**WatchApp开发**

第一章：AppleWatch开发入门——界面布局 1

第二章：AppleWatch开发入门——代码交互、控制器生命周期、界面跳转 9

第三章：AppleWatch开发入门——Table视图的应用 13

第四章：AppleWatch开发入门——Picker视图的应用 18

第五章：AppleWatch开发入门——Menu的使用 23

第六章：AppleWatch开发入门——watchOS中通知的应用 27

第七章：AppleWatch开发入门——AlertController 36

第八章：AppleWatch开发入门——动画 39

# 

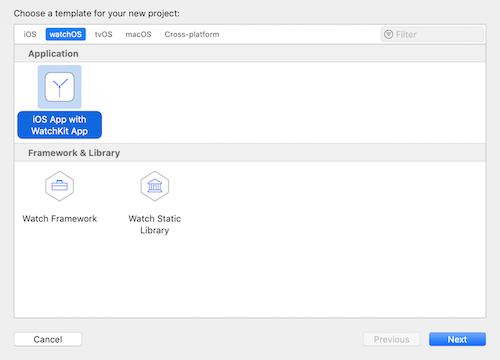
# 第一章：AppleWatch开发入门——界面布局

**1.1简介**

* 在 iPhone 开发中，最基本的布局方式是通过 frame 将控件的位置和大小固定在屏幕上，后来，由于手机屏幕的尺寸有了略微变化，有了 autoresizing 布局框架，可以设置子视图随父视图的改变做一些相应的变化，再后来，iPhone 尺寸与分辨率也越来越多，适配各个屏幕也成为了 iOS 开发者遇到的新的问题，幸运的是，autolayout 机制的出现，大大减少在适配方面的成本。但是在 iWatch 的布局方式中，**需要抛弃 iPhone 中的布局思路**，接受一套不太一样的布局框架。
* 首先，iWatch 的屏幕不大，目前有 38mm、40mm、42mm 和 44mm 四个尺寸，无法在这个有限的空间里做非常复杂的界面效果，因此，在开发中应便于使用、一目了然。iWatch 上的布局方式采用的是一种平面堆放的方式，没有 frame，也没有约束，控件的布局方式只是一个挨着一个的平面堆放，也不可重叠。但在 iWatch 中提供了 Group 这样一种布局方式，可以让我们在布局中体现自由与个性的方面。

**1.2创建项目工程**

创建工程就不做详细描述了，直接下一步就行。



**1.3基础的堆放布局**

打开 WatchKit APP中的 storyboard，这就是整个 iWatch 布局的地方。在 iWatch 开发中，目前只能用 storyboard 来开发。

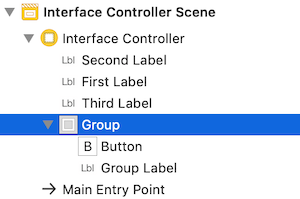
iWatch 的布局采用的是最基础的堆放方式，从上到下依次排开，例如，我们添加三个 Label，效果如下：



通过拖动控件改变顺序，注意，这里仅可改变其上下位置。这种方式的布局高度并没有限制，我们可以一直往下排列，在 iWatch上则会出现上下滑动的效果。

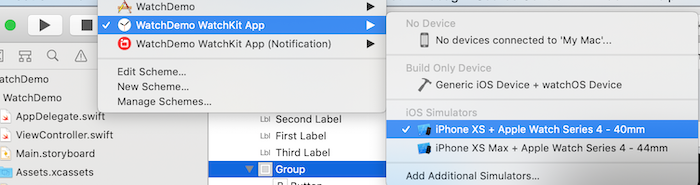
**1.4使用 Group 进行复杂的界面布局**

* 通过上面的布局方式，我们只能进行纵向的排列布局，这并不能达到我们的需求，WatchKit 中提供那一套布局的模型：Group。
* 可以这样理解，Group 默认透明，它的主要作用是提供布局，可以将屏幕分成了几各分区，我们可以设置各个分区的排列方式，例如水平或者垂直，通过这样的思路，完成复杂的 iWatch 界面布局。
* 直接拖一个 Group 到界面上，然后将需要水平布局的控件放在 Group 里面：



Group 布局

然后运行程序



**1.5所谓Group**

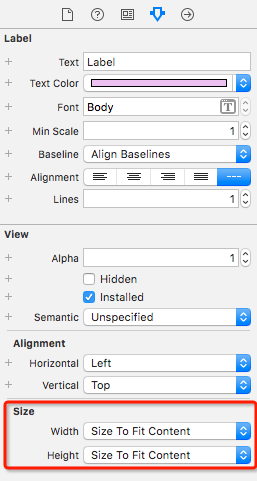
Group在界面布局上，不仅可以起到分区屏幕的作用，其还可以设置一些属性来使布局更加漂亮。在storyBoard右侧的设置菜单中，我们可以对这些属性进行操作：

* Layout：设置布局模式，分为 水平布局 和 垂直布局 两种；
* Insert：可以设置内容区域偏移量，通过这个属性，我们可以使其中填充的控件四周留白；
* Spacing：其中填充的控件的间距；
* Background：设置Group的背景图案；
* Mode：设置背景图案的填充方式；
* Animate：出现时带动画；
* Color：设置Group的背景颜色；
* Radius：设置Group的圆角度。

**1.6布局中控件的位置和尺寸设置**

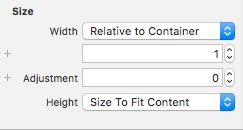
对于控件的尺寸，有三种模式，控件的 Width 和 Height 都是通过这三个模式设置的：  
在 iPhone 中，通过 frame 或者约束来控制控件的位置和尺寸，而在iWatch 中则简单很多，尺寸和位置都是固定的模式，我们只需要做一些设置即可。

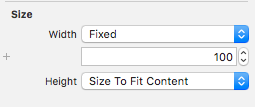
1.6.1控件尺寸的控制  
**Size To Fit Content**：控件尺寸与内容相关，例如，Label 中字数的多少决定了 Label的尺寸。



Size To Fit Content  
**Relative to Container**：控件尺寸按照容器尺寸的比例设置。例如设置为 0.5，则当前控件的尺寸就是容纳其 Group 的一半。

Relative to Container

  
**Fixed**：设置一个固定的值。



Fixed

1. 控件位置的控制  
   因为 iWatch 的界面十分简洁，对于控件的位置设置，是通过水平和垂直两个维度来设置的，通过设置每个维度的属性来控制其在容纳它的父控件中的位置：

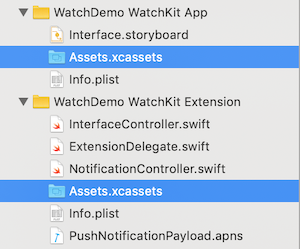


位置控制

**1.7图片设置**

图片的使用有大坑。

关于图片素材，你可以发现，在 Extension 和 App 文件夹中各有一个 Assets.xcasssets 组，只有将素材放入这里面，iWatch才能使用。

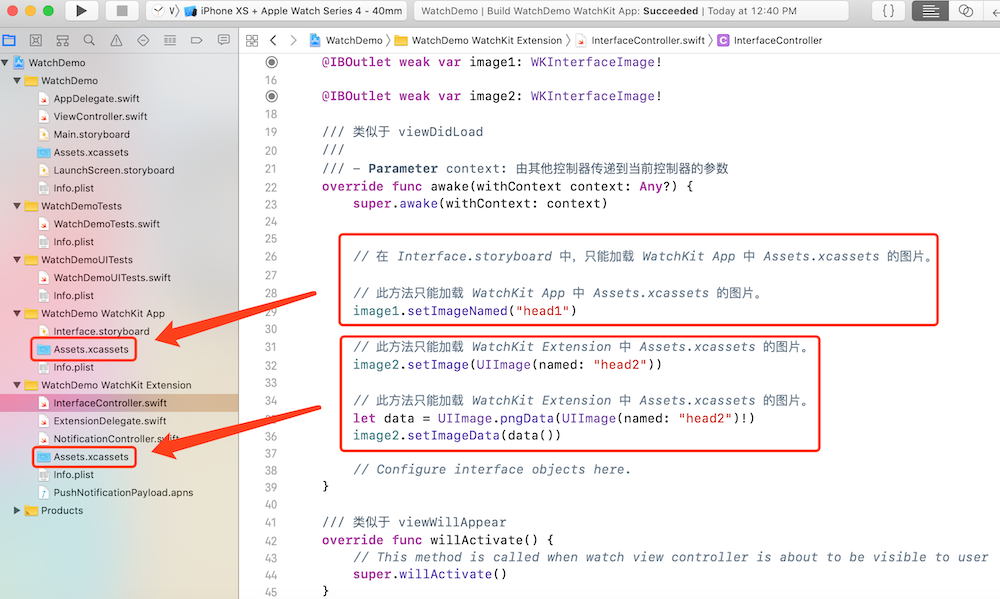


Assets.xcasssets

加载图片的方法有一下几种，这几种方法也有大坑。

@available(watchOS 2.0, \*) open class WKInterfaceImage : WKInterfaceObject, WKImageAnimatable { open func setImage(\_ image: UIImage?) open func setImageData(\_ imageData: Data?) open func setImageNamed(\_ imageName: String?) open func setTintColor(\_ tintColor: UIColor?) }

图片使用一定要特别注意：  
**1. 在 Interface.storyboard 中，只能加载 WatchKit App 中 Assets.xcassets 的图片。**  
**2. func setImageNamed(\_ imageName: String?)此方法只能加载 WatchKit App 中 Assets.xcassets 的图片。**  
**3. func setImage(\_ image: UIImage?)此方法只能加载 WatchKit Extension 中 Assets.xcassets 的图片。**  
**4. func setImageData(\_ imageData: Data?)此方法只能加载 WatchKit Extension 中 Assets.xcassets 的图片。**



# 第二章：AppleWatch开发入门（2）——代码交互、控制器生命周期、界面跳转

**一、引言**

在前面，讨论了关于 iWatch 开发中框架与界面布局相关，主要的逻辑，终究还是要通过代码来实现的，在我们创建了项目之后，就会生成 InterfaceController 这个文件，它就是我们 storyboard 中的入口视图控制器。此外，在 iWatch 开发中，目前只能用 storyboard 来开发。

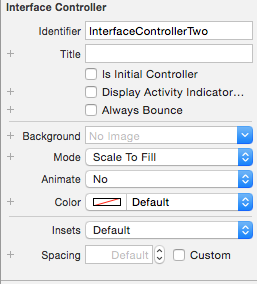
**二、代码交互与控制器声明周期**

* storyboard 中的控件我们可以通过拖拽的方式关联到文件中，Action 和 Outlet 两种关联方式基本可以达到我们修改控件和处理业务逻辑的需求。
* WKInterfaceController 类似于iOS中的 ViewController，是 iWatch 中主要用于展示界面的 Controller，我们的控件也都是基于这个容器中显示。在模板中，系统为我们提供了三个函数，这三个函数体现了watch一个界面的声明周期，如下：
* // 初始化界面时触发，通过context可以实现界面的传值 override func awake(withContext context: Any?) { super.awake(withContext: context) } // 界面即将展现时触发 类似于iOS中的ViewWillApear override func willActivate() { // This method is called when watch view controller is about to be visible to user super.willActivate() } // 界面消失后触发，类似于iOS中的ViewDidDisAppear override func didDeactivate() { // This method is called when watch view controller is no longer visible super.didDeactivate() }

**三、界面跳转与传值**

* 与iOS类似，watchOS的界面跳转也有两种方式：modal 和 push。同样，我们也可以通过 storyboard 或者代码来进行跳转。  
  1、通过**代码**跳转与传值  
  我们创建两个InterfaceController和与之对应的类，界面如下：

  
  
通过代码跳转，**一定**要给第二个控制器设置一个id标识符：



在按钮触发的方法中，如下跳转：

// modal @IBAction func presentMyController() { // 这里的context是传值的上下文，在被跳转控制器的awakeWithContext方法中会将这个值取到 // InterfaceControllerTwo 是在上文提到的 Identifier presentController(withName: "InterfaceControllerTwo", context: "我是传的值") } // push @IBAction func pushMyController() { pushController(withName: "InterfaceControllerTwo", context: nil) }

2、在 storyboard 中设置跳转关系  
也可以直接在 storyboard 中设置界面的跳转，右键拖拽触发跳转事件的控件，拖拽到将要跳转的控制器，会出现 push 和 model 菜单，如下图。



通过这种方式进行的跳转，在执行跳转之前，会执行如下这个函数：

// 这个方法的返回值就是 context 上下文传递的值。 override func contextForSegue(withIdentifier segueIdentifier: String) -> Any? { return "我是值" }

# 第三章：AppleWatch开发入门（3）——Table视图的应用

**一、简介**

iWatch 中的 Table 和 iPhone 中的 UITableView 还是有很大的区别，在开发之前，我们应该明白 WatchOS 中 Table 的局限性和特点。

1. Table只有行的概念，没有分区的概念，没有头尾视图的概念。
2. 可以通过创建多个Table，来实现分区的效果。
3. 因为 iWatch 上是通过 Gruop 进行布局适应的，所以没有行高等设置。
4. Table 没有代理，所有行的数据都是采用静态配置的方式。
5. 点击 Table 中的行触发的方法，是通过重写 Interface 中的方法来实现的。

**二、创建一个Table**

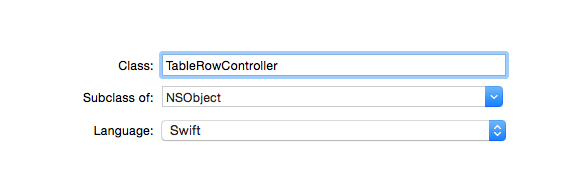
在 storyboard 中拖入 Table，如下：

# 

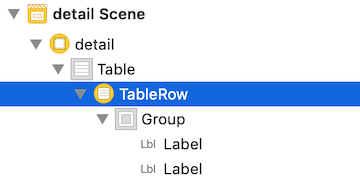
在 Table 上放两个 Label：

# 

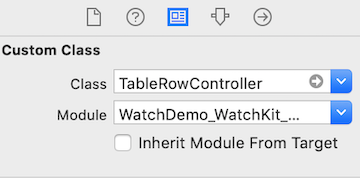
每个 Table 中包含一个 TableRowController，实际上我们 Table 上的控件都是通过 TableRowController 进行管理的，因此如果我们需要在代码中控制 TableRow 上的内容，我们需要新建一个类作为 Table 的 TableRowController，继承自 NSObject:

~~~~

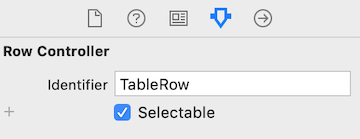
将 storyboard 中的 TableRowController 修改为我们创建的类，并指定 Identifier：



选中 TableRowController 时，注意不要选中 Group，同时 Group 也可以调整控件的布局方式。



修改为自己新建的类



指定 Identifier

然后，将两个 Label 关联到 TableRowController 中：

import WatchKit class TableRowController: NSObject { @IBOutlet var titleLabel: WKInterfaceLabel! @IBOutlet var numberLabel: WKInterfaceLabel! }

将 Table 关联到 InterfaceController 中：

class InterfaceControllerMain: WKInterfaceController { @IBOutlet var table: WKInterfaceTable! }

下面，我们开始在interface中对Table做相关配置，WatchOS的Table配置非常简单易用，例如我们如下配置：

override func awake(withContext context: Any?) { super.awake(withContext: context) let dict = [["name" : "中国建设银行", "money" : "￥1000"], ["name" : "中国农业银行", "money" : "￥5000"], ["name" : "中国工商银行", "money" : "￥2000"], ["name" : "中国招商银行", "money" : "￥4010"], ["name" : "中国邮政储蓄", "money" : "￥1100"]] table.setNumberOfRows(dict.count, withRowType: "TableRow") for (idx, info) in dict.enumerated() { let cell = table.rowController(at: idx) as! TableRowController cell.titleLabel.setText(info["name"]) cell.numberLabel.setText(info["money"]) } }

这样一个展示银行卡余额的界面我们就创建完成了，效果如下：



**三、关于Table的点击事件**

上面我们提到，Table 没有所谓代理方法，点击 cell 的时候，我们也是通过两种方式进行逻辑跳转的。

* 一种是在 storyboard 中，通过拖拽 TableRowController 拉线跳转，这时如需传值，我们需在 Interface 中重写如下方法：

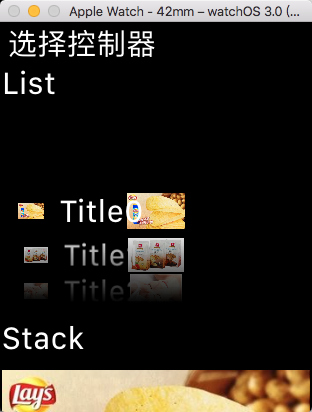
open func contextForSegue(withIdentifier segueIdentifier: String, in table: WKInterfaceTable, rowIndex: Int) -> Any?

* 另一种方式，重写如下方法，来处理Table的点击事件：
* open func table(\_ table: WKInterfaceTable, didSelectRowAt rowIndex: Int)
* 无论哪种方式，我们都可以通过参数 table 和 rowIndex 来确认点击的具体是那个table和哪一行，进行传值和处理我们的逻辑。

# 第四章：AppleWatch开发入门（4）——Picker视图的应用

# 一、简介

Picker到底是个什么东西？如下图所示：



# 

Picker有3种模式：List、Stack、Sequence：

* List：有点像tableview，一行一行展示
* Stack：有点像图片浏览器，一页一页展示，翻动时有动画效果
* Sequence：与Stack类似，只不过翻动时没有动画效果

# 二、创建Picker

在 storyboard 中拖入 Picker：

# 

因为Picker有3种模式（List、Stack、Sequence），我们每种模式弄一个：

# 

**三、代码部分**

* 工欲善其事，必先利其器，呃，必先看头文件：
* @available(watchOS 2.0, \*) public class WKInterfacePicker : WKInterfaceObject { // 设置聚焦，有点像becomeFirstResponder public func focus() // 失去聚焦，有点像resignFirstResponder public func resignFocus() // 改变选中的索引 public func setSelectedItemIndex(\_ itemIndex: Int) // 设置Picker中的内容 public func setItems(\_ items: [WKPickerItem]?) public func setCoordinatedAnimations(\_ coordinatedAnimations: [WKInterfaceObject]?) public func setEnabled(\_ enabled: Bool) } @available(watchOS 2.0, \*) public class WKPickerItem : NSObject, NSSecureCoding { // 设置文字，仅当Picker是'List'时有效. public var title: String? // Caption to show for the item when focus style includes a caption callout. public var caption: String? // 设置title前面的小图片，仅当Picker是'List'时有效，图片大小固定是13×13pt。 @NSCopying public var accessoryImage: WKImage? // 大图 @NSCopying public var contentImage: WKImage? }
* 具体实现  
  1、 Pick 中的数据源是 WKPickerItem，先懒加载[WKPickerItem]
* lazy var items: [WKPickerItem] = { var its = [WKPickerItem]() for i in 0...4 { let item = WKPickerItem() item.title = "Title" + "\(i)" item.caption = "Caption" + "\(i)" item.accessoryImage = WKImage(image: UIImage(named: "ad\_0" + "\(i)")!) item.contentImage = WKImage(image: UIImage(named: "ad\_0" + "\(i)")!) its.append(item) } return its }()
* 这里图片资源大家根据实际需求添加，加载图片时要特别注意，否则可能加载不出，详细见：[AppleWatch开发（1）——界面布局](https://www.jianshu.com/p/7c223446040c" \t "_blank) 这篇文章的末尾部分。
* 2、 连线
* @IBOutlet var listPicker: WKInterfacePicker! @IBOutlet var stackPicker: WKInterfacePicker! @IBOutlet var sequencePicker: WKInterfacePicker!
* 3、 设置数据
* override func awake(withContext context: Any?) { super.awake(withContext: context) listPicker.setItems(items) stackPicker.setItems(items) sequencePicker.setItems(items) sequencePicker.focus() }
* 这里要注意，在有多个Picker时，需要设置focus()，不然系统不知道你要滚动哪一个。

# 第五章：AppleWatch开发入门——Menu的使用

**一、简介**

菜单也是WatchOS中一个重要的交互方式，限于Watch的屏幕尺寸，若将所有用户交互控件都紧密的排列进展示的UI中，那样难免会使用户操作困难，也会影响界面布局的简洁美观。因此，WatchOS的菜单机制是一层覆盖在屏幕上的交互界面，有如下的特点：  
1、菜单是内置于InterfaceController中的，不需显式处理，只需对齐菜单项进行添加设置。  
2、菜单最多可以容乃**4个**选项按钮。  
3、通过**重按**可以呼出和隐藏菜单。

**二、创建菜单的方式**

1、通过storyBoard创建  
在storyBoard中，我们可以将一个菜单控件拖入到interfaceController中：

作者：小道萧兮  
链接：https://www.jianshu.com/p/d03419e28496  
来源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

# 

在Menu中可以添加一些item，每个item都可以设置图片和文字：

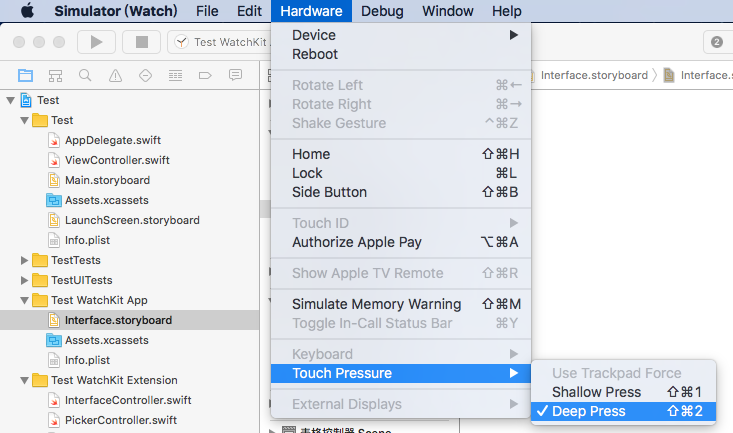
图片的设置分为，自定义和系统两种，我们可以使用自己的图片作为菜单的图片，也可以使用系统为我们提供的一些图片，系统的图片参数是一个枚举：

# 

这些枚举中提供了一些我们常用的功能图标。  
菜单按钮的触发方法，我们可以通过拖拽Action的方式来添加，在Xcode7的模拟器中，我们使用command+shift+2可以切换到重按模式，模拟器效果如下：

# 

**注意**  
想要换出这个界面需要重按，本人找了好多资料才知道。如何重按？如下图：



需要先选中这个选项，然后再点击。如果不需要重按，比如已经切换到别的界面，记得选中Shallow Press，不然你的界面点击都没效果了。

# 第六章：AppleWatch开发入门——watchOS中通知的应用

**一、引言**

在 iOS 系统中，支持的通知有两种类型：本地通知和远程通知。本地通知多用于计时类通知，远程的又称推送，多用于一些提示动态的提示信息。这里有相关通知的一些知识总结：

而 iWatch 中的通知是和 iPhone 同步的，在 iPhone 上的 App 收到通知的同时，会默认也推送到 iWatch 上，基于 iWatch的穿戴性，对用户来说，它上面的通知信息将比 iPhone 更加及时。

**二、WatchOS通知概览**

首先，iWatch 上的通知分为两部分：short-look 和 long-lock。

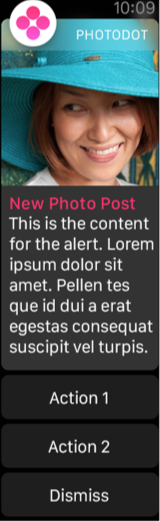
short-look 可以理解为一个简单的通知预览，它会将通知发起的APP和主要标题等信息展示给你，让你一目了然，当用户抬起手腕，查看这个通知一定时间，这个短通知就会转换为 long-look 通知。short-look 的通知界面不能够自定义，系统已经设计好了模样，如下：



short-look

long-look 的界面我们是可以进行一定程度上的自定义的，并且可以添加按钮等逻辑操作。

long-look 也分为两种界面，静态界面和动态界面。这个也好理解，静态界面是我们在写程序时就定义好的界面，在通知发送到watch上时，界面会自动匹配通知内容进行显示。动态的界面则是当收到通知时，会先执行我们相应的配置代码，之后在进行通知界面的展示。一个 long-look界面大致如下：



long lock

在 long-lock 中，界面定义为三个部分，头部标题栏，自定义视图栏和按钮交互区。头部的标题栏我们不能自定义，它是一个半透明的上面有App图标和名字的横栏。其下面是我们可以自定义的区域，我们可以在storyBoard中拉入文本和图片。最下面是一些交互按钮，其名称等配置信息在推送的文件中定义。

**三、在模拟器上测试本地推送**

无论本地推送还是远程推送，需要注意一点，iWatch 上的消息推送是依赖于 iPhone，也就是说只有当 iPhone 收到推送时，iWatch 才能收到通知。所以，应先让 iPhone 能收到本地通知。

为方便测试，直接在 AppDelegate 中发送本地推送。

import UserNotifications @UIApplicationMain class AppDelegate: UIResponder, UIApplicationDelegate { func application(\_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool { // 消息中心 let center = UNUserNotificationCenter.current() center.requestAuthorization(options: [.badge, .alert, .sound]) { (isSuccess, error) in if !isSuccess { print(error) } } // 5秒后触发消息推送 let timeTrigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false) let requestIdentifier = "requestIdentifier" // 消息推送的内容 let content = UNMutableNotificationContent() content.title = "通知来了" content.subtitle = "这里是一个测试用例通知的子标题" content.body = "这里是用于 Apple Watch 通知 Demo 的内容详情。" content.badge = 1 content.sound = UNNotificationSound.default content.categoryIdentifier = requestIdentifier // 添加按钮 let join = UNNotificationAction(identifier: "join", title: "接受邀请", options: .authenticationRequired) let look = UNNotificationAction(identifier: "look", title: "查看邀请", options: .foreground) let cancel = UNNotificationAction(identifier: "cancel", title: "取消", options: .destructive) let input = UNTextInputNotificationAction(identifier: "input", title: "请输入", options: .foreground, textInputButtonTitle: "发送", textInputPlaceholder: "tell me loudly") let notificationCategory = UNNotificationCategory(identifier: requestIdentifier, actions: [join, look, cancel], intentIdentifiers: [], options: .customDismissAction) var set = Set<UNNotificationCategory>() set.insert(notificationCategory) center.setNotificationCategories(set) let request = UNNotificationRequest(identifier: requestIdentifier, content: content, trigger: timeTrigger) center.add(request, withCompletionHandler: nil) return true } }

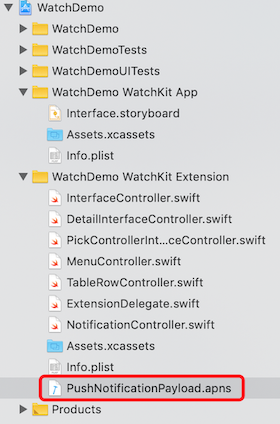
然后运行程序。**注意：只有 iPhone 在锁屏的状态下，并且 iWatch 在 Home 界面时，iWatch 才能收到通知。**



Apple Watch 收到来自 iPhone 的消息推送

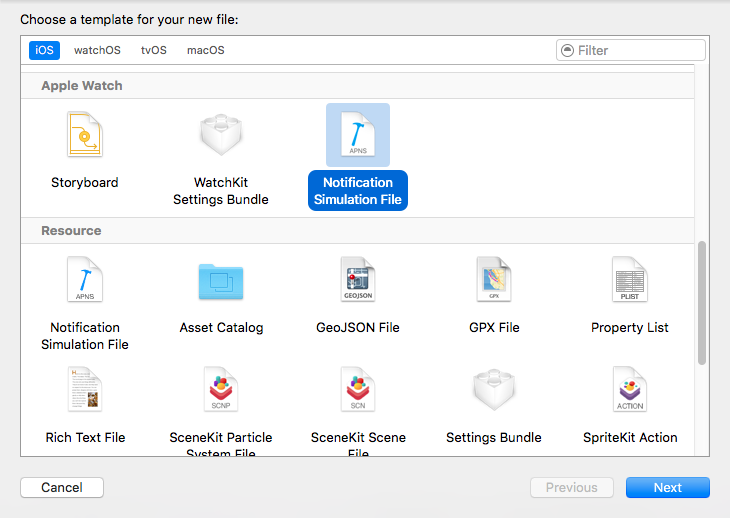
# 三、在真机上模拟远程推送

在 watchOS 模拟器上，Xcode 为我们准备好了一种可以模拟测试推送的方式。如果我们创建项目时，选择了 NotifacationScene，Xcode 会默认为我们创建一个 apns 文件：



PushNotificationPayload.apns

这个文件就是模拟推送的相关配置文件，如果没有，也可以手动创建：



RUTU

文件中的内容格式如下：

{ "aps": { "alert": { "body": "通知", "title": "通知来了" }, "category": "requestIdentifier" }, "WatchKit Simulator Actions": [ { "title": "First Button", "identifier": "firstButtonAction" } ], "customKey": "Use this file to define a testing payload for your notifications. The aps dictionary specifies the category, alert text and title. The WatchKit Simulator Actions array can provide info for one or more action buttons in addition to the standard Dismiss button. Any other top level keys are custom payload. If you have multiple such JSON files in your project, you'll be able to select them when choosing to debug the notification interface of your Watch App." }

这是一些 Json 格式的数据，其中 alert 是对推送内容的设置，body 会显示在 long-look 的标题栏，title 会显示在 short-look 的标题栏，Actions 数组中是对按钮就行配置，每一个按钮可以设置一个标题和 id，标题用于在推送界面显示，id 用于处理点击按钮后触发的逻辑。注意：这里的 category 一定要和上文代码中设置的 requestIdentifier 一致。

创建好这个，我们可以来试着测试一下推送的界面，选择推送工程，运行即可。

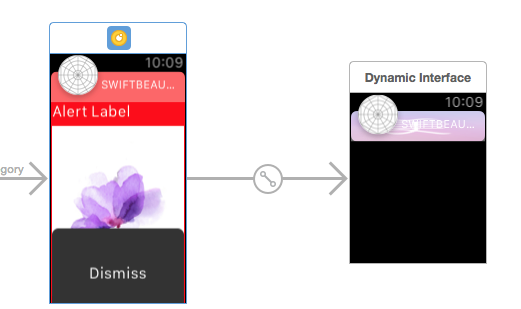
**四、long-look 的静态界面和动态界面**

上面提到过，long-look 分为静态界面和动态界面两种，当我们在 storyboard 中拉入 Notification Interface Controller 的时候，可以选择同时创建动态界面，勾选 Has Dynamic Interface：

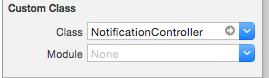
作者：小道萧兮  
链接：https://www.jianshu.com/p/856519d9f667  
来源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。



这时，在 storyboard 中是如下模样：



我们在创建一个文件，继承于 WKUserNotificationInterfaceController，并将 storyboard 中Dynamic Interface的class设置为我们创建的类：



注意，这里设置的是动态的 Interface，也就是上面右边的 controller。之后运行，你会发现效果并没有什么改变，那是因为系统默认会从静态界面加载推送界面，我们需要在 NotifacationController 代码中做一些操作：

//在NotificationController中重写下面方法 override func didReceive(\_ notification: UNNotification, withCompletion completionHandler: @escaping (WKUserNotificationInterfaceType) -> Void) { completionHandler(.custom) }

# 五、触发推送点击事件

首先，我们多配置几个点击按钮，在apns文件中如下配置：

"WatchKit Simulator Actions": [ { "title": "第一", "identifier": "one" }, { "title": "第二", "identifier": "two" }, { "title": "第三", "identifier": "three" } ],

在我们watch App的InterfaceController中实现如下的方法：

import UserNotifications //重写下面方法来响应点击事件 override func handleAction(withIdentifier identifier: String?, for notification: UNNotification) { }

现在，当你手机收到通知时，看看你的watch。

# 第七章：AppleWatch开发入门——AlertController

**一、简介**

AlertController是wiatch中的警告框，它有3种Style：

* alert
* actionsheet
* sldeBySldeButtonsAlert

3中Style分别如下图所示：

# 

# Alert Style

# 

Actionsheet Style



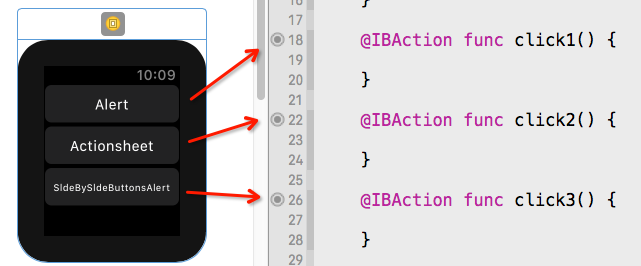
sldeBySldeButtonsAlert Style

这里需要注意的是：

* Alert和Actionsheet按钮数量不限，而sldeBySldeButtonsAlert只能有两个按钮，如果设置了多个按钮，则无法弹出。
* Alert取消按钮只能有一个，如果设置多个，也只有一个的效果。

# 二、创建3种AlertController

1. 界面布局与连线  
   如下图：



1. 详细代码
2. @IBAction func click1() { alertView(style: .actionSheet) } @IBAction func click2() { alertView(style: .alert) } @IBAction func click3() { alertView(style: .sideBySideButtonsAlert) } func alertView(style: WKAlertControllerStyle) { // 创建一个取消按钮 let action1 = WKAlertAction(title: "取消按钮", style: .cancel) { print("点击了取消按钮") } // 创建一个默认按钮 let action2 = WKAlertAction(title: "默认按钮", style: .default) { print("点击了默认按钮") } // 创建一个警告按钮 let action3 = WKAlertAction(title: "警告按钮", style: .destructive) { print("点击了警告按钮") } // 弹出alert，设置标题和信息，并把上面创建的按钮加入到alert中 presentAlert(withTitle: "消息标题", message: "在这里可以输入消息的内容", preferredStyle: style, actions: [action1, action2, action3]) }
3. 至此，已经大功告成，不过要注意，sideBySideButtonsAlert只能有两个按钮，所以在click3中要进行特殊处理。

# 第八章：AppleWatch开发入门——动画

**一、简介**

动画一直是 iOS 系统的一大亮点，CoreAnimation 和粒子效果的支持，开发者可以很容易的做出效果炫酷的动画特效。在 watchOS 中，由于性能和屏幕尺寸的限制，对于动画，并没有强大的框架支持，但是这并不是说开发者就没办法在watch上添加动画的特效了。

**二、移动动画**

在 watch 中的移动动画不像 iPhone 上那么复杂，目前只支持水平、垂直移动，并且移动的距离是受限的。水平移动的距离是一个枚举：left、center、right，也就是说你只能把控件移动到屏幕的左边、中间、右边，而不能控制具体移动到哪个位置；垂直方向移动的距离也一样是枚举：top、center、bottom，这两个枚举如下：

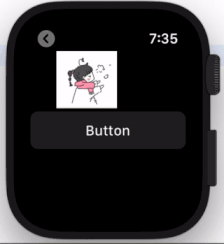
@available(watchOS 2.0, \*) public enum WKInterfaceObjectHorizontalAlignment : Int { case left case center case right } @available(watchOS 2.0, \*) public enum WKInterfaceObjectVerticalAlignment : Int { case top case center case bottom }

执行动画的方法如下：

// 第一个参数是动画时间，第二个参数是闭包，里面写如何移动 @available(watchOS 2.0, \*) open func animate(withDuration duration: TimeInterval, animations: @escaping () -> Void)

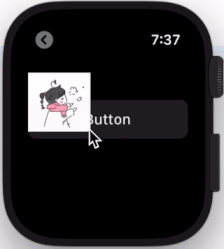
代码实现非常简单：

@IBAction func startHorizontalAnimate() { // 水平移动到右边 animate(withDuration: 2) { self.image.setHorizontalAlignment(.right) } }



水平移动

@IBAction func startVerticalAnimate() { // 垂直移动到底部 animate(withDuration: 2) { self.image.setVerticalAlignment(.bottom) } }



垂直移动

这里比较有趣，我只是把图片往下移动了，结果 button 自动移上去。

**三、帧动画**

和 iOS 类似，watchOS 中的帧动画也是通过 UIImage 对象的合集来展示的。只是设置和用法略有不同。  
首先，watchOS 中帧动画的操作被单独封装成了一个协议，当然，WKInterfaceImage 类已经遵守了这个协议的：

public protocol WKImageAnimatable : NSObjectProtocol { //从默认帧开始播放动画 public func startAnimating() //播放一个指定范围的帧动画 NSRange是帧的范围，durtion是播放一遍的时间，repeatCount是重复播放次数，0为无限循环 public func startAnimatingWithImages(in imageRange: NSRange, duration: TimeInterval, repeatCount: Int) //停止播放动画 public func stopAnimating() }

创建帧动画的步骤与一些注意：

1. 关联一个视图中的WKInterfaceImage对象
2. 所有帧动画的图片帧必须有统一的格式：比如image1.png，image2.png等等
3. 给WKInterfaceImage对象设置帧前缀：
4. image.setImageNamed("image")

**注意：**这里使用的方法和设置图片的方法一样，但是参数有别，图片的设置需要完整的图片名，动画帧前缀的设置只要设置帧图片的前缀，并且素材帧必须放入watchKit App这个Target中，才可以使用。

1. 开始动画
2. // range的location是图片的起始张，length是图片的总数 let range = NSRange(location: 1, length: 18) image.startAnimatingWithImages(in: range, duration: 2, repeatCount: 0)



如图