HZ BOOKS
华章IT

iOS/苹果
技术丛书

Xcode 江湖录

李俊阳 杜承垚 胡雪婷 卢力 编著

Xcode Jianghu

- 全面系统地介绍Xcode各种功能、配置方法、涉及众多细节，用一个贯穿全书的示例提供了最佳实操方案。
- 独特的构思，简洁的语言，让读者以轻松的方式去理解Xcode，嬉戏间便掌握了重量级工具。

机械工业出版社
China Machine Press

本书尝试以轻松的方式讲解Xcode这个高效工具，让读者在谈笑间便直通Xcode大门，进而掌握Xcode的常见使用方法，并且能够通过一些练习掌握Xcode的部分高级功能，从而开发出让世人惊叹的应用。

主要内容：

- 掌握Xcode的最基本使用方法，包括文件操作、编写代码以及编译运行应用。
- 熟悉Xcode的界面布局，能够在这些界面中快速找到所需的内容。
- 掌握Xcode的可视化界面设计方式，了解界面构造器（Interface Builder）、自动布局（Auto Layout）和屏幕分类（Size Classes）的使用方法。
- 掌握Xcode的常用高级编辑方式，包括如何进行搜索、本地化等操作。
- 了解属性列表（Property List）和Core Data的模型设计。
- 掌握编译方案（Scheme）、调试、测试等相关方法。
- 示例所在网址为https://github.com/Xcode-Jianghu。

上架指导：计算机/移动开发
ISBN 978-7-111-51912-6
9 787111 519126 >
定价：69.00元

投稿热线：(010) 88379604
客服热线：(010) 88379426 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn

iOS/苹果
技术丛书

Xcode 江湖录

Xcode Jianghu

李俊阳 杜承垚 胡雪婷 卢力 编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Xcode 江湖录 / 李俊阳，杜承垚，胡雪婷，卢力编著 . —北京：机械工业出版社，2015.10
（iOS/ 苹果技术丛书）

ISBN 978-7-111-51912-6

I. X… II. ①李… ②杜… ③胡… ④卢… III. 软件工具－程序设计 IV. TP311.56

中国版本图书馆 CIP 数据核字（2015）第 248944 号

本书尝试以轻松的方式讲解 Xcode 这个高效工具，让读者在谈笑间平蹚 Xcode 世界，进而掌握 Xcode 的常见使用方法，并且能够通过一些练习掌握 Xcode 的部分高级功能，从而开发出让世人惊叹的应用。本书分为四大部分："初入江湖——基础篇"包括第 1 ～ 3 章，介绍苹果开发者计划、最基本的项目开发流程、Xcode 主界面、基本概念、项目配置等。"外功修炼——设计篇"包括第 4 ～ 6 章，介绍界面生成器（Interface Builder）的用法以及自动布局、屏幕分类等，让初学者和设计师能够借助 Xcode 提供的可视化界面设计工具来快速设计想要的界面。"内功修炼——开发篇"包括第 7 ～ 14 章，介绍 Xcode 的高级用法，例如"编辑器"的用法，"属性列表"和"Core Data"两个存储技术的可视化编辑设计器，库、框架等共享代码的原理和使用方法，编译方案和运行目标两个对应用编译过程有影响机制的原理和相关操作，调试方法，比如断点、LLDB 等方式，代码测试方法，版本管理，一个真实应用应该如何上架等。"随身锦囊——附录篇"包括 4 个附录，介绍 Xcode 特有的小功能、小组件、小设置等，方便读者查询。附录 A 介绍了 Xcode 中的一些小技巧，包括快捷键、代码片段、系统设置等，附录 B 介绍 Xcode 额外提供的一些好用的功能，附录 C 介绍 Xcode 中提供的各种模板，附录 D 介绍获取 Xcode 帮助的相关方式。

Xcode 江湖录

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：吴　怡　　责任校对：董纪丽
印　　刷：北京市荣盛彩色印刷有限公司　　版　　次：2015 年 11 月第 1 版第 1 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：22.75
书　　号：ISBN 978-7-111-51912-6　　定　　价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

华章IT
HZBOOKS | Information Technology

Introduction 写在前面

梧桐松柏共秋色，驼缃葱倩两相宜。

每到秋日，武昌城郊的小茶馆中便坐满了人，在这清寂的冷风中觅得一个好去处可不见得是一件易事。

“啪。”

惊堂木一下，满座俱静。

“野草闲花遍地愁，龙争虎斗几时休。抬头吴越齐秦楚，转眼梁唐晋汉周。如今这手机应用的世道，群雄割据，可谓乱世之秋。最令人瞩目的当属‘安卓派’和‘苹果帮’，这二者早已牢牢占据了绝大部分市场。众所周知，安卓派控制了绝大部分的领土，而苹果帮在与安卓派的战争中，仍立于不败之地。相信在座的各位也都曾听说过，苹果帮拥有一个强大的法宝，其名为 Xcode。这个法宝啊，可谓是无所不能，神奇至极，从内功心法到外功招式皆能良好适应。而安卓派直到最近才推出了他们自己的法宝：Android Studio。不过其功能和如今的 Xcode 相比，仍然有一定的差距。Xcode 究竟何许来历？今日我们便来为诸君一一述说。”

“Xcode 是从曾经的 NeXT 帮所拥有的‘Project Builder’这件法宝中精炼出来的，可谓是取之精华，去其糟粕。话说这个 NeXT 呀，是由伟大的乔帮主一手创建，在乔帮主回归苹果帮之后，NeXT 自然也划归到了苹果帮的旗下。”

“2003 年，Xcode 1.0 版本横空出世。但是很可惜，这个版本的推出并未在 IT 江湖中激起太大风浪，这时候的江湖中仍然流传着以微软为主的‘VS’系列法宝。”

“直到 2008 年，在第一代 iPhone 诞生的一年之后，引入了 iOS SDK 这个心法的 Xcode 3 才如同一声惊雷，引发了江湖动荡。众多侠客纷纷转投苹果帮旗下，开始一心一意修炼起 Objective-C 这门内功，并且也纷纷研究出了众多精彩招式。”

“2011 年 Xcode 4 发布，苹果帮吸取了众多的外派武功，将诸多法宝的优势融入到了自身的法宝当中，譬如 Interface Builder。鄙人认为 Xcode 4 才真正能和 VS 等法宝隐隐抗衡。”

“也正是同一年，苹果帮的创始人乔帮主羽化登仙。苹果帮便被江湖众派看衰，也面临了诸多打压。这个时候，苹果帮抛弃了以 GCC 为基础的修炼方式，转而使用 LLVM 来修炼。”

“而苹果帮仍默默地前进，在 2013 年推出了 Xcode 5。Xcode 5 提供了一系列帮助学徒检测自己经脉的途径，例如可视化调试器。同时，在外功修炼方面，Xcode 也提供了诸如资产管理、自动布局等简单但强大的招式。”

“在 2014 年，苹果再次放出了一个重磅炸弹：Xcode 6。在这个版本中，加入了苹果帮的架构师 Chris Lattner 花费了 4 年时间研究出的新内功心法：Swift。Swift 面向 Cocoa 和 Cocoa Touch，几乎可以完美地与 Objective-C 兼容，不用担心冲突，而且 Swift 的入门曲线比 Objective-C 要小很多，学徒们不必耗费大量的时间来初窥门径。此外，Interface Builder 也提供了实时渲染的功能，修炼的内功可以即时显示出来。”

“书曰：天下风云出我辈，一入江湖岁月催。皇朝霸业谈笑中，不胜人生一场醉。这 Xcode 的玄妙之处啊，吾等视之枯燥无味，然帮众们却乐在其中。正所谓：仁者乐山，智者乐水，预知后事如何，请听下回分解。”

话毕尺落，说书人轻拈须发，合了扇子，向众人拱手。茶客们回味过来，皆齐声叫好。落在窗旁的一只云雀被众人的聒噪惊到，展翅掠出了茶馆。

茶客中的一位少年目光随着云雀，渐渐移向远方，忽然脸上露出了一丝笑容。茶杯旁边，蓝色的卷轴正泛着微弱的光芒。

我们的故事才刚刚开始……

本书目标

我们衷心希望以轻松的方式将苹果公司开发的最出色、最好用以及最可靠的 iOS 以及 OS X 集成开发环境（Integrated Development Environment，IDE）介绍给大家。让大家能够尽快地入门 Xcode，熟练掌握 Xcode 的常见使用方法，并且能够掌握 Xcode 的一些高级用法，从而开发出让世人惊叹的应用。

通过本书，你可以达到以下学习目标：

- 掌握 Xcode 的基本使用方法，包括文件操作、编写代码以及编译运行应用等。
- 熟悉 Xcode 的界面布局，能够在这些界面中快速找到所需的内容。
- 掌握 Xcode 的可视化界面设计方式，了解界面构造器（Interface Builder）、自动布局（Auto Layout）和屏幕分类（Size Classes）的使用方法。
- 掌握 Xcode 的常用高级编辑方式，包括如何进行搜索、本地化等操作。
- 了解属性列表（Property List）和 Core Data 的模型设计。
- 掌握编译方案（Scheme）、调试、测试等相关方法。

- 掌握如何向 App Store 上传应用。

“不积跬步，无以至千里”，任何编程语言、任何工具都需要不断地使用和练习才能够熟练掌握，从而达到“妙笔生花”的境界。通过本书的学习，你将对 iOS 或者 OS X 应用的开发流程不再陌生，从而踏上苹果开发者的江湖之路！

面向读者

在购买或者阅读本书之前，我们希望你是下列读者群体中的一员，我们不希望你花费冤枉钱来购买一本只能封存于书架最顶端的书籍，也不希望你购买这本书之后发现书中的内容对你来说是没有帮助的。因此，我们在此说明一下，本书不是武侠小说，希望你能够仔细阅检视一下本书的主要内容是否符合你的期望。

本书面向的读者包括如下几类：

- 对苹果开发感兴趣，已经粗略了解 Objective-C 或者 Swift 语言的读者，并且迫不及待地想尝试开发的开发者。
- 从别的平台转过来的开发者，想要了解一些关于 Xcode 的基本用法。
- 想要了解关于 Xcode 中比较高级的用法的读者。
- 英文比较吃力，不想费力去搜索 Xcode 相关资料的读者。
- 想要使用苹果新技术来进行设计的设计师。

如果你期待了解一些非常高级的用法，那么本书可能不是你的最佳选择，目前本书不会包括以下内容：

- Xcode Server、AppleScript 等方面的内容；很遗憾，本书对于这部分内容暂时没有介绍，在未来的版本中，我们会考虑适当增加这方面的内容。
- OS X 应用开发；我们建议你去阅读相关的英文书籍和查阅相关文档，关于 OS X 开发的中文版系列教程十分少，本书仅仅提及关于 OS X 的部分内容，不涉及代码部分。
- 有关 Objective-C 以及 Swift 编程语言的语法介绍；本书不是一本语法书，因此本书不会着重讲解书中每行代码的意思，我们希望你拥有 Objective-C 或者 Swift 语言的基础知识。

本书构思缘由

我们童年时期深受“侠客”、“修真”风格的故事、小说所影响，并且又不想让这本书变得晦涩难懂，因此构思本书的时候，自然而然地就想将武侠风格融入到本书当中。

然而，将一个颇具英文风格、科技感、现代感的事物——编程——转变为“武侠”风格是一件非常难的事情。因此我们尝试“以武侠故事带动知识”的原则来写作，使读者在学习技术的过程中不至太枯燥。

书名“江湖录”的由来，部分灵感来源于金庸先生的《书剑恩仇录》以及《笑傲江湖》。在我们的头脑中，Xcode 是一个极佳的法宝，整个“开发者的世界”是一个“腥风血雨”的“江湖”，而 Xcode 在这个江湖中占有一定的地位。少年青锋为了掌握这个法宝，不畏艰辛，努力学习，最终成长为一名大侠。我们期待读者也能跟随少年的步伐，平步青云。

本书架构

首先说明一下，本书是一本“工具书”，旨在讲解“开发工具”。如果你想要了解关于如何从头建立一个完好的应用，那么绝大多数“语言”层面的书籍都会有所介绍。

本书分为以下 4 个部分。

初入江湖——基础篇

这个部分主要简单讲述了如何创建一个简单的小项目，并且对 Xcode 的界面和基本操作进行一个粗略的介绍。

这部分是为初学 Xcode 和编程的读者准备的。

第 1 章　小试牛刀——Xcode 初体验

介绍了 Xcode 的下载方式，并对苹果开发者计划做了简要的介绍，并且带领读者完成一套最基本的流程——创建项目、运行项目以及移除项目。让初学者小试牛刀，尝一尝把玩 Xcode 的感受。

第 2 章　纵观全局——布局探索

介绍了 Xcode 的界面布局，详细讲解 Xcode 主界面每一部分的名称、作用，让读者对这些区域能够有一个粗略的概念，知道一些概念、术语，并且能够快速找到后文所说的部分。会当凌绝顶，一览众山小。

第 3 章　藏经阁——项目管理

介绍了如何对项目进行配置，包括对应用文件、应用对象以及应用资源等内容进行管理。只有东西收得整整齐齐，搭建良好，应用才能正常运转。

外功修炼——设计篇

这个部分主要介绍了界面生成器（Interface Builder）的用法以及相关技术，主要面对设计师，让初学者和设计师能够借助 Xcode 提供的可视化界面设计工具来快速设计想要的界面。

第 4 章　风水宝地——界面生成器

主要介绍了界面生成器的样式、种类以及它们的使用方法。

第 5 章　万物莫不有规矩——自动布局

主要介绍了自动布局（Auto Layout）技术的使用方式，该技术能够让布局能够更好地适应不同尺寸的设备。

第 6 章　万法归一——屏幕分类

主要介绍了屏幕分类（Size Classes）技术的使用方式，该技术能够良好地适应不同尺寸的 iOS 设备。

内功修炼——开发篇

这个部分主要介绍了一些 Xcode 的高级用法，让读者能够从中学习到更有用的 Xcode 使用技巧。

第 7 章　渐入佳境——高级编辑

介绍许多关于“编辑器”（Editor）的高级编辑用法，比如语法感知、重构、迁移、搜索等在文本层级上进行操作的功能，从而更好地完成代码。

第 8 章　气沉丹田——持久化存储编辑器

介绍“属性列表”和“ Core Data”两个存储技术的可视化编辑设计器，完成对存储技术的骨架设计。

第 9 章　前人栽树——共享代码

介绍库、框架等共享代码的方法原理、使用方法和制作方法，还介绍如何使用 CocoaPods 来帮助管理代码。

第 10 章　武功是怎样炼成的——编译系统

介绍编译方案（Build Scheme）和运行目标（Deployment Target）的原理和相关操作。

第 11 章　谨防走火入魔——调试

介绍 Xcode 上的调试方法，比如断点、LLDB 等方式。

第 12 章　功力精进的途径——单元测试

介绍 Xcode 上的代码测试方法，比如功能测试和性能测试等。

第 13 章　返老还童——版本管理

介绍 Xcode 上可以使用的代码管理方式，借此开发者可以方便地管理代码，开展多人协作。

第 14 章　实战是提升实力的唯一真理

介绍真实应用应该如何上架。

随身锦囊——附录篇

这个部分主要介绍一些 Xcode 有的小功能、小组件、小设置等。

附录 A　Xcode 小技巧

介绍 Xcode 中的一些小技巧，包括快捷键、代码片段、系统设置等。

附录 B　不二法门——Xcode 工具箱

介绍 Xcode 额外提供的一些好用的功能。

附录 C　武术套路——模板

介绍 Xcode 中提供的各种模板，包括文件模板、控件模板等。

附录 D　你不会独孤求败——求助渠道

介绍获取 Xcode 帮助的相关方式。

本书使用的 Xcode 版本

本书结笔于 2015 年 7 月，这个时候正值苹果发布了 iOS 9、OS X El Capitan、Xcode 7 beta 以及 Watch OS 2。但是由于这些版本还不是很稳定，因此本书是基于 iOS 8、OS X Yosemite、Xcode 6.4 以及 Watch OS 来写的。

本书示例代码

比起教各位如何从头搭建一个完好的应用，不如将一个完好的、已经上架的应用交付给各位。这样各位如果感兴趣在学习如何使用 Xcode 时也可以自行研究一下真实应用的实际效果。

本书的示例存储在 Github 上面，地址是：

OC 版本：https://github.com/SemperIdem/CrazyBounce-OC

Swift 版本：https://github.com/SemperIdem/CrazyBounce-Swift

这个示例提供了一个能够运行在 iOS、Mac、Apple Watch 三个平台上的简单的弹球小游戏，并且提供了 Objective-C 以及 Swift 两个版本。

Xcode 还支持许许多多的新奇技术，比如持续集成（Continuous Integration）、AppleScript、Apple 事件绑定等内容。这些内容都十分难，我们目前还不能将它们融会贯通，并以简单的语言跟各位分享，因此我们忍痛删除了这些章节。

由于我们学习和使用 Xcode 不过才几年时间，技术水平有限，因此本书在某些部分一定会有错误。对于每一名发现重大问题的读者，我们都会将你们的名字添加到本书的致谢清单当中。如果你发现了更为重大的错误，比如说大面积的理论误人子弟之类，我们会在本书的

下一个版本中为你寄去新书作为感谢。

我们计划在下一个版本中，让本书适用于最新的 Xcode、OS X、iOS 以及 Watch OS 版本，另外还会着重添加关于持续集成（Continuous Integration）、Instruments、Playgrounds 以及 Xcode 7 新特性等更多、更新的内容。

期待读者提出宝贵意见，作者邮箱：xcodejianghu@126.com。

李俊阳、杜承垚、胡雪婷、卢力

2015 年 7 月 28 日写于武汉理工大学

目　录 *Contents*

外功修炼——设计篇

内功修炼——开发篇

初入江湖——基础篇

Chapter 1

第 1 章

小试牛刀——Xcode 初体验

忆往初，谈笑中多少情仇恩怨；道不尽，生死间几许红尘缱绻。

世人常言，人在江湖，身不由己。但江湖之外何尝能体会到江湖中人的快乐？正有这样一位少年良辰，年方弱冠，便毅然决然地踏入了编程江湖。年少之志，意气风发，欲挥斥方遒。试问哪位初入编程江湖之人的梦想或一生浪迹天涯，或成就一番功名，随后有朝一日其风流往事在某个小茶馆被小老儿这般人物评说。诸君可是这样认为的？

初入江湖，首先就是要有一把趁手的兵器，恰逢苹果帮新推出了 Swift 这样一门简单、强大的内功心法，极大地降低了入门的难度，于是良辰便决心踏上了修炼 Xcode 之路。经过一番打听，他来到了岚风谷……

1.1 下载 Xcode

9 月，岚风谷。一位执剑侠客正立于少年面前，默然片刻，云：

既然你已拜入我帮门下，从今天起，就由我来教导你。看样子你已经初步掌握了 Swift 这门内功，不过，掌握了内功并不意味着你能够成为一名真正的编程侠客。试想，空有一身内力，没有武器，没有功法，很可能你连一只鸡都杀不了。因此，你就必须要牢牢掌握好 Xcode 这个极其有用的法宝，这也是本帮推荐帮众使用的唯一一件法宝。有了它，你才可能成为一名

仗剑天涯的编程侠客。

你首先检查一下自己的身体素质吧！要掌握 Xcode 这件法宝，还是有硬性要求的。如果根基不足，强行使用，轻则不成大器，重则伤筋断骨，有性命之忧。如果根骨不行，那就选择早一些版本的 Xcode 来使用。

Xcode 6.4 目前要求：

- Xcode 6.4 需要运行在 OS X Yosemite(10.10) 及其以上版本
- 硬盘至少余留 8G 以上的硬盘空间，当然越多越好
- 至少需要 2G 以上内存，才能够比较流畅地运行 Xcode
- 截至 7 月 30 日，Xcode 7 已经推出到了 beta1 版本，Xcode 7 推出了 Swift2.0、新的 Playground 特性、新的测试框架以及新的界面构造器元素，本书将以 Xcode 6.4 为蓝本，侧重介绍某些 Xcode 7 的新特性。

Xcode 这件法宝则能够帮助像你这样的“江湖菜鸟”迅速成长为“编程侠客”，它是一个十分强大的工具，而且自身集成了许多好用、强大的武器，如何完全掌握这些武器，就是你今后所需要下苦功进行学习的。那么对于你来说，应该从哪儿搞到 Xcode 这件威力十足的法宝呢？答案无不外乎以下几种：

一是前往我帮专门设立的 Mac App Store 当中购买，我帮要求购买 Xcode 的人必须拥有其 Apple ID，以证明其身份。当然，它是完全免费的。不过在等待入手的过程当中你可以去找个地喝杯茶，因为 Xcode 太大了，从总部调配过来需要很长时间。有些时候，由于某些原因，苹果官方的押运速度会非常慢，这点你可能清楚。

二是可以通过一些第三方的途径进行购买，然后通过第三方押运来快速地获得 Xcode。不过它真实体积放在那儿呢，再快也是快不了多少的。出于某种原因，这里恕我不能够告诉你如何寻找第三方途径去购买 Xcode。不过住在山顶的那位“百度”先生，以及墙外的“谷歌”先生，可都是远近闻名的百事通，你可以去他们那儿问一下，保准能够找到你想要的答案。

1.2　苹果开发者计划

可以说，当你开始使用 Xcode 的时候，就可以称呼自己为“苹果开发者”了，这也正是外界对于我帮帮众所给予的称号。不过，我帮并不会为普通帮众给予很多关注，只有交了供奉的帮众，才能成为“精英”，受到我帮的诸多优待。

加入开发者计划的方式可能会发生变化，不过如果你打算加入的话，我可以给你先描绘一个大概。前往苹果开发者计划网站（https://developer.apple.com/cn/programs/），会有专人来指引你加入的。那么问题是，所谓的“优待”都有哪些呢？

- 我帮将会给精英提供最新的开发套件以供尝鲜，包括最新的操作系统和开发套件。

- 能够提供专门的支持服务。如果你在修炼方面遇到了困难，可以去联系我帮，会有大师来为你解答。
- 能够发布应用，意味着你就能够在苹果帮的官方商店中销售各种产品。

目前，我帮的开发者计划面向于所有的苹果平台设备，包括 iPhone、iPad、Mac 和 Apple Watch，也就是说，加入了开发者计划，就能够在 App Store 里面分发 App 了，包括 iOS 上的 App Store 和 Mac 上的 Mac App Store。此外，Safari 扩展的分发权限也包含在里面。

稍安勿躁，看得出来你很关注其价格。我帮给出的价格仍然符合“高大上”情怀——688RMB，一年。没错，不要 998，只要 688，开发者计划带回家。

因此，你在考虑加入开发者计划之前，为何不先将 Xcode 练习到炉火纯青？这样就不会浪费手中的大洋了，当然如果你觉得自己财大气粗的话，就当我没说这句话。

当你在选择开发者计划的时候，要注意是选择以个人的身份还是以公司的身份发布应用。如果是后者，那么还需要向我帮提交一系列的文档证明才能完成申请，比如说法人实体名称和 D-U-N-S 码等。具体的操作，在此我就不再赘述了，我帮对此有详细的指南。

1.3 欢迎界面

就先让你一睹 Xcode 的风采吧。如你所见，Xcode 的表体是一张蓝色的卷轴，上面有一把锤子，这正是代表了我们“开发者”的标志所在。你要记住，我们“苹果开发者”所制作的东西，都是能让公众受益的，并且，我们都以公众喜爱我们的产品为荣。让我们打开它来一探究竟，首先出现的是 Xcode 的欢迎界面，如图 1-1 所示。

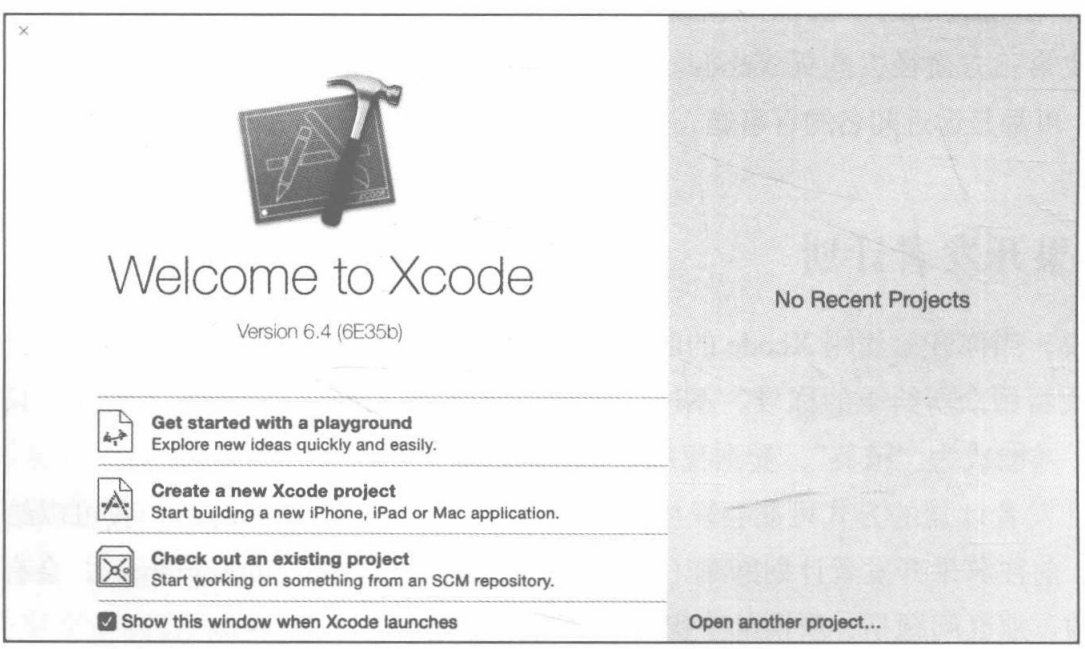


图 1-1 Xcode 欢迎界面

界面的左侧有三个选项，这是用来建立和导入项目的，第一个是使用 Playground 来开始项目，它能够帮助我们实现最新、最炫的想法；第二个是创建 Xcode 项目，这也是正常情况下创建项目的起点；第三个是从代码库下载代码，可以从代码库中直接下载到本地的 Xcode 中来进行运行或修改，往往用于版本管理和多人协作。

界面右侧是项目列表，上面列举了最近使用过的项目，方便快速打开常用的项目。如果要打开的项目不在这个列表中，那么可以点击列表最下方的“Open another project...”来打开项目。

如果不想再看到这个欢迎界面，可以取消勾选最下方的“ Show this windows when Xcode launches”复选框，这样再打开 Xcode 的时候欢迎界面就不会再出现了。

若要再次打开 Xcode 欢迎界面，那么依次选择菜单栏上的“ Window → Welcome to Xcode”即可，或者直接使用“Command + Shift + 1”快捷键。

1.4　认识 Playground

什么是 Playground？我更愿意称呼其为“训练场”。在这里，可以尽情地使用 Swift 这门功法来施展各式各样的招式，既可以用来练习 Swift，也可以用来实现某个功能然后将其融合到产品当中，还可以设计某一个算法然后观察它的显示结果。可以说，“训练场”让 Swift 展示了某些脚本语言的特性。图 1-2 显示了“训练场”的模样。

提示　实际上，Playground 只是提供了一个可实时编译运行并展示的交互式开发环境而已，类似于著名的 REPL。因此，Swift 并不是一门解释型语言，仍然还是一门编译型语言，这项功能完全借助于强大的 LLVM 编译器的能力。不过，目前 Playground 仅仅只支持 Swift 语言，但是国外开发者 Krzysztof Zabłocki 开发出了一个可以运行 Objective-C 语言的 Playground，感兴趣的读者可以前往他的 Github 上下载：https://github.com/krzysztofzablocki/KZPlayground。

在界面的右侧区域，可以实时地看到常量、变量的值、打印的内容以及循环次数，等等，这就是所谓代码版的“所见即所得”。

将鼠标指向 Playground 中所显示出来的值，便可以看到界面右侧出现了一只小眼睛和一个白色的圆圈，小眼睛就是“Quick Look”(快速查看)，即可以快速查看其完整的值，如图 1-2 所示，点击“快速查看”之后，从小眼睛的部位弹出了一个小对话框，对话框里面显示了该变量（常量）的值。

白色的圆圈就是“Show Result”(显示结果)，点击白色圆圈可以将对话框中显示的结果值嵌入到代码片段里面，如图 1-3 所示。这个功能大大地提升了 Playground 的可视化程度，使其变得更加有趣。再次点击白色圆圈，就可以将这个结果框从代码中移除，同样，在结果框的左上角点击关闭按钮也是可以移除结果的。

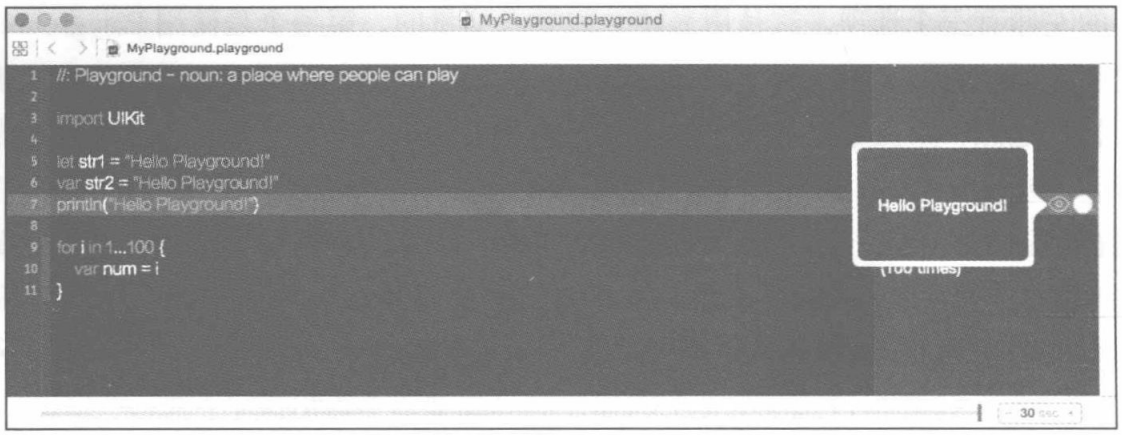


图 1-2 Playground 快速查看功能

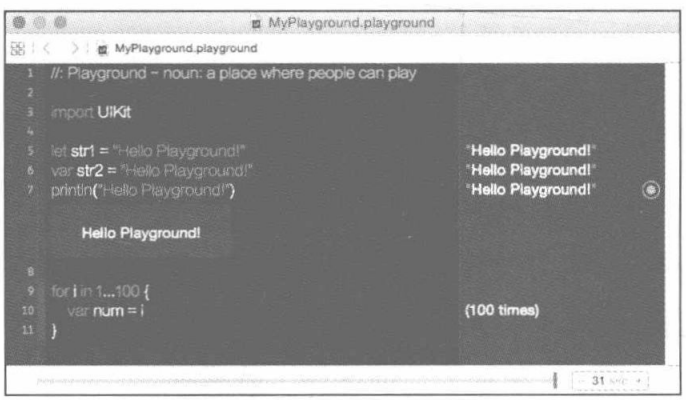


图 1-3 将结果显示在代码片段中

这里我们实现了一个循环次数为 100 次的循环结构，每次循环，i 的值就自增 1，然后用变量 num 来显示和记录这个值。单击“快速查看”或“显示结果”按钮，就会看到一个蓝色的“一次函数”，如图 1-4 所示。这个是以时间为横坐标轴显示的图形，主要是显示在循环体当中，借助这个功能，可以轻松查看在循环过程中变量值的变化情况。在这个结果框的右上角还存在三个按钮，分别是让其显示图表（Gragh），还是显示当前值（Current Value），抑或是显示全部的值，如图 1-5 所示。

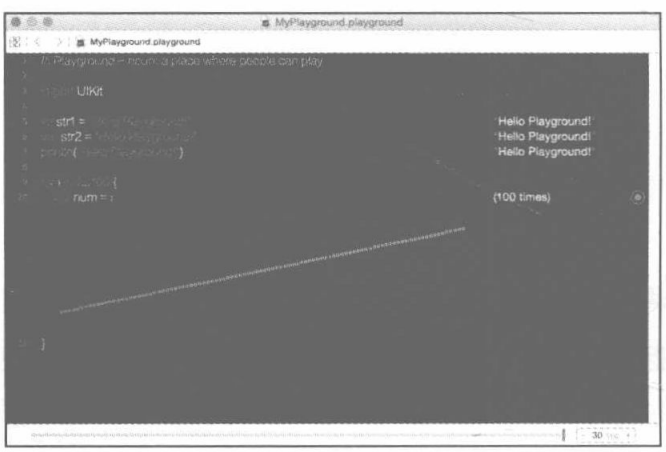


图 1-4 循环图表显示

图 1-5 显示效果选择

Playground 的下方则是时间控制栏，通过它可以调整当前时间，只要是对受时间控制的函数、方法体有效。右下角则是指定 Playground 运行最高层（top-level）代码的时间。

总之，Playground 是一个很有意思的东西，用它来练习 Swift 是再好不过的了，不过 Playground 仍然有一些限制，比如说 Playground 无法释放内存、性能受到极大的限制等问题。

1.5　创建项目

好了，是时候离开“训练场”了，现在我们要创建第一个项目了。

1）单击“Create a new Xcode project”，这时弹出一个“项目模板”选择窗口，如图 1-6 所示。有关“项目模板”的详细说明，请参阅本书的附录 C.2“项目模板”。这里我们选择“Mac”→“Application”→“Command Line Tool”（命令行工具模板），如图 1-6 所示，然后输入项目名称（Product Name）、组织名称（Organization Name）以及组织标识符（Organization Identifier），如图 1-7 所示。

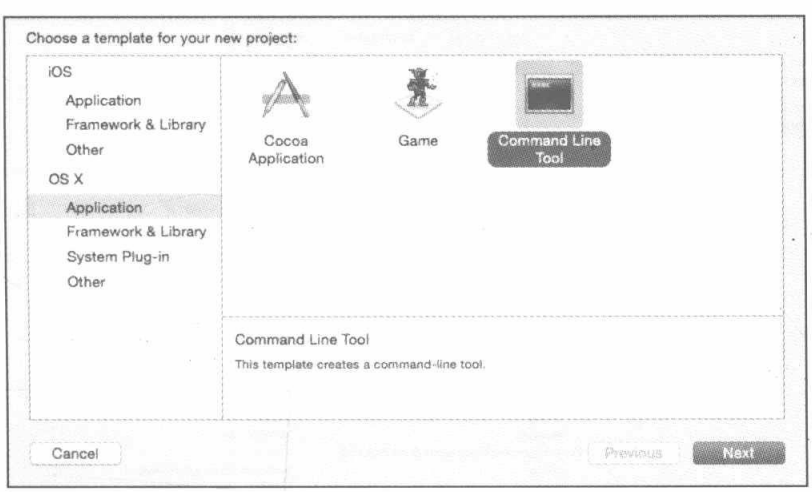


图 1-6　选择项目模板

图 1-7　新项目设置

要注意的是，这里不推荐用数字作为项目名称的首字符，也不推荐使用特殊字符来命名项目名称和组织标识符。因为如果这样做的话，那么下面的包标识符（Bundle Identifier）就会用“-”来代替不符合规范的字符，这可能会给应用上传带来麻烦，因此我不推荐你这样做。

组织标识符和包标志符都是用来唯一确定应用身份的标识符，一经创建后最好就不要修改，否则可能会带来麻烦。

语言（Language）有四种可以选择：Swift、Objective-C、C++ 和 C，你可以选择自己熟悉的编程语言进行下一步的操作，如果你使用的是 Swift 这门语言，那么在这里就选择 Swift。

2）单击“Next”，选择存储项目文件的路径。注意，下面勾选了“Create Git repository on”的版本管理选项，这可以让源代码使用 git 来进行版本控制。有关版本控制的相关内容请参阅本书的第 19 章。这里我们先不勾选。

3）最后单击 Create 完成项目创建，创建完的界面如图 1-8 所示。

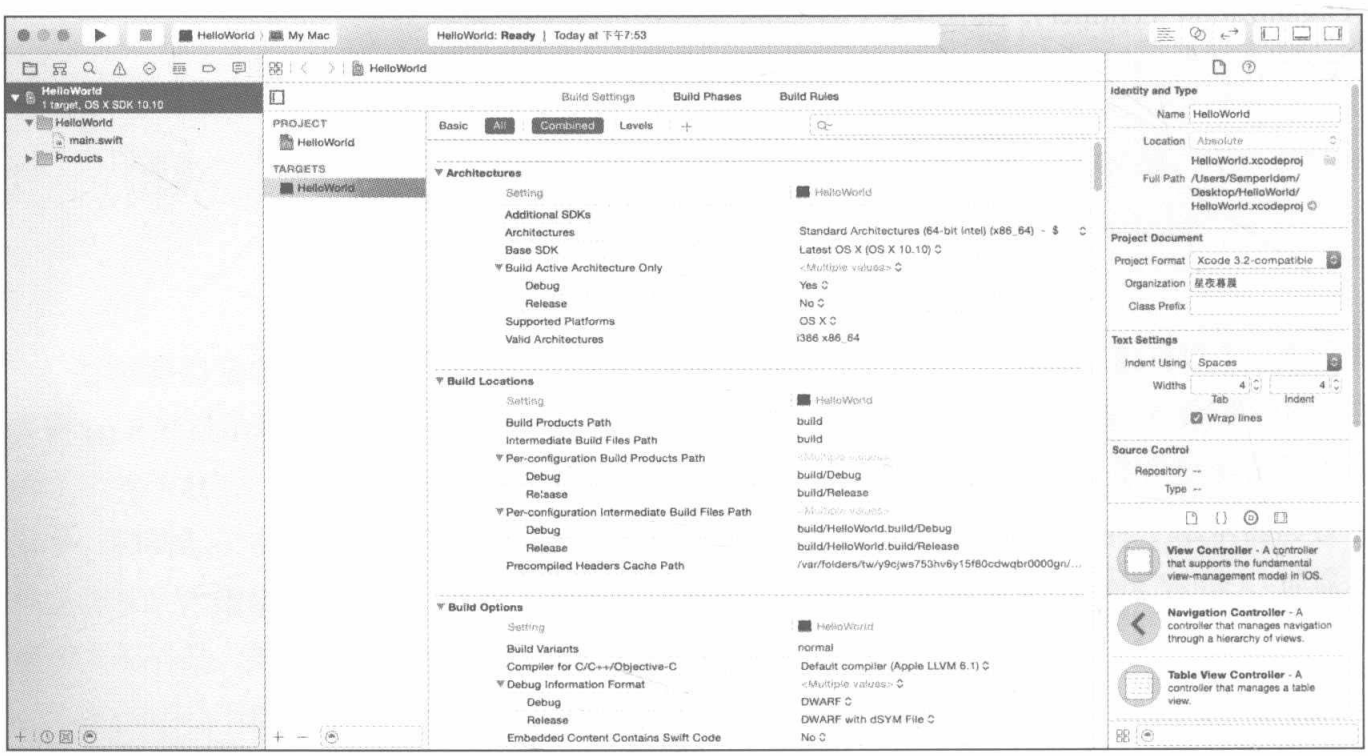


图 1-8　新创建好的项目界面

创建完成后，映入眼帘的便是“项目设置”页面，其中包含了 Build Settings、Build Phases 和 Build Rules 三个设置项，说明如下：

- Build Settings（编译设置）：每个选项由标题（Title）和定义（Definition）组成。这里主要定义了 Xcode 在编译项目时的一些具体配置。
- Build Phases（编译资源）：用于指定编译过程中项目所链接的源文件、依赖对象、库、图片等资源，也可以用于复制文件、运行脚本等辅助编译动作。

- Build Rules（编译规则）：决定编译过程中每一个文件要如何处理，由文件类型和处理动作组成。

关于项目设置的详细内容，请参阅本书的 3.2 节“对象管理”。至于如何配置这些项目，就不在我们的介绍范围内了，请自行查阅本帮提供的相关文献吧！

1.6 Hello world

打开“Main.swift”文件，可以看到，示例程序中已经为我们添加了如下的“Hello, World!”语句：

```
import Foundation

println("Hello, World!")
```

如果你选择 Objective-C，示例程序“main.m”是这个样子的：

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

int main(int argc, const char * argv[]) {
        @autoreleasepool {
        // insert code here...
        NSLog(@"Hello, World!");
        }
        return 0;
}
```

如果你选择 C++，示例程序“main.cpp”是这个样子的：

```
#include <iostream>

int main(int argc, const char * argv[]) {
        // insert code here...
        std::cout << "Hello, World!\n";
        return 0;
}
```

如果你选择 C，示例程序“main.c”是这个样子的：

```
#include <stdio.h>
int main(int argc, const char * argv[]) {
        // insert code here...
        printf("Hello, World!\n");
        return 0;
}
```

如果对文件进行了修改，那么页面的最上方会出现“Edited”标识，并且文件的图标会变

成灰色，如图 1-9 所示。

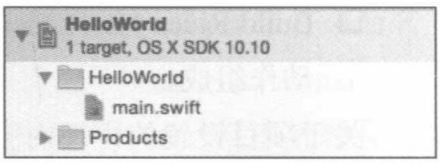


图 1-9 变成灰色的文件图标

要想保存对文件所进行的更改，请依次点击“File → Save”，便可以对文件进行保存，也可以使用快捷键“Command + S”快速保存。

1.7 生成并运行应用

完成了更改之后，就可以运行应用了！点击主界面左上角的三角形按钮，程序便可以运行了。或者依次点击“Product → Run”，也可以使用快捷键“Command + R”快速运行。

运行成功后，你会看到 Xcode 弹出了一个一闪即逝的提示框，提示“Build Succeeded”，即编译成功，如图 1-10 所示。

运行后的效果如图 1-11 所示，可以看到，“Hello, World!”出现在了“控制台”里面。

如果没有出现控制台窗口，那么依次点击“View → Debug Area → Activate Console”打开调试区域，或者直接使用快捷键“Command + Shift + C”打开。

图 1-10 Build Succeeded 提示

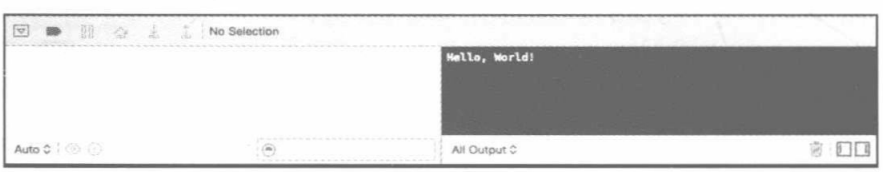


图 1-11 控制台中的“Hello World!”运行结果

1.8 移除项目

移除项目的方式很简单，直接找到项目所在的位置将其删除，但是这种方式会在 Xcode 中留下些许残留。如果没有强迫症的话完全就可以不必在意这些细节，不过有些时候还是清除为妙。

依次点击“Window → Projects”，打开项目管理窗口，然后选中刚刚删除的项目（被删除掉的项目颜色是红色的），然后按下“delete”键，或者选中左下角的齿轮，选择“Remove from Projects”选项，这样这些缓存文件便灰飞烟灭了，如图 1-12 所示。

“通过今天的学习，相信你也对 Xcode 的使用略有感悟了。现在，你可以称得上是入门了，不过离出师的标准还差得远呢！任重而道远，你回去之后要多加勤学苦练，熟能生巧，方能大有可为啊！”

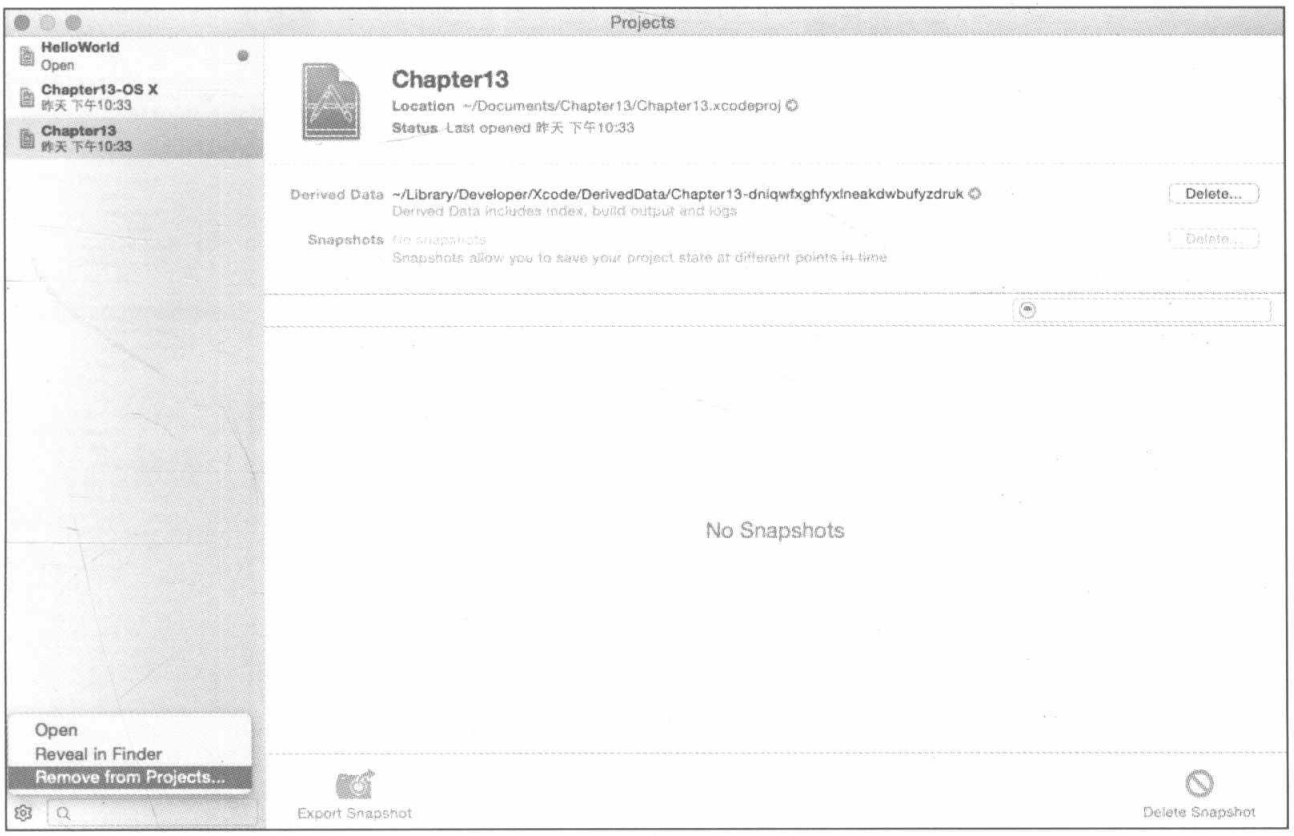
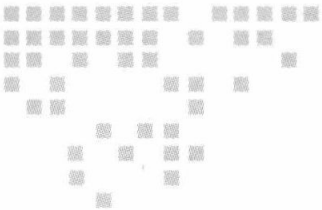


图 1-12　在组织器当中删除项目残留

说完，只听这位侠客大笑数声，一阵疾风略过，已不见了身影。远方，传来侠客的吟唱：“凌波逐朔风，步云踏飞鸿。彩云应日起，执剑对长空。”

夕阳渐下，少年良辰静静地注视着侠客远去的身影缓缓消逝在了无垠的地平线下。四周万籁俱寂，空余少年良辰孑然一人，伴着他的，唯有一影、一剑、一卷轴而已。



Chapter 2

第 2 章

纵观全局——布局探索

暮秋岚风残月起，叶落肩头影相顾。男儿未遂平生志，蓝卷银锤绘新说。

《孙子兵法》有云：知己知彼，百战不殆，故了解自己手中的法宝，是每一位苹果帮众的必经之路。正所谓人剑合一，试想，若一个武者不能掌握其手中之器，那么在战场上，何谈杀敌？不把自己击伤，就可要谢天谢地咯！诸君以为然也？

于是，与其师短暂分别之后，良辰便开始仔细探索起法宝的构造来，先囫囵个大概，做到知其然。随着良辰的学习不断深入，法宝与他的联系开始逐渐强了起来……

2.1 工作区

Xcode 的工作区是用来执行核心开发任务的区域，它是创建和管理项目的主要界面。项目（Project）是 Xcode 开发的主要单元，包含了构建产品所需的所有元素（源代码文件、资源文件等）、框架、插件等，并且工作区还用以维护这些元素之间的关系。

Xcode 的工作区可以根据其中项目的特性而自动适应，根据选择文件、对象的不同而打开不同的窗口。当然，Xcode 还提供了自定义工作区风格的功能，以便工作区能够更加符合开发者的习惯。有关自定义工作区的相关信息，请参阅本书附录 A 中的 A.3 节“Xcode 设置”。

打开项目（Project），就会立即弹出工作区的窗口，其界面基本上如图 2-1 所示。

工作区的窗口界面中央是编辑器区域，通常情况下的代码编写都是在这个区域完成的。一旦选中了项目中的一个文件，其内容便会出现在编辑器区域。Xcode 会自动识别文件类型，并使用相应的编辑器来打开这个文件。

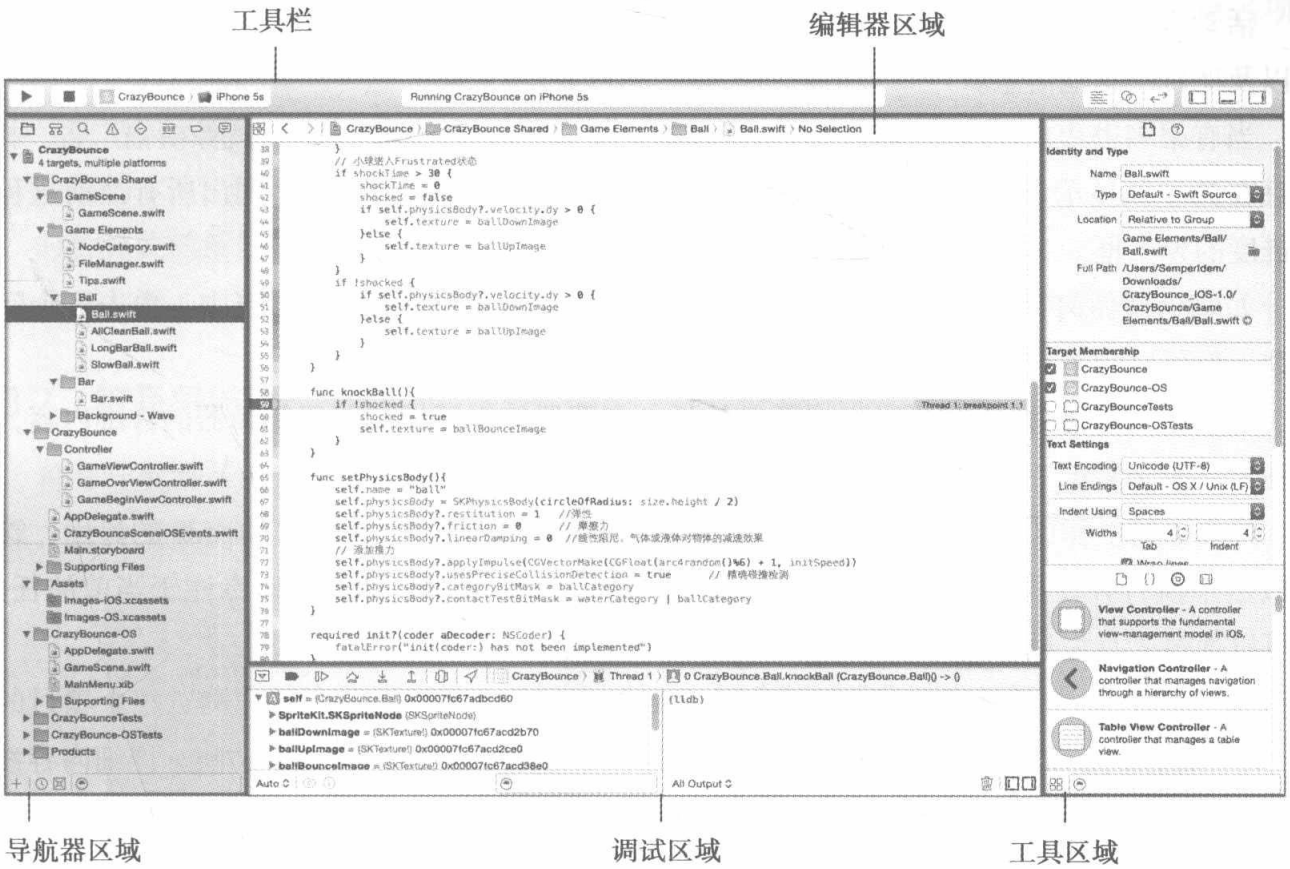


图 2-1　Xcode 工作区

比如说图 2-1 中，在工作区窗口的左侧导航器区域中，选中了一个 Swift 代码文件，中间的编辑器区域显示的就是 Ball.swift 文件的内容。与其他区域不同的是，编辑器区域可以出现多个，甚至可以单独以一个窗口的形式出现。

2.2　工具栏

工作区窗口顶部的工具栏可以让你快速访问频繁使用的命令，如图 2-2 所示。运行按钮可以直接编译和运行你的产品。停止按钮可以立即中止运行。方案选择菜单可以让你选择想要创建和运行的产品，关于方案选择菜单的相关介绍，请参考本书的 10.1 节“编译方案”。

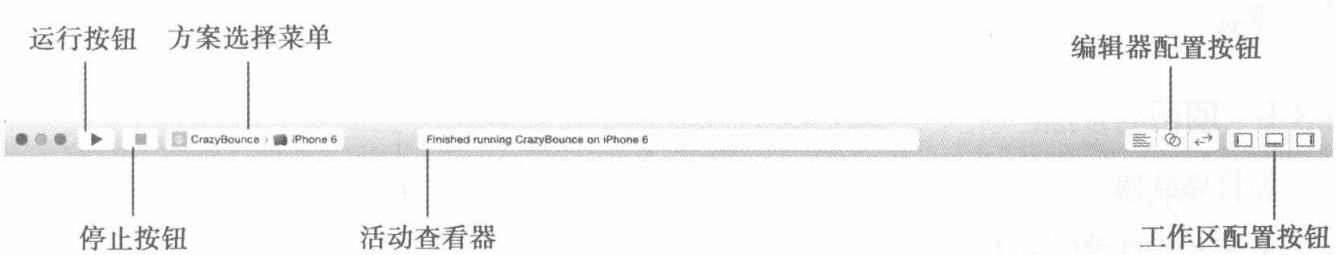


图 2-2　Xcode 工具栏

活动查看器位于工具栏正中央，通过展示状态信息来展示当前任务执行的进程、编译进程以及项目的其他信息。

当有很多进程在同步进行时，活动查看器会在这些进程中轮流切换（就像横幅广告一样），并且在其左侧显示出并行任务的数量。单击这个数字，就可以显示一个列出所有当前进程及其进度的弹出框。

如果出现任何错误或者警告，问题的数量也将会显示在活动查看器当中。单击这个问题计数，就可以切换到问题导航器，以便查找和修改问题。

“编辑器配置按钮”则可以根据既定的任务来配置编辑区域。关于编辑器的详细内容，请参阅本书的 2.5 节“编辑器区域”。

Xcode 的工作区被横向分为了 3 个主要的区域，它们肩并肩组成了 Xcode 的界面。这些区域用来执行开发周期中不同的任务，隐藏不用的区域可以让你更好地专注于当前的任务。你可以通过工具栏最右端的“工作区配置按钮”来隐藏或者展示这些区域。

- **显示 / 隐藏导航器区域**（navigator area）：你可以在导航器区域中浏览整个项目的内容，包括其中的文件、断点、测试报告等，也可以执行一些搜索功能。详细内容参见本章 2.3 节“导航器区域”。
- **显示 / 隐藏调试区域**（debug area）：你可以在调试区域查看正在运行项目中存在的变量、使用控制台与调试终端进行交互以及控制应用程序的执行。详细内容参见本章 2.6 节“调试区域”。
- **显示 / 隐藏工具区域**（utilities area）：你可以在工具区域检查或者更改元素的属性，也可以访问现成的资源库。详细内容参见本章 2.7 节“工具区域”。

使用菜单栏的 View → Hide/Show Toolbar 可以显示和隐藏工具栏。

2.3 导航器区域

导航器区域可以用来访问项目中的文件、符号、单元测试、测试诊断以及其他方面的内容，总而言之，项目中绝大多数元素都存放在此处。通过在导航器顶端的“导航栏”，选择当前所需的导航器，以执行所需的操作和任务。

导航栏中，可以选择 8 种导航器，下面分别介绍。

2.3.1 项目导航器

项目导航器（Project navigator）可以用来检索项目的源代码和资源文件，它统一显示了项目中文件夹的目录层级关系以及元素名称，如图 2-3 所示。

项目导航器是开发过程中最常用的导航器，在其中选中某个元素，就会在编辑器区域中

用合适的编辑器将其打开、显示出来。与 Finder 类似，开发者可以添加、删除和重命名项目导航器当中的资源。有关项目管理的相关内容，请参阅 3.1 节“文件管理”。

导航器底部有一个过滤器栏，可以让开发者过滤项目列表中的元素。

左边的“+”按钮用来在项目中添加新的文件、项目和资源。

时钟样式的按钮将只显示最近修改过的文件。

小正方形（10.10 以前的版本中这更像一个小抽屉）按钮将只显示能使用版本管理的文件。

2.3.2　符号导航器

符号导航器（Symbol navigator）用于显示代码中的类、对象、函数、变量、属性等元素信息，这些元素统称为“符号”。符号导航器如图 2-4 所示。

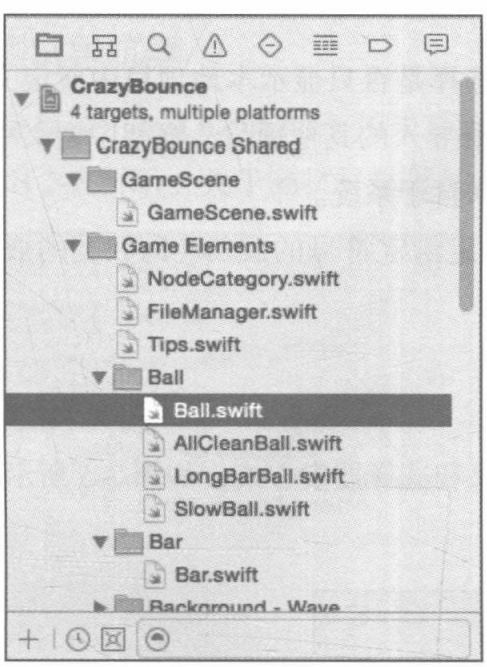


图 2-3　项目导航器

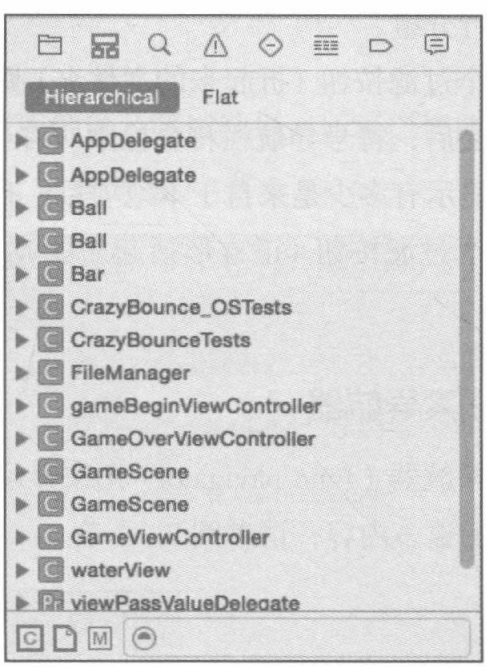


图 2-4　符号导航器

开发者可以更改符号导航器的显示方式，包括分层（Hierar-chical）显示和平坦（Flat）显示。分层显示将会按照类的层级来显示符号，子类将会包含在父类当中；而平坦显示则会将全部类显示出来，无所谓类的层级。

符号导航器中所显示出来的符号前面都有一个特殊的图标，用于表示符号元素的种类，如表 2-1 所示。

表 2-1　符号元素的图标

图　标	符　号	图　标	符　号
C	类（Class）	f	结构体（Struct）

（续）

图　标	符　号	图　标	符　号
Pr	协议（Protocol）	S	联合体（Union）
U	函数（Function）	K	类型定义（Typedef）
E	全局变量（Global Variable）	M	方法（Method）
T	枚举（Enum）	P	属性（Property）
V	枚举成员（Case）		

导航器底部的过滤器导航栏提供了多个列表过滤选项。

第一个过滤按钮（正方形括起来的 C）用以选择是只显示类和方法，还是显示全部类型的符号（比如协议、结构、枚举等）。如果选择显示全部类型的符号的话，符号导航栏将会按照类型来进行分组。

第二个过滤按钮（折起来的文件夹）则是用以选择是否只显示本地项目中所定义的符号。取消选中之后，符号导航栏将会检索所有当前项目所导入的其他项目、框架中定义中的符号，然后将会显示有多少是来自于本地项目，有多少是来自于系统。

第三个过滤按钮（正方形括起来的 M）用来指定给定符号的成员（比如类的成员）是否显示。

2.3.3 搜索导航器

搜索导航器（Find navigator）用来显示搜索结果和选择搜索方式，如图 2-5 所示。有关搜索导航器的更多内容，请参阅 7.7.2 节。

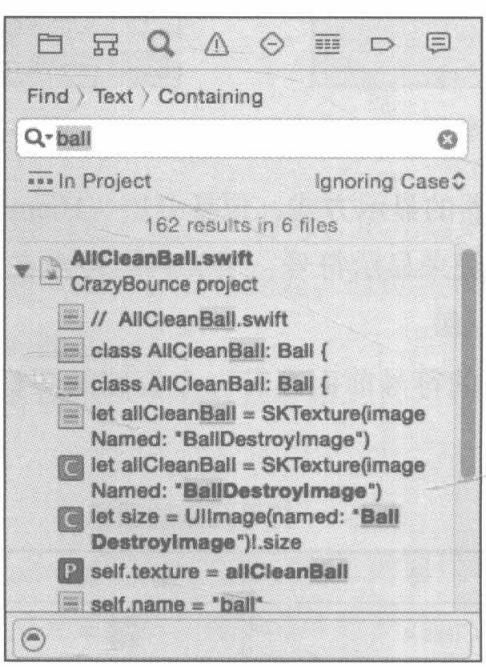


图 2-5　搜索导航器

2.3.4　事件导航器

事件导航器（Issue navigator）用来显示在工作区找到的任何“事件”，诸如编译错误、语法错误、库链接异常，以及错误提示等信息，如图 2-6 所示。通过这个导航器可以快速有效地查看出现问题的文件对象，方便开发者进行更正和修改。

事件导航器可以选择将事件按照文件分类（By File），还可以选择按照问题类型分类（By Type）。

底部的过滤器导航栏提供了多个列表过滤选项。

第一个过滤按钮（时钟）用来选择是否只显示最后一次编译中所发现的问题，也就是只显示“新”问题。当然，仅限于警告。

第二个过滤按钮（惊叹号）用来选择是否只显示错误信息，而隐藏警告的出现。

2.3.5　测试导航器

测试导航器（Test navigator）用来显示单元测试用例以及测试结果，还可以快速执行单元测试，如图 2-7 所示。关于测试导航器的相关内容，请查阅第 12 章。

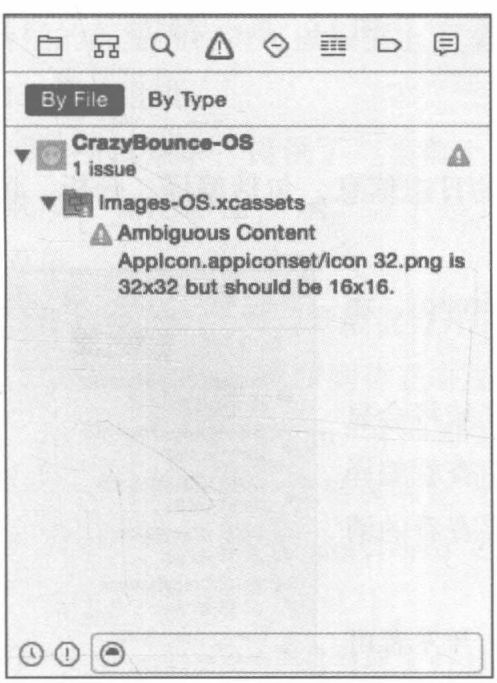


图 2-6　事件导航器

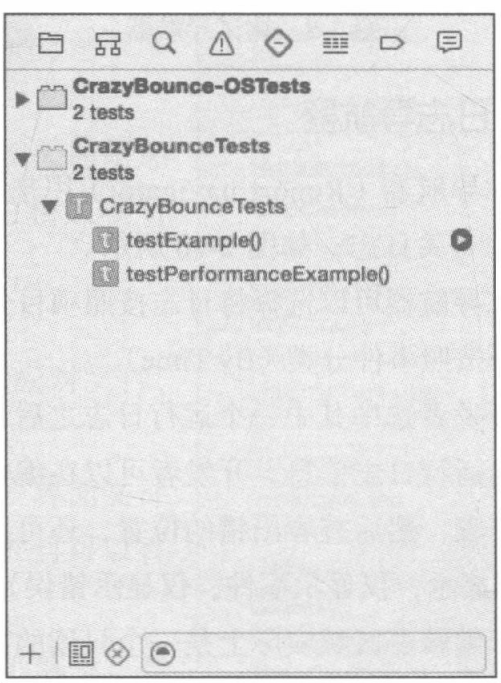


图 2-7　测试导航器

2.3.6　调试导航器

调试导航器（Debug navigator）用来显示应用程序在调试状态下的资源占用状态以及堆栈信息，如图 2-8 所示。关于调试导航器的相关内容，请查阅本书的第 11 章“谨防走火入魔——调试”。

2.3.7 断点导航器

断点导航器（Breakpoint navigator）用来显示应用程序中所标记的所有断点信息，如图 2-9 所示。关于断点导航器的相关内容，请查阅第 11 章。

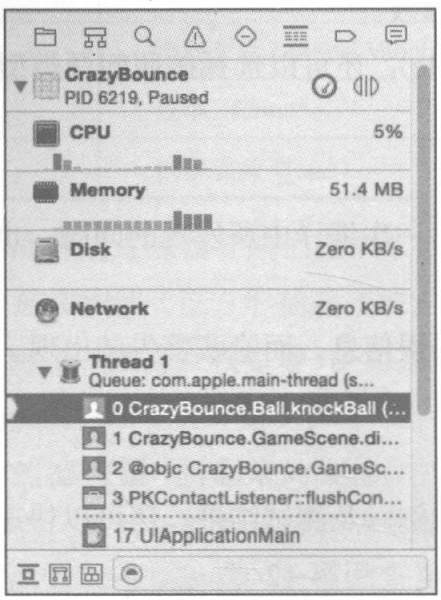


图 2-8 调试导航器

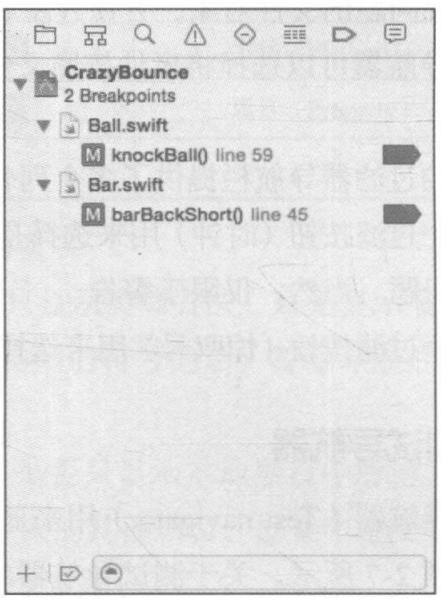


图 2-9 断点导航器

2.3.8 日志导航器

日志导航器（Report navigator）用来显示所有的日志信息，包括编译、分析、测试和调试方面的有关日志，如图 2-10 所示。

日志导航器可以选择将日志按照项目分类（By Group），还可以选择按照事件分类（By Time）。

当开发者选中其中一个运行日志之后，编辑器区域将会显示对应的运行日志信息。开发者可以在编辑器区域中查看编译的过程步骤，然后查看出错的位置，还可以过滤想要查看的消息（全部显示，仅显示事件，仅显示错误）。

这个编辑器区域实际上是一个只读的文本区域，开发者可以按行复制其中的内容。

还可以选择消息右侧的列表图标，显示消息相关的命令和输出。

底部的过滤器导航栏提供了多个列表过滤选项和操作选项。

第一个小齿轮图标是操作按钮，用来创建“机关”。

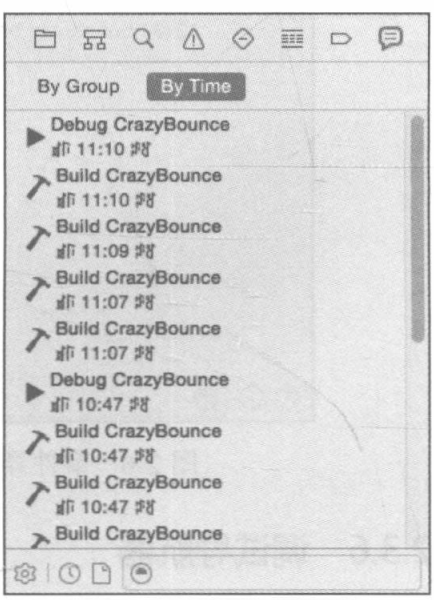


图 2-10 日志导航器

第二个时钟图标是过滤按钮，用来控制是否只显示最后一次操作之后所产生的日志。

第三个文件图标是过滤按钮，用来选择是否只显示连续集成操作的日志信息。

2.4　跳转栏

在每个编辑器以及辅助编辑器面板顶部都包含有一个跳转栏，这是一个交互式的分层机制，用以显示当前编辑器在项目组织结构中所处的位置，还可以直接跳转至项目中任意层次结构中的某个项目，如图 2-11 所示。

图 2-11　Xcode 跳转栏

跳转栏可以作为项目导航器的一个替代品，在项目导航器被隐藏的时候，可以用它来代替项目导航器来切换不同的文件。

跳转栏的配置和行为可以根据上下文环境来定制，基本的跳转栏由三部分组成。

- 相关项目菜单：提供了与当前上下文环境相关的附加选项，如图 2-12 所示。以下是几个常用的相关项目菜单。
  - Recent Files：跳转到最近打开的文件。
  - Counterparts：跳转到你正在编辑文件的配对文件，通常指的是执行（.m）文件和头 (.h) 文件。
  - User Interfaces：跳转到绑定当前类的用户界面文件当中，比如 xib 和 storyboard 文件当中，并且可以直接定位到该场景。
  - Preprocess：跳转到当前文件预处理之前的状态，通过这个选项可以查看在测试 / 编译 / 运行 / 分发之前，该文件所完整编译的状态。
  - Assembly：跳转到当前文件处理之后的状态，该文件会被编译成 LLVM 能够编译运行的汇编语言级别的“装配流水线”。
  - Disassembly：跳转到当前文件处理之前的状态，与 Assembly 配套使用，必须在助理编辑器打开，且应

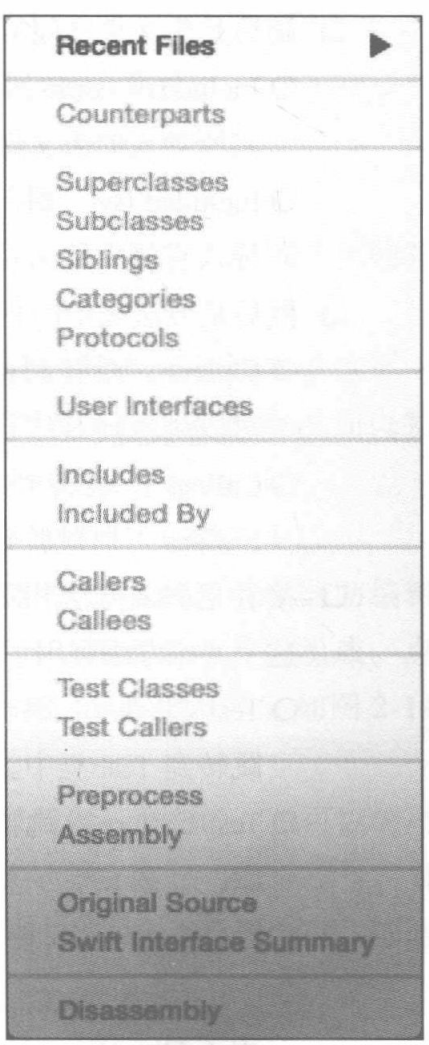


图 2-12　相关项目菜单

用运行并出于调试暂停状态的时候才能显示。

- 以下几个类级别的选项可以帮助我们更好地分析类的层级，并且在父类、子类等当中跳转。假设光标现在停留在了一个继承自 UIViewController 的类 View-Controller 当中。
  - Superclasses：跳转到当前类的父类，Superclasses 将可以把我们引导到 UIView-Controller 类当中，此外，还可以直接上溯到最终父类 NSObject。
  - Subclasses：跳转到当前类的子类，比如在 UIViewController 类中可以跳转到 ViewController 当中。
  - Siblings：跳转到当前类的同类，即继承了同一个父类的对象。使用 siblings 可以把我们引导到 UIAlertController 等多个同样继承 UIViewController 的类当中。
  - Categories：跳转到当前的扩展，如果当前这个类被 Category 扩展了功能，那么这个功能可以引导我们前往这个 Category。
  - Protocols：跳转到当前类所实现的协议，比如可以跳转到 UIViewController 所继承的 NSCoding 协议当中。
- 随后是头文件之间的包含关系，比如当前文件 A.m 导入了一个头文件 A.h：
  - Includes：跳转到当前文件所导入的头文件定义当中，也就是在 A.m 中可以跳转到其所导入的头文件 A.h 中。
  - Included By：跳转到导入当前头文件的文件定义当中，也就是在 A.h 中可以跳转到导入它的文件 A.m 中。
- 然后是方法之中的相互调用关系，比如 B 方法中调用了 A 方法：
  - Callers：跳转到调用当前方法的方法定义当中，也就是在 A 方法中，使用 Callers 可以跳转到 B 中。
  - Callees：跳转到当前方法所调用的方法定义当中，也就是在 B 方法中，使用 Callees 可以跳转到 A 中。
- 接着是测试方法和测试类之间的相互调用关系，比如说在 Test 类中有一个 test 方法，这个 test 方法调用了一个 check 方法。
  - Test Classes：跳转到引用当前测试方法的测试类当中，也就是在 check 方法中可以跳转到 Test 类中。
  - Test Callers：跳转到调用当前测试方法的测试方法当中，也就是在 check 方法中可以跳转到 test 方法中。
- 然后是 Swift 项目特有的选项：
  - Original Source：这个选项只能在 Objective-C 和 Swift 共用的项目当中使用，在调用了 Objective-C 方法的 Swift 方法当中，这个选项可以跳转到这个 Objective-C 方

法当中。

- 对于 xib 文件来说，它的相关项目菜单也有所不同，如下所示，见图 2-13。
  - Automatic：这个选项用来跳转到当前 nib 文件所绑定的类当中。
  - Top Level Objects ：顶层对象是没有父类的对象集合，通常情况下包括添加到 nib 文件中的窗体、菜单栏以及自定义的对象。
  - Localizations：这个选项用来跳转到当前 nib 文件所对应的本地化版本文件当中，这个选项要开启了本地化功能才能使用。
  - Notification Payloads ：这个选项用来跳转到当前 Storyboard 文件的通知载体控件上，需要开启 Notification 功能才能使用。

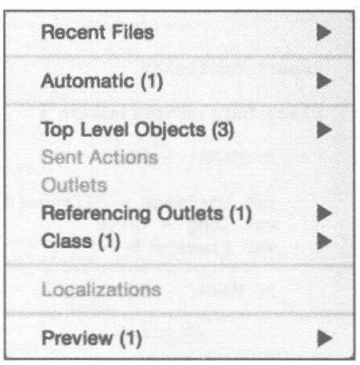


图 2-13　相关项目菜单 –Xib 文件

  - Preview：结合助理编辑器使用，可以查看 nib 文件在不同尺寸设备上的外观。
  - Sent Actions：查看当前 xib 文件所绑定的 IBAction 方法。
  - Outlet：查看当前 xib 文件所绑定的 IBOutlet 属性。
  - Referencing Outlets：查看当前 xib 文件所绑定的 IBOutlet 集合属性。
  - Class：查看当前 xib 文件所绑定的类。
- 对于 Storyboard 文件来说，相关项目菜单则是：
  - Timeline ：如果 Storyboard 使用了时间线功能，使用这个选项可以跳转到不同的时间线。

当然，对于不同的文件、不同的编辑器、不同的平台来说，其相关项目菜单也不一样，对于本书没有介绍到的项目，读者可以自行查阅相关资料和文档，在此就不加以赘述了。

- 后退 / 前进按钮：用来在导航历史中查看上一个或者下一个文件（见图 2-11 ）。
- 分层路径菜单：可以通过这个菜单跳转至一个新项目来更改编辑器或者辅助编辑器面板中展示的内容。根据你点击的路径，它可以由一个或者多个分段组成。点击分层路径菜单中的某个分段（segment）可看到相关项目的弹出菜单，如图 2-14 所示。

比如，你可以在项目中使用跳转栏导航至某个文件或者打开任何一个文件，也可以使用跳转栏打开文件夹内的文件，还可以在当前打开的文件中使用跳转栏来展示并选中符号。对于界面构造器文件，可以查看用户界面的层次结果。如果 Xcode 在工作区中发现问题，问题标记会出现在跳转栏的右边。

图 2-14 分层路径菜单

2.5 编辑器区域

Xcode 中大部分编辑工作都是在编辑器区域进行的，编辑器是动态出现的，适合于当前选择的文件，还可以来回切换编辑器，编辑器有很多种，例如：

- 源码编辑器（Source editor）：用以编写源代码。
- 界面生成器（Interface Builder）：用以图形化地创建和编辑用户界面文件。
- 项目编辑器（Project editor）：用以查看和编辑应用程序的编译方式，比如指定编译选项、目标架构以及相关信息。
- 数据模型编辑器（Core Data Model editor）：用以编辑 Core Data 相关的数据模型。

除了上面介绍的编辑器之外，Xcode 还含有很多不同形式的编辑器，其他的编辑器会在今后的章节中有所提及。使用工具栏（参见图 2-2）右侧的编辑器配置按钮，可以根据相应的需求来配置编辑器区域。下面分别介绍几个编辑器配置按钮。

2.5.1 标准编辑器

标准编辑器（Standard editor）显示方式使用选中文件的内容填充编辑区，这种显示方式是大多数时候所使用的编辑器区域显示方式，这时只能显示唯一一个编辑器区域，以便让开发者能够专注于使用当前的编辑器。

2.5.2 辅助编辑器

辅助编辑器（Assistant editor）显示方式用以在标准编辑器区域中展示一个在逻辑上与原内容匹配的单独的编辑器面板，如图 2-15 所示。

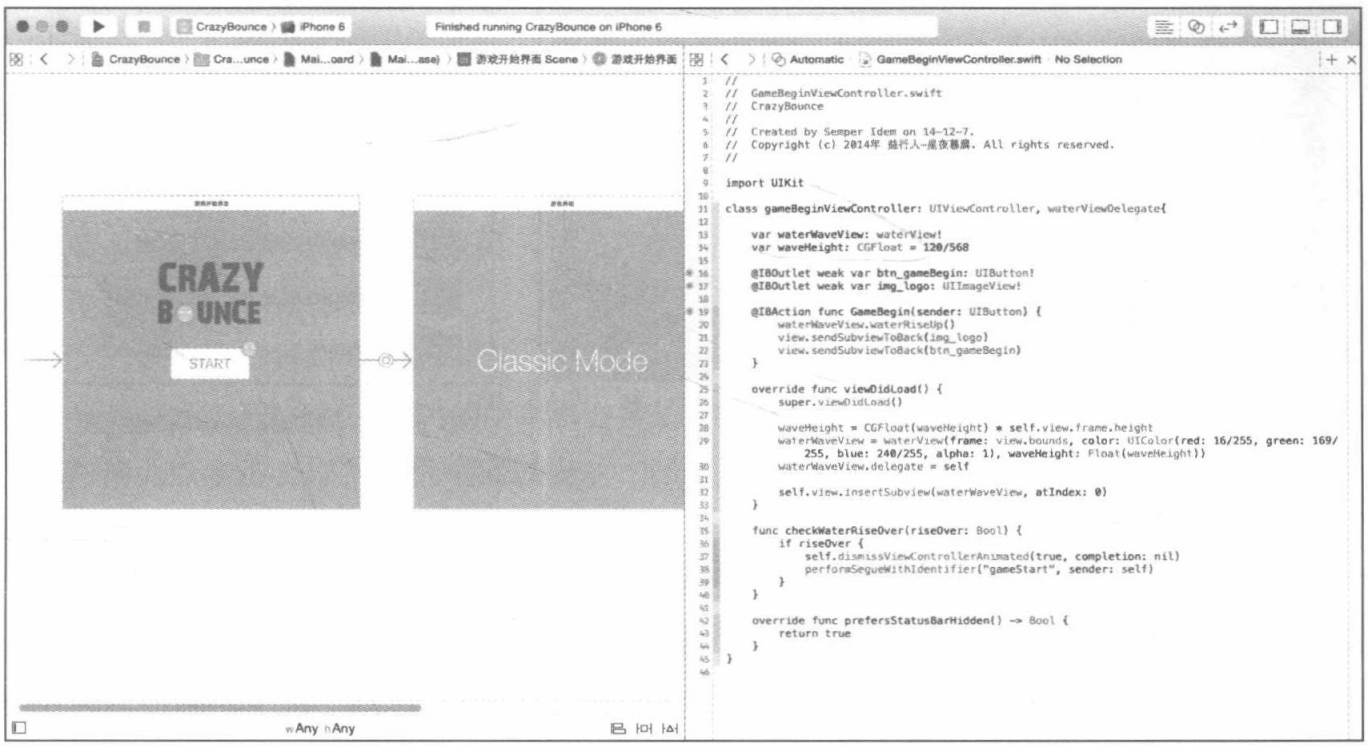


图 2-15　辅助编辑器

换句话说，辅助编辑器执行的是“拆分窗格”的功能，能够帮助开发者在多个文件中执行一些常见的辅助任务。比如，开发者可以在界面生成器中，将控件和代码连接起来（参见 7.1 节），从而建立动作和输出口链接。

默认情况下，当初始打开辅助编辑器的时候，它执行“匹配”的行为模式是“手动”（Manual），在这个模式中，开发者可以自行选择辅助窗格所显示的内容。当然，开发者还可以使用跳转栏的相关项目菜单，选择合适的编辑器行为模式。

开发者还可以在现有的辅助编辑器的右上角点击“添加”或者“删除”按钮来添加或者移除附加的辅助编辑器。

新添加的辅助编辑器具备自己独立的跳转栏，然后其出现的位置是点击添加按钮的辅助编辑器之后。注意，原先的标准编辑器不具备添加和删除编辑器的功能，也就是说，一般情况下编辑器区域都始终含有一个编辑器存在。

可以用快捷键来打开辅助编辑器。默认情况下，在标准编辑器显示模式下，按下 Option 键并选中项目导航器当中的文件，将会在辅助编辑器中显示选中的文件。而如果是在辅助编辑器视图下使用该快捷键，则是切换辅助编辑器中显示的内容。而如果存在不止一个辅助编辑器，或者按下 Option+Shift 键并选中文件的时候，那么 Xcode 就会弹出一个直观的位置选择器，来询问开发者要在哪个编辑器中显示这个文件，如图 2-16 所示。

此外，开发者还可以更改辅助编辑器的布局，通过标题栏上的 View → Assistant Editor 菜单，可以查看和选择可用的辅助编辑器布局模式，如图 2-17 所示。

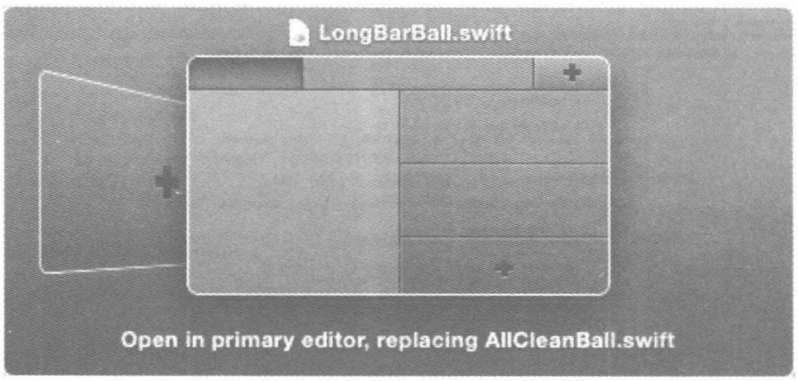


图 2-16 位置选择器

图 2-17 辅助编辑器布局模式

2.5.3 版本编辑器

版本编辑器（Version editor）在一个面板中展示当前选中文件，在第二个面板中展示该文件的另一个版本，以便比较两者之间的差异，如图 2-18 所示。只有当你的工程支持源码控制的时候，该编辑器才起作用。

版本编辑器有三种显示模式，分别如下：

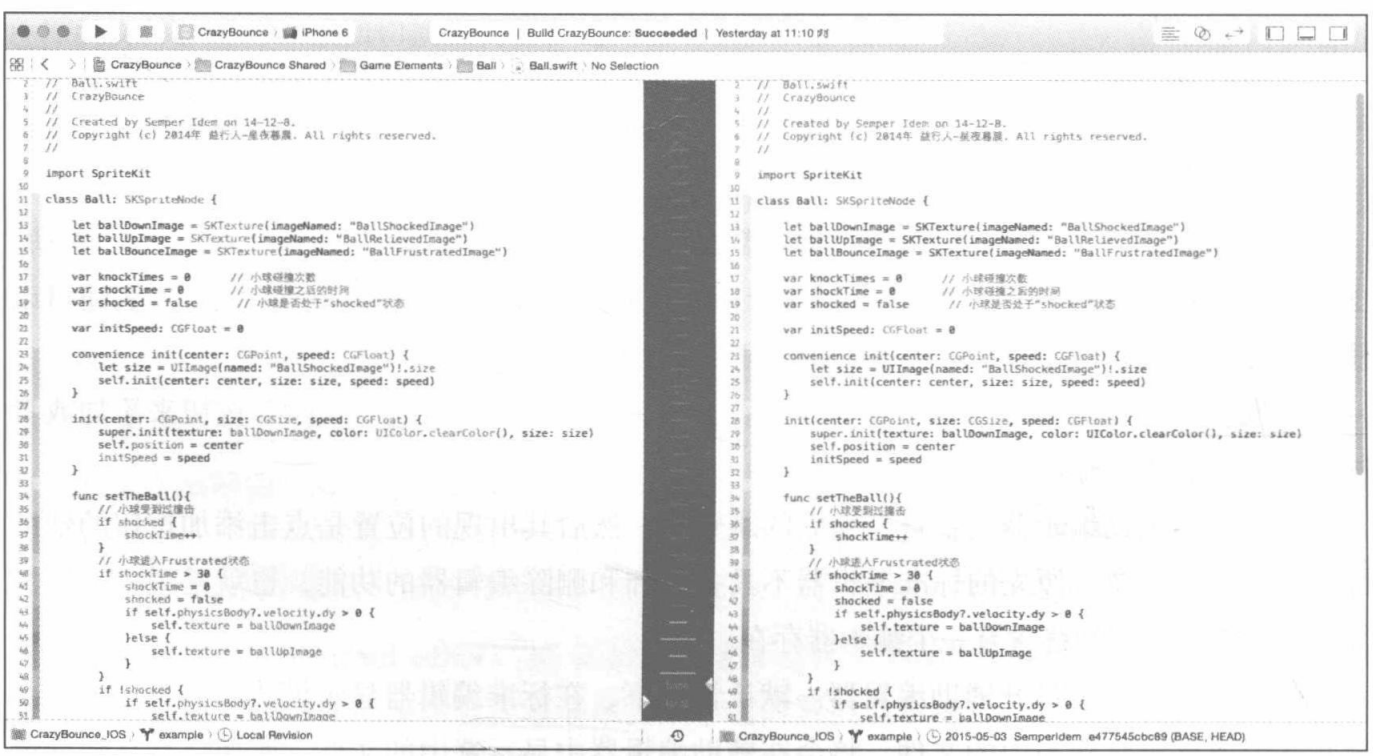


图 2-18 版本编辑器

比较模式（Comparison）是最常用也是默认的显示模式，通过这个模式可以进行不同版本之间的代码比较工作。

责任人模式（Blame）是在第二个面板中显示提交信息和修改代码的对应，通过这个功能可以有效地寻找到提交优质或者劣质代码的人员，从而做出相应的反应。

日志模式（Log）是在第二个面板中显示所有有记录的版本的提交信息。

比较模式的版本编辑器有一个很有用的功能：时间线（Timeline），通过时间线功能，我们可以回溯到有记录的时间段，选择相应的版本，来对代码之间进行比较。

时间线功能位于版本编辑器正下方，以一个小小的钟表形式显现。点击这个按钮，就会在版本编辑器的中央分隔区域显示一条黑色的时间线。通过将鼠标在时间线上下滑动，就可以浏览所有可用的版本。

时间线是按照时间顺序排列的，越新的版本就越靠下面。灰白色长实线则是将可用版本进行分组，更新不超过 24 小时的版本将会被长实线分隔为一组。

可用的版本将会以灰白色短实线表示，鼠标移到实线上面，会显示出版本的时间、描述等信息，如图 2-19 所示。

当你找到所需的版本之后，单击这个版本就可以在相应的版本编辑器中显示对应版本的代码了。

关闭时间线，就可以对比各版本之间的不同了。不同的区域用与背景色相似的颜色框起来，并且在中间予以连接和显示，中间的按钮还会显示不同之处的序号。点击序号可以选择“Discard changes”，以恢复原来版本的配置情况。

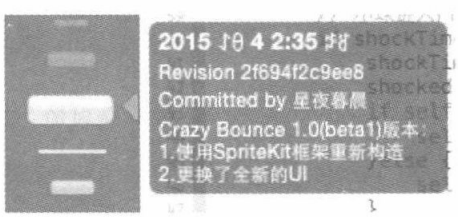


图 2-19　版本信息描述

2.6　调试区域

当应用运行的时候，Xcode 会开启调试功能。你可以使用图形化的工具直接在代码编辑器中调试应用，这时候，一些调试功能都会在调试区域中进行，调试区域如图 2-20 所示。

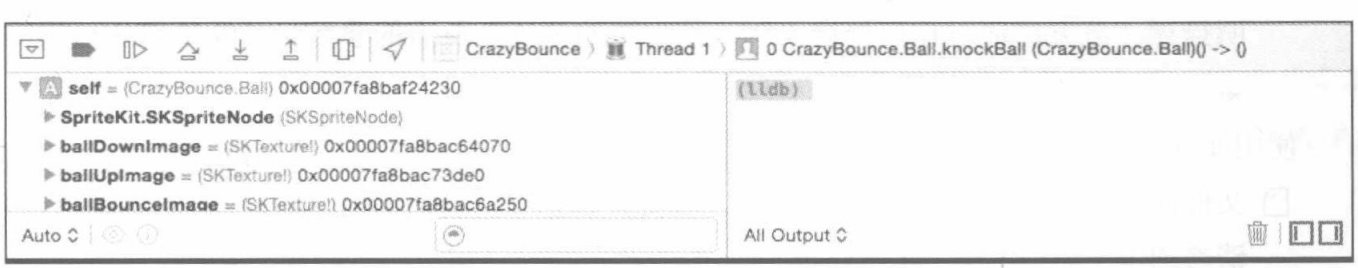


图 2-20　调试区域

调试区域可以用来控制程序的执行，比如执行断点、位置模拟等，调试区域还可以用来输出 NSLog、println 等命令控制行输出语句。

关于调试区域的更多内容，会在第 11 章进行更为详细的介绍。

2.7 工具区域

工具区域位于工作区窗口（见图 2-1）的最右边，你可以通过它快速访问很多资源如图 2-21 所示，下面分别介绍：

- 检查器（inspectors），用以查看和更改编辑器中选中元素的属性和相关特性。
- 现有的资源库，用来方便快捷地添加文件、代码片段、控件以及资源文件等。

与编辑器区域一样，工具区域也是会根据环境的不同而进行相应的变化。通常情况下，你通常会在检查器导航栏上看见两个检查器：文件检查器和快速帮助检查器。下面简介一下这两个检查器。

- 文件检查器（File inspector）：用以查看和管理选中文件的元数据。尤其是要进行本地化故事板和其他媒体文件，并更改用户界面文件设置操作的时候。

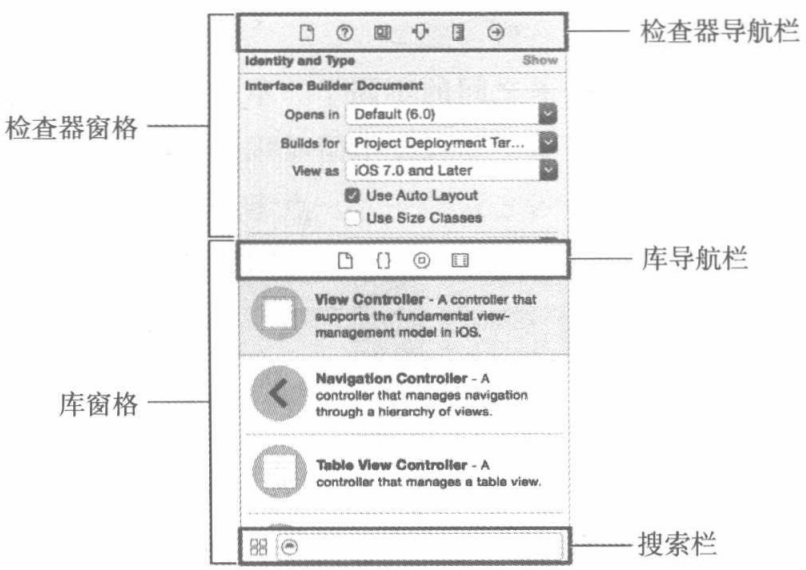


图 2-21 工具区域

- 快速帮助检查器（Quick Help inspector）：用以查看元素的帮助信息。比如对方法的简明描述，该方法何时在何处被声明，该方法的范围、所需的参数、适用平台和可用的架构等。

使用库导航栏则可以访问现有的资源库，资源库有如下一些：

- 文件模板（File templates）：拥有默认代码结构的文件模板，关于文件模板的更多内容，请参阅本书附录 C.1“文件模板”。
- {} 代码片段（Code snippets）：用于存储经常用到的源代码片段，比如说类声明、闭包等等。关于代码片段的更多内容，请参阅本书附录 A.2“代码片段”。
- 对象（Objects）：应用的用户界面的控件库，有关控件对象的更多内容，请参阅本书附录 C.4“控件模板”。
- 媒体（Media）：包含图形、图标、声音文件以及诸如此类的文件。

想要在你的项目中使用现有的库，可将其直接拖拽至适当的区域。比如，你要使用代码片段，可将其从库中直接拖放到源码编辑器中；想要使用文件模板创建一个源码文件，可将其模板拖放至项目导航器中。

想要限制某个库的项目，可在搜索栏的文本域输入相关的文本进行限制。比如，输入“button”便可展示对象库中的所有按钮。

2.8　标签页

Xcode 和 Safari 一样，可以使用“标签页”来在同一个工作区窗口中，打开多个不同的项目文件，如图 2-22 所示。

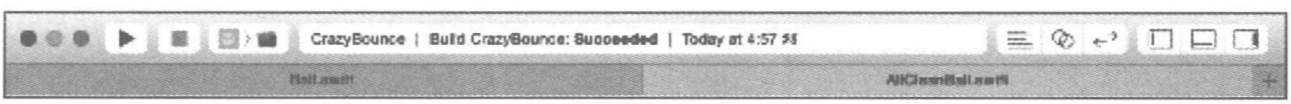


图 2-22　标签页

要创建一个新的标签页，从标题栏中依次选择 File → New → Tab，即可完成创建。开发者可以随意拖动标签页以重新排定顺序。也可以双击标签页的标题，来修改当前文件的名称。要关闭标签页，单击标签页上的关闭按钮即可，或者从标题栏中选择 File → Close Tab 完成关闭操作。

提示　可以使用快捷键来快速对标签页进行操作。比如，“Command + T”创建一个新的选项卡，“Command + W”关闭当前选项卡，“Command + { 或 }”用于切换前一个 / 后一个选项卡。

所谓“读书破万卷，下笔如有神”，对于法宝、武功等都是同样的道理。只有掌握了其基本用法，打好基础，方能直上青云，修为小成。

如此道理，无论对于修道、修学，还是技巧来说，都是一样的。熟能生巧，古有老翁滴油穿币，今有巧工善事利器。正可谓：闻百徵，操千曲，晓声引雀音绕梁。闻鸡舞，凿壁光，业精于勤无人当。

Chapter 3

第 3 章

藏经阁——项目管理

诸君都知晓，要想实现“万丈高楼平地起”的效果，那么好的架构是必不可少的。就拿咱所在的这座酒馆来说吧，可谓是：雨伴轻烟锦旌扬，笑随酒香日方长。雕栏玉砌坐席边，名家宣墨金壁上。花吹落风杨柳外，水洒粼光溪涧旁。何故那般众生相，且乐且歌入醉乡。

对于良辰来说，苹果帮所提供的那个法宝，也提供了一个管理材料的地方，那就是法宝中自带的“藏经阁”，一个神秘的储物空间。

因此，少年便要前往神秘而复杂的藏经阁当中，去探秘其中的构造，并且学习如何使用、习得这门管理的功夫，这样才能够开始造物的制作。

图 3-1　藏经阁

Project（项目）在 Xcode 中是用来组织应用程序所必需的文件和资源的基本单位，第 1 章已经介绍过如何创建新的项目。Xcode 为 iOS 和 Mac 应用程序提供了许多常用的项目模板。

项目中不仅包含了搭建多个应用所需的元素，还维护着这些元素之间的关系。这些元素包括：

- 源代码文件、库和框架、图片文件和用户界面文件。
- 在项目导航器中负责组织文件架构的分组（Group）。
- 项目级别的应用编译配置。
- 对应单个应用的对象（Target）。

因此，如何更好地管理项目中的这些元素和关系也是开发中的一个重要组成部分。简而言之，“项目管理”基本可以分为“文件管理”、“对象管理”以及“资源管理”。

3.1　文件管理

除了最简单的应用程序之外，基本上大多数应用程序都可能拥有许多源代码文件、资源文件以及框架或者静态库文件等。源代码文件一般都会被编译为二进制文件并链接到一起，而资源文件则都会被拷贝到项目的包文件当中进行管理。合理地管理文件资源是一个良好的编程习惯。

3.1.1　创建文件

向 Xcode 项目中添加文件和资源有多种方法。如果要创建全新文件，则可以使用“New File”功能，借助 Xcode 提供的文件模板来创建文件，也可以直接使用文件模板库创建新文件。对于现有的文件来说，可以使用 Add Files to“项目名”功能，也可以直接将文件拖放进项目导航器当中。

由于 Xcode 更为重视文件路径的作用，因此完全可以打开两个同名但是不在同一位置的文件进行编辑，Xcode 并不会混淆它们，但是并不推荐大家这样做，因为你本身很可能会自己混淆。总之，在将创建文件的时候，一定要重视项目路径的选定。一经选定，更改起来就比较麻烦。

3.1.1.1　创建全新文件

Xcode 当中包含了创建各种常见文件类型的模板以及将其添加到项目当中的选项。要创建全新文件，可以从菜单栏选择 File → New File，也可以在项目图标上右键选择“New File…”功能，还可以直接使用“Command + N”快捷键，如图 3-2 所示。

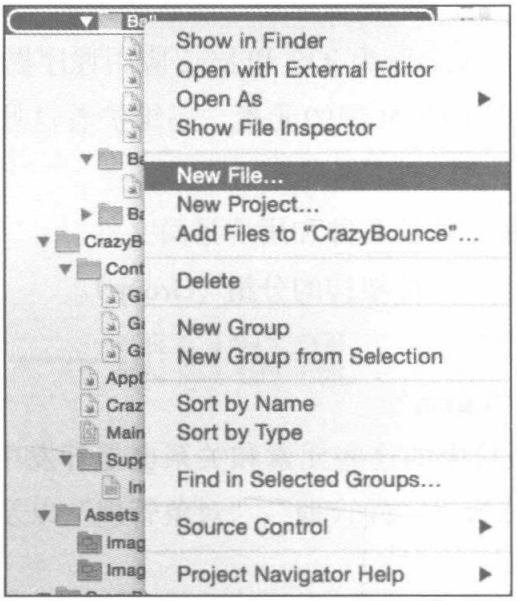


图 3-2 新建文件选项

选中之后，Xcode 将会弹出一个和之前“项目模板”选择窗口很类似的“文件模板”选择窗口，如图 3-3 所示。

左侧窗格包含了可用文件模板的全部分类，从当中可以选择相应平台的不同文件类型。右侧窗格包含了相应类别的文件模板，从当中可以选择你想要的文件模板。窗口的右下区域则是对该文件模板的相关说明，具体的文件模板说明请参阅本书的附录 C 中的 C.1 节。

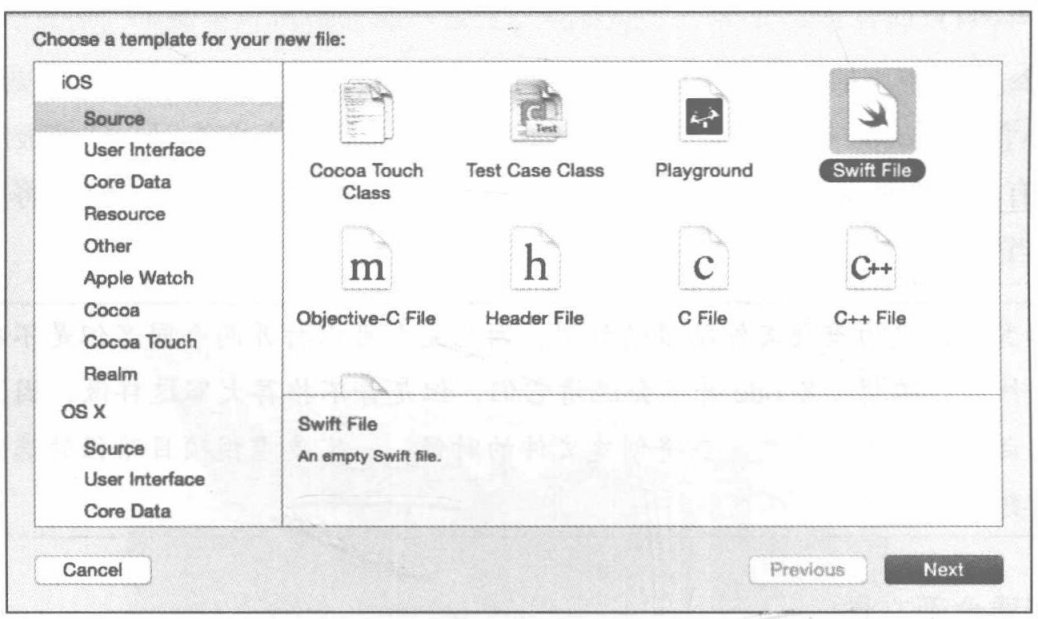


图 3-3 新建文件模板

单击“Next”按钮，在弹出的界面中，选择存储文件的路径，如图 3-4 所示。可以设置这个新文件所在的组及其需对应的对象。单击“Create”按钮即可完成新文件的创建。

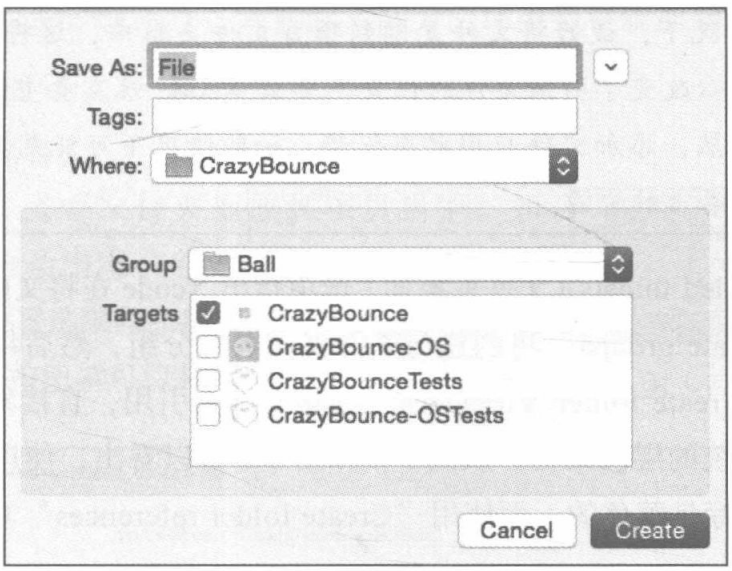


图 3-4　选择文件存储路径

3.1.1.2　使用文件模板库

上一章介绍过，Xcode 的工具区域中拥有一个 File Template Library（文件模板库），如图 3-5 所示。使用文件模板库可以简单地创建新文件，只需将所需要的模板直接拖入到项目导航器相应位置即可，然后 Xcode 将会弹出一个类似于图 3-3 的窗口，接下来就可以在窗口中命名该文件。指定组和对象了。

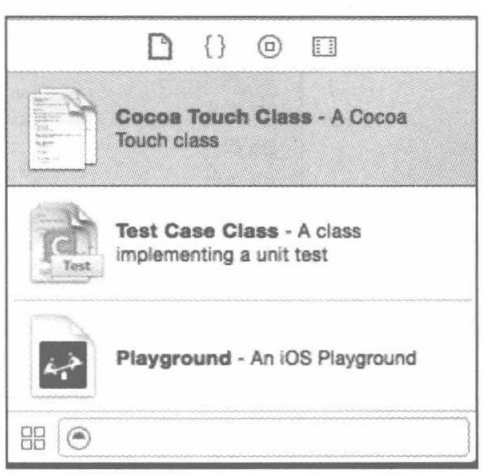


图 3-5　文件模板库

3.1.1.3　添加文件到项目

如果要添加现有的文件，从菜单栏选择 File → Add Files to “项目名”，项目名是指你当前要添加文件的项目名称。同样，你也可以在相应项目图标上右键选择 “ Add Files to ‘项目名’”，还可以直接使用 “ Option + Command + A” 快捷键。

选中之后，Xcode 将会弹出一个 “打开文件” 窗口，如图 3-6 所示。从窗口顶部选择要添加的文件，窗口底部有一些选项，用来设置添加文件时的操作。其中几个选项的说明如下。

在添加文件时，已经被添加进项目的文件将不可选。

目标　选中 Destination（目标）选项将会使 Xcode 将所选文件复制到项目的相应目录下面（如果文件存在）。如果不选择这个选项，将会导致 Xcode 仅仅添加对该文件的引用，而没有真正地将文件复制到项目当中。

注意 在绝大多数情况下，最好将文件复制到项目文件夹当中，这将会确保文件始终可用。因为如果不小心改变了引用文件的位置，那么 Xcode 就需要重新添加引用，产生不必要的麻烦。当然，添加文件应用也有优势，一般情况下可能是在协作编程时，直接使用保存在网络卷上的资源。

文件夹添加 Added folders（文件夹添加）选项指出 Xcode 在将文件夹添加到项目的时候采用何种操作。“Create groups”将创建与文件夹同名的分组，然后将该文件夹的所有内容置于该分组之下。“Create folder references”创建文件夹引用，直接将物理文件夹本身添加进来，保留了文件系统的层次结构。从图 3-6 和图 3-7 可以看出，使用“Create groups”功能添加的文件夹的图标是黄色的，而使用“Create folder references”功能添加的文件夹的图标是蓝色的。

图 3-6 添加文件到项目

添加到对象 Add to targets（添加到对象）选项根据文件类型将文件添加到项目中所选对象的 Build Phases（编译阶段）中。Xcode 会将源代码文件自动添加到所选目标的 Compile

Sources（编译源代码）编译阶段中，而其他大部分文件类型将添加到 Copy Files（复制文件）编译阶段中。

3.1.1.4　直接拖放文件

可以直接将文件或者文件夹拖放到项目当中，如图 3-7 所示。拖放之后会出现一个文件添加设置窗口，如图 3-8 所示。其中的设置选项与添加文件到项目的设置选项相同。单击“Finish”按钮就可以将文件添加到项目当中。

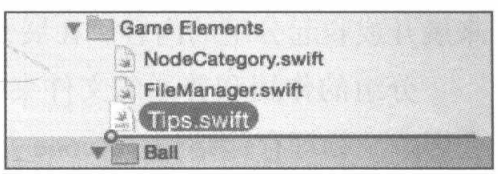


图 3-7　拖放文件到项目中

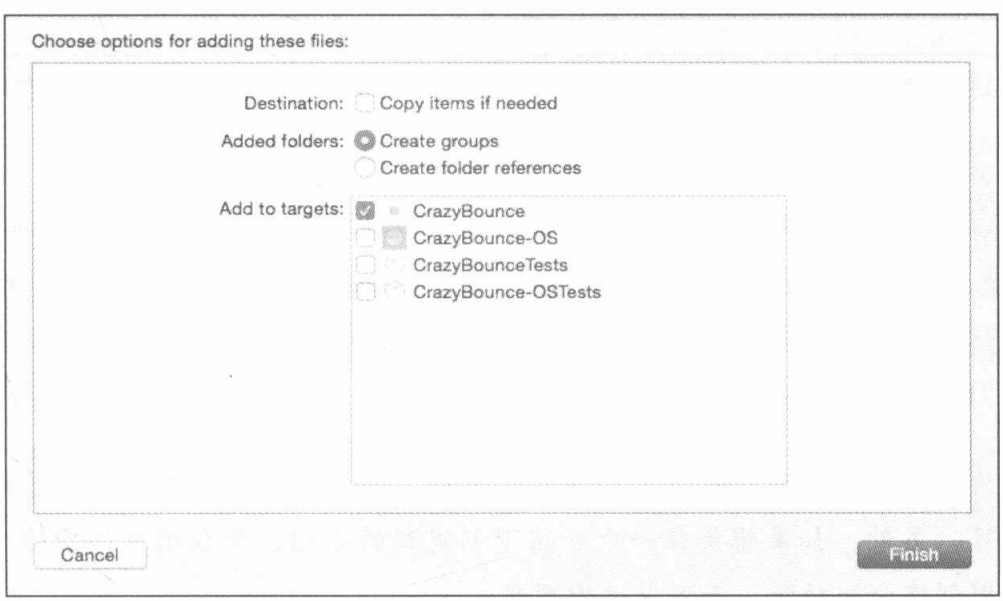


图 3-8　文件添加设置

注意　在添加现有文件的时候，最好将要添加的文件自行复制到该项目的物理文件夹当中。因为如果不这样做，当你改变了引用文件的路径地址时，就会破坏该引用文件的逻辑地址，导致必须重新添加文件。

3.1.1.5　重命名文件

重命名文件的方法也很简单，在项目导航器中选择需要修改名字的文件，然后单击即可开始修改，也可以在工具区域中文件检查器的“Identity and Type”项里面修改“Name”一栏。

提示　如果需要修改文件扩展名，那么仅仅修改扩展名是不够的。Xcode 已经记下了原文件的编译方式，因此在编译修改后的文件会发生问题。于是，在修改完文件扩展名之后，需要在工具区域中的文件检查器的“Identity and Type”项中修改“File Type”一栏，将文件类型修改为目前文件的扩展类型。

3.1.2 分组

项目的所有文件都以树型列表的形式显示在项目导航栏当中，这种结构叫做项目结构分组，简称为分组。分组中包含了项目中的全部内容，可以通过单击分组图标旁边的小三角形来展开或收起分组的内容。在展开或收起分组时按住 Option 键可以展开或收起全部的子分组。

分组的作用和普通的文件夹很相似，但它是一个逻辑结构。可以随意设计分组，这并不会影响到物理存储结构。Xcode 并没有对分组做出任何规定，虽然分组设计并不会显著改善代码编译的速度，但是一个好的分组可以让文件结构变得清晰明了、便于管理，比如将框架文件都置于 Framework 分组当中，将类文件都置于 Classes 分组当中。

虽然通常情况下可以根据自己的意愿来随意组织分组，但是请不要随意删除或重命名 Products 分组。此外，移动、重命名、删除分组的操作是不可恢复的，在操作之前请万分小心！

通过从菜单栏选择 File → New Group 或者右键单击项目导航栏上的某项，然后选择 New Group 来创建新的空白分组。新创建的分组名称为 New Group，可以自行修改分组名称。分组的快捷键为“Option + Command + N”。

可以为多个文件或分组创建一个共同的分组，只需要选择多个文件或分组然后创建分组即可。另外，如果想要在一个分组下创建新的分组，那么请先选中这个分组，然后再使用创建分组功能，而不是选中项目。

New Group 表示为所选项创建子分组，而 New Group from Selection 则表示为所选项创建父分组，如图 3-9 所示。

同样，可以任意重新排列分组的位置，直接进行拖曳操作即可。指示器会提示分组置放的位置。

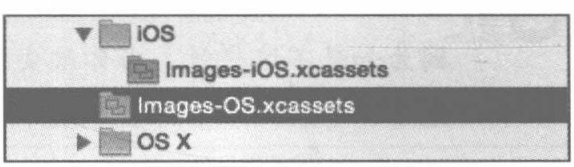


图 3-9 分组

要删除分组，选择 Edit → Delete 命令，或者直接按下键盘上的“Delete”键即可删除，还可以右键选择“Delete”选项删除。虽然简单，但缺点是分组下的其他分组和文件都会被一起删除。因此在删除分组前，请先确认分组下的文件是否需要，如果需要，则应当将这些文件移动出来。

3.1.3 删除及重命名文件

如果要从项目中删除文件，只需要在项目导航器中选中它并删除即可。删除文件的方式和删除分组的方式相同。Xcode 将会弹出一个对话框，如图 3-10 所示，以询问你是删除文件引用还是直接从磁盘上删除文件。根据需要选择删除的方式，一般情况下如果该文件是采用

引用方式引用的，就选择删除文件引用的方式。

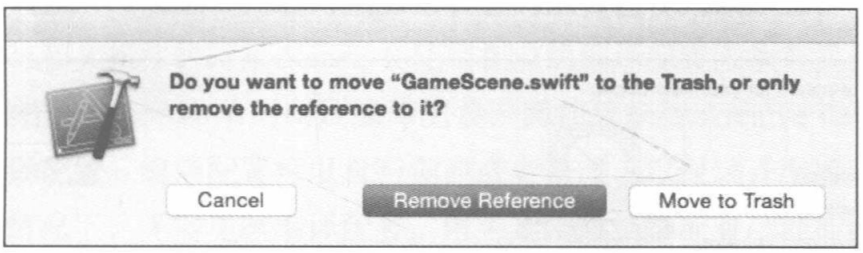


图 3-10　确认文件删除操作

3.2　对象管理

对象（Target）描述了被构建的产品（应用、扩展、单元测试包或者其他目标的集合）和用于构建它的指令。这些指令规定构建设置、阶段、规则和项目中的源代码和源文件。一个项目可能包含超过一个对象。例如，Xcode 默认创建的 iOS 项目中包含两个对象，一个是 iOS 应用本身，另一个则是该应用对应的单元测试对象。对象也可以关联另一个对象和运行脚本，这样就可以创建一个“部署包”对象。

简单来说，对象就代表了一个实际可运行的应用，可供 Xcode 来进行编译。实际上，使用项目模板创建项目时，当中的一个个模板就是一个个对象，只不过某些项目模板包含多个对象而已。关于项目模板的有关知识，请参阅本书的附录 C 中 C.2 节。

在项目导航器中选择项目文件，然后就可以在项目和对象列表中查看当前项目所拥有的全部对象，如图 3-11 所示。

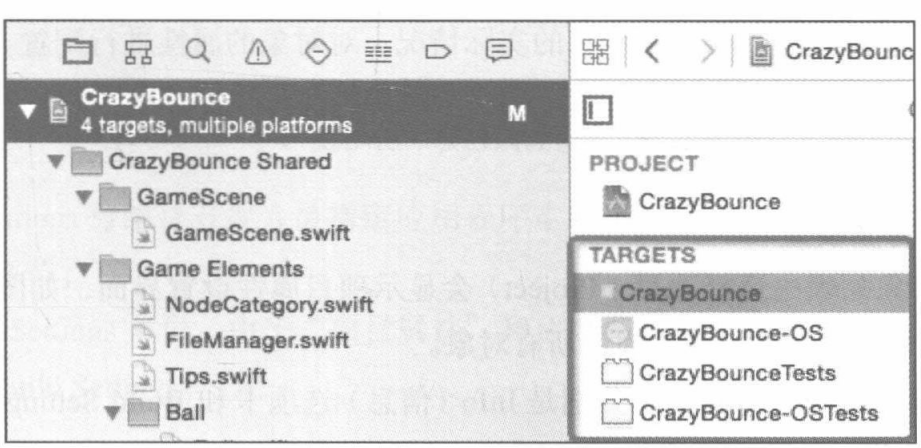


图 3-11　查看对象

3.2.1　添加对象

如果要在项目中添加多个对象，只需要选择菜单栏的 File → New → Target，就可以在其

中添加新的对象了。弹出的对象模板窗口如图 3-12 所示。可以看出，对象模板窗口和项目模板窗口是非常相似的，不过对象模板窗口比起项目模板窗口来说多了不少选项。

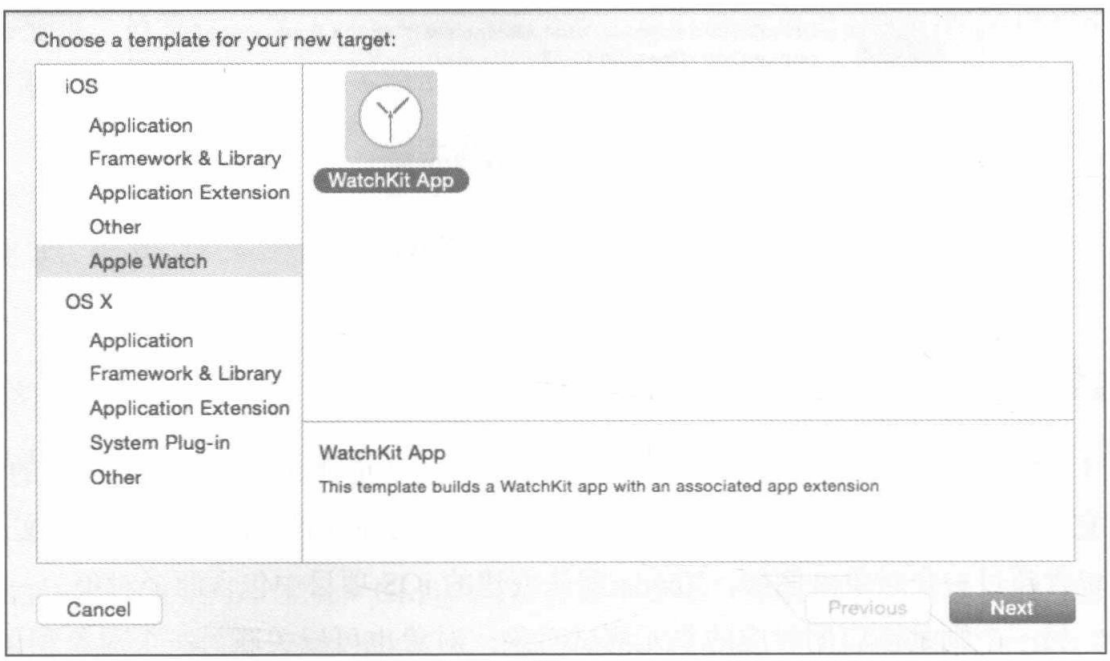


图 3-12 通过对象模板来添加对象

当然，还可以在项目和对象列表的下方点击“+”号来添加对象。点击“Next”按钮即可进行进一步的设置，这个设置方式和新建项目窗口非常相似，这里就不赘述了。

3.2.2 对象设置

对象描述了被构建的产品和用于构建它的指令，因此有些时候 Xcode 的默认设置并不是我们所想要的，我们往往需要根据项目的实际情况，对对象的属性进行配置。而在整个对象设置环节中，包括了“项目属性”的设置和“对象属性”的设置，对项目属性的设置会影响到对象属性的设置。

3.2.2.1 项目属性设置

在项目和对象列表中选择项目（Project）会显示项目属性设置界面，如图 3-13 所示，这些属性是全局属性，适用于该项目下的所有对象。

项目属性包含有两个选项卡，分别是 Info（信息）选项卡和 Build Settings 选项卡。Info 选项卡中有三个主要分组：Deployment Target（部署对象）、Configuration（配置）以及 Localizations（本地化），如图 3-13 所示。分组介绍如下：

- Deployment Target 分组用来定义项目所有对象的最低 OS 版本，这根据平台的不同而不同。
- Configurations 分组可以自定义项目所有对象的可用编译配置。编译配置主要描述了

编译环境的相关设置属性。默认配置主要有 Debug（调试）和 Release（发布）两种默认的编译配置。两者的不同之处主要在于它们是否包含调试信息，以及优化方式，等等。在执行相关操作的时候 Xcode 会使用相应的编译配置来执行操作。比如，如果执行 Run 操作，Xcode 则采用调试编译配置，而如果执行 Archive 操作，Xcode 则会采用发布编译配置。编译配置可以自行创建或者修改，以便编译配置更加符合应用的需求。一般情况下，无需更改编译配置，因为这两项编译配置已经能够满足一般的开发需求了。在这个组的底部有一个选项（for command-line builds），用于指定使用 Xcode 命令行界面时，使用哪一个编译配置进行构建。

图 3-13　项目属性设置

- Localizations 分组让开发人员指定应用程序本地化的语言，关于本地化的相关信息请参阅 7.8 节。

对于 Build Settings 来说，由于“项目属性”和“对象属性”基本上大同小异，因此我们将在下面介绍 Build Settings。

3.2.2.2　对象属性设置

在图 3-13 中选择左栏的 TARGETS，出现如图 3-14 所示的界面，在这里进行对象属性设置。应用程序对象的设置琳琅满目，让我们慢慢来梳理这些设置的具体内容。

1. 通用（General）选项卡

通用选项卡显示了对象的基本信息，也是最常用的对象属性设置面板，如图 3-14 所示。

Mac OS 应用程序和 iOS 应用程序各有不同。

Identity（标识符）栏主要定义了一些和应用发布有关的标识属性。对于 Mac OS 应用程序来说：

- Application Category（应用程序类别）用于在 Mac App Store 中定义该应用程序的分类；

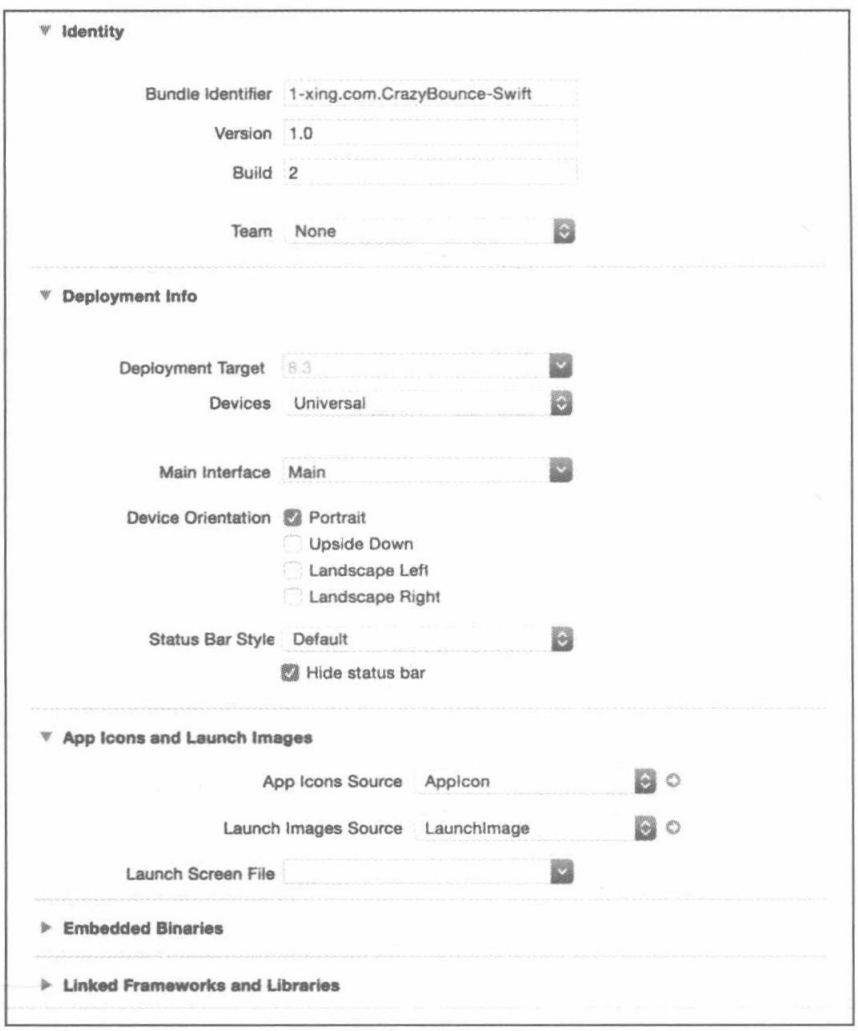


图 3-14　对象属性设置 –iOS 应用通用选项卡

- Bundle Identifier（包标识符）是该应用的唯一 ID，用来让操作系统和 App Store 识别。在创建项目或者对象过程中 Xcode 就自行创建了包标识符，一般情况下请不要修改它；
- Version（外部版本号）是用户所能够看到的版本号；
- Build（内部版本号）则是开发者自己所看到的版本号，以区分内部测试版本；
- Signing（签名）则是标定应用的签名 ID，如果选择“Mac App Store”或者“Developer ID”，就表明这个应用是经过验证的，Mac 系统默认情况下不会阻止这类应用程序的打开；
- Team（团队）则是在选择签名的时候使用，用于和加入了苹果开发者计划的开发团队

进行关联；

Deployment Info（部署信息）栏则定义了一些和应用配置相关的标识属性。对于 iOS 应用程序来说：

- Deployment Target（部署对象）和上节介绍的“项目属性设置”中的相同。
- Devices（设备）定义了该应用可以运行在 iPhone 上，还是 iPad 上，抑或是 Universal（通用）；
- Main Interface（主界面）是应用启动时预加载的主界面视图。
- Device Orientation(设备方向）定义了 iOS 应用所支持的方向。分别是 Portrait(竖直)、Upside Down（上下颠倒）、Landscape Left（左横置）和 Landscape Right（右横置）。
- Status Bar Style（状态栏样式）定义了 iOS 设备顶部状态栏的样式。

App Icons（应用图标）则定义了应用图标的来源，默认情况下定位到资源管理目录当中。对于 iOS 应用来说，还定义了加载界面的来源，Xcode 6 默认创建的 iOS 应用程序采用 LaunchScreen.xib 文件来定义加载界面。当然，还可以使用资源管理来定义加载界面。

Embedded Binaries（绑定二进制文件）则用来链接二进制文件，一般在使用第三方 SDK 的时候会用到。

Linked Frameworks and Libraries（链接的框架和库）则用来链接框架和库，从中可以选择系统 SDK 自带的框架，也可以选择第三方框架。

2. 功能（Capabilities）选项卡

功能选项卡中定义了 Xcode 提供的许多特殊功能，如图 3-15 所示。

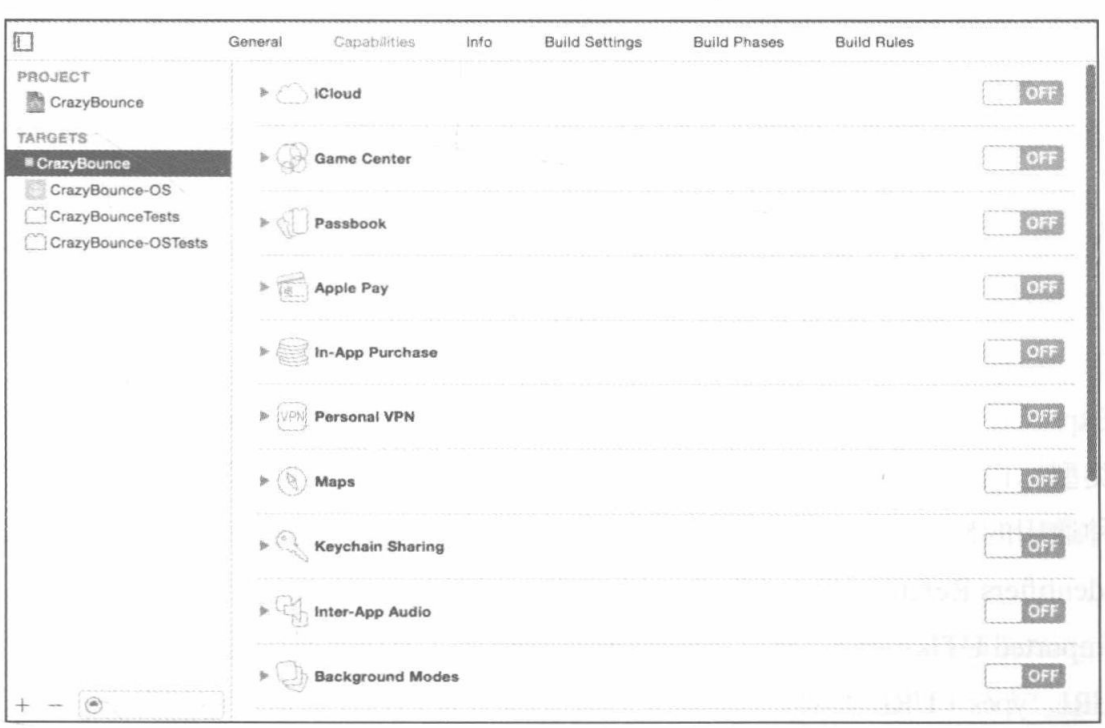


图 3-15　对象属性设置 – 功能选项卡

3. 信息（Info）选项卡

信息选项卡显示了应用程序相关的属性、应用程序能够创建和打开的文件类型，以及应用程序所能提供的服务，如图 3-16 所示。

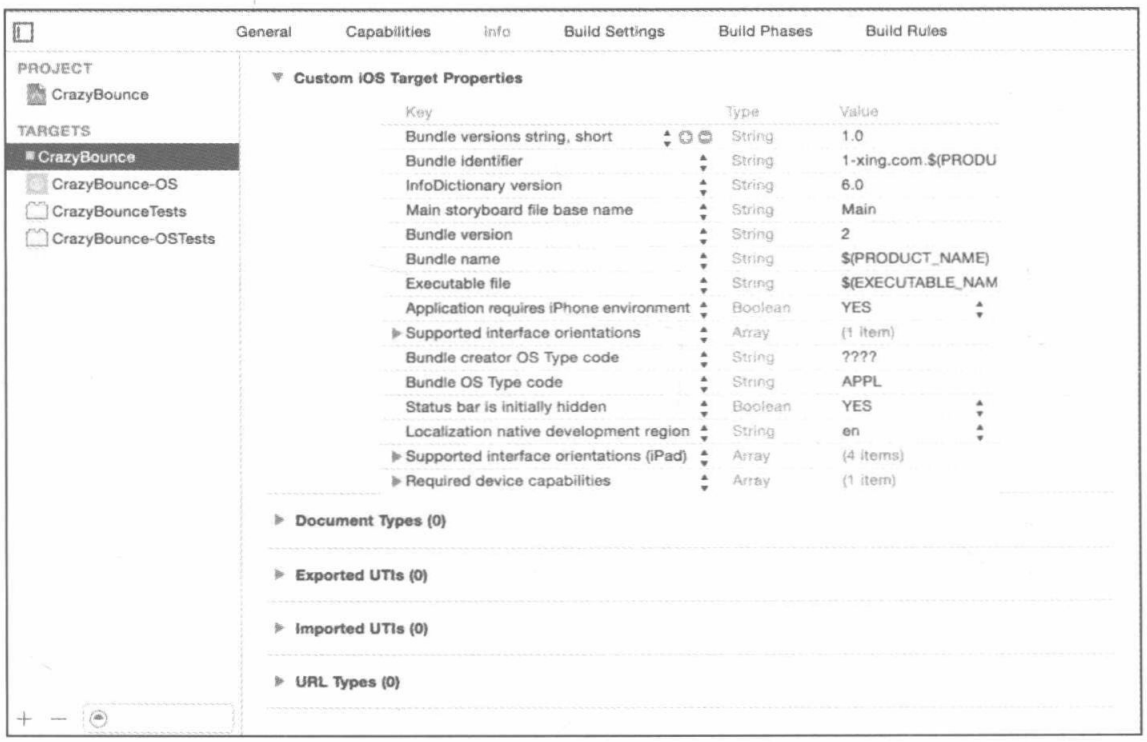


图 3-16　对象属性设置 – 信息选项卡

- Custom iOS Target Properties（自定义 iOS 目标属性）列出了应用程序属性列表的基本部分（包括版本号、应用图标、使用的主 nib 文件等）。其中有些信息是在 General 选项卡中互通的。如果需要，可以手动修改这里的值。关于关键字及其含义的更多信息请参考 Apple 文档的“Bundle Programming Guide”中的“Bundle Structures”小节。
- Documents Type（文档类型）定义了应用程序所能够识别的文档类型，并且还可以定义在系统中显示的该类型文档的自定义图标。如果应用程序不需要识别文档，那么默认为空即可。
- Exported UTIs（导出 UTI）可以让应用程序导出 UTI（Uniform Type Identifiers，统一类型标识符）。定义 UTI 可以让应用能够识别对应的文件类型，因为 UTI 指定的是一种通用的格式。关于支持类型列表的相关信息，请参阅 Apple 文档的“Uniform Type Identifiers Reference”一节。
- Imported UTIs（导入 UTI）则和导出 UTI 对应，是用来导入 UTI 的。
- URL Types（URL 类型）用来定义 URL 以便让应用程序理解应用间交换的数据结构，比如 HTTP、SMS 等等。有关 URL 的相关知识，请参阅 Apple 文档“App Programming

Guide for iOS”中的“Using URL Schemes to Communicate with App”一节。

- Services（服务）用来定义系统服务，从而使该服务可以从其他应用程序中访问。

4. 编译设置（Build Settings）选项卡

编译设置中包含对象架构和 SDK、中间编译文件的位置、编译器选择、链接器设置、搜索路径等。这些设置可以是作用于全项目范围的，也可以是作用于特定对象的，同时，也可以为某个特定的编译配置设置一个单独的值，如图 3-17 所示。

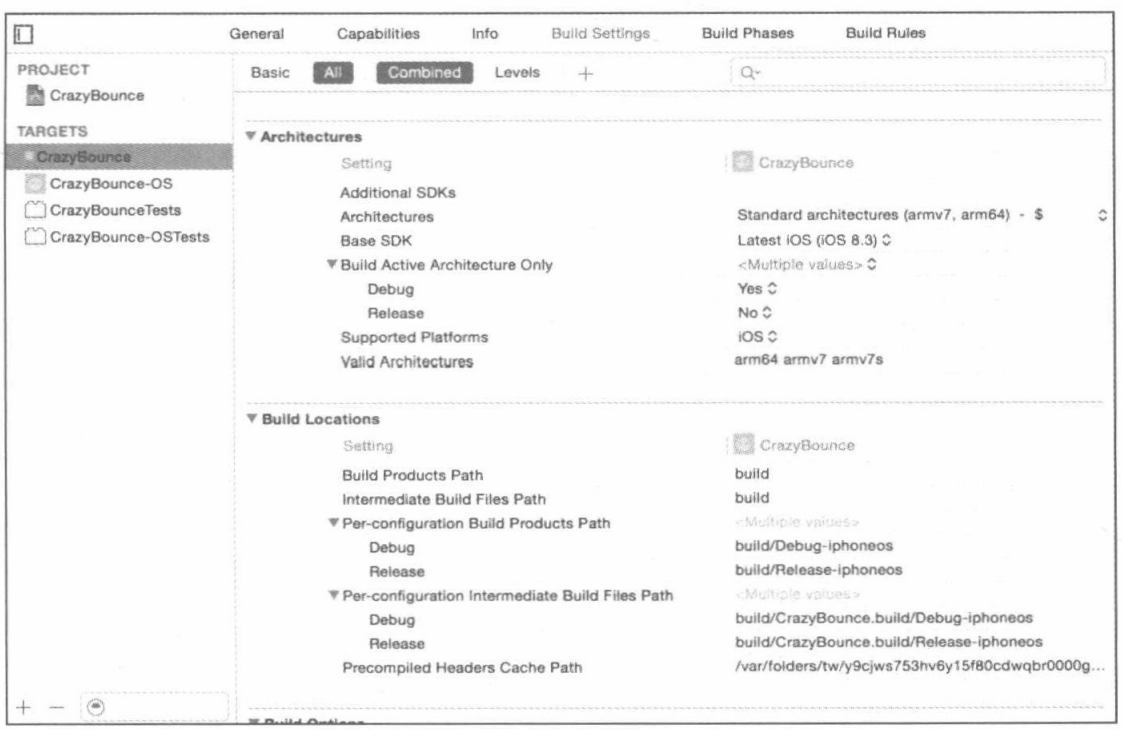


图 3-17　对象属性设置 – 编译设置选项卡

将过滤栏设置为“Basic”（基本），则编译设置选项卡将会显示最常用的设置及其设置值。如果设置为“All”（全部），那么编译设置选项卡则会显示全部设置内容。

如果只打算查看某个对象的编译设置，那么使用“Combined”（组合）即可满足要求，它仅仅只会显示该对象的编译设置。对于具有多个对象的项目，那么在过滤器栏中选择“Levels”（层级），就可以获得更为精确的视图模式。选择之后，就会显示出多个列来显示各个对象的编译配置，从左到右依次是所选对象的最终设置（Resp; ved）、对象编译设置、项目编译设置以及默认值（Default）。

如果某个编译配置被某个子结构中的编译配置（从左到右层级依次上升），那么这个值就会有绿色的轮廓。这有助于确定对象的准确设置。

当然，也可以使用过滤栏中间的“+”号按钮，选择“Add User-Defined Setting”（添加用户定义设置）选项来添加自定义编译设置。这可以极大地增加编译过程的自主性和灵活性，用户自定义的编译设置将出现在列表底部的 User Defined 组当中。

值得提醒的是，这个选项卡中的绝大多数设置会影响编译、链接、生成等操作，并且可能并不会有警告和错误提示。因此在没有必要的时候请不要随意改变这个选项卡中的内容，不过工具区域中的快速帮助检查器则提供了每个设置的详细帮助信息以及效果，要修改的时候请多参考这个帮助信息的内容。

5. 编译阶段（Build Phases）选项卡

编译阶段是当前对象编译过程中的一个阶段，一般系统默认有对象依赖、汇编资源以及复制包资源等（如图 3-18 所示）。此外，还可以自行添加额外的阶段，甚至还可以用 Shell 脚本以便在编译处理期间提供更为复杂、精确的操纵。

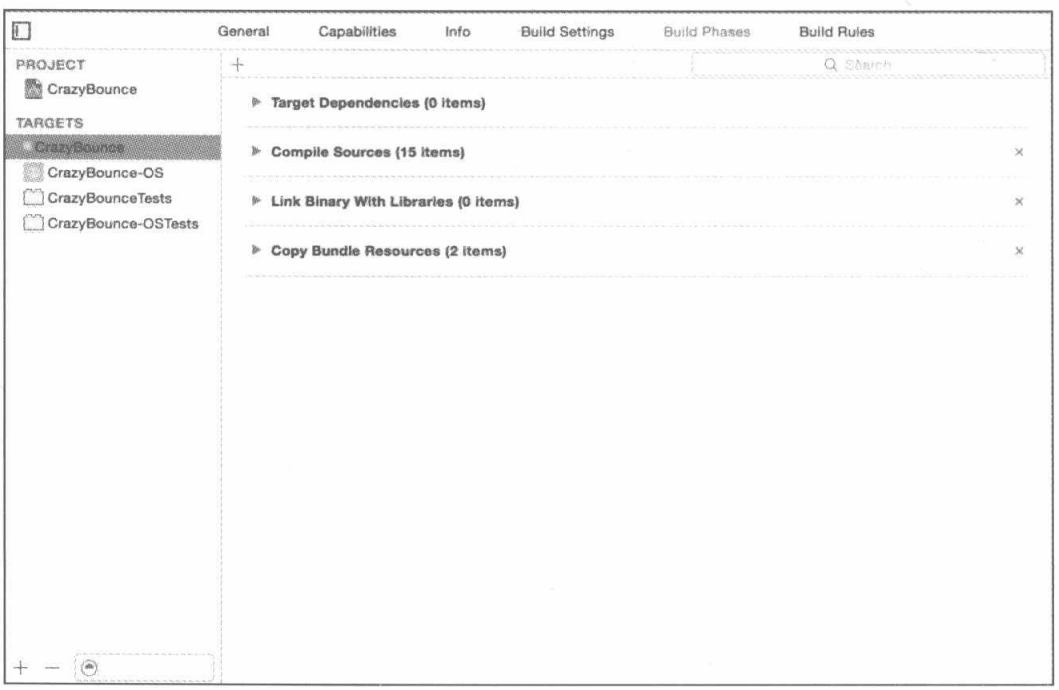


图 3-18　对象属性设置 – 编译阶段选项卡

对象依赖（Target Dependencies）阶段可以让 Xcode 知道必须在当前选择的对象编译之前编译的其他依赖对象（比如应用扩展、插件等等）。这个阶段是无法被删除的。

编译源文件（Compile Sources）阶段按照 Build Rules 选项卡中的定义，用合适的编译器来编译该对象的所有源文件。可以通过修改该阶段的 Compiler Flags（编译器标识）来为每个单独文件设置其编译器标识，比如优化设置等等。

链接二进制文件和库（Link Binary With Libraries）中，可以控制该对象需要链接哪些二进制文件和库。只要使用了二进制文件或者库，就必须要链接它们。有些常用的框架（比如说 Cocoa 框架）并不会显式显示在里面，但是实际上它是包含的，重复添加并不会造成框架的重复引用。

复制包资源（Copy Bundle Resources）阶段中定义了对象中的资源文件，包括应用程序、图标、界面构造器、视频、模板等等。这些资源都会被复制到安装包的 Contents/Resources 文

件夹下。

可以通过点击选项卡栏下方的“+”按钮来添加另外 3 种类型的编译阶段。这些阶段默认情况下不会在应用程序中出现。

文件复制（Copy Files）阶段用来标识文件复制的目标路径。可以从 Destination（目标）菜单中选择预定义的目标，也可以自己指定路径。文件复制阶段常常用来将文件复制到预定义的位置。预定义的路径包括程序安装包的 Resources、Frameworks 文件夹、共享框架路径等等。Subpath（子路径）字段则会在 Destination 菜单中选择的路径后附加一个子路径名称。“Copy only when installing”（仅在安装时复制）命令 Xcode 只在当前编译设置中包含 Install 标识时才复制文件。

头文件（Headers）阶段用来为产品指定其头文件以及可见性。作用域（Public、Private 和 Project）则用来确定头文件的可见性。Public 域则包含在最终产品当中，作为可读源代码存在；Private 域虽然也包含在最终产品当中，但是不能够直接读取；Project 域则不包含在最终产品当中，只有项目在编译对象时才使用。

运行脚本（Run Script）阶段可以使脚本得以运行，脚本可以在脚本编辑器区域当中编辑，也可以直接放入一个脚本文件进来。

6. 编译规则（Build Rules）选项卡

编译规则用于定义文件类型以及用于编译它们的处理器之间的关联信息。默认的编译规则是项目通用的，当然，也可以为新的文件类型定义编译规则，也可以指定处理该类型文件的工具，如图 3-19 所示。

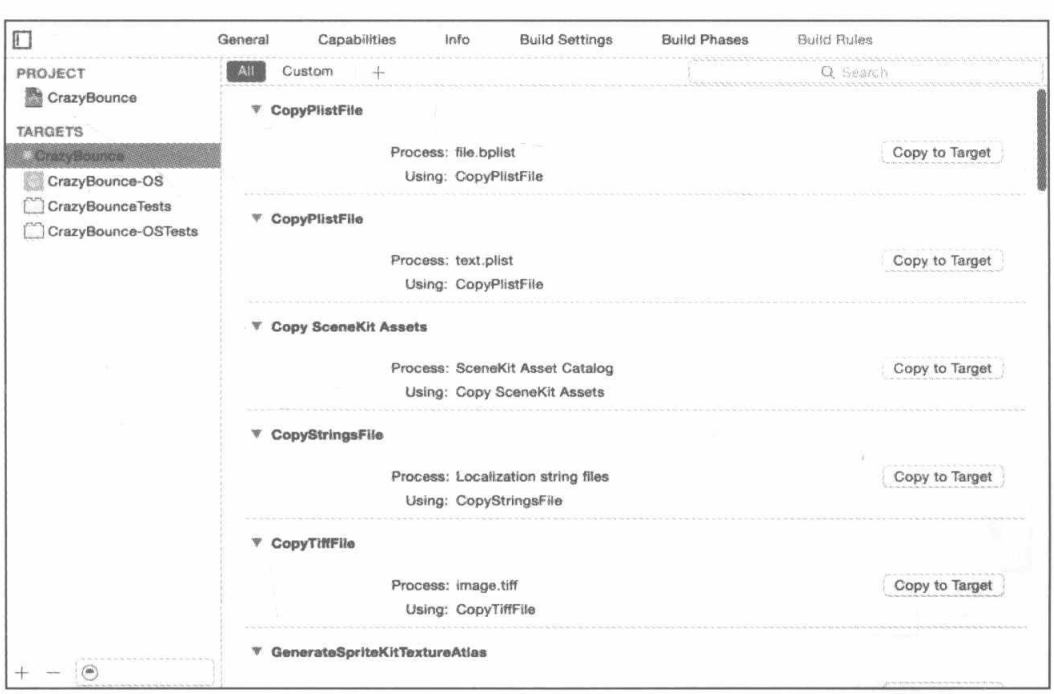


图 3-19　对象属性设置 – 编译规则选项卡

将编译规则选项卡的过滤栏设置为“Custom”，那么只会看到对象所有定义的编译规则，也就是过滤掉默认的系统编译规则。

要为给定的文件类型自定义系统编译规则，可以在“All”列表中找到该文件类型，然后单击“Copy to Target”（复制到对象）按钮。这样新的自定义编译规则就会添加到对象当中，然后就可以指定处理该文件类型的脚本或者预定义程序了。此外，单击选项卡下方的“+”按钮还可以添加新的编译规则。

3.2.3 对象联系

有些时候，某些文件可能需要在多个对象之间共享，因此可以在这个文件与多个对象之间建立联系。

在创建文件的时候，Xcode 就给出了一个选项用于为该文件指定一个或多个目标。

对于一个已经存在的文件来说，最简单的方法是在项目导航器中选择该文件并打开文件检查器。在“Target Membership”（对象成员）区域中就可以选择需要联系的对象，如图 3-20 所示。此外，也可以在项目编辑器中的相关对象中，选择“编译阶段”选项卡，然后将该文件拖动到 Compile Sources 构建阶段上面，这样，当这个对象在编译的时候，会自动将该文件一同编译进去。

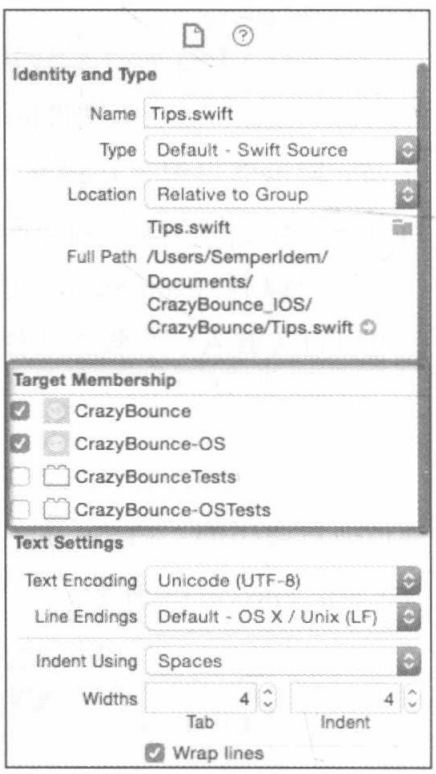


图 3-20 建立对象联系

3.2.4 删除对象

删除对象的方法很简单。在项目导航器中选择项目文件，然后定位到项目和对象列表上要删除的对象，按下“Delete”键或者使用底部的“–”按钮删除即可，Xcode 会弹出如图 3-21 所示的提示。

选择“Delete”按钮即可完成删除。要注意的是，该对象所关联的文件并不会被删除。

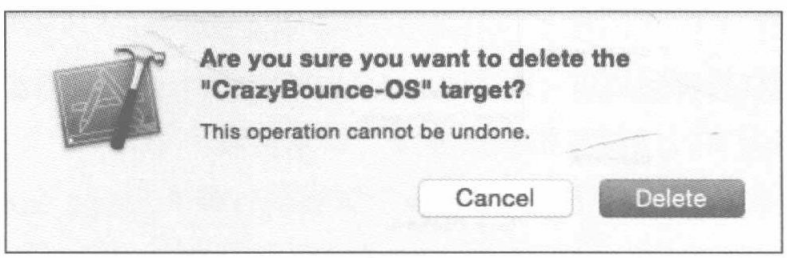


图 3-21 确认是否删除对象

3.3　资源管理

资源目录（Asset Catalog）是一个特殊的文件夹，可以很简单地管理一张图片的多个版本（比如：普通版、Retina 版、4 英寸 iPhone 版本、iPad 版本，等等），所有图片只有一个文件名。它可以减少项目导航栏上多个图片文件所造成的混乱，使图片管理更有条理。

资源目录可以存放以下几种图片类型。

- 图片集（Image Sets）：绝大多数可用的图片类型，一般用来存储需要在用户界面上显示的本地图片。对于同一个图片来说，图片集中可以设置在不同版本、不同尺寸设备上应该如何显示。
- 应用图标（App Icons）：iOS 应用、Mac 应用显示在主页上以及 App Store 上的图标。
- 加载界面（Launch Images）：打开 iOS 应用的过程中显示的图片。
- OS X 图标（OS X Icons）：OS X 应用在 Mac 上以及 Mac App Store 上显示的图标。

下面介绍资源管理的一些操作，包括创建、添加、移除图片、图片集等。

3.3.1　创建 Asset Catalog

通过 Xcode 6 创建的项目已经包含了一个名为“Images.xcassets”的资源目录（Assert Catalog）。如果要创建另外的资源目录来管理图片，那么创建方法和创建文件的方法很类似。在选择文件模板的时候，选择“Resource”一栏，然后选择“Asset Catalog”文件创建即可，如图 3-22 所示。

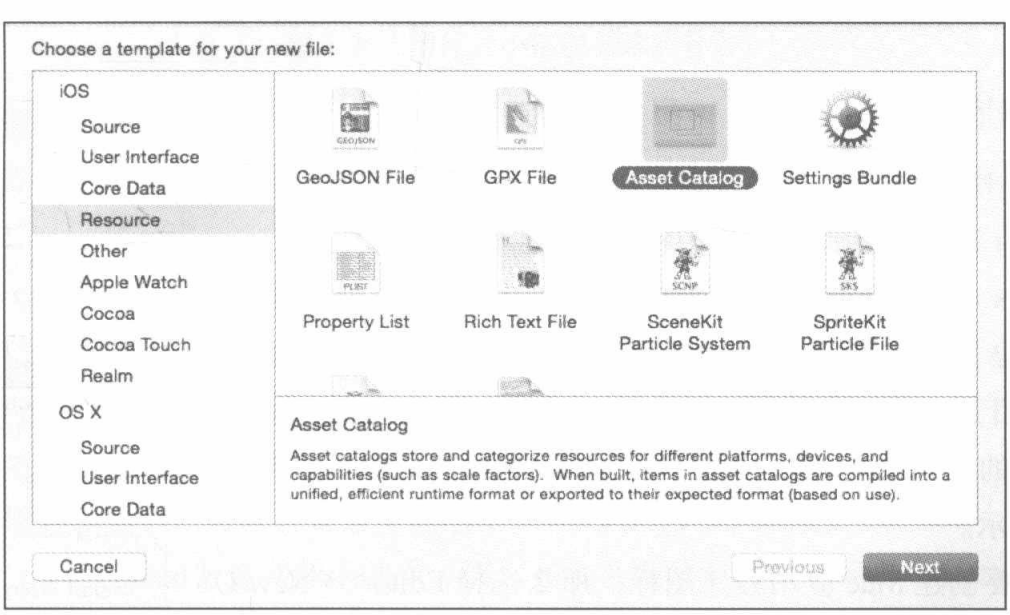


图 3-22　创建资源目录

创建完的资源目录视图如图 3-23 所示，资源目录从左到右分为三个区域，分别如下。

- 图片列表：用来显示资源目录下的所有图片集。

- 视图列表：用来显示某个图片集下适应所有尺寸、所有大小以及所有版本的图片。
- 属性检查器：用来设定某个图片集的属性。

图 3-23 资源目录界面

资源目录下的所有图片集都拥有一个统一的名字，可以通过在代码中直接引用这个名字来将图片加载到你的应用当中，操作系统会根据系统版本、设备尺寸等要素选择相应的图片来展现。

3.3.2 添加图标

对于应用程序来说，首先展现给用户的就是应用的名字以及应用的图标。一个好的名字和图标可以有效地吸引用户来使用你的应用。

添加 iOS 应用图标的方法很简单。选中一个资源目录，如果这个资源目录是系统自建的，那么你会看到一个 AppIcon 图片集；如果没有资源目录，那么选择菜单栏上的 Editor → New App Icon 即可创建一个空的应用图标集，在图片列表处点击右键也可以创建，如图 3-24 所示。

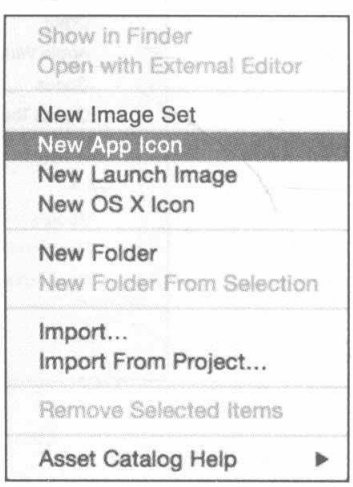


图 3-24 添加应用图标

如果是要创建 Mac 应用程序图标，那么选择 Editor → New OS X Icon，然后配置方式与 iOS 相同。

应用图标集如图 3-25 所示。从 Finder 中拖曳相应尺寸的图片文件到视图列表图标的格子中，即可完成图标的添加。

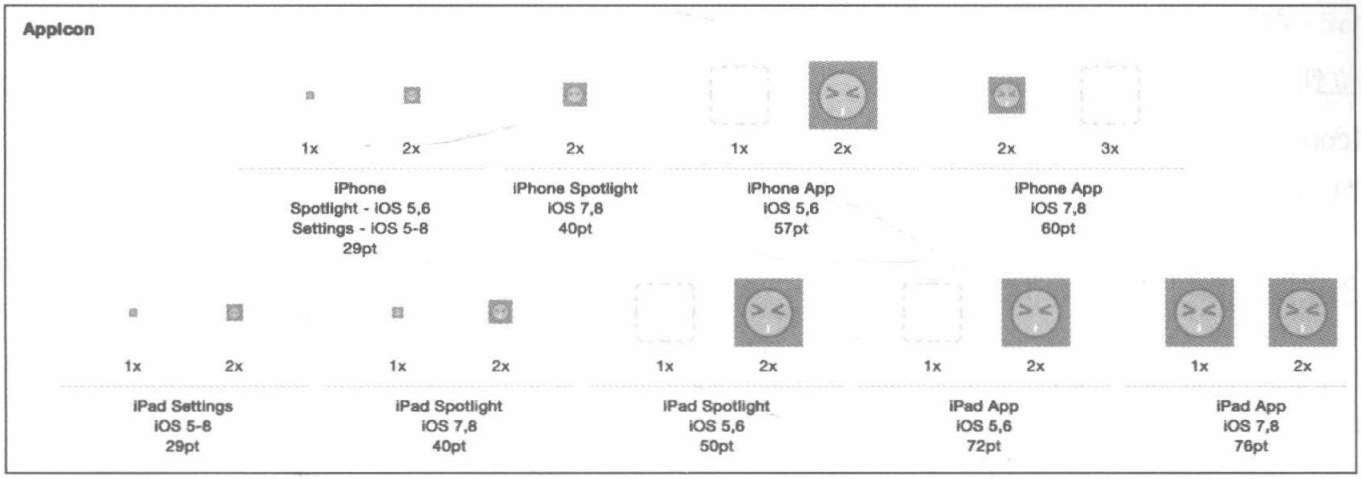


图 3-25　应用图标集

要注意的是，Xcode 对图标的尺寸有着严格的限制。在每个格子下方，有“29pt”、“40pt”系列等字样。这表明这个格子能够接受的图标尺寸为“29 像素”系列或者“40 像素”系列，等等。如果你把尺寸错误的应用图标拖放到了格子中，那么 Xcode 就会弹出一个警告，如图 3-26 所示。

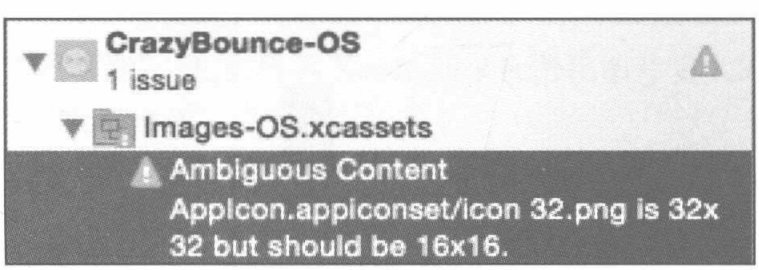


图 3-26　图标大小错误导致的警告

每个格子下方可能还有“1x”、“2x”的字样，这表明这个格子所接受的尺寸大小是“系列尺寸”的几倍。比如“29pt”系列的格子下方的显示的是“2x”，表明这个格子所接受的尺寸是“29×2”＝“58pt”。

在系列尺寸上方，是这个格子显示的图标所能够支持的最低系统版本。

在系统版本上方，或者左方，则是该格子显示的图标所能够支持的设备名称以及显示的位置。Spotlight 指的是在快速索引界面中显示的图标，Settings 则是在设置应用中显示的图标，而 App 则是在桌面、App Store 上显示的图标。

在属性检查器当中，可以设置这个应用图标集的名称，以及所支持的版本、尺寸。其中，“iOS icon is pre-rendered”选项如果选中，那么 iOS 系统就不会在主界面上给予这个图标添加圆角和高光。

如果你不是使用的项目默认的资源目录添加的图标，那么还需要配置一下项目才能够让 Xcode 识别这个应用图标集。

在项目导航器当中选中这个应用图标集所属的对象，定位到“General”选项卡，然后定位到“App Icons and Launch Images”栏目中的“App Icons Source”（Mac 应用则是唯一的“App Icons”栏目中的“Source”）。打开下拉列表（如果没有其他的资源目录，它可能显示的是“Use Asset Catalog 按钮”），然后在弹出的对话框中选择应用图标集所在的资源目录即可。

3.3.3 添加加载界面

添加加载界面的方式和添加图标的方式很类似，需要选择菜单栏上的 Editor → New Launch Image。建立好的界面如图 3-27 所示。


注意 Xcode 6 新创建的项目中并不会自动添加 Launch Image，相反它创建了一个 xib 文件（“LaunchScreen.xib”）来管理加载界面。加载界面文件是 iOS 8 提供的一个新特性，关于配置“加载界面文件”的相关内容，请参考后面 4.3 节。

默认情况下，加载界面图片并不会在 Xcode 6 当中自动配置，因此需要配置一下项目才能够让 Xcode 识别这个加载界面图片。

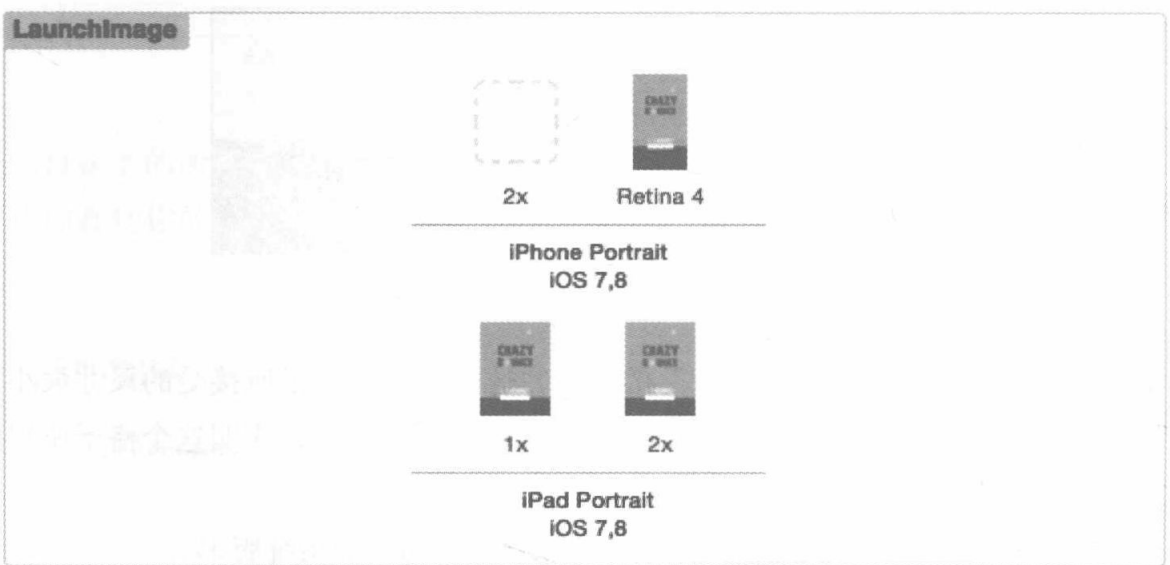


图 3-27 加载界面

和配置应用图标的方式相似，在“Launch images Source”当中定位到加载界面图片所在的资源目录即可。如果你的应用要兼容 iOS 8 以下的版本，那么同时配置加载界面图片和加载界面文件是必须的，只不过在 iOS 8 以上的版本中应用将会使用加载界面文件，其余版本使用加载界面图片。加载界面文件的配置在下面的“Launch Screen File”当中。

3.3.4 管理图片集

对于普通的图片文件来说，直接将图片文件拖到资源目录当中即可完成图片集的创建。

创建好的图片集名字和图片名字相同。同样地，也可用使用菜单栏的 Editor → New Image Set 来创建空白的图片集。

要变更图片集的名字，可以在属性检查器当中修改，也可以直接在图片列表当中进行修改。

要向图片集当中添加图片，直接将图片文件拖曳到相应的格子上即可，或者使用 Editor → Import 功能添加。

在属性检查器当中，可以设置该图片集所支持的设备类型以及其他一系列内容。

“Devices”用来配置该图片集所支持的设备类型，基本上只有“iPhone”、“Retina 4-inch”、“iPad”以及“Mac”四种类型。

“Width”和“Height”是用来配套“Size Classes”使用的，可以根据不同的屏幕分类类型来显示不同的图片。关于“Size Classes”的有关内容，请参考第 6 章。

“Type”是用来配置支持的图片类型的，“Bitmaps”是位图文件，也就是由像素点组成的文件，“Vectors”是矢量图片文件，也就是由矢量图形组成的文件。

“Render As”则是设置图片集的渲染方式。“Original Image”则表示原图渲染，“Template Image”则表示该图片将作为最终图像的遮罩使用。如果采用“Default”方式，那么该图片则会根据包含它的视图来选择渲染方式。

图片集还支持“切图功能”，它会告诉 Xcode 如果视图比图片更大的时候如何拉伸图片。拉伸的方向是竖直拉伸，水平拉伸，或者双向拉伸。

在 Xcode 中切图的优点就是设计师们提供的图片的宽度是不受限制的。你不必告诉 Xcode 图片的末端在哪里。Xcode 会在运行的时候就生成一些最小的图片以供需要。这就意味着它会在将图片打包之前就切掉所有不需要的部分。这样就意味着，在 Xcode 中你可以看到图片的完整大版本，但是打包的时候又只提供最小的部分。

要启用划分功能，可以在菜单栏中选定“Editor”→“Show Slicing”。

从图中可以看到，单击“Start Slicing”按钮即可开始进行划分。然后图片上将会显示三个按钮，用来设置需要显示的划分区域，分别是水平方向上的划分、竖直方向上的划分以及全局划分。

接下来就可以通过移动内部划分线来设置可调整区域的大小了。如图所示：

要设置不可调整大小的区域，则可以移动外部划分线来调整。

可以使用属性检查器来微调区域大小，并且可以指定可调整大小中心区域是应该伸展还是平铺。

要注意的是，划分功能仅支持 iOS 7 以及 OS X 10.10 以上版本。

3.3.5　移除图片集

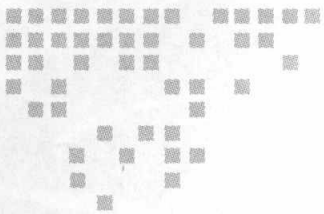
移除图片集中的图片以及移除图片集的方法很简单，按下 Delete 键或者右键选择 Remove

Selected Items 即可。

注意 移除图片集当中的图片并不会移除该图片所在的格子，这个格子仍然作为一个占位符存在。要移除这个格子，请使用属性检查器当中的 Devices 属性来进行删除。

神秘的藏经阁差点让良辰陷身其中不得出，其中经历的种种险境也是不由得外人肆意揣测，或许诸君也能感受得到其中的可怖之处。但是，自良辰从成功从藏经阁走出之后，他便开始踏入滚滚江湖，掀起了一番剑舞风云。正可谓是：此时风月那时楼，千经万卷述春秋。


外功修炼——设计篇



Chapter 4 第4章

风水宝地——界面生成器

少年良辰步履翩跹，扬袂走出了藏经阁。前路多舛，不过，此时的他正意气风发，心中一股犹如曹孟德当年临江感叹的激涌之情。不料，良辰抬头望见前方，龙首岩拔地千尺，危峰兀立，怪石磷峋，一块巨崖直立，另一块横断其上，直插天池山腰，势如苍龙昂首，气势非凡。

“此处为何地?”良辰一声喝道。

“风水宝地!”只见站在山崖边身着素衣，袭着飘飘白发的老者向良辰蹒跚走来，老者捻须说道：“风雨千山吾独行，天下无数豪杰，今日偏偏与少侠相遇于此阵法密布的风水宝地，真是有缘得很哪!”

“晚辈拜见前辈！不知大师……”良辰说道。

“惭愧惭愧，老夫无名耳，无非是对这阵法略有心得罢了。”老者微笑着说。

“阵法即作战队形，又可称为“布阵”。只有当布阵得法时，才能充分发挥军队的战斗力。白发老人在这不为人知的深山中独自钻研多年，岁月洪流洗礼，幽秘而智慧的阵法应运而生。老者见少年一脸迷茫，便拿起手中的拂尘在地面上比划起来。原来，这块所谓的“风水宝地”，便是奇妙的‘界面生成器’(Interface Builder)。排兵布阵在这“风水宝地”中乃得以实现。”

4.1 简介

在 Xcode 中，界面生成器可以让开发者简单快捷地开发出图形用户界面（GUI)。开发者只需要从工具箱中向窗口或菜单中拖曳控件即可完成界面的设计。然后，用连线将控件可以

提供的“动作”（Action）、对象“方法”（Method）、对象“接口”（Outlet）连接起来，就完成了整个创建工作。比起其他图形用户界面设计器，这样的过程减小了 MVC 模式中控制器和视图两层的耦合，提高了代码质量。

Xcode 6 中，有三种制作 UI 的方式：使用代码构建 UI 及布局、使用 Xib 文件构建 UI、使用 Storyboard 来构建 UI。下面分别介绍。

- 使用代码构建 UI：通常情况下，依赖多人合作的大型项目以及极客喜欢用代码构建 UI。除了使用纯代码构建 UI 对极客看起来很酷以外，代码 UI 相对容易进行多人协作，具有良好的重用性。缺点是编码效率低，完成一个并不太复杂的界面，可能需要写上数百行的 UI 代码。
- 使用 Xib 文件建 UI：相对于代码，使用 IB 来组织 UI，可以节省大量时间，从而得到更快的开发速度。尽快看到成品，至少尽快看到原型，是保持兴趣、继续深入和从事职业的有效动力。在 Xcode 5 中 Apple 大幅简化了 Xib 文件的格式以后，使用界面生成器在版本管理上其实和纯代码已经没有太大差异。
- 使用 Storyboard 构建 UI：它的出现，让开发变得像讲故事一样，UI 间的关系流程也一目了然。它其实是 Xib 的升级版本，将多个 Xib 统一管理了；相对于单个 Xib，其代码需求更少，使得对于界面的理解和修改的速度也得到了更大提升。缺点是，版本管理时，如果不分成多个 StoryBoard，多人修改可能会产生冲突。

在设计篇中，考虑到本篇面向的读者对象是设计师群体，我们不会介绍使用代码构建 UI 及布局的方式，只介绍使用 Xib 和 Storyboard 构建 UI 的方式。因此，也请各位认清楚本篇的舞台：Xib 或者 Storyboard 文件，这也是我们在上面大展身手的舞台！

4.2　界面生成器

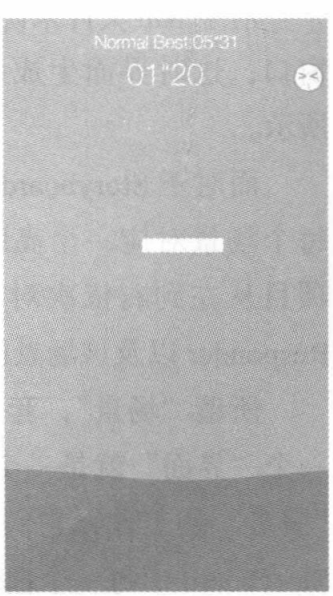


图 4-1　游戏界面

在设计篇中，我们会学到用 IB 来设计一个游戏“Crazy Bounce”的界面和交互。

先来看一下成品的界面，如图 4-1 所示。

在第 1 章中我们已经学会了如何创建项目，我们用同样的方法创建一个 Single View Application，系统会自动生成 Main.storyboard 和 LaunchScreen.xib 两个用户界面文件（具体区别下面会介绍）。单击这个文件，文件内容会在工作区窗口的编辑区域的界面生成器中打开。

Xcode 以 XML 格式储存 Xib 和 storyboard 文件内容。编译时，Xcode 将 Xib 和 storyboard 文件编译成二进制文件，即 nib 文件。运行时，nib 文件会被载入并实例化来创建新的视图。图 4-2 显示的是

storyboard 的界面，包括画布、对象窗口、检查器和组件区域。后面分别介绍界面生成器的各个功能。

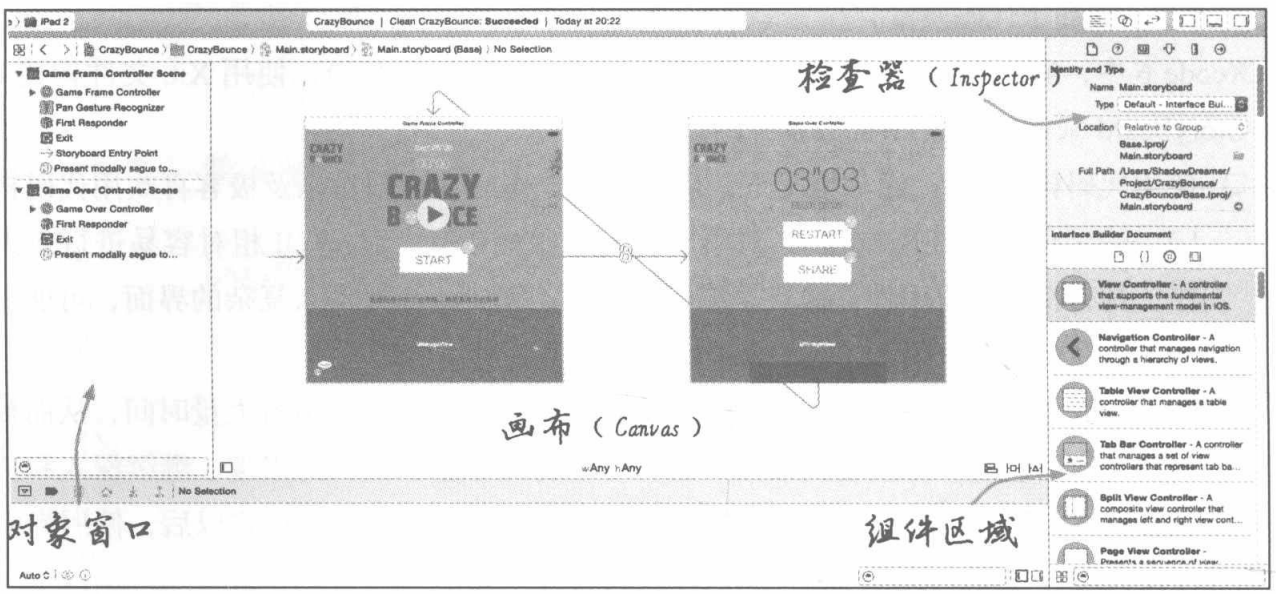


图 4-2　界面生成器界面

4.2.1　画布

画布（Canvas）用于设计用户界面，并对其中的控件进行布局等操作，就和 Phostoshop 的画布区域一样，设计师可以在上面进行设计。

简单来说，顾名思义，画布就像画家的画布一样，开发者和设计师都可以在上面以“可视化”的形式，将各式各样的控件放在合适的位置，完成应用的 UI 实现工作。

对于 Xib 文件来说，可以点击界面生成器左下角的按钮关闭对象窗口，这时界面生成器将会以 icon 视图形式展示高级对象，如图 4-3 所示。

图 4-3　icon 视图形式

而对于 Storyboard 文件来说，对象窗口内部显示的是一系列场景，每个场景对应一个高级对象视图，如图 4-4 所示。高级对象视图中的项目从左到右依次对应视图控制器（View Controller）、场景中的 First Responder 以及该场景的退出 segue（Exit）。

所谓“场景”，套用的是戏剧的概念，对于用户来说，他们看到的一个“界面”就是一个“场景”。场景里面，各种不同的“控件”做着“演员”的工作，它们进行不同的“演出”以便能够满足用户的需求。Storyboard 中每一个白色的矩形方格就代表了一个场景，场景和场景之间可以通过“跳转关系”来彼此连接。简单一点可以将场景理解为一个

“Xib”文件，毕竟 storyboard 文件是 Xib 文件的集合。

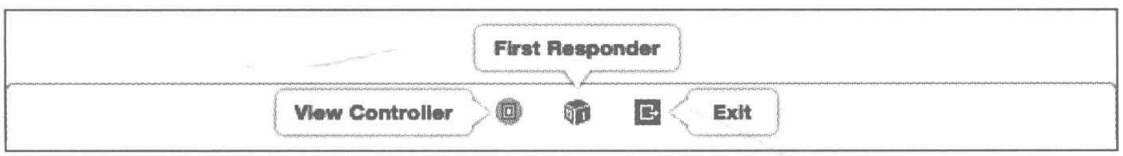


图 4-4　高级对象视图

至于视图控制器，主要涉及 MVC（Model-View-Controller）架构。关于 MVC 架构的相关内容，在第 8 章会有相应介绍。

first responder，则定义了界面加载后，哪一个演员最先开始准备响应用户的交互动作。最常见的场景就是用户名和密码文本输入框，用户名文本输入框一般都是 first responder。

退出 segue 则是退出这个场景的“出口”。关于 first responder 和退出 segue 的用法，在后面会介绍。

4.2.2　对象窗口

在对象窗口中，你可以看到 Xib 和 Storyboard 文件内的“场景”“布局”等对象及其具体结构。如果“对象窗口”没有显示，可以点击画布左下角的图标来开启对象窗口。

对于 storyboard 来说，对象窗口往往如图 4-5 所示。

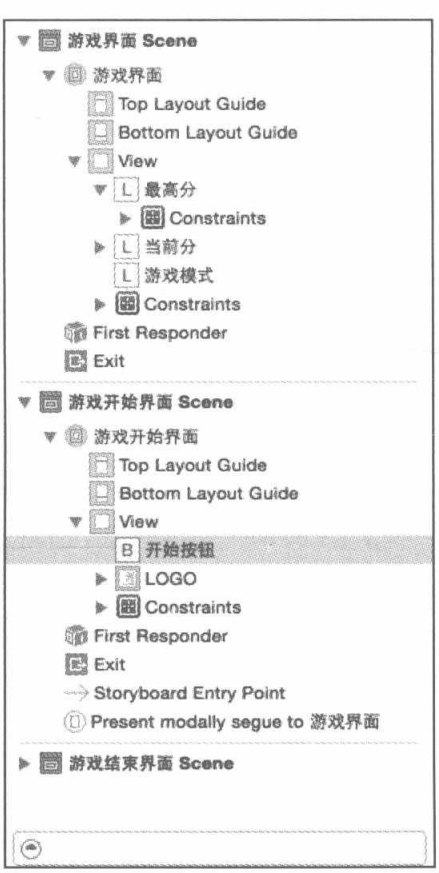


图 4-5　Storyboard 对象窗口

可以看出，storyboard 的顶级项目就是上一节介绍过的“场景”（）。在 CrazyBounce 当中，我们使用了三个场景，分别作为游戏开始界面、游戏界面和游戏结束界面。

每个场景中包含了视图控制器、first responder 和退出 segue。如果添加了入口点的场景，还会有一个“ Storyboard Entry Point”的入口点标识，这个标识表明这个应用将会从哪一个场景开始显示。如果还添加了 segue 的场景，那么还会有相应的标识。关于“ Segue”的内容，4.3.3 节会有介绍。

对于视图控制器来说，它当中还会有三个重要组成部分：Top Layout Guide、Bottom Layout Guide 以及 View。Top Layout Guide 和 Bottom Layout Guide 是占位的视图块，用在使用 Auto Layout 约束的时候，作为对齐的标杆。关于 Auto Layout 的相关内容，第 5 章会有更多介绍。

相比而言，Xib 的对象窗口要简单很多，如图 4-3 所示。一般而言，Xib 的 File’s Owner 就类似于视图控制器，做着类似的工作，提供代码和视图的转接工作。

我们套用设计师最熟悉的 Photoshop 概念来进行对比，可以发现，每一个场景就相当于一个 psd 文件中的画布区域，设计师可以在这个画布当中进行设计。Xib 文件就相当于一个单独的 psd 文件，而 Storyboard 文件相当于一堆 psd 文件的集合，并且它们之间有逻辑连接。

对于设计师来说，位于视图控制器当中的 View 部分就是作为基底的最重要图层了，设计师所有的设计元素都是在这个图层之上完成的。而视图里面添加的控件，就好比 Photoshop 中的文字工具、矩形工具等，一个设计图由这些元素组成，自然一个真实的应用由控件组成。关于控件的更多内容，请参阅 C.4 节。

4.2.3 检查器

在检查器中，我们可以方便地查看和编辑界面控件和对象的属性，如图 4-6 所示。就好比在 Photoshop 中，对某个元素进行属性设置一样。

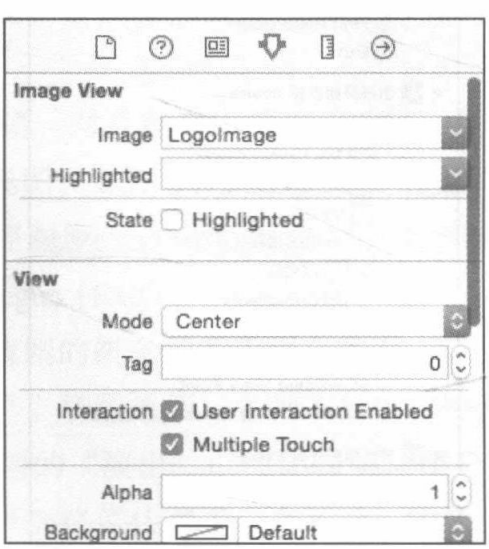


图 4-6 检查器

一般而言，界面生成器的检查器都包含了以下几个检查器。

- 文件检查器（File inspector）：第 2 章已经介绍过。
- 快速帮助检查器（Quick Help inspector）：第 2 章已经介绍过。
- 标识检查器（Identity inspector）：这个检查器主要定义该视图、控制器或者控件元素的相关自定义标识，诸如自定义类、标识符、运行时属性等内容。

属性检查器（Attribute inspector）：这个检查器定义视图、控制器或者控件元素的属性，比如显示模式、背景颜色等各式各样的属性，实现元素的自定义需求。

- 尺寸检查器（Size inspector）：这个检查器定义视图、控制器或者控件元素的尺寸，比如大小、位置以及约束等一系列信息。为了方便开发人员，iOS 中统一使用点（Point）对界面元素进行描述。对于普通屏幕，1 点＝ 1 像素；Retina 屏，1 点＝ 2 像素。
- 连接检查器（Connections inspector）：这个检查器主要定义连接口的相关设置。

检查器下面的组件区域在 2.7 节和附录 C 中均有详细介绍。

4.3　Xib 文件

使用 Xib 进行 iOS 应用的界面管理时，使用 IB（Interface Builder）只能对每个界面进行单独管理，界面之前的逻辑关系需要开发程序员来牢记，如果界面过多，那会是一个非常复杂的关系。

使用 Xib 来组织用户界面的好处在于，所见即所得，可以帮助开发者节省大量的编码时间，加快开发速度。Xib 不同于微软的 Visual 可视化界面，它的工作范围远比你想象到的大得多。自 Xcode 4 之后，Xib 就成为 Xcode IDE 当中密不可分的一部分了。

Xib 出现的目的是为了更好地让视图和控制器分离，因为一般来说，单独的 Xib 对应一个视图控制器，此外，Xib 也可以单独对应一个 view，这样可以帮助开发者满足 MVC 的开发要求。

自 Xcode 5 之后，苹果大幅度简化了 Xib 文件的格式，让其变得十分易读和维护。Xib 和安卓的界面设计类似，都是用 XML 来实现的。可以用文本应用打开 Xib 文件，从中可以轻易地读出其格式。所以，现在对于 Xib 文件来说，其和纯代码在版本管理上已没有太大的差异了。

不过 Xib 也不是完美的，因为 Xib 界面很有可能被另外的界面给替代，所以 Xib 界面一般适合于静态或者变动不是很大的 UI。因此，有一个原则是要遵循的，那就是：尽量将 Xib 的工作和代码隔离开来，能够使用 Xib 完成的工作就交给 Xib 来完成，而不要东一脚，西一脚，这样很可能导致 UI 显示结果不在你的预料之中了。

4.4 故事板

Storyboard（故事板）是现在 Apple 推荐的界面管理方式，我们可以在一个窗口中管理多个场景（Scene），多个场景之间的关系非常清晰，这极大地方便了开发者理清各个场景之间的逻辑关系。在 iPhone 或 iPod touch 上面，一个屏幕通常只包含一个场景。在 iPad 或 Mac 上面，一个屏幕可以由一个以上的场景组成。

Storyboard 的优点很多，但是缺点也显而易见，那就是多人协作。由于几乎所有的界面都完全放在同一个文件当中，这样会导致合并（Merge）的时候出现过多的错误。解决这个问题的方法就是将项目的不同部分分解成若干个 Storyboard。此外，Storyboard 对于自定义视图的处理功能也十分差，这是因为 Storyboard 更重视场景的层次，而不重视单个视图的高级功能。

在这个游戏中，我们用故事板来设置游戏运行时的所有场景。

4.4.1 添加新的场景

在 Main.storyboard 中，系统已经自动生成了一个默认场景（View Controller），我们用它作为游戏进行时的交互场景。此外，我们还需要添加两个新的场景，分别作为游戏结束和游戏开始时的界面。为了方便布局，我们先双击画布的空白区域，把画布的内容都缩小 50%。

添加新的场景很简单，从组件区域中找到合适的 Controller，把它拖动到画布上即可完成创建，如图 4-7 所示。有关各个组件的用法，2.7 节有详细介绍，在此使用 View Controller。

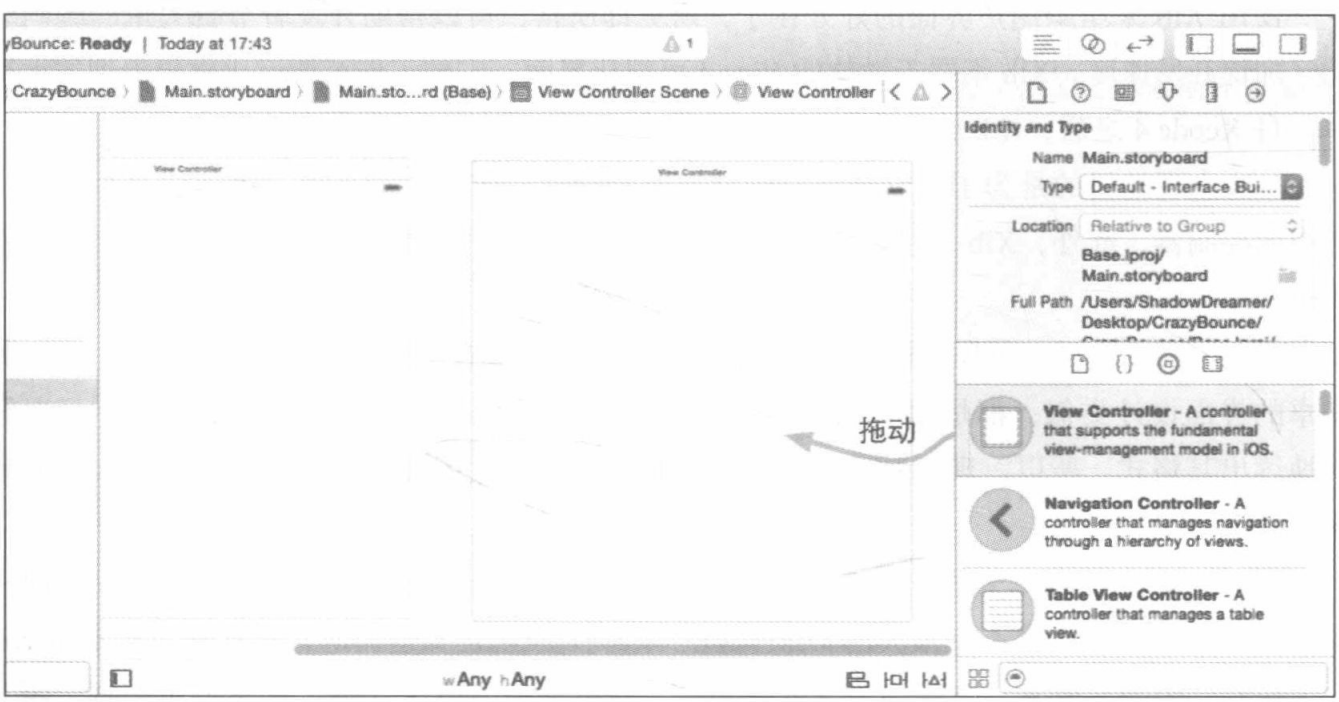


图 4-7 添加新的场景

提示　建议初学者打开 Storyboard 的文件检查器，然后在其中的 Interface Builder Document 栏目中，取消 Use Auto Layout 和 Use Size Classes 的选择，否则 Xcode 6 的默认布局样式可能会让你感到困惑。两者的用处会在后续章节讲解。

4.4.2　设置初始场景

初始场景是这个 Storyboard 被启动时首先显示的界面，也就是程序的 UI 入口。要修改初始场景，拖动画布的“起始标签”到新的页面即可。起始标签的样式如图 4-8 所示。

如果你找不到“起始标签”，还有一种通用的方法，选中你要设置为初始场景的 View Controller，打开属性检查器，勾选“Is Initial View Controller”即可，如图 4-9 所示。

图 4-8　起始标签

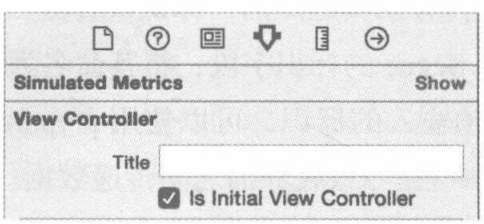


图 4-9　设置初始场景

4.4.3　添加页面间的转场

当我们需要从一个场景跳转到另外一个场景时，我们需要在场景之间添加“转场”（segue）。CrazyBounce 游戏一共有三个场景，载入界面（Launch Sreen）、游戏场景（Game Frame Controller）和“得分场景”（Game Over Controller）。载入游戏的时候，系统会自动显示载入界面“LaunchScreen.xib”；游戏载入完毕后，场景会跳转到 StoryBoard 的初始场景“Game Frame Controller”。

下面我们要添加转场，使得游戏结束后，场景跳转到“Game Over Controller”显示得分和分享界面；点击 Restart 按钮，系统又跳回“Game Frame Controller”重新开始游戏。操作步骤如下。

1）按住 Control 键将一个控件拖动到另外一个场景，如图 4-10 所示。

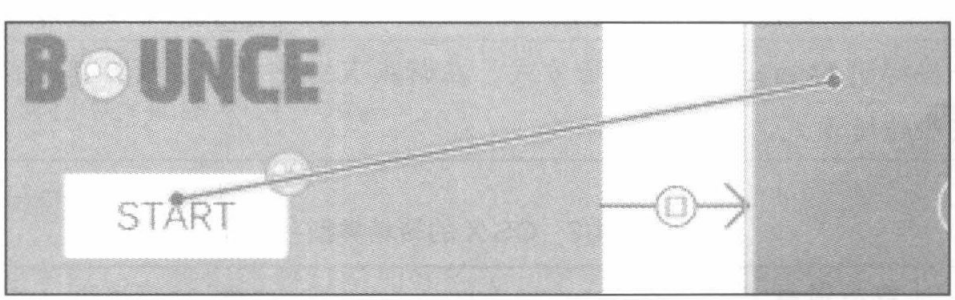


图 4-10　建立 Segue 连接

2）在弹出的菜单中选择 Segue 的方式，我们选择 Show 作为跳转方式，弹出的菜单如图 4-11 所示。

3）刚刚生成的 Segue 如图 4-12 所示，选择刚刚生成的 Segue。

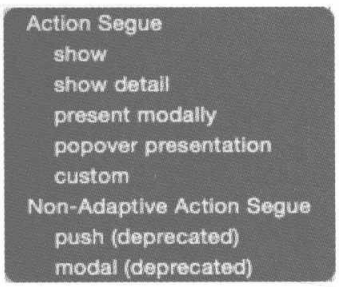


图 4-11　Segue 选择菜单

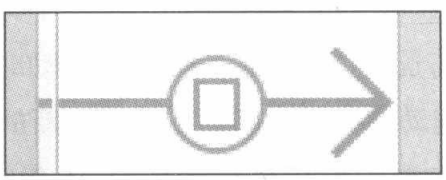


图 4-12　生成的 Segue

4）点击打开工具区域的“标识检查器”（Identity inspector）。

5）输入 Segue 的标识字段，将其命名为“gameStartSegue”。

通过刚才输入的标识，可以使用 prepareForSegue 或 performSegueWithIdentifier 方法来使跳转时执行一些指令或在界面之间传递数据。

转场类型有很多种，表 4-1 和表 4-2 分别介绍了 iOS 和 OS X 的转场类型。

表 4-1　iOS 的转场类型

转场名称	转场图标	介　绍
Show		根据当前屏幕中的内容，在 master area 或者 detail area 中展示内容。例如：如果 app 当前同时显示 master 和 detail 视图，内容将会压入 detail 区域。如果 app 当前仅显示 master 或者 detail 视图，内容则压入当前视图控制器堆栈中的顶层视图
Show Detail		在 detail area 中展现内容。例如：即使 app 同时显示 master 和 detail 视图，内容也将被压入 detail 区域。如果 app 当前仅显示 master 或者 detail 视图，那么内容将替换当前视图控制器堆栈中的顶层视图
Present Modally		使用模态展示内容。属性面板中提供（UIModalPresentationStyle）与（UIModalTransitionStyle）两种选项
Present as Popover		在某个现有视图中的某处使用弹出框展示内容
Custom		自定义转场

注意　还有一些标明 Deprecated 的转场方式，在新版 Xcode 中已经被弃用，我们要尽量避免使用这些转场方式。

表 4-2　OS X 的转场类型

转场名称	转场图标	介　绍
Show		在新窗口中显示内容

（续）

转场名称	转场图标	介　绍
Modal		在模态对话框中显示内容
Popover		在某个现有视图中使用弹出框展示内容，可以使用 NSPopoverBehavior 来控制弹窗的属性
Sheet		目前的内容作为一个 Sheet 贴附于原始窗口
Custom		自定义转场

4.5　配置界面

配置界面就是在画布里添加控件、图片等资源。

打开工具区域（见 2.7 节）。工具区域包含了检查器和组件区域。可以在检查器中设置对象外观和行为，在对象库和媒体库中查看构建 UI 的视觉、听觉元素。在工具区域中可以有如下操作。

- 添加对象：对象库和媒体库在工具区域里的右下角“库导航栏”中。点击对象库（）或媒体库（）按钮来选择资源的种类，拖动需要的元素到画布上即可完成添加。
- 改变尺寸：添加对象后，可以拖住其边角来改变其大小。移动对象时，会出现一些蓝色虚线来帮助定位和布局对象。
- 配置属性：在“库导航栏”上方是 Interface Builder 检查器，可以用它设置对象外观和行为，我们已经讲过每个检查器的作用。
- 设置委托对象：通过委托，在对象与对象、对象与代码之间传递数据。

4.5.1　添加对象和媒体

- 添加对象：打开“对象库”（Object Library）找到需要添加的对象并拖曳到画布的合适位置，如图 4-13 所示（具体每个对象的使用方法参见附录 C）。
- 添加媒体：参照 3.3 节的方法把需要的图片添加到“资源目录”里面，在这里添加到默认的 Images.xcassets 中。然后在“媒体库”（Media Library）中找到刚刚添加的图片，拖曳到画布中，如图 4-14 所示。

4.5.2　调整对象

对象添加完毕以后，我们需要对添加好的每一个对象的层次进行调整。首先，我们要学会使用左侧的“对象窗口”。在对象窗口中，我们可以方便地选择对象，调整对象的层次。如

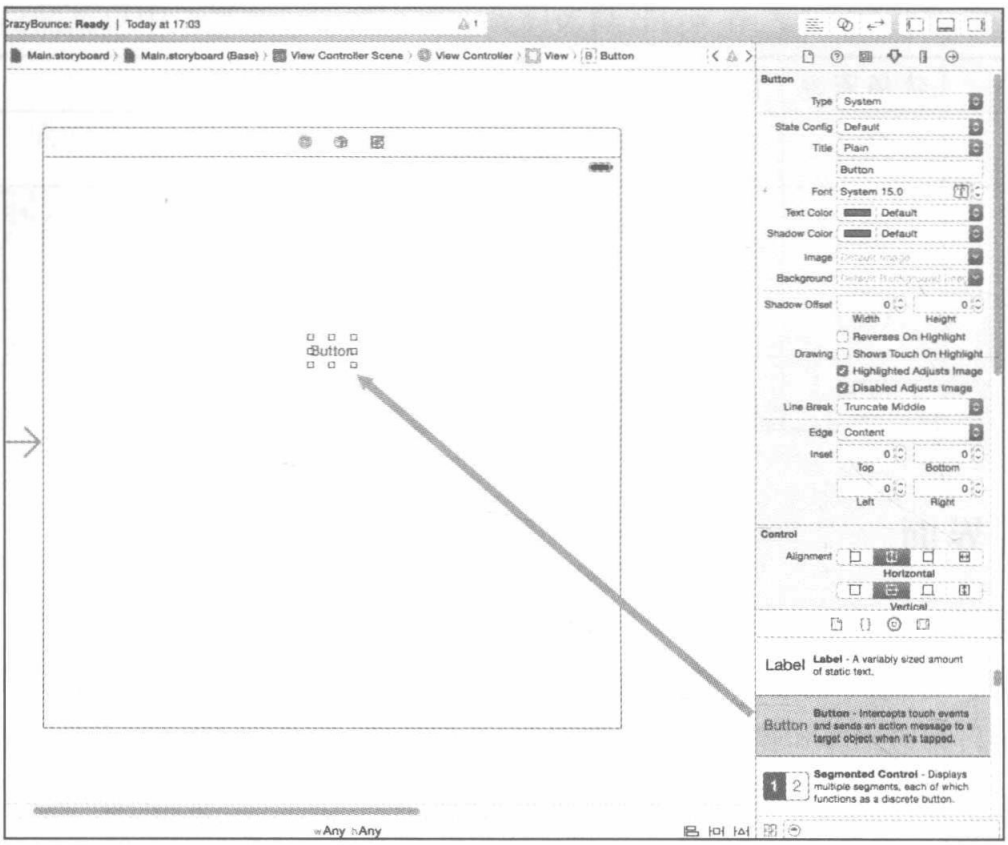


图 4-13　添加对象

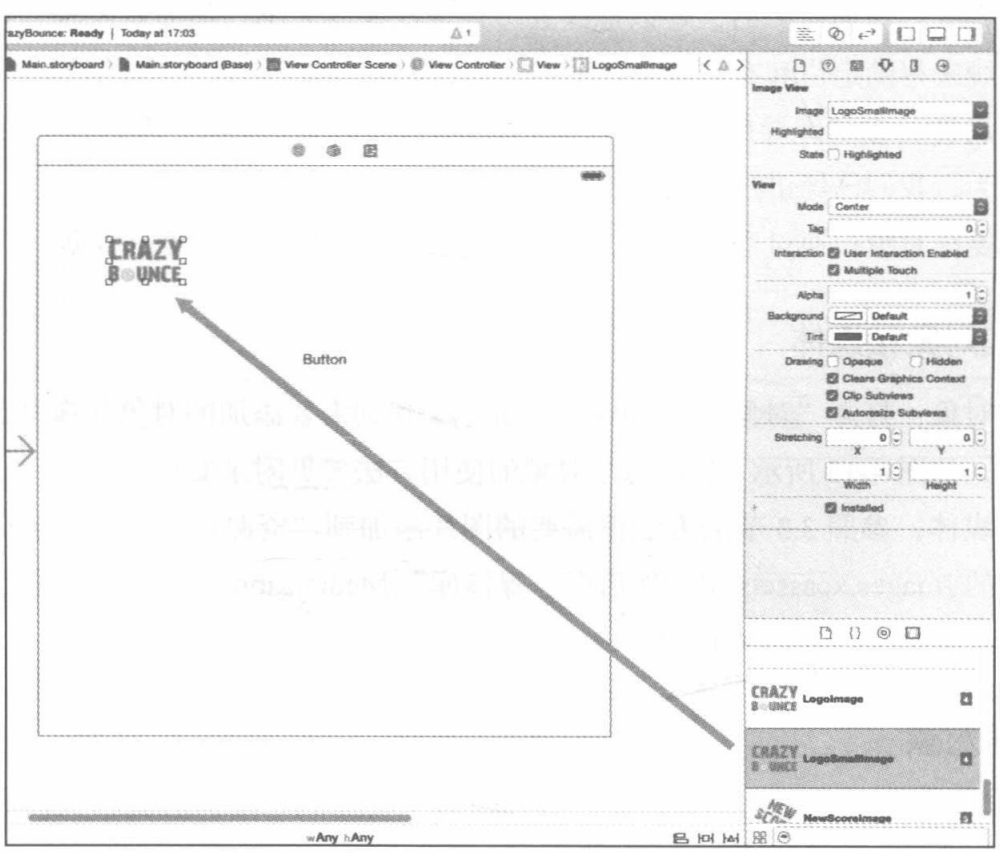


图 4-14　添加媒体

果你想要把控件向上层移动，那么需要把相应控件向下拖曳。

如图 4-15 所示，这样做将会把“继续按钮”放置在所有控件的最顶层，点击这个区域，最先选中的是“继续按钮”。

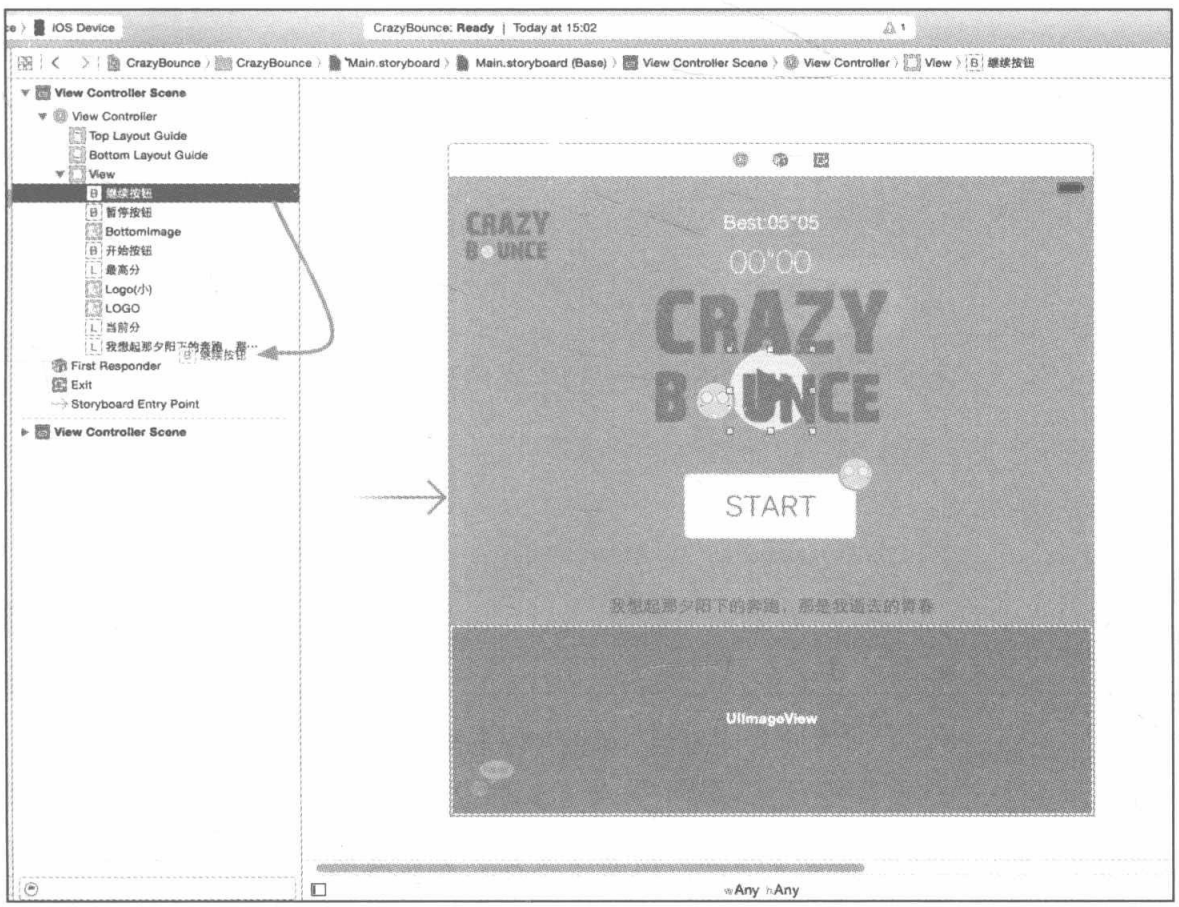


图 4-15　调整对象

4.5.3　配置属性

打开属性检查器，在画布或对象窗口中选择需要修改属性的控件。

比如说，我们更改刚刚拖进来的那个按钮控件，点击这个按钮，然后选择其属性检查器，如图 4-16 所示。我们对这个按钮控件的 Image 属性进行修改，就可以在这个按钮控件中显示一幅图片，点击这个图片也可以达到点击按钮的效果。

此外，通过修改 Title，可以修改按钮上面显示的文字，修改 Font 可以修改字体。关于 Button 的更多介绍，请参阅本书附录 C.4 节。

良辰听后如梦初醒，恍然大悟。白衣老人哈哈一笑：“少侠，请随老夫前来！”

二人随即来到了山崖旁的一处亭榭，此地银装素裹，翠松挺立，放眼望去，那蜿蜒起伏的群山，就好似被狂风卷起的海浪，连绵不绝。令人惊奇的是，这满天的雪花，竟无一朵飘进这亭榭内。

图 4-16　修改按钮属性

老者一拍衣袖，言道："学会了布阵，这苍穹天地，无论是这九天玄引，还是这天都之宝，皆听你号令，为你所用！正好，此处正有一茶一棋可供你我二人消遣，不如，就借此机会让少侠感受一下阵法的神奇之处，可好？"

良辰眼前一亮，连忙欠身："晚辈多谢前辈教诲！"

方圆中龙舞黑白，凌气间虎斗乾坤。这阵法玄妙之处，鄙人也是无从道来啊，只知晓那次，雪山之上风雪怒号，雷声滚滚，山川震动，不见日月。

第 5 章 Chapter 5

万物莫不有规矩——自动布局

技艺之下，万物莫不有规矩。动静有节，趋步商羽，进退周旋，自有尺寸法度规矩。

皋月亭，风雪怒号，少年良辰面对着黑白棋盘，正苦苦思索。老者轻捻白须，笑言道："少侠可曾发现，这几局下来，老夫这棋法可有什么特别之处？"

良辰心中一吟，缓然说道："前辈这棋法精巧得很，也绝妙得很。不过晚辈总有种似曾相识的感觉，仿佛前辈使得都是同一套路，然而却完全不同。上局藏龙卧虎，这局却龙飞凤翥，晚辈完全无力。"

老者哈哈一笑，言道："不错不错，少侠果然好眼力，我这几局使得全是同一套路。"良辰连忙拱手："还请前辈指教。"

"环境虽千变万化，阵法却不离其宗。这便是所谓阵法之'法度'，对阵法中每一棋子之位置产生一定的要求，苹果帮谓之'自动布局'，是阵法元素确认的一种手段。"

5.1 没有规矩，不成方圆

对于一个 iOS 应用来说，其用户界面组件由一个个相互独立的可视元素组成，这些元素有的负责输出界面（视图），有的感知用户的输入（控件）。这些元素最基本的做法是手动设置其在父视图中的坐标，但是如果确定了其相对于父视图的位置和大小，那么它的布局也就确定下来了，无论父视图怎么变化，它的位置和大小也都会随父视图变化而变化。

自动布局（Auto Layout）是 Xcode 提供的一种适配所有设备的技术，主要是给控件等元素添加约束，让它们保持与某一个元素的距离，从而能够唯一确认它的位置。比如将用户界

面比作一个阵法，只要确定了某个士兵与其周围士兵的相对位置，那么无论其周围士兵如何变化，这名士兵的位置仍然是能够确定的。

所以，自动布局可以帮助开发者适应不同的窗口尺寸、屏幕尺寸以及设备持有方向。要对界面中的元素进行具体定位，则要对其位置进行一定的约束，我们可以通过给视图对象添加关系约束来实现定位。例如，我们可以将一个图像居中放在故事板场景中，无论用户怎样旋转设备，图像将始终保持居中。

通过自动布局处理，界面中的元素会在以下情形中自动改变大小和位置：

- 用户改变了设备朝向。
- 用户在 Mac 应用中改变了窗口大小。
- 内容尺寸改变（例如字符串长度改变）。

如果要在界面生成器中使用自动布局，那么请打开故事板或者 Xib 文件，确保其文件检查器中的“Use Auto Layout”选项选中。

5.2 约束种类

约束分为两类：一类是对齐（Align），一类是布局（Pin）。下面分别介绍。

1. 对齐约束

对齐约束的设置如图 5-1 所示，它主要用于设定元素的对齐方式，包括边缘对齐（Edges）、中心对齐（Centers）和容器中心对齐（Center in Container）。其中，边缘对齐和中心对齐需要选中多个元素才能执行，而容器中心对齐只需要对单个控件执行即可。

图 5-1 对齐约束

对于边缘对齐来说，又分为：左侧边缘对齐（Leading Edges）、右侧边缘对齐（Trailing Edges）、顶端边缘对齐（Top Edges）、底部边缘对齐（Bottom Edges）。这些选项用于让选中的多个元素的边缘进行对齐，一般以最后选中的元素边缘为准。

对于中心对齐来说，又分为：水平中心对齐（Horizontal Centers）、垂直中心对齐（Vertical Centers）和基准线对齐（Baselines）。这些选项用于让选中的多个元素的中心进行对齐，一般以最后选中的元素边缘为准。基准线对齐则会让元素按照最后选中元素的基准线排列。

容器中心对齐则相对比较简单，只分为容器水平居中（Horizontal Center in Container）和容器垂直居中（Vertical Center in Container），这些选项会让选中的元素在父视图中居中。

2. 布局约束

布局约束设置如图 5-2 所示，它主要用于设定元素的具体位置，包括边缘位置、尺寸以及纵横比等。

图 5-2　布局约束

边缘位置（Spacing to nearest neightbor）指的是该元素的边缘与另外某个元素边缘的距离。它还拥有一个选项：Constrain to margins，这是 iOS 8 之后提供的 API，margin（边缘）指的是屏幕周围的一个特殊的空白区域，一般为 16px。勾选这个选项后，距离屏幕边缘的距离会相应地减去这个 margin 的值。

尺寸包括 Width（宽）和 Height（高），指的是该元素的尺寸大小，可以设定为具体数值。此外，还可以对多个元素设定宽高相等，这使用 Equal Widths 和 Equal Heights 选项来设定。

纵横比指的是该元素的宽高比。Aspect Ratio（宽高比）选项勾选后，会让元素无论怎么变化，都始终保持宽高的比例。

此外，布局约束还拥有一个选项 Update Frames。这个选项决定了当该约束成功添加后，界面生成器的界面更新行为。默认是 None，也就是需要手动更新，才能显示刚刚添加的约束效果；还有另外的选项，如 Items of new constraints 指拥有新约束的项目会被实时更新；All frames in Container 则是指所有的页面视图都会被更新。

5.3 添加约束

按住 Control 键选中当前视图元素并在内部拖动鼠标，或者拖动鼠标到另一个视图元素上都可以添加约束，如图 5-3 所示。

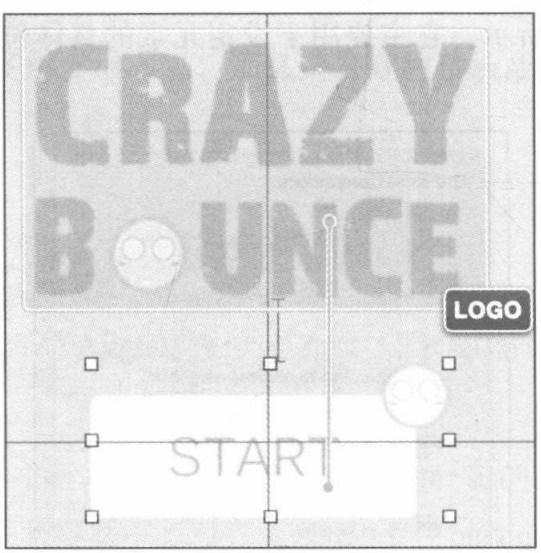


图 5-3 添加约束

在控件内部拖动鼠标添加的是“独立约束”，也就是尺寸、纵横比、容器中心对齐等只需要单独视图元素就能实现添加的约束。而拖动鼠标到另一个控件上添加的是“相对约束”，也就是需要多个视图元素才能够实现的约束，比如边缘位置、中心对齐，等等。

横向拖动鼠标可以显示水平方向上的约束，纵向拖动鼠标会显示垂直方向上的约束，斜着拖动鼠标则两个方向上的约束都会显示出来。拖动一个约束到另一个控件上，如果这两个控件可以添加约束，那么第二个控件会以蓝色高亮显示。

拖动鼠标完毕后，控件上方会弹出一个设置界面让你选择想要的约束。由于我们是纵向拖动鼠标建立的约束，因此出现的是 Vertical Spacing（垂直间距）设置界面，如图 5-4 所示。图中的选项可用于设置“ START”按钮和“ Crazy Bounce”的 LOGO 图片之间的距离。在此还可以选择“相对约束”，比如对齐、等宽、纵横比等。

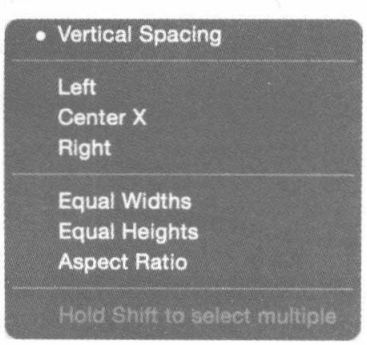


图 5-4 选择要添加的约束

如果不想在画布上拖动鼠标，那么也可以在界面生成器左边的输出视图中拖动鼠标来实现相同的功能。添加好的约束会显示为一

个点（·）。

当要添加多个约束时，按住 Shift 键可以选择多个约束。

添加完约束后，会显示刚刚添加好的约束。在画布上，蓝色的实线代表有效的约束，黄色线代表不完整的约束，红色线为有冲突的约束。

此外，还有一种方式就是选中当前控件，然后单击界面生成器右下方的按钮，如图 5-5 所示，弹出的对话框也如图 5-1 所示，在其中添加约束即可。

图 5-5　添加约束按钮

上一节已经说过，布局约束中的边缘位置约束是最常用的约束之一。当选中某一个控件之后，单击其四周的“I”字，它们就会从淡红色变为大红色，这样就表明了该控件将会与其某一侧的某个元素的边缘建立联系。通过单击下拉菜单，可以选择与之建立联系的元素，然后修改里面的数字即可完成约束的设定。

注意　在修改数字之后，一定不要单击另外的“I”字，否则下拉菜单中的约束值会变回原来的数值。

5.4　查看约束

选中相应对象后，在“尺寸检查器”（  ）中会显示完整的约束，我们可以查看或编辑这些约束的具体行为，如图 5-6 所示。

找到你想要编辑的约束，双击它就可以打开详细的约束定义。如果想要快速编辑约束，点击 Edit 按钮，在弹出的快捷编辑窗口中编辑即可，如图 5-7 所示。

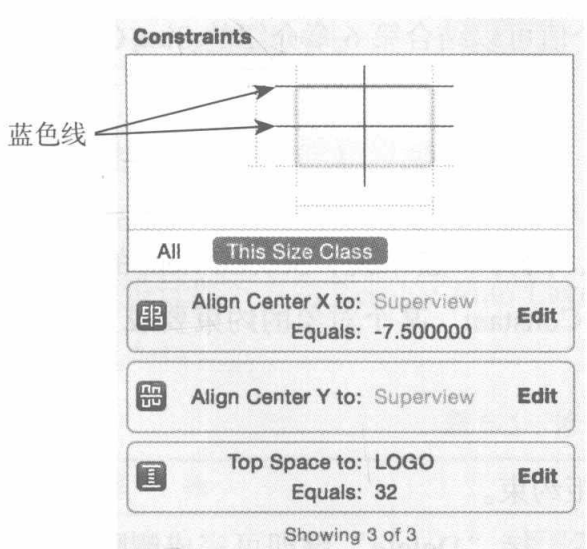


图 5-6　尺寸检查器中的约束

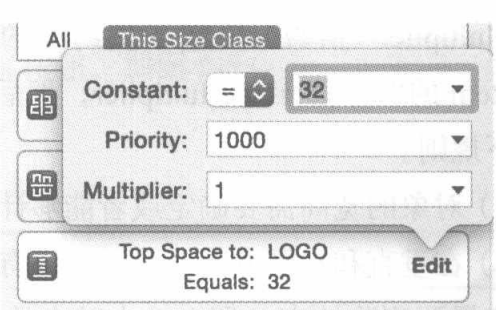


图 5-7　快速编辑约束

我们可以对约束进行过滤，约束界面的顶部显示了当前选中对象的所有约束。已添加的约束以蓝色线显示，通过单击蓝色线条可以快速过滤所选的约束类型。无文本的控件约束显示了大小和位置，有文本的控件约束则相比之多了一个文字的基准线。

选中界面生成器中的具体约束线，还可以查看约束更为具体的信息，如图 5-8 所示。

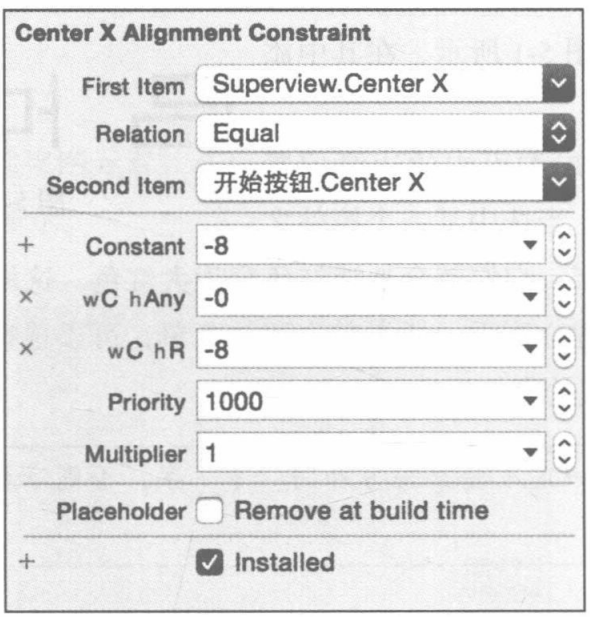


图 5-8　约束信息

First Item 和 Second Item 决定了约束所联系的对象，这里表明该约束联系了父视图中心的 X 坐标，以及开始按钮中心的 X 坐标。

Relation（关系）表明了所联系的对象之间的关系。这里的选项是 Equal（相等），表明这个开始按钮中心的 X 坐标和父视图中心的 X 坐标相等，即 5.2 节介绍的“容器水平居中”。此外，还可以选择“Less Than Or Equal”以及“Greater Than Or Equal”选项。

Constant（常量）决定了约束的实际值，这个值可以结合第 6 章介绍的 Size Classes 功能一同使用。

Priority（优先度）决定了该约束的优先值，如果 Xcode 检查到有两个约束作用类似，那么会检查两者的优先度，优先度高的约束将会被启用，低的将会被禁用。

Multiplier（倍数）决定了该约束的实际值，因为一般来说，Second Item 的实际值等于 First Item 的实际值乘以 Multiplier，然后再加上 Constant。某个对象的约束要成立，必须要满足以下规则。

1）对象的宽高需要确定或者能够计算出。

2）在垂直和水平方向，都至少要有一条间距约束。

如果要删除约束，那么选中某个约束线之后，接“Delete”键即可完成删除操作。

5.5　所谓“空白”

所谓“空白”就是占位符，设置占位符的内在尺寸表示自定义视图的宽度和高度，避免设计时间约束的模糊。设置占位符的步骤如下。

1）在 Interface Builder 中，选择自定义视图。

2）选择 View → Utilities → Show Size Inspector。

3）在弹出菜单中选择“Placeholder”。

4）设置合适的宽度和高度。

不同于标准的视图，自定义视图没有定义内在内容大小。在设计时，界面生成器不知道自定义视图的大小，所以设置占位符内容大小显示自定义视图的内容大小。如果不设置占位符的内容大小，视图可能产生意义不明确的布局约束，在界面生成器中会用橙色边框图显示。

5.6　修正约束错误

5.3 节提到过，在界面生成器中，如果出现错误的约束，会以特殊颜色显示。黄色线中显示的是意义不明的约束，红色线显示的是有冲突的约束，如图 5-9 和图 5-10 所示。在视列表或画布上选择约束可以进行修正。

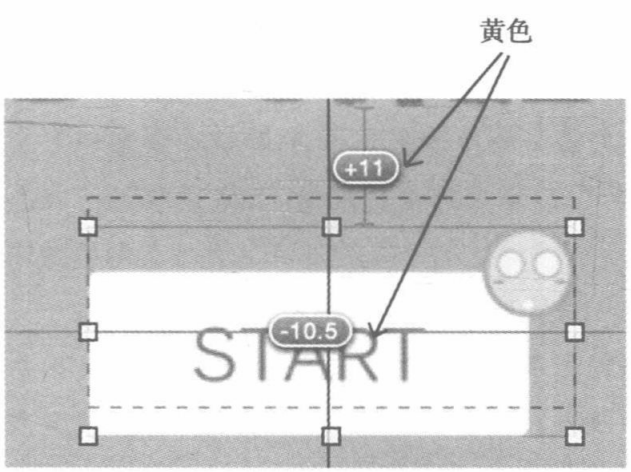


图 5-9　意义不明的约束

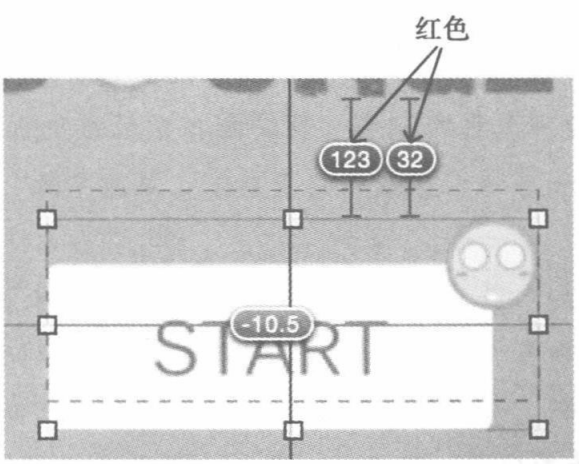


图 5-10　冲突的约束

点击界面设计器下方的 Resolve Auto Layout Issues 按钮，即图 5-5 中第三个按钮，修正操作的含义见表 5-1。

表 5-1　修正操作的含义

操　作	含　义
Update Frames	更新所选项目，使之匹配属性检查器中的约束
Update Constraints	改变属性检查器中的值，使之匹配所选的项目

（续）

操　作	含　义
Add Missing Constraints	给选择的项目添加必要的约束
Reset to Suggested Constraints	把约束重置为推荐的约束值
Clear Constraints	删除选定的项目布局约束，然后定义新的约束

我们可以看出，图 5-9 中会产生意义不明的黄色约束，是因为当前按钮控件的位置和约束不对应，其位置和上面那个视图的约束规定位置差了 11px 的距离。要解决这个问题，只需要更新所选项目的选项，更新一下视图即可。

对于图 5-10 上的红色约束，是因为我们添加了重复的当前按钮和上面视图之间的距离约束，Xcode 不知道要使用哪个，所以就产生了冲突。要解决这个问题，要么更改其中某个约束的优先度，要么选中该约束，删除掉即可完成错误修正。

万物皆有规则与尺度，只有将各部分放置于阵法中恰当之处才能充分发挥战力。在精妙的排兵布阵中，壮士出鞘一步外，平风破浪，引风雷，便无人敢挡；设防盾围内，铁壁铜墙，势如虹，还有谁人敢逞强呢？所以，习得自动布局一招，乃今后重要的御敌技巧，让你能够在编程江湖中更为自由地驰骋。

第6章 Chapter 6

万法归一——屏幕分类

檐下半昏黄，天边正夕阳。少年良辰伏在案前，手捧着眼前这本长卷面露难色。此时，白衣老人随着一阵温柔的清风又来到良辰的身边，他轻轻卷起了自己的袖角，一股书卷香气迎风而来。“少侠，休要偷懒！漫漫侠客路，耐得寂寥潇洒者，自当胸有万卷书，举身便可倾付江河！”说罢，他便随手一拿，将几滴墨汁洒于宣纸上。

良辰双目凝视，只见纸上玄妙阵法逐渐显现。几滴清墨的描摩之下，宣纸之上竟腾跃起苍穹一般令人惊叹的阵法，正所谓“墨生丹青花非花，帝托河洛画似画。天地由我千机变，也歌也颂迹天涯。”良辰心中的崇敬之意油然而生：“敢问前辈，这般复杂阵法怎得以在一张翩翩薄纸上显现？”

老者缓缓地捋了捋胡子，呵呵笑道：“老夫不过是在书中习得了一种绝密的技艺。我们在布置阵法的时候，无需再顾虑这阵法的规模大小问题，只需将阵法分为几种类型，再对于这几种类型的阵法使用自动布局进行规划即可。只要布置阵法的时候使用了这个秘诀，那么也就不用担心阵法必须要固定的大小才能够正常运作了，只要符合总结出的类型，天地之大，都可以拿来布置阵法。”

老者说罢，良辰依旧一副不太了悟的模样，于是他又接着说道：“多说无益，少侠还是看这秘籍罢了。”

话音刚落，老人旋即消失在了少年的眼前。向他隐没的方向望去，只见几缕烟丝在空中飞舞盘旋，好似一只在天边起舞的圣洁白鹤。良辰轻轻打开了手中的书，字字清晰可见，重重夜色已包围而来，皎洁月色绕梁，万籁俱寂。

6.1 为了适配，也是蛮拼的

在 iOS 8 以前，为了让一款应用能够同时适配 iPhone 和 iPad，项目中往往会使用两个 Storyboard，或者数目更多的 Xib 文件。随着 iPhone 6 的推出，iOS 开发的适配问题变得越来越困难，这样势必会导致开发者们倾向于脱离 Storyboard，专用 Xib 甚至纯代码的编码方式。这无疑是苹果公司不想看到的。

怎么办呢？ Xcode 6 带来了 Size Classes（屏幕分类）。

屏幕分类是一个奇思妙想，它对 UI 进行了一次全新的抽象：将各个设备的屏幕尺寸及其旋转状态都抽象成为屏幕尺寸的变化。简单来说，就是将各个设备和旋转状态分为以下几种类别。

Width（宽）：Regular（正常）、Any（任意）、Compact（紧凑）

Height（高）：Regular（正常）、Any（任意）、Compact（紧凑）

这几种类别任意组合，可以组合成 9 种不同类型，如图 6-1 所示，这也是设定 Size Classes 的主要界面。不过实际上，设备的尺寸只能够组合出 4 种类型，也就是正常和紧凑的相互组合。任意意味着无视这个分类条件。

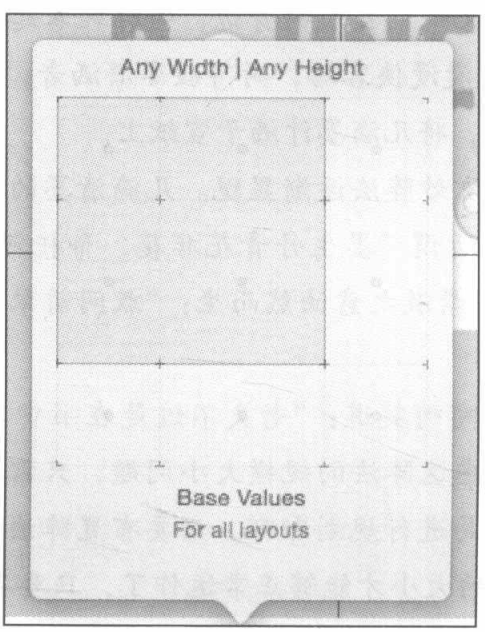


图 6-1 Size Classes 类型

在继续介绍 Size Classes 的概念之前，本书先把 iOS 设备及其方向的对应列表展示出来，以便大家能够理解 Size Classes 大致的作用，这样才能更好地理解它的概念。对应列表如表 6-1 所示（4S 以下的设备抛弃掉吧，它们已经不支持 iOS 8 了 TAT）。

如果不进行指定，默认情况下界面采取的是宽（任意）和高（任意），即 Any Width/Any Height 模式。此外，使用 Size Classes 之后，界面就要采取 Auto Layout（自动布局）来进行布

局，否则会成倍提高计算量。

表 6-1　Size Classes 与设备的对应列表

设 备 名 称	屏幕像素尺寸	屏幕分辨率比例	对 应 分 类
iPhone 4S	640×960	@1x	Compact Width | Regular Height
iPhone 5/5S	640×1136	@2x	Compact Width | Regular Height
iPhone 6	750×1334	@2x	Compact Width | Regular Height
iPhone 6 Plus	1242×2208	@3x	Compact Width | Regular Height
iPhone 4S(横屏)	960×640	@1x	Compact Width | Compact Height
iPhone 5/5S(横屏)	1136×640	@2x	Compact Width | Compact Height
iPhone 6(横屏)	1334×750	@2x	Compact Width | Compact Height
iPhone 6 Plus(横屏)	2208×1242	@3x	Regular Width | Compact Height
Retina iPad	1536×2048	@2x	Regular Width | Regular Height
iPad Mini	768×1024	@1x	Regular Width | Regular Height
iPad ½	768×1024	@1x	Regular Width | Regular Height
AppleWatch	312×390	@2x	Compact Width | Compact Height

启用 Size Classes 之后，Xcode 会自动给 Storyboard 另外创建 8 个子文件，分别对应各种类别的选择。这些文件采取的是延时加载（LazyLoad）模式，只在特定的 Size Class 配置过的元素中添加一条描述信息，描述这个元素在哪个 Size Class 下做了哪些改变。

这样做的好处是，约束、控件对于 Size Class 来说是相互独立的，不同 Size Class 之间的约束、控件不会互相影响（Any 除外，Any 会覆盖 Regular 和 Compact 的效果）。

启用 Size Classes 后，Xcode 的画布将以正方形的形式显示。此外，Size Classes 还将会在画布底部的中心位置添加一个选择控件，如图 6-2 所示。点击这个控件可以进入不同的 Size Class 当中。

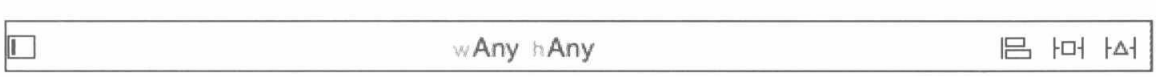


图 6-2　Size Classes 选择位置

如果要更改 Size Classes，点击下面的 Size Classes 控件，在弹出的菜单中选中你想要的 Size Classes，如图 6-1 所示。

6.2　激活这个技能

为了激活 Size Classes 这个技能，需要在文件检查器里面选中“Use Size Classes”这个选项，如图 6-3 所示（当新建项目时，Size Classes 是默认打开的）。

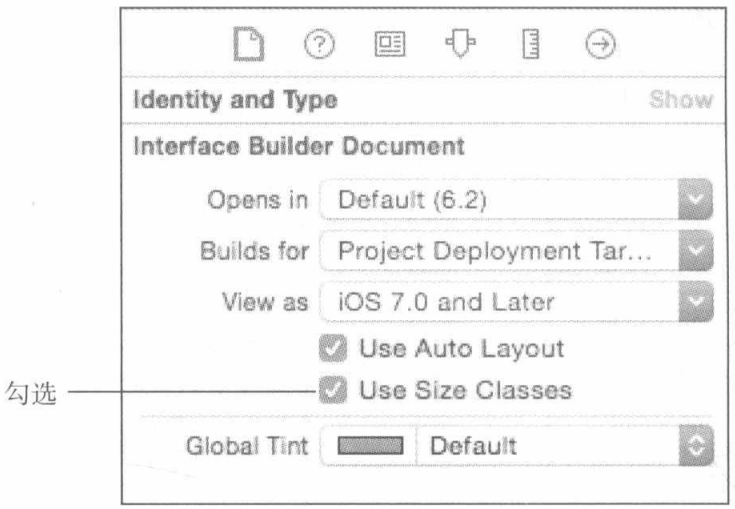


图 6-3 激活 Size Classes

在“项目导航器”（见图 2-3）中选择相应的 Storyboard 或者 Xib 文件，然后勾选“文件检查器”中的 Use Size Classes 复选框。Xcode 会询问你是否转换文件。打开 Size Classes 的同时会开启 Auto Layout。

如果不打算使用 Size Classes，那么取消这个选项就可以，Xcode 会弹出如图 6-4 所示的对话框，提示你选择何种设备作为转换后的尺寸。这里可以选择 iPhone 和 iPad。

图 6-4 取消 Size Classes

选择“Disable Size Classes”按钮后，即可关闭 Size Classes 功能。

激活了 Size Classes 之后，开发者就可以根据需求，前往不同的 Size Class 当中对元素进行相应的配置了，Auto Layout 仍然是最好的选择。

6.3 变更视图

Size Classes 可以完成很多奇妙的功能，最常见的功能就是利用 Auto Layout 来改变元素的大小和布局了。不仅限于此，Size Classes 还有许许多多的功能，下面接着介绍。

6.3.1　改变约束的值

可用 Size Classes 改变约束的值，这样可对不同的设备实现不同的布局。比如在 iPhone 6 下，一个 Button 是 200px 宽，但是在 iPad 下，想让它变成 300px 宽，那么只要跳转到 iPad 对应的 Size Class，修改约束的值即可。

选中需要修改的控件，然后选择需要修改的约束，接着打开尺寸检查器（见 4.2 节），如图 6-5 所示。

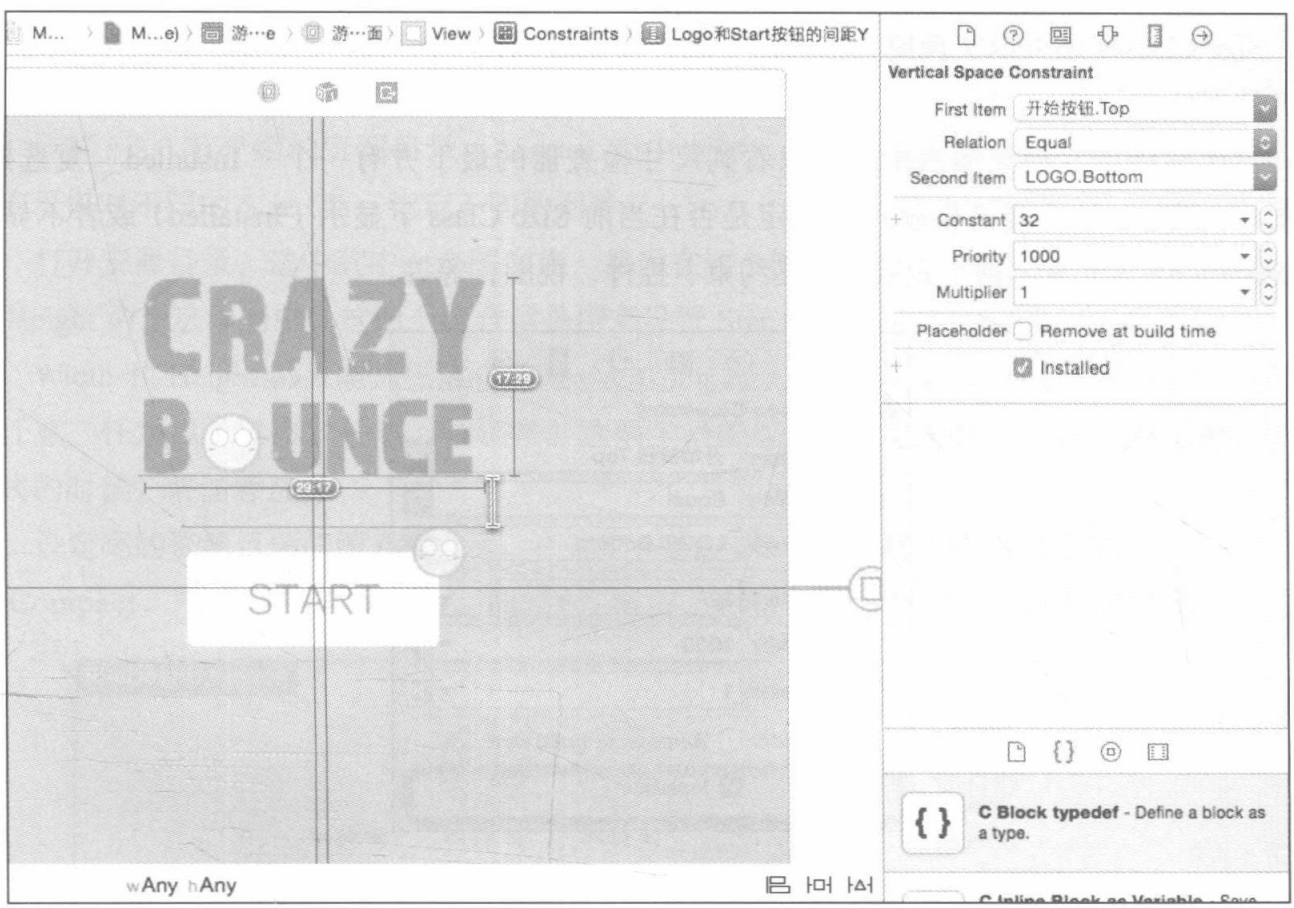


图 6-5　对应约束的尺寸检查器

然后选择宽（正常）和高（正常），即 Regular Width|Regular Height 模式，让这个约束能够适应 iPad 的尺寸。

进入这个 Size Class 之后，单击尺寸检查器中常量（Constant）旁边的“+”按钮，然后在如图 6-6 所示的弹出菜单中选择“Regular Width|Regular Height (current)”，也就是当前 Size Class。

选择之后，在新出现的输入框（wR hR）中输入想要的新约束值，按回车键确定即可完成约束值的修改。

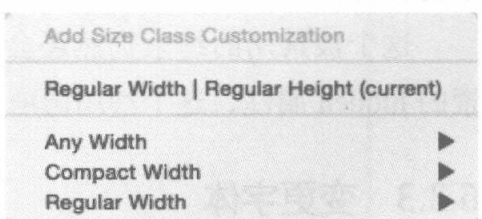


图 6-6　Size Classes 应用菜单

在 Size Classes 中，w 代表宽（Width），h 代表高（Height），c 代表紧凑（Compact），r 代表正常（Regular）。

还有一种更改方式是直接在尺寸检查器的弹出菜单中进行选择。选择弹出菜单下对应的 Size Class，比如依次选择 Regular Width → Regular Height。如果已经实现过 Size Class 它将会以灰色不可选状态显示。

6.3.2 启用、禁用元素

Size Classes 还可以实现更为强大的功能，那就是根据不同屏幕尺寸和屏幕方向，来启用或者禁用某个元素。

再次回到尺寸检查器当中，可以看到尺寸检查器的最下方有一个“Installed”复选框，如图 6-7 所示。这个复选框就用来决定是否在当前 Size Class 下显示（Installed）或者不显示（Uninstalled）该元素，这个元素可以是约束、控件、视图，等等。

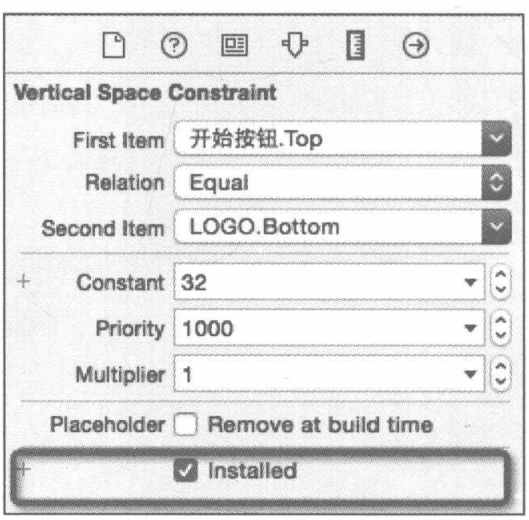


图 6-7 启用 / 禁用元素

和改变约束的值类似，我们可以先进入对应的 Size Class 中，单击旁边的“+”号按钮，选择合适的 Size Class 来应用这个选项。

对应 Size Class 的 Installed 复选框如果没有选中，那么当程序在符合这个 Size Class 状态下的设备上运行时，这个元素就不会显示在设备上，或者不能在设备上起作用。

这个被禁用的约束仍然还会生成，只不过它不会显示在视图里面。同时，它也不同于元素的 hidden 属性，这个禁用选项是不可更改的。

6.3.3 变更字体

Size Classes 还可以实现字体变更，实现在不同设备状态下显示不同颜色、不同字体的文字。对于 iOS 8 来说，Size Classes 支持以下控件的字体变更：

- UILabel
- UITextField
- UITextView
- UIButton

和改变约束的值类似，我们可以先进入对应的 Size Class 中，单击属性检查器中对应 Font 旁边的“+”号按钮，选择合适的 Size Class 来应用字体的变更。

6.4　资源目录

同样，3.3 节介绍的资源目录（Assets）当中也加入了对 Size Classes 的支持。也就是说，我们可以对不同的 Size Class 指定不同的图片。

打开资源目录，选中其中一个图像集，然后在这个图像集的属性检查器中，定位到 Width 和 Height 两个选项当中，这两个组合就是用来设置 Size Classes 的组合了。

Width 和 Height 都只能选择 Any、Any & Compact 以及 Any & Regular 三种选择，分别对应任意、任意以及紧凑、任意以及正常。这两个只能够拼凑出四种组合，因此在选择适合的模式的时候，要注意选择。

设定完的资源目录如图 6-8 所示。在这里，Size Classes 的属性改用符号来表示。“–”对应 Compact，“+”对应 Regular，而“*”对应 Any。比如，3x [* +] 就代表着宽 Any、高

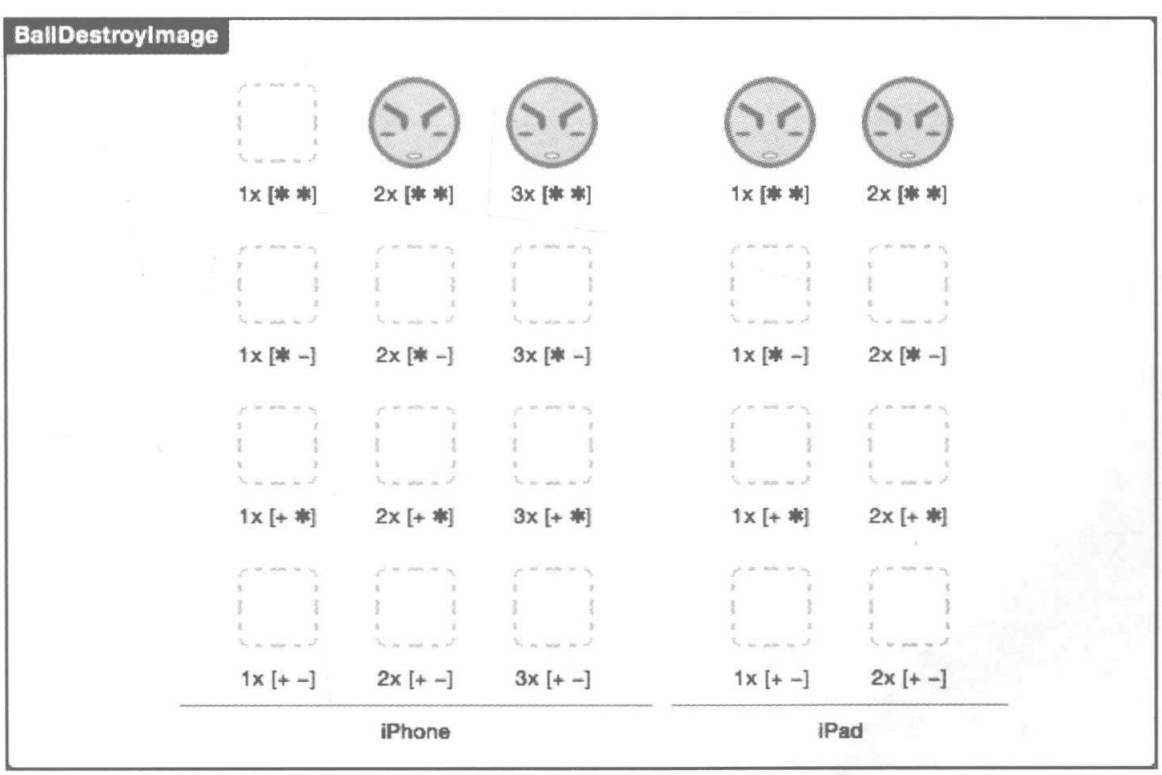


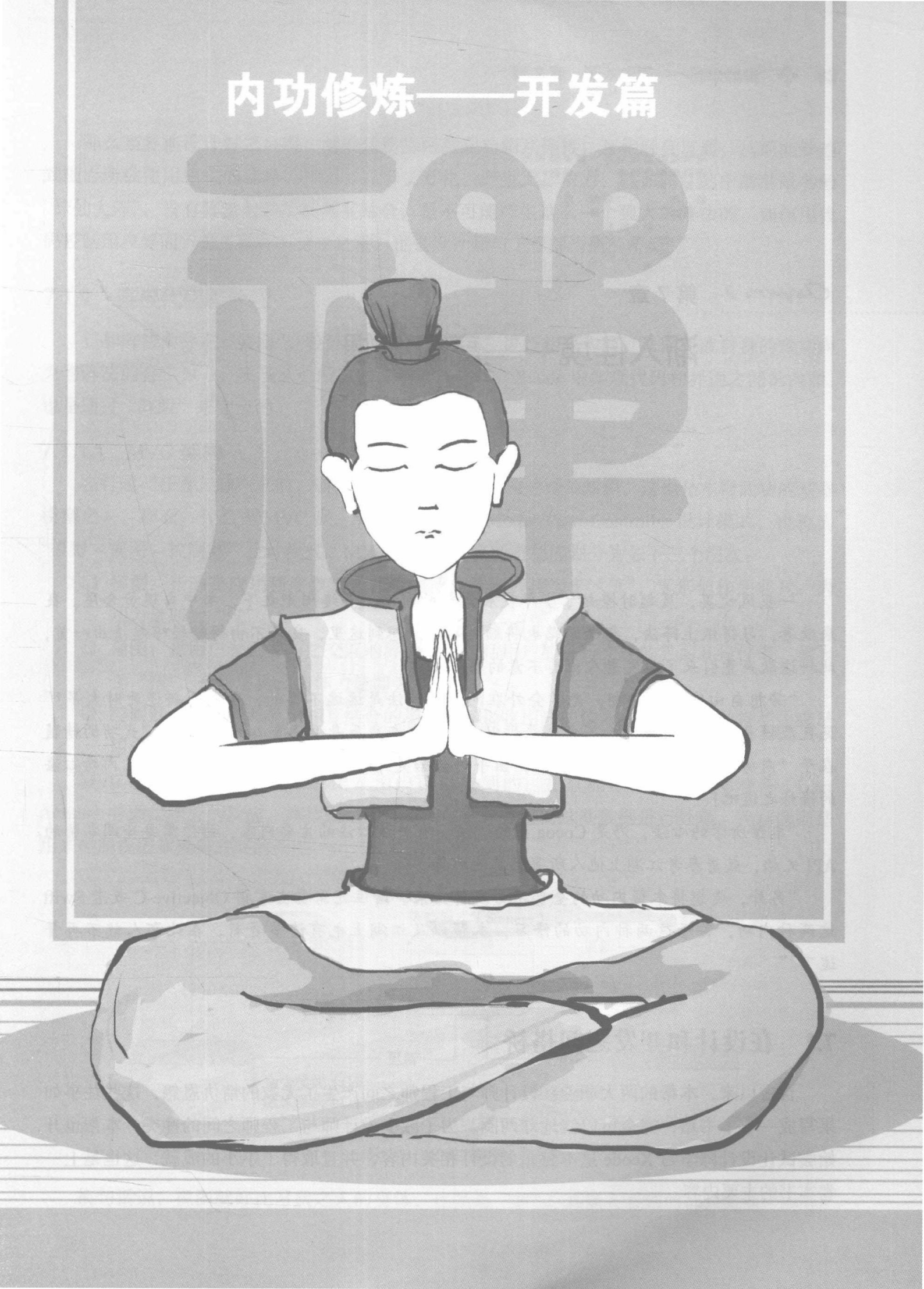
图 6-8　启用 Size Classes 的 Assets

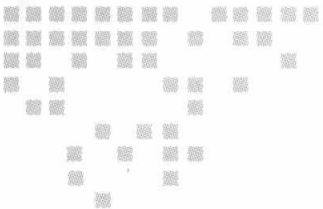
Regular 的设备状态对应的 @3x 图。

这一日，少年良辰捧书夜读，一直到翌日的清晨。他放下书卷，拿起窗台上的陈酒一饮而尽。在春日的早晨酌酒独饮的他，慢慢陷入了睡梦中。梦境里，是 9 月岚风谷中那一片广阔的星辰和撩人的夜风，他好似看见了那山峰上伫立着一个模糊的人影，手捧长卷，月下独饮。那人笑呵呵地看着他，手指向山下那一片灯火通明的村庄，那是编程的江湖，是无尽的旅途。

几度霜寒早花尽，些许浊酒侠胆沁。星游日转仗剑来，波凌浪涌踏云去。

内功修炼——开发篇





Chapter 7 第 7 章

渐入佳境——高级编辑

一夜风吹罢，晨起时撩起了少年良辰案前的窗帷。一缕阳光之下，书中自现黄金屋。良辰欣喜，习得纸上阵法，想必前路也将顺利不少。想到这里，良辰不由得轻轻哼起清曲一首，此料这歌声竟引来了白发老人出其不意的当头一棍。

“若想自由地闯荡江湖，光学会外在的那些阵法是远远不够的，现在高兴还为时太早！”见良辰摸着头直喊疼，老人不舍再严厉以待，便坐在良辰身旁语重心长地说道：“武功的精髓在于‘内功’。空有那些书上的阵法，出手仍会贻笑大方！今日，就让为师来你提点内功心法的修炼之道吧！”

“本帮所学的心法，乃是 Cocoa 系的心法，有关该心法的主要内容，诸位需要查阅本帮的 API 文档，或者参考江湖上他人所著的其他典籍。”

“另外，要想将本帮内功的全部功力发挥出来，诸位还需要去掌握 Objective-C 或者 Swift 这两种内功，关于这两种内功的修习，本帮以及江湖上也有诸多资料，在此鄙人就不再赘述了。”

7.1 在设计和开发之间搭桥

自古以来，本帮的两大职业：设计师与工程师之间产生了无数的情仇恩怨，这些往事如果写成一本本书籍，完全可以绕地球两圈。为了减少设计师与工程师之间的冲突，本帮也开始尝试让设计师学习 Xcode 这本秘籍的设计相关内容，并且取得了不小的成就，这也是上一卷本书的主要内容。

那么当界面设计好之后呢，就需要将阵法与武功相互连接，互相融会贯通，从而能够达到随心所欲使用招式的境界，并且可以千变万化，产生无尽功力，就如同江湖中赫赫威名的“诛仙大阵”，没有阵法与武功的相互结合，是不可能产生这么一个强大的阵法的。如何用代码控制用户界面，是工程师的重中之重，也是设计师与工程师沟通的桥梁。

7.1.1　连接代码和界面

正如阵法中存在“阵眼”，控阵者通过与“阵眼”进行相互沟通，从而完成自身的武功内力与阵法的合二为一，实现从心所欲的控制阵法一样，Xcode 中实现代码和界面之间的沟通，也是通过“阵眼”来完成的。

7.1.1.1　MVC 架构

在打通“任督二脉”之前，诸位需要了解 MVC 架构的相关知识，这也是本帮武功的基本模型之一。那么，什么是 MVC 呢，MVC 全称就是 Model-View-Controller 设计模式，也称为“模型 – 视图 – 控制器”设计模式。MVC 设计模式将所有功能划分为如下三个层次：

- **模型**：指的是应用程序数据及相应的业务规则，相当于“气”，无形但作用非凡，内功力量即为“气”。
- **视图**：是用户看到并与之交互的界面，其包括用户能进行交互的界面和控件等，相当于“精”，有形可见，肉体力量即为“精”。
- **控制器**：将模型和视图相互结合的场所，其接收用户的输入并调用模型和视图去满足用户的需求，相当于“神”，用于控制“精”和“气”。

MVC 模型如图 7-1 所示。视图是用户可以看到的，并且能够响应用户的动作（User Action）并做出相应的反应。模型则是用来封装数据，并且对这些数据进行处理。控制器则是用来控制视图和模型之间的调度。

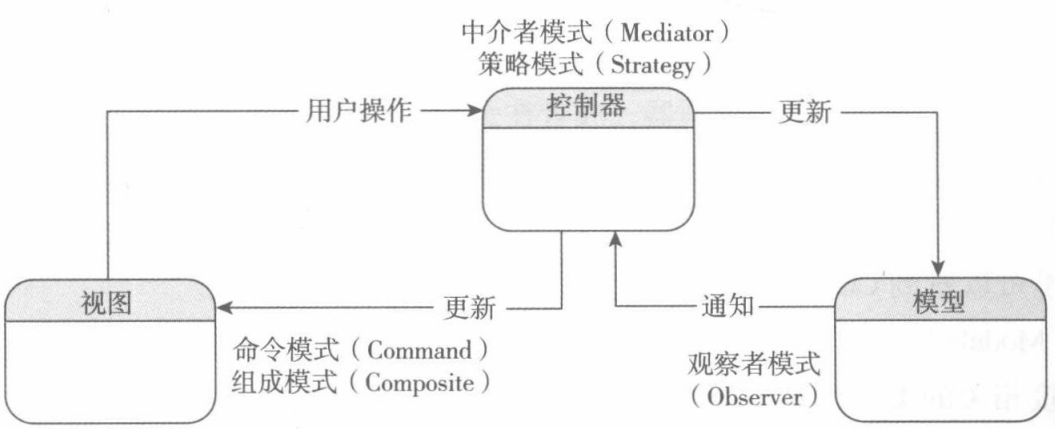


图 7-1　MVC 模型

换句话说，视图就好比习武之人的身体，也就是“精”。其他人看到的都只是这个身体，

别的内容都没办法看见。身体可以轻巧，可以强壮，这取决于我们对“视图”的设定，并且如果有人给这具身躯来了一拳，那么身体就会做出相应的反应，是本能还击，还是被打飞，这取决于内功力量。

当身躯被挨了一拳后，这个动作就会在“控制器”也就是“神”中被感受到，“神”会立刻调度身体中的全部“内功力量”，用来防御这次攻击。然后内功，即“气”，对这次攻击进行处理。如果这次攻击太厉害了，内功已经无法防御这次攻击了，那么“内功”就会紊乱，并发送“通知”给“神”，然后“神”感受到了这个信号，产生了“痛觉”的反应。

所以，总而言之，MVC 模型的执行过程是这样的：

1）视图接收到用户动作，然后将其传递给控制器。

2）控制器发送消息给模型，修改模型中的某些数据。

3）模型进行处理后，将完成的结果通知给控制器。

4）控制器接收到通知后，更新视图。

要注意的是，模型在一些简单的应用中并不需要，就好比某些简单的武功并不需要调动诸位自身的“气”一样。此外，视图控制器可以单独存在，也可以包含在其他视图控制器类之中，正如某些武林高手可以分神控制多把神兵一样，诸位了解即可。

或许在诸位看来，在实际的场景中，不过是一个个视图之间在不停地切换，但是实际上是一组组的 MVC 架构之间进行互动，从一个视图控制器到另一个视图控制器，并且视图控制器之间可以共享模型。正如高手过招，凡人看来不过是一招招的比试，然而实际上却是他们自身修为的各各种碰撞，强者为胜。

7.1.1.2 绑定视图和视图控制器

在创建项目的时候，Xcode 已经自动创建好了视图控制器类，并且完成了视图和视图控制器之间的绑定。但是对于新的视图或者新的视图控制器来说，则需要诸位自行完成。本章中的示例以“CrazyBounce”为例。

定位到 Main.storyboard，在其中选中“游戏开始界面”场景，然后选中其视图控制器，或者在大纲视图栏中进行选择。选中后打开界面右方的标识检查器，如图 7-2 所示。

图 7-2 Storyboard 的标识检查器

在这里可以看到 Custom Class 栏，这里面有 Class 属性以及 Module 属性，设置 Class 属性则可以设置与视图直接相关的类，这里给的属性值是 gameBeginViewController，这正是例子中该场景所对应的视图控制器类。从项目导航栏窗口中找到 gameBeginViewController.swift 文件，查看其定义，可以发现这个类继

承自 UIViewController，而 UIViewController 是系统自带的视图控制器，它提供了一系列控制器应有的功能，可以说，iOS 中一切控制类基本都继承自该类，如下所示：

```
import UIKit
class gameBeginViewController: UIViewController, waterViewDelegate{
/*
代码部分
*/
}
```

对于 nib 文件来说，与 stroyboard 文件的绑定大同小异。在 nib 文件的大纲视图栏中找到 File's Owner，其正是与 nib 文件相关联的视图控制器类，选中之后在其标识检查器设置即可。

接下来，诸位将了解到如何将视图控制器类和视图中的控件对应起来，这个过程称为“连接”。

7.1.2　输出口

打开 Main.storyboard 文件，定位到游戏开始界面，然后打开辅助代码编辑器，然后选中其中某个控件（比如图片控件“CrazyBounce 的 LOGO”）。

按住 Control 键，这样便可以从布局栏中拖曳出一根蓝色的线条。将这根线条拖曳到辅助编辑器上，寻找到一个合适的位置，这时辅助编辑器上便会显示一根蓝色的线条，以及一个“Insert…”的提示信息来提示插入的位置，如图 7-3 所示。

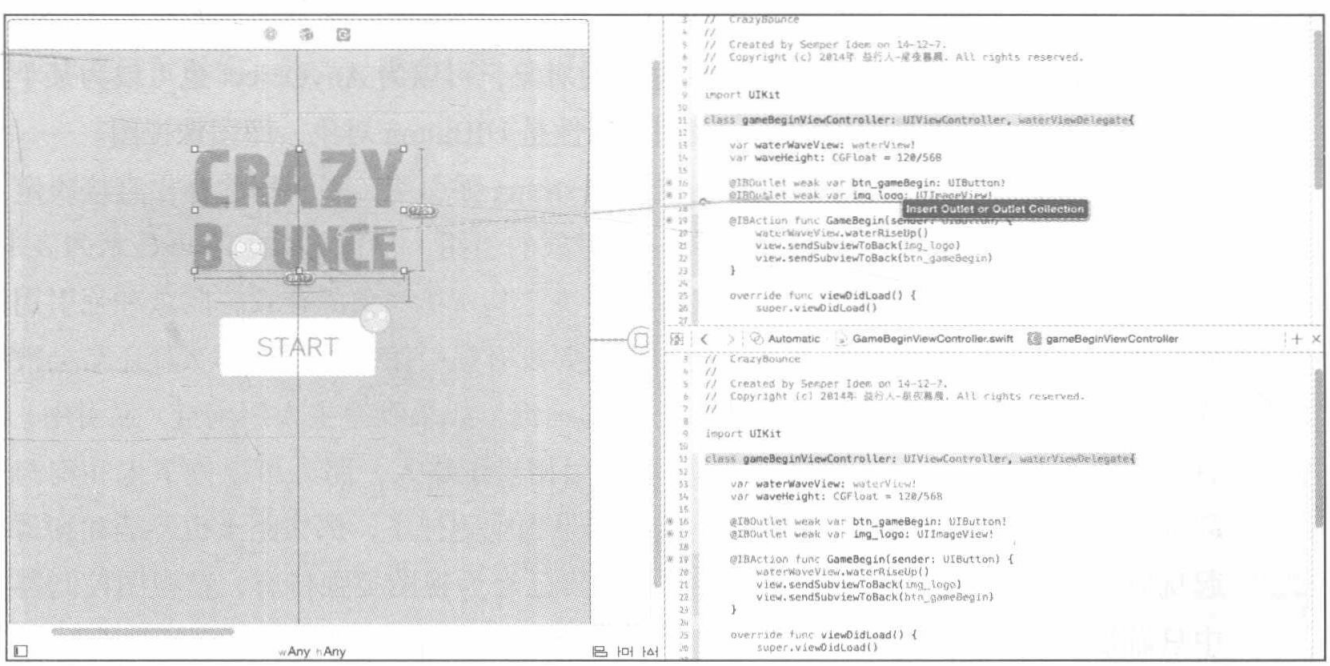


图 7-3　拖放连接条来连接代码和布局

松开鼠标之后，辅助编辑器中便会弹出如图 7-4 所示的输出口选择窗口。说明如下：

图 7-4 输出口选择窗口

- Connection 中拥有三个选项，分别是 Outlet、Action 和 Outlet Collection。这是用来选择该控件所需要绑定的对象类型。Outlet 即为输出口属性，用来绑定视图中的控件和代码中属性，以便诸位能够通过代码来操作视图中的控件；Action 为动作输出口属性，用来将视图中的控件动作和代码中的某段动作绑定，以便诸位能够响应用户与视图的交互；Outlet Collection 则为输出口集合，相当于保存了 Outlet 输出口的数组，适合于同时更新多个控件。这里选择 Outlet。

之后，为这个属性取一个标识名字，这里命名为“imgLogo”。

虽然说属性名字是任意的，但是根据苹果帮制定的相关规范，类和属性的名称应该按照驼峰式命名法来命名。也就是首字母是小写字母，名称中其他单词的首字母为大写。这里我们采取的是更为传统的“匈牙利命名法”来给输出口命名。

- Type 为响应类型，即表明该属性所要连接的对象，可以为 AnyObject 也可以为某个具体的对象类型。这里是 UIButton，表明该属性是 UIButton 对象，即图像视图。
- Storage 为引用类型，分别是 weak 弱引用和 strong 强引用。强引用的存亡直接决定了其所指对象的存亡，如果不存在指向一个对象的引用，并且此对象不再显示列表中，则此对象会从内存中被释放。弱引用除了不决定其所指对象的存亡，其余和强引用相同，即使一个对象被持有无数个弱引用，只要没有强引用指向它，那么其还是会被清除。换句话说，对象相当于一条活泼好动的狗狗，如果没有主人用狗链（强引用）栓着它，那么它就会跑掉（释放掉）。而弱引用相当于路人，路人可以停下来和狗狗一起玩耍，但是路人并不能拥有这条狗狗。如果狗狗跑掉了，那么路人也不能和狗狗一起玩耍了。这里选择 weak 的原因是控件本身已经为视图类强持有，在视图控制器类中只需要维持弱引用即可。

设置完毕后，点击 Connect，这样辅助编辑器中便出现了与控件绑定好了的类属性。如图 7-5 所示。可以看到，语句左边的小圆点是实心的，表明其已成功地与控件绑定，如果没有绑定成功的话，这个小圆点则显示为空心的。这个小圆点是 Xcode 自动识别到输出口而标

记的记号，也就是我们之前所说的神秘的“阵眼”所在。

图 7-5　IBOutlet 绑定成功

当然，还可以先添加相应的属性，再将其与控件进行连接。我们在之前创建的属性下方再添加以下代码，如图 7-6 所示：

```
@IBOutlet weak var btnGameBegin: UIButton!
```

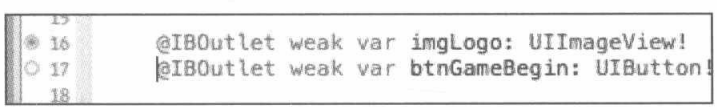


图 7-6　空白的 IBOutlet 连接口

可以看到，没有与控件连接的属性左边出现的是空心小圆点。将鼠标移到空心小圆点上面，会出现一个白色加号，按住鼠标左键便可以拖曳出一条蓝色的线，将其拖曳到控件上也可以完成“连接”操作。


提示　已经完成某一类型绑定的控件无法再次被连接绑定。

反过来也是可行的，选中 Button 控件，按住 Control 键将蓝色的线拖曳到对应的属性名上面，这时 Xcode 便会显示一个方框将该属性选中，并提示“ Connect Outlet”。松开鼠标即完成“连接”操作，Xcode 还会闪烁几下以提示属性已连接，如图 7-7 所示。

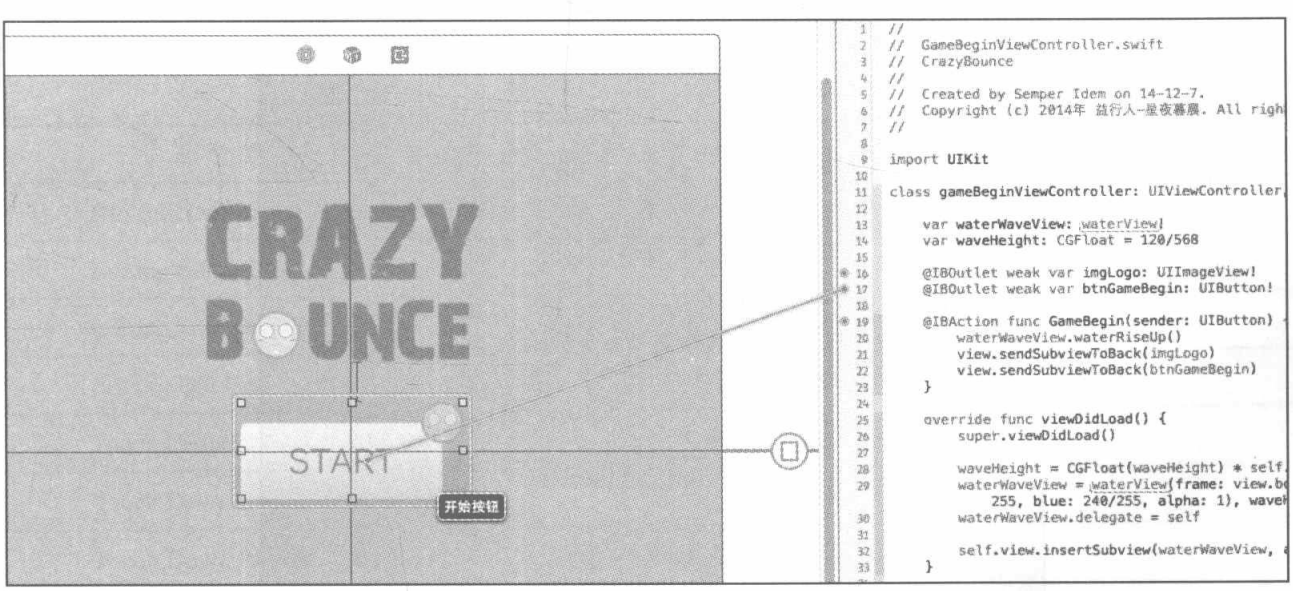


图 7-7　从代码区域连接布局区域

还可以打开“ View Controller”所对应的连接检查器，如图 7-8 所示。这里列出了所有的

属性变量到控件之间的连接关系。通过从某个连接的空心圆也可以拖曳出一根蓝色的线条到任意控件处，完成“连接”操作。

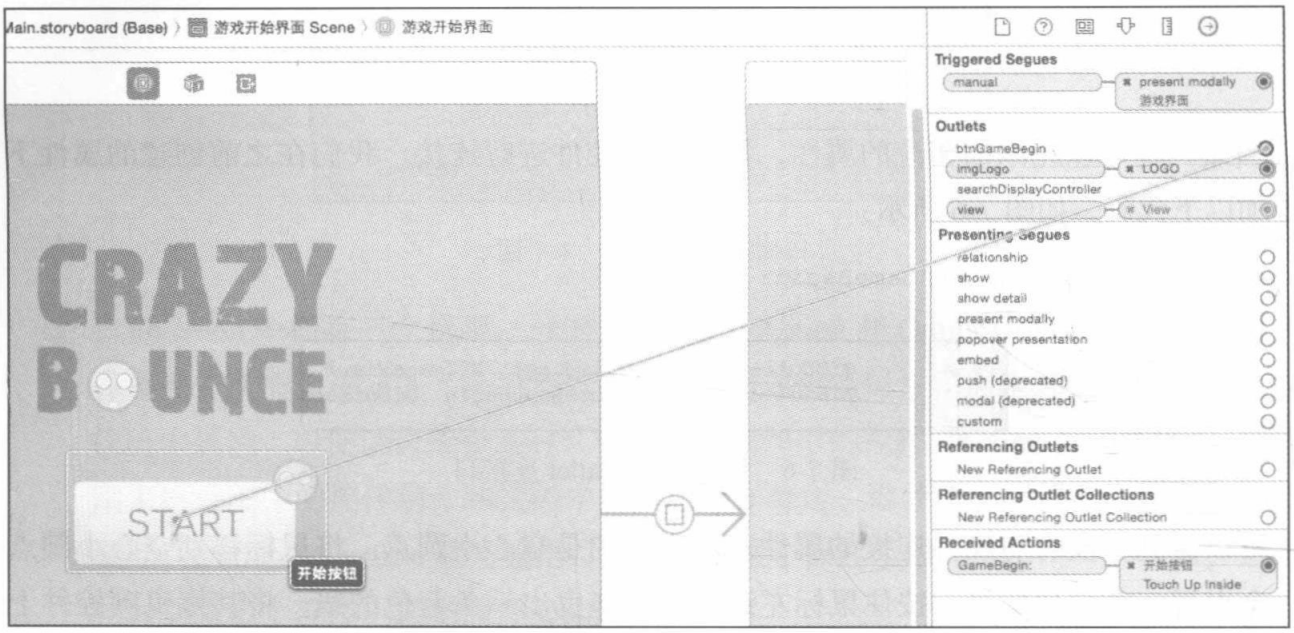


图 7-8　从连接检查器连接布局区域

还可以选中视图页面上的 View Controller，按住 Control 键拖曳出一条蓝色的线条的控件处，如图 7-9 所示。在弹出的如图 7-10 所示选择窗口中可以选择输出口。

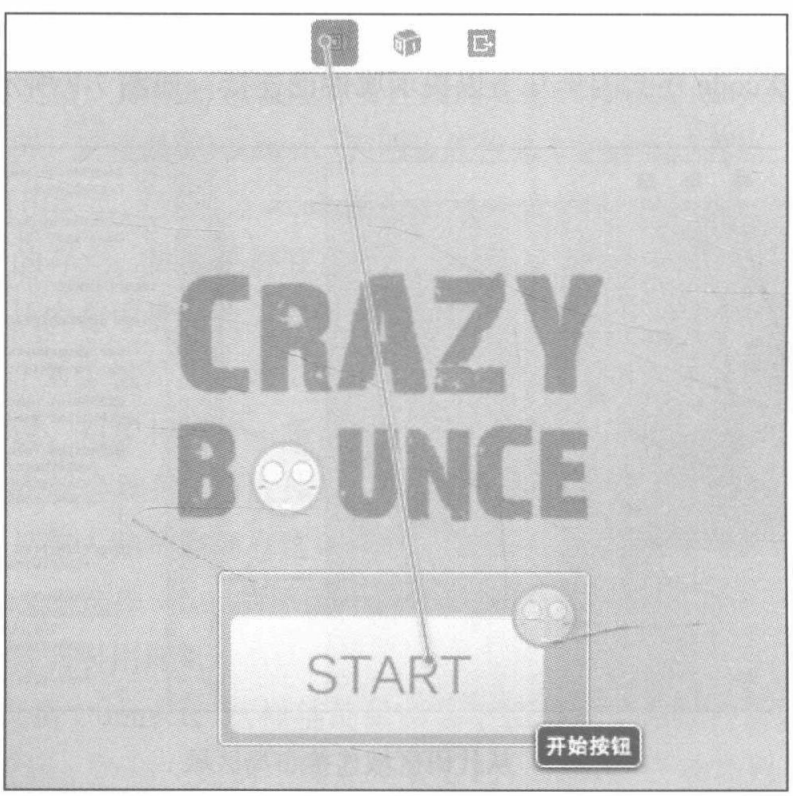


图 7-9　从布局区域连接 View Controller

注意　在删除某个属性后必须要到连接检查器中删除对应的连接，否则编译会报错。

在 Xcode 当中，创建输出口的方法还有很多种，并不仅仅本章所介绍的这几种，诸位可以自行尝试，然后选择一个自己最为习惯的方式来创建输出口。

图 7-10　输出口选择窗口

7.1.3　动作

创建动作（Action）输出口和创建 Outlet 输出口的方式大同小异，唯一不同的是在创建的过程中，要在输出口选择窗口中的 Connection 项指定为 Action 连接方式，如图 7-11 所示。

图 7-11　IBAction 设置界面

动作拥有两种响应方式，这两种响应方式并不会显式显示出来。分别是：即时响应和延时响应。即时响应将在用户产生动作时立即将动作返回，而延时响应在用户动作结束后将动作返回。虽然即时响应的返回速度非常快，但是苹果帮仍然建议尽可能地使用延时响应，以便给用户反应的时间以及反悔的机会。

与 Outlet 输出口不同的是，动作输出口并没有引用类型，取而代之的是事件（Event）和参数（Arguments）。

Event 当中包含了一系列响应事件，用来标记该动作应该响应控件的哪些事件。表 7-1 列出了常见的几种响应事件。

表 7-1　动作响应事件

事件名称	触发条件
Did End On Exit	用户点击 return 或者 done 按钮
Editing Changed	字符增减，焦点改变位置等
Editing Did Begin	文本域得到焦点
Editing Did End	文本域失去焦点
Touch Cancel	全局事件，取消当前区域的单击操作
Touch Down	某个区域中手指按下
Touch Down Repeat	区域内重复的手指按下
Touch Drag Enter	手指拖进入区域内
Touch Drag Exit	从区域内将手指拖出边界

（续）

事件名称	触发条件
Touch Drag Inside	手指在区域内拖动
Touch Drag Outside	手指在区域外拖动
Touch Up Inside	在某个区域中单击
Touch Up Outside	在某个区域中按下，然后在区域外放开
Value Changed	某个控件的值发生变化

Arguments 则意味着传进动作输出口中的参数类型，选项 Sender 一般为引发动作的控件类型，Event 一般为引发动作的事件类型。如果打算读取引发动作控件的相关属性，那么请将参数设置为 Sender，并且将其值设置为该控件类型，否则设置为 Anyobject 就可以满足要求。如果打算读取引发动作的事件相关属性，那么将参数设置为 Event。如果两者都要读取，那么设置为 Sender and Event。

创建完毕之后，Xcode 将会自动插入动作输出方法，代码示例如下所示：

```
@IBAction func GameBegin(sender: UIButton){ ... }
```

即时响应通常所创建的动作输出方法往往是公有的，也就意味着其他类可以在需要的时候调用它。而延时相应所创建的动作输出方法往往是私有的，意味着这个动作只能够在界面构造器以及视图控制器之间进行交互。这些适应范围并不会显式显示出来。

借助输出口打通了阵眼之后，诸位便可以轻松地用内力在阵法中施展各自的招式了。一定要记住，要想从心所欲地施展招式，内功心法一定要练好。否则，根基不牢，则之后难以精进。高手们精彩绝伦的招式无一不是数年的内功修炼才能得以炼成，诸位任重而道远！

7.2 语法感知

Xcode 拥有绝大多数 IDE 都有的语法感知功能，也就是可以自行识别出各种关键词，并藉此实现各种功能。这些功能都能够帮助我们更高效地管理代码，并且节省时间。

7.2.1 语法高亮

语法感知所提供的最明显的功能就是语法高亮了。这个功能通过对编程语言的语法进行分析，对源代码的注释、关键字以及属性等元素分配一个句法标签，每个句法标签都会标注为一种颜色和字体。语法高亮旨在提升源代码的可读性。

你可以通过选择 Xcode → Preferences 然后选择 Fonts & Colors 项来选择多种字体和颜色。关于 Fonts & Colors 的相关内容，请参阅本书附录 A.3.5 节“字体、颜色”。

7.2.2　聚焦和折叠代码

折叠代码是一个简单的功能，它可以让我们在不需要编辑某个代码模块的时候将其隐藏起来，这样就可以专注于其他代码模块的编辑。

选择菜单栏 Editor → Code Folding → Fold Methods & Functions，即可将当前文件中全部的方法和函数进行折叠。定位到你想要展开的方法上，然后双击它的省略按钮来展开该方法。

Fold Methods & Functions 是收起每个方法或者函数最外层的代码模块，Fold 则是收起包含当前所选文本的代码模块，Fold Comment Blocks 则是收起所有的多行注释。

或者将鼠标指针移动到编辑器左边缘上，它将会在编辑区域中显示一个焦点范围框，如图 7-12 所示。

```
//
//  GameBeginViewController.swift
//  CrazyBounce
//
//  Created by Semper Idem on 14-12-7.
//  Copyright (c) 2014年 益行人-星夜慕晨. All rights reserved.
//

import UIKit

class gameBeginViewController: UIViewController, waterViewDelegate{

    var waterWaveView: waterView!
    var waveHeight: CGFloat = 120/568

    @IBOutlet weak var imgLogo: UIImageView!
    @IBOutlet weak var btnGameBegin: UIButton!

    @IBAction func GameBegin(sender: UIButton) {
        waterWaveView.waterRiseUp()
        view.sendSubviewToBack(imgLogo)
        view.sendSubviewToBack(btnGameBegin)
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        waveHeight = CGFloat(waveHeight) * self.view.frame.height
        waterWaveView = waterView(frame: view.bounds, color: UIColor(red: 16/255, green: 169/255, blue: 240/255,
            alpha: 1), waveHeight: Float(waveHeight))
        waterWaveView.delegate = self

        self.view.insertSubview(waterWaveView, atIndex: 0)
    }

    func checkWaterRiseOver(riseOver: Bool) {
        if riseOver {
            self.dismissViewControllerAnimated(true, completion: nil)
            performSegueWithIdentifier("gameStart", sender: self)
        }
    }

    override func prefersStatusBarHidden() -> Bool {
        return true
    }
}
```

图 7-12　焦点范围框

代码模块的深度是通过焦点范围框的灰度来显示的，焦点范围框的颜色越深，则表示该代码模块的嵌套层次也就越深。

此外，还可以选择 Editor → Code Folding → Focus Follows Selection，这样直接单击代码

区域，就能显示该代码片段所在的模块范围。

通过单击编辑器左边缘的收起按钮，就可以收起聚焦的代码模块。折叠的代码会用一个省略号占位符代替掉，如图 7-13 所示。

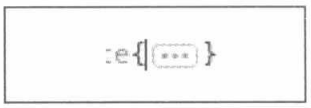


图 7-13 折叠后的代码块

通过使用相反的“Unfold”操作便可以展开所收起的代码块。

7.2.3 自动填充

语法感知中最复杂、也是最有用的功能就是代码自动填充（Code Completion），这个功能会分析所输入代码的上下文并提供一些建议。通过自动填充功能，可以极大地加快代码的输入速度，也减少了开发者的记忆量。

当你开始键入代码时，Xcode 会提供相关建议以帮助你补全代码。单击建议栏中的某项来选中它，或者使用向上或向下键来更改选中的建议。按下 Return 键接受该建议。一般情况下，Xcode 提供的默认建议选择会停留在你上次所接受的建议上。

提示 自动填充实际上是区分大小写的，它会自动纠正所输入代码的大小写。如果忽略了自动填充的建议，那么文本便会恢复你输入的样式。

当一个方法或者函数含有参数或者引用时，代码完成功能将用占位符来替代每个参数。要在这些占位符中跳转，请选择 Navigate → Jump to Next Placeholder（或者 Navigate → Jump to Previous Placeholder）。另外，使用 Tab 键可以跳转到下一个占位符，用 Shift + Tab 键跳转到上一个占位符。

注意 如果 Xcode 没有完成对项目的语法感知分析，那么自动填充功能将无法正常使用。

默认情况下，Xcode 是自动启用自动填充功能的。如果关闭了自动填充功能，那么你仍然可以使用“Control + Space”请求建议。

7.3 查看数据定义

将鼠标指针放到一个符号上，然后按住 Command 键并单击来显示这个符号的具体定义。源代码编辑器将导航到该符号的定义中，并将其高亮显示。如果该定义位于单独的文件中，那么代码编辑器将会显示这个文件（亦或者将鼠标指针放到符号上，然后选择 Navigate → Jump to Definition）。

将鼠标指针放到一个符号上，然后按住 Command + Option 键并单击，就可以在辅助编辑器窗格中显示其定义，如图 7-14 所示。这个方法可以让你在查看符号定义时，保持该符号一直在视野中。

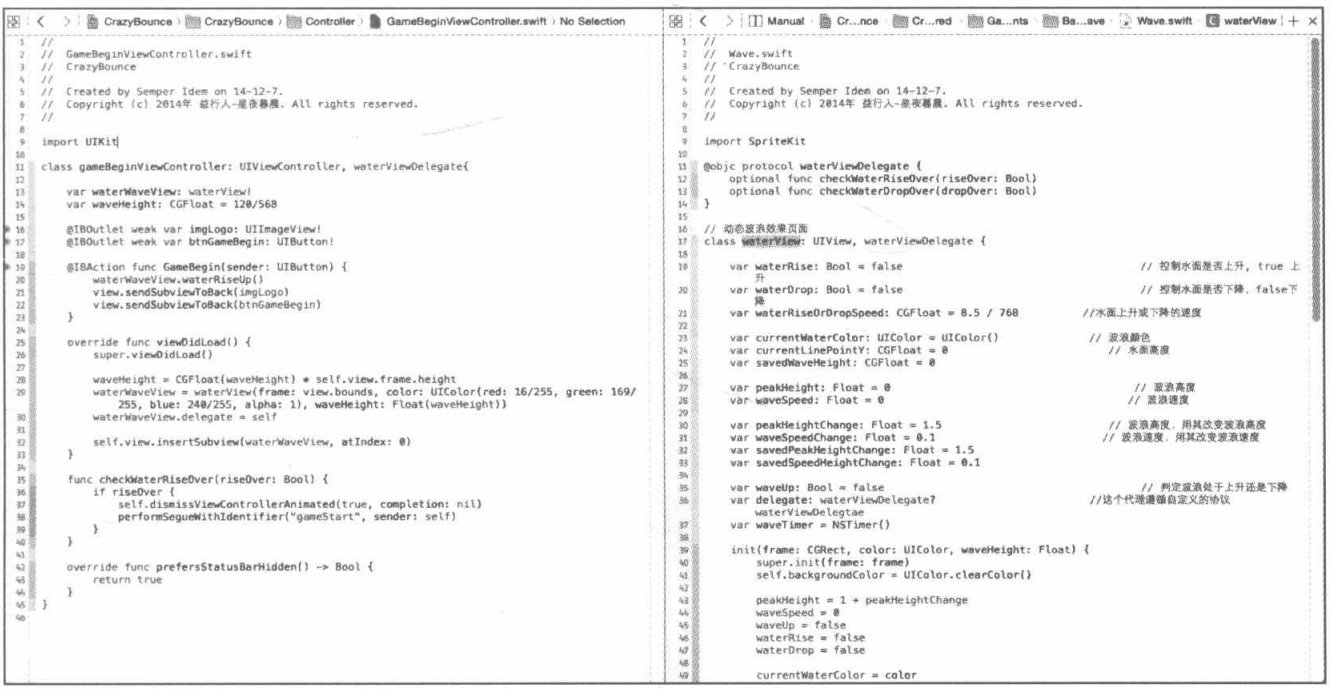


图 7-14　查看符号定义

7.4　全局修改数据

Xcode 提供了一个便捷方式来帮助修改符号名称。这个修改是全局发生的，也就是说，面对成千上百个相同的符号名，只需要修改一次就可以达成修改名称的目的。这个功能叫做“Edit All in Scope”，全局修改数据。

将鼠标悬浮在选中的符号名上，其右侧会出现一个按钮，如图 7-15 所示。单击该按钮将会弹出一个菜单，然后选中“ Edit All in Scope”，如图 7-16 所示。或者直接使用快捷键“ Control + Command + E”。Xcode 将高亮显示当前作用域内的所有该符号，接下来就可以修改符号名以完成全局修改的功能，如图 7-17 所示。

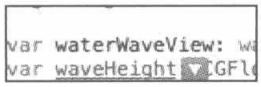


图 7-15　鼠标悬浮在符号名上出现的按钮

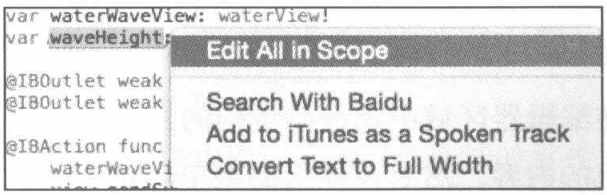


图 7-16　选中“Edit All in Scope”

注意　全局修改数据的操作只在该符号所在的作用域内有效，并且最好不要用全局修改数据功能修改输出口属性、容易混淆的变量等等内容，因为全局修改数据是基于文本分析的，很有可能修改到一些不应该修改的内容。一般情况下，修改这些东西请采用重命名重构（参见下一节内容）。

```
var waterWaveView: waterView!
var waveHei: CGFloat = 120/568

@IBOutlet weak var imgLogo: UIImageView!
@IBOutlet weak var btnGameBegin: UIButton!

@IBAction func GameBegin(sender: UIButton) {
    waterWaveView.waterRiseUp()
    view.sendSubviewToBack(imgLogo)
    view.sendSubviewToBack(btnGameBegin)
}

override func viewDidLoad() {
    super.viewDidLoad()

    waveHei = CGFloat(waveHei) * self.view.frame.height
```

图 7-17　全局修改数据

7.5　重构和迁移

重构（Refactoring）是指在不改变外部功能的情况下，对代码进行重新组织的一种方式。一般而言，重构代码是为了提升某种非功能性的属性，比如减少代码冗余度、提升代码可读性等等。重构并不是对代码的随意重写，相反，重构需要遵循严格的规则，并且其对代码的修改并不会非常大。

迁移（Converting）是 Xcode 提供的一套版本迁移工具，用来将旧版本的项目迁移为新版本的项目。一般而言，迁移是响应苹果的号召，摈弃不再被支持的旧功能而转为实现新功能。迁移工具出现的原因是因为有些时候新版本的内核、语言版本变换得特别大，手动更改几乎是不切实际的做法。因此，苹果特地推出了这个工具，来帮助大家来迁移项目。一般情况下，项目越大，迁移的出错几率越大，因此需要慎用这个功能。

重构和转换操作均是不可逆的操作，并且往往对代码的改动范围特别大。因此，在进行重构或者转换操作前，请做好备份。

7.5.1　重构操作

在编辑器区域中选在待重构的标识符、代码片段、文件等。右键单击选定的内容，然后在弹出的菜单中选择 Refactor 菜单，就可以使用重构的一些特性了。或者通过 Edit → Refactor 来访问，如图 7-18 所示。

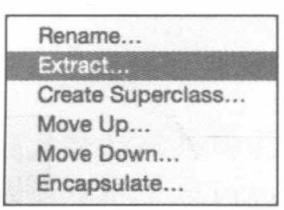


图 7-18　重构菜单

在语法感知操作未完成前，是不能够进行重构操作的。同样，如果你关闭了语法感知功能，那么重构操作也是无法进行的。

在重构菜单中选择你想要进行的重构操作，一旦选择了某种重构操作，Xcode 就会弹出一个对话框，让你提供执行该重构操作所需的必要信息。

设置完重构信息后，单击“ Preview”按钮开始处理过程，这个过程会扫描整个项目并给出其建议的变更列表，此时还不会进行实际的更改。Xcode 将会弹出一个类似于版本编辑器的窗口来让你预览更改，可以在其中取消一些重构操作。

确定之后，单击 Save 按钮来应用重构操作。完成后，重构工具就会关闭并返回到修改后的项目中。

注意　重构操作可能会耗费大量的时间，这取决于你所需重构的代码数。此外，重构目前只支持 C++ 和 Objective-C。

如果你使用 CrazyBounce-Swift 工程的话，那么请换用示例代码中的 CrazyBounce-OC 工程。下面，我们使用这个工程来进行重构的体验。

7.5.1.1　重命名

Rename（重命名）标识符，一般这种操作是为了让代码更加易读。标识符包括有类、方法、函数、变量的名称。不过要注意的是，在协议中声明的方法是不能被重命名的。

用鼠标选中要替换的某个类、方法或者变量，确保跳转栏上的最右边显示的是我们期望进行重命名的符号。然后进入到重构菜单中，选择“ Rename”选项。这里，我们选择了“Ball.m”中的“shockTime”属性进行重命名操作。

选择重命名操作后，Xcode 会弹出一个对话框，如图 7-19 所示。

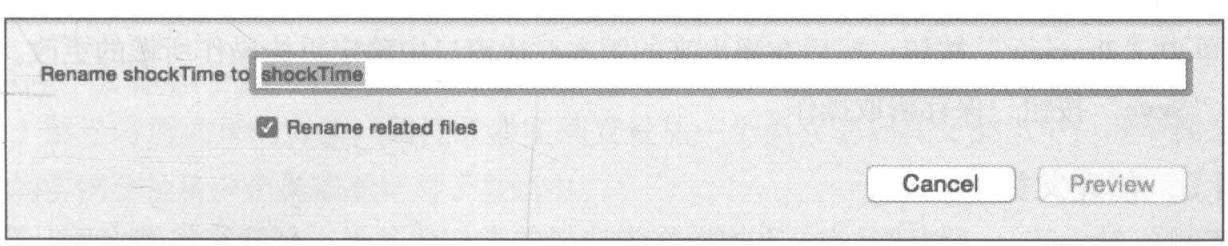


图 7-19　重命名操作

使用重命名重构时，需要提供一个替换的名字。除此之外，还有一个选项，可以重命名其所在的文件，让其名字与重命名的名字保持一致。一般情况下，这个选项只在类重命名的时候有效。

单击“ Preview”按钮，然后在弹出来的版本对比窗口中确定重构操作所做的更改。然后单击“Save”按钮，保存重命名操作。

重命名操作比上一节所介绍的全局修改数据要更为精确和方便。

7.5.1.2　析取

Extract（析取）的意思是将选择的代码片段抽取到一个新的方法或者函数当中。也就是说，析取操作将会创建一个新的方法或者函数，并且将所析取的代码用这个新的方法或者函

数来替代。

选中要执行析取操作的代码段，然后进入到重构菜单中，选择“ Extract”选项。这里，我们选择了“ Ball.m”中 "initWithCenter(_:withSize:AndSpeed) 方法中，对三个 image 属性进行初始化的代码段，即：

```
self.ballDownImage = [SKTexture textureWithImageNamed:@"BallShockedImage"];
   self.ballUpImage = [SKTexture textureWithImageNamed:@"BallRelievedImage"];
   self.ballBounceImage = [SKTexture textureWithImageNamed:@"BallFrustratedImage"];
```

和重命名操作一样，使用析取重构时，弹出一个类似的对话框，如图 7-20 所示。

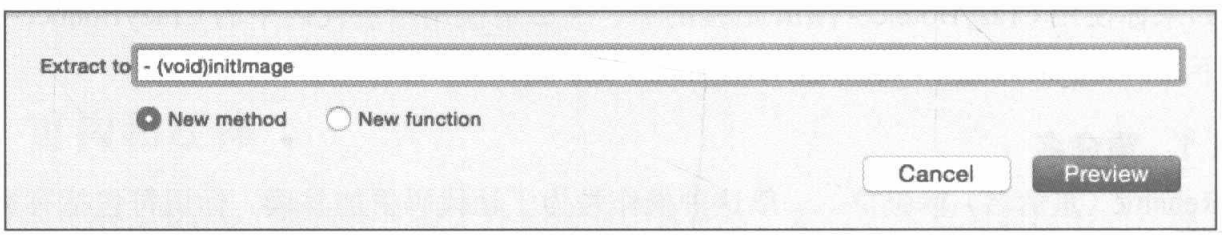


图 7-20 析取操作

我们需要提供一个新的方法或者函数的名字，然后选择要将其转换成方法还是函数。析取完成之后，原来的代码将被这个新创建的方法消息或函数调用取代。

在析取代码时，Xcode 会分析所选代码片段中所有变量引用的范围，任何不再该范围内的变量都会被转换为参数。

单击“ Preview”按钮，然后在弹出来的版本对比窗口中确定重构操作所做的更改。然后单击“Save”按钮，保存析取操作。

7.5.1.3 创建父类

Create Superclass（创建父类）重构操作为选择的类创建父类，创建完父类之后，选择的这个类会完成自动继承。

选中要执行创建父类操作的类，然后进入到重构菜单中，选择“Create Superclass”选项。在这里，我们选择“ Ball”类，然后创建一个全新的类“ BaseBall”，以作为全部小球的基类。弹出的对话框如图 7-21 所示。

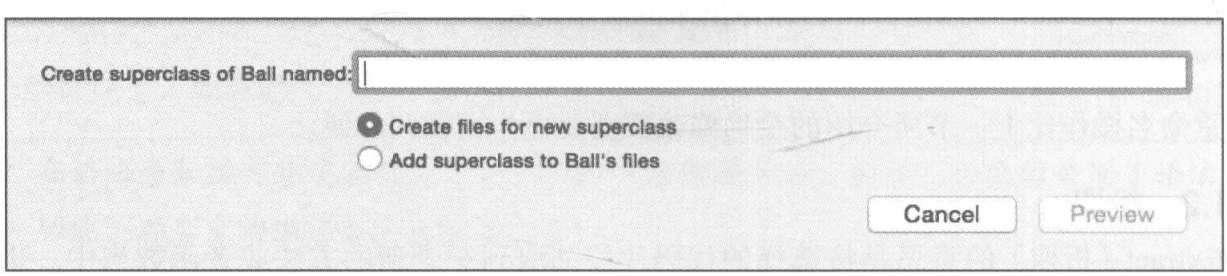


图 7-21 创建父类操作

使用创建父类重构时，需要提供一个新的父类名称，然后选择是创建一个新的文件来保存这个父类，还是将这个父类直接置于选择的类所在的文件当中。如果选择的这个类已经拥有了一个父类，那么新创建的类则会继承自原先的父类，然后这个类再继承自新创建的类。

单击 Preview 按钮，然后在弹出来的版本对比窗口中确定重构操作所做的更改。然后单击 Save 按钮，保存创建父类操作。

创建完父类之后，我们可以查看 BaseBall.h 和 Ball.h 的定义：

```
// BaseBall.h
...
@interface BaseBall : SKSpriteNode

@end
// Ball.h
...
@interface Ball : BaseBall
...
@end
```

我们可以看到，创建父类的重构操作成功创建了一个 Ball 的父类 BaseBall，并自动继承了 Ball 原先所继承的 SKSpriteNode 类，然后 Ball 就继承了 BaseBall。

注意　重构操作有时并不能够非常正确的分析头文件，因此，执行这个重构操作后，可能还需要对头文件进行一点小修改。

7.5.1.4　上移和下移

上移是将所选择的方法、属性或者实例变量从一个类移动到其父类当中，而下移是将所选择的实例变量从一个类移动到其子类当中。

如果选择上移变量的话，那么可能还要选择是否将使用该变量的方法一并上移到父类当中。

如果选择下移的话，那么 Xcode 会列出该类的所有直接子类，选择要移到的子类即可。

我们在“Ball.m”文件中新增加一个变量和一个方法：

```
@interface Ball ()

...
@property int test;
...
@end

@implementation Ball

...

-(void)testMove {
    self.test++;
}

@end
```

选中我们新创建的那个属性，然后进入到重构菜单中，选择“Move Up”选项，弹出的对话框如图 7-22 所示。

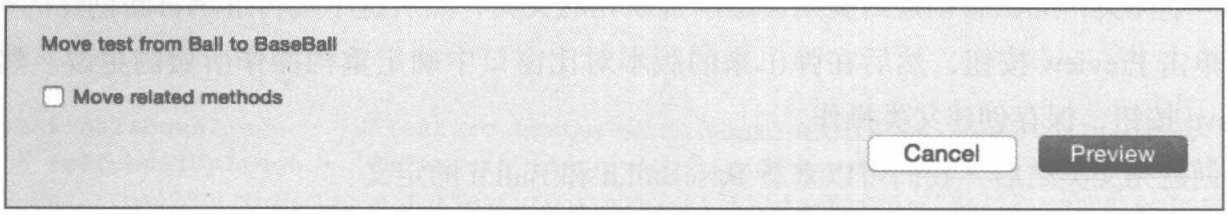


图 7-22 上移操作

Xcode 会提示这个符号的移动路径，在这里我们是将 test 属性从 Ball 类中上移到其父类 BaseBall 当中。这里我们勾选上“Move related methods”选项，这样会将 -(void)testMove 方法一并上移到 BaseBall 类当中。

单击“Preview”按钮，然后在弹出来的版本对比窗口中确定重构操作所做的更改。然后单击“Save”按钮，保存上移操作。

这个时候，我们就可以在 BaseBall 中看到 test 属性和 testMove 方法了。

然后我们给 BestBall 创建一个实例变量：

```
@interface BestBall () {
    int testDown;
}
```

选中我们新创建的这个实例变量，然后进入到重构菜单中，选择“Move Down”选项，弹出的对话框如图 7-23 所示。

图 7-23 下移操作

和上移操作类似，Xcode 会提示这个实例变量的移动路径。然后单击 Preview 按钮，然后在弹出来的版本对比窗口中，勾选该实例变量要移动进去的子类文件，确认所做更改后，单击“Save”按钮，保存下移操作。

这个时候，我们就可以在 Ball 中看到我们下移的实例变量 testDown 了。

7.5.1.5 封装

Encapsulate（封装）会为类的实例变量创建 getter 和 setter 方法，然后使用消息语法对该变量的所有引用进行替换。

我们在 Ball 类中选择下移操作中下移的实例变量 testDown，然后进入到重构菜单中，选择 Encapsulate 选项，弹出的对话框如图 7-24 所示。

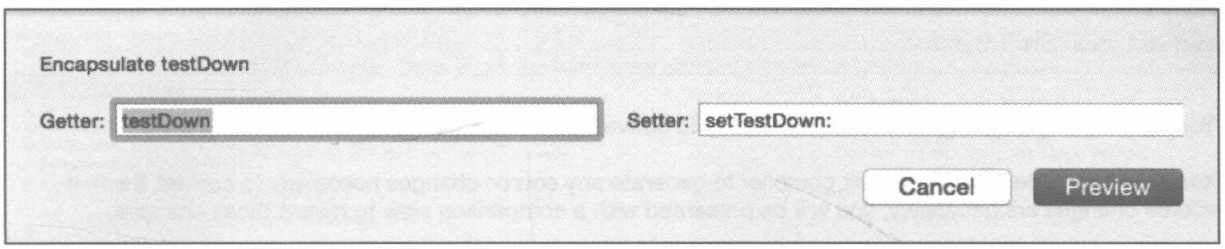


图 7-24　封装操作

在这个重构对话框中，我们可以重新设置该实例变量的 getter 和 setter 方法的名称，默认情况下，getter 方法名称和实例变量相同，setter 方法名称为“set 实例变量名 :”。

单击“ Preview”按钮，然后在弹出来的版本对比窗口中确定重构操作所做的更改。然后单击 Save 按钮，保存封装操作。

这样，我们就可以在 Ball 类当中看到新创建的实例方法的 getter 和 setter 方法了。

7.5.2　迁移操作

迁移操作是全项目范围之内的操作，它会将整个项目改得面目全非——有可能是好的方向，也有可能是坏的方向。

通过菜单栏的 Edit → Convert 可以访问迁移菜单，如图 7-25 所示。

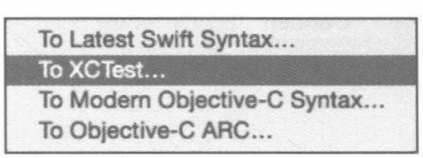


图 7-25　迁移菜单

注意　在语法感知操作未完成前，是不能够进行迁移操作的。同样，如果你关闭了语法感知功能，那么迁移操作也是无法进行的。

在迁移菜单中选择你想要进行的迁移操作，一旦选择了某种迁移操作，Xcode 就会弹出一个对话框，让你确认该项迁移操作所要进行的信息，如图 7-26 所示。

确认完毕之后，就可以开始执行一系列的迁移设置了，在这些设置中，基本上都是要对执行迁移操作的对象进行选择，如图 7-27 所示。

设置完毕后，Xcode 会自动分析整个项目，判定需要更改的内容，然后弹出一个类似于版本编辑器的窗口来让你预览更改，如图 7-28 所示。

这个窗口由两个窗格组成，一个将要被修改的文件列表和一个比较视图。在此中，可以取消要被修改的文件，也可以抛弃某个修改操作。此外，还可以在左侧的比较视图中自定义更改。

确认更改后，单击 Save 按钮来应用迁移操作。

注意　迁移操作会耗费大量的时间，并且不要寄希望于迁移工具的准确性。一般说来，项目越大，迁移操作越费时，出错概率也越大。此外，对于使用了最新功能的项目来说，没有必要使用迁移功能。

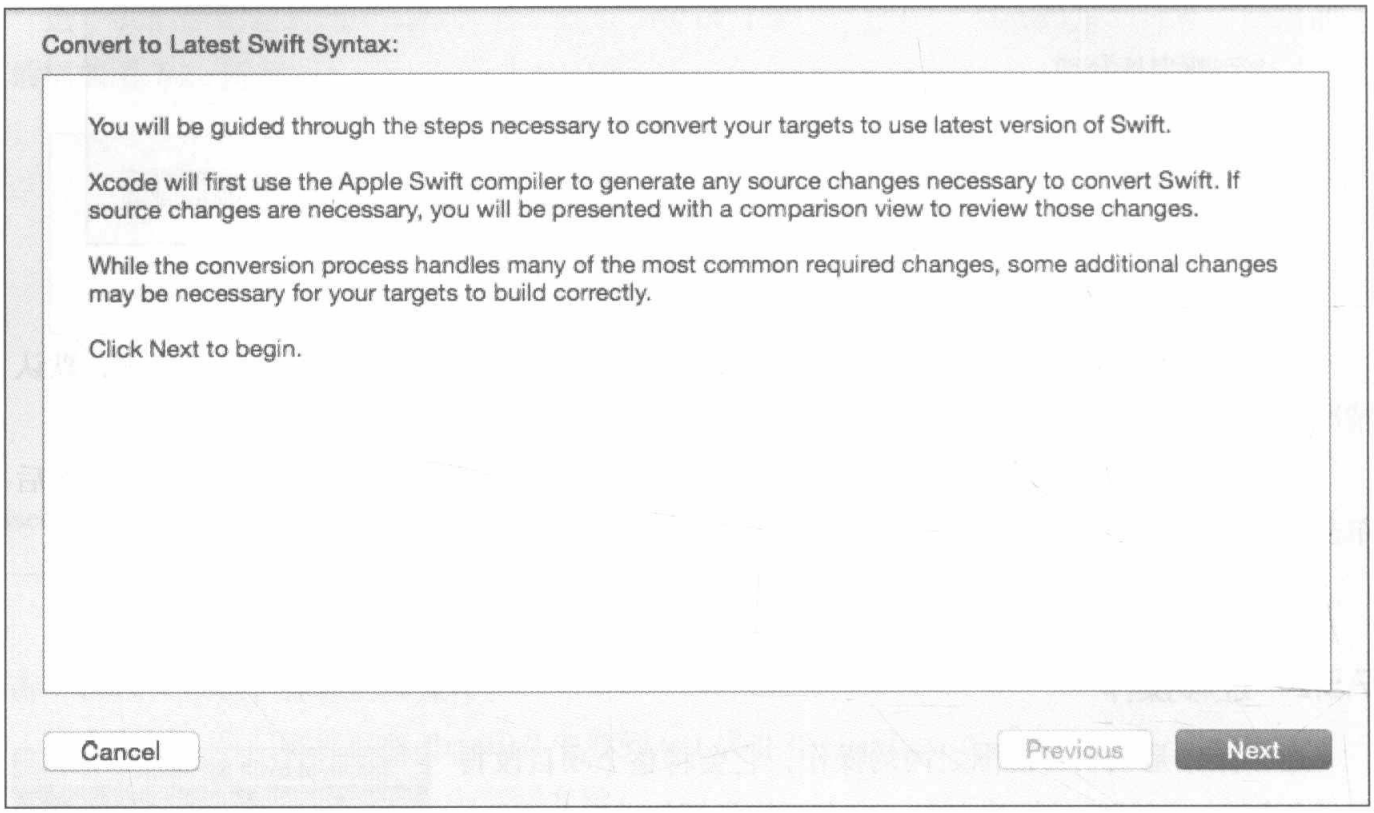


图 7-26 迁移操作确认信息

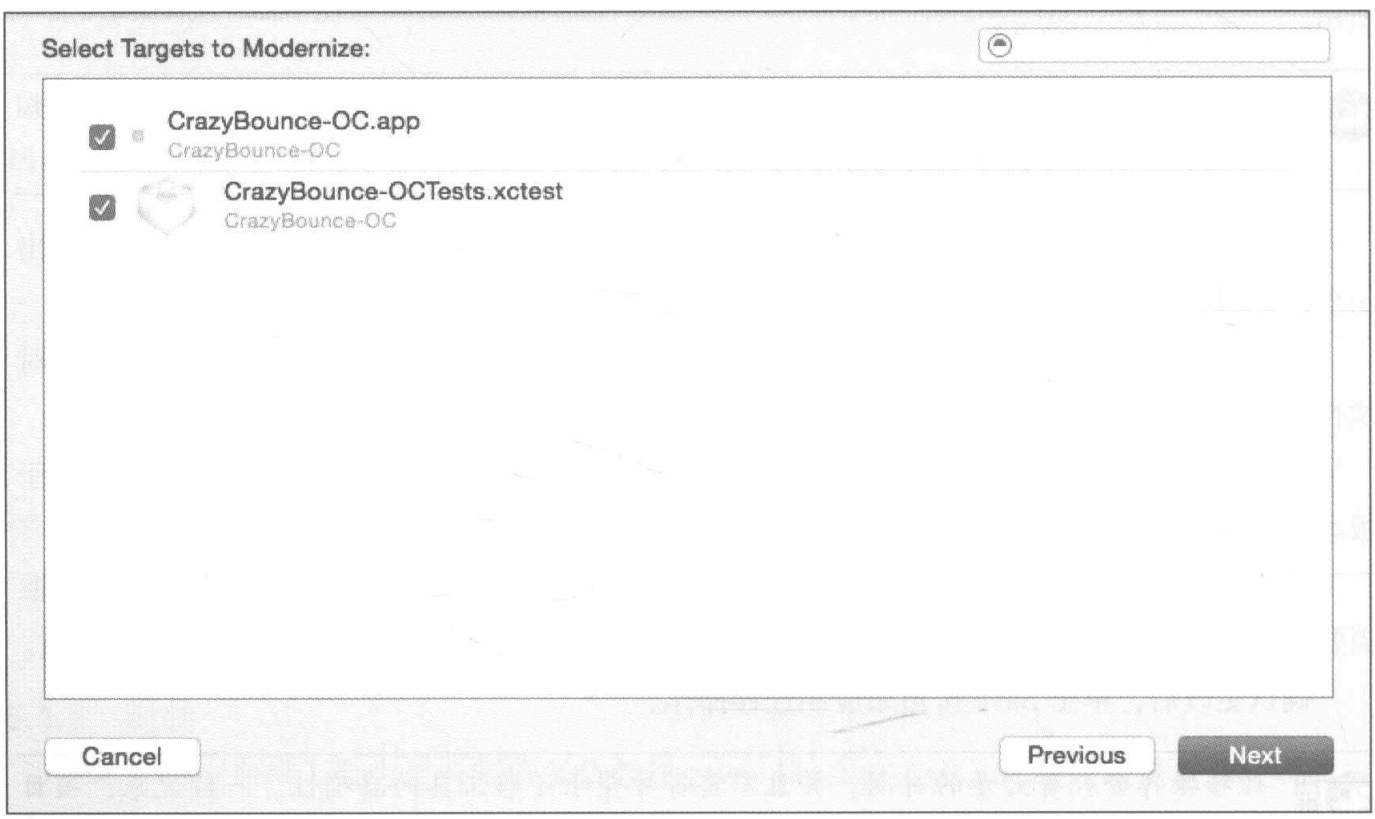


图 7-27 选择迁移对象

图 7-28　预览更改

7.5.2.1　迁移到最新的 Swift 语法

Swift 语法自它出现以来，在很短的时间内经历了许多极大的版本更迭。到 Xcode 6.4 为止，Swift 已经到了 1.2 的版本，它提供了许许多多的新特性，同时也导致旧版本的 Swift 项目出错频繁。出于对开发者的关爱，苹果提供了该迁移工具，帮助使用老版本 Swift 语言的开发者迁移到最新的 Swift 语法特性来。

点击 To Latest Swift Syntax…选项，Xcode 首先会弹出一个对话框，告诉我们：

通过以下必要的步骤，该向导将会帮助你将项目迁移到最新版本的 Swift。

Xcode 会首先使用 Apple Swift 编译器来生成迁移所需要的所有代码变化项。如果该变化项是必需的，那么将会向你展示一个比较页面，来查看这些变化项。

虽然迁移操作可以处理绝大多数常见且必需的变化项，但是为了让你的项目能够成功运行，你可能还需要手动进行额外的更改。

单击 Next，就可以开始迁移操作了。这时候会提示我们选择要执行迁移操作的对象列表，所有的迁移操作都必须要进行这样的选择。

随后，Xcode 会检视所有的源代码，然后显示预览界面。所有发生的更改都会在这个预览界面中显示。如果没有找到变化项，那么就会提示“没有必需的源代码变化项”。确认完修改项之后，单击 Next 按钮，经过一段时间的等待后，迁移操作就完成了。

7.5.2.2　迁移至 XCTest

将原先使用 OCUnit 测试框架的项目迁移到支持 XCTest 测试框架的项目。XCTest 是

Xcode 5 中新引入的一个测试框架，是上一代测试框架 OCUnit 的更现代化实现。XCTest 提供了与 Xcode 更好的集成并且奠定了未来改进 Xcode 测试能力的基础。XCTest 的许多的功能都类似于之前的 OCUnit。

迁移功能会检查你选择的对象，并且在执行自动转换后让你知道它们是否可以用 XCTest 运行。

点击 To XCTest…选项，Xcode 首先会弹出一个对话框，告诉我们：

通过以下必要的步骤，该向导将会帮助你将项目迁移到使用 XCTest 框架的版本。当选择完迁移的对象之后，你将会在迁移操作之前，得到一个比较界面，显示项目必要的更改。

7.5.2.3 迁移至新的 Objective-C 语法

将 Objective-C 代码转换成新的 Objective-C 版本。自 Xcode 3 以来，Objective-C 进化到了 2.0 版本，它提供了许许多多新的特性。如果你仍然还在使用很古老的 Objective-C 版本的话，那么使用这个功能可以让项目迅速匹配新特性。

点击 To Modern Objective-C Syntax…选项，Xcode 首先会弹出一个对话框，告诉我们：

通过以下必要的步骤，该向导将会帮助你将项目迁移到使用现代化 Objective-C 语法的版本。当选择完迁移的对象之后，你将会在迁移操作之前，得到一个比较界面，显示项目必要的更改。对象的现代化操作会使用 Apple LLVM 编译器以及最新版本 SDK。OS X 对象会被设置为只能使用 64 位编译，并且最低开发版本也会被设置为 10.6。

和前面的迁移操作不同的是，迁移至新的 Objective-C 语法操作还多了一个设置“现代化”选项的步骤。

这些选项分别是：

- Add attribute annotations（添加特性注释）：Xcode 会自动推断方法和属性，判断它们是否需要“特性注释”。所谓的“特性注释”，就是原先开发者在 GCC 中使用的“特性语法（Attribute Syntax）”。例如我们使用了 __attribute__((objc_returns_inner_pointer)) 特性，来告诉 GCC 编译器该函数将会返回一个“内部指针 (inner pointer)”，使用现代化操作后，Xcode 会给这个方法后添加 NS_RETURNS_INNER_POINTER 标识。如果不这样的话，那么 LLVM 将会忽略这个特性。
- Atomicity of inferred properties（推断属性的原子性）：默认情况下是“NS_ NONATOMIC_ IOSONLY”。所谓的“原子性”（Atomicity）指的是不可打断的操作，用来防止两个线程互相等待而导致的死锁现象产生。使用 atomic 特性，Objective-C 可以防止这种线程互斥的情况发生，但是会造成一定的资源消耗。这个特性是默认的。也就是说，atomic 选项是启动原子性操作功能，nonatomic 是关闭这项功能。而 NS_NONATOMIC_IOSONLY 则是在 iOS 平台上关闭原子性，而在 OS X 平台上则保持默认开启原子性功能，如果你的项目中同时含有 iOS 和 OS X 两个平台的对象，那么使

用这个选项是极佳的选择，否则还是使用特定的选项比较好。

- Infer designated initializer methods（推断指定初始化方法）：简单来说，指定初始化方法是指能正确构造此类的初始化方法，并且子类必须要调用父类的指定初始化方法。关于何为“指定初始化方法”就属于 Objective-C 的特性了，在此我们就不加以介绍了。Xcode 会自动寻找原先标记过 objc_designated_initializer 特性的初始化方法，然后使用 NS_DESIGNATED_INITIALIZER 宏指令来代替，这让我们声明指定初始化方法更加容易了。
- Infer instancetype for method result type（推断方法返回类型是否为实例类型）：Xcode 将会把 alloc、init 和 new 之类的方法的返回类型 id 替换为 instancetype。这项特性对于原先工厂方法的声明十分有效，因为对于原先的工厂方法来说，并不能返回和其所在类的类类型，而现在使用 instancetype，则可以返回其所在类的类类型了，这样我们就可以确定使用工厂方法创建的对象类型。
- Infer protocol confomance（推断协议一致性）：这是唯一一项默认被关闭的转换选项，这项功能用来添加向类中一些未被声明的协议，以保证“一致性”。比如说，我们在一个控制器中声明了两个表视图数据源方法 -tableView:numberOfRowsInSection: 和 -tableView:cellForRowAtIndexPath:，那么 Xcode 会自动推断出这两个协议方法，然后如果这个控制器中没有继承“UITableViewDataSource”协议的话，就会在这个控制器中添加这个协议。注意，这项功能似乎只能够推断出非 optional 协议方法。
- Infer readonly properties（推断只读属性）：Xcode 将会推断类中属性声明方法，如果该属性声明方法没有相应的 @property 声明，那么 Xcode 就会向类中添加所推断的属性。此项功能中，Xcode 将会推断 getter 方法。此外，这项功能并不会移除我们创建的属性声明方法，但是 @property 会覆盖掉之前创建的 getter 和 setter 方法。
- Infer readwrite properties（推断读写属性）：此项功能和上面类似，将会推断 getter 和 setter 方法。比如说，有这么两个没有声明属性的方法：-(NSString*) name; 和 -(void) setName:(NSString*) newName; 那么 Xcode 就会读取到这两个方法，然后向类接口中添加以下属性：@property (nonatomic, copy) NSString* name;
- ObjC literals（ObjC 字面量）：扩展的字面量语法是现代化 Objective-C 提供的新特性，这项功能可以将原先 NSNumber、NSArray 和 NSDictionary 的初始化声明应用上字面量语法功能。比如说原先创建一个 NSNumber 对象是这样子的：NSArray* array = [NSArray arrayWithObjects:@"1", @"2", @"3", nil]; 经过转换后，就会变成这样：NSArray* array = @[@"1", @"2"];
- ObjC subscripting（ObjC 下标）：和上一项功能类似，扩展的下标语法也是一项新特性。这项功能可以将原先访问 NSArray 和 NSDictionary 的方法应用上下标语法功

能。比如说原先要访问一个 NSArray 对象是这样子的：NSString* number = [array objectAtIndex:0]; 经过转换后，就会变成这样：NSString* number = array[0];

- Only modify public headers（只修改公共头文件）：如果某些对象使用了私有的头文件，那么我们对其进行现代化操作很有可能导致这个对象不可用（原因是我们无法访问并修改这些私有头文件），那么使用这个选项可以避免对私有头文件进行的修改。
- Use NS_ENUM/NS OPTIONS macros（使用 NSENUM/NS _OPTIONS 宏指令）：NS_ENUM 和 NS _OPTIONS 宏指令是创建指定了类型和大小的枚举的最快捷方法。比如说，原先我们定义一个位掩码枚举是这样的：

```
enum {
    BitMaskA = 0,
    BitMaskB = 1 << 0,
    BitMaskC = 1 << 1,
    BitMaskD = 1 << 2
};
typedef NSUInteger BitMask;

经过转换后，就会变成：

typedef NS_OPTIONS(NSUInteger, BitMask) {
    BitMaskA = 0,
    BitMaskB = 1 << 0,
    BitMaskC = 1 << 1,
    BitMaskD = 1 << 2
};
```

7.5.2.4 迁移至 Objective-C 自动引用计数

将旧项目转换为可以应用自动引用计数（Automatic Reference Counting，ARC）功能。自 Xcode 4 以来，苹果帮向广大帮众提供了这样一门新技术。ARC 消除了明确发送 retain、release 和 autorelease 消息来管理内存的做法，这些消息其实仍然还是存在的，只不过现在都是由编译器自动为我们完成。

和其他转换过程不同，首先 Xcode 会提示开发者选择当前项目中要转换为 ARC 的对象，该工具还会检测一个项目中是否已经使用了 ARC 功能，放置开发者毫无意义地转换操作。

单击“Check”将会启动代码分析功能，分析之后，将会提供一个对话框，告诉我们：

通过以下必要的步骤，该向导将会帮助你将项目迁移到使用 Objective-C 自动引用计数内存管理机制的版本。

Xcode 会首先使用 Apple LLVM 编译器来生成迁移所需要的所有代码变化项。如果该变化项是必需的，那么将会向你展示一个比较页面，来查看这些变化项。

一旦所有的必须的变化项被确认后，Xcode 会更改你对象的编译设置，以让其能够使用 Objective-C 自动引用计数内存管理机制。

7.6　建立工作区

工作区是项目的集合，它可以降低大型应用程序的复杂性。建立工作区并不会修改项目，它所包含的只是项目的引用，从工作区对项目进行删除不会导致项目被删除。当然，在工作区中也可以对项目进行更改。

工作区具有以下几个优点：

- 工作区的所有项目都可以访问同一工作区的其他项目，包括编译信息。
- 可以设置项目之间的依赖关系，这样一个简单的编译命令就可以编译所有选中对象所需的片段。
- 可以将静态库或模块包含进工作区，无论是你自己的还是第三方的。
- 可以将大项目分解成更小的项目，使功能维护和共享更加方便。

很明显，没有两个及其以上的项目，你都不好意思建立一个工作区，不然就使用不了工作区的优点和特性。但是，如果诸位抱有“无形 XX，最为致命”的想法，那么这一节的劝解便大可忽略。

使用工作区可以让应用项目能够访问共享的框架中的所有文件，同时也让诸如调试之类的任务更加轻松。添加该框架到应用程序为应用程序和框架之间创建了一种依赖关系。Xcode 在应用编译之前检查框架是否需要编译。

还有一点，工作区对于多个不相关的项目来说是没有任何帮助的，工作区只对能够共享彼此代码和资源的项目来说有用。

准备好 CrazyBounce-Swift 和 CrazyBounce-OC 两个项目，下面我们将这两个项目归并到一个工作区当中，方便今后的使用。

通过选择 File → New → Workspace 来创建一个工作区，选择工作区的保存地点，并将这个工作区命名为“ CrazyBounce”。当创建完一个工作区后，你可以在其中创建新项目，并且可以向其中添加现有项目。

将我们既有的“ CrazyBounce-Swift”和“ CrazyBounce-OC”两个项目，也就是那两个蓝色图标，拖入到工作区的项目导航器当中。

关闭已打开的项目，并且重新打开工作区。创建好的工作区如图 7-29 所示。

现有的项目转换成工作区请选择 File → Save As Workspace。该项目的当前窗口将被转换为工作区窗口，注意这个操作将不可逆，并且转换过程中不要退出，否则转换会失败。

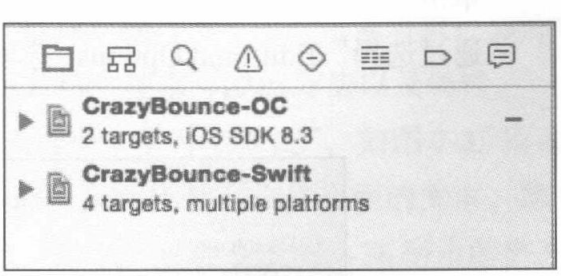


图 7-29　工作区界面

7.7 搜索

很多时候，搜索功能是十分常用的，比如寻找某个变量的所在位置，寻找某个字符串等等。最笨的办法是一行一行目测来寻找，这种方法不但伤眼睛，还容易出错。而掌握 Xcode 的搜索功能可以避免这种情况的发生。

Xcode 提供了几种搜索功能，以对项目中的文本和代码进行查找与替换：

- 单文件搜索。
- 用搜索导航器搜索。
- 重构。
- 全局修改数据（Edit All in Scope）。
- 快速打开。

在上面几节曾经介绍过的“全局修改数据”和“重构”从某种角度来说也包含了一定层次上的搜索功能，因为它们都需要定位到需要修改的数据上。因此我们在此不再加以赘述，而是重点介绍其他几种搜索方法。

7.7.1 单文件搜索

单文件搜索很简单，很类似于平时我们使用过的搜索功能。通过选择菜单栏的 Find → Find 或者 Find → Find and Replace 等一系列的功能，会发现编辑器上方出现了一条搜索栏，如图 7-30 所示。

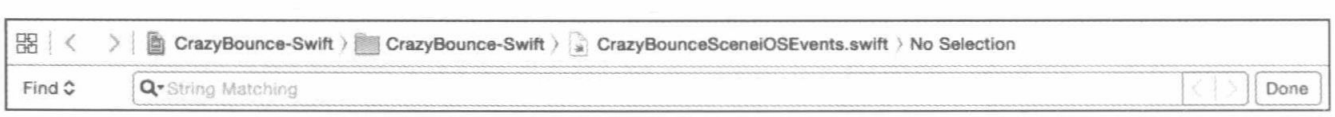


图 7-30 搜索栏

这个搜索栏可以选择两种模式，如果在左侧选择了 Find 模式，那么就实行查找操作，如果选择了 Replace 模式，那么就实行查找与替换操作。单击 Done 按钮或者 Esc 键来关闭搜索栏。

单击搜索栏中的“🔍”放大镜图标，会弹出一个菜单，如图 7-31 所示。

通过选择”Edit Find Options" 选项来进行搜索选项设置。设置界面如图 7-32 所示。

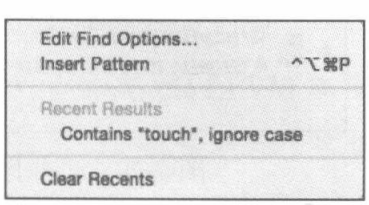


图 7-31 搜索行为菜单

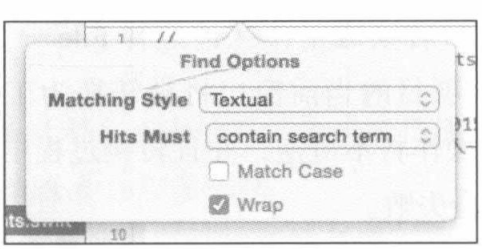


图 7-32 搜索选项设置

Matching Style（匹配模式）中选择搜索字符串的匹配模式，它控制着搜索对象的解释方式。“Textual”是普通的字符串匹配模式，也就是搜索正常的文字字符串。“Regular Expression”是正则表达式匹配模式，也就是根据正则表达式来进行搜索。

Hits Must 则控制着搜索行为，它只能在匹配模式中选择“Textual”时才能使用。通过对搜索行为的控制，可以对字符串匹配进行进一步限定。搜索行为如表 7-2 所示。

表 7-2　搜索行为

搜索行为选项	效　果
contain search term	匹配包含搜索文本的文本序列
start with search term	匹配以搜索文本开头的文本序列
match search term	匹配完全匹配该搜索文本的文本序列
end with search term	匹配以搜索文本结尾的文本序列

此外，搜索选项设置还有两个额外的选项，“Match Case”（匹配大写）则是表明搜索要严格遵守大小写字母的匹配，“Wrap”（折回）则是类似于返回到从头搜索，当本文件内搜索完毕后，那么就会返回到文件头部重新开始搜索。折回发生时，会出现一个大大的“调头”图标，如图 7-33 所示。

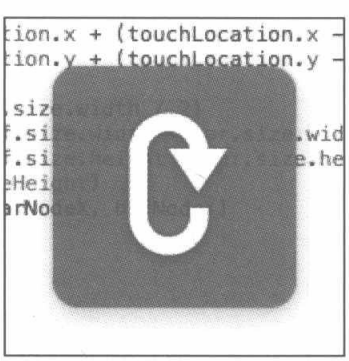


图 7-33　折回标识

7.7.2　搜索导航器

搜索导航器可以用来搜索并显示工程中的字符串信息，在第 2 章已经简要介绍过（见 2.3.3 节），其组成结构如图 7-34 所示。

搜索导航器由以下几个区域组成：

- **搜索规则**：搜索规则用来指定搜索模式、匹配模式、文本匹配方式等等。搜索模式有两种选择，分别是查找（Find）和替换（Replace）。匹配模式有四种选择，分别是字符串匹配（Text）、引用匹配（References）、定义匹配（Definitions）以及 正则表达式匹配（Regular Expression）。

字符串匹配将会搜索项目中所有包含搜索对象的文本，包括类、对象、方法、字符串等等；而引用匹配不会搜索字符串以及基本语句（赋值、查找、循环等）中的搜索对象；方法内定义匹配则直接对类、对象、变量名的搜索，而忽略包含这个搜索对象的方法以及属性。

假设我们使用“allcleanball”进行搜索，匹配模式选择“字符串匹配”，如图 7-35 所示，搜索结果中还包含了"“之类的注释、字符串等等，总之，只要出现在文件里面的文本，都可以用这个模式淘出来。而如果我们换用“引用匹配”查找的话，就会变成图 7-36 所示的情况。搜索结果中只剩下了类和属性了，这个时候，只要使用过“allcleanball”属性或者类的语句，都会被列出来。

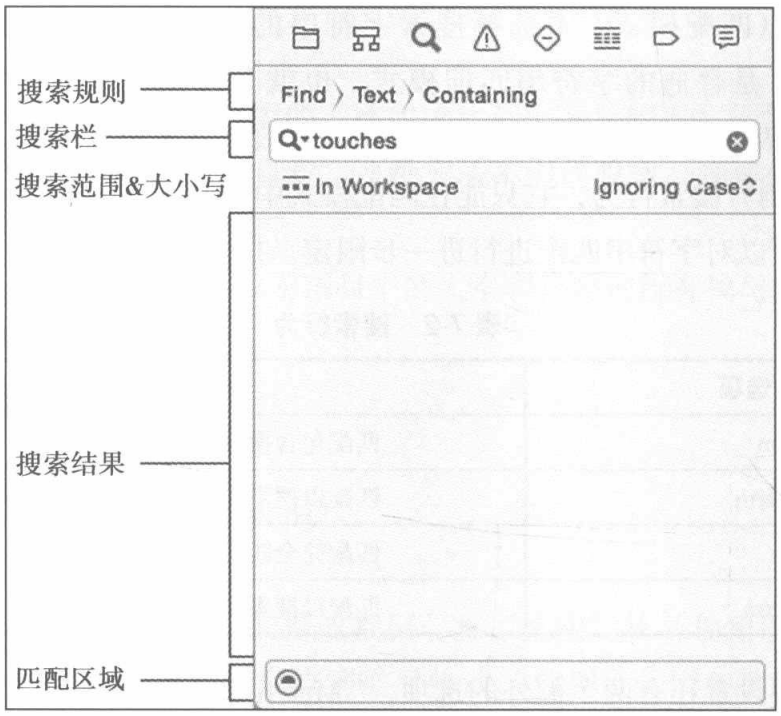


图 7-34 搜索导航器结构

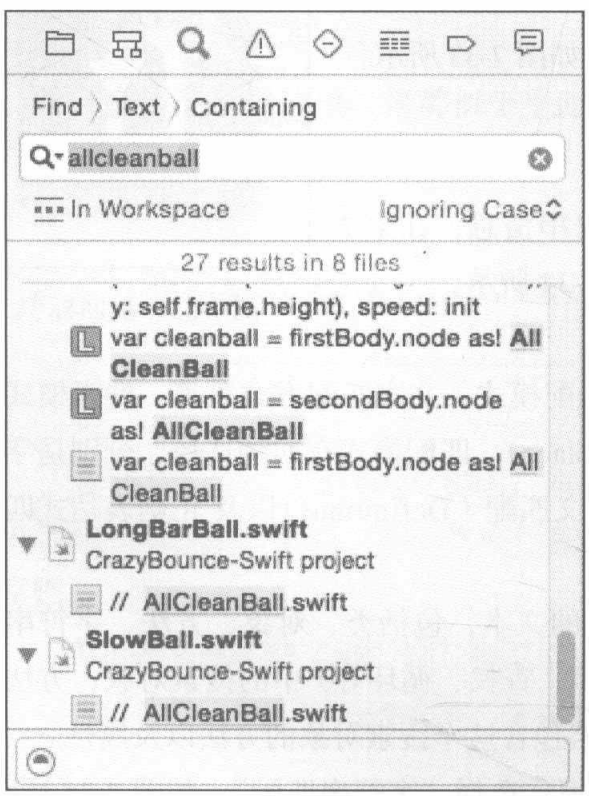


图 7-35 字符串匹配

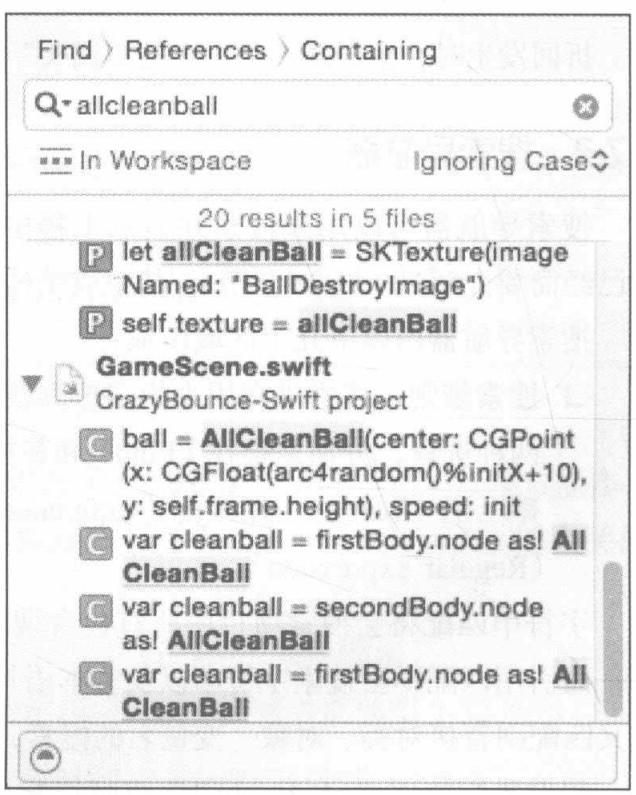


图 7-36 引用匹配

我们换用“定义匹配”进行查找，结果变成图 7-37 所示的情况。这时候，搜索结果中只显示“ allCleanBall”所对应的变量声明、属性声明、类定义了。文本匹配方式在上一节已经介绍过，在此就不赘述了。

- **搜索栏**：搜索栏用来输入要搜索的对象信息。
- **搜索范围**：用来指定执行搜索的区域范围，在这里还可以指定是否搜索项目所用框架内的头文件。
- **大小写敏感设定**：用来指定搜索对象是否对大小写敏感。
- **搜索结果**：用来显示搜索到的结果，通过选中相应的搜索结果，可以在代码编辑器中跳转到其所在位置。搜索结果按照源文件进行排列，其匹配到的内容将以黄色显示，以吸引开发者的注意力。
- **匹配区域**：用来对搜索结果再次进行匹配，借此可以对搜索结果进行更加精确的检索。

使用搜索导航器进行简单搜索和单文件搜索方式大同小异，我们下面只介绍一下搜索导航器的一些特殊功能，如自定义搜索范围、模式搜索。

7.7.2.1　创建自定义搜索范围

开发者可以按照自定义的规则来定义搜索范围，这些范围可以匹配位置、名称、路径扩展名或者文件类型。要创建自定义查找范围，单击当前搜索范围，打开搜索范围选择界面，如图 7-38 所示。

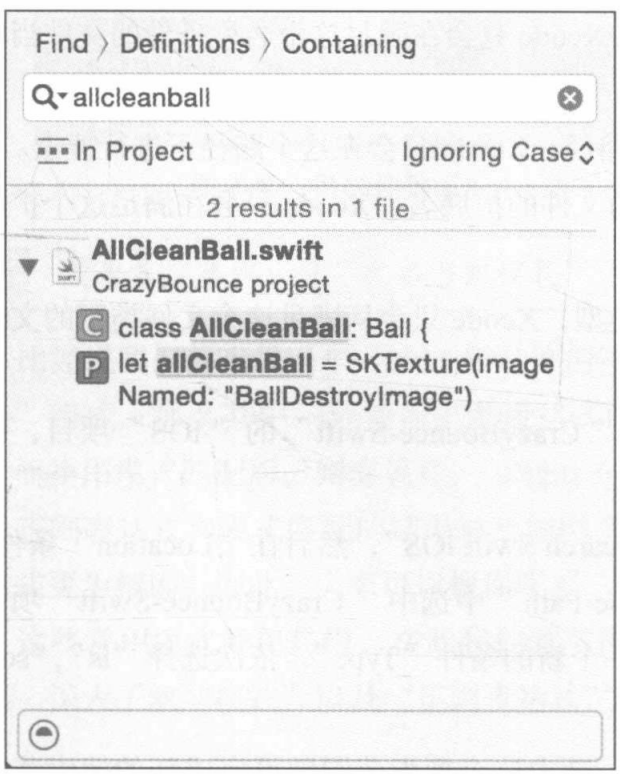


图 7-37　方法内定义匹配

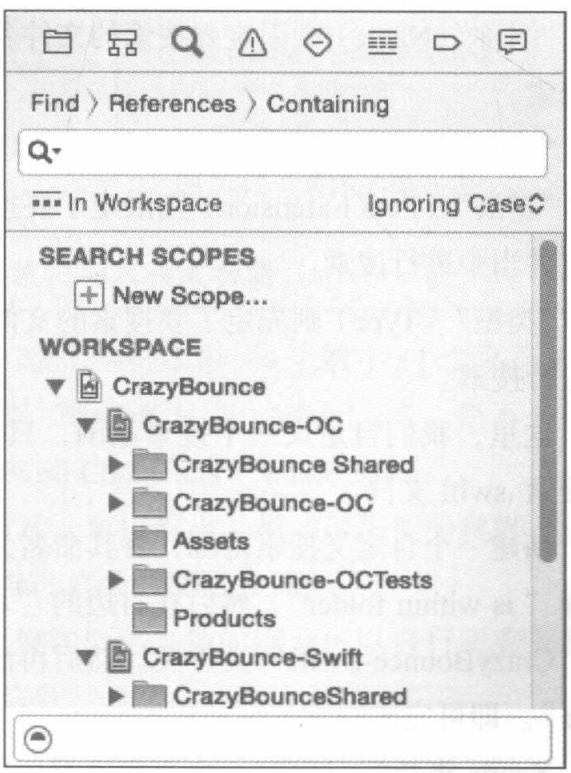


图 7-38　搜索范围选择界面

在搜索范围下方，选择“New Scope”来创建新的搜索范围。这个时候，搜索范围规则窗格会弹出一个弹出框，如图 7-39 所示。

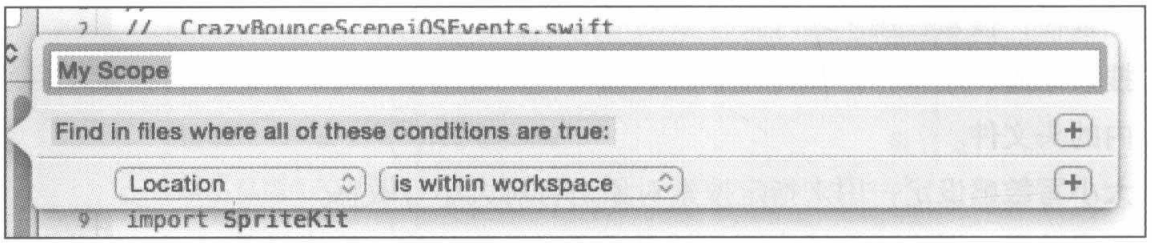


图 7-39 创建自定义搜索范围

通过这个弹出框，我们可以更改这个搜索范围的名称。“Find in files where all of these conditions are true”（搜索满足以下条件的文件）旁边有“+”按钮，从中可以完成搜索条件的指定。可指定五种搜索条件：“位置”、“名称”、“路径”、“扩展”以及“类型”。

“位置”（Location）可以配置 3 种搜索条件，用来指定搜索要在什么地方进行。分别是：“is within workspace”（工作区当中搜索），Xcode 将只在当前项目所在工作区内进行搜索，不会到项目所引用的框架中搜索；“is within workspace and frameworks”（工作区及框架当中搜索），Xcode 不仅在工作区内搜索，还会到搜索项目所引用框架中的头文件；“is within folder”（文件夹当中搜索），选择此项后，还需要在后面的“Choose Path”当中选择要搜索的文件夹，Xcode 将只在这个文件夹目录下进行搜索。

“名称”（Name）则限定要搜索的文件名称，Xcode 只会在满足这个名称条件的文件当中进行搜索。

“路径”（Path）则限定了要搜索的文件所在路径，Xcode 只会在这个路径下进行搜索。

“扩展”（Path Extension）则限定了要搜索的文件的扩展名，Xcode 只会在满足这个扩展名的文件当中进行搜索。

“类型”（Type）则先定了要搜索的文件的类型，Xcode 只会从满足这个文件类型的文件当中进行搜索。

这里，我们自定义一个搜索范围，只搜索“CrazyBounce-Swift”的“iOS”项目，并且只搜索 .swift 文件。

新建一个自定义搜索范围，将其命名为“Search Swift iOS”，然后在“Location”条件中，选择“is within folder”，然后在右边的“Choose Path”中选中“CrazyBounce-Swift”项目中的“CrazyBounce-Swift”文件夹。然后再添加一个新的条件“Type”，依次选择“is”，“source code”。即可完成搜索范围的制定。

Xcode 将会实时保存这个自定义搜索范围，选中这个搜索范围即可完成新的搜索范围的确定。这个时候，搜索范围就变成了我们新创建的搜索范围了，如图 7-40 所示。

尝试搜索一下“gamescene”，可以看到，搜索结果中不再出现 Objective-C 项目中的内容，同时，也没有出现“OS X”项目中的内容。我们使用自定义搜索范围，完成了进一步地搜索条件的配置。

7.7.2.2　模式搜索

在前面介绍的搜索方式中，我们不止一次地提到了“正则表达式搜索”，正则表达式是一种十分强大的模式匹配工具。正则表达式十分复杂，但也十分好用，关于正则表达式的具体语法，大家可以自行去查询相关资料。本书并不会对此进行介绍。

Xcode 也支持正则表达式来执行搜索操作，在搜索导航器中选择“正则表达式匹配模式”，即可开始进行正则表达式搜索。

不过，在 Xcode 当中，更常用的是“模式”（Pattern）搜索。通过选中搜索栏或者搜索导航器的放大镜按钮，在弹出框中选择“ Insert Pattern”（插入模式）。Xcode 为我们显示出几种常用的模式符，如图 7-41 所示。

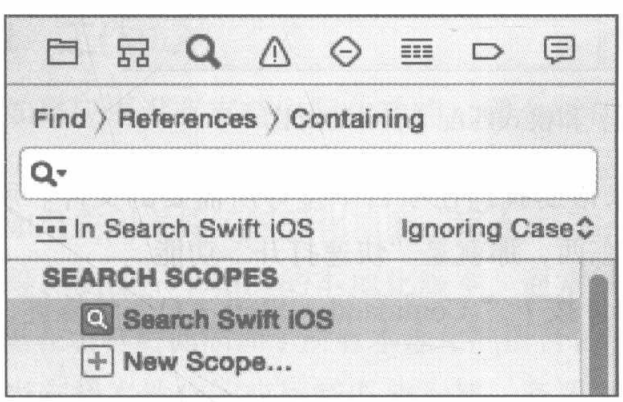


图 7-40　新的搜索范围

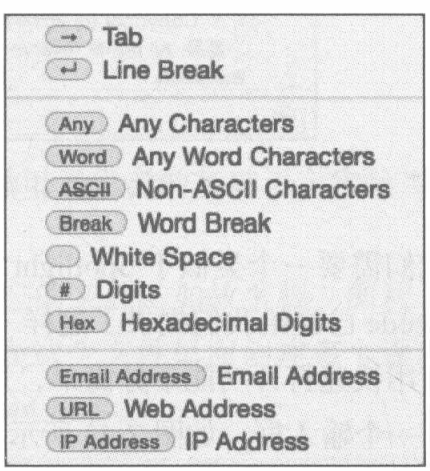


图 7-41　常用的模式符

搜索导航器中，仅“文本匹配模式”才能够使用插入模式功能。

比如说我们要查找一个“2015 年 ”的字符串，这个字符串由“4 个数字 +1 个字符 +1 个空格”组成，那么其中一种使用正则表达式的方式，则应该为“(\d\d\d\d.\s)”。是不是很复杂？而换用模式匹配后，则应该是：“Digits + Any Word Characters + White Space”。

正则表达式和模式匹配的结果对比如图 7-42 所示。可以看出，模式匹配更为模糊，正则表达式更为精确，因此，大家可以根据需求，选择自己喜爱的搜索方式。

这些常用模式简单易懂，在此我们就不再加以赘述，感兴趣的读者可以自行去查阅相关资料，深入了解“模式”以及“正则表达式”相关信息。

7.7.3　快速打开

有些时候，工程里面的目录层级和文件数目会达到令人吃惊的程度。这个时候，再在项目导航器中一个一个寻找文件显然是不现实的，而使用搜索导航器又会显得不直观，也会非常麻烦。

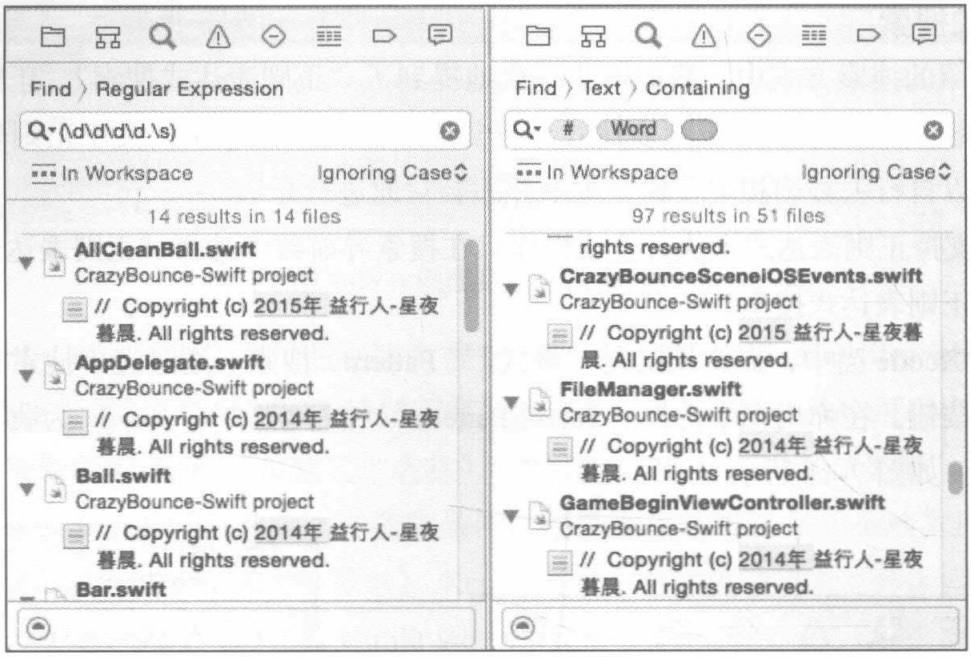


图 7-42 正则表达式（左）和模式匹配（右）的对比

我们需要一个类似于 Spotlight 的东西，来帮助我们在项目中搜寻所需要的文件。所幸的是，Xcode 已经给我们提供了这样一个类似的东西，那就是“快速打开”功能。

使用快速打开功能十分简单，只需在 Xcode 按下“ Command + Shift + O”快捷键，就可以打开一个输入框，如图 7-43 所示。

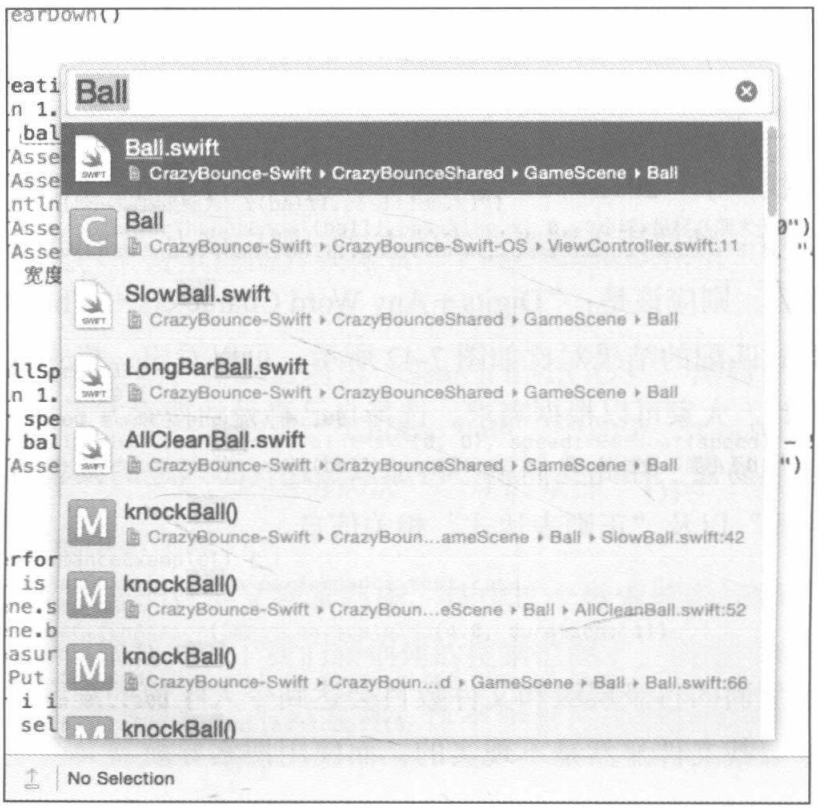


图 7-43 快速打开功能

我们可以借助快速打开来执行一些简单的文件搜索操作，此外，快速打开也可以完成简单的符号搜索功能，它可以匹配方法、变量等一系列的内容。可以说，快速打开就是一个简化版的“搜索导航器”。

7.8　国际化与本地化

如果你希望你的应用能够跨越国界，支持不同的语言和地区，那么国际化和本地化就是非常必要了。所谓国际化，就是为应用提供可以支持不同语言的框架，也就是相当于给它办一个“护照”。而本地化就是为应用添加所支持的地区或者语言的过程，也就是相当于办相应国家的“签证”。

7.8.1　工作机制

如果应用没有支持国际化功能，那么所有的文本都是以开发者的基础语言来呈现的，也就是所谓的**开发基础语言**（development base language）。

如果应用支持了国际化功能，那么 Xcode 就会为相应的文件，如故事板、图片、声音、文档等，创建一个相应的本地化版本，通常情况下就是在这个文件里面创建多个语言版本，也就是在名称后面添加了所有本地化的语言名称，如图 7-44 所示。

当支持本地化的应用需要载入某一资源时，它会检查用户的语言和地区，并查找该资源有没有与之对应的本地化版本。如果有的话，那么就载入这个本地化版本，否则的话就载入基础版本。

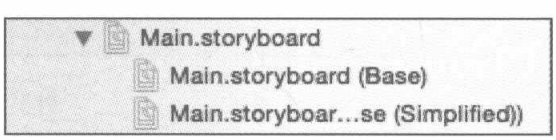


图 7-44　本地化后的 Stroyboard

7.8.2　国际化支持

要支持某一种语言，首先需要将这个语言添加到项目当中。选择项目文件，然后选择项目对象。在 Info 选项卡当中，定位到 Localizations 列表。从这个列表当中就可以管理该项目所支持的语言列表了，如图 7-45 所示。

列表当中的“ Resources”列显示了该语言所“本地化”的文件数量，由此可以看出是否有文件遗漏了本地化。

此外，列表中还提供了一个“ Use Base Internationalization”选项，这项功能用于 iOS 6 和 OS X 10.8 以后的版本。使用此项功能后，Xcode 将会提供“开发基础语言”功能，这样如果开发者没有提供用户所在的语言和地区的本地化版本，就会载入基础版本。如果没有提供这项功能的话，那么开发者必须要为每一个支持的语言创建对应的本地化版本。取消选择这个选项，Xcode 会弹出一个提示框，如图 7-46 所示。

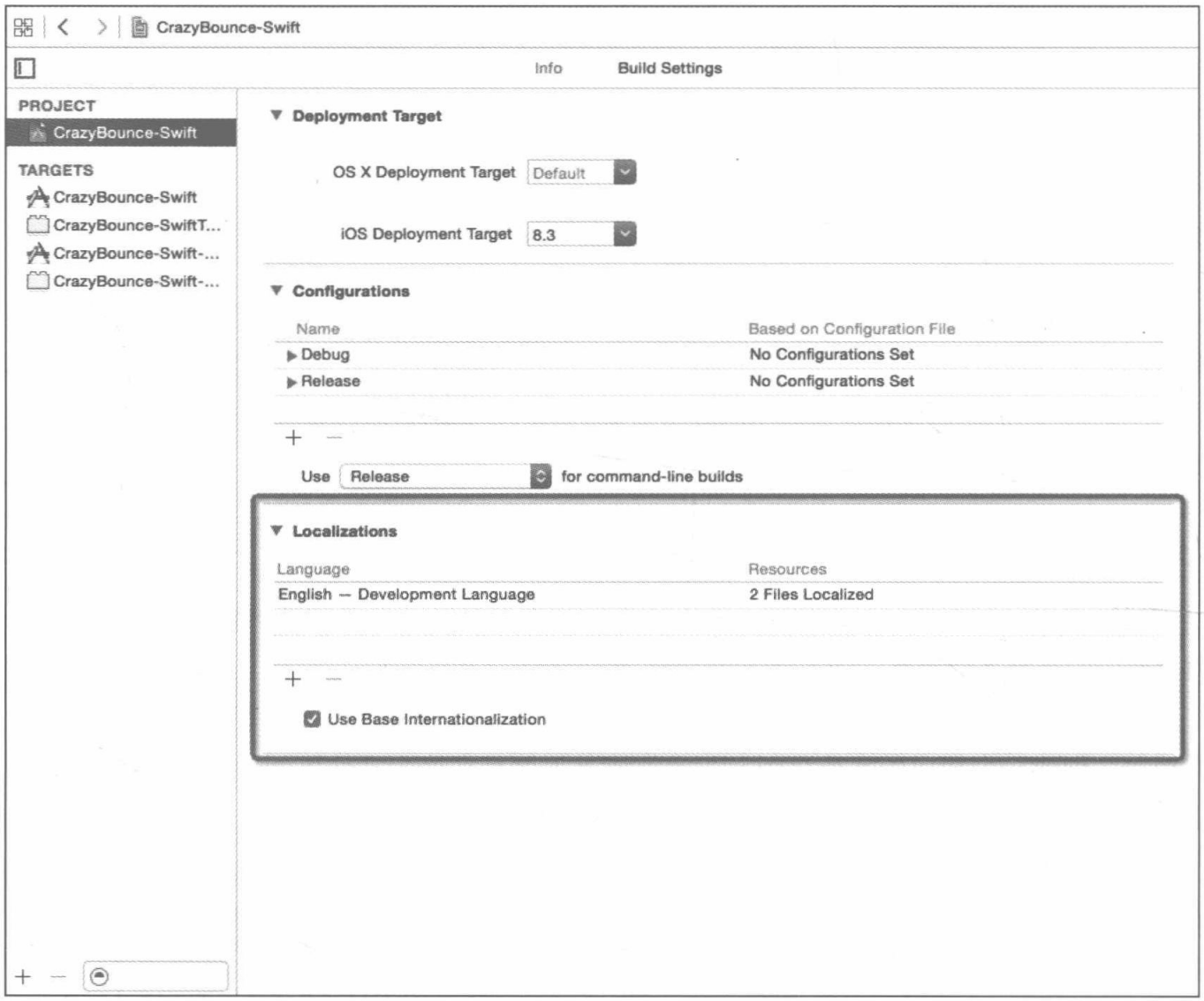


图 7-45　本地化选项

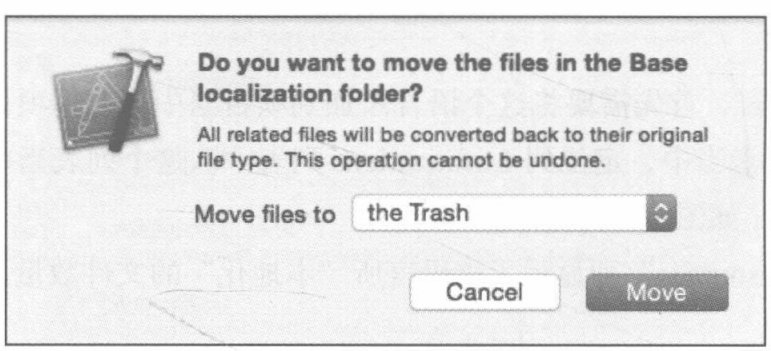


图 7-46　取消本地化提示框

Xcode 提示我们，是否要将文件从 Base 本地化文件夹中移除，这项操作是不可逆的。我们可以选择将这些文件移动到废纸篓（the Trash）还是移动到一个特定的语言文件中。

单击“+”按钮（见图 7-45 的左下角），就可以从弹出的列表当中选择一个要本地化支持的语言和地区。一旦项目当中包含有两个以上的语言或地区，那么这个项目就是支持“国际

化”的了，是不是很简单？这里，我们往项目中添加对“中文简体”的支持，在弹出的列表中选择“Chinese (Simplified) (zh-Hans)”选项。

选择本地化支持的语言和地区后，就会弹出一个对话框，其中提示我们选择要进行本地化文件，如图 7-47 所示。

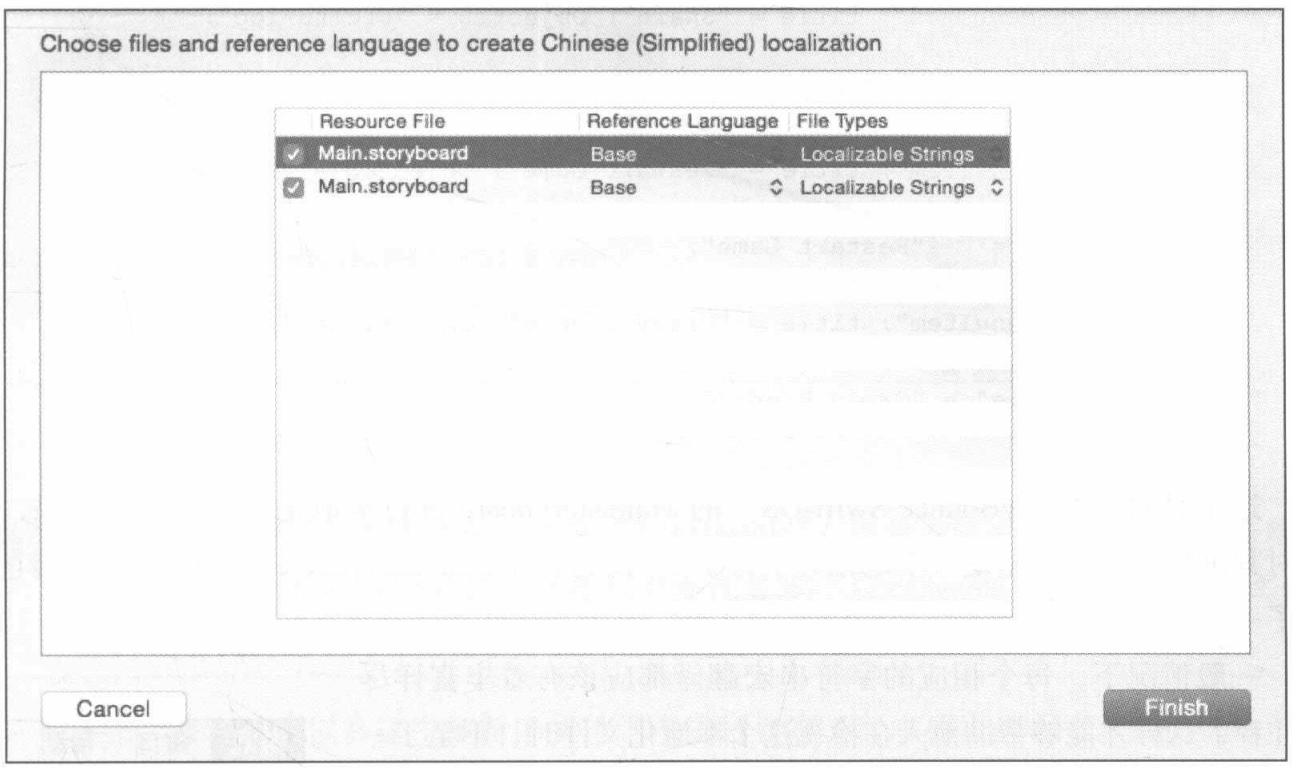


图 7-47　选择进行本地化的文件

Reference Language（参考语言）是选择该本地化文件版本是基于哪一个语言来构建的，这样可以在相似的语言之间采取相同的模板，从而减少修改。

File Types（文件类型）可以选择 Localizable Strings（本地化为字符串）和 Interface Builder …（本地化为…）。前者主要是以“strings”文件进行本地化的方式，是简单的一些文本替换的操作，如图 7-48 所示。后者主要是对相同文件进行本地化的方式，可能还涉及一些用户习惯等等方面的修改，如图 7-44 所示。

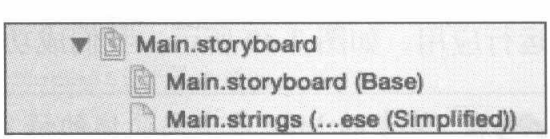


图 7-48　字符串方式的本地化

这里，我们将“CrazyBounce-Swift”的“Main.storyboard”的文件类型设置为“Interface Builder Storyboard”，“CrazyBounce-Swift-OS”的“Main.storyboard”设置为“Localizable Strings”。

7.8.3　字符串本地化

在本地化的工作中，往往会将一些固定显示的字符存储到一个特定的文本文件当中，然

后当使用的时候就直接进行调用，这个文本文件叫做“字符串文件”(String file)。

7.8.3.1 视图的字符串本地化

字符串文件是一个 Unicode 文本文件，这个文本文件由字符串配对列表组成，每项都标识了注释。下面的示例描述了应用程序中字符串文件的格式。

```
/* Class = "NSMenuItem"; title = "Share"; ObjectID = "0tC-tu-fqc"; */

"0tC-tu-fqc.title" = "Share";

/* Class = "NSMenuItem"; title = "Restart Game"; ObjectID = "1UK-8n-QPP"; */

"1UK-8n-QPP.title" = "Restart Game";

/* Class = "NSMenuItem"; title = "Crazy Bounce"; ObjectID = "1Xt-HY-uBw"; */

"1Xt-HY-uBw.title" = "Crazy Bounce";
...
```

这个是对“ CrazyBounce-Swift-OS”的 Main.storyboard 进行本地化之后的部分结果，我们对其进行本地化类型是“本地化为字符串”。我们展开 Main.storyboard 文件，就会发现它新增了一个名为 Main.strings(Simplified) 的文件，这个文件就是字符串文件。

一般情况下，每个相应的字符串宏翻译都应该有着丰富详尽的注释，这样才能够帮助他人在检视这个本地化文件时，不至于一头雾水。

然后注释下面就是所谓的字符串宏。左侧的字符串是键，无论用什么语言，它都能够对应到相应的值，并将右侧的字符串显示出来。这样也就完成了翻译操作。

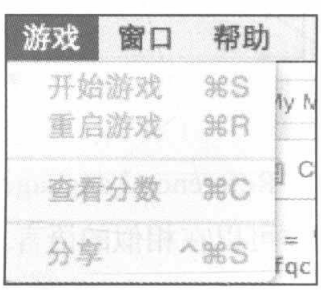


图 7-49 成功的本地化操作

我们根据需要简单修改一下这些字符串的效果，然后编译并运行应用，如图 7-49 所示，我们成功完成了本地化操作。


提示 笔者的 Mac 是中文简体的语言环境，因此可以正常显示出对“中文简体”的本地化的结果。而如果你没有使用该语言环境，或者添加了其他语言的本地化结果，那么你可以前往“系统偏好设置”修改语言环境，或者修改“编译方案”的语言环境，有关编译方案的相关内容，请参阅本书的 10.1 节“编译方案”。

7.8.3.2 自定义文本的本地化

有些时候，我们所需要显示的文本信息并不是在故事板中定义好的，而是在代码中写出来的，这个时候就需要我们对这个代码中的文本信息进行自定义本地化。

首先需要往项目当中添加一个 Strings 类型的文件，也就是通过 File → New → File → iOS

→ Resource → Strings File 来完成创建。然后定位到我们新创建的字符串文件，在工具区域中的文件检查器，找到 Localization 一栏，单击“ Localize”按钮，在弹出的下拉列表中选择想要本地化的目标语言，如图 7-50 所示。

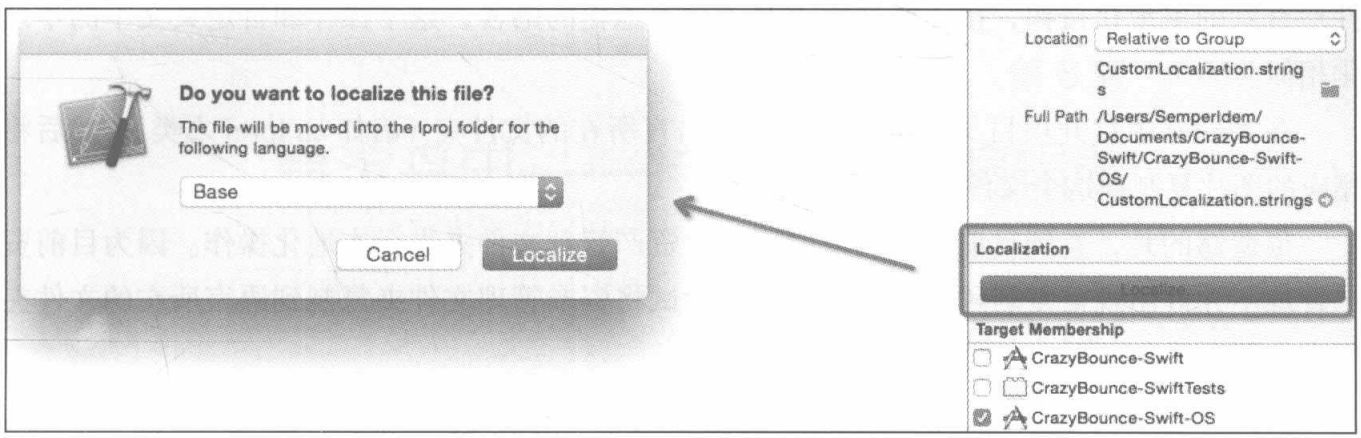


图 7-50　添加自定义本地化

注意　我们新创建的字符串文件最好名为“ Localizable”，因为 NSLocalizedString 方法默认是前往本地化项目内部的应用程序包中查找名为“ Localizable.strings”字符串文件，如果没有找到这个文件的话，那么就无法完成本地化。

接下来，我们就可以在这个字符串文件当中进行自定义文本的本地化了。我们选中其中的一个语言文件，然后在其中输入类似的代码：

```
// 以下两种形式都可以
GOOD_MORNING = "早上好! ";
"GOOD_MORNING" = "早上好! ";
```

接下来就可以使用 NSLocalizedString 宏来完成代码的本地化操作了：

```
// Swift 版本
var string = NSLocalizedString("GOOD_MORNING", comment: "早上好")

// Objective-C 版本
NSString* string = NSLocalizedString(@"GOOD_MORNING", @"早上好");
```

注意　在 Swift 当中，默认的 NSLocalizedString 宏将会显示成 NSLocalizedString:tableName:bundle:value:comment 这样的形式，这是 Swift 接口生成的原因，我们在实际当中，并不需要中间的那三个参数，因为那三个参数均有默认值。因此，手动删除它们即可。关于这个宏的更高级使用方式，不在本书的讨论范围当中。

7.8.4 图像本地化

相比文本来说，图像的本地化操作要麻烦一点，因为图像的本地化操作拥有很多限制。

如果图片文件是单独存放在项目组当中的，那么直接单击这个图片，选择工具区域的文件检查器的本地化按钮，接下来选择需要进行本地化的语言，这个操作和自定义文本的本地化相似。

在 Finder 中打开项目文件夹，找到对应语言所在的文件夹（诸如 en.lproj 之类），然后将相应的图片复制到这个文件夹当中即可。

很遗憾的是，直到目前，我们依旧不能对资产管理文件夹进行本地化操作。因为目前资产管理不允许出现重复名称的图片，如果你尝试将资产管理文件夹复制到语言所在的文件夹当中，那么就会得到一个编译错误：

```
The image set name "xxx" is used by multiple image sets
```

因此，如果要对图片进行本地化的话，那么唯一的办法就是将它们单独放在项目当中，而不是放在资产管理里面。

从外功到内功的修炼，需要前期外功的扎实基础。良辰虽有一身外功蔽体，但并不深知内功的奥妙。修炼内功，好比解开身体内的枷锁，快意江湖，又怎能不打开自由的锁链就妄想驰骋飞踏呢？体内的“气”犹如万匹不羁的烈马，每当夜深，警惕稍微放低，良辰就无法控制这四处乱窜的力量，让他无法安然入睡。内法修炼的路，现在才慢慢展开……

第 8 章 Chapter 8

气沉丹田——持久化存储编辑器

传说南疆巫师可以借助蛊、毒等外物，乃至天地灵气化为自身战力为战。然而借助外物也导致了南巫身体的羸弱，也无法长久为战。不如中原侠客将真气化为丹田，大战三天三夜仍不失精气神。然二者孰优孰劣完全不可考，只能说各有千秋。

因此，对于中原武学而言，锤炼丹田，炼化真气，是必不可少的修炼。

“先天之气宜稳，后天之气宜顺”才是沉稳的底气。气沉丹田讲究炼精化气，积累内气，以成内劲功夫。所以，当脑海中的武功秘籍积累到一定程度，就要学会将其化作内里存储里身体稳固的部分内。

为此，Xcode 为持久化存储提供了可视化的编辑器，包括“属性列表”编辑器和“Core Data 数据模型”编辑器。借助这些编辑器，“秘籍”才能有效地化作“内力”，我们也就能够更为方便快捷地管理数据存储。

8.1 属性列表

属性列表（Property lists）一般情况下存储着项目的相关设置。当创建了一个新的 iOS 或者 OS X 项目的时候，Xcode 便会自动创建一个名为 Info.plist 的文件。这个文件一般情况下位于项目的 Supporting Files 目录下，它里面包含了不少重要的项目设置：包括应用名称、应用图标、版本号等。

当然，作为持久化存储的一种方式，还可以创建新的自定义属性列表来存储数据。事实上，属性列表比它看上去的要复杂得多，因此，在本章中只是介绍属性列表视图编辑器的相

关内容。

8.1.1 属性列表简介

属性列表实质上是以 XML 格式存储的文件，只不过后缀名为“.plist”而已。属性列表中的内容是以字典的形式展现的，其键值要么是一个普通值，要么是一个包含内容的字典或数组。

如果你不熟悉字典的概念，你需要明白属性列表基本上是依靠键值对来存储数据的。所谓键（key）也就是指索引，是一个字符串，你可以通过这个索引读取到其对应的值；所谓值（value）也就是属性列表所存储的数据，比如字符串、数字、字典、数组等。

在属性列表当中，数组中的项目通常都是以数字来命名，比如说 item 0、item 1 之类的方式。非连续的数字是不允许使用的。

属性列表通常情况下用来在应用初始化时加载应用设置以及用户偏好设置，虽然属性列表的使用非常自由，但是一般情况下它并不用来进行数据持久化。当然，对于某些属性列表来说，它的某些键是固定的、有标准的。对于这部分属性列表来说，就不能随意删减其键，并且新添加的键也都有所限制。

在 Xcode 5 中，打开属性列表文件会同样打开该属性列表文件所在的项目。但是目前在 Xcode 6 当中，打开属性列表文件并不会连同项目一起打开。

8.1.2 项目属性列表

几乎每一个应用都会包含有 Info.plist 文件来存储应用设置。项目属性列表中的设置项都是标准的，如图 8-1 所示。这个文件一般位于“项目名→ SupportingFile”目录下

Key	Type	Value
▼ Information Property List	Dictionary	(15 items)
Localization native development r...	String	en
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	1-xing.com.$(PRODUCT_NAME:rfc1034identifier)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	2
Application requires iPhone envir...	Boolean	YES
Main storyboard file base name	String	Main
▶ Required device capabilities	Array	(1 item)
Status bar is initially hidden	Boolean	YES
▶ Supported interface orientations	Array	(1 item)
▶ Supported interface orientations (...	Array	(4 items)

图 8-1 Info.plist 文件

当应用启动时，应用将会读取它自身的项目属性列表，然后根据读取到的信息来进行应用初始化，并且也会根据系统平台和设备的不同来决定运行时的特征等功能。

通常情况下，我们并不需要去修改项目属性列表当中的键值，因为 Xcode 默认的项目属性列表中已经包含了基本的信息，并且我们也可以在对象设置中对某些键进行修改。

8.1.2.1　编辑项目属性列表

项目属性列表主要分为三栏，分别是 Key（键）、Type（类型）、Value（值）。类型就是某个键所对应的值应该遵守的类型规范。

要修改项目属性列表，只需找到要修改的键所对应值，双击进行修改即可。注意，一定要保证所修改的值满足该键的限定。

OS X 项目和 iOS 项目所使用的键值并不完全相同，因此在编辑项目属性列表的时候，要根据项目的平台不同来进行不同的设置。

关于项目属性列表中标准键的相关内容，请参阅苹果官方文档“Aboat Info.plist Keys”。

8.1.2.2　增加新的可选键

要增加新的可选键，只需要进行如下步骤即可：

1）右键单击某一键值。只要该键值不是数组或者字典，那么可以在任何一个键值上进行创建，项目属性列表中的键值是不分次序的。如果右键单击的该键值是展开的数组或者字典，创建出来的可选键将包含在数组或者字典当中。

2）在右键弹出的菜单中选择 Add Row。

3）在新创建的键值弹出的列表中选择某个键值。

4）进行键值的编辑。

当然，选中某一键值之后，单击键列上出现的“+”按钮，也可以完成创建操作。如果要删除某一键值的话，那么单击“–”按钮即可。

如果打算创建和数组或者字典位于同一层级的键值，那么请将数组或者字典键值设置为“合并”状态。

8.1.3　创建属性列表

创建新的属性列表与添加文件的方式相同，只需添加 Resource 目录下的 Property List 文件即可。关于属性列表文件模板的相关内容，请参阅本书的附录 C 的 C.1 节“文件模板”。

对于 iOS 系统来说，还可以创建应用设置束（setting bundle）文件，这同样也是一个属性列表文件，但是你可以用它来存储和管理应用偏好设置。关于应用设置束的相关内容，请参阅苹果官方文档“Preferences and Settings Programming Guide”。

对于新创建的属性列表来说，可以选择属性列表的类型。在属性列表编辑器的空白处单

击右键，在弹出的菜单中选择“ Property List Type”，便可以在当中选择相应的属性列表类型了，如图 8-2 所示。

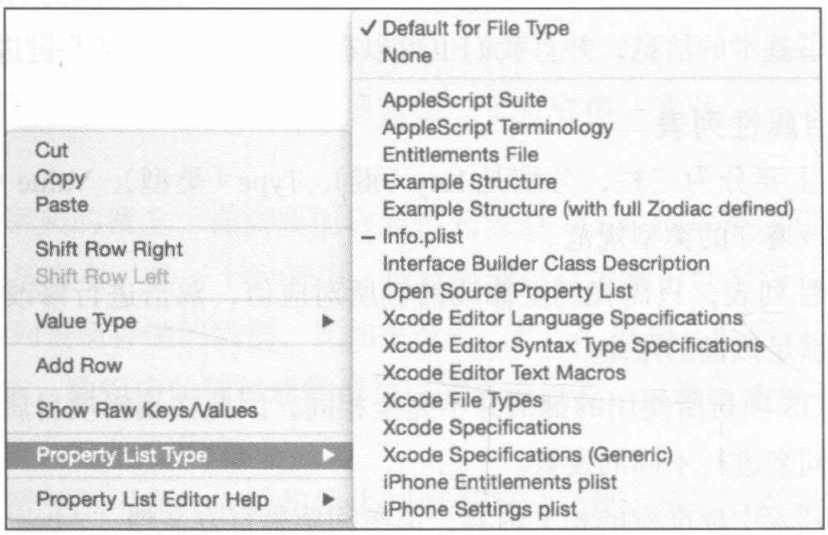


图 8-2　属性列表类型

此外，还可以使用“ View the raw key name”功能来查看某个键的类型名称，一般情况下键的类型名称和名称是相同的。但是对于某些键值来说，它们的键名往往是对该键的文本描述，而不是该键所对应的类型，比如 Cocoa 类的相关键。因此，使用这个功能可以方便编辑。

例如，说对于 Info.plist 来说，默认情况下，Information Property List 显示的是诸如“ Bundle name”之类的键描述信息，勾选上“View the raw key name”选项后，就会发现实际的键名就显示了出来。例如，原来的“Bundle name”对应的实际键名为“CFBundleName”。

8.2　Core Data 模型

Xcode 中可以对数据进行建模，将数据按照一定的规则进行管理和存储。建立的数据模型一般需要定义其数据对象以及对象间的关系。之后便可以应用当中使用数据模型来管理数据了。Xcode 当中管理数据模型有很多方法，但是最为常见的解决方案则是 Core Data。在 Xcode 当中可以很轻松、方便地使用 Core Data。

Core Data 的官方定义是“对象图管理和持久化框架”，实际上 Core Data 就是为 SQLite 关系数据库提供了一层图形化的管理界面以及面向对象的数据访问处理方法。同时 Core Data 也支持其他类型的数据存储方法，比如用 XML 格式进行存储。此外，Core Data 也支持扩展自定义存储类型。

当然，Core Data 是一个比较复杂的工具，因此本章将只是重点介绍用于开发和维护数据模型的工具，有关 Core Data 的具体使用方式，请参阅苹果官方文档“Core Data Program-

ming Guide”。

8.2.1　相关术语介绍

Core Data 拥有自己专门的一套术语，虽然这些术语非常的繁杂冗余，但是了解这些术语能够有效地帮助你理解 Core Data 模型编辑器的相关选项，并且能够帮助你在之后学习 Core Data 相关知识时更加轻松。有如下几个基础术语：

- **数据模型**（data model）是对数据和数据组件之间关系的一种定义，简单来说，数据模型定义了数据的结构、数据的组织方式、数据之间的关系，以及数据所具有的行为特征。
- **持久化存储**（Persistent store）是用托管对象模型进行数据存储的一种存储方式。Core Data 支持三种存储类型：XML、SQLite 或者二进制。托管对象模型并不关心数据是如何持久化存储的，甚至并不关心其是否持久化。
- **持久化存储协调器**（Persistent store coordinator）为一个或多个托管对象上下文提供了访问接口，使其下层的多个持久化存储可以表现为单一的聚合存储。一个托管对象上下文可以基于持久化存储协调器下的所有数据存储来创建一个对象图。持久化存储协调器只能与一个托管对象模型相关联。
- **托管对象**（managed object）是 MVC 架构中的“模型对象”，其代表了存储中一个实际的记录。因此，对托管对象的抽象化描述就是托管对象模型。

  托管对象模型允许 Core Data 在持久化存储中的记录与你在应用中使用的托管对象间建立映射。也可以说，托管对象模型是实体类型对象的集合，实体类型根据实体名称、用来代理实体的类名称以及实体所拥有的属性来描述一个实体对象（也可以看做是数据库中的一个表）。

  托管对象模型（MOM）是 Core Data 用于描述应用程序数据模型的机制。它可以用代码进行描述，但更多地是使用 Data Model（数据模型）编辑器来进行构建。如果在创建项目时勾选了“Use Core Data”选项的话，那么 Xcode 就会为项目添加必需的代码（代码存放在 AppDelegate 类当中），并且添加一个与项目同名的数据模型文件（xcdatamodeld），这个扩展名的全称是“Xcode Core Data model directory”（Xcode Core Data 模型目录），它实际上是一个保存应用程序数据模型的文件夹。
- **实体**（Entity）是一类对象及其属性的描述，托管对象（managed object）是实体的一个实例。实体在数据模型编辑器当中创建，而托管对象则是在代码当中创建。实体和托管对象之间的差别类似于类与对象之间的差别。实体的名称必须以大写字母开头，并且不能出现除数字、字母以及下划线之外的文字。

实体拥有三个属性，分别是**特性**（Attributes）、**关系**（Relationships）以及提取**属性**（Fetched

Property），介绍如下：

- **特性**用于保存数据，由一个名称以及类型组成，和代码中的示例变量作用基本相同。特性的名称必须以小写开头，同样名称也只能出现数字、字母以及下划线。
- **关系**用于定义实体与实体之间的联系。关系可以是一对一、多对一或者多对多。关系的命名规则和特性相同。举个例子，关系就像音乐播放软件当中的播放列表，歌曲可以根据一定的规则放在播放列表当中，歌曲可以存在于多个播放列表当中，删除播放列表并不会导致歌曲被删除。在添加歌曲的时候，一定要切实访问到特定的歌曲文件。
- **提取属性**是关系的一个备选方法，用提取属性可以创建一个可在提取时被评估的查询，从而确定哪些对象属于这个关系。关系在实体加载的时候同时加载，而提取属性则是按需加载。还是上面那个例子，iTunes 中拥有一个名为 Smart 播放列表的玩意儿，它可以基于某种规则，智能地对播放列表中的歌曲进行管理，在添加歌曲前，这个播放列表并不知道这个歌曲是否存在。
- **托管对象上下文**（managed object context）是创建、读取、修改以及存储托管对象的重要区域。Core Data 要求应用当中至少拥有一个托管对象上下文。

  对于多个上下文，可以用某个合并策略来进行合并，也就是所谓的并持久化。合并策略描述了冲突发生时应该如何处理。
- **提取请求**（Fetch Request）相当于一个查询语句，将你想要提取的托管对象中的实体通知给托管对象上下文；它还可以提取其他内容，例如对象属性的限制信息值、以及对象所返回的次序值。这些提取操作与数据库 SELECT 语句中的表名、WHERE 语句、ORDER BY 语句操作很类似。可以通过向托管对象上下文发送一个消息通知来运行提取请求，然后上下文便会返回一个数组，其中包含有满足请求的对象（如果存在的话）。
- **配置**实际上是一系列实体的集合，通过使用配置，可以让数据模型包含许多不同的实体组合，从而让应用有选择性地读取只包含所需实体的数据模型。

8.2.2 数据建模编辑器

一般情况下，在创建新项目时如果勾选了“Use Core Data”这一选项，那么这个项目就是具备了 Core Data 的功能了。我们提供的“CrazyBounce-Swift”示例在创建过程中选中了这个选项（虽然实际项目中并没有用到任何 Core Data 的内容）。

如果在创建项目时并没有勾选“Use Core Data”这一选项，那么就需要手动支持 Core Data 这个功能。本书并不会讲述如何手动支持，因为这涉及如何正确地创建

NSManagedObjectContext、NSPersistentStoreCoordinator 和 NSManagedObjectModel 对象，需要添加不少的代码和设置。本章之后的内容是以使用了 Xcode 提供的 Core Data 功能为基础来讲述的。

一般情况下，Xcode 已经给我们提供了一个数据模型文件了，名为“(项目名).xcdatamodeld”。但是如果我们还想要创建新的数据模型文件，那么与添加文件的方式相同，只需添加 Core Data 目录下的 Data Model 文件即可。关于 Core Data 文件模板的相关内容，请参阅本书附录 C 的 C.1 节“文件模板”。

当你选中这个数据模型文件之后，数据建模编辑器便会显示出来，如图 8-3 所示。默认的数据模型文件是空的。

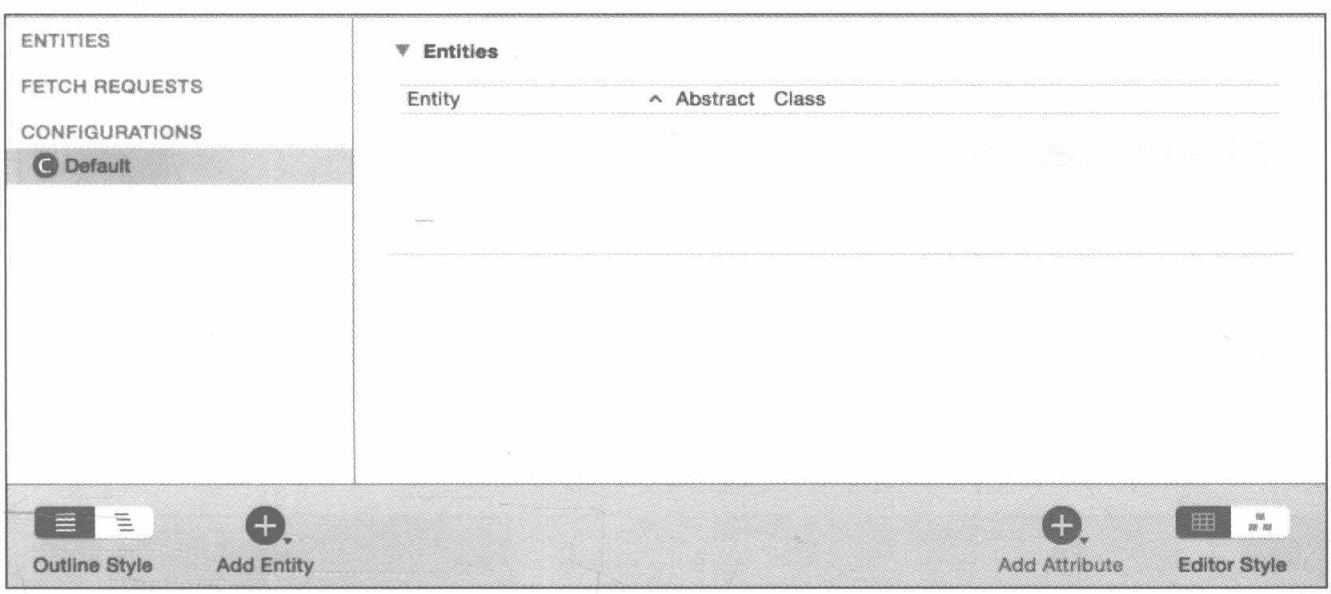


图 8-3　数据建模编辑器

数据建模编辑器中，最左边的是顶层组件窗格（Top-Level Compinents Panel），其中列出了数据模型中所定义的实体、数据提取请求和相关配置。在顶层组件窗格旁边的则是编辑器窗格，可以对实体、提取属性和配置中的属性进行修改。在最下面的是选项窗格，分别是大纲样式（Outline Style）、添加实体（Add Entity）、添加属性（Add Attribute）、编辑器样式（Editor Style）。

大纲样式可以切换顶层组件窗格的显示样式是列表还是层级。在层级大纲样式中，可以看到配置之间的层级关系。

如果只有一个配置的话，那么是不能够进行大纲样式的选择的。

添加实体的含义自然不用多说，是用来添加实体的，如果按住这个按钮不放，那么就会弹出一个菜单，就可以依次选择添加实体、添加提取请求（Add Fetch Request）、添加配置

（Add Configuration）。无论选择哪一个，按钮的外观便会变成与之对应的那个功能的外观。同样，添加提取请求和添加配置也可以通过在 Editor 菜单中选择。

然后接下来是添加属性（Add Attribute）按钮，与添加实体按钮相似，按住按钮不放将会弹出一个菜单，分别是添加属性、添加关系（Add Relationship）、添加提取属性（Add Fetched Property）。同样，单击其中的任何选项都会影响按钮的默认行为和外观。

只有在顶层组件窗格中选中了一个实体后，添加属性按钮才可用。

接下来是编辑器样式的选择，你可以在表格样式和图形样式之间切换。图形样式的界面如图 8-4 所示。图形样式主要用来表示实体之间的关系和继承性。

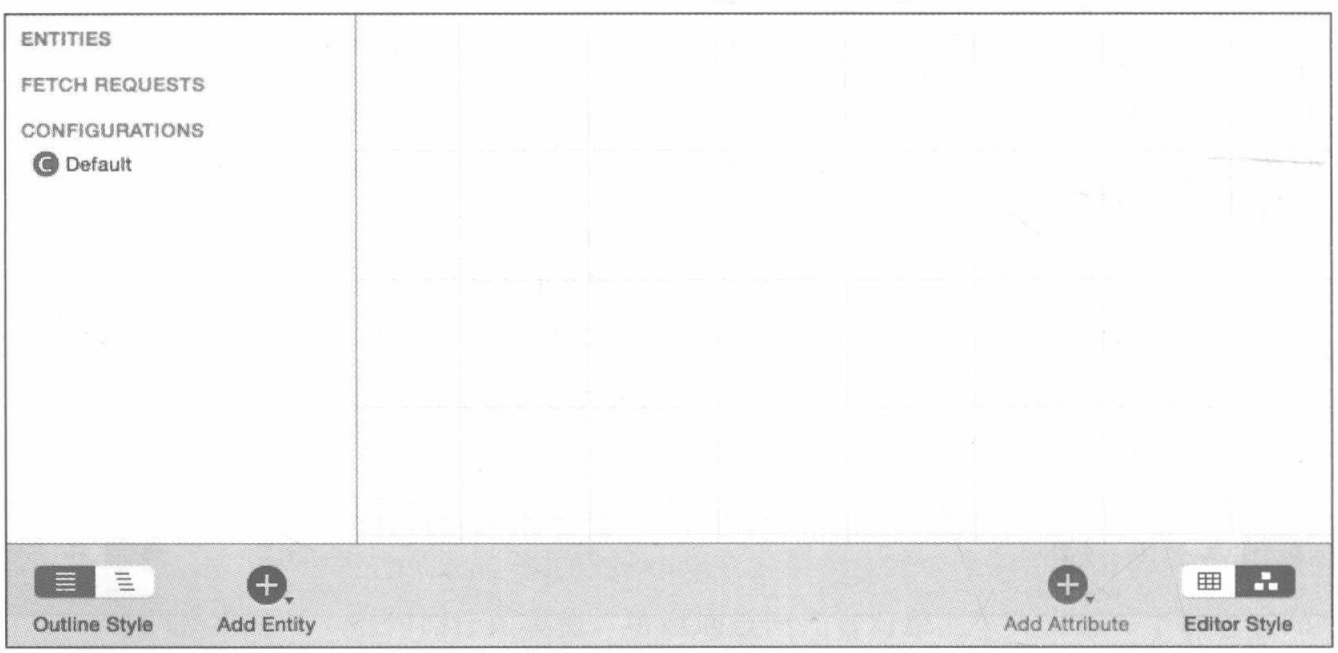


图 8-4　图形样式

8.2.2.1　添加实体

添加实体的操作前面已经介绍过了，单击选项窗格中的添加实体按钮，就可以创建实体了，然后在顶层组件窗格中的实体列表中就会出现我们刚刚创建好的实体，然后修改其名字即可。也可以通过选择 Editor → Add Entity 来实现同样的操作。

这里，我们添加一个叫 Student 的实体，如图 8-5 所示。

在 Xcode 右侧的数据模型检查器中，我们可以看到，如图 8-6 所示，实体主要有以下几种设置：

- Abstract Entity：选项这个勾选之后，该实体将不能够在运行时生成托管对象，这个和抽象类的概念相同，主要是用于继承。
- Parent Entity：可以在这里选择这个实体所应该继承的实体，子级实体拥有父级实体的

所有特征。这个和类继承的概念相同。

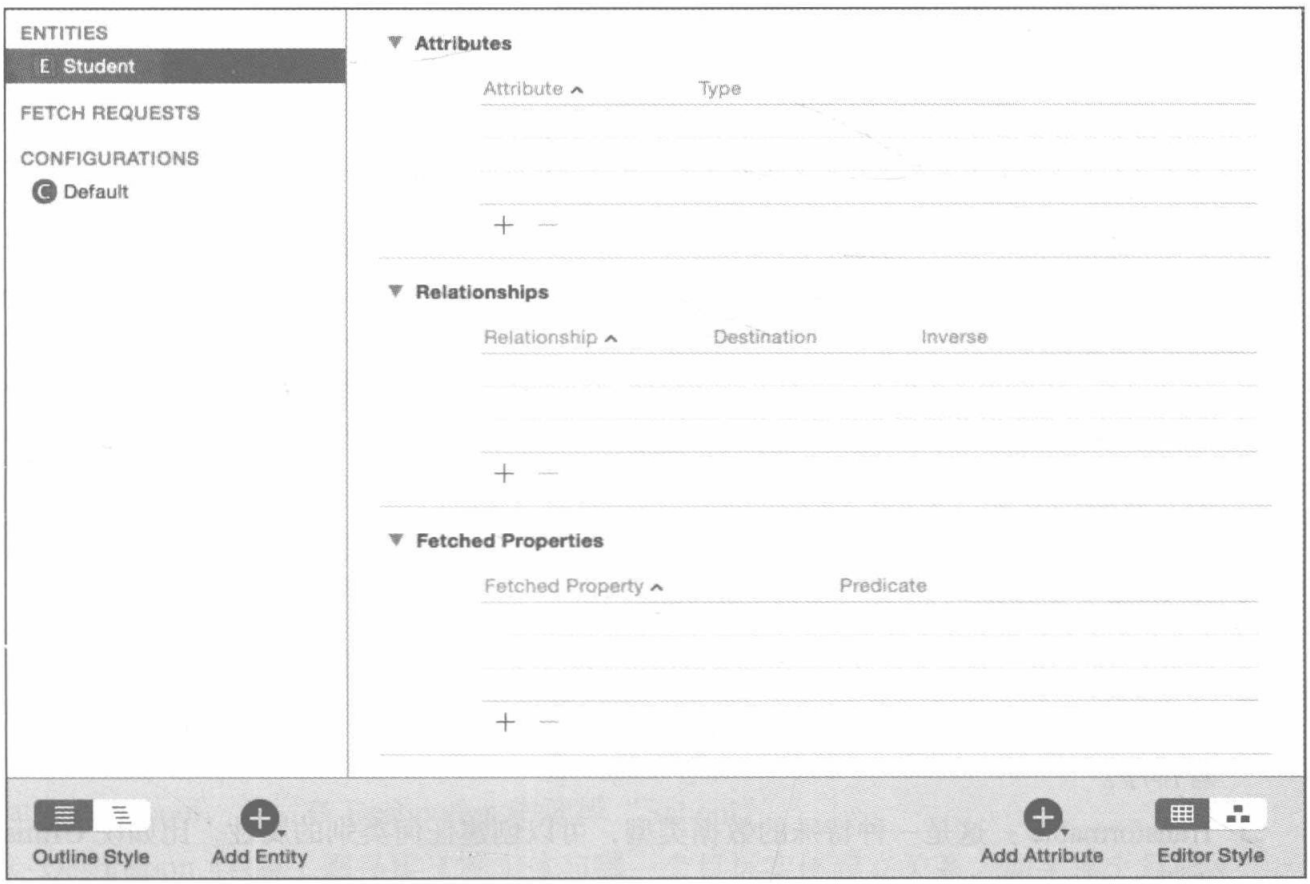


图 8-5　添加 Student 实体

8.2.2.2　添加属性

添加属性的方法有很多种，即可以单击选项窗格中的添加属性按钮，也可以通过选择 Editor → Add Attribute 来实现同样的操作，还可以在编辑器窗格中找到属性列表，然后单击“+”号按钮创建一个新的属性。

在这里，我们给 Student 实体创建“name”（String）、“school”（String）和“age”（Integer 32）属性，如图 8-7 所示。

创建完属性后，就可以修改其名字及类型了。属性类型指定了属性能够存储的数据种类。属性所拥有的数据类型包括以下几种：

- Integer：包含有 Integer 16、Integer 32、Integer 64，它们的唯一区别是所支持的最大和最小值。一般而言，应当尽可能小地选择满足需求的整数，

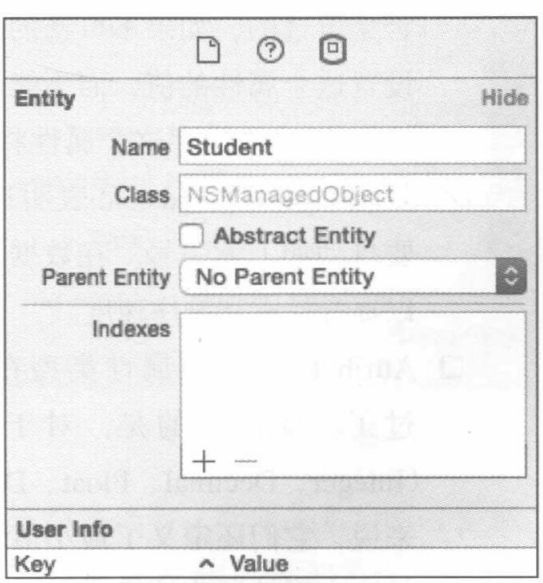


图 8-6　实体检查器

以便能够节省空间和加快读取速度。

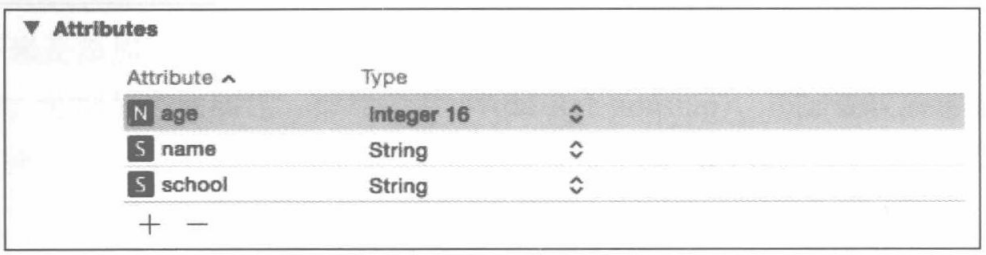


图 8-7 为 Student 实体添加属性

- Decimal、Double 和 Float：这三个属性均用来存储小数，Double 和 Float 是用浮点来存储小数，唯一的不同就是它们的精度不同。Decimal 则是采用定点的方式来存储小数，防止四舍五入情况发生。
- String：String 可以存储几乎所有语言的文本，因为其内部采用 Unicode 编码。
- Boolean：布尔值，用于逻辑是非的判断。
- Date：在 Core Data 中，日期和时间戳都可以用 Date 数据类型来存储。
- Binary：用于存储二进制数据，要注意的是，没有办法对 Binary 数据类型进行检索或者排序。
- Transformable：这是一种特殊的数据类型，可以创建任何类别的属性，比如说 UIImage 或者 UIColor 等。

在右侧的数据模型检查器中，我们可以看到，如图 8-8 所示，属性主要有以下几种设置：

- Properties：有三个选择，Transient 是用来表明是否设定为临时属性，临时属性并不会保存在持久存储区域当中，只是作为一个暂时的变量而存在，在不使用时 Core Data 会自动将其丢弃；Optional 表明这个属性是否是可选的，如果不可选的话，那么就必须设置这个属性的值，而不能为空，如果该属性值为 nil，保存这个属性将导致错误产生，阻止其保存；Indexed 表明这个属性是否需要自动加上索引号。在数据库当中，索引可以提升检索和排序速度。
- Attribute Type：属性类型在上文已经介绍过了。要注意的是，对于标量数值类型（Integer、Decimal、Float、Double 以及 Date）来说，它们还定义了最小值和最大值，也就是这个属性所应该满足的一些约束条件；对

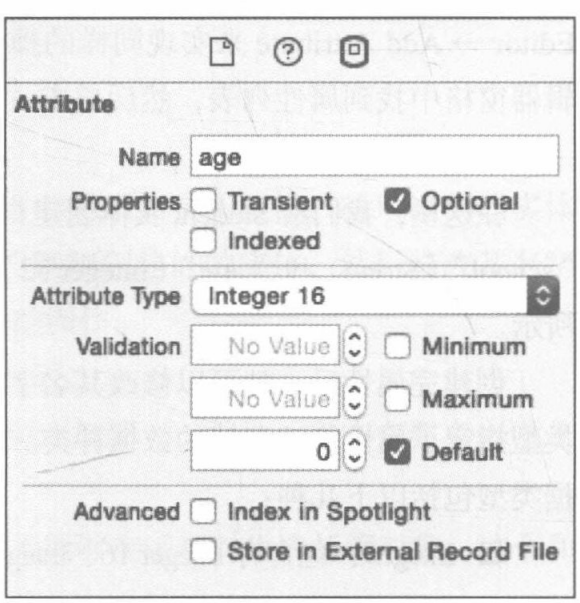


图 8-8 数据模型检查器

于字符串类型来说，则是定义的最小和最大长度，此外，它还可以使用正则表达式来进行限制。

注意　正则表达式是一个非常复杂的话题，如果你对正则表达式感兴趣的话，可以前往百度百科或者维基百科查询相关的知识。

- Index in Spotlight：勾选此选项后，这个实体关系将可以在 Spotlight 当中查询和管理，这个功能只能在 OS X 应用中使用。
- Store in External Record File：决定这个实体关系是否存储在额外的记录文件当中，这样可以提高加载速度。

8.2.2.3　添加关系

要添加关系，可以单击选项窗格中的添加提取关系按钮，也可以通过选择 Editor → Add Relationship 来实现同样的操作，还可以在编辑器窗格中找到关系列表，然后单击“+”号按钮创建一个新的关系。

在此之前，我们还要创建一个 School 实体，然后这个实体里面包含有“name”（String）、“studentNumber”（Integer 32）属性。然后我们在“Student”里面，添加一个关系，命名为 relationToSchool，然后在 Destination 中选择“School”。

Destination（目标）则是定义该实体与哪一个目标实体建立关系，如果那个目标实体已经与该实体建立过关系的话，那么 Inverse（反向关系）一栏就会显示那个关系的名称。

在右侧的数据模型检查器中，我们可以看到，如图 8-9 所示，关系主要有以下几种设置：

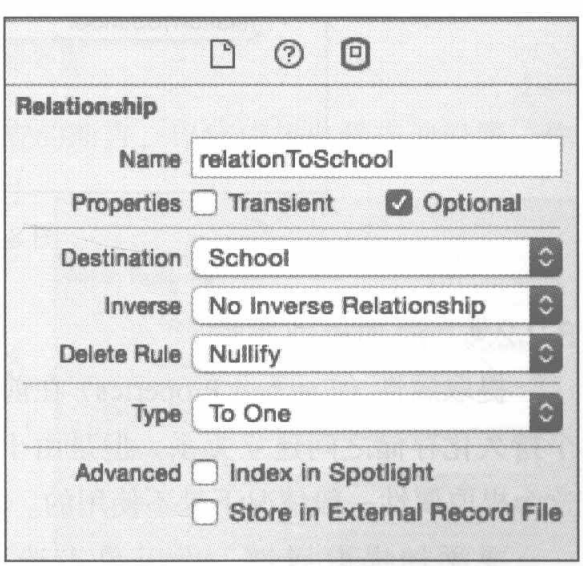


图 8-9　关系检查器

- Destination：上面已经介绍过这个选项，目标是定义该实体要与哪一个目标实体建立关系。
- Inverse：反向关系，反向关系的意思简单来说，就是不论哪一个实体发生变化，都会导致与它有反向关系的实体发生变化，也可以称之为“双向关系”。
- Delete Rule：这个规则定义了当关系中某个对象被删除之后要如何处理，Core Data 主要有 4 种删除规则：

作废（Nullify）：这是默认的删除规则，当一个对象被删除之后，反向关系会随之更新，不再指向任何东西。如果反向关系是个一对一关系，那么将被设置为 nil；而如果这个反向关系是一对多关系，那么被删除的对象将从反向关系中被移除。这个规则能够保证没有引用会

指向一个已经被删除的对象。

无操作（No Action）：如果从关系中删除一个对象，那么不会对其他对象产生影响。这个规则仅仅适用于没有反向关系的一对一关系当中。因为采用这个规则，可能会导致反向关系最终指向一个不存在的对象。

连锁（Cascade）：在删除一个托管对象的时候，所有存在于关系中的对象都将会被删除掉。能使用这个规则的场景不多，也就是在关系中的对象只在这个关系中出现，并且没有其他特殊理由时使用。防止删除一个对象后导致连锁删除的情况发生。

拒绝（Deny）：当要删除的某个对象仍然与其他对象有关联时，采用这个规则会拒绝执行。

- Type：定义关系类型是一对一（To one）还是一对多（To many）。一对一关系很简单，比如说因为一个人的生母只能有一个，那么这个人与其生母之间的关系是一对一关系；而一个人可以有多个孩子，那么这个人与其孩子之间的关系是一对多关系。至于前面介绍过的反向关系，那么生母与这个人之间的关系，以及孩子与这个人之间的关系就是反向关系。

建立好的关系，在“图形样式”中进行查看会得到非常直观的结果，如图 8-10 所示。

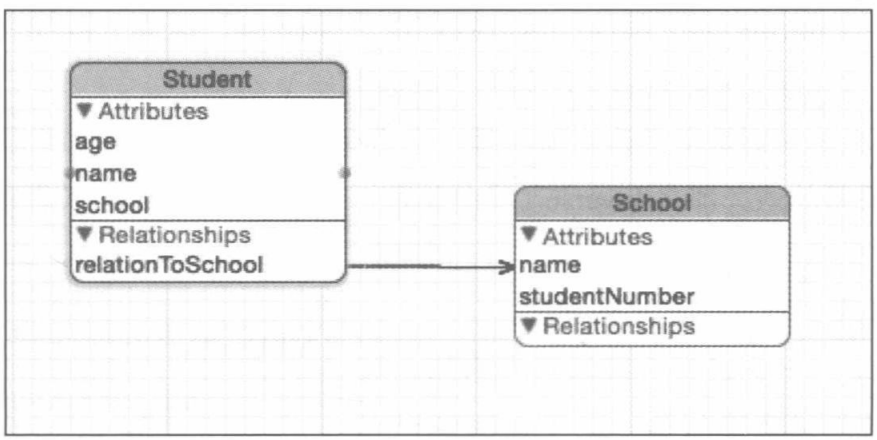


图 8-10　已建立好的关系

8.2.2.4　添加提取属性

提取属性（Fetched Propertes）在前面的术语中并没有介绍，是因为提取属性一般用于多个持久化存储之间建立关系，但是由于使用多个持久化存储不常见，并且十分高级的，因此这个提取属性一般情况下是不使用的。

要添加提取属性，可以单击选项窗格中的添加提取属性按钮，也可以通过选择 Editor → Add Fetched Property 来实现同样的操作，还可以在编辑器窗格中找到提取属性列表，然后单击“+”号按钮创建一个新的提取属性。

添加完提取属性后，就可以进行断言（Predicate）设置。所谓断言就是搜索关键字或限定条件。关于断言的具体设置项，请参阅苹果官方文档“Predicate Programming Guide”。

8.2.2.5　添加提取请求

要添加提取请求，可以单击选项窗格中的添加提取请求按钮，也可以通过选择 Editor → Add Fetch Request 来实现同样的操作。

在这里，我们添加一个名为“FetchAgeGreaterThan18”的提取请求。

提取请求的编辑器窗格如图 8-11 所示。从当中可以选择该提取请求需要提取哪一个实体

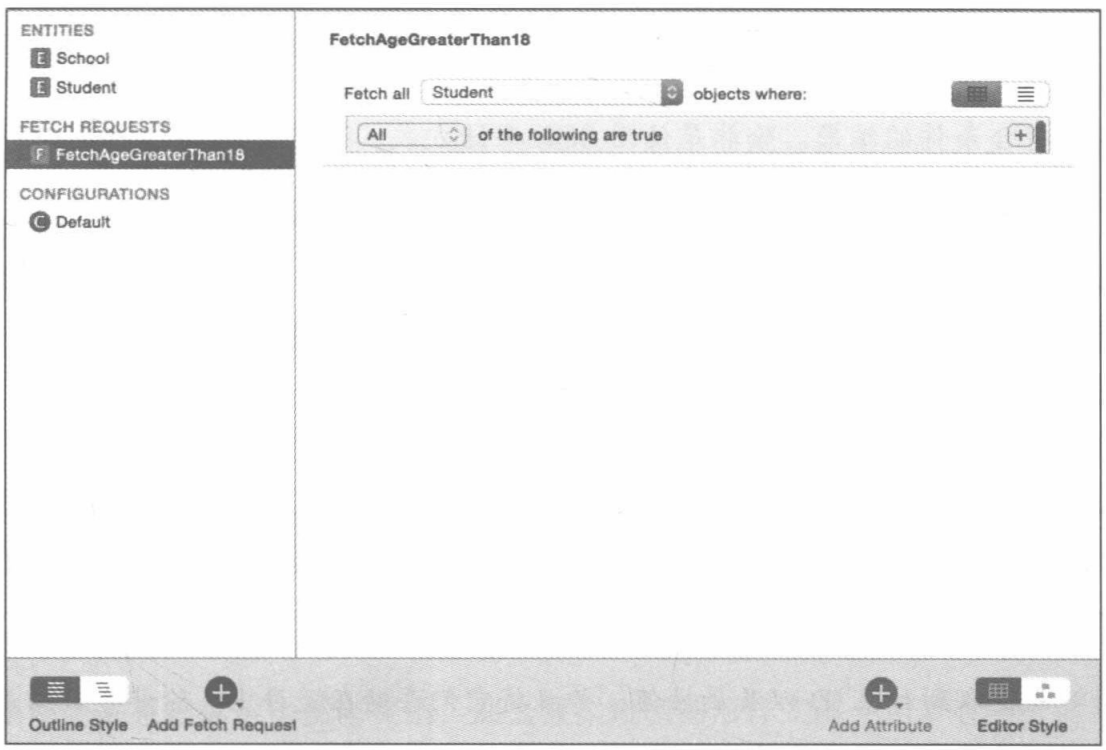


图 8-11　提取请求编辑器界面

中的数据，同时在 Expression（表达式）栏当中填写相应的断言，以完成提取请求的设置。你也可以通过单击“+”按钮来添加新的断言描述，通过编辑器右上角的视图设置可以切换断言描述的显示方式，是分行显示还是整体显示。

在这里，我们选择提取 Student 实体中的数据，然后添加以下设置：

```
(All) of the following are true
    (age) (is greater than or equal to) (18)
```

这样，我们的提取请求就会提取 Student 实体中所有满足“年龄大于等于 18 岁”的对象了。

在右侧的数据模型检查器中，我们可以看到，如图 8-12 所示，提取请求主要有以下几种设置：

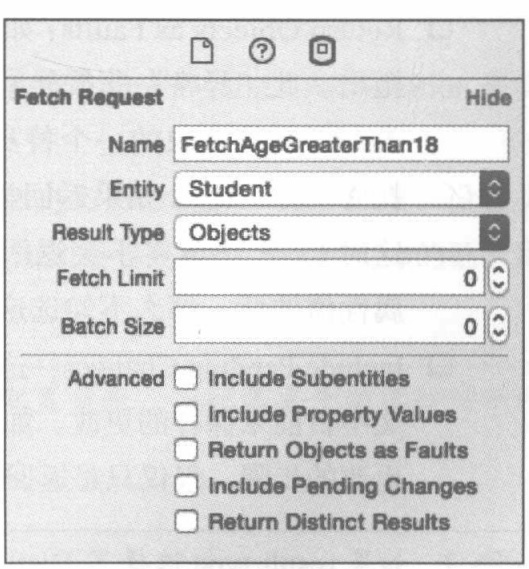


图 8-12　提取请求检查器

- Name：顾名思义，用来设置提取请求的名称。
- Entity：用来设置该提取请求所要提取的实体对象。
- Result Type：设置提取请求最后所返回的结果类型，Objects 是表示返回查询对象，Object IDs 则是返回查询对象的唯一 ID 号，Dictionaris 则是返回一个字典，以 [ID: 查询对象] 的形式存在，方便利用。
- Fetch Limit：设定最大查询对象数目。

如果你设置了 Fetch Limit，那么会极大地提高查询效率，但是并不能保证能完全显示所有符合条件的结果。除非是使用 SQL 存储方式，否则设定了 Fetch Limit 的提取请求实际上则是执行无限制的提取，然后再丢掉多余的值。

- Batch Size：设定批大小，就是每次执行提取请求，处理多少条数据。默认值是 0，也就是意味着无限。如果你设定了其他值，那么查询对象将会被划分为批，以你设定的值为一批。
- Include Subentities：设定提取请求的实体是否包括父级实体。
- Include Property Values：设定提取请求运行时是否能够从持久化存储中获取属性数据。如果将该值设为 NO，那么可以用来减少存储开销，因为避免创建了表示属性值的对象。

一定是你确认不需要读取属性值，否则一般情况下都勾选该选项。因为这样 Core Data 才可以提取到对象 ID 以及属性值，并且将它们存储在缓存中。除非查询结果被销毁，否则这些信息将一直保存在缓存中。如果该选项没有被勾选，那么 Core Data 仅仅只会提取对象的 ID 信息来作为匹配记录，不会向缓存中写入数据。如果这个查询请求只需返回对象 ID，那么 Core Data 仍然会返回托管对象。

- Return Objects as Faults：如果勾选了这个选项，那么查询结果将作为“中断”(faults) 抛出。此“异常”非彼异常（error），并不会导致程序崩溃和退出，准确来说，中断类似于托管对象的一个替身，其中包含了托管对象的部分数据，但是并不是完整的托管对象。这个如果返回类型选择了 Object IDs，那么这个选项就不要勾选，因为这时 Core Data 将不会返回任何对象。同时，如果你知道需要从返回的对象中访问属性值的话，那么不勾选这个选项可以获得更好的体验。
- Include Pending Changes：这个选项决定当提取请求运行时，是否匹配在托管对象上下文中目前未保存的更改。如果不勾选这个选项的话，那么提取请求会跳过检查未保存更改的步骤，仅仅只是返回匹配断言的对象。

如果 result type 选择了 Dictionaries，那么是不能够使用这个选项的，合计结果的计算也是如此（比如说 max 和 min）。对于字典来说，从提取请求中返回的数组反映了当

前持久化存储中的状态，并且将不会考虑任何在上下文中未定的更改、插入以及删除操作。

- Return Distinct Results：这个选项决定提取请求是否只返回相异值。

8.2.2.6　数据模型图形样式

数据模型图形样式中的矩形表示实体，实体中的方格又依次划分出属性、关系和提取属性。要注意的是，提取请求是不会出现在数据模型图形样式之中的，图形样式如图 8-10 所示。

如果取消对实体的选中，那么这个实体所对应的矩形标题将变成粉红色，而如果选中这个实体，那么这个矩形标题就会变成浅蓝色，表明其被选中。选中之后，可以随意调整窗口的大小。

实体之间的连线描述了关系和继承状态。箭头的形状则表明了关系的种类，参见表 8-1。

表 8-1　关系种类和箭头的对应关系

箭头形状	关　系
单箭头	一对一关系，箭头指向的是目标实体
双箭头	一对多关系，箭头指向的是目标实体
空心箭头	继承关系，箭头指向的是父级实体

如果两个关系被标记为反向关系，那么数据模型图形样式就会用带有两个箭头的连线来表示这种关系，这让数据模型变得更加好懂易读。

继承和单向关系的连线都是从起始实体开始，并指向目标实体。对于反向关系来说，Xcode 会精确地在两个互补的关系属性之间绘制连线。

8.2.2.7　添加配置

可以单击选项窗格中的添加配置按钮，也可以通过选择 Editor → Add Configuration 来实现同样的操作。

在配置的编辑器界面中，可以对配置中所拥有的实体进行修改名称、设置抽象、设置类等操作，还可以单击“-”按钮将实体从配置当中移除。而如果要向配置中添加实体的话，那么将相应的实体拖到配置上方，即可完成添加操作。

默认情况下，所有的实体都会被添加到 Default 配置当中，因此需要手动进行实体的管理。

8.2.2.8　其他功能

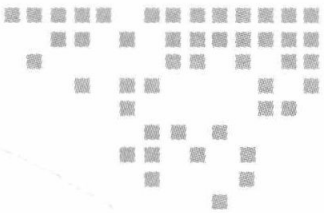
Core Data 数据建模编辑器还拥有三个功能，它们分别是：

- 模型导入（Import）
- 版本迁移（Add Model Version）
- 创建 NSManagedObject 子类（Create NSManagedObject Subclass）

这三个功能都可以通过选择 Editor 来访问到，关于这三种功能的使用方式，本书在此就不加以介绍了。

每天夕阳西至，皓月初升，少年良辰总会寻觅一个僻静之处，将白天所学、所练之心法、招式和真气一一回味，平心静气打坐，锤炼丹田。

道无道兮亘古存，空不空兮自得根。无象形兮潜造化，有乾坤兮出群伦。汲天地兮灵气生，觅日月兮隐元神。桑田话兮望沧海，无事为兮言天真。



第 9 章 Chapter 9

前人栽树——共享代码

蔽芾兮甘棠，遐声兮后旁。殷鉴兮不远，遗音兮绕梁。

所谓前人栽树，后人乘凉，后起之秀倘若能够拿到前人的武功秘籍，从中习得适合自身的招数，直至终有一日成为武林高手，甚至能开创出属于自己的独门武功，推陈出新，实乃善事。

少年良辰之梦，也正是从前人的成果上启程，在帮派和白衣老人这等前辈的指点下一步步通往武林的金顶。在编程这片江湖中，早有前人创造出的代码可供我们使用，我们只需从中抽取出需要的部分，再将其进行适当的编排便可最终建起属于自己的一片天地。

9.1 共享代码机制

在 Xcode 当中，常常提及三个重要的词汇：库（Library）、框架（Framework）和包（Bundle），这三者都是用来在多个应用当中共享代码的机制。它们预先进行打包，使用它们的开发者无需关注其内部的实现机制，只需使用暴露在外面的接口即可工作。正由于它们的存在，使得现在各种第三方框架百花齐放，开发工作越来越简单，大家只需要选择合适的第三方框架即可简单地完成想要的功能。下面分别介绍这三个机制。

9.1.1 库

库是共享代码的最基本方式，在 Xcode 当中，则存在两种库，分别是静态库（.a 文件）和动态库（.dylib 文件）。

静态库是在应用编译的过程中由链接程序链接到对象当中，如果在多个对象当中共享一个静态库，那么这个静态库都会被包含在每个对象当中。静态库构建简单，使用简单，但是会使应用的体积变得很大，尤其是应用包含多个对象的时候。

动态库是在应用运行时按需加载的，动态库的代码对于多个对象来说是共享的，因此动态库比静态库来说，可以使应用运行速度更快、体积更小。

关于静态库和动态库的原理，大家可以参考苹果的官方文档《Dynamic Library Programming Topics》的 Overview of Dynamic Libraries 一节。

目前 iOS 只支持静态库，而动态库是不受支持的，虽然从技术上来说也可以应用动态库，但是很有可能无法上架。

9.1.2 框架

框架相当于一个文件夹，其中包含了诸如库、界面构造器、图片、本地化字符串、媒体、头文件之类的资源，简而言之就是一个装有各种各样可重用资源的“包”。相比库来说，框架不但能够共享代码，还能够共享各种各样的资源文件。

开发者平时在开发的过程中，就已经在使用了各种各样的“框架”，比如说 iOS 开发者经常使用“UIKit”框架，OS X 开发者经常使用“Foundation”框架，这些框架为开发者们提供了各种各样的功能。

和库相仿，框架并没有严格意义上的静态和动态的区别，开发者可以根据实际情况选择是按需加载还是始终加载。

对于开发者来说，由于 Xcode 6 提供了“动态框架”的制作功能，因此，我们可以粗浅地认为“动态框架”只支持 OS X 系统以及 iOS 8 以上系统。

以上所述的库和框架，就是 iOS 平台上最常用的两种共享代码的机制。在之后的叙述当中，如果没有特别说明，我们所说的“库”，就都包含了“库”和“框架”。那么我们为什么要使用库呢？

静态库是最常见的库，因为其制作简单。当我们想要把一些常用的代码分享给别人使用，但是又不想让别人看到源代码的话，那么静态库是最好的选择。并且，当我们对这部分代码进行更改的时候，那么可以很轻松地维护库。就不存在大量复用代码堆积的情况出现。

那么动态库呢？

我们现在都知道，目前很多大平台级别的 APP，都提供了各式各样的插件，以便于扩充功能。但是如果在打包上传的时候把所有的功能都提供的话，那么软件的大小将会不可想象的。现在常用的解决方法是靠 HTML5 来解决，但是运行速度、反馈体验等都没法和原生应

用相比的。

不过，自 iOS 8 推出之后，我们完全可以把这些功能都编译成一个动态库文件，当用户想使用这个功能的时候从网络上下载，然后再加载这个动态库，就可以轻松地实现应用的插件功能了，而且还不会导致软件包大小变得很大。

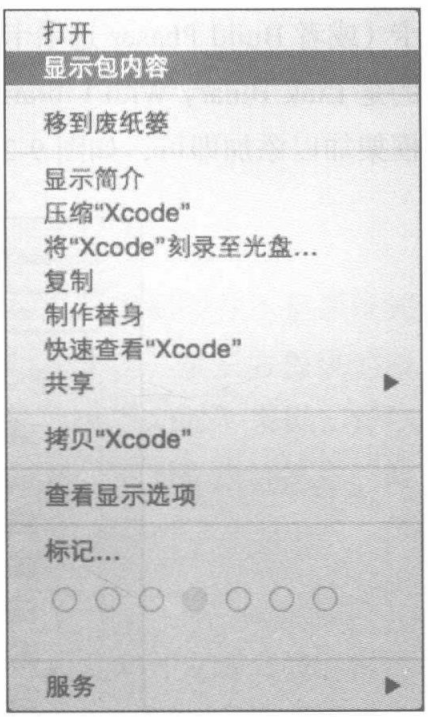


图 9-1　显示包内容

9.1.3　包

包是一种特殊的文件夹，它一般情况下是被视作单独文件的。在 OS X 上，应用程序、框架、插件等等都属于包的一种，我们可以右键选择这些包，然后选择“显示包内容”，来查看其中的具体信息，如图 9-1 所示。

对于 OS X 应用来说，“包”包含有可以在运行时加载和卸载的代码和资源，从而可以将应用加以模块化，或者作为插件为之提供扩展。就比如说屏幕保护程序，每个屏幕保护程序都是一个包。屏幕保护应用只需要负责加载这些包即可。

对于 iOS 应用来说，包只能够包含资源文件，不能够用它来运行代码。

提示　有一种包是例外，那就是单元测试包（Cocoa Touch Testing Bundle），这个包将可以用来批量对多个应用进行自动化测试。不过在 Xcode 当中，它是以“对象”的形式来展现的。

提示　对于 OS X 来说，还有一种奇特的共享代码的方式——XPC Service。XPC 是 OS X 中不同应用之间信息交流的一种机制，自 0S X 10.7 之后提供。通过创建 XPC，开发者可以自定义应用之间的通信机制。关于 XPC 的具体内容，在此就不再加以赘述了。

9.2　使用现有框架

在 Xcode 中使用现有框架是非常简单的一件事，特别是在使用系统框架的情况下。如果使用的是第三方框架，那么根据框架的不同，添加框架的方法也不同，在本节的最后，我们还会为大家介绍一种方便管理第三方框架的工具——CocoaPod。

9.2.1 使用系统框架

使用系统框架的方法十分简单，定位到要使用系统框架的对象，然后选择其 General 选项卡（或者 Build Phases 选项卡），展开 Linked Frameworks and Libraries 栏目（Build Phases 对应的是 Link Binary With Libraries），单击“+”按钮。然后就可以在弹出的表单当中选择相应的框架加以添加即可，如图 9-2 所示。

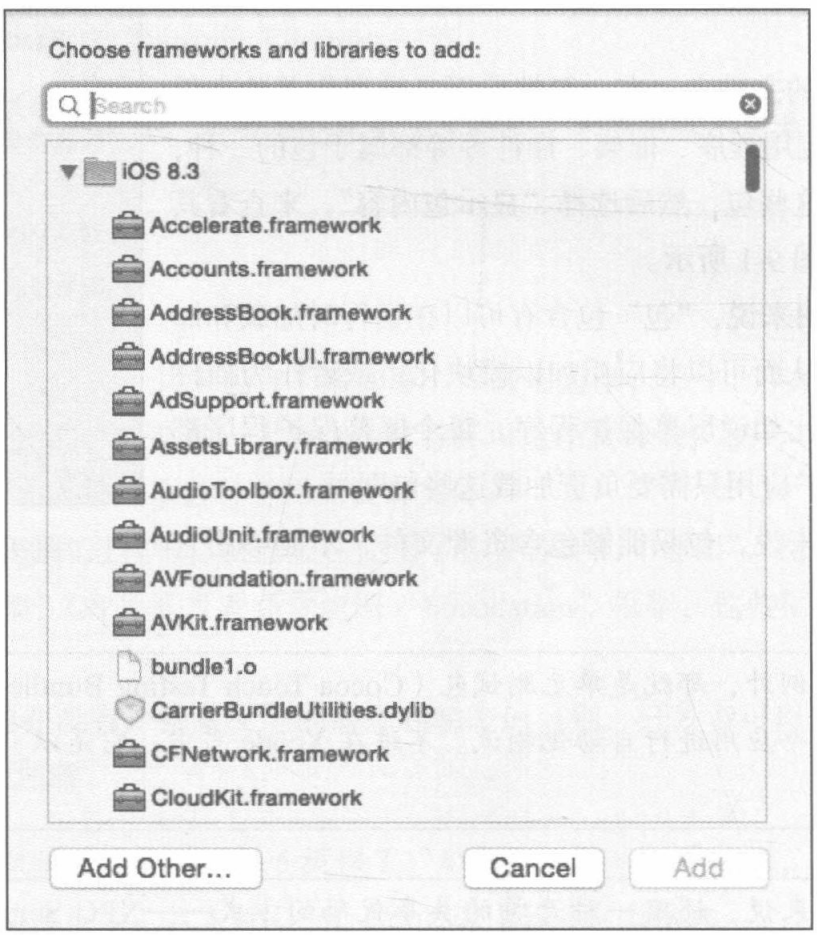


图 9-2 添加系统框架

对于 Swift 语言来说，似乎其拥有默认的部分系统框架（Framework）访问权限，因此对于某些常用的框架来说，只需要在工程当中 import 相应的框架名称即可，无需添加。

对于框架来说，通过更改框架的状态，可以更改它们的编译设置。在 General 选项卡或 Build Phases 选项卡的“Linked Frameworks and Libraries”栏目中，可以看到目前框架的状态（Status）是必须的（Required），也就是说，应用必须要包含有这个框架才能够运行，如果应用当中未将这个框架编译进去，那么这个应用无法通过编译。状态还可以更改为按需的（Optional），应用在编译的时候，如果需要这个框架才会编译这个框架。

添加完框架之后，不要忘记在头文件中使用 #import 将其框架对应的头文件导入进去。

删除框架的方法也很简单，将其从“ Linked Frameworks and Libraries”中移除，并删除其在项目中的引用即可。

对于系统框架来说，每次系统 SDK 的更新都会自动进行更新，因此无需担心系统框架的版本问题。

9.2.2　使用第三方框架

使用第三方框架需要做的操作相比系统框架而言更繁复，因为第三方框架并不是系统的一部分，因此它必须要“绑定”在应用当中，这样才能够使用。目前来说，第三方提供的框架更多的是集中在“静态库”、“动态库”和“框架”上，对于“包”和“ XPC”来说，并不常见，因此本书不对其进行介绍。有兴趣的同学可以查阅相关的资料。下面介绍静态库、动态库和框架。

9.2.2.1　使用第三方静态库

对于很多第三方 SDK 来说，它们往往采用静态库（.a 文件）的形式向开发者分发，因为静态库的制作方式简单，并且已经有很长的使用年头了，静态库在 iOS 的第三方 SDK 中较为常见。

将静态库文件直接拖到项目中的合适位置，选择“Copy Files If Needed”将静态库导入到项目当中。对于 Xcode 6 来说，它会自动识别到这个静态库文件，并自动将其添加到对象设置中的“ Linked Frameworks and Libraries”当中，如果 Xcode 没有自动添加，那么需要手动添加这个静态库文件。

接着，设置这个静态库的链接路径，在项目设置中，定位到 Build Settings → Search Paths → Library Search Paths 中添加静态库目录，指向静态库所在的位置。注意，如果移动了静态库文件，那么这里的链接路径也需要相应的改变。

当然，由于第三方 SDK 的实现机制不一样，因此开发者可能还需要做额外的工作才能够使用成功第三方静态库。不过一般情况下，第三方 SDK 都会对如何配置工程加以详细的说明，告知开发者如何使用。

9.2.2.2　使用第三方动态库

对于 OS X 来说，由于苹果官方推荐使用动态库来编译 OS X 应用，因此网上也涌现了很多优秀的第三方动态库。

和使用第三方静态库方法类似，将动态库文件添加到项目当中，并在“ Linked Frameworks and Libraries”中添加这个动态库文件。

在项目设置中，定位到 Build Settings → Search Paths → Library Search Paths 中添加动态库

目录，然后再 User Header Search Paths 中添加动态库的头文件所在目录。

注意 如果引用了第三方的动态库，那么必须要将这个第三方动态库和应用进行绑定，否则的话，如果用户没有这个动态库文件，那么这个程序就很可能无法运行。绑定第三方动态库的方法众说纷纭，并没有太好的方法，限于篇幅显示，本书不对此进行介绍。现在 OS X 上所使用的第三方框架大多都已经替换成框架了，因此不推荐大家使用第三方动态库文件。

如果你尝试在 iOS 项目中编译打包含有动态库的项目，那么很有可能无法通过编译。如果通过了编译，也很可能无法提交到 App Store 当中，因此在 iOS 上使用动态库需要深思熟虑（除非你没考虑过上架问题）。

9.2.2.3 使用第三方框架

添加第三方框架的方法很简单，将框架（.framework 文件）导入项目当中，Xcode 会自动将框架导入到合适的位置。如果是静态库框架，那么框架会被放置在“Linked Frameworks and Libraries”栏目中，如果是动态库框架，那么框架会被放置在“Embedded Binaries”栏目当中。如果 Xcode 没有成功添加框架，那么单击相应栏目的“+”按钮，选择“Add Other”添加第三方框架。

动态库框架是 Xcode 6 提供给 iOS 的新功能，动态库框架一般情况下是在同一应用的多个对象之间分享代码和资源而使用的。

9.2.3 使用 CocoaPods 管理框架

如果我们只是添加了少量的第三方框架，那么手动管理起来也不算什么难事，但是如果我们使用了大量的第三方框架的时候，管理起来就会变成十分繁杂的过程，此外，如果要获取第三方框架的更新，手动管理无异于重新添加一遍第三方框架。因此，我们需要使用工具来帮助管理这些第三方框架，CocoaPods 让管理框架的操作变得简单易用。

CocoaPods 是一款十分优秀的第三方框架管理工具，使用它我们可以下载框架的源代码并且导入到工程当中，此外，CocoaPods 对框架的更新管理也是得心应手，我们也不必考虑纷繁复杂的依赖关系了。下面就概述一下这个工具。

9.2.3.1 安装 CocoaPods

CocoaPods 是使用 Ruby 实现的，一般情况下 OS X 已经配置好了 Ruby 运行环境，因此，我们只需要打开终端，输入以下语句：

```
sudo gem install cocoapods
```

然后输入用户密码即可。

9.2.3.2　常见问题

安装 CocoaPods 可能会遇到许许多多的问题，下面讲几个常见问题。

1. 安装超时

比如说我们可能会碰到以下提示：

```
ERROR: Could not find a valid gem 'cocoapods' (>= 0), here is why: Unable to
download data from https://rubygems.org/ - Errno::ETIMEDOUT: Operation timed out -
connect(2) (https://rubygems.org/latest_specs.4.8.gz)
```

出现这个情况，一般就是安装超时。首先先检查一下当前网络的通畅情况，因为这是要连接到 rubygems.org 去进行下载，本来访问外网就慢，如果网速还缓慢的话，往往就会导致连接超时的情况出现。

可有些时候网络通畅的情况下，可能还是无法下载。这个时候就需要用到某宝的镜像服务器了，这个服务器每 15 分钟同步一次，因此我们完全可以用其来代替：

```
gem source -remove https://rubygems.org/
sudo gem source -a http://ruby.taobao.org/
```

为了验证我们当前使用的镜像服务器是淘宝的，输入以下命令：

```
gem sources -l
```

只有在终端中出现下面的文字才表明上面的命令成功了：

```
*** CURRENT SOURCES ***

http://ruby.taobao.org/
```

然后再次输入安装命令，稍事等待后，Cocoapods 就自动下载并安装完成了。

2. 权限问题

上节的命令有时不一定能够有效执行，因为很可能会弹出以下问题：

```
ERROR:  While executing gem ... (Errno::EACCES)
    Permission denied - /Users/SemperIdem/.gemrc
```

这个消息意味着，出现了权限问题，当前的系统配置不允许非管理员用户对其进行操作，解决方案很简单，在命令前面加上 sudo 即可。

3. Gem 版本问题

由于 Cocoapods 是以 Gem 包的形式安装的，如果 Gem 的版本过低，同样也会导致安装失败，可能会弹出以下问题：

```
ERROR:  While executing gem ... (Gem::DependencyError)
    Unable to resolve dependencies: cocoapods requires cocoapods-core (= 0.33.1),
```

```
    claide (~> 0.6.1), cocoapods-downloader (~> 0.6.1), cocoapods-plugins
    (~> 0.2.0), cocoapods-try (~> 0.3.0), cocoapods-trunk (~> 0.1.1), nap (~>
    0.7)
```

这个时候，只需要运行以下命令，升级 Gem 即可：

```
sudo gem update --system
```

9.2.3.3 向项目中添加第三方框架

安装后就是如何使用的问题了。很赞的是，使用 CocoaPods 同样也很简单，只需要几行命令即可搞定。

那么我们现在就使用 CocoaPods 往项目中添加第三方框架吧！我们在这里准备导入一个第三方数据库框架：Realm。它的地址为：https://github.com/realm/realm-cocoa。

为了确定 Realm 是否支持 CocoaPods，我们可以使用 CocoaPods 的搜索功能进行验证，在终端中键入以下命令：

```
pod search RealmSwift
```

我们也可以搜索更适合 Objective-C 的 Realm 库：Realm。

过上一会儿（这取决于你的网速和被墙的程度），我们就可以在终端中看见关于 RealmSwift 库的一些信息：

```
-> RealmSwift (0.92.3)
   Realm is a modern data framework & database for iOS & OS X.
   pod 'RealmSwift', '~> 0.92.3'
   - Homepage: https://realm.io
   - Source:   https://github.com/realm/realm-cocoa.git
   - Versions: 0.92.3, 0.92.2, 0.92.1 [master repo]
```

这就说明，Realm 是支持 CocoaPods 的，所以我们可以使用 CocoaPods 将这个框架导入到项目当中。

首先，我们需要在项目中添加对 CocoaPods 的支持。打开 CrazyBounce-Swift 项目，然后在里面添加一个文件，用来告诉 CocoaPods 这个项目想要哪些第三方框架。这个文件叫做“Podfile”（没有后缀），这也是我们在浏览 Github 的时候经常看见的文件之一。每个项目仅能拥有一个 Podfile 文件。

最常用的方法就是在终端中使用 vim 创建 Podfile 了，当然，对于某些不会用 vim 的人来说，我们使用 Mac 自带的“文本编辑”应用也能够达成这个效果。打开“文本编辑”应用，然后在其中输入以下语句：

```
platform :iOS, '7.0'  #所有第三方库所支持的iOS最低版本
use_frameworks!
```

```
pod 'RealmSwift', '~> 0' #版本号
```

关于版本号，有它独特的制定规则：

- '>1.0'：任何高于 1.0 的版本
- '>=1.0'：任何高于或等于 1.0 的版本
- '<1.0'：任何低于 1.0 的版本
- '<=1.0'：任何低于或等于 0.1 的版本
- '~>0.1'：任何高于或等于 0.1 的版本，但是不包含高于 1.0 的版本
- '~> 0'：任何版本，相等于不指定版本，默认采用最新的版本号

然后选择菜单栏的“格式→制作纯文本”选项（或者直接使用快捷键 Shift+Command+T），然后在弹出的对话框中选择“是”，保存文件，将其命名为“Podfile”（注意，一定要是这个名称），然后选择存储路径为项目的根目录，随后取消“如果没有提供扩展名，则使用 .txt”选项，保存即可。

注意　上面这段命令不是通用的，添加第三方库支持的 Pod 设置要查看具体的第三方库的说明，每一个第三方库的命令都不尽相同。并且，Podfile 文件一定要和工程文件 .xcodeproj 在同一个目录下。

此外，还有一个更为简单的方法，我们进入到项目文件目录当中，然后使用以下命令，就可以自动生成一个模板文件：

```
pod init
```

随后开启终端，进入（cd）到项目文件目录当中，然后执行以下语句：

```
pod install
```

运行结束之后，会弹出以下信息：

```
Analyzing dependencies
Downloading dependencies
Installing Realm (0.92.3)
Installing RealmSwift (0.92.3)
Generating Pods project
Integrating client project
[!] Please close any current Xcode sessions and use `CrazyBounce-Swift.xcworkspace`
   for this project from now on.
```

这些过程大致做了以下事情：

- 分析依赖：这个步骤会分析 Podfile，查看不同类库之间的依赖情况。如果有多个类库依赖于同一个类库，但是依赖于不同的版本，那么 CocoaPods 会自动设置一个兼容的版本。

- 下载依赖：根据分析依赖的结果，下载指定版本的类库到本地项目中。
- 生成 Pods 项目：创建一个 Pods 项目用来专门编译和管理第三方类库，CocosPods 会将所需的框架、库等内容添加到项目中，并且进行相应的配置。
- 整合 Pods 项目：将 Pods 和项目整合到一个工作区当中，并且设置库文件链接。

注意最后一句话，意思是：以后打开项目就得用 CrazyBounce-Swift.xcworkspace 打开，而不是之前的 .xcodeproj 文件。

这是为什么呢？我们之前也介绍过了，工作区的所有项目都可以访问同一工作区其他项目的资源和代码，包括其编译信息。因此，我们只需要管理好我们的项目即可，至于第三方框架的管理，交给 CocoaPods 去做就好啦！

9.2.3.4 运行 CocoaPods 管理的项目

我们有些时候经常能碰到这种情况，好不容易在 Github 上找到了一份非常棒的代码，然后立马下载了下来，结果发现根本无法编译，出现了各种各样的错误。其实，其根本原因是缺失了各种第三方框架，如果这个时候项目包含有 Podfile 的话，那么我们就可以用 CocoaPods 来下载所需要的类库了。

在这里，我们选择一个简单的图表库——PNChart，这个是 kevinzhow（微博：周楷雯 Kevin）的作品，这个作品是一个非常炫酷的图表库，同样，它也使用了 Podfile。

前往 Github(https://github.com/kevinzhow/PNChart) 下载该项目，然后在终端中进入该项目中，输入以下命令：

```
pod update
```

这个命令是根据本地的 Podfile 来更新本地的类库，如果本地类库不存在，那么这个命令也会下载本地缺失的类库。

这样，项目就可以成功运行了！

9.3 创建框架

如果开发者经常在多个项目中重用某一部分的代码，那么创建一个框架来存储这部分代码以便加以复用是一个很好的选择。因为如果这一部分代码出现了错误需要修改或者需要进行功能性的更新时，同时修改多个项目中的代码显得十分费时，而如果放到框架当中，那就简单了。

此外，如果开发者想要共享自己的某段代码，并且又不喜欢别人查看其中的源代码的话，那么将其做成框架再共享出去也是一个明智的选择。

9.3.1　创建静态库

选择 Xcode 菜单栏上的 File → New → Target 或者 Project（如果选择 Target 的话则是创建一个在当前项目中复用的框架，选择 Project 的话则是创建一个可以全局复用的框架），打开模板选择对话框，在 iOS 的 Framework && Library 栏目中选择“Cocoa Touch Static Library”（OS X 则是选择 Library，并且 Type 选择 Static），如图 9-3 所示。输入工程名称以及相应信息后，即可完成库文件的创建。这里，我们创建一个名为 Water Wave 的全新静态库项目。

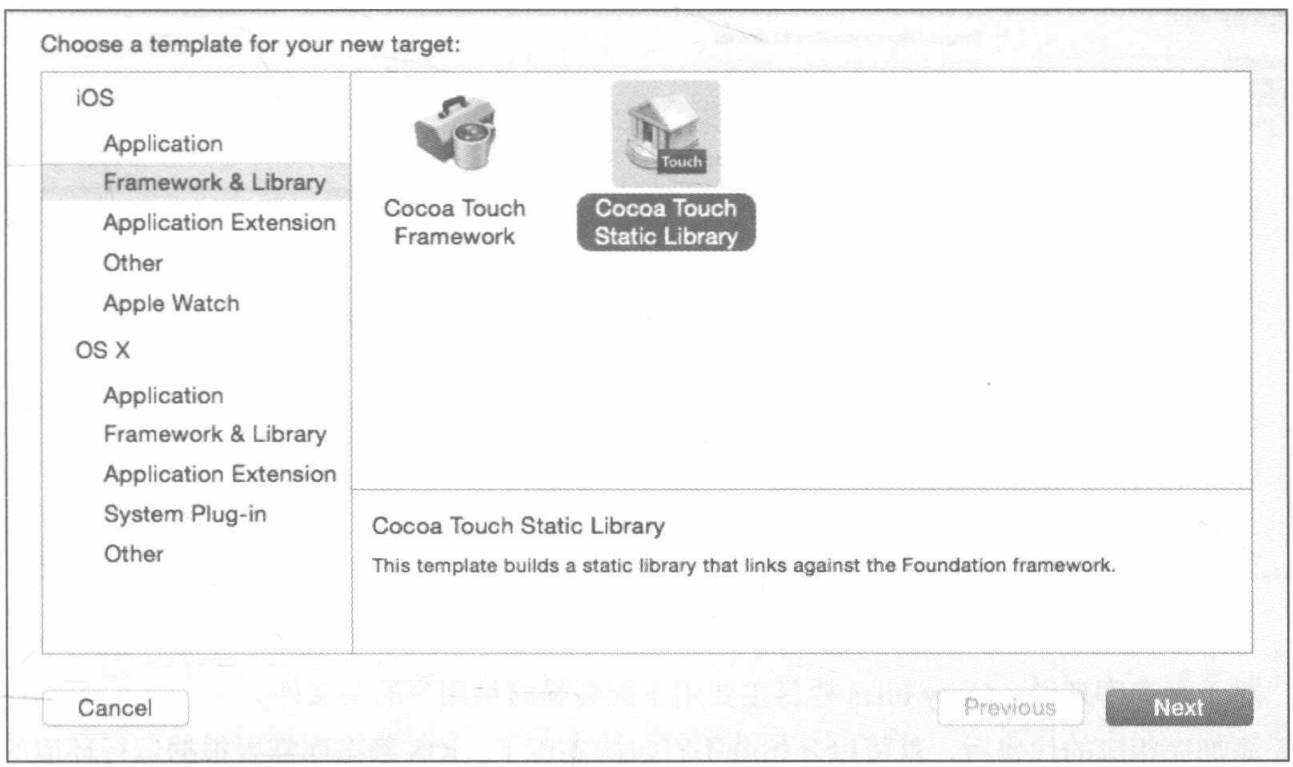


图 9-3　创建静态库

注意　静态库文件支持 Objective-C 语言，虽然用 Swift 语言也能创建出静态库文件，但是无疑是要比 Objective-C 麻烦得多的。

创建之后我们可以看到如图 9-4 所示的目录结构。默认的静态库模板中包含有一个头文件和一个 .m 文件，我们可以将需要复用的代码放置在其中，如果必要的话，可以创建多个文件。注意静态库文件只能够包含代码文件。

注意　框架名 .h 是默认的静态库对外接口，外部程序通过导入这个头文件来完成对静态库的使用。

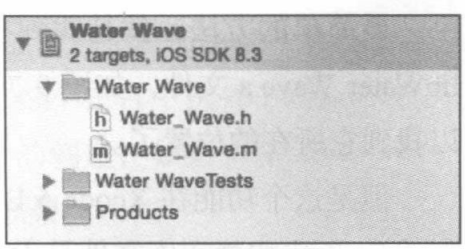


图 9-4　静态库模板目录结构

我们可以向库中添加一些常用的代码，比如，我们将 CrazyBounce 中的 Wave 视图进行一下封装。将 Crazy-Bounce-

OC 中的 waterView 的全部代码迁移到 Water_View 当中来。

然后打开项目设置，选中刚刚创建的静态库，这时就可以看出其和一般应用的区别来了。首先静态库拥有一个白色的小房子图标，并且它没有 General 和 Capabilities 以及 Info 选项卡，并且 Build Phases 选项卡中出现了一个“ Copy Files”栏目，如图 9-5 所示。（OS X 静态库的显示内容和 iOS 静态库有些区别，请参考下一节的内容。）

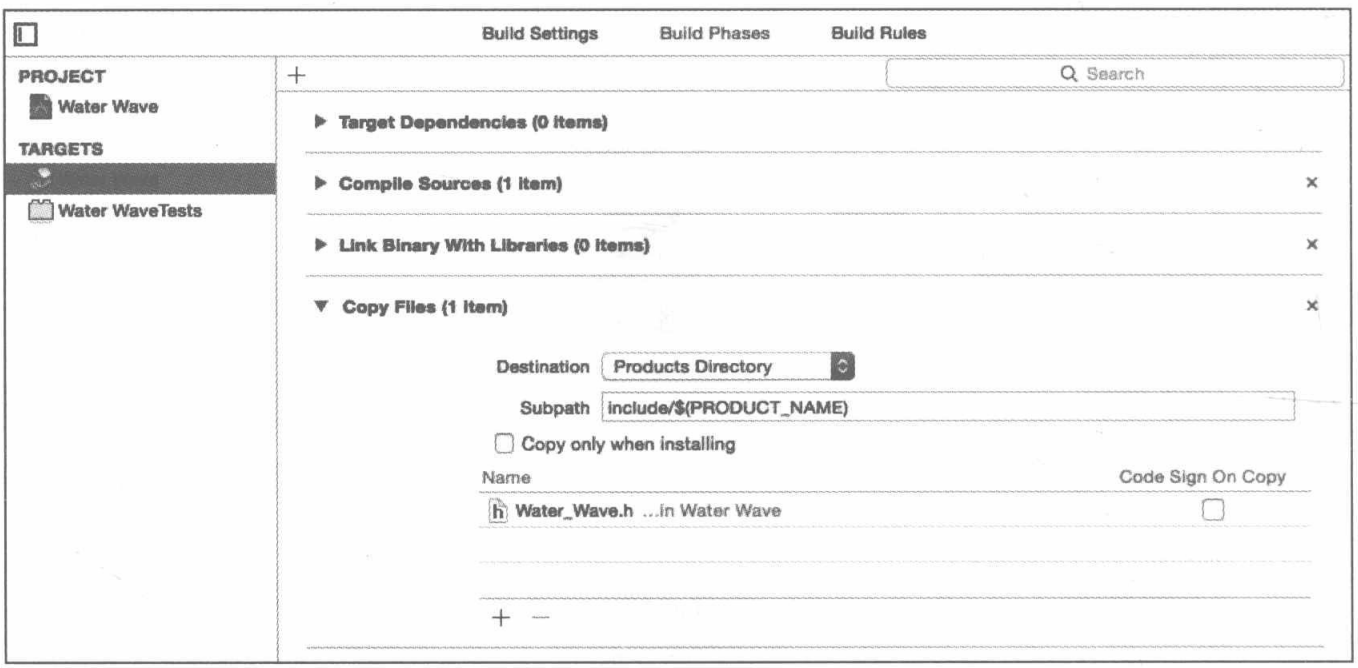


图 9-5　静态库的 Build Phases 界面

对于静态库来说，Copy Files 栏目主要用于配置暴露给用户的头文件。

添加完相应的代码后，就可以开始着手打包静态库了。iOS 静态库需要根据运行环境的不同制作不同的静态库，一个是真机运行的静态库，另一个是模拟器上运行的静态库。这两者不能够混用，并且制作过程有区别。（OS X 静态库没有这种区别。）

确保 Xcode 工具栏上的编译方案是静态库的编译方案，如图 9-6 所示。执行 Build 命令（Product → Build），这样 .a 文件就自动生成了。

嗯？你可能会问了，编译成功后什么都没有发生，这就好了？是的，这就好了，就是这么简单。那么我们生成的静态库文件呢，这个小家伙目前在什么地方呢？有两种方法可以找到。

最简单的方法，就是找到项目导航器当中的 Products 分组，展开就可以看到我们生成的 libWater Wave.a 文件，如图 9-7 所示。右键点击这个静态库文件，选择 Show in Finder，就可以找到它所在的位置了。

但是这个功能在 Xcode 5 以前都是十分好用的，但是自 Xcode 6 以后（或许跟笔者的配置有关），生成的静态库文件是无法被找到的，因此 Products 分组里面显示的静态库文件是红色

的，也就是没有找到。使用 Show in Finder 就没有任何效果。这该如何是好？

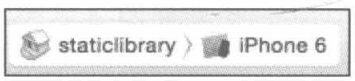


图 9-6　静态库的编译方案

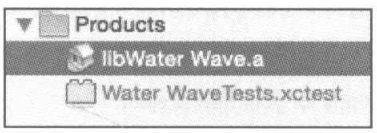


图 9-7　Products 中生成的静态库文件

这就需要第二种“百发百中”的方法了。选择菜单栏上的 Window → Projects，找到当前项目对应的“ Derived Data”，选择路径右边的小箭头，打开 Finder，如图 9-8 所示。然后找到文件夹当中的 Build → Products，就可以在“ Debug-iphonesimulator”文件夹中找到新创建的 .a 文件，并且还可以在 include 文件夹中看到对应静态库暴露的头文件，如图 9-9 所示。

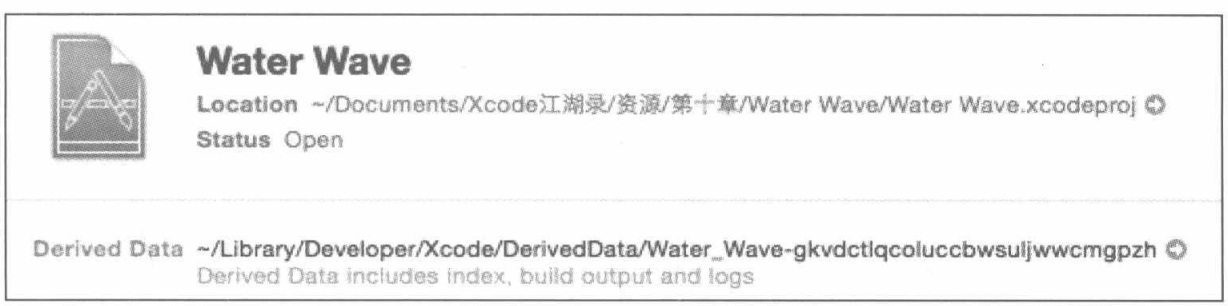


图 9-8　查找项目路径

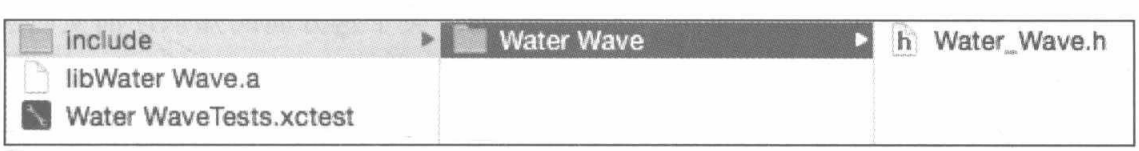


图 9-9　从项目路径中找到的静态库文件

对于真机运行的静态库，将设备切换为真机，然后执行同样的操作即可。新创建的 .a 文件出现在“Debug-iphoneos”文件夹当中。这样就完成了静态库的创建，添加这个静态库的方法和上一节介绍的一样，别忘了将头文件也要一并拷贝使用。


提示　可以使用 lipo 命令在终端中将这两个 .a 文件合并成同一个静态库文件。

9.3.2　创建动态库

动态库只能够在 OS X 平台上创建并使用。选择标题栏上的 File → New → Target 或者 Project，打开模板选择对话框，在 OS X 的 Framework && Library 栏目中选择 Library。输入工程名称以及相应信息后，Type 选择 Dynamic 即可完成库文件的创建，如图 9-10 所示。

在 OS X 平台上，我们可以选择所创建的库类型和其底层框架。

Framework 用于选择所创建的库是基于什么框架之上构建的，这样这个库就可以调用

相应框架的各种 API 函数。我们可以选择 Cocoa、STL(C++ Library) 和 None(Plain C/C++ Library)。选择不同的 Framework 所创建出来的库文件是不同的。

图 9-10 创建动态库

Type 可以用来选择这个库是动态的，还是静态的。关于这方面的知识，前文已经有所提及，在此不再加以赘述。

动态库的对象设置和静态库的相仿，不同之处是 Build Phases 中出现的不是 Copy Files，而是 Headers 栏目，如图 9-11 所示。

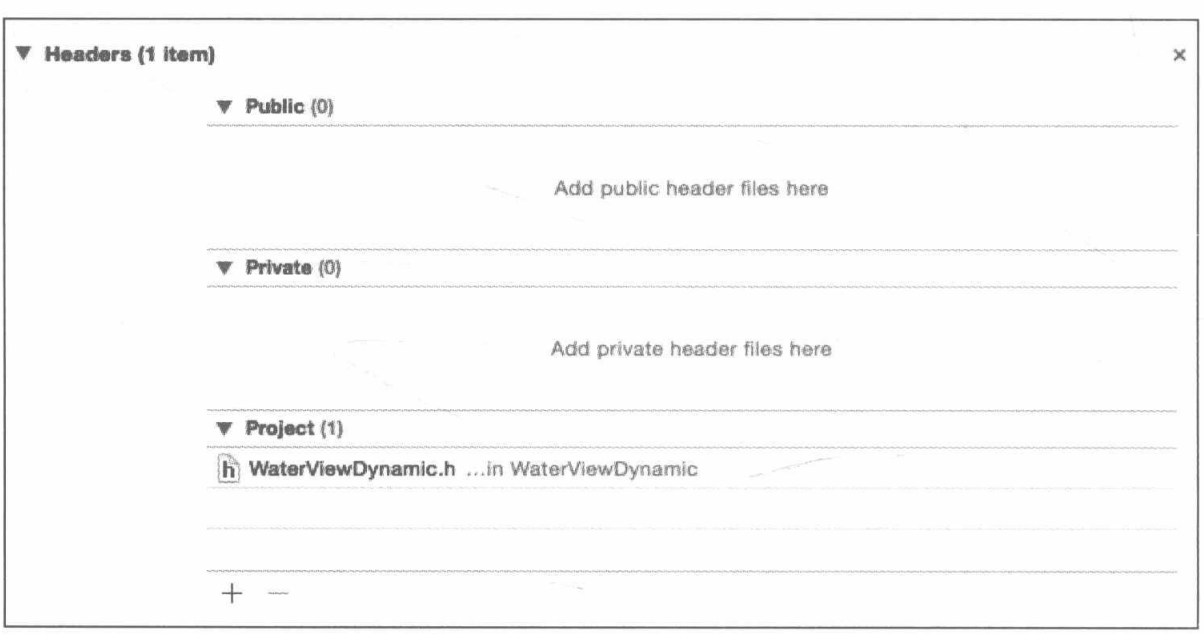


图 9-11 动态库的 Build Phases 界面

Headers 栏目有三个组，分别是 Public（公开）、Private（私有）和 Project（项目内），熟悉权限控制的同学不难看出它们的用处。我们需要将所创建的头文件放置到相应的栏目当中，从而决定哪些头文件需要暴露给用户，而哪些头文件不暴露给用户，默认的头文件权限是 Project。

和创建静态库的方法一样，选择对应的 scheme 之后，编译运行这个动态库即可获得对应的 dylib 文件。添加这个动态库的方法在 9.2.2 节已经介绍。

9.3.3　创建框架

框架是 Xcode 6 提供给 iOS 的新功能，OS X 平台上很早就存在了这项功能。由于是新功能，因此框架也支持最新的 Swift 语言。

选择标题栏上的 File → New → Target 或者 Project，打开模板选择对话框，在 iOS 的 Framework && Library 栏目中选择“Cocoa Touch Framework”（OS X 上对应的是“Cocoa Framework”）。输入工程名称以及相应信息后，即可完成库文件的创建。


提示　和库相比，框架对 Swift 的支持性更好。

创建完相应的代码后，打开框架的对象设置，可以看到相比正常的应用来说，框架仅仅只是少了 Capabilities 这一选项卡以及某些栏目而已。在“Build Phases”选项卡中，有一个 Headers 栏目，这个栏目和动态库当中的 header 是相同的作用。通过将需要公开的头文件放到 Public 栏目下，这样，生成的框架的头文件目录当中将只包含这些公开的头文件。

在 Build Setting → Lingking → Mach-O Type 选项中，可以选择生成的框架是包含“静态库”（Static Library）的框架还是包含“动态库”（Dynamic Library）的框架。默认情况下生成是包含动态库的框架，如果你打算让这个框架支持 iOS 7 或者 Xcode 5 的版本，那么就必须使用包含“静态库”的框架来生成最终目标。

和创建库的方法一样，选择对应的“scheme”之后，编译运行这个框架即可获得对应的“framework”文件。添加这个框架的方法在上一节已经有所介绍。


注意　和静态库相同，创建的框架也有适用于模拟器和适用于真机之分，因此需要使用不同的设备来对应生成不同的框架。同样，你也可以使用 lipo 命令将这两个框架合并。

淌过这一路泥泞，不知是多少江湖前人倾尽了无穷韶光华年才为后人铺起了一条弥漫着花香的捷径。同是浮生匆忙客，奈得杯深醇香绕指间。左手捧霞杯，右手举长明灯一盏，引君渡彼岸！后人风雨路三千，煮酒一壶，望却千年前，云翻腾，水缱绻，修得万纸经纶。

指落初叹，斑驳门扉后面的良辰，沉浸在前尘往事的清风中，醉意袭来，他半眯着双眼，朝那梦中白衣青衫叩门而唤的老人慢慢向前走去……

Chapter 10 第 10 章

武功是怎样练成的——编译系统

回首恍然兮入梦，剑舞落雁兮惊鸿。韬光匿影兮刃见，飘摇柳浪兮荷风。

老人剑舞落雁兮惊鸿。少侠藏剑一招，荷风柳浪。朦胧的云雾中，只能看见少年良辰紧随老人的动作，手提三尺青霄剑，以酒浇剑后，斩破了梦中那万重囚笼。廿载江湖美梦，只执剑划碧水一泓，天地皆入梦。少年青锋，不惧孤身，勇闯苍穹。

奈何云雾渐渐消散，良辰缓缓睁开睡梦惺忪的眼。梦里的他御剑出鞘，掀起万千波澜。他翻身坐起，长叹一声："练就这般武艺，如此我这般寥寥几日能够成就的呢？"

良辰匆匆来到门派大堂下求见大师。"少侠，要达到梦中向往的境界，不仅要身怀绝技，还必须要掌握各套武功招式的原理和编导这一套招式的脉络才行啊。"大师缓缓说道。

10.1 编译方案

编译方案（Scheme）是一系列编译设置的集合，它指定了构建项目的对象、对象所使用的编译配置，以及当应用启动时所使用的运行环境。

一旦创建了一个项目或者对象，或者打开一个既有的项目之后，Xcode 便会自动为项目中的每一个可编译的对象创建一个编译方案。默认的编译方案名称和你项目的名称相同（同时也是第一个创建的对象名称）。

项目中至少要存在一个编译方案，我们也可以按照需求包含任意数量的编译方案，但是一次只能够运行一个方案。

编译方案一般情况下定义了以下五种编译操作的构建设置：

- 运行应用。
- 针对某个对象来运行的单元测试。
- 应用性能特性的描绘。
- 执行代码静态分析。
- 归档应用。

编译方案可以和其他使用同一项目或者工作区的开发者共享，也可以导出以便用于其他项目当中，从而免去了重新设置编译方案的麻烦。在编译和运行应用之前，都必须要选择一个编译方案，从而指定 Xcode 按照何种配置来编译应用和构建产品。

如图 10-1 所示，编译方案位于 Xcode 的工具栏上，停止按钮的右边。编译方案的前面是对象的显示图标，后面则是编译方案的名称“CrazyBounce-OC”。通过单击这个编译方案，便弹出方案控制菜单，然后就可以选择相应的编译方案来编译项目。或者，可以选择菜单栏上的 Product → Scheme 来选择相应的编译方案。此外，Xcode 还允许开发者管理、编辑以及创建编译方案。

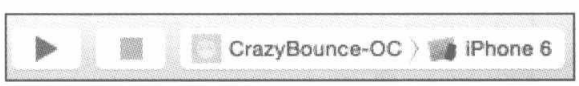


图 10-1　编译方案

10.1.1　管理方案

除了和 OS X 和 iOS 应用一同创建的单元测试对象外，Xcode 将会为每个对象自动创建一个编译方案。如图 10-2 所示，在工具栏上选择当前的编译方案，然后选择 Manage Schemes（管理方案）来开始对方案进行管理。

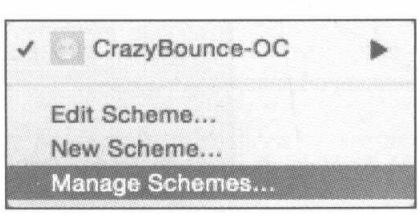


图 10-2　编译方案选择菜单

“管理方案”界面如图 10-3 所示。管理方案界面的中间显示的是当前项目所拥有的全部方案，在此可以查看这些编译方案的名称，以及拥有者（Container），用以确定该项目能够在某个项目或者某个工作区当中使用。

下面介绍管理方案中的操作。

1. 创建方案

单击“管理方案”界面底部的添加（+）按钮（见图 10-3），便会弹出一个表单，询问要创建的方案所链接的对象以及其名称，如图 10-4 所示。

我们已经介绍过，方案将对象与编译配置绑定在一起，因此创建的新方案应该选择一个链接对象。一般情况下，选择的链接对象应该是能够单独编译运行的目标对象，如图 10-5 所示。

选择完链接对象和方案名称后，单击“OK”即可完成创建。

在图 10-2 中“方案控制”菜单选择“New Scheme”也是一样的效果。

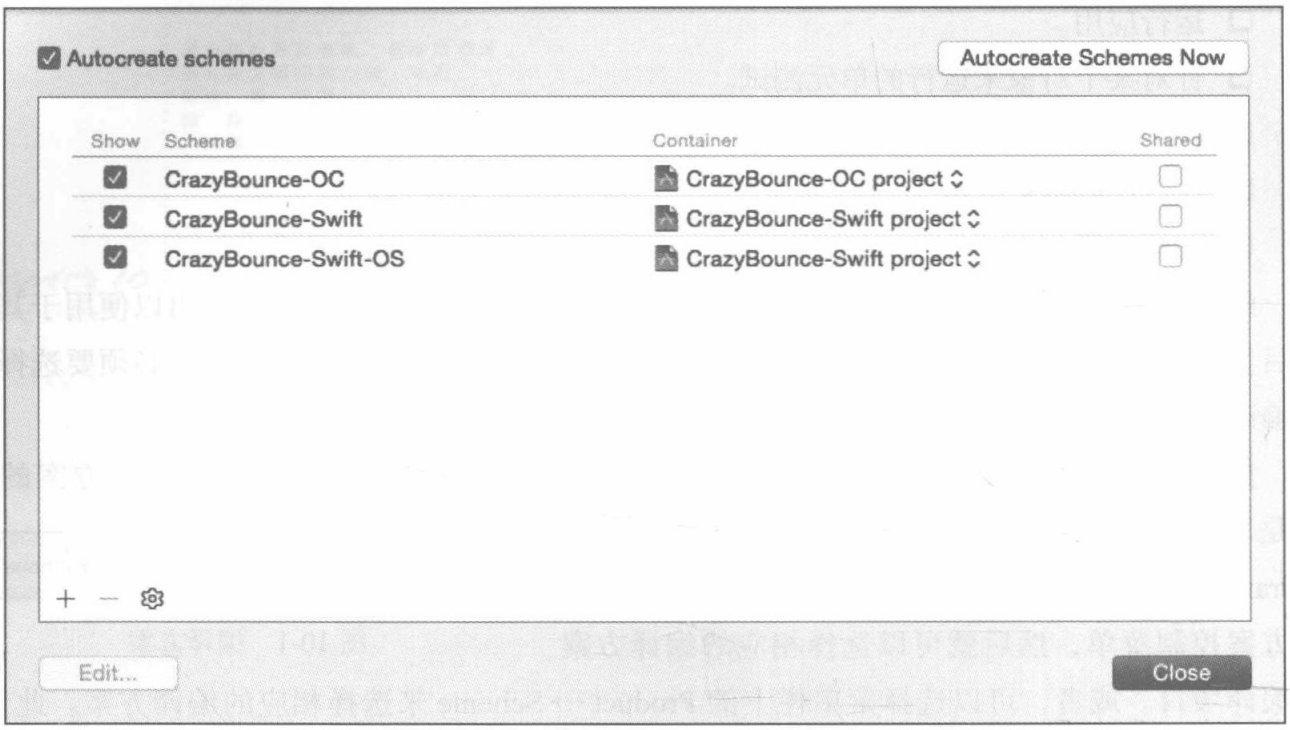


图 10-3　管理方案界面

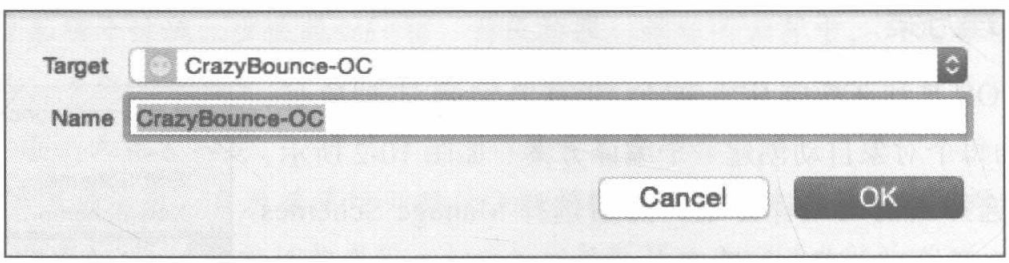


图 10-4　创建方案

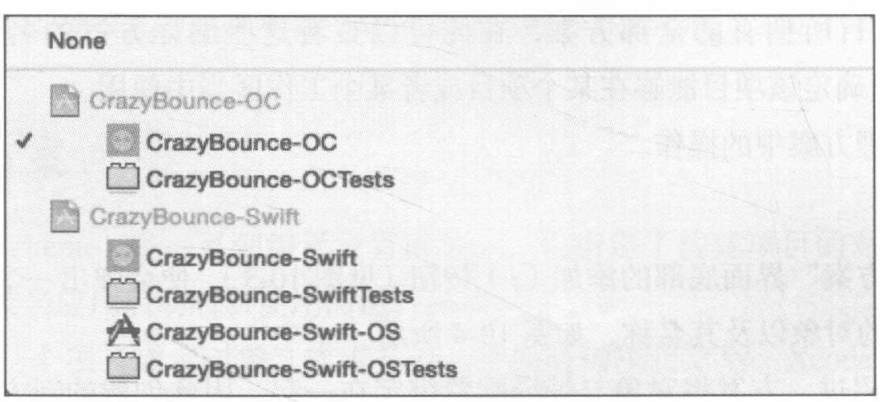


图 10-5　编译方案链接对象选择

2. 重排方案

就如同在项目导航器中管理文件一样，通过拖动操作可以将方案移到列表中所需的位置，以便完成重排操作。这个操作会影响方案在 Xcode 工具栏上的显示顺序。

3. 复制方案

选中要复制的方案，然后单击左下角的小齿轮按钮（⚙），在弹出的菜单中选择 Duplicate（复制），Xcode 便会复制一个一模一样的方案，并弹出方案编辑界面（方案编辑在 10.1.3 节讨论）。所复制的方案名称默认情况下是“Copy of 复制的方案名”。

4. 导入和导出方案

单击“管理方案”界面左下角的小齿轮按钮，就可以实现方案的导出（Export）和导入（Import），以便在其他的项目中使用。

5. 删除方案

选中要删除的方案，然后单击管理方案界面底部的删除（–）按钮，即可完成删除操作。如图 10-6 所示，Xcode 在方案删除前将会询问是否确认删除，因为这个操作是无法撤销的。如果确认删除的话，单击 Delete 按钮确认，这个方案就被删除掉了。

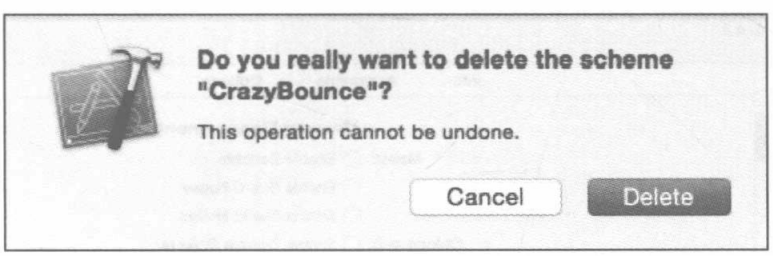


图 10-6　删除方案

注意　删除对象的时候，与之关联的编译方案并不会随之删除，因此这时候需要手动删除这些方案。

6. 选择方案容器

在“管理方案”界面的方案列表中，可以看到第三列的名称为 Container（容器）。方案必须要关联一个容器，默认情况下方案的容器都是初始对象所在的项目。

对于这个单个项目的应用来说，那么无需修改其容器，选择这个项目即可。如果应用使用的是包含多个项目的工作区的话，那么就可以选择将方案关联到工作区或者其他项目当中。

7. 方案共享

在“管理方案”界面方案列表的最右列上是名为“Shared”（共享）的栏目，每一个方案都可以选择是否进行共享操作。每个用户都拥有自己的方案集，这些方案将与用户名进行绑定。在多人协作的工程中，开发者都只能看到自己创建的方案以及共享的方案。因此，是否共享方案要根据实际情况来决定。

8. 自动创建方案

前面已经介绍过，Xcode 将在对象创建的时候为之自动创建新的编译方案，这一功能是被默认激活的，也就是管理方案界面左上角的“Autocreate schemes”（自动创建方案）复选框。如果不想自动创建方案的话，那么取消选择即可。

同样，右上角的“Autocreate Schemes Now”（立即自动创建方案）按钮将会根据当前项目中所拥有的对象，自动创建相应的编译方案。

10.1.2 编辑方案

实际上，方案里面的设置远比大家想象中要复杂。在管理方案界面中选中要编辑的某个方案，然后单击左下角的 Edit 按钮，或者在方案控制菜单中选中要编辑的某个方案，然后选择“Edit Scheme”。选中之后，便会显示“方案编辑”界面，如图 10-7 所示。

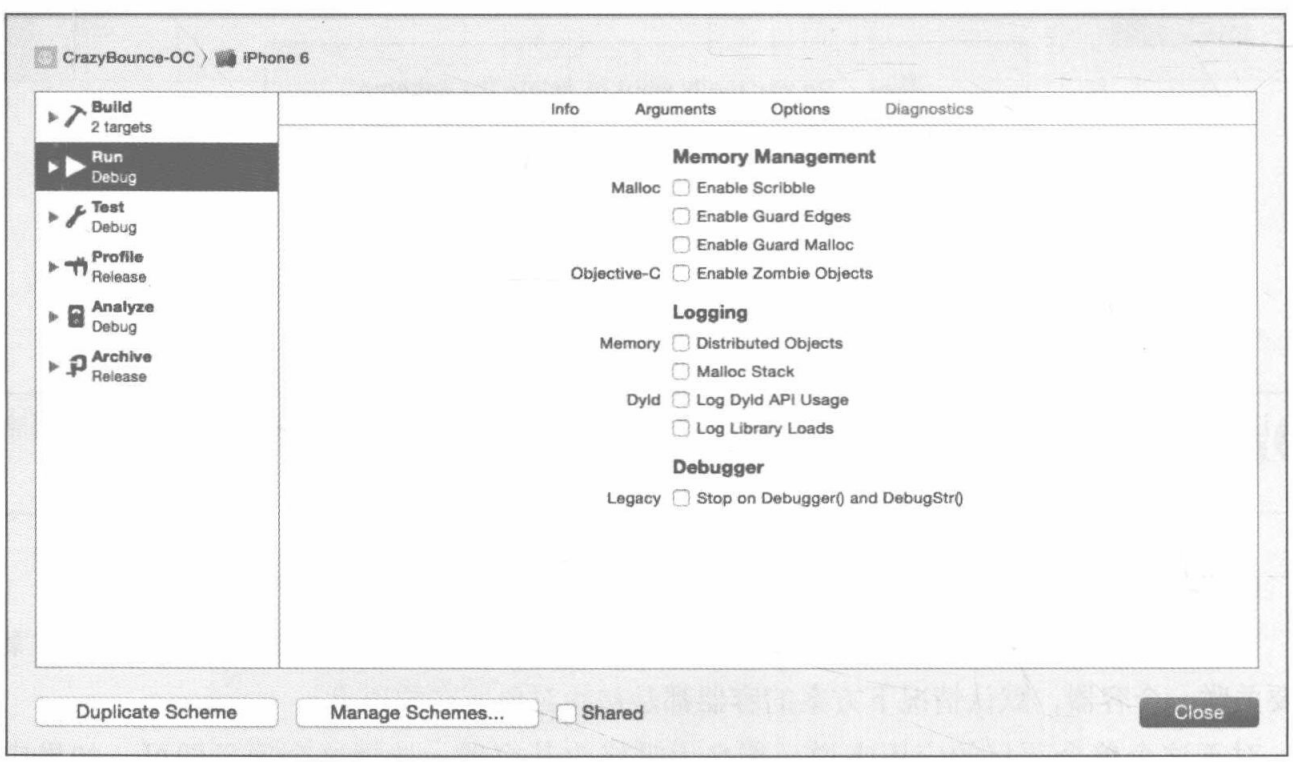


图 10-7 编辑方案界面

“方案编辑”界面的左上角选项可以用来选择当前正在编辑的方案以及该方案所绑定的运行目标。这个选项主要目的是让开发者无须关闭编辑界面就可以自由切换方案。除此之外，对这里进行变更也会同时改变工具栏上的方案和目标。

编辑界面的底部还拥有“复制方案”（Duplicate Scheme）按钮，用来复制当前正在编辑的方案。以及“管理方案”（Manage Schemes）按钮，这个按钮将切换回管理方案界面。同时，也可以在此设置是否共享此方案。

在界面中央有两个面板，用来编辑方案。左侧的面板显示对应菜单栏 Product 菜单下的操作列表，选择每个操作将会在右侧面板中显示该操作的方案设置。下面讲几个相关操作。

10.1.2.1　编译操作

Build（编译）操作是其他操作的基础操作，在项目运行、测试以及分发之前必须先将其编译，因此编译操作总是先于其他操作执行。要编辑这个操作设置，在方案编辑界面的左侧面板中选中“Build”选项，在右侧面板中将会显示编译操作的设置，如图 10-8 所示。

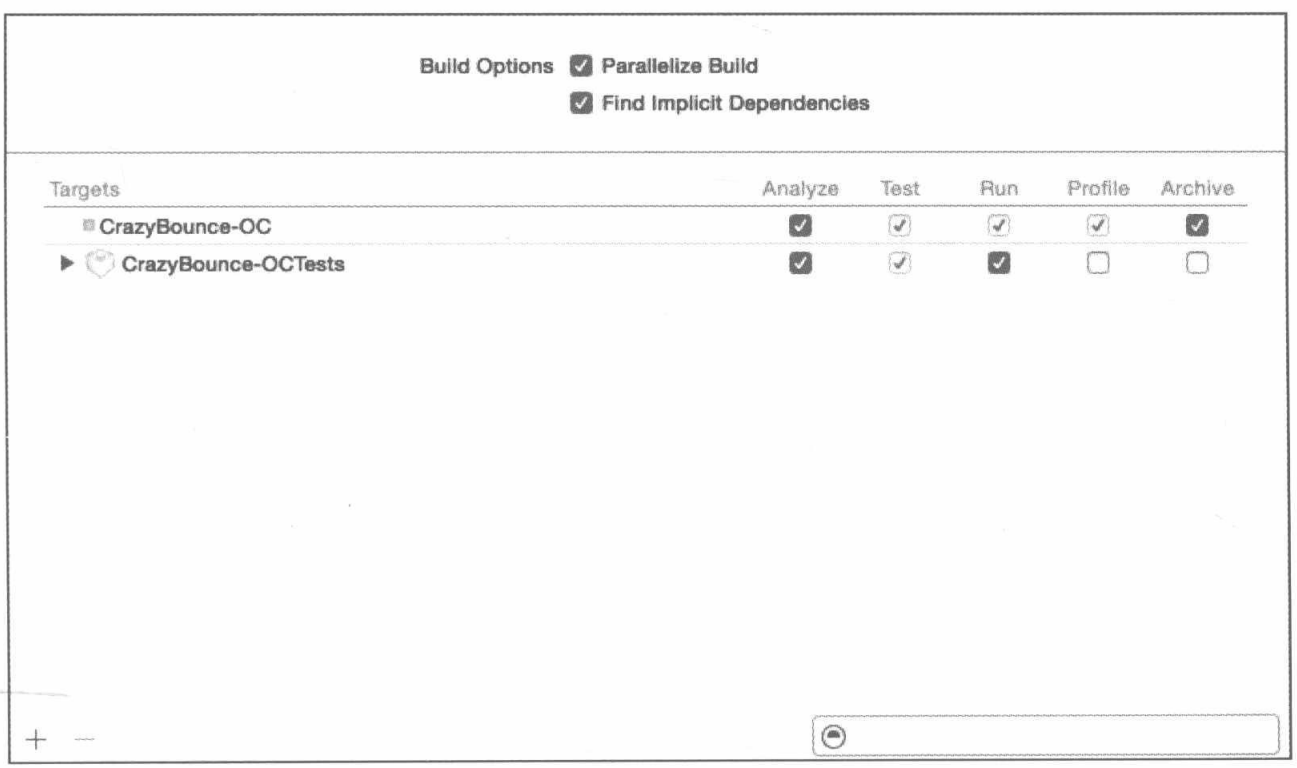


图 10-8　编译操作配置

编译操作选项中显示了当前项目可以编译的对象总数，在这里可以编译的对象有默认的对象以及其对应的单元测试对象，因此当前项目可以编译的对象总数是“2 targets”。

界面顶部可以看到两个编译选项。Parallelize Build（并行编译）将允许 Xcode 并行编译多个独立的对象（也就是不依赖于其他对象即可独立运行的对象），这个操作利用的是 Mac 电脑的多核并行操作。Find Implicit Dependencies（查找隐含依赖）将允许 Xcode 尝试自动寻找依赖。所谓依赖，就是指某个对象依赖于另一个对象的资源才能运行。Xcode 可以自行找出此类依赖，这样就可以以必要的顺序依次编译对象，以防止资源链接失效的情况出现，就不必自行定义依赖了。一般情况下，这两个选项保持默认选中即可。

编译选项下方则列出了当前方案中所要编译的全部对象。每个对象的右侧还有多个复选框，用来控制对应操作运行之前是否编译这个目标。比如，图中列出的两个对象，对于默认对象（iOS 应用）来说，无论是何种操作，默认都应该进行编译。而对于其对应的单元测试

包来说，那么在剖析和打包操作中，就无需将其打包，以减少最终应用的大小。具体的设置，应当根据该对象的特性以及需求来进行相应的更改。注意，某些操作的复选框是被禁用的，因为它们对于这些对象来说是必需的。如果需要取消选择这些操作，那么就需要转到该操作的 Info 选项卡，取消其对对象的约束。

通过这个列表，开发者也可以直接定义依赖，这在 Xcode 无法自行确定依赖的复杂条件下特别有用。使用底部的添加（+）按钮选择一个对象，这样在编译的过程中就会一并将全部对象进行编译。如果顺序对于对象编译十分重要的话，那么还必须取消选择并行编译选项。

10.1.2.2 运行操作

Run（运行）操作指定运行过程中所需要的可执行文件、所使用的调试器以及运行时的环境等等设置选项。如图 10-7 所示，Run 操作界面有四个选项卡：Info（信息）、Arguments（参数）、Options（选项）、Diagnostics（诊断），下面分别介绍。

1. 信息选项卡

Info（信息）选项卡保存了运行操作的基本设置情况，如图 10-9 所示。相关参数如下：

图 10-9 运行操作 – 信息选项卡

- Build Configuration（编译配置）菜单可以选择运行时所使用的配置（Debug 调试或者 Release 发型），默认情况下是调试配置，也就是可以使用调试器。
- Executable（可执行文件）菜单用来选择运行的可执行文件，也可以选择在运行时手动指定。如果将其改为没有包含在本方案中的对象，那么这个对象将会自动添加到这个

编译方案当中来。没有可执行文件的对象的可执行文件设置是 None，因为它们自己无法运行。

- “Debug Executable”（调试可执行文件）则是用来指定是否对当前选定的可执行文件执行调试操作，如果不执行调试，那么调试操作将不起作用。
- Debug Process As（以……身份调试进程）则允许开发者以自身的账户（默认选项）或者以根用户（root）的身份运行，如果开发者要调试某些必须以根用户权限才能运行的应用时，那么就必须使用根用户身份运行。

注意　iOS 程序只能够以当前账户进行调试，因为 iOS 不会对系统进行变更，只有 OS X 程序才能够选择这个选项。

- Launch（启动）选项命令 Xcode 在发起运行操作时是自动启动可执行文件（默认行为），还是等待开发者自行启动。

2. 参数选项卡

Arguments（参数）选项卡用来让开发者控制启动参数和环境变量，如图 10-10 所示。

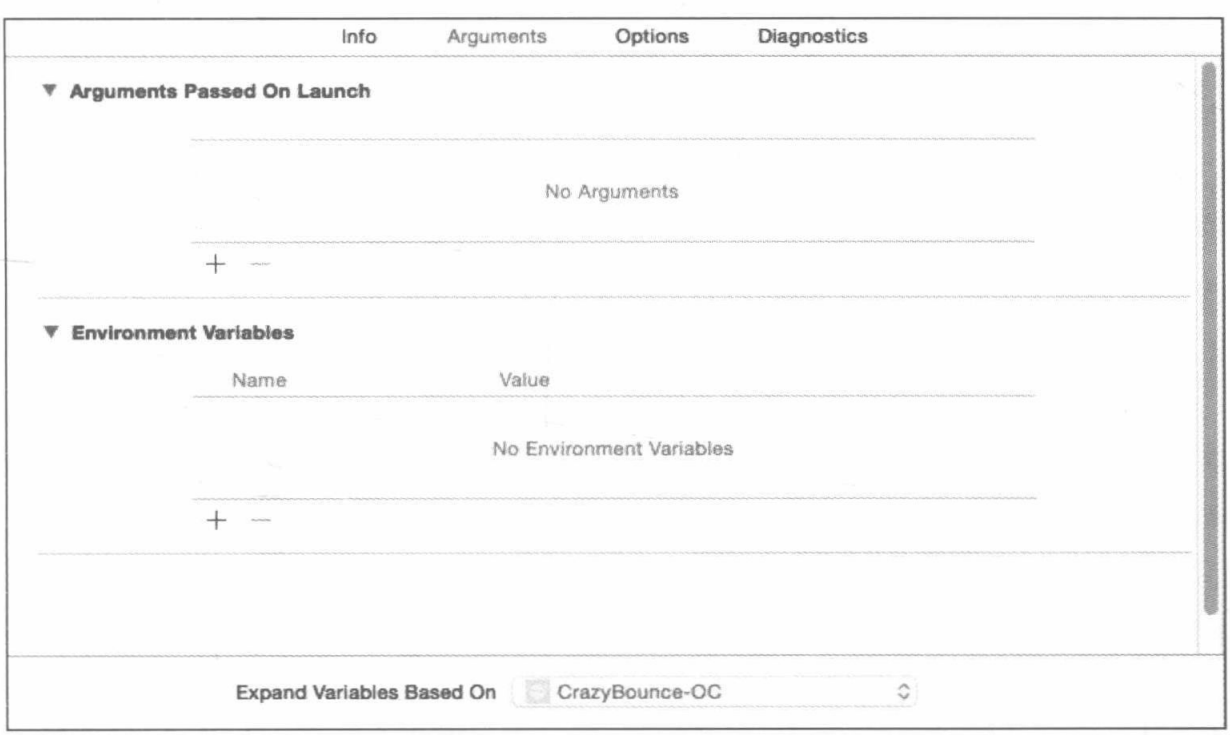


图 10-10　运行操作 – 参数选项卡

Arguments Passed On Launch（启动时传递的参数）列表中，可以使用添加（+）按钮，来添加当应用启动时所传递的特定参数。这些参数一般情况下是用在命令行程序上的，当然，如果有需要的话也可以运行在其他程序上。

Environment Variables（环境变量）列表中，可以添加或者覆盖当前应用环境中存在的环

境变量。

Exoand Variables Based On（根据……扩展变量）用来指定在扩展 Xcode 所提供的变量时，要使用哪个可执行文件的特定环境变量。

3. Options 选项卡

Options（选项）选项卡用来设置一些常用的运行选项。对于 OS X 项目和 iOS 项目来说，其他选项卡基本都是相同的，但是 Options 选项卡中有许多不同的选项。

OS X 项目的 Options 选项卡如图 10-11 所示，iOS 项目的 Options 选项卡如图 10-12 所示。

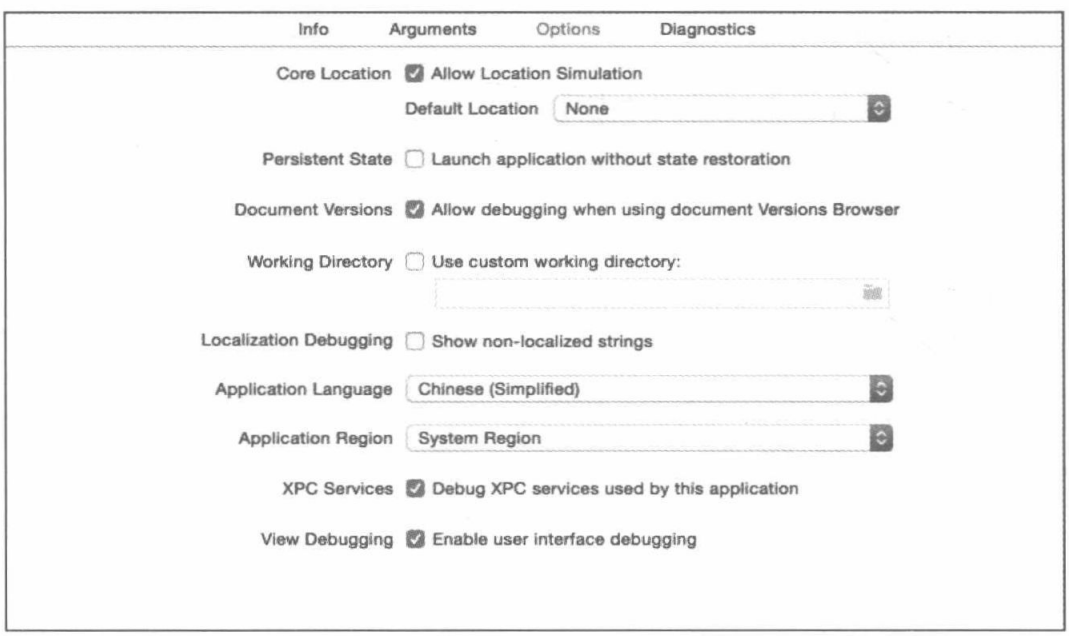


图 10-11　OS X 项目的 Options 选项卡

图 10-12　iOS 项目的 Options 选项卡

选项说明如下：

- Core Location 是两者都有的项目，这个选项用来设置定位模拟的相关选项。Allow Location Simulation（允许位置模拟）则允许模拟器在运行时模拟设备的地理位置。而 Default Location（默认位置）则是定义模拟器启动时，首选的地理模拟位置。这个下拉列表中有多个城市可供选择，当然，你也可以往项目中添加 GPX 地理位置文件来自定义地理模拟位置。
- 对于 OS X 来说，Persistent State（持久化状态）选项则用来设定应用在启动时，是否采用持久化状态来恢复启动。也就是说一般情况下，应用可以在程序退出前保存当前运行状态，然后在启动时，通过加载这个持久化状态来恢复应用退出前的状态。选择这个选项将会关闭这项功能。
- Document Versions（文档版本）选项则允许用户对文档版本浏览器进行相关调试，通过文档版本浏览器用户可以查看当前文档的历史版本，因此对这项功能进行调试还是很必要的。
- Working Directory(工作目录）则可以指定可执行文件运行时，其工作目录的所在路径。一般情况下不要对其进行变更，不过如果要限制可执行文件的访问路径的话，那么通过设置这个选项可以达成目的。
- Localization Debugging（本地化调试）是 iOS 和 OS X 应用都具备的项目，这个选项主要是让 Xcode 在调试的过程中，选择是否输出未经本地化操作的字符串。
- Application Language（应用程序语言）也是两个平台都具备的项目，这个选项主要用来定义应用程序的运行时所在平台或者所在环境的语言，和下面的 Application Region（应用程序区域）是类似的概念。通过这两个选项，可以模拟应用在不同国家、不同语言环境下所工作的情况，默认情况下是使用系统语言和区域。
- XPC Services（XPC 服务）则是设置如果当前应用使用了 XPC 服务的话，那么就对其进行调试。在上一章我们已经介绍过，XPC 是 OS X 处理多进程之间通信的服务，同时，它也解决 iOS 上扩展与应用通信的问题。
- View Debugging（视图调试）是 Xcode 6 提供的新功能，它允许开发者对用户界面进行实时的显示和调试，有关视图调试的相关内容，请参阅 11.9 节“视图调试”。
- 对于 iOS 来说，Application Data（应用程序数据）则允许开发者从项目中事先将数据放置到应用程序当中，这就需要 Application Data Package（应用程序数据包）的支持，如果当前工作区中不存在这个数据包，那么就不可以进行数据预加载的操作。
- Routing App Coverage File（航线应用覆盖文件）设置用于导航、交通相关应用，开发者可以在项目中指定一个 GeoJSON 文件，用来详细说明应用所覆盖的地理区域。
- GPU Frame Capture（GPU 帧捕获）为 OpenGL 之类的使用 GPU 的应用程序提供调试

支持，这样 Xcode 就可以捕获 GPU 绘制出来的每个帧画面，以供开发者调试。帧捕获可以选择 Automatically Enabled（自动启用），让 Xcode 自行根据应用代码来推断是否提供此项功能，当然也可以单独针对应用当中使用的 Metal、OpenGL ES 框架来进行调试，也可以手动关闭调试功能。此外，针对 Metal 调试，开发者还可以选择是否开启 Metal 的 API 验证功能，让 Xcode 检测 Metal 的 API 是否能够实现预期的效果。

- Background Fetch（后台获取）则允许开发者启用后台获取功能，这个功能开启后，应用将不会在前台自动显示，但是会在后台根据相关代码进行后台数据获取操作。这项功能需要实现“后台获取”的相关代码和开启 Xcode 的权限才能使用。

4. 诊断选项卡

Diagnostics（诊断）选项卡则指定了一些用于测试和调试应用相关设置。在这个选项卡中，开发者可以使用一系列的内存管理诊断，以及启动调试日志等服务，如图 10-13 所示。相关选项如下。

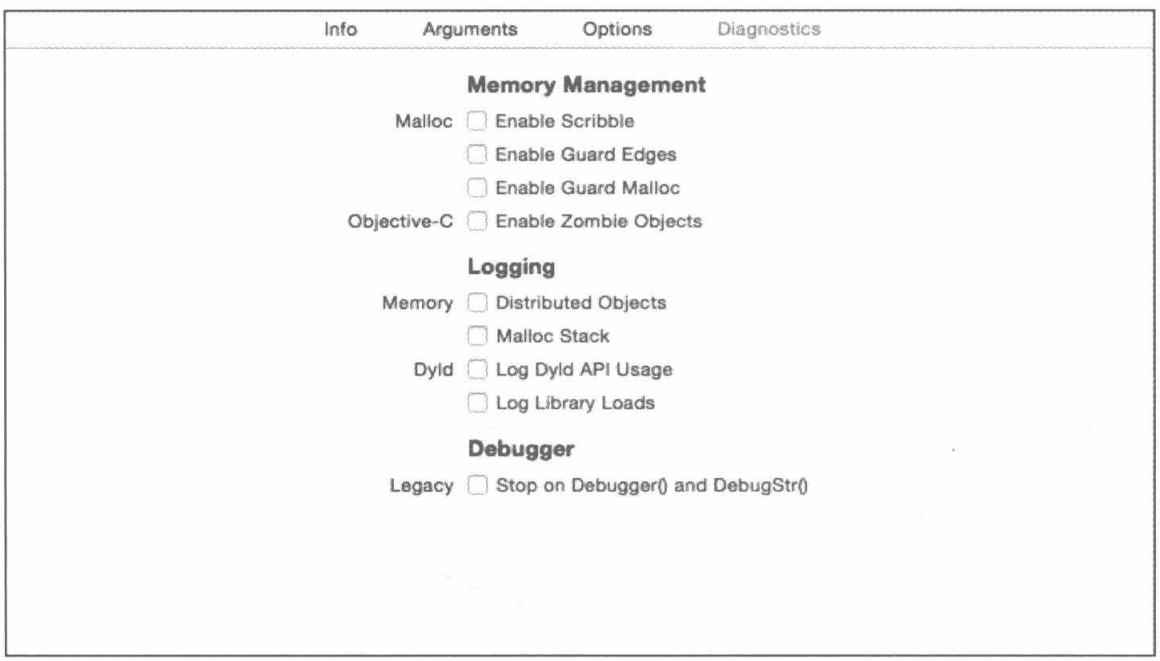


图 10-13　运行操作 – 诊断选项卡

Memory Managerment（内存管理）选项主要是用来开启一些内存管理相关的服务，这些工具将会在控制台调试区域中输出相关内容，它包含以下功能：

- Enable Scribble（启用内存涂抹）：向已分配的内存中填充 0xAA，向已销毁的内存中填充 0x55.
- Enable Guard Edges（启用边缘保护）：在分配大容量的内存前后添加保护页，保护现场。

- Enable Guard Malloc（启用动态内存分配保护）：使用 libgmalloc 来捕获常见的内存问题，例如缓冲区溢出。
- Enable Zombie Objects（启用僵尸对象）：以“僵尸”对象来替代被销毁的对象，以便尽可能地困住这个对象。如果向某个僵尸对象发送消息，那么运行时便会输出一个错误日志并且崩溃，并可以在追踪界面查看触发僵尸对象崩溃的调用语句。

对于 Logging（日志）选项来说，主要是用来设置应用调试过程当中，输出的日志包含什么内容，可以选择下述这些选项来向日志中添加额外内容：

- Distributed Objects（分布式对象）：启用对分布式对象（NSConnection、NSInvocation、NSDistantObject 以及 NSConcreteProtCoder）的日志记录
- Malloc Stack（动态内存分配栈）：记录分配内存和释放内存时相关栈的信息
- Log DYLD API Usage（记录 DYLD API 的使用）：记录动态链接相关 API 的调用信息（例如 dlopen）
- Log Library Loads（记录库加载信息）：记录动态链接库的加载信息

对于 Debugger（调试器）选项来说，目前只存在一个项目，那就是 Stop on Debugger() and DebugStr()，用来允许 Core Servives 服务例程以消息的形式进入调试器。这些例程讲法送一个 SIGNIT 标志给正确的进程。

10.1.2.3　测试操作

Test（测试）操作指定了当前方案所使用的单元测试包，如图 10-14 所示。测试操作拥有两个选项卡，分别是信息选项卡和参数选项卡。首先看到的是信息选项卡。

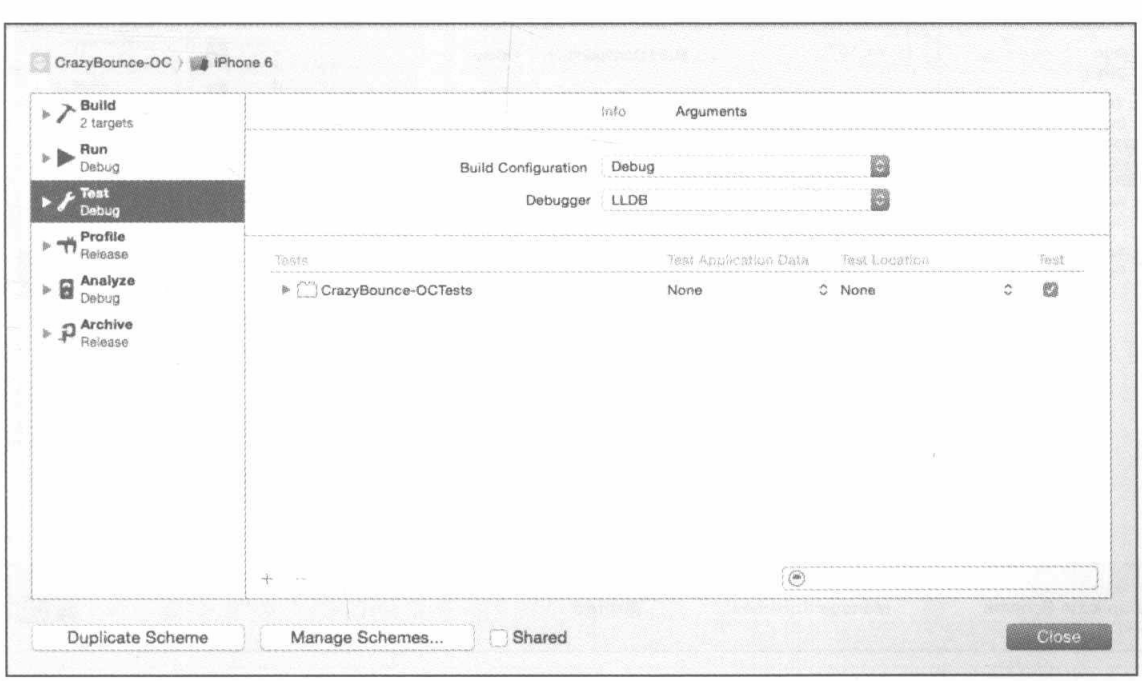


图 10-14　测试操作配置

列表中的单元测试包可以展开，里面包含了单元测试包中的所有单独测试类，对于这些单独测试类还可以继续展开，列出单独测试类当中的单独测试。使用 Test 复选框可以启用或者禁用某一个测试对象。此外，如果打算在列表中增加或者删除单元测试包，那么使用列表底部的添加（+）和删除（–）按钮即可完成增减操作。

测试操作界面的顶部有两个配置菜单。Build Configuratuib（编译配置）指定项目的编译方式，如果使用 Release 配置进行编译，那么单元测试中的调试符会被剥离，并且编译器优化将会启动，从而使项目难以进行调试。Debugger（编译器）菜单设置和运行操作中类似，用来指定运行单元测试时所使用的调试器。

参数选项卡允许开发者指定某些参数，作用和运行操作中相同，只不过更多是针对的是单元测试进行的配置。

测试操作的参数选项卡相比运行操作来说，多了一个选项：Use the Run action's arguments and environment variables（使用运行操作的参数和环境变量）。这样，测试操作便可以使用运行操作中所定义的参数和环境变量，如果想定义其他的参数，那么取消这个选项的选择即可。

10.1.2.4 剖析操作

Profile（剖析）决定了当我们使用剖析操作时，Xcode 会执行何种命令，如图 10-15 所示。Profile 操作拥有三个选项卡，分别是 Info（信息）选项卡、Arguments（参数）选项卡以及 Options（选项）选项卡。

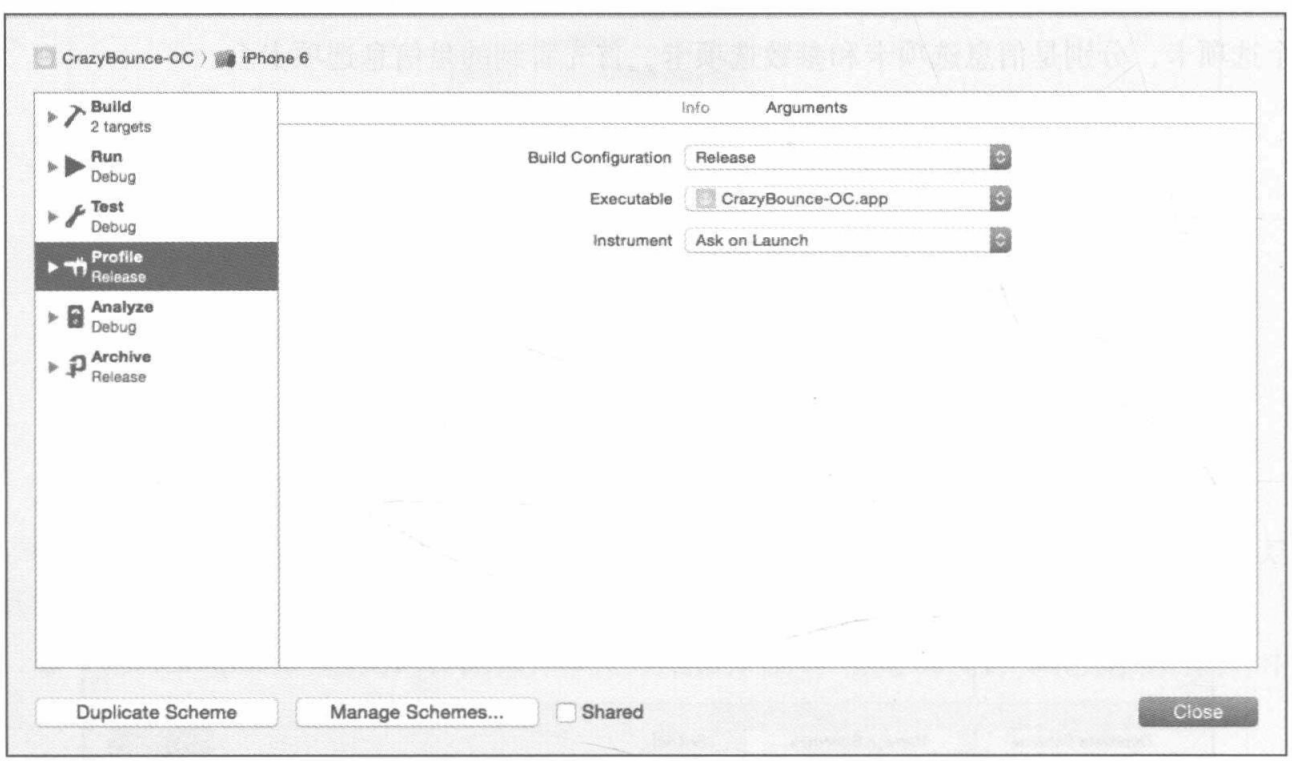


图 10-15 剖析操作配置

提示　iOS 应用的剖析操作界面并没有 Options 选项卡存在。

Build Configuration（编译配置）和 Executable（可执行文件）菜单在运行操作已经有所介绍，这里它们执行的操作是相通的。

最关键的是 Instruments 菜单，这个菜单允许开发者在应用启动时或者使用剖析操作时，选择何种 Instruments 工具运行，默认设置是启动时询问。关于 Instruments 当中的工具介绍，我们会在 11.10 节进行详细讨论。

剖析操作的参数选项卡和测试操作的完全相同。对于 OS X 项目才有的 Options 选项卡来说，它将允许开发者实现某些设置。这些设置和运行操作的 Options 选项卡的设置相同。

注意　如果选择启动时询问来执行剖析操作，并且 Instruments 工具正在运行，且已经执行了某个剖析工具，那么继续使用剖析操作将直接用上次执行的剖析工具再次运行可执行文件，而不会询问以何种工具运行。只有将 Instruments 工具窗口关闭，才能在下次选择剖析操作时出现提示。

10.1.2.5　分析操作

Analyze（分析）操作将会对编译操作中的指定对象运行静态分析器，关于静态分析器的有关内容，将在本书的 11.3 节“静态分析”中进行介绍。

分析操作只有一个选项，Build Configuration，如图 10-16 所示。这个选项和其他操作一样，指定了分析操作编译对象时所用的配置。

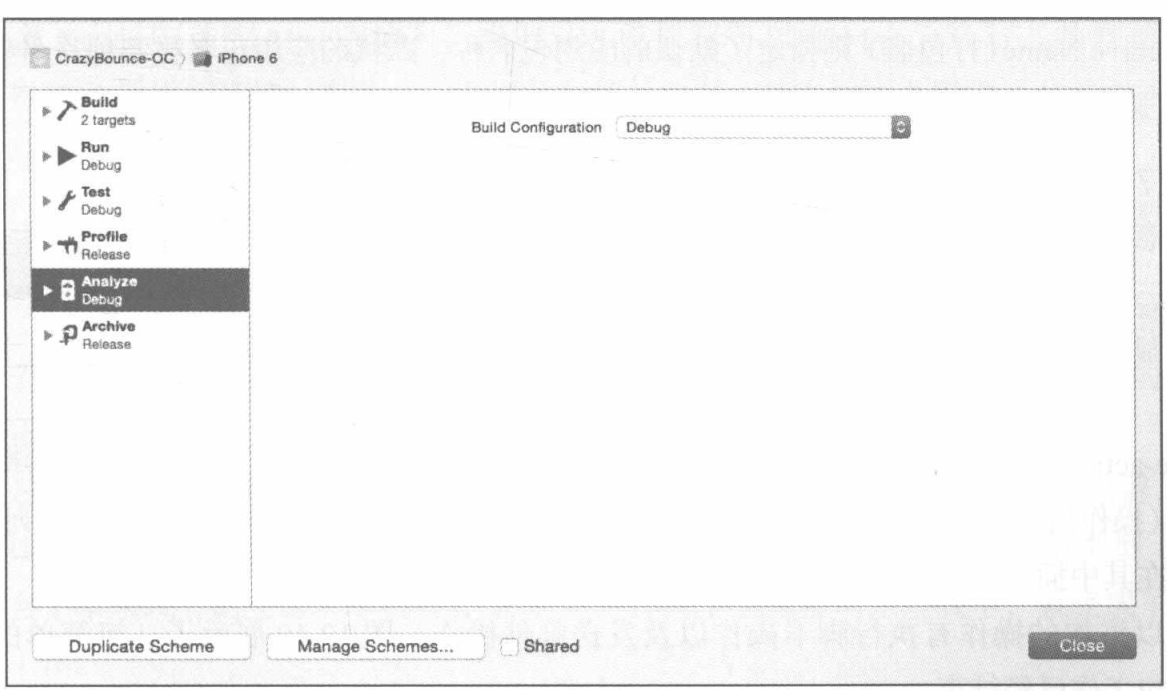


图 10-16　分析操作配置

10.1.2.6 打包操作

Archive（打包）操作让开发者在 Xcode 打包应用程序时进行一些个性化设置，如图 10-17 所示。关于打包的相关内容，将在第 14 章进行介绍。

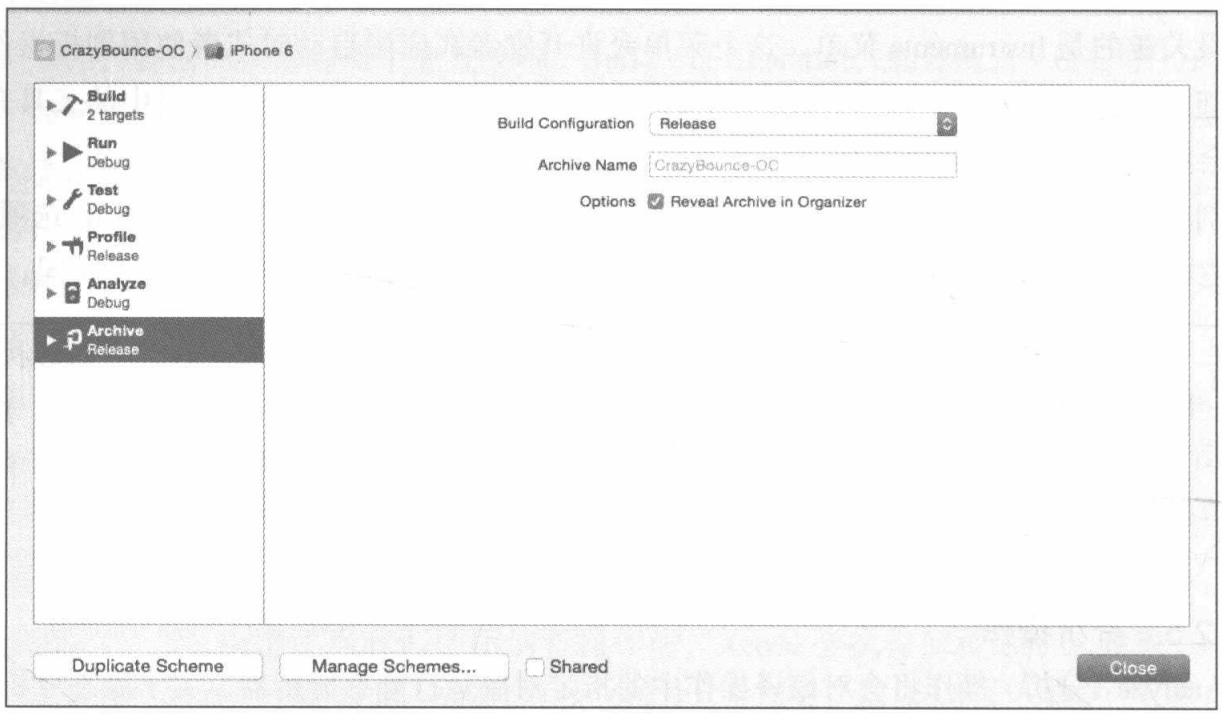


图 10-17 打包操作配置

Build Configuration 选项和其他操作一样，一般情况下执行打包操作采用的编译配置是 Release，因为应用的最终成品无需包含任何调试、测试的代码了。

Archive Name（打包名）则指定了默认的应用包名称，默认的应用包名称和项目名称相同。Options 选项决定 Xcode 是否当打包操作完成后，在组织器当中显示最终包对象。

10.1.2.7 额外操作

注意，每个操作旁边都有一个小三角形。展开这个三角形，就会看到三个条目，如图 10-18 所示。当选择操作时，会看到列表中间的项目显示的是操作本身。那么 Pre-actions 和 Post-actions 是什么意思呢？

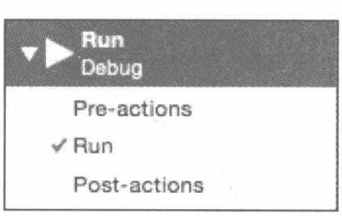


图 10-18 额外的操作选项

Pre-actions（操作前）是操作启用之前所完成的工作，而 Post-actions（操作后）则是操作启用之后所完成的工作。选中其中一项，就会看到一个空列表，然后可以在其中插入所要执行的操作。

可以添加的操作有执行脚本操作以及发送邮件操作。图 10-19 展示了一组简单的操作，通知打包工作已经结束。

图 10-19　操作脚本配置

10.2　运行目标

运行目标（destination）是应用运行的目的位置，也就是用来指定应用在哪一个设备上运行。运行目标可以是 Mac 电脑，也可以是 iOS 模拟器或者 iOS 真机。iOS 模拟器和 iOS SDK 一并随着 Xcode 安装到 Mac 电脑上，因此 iOS 模拟器可以在 Mac 上运行，并且可以模拟 iPhone 和 iPad 的运行环境。有关 iOS 模拟器的相关内容，请查看本书附录 B 中的 B.1 节。此外，运行目标除了支持平台的区别外，还有架构区别以及 SDK 版本的区别。

运行目标中的设备由 Devices 窗口来进行配置，通过菜单栏“ Window → Devices”，进入到设备管理界面，如图 10-20 所示。

在“设备管理”界面中，Devices 展示了当前操作系统所能够运行的真机，默认情况下总是包括了你的 Mac 电脑，如果你将 iPhone 连接上了电脑，那么就能够在这里看到你的手机。

Simulators 则显示了当前 Xcode 所拥有的模拟器设备名称和其 SDK 版本。

通过单击左下角的添加（+）按钮，就可以向设备管理界面当中添加新的模拟器，Xcode 会弹出如图 10-21 所示的界面。在这个界面中输入模拟器名称，然后选择模拟器类型（一般情况下包含了苹果现有的全部 iOS 机型），然后再选择这个模拟器的系统版本，就完成了模拟器的创建工作。

单击左下角的设置（小齿轮）按钮，便可以完成模拟器重命名、模拟器删除以及设定是否

将设备显示在运行目标菜单当中。

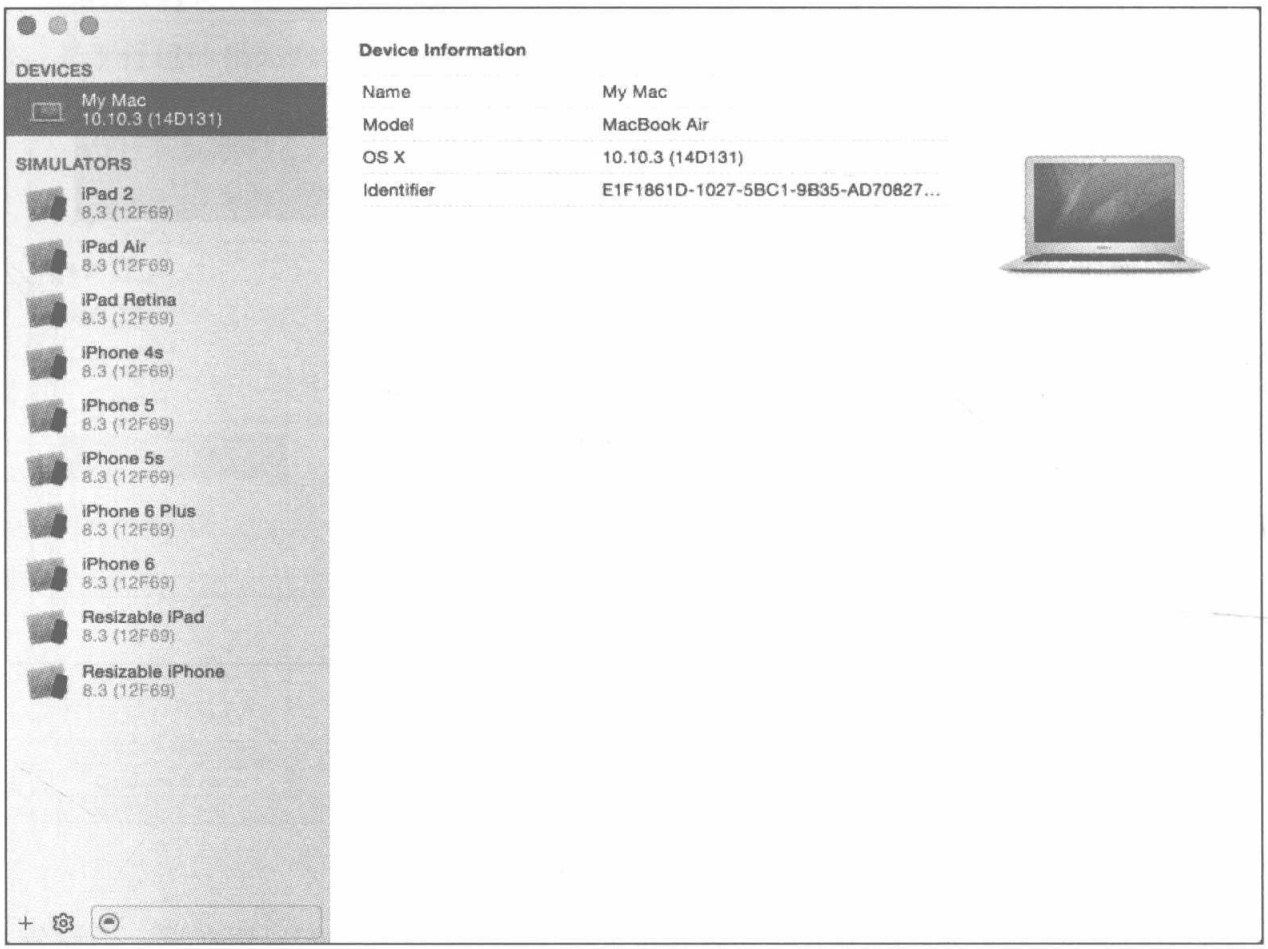


图 10-20 “设备管理”界面

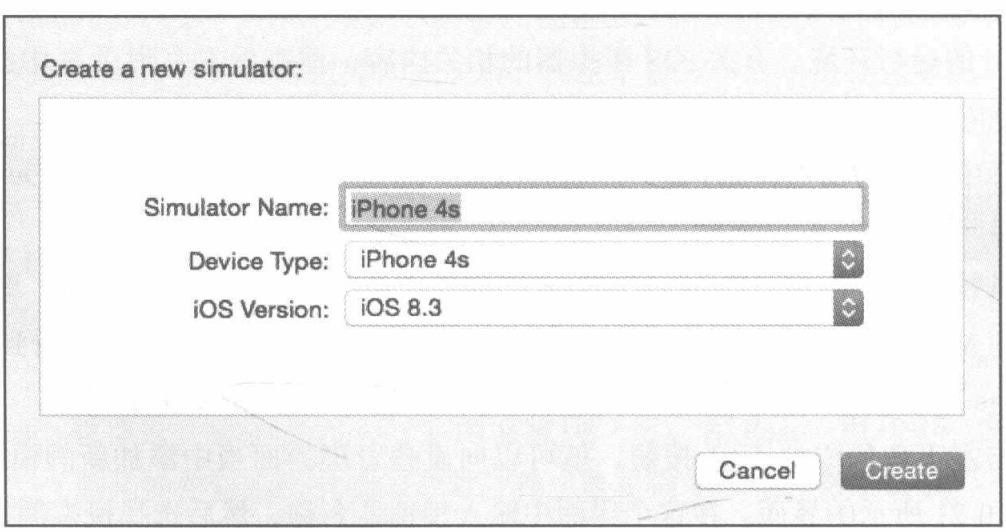


图 10-21 添加新设备

对于 iOS 真机来说，它的界面和 Mac 电脑以及 iOS 模拟器就不甚相同了，如图 10-22 所示。

图 10-22　连接 iPhone

iOS 真机除了显示一些通用的数据之外，还展示了当前真机上所有进行真机调试所安装的应用。除此之外，还可以通过 View Device Logs 按钮来查看真机上所保存的调试日志，然后通过 Take Screenshot 来截取当前 iOS 设备上的运行画面。

对于当前真机上所安装的应用，除了可以添加（+）以及删除（-）之外，还可以通过设置按钮，来查看这些应用的沙盒目录（Show Container），导出沙盒（Download Container）以及替换沙盒（Replace Container）。

“知晓这些知识之后，大致就可以开始实践了，那么鄙人再给你……”话音未落，少年良辰仿佛发现了新世界一般，眨眼就跑不见了。大师见此，叹了一口气：“年轻人心急躁动，实乃是习武之大忌啊，希望不会出什么差错。”

远方影影绰绰，仿佛还能看到良辰活跃的身影，旋即消失在了烟雨之中。

Chapter 11 第 11 章

谨防走火入魔——调试

为了能够早日达到梦中的境界，少年良辰日日将自己关在练功室内，一步不出。或有一日他的潜心修炼能够最终助他登上武林之巅，然而此刻的他，虽感到体内力量如泉水般涌上，却难以控制这股在体内四处乱窜的力量。

江湖上高手如林，但也有不少高手最后并没有葬送在敌人的手上，而是葬送在了自己所不能控制的力量之下。若不能将内力牢牢掌握在自己控制范围内，必将走火入魔，最后只落得妖魔反噬的下场。

11.1 语法错误

最容易发现并修改的错误就是语法错误了。Xcode 默认情况下会对开发者的代码进行语法检查，如果发现了不符合语法的地方，那么 Xcode 就会弹出警告，甚至错误提示。Xcode 会在出错的源代码处加以高亮显示，并在源代码左边的边列中显示对应的图标来标记问题。警告以一个黄色的三角形显示，有些像交通路标的警示牌。而错误提示测试以一个红色的圆形显示出来，如图 11-1 所示。

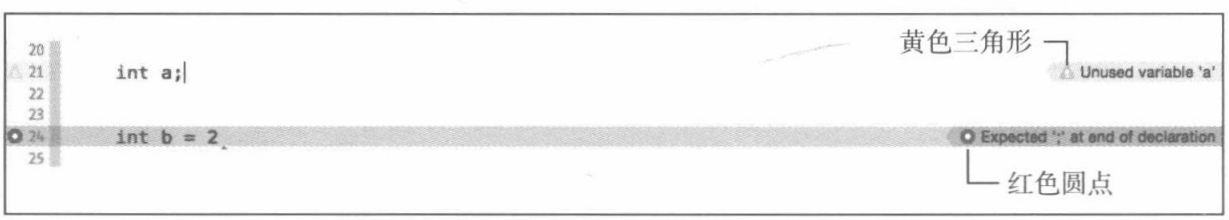


图 11-1 语法错误标识

借助 Xcode 的语法检查功能，开发者可以在第一时间及时地发现代码中的错误，并加以解决。

对于某些错误，一般情况下 Xcode 都能够提供修改建议。比如我们写了如下一行代码：

```
NSString* str = @"This is a String"
```

对于这行语句，有过 Objective-C 语言基础的开发者都能够一眼看出来这行语句少了一个分号“;”，Xcode 能够迅速查询到这个错误，然后给这行语句添加了一个红色的错误提示。

可以看到，Xcode 在这条语句的行标号前面添加了一个红色圆圈，中间含有了一个白色圆点，这就表明了 Xcode 能够提供修改建议。同时，Xcode 也在语法出错的地方显示了一个小小的三角形，提示开发者语法出错的位置。

点击这个红色圆圈，Xcode 便会弹出修改建议，如图 11-2 所示。此外，Xcode 也在这行语句后面显示出了出错的原因：`Expected ';' at end of declaration`（语句后缺少';'）。Xcode 弹出的修改建议便是：Insert ';'（插入';'）。点击确认这个修改建议，Xcode 便会自动在语句后面添加分号，完成修改。

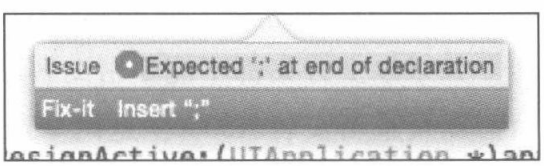


图 11-2　错误修改建议

有时候，Xcode 不确定当前的具体语法状态，因此可能会弹出多个修改建议，这个时候开发者就需要根据自己的实际情况选择正确的修改建议。


注意　Xcode 提供的修改建议并不总是正确的，有时候你使用了一些复杂语法的话，那么 Xcode 便可能会误解你的真实意思，然后给出错误的修改建议。因此，不要过分迷信 Xcode 的修改建议。

对于某些 Xcode 不能够给出修改建议的条目，往往都是开发者违背了某项语法规定而导致的，要解决这个问题，开发者必须查看所使用的语法相关文档，看看自己有没有违背了某项规定。如果还没有办法的话，可以在相关的搜索引擎中查询其错误提示，以获得更多的帮助。

11.2　编译时错误

编译时错误是应用在编译过程中产生的错误，一般情况下编译时错误会在程序编译的时候发现。语法错误也能引起编译时错误。

出现了编译时错误的话，Xcode 会自动打开问题导航器（Issue Navigator），在其中显示当前项目编译过程中所出现的问题。

出现这些问题的原因往往是某些设置没有配置完毕或者配置错误，以下步骤能够解决近

半数以上的 Xcode 的编译时错误问题，请在寻找其他解决方案之前或者无法解决编译时错误时之前尝试一下这些方案。

1）选择菜单栏上的 Product → Clean 选项，清除项目配置，然后再重新编译。

2）重启 Xcode。

3）检查应用设置的 Build Settings 的 User header paths 设置是否正确。

4）将 Always search user paths 设置为 YES。

5）对于项目中所有的对象来说，将 Skip Install 选项全部设置为 YES。

6）将项目 Build Phases 中的所有“公有”头文件移除。

7）将对象的 Build Settings 中的 scan all source files for includes 设置为 YES。

8）向对象的 Build Settings 中的 Installation Directory 添加 $(LOCAL_APPS_DIR)。

9）上网寻求帮助，推荐 StackOverFlow、Google 等。

如果以上方案都不能够帮助解决问题，那么似乎你只剩下几个选择了：重新建立新的项目，以及重新安装 Xcode。这两个选择是大家都不希望出现的。所以在开发过程中，对 Build Settings 的操作要十分小心，尤其是在添加和删除第三方库的时候，一定要按照第三方库的配置说明文档来进行操作。最最重要的是，要备份任何一个可以正常运作的版本，以提供一个反悔的机会。

11.3 静态分析

在 Xcode 中，警告、错误和分析器结果都称为问题（Issue），这些问题都会显示在问题导航器当中。而静态分析器所找出来的潜在问题将是以蓝色标识显示的。

11.3.1 使用静态分析器

Xcode 集成了 Clang 静态分析器，这是一个源代码分析工具，用来在 C、C++ 以及 Objective-C 项目中寻找出更多的隐藏在代码当中的问题，比如内存泄露、无用变量等问题。与编译器的语法检查更强的是，静态分析器能够显示大量的细节，包括导致问题的执行路径以及问题发生的原因。之所以称为静态分析器，是因为应用无需运行，即可对代码本身进行分析。

目前，Clang 静态分析器并不支持 Swift 语言，因此对于 Swift 语言来说 Analyze 分析功能就显得可有可无了，因为 Clang 并不能帮助 Swift 语言找到相应的错误。于是对于 Swift 代码来说，调启静态分析功能实际上是调启了编译功能。

不过 LLVM 对于 Swift 的语法分析功能十分强大，有很多在 Objective-C 上可能会引起

问题的代码等换到 Swift 上是无法通过编译的。对于 Swift 和 Objective-C 混编的应用来说，分析会对代码中存在的 Objective-C 代码进行分析，检查 Objective-C 中的问题。关于 Clang 静态分析器的有关信息，可以参见：http://clang.llvm.org/ger_started.html。

定位到 CrazyBounce-OC 项目，然后选择菜单栏的 Pruduct → Analyze（分析），或者长按运行按钮，在弹出的菜单中选择 Analyze，就可以调启 Xcode 的分析操作。调启时，分析器将会标识开发者代码中的问题。如果找到一个问题，就会在源代码编辑器和问题导航器中标记，和警告、错误一并显示，如图 11-3 所示。

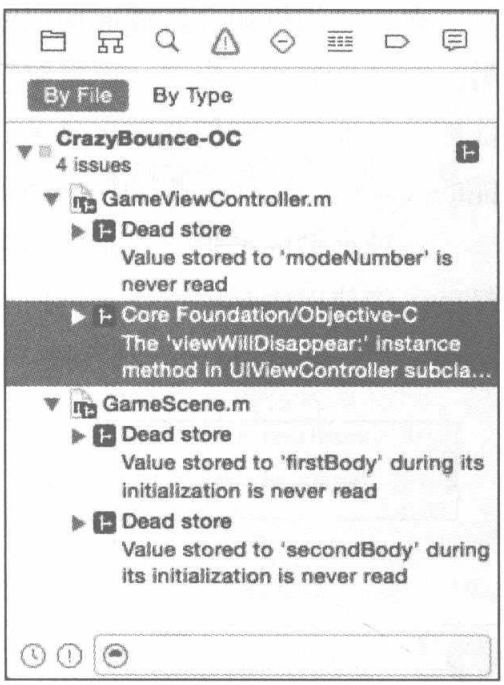


图 11-3　分析操作结果

单击编辑器或者问题导航器中的问题，会在编辑器中显示有关该问题的细节，包括指示有疑问的代码路径的蓝色线条，以及路径上错误操作的描述，如图 11-4 所示。

如果问题发生在很长很长的一段代码当中，那么查看这些蓝色线条可能会非常让人恼火，眼花缭乱的颜色很容易造成混淆和视觉疲劳。Xcode 提供了两种方法来更好地帮助开发者跟踪有疑问的代码路径。

第一种方法是在问题导航器中，点击三角形以展开问题，然后单击问题中的某个选项，以定位有疑问的代码路径上的某个位置。

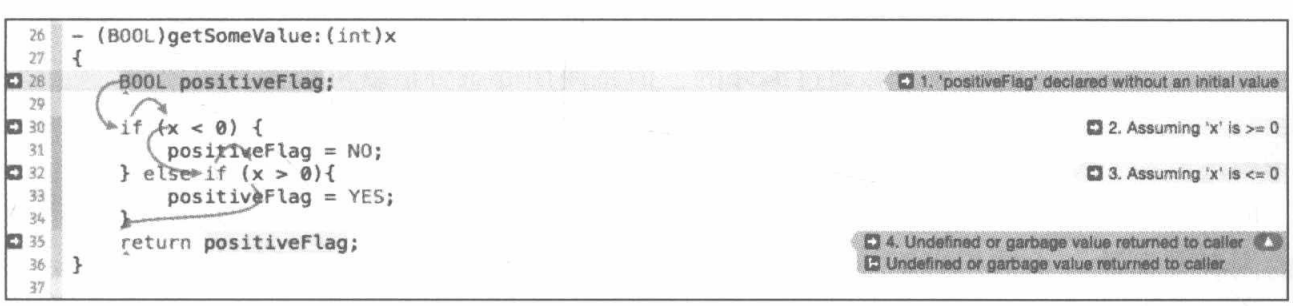


图 11-4　分析问题的细节

第二种方法更加简便，这个时候编辑器区域顶部会出现一个分析器结果栏，如图 11-5 所示。这个结果栏可以帮助开发者按次序浏览分析器所指出的问题。单击问题描述，即可展开一个问题的全部列表，从中可以进行选择。也可以使用结果栏右侧的按钮，查看上一个或者下一个分析器结果。

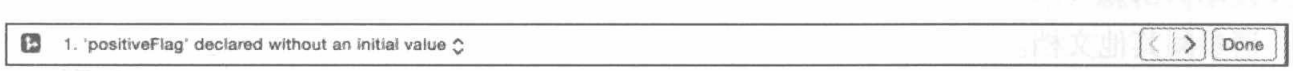


图 11-5　分析器结果栏

11.3.2 分析所解决的问题

发现了问题，那么就要着手解决问题。静态分析所能分析出来的问题，不外乎如下几个：逻辑错误、内存泄露、死存储问题、API 问题等，下面分别介绍。

11.3.2.1 逻辑错误

逻辑错误可能会造成应用崩溃或者一些奇奇怪怪的问题，导致应用不能够按预期正常运转。在复杂的代码当中，逻辑错误很可能会被忽略掉，从而很难进行调试。但是静态分析器能够轻易地找出它们并突出显示，苹果建议开发者有意识地执行分析操作，使其检查所有可能的代码路径，以检测其中复杂的逻辑错误。

一种类型的逻辑错误是使用未初始化的变量。在声明变量时没有指定初始值，那么该变量就可能使用内存中这块区域的垃圾值，也就是之前存放在这块内存当中的值。如果试图访问该变量，就可能会得到未预料到的值。图 11-6 用一个简单的场景说明了这个问题。

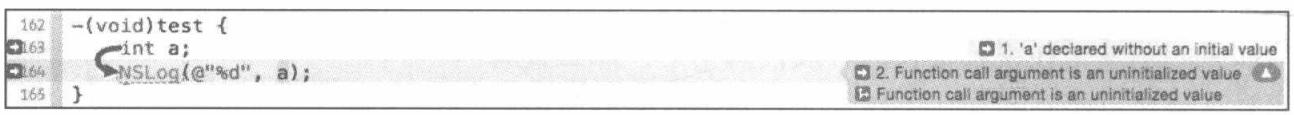


图 11-6 逻辑错误

对于 Swift 来说，这个错误是以“语法错误”的形式来提示的。

此外，另外一个会造成逻辑错误的例子就是解除空指针的引用。比如说以下代码：

```
int *p = 0;
*p;  // 解引用了一个空指针
```

含有这种类型的代码是能够通过编译的，但是使用静态分析就可以查找出错误。

11.3.2.2 内存泄露

简单来说，内存泄露就是应用丢失了对某个对象的引用，因而它不再会被告知要释放占用的内存并删除。也就相当于一个保险柜丢失了钥匙，这个保险柜里面的东西就永远占着这个位置了，无法读取，也无法放入新的东西。因此，这就是所谓的“内存泄露”，因为它无法在该应用生命周期的剩余时间内被回收。并且，这个对象一直占用着内存空间很有可能导致程序崩溃等问题发生。

使用静态分析，可以有效地找出任何可能的内存泄露问题。一般情况下，内存泄露在 ARC（自动循环计数）内存管理出现之后基本就很难再出现了，不过唯一要注意的是强引用循环。强引用循环是开启 ARC 后无法用静态分析寻找出来的内存泄露问题，关于它的更多信息，请参阅其他文档。

11.3.2.3　死存储问题

所谓的死存储指的是代码中不会被访问的变量。这里指的不会被访问，指的是在对某个变量进行处理后，在其生命周期内，不再对其进行访问。

我们的 CrazyBounce-OC 项目中包含了这种“死存储”的问题，这种问题我们无法通过编译找到，但是能够通过分析来寻找到，如图 11-7 所示。

```
104         default:
105             self.gamemode = @"Items";
106             self.lbl_GameMode.text = @"Items Mode";
107             break;
108     }
109     modeNumber = 0;                    Value stored to 'modeNumber' is never read
110 }
```

图 11-7　死存储问题

在这段代码中，我们将 modeNumber 赋值为 0 后，这个方法就结束了，此时 modeNumber 就会被自动释放。因此，modeNumber 就成了一个死存储代码，因为我们不再会对其访问，这个赋值语句是无效的。

11.3.2.4　API 问题

API 问题主要是由于没有遵循项目中所使用的框架或者库的要求而导致的，也就是说，API 问题是不规范的用法。一般情况下，分析器会分析框架的要求，然后对实现代码进行分析，查看代码是否符合框架的要求。

一般说来，分析器只能检测使用系统自带框架所导致的问题。

我们的 CrazyBounce-OC 项目同样包含了这种 API 问题，这个问题不会报错，只能通过分析器找到，如图 11-8 所示。

```
126 -(void)viewWillDisappear:(BOOL)animated {
127     [self.gameScene removeAllChildren];
128     [self.waterWaveView removeFromSuperview];
129 }          The 'viewWillDisappear:' instance method in UIViewController subclass 'GameViewController' is missing a [super viewWillDisappear:] call
130
```

图 11-8　API 问题

在这段源代码中，我们重写了父类 UIViewController 的 viewWillDisappear 方法，但是我们忘记了调用父类的该方法，这可能会导致某些严重错误。

11.4　断点调试

所谓断点（breakpoint），就如同人体的穴位，给程序的某处放置了断点，就相当于给人施展了“葵花点穴手”，让人动弹不得，停止在之前的状态不得动弹。

断点就正是这么一个让应用暂停运行的机制，以便让开发者执行调试，查看变量、寄存

器的值，等等。可以说，一般情况下的调试，都是基于断点来进行的，或者说都必须要先加断点。可见断点的重要性。

断点类似于“中断”，当应用运行到断点所在位置时，Xcode 将会保存现场，暂停应用的运行，然后将应用的控制权转交给调试器，然后等待开发者的调试指令。

一旦应用程序被暂停，所暂停的代码行将被高亮显示，并且以一个绿色箭头标识出来。

11.4.1 添加断点

通常情况下，最简便的添加断点的方法是：找到想要添加断点的源代码位置，然后在代码编辑器左侧的边列中点击该条语句的所在行位置，即可完成断点的添加。

如果觉得不好定位的话，那么前往菜单栏的 Xcode → Preferences，然后在 Test Editing 选项卡中，勾选 Show: 当中的 Line numbers，将代码行数字显示出来，这样添加断点就更加清晰明了。

如图 11-9 所示，我们在这个文件的第 38 行添加了一个断点。

```
38    var gamemode = ""
39
40    var waveHeight: CGFloat = 120/568
```

图 11-9　添加的断点

添加的断点以蓝色的箭头显示，这个时候我们编译并运行程序，就会发现应用停止在了这个断点处，如图 11-10 所示。应用暂停之后，我们就可以对这个应用为所欲为了。

```
37    var best = false
38    var gamemode = ""                                    Thread 1: breakpoint 1.5
39
40    var waveHeight: CGFloat = 120/568
```

图 11-10　程序暂停在断点处

通过单击断点，可以切换断点的激活状态。失效的断点将变成浅色，显示为半透明状态。要完全删除一个断点，最简单的方法是将其从边列拖出来“丢掉”，这个时候鼠标会变成含有“X”的样式，然后断点就会化为一股“白烟”，消失在茫茫代码中。

我们也可以使用快捷键。按下“Command + \”键将给当前代码行设置一个断点，再次按下该快捷键，即可删除断点。

此外，右键单击断点，将弹出一个如图 11-11 所示的菜单，可以用其来对断点进行控制和管理。Edit Breakpoint（编辑断点）选项可以对断点进行相应的设置（见本章 11.4.3 节“断点设置”）。Disable Breakpoint 选项可以令断点失效；Delete Breakpoint 选项可以删除断点；Reveal in Breakpoint Navigator（在断点导航器中显示）选项将打开断点导航器，并在其中高亮显示我们右键点击的这个断点。

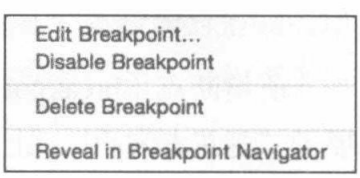


图 11-11　断点设置菜单

Xcode 允许在代码文件中的任意一行添加断点，但是要注意的是，在代码文件中设定断点的时候，实际上是在断点所在的那一行之后的第一条可执行指令处设置断点。比如说，图 11-12 中显示了三个断点，实际上这三个断点都指向了同一个位置。

```
83          // 添加水底
84          var waterRect = CGRectMake(self.frame.origin.x, self.frame.origin.y, self.frame.size.width, 1)
85          var water = SKNode()
86          var a:Int
87
88          water.physicsBody = SKPhysicsBody(edgeLoopFromRect: waterRect)
89          self.addChild(water)
90          water.physicsBody?.categoryBitMask = waterCategory
91      }
```

图 11-12　实际上相同的断点

第一个断点位于语句声明处，这个语句并没有进行任何赋值操作，因此它不会生成任何代码；第二个断点位于一个空行，只有第三个断点所在的地方才是一条可执行代码。最终，Xcode 将会跳过第一个和第二个断点，直接在第三个断点处停止。这三个断点实质上是一个断点，它们都指向了同一个内存地址。

11.4.2　断点导航器

断点导航器（Breakpoint navigator）用来统一显示和管理当前项目或者工作区中的所有断点，如图 11-13 所示。在这里，我们在 Ball.swift 中的 knockBall() 方法内添加了一个断点，以及在 Bar.swift 中的 barBackShort() 方法内也添加了一个断点。这个时候，最顶部的项目图标显示了当前该项目中总共有两个断点。

右键点击这些断点，弹出如图 11-14 所示的菜单。其中，Share Breakpoint（分享断点）

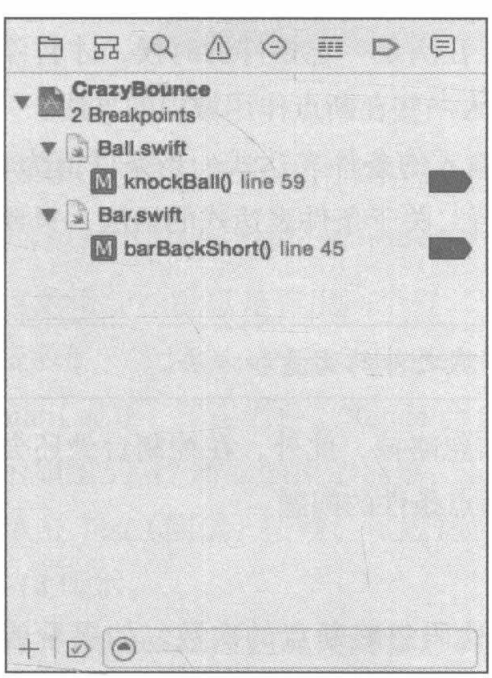


图 11-13　断点导航器

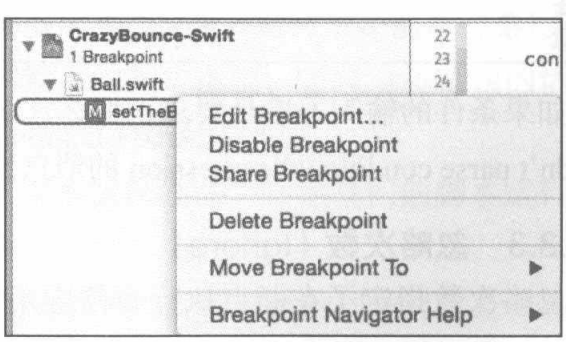


图 11-14　导航器断点菜单

这个选项用来与你开发团队中的其他成员共享断点。断点实际上是私有的项目，如果我们没有把断点进行共享的话，那么这个断点对于这个项目的其他成员来说是不可见的。

Move Breakpoint To（移动断点到）选项主要是用来在不同的项目、工作区之间移动断点。

匹配栏区域有两个按钮。第一个按钮清晰明了，用来添加不同类型的断点。第二个按钮则是用来控制是否显示失效的断点。

11.4.3 断点设置

右键点击某个断点，然后选择 Edit Breakpoint，可以对断点进行相关的自定义设置。这个选项将会弹出一个断点编辑器窗口，如图 11-15 所示。

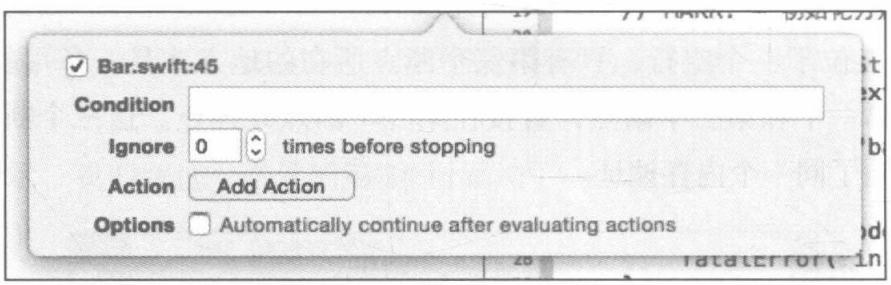


图 11-15 断点设置

11.4.3.1 激活 / 失效

断点编辑器的第一行显示的是该断点所处的文件名和行号，此外我们可以通过这个复选框来激活或者失效断点。在本例中，这个断点位于 Bar.swift 文件的第 45 行。

11.4.3.2 条件表达式（Condition）

条件表达式允许我们对断点触发设置条件，让断点在满足一定条件的时候，才暂停应用的执行。这里我们可以输入一些简单的条件，也可以输入一些在断点作用域内的方法。

当程序执行到该断点所在位置时，将会运行我们输入的条件表达式，当条件值为 YES、非零整数（Objective-C）或者 true(Swift) 时暂停程序执行。关于条件表达式的语法，只要输入文件对应的语言即可。


注意 条件不能够识别预处理宏，也不能识别断点作用域之外的变量和方法。

如果条件的输入不满足要求，那么这个条件就会被忽略掉，此外，在控制台处还会显示 Couldn't parse conditional expression 的消息来提示这个断点条件的问题。

11.4.3.3 忽略次数（Ignore）

忽略次数指定了在断点执行暂停应用运行之前，代码忽略断点的次数。如果希望应用要运行一段时间，那么就使用这个选项，尤其是在调试循环体的时候特别有用。在忽略次数

中输入一个非零整数，那么就可以激活该选项。将忽略次数设为 0，那么就可以恢复到默认情况。

11.4.3.4　动作（Action）

动作指定了当断点被触发后，Xcode 的行为动作。这个选项十分有用，它可以执行某些特定的脚本或者执行一些动作。

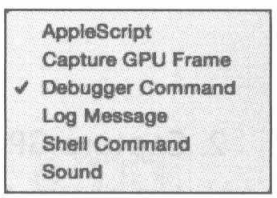


图 11-16　断点动作菜单

单击 Add Action 按钮，就会显示一条下拉菜单，标注了断点可执行的动作，如图 11-16 所示。

1. AppleScript

AppleScript 是苹果提供的一种脚本语言，用来执行一些预先指定的行为。选中 AppleScript 后，将会出现一个用来输入 AppleScript 脚本的输入框，如图 11-17 所示。

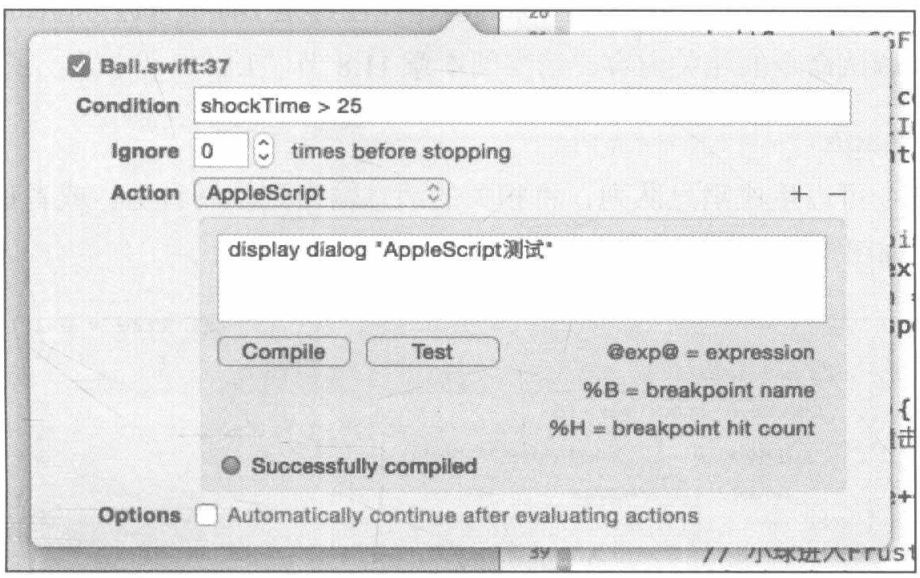


图 11-17　Apple Script

在图中，我们举了一个简单的 AppleScript 示例：

```
display dialog "AppleScript测试"
```

这条语句的意思是弹出一个以“ AppleScript 测试”为标题的对话框。Xcode 在动作窗口下方提示了一条信息：Press compile to verify your script（按下 compile 按钮来核查脚本）。点击 Compile（编译）按钮之后，Xcode 会自动检查我们输入的脚本语句，如果没有错误的话，就会提示如图 11-17 所示的成功信息：Successfully compiled（编译成功）。

单击 Test（测试）按钮，可以运行脚本，查看脚本运行效果，这里我们的脚本运行效果如 11-18 所示。

有关 AppleScript 的有关内容，不在本书的讨论内容当中，感兴趣的同学可以自行查看相应的文献。

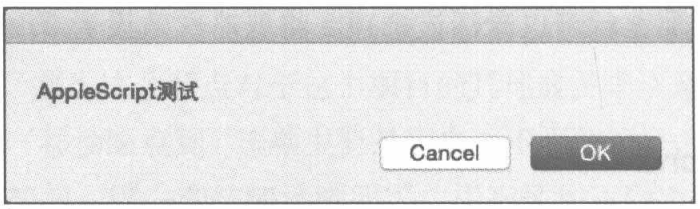


图 11-18　运行后的脚本

2. Capture GPU Frame

这个功能用于在执行到断点处，捕获 GPU 当前所绘制的帧。这个功能对于图形渲染调试十分有用，关于图形调试的相关内容，请参阅本章 11.9 节“视图调试”。

3. Debugger Command

Debugger Command（调试器命令）可以让断点执行绝大部分 LLDB 调试命令，这也是让断点动作最常用的部分，因此，我们新添加的动作往往都是 Debugger Command。

关于 LLDB 调试命令的相关内容，请参阅本章 11.8 节“LLDB 调试”。

4. Log Message

使用 Log 命令可以生成消息队列，将相关的消息输出到控制台上，或者让 Mac 语音播报所产生的消息，如图 11-19 所示。

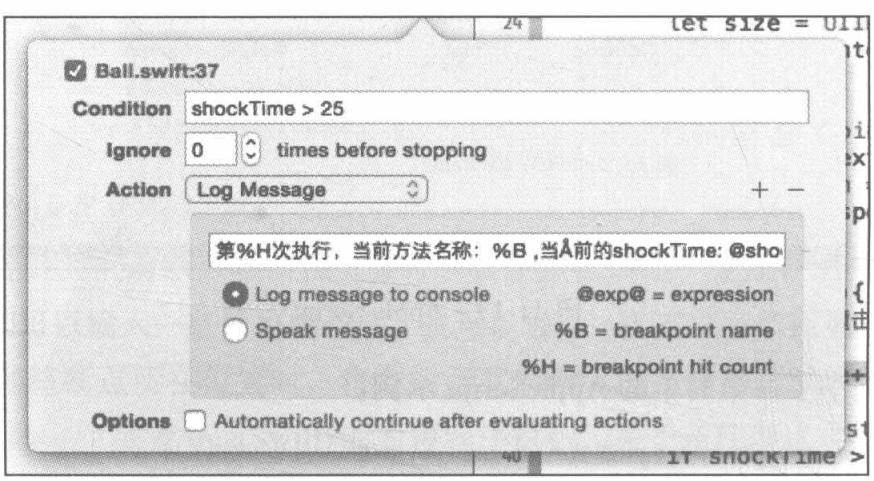


图 11-19　日志动作

消息中可以包含的特殊字符序列参见表 11-1。

表 11-1　特殊字符序列

消息标记	等价语义
%B	断点的名称
%H	遇到该断点的次数
@expression@	LLDB 表达式

在图 11-19 的示例中，输出的结果如图 11-20 所示，可以看到，相应的消息标记被转换成了等价的语义。

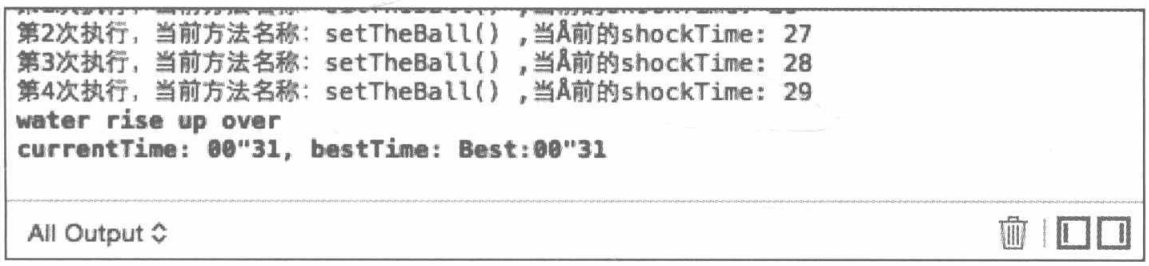


图 11-20　日志输出结果

5. Shell Command

Shell Command 动作要接受一个命令文件和一个参数列表，如图 11-21 所示。

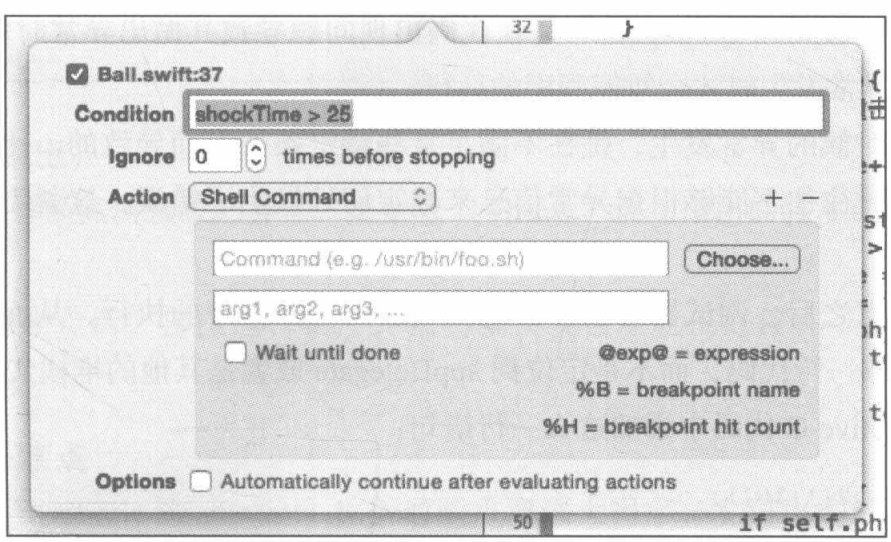


图 11-21　Shell Command

命令文件必须是一个可执行的二进制程序或者脚本。可以直接输入路径，或者可以单击 Choose 按钮来选择具体文件。输入的路径既可以是绝对路径，也可以是相对于项目的路径。

参数通过空格进行分隔，也可以在两个 @ 字符之间包含 LLDB 表达式。

一般情况下，Xcode 会异步执行 Shell Command，也就是说，Shell Command 和调试器将会同时运行。如果希望调试器在 Shell Command 完成后运行，则就要勾选 Wait Until Done（等待直到完成）选项。

6. Sound

Sound（声音）动作会在断点被触发的时候，弹出相应的声音。

11.4.3.5　选项（Options）

选项目前只有一条：Automatically continue after evaluating actions（在动作执行完毕后自

动继续)。这个功能简单明了，在遇到断点的时候，执行该断点当中的动作，然后不再暂停应用运行，而是继续执行。

为什么会有这么一个选项呢，这样子做岂不是让断点没了效果呢？其实不然，这个选项一般和“Log 消息”动作结合使用，这样就可以在调试模式下连续运行，在遇到断点时连续记录。

11.4.4 断点类型

除了之前介绍过的那种普通的断点外，断点还有四种类型，都可以通过匹配栏底部的“+”按钮添加。它们分别是：异常断点、符号断点、OpenGL ES 错误断点、测试失败断点，下面分别介绍。

11.4.4.1 异常断点

异常断点（Exception Breakpoint）是在代码出现问题导致其抛出异常时触发。换句话说，异常断点只有在异常发生时才会暂停程序的运行。

由于有一些时候的异常发生，是在不满足某些特定条件下而导致的，比如说在复杂循环中数组越界。有时往往不能够根据异常信息来确定何处发生了错误，这就需要异常断点来发挥作用了。

设置异常断点之后，调试器会在异常抛出的瞬间暂停程序的执行，从而可以让出错点定位到出现异常的那一行代码，而不是定位到 appDelegate 或者是其他的堆栈文件内。

我们向 Objective-C 代码中添加这么一行语句：

```
NSLog(@"%@", @[][10]);
```

运行后就会发现，应用停止在 main.m 文件当中，抛出异常的断点位于自动释放池中的 return 语句当中。这样我们如果不仔细阅读控制台或者日志里面的信息，我们是很难发现出错点在哪儿呢？

设置完异常断点之后，一切就都不一样了。点击断点导航器左下角的“+”添加按钮，然后在弹出的菜单当中，选择 Add Exception Breakpoint。随后在断点导航器中就会显示出我们刚刚添加的异常断点，如图 11-22 所示。

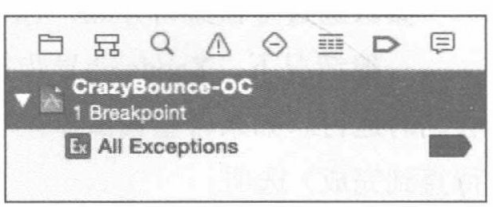


图 11-22 断点导航器中的异常断点

右键点击异常断点，然后选择 Edit Breakpoint，会弹出如图 11-23 所示的对话框，可以看到，和普通断点的设置有很大的不同。

1. Exception（异常）

异常指定的是，该断点要响应对象的类型，这个对象发出了抛出异常的操作。一般情况

下，我们可以选择响应 Objective-C 对象的抛出异常，也可以响应 C++ 对象的抛出异常，还可以两者都响应。

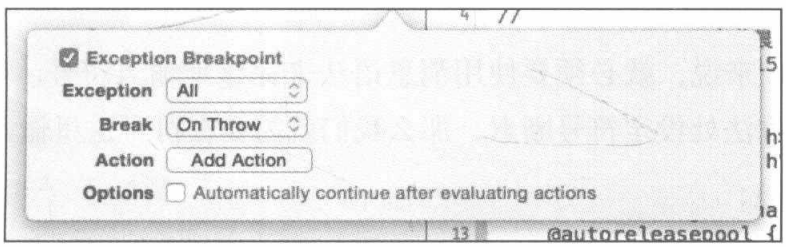


图 11-23　异常断点设置

对于响应 C++ 对象的抛出异常，我们还需要指定异常的名字，否则也是无法响应的。

提示　为什么没有 Swift 呢？其实 Swift 的断点可以比较精确地判定到出错的对象上，在出错的地方抛出异常，暂停应用。当然，对于 Cocoa 类的对象，我们也可以通过同样的设置来进行。

2. Break（跳出）

跳出则是选择断点所接收的异常，是接收“Throw”语句抛出的异常还是 Catch 语句捕获的异常。

添加完异常断点后，再次运行程序，我们就发现程序就正好停在了我们刚刚写的那条语句。

11.4.4.2　符号断点

符号断点（Symbolic Breakpoint）比普通断点要强大得多，它可以中断某个方法的调用，只要出现这个方法调用的地方，都会引发断点触发，从而导致程序暂停运行。

和添加异常断点类似，单击“+”按钮后，选择 Add Symbolic Breakpoint，然后 Xcode 就会弹出如图 11-24 所示的对话框，可以看到，和普通断点的设置有很大的不同。

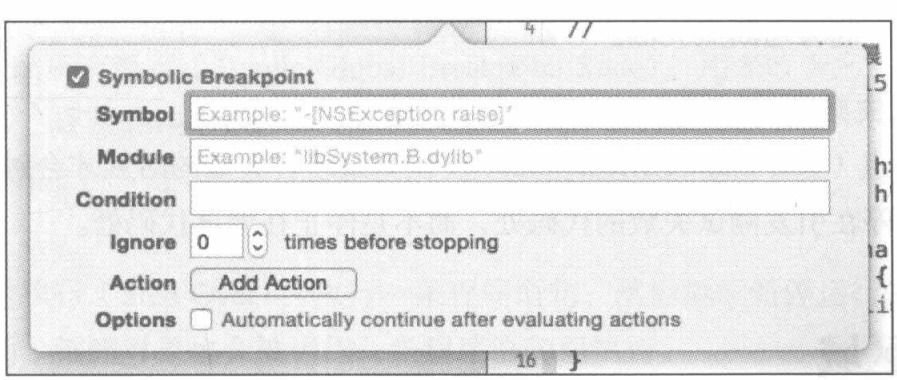


图 11-24　符号断点

1. Symbol（符号）

符号指的是当前断点作用域所能够识别的方法，这既可以是应用程序中的方法，也可以是系统 API 的方法。

对于 Objective-C 来说，就必须要使用消息语法来完整地输入符号。比如说，我们要想在 Ball 类的 knockBall 方法处设定符号断点，那么我们就需要在符号这里输入：

```
-[Ball knockBall]
```

注意，符号处必须要有一个 + 号或者 - 号来指明类方法或者成员方法，在这里键入的参数就如同在 @Selector 当中键入一样，knockBall 和 knockBall: 是两个不同的方法。对于 Swift 来说，直接输入方法名即可：

```
knockBall
```


和 Objective-C 不同的是，Swift 的符号断点会在运行之后，将所有能够触发符号断点的位置显示出来，如图 11-25 所示。似乎，Swift 并不支持带参数的方法名称，因此 func knockBall() 和 func knockBall(argument: Int) 都会被同时暂停。

2. Module（模组）

模组用来限制满足符号的方法，Xcode 将只会在断点满足这个模组的符号的时候才会暂停。比如，假设我们使用了第三方库，这个第三方库拥有 viewDidLoad 函数，如果我们只打算关注第三方库的 viewDidLoad 函数，那么我们在这里输入第三方库的库名称即可。

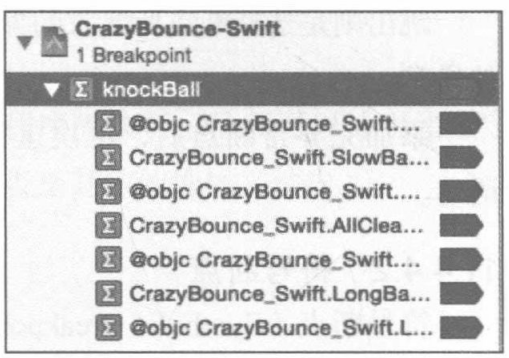


图 11-25　Swift 的符号断点

11.4.4.3　OpenGL ES 错误断点

OpenGL ES 错误断点（OpenGL ES Error Breakpoint）的作用和异常断点很类似，只不过这个断点只有在 OpenGL ES 错误发生的时候才会触发。添加这个断点的做法和异常断点类似，显示出来的对话框和普通断点类似。

11.4.4.4　测试失败断点

测试失败断点（Test Failure Breakpoint）仅在测试断言失败的时候才会触发执行，这个时候，应用将会暂停在引发测试失败的代码处，而不是停止在测试代码处。

11.5　调试区域

对于运行时调试，Xcode 现在集成了 LLDB 调试器，之前的 GDB 调试器已经被禁用。默

认情况下，所有新创建的 Xcode 项目在调试器中运行。当运行的应用程序与调试器连接时，调试区域将会出现，如图 11-26 所示。

图 11-26　调试区域

调试区域由三部分组成，分别是调试工具栏、变量视图、控制器。

11.5.1　调试工具栏

调试区域工具栏位于最上面，有很多控钮，主要是用来控制应用运行，并且还可以查看应用的线程信息。对于 iOS 应用来说，还可以进行地理位置的模拟，以及捕获 OpenGL ES 帧画面。

这些按钮从左到右分别完成以下功能：

- 隐藏 / 显示调试区域（Hide/Show the Dubeg Area）：用来显示或者隐藏这个调试区域，此外，也可以通过工具栏上的“显示 / 隐藏调试区域”按钮来进行控制。
- 启动 / 关闭所有断点（Toggle Global Breakpoint State）：用来改变所有断点的状态，是启用断点，还是禁用断点。
- 继续 / 暂停运行进程（Continue/Pause）：控制应用进程的执行状态，是继续执行还是暂停执行。
- 单步跳过执行（Step Over）：执行一条程序语句，然后继续暂停运行进程。注意的是，函数调用、创建对象也会被当作一条程序语句而执行。
- 单步跳入执行（Step Into）：和“单步跳过执行”的作用相似，不过这个操作在遇到函数调用、对象创建时，会跳转到相应的函数内部、类初始化方法内部执行。

- 跳出当前函数（Step Out）：执行完断点所在函数剩余的语句，然后跳出当前函数。
- 调试视图层次（Debug View Hierarchy）：打开视图调试功能，有关视图调试的相关内容，请查阅本章 11.10 视图调试一节。
- 地理位置模拟（Simulate location）：模拟一个地理位置信息。
- OpenGL ES 帧捕获（OpenGL ES Frame Capture）：使用了 OpenGL ES 框架的 iOS 应用才能使用此项，其用于捕获当前运行在 iOS 设备上的当前应用完整快照，能够捕获阴影信息以及一些虚拟架构。

按住 Control 或者 Contril+Shift 键，同时单击“单步跳过执行”和“单步跳入执行”按钮，可以执行不同的单步执行功能。主要是在指令（instruction）和线程（threads）级别上的运行。

11.5.2 变量视图

变量视图用来显示当前程序暂停处，存放在堆栈中的变量和寄存器信息。这些信息将会以字典的形式展现出来，单击变量左边的三角形可以展开或者合上变量的相关信息。

变量视图只会在暂停调试过程中显示，导致调试过程暂停的原因无不外乎于碰见一个断点或者程序在某处崩溃。

变量视图中的一条变量信息由三个部分组成：变量名、变量类型和总结信息。总结信息就是简要地对这个变量描述的一个总结，在图 11-26 中显示的是变量的内存地址信息。

此外，我们可以根据需要选择显示的变量内容，在变量视图的左下角，单击 Auto 按钮，就会弹出一个菜单栏，如图 11-27 所示。选项说明如下。

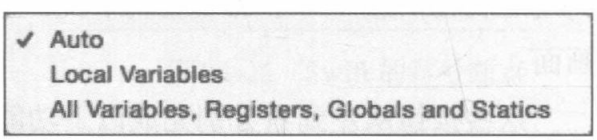


图 11-27 变量视图筛选

- Auto：在变量视图中显示与当前作用域相关的变量。
- Local Variables：在变量视图中显示当前作用域下的局部变量。
- All Variables, Registers, Globals and Statics：输出所有变量、寄存器值、全局变量和静态变量。

Auto 按钮旁边有两个按钮，如图 11-26 左下角所示。小眼睛样式的是“快速查看（Quick Look）”按钮，旁边是“打印描述信息（print Description）”按钮。有关这两个按钮的介绍，请参阅本章 11.7 节“快速查看”。

此外，通过变量视图右下角的过滤器，可以对纷繁复杂的变量和寄存器信息进行过滤。

11.5.3　控制台

控制台用于输出应用打印的内容（包括 printf、NSLog、println 等输出指令，以及错误信息等内容），以及执行 LLDB 调试指令。

调试指令只有在应用暂停并出于调试状态时才可用，可用时，会在控制台区域显示“(lldb)”的标识，然后就可以在其右侧输入调试器命令。

输出可以控制台区域的右下角进行过滤选择，默认是显示全部输出（All Output），还可以选择调试器输出（Debugger Output）和对象输出（Target Output）。

通过右下角的垃圾箱图标，可以清空控制台的内容。然后右边的两个按钮用来控制变量区域和控制台区域的显示。

11.5.4　查看线程

每调用一个方法或函数，调试器将会把这些信息存储在栈帧（stack frame）当中，而这些栈帧则是存储在堆栈里面。

一旦应用暂停运行，这时候就可以查看应用的线程和堆栈信息。让应用暂停的方法，前面已经介绍过，包括使用调试工具栏上的“继续 / 暂停运行进程”按钮，以及添加断点的方法。

通过线程和堆栈导航栏，可以选择应用的各种线程。单击线程信息，这时候会弹出一个弹出菜单，从中就可以对相应的线程和堆栈信息进行选择，如图 11-28 所示。

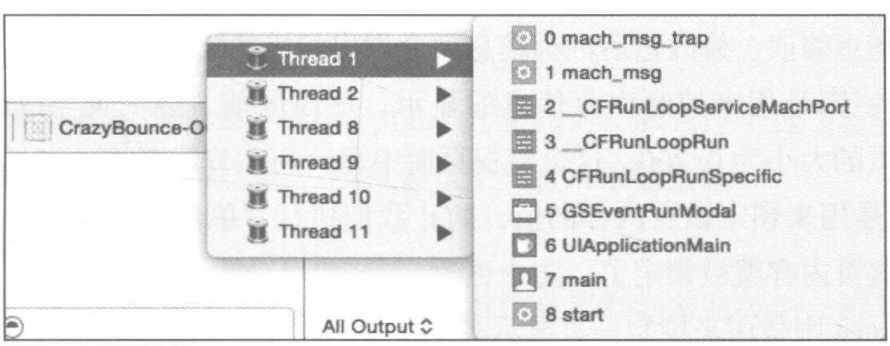


图 11-28　查看线程

当我们选中了一个线程或者堆栈后，Xcode 就会在代码编辑器中，显示其对应的源文件或者汇编代码。

关于 iOS 线程和堆栈相关内容，不在本书的介绍范围之内。

11.5.5　查看内存信息

对于一些好奇心极重的同学来说，Xcode 提供了传统 IDE 都拥有的变量内存查看机制。右键单击某个想要查看内存的变量，然后从弹出的菜单中选择 View Memory of 变量名。这时

候，Xcode 就会在主编辑器中打开该内存的信息，如图 11-29 所示。

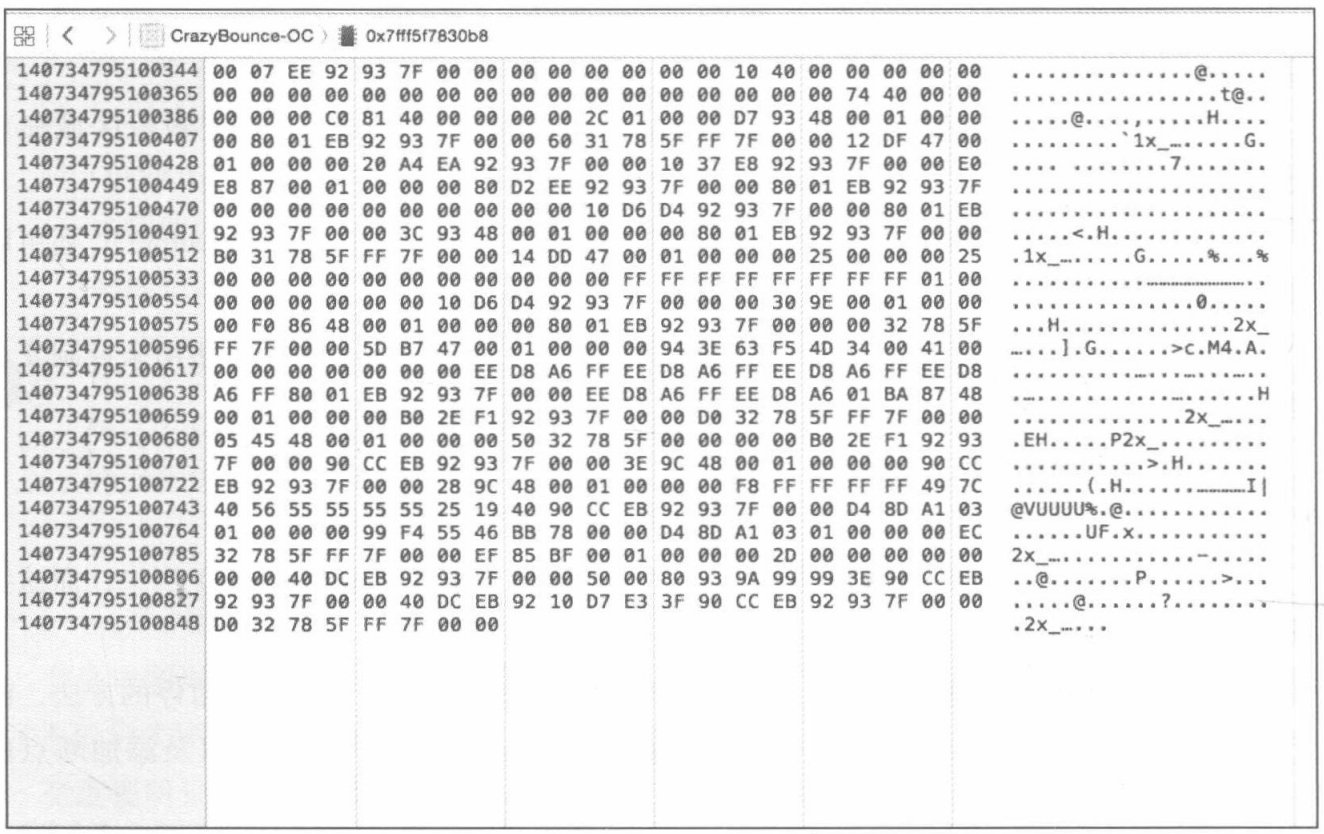


图 11-29 查看内存信息

在内存信息的下方，有一条控制栏。Address（地址）可以用来控制显示当前显示的内存地址信息，可以随意修改，修改过后内存信息就会做出相应的改变。

Page 选择按钮则是用来控制内存信息的显示，可以向前“翻一页”或向后“翻一页”。默认情况下，一页的大小为 0x200，这个值根页面中显示的字节数有关。

Lock 按钮则是用来锁定该页内存的值，防止我们执行“单步操作”后，内存的值发生改变。这个时候，这页内存就被锁定了，防止更改。

Number of Bytes 则是用来控制一页显示多少字节的信息，默认是 511 字节。

Swift 并不支持查看内存信息功能，这似乎跟 Swift 的类型安全有关。

11.5.6 模拟位置

通过使用调试工具栏上的“地理位置模拟”按钮，可以在应用运行时模拟一个地理位置信息。

单击这个按钮，这时候 Xcode 就会弹出如图 11-30 所示的菜单。从这个菜单中，我们可以选择一些默认的国家和地区，比如说中国香港（Hong Kong, China）、英国伦敦（London,

England）等等。

此外，我们还可以往项目中添加一个自定义的 GPX 文件，这样就可以使用这个 GPX 文件的地理位置信息来进行地理位置的模拟了。有关 GPX 文件的相关内容，请参阅本书附录 C 中的 C.1 节“文件模板”。

11.5.7　变量设置

在查看内存信息的时候，我们右键点击了变量视图中的变量，在弹出的菜单中，发现了许多其他的选项，如图 11-31 所示。在此我们简要对其进行介绍。

图 11-30　地理位置模拟

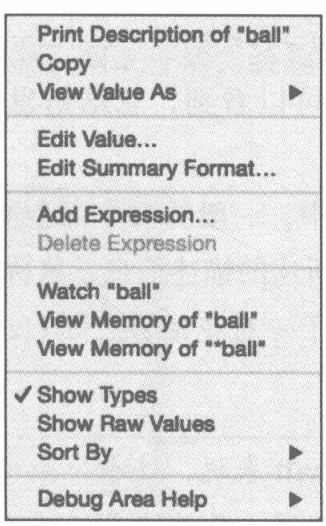


图 11-31　变量视图菜单

1. 打印变量名的描述信息（Print Description of ...）

如果这个变量中有 +(NSString*)description; 方法的话，那么这个选项将会调用这个方法，在控制台中打印这个方法的输出信息，也就是这个变量的描述信息。

2. 复制（Copy）

复制选项将会将该变量的变量名、类名、地址信息复制到剪贴板中，如下所示：

```
ball    Ball *  0x7f9392ee0700  0x00007f9392ee0700
```

3. 以……类型格式查看变量值（View Value As ...）

这个选项将可以控制是以什么类型来查看选中的变量，默认是 Default 默认类型，Xcode 自动推断这个变量，然后选择一个最合适的类型来进行显示。此外，也可以自行选择诸如 Binary（二进制）、Boolean（布尔值）等等类型来显示。甚至还可以自定义一个类型。

选择一个类型之后，Xcode 将会尝试使用该类型来转换所选定的变量，这对于 id 类型或者 AnyObject 类型的变量尤为有用，可以将其转换为特定的类型。

4. 编辑变量值（Edit Value）

这个选项将允许开发者对变量窗口内的变量进行修改，包括变量地址、变量值等等内容。

注意 出于内存保护等原因，Swift 不允许对变量值进行修改。

5. 编辑总结格式（Edit Summary Format）

在之前我们提到过，变量信息由三个部分组成，分别是变量名，变量类型和总结信息。这个选项将允许我们修改变量的总结格式信息。

比如说我们给图 11-26 中的 self 变量添加如下的描述信息：

```
{(NSString*)[$VAR description]}:s
```

之后我们可以看到，总结信息由之前的内存地址信息，变成了这个方法的 description 信息。

总结格式中，一般的文本都会被原封不动地显示出来，要想输入特殊字符，那么就要输入类似于上面所述的描述信息，这称之为：表达式。

表达式介于一对花括号之间（{expression}），其中可以使用 $VAR 宏，这个宏会被替换为变量的名称。

注意 对于 Swift 来说，这个命令似乎是无效的，会提示 Summary Unavaliable（总结信息无效）。不过，总结格式仍然可以输入字符串。关于如何使用更为高效的命令，这超出了笔者的学识范围，并且也没有找到相关资料。

6. 添加和删除表达式（Add/Delete Expression）

添加表达式可以在变量窗口中添加新的变量表达式，只需要输入断点当前作用域能够识别的语句形式即可。比如说对于图 11-26 来说，只需要输入：

```
[self description];
```

即可完成一个新的变量的添加，这个变量将会以这个表达式命名，然后其总结信息就是 description 的信息。然后使用“删除表达式”选项，即可对这个表达式进行删除。

7. 观察点（Watchpoint）

观察点相当于一种断点，我们可以将任意变量当中观察点，如果调试器检测到变量发生了变化，那么就会立刻暂停应用运行。虽然这听起来挺有用的，但是实际上观察点不能够观察被观察变量之外的代码。也就是说，观察点通常用于观察全局变量，否则其并不是那么好用。

观察点只能够观察变量，其他的量都无法被设置观察点。给一个变量设置了观察点之后，

在断点导航器中就会显示相应的观察点，如图 11-32 所示。

继续运行应用，Xcode 就会在变量作用域范围内，当变量发生变化的时候暂停应用。要注意的是，当应用停止运行之后，观察点便会失效，这时候需要手动删除重新添加。

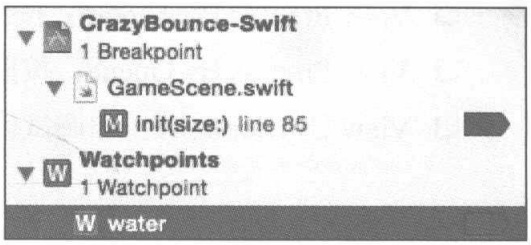


图 11-32　观察点

11.6　调试导航器

调试导航器（Debug navigator）用来显示应用程序在调试状态下的资源占用状态以及堆栈信息，如图 11-33 所示。调试导航器包括：过程视图选择器、调试仪器、线程和内存位置列表、匹配栏。

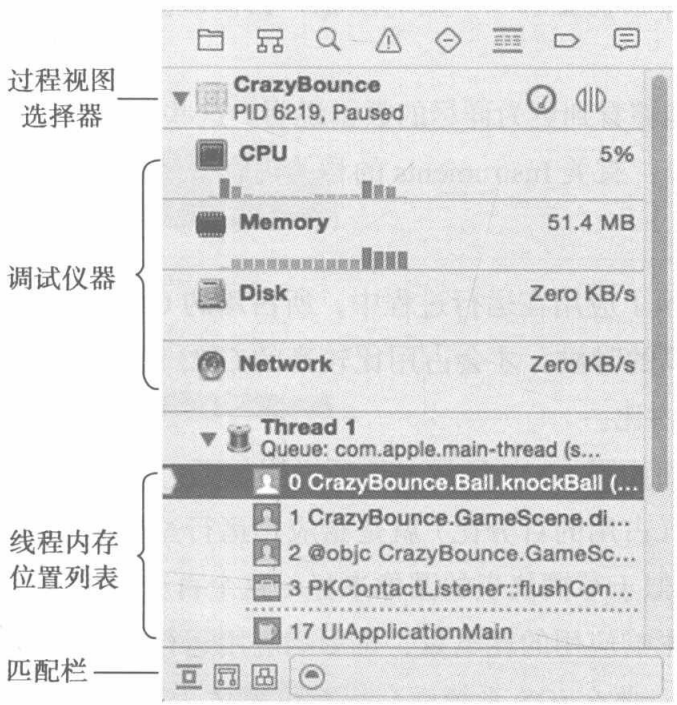


图 11-33　调试导航器

过程视图选择器（process view selector）显示了当前运行应用的名称，然后还显示了当前该应用的 PID，即主进程号。此外，还显示了当前该应用是处于运行状态（Running）还是暂停状态（Paused）。

此外，过程视图选择器还拥有两个选项。第一个选项是用来控制是否显示“调试仪器”界面的，默认情况下这个选项是选中的。第二个选项是控制过程视图的显示方式，这个主要影响“线程和内存位置列表”的显示方式，如图 11-34 所示，选项说明

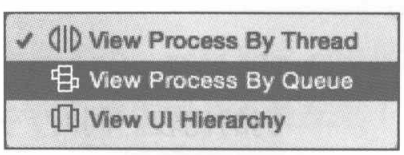


图 11-34　视图显示选项

如下：

- View Process By Thread：按线程显示。
- View Process By Queue：按照调度队列将线程进行分组。
- View UI Hierarchy：和调试区域的“调试视图层次”相同，打开视图调试界面。

11.6.1 调试仪器

调试仪器（debug gauges）显示了当前运行过程中，所占用的CPU百分比、内存大小（Memory）、硬盘读取速度（Disk），以及网络读取速度（Network）等数据。在OS X程序当中，还会显示能量消耗（Energy Impact）数据。使用了OpenGL ES或者Medal等图形框架的应用程序还会显示GPU数据。使用了iCloud功能的应用程序还会显示iCloud数据。

通过调试仪器，我们可以看到应用在运行过程中的资源消耗情况。如果某些资源消耗过高，或者不符合要求，那么就要采取一定的措施，去修改实现机制，尽可能将资源消耗放到最低。

点击这些数据，就能够看到更为详尽的数据说明。这些调试仪器都可以前往Instruments来进行更加复杂的分析操作，有关Instruments的相关内容，请查看本章11.10节“Instruments”。

11.6.1.1 CPU百分比

CPU百分比重点显示了应用在运行过程中，所占用的CPU的比率。一般来说，只有执行某些大规模的操作和运算的时候，才会占用比较高的CPU比率。运转流畅的应用应当在静默运行时保持较低的CPU占比。

CPU百分比详细界面如图11-35所示。选项说明如下：

- Percentage Used（占用的百分比）就是指应用运行过程中，所占用的CPU比率。这里以一个直观的类似于车辆仪表的图像来显示这个百分比值。

CPU占比会极大地影响应用的耗电量，也影响应用运行的流畅程度。此外，要注意的是，如果CPU持续占用过高，那么iOS系统就会很有可能认为其进程已经卡死，从而将其强制关闭，以保护设备。

注意 这里我们用的是iOS模拟器来运行的程序，因此CPU的最大可用比例会十分高，达到了400%。如果我们换用真机，那么大家就会发现不同了。同样地，由于Mac的不同，CPU的最大可用比例也会不同。

- Usage Comparison（占用比较）指的是当前系统中，当前的进程和其他进程的占用比例的比较。同时，这里还会显示空闲CPU的比例。通过这个栏目，我们可以看到当前该进程在整个系统中消耗CPU计算能力的占比。
- Usage over Time（CPU使用时间轴）则显示了一个以时间为横轴，以CPU占比为纵轴

的折线图。这个折线图则显示了随着时间的推移，CPU 占用比的变化情况。在这个视图的左边，显示了当前运行的总时间（40 sec），以及在这段时间内所消耗的 CPU 最大比例（25%）以及最小比例（2%）。

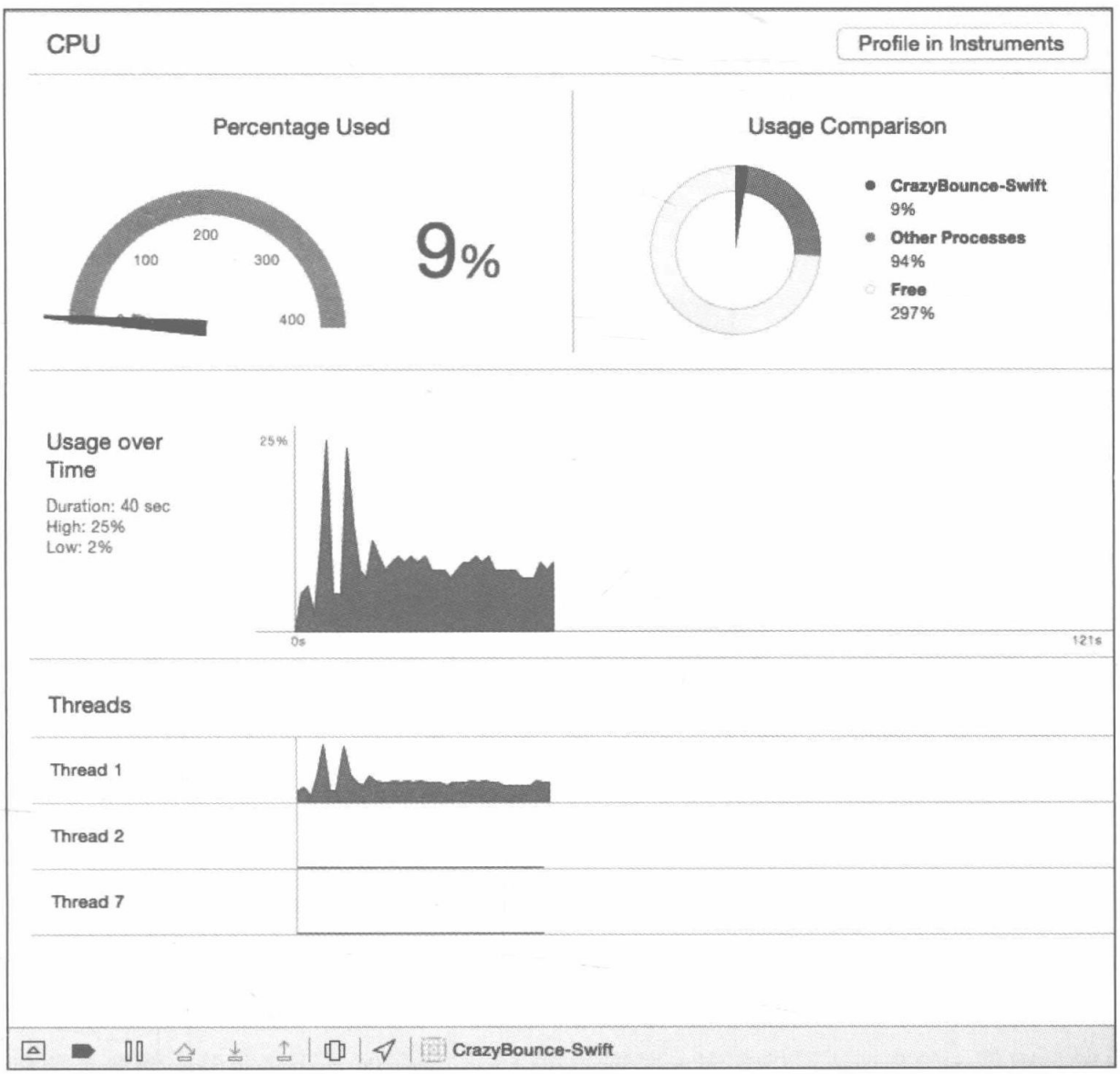


图 11-35　CPU 调试仪器界面

- Threads（线程）则显示了不同线程对 CPU 的占用成都，相当于 CPU 使用时间轴的分线程版本。

11.6.1.2　内存大小

内存大小重点显示了当前应用在运行过程中，所占用的设备内存大小。一般来说，内存的占用应该有升有降，如果你发现你的内存占用在持续增加，那么就要考虑到是不是发生了内存泄露问题，比如说循环强引用等。

内存大小详细界面如图 11-36 所示，选项说明如下：

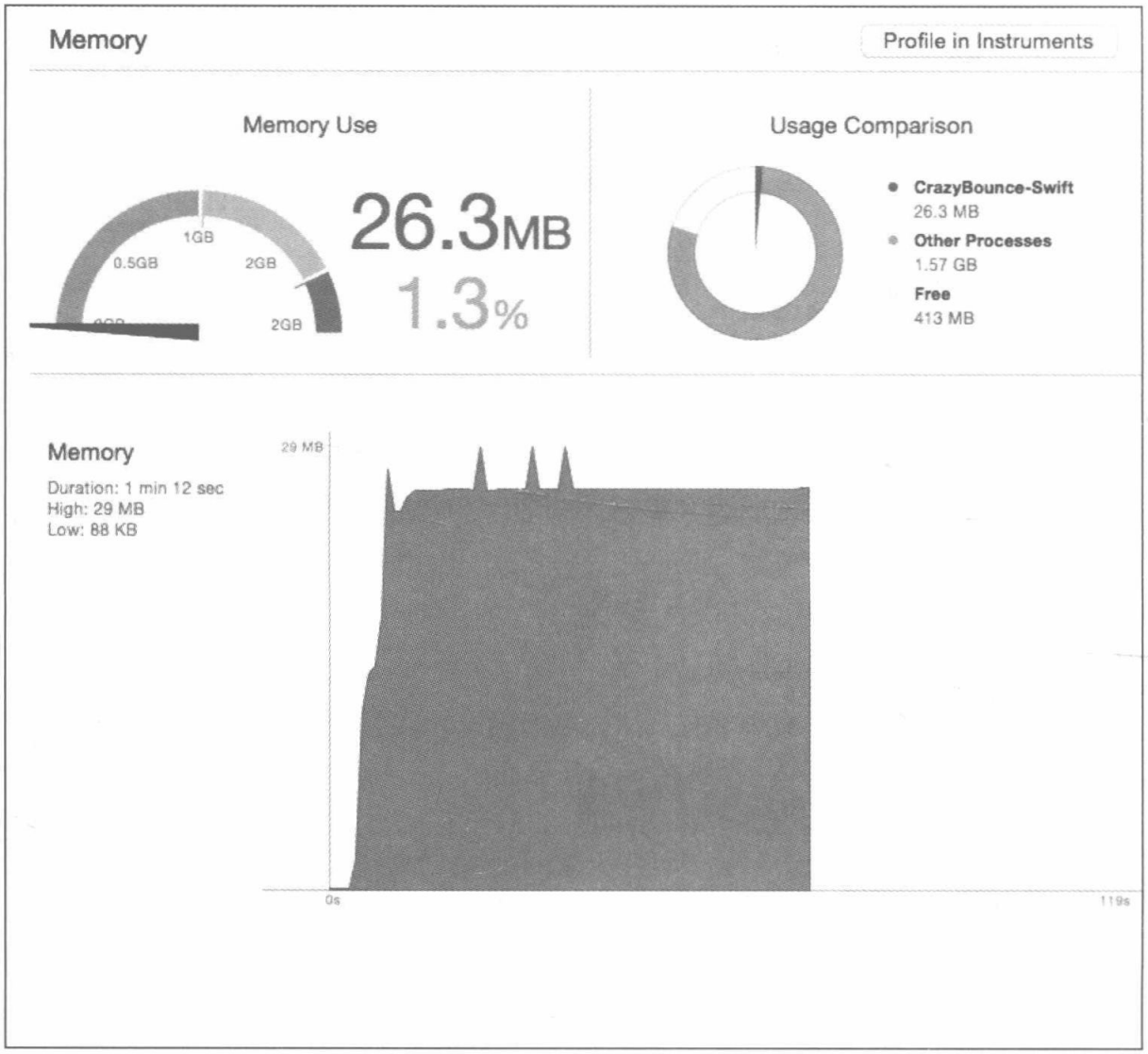


图 11-36 内存调试仪器界面

- Memory Use（内存使用）就是指应用运行过程中，所占用的内存大小，以及占用的内存比例。

注意 和 CPU 一样，这里内存的显示大小仍然是因为 iOS 模拟器的原因，这里的上限 2GB 是笔者的 Mac 的内存大小。

- Usage Comparison 和 CPU 的一样，在此我们就不多加介绍。
- Memory（内存）和 CPU 的占用比较类似，根据时间来展示内存的使用比例。

11.6.1.3 硬盘读取速度

硬盘读取速度重点显示了当前应用在运行过程中，进行读写操作的速度情况。一般来说，应用应该在需要的时候进行数据的读写，一直持续进行数据的读写会极大地占用 CPU 和内存，导致应用耗能大、卡顿现象严重，同时也可能影响其他正常的读写操作进行。

硬盘读取速度详细界面如图 11-37 所示，选项说明如下：

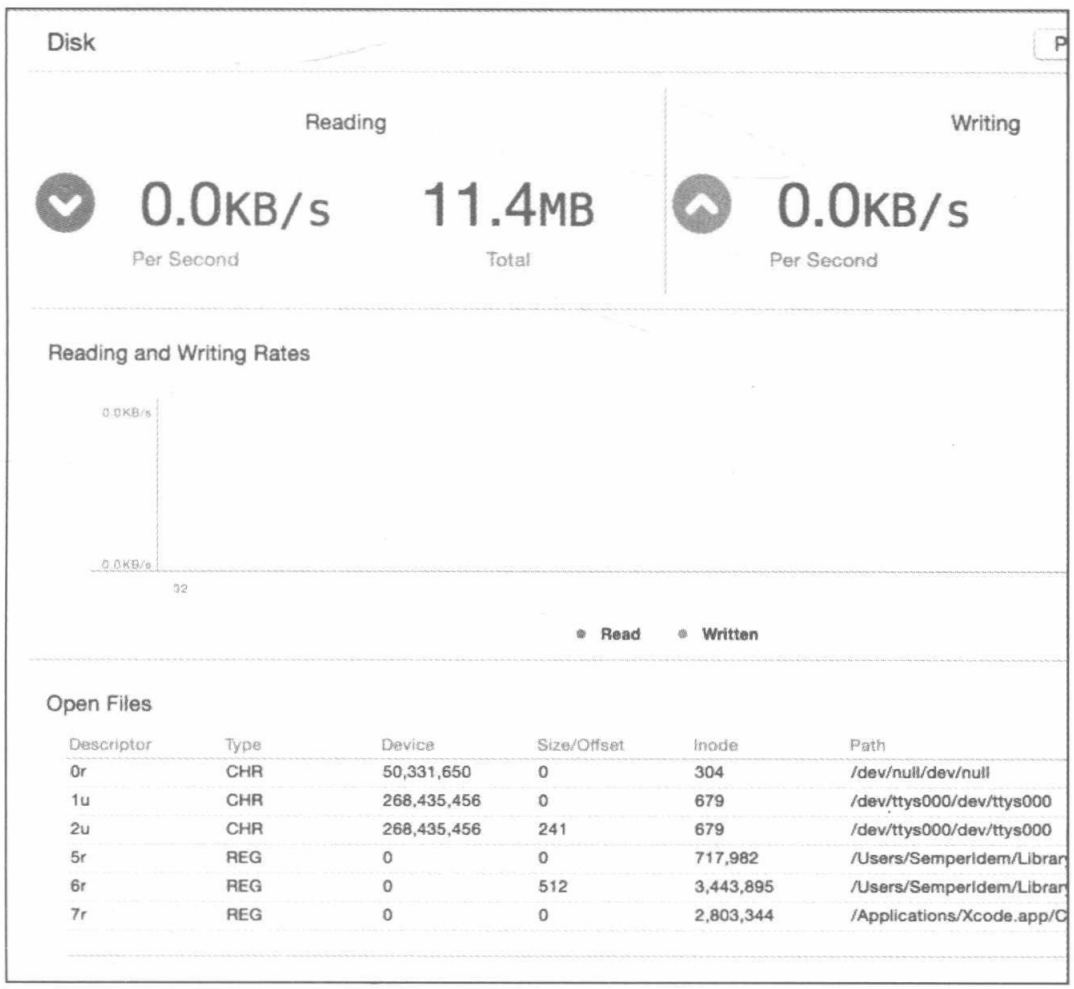


图 11-37　硬盘调试仪器界面

- Reading（读取速度）则显示了当前应用对硬盘的读取速度，还有其总共读取了多少数据。
- Writng（写入速度）则显示了当前应用对硬盘的写入速度。

这两个速度一般情况下，和采用的读写方法有关。读取和写入速度的上限则是用户设备所限制的。

- Reading and Writing Rates（读写速度比率）则和 CPU 的占用比较类似，根据事件来展示硬盘的读写速度对比。一般以蓝色线条来显示读取速度，以绿色线条来显示写入速度。
- Open Files（打开的文件）则显示了应用运行过程中，对哪些文件进行了操作。同时还显示了打开文件的描述符（Descriptor）、文件类型（Type）、设备（Device）、大小 / 偏移量（Size/Offset）、文件名所对应的 inode 代码，以及该文件所在路径（Path）。

11.6.1.4　网络读取速度

和硬盘读取速度类似，网络速度显示了在运行过程中，应用进行网络发送、接收操作的

速度情况。一般来说，应用应该在需要的时候才进行网络读取，否则会消耗用户太多的流量，导致体验不佳等情况发生。

网络读取速度详细界面如图 11-38 所示，选项说明如下：

图 11-38　网络调试仪器界面

- Receiving 和 Sending 类似于硬盘读取速度的 Reading 和 Writing，Reveiving 和 Sending Rates 也类似于硬盘读取速度的 Reading and Writing Rates。一般以紫色线条来显示接收速度，橘黄色线条显示发送速度。
- Active Connections（活动的连接）一栏则展示了当前应用都建立了哪些网络链接，从而可以校验网络链接的正确性与否。这个表中显示了链接协议（Protocol）、本地地址（Local Address）、本地端口（Local Port）、远程地址（Remote Address）、远程端口（Remote Port）、用户界面（interface）、链接状态（State），以及字节（Bytes）和包）Packect_ 信息。

11.6.1.5　能量消耗

能量消耗重点显示了该应用对电池的消耗情况。能量消耗的因素跟许多因素有关，良好的能量消耗比例能够增强用户体验。在 Xcode 6 及其以前，能量消耗只能够在 Mac 应用

中显示。

提示　在最新的 Xcode 7 中，iOS 应用也能够显示能量消耗了。

能量消耗详细界面如图 11-39 所示，选项说明如下：

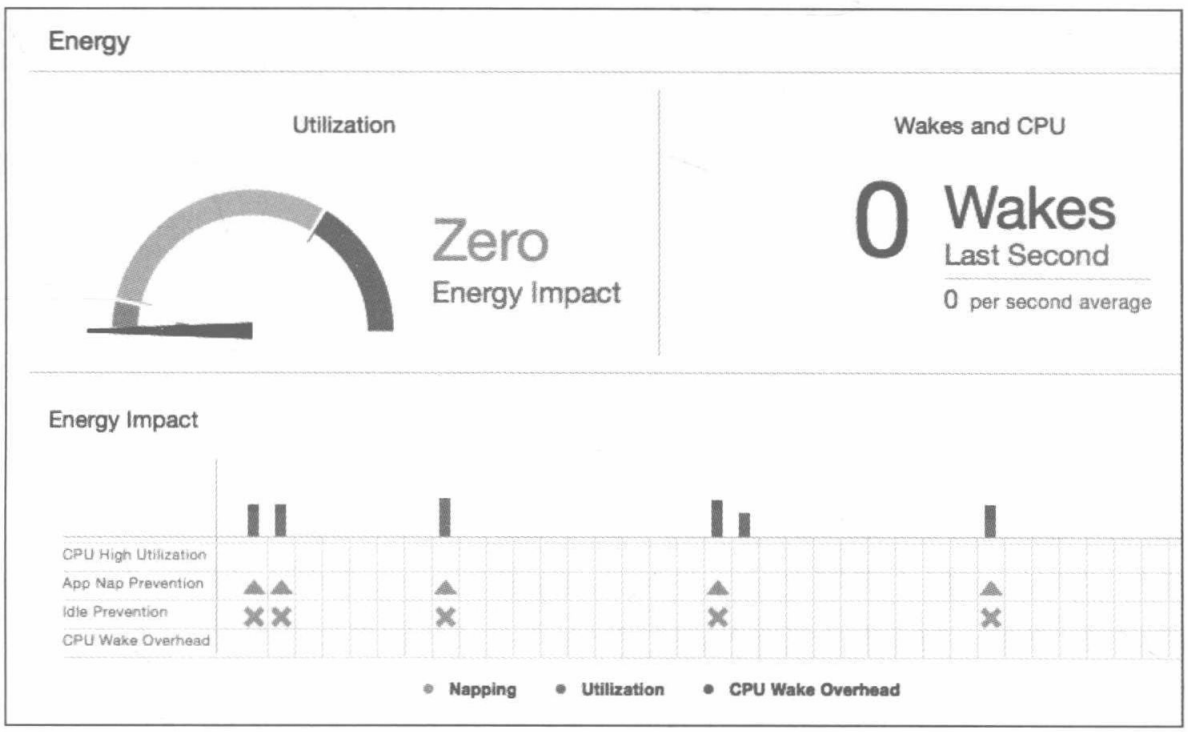


图 11-39　能量调试仪器界面

- Utilization（效用）显示了当前应用所利用能量的效用比率。绿色则表示“低能耗”、红色则表示“高能耗”。

关于能量消耗，我们不得不先说明一下四个概念：

- App Nap（应用小憩功能）

App Nap 是 OS X Mavericks 的一项新功能。它能在同时运行多个应用程序时节省电能。当一个应用程序在后台最小化运行时，OS X 将会把硬盘和 CPU 的优先级转移到前台应用来，这样可以极大地提升应用速度。这个功能将会以一个“对勾”符号（✓）显示出来。

- Idle Prevention（防止应用空转）

增加 CPU 的空转时间是 OS X 提高能源效率的重要组成部分之一，因此，在应用未使用 CPU 时，CPU 将会进入对到空转状态下。计数器（Timers）可以有效地防止 CPU 空转，计数器在其他计数器可用时暂停使用，因为计数器只能有一个进行。这个功能将会以一个“叉”符号（✗）显示出来。

- CPU Wake Overhead（过高的 CPU 唤醒次数）

每当 CPU 从空转状态中被唤醒的时候，会消耗极大的能源。如果唤醒次数过多，并且

CPU 占用率过低的话，那么你应当考虑进行批处理，以减小唤醒次数。这个功能将会以一个“三角”符号（▲）显示出来。

❑ High CPU Utilization（高 CPU 占用）

极高的 CPU 占用率会极快地消耗掉笔记本的电量，出现这个标识意味着 CPU 占用率达到了 20% 以上。这个功能将会以一个“闪电”符号（↯）显示出来。

现在我们可以来理解 Energy Impact（能量消耗）了，这个“能量消耗”图是一个随时间变化的过程图。最上方的蓝红色相间的条形图，表示的是 CPU 占用率和 CPU 唤醒次数的比例。下方的一串绿色条，则代表着应用正在处于小憩状态，如果应用被唤醒，那么绿色条就会消失。

然后再往下分别有四个格子，这些格子就代表了上述四个功能。应用在前台运行过程中，会时不时阻止“小憩”功能的发生，同时一般也会阻止 CPU 空转。

如果频繁出现了高 CPU 占用和过高的 CPU 唤醒次数的提示，那么就要注意了，这代表了该应用有过高能源消耗的问题。

11.6.1.6 FPS

FPS 调试仪器（GPU）只有在运行 OpenGL ES 或者 Metal 程序的时候，才能够显示在调试导航栏里面。此外，FPS 也只能够在真机调试的时候才能够显示出来。

FPS 详细界面如图 11-40 所示：

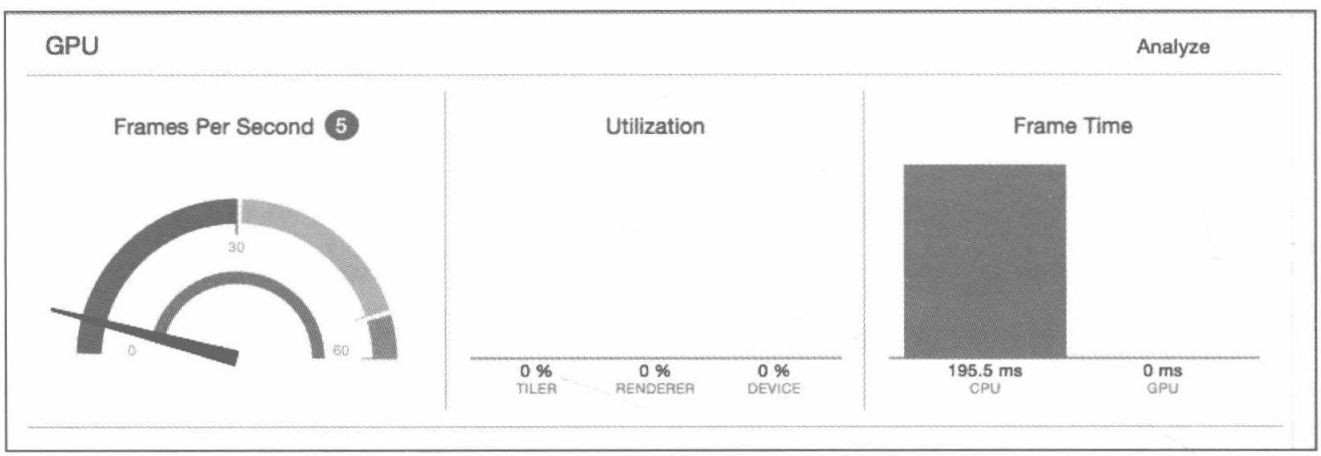


图 11-40 FPS 调试仪器界面

Frames Per Second（每秒帧数）则是平常见得最多的一个关于 FPS 的描述，每秒帧数越低，那么这个应用的体验就越差。可以看到，0 ～ 30 的 FPS 是以红色进行标注的，代表了极差的性能；30 ～ 50 左右则是以黄色进行标注的，代表了一般的性能；而一般的 FPS 值都是稳定在 60 左右，这也是最佳的性能。

Utilization（占用率），占用率有三个值，分别是 Tiler、Renderer 和 Device。这三个单词是 FPS 相关的术语，有着特定的语义。

Tiler 占用率主要是监视应用中的图层数量，如果这个值超过了 50%，那么就意味着动画由于几何结构太复杂，导致运算缓慢，这通常就是因为屏幕上有太多的图层导致的。

Renderer 占用率主要是监视应用中的渲染性能，如果这个值超过了 50%，那么就意味着动画由于进行了离屏渲染、过度混合重绘等因素导致了帧率受限等原因，从而导致 FPS 下降。

Device 占用率主要是监视设备的性能，如果这个值超过了 50%，就代表当前的渲染操作超过了设备 GPU 的绘制能力，而转用 CPU 帮助绘制，这导致应用帧率大幅下滑。

Frame Time（帧绘制时间）则是代表 CPU 或者 GPU 绘制一个帧所消耗的平均时间，从这里可以看出 FPS 受限的主要原因。

11.6.1.7　iCloud

iCloud 调试仪器只有在开启 iCloud 功能的时候，才能够显示在调试导航栏里面。此外，iCloud 也只能够在真机调试的时候才能够显示出来。

iCloud 详细界面如图 11-41 所示，选项说明如下：

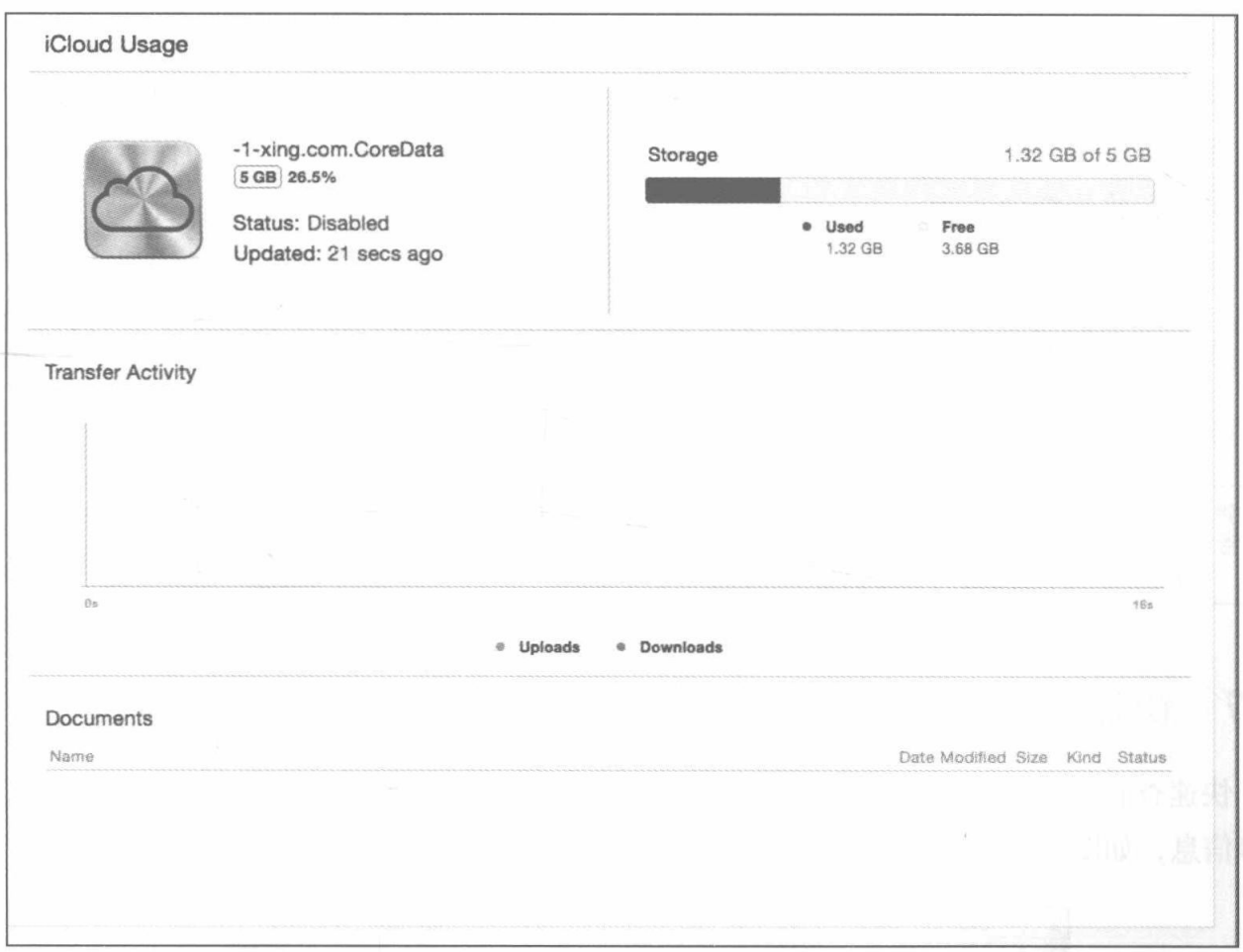


图 11-41　iCloud 调试仪器界面

- iCloud Usage（iCloud 使用量）显示了当前正在使用的 iCloud 存储量的信息，包括账

号、存储大小、当前使用容量、iCloud 状态、上次更新时间等等。

- Transfer Activity（转移活动）主要显示了 iCloud 上传和下载的数据量，上传以绿色显示，下载以蓝色显示。
- Documents（文档）则是显示了当前应用存储在 iCloud 的所有文档。

11.6.2 线程和队列

线程和内存位置列表（thread and memory location list）和“调试仪器”当中的相似，但是其更为详细，更为方便查看。

匹配栏位于导航器底部（见图 11-33），可以让开发者过滤线程和内存信息位置列表中的元素。

左边第一个按钮，用来控制是否只显示“存在于调试符号以及库之间的栈帧”对象，关于栈帧（stack frame）的概念，限于篇幅和主题限制不再多提，简要来说，它是一个数据结构。

第二个按钮，用来控制是否只显示“引发线程崩溃或者与调试符号相关的线程”对象。开启这个选项，就可以帮助开发者更好地关注出错的地方，从而更快地修正错误。

第三个按钮，用来控制当处于“队列显示”方式时，是否只显示“正在运行的代码块”。

此外，Xcde 还可以暂停一个线程，以防止其在调试过程中运行。使用暂停线程功能可以暂停某个即将崩溃的线程，或者暂停某个会干预应用剩余进程的线程。要想暂停线程，可右键单击某个想暂停的线程，然后选择 Suspend Thread（暂停线程）即可，此时线程的右边还会出现一个红色的圆点标识应用暂停。暂停掉的线程将不会运行，即使我们单步运行，或者继续运行代码。

要恢复线程的运行，则需再次右键选择这个线程，在弹出的菜单中选择 Resume Thread。

不要轻易尝试暂停线程的操作，因为这很可能导致其他线程产生死锁，无法运行。此外，最好不要暂停主线程的运行。

11.7 快速查看

快速查看（Quick Look）可以在调试过程中通过弹出一个视图，来快速查看变量的值、地址等信息，如图 11-42 所示。下面简单介绍这个功能的用法。

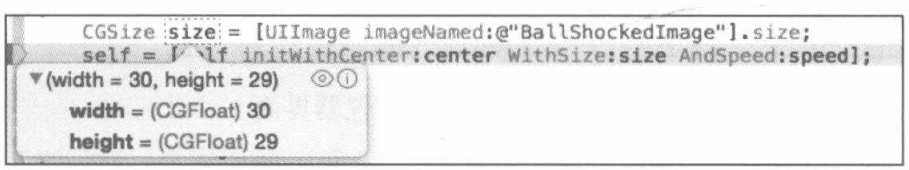


图 11-42 快速查看

11.7.1　查看变量

将鼠标指针悬停在代码编辑器中的任意一个变量上方，就会弹出一个小窗口，里面显示着该变量值的相关数据。点击变量旁边的检查器（inspector）图标（ ），就可以将这个对象的描述打印到控制台当中，并弹出一个额外的窗口来显示其描述信息。

点击快速查看图标（ ），就可以观察该变量的图形化展示信息，如图 11-43 所示。

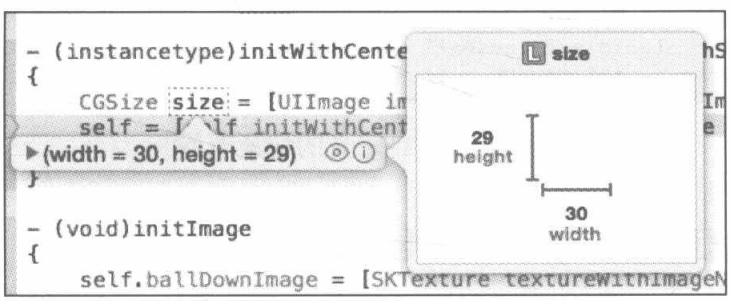


图 11-43　快速查看的图形化展示

当然，还可以使用调试区域的变量窗口下的按钮，也可以达成同样的效果。

11.7.2　为自定义类启用快速查看

很遗憾地是，默认的快速查看似乎只支持系统框架自带类型的变量查看，而我们有些时候，往往希望能够查看自己自定义类的显示情况。

为了实现能够快速查看功能，我们需要在类中实现 debugQuickLookObject 方法。

我们打开 Ball.m 文件，我们现在准备来为这个“小球”来提供快速查看功能，以便我们能够随时查看这个小球的信息。在这个类中添加以下代码：

```
- (id)debugQuickLookObject
{
    // 首先初始化你期望展示该自定义类信息的返回类型
    UIImage* quickLookImage;

    // 对这个类型进行处理
    if (self.texture == self.ballBounceImage) {
        quickLookImage = [UIImage imageNamed:@"BallFrustratedImage"];
    } else if (self.texture == self.ballDownImage){
        quickLookImage = [UIImage imageNamed:@"BallShockedImage"];
    } else {
        quickLookImage = [UIImage imageNamed:@"BallRelievedImage"];
    }

    // 返回快速查看对象
    return quickLookImage;
}
```

在 GameScene.m 中的 -(void)addClassicBall 方法内加上断点，然后编译并运行程序，

如图 11-44 所示，我们设定的快速查看方法就显示了出来，这里显示出了一个可爱的小球图片。

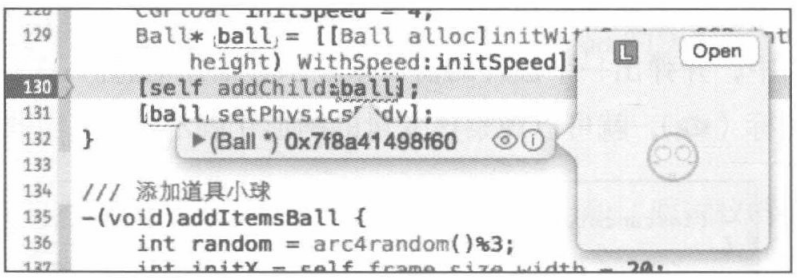


图 11-44 自定义的快速查看

11.7.3 自定义快速查看支持的返回类型

我们在上一节已经介绍了自定义快速查看的方法 debugQuickLookObject，虽然这个方法的返回类型是 id 类型或者 AnyObject 类型，但是实际返回的类型是又要求的。支持的返回类型如表 11-2 所示。下面分别介绍。

表 11-2 支持的返回类型

支持的类型	支持的返回类型
默认	—
图像	NSImage, UIImage, NSImageView, UIImageView, CIImage, NSBitmapImageRep
指针	NSCursor
颜色	NSColor, UIColor
贝赛尔曲线	NSBezierPath, UIBezierPath
位置	CLLocation
视图	NSView, UIView
字符串	NSString, String(Swift)
属性字符串	NSAttributedString
数据	NSData
URL	NSURL
SpriteKit	SKSpriteNode, SKShapeNode, SKTexture, SKTextureAtlas

1. 默认类型

默认类型如图 11-45 所示，默认情况下的快速查看显示了这个变量的名称（Name）、类型（Type）、值（Value）和总结描述（Summary）。

2. 图像类型

图像类型如图 11-46 所示，点击这个快速查看的 Open 按钮，可以使用预览应用打开这

个图片。

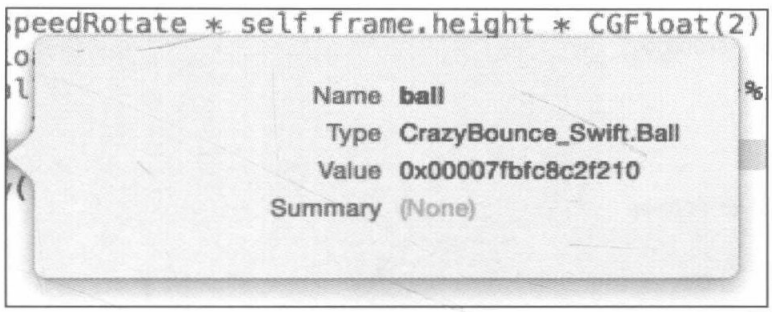


图 11-45　默认类型

3. 指针类型

指针类型如图 11-47 所示，点击这个快速查看的 Open 按钮，可以使用预览应用打开这个指针。

4. 颜色类型

颜色类型如图 11-48 所示，这个快速查看的左边显示的是对应的颜色区块，如果这个类有形状的话，那么就是显示这个颜色的形状区块。右边则是颜色的具体信息，包括三原色和透明度。

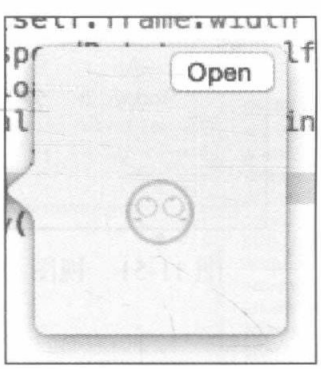


图 11-46　图像类型

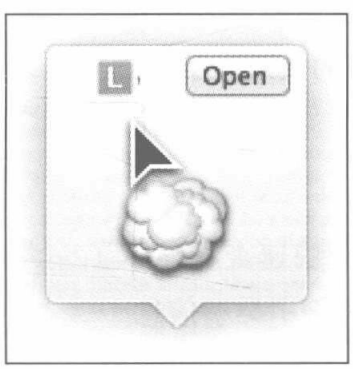


图 11-47　指针类型

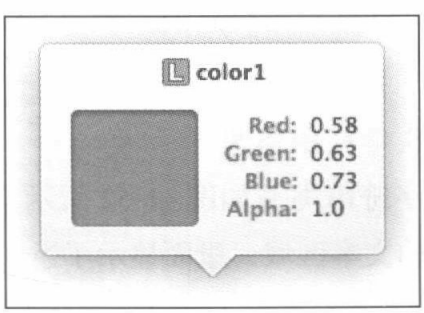


图 11-48　颜色类型

5. 贝济埃曲线类型

贝济埃曲线类型如图 11-49 所示，这个快速查看显示的是绘制出来的曲线。

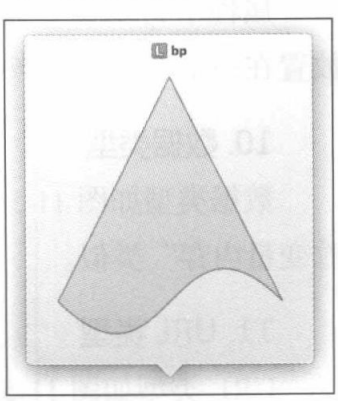


图 11-49　贝济埃曲线类型

6. 位置类型

位置类型如图 11-50 所示，这个快速查看显示的是对应位置的地图，在这个地图的左上角显示的是这个位置的相关信息。这个地图可以拖动，实际上它就是一个 MKMapView。

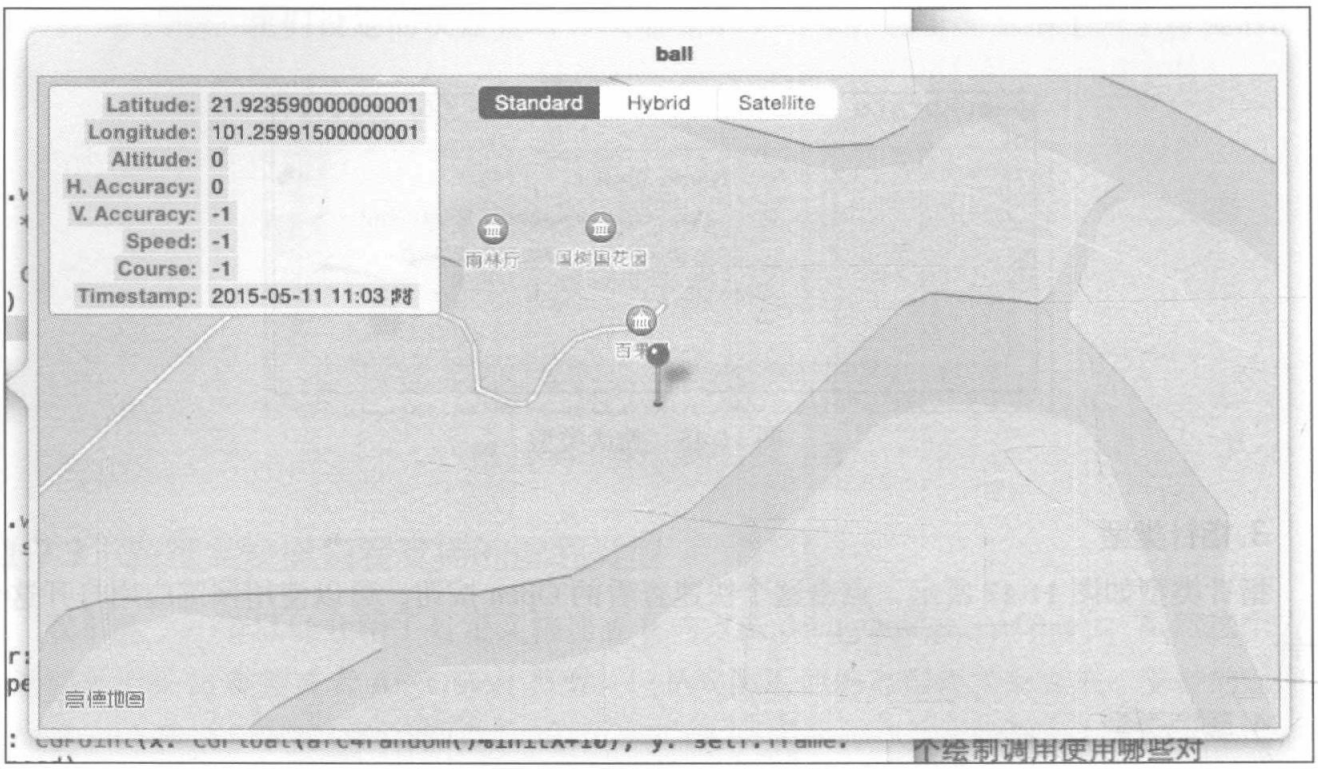


图 11-50 位置类型

7. 视图类型

视图类型如图 11-51 所示，这个快速查看显示的是一个日期选择器。在此，可以显示父类为 UIView 的任意一个视图类型，并且可以进行操作。

图 11-51 视图类型

8. 字符串类型

字符串类型如图 11-52 所示，这个快速查看显示了一大段可以操作的文本区域，里面显示了返回的字符串类型信息。

9. 属性字符串类型

属性字符串类型如图 11-53 所示，这个快速查看显示了带有属性的字符串，这些字符串放置在一个文本区域当中，可以进行操作。

10. 数据类型

数据类型如图 11-54 所示，这个快速查看显示了这个变量的数据值，和调试区域中的“查看变量内存”类似。

11. URL 类型

URL 类型如图 11-55 所示，这个快速查看显示了对应 URL 的网页。

图 11-52　字符串类型

图 11-53　字符串类型

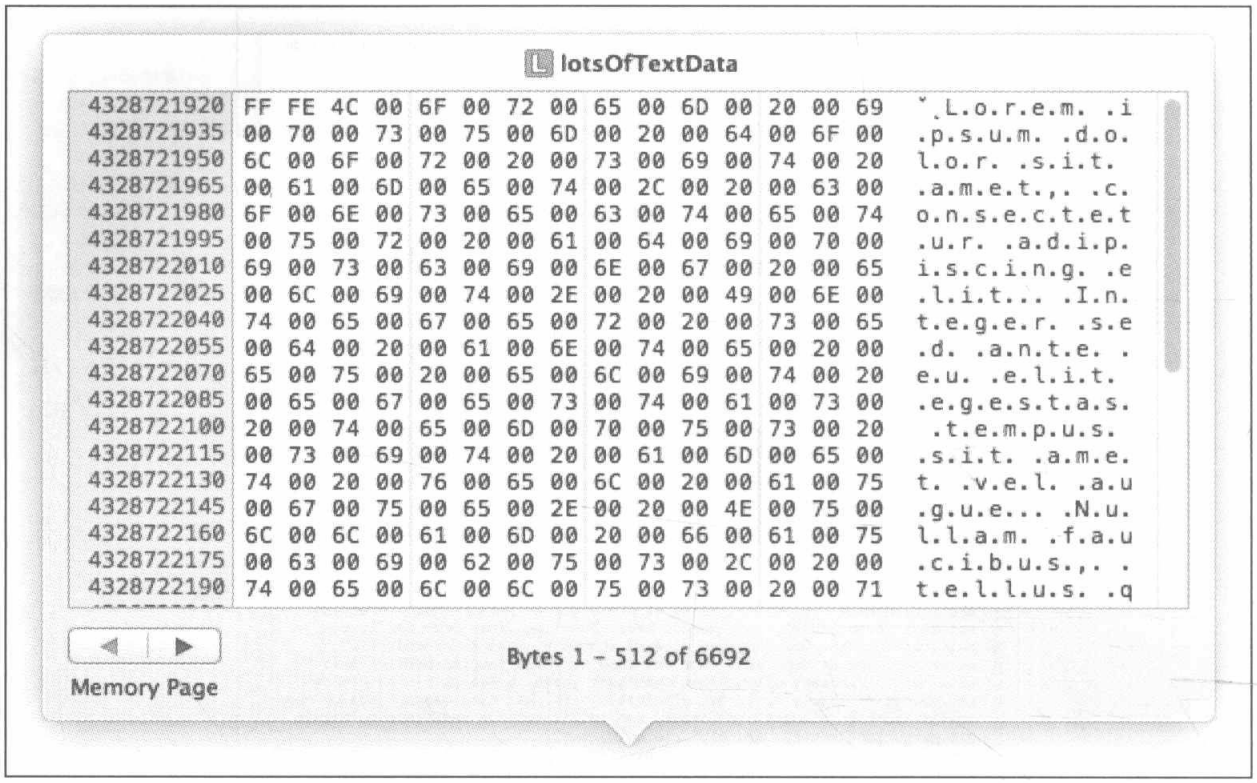


图 11-54 数据类型

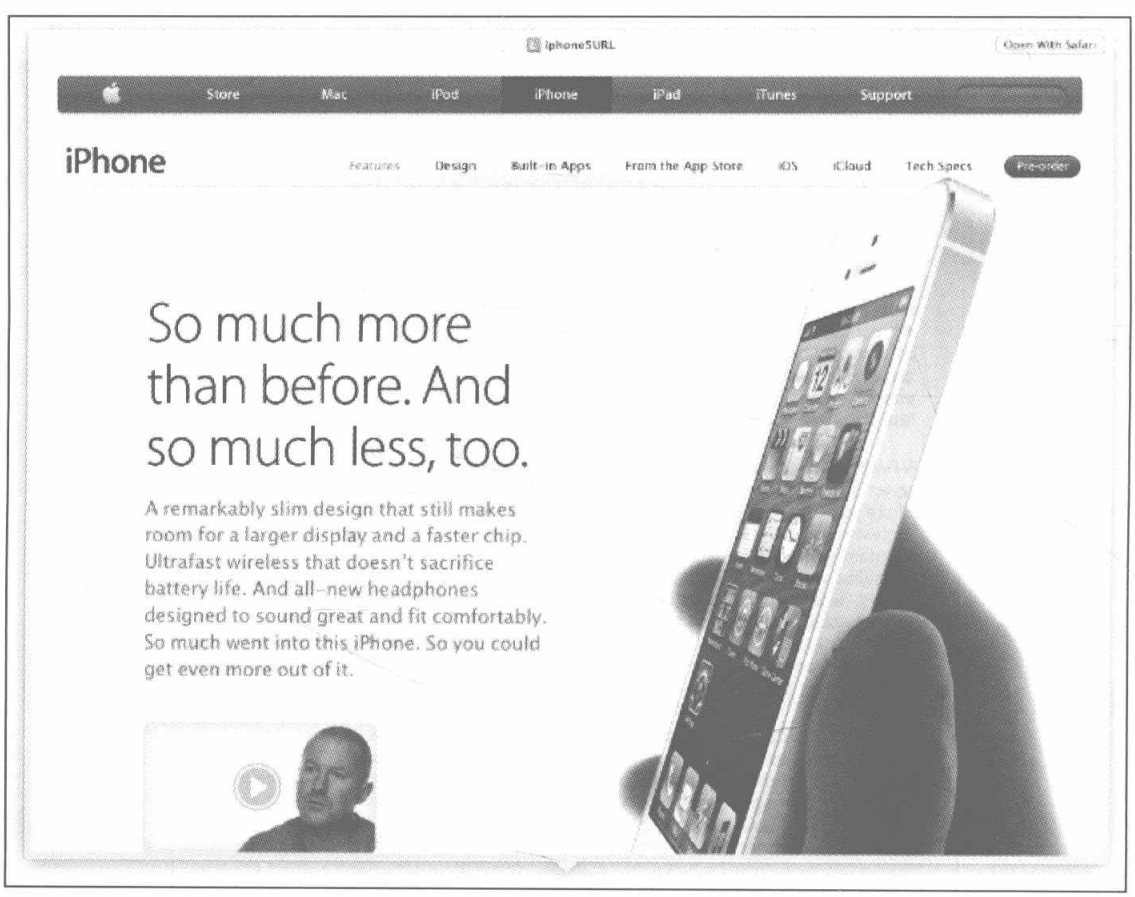


图 11-55 URL 类型

12. SpriteKit 类型

SpriteKit 如图 11-56 所示，这个快速查看显示了对应 SpriteKit 的模型，在这里，是系统自带的小飞机。

图 11-56　SpriteKit 类型

注意　Swift 的原生类并不支持某些类型的返回类型，比如说 UIColor。其实 Swift 有更好的替代品——Playground，调试时的快速查看并无太大必要。

11.8　LLDB 调试

前面我们介绍过，命令行可以执行 LLDB 调试命令。什么是 LLDB，LLDB 是一个高性能的调试器，之前我们做的所有调试操作，实际上都是使用 LLDB 命令，只不过 Xcode 使用图形化界面封装过了。

只有在应用暂停的时候才能够执行 LLDB 调试命令，在（lldb）后面即可开始输入指令。下面介绍常用的一些调试命令。

提示　实际上 LLDB 调试也会弹出语法提示的，但是截至目前 Swift 语言并未提供此项功能。

11.8.1 打印对象和值

print object（打印对象）是最常用的调试器命令，如果对象拥有 -description 方法的话，那么这个命令就会打印出这个方法的结果。一般最常用的是它的简要形式：po。


注意 Swift 打印的是 description 只读属性。

11.8.1.1 打印对象

如图 11-57 所示，我们在 GameScene.swift 的 addClassicBall() 方法中添加了一个断点，然后应用停止时，输入了 po ballLLDB 调试命令。这样，LLDB 便会去读取 ball 的 description 属性，然后将其打印在控制台上。这和变量设置当中的“打印变量名的描述信息”作用一致。

```
115     NSLog(@"开始道具游戏模式");
116
117     self.bestTime = [self.fileManager.bestTimeIdems intValue];
118     self.lbl_bestTime.text = [NSString stringWithFormat:@"Items Best:%@", [self
            timeTransformWithTime:self.bestTime]];
119     self.gamemode = Items;
120
121     [self addBar];
122     [self addClassicBall];
123 }
124
125 /// 添加经典小球
126 -(void)addClassicBall {
127     int initX = self.frame.size.width - 20;
128     CGFloat initSpeed = 4;
129     Ball* ball = [[Ball alloc]initWithCenter:CGPointMake(arc4random()%initX+10, self.frame.size.
            height) WithSpeed:initSpeed];
130     [self addChild:ball];                                          Thread 1: breakpoi
131     [ball setPhysicsBody];
132 }
133
134 /// 添加道具小球
135 -(void)addItemsBall {
136     int random = arc4random()%3;
137     int initX = self.frame.size.width - 20;
138     CGFloat initSpeed = SPEEDROTATE * self.frame.size.height * 2;
139     Ball* ball;
140     switch (random) {
141         case 0:
142             ball = [[AllCleanBall alloc]initWithCenter:CGPointMake(arc4random()%initX+10, self.
                    frame.size.height) WithSpeed:initSpeed];
143             break;
144         case 1:
145             ball = [[LongBarBall alloc]initWithCenter:CGPointMake(arc4random()%initX+10, self.fra
                    .size.height) WithSpeed:initSpeed];
```

CrazyBounce-OC 〉 Thread 1 〉 0 -[GameScene addClassicBall]

```
▶ self = (GameScene *) 0x7f8073c265f0
▶ ball = (Ball *) 0x7f8073c67510
  initSpeed = (CGFloat) 4
  initX = (int) 300

Auto
```

```
(lldb) po ball
当前小球的碰撞次数: 0，大小: {30, 29}，速度: 1.000000，当前位置:
568}

(lldb)

All Output
```

图 11-57 LLDB 打印对象


注意　和断点同样地道理，LLDB 调试命令一般情况下只能够读取当前作用域下的变量信息。

此外，LLDB 调试指令还可以访问对象的属性，并将其打印出来，具体的使用方法，和读取对象的语法相同，比如说我们可以输入 po ball.position 命令，读取当前小球的位置信息。

11.8.1.2　打印值

如果我们打算打印原始类型之类的值的话，那么很有可能会发现输出不符合我们的需求。比如说我们在 CrazyBounce 的 Objective-C 版本中的相同位置，输入 po [ball.description length] 命令，那么就会看到控制台输出的是：

```
0x000000000000007e
```

这个是以十六进制表示的字符串的长度，虽然信息没有错误，但是十六进制给我们带来了不小的困扰。

对于这种简单值，我们可以使用另外一种打印命令：

```
p (NSUInteger)[ball.description length]
```

这样调试器就会做出如下回应：

```
(NSUInteger) $1 = 116
```

p（print 的缩写）命令将我们提供的命令进行求值，然后将结果转换成预期的值类型（NSUInteger），以便知道如何以更常用的方式来显示它。

p 命令的格式为：p（类型）表达式。这个类型应该是表达式返回的类型，上面那个 length 方法返回的就是 NSUInteger 类型。

响应也和 po 极为不同，这个时候调试器不仅仅打印出了结果，还为其赋予了一个局部变量（$1），用于以后对其进行操作。上面的例子中，调试器将数字 116 赋给了局部变量 $1。我们在之后使用这个局部变量即可。


提示　对于 Swift 来说，没有必要添加“类型”信息，因为 Swift 可以自动推断类型，因此，上面的命令可以写成：p count(ball.description) 即可。此外，po 指令也可以达成用户期望的效果，因此对于 Swift 来说，p 指令几乎只剩下了赋予变量的作用。

11.8.1.3　自定义打印输出信息

在上面我们对不停地打印 ball 这个对象，但是 ball 这个对象是我们自定义类 Ball 的实例。因此，大家在打印的时候往往会输出以下信息：

```
<SKSpriteNode> name:'(null)' texture:[<SKTexture> 'BallShockedImage' (60 x 58)]
    position:{270, 568} size:{30, 29} rotation:0.00
```

这个是调用了 SKSpriteNode 的描述信息，那么要如何实现自定义的打印输出信息呢？答案在前面已经提到过，就是重写 description 方法 / 属性。

定位到 Ball 类当中，输入以下代码：

```
// Swift
override var description: String {
    get {
        return "当前小球的碰撞次数: \(knockTimes), 大小: \(size), 速度: \(speed), 当前位置:
            \(position)"
    }
}
// Objective-C
-(NSString*)description
{
    return [NSString stringWithFormat:@"当前小球的碰撞次数: %d, 大小: %@, 速度: %f, 当前位置: %@",
        self.knockTimes, NSStringFromCGSize(self.size), self.speed, NSStringFromCGPoint(self.
        position)];
}
```

再次运行程序，输入调试命令，duang~ 好了，我们成功地输出了自定义的描述信息了。

11.8.2 执行表达式

expression（表达式）也是比较常见的调试器命令之一，这个命令可以在调试过程中动态执行指定表达式，并且将结果打印出来。一般最常用的是它的简要形式：expr。

通过这个命令，我们可以实现修改变量值的功能，比如说，我们对 Ball 的速度进行修改，在与上面相同的环境下，输入以下 LLDB 调试命令：

```
expr ball.speed = 2.0
```

回车确认之后，然后再次输入 po ball 命令查看小球的相关信息，这时候我们发现，小球的速度被成功修改成了 2.0。

注意 表达式要符合当前语言环境，此外，expr 命令同样也会将表达式结果赋值给一个局部变量。由于 Swift 的复制命令不返回任何值，因此不会创建此局部变量。

11.8.3 控制程序执行

我们提到过，前面进行的调试操作实际上都是对 LLDB 调试命令的一层图形化封装，因此，我们也可以控制程序的执行状态。

表 11-3 展示了几条基本的 LLDB 命令。

表 11-3　LLDB 命令

命　令	用　法	描　述
process continue	c	继续执行当前程序
thread step-over	n	单步跳过执行一条语句，跳过函数、方法调用
thread step-in	s	单步进入执行一条语句，会进入函数和方法中执行
thread step-out	finish	完成当前函数、方法的执行，并返回到函数、方法的调用处

LLDB 还会记住过去使用的命令，和绝大多数控制台一样，使用上下箭头可以在命令历史中导航查看。如果没有输入命令的时候敲下回车，那么就会重复上一条指令，十分方便。

11.8.4　获取帮助

LLDB 还有许多许多的命令，面对这么多的命令和复杂的用法不必惊慌，我们只需要掌握上述三种常用命令即可，其他复杂的命令也可以通过帮助来学习。输入 help 命令，可以得到所有的主要命令列表，输入“help 命令名称”可以列出指定命令的所有子命令，输入“help 命令名称 子命令名称”可以获得具体子命令的帮助。

关于 LLDB 调试的更多内容，请参阅 LLDB 官方网站（http://lldb.llvm.org）来获取相关帮助和资料。

11.9　视图调试

视图调试（View Debugging）是 Xcode 6 带来的新功能。很多时候，应用的用户界面往往并不能按照预期所正常显示，比如某个视图并没有显示，或者显示的尺寸、内容不符合设想。借助视图调试，开发者可以在应用运行时，获取到当前应用全部视图的相关信息，同时也可以获取视图之间的层级关系，这些层级就好像“千层饼”一样。

11.9.1　启动视图调试

要启动视图调试这项功能，首先必须要运行应用，然后点击调试区域中的调试导航栏上的“Debug View Hierarchy”（视图层级调试）按钮，如图 11-58 所示。

图 11-58　通过调试导航栏启动视图调试

还可以选中调试检查器中的“进程操作选项”（Process View Options），在弹出的下拉菜单中，选择“View UI Hierarchy”（层级查看 UI），如图 11-59 所示。

此外，还可以选中菜单栏上的 Debug → View Debugging → Capture View Hierarchy，也可

以启动视图调试功能。

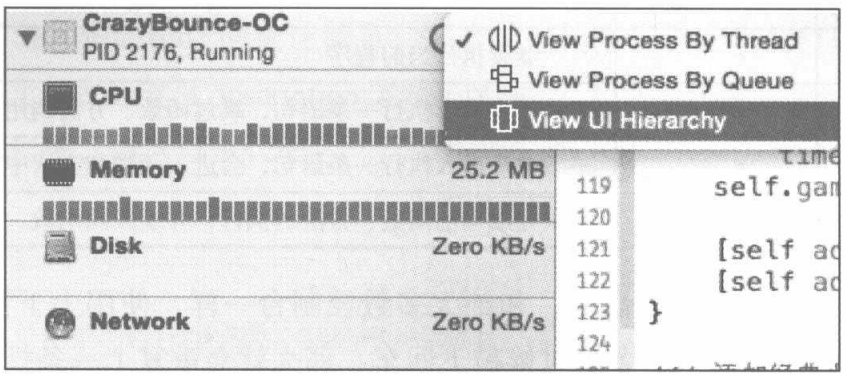


图 11-59 通过调试检查器启动视图调试

一旦视图调试功能启动，Xcode 将会立即暂停运行应用，然后捕获当前的应用视图，获取视图层级，接着在中间的用户界面编辑器中（之后统称为视图调试区域）显示当前应用的所有视图，如图 11-60 所示。

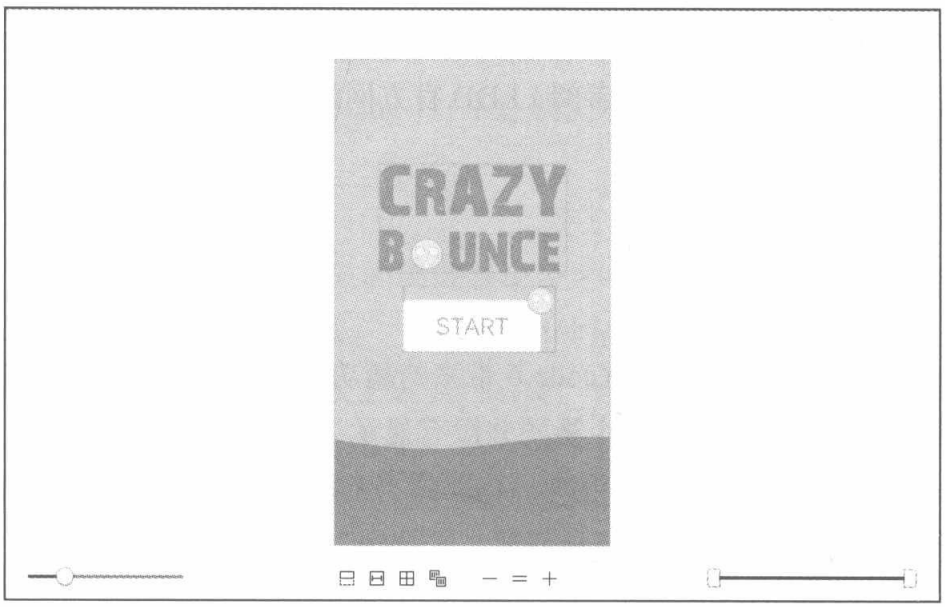


图 11-60 视图调试界面

显示的应用视图是以可交互的 3D 模型显示的，从这些视图中，还可以查看到视图间的位置关系、以及它们的尺寸大小。

11.9.2 视图调试功能

11.9.2.1 旋转视图

要旋转所展示的视图的话，点击视图调试区域的任意一个位置，然后摁住鼠标左键，拖动鼠标即可进行旋转。

当然，也可以使用触控板的“三指拖动”功能，这个操作比鼠标来得方便、舒适得多。

通过旋转视图，将视图旋转到一个你认为舒适的角度，以便于你能够直观地观察到视图之间的层级关系，如图 11-61 所示。

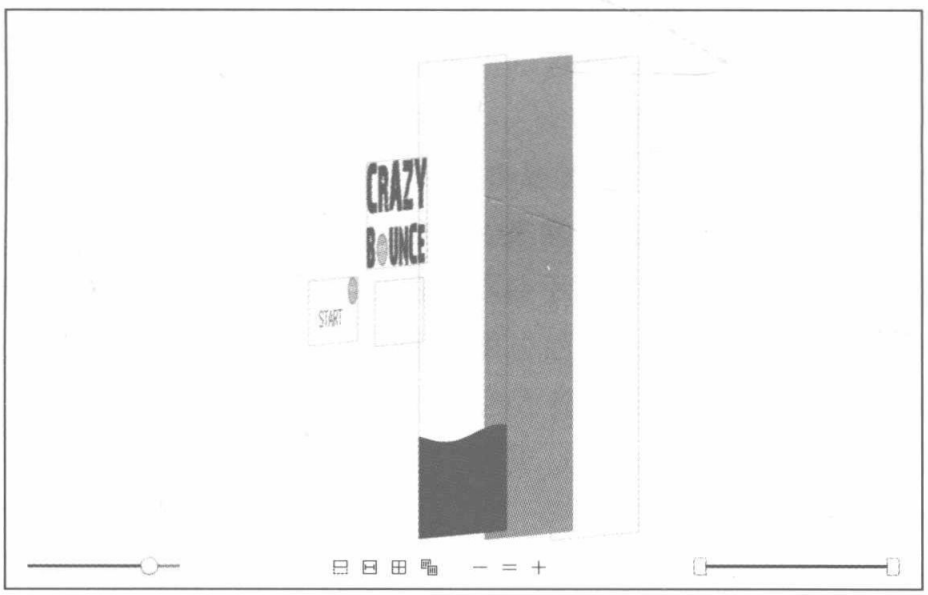


图 11-61　旋转视图

注意　最好不要旋转过多的角度，因为视图是以 3D 的形式显现的，所以这些视图可以“翻转”过来，也就是将内容进行了反转，容易影响对内容的判断。此外，尽量避免 90° 之类的角度，因为对于这些视图来说，它们只有长宽，基本不存在高度，因此一旦以这些角度查看视图，就会发现这些视图就像一张张薄薄的纸，就无法看见里面的内容了。

11.9.2.2　视图行为控制栏

在调试区域的上方，有一行视图行为控制栏，如图 11-62 所示。在这个控制栏上可以对视图调试区域进行更为精确的控制。

图 11-62　视图行为控制栏

左边的滑动条用来调整视图之间的距离，以便于开发者能够以一个舒服、合适的视角来查看视图之间的层次关系。这个距离最好不要太窄，不然的话就无法直观看到视图间的层级关系了，因为这些视图会“贴合”到一起。距离过宽的话，如果你的 Mac 屏幕尺寸不够，那么很可能就无法看到全部的视图。

位于中间的是编辑器控制按钮。最左边的是“ Show Clipped Content”（显示被剪切内

容）按钮，用来显示当前视图中被剪切掉的内容。往往选择了“AspectFill”之类的视图填充模式的时候，会造成视图的部分被丢失掉。选中这个按钮，可以查看哪些部分被“剪切”掉了。

是“Show Constraints”（显示约束）按钮，用来显示当前视图中的所有约束。借助这个功能，可以查看约束是否设置正确，或者是否成功发挥了效用。没有发挥效用的约束，将会以黄色，或者红色显现。

是“Reset the viewing area”（重置视图区域）按钮，用来将视图调试区域重置为初始状态。重置操作仅仅只会重置查看视图的角度，也就是恢复初始的查看状态。注意，重置操作并不会重置视图距离、显示约束等操作。

是“Adjust the view mode”（调整显示模式）。这个选项能够让视图是只显示内容（Content）、还是只显示线框图（Wireframes），还是两者一同显示。

接下来是的三个按钮是用来控制视图大小的，分别是缩小、以合适大小显示、放大。你同样可以在触控板中使用缩放手势来控制视图的大小。

最后一个控件是用来调整可见视图的范围的，通过调整这个控件，可以用来隐藏外层视图，从而能够看到内部的视图的显示方式。

请试着多实验一下这几个功能，这些功能在进行视图调试的时候非常常用，同时也便于你理解它们的实际作用。

11.9.2.3 调试导航器

启动了视图调试后，调试导航器上的内容就发生了变化。首先是调试仪表窗口被隐藏了起来（当然，我们也可以点击 Show Debug Gauges 来打开调试仪表，但是它实际上并不能发挥作用，因为当前应用暂停运行了），显示出来的是当前的所有视图。这些视图以层级的方式显现出来，就好比分组管理一样，从中我们可以看出视图的包含关系，如图 11-63 所示。

在跳转栏上，也能够展示视图之间的层级关系。可以通过跳转栏来选中某一个视图，以进行更进一步的操作。

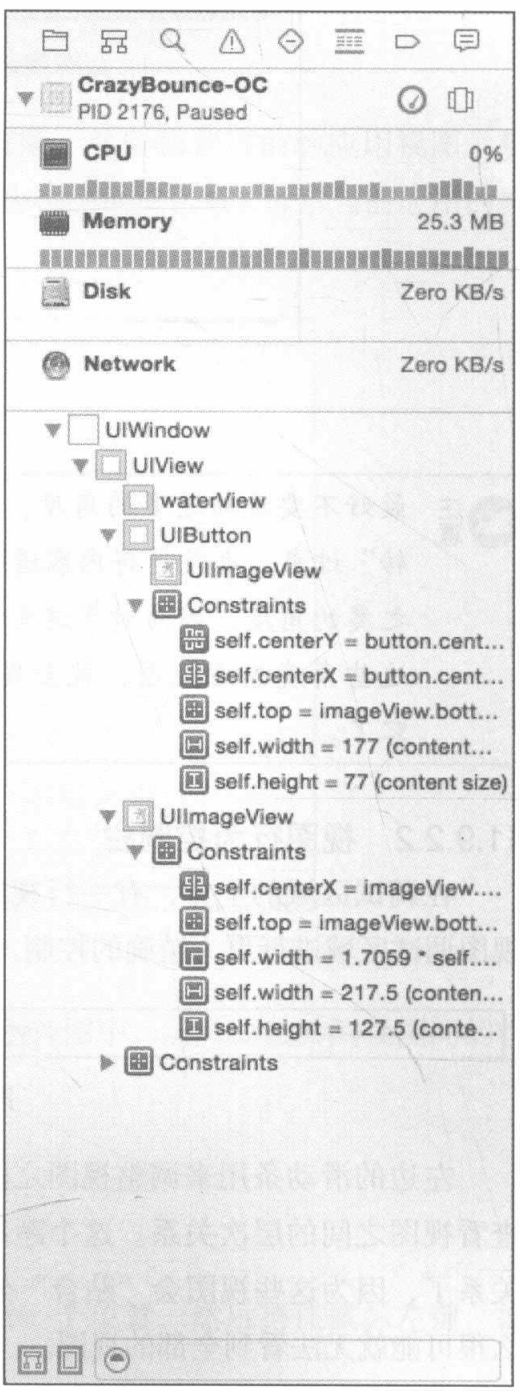


图 11-63 调试导航器

11.9.2.4　对象检查器和尺寸检查器

我们一旦选中了某一个视图，那么右边的工具区域中就会显现出对象检查器（Object inspector）和尺寸检查器（Size inspector）来，如图 11-64 所示。这两个检查器当中都包含了许多有用的信息，以便检视。

对于对象检查器来说，Xcode 会显示这个视图对象的基本信息，通常就是其类名和内存地址。对于某些特殊的视图，比如说 UIImageView 或者 UIButton，对象检查器还会显示这些视图的一些特殊属性，比如说图片、状态、文本信息等等，如图 11-64 所示。

在图中，我们可以看到，这个对象的类名为“UIImageView”，内存地址为“0x7fc9faeae150”。我们可以从 image 当中看到它的实际图像，然后看到一些属性。

对于尺寸检查器来说，Xcode 会显示这个图层的 bounds 信息，我们可以从图中直观地看出 frame 和 bounds 的区别（黑色边框的是 frame，而红色虚线是 bounds）。同时，Xcode 还会显示这个视图在父视图当中的位置，以及其锚点位置、约束等等一系列信息，如图 11-65 所示。

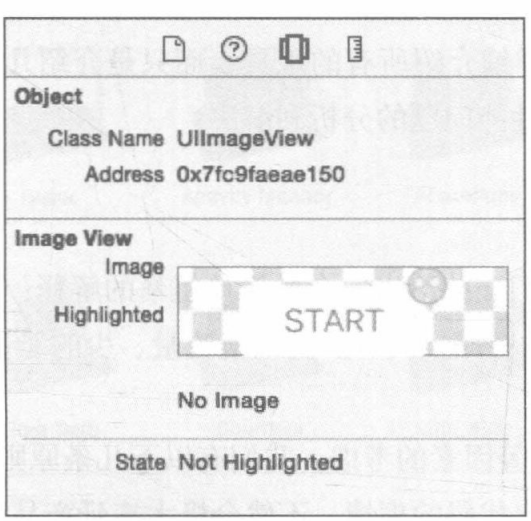


图 11-64　对象检查器

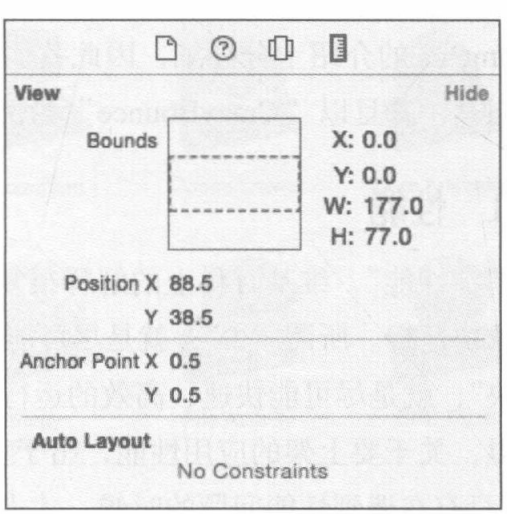


图 11-65　尺寸检查器

11.9.2.5　其他选项

在“调试导航器”的左下角还存在两个按钮（见图 11-63），这是用来控制视图的显示方式的。

左边的是“Show Primary View”（显示主要视图）按钮，这个选项用来设定是否过滤掉次要的视图对象。所谓次要视图对象，指的是系统自己实现的相关视图，这些视图是不为应用所能够控制的。

右边的是“Show Only Visible Views”（仅显示可见视图）按钮，这个选项用来设定是否过滤掉没有显示在视图上的视图对象。一般来说，只要某个视图的透明度设为 0 或者设置不可见，那么默认情况下是不会在视图调试区域中显示的。

最后，点击调试栏上的“继续按钮”就可以退出视图调试，然后继续运行应用。

11.10 Instruments

前面的所有调试操作，都是针对于“问题”来进行的，也就是发现应用运行不符合我们的预期的时候进行的操作。但是，这并不代表应用能够成功地运行就万事大吉了，用户还往往需要应用有很快的响应速度，在快节奏的今天，应用每多卡一秒，就可能会丧失好多用户，相信这是大家都不愿意看到的吧？

要想提升应用的运行速度，这就需要找到并且消除消耗极高的代码，也就是减少应用对系统资源的占用。要做到这一点，我们首先就要知道应用将时间花在了什么地方，做了些什么操作，什么原因导致其运行速度缓慢，这个过程就叫做调试的最后一站——性能调试。

Xcode 集成了 Instruments 工具包，它将很多用于性能调试的工具集中到了一起，我们可以利用它来完成性能调试的绝大部分工作。

Instruments 中的工具量十分丰富，可以说专门介绍 Instruments 已经可以写一本书了，并且 Instruments 的介绍十分详细，因此我们并不会详细介绍所有的工具，而只是介绍几种常用的调试工具，并且以“CrazyBounce”为例来进行性能问题的分析和解决。

11.10.1 性能

所谓“性能”，维基百科上的解释很复杂，我们在此给出一个最简单粗暴的解释：能让应用越少越快运行。所谓“少”，就是尽可能使用少的存储空间、少的网络流量、少的能量消耗；所谓“快”，就是尽可能快速、高效的运行。

所以，关于要上架的应用性能，出于上架时间等因素的考虑，我们有以下几条原则：

- 只有在遇到性能问题的时候，才去考虑优化代码的事情，不然会极大拖延产品进度。
- 在进行性能调试之前，要先记录下当前应用的性能，然后和性能调试之后的结果进行比较。

第一条原则是可选的，因为如果应用没有遇到崩溃问题，那么为什么一定要优化它呢？先不用说考虑到产品经理、用户等众人的要求，就算考虑到优化所进行的工作量本身，做这个工作实际上是得不偿失的。因为优化需要大量的工作量，优化后的代码极有可能会产生 BUG。因此，一般请不要修改没有出现性能问题的应用。

如果你是 SDK、库、框架的编写者，或者闲着无聊的话，请当我上面这句话没说。这里单纯仅仅指普通上架的 APP 应用，实际上，你是没有时间来考虑这个问题的，新功能才是 BOSS 们关心的重点。

第二条原则，则是关键。因为我们在进行任何调试之前，都要证明我们的应用性能确实“变好”了。那么如何证明呢？只有数据才不会说谎。（而且当 BOSS 问你干了什么的时候，你才能拿这个数据塞住他的嘴）。

11.10.2　打开 Instruments

打开 Instruments 的方法无不外乎以下几种：

- 定位到菜单栏的 Xcode → Open Developer Tool → Instruments，单击即可打开。
- 长按工具栏上的 Run（运行）按钮，然后在弹出的对话框中选择 Profile（剖析），然后点击执行。
- 选择菜单栏的 Product → Profile 选项，单击执行。

打开 Instruments 之后，其欢迎界面如图 11-66 所示。

图 11-66　Instruments 欢迎界面

Instruments 欢迎界面的顶端可以用来选择要执行 Instruments 分析的目标设备和目标应用。一般说来，目标设备可以是 Mac 电脑，也可以是 iOS 模拟器，还可以是真机。目标应用可以是目标设备上的自建应用，也可以是某些特殊的系统应用。

通过双击欢迎界面中间的 Instruments 模板，即可打开 Instruments 工具界面，开始执行分析。

11.10.3 Instruments 模板

Instruments 中包含有许多的模板，这些模板用于监视应用程序的运行过程，并且记录下相应的数据以供分析。

表 11-4 列出了 Instruments 中的所有工具的用途。

表 11-4 Instruments 工具列表

Instruments 模板	所用工具	用途
Blank	空白	一个空白的跟踪文件，可以使用库中的 Instrument 工具自由组合，组建一个自定义的跟踪工具
Activity Monitor	Activity Monitor	用于监视 CPU、内存、硬盘和网络使用量，是调试仪器的高级模式
Allocations	Allocations、VM Tracker	可以跟踪匿名的虚拟内存和堆内存区域，同时也会提供类名以及对象的保留或者释放历史
Automation	Automation	可以为 iOS 应用模拟的 UI 交互，主要是通过运行脚本执行的
Cocoa Layout	Cocoa Layout	观察并记录下 NSLayoutConstraint 对象的变化，以帮助用户确定约束消失的时间和地点
Core Animation	Core Animation、Time Profiler	这个模板用于测量应用的图形图像性能，同时也会记录下 CPU 的占用率
Core Data	Core Data Fetches、Core Data Cache Misses、Core Data Saves	这个模板用于跟踪 Core Data 文件系统活动，包括查询、清除缓存以及保存操作
Counters	Counters	收集基于抽样法的事件或者跟时间有关的性能监视计数器事件
Dispatch	Dispatch	这个模板用于监视调度队列的活动，并且记录闭包调用和闭包的运行周期
Energy Diagnostics	Energy Usage、CPU Activity、Network Activity、Display Brightness、Sleep/Wake、Bluetooth、Wi-Fi、GPS	这个模板提供了关于电源用量的诊断服务，同时也提供了主要设备组件的开启或关闭
File Activity	File Activity、Reads/Writes、File Attributes、Directory I/O	这个模板用于监视文件和目录的活动状态，包括打开和关闭文件、文件修改请求、目录创建、文件移动等等
GPU Driver	GPU Driver、Time Profiler	测量 GPU 使用量以及抽样记录活动 CPU 的使用量
Leaks	Allocations、Leaks	记录所有的内存使用记录，检查泄露的内存，并且提供类的实例化信息，同时也提供了所有活跃对象和泄露闭包的内存地址记录
Network	Connections	分析应用使用 TCP/IP 和 UDP/IP 连接的情况
OpenGL ES Analysis	OpenGL ES Analyzer、GPU Driver	这个模板测量并分析 OpenGL ES 的活动，以此来发现 OpenGL ES 的错误和性能问题。同时，它也提供了解决这些问题的建议

（续）

Instruments 模板	所 用 工 具	用　　途
Sudden Termination	Sudden Termination	当某个对象正在运行过程中，突然中止其运行，然后分析并报告文件系统访问的回溯记录，以及开启或禁用应用闪退的调用
System Trace	Scheduling、System Calls、VM Operations	提供了全面的系统行为信息，包括线程配置信息，以及全部通过系统调用或者内存操作的用户、系统的代码变化
System Usage	I/O Activity	这个模板记录了 I/O 系统的活动，包括文件、socket 以及共享内存
Time Profiler	Time Profiler	分析代码性能，执行基于时间的低开销抽样进程
Zombies	Allocations	测量所有的内存使用记录，重点在于查找过度释放的“僵尸”对象。同时，其也提供类了的实例化信息，同时也提供了所有活跃对象的内存地址记录

11.10.4　运行 Instruments

要在 Instruments 的监视下运行应用是十分简单的，只需要编译（Build）应用程序，然后在 Instruments 欢迎界面中选择目标设备和目标应用，随后选择一个 Instruments 模板，Xcode 便会自动分析应用，然后以相应的模式来启动 Instruments，并且进行相关的配置。

Instruments 工具界面如图 11-67 所示，这里我们选择 Zombie 模板。下面介绍其中的功能。

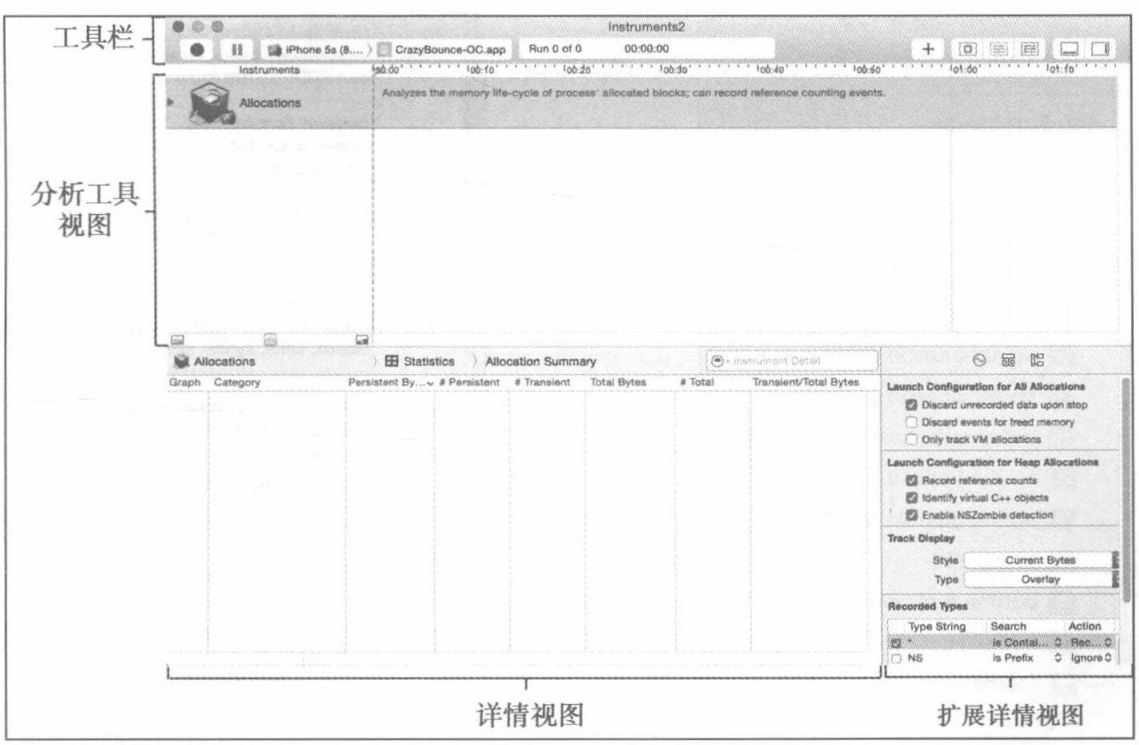


图 11-67　Instruments 工具界面

1. 工具栏

Instruments 界面的顶部是工具栏，它包含了一些非常重要的按钮。

最左边是录制（Record）按钮 ●，这个按钮用来控制 Instruments 的运行，让其跟踪应用的运行。选中这个按钮后，它将会变成停止按钮，用来停止分析工具的运行。

旁边是暂停（Pause）按钮 ⏸，用来暂停或者恢复应用跟踪。

再向右，是类似于“编译方案”的存在 Semper Pro › Choose Target...，这是对象（Target）区域。用来选择 Instruments 所分析的设备和所分析的应用。应用列表中将会列出设备当中存在的应用、应用扩展、系统进程等等。如图 11-68 所示。

中间的是活动栏区域，用来显示当前 Instruments 的活动状态和分析时间，以及会话数量。

右边的“+”按钮是“库”（Library）选项，用来展示 Instruments 工具库面板，如图 11-69

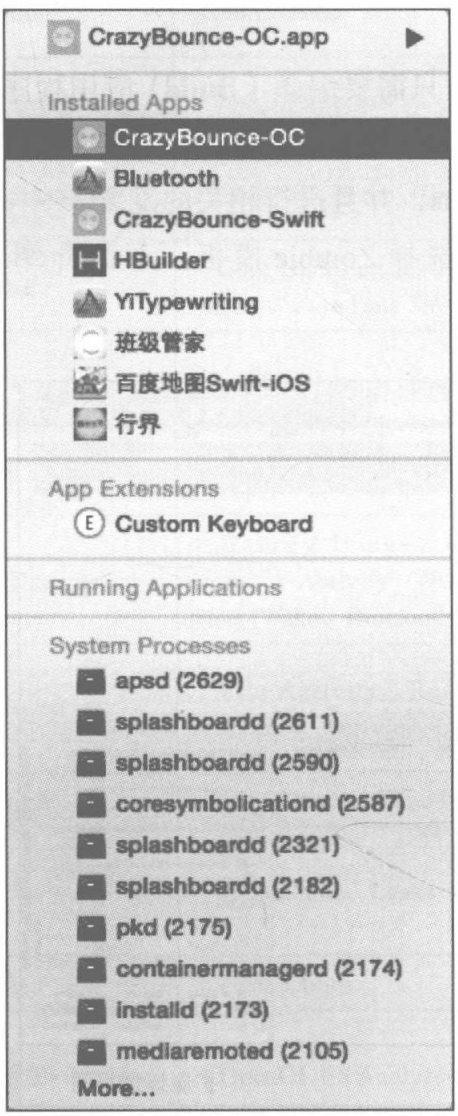


图 11-68 Instruments 应用列表

图 11-69 Instruments 工具库

所示。这个面板提供了所有可用的分析工具，我们可以在里面向 Instruments 中添加新的分析工具。

再右边选择的是 Instruments 的跟踪策略，主要有 CPU、Instruments 和线程三个策略。CPU 策略一般用于多个 CPU 性能的查看，线程策略一般用于多个线程性能的查看，一般而言，使用 Instruments 策略就足够满足正常的需求了。

最右边的是视图显示，分别用来控制展示“详情视图”（Detail）和“扩展详情视图”（Extended Detail）。

2. 分析工具视图

分析工具视图是最主要的视图，它从左到右依次显示了所有的分析工具列表和分析得到的数据时间轴。这些数据表示的含义根据分析工具的不同而不同，一般情况而言，时间轴的数据呈现深蓝色。

通过单击分析工具最左边的折叠三角形按钮，可以展开分析工具，查看该分析工具的内部详细信息，如图 11-70 所示。

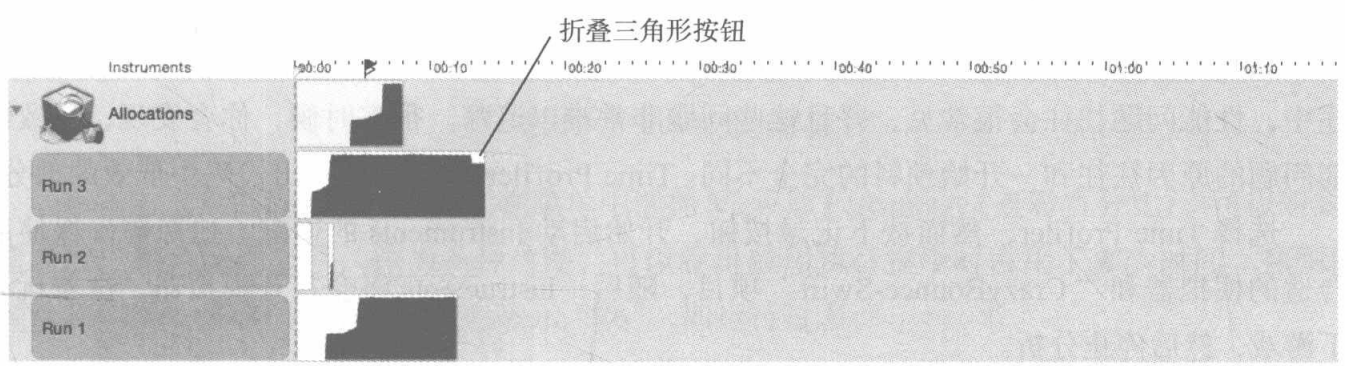


图 11-70　分析工具内部详细信息

分析工具视图的左下方还有一个小小的容易忽视的元件，这是用来控制缩小和放大时间轴的滑条，通过其可以控制时间轴的时间单位长度如图 11-71 所示。

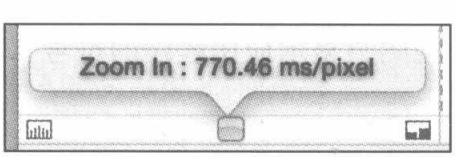


图 11-71　控制时间轴

3. 详情视图

详情视图是显示实际信息的地方，换句话说，这里面所显示的数据才是真实有效的。

详情视图的顶部是一个导航栏，如图 11-72 所示。最左侧是分析工具选择器，用来选择详情视图中显示哪个分析工具的详情信息。

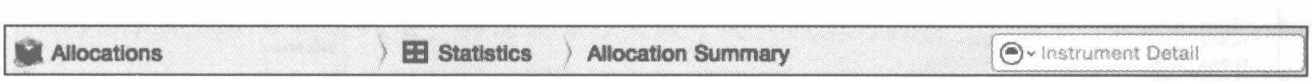


图 11-72　详情视图导航栏

下一项是选择跟踪信息的显示模式，这些显示模式取决于所选择的分析工具，但是大部

分工具都拥有 Call Tree（调用树）和 Console（控制台）显示模式，如图 11-73 所示。

导航栏的剩余部分则是专门用于导航的，类似于 Xcode 的跳转栏，通过其可以看到当前视图中的细节级别，还可以进行搜索操作。

导航栏下方则是详情信息的显示列表了。在这里就可以对 Instruments 分析的结果进行分析。

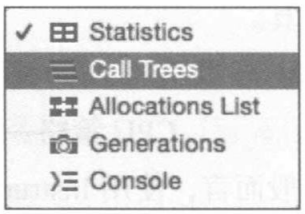


图 11-73 跟踪信息显示模式

4. 扩展详情视图

扩展详情视图显示了在详情视图中选择数据相关的扩展信息。它分为三个选项卡，分别是记录设置（Record Settings）、Display Settings（显示设置）和扩展细节（Extend Detail）。

关于这三个选项卡的详细内容，限于篇幅，就不加以详细叙述了。

11.10.5 Instruments 实例

现在我们已经了解了 Instruments 的一些基础知识了，然而至于 Instruments 如何使用，估计对大家来说还是一道大难题，那么我们就来举一个例子，起到抛砖引玉的效果。

最容易让人理解，也是最实用的功能就是 Time Profiler 了。前面介绍过，在许多应用程序中，性能问题往往会很常见，并且这些问题非常难以追踪。很多时候，你会发现，导致性能问题的原因往往和一开始预料的完全不同。Time Profiler 则可以显示出代码中低效的部分。

选择 Time Profiler，然而按下记录按钮，开始启动 Instruments 的追踪。追踪前确保使用合适的模拟器和“ CrazyBounce-Swift”项目，随后，Instruments 将会打开模拟器。试着玩一下游戏，然后停止分析。

将详情视图的显示模式设定为 Call Tree，然后在扩展详情视图的显示设置中，确保只选择了 Invert Call Tree（反转调用树）和 Hide System Libraries（隐藏系统库）选项。这将让详情视图只显示应用程序的符号，如图 11-74 所示。

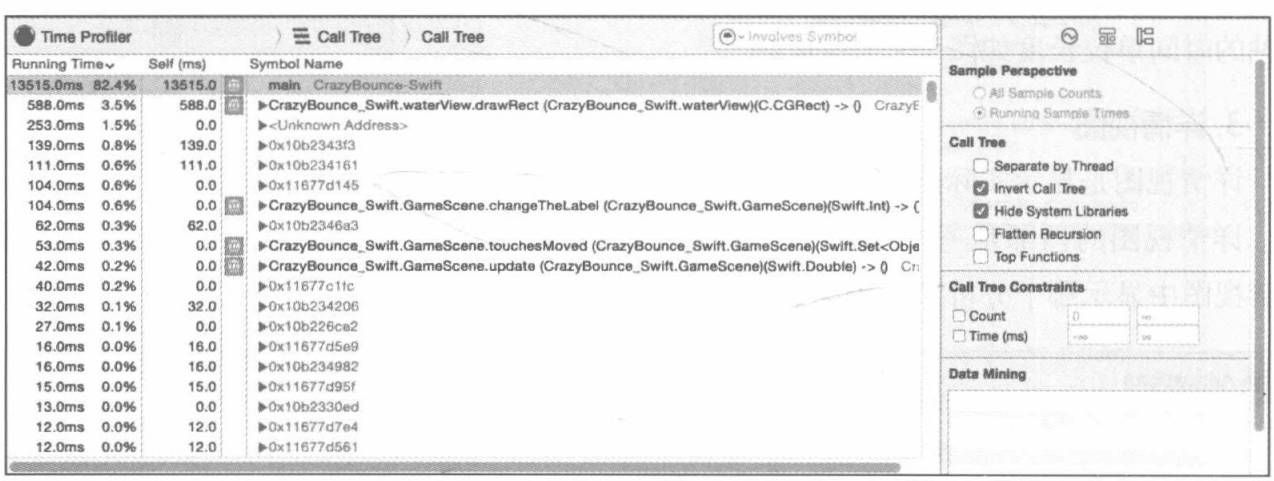


图 11-74 显示结果

可以看到，除了主界面之外，最占用资源的就是“ waterView”的 drawRect 方法了。可以想象，在这整个游戏中，底部的波浪视图吃掉了游戏多少资源！

双击列表中的 "CrazyBounceSwift.waterView.drawRect (CrazyBounceSwift.waterView)(C.CGRect) -> ()”符号，显示界面如图 11-75 所示。

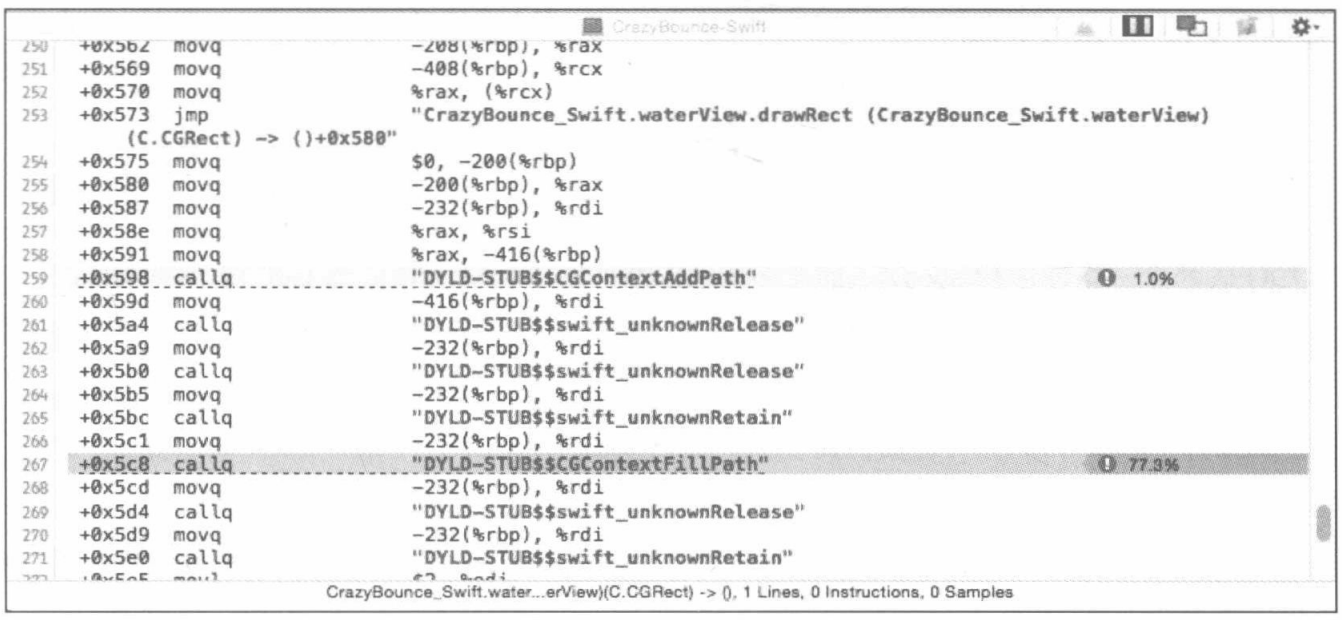


图 11-75　详细分析界面

单击右上角的齿轮按钮，然后在菜单中选择 View as Percentage（查看百分比）。可以看到，覆盖层以颜色编码显示，作为一种热图，可以看出应用执行指令时占用了多少时间。热图的颜色和“红外线热图”类似，红色表示“热”，即运行过多。

我们发现，导致这个方法占用过高的原因是 CGContextFillPath 这一个函数，花费了 77.3% 的时间，因此，导致性能的原因就被查找出来了。至于该怎么改进，那又是另外一个内容了。

关于 Instruments 的内容还有很多，苹果显然将其放在了和 Xcode IDE 同等重要的位置。单 Instruments 作为一个很好用的分析工具，应当是在掌握 IDE 之后，最应该学习和掌握的工具。

少年良辰将自己关在练功房内多日不出，房外有同门师兄弟多次路过时，却未曾听过房内有任何声音。师兄弟们按捺不住终于闯入，只见良辰瘫倒在案前，砚墨旁一滩血迹。“师父，师父，不好了！”他们边叫喊着边背着良辰奔向了大殿。

“这位少侠想必是走火入魔，将他放置在大殿中央，我来为他运气疗伤。”师父焦急地说道。

千里山峰，万事江湖。借一世，探人间，一念之差，堕入魔道，轻则武功全失，筋脉尽断，重则前功尽散，命赴黄泉。习武之人，最忌讳的便是拥有无穷力量却无法合理控制。此时的少年，命悬一线，能否再次苏醒就要看良辰的毅力和造化了……

Chapter 12 第 12 章

功力精进的途径——单元测试

“师父……”良辰渐渐从一片迷朦中苏醒过来。睁开眼，是白发苍苍的师父的脸和同门师兄弟的身影。”

良辰艰难地坐起身来，胸口还能依稀感到一股撕裂的疼痛。“现在感觉好点了吧，你这是走火入魔了。”大师轻声说道。“师父，弟子知错，恳请师父能够教授徒儿功力精进的秘诀，能让徒儿不再误入魔道！”

“你心态过于浮躁了，习武之事，万万不可急于求成。在每次练功之前，切记要检查自己的心脉，梳理经络，摒除杂质，这样才能防止走火入魔。这种检查的手法，今日为师就给你好好讲解下吧！”

所谓单元测试，就是测试代码“单元”的功能，以确保其在任何可能的条件下达到预期目的的一种测试方法。单元（Unit）是代码中一个可测试的逻辑部分。单元测试是应用开发的重要组成部分，因为单元测试可以帮助开发者找到错误和崩溃原因，而这正是苹果拒绝上架的首要原因。

测试方法应该要能够响应所有类型的输入，包括所有有效输入和无效输入的情况，以确保单元能够正常运行，也就是在有效输入下能够返回预料中的值，而在无效输入下不返回任何对象，甚至能够对错误进行处理和回应。无论开发者对单元进行了什么更改，现有的测试方法都应该能够成功运行，而新增加的测试也应该能够成功运行。

不过在很多情况下，开发者可能并不会对单元测试感兴趣。虽然它十分简洁、可以验证许许多多的问题，但是创建和维护这些单元测试却需要耗费大量的时间和精力，才能保证完全覆盖代码的所有功能和使用场景。

更为令人困扰的是，对于如何进行单元测试、什么样的单元测试方法才是合适的，这类问题很容易引起初学者的混乱，因为初学者并不清楚如何为代码设计测试。因此，如何设计单元测试并不在本书的介绍范围之内，我们仅仅介绍 Xcode 提供的单元测试功能。Xcode 为我们提供了丰富的测试功能，测试可以增强项目的稳定性，减少错误的发生，从而加快应用的分发和销售。一个经过良好测试的应用可以极高地提升用户的满意度，也可以帮助协调多人协作开发。

12.1　测试基础概念

Xcode 使用 XCTest 作为单元测试框架，在 Xcode 5 之前，Xcode 使用的是一个名为 OCUnit 的开源测试框架。XCTest 就是对 OCUnit 的替代品，能够更好地与 Xcode 协作。

单元测试的概念中有四个层级，分别是：

- **测试套件**（Test suite）：测试套件是项目中所有测试的集合，在 Xcode 当中，测试套件作为一个独立的对象存在。
- **测试用例类**（Test case classes）：测试功能是存放在类当中的，每个测试用例类通常是对应一个单独类来进行测试。比如说，对 GameScene 类的测试，应该由 GameSceneTests 类来完成。所有的单元测试类都必须要继承 XCTestCase 类。
- **测试用例方法**（Test case methods）：测试用例类包含多个方法，用来测试类的各种功能。测试用例方法应该专注于测试一个特定的步骤，并且最好能够涵盖所有的结果。
- **断言**（Assertions）：断言用于检查结果是否符合预期，如果不符合，那么断言则会失败。

打开 CrazyBounce-Swift 项目，然后找到 CrazyBounce-SwiftTests 对象，参见第 3 章中的图 3-1。CrazyBounce-SwiftTests 就是测试套件，这个测试套件是针对 CrazyBounce-Swift 应用进行测试的，我们可以在编译方案列表中看见这个测试套件对象。

CrazyBounce_SwiftTests.swift 是测试用例类，这个是 Xcode 创建的默认测试用例类，它不对应任何类，因此它也不具备任何实际功能，只是作为一个模板存在。

testExample() 是测试用例方法，这是一个函数测试用例。

XCTAssert() 则是断言，第一个参数是条件，第二个参数是断言失败输出的结果。


注意　测试方法必须以单词 test 开始，这样 Xcode 才能够找到。

我们还会发现，测试用例类中还包含了 setUp 和 tearDown 方法，这两个方法不属于测试用例方法，它们是测试用例类的初始化和析构方法。setUp 里面存放的是所有对象的设置代码，而 tearDown 里面存放的是诸如关闭文件、取消网络请求等清理活动代码。Xcode 会依次

调用 setUp、某个测试用例方法和 tearDown 方法。如果有多个测试方法的话，那么 setUp 和 tearDown 会在每调用一次测试方法的过程中调用一次。

对于 Xcode 来说，测试大致分为两种：一种是功能测试，另一种则是性能测试。性能测试需要设置基准线，一旦发现测试结果低于基准线，或者超出了最大标准差（STDDEV）限制，那么就会报告一个错误。

12.2 测试导航栏

在进行测试的时候，我们会频繁地使用 Xcode 的测试导航器。有关导航栏的初步介绍，我们已经在第 2 章介绍过了，参见 2.3.5 节。

简要来说，测试导航栏是用来快速创建、管理、运行和审核测试功能的管理区域。Xcode 现在默认为新创建的每一个项目都自动创建了一个“单元测试包”，比如说 CrazyBounce-Swift 项目，Xcode 自动为其创建了一个名为 CrazyBounce-SwiftTests 的单元测试包。如图 12-1 所示，目前测试导航栏中包括了三个单元测试包，分别对应不同的对象。

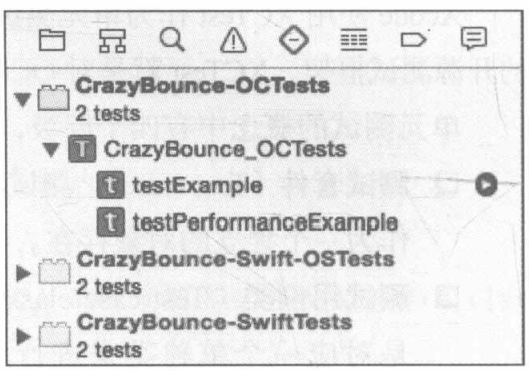


图 12-1 测试导航栏

注意 每创建一个新的对象，都会生成测试包，并且将其展示在测试导航栏当中。如果你需要使用文件、图片等类型的数据来执行测试，那么可以把它们添加到测试包中，并且使用 NSBundle 的 API 以便能够在执行测试操作时访问这些资源。

Xcode 自动创建的单元测试包的命名方式为“项目名 +Tests”，它默认包含了一个测试类，总共有 2 个测试方法。通过点击测试类和测试方法，可以在源代码编辑器中访问对应的测试类和测试方法。

12.2.1 添加测试对象和测试类

在测试导航栏的底部，有一个工具栏，如图 12-2 所示。其中有四个工具组件，分别是：

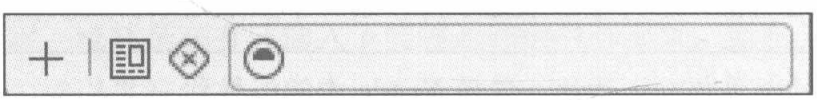


图 12-2 测试导航栏工具栏

- 添加按钮 +：单击以添加新的测试对象和测试类。
- 显示设置按钮：用来设置是显示全部测试对象，还是只显示当前编译方案下可用的

测试对象。

- 失败测试按钮：用来设置是显示所有的测试结果，还是只显示失败的测试结果。
- 搜索栏：用来搜索符合匹配的测试对象。

我们使用“添加按钮”，就可以完成添加测试对象和测试类的操作，单击添加按钮之后，会弹出一个菜单，如图 12-3 所示。

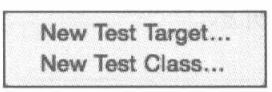


图 12-3　添加菜单

对于添加测试对象来说，和创建一个新的对象类似。输入测试对象的名称、设置好语言和项目以及绑定对象之后，就完成了测试对象的创建。

注意　测试对象仅支持 Objective-C 和 Swift 语言，此外，要确保测试对象绑定了你想对其测试的对象，不能够对测试对象创建新的测试对象。

对于添加测试类来说，和创建一个新的文件类似。但是必须要选择“ Test Case Class”文件，才能够创建一个有效的测试类。输入类名称、继承父类名称以及所用语言之后，就可以完成创建了。

注意　创建的测试类的语言必须要与测试对象的语言相同，测试类必须是 XCTestCase 的子类。

12.2.2　运行测试

当指向测试类和测试方法的时候，右边会出现一个“运行按钮”（），点击这个按钮会快速运行所选的这个测试对象。我们可以运行所有的测试，也可以运行某一个独立的测试方法。测试结束之后，Xcode 将会返回成功或者是失败结果。

当然，我们也可以选中我们想要运行的测试类，然后选择菜单栏的 Product → Perform Action → Run Test Methods 来运行选中的测试类。如果你只选择了一个测试类的话，那么相应的选项会变成 Run“测试方法名”。

Xcode 将会自动编译并运行应用程序，然后执行测试操作。测试操作完毕后，Xcode 会退出应用，短暂地显示测试结果，同时测试结果会以相应的图标表示测试是否成功。测试方法右边会出现测试结果，绿色的对勾标记是测试通过，红色的叉则是测试失败，如图 12-4 所示。

同样，测试结果也会出现在测试类当中，从测试类中可以看到测试失败的位置和原因，还有测试成功的执行时间。

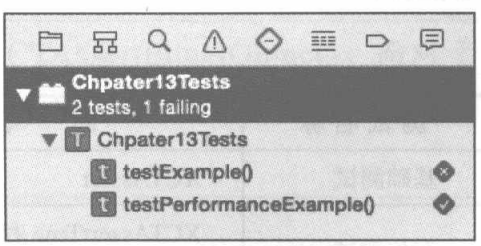


图 12-4　测试结果

在修复好经测试发现过的问题后，单击失败测试上的运行按钮，或者选择 Product → perform Action → Test Agian，来

再次对失败的测试进行测试。

要检视失败的测试结果，那么需要前往测试类所在的代码编辑器当中。在里面已经显示出了具体的错误信息。

单击失败信息末端的展开按钮，便可以完全显示测试方法所收集的错误信息，如图 12-5 所示。

```
1  //
2  //  Chpater13Tests.swift
3  //  Chpater13Tests
4  //
5  //  Created by 星夜暮晨 on 2015-05-02.
6  //  Copyright (c) 2015 益行人. All rights reserved.
7  //
8
9  import UIKit
10 import XCTest
11
12 class Chpater13Tests: XCTestCase {
13
14     override func setUp() {
15         super.setUp()
16         // Put setup code here. This method is called before the invocation of each test method in the class.
17     }
18
19     override func tearDown() {
20         // Put teardown code here. This method is called after the invocation of each test method in the class.
21         super.tearDown()
22     }
23
24     func testExample() {
25         // This is an example of a functional test case.
26         XCTAssert(false, "Pass")                                    XCTAssertTrue failed - Pass
27     }
28
29     func testPerformanceExample() {                                 Time: 0.000 sec (228% STDEV) 2
30         // This is an example of a performance test case.
31         self.measureBlock() {
32             // Put the code you want to measure the time of here.
33         }
34     }
35
36 }
37
```

图 12-5　测试类中的测试结果

可以看到，测试失败的原因是，断言的条件被设定成 false 了，导致了测试失败。

12.3　功能测试

功能测试的核心在于“断言”，测试结果取决于断言是否成功。

根据所测试的内容，开发者要设定断言必须满足的条件，比如相等、异常出现、返回值为空，等等。一般而言，功能测试大致可以分为如下几种：基础测试、布尔测试、相等测试、空值测试等，参见表 12-1。下面分别介绍。

表 12-1　功能测试列表

测试名称	断　言	特　性
基础测试	XCTAssert	最基础的断言，表达式为假测试失败
布尔测试	XCTAssertTrue 和 XCTAssertFalse	基础测试的扩展，当表达式结果和布尔测试不匹配时失败

（续）

测试名称	断　言	特　性
相等测试	XCTAssertEqual、XCTAssertEqualObjects 等	两个表达式不相等则测试失败
空值测试	XCTAssertNotNil	如果表达式为空则测试失败
无条件失败	XCTFail	总是测试失败

12.3.1　基础测试

所有的 XCTest 断言都是基于一个最基础的断言而扩展出来的：

```
XCTAssert(表达式, 消息)
```

如果表达式为真，那么测试将会通过；否则测试失败，打印 format 格式化后的信息。

虽然说所有的测试都可以使用 XCTAssert，但是 Xcode 仍然提供了一些更好、更有用的语法来帮助开发者理解正在进行的测试是在做些什么。只有在没有更好替代的情况下，才使用 XCTAssert。

12.3.2　布尔测试

布尔测试是对基础测试的简单扩展，只不过开发者不再局限于 XCTAssert 判定失败的条件。XCTAssertTrue 是当表达式为假时失败，和 XCTAssert 一样。XCTAssertFalse 是当表达式为真时失败。

比如，以下语句将导致断言失败：

```
XCTAssertFalse(2 == 2,为真导致失败)
```

12.3.3　相等测试

相等测试用来判断两个值是否相等，使用的是 XCTAssertEqual 断言，如下代码：

```
XCTAssertEqual(表达式1, 表达式2, 消息)
```

当表达式 1 和表达式 2 不相等时，则断言失败。表达式可以是标量、结构体和联合（Objective-C）。

对于 Objective-C 来说，判断对象是否相等要使用 XCTAssertEqualObjects。而对于 Swift 来说，判断对象是否相等直接使用 XCTAssertEqual 即可，标量值和对象间没有区别。

如果要测试 Double、Float 或者其他浮点值是否相等，使用 XCTAssertEqualWithAccuracy 可以来报告浮点数精度问题。

除此之外，还有 XCTAssertGreaterThan、XCTAssertLessThan 等等断言，一一对应于＞（大于）、＜（小于）等比较运算符。

12.3.4 空值测试

使用 XCTAssertNil 断言可以判断一个给定的值是否为空，如果为空，则断言失败。如下代码所示：

```
XCTAssertNotNil(表达式, 消息)
```

还存在一个判断给定值是否不为空的断言：

```
XCTAssertNil(表达式, 消息)
```

一般而言，所有的断言测试都有其对应的“非”断言测试，一般就是在其中插入了 Not 字符，起到与之相反的作用。

12.3.5 无条件失败

XCTFail 断言将会总是失败，它没有任何条件，反正就是导致失败。

```
XCTFail(消息)
```

无条件失败通常用在控制流中，比如说一个 if else 结构中，放在 else 当中表示某些前置条件失败的情况下，导致测试失败。

12.3.6 测试实例

多说无益，我们下面举一个例子，来进行一个小小的功能测试。

我们看到 Ball.swift 这个文件，里面存放了游戏当中重要元素 :Ball 的相关定义。我们可以想象，默认情况下，Ball 的初始速度是不能为负数的，否则就会导致出错，或者导致游戏无法进行。

那么我们就针对 Ball 当中的 initSpeed 这个属性写一个小小的测试方法。在此之前，我们要在 Ball.swift 文件的 Target Membership 将 CrazyBounce-SwiftTests 对象勾选上，确保测试用例能够访问这个文件。

随后，我们定位到测试用例类当中，在 testExample() 方法中添加以下语句：

```
func testExample() {
    var ball = Ball(center: CGPointMake(0, 0), speed: -1)
    XCTAssertGreaterThanOrEqual(ball.initSpeed, 0, "Ball的初始速度必须大于0! ")
}
```

运行该测试方法，果不其然，测试发现了一个错误，如图 12-6 所示。

```
33      func testExample() {
34          var ball = Ball(center: CGPointMake(0, 0), speed: -1)
35          XCTAssertGreaterThanOrEqual(ball.initSpeed, 0, "Ball的初始速度必须大于0！")
36      }
        XCTAssertGreaterThanOrEqual failed: ("-1.0") is less than ("0.0") - Ball的初始速度必须大于0！
```

图 12-6　测试失败的结果

要确保这个测试能够通过，我们的 Ball 类就必须要防止 initSpeed 不能小于 0。因此，我们需要在初始化方法中进行相关的设定，打开 Ball.swift 文件，将初始化方法更改为以下情况：

```
init(center: CGPoint, size: CGSize, speed: CGFloat) {
    super.init(texture: ballDownImage, color: UIColor.clearColor(), size: size)
    self.position = center
    if speed > 0 {
        initSpeed = speed
    }
}
```

再次运行测试，就可以发现测试成功通过了。

12.4　性能测试

性能测试是 Xcode 6 提供给 XCTest 框架的功能。通过定义基准性能标准，开发者可以对代码性能进行比较和分析，确认重要的算法和一些跟时间有关的代码是否符合需求，以保持应用的高性能。性能测试主要是捕获代码的运行时间和其标准差，然后和基准数值进行比较。它需要在 measureBlock: 方法当中调用。

下面我们以一个简单的例子，来说明如何进行性能测试。

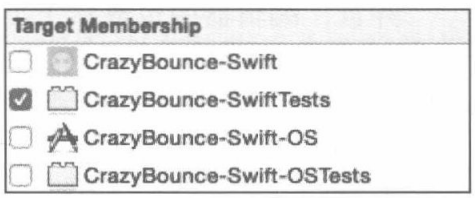


图 12-7　添加对象绑定

打开“CrazyBounce-Swift”项目，向相关的游戏文件的对象成员列表（Target Membership）中添加 CrazyBounce-SwiftTests 对象，如图 12-7 所示。

随后，向 CrazyBounce_SwiftTests.swift 文件中添加以下属性：

```
var gameScene:GameScene!
```

接着，在 setUp 方法中添加以下代码：

```
override func setUp() {
    super.setUp()
    // Put setup code here. This method is called before the invocation of each test
        method in the class.
    gameScene = GameScene(size: CGSizeMake(640, 1126))
    gameScene.gamemode = gameMode.Normal
```

```
    gameScene.saveCurrentTime = UILabel()
    gameScene.saveBestTime = UILabel()
}
```

接着是 testPerformanceExample 方法：

```
func testPerformanceExample() {
    // This is an example of a performance test case.
    self.measureBlock() {
        // Put the code you want to measure the time of here.
        self.gameScene.startGame()
        self.gameScene.bar.runAction(SKAction.scaleXTo(4.0, duration: 1))
    }
}
```

随后，运行该测试类，我们可以看到，控制台中输出了以下信息（仅挑选重要信息）：

```
Test Case '-[CrazyBounce_SwiftTests.CrazyBounce_SwiftTests testPerformanceExample]'
    started.
<unknown>:0: Test Case '-[CrazyBounce_SwiftTests.CrazyBounce_SwiftTests
    testPerformanceExample]' measured [Time, seconds] average: 0.031, relative standard
    deviation: 39.571%, values: [0.031480, 0.027363, 0.051209, 0.020990, 0.020961,
    0.046974, 0.020694, 0.020256, 0.046018, 0.019426], performanceMetricID:com.
    apple.XCTPerformanceMetric_WallClockTime, baselineName: "", baselineAverage: ,
    maxPercentRegression: 10.000%, maxPercentRelativeStandardDeviation: 10.000%,
    maxRegression: 0.100, maxStandardDeviation: 0.100
```

从控制台的信息中我们可以读取出，这个 testPerformanceExample 方法是对时间（Time）进行测量，而单位是秒。向游戏场景中添加小球的平均时间是 0.031 秒，标准差是 39.571%，说明时间在这个区间内上下浮动。并且，测试结果还列出了最近 10 次的测量结果。

并且，编辑器中也显示了类似的信息，如图 12-8 所示。

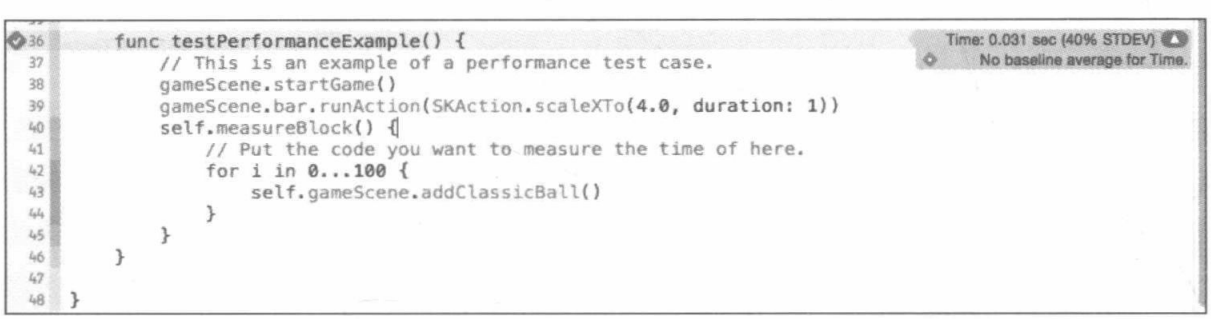


图 12-8　性能测试结果

点击 No baseline average for Time 消息旁边的灰色详情按钮 ()，可以查看具体的测试信息，如图 12-9 所示。

在其中，可以设定基准值和最大标准差，在性能测试中，没有进行基准值和标准差的设定，那么性能测试就没有太大的作用。

单击 Set Baseline（设定基准值）按钮，默认情况下，Xcode 会根据当前已测试的平均值设定基准值。

随后单击 Edit 按钮，然后就可以修改基准值和标准差了。单击“ Accept”按钮则是将基准值设定为平均值。单击 Save 按钮就可以保存基准值的设定了。

再次运行测试，就会发现结果已经有所变化了。

好的，到了这里，我们成功进行了功能测试和性能测试，完成了第一次测试驱动开发（TDD）周期，是不是很激动呢？

当然，关于如何制作好的测试，仍然是一项艰难的工作。鱼竿的使用方法你已经取得，至于如何钓鱼，则需要再多加努力和学习了。

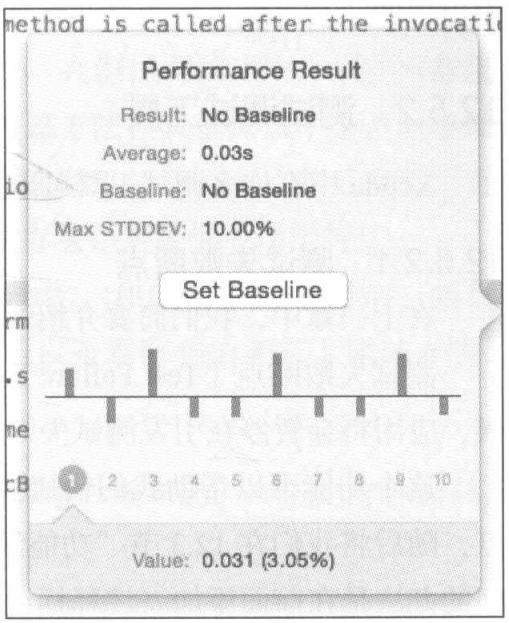


图 12-9　查看测试详情

12.5　测试调试

基本上所有的 Xcode 调试工具都可以在执行测试的时候使用，因此，在很多地方，人们所指的“测试”，往往是指“调试”和“测试”两者的集合，两者互为补充，共同帮助开发者能够发现和解决问题。

12.5.1　测试调试之前

在发现测试失败的时候，要先扪心自问一件事：是不是自己的测试方法存在问题？因为测试失败不仅仅是因为测试对象存在问题，还可能是因为测试方法存在某些不达意的地方，因为测试方法往往会引用很多内容，往往可能会由于测试方法的设计缺陷，导致测试方法没有按照预期进行工作。

因此，在进行测试调试之前，最好先检查一下测试的目的，然后再检查一下测试的过程是否符合要求。

首先要判断的是，测试的逻辑是否正确？实现是否正确？检查测试方法的笔误和错误是首要目标。

然后，检查测试对象是否满足要求。因为如果测试了错误的对象，那么很可能就会产生错误。

最后，检查是否使用了正确的断言。因为有些时候你需要使用 XTCAssertTrue 而不是 XCTAssertFalse，而这往往导致错误。尤其是断言冗长的情况下，一个 Not 往往能够毁坏整座大厦。

如果检查完上面几点，确保测试方法没有问题后，那么就说明错误一定是在测试的代码中出现的，就可以进行测试调试来修复它了。

12.5.2 测试调试工具

Xcode 中有许多调试工具可以用在测试当中，用于定位和调试代码。

12.5.2.1 测试失败断点

在上一章中，我们简要介绍过了测试失败断点。

测试失败断点（Test Failure Breakpoint）仅在测试断言失败的时候才会触发执行，这个时候，应用将会暂停在引发测试失败的代码处，而不是停止在测试方法处。

这个功能可以帮助我们快速找到问题发生的位置，在调试导航器中添加一个测试失败断点，随后将我们在 12.3 节“功能测试”的更改恢复到原来的样子，即把 if speed > 0 删除（最好的办法是在前面添加 // 注释标记）。

然后再次运行测试，就发现测试失败后，测试就停止在了问题发生的位置，方便我们进行修改。

这个操作的最大用处在于，当我们的测试类或者测试用例很庞大时，可以快速准确地定位到出错的断言位置，然后可以对其迅速做出更改。

要注意的是，当“测试失败断点”被触发之后，测试的执行也就停止了。因为是“测试失败”，才导致的该断点被“执行”，因此，最好的做法是在发生错误的断言前面设置一个常规的断点，修改后再次运行测试，来检查问题是否修复。

12.5.2.2 异常断点

在代码出现问题导致抛出异常的时候，可能会出现这样的情况，断点停留在造成崩溃的那条异常处，即使看了日志也不知道究竟发生了什么。这种现象算是比较常见，比如对于 NSArray、NSDictionary 中，一旦发生了“数组越界”的问题，那么断点并不会停留在出现越界的那行代码处，这导致开发者调试的困难。

这时候就需要异常断点来帮忙了，设置了异常断点后，调试器会在异常抛出的瞬间暂停程序的执行，将程序定位到出现异常的那一行代码。

关于“异常断点”的相关内容，在前面 11.4.4.1 节“异常断点”已经介绍过，在此不再加以赘述了。

不过要注意的是，和测试失败断点不同，我们在进行测试的时候，需要关闭异常断点，因为如果有异常被捕获的话，那么测试就会被中止执行。这样就可以避免异常抛出导致的测试失败。

12.5.2.3 辅助编辑器的跳转栏

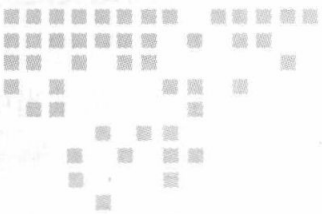
在第 2 章中，我们介绍过跳转栏的相关项目菜单，如图 2-12 所示。

我们曾经在 2.4 节中介绍了两个项目，分别代表了测试方法和测试类之间的相互调用关系，它们的作用如下：

- Test Classes：跳转到引用当前测试方法的测试类当中。在测试调试当中，当我们修复了一个引起测试失败的方法后，我们可能会想要检查该方法在其他测试中是否能够被成功运行。因此，打开辅助编辑器，然后在跳转栏中选择 Test Classes，然后就可以定位到任何调用它的测试方法，这样就可以来检查测试方法是否能够成功运行了。
- Test Callers：跳转到调用当前测试方法的测试方法当中。和 Test Classes 类似，当我们添加或者修改了一个测试方法后，用这个选项就可以查看哪些类拥有了测试方法，从而为没有添加测试方法的类或者方法添加测试。

单元测试并非易事，要想功力突进也不是一朝而成。若再想一蹴而就，即使是有神相助，少年良辰也难逃堕入魔道的下场了。

风萧萧，良辰独坐在山头上，手边放着一壶温酒。这一片深不可测的江湖，他也渐渐深入其中，而究竟何时，他才能站在巅峰眺望那万山起伏，万江奔腾呢？



Chapter 13 第 13 章

返老还童——版本管理

旧时月色易蹉跎，纵书已尽千册。夜深了，山谷中的夜风刺骨。

也曾有山头温酒伴月落，醉极便卧那云外山河。如今酒醒天寒，即便是一袭黑发的少年郎，面对着那弯孤月，心中也难以抑制住那岁月一去不再的凄凉吧。

正是伤心时，不料狂风四起，眼前一棵巨大的千年老树突然摇晃起茂盛的叶子。刹那间，少年良辰面前的景象开始变化，他仿佛觉得自己回到了多年前那个知了鸣叫，花儿低语的夜晚……？

原来，他面前这棵树，是一棵千年灵树。只要在这树前真心祈祷，便能看见自己曾经的时光。在编程的江湖里，也有这样一棵树，它叫做“版本管理”。这是一棵能够记录历史代码的“灵树”，以便开发者能够回到原来写的代码，同时也是方便多人协作的工具。换句话说，也就是一个存储代码的“网上仓库”。

13.1 工程快照

要说“网上仓库”，得先从工程快照讲起。快照为当前项目版本和工作区的备份提供了一个简单的方法。如果由于代码更改造成了程序错误，那么你可以通过快照来恢复整个工作区，包括将所有项目文件恢复到先前的状态。

快照是项目或者工作区中所有文档文件，以及所有项目和工作区设置的存档。快照支持恢复此前的 3 次更改，而 Revert Document 和 Undo 则不支持该特性。

- Xcode 操作涉及对多个文档文件和潜在的项目设置的更改。这些操作包括重构代码，

执行项目验证以及为现有项目添加 Automatic Reference Counting 等。

- 调整工作区和项目设置。
- 全局搜索和替代操作。

13.1.1　创建快照

可以通过选择 File → Create Snapshot 手动创建快照。Xcode 也可以自动创建快照。当你第一次对项目或者工作区做大量更改时，Xcode 会提示你为大量编辑操作打开自动创建快照选项。你也可以在项目或者工作区设置偏好中为大量编辑操作配置快照自动创建，选择 File → Project Settings 或者 File → Workspace Settings。在出现的设置窗口中，选中 Snapshots 标签，选中“Create snapshot of project before mass-editing operations”复选框。图 13-1 展示的是没有选中自动创建快照的设置窗口。

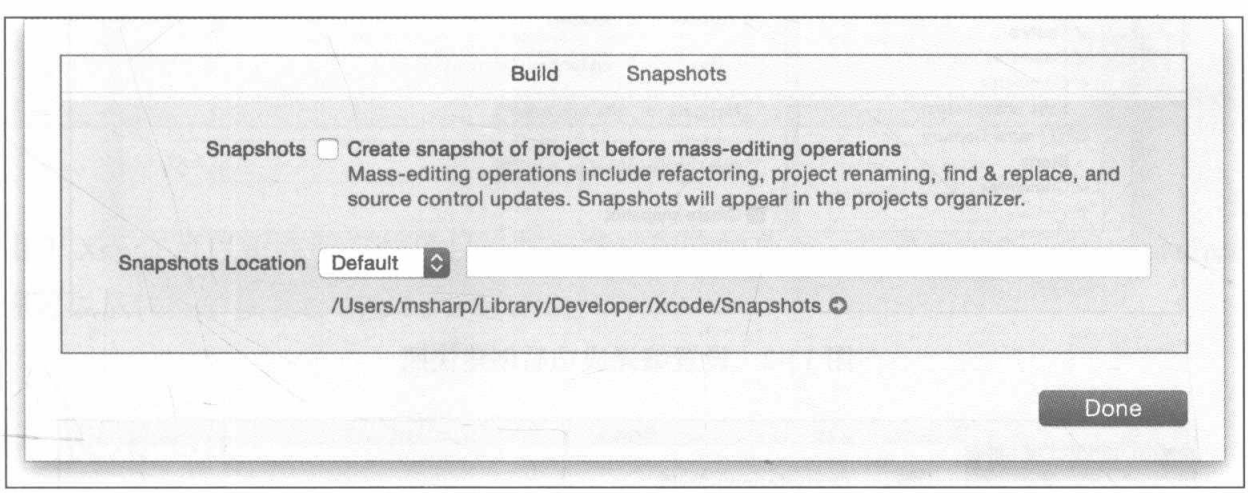


图 13-1　设置自动创建快照功能

想要在其他情况下将 Xcode 设置为自动创建快照，请选择 Xcode → Preferences，选中 Behaviors 面板，同时选择 Create Snapshot 选项。比如图 13-2 展示的是当编译成功时创建快照。创建快照的复选框位于配置行为面板的底部。

13.1.2　管理快照

想要查看项目或者工作区的快照，请选择 Window → Organizer，选中 Projects 来打开 Projects organizer，并点击 project，如图 13-3 所示。

13.1.3　从快照中恢复

你可以在工作区窗口选择 File → Restore Snapshot，从而对当前项目恢复快照版本。Xcode 会展示一个预览对话框，你可以通过它检查当前版本和快照版本之间的区别，如图 13-4 所

示。点击 Restore 选项，Xcode 会使用快照版本代替当前项目版本。Xcode 会在取代当前版本之前对其拍摄快照。

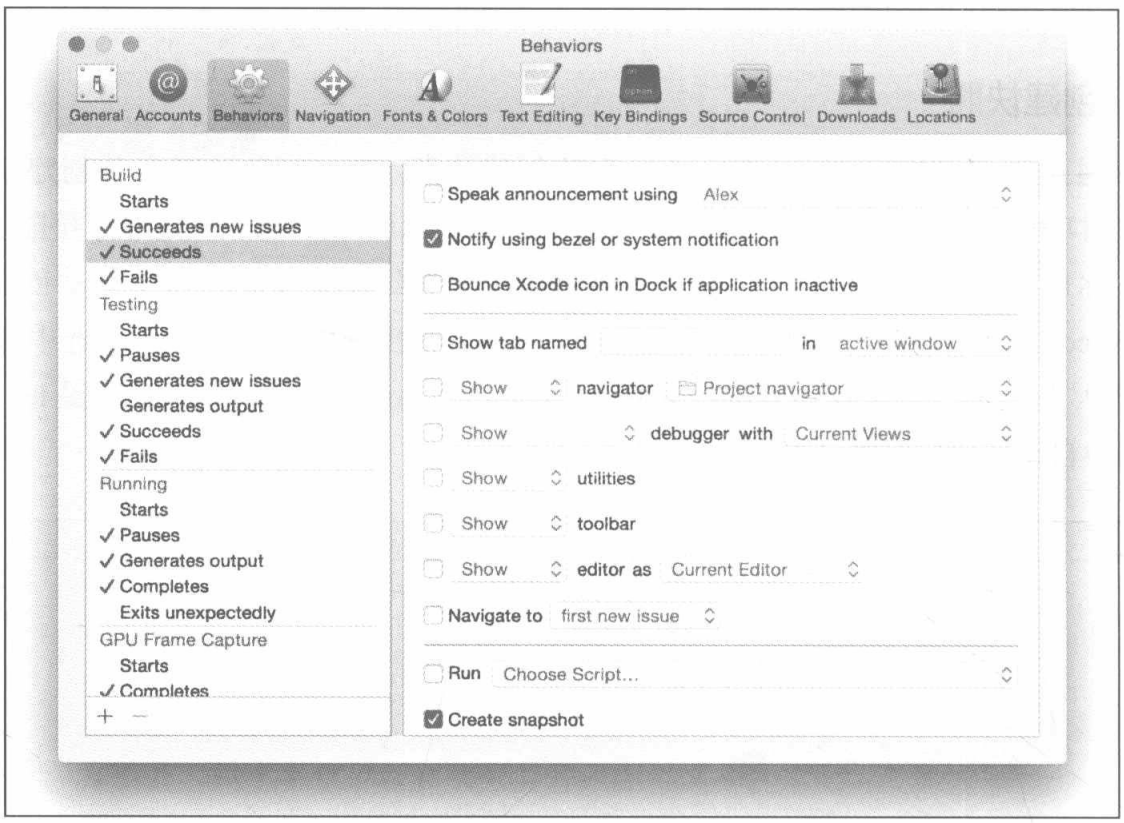


图 13-2　设置编译成功后创建快照

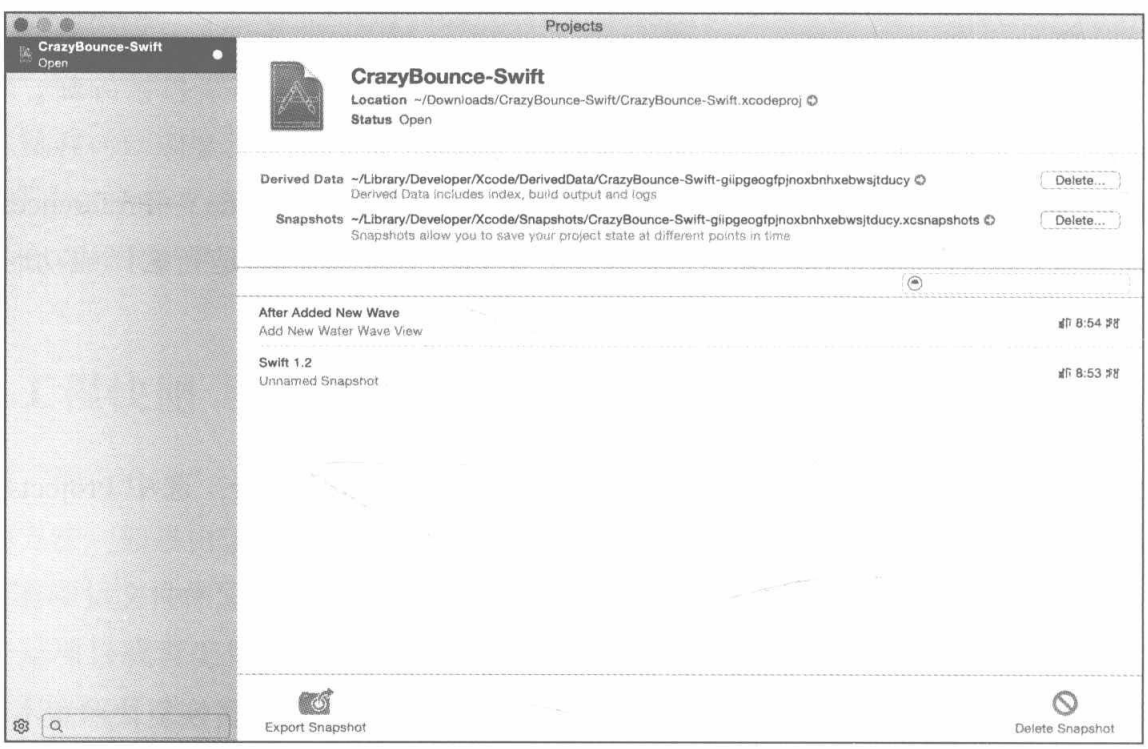


图 13-3　查看快照

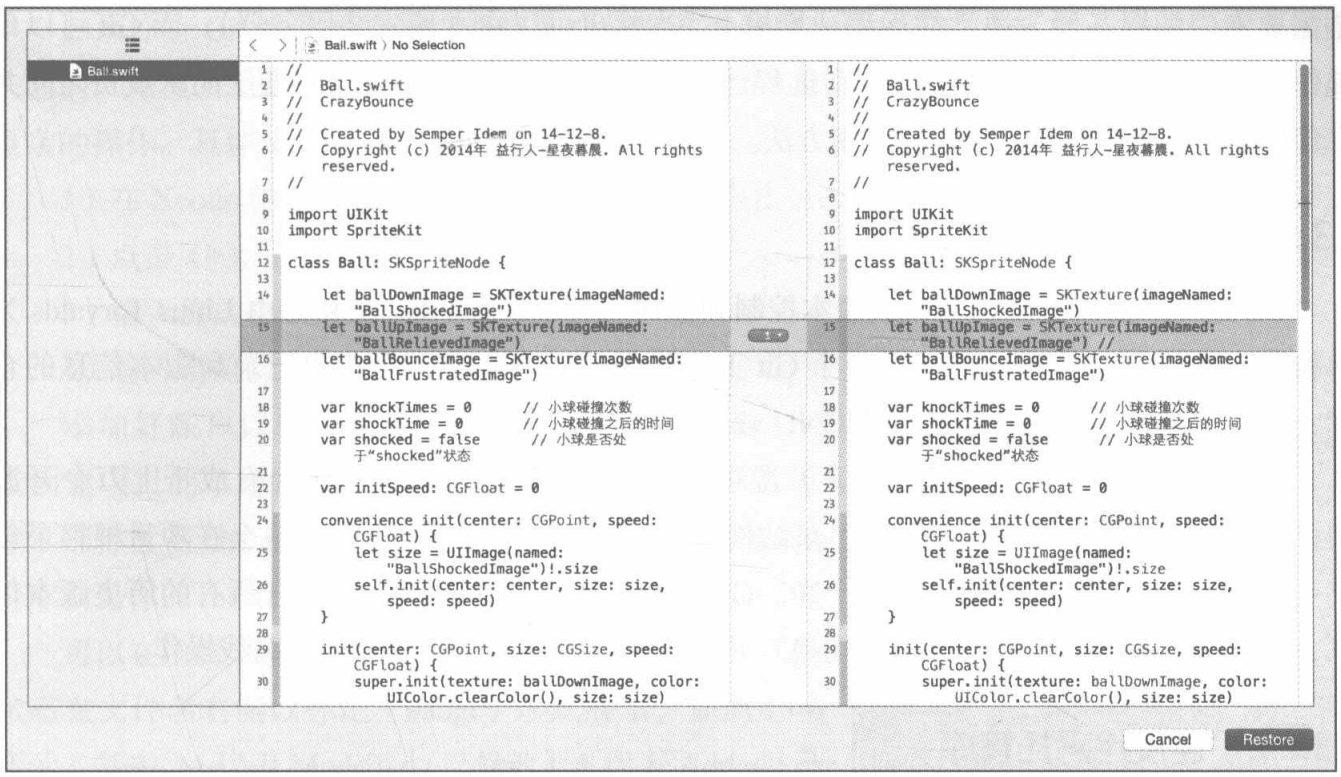


图 13-4　恢复快照

由于 Xcode 可以跟踪所有的项目，甚至是被删除的项目，并展示在 Projects organizer 中，所以你可以从快照中恢复已删除的项目版本。

13.2　使用 Git

使用源码控制菜单中的命令来管理项目文件源代码。源码仓库将文件的多个版本保存在磁盘上，储存每个版本的历史元数据。源码控制允许你精确地跟踪文件的更改，而不仅仅是快照。如果你有一个团队，那么源码控制也能帮助你的团队协同工作。

源码控制系统可帮你重建项目的过去的旧版本。在你每次进行了重大更改时可将文件提交到仓库中。如果你在提交的过程中引入了 bug，那么你可以使用 Xcode 版本编辑器来比较新旧版本之间的区别，从而定位出错的源码。

当多人同时操作一个项目时，源码控制有助于阻止冲突，并在冲突出现时解决问题。请使用核心仓库保存工程的原版代码，这样源码控制系统会允许每个程序员使用本地副本，而不会损坏原始版本。通过文件检测系统，可以确保不会有两个人同时操作相同的代码块儿。如果两个人同时更改相同的代码，那么系统将会帮你合并两个版本。

你也可以从项目的稳定版本衍生一个分支，添加新的特性，做一些其他更改，以及合并和协调这些更改到项目的稳定版本中。

Xcode 支持两种流行的版本控制系统：Git 和 Subversion。Subversion（一般简写为 svn）

是需要专门的服务器，通常在远程计算机上（当然也可以在本地安装服务器）。而 Git 可以单纯作为一个本地仓库，或者你可以在远程计算机上安装一个 Git 服务器以便于团队成员之间共享代码。本章我们只介绍 Git 的使用方法。

13.2.1 Git 简介

Git 是一个强调速度的分布式版本控制和源代码管理系统。Git 最初是由 Linus Torvalds 为内核开发而设计的管理软件。每一个 Git 工作目录是一个带有完全历史记录和版本信息的仓库，是一个免费的开源软件，遵从 GNU v2 协议。

自从 Git 推出以来，已经被很多开源项目所使用，用户群之大，已经有成千上万个开源项目采用 Git 来做项目管理。仓库目录结构简洁，用 Git 复制一个项目，只会在项目根目录创建一个 .git 的目录，而其他目录很干净。Git 分布式架构使得本地仓库包含所有的历史版本信息，你可以在不同的版本之间快速切换。并且，在开发过程中，完全可以离线操作。

13.2.2 连接代码托管库

如果是单人开发项目，使用 Git 是最简单的方式，因为无需设置服务器。当创建一个项目时，Xcode 会自动配置一个 Git 仓库，如图 13-5 所示。勾选 Create git repository on …选项即可完成代码托管库的创建，你可以选择将代码托管库建立在 Mac 上，也可以选择一个远程服务器进行存储。

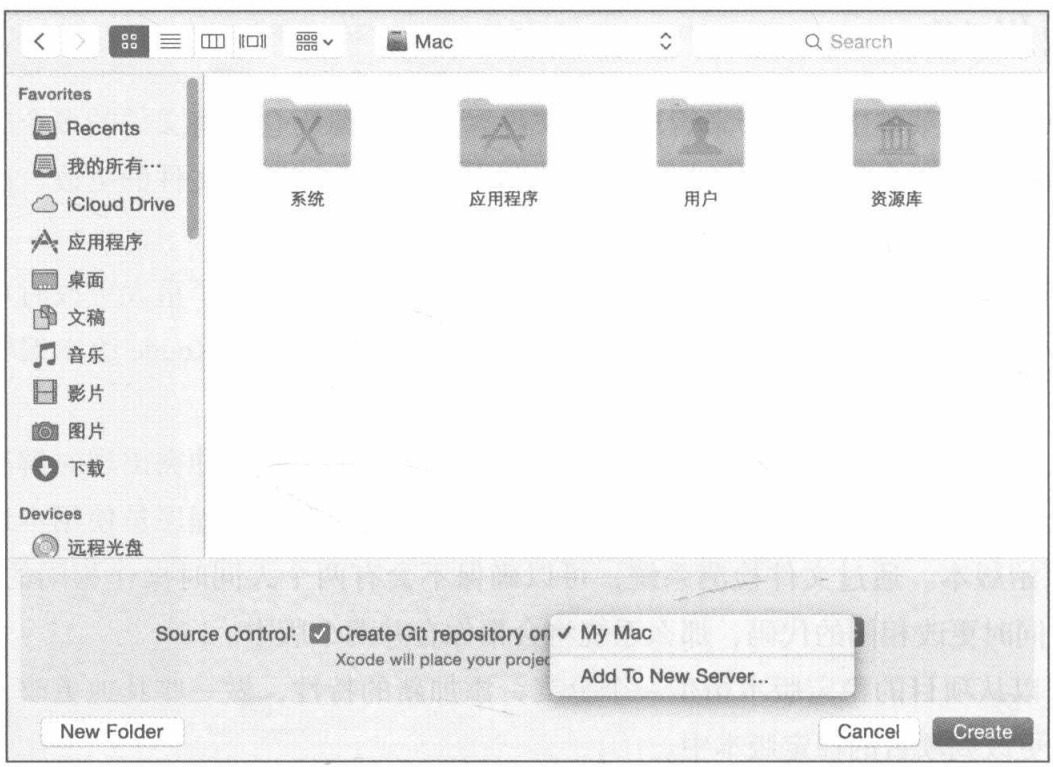


图 13-5 创建 Git

如果你有 Git 服务器，可以添加账户偏好库来存储你的凭据，这样就不用每一次重新输入它们。并且，Xcode 会自动配置好 SSH 等设置，Git 的使用也完全可视化，无需在终端进行复杂的操作。具体步骤如下：

1）在 Xcode 中，选择 Xcode → Preferences，点击 Accounts。

2）点击（+），选择 Add Repository。

3）输入你的仓库地址，点击 Next。

4）输入密码。

添加好账户以后，选择 Source Control → Check Out 来创建项目的本地工作副本。（如果你使用了本地 Git 仓库，那你无需 check out 当前使用的副本，因为你的本地仓库就是你的 master copy。）

可以在项目导航器中看到文件的源码控制状态，这些状态在文件名右侧以标记（badge）形式展示，如图 13-6 所示。其中，M 代表 Modified（已修改），A 代表 Added（已增加），从这些状态中我们就可以看出哪些文件发生了哪些变化。

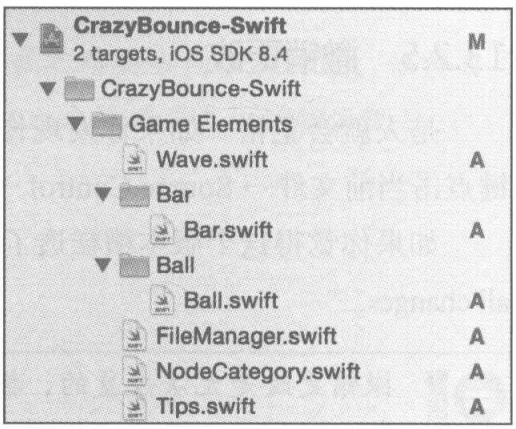


图 13-6　源码修改状态

13.2.3　提交更改

一般的 Git 工作流程是这样：修改过某些文件，然后把这些文件添加都暂缓区，再提交（Commit）到仓库中形成一个版本或快照，最后提交到 Git 服务器上（Push）。而在中间，可能伴随着分支（Branch）管理，分支切换，撤销与合并（Merge）。

如果你满意对文件的更改，可选择 Source Control → Commit 确保这些更改已保存到仓库中。需要提供注释解释你提交的代码的特性。如果你的 Git 仓库托管在服务器上，那么提交操作会将更改添加到你的本地仓库中。

在 commit 完之后，你可能想把自己的代码提交到 Git 服务器上，与他人交流共享，点击 Source Control → push 来推送到服务器上

还可以在 Commit 时就提交到远程服务器。在选择 Source Control → Commit 时，选中 Push to remote 选项，在弹出菜单中指定远程仓库，并点击 Commit Files。

13.2.4　查看更改

选择 View → Version Editor → Show Comparison View 来比较保存在仓库中的不同文件版本。基于仓库内文件所处的位置使用跳转栏来选择文件。每个跳转栏控制着内容面板中的选择。想要展示一个版本，可通过在文件层级中浏览来找到想要的版本，然后点击选中。阴影

区指示版本间的更改。

使用版本时间线基于时间顺序来选择文件版本。点击中间列底部的 Timeline Viewer 图标（）来展示两个编辑面板中的时间轴。在时间轴中上下移动指针来浏览可用的版本。当发现你想用的版本时，可点击左侧或右侧的提示按钮，从而在对应的编辑器面板展示选中的版本。

可以在版本编辑器中编辑当前的工作的文件副本。如果你想要恢复版本间的更改，可以从旧版本中复制代码并粘贴到当前的版本中。

13.2.5 撤销更改

是人就会犯错。如果你发现你编辑错了一个文件，想把它恢复到上一个版本的状态，右键点击当前文件→ Source Control → Discard changes in XXX。

如果你觉得这个版本糟糕透了，想完全回滚到上一个版本。点击 Source Control → Discard all changes。

撤销更改是无法恢复的，撤销更改意味着你所做的所有改动都白费了，代码将完全回到上一个版本。

13.2.6 分支

1. 新建分支

当你使用一个工程一段时间后，你已经有了一个可靠的稳定的代码主体。你可能会想写一些扩展性的功能，或者做一些小实验，但是你又不想影响你现在的项目。这时候，你可以新建分支，选择 Source Control → Working Copy → New Branch，然后在这个分支里写东西，当觉得不好的时候，你可以把这个分支删除掉，对你之前的主分支没有任何影响。当满意所做的更改，则可以将其合并到稳定的代码主体中。

创建了一个新的分支以后，你将默认使用新的分支。新分支包括了所有没有提交的更改。新建的分支储存在本地，如果需要把它提交到服务器上，则需要执行一次 Push 操作。

2. 合并分支

使 用 Source Control → Working Copy → Merge from Branch 和 Source Control → Working Copy → Merge into Branch 来合并两个分支。

Merge 的实质是把两个版本合在一起，然后在当前分支创建一个新的 commit。在你合并分支之前，你需要 Commit 你的工程，如果你在两个分支的同一个文件的同一个地方都做了修改，这时候 merge 就会失败，你需要手动选择这两个分支中的一个版本来解决合并时的冲突。

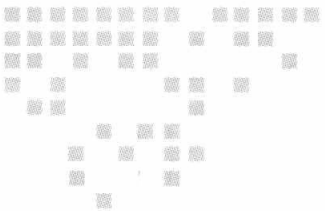
13.2.7　下载别人的版本

多人合作时，当别人写了一个功能，你需要点击 Source Control → Pull 把这个功能同步过来。Pull 可以取回远程主机某个分支的更新，再与本地的指定分支合并。

在你执行 Pull 命令之前，你需要 Commit 当前分支。执行 Pull 命令时，你可能需要手动解决两个分支的冲突。

岁月从不等人，但在版本管理这棵“灵树”下，我们却可以回到那时风轻云淡的年少时光。

可是少年不能再在回忆里更多停留，只因他还有更练武之路要去闯荡和征服。他将自己从回忆里拉回现实，眼前虽然还是寒月一弯，但温暖的记忆已永远保存在他的脑海里……



Chapter 14

第 14 章

实战是提升实力的唯一真理

“你已经习得我帮所有功法，从今天起，你就要下山闯荡了。”师父站在少年常独自饮酒的山头前，郑重地对少年说。

少年提了提肩上的包袱，望着岚风谷熟悉的一切，对师父说道：“弟子谢过师父谆谆教诲！”

“只有自己去这片江湖的深水里，才能知道自己究竟学到了多少啊。今后的路，就看你的造化了……”

只有实战，才能检验真实的功力。到了这里，我们也要将自己做好的 App 打包，放置到 App Store 里供人下载。让我们来看看如何把自己的成果放到实战里接受检验吧。

14.1 基础知识

1. 加入苹果开发者计划

只有加入苹果开发者计划才有可能使用真机调试和发布到 App Store，关于苹果开发者计划的有关内容，本书在第 1 章已经有所介绍。

开发者账户可以在以下链接注册：https://developer.apple.com/cn/programs/

点击右上角的” Enroll”（加入），即可开始加入苹果开发者计划的流程了。

进入之后，网页会告知你相关类型的开发者帐号的区别，点击” Start Your Enrollment”之后，登录你的 Apple ID。

在 Entity Type（实体类型）处，选择一个合适的类型，如图 14-1 所示。

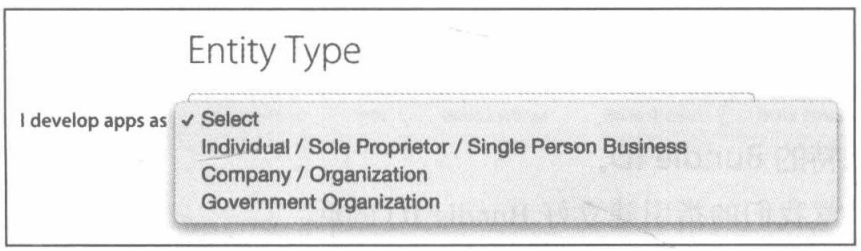


图 14-1　选择实体类型

我们可以选择三种类型：

- Individual / Sole Proprietor / Single Person Business（个人 / 独资企业 / 单人业务）只允许一个开发者账号拥有真机调试、上传应用的权限。
- Company / Organization（公司 / 组织）将允许多个开发者协助开发，拥有真机调试的权限，但是只有主账户才能够上传应用。
- Government Organization（政府部门）是专门为政府部门准备的类型。

注册公司 / 组织需要提供 D-U-N-S 码，这个是苹果用来专门确认企业 / 组织身份的标识码。D-U-N-S 码可以向苹果申请，是完全免费的，但是需要你提供营业执照和法人的相关证件证明。此外，苹果官方人员会通过电话或者邮件和你进行联系。

点击 Continue 之后，输入个人的相关信息，确认之后，交钱，即可完成开发者计划的加入。

个人用户要简单一些，公司、政府部门可能需要额外的一些操作。

2. 在 Xcode 中添加 Apple ID

点击 Xcode → Preferences，在 Account 中添加自己的 Apple ID，如图 14-2 所示。

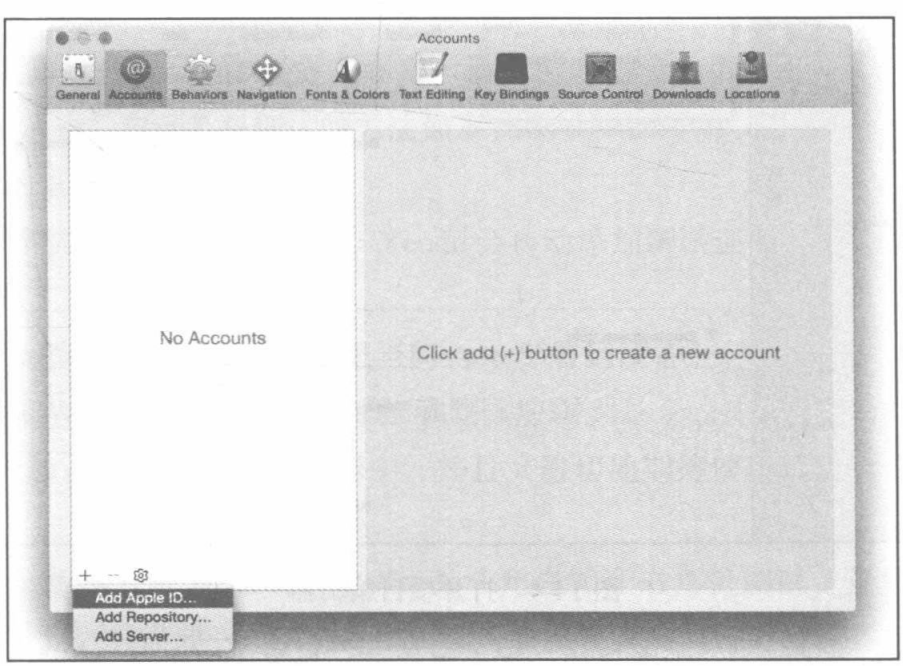


图 14-2　添加 AppleID

如果你还没有 Apple ID 点击 Join a Program 加入，Xcode 将会引导你跳转到一个网页，在这个网站上完成加入计划的操作。

3. 设置一个独特的 Bundle ID。

如果按照第 1 章我们的指引建立好 Bundle ID 的话，那么就无需更改它。一般而言，一个 Bundle ID 可以关联如图 14-3 所示的部分。

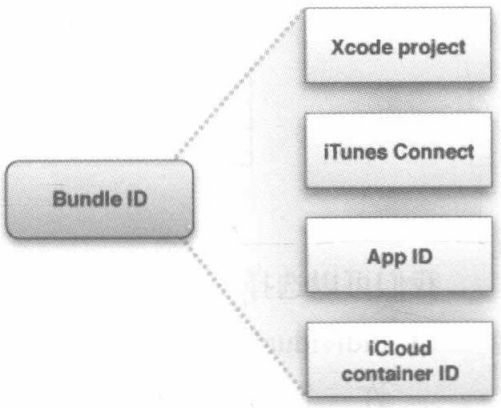


图 14-3 Bundle ID 的关联项

Bundle ID 关联了一个 Xcode 项目，一个特定的 Xcode 项目只有一个 Bundle ID。对于 iTunes Connect、App ID（App Store 上的）以及 iCloud ID 来说，也是一样的，这些服务通过 Bundle ID 来唯一确定一个应用。

14.2 配置 Xcode

App 调试、打包、上传，首先要配置好 Xcode 的开发者账号、Bundle ID 等内容。

如果你在网上看过类似的教程，它们可能会让你复杂地申请无数个证书、配置无数账号。其实一般来说这些操作 Xcode 都可以帮你完成。步骤如下：

1）首先点击 View → Navigators → Show Project Navigator，选择当前的项目，打开项目编辑器，如图 14-4 所示。

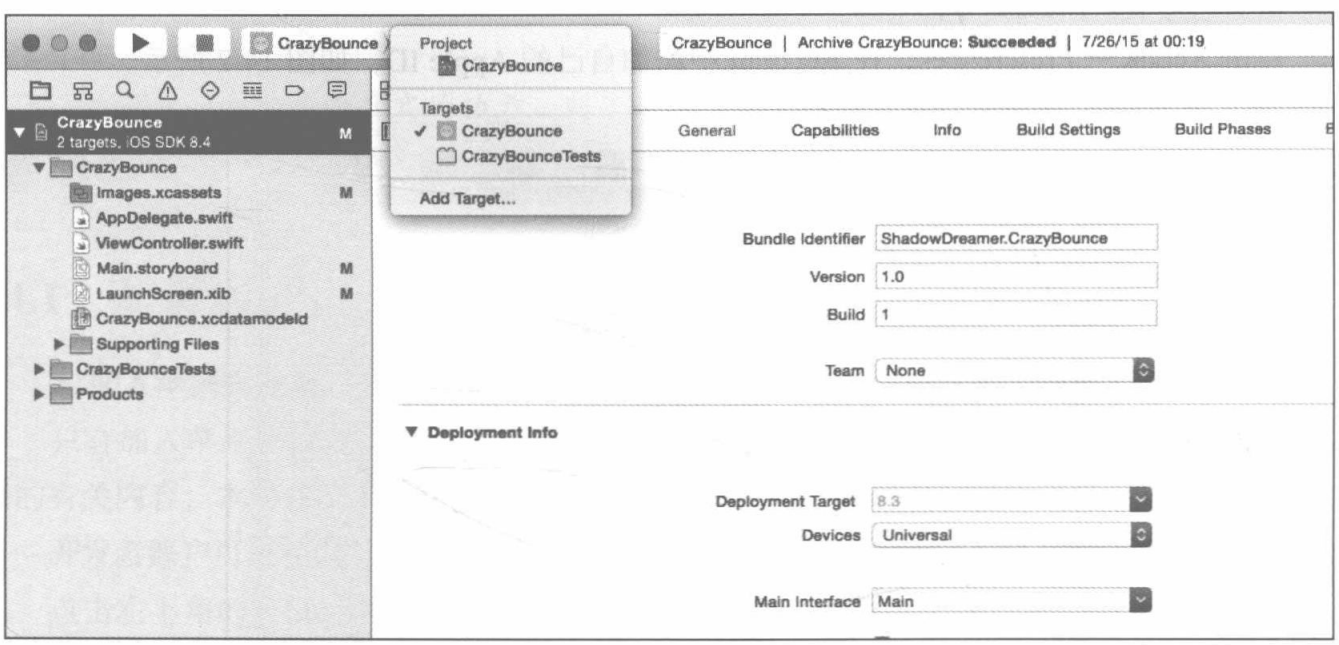


图 14-4 打开项目编辑器

2）接下来我们要在 Team 中选择你的开发者账号。如果出现错误，点击 Fix Issue，Xcode

将会自动给你配置好证书，如图 14-5 所示。

图 14-5　选择开发者账号

3）如果你的 Bundle ID 不符合要求，那么在项目编辑器中的“General”标签中配置项目的 Bundle ID 前缀。然后在项目编辑器中的“info”中，找到“Bundle identifier”编辑 Bundle identifier。

4）最后配置好 App 的版本号、Icon 等其他属性就大功告成了。

14.3　启用真机调试

苹果的真机测试比 Android 要复杂好多，最主要的是要申请 iOS 开发者账号，否则只能在虚拟机中测试 App。启动真机调试的方法如下：

1）连接你的 IOS 设备。

2）在项目导航栏中选择你的设备。Xcode 会自动添加调试证书，把你选择的设备注册成开发者模式。

3）点击 Run 按钮，Xcode 会自动把当前 App 部署到设备上。

4）你可以在 Window → Device 中添加和删除应用。

如果你尚未加入一个开发者计划当中，并且又迫切地期待使用真机调试的话，那么有两个选择：

- 将 Xcode 版本升级为 Xcode 7，Xcode 7 允许无证书真机调试，开发者无需加入一个开发者计划，只需要一个 Apple ID，即可直接在真实设备上编译并运行应用。
- 需要两个相关证书，开发者需要某个加入到开发者计划的组织或个人提供的相关证

书，一个 Certificates 以及一个 Provisioning Profiles。

生成和安装证书的方法如下：

1）进入苹果开发者网站：http://developer.apple.com，登录 member center（成员中心）。

2）单击页面中的 Certificates, Identifiers & Profiles。

3）选择 Devices，单击“+”号按钮，注册一个新的设备。

4）输入你想要的 Device 名字以及其 UDID 号码后即可完成注册，如图 14-6 所示。

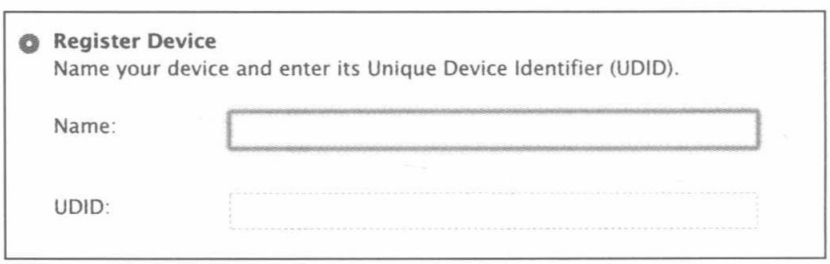


图 14-6 注册 Device

5）UDID 需要 Mac 连上真机后，打开 iTunes 软件，选中你的设备后，在“序列号”一栏中单击一下，即可看见 UDID 的信息，如图 14-7 所示。

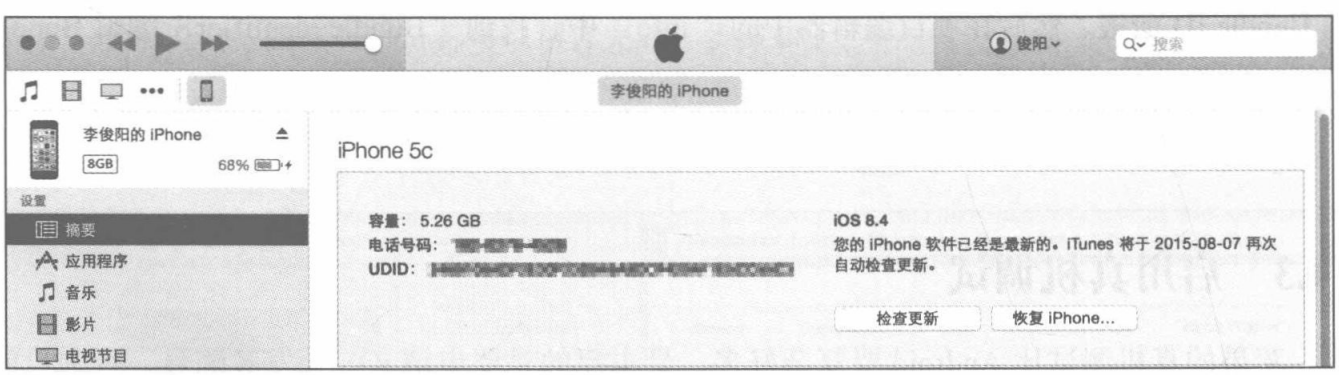


图 14-7 查看 UDID

6）在 Certificates 中，选择一个合适的证书，将其下载下来。此外，也可以自己新建一个证书。要注意的是，开发和发布的证书是分开来的。

7）安装下载下来的 .cer 证书。

8）在 Provisioning Profiles 中，选择一个合适的证书，将其下载下来。此外，也可以自己新建一个证书。要注意的是，开发和发布的证书是分开来的。

9）安装下载下来的 .mobileprovision 证书。

10）来到 Xcode 的项目设置的 Build Setting 中，找到 Code Signing Identity，将其替换为刚才安装过的 .cer 证书，然后将 Provisioning Profile 替换为刚才安装过的 .mobileprovision 证书，即可完成真机调试的配置。

14.4　把应用提交到 App Store

把应用提交到 App Store 需要几大步，如下所示。

1. 注册 App ID

1）打开“开发者门户网站”https://developer.apple.com/account/ios/identifiers/ bundle/bundleCreate.action

2）点击 App IDs 注册一个 App ID，注意 Bundle ID 要和 Xcode 上的相对应，如图 14-8 所示。

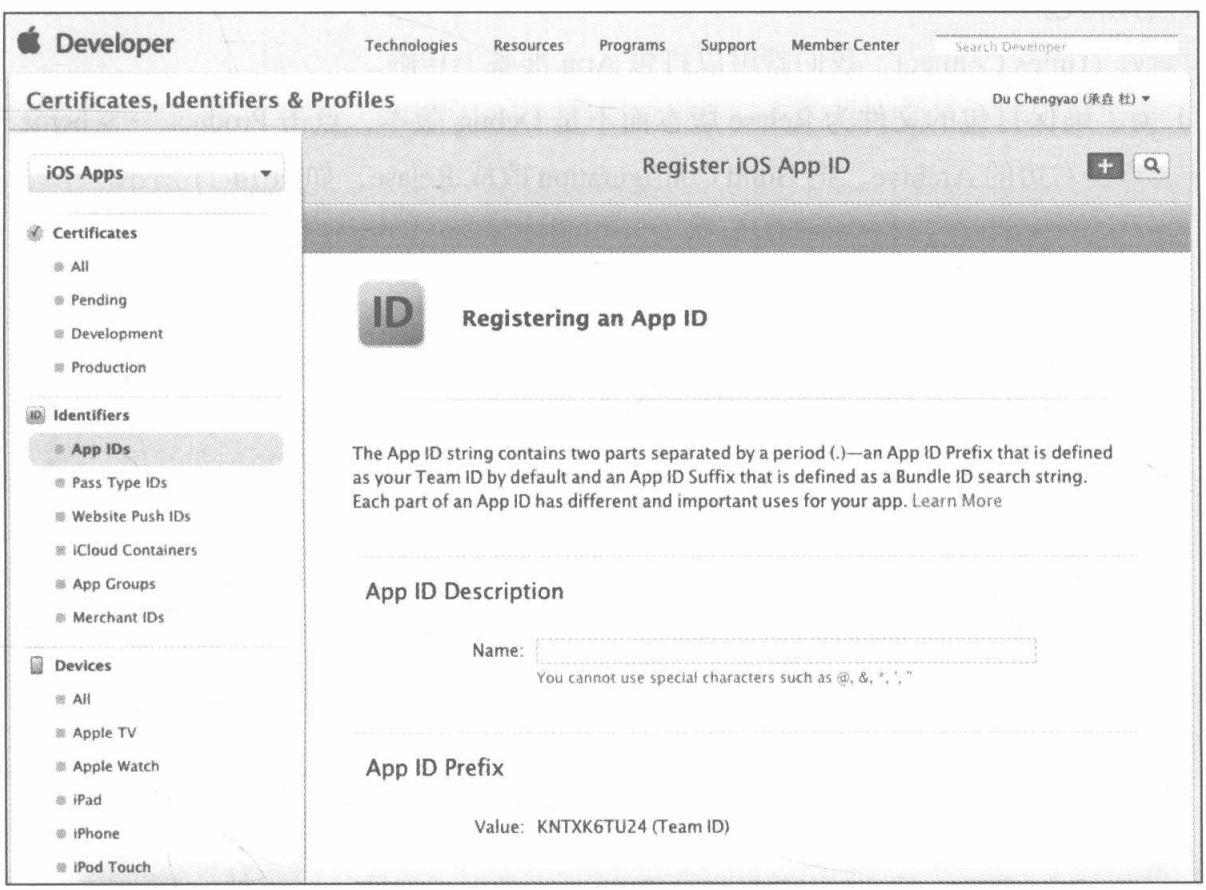


图 14-8　注册 App ID

2. 设置 iTunes Connect

想要在 AppStore 发布应用，必须在 iTunes Connect 中添加应用程序的资料。

1）登录到 iTunes Connect（https://itunesconnect.apple.com/）。

2）点击“我的 App”，进入新的页面后，点击左上角的“+”新建一个 App，如图 14-9 所示。

图 14-9　新建 App

3）填好所有的必填选项后保存，此时如果你点击了“提交以供审核”将会报错，如图 14-10 所示。因为你还没有上传

你打包好的 App。

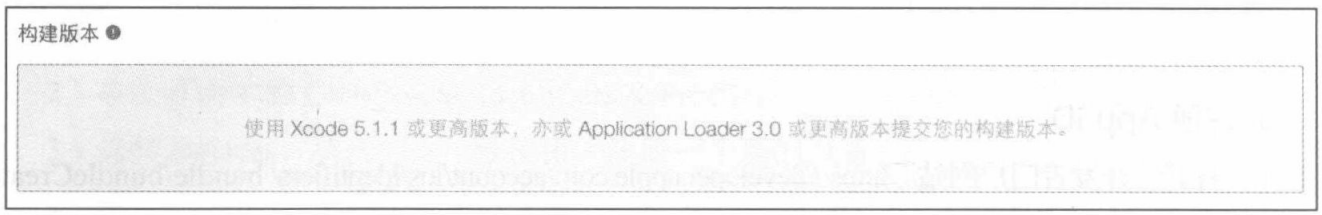


图 14-10　提交报错

3. 应用打包

设置好 iTunes Connect，我们就可以打包 App 准备上传啦。

1）为了确保打包的文件为 Relase 版本而不是 Debug 版本，点击 Product → Scheme → Edit Scheme。点击右边的 Archive，把 Build Configuration 改成 Relase，如图 14-11 所示。

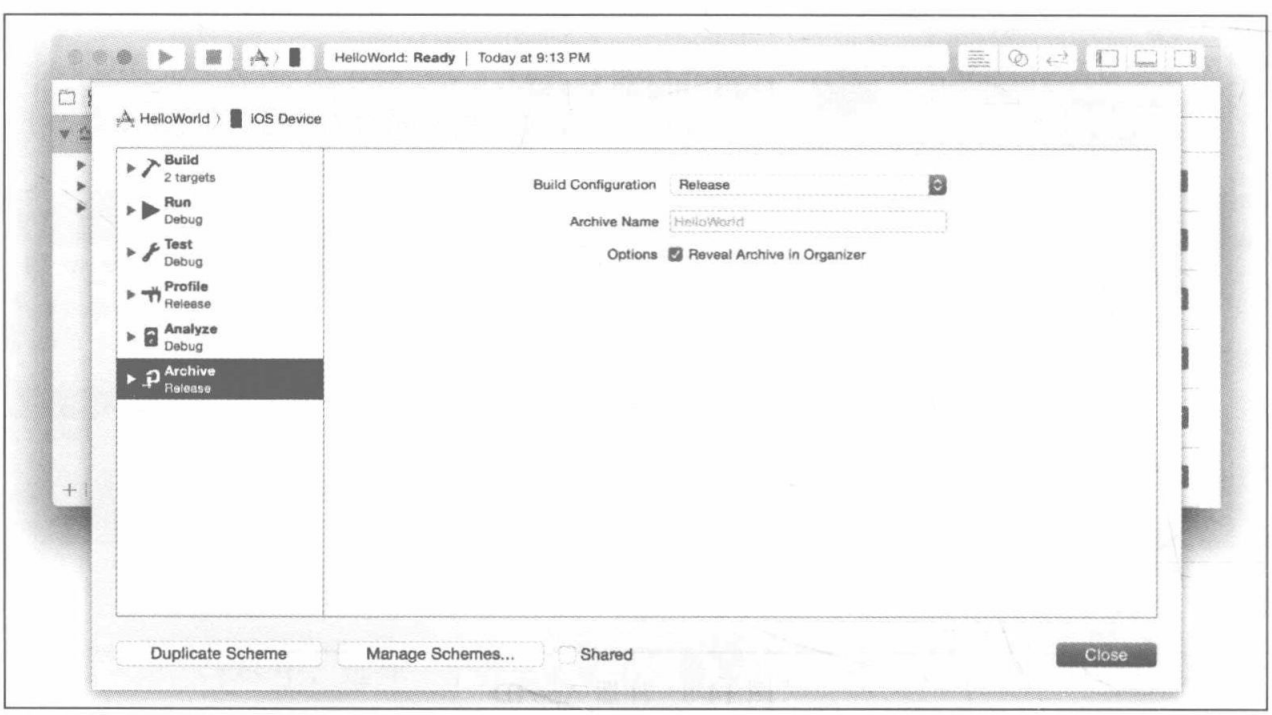


图 14-11　修改打包模式

2）点击 Product → Archive，创建一个存档。Xcode 会进行一些初步验证测试，当项目出现问题时会弹出警告信息。如果出现了问题，依次解决问题，并重新创建存档。

如果 Archive 按钮为灰色，请把部署方式由模拟器改为 ios Device

3）点击 Validate 验证此存档，Xcode 会自动生成 distribution certificate 和 distribution provisioning profile，并生成验证报告，如图 14-12 所示。如果出现了问题，修改后重新验证。

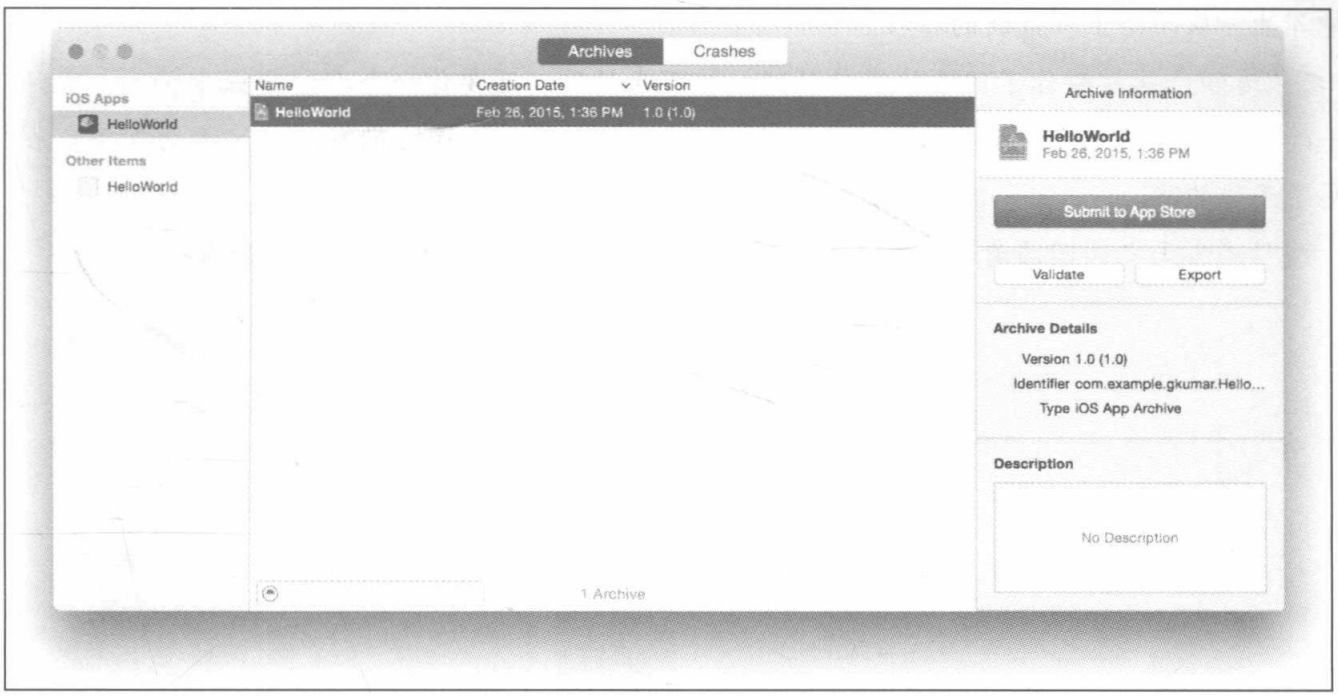


图 14-12　应用验证

4）如果验证没问题，就可以点击 Submit to App Store 直接上传了，Xcode 会帮你生成证书，并把你打包好的 App 上传到你的 iTunes Connect。

提示　还可以导出 ipa 文件，用 Application Loader 上传，具体方法见附录 B。

4. 提交以供审核

上传完毕后等待几分钟的时间，刷新 iTunes Connect 的页面，会出现你刚刚上传的 App，如图 14-13 所示。

最后点击“提交以供审核”。大约一周左右的时间苹果公司将会通知你是否通过审核，如果通过了审核就可以在 AppStore 里面看到了。

图 14-13　完成构建的版本

提示　如果提示你没有通过审核，可以在 iTunes Connect 中下载 Crash Reports，查看到底是哪个地方出错了，修改后重新上传。

一壶浊酒，师徒二人喜相逢，这些时日的艰辛都付与笑谈中。江湖多少事，书中驰疆纵马与共，踏月随风，佐酒倾情。天涯任你我漂萍，楚色云淡江清。

仿佛还能依稀看见昨日岚风谷上，站着一个血气方刚的少年。岚风谷的风吹起他的鬓发，他凝望着星辰，内心向往着那编程的江湖。如今，他已深入其中，在这片江湖中即将荡尽乾

坤浑浊，从而踏上一条新的征程。

“为师已传授给你了门派的武功精髓，接下来的路，还要你一个人慢慢探索……”大师在一阵清清笛声中化作一缕青烟，转眼，眼前就只剩下漫天飘舞的花瓣，为迷途的少年良辰指明了天边的路。

只闻青山之间回荡着老者爽朗的笑声：“青山不改，绿水长流，咱们后会有期！”

随身锦囊——附录

Appendix A

附录 A

Xcode 小技巧

A.1 常用快捷键

Xcode 中许多快捷键可以使编程变得更为方便、快速，熟练掌握这些常用的快捷键，不但可以在外行面前展现“强大”的能力，而且还确实能够提高工作效率。

图 A-1 以最直观的方式列出了 Xcode 的快捷键，其对 Xcode 进行了分区，并且集中地列出了相关区域中使用到的快捷键。

表 A-1 是对图 A-1 的详细说明。

表 A-1 Xcode 常用快捷键及说明

所属功能	名　称	快 捷 键	作　用
搜索	文件内搜索	Command + F	在某文件中进行搜索
	文件内搜索与替换	Command + Option + F	在某文件中进行搜索与替换
	项目内搜索	Command + Shift + F	打开搜索导航器，在项目中进行搜索
	项目内搜索与替换	Command + Shift + Option + F	打开搜索导航器，在项目中进行搜索与替换
标签页	新建标签页	Command + T	新建一个主编辑器区域的标签页
	上一个标签页	Command + {	跳转到上一个标签页
	下一个标签页	Command + }	跳转到下一个标签页
导航	欢迎界面	Command + Shift + 1	打开 Xcode 的欢迎界面
	移动光标到…编辑器	Command + J	打开编辑器选择窗口，可以从中跳转到选择的编辑器页面

（续）

所属功能	名　称	快 捷 键	作　用
导航	下一个相关文件	Command + Control + ↑	跳转到当前文件所对应的下一个关联文件，比如说 .m 跳转到 .h 文件
	上一个相关文件	Command + Control + ↓	跳转到当前文件所对应的上一个关联文件，比如说 .h 跳转到 .m 文件
	下一个最近文件	Command + Control + →	相当于“快进”，跳转到“后退”之前的文件中
	上一个最近文件	Command + Control + ←	相当于“后退”，跳转到之前打开的文件中
	符号定义	Command + 左键单击	进入所选符号的定义文件当中
	跳转到某行	Command + L	打开一个对话框，从中可以输入要跳转的行数，即可完成当前文件内的跳转
	折叠代码片段	Command + Option + ←	将光标所在的代码片段进行折叠
	展开代码片段	Command + Option + →	将光标所在的已折叠的代码片段展开
编辑	显示助理编辑器	Command + Option + Return	打开助理编辑器
	隐藏助理编辑器	Command + Return	隐藏助理编辑器
	自动完成	Control + 空格	打开或者关闭代码自动完成列表
	全局修改数据	Command + Control + E	选中某一符号，修改当前文件中该符号的名称
	向内缩进	Command +]	将光标所在行向内（右）缩进，默认是 4 个空格
	向外缩进	Command + [	将光标所在行向外（左）缩进，默认是 4 个空格
	自动缩进	Control + I	自动对当前文件进行排版缩进
	注释 / 取消注释	Command + /	对所选语句进行快速注释或者取消注释
	语句上移	Command + Option + [	将光标所在语句向上移动一行
	语句下移	Command + Option +]	将光标所在语句向下移动一行
导航器	显示 / 隐藏	Command + 0	显示或者隐藏导航器
	切换标签页	Command + 1 ～ 8	依次在项目导航器、符号导航器、搜索导航器等导航器之间切换
	在助理编辑器中打开文件	Option + 左键单击	在导航器中对某一文件执行该操作，将让其在助理编辑器中打开
	在新窗口中打开	左键双击	在一个新窗口中打开文件
	由用户决定在何处打开	Shift + Option + 左键单击	打开编辑器选择窗口，可以在这里决定是用新窗口代开，还是用助理编辑器打开
调试	调试区域	Command + Shift + Y	打开或者隐藏调试区域
	下一个问题	Command + ‘	跳转到下一个问题所在的地方
	上一个问题	Command + “	跳转到上一个问题所在的地方
	增加断点	Command + |	在光标所在行处，添加一个断点
调试	切换断点状态	Command + Y	激活或者关闭断点
	清空控制台	Command + K	光标在控制台中时，可以清空控制台里面的所有内容

（续）

所属功能	名　　称	快　捷　键	作　　用
文档	光标所在的快速帮助	Command + Control + ?	打开光标所在符号的快速帮助界面
	选中符号的快速帮助	Option + 左键单击	打开所选符号的快速帮助界面
	打开选中对象的帮助文档	Command + Option + Control + /	打开光标所在符号的帮助文档
	选中符号的帮助文档	Option + 左键双击	打开所选符号的帮助文档
实用区域	显示 / 隐藏	Command + Option + 0	显示或者隐藏实用区域
	切换标签页	Command + Option + 1~9	依次在各种实用区域标签页之间切换

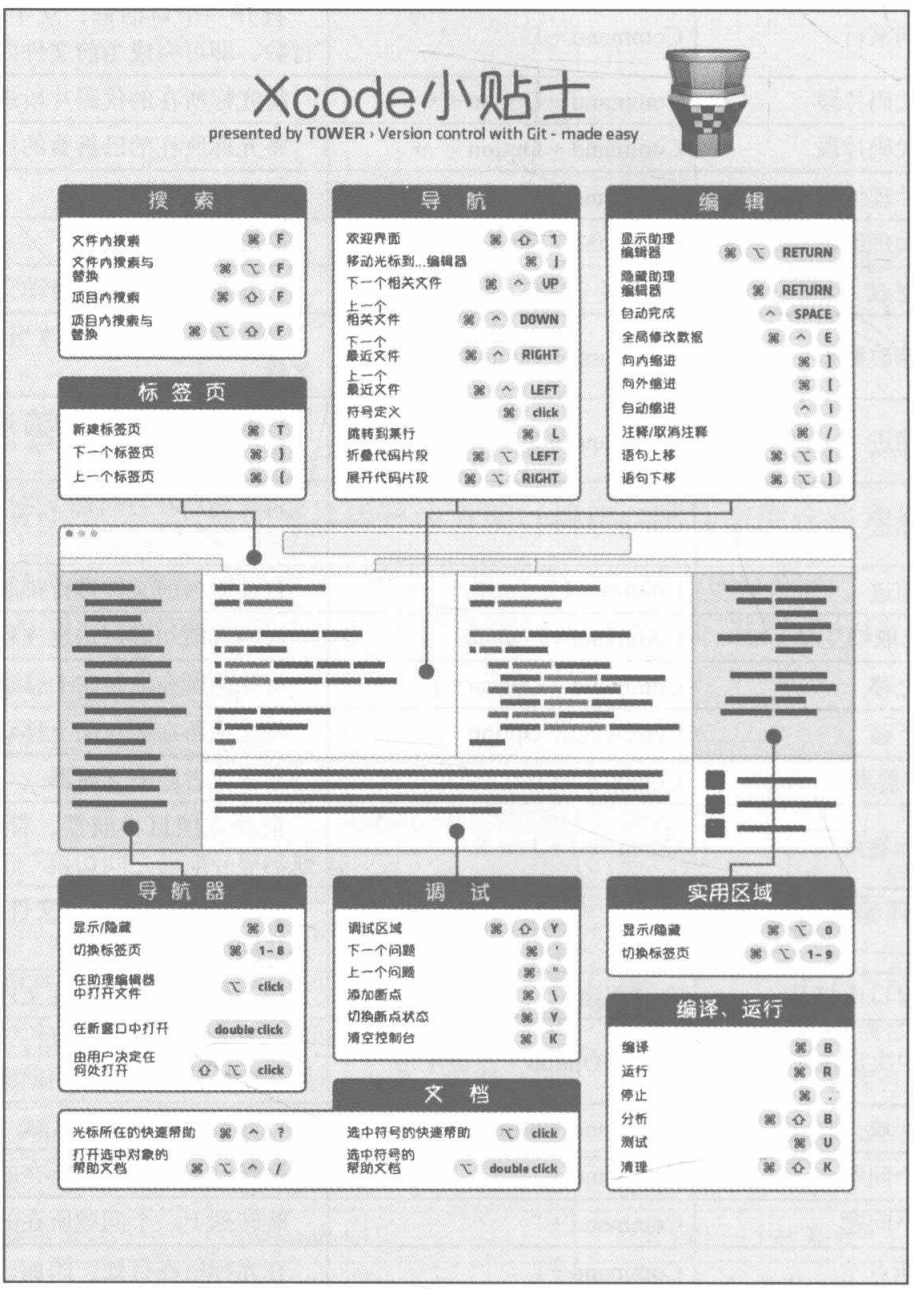


图 A-1　常用的 Xcode 快捷键

注：该图来源于 TOWER，地址：http://www.git-tower.com/blog/xcode-cheat-sheet/。

A.2　代码片段

何为代码片段？我们在 Xcode（.m 文件的某个方法中）中输入 if-else 指令，这时我们会发现 Xcode 显示了如图 A-2 所示的自动完成窗口。

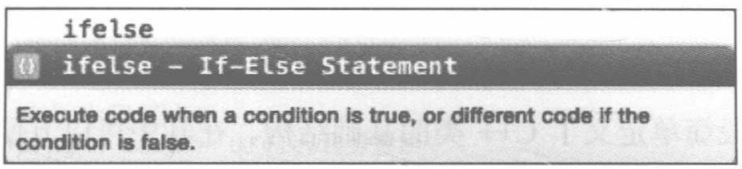


图 A-2　自动完成窗口

敲下回车后，Xcode 就能够提供带有占位符的代码结构。使用这个功能可以节约大量的时间，让开发者更轻松。

Xcode 中默认保存了许多预先定义好的、可以在工程中使用的代码片段。打开“实用工具”区域，然后在最下方的模板区域中打开代码片段（Code Snippets）选项卡，如图 A-3 所示。

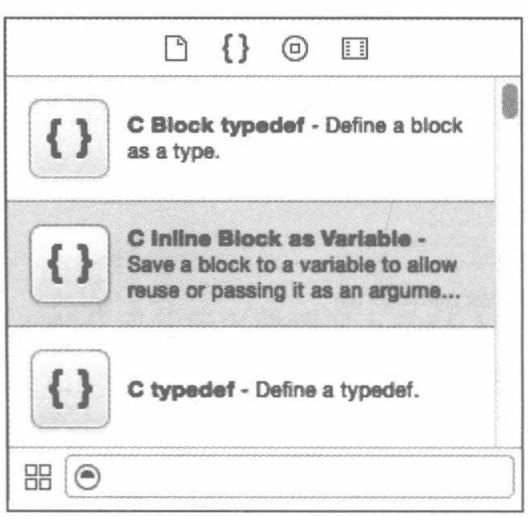


图 A-3　代码片段选项卡

在这里面有许多非常好用的标准片段，比如类定义、if-else 语句，等等。

注意　不是所有的代码片段都适合于 Swift 和 Objective-C，这里面还有 C 和 C++ 的代码段。此外，某个代码片段也只能适用于某个或某类语言，比如，上面所述的 if-else 语句就不能够在 Swift 中使用。

代码片段还有一个好处，它不仅是一个所谓的“宏定义”语句而已，它存在一个上下文环境，也就是说，代码片段会自动识别其所在环境，阻止其放置在了不对的地方。比如你打算在 @implement 声明中的方法外键入 if-else 语句，那么 Xcode 是不会弹出提示的。

除了使用“自动完成”的方法来实现代码片段的方法外，还可以直接将代码片段选项卡

中的代码片段直接拖到代码编辑器里面。这是代码片段最简单的使用方法。

下一节，我们将简要介绍所有的代码片段。

A.2.1 快速输入代码

1. 只适用于 C++ 语言

C++ Class Declaration

这个代码片段简单定义了 C++ 类的基础结构，在其中可以方便快速地定义类中的实例变量和成员函数。输入 classdef，自动完成后即可完成片段添加。

代码原型如下：

```
class class name {
  instance variables

public:
  member functions
};
```

C++ Class Template

这个代码片段简单定义了一个 C++ 类模板的基础结构。输入 templateclass，自动完成后即可完成片段添加。

代码原型如下：

```
template <template parameters>
class class name {
  instance variables

public:
  member functions
};
```

C++ Function Template

这个代码片段简单定义了一个 C++ 函数模板的基础结构。输入 templatefunction，自动完成后即可完成片段添加。

代码原型如下：

```
template <template parameters>
return type function name(function parameters) {
    statements
}
```

C++ Namespace Definition

这个代码片段简单定义了一个新的命名空间，还可以扩展一个已存在的命名空间。输入 namespace，自动完成后即可完成片段添加。

代码原型如下：

```
namespace namespace name {
    declarations
}
```

C++ Try/Catch Block

这个代码片段只能够在函数里面使用，定义了一个 Try-Catch 代码结构。输入 try，自动完成后即可完成片段添加。

代码原型如下：

```
try {
    statements
} catch (catch parameter) {
    statements
}
```

C++ Using Directive

这个代码片段很简单，也是很多 C++ 程序所拥有的语句声明，在此不加以赘述。输入 using，自动完成后即可完成片段添加。

代码原型如下：

```
using namespace namespace name
```

2. 只适用于 Objective-C 语言

只适用于 Objective-C 语言的代码片段多数情况下是对 API 某些方法的快速调用。因此，关于这些代码片段的具体作用和用法，本节不会耗费太多笔墨来解释。对这些代码片段感兴趣的读者，可以自行前往苹果开发者网站中进行查询。

Core Data -awakeFromFetch Method

输入 awakeFromFetch，自动完成后即可完成片段添加。

代码原型如下：

```
- (void)awakeFromFetch
{
    [super awakeFromFetch];
    code to be executed after the receiver has been fetched
}
```

这段代码只在使用了 Core Data 技术的时候有用，此方法在创建 NSManagedObject 对象后调用。

Core Data -awakeFromInsert Method

输入 awakeFromInsert，自动完成后即可完成片段添加。

代码原型如下：

```
- (void)awakeFromInsert
{
    [super awakeFromInsert];
    code to be executed the receiver is first inserted into a managed object context
}
```

这段代码同样也只在使用了 Core Data 技术时有用，此方法将在数据对象插入到管理对象上下文的时候调用。

Core Data Fetch

输入 fetch，自动完成后即可完成片段添加。

这段代码十分冗长，因此这里不予展示。这段代码的主要功能是实现了一个完整的检索（Fetch）操作，同样也只在使用了 Core Data 技术时有用。

Core Data Property Accessors (Object Type)

这个代码片段似乎没有快捷输入的方式，名为“Core Data 属性访问器（对象类型）”。它主要是为 Core Data 中的对象类型的属性建立一个 getter 和 setter 方法。

Core Data Property Accessors (Scalar Type)

这个代码片段似乎没有快捷输入的方式，名为“Core Data 属性访问器（标量类型）”。它主要是为 Core Data 中的标量类型的属性建立一个 getter 和 setter 方法，所谓“标量类型”，就是指整型、布尔型等这些基本数据类型。

Core Data Property Validation

这个代码片段似乎没有快捷输入的方式，名为“Core Data 属性认证”。它主要是用来对 Core Data 属性实现认证，所谓认证，就是确认这个属性是否满足所规定的要求。

Core Data To-Many Relationship Accessors

这个代码片段似乎没有快捷输入的方式，名为“Core Data 对多关系访问器”。它主要是用来为一对多或者多对多关系实现添加和删除元素的操作。

Enumerate Index Set

这个代码片段似乎没有快捷输入的方式，名为“枚举索引集”。它主要是对 NSIndexSet 索引集合进行枚举。

代码原型如下：

```
NSUInteger index = [indexSet firstIndex];
while (index != NSNotFound) {
    statements
    index = [indexSet indexGreaterThanIndex:index];
}
```

Enumerate Index Set In Reverse

与上一个代码片段类似，只不过枚举的方向相反了。

Enumerated Type Declaration

输入 enumdef，自动完成后即可完成片段添加。

代码原型如下：

```
typedef enum : NSUInteger {
        MyEnumValueA,
        MyEnumValueB,
        MyEnumValueC,
    } MyEnum;
```

这个代码片段定义了一个经典 C 风格的枚举，相信大家都不会陌生。

Enumerated Type Declaration(NS_ENUM)

输入 nsenum，自动完成后即可完成片段添加。

代码原型如下：

```
typedef NS_ENUM(NSUInteger, MyEnum) {
    MyEnumValueA,
    MyEnumValueB,
    MyEnumValueC,
};
```

这个枚举风格是 iOS 6 之后引入的宏，实际上这已经成为了 Objective-C 经典风格的枚举形式了。

Enumerated Type Declaration(NS_OPTIONS)

输入 nsoptions，自动完成后即可完成片段添加。

代码原型如下：

```
typedef NS_OPTIONS(NSUInteger, MyEnum) {
       MyEnumValueA = 1 << 0,
       MyEnumValueB = 1 << 1,
       MyEnumValueC = 1 << 2,
    };
```

这个枚举风格一般用来定义位移相关操作的枚举值。

Objective-C -compare: Method

输入 compare，自动完成后即可完成片段添加。

代码原型如下：

```
- (NSComparisonResult)compare:(id)other
{
    return comparison expression;
}
```

这个方法用于逐个比较字符，比较的结果放在 NSComparisonResult 中。

Objective-C -dealloc Method

输入 dealloc，自动完成后即可完成片段添加。

代码原型如下：

```
- (void)dealloc
{
    statements
}
```

这个方法很明了，内存回收方法。

Objective-C debugDescription Method

输入 debugDescription，自动完成后即可完成片段添加。

代码原型如下：

```
- (NSString *)debugDescription
{
    return [NSString stringWithFormat:@"<%@: %p> additional format string",
         [self class], self, additional arguments];
}
```

这个方法是开发者在调试器中以控制台命令打印对象时才调用的。

Objective-C -description Method

输入 description，自动完成后即可完成片段添加。

代码原型如下：

```
- (NSString *)description
{
    return [NSString stringWithFormat:@"format string", arguments];
}
```

这个方法是用于更改打印信息的，覆盖此方法会改变 NSLog 方法输出的内容。

Objective-C encodeWithCoder: Method

输入 encodeWithCoder，自动完成后即可完成片段添加。

代码原型如下：

```
- (void)encodeWithCoder:(NSCoder *)coder
{
    [super encodeWithCoder:coder];
    statements
}
```

这个方法一般用于实现 NSCoding 协议的类当中，用于归档。

Objective-C -init Method

输入 init，自动完成后即可完成片段添加。

代码原型如下：

```
- (instancetype)init
{
    self = [super init];
    if (self) {
        statements
    }
    return self;
}
```

这个方法自然不用多说，用来初始化方法。

Objective-C -initWithCoder: Method

输入 initWithCoder，自动完成后即可完成片段添加。

代码原型如下：

```
- (instancetype)initWithCoder:(NSCoder *)coder
{
    self = [super initWithCoder:coder];
    if (self) {
        statements
    }
    return self;
}
```

与 encodeWithCoder 遥相呼应，用于解包操作。

Objective-C -initWithFrame: Method

输入 initWithFrame，自动完成后即可完成片段添加。

代码原型如下：

```
- (instancetype)initWithFrame:(NSRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        statements
    }
    return self;
}
```

对 UIView 类的对象尺寸的默认初始化行为进行更正。

Objective-C -isEqual: and -hash Methods

输入 isEqual，自动完成后即可完成片段添加。

代码原型如下：

```
- (BOOL)isEqual:(id)other
{
    if (other == self) {
        return YES;
    } else if (![super isEqual:other]) {
        return NO;
    } else {
        return comparison expression;
    }
}
```

自定义实现判断元素或对象是否相等的方法。

Objective-C +initialize Method

输入 initialize，自动完成后即可完成片段添加。

代码原型如下：

```
+ (void)initialize
{
    if (self == [ClassName class]) {
        statements
    }
}
```

执行初始化的类方法。

Objective-C Autoreleasing Block

输入 @autoreleasepool，自动完成后即可完成片段添加。

代码原型如下：

```
@autoreleasepool {
    statements
}
```

实现自动释放池的声明。

Objective-C Category

输入 @interface-category，自动完成后即可完成片段添加。

代码原型如下：

```
@interface class name (category name)

@end
```

Objective-C 类别的接口声明。

Objective-C Category Implementation

输入 @implementation-category，自动完成后即可完成片段添加。

代码原型如下：

```
@implementation class (category name)

methods

@end
```

与上面那个方法遥相呼应，类别的实现方法。

Objective-C Class Declaration

输入 @interface，自动完成后即可完成片段添加。

代码原型如下：

```
@interface class name : superclass

@end
```

这个很明显，实现类的接口声明。

Objective-C Class Extension

输入 @interface-extension，自动完成后即可完成片段添加。

代码原型如下：

```
@interface class name ()

@end
```

这个是类的扩展声明。

Objective-C Class Implementation

输入 @implementation，自动完成后即可完成片段添加。

代码原型如下：

```
@implementation class

methods

@end
```

与类的接口声明遥相呼应，是类的实现方法。

Objective-C Fast Enumeration

输入 forin，自动完成后即可完成片段添加。

代码原型如下：

```
for (type *object in collection) {
        statements
    }
```

对 Objective-C 集合类的快速枚举方法，Objective-C 中的集合类包括 NSArray、NSDic-

tionary、NSSet 等。

Objective-C KVO: Observe Value For Keypath

输入 observeValueForKeyPath，自动完成后即可完成片段添加。

所谓 KVO，指的是“Key-Value Observing”，即“键 – 值观察”机制。简单来说，就是当指定的对象属性被修改后，允许对象接受到通知的机制。

这个方法主要用于实现观察者如何相应变化的消息。

Objective-C KVO: Values affecting key

输入 keyPathsForValuesAffecting，自动完成后即可完成片段添加。

这个方法可以让一个键观察多个属性值的改变情况。

Objective-C Protocol Definition

输入 @protocol，自动完成后即可完成片段添加。

代码原型如下：

```
@protocol protocol name <NSObject>

methods

@end
```

这个方法也很明了，声明一个协议。

Objective-C Try-Catch-Finally Block

输入 @try，自动完成后即可完成片段添加。

代码原型如下：

```
@try {
    可能会抛出异常的代码段
    }
    @catch (NSException *exception) {
        处理在@try代码块中抛出的异常
    }
    @finally {
        无论是否抛出异常，这里的代码都会被执行
    }
```

这段代码和绝大多数有异常捕获机制的语言一样，主要是用来捕获异常、处理异常和抛出异常的。

Test Method

输入 test，自动完成后即可完成片段添加。

代码原型如下：

```
- (void)testName {
```

```
    statements
}
```

这个方法一般用在单元测试包当中，用来测试某个类的某个功能。

3. 只适用于 Swift 语言

Swift Class

输入 class，自动完成后即可完成片段添加。

代码原型如下：

```
class name {
    属性和方法
}
```

Swift 语言的类声明。

Swift Struct

输入 struct，自动完成后即可完成片段添加。

代码原型如下：

```
struct name {
    属性和方法
}
```

Swift 语言的结构体声明。

Swift Subclass

输入 class，自动完成后即可完成片段添加。

代码原型如下：

```
class name: super class {
    属性和方法
}
```

带有父类继承功能的 Swift 语言的子类声明。

4. 适用于 C 类语言

C Block typedef

这个代码片段主要是定义了一个基础的闭包别名（typedef），定义完之后，我们就可以在函数、方法、类的参数类型定义中，直接使用这个别名来代替闭包参数类型。输入 typedefBlock，自动完成后即可完成片段添加。

代码片段原型如下：

```
typedef returnType(^name)(arguments);
```

一般来说，这项功能大多数都用于 Objective-C 编写的程序，但由于闭包是苹果为

C、C++ 和 Objective-C 所提供的一个扩展，因此我们将其归类于 C 类语言之下。

关于闭包的更多知识，请参阅其他资料。

C Inline Block as Variable

这个片段将一个完整的闭包声明包装成了一个变量，这样就可以方便重用，甚至可以将其当作参数进行传递。这个片段只能够手动添加。

代码片段原型如下：

```
returnType(^blockName)(parameterTypes) = ^(parameters) {
    statements
};
```

C typedef

这个代码片段就是一个简单的别名定义语句，为复杂的声明定义一个简单的名称。输入 typedef，自动完成后即可完成片段添加。

代码片段原型如下：

```
typedef existing new;
```

Do-While Statement

输入 dowhile，自动完成后即可完成片段添加。

代码原型如下：

```
do {
    statements
} while (condition);
```

For Statement

输入 for，自动完成后即可完成片段添加。

代码原型如下：

```
for (initialization; condition; increment) {
    statements
}
```

GCD: Dispatch After

输入 dispatch_after，自动完成后即可完成片段添加。

GCD 是协助并发代码在多核硬件上执行的库之一，在 Xcode 上，它其实是 libdispatch 的指代。简要来说，GCD 主要用来执行并行操作。

当我们需要延迟执行某个动作的时候，就可以使用 dispatch_after 来完成，它可以让闭包在等待一段时间后加入到主运行队列当中，完成操作。

关于 GCD 的更多内容，在此就不再赘述了。

GCD: Dispatch Once

输入 dispatch_once，自动完成后即可完成片段添加。

dispatch_once 函数也属于 GCD 的一部分，该函数里面的代码只会被执行一次，并且保证线程安全。

GCD: Dispatch Source(Timer)

输入 dispatch_source，自动完成后即可完成片段添加。

dispatch_source 用来监视某些类型事件，当这些事件发生后，它就自行将闭包加入到主运行队列当中，完成操作。

If Statement

输入 if，自动完成后即可完成片段添加。

代码原型如下：

```
if (condition) {
    statements
}
```

If-Else Statement

输入 ifelse，自动完成后即可完成片段添加。

代码原型如下：

```
if (condition) {
    statements
} else {
    statements
}
```

Struct Declaration

输入 structdef，自动完成后即可完成片段添加。

代码原型如下：

```
struct structname {
    struct fields
};
```

声明了一个 C 语言的结构体。

Switch Statement

输入 switch，自动完成后即可完成片段添加。

代码原型如下：

```
switch (expression) {
    case constant:
        statements
```

```
            break;

        default:
            break;
    }
```

Union Declaration

输入 uniondef，自动完成后即可完成片段添加。

代码原型如下：

```
union union name {
    union fields
};
```

While Statement

输入 while，自动完成后即可完成片段添加。

代码原型如下：

```
while (condition) {
    statements
}
```

A.2.2 自定义代码片段

当然，Xcode 自身提供的代码片段往往不能满足广大开发者的需求，尤其是使用 Swift 语言的开发者，能够使用的代码片段更是少得可怜。所幸的是，Xcode 提供了一个十分强大的功能，它允许开发者自己建立并管理一些可重用的代码，随时随地在需要的时候使用。

我们下面就来以 Swift 的 Extension 为例，创建一个自定义的代码片段，来感受一下这个功能的强大特性。

在某个 .swift 文件中，创建一个如下所示的 extension 扩展：

```
extension <#ClassName#> {
    var <#PropertyName#>: <#PropertyType#> {
        get {
            return <#Value#>
        }
        set {
            <#Statement#>
        }
    }
}
```

这个方法提供了一个带有计算属性的扩展，然后选中这个方法，长按然后将它拖到“代码片段”库当中，如图 A-4 所示。这时，代码片段中就会弹出如图 A-5 所示的弹出框（如果

弹出框没有显示，可双击代码片段中新出现的 My Code Snippet）。

在这个弹出框中，我们就可以看到刚刚自定义的代码片段的名称和具体代码段，但是这还远远达不到我们的要求。因此，我们单击弹出框的 Edit 按钮，如图 A-6 所示。具体选项说明如下。

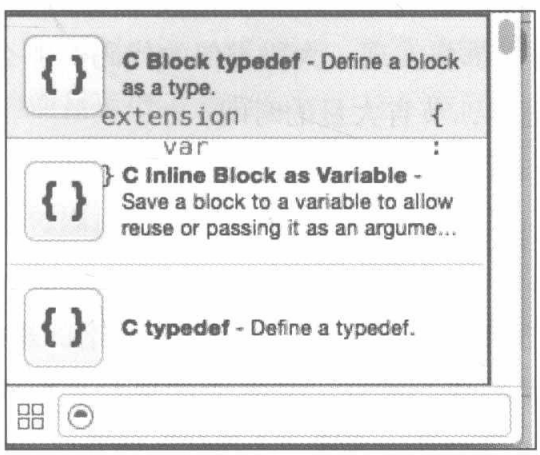


图 A-4　将代码拖入到代码库中

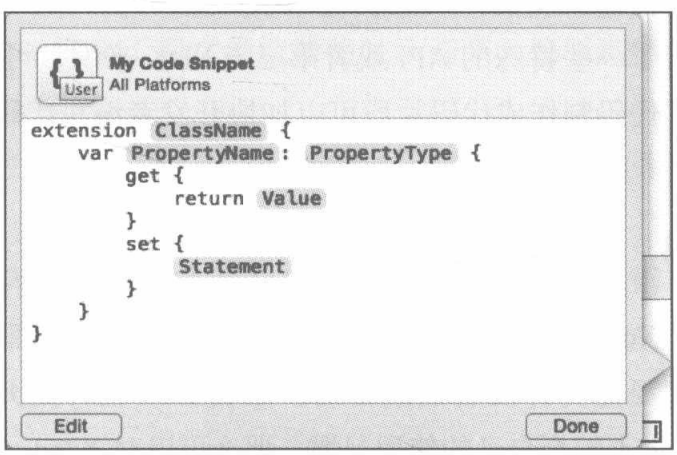


图 A-5　自定义代码片段

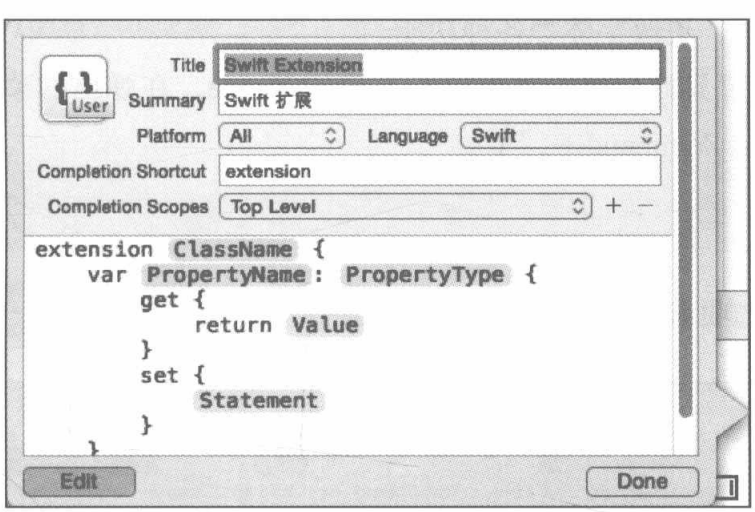


图 A-6　代码片段配置

- Title：这里你可以给代码段设置一个合适的标题。在这个例子里，我们命名它为 Swift Extension。
- Summary：这里是这个代码的简要描述信息。我们设置它的值为“Swift 扩展”。
- Platform ：选择合适的 platform，iOS 或者 MacOS。当然，如果你想保留默认值的话，也没什么关系。
- Language ：在这个菜单里你设置代码段的语言。默认情况下，显示的语言是当前代码所使用的语言。
- Completion Shortcut：这是个非常有用的功能，因为可以指定一个特定的关键值，每次在编辑器里输入这个值，它就会被代码段给替换。这里我们设置关键值为 extension。

❑ Completion Scopes：在这里实际上是定义了代码段的适用范围，决定其应该是在类里还是方法里。使用旁边的加号和减号按钮，可以添加或者删除适用范围。

点击 Done 之后，我们自定义的代码片段就创建成功了，之后我们可以在代码中输入 extension 就可以看到我们自定义的代码片段了。

借助此功能，我们就可以添加更多的方法，将它们分开成为一个个独立的代码段。尤其是对于一些特殊的 API 或者第三方 SDK 来说，可能每次都要实现一大段类似的代码，那么将这些代码制作成代码片段可以加快开发者编写代码的速度，节省大量的时间。

A.3 Xcode 设置

Xcode 和 OS X 平台上其他应用不同，它拥有非常多的设置项，多到令人发指的地步。这主要是为开发者们准备的，因为 DIY 精神是开发者的核心所在，开发者如果觉得默认的 Xcode 不符合自己的使用习惯，那么可以对 Xcode 进行配置，让它变成“个性化”的开发工具。

借助 Xcode 的配置，我们可以定制用于调用每一个 Xcode 命令和动作的快捷键，重新组织编译过程，添加工具，更改字体等各种功能。

要进入 Xcode 设置界面，单击菜单栏左上角的 Xcode，在弹出的菜单中选择 Preference，或者干脆使用快捷键“Command + ,”。

A.3.1 通用

通用（General）选项卡的界面如图 A-7 所示。

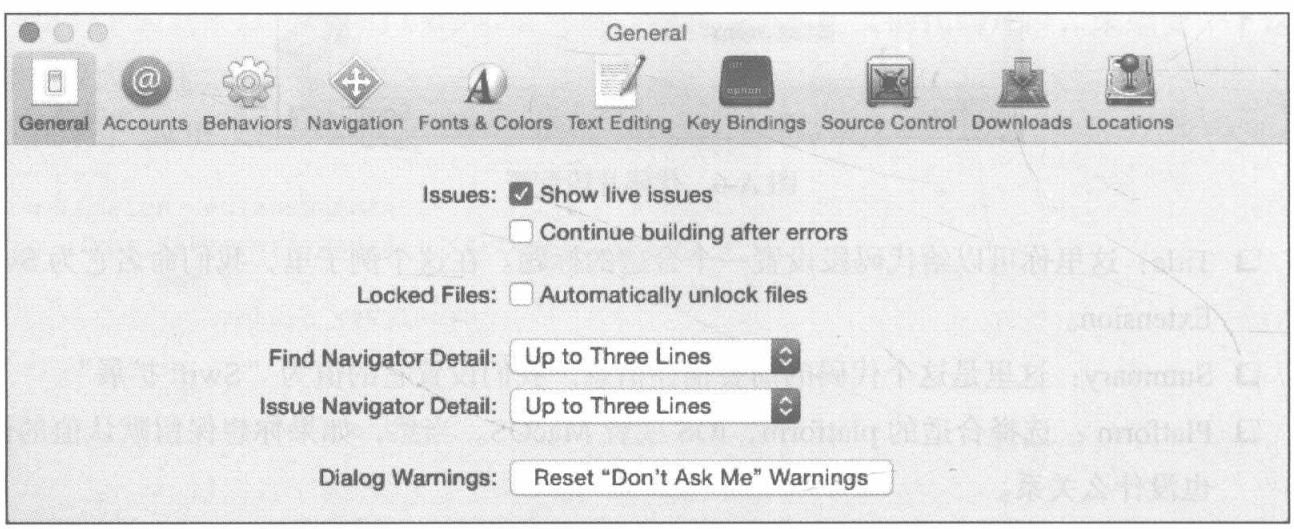


图 A-7 通用选项卡

Issues 选项主要是定义了 Xcode 对于“问题”的处理方式。Show live issues（显示当前问题）用来设置 Xcode 是否实时检测当前文件中存在的问题，如果关闭了此选项，Xcode 就不会

进行实时检测错误。Continue building after errors（出现错误后仍继续编译）选项，默认情况下，Xcode 在发现错误后，会立刻停止编译行为，选中此选项后，Xcode 会强行继续编译过程。

Automatically unlock files（自动解锁文件）：Xcode 本身有锁定文件的功能，当用户正在 Xcode 对某文件进行更改时，Xcode 就会锁定该文件，防止其被其他工具修改。选中此选项后，Xcode 将会根据需求自动解锁文件，让其他工具也能够修改正在 Xcode 当中编辑的文件。

Find Navigator Detail（搜索导航器细节）：定义了搜索导航器的显示细节，可以选择搜索到的项目可以占据多少行的显示空间。

Issue Navigator Detail（问题导航器细节）：定义了问题导航器的显示细节，可以选择当前问题可以占据多少行的显示空间。

Dialog Warnings（对话框警告）：这个选项可以重置 Xcode 中所有“不要再询问我（Don't Ask Me）”的警告选项，让其能够再次弹出。

A.3.2　账户

账户（Accounts）选项卡的界面如图 A-8 所示，这个选项卡主要是用来管理苹果开发者账号以及 Git 相关的账户信息。

图 A-8　账户选项卡

这个界面很通俗易懂，左下角分别有三个按钮，分别用来“添加账户”、“删除账户”以及“设置账户”。

“添加账户”可以添加苹果开发者账号、资源仓库以及服务器。选择“添加开发者账号”的话，输入用户名和密码后即可加入新的账户。选择“添加资源仓库”，输入资源仓库的地址之后，再输入相关配置即可加入新的资源仓库。选择“添加服务器”，选择本机 Mac 上搭建的服务器，即可完成添加。

“设置账户”的选项只允许开发者导入和导出开发者配置，当开发者需要转移自己的相关设置时，这个选项是非常有效的。

A.3.3 行为

行为（Behaviors）选项卡的界面如图 A-9 所示，这个选项卡配置了绝大多数的 Xcode 的行为配置。

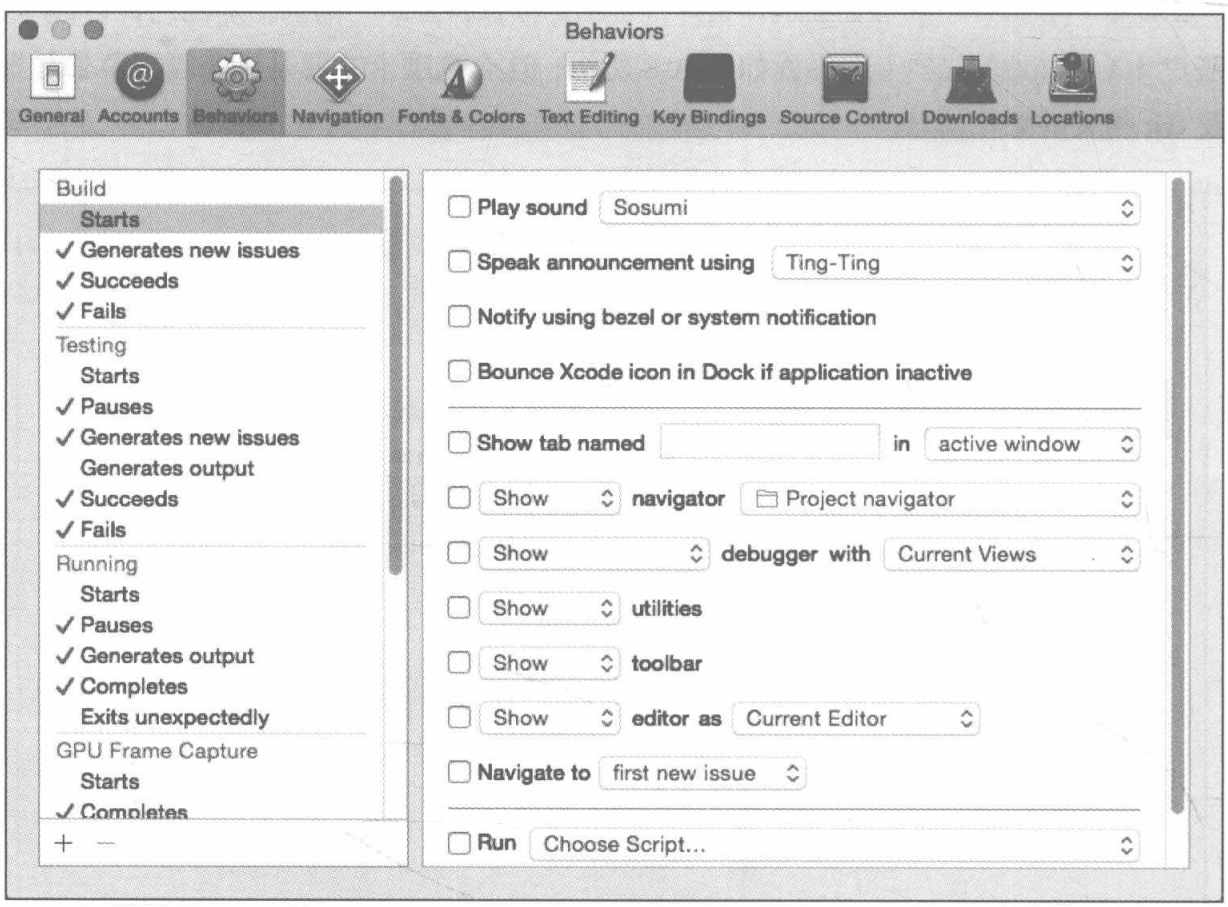


图 A-9 行为选项卡

整体上，行为选项卡分为 7 个部分，分别是编译（Build）部分、测试（Testing）部分、运行（Running）部分、GPU 帧捕获（GPU Frame Capture）部分、搜索（Search）部分、自动集成（Bots）部分以及文件（File）部分。

每个部分又依次分为了各个阶段，比如说编译部分就分为了开始（Starts）阶段、发现新问题（Generates new issues）阶段、成功（Succeeds）阶段以及失败（Fails）阶段。

对于每一个阶段，都可以依次进行相应的行为配置，如下所示：

- Play sound（播放声音）：设置这个选项可以让 Xcode 在执行到这个阶段的时候，播放一个自定义的声音，来提醒开发者程序进行到了何种阶段。声音可以选择系统自带的几种声音，也可以自己自行选择。
- Speak announcement using（使用……宣读）：设置这个选项可以让 Xcode 在执行到这个阶段的时候，使用系统自带的某个“声音”来为开发者播报当前程序出于何种阶段。
- Notify using bezel or system notifacation（使用系统通知或者提示框来通知）：勾选这个选项后，当 Xcode 执行到该选项的时候，就弹出一个通知来提醒开发者。
- Bounce Xcode icon in Dock if application inactive（若应用处于未活动状态，那么就在 Dock 中让图标跳动）：这个选项简单易懂，一般情况下主要是用来提醒开发者。
- Show...（显示）：Show 之类的 6 个选项，主要设定了当 Xcode 在执行到这个阶段的时候，要显示还是隐藏哪些导航栏、哪些窗格等。
- Navigate to（导航至）：设置这个选项可以让 Xcode 在执行到这个阶段的时候，跳转至第一条新发现的问题还是当前的日志。
- Run（运行）：设置这个选项，可以让 Xcode 在执行到这个阶段的时候，执行选项所设定的 Apple Script 脚本。
- Create snapshot（创建快照）：设置这个选项，可以让 Xcode 在执行到这个阶段的时候，创建一个快照，保存当前状态。

A.3.4　导航

导航（Navigation）选项卡的界面如图 A-10 所示，这个选项卡配置了 Xcode 导航区域的相关设置。

选项说明如下：

- Activation（活动）：这个选项的含义是：当打开某个标签页或者窗口的时候，让其保持活动状态。
- Navigation（导航）：这个选项可以选择在使用导航区域的时候，是用“Primary Editor（主编辑器）”还是“Focused Editor（助理编辑器）”来打开新的导航文件。
- Optional Navigation：这个选项可以选择当使用 Optional 键来进行导航的时候，是使用同一个助理编辑器打开，还是使用另一个助理编辑器打开，抑或是使用另一个标签页或另一个窗口打开。

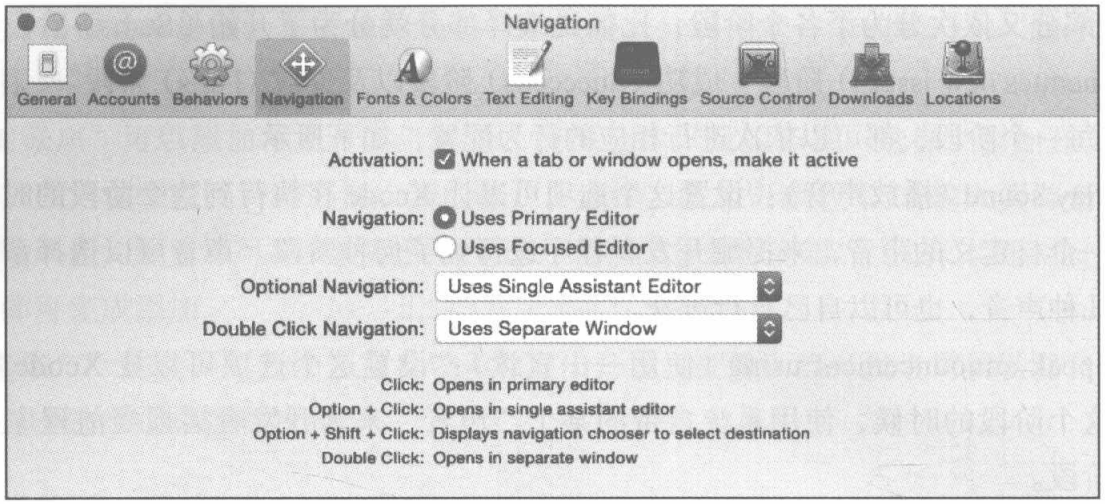


图 A-10　导航选项卡

- Double Click Navigation：这个选项可以选择当使用双击来进行导航的时候，是使用另一个标签页打开，还是使用另一个窗口打开，或者让其和单击的操作相同。

A.3.5　字体、颜色

字体 & 颜色（Fonts & Colors）选项卡的界面如图 A-11 所示，这个选项卡配置了代码编辑器的样式。

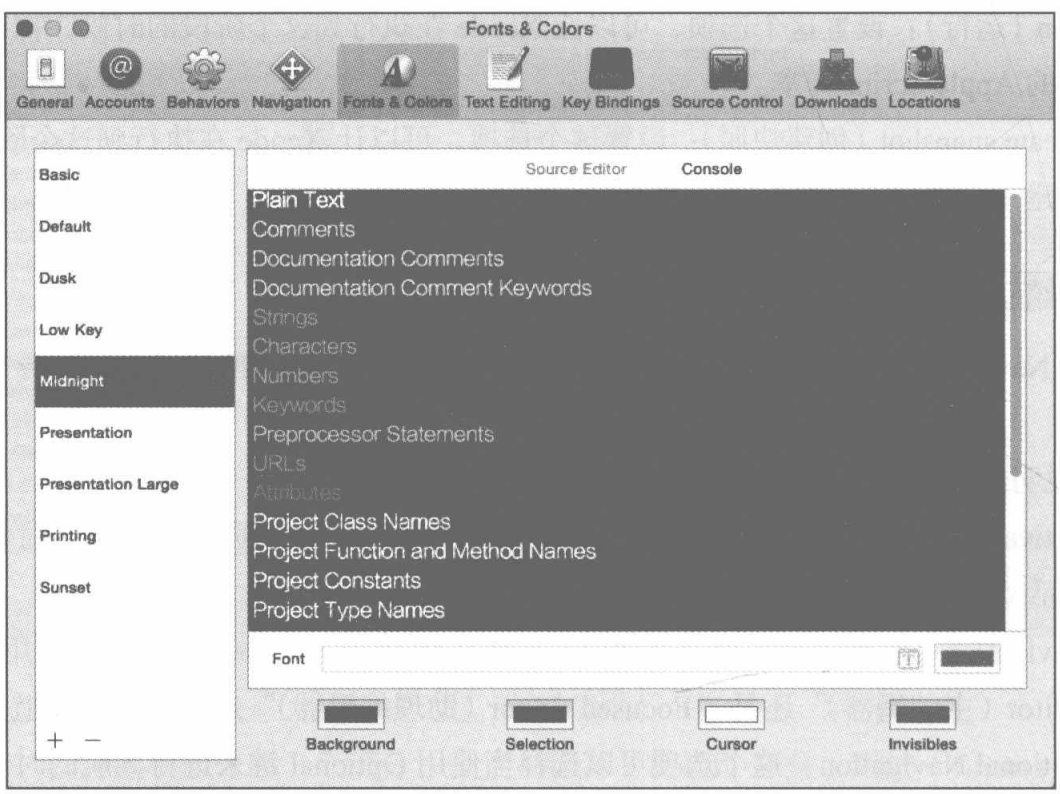


图 A-11　字体 & 颜色选项卡

Xcode 已经为我们提供了 9 种样式，如果不喜欢这些样式，还可以修改这些样式，甚至新建一个属于自己的样式。这个选项卡可以设置背景颜色、字体大小、字体颜色以及代码着色的相应颜色。

字体 & 颜色选项卡的配置影响源代码编辑器和控制台的显示样式。

A.3.6　文本编辑

文本编辑（Text Editing）选项卡的界面如图 A-12 所示，这个选项卡配置了代码编辑器的行为。

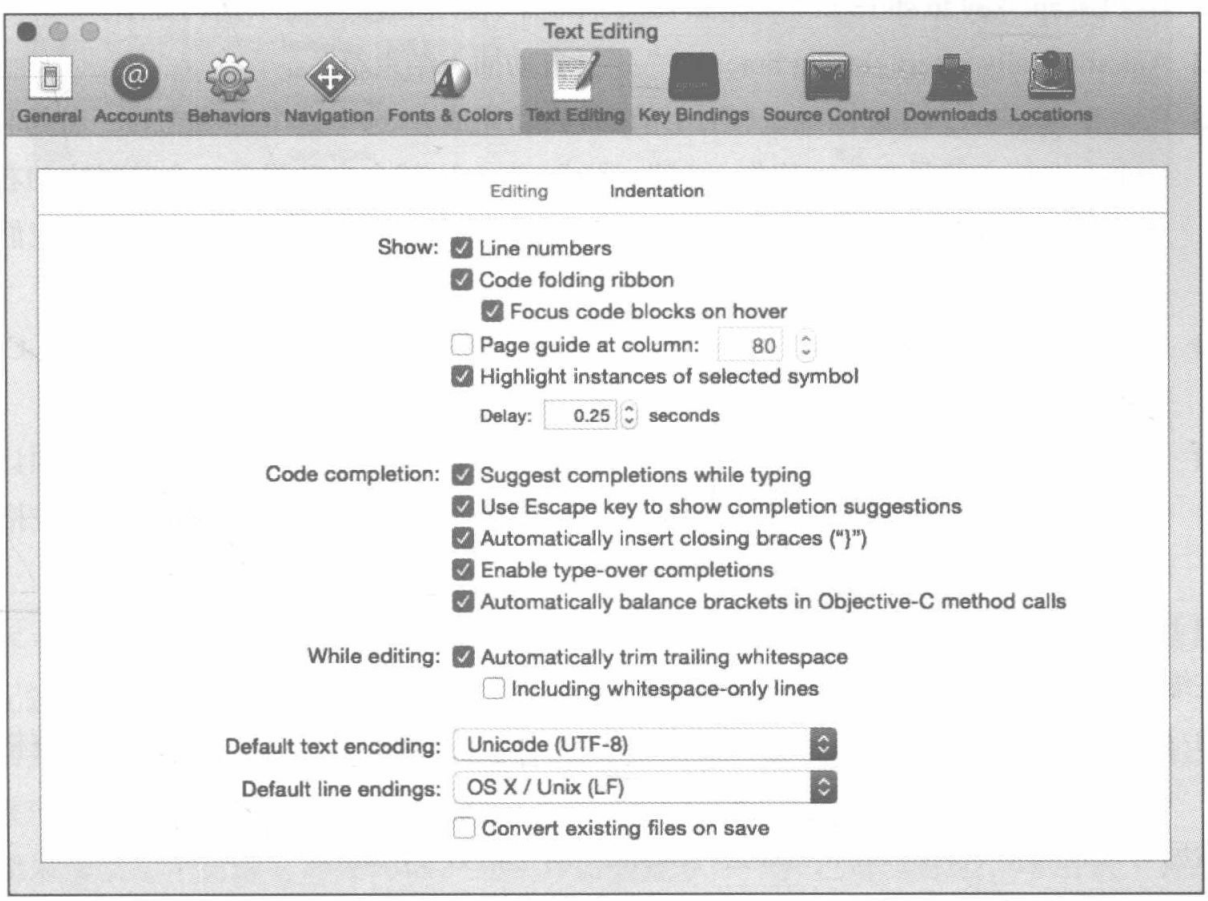


图 A-12　文本编辑选项卡

文本编辑选项卡分为编辑（Editing）和缩进（Indentation）两个模块。

在编辑模块中，我们可以设置如下选项：

- Line numbers（行号）：用来设置是否在代码的左侧显示该代码所在的行号。
- Code folding ribbon（代码折叠带）：属于同一个级别的代码段，都会在其左侧用一个灰度不同的色带来显示。灰度相同的代码段表示其属于一个类别，类别越深，则灰度的亮度越高。这个选项用来设置这个色带是否显示。
- Focus code blocks on hover（悬停时聚焦代码块）：这个功能在鼠标悬停在某一个代码

段的“折叠带”上时，以一个矩形边框来聚焦该代码所在的类别。

- Page guide at column（页面的最大行长）：设定代码编辑器中一行最多有几个字符，如果超过这个最大行长，那么编辑器将会自动换行。
- Highlight instances of selected symbol（高亮选中的符号）：高亮显示的符号是以一条虚线显示的，标记该符号是一个有特殊意义的符号。下面的 Delay（延迟），可以设定等待多少秒，这个高亮效果才会显示出来。
- Suggest Completions while typing（输入时显示代码提示）：这个选项选中后，将会在输入代码的过程中，显示提示。
- Use Escape key to show completion suggestions（使用 ESC 来显示代码提示）
- Automatically insert closing braces（“)”)（自动插入结束符）
- Enable type-over completions（启用结束符自动完成）：选中这个选项之后，当我们键入一个左向标点符号之后，比如，“{”、“(”，Xcode 就会自动显示一个其对应的右向标点符号，比如对应的“)”、“ }”，类似于自动完成。在完成这个符号之前，我们可以在这个符号之前输入任意的字符，而不用担心这个结束符被替换掉。
- Automatically balance brackets in Objective-C method calls（自动调整 Objective-C 方法调用中的括号）
- Automatically trim trailing whitespace（自动移除代码两侧的空格）：下一个选项让自动移除空格功能也包括了“空白行”，不过空白行并不会被移除，只是里面的空格会被移除。

剩下的选项是设定文本的编码方式和文件的结尾标志符。

在缩进模块中，我们可以设置如下选项：

- Prefer indent using（选用……进行排版）：这个选项设置了每次自动缩进使用空格还是制表符。

然后可以设置制表符（Tab）和缩进（Indent）的空格数（width）。

接着可以设置制表符的行为，是第一次敲击自动缩进（Indents in leading whitespace），还是一直自动缩进（Indents always），抑或是制表符缩进（Inserts tab character）。这几个行为的区别是，第一次敲击自动缩进，只有在第一次对某一行使用制表符的时候，Xcode 才会自动将该行代码缩进到合适位置，之后制表符都将行使制表符的标准行为；而一直自动缩进，则就不会有行使制表符的功能；制表符缩进则没有自动缩进的功能。

- Line wrapping（换行）：换行可以设置是否允许换行，如果允许的话，还可以设定换行之后的行的缩进字符长度。
- Syntax-aware indenting（语法换行）：根据语法来自动决定换行的行为。
- Automatic indent for（自动缩进）：设定哪些符号可以执行自动缩进功能。

A.3.7　快捷键

快捷键（Key Bindings）选项卡的界面如图 A-13 所示，这个选项卡配置了 Xcode 的快捷键。

图 A-13　快捷键选项卡

本书附录 A.1 节介绍了不少的快捷键，但是那只是 Xcode 默认的快捷键方式，如果你对默认配置不满意，就可以来这个选项卡中自定义某个操作的快捷键。

需要提醒的是，在修改快捷键的时候，请新建一个“键绑定集”（Key Bindings Set），以防如果做出了什么不满意或者错误的修改，还有后悔的机会。

如果修改的快捷键有冲突的话，选项界面会显示一个黄色警示标志，单击这个警示标志可以快速定位到有冲突的快捷键上。

A.3.8　源代码管理

源代码管理（Source Control）选项卡的界面如图 A-14 所示，这个选项卡配置了代码源的相关设置，说明如下：

- Enable Source Control（启用源代码管理）：这个选项决定了“源代码管理”这个功能是否可用。
- Refresh local status automatically（自动刷新本地状态）：这个选项决定了本地的代码源是否会自动刷新，也就是自动更新。

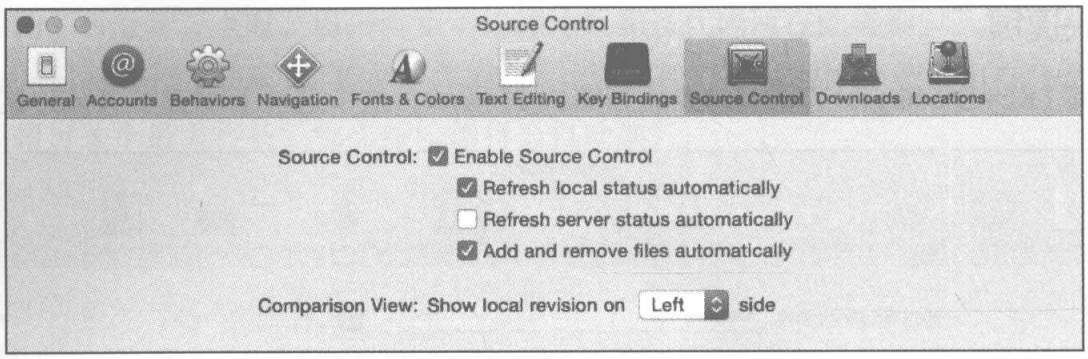


图 A-14　源代码管理选项卡

- Refresh server status automatically（自动刷新远程服务器状态）：这个选项决定了远程代码源中的代码源是否会自动保存，自动更新。
- Add and remove files automatically（自动添加和移除文件）：当本地或者远程代码库更新后，Xcode 会自动添加和移除本地项目所没有的文件。
- Show local revision on ... side（在……侧显示本地版本）：在执行代码源提交更改的时候，这个选项决定了本地的版本在哪一侧出现，一般出现在左侧。

A.3.9　下载

下载（Downloads）选项卡的界面如图 A-15 所示，这个选项卡配置了 Xcode 参考文档等附加项的下载管理。

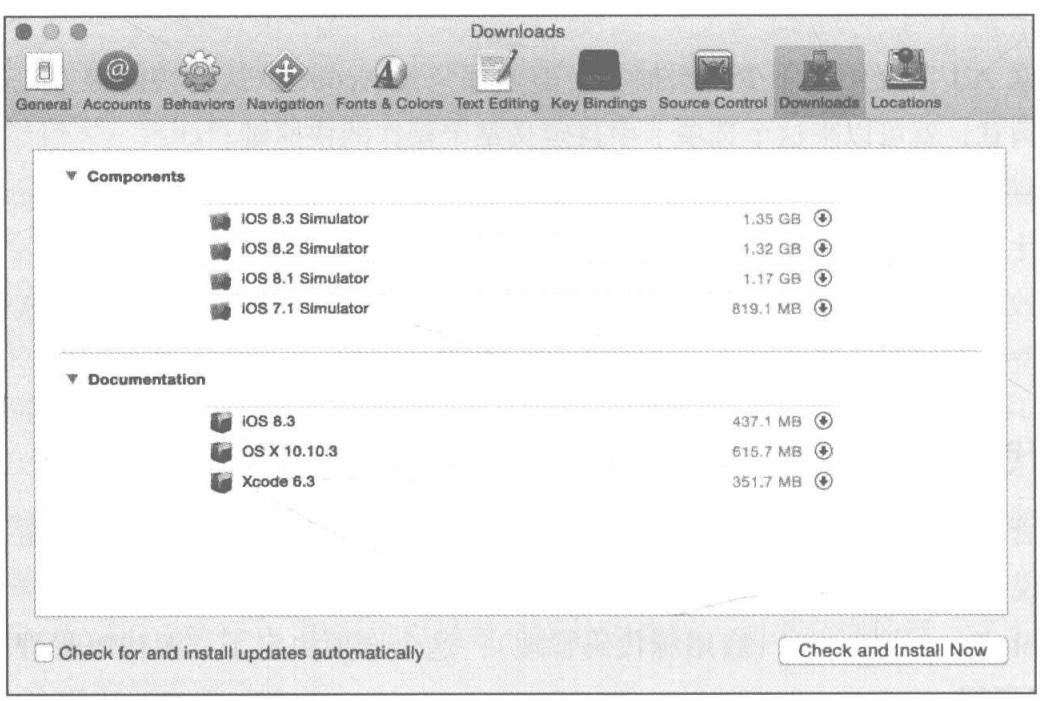


图 A-15　下载选项卡

在这个选项卡中，可以设置下载 SDK 组件（Components）以及参考文档（Documentation），

从中可以选择下载何种版本的附加项，还可以选择下载 iOS 7.1 ～ iOS 8.3 版本的模拟器，下载 iOS 8.3 以及 OS X 10.10.3 的离线参考文档。当然，对于 Xcode 6.4 来说，它内部已经包含了 iOS 8.4、OS X 10.10.4、Xcode 6.4 的模拟器和参考文档了。

A.3.10　位置

位置（Locations）选项卡的界面如图 A-16 所示，这个选项卡配置了一些常用操作的路径，选项说明如下：

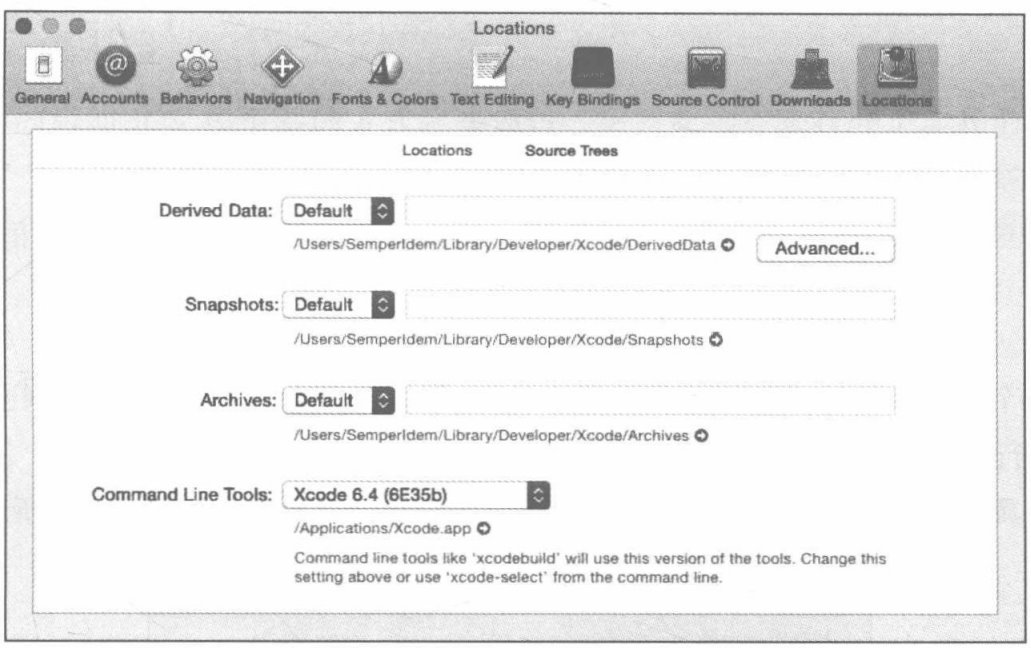


图 A-16　位置选项卡

- Derived Data（缓存文件）：这个地方存储了 Xcode 在运行过程中产生的缓存文件。
- Snapshots（快照）：这个地方存放了 Xcode“应用快照”。
- Archives（打包文件）：这个地方存放了成功打包后的文件。
- Command Line Tools（命令行工具）：如果我们有多个 Xcode 版本，那么就可以在这里选择 Xcode 用哪一个版本的命令行工具来执行程序。

对于 Source Trees（源代码树）来说，在这里可以定义一些路径的别名。比如，我们想使用 /Documents/Games/cocos-2d-x 目录下的资源，那么可以如图 A-17 所示来设置。

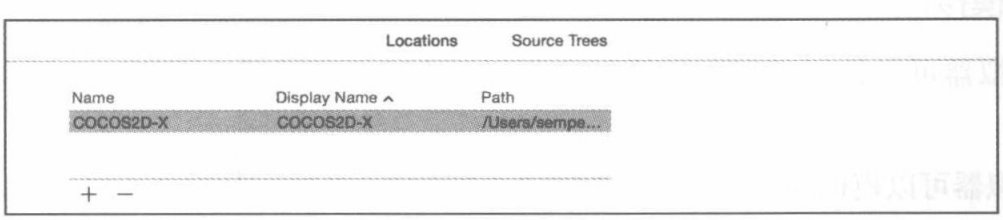


图 A-17　源代码树

然后，我们就可以在项目设置的编译设置中，用 $(COCOS2DX-ROOT) 来使用这个目录了。

Appendix B

附录 B

不二法门——Xcode 工具箱

除了上个附录介绍的一些强大的工具之外，Xcode 还提供了一系列非常好用的工具，帮助开发者进行开发。

这些工具就存放在菜单栏的 Xcode → Open Developer Tool 当中，如图 B-1 所示。

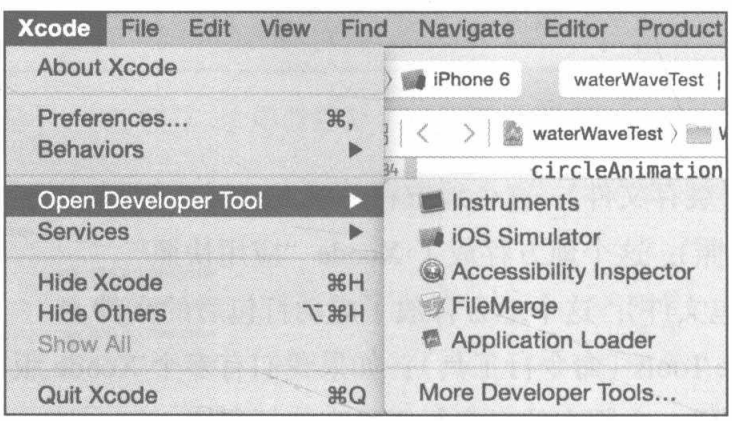


图 B-1　Xcode 工具箱

B.1　iOS 模拟器

iOS 模拟器可以让开发者在应用开发阶段，快速地实现 iOS 应用程序原型并对其进行测试。

iOS 模拟器可以模拟 iPhone、iPad 以及 Apple Watch 环境，这对苦于没有测试设备的开发者来说是一个极好的消息。此外，先在模拟器上成功运行程序，再去真机上运行，已经成为了业内的通用做法了。因此，掌握 iOS 模拟器的使用方式，能够更为有效地帮助开发。

一般而言，一个完整的 iOS 模拟器由其模拟的 iOS 设备加上版本号组成。比如说“iPhone 5s (8.4)”，代表了一个运行 iOS 8.4 系统的 iPhone 5s 设备。

除了模拟几种不同的 iOS 设备之外，iOS 模拟器还可以模拟不同的 iOS 系统版本，只要我们装了不同版本的 iOS 模拟器，那么这项功能是十分容易实现的。多个不同的 iOS 设备和多个不同的 iOS 版本可以在 Mac 上很好地共存，无需担忧其会出现冲突。它们都拥有自己的设置和文件系统，因此，在一个模拟器上进行测试的程序是不会安装到其他模拟器上的，并且对一个模拟器进行相关配置也不会影响到其他模拟器。

B.1.1　运行模拟器

最简单的启动 iOS 模拟器的方式就是通过“运行应用”来打开了。我们在工具栏的“运行目标”处，选择我们想要运行应用的模拟器设备，比如说 iPhone 6，然后通过点击“运行”按钮，Xcode 即可打开 iOS 模拟器，并且同时运行我们刚刚编译成功的应用，如图 B-2 所示。

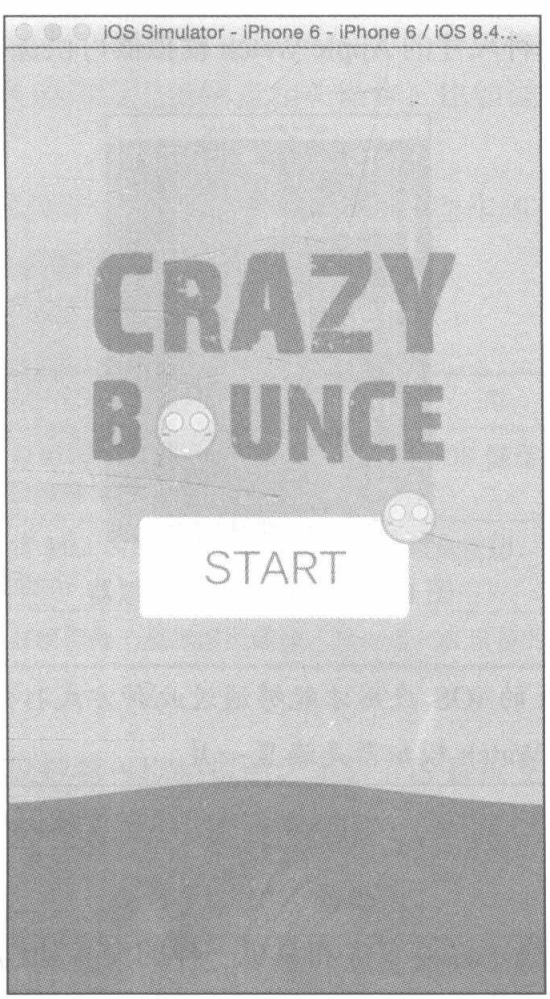


图 B-2　iPhone 模拟器

如果你项目的部署对象设置为了 iPad，那么只能够使用 iPad 模拟器来运行这个项目；而如果你选择的是 iPhone 或者 Universal，那么 iPhone 和 iPad 模拟器都可以运行该项目。

还有一种启动方式是直接打开，这就要通过 Xcode 工具箱的打开方式来打开了。选择菜单栏上的 Xcode → Open Developer Tool，在弹出的菜单中选择 iOS Simulator。

通过这种方式来打开 iOS 模拟器的话，就不会运行任何应用。iOS 模拟器显示的就是 iOS 系统的桌面，我们可以通过和 iOS 设备上进行同样的操作来使用模拟器。

对于 Xcode 6.2 之后的版本来说，iOS 模拟器还提供了 Apple Watch 的模拟。iOS 模拟器会在 Apple Watch 模拟设备中运行 WatchKit 的应用，并且其是和 iPhone 模拟器一同出现的。

请确保 iPhone 模拟器的版本号在 8.2 之上，否则不能够打开 Apple Watch 模拟器。

打开模拟器后，选择菜单栏上的 Hardware → External Displays，然后在弹出的菜单中选择 Apple Watch。目前，有两种尺寸的 Apple Watch 模拟器可供选择，如图 B-3 所示。

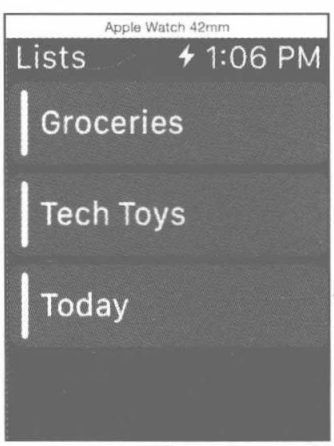


图 B-3　Apple Watch 模拟器

只有包含了 WatchKit 的 iOS 应用才能够通过此种方式打开 Apple Watch 模拟器，否则的话，打开的 Apple Watch 模拟器是漆黑一片。

B.1.2　与模拟器交互

模拟器最大的好处是提供了许许多多和真机一样的交互动作，这让模拟器的用处变得很大。但是，仍然有很多交互动作是模拟器不能完成的，这也是模拟器不能够代替真机进行完全测试的原因之一。

B.1.2.1　硬件交互动作

所谓“硬件交互动作”，指的是用户与物理实体进行相关交互的操作，比如说横屏、摇一摇等等动作。

表 B-1 列出了所有 iOS 模拟器支持的硬件交互动作，这些动作和功能都能够在 iOS 模拟器菜单栏的 Hardware 栏目中找到。

表 B-1　硬件交互动作

模拟交互动作	作　　用	对应的真实交互动作
左旋	将模拟器向左旋转 90°	用户将设备向左旋转，Home 键在右边
右旋	将模拟器向右旋转 90°	用户将设备向右旋转，Home 键在左边
摇动手势	模拟设备摇动	类似微信“摇一摇”
Home	模拟摁下 Home 键的效果	用户摁下设备上的 Home 键
锁定	在模拟器上显示锁屏界面	用户关闭屏幕又打开的情况
触发“通话中”状态栏	模拟“通话中”的状态	用户在后台进行通话操作

B.1.2.2　键盘动作

所谓“键盘动作”，指的是模拟器上的键盘相关操作，比如说打开、关闭键盘，切换键盘等等操作。

表 B-2 列出了所有 iOS 模拟器支持的键盘动作，这些动作和功能都能够在 iOS 模拟器菜单栏的 Hardware → Keyboards 栏目下找到。

表 B-2　键盘动作

键 盘 动 作	作　　用
iOS 使用和 OS X 相同的布局	自动选择最切合 Mac 上键盘布局的 iOS 键盘，改变 Mac 上的键盘布局也会同时改变模拟器上的键盘布局
连接物理键盘	将 Mac 的键盘作为模拟器的输入键盘使用，这样的话模拟器将不会显示虚拟键盘
触发软键盘	在模拟器上显示虚拟键盘，这也是 iOS 设备的普遍输入方式

B.1.2.3　用户手势动作

通过鼠标和触控板，iOS 模拟器可以模拟绝大多数 iOS 系统支持的手势操作。

表 B-3 列出了所有 iOS 模拟器支持的用户手势动作。

表 B-3　用户手势动作

用户手势动作	键盘或触控板所做的动作
单击（Tap）	鼠标左键单击或触控板单击
长按（Touch & Hold）	鼠标左键长按或触控板左键长按

（续）

用户手势动作	键盘或触控板所做的动作
双击（Double-Tap）	鼠标左键双击或触控板双击
拖拽（Drag）	鼠标左键摁住不放然后拖曳或触控板拖曳
轻扫（Swipe）	鼠标左键摁住不放然后拖曳或触控板拖曳
滑动（Flick）	快速拖曳
双指拖拽（Two-finger Drag）	按下 Option 键，在出现两个圆圈后再按下 Shift 键执行拖曳
缩放（Pinch）	按下 Option 键，在出现两个圆圈后执行缩放操作
旋转（Rotate）	按下 Option 键，在出现两个圆圈后执行旋转操作

B.1.2.4 Watch 手势动作

通过鼠标和触控板，iOS 模拟器可以模拟绝大多数 Apple Watch 支持的手势操作。

表 B-4 列出了所有 Apple Watch 模拟器支持的用户手势动作。

表 B-4 Watch 手势动作

用户手势动作	键盘或触控板所做的动作
单击（Tap）	鼠标左键单击或触控板单击
双击（Double-Tap）	鼠标左键双击或触控板双击
Force Touch	鼠标左键长按或触控板左键长按
顺时针转动王冠（crown）	向上拖动手表窗口
逆时针转动王冠	向下拖动手表窗口
快速转动王冠	快速拖动

B.1.3 测试与调试

作为一个在前期调试和测试阶段替代 iOS 真实设备的模拟工具，iOS 模拟器也提供了一系列特有的测试和调试工具和功能，帮助开发者更好地进行调试、测试。

B.1.3.1 iCloud 测试

iOS 模拟器支持一部分的 iCloud 功能，它可以进行 iCloud 数据同步，测试多设备之间的信息同步等等。

为了启动这项测试功能，必须先使用 Apple ID 登录到 iOS 模拟器上，否则的话 iCloud 测试功能将不能正常使用。

只有在 iOS 7.0 以上的模拟器版本才能够正常使用 iCloud。

iOS 模拟器仅支持一些简单的数值同步，开发者可以通过 Debug → Trigger iCloud Sync（触

发 iCloud 同步）功能来强制让当前支持 iCloud 的应用进行一次同步，来模仿苹果服务器发来的同步请求。

B.1.3.2　调试工具

在 Debug 菜单中，iOS 模拟器还提供了一系列功能的调试工具，如表 B-5 所示。

表 B-5　调试工具项

菜　单　项	作　　用
慢速动画（Slow Animations）	将应用中发生的所有动画帧率降低，以此来确定动画显示是否正确。此外，按下三次 Shift 键也可以开启这个效果
混合图层着色（Color Blended Layers）	显示混合视图图层，绿色代表不透明部分，红色代表透明部分，红色越多，性能越差
复制图层着色（Color Copied Layers）	给被 Core Animation 复制出来的图像覆盖上一层蓝色
未对齐图像着色（Color Misaligned Imageds）	给没有和预期尺寸对齐的图像覆盖上一层品红色
屏幕外渲染层着色（Color Off Screen Rendered）	给在屏幕外的渲染层覆盖上一层黄色
位置（Location）	可以一系列位置和运动模式

B.1.3.3　调启日志

如果在调试应用期间发生了崩溃，那么崩溃日志报告能够更好地帮助开发者发现并解决问题。

打开系统中的“控制台（Console）”应用，然后在里面即可查看相应的信息。通过 Debug → Open System Log 也可以达成目的。

B.1.4　iOS 模拟器的缺陷

iOS 模拟器是一个非常有用的工具，但是我们并不推荐大家使用它来代替真机进行测试，原因如下：

1. 硬件方面

虽然绝大多数硬件功能都能够在 iOS 模拟器上得以很好的仿真，但是有一些硬件功能就只能够在真机上进行测试。比如说：

动作支持：加速器、陀螺仪。

音像输入：相机、话筒。

近物体传感器：气压计、光线传感器。

此外，在 iOS 模拟器上测试 Apple Watch 的效果远比真机上好很多，因为它们都由一个应用运行，不存在信号干扰、丢包等传输错误现象。

2. OpenGL ES 方面

- iOS 模拟器不使用基于延迟的渲染技术（Tile-based deferred renderer）。

- iOS 模拟器不提供和真实图形硬件所拥有的像素分辨率。
- iOS 模拟器上的渲染性能和真实机器上的不等同。

3. API 方面

- iOS 模拟器不能够接收和发送苹果推送通知。
- iOS 模拟器不能够在访问相机、日历、联系人和提醒事项等应用的时候，提供隐私权限提醒。
- iOS 模拟器不能使用 UIBackgroundModes 键。
- iOS 模拟器不支持 iCloud 文档同步和键 – 值存储。
- iOS 模拟器不支持 Handoff。
- 此外，iOS 模拟器还不支持 External Accessory、Media Player、Message UI、EventKit 等框架。

4. 后台服务方面

iOS 模拟器不支持后台服务。

B.2 辅助功能测试器

辅助功能测试器（Accessibility Inspector）是 Xcode 中自带的一个检测工具，它可以非常方便快速地获取 iOS 应用中的各个控件元素的层级结构，并且可以模拟 VoiceOver 与这些元素进行交互。这个辅助功能测试器一般是和 iOS 模拟器结合使用的，如图 B-4 所示。

这个工具需要在 Mac 的安全性与隐私里面设置权限，允许其使用辅助功能，即允许它控制电脑。

打开这个工具，可以看到它一直悬浮在屏幕的最前端，并且能够检测到鼠标所指界面的层级结构，包括 Mac 中的各种应用，等等。

一般这个工具是结合 UIAutomation 使用的，用其可以获取应用中的控件相关信息，然后就可以对其实现测试自动化了。

如图 B-4 所示，这里我将指针放在了 Xcode 的“ Run”按钮上方，这时候我们就可以分析这个“按钮”的相关层级和属性了。

我们可以看出，在层级（Hierarchy）一栏中，我们可以看出该按钮位于一个 Toolbar（工具栏）上方，而 Toolbar 又位于 Window 窗口上，窗口依附于一个 Application。由此我们也就可以清晰直观地了解到一个基本 OS X 应用的层级构成了。

在属性（Attributes）一栏中，我们可以看到这个按钮的相关属性。比如说我们观察到 accessibilityLabel 项目，就可以知道这个按钮的名称为 Run。

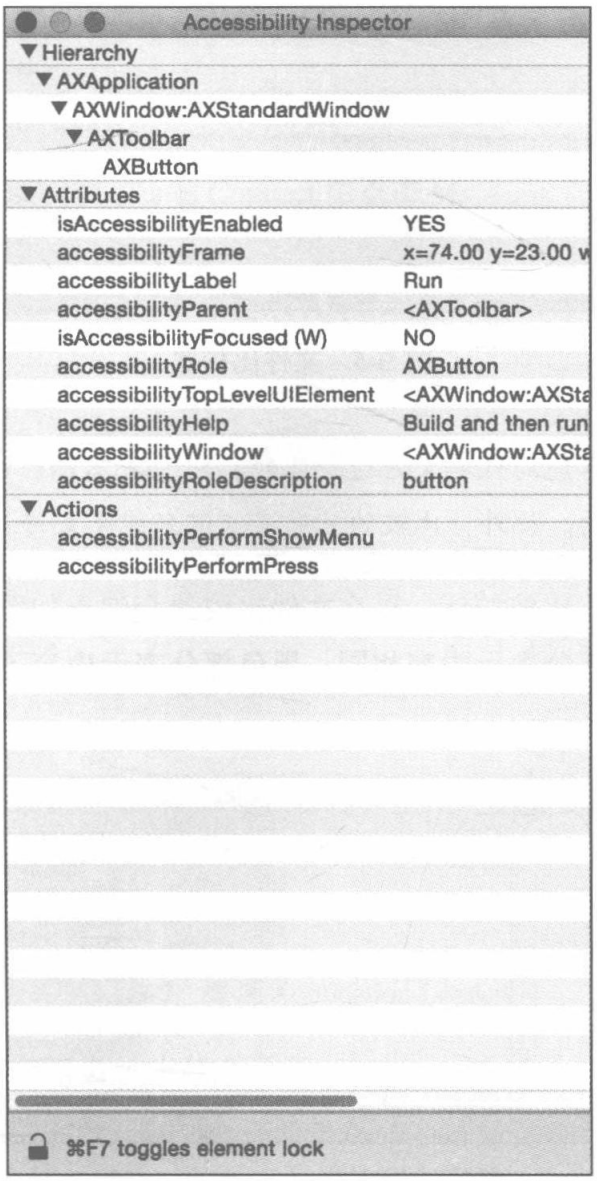


图 B-4　辅助功能测试器

此外，对于按钮来说，我们还可以查看它的动作（Actions），由此我们可以看到它支持两种操作：点击（Press）和显示菜单（ShowMenu）。

B.3　文件比较器

（FileMerge）文件比较器是一个管理代码合并的工具，在我们使用 Git 合并的时候，就用到了这个工具的部分功能，只不过 Xcode 深度集成了这个工具的部分功能，让我们感受不出来而已。这个工具的界面如图 B-5 所示。

它的界面十分简单，接下来就是设置其 Left 和 Right，作为比较的两个文件。选择完毕后，单击 Compare 进行比较操作。

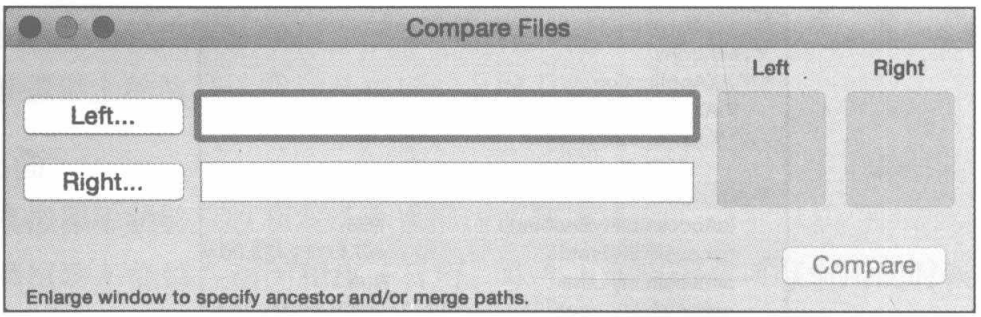


图 B-5　文件比较器

文件比较器不仅仅只能对代码文件进行比较，对于文本编辑的类似文档都可以使用文件比较器来进行比较的。此外，比较的对象不仅仅只能是文件，文件夹也是可以比较的。

如果选择对文件夹进行比较的话，那么文件比较器会弹出如图 B-6 所示的界面。左边部分是一个目录结构，灰色部分表示内容相同，黑色部分表示内容不同。通过选择内容不同的部分，就可以开始进行比较了。

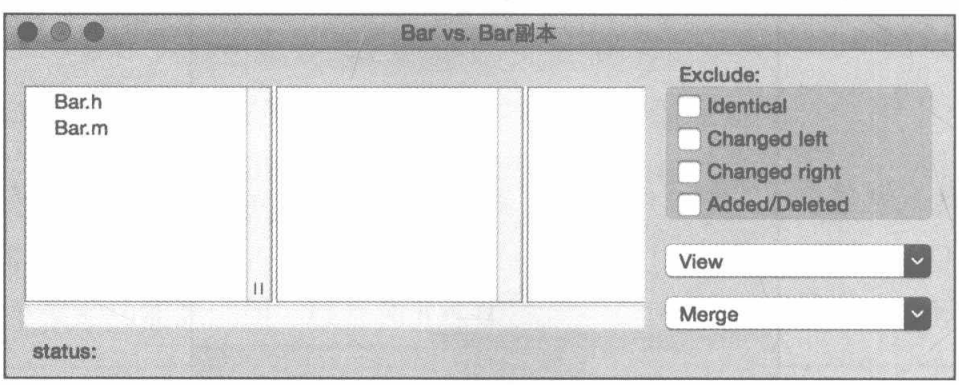


图 B-6　比较目录

Exclude 是过滤展示选项，Identical 是隐藏相同的文件和文件夹，Changed left 是只显示左侧文件夹相对右侧文件夹发生改变的文件，Changed right 与 Changed left 相反，而 Added/Deleted 则显示只有增加或删除的文件。

下面的选项用来控制打开行为。Comparison 则展示当前选中文件的比较结果，等同于直接双击打开要比较的文件。Left file 和 Right file 则是用编辑器分别打开左右两边的文件，Ancestor file 则是打开历史文件，Merge file 则是打开合并后的文件。

然后最下面的选项则是控制文件最终的行为。

B.4　应用加载器

Application Loader（应用加载器）可以让你打包好的 App 快速上传至 App Store，并可以

提供早期的验证警告。

B.4.1　使用 Application Loader 上传应用

1）安装 Applicatioin Loader 在 iTunes Connect 中点击 My Apps，点击“ Download Application Loader”链接下载安装包。

2）使用与 iTunes Connect 相同的账号登录 Application Loader，如图 B-7 所示。

图 B-7　登录 Application Loader

3）双击 Deliver Your App，在弹出的对话框中选择 ipa 文件。

4）点击 Next，即可开始上传，如图 B-8 所示。

在上传的过程中可以点击 Activity 来查看上传的详细信息。

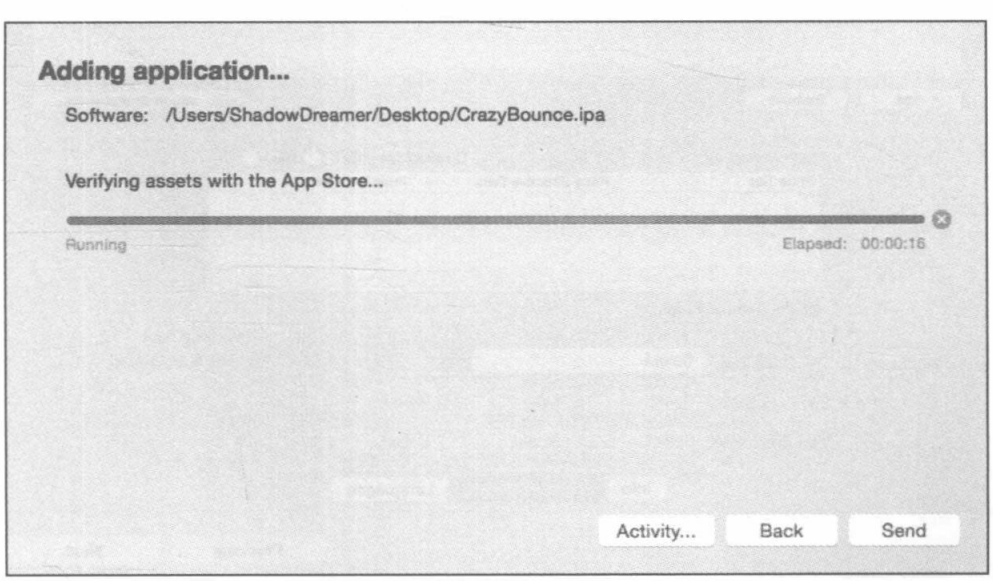


图 B-8　上传应用

B.4.2 使用 Application Loader 管理程序内购

1）登录后在主界面点击 New In-App purchases，弹出 Manage In-App Purchases（管理程序内购）页面，如图 B-9 所示。

2）双击你想管理的 App，即可显示已有的内购选项。

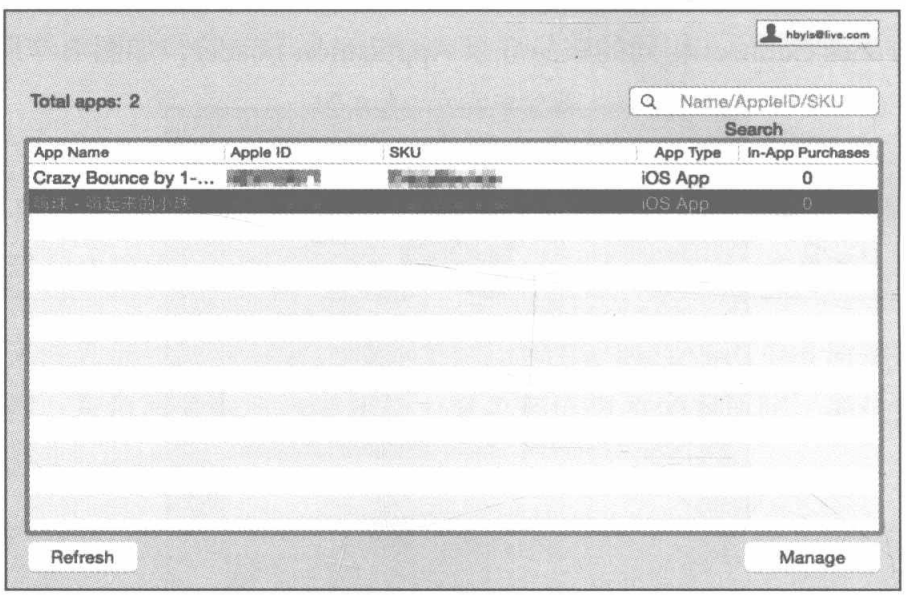


图 B-9 管理程序内购

3）点击 Add 添加一个内购，或者双击已有的内购列表编辑，如图 B-10 所示。

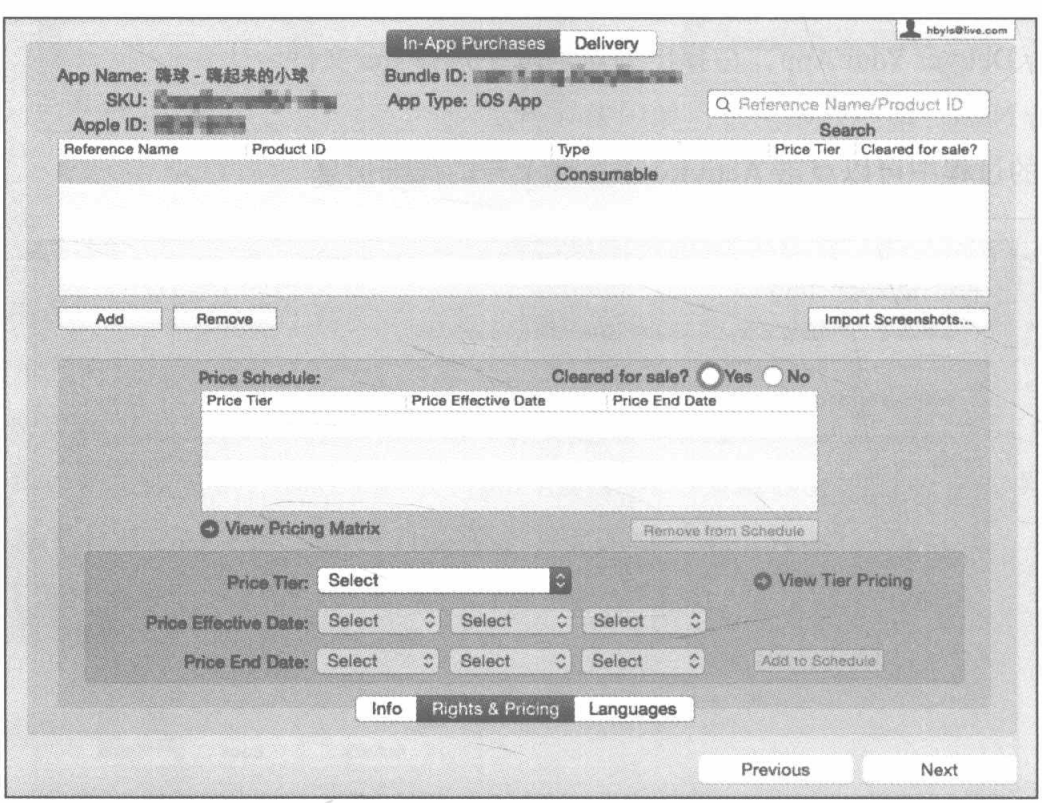


图 B-10 添加新的内购项目

4）最后点击 Next 保存。

B.5　还有更多工具

Xcode 工具箱中不仅只包含了上面这些好用的工具，如果有必要的话，你还可以前往 http://developer.apple.com/downloads 下载，登录 AppleID 之后，就可以查看可用下载列表，从中就可以下载命令行工具（Command Line Tools）、图形工具（Graphics Tools）、硬件接口工具（Hardware IO Tools），等等。如果你熟悉这些工具，那么就可以前去把这些工具带回到 Xcode 工具箱当中。

Appendix C

附录 C

武术套路——模板

C.1 文件模板

通过标题栏 File → New → File 菜单可以打开文件模板窗口，从这个窗口中可以创建各式各样的适用于 Xcode 的文件，可参考第 3 章的图 3-2。

C.1.1 iOS 文件

iOS 文件有几件：代码文件、用户界面文件、Core Data 文件、资源文件等，下面分别介绍。

C.1.1.1 代码（Source）文件

代码文件是 iOS 文件模板当中用来存储源代码的，绝大部分对数据、模型、视图的处理和加工都在这里发生。这也是开发者的舞台，开发者的绝大部分时间都花费在如何处理源代码，以实现各种各样的功能。

Cocoa Touch Class

Cocoa Touch 是 iOS 平台应用的核心框架，它提供了一系列的 API，为触摸这一操作方式提供了极大的优化效果。而 Cocoa Touch Class 则是一个用来创建一个基于 Cocoa Touch 框架的类模型文件。选择此文件后，Xcode 会弹出如图 C-1 所示的文件设置属性：

Class 一项用来定义这个类的名称，在 Subclass of 可以选择这个类的父类，这些父类都是基于 Cocoa Touch 框架所提供的。如果我们选择了 Controller 类别的父类，那么就还可以在下方的 Also create XIB File 选择是否同时创建一个关联的 XIB 文件。如果选择同时创建关联 XIB 文件，那么在下方还可以选择这个 XIB 文件所适配的设备类型，是 iPhone 还是 iPad 。

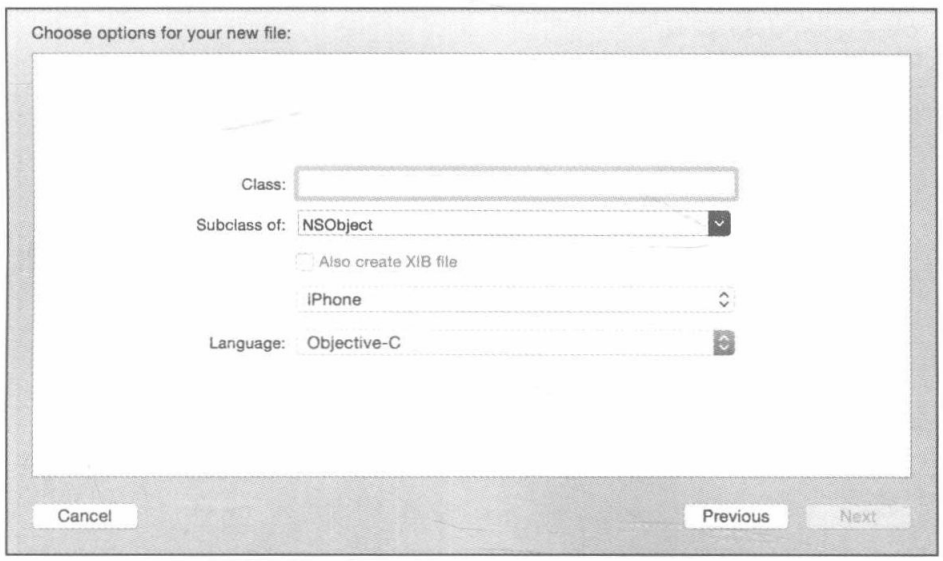


图 C-1　Cocoa Touch Class

然后在 Language 一项中，可以选择创建类所使用的语言，目前有两个语言可供选择：Swift 以及 Objective-C。

单击“Next”，选择存储的位置之后，即可完成创建。

一个以 UIViewController 为父类的 Swift 类模板代码如下：

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func didReceiveMemoryWarning() {
            super.didReceiveMemoryWarning()
    }
}
```

使用这个功能创建类的好处是，Xcode 已经事先完成了一些必须的方法，并且还提供了不少的注释来帮助开发者，极大地减轻了开发者的工作量。

Test Case Class

Test Case Class 是基于单元测试对象的一个子类，使用这个模板可以很方便的创建一个测试类，来进行相关的测试。选择此文件后，Xcode 会弹出如图 C-2 所示的文件设置属性。

Class 项当中输入该类文件的名称；Subclass of 项中选择 XCTest 框架中的一个类作为父类；Language 项选择要创建该类的语言，目前可以选择 Objective-C 和 Swift。

单击“Next”，选择存储的位置之后，即可完成创建。

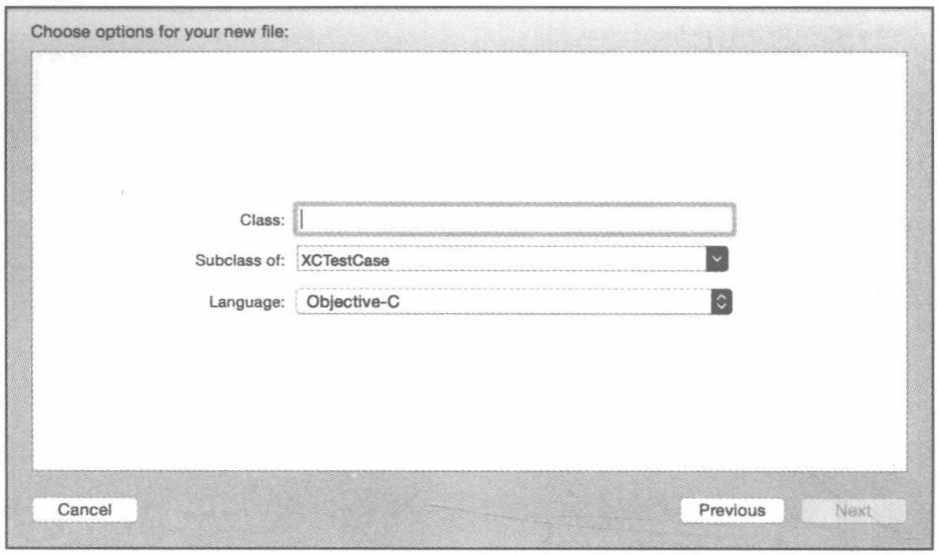


图 C-2 Test Case Class

Playground

该模板将创建一个 Playground 文件，选择此文件后，Xcode 会弹出如图 C-3 所示的文件创建属性。

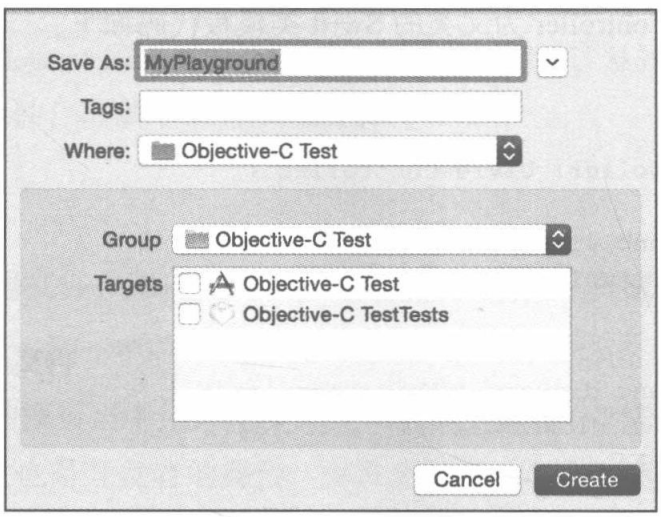


图 C-3 Playground

输入名称和存储位置，以及所属分组和对象之后，即可完成创建。

在项目中创建的 Playground 文件内含有两个文件夹：Sources 和 Resources 文件夹，分别用来存放源代码文件和资源文件。

Swift File

该模板将创建一个 Swift 文件，选择此文件后，Xcode 会弹出文件创建属性。

输入名称和存储位置，以及所属分组和对象之后，即可完成创建。

Objective-C File

该模板创建了一个 Objective-C 文件，选择此文件后，Xcode 会弹出如图 C-4 所示的

文件设置属性。

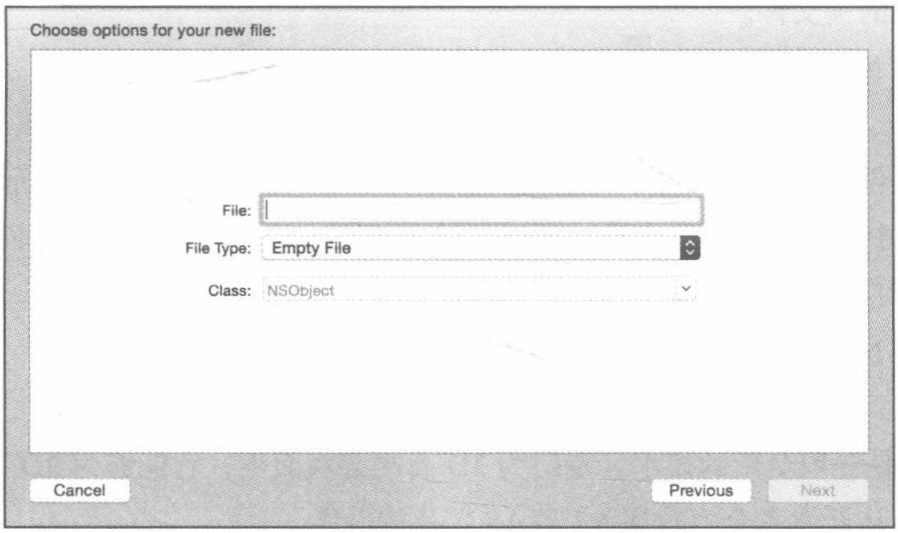


图 C-4　Objective-C 文件

File 项当中输入该文件的名称；File Type 项当中选择要创建的 Objective-C 文件类型，可以选择 Empty File（空文件）、Category（类别）、Protocol（协议）以及 Extension（扩展）。其中，类别和扩展还可以选择它们要继承的父类。

单击“Next”，选择存储的位置之后，即可完成创建。

Header File

该模板将创建一个头件，选择此文件后，Xcode 会弹出文件创建属性。

输入名称和存储位置，以及所属分组和对象之后，即可完成创建。

C File

该模板创建了一个 C 文件，选择此文件后，Xcode 会弹出如图 C-5 所示的文件设置属性：

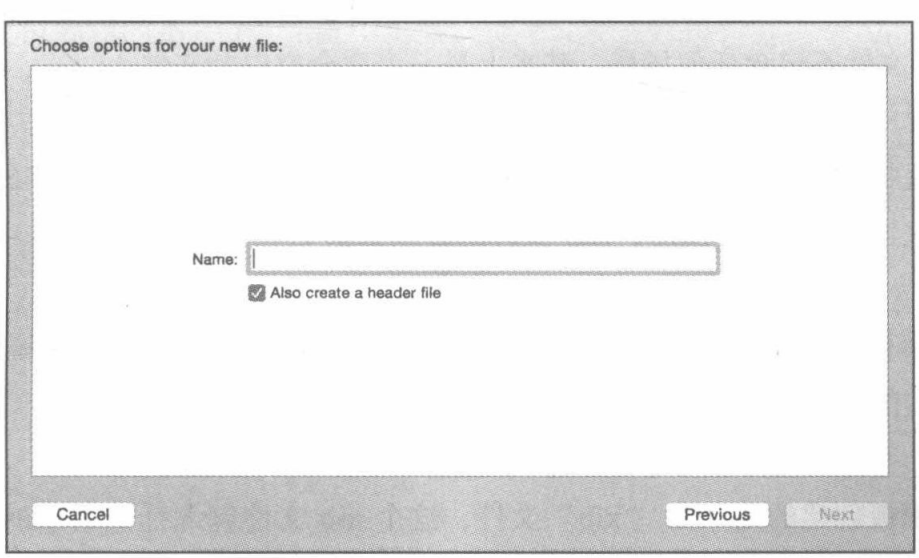


图 C-5　C 文件

Name 中输入该文件的名称，同时，该模板还允许开发者选择是否同时创建一个与该 C 文件对应的头文件。

单击“Next”，选择存储的位置之后，即可完成创建。

C++ File

该模板创建了一个 C 文件，选择此文件后，Xcode 会弹出如图 C-5 所示的文件设置属性。

Name 中输入该文件的名称，同时，该模板还允许开发者选择是否同时创建一个与该 C++ 文件对应的头文件。

单击“Next”，选择存储的位置之后，即可完成创建。

Metal File

Metal 是一项全新的技术，专门为 3D 高端游戏而生，它能够为 3D 图像提高 10 倍的渲染性能，让开发者全力发挥 A7 和 A8 芯片的性能。

选择此文件后，Xcode 会弹出文件创建属性。输入名称和存储位置，以及所属分组和对象之后，即可完成创建。

C.1.1.2 用户界面（User Interface）文件

用户界面文件是用来实现“所见即所得”效果的文件，是一种可视化的设计舞台，能够用非常简单的方式和技术将用户能够看见的界面展示出来，并且能够和代码良好的结合。随着 Xcode 的更新，用户界面文件也越来越收到开发者，甚至设计师的青睐。

Storyboard

Storyboard 也叫“故事板”，是 Xcode 提供的一个十分强大的界面布局文件。它主要擅长视图设计以及视图之间的跳转逻辑设计，通过它可以更好地管理视图之间的关系，故事板文件的界面如图 4-2 所示。

输入名称和存储位置，以及所属分组和对象之后，即可完成创建。新创建出来的 Storyboard 文件不包含任何场景，基本上是一个完全空白的文件。

View

这个模板创建的是一个“xib”文件，这个 xib 文件包含有一个基本的、已经和 File's Owner 建立连接的 View 视图，Xib 文件的界面如图 C-6 所示。

Empty

这个模板创建的是一个“xib”文件，这个 xib 文件内不包含任何视图，基本上是一个完全空白的文件。

Window

这个模板创建的是一个“xib”文件，这个 xib 文件包含有一个 Window 视图。所谓 Window 视图，是用来管理和协调一组视图的控件，一般情况下使用的不多。

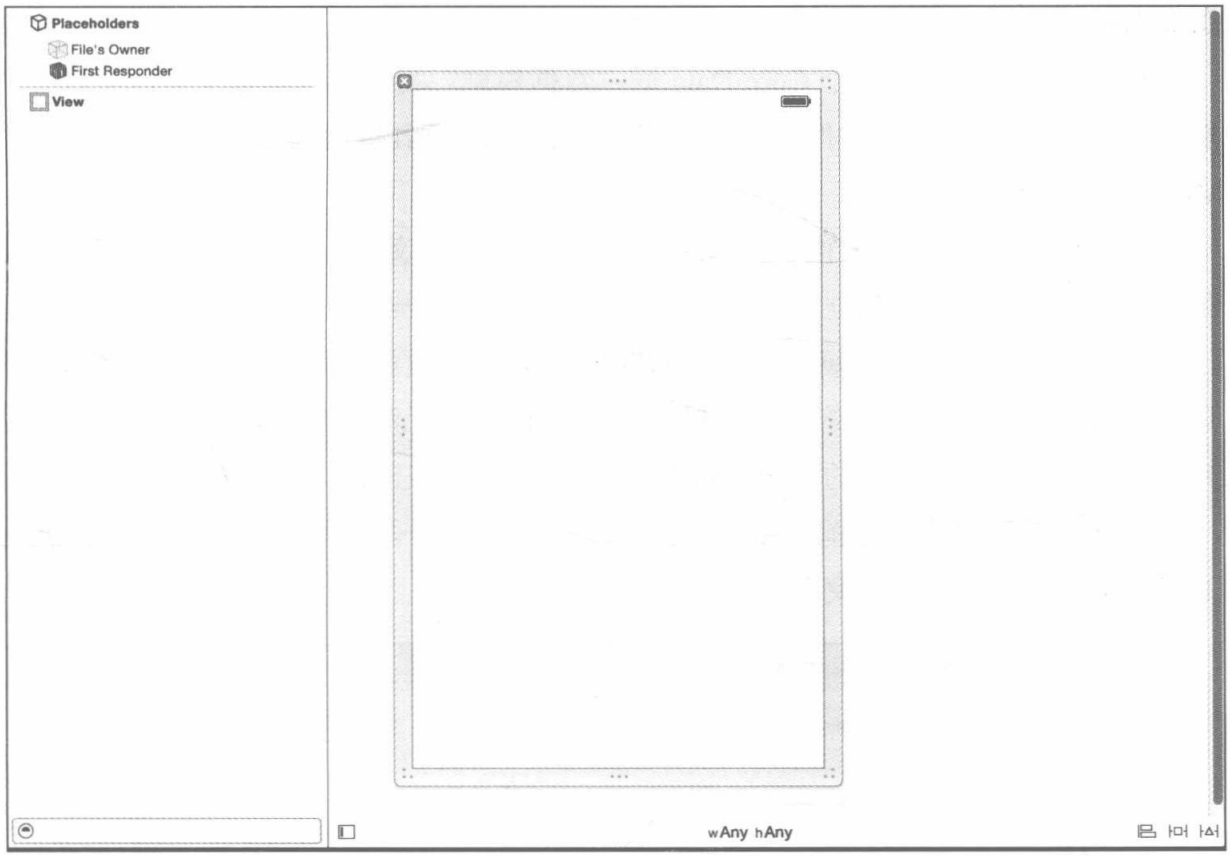


图 C-6　Xib 文件界面

Applcation

这个模板创建的是一个“xib”文件，这个 xib 文件包含有一个 Window 视图和一个 App Delegate 的对象。简而言之，这个新添加的 App Delegate 对象就是让该 xib 视图和 AppDelegate 这个类建立 IB 连接的转接点而已。

Launch Screen

这个模板创建的是一个“xib”文件，这个 xib 文件包含有一个 View 视图，这个视图里面已经有一部分由项目名、版权声明组成的“启动界面”元素。

这个文件主要是用来作为“启动界面”的，可以在对象设置的 General 属性卡的 App Icons and Launch Screen 栏目中设置 Launch Screen File 即可。在弹出的对话框中选择这个启动界面文件即可。

注意　Launch Screen 是 iOS 8 之后才提供的功能，如果你想要在 iOS 7 之前实现启动界面，那么只能在资源管理 Category Assets 文件夹当中进行设置。

C.1.1.3　Core Data 文件

Core Data 是一种“数据持久化”的框架，主要实现类似于 SQLite 数据库的数据管理、数据存储等功能。

Data Model

这个模板创建的是一个 Core Data 的“数据模型”文件，点击这个文件会打开“数据模型编辑器”，关于这个文件的更多内容，本书在第八章已经介绍过。

Mapping Model

这个模板创建的是一个 Core Data 的“映射模型”文件。所谓“映射模型”，就是在两个“数据模型”之间建立映射。

使用“映射模型”的场景在于：有时候要对数据模型的结构进行一次大的改动，甚至重新设计数据模型的结构，这时候，数据迁移已经无法完成此次功能，因此就需要映射模型来起到“中间层”的作用，帮助模型之间的数据迁移。

建立“映射模型”之前，Xcode 会要求开发者选择两个数据模型文件，因此这要求工程中至少拥有两个数据模型。

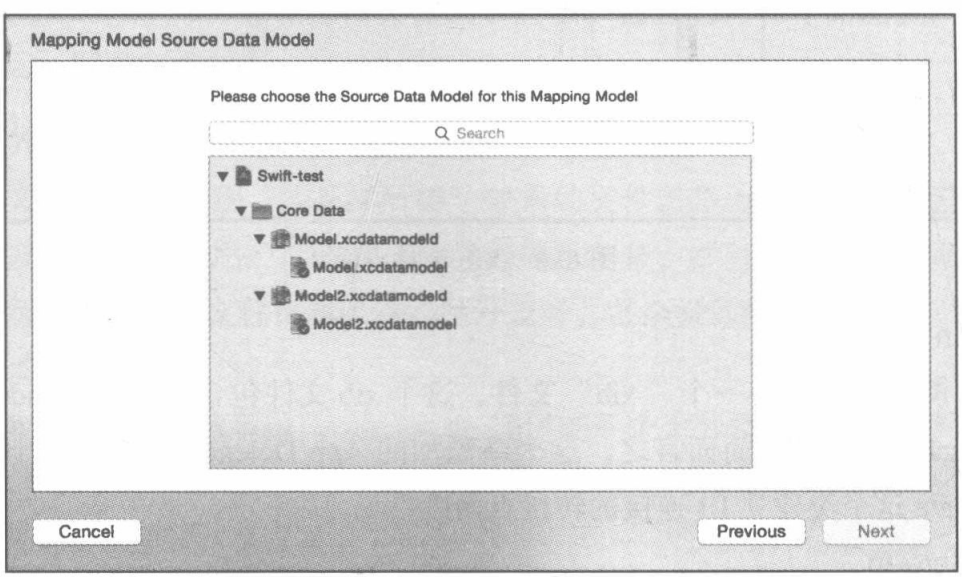


图 C-7　选择要迁移的数据模型

首先弹出来的对话框选择的是源数据模型，也就是要执行迁移的数据模型，如图 C-7 所示。选择其中一个数据模型后，单击“Next”，在弹出的对话框中选择目标数据模型，也就是要迁移到的数据模型，之前已选择的源数据模型将不可选。之后输入名称和存储位置，以及所属分组和对象之后，即可完成创建。

映射模型的界面如图 C-8 所示。

NSManagedObject subclass

这个模板创建的是一个继承 NSManagedObject 类的子类代码文件，用来建立与数据模型中的某个实体的关联绑定。

选择这个模板后，Xcode 会弹出一个对话框，要求开发者选择要绑定的数据模型。选择模型后，会要求选择该数据模型中的某个实体，来进行关联。单击“Next”后，输

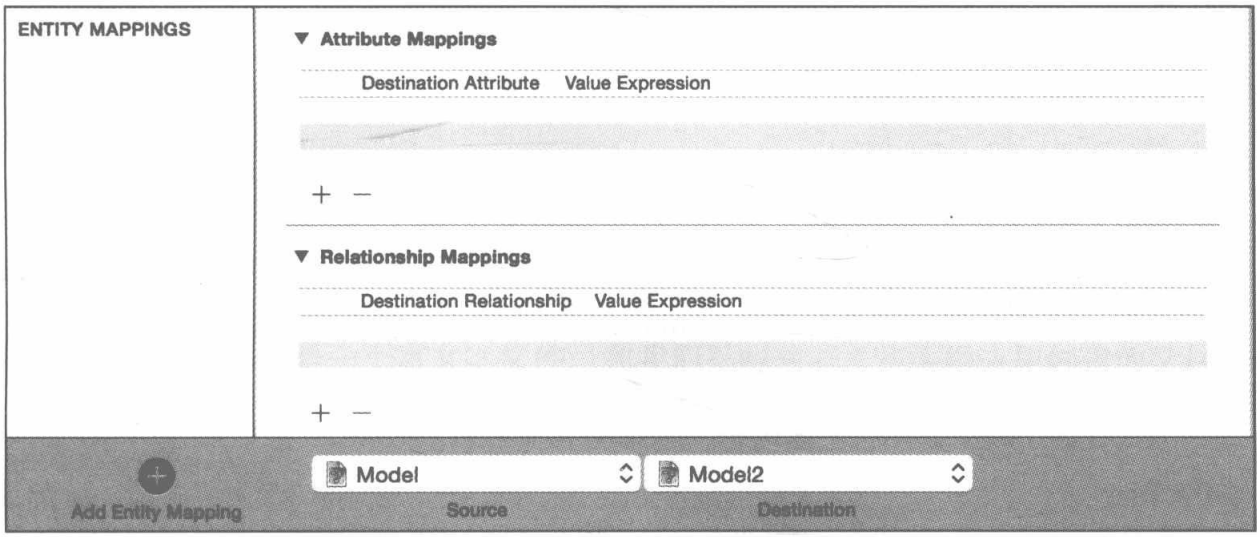


图 C-8　映射模型界面

入名称和存储位置，确定目标语言，以及所属分组和对象之后，即可完成创建。

无论在创建的时候选择了多少实体，Xcode 总会遵循一个实体对应一个管理对象文件的原则。

此外，建立文件的时候，该文件有一个属性：use scalar properties for primitive data types，意思是“使用原始数据类型的标量属性”。如果勾选上这个属性的话，那么建立的管理对象使用的就是我们定义的原始数据类型，否则的话会存在类型转化。转换规则如下：

- String 转换为 NSString (Objective-C) 或 String (Swift)
- Integer 16/32/64、Float、Double 和 Boolean 转换为 NSNumber
- Decimal 转换为 NSDecimalNumber
- Date 转换为 NSDate
- Binary 转换为 NSData
- Transformable 转换为 id (Objective-C) 或 AnyObject (Swift)

C.1.1.4　资源（Resource）文件

资源文件是 Xcode 在开发过程中，经常使用到的资源。它们都不是必须的，但是有了它们，可以实现加快开发进程，优化应用等功能。

GeoJSON File

该模板创建的是一个 GeoJSON 类型的文件，主要用在需要路径导航的应用当中。

所谓“ GeoJSON”，其实是一个基于 JavaScript 对象表示法的地理空间信息数据交换格式文件，简单来说就是记录位置的文件。它可以对各种地理数据结构进行编码，可以表示点、线、面、多点、多线、多面和几何集合等特征。

创建的 GeoJSON 的代码如下所示：

```
{
    "type": "MultiPolygon",
    "coordinates": [

    ]
}
```

大括号里面的整体内容代表了一个“对象”，type 表示其几何类型，MultiPolygon 代表了这是一个多边形类型的对象。coordinates 则是定义了其坐标，坐标必须遵循 x, y, z 的顺序。这两者是 GeoJSON 的基本组成要素。

关于 GeoJSON 文件的更多内容，请参阅其他资料。

GPX File

该模板创建的是一个 GPX 类型的文件，主要用在需要地理位置定位的应用当中。

GPX 文件主要是用于模拟一条路径或者一个地理位置，当我们在使用模拟器来运行应用时，会发现 Xcode 允许我们导入一个 GPX 路径文件，来模拟一条路径或者位置，从而达成测试应用的目的。

创建的 GPX 文件代码如下所示：

```
<?xml version="1.0"?>
<gpx version="1.1" creator="Xcode">

    <wpt lat="37.331705" lon="-122.030237">
        <name>Cupertino</name>
    </wpt>

</gpx>
```

可以看出，GPX 文件的本质上是一个 XML 格式的文件，标签内的内容就是 GPX 文件的主要内容，这里主要定义了一个经度 -122.030237，纬度 37.331705 的库比蒂诺市的地理位置坐标。这货也是苹果总部目前所在的地理位置。

Asset Catalog

该模板创建的是一个资源目录文件，关于资源目录的相关内容，本书在第三章已经介绍过。

Settings Bundle

该模板创建的是一个设置包文件，它属于我们介绍过的“包（Bundle）”的一种，主要是用来对 iOS 应用设置进行管理。也就是说，对这个文件进行操作，影响的是系统“设置”里面的设置属性。

默认情况下，创建的 Setting Bundle 包含一个 Root.plist 属性列表文件，以及其对应的本地化文件。Setting Bundle 除了提供接口供用户对程序进行个性化之外，还是一种数据存储的方式。

不过一般国内使用这个 Setting Bundle 供用户进行个性化设置的比较少，一般情况下设置都在应用内单独提供。

Property List

该模板创建的是一个属性列表文件，关于属性列表的相关内容，本书在第八章已经介绍过。

Rich Text File

该模板创建的是一个富文本文件，和我们平时使用文本编辑应用创建的文件是相同的，后缀名都是 rtf。

SceneKit Particle System

该模板创建的是一个用于 SceneKit 游戏框架的粒子文件，也就是绘制粒子效果的文件，如图 C-9 所示。

选择这个模板后，Xcode 会弹出一个对话框。从“Particle system template（粒子系统模板）”中可以选择一个系统自带的粒子效果作为模板，目前有 Bokeh（景深）、Confetti（彩屑）、Fire（火焰）、Leafs（落叶）、Rain（雨滴）、Reactor（喷射）、Smoke（烟雾）和 Stars（星耀）几种效果。

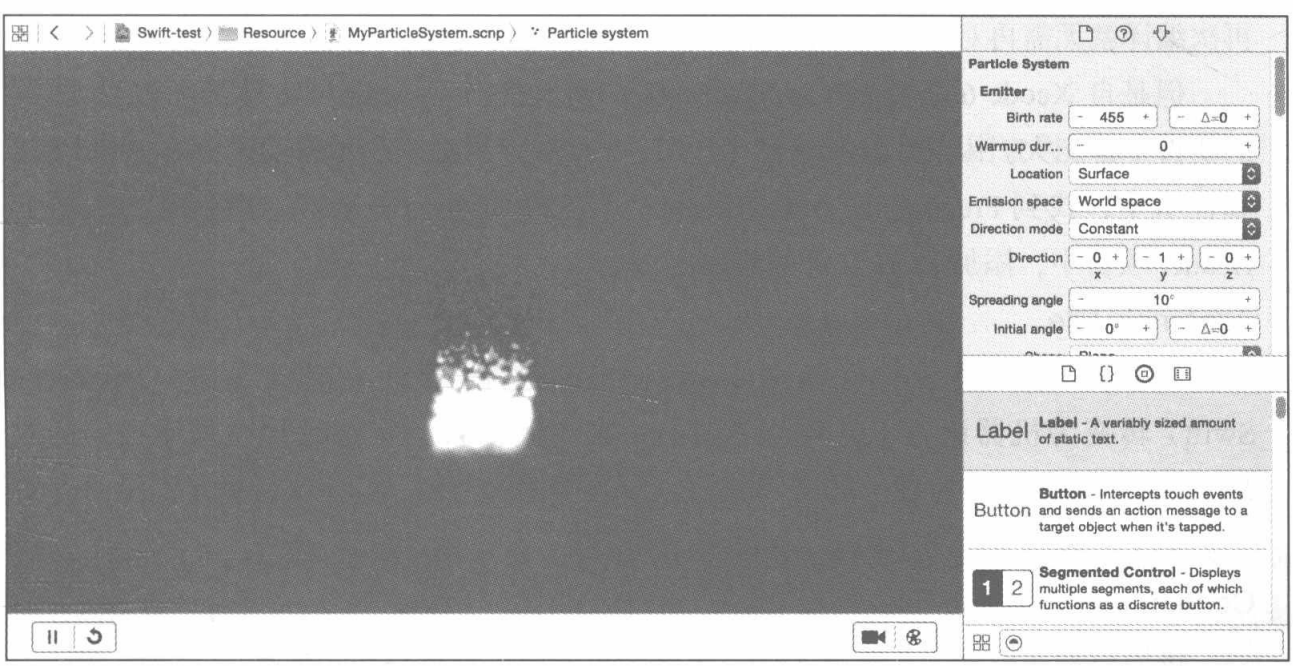


图 C-9　粒子文件模板

SpriteKit Particle File

和 SceneKit 的粒子文件类似，这个模板创建的是一个用于 SpriteKit 游戏框架的粒子文件。

创建文件过程中，粒子系统模板中选择系统自带的粒子效果有：Bokeh（景深）、Fire（火焰）、Fireflies（萤火）、Magic（魔幻）、Rain（雨滴）、Smoke（烟雾）、Snow（雪

花）和 Spark（烟花）。

SpriteKit Scene

这个模板创建的是一个 SpriteKit 的场景文件，对于 SpirteKit 来说，它的游戏界面都由“场景”来完成。

Strings File

这个模板创建的是一个 Strings 的本地化文件，关于国际化和本地化的相关内容，本书在第 7 章已经介绍过了。

C.1.1.5 其他（Other）文件

Empty

这个模板创建的是一个空文件，它仅有简单的文本编辑功能。

PCH File

这个模板创建的是一个特殊的头文件，它的全名为“ Precompile Prefix Header”，也就是预编译头文件。

所谓的预编译头文件，也就是预先将一些稳定代码编译好放在其中，这样就可以提高编译速度。因为引用头文件后，它们当中包含的东西往往非常庞大，预编译好之后，再次编译就无需再对这些头文件进行编译，减少等待时间。

但是自 Xcode 6 之后，PCH 就必须要手动添加支持了，这可能是因为 PCH 编译生成后变得十分庞大。因此，添加完这个 PCH 文件后，还必须在对象设置中的 Build Settings 属性卡中，找到 Precompile Prefix Header，将其值设置为“ Yes”，然后在下方的 Prefix Header 项目中，添加 PCH 文件的路径。

Assembly File

该模板创建的是一个汇编文件，所谓汇编文件，就是在高级语言（Objective-C 或 Swift）转变为机器语言（01 码）之间的一种中间语言。有些时候在程序报错的时候，Xcode 很有可能会停留在一堆诸如 mov a1,5 之类的文件内，这个文件就是汇编语言文件。

关于汇编语言的更多内容，在此不予介绍。

Configuration Setting File

该模板创建的是一个配置文件，主要是对编译过程进行相关配置。

Resource Rules

该模板创建的是一个资源规则文件，是一个类似于属性列表的文件，主要用来对资源（比如说 Info.plist）进行管理。

实际上，Xcode 创建的项目中已经默认带有这样一个资源规则文件了，如果要进行更改，那么在 Build Settings 属性卡中的 Code Signing Resource Rules Path 中改成新创建的资源规则文件地址就可以了。

Shell Script

该模板创建的是一个 Apple Script 脚本文件，这个脚本文件的用处非常大，可以在很多地方使用，主要是用来增加 Xcode 的功能。关于 Apple Script 的使用，请参阅其他资料。

C.1.1.6　Apple Watch 文件

Storyboard

和用户界面文件中的 Storyboard 文件类似，这个模板创建的也是一个 Stroyboard 文件，不过这个故事板文件是专门为 Apple Watch 设计的用户界面文件。

和一般的故事板文件有所不同，Apple Watch 故事板文件的底部不再是“Size Classes”，取而代之的是“Apple Watch Size”，用来选择不同尺寸的界面。目前有两种选择“Apple Watch 42 mm”和“Apple Watch 38 mm”。

此外，相比一般的故事板文件来说，Apple Watch 故事板文件取消了尺寸检查器这一个检查器。

Notification Simulation File

这个模板创建的是 apns 文件，它以 JSON 格式来模拟苹果推送服务，也就是模拟从 iPhone 上推送过来消息，从而测试 Watch 应用是否工作正常。

关于模拟推送的更多内容，请参阅 Apple Watch 相关的资料。

WatchKit Settings Bundle

和 Setting Bundle 类似，这个模板提供的是一个 bundle 包，用来为 Watch 应用进行系统配置设置。

C.1.2　Mac OS 文件

Mac OS 文件包括代码文件和用户界面文件等。

C.1.2.1　代码（Source）文件

Cocoa Class

Cocoa 是 OS X 平台应用的核心框架，它提供了一系列的 API，用来支持 OS X 平台上的应用。Cocoa Class 和 Cocoa Touch Class 的作用相似，用来创建一个基于 Cocoa 框架的类模型文件。

C.1.2.2　用户界面（User Interface）文件

Application

该模板创建的是一个 xib 类型的文件，适用于 OS X 的 xib 文件如图 C-10 所示。它里面包含有一个 Window 视图和两个对象。一个是字体管理器（Font Manager），用来管理应用中含有的字体；另一个是 Menu 菜单，也就是应用最顶部的那条菜单栏。

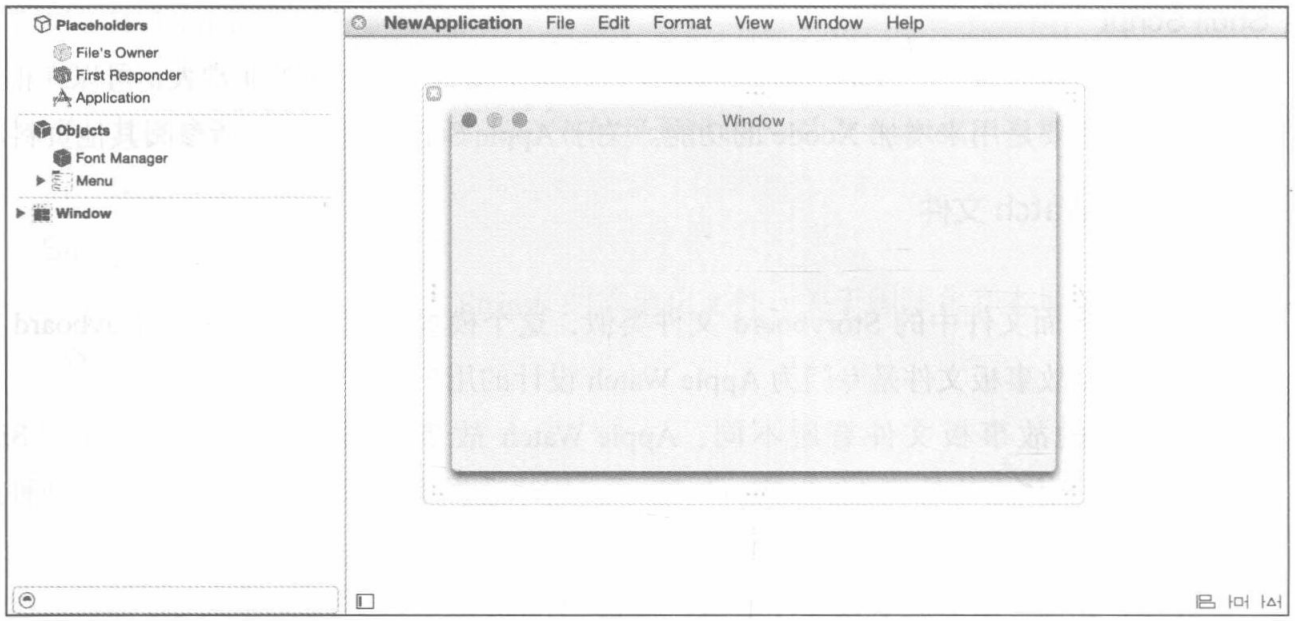


图 C-10　适用于 OS X 的 xib 文件

这个 Application 模板可以包含多个窗口（Window），每个 Window 可以包含多个视图（View）。这个目录层级关系在 OS X 上比在 iOS 上要清晰、明显得多。

Main Menu

该模板创建的是一个 xib 类型的文件，它是 Application 模板中两个对象的单独实现，也就是单独一个文件来管理字体管理器和菜单栏。

Window

该模板创建的是一个 xib 类型的文件，它是 Application 模板中 Window 视图的单独实现，也就是单独一个文件来管理窗口视图。

View

该模板创建的是一个 xib 类型的文件，它是 Window 模板中 View 视图的单独实现，也就是单独一个文件来管理视图。和 iOS 传统的 xib 功能类似。

Empty

该模板创建的是一个空的 xib 类型的文件。

Storyboard

该模板创建的是一个适用于 OS X 平台的故事板文件，和 iOS 的故事板文件是非常类似。

不过 OS X 的故事板文件并不支持 Size Classes 功能，因为系统的实际原因，OS X 并不能像 iOS 那样简单地对屏幕进行划分，因此在这个故事板里面我们无法使用 Size Classes 功能。

此外，OS X 的故事板文件还新增了两个检查器：绑定检查器（Bindings inspector）和视图特效检查器（View Effects inspector）。

C.1.2.3　其他（Other）文件

Exports File

Exports 文件是一种服务器端的配置文件，主要用来配置需要向客户端共享的目录和权限等设置。

C.2　项目模板

表 C-1　项目模板列表

项目模板名称	所属区域	功　能
Master-Detail Application	iOS-Application	这个项目模板将创建一个能够查看详情的信息应用，就如同“备忘录”应用的效果。模板提供了增加记录、删除记录功能，并且点击记录还能够查看该记录的详情信息
Page-Based Application		这个项目模板将创建一个带有翻页效果的展示应用
Single View Application		这个项目模板将创建一个带有一个空白视图的简单应用，这也是一般情况下创建新项目的入口
Tabbed Application		这个项目模板将创建一个底部带有多个标签的应用，就如同“微信”应用的效果
Game		这个项目模板将创建一个简单的游戏应用，在弹出的项目设置对话框中选择对应的游戏技术，即可完成游戏的创建。可以选择的游戏技术有 SpriteKit、SceneKit、Metal 和 OpenGL ES
Cocoa Touch Framework	iOS-Framework & Library	这个项目模板将创建一个框架项目，这个项目不能运行，但是能够编译出框架
Cocoa Touch Static Library		这个项目模板将创建一个静态库项目，这个项目不能运行，但是能够编译出静态库
In-App Purchase	iOS-Other	这个项目模板将创建一个内置付费项目，让用户无缝升级软件功能或者扩充软件内容，苹果规定所有的应用内付费项目都得使用此项功能
Empty		这个项目模板将创建一个没有任何文件的空项目
Cocoa Application	Mac-Application	这个项目模板将创建一个标准的 Mac 应用，带有完整的用户界面和代码设计
Command Line Tool		这个项目模板将创建一个控制台运行的程序
Cocoa Framework	Mac-Framework & Library	这个项目模板将创建一个框架项目，这个项目不能运行，但是能够编译出框架
Library		这个项目模板将创建一个库项目，这个项目不能运行，但能够编译出库。在创建该项目的时候可以选择创建哪种语言的动态库还是静态库
Bundle		这个项目模板将创建一个包项目，这个项目不能运行，但是能够编译出包

（续）

项目模板名称	所属区域	功　能
XPC Service	Mac-Framework & Library	这个项目模板将创建一个 XPC 服务项目，这个项目不能运行，但是能够编译出 XPC 服务
Address Book Action Plug-in	Mac-System Plug-in	这个项目模板将会创建一个通讯录插件项目，这个插件将可以给通讯录实现更多的自定义功能
Automator Action		这个项目模板将会创建一个 Automator 插件，扩展 Automator 当中的动作集
Generic C++ Plug-in		这个项目模板将会创建一个通用的 C++ 模板，采用 C++ 标准库和 Dwarf 调试技术，向外暴露一个 C 接口
Generic Kernel Extension		这个项目模板将会创建一个通用的内核扩展
Image Unit Plug-in		这个项目模板将会创建一个滤镜插件，用在 Core Image 和 Core Video 的图像处理上
Installer Plug-in		这个项目模板将会创建一个安装器项目，完成对目标项目打包以供安装的操作
IOKit Driver		这个项目模板将会创建一个驱动程序项目，用来控制 IO 设备
Preference Pane		这个项目模板将会为 OS X 程序创建一个 Preference 偏好设置窗格。一般情况下对应了菜单栏→应用名→系统偏好设置中的设置
Quartz Composer Plug-in		这个项目模板将会创建一个用于 Quartz Composer 工作区的插件，用做一个可供复用的单元
Quick Look Plug-in		这个项目模板将会创建一个快速查看的插件，让 OS X 系统使用快速查看的时候能够运行该插件，提供一些特殊功能
Screen Saver		这个项目模板将会创建一个屏幕保护程序
Spotlight Importer		这个项目模板将会创建一个 Core Data Spotlight 导入器
Cocoa-AppleScript	Mac-Other	这个项目模板将会创建一个使用 AppleScript 语言编写的 OS X 应用
External Build System		这个项目模板将会创建一个自定义的编译系统，开发者可以用自己的编译系统来完成代码编译

C.3 应用扩展

应用扩展（App Extension）是 iOS 8 以及 OS X Yosemite 带来的新功能，它能够让用户在使用其他应用时，使用一些由你的应用所提供的额外功能。

应用扩展和应用并不相同，应用是能够在 iOS 设备或者 Mac 设备上单独运行的程序，而应用扩展则必须依附于应用来开发，这也是其被称之为“扩展”的主要原因之一。不过，应用扩展的存储内容是独立于应用的，它的可运行代码都存放在一个独立的二进制文件当中，

但是它的调启和使用一般都需要依附应用来进行。

从 Xcode 当中看，应用扩展同样也是一个对象。你可以在项目中创建多个依附于单个应用的应用扩展对象，这个被依附的应用被称为载体应用（Containing App）。


提示　在 iOS 中，包含扩展的应用必须要提供一个扩展之外的功能。而在 OS X 中没有这个硬性要求。

要注意的是，苹果关于应用扩展有着非常严格的管理机制，在开发应用扩展的时候，最好参照苹果的相关规定进行开发，否则应用会面临审核遭拒或者下架的风险。

每个应用扩展都需要针对一个定义明确的使用场景，因此在创建应用扩展前，需要弄清楚这个扩展能够为用户提供什么样的功能，然后再根据这个功能来选择合适的扩展对象。

iOS 和 OS X 定义了几种应用扩展的类型，每种类型的扩展都对应系统中的一块区域，比如说分享、通知中心、输入法等等，我们将这个支持扩展的区域称之为扩展点（extension point）。每个扩展点都定义了其使用策略以及 API，因此我们可以使用该区域来创建应用扩展。要注意的是，当我们针对某个扩展点开发应用扩展时，这个扩展的功能必须要符合该扩展点的功能特性并遵守一定的规则。

表 C-2 展示了 iOS 和 OS X 中所提供的扩展点，并简要的说明了这些扩展所实现的基础功能。

表 C-2　扩展点及其基础功能

扩 展 点	适 用 平 台	基 础 功 能
Today	iOS & OS X	在通知中心的“今天”视图中提供最新消息或者执行一些简单操作
Share	iOS & OS X	向载体应用提交要分享的网站或者内容
Action	iOS & OS X	处理或者查看发生在载体应用当中的内容
Photo Editing	iOS	在“照片”应用中编辑图片或者视频
Finder Sync	OS X	直接在 Finder 中展示文件同步状态信息
Document Provider	iOS	提供访问和管理文件系统的能力
Custom Keyboard	iOS	为 iOS 系统提供一个自定义的输入法，可以用在所有应用当中，以替代系统输入法
Watch App	iOS	为 Apple Watch 提供应用、glance 或者通知 UI

创建应用扩展最好是使用 Xcode 所提供的应用扩展模板，这些模板都包含有这些应用扩展的具体实现文件以及相应的设置，并自动向应用程序的包中生成独立的二进制文件。

一般情况下，Xcode 提供的应用扩展模板就是可以直接运行的，编译成功后，就会生成一个扩展名为 .appex 的应用扩展包。

注意 应用扩展必须要包含 arm64 架构，否则在上传 App Store 时会被拒绝。

C.4 控件模板

C.4.1 控制器

控制器在 MVC 架构中扮演控制层的角色，也就是说，它负责与模型层进行数据交换，并且对视图层进行控制和接收反馈。在 Xcode 的 Stroyboard 当中，苹果将控制器在设计上一分为二，一个专门用来控制视图的静态显示，一个则是负责控制器的其他工作。负责控制静态视图显示的控制器，就是出现在 Storyboard 当中的 Controller 元素。

这个控制器需要与 Controller 类建立关联，从而让 Controller 类能够控制这个控制器界面，从而控制界面里面的视图，实际上它们是一个东西。

通过标识检查器的 Custom Class 栏目，可以设定这个控制器所关联的类，还可以通过 Identity 设置这个控制器的 ID，从而可以通过代码使用此 ID 来访问这个控制器。

下面我们来介绍这些控制器的大致功能和检查器属性。

1. 视图控制器

视图控制器（View Controller）的样式如图 C-11 所示，其对应类是 UIViewController。

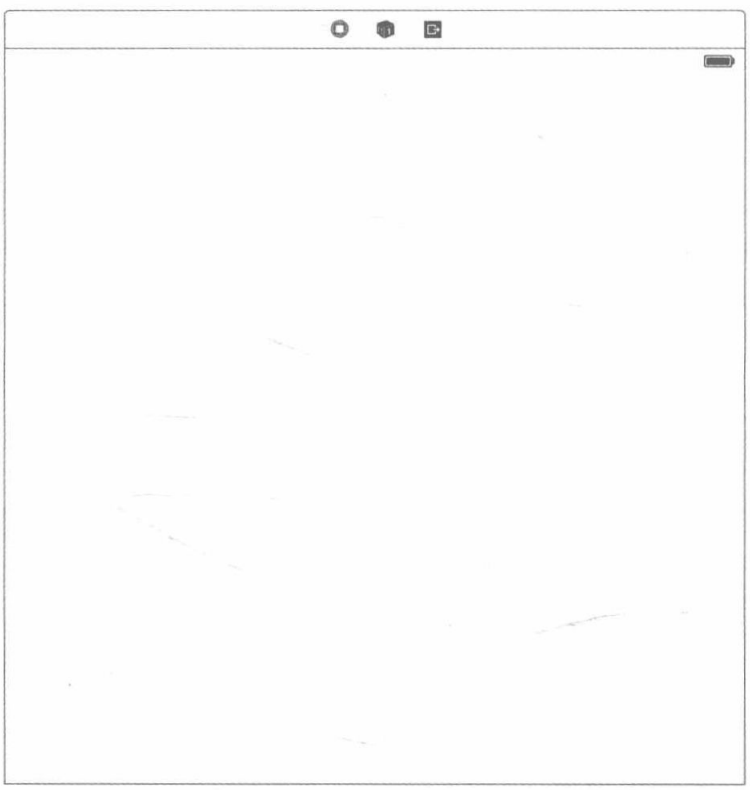


图 C-11 View Controller

对于视图控制器来说，它的属性检查器有以下几个属性可供设置。首先是 Simulated Metrics（模拟视图），这个属性是实现虚拟设计的，用来展示视图的最终模样，但是并不会在运行时出现。默认情况下这些设置都为“inferred”，这意味着这个控制器界面将按照项目设置来进行显示。

- Size（尺寸）：用来改变视图控制器的显示尺寸，可以选择多个不同的既定尺寸和显示模式。
- Orientation（方向）：用来改变视图控制器的显示方向，可以选择横向（Landscape）还是竖向（Portrait）
- Status Bar（状态栏）：用来确定状态栏的显示模式，可以选择隐藏状态栏或者显示状态栏，状态栏还有两种颜色模式可以选择。
- Top Bar（顶栏）：用来确定顶部导航栏的显示模式。
- Bottom Bar（底栏）：用来确定底栏的显示模式。

然后就是控制器的专属设置：

- Title（标题）：设置这个属性将会影响到导航栏标题的显示
- is Initial View Controller：设置入口点，确认某个控制器是否是最先加载的控制器。
- Layout（约束）：对这个控制器管理的布局相关属性进行设定。

Adjust Scroll View Insets（调整滚动视图插页）决定是否响应滚动视图的滚动动作，如果取消该属性，那么滚动视图将无法滚动；

Hide Bottom Bar on Push（在 Push 页面的时候隐藏底栏）决定是否在执行 Push 页面动作的时候，隐藏底栏的显示。

Resize View From NIB（重置 NIB 视图的尺寸）则决定当从 NIB 中加载视图时，是否根据设备的实际尺寸来调整视图的大小，如果不调整的话，那么视图将按照既定值加载；

Use Full Screen（使用全屏）是一个不赞成使用的方法，因为全屏之后，状态栏将会被隐藏。

- Extend Edges（边界扩展）：决定视图边缘应该朝哪一个方向扩展，Under Bottom Bars 是向底部边缘扩展，Under Top Bars 是向顶部边缘扩展，Under Opaque Bars 则意味着当栏目使用了不透明图片时，试图是否延伸至栏目所在区域。
- Transition Style（切换动画样式）：这个属性决定了当前控制器在场景切换时的动画样式。总共有 4 种动画样式，分别是 CoverVertical（底部滑入）、FlipHorizontal（水平翻转）、CrossDissolve（交叉溶解）以及 ParticalCurl（翻页）。
- Presentation（弹出风格）：这个属性决定了当前控制器在被弹出时的风格，总共有 4 种风格，分别是 FullScreen、PageSheet、FromSheet 和 CurrentContext，一般用在 iPad 上面。

FullScreen 模式很简单，弹出的控制器将充满全屏，无论横屏还是竖屏状态。

PageSheet 模式则是让弹出的控制器高度等同于当前屏幕高度，宽度等同于竖屏模式下的屏幕宽度，剩余区域将会变暗阻止用户点击。也就是说，这个模式在竖屏下和 FullScreen 没有任何区别，但是在横屏下两边将会留下变暗的区域。

FromSheet 模式则是类似于警告框的显示样式，控制器在视图中间显示，并且四周留下变暗的区域。

CurrentContext 模式则是让当前控制器的弹出方式和父控制器的弹出方式相同。

- Defines Context（定义上下文）属性则是设置 definesPresentationContext 属性的，将这个属性设置为 true 则表示该控制器弹出之后，将会覆盖其父控制器。
- Provides Context（提供上下文）属性则是设置 providesPresentationContextTransitionStyle 属性的，如果此属性的值设置为 true，那么该控制器的弹出风格将使用父控制器的弹出方式。
- Content Size（内容尺寸）属性则是用来确定控制器的内容大小的，应用这个属性后，控制器当中的内容视图将固定为这里设置的宽高，一般用在弹出框当中。
- Key Commands（快捷键命令）属性则是用来添加自定义的快捷键的。

2. 导航控制器

导航控制器（Navigation Controller）的样式如图 C-12 所示，它除了自带了一个导航控制器之外，还带了一个表视图控制器，买一赠一，十分划算。其对应类是 UINavigationController，继承自 UIViewController。

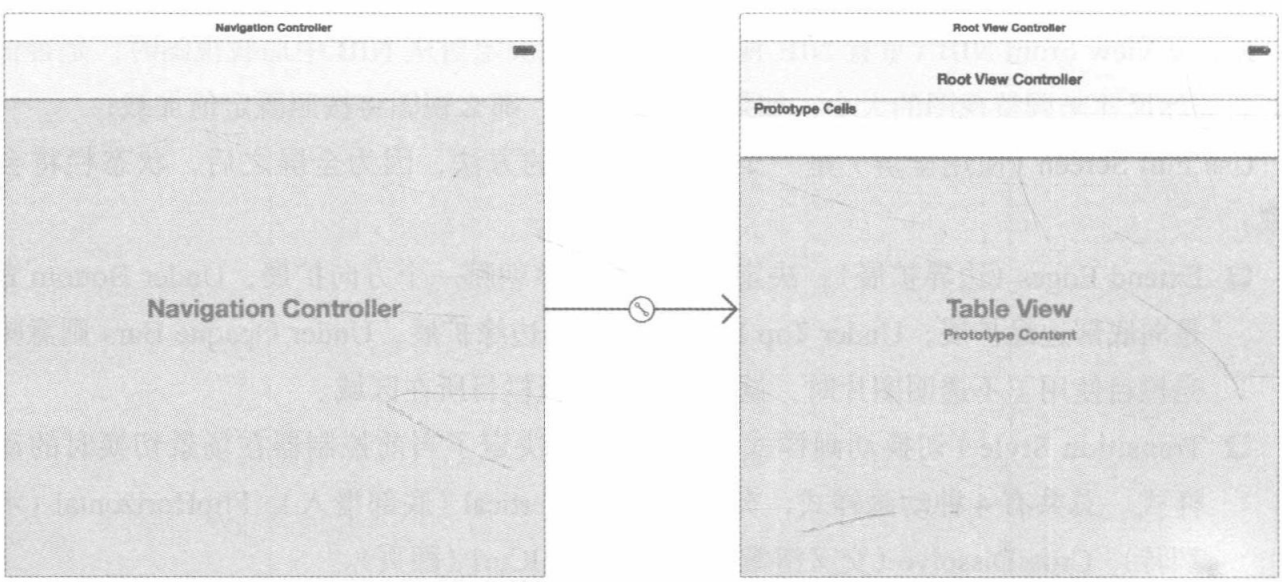


图 C-12　Navigation Controller

对于导航控制器来说，它的主要用途是给与之联系的控制器视图上添加一个共有的导航栏，以便用户能够方便的控制界面之间的层级关系，这也是最常见的 iOS 层级控制方法之一。

除了视图控制器有的属性外，导航控制器还拥有自己特有的属性：

- Bar Visibility（栏目可视性）：用来控制导航栏和底栏是否显示
- Hide Bars（隐藏栏目）：控制在何种状态下隐藏导航栏和底栏，属性分别有：Swipe（轻扫）手势下、Tap（单击）手势下、键盘出现时以及屏幕横置时。当这些情景出现时，导航栏和底栏都会被隐藏。

3. 表视图控制器

表视图控制器（Table View Controller）的样式如图 C-13 所示，其对应类是 UITableViewController，继承自 UIViewController。

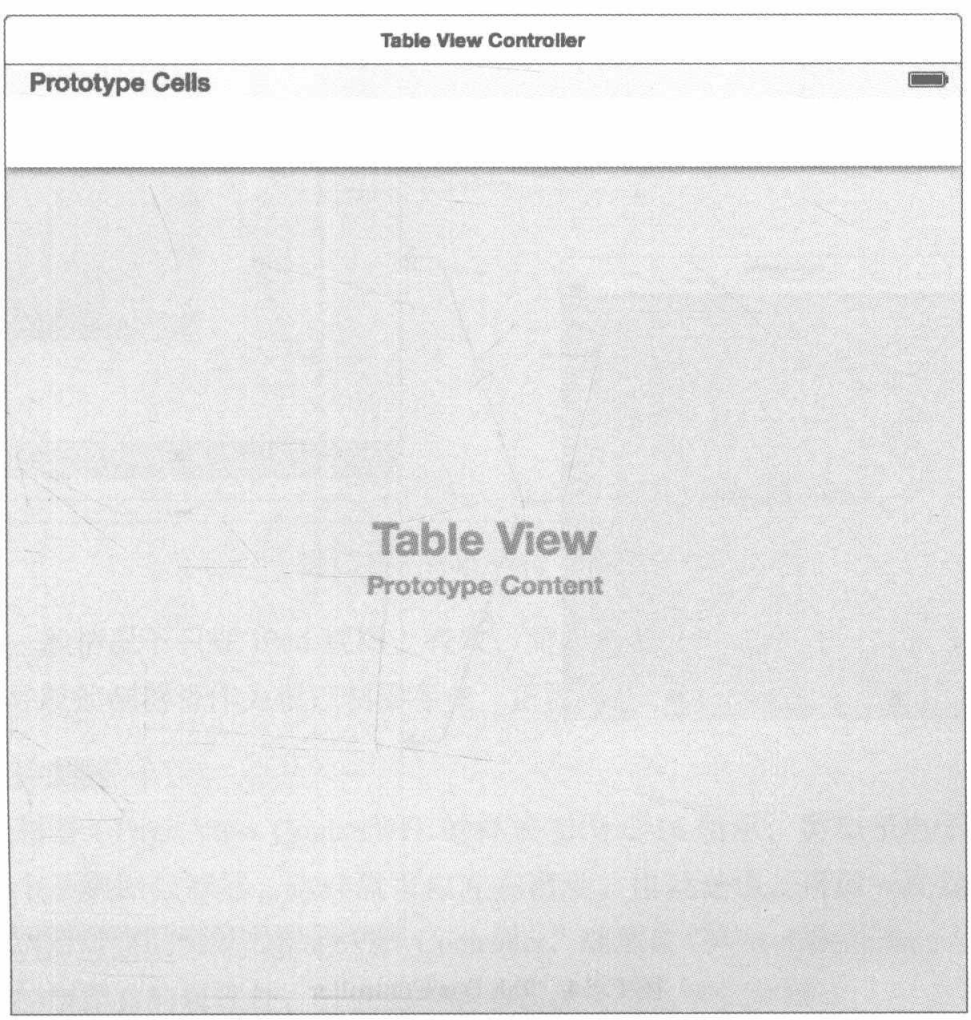


图 C-13　Table View Controller

对于表视图控制器来说，它的主要用途是以表格的形式来展示和实现各种各样的内容显示功能，可以说是最常用的视图控制器之一。

除了视图控制器有的属性外，表视图控制器还拥有自己特有的属性：

- Selection（选择）：Clear on Appearance（在显示时清除）指的是当点击表视图中的单元

格进入到子界面后，该单元格的选中样式将会被清除。

- Refreshing（刷新）：用来确定是否允许当前表视图进行刷新，如果允许的话，表视图在下拉之后就会显示一个加载视图，表示正在进行数据刷新，并且还可以配置刷新过程中的提示文字。

4. 选项卡控制器

选项卡控制器（Tab Bar Controller）的样式如图 C-14 所示，它除了自带了一个选项卡控制器之外，还带了两个视图控制器，分别对应模板中自带的两个属性卡，选项卡数目越多，其关联的视图控制器数目也越多。其对应类是 UITabBarController，继承自 UIViewController。

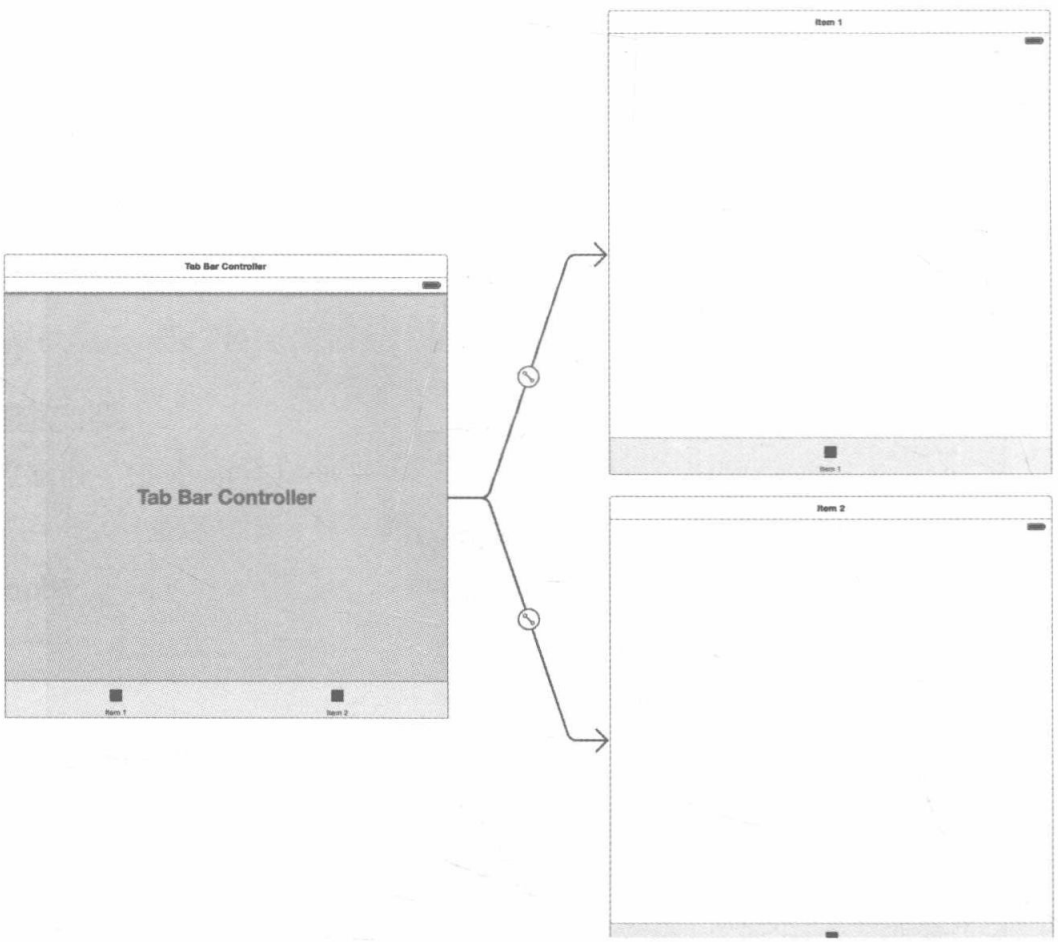


图 C-14　Tab Bar Controller

对于选项卡控制器来说，它一般情况下不要超过 5 个选项卡。如果多于 5 个选项卡，那么多余的选项卡就会被放置到一个特殊的 More View Controller 来管理这些多余的选项卡。和导航控制器类似，它的主要用途是给与之联系的控制器视图上添加一个共有的底部选项卡栏，以便用户能够方便地控制界面之间的层级关系，这也是最常见的 iOS 层级控制方法之一。

5. 分隔视图控制器

分隔视图控制器（Split View Controller）的样式如图 C-15 所示，它除了自带了一个分隔视图控制器之外，还带了一个导航控制器、一个表视图控制器和一个视图控制器。其中，导航控制器和表视图控制器用来作为侧滑时弹出的窗口，视图控制器作为主视图。这个控制器对应的类是 UISplitViewController，继承自 UIViewController。

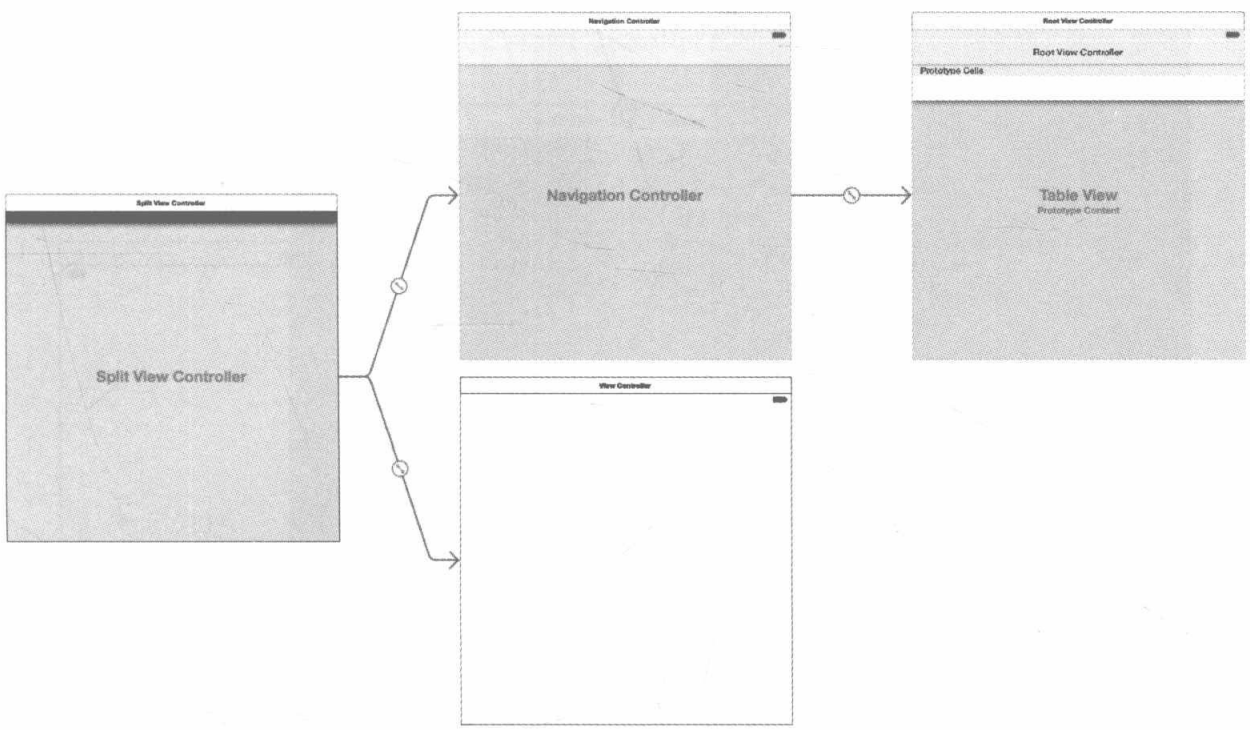


图 C-15　Split View Controller

这个效果一般情况下只在 iPad 视图上有效，而如果在 iPhone 视图上，这个控制器将会无效，这上面的导航控制器将作为根控制器存在。正因如此，这个样式在 iPad 上也十分常见。

6. 页视图控制器

页视图控制器（Page View Controller）的样式如图 C-16 所示，类似于电子书的形式，它当中可以放多个子视图控制器，这样就可以左右滑动，也就是说，可以一次显示多个视图控制器。这个控制器对应的类是 UIPageViewController，继承自 UIViewController。

除了视图控制器有的属性外，页视图控制器还拥有自己特有的属性：

- Navigation（导航）：设置页视图控制器导航的方向，可以是水平（Horizontal）方向，也可以是垂直（Vertical）方向。
- Transition Style（切换动画样式）：和其继承的视图控制器的切换动画样式一样，但是这个设置会覆盖掉其继承的视图控制器的切换动画样式。
- Page Spacing（页面间隔）：定义两个页面之间的间隔距离。
- Spine Location（书脊位置）：书脊位置定义了视图的分隔模式，比如说如果使用 None

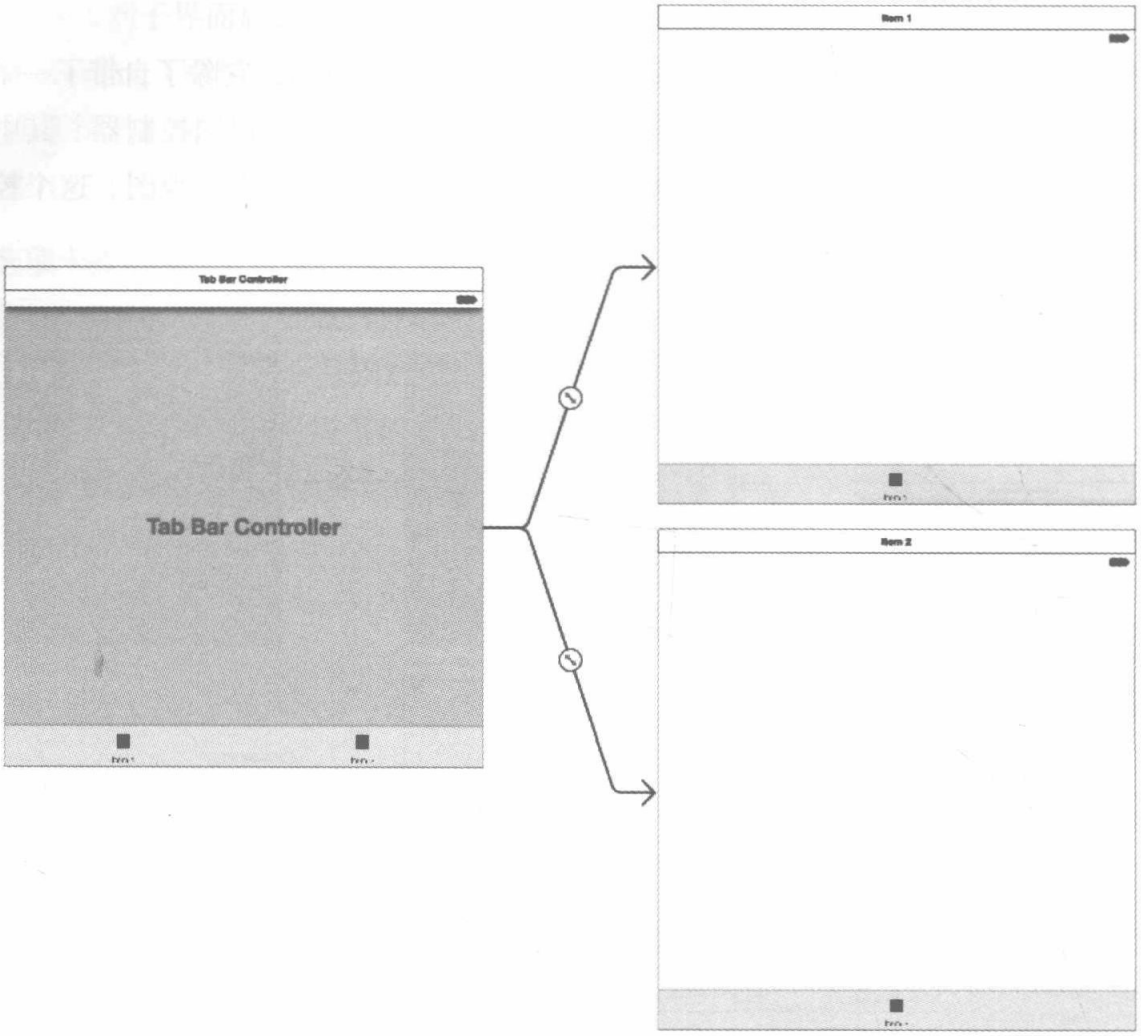


图 C-16 Page View Controller

模式，那么该控制器将一直使用一个视图；而使用 Min 模式，那么该控制器将只显示一个视图，当翻页时提供另一个新的视图；而如果是 Mid 模式，那么该控制器将在中间放置一个书脊，因此需要提供两个视图；如果是 Max 模式，则是提供多少视图就显示多少视图。

- Double Sided（双面显示）：当页面翻起的时候，偶数页面就会在书页的背后显示，即模拟真实的“书页显示”效果。

7. GLKit 视图控制器

GLKit 视图控制器（GLKit View Controlller）是一个用来显示 OpenGL ES 视图的控制器，它负责完成 OpenGL ES 项目的基本配置。这个控制器对应的类是 GLKViewController，继承自 UIViewController。

除了视图控制器有的属性外，GLKit 视图控制器还拥有自己特有的属性：

- Preferred FPS（FPS 最大值）：控制器将会多次调用 draw 方法，设置该值将限定每秒调用该方法的次数，也就是对应的 FPS 值。这个属性还带有两个设置项：Pause on

Deactivate（不活跃时暂停）属性将决定当该控制器不处于前台活跃状态时，是否暂停运行以节省内存消耗；Resume on Activate（活跃时重启）属性将决定该控制器转为前台活跃状态时，是否重新开始绘制。

8. 集合视图控制器

集合视图控制器（Collection View Controller）的样式如图 C-17 所示，其对应类是 UICollectionViewController，继承自 UIViewController。

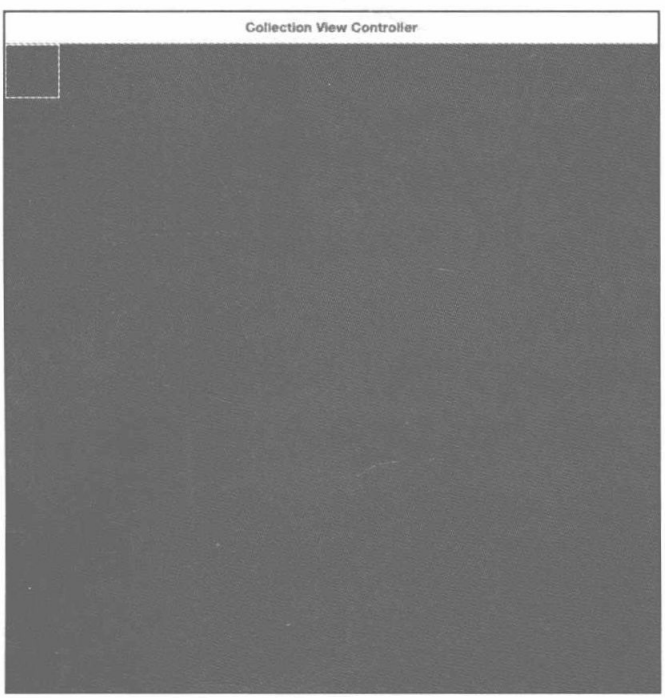


图 C-17　Collection View Controller

集合视图控制器也是一个非常常见的视图控制器之一，一样也是用来展示内容。“照片”应用的照片展示就使用了集合视图控制器来显示照片，实现“画廊”效果。

9. 视频播放器视图控制器

视频播放器视图控制器（AVKit Player View Controller）对应类是 AVPlayerViewController，继承自 UIViewController。

视频播放器视图控制器主要用来进行视频的播放效果，我们在一些应用上播放视频时会弹出一个专门的视频播放窗口，这个就是视频播放器视图控制器。

除了视图控制器有的属性外，视频播放器视图控制器还拥有自己特有的属性：

- Shows Playback Controls（显示回放控制）：决定该控制器是否显示控制状态栏，从而允许用户进行回放控制。
- Video Gravity（视频尺寸）：主要设定视频的尺寸匹配样式，可以选择 Resize、Resize Aspect 和 Resize Aspect Fill 三种。

10. 搜索栏和搜索显示控制器

这个元素除了带有一个搜索栏外，还带了一个搜索显示控制器（Search Display Controller），这个控制器没有独立的视图，但是可以通过该控制器来控制搜索栏的显示行为。

搜索显示控制器只有一个单独的属性，用来设置搜索结果视图的标题。

C.4.2 视图

视图在 MVC 架构中扮演展示层的角色，也就是说，它负责将 UI 元素展示给用户，以及接收用户的反馈。Storyboard 上的视图一般情况是静态的布局，如果要提供动态布局的话，那么只能使用 xib 或者纯代码了。

视图需要放置在某个控制器内，并且和控制器类建立 Outlet 连接，这样控制器类才能够对视图进行控制和操作。

通过标识检查器的 Custom Class 栏目，可以设定这个视图所关联的类。

下面我们来介绍这些视图的大致功能和检查器属性。

1. 视图

View 视图是最基本的视图，基本上所有的视图都继承于它，它的对应类是 UIView。

对于视图来说，它的属性检查器有许多属性可以配置：

- Mode（显示模式）：这个属性决定了当视图边界发生改变时，其中的内容应该如何适应这个改变。不过任何让图像进行缩放的操作都可能增加处理开销，因此最好不用。一般情况下有如下几个属性：

Scale To Fill	适当改变内容的比例，将内容填充整个屏幕
Aspect Fit	缩放内容到合适的大小，边界多余部分透明
Aspect Fill	缩放内容填充到指定大小，边界多余的部分省略
Redraw	重绘视图边界
Center	视图保持等比缩放
Top	视图顶部对齐
Bottom	视图底部对齐
Left	视图左侧对齐
Right	视图右侧对齐
Top Left	视图左上角对齐
Top Right	视图右上角对齐
Bottom Left	视图左下角对齐
Bottom Right	视图右下角对齐

- Tag（标签）：视图的标签，是一个整数，一般用在视图集合当中区分不同的视图。

- Interaction（交互）：规定了其响应用户交互的状态，User Interaction Enabled（响应用户交互）将决定该视图是否能够响应用户的交互动作，否则对其进行任何操作都无法被响应；Multiple Touch（多点触控）将决定该视图是否允许多点触控手势。
- Alpha（透明度）：决定视图的透明程度，数值从 0 到 1，0 表示完全透明。
- Background（背景颜色）：设置视图的背景颜色。
- Tint（渲染颜色）：当屏幕上的控件是“活跃（Active）”的，并且控件默认使用 tint color 的时候，那么它就会呈现这个颜色。也就是说，视图中的所有子视图都将继承这个颜色。
- Drawing（绘图属性）：这一系列属性决定了绘制这个视图的配置。

Opaque（不透明）属性表示当前视图是否不透明，当其为 true 的时候，GPU 将不会考虑其下方图层的颜色，也就是直接简单的绘制这个层，否则的话 GPU 将会根据其下方图层的颜色，绘制出一个“混合”的颜色。比如说，红色视图下方拥有了一层绿色视图，那么如果取消了 Opaque 属性，那么实际绘制出来的视图颜色是它们之间的混合色——黄色。需要注意的是，如果确认勾选了 Opaque 属性，那么 alpha 值就必须为 1，否则会发生一些不可预料的后果。

Hiiden（隐藏）属性表示当前视图是否隐藏，当勾选之后，该视图将会隐藏起来，不过其仍然还是会被绘制出来。

Clears Graphics Context（清除图层上下文）：选中之后，系统将会先使用黑色填充视图覆盖的区域，然后再绘制控件。

Clip Subviews（切割子视图）：选中情况下，只有在父视图范围内的子视图部分将被绘制出来。默认情况下是未选中的。一般情况下，子视图会在父视图的范围内，如果在范围外，裁剪也是需要耗费资源的，出于性能的考虑，一般是禁用的。

Autoresize Subviews（子视图自动重置尺寸）：让系统自动调整子视图的大小以适应父视图的大小变化。

- Stretching（拉伸）：只有当在屏幕上调整矩形视图大小并且需要重新绘制该视图时，才需要拉伸。例如，希望每条边的 10% 可以拉伸，x 和 y 就分别为 0.1，width 和 height 都设为 0.8。

2. 容器视图

Container View（容器视图）是一个非常特殊的视图，它包含了一个视图和其对应的视图控制器，两者通过“Embed segue”相互连接，如图 C-18 所示。

正如其名字所言，容器视图将一个视图直接和一个视图控制器关联起来，通过视图控制器来对该视图的界面进行控制和操作，就如同子视图一样，是一个非常方便的视图元素。

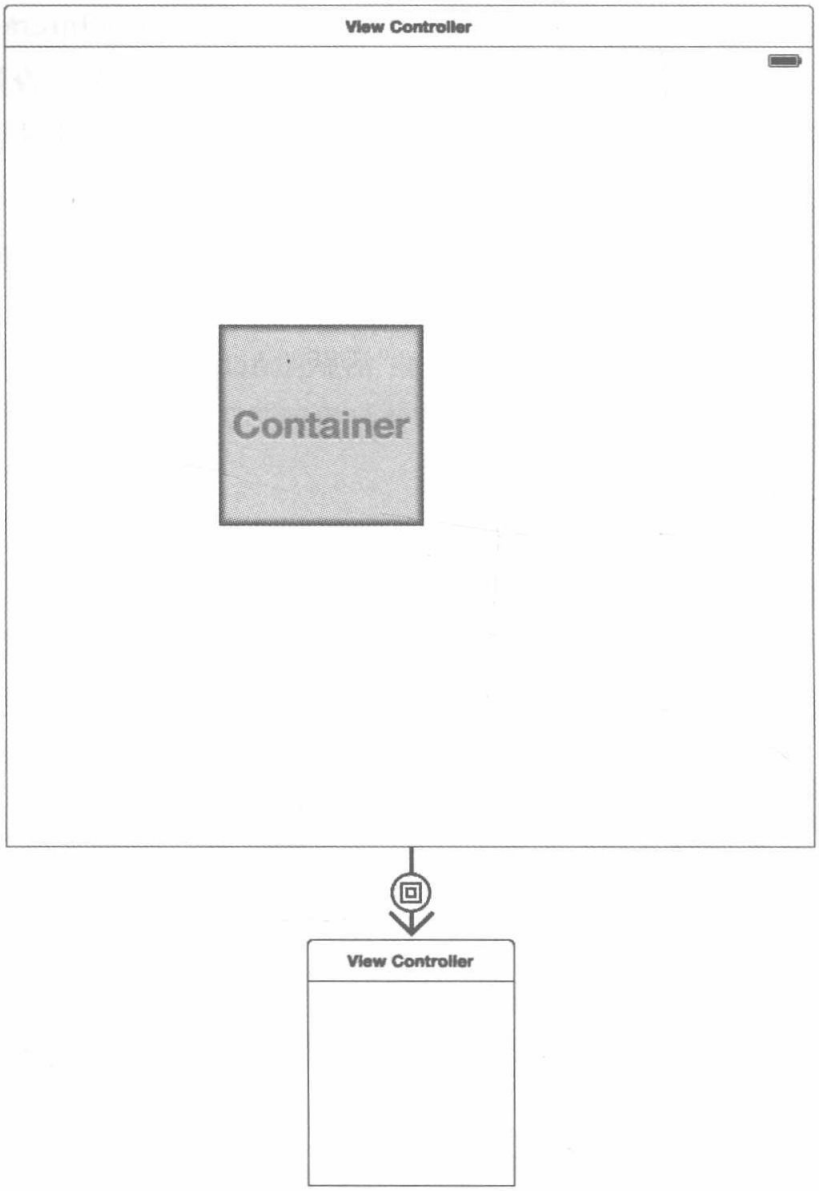


图 C-18　Container View

因此，在对容器视图进行编辑的时候，只需对其绑定的控制器进行操作即可。

3. 活动指示视图

活动指示视图（Activity Indicator View）是用来指示加载进度的一个视图，也就是俗称的“旋转的小菊花”，如图 C-19 所示。它的对应类是 UIActivityIndicatorView。

图 C-19　Activity Indicator View

对于活动指示视图来说，它一般和表视图配套使用，共同实现下拉刷新的加载功能。不过一般情况下只要出现需要等待加载的任务时，这个视图是经常使用的。

除了视图拥有的属性外，活动指示视图还拥有自己特有的属性：

- Style（风格）：在这里可以选择活动指示视图的风格，可以选择大白、白和默认的灰。
- Color（颜色）：这里可以选择进度指示的颜色。
- Behavior（行为）：Animating（动画）决定该活动指示视图是否启动动画，也就是开始“旋转”；Hides When Stopped（停止时隐藏）决定当动画停止后是否隐藏该视图。

4. 进度视图

进度视图（Progress View）是用来指示进度的一个视图，如图 C-20 所示，它的对应类是 UIProgressView。

图 C-20　Progress View

对于进度视图来说，它的最大用途就是用在视频、音乐播放的进度指示上面，当然对于下载进度等视图来说也是非常有用的。

除了视图拥有的属性外，进度视图还拥有自己特有的属性：

- Style（风格）：在这里可以设定进度视图的风格，可以选择默认 (Default) 和条状 (Bar)。所谓条状风格，也就是将进度视图的高度变高一些而已。
- Porgress（进度）：进度视图的核心所在，用来设定其当前的进度，取值在 0~1 之间。
- Progress Tint（进度填充颜色）：进度视图中已完成进度的颜色，也就是前面那部分的颜色。
- Track Tint（轨道填充颜色）：进度视图中未完成进度的颜色，也就是后面那部分的颜色。
- Progress Image & Track Image（填充图片）：进度视图可以接受图片来作为其填充。

5. 滚动视图

滚动视图（Scroll View）是一个能够左右、上下滚动的动态内容视图，也是能够响应最多用户手势的视图之一，它的对应类是 UIScroll View。

对于滚动视图来说，由于其动态特性，因此不适合在可视化设计中直接设计它的内容。它的出现目的就是为了显示多于一个屏幕的内容。

除了视图拥有的属性外，滚动视图还拥有自己特有的属性：

- Style（风格）：在这里可以设定滚动条的风格，可以选择默认（Default）、黑色（Black）和白色（White）
- Scroll Indicators（滚动条）：这个属性用来控制是否显示滚动条，分别是显示上下（Vertical）的滚动条以及显示左右（Horizontal）的滚动条。
- Scrolling（滚动属性）：这个属性用来决定滚动视图的行为。Scrolling Enabled（允许滚动）决定该滚动视图是否允许滚动行为，Paging Enabled（允许翻页）决定该滚动视图是否允许翻页行为，Direction Lock Enabled（允许锁定方向）决定该滚动视图是否

锁定方向，也就是只能够向一个方向滚动。

- Bounce（反弹效果）：当滚动视图滚动到边界的时候，再继续滚动会拥有反弹的效果。
- Zoom（放大）：这个属性用来决定滚动视图放大、缩小的倍数，都为 1 表示不允许放大、缩小。
- Touch（触控）：这个属性用来决定触控的行为属性。

Bounces Zoom（放大弹动）决定当进行放大缩小动作时，如果超过了最大和最小的允许值的话，是否显示反弹效果。

Delays Content Touches(延迟处理手势) 决定滚动视图是否要延迟处理对滚动视图中内容进行操作的手势，选择的话会有一定延迟，这样能够防止滚动手势和其他手势冲突。

Cancellable Content Touches（手势可取消）决定对滚动视图中的内容进行操作的手势，用户是否可以取消这个手势的效果。

- Keyboard（键盘）决定其什么时候消失，Dismiss on drag 是在进行滚动的时候会让键盘消失，Dismiss interactively 是用户在和滚动视图进行交互的时候让键盘消失。

6. 表视图

表视图控制器当中包含的就是表视图（Table View），关于表视图的作用，在此就不多加赘述了，它的对应类是 UITableView，它是 UIScrollView 的子类。

除了视图和滚动视图拥有的属性外，进度视图还拥有自己特有的属性：

- Content（内容）：这个属性决定了表视图中的单元格类型是“ Dynamic Prototypes（动态原型）”还是“ Static Cells（静态表格）”。动态表格的实现基本上依靠代码来完成，其样式和数目靠代码来确定，具备动态性；而静态表格则可以在界面生成器中实现，但是没有动态性。
- Prototype Cells（原型单元格数目）：这个属性是专门针对“动态原型”的，定义了动态原型样式中，有多少个单元格原型，每一个单元格原型都可以通过代码复用出一堆相同样式的单元格。
- Sections（节）：这个属性是专门针对“静态表格”的，一个表格由多个“节”组成，每个节又由多个“单元格”构成，这里是定义了静态表格样式中，有多少个节。
- Style（样式）：这里决定了表视图的显示样式，Plain(朴素) 样式是最简单的样式，每行数据都只是简单的用一条分隔线隔开，无论这个数据是否有值；Grouped(分组) 样式则可以根据设计将表视图分为许多组。
- Separator（分隔线）：这个属性决定了数据之间的分隔线的样式，分别是 None（无分割线）、Single Line（单线条）和 Single Line Etched（带浮雕效果的线条）。
- Separator Insets（分隔线间隔）：这个属性决定了分隔线与视图两边的距离，采用 Custom（自定义）模式可以自由设置分隔线距离视图两边的距离。

- Selection（选择）：这个属性决定了该表视图响应选择的配置，No Selection 表明这些单元格将不能被选中，Single Selection 表明一次仅有一个单元格被选中，Multiple Selection 表明可以有多个单元格被选中。
- Editing（编辑模式）：这个属性决定当表视图处于编辑状态时的选择模式，有三种模式：分别是不允许选中、允许选择一个和允许多项选择。此外，它还有一个属性 Show Selection on Touch（触摸时是否显示选择状态）。
- Index Row Limit（行数限制）：这个属性决定了当行数达到某个数值，则显示索引栏。
- Text 和 Background：这个属性决定了表视图中的相关颜色属性。

7. 表视图单元格

表视图中包含的就是表视图单元格（Table View Cell），单元格可以是静态的，也可以是动态的，一般其中都还包含了一个视图作为内容。在表视图上显示的每一条数据都是由单元格来承载的，它的对应类是 UITableViewCell。

除了视图拥有的属性外，表视图单元格还拥有自己特有的属性：

- Style（风格）：这个属性决定了单元格的显示样式。

Default	默认的样式，它提供了一个简单的左对齐文本标签和一个可选的图像视图，如果显示图像，那么图像将显示在左边。
Basic	最简单的样式，它只提供了一个简单的左对齐文本标签。
Right Detail	在 Basic 的基础上，提供一个简单的左对齐文本标签和一个较小的右对齐灰色文本标签。
Left Detail	在 Basic 的基础上，提供一个简单的居中显示的文本标签和灰色文本标签。
Subtitle	在 Default 的基础上，在左对齐文本标签的下方新增了一个副标题文本标签。

- Identifier（标识符）：每个单元格都需要拥有一个属于自己的标识符，用来标识自身；此外这个标识对于单元格重用也是非常有帮助的。
- Selection（选择）：这个属性用来确定当单元格被选中时的风格样式。
- Accessory（附件）：这个属性用来控制单元格右边附件的显示样式。附件是显示在单元格右边的一些标识图标，往往有箭头、详情框等多种样式。Disclosure Indicator 显示的是一个灰色的 > 箭头，Detail Disclosure 显示的是 > 箭头和①详情框，Checkmark 显示的是一个对勾，而 Detail 则只显示详情框。
- Editing Acc（编辑模式下的附件）：和上个属性相同，只不过是当单元格处于编辑模式下的显示样式。
- Indentation（文字缩进）：设定单元格的文字缩进大小，Level 指的是缩进层级，Width 指的是每次缩进的宽度。Indent While Editing（编辑模式下缩进）指的是在编辑模式

下，单元格仍然执行缩进指令；Shows Re-order Controls（显示重排序控制）用来显示重排序控制按钮，这样就可以拖动该单元格来完成重排序的操作。

- Separator（分隔符）：和表视图的分隔符属性一样。

8. 图像视图

图像视图（Image View）也是非常常用的一个视图，是存放图片的视图容器之一。它的对应类是 UIImageView。

除了视图拥有的属性之外，图像视图还拥有自己特有的属性：

- Image（图像）：设定图像视图在正常状态下的图像，一般项目中的图像文件和资源目录中的图像文件都会在里面显示出来。
- Highlighted（高亮状态）：设定图像视图在高亮状态下的图像。高亮状态一般是在图像被选中的时候被激活。

9. 集合视图

集合视图（Collection View）是非常常用的视图之一，在集合视图控制器中包含的就是这个集合视图，它的对应类是 UICollectionView，是 UIScrollView 的子类。

除了视图拥有的属性之外，集合视图还拥有自己特有的属性：

- Items（项目）：和表视图的“Prototype Cells”类似，规定了其要进行复用的单元格的数目。
- Layout（布局）：这个属性决定了集合视图单元格的布局样式，Flow 是系统默认的流动布局，而 Custom 则是可以选择自定义的布局样式。
- Scroll Direction（滚动方向）：这个属性决定了集合视图的滚动方向，这个属性会覆盖滚动视图的设定。
- Accessories（附件）：这个属性有两个项目，分别用来控制是否显示集合视图的 Header 顶部视图和 Footer 底部视图。

10. 集合视图单元格

集合视图单元格（Collection View Cell）和表视图单元格的作用非常类似，不过集合视图单元格并没有特别多的自定义属性。在集合视图上显示的每一项数据都由集合视图单元格来承载，它的对应类是 UICollectionViewCell。

除了视图拥有的属性之外，集合视图单元格还拥有自己特有的属性：

- Identifier（标识符）：每个单元格都需要拥有一个属于自己的标识符，用来标识自身；此外这个标识对于单元格重用也是非常有帮助的。

11. 集合重用视图

集合重用视图（Collection Reusable View）是一个可重用的视图，一般用作对集合视图的装饰和分隔作用。和单元格类似，它也是集合视图的一个元素之一，同样需要设定标识符。

只不过单元格只占有一个小正方形，而集合重用视图则占据了一整行视图。它的对应类是UICollectionReusableView。

12. 文本视图

文本视图（Text View）也是一个有用的视图。正如其名字所示，它的主要作用是提供富文本显示、编辑，就如同一个简单的文本编辑器。用户可以在这个视图内进行编辑操作，它的对应类是 UITextView。

除了视图拥有的属性之外，文本视图还拥有自己特有的属性：

- Text（文本）：这个属性是文本视图的重要属性，它上面初始显示的文本都在这里输入。这个属性可以选择两种样式，一个是 Plain（朴素）样式，另外一种是 Attributes（属性）样式，属性样式将提供更多的高级文本样式给开发者选择。
- Color（颜色）：这个属性决定了文本的颜色。
- Font（字体）：这个属性决定了字体的样式。
- Alignment（对齐）：这个属性决定了文本视图当中的文本对齐的模式，有左对齐、居中、右对齐、平均分布等模式。
- Behavior（行为）：这个属性有两个项目，Editable 决定该文本视图是否可供用户编辑，Selectable 决定该文本视图中的文本是否可供选择。
- Detection（检查）：这个属性将让 iOS 系统自动检测文本，从而将满足其中某个条件的文本以特定的形式显示出来。比如说超链接、电话号码、地址和事件等等。
- Capitalization（大写）：这个属性决定当文本视图中的文本为拉丁语系时，采用何种自动大写模式，可以选择每个字母都大写、每个单词首字母大写以及每句首单词首字母大写等属性。
- Correction（纠错）：这个属性决定系统是否会对文本中不符合语法的地方进行纠错。
- Spell Checking（拼写检查）：这个属性决定系统是否会对文本中拼写错误进行检查。
- Keyboard Type（键盘类型）：这个属性决定当用户编辑此文本视图中，弹出的键盘样式。
- Appearance（键盘风格）：这个属性决定键盘是采取深色风格还是浅色风格。
- Return Key（返回键）：这个属性决定键盘返回键的动作，这些动作都有其特定的含义。这个属性还有两个项目，Auto-enable Return Key（自动启用返回键）勾选后只有至少在文本框输入一个字符后键盘的返回键才有效；Secure Text Entry（安全文本输入）勾选后该文本视图中的文字将变为 * 号，即加密状态，这也会影响键盘的样式。

13. 标签

标签（Label）是最最常见，也是最最常用的控件，基本上没有哪一个应用会不使用标签的。标签的功能十分简单，将一段文字展示给用户，但是不像文本视图那样可供编辑，它的

对应类是 UILabel。

除了视图已有的属性外，标签还拥有自己特有的属性：

- Text（文本）：和文本视图的 Text 属性相同。
- Color（颜色）：和文本视图的 Color 属性相同。
- Font（字体）：和文本视图的 Font 属性相同。
- Alignment（对齐）：和文本视图的 Alignment 属性相同。
- Lines（分行）：决定了这个标签最大允许有多少行文字，设置为 0 则表明无限制。
- Behavior（行为）：这个属性有两个项目，Enable（可用）属性决定了该标签是否处于可用状态，如果不可用的话将不能对其进行修改；Highlighted（高亮）属性决定了该标签是否处于高亮状态。
- Baseline（基准线）：这个属性用来控制基准线的位置，当标签要根据内容来自适应大小的时候，基准线就是用来控制其对齐的位置。Align Baselines 是默认选项，文本最上端将和标签中线对齐；Align Center 是文本中线将和标签中线对齐；None 则是文本底端和标签中线对齐。
- Line Break（换行）：这个属性决定了当标签当中的文字过长时，标签的显示准则。

Clip（剪切）	剪切与文本宽度相同的内容长度，超出部分被删除。
Character Wrap（字符基准）	以字符为单位显示，超出部分省略不显示。
Word Wrap（单词基准）	以单词为单位显示，超出部分省略不显示。
Truncate Head（缩短首部）	前面部分文字以……的形式显示，只显示尾部文字内容。
Truncate Middle（缩短中部）	中间部分文字以……的形式显示，只显示首部和尾部部分文字内容。
Truncate Tail（缩短尾部）	后面部分文字以……的形式显示，只显示首部部分文字内容。

- Autoshrink（自动收缩）：设置标签是否允许自动收缩，有三种选项：

Fixed Font Size（固定字体大小）	如果标签宽度小于文字长度时，文字大小不自动缩放。
Minimum Font Scale（最小文字缩放比例）	设置最小收缩比例，如果标签宽度小于文字长度时，文字将自动进行收缩，收缩超过该比例后，停止收缩。
Minimum Font Size（最小字体大小）	设置最小收缩字体大小，如果标签宽度小于文字长度时，字体大小减小，低于该字体大小后，将不再减小。这一项在 iOS 6 之后就已不推荐使用了。

Tighten Letter Spcaing（收缩字符间距）	让字符之间的间隔变小，以便更好地进行自动收缩。

- Highlighted（高亮）：设置标签处于高亮状态时的文本颜色。
- Shadow（阴影）：设置标签阴影的颜色。
- Shadow Offset（阴影偏移量）：设置阴影的偏移量。

14. 视觉特效视图

视觉特效视图（Visual Effect View）是 iOS 8 提供的一个视图，用来作为图层蒙版，提供一个“毛玻璃”的效果。因此，要使用这个视图就必须要额外提供一个用来覆盖“毛玻璃”效果的图层。这里这个视图是带有 Blur（毛玻璃）效果的视觉特效视图，它的对应类是 UIVisualEffectView。

除了视图已有的属性外，带有毛玻璃效果的视觉特效视图还拥有自己特有的属性：

- Blur Style（毛玻璃样式）：设定毛玻璃的显示样式，light 表示毛玻璃视图的色相和背景视图的色相相同，dark 表示色相加深，extra light 表示色相变浅。通俗点来说，就是正常、变暗和变亮。
- Vibrancy（活力）：所谓的 Vibrancy 效果，是一种应用在毛玻璃样式上的特殊效果，它会在毛玻璃上留下一些特殊的空洞，让内容更生动。形象点来说，毛玻璃效果就好比雨天时候的带雾玻璃，而 Vibrancy 效果就是在这个带雾玻璃上用手画出的一道道痕迹。

15. Visual Effect Views with Blur and Vibrancy

和上面的“带有毛玻璃的视觉特效视图”类似，这个视图是“带有 Vibrancy 效果的视觉特效视图”。不过和“毛玻璃视图”相比，Vibrancy 视图不能够单独实现，它必须是“毛玻璃视图”的子视图，否则的话它就是一个普通的“毛玻璃视图”，就没有 Vibrancy 效果了。

16. 地图视图

地图视图（MapKit View）是专门对应苹果的地图服务 MapKit 的显示视图，也就是专门用来显示地图的视图，它自带有许多手势响应，能够根据用户手势来控制地图操作，无需用户实现。它的对应类是 MKMapView。

除了视图已有的属性外，地图视图还拥有自己特有的属性：

- Type（类型）：这个属性定义了地图的显示类型，有三种类型可供选择，Standard（标准）是普通地图，Satellite（卫星）是卫星地图，Hybrid() 是混合地图，揉和了标准和卫星地图组成的地图。
- Allows（准许）：这个属性决定了地图视图准许的行为。包括是否允许缩放手势（Zooming）、是否允许滑动手势（Scrolling）、是否允许旋转手势（Rotating）以及是否显示 3D 地图（3D View）。

- Shows（显示）：这个属性决定了地图视图能否显示哪些元素。包括用户位置（User Location）、建筑（Buildings）以及兴趣点（Points of Interest）。

17. GLKit 视图

GLKit 视图（GLKit View）是专门用来显示 OpenGL ES 框架绘制的元素的视图，其功能和作用在 GLKit 视图控制器中已经有所提及，它的对应类是 GLKVIew。

除了视图已有的属性外，OpenGL 视图还拥有自己特有的属性：

- Color Format（颜色编码）：OpenGL 中支持两种颜色编码方式，一个是 RGBA8888，另一个是 RGB565。这个编码方式的选择取决于 OpenGL 中使用的图片或者颜色的具体编码方式。
- Depth Format（深度缓冲）：深度缓冲是一个与你的渲染目标相同大小的缓冲，这个缓冲记录每个像素的深度。所谓深度，指的是某个目标前面覆盖有多少元素。一般深度缓冲越大，图像绘制、更新的速度越快，但是占用的内存也越高。
- Stencil Format（模板缓冲）：类似于深度缓冲，同样也是用来加快图像绘制、更新速度的缓冲区，用空间来换取时间。
- Multisample（多重采样）：也就是俗称的“抗锯齿”。
- Enable setNeedsDisplay（启用重绘）：这个选项将会在每次视图更新时，重新调用依次绘制方法。

18. 广告横幅视图

广告横幅视图（iAd BannerView）是专门针对苹果的广告服务 iAd 的显示视图，它可以将苹果服务器中的广告加载出来，展示给用户，并且给开发者带来收益。不过很遗憾，目前 iAd 并不支持大陆地区，它的对应类是 ADBannerView。

除了视图已有的属性外，广告横幅视图还拥有自己特有的属性：

- Type（类型）：这个属性决定了广告视图的类型，分别是横幅（Banner）类型和 Medium Rectangle（矩形弹出框）类型。

19. SceneKit View

SceneKit 视图是专门针对苹果的游戏框架 SceneKit 的显示视图，它对 3D 显示有着特有的处理方法，并且能够更好地显示 SceneKit 框架制作的游戏。它的对应类是 SCNView。

除了视图已有的属性外，SceneKit 视图还拥有自己特有的属性：

- Scene（场景）：这个属性用来将该视图和一个特定的场景关联起来，两者之间直接建立联系。
- Behavior（行为）：这个属性只有一个选项，那就是 Allow camera control，即是否允许摄像头控制，开发者可以借助摄像头的功能来实现对该视图的控制。

- Rendering（渲染）：这个属性用来控制该视图的渲染行为，包括是否允许多重采样（multisamping）、是否启动默认光源（default lighting）和是否启动抖动（jittering）。
- Animations（动画）：这个属性用来控制该视图中的所有动画行为，Plat 勾选上后将会在视图加载完毕后立刻播放所有动画，Loop 勾选上后该视图上的所有动画将会持续播放。

20. 网页视图

网页视图（Web View）是专门用来访问网页的现实视图，它提供解析 URL、加载 HTML 界面等多种浏览器都提供的功能，对应类是 UIWebView。

除了视图已有的属性外，网页视图还拥有自己特有的属性：

- Scaling（缩放比例）：这个属性决定了是否需要自适应网页上的内容。
- Detection（检查）：这个属性和文本视图的 Detection 属性相同。
- Options（选项）：这个属性设定了一系列选项，用来控制网页视图的行为：

Allows Inline Playback（允许内置播放）	这个属性决定了用内嵌 HTML5 播放视频还是用本地的全屏控制。
Playback Requires User Action（需要用户操作才播放）	这个属性决定了是需要用户点击才播放视频，还是加载完毕立即播放。
Playback Allows AirPlay（允许 AirPlay 播放）	这个属性决定了视频是否可以在 AirPlay 的相关设备上进行播放。
Suppress Incremental Rendering（阻止继续绘制）	这个属性决定了当网页视图完全加载进内存中时，是否停止继续绘制视图。

- Pagination（分页）：这个属性用来控制网页视图上的分页效果。

21. 选择器视图

选择器视图（Picker View）是专门供用户进行选择的视图，它提供一系列开发者自定义的值，用户通过上下滚动选择其期望的值，从而完成选择，对应类是 UIPickerView，如图 C-21 所示。

图 C-21　Picker View

除了视图已有的属性外，选择器视图还拥有自己特有的属性：

- Behavior（行为）：这个属性决定了是否显示指示线，所谓的指示线指的是用户在进行选择的时候。

22. 导航栏

导航栏（Navigation Bar）是使用导航视图应用最主要的视图，通过它可以完成视图之间跳转的控制，当前页面的显示等多种常见的功能，一般情况下使用了导航控制器的界面就包

含了这个导航栏视图，它的对应类是 UINavigationBar。

除了视图已有的属性外，导航栏视图还拥有自己特有的选项：

- Style（风格）：这个属性决定了导航栏的风格，可以选择默认的白色风格，也可以选择黑色风格，而黑色半透明风格目前已经不赞成使用。此外，这个属性下方还有一个 Translucent（半透明）的选项，这个选项控制导航栏是否需要半透明。
- Bar Tint（栏目着色）：这个属性决定了导航栏上返回键、标题以及右键的默认着色。
- Shadow Image（阴影图片）：这个属性决定了导航栏的阴影样式，可以用图片来实现阴影。
- Back Image（返回键图片）：这个属性决定了导航栏返回键的样式，可以用图片来实现返回键。
- Back Mask（返回键遮罩图片）：这个属性配合返回键使用，产生遮罩的效果。
- Title（标题）：这个属性设定了导航栏中间标题的相关配置，包括字体、颜色、阴影以及偏移量的设置。

23. 工具栏

工具栏（Toolbar）是一个不常用的视图，它上面可以提供一系列的按钮，用户可以通过这些按钮完成一些特定的操作，和电脑上的工具栏类似。它的对应类是 UIToolBar。

除了视图已有的属性外，工具栏视图还拥有自己特有的选项，不过这些选项和导航栏视图的十分类似。

24. 标签栏

标签栏（Tab Bar）是一个比较常用的视图，它提供了一系列选项卡，可供用户进行视图的切换。选项卡控制器中就包含有这个标签栏视图，它的对应类是 UITabBar，如图 C-22 所示。

除了视图已有的属性外，标签栏视图还拥有自己特有的选项，部分选项和导航栏视图的类似。

- Background（背景）：设置标签栏的背景图片。
- Selection（选择）：设置标签栏被选中时显示的效果图片。
- Item Positioning（项目位置）：设置标签栏中项目的位置，是自动，还是充满，也可以是居中。

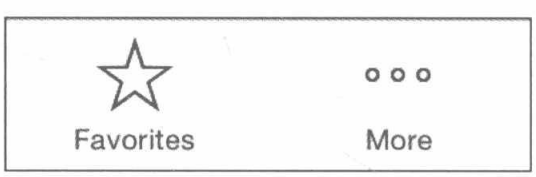


图 C-22 Tab Bar

25. 搜索栏

搜索栏（Search Bar）也是一个非常常用的视图，通过它可以完成一些搜索工作。搜索栏的对应类是 UISearchBar，如图 C-23 所示。

图 C-23 Search Bar

除了视图已有的属性外，搜索栏视图还拥有自己特有的选项，部分选项和上述视图的类似。

- Text（文本）：这个属性是用户实际搜索的内容。
- Placeholder（占位符）：这个属性定义了搜索栏中的提示信息，提示用户可以搜索哪些内容。
- Prompt（提示）：这个属性定义了搜索栏顶部的提示信息，引导用户如何进行搜索。
- Search Style（搜索栏样式）：这个属性决定了搜索栏的样式，有系统默认的白色样式，突出（Prominent）样式以及半透明（Minimal）样式。
- Scope Bar（类别栏）：设置搜索栏中的类别栏的样式图片。
- Search Text（搜索文本）：这个属性设置了搜索文本的位置信息，可以自己添加偏移量。
- Background（背景）：这个属性设置了背景图片的位置信息，可以自己添加偏移量。
- Options（选项）：这个属性设置了搜索栏的相关配置。Shows Search Results Button 选项将可以显示搜索栏的搜索结果按钮，点击这个按钮可以打开搜索结果；Shows Bookmarks Button 选项将可以显示搜索栏的书签按钮，这个书签按钮将在有搜索书签记录的时候显示出来，从而跳转到书签中记录的搜索条件；Shows Cancel Button 选项将可以显示搜索栏的取消按钮，通过这个按钮将可以取消搜索；Shows Scope Bar 选项将可以显示搜索栏的类别栏，从而控制搜索的类别。
- Scope Title（类别文本）：这个属性决定了类别栏的文本。

C.4.3　控件

实际上，控件按理来说应该属于“视图”这一部分，因为它们本质上也是继承自 UIView，拥有视图层的功能。但是在本书中，“控件”的定义是用户能够通过这些元素，来达成一定的控制目的的视图。换句话说，控件的主要目的不是展示内容，而是帮助用户来进行控制。

控件和视图在属性控制器的区别在于，除了视图特有的属性之外，还拥有控件特有的属性。

- Alignment（对齐）：这个属性决定了控件中内容在水平（Horizontal）方向和垂直（Vertical）方向上的对齐方式。
- Content(内容)：这个属性决定了空间的显示样式。Selected 选中控件将一直处于“选中”状态，Enabled 选中控件将一直处于“可用”状态，Highlighted 选中控件将一直处于“高亮”状态。

下面我们来介绍这些控件的大致功能和检查器属性。

1. 按钮

按钮（Button）控件的作用无需多说，它也是用的非常多的控件之一，它的对应类是 UIButton，如图 C-24 所示。

图 C-24　Button

除了视图和控件已有的外，按钮控件还拥有自己特有的属性：

- Type（类型）：这个属性决定了按钮的默认外观，有多种系统默认的样式可供选择，还可以自定义按钮的外观。
- State Config（状态配置）：这个属性将会选中的某个状态，开发者可以对这些状态的标题、字体、颜色、图片等多个属性进行配置。状态之间互不影响，可以分别对这几种状态进行配置。目前按钮有的状态有：默认、高亮、选中和不可用四种状态。
- Title（标题）：这个属性决定了按钮上显示的文本。
- Font（字体）：这个属性决定了文本的字体。
- Text Color（文本颜色）：这个属性决定了文本的颜色。
- Shadow Color（阴影颜色）：这个属性决定了按钮阴影的颜色。
- Image（图片）：这个属性可以让按钮以图片的样式显示。
- Background（背景颜色）：这个属性决定了按钮的背景颜色。
- Shadow Offset（阴影偏移量）：这个属性决定了按钮阴影的偏移量
- Drawing（绘图）：这个属性决定了一系列视图绘制时的配置。

  Shows Touch On Highlight　当按钮被按下时，是否显示高亮状态。
  Highlighted Adjusts Image　当按钮处于高亮状态时，是否改变其覆盖的图片。
  Disabled Adjusts Image　当按钮处于不可用状态时，是否改变其覆盖的图片。
- Line Break（换行）：这个属性等同于标签视图的对应属性。
- Edge（边界）：设置按钮的某个元素要设置间隔距离，这个属性和下面的 Inset 属性共同使用，Inset 可以设定选择内容、文本和图像，这样 Inset 就可以分别对它们的间隔距离进行设定。
- Inset（间隔）：设定按钮内容与视图边缘的间隔距离。

2. 分段控件

分段控件（Segmented Control）一般用在多视图的控制，用来选择不同的视图，就如同“QQ”消息界面最顶部的“消息”和“电话”这两个分段，它们就是一个分段控件，对应类是 UISegmentedControl，如图 C-25 所示。

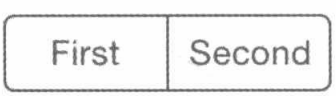


图 C-25　Segmented Control

分段控件并不会和任何视图进行绑定，因此开发者需要先判断当前分段控件处于何种状态，然后再对其做出相应的反应，切换不同的视图。

除了视图和控件已有的属性外，分段控件还拥有自己特有的属性：

- Style（风格）：这个属性决定了分段控件的风格。iOS 7 之后，这些风格在视觉效果上似乎没有任何区别。
- State（状态）：Momentary（瞬间）设置在点击后是否恢复原样，如果勾选的话，分段

控件将不会保持某个段一直被选中的状态。

- Segments（分段）：设定该分段控件拥有多少个分段。
- Segment（当前段）：选择具体要进行设置的某个段，段和段之间的设置互不影响。
- 其余属性：其余属性的作用都十分明显，并且在前面的控件或视图处已经有所介绍，在此就不再赘述了。

3. 文本框

文本框（Text Field）是最常用的控件之一，一般用于接收用户的输入，比如说登录框、密码框等等，它的对应类是 UITextField，如图 C-26 所示。

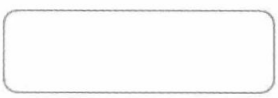


图 C-26　Text Field

除了视图和控件已有的属性外，文本框还拥有自己特有的属性（某些通用属性将不予介绍）：

- Placeholder（占位符）：当文本框中文本为空时，以一个灰色文本显示的提示文本，当用户开始输入后，这个占位符将被消除。
- Border Style（边框样式）：这个选项用来决定文本框的边框样式，可以选择无边框、矩形边框、阴影边框和圆角矩形边框。
- Clear Button（清除按钮）：这个选项用来控制清除按钮出现的行为，清除按钮是当文本框有文字的时候，出现在文本框右侧的一个小 X 按钮，用来清空文本框的所有文字。这里可以选择永不出现、编辑时一直出现、编辑时才出现以及一直出现。下方还有一个选项，决定是否在用户编辑时清除文本，也就是当用户点击文本框后就立即删除所有文本。
- Min Font Size（最小的字体大小）：这个选项设置当文本框的文本显示出来时候的最小字体大小，在下方可以选择是否让字体自适应文本框的大小。

4. 滑动条

滑动条（Slider）是用来控制“进度”的重要控件，通过其可以对有一定范围的进度进行调整，比如说音量、播放进度等等选项，它的对应类是 UISlider，如图 C-27 所示。

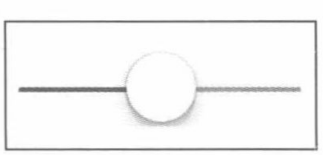


图 C-27　Slider

除了视图和控件已有的属性外，滑动条还拥有自己特有的属性：

- Value（值）：这个属性是滑动条的主要属性，用来控制滑动条的进度。Current 为当前进度，也就是滑块当前所在的位置。Minimum 和 Maximum 分别是滑动条允许的最小和最大值，滑块只能在这两个值之间滑动。这几个值只允许 0 ～ 1。
- Image（滑动条图片）：这个属性决定了滑动条的图片样式，Min Image 表示左侧滑条的图片样式，为空表示不使用图片。Max Image 和它相反，是决定滑动条右侧滑条的图片样式。

- Track Tint（滑动条颜色）：这个属性决定了滑动条滑条的颜色，Min Track Tint 决定左侧滑条的图片样式，Max Track Tint 和它相反，是决定滑动条右侧滑条的图片样式。Thumb Tint（滑块颜色）决定了中间滑块的颜色。
- Continuous Updates（连续变化）：这个属性决定了当用户在拖动滑动条的时候，它的值是不是实时更新，否则的话只有当用户放开滑动条，其值才会更新。

5. 开关

开关（Switch）是用来控制“真 / 假”值的重要控件，主要用来控制开启 / 关闭某项功能等，它的对应类是 UISwitch，如图 C-28 所示。

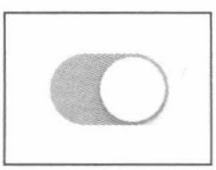


图 C-28 Switch

除了视图和控件已有的属性外，开关还拥有自己特有的属性：

- State（状态）：开关的状态，有 On（开）和 Off（关）两种。
- Tint（着色）：这个属性决定了开关的颜色，On Tint 表示的是开关处于“开”状态时的背景色，如果开关处于“关”状态那么背景则是透明的。Thumb Tint 决定了中间滑块的颜色。
- Image（图片）：这个属性决定了开关的图片，和颜色类似，只不过使用图片代替了颜色进行显示。

6. 页面控件

页面控件（Page Control）由一系列小圆点组成，用来标识一个可以滚动页面的滚动状态，包括有多少页以及当前视图处于哪一页等等，同时也可以通过这个页面控制控件来直接定位到相应的页面上去，它的对应类是 UIPageControl，如图 C-29 所示。

图 C-29 Page Control

除了视图和控件已有的属性外，页面控件还拥有自己特有的属性：

- Pages（页面数）：这个属性用来决定页面控制控件有多少个圆点，也就是对应多少个页面，并且还可以选择当前页面控制控件所在的页面。
- Behavior（行为）：这个属性用来定义页面控制控件的某些行为，Hides for Single Page 指的是当其只对应一个页面的时候，自动隐藏该控件；Defers Page Display 指的是是否推迟页面的显示，让页面控制控件变化完毕后，再进行页面的变化。
- Color（颜色）：这个属性定义了页面控制控件的颜色，Current Page 定义了当前圆点的颜色，Tint Color 定义了除当前圆点之外的其他圆点颜色。

7. 步进控件

步进控件（Stepper）由两个按钮组成，分别用来控制某项内容的增加或者减少，就如同遥控板上的音量控制按钮一样，它的对应类是 UIStepper，如图 C-30 所示。

图 C-30 Stepper

除了视图和控件已有的属性外，步进控件还拥有自己特有的属性：

- Value（值）：用来决定步进控件的相关值，Minimum 决定了步进控件允许出现的最小值，Maximum 决定了控件允许出现的最大值，Current 决定了控件的当前值，Step 决定了每一次增加 / 减少步进控件所增加 / 减少的值。
- Behavior（行为）：这个属性决定了步进控件的相关行为特性。Autorepeat 指的是当用户按住加号或者减号不松手，步进控件的数值会持续变化；Continuous 指的是当用户点击步进控件时是否立即响应，否则就要等用户放开控件才会响应；Wrap 指的是当步进器达到最大值或者最小值时，重新从最小值或者最大值开始计算。

8. 日期选择器

日期选择器（Date Picker）控件常常出现在某些要选择日期的功能当中，通过这个控件用户可以轻松地选择时间，而不用自己键入，它的对应类是 UIDatePicker。

除了视图和控件已有的属性外，日期选择器还拥有自己特有的属性：

- Mode（模式）：这个属性决定了日期选择器当中可以供用户进行选择的日期属性，可以只选择时间、日期，或者两者都选择，还可以实现计时器的选择。
- Locale（地理位置）：这个属性决定了日期选择器的地区，以及其显示语言。
- Interval（分钟区间）：这个属性将分钟表盘设置为以不同的时间间隔来显示分钟。
- Date（日期）：这个属性设置了日期选择器打开的时候，默认的日期是使用当前日期还是用户自定义的日期。
- Constraints（常量）：这个属性可以设定日期选择器的最小允许选择的时间以及最大允许选择的时间。
- Timer（计时器）：这个属性设置了当日期选择器位于计时器模式时，倒计时的时长。

C.4.4　手势

手势（Gesture）指的是用户在 iOS 系统上的各种手势动作，这也是 iOS 的精华所在，即通过手势来控制一切。使用好这些手势可以给用户带来不少的便利。

手势拥有自己特有的属性：

- State（状态）：用 Enabled 属性标记该手势是否可用，是否可以对用户的手势进行响应。
- Behavior（行为）：这个属性决定了手势的一些表现行为。Cancels touches in view 指的是用户可以通过改用其他手势或者将手势移出屏幕的方式来取消手势操作；Delays touches began 指的是延迟响应用户的触摸动作；Delays touches ended 指的是当用户的触摸动作结束后，是否延迟响应。

1. 点击

点击手势识别器（Tap Gesture Recognizer）是最简单的手势，通过它可以轻松实现单击、双击、三击等等点击手势的响应，它的对应类是 UITapGestureRecognizer。

除了手势已有的属性外，点击手势识别器还拥有自己特有的属性：

- Recognize（识别）：这个属性决定了点击手势识别器的识别状态，Taps 是指需要多少次点击才能激活这个识别器，Touches 指的是需要多少根手指才能够响应。

2. 缩放

缩放手势识别器（Pinch Gesture Recognizer）一般用在图片的放大和缩小上，手势动作是两根手指互相靠近或者分开，从而达到缩放的效果，它的对应类 UIPinchGestureRecognizer。

除了手势已有的属性外，缩放手势识别器还拥有自己特有的属性：

- Scale（缩放）：这个属性决定了缩放时的缩放比率，缩放比率越大，手势移动时造成的缩放效果也越明显。因为缩放效果的位移比率跟这个缩放比率成正比关系。

3. 旋转

旋转手势识别器（Rotation Gesture Recognizer）一般用在某个元素的旋转上，手势动作是两根手指成圆形滑动，从而达到旋转的效果，它的对应类是 UIRotationGestureRecognizer。

除了手势已有的属性外，旋转手势识别器还拥有自己特有的属性：

- Rotation（旋转）：用来设置旋转的方向和旋转的弧度，旋转的弧度是从手指的初始位置到最终位置决定的。

4. 轻扫

轻扫手势识别器（Swipe Gesture Recognizer）一般用于视图的切入和切出，通过用手指在界面上向左或向右轻滑，从而让视图进入到下一个页面或者回到上一个页面，它的对应类是 UISwipeGustureRecognizer。

除了手势已有的属性外，轻扫手势识别器还拥有自己特有的属性：

- Swipe（轻扫方向）：这个属性决定了这个识别器能够识别的用户执行轻扫动作的方向，有四种方向可供选择。
- Touches（触碰数目）：这个属性决定了识别器需要多少个手指执行轻扫动作才能够被触发。

5. 拖动

拖动手势识别器（Pan Gesture Recognizer）一般用于对视图中某个元素的拖动，将某个元素从某个位置移动到另外一个位置。它的对应类是 UIPanGestureRecognizer。

除了手势已有的属性外，拖动手势识别器还拥有自己特有的属性：

- Touches（触碰数目）：和轻扫手势类似，只不过这个手势可以决定岂能够接收的最小

和最大手指数目。

6. 屏幕边缘拖动

屏幕边缘拖动手势识别器（Screen Edge Pan Gesture Recognizer）继承自拖动手势识别器，一般用在屏幕边缘，用来切换视图，它的对应类是 UIScreenEdgePanGestureRecognizer。

除了手势和拖动手势识别器已有的属性外，屏幕边缘拖动手势识别器还拥有自己特有的属性：

- Edges（边缘）：决定这个识别器能够响应用户从哪个边缘发起的手势，从而做出相应的回应。

7. 长按

长按手势识别器（Long Press Gesture Recognizer）一般用于长按弹出对话框等操作，用户通过长时间按压屏幕，触发该识别器，它的对应类是 UILongPressGestureRecognizer。

除了手势已有的属性外，长按手势识别器还拥有自己特有的属性：

- Min Duration（最小时段）：指的是用户长按多长时间才能够触发该识别器。
- Recognize（识别）：和单击手势识别器类似。
- Tolerance（限度）：指的是该识别器最大能够允许的手指数目。

C.4.5　栏目项

所谓项目项（BarItem），是一个单独独立出来的 iOS 设计元素，它们和导航栏、工具栏、选项卡等栏目联系紧密，是这些栏目当中的项目单元，完成类似于“按钮”的工作，对应类是 UIBarItem。

栏目项拥有自己特有的属性：

- Title（标题）：这个属性决定了栏目项的标题。
- Image（图片）：这个属性决定了栏目项的显示图片。
- Tag（标签）：这个属性决定了当前栏目项的标签，通过这个标签可以区分某个栏目中的栏目项。

1. 导航

导航项（Navigation Item）是存在于导航栏当中的，必须要结合导航栏才能使用。它完成“按钮”的工作，响应用户的操作，然后执行相关功能，它的对应类是 UINavigationItem。

导航项拥有自己特有的属性：

- Title（标题）：这个属性决定了该导航项上的标题文本。
- Prompt（提示）：这个属性决定了该导航项上的提示文本，提示文本一般位于标题的上方。

- Back Button（返回按钮）：这个属性决定了返回按钮的文本信息。

2. 栏按钮

栏按钮项（Bar Button Item）一般结合工具栏使用，但是导航栏也可以使用这个按钮项，它的对应类是 UIBarButtonItem，继承自栏目项。

除了栏目项已有的属性外，栏按钮项还拥有自己特有的属性：

- Style（风格）：设定栏按钮项的风格，可以是边框（Bordered）类型，也可以是 Done（完成）类型。一般来说，在扁平化风格当中，这几种风格是看不出来什么太大的区别的，Done 风格也就是按钮保持一直按下去的状态风格。Plain 风格在目前已经不被支持。
- Identifier（标识符）：这个属性设置栏按钮项的外观标识，有许多系统自带的外观标识，包括垃圾桶、书签等图片样式，因此可以自由选择需要的样式，甚至可以实现自定义。

3. 选项卡

选项卡栏目项（Tab Bar Item）是存在于选项卡视图当中的，必须结合选项卡视图才能使用。它的对应类是 UITabBarItem。

除了栏目项已有的属性外，选项卡栏目项还拥有自己特有的属性：

- Badge（标记）：选项卡右上角出现的一个红色标记文本，一般可以用其来标注一些特殊的状态，比如说 NEW 或者消息数目等等。
- System Item（系统项）：这个属性可以设置选项卡的外观，可以选择系统自带的外观标识，还可以自定义实现。
- Selected Image（选中图像）：这个属性决定了当当前选项卡被选中时，它的显示图像。

4. Fixed Space Bar Button Item（固定占位符栏按钮）

一般和栏按钮项结合使用，作为占位符来使用，是一个用来分隔栏按钮项的占位符，它的占位大小是固定的。

5. Flexible Space Bar Button Item（不固定占位符栏按钮）

和上面类似，只不过它的占位大小是不固定的，根据屏幕的宽度而变化。

C.4.6 游戏元素

SpriteKit 有一个专门用来建立游戏场景的文件 sks，全名就是 SpirteKit Scene，它起到的作用是类似于故事板的作用，开发者可以将游戏元素像故事板上的视图一样放置上去，从而达成可视化设计的目的。

1. 颜色精灵

精灵（Sprite）是游戏里面的角色，比如敌人，游戏里面运动的物体等等，它可以用一张图片或者一张图片的一块矩形部分来定义，如图 C-31 所示。

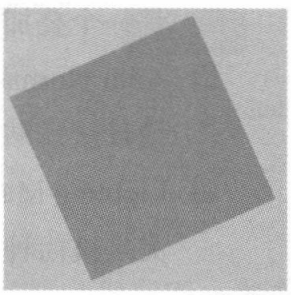


图 C-31　Color Sprite

那么颜色精灵（Color Sprite）元素的概念就十分简单了，它是一个带有颜色的矩形视图。开发者可以任意变换其大小、角度，从而达成设计，它的对应类是 SKSpriteNode。

精灵拥有自己特有的属性：

- Name（名字）：当前精灵元素的名称，开发者可以通过这个名称在代码中对 Sprite 进行控制。
- Parent（父级视图）：设定精灵所依附的父级视图，精灵可以依附到精灵当中。对父级视图进行重绘、移动等操作都会影响到其中的子类对象。
- Texture（纹理）：设定当前精灵的纹理，纹理会被处理并绘制到该精灵上。注意，如果想让一个图片表现为一个精灵的话，那么就需要使用纹理来添加图片。
- Position（位置）：设定当前精灵在父级视图的位置。
- Size（尺寸）：设定当前精灵的尺寸大小。
- Anchor Point（锚点）：设定当前精灵的锚点，旋转、平移等多种变换操作都是基于锚点来进行的。默认锚点是精灵的中心。
- Color（颜色）：设定当前精灵的颜色，如果精灵没有任何纹理的话，设定该选项就是设定矩形部分的颜色。
- Blend factor（混合因子）：这个选项也是用来设定精灵颜色的，通过它可以实现颜色渐变的效果。
- Blead Mode（混合模式）：和绝大多数图像处理的混合模式一样，这个属性决定了当出现多个颜色图层叠加时，精灵对实际显现出来的颜色处理方式。一般情况下，混合模式有以下选择：

Alpha	两个显示对象的透明通道互相叠加，且都能正常显示。
Add	底层对象的颜色值加上上层对象的颜色值，混合成新图像。
Subtract	底层对象的颜色值减去上层对象的颜色值，混合成新图像。
Multiply	加深颜色，两个显示对象的色彩相乘，混合成新图像。
MultiplyX2	加深颜色，两个显示对象的色彩相乘再乘以 2，混合成新图像。
Screen	产生漂白效果，两个显示对象的反色相乘，混合成新图像。
Replace	上层对象的颜色直接替换掉底层对象的颜色。

- IK Constrains（反向运动约束）：IK 是 Inverse Kinematics 的缩写，意思是反向运动。通过这个约束可以实现踩踏关键帧的效果，也就是动画中常说的“定脚”效果。

- Z（Z 轴）：决定了精灵的旋转角度和 Z 轴位置等属性。
- Scale（比例）：决定了精灵的缩放比例。
- Normal Map（法线贴图）：法线贴图可以用来合成带有质感的光照纹理效果，可以说是一个叠加的贴图图层。
- Auto Normal Map（自动法线贴图）：和法线贴图的作用类似，只不过无需开发者自己设计法线贴图，它会根据一系列算法运算，自动生成一个最佳的法线贴图。
- Lighting Mask（光线遮罩）：光线遮罩可以为精灵添加一层光照遮罩模板，决定了该精灵是被何种光照亮。
- Shadow Cast Mask（阴影投射遮罩）：阴影投射遮罩可以为精灵增加阴影效果，决定了该精灵要遮挡何种光线，并会产生阴影。
- Shadowed Mask（阴影遮罩）：阴影遮罩可以为精灵表面增加阴影效果，决定了该精灵被何种光产生的阴影覆盖。
- Body Type（实体类型）：这个属性决定了这个精灵实体类型，可以选择无实体、矩形实体、圆形实体和描边实体。每个实体都有各自的具体属性设置。

2. 空节点（Empty Node）

节点（Node）是 SpriteKit 的基本元素，就如同 NSObject 对于 Cocoa。SpriteKit 的大多数元素都继承自节点，它的对应类是 SKNode，如图 C-32 所示。

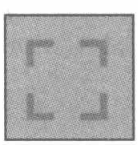


图 C-32　Empty Node

节点拥有一些基本的属性，某些在上面的精灵中已经有所提及：

- X Scale（X 轴比例）：这个属性决定了节点 X 轴方向的放大比例。
- Y Scale（Y 轴比例）：这个属性决定了节点 Y 轴方向的放大比例。
- Rotation（旋转角度）：这个属性决定了节点当前的旋转角度。
- Alpha（透明度）：这个属性决定了节点当前的透明度，Hidden 表示该节点是否可视。

3. 光线

光线（Light）是 SpriteKit 的一个重要元素，通过它可以给游戏场景创建一个光源，从而 SpriteKit 会自动给场景中的元素添加这个光源效果，从而渲染出合适的环境，它的对应类是 SKLightNode，如图 C-33 所示。

图 C-33　Light

光线除了一些已经出现的属性外，还拥有自己特有的属性：

- Shadow Color（阴影颜色）：这个属性决定了当前光线所产生的阴影颜色。
- Ambient Color（环境颜色）：这个环境颜色会无视自身的透明度和其他相关属性，分分钟照亮全场。
- Falloff（衰减比率）：这个属性决定了光源强度的衰减比率，比率越大则光线照明的地方越少。

4. 发射器

发射器（Emitter）是一个很常用的元素，通过它可以创建诸如火焰、闪电等炫目的特效，它的对应类是 SKEmitterNode，如图 C-34 所示。

图 C-34　Emitter

粒子发射器除了一些已经出现的属性外，还拥有自己特有的属性：

- Target Node（目的节点）：总要有一个节点作为“粒子发射”的基点，也就是设定哪一个节点发射这些粒子。
- Particle Texture（粒子纹理）：这个属性确定了要发射粒子的纹理效果。
- Particles（粒子）：这个属性设置了粒子产生的比率以及粒子的最大数目。
- Lifetime（生命周期）：这个属性设置了粒子产生的起始时间和能够移动的范围，在这个范围内粒子会逐渐缩减。
- Position Range（位置范围）：这个属性决定了发射例子的起始位置。
- Angle（角度）：这个属性决定了粒子发射的角度范围。
- Speed（速度）：这个属性决定了粒子的发射速度。
- Acceleration（加速度）：这个属性决定了粒子持续受到的加速度影响。
- Color Blend（颜色混合）：这个属性决定了粒子颜色混合的因子、范围和速率。
- Color Ramp（颜色带）：这个属性可以制作渐变的粒子效果。

5. 标签

标签（Label）是 SpriteKit 中特用的标签文本，它的对应类是 SKLabelNode。

标签除了一些已经出现过的属性外，以及一些 UILabel 上特有的属性外，还拥有自己特有的属性：

- Horizontal Alignment（水平对齐）：这个选项决定了标签上的文字的水平对齐方式。
- Vertical Alignment（垂直对齐）：这个选项决定了标签上的文字的垂直对齐方式。

6. 形状节点

形状节点（Shape Node）用于渲染基于 Core Graphics 路径的形状，它的对应类是 SKShapeNode，如图 C-35 所示。

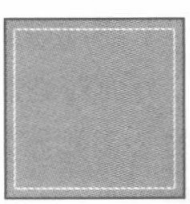


图 C-35　Shape Node

形状节点除了一些已经出现过的属性外，还拥有自己特有的属性：

- Line Width（粗细度）：这个属性决定了绘制的路径线的粗细度。
- Glow Width（发光半径）：这个属性决定了绘制路径的发光半径。
- Antialiased（抗锯齿）：这个属性决定是否开启抗锯齿功能。

7. 场

场是一种带有“力”效果的区域，节点在场里面会受到“力”的作用和影响，比如说重力场等等，它的对应类是 SKFieldNode，如图 C-36 所示。

目前一共有 8 个重力场模板提供给开发者，它们分别是：

图 C-36 Field

- Linear Gravity Field（线性重力场）
- Radial Gravity Field（放射重力场）
- Spring Field（弹簧场）
- Drag Field（阻力场）
- Vortex Field（涡流场）
- Turbulence Field（紊乱场）
- Noise Field（噪声场）
- Velocity Field（速度场）

除了一些已经出现过的属性外，场还拥有一些特有的属性：

- Strength（场强）：这个属性决定了场中“力”的作用强度，场强越大，节点受到的影响越大。
- Falloff（衰减）：这个属性决定了场的衰减程度，用来计算某个距离的作用力大小，从最小半径开始计算。
- Min Radius（最小半径）：这个属性决定了场的最小影响半径。
- Category Mask（类别掩码）：这个属性决定了场的类别掩码，通过该掩码可以判断节点目前处于哪个场内。
- Direction（方向）：阻力场特有的属性，用来决定阻力的方向，节点在这个阻力相反的方向上运动会得到加速。
- Velocity（速度）：速度场特有的属性，用来决定当前场内允许的最高速度。

C.4.7 Apple Watch 元素

Apple Watch 是 2015 年才上市的一个新的人机交互设备，其以手表的形式给用户带来了更有趣、不一样的交互体验。Apple Watch 的界面推荐采用 Storyboard 来设计，因此熟悉其故事板中的元素是非常重要的一步。

1. 界面控制器

界面控制器（Interface Controller）对于 Apple Wathc 来说是非常重要的控制器，其应用内的大多数界面都由这个控制器来控制。它可以完成获取数据、设置控件、显示界面对象等多种功能，类似于 iOS 的 UIViewController，它的对应类是 WKInterfaceController，如图 C-37 所示。

界面控制器包含有以下几种属性可供设置：

- Identifier（标识符）：这个属性决定了这个控制器在故事板文件中的身份标识，开发者可以通过此标识符定位到该控制器，从而对其做出控制。

- Title（标题）：这个属性设置了当前控制器的标题
- is Initial Controller（是否是初始控制器）：Watch 应用加载完毕后，首先启动的是哪一个控制器。
- Hides Whtn Loading（加载时隐藏）：这个选项决定了当应用加载时是否显示该控制器，否则的话应用会首先加载该控制器。
- Background（背景图片）：设置该控制器的背景图片
- Mode（显示模式）：和 iOS 视图的显示模式类似。
- Animate（动画）：这个属性决定了是否在其被加载后自动播放动画。
- Color（颜色）：这个属性决定了背景颜色。
- Insets（间隔）：这个属性决定了元素的间隔。
- Spacing（间距）：这个属性决定了界面控制器之间的间隔。

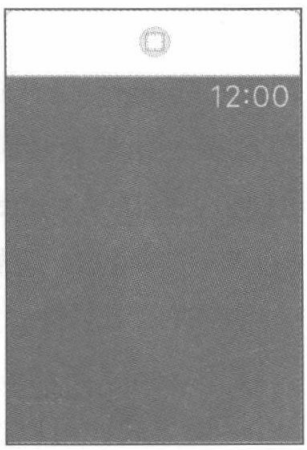


图 C-37　Interface Controller

2. Glance 界面控制器

Glance 是 Apple Watch 三种用户交互之一，就和它的字面意思一样，它将重要信息展示在一个视图里，让用户能在一瞥之间快速获取，是对一个完整的 Watch app 的有益补充。Glance 界面控制器（Glance Interface Controller）也属于 WKInterfaceController，并且一个应用中有且只能有一个 Glance 界面控制器，如图 C-38 所示。

对于 Glance 界面控制器来说，它只有两种属性可以设置：

- Upper（上部）：这个属性决定了 Glance 视图上面部分的显示模式，可以从 12 种模式中选择，包括图像、标签和数字的各种组合。
- Lower（下部）：这个属性决定了 Glance 视图下面部分的显示模式，可以从 10 种模式中选择，包括图像、标签和数字的各种组合。

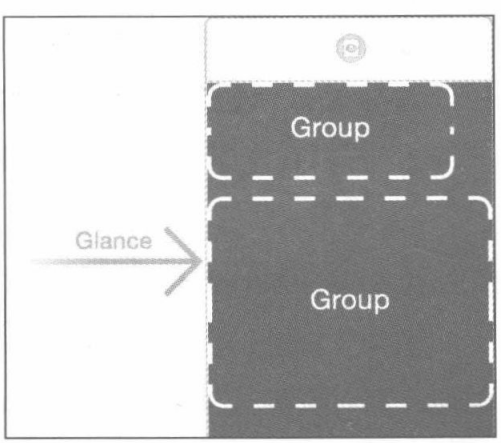


图 C-38　Glance Interface Controller

3. 通知界面控制器

通知界面控制器（Notification Interface Controller）用于手表给用户推送通知消息，来提醒用户某些事件发生，就如同 iOS 的通知一样，如图 C-39 所示。其对应类是 WKUserNotificaitonInterfaceController，继承自 WKInterfaceController。

除了界面控制器拥有的属性外，通知界面控制器还拥有以下属性：

- Has Dynamic Interface（拥有动态界面）：这个选项设置了这个通知视图是否需要动态

界面，动态界面的提示由代码来指定，否则的话就只能以固定字符来弹出通知。动态界面夜市由界面控制器来进行控制的。

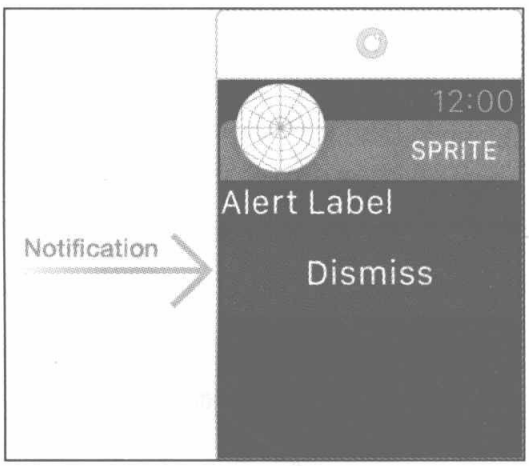


图 C-39 Notification Interface Controller

4. 组

组（Group）是 Watch 中多个界面元素的容器，通过包含多个元素，从而可以达到一定的“布局”效果。组的对应类是 WKInterfaceGroup。在 Glance Interface Controller 中就包含了两个组。

组拥有以下属性，其余属性和界面控制器的类似：

- Layout（布局）：这个属性决定了组内元素的布局方式，可以选择水平布局（Horizontal），也可以选择垂直布局（Vertical）。
- Radius（半径）：这个属性决定了组的半径，在组中的元素也一样会被这个元素给切割，从而制作处“圆角”的效果。

5. 表

表（Table）是 Watch 中存放多条数据的容器，可以以列表的形式向用户展现多条数据，和 iOS 上的表视图非常类似。表的对应类是 WKInterfaceTable，如图 C-40 所示。

表拥有以下属性，其余属性和界面控制器的类似：

- Rows(行): 这个属性决定了表元素中最多支持的行的数目。
- Positon（位置）: 这个属性决定了表视图的首行所在的位置，可以在水平或者垂直方向上进行选择，让其依次靠边或者居中。

图 C-40 Table

6. 图片

图片（Image）是 Watch 中显示图片的视图，将图片展示出来，和 iOS 上的图片视图非常类似，其对应类是 WKInterfaceImage。

图片拥有以下属性，其余属性和界面控制器的类似；

- Image（图片）：这个属性决定了图片视图显示的图片内容。
- Size（尺寸）：这个属性决定了图片视图的宽高设置，可以让其和内容大小相同，也可以让其和容器大小相同，或者固定其大小。

7. 分隔符

分隔符（Separator）是一个特殊的元素，它和表视图的分隔符很类似，用来将不同的内容分隔开来，给用户提供一个良好的 UI 设计，其对应类是 WKInterfaceSeparator，如图 C-41 所示。

图 C-41 Separator

分隔符的属性在前面介绍过的元素中都能找到。

8. 按钮

按钮（Button）是一个可以响应用户操作的元素，从而做出相应的动作，其对应类是 WKInterfaceButton，如图 C-42 所示。

按钮的属性在前面介绍过的元素中都能找到，并且和 UIButton 十分类似。

9. 开关

开关（Switch）是一个有两种状态的元素，分别对应某件事物的“正 / 反”两面，其对应类是 WKInterfaceSwitch，如图 C-43 所示。

开关的属性在前面介绍过的元素中都能找到，并且和 UISwitch 十分类似。

10. 滑动条

滑动条（Slider）是一个用来控制“进度”的重要元素，其对应类是 WKInterfaceSlider，如图 C-44 所示。

图 C-42　Button

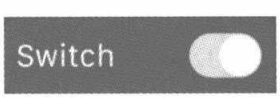


图 C-43　Switch

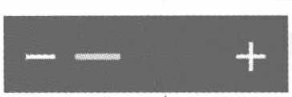


图 C-44　Slider

滑动条的属性在前面介绍过的元素中都能找到，并且和 UISlider 十分类似。

11. 标签

标签（Label）是一个用来显示某种文字内容的重要元素，其对应类是 WKInterfaceLabel。

标签的属性在前面介绍过的元素中都能找到，并且和 UILabel 十分类似。

12. 日期

日期（Date）是一个用来专门显示日期的元素，其对应类是 WKInterfaceDate，继承自 WKInterfaceLabel。

日期除了标签拥有的属性外，还拥有自己特有的属性：

- Format（格式化）：决定日期元素的显示格式化方式，Standard 表示采取系统设置的格式化方式，Custom 表示自行设置日期格式化方式。如果选择自定义的话，那么就需要自行设定日期格式化符。
- Date（日期）：决定年、月、日的显示状态，可以选择不显示、短段格式、中段格式、长段格式以及完整格式。
- Time（时间）：决定时、分、秒的显示状态，可以选择不显示、短段格式、中段格式、长段格式以及完整格式。
- Preview（预览）：决定初始状态下日期元素默认显示的日期。

13. 计时器

计时器（Time）是一个用来专门显示计时器时间的元素，其对应类是 WKInterfaceTimer，继承自 WKInterfaceTimer。

计时器除了标签拥有的属性外，还拥有自己特有的属性：

- Format（格式化）：决定计时器的显示格式化方式，有几种模式可供选择，Positional（按位显示）则是 0:59:59 的样式，Abbreviate（缩略显示）则是 59m 59s 的样式，Short（短段显示）则是 59 min, 59 secs 的样式，Full（完整显示）则是 59 minutes, 59 seconds 的样式，Spelled Out（完整拼写）则是 fifty-nine minutes, fifty-nine seconds 的样式。
- Units(显示单元)：决定该计时器当前显示的计时格式，可以选择显示年、月、周、日、时、分、秒。
- Preview Secs（预览秒数）：决定初始状态下计时器默认显示的时间。

14. 地图

地图（Map）是一个专门用来显示地图的元素，其对应类是 WKInterfaceMap。

地图的属性很简单，只有一条 Enabled，表示是否可用。

15. 菜单

菜单（Menu）是一个用来显示菜单选项的元素，其对应类是 WKInterfaceMenu。菜单是 Watch 特有的交互，长按手表屏幕可以弹出，最多提供 4 个菜单项。

菜单的属性很简单，决定了菜单当前拥有多少菜单项。

16. 菜单项

菜单项（Menu Item）是菜单中用来显示并响应用户操作的元素，其对应类是 WKInterfaceMenuItem。

菜单项的属性很简单，可以设定其标题、图标等内容。

附录 D Appendix D

你不会独孤求败——求助渠道

苹果提供了多种方式来帮助开发者获取 Xcode 以及 iOS/Mac 开发的相关帮助，本章将帮助你找到所需的答案。

D.1 帮助菜单

最明显的地方就是帮助菜单了，它位于菜单栏上面，直接以 Help 的名字显示。帮助菜单的界面如图 D-1 所示。

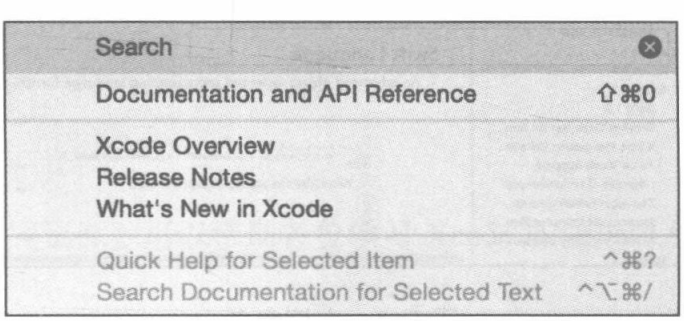


图 D-1 帮助菜单

帮助菜单提供了一个搜索栏，可以用它来查找帮助，并且还可以搜索对应的菜单项（Menu Item），直接定位到菜单项上。比如说可以搜索“ Run”，这时候搜索栏就会弹出一个搜索结果列表，显示搜索到的菜单项，将鼠标指向菜单项，Xcode 会自动弹出该菜单项的所在位置，如图 D-2 所示。

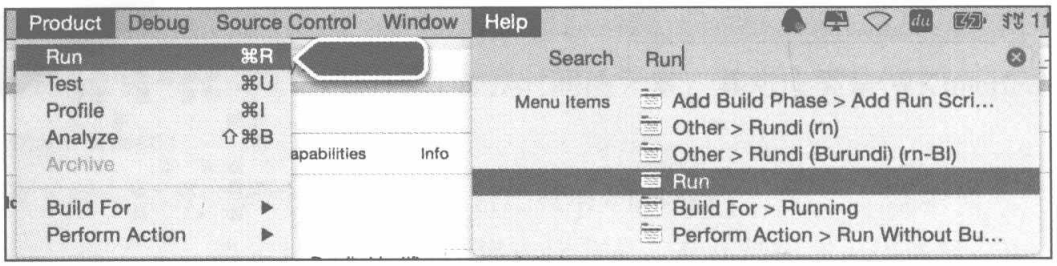


图 D-2 搜索菜单项

在搜索栏下面就是一些具体的帮助内容了，绝大部分菜单项都将打开“文档界面”（Documentation）。

1. 文档和 API 参考

Documentation and API Reference（文档和 API 参考）菜单项将会打开文档界面，如图 D-3 所示。在这个文档界面可以浏览当前电脑上已安装的文档库，还可以在本地文档库中进行搜索。

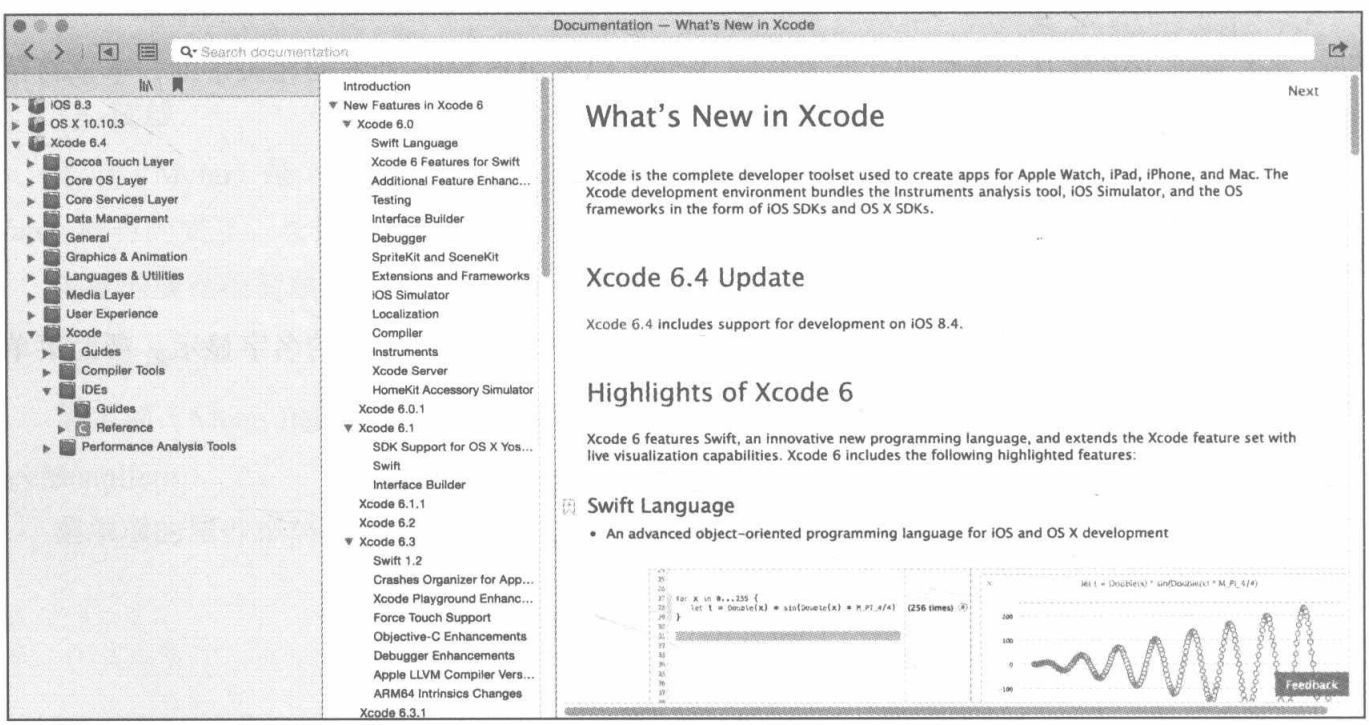


图 D-3 文档参考

2. Xcode 概览

Xcode Overview 将会打开 Xcode Overview 的相关内容，该概览的相关中译版请参阅 CocoaChina 网站上的相关翻译专题。

3. 更新日志

Release Notes 将会打开 Xcode 的更新日志，在这里可以看到以往的版本都有哪些变化。

4. Xcode 的新功能

What's new in Xcode 将会介绍当前 Xcode 拥有哪些全新的，令人振奋的新功能。

5. 其余菜单项

剩下的菜单项用来触发代码编辑区中的快速帮助，关于“快速帮助”的有关内容，本书已经介绍过，在此不再加以赘述。

D.2　文档界面

文档界面是查看大多数帮助文档的主要界面，它的界面如图 D-3 所示。

1. 导航器

文档界面的左侧是导航器窗口。这个导航器窗口拥有两个选项卡，分别是文档库（Documentation Library）和书签页（Documentation Bookmarks）。

其中，文档库列出了当前本机上拥有的所有版本的文档。比如说 iOS 8.3、OS X 10.10.3 以及 Xcode 6.4，说明本机上拥有这三个版本的文档说明，这些文档都可以离线查看。

对于书签来说，则是可以快速跳转到已经记录书签的文档界面。添加书签的方式很简单，右键单击页面的任何位置，在弹出的菜单中选择“Add Bookmark”即可完成添加；或者在每段文字的开头有一个小小的书签标识，选择它也可以完成书签的添加。

2. 目录和分享

目录界面列出了当前所选文档的目录结构。

而分享则可以将当前所选的文档分享至社交网络。

至于如何添加新的文档，请参阅本书的附录 A。

D.3　其他资源

有一些非常赞的在线资源，可以帮助各位学习 Xcode 的使用以及开发，这里列出了我们最喜欢的资源。

1. Apple 官方资源

- 开发人员论坛（http://devforums.apple.com）开发人员论坛有一个专门针对 Xcode 的板块，这里面拥有很多大神，里面解答了许许多多的问题。这需要 Apple 开发者账号才能够登录进入。
- 开发者官方网站（https://developer.apple.com）这个是苹果的官方开发者网站，在上面可以找到 iOS、Xcode、Swift 等多个资源的官方文档、示例等等。

2. 第三方资源

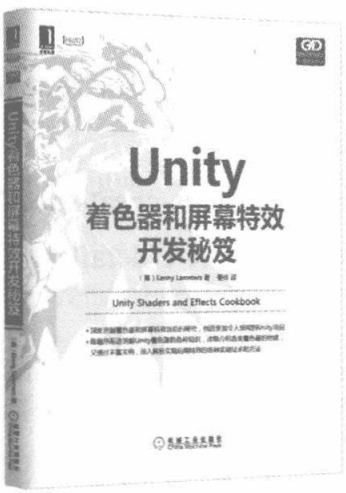
- Stack Overflow（http://stackoverflow.com）这是一个可以说是全球最大的“开发者问答中心”，这里面有很多涵盖各种方面的问题，并且几乎都有大神来解决。绝大部分使用 Xcode 和进行 iOS 开发的问题在上面都能找到，不过由于这是外站，访问会比较缓慢。
- CocoaChina（http://www.cocoachina.com）这是中文世界一个比较大的 iOS 开发文章平台，在此上面可以找到许许多多汇总过后的文章，总之是一个值得关注学习的网站。
- Ray Wenderlich（http://www.raywenderlich.com）这是一个关于 iOS 开发的教程网站，上面有海量的文章、视频教程。
- NSHipster（http://nshipster.com）这个网站以 NS 作为开头，表明了其是一个介绍 Cocoa 相关小技巧的集体博客。

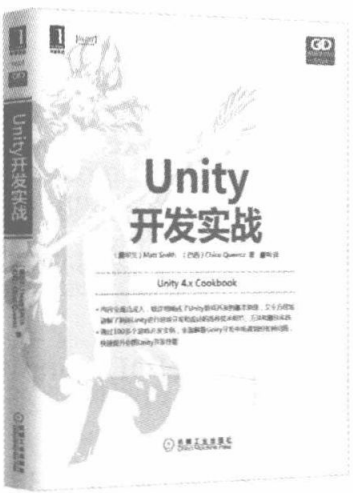
D.4 本书参考文献

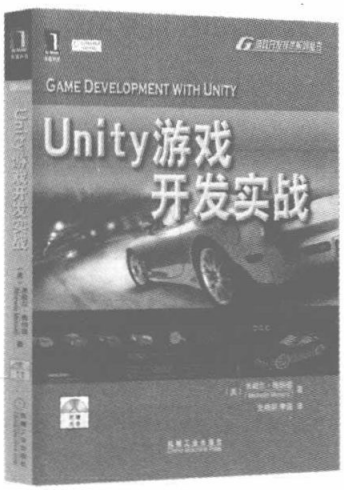
- Maurice Kelly, Joshua Nozzi. Xcode 实战开发 [M]. 姚军，译 . 北京：人民邮电出版社，2014.
- James Bucanek. Xcode 3 高级编程 [M]. 张龙，译 . 北京：清华大学出版社，2012.
- Richard Wentk. Xcode 5[M]. the United States of America: John Wiley & Sons, Inc., 2014.
- Fritz Anderson. Xcode 5 start to finish: iOS and OS X development[M]. the United States of America: Pearson Education, Inc., 2014.
- Apple Inc. Testing with Xcode[DB/OL]. https://developer.apple.com/library/prerelease/ios/documentation/DeveloperTools/Conceptual/testing_with_xcode, 2015-06-08.
- Apple Inc. Xcode Concepts[DB/OL]. https://developer.apple.com/library/prerelease/ios/featuredarticles/XcodeConcepts, 2011-05-09.
- Apple Inc. Xcode Overview[DB/OL]. https://developer.apple.com/library/prerelease/watchos/documentation/ToolsLanguages/Conceptual/Xcode_Overview/, 2015-06-08.
- CocoaChina, 星夜暮晨 [新浪微博], Creolophus[Github], 唧唧歪歪 [新浪微博]. Xcode 概览（Xcode 6 版）[DB/OL]. http://www.cocoachina.com/ios/20141212/10626.html, 2014-12-12.
- Apple Inc. Debugging with Xcode[DB/OL]. https://developer.apple.com/library/prerelease/ios/documentation/DeveloperTools/Conceptual/debugging_with_xcode/, 2015-06-22.
- Apple Inc. Playground Help[DB/OL]. https://developer.apple.com/library/prerelease/ios/recipes/Playground_Help/_index.html, 2015-06-22.

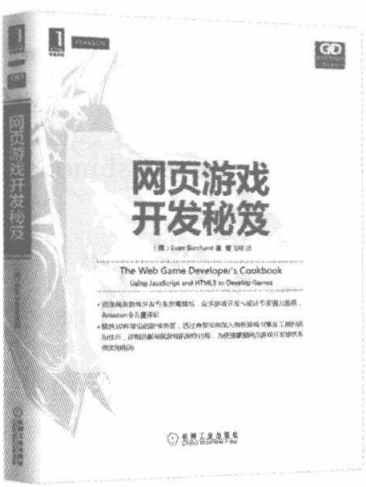
- 老码团队 . 老码说编程之玩转 Swift 江湖 [M]. 北京：电子工业出版社 , 2015.
- Apple Inc. Simulator User Guide[DB/OL]. https://developer.apple.com/library/prerelease/watchos/documentation/IDEs/Conceptual/iOS_Simulator_Guide/TestingontheiOSSimulator.html, 2015-06-08.
- gabriel theodoropoulos, yuewang（译）. Xcode 6 Tips: Vector Images, Code Snippets and Many More[OL]. http://www.appcoda.com/xcode6-tips-tricks/, 2015-5-5.
- git-tower. Xcode Cheat Sheet[OL]. http://www.git-tower.com/blog/xcode-cheat-sheet/, 2015.
- [CSDN] 弦苦 . Xcode 基本操作 [OL]. http://blog.csdn.net/phunxm/article/details/17044337, 2013-11-30.
- [CSDN]qiummm. Xcode 之 File Merge[OL]. http://blog.csdn.net/tingxuan_qhm/article/details/19107881, 2014-02-12.
- SLBoat Insight. Xcode Accessibility Inspector[DB/OL]. http://see.sl088.com/wiki/Xcode_Accessibility_Inspector, 2014.
- Scott Berrevoets. View Debugging in Xcode 6[OL]. http://www.raywenderlich.com/98356/view-debugging-in-xcode-6, 2015-04-23.
- Sam Davies. iOS7 Day-by-Day::Day 2:: Asset Catalog[OL]. https://www.shinobicontrols.com/blog/ios7-day-by-day-day-2-asset-catalog, 2013-09-23.
- Simon ng. Creating Hello World App Using Xcode 5 and Interface Builder[OL]. http://www.appcoda.com/hello-world-app-using-xcode-5-xib, 2013-10-12.
- 葛布林大帝 [CNBlog]. [iOS 翻译]《 iOS7 by Tutorials 》系列：在 Xcode 5 里使用单元测试 [OL]. http://www.cnblogs.com/yangfaxian/p/3782856.html, 2014-06-11.
- Mattt Thompson. XCTestCase / XCTestExpectation / measureBlock()[OL]. http://nshipster.com/xctestcase, 2014-07-21.
- SUNNYXX. iOS8 Size Classes 初探 [OL]. http://blog.sunnyxx.com/2014/09/09/ios8-size-classes/, 2014-09-09.

推荐阅读









Unity着色器和屏幕特效开发秘笈

作者：Kenny Lammers ISBN：978-7-111-48056-3 定价：49.00元

Unity开发实战

作者：Matt Smith 等 ISBN：978-7-111-46929-2 定价：59.00元

Unity游戏开发实战

作者：Michelle Menard ISBN：978-7-111-37719-1 定价：69.00元

网页游戏开发秘笈

作者：Evan Burchard ISBN：978-7-111-45992-7 定价：69.00元

游戏开发工程师修炼之道（原书第3版）

作者：Jeannie Novak ISBN：978-7-111-45508-0 定价：99.00元

HTML5 Canvas核心技术：图形、动画与游戏开发

作者：David Geary ISBN：978-7-111-41634-0 定价：99.00元

{
  "filename": "WENPREXmsZ/muZblvZVfMTM4Nzc1MTAuemlw",
  "filename_decoded": "XCODE\u6c5f\u6e56\u5f55_13877510.zip",
  "filesize": 124701671,
  "md5": "2886fa0a03f6a57aadc16f553988b13a",
  "header_md5": "6fccae5af7d7b44d0982efa997f001c5",
  "sha1": "aac487e201a7ed73c8288d7309a69b99e34a9a0b",
  "sha256": "46ed2f16578643064402bb15e77cf61f8ebd4d89762aff7ce39baf3630f30be7",
  "crc32": 918524989,
  "zip_password": "",
  "uncompressed_size": 142694785,
  "pdg_dir_name": "XCODE\u255c\u00a1\u2551\u25a0\u252c\u255d_13877510",
  "pdg_main_pages_found": 350,
  "pdg_main_pages_max": 350,
  "total_pages": 366,
  "total_pixels": 2013133248,
  "pdf_generation_missing_pages": false
}