Apple.Inc 2012/12/5 发布

iOS 应用程序开发中文手册

开发 iOS 应用程序既有趣又回报丰厚，如果您是一位新手，自然想知道从哪里入手。本路线图提供了 iOS 应用程序开发的绝佳起点。在 Mac 电脑上，您可以创建在 iPad、iPhone 和 iPod touch 上运行的 iOS 应用程序。遵循本路线图以了解如何取得开发工具，理解主要概念及最佳实践，并学会查找更多信息。

继续遵循此路线图，您将使用到 Xcode 和 iOS SDK（Apple 提供的开发工具）。您将了解 Objective-C（驱动所有 iOS 应用程序和框架的程序设计语言）的编程基础知识，并将探索 Cocoa Touch 框架。您将创建一个简单的 iOS 应用程序，并学会在设备上进行测试。最后，您还会学到如何将应用程序提交到 App Store。

路线图中的每个页面，都介绍一个新主题，并链接到一篇或多篇有关该主题的简短文章。路线图只是提供基本的入门知识，最后一页“接下来做什么”包含您接下来应该阅读的文稿链接。完成了路线图，您就具备了进一步了解各个主题的能力，并且掌握到着手开发 iOS 应用程序的知识、工具和技能。

如果您是 Mac 开发者：您其实已经掌握了不少开发 iOS 应用程序的知识。可是，尽管 iOS 和 OS X 使用相同的开发工具和开发语言，两者仍然存在着一些重大的差异（您将在路线图的学习过程中发现这点）。有关这些平台差异的全面描述，请参阅 *iOS Technology Overview*（iOS 技术概述）中的“Migrating from Cocoa”（从 Cocoa 迁移）。

设置

您在开发应用程序时，会使用到 iOS 软件开发套件 (SDK) 以及 Xcode，即 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及其他许多功能，为您开发完美的 iPhone、iPod touch 和 iPad 应用程序，提供了所需要的全部资源。大多数应用程序开发工具集中显示在一个窗口中，Xcode 称之为**工作区窗口**。在此窗口内，您可以顺畅地从代码编写转换到代码调试，再到用户界面设计。iOS SDK 扩展了 Xcode 工具集，包含 iOS 专用的工具、编译器和框架。

开始之前：

1. 下载最新版本的 Xcode。

   在 Mac 上打开 **Mac App Store** 应用程序，搜索 Xcode，然后点按“免费”按钮下载 Xcode。您下载的 Xcode 已包含 iOS SDK。（Mac OS X v10.7 以及更高版本已经预装 Mac App Store 应用程序。如果您使用的是较早版本的 Mac OS X，则需要升级。）

2. 加入 iOS Developer Program 成为 Apple 开发者。

   您无需加入该计划也可编写应用程序并在 iOS Simulator 中测试。但是，您只有加入该计划，才能在设备上测试与分发应用程序。加入该计划后，您还可以全权访问 iOS Dev Center 和 iOS Provisioning Portal。如果您现在加入，就可以执行路线图中的所有操作步骤，包括在设备上测试应用程序。

马上开始

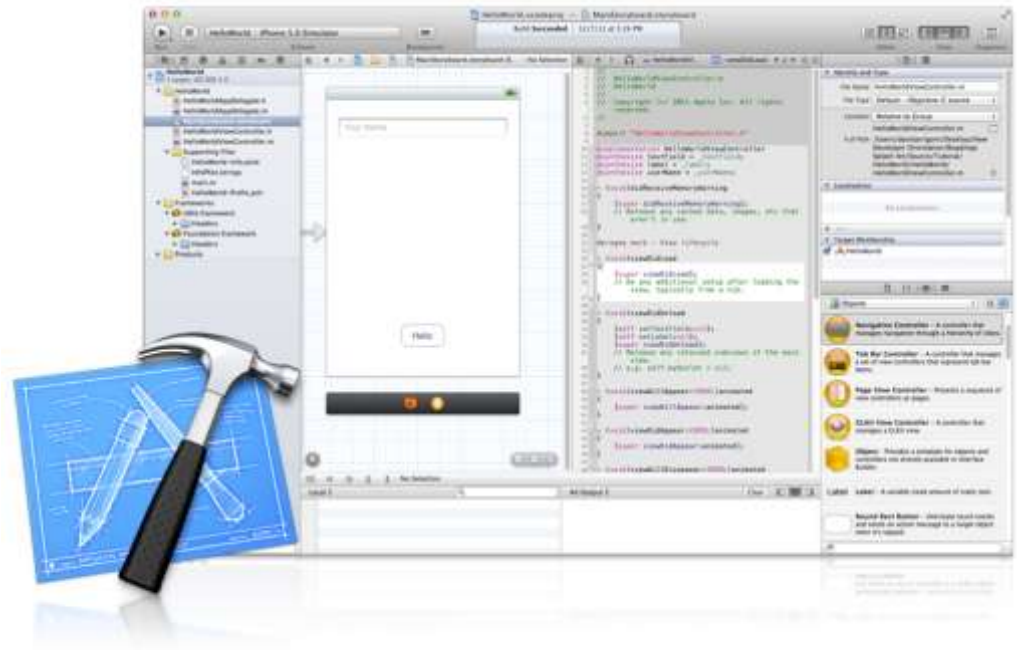
开发优秀的 iOS 应用程序，需要大量的学习和实践。不过，有了这些工具和 iOS SDK，开发一个简单可用的程序并非难事。**您的首个 iOS 应用程序**教程，介绍了这些工具、基本设计模式和应用程序开发过程。通过这个教程，您将学习创建一个能接收用户文本输入并能将文本显示在屏幕上的应用程序。您还将学到如何在 Mac 上的 iOS Simulator 中运行这一程序。本教程中的简单步骤引入了一些简练实用的概念，将在今后的程序开发中不断地用到。

您的首个 iOS 应用程序是路线图中最长的文章，但请不要仓促地完成它。它为您在后面将详细学习的概念奠定了基础。其他每篇文章，都是围绕当中某个概念再深入说明。

立即学习本教程：您的首个 iOS 应用程序告诉您，如何在 iOS 开发环境中，创建一个简单的应用程序。完成教程后，可返回本页继续学习**马上着手开发 iOS 应用程序**。

工具

Xcode 提供整套管理开发工作流程的工具——从创建应用程序、设计用户界面，到测试、优化，并将其提交到 App Store。您可以自定 Xcode，来符合您的工作风格，让您专注于手头的任务。



您的首个 iOS 应用程序向您展示了如何创建新项目、添加用户界面元素，以及编辑源代码。创建应用程序后，您可使用 Xcode 来测试和调试源代码、分析和改进应用程序的性能、执行源代码控制操作、归档并将应用程序提交到 App Store 等等。

立即阅读此文章：在 Xcode 中管理工作流程向您展示 Xcode 用于 iOS 应用程序开发的重要功能。

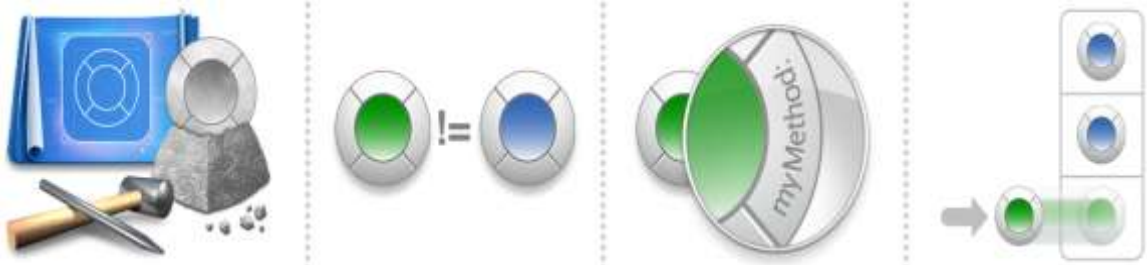
程序设计语言

Objective-C 是一种简洁的、面向对象的程序设计语言，所有 iOS 应用程序都由它来驱动。您需要编写 Objective-C 代码来创建应用程序，同时您需要懂得该语言，才能使用大多数的框架。尽管您可以使用其他编程语言来开发，但不使用 Objective-C 就无法生成 iOS 应用程序。

Objective-C 是一种简单的程序设计语言，其语法和规范简单易学。如果您有其他面向对象程序设计语言（例如 Java 或 C++）的编程经验，那么它对您来说，将更容易上手。如果您是 C 语言程序员，您会发现熟悉面向对象编程和 Objective-C 后，应用程序的设计和修改变得更加容易。

基本任务

现在，您学到的 Objective-C 的知识，足可用来阅读和编写基本代码，您可以开始以对象的方式来考虑问题了。如同思考现实世界中的对象一样，您应该思考一个对象包含什么内容，可以用来做什么，以及如何与其他对象关联。



要创建 iOS 应用程序，您需要了解如何创建对象、比较对象、查询对象的相关信息，以及访问数据集（如数组）中的对象。这些任务在 iOS 应用程序中很常见。掌握这些技能后，您就可以编写更复杂的 Objective-C 代码了。

框架

应用程序由您编写的代码和 Apple 提供的框架组成。框架包含方法资源库，供您的应用程序调用。多个应用程序可同时访问一个框架资源库。

您开发的应用程序都会链接多种框架。您可以通过框架的应用编程接口 (API) 来利用框架。API（已发布在头文件中）指定可用的类、数据结构和协议。Apple 编写的框架，预计了您可能想要实现的基本功能。使用框架既省时省力，又可确保代码高效、安全。系统框架是访问底层硬件的唯一途径。

立即阅读以下文章：

- 研究主要框架描述应用程序开发中最常用的框架。它还简要阐述了 OS X API 和 iOS API 之间的一些异同点。
- 将代码与框架整合描述 Objective-C 框架中的方法种类，并解释如何将应用程序代码与框架代码整合。尽管 OS X API 和 iOS API 之间存在不同之处，但应用程序和框架之间的关系通常相同。

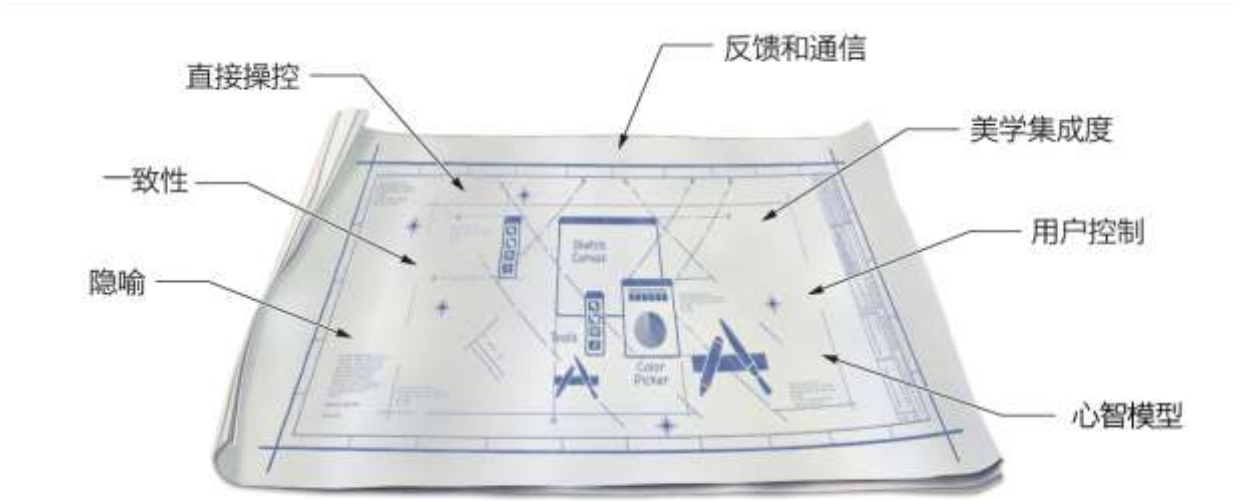
设计模式

- 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具体需求。

- 不管是创建哪种类型的应用程序，您都应该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。
- **立即阅读此文章：**采用设计模式使您的应用程序合理化描述关键的设计模式，并解释如何在应用程序开发中使用这些模式。这些设计模式在 OS X 和 iOS 中基本相同。

用户界面设计

- 仅仅创建一个能用的应用程序是不够的，用户期望的 iOS 应用程序是直观易用、交互性强和引人入胜的。在设计应用程序时，从选取的功能到应用程序响应手势的方式，每个方面都要考虑用户体验。

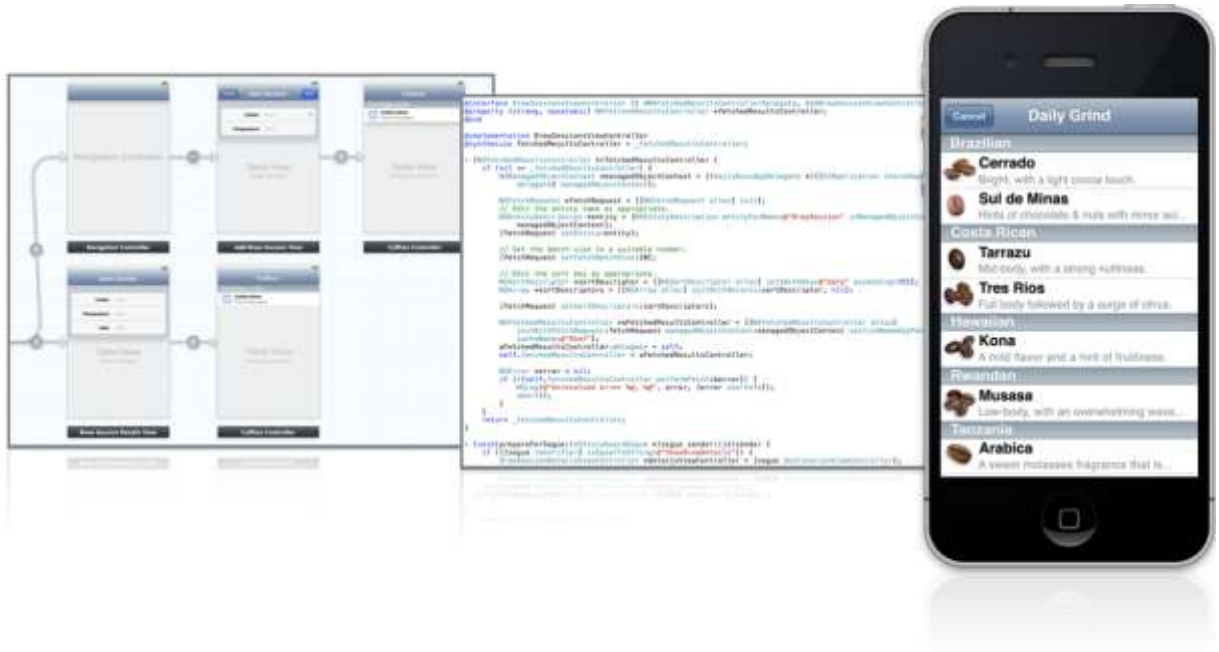


- 用户或许不知道用户界面的设计原则，但是应用程序有没有遵循指南，他们是可以分辨出来的。请遵循 *iOS Human Interface Guidelines*（iOS 用户界面指南）中清楚说明的原则和约定，为您的产品设计最好的用户界面和用户体验。

- **立即阅读此文章：**从用户角度进行设计阐述了如何创建具有卓越用户体验的应用程序。本文稿介绍编程时需遵循的指南，但不涉及如何在代码中实现设计。大多数 iOS 指南，都与 OS X 的不同，因为移动设备上的应用程序，在本质上是不同的。

应用程序设计

在开始编写第一行代码之前，您应该做出一些必要的设计决策。应用程序的用途和功能，应该尽可能的具体。选取应用程序将使用的数据模型种类。决定应用程序的用户界面风格，例如，是应该遵循主从复合模式 (master-detail pattern) 还是实用工具应用程序 (utility app) 的模式？您要应用程序通用吗？也就是说，在 iPad、iPhone 和 iPod touch 上均可运行该应用程序吗？诸如此类的设计决策有助于构建应用程序的架构。



但在根据架构进行开发之前，请务必熟悉 Cocoa Touch 框架。毕竟，应用程序并不会凭空出现——您使用框架构建对象，然后才能生成应用程序。框架对象既是应用程序的基础构造，也是数据模型的组成部分，同时还向用户传递出应用程序的独特体验，应好好掌握它。

设计优良的应用程序，会吸引用户，而且具有适当和有用的功能。例如，应用程序可能适当而有效地使用动画；如果它允许用户选择对象，可能会允许用户拷贝、剪切和粘贴；又或者，它会根据不同的语言，来呈现不同的文本、图像和声音。

立即阅读以下文章：

- 用心设计您的应用程序提出了一些您需要自己解决的初始设计问题。它还有助于理清如何将设计决策付诸实践。
- 了解您的应用程序的核心对象叙述作为所有 iOS 应用程序的一部分的重要框架对象，以及这些对象如何协作。
- 将您的应用程序国际化带您逐步完成将 HelloWorld 应用程序国际化和本地化的过程。

App Store

- 您目前所读到的信息主要描述在 Xcode 中创建应用程序的方法。然而，要在 App Store 上发布应用程序，您还需要进一步了解相关内容。

- 要制作出很好的应用程序，您需要在开发过程中用真实设备测试，而不只是借助 Simulator。要在基于 iOS 的设备上运行应用程序，您需要注册测试设备，创建证书来授权开发者给应用程序签名，以及创建应用程序 ID 来标识应用程序。
- 测试和改进应用程序后，您需要通过 iTunes Connect 提交应用程序。您的应用程序必须通过 App Store 审查人员审批后才能发布。

- **立即阅读此文章：**准备提交到 App Store 描述为 App Store 开发应用程序的管理方面的任务。

查找信息

- 开发应用程序时，需要能轻易得到详细的技术信息。Xcode 可让您在编程时轻松查到所需信息。

- Xcode Quick Help 显示简明的参考信息，不会分散您对正在编辑的文件的注意力。请点按符号、界面对象或生成设置，以查看更多信息。按住 Control 键点按 Xcode 本身的各个区域，获得任务导向的说明，介绍如何执行常见操作。每篇帮助文章都提供逐步的指导，并且通常包括视频或插图来进一步说明。

Xcode 中的“Documentation”管理器，提供深入的编程指南、指导教程、示例代码、开发者工具使用手册、详细的框架 API 参考，以及由 Apple 工程师讲解的视频演示。“Documentation”管理器提供了一个一体化视图，可在其中搜索和浏览所有 Apple 开发者文稿。iOS Developer Library 也在网上提供。

- **立即阅读此文章：**快速查找文稿图解如何使用这些重要的文稿资源。

接下来做什么

祝贺您，您已学完**马上着手开发 iOS 应用程序**中的所有内容，并阅读了所要求的关联文章，现在可以进行 iOS

应用程序的开发了。但是，要成为高效多产的 iOS 开发者，首先，应该加入 iOS Developer Program。然后，您就可以从 iOS Developer Library 下载示例应用程序项目，并从中了解各个方面是如何配合工作的，从而深化已学到的知识和技能。

阅读以下文稿来增进您的 iOS 应用程序开发知识

以下文稿是任何 iOS 应用程序开发者都必不可少的读物：

- *iOS Technology Overview*（iOS 技术概述）介绍可在 iOS 应用程序中使用的框架和其他技术。
- *iOS Human Interface Guidelines*（iOS 用户界面指南）教您如何让您的应用程序符合 iOS 用户界面规范。
- *Developing for the App Store*（为 App Store 开发）带您逐步完成这些过程：开发应用程序，预备测试设备，提交应用程序到 App Store。
- *Programming with Objective-C*（使用 Objective-C 编程）描述如何使用 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。
- *iOS App Programming Guide*（iOS 应用程序编程指南）讲解在开发 iOS 应用程序时，您必须要了解并做到的基本事情。

阅读以下教程来探索 iOS 应用程序开发

学完这些教程以取得应用程序开发的其他方面的经验：

- *App Store Submission Tutorial*（向 App Store 提交应用程序教程）向您讲解预备设备、提交应用程序到 App Store 的过程。
- *Your Second iOS App: Storyboards*（您的第二个 iOS 应用程序：串联图）向您讲解如何使用串联图，实现主从复合应用程序。
- *Your Third iOS App: iCloud*（您的第三个 iOS 应用程序：iCloud）教您如何将 iCloud 集成到基于文稿的应用程序。

关于创建您的首个 iOS 应用程序

您的首个 iOS 应用程序介绍 iOS 应用程序开发的“三 T”：

- **工具 (Tools)**。如何使用 Xcode 创建和管理项目。
- **技术 (Technologies)**。如何创建一个响应用户输入的应用程序。
- **技巧 (Techniques)**。如何利用一些基础设计模式——所有 iOS 应用程序开发的基础。

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：

如上图所示，您的应用程序有三个主要的用户界面元素：

- 文本栏（用于用户输入信息）
- 标签（用于显示信息）
- 按钮（让应用程序在标签中显示信息）

运行编写完成的应用程序时，点按文本栏会调出系统提供的键盘。使用键盘键入您的姓名后，点按“Done”键将键盘隐藏，然后点按“Hello”按钮，即可在文本栏和按钮之间的标签中看到字符串“Hello，**您的姓名**!”。

稍后在**马上着手开发 iOS 应用程序**中，您将会学习另一个教程**将您的应用程序国际化**，为您在此教程中创建的应用程序添加中文本地化。

如果您有计算机编程的基础知识，特别是面向对象编程和 Objective-C 程序设计语言的知识，您将能更好地理解本教程。如果您以前从未使用过 Objective-C，也不必担心本教程中的代码难以理解。您学完**马上着手开发 iOS 应用程序**之后，就能更好地理解这些代码了。

注：尽管本教程显示的是 iPhone 用户界面，但使用的工具和技巧与开发 iPad 应用程序所用到的完全相同；所以即使您打算只开发用于 iPad 的程序，您仍然可以将本教程当作入门指南。

立即开始

要创建本教程中的 iOS 应用程序，您需要 Xcode 4.3 或更高版本。Xcode 是 Apple 的集成开发环境（又称 IDE），用于 iOS 和 Mac OS X 的开发。在 Mac 上安装 Xcode，也会同时安装了 iOS SDK，它包含 iOS 平台

的编程接口。

创建和测试新项目

要着手开发应用程序，请创建一个新 Xcode 项目。

▶

创建新项目

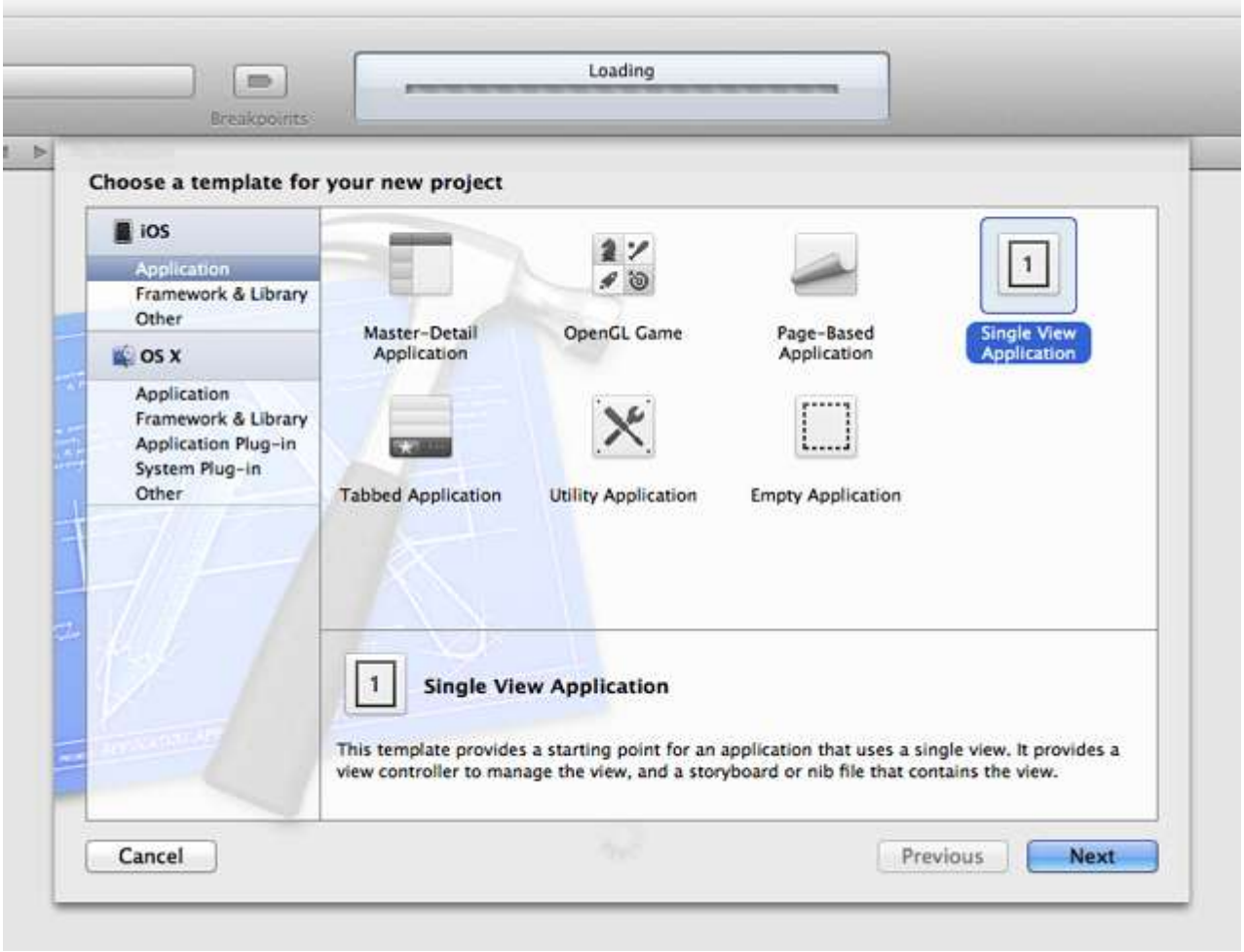
1. 打开 Xcode（默认位置在“/应用程序”中）。

   如果从未在 Xcode 中创建或打开过项目，您应该会看到一个与下图类似的“Welcome to Xcode”窗口：

   如果曾在 Xcode 中创建或打开过项目，您会看到一个项目窗口，而不是“Welcome to Xcode”窗口。

2. 在“Welcome to Xcode”窗口中，点按“Create a new Xcode project”，或选取“File”>“New”>“New project”。

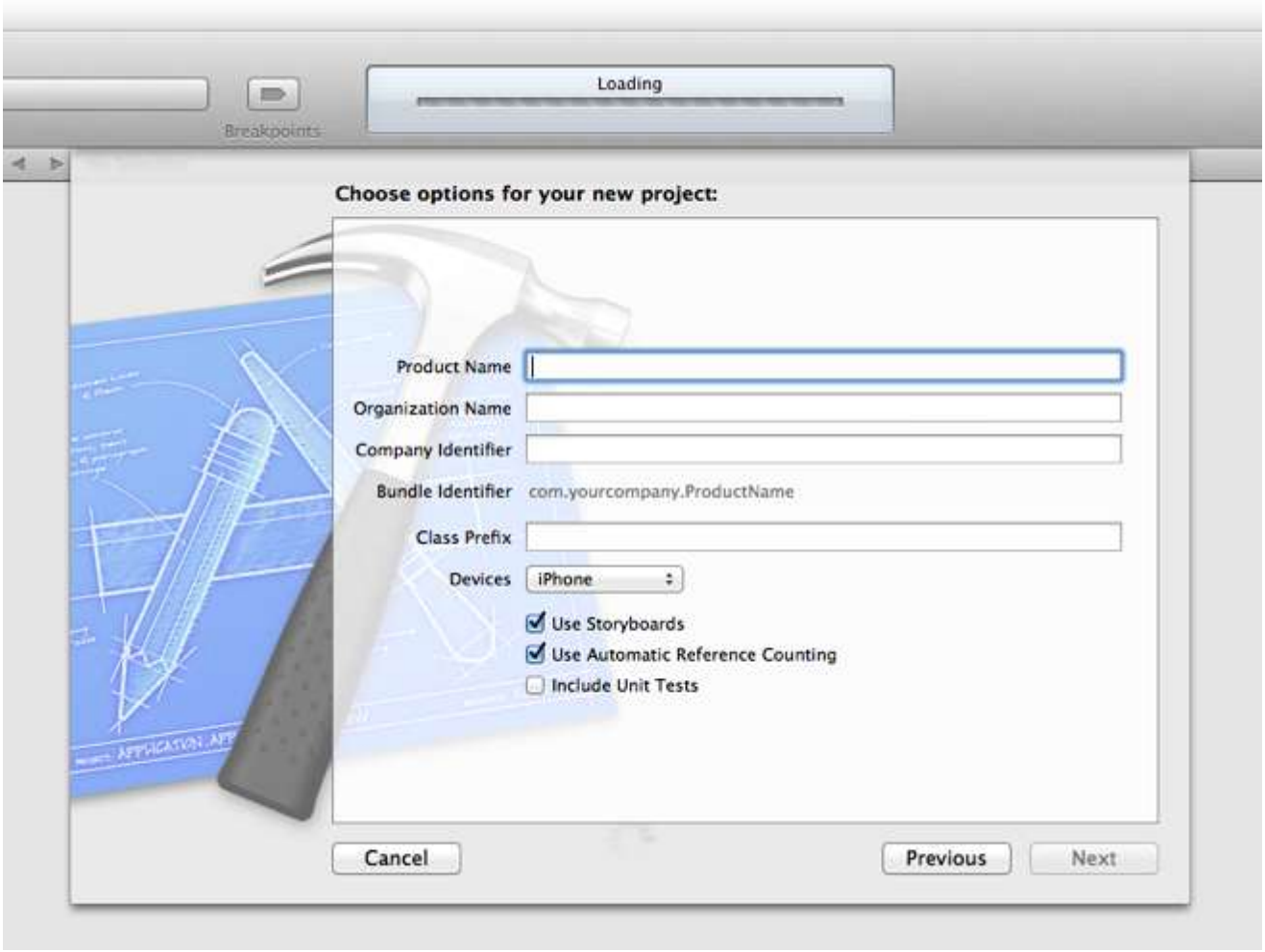
   Xcode 将打开一个新窗口并显示对话框，让您从中选取一个模板。Xcode 内建了一些应用程序模板，您可以使用这些模板来开发常见类型的 iOS 应用程序。例如“Tabbed”模板可以创建与 iTunes 类似的应用程序，“Master-Detail”模板可以创建与“邮件”类似的应用程序。



3. 在对话框左边的 iOS 部分中，选择“Application”。

4. 在对话框的主区域中，选择“Single View Application”，然后点按“Next”。

   一个新对话框会出现，提示您为应用程序命名，并为项目选取附加选项。



5. 填写“Product Name”、“Company Identifier”和“Class Prefix”等栏位。

   您可以使用以下值：

   - Product Name：`HelloWorld`
   - Company Identifier：您的公司标识符（如果有）。如果没有公司标识符，可以使用 `edu.self`。
   - Class Prefix：`HelloWorld`

   注：Xcode 使用输入的产品名称来命名您的项目和应用程序。Xcode 使用类前缀名称来命名为您所创建的类。例如，Xcode 会自动创建一个应用程序委托类，命名为 `HelloWorldAppDelegate`。如果输入不同的值作为类前缀，则应用程序委托类将命名为“**您的类前缀名称** `AppDelegate`”。（您将在“了解应用程序如何启动”中了解更多应用程序委托的信息。）

   简单来说，本教程假设您将产品命名为 `HelloWorld` 并使用 `HelloWorld` 作为类前缀值。

6. 在“Device Family”弹出式菜单中，确定选取 iPhone。

7. 确定选取“Use Storyboard”和“Use Automatic Reference Counting”选项，但不选定“Include Unit Tests”选项。

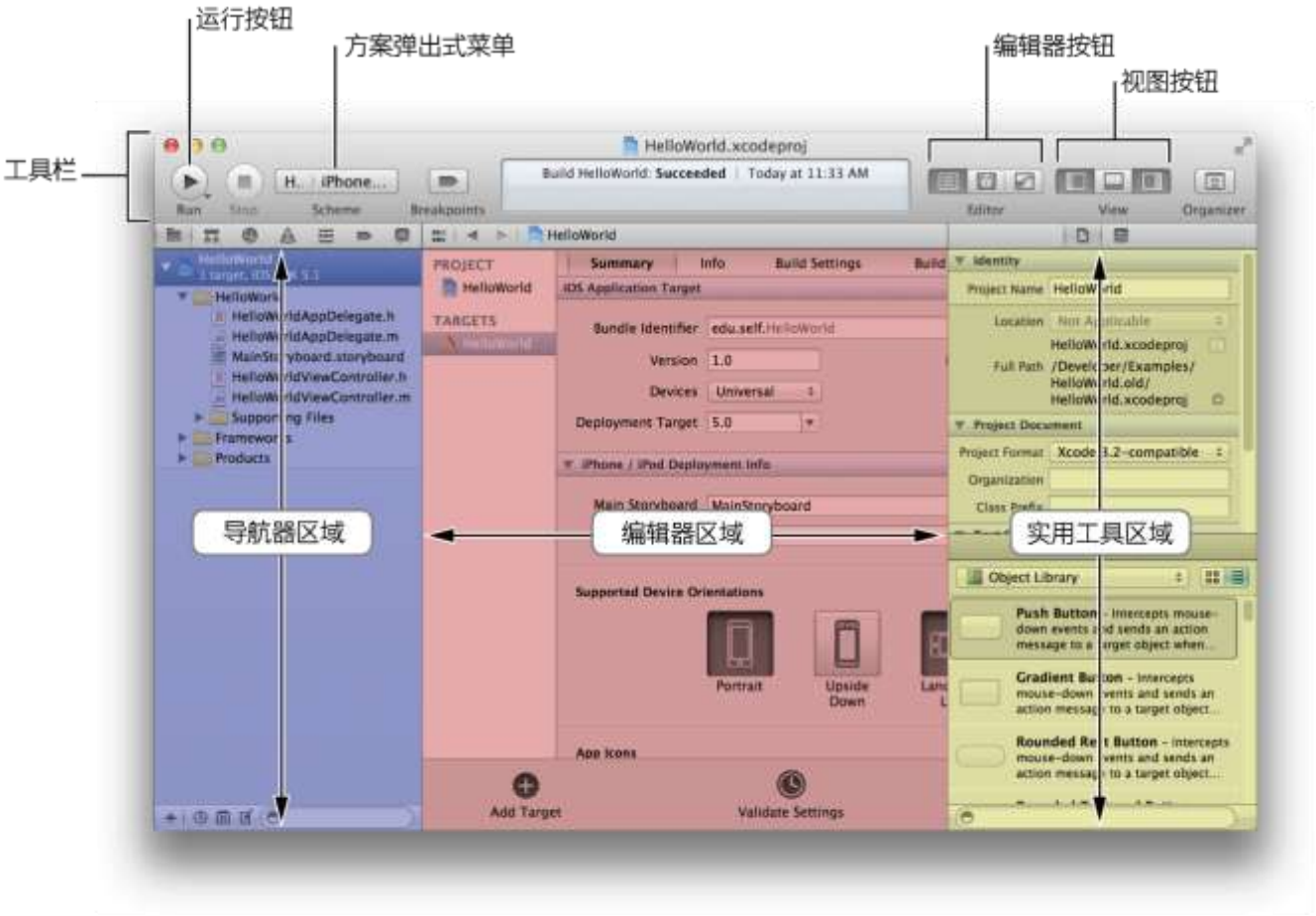
8. 点按“Next”。

此时出现另一个对话框，让您指定项目存储的位置。

9. 为项目指定位置，不要选定“Source Control”选项，然后点按“Create”。

Xcode 在**工作区窗口**中打开新项目，窗口的外观近似于：

请花一些时间来熟悉 Xcode 的工作区窗口。在接下来的整个教程中，您将会用到下面窗口中标识出的按钮和区域。



如果工作区窗口中的实用工具区域已打开（如上图窗口中所示），您可以暂时把它关闭，因为稍后才会用到它。最右边的“View”按钮可控制实用工具区域。实用工具区域可见时，该按钮是这样的：

如有需要，点按最右边的“View”按钮来关闭实用工具区域。

即使您还未编写任何代码，您都可以构建您的应用程序，并在 Simulator（已包含在 Xcode 中）中运行它。顾名思义，Simulator 可模拟应用程序在 iOS 设备上运行，让您初步了解它的外观和行为。

▶

在 Simulator 中运行您的应用程序

1. 确定在 Xcode 工具栏的“Scheme”弹出式菜单中选定“HelloWorld”>“iPhone 6.0 Simulator”选项。

   如果弹出式菜单中该选项未被选定，请把它打开，然后从菜单中选取“iPhone 6.0 Simulator”。

2. 点按 Xcode 工具栏中的“Run”按钮，或选取“Product”>“Run”。

   Xcode 会报告生成的进度。

Xcode 完成生成项目后，Simulator 应该会自动启动。因为您指定的是 iPhone 产品而非 iPad 产品，Simulator 会显示一个看起来像 iPhone 的窗口。在模拟的 iPhone 屏幕上，Simulator 打开您的应用程序，外观应该是这样的：

此刻，您的应用程序还不怎么样：它只显示一个空白的画面。要了解空白画面是如何生成的，您需要了解代码中的对象，以及它们如何紧密协作来启动应用程序。现在，退出 Simulator（选取“iOS Simulator”>“Quit iOS Simulator”；请确定您不是退出 Xcode）。

了解应用程序如何启动

您的项目是基于 Xcode 模板开发的，所以运行应用程序时，大部分基本的应用程序环境已经自动建立好了。例如，Xcode 创建一个应用程序对象（以及其他一些东西）来建立运行循环（运行循环将输入源寄存，并将输入事件传递给应用程序）。该工作大部分是由 `UIApplicationMain` 函数完成的，该函数由 UIKit 框架提供，并且在您的项目的 `main.m` 源文件中自动调用。

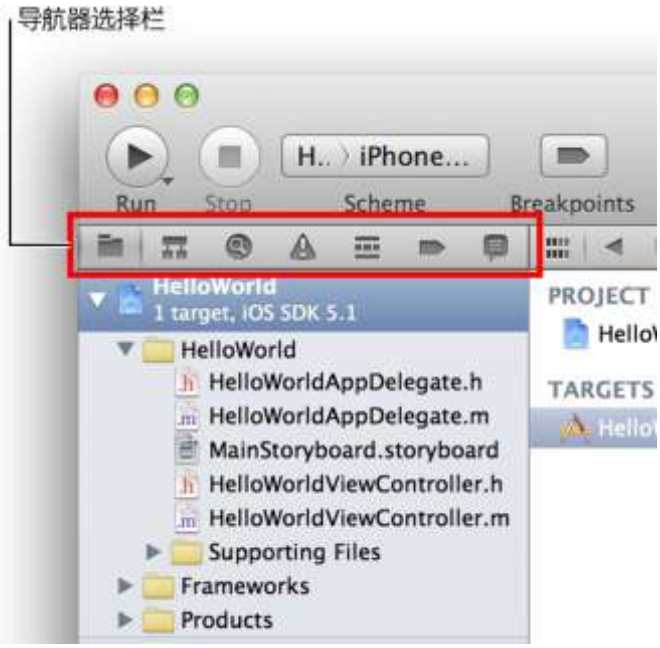
注：UIKit 框架提供应用程序构建和管理其用户界面所需的全部类。UIKit 框架只是 Cocoa Touch 提供的面向对象的众多框架中的一个，而 Cocoa Touch 是所有 iOS 应用程序的应用环境。

▶

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。

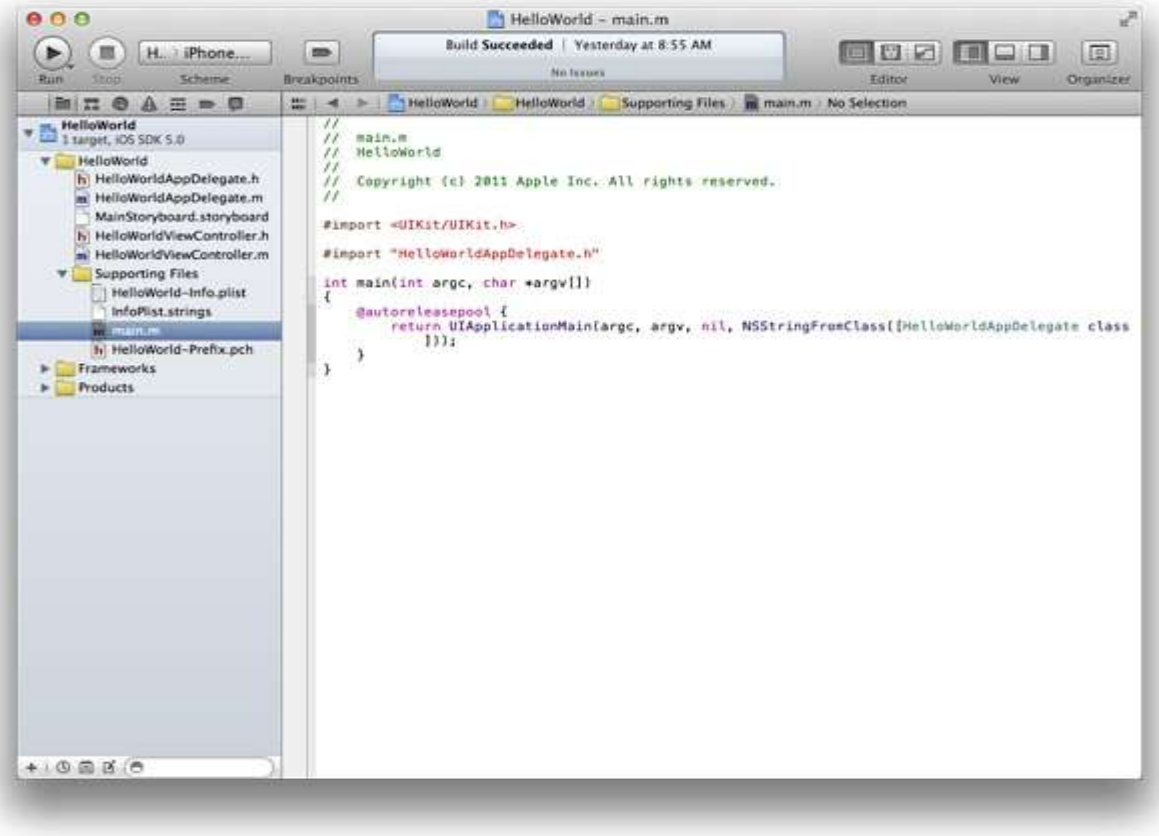
   项目导航器显示项目中的所有文件。如果项目导航器未打开，请点按导航器选择栏最左边的按钮：



2. 点按项目导航器中“Supporting Files”文件夹旁边的展示三角形，打开文件夹。

3. 选择 `main.m`。

   Xcode 在窗口的主编辑器区域打开源文件，外观应该类似这样：



main.m 中的 main 函数调用自动释放池 (autorelease pool) 中的 UIApplicationMain 函数：

```
@autoreleasepool {

    return UIApplicationMain(argc, argv, nil, NSStringFromClass([HelloWorldAppDelegate class]));

}
```

@autoreleasepool 语句支持“自动引用计数 (ARC)”系统。ARC 可自动管理应用程序的对象生命周期，确保对象在需要时一直存在，直到不再需要。

调用 UIApplicationMain 会创建一个 UIApplication 类的实例和一个应用程序委托的实例（在本教程中，应用程序委托是 HelloWorldAppDelegate，由“Single View”模板提供）。**应用程序委托**的主要作用是提供呈现应用程序内容的窗口，在应用程序呈现之前，应用程序委托也执行一些配置任务。（**委托**是一种设计模式，在此模式中，一个对象代表另一个对象，或与另一个对象协调工作。）

在 iOS 应用程序中，**窗口**对象为应用程序的可见内容提供容器，协助将事件传递到应用程序对象，协助应用程序对设备的摆放方向做出响应。窗口本身是不可见的。

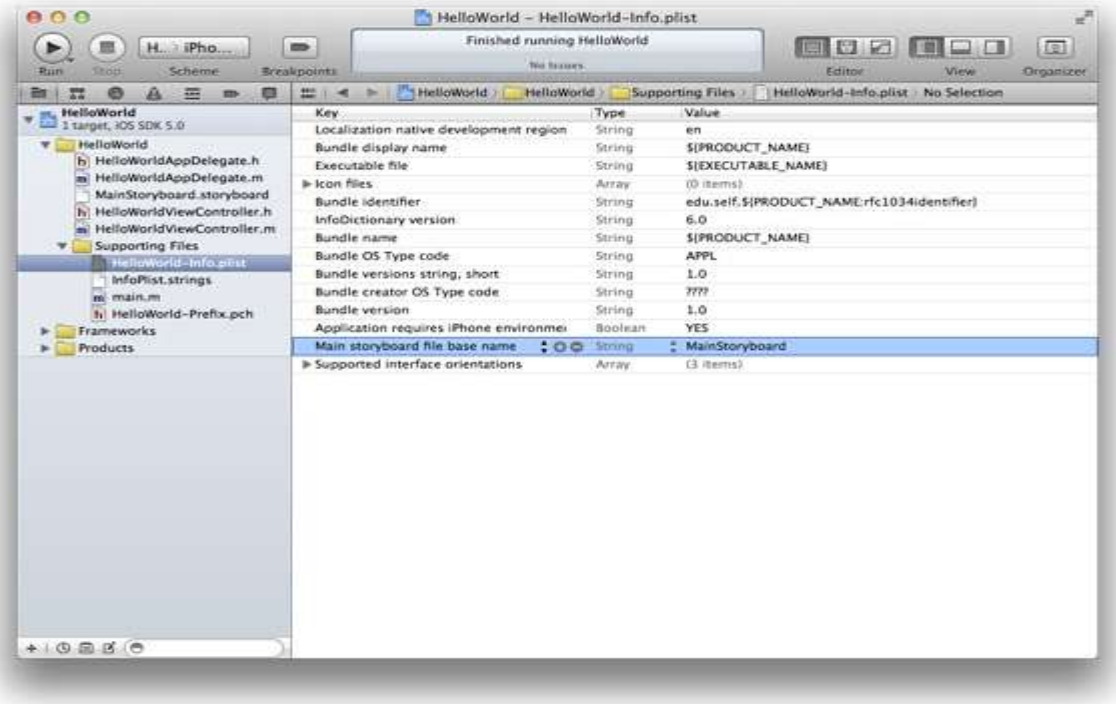
调用 UIApplicationMain 也会扫描应用程序的 Info.plist 文件。Info.plist 文件为信息属性列表，即键和值配对的结构化列表，它包含应用程序的信息，例如名称和图标。

▶

查看属性列表文件

- 在项目导航器的“Supporting Files”文件夹中，选择 HelloWorld-Info.plist。

  Xcode 在窗口的编辑器区域打开 Info.plist 文件，外观应该类似这样：



在本教程中，您不需要查看“Supporting Files”文件夹中的文件，因此可以在项目导航器中关闭此文件夹来避免分散注意力。同样的，点按“Supporting Files”文件夹图标旁边的展示三角形以关闭该文件夹。

因为您已选取在项目中使用串联图，所以 `Info.plist` 文件还包含应用程序对象应该载入的串联图的名称。**串联图**包含对象、转换以及连接的归档，它们定义了应用程序的用户界面。

在“HelloWorld”应用程序中，串联图文件命名为 `MainStoryboard.storyboard`（请注意 `Info.plist` 文件只显示这名称的第一部分）。应用程序启动时，载入 `MainStoryboard.storyboard`，接着根据它对初始视图控制器进行实例化。**视图控制器**是管理区域内容的对象；而**初始视图控制器**是应用程序启动时载入的第一个视图控制器。

“HelloWorld”应用程序仅包含一个视图控制器（具体来说就是 `HelloWorldViewController`）。现在，`HelloWorldViewController` 管理由单视图提供的一个区域的内容。**视图**是一个对象，它在屏幕的矩形区域中绘制内容，并处理由用户触摸屏幕所引起的事件。一个视图也可以包含其他视图，这些视图称为**分视图**。当一个视图添加了一个分视图后，它被称为**父视图**，这个分视图被称为**子视图**。父视图、其子视图以及子视图的子视图（如有的话）形成一个**视图层次**。一个视图控制器只管理一个视图层次。

注：“模型－视图－控制器”(Model-View-Controller, MVC) 设计模式定义了应用程序对象的三种角色，“HelloWorld”应用程序中的视图和视图控制器，体现了其中的两种，而第三种为模型对象。在 MVC 中，模型对象表示数据（例如日历应用程序中的待办事项或绘图程序中的图形），视图对象知道如何显示模型对象所表示的数据，控制器对象充当模型和视图的媒介。在“HelloWorld”应用程序中，模型对象为字符串，用来保存用户输入的名称。现在您不需要了解更多有关 MVC 的信息，但最好开始思考应用程序中的对象如何扮演不同的角色。
在接下来的步骤，您要给由 `HelloWorldViewController` 管理的视图添加三个分视图，以创建视图层次；这三个子视图分别表示文本栏、标签和按钮。

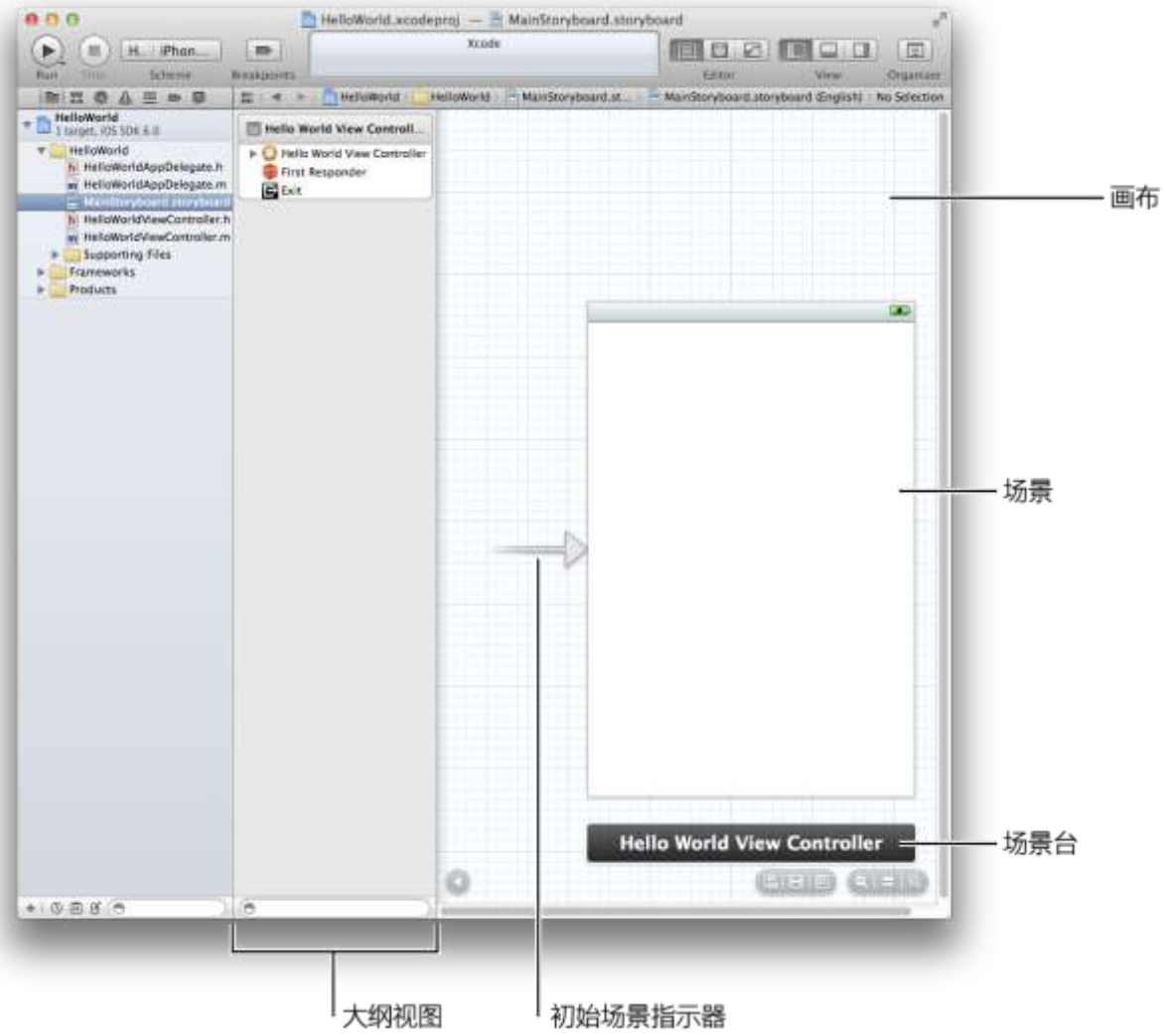
您可以在串联图中看到视图控制器及其视图的模样。

▶
查看串联图

- 在项目导航器中选择 `MainStoryboard.storyboard`。

  Xcode 在编辑器区域打开串联图。（串联图对象后面的区域，即看起来像图纸的区域，称为**画布**。）

打开默认串联图后，工作区窗口看起来应该类似这样：



串联图包括场景和过渡。**场景**代表视图控制器，**过渡**则表示两个场景之间的转换。

因为“Single View”模板提供一个视图控制器，应用程序中的串联图只包含一个场景，没有过渡。画布上指向场景左侧的箭头是“**initial scene indicator**”（初始场景指示器），它标识出应用程序启动时应该首先载入的场景（通常初始的场景就是初始视图控制器）。

在画布上看到的场景称为“Hello World View Controller”，因为它是由 `HelloWorldViewController` 对象来管理的。“Hello World View Controller”场景由一些项目组成，显示在 Xcode **大纲视图**（在画布和项目导航器之间的面板）。现在，视图控制器由以下项目组成：

- 一个第一响应器占位符对象（以橙色立方体表示）。

  “first responder”是一个动态占位符，应用程序运行时，它应该是第一个接收各种事件的对象。这些事件包括以编辑为主的事件（例如轻按文本栏以调出键盘）、运动事件（例如摇晃设备）和操作消息（例如当用户轻触按钮时该按钮发出的消息）等等。本教程不会涉及第一响应器的任何操作。

- 名为 **Exit** 的占位符对象，用于展开序列。

  默认情况下，当用户使子场景消失时，该场景的视图控制器**展开**（或返回）父场景——即转换为该子场景的原来场景。不过，Exit 对象使视图控制器能够展开任意一个场景。

- HelloWorldViewController 对象（以黄色球体内的浅色矩形表示）。

  串联图载入一个场景时，会创建一个视图控制器类的实例来管理该场景。

- 一个视图，列在视图控制器下方（要在大纲视图中显示此视图，您可能要打开“Hello World View Controller”旁边的展示三角形）。

  此视图的白色背景就是在 Simulator 中运行该应用程序时所看到的背景。

注：应用程序的窗口对象在串联图中并未表示出来。
画布上，场景下方的区域称为**场景台**。现在，场景台显示了视图控制器的名称，即“Hello World View Controller”。其他时候，场景台可包含图标，分别代表第一响应器、Exit 占位符对象和视图控制器对象。

小结

在本章中，您使用 Xcode 创建了一个基于“Single View”模板的新项目，生成并运行了该模板定义的默认应用程序。然后您查看了该项目的一些基本组成部分，例如 main.m 源文件、Info.plist 文件以及串联图文件，并了解了应用程序如何启动。您也学习了“模型－视图－控制器”设计模式是如何为应用程序中的对象定义角色的。

在下一章，您将了解有关视图控制器及其视图的更多信息。

检查视图控制器及其视图

正如先前所学习的，一个视图控制器负责管理一个场景，而一个场景代表一个内容区域。在该区域中看到的内容，是在视图控制器的视图中定义的。在本章中，您可以更仔细地查看由 HelloWorldViewController 所管理的场景，并学习如何调整视图的背景颜色。

使用检查器来检查视图控制器

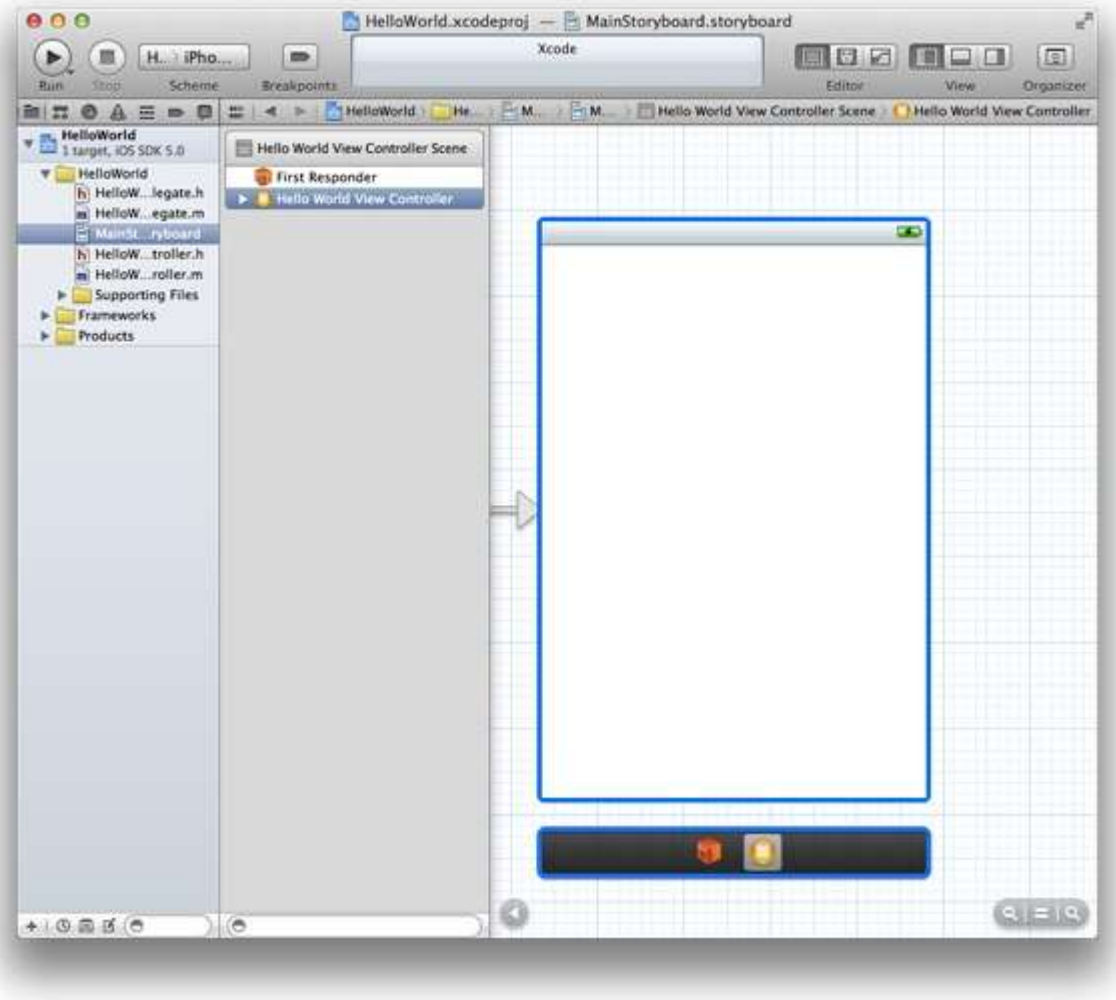
应用程序启动时，载入主串联图文件，然后实例化初始视图控制器。初始视图控制器管理用户打开应用程序时看到的第一个场景。因为“Single View”模板只提供一个视图控制器，该视图控制器自动设定为初始视图控制器。您可以使用 Xcode 检查器来验证视图控制器的状态，并查看关于它的其他信息。

▶

打开检查器

1. 如有需要，点按项目导航器中的 MainStoryboard.storyboard，在画布上显示场景。
2. 在大纲视图中，选择“Hello World View Controller”，列在“first responder”下方。

   您的工作区窗口外观应该像这样：



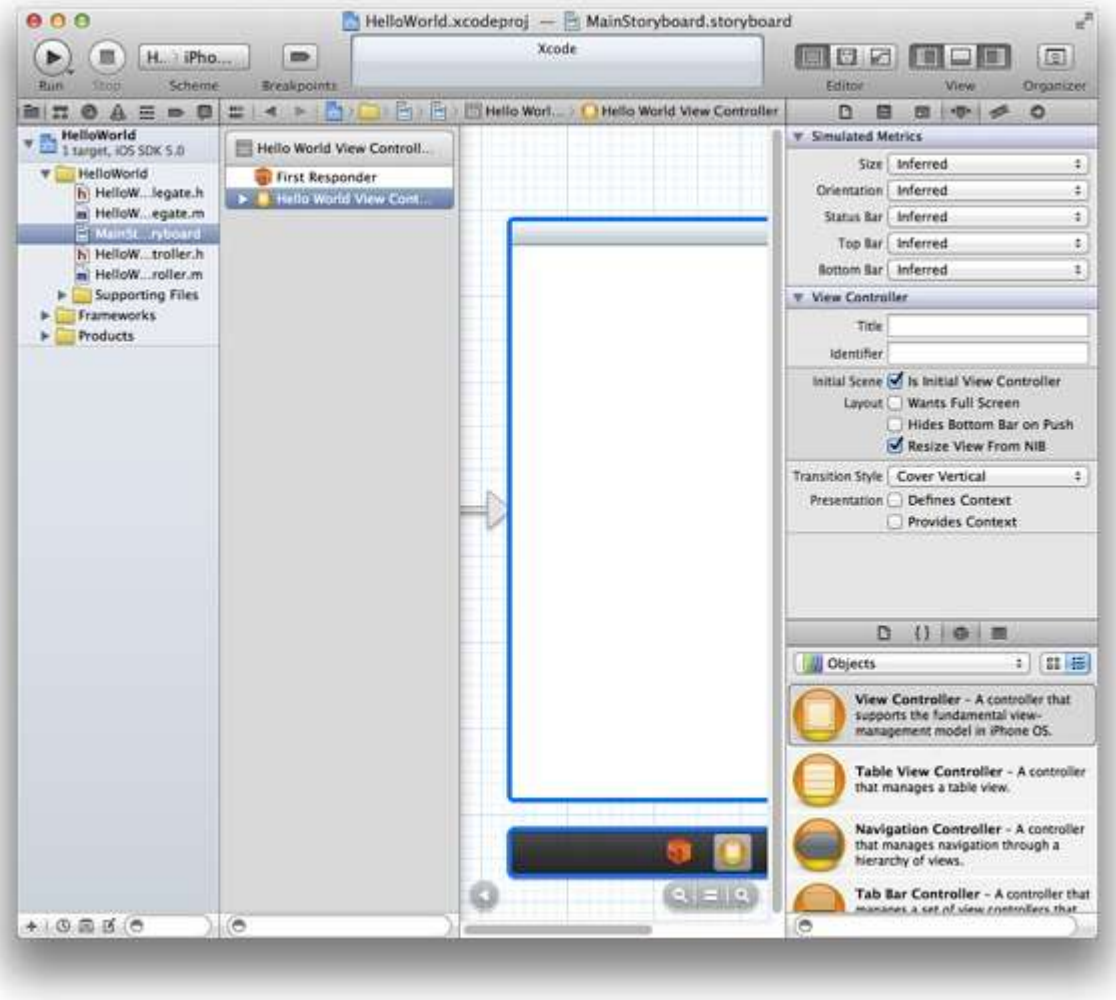
请注意场景和场景台都有蓝色外框，并已选定场景台中的视图控制器对象。

3. 在工具栏中点按最右边的“View”按钮，在窗口右边显示实用工具区域（或者选取“View”>“Utilities”>“Show Utilities”）。

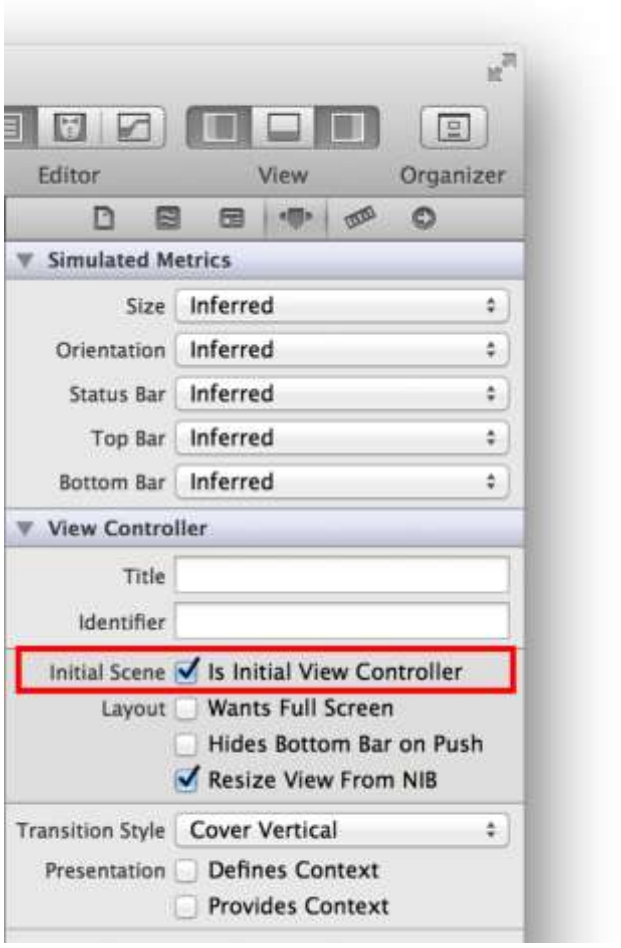
4. 在实用工具区域中点按“Attributes”检查器按钮，打开“Attributes”检查器。

   “Attributes”检查器按钮位于实用工具区域顶部的检查器选择栏中，为从左边起的第四个按钮。

“Attributes”检查器打开时，工作区窗口外观应该像这样（您可能需要调整窗口的大小以便看到所有内容）：



在“Attributes”检查器的“View Controller”部分，可以看到已选定“Initial Scene”选项。



请注意，如果取消选定这个选项，初始场景指示器会从画布中消失。在本教程中，请确定“Initial Scene”选项一直是选定的。

更改视图的背景颜色

在本教程的前面部分，您已了解到视图提供了在 Simulator 中运行应用程序时所看到的白色背景。要确定应用程序工作正常，您可以将视图的背景设定为白色以外的其他颜色，并再次在 Simulator 中运行应用程序来验证新颜色的显示。

在更改视图的背景之前，请确定串联图仍打开在画布上。（如有需要，点按项目导航器中的 `MainStoryboard.storyboard`，在画布上打开串联图。）

▶

设定视图控制器的视图的背景颜色

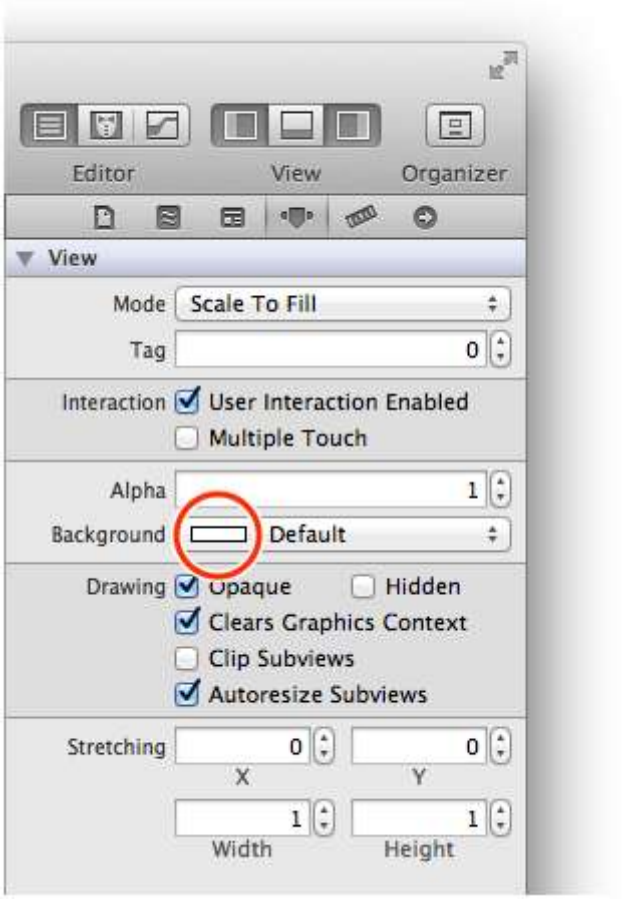
1. 在大纲视图中，点按“Hello World View Controller”旁边的展示三角形（如果还未打开的话），然后选择“View”。

   Xcode 在画布上高亮显示视图区域。

2. 在实用工具区域顶部的检查器选择栏中点按“Attributes”按钮，打开“Attributes”检查器（如果还未打开的话）。

3. 在“Attributes”检查器中，点按“Background”弹出式菜单中的白色矩形，打开“Colors”窗口。

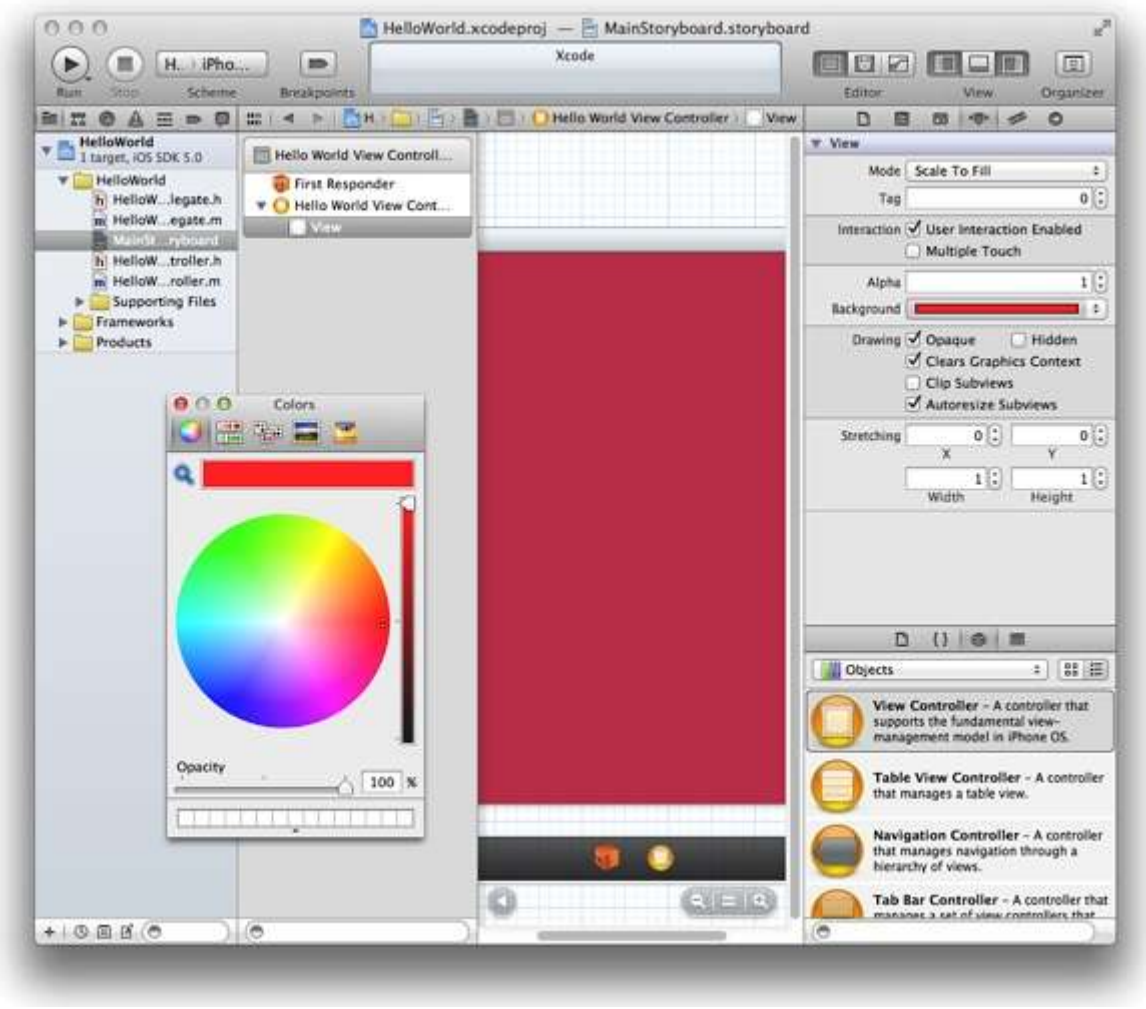
   矩形显示该项目的背景的当前颜色。“Background”弹出式菜单外观像这样：



   注：如果并非点按白色矩形，而是点按“Default”并打开弹出式菜单，请从出现的菜单中选取“Other”。

4. 在“Colors”窗口中，选择白色以外的颜色。

   您的工作区窗口和“Colors”窗口外观应该像这样：



请注意，在您选择视图时 Xcode 高亮显示该视图，所以画布上的颜色可能和“Colors”窗口中的颜色看起来不同。

5. 关闭“Colors”窗口。

点按“Run”按钮或选取“Product”>“Run”，在 Simulator 中测试您的应用程序。请确定 Xcode 工具栏中的“Scheme”弹出式菜单仍然显示“HelloWorld”>“iPhone 5.0 Simulator”。您看到的应该大致是这样的：

提示 在运行应用程序前，可以不必存储您的工作，因为点按“Run”或选取“Product”>“Run”时，Xcode 会自动存储您更改过的文件。

在继续本教程之前，请将视图的背景颜色恢复成白色。

恢复视图的背景颜色

1. 在“Attributes”检查器中，点按箭头打开“Background”弹出式菜单。

   请注意，“Background”弹出式菜单中的矩形已改变成显示在“Colors”窗口中所选取的颜色。如果点按的是有颜色的矩形而不是箭头，“Colors”窗口会重新打开。因为您想重新使用视图原来的背景颜色，从“Background”菜单中选取该颜色，比从“Colors”窗口中找要简单得多。

2. 在“Background”弹出式菜单中，选取“Recently Used Colors”部分中列出的白色方块。

3. 点按“Run”按钮来编译和运行应用程序（并存储所做的更改）。

验证了应用程序重新显示白色背景后，退出 Simulator。

运行应用程序时，Xcode 可能会在工作区窗口的底部打开调试区。本教程不会用到该面板，您可以将其关闭，以腾出更多空间。

关闭调试区

- 点按工具栏中的“Debug View”按钮。

“Debug View”按钮为中间的那个视图按钮，它是这样的：  。

小结

在本章中，您检查了场景，更改和恢复了视图的背景颜色。

在下一章，您将文本栏、标签以及按钮添加到视图中，让用户与应用程序进行交互操作。

配置视图

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。其他对象为高级对象，例如视图控制器和手势识别器。

“Hello World View Controller”场景已经包含了一个视图。现在需要添加一个按钮、一个标签和一个文本栏。然后，在这些元素和视图控制器类之间建立连接，以便元素提供您想要的行为。

添加用户界面元素

将对象库中的用户界面 (UI) 元素拖移到画布上的视图中，来添加用户界面元素。UI 元素添加到视图后，可以适度移动它们的位置和调整大小。

▶

将 UI 元素添加到视图并适当进行布局

1. 如有需要，选择项目导航器中的 `MainStoryboard.storyboard`，在画布上显示“Hello World View Controller”场景。

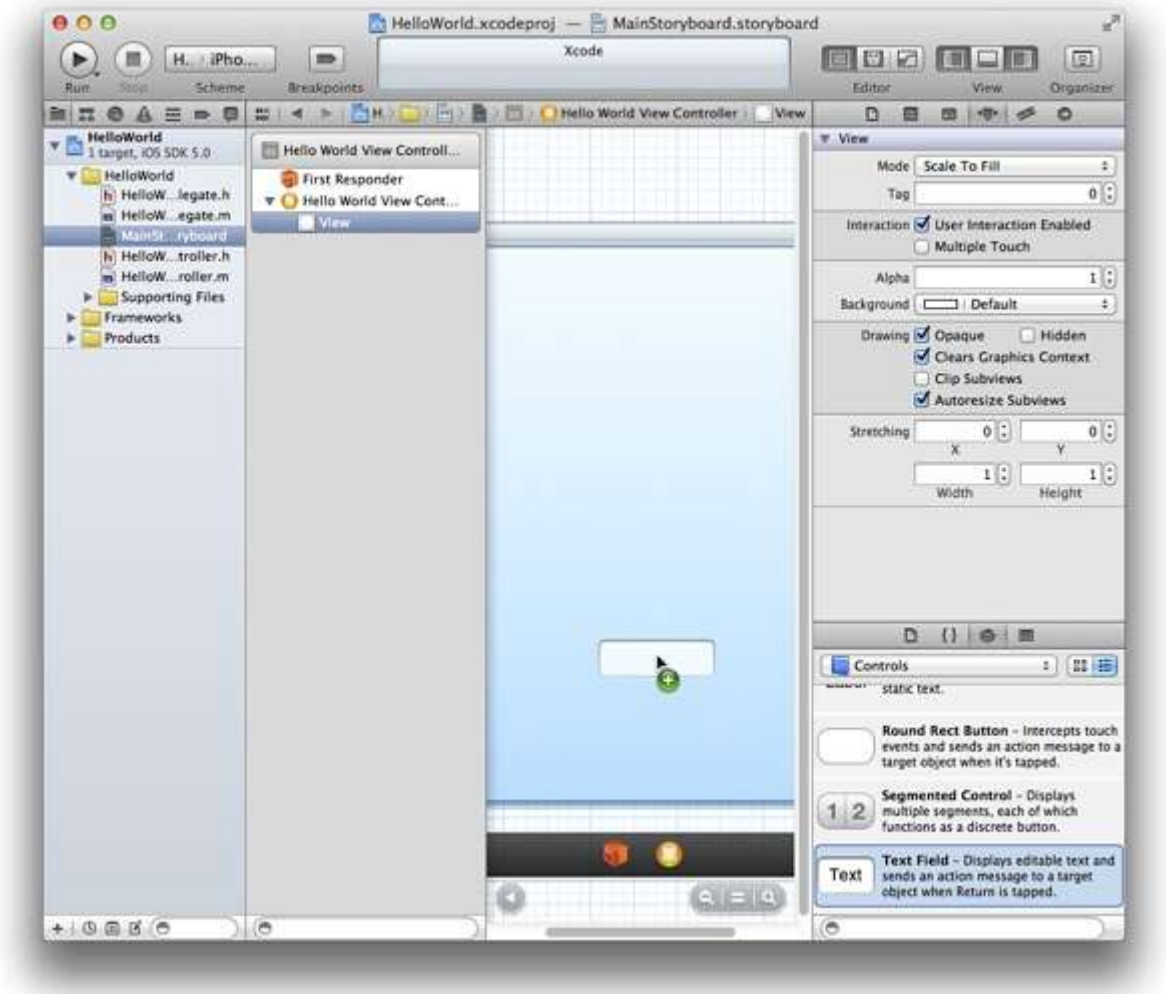
2. 如有需要，打开对象库。

   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，即库选择栏中从左边起的第三个按钮。

3. 在对象库中，从“Objects”弹出式菜单中选取“Controls”。

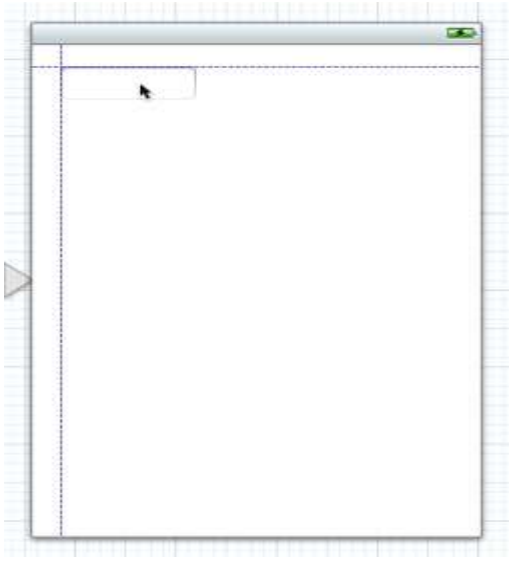
   Xcode 将控制列表显示在弹出式菜单下方。该列表显示每个控制的名称、外观及其功能的简短描述。

4. 从列表中拖一个文本栏、一个圆角矩形按钮和一个标签到视图上，一次一个。



5. 在视图中，拖移文本栏将其放置在视图的左上角附近。

   在移动文本栏或任何其他 UI 元素时，会出现蓝色的虚线（称为**对齐参考线**），有助于将项目与视图的中心和边缘对齐。当您看到视图的左方和上方对齐参考线（如下图所示）时，停止拖移文本栏：

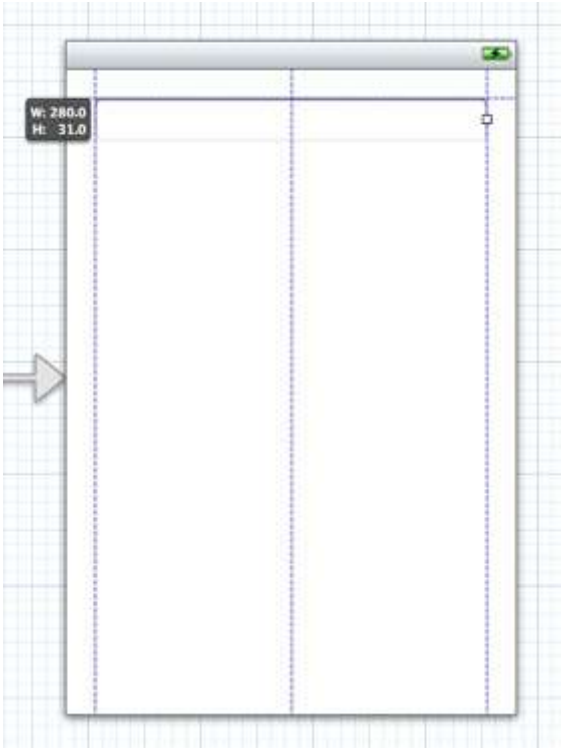


6. 在视图中，准备调整文本栏的大小。

   通过拖移**调整大小控制柄**（显示在元素边框上的小白方块）来调整 UI 元素的大小。一般情况下，在画布上或在大纲视图中选择一个元素，该元素的调整大小控制柄就会显露出来。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；否则请在画布上或在大纲视图中选择它。

7. 拖移文本栏右侧的调整大小控制柄，直到视图最右侧的对齐参考线出现。

   当看到画布像下图这样时，停止调整文本栏大小：



8. 在仍然选定文本栏时，打开“Attributes”检查器（如有需要的话）。

9. 在“Text Field Attributes”检查器顶部附近的“Placeholder”栏中，键入短语 `Your Name`。

   顾名思义，“Placeholder”栏提供的浅灰色文本是为了帮助用户理解能够在文本栏中输入何种信息。在运行的应用程序中，用户只要在文本栏内轻按，占位符文本就会立即消失。

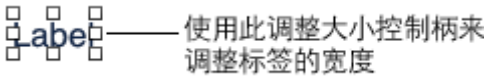
10. 还是在“Text Field Attributes”检查器中，点按中间的“Alignment”按钮，使文本栏的文本居中显示。

    在输入占位符文本和更改对齐设置后，“Text Field Attributes”检查器外观应该像这样：

11. 在视图中，拖移标签到文本栏下方，并使其左边缘和文本栏的左边缘对齐。

12. 拖移标签的右侧调整大小控制柄，使标签与文本栏同宽。

    比起文本栏，标签有更多的调整大小控制柄。这是因为您可以调整标签的高度和宽度，但只能调整文本栏的宽度。现在不是要更改标签的高度，因此不要拖移标签四个角的调整大小控制柄。要拖移的是标签右侧中间的那个调整大小控制柄。

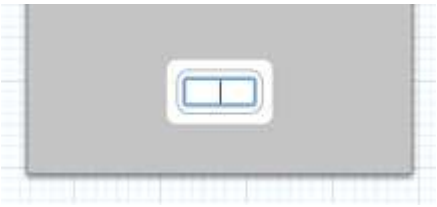


13. 在“Label Attributes”检查器中，点按中间的“Alignment”按钮，使出现在标签中的文本居中显示。

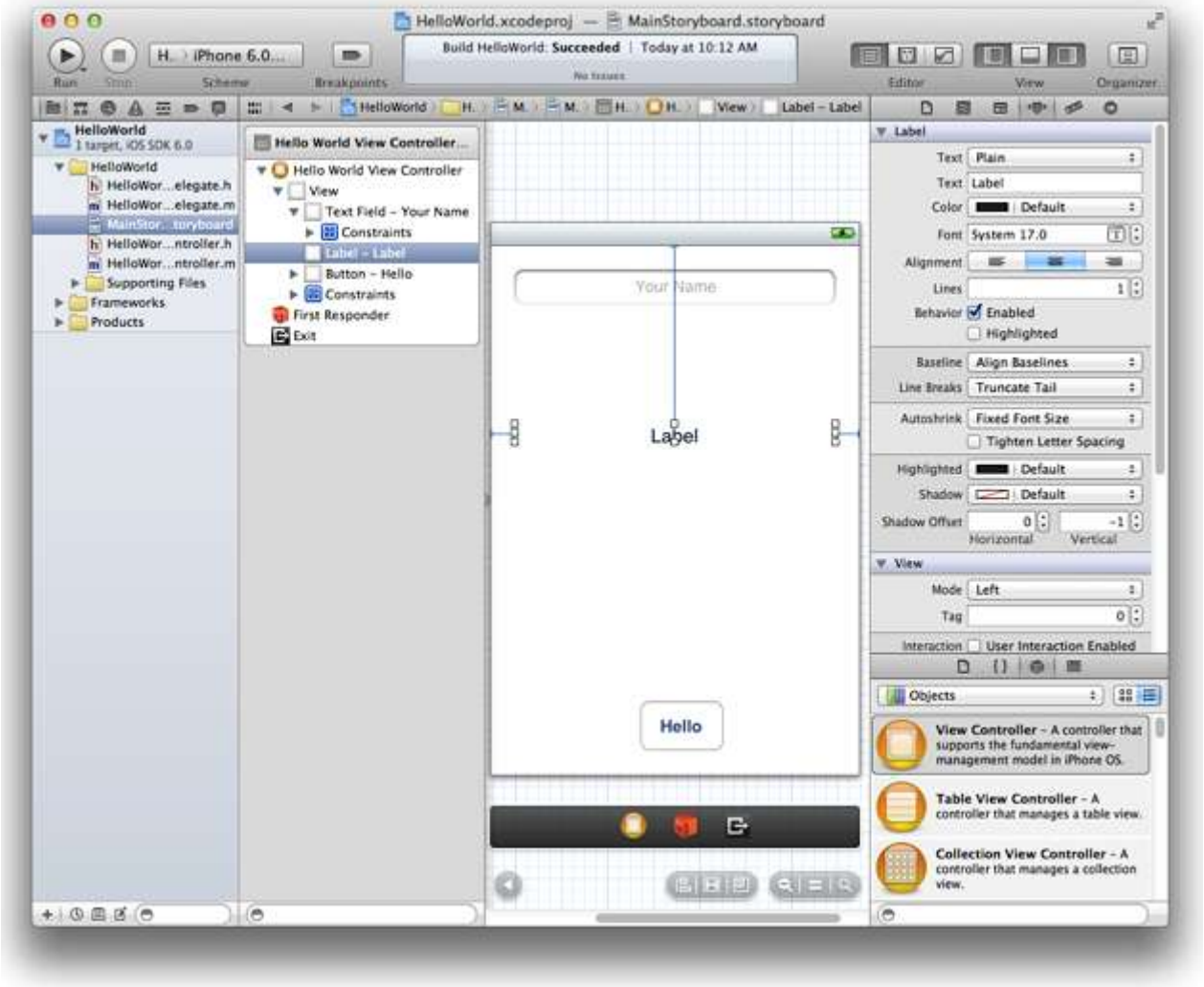
14. 拖移按钮使其靠近视图底部并且水平居中。

15. 在画布上，连按该按钮，然后输入文本 `Hello`。

    在视图中连按该按钮后，而还未输入文本时，您看到应该是这样的：



在添加文本栏、标签和按钮 UI 元素，并对布局做出建议的修改后，您的项目看起来应该是这样的：



您可能注意到，当您将文本栏、标签和按钮添加到背景视图时，Xcode 在名为 **Constraints** 的大纲视图中插入了项目。Cocoa Touch 具有一个自动布局系统，让您定义用户界面元素的布局约束。这些约束定义用户界面元素之间的关系，当其他视图的大小改变或设备摆放方向改变时，该关系影响各界面元素如何改变其位置和几何形状。现在不要改变您添加到用户界面的视图的默认约束。

您还可以对文本栏进行一些修改，使文本栏的行为符合用户的期望。首先，因为用户要输入自己的姓名，您可以确保 iOS 对用户键入的每个英文单词的首字母实施大写。其次，还可以确保与文本栏相关联的键盘配置为输入姓名（而不是数字），并且键盘显示“Done”按钮。

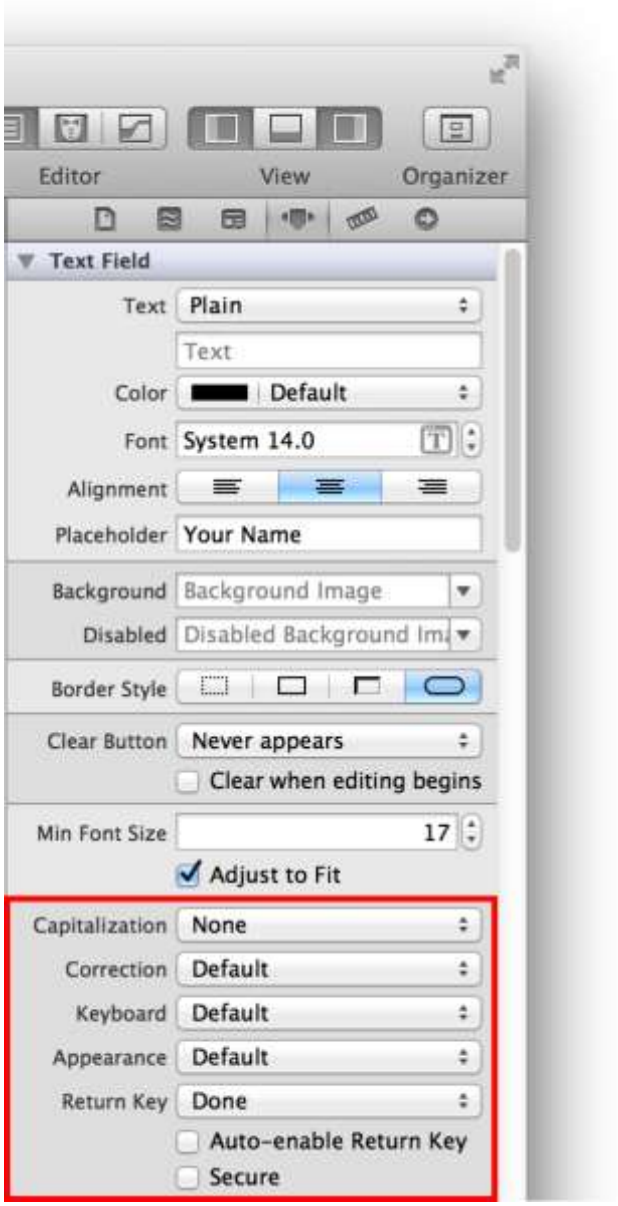
这些更改所遵循的原则是：因为在设计时，您已知道文本栏将包含何种类型的信息，您可以配置文本栏使它运行时的外观和行为符合用户的任务。这些配置都可以在“Attributes”检查器中修改。

▶

配置文本栏

1. 在视图中，选择文本栏。
2. 在“Text Field Attributes”检查器中，进行以下选择：
   - 在“Capitalization”弹出式菜单中，选取“Words”。
   - 确保“Keyboard”弹出式菜单设定为“Default”。
   - 在“Return Key”弹出式菜单中，选取“Done”。

   选定以上选项后，“Text Field Attributes”检查器应该是这样的：



在 Simulator 中运行应用程序，以确定所添加的 UI 元素外观正如所期望的样子。如果点按“Hello”按钮，该按钮应

该高亮显示；如果在文本栏内点按，键盘应该出现。不过，这时按钮没有任何功能，标签还是空的，而且键盘出现后无法使它消失。要添加此功能，需要在 UI 元素和视图控制器之间进行适当的连接。下面将说明如何建立连接。

注：因为只是在 Simulator 中（而不是在设备上）运行应用程序，所以是通过点按来激活控制，而不是用手轻按的方式。

为按钮创建操作

当用户激活一个 UI 元素时，该元素可以向知道如何执行相应操作方法的对象发送一则操作消息，例如“将此联系人添加到用户的联系人列表”。这种互动是**目标-操作**机制的一部分，该机制是另一种 Cocoa Touch 设计模式。

在本教程中，当用户轻按“Hello”按钮时，您想要按钮发送一则“更改问候语”的消息（**操作**）给视图控制器（**目标**）。视图控制器通过更改其管理的字符串（即模型对象）来响应此消息。然后，视图控制器更新在标签中显示的文本，以反映模型对象值的变动。

使用 Xcode，您可以将操作添加到 UI 元素，并设置其相应的操作方法。方法是按住 Control 键并将画布上的元素拖移到源文件中的合适位置（通常是类扩展在视图控制器的实现文件中）。串联图将您通过这种方式创建的连接归档存储下来。稍后，应用程序载入串联图时，会恢复这些连接。

▶

为按钮添加操作

1. 如有需要，选择项目导航器中的 `MainStoryboard.storyboard`，将场景显示在画布上。
2. 在 Xcode 工具栏中，点按“Utilities”按钮以隐藏实用工具区域，点按“Assistant Editor”按钮以显示辅助编辑器面板。

   “Assistant Editor”按钮为中间的那个编辑器按钮，外观是这样的：  。
3. 确定“Assistant”显示视图控制器的实现文件，即 `HelloWorldViewController.m`。

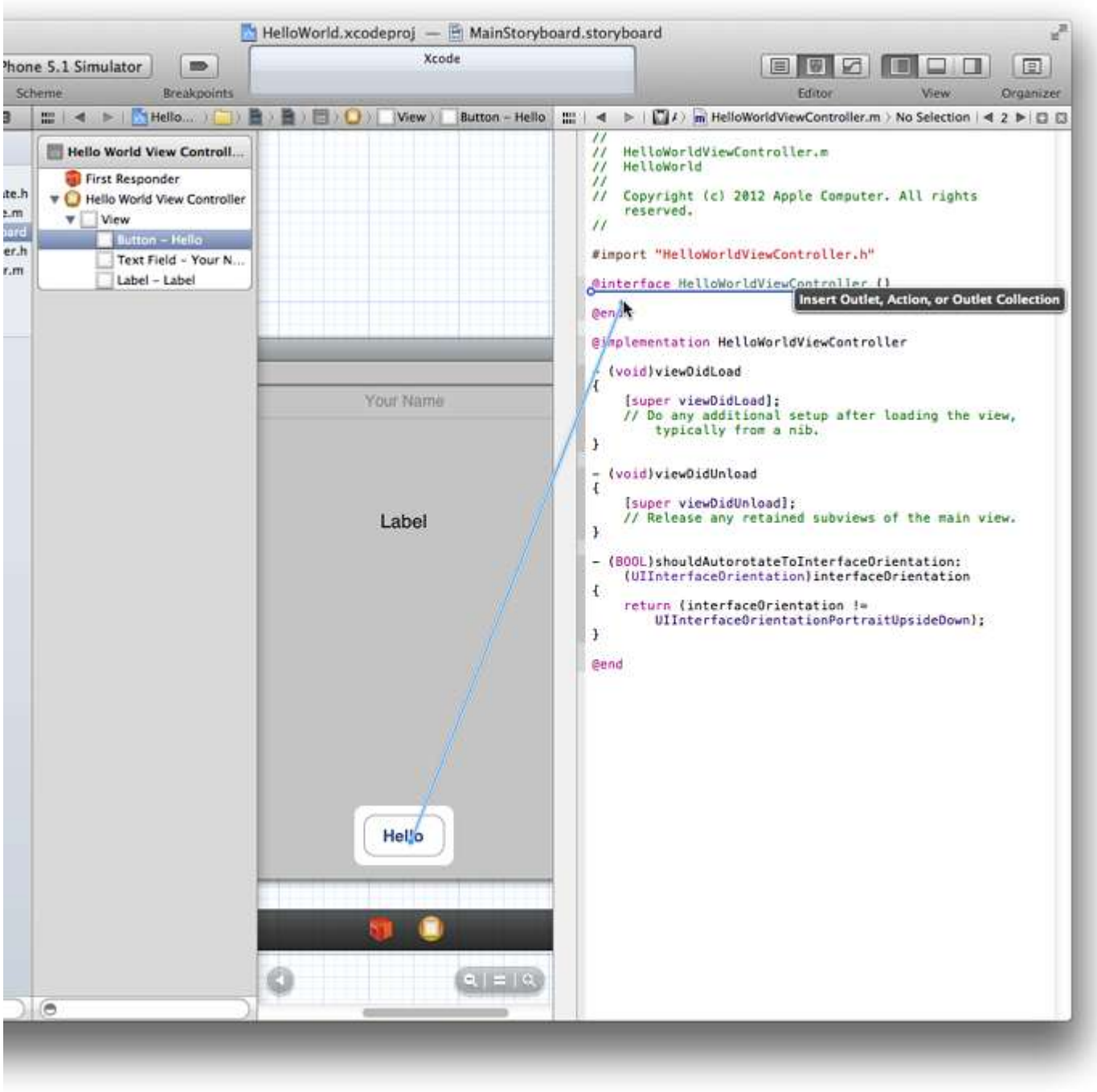
   万一显示的是 `HelloWorldViewController.h`，请在项目导航器中选择 `HelloWorldViewController.m`。
4. 在画布上，按住 Control 键将“Hello”按钮拖移到 `HelloWorldViewController.m` 中的类扩展。

   实现文件中的类扩展是申明类的专有属性和方法的地方。（在**编写 Objective-C 代码**中，您将学到有关类扩展的更多信息。）Outlet 和操作应该专有。视图控制器的 Xcode 模板包含实现文件中的类扩展。以“HelloWorld”项目为例，类扩展看起来像这样：

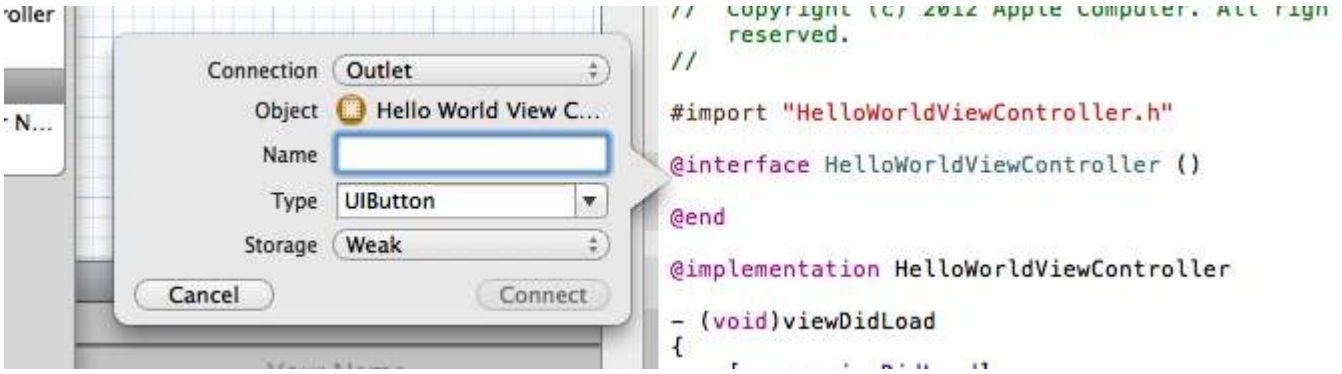
```
@interface HellowWorldViewController()



@end
```

   要按住 Control 键拖移，请按住 Control 键不放，并将按钮拖移到辅助编辑器中的实现文件。随着您按住

Control 键拖移，看到的应该是这样的：



松开 Control 键并停止拖移后，Xcode 会显示一个弹出式窗口，在窗口中可以设置刚进行的操作连接：



注：如果在 `HelloWorldViewController.m` 类扩展区域以外的其他地方松开 Control 键并停止拖移，可能会看到不同类型的弹出式窗口，或者是什么都没有。如果出现这种情况，请在画布上的视图内部点按来关闭弹出式窗口（如有需要），并再试一次按住 Control 键拖移。

5. 在弹出式窗口中，配置按钮的操作连接：

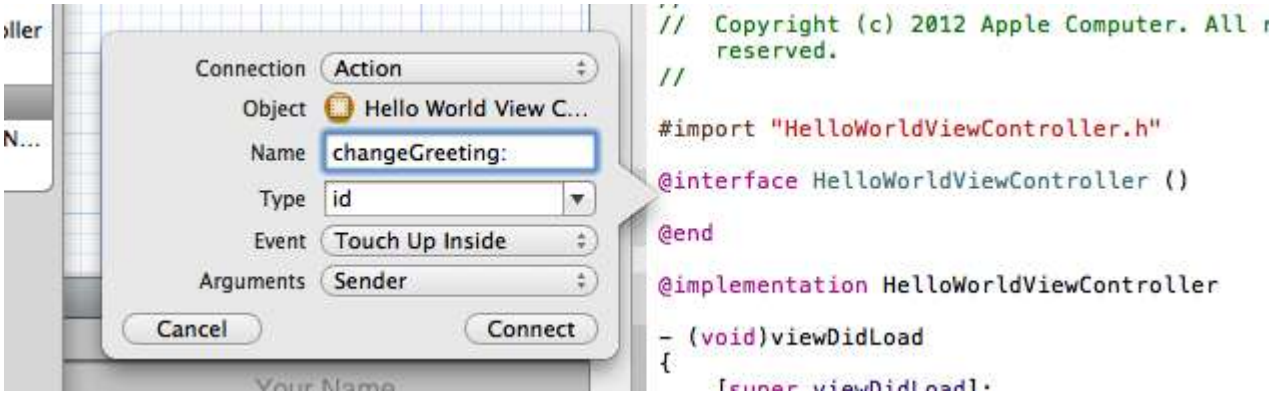
- 在“Connection”弹出式菜单中，选取“Action”。
- 在“Name”栏中，输入 `changeGreeting:`（请确保包括冒号）。

  在稍后步骤中，您将实施 `changeGreeting:` 方法，让它把用户输入文本栏的文本载入，然后在标签中显示。
- 确定“Type”栏包含 `id`。

  `id` 数据类型可指任何 Cocoa 对象。在这里使用 `id` 是因为无论哪种类型的对象发送消息都没有关系。
- 请确定“Event”弹出式菜单包含“Touch Up Inside”。

  指定“Touch Up Inside”事件是因为您想要在用户触摸按钮后提起手指时发送消息。
- 请确定“Arguments”弹出式菜单包含“Sender”。

配置完操作连接后，弹出式窗口应该是这样的：



6. 在弹出式窗口中，点按“Connect”。

Xcode 为新的 `changeGreeting:` 方法添加一个存根实现，并通过在该方法的左边显示一个带有填充的圆圈，

以标示已经建立连接。

按住 Control 键将“Hello”按钮拖移到 HelloWorldViewController.m 文件中的类扩展，并配置生成的操作后，您完成了两件事情：通过 Xcode 将合适的代码添加到了视图控制器类中（在 HelloWorldViewController.m 中），并在按钮和视图控制器之间创建了连接。具体来说，Xcode 做了以下事情：

- 在 HelloWorldViewController.m 中，将以下操作方法声明添加到了类扩展：

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

- 并将以下的存根方法添加到了实现区域：

```
- (IBAction)changeGreeting:(id)sender {

}
```

- **注：**IBAction 是一个特殊关键词，用于告诉 Xcode 将一个方法作为目标-操作连接的操作部分来处理。IBAction 被定义为 void。
- 操作方法中的 sender 参数指向发送操作消息的对象（在本教程中，发送对象为按钮）。
- 它在按钮和视图控制器之间创建了连接。

接下来，您要在视图控制器和剩余的两个 UI 元素（即标签和文本栏）之间创建连接。

为文本栏和标签创建 Outlet

Outlet 描述了两个对象之间的连接。当您想要一个对象（例如视图控制器）和它包含的对象（例如文本栏）进行通讯时，须将被包含的对象指定为 outlet。应用程序运行时，会恢复在 Xcode 中创建的 outlet，从而使对象在运行时可以互相通讯。

在本教程中，您想要视图控制器从文本栏获取用户的文本，然后将文本显示在标签上。为确保视图控制器可以和这些对象通讯，您在它们之间创建 outlet 连接。

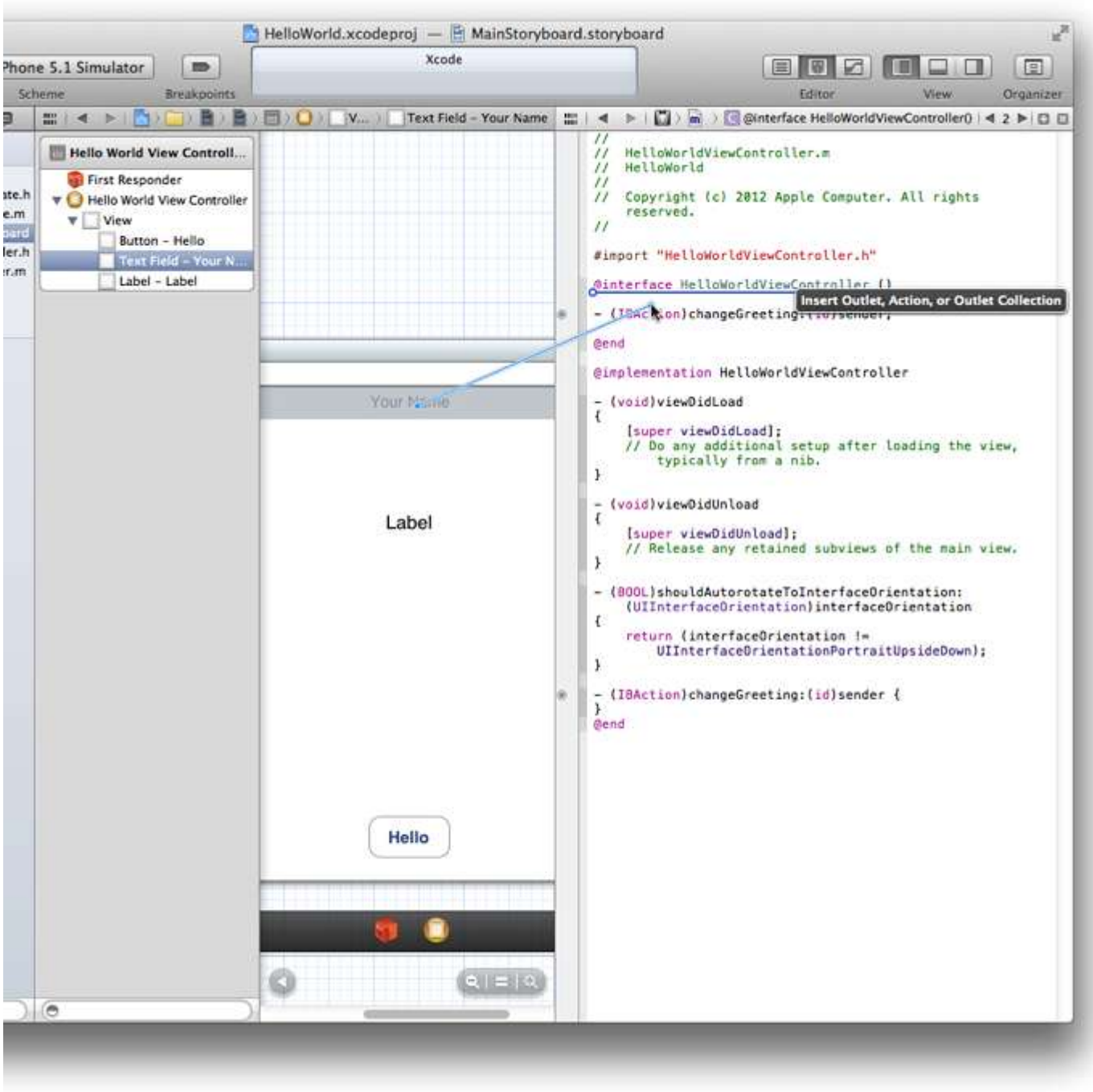
为文本栏和标签添加 outlet 的步骤与添加按钮操作的步骤非常相似。开始之前，请确定主串联图文件仍然显示在画布上，而 HelloWorldViewController.m 在辅助编辑器中保持打开。

▶

为文本栏添加 outlet

1. 按住 Control 键将视图中的文本栏拖移到实现文件中的类扩展。

   随着您按住 Control 键拖移，看到的应该是这样的：



   在类扩展内部的任何地方松开 Control 键并停止拖移都没有关系。在本教程中，文本栏和标签的 outlet 声明出现在“Hello”按钮的方法声明的上面。

2. 在松开 Control 键并停止拖移时出现的弹出式窗口中，配置文本栏的连接：

   - 确定“Connection”弹出式菜单包含“Outlet”。
   - 在“Name”栏中，键入“textField”。

您可以给 outlet 随意命名，但是如果 outlet 名称与其表示的项目有关联，您的代码会更便于阅读。

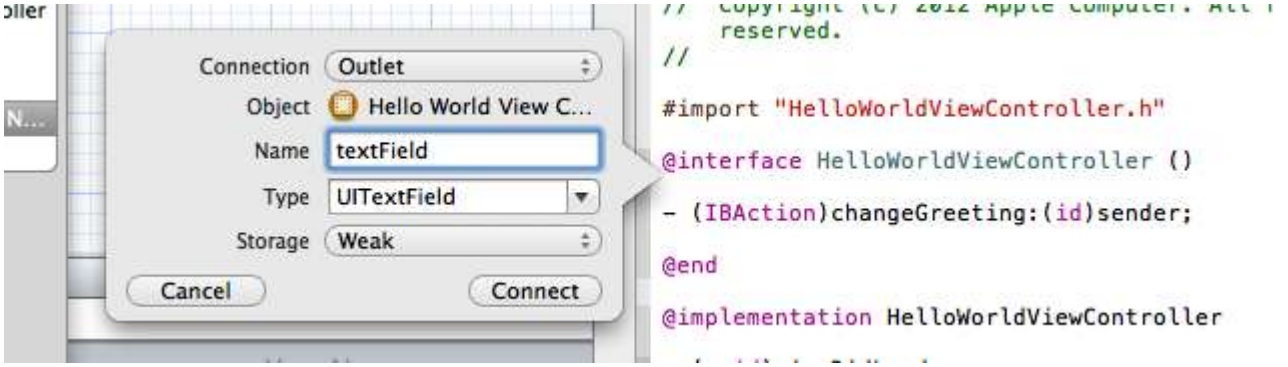
- 确定“Type”栏包含“UITextField”。

  将“Type”栏设定为“UITextField”可确保 Xcode 将 outlet 仅与文本栏连接。

- 确定“Storage”弹出式菜单包含默认值“Weak”。

  稍后在**掌握基本的编程技能**中，您将了解有关强储存和弱储存的更多信息。

3. 完成这些设置后，弹出式窗口看起来应该是这样的：

4. 

5. 在弹出式窗口中，点按“Connect”。

通过为文本栏添加 outlet，您完成了两件事情：在这个过程中：

- Xcode 将合适的代码添加到了视图控制器类的实现文件 (`HelloWorldViewController.m`)。

  具体来说，Xcode 将以下声明添加到了类扩展：

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

  注：`IBOutlet` 是一个特殊关键词，仅用于告诉 Xcode 将对象作为 outlet 处理。它实际上的定义为什么都不是，因此在编译时不起作用。

- Xcode 在视图控制器和文本栏之间建立了连接。

  通过在视图控制器和文本栏之间建立连接，用户输入的文本可以传递给视图控制器。和处理 `changeGreeting:` 方法声明一样，Xcode 在文本栏声明的左边显示带有填充的圆圈，以表示已经建立连接。

注：较早版本的 Xcode 使用按住 Control 键拖移的方式将 `@synthesize` 指令添加到您所声明的每个属性的实现块。因为编译器自动合成存取方法，所以这些指令是不必要的。您可以放心地将它们全部删除。

现在为标签添加 outlet 并配置连接。在视图控制器和标签之间建立连接，会让视图控制器以字符串（该字符串包含用户输入的文本）来更新标签。完成此任务的步骤与为文本栏添加 outlet 的步骤相同，只不过在配置时要做适当修改。（确定 `HelloWorldViewController.m` 仍然显示在辅助编辑器中。）

▶

为标签添加 outlet

1. 按住 Control 键将视图中的标签拖移到辅助编辑器中的 `HelloWorldViewController.m` 的类扩展。
2. 在松开 Control 键并停止拖移时出现的弹出式窗口中，配置标签的连接：
   - 确定“Connection”弹出式菜单包含“Outlet”。
   - 在“Name”栏中，键入“label”。
   - 确定“Type”栏包含“UILabel”。
   - 确定“Storage”弹出式菜单包含“Weak”。
3. 在弹出式窗口中，点按“Connect”。

到此为止，您一共创建了三种到视图控制器的连接：

- 按钮的操作连接
- 文本栏的 Outlet 连接
- 标签的 Outlet 连接

您可以在“Connections”检查器中验证这些连接。

▶

为视图控制器打开“Connections”检查器

1. 点按标准编辑器按钮以关闭辅助编辑器，并切换到标准编辑器视图。

   标准编辑器按钮是最左边的那个“Editor”按钮，它是这样的： 

2. 点按“Utilities”视图按钮以打开实用工具区域。
3. 在大纲视图中选择“Hello World View Controller”。
4. 在实用工具区域显示“Connections”检查器。

   “Connections”检查器按钮是检查器选择栏最右边的那个按钮，它是这样的： 

在“Connections”检查器中，Xcode 显示了所选对象（在本例中为视图控制器）的连接。在工作区窗口中，您看到的应该是这样的：

您会发现，在视图控制器和其视图之间，除了您创建的三个连接之外，还有一个连接。Xcode 提供了视图控制器和其视图之间的默认连接；您不必用任何方式访问它。

建立文本栏的委托连接

您还需要在应用程序中建立另一个连接：您需要将文本栏连接到您指定的委托对象上。在本教程中，您将视图控制器用作文本栏的委托。

您需要为文本栏指定一个委托对象。这是因为当用户轻按键盘中的“Done”按钮时，文本栏发送消息给它的委托（前面提到过委托是代表另一个对象的对象）。在后面的步骤中，您将使用与此消息相关联的方法让键盘消失。

确定串联图文件已在画布上打开。如果未打开，则在项目导航器中选择 `MainStoryboard.storyboard`。

▶

设定文本栏的委托

1. 在视图中，按住 Control 键将文本栏拖移到场景台中的黄色球体（黄色球体代表视图控制器对象）。

   松开 Control 键并停止拖移时，看到的应该是这样的：

2. 在出现的半透明面板的“Outlets”部分中选择“delegate”。

让应用程序具有辅助功能

iOS 操作系统提供了许多功能，让应用程序可供所有用户使用，包括有视觉障碍、听觉障碍和身体残疾的用户。让应用程序具有辅助功能，也就让应用程序接触到了数以百万计原本不能够使用它的用户。

Apple 的创新性读屏技术 VoiceOver 是一个重要的辅助功能。使用 VoiceOver，用户可以在不看屏幕的情况下导航和控制应用程序的各部分。通过触摸用户界面中的控制或其他对象，用户可以知道他们的位置、可以执行的操作以及执行某些操作后将发生什么。

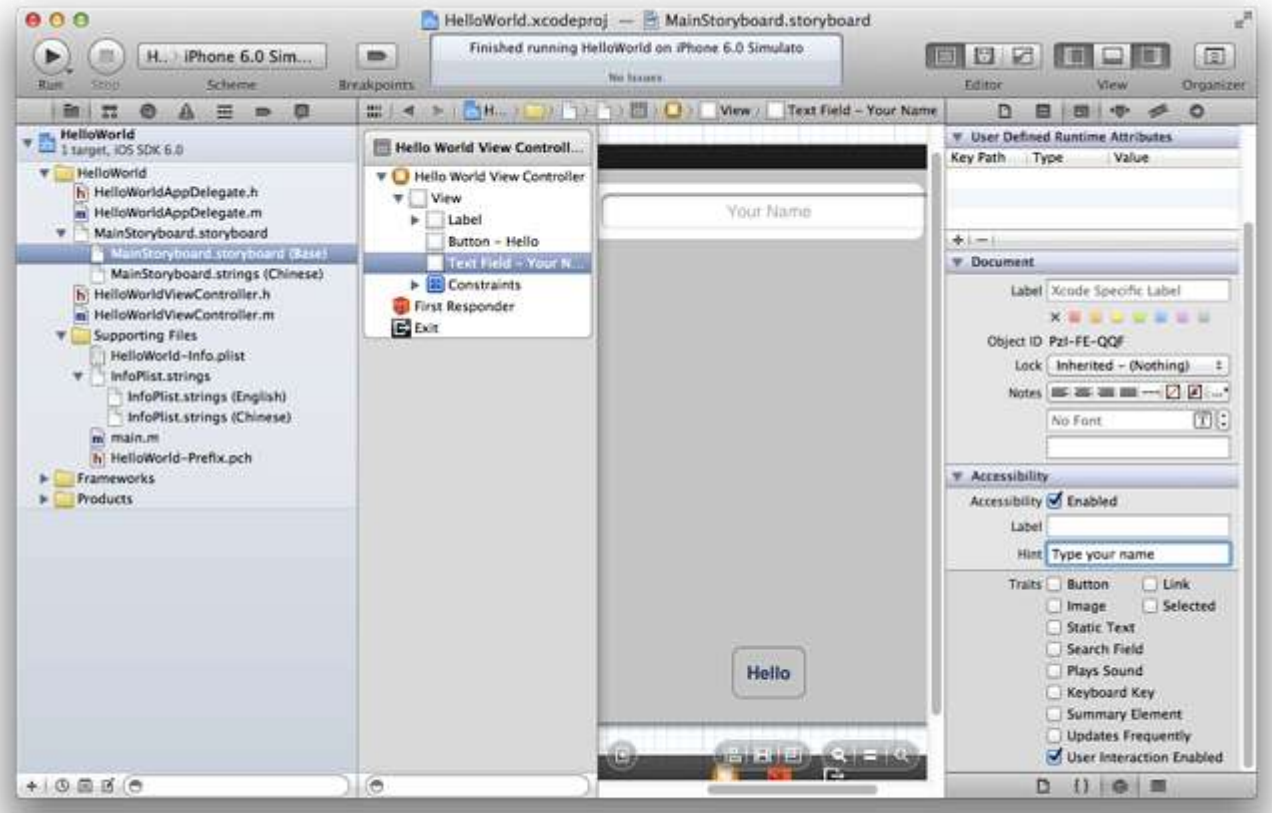
您可以将一些辅助功能属性添加到用户界面中的任何视图。这些属性包括视图的当前值（例如文本栏中的文本）、其标签、提示以及很多特征。就 HelloWorld 应用程序而言，您将要给文本栏添加一则提示。

注：您所添加的任何辅助功能文本都应该本地化。要了解如何将辅助功能文本本地化，请参阅“应用程序设计”中的**将您的应用程序国际化**。

▶

添加辅助功能提示

1. 在项目导航器中选择该串联图文件 (Base Internationalization)。
2. 选择该文本栏。
3. 在“Identity”检查器的“Accessibility”部分，在“Hint”栏中键入“Type your name”。



测试应用程序

点按“Run”以测试您的应用程序。

在您点按“Hello”按钮时，应该看到它高亮显示。您还应该发现，如果在文本栏中点按，键盘会出现，您可以输入文本。然而还没有办法让键盘消失。要让键盘消失，您必须实施相关的委托方法。下一章会教您如何做。现在请退出 Simulator。

小结

当您在画布上的视图控制器与辅助编辑器中实现文件（即 `HelloWorldViewController.m`）里的类扩展之间建立适当的连接时，也就更新了实现文件以支持 Outlet 和操作。

您不必使用 Xcode 自动添加代码（即通过按住 Control 键从画布拖移到源文件来建立连接时）的功能。而是可以自行编写类扩展的属性和方法声明，或公共属性和方法声明的头文件，然后就像建立文本栏的委托那样进行连接。

然而通常情况下，Xcode 做得越多，您犯错的机会就越少，需要键入的内容也会越少。

实施视图控制器

实施视图控制器包括这几部分：为用户姓名添加属性，实施 changeGreeting: 方法，确保用户轻按“Done”时键盘消失。

为用户姓名添加属性

您需要为保存用户姓名的字符串添加属性声明，这样您的代码就总能引用该字符串。因为此属性必须是公共的，即对客户端和子类为可见，所以须将此声明添加到视图控制器的头文件，即 HelloWorldViewController.h。公共属性表示您打算如何使用这一类的对象。

属性声明是一个指令，它告诉编译器如何为变量（例如用来保存用户姓名的变量）生成存取方法。（添加属性声明后，您将了解到有关存取方法的信息。）

到此为止，不需要对串联图文件做出任何进一步的修改。要腾出更多空间以按照以下步骤来添加代码，请再次点按“Utilities View”按钮来隐藏实用工具区域（或者选取“View”>“Utilities”>“Hide Utilities”）。

▶

为用户姓名添加属性声明

1. 在项目导航器中，选择 HelloWorldViewController.h。
2. 在 @end 语句前，为字符串编写一个 @property 语句。

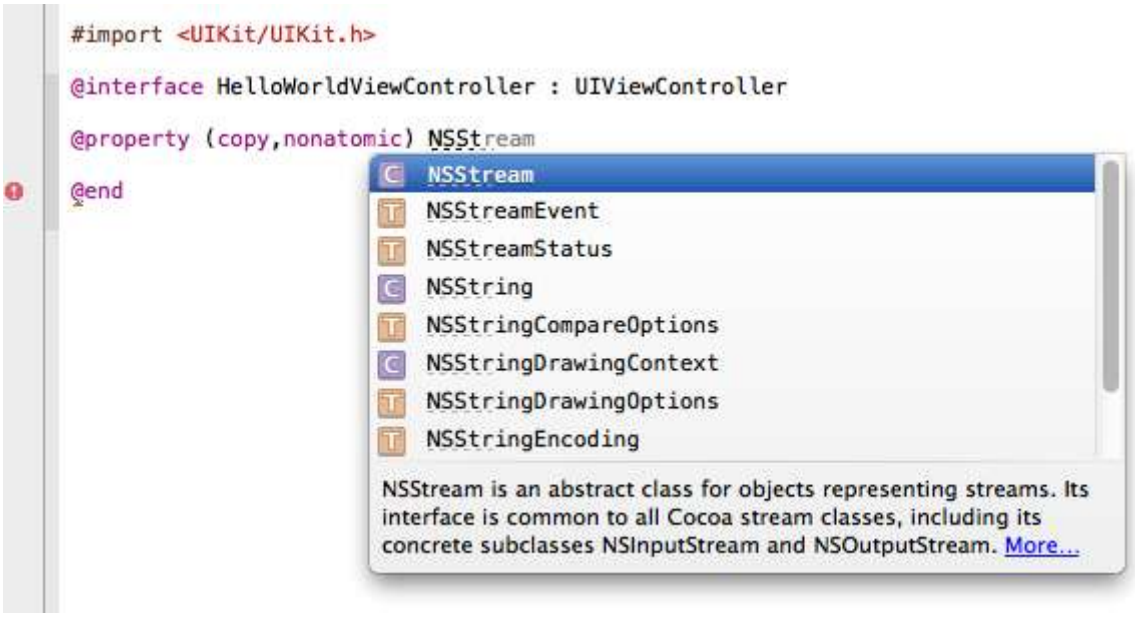
   属性声明应该是这样的：

```
@property (copy, nonatomic) NSString *userName;
```

   可以拷贝和粘贴以上代码，也可以在编辑器面板中键入以上代码。如果您决定键入代码，请注意 Xcode 会根据键入内容提供自动补齐的建议。例如，开始键入@prop...Xcode 猜测您想要键入 @property，因此会在内联建议面板中显示这样一个符号（如下图所示）：

   如果建议合适（如上述示例所示），则按下 Return 键接受建议。

   随着您继续键入，Xcode 可能提供一个建议列表供您选取。例如随着您键入 NSStr...，Xcode 可能显示如下补齐列表：



Xcode 显示补齐列表时，按下 Return 键以接受高亮显示的建议。如果高亮显示的建议不正确（如上图所示列表），可使用箭头键从列表中选择合适的项目。

编译器自动为您声明的任何属性合成存取方法。**存取方法**是一种获取或设定一个对象的属性的值的方法（因此，存取方法有时也称为“getter”和“setter”）。例如，编译器为刚刚声明的 `userName` 属性生成以下的 getter 和 setter 声明及其实现：

- `- (NSString *)userName;`
- `- (void)setUserName:(NSString *)newUserName;`

编译器也自动声明专有实例变量以支持每一个经声明的属性。例如，编译器声明名为 `_userName` 的实例变量以支持 `userName` 属性。

注：编译器将生成的存取方法添加到编译代码，而不是添加到您的源代码中。

实施 changeGreeting: 方法

在上一章“配置视图”中，您已配置了“Hello”按钮，因此在用户轻按该按钮时，它发送 `changeGreeting:` 消息给视图控制器。作为响应，您想要视图控制器将用户在文本栏中输入的文本显示在标签中。具体来说，`changeGreeting:` 方法应该：

- 从文本栏取回字符串，并将视图控制器的 `userName` 属性设定为此字符串。
- 基于 `userName` 属性，创建新的字符串，并将其显示在标签中。

▶

实施 changeGreeting: 方法

1. 如有需要，在项目导航器中选择 `HelloWorldViewController.m`。

您可能需要滚动到文件的末尾才能看到 changeGreeting: 存根实现，它是 Xcode 为您添加的。

2. 添加以下代码来完成 changeGreeting: 方法的存根实现：

```
- (IBAction)changeGreeting:(id)sender {

    self.userName = self.textField.text;

    NSString *nameString = self.userName;
    if ([nameString length] == 0) {
        nameString = @"World";
    }
    NSString *greeting = [[NSString alloc] initWithFormat:@"Hello, %@!", nameString];
    self.label.text = greeting;
}
```

changeGreeting: 方法中有几项有趣的事值得注意：

- self.userName = self.textField.text; 从文本栏取回文本，并将视图控制器的 userName 属性设定为该结果。

  在本教程中，您不会在其他任何地方用得上那个保存着用户姓名的字符串，但重要的是您要记住它的角色：这正是视图控制器所管理的非常简单的模型对象。一般情况下，控制器应在它自己的模型对象中维护应用程序数据的相关信息。换句话说，应用程序数据不应储存在用户界面元素（例如 HelloWorld 应用程序的文本栏）中。

- NSString *nameString = self.userName; 创建一个新的变量（为 NSString 类型）并将其设为视图控制器的 userName 属性。

- @"World" 是一个字符串常量，用 NSString 的实例表示。如果用户运行应用程序但不输入任何文本（即 [nameString length] == 0），nameString 将包含字符串“World”。

- initWithFormat: 方法是由 Foundation 框架提供给您的。它创建一个新的字符串，按您提供的格式字符串所规定的格式（很像 ANSI C 库中的 printf 函数）。

  在格式字符串中，%@ 充当字符串对象的占位符。此格式字符串的双引号中的所有其他字符都将如实显示在屏幕上。

将视图控制器配置为文本栏的委托

如果生成并运行应用程序，在点按按钮时应该会看到标签显示“Hello, World!”。如果您选择文本栏并开始在键盘上键入，您会发现完成文本输入后，仍然无法让键盘消失。

在 iOS 应用程序中，允许文本输入的元素成为第一响应器时，键盘会自动出现；元素失去第一响应器状态时，键盘会自动消失。（前面提到过第一响应器是第一个接收各种事件通知的对象，例如轻按文本栏来调出键盘。）虽然无法从应用程序直接将消息发送给键盘，但是可以通过切换文本输入 UI 元素的第一响应器状态这种间接方式，使键盘出现或消失。

`UITextFieldDelegate` 协议是由 UIKit 框架定义的，它包括 `textFieldShouldReturn:` 方法，当用户轻按“Return”按钮（不管该按钮的实际名称是什么）时，文本栏调用该方法。因为您已经将视图控制器设定为文本栏的委托（在“设定文本栏的委托”中），可以实施该方法，通过发送 `resignFirstResponder` 消息强制文本栏失去第一响应器状态，以该方法的副作用使键盘消失。

注：协议基本上只是一个方法列表。如果一个类符合（或**采用**）某个协议，则保证它可以实施该协议所要求的方法。（协议也可以包括一些可选的方法。）委托协议指定了一个对象可能向其委托发送的所有消息。

▶

将 HelloWorldViewController 配置为文本栏的委托

1. 如有需要，在项目导航器中选择 `HelloWorldViewController.m`。
2. 实施 `textFieldShouldReturn:` 方法。

   此方法应该指示文本栏放弃第一响应器的状态。实现结果应该是这样的：

```
- (BOOL)textFieldShouldReturn:(UITextField *)theTextField {

    if (theTextField == self.textField) {

        [theTextField resignFirstResponder];

    }

    return YES;

}
```

   在本应用程序中，没有必要真正测试 `theTextField == self.textField` 表达式，因为只有一个文本栏。但这是一个很好的模式，因为有些场合您的对象可能是不只一个同类对象的委托，所以可能有需要对它们加以区分。

3. 在项目导航器中选择 `HelloWorldViewController.h`。
4. 在 `@interface` 行的末尾，添加 `<UITextFieldDelegate>`。

   您的接口声明应如下图所示：

```
@interface HelloWorldViewController :UIViewController <UITextFieldDelegate>

…
```

此声明指定 HelloWorldViewController 类采用 UITextFieldDelegate 协议。

测试应用程序

生成并运行应用程序。这一次，一切的表现都应该如您所期望的那样。在 Simulator 中，输入您的姓名后，点按“Done”按钮使键盘消失，然后点按“Hello”按钮将“Hello, **您的姓名**!”显示在标签中。

如果应用程序的表现不是您所期望的，则需要进行故障排除。对于某些要检查的区域，请参阅“排除故障和检查代码”。

小结

既然您已完成了视图控制器的实施，您的首个 iOS 应用程序也就圆满完成了。恭喜您！

返回**马上着手开发 iOS 应用程序**以继续学习有关 iOS 应用程序开发的内容。如果应用程序未能正确工作，请尝试下一章中描述的解决问题的方法，然后再返回**马上着手开发 iOS 应用程序**。

排除故障和检查代码

如果应用程序未能正确工作，请尝试本章描述的解决问题方法。如果应用程序仍然不能正确工作，请将您的代码与本章末尾给出的清单进行比较。

代码和编译器警告

代码编译时应该不会有任何警告。如果真的收到警告，就很有可能是代码出错了。因为 Objective-C 是一种非常灵活的程序设计语言，有时候编译器给出的也仅仅是一些警告而已。

检查串联图文件

如果程序未能正确工作，开发者会很自然地去检查源代码来找出错误。但使用 Cocoa Touch，又增添了另一个层面。应用程序的大部分配置可能是“编码”在串联图中。例如，如果连接不正确，应用程序的行为就会与您的期望不符。

- 如果点按按钮时文本没有更新，可能是没有将按钮的操作连接到视图控制器，或是没有将视图控制器的 outlet 连接到文本栏或标签。
- 如果点按“Done”按钮时键盘不消失，可能是没有将文本栏的委托连接好，或者把视图控制器

的 textField Outlet 连接到了文本栏。务必在串联图上检查文本栏的连接：按住 Control 键点按文本栏以显示半透明的连接面板。您应该会在 delegate outlet 和 textField 引用 outlet 的旁边，看到带有填充的圆圈。

如果您已连接了委托，可能有更微妙的问题（请参阅下一部分“委托方法的名称”）。

委托方法的名称

与委托有关的一个常见错误是拼错委托方法的名称。即使已经正确设定了委托对象，但是如果委托未在其方法实现中使用正确的名称，则不会调用正确的方法。通常最好的做法是从文稿中拷贝和粘贴委托方法声明（例如 textFieldShouldReturn:）。

代码清单

这一部分提供 HelloWorldViewController 类的接口和实现文件的代码清单。请注意，该清单并未列出 Xcode 模板提供的注释和其他方法的实现。

接口文件：HelloWorldViewController.h

```
#import <UIKit/UIKit.h>


@interface HelloWorldViewController :UIViewController <UITextFieldDelegate>


@property (copy, nonatomic) NSString *userName;


@end
```

实现文件：HelloWorldViewController.m

```
#import "HelloWorldViewController.h"


@interface HelloWorldViewController ()
```

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

@property (weak, nonatomic) IBOutlet UILabel *label;


- (IBAction)changeGreeting:(id)sender;


@end


@implementation HelloWorldViewController


- (void)viewDidLoad

{

    [super viewDidLoad];

    // Do any additional setup after loading the view, typically from a nib.

}


- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation

{

    return (interfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);

}


- (IBAction)changeGreeting:(id)sender {


    self.userName = self.textField.text;


    NSString *nameString = self.userName;
```

```
    if ([nameString length] == 0) {
        nameString = @"World";
    }
    NSString *greeting = [[NSString alloc] initWithFormat:@"Hello, %@!", nameString];
    self.label.text = greeting;
}

- (BOOL)textFieldShouldReturn:(UITextField *)theTextField {

    if (theTextField == self.textField) {
        [theTextField resignFirstResponder];
    }
    return YES;
}

@end
```

在 Xcode 中管理工作流程

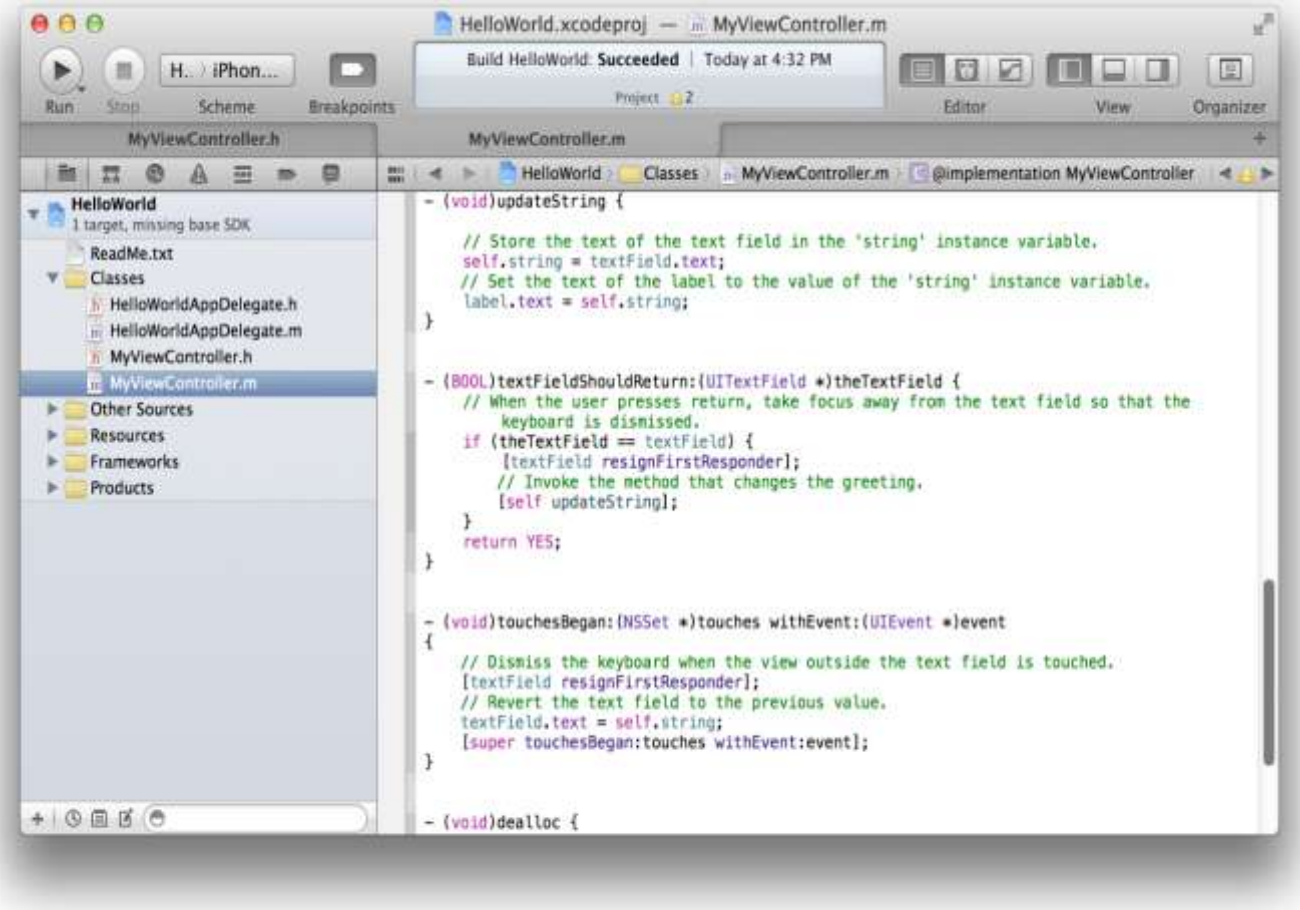
正如在**您的首个 iOS 应用程序**教程中看到的，主要的工作流程任务都在 Xcode 工作区窗口中执行。而辅助任务会在单独的管理器窗口中执行，如阅读文稿、启用设备进行测试，以及准备应用程序用于提交到 App Store。

工作区窗口具有导航器区域、编辑器区域和实用工具区域。在“您的首个 iOS 应用程序”中，使用了导航器区域来选择要编辑的文件，使用编辑器区域来编辑源文件并设计用户界面组件，同时还在实用工具区域中，设定了标签文本和按钮标题。

自定工作区

您可以通过隐藏导航器、编辑器和实用工具区域中的一个或多个来自定工作区。在“您的首个 iOS 应用程序”中，您使用了工具栏中的“View”选择器，来隐藏和显示实用工具区域。隐藏实用工具区域，可让您查看较大的编辑器区域，而显示实用工具区域，可让您检查和选择各种对象属性。

您还可以采用其他方式自定工作区，例如用类似 Safari 的标签，在工作区窗口使用多个，而又有特定工作流程的布局。例如，您可以使用一个标签查看头文件，另一个标签查看实现文件。



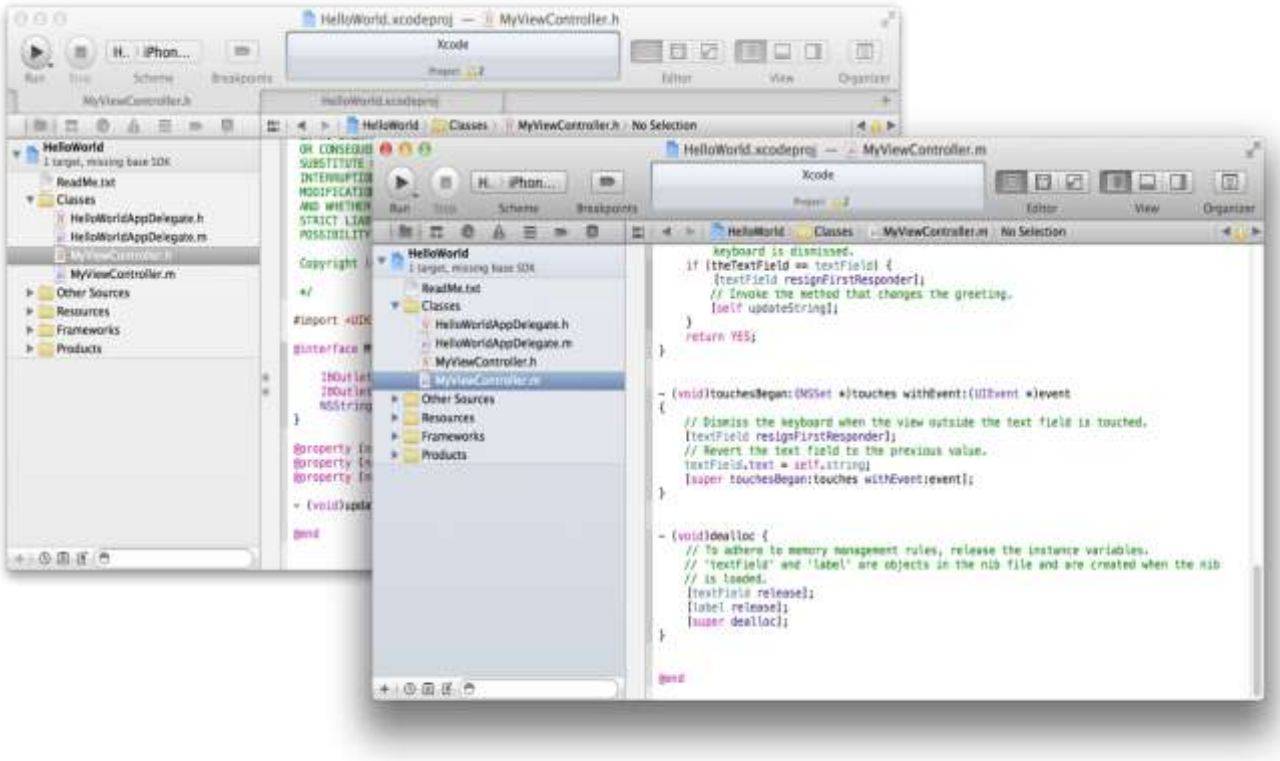
在标签中查看源代码文件

1. 在项目导航器中，选择 HelloWorldViewController.h，以在源代码编辑器中，显示头文件。
2. 选取“View”>“Show Tab Bar”。
3. 选取“File”>“New”>“Tab”。
4. 在项目编辑器中，选择 HelloWorldViewController.m，以在标签式源代码编辑器窗口中，显示实现文件。
5. 点按这些标签，在源文件之间切换。

6. 要移走标签，请将指针移到标签，并点按其关闭框。

7. 通过选取“View”>“Hide Tab Bar”，您可以隐藏标签栏。

您还可以创建多个工作区窗口。每个标签或工作区窗口，都可以相互独立自定。



▶

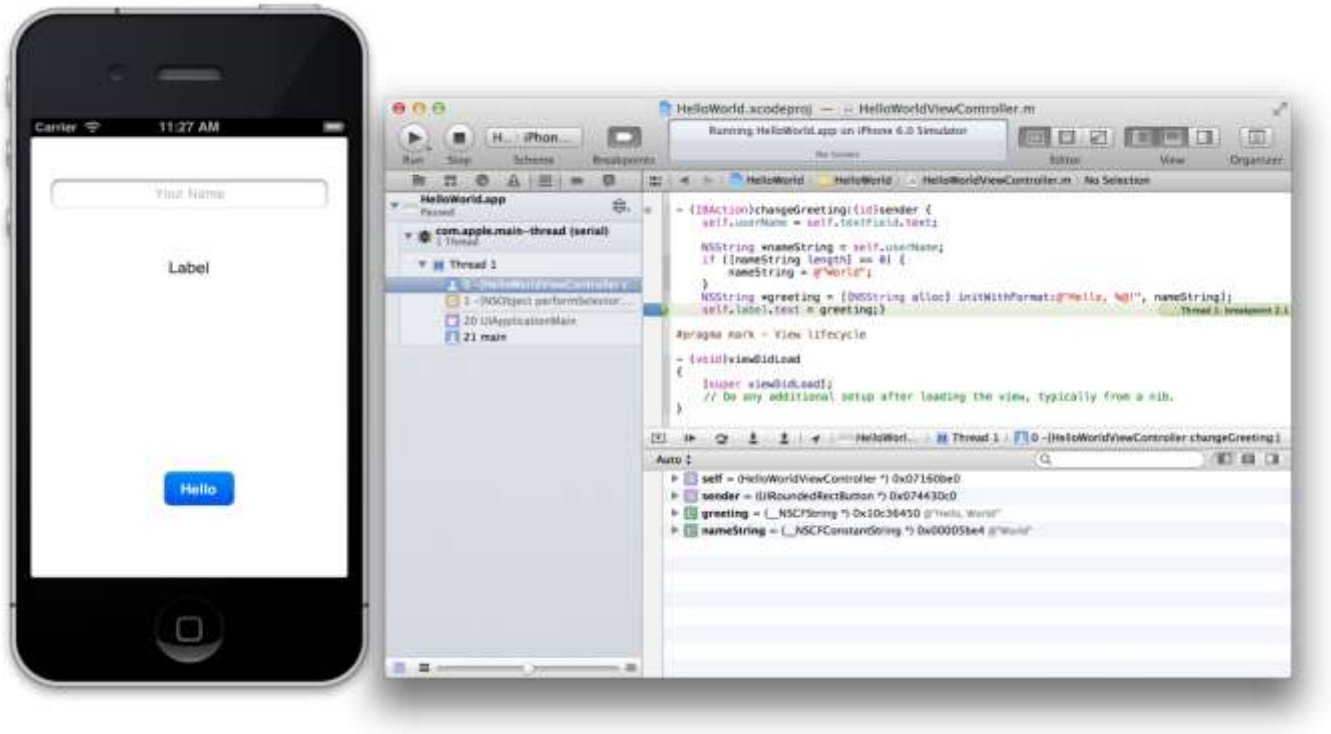
在多个窗口中查看源代码文件

1. 在项目导航器中，选择 HelloWorldViewController.h，以在源代码编辑器中，显示头文件。
2. 选取“File”>“New”>“Window”打开一个新工作区窗口。
3. 在项目编辑器中，选择 HelloWorldViewController.m，以在新窗口中，显示实现文件。
4. 例如，通过“View”选择器显示和隐藏实用工具区域，自定任一窗口。

在 iOS Simulator 中测试应用程序

测试或调试应用程序，可在 Mac 上的 iOS Simulator 中进行。使用 iOS Simulator，您可以确保应用程序按预期方式运行。

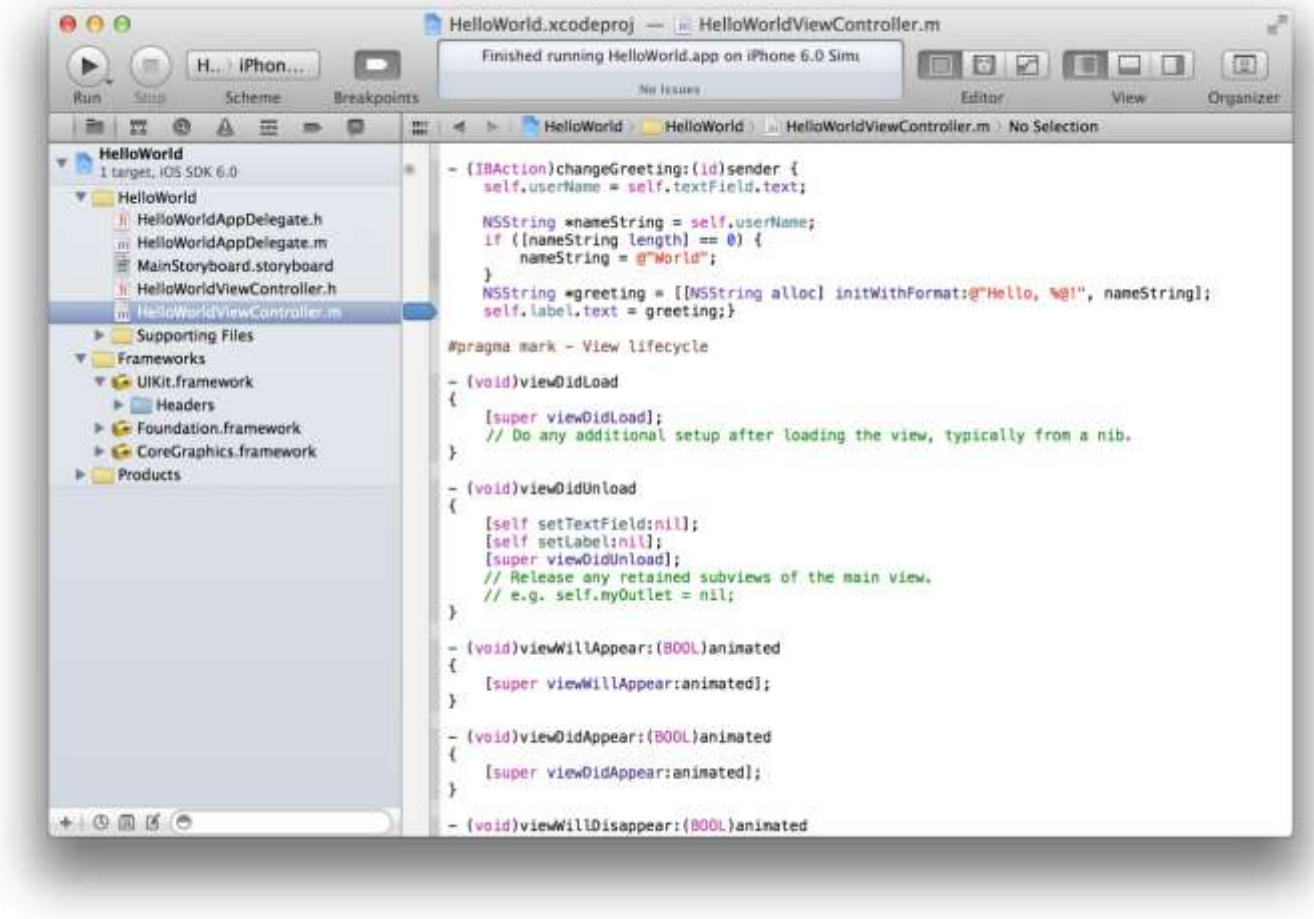
Xcode 内建调试环境。在应用程序运行时，调试导航器会显示堆栈踪迹。调试时，您可以将其展开或收缩来显示或隐藏堆栈结构。随着逐步运行，您可以锁定单个线程，并跟踪该特定线程的执行。



在 Xcode 调试器中运行应用程序

1. 在 HelloWorld 项目的项目导航器中，选择 HelloWorldViewController.m，以在源代码编辑器中显示文件。
2. 找到语句 `self.label.text = greeting;}`。
3. 点按此语句左边的边槽，插入断点。

   出现一个蓝色的断点指示器。



4. 点按工具栏中的“Run”按钮，生成 HelloWorld，并在 iOS Simulator 中运行它。
5. 将 `World` 键入文本栏，然后点按“Done”按钮来关闭键盘。
6. 点按“Hello”按钮。

   断点导致 HelloWorld 停止执行。工作区窗口移到前台，其中调试区域在编辑器区域底部打开。调试区域显示局部变量及其当前值。要去掉该断点，请点按它，并将它拖出边槽。

在 iOS 设备上测试应用程序

尽管您可以在 iOS Simulator 中测试应用程序的基本行为，但还应该在连接到 Mac 的设备上运行它。这些设备提供终极测试环境，您可以在此环境中，观察应用程序的表现，就像它将在客户的设备上表现的那样。进行此类测试是必要的，因为 iOS Simulator 没有运行在设备上运行的所有线程。理想情况下，您应该在所有您想要支持的设备和 iOS 版本上，进行应用程序的测试。

如果您加入了 iOS Developer Program，您可以立即使用 Xcode 在设备上开始运行、测试和调试应用程序。（本路线图前面的**设置**部分，包含了有关加入此计划成为 iOS 开发者的信息。）

您必须从 Apple 获得开发证书，方可在设备上运行您的应用程序。证书作为签名之用，而应用程序必须经过加密

签名，才能在设备上运行。您可通过 Xcode 管理器窗口获得此证书。

注：如果您正在 Xcode 的“Documentation”管理器中阅读以下说明，请按住 Control 键，点按本页的任意位置，然后从关联菜单中，选择“Open Page in Browser”。本页面会出现在默认的网页浏览器中。用网页浏览器查看说明有时会很有用，因为在以下步骤中，选择“Devices”管理器后，“Devices”管理器会替换“Documentation”管理器。阅读完这些说明后，如果您想要继续在 Xcode 中阅读本页，请在“Organizer”窗口的工具栏中，点按“Documentation”。

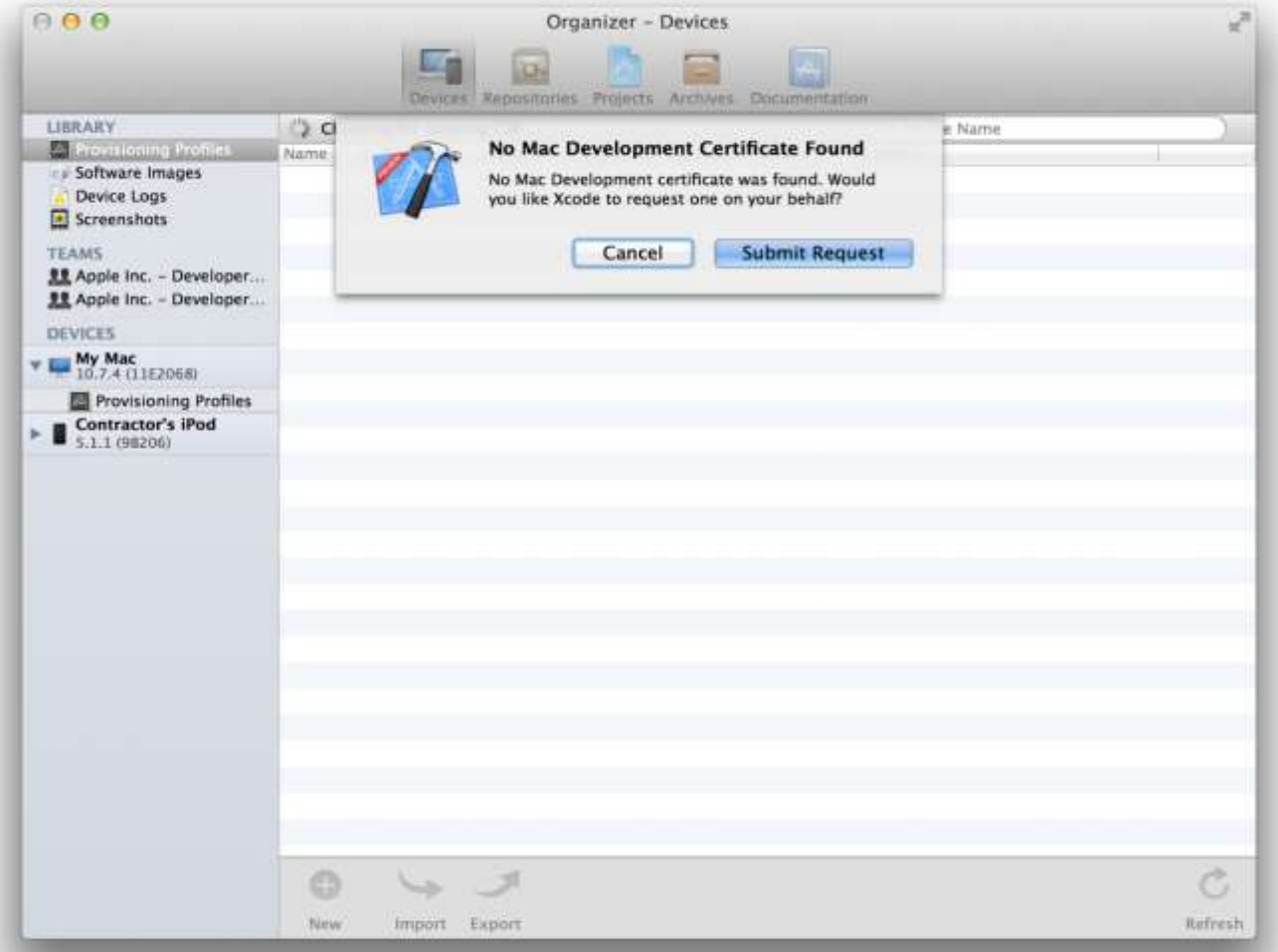
▶

在 Xcode 中获取开发证书

1. 选取“Window”>“Organizer”。
2. 点按“Devices”。
3. 选择“Library”下方的“Provisioning Profiles”。
4. 点按窗口底部的“Refresh”按钮。
5. 输入您的 Apple 开发者用户名称和密码，然后点按“Log in”。

   登录帐户后，会出现一则提示，询问 Xcode 是否应该请求开发证书。
6. 点按“Submit Request”按钮。

   开发证书会添加到钥匙串中，稍后还会添加到 iOS 团队预置描述文件中。可能还会出现另一则提示，询问 Xcode 是否应该请求分发证书，以后将应用程序提交到 App Store 时，需要此证书。如果适用，再次点按“Submit Request”按钮。



要在一个设备上运行应用程序，您还必须在该设备上安装相关的预置描述文件。此预置描述文件可识别您（通过您的开发证书）和设备（通过列出其唯一设备标识符），让该设备能够运行您的应用程序。

▶

在 Xcode 中预备设备

1. 将设备连接到 Mac。
2. 打开“Devices”管理器。
3. 在“Devices”下方选择该设备。
4. 点按“Use for Development”按钮。

首次将一个设备 ID 添加到您的帐户时，Xcode 会创建 iOS 团队预置描述文件（使用 Xcode 通配符应用程序 ID、开发证书和设备 ID）。iOS 团队预置描述文件也要安装在您的设备上。

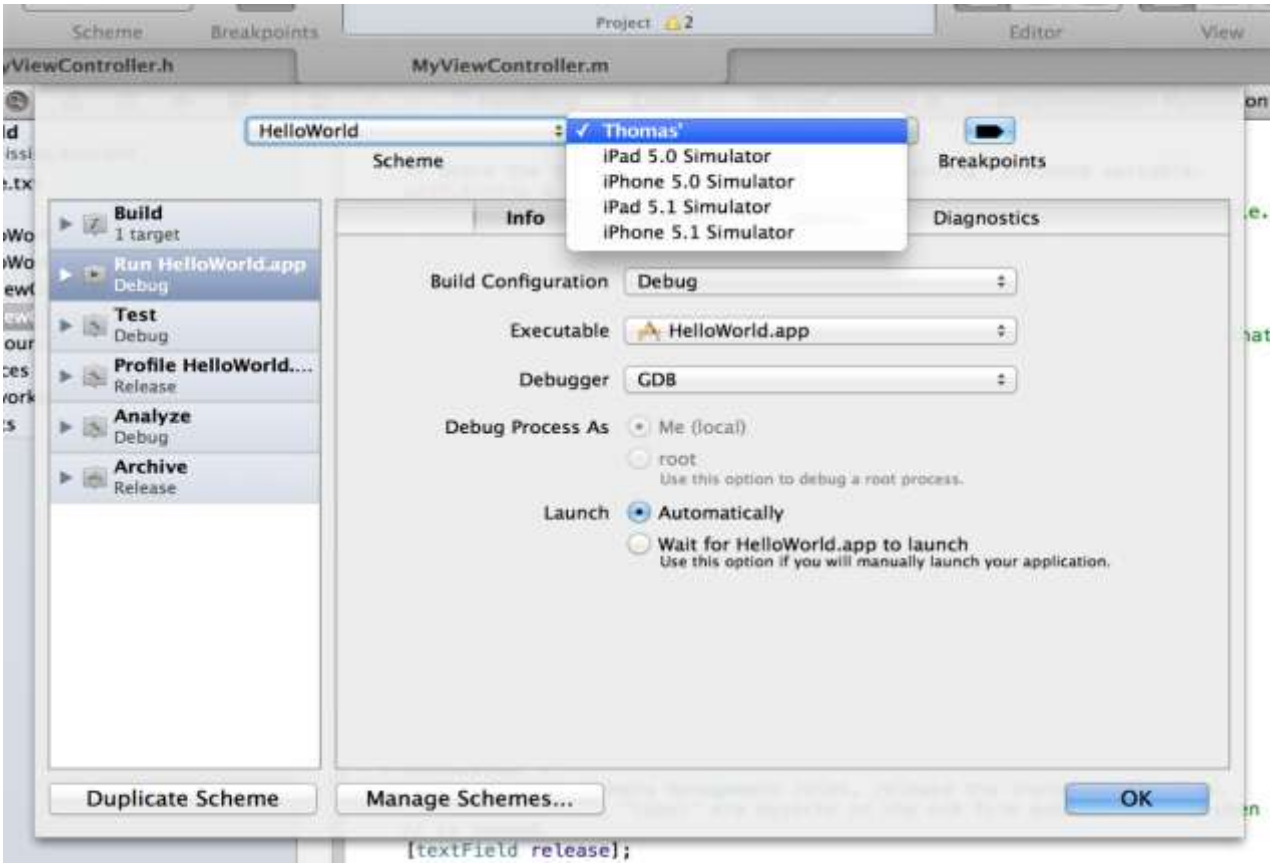
开发证书和预置描述文件就位后，设备就可运行您的应用程序了。程序运行过程中，您还可以使用 Xcode 的工具进行调试和性能分析。

▶

在连接的设备上开启应用程序

1. 在项目的 Xcode 工作区窗口中，选取“Product”>“Edit Scheme”，打开方案编辑器。
2. 从“Destination”弹出式菜单中选择设备。

   将带有有效预置描述文件的设备连接到 Mac 时，设备名称和其运行的 iOS 版本，会整体作为一个选项出现在“Destination”弹出式菜单中。



3. 点按“OK”关闭方案编辑器。

4. 点按“Run”按钮。

   如果出现一则提示，询问代码签名工具是否可以使用钥匙串中的密钥，对应用程序进行签名，请点按“Allow”或“Always Allow”。

快速查找 Xcode 帮助

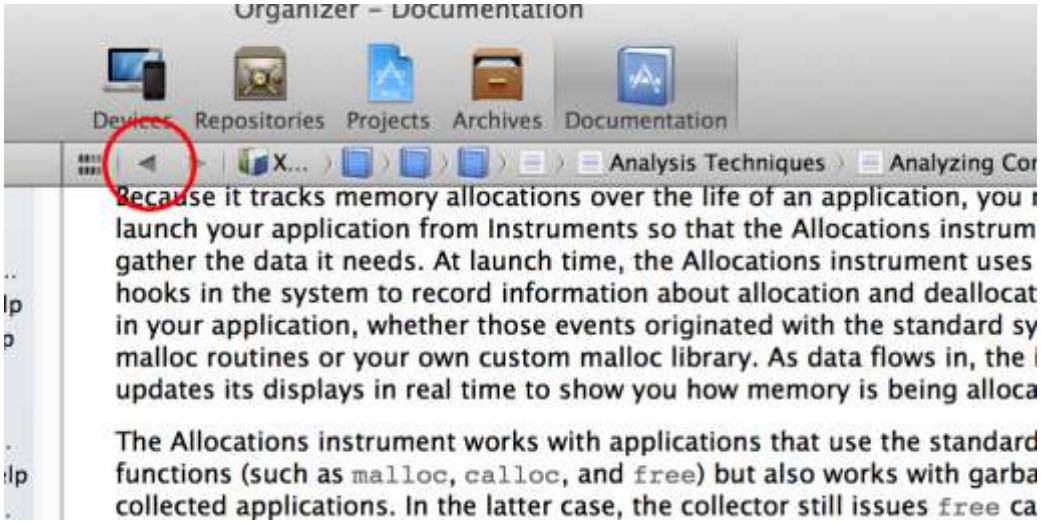
在应用程序开发过程中，您会在 Xcode 中执行众多操作。Xcode 提供工作流程关联型的帮助，如果需要与任务有关的帮助，您可以从 Xcode 用户界面直接访问。此类帮助包含易于遵循的步骤、视频或屏幕快照以及简明描述，有助您快速投入工作。

▶

查看 Xcode 帮助

1. 在 HelloWorld 项目中，从项目导航器选择 HelloWorldViewController.h，以在源代码编辑器中显示头文件。

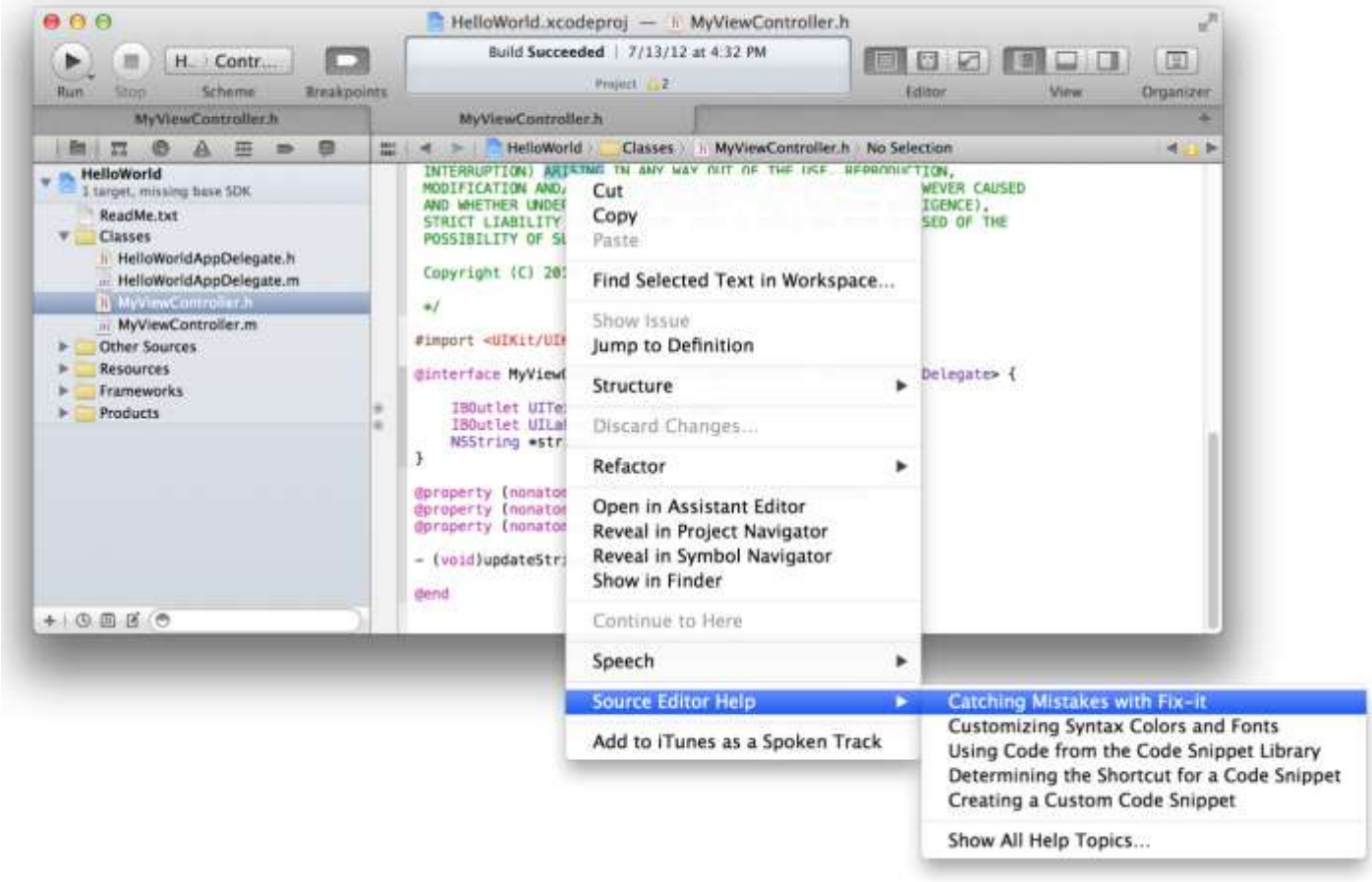
2. 如果您正在 Xcode 的“Documentation”管理器中阅读本文稿，请找到其“Go Back”按钮。在执行其余步骤后，您需要点按它返回本文稿。



3. 按住 Control 键，点按源代码编辑器中的任意位置。

   一个关联菜单会打开，其中“Source Editor Help”是最后一项。

4. 选取“Source Editor Help”，显示常见源代码编辑器任务列表。

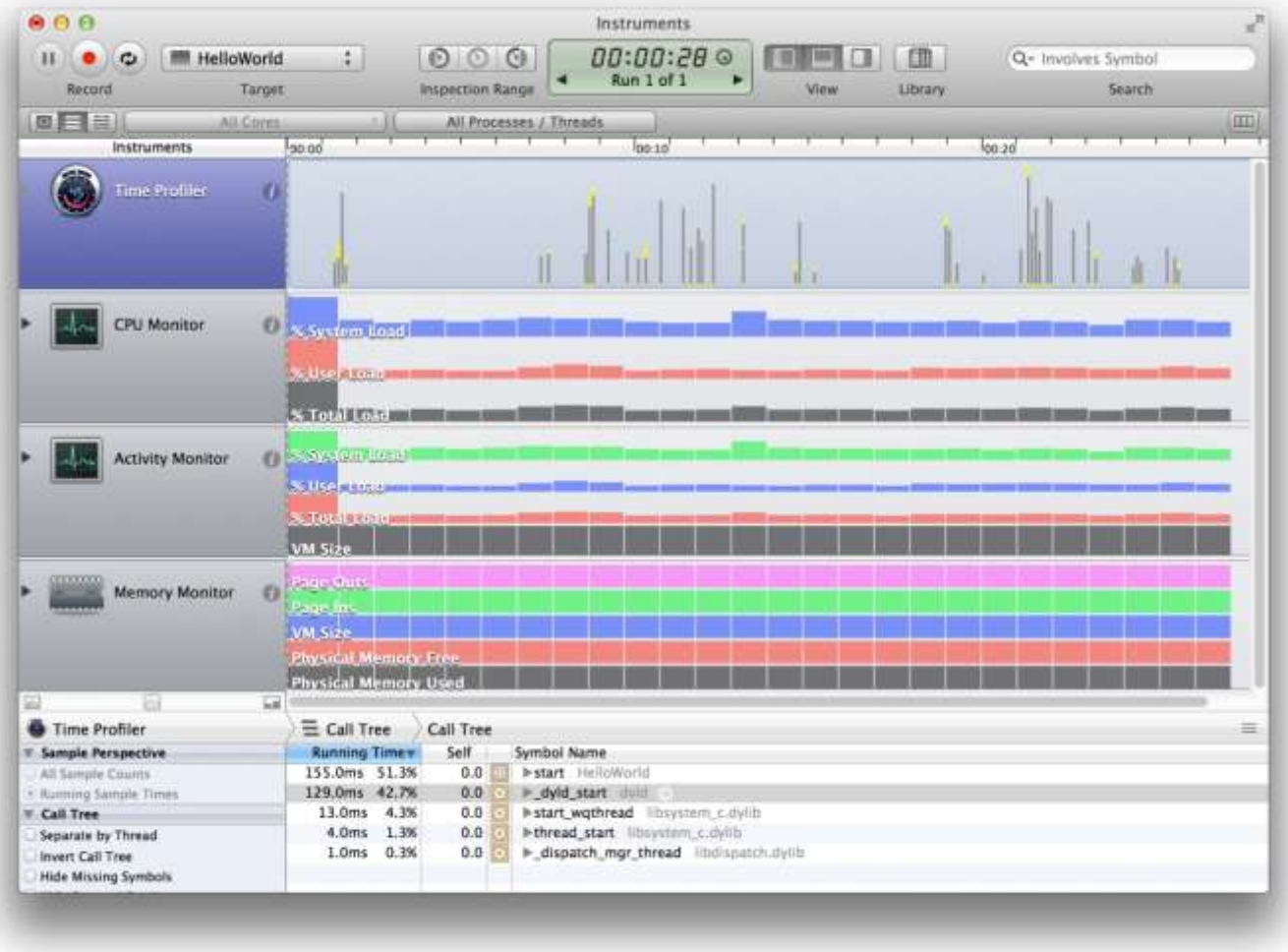


5. 选取“Source Editor Help”>“Catching Mistakes with Fix-it”，可在“Documentation”管理器中查看帮助文章。

6. 点按缩略图图像可播放教学视频。

提高应用程序的性能

要确保软件提供最佳用户体验，请从 Xcode 启动 Instruments 来分析应用程序在 iOS Simulator 或设备上运行时的性能。Instruments 会从运行的应用程序收集数据，并将此类数据呈现在图形时间线中。



您可以收集有关应用程序的内存使用、磁盘活动、网络活动和图形性能的数据，以及其他测量数据。通过统一查看数据，您可以分析应用程序性能的不同方面，以找出可以改进的地方。您可以使应用程序用户界面元素的测试自动化。您还可以对应用程序在不同时间的行为进行比较，以确定您的修改是否提高了应用程序的性能。

▶

开始分析应用程序的性能

1. 从 Xcode 中的 HelloWorld 项目，选取“Product”>“Perform Action”>“Profile Without Building”。
2. 在左栏的 iOS Simulator 下方，点按“All”，查看可用的跟踪模板。
3. 选择“Leaks”模板，并点按“Profile”。

Instruments 应用程序会随运行 HelloWorld 的 iOS Simulator 一起启动。

4. 在 HelloWorld 文本栏中键入您的姓名，点按“Done”按钮关闭键盘，然后点按“Hello”。

5. 选取“iOS Simulator”>“Quit iOS Simulator”，停止记录性能数据。

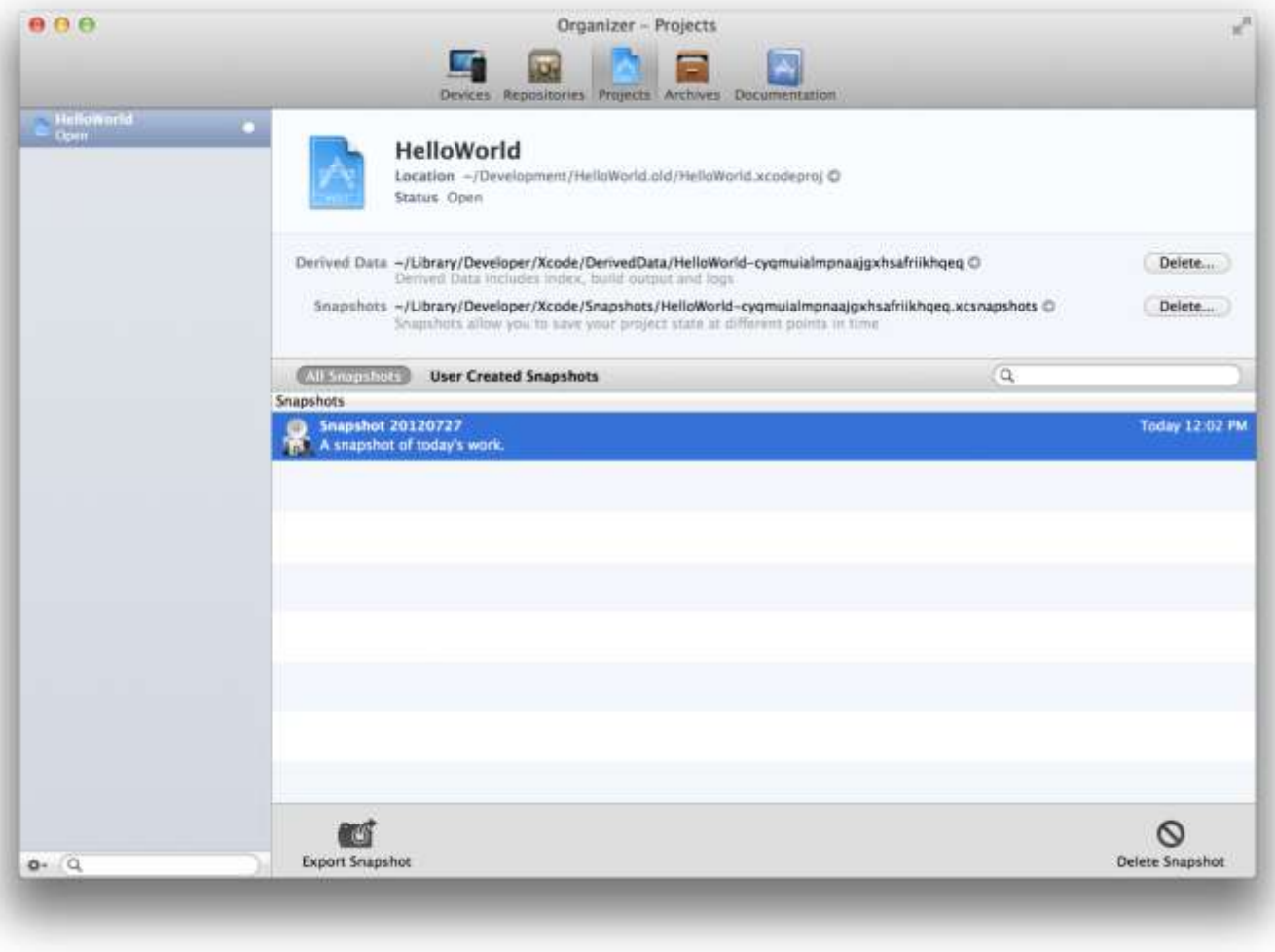
6. 在 Instruments 面板中，点按“Allocations”来检查 HelloWorld 项目的内存分配。

   例如，跟踪面板将内存分配发生点绘制成图形，让您查看整个程序的内存分配状况。（跟踪面板中的尖峰点，标示出潜在的瓶颈；您可以通过预先分出某些内存块，或者降低其他内存块的响应速度，来进行改善。）

管理应用程序的版本

Xcode 快照能够轻易地恢复项目（甚至已删除的项目），可轻松解决因代码更改而出错的问题。快照将项目的当前状态存储在磁盘上，用于以后可能的恢复。Xcode 中的“Projects”管理器会列出快照。

您可以随时根据需要手动创建快照，并且可以将 Xcode 设定为在某些情况下自动创建快照，如在每次生成或每次执行“查找和替换”操作之前。



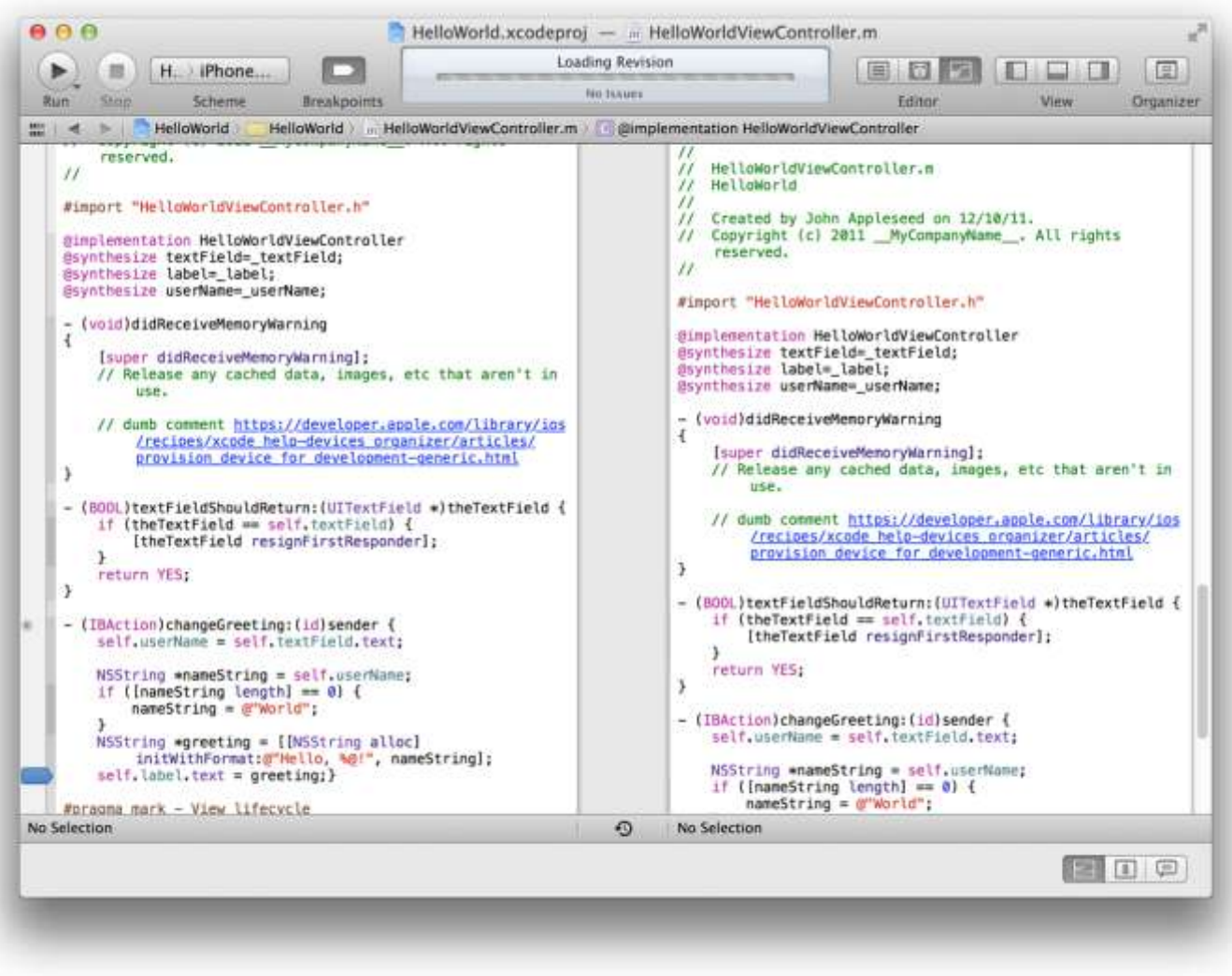
创建和恢复项目的快照

1. 在 HelloWorld 项目打开时，选取“File”>“Create Snapshot”。
2. 在提供的栏中，键入名称和描述。
3. 点按“Create Snapshot”。
4. 要查看快照，请选取“Window”>“Organizer”来显示管理器窗口。
5. 点按“Projects”按钮。

   您应该会看到所有快照的一个列表。

源代码控制管理 (SCM) 可让您跟踪修改，精细程度比快照所允许的更细。（对于团队合作开发，源代码控制管理还可帮助协调工作。）SCM 系统将每个文件的多个版本存储在磁盘上，并且将文件的各个版本的相关元数据储存在 SCM 存储库中。

Xcode 支持两种流行的 SCM 系统：Git 和 Subversion。Xcode 包含版本编辑器，可以轻松地比较任一系统库存的各个文件版本。如果您发现在代码中引入了错误，可以比较文件的最新版本和较早版本（正常运行）之间的更改，有助您追查出错源头。



分发和发布您的应用程序

利用 Xcode，您可在发布前轻松地跟测试员共享应用程序，也使 App Store 的发布工作变得容易。您可使用方案编辑器在 Xcode 创建应用程序的归档，从而开始分发过程。接着可以使用 Xcode 中的“Archives”管理器，与他人共享应用程序，以进行测试。

准备好发布应用程序时，您可使用“Archives”管理器，执行 App Store 发布所需的基本验证测试（这些测试获得通过，可加快应用程序的审批）。接着即可将应用程序直接从 Xcode 发布到 App Store。

您将在本路线图后面的**准备提交到 App Store** 文章中，学习更多有关分发和发布应用程序的知识。

编写 Objective-C 代码

如果您没有为 iOS 或 Mac OS X 编过程序，那就需要开始了解主要的程序设计语言 Objective-C。Objective-C 并不是一种很难的语言，如果您花一点时间学习，相信会慢慢领会到它的优雅之处。Objective-C 程序设计语言使您能进行复杂的、面向对象的编程。通过提供用于定义类和方法的语法，它扩展了标准的 ANSI C 程序设计语言。它还促进类和接口（任何类可采用）的动态扩展。

如果您熟悉 ANSI C，那么下述信息应该能帮助您学习 Objective-C 的基本语法。如果您使用其他面向对象程序设

计语言进行过编程，您会发现许多传统的面向对象概念，例如封装、继承、多态，都出现在 Objective-C 中。如果您不熟悉 ANSI C，在尝试阅读此文章时，最好先阅读一下 C 语言的概述。

Objective-C 语言在 *The Objective-C Programming Language*（Objective-C 程序设计语言）中有完整说明。

Objective-C 是 C 语言的超集

Objective-C 程序设计语言采用特定的语法，来定义类和方法、调用对象的方法、动态地扩展类，以及创建编程接口，来解决具体问题。Objective-C 作为 C 程序设计语言的超集，支持与 C 相同的基本语法。您会看到所有熟悉的元素，例如基本类型（`int`、`float` 等）、结构、函数、指针，以及流程控制结构，如 `if...else` 语句和 `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。

Objective-C 为 ANSI C 添加了下述语法和功能：

- 定义新的类
- 类和实例方法
- 方法调用（称为**发消息**）
- 属性声明（以及通过它们自动合成存取方法）
- 静态和动态类型化
- 块 (block)，已封装的、可在任何时候执行的多段代码
- 基本语言的扩展，例如协议和类别

如果您现在还不太熟悉 Objective-C 的这些方面，也不必担心。随着您读完这篇文章的剩余部分，将会逐渐了解它们。如果您是过程化程序开发人员，不懂面向对象的概念，那么先将对象从本质上视为具有关联函数的结构，可能会有助于理解。这个概念与事实差不多，特别是在运行时实现方面。

除了提供在其他面向对象语言中已有的多数抽象和机制之外，Objective-C 还是一种非常动态的程序设计语言，而且这种动态是其最大优势。这种动态体现在它允许在运行应用程序时（即运行时）才去确定其行为，而不是在生成期间就已固定下来。因此，Objective-C 的动态机制让程序免受约束（编译和链接程序时施加的约束）；进而在用户控制下，将大多数符号解析责任转移到运行时。

类和对象

如同其他大多数面向对象语言那样，Objective-C 中的类，支持数据的封装，并定义对这些数据执行的操作。对象是类的运行时实例。它包含自己的实例变量（由其类声明）的内存副本，以及类方法的指针。您可以采用两步法（称为**分配和初始化**）创建对象。

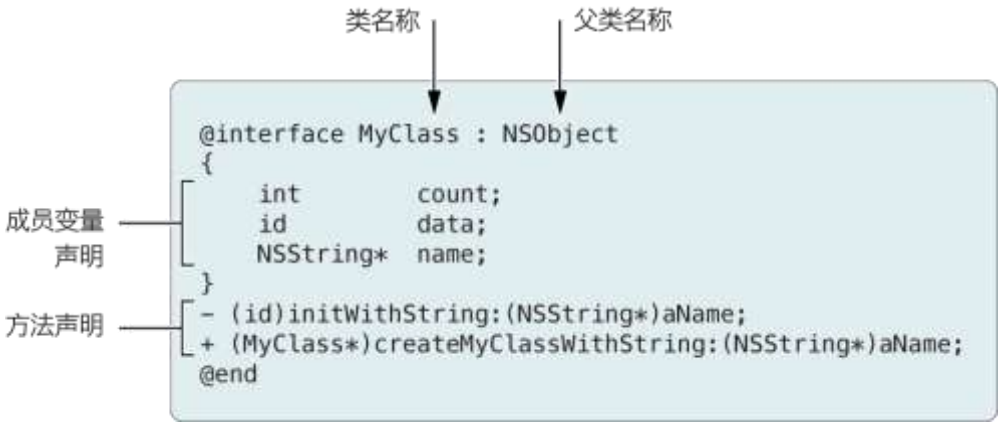
Objective-C 中某个类的规格需要两个不同的部分：接口和实现。接口部分包含类声明，并定义该类的公共接口。如同 C 代码那样，您定义头文件和源代码文件，将公共声明与代码的实现细节分开。（如果其他声明是编程接口的一部分，并且打算专有，您可以将它们放在实现文件中。）这些文件的文件扩展名，列在下表中。

扩展名	源类型
.h	头文件。头文件包含类、类型、函数和常量声明。
.m	实现文件。具有此扩展名的文件可以同时包含 Objective-C 代码和 C 代码。有时也称为**源文件**。
.mm	实现文件。具有此扩展名的实现文件，除了包含 Objective-C 代码和 C 代码以外，还可以包含 C++ 代码。仅当您实际引用您的 Objective-C 代码中的 C++ 类或功能时，才使用此扩展名。

当您想要在源代码中包括头文件时，请在头文件或源文件的前几行之中，指定一个导入 (#import) 指令，#import 指令类似于 C 的#include 指令，不过前者确保同一文件只被包括一次。如果您要导入框架的大部分或所有头文件，请导入框架的包罗头文件 (umbrella header file)，它具有与框架相同的名称。导入（假设的）Gizmo 框架的头文件的语法是：

```
#import <Gizmo/Gizmo.h>
```

下列框图中的语法声明名为 MyClass 的类，它是从基础类（或根类）NSObject 继承而来的。（根类是供其他类直接或间接继承的类。）类声明以编译器指令 @interface 开始，以 @end 指令结束。类名称后面（以冒号分隔），是父类的名称。在 Objective-C 中，一个类只能有一个父类。



在 @interface 指令和 @end 指令之间，编写属性和方法的声明。这些声明组成了类的公共接口。（已声明的属性在"已声明的属性和存取方法"中有介绍。）分号标记每个属性和方法声明的结尾。如果类具有与其公共接口相关的自定函数、常量或数据类型，请将它们的声明放在 @interface ...@end 块之外。
类实现的语法是相似的。它以 @implementation 编译器指令开始（接着是该类的名称），以 @end 指令结束。中间是方法实现。（函数实现应在 @implementation ...@end 块之外。）一个实现应该总是将导入它的接口文件作为代码的第一行。

```
#import "MyClass.h"


@implementation MyClass
```

```
- (id)initWithString:(NSString *)aName
{
    // code goes here
}

+ (MyClass *)myClassWithString:(NSString *)aName
{
    // code goes here
}
@end
```

对于包含对象的变量，Objective-C 既支持动态类型化，也支持静态类型化。静态类型化的变量，要在变量类型声明中包括类名称。动态类型化的变量，则要给对象使用类型 id。您会发现在某些情况下，会需要使用动态类型化的变量。例如，集 (collection) 对象，如数组，在它包含对象的类型未知的情况下，可能会使用动态类型化的变量。此类变量提供了极大的灵活性，也让 Objective-C 程序拥有了更强大的活力。

下面的例子，展示了静态类型化和动态类型化的变量声明：

```
MyClass *myObject1;  // Static typing
id      myObject2;  // Dynamic typing
NSString *userName;  // From Your First iOS App (static typing)
```

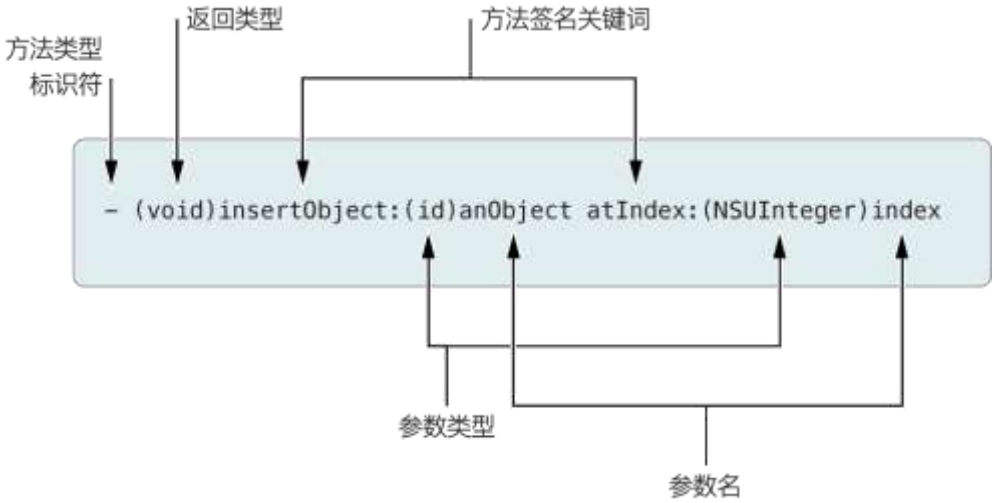
请注意第一个声明中的星号 (*)。在 Objective-C 中，执行对象引用的只能是指针。如果您还不能完全理解这个要求，不用担心。并非一定要成为指针专家才能开始 Objective-C 编程。只需要记住，在静态类型化的对象的声明中，变量的名称前面应放置一个星号。id 类型意味着一个指针。

方法和发消息

如果您不熟悉面向对象编程，则将方法想像成一个规范特定对象的函数，可能会有所帮助。通过将一则消息发送到——或**发消息**给——一个对象，您可调用该对象的一个方法。Objective-C 中有两种类型的方法：实例方法和类方法。

- **实例方法**，由类的实例来执行。换言之，在调用实例方法之前，必须先创建该类的实例。实例方法是最常见的方法类型。
- **类方法**，可由它所在的类直接执行。它不需要对象的实例作为消息的接收者。

方法声明包含方法类型标识符、返回类型、一个或多个签名关键词，以及参数类型和名称信息。以下是 `insertObject:atIndex:` 实例方法的声明。



对于实例方法，声明前面是减号 (-)；对于类方法，对应指示器是加号 (+)。下面的“类方法”将详细介绍类方法。

一个方法的实际名称 (`insertObject:atIndex:`) 是所有签名关键词的串联，包括冒号字符。冒号字符表明有参数存在。在上述示例中，该方法采用两个参数。如果方法没有参数，则省略第一个（也是仅有的一个）签名关键词后面的冒号。

当您想要调用一个方法时，通过给实施该方法的对象发送一则消息来实现。（虽然“发送消息”常用作“调用方法”，但实际上，Objective-C 在运行时才会执行实际地发送。）消息包含方法名称，以及方法所需的参数信息（类型要匹配）。您发送到一个对象的所有消息，都被动态地分派，这样使 Objective-C 类的多态行为更加容易。（**多态性**是指不同类型的对象响应同一消息的能力。）有时被调用的方法，是由接收消息对象的类之超类来实现。

要分派消息，运行时需要一个消息表达式。**消息表达式**使用方括号（`[` 和 `]`）将消息本身（以及任何所需参数）括起来，同时将接收消息的对象放在最左边方括号右侧。例如，要将 `insertObject:atIndex:` 消息发送给 `myArray` 变量保存的对象，您会使用以下语法：

```
[myArray insertObject:anObject atIndex:0];
```

为避免声明大量局部变量来储存临时结果，Objective-C 让您嵌套消息表达式。每个嵌套表达式的返回值，都用作另一个消息的一个参数或接收对象。例如，可以将上个示例中使用的任何变量替换为取回值的消息。因此，如果您具有另一个名为 `myAppObject` 的对象，并且此对象具有访问数组对象的方法以及要插入数组的对象，则可以将上个示例编写为如下形式：

```
[[myAppObject theArray] insertObject:[myAppObject objectToInsert] atIndex:0];
```

Objective-C 还提供用于调用存取方法的点记法语法。**存取方法**获取并设定对象的状态，因此对于封装很重要，是所有对象的重要功能。对象隐藏或**封装**其状态，并显示接口，该接口是访问该状态的所有实例都通用的。使用点记法语法，您可以将上个示例重新编写为如下形式：

```
[myAppObject.theArray insertObject:myAppObject.objectToInsert atIndex:0];
```

您还可以使用点记法语法进行赋值：

```
myAppObject.theArray = aNewArray;
```

此语法只是编写 `[myAppObject setTheArray:aNewArray];` 的另一种方式。在点记法表达式中，您不能使用对动态类型化的对象（类型为 `id` 的对象）的引用。

在**您的首个 iOS 应用程序**中，您已经使用点语法来进行变量赋值：

```
self.userName = self.textField.text;
```

类方法

尽管前几个示例将消息发送到了类的实例，但您也可以将消息发送到类本身。（类是运行时创建的、类型为 `Class` 的对象。）向类发送消息时，您指定的方法必须定义为类方法，而非实例方法。类方法是一种功能，类似于 C++ 中的静态类方法。

您通常这样使用类方法：要么将类方法用作工厂方法创建类的新实例，要么访问与该类关联的一些共享信息。类方法声明的语法，与实例方法声明的语法相同，只是方法类型标识符使用加号 (+)，而非减号。

以下示例说明如何将类方法用作类的工厂方法。在这种情况下，`array` 方法是 `NSArray` 类上的类方法——被 `NSMutableArray` 继承——它分配并初始化类的新实例，并将其返回给您的代码。

```
NSMutableArray *myArray = nil;  // nil is essentially the same as NULL



// Create a new array and assign it to the myArray variable.

myArray = [NSMutableArray array];
```

已声明的属性和存取方法

属性通常是指某些由对象封装或储存的数据。它可以是标志（如名称或颜色），也可以是与一个或多个其他对象的关系。一个对象的类定义一个接口，该接口使其对象的用户能获取并设定所封装属性的值。执行这些操作的方法，称为**存取方法**。

存取方法有两种类型，每个方法都必须符合命名约定。“getter”存取方法返回属性的值，且名称与属性相同。“setter”存取方法设定属性的新值，且形式为 `set`*PropertyName*`:`，其中属性名称的第一个字母大写。正确命名的存取方法是 Cocoa 和 Cocoa Touch 框架的多种技术的关键元素，如键－值编码 (KVC)，它的机制是，通过对象的名称间接访问对象的属性。

Objective-C 提供**已声明的属性**作为一种方便的写法，用于存取方法的声明和实现。在**您的首个 iOS 应用程序**中，您声明了 `userName` 属性：

```
@property (nonatomic, copy) NSString *userName;
```

使用已声明的属性后，就不必为该类中用到的每个属性实现 getter 和 setter 方法。而是使用属性声明，指定您想要的行为。编译器接着可以根据该声明，创建或**合成**实际的 getter 和 setter 方法。已声明的属性减少了您必须编写的样板文件代码量，因此使代码更简洁、更少机会出错。使用已声明的属性或存取方法，来获取和设定各项对象状态。

您在类接口中包括方法声明和属性声明。您在类的头文件中声明公共属性；而在源文件的类扩展中声明专有属性。（有关类扩展的简短说明及其示例，请参阅“协议和类别”。）控制器对象（如委托和视图控制器）的属性通常应该为专有的。

属性的基本声明使用 `@property` 编译器指令，后面紧跟属性的类型信息和名称。您还可以使用自定选项来配置属性，以定义存取方法如何表现、属性是否为弱引用，以及是否为只读。选项位于圆括号中，前面是 `@property` 指令。

以下代码行说明了更多的属性声明：

```
@property (copy) MyModelObject *theObject;  // Copy the object during assignment.

@property (readonly) NSView *rootView;      // Declare only a getter method.

@property (weak) id delegate;              // Declare delegate as a weak reference
```

编译器自动合成所声明的属性。在合成属性时，它创建自己的存取方法，以及“支持”该属性的专有实例变量。实例变量的名称与属性的名称相同，但具有下划线前缀 (_)。只有在对象初始化和取消分配的方法中，您的应用程序应该直接访问实例变量（而不是其属性）。

如果您想要让实例变量采用不同名称，可以绕过自动合成，并明确地合成属性。在类实现中使用 `@synthesize` 编译器指令，让编译器产生存取方法，以及进行特殊命名的实例变量。例如：

```
@synthesize enabled = _isEnabled;
```

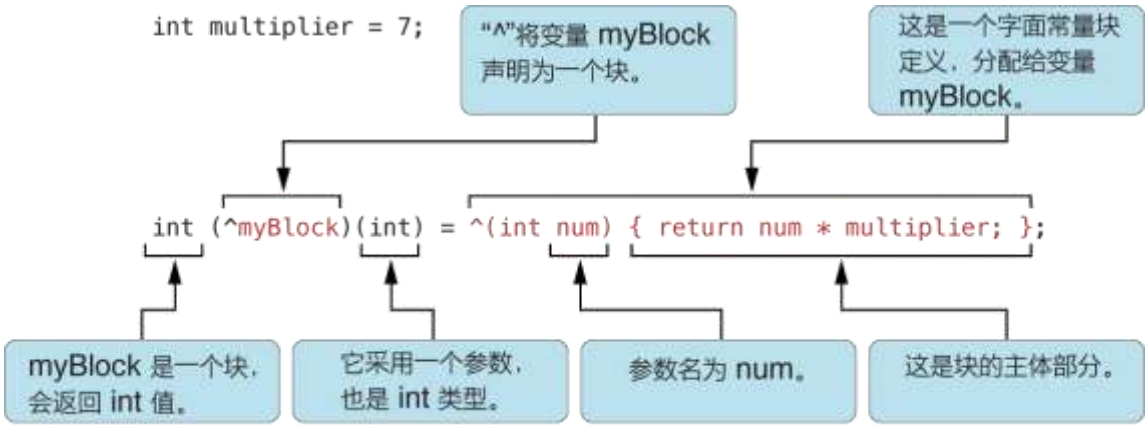
同时，在声明属性时，您可以指定存取方法的自定名称，通常是使 Boolean 属性的 getter 方法遵循约定形式，如下所示：

```
@property (assign, getter=isEnabled) BOOL enabled; // Assign new value, change name of gette
```

块

块是封装工作单元的对象，即可在任何时间执行的代码段。它们在本质上是可移植的匿名函数，可作为方法和函数的参数传入，或可从方法和函数中返回。块本身具有一个已类型化的参数列表，且可能具有推断或声明的返回类型。您还可以将块赋值给变量，然后像调用函数一样调用它。

插入符号 (^) 是用作块的语法标记。块的参数、返回值和正文（即执行的代码）存在其他类似的语法约定。下图解释了该语法，尤其是在将块赋值给变量时。



您接着可以调用块变量，就像它是一个函数一样：

```
int result = myBlock(4); // result is 28
```

块共享局部词法作用范围内的数据。块的这项特征非常有用，因为如果您实现一个方法，并且该方法定义一个块，则该块可以访问该方法的局部变量和参数（包括堆栈变量），以及函数和全局变量（包括实例变量）。这种访问是只读的，但如果使用 `__block` 修饰符声明变量，则可在块内更改其值。即使包含有块的方法或函数已返回，并且其局部作用范围已销毁，但是只要存在对该块的引用，局部变量仍作为块对象的一部分继续存在。

作为方法或函数参数时，块可用作回调。被调用时，方法或函数执行部分工作，并在适当时刻，通过块回调正在调用的代码，以从中请求附加信息，或获取程序特定行为。块使调用方在调用时能够提供回调代码。块从相同的词法作用范围内采集数据（就像宿主方法或函数所做的那样），而非将所需数据打包在“关联”结构中。由于块代码无需在单独的方法或函数中实现，您的实施代码会更简单且更容易理解。

Objective-C 框架具有许多含块参数的方法。例如，Foundation 框架的 `NSNotificationCenter` 类声明以下方法，该方法具有一个块参数：

```
- (id)addObserverForName:(NSString *)name object:(id)obj queue:(NSOperationQueue *)queue
usingBlock:(void (^)(NSNotification *note))block
```

此方法将一个观察者添加到通知中心（通知在**采用设计模式使您的应用程序合理化**中有介绍）。指定名称的一则通知发布时，块被调用以处理该通知。

```
    opQ = [[NSOperationQueue alloc] init];

    [[NSNotificationCenter defaultCenter] addObserverForName:@"CustomOperationCompleted"

            object:nil queue:opQ

       usingBlock:^(NSNotification *notif) {

       // handle the notification

    }];
```

协议和类别

协议可声明由任何类实施的方法，即使实施该协议的那些类没有共同的超类。协议方法定义了独立于任何特定类的行为。协议简单定义了一个由其他类负责实现的接口。当您的类实施了一个协议的方法时，就是说您的类**符合**该协议。

从实践角度而言，一个协议定义了一列方法，这些方法在对象之间建立合约，而无需那些对象成为任何特定类的实例。此合约使那些对象能够相互通信。一个对象想要告知另一个对象它遇到的事件，或者它可能想要寻求有关那些事件的建议。

`UIApplication` 类实现一个应用程序必需的行为。`UIApplication` 类不是强制您对 `UIApplication` 进行子类化，来接收有关应用程序当前状态的简单通知，而是通过调用指定给它的委托对象的特定方法，为您传送那些通知。实施 `UIApplicationDelegate` 协议的对象，可以接收那些通知，并提供适当的响应。

在接口块中，您指定您的类符合或**采用**一个协议，方法是将该协议的名称，放在尖括号 (`<...>`) 中，并且放在它继承的类的名称后面。在**您的首个 iOS 应用程序**中，您指明了采用 `UITextFieldDelegate` 协议，代码行如下：

```
@interface HelloWorldViewController :UIViewController <UITextFieldDelegate> {
```

您无需声明所实现的协议方法。

协议的声明，看起来类似于类接口的声明，只是协议没有父类，并且不定义实例变量（尽管它们可以声明属性）。以下示例展示使用一个方法进行一个简单的协议声明：

```
@protocol MyProtocol

- (void)myProtocolMethod;

@end
```

对于许多委托协议来说，采用一个协议仅仅是实现该协议定义的方法。有些协议要求您明确地声明支持该协议，而协议可以指定必需方法和可选方法。

当您开始探索 Objective-C 框架的头文件时，将很快遇到如下的代码行：

```
@interface NSDate (NSDateCreation)
```

这行代码通过使用圆括号将类别名称括起来的语法约定，声明了该类别。**类别**是 Objective-C 语言的一项功能，可让您扩展类的接口，而无需对类进行子类化。类别中的方法成为类类型的一部分（在程序的作用范围内），而这些方法由类的所有子类继承。您可以将消息发送给类（或其子类）的任何实例，以调用在类别中定义的方法。

您可以将类别用作一种手段，来对头文件内的相关方法声明进行分组。您甚至还可以将不同的类别声明放在不同的头文件中。框架在其所有头文件中使用这些技巧，来达到清晰明确。

您还可以使用称为**类扩展**的匿名类别，在实现 (`.m`) 文件中声明专有属性和专有方法。类扩展看起来类似于类别，只是圆括号之间没有文本。例如，以下是一个典型的类扩展：

```
@interface MyAppDelegate ()

@property (strong) MyDataObject *data;

@end
```

已定义的类型和编码策略

Objective-C 有几个不能用作变量名称的术语，保留用于特殊用途。其中部分术语是以 @ 符号为前缀的编译器指令，如 @interface 和@end。其他保留的术语，包括已定义的类型以及与这些类型相配的字面常量。Objective-C 使用很多已定义的类型和字面常量，这些却不会出现在 ANSI C 中。在某些情况下，这些类型和字面常量替换 ANSI C 相应的类型和字面常量。下表介绍几种重要类型，包括每种类型的允许字面常量。

类型	描述和字面常量
id	动态对象类型。动态类型化的对象和静态类型化的对象的否定字面常量，都是 nil。
Class	动态类类型。其否定字面常量是 Nil。
SEL	选择器的数据类型 (typedef)；此数据类型表示运行时的方法签名。其否定字面常量是 NULL。
BOOL	Boolean 类型。字面常量值是 YES 和 NO。

在错误检查和控制流代码中，通常使用已定义类型和字面常量。在程序的控制流语句中，您可以测试合适的字面常量，来确定如何继续。例如：

```
NSDate *dateOfHire = [employee dateOfHire];

if (dateOfHire != nil) {

    // handle this case

}
```

解释一下此代码，如果表示雇用日期的对象不是 nil（换言之，如果它是一个有效对象），则逻辑在某个方向继续。以下是执行相同分支的简写形式：

```
NSDate *dateOfHire = [employee dateOfHire];

if (dateOfHire) {

    // handle this case

}
```

您甚至可以进一步减少代码行数（假设无需引用 dateOfHire 对象）：

```
if ([employee dateOfHire]) {

    // handle this case

}
```

您可采用大体相同方式，来处理 Boolean 值。在下面的示例中，isEqual: 方法返回一个 Boolean 值。

```
BOOL equal = [objectA isEqual:objectB];

if (equal == YES) {

    // handle this case

}
```

您可以采用与测试是否存在 nil 的代码相同的方式，来缩短此代码。

在 Objective-C 中，您可以将消息发送到 nil，而没有副作用。事实上，完全没有影响，只是如果方法应该返回一个对象，运行时就会返回 nil。只要返回的内容类型化为一个对象，即可保证发送给 nil 的消息的返回值正常运行。

Objective-C 中其他两个重要的保留术语，是 self 和 super。第一个术语 self 是可在消息实现内使用的局部变量，用于引用当前对象；它等同于 C++ 中的 this。您可以用保留字 super 替换 self，但在消息表达式中，只能作为接收者。如果您将消息发送到 self，运行时先在当前对象的类中查找方法实现；如果在那里找不到方法，则在其超类中查找（依此类推）。如果您将消息发送到 super，运行时先在超类中查找方法实现。

self 和 super 的主要用途，都是发送消息。当要调用的方法是由 self 的类实现时，您将消息发送到 self。例如：

```
[self doSomeWork];
```

self 还用于点记法，用于调用由已声明属性合成的存取方法。例如：

```
NSString *theName = self.name;
```

在继承自超类的方法的覆盖（即重新实现）中，通常将消息发送到 super。在这种情况下，被调用方法与被覆盖方法的签名，都是相同的。

掌握基本的编程技能

Foundation 框架，顾名思义，是用于所有 iOS 和 OS X 编程的基础工具箱。您需要熟悉此工具箱，才能成功地在这些平台上开发。

Foundation 定义了几十个用途广泛的类和协议，其中有三种类和协议是极其基础的：

- **根类及相关协议。**根类 NSObject 及其同名协议指定了所有 Objective-C 对象的基本接口和行为。还

有一些协议可以由类采用，以便客户端可以拷贝类的实例并对其状态进行编码。

- **值类。**值类产生的实例称为**值对象**，是一种面向对象的包装器 (wrapper)，用于基本的数据类型（如字符串、数字、日期或二进制数据）。同一值类的实例，如果具有相同的封装值，则视为是相等的。
- **集 (collection) 类。**集类的实例（称为**集**）管理一组的对象。区分特定集类型的关键，在于它让您如何使用所包含的对象。集 (collection) 中的项通常是值对象。

在 Objective-C 编程中，集与值对象极其重要，因为它们经常用作方法的参数和返回值。

根类和 Objective-C 对象

在类层次中，根类不自其他类继承，而类层次中的所有其他类最终都是继承自根类的。NSObject 是 Objective-C 类层次中的根类。其他类从 NSObject 继承了 Objective-C 运行时系统的基本接口。这些类的实例从 NSObject 衍生出它们作为 Objective-C 对象的基本特性。

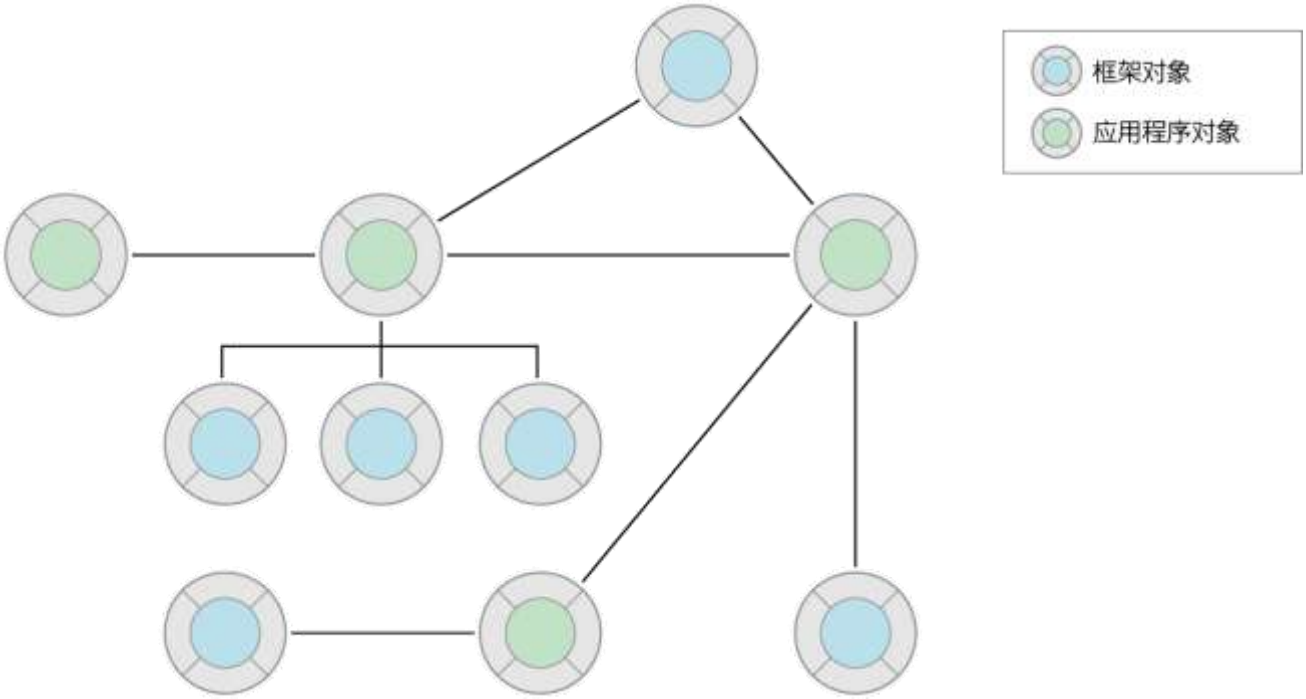
对于 NSObject 实例本身来说，除了是一个简单的对象外，没有其他实质作用。要给您的程序添加任何特定的属性和逻辑，您创建的一个或多个类，必须从 NSObject 继承，或者从任何其他直接或间接继承自 NSObject 的类来继承。

NSObject 采用 NSObject 协议，该协议声明的一些附加方法是所有对象的接口所共用的。此外，NSObject.h（该头文件包含 NSObject 的类定义）包含对 NSCopying、NSMutableCopying 和 NSCoding 协议的声明。当一个类采用这些协议时，它使用对象拷贝和对象编码功能来增加基本对象的行为。模型类（这些类的实例封装和管理应用程序的数据）经常采用对象拷贝和对象编码协议。

NSObject 类及相关协议定义的方法，用于创建对象、导航继承链、询问对象的特征和功能、比较对象、拷贝对象和对对象进行编码。本文章后面的章节描述了绝大多数这些任务的基本要求。

按照对象进行思考

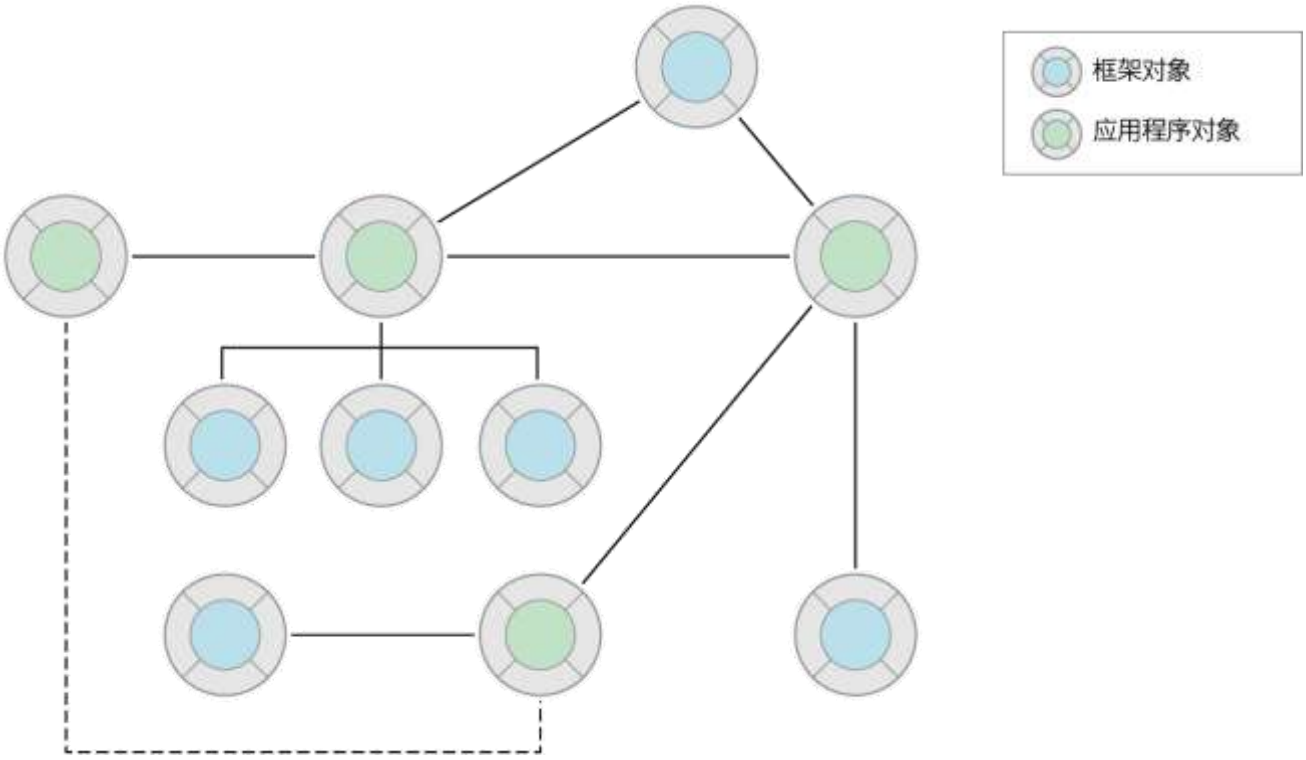
在运行时，每个应用程序都是一个由互相合作的对象组成的网络；这些对象相互通信以完成应用程序的工作。每个对象扮演一个角色，负责至少一项任务，并连接到起码一个另外的对象。（孤立的对象没有多少价值。）如下图所示，对象网络中的对象同时包含框架对象和应用程序对象。应用程序对象是自定子类的实例，通常属于框架超类。一个对象网络的通称为**对象图**。



您通过引用在对象之间建立这些连接或关系。引用有多种语言形式，其中包括实例变量、全局变量，甚至局部变量（在有限的范围内）。关系可以是一对一或一对多，可以表达主从关系或父子关系的概念。这些关系是一个对象访问其他对象、与其他对象通信或控制其他对象的一种手段。被引用的对象成为消息的自然接收者。

应用程序中对象之间的消息传递，对于应用程序的功能连贯性至关重要。就像管弦乐队中的音乐家，应用程序中的每个对象都各自有其角色和特定的行为表现，共同组成了一个应用程序。一个对象可能会显示椭圆表面对轻按操作作出响应，或者管理一组包含数据的对象，或者协调应用程序生命周期中的主要事件。但是为了实现它的作用，它必须能够与其他对象通信。它必须能够发送消息给应用程序中的其他对象，或者能够接收来自其他对象的消息。

对于强耦合对象（即通过直接引用建立相互连接的对象），发送消息轻而易举。但是对于松耦合对象（也就是说，在对象图中相隔很远），应用程序不得不寻找其他的通信方式。Cocoa Touch 和 Cocoa 框架具有许多机制和技巧，使得松耦合对象之间能够通信（正如下图所示）。这些机制和技巧，全部基于设计模式（您稍后将会学到更多内容），使得有效地构建稳固的和可扩展的应用程序成为可能。



创建对象

创建对象时，您通常会先分配再初始化。尽管这是两个分离的步骤，但它们却是紧密相连的。很多类还可以让您通过调用一个类工厂方法来创建对象。

通过分配并初始化对象来创建对象

为了分配对象，您发送 `alloc` 消息给该对象的类，来获得该类的一个“原始”（未初始化）的实例。分配对象时，Objective-C 运行时会从应用程序的虚拟内存，为对象分配足够的内存。除分配内存以外，运行时在分配期间还做了一点别的事，例如将所有实例变量设定为零。

分配原始实例后，您必须立即对它初始化。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。

在框架中，您会发现很多称为**初始化程序**的方法，它们将对象初始化并且具有相似的格式。初始化程序为实例方法，它们以 `init` 开始，并返回一个类型为 `id` 的对象。根类 `NSObject` 声明 `init` 方法供其他所有类继承。其他类可以声明各自的初始化程序，各有专属的关键词和参数类型。例如，NSURL 类声明以下初始化程序：

```
- (id)initFileURLWithPath:(NSString *)path isDirectory:(BOOL)isDir
```

分配对象和将对象初始化时，您将分配调用嵌套在初始化调用内。以上面的初始化程序为例：

```
NSURL *aURL = [[NSURL alloc] initFileURLWithPath:NSTemporaryDirectory() isDirectory:YES];
```

作为一种安全的编程实践，您可以测试返回的对象以验证该对象已经创建。无论在哪个阶段发生对象不可创建的事

情，初始化程序都会返回 nil。尽管 Objective-C 可让您发送消息到 nil 而不会有负面后果（例如，不会抛出异常），您的代码可能没有预期结果，因为它没有调用任何方法。您应该使用初始化程序返回的实例，而不是由 alloc 方法返回的实例。

通过调用类工厂方法创建对象

您还可以通过调用类工厂方法来创建对象——类工厂方法是一种用于分配、初始化实例并返回一个它自己的实例的类方法。类工厂方法很方便，因为它们允许您只使用一个步骤（而不是两个步骤）就能创建对象。它们采用以下形式：

- + (*type*)*className*...（在这里 *className* 不包括任何前缀）

Objective-C 框架的某些类定义了类工厂方法，这些方法与这些类的初始化程序相对应。例如，NSString 声明了以下两个方法：

```
- (id)initWithFormat:(NSString *)format, ...;

+ (id)stringWithFormat:(NSString *)format, ...;
```

以下是您可以如何使用 NSString 这个类工厂的例子：

```
NSString *myString = [NSString stringWithFormat:@"Customer:%@", self.record.customerName];
```

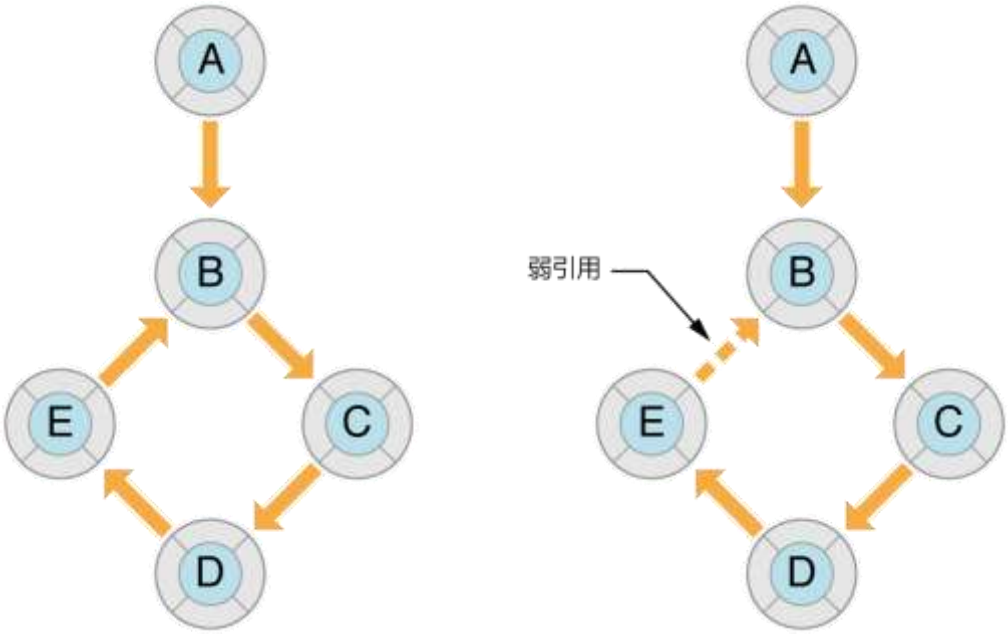
管理对象图以避免内存泄漏

Objective-C 程序中的对象可构成对象图：即通过每个对象与其他对象的关系，或对其他对象的引用，而构成的一个对象网络。对象具有的引用可以是一对一或一对多（通过集对象）。对象图很重要，因为它是对象存在多久的一个要素。编译器检查对象图中的引用强度，并在合适的地方添加保留和释放消息。

注：最近的 Objective-C 运行时版本实现了自动引用计数 (Automatic Reference Counting, ARC)。ARC 使显式内存管理（也就是说，保留和释放对象）变得不再必要。您应该经常在新应用程序项目中使用 ARC，这也是默认的做法。

您通过基本 C 和 Objective-C 结构（如全局变量、实例变量和局部变量）在对象之间形成引用。其中每个结构都附带有隐含的作用范围；例如，被局部变量引用的对象的作用范围，正是声明它的函数块。同样重要的是，对象之间的引用还有强弱之分。强引用表示从属关系；引用对象拥有被引用的对象。弱引用则暗示引用对象不拥有被引用的对象。一个对象的寿命是由它被强引用多少次来决定的。只要对象还存在强引用，就不会释放该对象。

默认情况下，Objective-C 中的引用是强引用。这通常是件好事，让编译器能够管理对象的运行时长，这样对象就不会在您使用时被释放。但是，一不小心，对象之间的强引用会形成一个不能断开的引用链，如下图左侧所示。这种不间断链可以导致运行时不释放任何对象，因为每个对象都具有强引用。结果，强引用循环可导致程序发生内存泄漏。



对于图中的对象，如果断开 A 和 B 之间的引用，则包含 B、C、D 和 E 的子图形将“永远”存在，因为这些对象通过强引用循环绑定在一起。通过引入 E 至 B 的弱引用，您断开了此强引用循环。

对强引用循环的解决之道，在于明智地使用弱引用。运行时同时跟踪对象的弱引用和强引用。一个对象未被强引用时，该对象将被释放，对该对象的所有弱引用都会设定为 `nil`。对于变量（全局、实例和局部），请在变量名称前面使用 `__weak` 限定符，以将引用标记为弱。有关属性，请使用 weak 选项。您应该将弱引用用于以下种类的引用：

- 委托

```
@property(weak) id delegate;
```

- 您将在“设计模式”文章“采用设计模式使您的应用程序合理化”中了解有关委托和目标的信息。
- 未引用顶级对象的 Outlet

```
@property(weak) IBOutlet NSString *theName;
```

- Outlet 是对象之间的连接（或引用），归档在串联图或 nib 文件中，且在应用程序载入该串联图或 nib 文件时得到恢复。串联图或 nib 文件中顶级对象(通常是窗口、视图、视图控制器或其他控制器)的 outlet 应该为 `strong`（默认值，因此无标记）。
- 目标

```
(void)setTarget:(id __weak)target
```

- 在块内对 `self` 的引用

```
__block typeof(self) tmpSelf = self;
```

```
[self methodThatTakesABlock:^ {

    [tmpSelf doSomething];

}];
```

- 一个块对它所捕捉的变量构成强引用。如果在块内使用 self，该块对 self 构成强引用。因此，如果 self 也具有对该块的强引用（通常都有），将产生强引用循环。要避免循环，您需要**在块外**创建对 self 的 weak（或 __block）引用，如以上例子所示。

对象可变性的管理

可变对象是指在创建后，可以更改其状态的对象。您通常通过属性或存取方法进行更改。不可变对象是指在创建后，不可以更改其封装状态的对象。您从 Objective-C 框架的大多数类所创建的实例是可变的，还有少数是不可变的。不可变对象具有以下好处：

- 不可变对象被使用时，其值不会意外更改。
- 对于许多对象而言，如果不可变，可提高其应用性能。

在 Objective-C 框架中，不可变类的实例通常用来封装离散的或缓冲的一组值，如数组和字符串。这些类通常具有可变变体，其名称包含“Mutable”。例如，有 NSString 类（不可变）和 NSMutableString 类。请注意，封装离散值的部分不可变对象（如 NSNumber 或 NSDate）没有可变类变体。

当您期望以增量方式频繁地更改对象的内容时，应使用可变对象，而不使用不可变变体。如果从框架接收到一个对象，其类型被定为不可变对象，请遵循该返回类型；请勿尝试更改该对象。

创建和使用值对象

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。值对象以对象形式表示标量类型。Foundation 框架向您提供了以下类（这些类产生对象，用于字符串、二进制数据、日期与时间、数字以及其他值）：

- NSString 和 NSMutableString
- NSData 和 NSMutableData
- NSDate
- NSNumber
- NSValue

值对象在 Objective-C 编程中很重要。您会频繁遇到这些对象，作为应用程序调用的方法和函数的参数和返回值。通过传递值对象，同一框架的不同部分以至不同的框架都可交换数据。因为值对象表示标量值，您可以在集 (collections) 中使用它们，也可以在任何需要对象的地方使用它们。但是，对象值除这些共性和由此产生的必然性之外，它们在其封装的基本类型上还具有另一项优势：它们让您能采用简单但高效的方式，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。

有时，使用基本类型（即类型为 int（整型）、float 等的值）更高效、更直接。这种情况的一个主要例子是计算数值。因此，NSNumber 和 NSValue 对象在框架方法中，较少用作参数和返回值。但是，许多框架声明了它们自己的数值数据类型，并将这些类型用于参数和返回值；例如 NSInteger 和 CGFloat。您应该在合适的地方使用框架定义的这些类型，因为它们有助于让您的代码不拘泥于底层平台。

使用值对象的基本知识

创建值对象的基本模式，是让您的代码或框架代码从基本类型的数据创建值对象（接着也许在一个方法参数中传递它）。在您的代码中，稍后会从该对象访问被封装的数据。NSNumber 类提供了此方法的最清晰示例：

```
int n = 5; // Value assigned to primitive type

NSNumber *numberObject = [NSNumber numberWithInt:n]; // Value object created from primitive

int y = [numberObject intValue]; // Encapsulated value obtained from value object (y == n)
```

大多数“值”类同时声明初始化程序和类工厂方法来创建实例。某些类（特别是 NSString 和 NSData）还提供初始化程序和类工厂方法，来根据储存在本地或远程文件中的基本数据以及内存中的数据创建实例。这些类还提供补充方法，来将字符串和二进制数据写入文件或 URL 指定的位置。以下示例中的代码调用 initWithContentsOfURL: 方法，根据 URL 对象定位到的文件的内容，创建 NSData 对象；使用数据后，代码会将数据对象写回文件系统：

```
NSURL *theURL = // Code that creates a file URL from a string path...

NSData *theData = [[NSData alloc] initWithContentsOfURL:theURL];

// use theData...

[theData writeToURL:theURL atomically:YES];
```

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。

将值类的实例声明为属性时，应该使用 copy 选项。

字符串和 NSString 字面常量

作为 C 的超集，Objective-C 支持的、用于指定字符串的约定与 C 相同：单个字符使用单引号括起来，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，它们使用 NSString 对象。

在**您的首个 iOS 应用程序**中创建 HelloWorld 应用程序时，创建了一个格式化字符串：

```
NSString *greeting = [[NSString alloc] initWithFormat:@"Hello, %@!", nameString];
```

NSString 类为字符串提供对象包装器，这提供了很多便利，如内建内存管理用于储存任意长度的字符串、支持各

种字符编码（特别是 Unicode），以及提供了 printf 样式的格式化实用工具。因为您通常使用这样的字符串，所以 Objective-C 提供速写记法来根据常量值创建 NSString 对象。要使用此 NSString 字面常量，只需在普通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:@"%d %@", 1, @"String"];

// Create an Objective-C string from a C string.

NSString *fromCString = [NSString stringWithCString:"A C string" encoding:NSASCIIStringEncod
```

NSNumber 字面常量

Objective-C 还提供速写记法来创建 NSNumber 对象，从而无需调用初始化程序或类工厂方法，来创建此类对象。只需在数值前面添加 @符号，并可选择在后面添加一个值类型指示。例如，要创建封装一个整数值和一个双精度值的 NSNumber 对象，您可以编写以下代码：

```
NSNumber *myIntValue    = @32;

NSNumber *myDoubleValue = @3.22346432;
```

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

```
NSNumber *myBoolValue = @YES;

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

您可以创建 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，要创建一个 NSNumber 封装一个浮点值，您可以编写以下代码：

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

日期和时间

NSDate 对象与其他种类的值对象不同，这是因为以时间作为基本值的独特性质。日期对象封装自参考日期算起的时间间隔（以秒为单位）。该参考日期是 GMT 2001 年 1 月 1 日开始的那一刻。

仅通过 NSDate 的实例本身，您无法做些什么。它的确表示某个时刻，但这种表示方式没有由一个区域的日历、时区和时间约定所提供的关联。幸运的是，有一些 Foundation 类可以表示这些概念性实体：

- NSCalendar 和 NSDateComponents——您可以将日期与日历关联，然后从该日期的日历派生时间单位，如年、月、小时和星期几。您还可以执行日历运算。
- NSTimeZone——在日期和时间必须反映某个区域的时区时，可以将时区对象与日历关联。
- NSLocale——区域对象封装了文化和语言约定，包括与时间相关的那些约定。

以下代码片段说明了如何将 NSDate 对象与其他此类对象配合使用，以获取您想要的信息（在本示例中，当前时间按照小时、分钟和秒数打印出来）。

```
NSDate *now = [NSDate date]; // 1

NSCalendar *calendar = [[NSCalendar alloc] initWithCalendarIdentifier:NSGregorianCalendar];

[calendar setTimeZone:[NSTimeZone systemTimeZone]]; // 3

NSDateComponents *dc = [calendar components:(NSHourCalendarUnit|NSMinuteCalendarUnit|

    NSSecondCalendarUnit) fromDate:now];  // 4

NSLog(@"The time is %d:%d:%d", [dc hour], [dc minute], [dc second]); // 5
```

本列表解释代码的每个编号行：

1. 创建表示当前时刻的日期对象
2. 创建表示公历的对象
3. 使用表示时区（在“系统偏好设置”中指定的时区）的对象设定日历对象。
4. 在日历对象上调用 components:fromDate: 方法，传入在步骤 1 中已创建的日期对象。此调用返回一个对象，该对象包含该日期对象的小时、分钟和秒数分量。
5. 将当前小时、分钟和秒数记录到控制台

尽管此示例记录了结果，但在应用程序用户界面上显示日期信息的首选方法，是使用日期格式化程序（NSDateFormatter 类的实例）。您应该经常将合适的类和方法用于日历计算；对于分钟、小时和天等单位，请勿对数值进行硬编码。

创建和使用集

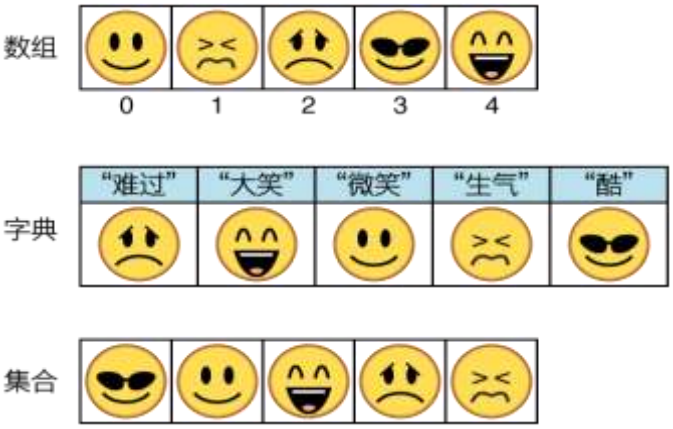
集 (collection) 是一个对象，它以特定方式储存其他对象，并且允许客户端访问那些对象。您通常将集作为方法和函数的参数进行传递，且通常获取集作为方法和函数的返回值。集经常包含值对象，但可以包含任何类型的对象。大多数集都具有对其包含的对象的强引用。

Foundation 框架有几种类型的集，但其中有三种在 Cocoa Touch 和 Cocoa 编程中尤其重要：数组 (array)、字典 (dictionary) 和集合 (set)。这些集的类具有不可变变体和可变变体。可变集允许您添加和移除对象，而不可变集仅可包含用来创建该集的那些对象。所有集都允许您枚举其内容——换言之，允许您依次检查它所包含的每个对

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——数组是多个对象的有序集。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）。
- NSDictionary 和 NSMutableDictionary——字典将其条目储存为键－值对；键是一个唯一标识符，通常是字符串，而值则是您要储存的对象。通过指定键，您可以访问该对象。
- NSSet 和 NSMutableSet——集合储存一组无序对象，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。



由于其储存、访问和性能特征，一种集可能比另一种集更适合某个特定任务。

通过调用 NSArray 和 NSDictionary 的方法，或使用特殊的 Objective-C 容器字面常量和下标技巧，您可以创建数组和字典，并访问其包含的值。以下章节描述了这两种方法。

将对象按某种顺序储存在数组中

数组以有序序列储存对象。因此，一组对象的顺序很重要时，就该使用数组。例如，许多应用程序使用数组向表格视图中的行或菜单中的项目提供内容；索引为 0 的对象对应于第一行，索引为 1 的对象对应于第二行，依此类推。访问数组中对象的时间，比访问集合中对象的时间长。

创建数组

NSArray 类提供许多初始化程序和类工厂方法，用于创建数组和对数组进行初始化，但有几个方法尤其常见和实用。您可以使用 arrayWithObjects:count: 和 arrayWithObjects: 方法（及其对应的初始化程序），从一系列对象创建数组。使用前一种方法时，第二个参数指定第一个参数中的对象数；使用后一种方法时，使用 nil 终止以逗号分隔的对象序列。

```
// Compose a static array of string objects

NSString *objs[3] = {@"One", @"Two", @"Three"};
```

```
// Create an array object with the static array

NSArray *arrayOne = [NSArray arrayWithObjects:&(*objs) count:3];

// Create an array with a nil-terminated list of objects

NSArray *arrayTwo = [[NSArray alloc] initWithObjects:@"One", @"Two", @"Three", nil];
```

创建可变数组时，可以使用 `arrayWithCapacity:`（或 `initWithCapacity:`）方法创建数组。容量参数将有关数组预期大小的提示提供给类，从而使数组在运行时更高效。数组甚至可以超过所指定的容量。

您还可以使用容器字面常量 `@[ ...]` 创建数组，其中方括号之间的项目是以逗号分隔的对象。例如，要创建包含一个字符串、一个数字和一个日期的数组，您可以编写以下代码：

```
NSArray *myArray = @[ @"Hello World", @67, [NSDate date] ];
```

访问数组中的对象

通常，您调用 `objectAtIndex:` 方法访问数组中的对象，方法是指定该对象在该数组中的索引位置（从 0 开始）。

```
NSString *theString = [arrayTwo objectAtIndex:1]; // returns second object in array
```

`NSArray` 提供其他方式来访问数组中的对象或其索引。例如，有 `lastObject`、`firstObjectCommonWithArray:` 和 `indexOfObjectPassingTest:`。

您可以使用下标记号（而非使用 `NSArray` 的方法）访问数组中的对象。例如，要访问 `myArray`（上面已创建）中的第二个对象，您可以编写如下代码：

```
id theObject = myArray[1];
```

与数组有关的另一个常见任务，是对数组中的每个对象执行某种操作——这是称为**枚举**的过程。您通常枚举数组，来决定一个对象或多个对象是否与某个值或条件匹配；如果有一个对象匹配，则使用该对象完成一项操作。您可以采用以下三种方式之一枚举数组：快速枚举、使用块枚举或使用 `NSEnumerator` 对象。顾名思义，快速枚举通常比使用其他技巧访问数组中的对象要快。快速枚举是一项需要特定语法的语言功能：

`for (`*type variable* `in` *array*`){ /* inspect` *variable*`, do something with it */ }`

例如：

```
NSArray *myArray = // get array

for (NSString *cityName in myArray) {

    if ([cityName isEqualToString:@"Cupertino"]) {

        NSLog(@"We're near the mothership!");
```

```
        break;

    }

}
```

数个 NSArray 方法使用块来枚举数组，其中最简单的是 enumerateObjectsUsingBlock:。此块具有三个参数：当前对象、其索引和引用的 Boolean 值（设置为 YES 时终止枚举）。此块中的代码执行的工作，与快速枚举语句中大括号内的代码完全相同。

```
NSArray *myArray = // get array

[myArray enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {

    if ([obj isEqual:@"Cupertino"]) {

        NSLog(@"We're near the mothership!");

        *stop = YES;

    }

}];
```

管理可变数组

NSArray 具有其他方法用于給数组排序、搜索数组和在数组中的每个对象上调用方法。

通过调用 addObject: 方法，可将对象添加到可变数组；对象放在数组末尾。您也可以使用 insertObject:atIndex:，将对象放在可变数组中的特定位置。通过调用 removeObject: 方法或 removeObjectAtIndex: 方法，可以从可变数组中移除对象。

您还可以使用下标记号，将对象插入可变数组中的特定位置。

```
NSMutableArray *myMutableArray = [NSMutableArray arrayWithCapacity:1];

NSDate *today = [NSDate date];

myMutableArray[0] = today;
```

将键－值对储存在字典中

您使用字典将对象储存为键－值对，即标识符（键）和对象（值）对。字典是无序集，因为键－值对可采用任何顺序。尽管键几乎可以是任何内容，但通常是描述值的字符串，
如 NSFileModificationDate 或 UIApplicationStatusBarFrameUserInfoKey（为字符串常量）。存在

公共键时，字典是在对象之间传递任何种类的信息的绝佳方式。

创建字典

NSDictionary 类通过初始化程序和类工厂方法，向您提供多种创建字典的方法，但是有两个类方法特别常用：dictionaryWithObjects:forKeys: 和 dictionaryWithObjectsAndKeys:（或它们对应的初始化程序）。使用前一种方法时，您传入对象数组和键数组；键在位置上与其值匹配。使用第二种方法时，您指定第一个对象值及其键，第二个对象值及其键，依此类推；您使用 nil 标记此对象序列的结尾。

```
// First create an array of keys and a complementary array of values

NSArray *keyArray = [NSArray arrayWithObjects:@"IssueDate", @"IssueName", @"IssueIcon", nil]

NSArray *valueArray = [NSArray arrayWithObjects:[NSDate date], @"Numerology Today",

    self.currentIssueIcon, nil];

// Create a dictionary, passing in the key array and value array

NSDictionary *dictionaryOne = [NSDictionary dictionaryWithObjects:valueArray forKeys:keyArra

// Create a dictionary by alternating value and key and terminating with nil

NSDictionary *dictionaryTwo = [[NSDictionary alloc] initWithObjectsAndKeys:[NSDate date],

    @"IssueDate", @"Numerology Today", @"IssueName", self.currentIssueIcon, @"IssueIcon", nil
```

如同数组，创建 NSDictionary 对象时，您可使用容器字面常量 @{*key* :*value*, …}，其中“…”表示任意数量的键－值对。例如，以下代码创建含三个键－值对的不可变字典对象：

```
NSDictionary *myDictionary = @{

   @"name" :NSUserName(),

   @"date" :[NSDate date],

   @"processInfo" :[NSProcessInfo processInfo]

};
```

访问字典中的对象

您通过调用 objectForKey: 方法并将键指定为参数，访问字典中的对象值。

```
NSDate *date = [dictionaryTwo objectForKey:@"IssueDate"];
```

您还可以使用下标访问字典中的对象。键出现在方括号内（方括号紧接在字典变量后面）。

```
NSString *theName = myDictionary[@"name"];
```

管理可变字典

您通过调用 `setObject:forKey:` 和 `removeObjectForKey:` 方法，在可变字典中插入和删除项目。`setObject:forKey:` 方法替换给定键的任何现有值。这些方法都很快捷。

您还可以使用下标，将键－值对添加到可变字典中。键下标位于赋值左侧，而值位于右侧。

```
NSMutableDictionary *mutableDict = [[NSMutableDictionary alloc] init];

mutableDict[@"name"] = @"John Doe";
```

将无序对象储存在集合中

集合是类似于数组的一组对象，只是其中包含的项目是无序的（而数组是有序的）。您通过枚举集合中的对象，或者将过滤器或测试应用到集合，来随机访问集合中的对象（使用 `anyObject` 方法），而不是按索引位置或通过键访问它们。

尽管集合对象在 Objective-C 编程中不如字典和数组那么常用，但它们在某些技术中是重要的集类型。在 Core Data（一种数据管理技术）中，当您声明对多关系的属性时，属性类型应该是 `NSSet` 或 `NSOrderedSet`。集合对于 UIKit 框架中的原生触摸事件处理也很重要，例如：

```
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {

    UITouch *theTouch = [touches anyObject];

    // handle the touch...

}
```

有序集合是集合基本定义的一个例外。在有序集合中，集合中的项目顺序很重要。有序集合中测试成员资格比数组中要快。

运行时验证对象功能

内省是 Objective-C 和 `NSObject` 类的强大且实用的功能，使您能在运行时了解有关对象的某些东西。您因此可避免代码出错，例如将消息发送到无法识别它的对象，或者误以为对象从一个给定的类继承。

对象会在运行时透露三种重要信息：

- 它是否是特定类或其子类的实例
- 它是否响应消息
- 它是否遵守协议

发现对象是否是特定类或其子类的实例

要发现对象是否是某类或其子类的实例，请在对象上调用 `isKindOfClass:` 方法。当应用程序需要发现其响应的消息（实现的或继承的），它有时进行以上的检查。

```
static int sum = 0;

for (id item in myArray) {

    if ([item isKindOfClass:[NSNumber class]]) {

        int i = (int)[item intValue];

        sum += i;

    }

}
```

`isKindOfClass:` 方法将类型为 `Class` 的对象视为参数；要获取此对象，请在类符号上调用 `class` 方法。接着评估此方法返回的 Boolean 值，并继续相应的操作。

`NSObject` 会声明其他方法来发现有关对象继承的信息。例如，`isMemberOfClass:` 方法告诉您，对象是否是特定类的实例；而 `isKindOfClass:` 告诉您，对象是否是该类或任何其后代类的成员。

发现对象是否响应消息

要发现一个对象是否响应一则消息，请在该对象上调用 `respondsToSelector:` 方法。应用程序代码通常验证一个对象响应一则消息后，才将消息发送给该对象。

```
if ([item respondsToSelector:@selector(setState:)]){

    [item setState:[self.arcView.font isBold] ?NSOnState :NSOffState];

}
```

`respondsToSelector:` 方法将选择器视为其参数。选择器是一种 Objective-C 数据类型，用于方法的运行时标识符 (runtime identifiers)；您使用 `@selector` 编译器指令指定选择器。在您的代码中，评估此方法返回的 Boolean 值，并继续相应的操作。

要识别对象响应的消息，调用 `respondsToSelector:` 通常比评估类的类型更有用。例如，一个类的较新版本可

能实现以前版本没有的方法。

发现对象是否遵守协议

要发现对象是否遵守协议，请在对象上调用 conformsToProtocol: 方法。

```
- (void) setDelegate:(id __weak) obj {

    NSParameterAssert([obj conformsToProtocol:

        @protocol(SubviewTableViewControllerDataSourceProtocol)]);

    delegate = obj;

}
```

conformsToProtocol: 方法将协议的运行时标识符视为参数，您使用 @protocol 编译器指令指定此标识符。评估此方法返回的 Boolean 值，并继续相应的操作。请注意，对象可以遵守协议，而不实现其可选方法。

比较对象

您可以使用 isEqual: 方法比较两个对象。让接收消息的对象与传入的对象进行比较；如果相同，该方法返回 YES。例如：

```
BOOL objectsAreEqual = [obj1 isEqual:obj2];

if (objectsAreEqual) {

    // do something...

}
```

请注意，对象相等与对象相同是有分别的。对于后者，请使用相同运算符 == 测试两个变量是否指向同一个实例。

当您比较同一个类的两个对象时，到底在比较什么？这取决于类。根类 NSObject 将指针相等用作比较基础。任何级别的子类都可覆盖其超类的实现，以让比较基于类特定标准，如对象状态。例如，如果一个假设的 Person 对象和另一个 Person 对象的名字、姓氏和出生日期属性都相符，则这两个对象可能相等。

Foundation 框架的值和集类，声明的比较方法为 isEqualTo*Type*: 格式，其中 *Type* 是类类型减去 NS 前缀，如 isEqualToString:和 isEqualToDictionary:。此比较方法会假定传入的对象属于给定类型，否则会引发异常。

拷贝对象

您通过将 copy 消息发送给对象，以制作对象的副本。

```
NSArray *myArray = [yourArray copy];
```

要拷贝，接收对象的类必须遵守 NSCopying 协议。如果想要对象可供拷贝，必须采用并实施此协议的 copy 方法。

有时，当您想要确保对象的状态在使用时不会更改，会拷贝从程序的其他地方获取的对象。

拷贝行为是特定于某一个类的，具体取决于实例的特定性质。大多数类实现深拷贝，即复制所有实例变量和属性；部分类（如集类）实现浅拷贝，即仅复制对这些实例变量和属性的引用。

具有可变变体和不可变变体的类也声明 mutableCopy 方法，来创建对象的可变副本。例如，如果在 NSString 对象上调用 mutableCopy，您会获得 NSMutableString 的实例。

研究主要框架

框架是一个目录，包含了共享资源库，用于访问该资源库中储存的代码的头文件，以及图像、声音文件等其他资源。共享资源库定义应用程序可以调用的函数和方法。

iOS 提供了许多可在应用程序开发中使用的框架。要使用一个框架，请将它添加到项目，以便应用程序可以链接到它。大多数应用程序都链接到 Foundation、UIKit 和 Core Graphics 框架。根据您为应用程序选取的模板，可能也包括其他框架。如果一组核心框架无法满足应用程序的要求，您总是可以将其他框架添加到项目。

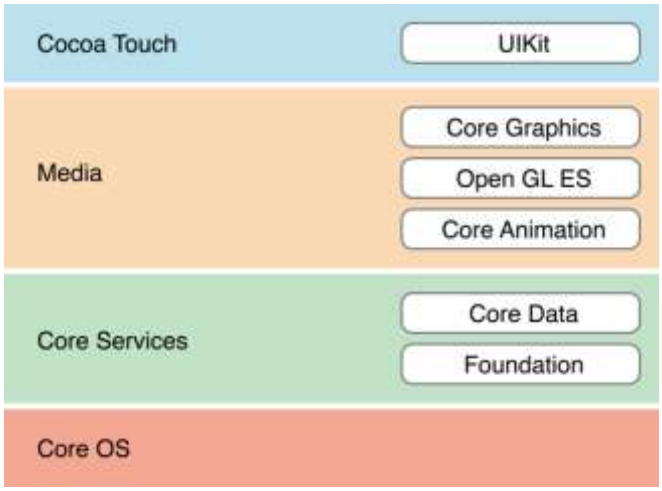
▶

查看 HelloWorld.xcodeproj 中包括的框架

1. 在 Xcode 中打开 HelloWorld.xcodeproj 项目（如果尚未打开的话）。您早前已在教程**您的首个 iOS 应用程序**中创建了此项目。
2. 在项目导航器中，点按“Frameworks”文件夹旁边的展示三角形，以打开此文件夹。

   您应该看到 UIKit.framework、Foundation.framework 和 CoreGraphics.framework。
3. 点按框架旁边的展示三角形，然后点按“Headers”文件夹旁边的展示三角形，可以查看框架中的头文件。

每个框架都属于 iOS 系统的一个层。每个层都建立在它的下层之上。尽可能使用较高级的框架，而非较低级的框架。较高级的框架向较低级的结构提供面向对象的抽象。



iOS 应用程序基于 Foundation 和 UIKit 框架

开始编程时，您主要使用 Foundation 和 UIKit 框架，因为它们满足大多数应用程序开发的需求。

Foundation 框架为所有应用程序提供基本的系统服务

您的应用程序以及 UIKit 和其他框架，都建立在 Foundation 框架的基础结构之上。Foundation 框架提供许多基本的对象类和数据类型，使其成为应用程序开发的基础。它还制定了一些约定（用于取消分配等任务），使您的代码更加一致，可再用性更好。

使用 Foundation：

- 创建和管理集，如数组和字典
- 访问储存在应用程序中的图像和其他资源
- 创建和管理字符串
- 发布和观察通知
- 创建日期和时间对象
- 自动发现 IP 网络上的设备
- 操控 URL 流
- 异步执行代码

在**您的首个 iOS 应用程序**中，您就使用了 Foundation 框架。例如，您使用了 `NSString` 类的实例，将用户的输入储存在 `userName` 中。您还使用了 Foundation 实例方法 `initWithFormat:`，创建问候语字符串。

UIKit 框架提供的类，可用于创建基于触摸的用户界面

所有 iOS 应用程序都基于 UIKit。没有这个框架，就无法交付应用程序。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。UIKit 还通过管理屏幕上显示的内容，来组织复杂的应用程序。

使用 UIKit：

- 构建和管理用户界面
- 处理基于触摸和运动的事件
- 显示文本和网页内容
- 优化应用程序以实现多任务
- 创建自定用户界面元素

在**您的首个 iOS 应用程序**中，您使用了 UIKit。检查应用程序如何启动时，您看到了 `UIApplicationMain` 函数，它创建了 `UIApplication` 类（处理传入的用户事件）的一个实例。您实现了 `UITextFieldDelegate` 协议，以便在用户轻按“Done”键时，让键盘消失。事实上，您使用了 UIKit 中的 `UITextField`、`UILabel` 和 `UIButton` 类，创建了整个界面。

您应该了解的其他重要框架

Core Data、Core Graphics、Core Animation 和 OpenGL ES 框架，是对于应用程序开发很重要的高级技术，因此需要花时间来学习和掌握。

Core Data 框架管理应用程序的数据模型

Core Data 管理对象图。借助 Core Data，您可以创建模型对象（称为被管理的对象）。您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。

使用 Core Data：

- 存储对象和从储存处取回对象
- 支持基本的撤销/重做
- 自动验证属性值
- 对内存中的数据进行过滤、分组和整理
- 使用 `NSFetchedResultsController` 管理表格视图中的结果
- 支持基于文稿的应用程序

Core Graphics 框架帮助您创建图形

高质量的图形，是所有 iOS 应用程序的一个重要组成部分。在 iOS 中创建图形的最简易便捷方法，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。UIKit 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。尽可能地使用 UIKit（较高级的框架），而非 Core Graphics（较低级的框架）。

当您想要编写在 iOS 和 OS X 之间直接共享的绘图代码时，使用 Core Graphics。Core Graphics 框架也称为 *Quartz*，它在这两个平台上几乎相同。

使用 Core Graphics：

- 制作基于路径的绘图
- 使用边缘模糊化渲染
- 添加渐变、图像和颜色
- 使用坐标空间变换
- 创建、显示和解析 PDF 文稿

Core Animation 可让您制作高级动画和视觉效果

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，而无需使用低级的图形 API，如 OpenGL ES 等。

使用 Core Animation：

- 创建自定动画
- 给图形添加时序功能
- 支持关键帧动画
- 指定图形布局约束
- 将多层更改分组为原子更新

OpenGL ES 框架提供 2D 和 3D 绘图工具

OpenGL ES 支持基础的 2D 和 3D 绘图。Apple 实施的 OpenGL ES 标准，与设备硬件紧密协作，为全屏幕游戏类应用程序提供很高的帧速率。

使用 OpenGL ES：

- 创建 2D 和 3D 图形
- 制作更复杂的图形，如数据可视化、飞行模拟或视频游戏。
- 访问底层图形硬件

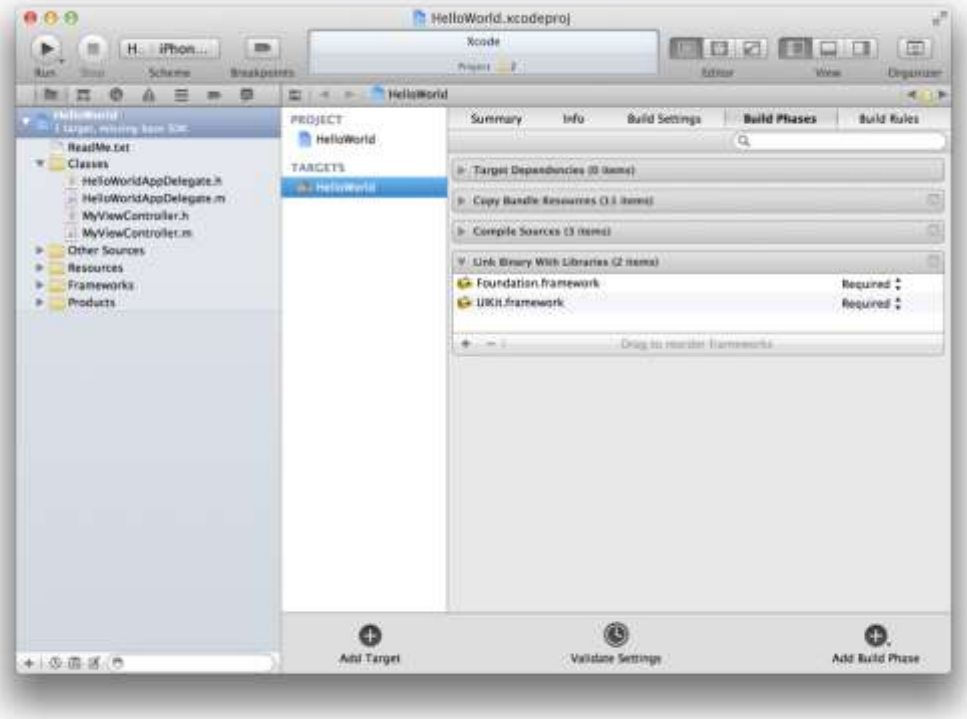
了解 iOS API 和 OS X API 之间的异同

如果您是 Mac 开发者，您会发现 Cocoa 和 Cocoa Touch 应用程序都基于类似的技术。它们具有共同的 API，使得从 Cocoa 迁移更简单。事实上，部分框架是相同（或几乎相同）的，例如 Foundation 和 Core Data。但是，其他框架与其 OS X 相应的框架有差异。AppKit 和 UIKit 尤其如此。因此，在将 Mac 应用程序迁移到 iOS 时，必须替换大量界面相关的类，以及与这些类相关的代码。

有关这两个平台之间异同的更多信息，请参阅 *iOS Technology Overview*（iOS 技术概述）中的“Migrating from Cocoa”（从 Cocoa 迁移）。

根据需要将其他框架添加到项目

在应用程序中还可以使用许多其他框架。决定要使用一个尚未包含的框架时，请将该框架添加到项目，以便应用程序可以链接到它。



▶

将 HelloWorld.xcodeproj 链接到其他框架

1. 在 Xcode 中打开 `HelloWorld.xcodeproj` 项目（如果尚未打开的话）。
2. 在项目导航器中，点按 HelloWorld 项目，显示项目编辑器。
3. 在“Targets”列表中，点按 HelloWorld，将 HelloWorld 指定为目标，以便向其添加框架。
4. 点按项目编辑器顶部的“Build Phases”。
5. 点按展示三角形，打开“Link Binary With Libraries”部分。
6. 点按添加按钮 (+) 添加框架。
7. 从列表中选择框架，并点按“Add”。

有关框架的完整列表，或要了解有关框架的更多信息，请参阅 *iOS Technology Overview*（iOS 技术概述）。

将代码与框架整合

您为 iOS 或 OS X 开发应用程序时，不会是完全孤立的。您将沿用 Apple 和其他人的劳动成果，沿用他们在 Objective-C 框架中开发和收集的类。框架是运行时可供多个进程共享的类资源库；它包含支持采用该资源库进行软件开发的资源。Cocoa 和 Cocoa Touch 框架，为您提供了一组相互依赖的类，它们共同工作，以构成应用程序的一部分（通常是相当重要的一部分）。

针对您要编写的程序，您可以按需从一个 C 函数库中选用函数，并确定何时调用它们。另一方面，框架会将某种设计强加于您的程序，或至少强加于您的程序要尝试解决的某个问题空间。使用面向对象的框架时，可以调用框架中的类的方法，来执行程序的大部分工作，就和过程化程序中一样。但是，您也还需要定制泛型框架行为，并通过实施一些方法，让框架在适当时候调用这些方法，对框架行为进行调整，以满足您的需要。这些方法作为钩子 (hook)，将您的代码引入到框架所施加的结构中，增加了能使您的应用程序特征化的行为。

以下部分探索框架代码和应用程序代码之间的关系。

应用程序由事件驱动

通过考虑应用程序启动时会发生什么，可以一窥您编写的代码和框架代码之间的关系。基本上，应用程序建立一组核心对象，然后将控制权移交给这些对象。随着程序的运行，越来越多的对象会被创建，但是程序最初所需的，却只是足以处理初始任务的结构（即足够的核心对象网络）。有两个主要任务：

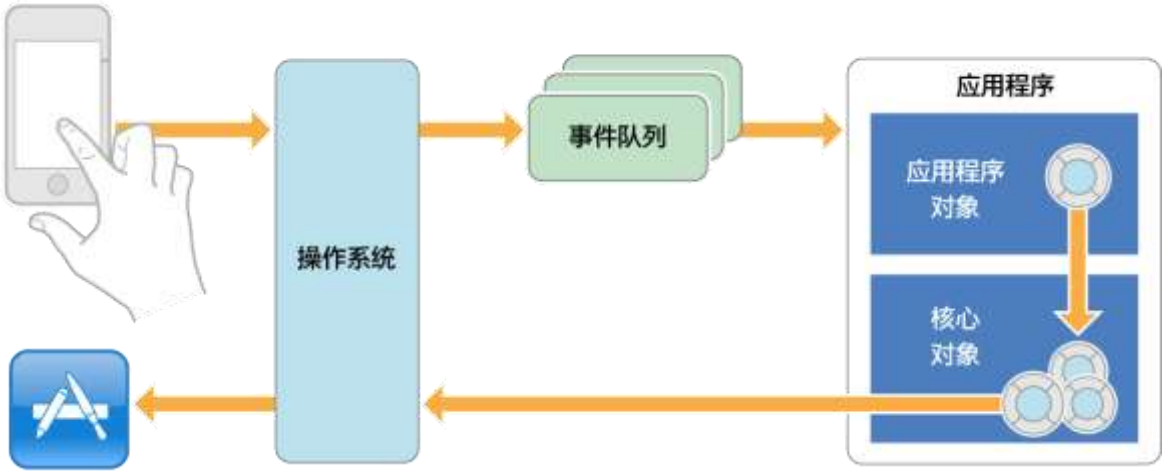
- 绘制应用程序的初始用户界面。
- 处理用户与该用户界面互动时收到的事件。

初始用户界面显示在屏幕上之后，应用程序由外部事件驱动。最重要的外部事件源自用户（例如轻触按钮）。操作系统将这些事件及其相关信息一起报告给应用程序。该应用程序（由您的代码和框架代码组成）处理事件，并相应地更新用户界面。

应用程序获取事件，并作出响应（通常是通过绘制部分用户界面），然后等待下一个事件。应用程序不断获取事件，一个接一个，只要用户或其他源（例如计时器）发起事件。从应用程序启动到终止，它所做的几乎所有事情，都是

由用户的操作，以事件的形式来驱动。

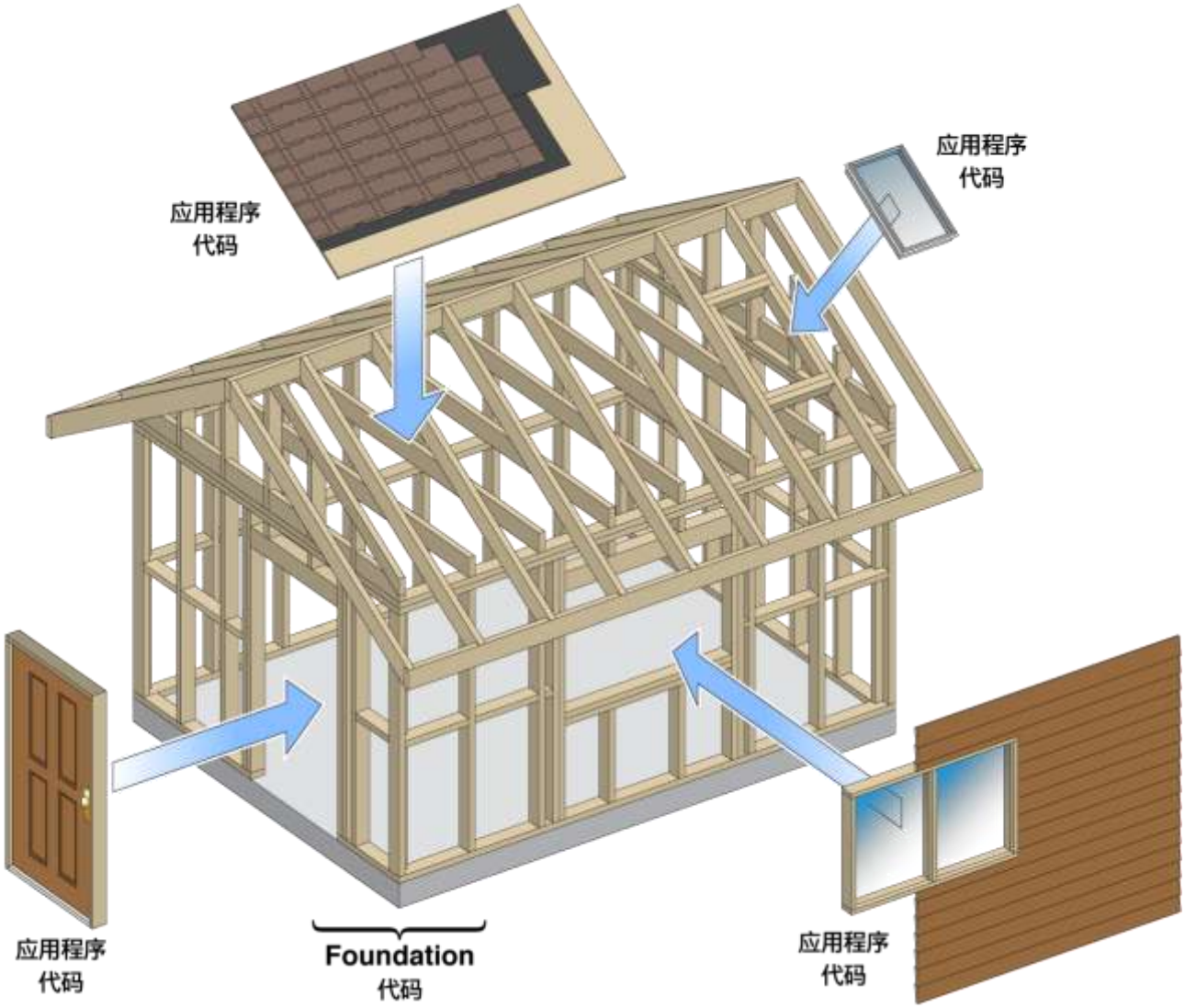
获取事件和对事件做出响应的机制，就是**主要事件循环**。在应用程序的一组核心对象中，有一个对象（即全局应用程序对象）负责管理主要事件循环。它获取事件，将事件分派给该对象或能最好地处理事件的对象，然后获取下一个事件。下图说明了 iOS 中 Cocoa Touch 应用的主要事件循环。



使用面向对象的框架

Cocoa 和 Cocoa Touch 框架不光是将提供各自服务的各个类混杂在一起的杂物袋。这些面向对象的框架是类的集合，每个类构建一个问题空间，并提供完整的解决方案。框架提供的不是根据需要可以使用的离散服务（如函数库），而是制订并实现整个应用程序的结构，您编写的代码则必须适应该结构。因为此应用程序结构是通用的，您可以使其特殊化，以满足特定应用的要求。与其说是设计一个程序，并插入资源库函数，不如说是将应用程序代码，插入到框架提供的设计中。

要使用框架，您必须接受它定义的应用程序结构，然后根据需要，尽可能多的使用和定制它的类，将特定的应用进行改造，以适合该结构。框架中的类相互依赖构成一个整体，并不是单个的。乍一看，在应用程序中要求您的代码适应框架的结构，似乎有限制性。事实刚好相反。框架为您提供了很多方式，来修改和扩展其通用行为。它只要求您接受这样的观点，所有应用都以同样的基本方式来运行，因为这些应用全部基于同样的结构。从广义的隐喻层面而言，Objective-C 框架就像房屋的框架，而应用程序代码就好比大门、窗户、壁板和其他元素，是这些东西让房子与众不同。



从使用框架和将代码与框架整合的观点看，有两种总的类：

- **现成的。**一些类定义了现成对象，也就是可以随时使用的对象。您只需要创建类的实例，然后根据需要使用这些实例。
- **通用的（或泛型）。**对于泛型框架类，您可以（在有些情况下**必须**）创建它们的子类，并覆盖某些方法的实现。通过对它们进行子类化，将代码引入到应用程序结构中。框架会在适当的时刻，调用子类的方法。

将泛型框架类子类化，是将程序特定的代码，整合到框架所提供的结构中的主要技巧。但这并不是唯一技巧，而且在很多情况下，也不是首选的技巧。在后面的文章中，您将会学习到 Cocoa Touch 和 Cocoa 框架，也包括架构和机制，二者都基于设计模式，可以实现框架对象和自定对象间的更大合作与协调。

创建子类时，需要做出两个基本决定：决定从哪个类（超类）继承和要覆盖该类的哪个方法。下面的部分探索做出这些决定的来龙去脉。

继承 Cocoa 或 Cocoa Touch 类

框架（如 UIKit）定义的结构，因为是泛型结构，可供很多类型的应用程序共享。因为结构是泛型的，所以有一些框架类是抽象的或有意不完整的，这也不足为奇。这样的类通常实现大量的常见代码，但却让工作的重要部分，要么未完成，要么以安全的默认方式完成。

将特定于应用的行为，添加到框架的主要方式，就是创建其中一个框架类的自定子类。子类填补了其超类中的这些空隙，提供了框架类所缺少的部分。自定子类的实例，占据其在框架所定义的对象网络中的位置，也继承框架与其他对象合作的能力。要让应用程序做任何有用的事情，它必须创建至少一个子类，大多数情况下需要创建很多子类。

下面的讨论探索有关子类化的一些决定和策略，并说明一些总的要求。该讨论并没有详细介绍**如何**创建子类。*The Objective-C Programming Language*（Objective-C 程序设计语言）的“Defining a Class”（定义类）描述了该技巧。

何时创建子类

子类化是这样一个过程：重新使用现有的类并将其特殊化，以满足您的需要。有时候一个子类所需要做的，仅是覆盖单个继承的方法，并让该方法做一些与原来的行为稍微不同的事情。其他子类可能添加一个或两个属性到其超类中（作为实例变量），然后定义方法来访问并操作这些属性，将它们整合到超类行为。

子类化从确定子类来自哪个框架类开始。以下是供您参考的一些考虑因素：

- **了解框架。**您应该熟悉框架中的每个类的目的和功能。首先，阅读开发者资源库中框架的介绍，然后扫描框架的类的列表。可能存在着某个类，已经完成了您想要执行的操作。如果找到了一个类，可以完成**几乎**所有您想要执行的操作，您算走运了。该类很大可能会是您的自定类的超类。

  例如，创建**您的首个 iOS 应用程序**时，遇到了 `UIViewController` 和 UIKit 框架的其他类。要找出关于这些类的更多信息，可以执行以下操作：

  1. 在 Xcode 中，选取“Window”>“Organizer”。
  2. 点按工具栏中的“Documentation”按钮。
  3. 点按导航区域顶部的“Browse”，以浏览已经安装的开发者资源库。
  4. 点按最近浏览的 iOS 资源库（可能必须先登录 Apple Developer）。

     iOS Developer Library 打开在内容区域。
  5. 在文稿区域的文稿列表上的“Documents”过滤栏中，键入“UIKit Framework Reference”。

     现在，过滤后的列表只显示输入的文稿名称。
  6. 点按文稿名称，打开列出所有 UIKit 类和协议的页面。

  阅读介绍并点按列出的类或协议，以了解关于它的更多信息。

- **应该十分明确您要应用程序做什么。**这个忠告适用于应用程序整体，也适用于应用程序的特定部分。一些框架架构会施加它们自己的子类化要求。例如，如果应用程序是基于文稿的，则必须创建一个抽象的

文稿类子类。

- **定义子类实例扮演的角色。**在开发 iOS 或 OS X 应用程序时，“模型－视图－控制器”设计模式用于为对象分配角色。视图对象出现在用户界面上；模型对象保存应用程序数据（并实施作用于该数据的算法）；控制器对象充当视图和模型对象之间的媒介。了解对象所扮演的角色，可以缩窄使用哪个超类的决定。例如，如果想要在 iOS 应用程序中自定绘制，可能必须创建 `UIView`（为 UIKit 框架中的基础视图类）的子类。

尽管子类化在为 iOS 和 OS X 应用程序编程中具有重要作用，但有时候不是解决问题的最佳方式。如果只想要为一个类添加一些便捷方法，您可能要创建类别，而不是子类。或者，要加入应用程序特定的行为，可以基于设计模式，使用框架中众多其他资源中的一种，例如委派。（“设计模式”文章的“采用设计模式使您的应用程序合理化”中描述了这些模式。）请记住，一些框架类不可以创建子类。参考文稿告诉您某一个框架类是否该用于创建子类。

覆盖方法

您可以创建不重新实施任何超类方法的子类，例如该子类可能添加额外状态，并定义新的方法来访问状态及调用该超类的方法。但是，对于一些子类而言，主要任务便是实施一组特定的、由超类（或超类所采用的协议中）所声明的方法。重新实施继承的方法，称为**覆盖**该方法。

在框架类中定义的大多数方法，都属于完整实施；它们的存在是让您可以调用它们，以获取该类所提供的服务。您很少需要覆盖这些方法，也不应尝试这么做。您可以覆盖其他框架方法，但是很少有必要这么做。

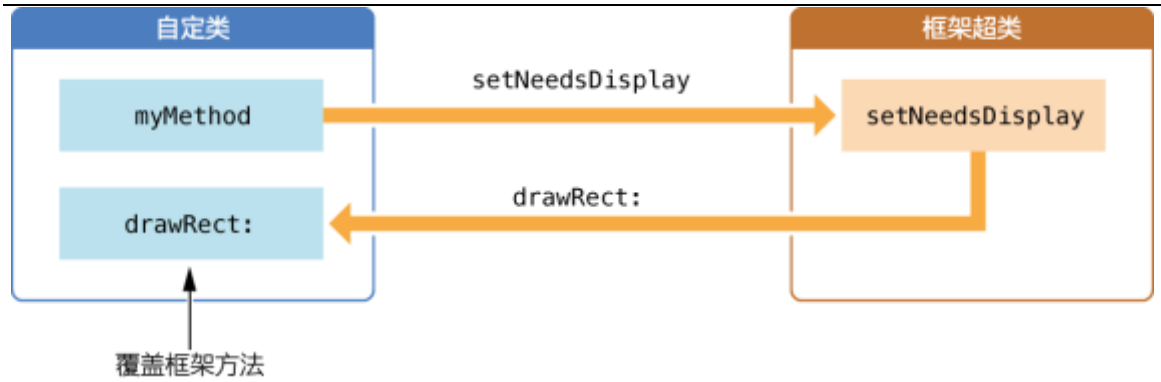
然而，一些框架方法是用于被覆盖的；它们的存在是让您将程序特定的行为添加到框架。一般来说，由这些框架实施的方法，所做的工作对您的应用程序来说没有多大价值，甚至没有价值。要将内容赋予这些方法，应用程序必须根据自身要求实施这类方法。应用程序运行期间，框架在适当的时候调用这些方法。

调用还是覆盖？

在子类中覆盖的框架方法，一般不是要亲自调用的方法，至少不是直接的调用。只需要重新实施该方法，剩下的事情就交给框架处理。事实上，您越是编写应用程序特定版本的方法，在自己的代码中调用它的可能性就越小。一般来说，框架类会声明公共方法，以便作为开发者的您可以使用它们来执行以下两项操作中的一项：

- 调用它们，以使用该类提供的服务。
- 覆盖它们，以便将您的代码引入到框架定义的程序模型中。

有时候，一个方法同时归入这两个类别；在调用时提供有价值的服务，也可以加以策略性的覆盖。但是，在大多数情况下，如果方法可供调用，则它已经由框架完全定义，您就不需要再在代码中重新定义它。如果是需要在子类中重新实施的方法，框架会用它来执行特定的任务，因此，它会在适当的时候，自行调用该方法。下图描述两大类型的框架方法。



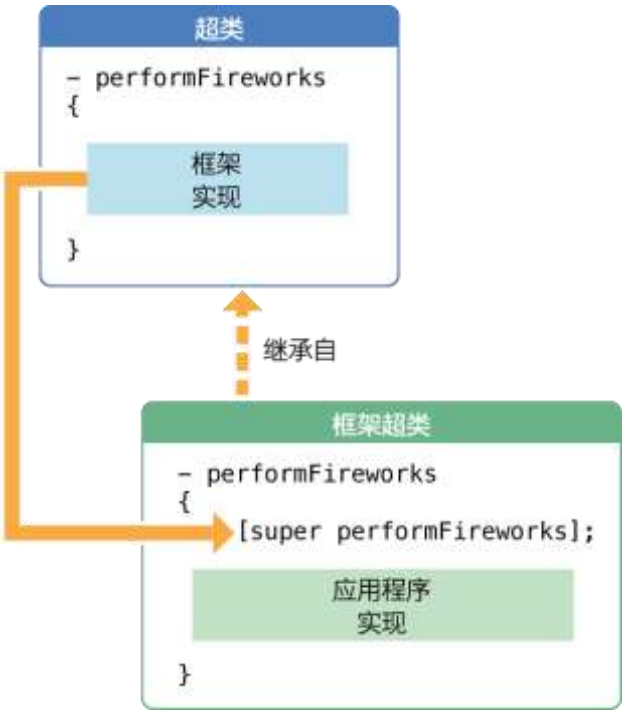
上图中，自定类 (myMethod) 的假设方法**调用**由框架实施的 setNeedsDisplay: 方法。框架做一些工作为绘制建立环境，然后调用框架声明的方法 drawRect:，该方法则由自定类**覆盖**，以执行实际的绘制。

覆盖一个方法不必是一项艰巨的任务。通过编写一两行代码，小心地重新实施一个方法，常常就可以对超类行为做出重大改变。

调用超类实现

覆盖框架方法时，必须决定是否要替换继承的方法的行为，或者扩展或补充该行为。如果想要替换现有的行为，提供自己的方法实现即可；如果想要扩展该行为，调用超类实现**并**提供自己的代码即可。

通过发送消息（与调用方法的消息相同）到 super 来调用超类实现。通过将消息发送给 super，您就将该方法的超类代码插入到重新实现的调用点。举个例子，比如一个假设的 Celebrate 类，定义了一个称为 performFireworks 的方法。在框架于视图中绘画并播放烟火动画后，您想在视图显示一个横幅。下图说明在这种情况下调用 super 的方式：



因此，决定是否调用 super，基于您打算如何重新实施方法：

- 如果打算**补充**超类实现的行为，请调用 `super`。
- 如果打算**替换**超类实现的行为，就不要调用 `super`。

如果您要补充超类行为，另一个需要重点考虑的，是何时调用一个方法的超类实现。您可能想超类代码在执行您的代码前，就做好它的工作，反之亦然。

采用设计模式使您的应用程序合理化

在 Objective-C 编程中，继承是添加应用程序特定行为的一种方式。创建的现有类的子类，要么增加超类的属性和行为，要么在某种程度上修改它们。但是，它也有其他更加动态的方式，可添加应用程序特定行为，而不涉及到子类化。这些动态技巧和方式，是基于设计模式的。正如本文章所解释的，在代码中采用设计模式，有助于增加类和框架类的可再用性和扩展性。

设计模式：解决编程问题的设计模板

设计模式是一个抽象工具，用于面向对象的软件开发，以及其他领域。它是一个设计模板，在特定背景中，解决一般性的、重复出现的问题。因此，设计模式是一种针对特定的、具体的设计的准则：在某种意义上，它是模式的“实例化”。在如何应用设计模式上，有一定的灵活性，通常例如程序设计语言和现有架构等事物，会影响如何应用模式。

有几个设计主题或原则对设计模式产生影响。这些设计原则是构建面向对象系统的经验法则，例如“封装发生变化的系统结构方面”(encapsulate the aspects of system structure that vary) 和“面向接口编程，而非面向实现编程”(program to an interface, not an implementation)。它们表达了重要的见解。例如，封装原则告诉我们，如果隔离并封装系统中发生变化的部分，它们可以独立于系统其他部分进行变化，特别是如果您为它们定义了不依赖实现特性的接口。您稍后可以修改或扩展这些可变部分，而不影响系统的其他部分。这样一来，您清除了各部分之间的互相依赖，减少了各部分的耦合性，系统就会变得更加灵活、更容易修改。

这样的优点，让设计模式成了编写软件的重点考虑因素。如果您在应用程序的设计中找到了模式，加以调整和使用，该程序（以及其所包含的对象和类）在将来需要时复用程度更高、扩展能力更强和更容易修改。此外，基于设计模式的应用程序，与没有基于设计模式的应用程序比较，会更见优雅和更具效率，因为它们只需较少的代码就能达到同样目的。

您会发现，设计模式的应用贯穿于整个 Cocoa Touch 和 Cocoa 框架、Objective-C 的运行时及程序设计语言自身。您可以几乎“免费”获得部分基于模式的机制，而其他部分则需要您做一些工作。情况合适时，可以将设计模式应用到您自己的应用程序代码中。如果使用与 Cocoa Touch 和 Cocoa 框架相同的模式，您的代码往往会更好地与框架的代码匹配，运行也更为优雅。

最重要的设计模式：模型－视图－控制器

“模型－视图－控制器”(Model-View-Controller) 设计模式，通常被称为“MVC”，将以下一种角色分配给应用程序中的对象：“模型”、“视图”或“控制器”。模式不仅定义了对象在应用程序中扮演的角色，还定义了对象之间通信的方式。这三类对象的每一个，都由抽象边界与其他对象分隔，穿过这些边界与其他类型的对象进行通信。应用程序中某一 MVC 类型的对象的集合，有时统称为层，例如模型层。

对于任何 iOS 应用程序或 Mac 应用程序而言，MVC 对一个好的设计至关重要。采用此设计的好处多不胜数。这些应用程序中的很多对象，倾向于更可再用，它们的接口倾向于定义得更好。采用 MVC 设计的应用程序，也比其他应用程序更容易扩展。此外，您的应用程序可以用到的很多技术和架构，都是基于 MVC 的，也要求您的自定对象扮演其中一个 MVC 角色。

您可能还没有意识到，就已经创建了一个基于 MVC 的应用程序：**您的首个 iOS 应用程序**中的 HelloWorld。模型对象是 `userName` 属性（`NSString` 对象），由 `HelloWorldViewController` 类声明和管理。`HelloWorldViewController` 类和 `HelloWorldAppDelegate` 类的实例，是应用程序的控制器对象；而应用程序的视图对象，是文本栏、标签、按钮和背景视图。

有关“模型－视图－控制器”的完整信息，请参阅 *Concepts in Objective-C Programming*（Objective-C 编程中的概念）中的“Model-View-Controller”。

模型对象

模型对象封装了应用程序的数据，并定义操控和处理该数据的逻辑和运算。例如，模型对象可能是表示游戏中的角色或地址簿中的联系人。有时应用程序的模型层，实际是相关对象的一个或多个图形。数据载入应用程序后，作为应用程序的持续状态（不论该持续状态是储存在文件中，还是在数据库中）一部分的大部分数据，应该驻留在模型对象中。因为模型对象代表了与特定问题领域相关的知识和专长，在相似问题领域，就可以重复使用它们。“纯”模型对象应该和视图对象不发生明确连接（视图对象显示其数据并允许用户编辑数据），它不该管到用户界面和显示的问题。

用户在视图层中所进行的创建或修改数据的操作，通过控制器对象传达出去，最终会创建或更新模型对象。模型对象更改时（例如通过网络连接接收到新数据），它通知控制器对象，控制器对象更新相应的视图对象。

视图对象

视图对象是应用程序中用户可以看见的对象。视图对象知道如何将自己绘制出来，并可能对用户的操作作出响应。视图对象的主要目的，就是显示来自应用程序模型对象的数据，并使该数据可被编辑。尽管如此，在 MVC 应用程序中，视图对象通常与模型对象分离。

因为您通常重新使用并重新配置对象，视图对象保证了应用程序之间的一致性。对于 iOS，UIKit 框架提供了视图类的集合；对于 OS X，AppKit 框架提供了类似的集合。在 UIKit 中，视图对象最终继承自 `UIView` 类；在 AppKit 中，视图对象最终继承自 `NSView` 类。

视图对象通过应用程序的控制器对象，了解模型数据的更改，并通过控制器对象，将用户发动的修改（例如，在文本栏输入的文本），传达到应用程序的模型对象。

控制器对象

在应用程序的一个或多个视图对象和一个或多个模型对象之间，控制器对象充当媒介。控制器对象因此是同步管道程序，通过它，视图对象了解模型对象的更改，反之亦然。控制器对象还可以为应用程序执行设置和协调任务，并管理其他对象的生命周期。

控制器对象解释在视图对象中进行的用户操作，并将新的或更改过的数据传达给模型对象。模型对象更改时，一个控制器对象会将新的模型数据传达给视图对象，以便视图对象可以显示它。

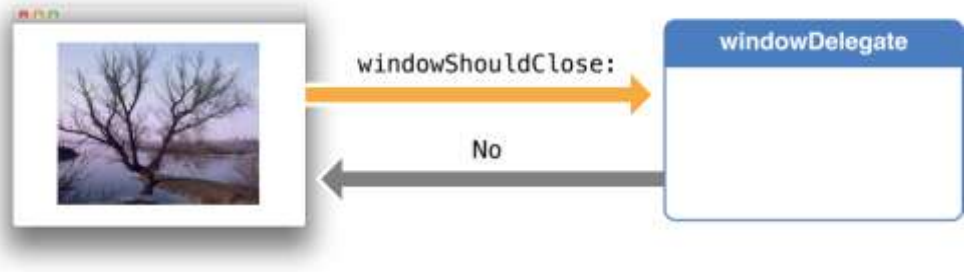
使用设计模式解决问题

面向对象的系统（例如应用程序）是动态的。对象在运行时所能做的，并不局限于编写时所设定的行为。一个对象可以向另一个对象发送消息，而同一消息的目标，会根据运行时的情况而变化。一个对象也可以在运行时与可变的一组其他对象合作，并使用多种技巧，有效地完成应用程序的工作。一个对象或一群对象要这样做，必须利用许多技巧和框架架构，它们都是设计模式的派生。

下面部分说明许多这样的技巧和架构。请将它们作为您 Objective-C 编程工具箱的一部分。

委托：代表另一个对象

在委托中，一个称为**委托**的对象应另一个对象的请求，作为该对象的代表。作出委托的对象，通常是框架模型。在执行的某些时候，它会向其委托发送消息，告诉委托即将发生某些事件，并要求给它回应。委托（通常是自定类的实例）实施供该消息调用的方法，并返回相应的值。通常该值是一个 Boolean 值，告诉作出委托的对象是否继续操作。



委托因此是一种将应用程序特定行为加入框架类工作的手段，而无需给该类创建子类。它是一种常见的、强大的设计，来扩展和影响框架的行为。

您应该记得，在编写**您的首个 iOS 应用程序** HelloWorld 时，创建了 `HelloWorldAppDelegate` 对象。Xcode 自动将其分配为应用程序对象（为框架对象）的委托。应用程序委托可以处理 `application:didFinishLaunchingWithOptions:`，以及应用程序对象发送给它的其他委托消息。

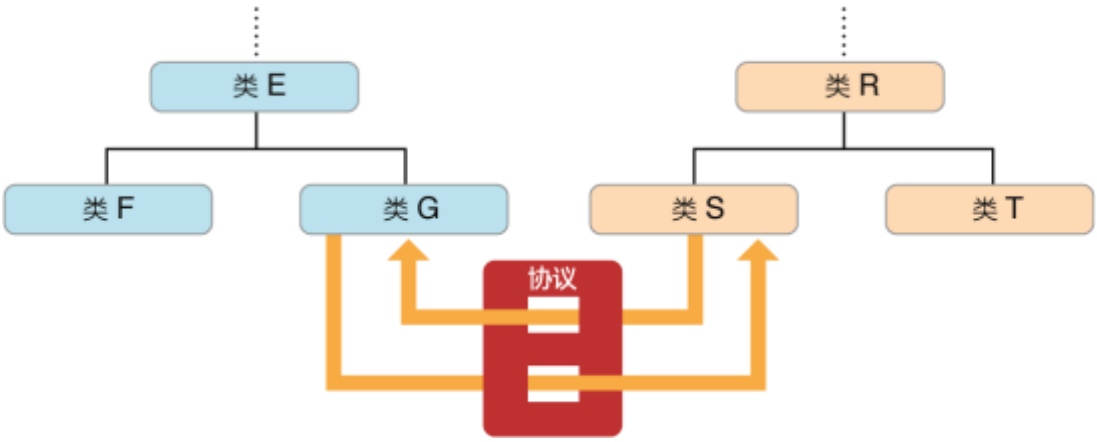
有两个可编程的组件用于委托。委托类必须定义属性（通过名称为 `delegate` 的约定），以保存一个指向委托的参考。它还必须声明委托类必须采用的协议（请参阅以下部分以获得有关协议的更多信息）。Cocoa Touch 和 Cocoa 框架的许多类，都提供委托作为一种方式，给应用程序用来增加其特定的框架行为。

但是委托并不局限于框架类。您可以在应用程序的两个自定对象之间实施委托。Cocoa Touch 应用程序常见的设

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。

协议：使不相关的对象之间能通过继承进行通信

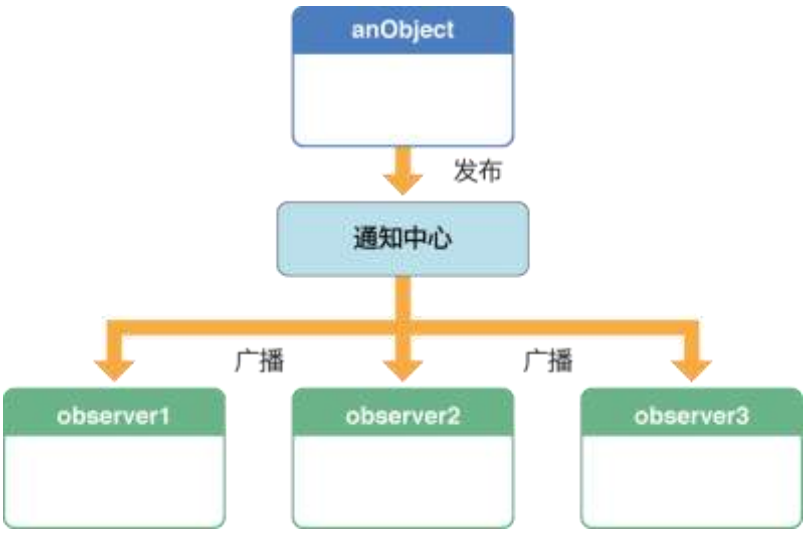
协议是可编程接口的声明，任何类都可以实施它的方法。与协议相关联的类实例，调用协议的方法，并获取由该类正式采用和实现该协议所返回的值。对象之间的此类通信，产生了一个特定目标，例如解析 XML 代码或拷贝对象。协议接口两边的对象可以通过继承，实施远距离彼此相关。协议因此和委托一样，可作为子类化的替换手段，通常是框架实施委托的一部分。



Apple 提供的框架，声明了数十个协议。此外，您的应用程序可以声明自定协议，让类可以采用。协议是您编程工具箱的一部分。*Programming with Objective-C*（使用 Objective-C 编程）对协议进行了综合描述。

通知中心：通知对事件感兴趣的观察者

通知中心是 Foundation 框架的一个子系统，它向应用程序中注册为某个事件观察者的所有对象广播消息（即通知）。（从编程角度而言，它是 `NSNotificationCenter` 类的实例）。该事件可以是发生在应用程序中的任何事情，例如进入后台状态，或者用户开始在文本栏中键入。通知是告诉观察者，事件已经发生或即将发生，因此让观察者有机会以合适的方式响应。通过通知中心来传播通知，是增加应用程序对象间合作和内聚力的一种途径。



例如，iOS 应用程序中的视图控制器，可以观察 `UIKeyboardWillShowNotification` 通知，以调整其视图的

几何图形，来容纳虚拟键盘。正如此例所示，通知是一个对象，该对象的名称指明了一个特定事件，以及该事件是已经发生或将要发生。它还将一个引用（指向发布或发送通知的对象）送到通知中心，而它可以包含补充信息字典。

任何对象都可以观察通知，但要做到这一点，该对象必须注册，以接收通知。在注册时，它必须指定选择器，以确定由通知传送所调用的方法；方法签名必须只有一个参数：通知对象。注册后，观察者也可以指定发布对象。

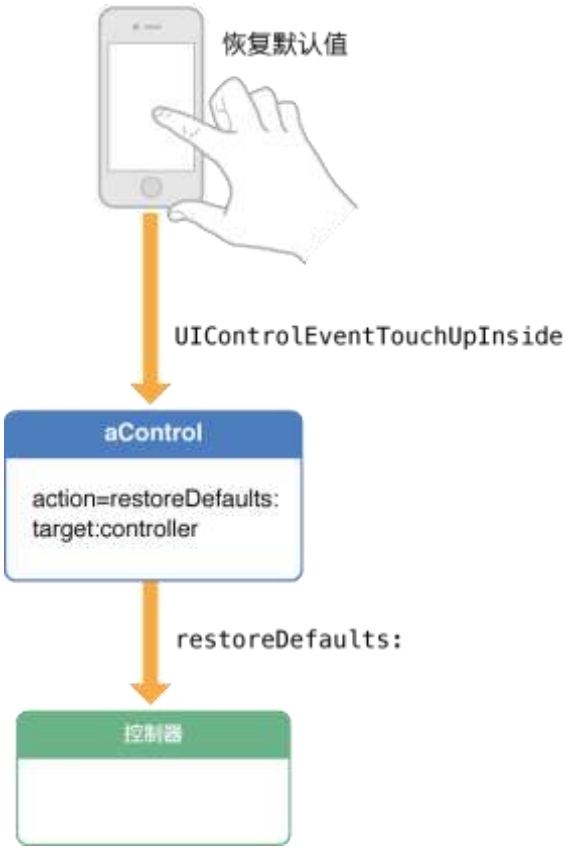
通知中心的通知跟委托消息相似；当某些事件发生时，两者都发送给任意对象。但是，处理通知的方法与委托方法不同，它不能返回值。通过通知中心的通知是同步的，与委托一样。

应用程序的自定对象可定义和发布自己的通知，其他自定对象则可以观察该通知。

目标-操作：事件发生时封装待发送的消息

目标-操作设计在概念上很简单。一个对象储存着组成消息表达式的元素，某些事件发生时，将这些元素放在一起，并发送一则消息。这些元素为一个选择器，用来确定消息（即操作）和接收消息的对象（即目标）。目标的类会实现与操作和目标相对应的方法，当它在运行中接收到消息时，会通过执行方法来响应事件。

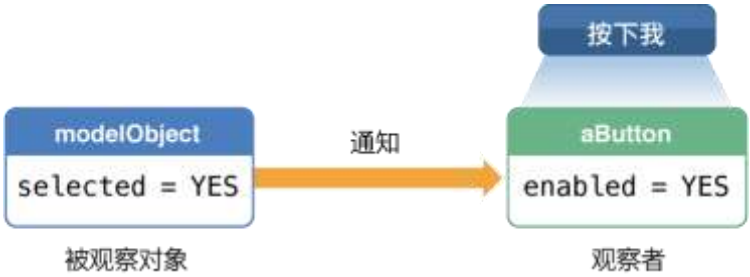
目标-操作主要是 Cocoa Touch 和 Cocoa 框架中的一种控制功能。控制是用户界面对象，例如用户通过轻按、拖移等进行操控的按钮、滑块或开关，将用户的意图通过信号发送给应用程序。Cocoa Touch 的控制储存了操作和目标；大多数 Cocoa 控制与一个或多个单元对象进行了配对，这些单元对象储存了目标和操作。



一些框架在对象中使用目标-操作而不是控制。例如，在设计手势识别器时，UIKit 框架使用了目标-操作。手势识别对象识别手势后，它将操作消息发送给目标对象。

键值观察：值更改时通知观察者

键值观察（Key-value observing，或简称 KVO）允许对象观察另一个对象的属性。该属性值改变时，会通知观察对象。它了解新值以及旧值；如果观察的属性为对多的关系（例如数组），它也要了解哪个包含的对象发生了改变。KVO 有助于使应用程序变得更内聚，保持模型、控制器和视图层中的对象与改变同步。



与 `NSNotificationCenter` 通知相似，多个 KVO 观察者可以观察单一属性。此外，KVO 更动态，因为它允许对象观察任意属性，而不需任何新的 API，例如通知名称。KVO 是一个轻量级点对点通信机制，不允许观察所有实例的特定属性。

基于设计模式的其他框架设计

Cocoa Touch 和 Cocoa 框架也包含基于设计模式的其他设计，有以下模式：

- **视图层次。**应用程序所显示的视图，会排列成层次结构（直观上基于包含）。此模式允许应用程序将单个视图和合成视图同等对待。层次的根部为一个窗口对象；根部以下的每个视图，都有一个父视图，以及零个或多个子视图。父视图包含子视图。视图层次是绘图和事件处理的结构性组件。
- **响应器链。**响应器链是一系列的对象（主要是视图，但也有窗口、视图控制器和应用程序对象本身），事件或操作消息可以沿着响应器链传递，直到链中的一个对象处理该事件。因此，它是一个合作性事件处理机制。响应器链与视图层次密切相关。
- **视图控制器。**虽然 UIKit 和 AppKit 框架都有视图控制器类，它们在 iOS 中尤其重要。视图控制器是一种特殊的控制器对象，用于显示和管理一组视图。视图控制器对象提供基础结构，来管理内容相关的视图并协调视图的显示与隐藏。视图控制器管理应用程序视图的子层次结构。
- **前台。**在前台模式中，应用程序所执行的工作，从一个执行环境重定向（或弹回）到另一个环境。（执行环境是一个与主线程或辅助线程相关联的调度队列或操作队列。）您将前台模式主要应用于这样的情形：在次队列执行的工作，产生了必须在主队列执行的任务，例如更新用户界面的操作。
- **类别。**类别提供了一种方式，通过将方法添加到一个类，以使该类得到扩展。与委托一样，它可以让您自定行为，而不子类化。类别是 Objective-C 的一个功能，在**编写 Objective-C 代码**中有说明。

从用户角度进行设计

iOS 应用程序的成功，很大程度上取决于其用户界面的质量。如果用户发现应用程序不具有吸引力，又不容易使用，那么即使它是最快、最强大、功能最完整的应用程序，也会在 App Store 中沉没。

有许多方法可将一时的灵感转化为流行的应用程序，但没有必胜的独门偏方。不过，所有成功的应用程序开发，都

遵循同一个指导原则：从用户角度进行设计。以下总结的策略和最佳实践，全都基于此指导原则，它们指出了在设计应用程序时，您需要遵循的一些原则和指南。当您准备好着手开发时，请务必阅读 *iOS Human Interface Guidelines*（用户界面指南）以获得完整的信息。

理解用户如何使用他们的设备

如果是刚接触 iOS，您需要做的第一步，就是自己成为 iOS 用户。然后，以用户身份（而不是开发者身份）尽可能探索 iOS 平台的特征。无论一开始您就用过基于 iOS 的设备，还是从未接触过，请在使用设备时，花时间弄清楚您的期望并分析您的操作。

例如，考虑以下的设备和软件功能，如何影响用户的体验：

- iPhone、iPad 和 iPod touch 为手持设备，能够促使用户随时随地使用。用户期望应用程序能够迅速启动，并且在各种环境下都容易使用。
- 在所有基于 iOS 的设备上，不管设备尺寸大小，显示屏都最为重要。当用户专注于使用应用程序时，相对较小的显示屏不会构成障碍。
- 多触摸界面让用户无需借助别的设备（例如鼠标），就可操控内容。用户通常会有更能主导应用程序的体验，因为他们能够以触控来操控屏幕上的元素。
- 一次只有一个应用程序在最前面。用户可以使用多任务栏在应用程序之间迅速而轻松地切换，但是这种体验和在电脑显示器上看到多个应用程序同时打开是不同的。

- 一般情况下，应用程序不会同时打开几个单独窗口。相反，用户在屏幕内容之间转换，每个屏幕内容可以包含多个视图。
- 内建的“设置”应用程序包含用户偏好设置，用来设置设备以及在设备上运行的一些应用程序。要打开“设置”，用户必须从当前使用的应用程序切换出去，因此，这些偏好设置应该是那些“设好就不常改”的类型。大多数应用程序可以避免把偏好设置添加到“设置”中，反而在应用程序的主用户界面，给用户进行选择。

请记住，完美的应用程序需发挥其运行平台的优点，其使用体验亦要完全融合设备和平台的特性。

学习基本的用户界面原则

作为一名用户，当应用程序令您不清楚它是否接收到您的输入，或者弹出式窗口不明所以地在屏幕上到处冒出，您都会特别在意。在这些情况下，您在意的其实就是应用程序没有遵循用户界面的基本设计原则。

言下之意，用户界面指的是用户和设备（包括在设备上运行的软件）之间的互动。当应用程序（或设备）的用户界面按照用户实际思考和操作的方式构建时，用起来就会令人觉得很容易、很顺心。

Apple 的用户界面设计原则，归纳了人机互动的几个高层次范畴，对用户体验有着深刻影响。设计应用程序时，请记住以下用户界面设计的原则：

- **美学集成度**。美学集成度 (aesthetic integrity) 不是衡量应用程序有多好看；而是看应用程序的外观和它的功能融合得有多紧密。
- **一致性**。界面的一致性让用户把对一个应用程序的知识和技巧，应用到另一个应用程序。理想情况下，应用程序应与 iOS 标准、其本身及较早版本相一致。
- **直接操控**。当用户直接操控屏幕上的对象（而不是透过别的控制），他们会更加专注于任务，并且更容易理解操作的结果。
- **反馈**。通过反馈来确认用户的操作，并让他们确信处理正在进行。例如，当用户操作控制时，他们期望得到即时的反馈；进行长时间的操作时，则希望看到状态的更新。
- **隐喻**。如果应用程序中的虚拟对象和操作，隐喻着现实世界中的对象和操作，用户会迅速理解如何使用应用程序。最适当的隐喻，会启发一种用法和体验，而不是把所基于的现实世界对象或行动的限制强加实施。
- **用户控制**。虽然应用程序可以建议一系列操作，或者对危险的后果提出警告，但是如果不给用户做决定，则通常是错误的。最好的应用程序，会在给予用户所需的功能和帮助他们避免危险的结果之间，找到正确的平衡。

遵循指南

iOS Human Interface Guidelines（iOS 用户界面指南）给出了大量指导准则，范围从用户体验的建议，到使用 iOS 技术与屏幕元素的具体规则。这一部分并不是 *iOS Human Interface Guidelines*（iOS 用户界面指南）的摘要；而是让你接触一些指南，有助于您设计一个成功的应用程序。

好的 iOS 应用程序，会让用户流畅地访问他们所关心的内容。为了达到这样的目的，这些应用程序整合了这些用户体验指南：

- 聚焦于主要任务。
- 让用法简单和明确。
- 使用以用户为中心的术语。
- 制作指尖大小的目标。
- 不强调设置。
- 一致地使用用户界面 (UI) 元素。
- 使用令人会意的动画来沟通。
- 仅在必要时要求用户进行存储。

用户期望应用程序整合平台的功能，例如多任务、iCloud、VoiceOver 和打印。用户可能认为这些功能是自然而然就该有的，应用程序开发者则清楚他们必须花工夫来整合这些功能。要确保应用程序为这些功能提供预期的用户体验，开发者须遵循这些 iOS 技术指南：

- 简单清晰地支持 iCloud 储存。
- 做好与多任务相关的中断和恢复准备。
- 处理本地通知和推送通知时，遵循用户的“通知中心”设置。
- 提供叙述信息，以便 VoiceOver 用户操作应用程序。
- 依赖系统提供的打印 UI，为用户带来满意的打印体验。
- 确保声音在各种情形下，都满足用户的期望。

当应用程序正确地使用 UI 元素（例如按钮和标签栏）时，用户就会专注于应用程序是否按他们所预期的方式运行。但是，当应用程序错误地使用 UI 元素时，用户很快就会表现不满。好的 iOS 应用程序，会小心遵循 UI 元素使用指南。例如：

- 确保导航栏中的返回按钮，显示上一个屏幕的标题。
- 当标签功能不可用时，不要将标签从标签栏移除。
- 避免提供“隐藏弹出式窗口”按钮。
- 当用户选择表格视图所列出的项目时，要提供反馈。
- 在 iPad 上，仅在弹出式窗口内显示挑选器。
- 使用系统提供的按钮和图标时，遵从它们定义好的含义。
- 设计自定图标和图像时，使用所有用户都理解的通用图像，避免复制 Apple 的 UI 元素或产品。

再次说明，本节中列出的指南，只是 *iOS Human Interface Guidelines*（iOS 用户界面指南）中的一部分。在应用程序开发过程中，完整阅读该文稿是非常重要的一步。

利用一些经过验证的设计策略

最成功的 iOS 应用程序，通常是深思熟虑、反复设计的结果。当开发者聚焦于主要任务，使功能更加精炼，是可以创建优秀的用户体验。本节总结的策略，可以帮助改进您的想法、审视设计选项，并专注于用户会欣赏的应用程序上。

提炼功能列表。在设计过程中，尽早确定应用程序的功能和目标用户。使用此定义（称为**应用程序定义语句**）过滤掉不必要的功能，并指导应用程序的风格。虽然，功能越多应用程序就越好的想法很诱人，很多时候，却是反面教材。最好的应用程序，通常聚焦于一个主要任务，只提供用户完成该任务所需的那些功能。

为设备而设计。除了整合 iOS 用户界面和用户体验的模式之外，请确定您的应用程序在设备上运行自如。如果计划开发一个通用应用程序（即同时运行在 iPhone 和 iPad 上的应用程序），这就意味着必须为每个设备设计不同的 UI，即使大多数底层代码可以是相同的。同样，如果计划采用基于网上的内容，有必要重新设计这些内容，使其看起来和感觉起来像是原生的应用程序。

适当地定制。每个应用程序都包括一些自定 UI（即使只在其 App Store 图标中）。iOS SDK 可以让您自定 UI 的各个方面，至于多少自定才合适就完全由您决定。最好的应用程序，会以目的明确和易用作为自定的考量。理想情况下，您想要用户觉得您的应用程序与众不同，又同时欣赏到其直观和易用，与其他应用程序保持一致。

原型和迭代。在决定好包括哪些功能后，您就可以开始创建可测试的原型。早期的原型不需要显示真实的 UI 或美工图样，也不需要处理真实的内容。但是，它们需要给测试员准确的概念，知道应用程序是如何使用的。在测试过程中，要特别注意测试员尝试过但失败的地方，因为这些尝试，可以暴露出应用程序本该有却未实现的行为。继续测试直到您感到满意，认为用户可轻松理解应用程序是如何使用的，并能操作全部功能。

用心设计您的应用程序

如果您是 iOS 应用程序开发的新手，可能想知道从哪里开始应用程序的开发过程。有了应用程序的初步构思后，您需要将这种想法转换为实现应用程序的行动计划。从设计角度而言，您需要作出把想法实现的最佳步骤的一些高层次的决定。接着您可以开始开发应用程序。

iOS App Programming Guide（iOS 应用程序编程指南）详细解释了本文中提及的许多概念、架构和技巧。

做最初设计

设计应用程序的方法有很多，而且许多最佳方式都不涉及编写代码。好的应用程序源自好的想法，然后将这些想法扩展为更加完整的产品描述。在设计阶段早期，你需要搞清楚您要应用程序做到些什么。记下实现您的想法所需的那些高级功能。根据您认为用户的需要，确定那些功能的优先级。对 iOS 本身做一点调研，以便了解其功能，以及您可如何使用它们来实现目标。在纸上草拟一些粗略的界面设计，以直观显示您的应用程序可能的样子。

初始设计的目标，是回答有关应用程序的一些非常重要的问题。功能集合和界面的粗略设计，有助于思考在开始编写代码后，需要哪些东西。在某个阶段，您需要将应用程序显示的信息转换为一组数据对象。同样，应用程序的外

观，对您在实施用户界面代码时必须做出的选择，具有压倒性的影响。在纸上（而不是在电脑上）做最初的设计，尽可天马行空，想出的答案不必限于那些容易做到的东西。

当然，在开始设计前可以做的最重要事情，是阅读 *iOS Human Interface Guidelines*（iOS 用户界面指南）。这本书介绍了进行最初设计的一些策略。它还为如何创建在 iOS 中运行良好的应用程序，给出了提示和指导。*iOS Technology Overview*（iOS 技术概述）描述 iOS 的功能，以及如何使用这些功能实现您的设计目标。

将您的最初设计转换为行动计划

iOS 假定所有应用程序都是使用“模型-视图-控制器”设计模式构建的。因此，您迈向实现此目标的前几步，是选取适合应用程序的数据和视图部分的方法。

- 选取适合数据模型的基本方法：
  - **现有数据模型代码**——如果您已经采用基于 C 程序设计语言编写的数据模型代码，可以将该代码直接集成到 iOS 应用程序。由于 iOS 应用程序是采用 Objective-C 编写的，它们正好配合用其他基于 C 程序设计语言编写的代码。当然，还有一个好处，是能够针对任何非 Objective-C 的代码编写 Objective-C 包装器。
  - **自定对象数据模型**——自定对象通常将某些简单数据（字符串、数字、日期、URL 等）与业务逻辑相结合，业务逻辑是管理此类数据并确保其一致性所需要的。自定对象可将标量值和指针的组合储存到其他对象中。例如，Foundation 框架定义的类，用于许多简单数据类型，并用于储存一组其他对象。这些类使得定义您自己的自定对象更轻松。
  - **结构化数据模型**——如果您的数据是高度结构化的（也就是说，该数据适合储存在数据库中），请使用 Core Data（或 SQLite）储存数据。Core Data 提供简单的、面向对象的模型来管理结构化数据。它还提供对部分高级功能（如撤销和 iCloud）的内建支持。（SQLite 文件不能与 iCloud 结合使用。）
- 决定是否需要支持文稿：

  文稿的工作是管理应用程序的内存数据模型对象，并协调将此类数据储存在磁盘上的对应文件（或一组文件）中。文稿通常意味着用户创建的文件，但应用程序也可以使用文稿来管理那些不面向用户的文件。使用文稿的一大好处，是 `UIDocument` 类让其与 iCloud 和本地文件系统的交互变得更简单。对于使用 Core Data 储存内容的应用程序，`UIManagedDocument` 类提供类似支持。
- 选取用于用户界面的方法：
  - **构造块方法**——创建用户界面的最简单方法，是使用现有的视图对象来组装界面。视图表示视觉元素，如表格、按钮、文本栏等。您按原样使用许多视图，但也可以根据需要，自定标准视图的外观和行为，以满足您的需求。您还可以使用自定视图，实现新的视觉元素，并将此类视图与界面中的标准视图自由混合。视图的优势是它们提供一致的用户体验，以及可让您使用相对较少的代码，快速定义复杂的界面。
  - **基于 OpenGL ES 的方法**——如果应用程序需要频繁更新屏幕或复杂的渲染，您可能需要直接使用 OpenGL ES 绘制内容。OpenGL ES 主要用于大程度利用复杂的图形，并因此需要尽可能最

佳的性能的游戏和应用程序。

开始应用程序创建过程

制定好行动计划后，该是时候开始编程了。如果您是编写 iOS 应用程序的新手，最好花时间浏览提供用于开发的初始 Xcode 模板。这些模板大幅简化了您必须完成的工作，使得您可以在几分钟内就做好一个应用程序来运行。这些模板还可让您自定初始项目，以更精确地支持具体需求。为了实现这一目标，在创建 Xcode 项目时，您心中应该已经有了以下问题的答案：

- **应用程序的基本界面风格是什么？**不同类型的应用程序，需要不同的一组初始视图和视图控制器。了解您计划如何组织用户界面，可让您选择最能满足需求的初始项目模板。您还是可以在后面更改用户界面，但一开始就选取最适合的模板，可使启动项目更加简单。
- **您是要创建通用应用程序，还是专门针对 iPad 或 iPhone 的应用程序？**创建通用应用程序，需要为 iPad 和 iPhone 指定不同的一组视图和视图控制器，并在运行时动态选择合适的那一组。首选是通用的应用程序，因为它们支持更多的 iOS 设备，但需要您更好地分解代码以适合每个平台。
- **您要应用程序使用串联图吗？**串联图通过显示用户界面的视图和视图控制器及它们之间的转换，简化了设计流程。iOS 5 和更高版本支持串联图，新项目是默认启用的。如果应用程序必须在较早版本的 iOS 上运行，则不能使用串联图，而应该继续使用 nib 文件。
- **您要将 Core Data 用于数据模型吗？**某些类型的应用程序本身就适合结构化数据模型，这让使用 Core Data 成为它们的理想候选方式。

在安装 Xcode、配置 iOS 开发团队，并在 Xcode 中创建应用程序项目后，就可以开始开发应用程序了。以下应用程序开发阶段是通用的：

1. 开始编写应用程序的主要代码。

   对于新的应用程序，最好先开始创建与应用程序的数据模型关联的类。这些类通常不依赖应用程序的其他部分，而且应该是最初可处理的内容。然后是着手构建用户界面的设计，方法是将视图添加到主串联图或 nib 文件。从这些视图，您还可以着手确定代码中哪些部分需要反应界面相关的变动。如果应用程序支持 iCloud，应该在早期阶段就将 iCloud 支持集成到类。

2. 添加对应用程序状态更改的支持。

   在 iOS 中，应用程序的状态，决定了允许执行什么操作及何时执行操作。应用程序状态由应用程序中的高级对象管理，但也可以影响许多其他对象，因此，您需要考虑当前应用程序状态，如何影响数据模型和视图代码，并相应更新代码。

3. 创建支持应用程序所需的资源。

   要求提交到 App Store 的应用程序具有特定资源（如图标和启动画面），以提高整体的用户体验。分解良好的应用程序，会大量使用资源文件，来保持代码与代码所操控的数据相分离。此类分解使对应用程序进行本地化、对其外观进行调整和执行其他任务更加简单，并且无需重新编写任何代码。

4. 根据需要，实施任何与应用程序相关的应用程序特定行为。

有多种方法用于修改应用程序启动或与系统交互的方式。例如，您可以针对某个功能实施本地通知。

5. 添加高级功能，使应用程序变得独一无二。

   iOS 包括许多其他框架，用于管理多媒体、高级渲染、游戏内容、地图、通讯录、位置跟踪和许多其他高级功能。*iOS Technology Overview*（iOS 技术概述）概述可集成到应用程序的框架和功能。

6. 为应用程序调整一些基本的性能。

   所有 iOS 应用程序，都应该经过调整来实现可能的最佳性能。调整后的应用程序运行得更快，也更高效地使用系统资源，如内存和电池电量。

7. 迭代。

   应用程序开发是一个迭代过程。添加新功能时，您可能需要重新访问前期的部分或全部步骤，才能调整现有代码。

了解您的应用程序的核心对象

UIKit 框架为所有应用程序提供基础结构，但仍然需要您自定对象来赋予应用程序的具体行为。您的应用程序，由一些特定的 UIKit 对象组成，这些 UIKit 对象管理事件循环以及与 iOS 的主要交互。通过组合运用子类化、委托和其他技巧，您可以修改 UIKit 所定义的默认行为，来实现您的应用程序。

除了自定 UIKit 对象以外，您还需要负责提供或定义其他多组关键的对象。最大的一组对象，是应用程序的数据对象，它们的定义，完全由您负责。您还必须提供一套用户界面对象。幸运的是，UIKit 提供了许多类，让您轻松地定义界面。除编程以外，您还必须提供资源和数据文件，这些文件是交付可发行应用程序所需要的。

iOS App Programming Guide（iOS 应用程序编程指南）详细描述了此文章中提及的许多主题。

应用程序的核心对象

从用户启动您的应用程序直到退出，UIKit 框架管理着应用程序的许多核心行为。应用程序的心脏，是 `UIApplication` 对象，它从系统接收事件，然后将事件分派到您的自定代码进行处理。其他 UIKit 类也参与应用程序行为的部分管理，所有这些类都使用相似的方式，调用您的自定代码来处理细节。

要理解 UIKit 对象如何配合您的自定代码来工作，需要理解一下组成 iOS 应用程序的对象。图 1 显示 iOS 应用程序中最常见的对象，表 1 描述每个对象的角色。从框图中，您可以看到，iOS 应用程序是围绕“模型－视图－控制器”设计模式来组织的。此模式将模型中的数据对象，从用来显示该数据的视图中分离出来。这种分离使得随着需要来交换视图成为可能，从而提高了代码重用性，这在创建通用应用程序时特别有用（即可以同时在 iPad 和 iPhone 上运行的应用程序）。

图 1 iOS 应用程序中的关键对象

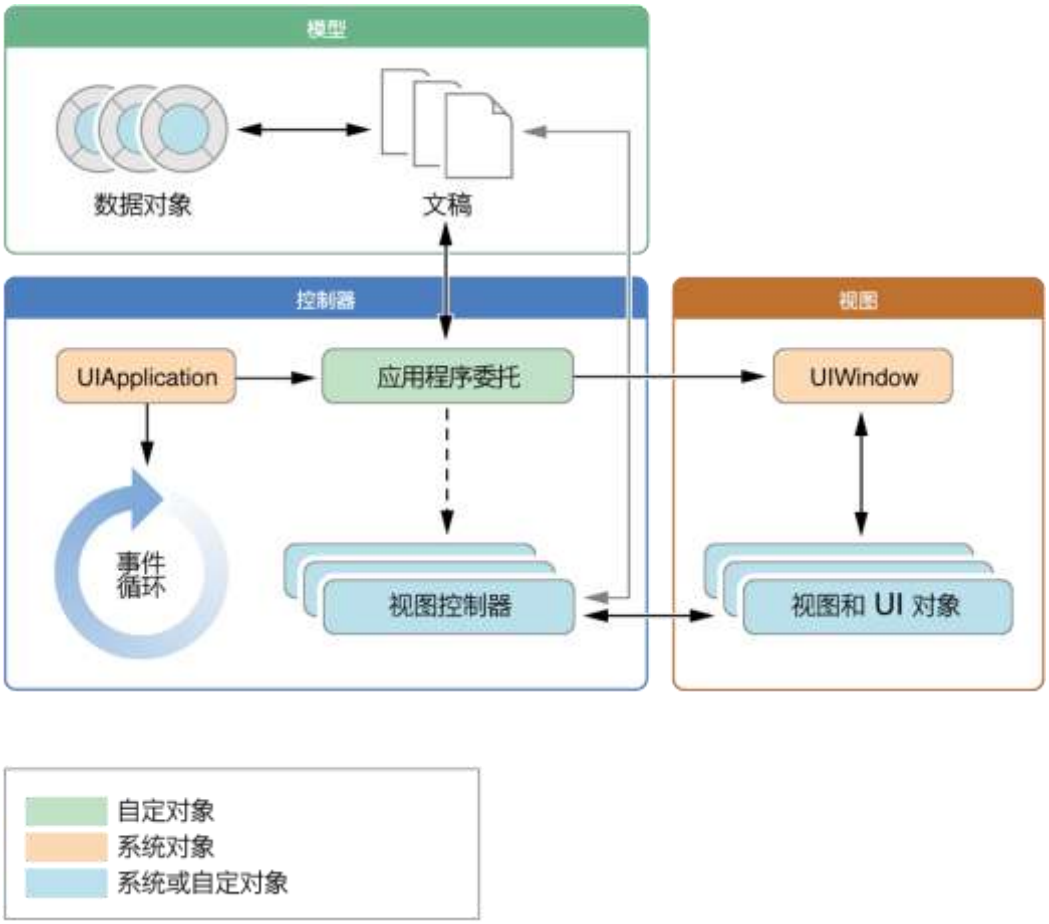


表 1 iOS 应用程序中的对象角色

对象	说明
`UIApplication` 对象	您基本上按原样使用 `UIApplication` 对象，就是说，不对它进行子类化。此控制器对象管理应用程序事件循环，并协调其他高级的应用程序行为。由您定制的应用程序级的逻辑，存在于应用程序的委托对象中，并与此对象紧密合作。
应用程序委托对象	**应用程序委托**是在应用程序启动时，创建的一个自定对象，通常由 `UIApplicationMain` 函数创建。此对象的主要工作，是处理应用程序内部的状态转换。例如，此对象负责启动时的初始化，并负责处理进入背景和从背景出来的转换。 在 iOS 5 和更高版本中，您可以使用应用程序委托，来处理其他与应用程序相关的事件。Xcode 项目模板将应用程序委托声明为 `UIResponder` 的子类。如果 `UIApplication` 对象不处理事件，它会将事件分派给应用程序的委托，以进行处理。

文稿和数据模型对象	**数据模型对象**储存应用程序的内容，是您的应用程序专有的。例如，银行业务应用程序，可能储存包含金融交易的数据库；而绘图应用程序，则可能储存图像对象，甚至储存创建该图像的绘图命令序列。（就绘图命令来说，图像对象仍是一个数据对象，因为它只是图像数据的容器。） 应用程序还可以使用**文稿对象**（UIDocument 的自定子类），来管理其数据模型对象中的一部分或全部。文稿对象不是必需的，但它们提供了一种便利的方式，对属于单个文件或文件包中的数据进行分组。有关文稿的更多信息，请参阅“定义基于文稿的数据模型”。
视图控制器对象	**视图控制器对象**管理应用程序内容在屏幕上的呈现。视图控制器管理单个视图及其分视图。呈现时，视图控制器将视图安装到应用程序的窗口中，使它们显示出来。 UIViewController 类是所有视图控制器对象的基础类。它提供了一些默认功能，用于载入视图、呈现视图和旋转视图，以响应设备的旋转以及几个其他标准的系统行为。UIKit 和其他框架定义附加的视图控制器类，来实现标准系统界面，如图像挑选器、标签栏界面和导航界面。
UIWindow 对象	UIWindow 对象协调一个或多个视图在屏幕上的呈现。大多数应用程序只有一个窗口，用于在主屏幕上呈现内容，但应用程序可能会有另外一个窗口，将内容显示在外接显示器上。 要更改您的应用程序的内容，需使用视图控制器，来更改在对应窗口中显示的视图。您不会把窗口本身替换。 除了充当视图的宿主以外，窗口还配合 UIApplication 对象工作，将事件传送到视图和视图控制器。
视图、控制和层对象	视图和控制将应用程序的内容直观地呈现出来。**视图**是一种对象，将内容绘制在指定的矩形区域内，并响应该区域的事件。**控制**是一类专门的视图，负责实施常见的界面对象，如按钮、文本栏和切换开关。 UIKit 框架提供标准的视图，用于呈现许多类型的内容。通过直接将 UIView（或它的子类）子类化，您还可以定义自己的自定视图。 除了包括视图和控制以外，应用程序还可以将 Core Animation 层并入其视图和控制分层结构中。**层对象**实际是代表视觉内容的数据对象。视图在幕后大量使用层对象，来渲染其内容。您还可以将自定的层对象，添加到界面，以实施复杂的动画和其他类型的复杂视觉效果。

iOS 应用程序之间的区别，在于所管理的数据（和相应的业务逻辑），以及如何将数据呈现给用户。大多数与 UIKit 对象的交互，并不是替您定义应用程序，而是让它的行为纳入规范。例如，您的应用程序委托的方法，让您知道应用程序的状态何时改变，以便您的自定代码可以适当地作出响应。

数据模型

应用程序的数据模型，由数据结构以及业务逻辑组成，业务逻辑是让数据保持一致状态所必要的。不要把数据模型的设计，脱离应用程序的用户界面来进行；但是，数据模型对象的实现，应该是分开的，而不依赖于特定视图或视图控制器存在与否。保持数据与用户界面分开，有助于实现通用应用程序（可在 iPad 和 iPhone 双平台上运行的

应用程序），也让重复使用部分代码变得容易。

如果您尚未定义数据模型，iOS 框架会就这方面为您提供帮助。以下部分重点描述了一些技术，让您使用定义特定类型的数据模型。

定义自定数据模型

定义自定数据模型时，请创建自定对象，来表示任何高级结构。但对简单的数据类型，大可利用系统提供的对象。Foundation 框架提供了许多对象（大部分对象已在表 2 中列出），以面向对象的方式管理字符串、数字以及其他类型的简单数据。使用这些对象，比定义新的对象要好，因为节省时间，而且许多系统例程要求使用内建的对象。

表 2 Foundation 框架中的数据类

数据	类	说明
字符串和文本	NSString (NSMutableString) NSAttributedString(NSMutableAttributedString)	iOS 中的字符串是基于 Unicode 的。字符串类提供了支持，可以用多种方式创建和操控字符串。带属性的字符串类，支持样式化的文本，须配合 Core Text 使用。
数字	NSNumber NSDecimalNumber NSIndexPath	当您要把数值储存在集 (collection) 中时，请使用数字对象。NSNumber 类可以表示整数值、浮点数值、Boolean 值和 char 值。NSIndexPath 类储存数字序列，通常用来指定层次列表中的多层选择。
原始字节	NSData (NSMutableData) NSValue	每当需要储存原始字节流时，请使用数据对象。数据对象也常用来以归档形式储存对象。NSValue 类通常已经扩展（使用类别），用来归档常见数据类型，如点和矩形。
日期和时间	NSDate NSDateComponents	使用日期对象来储存时间戳、日历日期，以及其他时间相关的信息。
URL	NSURL	除了指向网络资源这一传统用途以外，iOS 中的 URL 还是储存文件路径的首选方式。NSURL 类甚至支持获取和设定文件相关的属性。

集	NSArray (NSMutableArray) NSDictionary(NSMutableDictionary) NSIndexSet(NSMutableIndexSet) NSOrderedSet(NSMutableOrderedSet) NSSet (NSMutableSet)	使用集将相关的对象聚合在一个地方。Foundation 框架提供了几种不同类型的集类。

除了与数据相关的对象以外，还有一些其他的数据类型，通常由 iOS 框架用来管理常见类型的数据。鼓励您在自定的对象中使用这些数据类型，来表示相似类型的数据。

- NSInteger/NSUInteger——有符号的标量和无符号的整数的抽象，基于架构定义了整数的大小。
- NSRange——一种结构，用来定义一个序列的连续部分。例如，您可以使用范围定义字符串中被选定的字符。
- NSTimeInterval——给定时间间隔中的秒数（整数和小数）。
- CGPoint——x 和 y 坐标值，定义一个位置。
- CGSize——坐标值，定义一组水平和垂直的范围。
- CGRect——坐标值，定义一个矩形的区域。

当然，在定义自定对象时，您总是可以将标量值直接并入到类的实现中。实际上，自定数据对象的成员变量，可以混用标量和对象类型。列表 1 是一个类定义的示例，用于一组图片的集。此示例中的类，包含一个图像数组，以及在这个数组中代表所选项目的索引列表。该类还包含集标题名称的字符串，以及一个标量 Boolean 变量，指示当前的集是否可编辑。

列表 1 自定数据对象的定义

```
@interface PictureCollection :NSObject



@property (nonatomic, copy) NSString * title;

@property (nonatomic) BOOL editable;

@property (nonatomic, readonly) NSOrderedSet* pictures;

@property (nonatomic) NSMutableIndexSet *selection;



// Method definitions...
```

```
@end
```

要考虑自定对象上的撤销操作如何处理。支持撤销，意味着能够干净利落地复原对您的对象所做的更改。如果您的对象包含复杂的业务逻辑，就需要将此逻辑分解，赋予容易撤销的方式。以下是在自定对象中实现撤销支持的一些技巧：

- 定义您需要的方法，以确定对您的对象的修改是对称的。例如，如果您定义了一个方法来添加项目，请确定也有一个方法，用相似的方式来移除项目。
- 把您的业务逻辑，从您用来修改成员变量值的代码中分解出来。
- 对于多步骤操作，请使用现有的 `NSUndoManager` 对象，将步骤组合起来。

使用 Core Data 定义结构化数据模型

Core Data 是一种模式驱动的 (schema-driven) 对象图管理和持久性框架。从根本上讲，Core Data 帮助您将模型对象（以“模型－视图－控制器”设计模式的方式）存储到一个文件中，然后再将它们取回来。这类似于归档，但 Core Data 远不止那些。

- Core Data 提供了一个基础结构，来管理对模型对象所做的修改。它让您自动支持撤销和重做，以及维持对象之间的相互关系。
- 它允许您在任何给定的时间内，仅将模型对象的子集保存在内存中，这对于 iOS 应用程序是非常重要的。
- 它使用模式来描述模型对象。在基于 GUI 的编辑器中，您定义模型类的主要功能（包括模型类之间的关系）。模式“免费”提供了丰富的基本功能，包括设定默认值和验证属性值。
- 它允许您维护编辑对象的不相交集合。不相交集合很实用，例如，您想要允许用户在一个视图中进行可被放弃的编辑，而不影响另一个视图中显示的数据。
- 它具有的基础结构，用于数据储存版本管理和迁移。版本管理可让您轻松地将旧版本的用户文件升级到当前版本。
- 它允许您在 iCloud 中储存数据，然后从多个设备访问数据。

定义基于文稿的数据模型

基于文稿的数据模型，可以便捷地用于管理被应用程序写入磁盘的文件。这种类型的数据模型，让您使用文稿对象来表示磁盘上单个文件（或文件包）的内容。该文稿对象负责读写文件的内容，并配合应用程序的视图控制器工作，将文稿的内容显示在屏幕上。文稿对象的传统用途，是管理包含用户数据的文件。例如，创建并管理文本文件的应用程序，会使用单独的文稿对象管理每个文本文件。但是，您可以将文稿对象用于专用的应用程序数据（也通过文件来支持）。

图 2 说明了应用程序的数据模型中的文稿、文件和对象之间的典型关系。在少数例外情况下，每个文稿是自包含的，不直接与其他文稿交互。文稿管理单一的文件（或文件包），并在内存中标识在该文件中找到的任何数据。因为每个文件的内容是唯一的，与每个文稿相关联的数据结构，也是唯一的。

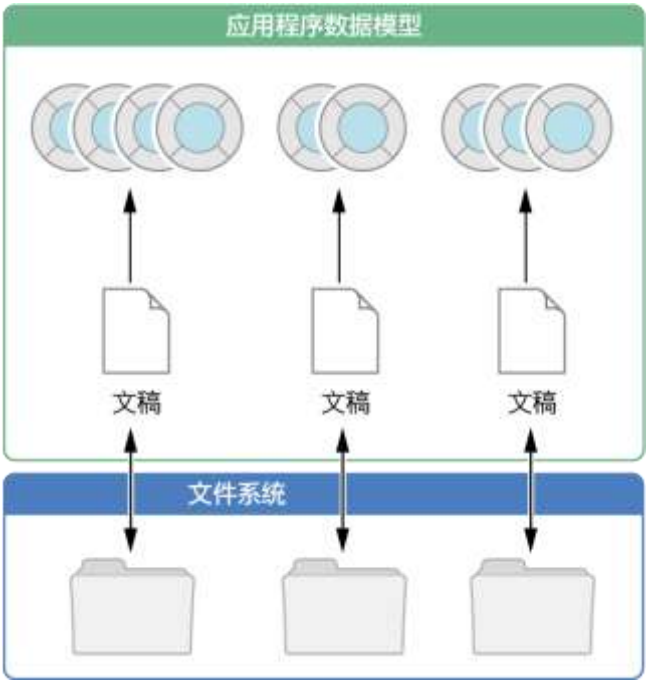


图 2 使用文稿来管理文件的内容

您使用 `UIDocument` 类来实现 iOS 应用程序中的文稿对象。此类提供基本的基础结构，以处理文稿的文件管理各方面所需。`UIDocument` 的其他好处包括：

- 它支持在合适的时间自动存储文稿内容。
- 它为储存在 iCloud 中的文稿处理所需要的文件协调。它还为解决版本冲突提供了钩子 (hook)。
- 它为撤销操作提供了支持。

为了实施应用程序的文稿所要求的特定行为，您必须创建 `UIDocument` 的子类。

用户界面

每个 iOS 应用程序，都至少有一个窗口和一个视图，来显示它的内容。窗口提供了在其中显示内容的区域，是 `UIWindow` 类的实例。视图负责管理内容的绘制（并处理触摸事件），是 `UIView` 类的实例。对于使用视图对象构建的界面而言，应用程序的窗口自然包含多个视图对象。对于使用 OpenGL ES 构建的界面而言，通常只有一个视图，并使用该视图来渲染内容。

视图控制器也在应用程序的用户界面中，扮演很重要的角色。视图控制器是 `UIViewController` 类的实例，负责管理一组视图，以及那些视图与应用程序的其他部分之间的交互。因为 iOS 应用程序显示内容的空间很有限，视图控制器也提供了所需要的基础结构，从一个视图控制器中撤出视图，以另一个视图控制器中的视图来替换。因此，视图控制器是您实施各种类型的内容转换的方式。

您要经常将一个视图控制器对象，作为自包含的单元来看。它处理其自身视图的创建和销毁，处理其视图在屏幕上的显示，并协调视图和应用程序中的其他对象之间的交互。

使用 UIKit 视图构建界面

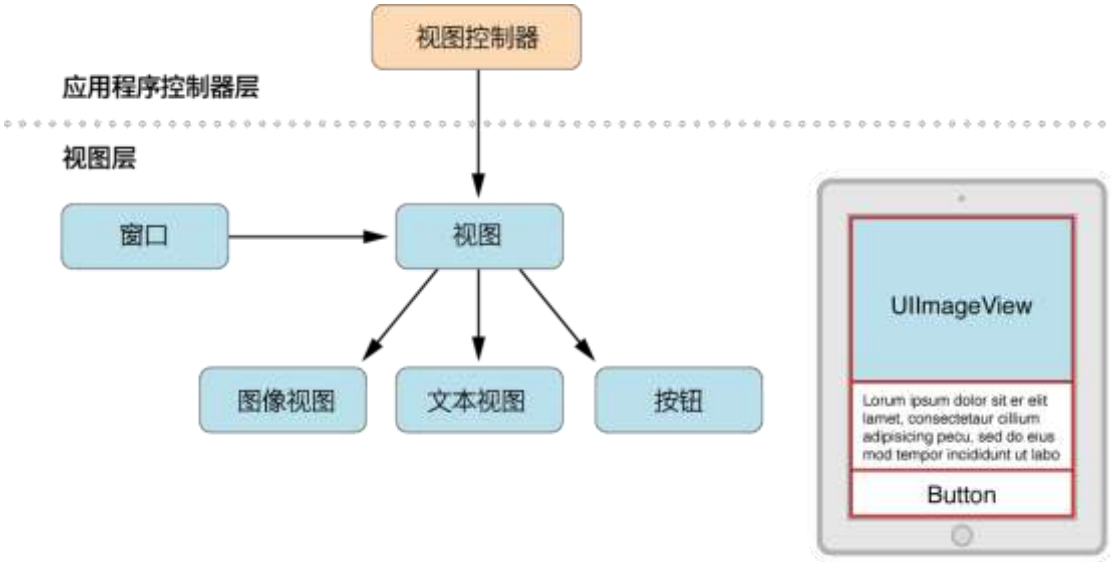
使用 UIKit 视图进行绘图的应用程序很容易创建，因为您可以快速组装一个基本界面。UIKit 框架提供了许多类型的视图，来帮助呈现和组织数据。控制是一种特殊类型的视图，它提供了一个内建机制，每当用户执行了合适的操作，即执行自定代码。例如，点按按钮，导致与该按钮关联的操作方法被调用。

基于 UIKit 视图的界面的好处是，您可以使用 Interface Builder（内建在 Xcode 中的可视化界面编辑器）以图形方式组装它们。Interface Builder 提供了一个资源库，包含了标准视图、控制，以及构建界面所需要的其他对象。将对象从资源库拖出并放置在工作前台，而且可以根据需要随意进行安放。接着可使用检查器配置对象，然后再将它们存储到串联图或 nib 文件中。以图形方式组装界面的过程，比编写同等代码要快得多，您立即看到结果，而不需要先生成并运行应用程序。

注：您也可以将自定视图并入到 UIKit 视图层次中。自定视图是 `UIView` 的子类，您自行在其中处理所有绘图和事件处理任务。

图 3 显示了应用程序的基本结构，其界面仅使用视图对象构建。在此实例中，主视图利用了窗口的可见区域（状态条除外），提供一个简单的白色背景。主视图还包含三个分视图：一个图像视图、一个文本视图和一个按钮。应用程序使用那些分视图，来向用户显示内容并响应交互。层次中的所有视图，都是由一个视图控制器对象来管理。

图 3 使用视图对象构建界面



在典型的基于视图的应用程序中，您使用视图控制器对象来协调屏幕视图。应用程序总是有一个视图控制器，负责在屏幕上显示所有内容。该视图控制器具有一个内容视图，其本身可能包含其他视图。有些视图控制器也可以用作容器，供其他视图控制器提供的内容使用。例如，分离视图控制器会并排显示两个视图控制器的内容。

使用视图和 OpenGL ES 构建界面

游戏以及其他需要很高帧速率或复杂绘图功能的应用程序，可以将专门为 OpenGL ES 绘图而设计的视图添加到其视图层次中。最简单的 OpenGL ES 应用程序类型，具有一个窗口对象、一个视图（用于 OpenGL ES 绘图），以及一个视图控制器，来管理内容的显示和旋转。更复杂的应用程序，可以混用 OpenGL ES 视图和 UIKit 视图，来实施其界面。

图 4 显示了一个应用程序的配置，该程序只使用一个 OpenGL ES 视图来绘制其界面。不像 UIKit 视图，OpenGL

ES 视图是由一种不同类型的层对象（CAEAGLLayer 对象）来支持的，而不是基于视图的应用程序所使用的标准层。CAEAGLLayer 对象提供了 OpenGL ES 可以将内容渲染在其中的绘图表面。要管理绘图环境，应用程序也创建 EAGLContext 对象，并将该对象连同视图储存下来，使它容易取回。

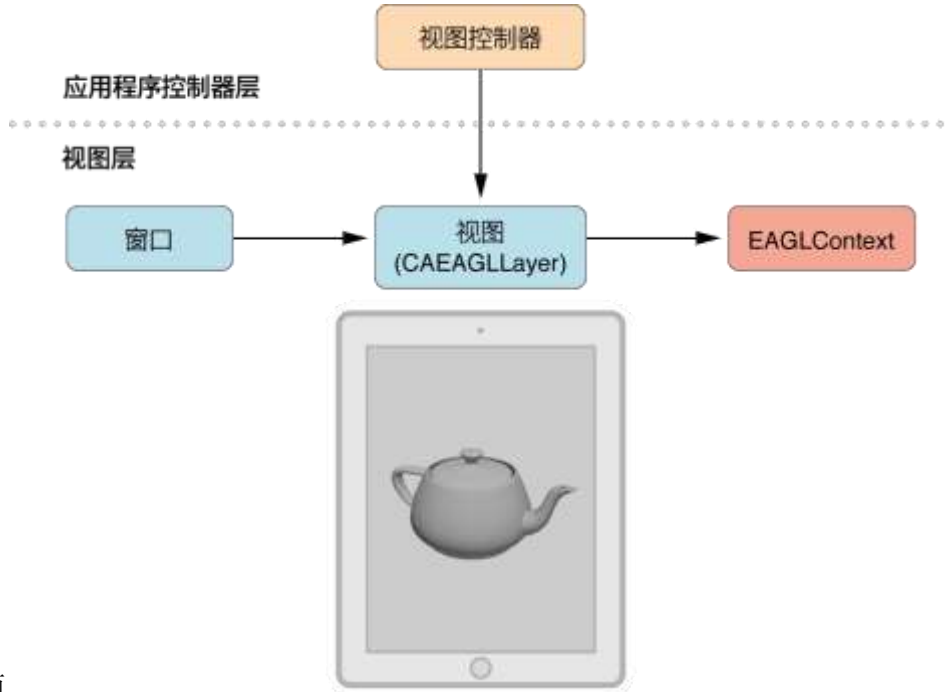


图 4 使用 OpenGL ES 构建界面

应用程序包

当您生成 iOS 应用程序时，Xcode 将它捆绑成一个包。**捆绑包** (bundle) 是文件系统中的一个目录，它将相关资源成组在一个地方。一个 iOS 应用程序捆绑包中，含有其可执行文件和支持资源文件（如应用程序图标、图像文件和已本地化的内容）。表 3 列出了一个典型的 iOS 应用程序捆绑包（名为 MyApp，用于演示目的）的内容。此示例仅用于图解目的。此表中列出的某些文件，可能不会出现在您自己的应用程序捆绑包中。

表 3 一个典型的应用程序捆绑包

文件	示例	说明
应用程序可执行文件	MyApp	可执行文件包含应用程序的已编译代码。将应用程序名称去掉 .app 扩展名之后，就是其可执行文件的名称。 此文件是必需的。
信息属性列表文件	Info.plist	Info.plist 文件包含应用程序的配置数据。系统使用此数据来确定如何与应用程序交互。 此文件是必需的，而且名称必须为 Info.plist。

应用程序图标	Icon.png Icon@2x.png Icon-Small.png Icon-Small@2x.png	应用程序图标用来在设备的主屏幕上表示应用程序。其他图标由系统用在合适的地方。其文件名中含有 @2x 的图标，专用于配备 Retina 显示屏的设备。 应用程序图标是必需的。
启动画面	Default.png Default-Portrait.png Default-Landscape.png	应用程序启动时，系统将此文件用作临时背景。一旦应用程序准备好显示其用户界面，它就会被移走。 至少需要一个启动画面。
串联图文件（或 nib 文件）	MainBoard.storyboard	串联图含有视图和视图控制器，应用程序通过它们，将内容呈现在屏幕上。串联图中的视图，是根据显示它们的视图控制器来组织的。串联图也确定一组视图的转换（称为过渡），将用户从一组视图带到另一组。 当您创建项目时，主串联图文件的名称由 Xcode 设定。您可以通过给 Info.plist 文件中的 NSMainStoryboardFile 键指定其他值，来更改该名称。使用 nib 文件（而不是串联图）的应用程序，可以使用 NSMainNibFile 键替换 NSMainStoryboardFile 键，并使用该键来指定其主 nib 文件。 使用串联图不是硬性规定，但建议您还是使用它。
临时分发图标	iTunesArtwork	如果您准备临时分发应用程序，请准备好一个 512 x 512 像素版本的应用程序图标。此图标正常由 App Store 从您提交到 iTunes Connect 的材料中提供。然而，由于应用程序是临时分发的，没有通过 App Store 审批，您的图标必须存在于捆绑包中。iTunes 使用此图标来表示您的应用程序。（如果您准备用那种方式来分发应用程序，则指定的文件就应该与您已提交到 App Store 的那个文件一样。） 此图标的文件名必须是 iTunesArtwork，不包括文件扩展名。此文件对于临时分发是必需的，其他情况下属可选。
设置捆绑包	Settings.bundle	如果想要通过“设置”应用程序，来展示自定的应用程序偏好设置，您必须准备好一个设置捆绑包 (settings bundle)。此捆绑包含有属性列表数据，还包含定义应用程序偏好设置的其他资源文件。“设置”应用程序使用此捆绑包中的信息，来组装所需要的界面元素。 此捆绑包是可选的。
非本地化资源文件	sun.png mydata.plist	非本地化资源包括应用程序所使用的图像、声音文件、影片和自定数据文件之类的资源。所有这些文件，都应该放置在应用程序捆绑包的顶层位置。

本地化资源的子目录	en.lproj fr.lproj es.lproj	本地化资源必须放置在语言特定的项目目录中，目录的名称由 ISO 639-1 语言缩写和 .lproj 后缀组成。（例如，en.lproj、fr.lproj 和 es.lproj 目录，含有本地化为英文、法文和西班牙文的资源。） iOS 应用程序应该是国际化的，对于所支持的每一种语言，都有一个**语言**.lproj 目录。除了为应用程序的自定资源提供本地化版本以外，您还可以本地化其图标、启动画面和“设置”图标，方法是将同名文件放入特定语言的项目目录中。

将您的应用程序国际化

App Store 中很多流行的应用程序有多种语言版本。虽然这些应用程序可能因为很多因素而变得流行，但是具有多种本地化版本，肯定是其中一个因素。越多的人可以理解并使用您的应用程序，潜在的买家也就越多。

若要让您的应用程序拥有多个语言版本，必须先将它国际化，然后将它本地化。国际化是整理本地化资源的一种技巧，以便应用程序在运行时，可以选择用户首选的资源集。本地化就是翻译应用程序所显示或读出（例如 VoiceOver）的文本。它还可以包括某个区域专用的额外图像和其他资源。（本地化也可以指将一组资源本地化为某个特定语言和区域设置——例如简体中文本地化。）

应用程序选择用户界面本地化语言的方法：在“设置”应用程序（“通用”>“多语言环境”>“语言”）中，iOS 设备用户选取想在应用程序用户界面和系统本身所显示的语言。应用程序将此偏好设置作为钥匙，来访问所请求的语言本地化资源。

若要了解国际化的所有方面，请参阅**国际化编程主题**。

本教程将教您进行的操作

在本教程中，您将会使 HelloWorld 应用程序国际化，以便其支持两种本地化语言：英文和简体中文。接下来您会将以下文本本地化：

- 串联图中两种语言的文本
- 用户点按“Hello”按钮后，应用程序所构建和显示的字符串
- 显示给用户的应用程序名称

注：本教程使用简体中文作为示例语言。如果不懂中文，您可以拷贝本教程中本地化文本项的中文字，然后将它们粘贴到项目中的合适位置。或者，如果您喜欢，可以在添加的本地化语言中，使用简体中文之外的某种语言。

在本教程中，您还将修改用户界面并配置国际化的布局约束。

本教程不会教您如何将本地化资源（例如图像文件或声音文件）添加到项目。若要了解如何将本地化资源添加到项目，请阅读**国际化编程主题**中的相应章节。

当您选择简体中文作为首选语言，然后启动国际化的 HelloWorld 后，该应用程序外观应该是这样的：

使用 Base Internationalization

“Base Internationalization”是 Xcode 4.5 推出的一项功能，使用该功能，本地化工程师（即翻译人员）不再需要为应用程序支持的每种语言修改串联图和 nib 文件。相反，一个应用程序只有一组串联图或 nib 文件会本地化为默认语言，这些串联图和 nib 文件称为 *Base Internationalization*。当您将本地化语言添加到某个应用程序时，Xcode 会生成一个包含所有文本的字符串文件，这些文本包含每个串联图或 nib 文件显示的文本，或者作为辅助功能标签

或提示的文本。Xcode 会以串联图的名称命名该文件，文件的扩展名为 strings。因此，如果串联图的名称为 MyStoryboard.storyboard，所生成的字符串文件的名称就是 MyStoryboard.strings。

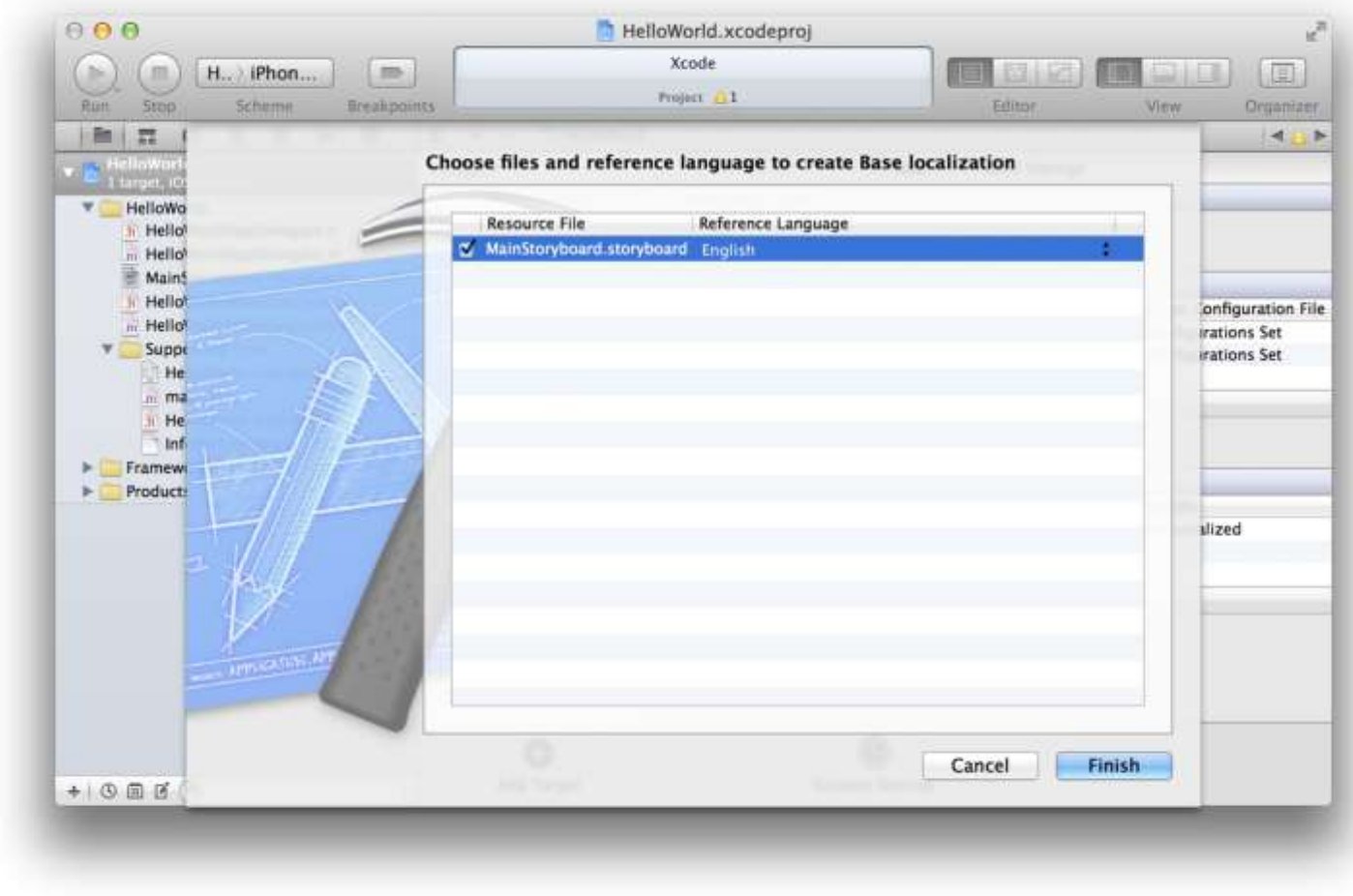
正如您将看到的，字符串文件将应用程序中的文本（值）与其他字符串（键）相关联。本地化工程师使用该键以帮助识别用户界面中的文本，然后翻译该文本。在“Base Internationalization”中构建应用程序的用户界面时，必须使用“Auto Layout”功能，以确保显示翻译字符串的对象，在其相邻视图更改时，适当地调整其相对位置和大小。

使用 Base Internationalization

1. 在 Xcode 中，选择 HelloWorld 项目，并显示“Info”面板。

2. 选择“Localizations”表格下方的“Use Base Internationalization”复选框。

   Xcode 显示一张表单，要求您选择要用于“Base Internationalization”的串联图。



3. 请确定已选中 `MainStoryboard.storyboard`，并且参考语言为“English”。点按“Finish”按钮。
4. 在项目导航器中选择 `MainStoryboard.storyboard (Base)`。
5. 如果“Label”显示在标签对象中，请连按它以选择该文本，然后按下“Delete”。

   这最后一步是必须的，因为用户应该是看不到“标签”的，因此不需要翻译它。

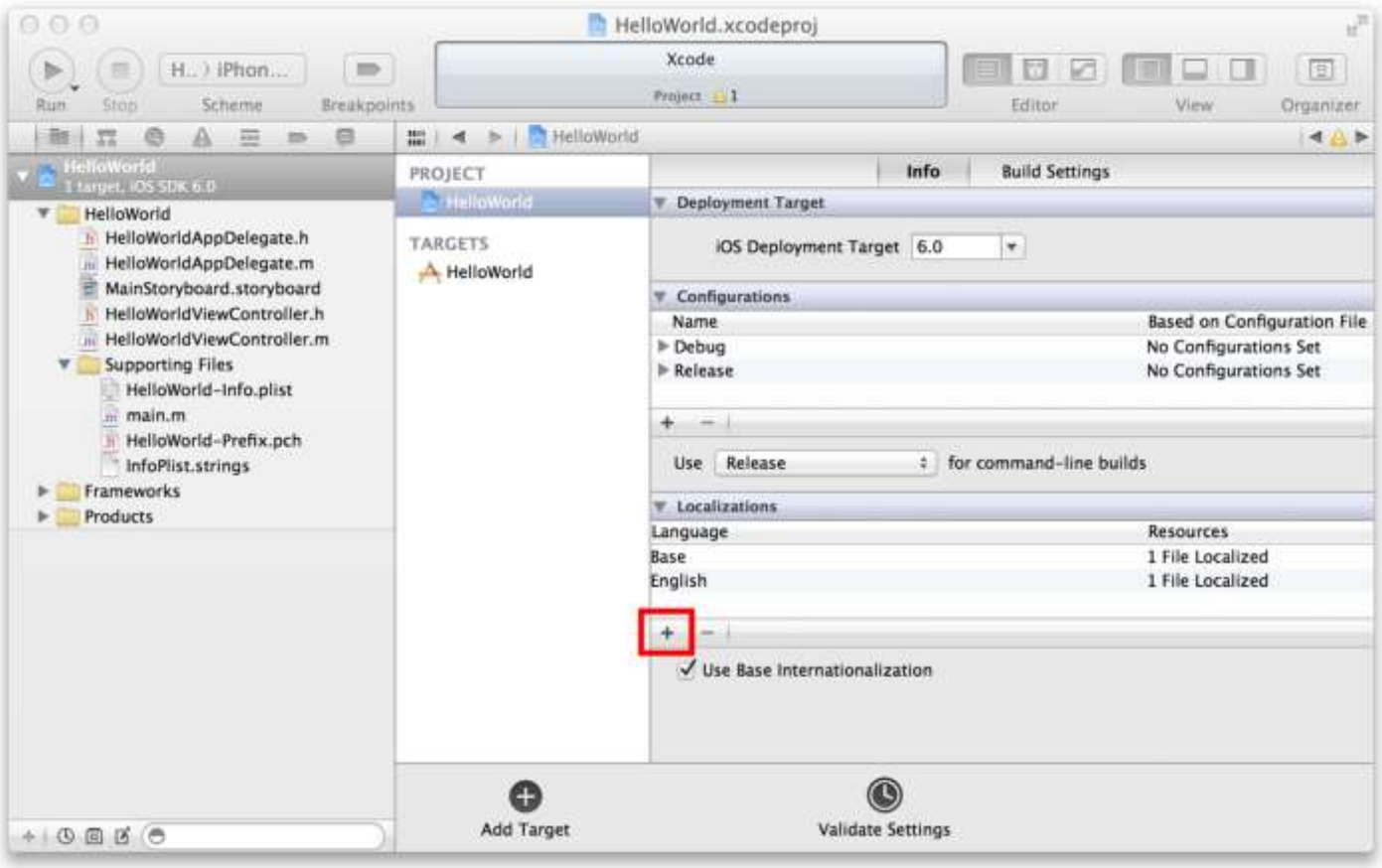
注：在 Xcode 4.5 以前的版本中，开发者和本地化工程师必须为应用程序支持的每种语言修改串联图和 nib 文件。这项任务需要他们不仅要翻译串联图或 nib 文件中的可见或可听文本，还要在翻译完成后，调整视图的大小和位置。

添加本地化语言

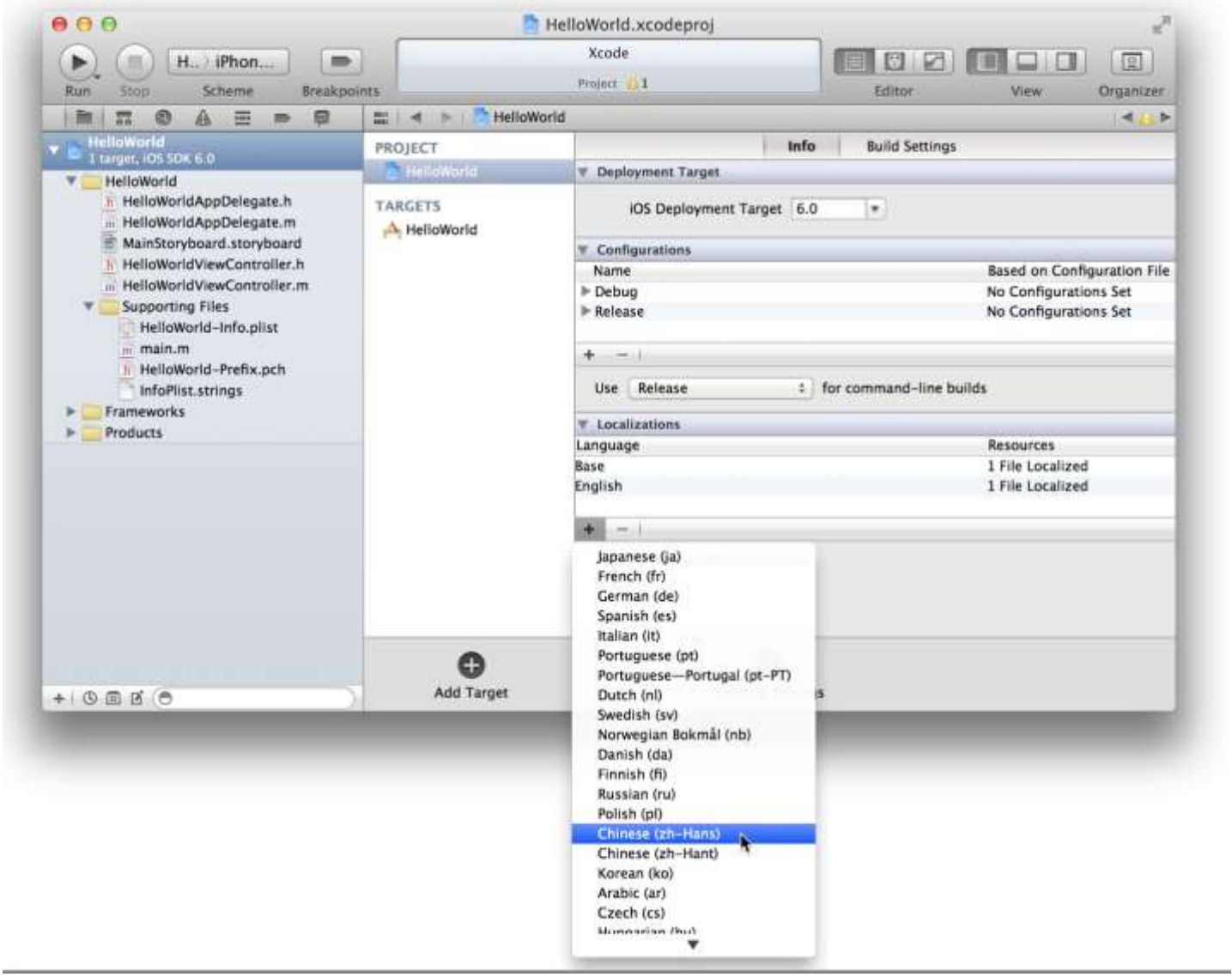
创建 HelloWorld 时，Xcode 已自动将一个文件夹添加到了 Xcode 项目。此文件夹用于存放英文本地化的资源。稍后，当您请求某个“Base Internationalization”时，Xcode 会将另一个文件夹添加到项目。如果在 Finder 中查看项目文件，您将看到一个名为 `Base.lproj` 的文件夹和另一个名为 `en.lproj` 的文件夹（用于英文本地化）。第一个文件夹中的是串联图文件；第二个文件夹中的是 `InfoPlist.strings` 文件。（您将在“将应用程序名称本地化”中了解有关 `InfoPlist.strings` 文件的更多信息。）英语是项目的默认语言。如果想要添加其他的本地化语言，您必须在 Xcode 中添加它们。

将某种本地化语言添加到项目

1. 选择项目设置的“Info”面板。

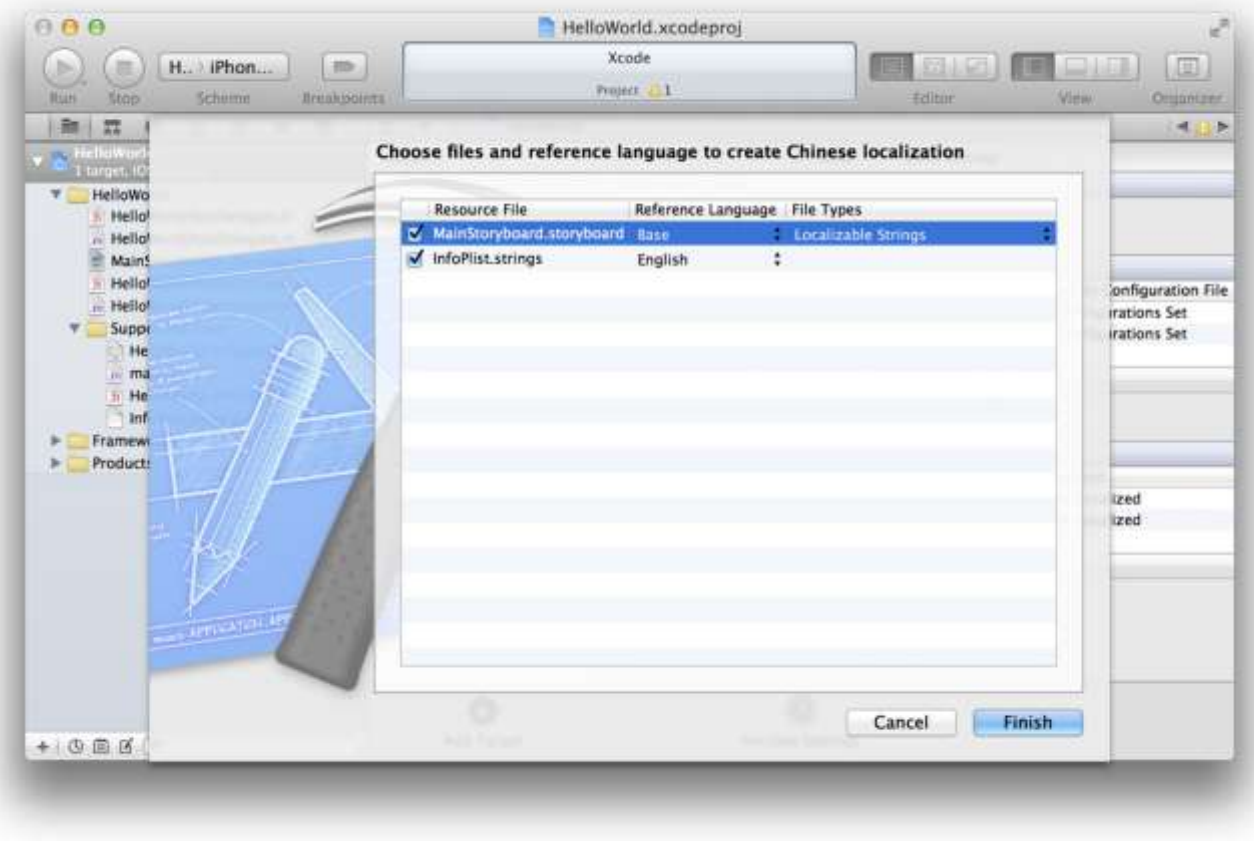


2. 点按“Localization”表格中的加号按钮 (+)，然后从弹出式菜单中选取简体中文。



简体中文的语言 ID 标识为 `zh-Hans`。

3. 在要求您选取文件和参考语言的表单中，请确定所有设置和选择都如下所描述。然后点按“Finish”按钮。



在完成此项任务后，Xcode 会更新该项目导航器，以显示新的本地化语言。点
按 `MainStoryboard.storyboard` 旁边的展示三角形，以显示这些文件的基本（英文）和简体中文本地化。

在项目导航器中，您可能注意到有两种类型的字符串文件。当您添加一种本地化语言时，Xcode 生成两种字符串文件。第一种是来自“Base Internationalization”的字符串文件 (`MainStoryboard.strings`)。该字符串文件包含一个或多个键－值对，它们由 Xcode 从串联图文件找到的文本自动生成。第二种字符串文件，是 `InfoPlist.strings`，您使用该文件将用户可见的应用程序属性（例如其名称）本地化。该文件开始内容为空。您将在接下来的部分了解有关这两种字符串文件的更多信息。

应用程序捆绑包中的本地化文件夹：应用程序就是捆绑包，而捆绑包即是结构化目录，内有可执行代码及其使用的资源。这些资源可以本地化或不加以本地化（所有语言都如此）。本地化的资源，放在名为 *code*`.lproj` 的捆绑包的文件夹中，而 *code* 可由以下组成：

- **语言 ID**。语言的 ISO 639-1（两个字母）或 ISO 639-2（三个字母）标识符。（ISO 代表“国际标准化组织”。）通过添加带有连字符的后缀，您可以指定某种语言的方言，例如简体中文 (`-Hans`) 和繁体中文 (`-Hant`)。
- **区域 ID**。一个区域或地区的可选 ISO 3166-1 标识符。区域 ID 是一个由两个字母组成的大写代码，该代码使用下划线字符连结语言指示符。例如，ID `en_GB.lproj` 指的是英文–英国。

这些约定的一个例外是 `Base.lproj`，它标识串联图或 nib 文件的 Base Internationalization。

使用正确的输入源

如果要为简体中文本地化键入中文字，您需要一个合适的输入源，来输入这些字符。您在 OS X 的“系统偏好设置”中请求此输入源。

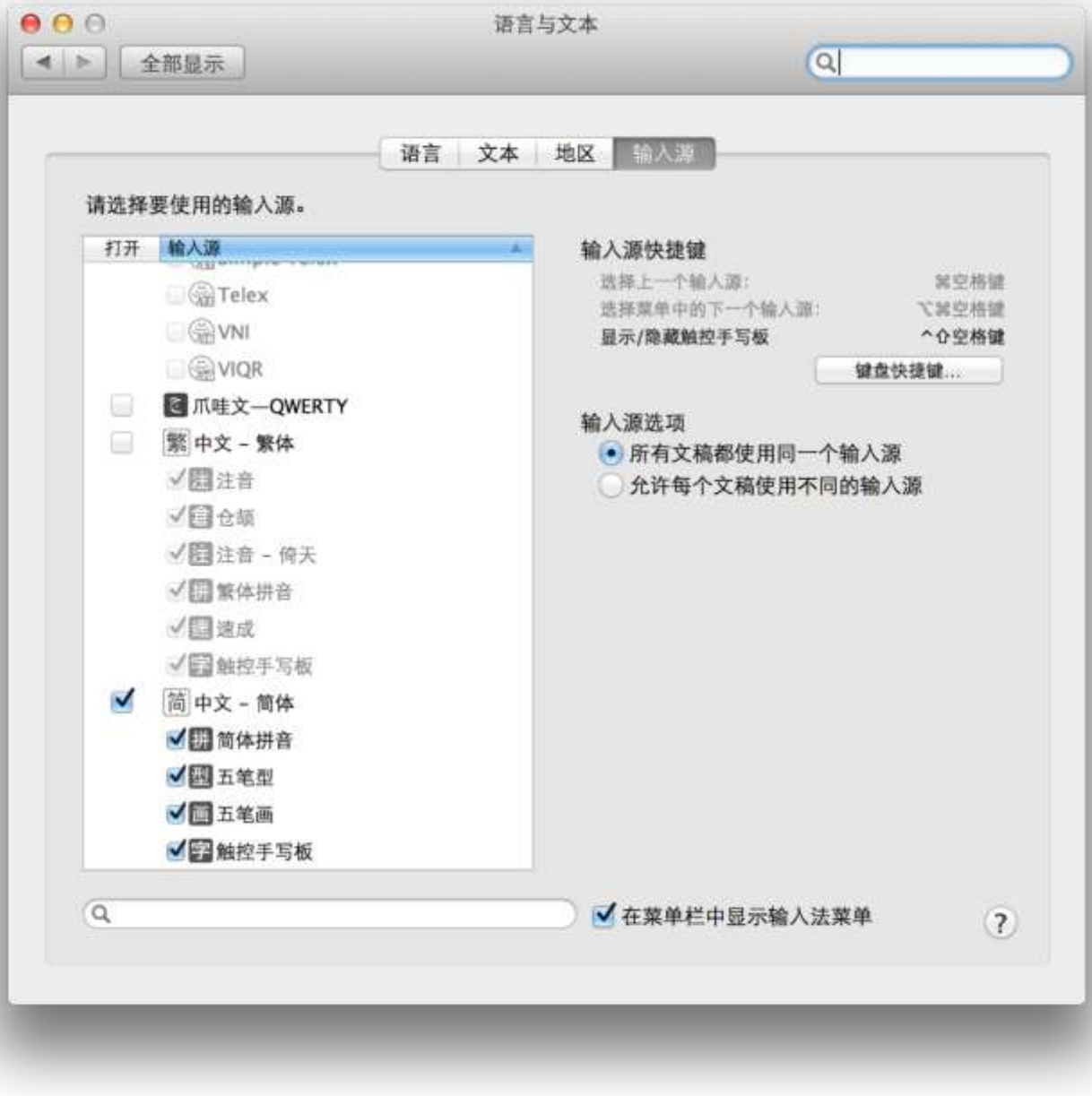
选择适合简体中文的输入源

1. 启动“系统偏好设置”应用程序，然后选择“语言与文本”。

   若要启动“系统偏好设置”，请从苹果菜单中选取“系统偏好设置”。（这不是 Xcode 的一项功能。）

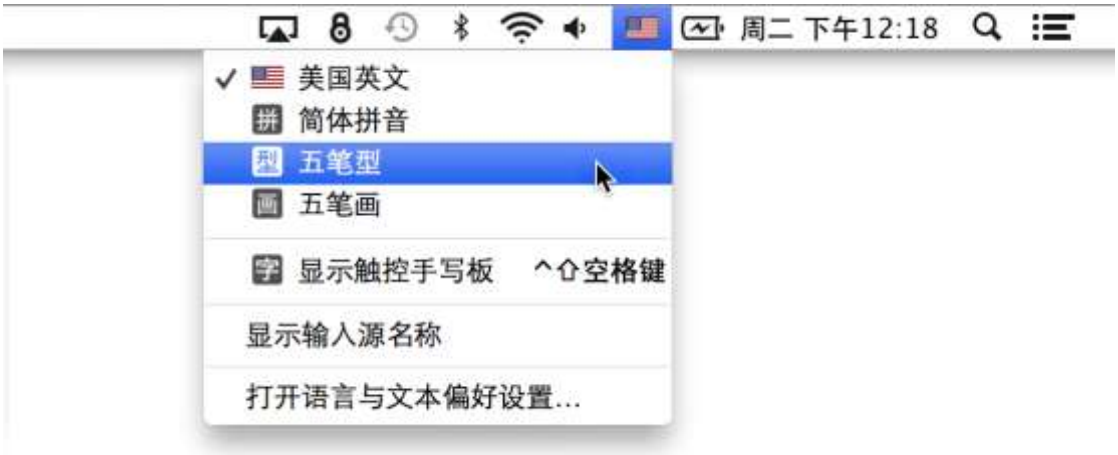
2. 选取“输入源”视图。

3. 在输入源列表中，选择“中文 - 简体”。



4. 请确定已选定“在菜单栏中显示输入法菜单”。

现在，您可以在菜单栏中选取一种简体中文输入源，以将其设为活跃的输入源。



关于字符串文件

字符串文件包含文本项，这些文本项已本地化为特定语言，并与用作键的任何字符串相匹配。该文件的扩展名为 `strings`。当应用程序在设备上运行时，它会从用户首选语言所属的字符串文件中，找出并显示这些字符串。（当然，该应用程序必须支持该语言的本地化。）在 iOS 中，国际化使用三种不同类型的字符串文件：

- Xcode 从“Base-internationalization”串联图或 nib 文件自动生成的字符串文件
- 用于本地化用户可见属性的字符串文件，这些属性存在于应用程序的信息属性列表，如应用程序的显示名称
- 由应用程序代码所创建和显示的字符串的文件

以下部分描述了，将应用程序国际化时如何配置各种类型的字符串文件。

所有字符串文件中的条目，具有以下基本格式：

```
/* Comment to localizers */

"Key" = "Value";
```

该注释旨在通过澄清已本地化字符串的上下文，来帮助本地化工程师。注释（位于单独一行）之后是一个键、一个等号、一个值和一个表示终止的分号。值是一个总被引号括起来的字符串。键是通常被引号括起来的字符串，但也可以是声明为全局字符串的符号。

将串联图文本本地化

对于每种添加到应用程序项目的本地化语言，Xcode 从“Base-internationalization”串联图生成名为 *StoryboardName*`.strings` 的字符串文件，并将该文件写入该项目本地化语言（例如 `zh-Hans.lproj`）的文件夹。

检查简体中文的字符串文件

1. 在项目导航器中，点按展示三角形，以显示 `MainStoryboard.storyboard` 下方的项目。
2. 选择名称为 `MainStoryboard.strings (Chinese)` 的项目。

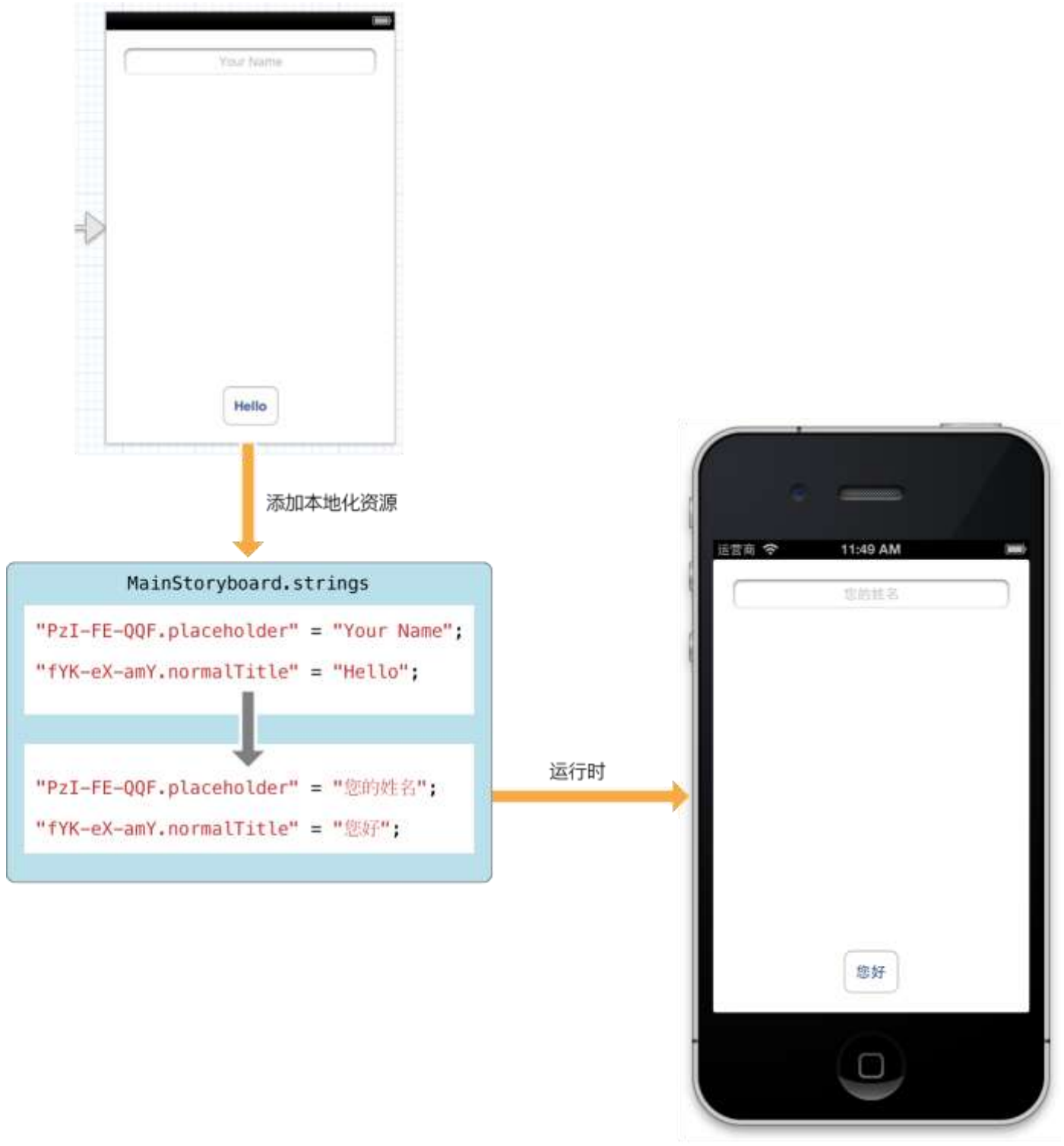
   该字符串文件的内容，会显示在项目编辑器中。

就 HelloWorld 应用程序而言，Xcode 从“Base Internationalization”生成以下字符串文件：

```
/* Class = "IBUITextField"; accessibilityHint = "Type your name"; ObjectID = "PzI-FE-QQF"; *
"PzI-FE-QQF.accessibilityHint" = "Type your name";

/* Class = "IBUITextField"; placeholder = "Your Name"; ObjectID = "PzI-FE-QQF"; */
"PzI-FE-QQF.placeholder" = "Your Name";

/* Class = "IBUIButton"; normalTitle = "Hello"; ObjectID = "fYK-eX-amY"; */
"fYK-eX-amY.normalTitle" = "Hello";
```

对于串联图中的每个字符串，Xcode 将文本显示对象标识符的键、句点 (.) 和指定给该字符串的属性连接起来。值（等号后面的字符串）是“Base-internationalization”串联图中的可见或可听字符串。本地化工程师会翻译这些字符串。（每个条目的注释，会重复此信息，但会添加 Xcode 已知的对象类。）如果用户的首选语言与“Base Internationalization”的语言不同，应用程序在运行时会将“Base-internationalization”串联图中的每个字符串，动态替换为当前本地化语言的 *StoryboardName*`.strings` 文件中相匹配的字符串。图 1-1 说明了此流程。

图 1-1 将“Base Internationalization”中的字符串本地化



将字符串本地化为简体中文

1. 从菜单栏中选取一种简体中文输入源。

   您可以使用中文键盘（如果有）。您也可以使用其他键盘（例如，“美国英文”键盘），但应该知道它的按键映射。

   如果拷贝下面示例中的中文字，并将它们粘贴到字符串文件中（或者如果将文本本地化为其他语言），您可以

跳过此步骤。

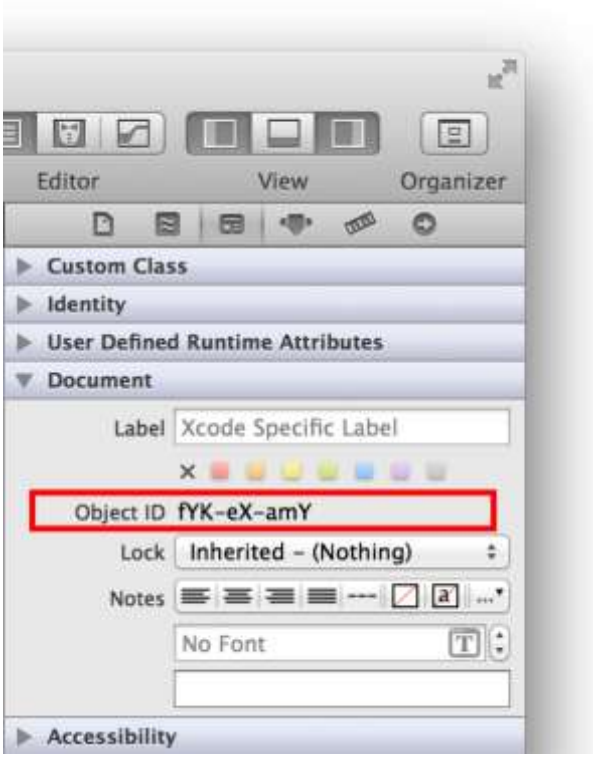
2. 选择项目导航器中的 MainStoryboard.strings (Chinese) 文件，以在项目编辑器中显示该文件。

3. 将每个值字符串（即等号右边的字符串）翻译成简体中文。

   如果您要拷贝并粘贴中文，以下是可供拷贝的字符。

4. 存储该文件（“File”>“Save”）。

HelloWorld 用户界面十分简单，仅有一个串联图、一个场景和三个字符串。用户界面更为复杂的应用程序，含有许多带文本的视图，因此有许多字符串需要本地化。Xcode 使用嵌入键中的对象 ID，帮助您识别显示文本的视图。选择串联图中的视图（例如，“Hello”按钮），然后打开“Identity”检查器。在此检查器的“Document”部分，查找对象 ID，并将其与字符串文件中的对象 ID 相匹配。



MainStoryboard.strings 文件的中文本地化内容，现在看起来应该是这样的：

```
/* Class = "IBUITextField"; accessibilityHint = "Type your name"; ObjectID = "PzI-FE-QQF"; *
"PzI-FE-QQF.accessibilityHint" = "键入您的姓名";

/* Class = "IBUITextField"; placeholder = "Your Name"; ObjectID = "PzI-FE-QQF"; */
"PzI-FE-QQF.placeholder" = "您的姓名";

/* Class = "IBUIButton"; normalTitle = "Hello"; ObjectID = "fYK-eX-amY"; */
"fYK-eX-amY.normalTitle" = "您好";
```

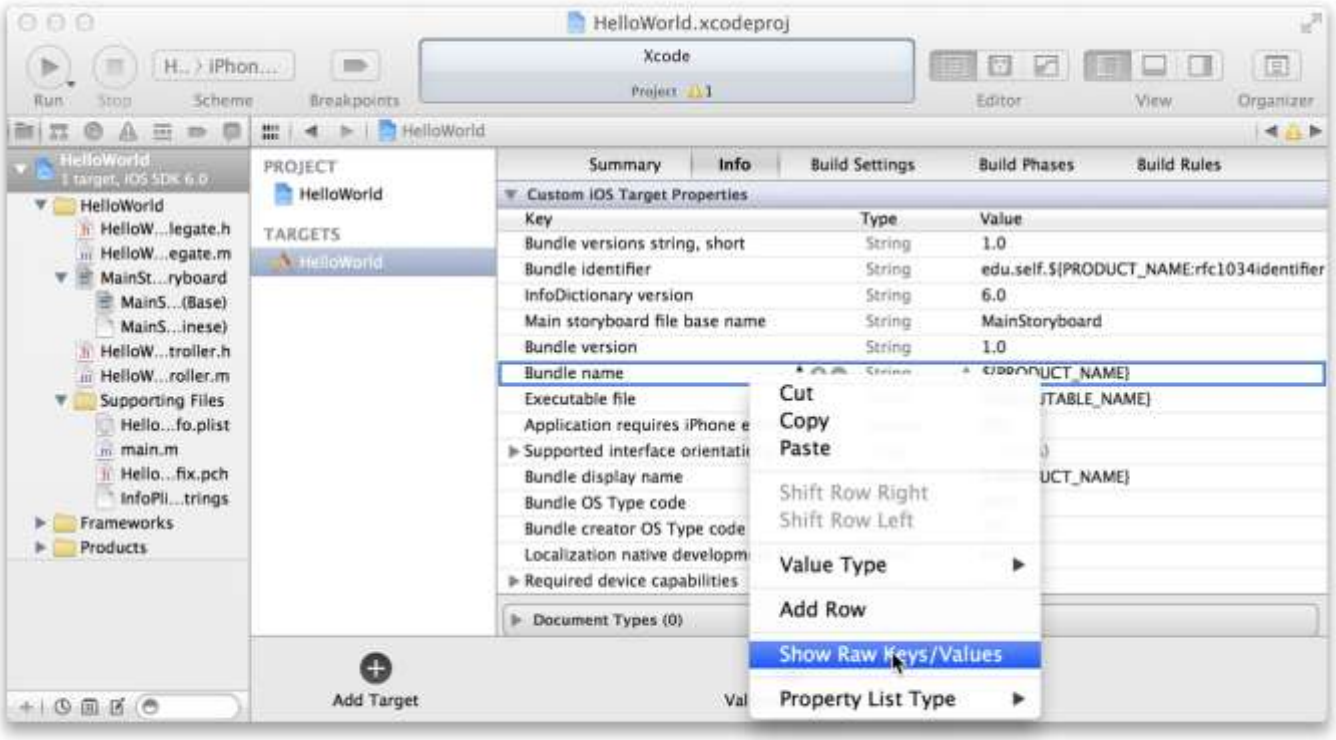
将应用程序的名称本地化

将本地化语言添加到某个应用程序项目时，Xcode 在默认情况下，会创建一个名称为 InfoPlist.strings 的文件，并将其写入该文件系统的本地化文件夹（例如，zh-Hans.lproj）中。该文件（最开始内容为空）应该包含信息属性列表中用户可见的本地化字符串的键－值对；就 HelloWorld 应用程序而言，该属性列表在 HelloWorld-Info.plist 文件中。

最重要的用户可见属性，是应用程序的显示名称，现在您会将其本地化。InfoPlist.strings 文件中的键，是由框架声明的符号，可在信息属性列表中找到。

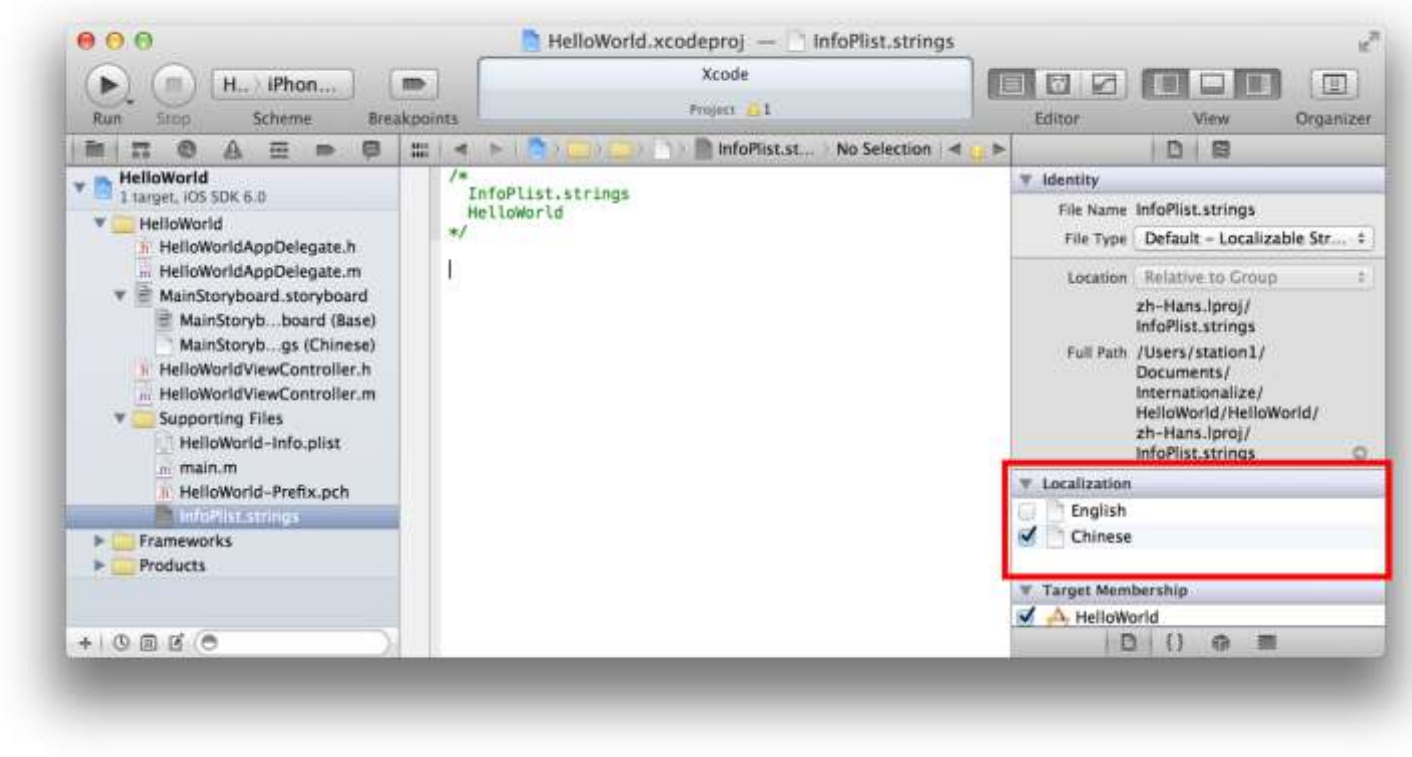
将应用程序名称本地化

1. 选择 HelloWorld 目标，然后选取“Info”视图。
2. 按住 Control 键，点按属性表格中的任何地方，然后从关联菜单中，选取“Show Raw Keys/Values”。



3. 找到属性 CFBundleDisplayName（它是表示某个应用程序的显示名称的 Core Foundation 符号）的位置。
4. 在项目导航器中，选择 InfoPlist.strings 文件，以在编辑器中显示该文件。

   该文件用于简体中文本地化。可以在“Identity”检查器的“Localization”部分，验证所选文件的本地化情况。



5. 在 InfoPlist.strings 文件中，键入以下代码行，然后从“File”菜单中，选取“Save”以存储该文件。

```
/* The name of the app displayed on the device*/

CFBundleDisplayName = "您好，世界";
```

6. 在本例中，可以不必给键加引号。

不需要为原始的英文本地化编辑 InfoPlist.strings 文件，因为信息属性列表中用户可见的属性已经是英文的。

将代码显示的字符串本地化

应用程序的代码，经常在运行时创建和显示文本字符串。HelloWorld 项目的 changeGreeting: 方法，就是一个以编程方式生成文本的例子。

```
- (IBAction)changeGreeting:(id)sender {

    self.userName = self.textField.text;



    NSString *nameString = self.userName;

    if ([nameString length] == 0) {
```

```
        nameString = @"World";

    }

    NSString *greeting = [[NSString alloc] initWithFormat:@"Hello, %@", nameString]; // program
created

    self.label.text = greeting;

}
```

该方法从文本栏获取要显示的姓名，然后使用 `NSString` 类的 `initWithFormat:` 方法，将字符串“Hello, ”与该姓名连接起来。为了显示此文本，该方法将标签的 `text` 属性设定为构建的字符串。这里国际化的问题，在于必须将单词“Hello”翻译为该应用程序支持的每种本地化语言。为了将代码生成的字符串本地化，iOS 的国际化功能提供了字符串文件和 `NSLocalizedString` 宏。以下部分描述如何使用该文件和宏。

配置 Localizable.strings 文件

包含以编程方式显示的文本，其本地化字符串文件的惯用名称是 `Localizable.strings`。您将创建两个此类文件，一个用于英文，一个用于简体中文，然后将合适的键－值对添加到其中。

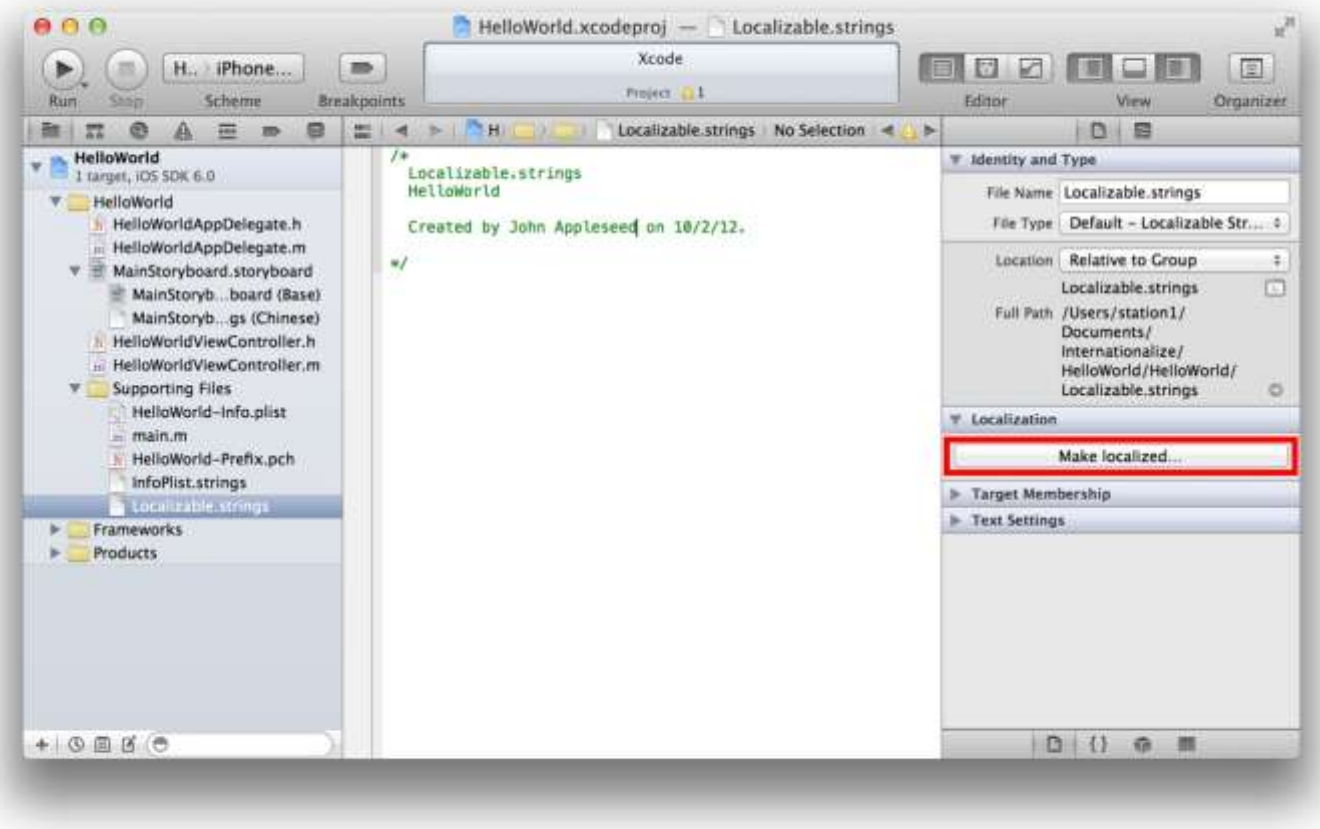
为每种本地化添加 Localizable.strings 文件

1. 在 Xcode 项目导航器中，选择“Supporting Files”文件夹。
2. 选取“File”>“New”>“File”。
3. 在“iOS”的“Resources”类别中，选择“Strings File”模板，然后点按“Next”按钮。

4. 在“Save As”栏中，给 `Localizable.strings` 文件重新命名，然后点按“Create”按钮。

   保持默认设置（例如，选定的目标）不变。

5. 选择项目导航器中的 `Localizable.strings`，然后点按“Identity”检查器中的“Make localized”按钮。



6. 在出现的确认对话框中，请确定已在下拉式菜单中选定“English”，然后点按“Localize”按钮。

此步骤完成后，“Make localized”按钮已由列出当前本地化语言的表格所替换。

7. 选择“Chinese”复选框，将 `Localizable.strings` 文件添加到简体中文本地化中。

现在，有了两种本地化语言的 `Localizable.strings` 文件，是时候将键－值对添加到其中了。

将“Hello”字符串本地化

1. 在项目导航器中，选择 `Localizable.strings (English)`。
2. 在编辑器中，键入以下文本：

```
/* The string displayed */

"HELLO" = "Hello, %@";
```

3. 为了帮助将左边的字符串（“HELLO”）识别为键，“Hello”为大写。
4. 在项目导航器中选择 `Localizable.strings (Chinese)`。
5. 在编辑器中，键入以下文本：

```
/* The string displayed */

"HELLO" = "您好，%@";
```

调用 NSLocalizedString 宏

NSLocalizedString 宏从 Localizable.strings 文件获取当前本地化语言的本地化字符串。在 HelloWorld 应用程序中，您要以此宏替换文本 @"Hello, %@"。

▶

请求代码中的本地化字符串

1. 在项目导航器中，选择 HelloWorldViewController.m。
2. 修改 changeGreeting: 方法中的以下语句：

```
NSString *greeting = [[NSString alloc] initWithFormat:@"Hello %@", nameString];
```

3. 使它看起来是这样的：

```
NSString *greeting = [[NSString alloc] initWithFormat:NSLocalizedString(@"HELLO", @"The s
displayed"), nameString];
```

该宏中的第一个参数，采用字符串文件中的一个键，第二个参数采用给本地化工程师的注释。该宏会返回与用户首选语言相对应的本地化语言中键的值。您可能想知道此示例将给本地化工程师的注释作为该宏的参数的原因；毕竟，您已经在 Localizable.strings 文件中，为英文和中文写了一则注释。

大型应用程序项目，通常使用名称为 genstrings 的命令行程序，从该程序在代码中找到的信息生成一个字符串文件（默认情况下，该文件名称为 Localizable.strings）。该实用工具查找 NSLocalizedString 宏的每次调用，提取键（也是初始值）和注释，并将这些项目写入字符串文件。然后，您可以将该字符串文件，添加到项目中的每种本地化语言中，并翻译这些值。

测试您的国际化

恭喜您。您已经完成了将应用程序国际化，并本地化为简体中文。现在该开始测试该应用程序，以检查当简体中文为用户的首选语言时，该应用程序是否显示简体中文字符串。

▶

测试应用程序的简体中文版本

1. 运行 iOS Simulator，启动 Settings 应用程序，选取简体中文作为首选语言。

   如果 iOS Simulator 正在运行 HelloWorld 或任何其他应用程序，点按主屏幕按钮。

   在 Settings 应用程序中，选取“General”>“International”>“Language”。然后点按示例中所描述表格中的行。点

按“Done”按钮。

2. 在 Xcode 中，请确定“iPhone 版本号 Simulator”是处于活跃状态的方案，然后点按“Run”按钮。

Xcode 会在 iOS Simulator 中运行 HelloWorld 应用程序。它的外观应该是这样的：

3. 从菜单栏中，选取简体中文作为输入源，在文本栏中输入一些中文字，然后点按按钮。

   该应用程序应该显示“您好，**键入的文字**”。如果发现有未翻译为当前语言的任何字符串，请确定在字符串文件中翻译它们。

4. 从“Hardware”菜单中，选取“Rotate Left”或“Rotate Right”，以模拟方向的更改，并查看方向更改的结果。

5. 点按主屏幕按钮，以退出该应用程序，并在模拟的 iPhone 中，找到该应用程序。

您会注意到，该应用程序的名称已经本地化为简体中文。

当然，大多数商业应用程序显示的文本，要比 HelloWorld 多；它们也可能包括本地化的图像和其他资源。这些应用程序也可能含有多种本地化语言版本。对于每种本地化语言，您应该运行该应用程序，并前往每个显示文本的屏幕或对话框，以查看是否存在任何未本地化的字符串或资源。

如果已配置应用程序使用 VoiceOver，您也要检查该应用程序的辅助功能是否已正确本地化。您可以使用 iOS Simulator 内建的“Accessibility Inspector”来检查。有关“Accessibility Inspector”的更多信息，请参阅 *Accessibility*

Programming Guide for iOS（iOS 辅助功能编程指南）。

使用自动布局进行国际化

测试简体中文版的 HelloWorld 应用程序时，您模拟了方向的更改。该应用程序显示文本的方式，适合任何本地化语言。因为文本栏中的占位符文本和以编程方式显示在标签中的文本，在各自对象中居中显示（占据屏幕的宽度），无论方向如何，它们可以是任何合理的长度。并且，该按钮会调整自身的大小，以适合任何本地化字符串。

当然，像 HelloWorld 用户界面一样简单的应用程序很少（即对象在屏幕上居中显示，并且旁边没有别的对象）。用户界面文本的一个常见设计，就是标签后跟着一个文本栏。当本地化工程师把 *StoryboardName*`.strings` 文件中的标签文本翻译成特定的语言时，该标签应该调整自身的大小，以适合新文本。而该文本栏的宽度应该随之更改，以配合新的标签尺寸，如图 1-2 所示。

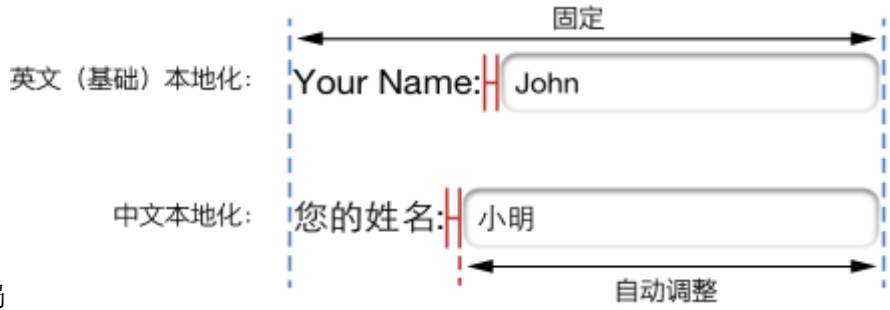


图 1-2 常见本地化模式中的自动布局

在本教程的余下部分，您将了解一些有关 iOS 的自动布局系统，以及如何使用 Xcode 的“Auto-Layout”工具，来指定用于应用程序国际化的布局规则的信息。之后，您将修改该用户界面，以拥有标签-文本栏设计，并对其进行配置，以拥有如图 1-2 所示的行为。
若要了解有关“Auto Layout”的更多信息，请参阅 *Cocoa Auto Layout Guide*（Cocoa 自动布局指南）。

Xcode 中的“Auto Layout”演示

iOS 的自动布局系统，可让您定义应用程序视图布局的规则，以便当一个视图更改其尺寸或位置后，相邻的视图会适当地调整自己的尺寸和位置。“Auto Layout”是对设备摆放方向的更改作出响应的关键功能。它也是将应用程序国际化的关键功能，特别是使用“Base Internationalization”的时候。

使用“Auto Layout”时，您使用**布局约束**来描述您想应用程序如何为用户界面进行布局。在**您的首个 iOS 应用程序**中配置 HelloWorld 用户界面时，您接受了“Auto Layout”系统（符合 Aqua 指南）施加的自动约束，例如使用建议的对象间距，并使对象的前置或后置边缘固定到其父视图。串联图场景中，视图周围的蓝色布局细线表示自动约束。现在，您要检查这些约束。

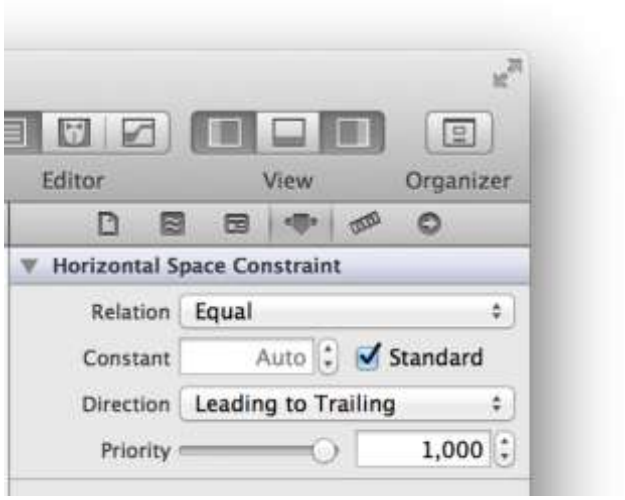
▶
检查布局约束

1. 在项目导航器中，选择 `MainStoryboard.storyboard` (Base Internationalization)。
2. 在 Hello World 视图控制器的大纲视图中，点按每个“Constraints”项旁边的展示三角形，以显示其内容。

   大纲视图应该是这样的：

3. 选择一种约束（例如“Horizontal Space - Label - View”），然后打开“Attributes”检查器。

“Attributes”检查器显示有关该约束的如下信息：

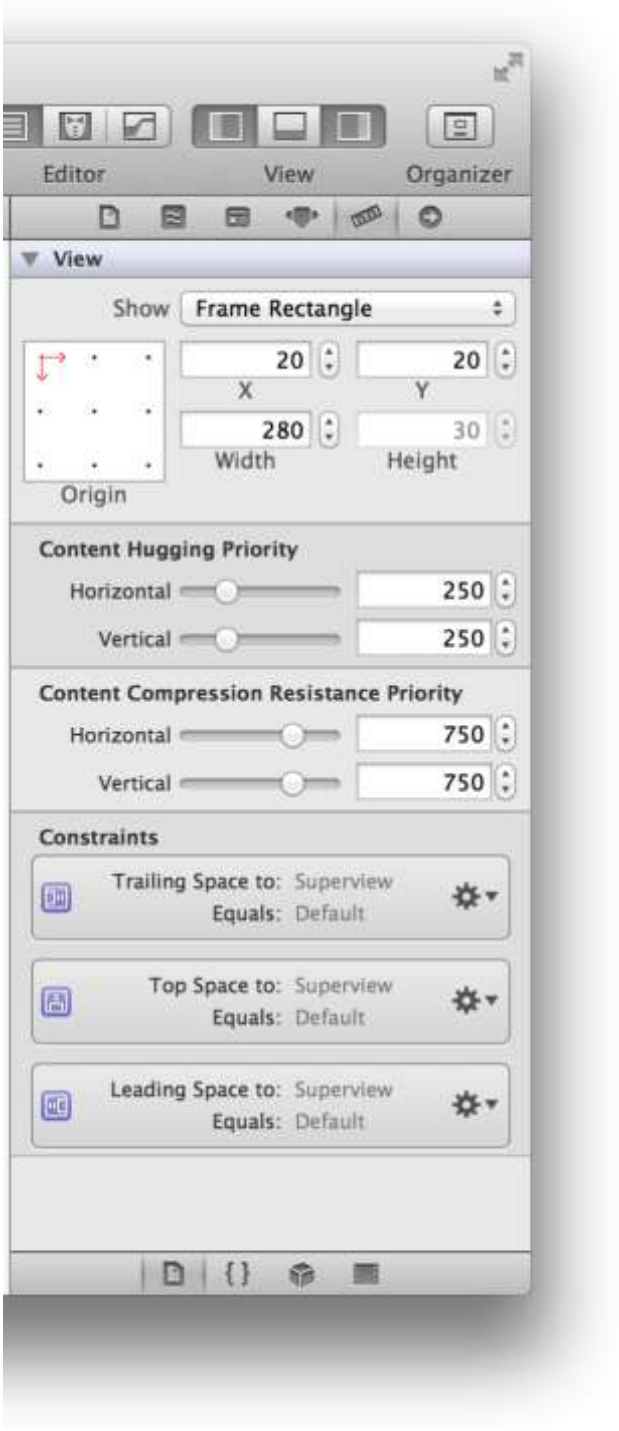


检查器中的值与该标签的父视图有关。选中“Standard”，意味着约束是自动的，换句话说，“Auto Layout”系统将根据 Aqua 指南布局对象。

Xcode 也可让您查看与下面步骤中所描述的特定视图相关联的所有约束。

4. 选择该文本栏，然后选取“Size”检查器。

   除了显示约束外，“Size”检查器还会显示视图的“Content Hugging Priority”值和“Content Compression Resistance Priority”值。



您可以覆盖自动约束的设置，从而创建用户约束，但是没必要那么做。正如先前所述，HelloWorld 的简单用户界面适合本地化。该文本栏（及其占位符文本）和标签可在屏幕上扩展，从而可以容纳几乎任何长度的字符串。而且，

“Hello”按钮会调整自己的大小，以适合每种本地化的标题，并在其父视图中保持居中。

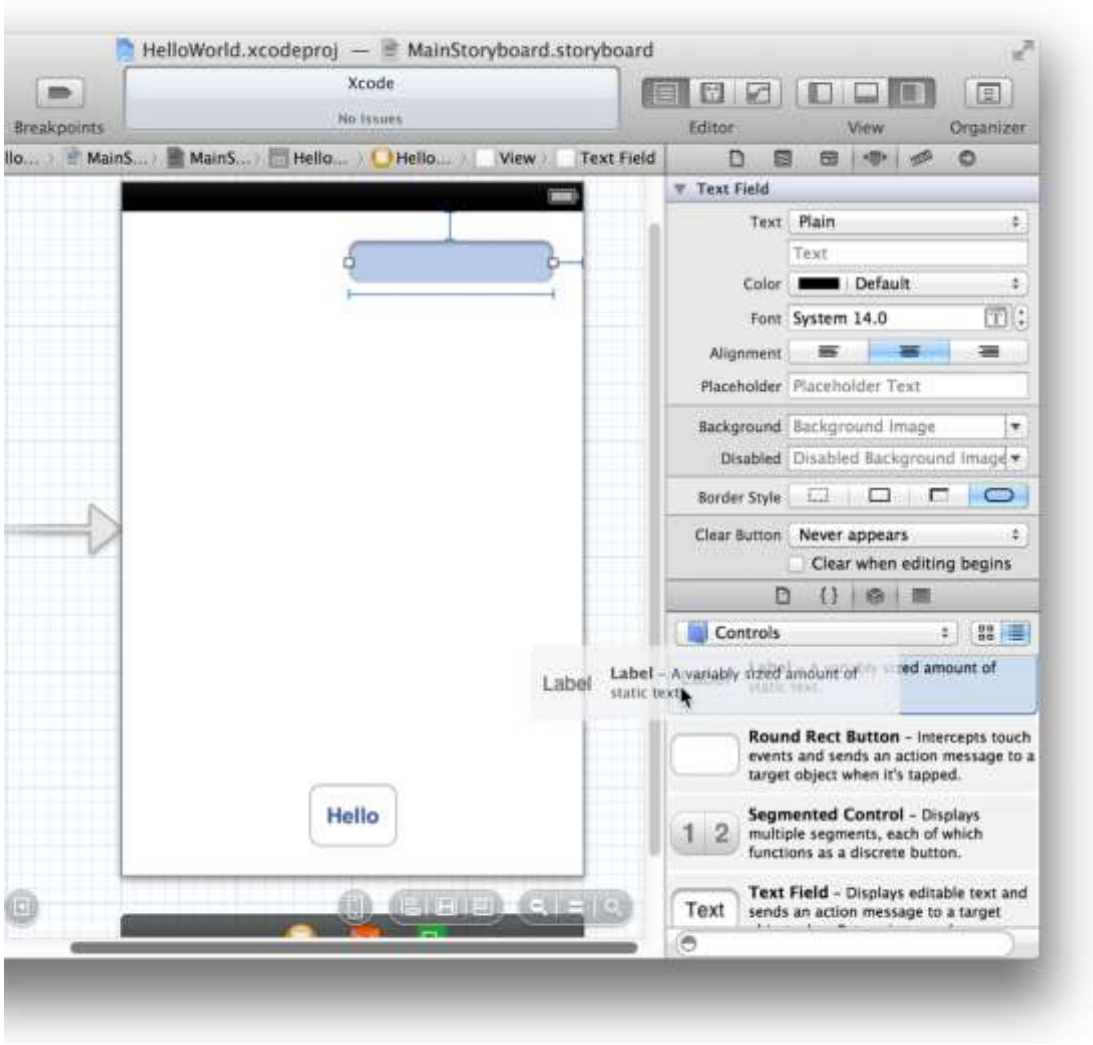
修改用户界面以进行国际化

现在您将修改 HelloWorld 的用户界面，以将标签和文本栏配对，这是应用程序中一种更为常见的用户界面设计。

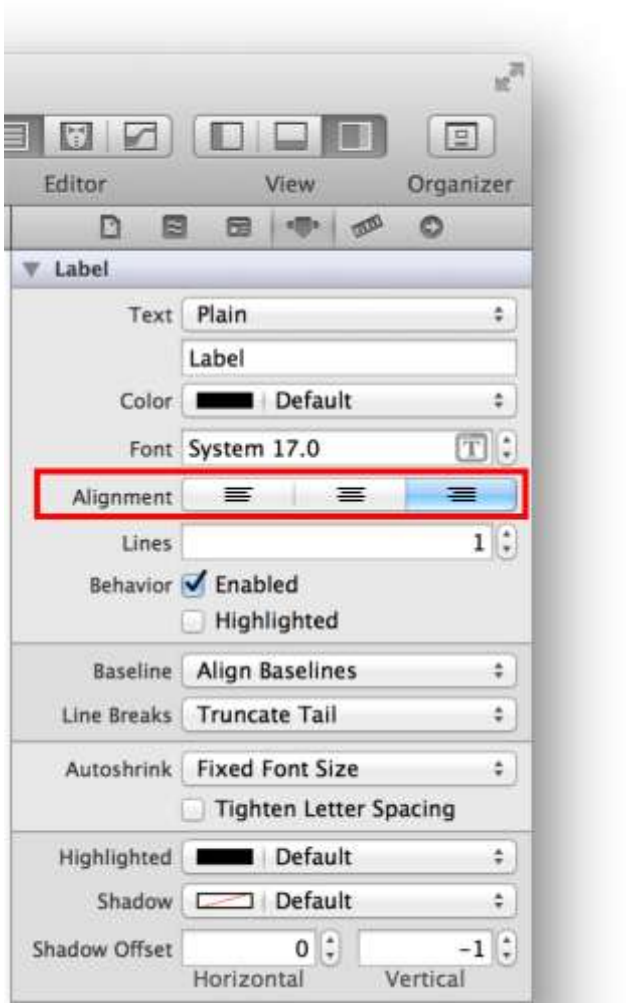
▶

更改 HelloWorld 应用程序的用户界面和布局约束

1. 选择该文本栏，在“Attributes”检查器中删除“Placeholder”栏中的 `Your Name`。
2. 调整该文本栏的宽度，以使其占据其父视图的右半部分。
3. 在“Utility”区域中打开对象库，并将一个标签对象拖到串联图场景上。



4. 选择该标签，然后点按“Attributes”检查器中的“Align Right”按钮。

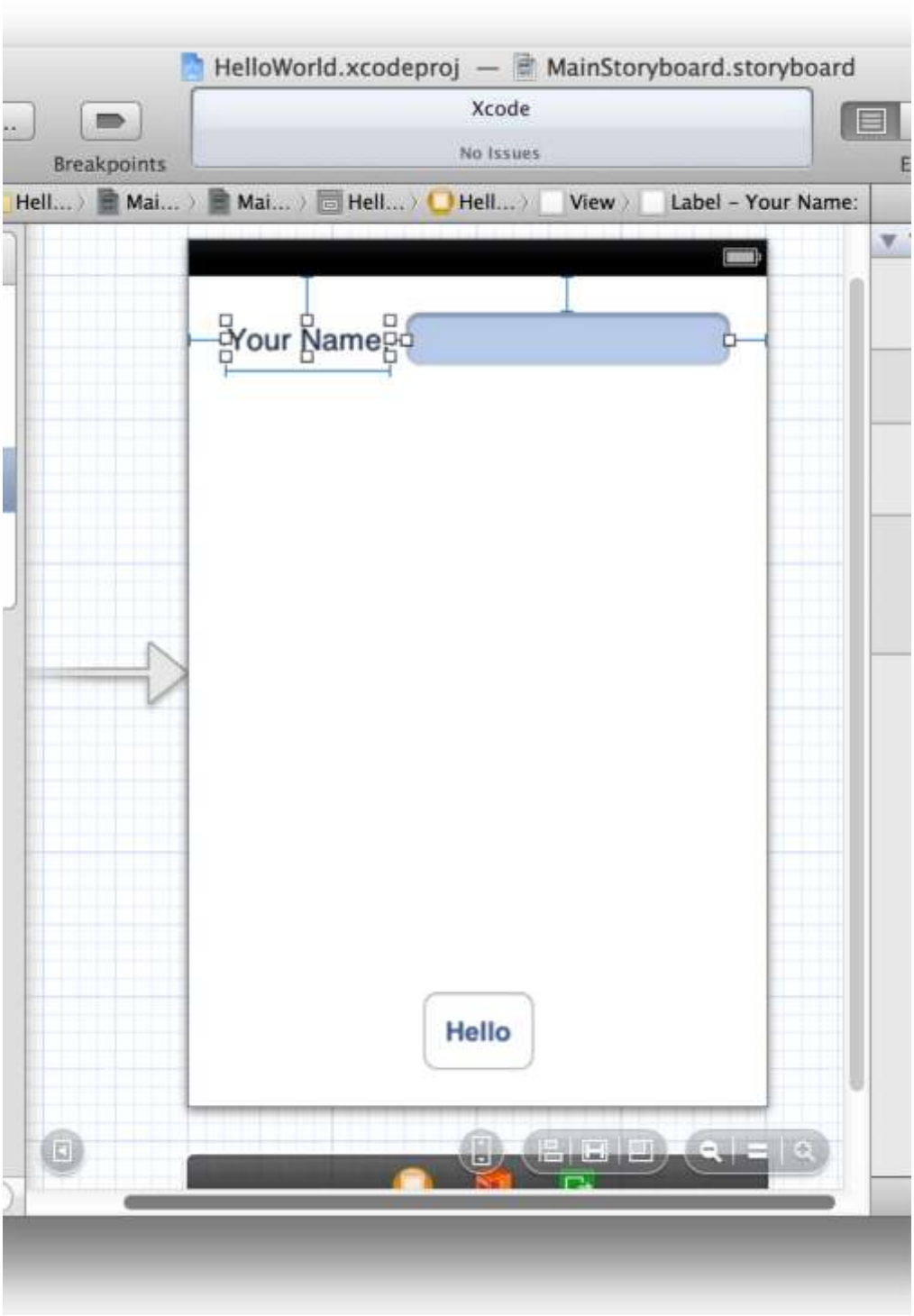


5. 选择该文本栏，然后点按“Align Left”按钮。

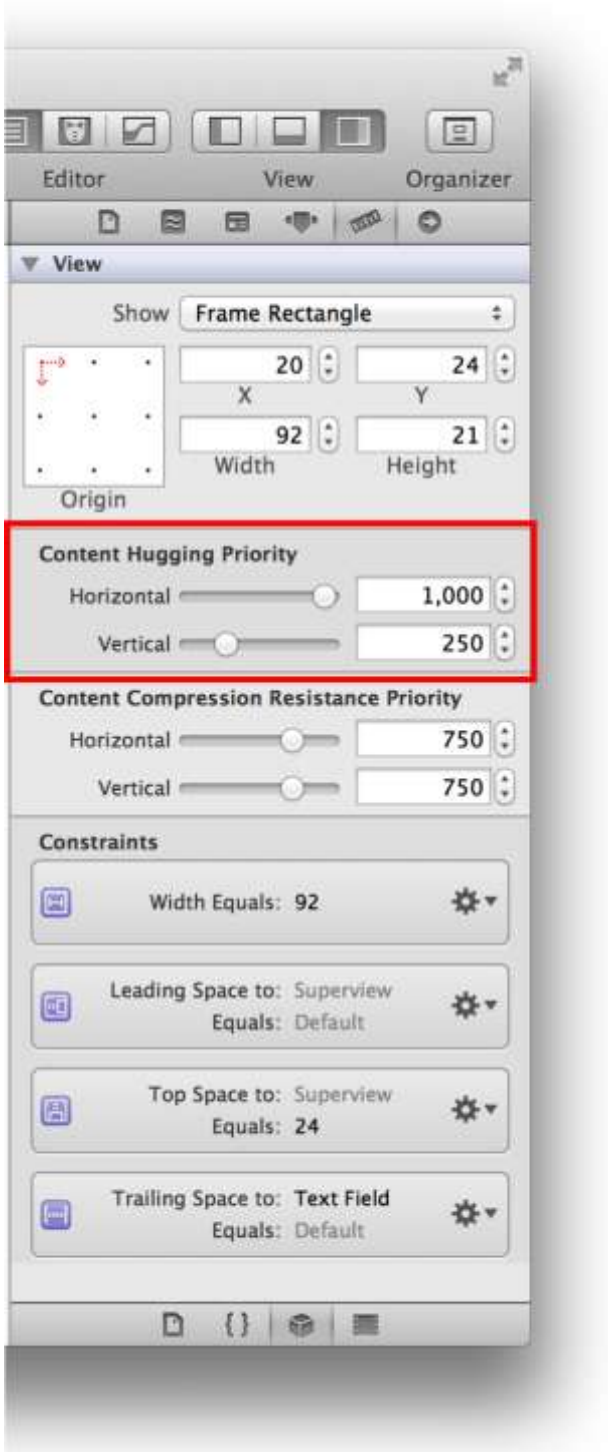
6. 连按该标签的默认文本 (`Label`)，以将其选中，然后使用 `Your Name:` 将其替换。

7. 调整标签和文本栏的位置和大小，以便该标签适合其文本，并且使自动约束（蓝色细线）能够生效。

   两个对象都选中后，现在用户界面应该是这样的：



8. 选择新标签对象，在“Size”检查器中，将“Content Hugging Priority”的“Horizontal”值设为 1000。



“Content Hugging Priority”告诉“Auto Layout”系统，相对其他视图来说，视图应该按其内容调整多少尺寸；就该标签而言，值 1000 实际上代表该标签应该总是调整其大小，以包含其字符串。因为该文本栏的“Content Hugging Priority”仍使用 250 的默认值，它是一个当标签宽度变更后也随之更改其宽度的对象。

现在已将标签-文本栏对国际化了。如果键入 `A Very Long Name` 作为标签文本，然后按下 Return 键，您会看到文本栏缩短自己的宽度，以容纳新标签值。您也可以运行这个程序的英文本地化版本和简体中文本地化版本，并观察各自行为的变化。

当串联图更改时更新串联图字符串文件

在刚完成的“Auto Layout”练习中，您将新文本显示对象（“Your Name”标签）添加到了 HelloWorld 用户界面。因此，现在中文 MainStoryboard.strings 文件与“Base Internationalization”(MainStoryboard.storyboard) 不同步。您可以使用命令行实用工具 ibtool，从“Base Internationalization”生成新字符串文件，然后将该新字符串文件条目添加到现有的 MainStoryboard.strings 文件

▶

将新用户界面对象与现有的中文 MainStoryboard.strings 文件合并

1. 启动“终端”应用程序。

   此应用程序位于“/应用程序/实用工具”中。

2. 在“终端”的“Shell”中，连接到项目文件夹的 Base.lproj 目录。

   例如：

```
cd /Users/UserName/Projects/HelloWorld/HelloWorld/Base.lproj
```

3. 在提示符后输入以下命令：

```
ibtool MainStoryboard.storyboard --generate-strings-file NewStuff.strings
```

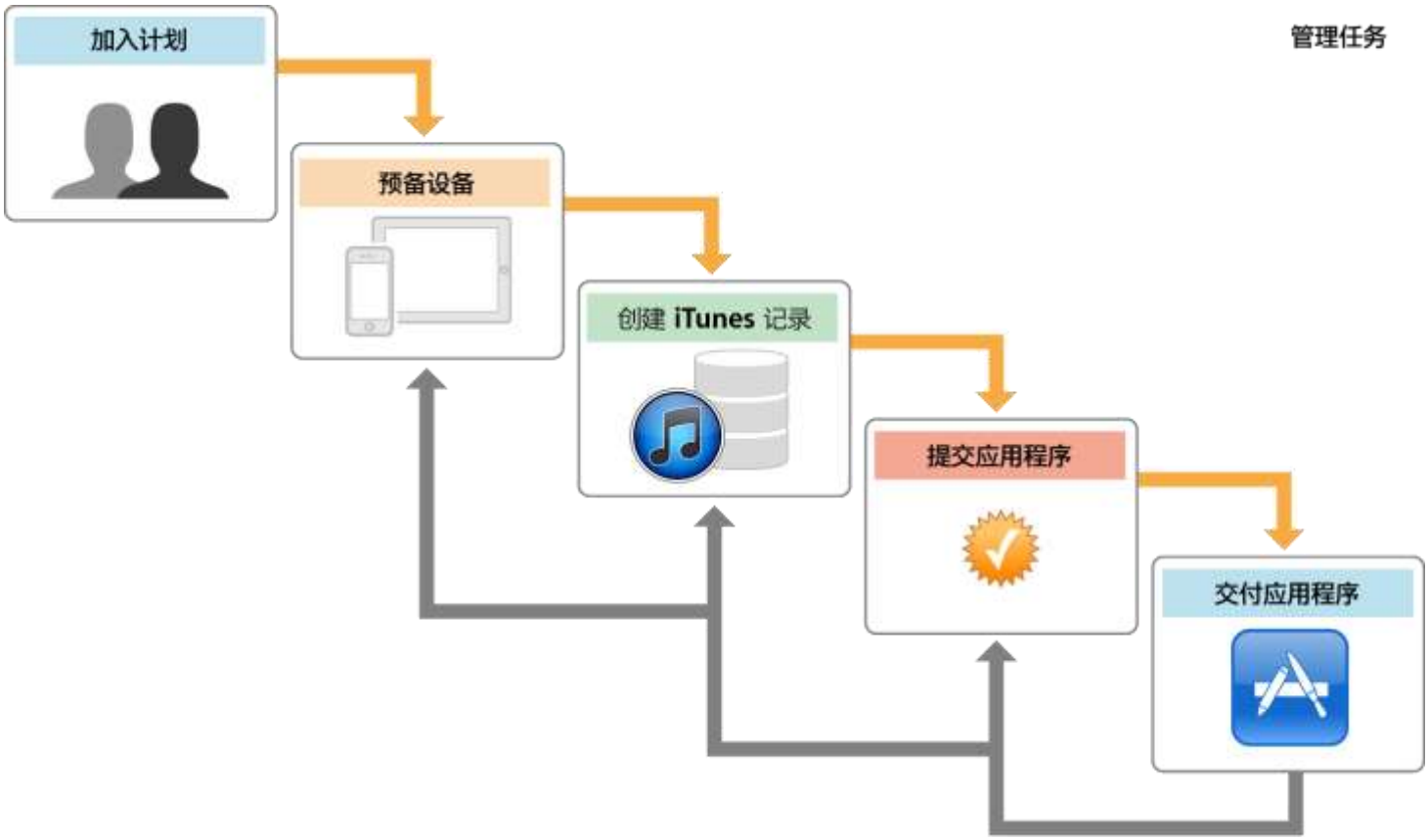
4. 可以给输出文件命名随您选取的任何名称（本示例是用 NewStuff.strings）。

5. 在 Xcode 中，打开生成的输出文件，并将新字符串文件条目（即“Your Name：”标签的注释和键－值对）拷贝到中文 MainStoryboard.strings 文件。

6. 翻译新字符串值。

准备提交到 App Store

您的大部分时间都花在了编程任务上，但是要为 App Store 开发应用程序，您还需要在应用程序的整个生命周期中，使用 Xcode 和其他工具来执行一些管理任务。App Store 是一个受监管的商店，限制哪些应用程序可以销售。Apple 这么做是为了尽可能地为用户提供最佳体验。例如，在 App Store 上出售的应用程序不得崩溃或出现其他主要错误。



Apple 为您提供了所需的工具，来进行开发和测试，以及将应用程序提交到 App Store。要在设备上运行应用程序，设备需要为开发和稍后的测试做好预备工作。还需要提供应用程序的相关信息，以供 App Store 显示给客户，并且还需要上传屏幕快照。然后将应用程序提交给 Apple 审批。应用程序审批通过后，您设定应用程序在 App Store 上架销售的日期。最后，使用 Apple 的工具来监测应用程序的销售、客户评论和崩溃报告。然后再次重复整个流程，来提交应用程序的更新。

如果使用某些技术（例如 iCloud 储存或应用程序内购买），则需要执行额外的配置和管理任务。您还要执行管理开发者团队的任务。

加入 iOS Developer Program

要为 App Store 开发应用程序，首先需要加入 iOS Developer Program。加入该计划之后，您可以访问所需的资源和工具，来管理您的帐户，以及在设备上测试应用程序。

您将成为与 Apple 联络的主要人员，负责签订法律条款、创造资产并推广您的应用程序。您将要回答是个人开发者，还是公司开发者。如果是公司开发者，您可以将其他人添加到您的团队，并授予权限给他们中的某些人来管理帐户。在开发期间，需要在设备上运行应用程序的个别人士，要先加入您的团队。

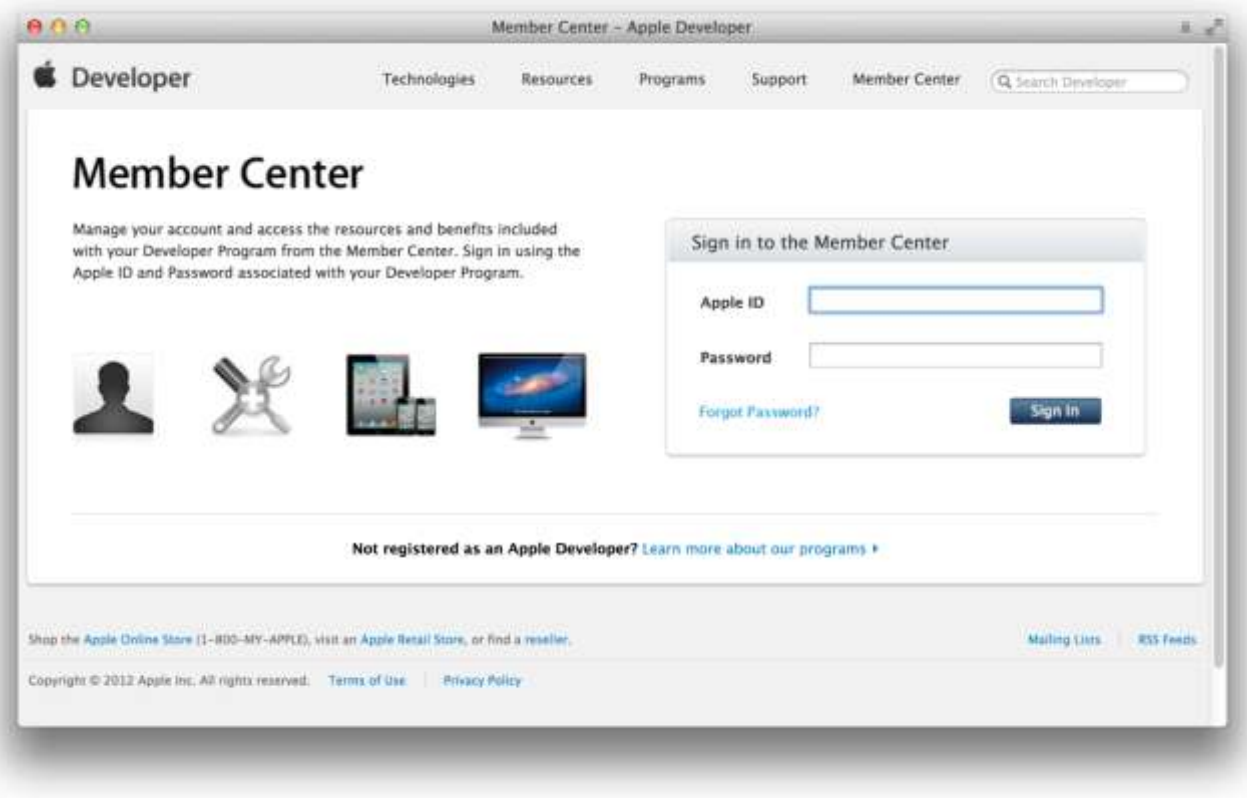
您将使用以下 iOS Developer Program 网上工具来管理您的帐户：

- Member Center 是主要工具，用来管理开发者计划帐户、邀请团队成员、购买技术支持和申请兼容实验室。Member Center 也是通向其他资源和工具的大门。
- iOS Provisioning Portal 是网上工具，用来注册应用程序 ID、注册设备、制作签名证书和创建预置描述

文件 (provisioning profile)。这些步骤能够确保安全性，同时能避免应用程序被贸然发布。

- iTunes Connect 是营销和商务工具，用来检查合同状态、设置税务及银行信息、获取销售及财务报告，以及管理应用程序元数据。

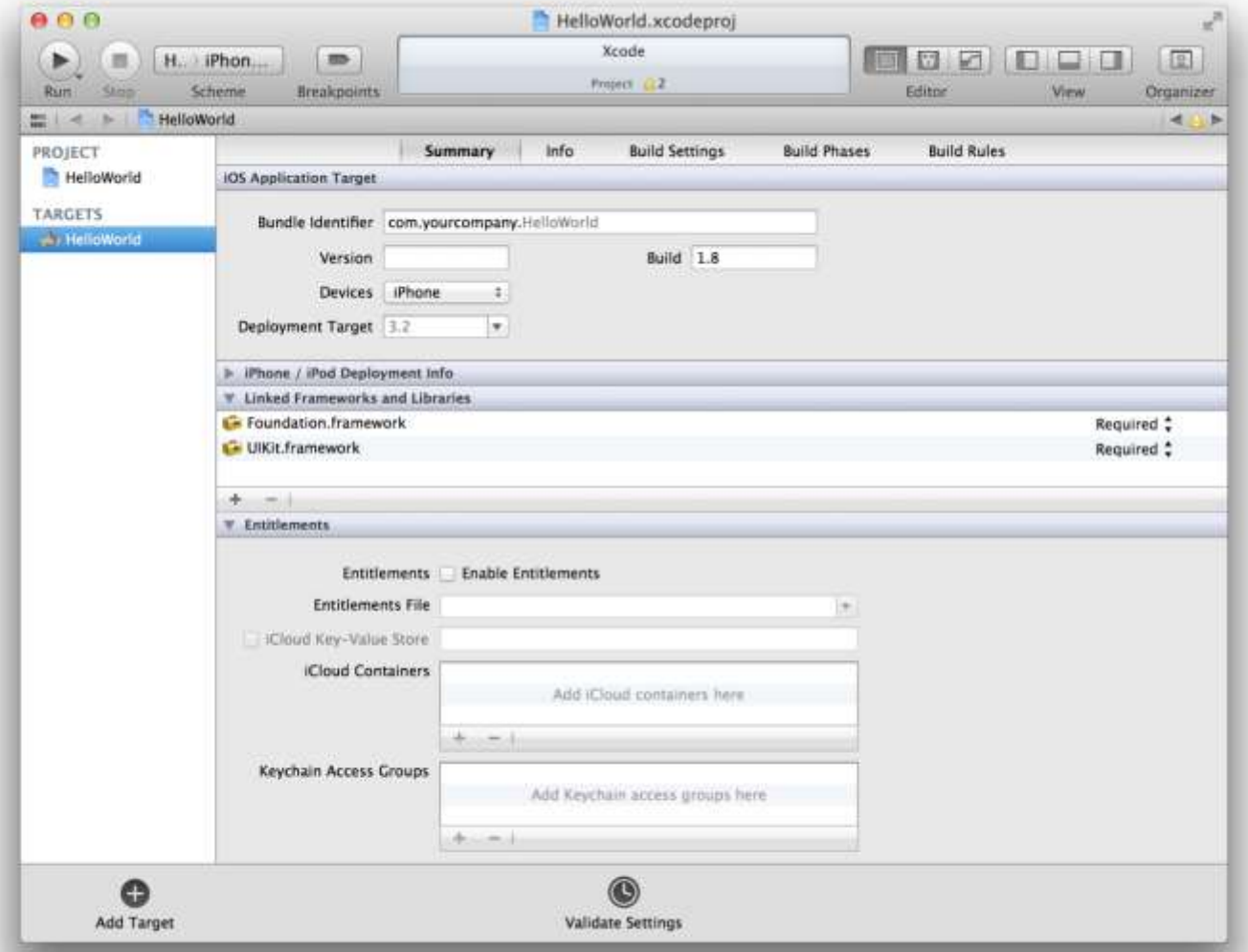
您可以使用 Xcode 执行某些 iOS Provisioning Portal 管理任务，再根据需要通过访问 Member Center 返回到这些网上工具，网址为 `http://developer.apple.com/membercenter`。



为 App Store 创建项目并进行配置

从模板创建 Xcode 项目时，某些 App Store 配置已经完成。Xcode 会提示您输入产品名称和公司标识符。捆绑包 ID 就来自这两项属性。例如，在 HelloWorld 项目中，产品名称是 HelloWorld，公司标识符是 `edu.self`。因此，默认的捆绑包 ID 为 `edu.self.HelloWorld`。Xcode 也为其他值使用合理的默认值。您应该认真考虑，使用哪个模板来创建应用程序，使用什么设置来配置项目；从正确的模版开始，有助于加速开发过程。

如果想要稍后更改这些设置，或使用 iCloud 储存，您可在 Xcode 的目标“Summary”面板中找到大部分设置，包括启用权利。例如要通过验证测试，您需要设定应用程序图标和启动画面，它们出现在“Summary”面板上的“iPhone/iPod Deployment Info”下面。这些图像用来在 App Store 中代表您的应用程序。



为开发预备好设备

开发期间，要在设备上运行应用程序，该设备必须连接到 Mac、已启动开发功能，并经过 Apple 识别。只需提供应用程序、您本人和设备的一些相关信息，就可以完成以上准备工作。您创建一种名为 **development certificate** 的签名证书来标识您自己。所有这些信息都会纳入开发预置描述文件，该文件最终要安装到设备上并允许应用程序开启。

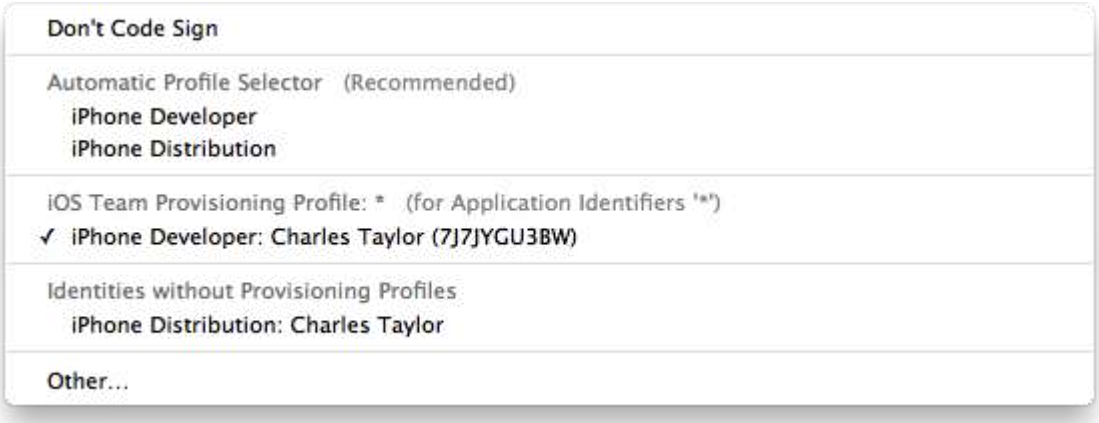
通过使用 Xcode 为您创建的默认应用程序 ID 和 iOS 团队预置描述文件 (iOS Team Provisioning Profile)，您可以使用 Xcode 中的“Devices”管理器来预备设备，以进行开发。（但是，如果使用 iCloud 储存、推送通知、应用程序内购买或 Game Center，则需要创建一个专用预置描述文件。）

第一次在“Devices”管理器中刷新预置描述文件时，Xcode 会创建您的签名证书。Xcode 代表您创建开发和分发证书 (development and distribution certificates)。（分发证书在稍后测试和提交应用程序到 App Store 时需要。）

iOS 团队预置描述文件可让您立即开始在设备上运行应用程序。首次将设备添加到您的帐户时，Xcode 会使用默认的应用程序 ID、您设备的 ID 和您的开发证书来创建 iOS 团队预置描述文件。只需要将设备与 Mac 连接，然

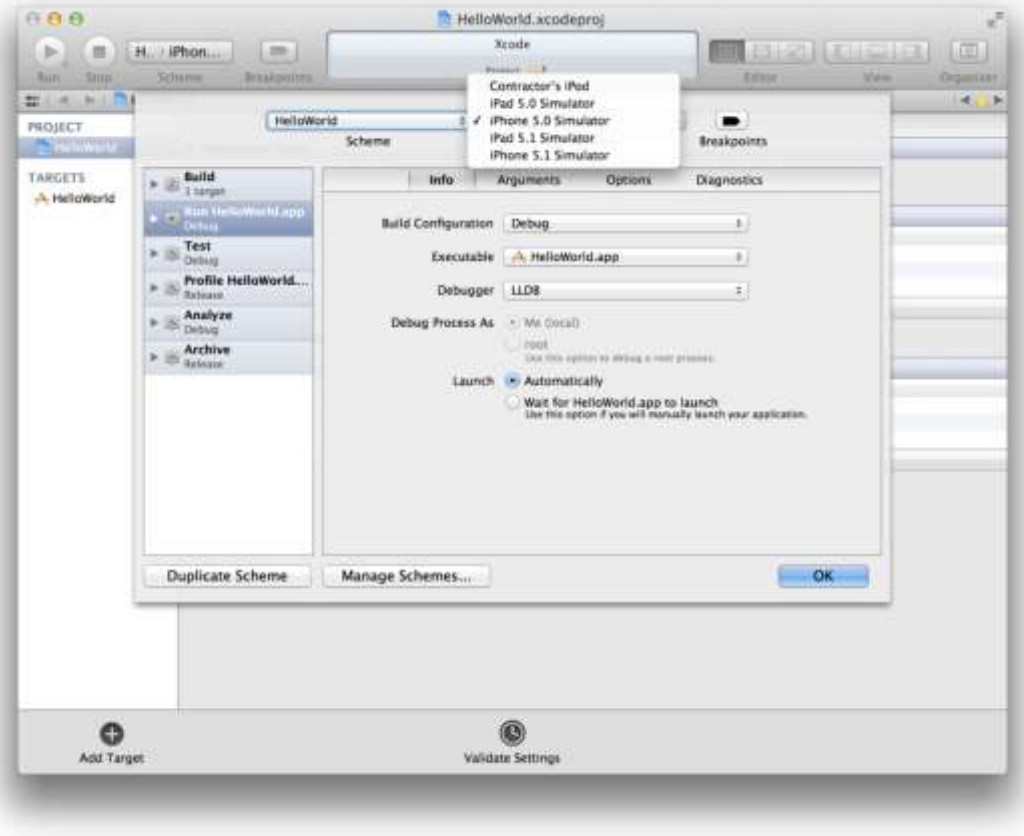
后点按“Use for Development”按钮，将设备添加到 iOS 团队预置描述文件。然后，Xcode 自动将此描述文件安装在您的 Mac 连接着的设备上。预备新设备以用于开发时，Xcode 也更新此预置描述文件。

生成应用程序时，您要进行代码签署，采用的签名证书就包含在要使用的预置描述文件中。在 Xcode 项目编辑器中，使用“Code Signing Identity”生成设置弹出式菜单，将“Code Signing Identity”设定为 iOS 团队预置描述文件中包含的开发者证书。



将设备预备好用于开发后，可以告诉 Xcode 在设备上启动应用程序。方法是在生成应用程序前，在“Scheme”弹出

式菜单中，更改运行目的位置的设置。将附带有效预置描述文件的设备连接到 Mac 时，设备名称和其运行的 iOS 版本，会作为选项出现在目的“Scheme”弹出式菜单中。选取“Product”>“Edit Scheme”以打开方案编辑器。

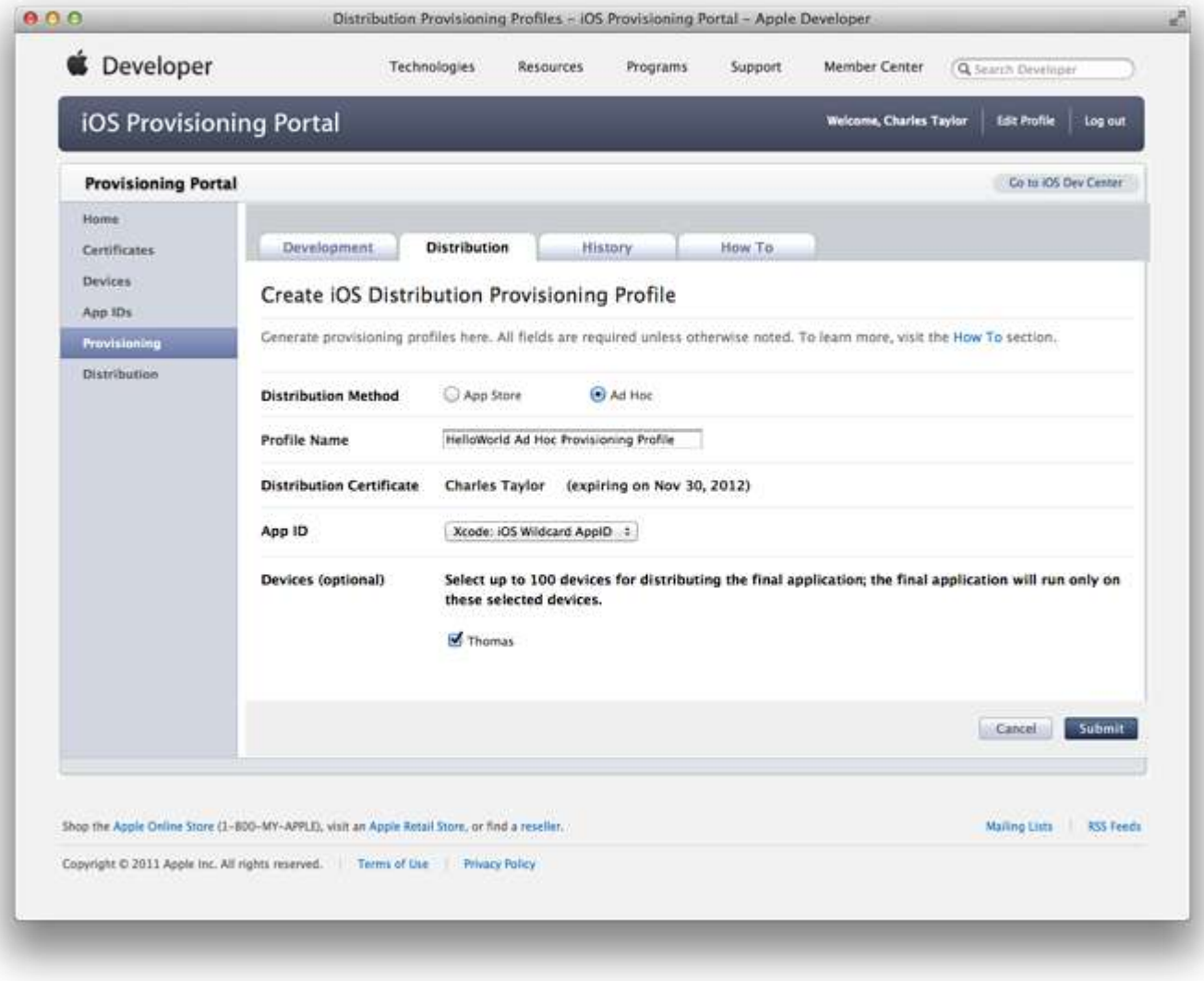


在多个设备和多个 iOS 版本上测试应用程序

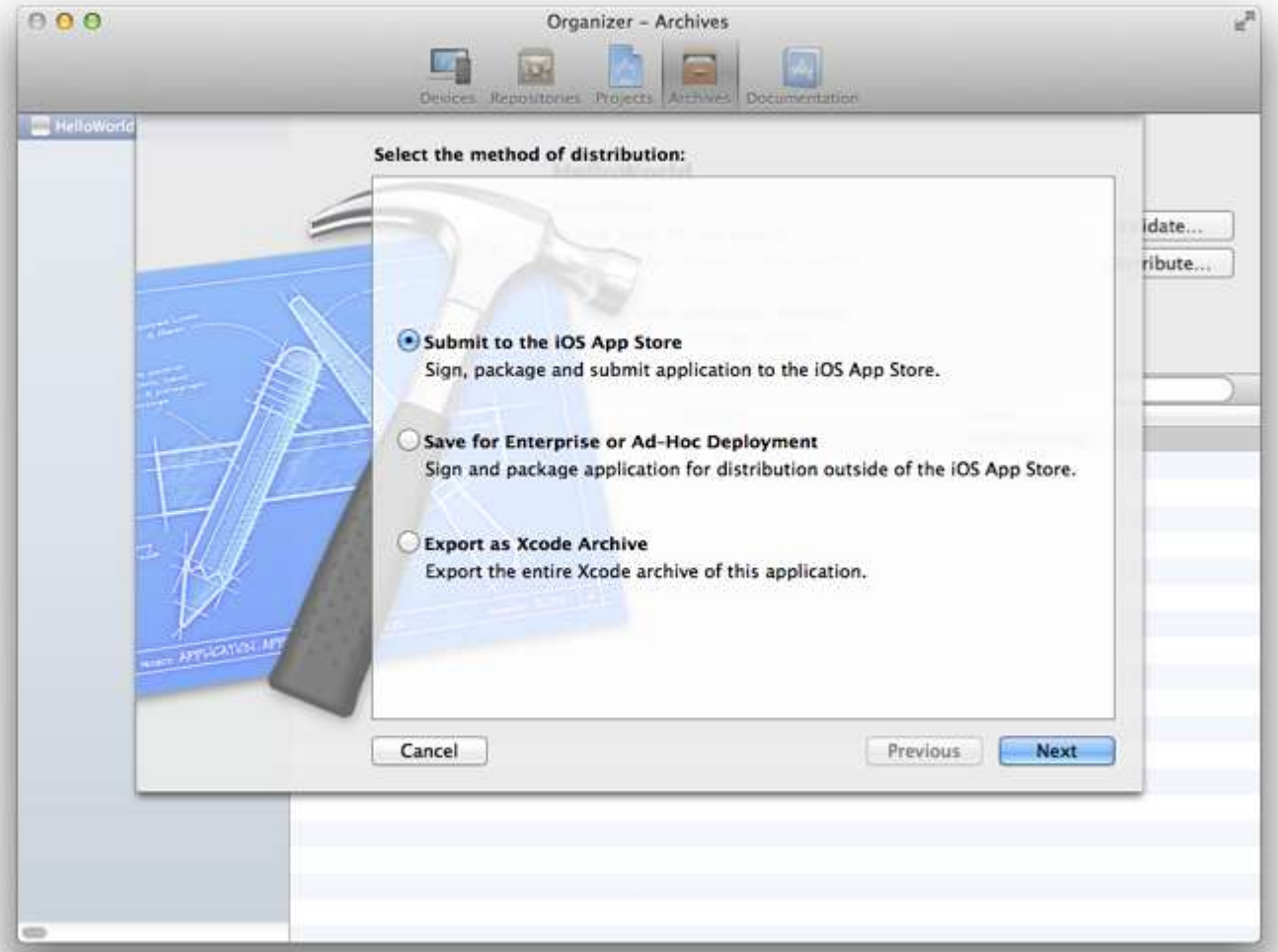
您应该制定计划，在各种设备和 iOS 版本上严格测试应用程序。仅使用模拟器并仅在预备用于开发的设备上测试应用程序，是不够的。模拟器不能运行在设备上运行的所有线程，使用 Xcode 在设备上开启应用程序，会停用某些监察定时器 (watchdog timer)。至少，您应该在所有能找到的设备上测试应用程序。最理想的做法是，在打算支持的所有设备和 iOS 版本上测试应用程序。

做法是创建一个名为 **adhoc provisioning profile**（临时预置描述文件）的特殊分发预置描述文件，并将其和应用程序一起发送给测试员。临时预置描述文件不需要将测试员添加到您的团队，不需要创建签名证书或使用 Xcode 运行应用程序。应用程序测试员仅需在他们的设备上安装该应用程序和临时预置描述文件，就可启动应用程序。然后，可以从这些测试员收集和分析崩溃报告或日志，从而解决问题。

首先，从测试员那里收集所有的设备 ID，并将它们添加到 iOS Provisioning Portal 中。测试员可使用 iTunes 来获得他们设备的 ID。使用 iOS Provisioning Portal，您创建包含应用程序 ID 和这些设备 ID 的临时预置描述文件。



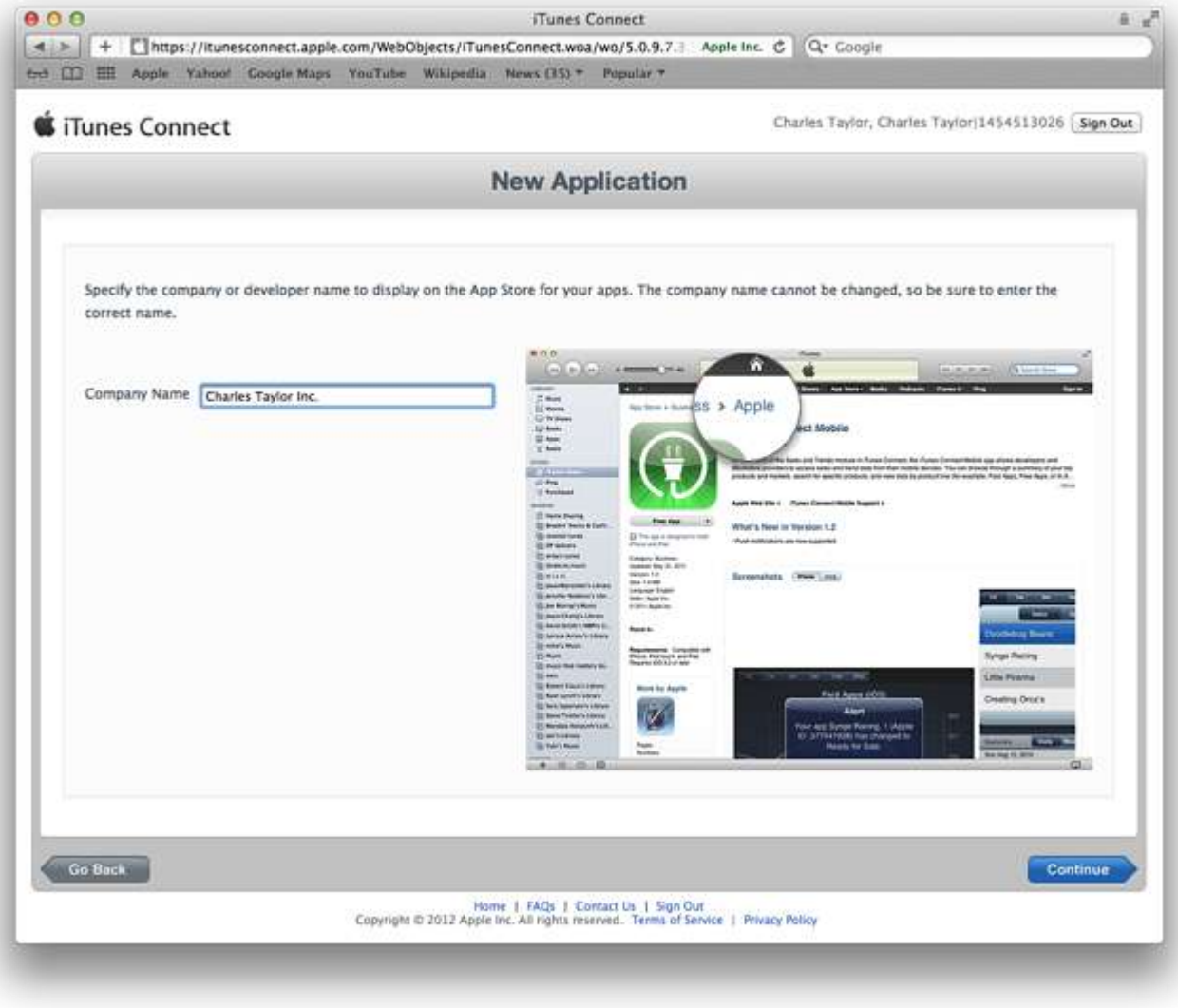
应用程序可用于测试时，使用 Xcode 来创建归档和生成 iOS App Store 软件包（文件扩展名为 .ipa 的文件）。在“Archives”管理器中，选择归档，点按“Distribute”按钮，然后点按“Save for Enterprise or Ad-Hoc Deployment”选项来创建软件包。创建软件包时，您使用开发证书给归档签名。然后从 iOS Provisioning Portal 下载临时预置描述文件，并将其和 IPA 文件一起发送给测试员。



测试员使用 iTunes 在他们的设备上安装预置描述文件和应用程序。应用程序在设备上崩溃时，iOS 会创建该事件的记录。下次测试员将设备连接到 iTunes 时，iTunes 会将这些记录（称为“**崩溃日志**”）下载到测试员的 Mac 上。测试员应该将这些崩溃日志发送给您。

在 iTunes Connect 中配置应用程序数据

应用程序在 App Store 销售时，该商店会显示应用程序的很多信息，包括名称、描述、图标、屏幕快照和您公司的联系信息。要提供这些信息，请登录到 iTunes Connect，为应用程序创建记录并填写一些表单。iTunes Connect 中的记录包括捆绑包 ID 栏；在此栏中输入的值必须完全匹配应用程序的捆绑包 ID。应用程序名称和版本也需要与 Xcode 项目配置相符。插图需要上传到 App Store 以通过验证测试，App Store 要用它们将应用程序展示给客户。应用程序记录状态至少应该是“Waiting for Upload”，才可将应用程序提交到 App Store。

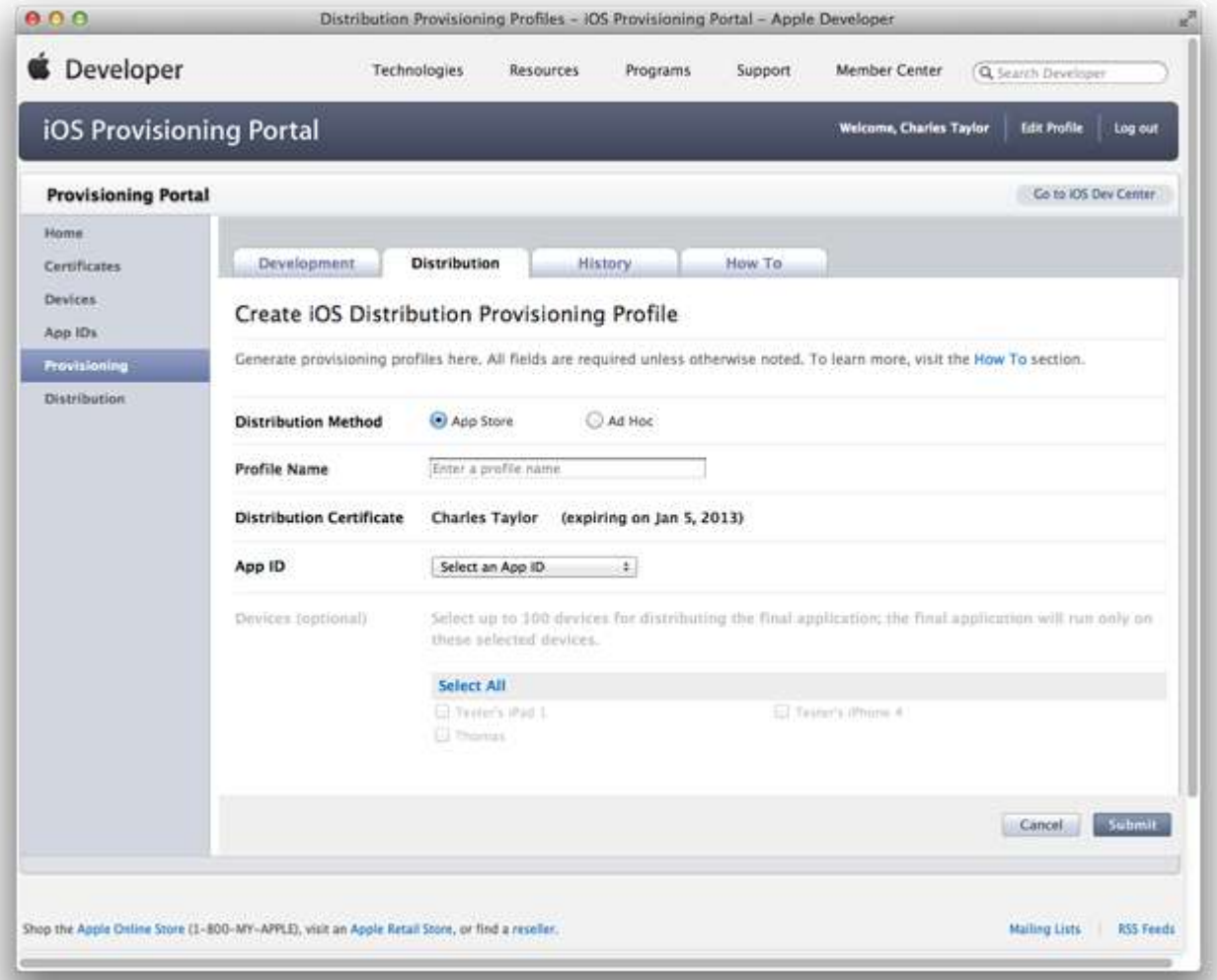


通常在开发过程的较后阶段，才创建 iTunes Connect 应用程序记录，因为从创建记录到提交应用程序之间有时间限制。但是，一些 Apple 技术（包括 Game Center 和应用程序内购买）要求早一点创建 iTunes Connect 记录。例如，对应用程序内购买而言，需要创建应用程序记录以便添加您想要出售项目的详细信息。此内容需要在开发过程完成之前创建，以便使用它来测试实现应用程序内购买所添加的代码。

将应用程序分发到 App Store

将应用程序提交到 App Store 需要很多步骤，还会用到几个工具。首先登录到 iTunes Connect，将应用程序记录的状态更改为“Waiting for Upload”或靠后的状态。然后使用 iOS Provisioning Portal，创建分发证书并分发预置描述文件。使用 Xcode 创建归档、验证归档，并将其提交到 App Store。应用程序通过审批后，使用 iTunes Connect 设定让客户购买该应用程序的日期。

当应用程序准备发布时，您需要创建分发预置描述文件 (distribution provisioning profile)，选择 App Store 作为分发方法。创建这种类型的预置描述文件时，只需选择一个应用程序 ID，而不选择任何签名证书或设备 ID。



使用 Xcode 中的“Archives”管理器来验证和提交应用程序。首先创建归档，然后使用分发证书为其签名。然后验证归档，完成对归档中的应用程序以及您在 iTunes Connect 记录中提供的信息的自动化检查。如果在验证过程中发现问题，您需要修正这些问题才能继续。

在提交应用程序前，您应该阅读 App Store Review Guidelines 以避免出现问题。点按“Distribute”按钮并选中“Submit to the iOS App Store”选项时，Xcode 将归档传输到 Apple——Apple 检查归档以测定它是否符合应用程序指南。如果应用程序遭拒，请修正应用程序审批过程中提出的问题，然后重新提交应用程序。

使用 iTunes Connect 设定应用程序即将发布到 App Store 的日期。例如，您可以选取在应用程序通过审批后，立即将应用程序发布到 App Store，也可以设定审批日期之后的某一天。使用晚一些的发布日期，可让您在应用程序首发日前后安排其他营销活动。

回应用户问题

不能将应用程序提交到 App Store 后就置之不理。您应该在应用程序的整个生命周期中管理应用程序记录，并维护应用程序。应用程序一旦发布到 App Store，您就需要监控其状态，回应用户的问题，并提交所需的更新。

您要关注用户对您的应用程序有什么样的感受。App Store 中的客户评级和评论，极大地影响着应用程序的成功。如果用户遇到问题，您需要迅速确定错误，然后通过审批流程提交应用程序的新版本。

iTunes Connect 提供的数据能帮助您判断应用程序有多成功，这些数据包括销售和财务报告、客户评论，以及用户提交给 Apple 的崩溃日志。崩溃日志至关重要，因为它们表示用户在应用程序中遇到的重大问题。您应该优先研究这些报告。

除了低内存崩溃日志外，所有崩溃日志都包含应用程序终止时每个线程的堆栈跟踪。要查看崩溃日志，您需要在 Xcode 管理器窗口中打开它。只要您的 Mac 上的归档与产生崩溃日志的应用程序版本相一致，Xcode 就自动将崩溃日志中的所有地址解析为应用程序中的实际类和函数。

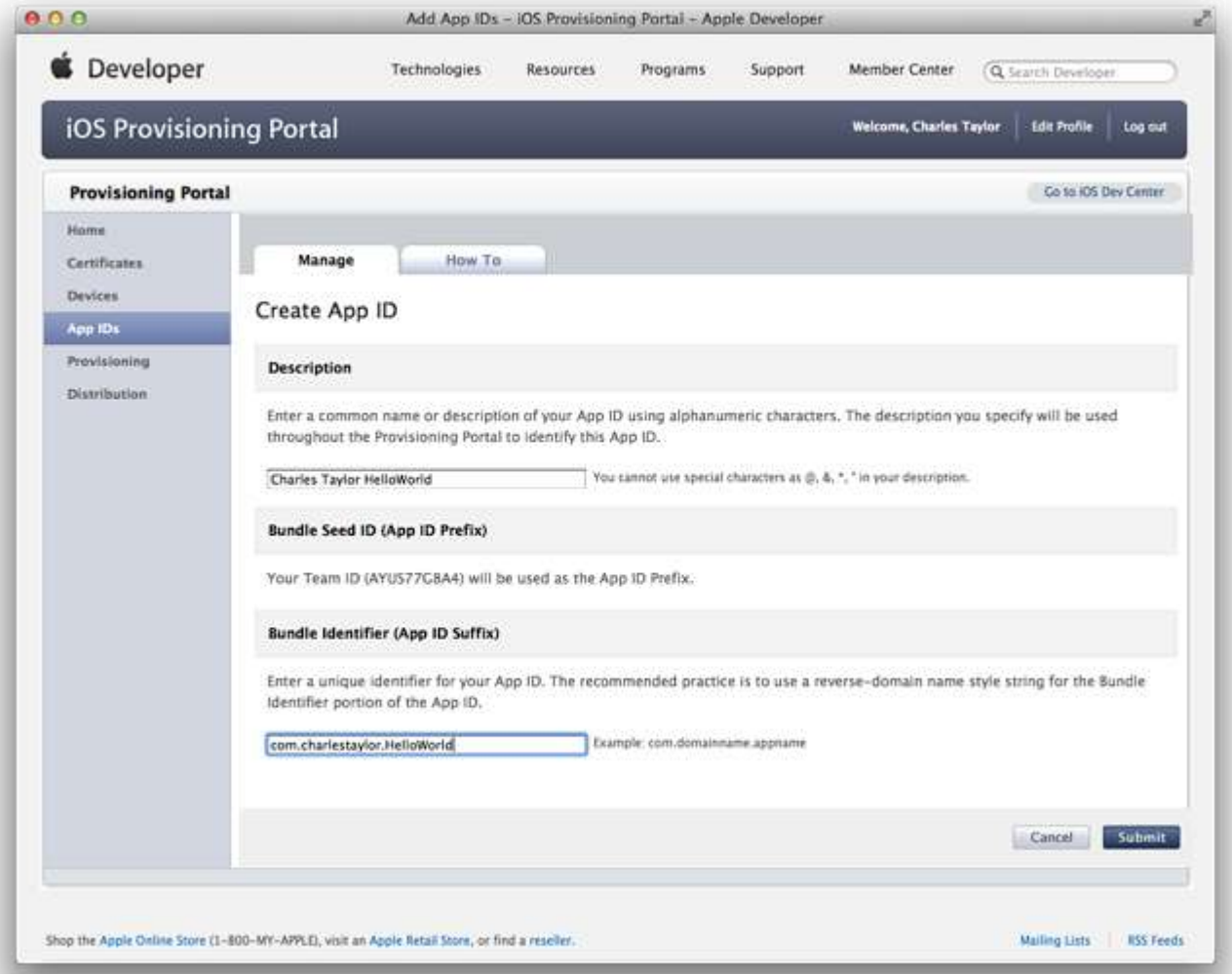
预备您的设备用于 iCloud 储存或应用程序内购买

如果使用某些技术，您需要创建专用预置描述文件（该文件使用明确的应用程序 ID），并相应配置应用程序。Apple 使用此应用程序 ID，在整个 iOS、App Store 和 Apple 的服务器中，作为使用了这些技术的应用程序的唯一识别。需要这些预置描述文件的技术有：

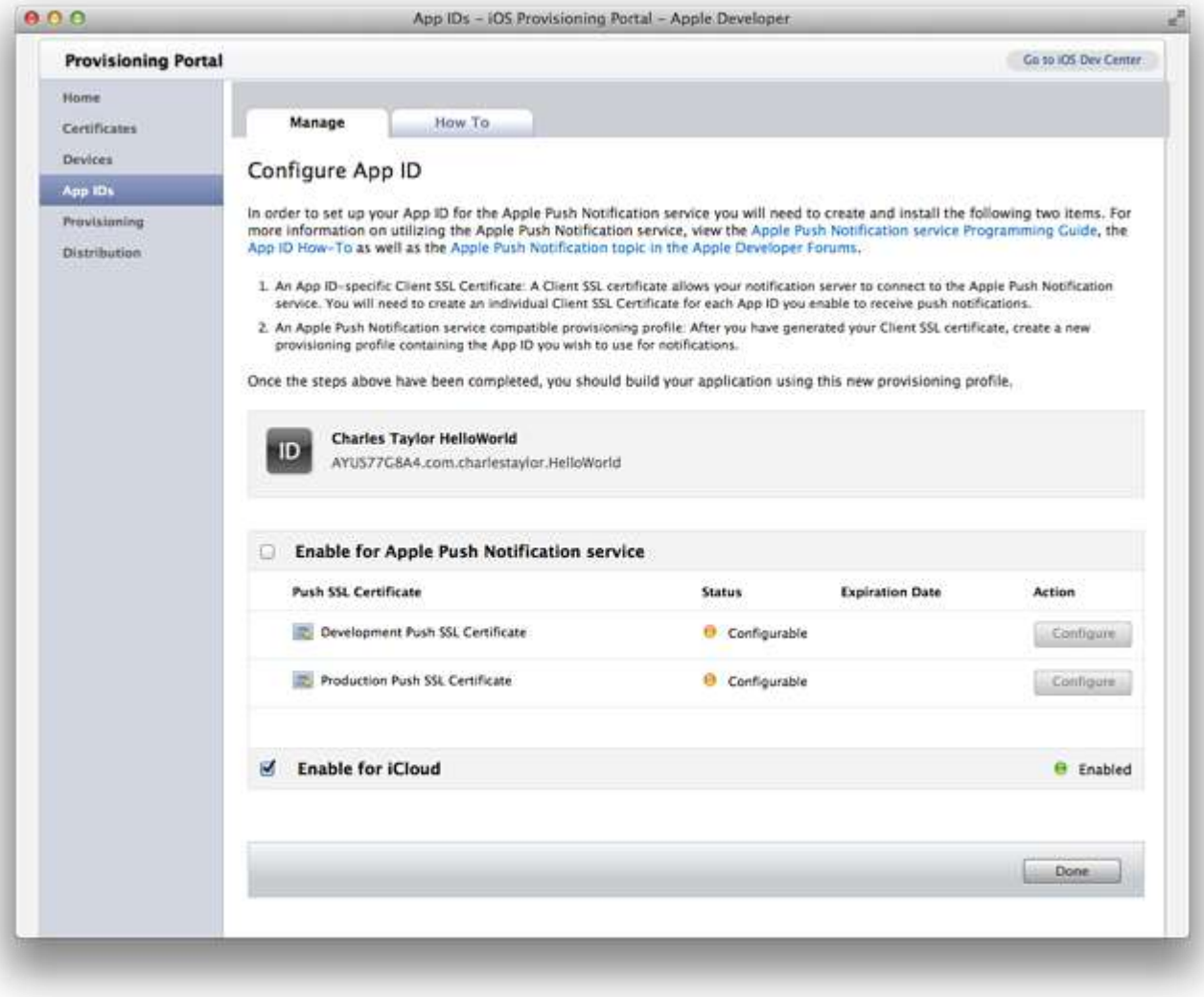
- **iCloud 储存**，允许您与不同 iOS 和 Mac OS X 设备上运行的应用程序的多个实例共享用户数据。
- **推送通知**，允许不在前台运行的应用程序，在有信息时通知用户。
- **应用程序内购买**，允许您连接至 App Store 并安全地处理用户的付款，即直接将商店嵌入应用程序内。
- **Game Center**，它是一项社交游戏服务，允许玩家分享他们正在玩的游戏的信息，并参与多人游戏比赛。

开发预置描述文件 (development provisioning profile) 包含一个签名证书列表、一个应用程序 ID 和一个设备 ID 列表。如果以前已经使用 iOS 团队预置描述文件来预置设备以用于开发，则签名证书和设备 ID 已经存在于您的帐户中。Xcode 提供的应用程序 ID 为匹配所有捆绑包 ID 的通配符 ID。您需要创建一个完全匹配应用程序捆绑包 ID 的应用程序 ID，并在新建的开发预置描述文件中，使用该应用程序 ID，而非通配符应用程序 ID。如果使用 iCloud 储存或推送通知，需要启用应用程序 ID 以使用这些技术。

使用 iOS Provisioning Portal，向 Apple 注册应用程序 ID 并创建开发预置描述文件。一个明确的应用程序 ID，与您的捆绑包 ID 完全相符。



创建明确的应用程序 ID 时，应用程序内购买和 Game Center 在默认情况下是启用的。如果想要启用推送通知或 iCloud 储存，请在“App IDs”页面上点按应用程序 ID 旁的“Configure”，然后选择合适的选项。需要先启用这些技术，才能在专用预置描述文件中使用该应用程序 ID。



创建开发预置描述文件时，请选择明确的应用程序 ID、您的签名证书和想要使用的设备 ID。预置描述文件的状态从“Pending”更改为“Active”时，请在 Xcode 中刷新预置描述文件，然后使用新的描述文件给应用程序签名。同样，使用明确的应用程序 ID 来创建用于测试的临时预置描述文件，以及用于提交的分发预置描述文件。

如果想要使用 iCloud 储存，请在 Xcode 中启用权利，并在目标“Summary”面板中的“Entitlements”下方配置 iCloud。

快速查找文稿

Apple 提供文稿以帮助您成功生成并部署应用程序，包括示例代码、常见问题解答、技术说明、视频，以及概念性文稿和参考文稿。要获得最新和最准确的信息，请直接从 Apple 处获取您需要的文稿资源。安装了 Xcode，就可以访问这些资源。如果您更喜欢在 iPad 上使用浏览器或查阅 PDF 文件，可以查看网上的 iOS Developer Library。不管哪种方式，您都需要熟悉所提供的各种导航和搜索技巧。

使用 Xcode 的“Documentation”管理器来查找文稿

Xcode 的“Documentation”管理器是一个功能完备的查看器，提供对开发者文稿的综合搜索和查阅。您可以使用

“Documentation”管理器指定想要查阅的文稿集。

文稿集是一组与关键 Apple 技术有关的资源。每个集包含的资源，跟网上 iOS Developer Library 内的相同。默认情况下，Xcode 会自动安装并更新可用的文稿集。作为一名 iOS 开发者，您会发现这两个文稿集已经自动安装：

- iOS Developer Library，包含专用于编写 iOS 应用程序的文稿
- Xcode Developer Library，包含专用于使用 Xcode 作为开发环境的文稿

▶ **在 Xcode 中查看 iOS 文稿**

1. 选取“Window”>“Organizer”，然后点按“Organizer”窗口中的“Documentation”。

   “Documentation”管理器出现，搜索导航器已启用。

2. 点按导航器选择栏中的浏览按钮 ()。

   已安装的开发者资源库会出现在文稿导航器中。

3. 选择“iOS Library”。

点按一个导航器按钮，以选取您想如何查找文稿：

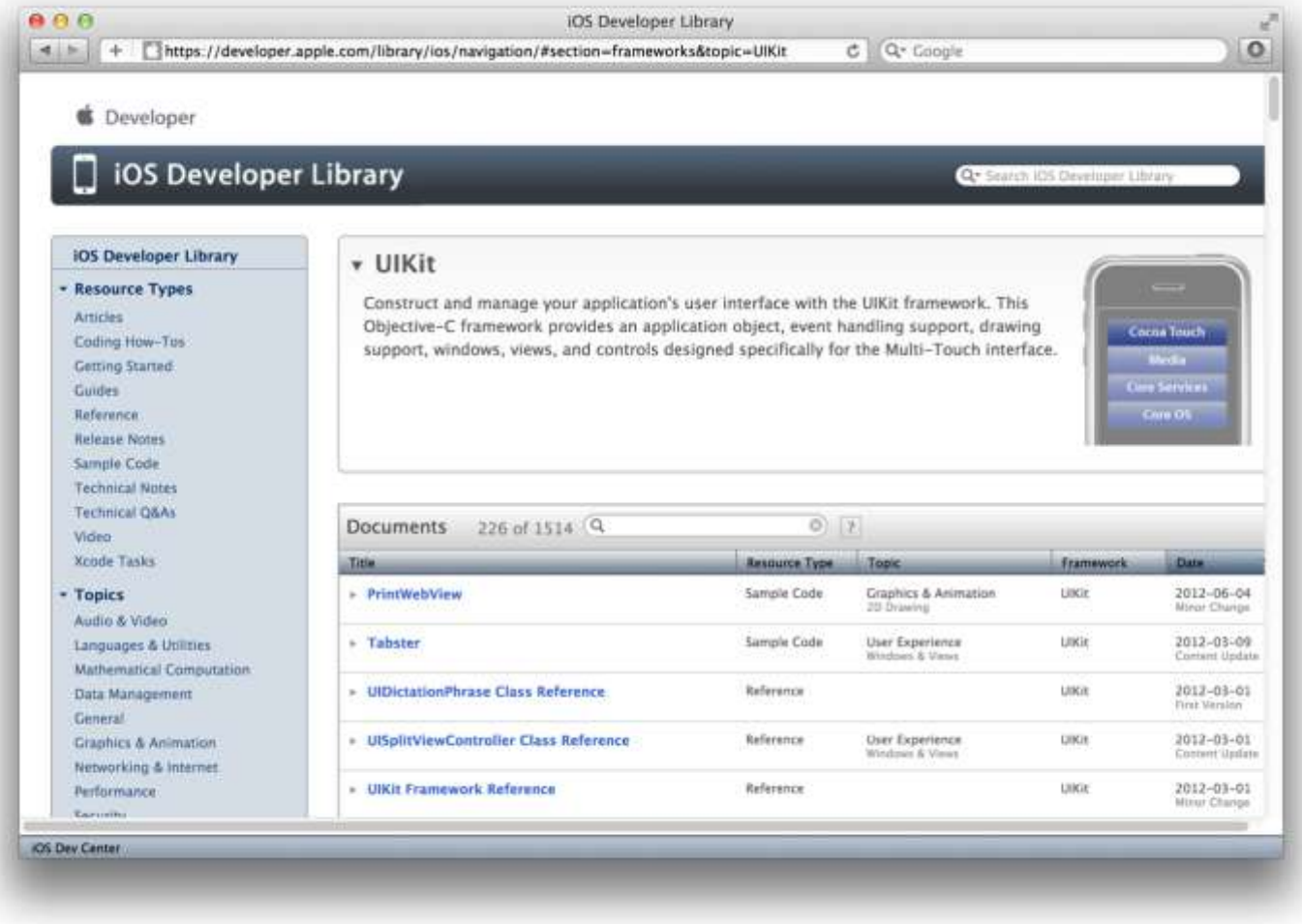
-  浏览文稿层次。
-  搜索特定术语。
-  使用书签返回到查阅过的文稿。

选择文稿后，它会在内容区域中以 HTML 格式开启。如果文稿也有 PDF 格式，则可在相关项目菜单中选取它以打开 PDF 版本。

内容区域中的跳转条，可让您进一步探索文稿和文稿所在的资源库。在熟练使用跳转条后，不妨通过选取“Editor”>“Hide Navigator”来增大内容显示区域。

在线查找开发者文稿

如果喜欢，您可以浏览网上的开发者资源库。例如，您可能想在 iPad 上或在未安装 Xcode 的电脑上阅读文稿。要访问 iOS Developer Library，请输入网址 `http://developer.apple.com/library/iOS`，或登录到 iOS Dev Center，然后点按 iOS Developer Library 链接。



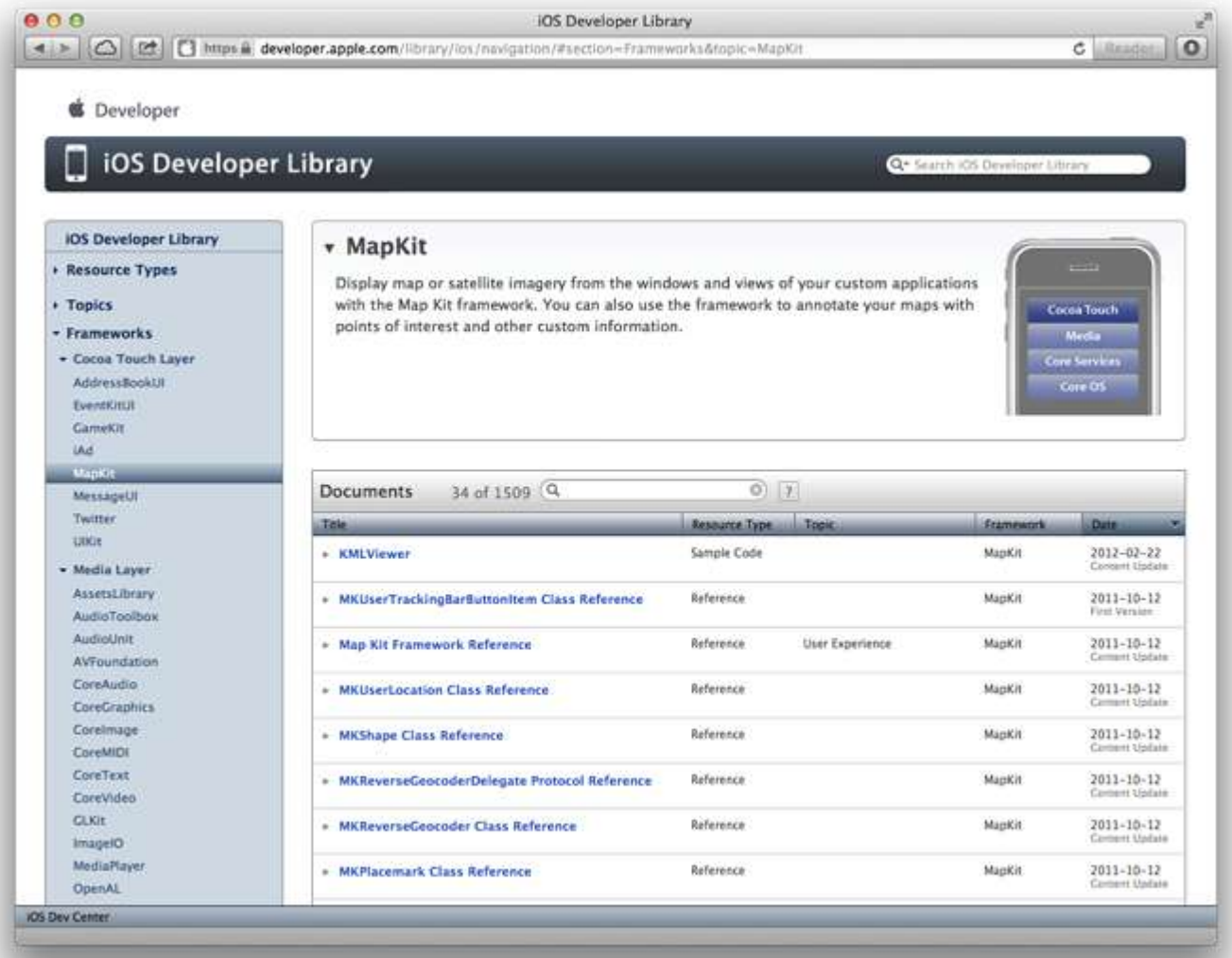
您也可以查看网站上 PDF 格式的文稿。如果文稿有 PDF 版本，则当文稿显示时，其右上角会显示一个 PDF 按钮。点按该按钮，就会在浏览器中显示 PDF 文件，或者按住 Control 键并点按该按钮下载 PDF 文件。

按资源类型、主题或框架过滤文稿

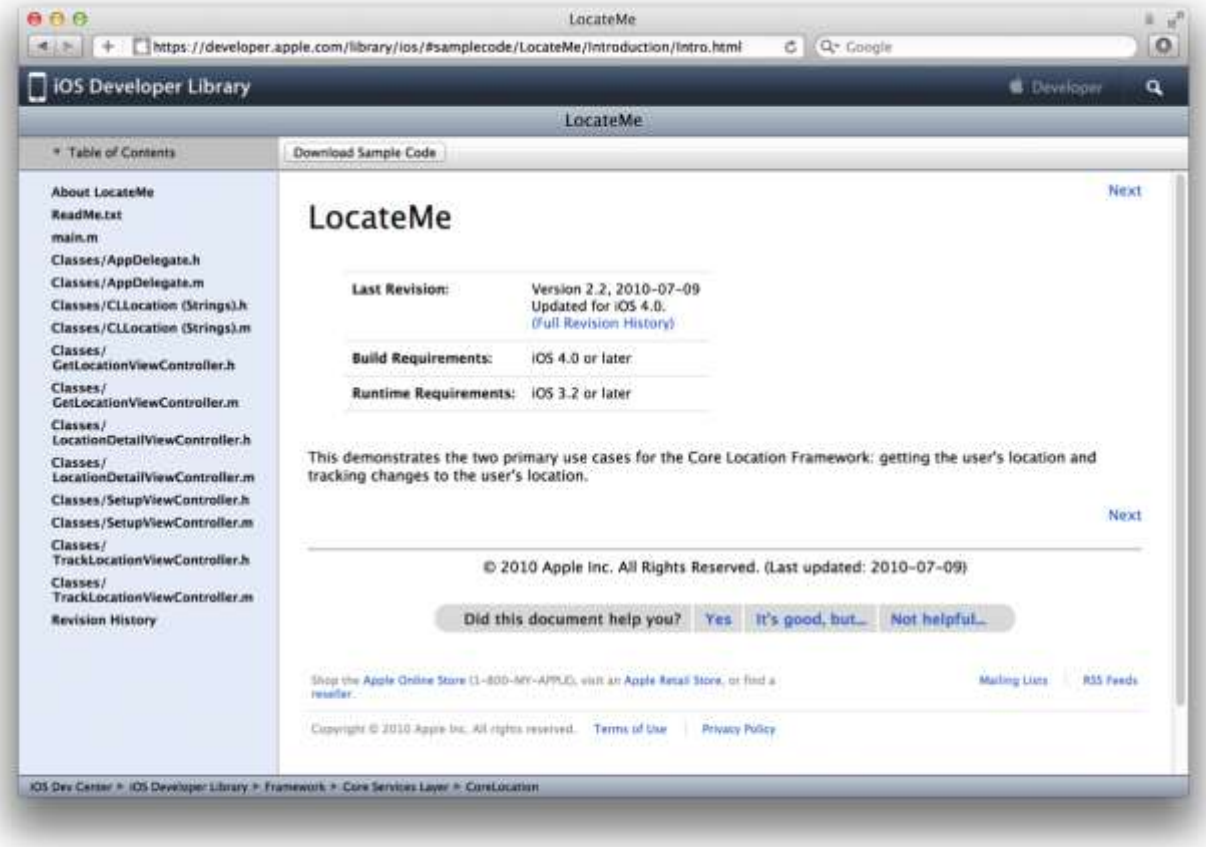
Xcode 和 iOS Developer Library 网页界面为您提供了几种方式来导航和查找资源。您可以按资源类型、主题或框架查看文稿。资源类型包括指南（描述概念和任务）、参考文稿（包含 API 详细信息）、可供下载的示例代码，以及由 Apple 工程师讲解的视频演示。也有为某些技术提供逐步指导的教程。导航“Topics”部分按主题区域查找资源，例如主题“Audio & Video”、“Security”和“User Experience”。使用“Frameworks”部分以导航至专属框架的文稿，例如“UIKit”、“Foundation”和“Core Data”的相关文稿。

Xcode 和网上 iOS Developer Library 使用同样的用户界面来过滤文稿。在 Xcode 中，选择左栏的资源库可以看到和 iOS Developer Library 中相同的起点页面。然后，使用内容区域中的目录边栏来浏览资源库中的内容。

当您选择目录中的一项时，资源列表会显示在右侧的内容区域中。使用列标题将资源按标题、资源类型、主题、框架（如果适用）或上次修改日期排序。还可以使用文稿栏中的搜索栏来过滤列表。例如，如果想要查看所有有关 UIKit 的参考文稿，可从目录中的“Frameworks”类别中选择“UIKit”，然后将列表按资源类型进行排序。所有参考文稿首先出现，接着是示例代码。如果知道一项技术的名称，但不确定它在哪里，您可在目录中选择“iOS Library”标题以显示该开发者资源库中的所有文稿，然后在搜索栏中输入搜索字符串。



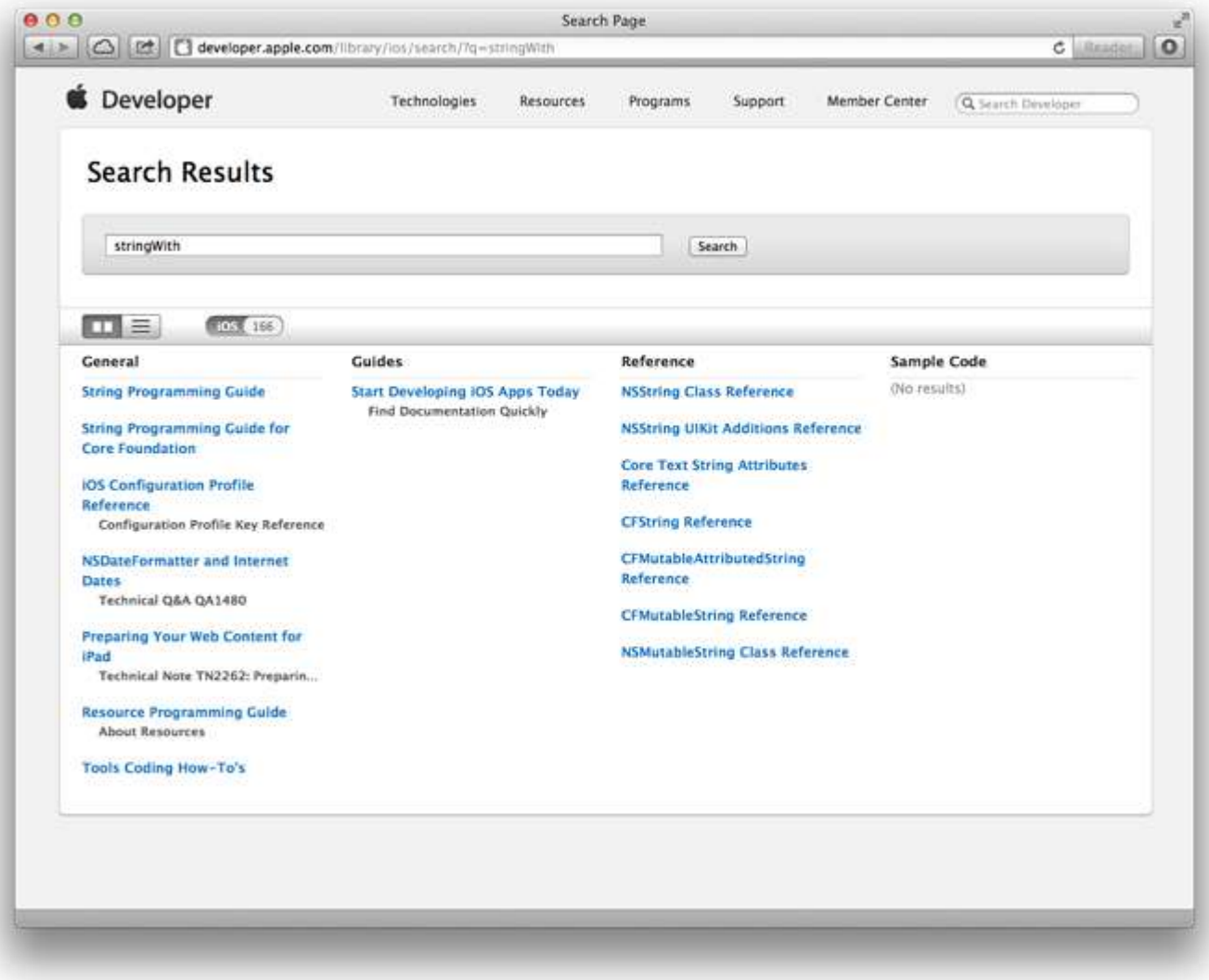
在资源库中，也可以轻松地查阅所有可用的示例代码。点按目录中的“Sample Code”（在“Resource Types”下面），然后使用栏标题和搜索栏，来过滤列表并对其排序。选择您有兴趣查阅的示例代码。如果您在 Xcode 中查阅示例，可以查阅所有 Xcode 项目文件，或者点按“Open Project”。如果您在 iOS Developer Library 中查阅示例，可以点按“Download Sample Code”。



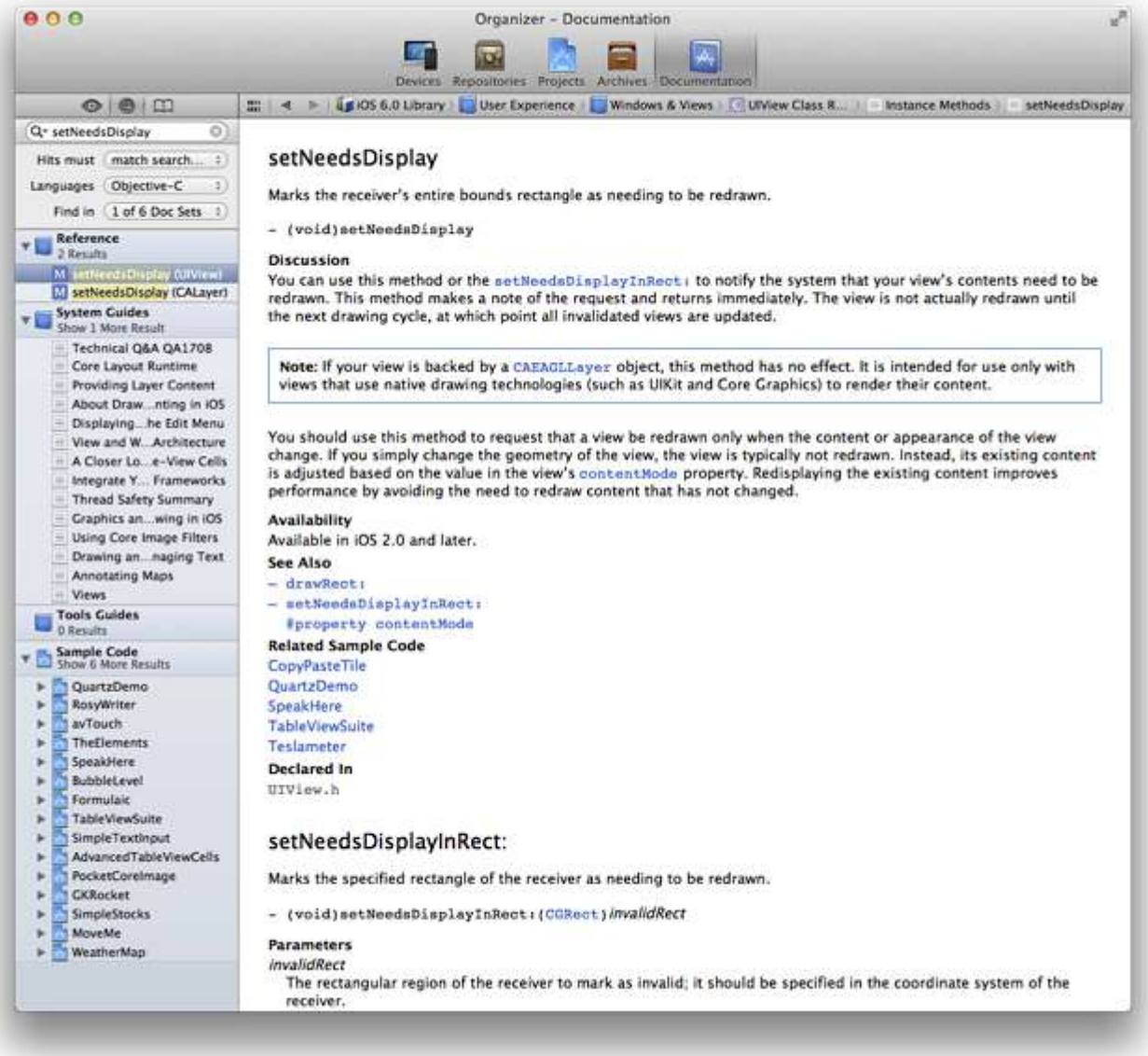
搜索开发者文稿

搜索开发者文稿以找到符合当前需要的信息。

使用网上 iOS Developer Library 右上角的搜索栏，来查询整个资源库。搜索结果按资源类型分组。例如，如果在搜索栏中输入 `stringWith`，搜索结果会将 *String Programming Guide* 显示为最相关的指南，并将 *String* 显示为示例代码项目。



在 Xcode 中，点按导航器选择栏中的搜索按钮 ()，以显示搜索导航器。搜索结果显示在导航器区域中，按资源类型（例如参考或指南）整理，并按相关性排序。



每个文稿都以文稿类型图标来标识。图标包括：

- 主题类别或概念文稿
- API 参考文稿
- 示例代码项目
- 文稿页面或章节
- 帮助文章

参考文稿中 API 符号的搜索结果，会以符号类型图标作进一步标识，显示的内容中搜索词会高亮显示。

使用查找选项，以将结果限定为与您所需信息最相关的资源。点按放大镜图标，然后选取“Show Find Option”。使用“Match Type”菜单，来指定搜索词必须在搜出的文稿中出现至少一次。使用“Doc Sets”菜单，来选取文稿集的任意组合，以进行搜索。使用“Languages”菜单，来将搜索结果限定为特定一组程序设计语言的文稿。

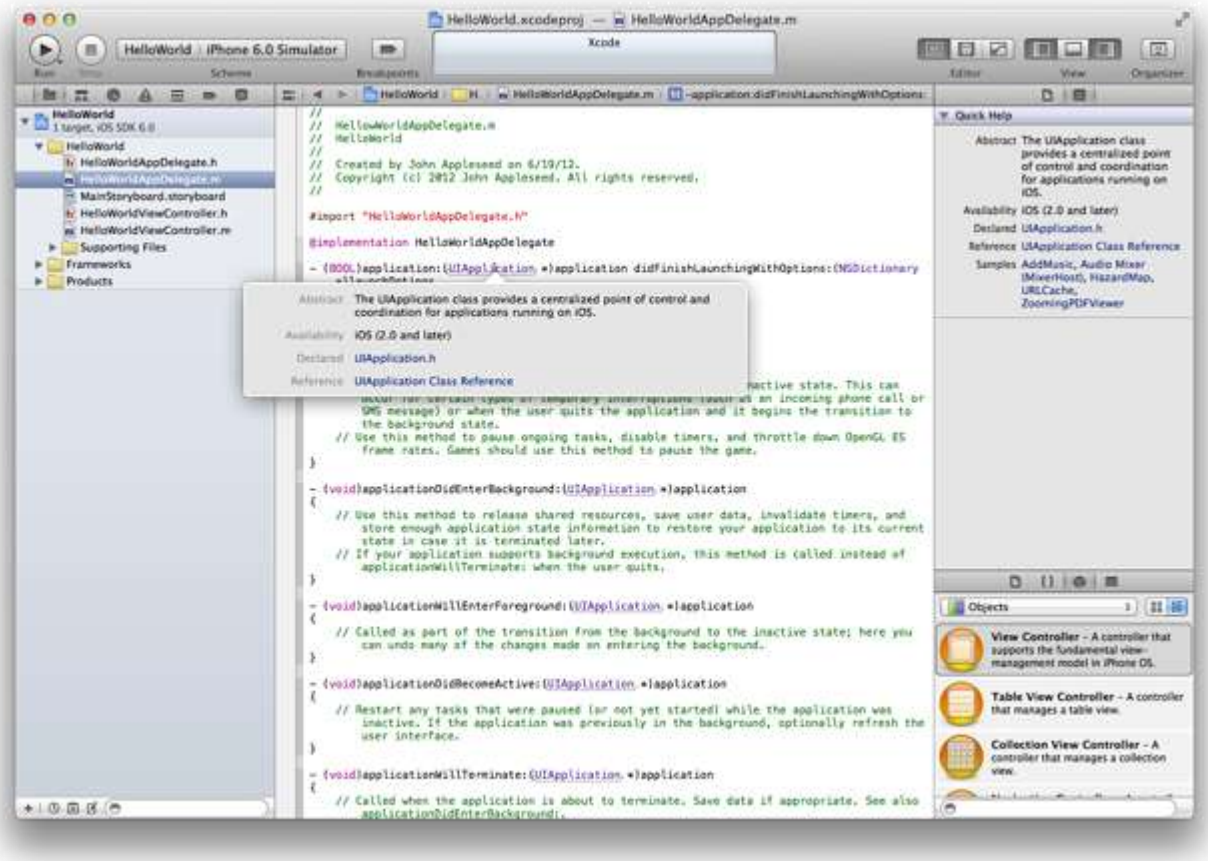
查看与 Xcode 关联的参考信息

可以直接从 Xcode 源代码编辑器中访问参考文稿。要打开源代码编辑器，请在导航区域选择一个源文件。源文件打开时，显示实用工具区域，点按其中的“Quick Help”检查器按钮。将插入点放在源代码编辑器中的 API 符号中，然后在检查器中查看文稿。

所显示的信息包括以下内容的链接：

- 该符号的完整参考文稿
- 符号声明所在的头文件
- 相关的编程指南
- 相关的示例代码

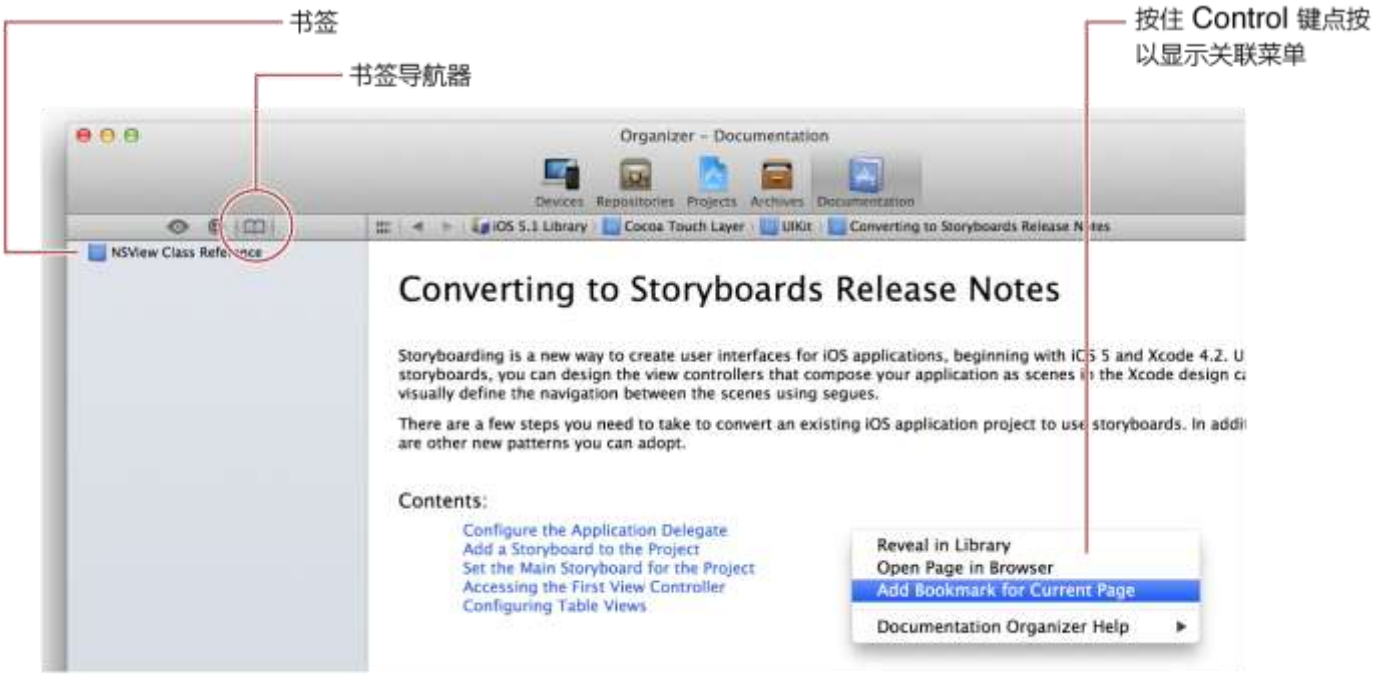
您还可以通过按住 Option 键，并点按源代码中的 API 符号，在弹出式窗口中查看有关该符号的简明参考信息。



使用 Xcode 书签轻松返回某个页面

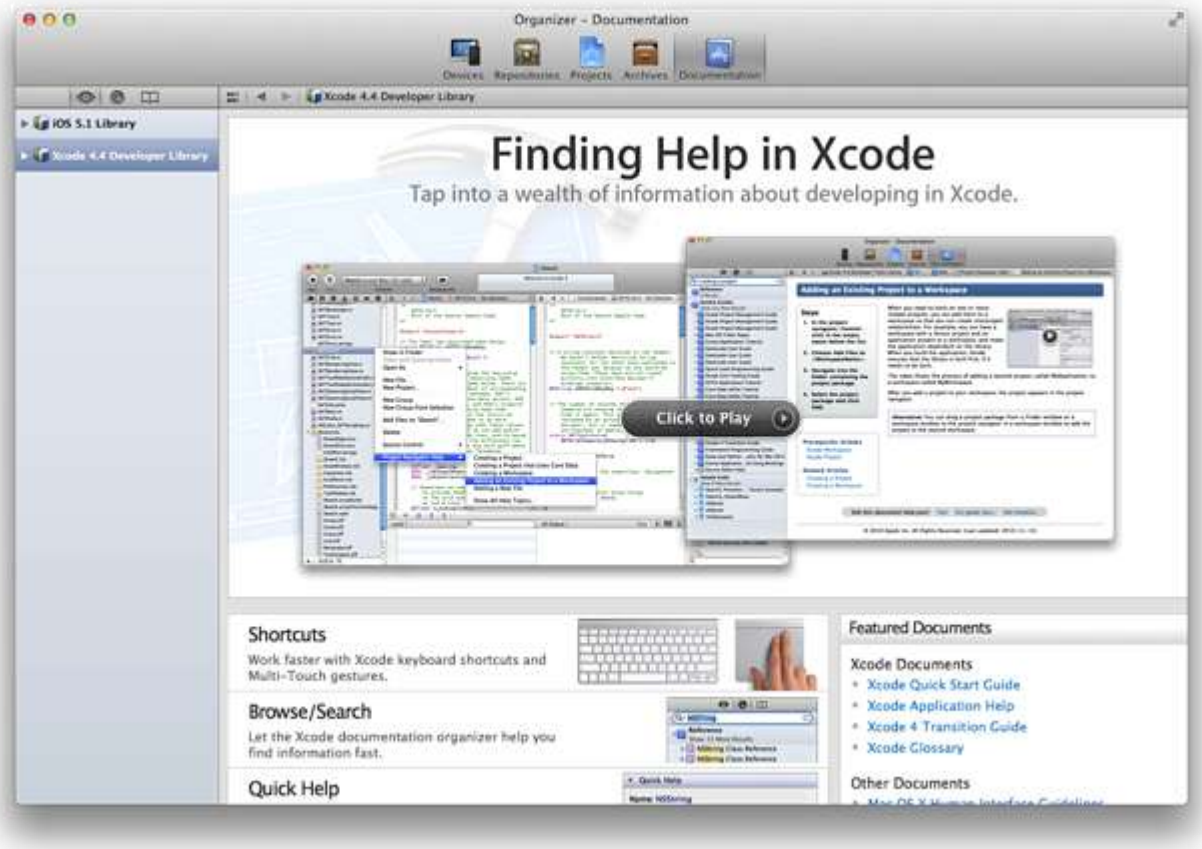
在 Xcode 的“Documentation”管理器中添加书签，以轻松返回某个页面、文稿或类别。您可以给任何页面或文稿添加书签。

在书签导航器中管理您的书签。默认情况下，书签会按您添加它们的顺序来排列。您可以将书签拖移到列表中的新位置，以更改它的顺序。您可以通过选择书签并按下 Delete 键，以删除书签。指定给书签的名称是其相应 HTML 页面的标题。（您不能更改书签的名称。）

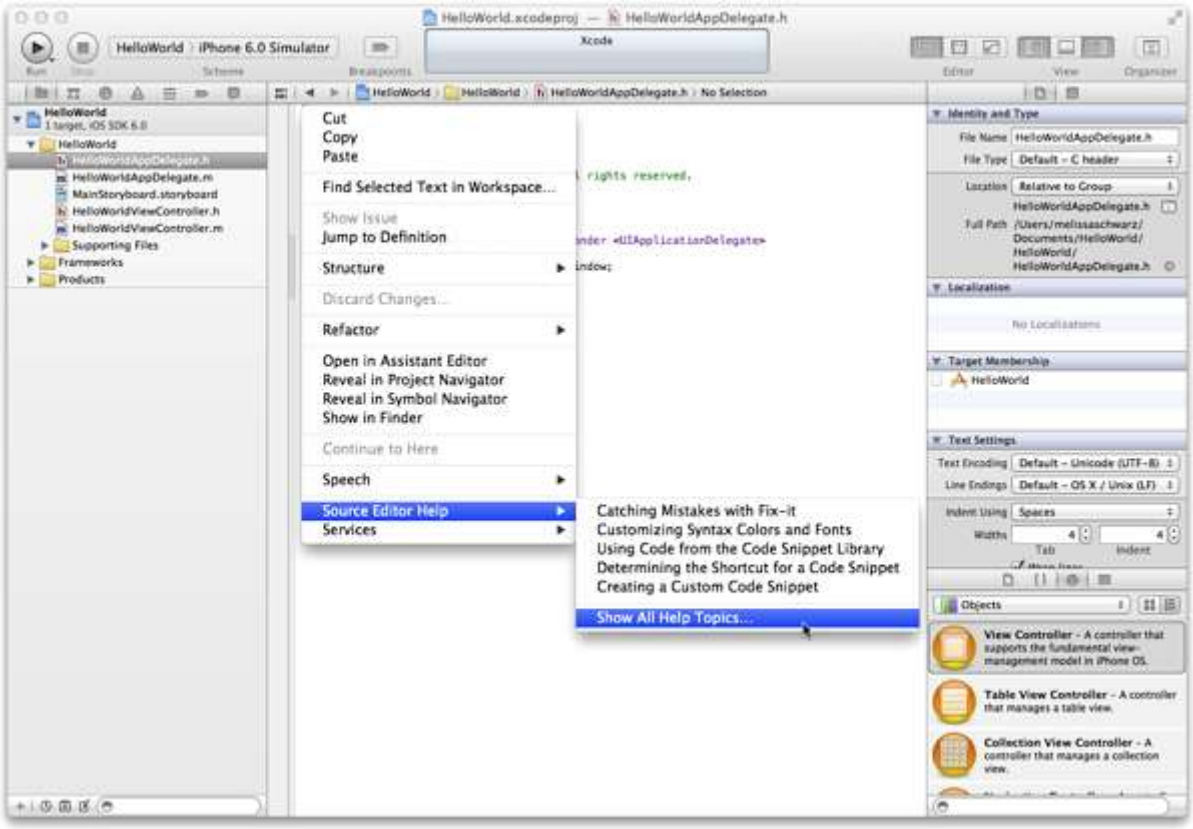


查找有关 Xcode 的帮助

Apple 也提供了关于如何使用 Xcode 的文稿。要浏览 Xcode 的在线帮助，请选取“Help”>“Xcode Help”，此时“Documentation”管理器会转变成“Finding Help in Xcode”开始页面。点按“Xcode Application Help”，来查看按照主题整理的帮助手册列表。每份帮助手册包含的文章，逐步讲解如何在 Xcode 中执行常见操作。Xcode 帮助中的许多文章，还包括解释所描述任务的视频短片。点按视频缩略图可播放视频。



在整个 Xcode 中，也可以通过快捷菜单访问许多 Xcode 帮助文章。按住 Control 键，并点按 Xcode 中的任何一个主用户界面区域，查看该区域可用的帮助文章的列表。如果文章很多，菜单不能一一列出，请选取“Show All Help Topics”，以查看所有相关的帮助文章。



在 iPad 上查看 iOS Developer Library

iPad 上的 Safari 有量身而设的界面供查看文稿。为节省屏幕空间，可以使用“Library”按钮（而不是目录）按照类型、主题或框架进行导航。

要在 iPad 上查看 PDF 文件，请选择想要阅读的文稿，轻按目录控制，然后轻按“PDF”。

与其他开发者和 Apple 工程师讨论问题

如果未能在 Xcode 或 iOS Developer Library 中找到所需信息，可以将问题发布到 Apple 开发者论坛。登录到 iOS Dev Center 时，Apple 开发者论坛的链接出现在页面的底部。

注：2012 年 12 月 5 日 Apple.inc 在 https://developer.apple.com/library/ios/navigation/ 发布第一版《马上着手开发 iOS 应用程序（Start Developing iOS Apps Today）》文档 整理一下生成 PDF 文档提供大家离线参考！

另外我是一名明年 6 月份毕业的大四学生 要是你们想招 IOS 实习生 或者

工程师记得联系我哦！谢谢~

我的 QQ：853693514

e-mail：guozbang@163.com