

# 马上着手开发 iOS 应用程序

ltryee

Published  
with GitBook



# 目錄

---

1. [Introduction](#)
2. [设置](#)
3. [教程：基础](#)
4. [应用程序开发过程](#)
5. [设计用户界面](#)
6. [定义交互](#)
7. [Tutorial\\_UI](#)
8. [Incorporating\\_the\\_Data](#)
9. [Using\\_Design\\_Patterns](#)
10. [Tutorial\\_Add\\_Data](#)

《马上着手开发 iOS 应用程序》给 iOS 开发带来一个完美开局。在 Mac 上，您可以创建在 iPad、iPhone 和 iPod touch 上运行的 iOS 应用程序。本指南四个简短的部分为您构建自己的首个应用程序提供了入门指导，包括需要的工具、主要概念以及助您上路的最佳实践。



在前三部分中，每一部分的结尾都附有教程，有助于实践所学内容。当您完成最后一篇教程时，一个简单的待办事项列表应用程序也将随之创建完毕。

根据本指南构建了自己的首个应用程序，并考虑尝试下一步时，请阅读第四部分。这一部分探究了您可能会在下一个应用程序中考虑采用的技术和框架。您将逐步吸引客户的关注，促使您推出更好的产品。

本指南会悉心带您完成构建简单应用程序的每一步，但如果您具有计算机编程的基础知识，特别是面向对象编程的知识，则将对您的理解和掌握大有裨益。

## 获取工具

在着手开发精彩的应用程序之前，请先设置好开发环境，并确保工具齐备。




开发 **iOS** 应用程序，您需要：

- Mac 电脑，运行 OS X 10.8 (Mountain Lion) 或更高版本
- Xcode
- iOS SDK

Xcode 是 Apple 的集成开发环境 (IDE)。Xcode 包括源代码编辑器、图形用户界面编辑器和许多其他功能。iOS SDK 扩展了 Xcode 工具集，包含 iOS 开发专用的工具、编译器和框架。

您可以从 Mac 上的 App Store 中免费下载最新版本的 Xcode。（下载 Xcode 需要 OS X v10.8。如果您使用的是较早版本的 OS X，请升级。）Xcode 中包含了 iOS SDK。

### 下载最新版本的 Xcode

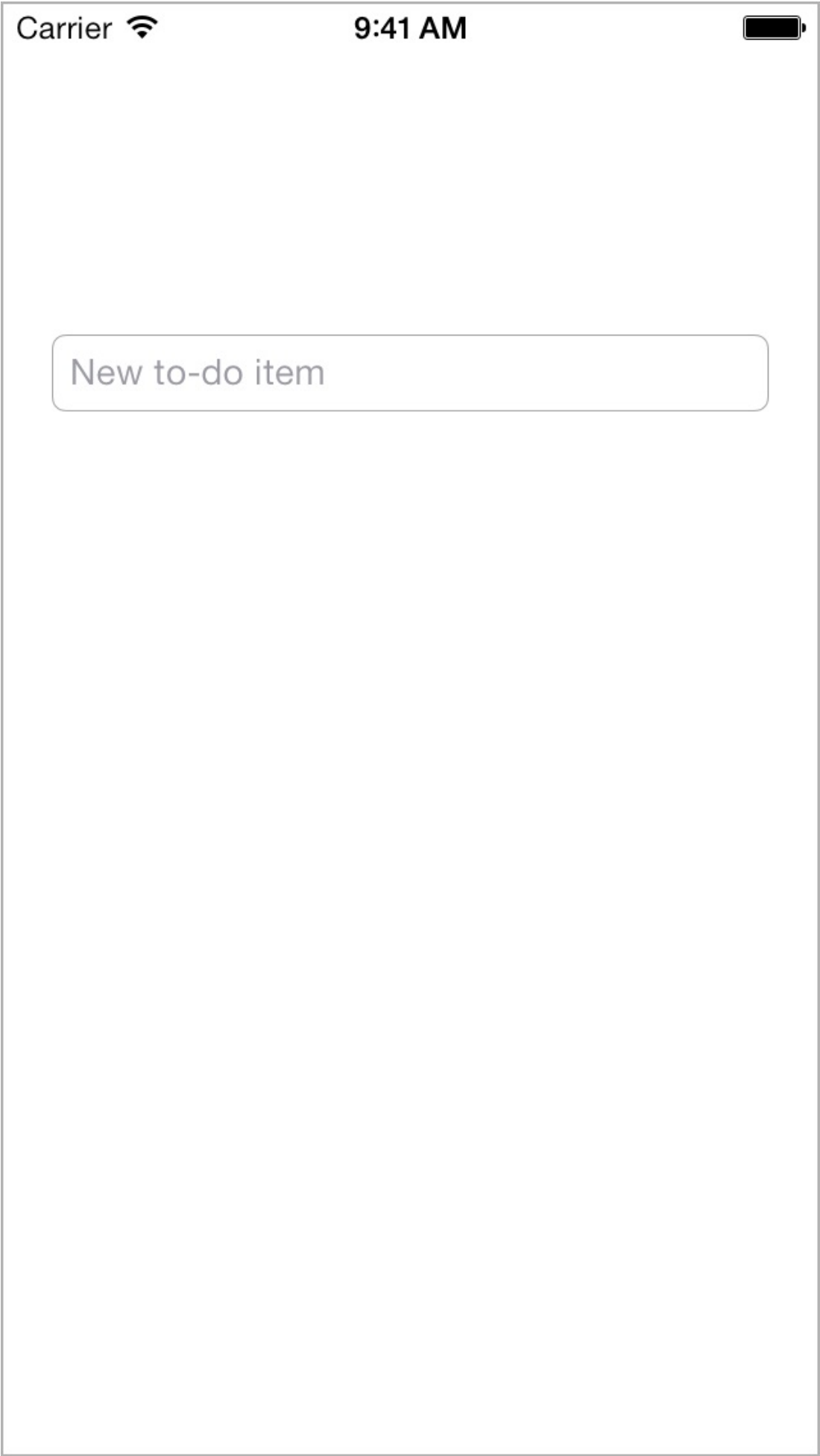
1. 请打开 Mac 上的 App Store 应用程序（默认位于 Dock 中）。
2. 在右上角的搜索栏中，键入 Xcode，然后按下 Return 键。
3. 点按“免费” Xcode 将下载到您的 /Applications 目录中。

本教程描述了什么是应用程序、创建简单用户界面的过程，以及如何添加自定义行为，将界面转变成可运行的应用程序。

遵循本教程，可了解 iOS 应用程序开发的基础内容，包括：

- 如何使用 Xcode 来创建和管理项目
- 如何识别 Xcode 项目的关键部分
- 如何将标准用户界面元素添加到应用程序
- 如何构建和运行应用程序

完成教程后，您会得到类似于下图的应用程序：



开发 iPad 应用程序的工具和技术与 iPhone 完全相同。为简单起见，本教程只针对 iPhone。教程使用 Xcode 5.0 和 iOS SDK 7.0。

## 创建新项目

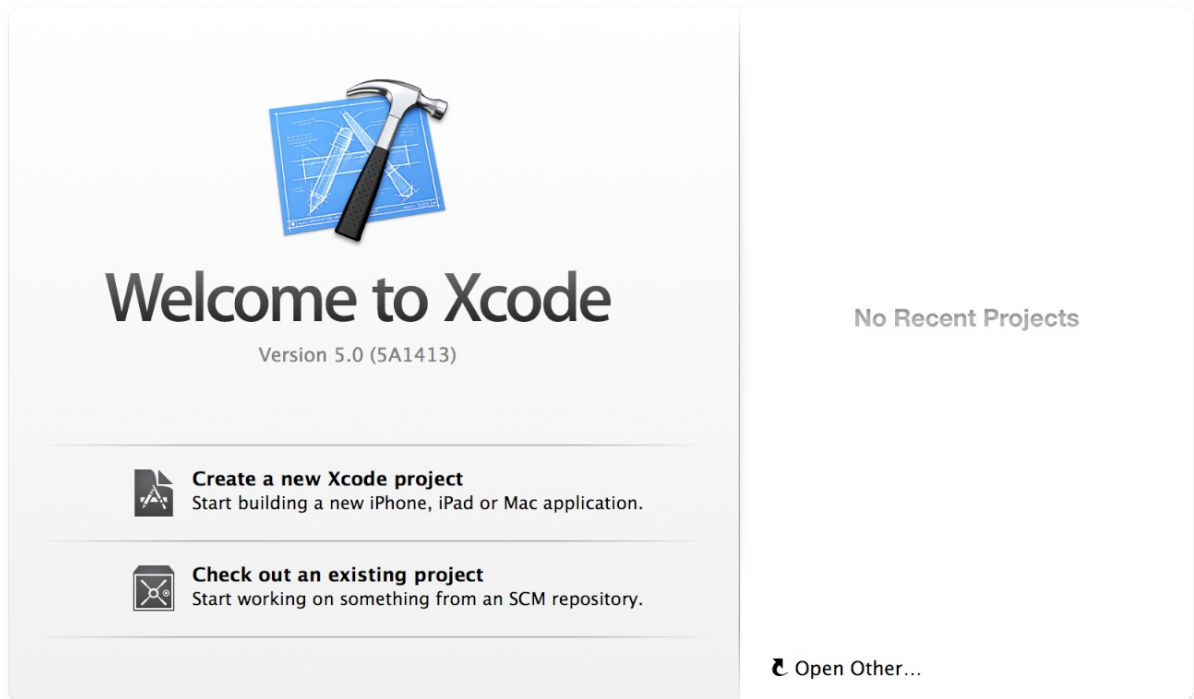
要开发应用程序，首先请创建一个新的 Xcode 项目。

Xcode 随附了几个内建应用程序模板，可用于开发常见的 iOS 应用程序，如游戏、基于标签浏览的应用程序和基于表格视图的应用程序。这些模板大都预先配置了界面和源代码文件，可作为您进行开发工作的起点。本教程会从最基础的模板开始：Empty Application。

使用 Empty Application 模板有助于理解 iOS 应用程序的基本结构，以及如何将内容呈现给用户。了解完所有组件的工作方式后，您可以将其他模板用在自己的应用程序上，来节省一些配置时间。

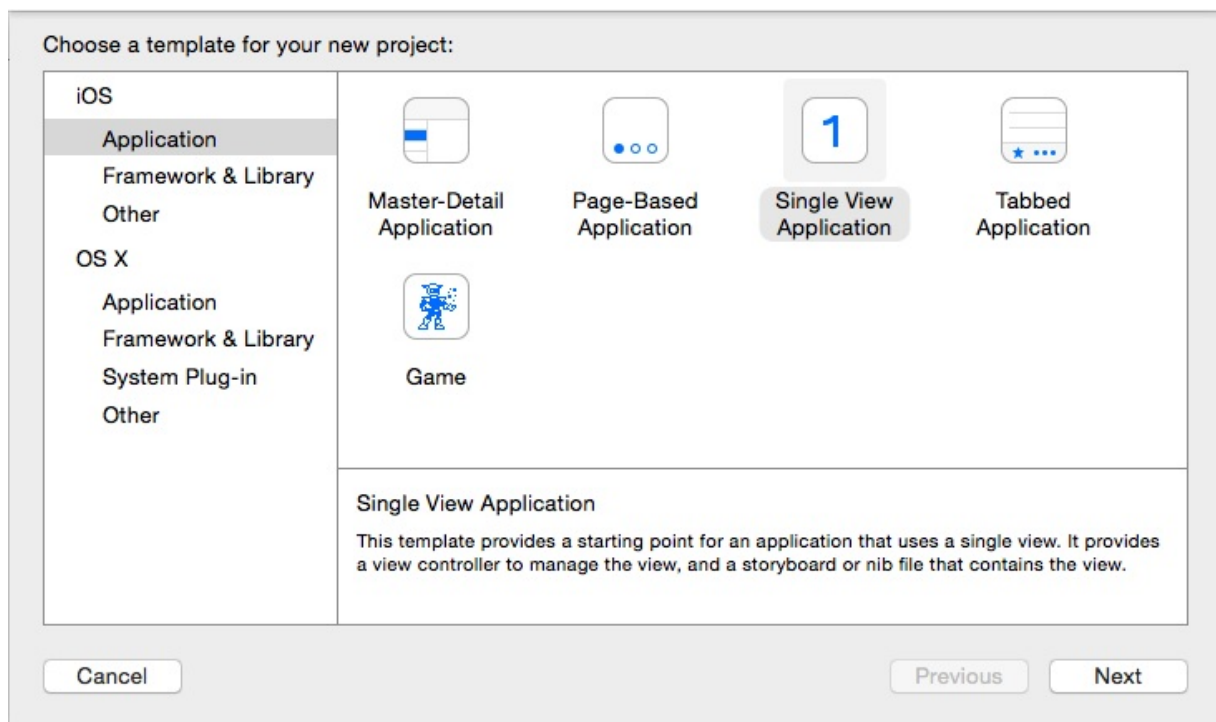
### 创建新的空项目

1. 从 /Applications 目录打开 Xcode。Xcode 欢迎窗口会出现。

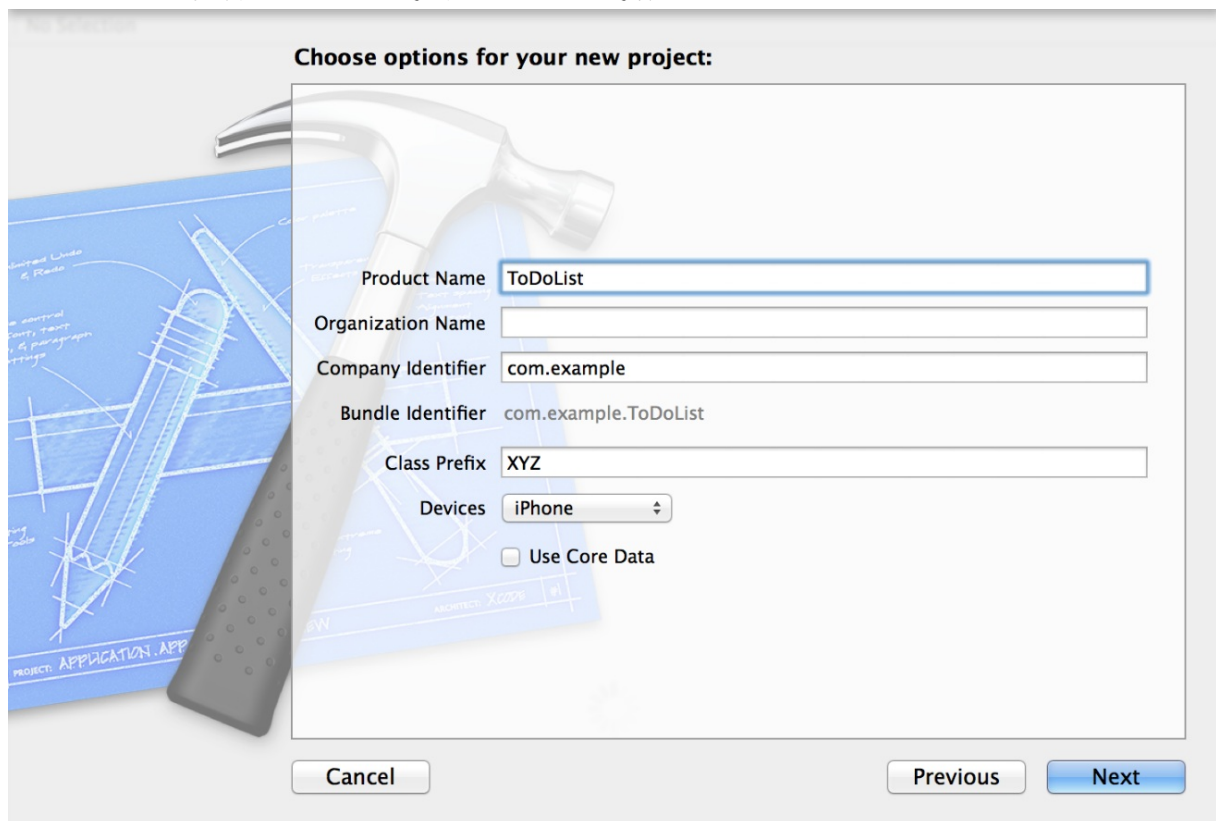


如果出现的是项目窗口，而不是欢迎窗口，请不要着急；这说明您可能曾在 Xcode 中创建或打开过项目。您只需在接下来的步骤中，使用菜单项来创建项目。

2. 在欢迎窗口中，点按“Create a new Xcode project”（或选取“File”>“New”>“Project”）。Xcode 将打开一个新窗口并显示对话框，让您从中选取一个模板。



3. 在对话框左边的 iOS 部分，选择“Application”。
4. 在对话框的主区域中，点按“Single View Application”，然后点按“Next”。
5. 在出现的对话框中，给应用程序命名并选取应用程序的其他选项。

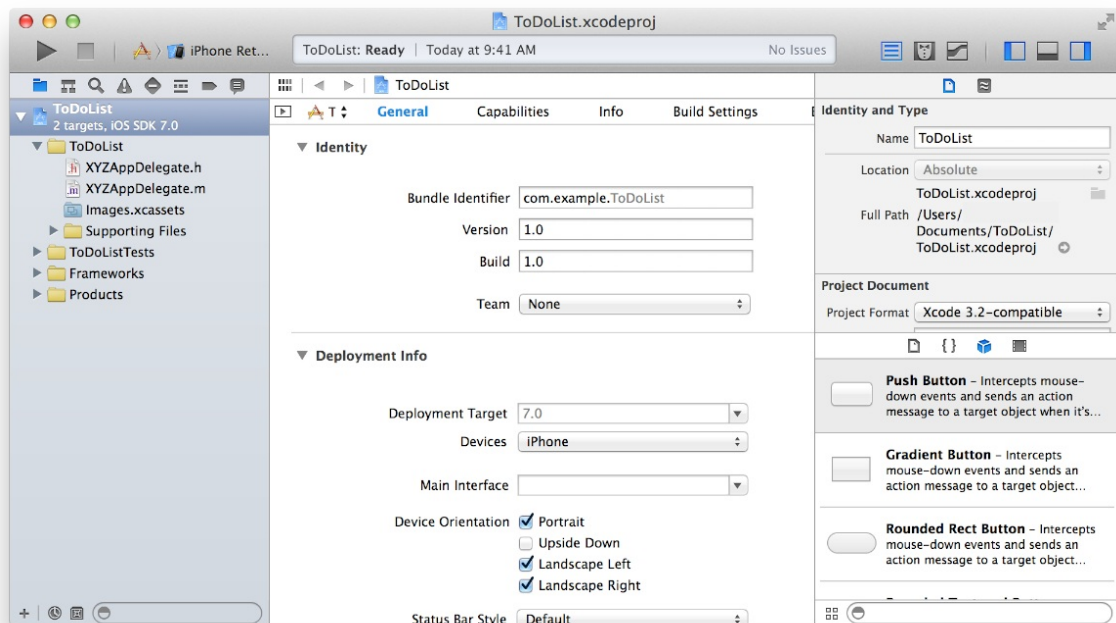


请使用以下值：

- Product Name : ToDoList
- Xcode 会使用您输入的产品名称给您的项目和应用程序命名。
- Company Identifier : 您的公司标识符（如果有）。如果没有，请使用 com.example。
- Class Prefix : XYZ Xcode 会使用类前缀名称来命名为您创建的类。Objective-C 类的名称必须是代码中唯一的词，并区别于任何可能在框架或捆绑包中使用的词。为使类名称保持唯一性，通常要为所有类添加前缀。Apple 已经为框架类保留了两个字母组成的前缀，所以请使用三个字母或更长的前缀。



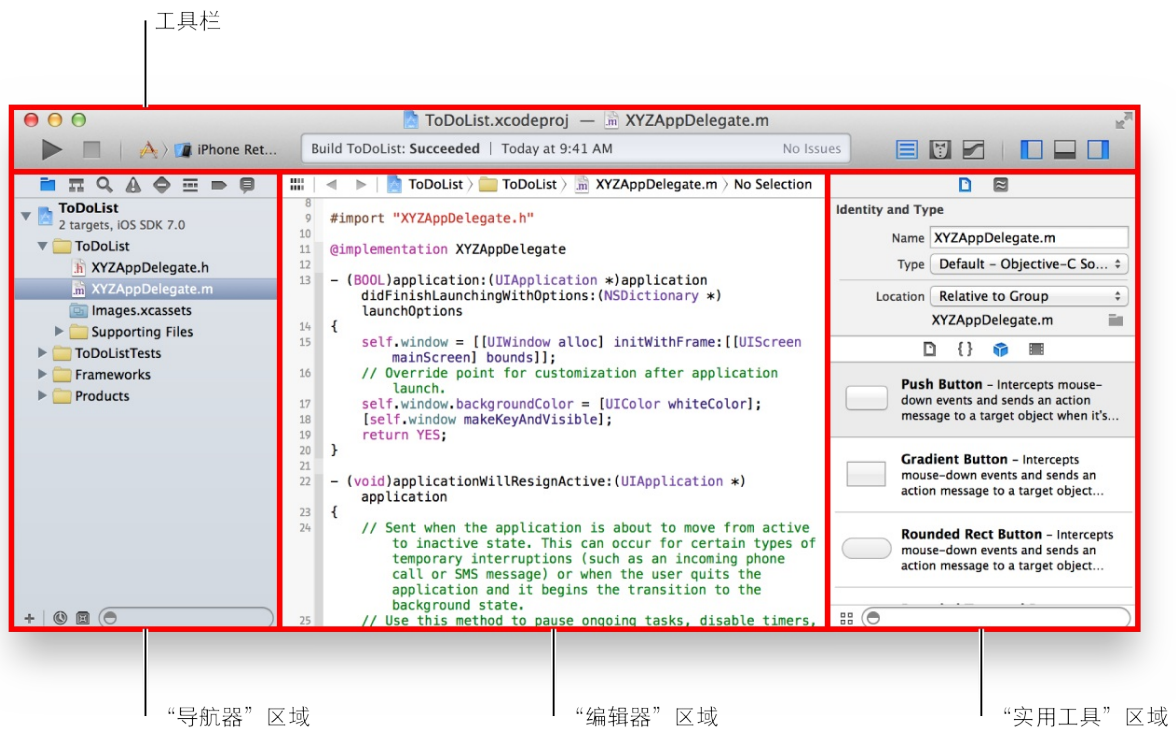
6. 从“Devices”弹出式菜单中选取“iPhone”。前文中已经提到，使用 iPhone 界面创建应用程序是最简单的入门方式。为 iPad 创建应用程序或创建通用应用程序的技术与此相同。
7. 点按“Next”。
8. 在出现的对话框中，选取项目的存放位置，然后点按“Create”。Xcode 会在工作区窗口中打开新项目，窗口的外观类似：



## 熟悉 Xcode

Xcode 包括了您创建应用程序时所需的一切。它不仅整理了创建应用程序时所需的文件，还提供了代码和界面元素编辑器，可让您构建和运行应用程序，并拥有强大的集成调试程序。

请花几分钟时间来熟悉 Xcode 工作区窗口。在接下来的整个教程中，您将会用到下面窗口中标识出的控制。点按不同的按钮，体验一下它们的工作方式。如果要了解有关界面某个部分的更多信息，请阅读其帮助文章。方法是按住 Control 键点按 Xcode 中的区域，然后从出现的快捷菜单中选取文章。



## 运行 iOS Simulator

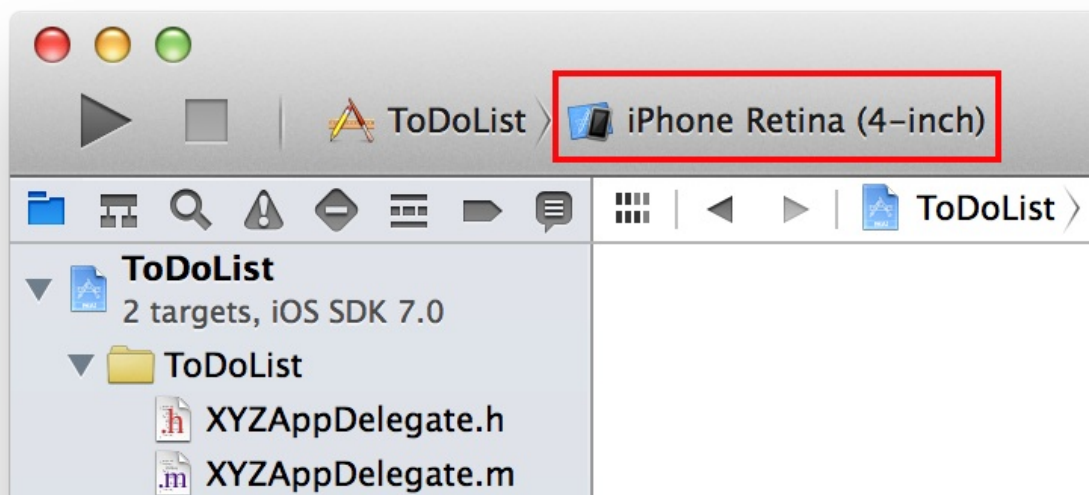
由于项目是基于 Xcode 模板创建的，因此基本的应用程序环境已经自动为您设置好了。即使没有编写任何代码，也可以构建和运行 Empty Application 模板，而无需进行任何额外的配置。

构建和运行您的应用程序，可以使用 Xcode 自带的 iOS Simulator 应用程序。顾名思义，iOS Simulator 可模拟在 iOS 设备上运行应用程序，让您初步了解它的外观和行为。

它可模拟多种不同类型的硬件，包括屏幕大小不同的 iPad、iPhone 等等。因此，您可以模拟在任何一款开发目标设备上运行应用程序。在本教程中，我们选择使用“iPhone Retina (4-inch)”。

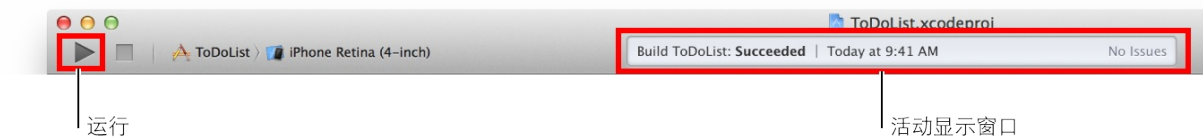
### 在 iOS Simulator 中运行应用程序

1. 从 Xcode 工具栏的“Scheme”弹出式菜单中选取“iPhone Retina (4-inch)”。



继续浏览菜单，查看 iOS Simulator 中的其他硬件选项。

2. 点按 Xcode 工具栏左上角的“Run”按钮。



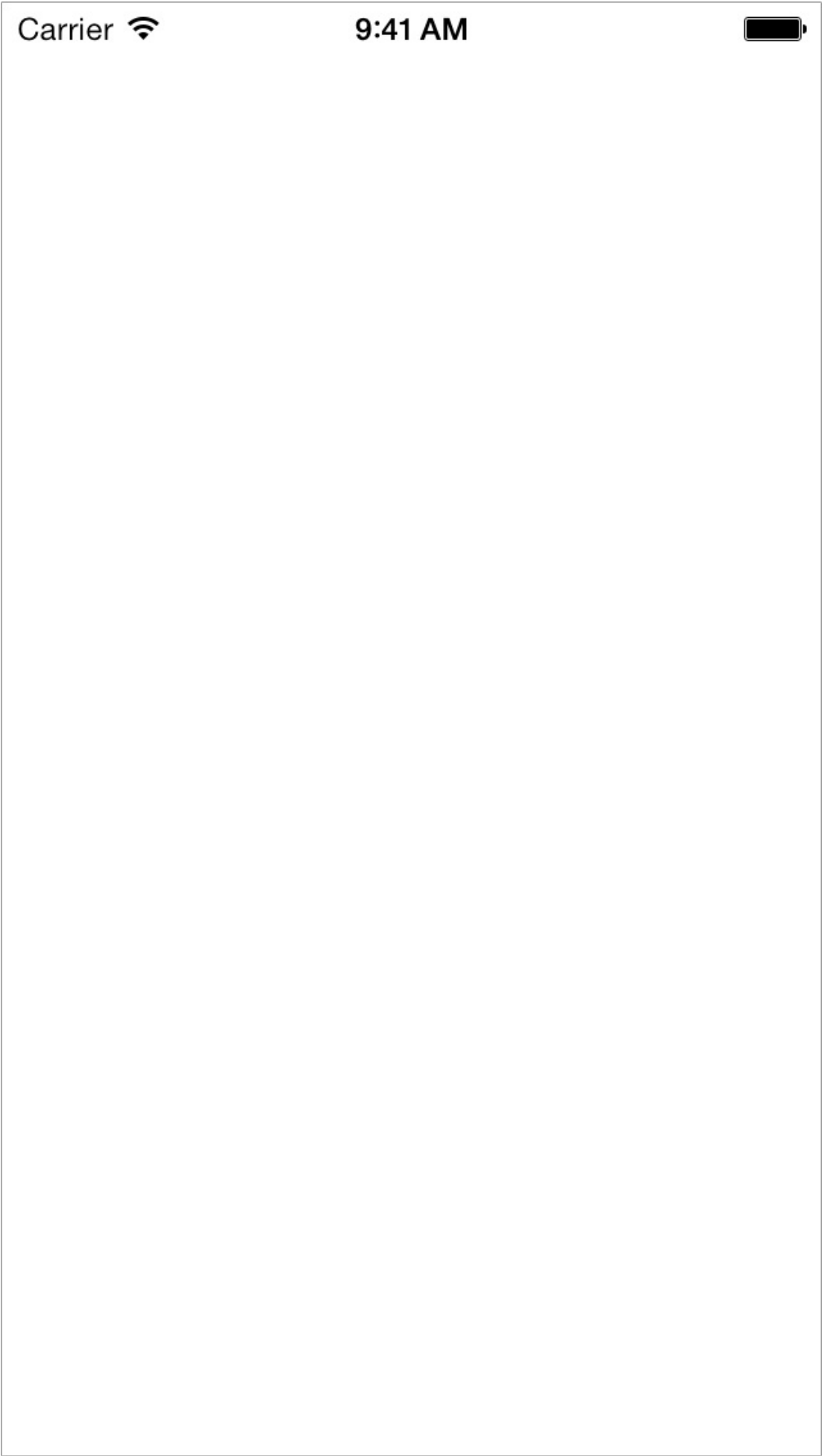
或者，可以选取“Product”>“Run”（或按下 Command-R）。

如果是首次运行应用程序，Xcode 会询问您是否要在 Mac 上启用开发者模式。开发者模式可让 Xcode 访问特定的调试功能，无需每次都输入密码。请决定是否要启用开发者模式，然后按照提示操作。如果选取不启用，可能稍后会要求您输入密码。本教程假定已启用了开发者模式。

3. 构建过程完成后，请看 Xcode 工具栏。Xcode 会在工具栏中间的活动显示窗口中显示有关构建过程的消息。

Xcode 完成项目生成后，iOS Simulator 会自动启动。首次启动时可能需要几分钟时间。

iOS Simulator 会按照您的指定，以 iPhone 模式打开。在模拟的 iPhone 屏幕上，iOS Simulator 会打开您的应用程序。（如果此时在 Xcode 调试程序中看到一则信息，请不必担心，稍后的教程中会有解释。）



一如其名，Empty Application 模板并未包括过多的代码，仅会显示一个白色的屏幕。其他模板会有更多复杂的行为，因此在扩展模板制作自己的应用程序之前，先要弄清楚模板的用处，这一点很重要。而要做到这一点，一个很好的方式，就是先不做任何修改，直接运行模板。

探索完应用程序后，请选取“iOS Simulator”>“Quit iOS Simulator”（或按下 Command-Q）来退出 iOS Simulator。

## 检查源代码

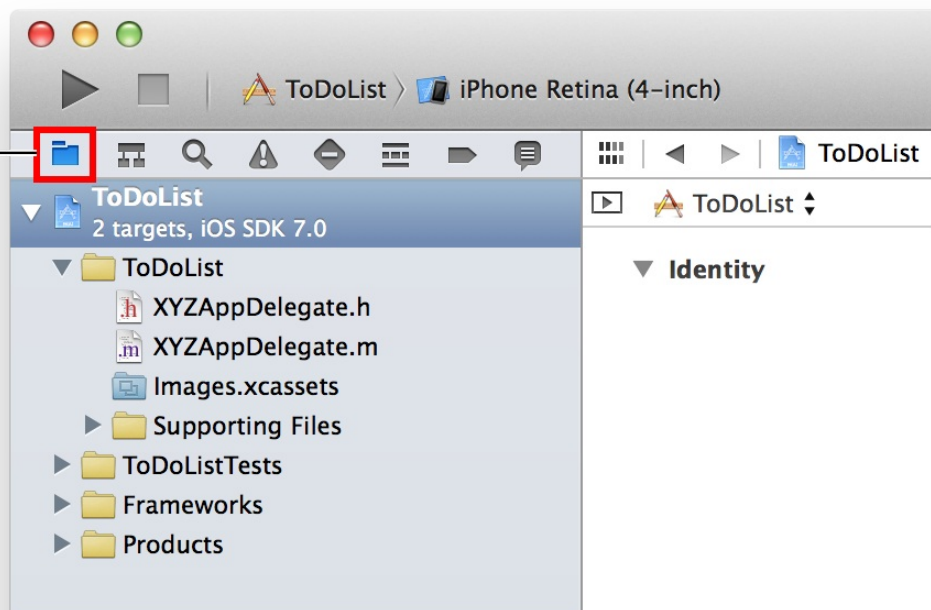
Empty Application 模板附带了少量现成的源代码，用于设置应用程序环境。大多数工作都由 UIApplicationMain 函数来完成，它在项目的主文件 main.m 源文件中会被自动调用。UIApplicationMain 函数会创建一个应用程序对象来设置应用程序基础结构，以配合 iOS 系统运作。包括创建一个运行循环，将输入事件传递给应用程序。

您不需要直接处理 main.m 源文件，但是了解一下它的工作方式也是颇有趣味的。

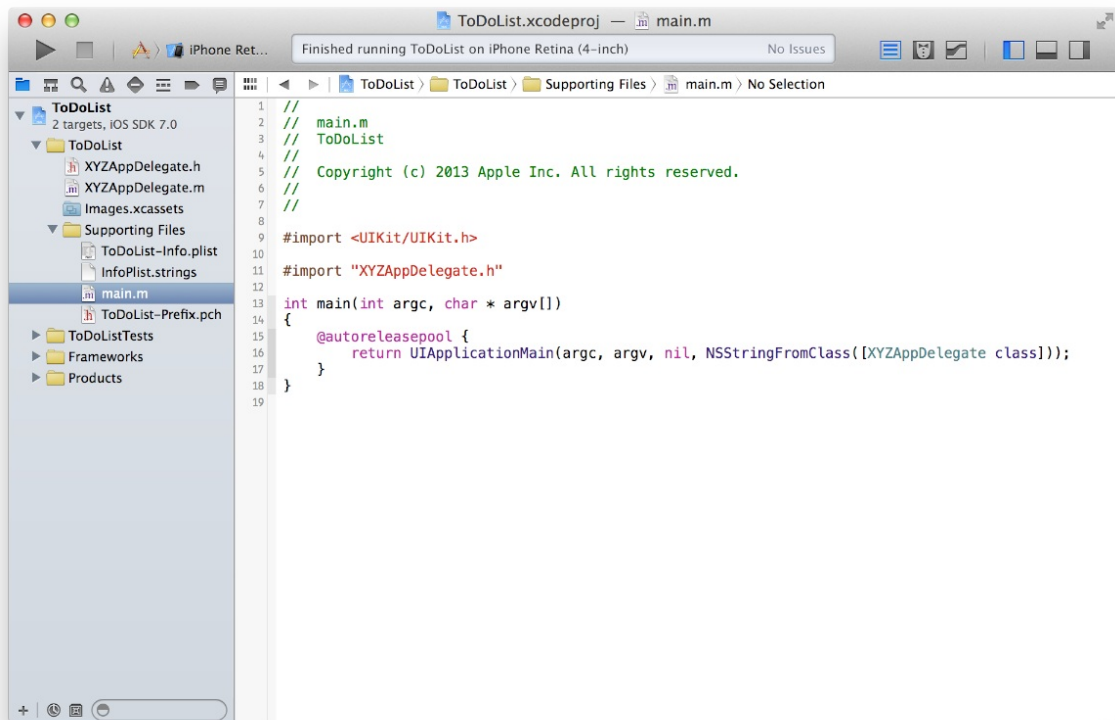
### 查看 main.m 源文件

1. 请确定项目导航器已在导航器区域中打开。项目导航器会显示项目中的所有文件。如果项目导航器未打开，请点按导航器选择栏最左边的按钮。

导航器选择栏



2. 点按项目导航器中“Supporting Files”文件夹旁边的显示三角形，打开文件夹。选择 main.m。
3. Xcode 会在窗口的主编辑器区域打开源文件，外观类似于：



如果连接该文件，它会在单独的窗口中打开。您可以根据需要进行选择：点按文件一次，将其在主项目窗口中打开；或是连接文件，将其在单独的窗口中打开。

main 中的 main.m 函数会调用自动释放池 (autorelease pool) 中的 UIApplicationMain 函数。

```
@autoreleasepool {
    return UIApplicationMain(argc, argv, nil, NSStringFromClass([XYZAppDelegate class]));
}
```

@autoreleasepool 语句支持应用程序的内存管理。自动引用计数 (Automatic Reference Counting, ARC) 利用编译器追踪对象的所有者，使内存管理变得简单；@autoreleasepool 是内存管理基础结构的一部分。

调用 UIApplicationMain 会创建应用程序的两个重要初始组件：

- UIApplication 类的实例，称为应用程序对象。应用程序对象可管理应用程序事件循环，并协调其他高级的应用程序行为。定义在 UIKit 框架中的这个类，不要求您编写任何额外的代码，就可以达成其任务。
- XYZAppDelegate 类的实例，称为应用程序委托。Xcode 创建此类，作为设置 Single View Application 模板的一部分。应用程序委托会创建一个呈现应用程序内容的窗口，并为响应应用程序内的状态转换提供位置。这个窗口是您编写自定义应用程序级代码的地方。与所有的类一样，XYZAppDelegate 类在应用程序的两个源代码文件中被定义：接口文件 XYZAppDelegate.h；实现文件 XYZAppDelegate.m。

以下是应用程序对象和应用程序委托互动的方式。应用程序启动时，应用程序对象会调用应用程序委托上已定义的方法，使自定义代码有机会执行其操作，这正是运行应用程序的有趣之处。为了深入理解应用程序委托的角色，请从接口文件开始查看其源代码。如果要查看应用程序委托的接口文件，请在项目导航器中选择 XYZAppDelegate.h。应用程序委托的界面包含了单一属性：window。有了这个属性，应用程序委托才会跟踪能呈现所有应用程序内容的窗口。

下一步，请查看应用程序委托的实现文件。请在项目导航器中选择 XYZAppDelegate.m。应用程序委托的实现包含了一些重要方法的“骨架”。这些预定义的方法可让应用程序对象与应用程序委托进行沟通。在一个重要的运行时事件（例如，应用程序启动、低内存警告和应用程序终止）中，应用程序对象会调用应用程序委托中相应的方法，使其有机会进行适当的响应。您无需执行任何特殊的操作，来确定这些方法是否会在正确的时间被调用，因为应用程序对象会帮您处理这部分的工作。

这些自动实现的方法都具有一个默认的行为。就算将骨架实现留空，或将它从 XYZAppDelegate.m 文件中删除，这些行为在方法被调用时，都会默认执行。您可以使用这些骨架来放置附加的自定义代码，以在方法被调用时执行。例如，XYZAppDelegate.m 文件中的第一个方法包含了几行代码，用于设置应用程序的窗口，并让应用程序首次运行时显示白色的背景颜色。在本教程中，您不会使用到任何自定义应用程序委托代码，因此可以移除这段代码。

## 配置应用程序委托的实现文件

1. 请在 XYZAppDelegate.m 中查找 application:didFinishLaunchingWithOptions: 方法。它是文件中的第一个方法。
2. 从该方法中删除前三行代码，然后它会显示为：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    return YES;
}
```

Xcode 会自动存储更改。它会时刻跟踪并存储您的所有操作。（您可以通过选取“Edit”>“Undo Typing”来撤销所作的更改。）

## 创建第一个视图

现在，您应该已经准备好创建您的第一个视图(UITableView)了。

视图是屏幕上的一块矩形区域，它在App中占有绝对重要的地位，因为iOS中几乎所有可视化控件都是UIView的子类。负责渲染区域的内容，并且响应该区域内发生的触摸事件。

### 创建一个标签(UILabel)

1. 在 XYZAppDelegate.m 中查找 application:didFinishLaunchingWithOptions: 方法。
2. 在该方法中添加如下代码：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];

    UILabel * label = [[UILabel alloc] initWithFrame:self.window.bounds];
    label.text = @"Hello, World!";
    label.textAlignment = NSTextAlignmentCenter;
    [self.window addSubview:label];

    return YES;
}
```

- UILabel是一种专门用来显示文字视图
- 这一行代码创建了一个名为label的UILabel

```
UILabel * label = [[UILabel alloc] initWithFrame:self.window.bounds];
```

- 这一行代码是告诉label显示的文字"Hello, World!"

```
label.text = @"Hello World";
```

- 设置label的文字居中对齐

```
label.textAlignment = NSTextAlignmentCenter;
```

- 将label加到窗口上

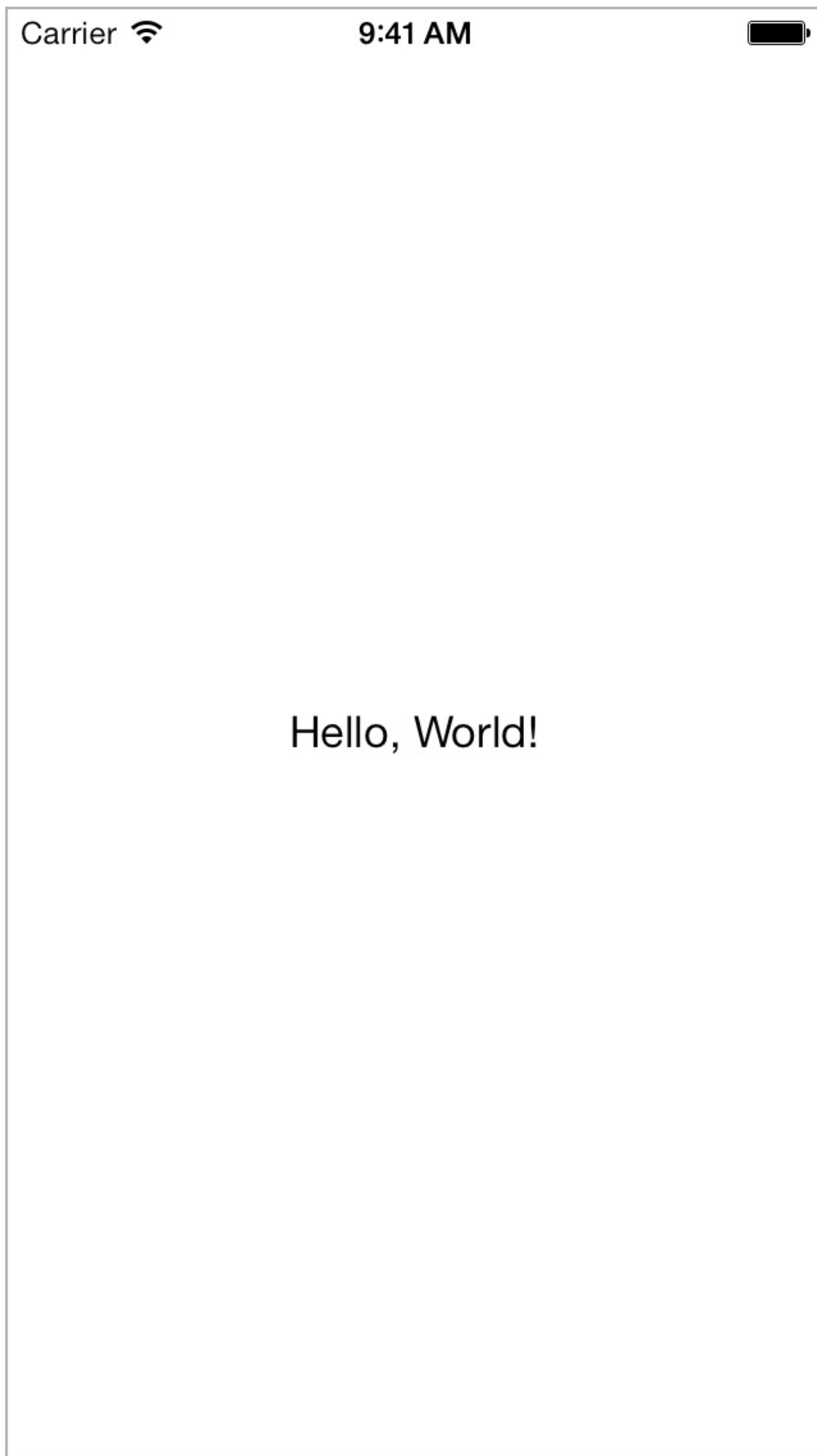
```
[self.window addSubview:label];
```

## 测试更改

---

最好在 iOS Simulator 中运行应用程序进行定期检查，看看是否一切都如预期般正常。此时，当应用程序启动时，应会载入您在主串联图中创建的场景。点按 Xcode 中的“Run”按钮。您看到的应该大致是这样的：





借此机会，您可以试验一下界面可添加的内容。

- 标签的文本 `label.text = @"test";`
- 标签的字体 `label.font = [UIFont systemFontOfSize:25];`
- 文本的颜色 `label.textColor = [UIColor redColor];`

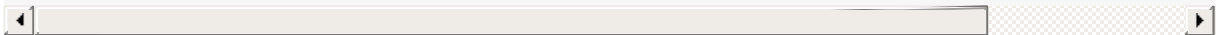
## 构建基本界面

现在您已经知道如何在场景中添加内容，接下来让我们开始构建场景的基本界面，将新项目添加到待办事项列表中。

将项目添加到待办事项列表需要一则信息：项目名称。您可以从文本栏中获得此信息。文本栏是界面元素，可让用户使用键盘输入单行文本。但首先，您需要移除前面所添加的标签。

1. 删除UILabel相关的代码
2. 创建一个UITextField 在 `application:didFinishLaunchingWithOptions:` 方法中添加如下代码

```
UITextField * textField = [[UITextField alloc] initWithFrame:CGRectMake(20, 100, CGRectGetWidth(self.window.bounds)
textField.borderStyle = UITextBorderStyleRoundedRect;
[self.window addSubview:textField];
```



3. 配置文本栏的的占位符文本。添加一行代码：

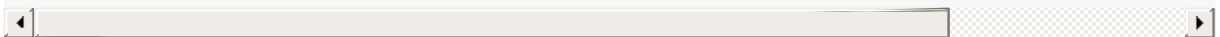
```
textField.placeholder = @"New to-do item";
```

此时 `application:didFinishLaunchingWithOptions:` 方法应该如下所示：

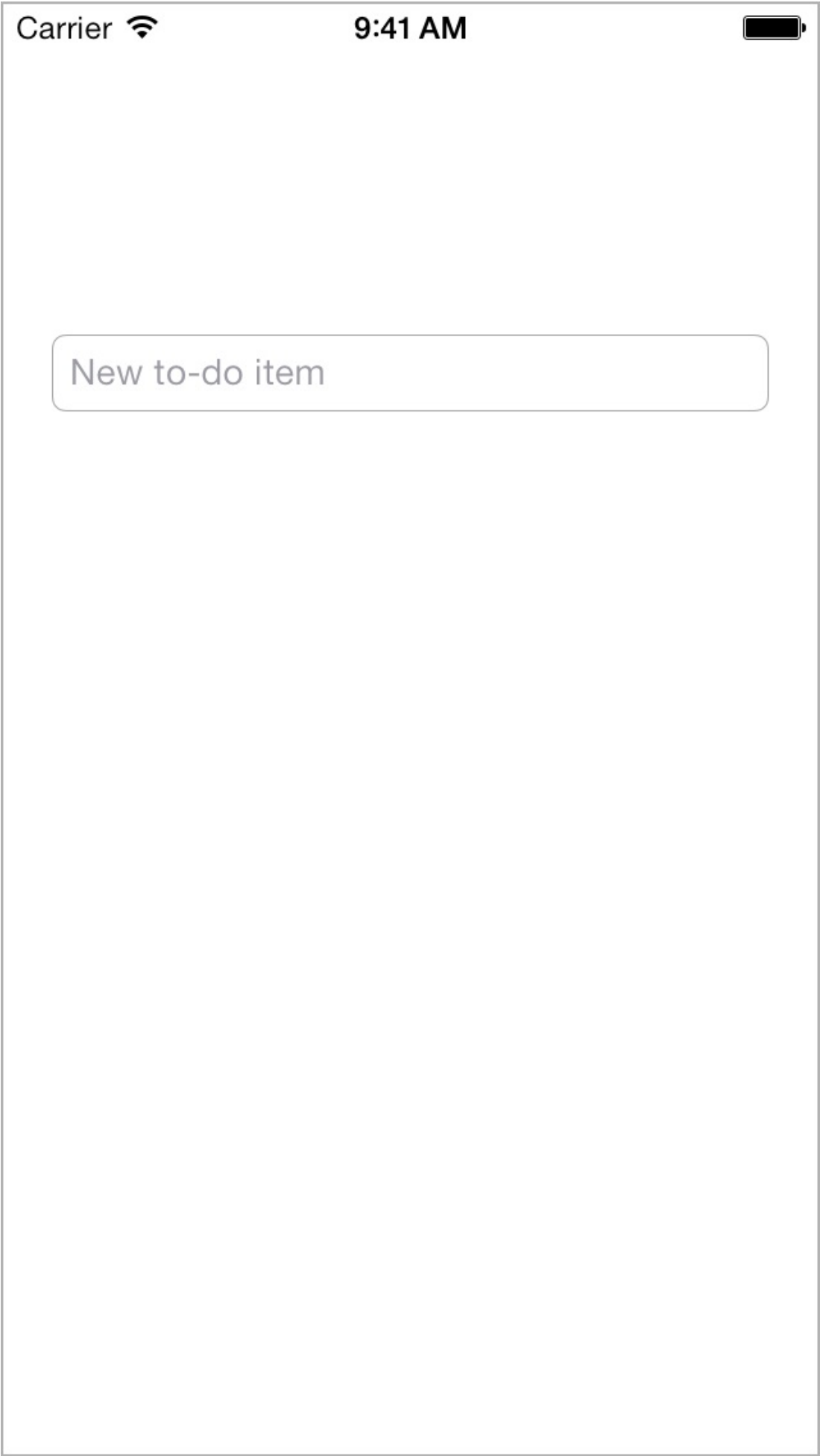
```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];

    UITextField * textField = [[UITextField alloc] initWithFrame:CGRectMake(20, 100, CGRectGetWidth(self.window.
    textField.placeholder = @"New to-do item";
    textField.borderStyle = UITextBorderStyleRoundedRect;
    [self.window addSubview:textField];

    return YES;
}
```



4. 在 iOS Simulator 中运行应用程序，确定一下所创建的场景是否令您满意。您应该能够在文本栏中点按，而且可以使用键盘输入字符串。

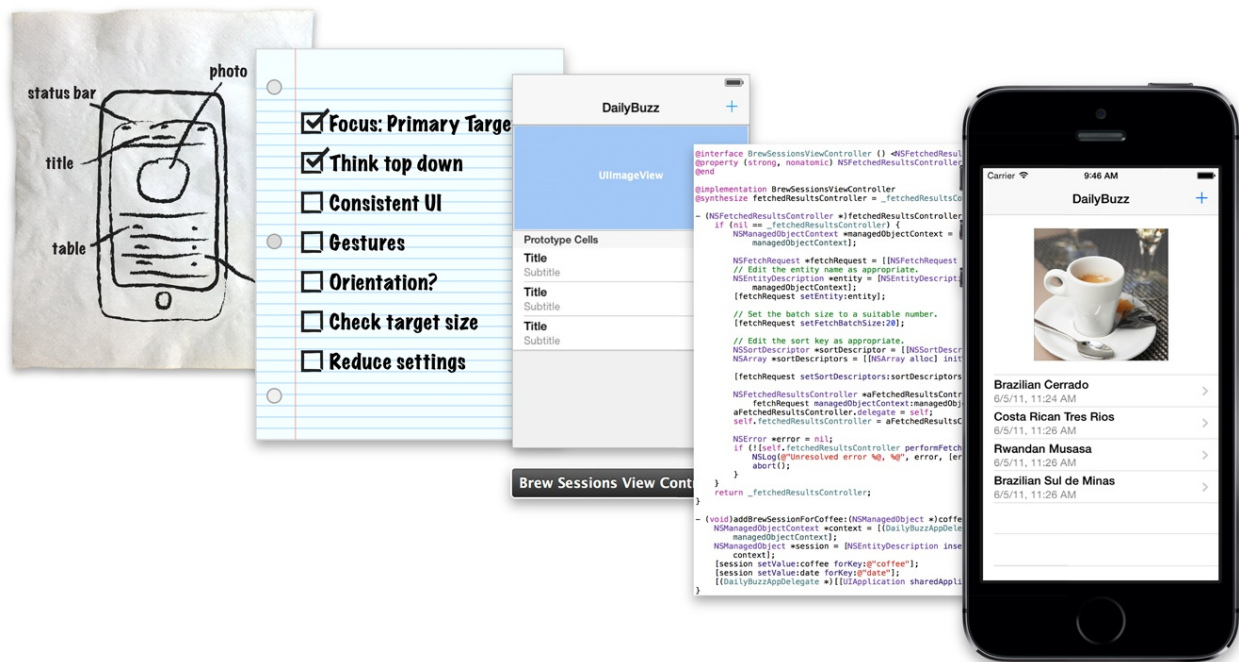


## 小结

---

现在，您学会了如何使用代码来创建基本的界面。在接下来的教程中，会了解到如何给界面添加交互，以及如何编写代码来创建自定义行为。教程中的各个章节介绍了一些概念，可让您在处理自己的应用程序时学以致用。

开发应用程序看似十分艰巨，其实整个过程可以浓缩为几个易于理解的步骤。下面的步骤可以帮助您立即开始并正确引导您开发第一个应用程序。



## 定义概念

概念是优秀应用程序的源头。

而形成概念的最佳方式便是考虑应用程序所要解决的问题。好的应用程序解决的是单个明确的问题。例如，“设置”应用程序能让用户调整设备上的所有设置。每个任务的相关设置都会在单独界面里完成。

形成概念时，要考虑这些关键的问题：

您的用户是谁？不同应用程序的内容和用户体验大不相同，这取决于您想要编写的是什么应用程序，它可能是儿童游戏，也可能是待办事项列表应用程序，又或者是测试自己学习成果的应用程序。

应用程序的用途是什么？赋予应用程序一个明确的用途十分重要。了解激发用户使用应用程序的动因是界定用途的一个出发点。

应用程序尝试解决什么问题？应用程序应该完美解决单个问题，而不是尝试解决多个截然不同的问题。如果发现应用程序尝试解决不相关的问题，那么最好考虑编写多个应用程序。

应用程序要呈现什么内容？考虑应用程序将向用户呈现的内容类型，以及用户与应用程序的互动方式，然后设计与之相称的用户界面。

刚开始开发应用程序时，不必定义完美或完整的应用程序概念。但有了概念之后，您便会明确自己的开发目标和实现方法。

## 设计用户界面

形成了应用程序的概念后，接下来是设计一个良好的用户界面，这是成功的关键一步。用户需要以尽可能简单的方式与应用程序界面进行交互。为此，您需要从用户的角度来设计界面，使其高效、简洁且直观。

构建用户界面最大的挑战可能在于将概念转化为设计并实现该设计。您可以使用串联图来简化这个过程。串联图能让您使用图形环境来一步设计并实现界面。构建界面时，您可以完全看到构建的内容，马上获得相关界面能否正常工作的反馈，并立

即以可视化方式对界面进行更改。

在串联图中构建界面时，您是以视图进行工作。视图向用户显示内容。在“教程：基础”中，您通过使用串联图场景中的单视图，定义了 ToDoList 应用程序的用户界面。随着应用程序开发的复杂化，您将会创建包含更多场景和视图的界面。

在教程：串联图中，您将使用多种不同的视图来完成构建 ToDoList 应用程序的用户界面，从而显示不同类型的内容。在设计用户界面中，您会了解有关使用视图和串联图来设计和创建用户界面的更多知识。

## 定义行为

定义了用户可以在应用程序中执行的操作后，可以编写代码来实现行为。

为 iOS 应用程序编写代码时，大多数时间都要用到 Objective-C 程序设计语言。在第三个模块中，您会了解有关 Objective-C 的更多知识，但是现在基本熟悉一下 Objective-C 语言的词汇会有裨益。

Objective-C 源于 C 程序设计语言，它提供了面向对象的功能以及动态运行时。它包含您熟悉的所有元素，例如基本类型（int、float 等）、结构、函数、指针以及流程控制结构（while、if...else 以及 for 语句）。您还可以访问标准 C 库例程，例如在 stdlib.h 和 stdio.h 中声明的那些例程。

## 对象是应用程序的基石

构建 iOS 应用程序时，大多数时候接触的是对象。

对象会将具有相关行为的数据包装起来。您可以将应用程序设想为一个大型生态系统，其中互连的对象相互通信来解决特定的问题，例如显示可视化的界面，响应用户的输入或者储存信息。构建应用程序要用到多种不同类型的对象，从界面元素（例如按钮和标签）到数据对象（例如字符串和数组）。

## 类是对象的蓝图

类描述了特定类型的对象所共有的行为和属性。

按照同一个蓝图进行施工的建筑物，它们的结构是相同的。与之类似，类的每个实例的行为和属性与该类的所有其他实例的行为和属性也是相同的。您既可以编写自己的类，也可以使用已经定义好的框架类。

可以通过创建特定类的实例来新建对象。途径是为对象分配并初始化合适的默认值。分配对象时，您为该对象预留了足够的内存并将所有的实例变量设定为 0。初始化将一个对象的初始状态（即它的实例变量和属性）设定为合理的值，然后返回对象。初始化的目的在于返回有用的对象。您需要分配并初始化对象，这样才能使用它。

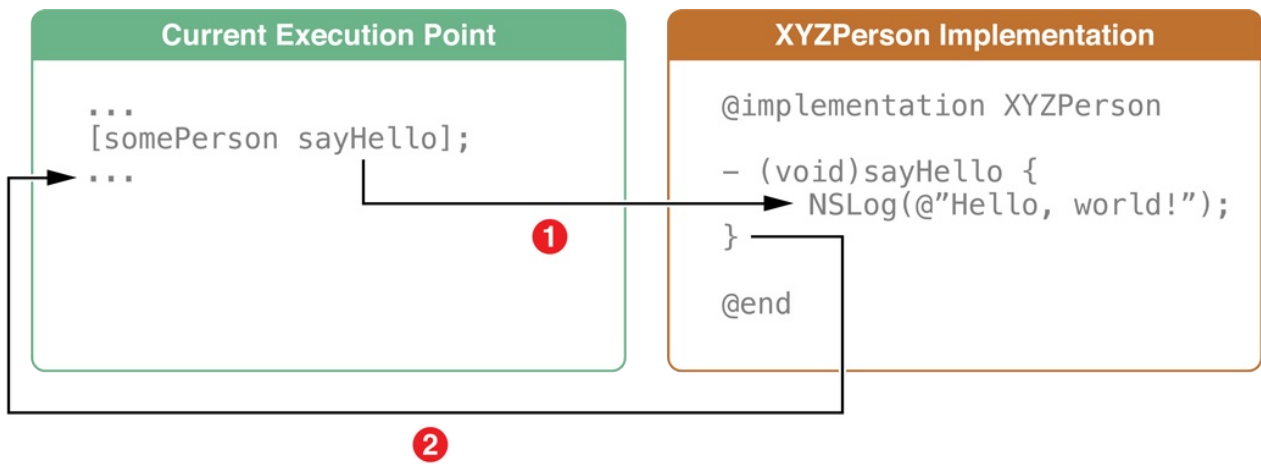
Objective-C 程序设计语言中的一个基本概念就是类继承，即类会继承父类的行为。一个类继承另一个类时，继承类（或子类）会继承由父类定义的所有行为和属性。您可以为子类定义属于它自己的其他行为和属性或者覆盖父类的行为。这样，您就可以扩展类的行为，而无需复制其现有的行为。

## 对象通过消息通信

对象在运行时通过互相发送消息来交互。在 Objective-C 术语中，一个对象通过调用另一个对象的方法来向该对象发送消息。

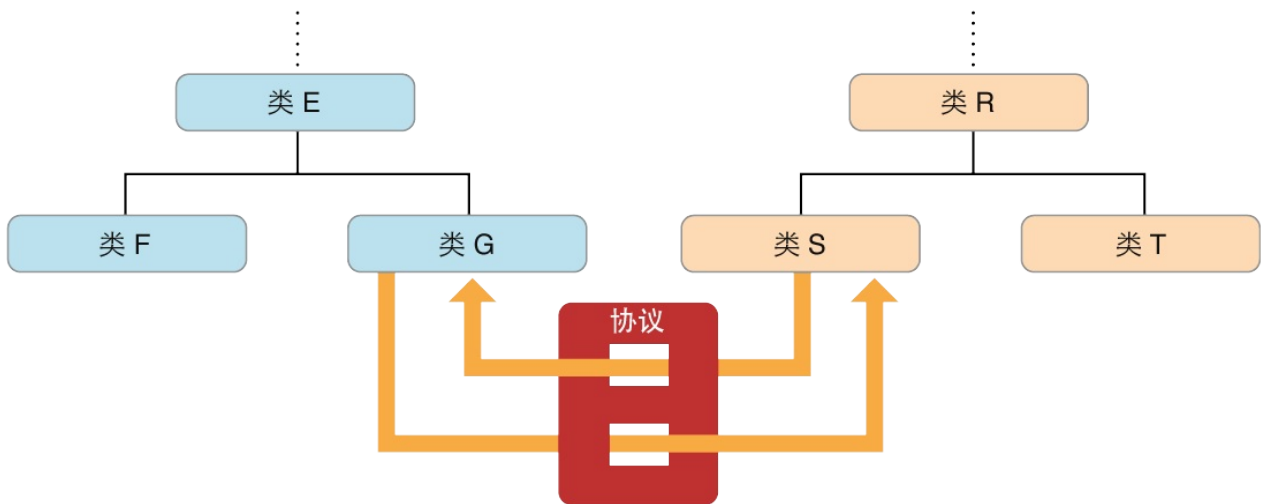
在 Objective-C 中，虽然可使用多种方法在对象之间发送消息，但是目前最常用的方法是使用方括号的基本语法。如果您有一个 Person 类的对象 somePerson，那么可以按照如下所述来向它发送消息 sayHello：

[somePerson sayHello]; 左侧的引用 somePerson 是消息的接收者。右侧的消息 sayHello 是调用其方法的名称。换句话说，执行以上代码行时，会向 somePerson 发送 sayHello 消息。



## 协议定义消息发送契约

协议定义对象在给定条件下的一组预期行为。它采用可编程的接口形式（任何类都可以选择来实现）。通过使用协议，两个因为继承而略有关联的类可以彼此通信来完成某个目标，例如解析 XML 代码或拷贝对象。



如果类能够提供为其他类使用的行为，那么该类可以声明可编程的接口，以匿名方式来供应该行为。任何其他类都可以选择采用该协议，并实现该协议的一个或多个方法，从而利用该行为。

## 整合数据

实现应用程序的行为后，您需要创建数据模型来支持应用程序的界面。应用程序的数据模型定义了维护应用程序中数据的方式。数据模型的范围既包括对象的基本词典，也包括复杂的数据库。

应用程序的数据模型应该反映该应用程序的内容和用途。虽然用户不会直接和数据交互，但界面和数据之间应该有明显的相关性。

若要为应用程序打下良好的基石，一个好的数据模型必不可少。有了数据模型，构建可扩展的应用程序、改进功能以及修改特性会变得易如反掌。在整合数据中，您会了解有关定义自己的数据模型的更多知识。

## 使用正确的资源

设计模式是解决应用程序中常见问题的最佳实践。它能帮助您定义数据模型的结构以及它与应用程序其他部分的交互方式。理解并使用正确的设计模式，便能轻松地创建简单且实用的应用程序。在使用设计模式中，您会了解有关设计模式的更多知识。

请记住，刚开始实现模型时，不必一切从零开始。您可以以一系列提供了现有功能的框架为基础进行构建。例如，Foundation 框架包括了表示基本数据类型的类（例如字符串和数字），以及用于储存其他对象的集类。建议您尽可能地使用现有框架类，或者对其进行子类化来为应用程序添加自己的功能，而不是尝试重新实现它们的功能。这样，您就可以创建一个高效、实用且精致的软件。在“处理 Foundation”中，您会了解有关 Foundation 框架功能的更多知识。

通常，您会编写自己的自定义类来作为数据模型的一部分。通过编写自定义类，您可以控制应用程序内部结构的整理方式。在写自定义类中，您会了解有关创建自定义类的更多知识。

## 整合真实的数据

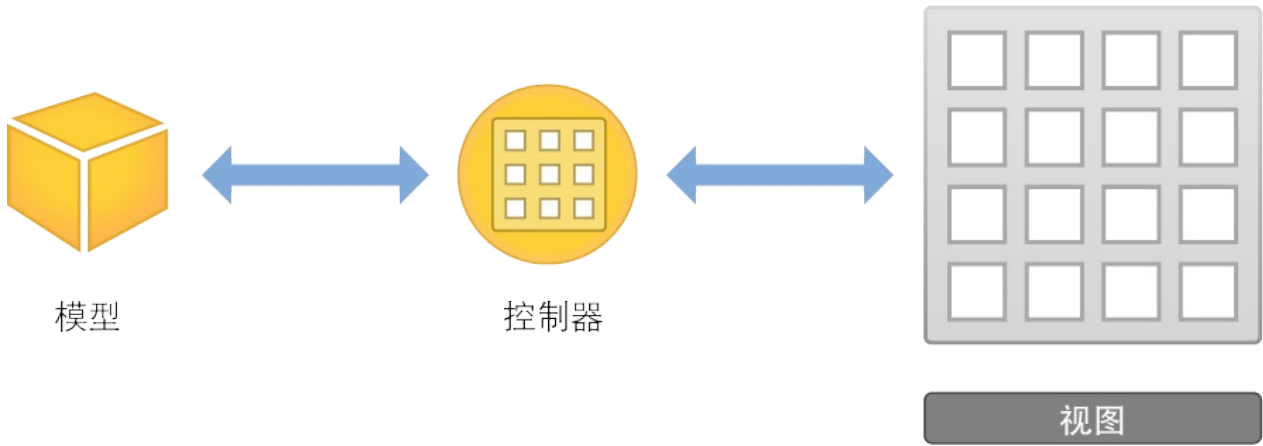
---

首次测试数据模型时，您不妨使用静态数据或假数据。这样，在正确组装和连接该模型前，您都无需为提供真实数据而担心了。定义的模型能够正常工作后，再将真实的数据引入应用程序中。

本指南的剩余部分会更详细地介绍这些步骤。随着应用程序开发过程的深入，您将会学习必要的概念性知识，然后在教程中进行实践。



视图是构建用户界面的基石。理解如何使用视图，以优美且实用的方式清晰地呈现您的内容十分重要。想要开发一个成功的应用程序，至关重要的一点便是创建一个优秀的用户界面，有效地展示应用程序的内容。在本章中，您将会学习如何在串联图中创建和管理视图来定义您的界面。



## 视图层次

视图不仅显示在屏幕上并对用户的输入作出响应，它们还可以充当其他视图的容器。因此，应用程序中的视图可按层次结构进行排列，我们将其称为视图层次。视图层次定义了视图相对于其他视图的布局。在该层次内，包含在某个视图中的视图实例称为分视图，而包含视图的父视图则被称为该视图实例的超视图。虽然一个视图实例可以有多个分视图，但它只能有一个超视图。


位于视图层次顶部的是窗口对象。窗口由 `UIWindow` 类的实例表示，它可以作为基本的容器，您可以将要在屏幕上显示的视图对象添加到其中。窗口本身不会显示任何内容。如果要显示内容，请将内容视图（及其分视图的层次）添加到窗口。

为了让用户看见内容视图及其分视图，必须将内容视图插入到窗口的视图层次中。使用串联图时，系统会自动将内容视图插入到窗口的视图层次中。应用程序对象会载入串联图，创建相关视图控制器类的实例，解压缩每个视图控制器的内容视图层次，然后将初始视图控制器的内容视图添加到窗口中。在下一章中，您会学到有关管理视图控制器的更多内容。目前，您只需专注于在串联图的单个视图控制器中创建层次即可。

## 使用视图构建界面

设计应用程序时，了解将何种视图用于何种目的十分重要。例如，用来收集用户的输入文本的视图（如文本栏）与可能用来显示静态文本的视图（如标签）是不同的。使用 `UIKit` 视图进行绘图的应用程序很容易创建，因为您可以快速组装一个基本界面。`UIKit` 视图对象是 `UIView` 类或其中一个子类的实例。`UIKit` 框架提供了许多类型的视图，来帮助呈现和组织数据。

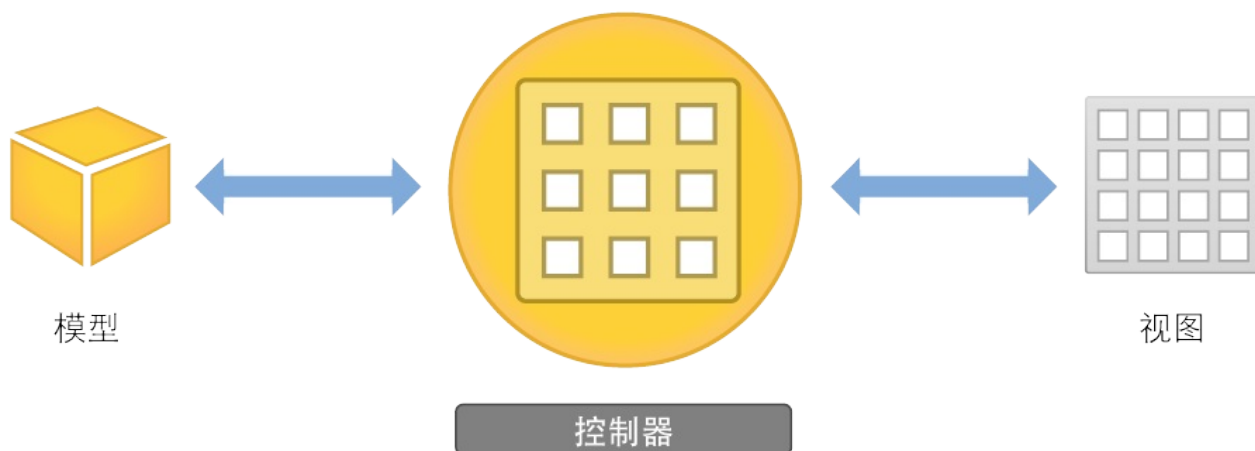
每个视图都有其特定的功能，不过 `UIKit` 大体可分为以下七种常见类型：

类型	用途	示例
内容 	显示特定类型的内容，例如图像或文本。	图像视图( <code>UIImageView</code> )，标签( <code>UILabel</code> )
集		

	显示视图集或视图组。	集视图(UICollectionView), 表格视图(UITableView)
控制 	执行操作或显示信息。	按钮(UITableView), 滑块(UISlider), 开关(UISwitch)
栏 	导航或执行操作。	工具栏(UIToolbar), 导航栏, 标签栏
输入 	接收用户输入的文本。	搜索栏(UISearchBar), 文本视图(UITextView)
容器 	充当其他视图的容器。	视图(UIView), 滚动视图(UIScrollView)
模态	中断应用程序的正常流程, 允许用户执行某种操作。	操作表单(UIActionSheet)、提醒视图(UIAlertView)

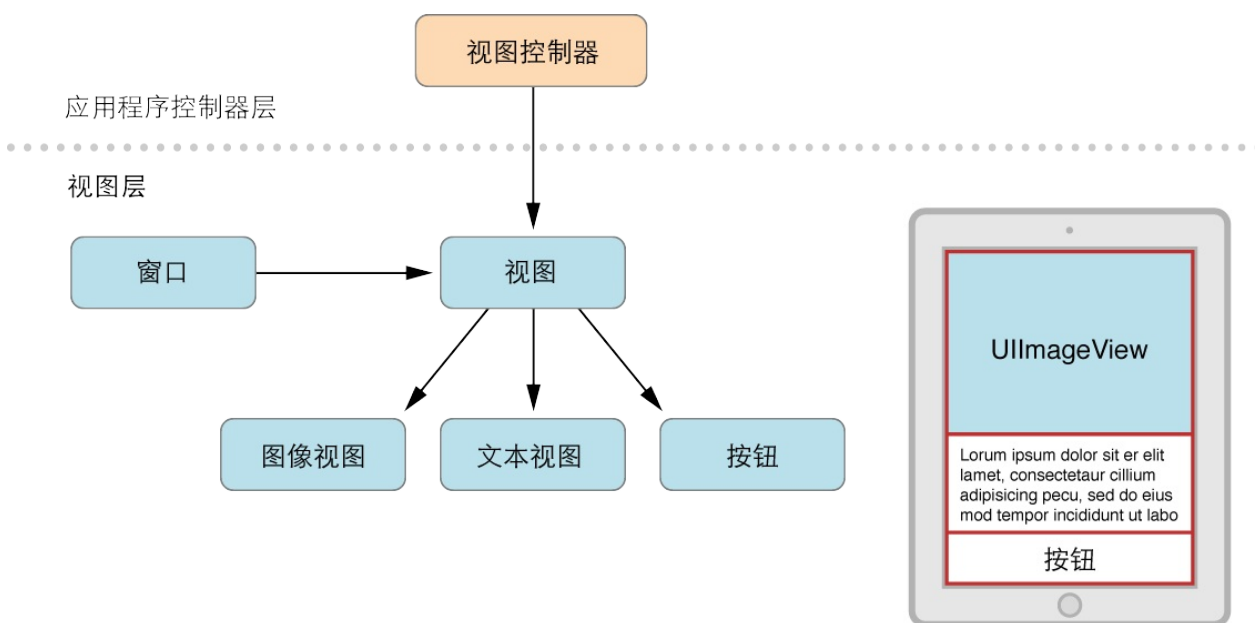
您可以使用 UIKit 框架提供的标准视图来显示各种类型的内容, 但也可以自定义视图, 方法是子类化 UIView (或其后代)。自定义视图是 UIView 的子类, 您可以自行在其中处理所有绘图和事件处理任务。在这些教程中, 您将不会使用自定义视图, 但是在[Defining a Custom View \(定义自定义视图\)](#)中, 可以了解到有关实现自定义视图的更多知识。

对用户界面进行布局后，接下来就需要让用户与界面进行交互。这时就要使用控制器了。控制器能响应用户操作并使用内容填充视图，从而支持您的视图。控制器对象是一个管道，通过它，视图能了解数据模型的修改，反之亦然。应用程序的控制器会通知视图有关模型数据中的修改，之后控制器会将用户发起的修改（例如，在文本栏中输入的文本）传达到模型对象。不论模型对象是响应用户操作，还是定义浏览，控制器都会实现应用程序的行为。



## 视图控制器

构建了基本的视图层次后，下一步就需要控制可视元素并响应用户输入。在 iOS 应用程序中，您可以使用视图控制器 (UIViewController) 来管理内容视图及其分视图层次。



视图控制器并不是视图层次的一部分，也不是界面中的元素。相反，它管理着层次中的视图对象，并为它们提供行为。在串联图中构建的每个内容视图层次，都需要一个对应的视图控制器来管理界面元素，并执行任务来响应用户的交互操作。通常，这意味着您需要为每个内容视图层次编写一个自定义 UIViewController 子类。如果应用程序有多个内容视图，那么就需要为每个内容视图使用不同的自定义视图控制器类。

视图控制器扮演着多种角色。它们负责协调应用程序的数据模型与显示该数据的视图之间的信息传输，管理应用程序的内容视图的生命周期，并处理设备旋转时方向的更改。但其最主要的作用可能是响应用户输入。

您还可以使用视图控制器来转换各种类型的内容。由于 iOS 应用程序显示内容的空间很有限，因此视图控制器提供了所需要的基础结构，可让您移除一个视图控制器的视图，替换为另一个视图控制器中的视图。

通过让视图控制器文件与串联图中的视图进行通信，可以定义应用程序中的交互方式。方法是通过 Action 与 Outlet 来定义串

联图与源代码文件之间的连接。

## 操作 (Action)

操作 是一段代码，它与应用程序中可能会发生的某类事件相链接。该事件发生后，代码就会执行。您可以定义操作来完成任何事情：从操控数据到更新用户界面。操作可驱动应用程序的流程来响应用户事件或者系统事件。

可采用如下方法来定义：使用 `IBAction` 返回类型和 `sender` 参数来创建并实现方法。

```
- (IBAction)restoreDefaults:(id)sender;
```

`sender` 参数指向负责触发操作的对象。`IBAction` 返回类型是个特殊的关键词。它与 `void` 关键词类似，但它表示该方法是一种操作，您可以在 Interface Builder 的串联图中连接到这种操作（这就是这个关键词有 `IB` 前缀的原因）。在“教程：串联图”中，您会了解如何将 `IBAction` 操作链接到串联图中元素的更多知识。

## Outlet

Outlet 可让您从源代码文件引用界面中的对象（添加到串联图的对象）。您可以按住 `Control` 键，并将串联图中的特定对象拖移至视图控制器文件来创建 Outlet。这就为视图控制器文件中的对象创建了属性，通过该属性，您可以在运行时通过代码来访问并操控该对象。例如，在第二个教程中，您将为 `ToDoList` 应用程序中的文本栏创建 Outlet，这样就可以用代码的形式访问文本栏的内容。

Outlet 被定义为 `IBOutlet` 属性。

```
@property (weak, nonatomic) IBOutlet UITextField *textField;
```

`IBOutlet` 关键词告诉 Xcode，您可以从 Interface Builder 连接到该属性。在“教程：串联图”中，您会了解有关如何从串联图将 Outlet 连接到源代码的更多知识。

## 控制 (Control)

控制是用户界面对象（例如按钮、滑块或者开关），用户可以操控它们来与内容进行交互、提供输入、在应用程序内导航，以及执行所定义的其他操作。代码可通过控制来接收用户界面的消息。

用户与控制进行交互，会创建控制事件。控制事件表示用户可在控制上使用的各种手势，例如将手指抬离控制、手指拖移到控制上，以及在文本栏中按下。

常见的事件类型有三种：

- 触碰和拖移事件。用户通过触碰或者拖移与控制交互时，发生的就是触碰和拖移事件。触碰事件分几个阶段。例如，当用户初次用手指触碰按钮，就会触发 `Touch Down Inside` 事件；如果用户手指拖离按钮，则会触发相应的拖移事件。当用户的手指抬离按钮但仍停留按钮边缘的范围内，就会发送 `Touch Up Inside`。如果用户在抬起手指前，手指已经拖离了按钮（实际上是取消了触碰），就会触发 `Touch Up Outside` 事件。
- 编辑事件。用户编辑文本栏，发生的是编辑事件。
- 值更改事件。用户对控制进行操控，从而导致控制产生一系列不同的值，发生的是值更改事件。

定义交互时，您应该了解与应用程序中每个控制相关联的操作，然后明确地向用户展示应用程序中控制的作用。

## 导航控制器

如果应用程序有多个内容视图层次，就需要能够在它们之间进行切换。为此，可以使用专门的视图控制器：导航控制器 (UINavigationController)。导航控制器管理在一系列视图控制器中向后和向前切换的操作，例如用户在 iOS 版“邮件”应用程序的电子邮件帐户、收件箱邮件和单封电子邮件之间导航。

我们将由特定导航控制器所管理的一组视图控制器称为其导航栈。导航栈是一组后进先出的自定视图控制器对象。添加到堆栈的第一个项目将变成根视图控制器，永不会从堆栈中弹出。而其他视图控制器可被压入或弹出导航栈。

虽然导航控制器的最主要作用是管理内容视图控制器的显示方式，但它还负责显示自己的自定视图。具体来说，它会显示导航栏（位于屏幕顶部的视图，提供有关用户在导航层次中位置的上下文）。导航栏包含一个返回按钮和其他可以自定的按钮。添加到导航栈的每个视图控制器都会显示这个导航栏。您需要配置导航栏。

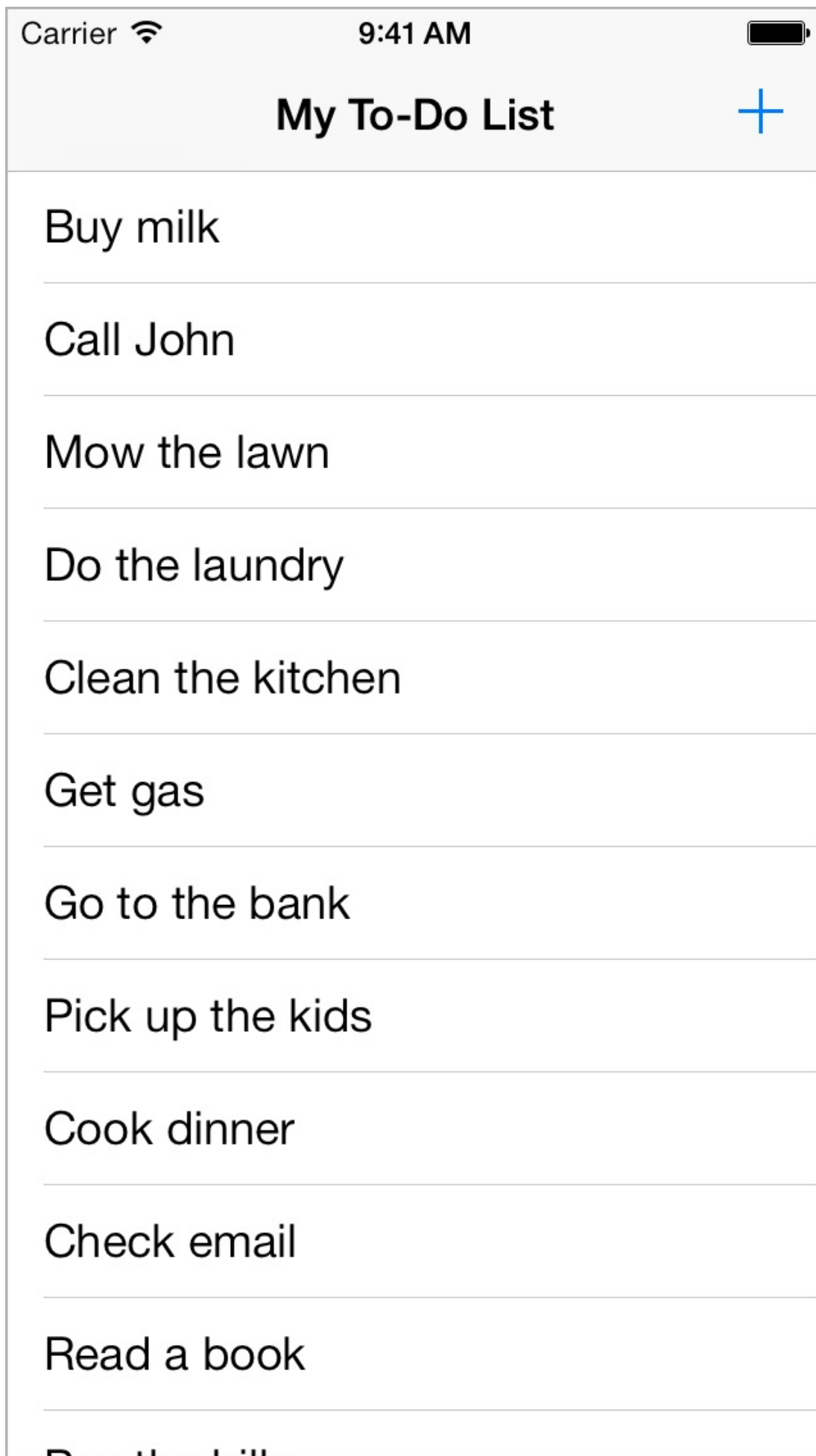
一般而言，您不必执行任何操作来将视图控制器弹出导航栈；导航控制器提供的返回按钮会实现该操作。但您需要手动将视图控制器压入堆栈中。

本教程以您在首个教程（“教程：基础”）中创建的项目为基础。您将会实践已学到的知识，包括视图、视图控制器、操作和导航方法。按照界面优先的设计流程，您还将为 `ToDoList` 应用程序创建某些关键的用户界面，并将行为添加到所创建的场景中。

本教程将向您讲述如何：

- 采用 `UIViewAutoresizing` 为用户界面增添灵活性
- 使用串联图来定义应用程序内容和流程
- 管理多个视图控制器
- 给用户界面中的元素添加操作

完成本教程中的所有步骤后，您的应用程序外观大致是这样的：

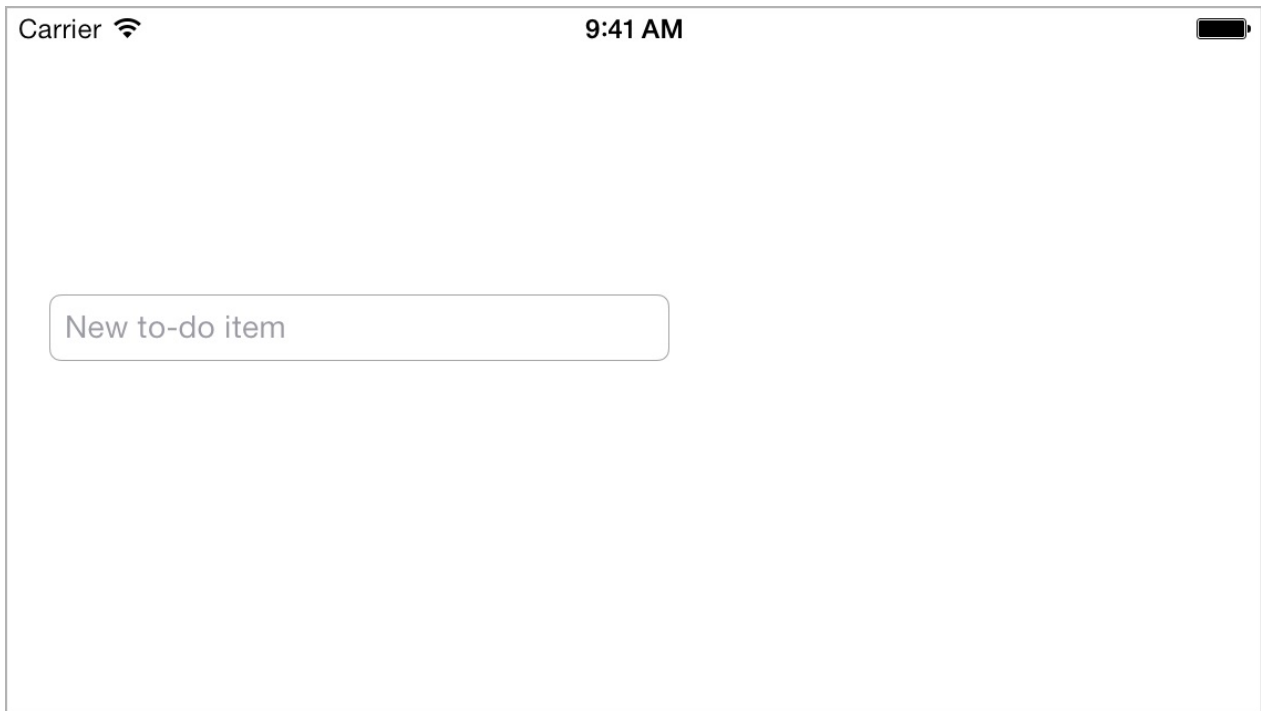


## UIViewAutoresizing

“add-to-do-item”场景的工作模式为竖排，与创建时一致。要是用户旋转了设备会如何？请试着在 Simulator 中运行应用程序来模拟这种情况。

### 在 iOS Simulator 中进行旋转

1. 请在 iOS Simulator 中启动应用程序。
2. 选取“Hardware”>“Rotate Left”（或按下 Command-左箭头键）。



如您所见，文本栏看起来不大对劲。它只占了屏幕约一半的位置。文本栏本应延伸至整个屏幕，正如竖排模式中显示的那样。关于场景中要如何放置元素，您可以通过 `UIViewAutoresizing` 将您的意图描述出来，然后由该布局引擎确定如何以最优方案实现该意图。使用约束规则描述您的意图，它说明了应当如何放置一个元素以与另一个元素相关联、元素应有的大小，或者在适用的空间减小时，两个元素中的哪一个应当先缩小。对于“add-to-do-item”场景，则需要使用两类约束：一种用于放置文本栏，另一种用于设定其大小。

### 设置 UIViewAutoresizing

打开文件 `XYZAddToDoItemViewController.m`，修改 `viewDidLoad` 方法：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    UITextField * textField = [[UITextField alloc] initWithFrame:CGRectMake(20,
                                                                              100,
                                                                              CGRectGetGetWidth(self.view.bounds) - 2 * 20,
                                                                              30)];

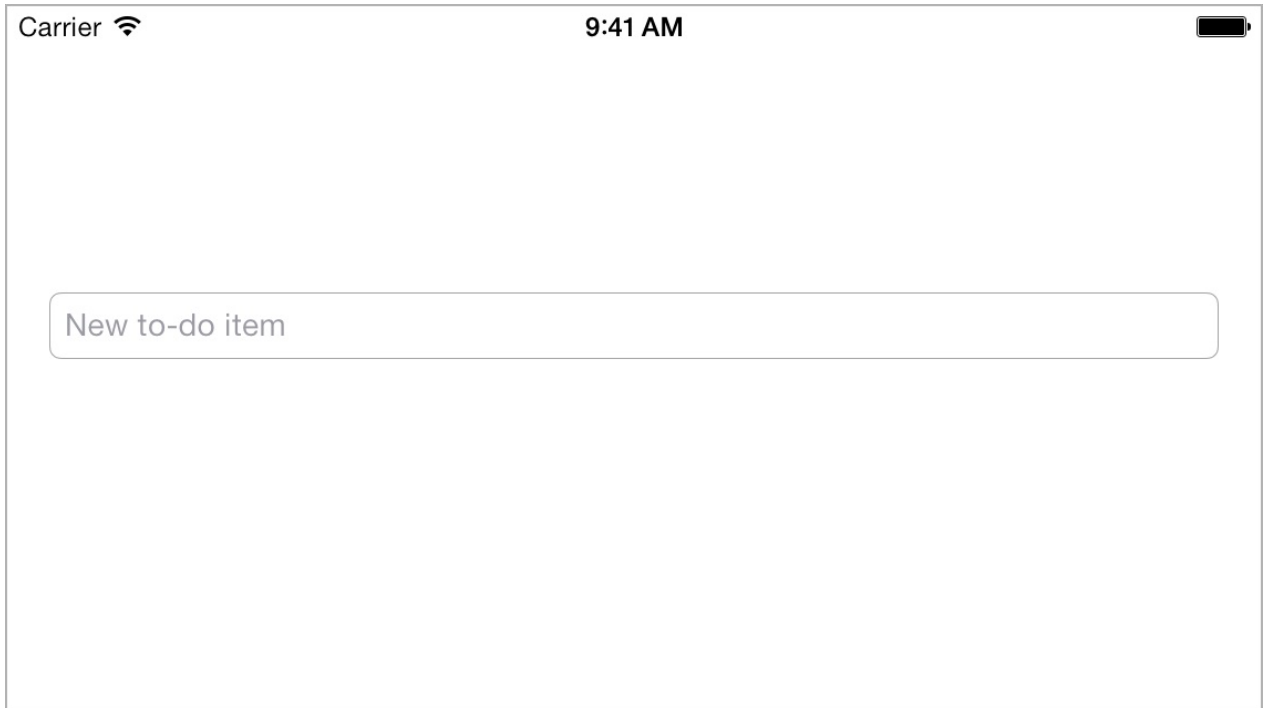
    textField.placeholder = @"New to-do item";
    textField.borderStyle = UITextBorderStyleRoundedRect;
    textField.autoresizingMask = UIViewAutoresizingFlexibleWidth; // 1
    [self.view addSubview:textField];
}
```



```
}  
}
```

1. `autoresizingMask`属性标记了当`textField`的`superview`的大小发生变化时, `textField`的大小该如何变化。当一个`UIView`的`bounds`发生变化时, 会自动修改所有`subview`的`bounds`。我们可以设置`autoresizingMask`属性来标示需要何种类型的变化, 可以通过使用C位运算或OR运算符组合给这个属性赋值, 结合`UIViewAutoresizing`这些常量, 可以指定该视图相对于`superview`的尺寸应增长或收缩。

检查点: 运行您的应用程序。如果您旋转设备, 文本栏将根据设备的方向伸展或收缩到适当大小。

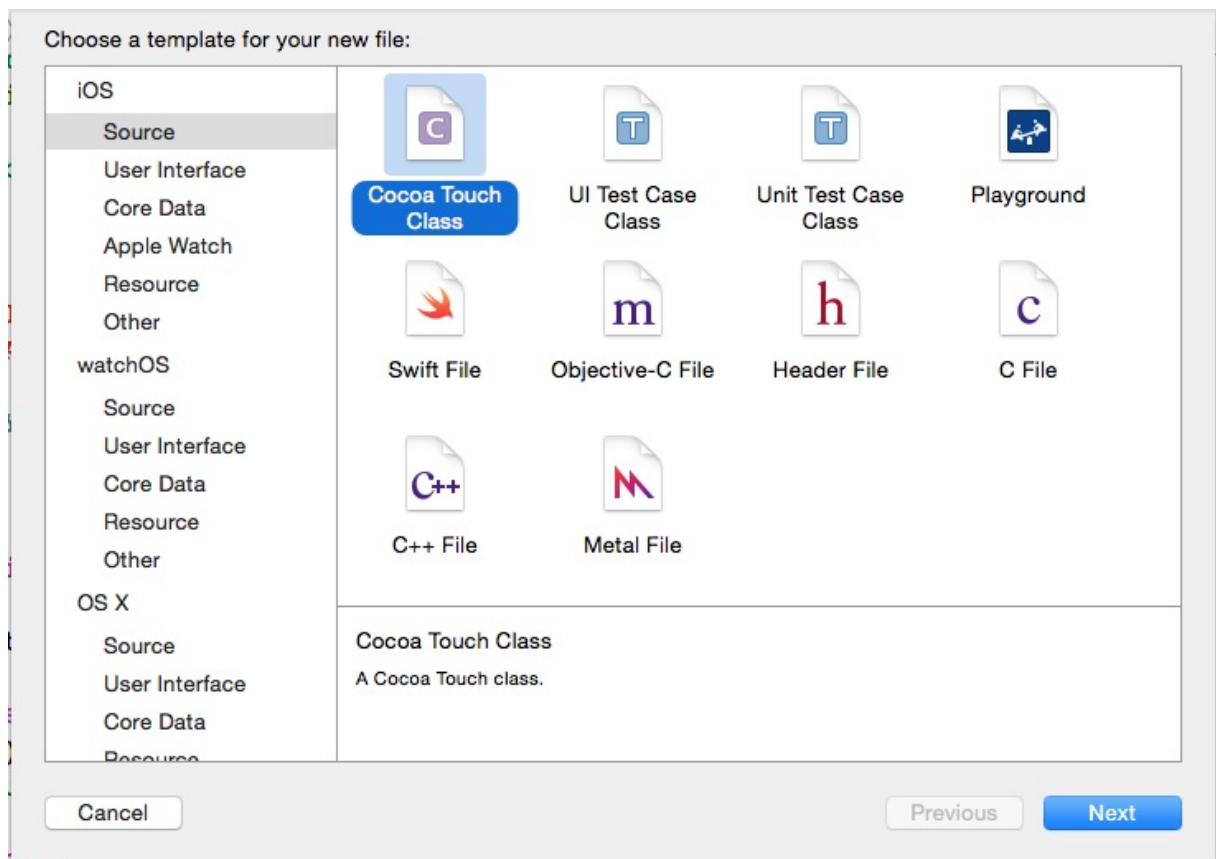


## 创建第二个ViewController

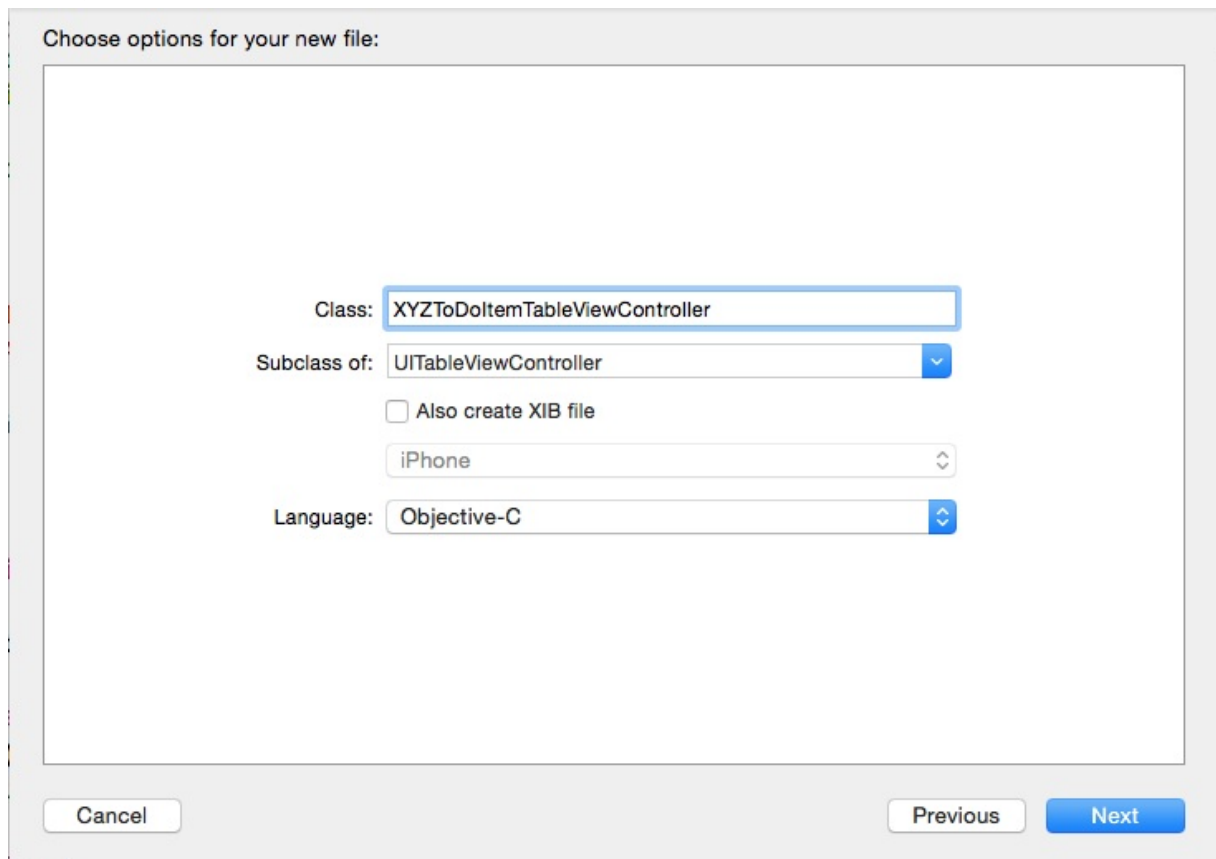
目前, 您已经有一个可让您将项目添加到待办事项列表的页面。现在, 是时候创建一个能显示整个待办事项列表的界面了。真是幸运, iOS 自带一个名为表格视图的内建类, 它功能强大, 专门设计用于显示项目的滚动列表。

## 添加一个UITableViewController

1. 新建文件



2. 选择Cocoa Touch Class，然后点击Next
3. Class输入XYZToDoltemTableViewController，Subclass of输入UITableViewController，点击Next



4. 修改AppDelegate.m，给self.window.rootViewController赋值一个XYZToDoltemTableViewController类型的对象

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
```

```
// Override point for customization after application launch.
self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen] bounds];
self.window.backgroundColor = [UIColor whiteColor];
[self.window makeKeyAndVisible];

UIViewController * toItemViewController = [[XYZToDoItemTableViewController alloc] init];

UIViewController * rootViewController = [[UINavigationController alloc] initWithRootViewController:toItemViewCont
self.window.rootViewController = rootViewController;

return YES;
}
```

## 在表格视图中显示静态内容

由于您尚未学习如何储存数据，因此要创建和储存待办事项，并将它们显示在表格视图中还为时过早。但是要展示用户界面原型，并不需要用到真实数据。Xcode 允许您通过 Interface Builder 在表格视图中创建静态内容。此功能便于您查看用户界面的表现，并且对于尝试不同的设计概念来说也很有价值。

## 在表格视图中创建静态单元格

打开XYZToDoItemTableViewController.m，添加代码

```
@interface XYZToDoItemTableViewController ()
@property NSMutableArray * toItemItems; // 0
@end

- (void)viewDidLoad
{
    [super viewDidLoad];

    self.toItemItems = [[NSMutableArray alloc] init]; // 1
    [self loadInitialData];
    [self.tableView registerClass:[UITableViewCell class] forCellReuseIdentifier:@"ListPrototypeCell"]; // 2
}

- (void)loadInitialData {
    XYZToDoItem *item1 = [[XYZToDoItem alloc] init];
    item1.itemName = @"Buy milk";
    [self.toItemItems addObject:item1];
    XYZToDoItem *item2 = [[XYZToDoItem alloc] init];
    item2.itemName = @"Buy eggs";
    [self.toItemItems addObject:item2];
    XYZToDoItem *item3 = [[XYZToDoItem alloc] init];
    item3.itemName = @"Read a book";
    [self.toItemItems addObject:item3];
}

#pragma mark - Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView // 3
{
    return 1;
}

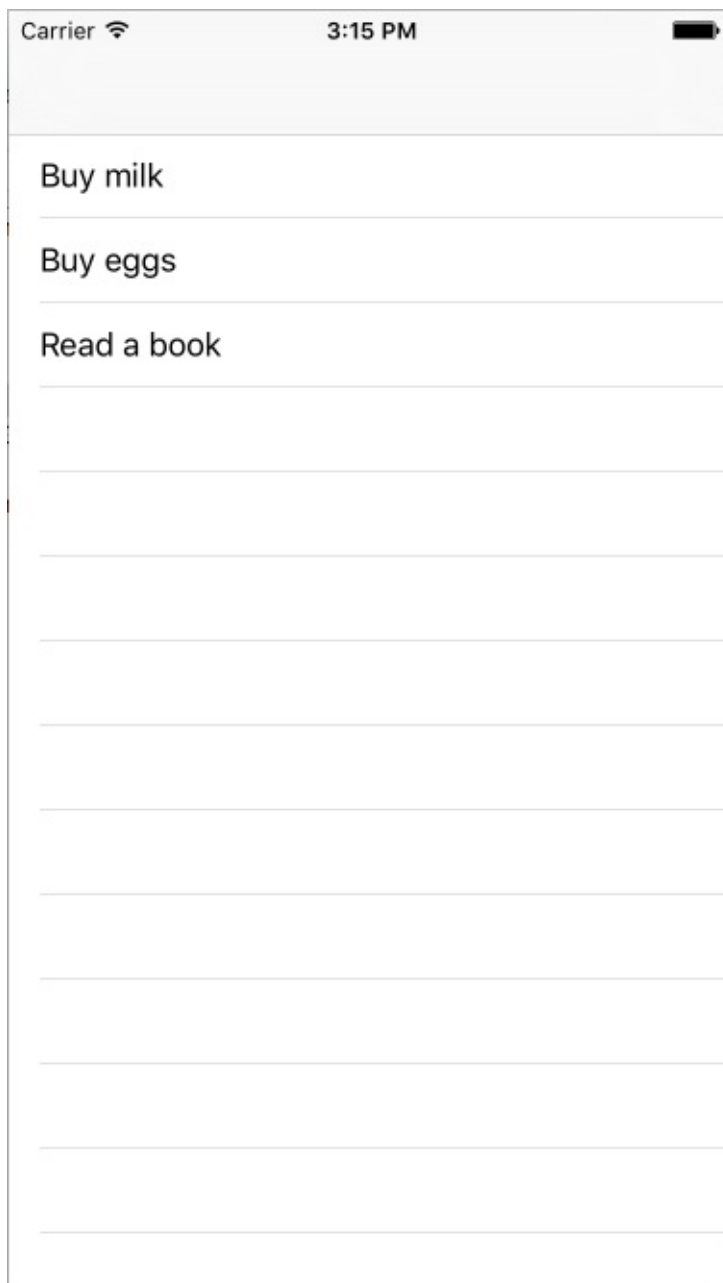
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSectionSection:(NSInteger)section // 4
{
    return [self.toItemItems count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath // 5
{
    static NSString *CellIdentifier = @"ListPrototypeCell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier forIndexPath:indexPath];
    XYZToDoItem *toDoItem = [self.toItemItems objectAtIndex:indexPath.row];
    cell.textLabel.text = toDoItem.itemName;
}
```

```
        if (todoItem.completed) {
            cell.accessoryType = UITableViewCellAccessoryCheckmark;
        } else {
            cell.accessoryType = UITableViewCellAccessoryNone;
        }
        return cell;
    }
}
```

1. 定义一个数组，用于存放ToDoItem数据
2. 初始化数组
3. 给UITableView注册UITableViewCell类型的table view cell
4. 以下是UITableViewDelegate, 在XYZToDoItemTableViewController实现，以便表格视图调用。本方法返回表格式图的section数
5. 本方法返回表格视图中显示的数据个数
6. 本方法返回一个UITableViewCell，即具体某一条目

检查点：运行您的应用程序。现在，您应该能看到一个表格视图，可以查看一下新表格视图在滚动时的整体效果。试试旋转模拟设备，注意观察单元格视图如何配置，才对其内容进行了正确布局。在表格视图中，您可以自由地进行大量操作。



完成后，应该设计如何从这个表格视图及其待办事项列表，浏览到您所创建的首个场景（即用户可以创建新待办事项的场景）。

## 添加过渡以向前浏览

您已经在应用中添加了两个视图控制器，但它们彼此之间没有连接。场景之间的转场，我们称为过渡。

创建过渡前，您需要先配置场景。首先，在导航控制器中包括待办事项列表表格视图控制器。回想之前的“定义交互”，我们知道导航控制器能提供导航栏，并且能够跟踪导航堆栈。您将在“add-to-do-item”场景的转场中添加导航栏按钮。

## 将导航控制器添加到表格视图控制器

打开XYZToDoItemTableViewController.m，添加代码

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

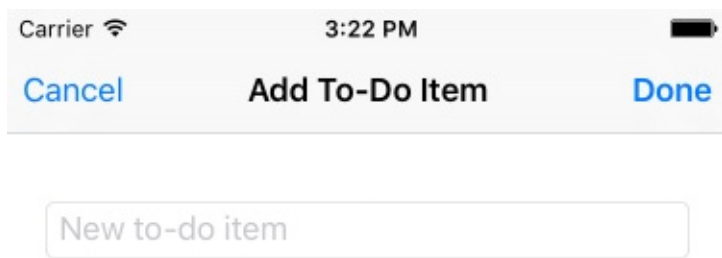
    self.navigationController.navigationBarHidden = NO; // 1
    self.navigationItem.title = @"My To-Do Item"; // 2
    self.navigationItem.rightBarButtonItem = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
                                                target:self
                                                action:@selector(triggerAddToDoItem)];

    self.toDoItems = [[NSMutableArray alloc] init];
    [self loadInitialData];
    [self.tableView registerClass:[UITableViewCell class] forCellReuseIdentifier:@"ListPrototypeCell"];
}

- (void)triggerAddToDoItem:(id)sender {
    XYZAddToDoItemViewController * vc = [[XYZAddToDoItemViewController alloc] init];
    [self.navigationController pushViewController:vc animated:YES]; // 4
}
```

1. 设置导航栏可见
2. 在导航栏上显示此视图控制器的名称
3. 在导航栏右侧添加一个按钮，用于启动另一个视图控制器。UIBarButtonItemAdd表示按钮样式是一个加号，@selector(triggerAddToDoItem:)表示按下按钮后将调用self的triggerAddToDoItem:方法。
4. 点击加号按钮后，我们给导航控制器压入一个新的视图控制器XYZAddToDoItemViewController

检查点：运行您的应用程序。点按添加按钮。您仍将看到添加项目场景，但返回浏览待办事项列表的按钮将消失，取而代之的是您添加的“Done”和“Cancel”这两个按钮。这些按钮尚未链接到任何操作，因此您虽然可以点按它们，但不会有任何反应。下个任务，我们将配置按钮以完成或取消编辑新的待办事项，并使用户返回待办事项列表。



---

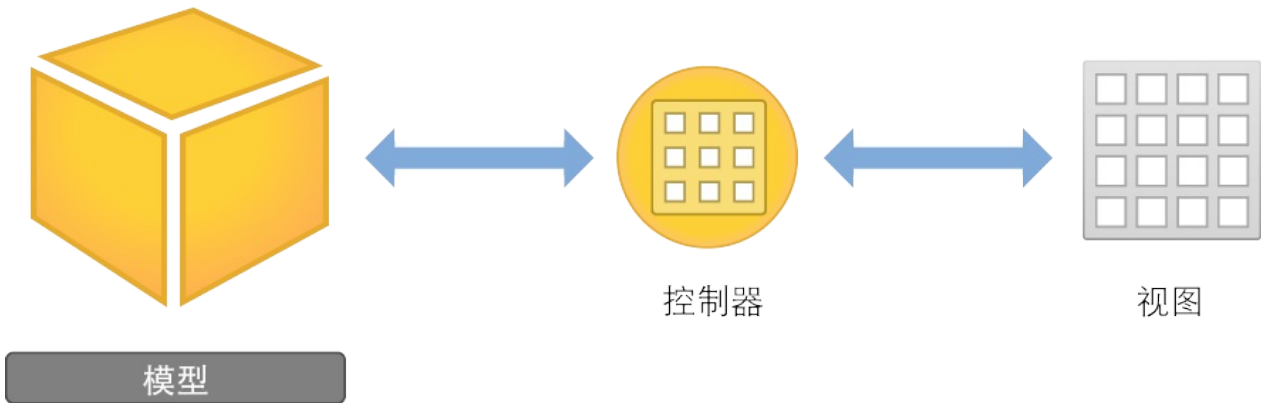
## 小结

---

现在，您已经完成了应用程序界面的开发工作。现在有两个视图控制器：一个用于给待办事项列表添加项目，另一个用于查看该列表；并且您可以在两个场景之间浏览。接下来要实现的功能，是让用户添加新的待办事项，并使其显示在列表中。下一部分讲述了如何处理数据以实现该行为。

## 整合数据

应用程序的数据模型由数据结构和（可选）自定义业务逻辑组成；自定义业务逻辑是让数据保持一致状态所必要的。在设计数据模型时，不应完全忽略应用程序的用户界面。但是，您肯定会想单独实现数据模型对象，而不依赖于特定的视图或视图控制器是否存在。保持数据与用户界面分开，有助于通用应用程序（可在 iPad 和 iPhone 双平台上运行的应用程序）的实现，也让代码复用变得更容易。



## 模型设计

如果需要储存的数据很小，那么 Foundation 框架类可能是您的最佳选择。您可以搜索现有的 Foundation 类，查看您可以使用哪些行为，而无需自己尝试实施同样的事情。例如，如果应用程序只需要跟踪字符串列表，则可以依赖 NSArray 和 NSString 来替您操作。在“处理 Foundation”中，您可以了解有关这些以及其他 Foundation 类的更多信息。

如果数据模型不仅要储存数据，还要求自定义业务逻辑，那么您可以编写一个自定义类。您应考虑如何将现有框架类合并到您自己的类的实现中。在自定义类中使用现有框架类，往往比重写更省时省力。例如，自定义类可能使用 NSMutableArray 来储存信息，但是会定义其自己的功能来处理该信息。

以下是设计数据模型时需要注意的一些问题：

您需要储存哪种类型的数据？您设计的数据模型应当能恰当地处理特定类型的内容，不管是储存文本、文稿、大图像，还是其他类型的信息。

您可以使用哪种数据结构？决定了什么地方应该使用框架类，什么地方需要定义具有自定义功能的类。

您如何将数据提供给用户界面？您的模型不应该直接与界面通信。如果要处理模型与界面之间的互动，需要为您的控制器添加逻辑。

## 模型实现

您需要了解更多有关 Objective-C 及其功能的信息，才能编写出优秀且高效的代码。虽然本指南描述了如何构建简单的应用程序，但在您自行编写具备完整功能的应用程序前，还需要更加熟悉该语言。

学习 Objective-C 的好方法有很多种。有的人通过阅读《Programming with Objective-C》（使用 Objective-C 编程）来了解其概念，然后编写一些小的测试应用程序来巩固对该语言的理解，并练习编写更好的代码。

有的人则直接跳到编程阶段，并在无法完成某些操作时，再去查找更多信息。如果您更喜欢这种方式，请将《Programming with Objective-C》（使用 Objective-C 编程）留作参考，当作了解各种概念的练习资料，并在开发时应用到应用程序中。

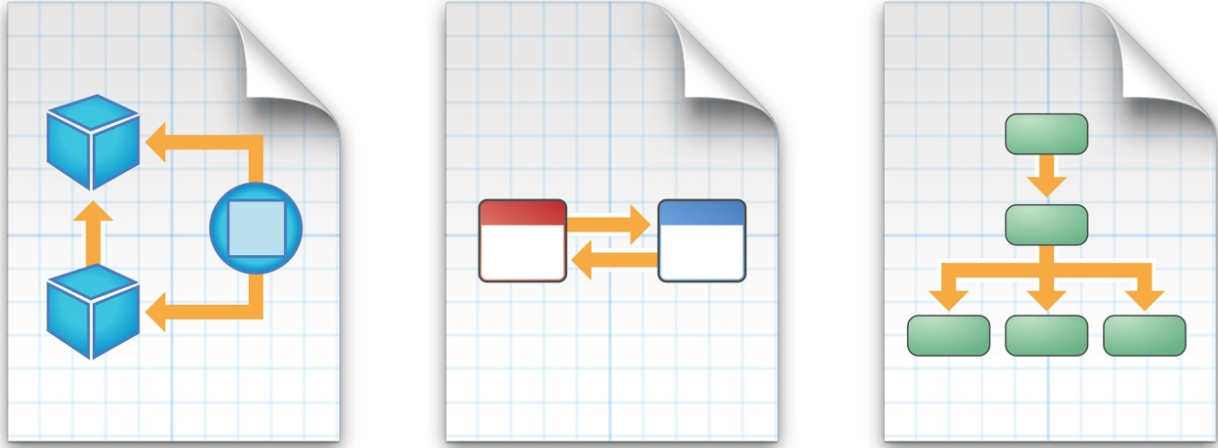
开发您的首个数据模型时，最重要的目标是使它能正常运作。仔细思考数据模型的结构，而不要急于将其完美化。开始实现

它之后，则要勇于反复重做和改进您的模型。



## 使用设计模式

设计模式可以解决常见的软件工程问题。模式是抽象设计，而非代码。采用一种设计，就是应用它的通用模式来满足您的特定需求。不管是创建哪种类型的应用程序，最好能先了解框架中使用的基本设计模式。了解设计模式有助于更高效地使用框架，并且可让您编写的程序复用程度更高、扩展能力更强和更容易修改。



## MVC

对于任何 iOS 应用程序而言，模型—视图—控制器 (MVC) 都是一个优秀设计的关键所在。MVC 会将应用程序中的对象分配给以下三种角色中的一种：模型、视图或者控制器。在这种模式中，模型会记录应用程序的数据，视图会显示用户界面并构成应用程序的内容，而控制器则会管理您的视图。通过响应用户的操作并使用内容填充视图，控制器充当了模型和视图二者之间通信的通道。

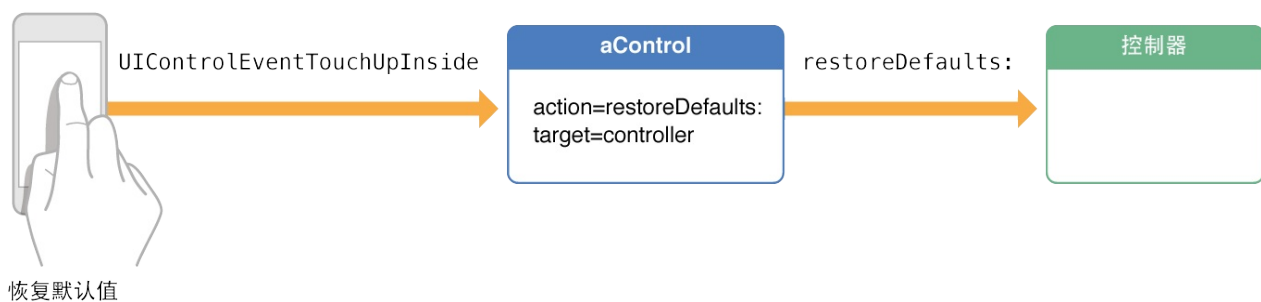


在构建 `ToDoList` 应用程序的过程中，您遵循的就是以 MVC 为中心的设计模式。在串联图中构建的界面组成了视图层 `XYZAddToDoItemViewController` 和 `XYZToDoListViewController` 就是管理视图的控制器。在“教程：添加数据”中，您需要结合数据模型来处理应用程序中的视图和控制器。在开始设计自己的应用程序时，很重要的一点就是以 MVC 为中心进行设计。

## 目标-操作

目标-操作从概念上讲是一个简单的设计：特定事件发生时，一个对象会向另一个对象发送信息。操作信息就是在源代码中定义的选择器，而目标（即接收信息的对象）则是能够执行该操作的对象（通常为视图控制器）。发送操作信息的对象通常为控制，例如按钮、滑块或开关，它能够触发事件对用户的交互操作（例如轻按、拖移或者值更改）作出响应。

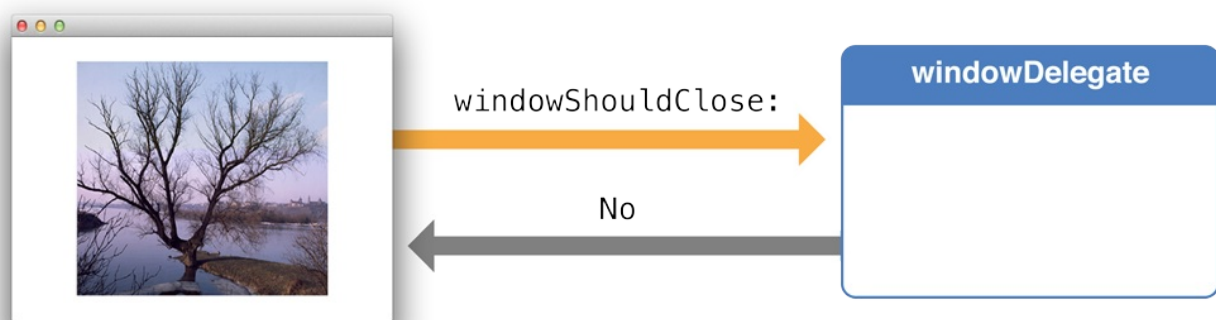
例如，假设您想要在每次用户轻按“恢复默认”按钮（在用户界面中创建）时恢复应用程序中的默认设置。首先，您需要实施操作 `restoreDefaults:` 来执行恢复默认设置的逻辑。接着，注册按钮的 `Touch Up Inside` 事件，将 `restoreDefaults:` 操作方法发送到实施该方法的视图控制器。



在 `ToDoList` 应用程序中，您已经使用过目标-操作模式。当用户轻按 `XYZAddToDoItemViewController` 中的“完成”按钮时，会触发 `unwindToList:` 操作。在这种情况下，“完成”按钮是发送信息的对象，目标对象是 `XYZToDoListViewController`，操作信息是 `unwindToList:`，而触发操作信息被发送的事件则是用户轻按“完成”按钮这一操作。目标-操作是定义交互以及在应用程序各部分之间发送信息的强大机制。

## 委托

委托是一种简单而强大的模式。在此模式中，应用程序中的一个对象代表另一个对象，或与另一个对象协调工作。授权对象保留对另一个对象（委托对象）的引用，并适时向委托对象发送信息。该信息会告诉事件的委托对象，授权对象即将处理或刚处理了某个事件。委托对象可能会对该信息作出如下响应：更新其本身或应用程序中其他对象的外观或状态，在某些情况下，它会返回一个值来反映待处理的事件该如何处理。



委托模式不仅普遍用于既有的框架类，而且也可应用在应用程序的两个自定义对象之间。常见的设计是将委托作为一种手段，允许子视图控制器将某些值（通常为用户输入的值）传达到父视图控制器。

目前您还没有使用过委托，但是在“教程：添加数据”中，当您将其他行为添加到 `XYZToDoListViewController` 类时，这就是委托的示例。

在 iOS 应用程序的开发过程中，会经常遇到一些较为常见的设计模式，但它们只是冰山一角。随着对 Objective-C 学习的深入，您还会发现其他可在应用程序中使用的设计模式。

## 教程：添加数据

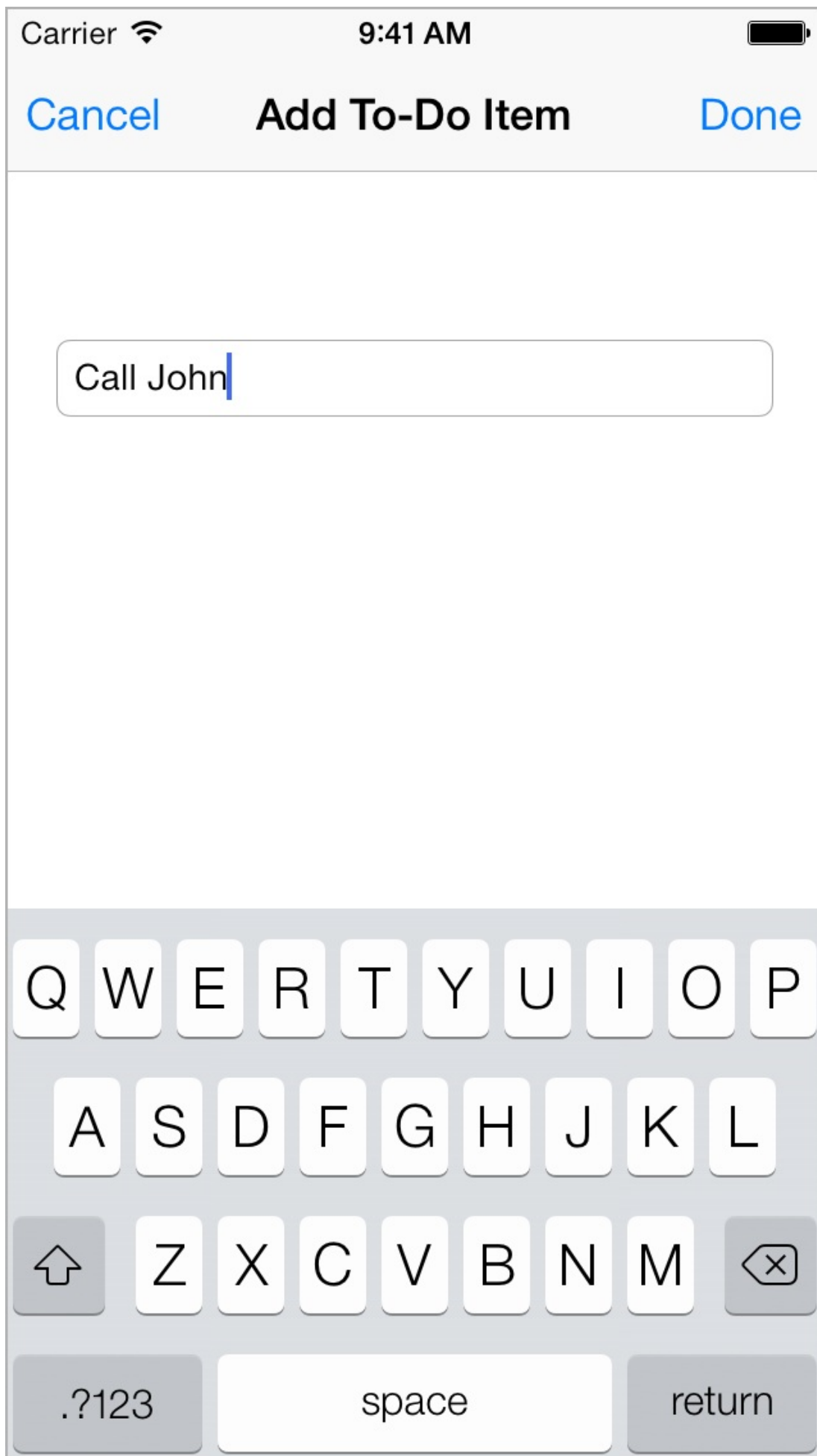
---

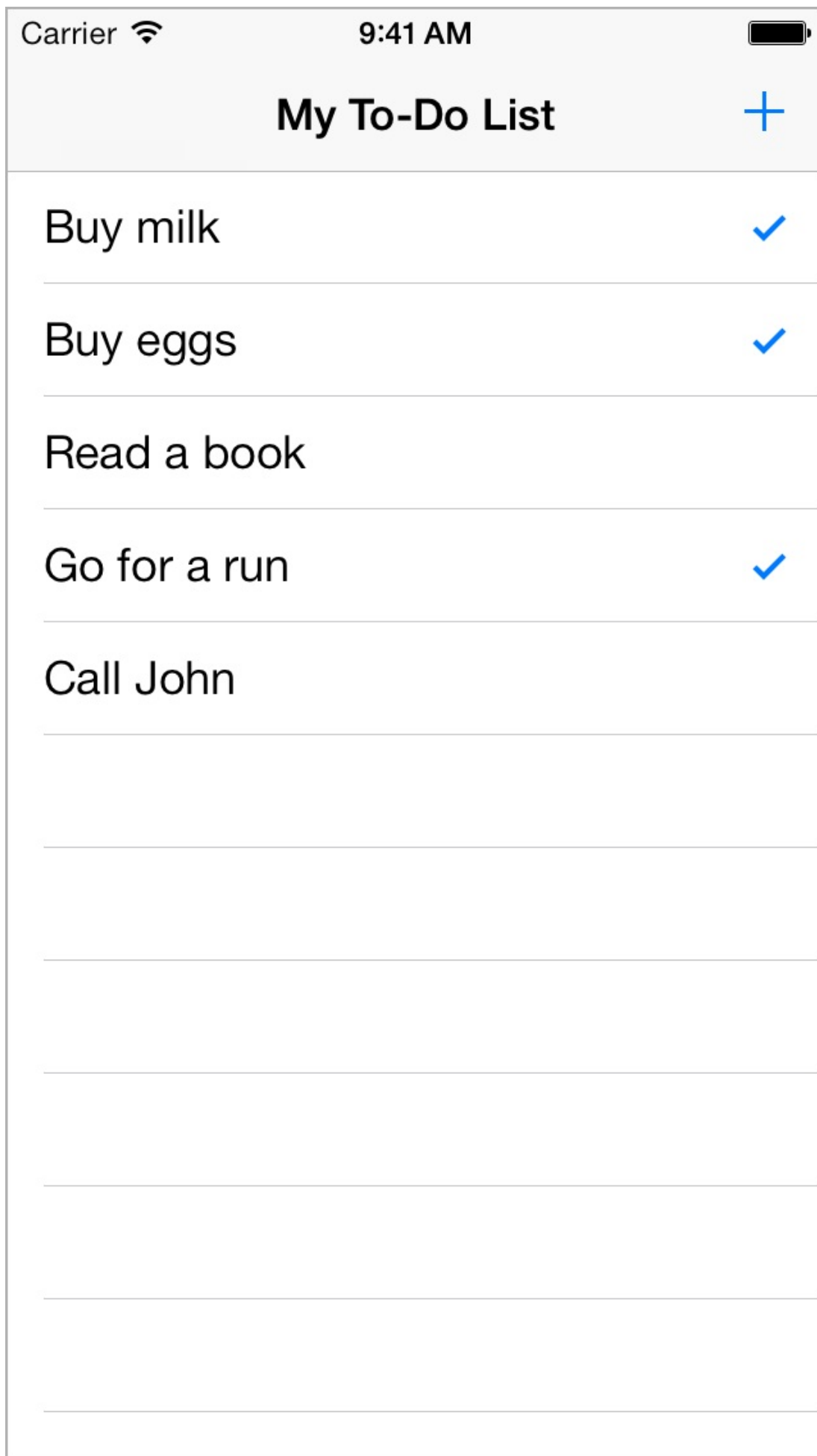
本教程以第二个教程（“教程：串联图”）中创建的项目为基础。您将用到从使用设计模式、使用 Foundation 以及编写自定义类中学到的知识，在 *ToDoList* 应用程序中添加对动态数据的支持。

本教程讲述了以下操作：

- 使用常见的 Foundation 类
- 创建自定义数据类
- 实现委托和数据源协议
- 在视图控制器之间传递数据

完成本教程中的所有步骤后，您的应用程序外观大致是这样的：





## 创建数据类

现在就开始吧，请在 Xcode 中打开您的现有项目。

目前，使用串联图的 `ToDoList` 应用程序有一个界面和一个导航方案。现在，是时候使用模型对象来添加数据储存和行为了。

应用程序的目标在于创建一个待办事项列表，因此首先您将创建一个自定义类 `XYZToDoItem` 来表示单个待办事项。您应该记得，`XYZToDoItem` 类已经在编写自定义类中讨论过。

### 创建 `XYZToDoItem` 类

1. 选取“File”>“New”>“File”（或按下 `Command-N`）。
2. 这时将会出现一个对话框，提示您为新文件选取模板。从左侧的 iOS 下方选择“Cocoa Touch”。
3. 选择“Objective-C Class”，并点按“Next”。
4. 在“Class”栏中，在 XYZ 前缀后键入 `ToDoItem`。
5. 从“Subclass of”弹出式菜单中选取“`NSObject`”。如果您完全按照本教程操作，那么在这个步骤之前，“Class”标题可能是 `XYZToDoItemViewController`。选取 `NSObject` 作为“Subclass of”后，Xcode 会知道您创建了一个正常的自定义类，并移除了它先前添加的 `ViewController` 文本。
6. 点按“Next”。
7. 存储位置默认为您的项目目录。此处无需更改。
8. “Group”选项默认为您的应用程序名称“`ToDoList`”。此处无需更改。
9. “Targets”部分默认选定您的应用程序，未选定应用程序的测试。好极了，这些都无需更改。
10. 点按“Create”。

`XYZToDoItem` 类很容易实现。它具有项目名称、创建日期，以及该项目是否已完成等属性。继续将这些属性添加到 `XYZToDoItem` 类接口。

### 配置 `XYZToDoItem` 类

1. 在项目导航器中，选择 `XYZToDoItem.h`。
2. 将以下属性添加到该接口，使声明如下所示：

```
@interface XYZToDoItem : NSObject

@property NSString *itemName;
@property BOOL completed;
@property (readonly) NSDate *creationDate;

@end
```

检查点：通过选取“Product”>“Build”（或按下 `Command-B`）来生成项目。尽管该新类尚未实现任何功能，但是生成它有助于编译器验证任何拼写错误。如果发现错误，请及时修正：通读编辑器提供的警告或错误，然后回顾本教程中的说明，确保所有内容与此处的描述相符。

## 载入数据

您现在有一个类，可以用它作为基础来为单个列表项目创建并储存数据。您还需要保留一个项目列表。在 `XYZToDoItemViewController` 类中跟踪此内容较为合适，视图控制器负责协调模型和视图，所以需要对该模型进行引用。

Foundation 框架有一个 `NSMutableArray` 类，很适合跟踪项目列表。此处必须使用可变数组，这样用户就可以将项目添加到数组。因为不可变数组 `NSArray` 在其初始化后将不允许添加项目。

要使用数组，您需要声明并创建它。可以通过分配并初始化数组来完成。

## 要分配并初始化数组

1. 在项目导航器中，选择 XYZToDoListViewController.m。由于项目数组是表格视图控制器的实现细节，所以应该在 .m 文件中进行声明，而不是 .h 文件。此操作可让项目数组成为您自定类的私有数组。
2. 将以下属性添加到接口类别中，它是由 Xcode 在您的自定义表格视图控制器类中创建的。声明应该是这样的：

```
@interface XYZToDoListViewController ()

@property NSMutableArray *toDoItems;

@end
```

1. 在 viewDidLoad 方法中分配并初始化 toDoItems 数组：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    self.toDoItems = [[NSMutableArray alloc] init];
}
```

viewDidLoad 的实际代码中有一些附加行被注释掉了，那些行是 Xcode 创建 XYZListViewController 时插入的。保留与否都没有影响。

现在，您已经拥有了一个可以添加项目的数组。添加项目将在单独的方法 loadInitialData 中进行，并将通过 viewDidLoad 调用该方法。由于此代码是一个模块化任务，所以会进入其自身的方法中。当然您也可以将方法分离出来，从而提高代码的可读性。在真正的应用程序中，此方法可能会从某种永久储存形式载入数据，例如文件。现在，我们的目标是了解表格视图如何处理自定义数据项目，那么让我们创建一些测试数据来体验一下吧。

以创建数组的方式创建项目：分配并初始化。然后，给项目命名。该名称将显示在表格视图中。按照此方法创建一组项目。

## 载入初始数据

1. 在 @implementation 行下方，添加一个新方法 loadInitialData。

```
- (void)loadInitialData {
}
```

1. 在此方法中，创建几个列表项目，并将它们添加到数组。```
2. (void)loadInitialData { XYZToDoItem item1 = [[XYZToDoItem alloc] init]; item1.itemName = @"Buy milk"; [self.toDoItems addObject:item1]; XYZToDoItem item2 = [[XYZToDoItem alloc] init]; item2.itemName = @"Buy eggs"; [self.toDoItems addObject:item2]; XYZToDoItem \*item3 = [[XYZToDoItem alloc] init]; item3.itemName = @"Read a book"; [self.toDoItems addObject:item3]; } ```
3. 在 viewDidLoad 方法中调用 loadInitialData。

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    self.toDoItems = [[NSMutableArray alloc] init];
    [self loadInitialData];
}
```

检查点：通过选取“Product”>“Build”来生成项目。您应该会在 `loadInitialData` 方法的代码行上看到大量错误。第一行是出错的关键所在，错误提示应该是“Use of undeclared identifier XYZToDoItem”。这说明编译器在编译 `XYZToDoListViewController` 时不知道 `XYZToDoItem`。编译器比较特别，您需要明确告知它应当注意什么。

## 让编译器注意您的自定列表项目类

1. 在 `XYZToDoListViewController.m` 文件的顶部附近找到 `#import "XYZToDoListViewController.h"` 行。
2. 紧接着在其下方添加以下行：

```
#import "XYZToDoItem.h"
```

检查点：通过选取“Product”>“Build”来生成项目。项目生成时应该没有错误。

## 显示数据

目前，表格视图具有一个可变数组，预填充了几个示例待办事项。现在您需要在表格视图中显示数据。

通过让 `XYZToDoListViewController` 成为表格视图的数据源，可以实现这一点。无论要让什么成为表格视图的数据源，都需要实施 `UITableViewDataSource` 协议。需要实施的方法正是您在第二个教程中注释掉的那些。创建有效的表格视图需要三个方法。第一个方法是 `numberOfSectionsInTableView:`，它告诉表格视图要显示几个部分。对于此应用程序，表格视图只需要显示一个部分，所以实现比较简单。

### 在表格中显示一个部分

1. 在项目导航器中，选择 `XYZToDoListViewController.m`。
2. 如果您在第二个教程中注释掉了表格视图数据源方法，现在请移除那些注释标记。
3. 模板实现的部分如下所示。您想要单个部分，所以需要移除警告行并将返回值由 0 更改为 1。

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    #warning Potentially incomplete method implementation.
    // Return the number of sections.
    return 0;
}
```

1. 更改 `numberOfSectionsInTableView:` 数据源方法以便返回单个部分，像这样：

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    // Return the number of sections.
    return 1;
}
```

下一个方法 `tableView:numberOfRowsInSection:` 告诉表格视图要在给定部分中显示几行。现在表格中有一个部分，并且每个待办事项在表格视图中都应该有它自己的行。这意味着行数应该等于 `toDoItems` 数组中的 `XYZToDoItem` 对象数。

### 返回表格中的行数

1. 在项目导航器中，选择 `XYZToDoListViewController.m`。
2. 您可看到模板实现的部分是这样的。您想要返回所拥有的列表项目的数量。幸运的是，`NSArray` 有一个很方便的方法，称为 `count`，它会返回数组中的项目数，因此行数是 `[self.toDoItems count]`。



```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    #warning Incomplete method implementation.
    // Return the number of rows in the section.
    return 0;
}
```

1. 更改 tableView:numberOfRowsInSection: 数据源方法，使其返回正确的行数。

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    // Return the number of rows in the section.
    return [self.todoItems count];
}
```

最后一个方法，tableView:cellForRowAtIndexPath: 请求一个单元格来显示给定行。到现在为止，您只是处理了代码，但是界面的绝大部分是针对行显示的单元格。幸运的是，Xcode 可让您轻松地在 Interface Builder 中设计自定义单元格。首个任务是设计您的单元格，并告诉表格视图不要使用静态内容，而要使用具有动态内容的原型单元格。

## 在表格中显示单元格

1. 在项目导航器中，选择 XYZToDoListViewController.m。
2. 找到 tableView:cellForRowAtIndexPath: 数据源方法。模板实现是这样的：

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier forIndexPath:indexPath];

    // Configure the cell...

    return cell;
}
```

该模板执行多个任务。它会创建一个变量来保存单元格的标识符，向表格视图请求具有该标识符的单元格，添加一个注释注明配置该单元格的代码应该写在哪里，然后返回该单元格。要让此代码为您的应用程序所用，需要将标识符更改为您在串联图中设定的标识符，然后添加代码来配置该单元格。

1. 将单元格标识符更改为您在串联图中设定的标识符。为了避免拼写错误，请将串联图中的标识符拷贝并粘贴到实现文件中。该单元格标识符行现在应该是这样的：

```
static NSString *CellIdentifier = @"ListPrototypeCell";
```

1. 在 return 语句前，添加以下代码行：

```
XYZToDoItem *todoItem = [self.todoItems objectAtIndex:indexPath.row];
cell.textLabel.text = todoItem.itemName;
```

您的 tableView:cellForRowAtIndexPath: 方法应如下图所示：

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"ListPrototypeCell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier forIndexPath:indexPath];
    XYZToDoItem *todoItem = [self.todoItems objectAtIndex:indexPath.row];
```

```
cell.textLabel.text = toItem.itemName;
return cell;
}
```

检查点：运行您的应用程序。您在 `loadInitialData` 中添加的项目列表应该在表格视图中显示为单元格。

## 将项目标记为已完成

如果无法将待办事项列表中的项目标记为已完成，那么这个待办事项列表还不够好。现在，让我们来添加这样的支持。一个简单的界面应该可以在用户轻按单元格时切换完成状态，并在已完成的项目旁边显示勾号。幸运的是，表格视图附带了一些内建行为，您可以利用这些行为来实现这样的简单界面。需要注意的是，在用户轻按单元格时，表格视图要通知它们的委托。所以我们的任务是写一段代码，对用户轻按表格中的待办事项这个操作作出响应。

您在串联图中配置 `XYZToDoListViewController` 时，Xcode 就已经让它成为表格视图的委托了。您要做的只是实现 `tableView:didSelectRowAtIndexPath:` 委托方法，使其响应用户轻按，并在适当时候更新您的待办事项列表。

选定单元格后，表格视图会调用 `tableView:didSelectRowAtIndexPath:` 委托方法，来查看它应如何处理选择操作。在此方法中，您需要编写代码来更新待办项目的完成状态。

## 将项目标记为已完成或未完成

您的 `tableView:didSelectRowAtIndexPath:` 方法应如下图所示：

```
(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [tableView deselectRowAtIndexPath:indexPath animated:NO]; // 1
    XYZToDoItem *tappedItem = [self.toDoItems objectAtIndex:indexPath.row]; // 2
    tappedItem.completed = !tappedItem.completed; // 3
    [tableView reloadRowsAtIndexPaths:@[indexPath] withRowAnimation:UITableViewRowAnimationNone]; // 4
}
```

1. 您想要响应轻按，但并不想让单元格保持选定状态。
2. 在 `toDoItems` 数组中搜索相应的 `XYZToDoItem`
3. 切换被轻按项目的完成状态
4. 告诉表格视图重新载入您刚更新过数据的行

检查点：运行您的应用程序。您在 `loadInitialData` 中添加的项目列表在表格视图中显示为单元格。但当您轻按项目时，没有任何反应。为什么呢？

原因是您尚未配置显示项目完成状态的表格视图单元格。要实现此功能，您需要回到 `tableView:cellForRowAtIndexPath:` 方法，并配置单元格以在项目完成时显示指示。

指示项目已完成的一种方式是在其旁边放置一个勾号。幸运的是，表格视图右边可以有一个单元格附属物。默认情况下，单元格中没有任何附属物；不过您可以进行更改，使其显示不同的附属物。其中的一个附属物就是勾号。您要做的就是根据待办事项的完成状态，设定单元格的附属物。

## 显示项目的完成状态

您的 `tableView:cellForRowAtIndexPath:` 方法现在应如下图所示

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"ListPrototypeCell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier forIndexPath:indexPath];
    XYZToDoItem *toDoItem = [self.toDoItems objectAtIndex:indexPath.row];
```

```
cell.textLabel.text = todoItem.itemName;
if (todoItem.completed) {
    cell.accessoryType = UITableViewCellAccessoryCheckmark;
} else {
    cell.accessoryType = UITableViewCellAccessoryNone;
}
return cell;
}
```

检查点：运行应用程序。您在 loadInitialData 中添加的项目列表在表格视图中显示为单元格。轻按项目时，其旁边应该出现一个勾号。如果您再次轻按同一项目，勾号会消失。