火龙果•整理
uml.org.cn

第 2 章

第一个iOS应用程序

从控制台输出Hello World是我学习C语言的第一步，也是我人生中非常重要的一步。在多年后的今天，我希望仍以Hello World作为第一步，为大家开启一个神奇、瑰丽的世界——iOS。

下面，让我们用Hello World开启iOS的大门，开始我们的探索之旅。

本章是以Hello World作为切入点，向大家系统介绍什么是iOS应用以及如何使用Xcode创建iOS应用。

2.1 创建基于 nib 的 HelloWorld 工程

在学习之初，我们有必要对iOS工程做一个整体概览，而这正是HelloWorld的重要使命所在。之后，我们将详细阐述操作过程中涉及的重要知识点。

- HelloWorld 的使命：尽可能简单明了地向大家介绍什么是 iOS 工程以及如何创建工程。
- 工程实现的功能：在页面上展示字符 Hello World（效果如图 2-1 所示）。
- 包含控件：Label（标签控件）。

图2-1　Hello World的iPhone画面

2.1.1 创建工程

启动 Xcode，点击 File→New→Project 菜单，在打开的 Choose a template for your new project 界面中，选择 Single View Application 工程模板（如图 2-2 所示）。

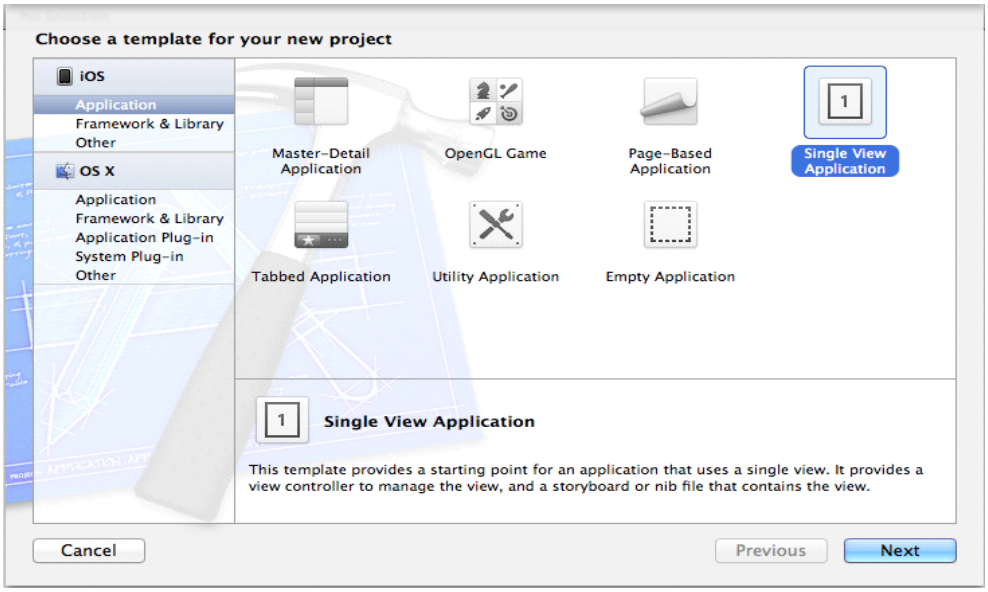

图2-2　选择工程模板

然后点击 Next 按钮，随即出现图 2-3 所示的界面。

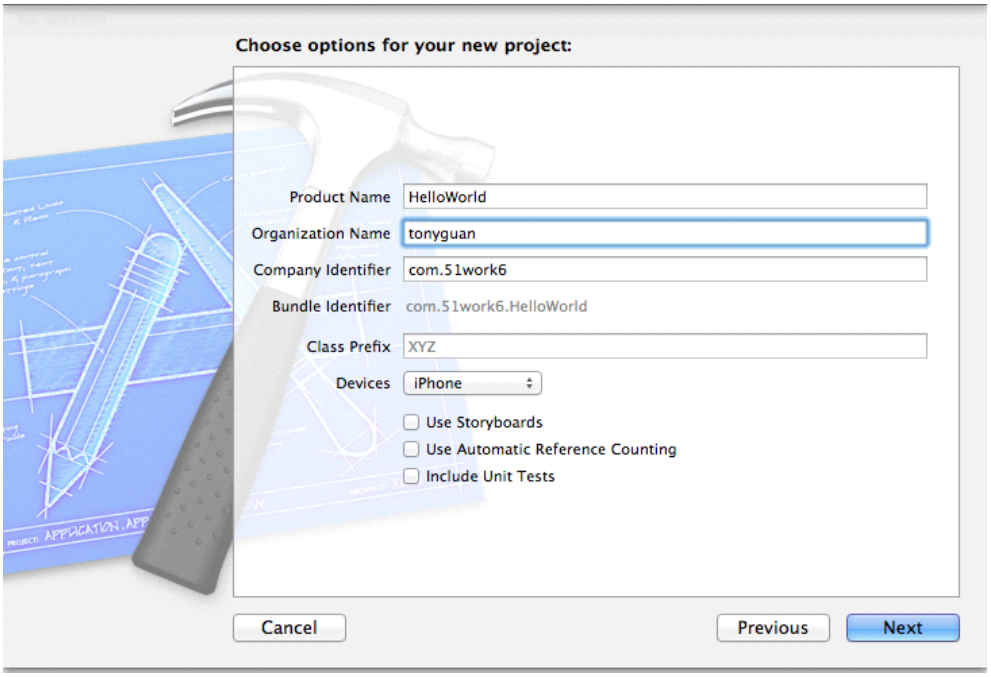

图2-3　新工程中的选项

这里我们需要按照提示并结合自己的实际情况和需要输入相关内容。下面简要说明图 2-3 中的选项。

- **Product Name：工程名字。**
- **Organization Name：组织名字。**
- **Company Identifier：公司标识（很重要）。**一般是将公司的域名倒过来输入进去（如

com.51work6），这类似于Java中的包命名。

- **Bundle Identifier：捆绑标识符（很重要）。**该标识符由Product Name+ Company Identifier构成。因为在App Store发布应用的时候会用到它，所以它的命名不可重复。
- **Class Prefix：类的前缀。**为生成的类加前缀（如XZY ViewController）。
- **Devices：选择设备。**可以构建基于iPhone或iPad的工程，也可以构建通用工程。通用工程是指一个工程可以同时适应iPhone和iPad，不论在iPhone上还是iPad上都可以正常运行。
- **Use Storyboards：工程是否采用故事板技术。**
- **Use Automatic Reference Counting：工程是否采用ARC（自动引用计数）技术。**ARC可以帮助我们管理工程的内存。
- **Include Unit Tests：是否产生单元测试相关的类。**

设置完相关的工程选项后，点击 Next 按钮，进入下一级页面。根据提示选择存放文件的位置，然后点击 Create 按钮，将出现如图 2-4 所示的界面。

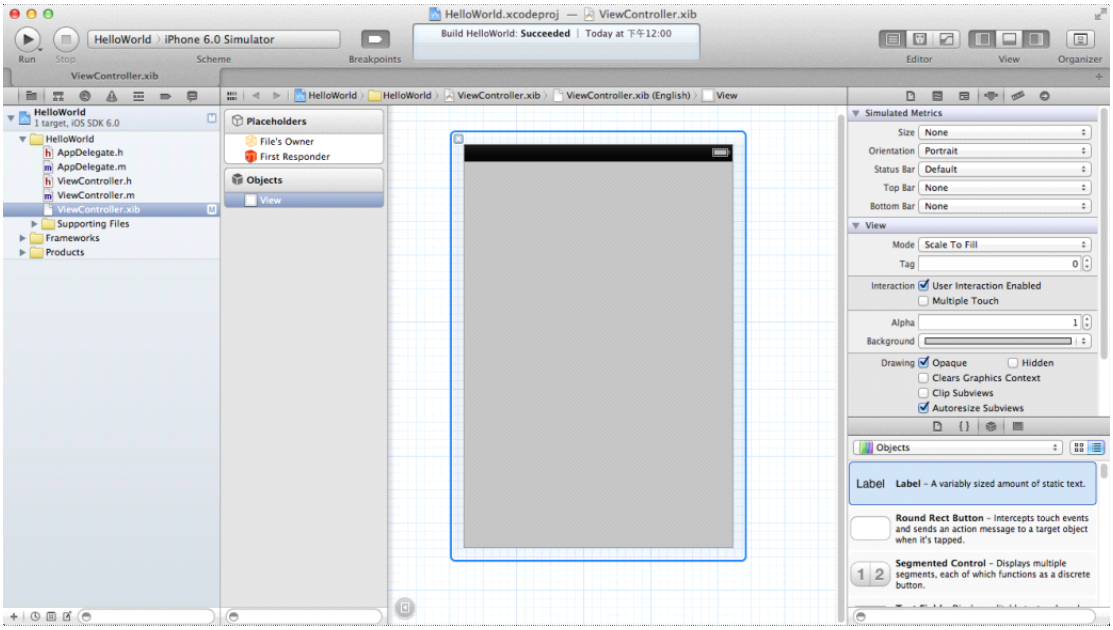

图2-4　新创建的工程

在右下角的对象库中选择 Label 控件，将其拖曳到 View 上并调整其位置。双击 Label 控件，使其处于编辑状态（也可以通过控件的属性来设置），在其中输入 Hello World，如图 2-5 所示。

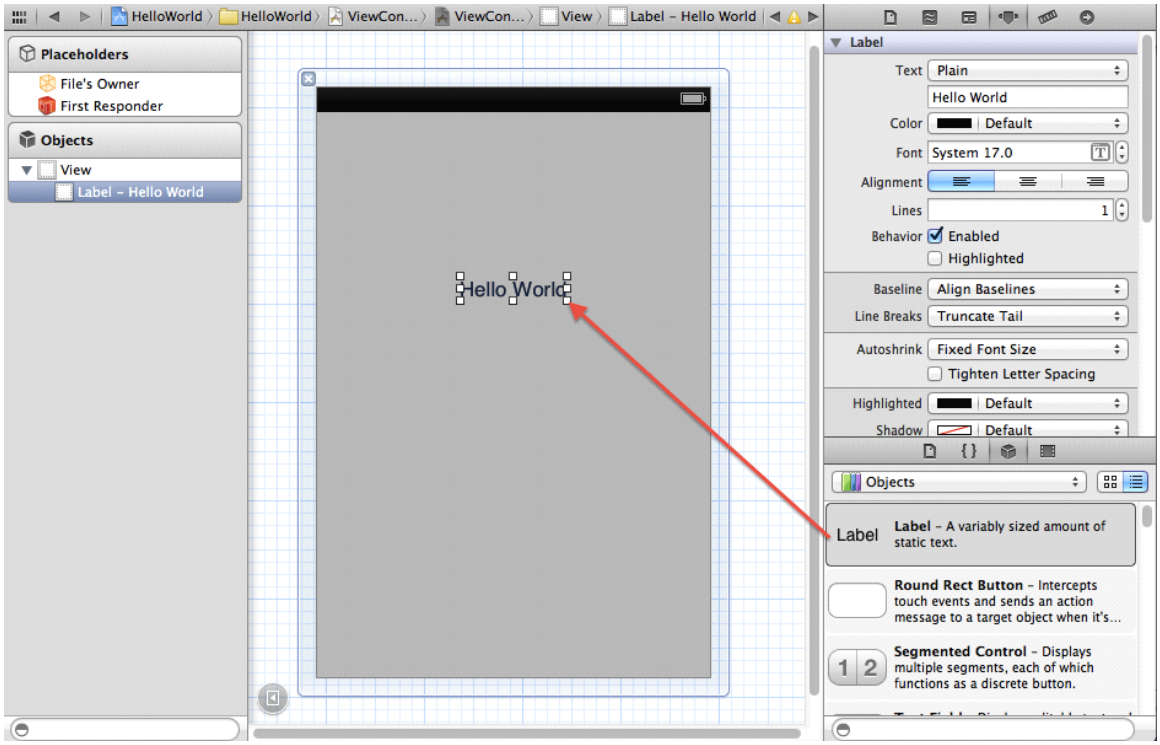

图2-5　添加Label控件

至此，整个工程创建完毕。

点击图 2-4 左上角的 Run 按钮 ，即可看到运行结果。

我们在没有输入一行代码的情况下，就已经利用 Xcode 工具的 Single View Application 模板创建了一个工程，并成功运行，Xcode 之强大可见一斑。

2.1.2　Xcode 中的 iOS 工程模板

从图 2-2 可以看出，iOS 工程模板分为 3 类：Appliction、Framework & Library 和其他，下面详细介绍一下。

1. Application类型

Application 类型可以创建 iOS 应用程序，我们大部分的开发工作都是从这里开始的。该类共包涵 7 个模板，具体如下所示。

- **Master-Detail Application**。可以构建树形结构导航模式应用，生成代码中包含了导航控制器、表视图控制器等。
- **OpenGL Game**。可以构建基于OpenGL ES的游戏应用。
- **Page-Based Application**。可以构建类似于电子书效果的应用，这是一种平铺页面导航。
- **Single View Application**。可以构建简单的单个视图应用。
- **Tabbed Application**。可以构建标签导航模式的应用，生成的代码中包含了标签控制器、标签栏等。
- **Utility Application**。可以构建实用型的应用程序，它会生成两个视图控制器——主视图控制器和子视图控制器。在iPhone中子视图以模态方式呈现，在iPad中子视图以Popover（弹出气泡）形式呈现。
- **Empty Application**。可以构建一个空应用程序，需要我们自己添加视图等对象。该模板很少使用。

2. Framework & Library类型

Framework & Library 类型的模板如图 2-6 所示，它可以构建基于 Cocoa Touch 的静态库。

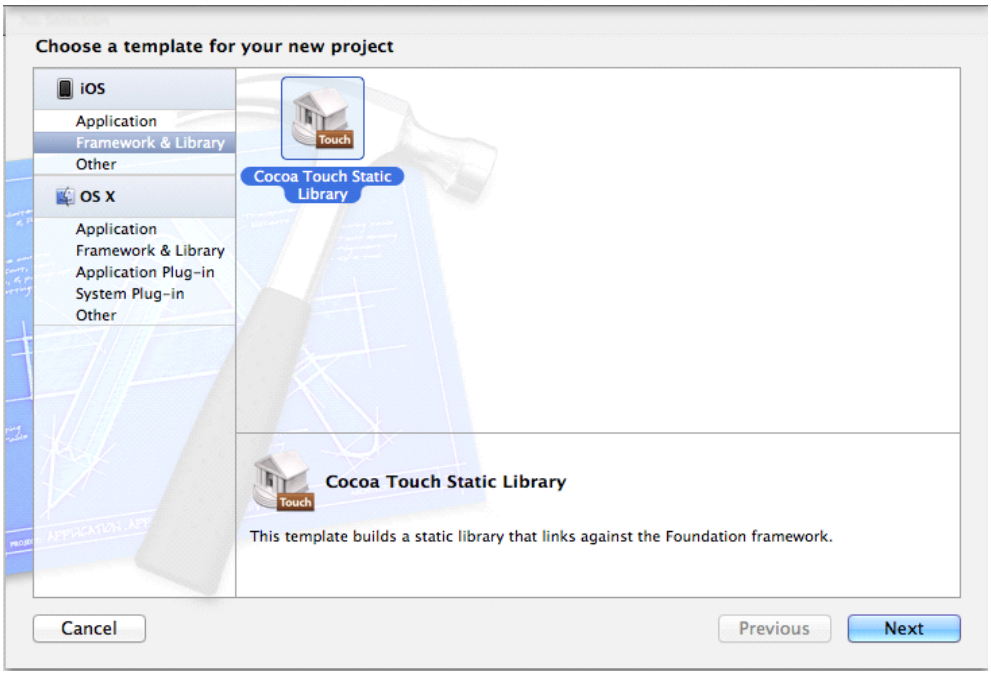

图2-6　Framework & Library类型模板

说明　出于代码安全和多个工程重用代码的考虑，我们需要将一些类或者函数编写成静态库。静态库不能独立运行，编译成功时会生成名为lib*XXX*.a的文件（例如libHelloWorld.a）。

3.　其他类型

如图 2-7 所示，利用该类型我们可以构建应用的内置付费内容包（In-App Purchase Content）和空工程。使用内置付费内容包，可以帮助我们构建具有内置收费功能的应用。

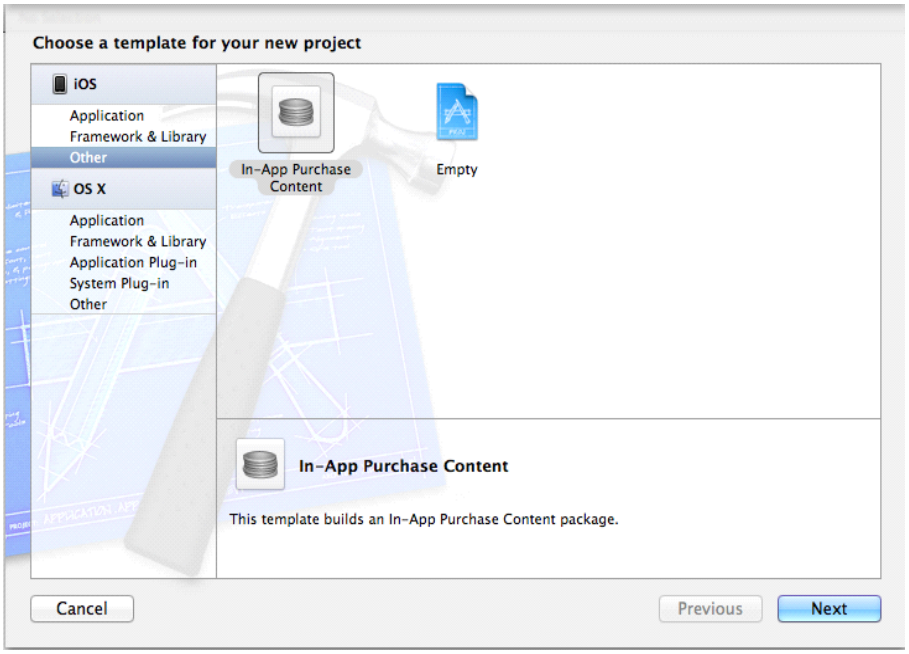

图2-7　Other类型模板

根据需要选用不同的工程模板，可以大大减少我们的工作量。

2.1.3 应用剖析

在创建 HelloWorld 的过程中，生成了很多文件（展开 Xcode 左边的项目导航视图可以看到，如图 2-8 所示），它们各自的作用是什么，彼此间又是怎样的一种关系呢？

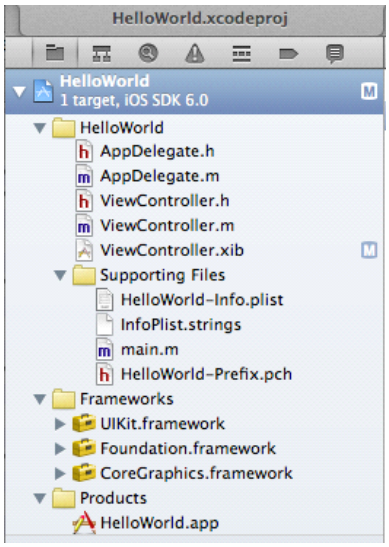

图2-8　项目导航视图

如图 2-8 所示，导航视图下有 HelloWorld、Frameworks 和 Products 三个组。

在 **HelloWorld 组**中共有两个类:AppDelegate 和 ViewController，以及一个组 Supporting Files。

我们主要的编码工作就是在 AppDelegate 和 ViewController 这两个类中进行的，它们的类图如图 2-9 所示。

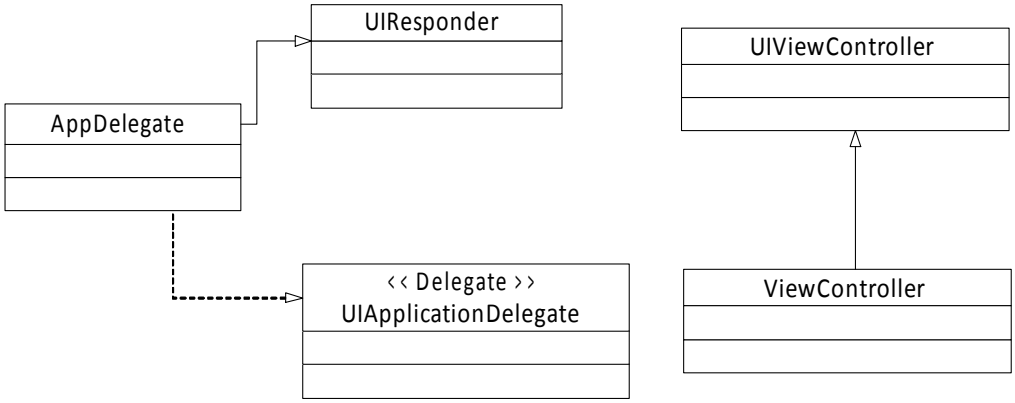

图 2-9　HelloWorld 工程中的类图

AppDelegate是应用程序委托对象，它继承了UIResponder类，并实现了UIApplicationDelegate委托协议。UIResponder类可以使子类AppDelegate具有处理相应事件的能力，而UIApplicationDelegate委托协议使AppDelegate能够成为应用程序委托对象，这种对象能够响应应用程序的生命周期。相应地，AppDelegate的子类也可以实现这两个功能。

ViewController类继承自UIViewController类，它是视图控制器类，在工程中扮演着根视图、用户事件控制类的角色。需要特别指出的是，ViewController.xib文件也是视图控制文件，起描述作用，与ViewController配套存在。

AppDelegate和ViewController类与main代码模块的主函数存在一种直接的调用关系，下面我们借助UML时序图来进行详细说明，见图2-10。

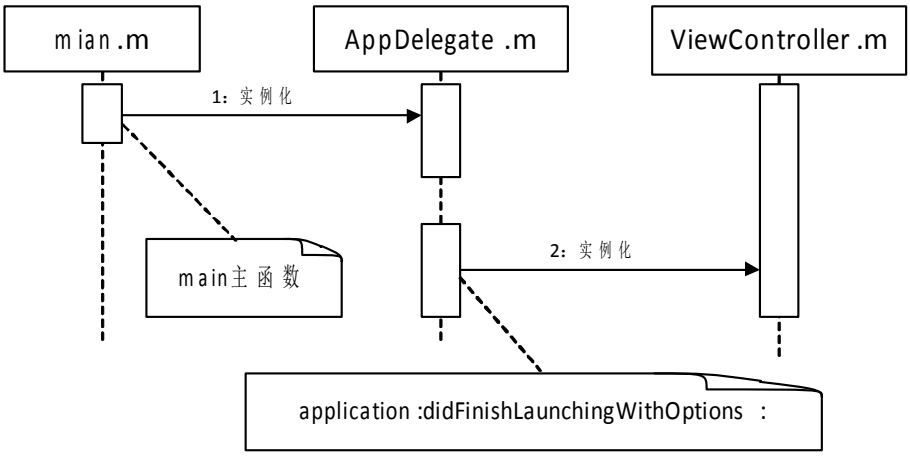

图2-10　HelloWorld启动时序图

在图2-10中，在HelloWorld启动过程中，首先调用main.m代码模块的main主函数进行AppDelegate的实例化，具体代码如下所示：

```
int main(int argc, char *argv[])
{
    @autoreleasepool{
        return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
    }
}
```

AppDelegate类是应用程序委托对象，这个类中继承的一系列方法在应用生命周期的不同阶段会被回调。启动HelloWorld时，首先会调用application: didFinishLaunchingWithOptions:方法，该方法的代码如下：

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]] autorelease];
    self.viewController = [[[ViewController alloc] initWithNibName:@"ViewController" bundle:nil] autorelease];
    self.window.rootViewController = self.viewController;
    [self.window makeKeyAndVisible];
    return YES;
}
```

在该方法中共做了3件事：实例化UIWindow，实例化ViewController，然后把ViewController作为根视图控制器放到UIWindow上。每一个iOS应用都有一个UIWindow对象，而每一个UIWindow对象上面都一个根视图，它所对应的控制器为根视图控制器（ViewController），UIWindow对象与根视图直接的关系如图2-11所示。在根视图上可以添加子视图。各种控件（包括UIWindow）都继承了UIView。

图2-11　UIWindow对象与根视图

Supporting Files组共有4个文件，为便于大家理解，经过分析、提炼，我们将各文件的作用以表的形式向大家总结说明，见表2-1。

表 2-1　Supporting Files 组文件说明表

文件名	说明
HelloWorld-Info.plist	工程属性描述文件，它的命名必须是“工程名+Info.plist”
InfoPlist.strings	工程本地化的字符串文件
main.m	应用程序的入口程序文件，它不是一个类，其中只有一个main主函数，应用的运行都是由此开始
HelloWorld-Prefix.pch	这个文件中可以引入一些头文件，这样工程中的其他文件就不需要再引入了。 它的命名必须是“工程名+ Prefix.pch”

Frameworks 组包含工程里引用到的框架或类库，而 **Products** 组是工程将要生成的产品包。

2.2　基于故事板的 HelloWorld 工程

Storyboard（故事板）是用来替代xib的技术，也是iOS 5最重要的新特性之一。本节中我们将用Storyboard（故事板）重构HelloWorld。

2.2.1　使用故事板重构 HelloWorld

参见 2.1 节描述的 HelloWorld 创建过程，我们在图 2-3 中勾选“Use Storyboards”项。工程创建完成之后，通过导航进入 MainStoryboard.storyboard，页面如图 2-12 所示。

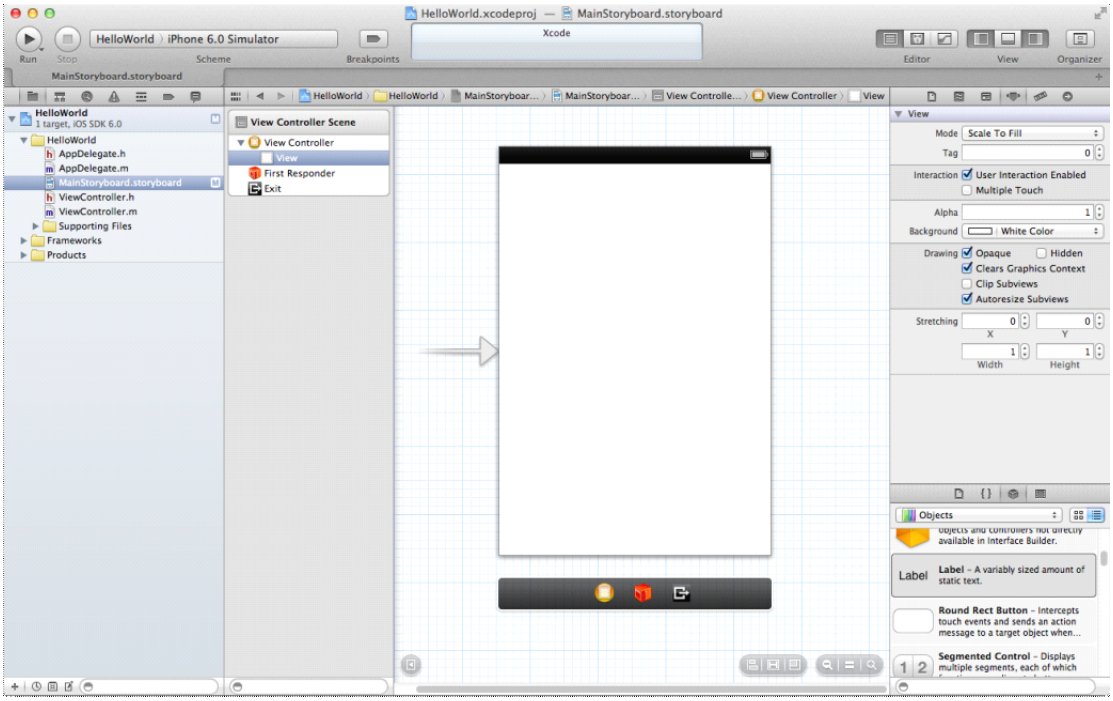

图2-12　创建故事板工程

添加 Label 控件的操作与 2.1 节相同，如图 2-13 所示。

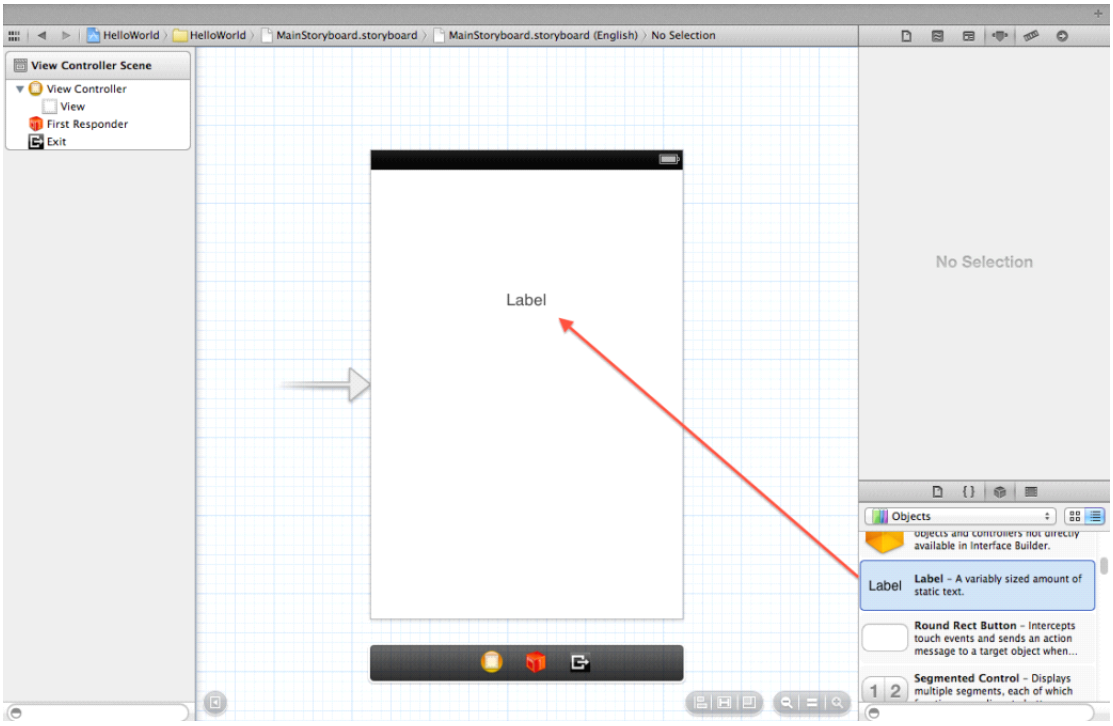

图2-13　添加Label控件

点击 Run 按钮运行，其结果与 2.1 节相同。

2.2.2　nib、xib 与故事板

如果大家使用过苹果的官方资料，一定会发现某些资料上会提到 nib 文件，那么 nib 与 xib

是怎样的一种关系呢？

最初只有 nib 文件，后来将其更名为 xib，但大家一直沿袭 nib 这个叫法（即称 xib 文件为 nib 文件），所以目前为止，nib 等同于 xib。xib 文件采用 xml 格式。

前文已提到故事板是用来替代 xib 的，那么两者除后缀名外，还存在哪些差异呢？

- 首先，在数量上，使用故事板技术时，一个工程只有一个故事板文件。当使用 xib 技术时，xib 在数量上与视图控制器相对应，而一个工程可能会有很多视图控制器，相应地就会有很多 xib 文件。
- 其次，故事板与视图的关系可以在 IB 设计器中很明显地体现，而 xib 与视图的关系则需要查看相关代码或利用其他资源。下面我们来举例说明。

我们要做这样一个应用：两个不同的画面，有两个标签分别与其对应，点击标签两个画面实现互相切换，该应用采用标签栏导航模式，设计原型草图见图 2-14。

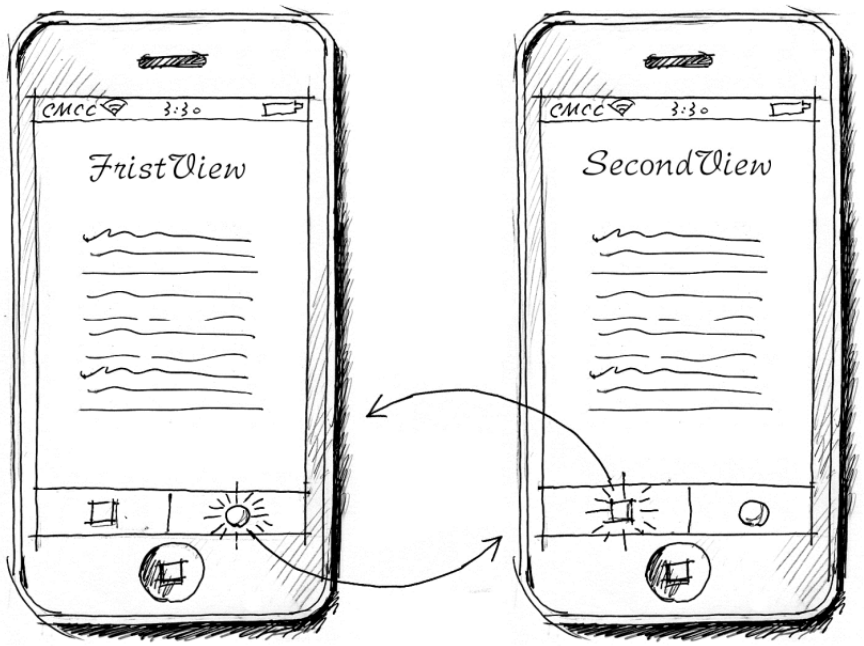

图2-14　设计原型草图

选择 Tabbed Application 模板，分别采用 xib 和故事板文件实现，生成的文件如图 2-15 和图 2-16 所示。

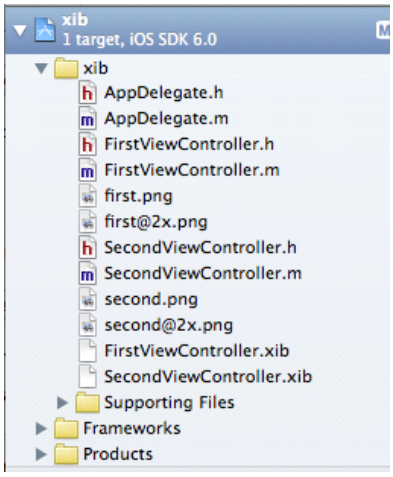

图2-15　采用xib构建文件

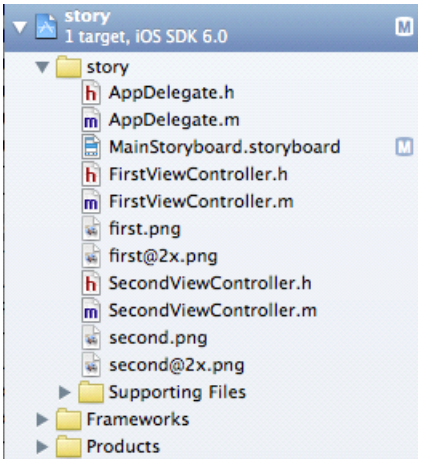

图2-16　采用故事板构建文件

从图 2-15 可以看到采用 xib 技术的时候两个画面有两个 xib 文件，而采用故事板时候两个画面只有一个 MainStoryboard.storyboard 文件，如图 2-16 所示。

打开 MainStoryboard.storyboard 文件，我们会看到如图 2-17 所示设计视图。

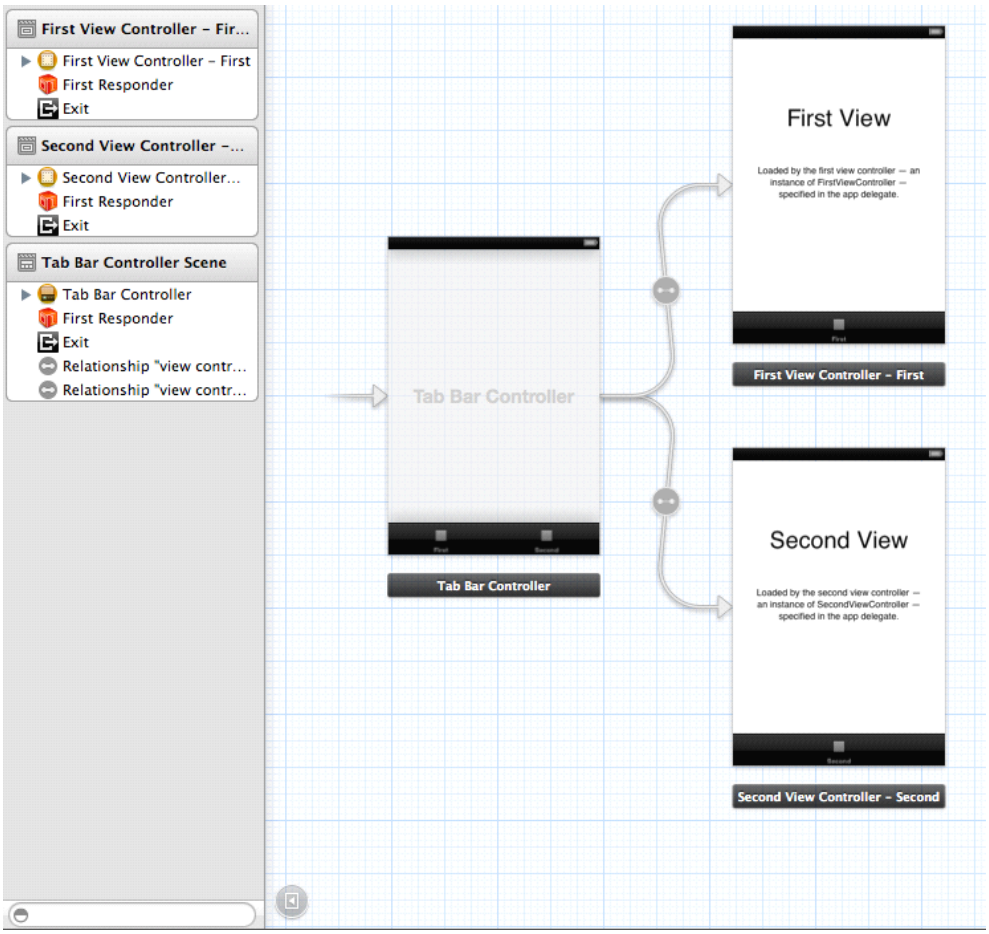

图2-17　故事板设计视图

应用包含两个视图，两个视图存在切换关系，这些信息从上图中一目了然。事实上故事板是多个xib文件集合的描述文件，也采用xml格式。

需要特别提出的是，虽然苹果官方主张使用故事板，但最正确的做法是我们要根据具体情况、

具体问题对故事板和 xib 做以取舍，而不是一概而论。当应用数据量很大、画面很多、关系很复杂的时候，如果使用 storyboad 技术，那我们在 IB 设计器中的工作就会变得庞大而复杂，除此之外整个工程的性能也会受到一定影响。

2.2.3 故事板中的 Scene 和 Segue

Scene 和 Segue（参见图 2-18）是故事板中非常重要的两个概念。 每个视图控制器都会对应一个 Scene 屏幕，在这个屏幕中有很多控件，相当于一个 xib。这些 Scene 之间通过 Segue 连接，Segue 不但定义了 Scene 之间的跳转方式还体现了 Scene 之间的关系。跳转的类型分为：Push、Modal、Popover 和自定义方式。Scene 跳转类型还要跟具体的控制器结合使用。Push 使用 UINavigationController 控制器的树形导航模式，Modal 是模态导航模式，Popover 是弹出一个气泡，这些导航模式我们会在后面介绍。

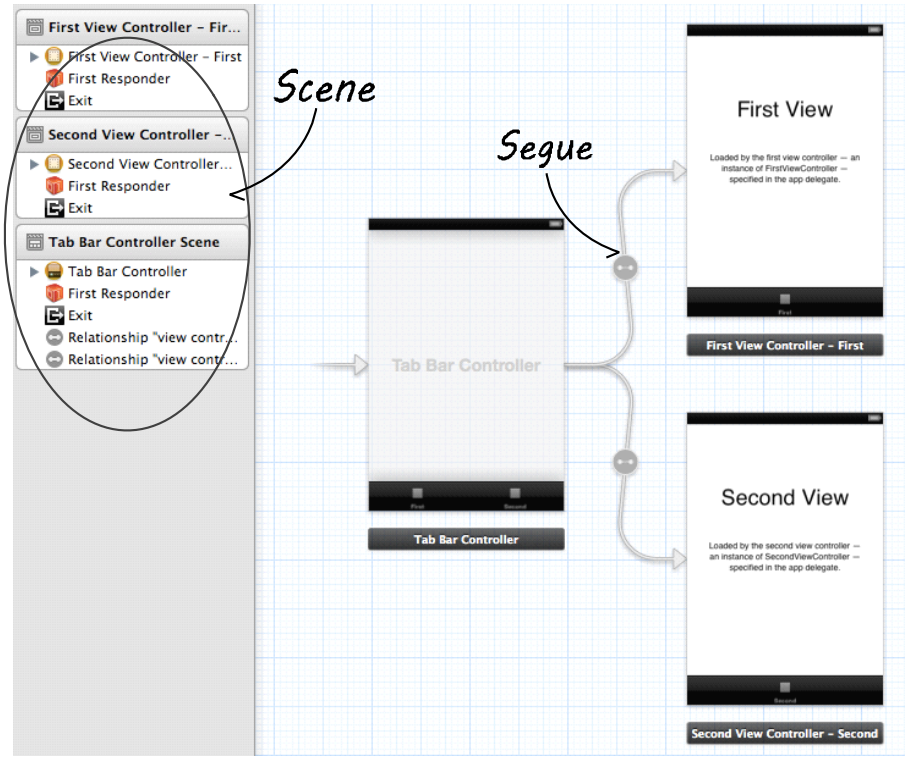

图2-18　故事板Scene和Segue

故事板中除了 Scene 和 Segue 以外，还有关于表视图的单元格一些新东西，我们也将在后面逐一介绍。

2.3 应用生命周期

作为应用程序的委托对象，AppDelegate 类在应用生命周期的不同阶段会回调不同的方法。首先让我们先了解一下 iOS 应用的不同状态及它们彼此间的关系，见图 2-19。

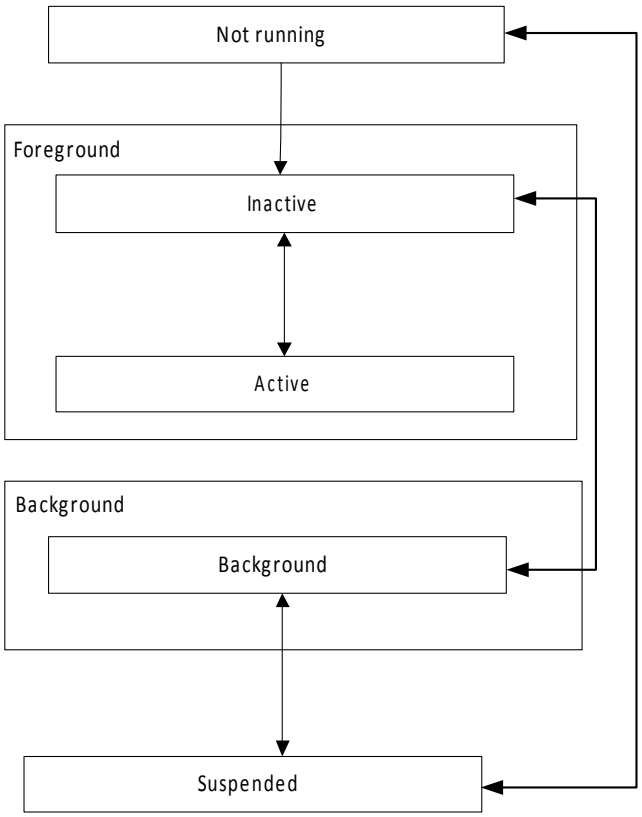

图2-19　iOS应用状态图

下面简要介绍一下 iOS 应用的 5 种状态。

- **Not Running（非运行状态）**：应用没有运行或被系统终止。
- **Inactive（前台非活动状态）**：应用正在进入前台状态，但是还不能接受事件处理。
- **Active（前台活动状态）**：应用进入前台状态，能接受事件处理。
- **Background（后台状态）**：应用进入后台后，依然能够执行代码。如果有可执行的代码就会执行代码，如果没有可执行的代码或者将可执行的代码执行完毕应用会马上进入挂起状态。
- **Suspended（挂起状态）**：处于挂起的应用进入一种“冷冻”状态，不能执行代码。如果系统内存不够，应用会被终止。

在应用状态跃迁过程中，iOS 系统会回调 AppDelegate 中的一些方法，并且发送一些通知。实际上，在应用的生命周期中使用到的方法和通知很多，这里挑出主要的几个来介绍，具体如表 2-2 所述。

表 2-2　状态跃迁过程中应用回调的方法和本地通知

方法	本地通知	说明
application:didFinishLau	UIApplicationDidFinis	应用启动并进行初始化时会调用该方法并发出通知。这个阶段会实例

nchingWithOptions:	hLaunchingNotification	化根视图控制器
applicationDidBecomeActive:	UIApplicationDidBecomeActiveNotification	应用进入前台并处于活动状态时调用该方法并发出通知。这个阶段可以恢复UI的状态（例如游戏状态等）
applicationWillResignActive:	UIApplicationWillResignActiveNotification	应用从活动状态进入到非活动状态时调用该方法并发出通知。这个阶段可以保存UI的状态（例如游戏状态等）
applicationDidEnterBackground:	UIApplicationDidEnterBackgroundNotification	应用进入后台时调用该方法并发出通知。这个阶段可以保存用户数据，释放一些资源（例如释放数据库资源等）
applicationWillEnterForeground:	UIApplicationWillEnterForegroundNotification	应用进入到前台，但是还没有处于活动状态时调用该方法并发出通知。这个阶段可以恢复用户数据
applicationWillTerminate:	UIApplicationWillTerminateNotification	应用被终止时调用该方法并发出通知，但内存清除时除外。这个阶段释放一些资源，也可以保存用户数据

为了便于观察应用程序的运行状态，我们为AppDelegate.m中的方法添加一些日志输出，具体代码如下：

```
@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
        didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    NSLog(@"%@",@"application:didFinishLaunchingWithOptions:");
    …… 
    return YES;
}
- (void)applicationWillResignActive:(UIApplication *)application
{
    NSLog(@"%@",@"applicationWillResignActive:");
}
- (void)applicationDidEnterBackground:(UIApplication *)application
{
    NSLog(@"%@",@"applicationDidEnterBackground:");
}
- (void)applicationWillEnterForeground:(UIApplication *)application
{
    NSLog(@"%@",@"applicationWillEnterForeground:");
}
- (void)applicationDidBecomeActive:(UIApplication *)application
{
    NSLog(@"%@",@"applicationDidBecomeActive:");
}
- (void)applicationWillTerminate:(UIApplication *)application
{
    NSLog(@"%@",@"applicationWillTerminate:");
}
@end
```

为了让大家更直观地了解各状态与其相应的方法、通知间的关系，下面我们以几个应用场景为切入点进行系统的分析。

2.3.1 非运行状态——应用启动场景

场景描述：用户点击应用图标的时候，可能是第一次启动这个应用，也可能是应用终止后再次启动。该场景的状态跃迁过程见图 2-20，共经历两个阶段三个状态：Not running→Inactive→Active。

- 在 Not running→Inactive 阶段，调用 application:didFinishLaunchingWithOptions:方法，发出 UIApplicationDidFinishLaunchingNotification 通知。
- 在 Inactive→Active 阶段，调用 applicationDidBecomeActive: 方法，发出 UIApplicationDidBecomeActiveNotification 通知。

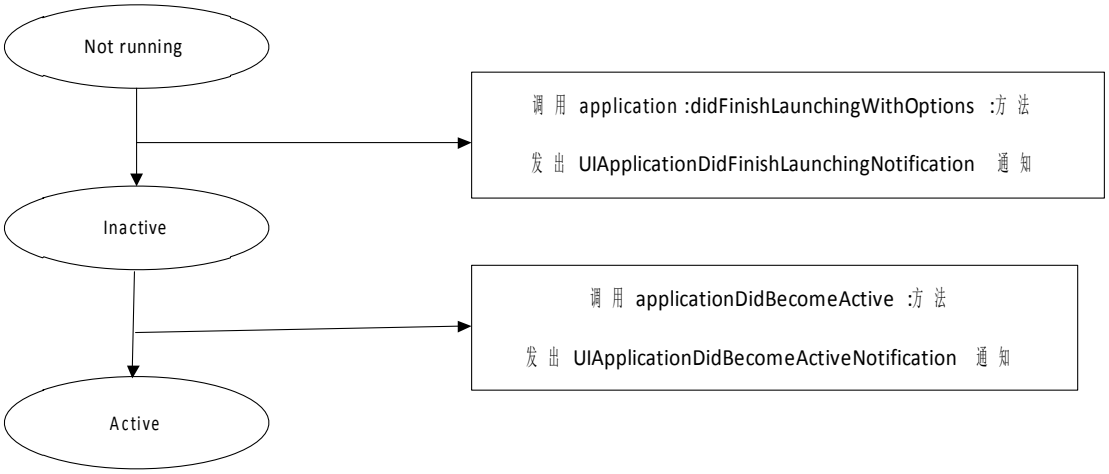

图2-20　非运行状态应用启动场景

2.3.2　点击 Home 键——应用退出场景

应用处于运行的状态即Active状态时，点击Home键或者有其他的应用导致当前应用中断。根据产品属性文件(如:HelloWorld-Info.plist)中的相关属性是否可以全部改成Application does not run in background（如图2-21所示），该场景的状态跃迁过程可以分成两种情况。如果采用文本编辑器打开HelloWorld-Info.plist文件该设置项对应的键是UIApplicationExitsOnSuspend。

Xcode产品属性文件HelloWorld-Info.plist中的相关属性Application does not run in background（如图2-21所示），如果采用文本编辑器打开HelloWorld-Info.plist文件对应的键是UIApplicationExitsOnSuspend。

UIApplicationExitsOnSuspend 键的具体设置见图 2-21。

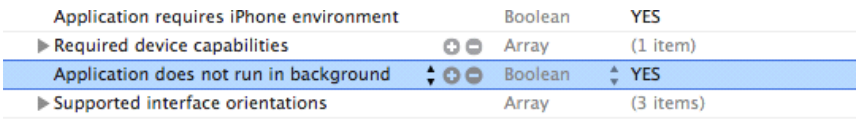

图2-21　UIApplicationExitsOnSuspend键设定

状态跃迁的第一种情况：应用可以在后台运行或者挂起，该场景的状态跃迁过程见图 2-22，共经历 3 个阶段 4 个状态：Active → Inactive → Background→Suspended。

- 在 Active → Inactive 阶段，调用 applicationWillResignActive: 方法，发出 UIApplicationWillResignActiveNotification 通知。
- Inactive → Background 阶段，应用从非活动状态进入到后台（不涉及我们要重点说明的方法和通知）。
- 在 Background→Suspended 阶段，调用 applicationDidEnterBackground:方法，发出

UIApplicationDidEnterBackgroundNotification 通知

图2-22　点击Home键应用退出场景

状态跃迁的第二种情况：应用不可以在后台运行或者挂起，其状态跃迁情况见图2-23，共经历4个阶段5个状态：Active → Inactive → Background→Suspended→Not running 。

- 在 Active → Inactivd 阶段：应用由活动状态转为非活动状态（不涉及我们要重点说明的方法和通知）。
- Inactive → Background 阶段，应用从非活动状态进入到后台（不涉及我们要重点说明的方法和通知）。
- 在 Background→Suspended 阶段：调用 applicationDidEnterBackground:方法，发出 UIApplicationDidEnterBackgroundNotification 通知。
- 在 Suspended → Not running 阶段，调用 applicationWillTerminate:方法，发出 UIApplicationWillTerminateNotification 通知。

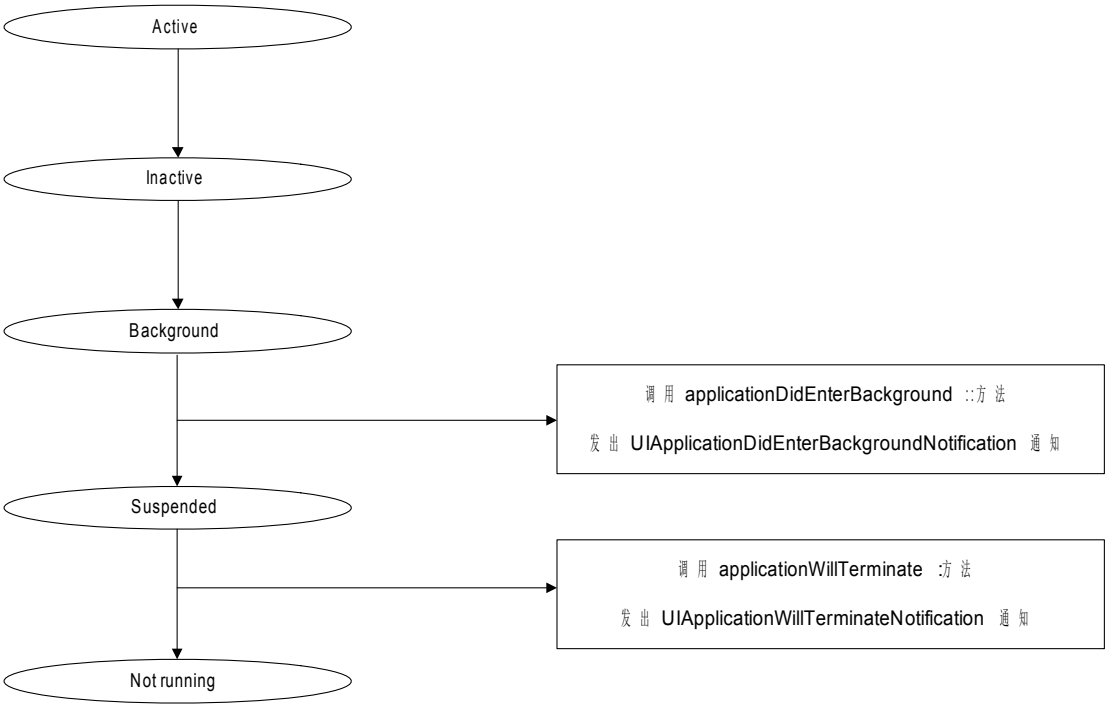

图2-23　点击Home键应用退出场景

iOS在4之前不支持多任务，点击Home键应用会退出并中断，而在iOS 4之后（包括iOS 4）操作系统能够支持多任务处理，点击Home键应用会进入后台但不会中断（内存不够的情况除外）。

应用在后台也可以进行部分处理工作，处理完成则进入挂起状态。

说明　双击Home键可以快速进入iOS多任务栏，如图2-24所示。在此处可以看到处于后台运行或挂起状态的应用，也可能有处于终止状态的应用驻留在这里。长按这些图标可删除这些应用以手动释放内存。

图2-24　iOS多任务栏

2.3.3　挂起重新运行场景

场景描述：挂起状态的应用重新运行。该场景的状态跃迁过程如图2-25所示，共经历3个阶段4个状态：Suspended → Background → Inactive → Active。

- Suspended → Background 阶段，应用从挂起状态进入后台（不涉及我们要讲述的这几个方法和通知）
- 在 Background→ Inactive 阶段，调用 applicationWillEnterForeground: 方法，发出 UIApplicationWillEnterForegroundNotification 通知。
- 在 Inactive→Active 阶段，调用 applicationDidBecomeActive: 方法，发出 UIApplicationDidBecomeActiveNotification 通知。

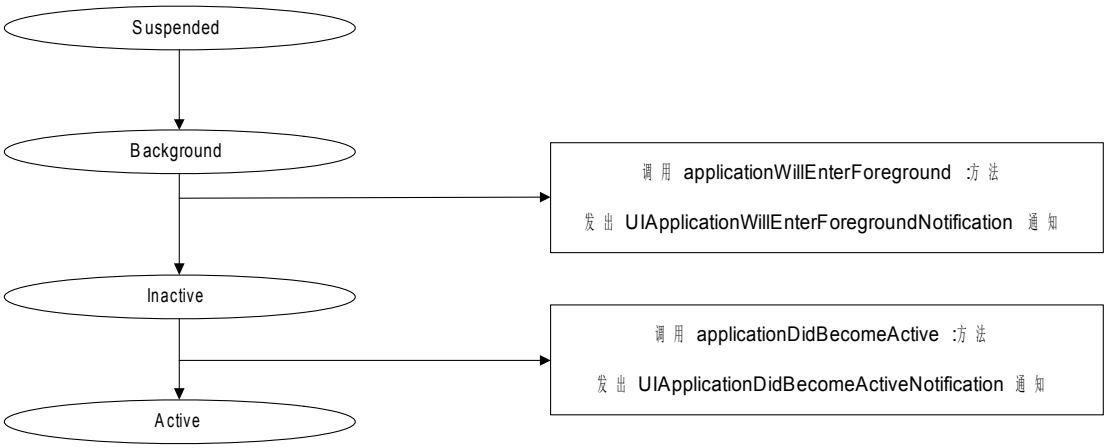

图2-25　挂起→重新运行场景

2.3.4 内存清除——应用终止场景

场景描述：在后台处理完成时则进入挂起状态，挂起状态是一种休眠状态，如果这个时候iOS发出低内存警告，为了满足其他的应用对于内存的需要，该应用就会被清除内存从而终止运行，该场景的状态跃迁见图2-26。

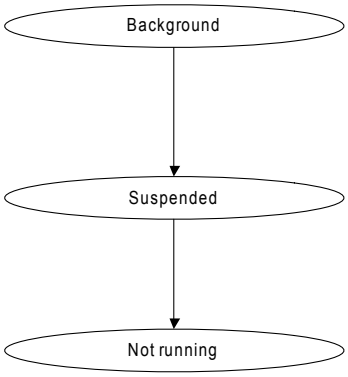

图2-26 内存清除终止场景

内存清除的时候应用终止运行。内存清除有两种情况，可能是系统强制清除内存也可能是由使用者从任务栏中手动清除（即删掉应用）。内存清除后如果应用再次运行，上一次的运行状态不会被保存，相当于应用第一次运行。

在内存清除场景下，应用不会调用任何方法也不会发出任何通知。

2.4 视图生命周期

视图是一个应用的重要组成部分，功能的实现与其息息相关，而视图控制器控制着视图，其重要性在整个应用中不言而喻。

2.4.1 视图生命周期与视图控制器关系

以视图的5种状态为基础，我们来系统的了解一下视图控制器的生命周期。在视图不同的生命周期，视图控制器会回调不同的方法，具体如图2-27所示。

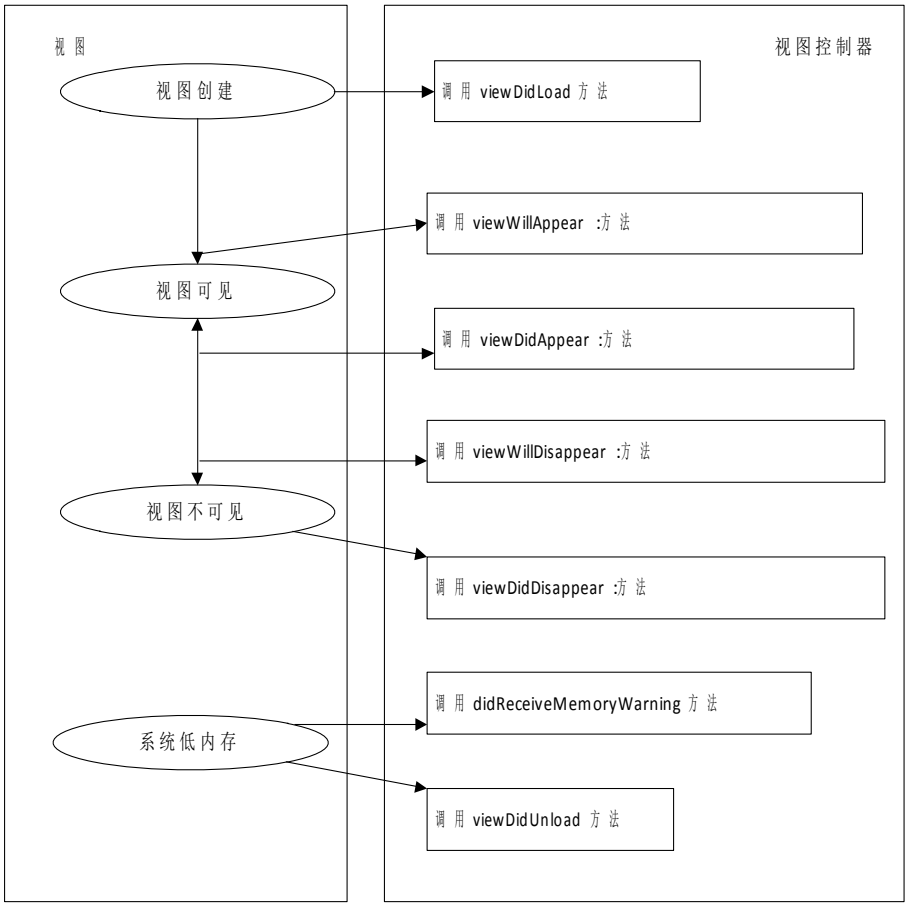

图2-27　视图控制器一些主要的方法

viewDidLoad方法：视图控制器已被实例化，在视图被加载到内存中的时候调用该方法，这个时候视图并未出现。在该方法中通常进行的是对所控制的视图进行初始化处理。

视图可见前后会调用viewWillAppear:方法和viewDidAppear:方法；视图不可见前后会调用viewWillDisappear:方法和viewDidDisappear:方法。 4个方法调用父类相应的方法以实现其功能，编码时该方法的位置可根据实际情况做以调整，参见如下代码：

```
-(void)viewWillAppear:(BOOL)animated
 {
     [super viewWillAppear:YES];
     ……
 }
```

viewDidLoad方法在应用运行的时候只调用一次，而这上述4个方法可以被反复调用多次，它们的使用很广泛但同时也具有很强的技巧性。例如：有的应用会使用重力加速计，重力加速计会不断轮询设备以实时获得设备在z轴、x轴和y轴方向的重力加速度。不断的轮询必然会耗费大量电能进而影响电池使用寿命，我们通过利用这4个方法适时地打开或者关闭重力加速计来达到节约电能的目的。怎么使用这4个方法才能做到“适时”是一个值得思考的问题。

iOS系统在低内存时情况下会调用didReceiveMemoryWarning:和viewDidUnload:方法。iOS6之后就不再使用viewDidUnload:，而仅支持didReceiveMemoryWarning：。didReceiveMemoryWarning：方法的主要职能是释放内存，包括视图控制器中的一些成员变量和视图的释放。现举例如下：

```
- (void)didReceiveMemoryWarning{
      self.button = nil;
        self.myStringD = nil;
        [myStringC release];
```

```
        [superdidReceiveMemoryWarning];
}
```

除了上述 5 个方法视图控制器还有很多其他的方法，随着学习的深入，我们会逐一向大家介绍。

2.4.2 iOS6 UI 状态保持和恢复

iOS 设计规范中要求，当应用退出的时候（包括被终止运行时候），画面中 UI 元素状态需要保持的，当再次进来的时候看状态与退出是一样的。iOS6 之后苹果提供以下 API 使得 UI 状态保持和恢复变得很容易了。

在 iOS6 中我们可以在 3 地方实现状态保持和恢复：

- 应用程序委托对象
- 视图控制器
- 自定义视图

为了演示这个功能实现，我们把基于故事板的 HelloWorld 工程改造一下，在画面中添加一个文本框，操作与 2.1 节添加 Label 操作类似相同，如图 2-28 所示，关于文本框的技术细节我们会在后面的章节介绍。

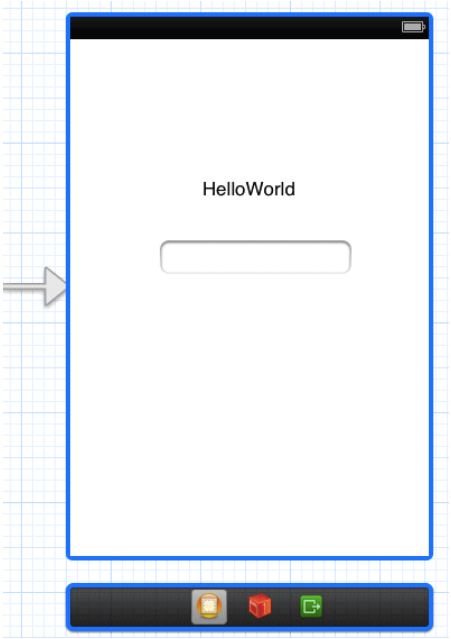

图 2-28　视图添加文本框

用户在文本框中输入一些内容，应用程序退出并且终止后再次进来的时候，文本框中还会保持原来输入的内容。然后在 IB 的 Scene 中选中 View Controller，打开右边的标识检查器，如图 2-29 所示，设置 Restoration ID（恢复标识）为 viewController。

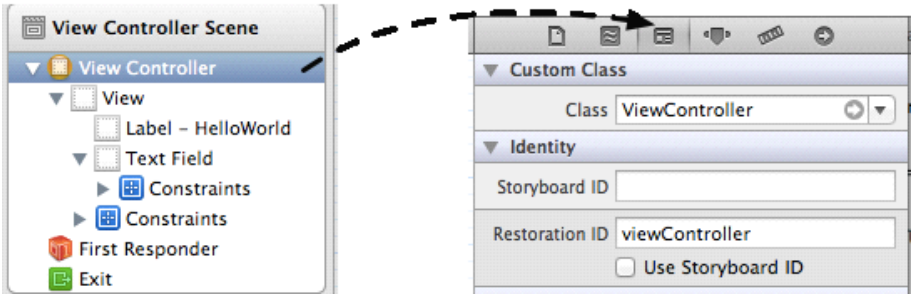

图 2-29　设置控制器的恢复标识

恢复标识是 iOS6 为了实现 UI 状态保持和恢复添加的设置项目。我们还需要在应用程序委托对象 AppDelegate 代码部分做一些修改，添加的代码如下：

```
-(BOOL) application:(UIApplication*)application shouldSaveApplicationState:(NSCoder *)coder
{
    return YES;
}

-(BOOL) application:(UIApplication*)application shouldRestoreApplicationState:(NSCoder *)coder
{
    return YES;
}

- (void)application:(UIApplication *)application willEncodeRestorableStateWithCoder:(NSCoder *)coder
{
    [coder encodeFloat:2.0 forKey:@"Version"];
}

- (void)application:(UIApplication *)application didDecodeRestorableStateWithCoder:(NSCoder*)coder
{
    float lastVer = [coder decodeFloatForKey:@"Version"];
     NSLog(@"lastVer= %f",lastVer);
}
```

其中 application:shouldSaveApplicationState:在应用退出的时候调用，负责控制是否允许保存状态，返回 YES 情况是可以保存，NO 是不保存。

application:shouldRestoreApplicationState:是应用启动时候调用，负责控制是否恢复上次退出的时候的状态，返回 YES 情况是可以恢复，NO 是不恢复。

application:willEncodeRestorableStateWithCoder:方法是保存时候调用，在这个方法中实现 UI 状态或数据的保存，其中[coder encodeFloat:2.0 forKey:@"Version"]是保存简单数据。

application:didDecodeRestorableStateWithCoder:方法是恢复时候调用，在这个方法中实现 UI 状态或数据的恢复，其中[coder decodeFloatForKey:@"Version"]语句是恢复上次保存的数据。

要想实现具体画面中控件的保持和恢复，还需要在它视图控制器中添加一些代码，ViewController.m 中添加的代码如下：

```
-(void)encodeRestorableStateWithCoder:(NSCoder *)coder
{
    [super encodeRestorableStateWithCoder:coder];
    [coder encodeObject:self.txtField.textforKey:kSaveKey];
}

-(void)decodeRestorableStateWithCoder:(NSCoder *)coder
{
    [super decodeRestorableStateWithCoder:coder];
    self.txtField.text = [coder decodeObjectForKey:kSaveKey];
}
```

在 iOS6 之后视图控制器都添加了两个：encodeRestorableStateWithCoder:和 decodeRestorableStateWithCoder:用来实现该控制器中的控件或数据的保存和恢复。其中 encodeRestorableStateWithCoder: 方法是在保存时候调用，[coder encodeObject:self.txtField.text forKey:kSaveKey]语句是按照指定的键保存文本框的内容，decodeRestorableStateWithCoder: 方法是在恢复时候调用，[coder decodeObjectForKey:kSaveKey]是恢复文本框内容时候调用，保存和恢复事实上就是向一个归档文件中编码和解码的过程。

2.5 设置产品属性

在前面讲解应用生命周期时候，为了禁止应用在后台运行，我们将 HelloWorld-Info.plist 文件中的 Application does not run in background 属性修改为 YES（即 UIApplicationExitsOnSuspend = YES）。这项操作就属于产品属性的设置。在 Xcode 中产品与 Target 直接相关，而 Target 与 Project 直接相关。

2.5.1 Xcode 中 Project 和 Target

打开 HelloWorld 工程时，我们会看到如图 2-30 所示的界面。产品属性包括 Project 和 Target 两块内容。一个工程只有一个 Project，但可以有一个或多个 Target。

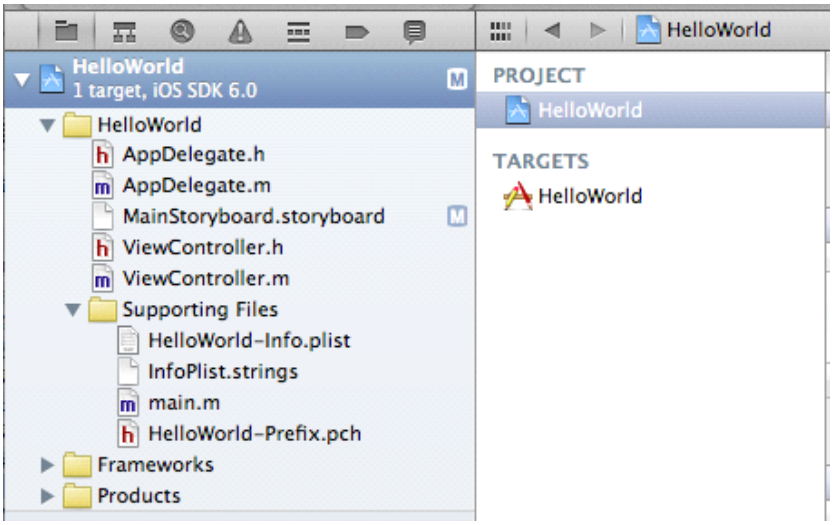

图2-30 Xcode的Project和Target

我们所创建的 HelloWorld 只有一个 Target，下面我们为之前使用故事板实现的 HelloWorld 工程增加一个 Target。

首先，依次选择 File→New→Target 菜单，之后会弹出一个模板选择对话框，如图 2-31 所示。

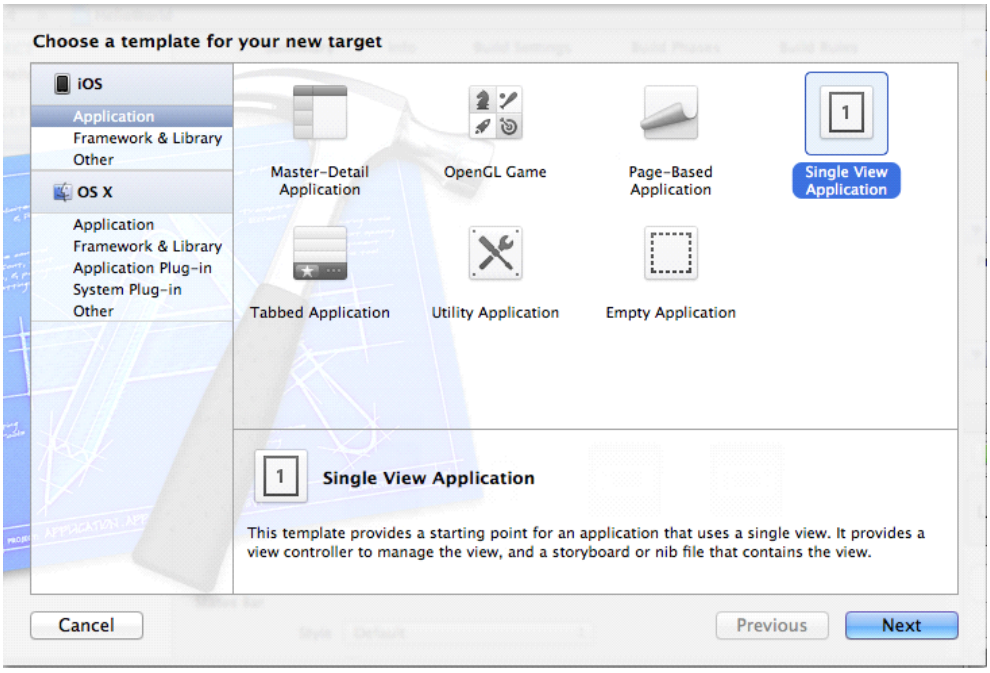

图2-31　选择Target模板对话框

这里选择的模板与新建工程时选择的模板完全一样，然后点击 Next 按钮，将出现如图 2-32 所示的对话框。

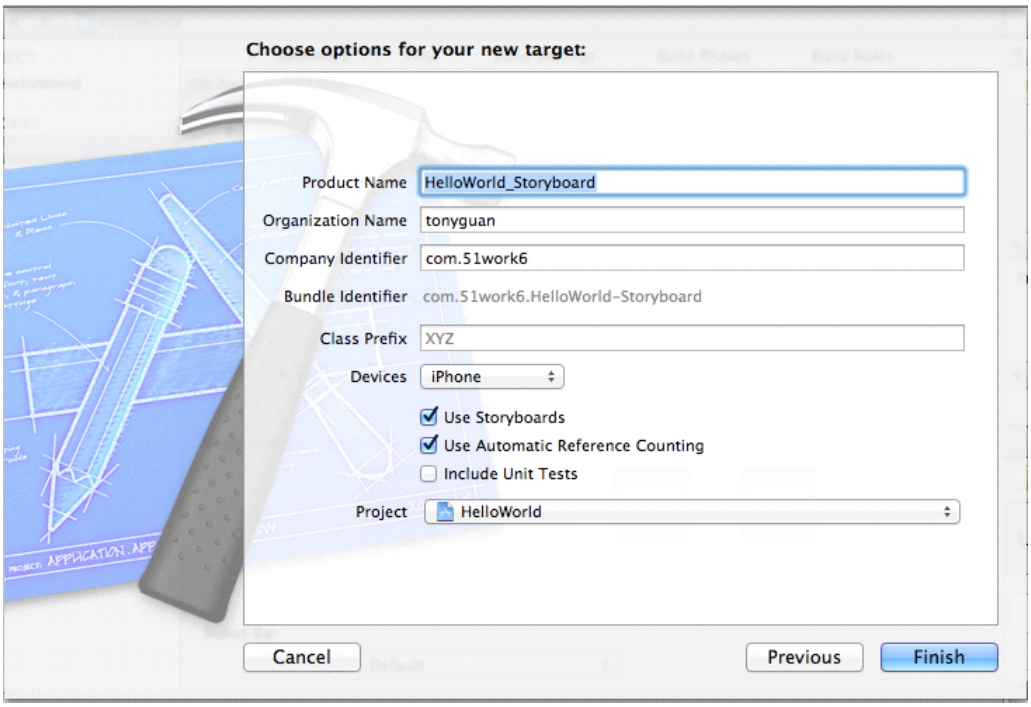

图2-32　Target一些选项设定

根据情况逐一进行设定后，点击 Finish 按钮，现在我们已经成功为 HelloWorld 新增了一个 Target。查看左边的导航面板，可以发现右边有两个 Target，并同时生成一套完整的文件——main.m、AppDelegate、ViewController 和 MainStoryboard.storyboard，它们独立于原来的 Target 而存在，如图 2-33 所示。

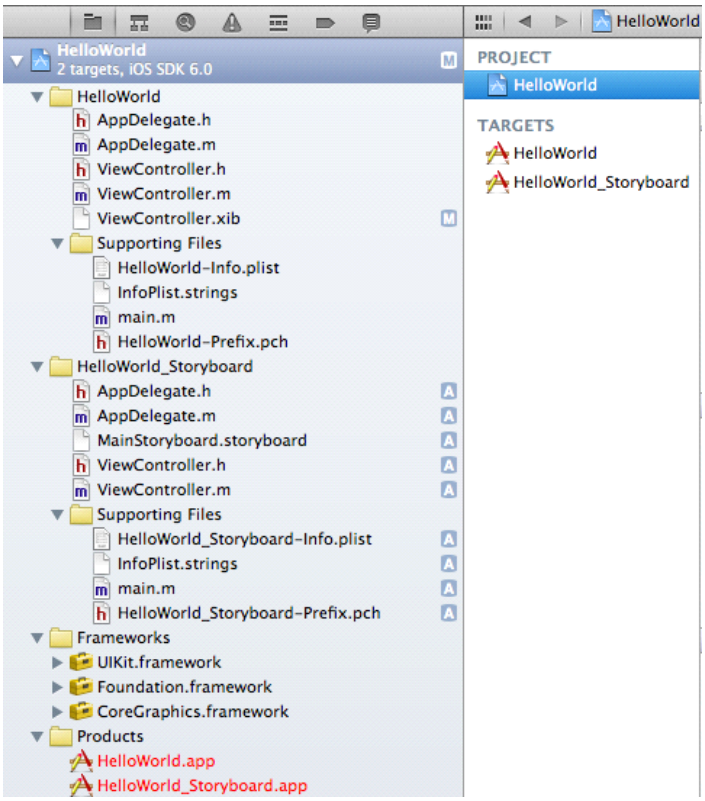

图2-33　新创建Target

要指定运行哪一个 Target，可以通过选择不同的 Scheme 来实现。如图 2-34 所示，在 Xcode 的左上角选择 HelloWorld_Storyboard→iPhone 6.0 Simulator，就可以在 iPhone 6.0 Simulator 上运行 HelloWorld_Storyboard Target 了。

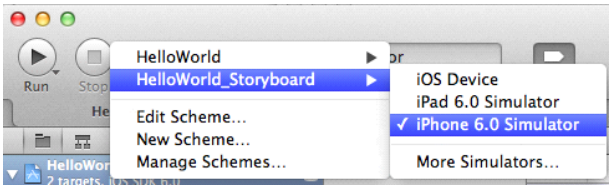

图2-34　选择Scheme

2.5.2　设置常用的产品属性

Target 继承了 Project，但对于 Target 和 Project 下都有的设置项可根据需要可对 Target 进行再设置，此设置可覆盖 Project 的设置。

Project 中的属性设置相对比较简单，大家可以参考官方的相关资料。这里为大家介绍 Target 下的 4 个常用的产品属性。

1.　**设定屏幕方向**

如图 2-35 所示，在导航面板中选择 HelloWorld_Storyboard Target，然后再右侧选择 Summary 选项卡，可以发现下面的 Supported Interface Orientations 区域中有 4 个按钮，它们代表设备支持的 4 个方向，其中按下状态（深色）代表支持该指定方向。

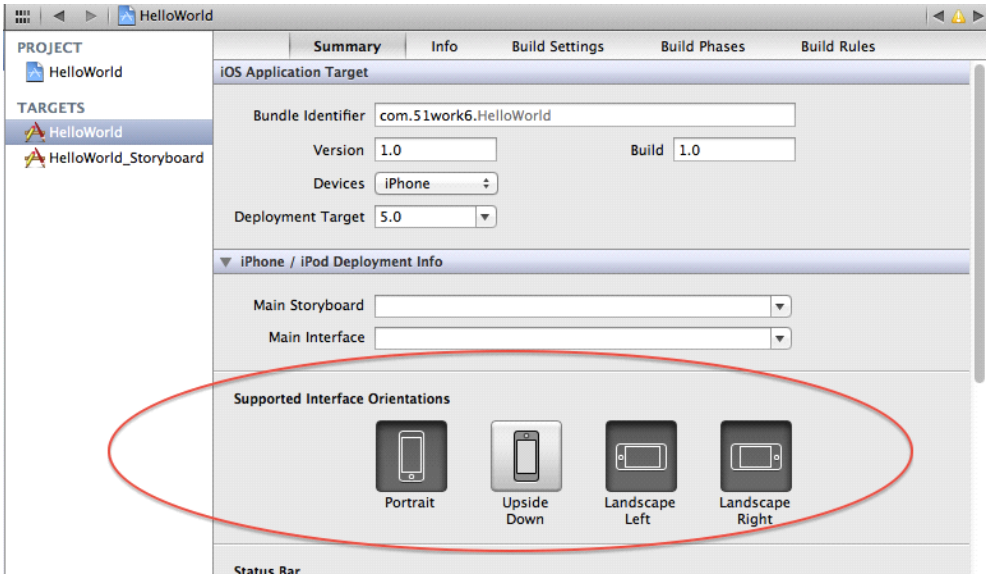

图2-35　屏幕选择方向支持设定

说明　鉴于Target的属性存储在HelloWorld_Storyboard-Info.plist文件中，也可以直接修改HelloWorld_Storyboard-Info.plist文件以完成属性各项的设置，如图2-36所示。

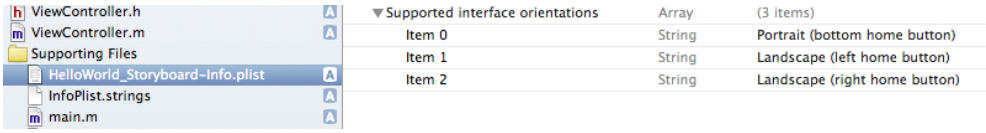

图2-36　在HelloWorld_Storyboard-Info.plist文件中设定屏幕方向

2.　设置应用图标

如图 2-37 所示，在 App Icons 区域中，右击选择图片文件，然后左击为应用添加图标。第二个设置项是用来为具备 Retina 显示技术（iPhone 4 以后才支持）的设备提供图标的，操作同前。图标对其图片文件的尺寸和名称要求很严格，第一个设置项的图片大小为 57x57 像素，名称为 Icon.png；第二个设置项的图片大小为 114x114 像素，名称为 Icon@2x.png。

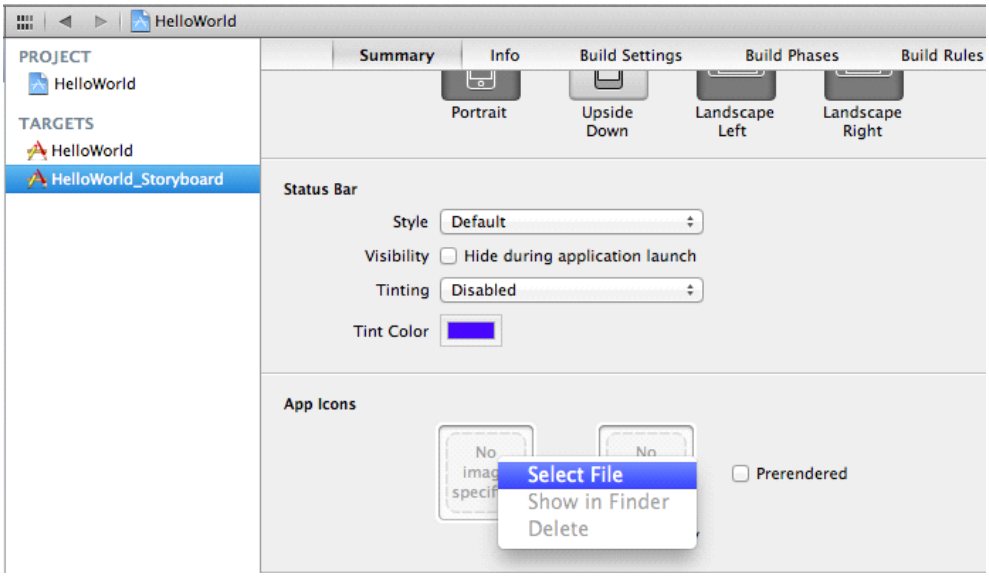

图2-37　应用图标的设定

说明　应用图标名可以不做修改，直接采用原来的自定义名称，因为Xcode会自动给图标改名，但文件的格式（png）和大小必须按照要求设定。

3.　设置启动屏幕

启动屏幕是应用启动时一闪而过的图片，它也在 Summary 选项卡中设置。如图 2-38 所示，在 Launch Images 区域中，右击选择图片文件。对于普通的 iPhone 设备，这个图片的大小为 320×480 像素，文件以 Default.png 命名。右边的选择框是为 Retina 显示技术设备提供图片的，大小是 640 x 960 像素，文件命名为 Default@2x.png。

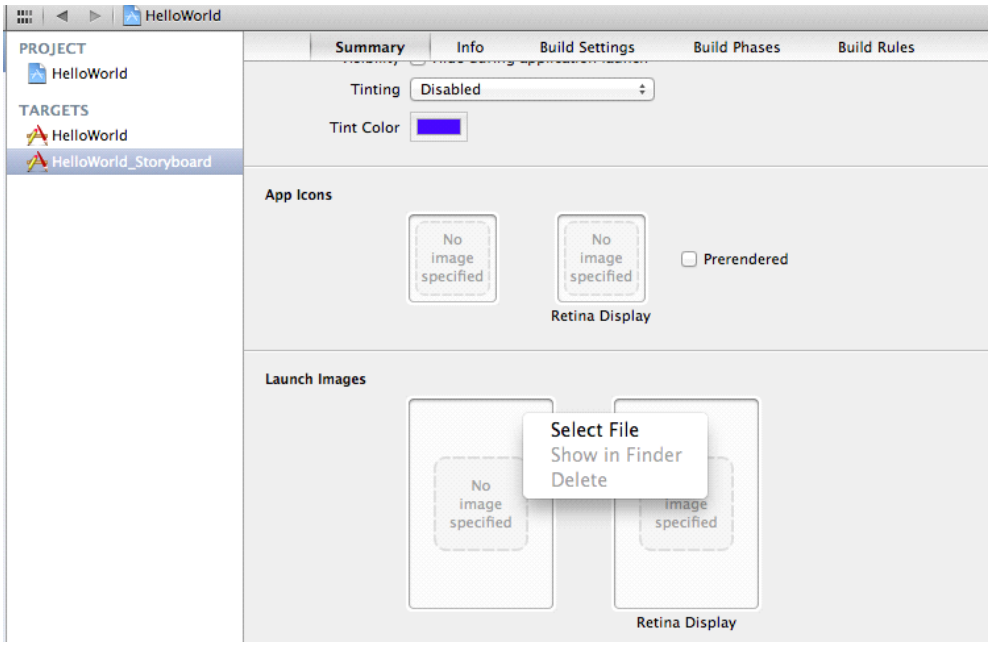

图2-38　应用屏幕的设定

说明　启动屏幕图片的命名可以不做修改，因为Xcode会自动给它改名，但文件的格式（png）和大小必须按照要求进行设定。iPad设备会有4种图片，命名也与iPhone不同。

4.　设置设备支持情况

我们可以让应用支持 iPhone 设备或 iPad 设备，或者同时支持 iPhone 和 iPad 设备。如图 2-39 所示，在 Summary 选项卡中的 iOS Application Target 区域中单击 Device 下拉列表，选择 iPhone 或者 iPad 也或者 Universal，其中 Universal 是同时支持 iPhone 和 iPad 设备。

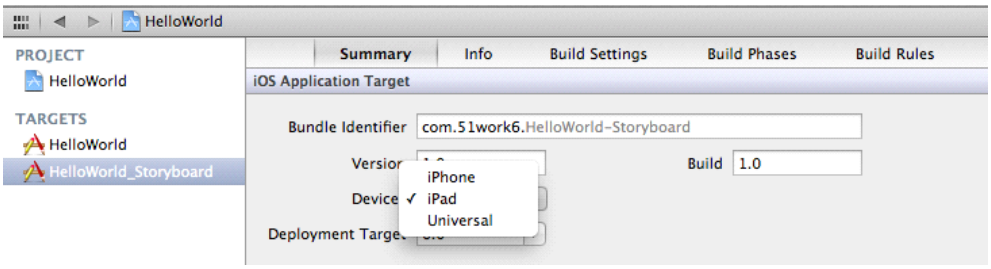

图2-39　设置设备支持情况

事实上，产品的相关属性还有很多，我们会在以后继续了解。

2.6 iOS API 的介绍

苹果的 iOS API 在不同版本间有很多变化，本书采用的是 iOS 6。本节中，我们会介绍 iOS 6 有哪些 API，如何使用这些 API 的帮助文档，以及如何使用官方案例。

2.6.1 API 概述

iOS 的整体架构图如图 2-40 所示，分为 4 层——Cocoa Touch 层、Media 层、Core Services 层和 Core OS 层，下面概要介绍一下这 4 层。

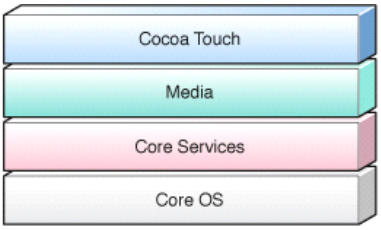

图2-40 iOS整体架构图

1、Cocoa Touch 层

该层提供了构建 iOS 应用的一些基本系统服务（如多任务、触摸输入、推送通知等）和关键框架（见表 2-3）。

表 2-3 Cocoa Touch 层包括的框架

框架	前缀	说明
Address Book UI	AB	访问用户的联系人信息
Event Kit UI	EK	访问用户的日历事件数据
Game Kit	GK	提供能够进行点对点的网络通信的API
iAd	AD	在应用中嵌入广告
Map Kit	MK	在应用中嵌入地图和地理信息编码等
Message UI	MF	提供与发送E-mail相关的API
Twitter	TW	提供发送Twitter的接口
UIKit	UI	提供UI类

2、Media 层

Media 层提供了图形、音频、视频和 AirPlay 技术，包括的框架如表 2-4 所示。

表 2-4 Media 层包括的框架

框架	前缀	说明
Assets Library	AL	提供访问用户的图片和视频的接口
AudioToolbox	Audio	录制或播放音频、音频流以及格式转换
AudioUnit	Audio, AU	提供使用内置音频单元服务，以及音频处理模块
AV Foundation	AV	提供播放与录制音频和视频的Objective-C接口
Core Audio	Audio	提供录制、制作、播放音频的C语言接口
Core Graphics	CG	提供Quartz 2D接口
Core Image	CI	提供操作视频和静态图像的接口
Core MIDI	MIDI	提供用于处理MIDI数据低层的API
Core Text	CT	提供渲染文本和处理字体的简单、高效的C语言接口
Core Video	CV	提供用于处理音频和视频的API
Image I/O	CG	包含一些读写图像数据类
GLKit	GLK	包含了构建复杂OpenGL ES应用的Objective-C实用类
Media Player	MP	包含全屏播放接口
OpenAL	AL	包含了OpenAL（跨平台的音频）的C语言接口
OpenGL ES	EAGL, GL	包含OpenGL ES（跨平台的2D/3D图形库）的C语言接口
Quartz Core	CA	提供动画接口类

3、Core Services 层

该层提供了 iCloud、In-App Purchase（应用内购买）、SQLite 数据库和 XML 支持等技术，包括的主要框架如表 2-5 所示。

表 2-5　Core Services 层包括的框架

框架	前缀	说明
Accounts	AC	用于访问用户的Twitter账户（iOS 5之后才有此API）
AddressBook	AB	访问用户的联系人信息
AdSupport	AS	获得iAD广告标识
CFNetwork	CF	提供了访问Wifi网络和蜂窝电话网络的API
Core Data	NS	提供管理应用数据的ORM接口
CoreFoundation	CF	它是iOS开发中最基本的框架，包括数据集
Core Location	CL	提供定位服务的API
Core Telephony	CT	提供访问电话基本信息的API
CoreMedia	CM	提供AV Foundation框架使用的底层媒体类型。可以对音频或视频创建及展示进行精确控制
CoreMotion	CM	接收和处理重力加速计以及其它的运动事件
Core Telephony	CT	提供访问电话基本信息的API
Event Kit	EK	访问用户的日历事件数据
Foundation	NS	为Core Foundation 框架的许多功能提供Objective-C封装，是

		Objective-C最为基本框架
MobileCoreServices	UT	定义统一类型标识符 (UTI)使用的底层类型
Newsstand Kit	NK	提供在后台下载杂志和新闻API接口（iOS 5之后才有此API）
Pass Kit	PK	提供访问各种优惠券的API（iOS 5之后才有此API）
QuickLook	QL	该框架可以预览无法直接查看的文件内容，例如：打开PDF文件。
Social	SL	提供社交网络访问API，中国区提供新浪微博API（iOS 6之后才有此API
Store Kit	SK	提供处理应用内置收费的资金交易
SystemConfiguration	SC	可用于确定设备的网络配置。例如：使用该框架判断Wi-Fi或者蜂窝连接是否正在使用中，也可以用于判断某个主机服务是否可以使用

4、Core OS 层

该层提供了一些低级功能，开发中一般不直接使用它。该层包括的主要框架如表 2-6 所示。

表 2-6　Core OS 层包括的框架

框架	前缀	说明
Accelerate	AC	访问重力加速计API
Core Bluetooth	CB	访问低能耗蓝牙设备API
External Accessory	EA	访问硬件附件API接口
Security	gss	提供一组安全相关的服务
Security	CSSM, Sec	管理证书、公钥、私钥和安全信任策略API

2.6.2　如何使用 API 帮助

对于一个初学者来说，学会在Xcode中使用API帮助文档是非常重要的。下面我们通过一个例子介绍API帮助文档的使用。

在编写HelloWorld程序时候，我们可以看到ViewController.m的代码，如下所示：

```
@implementation ViewController
```

```
- (void)viewDidLoad
{
    [super viewDidLoad];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
}
@end
```

对于didReceiveMemoryWarning方法，我有很多困惑，这时候需要自己查找帮助文档。如果只是简单查看帮助信息，可以双击选择didReceiveMemoryWarning方法，然后选择右边的快捷帮助检查器，如图2-41所示。

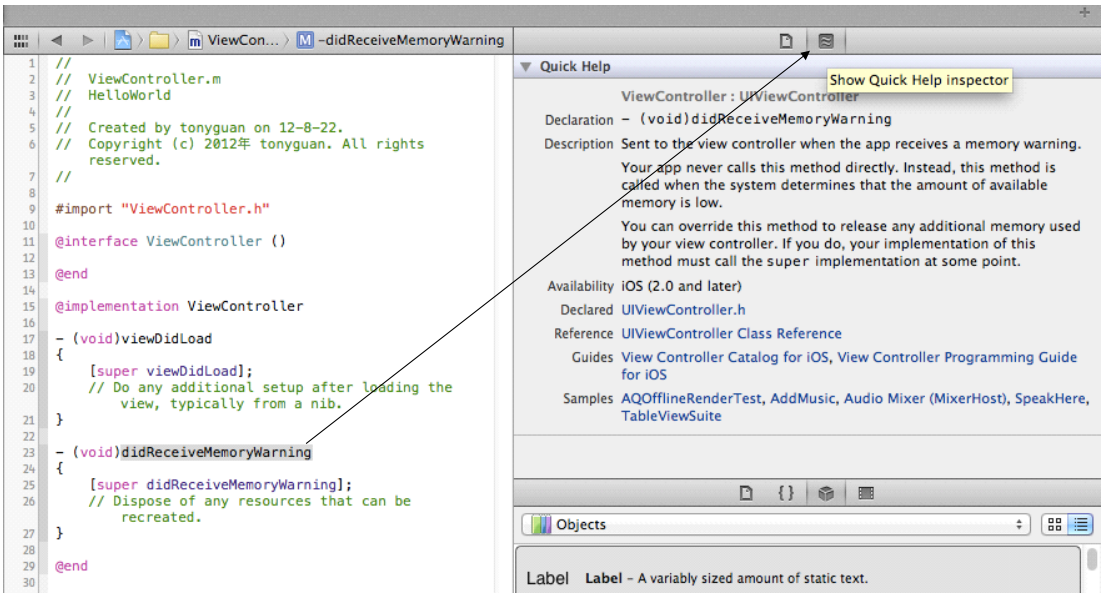

图2-41　Xcode快捷帮助检查器

在Xcode快捷帮助检查器窗口中，我们可以看到有关这个方法的描述，其中包括使用的iOS版本、相关主题以及一些有关案例。这里需要说明的是，如果需要查看官方的案例，直接从这里下载即可。

如果想查询比较完整的、全面的帮助文档，可以按住Alt键双击didReceiveMemoryWarning方法名，这样就会打开一个Xcode的API帮助文档，如图2-42所示。

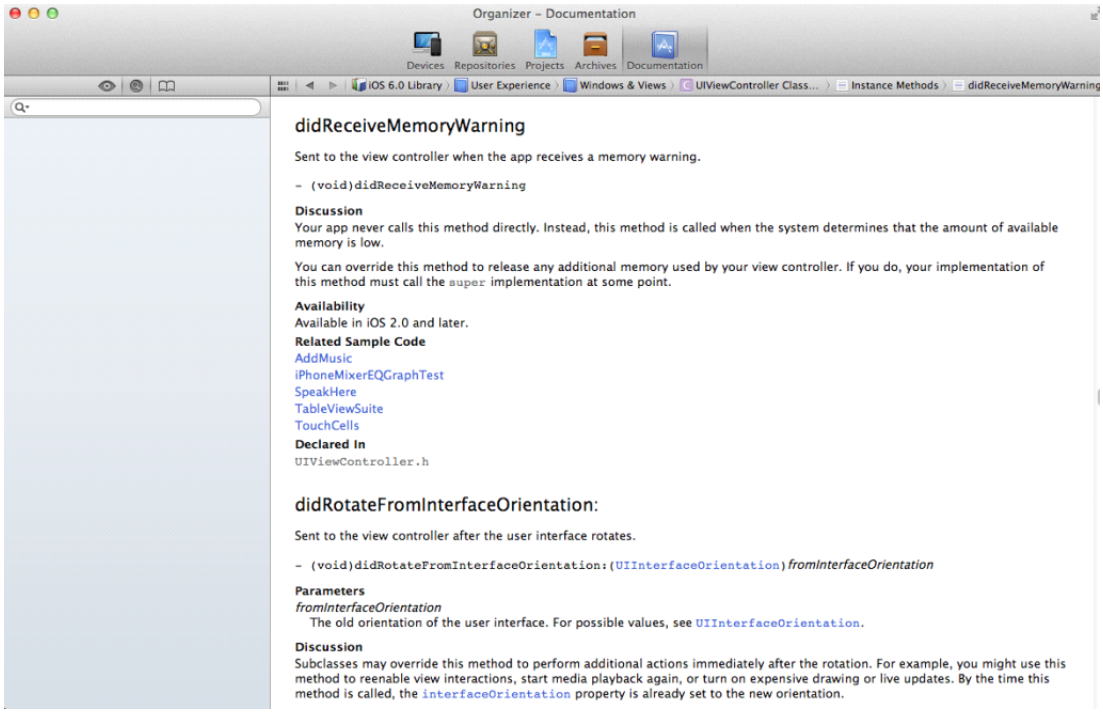

图2-42　Xcode帮助窗口

API 帮助文档还提供给我们一些官方案例（如图 2-43），选择案例会打开一个新的窗口，如图 4-43 所示，在这里可以查看案例介绍，点击按钮 Open Project，可以打开并下载这个案例工程。

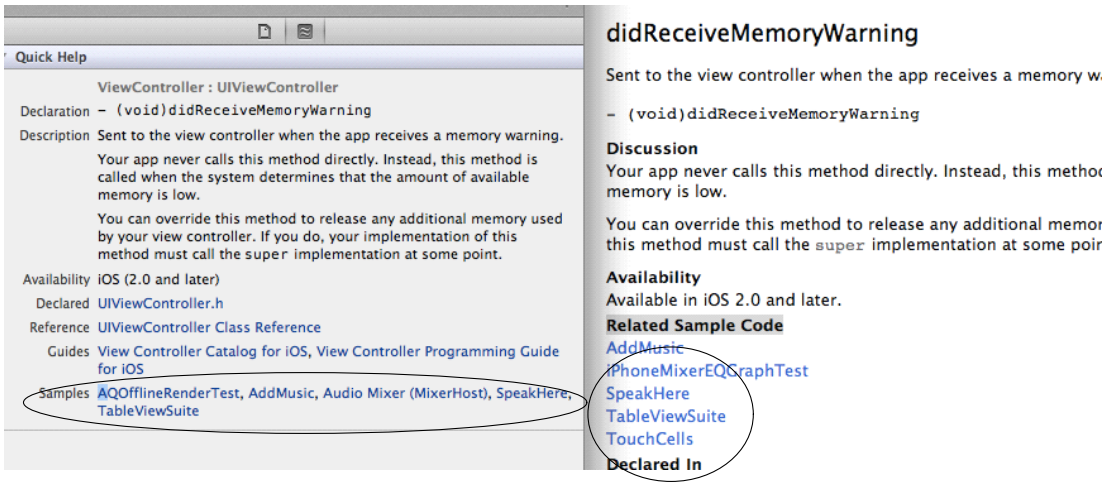

图2-43　官方案例

本章小结

在本章中，我们首先介绍了如何使用nib和故事板技术创建工程，通过对nib和故事板技术的使用，大家对二者的优缺点和彼此间的差异有了一定的了解。接着通过HelloWorld我们讨论了iOS工程模板、应用的运行机制和生命周期、视图控制器的生命周期，然后介绍了4项常用产品属性的设置。最后我们向大家介绍了API帮助文档和官方案例的使用。