

# 马上着手开发 iOS 应用程序 (Start Developing iOS Apps Today)

# 目录

介绍 5

设置 6

获取工具 7

教程：基础 8

创建新项目 9

熟悉 Xcode 11

运行 iOS Simulator 12

检查源代码 14

创建串联图 17

将场景添加到串联图中 19

测试更改 23

构建基本界面 23

小结 28

构建应用程序 29

应用程序开发过程 30

定义概念 30

设计用户界面 31

定义交互 31

实现行为 32

    对象是应用程序的基石 32

    类是对象的蓝图 32

    对象通过消息通信 33

    协议定义消息发送契约 33

整合数据 34

    使用正确的资源 34

    整合真实的数据 34

设计用户界面 35

视图层次 35

使用视图构建界面 36

使用串联图来布局视图 37  
使用检查器来配置视图 38  
使用 Auto Layout 来定位视图 39

定义交互 40  
视图控制器 40  
操作 (Action) 41  
**Outlet** 41  
控制 (Control) 42  
导航控制器 42  
使用串联图来定义导航 43

教程：串联图 45  
采用 Auto Layout 46  
创建第二场景 48  
在表格视图中显示静态内容 50  
添加过渡以向前浏览 51  
创建自定视图控制器 59  
跳转过渡以返回 62  
小结 64

应用程序的实现 65

整合数据 66  
模型设计 66  
模型实现 67

使用设计模式 68  
**MVC** 68  
目标-操作 69  
委托 69

处理 **Foundation** 71  
值对象 71  
    字符串 72  
    数字 73  
集对象 73  
    数组 74  
    集合 77  
    字典 78

使用 NSNull 表示 nil	80
编写自定类	81
声明并实现类	82
接口	82
实现	82
储存对象的数据的属性	83
方法用来定义对象的行为	84
方法参数	85
实现方法	85
教程：添加数据	87
创建数据类	88
载入数据	89
显示数据	91
将项目标记为已完成	95
添加新项目	98
小结	104
后续步骤	105
iOS 技术	106
用户界面	106
游戏	107
数据	107
Media	108
查找信息	109
通过关联帮助文章来获得 Xcode 指导	109
使用指南来获得通用概述和概念概述	111
使用 API 参考来获得类信息	112
使用 Quick Help 来获得关联的源代码信息	117
通过示例代码来查看实际用法	119
接下来做什么	121
让 ToDoList 应用程序提高一个档次	122
文稿修订历史	123

# 介绍

- [设置](#) (第 6 页)
- [教程：基础](#) (第 8 页)

# 设置

《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”FREE。

Xcode 将下载到您的 /Applications 目录中。

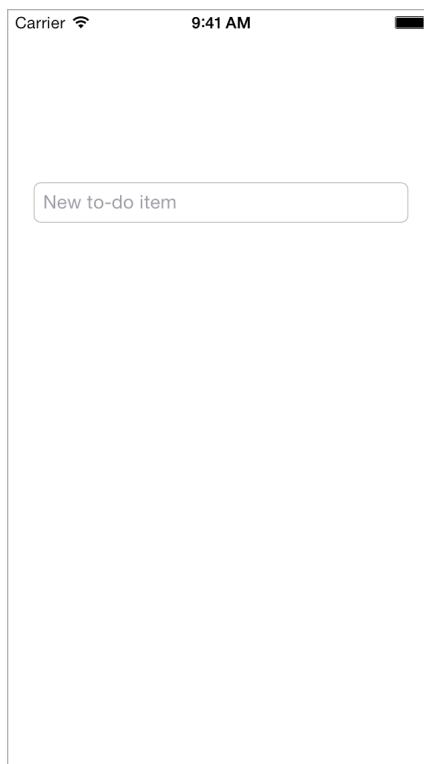
# 教程：基础

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。

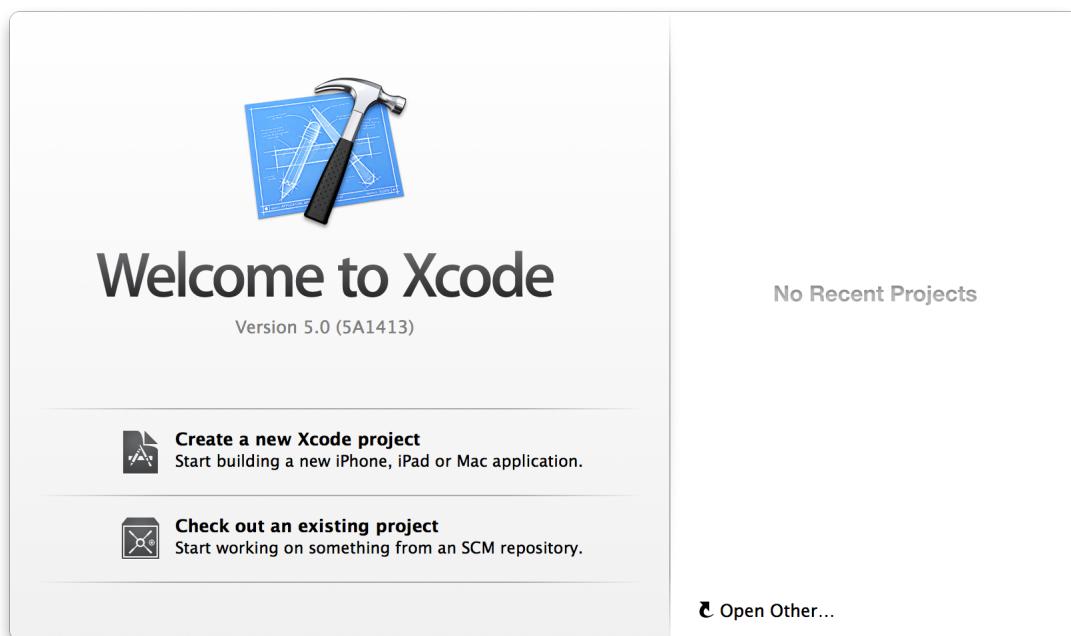
Xcode 随附了几个内建应用程序模板，可用于开发常见的 iOS 应用程序，如游戏、基于标签浏览的应用程序和基于表格视图的应用程序。这些模板大都预先配置了界面和源代码文件，可作为您进行开发工作的起点。本教程会从最基础的模板开始：Empty Application。

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。

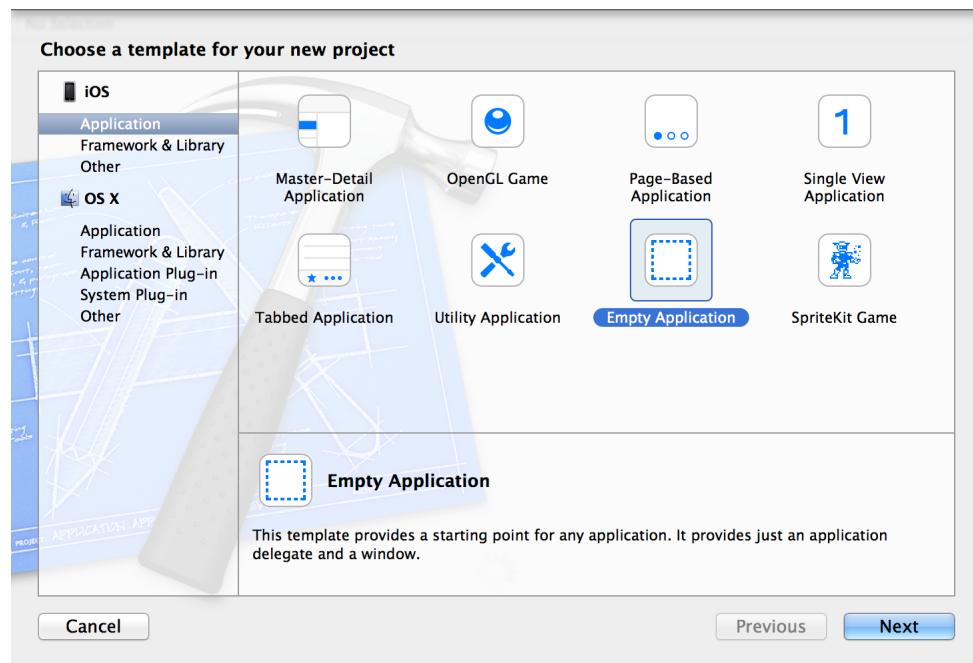
Xcode 欢迎窗口会出现。



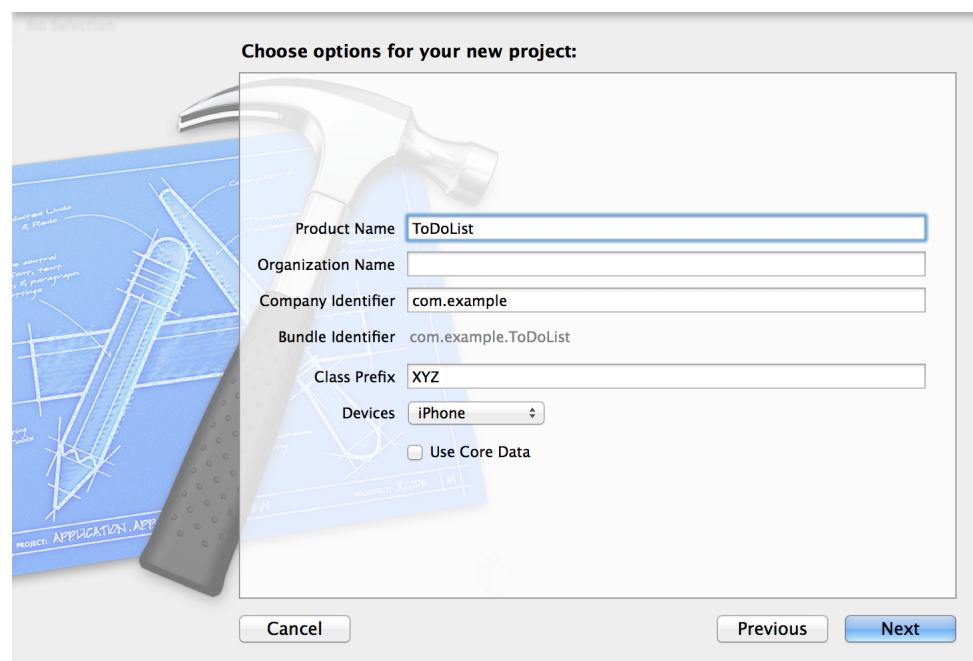
如果出现的是项目窗口，而不是欢迎窗口，请不要着急；这说明您可能曾在 Xcode 中创建或打开过项目。您只需在接下来的步骤中，使用菜单项来创建项目。

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“Empty 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单为您创建的类。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。

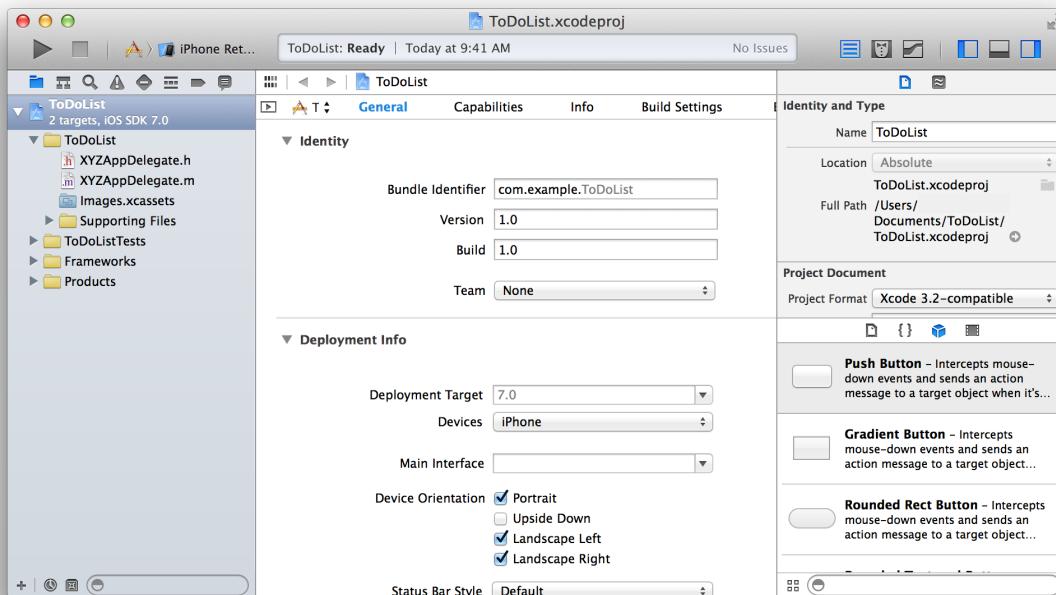
## 6. 从“Devices”弹出式菜单中选取“iPhone”。

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”。

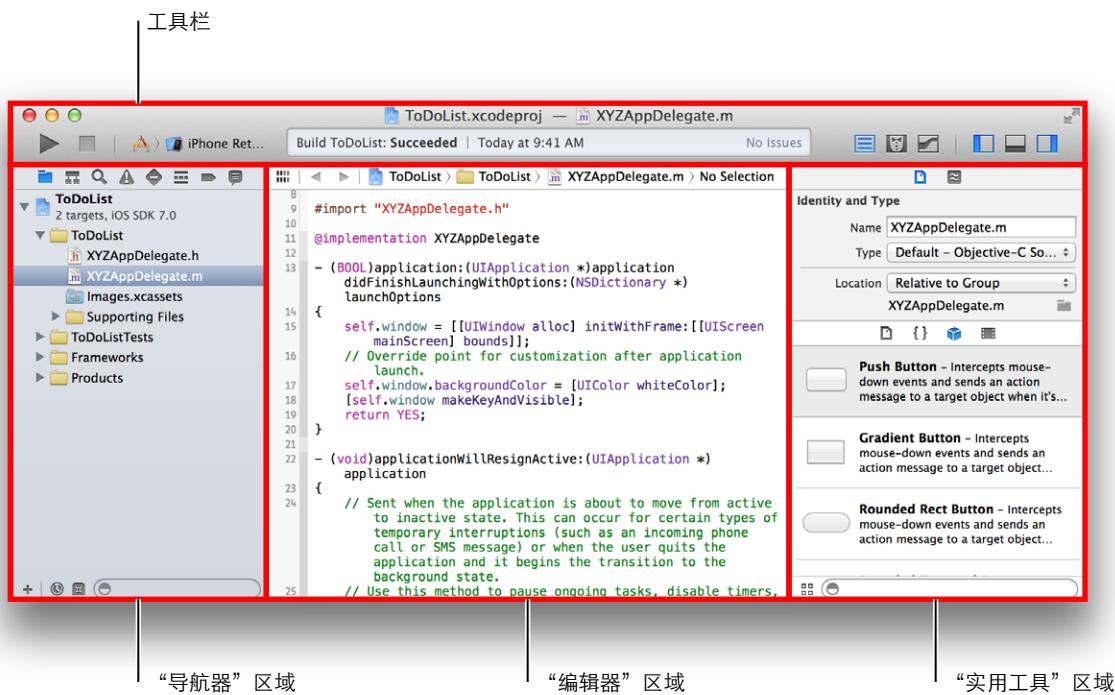
Xcode 会在工作区窗口中打开新项目，窗口的外观类似：



## 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

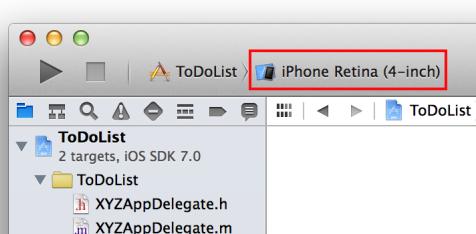
由于项目是基于 Xcode 模板创建的，因此基本的应用程序环境已经自动为您设置好了。即使没有编写任何代码，也可以构建和运行 **Empty Application** 模板，而无需进行任何额外的配置。

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)**”。

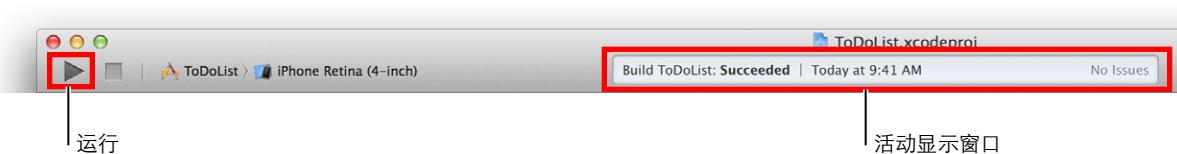
在 **iOS Simulator** 中运行应用程序

- 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。

- 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）。

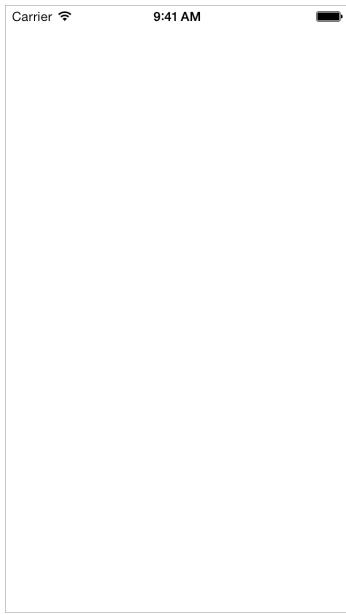
如果是首次运行应用程序，Xcode 会询问您是否要在 Mac 上启用开发者模式。开发者模式可让 Xcode 访问特定的调试功能，无需每次都输入密码。请决定是否要启用开发者模式，然后按照提示操作。如果选取不启用，可能稍后会要求您输入密码。本教程假定已启用了开发者模式。

- 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

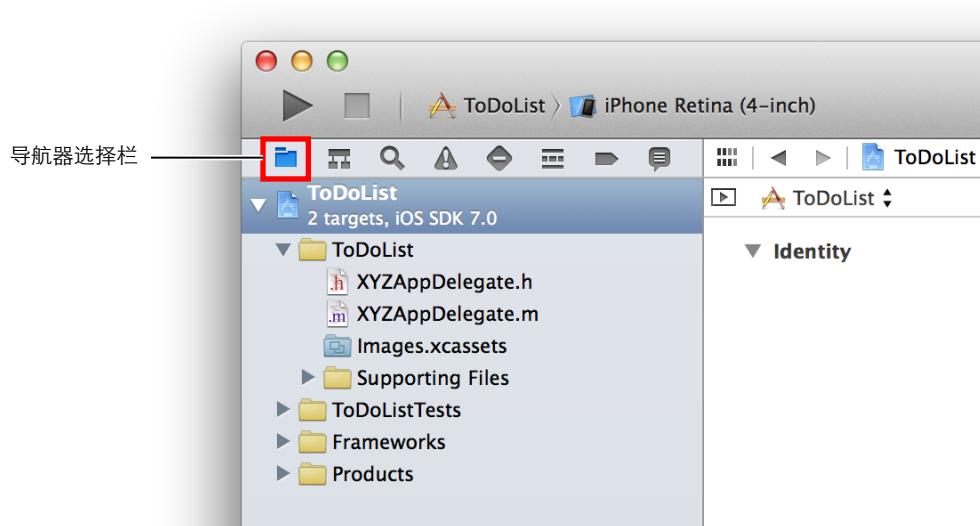
Empty Application 模板附带了少量现成的源代码，用于设置应用程序环境。大多数工作都由 UIApplicationMain 函数来完成，它在项目的 main.m 源文件中会被自动调用。UIApplicationMain 函数会创建一个应用程序对象来设置应用程序基础结构，以配合 iOS 系统运作。包括创建一个运行循环，将输入事件传递给应用程序。

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

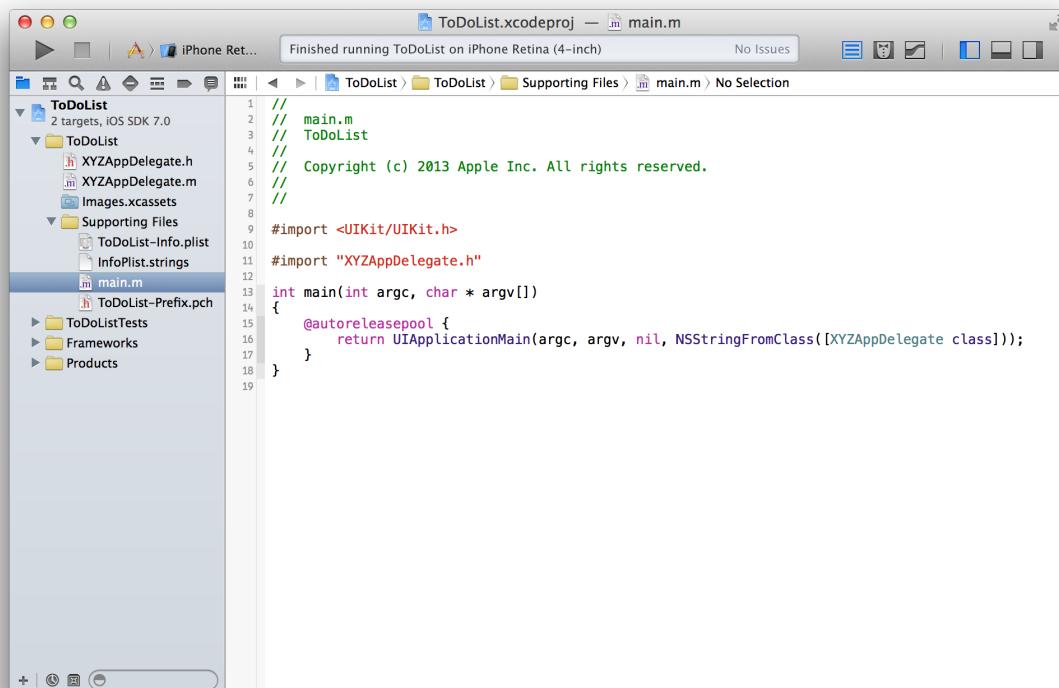
1. 请确定项目导航器已在导航器区域中打开。

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。



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。
3. 选择 main.m。

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 Empty 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。）

## 创建串联图

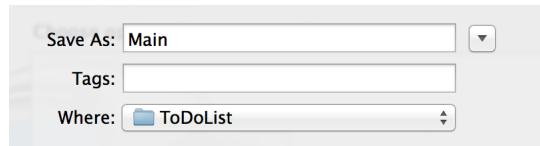
现在，您应该已经准备好创建应用程序的串联图了。串联图能直观展示应用程序的用户界面、显示内容屏幕以及它们之间的转换。您可以使用串联图对驱动应用程序的流程或构思进行布局。

要了解串联图融入应用程序的方法，在本教程中您可以手动创建一个，然后将其添加到应用程序内。与开始使用的 **Empty Application** 模板不同，其他 Xcode 模板包含了预配置的串联图，提供了视图、视图控制器和相关的源文件，用于设置该类型的应用程序的基本架构。手动配置完串联图后，您将会看到各个部分是如何配合工作的。然后，您就可以使用预配置有串联图的项目模板，开始应用程序开发了，这会节省不少时间。

### 创建新的串联图

1. 选取“File”>“New”>“File”（或按下 Command-N）。  
这时将会出现一个对话框，提示您为新文件选取模板。
2. 在左边，选择 iOS 下方的“User Interface”。
3. 点按“Storyboard”，然后点按“Next”。
4. 在“Devices”选项中，选择“iPhone”。
5. 点按“Next”。  
这时会出现一个对话框，提示您选取一个位置并为新串联图命名。
6. 在“Save As”栏中，将文件命名为 Main。

7. 请确定将文件与项目存储在同一个目录中。



8. 在“Group”选项中，选择“ToDoList”。  
9. 对于“Targets”，选择“ToDoList”旁边的复选框。  
此选项可让 Xcode 在构建应用程序时包括新的串联图。  
10. 点按“Create”。

这时，一个新的串联图文件将创建完成，并被添加到项目中。您可以在这个文件中工作，对应用程序的内容进行布局。

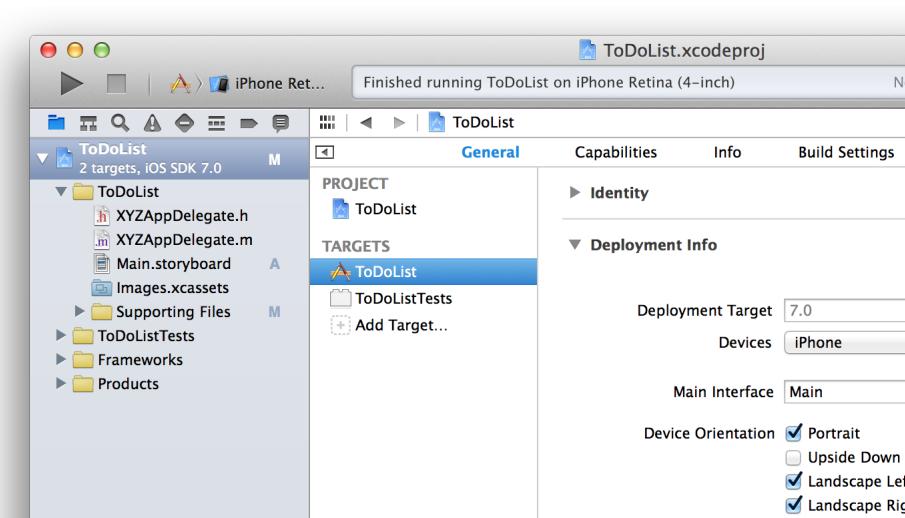
现在，需要让 Xcode 知道您想将此串联图用作应用程序中的界面。开始时，应用程序对象会检查该应用程序是否配置了主界面。如果已配置，应用程序对象会在应用程序启动时载入已定义的串联图。

#### 将串联图设为应用程序的主界面

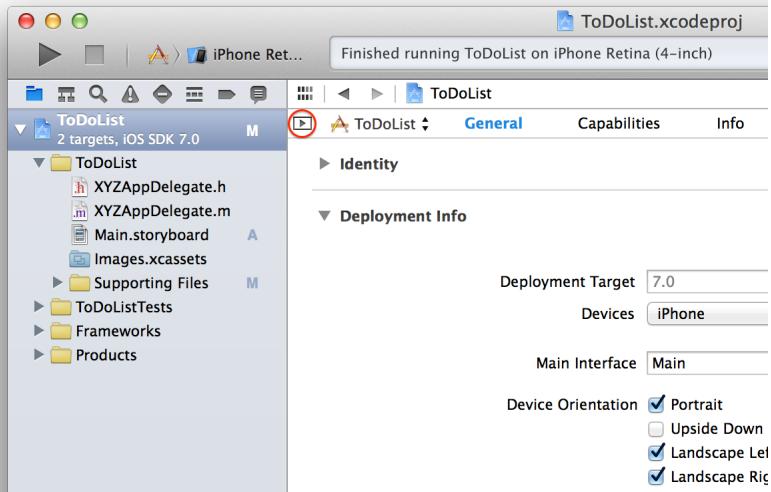
1. 在项目导航器中，选择您的项目。

在工作区窗口的编辑器区域，Xcode 会显示项目编辑器，可让您查看和编辑与应用程序构建有关的细节。

2. 在“Targets”的下方，选择“ToDoList”。



如果“Project”和“Targets”列表未显示在项目编辑器中，请点按编辑器面板左上角的显示三角形。



3. 选择“General”标签。
4. 在“Deployment Info”的下方，找到“Main Interface”选项。
5. 选择您的串联图：Main.storyboard。

## 将场景添加到串联图中

现在，串联图已经创建完成，接下来让我们开始添加应用程序的内容吧。Xcode 提供了对象库，您可以将库中的对象添加到串联图文件。其中的一些对象属于视图中的用户界面元素，例如按钮和文本栏。其他的对象则定义了应用程序的行为，但它们不会显示在屏幕上，如视图控制器和手势识别器。

首先，请将视图控制器添加到串联图中。视图控制器管理了相应的视图及其分视图。在下一章，“[App Development Process](#)”（第 30 页）中，您将会了解到有关视图角色和视图控制器的更多信息。

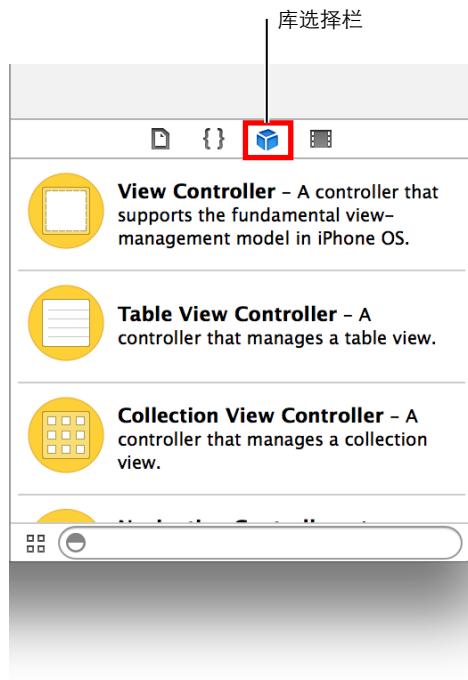
### 将视图控制器添加到串联图

1. 在项目导航器中，选择 Main.storyboard。

Xcode 会在编辑器区域的 **Interface Builder**（其可视化界面编辑器）中打开串联图。由于串联图是空的，因此您看到的是空白画布。画布可用来添加和排列用户界面的元素。

2. 打开对象库。

对象库出现在实用工具区域的底部。如果看不到对象库，您可以点按其按钮，即库选择栏中左起第三个按钮。（如果看不到实用工具区域，可以选取“View”>“Utilities”>“Show Utilities”来显示。）



列表显示每个对象的名称、描述和可视化表示。

### 3. 将“View Controller”对象从列表拖到画布中。

如果在对象库中找不到标题为“View Controller”的对象，请在列表下方的文本栏中键入内容来过滤对象列表。键入 View Controller，那么过滤后的列表中就只会显示视图控制器对象。

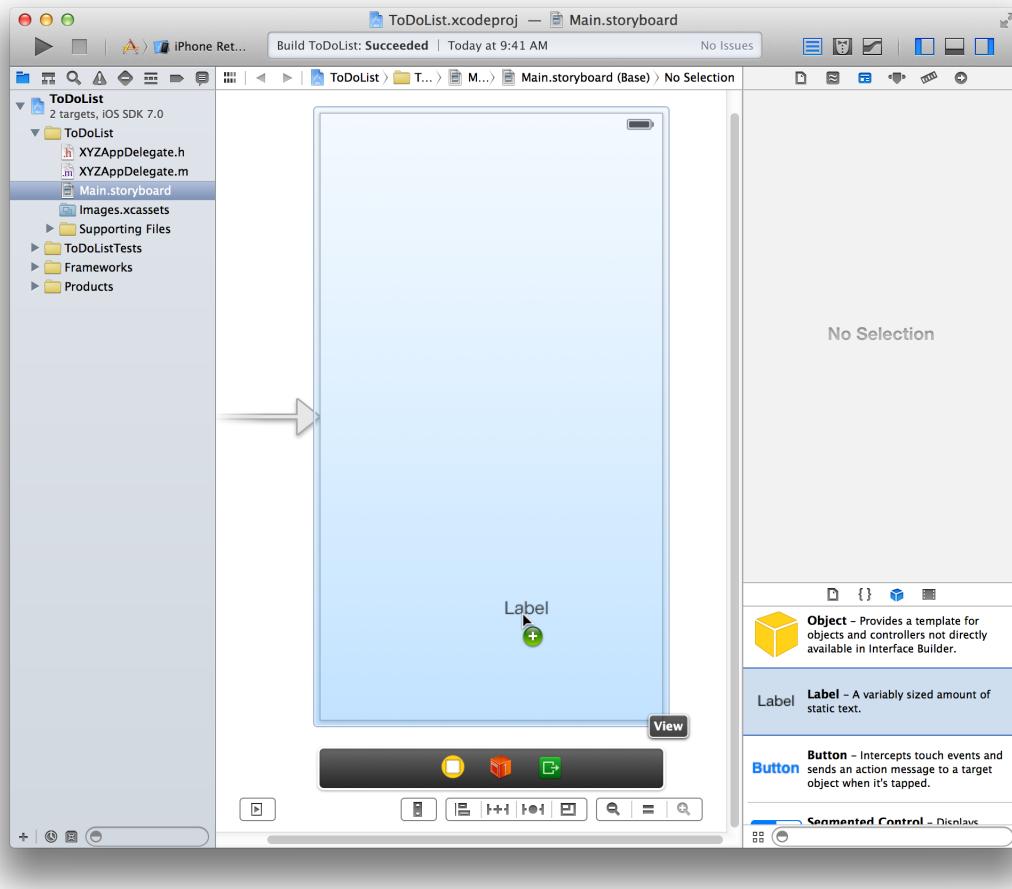
现在，应用程序中的串联图包含了一个场景。画布上指向场景左侧的箭头是“**initial scene indicator**”（初始场景指示器），它表示此场景是应用程序启动时首先载入的场景。现在，您在画布上看到的场景包含了单个视图，由视图控制器管理。虽然是在 iOS Simulator 中运行应用程序，但实际上这也是您将在设备屏幕上看到的视图。在 iOS Simulator 上运行应用程序有助于验证所有配置正确与否。执行该操作前，请在场景中添加一些可以在应用程序运行时看见的内容。

### 将标签添加到场景

#### 1. 在对象库中，找到“Label”对象。

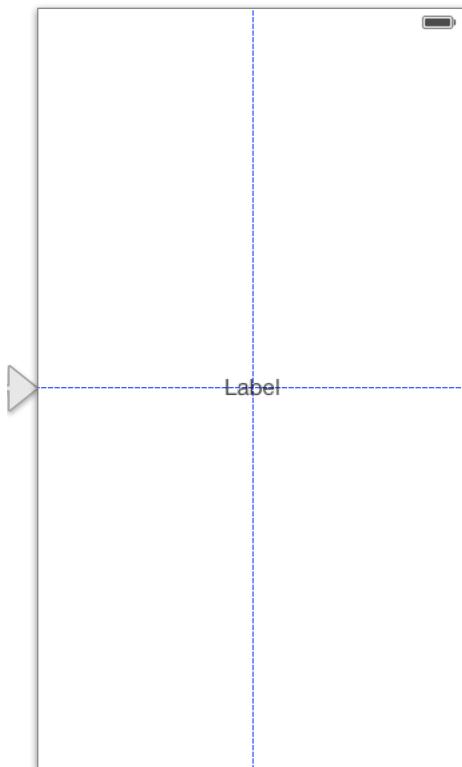
如果曾在过滤文本栏中输入过内容，那么需要先清除原有内容，才能看到“Label”对象。您也可以在过滤栏中键入“Label”来快速查找“Label”对象。

2. 将“Label”对象从列表拖到场景中。



3. 将标签拖到场景的中央，直到出现水平和垂直参考线。

看到以下图标时，停止拖移标签：



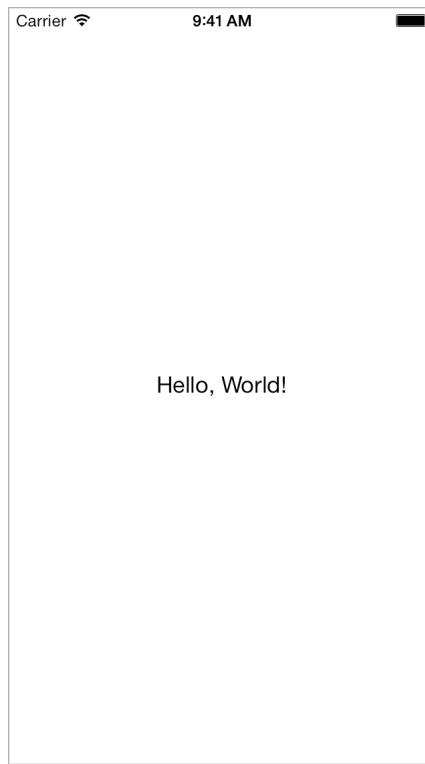
参考线表示目前标签已水平和垂直居中。（只有在参考线旁拖移对象或调整其大小时，参考线才可见；因此当您松开标签时，参考线会消失。）

4. 连接标签的文本，选中并进行编辑。
5. 键入“Hello, World!”并按下 Return 键。

如有需要，请将标签重新居中。

## 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：



如果看不到添加的标签，请确定所创建的串联图已配置为应用程序的主界面，并确定在应用程序委托中用于创建空白色窗口的代码已移除。如有需要，请返回到前面，并重复相关章节中的步骤。

借此机会，您可以试验一下界面可添加的内容。通过更改以下设置来探索 Interface Builder：

- 标签的文本
- 标签的字体
- 文本的颜色

## 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。

### 从场景移除标签

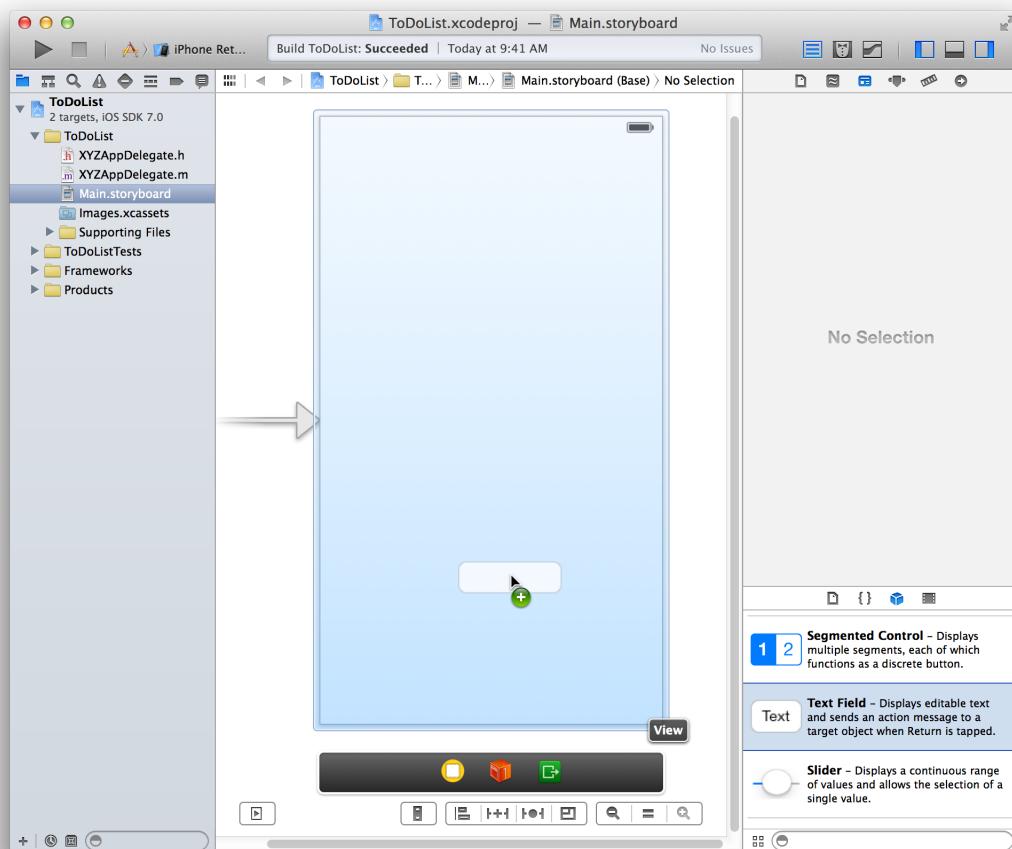
1. 点按标签进行选择。
2. 按下 **Delete** 键。

该标签会从场景中移除。如果操作与期望不符，可以选取“Edit”>“Undo Delete Label”。（每个编辑器都有可撤销上一个操作的“Edit &gt; Undo”命令。）

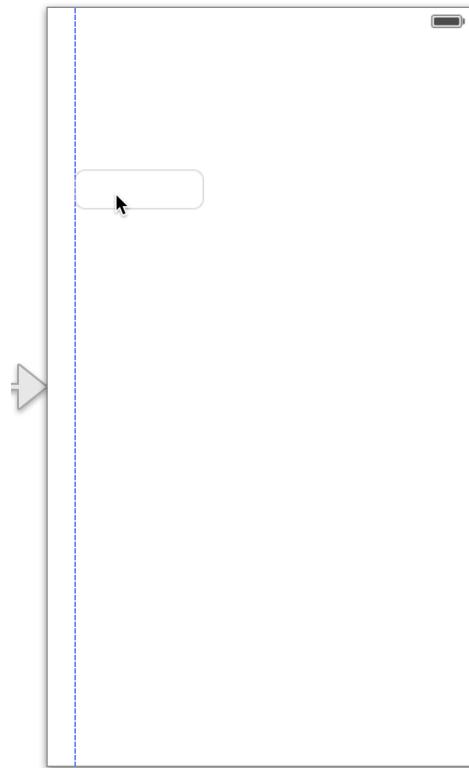
现在画布又重新恢复为空白，可创建用于添加待办事项的场景。

### 将文本栏添加到场景

1. 如有需要，请打开对象库。
2. 将“Text Field”对象从列表拖到场景中。

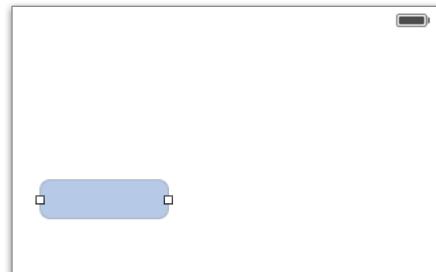


3. 拖移文本栏，然后放置在距屏幕底部三分之二的位置。



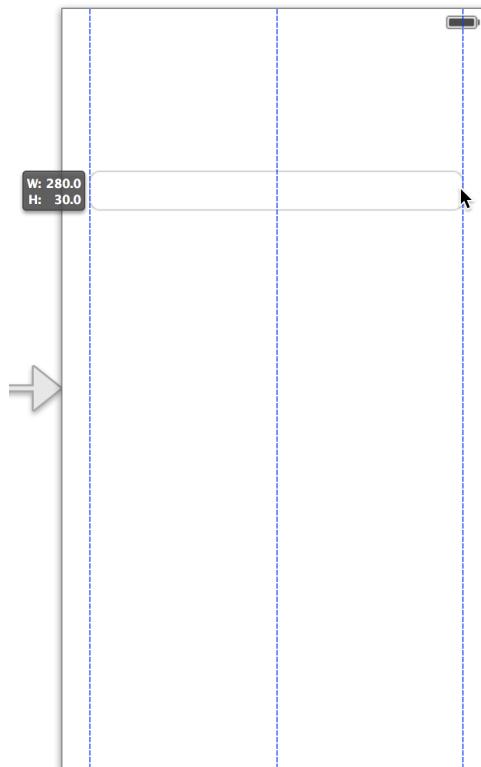
4. 如有必要，请点按文本栏来显示调整大小控制柄。

通过拖移调整大小控制柄（显示在元素边框上的白色小方块）来调整 UI 元素的大小。您可以选择元素来显示其调整大小控制柄。在本例中，因为您刚刚停止拖移，文本栏应该已被选定。如果文本栏外观如下图所示，就可以调整它的大小；否则请在画布上选择它。



5. 调整文本栏的左侧和右侧边缘，直到垂直参考线显示。

当看到画布像下图这样时，停止调整文本栏大小：

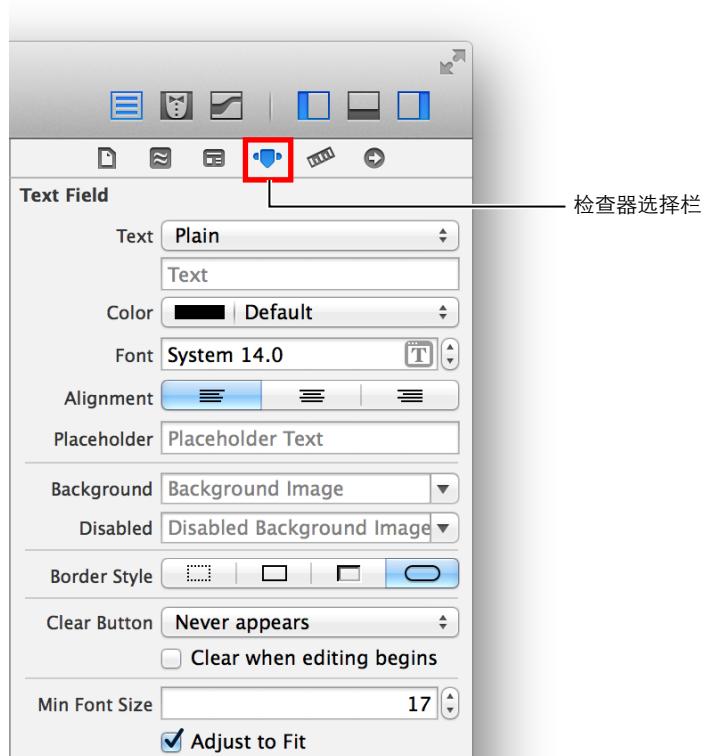


虽然场景中已经有了文本栏，但是尚未告知用户应当在栏中输入什么内容。使用文本栏的占位符文本，提示用户输入新待办事项的名称。

#### 配置文本栏的占位符文本

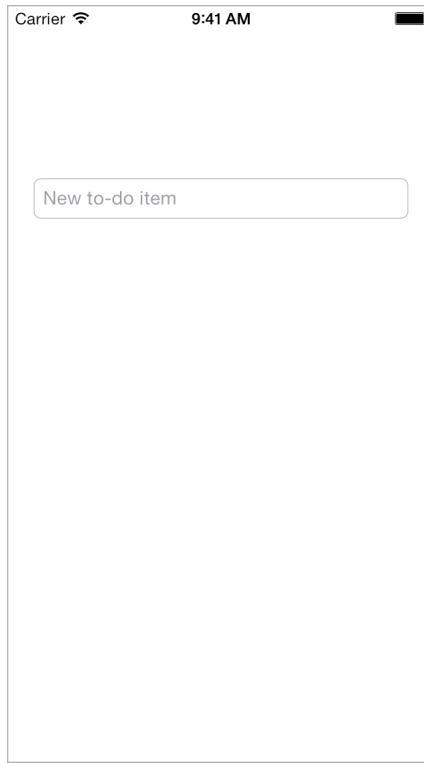
1. 选定文本栏，打开实用工具区域中的“Attributes”检查器 。

选择检查器选择栏中左起第四个按钮时，“**Attributes**”检查器会出现。它可让您编辑串联图中对象的属性。



2. 在“Attributes”检查器中，找到标有“Placeholder”的栏，然后键入“New to-do item”。  
如果要在文本栏中显示新的占位符文本，请按下 Return 键。

检查点：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

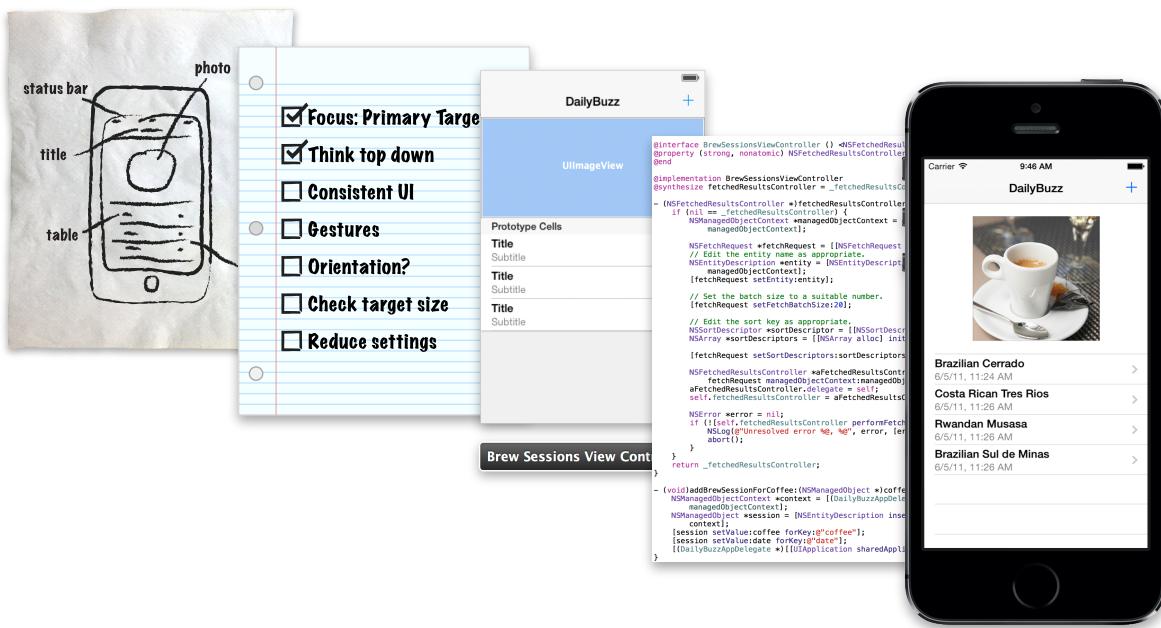
现在，您学会了如何使用串联图来创建基本的界面。在接下来的教程中，会了解到如何给界面添加交互，以及如何编写代码来创建自定行为。教程中的各个章节介绍了一些概念，可让您在处理自己的应用程序时学以致用。

# 构建应用程序

- 应用程序开发过程 (第 30 页)
- 设计用户界面 (第 35 页)
- 定义交互 (第 40 页)
- 教程: 串联图 (第 45 页)

# 应用程序开发过程

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（第 8 页）](#)”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](#)”（第 45 页）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](#)”（第 35 页）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。

## 定义交互

没有逻辑的支持，用户界面的功能便会很有限。创建界面后，可以通过编写代码以响应界面中的用户操作来定义用户与他们所看到内容的交互方式。

在考虑为界面添加行为之前，了解 iOS 应用程序是基于事件驱动编程这一点很重要。在事件驱动编程中，应用程序的流程由事件决定：系统事件或用户操作。用户在界面中执行的操作会触发应用程序中的事件。这些事件会促使执行应用程序的逻辑并处理其数据。应用程序对用户操作的响应则会如实地反映在界面中。

请从事件驱动编程的角度来定义用户与界面的交互方式。因为是用户而非开发者控制着何时执行应用程序的某部分代码。您想要确认用户可以执行哪些操作以及如何响应这些操作。

在视图控制器中，您可以定义大多数的事件处理逻辑。在“[定义交互](#)（第 40 页）”中，会了解有关使用视图控制器的更多知识。之后，在“[教程：串联图](#)（第 45 页）”中，会应用这些概念，为 **ToDoList** 应用程序添加功能和交互性。

## 实现行为

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 语言的词汇会大有裨益。

**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。

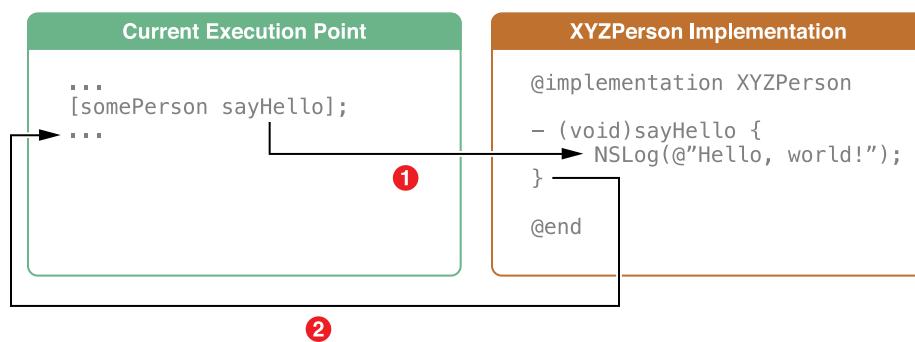
## 对象通过消息通信

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：

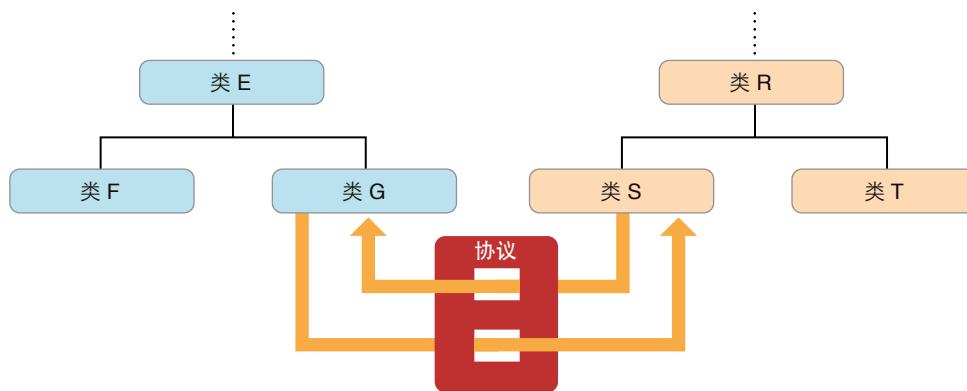
```
[somePerson sayHello];
```

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](#)”（第 66 页）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](#)”（第 68 页）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](#)”（第 71 页）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类来作为数据模型的一部分。通过编写自定类，您可以控制应用程序内部结构的整理方式。在“[写自定类](#)”（第 81 页）中，您会了解有关创建自定类的更多知识。

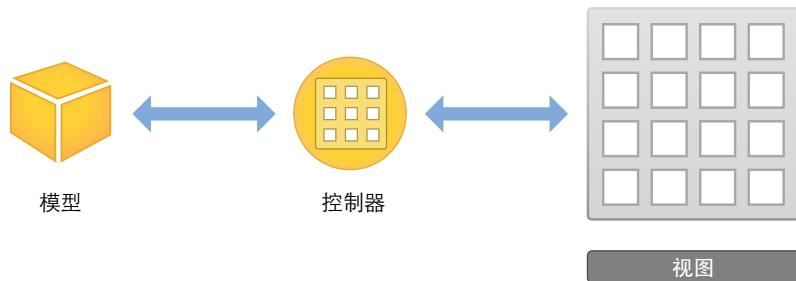
## 整合真实的数据

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。

# 设计用户界面

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，标签
集	显示视图集或视图组。	集视图，表格视图
控制	执行操作或显示信息。	按钮，滑块，开关
栏	导航或执行操作。	工具栏，导航栏，标签栏
输入	接收用户输入的文本。	搜索栏，文本视图
容器	充当其他视图的容器。	视图，滚动视图
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、提醒视图

您可以使用 **Interface Builder**，以图形方式构建视图。**Interface Builder** 提供了一个资源库，其中包含了标准视图、控制，以及构建界面所需要的其他对象。从资源库拖出这些对象之后，您可以将它们放到画布上，并根据需要进行排列。接着可使用检查器配置这些对象，然后再将它们存储到串联图中。您可以立即看到结果，无需编写代码以及生成并运行应用程序。

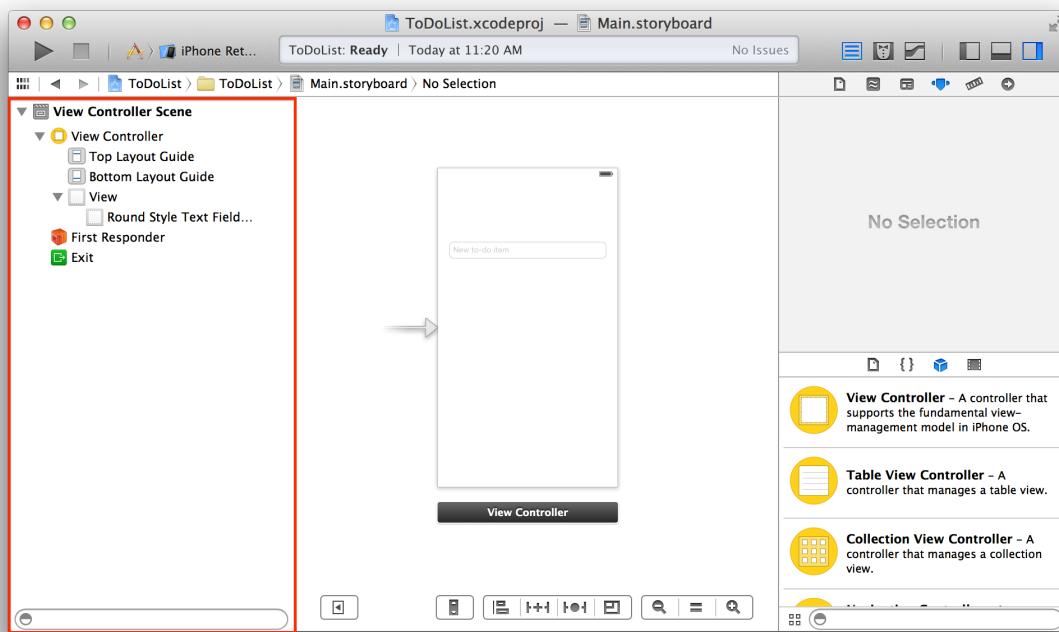
可以使用UIKit框架提供的标准视图来显示各种类型的内容，但也可以自定义视图，方法是子类化UIView（或其后代）。自定视图是UIView的子类，您可以自行在其中处理所有绘图和事件处理任务。在这些教程中，您将不会使用自定视图，但是在“Defining a Custom View”（定义自定视图）中，可以了解到有关实现自定视图的更多知识 in *View Programming Guide for iOS*。

## 使用串联图来布局视图

使用串联图在图形环境中布局视图的层次。您可以使用串联图，以一种直接且直观的方式来处理视图和构建界面。

正如在第一个教程中所学到的，串联图由场景组成，每个场景有关联的视图层次。通过将视图拖出对象库并将其放在串联图场景中，可以自动将它添加到该场景的视图层次。视图在该层次中的位置由您放置的位置决定。将视图添加到场景后，您可以在画布上对其进行大小调整、操控、配置和移动操作。

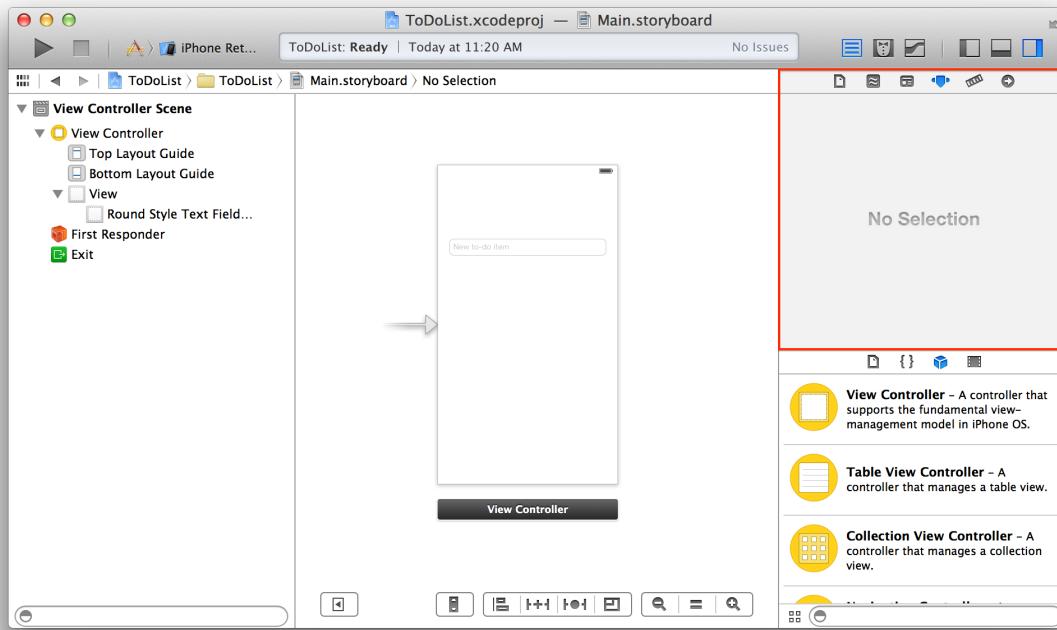
画布还会显示界面中对象的大纲视图。大纲视图显示在画布的左侧，可让您看到对象在串联图中的层次示意。



在串联图场景中以图形方式创建的视图层次实际上就是Objective-C对象的“紧缩”集合。运行时，这些紧缩的对象会被解压缩。运行的结果就是配置了各种属性的相关类（使用实用工具区域中各种检查器以直观方式来设置）的实例层次。

## 使用检查器来配置视图

在串联图中处理视图时，检查器面板是不可或缺的工具。检查器面板显示在对象库上方的实用工具区域中。



每个检查器都提供了配置界面中元素的重要选项。选择串联图中的对象（例如视图）后，可以使用各个检查器来自定义该对象的不同属性。

- **File**。让您指定串联图的常规信息。
- **Quick Help**。提供有关对象的实用文稿。
- **Identity**。让您指定对象的自定类并定义其辅助功能属性。
- **Attributes**。让您自定对象的可视化属性。
- **Size**。让您指定对象的大小以及 Auto Layout 属性。
- **Connections**。让您创建界面和源代码之间的连接。

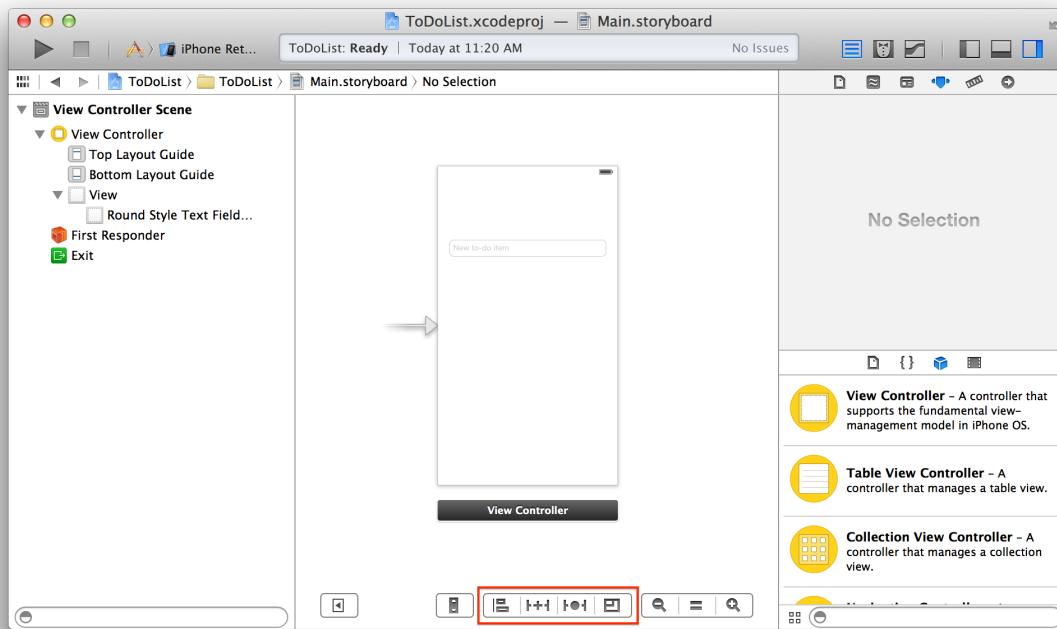
在第一个教程中，您使用了 **Attributes** 检查器。而在接下来的整个教程中，您会继续使用这些检查器来配置串联图中的视图和其他对象。特别需要指出的是，您可以使用 **Attributes** 检查器来配置视图，使用 **Identity** 检查器来配置视图控制器，使用 **Connections** 检查器来创建视图和视图控制器之间的连接。

## 使用 Auto Layout 来定位视图

开始在串联图中定位视图时，您需要考虑各种情况。iOS 应用程序会运行在一系列不同的设备上，这些设备的屏幕大小、方向和语言各不相同。您应该设计一个动态而非静态的界面，且该界面能无缝地响应屏幕尺寸、设备方向、本地化语言以及制式的更改。

为了帮助您使用视图来创建通用的界面，Xcode 提供了一种名为 Auto Layout 的工具。**Auto Layout** 是一种用来表示应用程序的用户界面中各种视图之间关系的体系。它可让您基于单个视图或视图组之间的约束来定义这些关系。

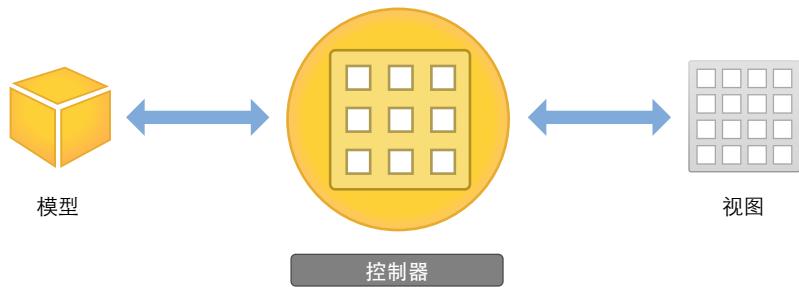
**Auto Layout** 菜单位于画布的右下角，有四个部分。您可以使用该菜单来各种类型的约束添加到画布上的视图中，解决布局问题以及确定约束调整大小行为。



在第二个教程中，您将短暂地使用 Auto Layout 给 ToDoList 应用程序添加支持横排模式的功能。

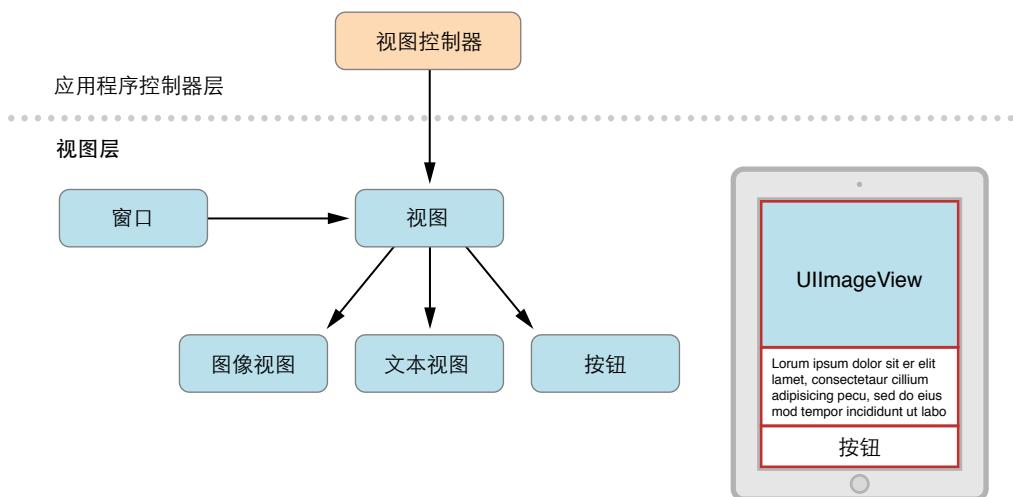
# 定义交互

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 `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 \(第 45 页\)](#)”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（第 45 页）](#)”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。

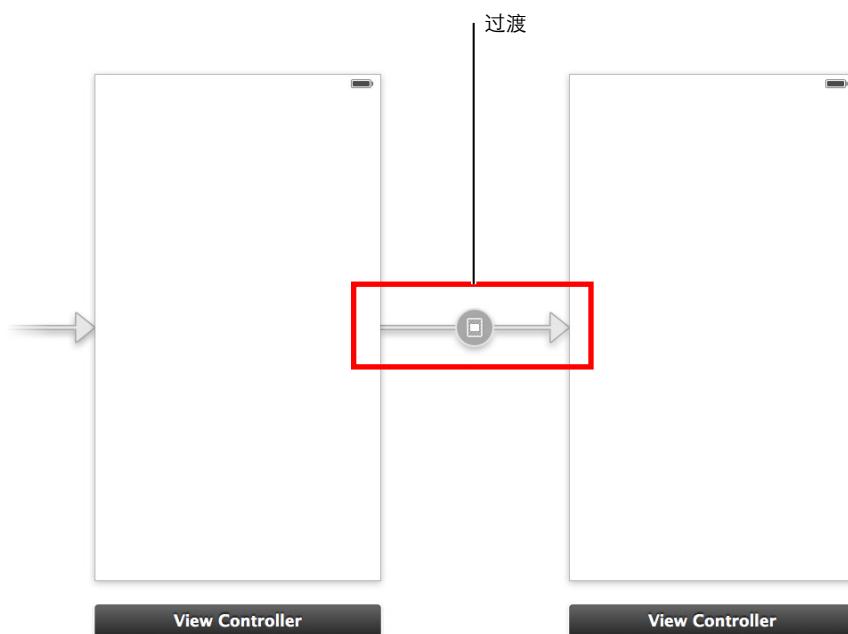
虽然导航控制器的最主要作用是管理内容视图控制器的显示方式，但它还负责显示自己的自定视图。具体来说，它会显示导航栏（位于屏幕顶部的视图，提供有关用户在导航层次中位置的上下文）。导航栏包含一个返回按钮和其他可以自定的按钮。添加到导航栈的每个视图控制器都会显示这个导航栏。您需要配置导航栏。

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。可使用串联图来操作。

## 使用串联图来定义导航

到目前为止，您已经学习了使用串联图来创建包含单个屏幕内容的应用程序。接下来，将要了解如何使用它们来定义应用程序中多个场景之间的流程。

在第一个教程中，所使用的串联图只有一个场景。而在大多数应用程序中，串联图是由一系列的场景组成，每个场景表示一个视图控制器及其视图层次。场景则由过渡连接。过渡表示两个视图控制器之间的切换：源视图控制器和目的视图控制器。



您可以创建以下几种类型的过渡：

- **Push。** Push过渡将目的视图控制器添加到导航栈。只有当源视图控制器与导航控制器连接时，才可以使用Push过渡。

- **Modal。** 简单而言，**Modal** 过渡就是一个视图控制器以模态方式显示另一个控制器，需要用户在显示的控制器上执行某种操作，然后返回到应用程序的主流程。**Modal** 视图控制器不会添加到导航栈；相反，它通常被认为是所显示的视图控制器的子视图控制器。显示的视图控制器的作用是关闭它所创建和显示的 **Modal** 视图控制器。
- **Custom。** 可以通过将 `UIStoryboardSegue` 子类化来定义自定过渡。
- **Unwind。** **Unwind** 过渡通过向后移动一个或多个过渡，让用户返回到视图控制器的当前实例。使用 **Unwind** 过渡可以实现反向导航。

除了过渡之外，还可以通过关系来连接场景。例如，导航控制器与其根视图控制器之间就存在关系。就此而言，这种关系表示导航控制器包含根视图控制器。

使用串联图规划应用程序的用户界面时，要确定将其中一个视图控制器标记为初始视图控制器，这一点尤为重要。运行时，该视图控制器的内容视图会在应用程序首次启动时显示，且在需要时，您可以从该视图控制器切换到其他视图控制器的内容视图。

现在您已经了解了在串联图中处理视图和视图控制器的基础知识。在下一个教程中，我们就要将这些知识运用到 **ToDoList** 应用程序的开发中。

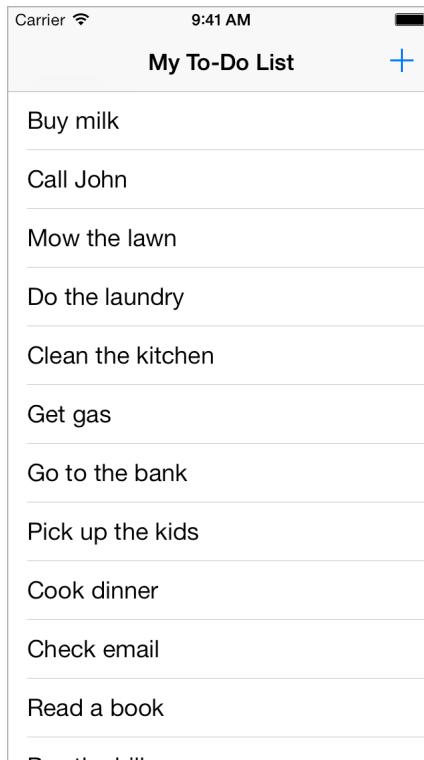
# 教程：串联图

本教程以您在首个教程（“[教程：基础（第 8 页）](#)”）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“Auto Layout”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：

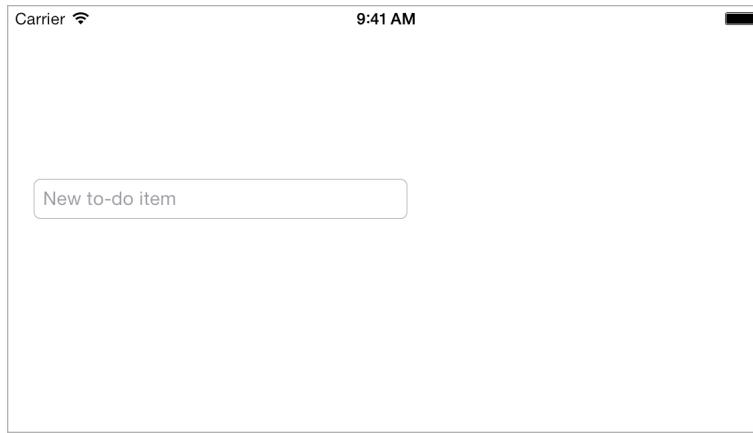


## 采用 Auto Layout

“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）。



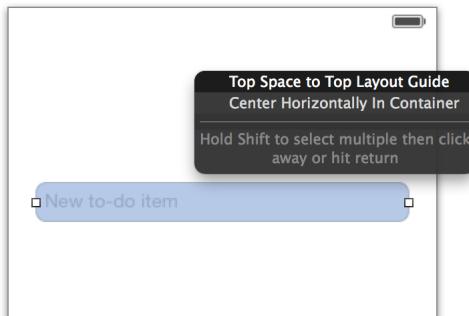
如您所见，文本栏看起来不太对劲。它只占了屏幕约一半的位置。文本栏本应延伸至整个屏幕，正如竖排模式中显示的那样。幸好，Xcode 内建有一款功能强劲的布局引擎，称为“Auto Layout”。关于场景中要如何放置元素，您可以通过 Auto Layout 将您的意图描述出来，然后由该布局引擎确定如何以最优方案实现该意图。使用约束规则描述您的意图，它说明了应当如何放置一个元素以与另一个元素相关联、元素应有的大小，或者在适用的空间减小时，两个元素中的哪一个应当先缩小。对于“add-to-do-item”场景，则需要使用两类约束：一种用于放置文本栏，另一种用于设定其大小。

这些约束的设定工作均可在 Interface Builder 中轻松完成。

使用 **Auto Layout** 放置文本栏

1. 在项目导航器中，选择 Main.storyboard。
2. 在串联图中，选择文本栏。
3. 在画布上，按住 Control 键从文本栏拖向场景顶部，直到文本栏四周皆为空白区域。此区域为文本栏的父视图。

停止拖移操作时，其位置将出现一个快捷菜单。



4. 从快捷菜单中选取“Top Space to Top Layout Guide”。

在文本栏顶部和导航栏之间，将创建一个间距约束。

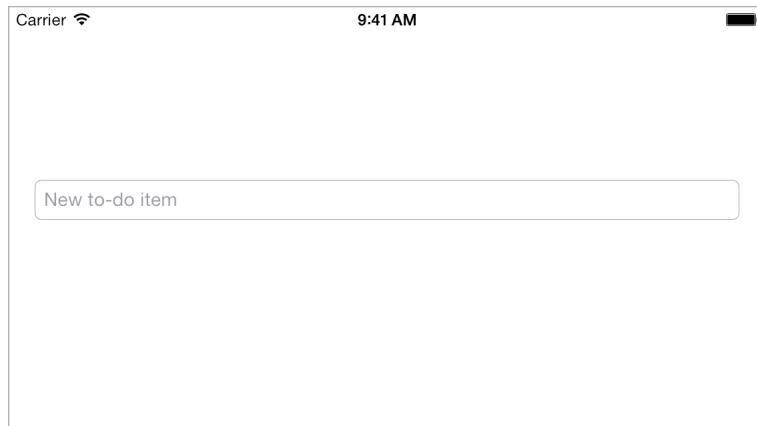
如果出现了其他菜单（比如带有“Leading Space to Container”菜单项），则表示您没有垂直地拖到屏幕顶部。Xcode 会根据您拖移的方向来判断您打算生成哪一类约束，并根据拖移的起始点和结束点来判断约束要关联哪些对象。您可以继续尝试不同的拖移方向，以便了解有哪些约束可用。

5. 尝试完之后，请按住 Control 键从文本栏向右拖移到父视图，以便创建“Trailing Space to Container”约束。
6. 按住 Control 键从文本栏向左拖移到其父视图，以便创建“Leading Space to Container”约束。

这些约束指定了文本栏边缘与其父视图之间的距离，使它们不发生变化。也就是说，如果设备方向发生改变，文本栏将自动伸展以满足这些约束。

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。

如果行为没能达到预期，请启用 Xcode Auto Layout 调试功能以获取帮助。文本栏选定后，请选取“Editor”>“Resolve Auto Layout Issues”>“Reset to Suggested Constraints”，让 Xcode 设置上述步骤所描述的约束。或选取“Editor”>“Resolve Auto Layout Issues”>“Clear Constraints”，移除文本视图上的所有约束，然后试着再次按照上述步骤操作。



尽管添加项目场景能做的有限，但基本的用户界面都已成型并且初具功能。在一开始考虑清楚布局问题，将为您之后的构建奠定坚实基础。

## 创建第二场景

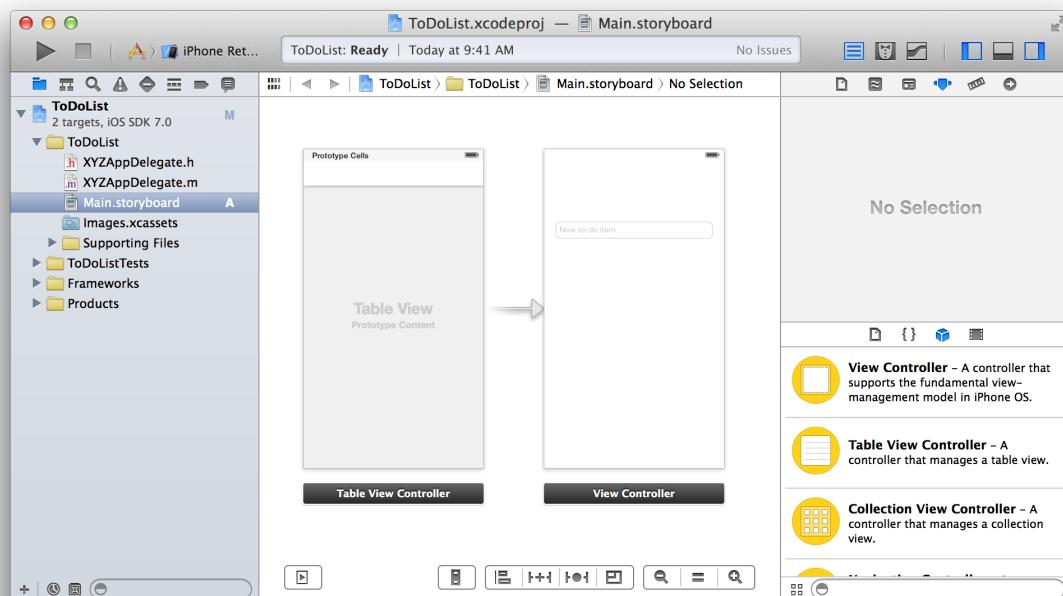
目前，您已经处理过由视图控制器管理的单个场景，即一个可让您将项目添加到待办事项列表的页面。现在，是时候创建一个能显示整个待办事项列表的场景了。真是幸运，iOS自带一个名为表格视图的内建类，它功能强大，专门设计用于显示项目的滚动列表。

将带有表格视图的场景添加到串联图

1. 在项目导航器中，选择 Main.storyboard。
2. 在实用工具区域中打开对象库。（若要通过菜单命令打开资源库，请选取“View”>“Utilities”>“Show Object Library”。）
3. 将“Table View Controller”对象从列表中拖出，并放置在“add-to-do-item”场景左侧的画布上。如有需要，您可以使用画布右下方的“缩小”按钮  来获取足够的拖放空间。

如果表格视图中包含内容，但尝试拖到画布时却毫无反应，那么您拖移的很可能是表格视图，而不是表格视图控制器。表格视图是表格视图控制器所管理的多个项目之一，但您需要的是整个套件，因此请找到表格视图控制器，并将其拖到画布。

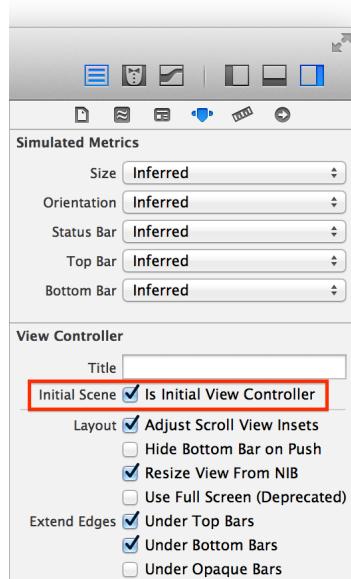
现在您有了两个场景，一个用于显示待办事项列表，另一个用于添加待办事项。



当用户启动应用程序时，让他们先看到列表很有必要。因此请告诉 Xcode 您的意图，把表格视图控制器设置为首个场景。

将表格视图控制器设定为初始场景

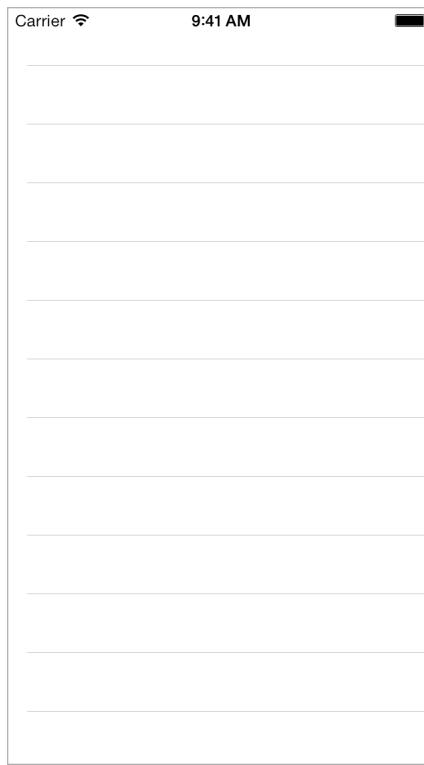
1. 如有需要，请使用画布左下方的按钮来打开大纲视图 。
2. 在大纲视图中，请选择新添加的表格视图控制器。
3. 表格视图控制器选定后，请打开实用工具区域中的“Attributes”检查器 。
4. 在“Attributes”检查器中，选择“Is Initial View Controller”选项旁的复选框。



或者，您可以将初始场景指示器从“add-to-do-item”场景中直接拖到画布上的表格视图控制器。

表格视图控制器将设定为串联图中的初始视图控制器，从而成为应用程序启动时载入的首个场景。

检查点：运行您的应用程序。现在，您看到的不是“add-to-do-item”场景及其文本栏，而应该是空的表格视图：一个由多条水平分隔条划分成行，但每一行均不包含内容的屏幕。



## 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

1. 在界面的大纲视图中，选择“Table View Controller”下方的“Table View”。
2. 表格视图选定后，请打开实用工具区域中的“Attributes”检查器 。
3. 在“Attributes”检查器中，从“Content”选项旁的弹出式菜单中选取“Static Cells”。

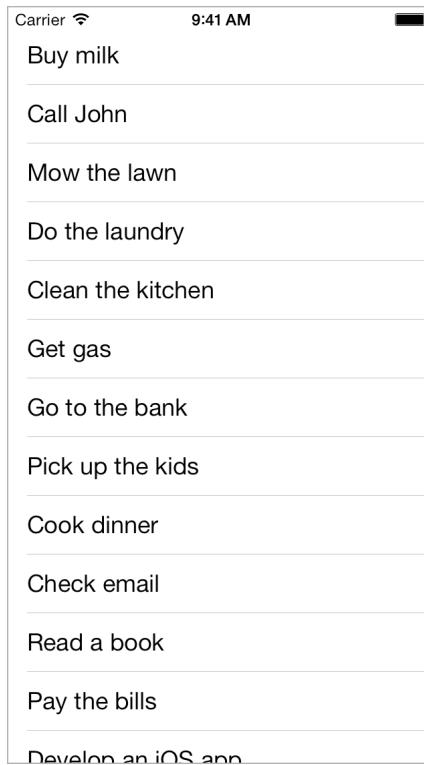
表格视图中将出现三个空的表格视图单元格。

4. 在大纲视图中或画布上，选择顶部单元格。
5. 在“Attributes”检查器中，从“Style”选项旁的弹出式菜单中选取“Basic”。

“Basic”样式包括标签，因此 **Xcode** 将在表格单元格中创建一个带有“Title”文本的标签。

6. 在大纲视图中或画布上，选择该标签。
7. 在“Attributes”检查器中，将标签文本从“Title”更改为“Mow the Lawn”。请按下 **Enter** 键，或点按实用工具区域的外部，使更改生效。  
或者，您可以通过连接两下标签，然后直接编辑文本来编辑标签。
8. 对于其他单元格，请重复步骤 4–7，并根据其他类似的待办事项给单元格安排文本。
9. 请创建足够的单元格，使项目数量超过屏幕的显示范围。您可以通过拷贝和粘贴来创建新单元格，或通过按住 **Option** 键拖移单元格来进行创建。

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，其中含有在 Interface Builder 中添加的预先配置的单元格。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

您已经在串联图中配置了两个视图控制器，但它们彼此之间没有连接。场景之间的转场，我们称为过渡。

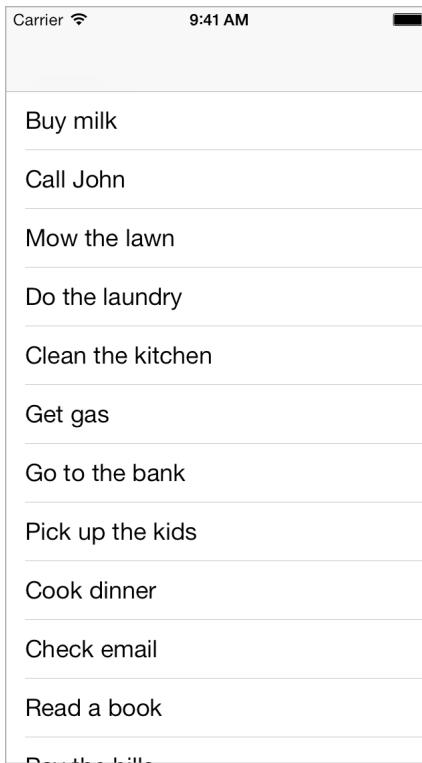
创建过渡前，您需要先配置场景。首先，在导航控制器中包括待办事项列表表格视图控制器。回想之前的“[定义交互（第 40 页）](#)”，我们知道导航控制器能提供导航栏，并且能够跟踪导航堆栈。您将在“add-to-do-item”场景的转场中添加导航栏按钮。

### 将导航控制器添加到表格视图控制器

1. 在大纲视图中，请选择“Table View Controller”。
2. 视图控制器选定后，请选取“Editor”>“Embed In”>“Navigation Controller”。

Xcode 能给串联图添加新的导航控制器、为其设定初始场景，并在新的导航控制器和现有的表格视图控制器之间建立关系。在画布上，如果您选择了连接两个场景的图标，就会发现这其实是一种根视图控制器关系。这表示导航栏下方显示的内容视图将为表格视图。初始场景设定为导航控制器，因为导航控制器保留了将显示在应用程序中的所有内容，它是待办事项列表和“add-to-do-item”场景的容器。

检查点：运行您的应用程序。在表格视图上方，您将看到额外的区域。这是导航控制器所提供的导航栏。



现在，您将为导航栏添加标题（关于待办事项列表）和按钮（用于添加更多待办事项）。

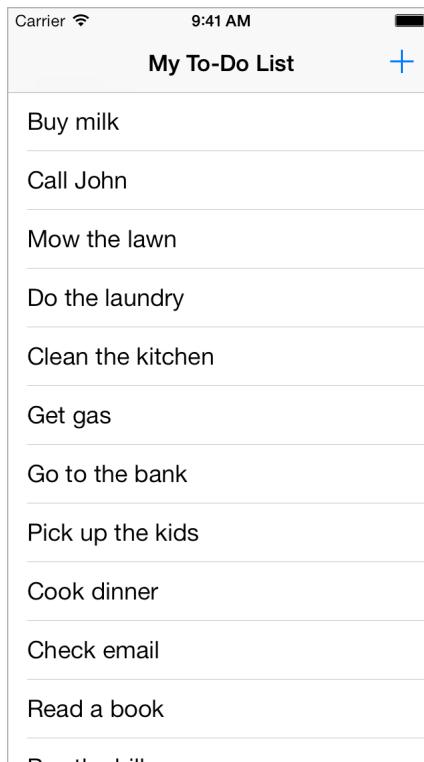
### 配置导航栏

1. 在大纲视图中或画布上，选择“Table View Controller”下方的“Navigation Item”。

导航栏本身并不带标题，它们的标题来自导航控制器当前显示的视图控制器。您需要使用待办事项列表的导航项目（表格视图控制器）来设定标题，而不是直接在导航栏上设定。

2. 在“Attributes”检查器中，在“Title”栏键入“*My To-Do List*”。
  3. 如有需要，请打开对象库。
  4. 在表格视图控制器中，将“Bar Button Item”对象从列表拖到导航栏的最右端。  
在您拖移栏按钮项目的位置，将出现一个包含“Item”文本的按钮。
  5. 在大纲视图或画布上，选择栏按钮项目。
  6. 在“Attributes”检查器中，找到“Bar Button Item”部分中的“Identifier”选项。从“Identifier”弹出式菜单中，选取“Add”。
- 该按钮将变成添加按钮(+)。

检查点：运行您的应用程序。导航栏现在应该有了标题，并显示有一个添加按钮。该按钮暂时没有任何作用。之后您将修复这个问题。

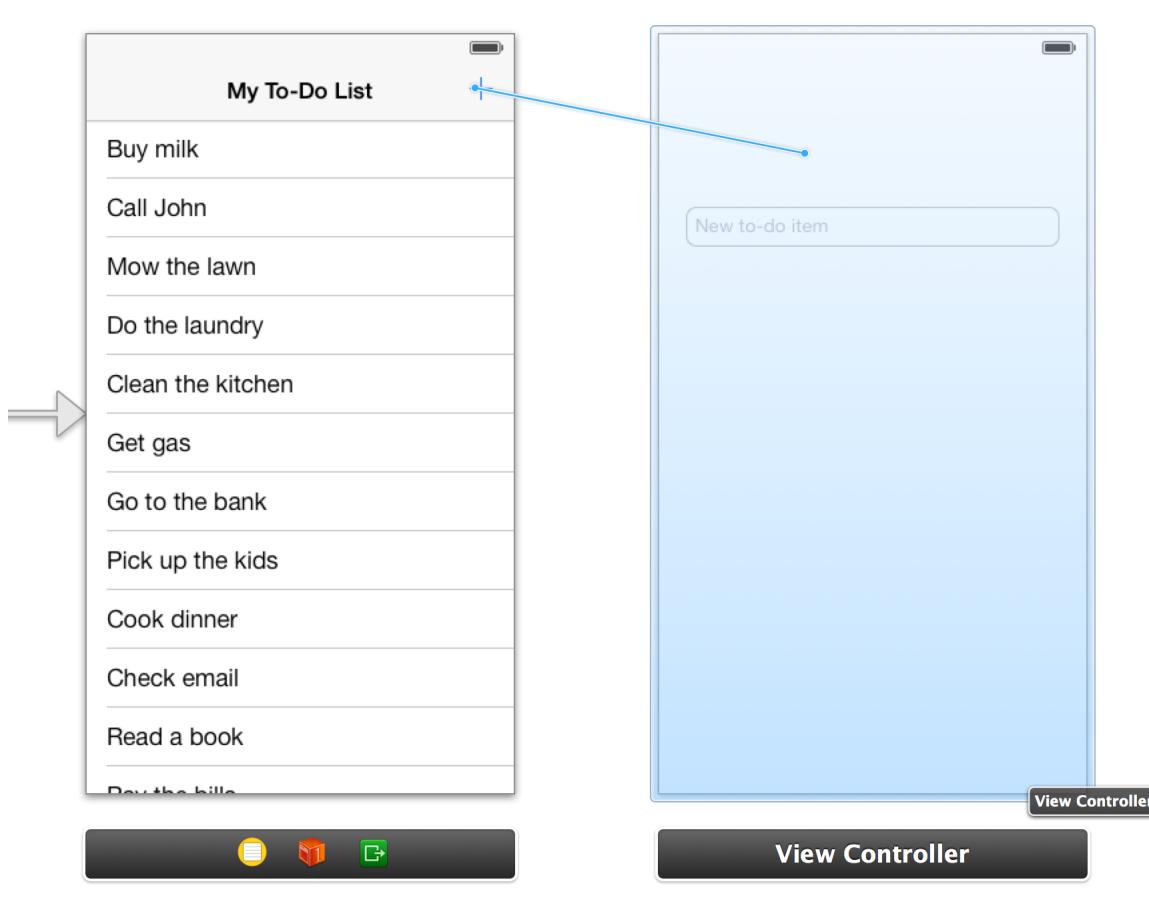


您想让添加按钮调用“add-to-do-item”场景。该场景已经配置好了，它正是您所创建的首个场景，只是还没有连接其他场景。通过 Xcode，您能轻松地配置添加按钮，使其轻按一下便可调出其他场景。

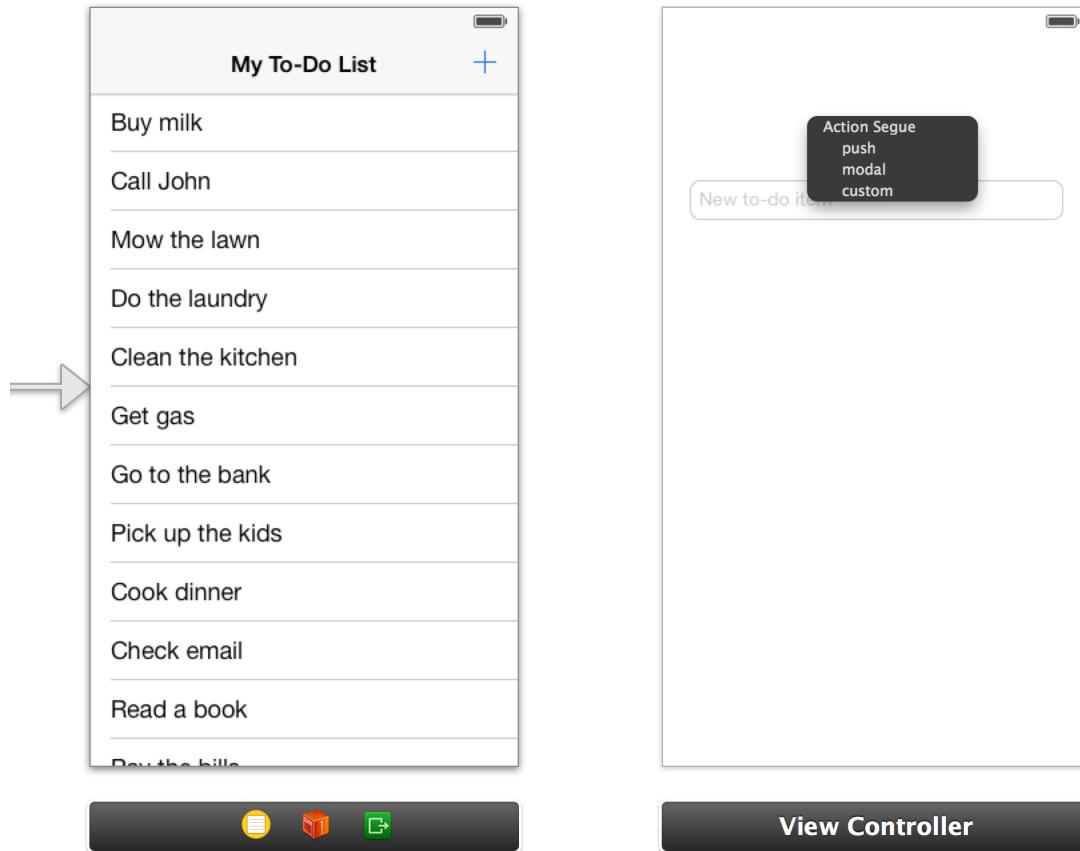
### 配置添加按钮

1. 在画布上，选择添加按钮。

2. 按住 Control 键从按钮拖到“add-to-do-item”视图控制器。



拖移停止的位置，将出现一个标题为“Action Segue”的快捷菜单。

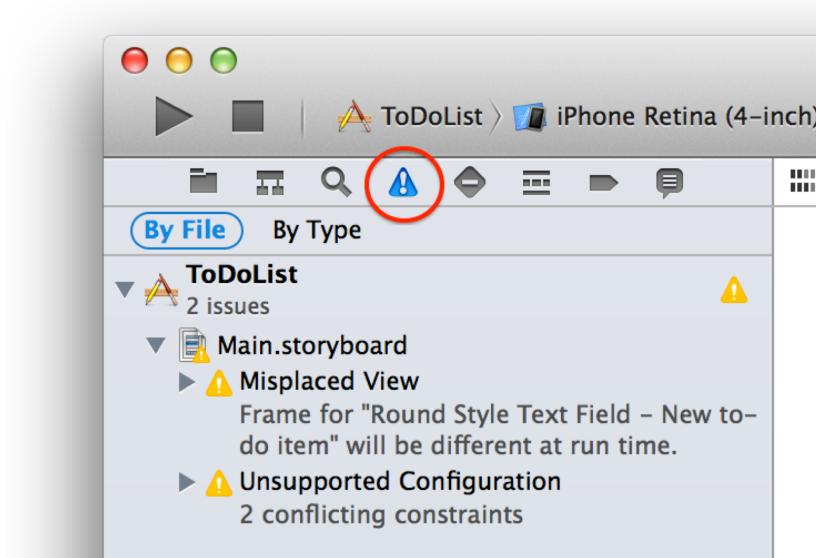


通过该菜单，Xcode能让您选取当用户轻按添加按钮时，待办事项列表到“add-to-do-item”视图控制器之间的转场所使用的过渡类型。

### 3. 从快捷菜单中选取“push”。

Xcode将设置过渡，并将“add-to-do-item”视图控制器配置为显示在导航控制器中，您将在Interface Builder中看到导航栏。

现在，项目中可能出现了数个警告。请继续并打开“Issue”导航器来查看问题所在。



由于您将“add-to-do-item”场景添加到了导航堆栈，它现在将显示导航栏。该栏将导致您文本栏的边框下移，从而无法满足之前指定的 Auto Layout 约束。好在这并不难解决。

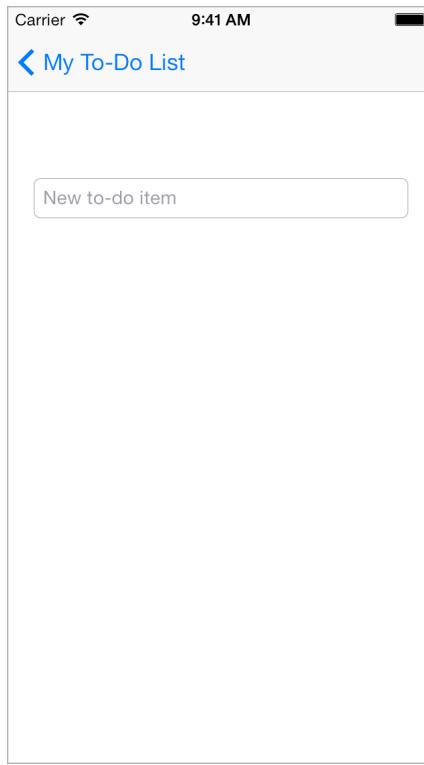
#### 更新 Auto Layout 约束

1. 在大纲视图或画布上，选择文本栏。
2. 在画布上，打开“Resolve Auto Layout Issues”弹出式菜单 ⓘ，然后选取“Update Constraints”。

或者，您可以选取“Editor”>“Resolve Auto Layout Issues”>“Update Constraints”。

约束将更新，同时 Xcode 警告会消失。

检查点：运行您的应用程序。您可以点按添加按钮，并浏览到表格视图中的“add-to-do-item”视图控制器。由于您使用的是带 **push** 过渡的导航控制器，因此向后导航已为您处理完毕。这表示您可以通过点按返回按钮来回到表格视图。



一般情况下，**push** 导航的功能是正常的，但添加项目时则不太一样。**Push** 导航设计用于深层次界面，即无论用户选定哪一项，您都能为其提供更多相关信息。而另一方面，添加项目是一种模态操作，即用户执行某个完整且自包含的操作，然后从场景返回到主导航。此类型场景恰当的表述方式为 **modal** 过渡。

#### 更改过渡样式

1. 在大纲视图中或画布上，选择表格视图控制器到“add-to-do-item”视图控制器的过渡。
2. 在“Attributes”检查器中，从“Style”选项旁的弹出式菜单中选取“Modal”。

**Modal** 视图控制器不会添加到导航堆栈，因此它没有从表格视图控制器的导航控制器中获得导航栏。但是您可能想要保留导航栏，给予用户视觉上的连续性。要在模态展示时为“add-to-do-item”视图控制器添加导航栏，请将它嵌入其自身的导航控制器中。

#### 将导航控制器添加到“add-to-do-item”视图控制器

1. 在大纲视图中，请选择“View Controller”。
2. 视图控制器选定后，请选取“Editor”>“Embed In”>“Navigation Controller”。

和之前一样，Xcode会在视图控制器的顶部添加导航控制器，并显示导航栏。接着，配置此栏以将标题及“Cancel”和“Done”这两个按钮添加到该场景。稍后，会将这些按钮链接到操作。

#### 配置“add-to-do-item”视图控制器中的导航栏

1. 在大纲视图中或画布上，选择“View Controller”下方的“Navigation Item”。如有需要，请打开“Attributes”检查器 。

2. 在“Attributes”检查器中，在“Title”栏键入“Add To-Do Item”。

Xcode 会将视图控制器的描述从“View Controller”更改为“View Controller – Add To-Do Item”，方便您辨别场景。描述将出现在大纲视图中。

3. 在“add-to-do-item”视图控制器中，将“Bar Button Item”对象从对象库拖到导航栏的最右端。

4. 在“Attributes”检查器中，从“Identifier”选项旁的弹出式菜单中选取“Done”。

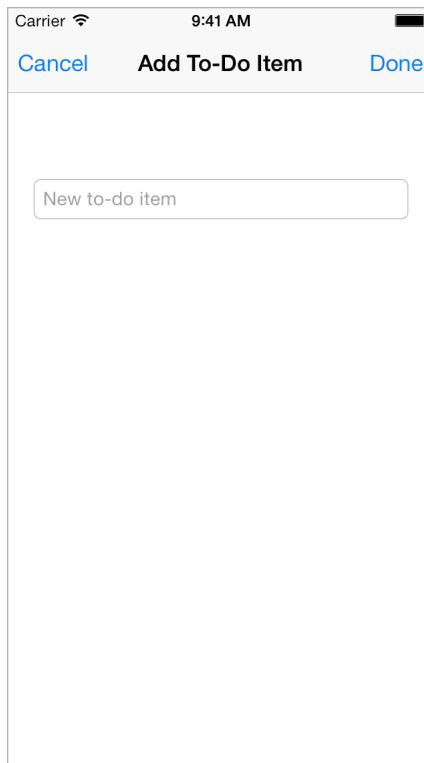
按钮文本将更改为“Done”。

5. 在“add-to-do-item”视图控制器中，将另一个“Bar Button Item”对象从对象库拖到导航栏的最左端。

6. 在“Attributes”检查器中，从“Identifier”选项旁的弹出式菜单中选取“Cancel”。

按钮文本将更改为“Cancel”。

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。



## 创建自定视图控制器

至今您所完成的配置工作，都无需编写任何代码。不过，配置整个“add-to-do-item”视图控制器还是需要代码的，同时还得有放代码的位置。此时，Xcode 已将“add-to-do-item”视图控制器，以及表格视图控制器，配置为基本的视图控制器。如果给自定代码预留位置，您需要为每一个视图控制器创建子类，然后配置界面以使用这些子类。

首先，需要处理“add-to-do-item”视图控制器场景。自定视图控制器类的名称为 XYZAddToDoItemViewController，因为此视图控制器所控制的场景，会为您的待办事项列表添加新项目。

### 创建 **UIViewController** 的子类

1. 请选取“File”>“New”>“File”（或按下 Command-N）。
2. 在出现的对话框左侧，选择“iOS”下方的“Cocoa Touch”模板。
3. 选择“Objective-C Class”，然后点按“Next”。

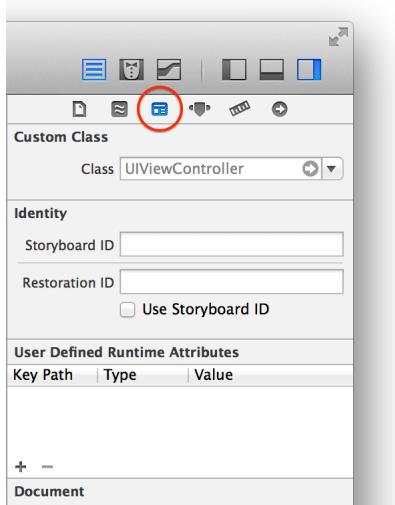
4. 在“Class”栏中，在 XYZ 前缀之后键入 AddToDoItem。
5. 选取“Subclass of”弹出式菜单中的 UIViewController。  
类标题将变更为“XYZAddToDoItemViewController”。创建自定视图控制器时，Xcode 能帮您清楚区分出该名称。好的，那这个新名称可以不用管了。
6. 请确定“Targeted for iPad”和“With XIB for user interface”选项并未选定。
7. 点按“Next”。
8. 存储位置默认为您的项目目录。保持该项不变。
9. “Group”选项默认为您的应用程序名称，“ToDoList”。保持该项不变。
10. 在“Targets”部分中，应用程序默认为选定，而应用程序的测试默认为未选定。很好，那这一项也保持不变。
11. 点按“Create”。

现在，自定视图控制器的子类已经创建，您需要让串联图使用自定类，而不是使用基本视图控制器。串联图文件是应用程序运行时所使用的对象的相关配置。应用程序的运行机理，能将串联图最初使用的基  
本视图控制器，智能地替换成自定视图控制器，但首先您需要向串联图下达该命令。

#### 将类识别为某个场景的视图控制器

1. 在项目导航器中，请选择 Main.storyboard。
2. 如有需要，请打开大纲视图 。
3. 在大纲视图中，选择“View Controller – Add To-Do Item”视图控制器。  
点按“View Controller – Add To-Do Item”场景旁的显示三角形来显示场景中的对象。第一个对象应该是视图控制器。点按以选择它。注意场景行的图标与视图控制器行的图标并不相同。
4. 选定视图控制器后，请打开实用工具区域中的“Identity”检查器 .

当您点按左起第三个按钮时，“**Identity**”检查器将出现在实用工具区域的顶部。它能让您编辑串联图中对象的属性（与对象的识别信息相关），例如它是什么类。



5. 在“Identity”检查器中，打开“Class”选项旁的弹出式菜单。

您将看到一个列表，其中包含了所有 Xcode 了解的视图控制器类。列表的最后一个应该是自定视图控制器 XYZAddToDoItemViewController。选取它，从而让 Xcode 为此场景使用视图控制器。

运行时，串联图将创建 XYZAddToDoItemViewController 的实例，即您的自定视图控制器子类来代替基本的 UIViewController。请注意 Xcode 已将“add-to-do-item”视图控制器场景的描述，从“View Controller – Add To-Do Item”更改为“Add To Do Item View Controller – Add To-Do Item”。Xcode 知道此场景当前正在使用自定视图控制器，因此它解释了自定类的名称，以方便您理解串联图上的内容。

现在，请对表格视图控制器执行相同操作。

#### 创建 **UITableViewController** 的子类

1. 请选取“File”>“New”>“File”（或按下 Command-N）。
  2. 在左侧选择“iOS”下方的“Cocoa Touch”，然后选择“Objective-C Class”。如果在本教程前面的步骤中您尚未创建任何类，那么 Xcode 可能已经为您选好了一个。
  3. 点按“Next”。
  4. 在“Class”栏中，键入“ToDoList”。请注意 Xcode 将插入点放置在 XYZ（类前缀）和 ViewController（所创建的项目类型）之间。
  5. 从“Subclass of”弹出式菜单中选取 UITableViewController。
  6. 请确定“Targeted for iPad”和“With XIB for user interface”选项并未选定。
  7. 点按“Next”。
- 存储位置默认为您的项目目录。保持该项不变。
8. “Group”选项默认为您的应用程序名称，“ToDoList”。保持该项不变。

9. 在“Targets”部分中，应用程序默认为选定，而应用程序的测试默认为未选定。很好，那这一项也保持不变。
10. 点按“Create”。

请再次确定，已在串联图中配置了自定表格视图控制器 XYZToDoListViewController。

#### 配置串联图

1. 在项目导航器中，选择 Main.storyboard。
2. 如有需要，请打开大纲视图。
3. 在大纲视图中，选择表格视图控制器，然后打开实用工具区域中的“Identity”检查器 。
4. 在“Identity”检查器中，从“Class”选项旁的弹出式菜单中选取 XYZToDoListViewController。

现在，您可以给视图控制器添加自定义代码了。

## 跳转过渡以返回

除了 push 和 modal 过渡外，Xcode 还提供 unwind 过渡。此过渡允许用户从一个给定场景返回上一个场景，并提供位置让您添加代码，以便在用户导航切换场景时执行。您可以使用 unwind 过渡从 XYZAddToDoItemViewController 返回浏览 XYZToDoListViewController。

通过向目的视图控制器（要跳转到的视图控制器）添加操作方法，可以创建 unwind 过渡。可以跳转到的方法必须返回一个操作 (IBAction)，并且在串联图过渡 (UIStoryboardSegue) 中作为参数采用。您可能想跳转回 XYZToDoListViewController，因此需要将带有该格式的操作方法添加到 XYZToDoListViewController 界面并进行实现。

#### 跳转回 XYZToDoListViewController

1. 在项目检查器中，打开 XYZToDoListViewController.h。
2. 在 @interface 行下方添加以下代码：

```
- (IBAction)unwindToList:(UIStoryboardSegue *)segue;
```

3. 在项目检查器中，打开 XYZToDoListViewController.m。
4. 在 @implementation 行下方添加以下代码：

```
- (IBAction)unwindToList:(UIStoryboardSegue *)segue
```

```
{  
}  
}
```

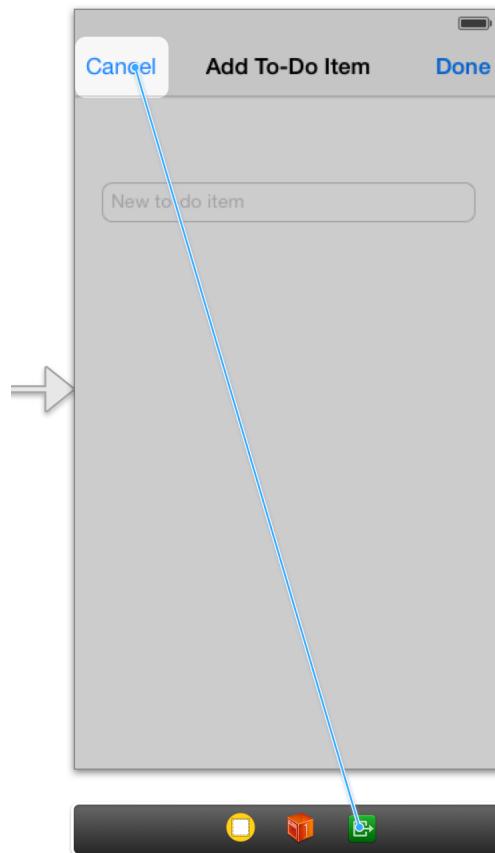
您可以给跳转操作随意命名。可以将它命名为 `unwindToList:`，让跳转操作返回的位置一目了然。对于今后的项目，可以采用类似的命名惯例，即操作的名称能清楚表示跳转操作将返回的位置。

当前项目中，请将此方法实现留空。稍后，您将使用此方法从 `XYZAddToDoItemViewController` 中取回数据，以将项目添加到待办事项列表。

若要创建 `unwind` 过渡，请通过源视图控制器 `XYZAddToDoItemViewController` 场景台中的“Exit”图标，将“Cancel”和“Done”按钮链接到 `unwindToList:` 操作。

#### 将按钮链接到“`unwindToList:`”操作

1. 在项目导航器中，选择 `Main.storyboard`。
2. 在画布上，按住 Control 键从“Cancel”按钮拖到“add-to-do-item”场景台中的“Exit”项。



如果您在场景台中看到的是场景的描述，而不是“Exit”项，请点按画布上的“放大”按钮，直到看到该项为止。

一个菜单将出现在拖移停止的位置。

3. 从快捷菜单中选取 `unwindToList:`。

这是您刚才添加到 `XYZToDoListViewController.m` 文件的操作。表示当轻按“Cancel”按钮时，将跳转过渡并调用此方法。

4. 在画布上，按住 **Control** 键从“Done”按钮拖到 `XYZAddToDoItemViewController` 场景台中的“Exit”项。

5. 从快捷菜单中选取 `unwindToList:`。

请注意，“Cancel”和“Done”按钮使用了相同的操作。在下个教程中，您将在编写处理 `unwind` 过渡的代码时区分这两种情况。

检查点：现在，请运行应用程序。启动时，您将看到表格视图，但其中不含数据。您可以点按添加按钮，从 `XYZToDoListViewController` 浏览到 `XYZAddToDoItemViewController`。也可以点按“Cancel”和“Done”按钮返回浏览表格视图。

那为什么显示不了数据呢？表格视图有两种获取数据的方式，即静态获取或动态获取。当表格视图的控制器实现所要求的 `UITableViewDataSource` 方法时，表格视图将向其视图控制器请求数据以进行显示，不管 **Interface Builder** 中是否配置了静态数据。如果您留意下 `XYZToDoListViewController.m`，就会发现它实现了三种方法：`numberOfSectionsInTableView:`，`tableView:numberOfRowsInSection:` 以及 `tableView:cellForRowAtIndexPath:`。您可以通过注释掉这些方法的实现，在表格视图中再次显示静态数据。如果感兴趣，您可以继续试试。

## 小结

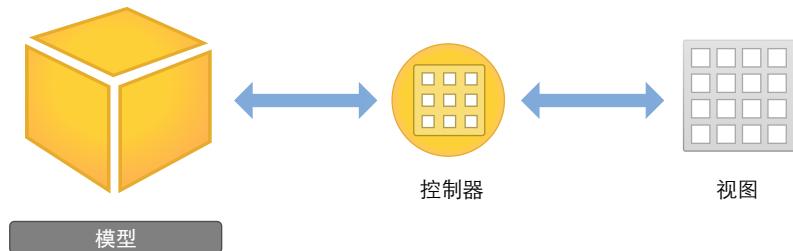
现在，您已经完成了应用程序界面的开发工作。现在有两个场景：一个用于给待办事项列表添加项目，另一个用于查看该列表；并且您可以在两个场景之间浏览。接下来要实现的功能，是让用户添加新的待办事项，并使其显示在列表中。下一部分讲述了如何处理数据以实现该行为。

# 应用程序的实现

- 整合数据（第 66 页）
- 使用设计模式（第 68 页）
- 处理 Foundation（第 71 页）
- 编写自定类（第 81 页）
- 教程：添加数据（第 87 页）

# 整合数据

应用程序的数据模型由数据结构和（可选）自定业务逻辑组成；自定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](#)（第 71 页）”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，还要求自定业务逻辑，那么您可以编写一个自定类。您应考虑如何将现有框架类合并到您自己的类的实现中。在自定类中使用现有框架类往往比重写更省时省力。例如，自定类可能使用 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。

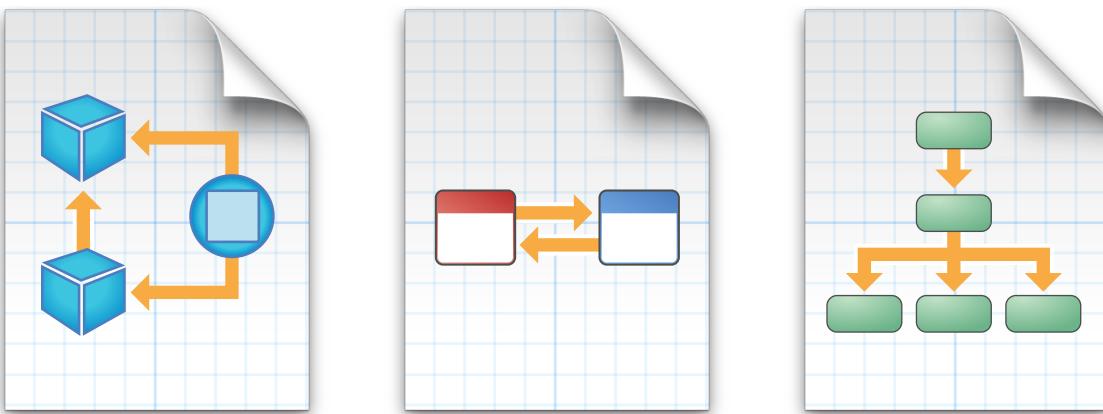
学习 Objective-C 的好方法有很多种。有的人通过阅读《*Programming with Objective-C*》（使用 Objective-C 编程）来了解其概念，然后编写一些小的测试应用程序来巩固对该语言的理解，并练习编写更好的代码。

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。

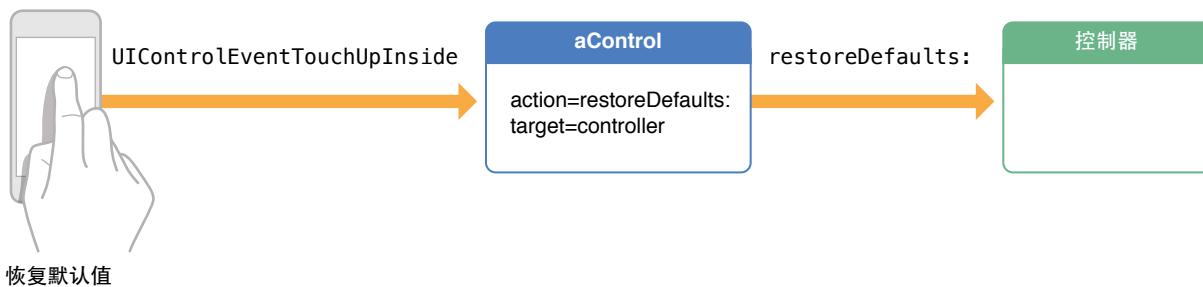


在构建 **ToDoList** 应用程序的过程中，您遵循的就是以 **MVC** 为中心的设计模式。在串联图中构建的界面组成了视图层 `XYZAddToDoItemViewController` 和 `XYZToDoListViewController` 就是管理视图的控制器。在“[教程：添加数据](#)（第 87 页）”中，您需要结合数据模型来处理应用程序中的视图和控制器。在开始设计自己的应用程序时，很重要的一点就是以 **MVC** 为中心进行设计。

## 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。

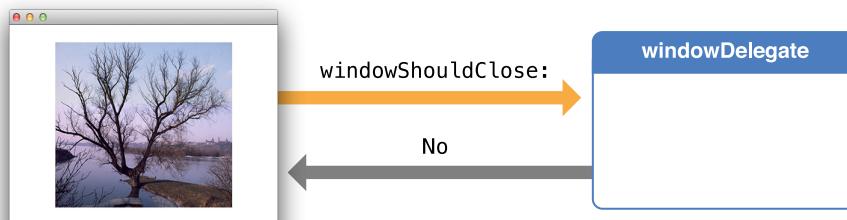
例如，假设您想要在每次用户轻按“恢复默认”按钮（在用户界面中创建）时恢复应用程序中的默认设置。首先，您需要实施操作 `restoreDefaults:` 来执行恢复默认设置的逻辑。接着，注册按钮的 `Touch Up Inside` 事件，将 `restoreDefaults:` 操作方法发送到实施该方法的视图控制器。



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。



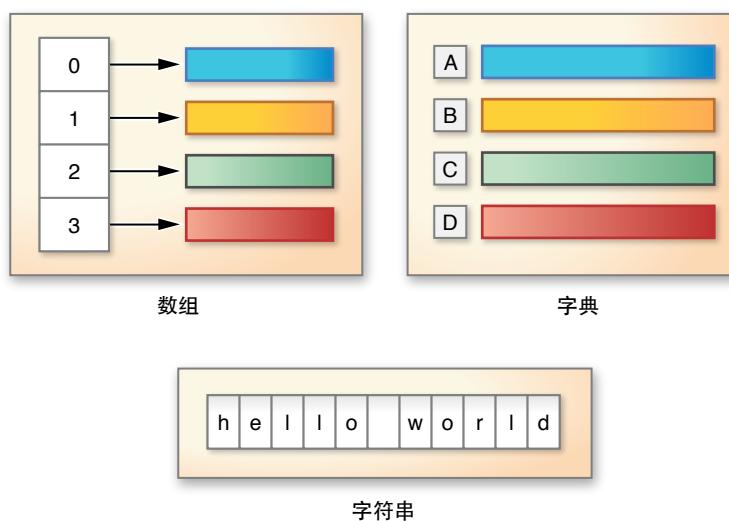
委托模式不仅普遍用于既有的框架类，而且也可应用在应用程序的两个自定对象之间。常见的设计是将委托作为一种手段，允许子视图控制器将某些值（通常为用户输入的值）传达到父视图控制器。

目前您还没有使用过委托，但是在“[教程：添加数据](#)（第 87 页）”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。

# 处理 Foundation

开始编写应用程序的代码时，可以利用大量的 Objective-C 框架。其中，为所有应用程序提供基本服务的 **Foundation** 框架尤为重要。**Foundation** 框架包括表示基本数据类型的值类（如字符串和数字）以及用于储存其他对象的集（**collection**）类。**ToDoList** 应用程序中的大量代码都可以依靠值类和集类来编写。



## 值对象

Foundation 框架提供了为字符串、二进制数据、日期与时间、数字以及其他值产生值对象的类。

值对象是指封装了基本值（属于C数据类型）且提供与该值相关的服务的对象。您会频繁遇到值对象，作为应用程序调用的方法和函数的参数和返回值。框架的不同部分，甚至不同的框架都可以通过传递值对象来交换数据。

以下是 Foundation 框架中值对象的几个示例：

NSString 和 NSMutableString

NSData 和 NSMutableData

NSDate

NSNumber

NSValue

由于值对象表示标量值，因此您可以在集(**collection**)中使用，也可以在任何需要对象的地方使用。基于值对象所封装的基本类型，它们有一项天然优势：可让您采用简单而高效的方式对封装的值执行某些操作。例如，**NSString** 类具有用于搜索和替换子字符串、将字符串写入文件或（首选）**URL** 以及构建文件系统路径的方法。

您可以从基本类型的数据创建值对象（然后在方法参数中传递它）。稍后，您可通过代码从该对象访问被封装的数据。**NSNumber** 类是这一方法最清晰的示例。

```
int n = 5; // Value assigned to primitive type
NSNumber *numberObject = [NSNumber numberWithInt:n]; // Value object created from
// primitive type
int y = [numberObject intValue]; // Encapsulated value obtained from value object
(y == n)
```

大多数值类会通过声明初始化程序和类工厂方法来创建其实例。类工厂方法由类实施，作为提供给客户的简单方法；它将分配和初始化结合为一个步骤，并返回已创建的对象。例如，**NSString** 类可声明 **string** 类方法，以便分配和初始化类的新实例，并将其返回到代码中。

```
NSString *string = [NSString string];
```

除创建值对象和让您访问其封装值之外，大多数值类都提供用于简单操作（如对象比较）的方法。

## 字符串

**Objective-C** 指定字符串的约定与 C 相同：单个字符会使用单引号括起来，而字符串则使用双引号括起来。但是，**Objective-C** 框架通常不使用 C 字符串。相反，它们会使用 **NSString** 对象。

**NSString** 类为字符串提供了一个对象包装器，它具有诸多优势，如内置了可用于储存任意长度字符串的内存管理、提供了对各种字符编码（特别是**Unicode**）的支持，以及用于格式化字符串的实用工具等。因为您通常会使用此类字符串，所以 **Objective-C** 提供了速写记法，即根据常量值来创建 **NSString** 对象。要使用此 **NSString** 字面常量，只需在双引号字符串前面添加 @ 符号，如下例所示：

```
// Create the string "My String" plus carriage return.
NSString *myString = @"My String\n";
// Create the formatted string "1 String".
NSString *anotherString = [NSString stringWithFormat:@"%@", 1, @"String"];
// Create an Objective-C string from a C string.
NSString *fromCString = [NSString stringWithCString:@"A C string"
encoding:NSUTF8StringEncoding];
```

## 数字

Objective-C 提供了创建 `NSNumber` 对象的速写记法，从而无需调用初始化程序或类工厂方法就可以创建此类对象。只需在数值前面添加 @ 符号，并选择一个添加在其后面的值类型指示。例如，创建封装整数值和双精度值的 `NSNumber` 对象，可以编写如下代码：

```
NSNumber *myIntValue = @32;
NSNumber *myDoubleValue = @3.22346432;
```

您甚至可以使用 `NSNumber` 字面常量来创建封装的 Boolean 值和字符值。

```
NSNumber *myBoolValue = @YES;
NSNumber *myCharValue = @'V';
```

可以创建 `NSNumber` 对象，表示无符号整型 (`unsigned integers`)、长整型 (`long integers`)、长长整型 (`long long integers`) 和浮点值 (`float values`)，方法是将字符“U”、“L”、“LL”和“F”分别追加到记号值末尾。例如，创建封装浮点值的 `NSNumber` 对象，可以编写如下代码：

```
NSNumber *myFloatValue = @3.2F
```

## 集对象

Objective-C 代码中的大多数集对象都是一种基础集类 (`NSArray`、`NSSet` 和 `NSDictionary`) 的实例。这些类用于管理对象组，因此要添加到集 (`collection`) 中的任何项目都必须是 Objective-C 类的实例。如果要添加标量值，就必须先创建合适的 `NSNumber` 或 `NSValue` 实例来表示它。

添加进集的任何对象的生命周期都将不短于集。因为集类会使用强引用来跟踪其内容。除了跟踪其内容之外，每个集类都便于您执行特定的任务，如枚举、访问特定项目或是找出特殊的对象是否属于集的一部分。

`NSArray`、`NSSet` 和 `NSDictionary` 类的内容在创建时就应设定。因为它们不能随时间而变化，所以被称为不可变。每个类还有一个可变的子类，允许您随意添加或移除对象。不同类型的集采用不同的方式组织它们所包含的对象：

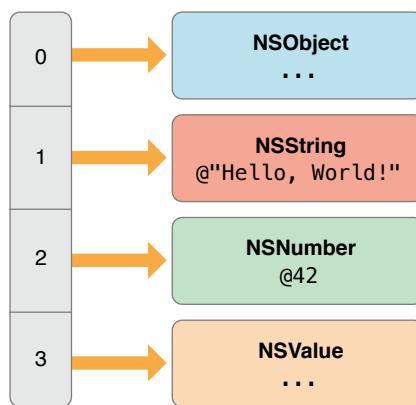
- `NSArray` 和 `NSMutableArray`—数组，包含有序的对象集。通过在数组中指定对象的位置（即索引）来访问对象。数组中首个元素的索引是 0（零）。
- `NSSet` 和 `NSMutableSet`—集合，储存无序的对象集，其中每个对象仅出现一次。一般是将测试或过滤器应用到集合中的对象，来访问这些集合中的对象。

- `NSDictionary` 和 `NSMutableDictionary`—字典，其条目储存为键—值对；键是唯一的标识符，通常为字符串，而值则是您要储存的对象。通过指定键，您可以访问该对象。

## 数组

数组 (`NSArray`) 用于表示有序的对象列表。只要求每个项目都是 Objective-C 对象；不要求每个对象都是同一个类的实例。

如果要保持数组中的顺序，每个元素都应储存在从 0 开始的索引中。



## 创建数组

与本章前文所述的值类一样，您可以通过分配和初始化、类工厂方法或数组字面常量来创建数组。

根据对象数量的不同，可用的初始化和工厂方法也多种多样。

```
+ (id)arrayWithObject:(id)anObject;
+ (id)arrayWithObjects:(id)firstObject, ...;
- (id)initWithObjects:(id)firstObject, ...;
```

由于 `arrayWithObjects:` 和 `initWithObjects:` 方法都采用了以 `nil` 结束且数量可变的参数，所以您必须包括 `nil` 并将其作为最后一个值。

```
NSArray *someArray =
[NSArray arrayWithObjects:someObject, someString, someNumber, someValue, nil];
```

此示例会创建一个如上文所示的数组。第一个对象 `someObject` 的数组索引为 0；最后一个对象 `someValue` 的索引则为 3。

如果所提供的其中一个值为 `nil`，则有可能使项目列表意外截断。

```
id firstObject = @"someString";
id secondObject = nil;
id thirdObject = @"anotherString";
NSArray *someArray =
[NSArray arrayWithObjects:firstObject, secondObject, thirdObject, nil];
```

在这种情况下，`someArray` 只会包含 `firstObject`，因为 `secondObject`（即 `nil`）会被解析为项目列表的末尾。

使用紧凑语法创建数组字面常量也是可能的。

```
NSArray *someArray = @[firstObject, secondObject, thirdObject];
```

使用此语法时，请勿使用 `nil` 来结束对象列表；实际上，`nil` 是无效值。例如，如果您尝试执行以下代码，那么会在运行时中捕获到一个异常：

```
id firstObject = @"someString";
id secondObject = nil;
NSArray *someArray = @[firstObject, secondObject];
// exception: "attempt to insert nil object"
```

## 查询数组对象

创建数组后，可以通过查询来获得信息，如其中有多少个对象，或者其中是否包含给定的项目。

```
NSUInteger numberOfRowsInSection = [someArray count];

if ([someArray containsObject:someString]) {
    ...
}
```

还可以按照给定索引查询数组来找到项目。如果请求的索引无效，那么会在运行时中获得越界异常。为了避免得到异常，应始终首先检查项目的数量。

```
if ([someArray count] > 0) {
```

```
    NSLog(@"First item is: %@", [someArray objectAtIndex:0]);  
}
```

此示例用于检查项目的数量是否大于 0。如果数量大于 0，Foundation 函数 `NSLog` 会记录第一个项目（索引为 0）的描述。

除了使用 `objectAtIndex:` 之外，还可以使用下标语法来查询数组，就像在标准的 C 数组中访问值一样。上一个示例可被重写为：

```
if ([someArray count] > 0) {  
    NSLog(@"First item is: %@", someArray[0]);  
}
```

## 排序数组对象

`NSArray` 类提供了多种方法对其收集的对象进行排序。由于 `NSArray` 是不可变的，因此这类方法都会返回新的数组，并在其中包含排好序的项目。

例如，您可以通过在每个字符串上调用 `compare:`，对字符串数组进行排序。

```
NSArray *unsortedStrings = @[@"gammaString", @"alphaString", @"betaString"];  
NSArray *sortedStrings =  
    [unsortedStrings sortedArrayUsingSelector:@selector(compare:)];
```

## 可变性

虽然 `NSArray` 类本身不可变，但它仍可包含可变对象。例如，如果将可变字符串添加到不可变的数组，如下所示：

```
NSMutableString *mutableString = [NSMutableString stringWithString:@"Hello"];  
NSArray *immutableArray = @[mutableString];
```

就可让您变异该字符串。

```
if ([immutableArray count] > 0) {  
    id string = immutableArray[0];  
    if ([string isKindOfClass:[NSMutableString class]]) {  
        [string appendString:@" World!"];  
    }  
}
```

```
}
```

如果要在初始创建数组后添加或移除对象，可使用 `NSMutableArray`，它提供了很多方法来添加、移除或替换一个或多个对象。

```
NSMutableArray *mutableArray = [NSMutableArray array];
[mutableArray addObject:@"gamma"];
[mutableArray addObject:@"alpha"];
[mutableArray addObject:@"beta"];

[mutableArray replaceObjectAtIndex:0 withObject:@"epsilon"];
```

此示例创建了由对象 `@"epsilon"`、`@"alpha"` 和 `@"beta"` 构成的数组。

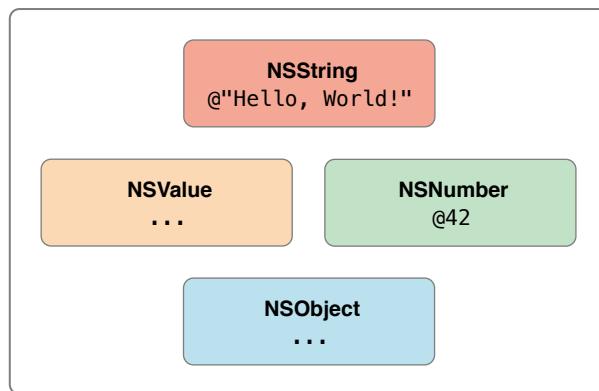
还可以对可变数组进行适当排序，而无需创建二级数组。

```
[mutableArray sortUsingSelector:@selector(caseInsensitiveCompare:)];
```

在这种情况下，包含在内的项目会按升序且不区分大小写的顺序排列 (`@"alpha"`、`@"beta"` 和 `@"epsilon"`)。

## 集合

集合 (`NSSet`) 对象与数组类似，只是其中包含的是各种无序的对象。



因为集合不包含顺序，所以测试成员资格时，集合比数组更快。

由于基础NSSet类是不可变的，因此在创建时就必须指定其内容，其中可使用分配和初始化或者类工厂方法。

```
NSSet *simpleSet =  
[NSSet setWithObjects:@"Hello, World!", @42, aValue, anObject, nil];
```

如同NSArray，initWithObjects:和setWithObjects:方法都采用了以nil结束且数量不固定的参数。可变的NSSet子类名称是NSMutableSet。

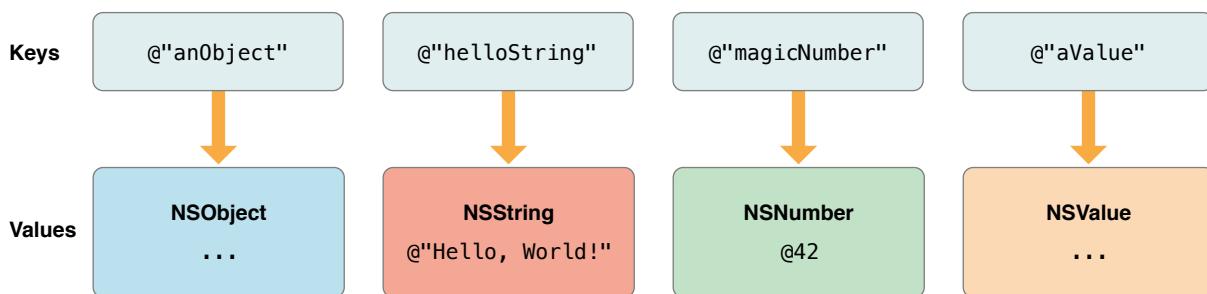
即使您多次尝试添加对象，集合也只会储存对单个对象的一次引用。

```
NSNumber *number = @42;  
NSSet *numberSet =  
[NSSet setWithObjects:number, number, number, number, nil];  
// numberSet only contains one object
```

## 字典

与简单汇集有序或无序的对象集不同，字典(NSDictionary)会储存与给定键相关的对象，用于以后的检索。

最佳实践是将字符串对象用作字典键。



虽然其他对象也可以用作键，但要注意，每个键都会被拷贝以供字典使用，并且必须支持NSCopying。不过，如果要使用键—值编码，则必须为字典对象使用字符串键。若要了解更多信息，请参阅《Key-Value Coding Programming Guide》（键值编码编程指南）。

## 创建字典

您可以使用分配、初始化，或者类工厂方法来创建字典，如下所示：

```
NSDictionary *dictionary = [NSDictionary dictionaryWithObjectsAndKeys:  
    someObject, @"anObject",  
    @"Hello, World!", @"helloString",  
    @42, @"magicNumber",  
    someValue, @"aValue",  
    nil];
```

对于 `dictionaryWithObjectsAndKeys:` 和 `initWithObjectsAndKeys:` 方法，每个对象都会在其键前进行声明，并且对象列表和键必须以 `nil` 结束。

Objective-C 提供了一种简洁的语法来创建字典字面常量。

```
NSDictionary *dictionary = @{  
    @"anObject" : someObject,  
    @"helloString" : @"Hello, World!",  
    @"magicNumber" : @42,  
    @"aValue" : someValue  
};
```

对于字典字面常量，键会在其对象前被指定，并且对象列表和键不以 `nil` 结束。

## 查询字典

创建字典后，您可以查询储存在给定键中的对象。

```
NSNumber *storedNumber = [dictionary objectForKey:@"magicNumber"];
```

如果找不到该对象，`objectForKey:` 方法会返回 `nil`。

同样也可以用下标语法来替代 `objectForKey:`。

```
NSNumber *storedNumber = dictionary[@"magicNumber"];
```

## 可变性

创建字典后，如果需要添加或移除对象，可使用 `NSMutableDictionary` 子类。

```
[dictionary setObject:@"another string" forKey:@"secondString"];
```

```
[dictionary removeObjectForKey:@"anObject"];
```

## 使用 NSNull 表示 nil

因为在 Objective-C 中，nil 表示“无对象”。因此不可能将 nil 添加到此节所描述的集类中。如果要在集 (collection) 中表示“无对象”，应使用 NSNull 类。

```
NSArray *array = @[@"string", @42, [NSNull null]];
```

使用 NSNull，null 方法始终都会返回相同的实例。按此方式工作的类称为单例类。您可以按如下所示的方法来检查数组中的对象是否等于已共享的 NSNull 实例：

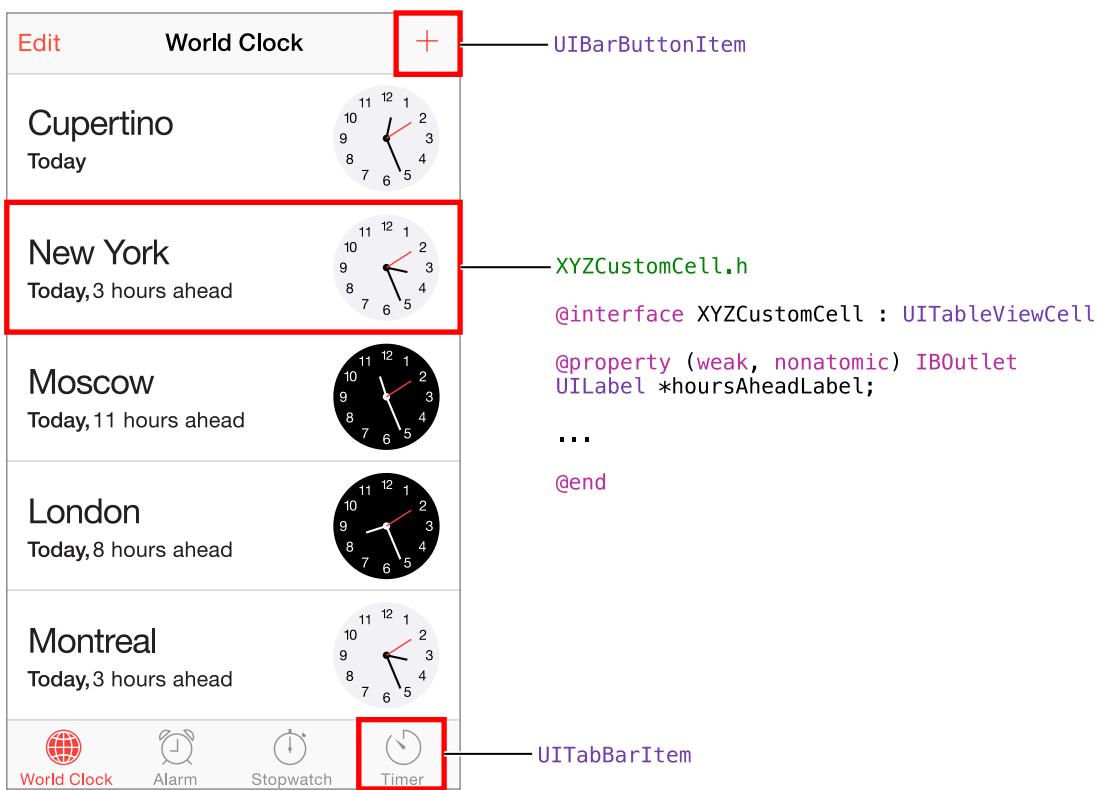
```
for (id object in array) {
    if (object == [NSNull null]) {
        NSLog(@"Found a null object");
    }
}
```

虽然 Foundation 框架包含的功能比文中所述要丰富的多，但也不需要您立即了解每一处细节。如果确实想更深入学习 Foundation，可以参阅《*Foundation Framework Reference*》（Foundation 框架参考）。现在，您已经掌握了足够的信息来继续完成 ToDoList 应用程序了，那么就从编写自定数据类开始吧。

# 编写自定类

开发 iOS 应用程序时，很多情况下都需要编写自定类。当您需要将自定行为与数据打包在一起时，自定类非常有用。在自定类中，可以定义您自己的行为来储存、操控和显示数据。

例如，iOS 的“时钟”应用程序中的“世界时钟”标签。比起标准表格视图单元格，此表格视图中的单元格需要显示更多内容。这是实现子类的绝好机会，可以扩展 `UITableViewCell` 的行为，从而在给定的表格视图单元格中显示更多自定数据。如果要设计此自定类，您可能要为标签添加 `Outlet`，使信息前显示小时数；还要添加图像视图，使单元格的右侧显示自定时钟。



本章会告诉您需要了解哪些 Objective-C 语法和类结构，才能实现 `ToDoList` 应用程序的行为。还讨论了 `XYZToDoItem` 的设计，该自定类将表示待办事项列表上的单个项目。在第三个教程中，您将真正实现该类，并将它添加到您的应用程序中。

## 声明并实现类

在 **Objective-C** 中，类的说明需要两个不同的部分：接口和实现。接口准确指定了一个给定类型的对象，如何专用于其他对象。换句话说，它定义了类的实例与外部世界之间的公共接口。这个实现包括的可执行代码，涵盖了接口中声明的每个方法。

对象的设计，应当隐藏其内部实现的细节。在 **Objective-C** 中，接口和实现通常放在单独的文件中，这样您只需要将接口设定为公共属性。和 C 代码一样，您需要定义头文件和源代码文件，将公共声明与代码的实现细节分开。接口文件具有 .h 扩展名，实现文件具有 .m 扩展名。（您将在“[教程：添加数据](#)（第 87 页）”中为 XYZToDoItem 类真正创建这些文件；眼下我们只需跟着文稿来，因为所有内容都将讲到。）

### 接口

**Objective-C** 语法经常采用以下方式来声明类接口：

```
@interface XYZToDoItem : NSObject  
  
@end
```

该示例声明了名为 XYZToDoItem 的类，它继承自 NSObject。

公共属性和行为在 @interface 声明内部定义。在此示例中，除了超类外，没有指定任何内容，所以 XYZToDoItem 实例唯一可用的行为继承自 NSObject。所有对象都要有一个最基本的行为，因此在默认情况下，它们必须继承自 NSObject（或它的一个子类）。

### 实现

**Objective-C** 语法经常采用以下方式来声明类实现：

```
#import "XYZToDoItem.h"  
  
@implementation XYZToDoItem  
  
@end
```

如果您在类接口中声明了任何方法，那么需要在此文件中实现它们。

## 储存对象的数据的属性

考虑待办事项需要保存的信息。您可能需要知道事项的名称，创建时间，以及是否已经完成。在自定 XYZToDoItem 类中，此信息储存在属性中。

这些属性的声明在接口文件 (XYZToDoItem.h) 内部。如下所示：

```
@interface XYZToDoItem : NSObject

@property NSString *itemName;
@property BOOL completed;
@property NSDate *creationDate;

@end
```

在此示例中，XYZToDoItem 类声明了三个公共属性。这些属性完全可供公开访问。在公开访问的情况下，其他对象可以同时读取和更改这些属性的值。

您可能想要声明某个属性不能被更改（也就是说，只能读取）。指明一个属性是否只读（相对于其他属性），需要在 Objective-C 的属性声明中包括属性的特性。例如，如果您不想 XYZToDoItem 的创建日期被改变，可能需要将 XYZToDoItem 类接口更新为这样：

```
@interface XYZToDoItem : NSObject

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

@end
```

属性可以是私有的或公共的。有时让属性私有是合理的选择，这样其他类就不能查看或访问它。例如，如果您想要跟踪的标记是项目完成的日期属性，并且不想让其他类访问此信息，请将此属性设为私有，方法是将它放在实现文件 (XYZToDoItem.m) 顶部的类扩展中。

```
#import "XYZToDoItem.h"

@interface XYZToDoItem ()
@property NSDate *completionDate;
```

```
@end

@implementation XYZToDoItem

@end
```

您可以使用 **getter** 和 **setter** 来访问属性。**getter** 返回属性的值，**setter** 更改属性的值。访问 **getter** 和 **setter** 的常见语法简写是 **dot notation**（点表达式）。对于具有读写访问权限的属性，您可以使用点表达式来获取和设定属性的值。如果您有 XYZToDoItem 类的一个对象 toDoItem，可以执行以下操作：

```
toDoItem.itemName = @"Buy milk";           //Sets the value of itemName
NSString *selectedItemName = toDoItem.itemName; //Gets the value of itemName
```

## 方法用来定义对象的行为

方法用来定义对象能做些什么。方法是您定义的一段代码，用来执行类中的任务或子程序。它可以访问类中储存的数据，并使用该信息来执行一些操作。

例如，要让待办事项 (XYZToDoItem) 能够被标记为已完成，您可以将 markAsCompleted 方法添加到类接口。稍后，您将在类实现中实现该方法的行为，正如“[实现方法（第 85 页）](#)”中所述。

```
@interface XYZToDoItem : NSObject

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

@end
```

方法名称前面的减号 (-) 指示它是一个实例方法，可以用该类的对象调用。减号将它与类方法区分开来，类方法用加号 (+) 来表示。类方法可通过类本身来调用。类方法的一个常见示例是类工厂方法，您已经在“[处理 Foundation（第 71 页）](#)”中学习过。您也可以使用类方法来访问与类相关的一些共享信息。

在声明的开始，用圆括号括住 void 关键词，表示方法不会返回值。这种情况下，markAsCompleted 方法没有任何参数。“[方法参数（第 85 页）](#)”中会讲述有关参数的更多信息。

## 方法参数

在声明方法时加上参数，可在调用方法时传递一些信息。

例如，您可以根据之前的代码片段修改 `markAsCompleted` 方法，让其带单个参数，借此确定项目是否标记为完成或未完成。用这种方式，您可以切换项目的完成状态，而不只是将它设为完成。

```
@interface XYZToDoItem : NSObject

@property NSString *itemName;
@property BOOL completed;
@property (readonly) NSDate *creationDate;
- (void)markAsCompleted:(BOOL)isComplete;

@end
```

现在，您的方法采用了一个参数 `isComplete`，其类型为 `BOOL`。

通过名称引用带一个参数的方法时，冒号将作为方法名称的一部分，所以更新后的方法名称是 `markAsCompleted:`。如果方法有多个参数，那么它会被分解，并插入参数名称。如果要将另一个参数添加到此方法，其声明将会是这样的：

```
- (void)markAsCompleted:(BOOL)isComplete onDate:(NSDate *)date;
```

在这里，方法的名称写为 `markAsCompleted:onDate:`。实现时会使用名称 `isComplete` 和 `date`（会将这些名称当作变量），来访问方法被调用时所提供的值。

## 实现方法

方法实现使用大括号来包括相关代码。方法的名称必须与接口文件中的对应方法相同，并且参数和返回类型必须完全匹配。

这是添加到 `XYZToDoItem` 类接口的 `markAsCompleted:` 方法的简单实现：

```
@implementation XYZToDoItem
- (void)markAsCompleted:(BOOL)isComplete {
    self.completed = isComplete;
}
@end
```

与属性一样，方法也可以是私有的或公共的。公共方法在公共接口中声明，因此可以被其他对象查看和调用。它们的相应实现存在于实现文件中，其他对象看不到。私有方法仅有一个实现，并与类相同，这意味着只可以在类实现中调用它们。这是一个强大的机制，可将内部行为添加到类而不允许其他对象进行访问。

例如，假设您想要让待办事项的 `completionDate` 保持更新。当待办事项被标记为完成，就将 `completionDate` 设定为当前日期。当它被标记为未完成，就将 `completionDate` 设定为 `nil`，因为它尚未完成。由于更新待办事项的 `completionDate` 是自包含任务，所以最好的做法是为其编写专属方法。但是，应当确定其他对象不能调用此方法。否则，其他对象可以随时将待办事项的 `completionDate` 设定为任何内容。因此，请将此方法设为私有。

现在，更新 `XYZToDoItem` 的实现来包括私有方法 `setCompletionDate`，该方法将会在 `markAsCompleted:` 内部调用，以便在待办事项被标记为已完成或未完成时，更新待办事项的 `completionDate`。请注意，您不会添加任何内容到接口文件，因为不想其他对象看到此方法。

```
@implementation XYZToDoItem
- (void)markAsCompleted:(BOOL)isComplete {
    self.completed = isComplete;
    [self setCompletionDate];
}
- (void)setCompletionDate {
    if (self.completed) {
        self.completionDate = [NSDate date];
    } else {
        self.completionDate = nil;
    }
}
@end
```

现在，您已经使用 `XYZToDoItem` 类定义了待办事项列表的基本实现。`XYZToDoItem` 以属性的形式储存自身相关的信息——名称、创建日期和完成状态，并且会使用方法定义行为——被标记为已完成或未完成。这是您在下一个教程中实现 `ToDoList` 应用程序所需的功能扩展。不过，您随时可以尝试将自己的属性和方法添加到类，将新的行为集成到您的应用程序中。

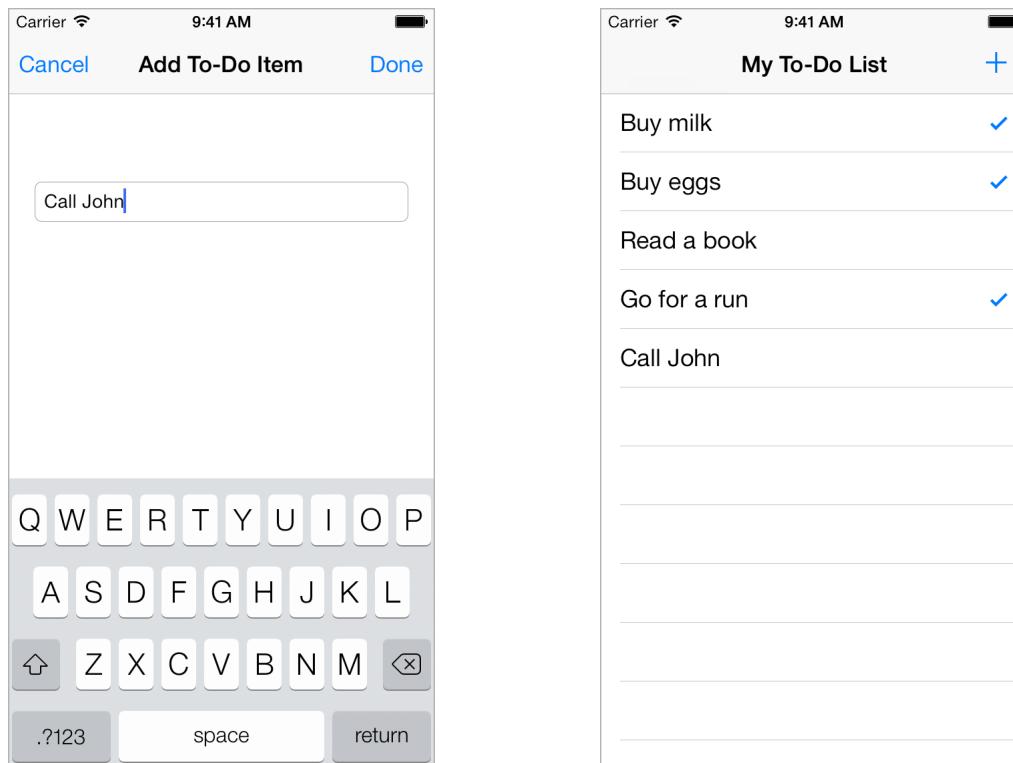
# 教程：添加数据

本教程以第二个教程（“[教程：串联图（第45页）](#)”）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[编写自定类](#)（第 81 页）中讨论过。

### 创建 `XYZToDoItem` 类

1. 选取“File”>“New”>“File”（或按下 **Command-N**）。

这时将会出现一个对话框，提示您为新文件选取模板。

2. 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 XYZToDoListViewController 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
```

3. 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 {
}
```

2. 在此方法中，创建几个列表项目，并将它们添加到数组。

```
- (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。

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    #warning Potentially incomplete method implementation.

    // Return the number of sections.
    return 0;
}
```

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。

4. 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：

```
- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section
{
    #warning Incomplete method implementation.

    // Return the number of rows in the section.
}
```

```
    return 0;  
}
```

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]`。

### 3. 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:cellForRowIndexPath:` 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。

#### 配置表格视图

1. 打开串联图。
2. 在大纲中选择表格视图。
3. 选定表格视图后，在实用工具区域中打开“Attributes”检查器 。
4. 在“Attributes”检查器中，将表格视图的“Content”属性从“Static Cells”更改为“Dynamic Prototypes”。

**Interface Builder**会采用您配置的静态单元格，并将它们全部转换为原型。原型单元格，顾名思义，是使用您要显示的文本样式、颜色、图像或其他属性进行配置，并在运行时从数据源获取其数据的单元格。数据源会为每一行载入一个原型单元格，然后配置该单元格来显示该行的数据。

要载入正确的单元格，数据源需要知道单元格的名称，并且该名称也必须在串联图中进行配置。

设定原型单元格名称时，也将配置另一个属性—单元格选择样式，该样式用于确定用户轻按单元格时单元格的外观。将单元格选择样式设定为“None”，使用户轻按单元格时单元格不会高亮显示。这是当用户轻按待办事项列表中的项目，将其标记为已完成或未完成时，您想要单元格呈现的行为。稍后会在本教程中实现该功能。

#### 配置原型单元格

1. 在表格中选择第一个表格视图单元格。
2. 在“Attributes”检查器中，找到“Identifier”栏并键入 ListPrototypeCell。
3. 在“Attributes”检查器中，找到“Selection”栏并选取“None”。

您也可以更改原型单元格的字体或其他属性。基本配置很容易完成，您可以轻松记下。

下一步是实现 `tableView:cellForRowAtIndexPath:` 方法，让数据源为给定行配置单元格。表格视图在想要显示给定行时会调用此数据源方法。对于行数较少的表格视图，所有行可能会同时出现在屏幕上，所以表格中的每一行都会调用此方法。但是，行数很多的表格视图在给定时间内只会显示全部项目中的一小部分。最有效的方式是让表格视图仅请求要显示行的单元格，而这一点可通过 `tableView:cellForRowAtIndexPath:` 让表格视图实现。

对于表格中的任何给定行，取回 `ToDoItems` 数组中的相应条目，然后将单元格的文本标签设定为项目的名称。

#### 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。

3. 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";
```

4. 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 = ToDoItem.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

1. 在项目导航器中，选择 XYZToDoListViewController.m。
2. 将以下代码行添加到文件末尾，在 @end 行正上方：

```
#pragma mark - Table view delegate

- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
}
```

请试着键入第二行，而不是拷贝和粘贴。您将发现代码补全是 Xcode 最节省时间的功能之一。当 Xcode 显示出可能的补全建议列表时，请滚动浏览列表，找到想要的建议，然后按下 **Return** 键。Xcode 会为您插入整行。

3. 您想要响应轻按，但并不想让单元格保持选定状态。添加以下代码，让单元格在选定后立即取消选定：

```
[tableView deselectRowAtIndexPath:indexPath animated:NO];
```

4. 在 `ToDoItems` 数组中搜索相应的 `XYZToDoItem`。

```
XYZToDoItem *tappedItem = [self.toDoItems objectAtIndex:indexPath.row];
```

5. 切换被轻按项目的完成状态。

```
tappedItem.completed = !tappedItem.completed;
```

6. 告诉表格视图重新载入您刚更新过数据的行。

```
[tableView reloadRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationNone];
```

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

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

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，并没有任何反应。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

1. 前往 `tableView:cellForRowAtIndexPath:` 方法。
2. 在设定单元格的文本标签的代码行下方添加以下代码：

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

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";
```

```
UITableViewController *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。

## 添加新项目

构建待办事项列表应用程序功能的最后一步是实现添加项目的能力。当用户在 `XYZAddToDoItemViewController` 场景的文本栏中输入项目名称，并轻按“Done”按钮时，您想要视图控制器创建一个新的列表项目并将其传递回 `XYZToDoListViewController`，以显示在待办事项列表中。

首先，您需要拥有一个列表项目来进行配置。就像表格视图那样，视图控制器是将界面连接到模型的逻辑位置。为 `XYZAddToDoItemViewController` 添加一个属性来保存新的待办事项。

### 将 `XYZToDoItem` 添加到 `XYZAddToDoItemViewController` 类

1. 在项目导航器中，选择 `XYZAddToDoItemViewController.h`。

由于稍后需要从表格视图控制器访问列表项目，所以务必将其设为公共属性。这就是为什么要在接口文件 `XYZAddToDoItemViewController.h` 中声明它，而不是在实现文件 `XYZAddToDoItemViewController.m` 中声明的原因所在。

2. 将 `import` 声明添加到 @interface 行上方的 `XYZToDoItem` 类中。

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

3. 将 `ToDoItem` 属性添加到该接口。

```
@interface XYZAddToDoItemViewController : UIViewController  
  
@property XYZToDoItem *ToDoItem;  
  
@end
```

要获得新项目的名称，视图控制器需要访问用户输入名称的文本栏。要实现此功能，请创建从 XYZAddToDoItemViewController 类到串联图中的文本栏的连接。

#### 将文本栏连接到视图控制器

1. 在大纲视图中，选择 XYZAddToDoItemViewController 对象。
2. 点按窗口工具栏右上角的“Assistant”按钮，打开辅助编辑器。

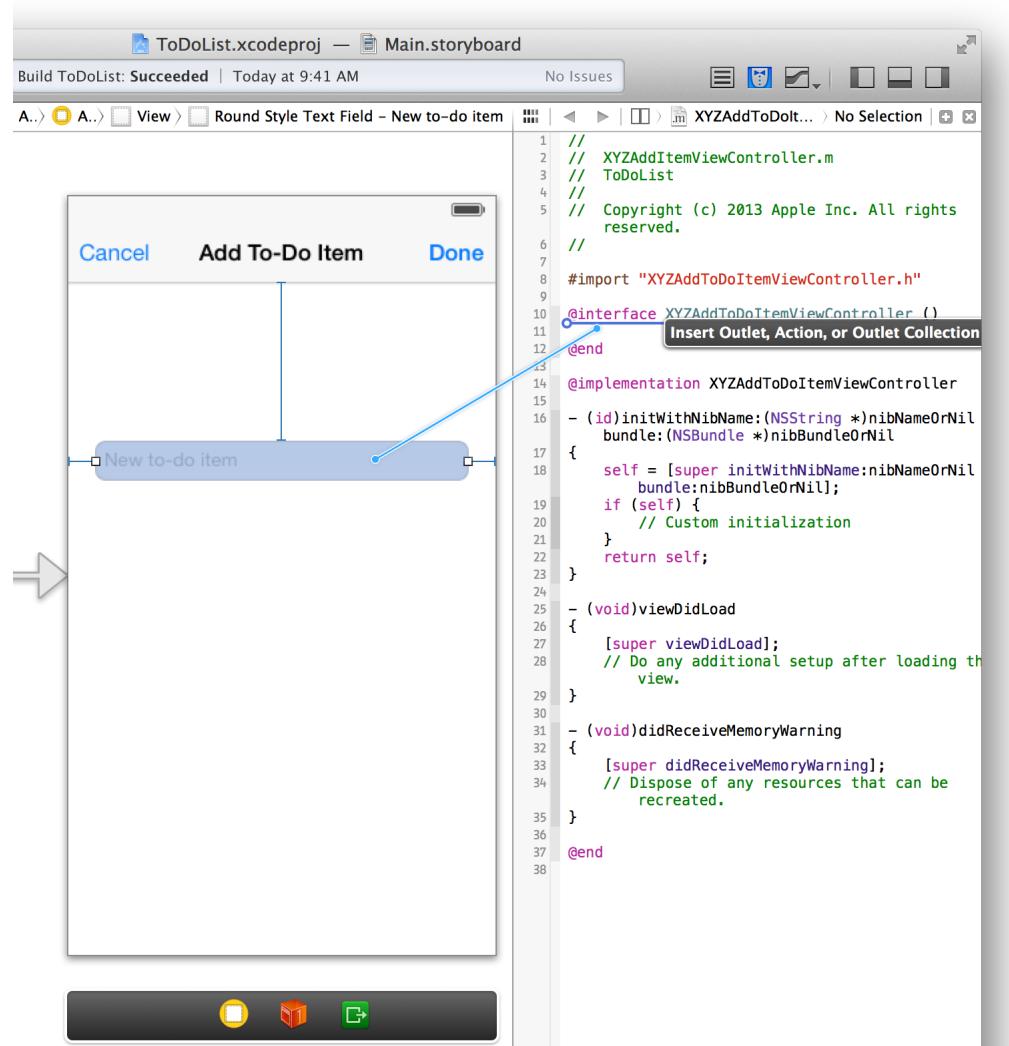


右边的编辑器出现时，应该显示有 XYZAddToDoItemViewController.m。如果未显示，请点按右边的编辑器中的文件名，并选取 XYZAddToDoItemViewController.m。

辅助编辑器可让您一次打开两个文件，以便能够在它们之间执行操作。例如，在源文件中键入一个属性，而源文件的对象又在接口文件中。

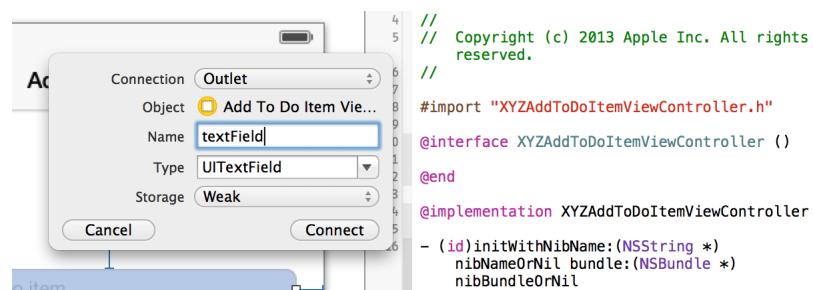
3. 在串联图中选择文本栏。

4. 按住 Control 键从画布上的文本栏拖到右边编辑器中的代码显示窗口，到达 XYZAddToDoItemViewController.m 中的 @interface 行正下方时停止拖移。



5. 在出现的对话框中，为“Name”栏键入“textField”。

让选项的其余部分保持不变。您的对话框应如下图所示：



6. 点按“Connect”。

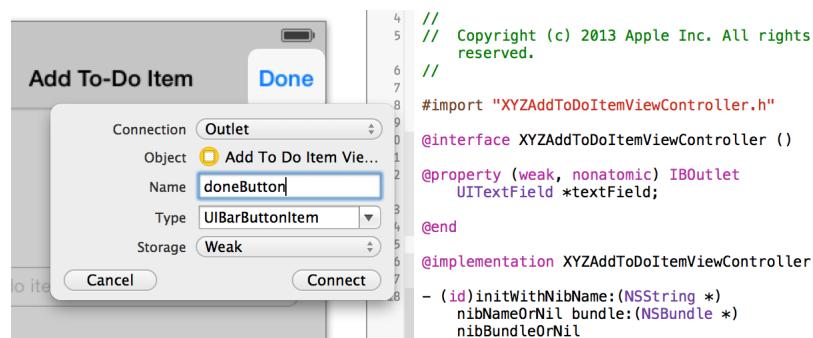
Xcode 会将必要的代码添加到 XYZAddToDoItemViewController.m，用于储存指向文本栏的指针，并配置串联图来设置该连接。

另外，您需要知道何时创建项目。如果想要仅在“Done”按钮被轻按时才创建项目，那么请将“Done”按钮添加为 Outlet。

#### 将“Done”按钮连接到视图控制器

1. 在串联图中，打开辅助编辑器，将最右边的窗口设定为 XYZAddToDoItemViewController.m。
2. 在串联图中选择“Done”按钮。
3. 按住 **Control** 键从画布上的“Done”按钮拖到右边的编辑器中的代码显示窗口，到达 XYZAddToDoItemViewController.m 中的 textField 属性正下方的行时停止拖移。
4. 在出现的对话框中，在“Name”栏键入“doneButton”。

保持选项的其余部分不变。对话框应如下图所示：



5. 点按“Connect”。

您现在有了识别“Done”按钮的方式。由于想在轻按“Done”按钮时创建一个项目，所以需要知道该按钮何时被按下。

当用户轻按“Done”按钮时，它会启动一个 `unwind segue`，返回到待办事项列表，这正是您在第二个教程中配置的接口。在 `segue` 执行前，系统通过调用 `prepareForSegue:`，给所包含的视图控制器一次准备机会。这便是检查用户是否轻按了“Done”按钮的时候。如果是，将会创建一个新的待办事项。您可以检查轻按了哪一个按钮，如果是“Done”按钮，将会创建项目。

#### 轻按“Done”按钮后创建项目

1. 在项目导航器中选择 XYZAddToDoItemViewController.m。
2. 在 @implementation 行下方添加 `prepareForSegue:` 方法：

```
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender  
{  
}
```

3. 在此方法中，请查看“Done”按钮是否按下。

如果未按下就不存储项目，而是让方法返回但不执行任何其他操作。

```
if (sender != self.doneButton) return;
```

4. 查看一下文本栏中是否有文本。

```
if (self.textField.text.length > 0) {  
}
```

5. 如果有文本，将会创建一个新项目，并用文本栏中的文本为其命名。另外，请确保完成状态被设定为“NO”。

```
self.todoItem = [[XYZToDoItem alloc] init];  
self.todoItem.itemName = self.textField.text;  
self.todoItem.completed = NO;
```

如果没有文本，您就不需要存储项目，也不需要执行任何其他操作。

您的 `prepareForSegue:` 方法应如下图所示：

```
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender  
{  
    if (sender != self.doneButton) return;  
    if (self.textField.text.length > 0) {  
        self.todoItem = [[XYZToDoItem alloc] init];  
        self.todoItem.itemName = self.textField.text;  
        self.todoItem.completed = NO;  
    }  
}
```

既然创建了一个新项目，那么就需要将该项目传递回 `XYZToDoListViewController`，以便它可以将项目添加到待办事项列表。要完成此功能，请重新访问您在第二个教程中编写的 `unwindToList:` 方法。当用户轻按“Cancel”或“Done”按钮，`XYZAddToDoItemViewController` 场景会在关闭时调用该方法。

就像作为 `unwind segue` 目标的所有方法一样，`unwindToList:` 方法也采用 `segue` 作为参数。`segue` 参数是从 `XYZAddToDoItemViewController` 展开并返回到 `XYZToDoListViewController` 的过渡。由于 `segue` 是两个视图控制器之间的过渡，所以知道它的源视图控制器是 `XYZAddToDoItemViewController`。

通过向 `segue` 对象请求其源视图控制器，您可以用 `unwindToList:` 方法访问储存在源视图控制器中的任何数据。目前，您想要访问 `ToDoItem`。如果它是 `nil`，则该项目并未创建。原因可能是文本栏没有文本，或者是用户轻按了“Cancel”按钮。如果 `ToDoItem` 有值，则可以取回该项目，再添加到 `ToDoItems` 数组，并通过重新载入表格视图中的数据将其显示在待办事项列表中。

### 储存并显示新项目

1. 在项目导航器中，选择 `XYZToDoListViewController.m`。
2. 将 `import` 声明添加到 @interface 行上方的 `XYZAddToDoItemViewController` 类中。

```
#import "XYZAddToDoItemViewController.h"
```

3. 找到您在第二个教程中添加的 `unwindToList:` 方法。
4. 在此方法中，取回源视图控制器 `XYZAddToDoItemViewController`，即您要展开的控制器。

```
XYZAddToDoItemViewController *source = [segue sourceViewController];
```

5. 取回控制器的待办事项。

```
XYZToDoItem *item = source.todoItem;
```

这就是轻按“Done”按钮时创建的项目。

6. 看看这个项目是否存在。

```
if (item != nil) {  
}
```

如果是 `nil`，可能是“Cancel”按钮关闭了屏幕，或者是文本栏中没有文本，因此不必存储该项目。

如果项目存在，请添加到 `ToDoItems` 数组。

```
[self.todoItems addObject:item];
```

7. 重新载入表格中的数据。

因为表格视图不会跟踪其数据，所以数据源（在这个程序中是表格视图控制器）有责任通知表格视图何时有新数据供其显示。

```
[self.tableView reloadData];
```

您的 `unwindToList:` 方法应如下图所示：

```
- (IBAction)unwindToList:(UIStoryboardSegue *)segue
{
    XYZAddToDoItemViewController *source = [segue sourceViewController];
    XYZToDoItem *item = source.todoItem;
    if (item != nil) {
        [self.todoItems addObject:item];
        [self.tableView reloadData];
    }
}
```

检查点：运行您的应用程序。现在，当您点按添加按钮 (+) 并创建一个新项目时，它应该会显示在待办事项列表中。恭喜您！您已经创建了一个应用程序，它能接收用户的输入，将其储存在对象中，并在两个视图控制器之间传递该对象。对于基于串联图的应用程序，这是在场景之间移动数据的基础。

## 小结

您差不多完成了开发 iOS 应用程序的入门之旅。最后一部分更详细地讲述了如何查询文稿材料，并且为您学习创建更高级的应用程序给出了一些后续建议。

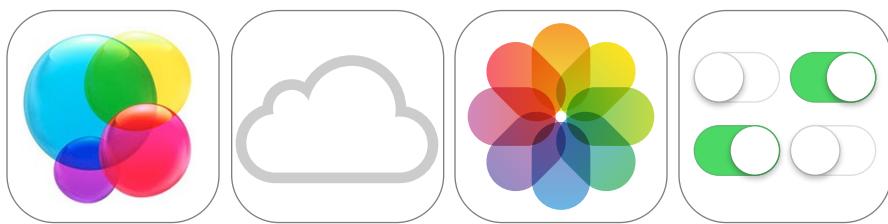
# 后续步骤

- [iOS 技术](#) (第 106 页)
- [查找信息](#) (第 109 页)
- [接下来做什么](#) (第 121 页)

# iOS 技术

之前您已了解了如何编写一个具有简单用户界面和基本行为的应用程序。现在您或许在考虑实现更多的行为，使项目成为一款功能完备的应用程序。

在思考要添加哪些功能之前，谨记一条：您无需一切从头开始。**iOS** 提供了定义特殊功能集的框架，从游戏、媒体到密保和数据管理，每样均可以整合到您的应用程序中。您已使用 **UIKit** 框架来设计应用程序的用户界面，并使用 **Foundation** 框架将常见数据结构和行为归并到代码。这是 **iOS** 应用程序开发中两个最常用的框架，而您能使用的远不止这些。



本章节大致概述了可能在应用程序中采用的技术和框架。您不妨将本章节当做探索可行技术的起点。有关 **iOS** 中可用技术的完整概述，请参阅《*iOS Technology Overview*》（**iOS** 技术概述）。

## 用户界面

**iOS** 具有许多框架和技术，用于创建和润色应用程序的用户界面。

**UIKit**。**UIKit** 框架提供的类可用于创建触摸式用户界面。所有 **iOS** 应用程序都基于 **UIKit**，因此您无法在没有框架的情况下交付应用程序。**UIKit** 提供基础结构，用于在屏幕上绘图、处理事件，以及创建通用用户界面元素。通过管理屏幕上显示的内容，**UIKit** 还能组织复杂的应用程序。有关更多信息，请参阅《*UIKit Framework Reference*》（**UIKit** 框架参考）。

**Core Graphics**。**Core Graphics** 是一种基于 C 语言的低层次框架，在处理高品质矢量图形、基于路径的绘图、变换、图像和数据管理等方面，它将是您的得力助手。当然，在 **iOS** 中创建图形，最简而有效的方法是将预渲染的图像与 **UIKit** 框架的标准视图和控制配合使用，并让 **iOS** 完成绘图。毕竟，**UIKit** 是一种高层次的框架，它同时还提供用于自定绘图的类，包括路径、颜色、图案、渐变、图像、文本和变换，请尽可能地使用它们来代替 **CoreGraphics**。有关更多信息，请参阅《*CoreGraphics Framework Reference*》（**Core Graphics** 框架参考）。

**Core Animation。** Core Animation 是一种能让您制作高级动画和视觉效果的技术。UIKit 提供的动画，是建立在 Core Animation 技术之上的。如果您需要超出 UIKit 功能的高级动画，可以直接使用 Core Animation。Core Animation 接口包含在 Quartz Core 框架中。借助 Core Animation，您将能创建不同层次的层对象，并对它们进行操控、旋转、缩放、变换等等。通过使用大家所熟悉的 Core Animation 视图式抽象，您可以创建动态用户界面，而无需使用底层的图形 API，如 OpenGL ES 等。有关更多信息，请参阅《Core Animation Programming Guide》（Core Graphics 框架参考）。

## 游戏

在开发 iOS 游戏时，您需要尝试多种技术。

**Game Kit。** Game Kit 框架提供了排行榜、成就和其他功能，可添加到 iOS 游戏中。有关更多信息，请参阅《Game Kit Framework Reference》（Game Kit 框架参考）。

**Sprite Kit。** Sprite Kit 框架在使任意纹理图像或角色产生动画效果方面提供图形支持。它不仅是一台图形引擎，还能提供物理支持使对象变得更加生动。对于游戏和其他要求复杂动画链的应用程序，Sprite Kit 不失为一个好的选择。（其他类型的用户界面动画，则可使用 Core Animation 代为处理。）有关更多信息，请参阅《Sprite Kit Programming Guide》（Sprite Kit 编程指南）。

**OpenGL ES。** OpenGL ES 是一种底层框架，它为硬件加速的 2D 和 3D 绘图提供工具支持。Apple 实施的 OpenGL ES 标准，能与设备硬件紧密协作，从而为全屏幕游戏类应用程序提供很高的帧速率。OpenGL ES 是一种底层的、专注于硬件的 API，因此具有较高的学习难度，并将对您的应用程序的整体设计产生显著影响。（对于要求高性能图形以用于更多特定用途的应用程序，请考虑使用 Sprite Kit 或 Core Animation。）有关更多信息，请参阅《OpenGL ES Programming Guide for iOS》（iOS 的 OpenGL ES 编程指南）。

**Game Controller。** Game Controller 框架能让您快速找到接入 Mac 或 iOS 设备的控制器。在设备上找到控制器后，您的游戏会将控制输入读取为正常游戏设置的一部分。这些控制器为玩家操作游戏提供了新的方式。Apple 具有为硬件控制器专设的技术规格，以确保控制器的控制元素类别一致，玩家和游戏设计者们均可以此为依据。有关更多信息，请参阅《GameControllerFrameworkReference》（GameController 框架参考）。

## 数据

处理应用程序数据时，请思考现有框架中有哪些可用的功能。

**Core Data。** Core Data 框架管理应用程序的数据模型。借助 Core Data，您可以创建模型对象（称为被管理的对象）。管理那些对象之间的关系，并通过框架更改数据。Core Data 利用内建的 SQLite 技术，高效地储存和管理数据。有关更多信息，请参阅《Core Data Framework Reference》（Core Data 框架参考）。

**Foundation。**您已在本指南的前文中接触过 Foundation。Foundation 框架定义了 Objective-C 类的最底层。除了提供一组基本且实用的对象类，本框架还介绍了数个范例来阐明 Objective-C 语言中未涉及的行为。与其他框架相比，本框架包括了表示基本数据类型的类（如字串和数字），以及用于储存其他对象的集类。有关更多信息，请参阅《*Foundation Framework Reference*》（Foundation 框架参考）。

## Media

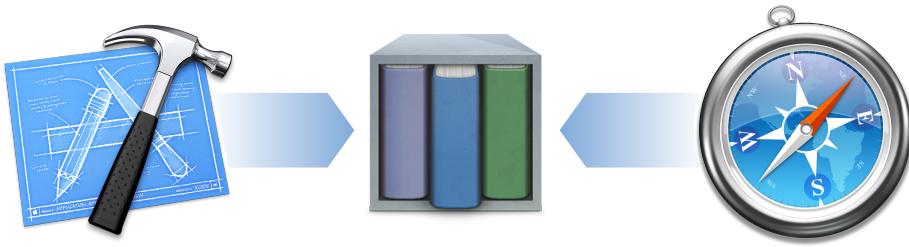
Media 框架提供多种功能，用于处理应用程序中的音频和视频。

**AV Foundation。** AV Foundation 可用于播放和创建基于时间的音频视觉媒体，是数个具有该功能框架中的一个。例如，您可以使用 AV Foundation 来检查、创建、编辑媒体文件，或对其进行重新编码。您还可以通过它获取设备的输入流，以及在实时捕捉和回放过程中处理视频。有关更多信息，请参阅《*AV Foundation Framework Reference*》（AV Foundation 框架参考）。

# 查找信息

开发应用程序时，您会希望先前了解或不了解的信息都随手可查。其实无需离开 Xcode，您便能获得所需的一切信息。

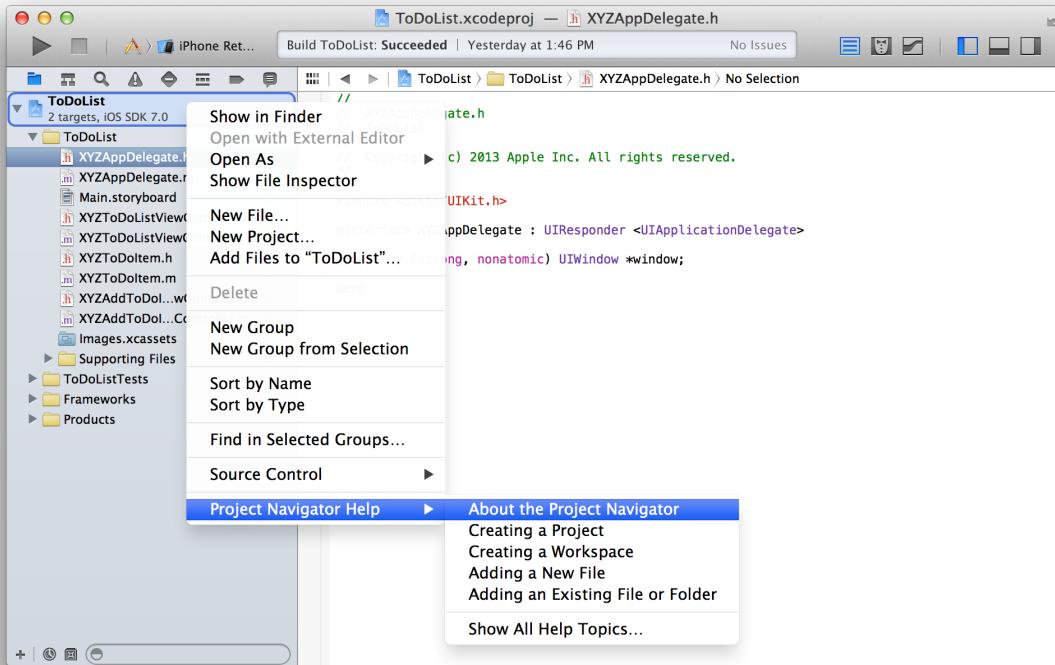
Xcode 附带了大量不同类型的系列文稿，包括通用指南和概念指南、框架和类参考资料，以及重点帮助文章。访问此类文稿的方式多种多样，如按住右键点按 Xcode 的各个区域来了解其使用方式、打开主项目窗口中的“Quick Help”面板来获得上下文相关的代码帮助，或者在“Documentation”窗口中进行搜索来查找指南和完整的 API 参考。



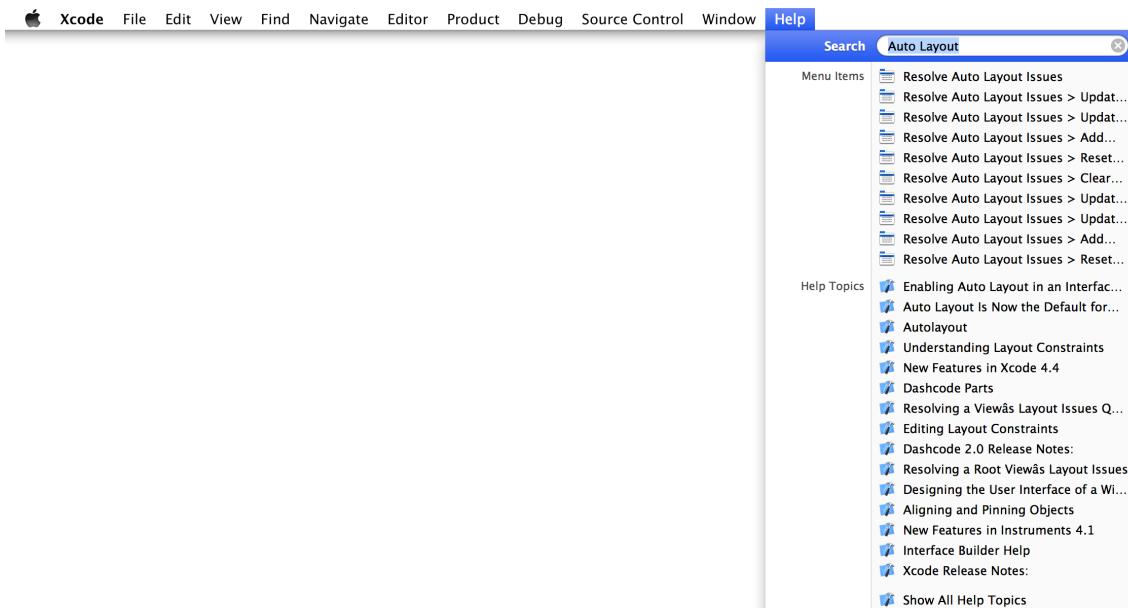
## 通过关联帮助文章来获得 Xcode 指导

若要在使用 Xcode 时获得帮助，请阅读帮助文章。帮助文章会说明如何完成常见的任务，如创建新类、在 Interface Builder 中设置自定类，以及使用 Auto Layout 解决问题。

根据您尝试执行的操作内容，可以按住 Control 键点按 Xcode 中的 UI 元素来访问某些帮助文章。查找关联菜单中上一次输入的内容（此图像中为“Project Navigator Help”）。



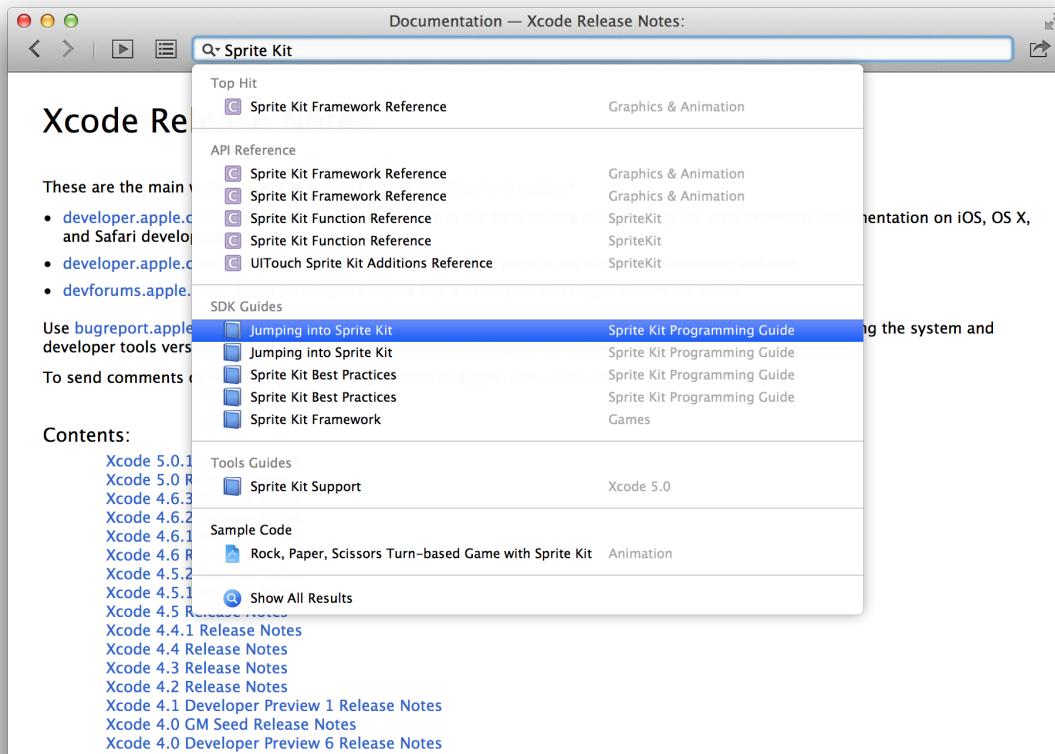
如果查找的是通用帮助，或者任务相关的元素不支持按住 Control 键点按来访问信息，那么您还可以在“Xcode Help”菜单中搜索关联帮助。



## 使用指南来获得通用概述和概念概述

请阅读概念指南中的相关章节，了解新技术或深入理解框架中不同的类之间配合工作的方式。大多数 Cocoa 框架和技术都有相应的编程指南，如《*Sprite Kit Programming Guide*》（Sprite Kit 编程指南）、《*Programming with Objective-C*》（使用 Objective-C 编程），以及《*Location and Maps Programming Guide*》（位置和地图编程指南）。

在 Xcode 中，可以使用文稿显示窗口来查看此类文稿，访问方法是选取“Help”>“Documentation and API Reference”（Option–Command–问号）。只需键入技术名称即可，如“Sprite Kit”。

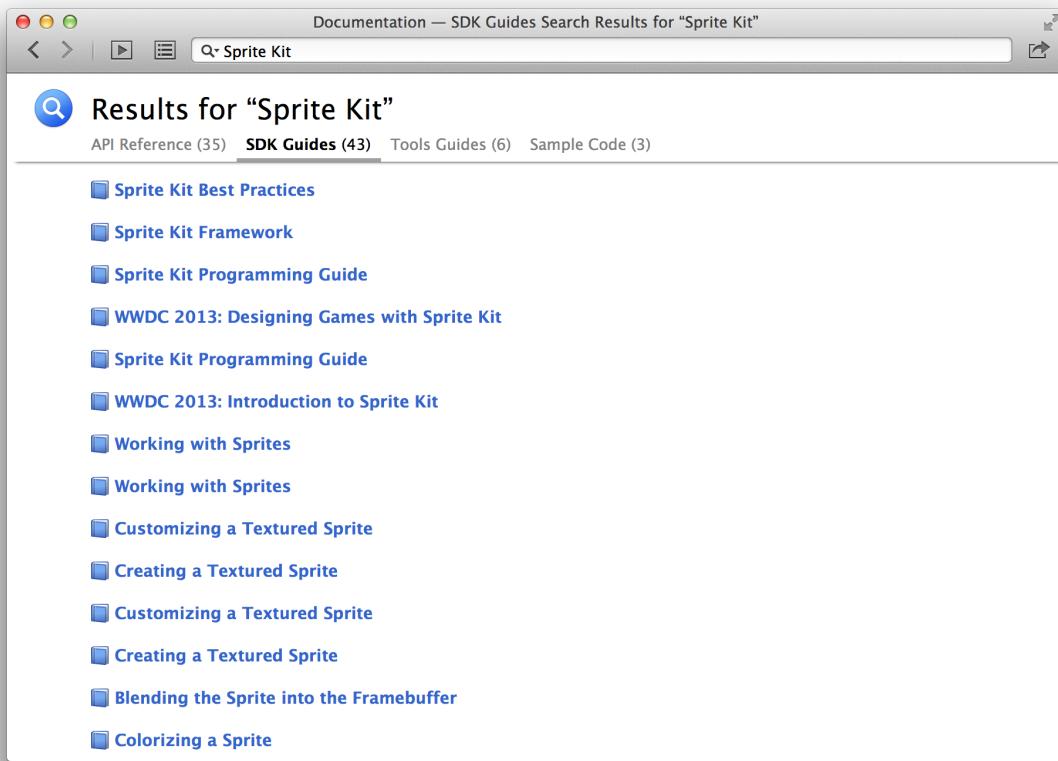


结果将有序显示，对您写代码最有帮助的内容排在前列。也就是说 API 参考条目将首先被列出，接着是 SDK 和“工具”指南。

查找信息

使用 API 参考来获得类信息

如果弹出式列表中没有显示合适的结果，请选取“Show All Results”以显示可供过滤的完整结果。



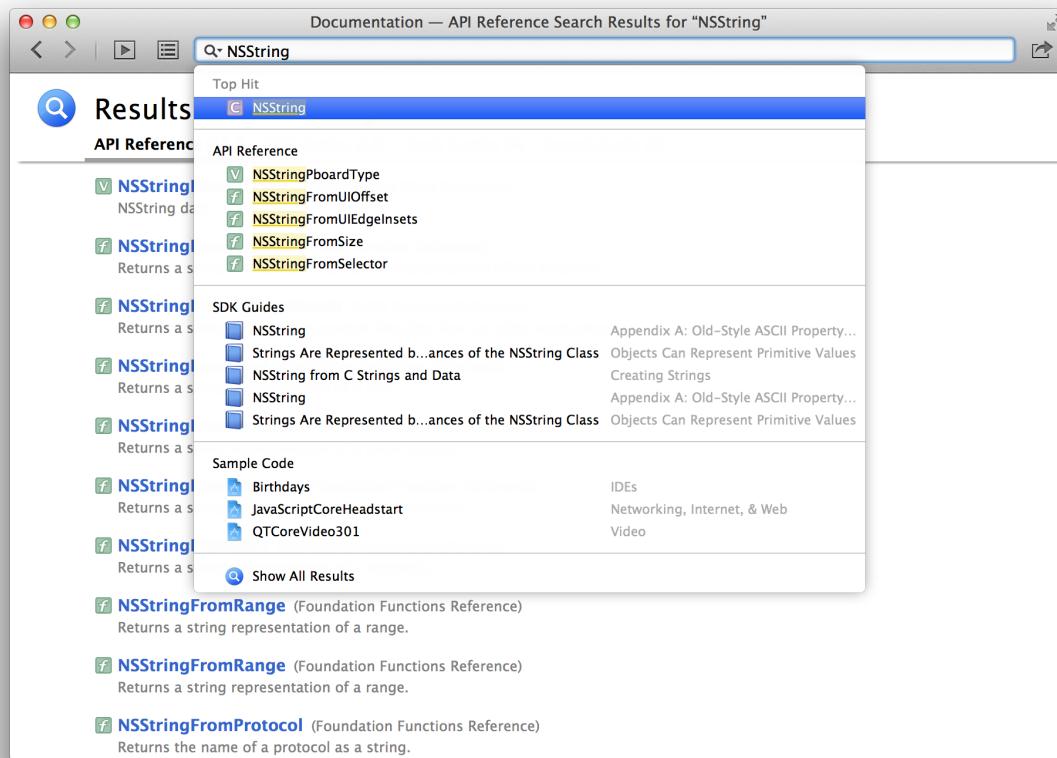
## 使用 API 参考来获得类信息

阅读完指南，了解了技术的方方面面，并开始应用该技术编写代码时，您就会发现还需要进一步了解各个类能做些什么，或者需要掌握如何正确地调用特定方法。**API** 参考文稿提供了这些信息。

查找信息

使用 API 参考来获得类信息

例如，要了解前面教程中使用的 `NSString` 类的更多信息，只需在文稿显示窗口的搜索栏中键入类的名称即可。



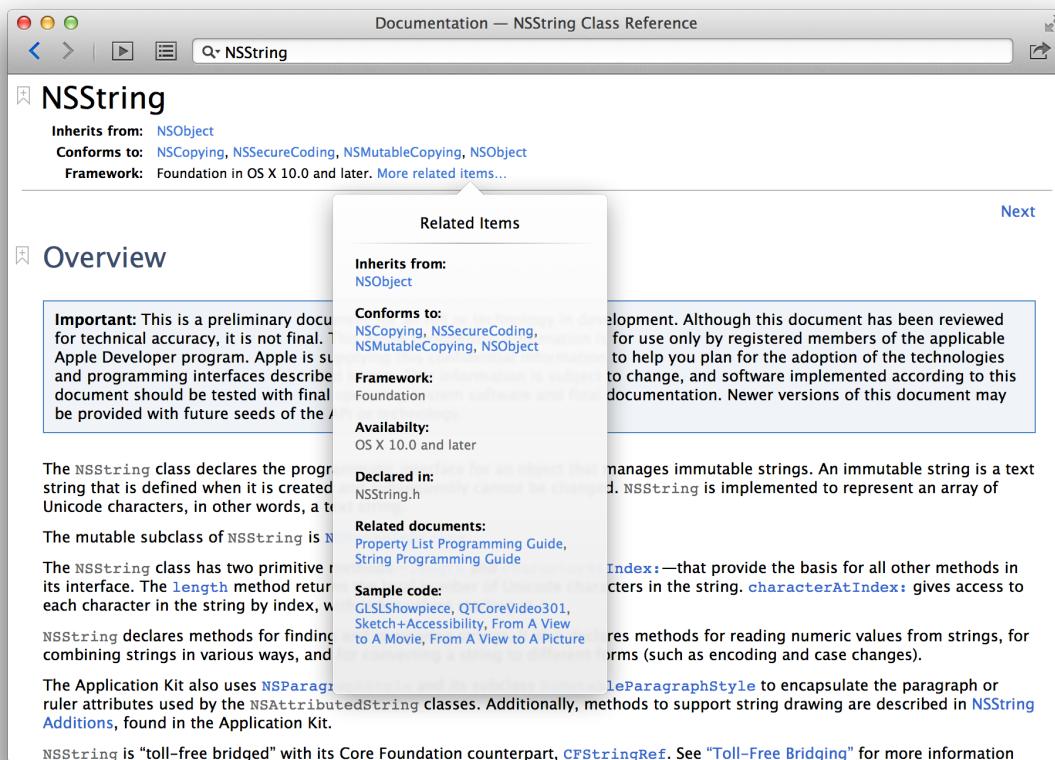
查找信息

使用 API 参考来获得类信息

最常点选的内容通常是所需内容；按下 Return 键进行选择，然后就可以看见该类的 API 参考。

The screenshot shows the 'Documentation — NSString Class Reference' window in Xcode. The title bar says 'Documentation — NSString Class Reference'. The search bar contains 'NSString'. The main content area is titled 'NSString' with a sidebar icon. Below the title, it says 'Inherits from: NSObject', 'Conforms to: NSCopying, NSSecureCoding, NSMutableCopying, NSObject', and 'Framework: Foundation in OS X 10.0 and later. More related items...'. A 'Next' button is visible at the bottom right of the content area. The 'Overview' section is expanded, showing a note in a box: 'Important: This is a preliminary document for an API or technology in development. Although this document has been reviewed for technical accuracy, it is not final. This Apple confidential information is for use only by registered members of the applicable Apple Developer program. Apple is supplying this confidential information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology.' Below the note, the text continues: 'The NSString class declares the programmatic interface for an object that manages immutable strings. An immutable string is a text string that is defined when it is created and subsequently cannot be changed. NSString is implemented to represent an array of Unicode characters, in other words, a text string. The mutable subclass of NSString is NSMutableString. The NSString class has two primitive methods—`length` and `characterAtIndex:`—that provide the basis for all other methods in its interface. The `length` method returns the total number of Unicode characters in the string. `characterAtIndex:` gives access to each character in the string by index, with index values starting at 0. NSString declares methods for finding and comparing strings. It also declares methods for reading numeric values from strings, for combining strings in various ways, and for converting a string to different forms (such as encoding and case changes). The Application Kit also uses `NSParagraphStyle` and its subclass `NSMutableParagraphStyle` to encapsulate the paragraph or ruler attributes used by the `NSAttributedString` classes. Additionally, methods to support string drawing are described in `NSString Additions`, found in the Application Kit. NSString is "toll-free bridged" with its Core Foundation counterpart, `CFStringRef`. See "Toll-Free Bridging" for more information'

API参考文稿可让您快速访问各个类的相关信息，包括类所提供的方法列表、父类，以及所采用的协议。点按“More related items”，查看关于类的通用信息。



“Related Items”弹出式窗口还会显示相关指南的列表。例如，对于 NSString 来说，如果您更想要了解概念概述，而不是深究参考资料，请阅读《*String Programming Guide*》（String 编程指南）。

查找信息

使用 API 参考来获得类信息

除了描述特定的方法或属性之外，API 参考文稿还会概述类可以执行的所有任务。

The screenshot shows the 'Documentation — NSString Class Reference' window in Xcode. The search bar at the top contains the text 'nsstring'. The main content area displays the class hierarchy and various sections of the NSString class:

- Inherits from:** NSObject
- Conforms to:** NSCopying, NSSecureCoding, NSMutableCopying, NSObject
- Framework:** Foundation in OS X 10.0 and later. [More related items...](#)

The content is organized into sections:

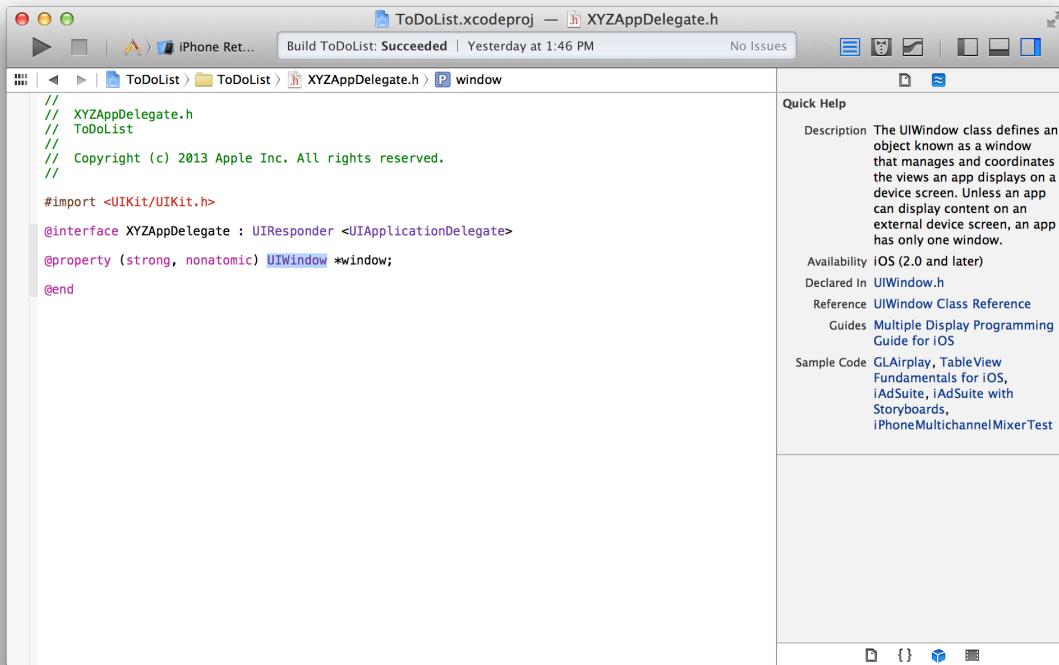
- Combining Strings**:
  - stringByAppendingFormat:
  - stringByAppendingString:
  - stringByPaddingToLength:withString:startingAtIndex:
- Dividing Strings**:
  - componentsSeparatedByString:
  - componentsSeparatedByCharactersInSet:
  - stringByTrimmingCharactersInSet:
  - substringFromIndex:
  - substringWithRange:
  - substringToIndex:
- Finding Characters and Substrings**:
  - rangeOfCharacterFromSet:
  - rangeOfCharacterFromSet:options:
  - rangeOfCharacterFromSet:options:range:
  - rangeOfString:
  - rangeOfString:options:
  - rangeOfString:options:range:
  - rangeOfString:options:range:locale:
  - enumerateLinesUsingBlock:
  - enumerateSubstringsInRange:options:usingBlock:
- Replacing Substrings**:
  - stringByReplacingOccurrencesOfString:withString:

查找信息

使用 Quick Help 来获得关联的源代码信息

## 使用 Quick Help 来获得关联的源代码信息

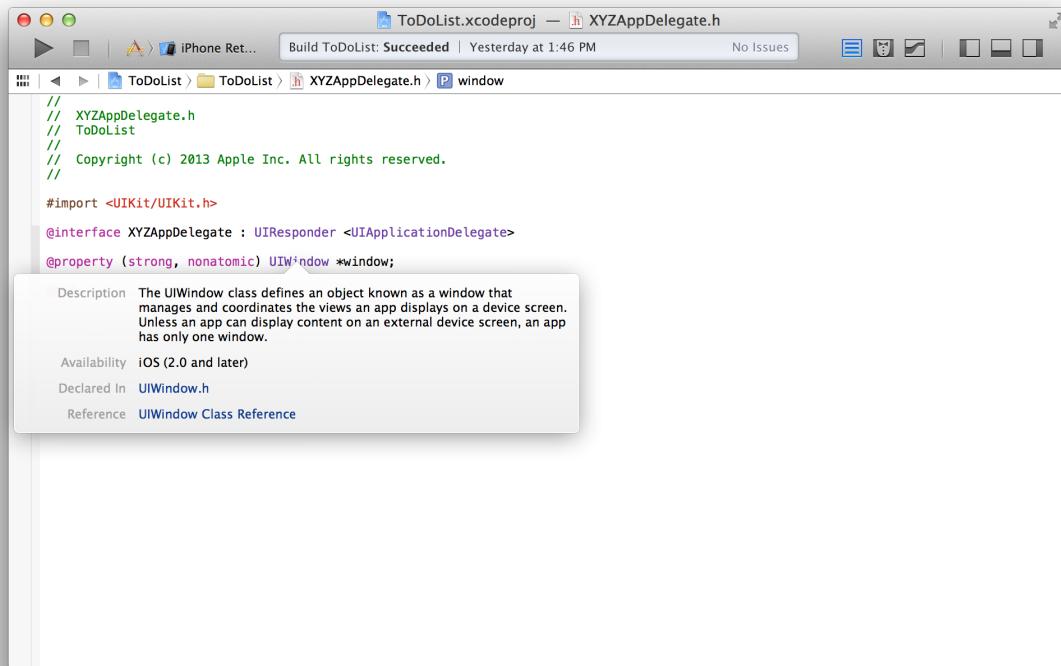
在源代码编辑器中写代码时，可在“Quick Help”面板中轻松访问 API 参考文稿（选取“View”>“Utilities”>“Show Quick Help Inspector”）。 “Quick Help”面板会在您写代码的过程中不断更新，显示当前正在键入的符号的相关信息。



查找信息

使用 Quick Help 来获得关联的源代码信息

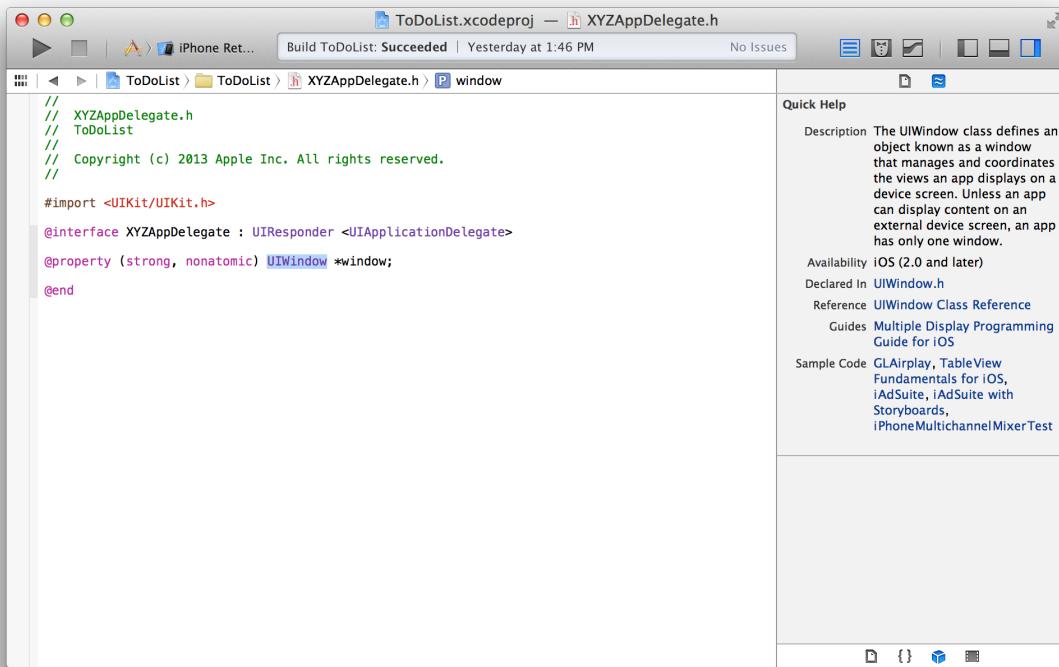
另外，您可以在源代码编辑器中按住 Option 键点按符号，从而显示带有“Quick Help”信息的弹出式窗口。



通过“Quick Help”面板或弹出式窗口，您可以在单独的文稿显示窗口中打开 API 参考，还可以查看包含所点按符号的声明的原始头文件。

## 通过示例代码来查看实际用法

除了书面文稿，您还可以访问示例代码资源库。无论何时在文稿显示窗口中阅读快速帮助或指南和参考，都有一些条目会显示给定技术或类的相关示例代码项目。

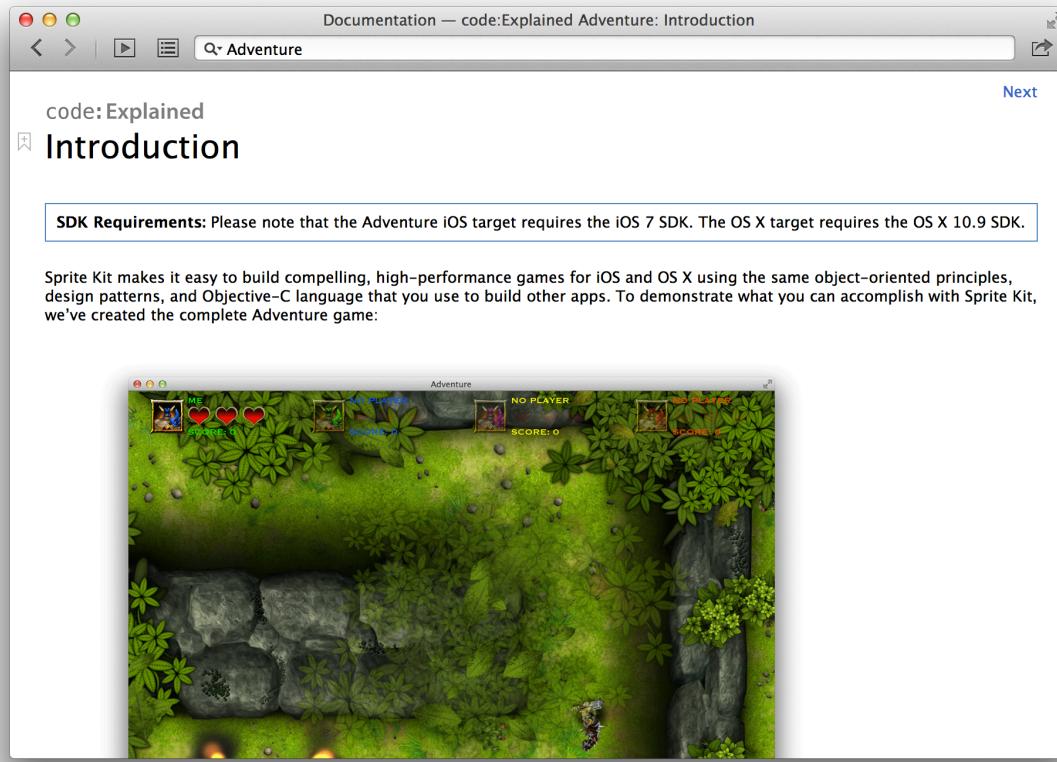


点按任何一个示例代码条目都可以在 Xcode 中下载并打开项目，以便您检查代码。

查找信息

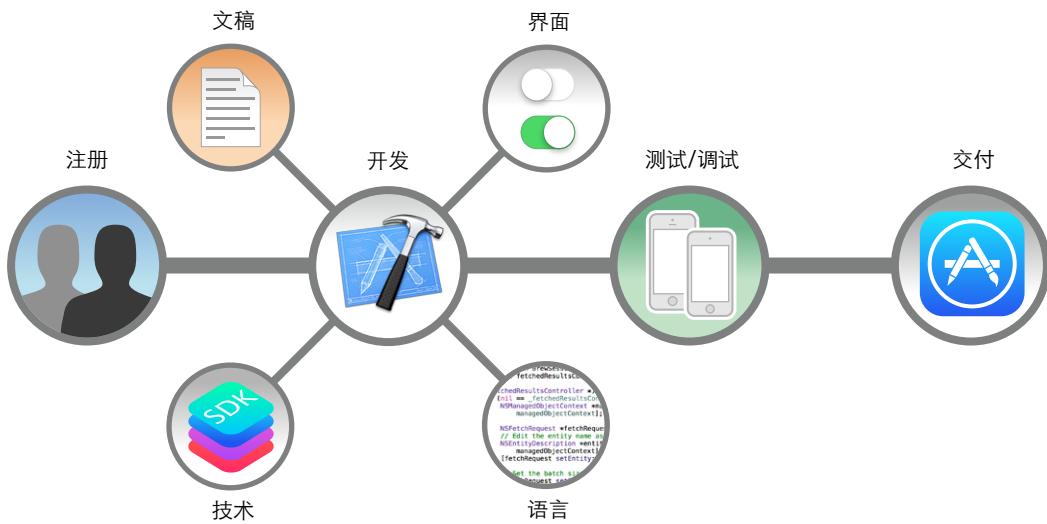
通过示例代码来查看实际用法

除了在整个项目中进行代码注释之外，某些较大的代码示例还有附加文稿。例如，Sprite Kit Adventure项目附加了《*code:Explained Adventure*》（代码：解释冒险）文稿。



# 接下来做什么

在《马上着手开发 iOS 应用程序》中，您学习了 iOS 应用程序开发的基础知识。现在，应该可以开发出您第一个功能完整的应用程序了。虽然将一个简单的概念转变成应用程序，再发布到 App Store 绝不简单，但是您在本文中学到的流程与实践，将帮助您找到正确的方向。



关于接下来做什么，这里有一些提示：

- 注册为开发者。

《App Distribution Guide》（应用程序分发指南）中的“Managing Accounts”（管理帐户）会带您逐步完成注册为 Apple 开发者的过程。

- 学习设计美观的应用程序界面。

《iOS Human Interface Guidelines》（iOS 用户界面指南）会指导您如何让应用程序符合 iOS 用户界面规范。

- 学习语言。

《Programming with Objective-C》（使用 Objective-C 编程）描述了如何使用 Objective-C 程序设计语言定义类、发送消息、封装数据，以及完成各种其他任务。

- 学习开发优秀的应用程序。

《iOS App Programming Guide》（iOS 应用程序编程指南）讲解了在开发 iOS 应用程序时，您必须了解并做到的基本事项。

- 了解可用的技术。

《*iOS Technology Overview*》（iOS 技术概述）介绍了可以在 iOS 应用程序中使用的框架和其他技术。

- 浏览文稿。

“[查找信息](#)（第 109 页）”讲解了如何充分利用所提供的文稿。

- 调试并测试您的应用程序。

《*Xcode Overview*》（Xcode 概述）中的“[Debug Your App](#)”（“调试您的应用程序”）会教您如何在 Xcode 中彻底调试和测试应用程序。

- 发布应用程序。

《*App Distribution Guide*》（应用程序分发指南）会带您逐步完成这些过程：预备测试设备，提交应用程序到 App Store。

## 让 **ToDoList** 应用程序提高一个档次

您刚创建的待办事项列表应用程序得益于多个内建行为。您可以继续体验此应用程序、巩固理解，或者开发一些新东西。如果要继续做待办事项列表应用程序，需要探究以下几个方面：

- 现在退出并重启应用程序时，待办事项列表会消失。不妨探索一下让列表不消失的方法。
- 您为应用程序中的所有控制都使用了默认外观。**UIKit**包括了许多控制自定外观的功能。不妨使用该技术体验一下不同的用户界面选项。
- 您已经可以让用户将项目添加到列表，并将项目标记为已完成，但用户还无法删除项目。表格视图具有支持编辑功能的内建行为，包括删除和重新排列行，您可以考虑集成到应用程序中。

随着 iOS 应用程序开发的继续，您将发现还有非常多的概念和技术需要钻研，包括本地化、可访问性和外观自定。就从您感兴趣的方向开始吧。记得边学习概念边学以致用。遇到有趣的新技术、框架或设计模式时，不妨写一个小应用程序来测试它，要大胆尝试。

开发应用程序涉及到方方面面，您可能会对此感到畏惧，但按本文稿中讲述的“分而治之”方式来做的话，会发现很快就能发布您的第一个应用程序了。在 App Store 中发布应用程序后，您可以不断地加入更多功能。只要不断创新，就能引起客户的注意，让他们对您的下一个杰作翘首企盼。

# 文稿修订历史

此表描述对《马上着手开发 iOS 应用程序 (*Start Developing iOS Apps Today*)》所做的修订。

日期	备注
2014-07-15	修订摘要：“修改后的教程分为多个部分，描述了创建 iOS 应用程序所需的基本技能。”



Apple Inc.  
Copyright © 2014 Apple Inc.  
保留一切权利。

事先未经 **Apple Inc.** 书面许可，此出版物的任何部分均不得以任何形式或通过任何方式（包括机械、电子、影印、记录或其他方式）复制、储存在检索系统中或传播，但以下情况例外：任何人在此被授权将文稿存储在单台电脑上以仅供个人使用并打印文稿的副本以用于个人用途，只要文稿包括 **Apple** 的版权声明。

**Apple** 标志是 **Apple Inc.** 的商标。

事先未经 **Apple** 书面同意，将“键盘”**Apple** 标志 (Option-Shift-K) 用于商业用途可能会违反美国联邦和州法律，并可能被指控侵犯商标权和进行不公平竞争。

本文稿不对所描述的任何技术授予明示或暗示的许可。**Apple** 保留与本文稿中所描述的技术相关的所有知识产权。本文稿只可用来帮助应用程序开发者仅为贴有 **Apple** 标签的电脑开发应用程序。

我们已尽力确保本文稿中的信息准确。**Apple** 对印刷错误概不负责。

Apple Inc.  
1 Infinite Loop  
Cupertino, CA 95014  
408-996-1010

Apple, the Apple logo, Cocoa, Cocoa Touch, iPad, iPhone, iPod, iPod touch, Mac, Objective-C, OS X, Quartz, and Xcode are trademarks of Apple Inc., registered in the U.S. and other countries.

Retina is a trademark of Apple Inc.

App Store is a service mark of Apple Inc.

iOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

OpenGL is a registered trademark of Silicon Graphics, Inc.

同时在美国和加拿大出版。

虽然 **Apple** 已检查了本文稿，但 **Apple** 对本文稿及其质量、准确度、适销性、特定用途的适用性不作任何明示或暗示的保证或表示。因此，本文稿按“原样”提供，而读者要承担有关其质量和准确度的整个风险。

在任何情况下，**Apple** 将不对因本文稿中的任何瑕疵或错误而造成的直接、间接、特殊、偶然或必然损失承担责任，即使 **Apple** 已告知这类损失的可能性。

上述保证和补救措施是排他性的，替代所有其他口头或书面以及明示或暗示的保证和补救措施。未经授权，任何 **Apple** 经销商、代理商或雇员都不得对此保证进行任何修改、扩充或添加。

某些州不允许排除或限制对某些偶然或必然损失的暗示保证或责任，因此上述限制或排除可能不适用于您。此保证给予您特定的法定权利，因州而异，您可能还拥有其他权利。