

---

# iOS 9 应用开发基础教程

(内部资料 v1.0)



大学霸

[www.daxueba.net](http://www.daxueba.net)



---

# 前言

iOS 9 是由苹果公司开发的移动操作系统。它是苹果公司在 2015 年 6 月 8 日的 WWDC 大会上发布的最新的移动操作系统。苹果公司开发人员为 iOS 9 新增了很多的新的功能，例如 iPad 可以支持多屏功能；在地图方面新增 Transit 功能，也就是可以支持公交导航；还添加了自动建议功能，可以根据用户的习惯、时间、地点、App 等来预测用户的行动、给出合理的建议等等，对于 iOS 9 用户来说会感觉到该系统的更加便捷与实用。

现在很多的苹果开发人员已经开始由原来的 iOS 8 开发转向了 iOS 9 开发，稍微关注一下，就会发现 iOS 9 已成为了 iOS 开发的主流。但是市面上对 iOS 9 开发方面的介绍很少，特别是使用 Swift 2.0 语言去进行 iOS 9 开发更是凤毛菱角。

基于以上不可忽略的事实，本书决定着眼于讲解用 Swift 2.0 语言去开发 iOS 9 应用程序基础，并且书中将对每一个知识点都会配有大量的图片以及示例，相信这样会更好的帮助读者去学习本书。

## 1.学习所需的系统和软件

- ☐ 安装 Mac OS 10.10.4 操作系统
- ☐ 安装 Xcode 7.0

## 2.学习建议

大家学习之前，可以致信到 xxxxxxxxxx，获取相关的资料和软件。如果大家在学习过程遇到问题，也可以将问题发送到该邮箱。我们尽可能给大家解决。



---

# 上册目录

第 1 章	iOS 9 开发概述	1
1.1	iOS 9 新特性	1
1.1.1	Siri 语音助手智能化	1
1.1.2	新增 News 新闻聚合应用	1
1.1.3	ApplePay 支持购物 Wallet 取代 Passbook	1
1.1.4	地图引入 Transit 通勤路线功能	1
1.1.5	备忘录进入核查清单组件	1
1.1.6	键盘加入 Shortcut Bar	2
1.1.7	iPad 引入双屏模式，支持 QuickType 键盘	2
1.2	编写第一个 iOS 9 应用	2
1.2.1	创建项目	2
1.2.2	运行程序	5
1.2.3	iOS 模拟器介绍	7
1.2.4	编辑界面	11
1.2.5	编写代码	15
1.2.6	定制应用程序图标	20
1.2.7	真机测试	24
1.3	了解视图	24
1.3.1	视图库介绍	24
1.3.2	视图始祖——UIView	26
1.3.3	添加视图	26
1.3.4	视图的位置和大小	30
1.3.5	删除空白视图	30
第 2 章	丰富的用户界面	32
2.1	使用按钮接收用户输入	32
2.1.1	使用代码添加按钮	32
2.1.2	美化按钮	33
2.1.3	实现按钮的响应	35
2.2	显示、编辑文本	42
2.2.1	只读文本——标签	42
2.2.2	单行读写文本——文本框	45
2.2.3	多行读写文本——文本视图	47
2.3	使用开关、滑块控件	51
2.3.1	开关	51
2.3.2	滑块控件	53
2.4	屏幕滚动视图——滚动视图	55
2.5	iOS 9 新增——部署视图	57
第 3 章	键盘输入	62

3.1	显示键盘 .....	62
3.1.1	用户输入时显示键盘 .....	62
3.1.2	直接显示键盘 .....	63
3.2	定制键盘的输入类型 .....	63
3.3	关闭键盘 .....	65
3.3.1	使用 <b>Return</b> 键关闭键盘 .....	65
3.3.2	触摸背景关闭键盘 .....	67
3.3.3	使用菜单项关闭键盘 .....	69
3.4	为键盘添加工具栏 .....	72
3.5	显示键盘时改变输入视图的位置 .....	73
第 4 章	图形图像 .....	76
4.1	显示图像 .....	76
4.2	定制特殊的图像 .....	80
4.2.1	旋转图像 .....	80
4.2.2	缩放图像 .....	81
4.3	页面控件 .....	82
4.4	绘制路径 .....	85
4.4.1	绘制线段 .....	85
4.4.2	绘制不同的线段 .....	88
4.4.3	绘制曲线 .....	91
4.4.4	绘制形状 .....	93
4.4.5	路径函数总结 .....	94
4.4.6	美化路径 .....	95
4.5	绘制位图 .....	97
4.5.1	绘制单个位图 .....	97
4.5.2	绘制多个位图 .....	98
4.6	绘制文字 .....	99
第 5 章	引起用户注意 .....	101
5.1	进度条 .....	101
5.2	活动指示器 .....	103
5.3	警告视图 .....	106
5.3.1	为主视图添加警告视图 .....	107
5.3.2	常见的警告视图样式 .....	108
5.3.3	响应警告视图 .....	111
5.4	操作表 .....	114
5.4.1	为主视图添加操作表 .....	114
5.4.2	响应操作表 .....	115
第 6 章	表 .....	118
6.1	表视图的分类 .....	118
6.1.1	普通表视图 .....	118
6.1.2	分组表视图 .....	118

6.2	在表中显示数据 .....	119
6.2.1	在普通表视图中显示数据 .....	119
6.2.2	在分组表视图中显示数据 .....	122
6.3	美化表 .....	124
6.3.1	添加页眉、页脚 .....	124
6.3.2	添加索引 .....	126
6.3.3	缩进 .....	127
6.3.4	设置行高 .....	128
6.4	设置表单元格 .....	129
6.4.1	设置表单元格的风格 .....	129
6.4.2	设置表单元格的标记 .....	132
6.4.3	在表单元格中添加其他视图 .....	133
6.5	对表进行操作 .....	135
6.5.1	选取行 .....	135
6.5.2	插入行 .....	137
6.5.3	删除行 .....	139
6.5.4	移动行 .....	141
6.6	在表中添加搜索功能 .....	143
第 7 章	选择器 .....	147
7.1	日期选择器 .....	147
7.2	自定义选择器 .....	151
7.2.1	自定义选择器的分类 .....	151
7.2.2	填充数据 .....	151
7.2.3	响应自定义选择器 .....	155
第 8 章	多视图应用 .....	159
8.1	多视图的实现 .....	159
8.2	多视图的切换 .....	163
8.2.1	启动界面 .....	163
8.2.2	Storyboard 实现切换 .....	163
8.2.3	分段控件 .....	168
8.2.4	导航控制器 .....	171
8.2.5	标签栏控制器 .....	180
第 9 章	屏幕旋转 .....	191
9.1	实现旋转屏幕的常用方式 .....	191
9.1.1	按键 .....	191
9.1.2	目标窗口 .....	192
9.1.3	Info.plist 文件 .....	193
9.1.4	使用代码 .....	193
9.2	旋转时重新定位视图 .....	194
第 10 章	触摸与手势 .....	198
10.1	触摸 .....	198

10.1.1 触摸阶段 .....	198
10.1.2 处理触摸 .....	198
10.2 常用手势 .....	204
10.2.1 轻拍 .....	204
10.2.2 捏 .....	207
10.2.3 滑动 .....	209
10.2.4 旋转 .....	211
10.2.5 移动 .....	212
10.2.6 长按 .....	214
10.3 自定义手势 .....	216

## 下 册 目 录

第 11 章 数据管理 .....	1
11.1 文件管理 .....	1
11.1.1 创建文件 .....	1
11.1.2 写入数据 .....	3
11.1.3 读取数据 .....	6
11.1.4 删除文件 .....	9
11.2 使用 SQLite 数据库 .....	12
11.2.1 SQLite 数据类型 .....	12
11.2.2 创建数据库 .....	13
11.2.3 创建表 .....	18
11.2.4 插入数据 .....	19
11.2.5 查询数据 .....	22
第 12 章 访问网络 .....	24
12.1 Web 浏览器——网页视图 .....	24
12.1.1 加载网页视图的内容 .....	24
12.1.2 设置网页视图 .....	29
12.1.3 网页视图与 Javascript 交互 .....	32
12.1.4 为网页视图添加导航动作 .....	34
12.1.5 网页视图进行加载时常用方法 .....	37
12.2 网络服务 .....	40
第 13 章 定位服务与地图应用 .....	44
13.1 获取位置数据 .....	44
13.2 获取位置方向 .....	47
13.3 区域监听 .....	48
13.4 地图应用 .....	50
13.4.1 显示地图 .....	50



13.4.2	地图显示模式 .....	51
13.4.3	显示当前位置 .....	53
13.4.4	指定位置 .....	55
13.4.5	添加标记 .....	57
13.4.6	添加标注 .....	58
13.4.7	限制地图的显示范围 .....	60
13.4.8	3D 地图 .....	61
13.5	地理编码 .....	62
13.6	反地理编码 .....	64
13.7	iOS 9 地图新特性——Transit 通勤路线功能 .....	67
13.8	使用谷歌 Web 地图 .....	71
第 14 章	多媒体 .....	74
14.1	图像的选取——图像选取器 .....	74
14.1.1	选取图像 .....	74
14.1.2	设置图像的来源 .....	77
14.1.3	在模拟器中对图像的操作 .....	80
14.1.4	选取视频 .....	84
14.2	使用相机 .....	86
14.2.1	打开相机 .....	86
14.2.2	设置相机 .....	88
14.2.3	捕获媒体 .....	90
14.3	音频 .....	93
14.3.1	播放系统声音 .....	93
14.3.2	播放音频文件 .....	96
14.3.3	访问音乐库 .....	99
14.3.4	选取音频 .....	101
14.4	使用麦克风录制声音 .....	103
14.5	视频 .....	106
第 15 章	动画 .....	110
15.1	图像视图动画 .....	110
15.2	视图动画 .....	112
15.2.1	动画块 .....	113
15.2.2	修改动画块 .....	115
15.2.3	基于块的视图动画 .....	116
15.3	过渡动画 .....	117
15.3.1	翻页动画 .....	118
15.3.2	旋转动画 .....	120
15.4	CATransition 动画 .....	122
15.5	定时器 .....	126
第 16 章	内置应用程序 .....	129
16.1	打电话 .....	129

16.2	使用 Safari.....	131
16.3	管理通讯录 .....	134
16.3.1	打开通讯录 .....	134
16.3.2	选取联系人 .....	135
16.3.3	添加联系人 .....	138
16.3.4	显示联系人信息 .....	140
16.3.5	完善联系人信息 .....	143
16.4	管理日历事件 .....	144
16.4.1	打开日历事件界面 .....	145
16.4.2	添加日历事件 .....	146
16.5	发送信息 .....	152
16.5.1	使用信息应用发送信息 .....	152
16.5.2	使用邮件应用发送信息 .....	157
第 17 章	与外部设备交互 .....	161
17.1	获取设备信息 .....	161
17.2	获取电池信息 .....	163
17.3	处理运动事件 .....	166
17.4	获取加速计数据 .....	170
17.5	访问陀螺仪数据 .....	174
17.6	近距离传感器 .....	177

---

# 第 1 章 iOS 9 开发概述

iOS 9 是目前苹果公司用于苹果手机和苹果平板电脑的最新的操作系统。该操作系统于 2015 年 6 月 8 号（美国时间）被发布。本章将主要讲解 iOS 9 的新特性、以及使用 Xcode 7.0 如何编写一个简单的 iOS 9 的应用程序等内容。

## 1.1 iOS 9 新特性

在 2015 年 6 月 8 日即北京时间 2015 年 6 月 9 日的 WWDC 大会上，苹果公司代表讲解了很多 iOS 9 带来的新特性。本节将讲解一些主要的特性。

### 1.1.1 Siri 语音助手智能化

Siri 语音助手的智能性能主要表现在强大的内容检索和管理方面，支持快速自动整理历史文档，根据使用场景的不同为用户提供内容和服务，来电联系人匹配，相关内容推荐甚至是第三方应用的内容检索。具体到使用场景方面，Siri 可以自动整理历史照片、联系人历史邮件往来、健身应用下接入电源开启 iTunes 音乐应用以及深度检索第三方应用中的内容。

### 1.1.2 新增 News 新闻聚合应用

苹果在新的 iOS9 系统中推出了全新的系统级新闻聚合应用——News，新加入的 News 应用将会根据用户习惯推送用户可能关注的新闻，新闻将按话题分类，版面也将进行特殊调整。特别需要注意 News 应用中的内容将注重隐私保护，不会分享给第三方。

### 1.1.3 ApplePay 支持购物 Wallet 取代 Passbook

ApplePay 将登陆英国并得到了大量第三方应用和商户甚至是公交系统的支持，并且支持在线消费和购物。Passbook 更名为 Wallet，如图 1.1 所示。用户的信用卡、借记卡、积分卡、登机牌、票券等都可以存放于此。

### 1.1.4 地图引入 Transit 通勤路线功能

iOS 9 的地图应用中加入了 Transit 的通勤路线功能，如图 1.2 所示。它可以为用户提供从步行到乘车整个完整的通勤路线，支持公交、火车、地铁、轮渡等交通工具，支持全球多个地区，其中包括国内 300 多个城市。

### 1.1.5 备忘录进入核查清单组件

iOS 9 在备忘录中加入了核查清单组件，用户可以从备忘录里直接启动相机来添加照片和画草图，浏览器、地图等其他应用程序中的内容也可以直接添加。

### 1.1.6 键盘加入 Shortcut Bar

不管在 iPhone 还是在 iPad 上，iOS 9 的键盘都加入了全新的 Shortcut Bar，如图 1.3 所示。通过这个新的功能条可以更便捷的进行剪切、粘贴等基本操作。

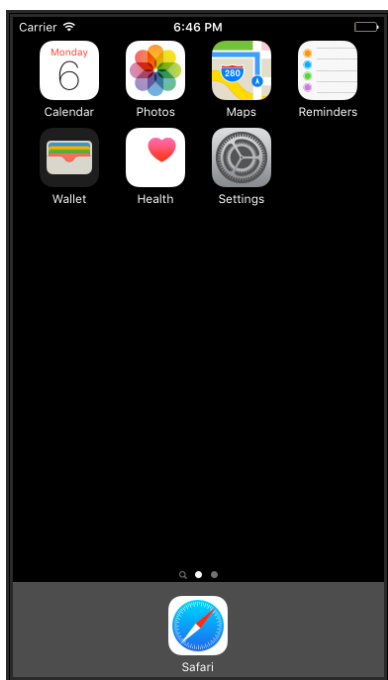


图 1.1 Passbook 更名为 Wallet

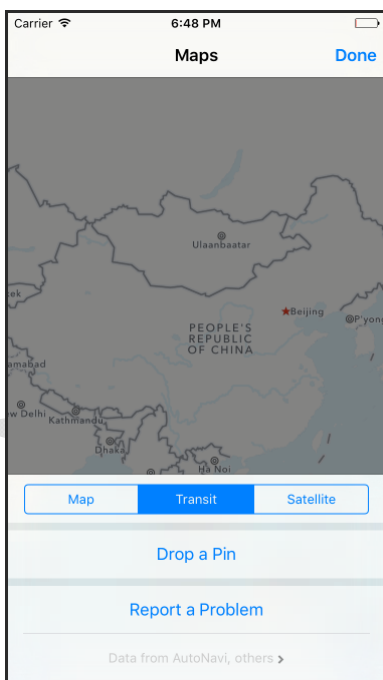


图 1.2 地图引入 Transit 通勤路线功能

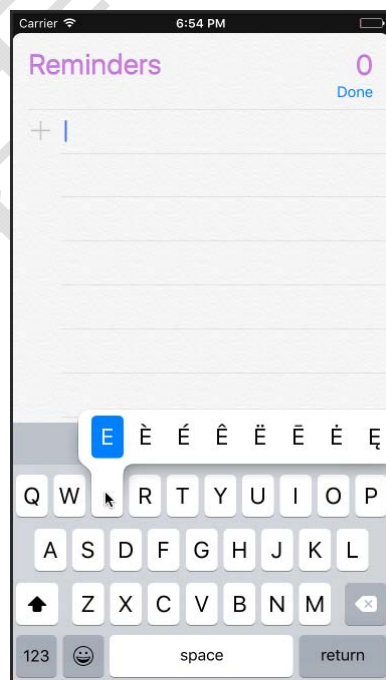


图 1.3 键盘加入 Shortcut Bar

### 1.1.7 iPad 引入双屏模式，支持 QuickType 键盘

iOS 9 分屏功能支持 iPad Air、iPad Air 2，iPad mini 2 和 iPad mini 3，可以让两个不同的应用在屏幕上同时工作，界面可以有 5: 5 和 7: 3 两种比例选择，用户可以自行切换某一个屏幕上的程序。另外，视频应用可以变成小窗悬浮在界面之上。与此同时，苹果在 iPad 中推出 QuickType 键盘应用，它可以用两根手指在键盘上变换操作，支持剪切、复制、粘贴快捷键，并且在搜索、全局搜索中都将可以使用此功能。

## 1.2 编写第一个 iOS 9 应用

本节将以一个 iOS 9 应用程序为例，为开发者讲解如何使用 Xcode 7.0 去创建项目，以及 iOS 模拟器的一些功能、编辑界面等内容。

### 1.2.1 创建项目

一个 iOS 应用的所有文件都在一个 Xcode 项目下。项目可以帮助用户管理代码文件和资源文件。以下是使用 Xcode 创建项目的具体操作步骤

(1) 打开 Xcode，弹出 Welcome to Xcode 对话框，如图 1.4 所示。

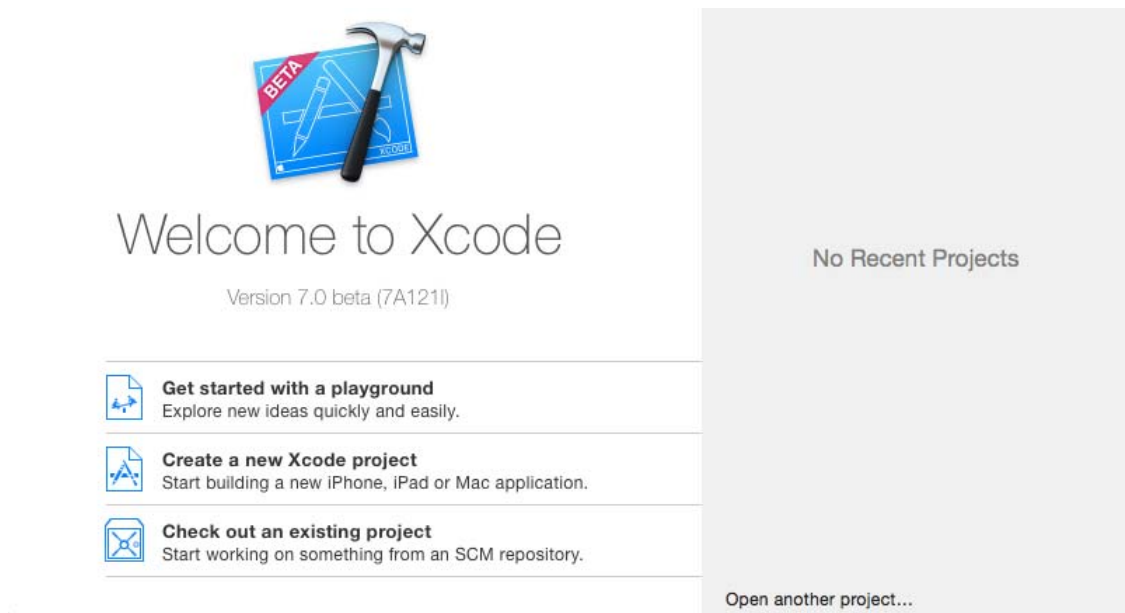


图 1.4 Welcome to Xcode 对话框

(2) 选择 Create a new Xcode project 选项，弹出 Choose a template for your new project:对话框，如图 1.5 所示。

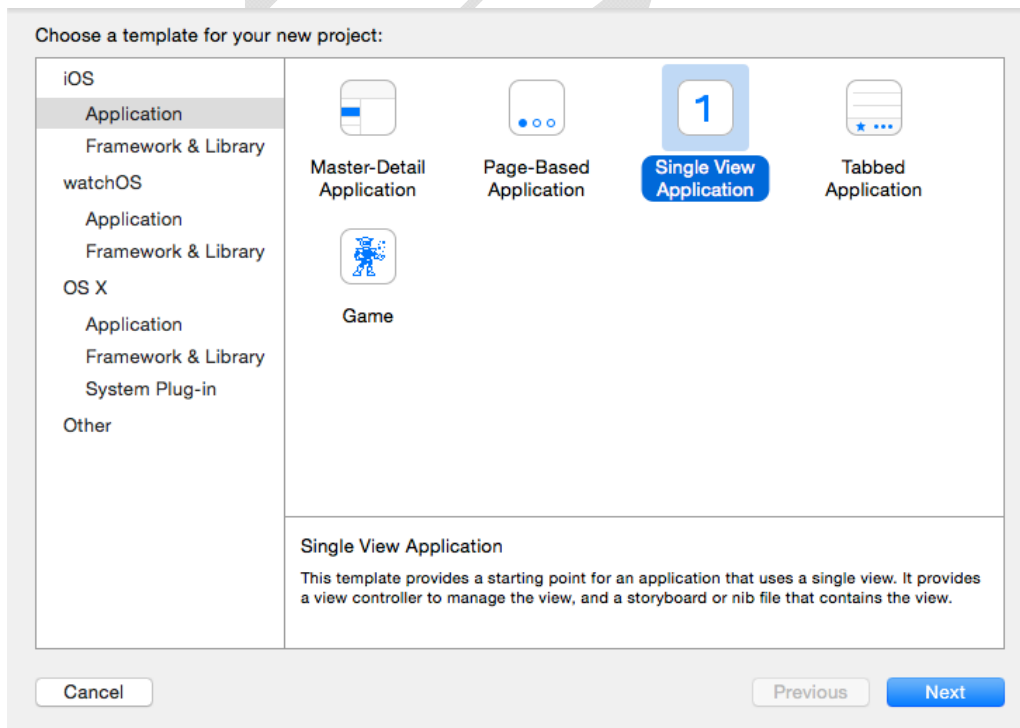


图 1.5 Choose a template for your new project:对话框

(3) 选择 iOS|Application 中的 Single View Application 模板, 单击 Next 按钮后, 弹出 Choose options for your new project:对话框, 如图 1.6 所示。

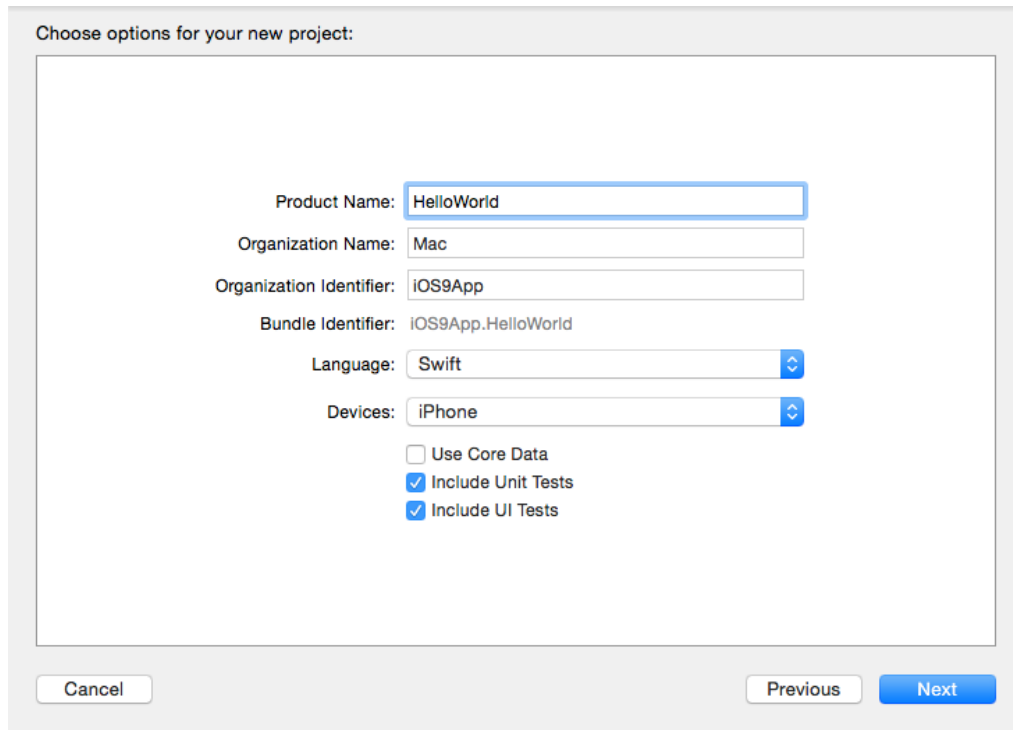


图 1.6 Choose options for your new project:对话框

(4) 填入 Product Name (项目名)、Organization Identifier (标识符) 信息以及选择 Language (编程语言) 和设备 Devices (设备), 如表 1-1 所示。

表 1-1 填写的内容

需要填写的项	填入的内容
Product Name:	Hello World
Organization Identifier	iOS9App
Language	Swift
Devices	iPhone

注意: 在图 1.6 中出现的 UI Tests 是 Xcode 7.0 新增的内容。UI Tests 是一个自动测试 UI 与交互的 Testing 组件。它可以通过编写代码、或者是记录开发者的操作过程并代码化, 来实现自动点击某个按钮、视图, 或者自动输入文字等功能。(在实际的开发过程中, 随着项目越做越大, 功能越来越多, 仅仅靠人工操作的方式来覆盖所有测试用例是非常困难的, 尤其是加入新功能以后, 旧的功能也要重新测试一遍, 这导致了测试需要花非常多的时间来进行回归测试, 这里产生了大量重复的工作, 而这些重复的工作有些是可以自动完成的, 这时候 UI Tests 就可以帮助解决这个问题了。)

(5) 内容填写完毕后, 单击 Next 按钮, 打开项目的保存位置对话框, 如图 1.7 所示。

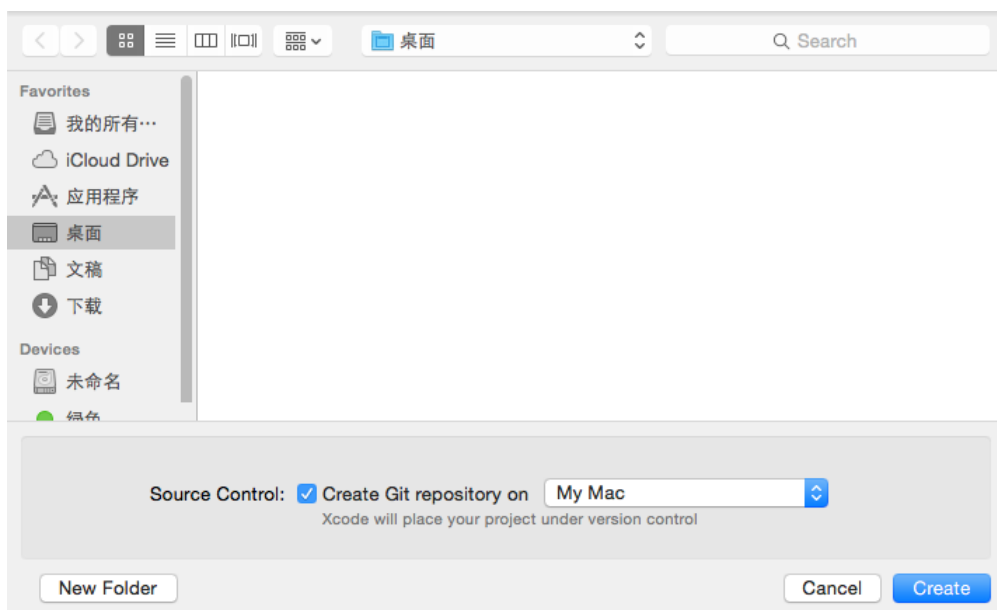


图 1.7 保存位置对话框

(6) 单击 Create 按钮，这时一个项目名为 Hello World 的项目就创建好了，如图 1.8 所示

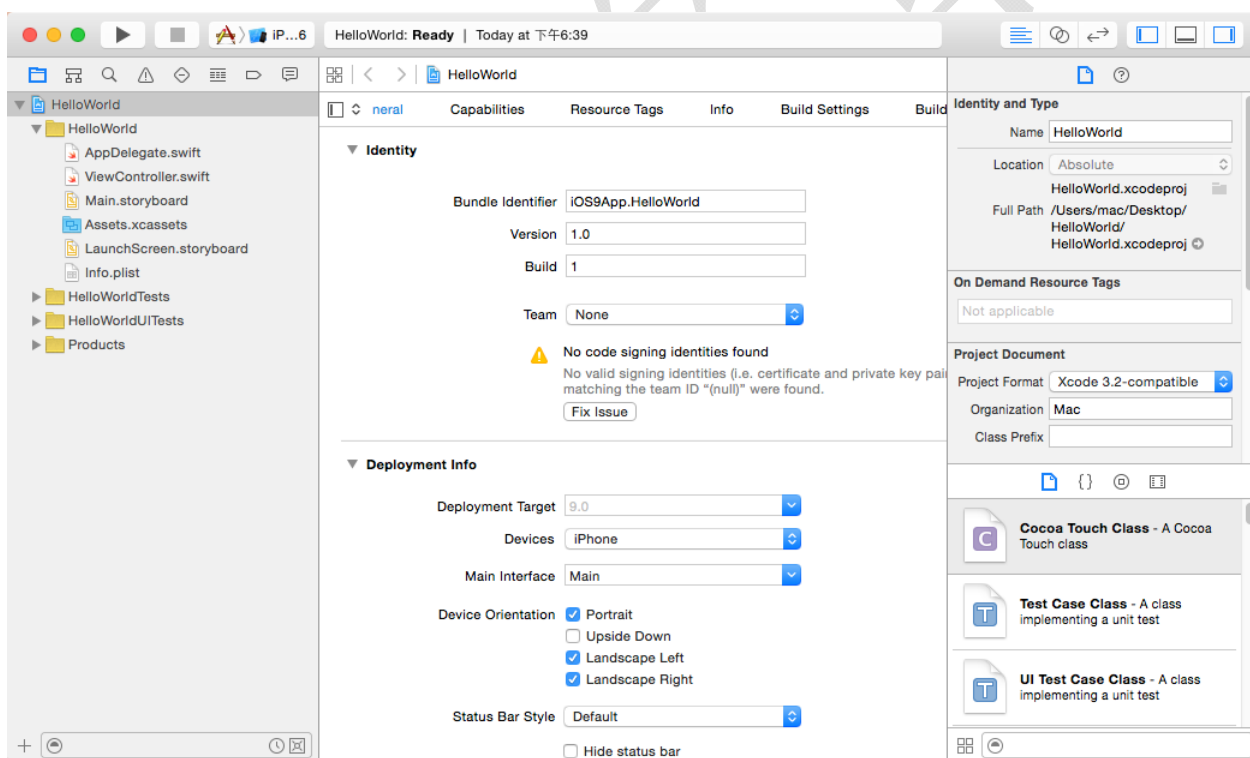


图 1.8 HelloWorld 项目

## 1.2.2 运行程序

创建好项目之后，就可以运行这个项目中的程序了。单击运行按钮，如果程序没有任何问题的话，会看到如图 1.9 和 1.10 的运行效果。

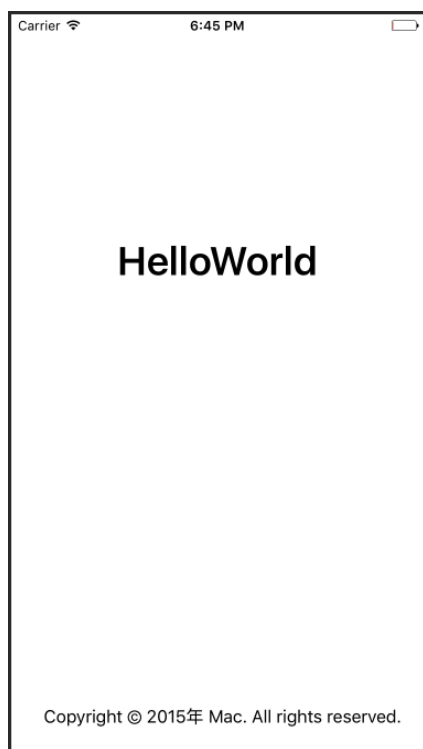


图 1.9 运行效果

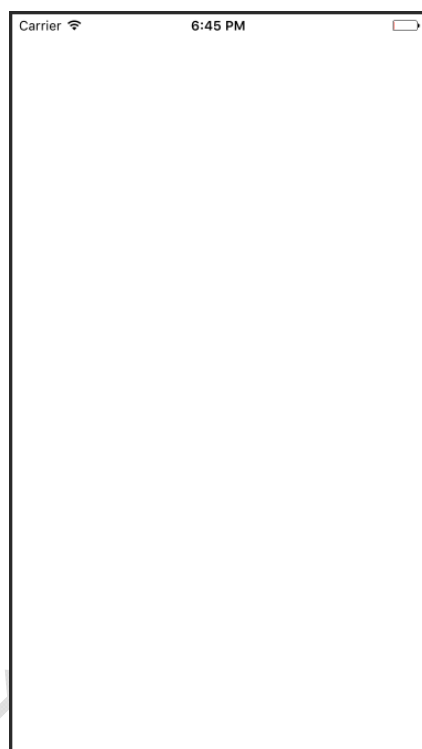


图 1.10 运行效果

注意：由于没有对程序进行编写，也没有对编辑界面进行设置，所有这时运行结果是不会产生任何效果的。对于编辑界面会在后面做一个详细的介绍。

其中，图 1.9 是应用程序的一个启动界面，它是系统自带的。运行程序后，会在此界面停留几秒，然后进入应用程序的主界面，也就是开发者真正要使用到的界面，即图 1.10 所示的界面。如果开发者不想在程序运行时启动界面，可以打开 Info.plist 文件，在此文件找到 Launch screen interface file base name，将其 value 后面的内容删除，如图 1.11 所示，

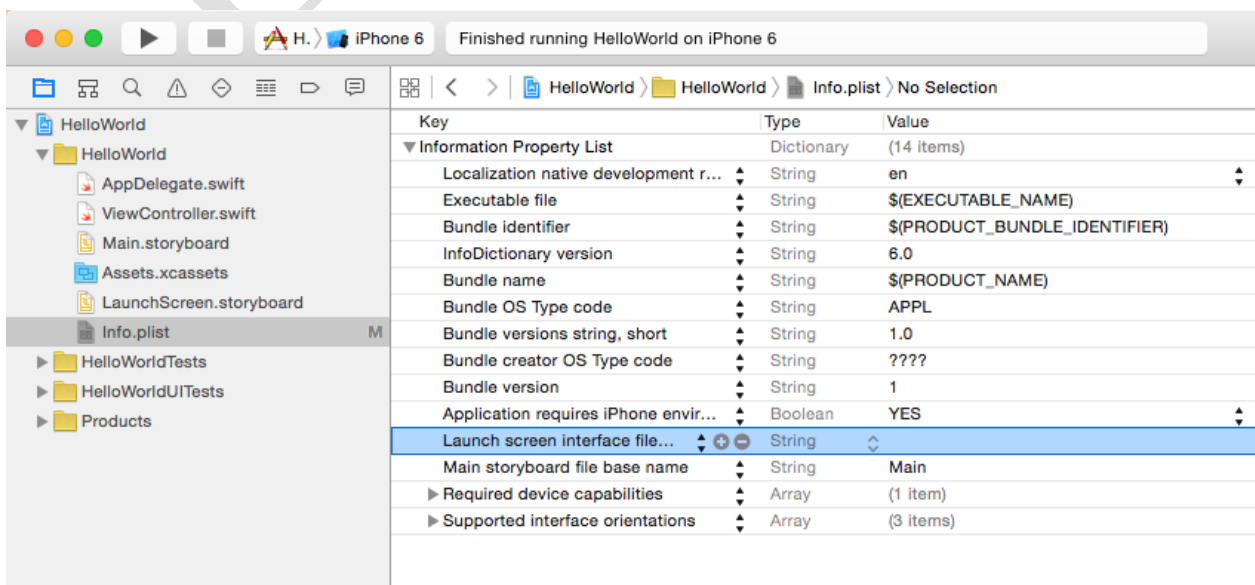


图 1.11 Info.plist 文件



1.2.3 iOS 模拟器介绍

在图 1.9 或者 1.10 中所看到的类似于手机的模型就是 iOS 模拟器。iOS 模拟器是在没有 iPhone 或 iPad 设备时，对程序进行检测的设备。iOS 模拟器可以模仿真实的 iPhone 或 iPad 等设备的一些功能。本小节将讲解一些有关模拟器的操作。

1.模拟器与真机的区别

iOS 模拟器可以模仿真实的 iPhone 或 iPad 等设备的功能各种功能，如表 1-2 所示。

表 1-2 iOS模拟器

方面	功能
旋转屏幕	向上旋转
	向下旋转
	向右向左
手势支持	轻拍
	触摸与按下
	轻拍两次
	猛击
	轻弹
	托
	捏

iOS 模拟器只能实现表 1-4 中的这些功能，其它的功能是实现不了的，如打电话、发送 SMS 信息、获取位置数据、照照相、麦克风等。

2.退出程序

如果想要将图 1.10 所示的应用程序退出(为用户完成某种特定功能所设计的程序被称为应用程序)，该怎么办呢？这时就需要选择菜单栏中的 Hardware|Home 命令，退出应用程序后的效果，如图 1.12 所示。

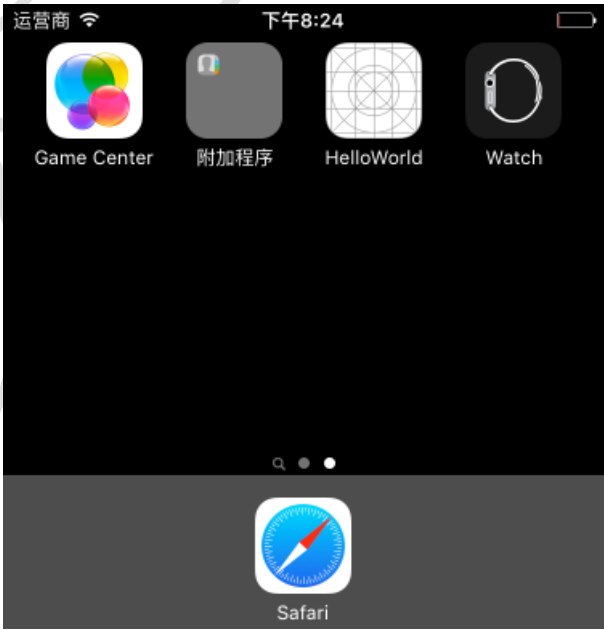


图 1.12 主界面

注意：在图 1.12 中可以看到类似于网状白色图像的图标就是刚才运行的 HelloWorld 应用程序。在 Xcode 中凡是运行后的程序都会显示在 iOS 模拟器的主界面中。当轻拍对应的应用程序的图标后就会进入对应的应用程序。

#### 4. 设置语言

对于不同国家的人来说，使用到的语言是不一样的。一般情况下 iOS 模拟器默认使用的 English（英语）。对于英文不好的开发者来说，英文就像天书，怎么看也看不懂。这时，就需要将 iOS 模拟器的语言进行设置。要设置语言，需要切换到模拟器的主界面，向左拖动，找到 Settings 应用程序。找到后既可以对 iOS 模拟器的语言进行设置了，以下将 iOS 模拟器的语言变为中文，具体操作步骤如下：

（1）切换到主界面，找到 Settings 应用程序，如图 1.13 所示。

（2）选择 Settings 应用程序图标，进入 Settings 界面中，如图 1.14 所示。

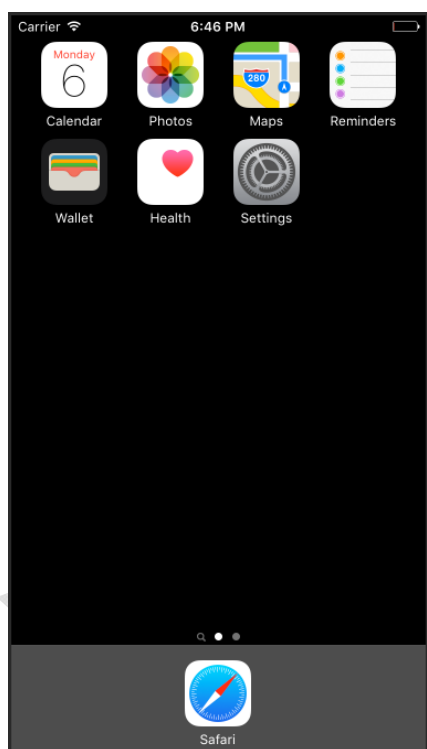


图 1.13 Settings 应用程序

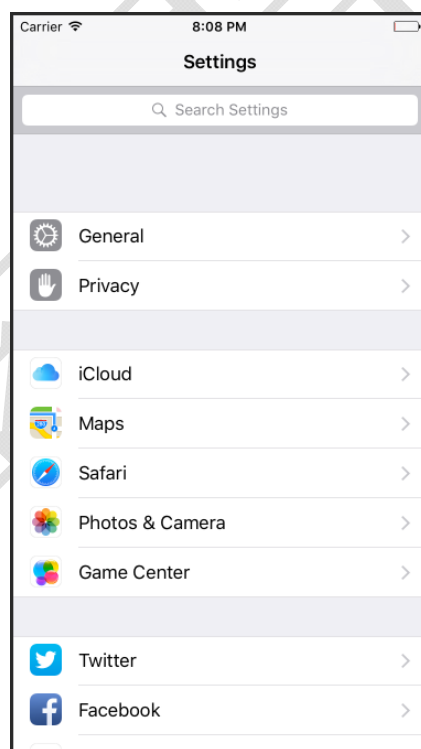


图 1.14 Settings 界面

（3）选择 General 选项，进入 General 界面，如图 1.15 所示。

（4）选择 Language&Region 选项，进入 Language&Region 界面中，如图 1.16 所示。

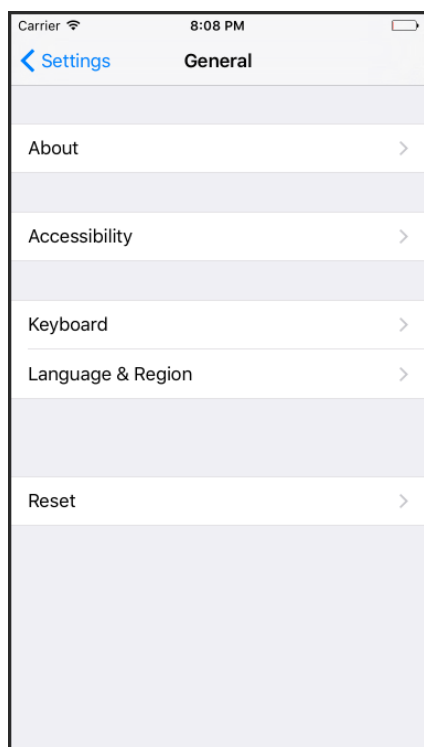


图 1.15 General 界面

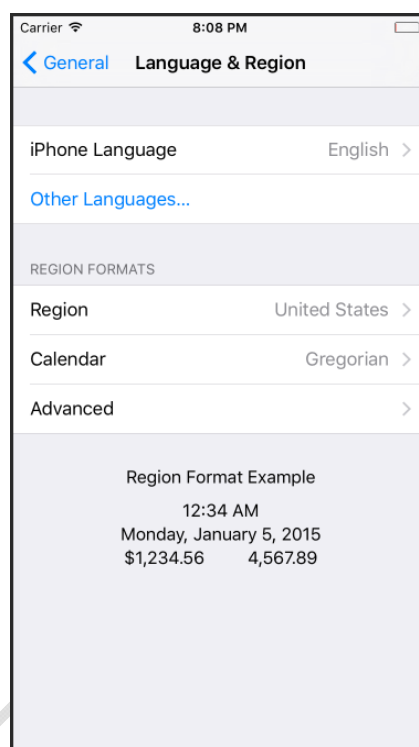


图 1.16 Language&amp;Region 界面

(5) 选择 iPhone Language 选项，进入 iPhone Language 界面，如图 1.17 所示。

(6) 选择“简体中文”选项，轻拍 Done 按钮，弹出动作表单，如图 1.18 所示。

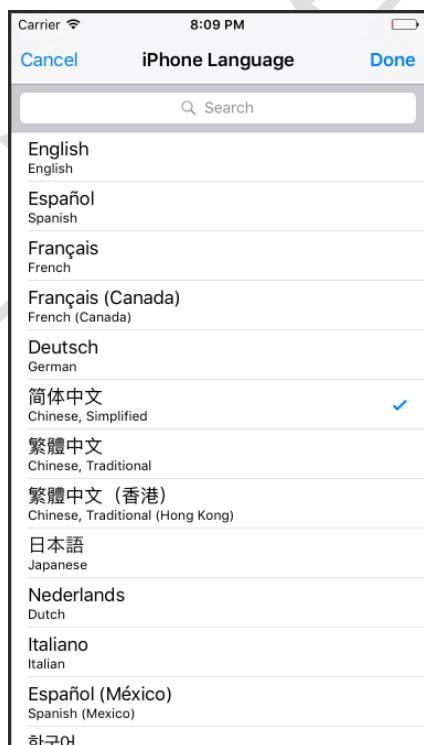


图 1.17 iPhone Language 界面



图 1.18 设置语言



图 1.19 正在设置语言的界面



图 1.20 中文界面

(7) 选择 **Change to Chinese, Simplified** 选项，进入正在设置语言的界面，如图 1.19 所示。当语言设置好后，iOS 模拟器将会退出到主界面，此时主界面的应用程序的标题名就变为了中文，如图 1.20 所示。

## 5. 删除应用程序

随着运行程序的增多，在 iOS 模拟器上显示的图标即应用程序也会增加，开发者可以将那些不再使用到的应用程序删除，这样一来可以为设备节省内存空间，也可以使用户或者开发者便于管理自己的应用程序。以下是删除 Hello World 应用程序的具体操作步骤。

(1) 长按要删除的 **Hello World** 应用程序，直到所有的应用程序都开始抖动，并在每一个应用程序的左上角出现一个“x”，它是一个删除标记，如图 1.21 所示。

(2) 轻拍 **Hello World** 程序左上角出现的删除标记，会弹出一个删除“**Hello World**”对话框，选择其中的“删除”按钮，如图 1.22 所示。这时 **Hello World** 应用程序就在 iOS 模拟器上删除了。

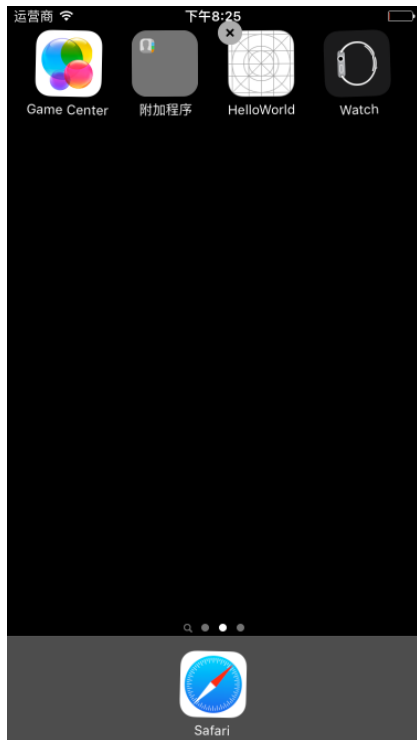


图 1.21 长按 Hello World 应用程序



图 1.22 删除“Hello World”对话框

## 1.2.4 编辑界面

在 1.2.2 小节中提到过编辑界面（Interface builder），编辑界面是用来设计用户界面的，单击打开 Main.storyboard 文件就打开了编辑界面。在 Xcode 5.0 以后中，编辑界面直接使用故事板。本小节将对编辑界面进行介绍

### 1.界面的构成

单击 Main.storyboard 打开编辑界面后，可以看到编辑界面会有 4 部分组成，如图 1.23 所示。

- ☐ 编号为 1 的部分为 dock。
- ☐ 编号为 2 的部分为画布：用于设计用户界面的地方，在画布中用箭头指向的区域就是设计界面，在画布中可以有多个设计界面，一般将设计界面称为场景或者说是主视图。
- ☐ 编号为 3 的部分为工具窗格的检查器：用于编辑当前选择的对象的属性。
- ☐ 编号为 4 的部分为工具窗格的库：如果选择的是 Objects，里边存放了很多的视图。

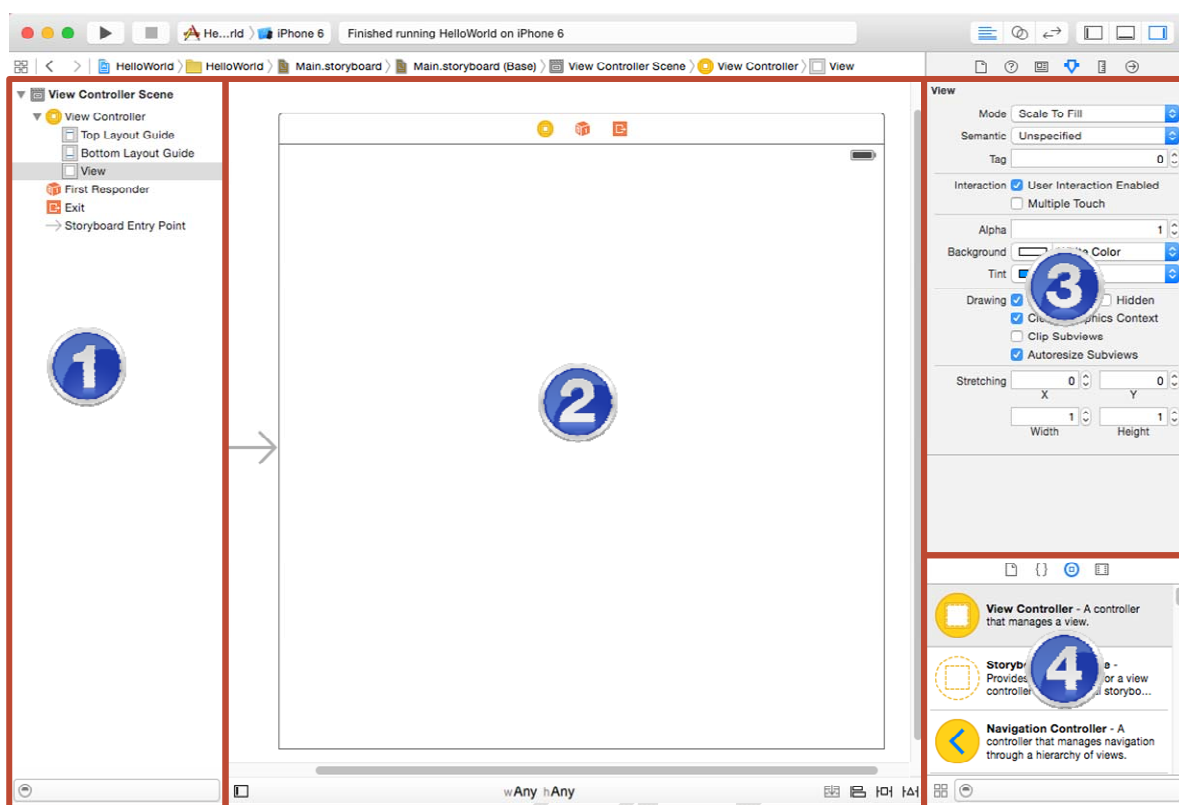


图 1.23 编辑界面构成

- 在画布的设计界面下方有一个小的 dock，它是一个文件管理器的缩减版。dock 展示场景中第一级的控件，每个场景至少有一个 ViewController、一个 FirstResponder 和一个 Exit。但是也可以有其他的控件，dock 还用来简单的连接控件，如图 1.24 所示。

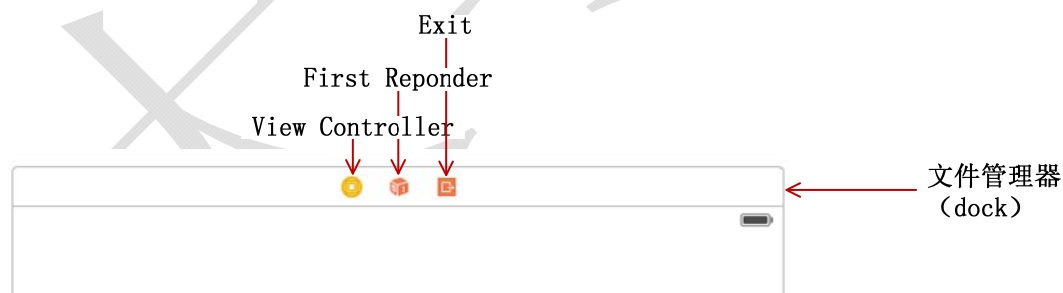


图 1.24 dock

## 2. 设置主视图尺寸

在图 1.23 中看到主视图的尺寸并非是手机的尺寸，其实这个主视图是可以进行调节的。为了让开发者在设计手机界面时可以更加的方便准确，我们可以将其视图尺寸调节成合适的大小。以下是将主视图的尺寸调整为 iPhone 6 手机的尺寸，具体的操作步骤如下：

(1) 选择主视图上方 dock 中的 View Controller。

(2) 在右边的工具窗格的检查器中，选择 Show the Attributes inspector 即属性检查器，在出现的属性检查器面板中将 Size 设置为 iPhone 4.7-inch，如图 1.25 所示。

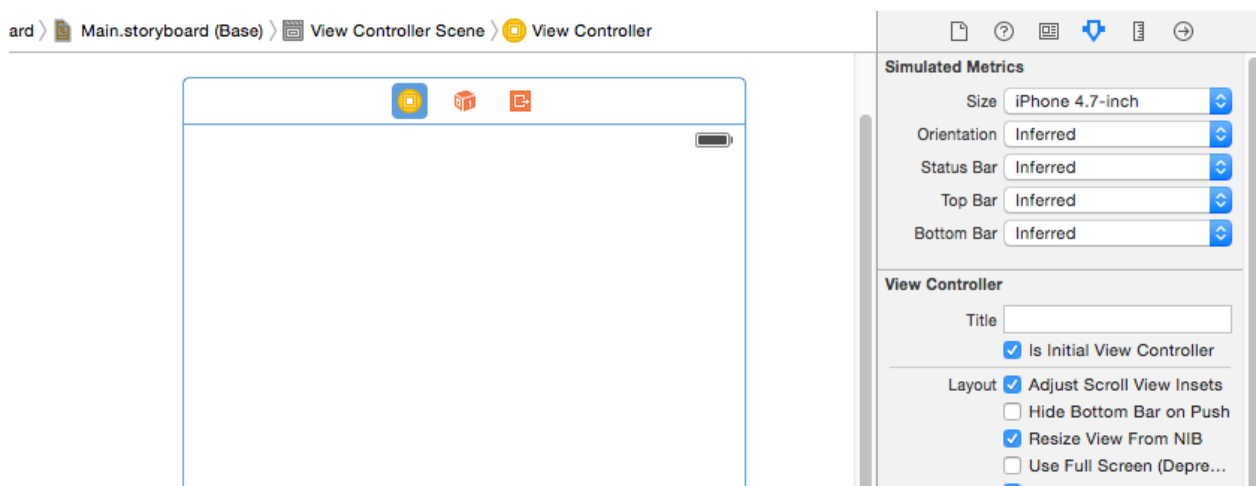


图 1.25 设置设计界面的尺寸

注意：在属性检查器面板中除了可以设置主视图的尺寸外，可以设置方向、状态栏等。对 Size 进行设置后，画布的效果如图 1.26 所示。

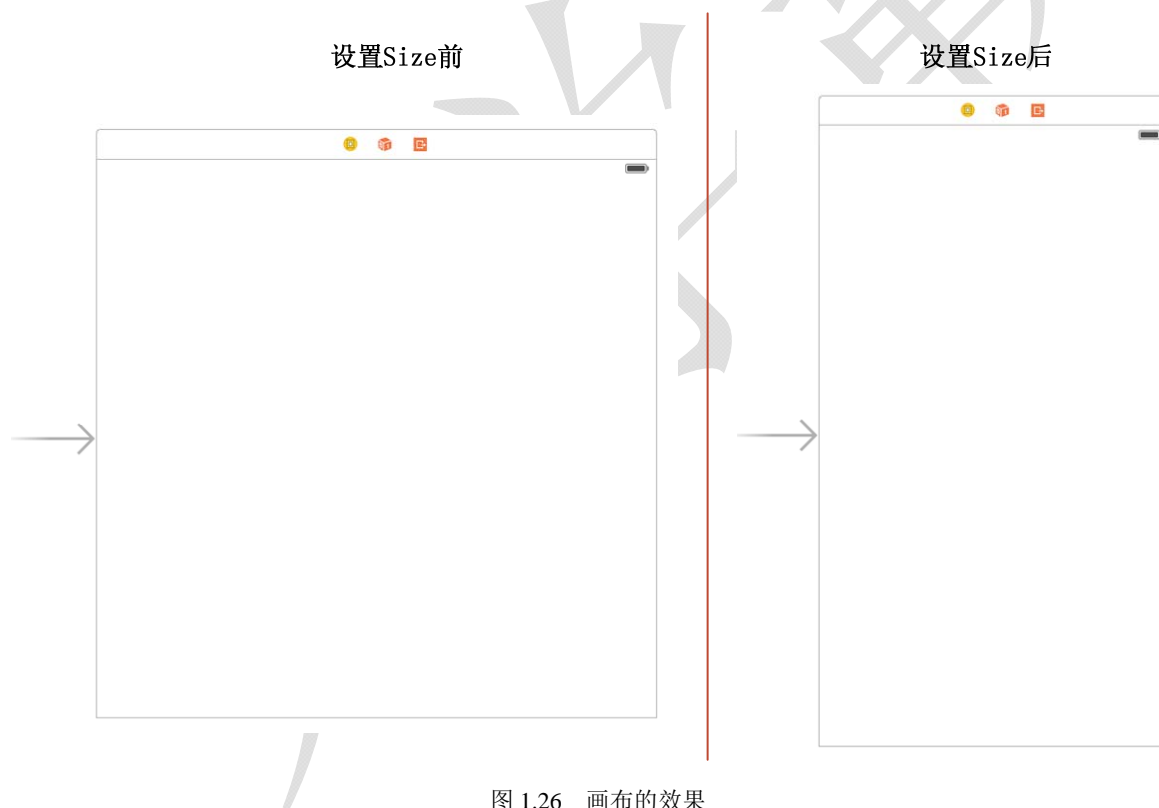


图 1.26 画布的效果

### 3. 添加视图对象

如果想要在 iOS 模拟器上显示一个文本框，就要为主视图添加对象。单击工具窗格库中的 Show the Object Library 即视图库窗口，在里面找到 Text Field 文本框对象将其拖动到画布的主视图中，如图 1.27 所示

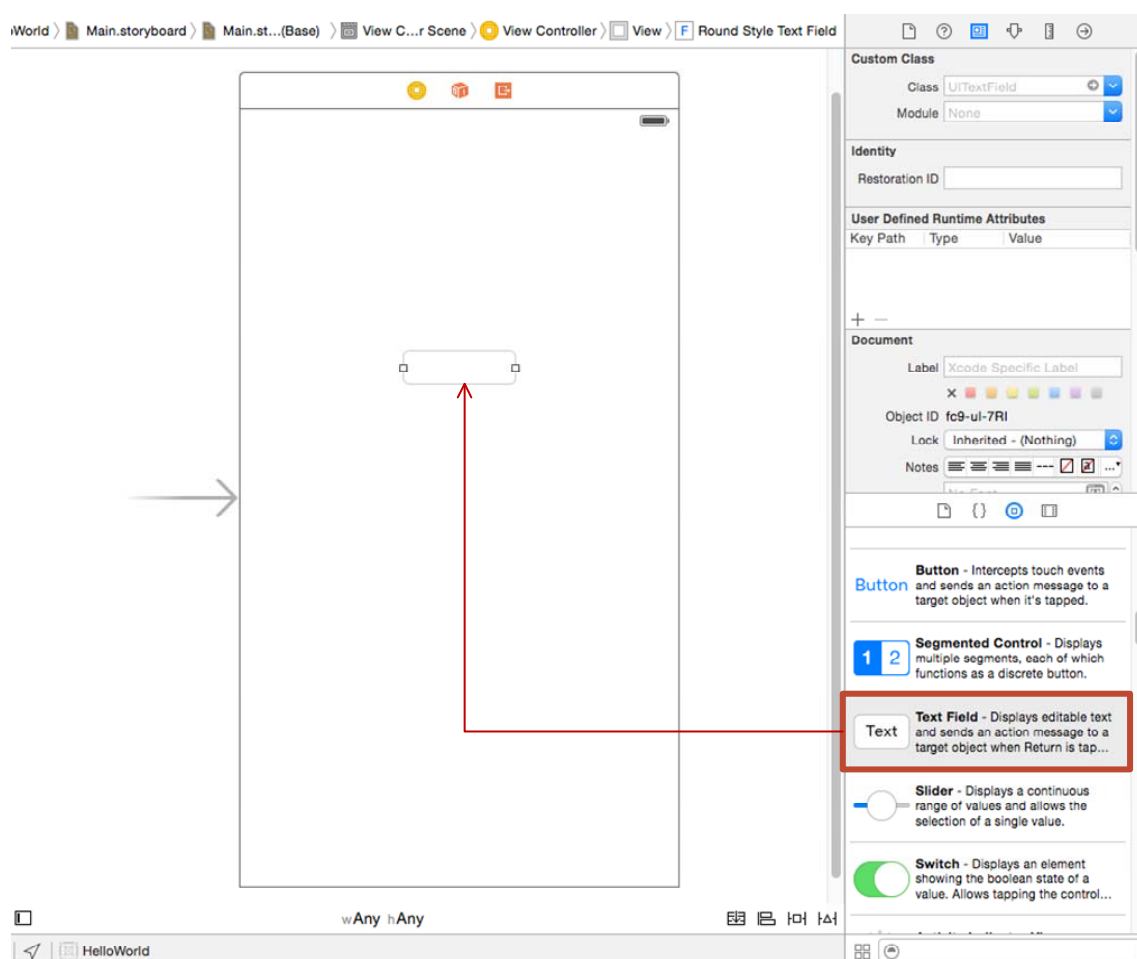


图 1.27 添加视图对象

此时运行程序，会看到如图 1.28 所示的效果。轻拍模拟器中的文本框就会出现键盘，可以通过键盘来实现字符串的输入，如图 1.29 所示。



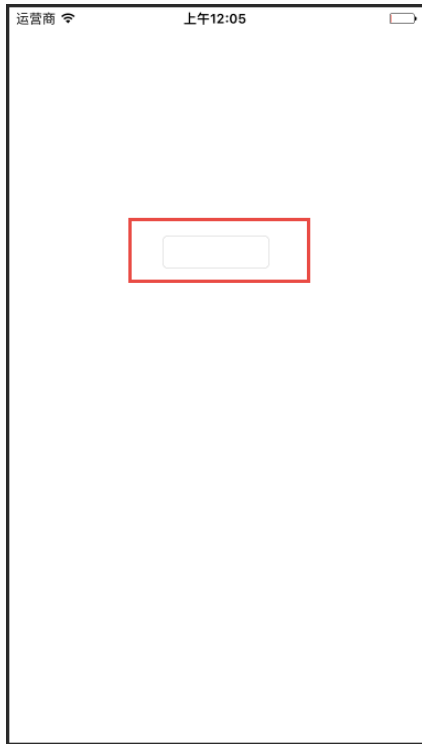


图 1.28 运行效果

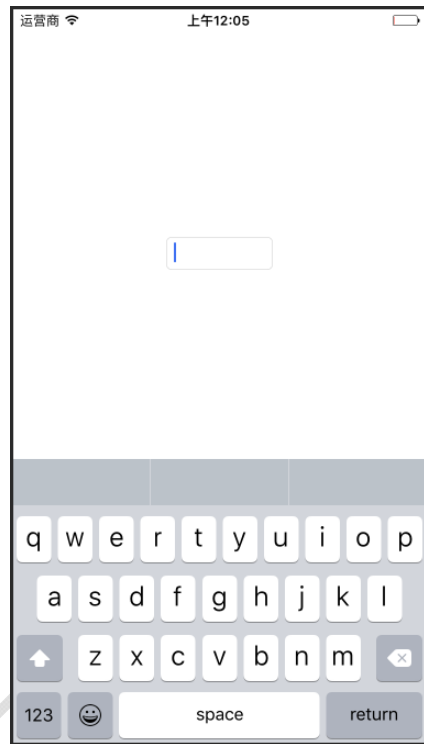


图 1.29 运行效果

### 1.2.5 编写代码

代码就是用来实现某一特定的功能，而用计算机语言编写的命令序列的集合。现在就来通过代码在文本框中实现显示“Hello,World”字符串的功能，具体的操作步骤如下：

(1) 使用设置编辑器的三个视图方式的图标，如图 1.30 所示，将 Xcode 的界面调整为如图 1.31 所示的效果。



图 1.30 编辑器的三个视图方式的图标

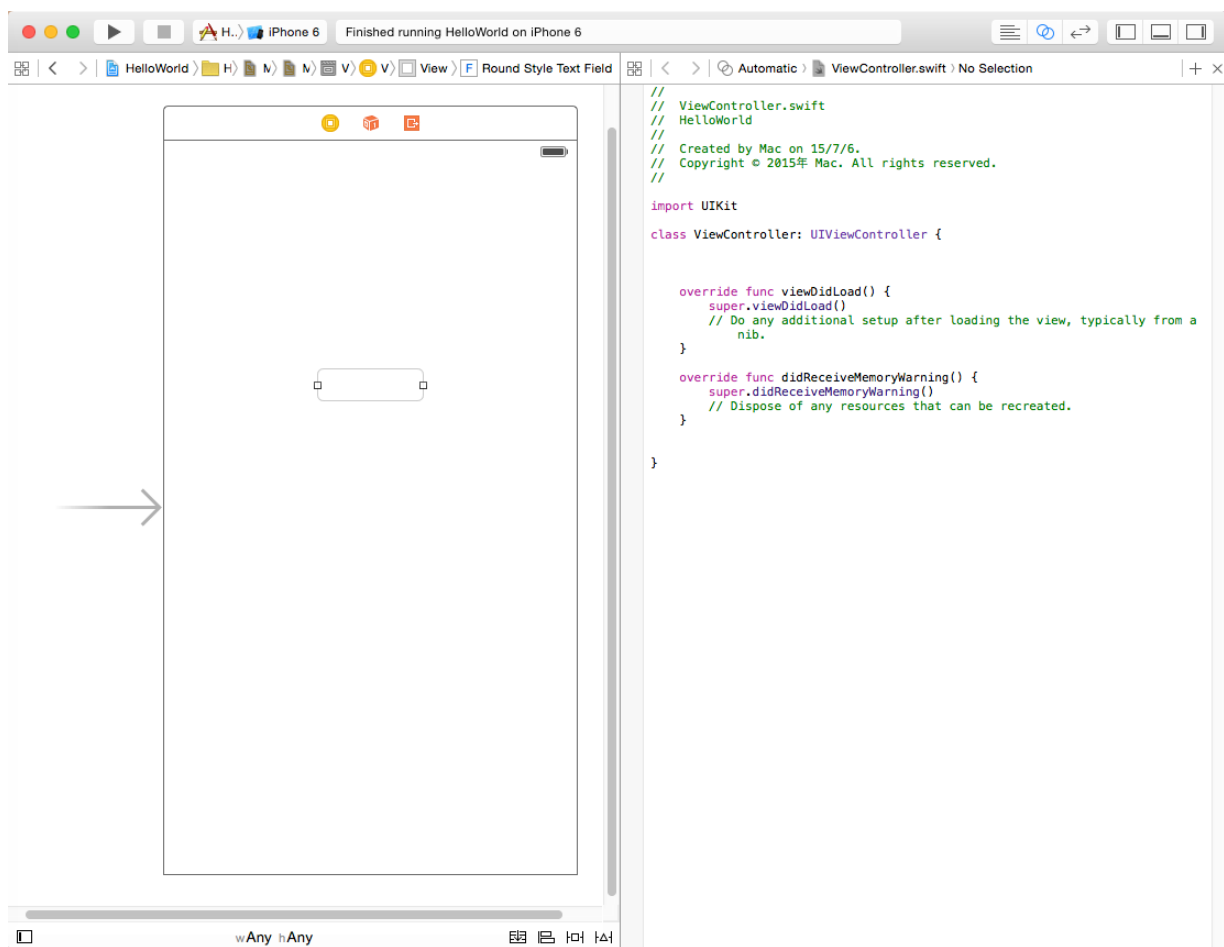


图 1.31 调整界面

(2) 按住 Ctrl 键拖动主视图中的文本框对象，这时会出现一个蓝色的线条，将这个蓝色的线条拖动到 ViewController.swift 文件中，如图 1.32 所示。

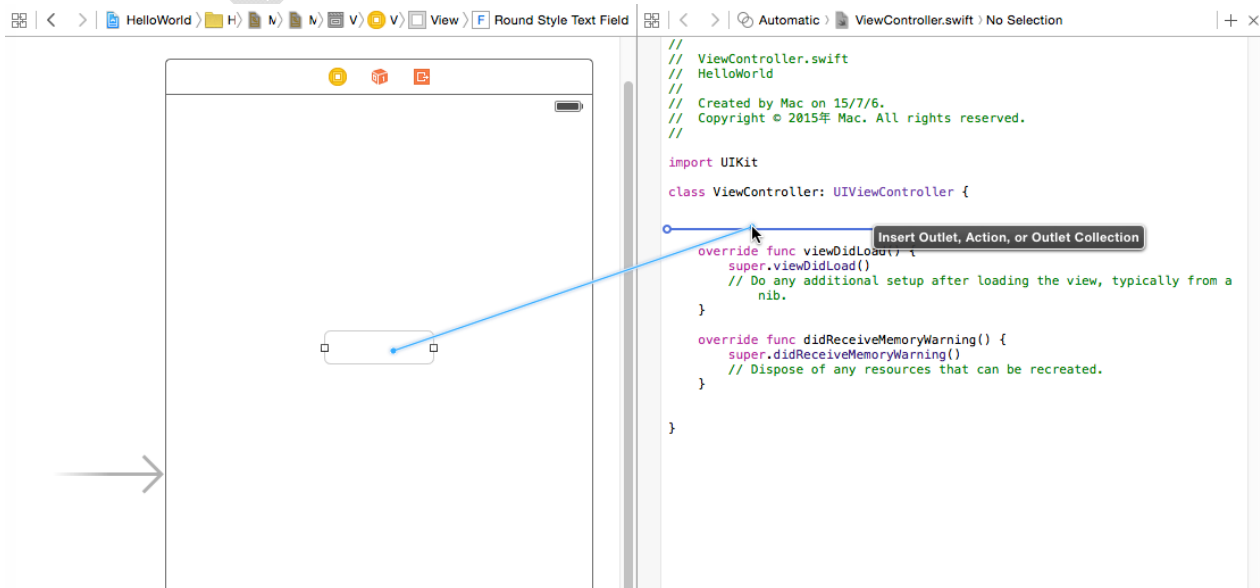


图 1.32 出现蓝色的线条

(3) 松开鼠标后，会弹出一个对话框，如图 1.33 所示。

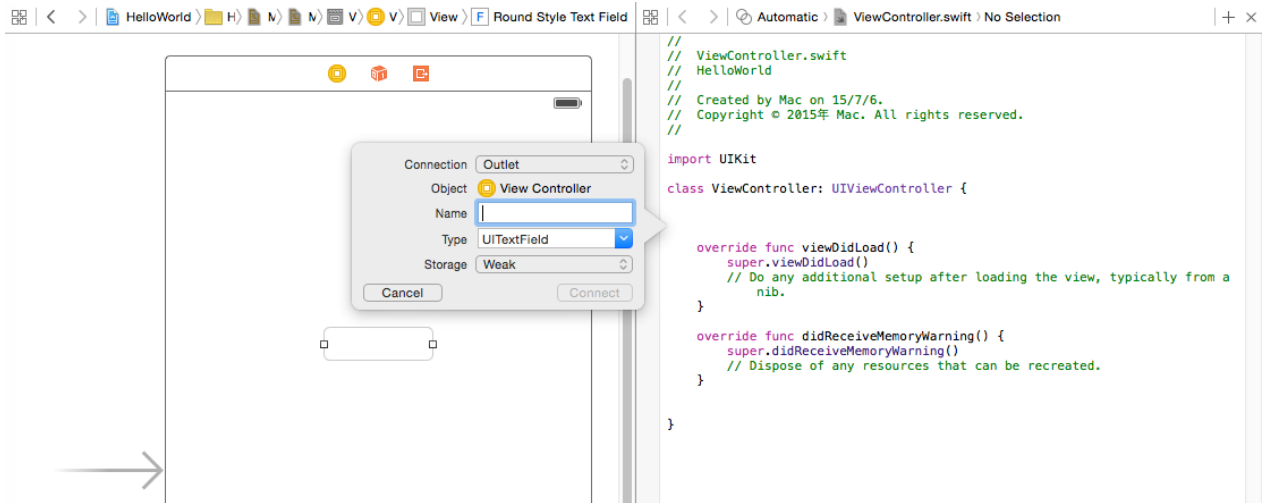


图 1.33 弹出对话框

(4) 弹出的对话框中，找到 Name 这一项，在其中输入名称 tf，如图 1.34 所示。

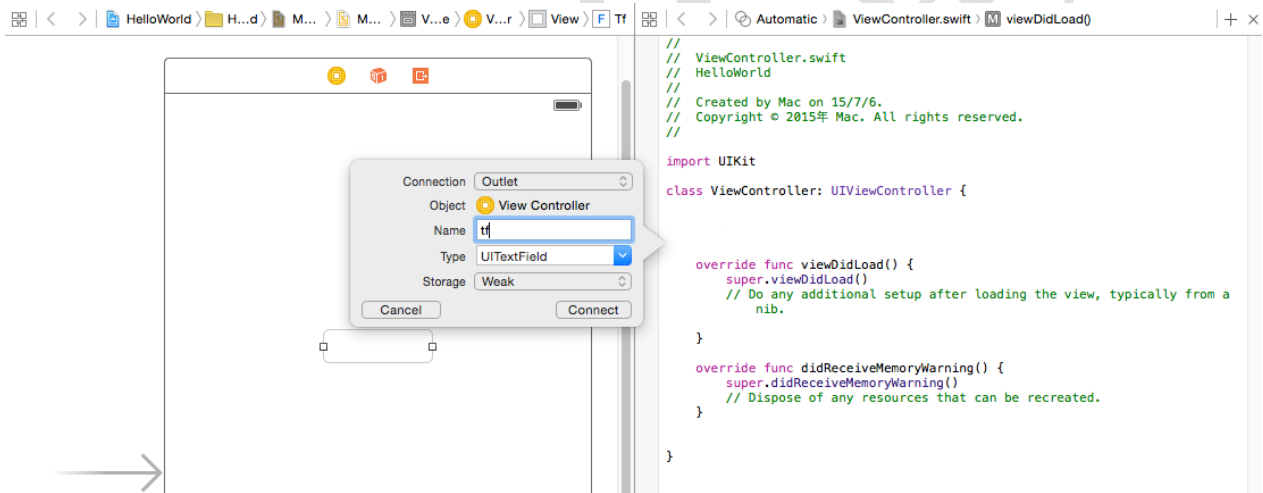


图 1.34 输入名称

注意：Name 这一项输入的名称是任意的。

(5) 选择 Connect 按钮，关闭对话框，这时在 ViewController.swift 文件中自动生成一行代码，如图 1.35 所示。

```
// ViewController.swift
// HelloWorld
// Created by Mac on 15/7/6.
// Copyright © 2015年 Mac. All rights reserved.

import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var tf: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```

插座变量

图 1.35 操作变量

注意：生成的代码叫做插座变量，插座变量其实就是为关联的对象起了一个别名。开发者就可以对此插座变量进行操作，从而对关联的对象进行操作。以上这一种方式是插座变量声明和关联一起进行的，还有一种先声明动作后关联的方式。具体操作步骤如下：

首先，打开 ViewController.swift 文件，使用 IBOutlet 关键字对文本框的插座变量进行声明，其代码如图 1.36 所示。

```
import UIKit

class ViewController: UIViewController {

    @IBOutlet var tf:UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```

声明插座变量

图 1.36 声明插座变量

注意：声明好的插座变量会在代码的前面出现一个空心的小圆圈。此小圆圈表示该插座变量还未进行关联。

其次，使用设置编辑器的三个视图方式的图标，将 Xcode 的界面进行调整，将其调整为和图 1.31 一样的效果。

然后，按住 Ctrl 键拖动主视图中的文本框对象，这时会出现一个蓝色的线条，将这个蓝色的线条和文件 ViewController.swift 文件中的插座变量进行关联，如图 1.37 所示。

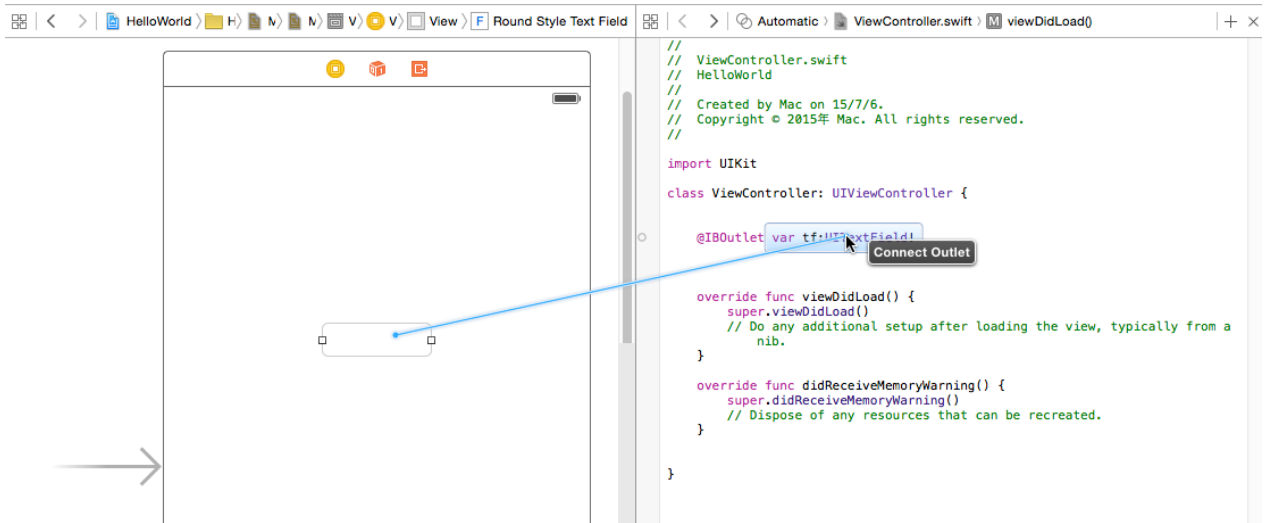


图 1.37 关联插座变量

最后松开鼠标后，文本框对象就与插座变量成功的关联在一起了，此时插座变量前面的空心小圆圈就变为了实心的小圆圈，它表示此插座变量已被关联。

(6) 打开 ViewController.swift 文件，编写代码，此代码实现的功能是在文本框中显示字符串 Hello, World。代码如下：

```
import UIKit
class ViewController: UIViewController {
    @IBOutlet weak var tf: UITextField!
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        tf.text="Hello,World"           //设置文本内容
    }
    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```

注意：为了方便开发者更好的理解代码，本书会将多余的代码省去，使用“……”省略号表示，以上的代码就会变为如下的代码：

```
import UIKit
class ViewController: UIViewController {
    @IBOutlet weak var tf: UITextField!
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        tf.text="Hello,World"           //设置文本内容
    }
    .....
}
```

此时运行程序，会看到如图 1.38 所示的效果。

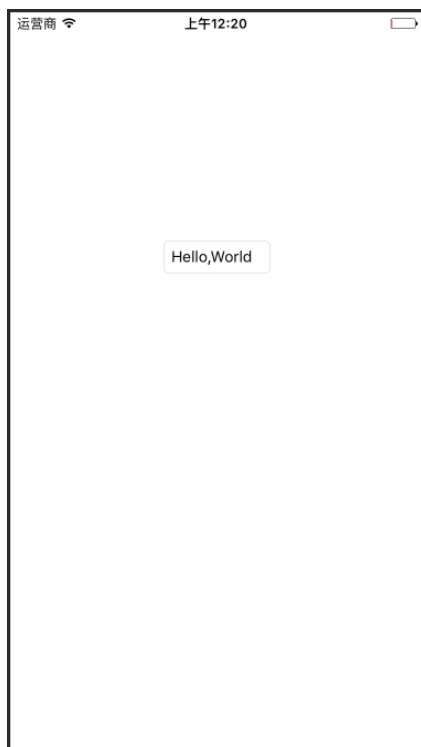


图 1.38 运行效果

## 1.2.6 定制应用程序图标

在图 1.12 中可以看到应用程序的图标是网状白色图像，它是 iOS 模拟器上的应用程序默认的图标。这个图标是可以进行改变的。以下就来实现在 iOS 模拟器上将 HelloWorld 应用程序的图标进行更改。

(1) 添加图像 logo.png 到创建的项目中，添加图像的具体步骤如下。首先右击项目文件夹中的任意位置，弹出快捷菜单，如图 1.39 所示。

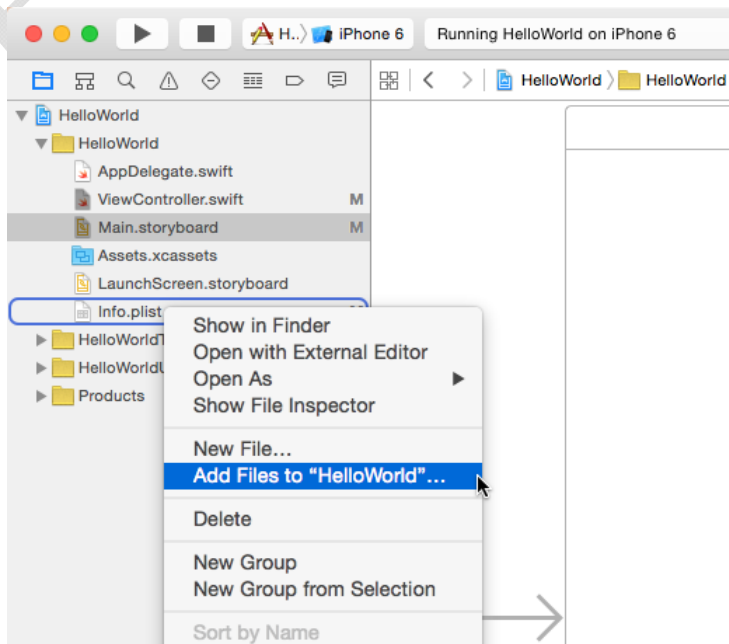


图 1.39 快捷菜单

然后，选择 Add Files to "HelloWorld"...命令，弹出选择文件对话框，如图 1.40 所示。

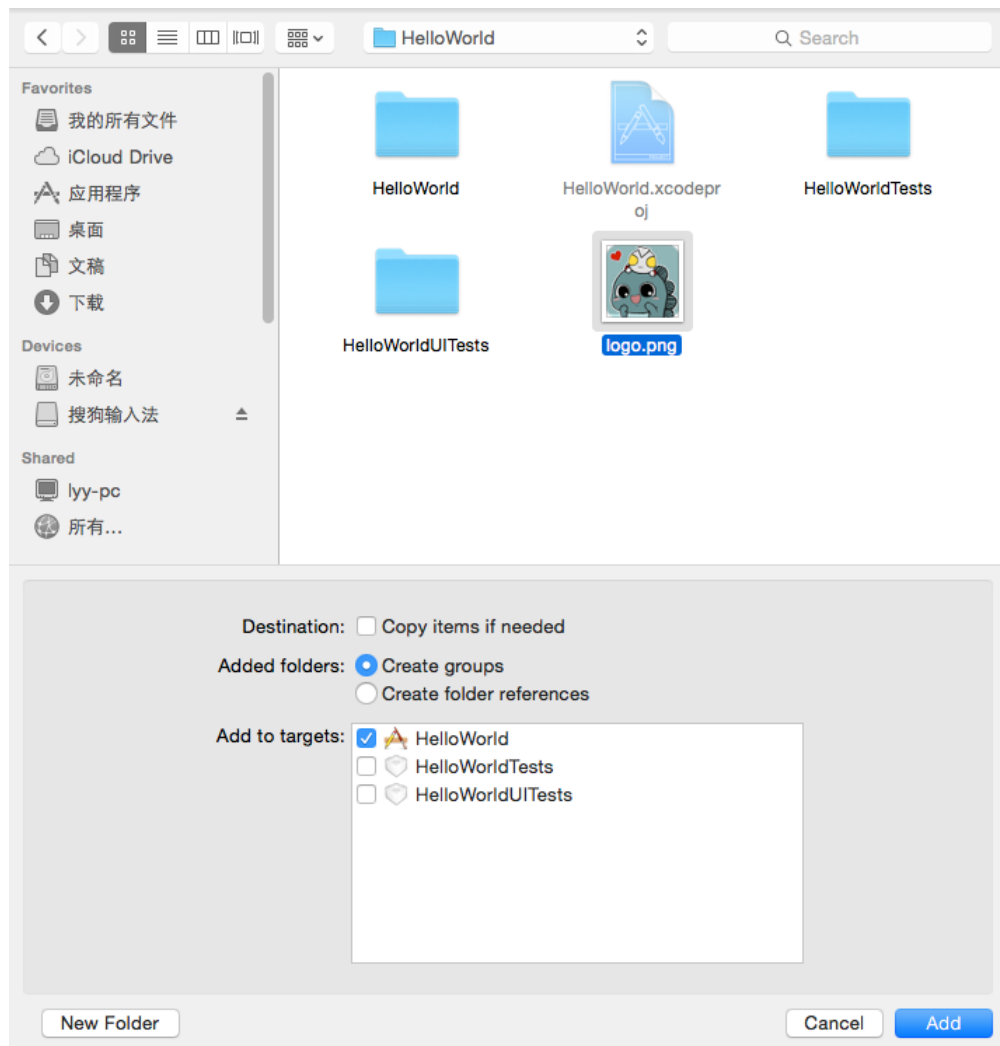


图 1.40 选择文件对话框

最后，选择需要添加的图像，单击 Add 按钮，实现图像的添加。添加后的图像就会显示在项目文件夹中。

注意：除了上面介绍的添加图像的方法外，还有一种拖动的方法。具体步骤如下：

打开 HelloWorld 项目和 logo.png 所在的文件夹，如图 1.41 所示。

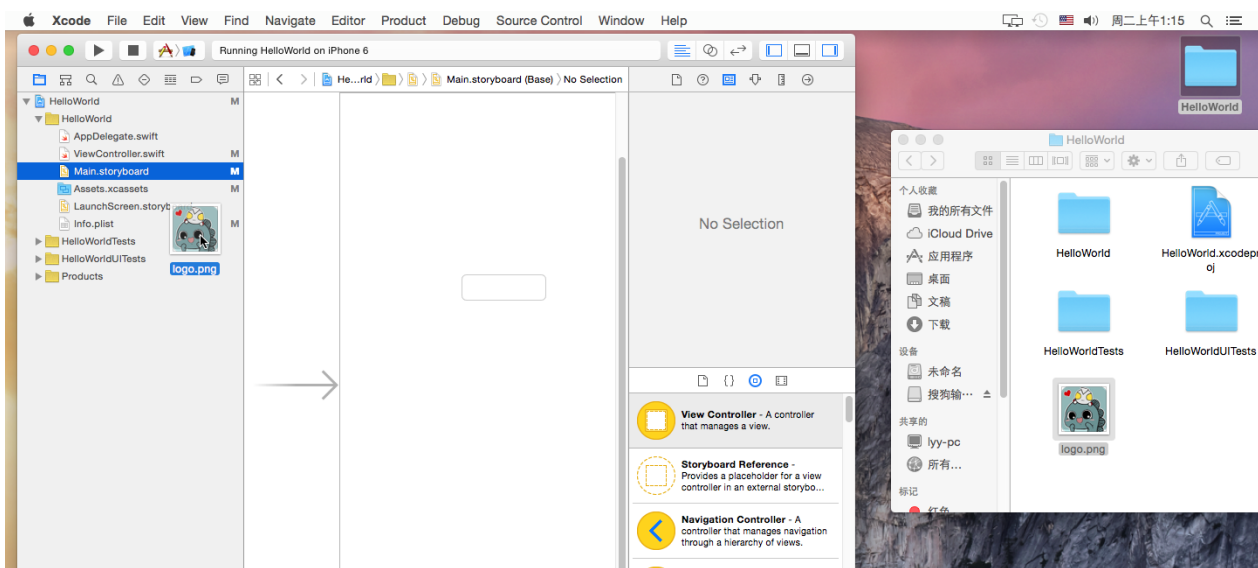


图 1.41 打开文件夹

然后，拖动 logo.png 图像到 HelloWorld 项目的项目文件夹中，松开鼠标，弹出 Choose options for adding these files:对话框，如图 1.42 所示。

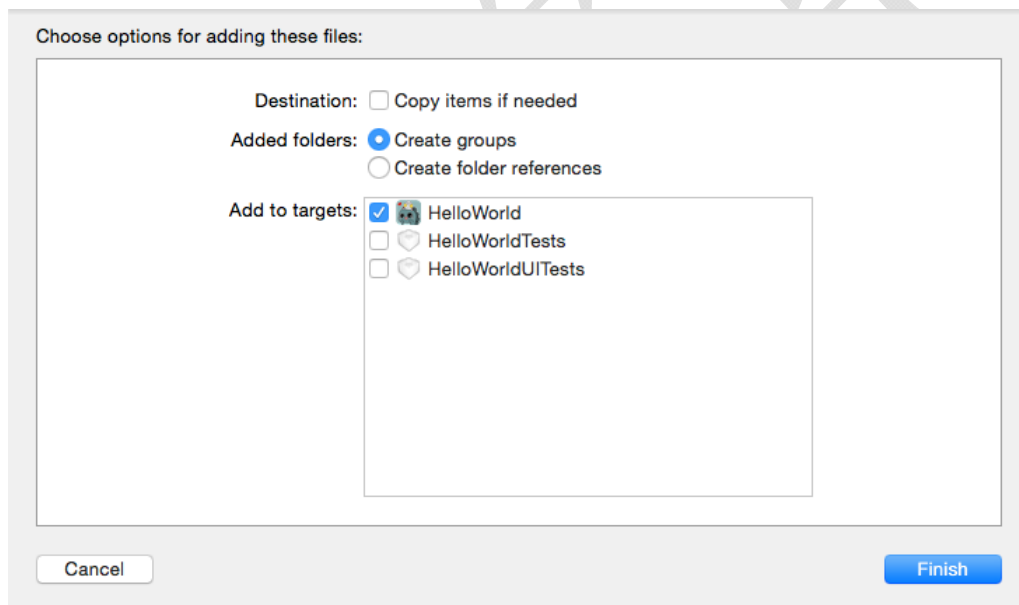


图 1.42 Choose options for adding these files:对话框

最后，单击 Finish 按钮，图像 logo.png 就被添加到 HelloWorld 项目的项目文件夹中了。

注意：iOS 9 的图标大小必须是 120\*120 像素的。

(2) 单击打开项目文件夹中的 Info.plist 文件，在其中添加一项 Icon files，在其下拉菜单的 Value 中输入添加到项目文件夹中的图片，如图 1.43 所示。



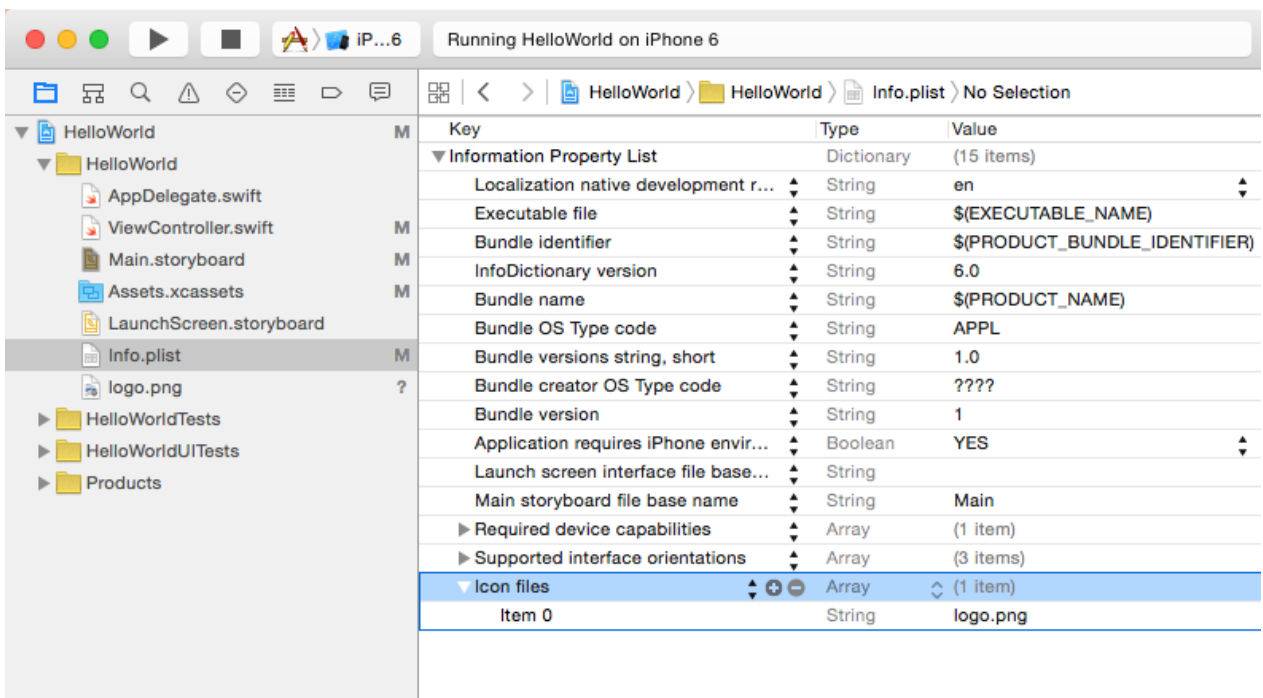


图 1.43 Info.plist 文件

此时运行程序，在返回 iOS 模拟器的主界面后，会看到如图 1.44 所示的效果。



图 1.44 运行效果

## 1.2.7 真机测试

在 Xcode 7.0 中，苹果公司在开发许可权限上做了很多的改变，在测试 app 方面取消了一些限制。在 Xcode7 之前的版本，苹果公司只向注册过的开发者帐号（99 美金收费帐号）的开发者提供 Xcode 下载以及真机测试功能，但在 Xcode 7.0 中，开发者无需注册收费的开发者账号，只要开发者感兴趣就可以使用免费的 Apple ID 在设备上免费测试 app。

接下来我们讲解一下如何在打开的 Xcode 7.0 中进行真机测试，首先选择菜单栏上的 Product|Destination|真机（本书中的真机为“Mac”的 iPhone）命令，如图 1.45 所示。然后，再一次运行程序，程序就会显示在真机上，而非 iOS 模拟器中。

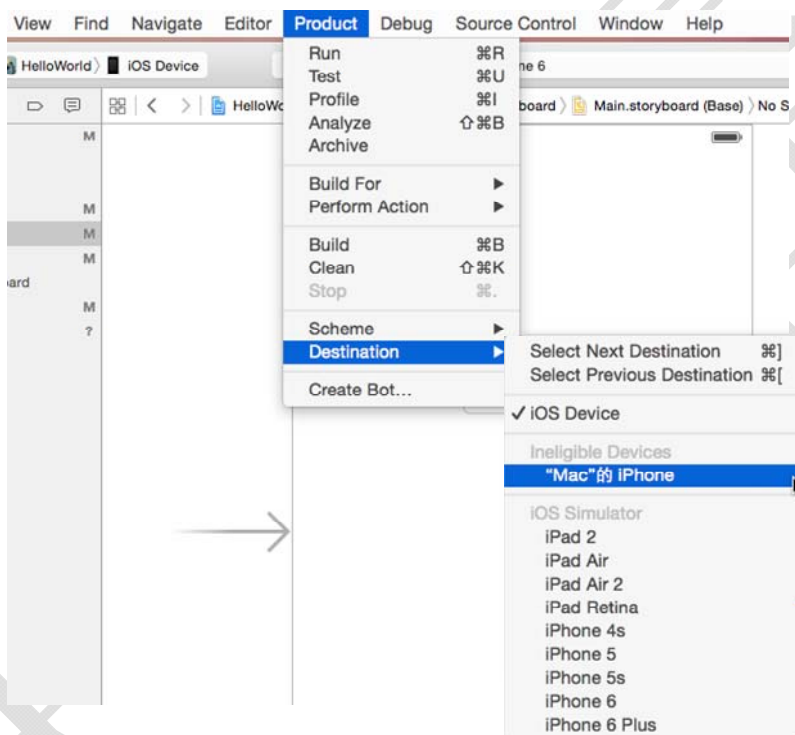


图 1.45 选择设备

## 1.3 了解视图

在 iPhone 或者 iPad 中，用户看到的和摸到的都是视图。视图是用户界面的重要组成元素。本节将主要讲解视图的添加、删除以及位置和大小等的设置内容。

### 1.3.1 视图库介绍

在视图库中存放了 iOS 开发中所需的所有视图。开发者可以在创建好 iOS 应用程序的项目以后，打开画布的设计界面，这时在工具窗口的下半个窗口中，单击 Show the Object library 图标，就会显示出视图库，如图 1.46 所示。



图 1.46 视图库

在视图库中存放的视图是可以根据功能的不同进行分类的，如表 1-3 所示。

表 1-3 视图库分类

名称	功能
Controls（控件）	用于接收用户输入的信息
Data View（数据视图）	用于显示信息
Gesture Recognizers（手势识别器）	用于识别轻击、轻扫、旋转和捏合
Objects&Controllers（控制器）	用于控制其它视图
Arranges View（部署视图）	用于对视图进行部署
Effect View（效果视图）	提供一个模糊效果
Windows&Bars（其它）	用于显示其它各种视图

在视图库的最下边有两个图标，一个是用来进行显示视图排列方式的，一个是搜索视图的，如图 1.47 所示。

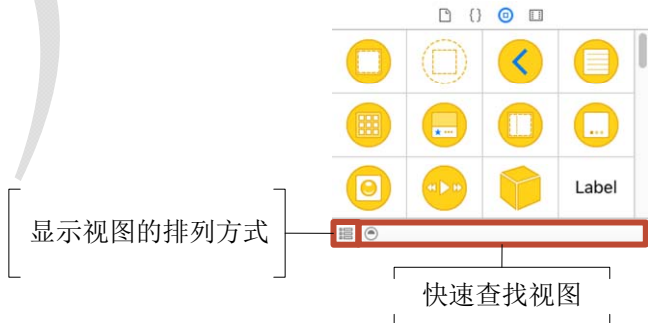


图 1.47 视图库

### 1.3.2 视图始祖——UIView

在 Swift 中，NSObject 是所有类的根类。同样在 UIKit 框架（UIKit 框架为 iOS 应用程序提供界面对象和控制器）中，也存在一个如此神奇的类 UIView。从继承关系上看，UIView 是所有视图的根。一般称 UIView 为空白视图。

### 1.3.3 添加视图

在 iOS 中添加视图的方式有两种：一种是使用编辑界面添加视图；另一种是使用代码添加视图。以下是这两个方式的详细介绍。

#### 1. 编辑界面添加视图

使用编辑界面添加视图是一个相当简单的工作，即从视图库中拖动视图到主视图中即可。

【示例 1-1】以下将实现如何使用编辑界面添加一个空白视图。具体的操作步骤如下：

(1) 创建一个 Single View Application 模板类型的项目，命名为 UIView-InterfaceBuilder。

(2) 打开 Main.storyboard 文件，单击 Show the Object library，在显示的视图库中找到 View 视图即空白视图，将其拖到画布的主视图中，如图 1.48 所示。

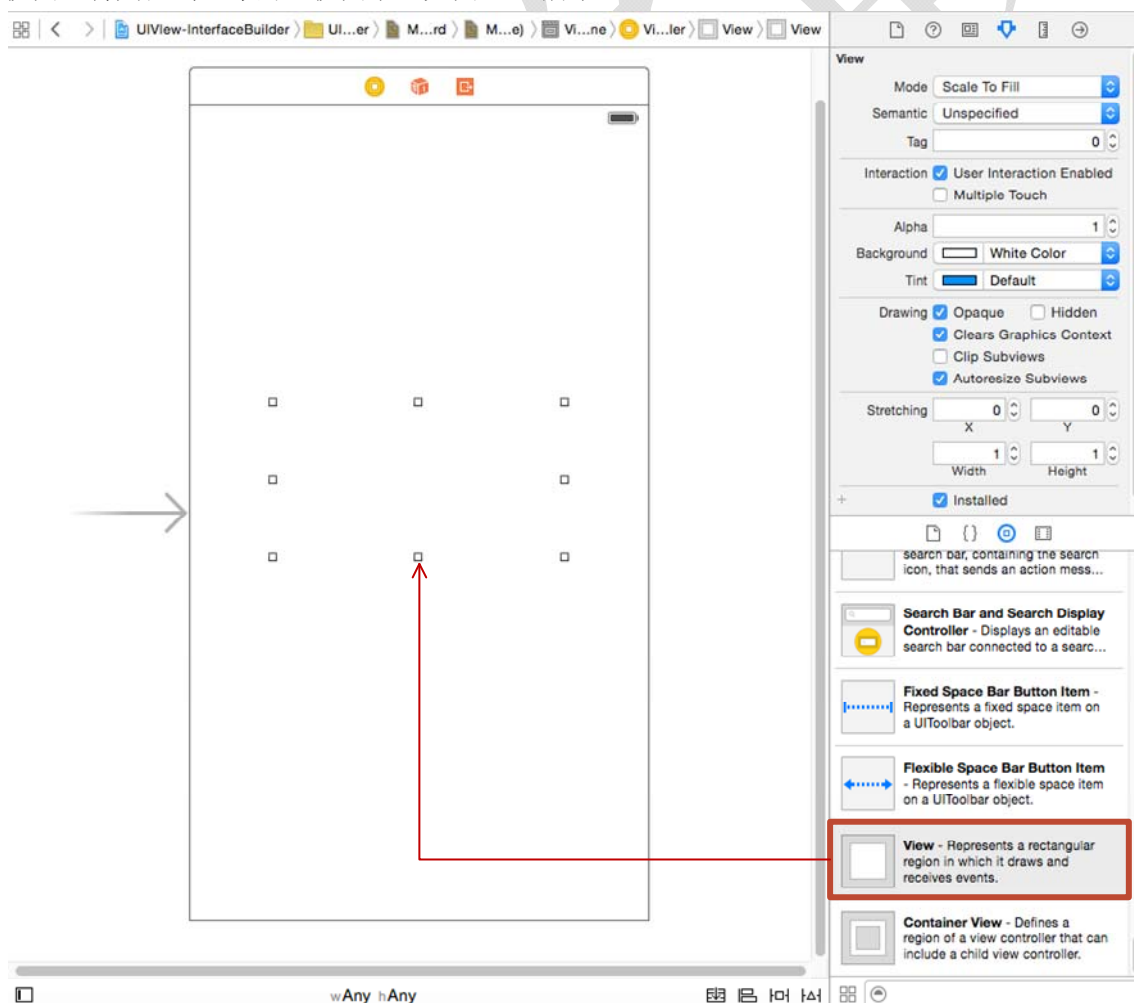


图 1.48 添加视图

此时运行程序，会看到如图 1.49 所示的效果。

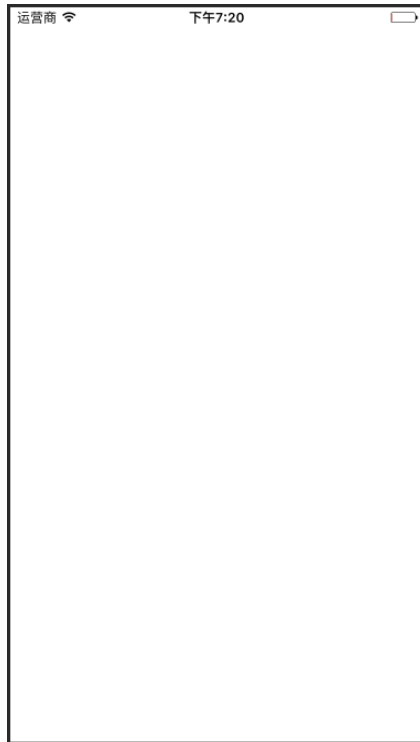


图 1.49 运行效果

由于使用编辑界面添加的 View 空白视图默认的背景颜色为白色，所以在模拟器上是看不出效果的。那么该如何在模拟器上看到添加的 View 空白视图呢？开发者需要回到 Main.storyboard 文件，选择主视图上添加的 View 空白视图。然后，选择在属性检查器面板中会打开相应的属性设置，找到 Background 属性将其设置为 Dark Gray Color，如图 1.50 所示。

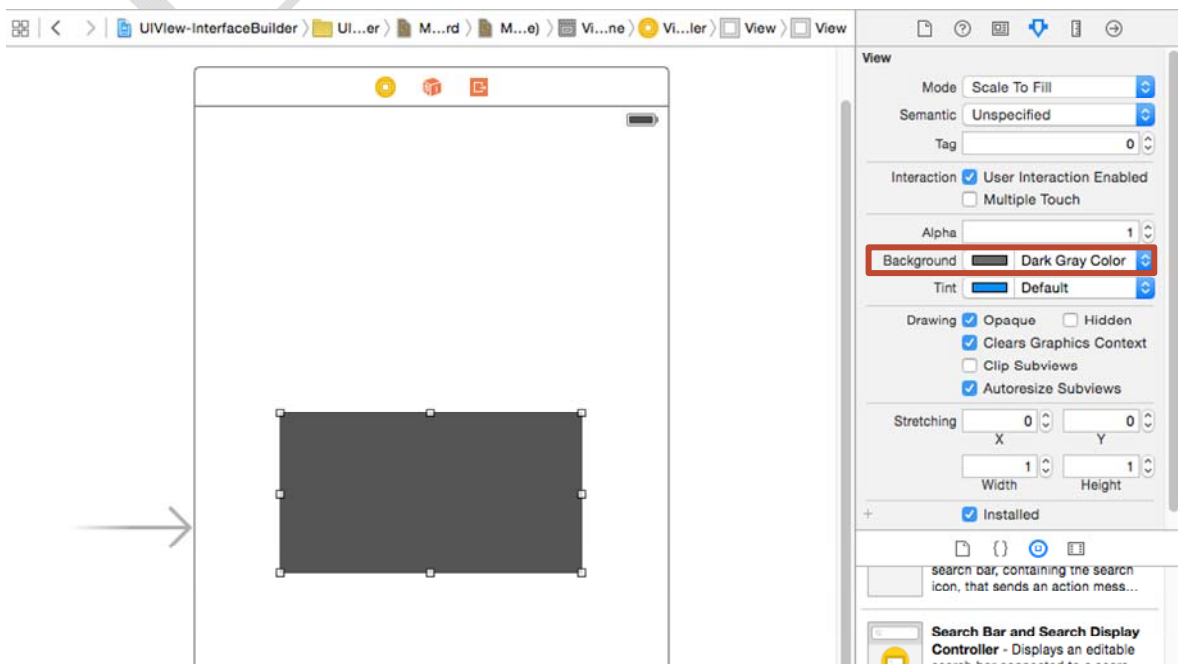


图 1.50 设置背景颜色

此时运行程序，会看到如图 1.51 所示的效果。

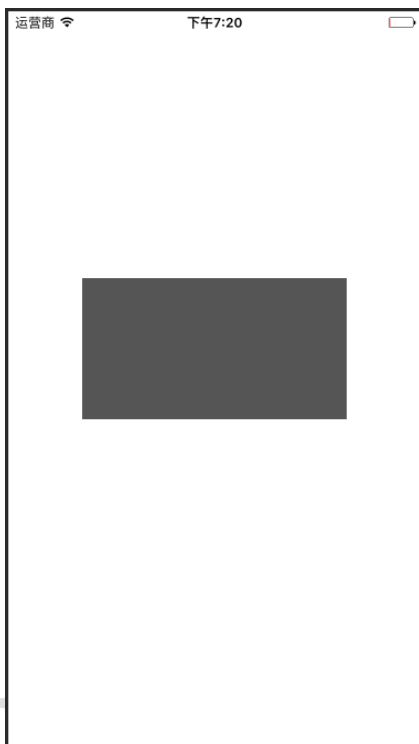


图 1.51 运行效果

注意：在属性检查器面板中存放了一些属性，当开发者单击主视图中的某一视图后，属性检查器面板中就会出现对应的属性设置，开发者可以通过这些属性对视图进行美化。

## 2. 代码添加视图

如果开发者想要使用代码为主视图添加视图，该怎么办呢。以下将为开发者解决这一问题。要使用代码为主视图添加视图需要实现 3 个步骤。

### (1) 实例化视图对象

每一个视图都是一个特定的类。在 Swift 中，经常会说，类是一个抽象的概念，而非具体的事物，所以要将类进行实例化。实例化一个视图对象的具体语法如下：

```
let/var 对象名=视图类()
```

以我们接触的第一个视图 View 为例，它的实例化对象如下：

```
let newView=UIView()
```

其中，UIView 是空白视图的类，newView 是 UIView 类实例化出来的一个对象。

### (2) 设置视图的位置和大小

每一个视图都是一个区域，所以需要为此区域设置位置和大小。设置位置和大小属性为 frame，其语法形式如下：

```
对象名 frame=CGRectMake (X ,Y ,Width,Height)
```

其中，X 和 Y 表示视图在主视图中的位置，Width 和 Height 表示视图的大小。以下为实例化的对象 newView 设置位置和大小：

```
newView.frame=CGRectMake(67, 264, 240, 128)
```

其中，67 和 264 表示此视图在主视图中的位置，240 和 128 表示此视图的大小。

注意：步骤 1 和步骤 2 也可以进行合并。例如，以下的代码是将 UIView 类的实例化对象和设置位置大小进行了合并：

```
let newView=UIView(frame: CGRectMake(67, 264, 240, 128))
```

(3) 将视图添加到当前的视图中

最后，也是最为关键的一步，就是将实例化的对象添加到主视图中。这样才可以进行显示。此时需要使用到 addSubview() 方法，其语法形式如下：

```
this.view.addSubview (视图对象名)
```

以下将实例化的对象 newView 添加到当前的主视图中，代码如下：

```
self.view.addSubview(newView)
```

【示例 1-2】以下将使用代码为主视图添加一个 View 空白视图。代码如下：

```
import UIKit
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        let newView=UIView(frame: CGRectMake(67, 264, 240, 128))
        self.view.addSubview(newView)
    }
    .....
}
```

此时运行程序，会看到如图 1.52 所示的效果。在此运行效果中也是看不到添加的视图的。这是因为添加的视图默认是白色的背景，如果想要看到视图，需要设置它的背景。例如以下的代码，将背景颜色设置为了灰色：

```
newView.backgroundColor=UIColor.grayColor()
```

此时运行程序，会看到如图 1.53 所示的效果。

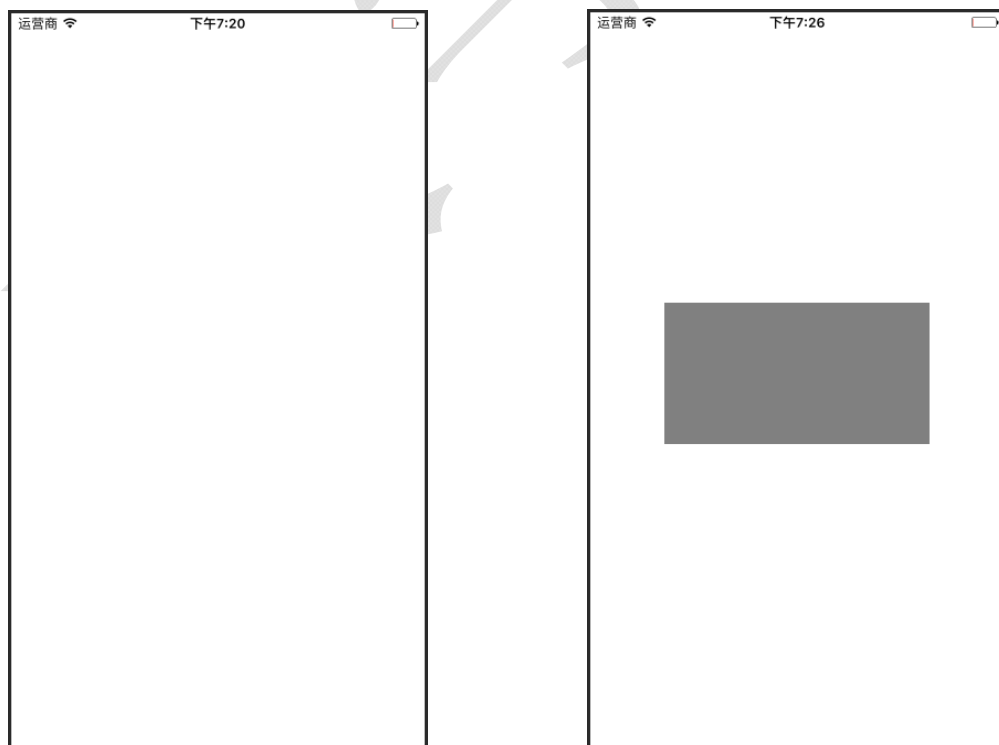


图 1.52 运行效果

图 1.53 运行效果

### 1.3.4 视图的位置和大小

当一个视图使用拖动的方式添加到主视图后，它的位置和大小可以使用拖动的方式进行设置，也可以使用尺寸检查器面板中的内容进行设置，如图 1.54 所示。

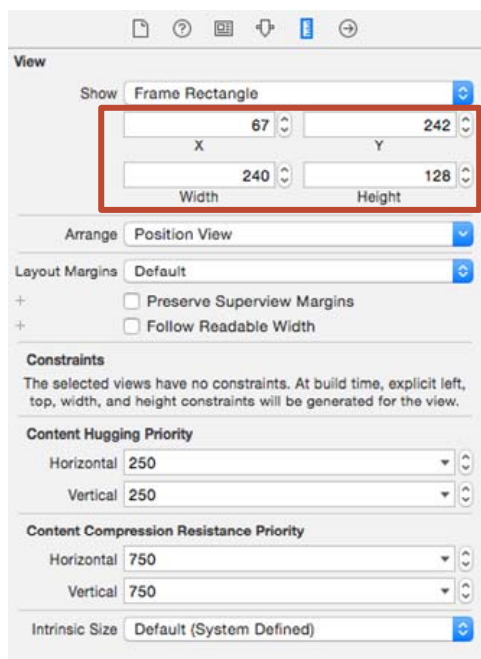


图 1.54 位置尺寸设置

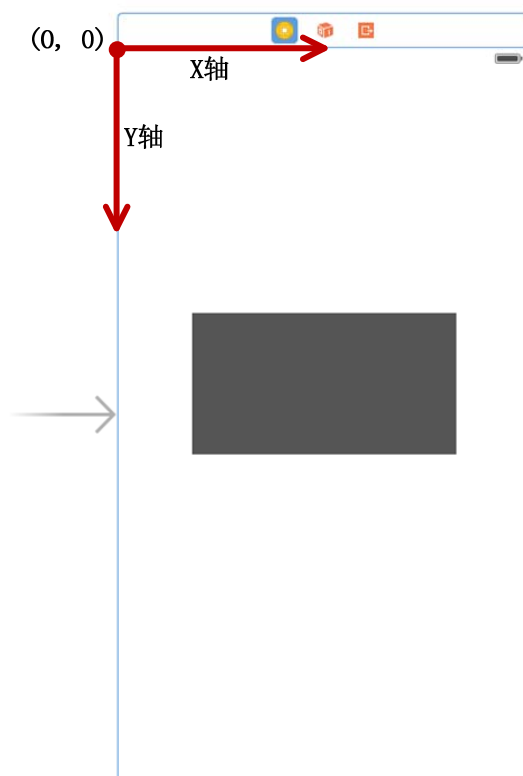


图 1.55 坐标

注意：在默认的情况下，坐标系统的原点位于左上角，并向底部和右侧延伸，如图 1.55 所示。

### 1.3.5 删除空白视图

当开发者不再需要主视图的某一视图时，可以将该视图删除。实现此功能需要使用到 `removeFromSuperview()` 方法，其语法形式如下：

要删除的视图对象名.`removeFromSuperview()`

【示例 1-3】以下代码将在主视图中添加两个视图，然后再使用 `removeFromSuperview()` 方法删除其中一个视图。代码如下：

```
import UIKit
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        //添加空白视图 newView1
    }
}
```



```
let newView1=UIView(frame: CGRectMake(0, 75, 375, 232))
newView1.backgroundColor=UIColor.cyanColor()
self.view.addSubview(newView1)
//添加空白视图 newView2
let newView2=UIView(frame: CGRectMake(0, 352, 375, 232))
newView2.backgroundColor=UIColor.orangeColor()
self.view.addSubview(newView2)
}
.....
}
```

此时运行程序，会看到如图 1.56 所示的效果。如果想要删除视图对象 newView1 的话，需要使用 removeFromSuperview() 方法，代码如下：

```
newView1.removeFromSuperview() //删除视图对象 newView1
```

运行效果如图 1.57 所示。

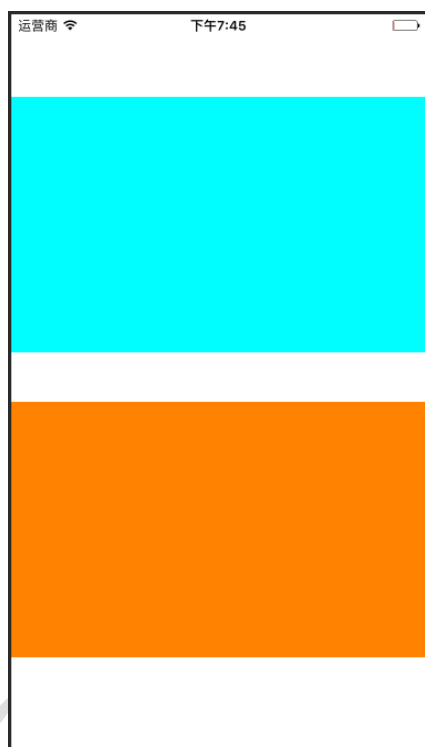


图 1.56 运行效果



图 1.57 运行效果

## 第 2 章 丰富的用户界面

在 iOS 中提供了很多的控件以及视图来丰富用户界面，对于这些视图以及控件我们在上一章中做了简单的介绍。本章我们将详细讲解这些视图。

### 2.1 使用按钮接收用户输入

按钮是 iOS 应用中最常使用也是最简单的控件，它常用来响应用户的点击事件，如图 2.1 所示。在图 2.1 中，蓝色的矩形就是一个按钮，它的标题为“登录”。在 iOS 7 以后按钮只是一块普通的文本，没有轮廓，边框，背景颜色，或其他装饰功能（为了美观，很多的应用程序中的按钮还是有背景的，就像图 2.1 中的按钮）。一般使用 `UIButton` 类来实现按钮。本节将主要讲解按钮的添加、美化按钮以及如何实现按钮的响应等内容。



图 2.1 QQ 登录界面

#### 2.1.1 使用代码添加按钮

由于使用编辑界面添加视图的方式比较简单，所以不在介绍。这里，直接讲解代码中如何添加。使用代码为主视图添加一个按钮的方式和在 1.3.3 节中讲解的步骤是一样的。首先需要使用 `UIButton` 类实例化一个按钮对象，然后是设置位置和大小，最后是使用 `addSubview()` 方法将按钮对象添加到主视图中。（由于视图的添加方式都一样，后面将省略使用代码添加视图这块内容。）。

【示例 2-1】以下将为主视图添加一个背景颜色为橘黄色的按钮对象。代码如下：

```
import UIKit
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        //添加按钮对象
        let button=UIButton(frame: CGRectMake(143, 241, 88, 30))
        button.backgroundColor=UIColor.orangeColor()
        self.view.addSubview(button)
    }
    .....
}
```

此时运行程序，会看到如图 2.2 所示的效果。

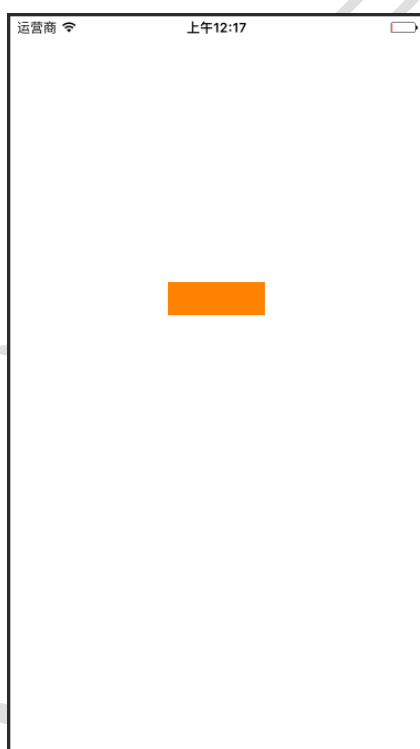


图 2.2 运行效果

注意：在图 2.2 中所显示的橘黄色区域其实就是添加的按钮。

## 2.1.2 美化按钮

美化按钮说白了就是对按钮的属性进行设置，设置按钮的属性有两种方法：一种是使用编辑界面中的属性检查器；另一种是使用代码进行设置。以下将主要讲解如何使用代码对按钮进行设置。

### 1. 设置按钮的外观

设置按钮的外观其实就是对按钮的标题、图像等进行的设置。表 2-1 列出了常用的一些设置按钮外观的属性。

表 2-1 常用属性

属性	功能
setBackgroundImage	设置指定状态下按钮的背景图像
setImage	设置指定状态下按钮的图像
setTitle	设置指定状态下按钮的标题
setTitleColor	设置指定状态下按钮的标题颜色
setTitleShadowColor	设置指定状态下按钮标题的应用

【示例 2-2】下面将在主视图中添加一个按钮。此按钮的标题为 I am button，标题的颜色为黑色。代码如下：

```
import UIKit
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        //添加按钮对象
        let button=UIButton(frame: CGRectMake(135, 241, 97, 30))
        button.setTitle("I am Button", forState: UIControlState.Normal)           //设置标题
        button.setTitleColor (UIColor.blackColor(), forState: UIControlState.Normal) //设置标题颜色
        self.view.addSubview(button)
    }
    .....
}
```

此时运行程序，会看到如图 2.3 所示的效果。

## 2. 设置按钮的状态

在示例 2-2 中，设置按钮的标题和颜色时，需要对按钮的状态进行设置，表示按钮在某一状态下的标题和标题颜色是什么样子。例如，UIControlState.Normal 就表示按钮的一种状态。对于像按钮的这类视图，即可以接受用户输入的视图也被称为控件。这些控件都有自己的状态。表 2-2 就为开发者详细介绍了控件的状态。

表 2-2 控件的状态

按钮的状态	解释说明
Normal	常规状态
Highlighted	高亮状态
Disabled	禁用状态，不接受任何事件
Selected	选中状态
Application	应用程序标志
Reserved	为内部框架预留，可以不管他

## 3. 设置按钮的类型

按钮的形式是多种多样的。例如，在通讯录中，添加新联系人的按钮是一个加号；查看来电的详细信息时是一个感叹号等。这些按钮的实现，可以在实例化按钮对象时使用 UIButtonType 来实现。UIButtonType 中的内容如表 2-3 所示。

表 2-3 UIButtonType 的内容

按钮类型	解释说明
System	默认的风格按钮
Custom	自定义风格的按钮
DetailDisclosure	蓝色感叹号按钮，主要用于详细说明
InfoLight	亮色感叹号

InfoDark	暗色感叹号
ContactAdd	十字加号按钮，此按钮通常在添加联系人条目中显示

【示例 2-3】以下将在主视图添加两个不同风格的按钮。代码如下：

```
import UIKit
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        //添加按钮对象
        let button1=UIButton(type: UIButtonType.ContactAdd)
        button1.center=CGPointMake(190, 250)
        self.view.addSubview(button1)
        //添加按钮对象
        let button2=UIButton(type: UIButtonType.DetailDisclosure)
        button2.center=CGPointMake(190, 450)
        self.view.addSubview(button2)
    }
    .....
}
```

此时运行程序，会看到如图 2.4 所示的效果。

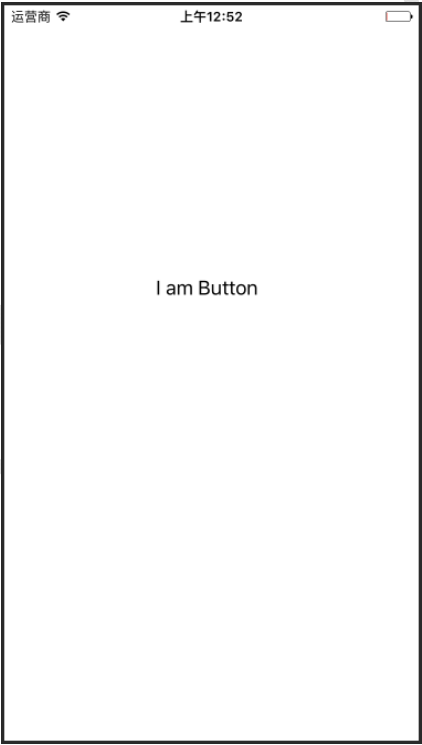


图 2.3 运行效果

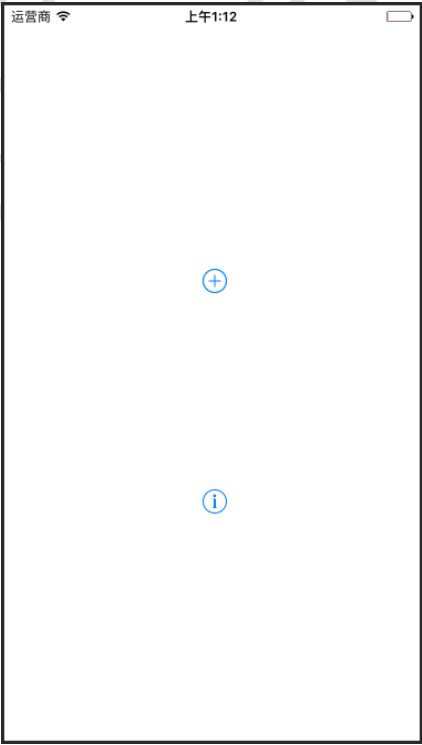


图 2.4 运行效果

2.1.3 实现按钮的响应

按钮主要是实现用户交互的，即实现响应。按钮实现响应的方式可以根据添加按钮的不同分为两种：一种是编辑界面添加按钮实现的响应；另一种是使用代码添加按钮实现的响应。

## 1. 编辑界面添加按钮实现的响应

使用编辑界面添加按钮可以使用拖动的方式来实现按钮的响应，它也是最简单的一种实现响应的方式。

【示例 2-4】以下将实现轻拍按钮，改变主视图背景颜色的功能。具体的操作步骤如下：

(1) 创建一个 Single View Application 模板类型的项目，命名为 UIButton-response。

(2) 打开 Main.storyboard 文件，将主视图的尺寸设置为 iPhone 4.7-inch。从视图库中拖动按钮控件到主视图中，将 Title 设置为 Tap me,Change View Color。

(3) 使用设置编辑器的三个视图方式的图标，将 Xcode 的界面调整为如图 2.5 所示的效果。这一过程在前面的章节中讲解过。

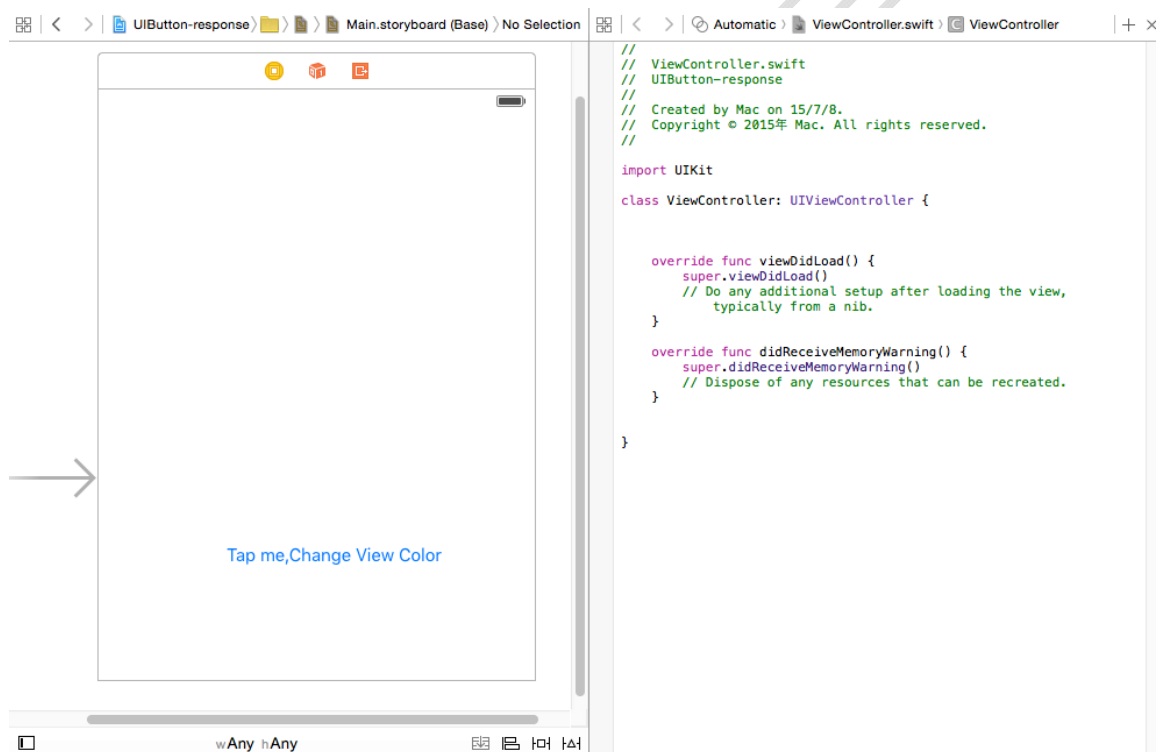


图 2.5 调整 Xcode 的界面

(4) 按住 Ctrl 键拖动界面中的按钮对象，这时会出现一个蓝色的线条，将这个蓝色的线条拖动到 ViewController.swift 文件的空白处中，如图 2.6 所示。

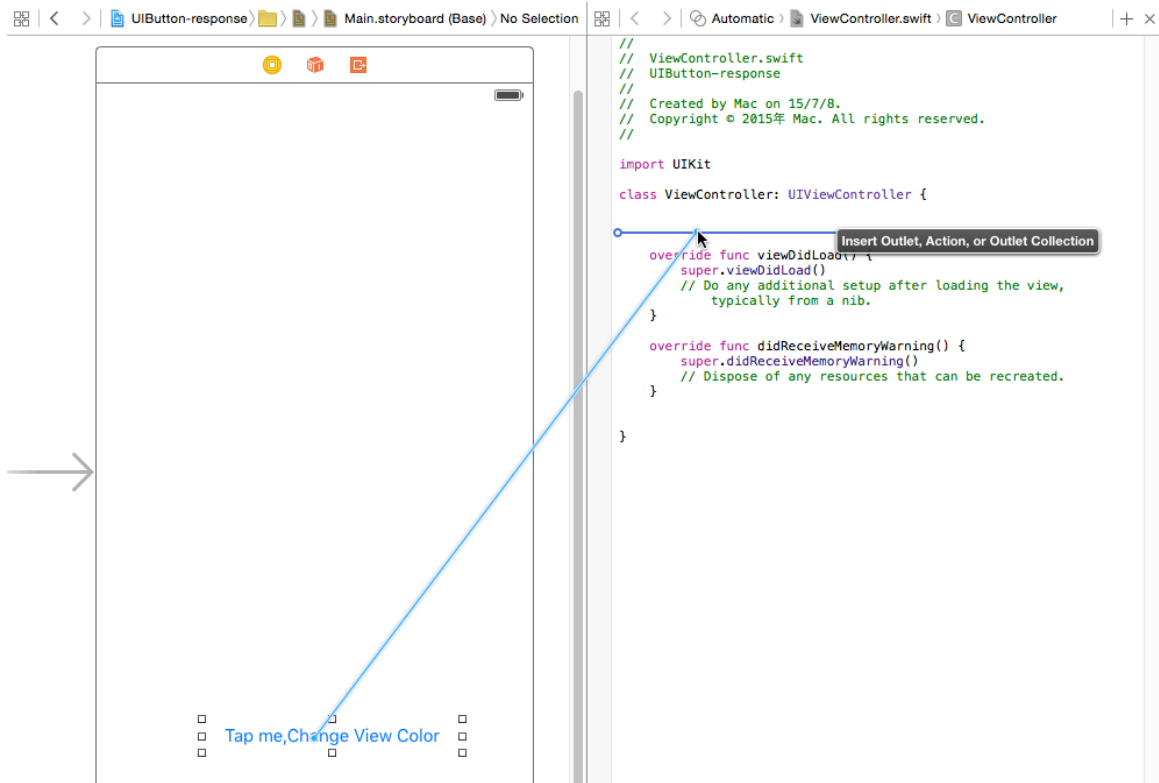


图 2.6 按住 Ctrl 键拖动界面中的按钮对象

(5) 松开鼠标后，会弹出声明关联插座变量一起进行的对话框（在前面章节中讲解过），如图 2.7 所示。

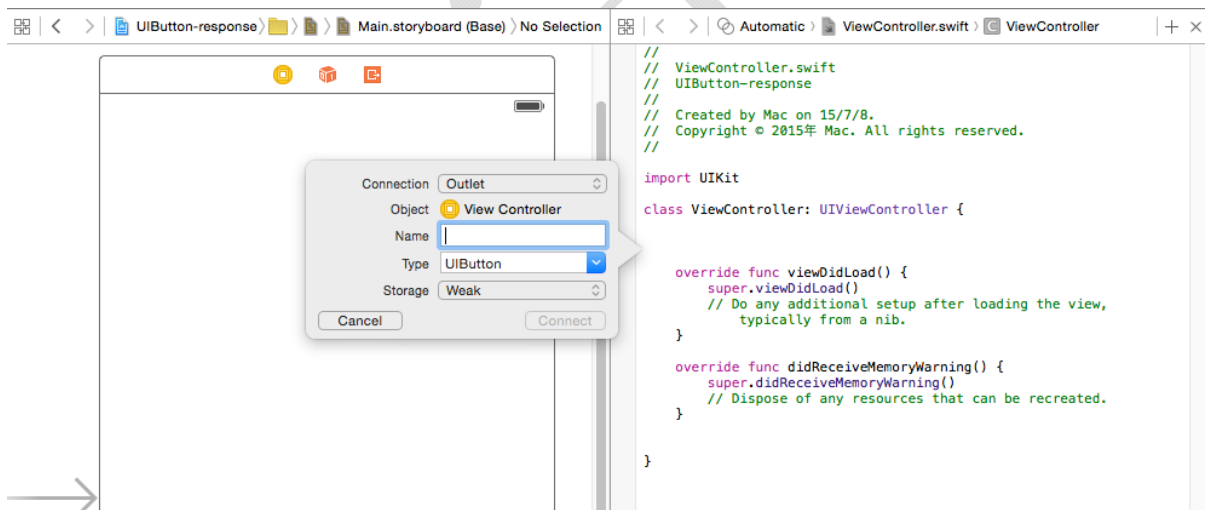


图 2.7 弹出声明关联插座变量一起进行的对话框

(6) 将 Connection 选项设置为 Action，表示关联的是一个动作；将 Name 设置为 tapButton，表示关联的动作名为 tapButton，如图 2.8 所示。

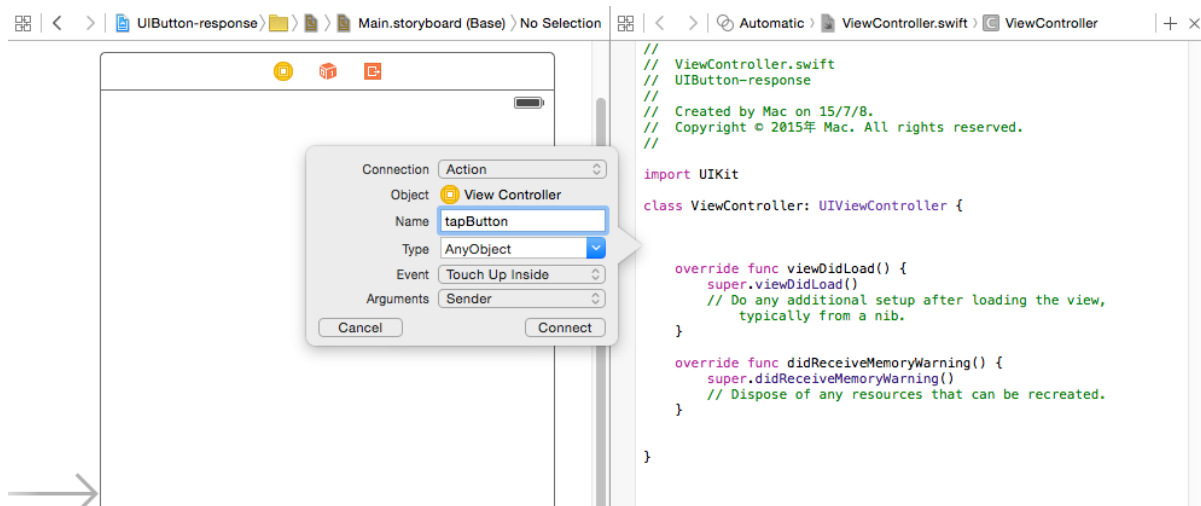


图 2.8 填写对话框

注意：这里的 Name 可以是任意的。

(7) 单击 Connect 按钮，会在 ViewController.swift 文件中看到如图 2.9 所示的代码。

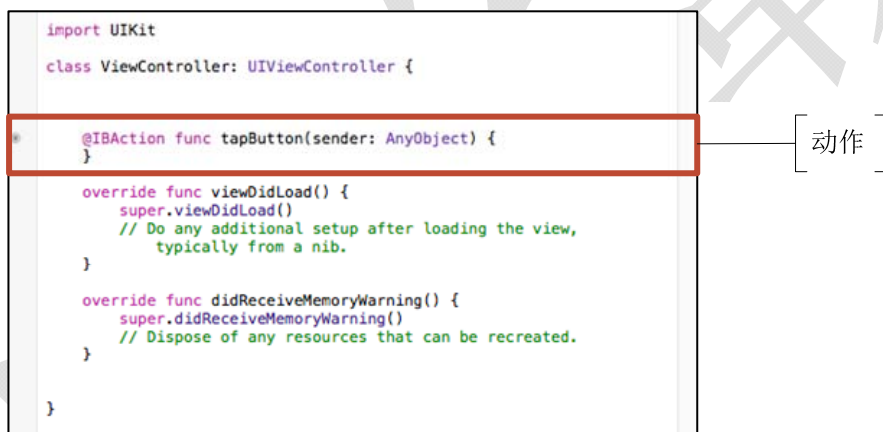


图 2.9 动作

此时，当用户轻拍按钮后，一个叫 tapButton() 的方法就会被触发。

注意：以上这一种方式是动作声明和关联一起进行的，还有一种先声明动作后关联的方式。声明动作可以使用关键字 IBAction。该关键字可以告诉故事面板的界面，此方法是一个操作，且可以被某个控件触发。声明动作的语法形式如下：

```
@IBAction func 动作名(参数:参数类型){
}
```

如图 2.10 所示，就是在 ViewController.swift 文件中编写的动作的声明代码。





图 2.10 声明的动作

注意：在声明动作后，会在代码的前面出现一个空心的小圆圈，它表示此动作还未进行关联。

声明好动作后，就可以进行关联了，首先使用调整窗口中的工具，将 Xcode 的界面进行调整。将其调整为和图 2.5 一样的效果。

然后，按住 Ctrl 键拖动界面中的按钮对象，这时会出现一个蓝色的线条，将这个蓝色的线条和文件 ViewController.swift 中的动作进行关联，如图 2.11 所示。

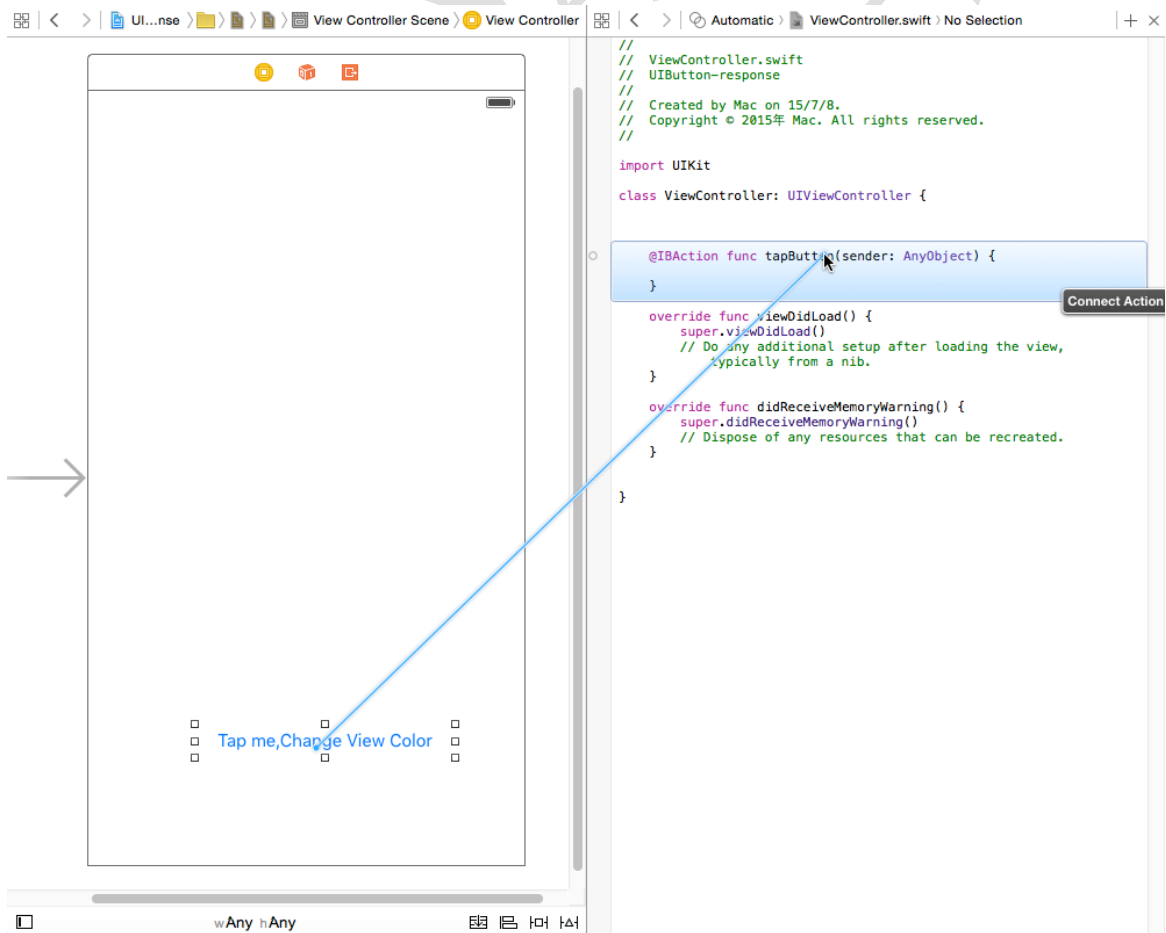


图 2.11 关联动作

最后，松开鼠标后，按钮对象就与动作成功的关联在一起了，此时动作前面的空心小圆圈就变为了实心的小圆圈，它表示此动作已被关联，效果和图 2.9 一样。

(8) 打开 ViewController.swift 文件，编写代码，此代码将实现按钮的响应。代码如下：

```
import UIKit
class ViewController: UIViewController {
    var isYellow:Bool=false
    @IBAction func tapButton(sender: AnyObject) {
        //判断主视图的背景是否为黄色
        if(isYellow){
            self.view.backgroundColor=UIColor.whiteColor()           //设置主视图的背景颜色
            isYellow=false
        }else{
            self.view.backgroundColor=UIColor.yellowColor()
            isYellow=true
        }
    }
    .....
}
```

此时运行程序，首先会看到如图 2.12 的效果。当轻拍 Tap me,Change View Color 按钮后，主视图的背景变为黄色，如图 2.13 所示。当再一次轻拍 Tap me,Change View Color 按钮，主视图的背景颜色将会变回原来的白色。

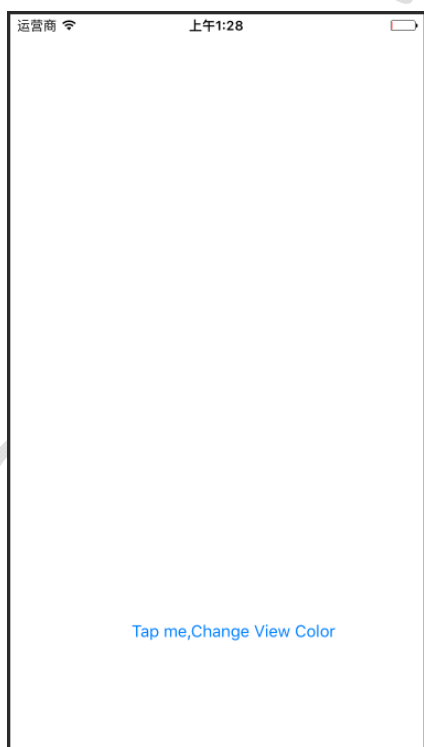


图 2.12 运行效果

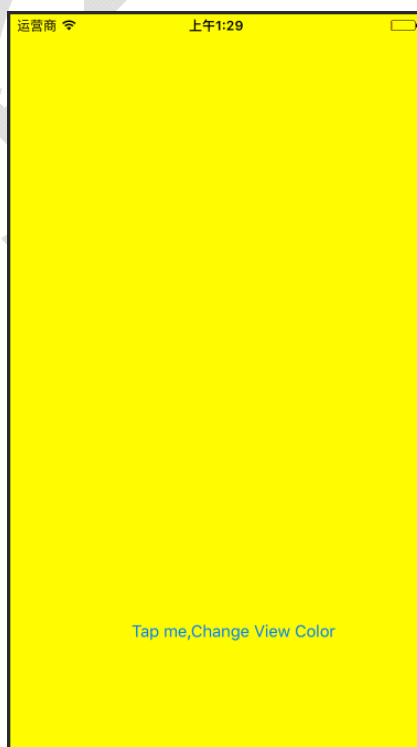


图 2.13 运行效果

## 2.使用代码添加按钮实现的响应

使用代码添加的按钮，实现响应需要使用到 addTarget(\_:action:forControlEvents:)方法，其语法形式如下：

```
func addTarget(_ target: AnyObject?,
              action action: Selector,
              forControlEvents controlEvents: UIControlEvents)
```

其中，参数说明如下：

- ❑ **target**：表示目标对象。它是动作消息的发送方。
- ❑ **action**：表示选择器，用来识别动作消息。它不可以为空。
- ❑ **controlEvents**：表示控件事件。在 iOS 中有 19 种控件事件，如表 2-4 所示。

表 2-4 控件事件

控件事件	解释
TouchDown	单点触摸按下事件：用户点触屏幕，或者又有新手指落下的时候
TouchDownRepeat	多点触摸按下事件，点触计数大于1：用户按下第二、三、或第四根手指的时候。
TouchDragInside	当一次触摸在控件窗口内拖动时。
TouchDragOutside	当一次触摸在控件窗口之外拖动时。
TouchDragEnter	当一次触摸从控件窗口之外拖动到内部时。
TouchDragExit	当一次触摸从控件窗口内部拖动到外部时。
TouchUpInside	所有在控件之内触摸抬起事件。
TouchUpOutside	所有在控件之外触摸抬起事件(点触必须开始与控件内部才会发送通知)。
TouchCancel	所有触摸取消事件，即一次触摸因为放上了太多手指而被取消，或者被上锁或者电话呼叫打断。
ValueChanged	当控件的值发生改变时，发送通知。用于滑块、分段控件、以及其他取值的控件。开发者可以配置滑块控件何时发送通知
EditingDidBegin	当文本控件中开始编辑时发送通知。
EditingChanged	当文本控件中的文本被改变时发送通知。
EditingDidEnd	当文本控件中编辑结束时发送通知。
EditingDidEndOnExit	当文本控件内通过按下回车键（或等价行为）结束编辑时，发送通知。
AllTouchEvents	通知所有触摸事件。
AllEditingEvents	通知所有关于文本编辑的事件。
ApplicationReserved	提供一系列应用程序使用的控制事件的值
SystemReserved	控制事件值的范围内保留供内部使用的框架
AllEvents	通知所有事件。

【示例 2-5】以下将实现轻拍按钮，改变主视图背景颜色的功能。代码如下：

```
import UIKit
class ViewController: UIViewController {
    var isCyan:Bool=false
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        //添加按钮对象
        let button=UIButton(frame: CGRectMake(90, 545, 225, 30))
        button.setTitle("Tap me,Change View Color", forState: UIControlState.Normal) //设置按钮的标题
        button.setTitleColor (UIColor.blackColor(), forState: UIControlState.Normal) //设置按钮标题的颜色
        self.view.addSubview(button)
        //实现按钮的响应
        button.addTarget(self, action: "tapbutton", forControlEvents: UIControlEvents.TouchUpInside)
    }
    func tapbutton(){
        //判断主视图的背景颜色是否为青色
        if(isCyan){
            self.view.backgroundColor=UIColor.whiteColor()
            isCyan=false
        }
    }
}
```

```
}else{
    self.view.backgroundColor=UIColor.cyanColor()
    isCyan=true
}
}
.....
}
```

此时运行程序，首先会看到如图 2.14 的效果。当轻拍 Tap me,Change View Color 按钮后，主视图的背景变为青色，如图 2.15 所示。当再一次轻拍 Tap me,Change View Color 按钮，主视图的背景颜色将会变回原来的白色。

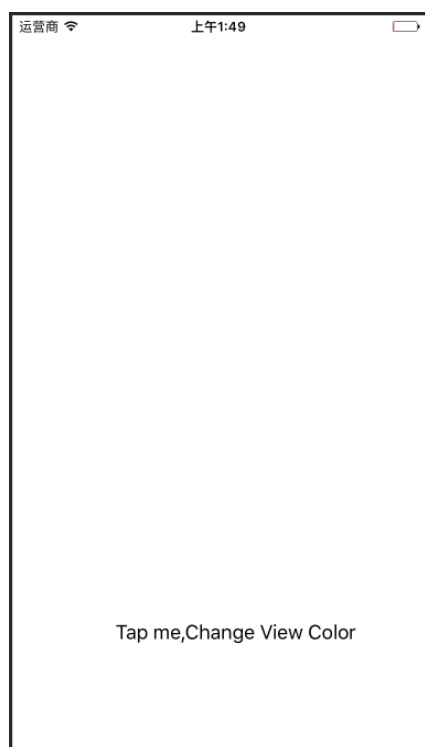


图 2.14 运行效果

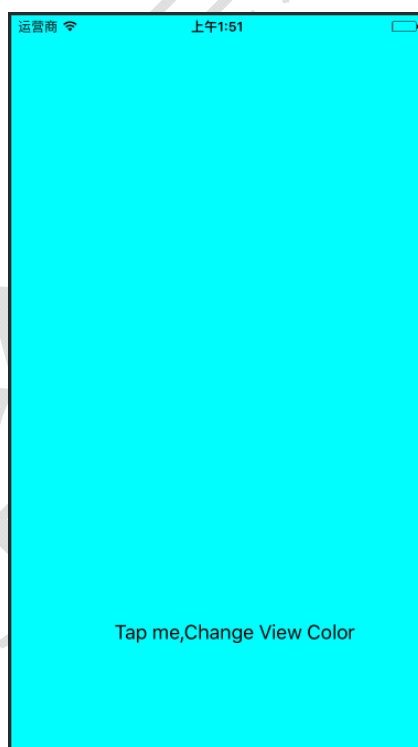


图 2.15 运行效果

## 2.2 显示、编辑文本

在 iOS，经常会看到一些文本的显示。文字就是这些不会说话的设备的嘴巴。通过这些文字，可以很清楚的指定这些设备要表达的信息。本节将主要讲解在 iOS 中，用来显示和编辑文本的三个视图：标签、文本框和文本视图。

### 2.2.1 只读文本——标签

标签视图是一个只读的文本视图，它用于在应用程序中为用户显示少量的信息，如图 2.16 所示。在此图中文字的显示使用的就是标签视图。标签视图一般使用 UILabel 类实现。

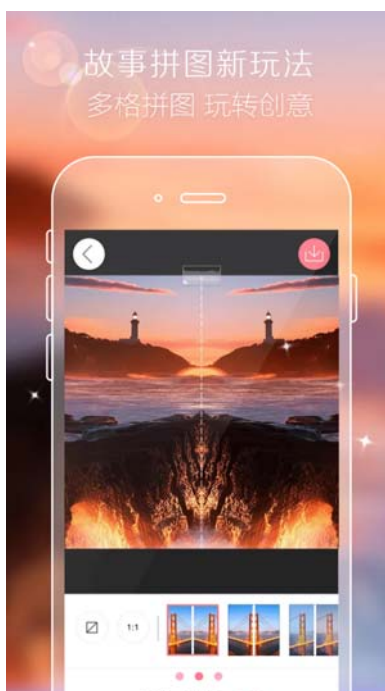


图 2.16 美容相机

【示例 2-6】以下就是通过标签视图显示一首诗的效果。代码如下：

```
import UIKit
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        //添加标签对象 mytitle
        let mytitle=UILabel(frame: CGRectMake(88, 100, 198, 58))
        mytitle.font=UIFont.boldSystemFontOfSize(44)           //设置字体:粗体
        mytitle.textAlignment=NSTextAlignment.Center           //设置对齐方式
        mytitle.text="送友人"
        self.view.addSubview(mytitle)
        //添加标签对象 verse1
        let verse1=UILabel(frame: CGRectMake(44, 197, 303, 63))
        verse1.font=UIFont.systemFontOfSize(24)                //设置字体
        verse1.textAlignment=NSTextAlignment.Center
        verse1.text="青山横北郭，白水绕东城。"
        self.view.addSubview(verse1)
        //添加标签对象 verse2
        let verse2=UILabel(frame: CGRectMake(44, 268, 303, 63))
        verse2.font=UIFont.systemFontOfSize(24)
        verse2.textAlignment=NSTextAlignment.Center
        verse2.text="此地一为别，孤蓬万里征。"
        self.view.addSubview(verse2)
        //添加标签对象 verse3
        let verse3=UILabel(frame: CGRectMake(44, 339, 303, 63))
        verse3.font=UIFont.systemFontOfSize(24)
        verse3.textAlignment=NSTextAlignment.Center
```

```

verse3.text="浮云游子意，落日故人情。"
self.view.addSubview(verse3)
//添加标签对象 verse4
let verse4=UILabel(frame: CGRectMake(44, 410, 303, 63))
verse4.font=UIFont.systemFontOfSize(24)
verse4.textAlignment=NSTextAlignment.Center
verse4.text="挥手自兹去，萧萧班马鸣。"
self.view.addSubview(verse4)
}
.....
}

```

此时运行程序，会看到如图 2.17 所示的效果。

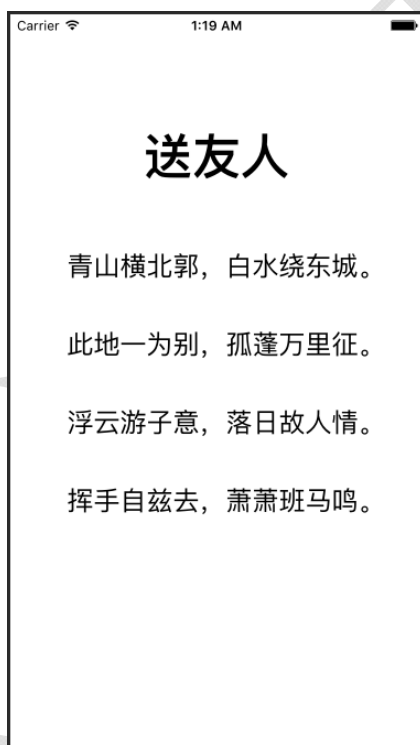


图 2.17 运行效果

注意：标签视图默认是显示一行的，但是，也可以将标签的内容显示为多行。

【示例 2-7】以下将为在标签中显示多行内容。代码如下：

```

import UIKit
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        let verse=UILabel(frame: CGRectMake(20, 191, 333, 196))
        verse.text="    像阳光一样的人，像阳光一样的事，像阳光一样的爱，像阳光一样的慈悲，世界上遍地都是。一声温暖的问候，一个甜甜的微笑，一句简单的祝福，一回小小的帮助，一次善意的举动，无不是我们生命的阳光。我们何不把它们收藏起来，让它照耀我们的人生路，温暖我们需要温暖的内心？心底的阳光多了，自己就是个太阳，还有什么风雨大到能够浇灭太阳？"
        self.view.addSubview(verse)
    }
}

```

```
.....
}
```

此时运行程序，会看到如图 2.18 所示的效果。在图中可以看到标签中只显示了一行文本内容，并没有将所有的内容显示出来。为了解决这一问题，iOS 提供了一个显示多行的属性 `numberOfLines`，这样一来，标签中要显示的内容就可以以多行的形式全部显示出来。在此示例中，我们的标签内容需要 9 行才可以显示出来，所以在代码中添加以下内容：

```
verse.numberOfLines=9
```

此时运行程序，会看到如图 2.19 所示的效果。

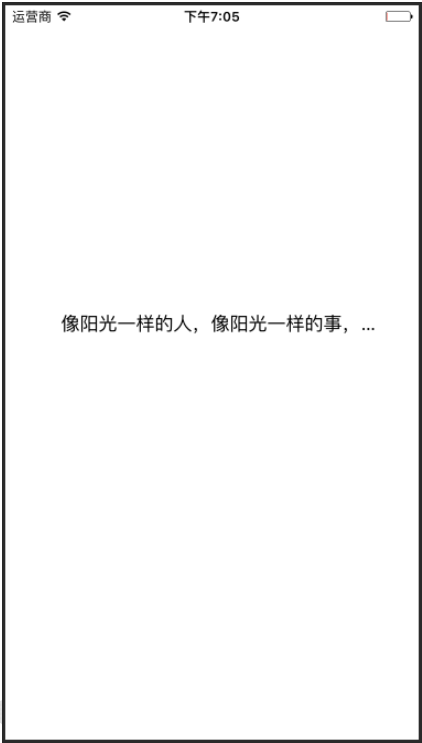


图 2.18 运行效果

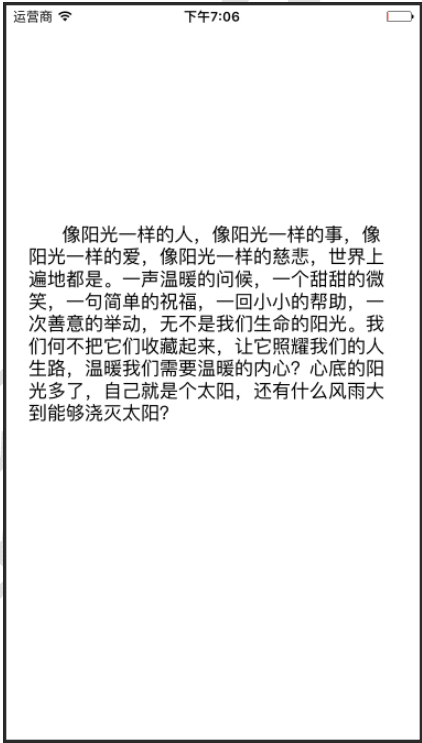


图 2.19 运行效果

2.2.2 单行读写文本——文本框

文本框是一种常见的单行读写文本视图，简单的说就是输入控件，例如一种登录界面要求用户输入用户名和密码等，如图 2.1 所示。文本框一般使用 `UITextField` 实现。

【示例 2-8】以下通过使用文本框实现 QQ 中添加账号的功能。具体操作步骤如下：

- (1) 创建一个 Single View Application 模板类型的项目，命名为 `UITextField`。
- (2) 打开 `Main.storyboard` 文件，对主视图进行设计，效果如图 2.20 所示。

需要添加的视图以及对它们的设置如表 2-5 所示。

表 2-5 对视图对象的设置

视图	设置
View Controller	Size: iPhone 4.7-inch
主视图	Background: 浅灰色
Label	Text: 添加账号 Color: 白色

	Font: Bradley Hand Bold 17.0 Alignment: 居中 位置和大小: (146, 21, 89, 37)
Button	Title: 取消 Font: System 17.0 Text Color: 白色 位置和大小: (317, 21, 46, 30)
View	Background: 天蓝色 位置和大小: (0, 0, 380, 59)

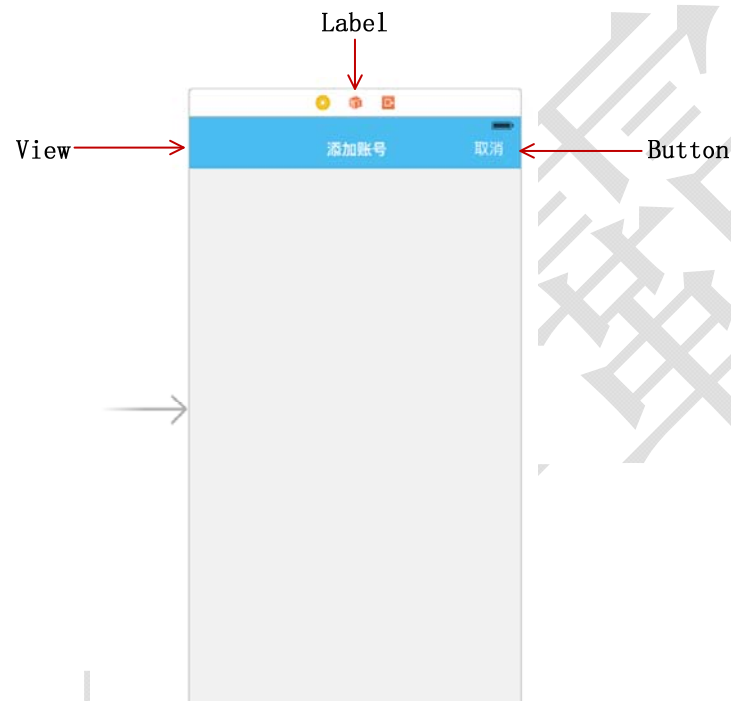


图 2.20 主视图的效果

(3) 打开 ViewController.swift 文件, 编写代码, 此代码实现的功能是账号的添加。代码如下:

```
import UIKit
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        //添加输入账号的文本框
        let accountnumber=UITextField(frame: CGRectMake(0, 116, 375, 50))
        accountnumber.borderStyle=UITextBorderStyle.RoundedRect //设置文本框的边框风格
        accountnumber.placeholder="QQ 号/手机号/邮箱" //设置文本框的占位符
        self.view.addSubview(accountnumber)
        //添加输入密码的文本框
        let password=UITextField(frame: CGRectMake(0, 165.5, 375, 50))
        password.borderStyle=UITextBorderStyle.RoundedRect
        password.placeholder="密码"
        self.view.addSubview(password)
    }
    .....
}
```



```
}
```

此时运行程序，会看到如图 2.21 所示的效果。当开发者轻拍文本框后，会自动弹出键盘，如图 2.22 所示。



图 2.21 运行效果

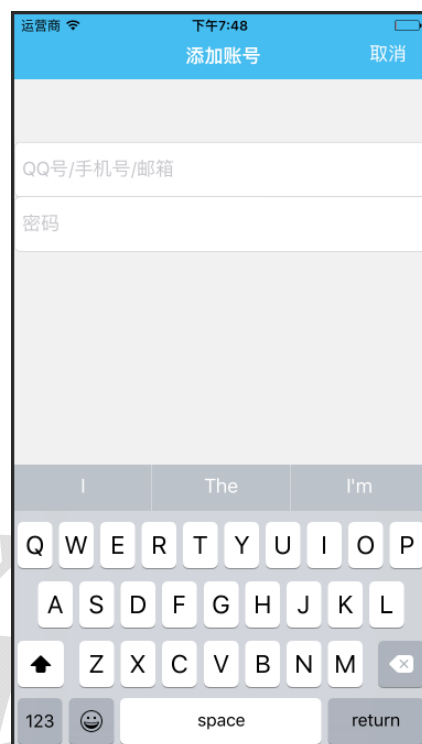


图 2.22 运行效果

### 2.2.3 多行读写文本——文本视图

文本视图也是输入控件，与文本框不同的是，文本视图可以让用户输入多行，如图 2.23 所示。在此图中字符串“说点什么吧”这一区域就是使用文本视图实现的，用户可以在此区域中写大量的文本内容。一般文本框视图使用 `UITextView` 实现。



图 2.23 写日志

【示例 2-9】以下将使用文本视图实现 QQ 中写说说并发表的功能。具体的操作步骤如下：

(1) 创建一个 Single View Application 模板类型的项目，命名为 UITextView。

(2) 打开 Main.storyboard 文件，对主视图进行设计，效果如图 2.24 所示。

需要添加的视图以及对它们的设置如表 2-6 所示。

表 2-6 对视图对象的设置

视图	设置
View Controller	Size: iPhone 4.7-inch
主视图	Background: 浅灰色
Label	Text: 写说说 Color: 白色 Font: Bradley Hand Bold 17.0 Alignment: 居中 位置和大小: (146, 21, 89, 37)
Button1	Title: 取消 Font: System 17.0 Text Color: 白色 位置和大小: (14, 21, 46, 30) 声明和关联动作cancel
Button2	Title: 发表 Font: System 17.0 Text Color: 白色 位置和大小: (317, 21, 46, 30) 声明和关联动作issue
View	Background: 天蓝色 位置和大小: (0, 0, 380, 59)

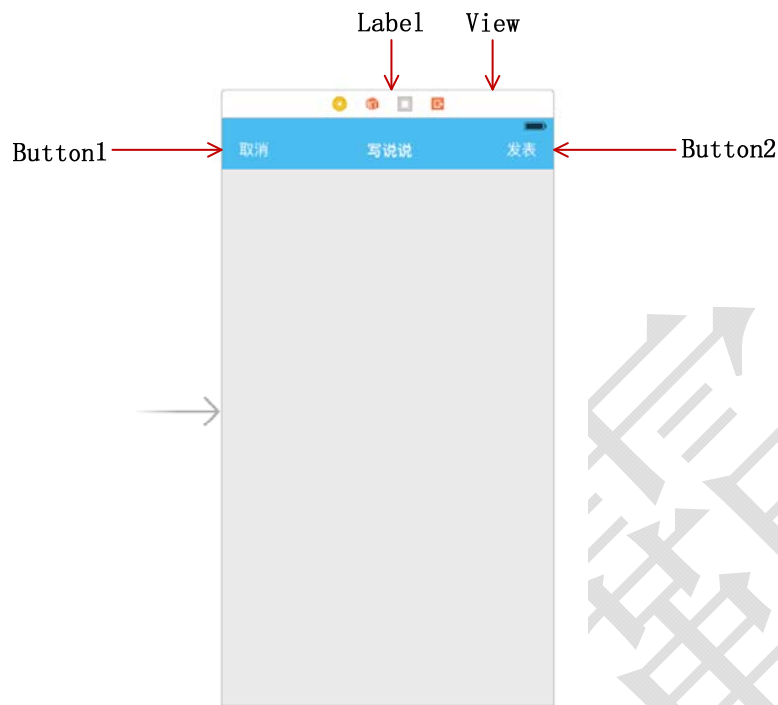


图 2.24 主视图的效果

(3) 打开 ViewController.swift 文件，编写代码，此代码实现的功能是写说说并发表的功能。代码如下：

```
import UIKit
class ViewController: UIViewController, UITextViewDelegate {
    let wtv=UITextView(frame: CGRectMake(0, 97, 375, 232))
    let rtv=UITextView(frame: CGRectMake(0, 372, 375, 232))
    let label=UILabel(frame: CGRectMake(3, 105, 123, 21))
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        self.view.addSubview(wtv) //添加文本视图
        label.text="说点什么吧..." //设置标签的文本内容
        label.enabled=false //禁用标签
        label.backgroundColor=UIColor.clearColor()
        wtv.delegate=self //设置文本视图的委托
        self.view.addSubview(label)
        self.view.addSubview(rtv)
        rtv.backgroundColor=UIColor.clearColor()
        rtv.editable=false //禁用文本视图
        rtv.hidden=true //隐藏文本视图
    }
    //监听文字改变的消息
    func textViewDidChange(textView: UITextView) {
        //判断文本视图的内容是否为空
        if(wtv.text==""){
            label.text="说点什么吧..."
        }else{

```

```

        label.hidden=true
    }
}
//隐藏键盘
@IBAction func cancel(sender: AnyObject) {
    wtv.resignFirstResponder()
}
//发表说说，隐藏键盘
@IBAction func issue(sender: AnyObject) {
    rtv.hidden=false
    rtv.text=wtv.text
    wtv.resignFirstResponder()
}
}
.....
}

```

运行程序后，会看到如图 2.25 所示的效果。当开发者轻拍文本视图后，会自动弹出键盘，如图 2.26 所示。

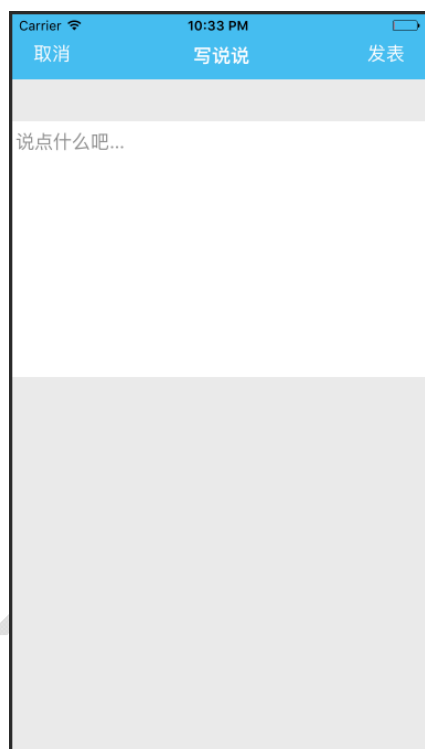


图 2.25 运行效果

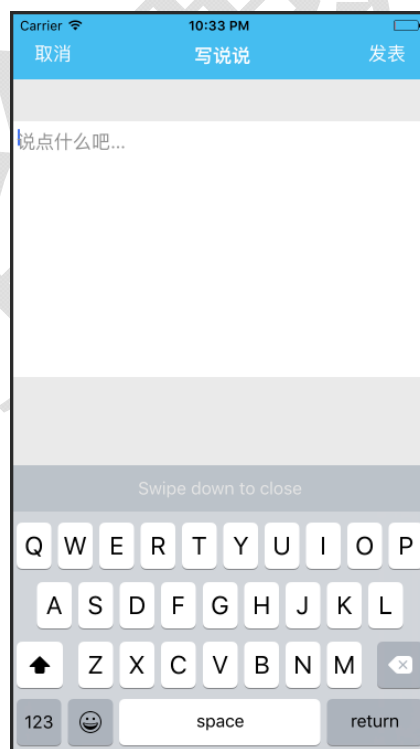


图 2.26 运行效果

当开发者在文本视图输入内容后，字符串“说点什么吧...”就会自动消失，如图 2.27 所示。当轻拍发表按钮后，在文本视图中写入的内容就会显示在另一个文本视图中，并且键盘消失，如图 2.28 所示。

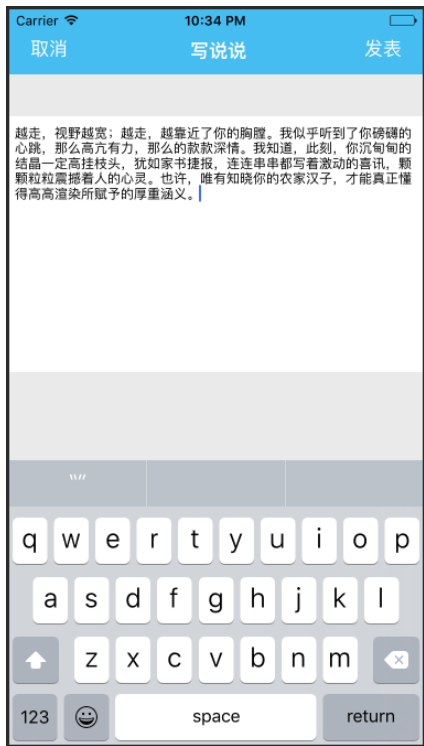


图 2.27 运行效果

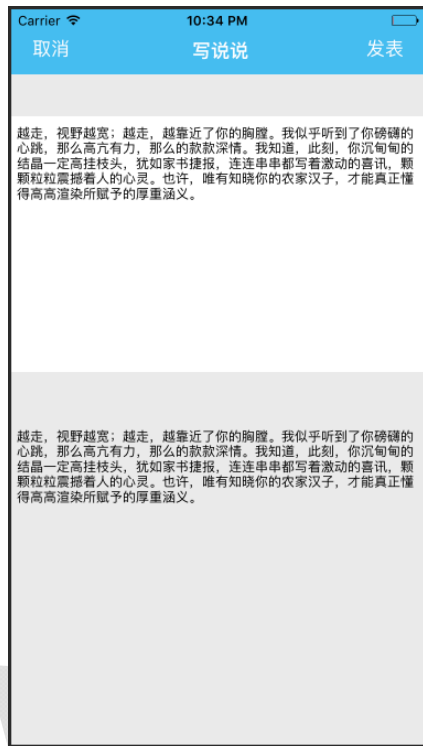


图 2.28 运行效果

## 2.3 使用开关、滑块控件

开关和滑块也是用于和用户进行交互的控件。本节将主要讲解这两种控件。

### 2.3.1 开关

开关控件常用来控制某个功能的开发状态，如蓝牙、GPS、WiFi 信号等。如图 2.29 所示就是一个在 WiFi 中的开关。开关控件一般使用 UISwitch 来实现。



图 2.29 开关

【示例 2-10】以下将使用开发控件实现手电筒的功能。代码如下：

```
import UIKit
class ViewController: UIViewController {
    var isOn:Bool=true
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        //添加开关控件
        let mySwitch=UISwitch(frame: CGRectMake(163, 318, 51, 318))
        self.view.addSubview(mySwitch)
        mySwitch.addTarget(self, action: "switchAction", forControlEvents:
        UIControlEvents.ValueChanged) //响应开发
        self.view.backgroundColor=UIColor.blackColor()
    }
    func switchAction(){
        //判断开关的状态
        if (isOn) {
            self.view.backgroundColor=UIColor.whiteColor() //设置背景颜色
            isOn=false
        } else {
            self.view.backgroundColor=UIColor.blackColor() //设置背景颜色
            isOn=true
        }
    }
    .....
}
```

此时运行程序，会看到如图 2.30 所示的效果。当开发者将开关滑动到开的状态后，就打开了灯光，即界面的背景颜色就变为了白色，如图 2.31 所示。

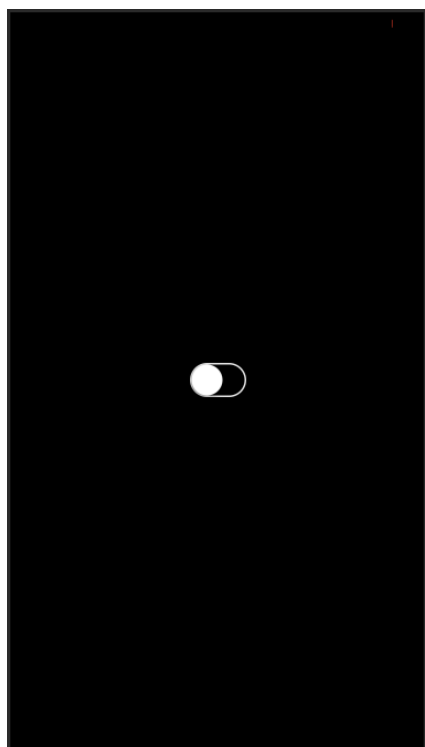


图 2.30 运行效果

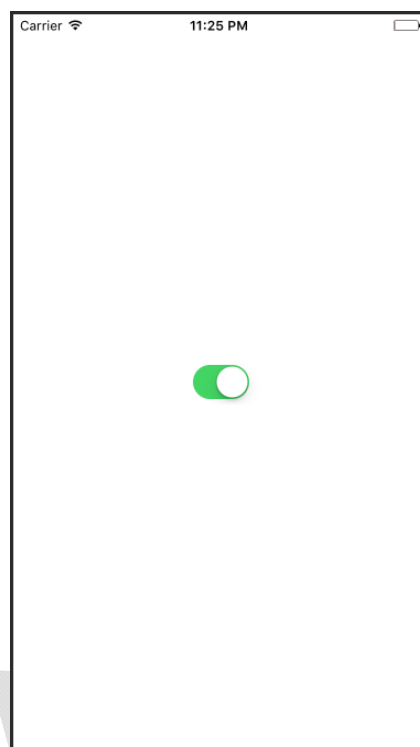


图 2.31 运行效果

### 2.3.2 滑块控件

滑块控件可以从一个连续的区间中选择一个值。例如，可以控制设备的当前音量、亮度等等。图 2.32 中就显示了两个滑块控件，其中一个控制设备的亮度，另一个控制设备的音量。一般使用 `UISlider` 来实现滑块控件。



图 2.32 滑块

【示例 2-11】以下通过滑块控件实现调节屏幕亮度的功能。具体的操作步骤如下：

(1) 创建一个 Single View Application 模板类型的项目，命名为 UISlider。

(2) 打开 Main.storyboard 文件，在主视图中添加一个空白视图，空白视图的位置为(0, 0, 375, 614)。为空白视图声明和关联插座变量，变量名称为 myView。

(3) 打开 ViewController.swift 文件，编写代码。此代码实现的功能是调节屏幕亮度。代码如下：

```
import UIKit
class ViewController: UIViewController {
    @IBOutlet weak var myView: UIView!
    let mySlider=UISlider(frame: CGRectMake(0, 625, 376, 31))
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        self.view.addSubview(mySlider)
        mySlider.value=0.5 //设置滑块控件当前的值
        myView.alpha=CGFloat(mySlider.value) //设置空白视图的透明度
        mySlider.addTarget(self, action: "sliderAction", forControlEvents:
        UIControlEvents.ValueChanged) //实现响应
    }
    func sliderAction(){
        myView.alpha=CGFloat(mySlider.value) //设置空白视图的透明度
    }
    .....
}
```

此时运行程序，会看到如图 2.33 所示的效果。当开发者滑动滑块控件中的滑块就可以对屏幕的亮度进行调节了，如图 2.34 所示。

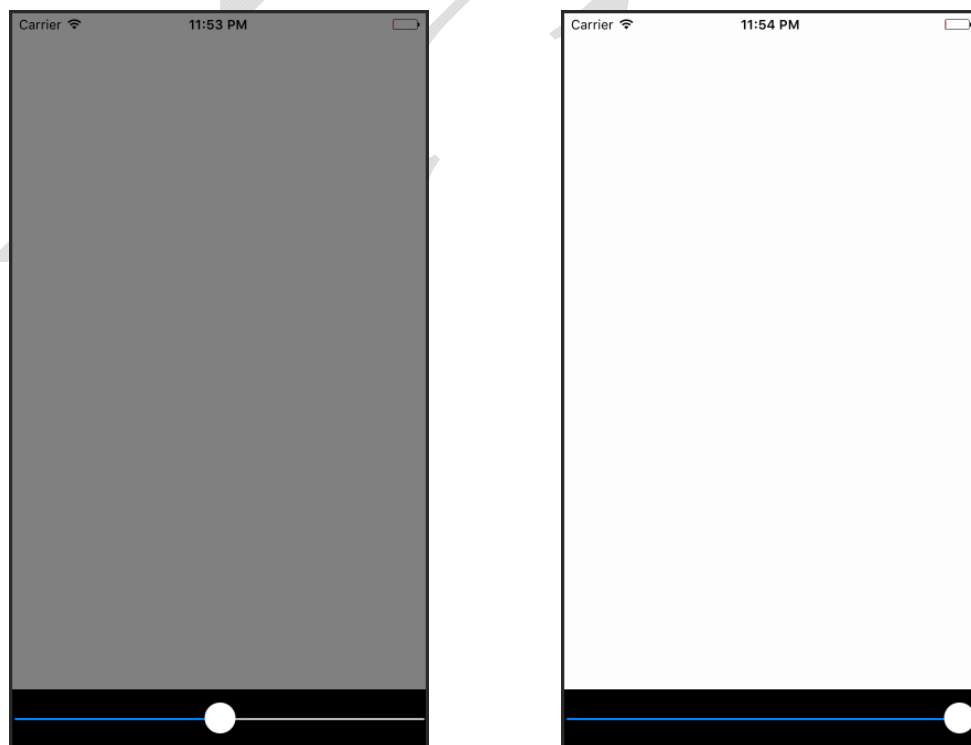




图 2.33 运行效果

图 2.24 运行效果

## 2.4 屏幕滚动视图——滚动视图

由于 iPhone 或者是 iPad 屏幕边界的影响，使我们添加的控件和界面元素受到限制。但是在 iPhone 或者 iPad 开发中，人们使用滚动视图解决了这一受到限制的问题。滚动视图由 UIScrollView 类的一个实例对象实现。

【示例 2-12】以下将使用滚动视图实现标签的滚动。代码如下：

```
import UIKit
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        //添加滚动视图
        let myScroller=UIScrollView(frame: CGRectMake(0, 0, 375, 667))
        myScroller.contentSize=CGSizeMake(375, 10000) //设置滚动范围的大小
        self.view.addSubview(myScroller)
        var y: CGFloat = 10
        var i=1
        for (i;i<=100;i++) {
            //添加标签对象
            let mylabel=UILabel()
            mylabel.frame=CGRectMake(3, y, 375, 50)
            mylabel.text="\i"
            myScroller.addSubview(mylabel)
            y+=100
        }
        .....
    }
}
```

此时运行程序，会看到如图 2.35 所示的效果，当开发者滑动屏幕，会看到滚动视图实现的滚动，如图 2.36 所示。

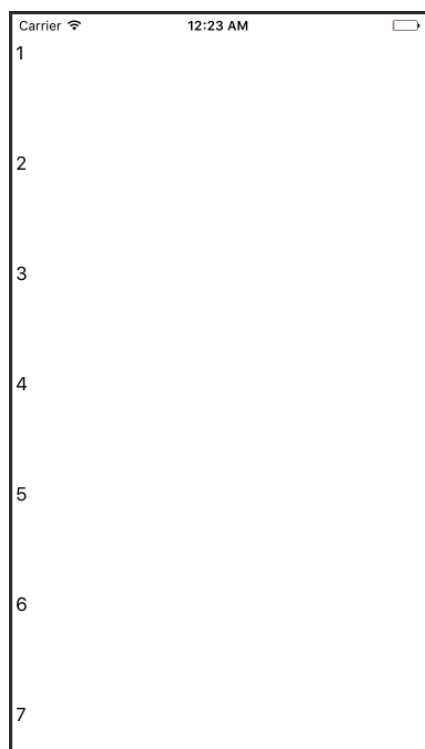


图 2.35 运行效果

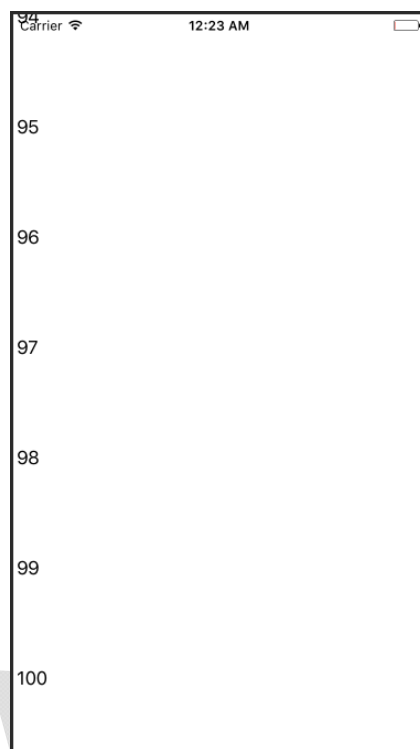


图 2.36 运行效果

注意：滚动视图中需要注意以下点

### 1.常用属性

滚动视图的属性有很多，表 2-7 就总结了滚动视图常用的一些属性。

2-7 滚动视图的属性

属性	功能
contentOffset	滚动视图目前滚动的位置
contentSize	滚动范围的大小
contentInset	其他视图在滚动视图中的位置
delegate	设置委托
directionalLockEnabled	指定滚动视图是否只能在一个方向上滚动
bounces	指定滚动视图遇到边框后是否反弹
alwaysBounceVertical	指定垂直方向遇到边框是否反弹
alwaysBounceHorizontal	指定水平方向遇到边框是否反弹
pagingEnabled	是否能滚动
scrollEnabled	滚动视图是否整页翻动
showsHorizontalScrollIndicator	是否显示水平方向的滚动条
showsVerticalScrollIndicator	是否显示垂直方向的滚动条
scrollIndicatorInsets	指定滚动条在滚动视图中的位置
indicatorStyle	设定滚动条的样式
minimumZoomScale	缩小的最小比例
maximumZoomScale	放大的最大比例
zoomScale	设置变化比例
bouncesZoom	指定缩放的时候是否会反弹

### 2.滚动视图常用事件

在滚动视图中一般会使用到一些事件。这里将常用到的一些事件做了总结，如表 2-8 所示。

表 2-8 滚动视图常用事件

事件	功能
scrollViewDidScroll(_:)	已经滑动
scrollViewWillBeginDragging(_:)	开始拖动
scrollViewDidEndDragging(_:willDecelerate:)	结束拖动
scrollViewShouldScrollToTop(_:)	是否支持滑动至顶部
scrollViewDidScrollToTop(_:)	滑动到顶部时调用该方法
scrollViewWillBeginDecelerating(_:)	开始减速
scrollViewDidEndDecelerating(_:)	减速停止
viewForZoomingInScrollView(_:)	返回一个放大或者缩小的视图
scrollViewWillBeginZooming(_:withView:)	开始放大或者缩小
scrollViewDidEndZooming(_:withView:atScale:)	缩放结束时
scrollViewDidZoom(_:)	视图已经放大或缩小

## 2.5 iOS 9 新增——部署视图

同以往的 iOS 一样，iOS 9 为开发者以及用户带来了很多的新特性。其中，比较特别的是部署视图，它将从根本上改变开发者在 iOS 上创建用户界面的方式。部署视图提供了一个高效的接口，用于平铺一行或一系列的视图组合。部署视图一般使用 `UIStackView` 视图。

【示例 2-13】以下将实现类似淘宝上评分的效果。具体的操作步骤如下：

(1) 创建一个 Single View Application 模板类型的项目，命名为 `UIStackView`。

(2) 打开 `Main.storyboard` 文件，在主视图中添加一个 Vertical Stack View 视图和一个 Horizontal Stack View 视图。

(3) 为 Vertical Stack View 视图添加约束（约束描述了视图间的几何关系），即 `height=350`，`width=375`。添加约束的具体步骤如下：

首先，选中要添加约束的视图，即 Vertical Stack View 视图。

然后，选择 `pin` 图标，如图 2.37 所示。



图 2.37 选中 pin 图标

弹出 Add New Constraints 对话框，如图 2.38 所示。

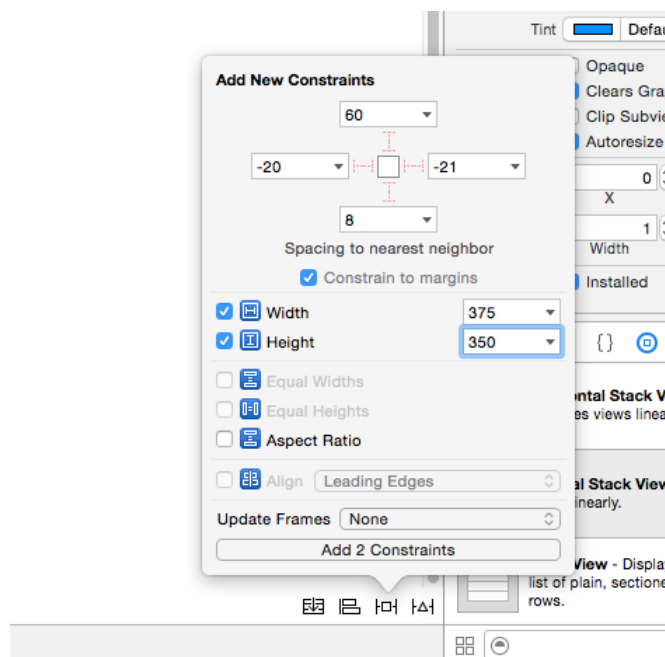


图 2.38 Add New Constraints 对话框

最后，在输入框中输入内容后，在对应项前面打对勾，按下回车后，约束就添加到了 Horizontal Stack View 视图中。

(4) 为 Horizontal Stack View 视图添加约束，即 height=60，width=375。

(5) 添加视图对象到 Vertical Stack View 视图和 Horizontal Stack View 视图中，以及对主视图中的视图对象进行设置，如图 2.39 所示。

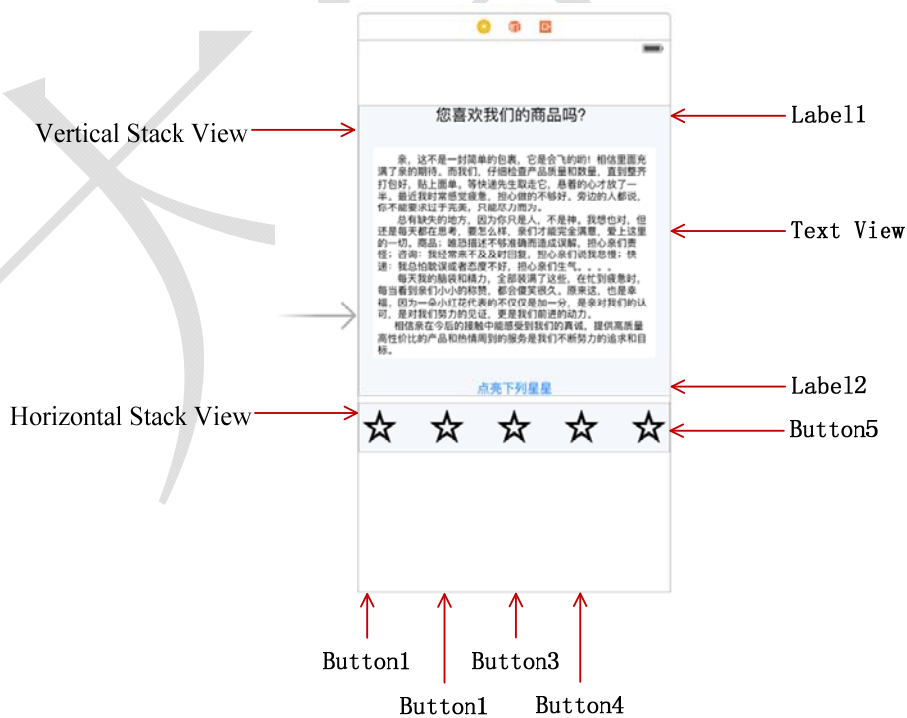


图 2.39 主视图效果

需要添加的视图以及对它们的设置如表 2-9 所示。

表 2-9 对视图对象的设置

视图	设置
View Controller	Size: iPhone 4.7-inch
Vertical Stack View	Alignment: Center Distribution: Equal Spacing Spacing: 10
Label1	Text: 您喜欢我们的商品吗? Font: System 19.0
Text View	Text: 亲,这不是一封简单的包裹,它是会飞的哟!相信里面充满了亲的期待。而我们,仔细检查产品质量和数量,直到整齐打包好,贴上面单。等快递先生取走它,悬着的心才放了一半。最近我时常感觉疲惫,担心做的不够好。旁边的人都说,你不能要求过于完美,只能尽力而为。 总有缺失的地方,因为你只是人,不是神。我想也对,但还是每天都在思考,要怎么样,亲们才能完全满意,爱上这里的一切。商品:唯恐描述不够准确而造成误解,担心亲们责怪;咨询:我经常来不及及时回复,担心亲们说我怠慢;快递:我总怕耽误或者态度不好,担心亲们生气。。。。 每天我的脑袋和精力,全部装满了这些,在忙到疲惫时,每当看到亲们小小的称赞,都会傻笑很久。原来这,也是幸福。因为一朵小红花代表的不仅仅是加一分,是亲对我们的认可,是对我们努力的见证,更是我们前进的动力。 相信亲在今后的接触中能感受到我们的真诚,提供高质量高性价比的产品和热情周到的服务是我们不断努力的追求和目标。 Font: System 12.0
Label2	Text: 点亮下列星星 Color: 蓝色 Font: System 15.0
Horizontal Stack View	Alignment: Center Distribution: Fill Equally Spacing: 30
Button1	Title: ☆ Font: System 51.0 Text Color: 黑色 声明和关联插座变量button1 声明和关联动作action1
Button2	Title: ☆ Font: System 51.0 Text Color: 黑色 声明和关联插座变量button2 声明和关联动作action2
Button3	Title: ☆ Font: System 51.0 Text Color: 黑色 声明和关联插座变量button3 声明和关联动作action3
Button4	Title: ☆ Font: System 51.0 Text Color: 黑色 声明和关联插座变量button4 声明和关联动作action4
Button5	Title: ☆ Font: System 51.0 Text Color: 黑色 声明和关联插座变量button5

(6) 打开 ViewController.swift 文件，编写代码，此代码实现的功能是轻拍按钮，改变按钮的标题和颜色。代码如下：

```
import UIKit
class ViewController: UIViewController {
    @IBOutlet weak var button1: UIButton!
    @IBOutlet weak var button2: UIButton!
    @IBOutlet weak var button3: UIButton!
    @IBOutlet weak var button4: UIButton!
    @IBOutlet weak var button5: UIButton!
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }
    //触摸按钮改变第一个按钮的标题和颜色
    @IBAction func action1(sender: AnyObject) {
        self.addstar(button1)
    }
    //触摸按钮改变第一个按钮和第二个按钮的标题和颜色
    @IBAction func action2(sender: AnyObject) {
        self.addstar(button1)
        self.addstar(button2)
    }
    //触摸按钮改变第一个按钮、第二个按钮和第三个按钮的标题和颜色
    @IBAction func action3(sender: AnyObject) {
        self.addstar(button1)
        self.addstar(button2)
        self.addstar(button3)
    }
    //触摸按钮改变第一个按钮、第二个按钮、第三个按钮和第四个按钮的标题和颜色
    @IBAction func action4(sender: AnyObject) {
        self.addstar(button1)
        self.addstar(button2)
        self.addstar(button3)
        self.addstar(button4)
    }
    //触摸按钮改变全部按钮的标题和颜色
    @IBAction func action5(sender: AnyObject) {
        self.addstar(button1)
        self.addstar(button2)
        self.addstar(button3)
        self.addstar(button4)
        self.addstar(button5)
    }
    //改变按钮的标题以及颜色
    func addstar(button:UIButton){
        button.setTitle("★", forState: UIControlState.Normal)
        button.setTitleColor (UIColor.orangeColor(), forState: UIControlState.Normal)
    }
    .....
}
```

}

此时运行程序，会看到如图 2.40 所示的效果。当开发者触摸按钮后，会看到类似点亮星星的效果，如图 2.41 所示。

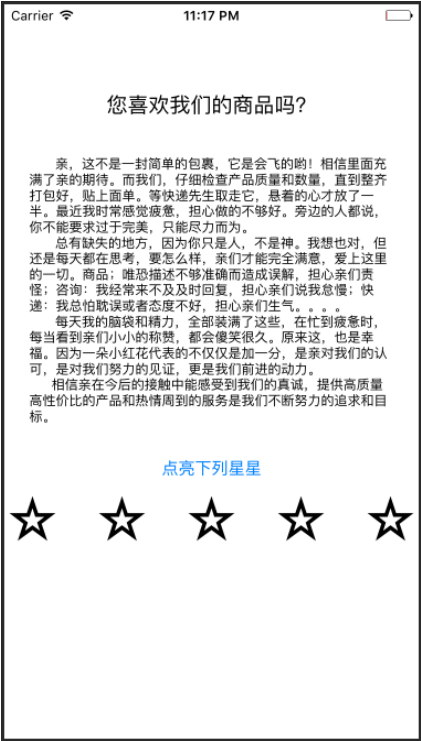


图 2.40 运行效果



图 2.41 运行效果

注意，在示例中 Vertical Stack View 视图和 Horizontal Stack View 视图都属于布局视图，只是他们平铺视图的方式不同。Vertical Stack View 视图用于平铺一列视图组合；Horizontal Stack View 视图用于平铺一行视图组合。在使用这些视图时，遇到的属性如表 2-10 所示。

表 2-10 Stack View视图的属性

属性	功能
Axis	决定了Stack View的subview是水平排布还是垂直排布
Alignment	控制subview对齐方式
Distribution	定义subview的分布方式
Spacing	定义了subview间的最小间距
Baseline Relative	决定了其视图间的垂直间隙是否根据基线测量得到
Layout Margins Relativet	决定了Stack View视图平铺其管理的视图是否要参照它的布局边距