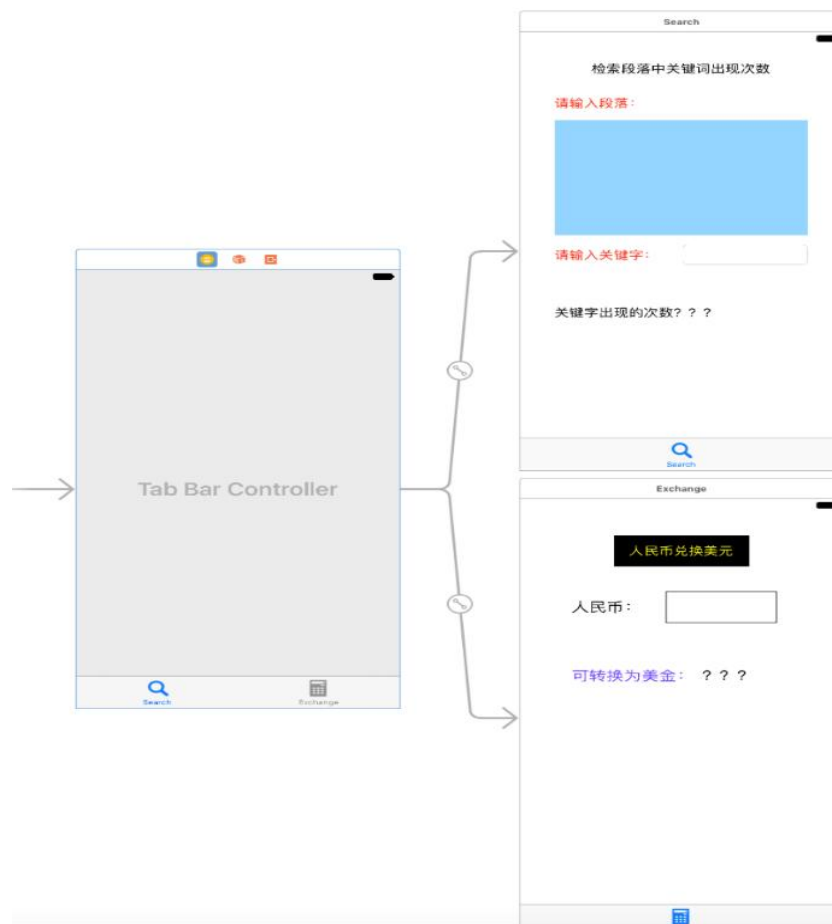
# 设置标签属性

- 将事先准备好的图片分别拖入到searchIcon和exchangeIcon的图片区。
- 设置Tab Item的标题和图标
  - 打开故事板文件，选中Item Scene中的Item，在右侧的属性面板中分别设置该标签的标题和图标文件。

代码参见iosPrj的Chapter06-TableTwoScreens



# 设置后的视图面板



# 最终运行效果

iPhone 7 Plus – iOS 10.3 (14E8301)

Carrier 7:34 PM

检索段落中关键词出现次数

请输入段落：

请输入关键字：

关键字出现的次数？？？

Search Exchange



## 第七部分 导航

---

本章目的在于使学生掌握介绍几种导航控制器，包括：标签栏控制器、分页控制器、导航控制器等，并结合上一讲的表格视图，用导航控制器来构建相对复杂的树状导航。

- 标签栏导航
- 分页控制器
- 导航控制器
- 树状导航





# 分页控制器

---

- 分页控制器 `UIPageViewController` 是一个容器视图控制器，它负责管理多个页面视图的切换。分页控制器需要通过编写代码的方式来实现，没有相应的视图控件可以直接使用。
- 在使用分页控制器时，需要为所有页面定义好相应的视图控制器。
- 当显示某一个页面的内容时，使用方法 *`setViewContronllers(_:direction:animated:completion:)`* 来设置当前显示页面的控制器。
- 分页控制器需要遵守两个协议
  - `UIPageViewControllerDataSource`（分页控制器数据源协议）
  - `UIPageViewControllerDelegate`（分页控制器委托协议）。



# 两个重要协议

---

- 分页控制器数据源协议：负责根据分页控制器接收到的导航手势，提供相应的页面视图控制器，显示当前页面。必须要重载的方法：
  - `pageViewController (UIPageViewController, viewControllerBefore: UIViewController)`：该方法返回当前显示的视图控制器前面的一个视图控制器，用于向前翻页。
  - `pageViewController (UIPageViewController, viewControllerAfter: UIViewController)`：该方法返回当前显示的视图控制器后面的一个视图控制器，用于向后翻页。
- 分页控制器委托协议：负责接收用户导航到新页面的通知。常用的方法：
  - `pageViewController (UIPageViewController, didFinishAnimating: Bool, previousViewControllers: [UIViewController], transitionCompleted: Bool)`：该方法在翻页动作完成后被调用。
  - `pageViewController (UIPageViewController, spineLocationFor: UIInterfaceOrientation)`：该方法返回页脊的位置。



# 实例：古文浏览

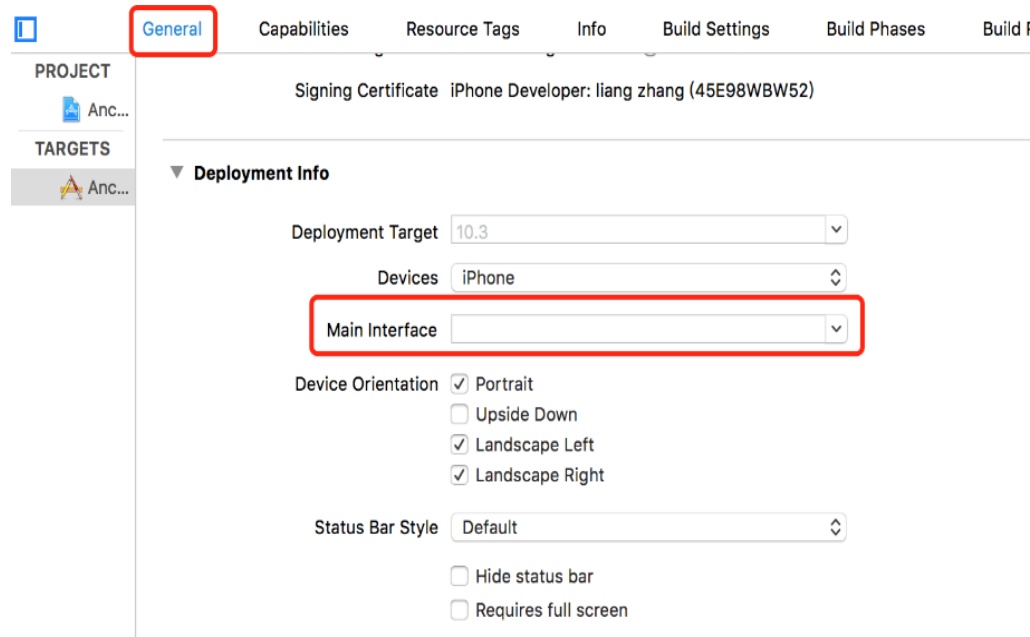
---

- 要求：
  - 通过UIPageViewController来实现古文《岳阳楼记》的五个段落分为5页来显示。

代码参见iosPrj的Chapter06-  
AncientArticleShow

# 删除缺省的故事板和视图控制器

- 通过模板来创建工程，会缺省建立一个故事板文件和一个视图控制器文件。本例中删掉这两个文件。
- 重新创建一个 `ViewController.swift` 文件，用来编写 `UIPageViewController` 的子类。
- 由于模板将缺省创建的故事板作为项目的主界面，删掉该文件后需要将项目主界面设置为空，否则系统会找不到该文件而报错。
- 具体操作是：
  - 选中项目文件树中的根节点，
  - 然后在打开的右侧配置页面中，选中“General”页面，
  - 在表单中将选项“Main Interface”设置为空





# 编写视图显示部分代码

---

- 由于没有故事板文件，所以要手动编写视图显示部分的代码。
- 打开文件AppDelegate.swift，添加设置显示界面的代码。
- 为应用创建一个UIWindow，然后设置该window的根视图为新创建的ViewController的实例，最后显示该window。

```
func application(_ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions:
        [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

    self.window = UIWindow(frame: UIScreen.main.bounds)
    self.window?.rootViewController = ViewController()
    self.window?.makeKeyAndVisible()

    return true
}
```

代码参见iosPrj的Chapter06-AncientArticleShow



# 编写视图控制器ViewController类

- 定义该类遵守分页控制器数据源协议和委托协议。
- 在类的声明部分定义相关变量并赋初始值。
  - pageIndex表示当前页序号，初始值为0；
  - pageCounts表示页数，初始值为5；
  - forward表示向前翻页，用整数1表示；
  - back表示向后翻页，用整数2表示；
  - direction表示当前的翻页方向，初始为向后翻页，故初始值为2；
  - pageViewController为分页控制器；
  - viewControllers为视图控制器数组，每一个元素为一个页面的视图控制器。

```
import UIKit

class ViewController: UIViewController, UIPageViewControllerDataSource,
    UIPageViewControllerDelegate {

    var pageIndex = 0

    let pageCounts = 5

    let forward = 1

    let back = 2

    var direction = 2

    var pageViewController: UIPageViewController!

    var viewControllers = [UIViewController]()
}
```

# 重载方法viewDidLoad

- 设置显示的页面视图的尺寸，这里先计算出系统状态栏的高度，把这个高度空出来显示状态栏，避免状态栏与页面视图内容的重叠。另外，还设置了字体的大小。
- 定义字符串数组变量 paragraph，每一个数组元素储存一段文字。另外，定义了UITextView数组类型的变量textViews，每一个文本编辑区对应一段文字。

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    let statusBarHeight = UIApplication.shared.statusBarFrame.height  
  
    let insets = UIEdgeInsets(top: statusBarHeight, left: 0, bottom: 0,  
                              right: 0)  
  
    let theAttributes = [NSFontAttributeName: UIFont.systemFont(ofSize:  
60), NSParagraphStyleAttributeName: NSMutableParagraphStyle()]  
  
    var paragraphs = [String]()  
    paragraphs.append("庆历四年春，滕子京谪守巴陵郡。越明年，政通人和，百废具兴。  
        乃重修岳阳楼，增其旧制，刻唐贤今人诗赋于其上。属予作文以记之。")  
    paragraphs.append("予观夫巴陵胜状，在洞庭一湖。衔远山，吞长江，浩浩汤汤，横无  
        际涯；朝晖夕阴，气象万千。此则岳阳楼之大观也，前人之述备矣。然则北通巫峡，南极  
        潇湘，迁客骚人，多会于此，览物之情，得无异乎？")  
    paragraphs.append("若夫淫雨霏霏，连月不开，阴风怒号，浊浪排空；日星隐曜，山岳  
        潜形；商旅不行，樯倾楫摧；薄暮冥冥，虎啸猿啼。登斯楼也，则有去国怀乡，忧谗畏  
        讥，满目萧然，感极而悲者矣。")  
    paragraphs.append("至若春和景明，波澜不惊，上下天光，一碧万顷；沙鸥翔集，锦鳞  
        游泳；岸芷汀兰，郁郁青青。而或长烟一空，皓月千里，浮光跃金，静影沉璧，渔歌互  
        答，此乐何极！登斯楼也，则有心旷神怡，宠辱偕忘，把酒临风，其喜洋洋者矣。")  
    paragraphs.append("嗟夫！予尝求古仁人之心，或异二者之为，何哉？不以物喜，不以  
        己悲；居庙堂之高则忧其民；处江湖之远则忧其君。是进亦忧，退亦忧。然则何时而乐  
        耶？其必曰“先天下之忧而忧，后天下之乐而乐”乎。噫！微斯人，吾谁与归？")  
  
    var textViews = [UITextView]()
```



# 重载方法viewDidLoad

- 创建UITextView和UIViewController的实例，并通过循环的方式，赋值给变量textViews和viewController。
- 通过循环的方式，将textViews中的每一个元素依次添加到viewController中的每一个视图控制器中。
- 创建分页控制器实例，并将其委托对象和数据源对象设置为viewController。然后，将viewController中的第一个元素设置为分页控制器的第一页。最后，将分页控制器添加到viewController的视图中。

```
for _ in 0...pageCounts-1 {  
    let textView = UITextView(frame: self.view.frame)  
    textView.contentInset = insets  
    textViews.append(textView)  
    let viewController = UIViewController()  
    viewControllers.append(viewController)  
}
```

```
for i in 0...pageCounts-1 {  
    let theTextView = textViews[i]  
    theTextView.attributedText = NSAttributedString(string:  
        paragraphs[i], attributes: theAttributes)  
    self.viewControllers[i].view.addSubview(theTextView)  
}
```

```
self.pageViewController = UIPageViewController(transitionStyle: .  
    pageCurl, navigationOrientation: .horizontal, options: nil)  
  
self.pageViewController.delegate = self  
  
self.pageViewController.dataSource = self  
  
self.pageViewController.setViewControllers([viewControllers[0]],  
    direction: .forward, animated: true, completion: nil)  
  
self.view.addSubview(self.pageViewController.view)
```





# 重载数据源协议中的方法

- 分页控制器向前翻页的处理方法。
- 该方法中，要对当前页码减一，设置翻页方向为向前翻页。最后要返回目标页的视图控制器。

```
func pageViewController(_ pageViewController: UIPageViewController,
viewControllerBefore viewController: UIViewController) ->
UIViewController? {

    pageIndex -= 1

    if (pageIndex < 0){
        pageIndex = 0
        return nil
    }

    direction = forward

    return self.viewControllers[pageIndex]
}
```



# 重载数据源协议中的方法

---

- 分页控制器向后翻页的处理方法。该方法中，要对当前页码加一，设置翻页方向为向后翻页。最后要返回目标页的视图控制器。

```
func pageViewController(_ pageViewController: UIPageViewController,  
viewControllerAfter viewController: UIViewController) ->  
UIViewController? {  
  
    pageIndex += 1  
  
    if (pageIndex > pageCounts-1){  
        pageIndex = pageCounts-1  
        return nil  
    }  
  
    direction = back  
  
    return self.viewControllers[pageIndex]  
}
```



# 重载委托协议中的两个方法

- 设置页脊的位置在最左侧，同时禁用双页显示。

```
func pageViewController(_ pageViewController: UIPageViewController,
    spineLocationFor orientation: UIInterfaceOrientation) ->
    UIPageViewControllerSpineLocation {

    self.pageViewController.isDoubleSided = false

    return .min
}
```

- 当翻页动作完成后，根据翻页的方向，调整当前页码的值。

```
func pageViewController(_ pageViewController: UIPageViewController,
    didFinishAnimating finished: Bool, previousViewControllers:
    [UIViewController], transitionCompleted completed: Bool) {

    if (completed == false) {

        if (direction == back) {
            pageIndex -= 1
        }

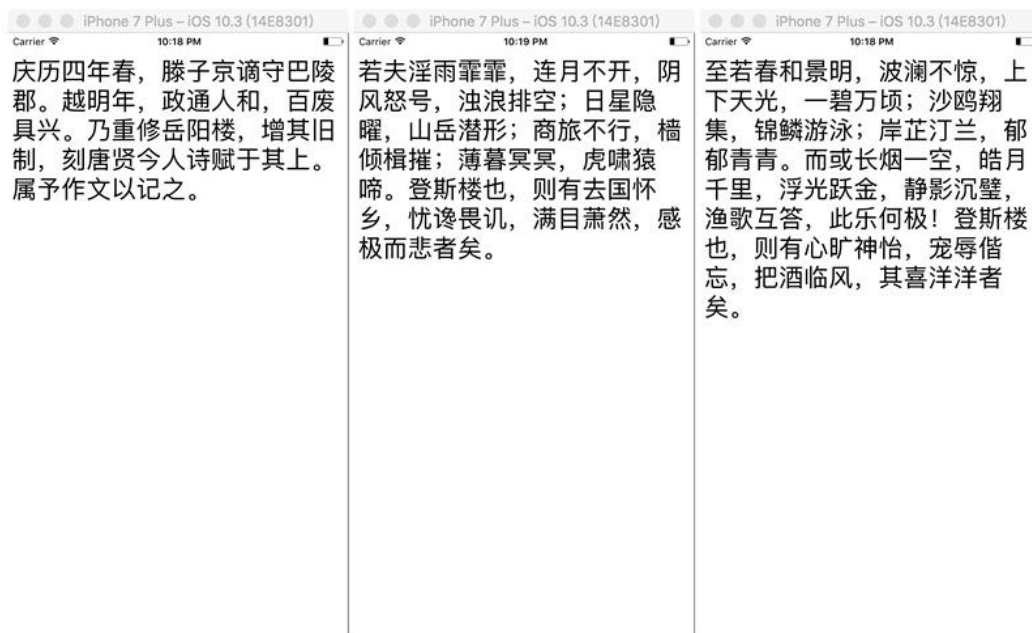
        if (direction == forward) {
            pageIndex += 1
        }

    }
}
```

代码参见iosPrj的Chapter06-AncientArticleShow



# 运行效果





## 第七部分 导航

---

本章目的在于使学生掌握介绍几种导航控制器，包括：标签栏控制器、分页控制器、导航控制器等，并结合上一讲的表格视图，用导航控制器来构建相对复杂的树状导航。

- 标签栏导航
- 分页控制器
- 导航控制器
- 树状导航

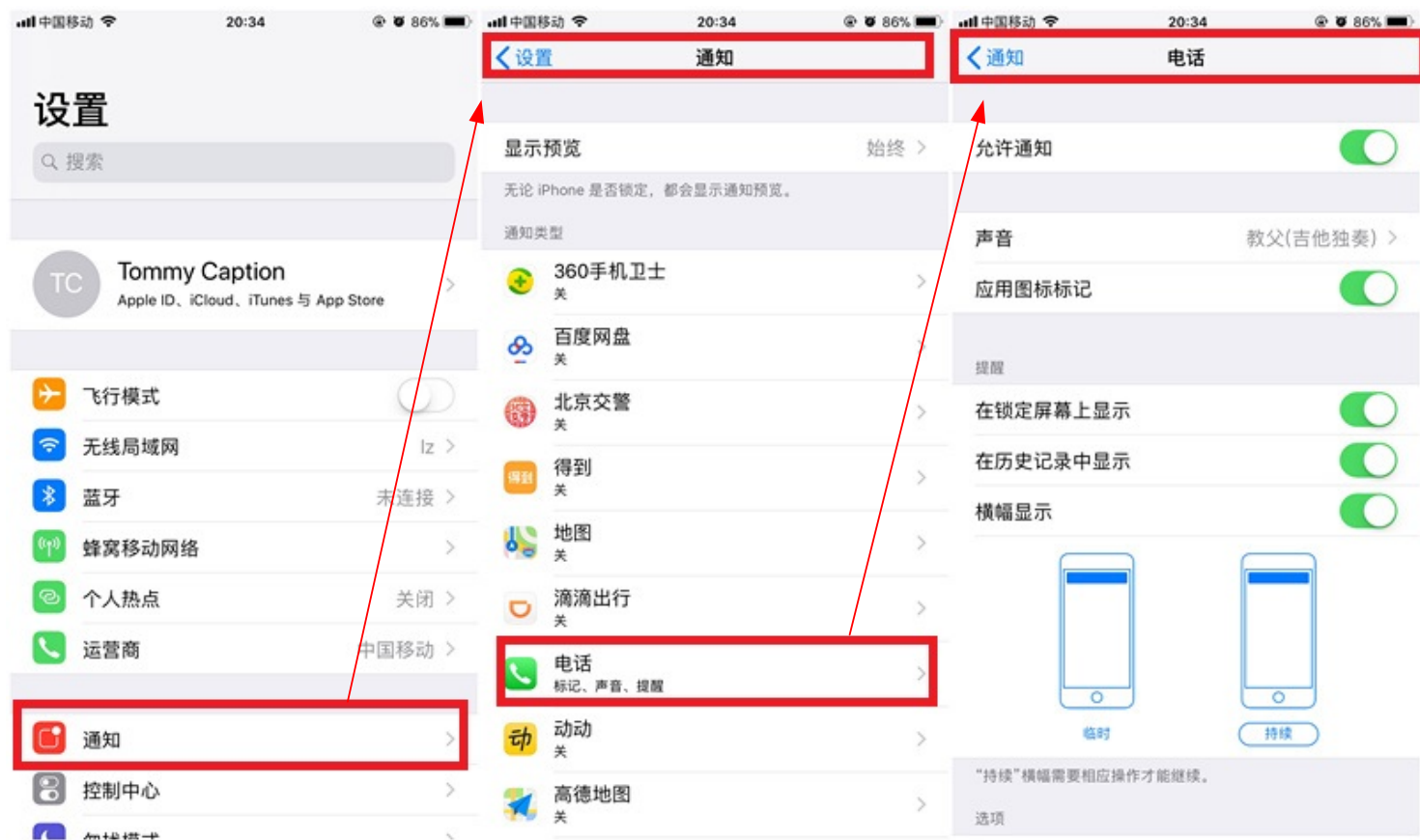


# 导航控制器

---

- UINavigationController 导航控制器是一种容器型的视图控制器，它通过导航界面来管理一个或多个子视图控制器，同一时间只能有一个子视图控制器是可见的。
- 它基于堆栈原理来实现分层内容的导航。
  - 当通过导航控制器界面选择了一个视图控制器后，该视图控制器将会从视图堆栈中被弹出栈并显示出来，同时隐藏原来显示的视图控制器。
  - 导航控制器中，一般都会在顶部有一个导航栏，用来操作视图的导航。

# 典型实例





# 导航视图控制的实现

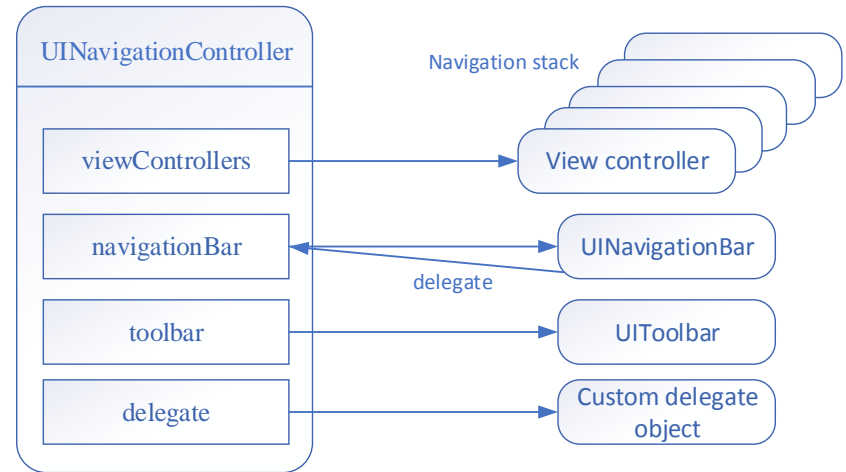
---

- 导航视图控制是通过堆栈原理来对它的子视图进行管理的，视图的存储通过子视图控制器数组来实现的。
- 该数组中的第一个元素就是根节点视图控制器，对应视图控制器堆栈中的底部。数组中的最后一个元素就是当前正在显示的视图控制器，对应视图控制器堆栈中的顶部。
- 在编码时，一般通过视图跳转“**segue**”来向视图控制器堆栈中添加或者删除视图控制器。
- 对应到用户的操作：点击导航栏的“返回”按钮是当前视图弹出堆栈，点击列表中条目来打开下一级页面则是将即将显示的视图压入堆栈。



# 导航控制器的对象关系图

- 导航控制器  
UINavigationController通过属性viewControllers、navigationBar、toolbar、delegate来管理相关的对象  
View Controller数组、导航栏UINavigationController、工具栏UIToolbar、自定义的委托对象delegate object。





# 通知中心

---

- 通知中心是一种通知分发机制，它将通知信息广播给注册的观察器。
- 一个对象要接收到通知就需要向通知中心进行注册观察器，可以通过方法 `addObserver` 来实现。
- 为了接收多个不同的通知，一个对象需要向通知中心注册多个观察器，因为每个观察器都只能观察特定的通知。
- 每个应用都有一个缺省的通知中心。
- 常用方法：
  - `func addObserver(Any, selector: Selector, name: NSNotification.Name?, object: Any?)`  
：它负责通知中心注册一个观察器，需要提供处理函数、通知名及发送对象。
  - `func removeObserver(Any, name: NSNotification.Name?, object: Any?)`  
：它负责将一个观察器从通知中心里移除，需要提供通知名。
  - `func post(name: NSNotification.Name, object: Any?, userInfo: [AnyHashable : Any]? = nil)`  
：它负责创建一条通知，需要提供通知名，发送对象以及传递的信息。



# 实例：教务系统登录注册

---

- 要求：

- 设计一个教务系统登录注册的原型，包含三个页面，初始页面为登录界面。在登录页面上，学生可以输入学号和密码，点击登录将跳转到教务系统的业务页面。如果学生是第一次使用，还可以点击注册，进入到注册页面，完成注册后再次回到初始页面。



# 创建视图控制器文件

- 删掉模板缺省创建 ViewController.swift文件
- 创建登录视图的控制器
- 创建注册视图的控制器
- 创建教务系统业务视图的控制器

```
import UIKit

class EducationalSystemViewController: UIViewController {

    override func viewDidLoad() {

        super.viewDidLoad()

    }

    override func didReceiveMemoryWarning() {

        super.didReceiveMemoryWarning()

    }

}
```

```
import UIKit

class LoginViewController: UIViewController {

    override func viewDidLoad() {

        super.viewDidLoad()

    }

    override func didReceiveMemoryWarning() {

        super.didReceiveMemoryWarning()

    }

}
```

```
import UIKit

class RegisterViewController: UIViewController {

    override func viewDidLoad() {

        super.viewDidLoad()

    }

    override func didReceiveMemoryWarning() {

        super.didReceiveMemoryWarning()

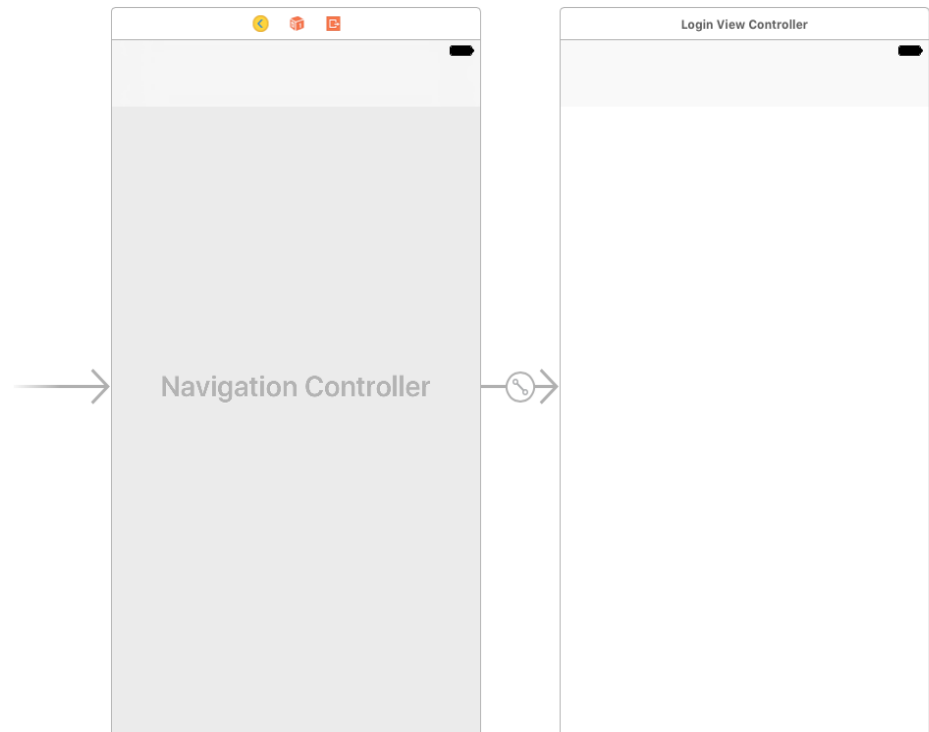
    }

}
```

代码参见iosPrj的Chapter06-LoginRegister

# 将视图控制器嵌入到导航控制器

- 点击菜单栏的editor->embed in->Navigation Controller
- 设置View Controller的custom class为LoginViewController
- 继续将注册视图控制器、教务系统视图控制器嵌入到导航控制器中。



# 设计用户界面



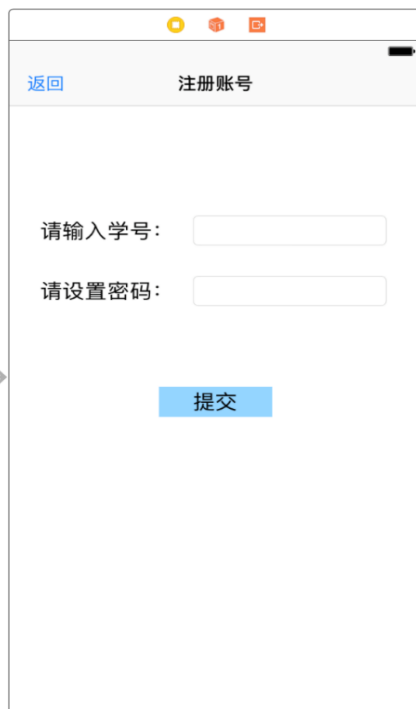
登录教务系统

学号

密码

[登录](#)

第一次使用请先注册账号



[返回](#) 注册账号

请输入学号:

请设置密码:

[提交](#)



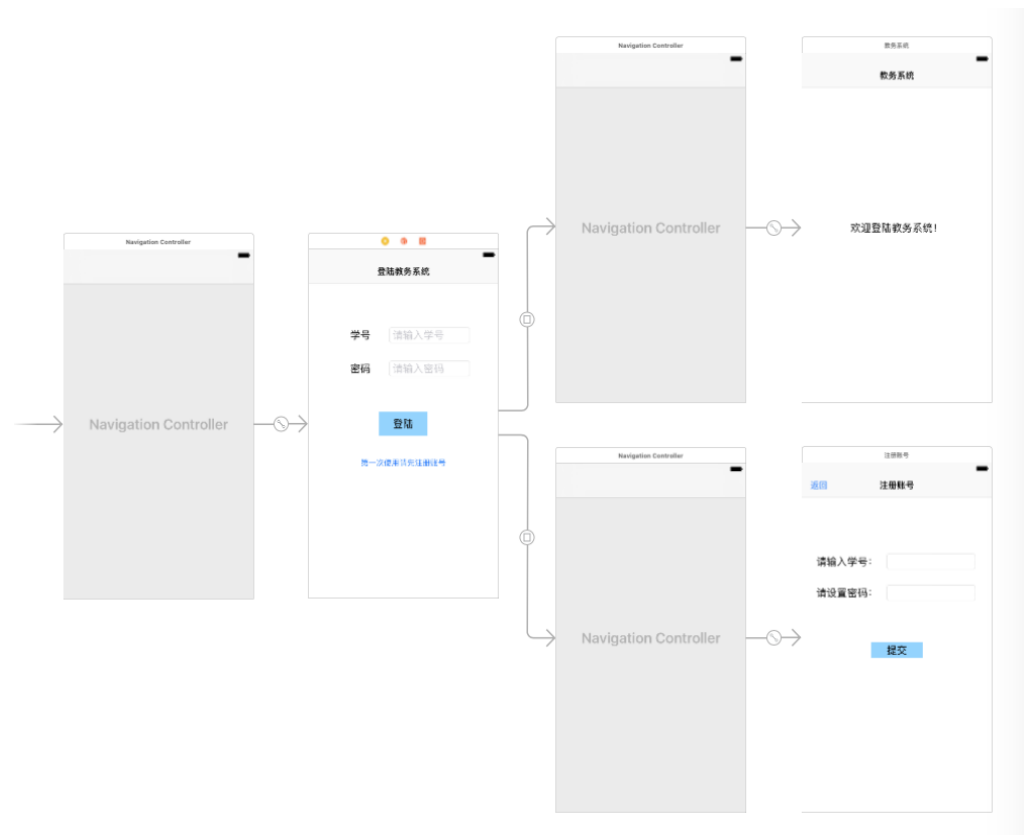
教务系统

教务系统

欢迎登陆教务系统!

# 设置页面segue跳转

- 添加登录、注册及教务系统这三个页面之间的segue跳转：
  - 选中LoginViewController，继续选中按钮“登录”，按住ctrl和鼠标左键，移动鼠标拖出一条线到教务系统视图控制器的Navigation Controller，在弹出的对话框中选择“present modally”。
  - 选中按钮“第一次使用请先注册账号”，拖出一条线到注册视图控制器的Navigation Controller。





# 连接控件与视图控制器

- 为LoginViewController的编辑框建立连接
- 为RegisterViewController的两个编辑框分别建立连接，同时为按钮“提交”和“返回”建立动作处理函数

```
import UIKit

class LoginViewController: UIViewController {

    @IBOutlet var studentID: UITextField!

    @IBOutlet var password: UITextField!
```

```
import UIKit

class RegisterViewController: UIViewController {

    @IBOutlet var studentIDTextField: UITextField!

    @IBOutlet var passwordTextField: UITextField!

    @IBAction func submit(_ sender: Any) {

    }

    @IBAction func back(_ sender: Any) {

    }
```

代码参见iosPrj的Chapter06-LoginRegister





# 向通知中心添加观察器

---

- 打开文件  
loginViewController.swift  
，重载viewDidLoad方法，  
当视图载入完成之前调用该方法。
- 在方法中，向通知中心添加  
一个观察器，监测名为  
“RegisterSubmitNotification”的通知，并为其指定处  
理函数为  
autoFillInformation。

```
override func viewDidLoad() {  
  
    super.viewDidLoad()  
  
    NotificationCenter.default.addObserver(self, selector:  
        #selector(autoFillInformation(_:)), name: Notification.  
        Name(rawValue: "RegisterSubmitNotification"), object:  
        nil)  
}
```



# 实现通知的处理函数

- 添加通知  
“RegisterSubmitNotification”的处理函数  
autoFillInformation。
- 该函数在接收到通知后，将负责通知中传过来的学生信息（学号和密码）添加到对应的编辑框中。
- 重载方法  
didReceiveMemoryWarning  
，在函数中添加一行代码，从通知中心移除观察器

```
func autoFillInformation(_ notification: Notification) {  
    let studentInfo = notification.userInfo!  
    let theStudentID = studentInfo["studentID"] as! String  
    let thePassword = studentInfo["password"] as! String  
    studentID.text = theStudentID  
    password.text = thePassword  
}
```

```
override func didReceiveMemoryWarning() {  
    super.didReceiveMemoryWarning()  
    NotificationCenter.default.removeObserver(self)  
}
```

代码参见iosPrj的Chapter06-LoginRegister



# 按钮的动作处理函数

---

- 在文件 registerViewController.swift 中，在函数 back 中添加一行代码来消除当前视图。
- 在按钮“提交”的动作处理函数 submit 中添加代码

```
@IBAction func back(_ sender: Any) {  
    self.dismiss(animated: true, completion: nil)  
}
```

```
@IBAction func submit(_ sender: Any) {  
    self.dismiss(animated: true) { () -> Void in  
        let studentInfo = ["studentID" :  
            self.studentIDTextField.text!, "password": self.  
            passwordTextField.text!]  
  
        NotificationCenter.default.post(name: Notification.Name  
            (rawValue: "RegisterSubmitNotification"), object:  
            nil, userInfo: studentInfo)  
    }  
}
```

# 运行效果

The image displays two side-by-side screenshots of an iPhone 7 Plus screen, showing a login and registration interface. The status bar at the top of both screens indicates 'Carrier', signal strength, '12:20 AM' (left) and '12:21 AM' (right), and battery level.

**Left Screenshot: 登录教务系统 (Login to the Academic System)**

- Header: 登录教务系统
- Form Fields:
  - 学号 (Student ID): 请输入学号 (Please enter student ID)
  - 密码 (Password): 请输入密码 (Please enter password)
- Button: 登录 (Login)
- Text: 第一次使用请先注册账号 (First time use, please register an account first)

**Right Screenshot: 注册账号 (Register Account)**

- Header: 注册账号
- Buttons: 返回 (Return)
- Form Fields:
  - 请输入学号: 37060115 (Please enter student ID: 37060115)
  - 请设置密码: ..... (Please set password: .....)
- Button: 提交 (Submit)



## 第七部分 导航

---

本章目的在于使学生掌握介绍几种导航控制器，包括：标签栏控制器、分页控制器、导航控制器等，并结合上一讲的表格视图，用导航控制器来构建相对复杂的树状导航。

- 标签栏导航
- 分页控制器
- 导航控制器
- 树状导航



# 树状导航

---

- 树状导航：使用导航控制器和表格视图控制器共同构造相对复杂的导航结构。
- 苹果官方称其为drill-down导航，即：不断深入。
- 在树状导航中，导航控制器将管理多个视图控制器。



# 实例：学生成绩表5.0

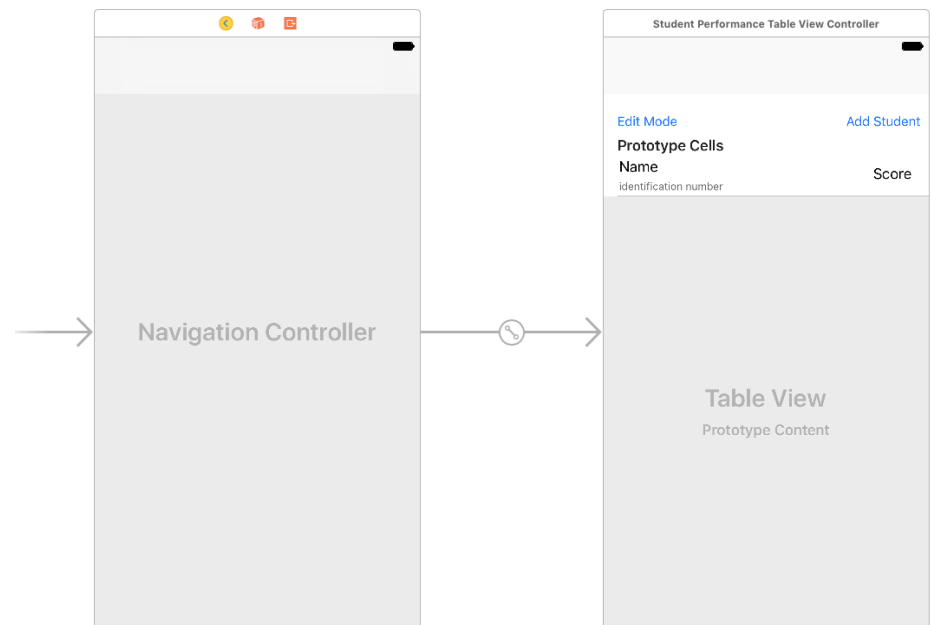
---

- 要求：在前例“学生成绩表4.0”的基础上，通过导航控制器来实现树状导航。
  - 也就是在原来的学生成绩列表的基础上，点击每一行的学生信息时，打开该学生的详情页。在学生的详情页中，可以对信息进行编辑，编辑的结果将反映在学生成绩列表中。

代码参见iosPrj的Chapter06-  
StudentPerformance5.0

# 将表视图嵌入到导航视图控制器中

- 打开学生成绩表4.0项目，选中Main.storyboard中的StudentPerformanceTableViewController
- 依次点击菜单栏的Editor->Embedded In->Navigation Controller。



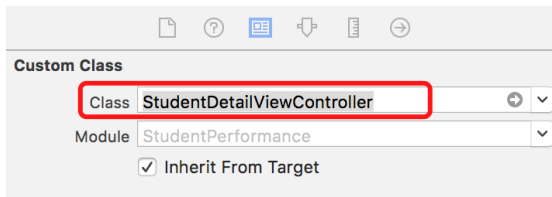


# 创建学生详情页

- 创建一个视图控制器文件 StudentDetailViewController.swift
- 打开故事板文件，向编辑面板中拖入一个 View Controller 控件，然后在其属性面板中设置 Custom Class 的 Class 属性为 StudentDetailViewController
- 向视图控制器 StudentDetailViewController 中添加三个 UILabel 用来提示用户输入相关信息，再添加三个 UITextField 对应学生的三个信息字段。

```
import UIKit

class StudentDetailViewController: UIViewController {
}
```





# 连接控件与视图控制器

---

- 连接视图控制器中的三个UITextField与StudentDetailViewController.swift文件

```
import UIKit

class StudentDetailViewController:
    UIViewController {

    @IBOutlet var nameTextField:
        UITextField!

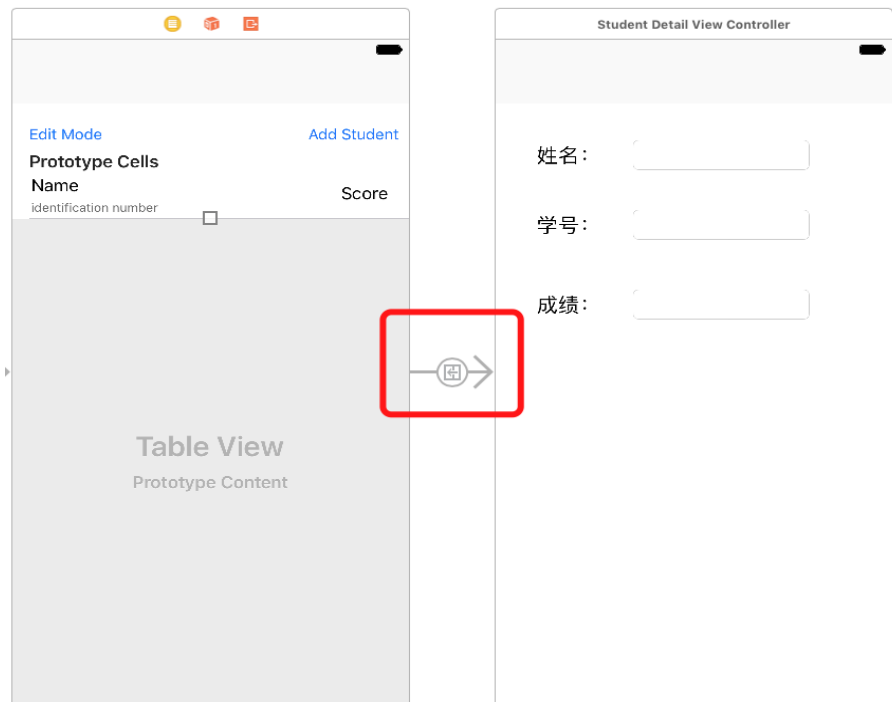
    @IBOutlet var studentIDTextField:
        UITextField!

    @IBOutlet var scoreTextField:
        UITextField!

}
```

# 建立页面间的导航关系

- 通过UIStoryboardSegue来实现，从学生成绩表格视图控制器，到学生详情视图控制器的树状导航关系
  - 选中表格视图中的UITableViewController，按住Ctrl和鼠标左键，拖一条线到学生详情视图控制器上。
  - 在弹出的窗口中的Selection Segue中选中“Show”





# 视图控制器之间的传值

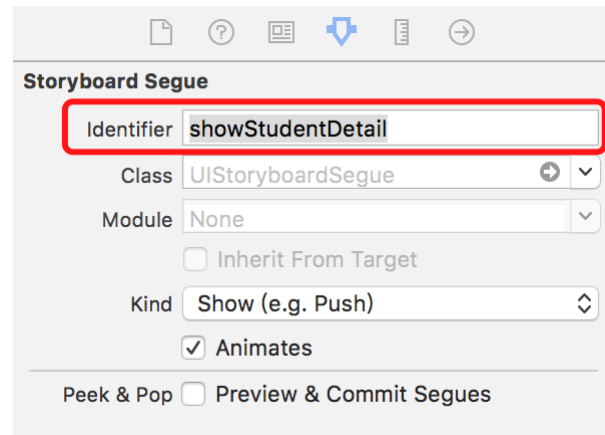
- 使学生详情页中能够显示相应的学生信息，就是要实现两个视图控制器之间的传值问题。
- 在学生详情页的视图控制器中，添加数据接收和显示部分的代码。
  - 在类的声明部分定义一个Student类的变量theStudent
  - 在传值的时候，该变量用来接收传入的值。
  - 重载方法viewWillAppear，在视图即将呈现的时候调用该方法，将变量theStudent中的值赋给相应视图控件中。

```
var theStudent: Student!  
  
override func viewWillAppear(_ animated: Bool) {  
    super.viewWillAppear(animated)  
  
    nameTextField.text = theStudent.name  
  
    studentIDTextField.text = theStudent.id  
  
    scoreTextField.text = "\(theStudent.score)"  
}
```

代码参见iosPrj的Chapter06-StudentPerformance5.0

# 视图控制器之间的传值

- 选中两个视图控制器之间的segue，在右侧打开的属性编辑栏中设置该segue的Identifier
- 这样就可以通过Identifier找到这个segue了





# 视图控制器之间的传值

---

- 在StudentPerformanceTableViewController类中重载segue的方法prepare。
- 该方法在实现segue跳转之前被调用。

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if segue.identifier == "showStudentDetail" {  
        if let row = tableView.indexPathForSelectedRow?.row {  
            let theStudent = studentsInfo.studentsCollection[row]  
  
            let theStudentDetailViewController = segue.destination  
                as! StudentDetailViewController  
  
            theStudentDetailViewController.theStudent = theStudent  
        }  
    }  
}
```

# 传值效果

iPhone 7 Plus – iOS 10.3 (14E8301)

Carrier 6:41 PM

Edit Mode

Tommy	98
37060101	
Jerry	65
37060102	
Kate	78
37060103	
Jack	100
37060104	
Ben	85
37060105	
Jimmy	79
37060106	
Ada	95
37060107	
Susan	80
37060108	

Add Student

iPhone 7 Plus – iOS 10.3 (14E8301)

Carrier 6:41 PM

< Back

姓名: Jack

学号: 37060104

成绩: 100

# 实现学生详情页的修改反映到成绩表中

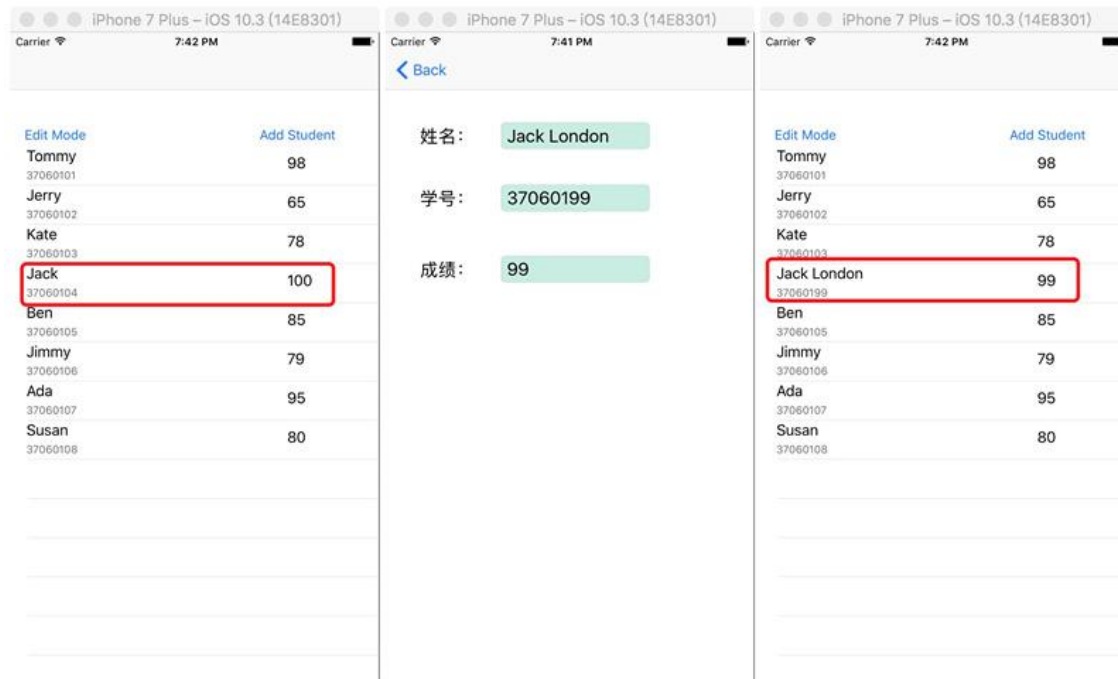
- 在学生详情页视图控制器类中
  - 重载方法 `viewWillDisappear`。该方法在视图控制器即将消失时被调用，在方法中将编辑后的结果赋值给类中的变量 `theStudent`。
  - 重载方法 `viewWillAppear`。该方法在视图控制器即将显示时被调用，在方法中通过 `tableView` 的方法 `reloadData` 刷新表格视图的数据。

```
override func viewWillDisappear(_ animated: Bool) {  
    super.viewWillDisappear(animated)  
  
    theStudent.name = nameTextField.text!  
  
    theStudent.id = studentIDTextField.text!  
  
    theStudent.score = Int(scoreTextField.text!)  
}
```

```
override func viewWillAppear(_ animated: Bool) {  
    super.viewWillAppear(animated)  
  
    tableView.reloadData()  
}
```



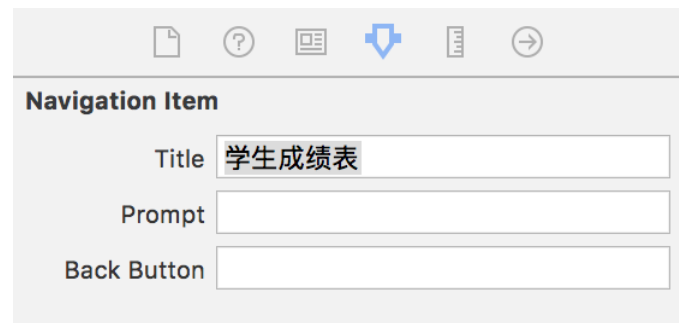
# 运行效果



# 设置导航当前页的标题

- 学生成绩表格视图中的导航栏主题可显示“学生成绩表”，学生详情页视图中的导航栏主题可显示学生姓名。
- 在学生详情视图控制器中为变量theStudent添加一个属性观察器，一旦theStudent被赋值，就将学生姓名字段赋值该视图的导航栏的title。
- 打开故事板文件，选中学生表格视图控制器中的导航栏，在右侧的属性面板中设置Title为“学生成绩表”。

```
var theStudent: Student! {  
    didSet {  
        navigationItem.title = theStudent.name  
    }  
}
```



The screenshot shows the Xcode interface with the 'Navigation Item' property panel on the right. The 'Title' field is set to '学生成绩表' (Student Grade Table). The 'Prompt' and 'Back Button' fields are empty.

Navigation Item	
Title	学生成绩表
Prompt	
Back Button	

The image displays three sequential screenshots of an iPhone 7 Plus screen, illustrating the user interface of a student performance application. The status bar at the top of each screen indicates the device is an iPhone 7 Plus running iOS 10.3 (14E8301) at 9:10 PM.

**Screenshot 1 (Left):** The app is in 'Edit' mode. The title bar shows 'Carrier' and a signal strength indicator. The main title is '学生成绩表' (Student Performance Table). Below the title is a table with two columns: student names and their scores. The students listed are Tommy (98), Jerry (65), Kate (78), Jack (100), Ben (85), Jimmy (79), Ada (95), and Susan (80). A red '+' button is visible in the top right corner of the table area.

**Screenshot 2 (Middle):** The app is in 'Done' mode. The title bar shows 'Carrier' and a signal strength indicator. The main title is '学生成绩表' (Student Performance Table). Below the title is a table with two columns: student names and their scores. The students listed are Tommy (98), Jerry (65), Kate (78), Jack (100), Ben (85), Jimmy (79), Ada (95), and Susan (80). A red '-' button is visible in the top left corner of the table area.

**Screenshot 3 (Right):** The app is in a form view. The title bar shows 'Carrier' and a signal strength indicator. The main title is '学生成绩表' (Student Performance Table). Below the title is a form with three fields: '姓名' (Name) with the value 'Jack', '学号' (Student ID) with the value '37060104', and '成绩' (Score) with the value '100'. A red '+' button is visible in the top right corner of the form area.



谢 谢

---