

Join the discussion @ p2p.wrox.com



Wrox Programmer to Programmer™

移动开发经典丛书



Professional iOS Programming

iOS高级编程

[法] Peter van de Put
李 军

著
译



清华大学出版社

本书仅提供部分阅读，如需完整版，请联系QQ: 461573687

提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

备用QQ:2404062482

移动开发经典丛书

iOS 高级编程

[法] Peter van de Put 著

李 军 译

清华大学出版社

北 京

Peter van de Put
Professional iOS Programming
EISBN: 978-1-118-66113-0
Copyright © 2014 by John Wiley & Sons, Inc.
All Rights Reserved. This translation published under license.

本书中文简体字版由 Wiley Publishing, Inc. 授权清华大学出版社出版。未经出版者书面许可, 不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号 图字: 01-2014-0760

Copies of this book sold without a Wiley sticker on the cover are unauthorized and illegal.

本书封面贴有 Wiley 公司防伪标签, 无标签者不得销售。

版权所有, 侵权必究。侵权举报电话: 010-62782989 13701121933

图书在版编目(CIP)数据

iOS 高级编程 / (法) 范德普特 著; 李军 译. —北京: 清华大学出版社, 2014
(移动开发经典丛书)
书名原文: Professional iOS Programming
ISBN 978-7-302-38225-6

I. ①i… II. ①范… ②李… III. ①移动终端—应用程序—程序设计 IV. ①TN929.53

中国版本图书馆 CIP 数据核字(2014)第 230631 号

责任编辑: 王 军 于 平
装帧设计: 牛静敏
责任校对: 成凤进
责任印制: 杨 艳

出版发行: 清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>
地 址: 北京清华大学学研大厦 A 座 邮 编: 100084
社 总 机: 010-62770175 邮 购: 010-62786544
投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn
质 量 反 馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者: 清华大学印刷厂

装 订 者: 三河市吉祥印务有限公司

经 销: 全国新华书店

开 本: 185mm×260mm 印 张: 33.75 字 数: 821 千字

版 次: 2014 年 10 月第 1 版 印 次: 2014 年 10 月第 1 次印刷

印 数: 1~3000

定 价: 79.80 元

译者序

iOS 系统随着 iOS 设备在全世界的热销，这些年一直是手机市场的宠儿，凭借其操作简便、体验流畅和性能强大等特点，自诞生以来就受到全球各地人们的追捧。iOS 编程是基于简单强悍的 Objective-C 语言，不需要 C、C++ 和 Java 语言深厚的功底，就可以开发出优秀的 iOS 应用。而且，凭借 Xcode 开发环境及相关工具，初学者可以毫不费力地搭建起 iOS 的应用框架，迅速实现产品的原型，甚至可以很快转化为可以发布到 App Store 上的应用，造就了非常多的财富神话。

对于初学者来说，目前市面上介绍 iOS 编程的书籍太多了，可谓车载斗量，浩若繁星。《iOS 高级编程》则是众多 iOS 编程学习资源中的经典作品。本书的作者是经验丰富的 iOS 编程大师 Peter van de Put。Peter van de Put 从 1980 年开始开发软件，具有丰富的实践经验，其开发的软件被多家全球知名企业所使用。并且，作为一位经验丰富的导师，他先后培训了数以百计的开发者，可以为世界各国的 iOS 开发人员提供培训的课程。《iOS 高级编程》作为 Peter van de Put 的经典之作，将会培养一代又一代的 iOS 开发人员，对 iOS 编程的推广可谓是功不可没。

编程语言的学习十分枯燥，学习的过程也特别艰辛，但是学成之后所获得的成就感也是无与伦比的。iOS 应用开发在很大程度上得益于框架，而各种框架也往往是让初学者头疼的地方，甚至成为一个梦魇。我深知其学习的艰辛，对于自己在学习过程中的彷徨犹豫，挫折困顿，至今还历历在目，犹如昨日。《iOS 高级编程》循序渐进、全面系统的讲解，曾经让自己大惑不解的地方，从本书看来则是如此的理所当然，水到渠成。恨自己没在初学之时，早点读到此书。

纸上得来终觉浅，绝知此事要躬行。本书的作者深谙此理，在每章都结合实际案例，配合大量的截图和示例代码，使读者可以亲自实践。学习一门语言没有比动手实践更快、更好的方法了。所以建议读者在阅读每一章的时候，亲自动手实现每个案例，而不是“看过代码，如此方能成为理论和行动上的“巨人”。有人问大师，如何能技近乎道？大师曰：读书，读好书，然后实践之。万事无他，惟手熟尔！

在这里要感谢清华大学出版社的编辑们，她们为本书的翻译投入了巨大的热情并付出了很多心血。没有她们的帮助和鼓励，本书不可能顺利付梓。参与本次翻译活动的还有赵丽影、孔祥亮、陈跃华、杜思明、熊晓磊、曹汉鸣、陶晓云、王通、方峻、李小凤、曹晓松、蒋晓冬、邱培强、洪妍、李亮辉。在此一并谢过！

对于这本经典之作，译者本着“诚惶诚恐”的态度，在翻译过程中力求“信、达、雅”，但是鉴于译者水平有限，错误和失误在所难免，如有任何意见和建议，请不吝指正。感激不尽！

最后，希望读者通过阅读本书能早日步入 iOS 高级编程的殿堂，领略 iOS 编程之美！

译 者

序 言

自从在 2008 年引入 App Store 以来，已经产生了超过 90 万款的应用，总计 5 千万次下载(截至 2013 年 6 月)。

这一现象的效应很简单：移动应用的新兴市场非常赚钱。

在 2007 年 1 月份，当乔布斯在苹果大会和世博会上揭开 iOS 系统的第一部 iPhone 时，引起了全球媒体的广泛报道。此外，在 2007 年 6 月份，第 1 版 iOS 系统的发布，激起了传统的苹果粉丝巨大的兴趣。围绕着用户使用触摸和手势进行交互这一理念而打造的外观惊异的智能手机，也引起了开发人员对这一新的操作系统的兴趣。

随着越来越多的“如何做”的视频可以在互联网上找到，人们开始通过复制和粘贴零碎的代码来完成他们的应用，而并未真正懂得他们在做什么。当他们的应用需要显而易见之外的那些功能时，他们开始遇到问题。

《iOS 高级编程》就是着眼于这些开发人员而编写的。本书通过使用遵循现代 Objective-C 编程准则的可用 iOS 框架，非常详细地逐步讲解了如何创建高级的 iOS 应用。本书中(也可以下载)所包含的 70 个现实世界的示例程序用来讲解功能，而且，你可以自由地使用这些程序作为应用的起点。

本书的作者 Peter van de Put，从 1980 年就在 Sinclair ZX-81 上开始他的编程生涯，这是一台仅有 16KB 的全部内存，能够用汇编编程的机器。在随后的若干年里，Peter 学习了许多其他的编程语言，从基于寄存器使用汇编编程，过渡到了使用 Delphi、Java、C#和 C++等语言的面向对象编程。

在 2009 年，Peter 开始钻研 iOS 和 iPhone 设备背后的技术，同时掌握了 Objective-C 语言。

从那时开始，他和他的专业团队为澳大利亚、欧洲和美国的客户开发 iPhone 和 iPad 应用，他们使用最新的现代编程实现和 SDK，交付端对端的解决方案，从后端管理系统到 iOS 应用。

如果你是一名有一些经验的 iOS 开发人员，而且想要提高和扩展技能，从而能够开发现实生活中的 iOS 应用；或者，你是一名具有很少经验的开发人员，需要频繁登录谷歌查找与 Objective-C 语言或者与某个 iOS 框架有关的“如何做”的答案，那么，本书一定值得添加到你的书架上。阅读本书会为你节省许多宝贵的时间。

“《iOS 高级编程》将成为 iOS 所有开发人员必备的著作，并且，我们认为本书也应该同样适用于你们。”

—Pertti Karjalainen
生产经理, Northern Lights Software
www.northernlightssoftware.com

作者简介

Peter Van De Put 是 YourDeveloper 公司的 CEO(Chief Executive Officer, 首席执行官)兼首席开发者。这家公司总部位于法国, 是一家全球性的软件开发公司。他从 1980 年开始开发软件, 并且为一些大公司交付高端软件的解决方案, 例如像富士胶片公司(Fuji Photo Film), 壳牌公司(Shell), 联合利华公司(Unilever), 普利斯通公司(Bridgestone), 阿尔卡特(Alcate), 理光公司(Ricoh)及其他许多知名公司。2006 年, 他创立了一家软件公司, 专注于开发 iOS 应用和后台软件解决方案, 并为全球客户开发出各种应用, 例如银行、政府机构, 电信和公用事业部门。伴随着他个人的软件开发事业的发展, 他也先后培训了数以百计的开发者并联合创立了几家服务公司, 并担任这些公司的项目经理和商业顾问。通过掌控和管理顾问公司, 他已经掌握了这些项目的各个方面, 从计划到设计, 再到部署及维护。

而且, 作为一位经验丰富的导师, 他也可以提供给世界各国的 iOS 开发人员培训的课程。

技术编辑简介

Abhishek Mishra 已经具有超过 13 年的软件开发经验，并且经历了多个不同种类的编程语言和平台。他曾为 EURO RSCG 公司和 MusicQubed 公司效力，并做过多个 iOS 项目，并且最近成为 iOS 首席顾问在英国天然气公司(英国森特理克集团的分部)工作。Abhishek 是 *iPhone and iPad App 24 Hour Trainer*(Wiley, 2010)一书的作者，他持有伦敦大学计算机科学专业的硕士学位。他居住在伦敦，和他的妻子在闲暇时间里从事跨平台游戏引擎的开发和动画短片的制作。

致 谢

我首先要感谢的是我的妻子 **Miranda** 和女儿 **Anique**，她们在我的创作过程中给予我不断的支持和鼓励。

感谢 **Andre Smits**，我最好的朋友，因为他是支持和鼓励我，并且完成所有的校对工作并提供非常有益的反馈的那个人。

感谢我的所有客户，正是由于他们，才使得我可能获得这一经验水平，这些经验是在实现他们的项目的过程中积累的。

最后，向在著作过程中对本书感兴趣的朋友们和用户们致谢。

前 言

我第一次与计算机结缘是 15 岁那年在荷兰读高等技术学校时,在那里我结识了 Sinclair ZX-81。这是一台只有 16KB 内存的小型计算机,但是它很神奇,我从接触它的第一天起就开始了我的编程生涯。

在 1981 年,当硬件和软件革命开始时,我又使用了 Exidy Sourcer 和 Commodore 64 计算机,很快我又有了我的第一台个人电脑 XT(IBM Personal Computer XT)。让我着迷的事实是,你可以通过编程让计算机去精确地执行想要它们做的事情。而且,我能够为这些计算机的系统开发软件。

我开始用汇编语言编程,在一段时间后,我学会了用 C、Basic、QuickBasic、Delphi、Pascal、Turbo Pascal、C++、Java、Microsoft VB.NET、Microsoft C#和 Objective-C 等语言编程。编程成为了我的职业,我热爱这份职业。

不管怎样,像许多程序员一样,由于之前已经获得的知识和技能,我做了分析师、软件工程师和顾问,并最终进入了管理岗位。我的编程时间开始变得越来越少,而这却是我最喜欢做的事。

在我的职业生涯中,我管理和共同成立了几家 IT 服务公司,并在这些公司中担任项目经理、商业顾问和董事,但是我总是能够在这些项目中找到时间去做一些软件开发工作。

在这些年里,我曾经给一些大公司交付过高端软件解决方案,例如富士胶片公司(Fuji Photo Film)、壳牌公司(Shell)、联合利华公司(Unilever)、普利司通公司(Bridgestone)、阿尔卡特(Alcate)、理光公司(Ricoh)以及全世界的其他许多公司。我不是图像设计师,并且设计应用的视觉效果并不是我的强项。我向来关注于代码优化和新技术的探索。开发出高阶面向对象的代码而又使用极少内存占用空间的设计仍旧是一项挑战。

在 2006 年,我决定抛弃荷兰的繁忙生活,搬到法国,在那里我成立了一家软件公司,专注于开发后端的软件解决方案,不久后,也专注于开发 iOS 应用。

我的专业团队现在为澳大利亚、欧洲和美国的客户开发 iPhone 和 iPad 应用,在这一领域内,我们使用最新的 SDK 和现代程序开发实践,并交付从后端的管理系统到 iOS 应用的端到端的解决方案。

本书读者对象

这本书适用于想要提高和扩展 Objective-C 语言编程技能的开发人员、有经验的 iOS 开发者以及毫无经验的 iOS 开发新人。

本书需要读者掌握 Xcode 开发环境的一些基本知识和 Objective-C 语言开发的基础，这样能够理解本书提供的详细而深入的说明，以及 70 个编程示例代码。

本书在介绍开发高级 iOS 应用所涉及的技术方面，对想要加深理解其内容的每一位读者和 iOS 开发人员都很有价值。本书可以被当作参考书来使用，用以理解 iOS SDK 的细节。

本书包含编程技巧的详细说明，以及 70 个示例代码，这也使本书成为大学教授和培训师的理想教材。

本书主要内容：

本书涵盖了开发高级 iOS 应用所需要的所有主题。所有的说明和代码示例都适用于最新的 iOS 7 SDK 并经过测试。

本书涵盖以下主题范围：

- 创建 Personal Library
- UITableView
- Map Kit 框架
- Action 和 Alert 概述
- 国际化
- 在你的应用中使用多媒体
- 使用 Web 服务和 XML-JSON 解析
- 开发 FTP 客户端
- Core Data 框架的实现
- 使用通知
- 实现 E-mail、SMS 和拨号功能
- Address Book 框架的介绍和使用
- Event Kit 框架编程
- Social and Accounts 框架的使用
- 分析你的应用
- 从应用获利和 Store Kit 框架的使用
- 证书和配置文件
- 打包和发布你的应用

Interface Builder 和 Storyboard 的用法，以及用户界面设计范例

每个 iOS 程序员在他们的应用中如何创建用户界面元素都有个人的喜好。在我自己的

工作中，我使用代码创建所有的用户界面元素，因为我喜欢这种方式带来的可控性和可扩展性。因此，在这本书的绝大多数的课程中，你将发现使用 **Interface Build** 文件。例如，有些开发者更喜欢使用 **Storyboard**，在任何项目中，他们都可以使用 **Storyboard** 代替 **Interface Builder**，你也可以如此选择。这本书在两种情况下对开发者来说应该都有用处。

本书篇章结构

本书分为 4 个部分：

- 第 I 部分：开发高级 UI
- 第 II 部分：网络和数据处理
- 第 III 部分：集成应用
- 第 IV 部分：将应用运用到生产环境中

第 I 部分：开发高级 UI 包含以下章节：

- 第 1 章：创建 **Personal Library**
- 第 2 章：**Tableview** 进阶指南
- 第 3 章：**Map Kit** 框架
- 第 4 章：**ActionView** 和 **Alert** 概述
- 第 5 章：国际化：为全世界构建应用
- 第 6 章：多媒体的使用

第 1 章以开发包含可重用代码的 **Personal Library** 类为开头，你可以将这些代码用作基础框架在每一个应用中使用。在后续的章节中，你将用更多的功能扩展这个 **Personal Library** 类。

第 2 章将教你构建让人眼前一亮的表视图的方法，你将学会如何构建带有下拉即出现搜索条功能的表视图。

第 3 章全部是关于 **Map Kit** 框架的内容。你将了解位置管理器(**Location Manager**)的概念，并且会以开发为目的扩展 **GPS** 模拟器。你将会开发带有自定义标注的群集地图视图。

动作视图和提醒控件是第 4 章要讨论的主题内容。你将学会如何在应用中实现动作视图和提醒控件，与使用应用的用户交互。

第 5 章将通过对本地化的技巧的讲解，以及对国际化日期和数字格式的方法的讲解，教会你如何为全世界的用户国际化应用。

第 6 章全部介绍的是多媒体。你将学会显示和创建 **PDF** 文档的有效方法，以及使用不同的可用框架播放和录制音频或者视频。

第 II 部分：网络和数据处理包含以下章节：

- 第 7 章：使用 **Web** 服务和解析
- 第 8 章：使用 **FTP**
- 第 9 章：**Core Data** 框架的实现

第 7 章将教会你如何通过 REST 或 SOAP 协议使用 Web 服务，以及如何使用 GET 和 POST 动作将数据发送给这些 Web 服务。对以 XML 和 JSON 格式返回的应答信息的解析将在本章的结尾详细阐述。

第 8 章将讲解文件传送协议(File Transfer Protocol, FTP)适配 iOS 应用架构的方法。还将讲解使用 Objective-C 语言怎样能够编写出简单的 FTP 客户端。应大多数读者的进一步要求，本章也会讲解基于 Objective-C 的类编写一条 FTP 命令的方法。

第 9 章将讲解所有关于 Core Data 框架的内容。它阐述了 Core Data 这个框架的概念，存储方法、实体、关系以及读取数据的方法。

第III部分：集成应用包含以下章节：

- 第 10 章：通知
- 第 11 章：发送电子邮件、SMS 和拨打电话
- 第 12 章：了解 Address Book
- 第 13 章：事件编程
- 第 14 章：与社交媒体集成

第 10 章将会讲解实现内部通知和外部推送通知的方法。

第 11 章将会讲解从应用内发送 E-mail 和短信消息的方法，以及拨打电话的方法。

第 12 章将会讲解使用 Address Book 框架读取和写入联系人数据库的方法。你将学会如何请求访问联系人数据库的权限，以及为了使用联系人数据而展示用户界面。

第 13 章将会讲解从应用内创建和管理事件和提醒的方法。

第 14 章将会讲解在应用中内部集成 Facebook 和 Twitter 的方法。你将学会如何展示用户的 Tweet(注：Twitter 上用户发送的一条消息)和 Facebook 用户墙上的留言，以及如何发表留言到 Facebook，或者发送 Tweet。

第IV部分：将应用转化为产品包含以下章节：

- 第 15 章：分析应用
- 第 16 章：从应用中获利
- 第 17 章：了解 iTunes Connect
- 第 18 章：构建与发布

第 15 章讲解在应用中实现耗用分析的方法。

第 16 章讲解从应用中获利的方法。本章包含了深入的阐述和程序内购买的辅助类。本章也涵盖包括像 iAd 和 AdMob 在内的 Advertisement 框架的实现。

第 17 章讲解使用 iTunes Connect 为应用的提交作准备的方法。理解 provisioning profile、certificate 和 device 这几个名词对能够发布应用来说是至关重要的。

第 18 章，最后一章，讲解以 Ad-Hoc 发布版，抑或是以 App Store 发布版构建和发布应用的方法。

使用本书条件

为了编写 iOS 应用，你需要下载包含有最新 iOS SDK 的最新版本的 Xcode。你可以在这里下载 <http://developer.apple.com>。

本书约定

为了帮助你最大限度地理解文字内容并提醒你注意正在介绍的内容，我们在本书中使用了大量的规范。



警告：此处内容表示非常重要、不可遗忘的信息，这些信息与其周围的内容密切相关。



提示：提示、贴士、暗示、技巧以及当前内容的延伸介绍会以此形式进行补充。

源代码

在练习书中的示例时，可以选择手动输入代码或者使用本书附带的源代码文件。书中用到的所有源代码都可以从 www.wrox.com 下载。进入站点 <http://www.wrox.com> 后，只需要找到本书的书名(使用 Search 搜索框或书名列表)，单击本书详细信息页面上的 Download Code 链接，就可以得到本书所有的源代码。



注意：因为很多书的书名都相似，所以用 ISBN 搜索更为容易。本书英文版的 ISBN 是 978-1-118-49582-7。

下载完代码后，用你喜欢的压缩工具把它解压缩。此外，也可以去 Wrox 的主下载页面 www.wrox.com/dynamic/books/download.aspx 找到本书或 Wrox 出版的其他书籍的代码。

勘误表

尽管我们已经尽了各种努力来保证文章或代码中不出现错误，但是错误总是难免的，如果你在本书中找到了错误，例如拼写错误或代码错误，请告诉我们，我们将非常感激。通过勘误表，可以让其他读者避免受挫，当然，这还有助于提供更高质量的信息。

要在网站上找到本书的勘误表，可以登录 <http://www.wrox.com>，通过 Search 工具或书名列表查找本书，然后在本书的细目页面上，单击 Book Errata 链接。在这个页面上可以查看 Wrox 编辑已提交和粘贴的所有勘误项。完整的图书列表还包括每本书的勘误表，网址是 www.wrox.com/misc-pages/booklist.shtml。

如果在 Book Errata 页面上没有看到你找出的错误，请进入 www.wrox.com/contact/techsupport.shtml，填写表单，发电子邮件，我们就会检查你的信息，如果是正确的，就在本书的勘误表中粘贴一个消息，我们将在本书的后续版本中采用。

p2p.wrox.com

P2P 邮件列表是为作者和读者之间的讨论而建立的。读者可以在 p2p.wrox.com 上加入 P2P 论坛。该论坛是一个基于 Web 的系统，用于传送与 Wrox 图书相关的信息和相关技术，与其他读者和技术用户交流。该论坛提供了订阅功能，当论坛上有新帖子时，会给你发送你选择的主题。Wrox 作者、编辑和其他业界专家和读者都会在这个论坛上进行讨论。

在 <http://p2p.wrox.com> 上有许多不同的论坛，帮助读者阅读本书，在读者开发自己的应用程序时，也可以从这个论坛中获益。要加入这个论坛，必须执行下面的步骤：

- (1) 进入 p2p.wrox.com，单击 Register 链接。
- (2) 阅读其内容，单击 Agree 按钮。
- (3) 提供加入论坛所需的信息及愿意提供的可选信息，单击 Submit 按钮。
- (4) 然后就可以收到一封电子邮件，其中的信息描述了如何验证账户，完成加入过程。



提示：不加入 P2P 也可以阅读论坛上的信息，但只有加入论坛后，才能发送自己的信息。

加入论坛后，就可以发送新信息，回应其他用户的帖子。可以随时在 Web 上阅读信息。如果希望某个论坛给自己发送新信息，可以在论坛列表中单击该论坛对应的 Subscribe to this Forum 图标。

对于如何使用 Wrox P2P 的更多信息，可阅读 P2P FAQ，了解论坛软件的工作原理，以及许多针对 P2P 和 Wrox 图书的常见问题解答。要阅读 FAQ，可以单击任意 P2P 页面上的 FAQ 链接。

目 录

第 I 部分 开发高级 UI

第 1 章 创建 Personal Library	3
1.1 创建 Personal Library	4
1.1.1 项目基础知识	4
1.1.2 启动新项目	5
1.1.3 配置项目	6
1.1.4 定义常量	8
1.1.5 使用配置文件	8
1.1.6 导入头文件	10
1.2 注册——登录	11
1.2.1 创建注册逻辑	12
1.2.2 初始化数据	15
1.2.3 初始化应用的默认设置	15
1.2.4 创建登录逻辑	16
1.2.5 保护密码的安全	19
1.2.6 在 keychain 中存储密码	20
1.3 崩溃管理	21
1.3.1 理解崩溃	21
1.3.2 实现崩溃处理程序	22
1.4 本章小结	28
第 2 章 Tableview 进阶指南	31
2.1 理解 UITableView	31
2.1.1 datasource 和 delegate	32
2.1.2 滚动	36
2.2 构建聊天视图控制器	39
2.2.1 构建 datasource	40
2.2.2 构建聊天数据对象	40
2.2.3 构建定制的 UITableView 控件	43
2.2.4 灵活的单元格高度	47

2.2.5 开发定制的单元格	47
2.2.6 创建聊天用户对象	51
2.2.7 融会贯通	52
2.3 UITableView 的下拉功能	58
2.3.1 实现 UISearchBar	68
2.3.2 添加字母表索引	73
2.4 本章小结	78
第 3 章 Map Kit 框架	79
3.1 模拟 iOS 设备的位置移动	80
3.1.1 为何需要一个 GPS 模拟器	80
3.1.2 创建模拟器	80
3.1.3 使用 Google Maps 创建 GPS 路线文件	84
3.1.4 实现 YDLocation- Simulator 类	88
3.2 使用标记	90
3.2.1 创建定制的标记	91
3.2.2 响应标记的批注	95
3.2.3 标记群集	101
3.3 本章小结	119
第 4 章 Action View 和 Alert 概述	121
4.1 请求用户输入	121
4.2 使用多个选项创建 UIAlertSheet	122
4.3 呈现 UIAlertController	126
4.3.1 使用 showInView 方法来 呈现	126
4.3.2 使用 showFromTabBar 方法 来呈现	126
4.3.3 使用 showFromBarButtonItem 方法来呈现	129

4.3.4	使用 showFromRect 方法来呈现	130
4.3.5	使用 showFromToolbar 方法来呈现	132
4.3.6	用户输入的响应	134
4.3.7	处理用户选项	134
4.3.8	扩展 UIAlertView	137
4.3.9	在 UIAlertView 上添加 UITextField	137
4.4	本章小结	141
第 5 章	国际化：为全世界构建应用	143
5.1	本地化应用	143
5.1.1	建立本地化	144
5.1.2	本地化 Interface Builder 文件	145
5.1.3	本地化字符串	147
5.1.4	本地化图片	149
5.1.5	本地化应用的名称	152
5.2	使用日期格式	152
5.2.1	区域的概念	152
5.2.2	日历概述	156
5.2.3	以通用方式存储日期	158
5.3	使用数字	159
5.4	本章小结	164
第 6 章	多媒体的使用	165
6.1	便携式文档格式	165
6.2	使用 UIWebView 显示 PDF 文档	166
6.3	Instruments 分析工具介绍	168
6.4	使用 QuickLook 显示 PDF 文档	170
6.5	从 PDF 文档创建缩略图	173
6.6	创建 PDF 文档	177
6.7	播放和录制音频	181
6.8	相关框架介绍	181
6.8.1	AVFoundation 框架	181
6.8.2	Audio Toolbox 框架	182
6.8.3	Media Player 框架	182

6.9	播放来自应用包的音频文件	182
6.10	从 iTunes 库中播放音频	187
6.11	播放音频流	190
6.12	录制音频	193
6.13	播放和录制视频	198
6.14	从 iTunes 库播放视频	201
6.15	播放 YouTube 视频	204
6.16	录制视频	206
6.17	本章小结	209

第 II 部分 网络和数据处理

第 7 章	使用 Web 服务和解析	213
7.1	为什么需要使用 Web 服务	213
7.2	了解基本网络	214
7.2.1	了解协议	214
7.2.2	了解操作	215
7.2.3	了解响应代码	215
7.3	Web 服务简介	215
7.4	调用 HTTP 服务	216
7.4.1	请求网站	216
7.4.2	从 HTTP URL 下载图片	219
7.4.3	使用 HTTPS 请求安全网站	224
7.4.4	使用数据块	227
7.5	调用 REST 服务	231
7.5.1	构建请求	232
7.5.2	处理响应	235
7.5.3	发布到 RESTful 服务	241
7.6	发出 SOAP 请求	248
7.6.1	为请求做准备	250
7.6.2	将值传递到操作	252
7.6.3	了解安全的 SOAP 请求	257
7.7	更多解析	259
7.7.1	逗号分隔值文件	260
7.7.2	将 XML 转换为 NSDictionary	266
7.8	本章小结	269

第 8 章 使用 FTP	271		
8.1 开发 FTP 客户端.....	271		
8.1.1 编写简单的 FTP 客户端.....	272		
8.1.2 下载远程文件.....	276		
8.1.3 创建远程目录.....	278		
8.1.4 列出远程目录.....	279		
8.1.5 上传文件.....	283		
8.1.6 从 NSStream 中读取.....	284		
8.1.7 写入 NSStream.....	284		
8.1.8 编写复杂的 FTP 客户端.....	288		
8.2 使用 FTP 客户端.....	296		
8.3 本章小结.....	296		
第 9 章 实现 Core Data	297		
9.1 Core Data 简介.....	297		
9.1.1 为什么应使用 Core Data.....	298		
9.1.2 托管对象上下文简介.....	298		
9.1.3 托管对象模型简介.....	298		
9.1.4 托管对象简介.....	298		
9.1.5 持久性存储简介.....	299		
9.1.6 获取请求简介.....	299		
9.2 在应用中使用 Core Data.....	299		
9.2.1 创建托管对象模型.....	300		
9.2.2 创建托管对象.....	302		
9.2.3 创建持久性存储.....	303		
9.2.4 设置 AppDelegate.....	304		
9.3 在应用中使用 Core Data.....	307		
9.3.1 使用托管对象.....	307		
9.3.2 获取托管对象.....	308		
9.3.3 使用关系.....	313		
9.3.4 了解模型更改.....	317		
9.4 针对性能进行调优.....	322		
9.4.1 优化保存.....	326		
9.4.2 配置托管对象上下文.....	327		
9.5 通过 Core Data 实现并发.....	328		
9.6 本章小结.....	330		
第 III 部分 集成应用			
第 10 章 通知	333		
10.1 实现本地通知.....	333		
10.1.1 了解本地通知.....	333		
10.1.2 创建通知.....	335		
10.1.3 接收通知.....	338		
10.2 了解推送通知.....	338		
10.2.1 配置开发者门户.....	340		
10.2.2 获取证书.....	343		
10.2.3 通过 Urban Airship 实现.....	345		
10.3 外部通知.....	348		
10.3.1 自定义 URL 模式.....	348		
10.3.2 响应 URL 请求.....	349		
10.4 本章小结.....	350		
第 11 章 发送电子邮件、SMS 和 拨打电话	351		
11.1 发送电子邮件.....	351		
11.1.1 撰写电子邮件.....	352		
11.1.2 使用附件.....	354		
11.2 发送 SMS(文本消息).....	354		
11.2.1 验证 SMS 是否可用.....	355		
11.2.2 撰写文本消息.....	355		
11.3 拨打电话号码.....	356		
11.4 本章小结.....	357		
第 12 章 了解 Address Book	359		
12.1 Address Book 框架简介.....	359		
12.2 访问 Address Book.....	360		
12.2.1 选择联系人.....	360		
12.2.2 请求访问权限.....	362		
12.2.3 显示并编辑联系人.....	365		
12.2.4 创建联系人.....	367		
12.2.5 删除联系人.....	369		
12.3 以编程方式访问 Address Book.....	370		
12.3.1 了解 Address Book.....	370		
12.3.2 了解记录.....	373		
12.3.3 了解属性.....	374		
12.3.4 以编程方式创建联系人.....	375		
12.3.5 以编程方式删除联系人.....	378		
12.4 本章小结.....	378		

第 13 章 事件编程	379		
13.1 Event Kit 框架简介	379		
13.2 使用 EventKitUI 框架	380		
13.2.1 请求访问权限	380		
13.2.2 访问日历	382		
13.2.3 创建和编辑日历事件	384		
13.3 以编程方式访问 Calendar 数据库	385		
13.3.1 创建事件	385		
13.3.2 编辑事件	390		
13.3.3 删除事件	390		
13.3.4 保持同步	390		
13.4 使用提醒	391		
13.4.1 创建提醒	391		
13.4.2 编辑提醒	392		
13.4.3 删除提醒	392		
13.4.4 使用警报	392		
13.5 本章小结	394		
第 14 章 与社交媒体集成	395		
14.1 社交媒体集成简介	395		
14.2 了解 Accounts 框架	396		
14.3 了解 Social 框架	400		
14.3.1 发帖	401		
14.3.2 检索推文	409		
14.4 与 Facebook 集成	411		
14.5 创建单点登录应用	418		
14.6 本章小结	423		
第IV部分 将应用运用到 生产环境中			
第 15 章 分析应用	427		
15.1 执行技术分析	427		
15.1.1 应用崩溃	428		
15.1.2 阻止主线程	428		
15.1.3 内存泄漏	429		
15.1.4 使用同步的 HTTP 请求	429		
15.1.5 广泛的带宽使用率	430		
15.1.6 电池消耗	434		
15.1.7 糟糕的用户界面	436		
15.2 执行商业分析	436		
15.3 本章小结	437		
第 16 章 从应用中获利	439		
16.1 获利简介	439		
16.1.1 付费应用	439		
16.1.2 广告	440		
16.1.3 In-App Purchases	440		
16.1.4 订阅	440		
16.1.5 潜在客户开发	440		
16.1.6 加盟销售	441		
16.2 开发 In-App Purchases	441		
16.2.1 In-App Purchase 简介	441		
16.2.2 注册产品	441		
16.2.3 选择产品类型	441		
16.2.4 了解 In-App Purchase 进程	442		
16.2.5 实现 In-App Purchase	443		
16.3 从广告中获利	464		
16.3.1 iAd 框架简介	464		
16.3.2 实现 AdMob 网络	467		
16.4 本章小结	470		
第 17 章 了解 iTunes Connect	471		
17.1 iOS 开发者会员中心	472		
17.1.1 获取开发者证书	472		
17.1.2 管理设备	476		
17.1.3 管理应用	479		
17.1.4 创建开发配置文件	484		
17.1.5 创建发布配置文件	488		
17.2 本章小结	491		
第 18 章 构建与发布	493		
18.1 App Store 审核	493		
18.1.1 了解审核指南	493		
18.1.2 了解审核流程	494		
18.1.3 了解拒绝状态	496		
18.1.4 避免各种常见误区	496		
18.2 为 Ad Hoc 发布构建应用	497		

18.2.1	构建应用	497	附录 A	音频代码	509
18.2.2	发布后进行测试.....	499	附录 B	图片尺寸	513
18.3	为 App Store 发布构建应用 ..	501			
18.4	本章小结	507			

第 I 部分

开发高级UI

- 第 1 章：创建 Personal Library
- 第 2 章：Tableview 进阶指南
- 第 3 章：Map Kit 框架
- 第 4 章：Action View 和 Alert 概述
- 第 5 章：国际化：为全世界构建应用
- 第 6 章：多媒体的使用



第 1 章

创建 Personal Library

本章内容:

- 创建注册和登录逻辑
- 配置应用设置
- 保存设置和保护密码安全
- 处理应用的崩溃

本章的 wrox.com 代码下载

可以通过 www.wrox.com/go/proiosprog 的 Download Code 选项卡下载本章的 wrox.com 代码。这些代码位于 Chapter 1 下载栏内并根据本章的内容独立命名。

本章将学习如何开发一个 Personal Library，这是一系列类和方法，你可以在项目中使用这些类并因此而领先一步。创建 Personal Library 会节省你在后续项目中投入的时间。在本章将要创建的这个 Personal Library，将教会你如何实现用户注册和用户登录的逻辑，以及如何保护密码存储的安全。

如果你之前开发过软件，很有可能会注意到在开发每个应用时都会重复进行大量冗余的工作。

为 iOS 系统开发应用时没有什么不同，因此，作为苹果开发者平台(Apple Developer Platform, ADP)的 Xcode 提供了一些各不相同的模板，我们称之为项目模板。这些项目模板存在的问题是，它们只提供主要对象的容器，但不提供在每个应用中所需要的立即可用的可配置特性。在本章中，开发属于自己的可配置的 Personal Library，这样就可以在所有应用中使用这个库了。在本书的其他章节中，可以通过添加附加功能来扩展这个 Personal Library。

1.1 创建 Personal Library

即将创建的这个 **Personal Library** 基本上是包含了多个不同的类和功能的项目骨架，它灵活而有组织地实现了冗余重复的工作。这个 **Personal Library** 中包含的功能有：

- 管理配置的设置选项
- 使用 `UIViewController` 类创建用户注册流程
- 使用 `UIViewController` 类实现用户登录流程
- 在 `Keychain` 中保存用户信息和安全的密码存储

在后续开发的项目中，仅需要将 **Personal Library** 库中需要的文件拖放在新项目下即可。

1.1.1 项目基础知识

在开发 iOS 应用时，当创建新项目时，需要配置一些基本选项。其中两个重要的选项是：

- 是否使用自动引用计数(Automatic Reference Counting, ARC)
- 是使用 Interface Builder 搭建 UI，还是在控制器(controller)类中编写代码创建 UI 组件

1. 自动引用计数(ARC)

ARC 是在 iOS 5.x 甚至更高版本中才引入的一项技术，它能够自动管理应用的内存分配。顾名思义，它会计算引用对象的次数，并且在需要时自动持有或释放对象。使用 ARC 的主要优点有：

- 不必将 `release` 或者 `autorelease` 消息发送给对象
- 在忘记 `release` 对象的情况下，减少出现内存泄漏的机会
- 可以少写代码，原因是可以跳过对象的释放，也可以在绝大多数情况下跳过 `dealloc` 方法的实现

保留 `dealloc` 方法的唯一原因在于，当需要释放不受 ARC 保护的资源时，需要调用 `dealloc` 方法，例如以下情形：

- 对 Core Foundation 框架生成的对象调用 `CFRelease` 方法
- 对使用 `malloc()`方法分配的内存调用 `free()`方法
- 取消定时器

当正在使用来自其他方的源代码和/或 `lib` 库时，必须保持头脑清醒，这些代码或库并非全部都已经更新并支持 ARC。如果需要使用一个不支持 ARC 的类，而且仍然想要在项目中使用 ARC，这时可以为出现问题的文件设置编译器标识，并赋值为 `-fno-objc-arc`，如图 1-1 所示。

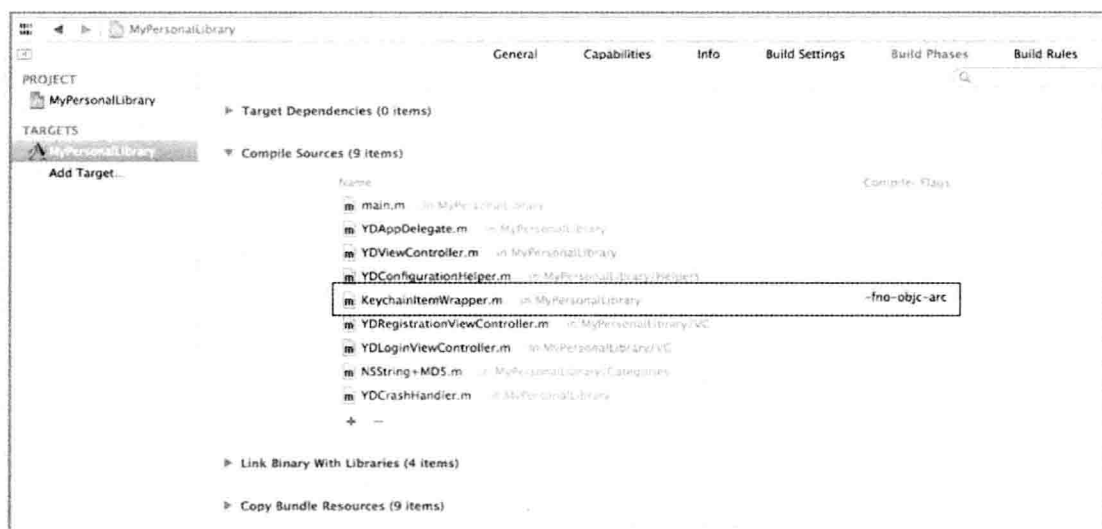


图 1-1

2. Interface Builder 工具

由于是一名经验丰富的程序员，因此你知道 Interface Builder 工具是 Xcode 的一部分，使用它能够在项目中创建和配置 UI 元素。就个人而言，我不喜欢使用 Interface Builder 工具，因为我喜欢通过编写代码的方式完全控制 UI 的各个方面。然而，副作用是，我的代码中有很很大一部分是 UI 相关的代码。在绝大部分例子中，为了保证与主题紧密相关的源代码清单引起足够的关注，会使用 Interface Builder 工具创建用户界面，使用 Assistant Editor 工具创建 IBAction 类型的方法和 IBOutlet 类型的变量。

1.1.2 启动新项目

启动 Xcode 开发环境，使用 Single View Application Project 模板创建一个新项目，使用图 1-2 所示的配置设置将其命名为 MyPersonalLibrary。

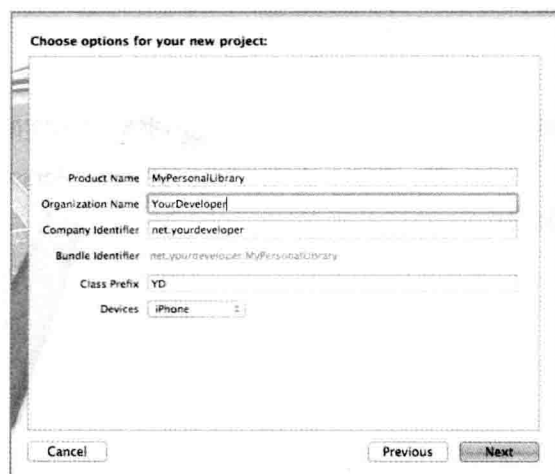


图 1-2

Class Prefix 字段将有助于遵循苹果的编码指导原则，因为项目中的每个类都需要有个独一无二的名字。当创建项目时，能够使用这个字段的值定义默认类名前缀，这个前缀会在创建新类时使用。尽管苹果的指导原则建议前缀为三个字符，但我通常使用两个字符作为前缀——我的公司名称 YourDeveloper 的缩写——所以我选择 YD 作为创建这些类的默认前缀。

当单击 Next 按钮时，会要求你选择保存这个项目的文件位置，在屏幕底部会看到一个复选框，这个复选框上面的提示文字是 Create Local git Repository for This Project，如图 1-3 所示。选中这个选项会创建本地 git 存储库，并单击 Create 按钮使用所选配置创建这个项目。

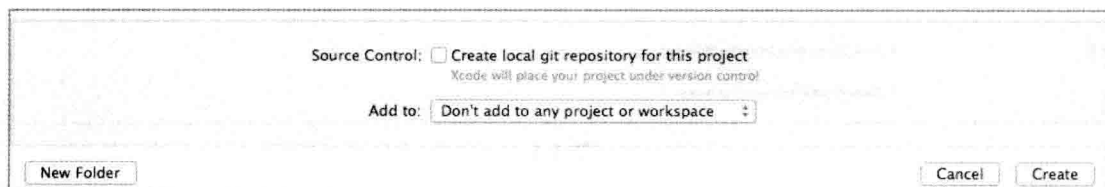


图 1-3

使用本地 git 存储库

当选择这个选项创建本地 git 存储库时，Xcode 工具会为你自动创建这个存储库，使你能够在项目中使用版本控制。使用版本控制后，Xcode 会对源代码的变更保持跟踪，使你能够回滚代码到某个版本，提交变更，比较不同版本之间的代码差异等。如果你正在使用基于 SVN(subversion system)的服务器管理源代码，那么可以使用 SVN 客户端或者 SVN 命令来管理源代码，因此不需要本地 git 存储库。如果你不使用基于 SVN 的服务器，强烈建议在创建新项目时选中 Create Local git Repository 这个选项。

在项目创建后，会在项目中看到两个类：YAppDelegate 类和 YDViewController 类。Xcode 开发工具会默认创建这两个类。也会看到 YDViewController.xib 文件，这个文件会被 Interface Builder 工具使用。

1.1.3 配置项目

一个好的实践是在项目目录结构中使用分组(group)来组织不同的元素，例如，images、sounds、classes、views、helpers 等。所以，开始时我们会在 MyPersonalLibrary 节点下创建一些分组。请创建以下分组：

- Externals
- Categories
- CrashHandler
- Helpers
- Definitions
- Images

- VC

由于分组仅仅是一个容器，并不是一个存在于文件系统上的物理文件夹，因此应该也在文件系统的项目根文件夹下创建这些文件夹。

由于在所开发的应用中有大量的重复工作，并且在不需要录制、复制和粘贴部分代码的情况下，你仍然需要灵活地打开或者关闭一些功能，因此可以按照如下的方式创建一个配置文件。

在 Project Explorer 窗口中，导航至 Definitions 这个分组，从上下文菜单选择 New File 选项。创建一个 C 头文件，如图 1-4 和图 1-5 所示，将其命名为 YDConstants.h。

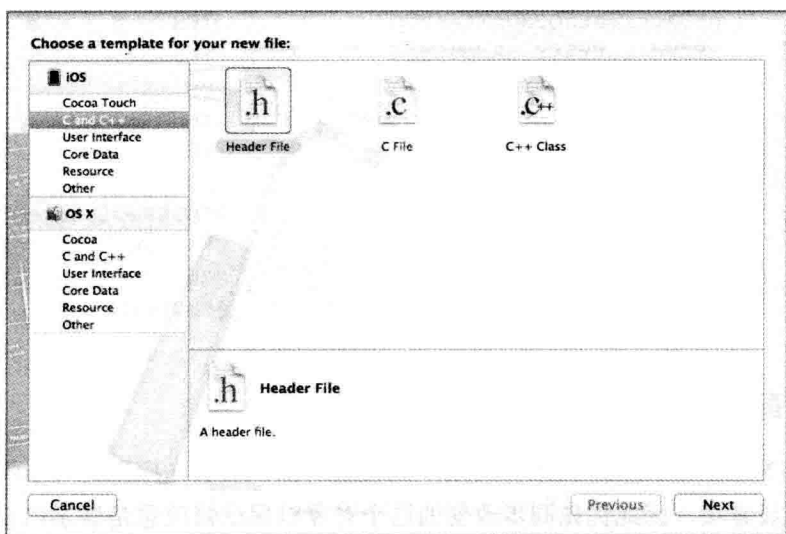


图 1-4

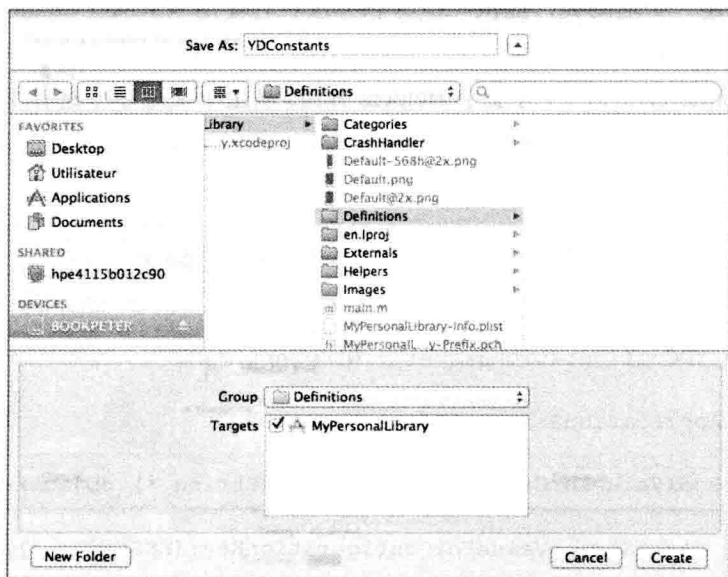


图 1-5

1.1.4 定义常量

正在开发的 **Personal Library** 这个库包含了用户注册、用户登录和一个设置控制器等功能。另外，还将会实现一个崩溃处理程序作为这个项目的补充。

在 **YDConstants.h** 文件中实现代码清单 1-1 中所示的代码。

代码清单 1-1 Chapter1/MyPersonalLibrary/YDConstants.h

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

// My application switches
#define bYDActivateGPSONStartUp YES
#define bYDRegistrationRequired YES
#define bYDLoginRequired NO
#define bYDShowLoginAfterRegistration YES
#define bYDInstallCrashHandler YES
//keys that are used to store data
#define bYDRegistered @"bYDRegistered"
#define bYDAuthenticated @"bYDAuthenticated"
#define bYDFirstLaunch @"bYDFirstLaunch"
#define bYDVibrate @"bYDVibrate"
```

1.1.5 使用配置文件

可以使用 **NSUserDefaults** 类来保存用户的设置。由于能够在应用的许多不同地方保存或者改变个别设置项，因此优先同步改变的这个对象以保证修改总是展示出正确的值，这一点就变得很重要。

下一步，创建一个辅助类，提供一些静态方法。它使用 **NSUserDefaults** 类读写设置项的值。

在 **Project Explorer** 窗口中，导航至 **Helpers** 分组，从上下文菜单中选择 **File | New** 选项。创建一个新的 **Objective-C** 类，继承于 **NSObject** 类，命名为 **YDConfigurationHelper**，如代码清单 1-2 中所示。

代码清单 1-2 Chapter1/MyPersonalLibrary/YDConfigurationHelper.h

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

@interface YDConfigurationHelper : NSObject

+ (void)setApplicationStartupDefaults;

+ (BOOL)getBoolValueForConfigurationKey:(NSString *)_objectkey;

+ (NSString *)getStringValueForConfigurationKey:(NSString *)_objectkey;

+ (void)setBoolValueForConfigurationKey:(NSString *)_objectkey withValue:(BOOL)_boolvalue;
```

```

+ (void) setStringValueForConfigurationKey: (NSString *)
_objectkey withValue: (NSString *) _value;

@end

```

在代码清单 1-3 中, YDConfigurationHelper 类包含了一些静态的方法, 能够帮助你访问 NSUserDefaults 对象。它包含对 NSString 类型的值和 BOOL 类型的值的获取(get)和设置(set)的方法, 这使操作变得简单很多。

代码清单 1-3 Chapter1/MyPersonalLibrary/YDConfigurationHelper.m

```

#import "YDConfigurationHelper.h"

@implementation YDConfigurationHelper
+ (void) setApplicationStartupDefaults
{
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    [defaults synchronize];
    [defaults setBool:NO forKey:byYDFirstLaunch];
    [defaults setBool:NO forKey:byYDAuthenticated];
    [defaults synchronize];
}

+ (BOOL) getBoolValueForConfigurationKey: (NSString *) _objectkey
{
    //create an instance of NSUserDefaults
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    [defaults synchronize]; //let's make sure the object is synchronized
    return [defaults boolForKey:_objectkey];
}

+ (NSString *) getStringValueForConfigurationKey: (NSString *) _objectkey
{
    //create an instance of NSUserDefaults
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    [defaults synchronize]; //let's make sure the object is synchronized
    if ([defaults stringForKey:_objectkey] == nil )
    {
        //I don't want a (null) returned
        return @"";
    }
    else
    {
        return [defaults stringForKey:_objectkey];
    }
}

+ (void) setBoolValueForConfigurationKey: (NSString *)
_objectkey withValue: (BOOL) _boolvalue

```

```

{
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    [defaults synchronize]; //let's make sure the object is synchronized
    [defaults setBool:_boolvalue forKey:_objectkey];
    [defaults synchronize]; //make sure you're synchronized again
}

+ (void) setStringValueForConfigurationKey: (NSString *)
    _objectkey withValue: (NSString *) _value
{
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    [defaults synchronize]; //let's make sure the object is synchronized
    [defaults setValue:_value forKey:_objectkey];
    [defaults synchronize]; //make sure you're synchronized again
}

@end

```

当 `getStringValueForConfigurationKey:` 方法尝试传入一个空指针时，它会返回一个空的 `NSString` 而不是这个空指针。这样做的原因是，假如你想要获取一个值，并且把它赋值给 `UILabel` 对象的 `text` 属性时，你不会想在这个控件上显示出 `(null)` 字样。

1.1.6 导入头文件

如你所知，当想要在应用中使用含有变量定义的头文件时，需要使用 `#import` “YDConstants.h” 这个语句将其导入。

在每一个后续的和 `ViewController` 类中重复使用这个语句导入头文件，这实在是不方便。不过，每一个应用还存在一个预编译的头文件，它在整个应用中都是可用的。可以在 `Project Explorer` 窗口的 `Supporting Files` 分组下找到这个文件，名为 `MyPersonalLibrary-Prefix.pch`。

如果在这个文件中导入头文件，那么它就是全局可用的。在此为 `YDConstants` 和 `YDConfigurationHelper` 这两个头文件添加一个导入语句，如代码清单 1-4 中所示。

代码清单 1-4 Chapter1/MyPersonalLibrary/MyPersonalLibrary-Prefix.pch

```

#import <Availability.h>
#ifdef __OBJC__
    #import "YDConstants.h"
    #import "YDConfigurationHelper.h"
    #import <UIKit/UIKit.h>
    #import <Foundation/Foundation.h>
#endif

```

在下一节中，你将创建登录、注册以及设置的 `ViewController` 类。

1.2 注册——登录

许多应用需要用户注册和登录，或者仅仅是登录。如果在应用的外部处理用户管理，则可以要求用户使用不同的方式进行注册。可以要求得到像 e-mail 地址或者用户名及密码那样的凭证，也可以——现在可以经常看到——要求用户通过 Facebook 账号注册。

有关 Facebook 注册的内容将在第 14 章“集成社交媒体”中全面地讲述。在该章中，将会扩展 Personal Library。

iOS Keychain Services(iOS 密钥链服务)介绍

iOS Keychain Services 提供了针对用户设备上的密码、密钥、证书、笔记和自定义数据的安全存储的解决方案。

在 Personal Library 中，会使用 iOS Keychain Services 保存在后续开发的注册流程中用户输入的密码。

为了与 iOS Keychain Services 进行的交互更加易用，苹果公司已经发布了 KeyChainItemWrapper 类，这个类会提供一个更高级别的 API 来使用 keychain。

可以从下面的网址下载需要在 Personal Library 项目中包含的 KeyChainItemWrapper 类的示例项目：

http://developer.apple.com/library/ios/#samplecode/GenericKeychain/Listings/Classes_KeyChainItemWrapper_h.html。

从上面的 URL 下载这个示例项目，使用 Finder 导航至它的文件。在 Xcode 项目中，使用 Project Explorer 窗口，导航至 Externals 分组，从上下文菜单中选择 New Group 选项，并将它命名为 KeyChain。在文件系统上的 externals 文件夹下创建一个名为 keychain 的文件夹。在下载的文件夹中，以拖放动作(drag-and-drop)的方式将 KeyChainItemWrapper.h 和 KeyChainItemWrapper.m 两个文件复制到项目文件夹下。Xcode 开发工具会给出提示，提示你对添加这些文件的选项进行选定。如图 1-6 所示，请做出与之相同的选择。

KeyChainItemWrapper 类没有以使用 ARC 的方式开发，因此你必须按照之前图 1-1 所示的那样，为 KeyChainItemWrapper.m 文件设置 -fno -objc -arc 编译器标记。

关于 Keychain Services 编程的更多信息，请访问 <https://developer.apple.com/library/ios/#documentation/security/Conceptual/keychainServConcepts/01introduction/introduction.html>。

为了能够通过运用 KeyChainItemWrapper 类来使用 Keychain Services，还需要为其添加一个指向 Security.framework 的引用。

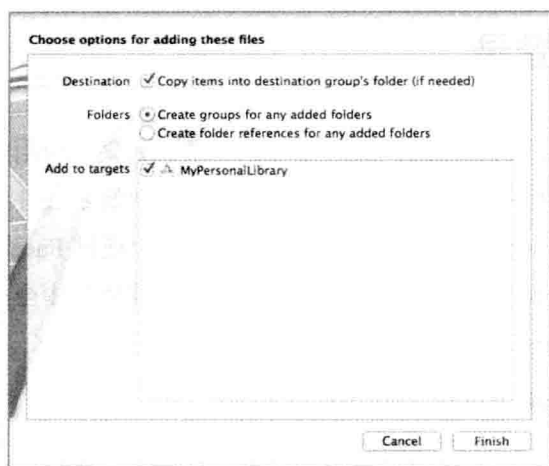


图 1-6

1.2.1 创建注册逻辑

因为每一个应用的设计将是不相同的，所以在其中的用户注册界面看起来可能并不相同；尽管如此，仍可以把注册逻辑中的一部分标准化，原因在于你总是需要遵循相同的步骤：

- 验证这个用户是否之前已经注册过，因此不必再次呈现注册视图。
- 如果这个用户之前已经注册过，规定仍需要哪些后续流程；例如，呈现登录视图或者从 Web 服务中加载数据。

你已经在 `YDConstants` 类的头文件中定义了名称为 `bYDRegistrationRequired` 的一个常量，可以在应用委托中使用这个常量检查是否需要呈现一个 `ViewController` 类来捕获注册凭证。

使用 `Project Explorer` 窗口，导航至 `VC` 分组，从上下文菜单中选择 `New File` 选项来创建 `UIViewController` 类的子类，命名为 `YDRegistrationViewController`，如图 1-7 所示。



图 1-7

在 YDRegistrationViewController 类中，需要设立一个委托，这个委托会将注册流程的结果通知给委托的原型。使用 Interface Builder 工具和 Assistant Editor 工具创建用户界面，这个界面包含一个 UILabel 控件、两个 UITextField 控件，还有两个 UIButton 对象。YDRegistrationViewController.xib 文件看起来将会如图 1-8 所示。

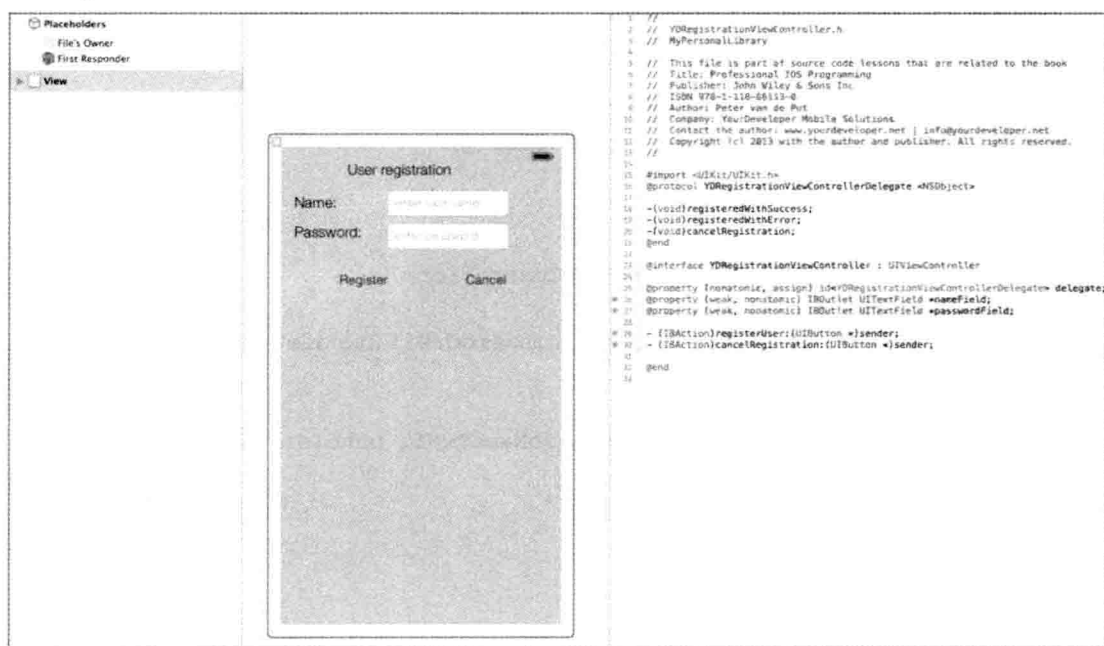


图 1-8

YDRegistrationViewController.h 的代码如代码清单 1-5 中所示。

代码清单 1-5 Chapter1/MyPersonalLibrary/YDRegistrationViewController.h

```
#import <UIKit/UIKit.h>
@protocol YDRegistrationViewControllerDelegate <NSObject>

- (void)registeredWithSuccess;
- (void)registeredWithError;
- (void)cancelRegistration;
@end

@interface YDRegistrationViewController : UIViewController

@property (nonatomic, assign) id<YDRegistrationViewControllerDelegate> delegate;
@property (weak, nonatomic) IBOutlet UITextField *nameField;
@property (weak, nonatomic) IBOutlet UITextField *passwordField;

- (IBAction)registerUser:(UIButton *)sender;
- (IBAction)cancelRegistration:(UIButton *)sender;

@end
```

如代码清单 1-6 所示，在 YDRegistrationViewController.m 文件中，实现 registerUser: 和 cancelRegistration: 方法。

registerUser: 方法先将输入的密码哈希成一个 MD5 字符串，然后将其保存到 keychain 中。

代码清单 1-6 Chapter1/MyPersonalLibrary/YDRegistrationViewController.m

```
#import "YDRegistrationViewController.h"
#import "NSString+MD5.h"
#import "KeychainItemWrapper.h"
@interface YDRegistrationViewController ()

@end

@implementation YDRegistrationViewController
@synthesize delegate;
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

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

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (IBAction)registerUser:(UIButton *)sender
{
    if ([self.nameField.text length] == 0 ||
        [self.passwordField.text length] == 0)
    {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:
            @"Error" message:@"Both fields are mandatory"
            delegate:self cancelButtonTitle:@"Ok"
            otherButtonTitles:nil, nil];
        [alert show];
    }
    else

```



```

{
    KeychainItemWrapper* keychain = [[KeychainItemWrapper alloc]
        initWithIdentifier:@"YDAPPNAME" accessGroup:nil];
    [keychain setObject:self.nameField.text
        forKey:(__bridge id)kSecAttrAccount];
    [keychain setObject:[self.passwordField.text MD5]
        forKey:(__bridge id)kSecValueData];
    //reading back a value from the keychain for comparison
    //get username [keychain objectForKey:(__bridge id)kSecAttrAccount]];
    //get password [keychain objectForKey:(__bridge id)kSecValueData]];
    [YDConfigurationHelper setBoolValueForConfigurationKey:
        bYDRegistered withValue:YES];
    [self.delegate registeredWithSuccess];
    //or
    //[self.delegate registeredWithError];
}
}

- (IBAction)cancelRegistration:(UIButton *)sender
{
    [self.delegate cancelRegistration];
}

@end

```

1.2.2 初始化数据

每个应用需要一些默认的设置，这些设置在应用启动的过程中都是可用的。因此，在本节中，你将学会如何初始化应用的默认设置。

1.2.3 初始化应用的默认设置

到目前为止，你一直在创建 **Personal Library**，它包含了可重用、可配置的代码，因此你不必在每一个应用中编写相同的代码。凭借定义的 **YDConfigurationHelper** 类，你一直使用 **NSUserDefaults** 类来保存设置选项和操作结果。

应用第一次启动时，这些设置和操作的值都没有进行设置。为了支持默认的设置进行初始化，**YDConfigurationHelper** 类有一个名叫 **setApplicationStartupDefaults** 的静态方法。在这个方法的实现中，设置了每一个属性的初始值，从而在应用的开始启动阶段控制应用的逻辑，如下面的这个例子：

```

+ (void)setApplicationStartupDefaults
{
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    [defaults synchronize];
    [defaults setBool:NO forKey:bYDFirstLaunch];
    [defaults setBool:NO forKey:bYDAuthenticated];
}

```

```
[defaults synchronize];
}
```

如果上一次的安装残留了一些 `keychain` 条目，那么你可能也想删除它们。而且，`NSUserDefaults` 类也是一个理想的地方，比方说，如果需要创建某些处于应用包之外的特殊目录，那么在应用的进一步处理中，不必每一次都测试目录是否存在。

`bYDFirstLaunch` 键的目的是为了支持这样一种事实：这段初始化代码仅被调用一次。

下面的代码片段说明了应该如何在 `YDAppDelegate` 类中调用 `application:didFinishLaunchingWithOptions:` 方法。它会检测如果存储的值返回 `YES`，就调用 `setApplicationStartupDefaults` 方法，这个方法将值设为 `NO`。

```
if (![YDConfigurationHelper
getBoolValueForConfigurationKey:bYDFirstLaunch])
    [YDConfigurationHelper setApplicationStartupDefaults];
```

1.2.4 创建登录逻辑

用户现在能够注册，这的确很好，但是也需要支持这样一个事实：已经注册的用户能够登录。

作为第一个步骤，首先创建一个新类，作为输入用户名和密码的 `ViewController` 类。创建一个新的继承于 `UIViewController` 类的 `YDLoginViewController` 类，编写类似于代码清单 1-7 中的代码。像在 `YDRegistrationViewController` 文件中那样，定义一个协议，使其能够被委托使用，因此登录流程的结果会被通知给这个委托对象。依据登录流程的不同结果，三个方法会分别被调用。

代码清单 1-7 Chapter1/MyPersonalLibrary/YDLoginViewController.h

```
#import <UIKit/UIKit.h>
@protocol YDLoginViewControllerDelegate <NSObject>

-(void)loginWithSuccess;
-(void)loginWithError;
-(void)loginCancelled;
@end

@interface YDLoginViewController : UIViewController

@property (weak, nonatomic) IBOutlet UITextField *nameField;
@property (weak, nonatomic) IBOutlet UITextField *passwordField;

@property (nonatomic, assign) id<YDLoginViewControllerDelegate> delegate;

- (IBAction)loginUser:(UIButton *)sender;
- (IBAction)cancelLogin:(UIButton *)sender;

@end
```

YDLoginViewController 类的实现方式如代码清单 1-8 所示。

代码清单 1-8 Chapter1/MyPersonalLibrary/YDLoginViewController.m

```
#import "YDLoginViewController.h"
#import "YDLoginViewController.h"
#import "NSString+MD5.h"
#import "KeychainItemWrapper.h"
@interface YDLoginViewController ()

@end

@implementation YDLoginViewController
@synthesize delegate;
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

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

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (IBAction)loginUser:(UIButton *)sender
{
    if (([self.nameField.text length] == 0) ||
        ([self.passwordField.text length] == 0))
    {
        [self showErrorWithMessage:@"Both fields are mandatory!"];
    }
    else
    {
        KeychainItemWrapper* keychain = [[KeychainItemWrapper alloc]
            initWithIdentifier:@"YDAPPNAME" accessGroup:nil];
        if ([self.nameField.text isEqualToString:
            [keychain objectForKey:(__bridge id)kSecAttrAccount]])
        {
            if ([self.passwordField.text MD5]
```

```

        isEqualToString:[keychain objectForKey:
            (__bridge id)kSecValueData]])
    {
        [self.delegate loginWithSuccess];
    }
    else
        [self showErrorWithMessage:@"Password not correct."];
    }
    else
        [self showErrorWithMessage:@"Name not correct."];
    }
}

- (IBAction)cancelLogin:(UIButton *)sender
{
    [self.delegate loginCancelled];
}

- (void)showErrorWithMessage:(NSString *)msg
{
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Error"
        message:msg delegate:self
        cancelButtonTitle:@"Ok"
        otherButtonTitles:nil, nil];

    [alert show];
}

@end

```

在 `loginUser:` 方法中, 实现的第一项检查是, 两个输入框是否已经填入数据。下一步, 逻辑会检查用户名是否保存过, 如果保存过, 检查在 `keychain` 中保存的密码是否与输入的密码一致。

`YDLoginViewController.xib` 文件应该看起来如图 1-9 所示。

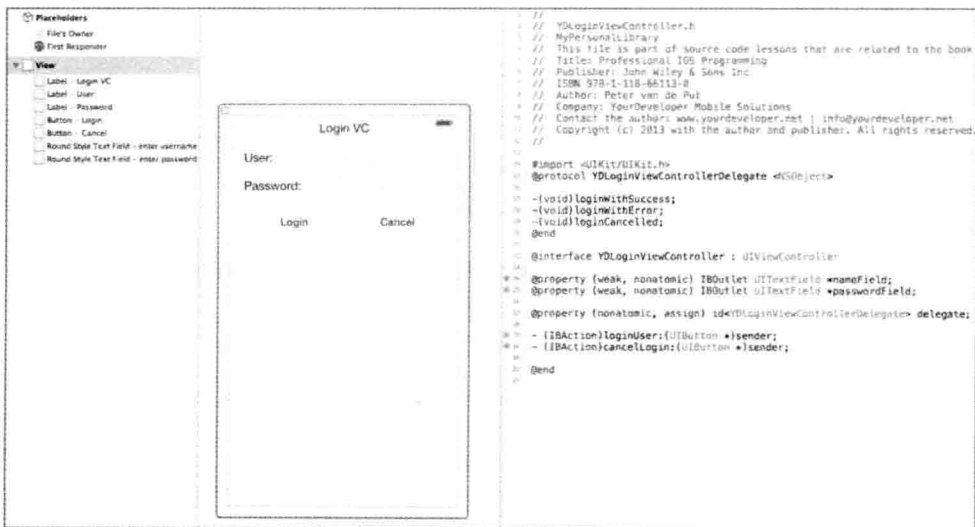


图 1-9

1.2.5 保护密码的安全

在 `YDRegistrationViewController` 类中，接受值作为用户名和密码，并将它保存在设备的 `keychain` 中。许多开发者仍然使用 `NSUserDefaults` 类来保存用户名和密码，这不是一个安全的解决方案。

这并非是一个非常安全的解决方案，原因在于密码以明文保存，而且 `NSUserDefaults` 对象并不是 100% 的安全，它能够被人为破解。

为了使 `Personal Library` 更安全，实现一个 `NSString` 的分类，为密码创建一个 MD5 的哈希值，并将其保存在 `keychain` 中；`keychain` 是在设备中保存关键数据的唯一安全的地方。

在 `Project Explorer` 窗口中，导航至 `Categories` 分组，在上下文菜单中选择 `New File` 选项。在如图 1-10 所示的模板列表中选择 `Objective-C Category` 这一项。

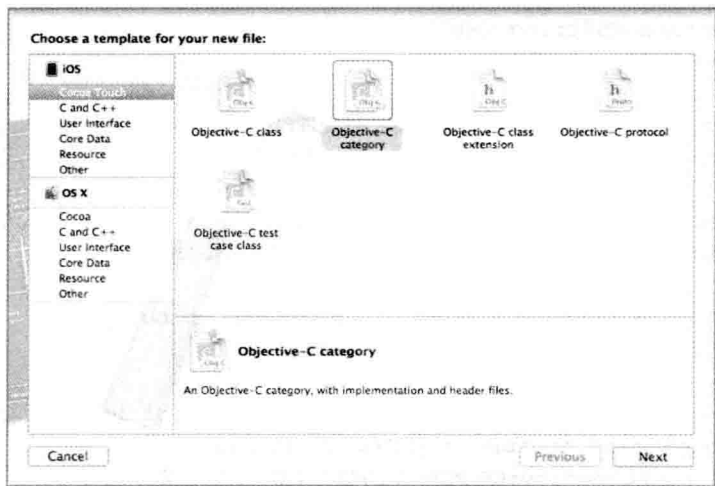


图 1-10

单击 `Next` 按钮，在下一个界面中输入 `MD5` 作为正在创建的这个分类的名字。从 `Category On` 下拉菜单中选择 `NSString`，如图 1-11 所示。

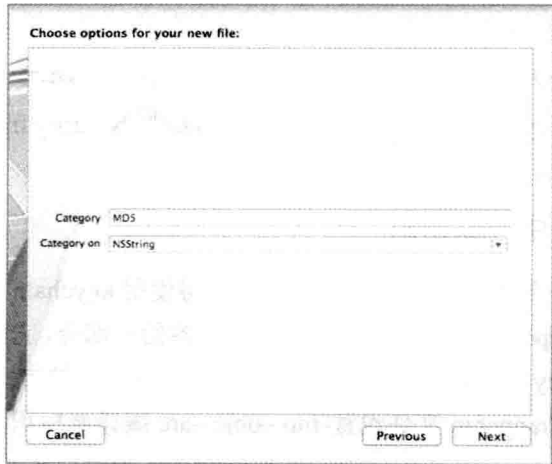


图 1-11

头文件如代码清单 1-9 中所示。

代码清单 1-9 Chapter1/MyPersonalLibrary/NSString+MD5 头文件

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

```
@interface NSString (MD5)
```

```
- (NSString *)MD5;
```

```
@end
```

分类的实现如代码清单 1-10 中所示，说明文字位于代码的注释中。

代码清单 1-10 Chapter1/MyPersonalLibrary/NSString+MD5 implementation

```
#import "NSString+MD5.h"
```

```
#import <CommonCrypto/CommonDigest.h>
```

```
@implementation NSString (MD5)
```

```
- (NSString*)MD5
```

```
{
```

```
    // Create pointer to the string as UTF8
```

```
    const char *ptr = [self UTF8String];
```

```
    // Create byte array of unsigned chars
```

```
    unsigned char md5Buffer[CC_MD5_DIGEST_LENGTH];
```

```
    // Create 16 bytes MD5 hash value, store in buffer
```

```
    CC_MD5(ptr, strlen(ptr), md5Buffer);
```

```
    // Convert unsigned char buffer to NSString of hex values
```

```
    NSMutableString *output = [NSMutableString  
        stringWithCapacity:CC_MD5_DIGEST_LENGTH * 2];
```

```
    for(int i = 0; i < CC_MD5_DIGEST_LENGTH; i++)
```

```
        [output appendFormat:@"%02x",md5Buffer[i]];
```

```
    return output;
```

```
}
```

```
@end
```

在这一分类中，导入了<CommonCrypto/CommonDigest.h>头文件。为了能够导入这个头文件，如果在添加 KeychainItemWrapper 类时没有添加 Security.framework 的链接，那么必须将其添加到项目中。

1.2.6 在 keychain 中存储密码

苹果公司开发了一个 keychain 的包装类，这使得使用 keychain 变得简单。

KeychainItemWrapper 等相关类是本书的下载内容的一部分，它是直接从苹果公司的开发者门户上拿来的。KeychainItemWrapper 类没有以使用 ARC 的方式开发，基于这个原因，必须为 KeychainItemWrapper.m 文件设置-fno-objc-arc 编译器标识，如之前图 1-1 中所示。

1.3 崩溃管理

尽管经过了所有的编程努力和广泛的测试流程，但是应用可能仍然会崩溃。一个应用出现崩溃可以存在许多不同的原因，而且有时它可能与应用完全没有关联。在本节中，你会了解到发生崩溃的种类，以及如何实现有关崩溃处理程序的内容。这个崩溃处理程序恰巧在应用崩溃前，显示带有标准提示消息的 `UIAlertView` 控件，因此用户知道这个应用正要崩溃，从而减轻不快的感觉。

应用崩溃的两个主要原因是：

- 内存泄漏
- 对象的过度释放

在大多数情况下，没有释放对象会引起内存泄漏。`UIImageView`、`UIImage`、`UIWebView` 等对象会消耗大量的内存；在合适的时间，一定要释放这些对象的内存。

当试图访问一个已经被释放的对象时会出现过度释放的情况。

使用 Instruments 工具

Instruments 工具是针对 iOS 和 OS X 的代码进行动态追踪和测试的一个性能、分析和检测的工具。Instruments 工具可以帮助你识别内存泄漏，帮助你从内部观察内存消耗，以及帮助你进行许多其他性能相关的测量等。通过这些方式，有助于你提高代码质量。对 Instruments 工具，以及它如何在上述方面给予帮助，有一个很好的理解是很重要的。请访问 <http://developer.apple.com/library/ios/#documentation/DeveloperTools/Conceptual/Instruments-UserGuide/Introduction/Introduction.html> 获取关于 Instruments 工具的完整文档。

可以使用 Instruments 工具并分析应用来识别内存泄漏。应该使用 `Zombie` 这个仪器来识别对已经释放的对象(它被称为僵尸)的访问。

1.3.1 理解崩溃

在运行时，假如一个应用崩溃了，它的信号处理功能默认可以启动 6 种不同的标准信号，这些信号是：

- `SIGABRT`：异常终止
- `SIGFPE`：浮点异常
- `SIGILL`：无效指令
- `SIGINT`：应用收到交互式关注请求
- `SIGSEGV`：访问到无效的内存地址
- `SIGTERM`：终止请求发送给应用

在下一节中，将构建一个全局的异常处理程序，捕捉这些信号并呈现 `UIAlertView` 控件以通知用户。

1.3.2 实现崩溃处理程序

创建一个名称为 YDCrashHandler 的新类，如代码清单 1-11 中所示。

代码清单 1-11 Chapter1/MyPersonalLibrary/YDCrashHandler.h

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

@interface YDCrashHandler : NSObject
{
    BOOL dismissed;
}
void InstallCrashExceptionHandler();
@end
```

类的实现如代码清单 1-12 中所示。

这个类为每一个可用的信号都安装了 `NSSetUncaughtExceptionHandler` 处理程序。如果出现异常，会调用这个处理程序并访问当前 `RunLoop` 下的全部模式。进程会被杀掉，并且 `backtrace`(译者注：堆栈的回溯信息)被读入到一个 `NSDictionary` 类的对象中，并且传递给 `handleException` 方法，这个方法呈现了一个 `UIAlertView` 控件。

代码清单 1-12 Chapter1/MyPersonalLibrary/YDCrashHandler.m

```
#import "YDCrashHandler.h"
#include <libkern/OSAtomic.h>
#include <execinfo.h>

NSString * const YDCrashHandlerSignalExceptionName =
    @"YDCrashHandlerSignalExceptionName";
NSString * const YDCrashHandlerSignalKey = @"YDCrashHandlerSignalKey";
NSString * const YDCrashHandlerAddressesKey =
    @"YDCrashHandlerAddressesKey";

volatile int32_t UncaughtExceptionCount = 0;
const int32_t UncaughtExceptionMaximum = 10;

const NSInteger UncaughtExceptionHandlerSkipAddressCount = 4;
const NSInteger UncaughtExceptionHandlerReportAddressCount = 5;
@implementation YDCrashHandler
+ (NSArray *)backtrace
{
    void* callstack[128];
    int frames = backtrace(callstack, 128);
    char **strs = backtrace_symbols(callstack, frames);

    int i;
    NSMutableArray *backtrace = [NSMutableArray arrayWithCapacity:frames];
    for (
        i = UncaughtExceptionHandlerSkipAddressCount;
        i < UncaughtExceptionHandlerSkipAddressCount +
```



```

        UncaughtExceptionHandlerReportAddressCount;
        i++)
    {
        [backtrace addObject:[NSString stringWithUTF8String:strs[i]]];
    }
    free(strs);

    return backtrace;
}

- (void)alertView:(UIAlertView *)alertView
clickedButtonAtIndex:(NSInteger)anIndex
{
    //if (anIndex == 0)
    //{
        dismissed = YES;
    //}
}

- (void)handleException:(NSEException *)exception
{
    UIAlertView *thisAlert = [[UIAlertView alloc] initWithTitle:@"Sorry"
        message:@"An unexpected event happened causing the application to
        shutdown." delegate:nil cancelButtonTitle:@"Ok"
        otherButtonTitles:nil, nil];

    [thisAlert show];

    CFRunLoopRef runLoop = CFRunLoopGetCurrent();
    CFArrayRef allModes = CFRunLoopCopyAllModes(runLoop);

    while (!dismissed)
    {
        for (NSString *mode in (NSArray *)CFBridgingRelease(allModes))
        {
            CFRunLoopRunInMode((CFStringRef)CFBridgingRetain(mode), 0.001, false);
        }
    }

    CFRelease(allModes);

    NSSetUncaughtExceptionHandler(NULL);
    signal(SIGABRT, SIG_DFL);
    signal(SIGILL, SIG_DFL);
    signal(SIGSEGV, SIG_DFL);
    signal(SIGFPE, SIG_DFL);
    signal(SIGBUS, SIG_DFL);

```

```

    signal(SIGPIPE, SIG_DFL);

    if ([[exception name] isEqual:YDCrashHandlerSignalExceptionName])
    {
        kill(getpid(), [[exception userInfo]
            objectForKey:YDCrashHandlerSignalKey] intValue]);
    }
    else
    {
        [exception raise];
    }
}

@end

void HandleException(NSException *exception)
{
    int32_t exceptionCount = OSAAtomicIncrement32(&UncaughtExceptionCount);
    if (exceptionCount > UncaughtExceptionMaximum)
    {
        return;
    }

    NSArray *callStack = [YDCrashHandler backtrace];
    NSMutableDictionary *userInfo =
    [NSMutableDictionary dictionaryWithDictionary:[exception userInfo]];
    [userInfo
        setObject:callStack
        forKey:YDCrashHandlerAddressesKey];

    [[[YDCrashHandler alloc] init]
        performSelectorOnMainThread:@selector(handleException:)
        withObject:
        [NSException
            exceptionWithName:[exception name]
            reason:[exception reason]
            userInfo:userInfo]
        waitUntilDone:YES];
}

void SignalHandler(int signal)
{
    int32_t exceptionCount = OSAAtomicIncrement32(&UncaughtExceptionCount);
    if (exceptionCount > UncaughtExceptionMaximum)
    {
        return;
    }

    NSMutableDictionary *userInfo =
    [NSMutableDictionary

```

```

        dictionaryWithObject:[NSNumber numberWithInt:signal]
        forKey:YDCrashHandlerSignalKey];

    NSArray *callStack = [YDCrashHandler backtrace];
    [userInfo
     setObject:callStack
     forKey:YDCrashHandlerAddressesKey];

    [[[YDCrashHandler alloc] init]
     performSelectorOnMainThread:@selector(handleException:)
     withObject:
     [NSException
      exceptionWithName:YDCrashHandlerSignalExceptionName
      reason:
      [NSString stringWithFormat:@"Signal %d was raised.", signal]
      userInfo:
      [NSDictionary
       dictionaryWithObject:[NSNumber numberWithInt:signal]
       forKey:YDCrashHandlerSignalKey]]
     waitUntilDone:YES];
}

void InstallCrashExceptionHandler()
{
    NSSetUncaughtExceptionHandler(&HandleException);
    signal(SIGABRT, SignalHandler);
    signal(SIGILL, SignalHandler);
    signal(SIGSEGV, SignalHandler);
    signal(SIGFPE, SignalHandler);
    signal(SIGBUS, SignalHandler);
    signal(SIGPIPE, SignalHandler);
}

```

Personal Library 现在已经准备就绪可供使用了。打开 YAppDelegate.h 文件并实现代码，如代码清单 1-13 中所示。

代码清单 1-13 Chapter1/MyPersonalLibrary/YAppDelegate.h

```

#import <UIKit/UIKit.h>
#import "YDRegistrationViewController.h"
#import "YDLoginViewController.h"

@class YDViewController;
@interface YAppDelegate : UIResponder <UIApplicationDelegate,
    YDLoginViewControllerDelegate,
    YDRegistrationViewControllerDelegate>

@property (strong, nonatomic) UIWindow *window;
@property (strong, nonatomic) YDViewController *viewController;
@property (strong, nonatomic) YDLoginViewController *loginVC;

```

```
@property (strong, nonatomic) YDRegistrationViewController *registrationVC;

@end
```

打开 YDAppDelegate.m 实现文件并写入代码，如代码清单 1-14 中所示。

application: didFinishLaunchingWithOptions: 方法首先检查是否需要安装 YDCrashHandler 对象。下一步，检查这个应用是否是首次运行，如果是，则使用 YDConfigurationHelper 类设置应用的默认启动参数。

下一步，它继续执行逻辑检测是否需要注册。如果之前的检测结果返回 true，就会实施第二项检测，看看之前是否执行过用户注册。如果需要完成注册，就会呈现 YDRegistrationViewController 界面，同时委托的实现逻辑会检查是否需要呈现 YDLoginViewController 界面。在登录逻辑成功执行后，会呈现 YDMainViewController 界面。完整的实现代码如代码清单 1-14 中所示。

代码清单 1-14 Chapter1/MyPersonalLibrary/YDAppDelegate.m

```
#import "YDAppDelegate.h"

#import "YDViewController.h"
#import "YDCrashHandler.h"
@implementation YDAppDelegate

- (void)installYDCrashHandler
{
    InstallCrashExceptionHandler();
}

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions
{
    if (bYDInstallCrashHandler)
    {
        [self performSelector:@selector(installYDCrashHandler)
        withObject:nil afterDelay:0];
    }

    self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen]
    bounds]];

    if (![YDConfigurationHelper getBoolValueForConfigurationKey:bYDFirstLaunch])
        [YDConfigurationHelper setApplicationStartupDefaults];

    if (bYDActivateGPSONStartup)
    {
        //Start your CLLocationManager here if you're application needs the GPS
    }
}
```

```

if (bYDRegistrationRequired && ![YDConfigurationHelper
    getBoolValueForConfigurationKey:bYDRegistered])
{
    //Create an instance of your RegistrationViewController
    self.registrationVC = [[YDRegistrationViewController alloc] init];
    //Set the delegate
    self.registrationVC.delegate = self;
    self.window = [[UIWindow alloc] initWithFrame:
        [[UIScreen mainScreen] bounds]];
    self.window.rootViewController = _registrationVC;
    self.window.backgroundColor = [UIColor clearColor];
    [self.window makeKeyAndVisible];
}
else
{
    // you arrive here if either the registration is not required
    // or yet achieved
    if (bYDLoginRequired)
    {
        self.loginVC = [[YDLoginViewController alloc] init];
        self.loginVC.delegate = self;
        self.window = [[UIWindow alloc] initWithFrame:
            [[UIScreen mainScreen] bounds]];
        self.window.rootViewController = _loginVC;
        self.window.backgroundColor = [UIColor clearColor];
        [self.window makeKeyAndVisible];
    }
    else
    {
        self.viewController = [[YDViewController alloc] init];
        self.window.rootViewController = self.viewController;
        [self.window makeKeyAndVisible];
    }
}

#pragma Registration Delegates
- (void)registeredWithError
{
    //called from RegistrationViewController if registration failed
}
- (void)registeredWithSuccess
{
    //called from RegistrationViewController if the registration with success
    //
    if (bYDShowLoginAfterRegistration)
    {

```

```
self.loginVC = [[YDLoginViewController alloc] init];
self.loginVC.delegate=self;
self.window = [[UIWindow alloc] initWithFrame:
               [[UIScreen mainScreen] bounds]];
self.window.rootViewController = self.loginVC;
self.window.backgroundColor = [UIColor clearColor];
[self.window makeKeyAndVisible];
}
else
{
    self.viewController= [[YDViewController alloc] init];
    self.window.rootViewController =self.viewController;
    [self.window makeKeyAndVisible];
}
}
-(void)cancelRegistration
{
    //called from RegistrationViewcontroller if cancel is pressed
}
#pragma Login delegates
-(void)loginWithSuccess
{
    //called when login with success
    self.viewController= [[YDViewController alloc] init];
    self.window.rootViewController =self.viewController;
    [self.window makeKeyAndVisible];
}
-(void)loginWithError
{
    //called when login with error
}
-(void)loginCancelled
{
    //called when login is cancelled
}

@end
```

1.4 本章小结

在本章中，创建了一个可配置的 **Personal Library**，能够使用它作为后续应用的起始点。这个 **Personal Library** 支持以下特性：

- 配置设置的唯一存放位置
- 带有 **ViewController** 类的注册流程
- 带有 **ViewController** 类的登录流程
- 在 **User Defaults** 中存储多个值

- 使用 MD5 加密算法和 keychain 存储使密码存储安全
- Settings ViewController 类，允许使用复选框获取和存储应用的设置选项
- 崩溃处理程序，避免应用中未预期的崩溃

在下一章中，你将学会如何使用 UITableView 类相关的对象来增强应用，并且根据需要定制这些对象。在已经理解了如何使用 UITableView 类相关对象的基本知识后，你将开发一个定制的 UITableView 控件的解决方案，从而引出一个聊天视图控制器，就像 iMessage 程序中使用的那个视图一样。最终，你会开发出一个定制的 UITableView 控件，它支持下拉功能以增强用户体验。

第 2 章

Tableview 进阶指南

本章内容:

- 学习如何进阶使用 UITableView, 带给应用更高级的观感(look and feel)
- 学习如何开发自己定制的 UITableView 类, 模仿 iMessage 应用的观感
- 为一个基于分组的 UITableView 实现下钻逻辑

本章的 wrox.com 代码下载

可以通过 www.wrox.com/go/proiosprog 的 Download Code 选项卡下载本章的 wrox.com 代码。这些代码位于 Chapter 2 下载栏内并根据本章的内容独立命名。

在 iOS 应用中呈现数据时, UITableView 可能是最经常使用的用户界面对象。在本章中,将学习到以超越标准实现的方式使用 UITableView,并理解 UITableView 类的工作方式。你会创建一个聊天视图控制器,它支持定制的单元格和灵活的行高,以及下钻功能的实现,能够将多个对象的多个分类进行分组,从而生成一个高级的用户界面。最后,你会为表格视图的实现添加搜索功能。

2.1 理解 UITableView

UITableView 直接继承于 UIScrollView 类,从而给它带来直向(译者注:横向和纵向)滚动的能力。当想要使用 UITableView 时,必须首先创建 UITableView 类的实例,将它指向 UIView 控件而使其可见,并且建立一个 datasource 对象和一个负责与 UITableView 进行交互的 delegate 对象。

2.1.1 datasource 和 delegate

每一个 UITableView 都需要 datasource 和 delegate 这两个对象。datasource 对象为 UITableView 提供数据。通常，datasource 对象使用 NSArray 类或者 NSDictionary 类在内部存储数据，并且根据需要将数据提供给表视图。delegate 对象必须实现 UITableViewDelegate 和 UITableViewDataSource 这两个协议。

UITableViewDelegate 协议定义了几个方法，delegate 对象需要实现其中至少三个方法。delegate 对象必须实现的方法有：

- tableView:numberOfRowsInSection:
- numberOfSectionsInTableView:
- tableView:cellForRowAtIndexPath:

启动 Xcode 开发环境，使用 Single View Application Project 模板创建新项目，并使用如图 2-1 中所示的配置将其命名为 PlainTable。

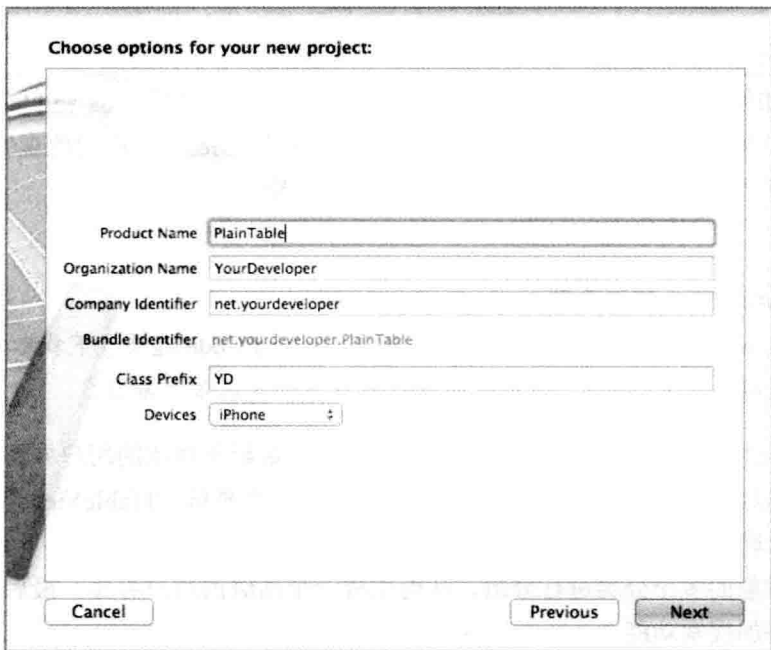


图 2-1

使用 Interface Builder 工具打开 YDViewController.xib 文件，并将一个 UITableView 控件添加到该窗口中。使用 Assistant Editor 工具为这个 UITableView 控件创建一个属性。也需要设置 Referencing Outlets 一栏中的 datasource 和 delegate 指向 UITableView 对象。确保 YDViewController.xib 文件看起来如图 2-2 中所示。

打开 YDViewController.h 文件，创建名为 rowData 的 NSMutableArray 对象充当 datasource，如代码清单 2-1 中所示。



图 2-2

代码清单 2-1 Chapter2/PlainTable/YDViewController.h

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

@interface YDViewController : UIViewController

@property (weak, nonatomic) IBOutlet UITableView *mTableView;
@property (nonatomic, strong) NSMutableArray* rowData;

@end
```

打开 YDViewController.m 文件，实现如代码清单 2-2 中所示的代码，关于这段代码，会在代码清单后详细说明。

代码清单 2-2 Chapter2/PlainTable/YDViewController.m

```
#import "YDViewController.h"

@interface YDViewController ()

@end

@implementation YDViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    [self loadData];
}

- (void)loadData
{

```

```
if (self.rowData!=nil)
{
    [self.rowData removeAllObjects];
    self.rowData=nil;
}

self.rowData = [[NSMutableArray alloc] init];
for (int i=0 ; i<100;i++)
{
    [self.rowData addObject:[NSString stringWithFormat:@"Row: %i",i]];
}

//now my datasource if populated let's reload the tableview
[self.mTableView reloadData];
}

#pragma mark UITableView delegate
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section {

    return [self.rowData count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = (UITableViewCell *)[tableView
        dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
            reuseIdentifier:CellIdentifier];
    }
    cell.selectionStyle = UITableViewCellSelectionStyleNone;
    cell.textLabel.text = [self.rowData objectAtIndex:indexPath.row];
    return cell;
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:
    (NSIndexPath *)indexPath
{
    [tableView deselectRowAtIndexPath:indexPath animated:YES];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

下面对这段代码进行分解，向你解释代码中各方法的作用。

在 `viewDidLoad` 方法中，调用了本地方法 `loadData`，该方法创建了一个带有 100 个记录的 `NSMutableArray` 对象，并将 `reloadData` 消息发送给 `self.mTableView` 对象。

`reloadData` 方法迫使 `mTableView` 对象通过调用 `delegate` 方法重新加载数据，并更新用户界面。

在 `#pragma mark UITableView delegate` 标记语句之后，需要实现表视图运行所必须的 `delegate` 对象的最小方法集合。

调用 `numberOfSectionsInTableView`：这个 `delegate` 方法来决定 `UITableView` 控件的 `section` 的数量。如果使用 `UITableViewStylePlain` 风格，`UITableView` 控件的 `section` 数通常是 1。后面将会学习到带有下钻功能的例子，如果使用例子中那种风格的 `section`，则需要返回实际的 `section` 的数量。

当渲染单元格时，会调用 `tableView:cellForRowAtIndexPath`：这个 `delegate` 方法。这个方法恰好是布局 `UITableViewCell` 的地方(`UITableView` 中的一行)。现在，先简单地创建一个 `UITableViewCell`，如果单元格仍然可用的话，试着在内存中重用它。

为了显示 `rowData` 数组中的正确的行，需要将 `[rowData objectAtIndex:indexPath.row]` 方法的返回值赋给 `cell.textLabel.text` 属性。

当用户以单击某行的方式选择该行时，会调用 `tableView:didSelectRowAtIndexPath`：这个 `delegate` 方法。`deselectRowAtIndexPath:animated:` 的 `delegate` 方法会取消这一行的选择，因此单元格不会保持高亮的状态。

如果想要保持选择状态仍然可见，那么请省略这行代码。

当应用运行时，结果如图 2-3 中所示。

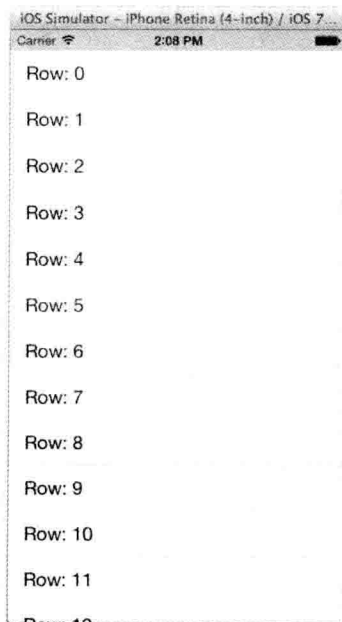


图 2-3

2.1.2 滚动

由于 UITableView 对象继承于 UIScrollView 类，因此它本身拥有完全的滚动功能。然而，在某些情况下，例如在 UITableView 中添加一个新行，或者删除一行时，可能要直接滚动到 UITableView 中的某个位置。

可以通过调用 UITableView 的 `scrollToRowAtIndexPath:atScrollPosition:animated:` 方法，获得 UITableView 上基于代码的滚动效果。这个方法传入的第一个参数是 NSIndexPath 类型的对象。NSIndexPath 对象表示到嵌套数组集合树上的某一特定节点的路径。这个路径称为索引路径。在 iOS 应用中，用 NSIndexPath 对象来确定到表格视图内的行和 section 的路径。调用 NSIndexPath 类的 `indexPathForRow:inSection:` 方法，传入行和 section 的索引数字，通过这种方式可以创建 NSIndexPath 的实例。

启动 Xcode 开发环境，使用 Single View Application Project 模板创建一个新项目，并使用如图 2-4 中所示的配置，将其命名为 ScrollingTable。

使用 Interface Builder 工具打开 YDViewController.xib 文件，创建一个用户界面，如图 2-5 中所示。

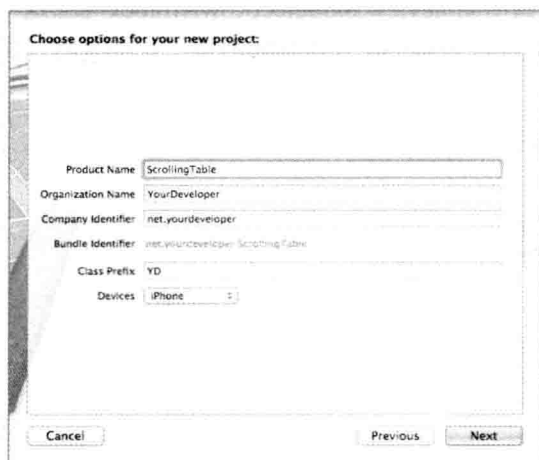


图 2-4

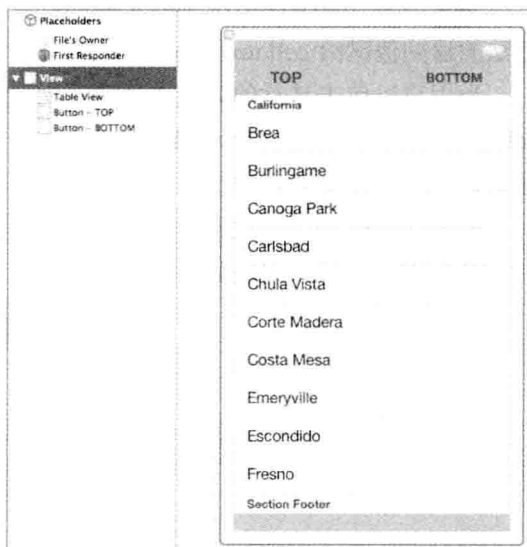


图 2-5

如代码清单 2-3 中所示，建立 YDViewController.h 文件。作为前一个例子的补充，为引入的两个 UIButton 添加两个动作。

代码清单 2-3 Chapter2/ScrollingTable/YDViewController.h

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

@interface YDViewController : UIViewController

@property (weak, nonatomic) IBOutlet UITableView *mTableView;
@property (nonatomic, strong) NSMutableArray* rowData;
```

```

- (IBAction)scrollToTop:(UIButton *)sender;
- (IBAction)scrollToBottom:(UIButton *)sender;

@end

```

YDViewController.m 文件的实现与前面的代码清单 2-2 中的类似，唯一的区别在于此时 scrollToTop:和 scrollToBottom:这两个方法的实现如代码清单 2-4 中所示。

代码清单 2-4 Chapter2/ScrollingTable/YDViewController.m

```

#import "YDViewController.h"

@interface YDViewController ()

@end

@implementation YDViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    [self loadData];
}

- (void)loadData
{
    if (self.rowData!=nil)
    {
        [self.rowData removeAllObjects];
        self.rowData=nil;
    }

    self.rowData = [[NSMutableArray alloc] init];
    for (int i=0 ; i<100;i++)
    {
        [self.rowData addObject:[NSString stringWithFormat:@"Row: %i",i]];
    }

    //now my datasource if populated let's reload the tableview
    [self.mTableView reloadData];
}

#pragma mark UITableView delegates
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section {

```

```

        return [self.rowData count];
    }
- (UITableViewCell *)tableView:(UITableView *)tableView
  cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = (UITableViewCell *)[tableView
      dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
          reuseIdentifier:CellIdentifier];
    }
    cell.selectionStyle = UITableViewCellSelectionStyleNone;
    cell.textLabel.text = [self.rowData objectAtIndex:indexPath.row];
    return cell;
}
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:
  (NSIndexPath *)indexPath
{
    [tableView deselectRowAtIndexPath:indexPath animated:YES];
}
- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (IBAction)scrollToTop:(UIButton *)sender
{
    NSIndexPath *topRow = [NSIndexPath indexPathForRow:0 inSection:0];
    [self.mTableView scrollToRowAtIndexPath:topRow
      atScrollPosition:UITableViewScrollPositionTop animated:YES];
}

- (IBAction)scrollToBottom:(UIButton *)sender
{
    NSIndexPath *bottomRow = [NSIndexPath indexPathForRow:
      [self.rowData count]-1 inSection:0];
    [self.mTableView scrollToRowAtIndexPath:bottomRow
      atScrollPosition:UITableViewScrollPositionBottom animated:YES];
}
@end

```

在 `scrollToTop:` 方法中, 创建一个 `NSIndexPath` 对象的实例, 把 `indexPathForRow` 的值置为 0, 可以将表视图滚动至顶部。在 `scrollToBottom:` 方法中, 使用 `[self.rowData count]-1` 的值创建 `NSIndexPath` 实例, 可以将表视图滚动至底部。

本书仅提供部分阅读，如需完整版，请联系QQ: 461573687

提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

备用QQ:2404062482