

The iPhone Developer's Cookbook
Building Applications with the iPhone SDK

iPhone开发秘籍

[美] Erica Sadun 著
漆振 解巧云 郎亚妹 等译

- Amazon超级畅销书
- 大量未公开的绝技，带你深入iPhone开发秘境
- 任务驱动，丰富的实战代码，让你触类旁通

CocoaChina、dojo中国、WeiPhone三大网站联袂推荐



人民邮电出版社
POSTS & TELECOM PRESS

“难以想象，竟然有人还没有读Erica Sadun的书就开始从事iPhone商业开发……本书将使你成为AppStore上用户热捧的那种五星级开发者。”
——著名IT媒体InfoWorld

“本书全面剖析了iPhone开发的方方面面，将成为各层次开发人员的求生秘籍。丰富的代码示例和屏幕截图更为书中大量技巧和诀窍添色不少。”
——Jacqui Cheng，著名IT网站Ars Technica副主编

The iPhone Developer's Cookbook Building Applications with the iPhone SDK

iPhone开发秘籍

iPhone和iPod touch吸引的不只是全世界数千万痴迷的用户，它们也为成千上万锐意创新的程序员提供了一个平台，使他们得以为未来构建最酷最炫的软件应用。

本书由业界著名的iPhone程序员撰写，针对iPhone / iPod touch开发流程中的常见任务和难点逐一提供解决方案，并给出深入的剖析。书中讲解了构建视图、组织界面元素、响应用户操作和访问地址簿，还介绍了如何连接到因特网、显示媒体内容和创建安全密钥链等。尤其难能可贵的是，作者还探讨了Cover Flow等未公开的API，使你能够创建更加令人赏心悦目的用户体验。

本书各项主题按相关性归类，编排合理，便于查找，既适合学习也适合参考。书中示例代码贴近开发实际，可以很方便地应用于实战，节省大量开发时间。

作者为本书开设了配套网站ericasadun.com，其中提供了代码下载、开发视频下载和最新更新。

Erica Sadun 著名iPhone技术专家、佐治亚理工学院计算机科学博士，拥有多年Apple开发经验。参与过二十多部畅销图书的编写，领域涉及程序设计、数字视频、数码摄像等。

延伸阅读

iPhone开发基础教程 (978-7-115-19733-7, 65.00元)

Objective-C基础教程

iPhone开发实战



Addison
Wesley

www.informit.com

本书相关信息请访问：图灵网站 <http://www.turingbook.com>

读者/作者热线：(010)51095186

反馈/投稿/推荐信箱：contact@turingbook.com

分类建议 计算机/程序设计/移动开发

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-20763-0



9 787115 207630 >

ISBN 978-7-115-20763-0/TP

定价：65.00元

TURING 图灵程序设计丛书 移动开发系列

The iPhone Developer's Cookbook
Building Applications with the iPhone SDK

iPhone开发秘籍

[美] Erica Sadun 著
漆振 解巧云 郎亚妹 等译

人民邮电出版社
北京



图书在版编目 (CIP) 数据

iPhone 开发秘籍 / (美) 萨丹 (Sadun, E.) 著; 漆振等译. —北京: 人民邮电出版社, 2009.7

(图灵程序设计丛书)

书名原文: The iPhone Developer's Cookbook: Building Applications with the iPhone SDK

ISBN 978-7-115-20763-0

I. i… II. ①萨…②漆… III. 移动通信-携带电话机-应用程序-程序设计 IV. TN929.53

中国版本图书馆CIP数据核字 (2009) 第057751号

内 容 提 要

本书提供了关于 iPhone SDK 和 iPhone 开发的全面信息, 覆盖面广、结构清晰。书中深入浅出地介绍 iPhone SDK 中的各种组件, 同时详细和通俗地讲解了 iPhone 开发的基本流程、基本原理和基本原则, 还结合实例讲述了如何使用各种 iPhone 特性, 包括视图、视图控制器、警告、表格、媒体、控件、人物、位置和事件等。

本书适合各层次的 iPhone 开发人员学习和参考。书中的示例具有很强的通用性, 特别适合作为参考指南。

图灵程序设计丛书

iPhone开发秘籍

-
- ◆ 著 [美] Erica Sadun
 - 译 漆 振 解巧云 郎亚妹 等
 - 责任编辑 傅志红
 - 执行编辑 谢灵芝
 - ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
 - 邮编 100061 电子函件 315@ptpress.com.cn
 - 网址 <http://www.ptpress.com.cn>
 - 北京隆昌伟业印刷有限公司印刷
 - ◆ 开本: 800×1000 1/16
 - 印张: 18.5
 - 字数: 437千字
 - 印数: 1-4 000册
 - 2009年7月第1版
 - 2009年7月北京第1次印刷
 - 著作权合同登记号 图字: 01-2009-1526号
 - ISBN 978-7-115-20763-0/TP
-

定价: 65.00元

读者服务热线: (010)51095186 印装质量热线: (010)67129223

反盗版热线: (010)67171154

版 权 声 明

Authorized translation from the English language edition, entitled *The iPhone Developer's Cookbook: Building Applications with the iPhone SDK*, 978-0-321-55545-8 by Erica Sadun, published by Pearson Education, Inc., publishing as Addison-Wesley Professional, Copyright © 2009 by Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by PEARSON EDUCATION ASIA LTD. and POSTS & TELECOM PRESS Copyright © 2009.

本书中文简体字版由Pearson Education Asia Ltd.授权人民邮电出版社独家出版。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。

本书封面贴有Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。版权所有，侵权必究。



**本书献给我最爱的丈夫Alberto，感谢他多年来一直容忍我谈论太多
的小工具和SDK，始终亲切耐心地陪伴着我。**

对本书的赞誉

“本书物超所值！如果你正在编写iPhone软件，它将为你节省数周的开发时间。Erica在书中包含了大量简单易懂的示例，演示了基本的iPhone开发技巧，并通过其他一些技巧展示了苹果公司官方文档所未表现的特殊效果。”

——Tim Burks, TootSweet软件公司的iPhone软件开发人员

“Erica Sadun的技术经验与Addison-Wesley的品牌交相辉映。本书全面剖析了iPhone开发的方方面面，将成为各层次开发人员的求生秘籍。丰富的代码示例和屏幕截图更为书中大量技巧和诀窍添色不少。”

——Jacqui Cheng, 著名IT网站*Ars Technica*副主编

“我深深折服于Erica掌控主题和传达信息的能力：文风亲切，又有技术深度。这将是iPhone开发人员的必备图书。”

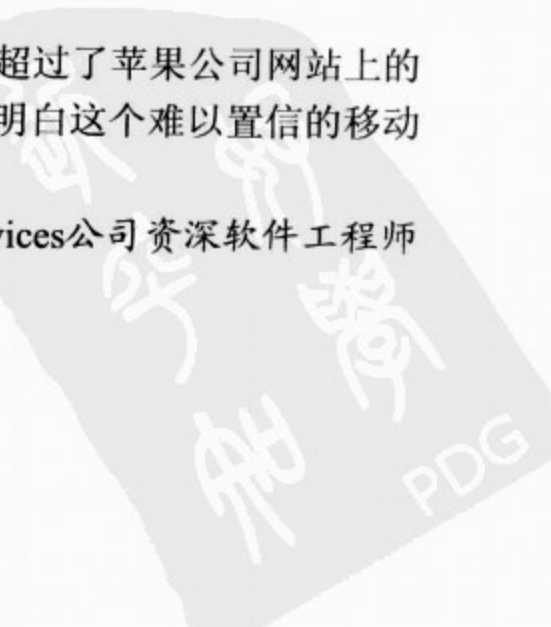
——Daniel Pasco, Black Pixel Luminance公司首席开发人员和首席执行官

“本书应是iPhone编程入门的首选，它是对苹果公司自身文档的绝佳补充。”

——Alex C. Schaefer, MeLLmo公司首席程序员, ApolloIM和iPhone应用开发专家

“Erica的书确实是Cocoa Touch开发人员的绝佳读物。本书内容远远超过了苹果公司网站上的文档，并且通过各种方法让开发人员更深入地理解iPhone OS，让他们明白这个难以置信的移动平台内究竟有哪些不为人知的秘密。”

——John Zorko, Mobile Devices公司资深软件工程师



译者序

iPhone SDK是苹果公司提供的一种软件开发工具包，它允许第三方为iPhone和iPod touch创建应用程序。由于苹果公司不能满足所有人在iPhone上的应用需求，因而该软件包意义重大，它为开发人员提供了一整套丰富的应用程序编程接口和工具。开发人员可以使用iPhone SDK轻松地创建iPhone和iPod touch应用程序。其简易的操作方式和强大的功能也为开发人员带来了无限可能。

本书向开发新手讲解最典型、最通用的iPhone开发最佳实践。全书采用理论与实践相结合的方式，配以详尽的说明，指导读者创建一系列应用程序，在实践中理解iPhone应用程序的运行方式和构建方式，并掌握具体的iPhone特性以及如何控制这些特性和与其交互。

全书共11章，大致可分为3个部分。前6章介绍iPhone开发中经常用到的一些基本组件，包括视图、视图控制器、警告和表格。第7章至第9章深入介绍如何利用各种高级iPhone特性，其中包括媒体、控件、人物、位置和事件。最后两章介绍了连接服务和高级的Cover Flow编程技巧。

iPhone SDK是一个新兴的软件开发平台，且目前国内关于iPhone开发的图书较少，iPhone开发也有待普及。因此，本书的出版可以为iPhone开发起到推波助澜的作用。

本书由漆振、解巧云、郎亚妹、张波、杨越、张莉、吴爱金和盛海艳等翻译和审校，在翻译过程中，得到了图灵公司的帮助和支持，在此一并致谢。但由于译者知识有限，加之时间仓促，译文中难免出现一些疏漏，诚恳地希望广大读者给予批评指正。

译者
2009年3月



前言

很少有开发平台能够与特立独行的iPhone相提并论。iPhone将基于OS X的移动计算与创新的多点触摸屏幕、位置感知、机载加速计等结合在一起。苹果公司于2008年3月初发布iPhone Cocoa Touch SDK Beta版时，开发人员的热烈反应让苹果公司的服务器忙得不可开交。在不到一周的时间内，SDK就被下载了10万多次。本书就是在此背景下应运而生的。

本书的目标读者

本书面向即将开发实际项目但对SDK全然不知的新iPhone开发人员。虽然每位程序员对此有不同的目标，且经验不同，但大多数人都需要解决其开发工作中类似的任务：如何构建表格，如何创建安全的密钥链条目，如何搜索地址簿，如何在视图之间切换，以及如何使用Core Location。

本书针对的正是刚开始接触iPhone编程的学习者。通过清晰、完备的示例，读者可以迅速起步并进行有效的开发。书中给出了经过测试且可行的解决方案，程序员可以专注于应用程序的具体部分，而不必为样本式的任务劳神。

本书的组织结构

针对iPhone开发新手面对的大多数常见问题：布置界面元素，响应用户操作，访问本地数据源以及连接到因特网，本书逐一提供了解决方案。书中按示例进行组织，因此代码拿来就可以使用，非常方便。程序员可以在自己的项目中使用书中的源代码，然后根据需要进行定制。每章将相关的任务归类在一起，读者可以直接跳到所查问题并寻找解决方案，而不需要琢磨哪个类或框架最适合当前问题。

下面概述了本书中各章的内容。

第1章：iPhone SDK简介

这一章介绍iPhone SDK并将iPhone作为交付平台进行研究，包括平台的限制等。它对标准iPhone应用程序进行了细分，并指导你构建第一个Hello World样式的示例。

第2章：视图

这一章介绍屏幕上的iPhone视图及对象。你将了解如何对视图进行布置、创建和排序，以创建iPhone应用程序主干；还会了解视图层次结构、几何方法和动画，以及用户如何通过触摸与视

目 录

第 1 章 iPhone SDK 简介	1
1.1 苹果公司的 iPhone SDK	1
1.2 组建 iPhone 项目	2
1.3 iPhone 应用程序组件	3
1.3.1 应用程序文件夹层次结构	3
1.3.2 可执行文件	3
1.3.3 Info.plist 文件	4
1.3.4 图标和默认图像	5
1.3.5 XIB (NIB) 文件	5
1.3.6 应用程序束中不存在的文件	5
1.3.7 沙盒	6
1.4 平台限制	6
1.4.1 存储限制	6
1.4.2 数据访问限制	7
1.4.3 内存限制	7
1.4.4 交互限制	7
1.4.5 电量限制	7
1.4.6 应用程序限制	8
1.4.7 用户行为限制	8
1.5 SDK 限制	8
1.6 编程范型	9
1.6.1 面向对象编程	9
1.6.2 模型-视图-控制器	9
1.7 构建 iPhone 应用程序主干	15
1.8 Hello World 应用程序	15
1.8.1 类	17
1.8.2 代码	18
1.8.3 关于示例代码和内存管理的 注意事项	18
1.9 构建 Hello World 应用程序	19
1.9.1 创建 iPhone 项目	19
1.9.2 运行主干	20
1.9.3 定制 iPhone 项目	20
1.9.4 编辑标识信息	21
1.9.5 使用调试器	21
1.10 苹果公司的 iPhone 开发人员计划	23
1.10.1 开发电话	23
1.10.2 应用程序标识符	23
1.11 从 Xcode 到 iPhone: Organizer 界面	24
1.11.1 PROJECTS & SOURCES 列表	25
1.11.2 DEVICES 列表	25
1.11.3 Summary 选项卡	25
1.11.4 Console 选项卡	25
1.11.5 Crash Logs 选项卡	25
1.11.6 Screenshot 选项卡	25
1.11.7 线缆连接	26
1.11.8 在 iPhone 上测试应用程序	26
1.11.9 编译以分发	27
1.12 使用文档中未记录的 API 调用	28
1.13 临时分发	28
1.14 小结	29
第 2 章 视图	30
2.1 UIView 和 UIWindow	30
2.1.1 层次结构	30
2.1.2 几何特征	31
2.1.3 手势	34
2.2 秘诀: 添加递进式子视图	34
2.3 秘诀: 拖动视图	36
2.3.1 UITouch	37
2.3.2 添加持久性	39
2.4 秘诀: 剪辑视图	42
2.4.1 通过剪辑平衡触摸	43

2.4.2 访问逐个像素值	43	4.2 秘诀: 创建多行按钮显示	90
2.5 秘诀: 检查多点触摸	45	4.3 秘诀: 自动计时的无按钮警告	91
2.6 UIView 动画	48	4.4 秘诀: 请求用户的文本输入	92
2.7 秘诀: 淡入和淡出视图	49	4.5 秘诀: 显示简单菜单	94
2.8 秘诀: 交换视图	50	4.6 “请稍候”: 向用户显示进度	95
2.9 秘诀: 翻转视图	52	4.7 秘诀: 调用基本的文档中未记录的 UIProgressHUD	95
2.10 秘诀: 将 CATransition 应用于层	54	4.8 秘诀: 使用 UIActivity- IndicatorView	97
2.10.1 文档中未记录的动画类型	54	4.9 秘诀: 构建 UIProgressView	98
2.10.2 通用 Core Animation 调用	56	4.10 秘诀: 添加自定义、可轻击的 覆盖层	101
2.11 秘诀: 滑动视图	57	4.11 秘诀: 构建下滑式警告	104
2.12 秘诀: 转换视图	59	4.12 秘诀: 添加状态栏图像	107
2.13 小结	61	4.13 添加应用程序标记	108
第3章 视图控制器	63	4.14 秘诀: 简单的音频警告	110
3.1 视图管理	63	4.15 小结	112
3.1.1 核心类	63	第5章 基本表格	113
3.1.2 专用类	64	5.1 UITableView 和 UITableView- Controller 简介	113
3.1.3 创建 UINavigationController	64	5.1.1 创建表格	113
3.2 使用 Interface Builder 为 UIView- Controller 构建视图	66	5.1.2 UINavigationController 的 作用	115
3.2.1 温度转换器示例	66	5.2 秘诀: 创建简单的列表表格	115
3.2.2 直接加载 XIB 文件	73	5.2.1 数据源函数	116
3.3 导航控制器	73	5.2.2 重用单元格	116
3.3.1 设置导航控制器	74	5.2.3 字体表格示例	116
3.3.2 推入和弹出视图控制器	74	5.3 秘诀: 创建基于表格的选择表	118
3.3.3 导航项类	75	5.4 秘诀: 将图像加载到表格单元中	122
3.4 秘诀: 构建简单的双项菜单	75	5.5 秘诀: 设置单元格的文本特性	123
3.5 秘诀: 添加分段控件	77	5.6 秘诀: 删除单元格选择	124
3.6 秘诀: 在导航栏中添加 UIToolbar	79	5.7 秘诀: 创建复杂的单元格	125
3.7 秘诀: 在视图控制器之间导航	81	5.8 秘诀: 创建选中的选择	127
3.7.1 返回根	83	5.9 秘诀: 删除单元格	128
3.7.2 加载视图控制器数组	83	5.9.1 创建和显示删除控件	130
3.8 选项卡栏	83	5.9.2 关闭删除控件	131
3.9 小结	86	5.9.3 处理删除请求	131
第4章 警告用户	87	5.9.4 滑动单元格	131
4.1 通过警告直接与用户对话	87	5.9.5 添加单元格	131
4.1.1 记录结果	88		
4.1.2 构建警告	88		
4.1.3 显示警告	89		

5.10 秘诀: 对单元格重新排序.....	131	8.2 秘诀: 向按钮添加动画元素.....	194
5.11 秘诀: 使用公开.....	132	8.3 秘诀: 为按钮响应制作动画效果.....	196
5.12 小结.....	134	8.4 秘诀: 定制开关.....	197
第6章 高级表格	135	8.5 秘诀: 添加自定义滑块缩略图.....	200
6.1 秘诀: 对表格选择进行分组.....	135	8.6 秘诀: 关闭 UITextField 键盘.....	204
6.1.1 构建基于部分的数据源.....	139	8.7 秘诀: 关闭 UITextView 键盘.....	205
6.1.2 添加部分标题.....	139	8.8 秘诀: 向文本视图添加一个撤销 (Undo) 按钮.....	207
6.2 秘诀: 构建带索引的部分表格.....	140	8.9 秘诀: 创建一个基于文本视图的 HTML 编辑器.....	209
6.3 秘诀: 定制单元格背景.....	141	8.10 秘诀: 构建一个交互搜索栏.....	211
6.4 秘诀: 创建蓝白交替的单元格.....	145	8.11 秘诀: 添加标注 (callout) 视图.....	213
6.5 秘诀: 设置表格边框.....	146	8.12 添加一个页面指示器控件.....	216
6.6 秘诀: 添加耦合的单元格控件.....	148	8.13 秘诀: 定制工具栏.....	218
6.7 秘诀: 构建多滚轮表格.....	150	8.14 小结.....	221
6.8 秘诀: 使用 UIDatePicker.....	153	第9章 人物、地点和事件	223
6.9 秘诀: 创建完全自定义的分组表格.....	155	9.1 地址簿框架.....	223
6.10 小结.....	160	9.1.1 Address Book UI.....	223
第7章 媒体	161	9.1.2 Address Book.....	224
7.1 秘诀: 按照文件类型浏览 Documents 文件夹.....	161	9.2 秘诀: 访问地址簿图像数据.....	225
7.2 加载和查看图像.....	163	9.3 秘诀: 显示地址簿信息.....	227
7.3 秘诀: 显示小图像.....	164	9.4 秘诀: 浏览地址簿.....	228
7.4 秘诀: 使用 UIWebView 显示图像.....	167	9.4.1 (只) 浏览电子邮件地址.....	230
7.5 秘诀: 浏览图像库.....	169	9.4.2 添加新的联系人.....	230
7.6 秘诀: 选择和定制相册中的图像.....	171	9.5 Core Location.....	231
7.7 秘诀: 使用 iPhone 照相机拍照.....	174	9.6 秘诀: Core Location 简介.....	232
7.8 处理 iPhone 音频.....	175	9.7 秘诀: 将地理编码转化为地址.....	235
7.9 秘诀: 使用 Celestial 播放音频.....	176	9.8 秘诀: 使用 Core Location 数据访问 地图.....	238
7.10 秘诀: 使用媒体播放器实现音频和 视频重放.....	178	9.9 秘诀: 访问核心设备信息.....	240
7.11 秘诀: 录制音频.....	179	9.10 秘诀: 启用和禁用近程传感器.....	241
7.12 读入文本数据.....	187	9.11 秘诀: 使用加速度将方向定位到 “向上”.....	241
7.13 从备份文件还原媒体.....	187	9.12 秘诀: 使用加速度移动屏幕上的 对象.....	243
7.14 小结.....	189	9.13 小结.....	246
第8章 控件	190	第10章 连接服务	247
8.1 秘诀: 构建简单的按钮.....	190	10.1 秘诀: 添加自定义设置束.....	247
8.1.1 UIButton 类.....	191		
8.1.2 构建自定义按钮.....	192		
8.1.3 玻璃按钮 (glass button).....	194		

10.2	秘诀: 使应用程序支持自定义 URL 模式	251
10.3	秘诀: 检查网络状态	253
10.3.1	测试网络状态	253
10.3.2	恢复本地 IP 地址	254
10.3.3	查询站点的 IP 地址	255
10.3.4	检查站点可用性	255
10.4	秘诀: 与 iPhone 数据库交互	257
10.5	秘诀: 将 XML 转换为树	259
10.6	秘诀: 存储和检索密钥链项	261
10.6.1	存储多个密钥链值	265
10.6.2	密钥链持久化	267
10.7	发送和接收文件	267
10.8	秘诀: 构建一个简单的基于 Web 的服务器	268
10.9	通知推送	272
10.10	小结	272
第 11 章	Cover Flow 编程	274
11.1	UICoverFlowLayer 类	274
11.2	构建 Cover Flow 视图	276
11.3	构建 Cover Flow 视图控制器	278
11.3.1	Cover Flow 数据源方法	279
11.3.2	Cover Flow 委托方法	279
11.4	小结	282

iPhone和iPod touch为我们提供了创新性的移动平台，它们使编程充满乐趣。它们是苹果公司手持式计算设备家族的第一批成员。尽管体型偏小，但它们运行的是最新版本的OS X，并且配备了功能全面的SDK，可用来设计和实现各种应用程序。借助Xcode，你可以在项目中利用iPhone的多触摸（multitouch）界面和强大的机载（onboard）特性。本章将探讨SDK的内部机理，介绍如何构建一个基于iPhone的Hello World应用程序。

1.1 苹果公司的 iPhone SDK

准备好开始iPhone编程之旅了吗？你需要苹果公司的SDK（Software Developer Kit，iPhone软件开发包）。参与苹果公司在线（免费）开发人员计划的人可以从苹果公司网站（<http://developer.apple.com/iphone>）免费下载该开发包。iPhone SDK由组成iPhone开发环境基础的以下若干组件构成。

- ❑ **Xcode**。Xcode是iPhone开发工具库中最为重要的一款工具。它提供了一个全面的项目开发和管理环境，包括源代码编辑、丰富的文档和一个图形化调试器。Xcode围绕开源GNU工具即gcc（编译器）和gdb（调试器）构建而成。
- ❑ **Instruments**。Instruments用于分析iPhone应用程序的内部运行状况。它监控内存利用率和性能指标。这样你就可以准确找到并锁定应用程序中的问题，并采取措施提高程序的效率。Instruments提供图形化的性能变化图，可随时显示应用程序中占用资源最多的地方。Instruments围绕Sun公司开发的开源DTrace包构成。在跟踪内存泄漏及确保应用程序在iPhone平台上有效运行方面，Instruments发挥着重要作用。
- ❑ **Dashcode**。Dashcode可创建在传统浏览器环境之外运行的、基于Web的独立应用程序。从概念上说，iPhone版本Dashcode的运行原理与其桌面版本类似，具备布局和调试工具。Dashboard并未采用本机应用程序编译方式，而是提供基于Web的开发方式，这不在本书的讨论范围内。
- ❑ **仿真器（Simulator）**。iPhone仿真器运行于Macintosh之上，借助它，无需连接到实际的iPhone或iPod touch，就可以在台式机上创建和测试应用程序。仿真器提供的API与在iPhone上使用的相同，并针对你的概念设计提供相应的预览。在使用仿真器时，Xcode将编译在Macintosh本地运行的Intel x86代码，而不是iPhone上使用的基于ARM的代码。

□ **IB (Interface Builder)**。IB提供了一个快速的原型工具，可以图形化的方式布局用户界面，以及从Xcode源代码链接到这些预构建的界面。借助IB，可以使用可视设计工具绘制界面，然后将这些屏幕元素连接到应用程序中的对象和方法调用。

利用iPhone SDK套件中的这些组件，既可以开发传统的应用程序，又可以开发基于Web的应用程序。从本机应用程序开发人员的视角来看，最重要的组件是Xcode和仿真器，而Instruments则提供了不可或缺的调优工具。除了这些工具之外，还有一个重要的组件并未提及。该组件随带在SDK中，但容易被人们忽略，它就是Cocoa Touch。

Cocoa Touch是苹果公司针对iPhone应用程序快速开发提供的一个类库。此库以一系列框架库的形式存在，支持开发人员使用用户界面元素（如窗口、文本和表）构建图形化的事件驱动的应用程序。iPhone上的Cocoa Touch与Mac OS X上的AppKit相类似，并且支持在iPhone上创建丰富、可重用的界面。

许多开发人员对iPhone应用程序的大小感到惊异不已，它们确实太小了。这在很大程度上得益于Cocoa Touch强大的库支持。通过让Cocoa Touch处理所有复杂的UI事务，应用程序可以专注于自己的任务。其结果是得到简洁的、有针对性的、一次只处理一项任务的代码。

1.2 组建 iPhone 项目

iPhone Xcode项目包含各种不同的标准和自定义组件。图1-1显示了一个典型的项目。项目元素包括源代码、链接框架和媒体（如图像和音频文件）。Xcode编译源代码，将其链接到框架，然后构建适合iPhone安装的应用程序束。它将媒体添加到此应用程序束中，使程序在iPhone上运行时可以访问这些媒体。

iPhone代码通常采用Objective-C 2.0编写。它是ANSI C的一个面向对象的子集，而ANSI C是结合C和Smalltalk开发而来的。针对不了解该语言的读者，苹果公司在其iPhone开发人员网站上提供了一些优秀的在线教程。其中包括Objective-C面向对象编程简介和Objective-C 2.0参考指南。这些资源将帮助你快速了解该语言。

框架是苹果公司提供的软件库，用于为Cocoa Touch提供可重用的类定义。在Xcode中添加框架的方法是将它们拖到项目的Frameworks文件夹中。添加合适的头文件（如UIKit/UIKit.h或QuartzCore/QuartzCore.h）之后，就可以在程序中调用它们的例程。

相关的媒体还包括需绑定在包中的音频、图像和视频文件，以及帮助向iPhone操作系统定义应用程序的基于文本的文件。请将媒体添加到项目中，然后在代码中引用它们。

图1-1所示是一个极其简单的项目。它仅包括一个源文件（main.m），以及默认的iPhone项目框架（UIKit、Foundation和Core Graphics）和一些支持文件（helloworld.png、Default.png、Icon.png、Info.plist）。这些项目是创建Hello World这种应用程序所需的所有内容。

说明 HelloWorld_Prefix.pch文件是由Xcode自动创建的。它包含预编译的头文件。NIB和XIB文件（.nib、.xib）表示在Interface Builder中创建的文件。这些用户界面定义文件与应用程序链接在一起，并由应用程序在运行时调用。

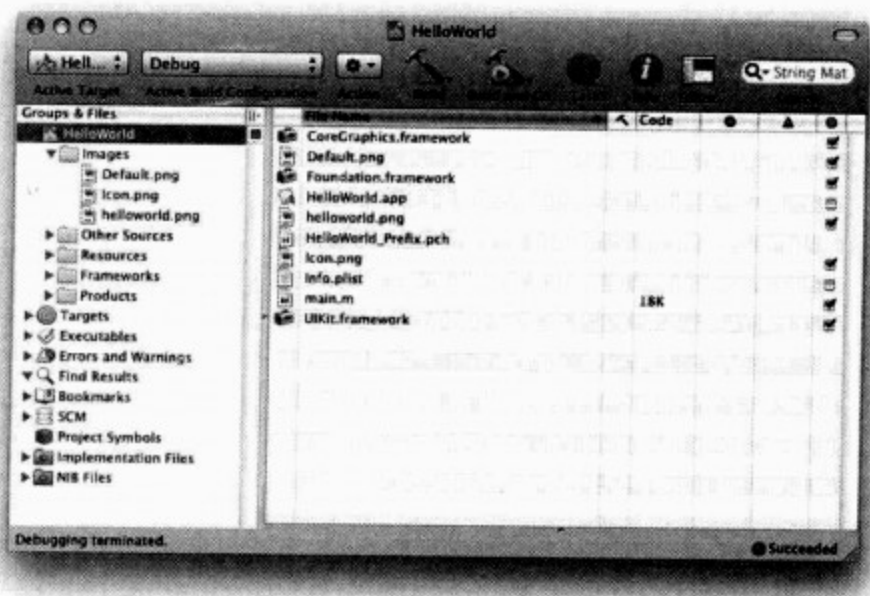


图1-1 Xcode项目整合了源代码、框架和媒体，形成了iPhone应用程序的基础

1.3 iPhone 应用程序组件

与Macintosh的情况类似，iPhone应用程序也以应用程序束（**application bundle**）的形式存在。应用程序束就是采用.app扩展名的文件夹。程序的内容和资源位于此文件夹中，包括编译后的可执行文件、支持性媒体（如图像和音频）和一些向OS描述应用程序的特殊文件。操作系统将文件夹作为一个束对待。

1.3.1 应用程序文件夹层次结构

与Mac不同的是，iPhone束不使用Contents和Resources文件夹保存数据，也不使用MacOS文件夹来保存可执行文件。所有资源都位于顶层文件夹中。例如，Xcode并未将语言支持.lproj文件夹置于Contents/Resources/中，而是将它直接放置到顶层的.app文件夹中。你仍然可以使用子文件夹来组织项目，但它们只是用户定义的临时文件夹，并且无任何标准可循。

iPhone SDK的核心OS支持包括NSBundle类。通过这个类可以轻松地找到应用程序的根文件夹，由此导航到自定义子文件夹以引用和加载资源。

说明 与Macintosh相同，用户域有相应的系统镜像。苹果公司官方分发的应用程序位于主/Applications文件夹中。第三方应用程序则位于/var/mobile/Applications文件夹中。大多数情况下，底层UNIX文件系统已被iPhone的沙盒遮盖，本节将在稍后讨论此话题。

1.3.2 可执行文件

应用程序的可执行文件位于应用程序束的顶层文件夹。它必须携带可执行许可才能正确运行，并且必须经过SpringBoard（iPhone版的Finder）的验证。从固件1.2（仅向开发人员发行）开始，SpringBoard具备一项监视程序特性，以防止代码随意执行。此特性限制了对其他UNIX平台

上的命令行实用工具的使用。SpringBoard的监视程序特性还增加了内存使用限制。系统会关闭任何占用过多系统资源的进程。

1.3.3 Info.plist 文件

和Macintosh一样，iPhone应用程序文件夹包含那个最为重要的Info.plist文件。Info.plist文件是向操作系统描述应用程序的XML属性列表。属性列表存储各种用途的键值对，并且可以保存为可读文本格式或经过压缩的二进制格式。在Info.plist文件中，开发人员将指定应用程序的可执行文件（CFBundleExecutable）和标识符（CFBundleIdentifier）。此标识符对于程序正确运转和顺利执行至关重要。

需要在应用程序中使用苹果公司的标准域命名格式（如com.sadun.appname），操作方法是在Xcode中编辑项目的设置（参见图1-2）。指定你自己的个人域并让Xcode附加产品标识符。要更改标识符，在Xcode中右键单击项目文件并从弹出菜单中选择Get Info。使用Search字段查找Product_Name，然后根据需要编辑该值。

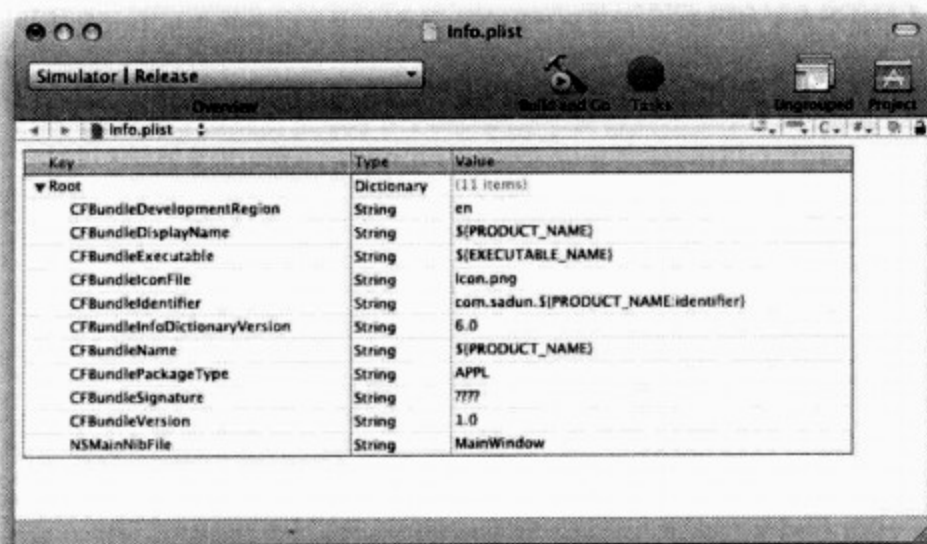


图1-2 通过编辑Info.plist文件来定制应用程序的束标识符。Product_Name标识符在项目的设置中指定

通过产品标识符，你可以与其他应用程序交流并使用SpringBoard（iPhone版的Finder）正确注册应用程序。SpringBoard运行主屏幕，由此可以启动应用程序。产品标识符还形成了内置首选项系统（即用户默认）的基础。

应用程序首选项会使用应用程序标识符自动存储在用户的库中（位于/var/mobile/Library/Preferences）。此标识符将附加.plist扩展名（如com.sadun.appname.plist），并且首选项将以二进制plist格式存储。可以通过将二进制plist传输到Macintosh中来读取它。使用苹果公司的plutil实用工具可以将其从二进制格式转换为基于文本的XML格式：plutil -convert xml1 plistfile。苹果公司使用二进制plist降低存储需求并提高系统性能。

与Macintosh上的情况相同，Info.plist文件提供了进一步的灵活性和高度的可定制性。可以用它

们设置SpringBoard变量（如SBUsesNetwork）或指定图标显示方式（如UIPrerenderedIcon）。一些SpringBoard变量可用于为单个应用程序定义多个角色。例如，Photos和Camera实用工具实际上是相同的应用程序（MobileSlideShow），但却扮演着不同的“角色”。你可以指定应用程序在视图中是否为隐藏。

其他标准的Info.plist键包括用于设置状态栏外观和颜色的UIStatusBarStyle，以及用于隐藏状态栏的UIStatusBarHidden。UIInterfaceOrientation允许覆盖加速计，以创建仅支持横向模式的（UIInterfaceOrientationLandscapeRight）演示。通过设置CFBundleURLTypes可以注册自定义应用程序URL方案（如myCustomApp://）。有关URL方案的更多信息，请参阅第10章。

1.3.4 图标和默认图像

Icon.png和Default.png是两个重要的图像文件。Icon.png充当应用程序的图标，用于在SpringBoard主屏幕上表示应用程序。Default.png（官方称为“启动图像”）提供在应用程序启动时显示的初始屏幕。与Default.png不同，图标的文件名可以随意指定。如果不想使用“Icon.png”，可以在Info.plist文件中将CFBundleIconFile键修改为任意的文件名。

苹果公司建议将Default.png与应用程序背景保持一致。许多开发人员使用Default.png启动图像作为初始图标或作为“请稍候”消息。这与苹果公司的人性化界面指导方针（启动图像应显示连贯的画面，而不是显示广告画面或为延迟表示歉意）相违背，但却是完全可以理解的应用。

“官方”应用程序图标的大小为57×57像素。SpringBoard会自动调整较大的图像。若提供了方角且无光泽的图像，SpringBoard还会平滑处理图像的角并自动添加光亮效果。如果由于某些不可抗拒的原因需要使用预渲染的图像，就在Info.plist文件中将UIPrerenderedIcon设置为<true/>。

说明 如果你计划将应用程序提交到App Store，则需要创建高分辨率（512×512像素）的图标。虽然你可以提高57×57像素icon.png的采样率，但其效果并不理想。而前一种方法可以保证高质量的图像，便于根据需要对图标进行压缩。

1.3.5 XIB（NIB）文件

Interface Builder创建XIB（在Macintosh上又称为NIB）文件，存储预定义、可寻址的用户界面类。这些文件位于应用程序束的顶层目录，并且将在程序中直接调用。在撰写本书时，Interface Builder文件名尚未最终确定，但.xib（Xcode Interface Builder）扩展名在iPhone中的胜算较大。

说明 开发不使用XIB或NIB Interface-Builders的程序时，需要从Info.plist中删除NSMainNibFile键，并放弃项目自动生成的MainWindow.xib文件。

1.3.6 应用程序束中不存在的文件

和Macintosh一样，应用程序束中不存在的文件包括首选项文件（通常存储在Library/

Preferences下的应用程序沙盒中)、应用程序插件(目前存储在/System/Library中,且对于一般开发不可用)和文档(存储在Documents下的沙盒中)。

另一项缺少的内容是Application Support文件夹(至少在Macintosh编程人员看来是如此)。要将支持数据(经常是放置在Application Support结构中)复制到Documents或Library文件夹中。

1.3.7 沙盒

为安全起见,iPhone OS将所有SDK开发限制在应用程序“沙盒”中。iPhone沙盒将应用程序对文件系统的访问只限于一组最基本的文件夹、网络资源和硬件。这类似于在过分受限的学校中接受教育,同时还要遵循如下近乎无理的原则。

- 应用程序可以在自己的沙盒中运作,但不能访问任何其他沙盒。
- 不能共享玩具。不能共享数据。不得弄乱办公室。文件必须位于沙盒提供的文件夹中,并且不能将文件复制到其他应用程序文件夹中,也不能从其他应用程序文件夹复制文件。
- 应用程序拥有自己的Library、Documents和/tmp文件夹。它们类似于限制性较低的平台上的标准文件夹,主要限制对数据的写入和访问。

除了这些限制之外,应用程序必须具有数字签名,通过编码的应用程序标识符向操作系统证明自己的身份,该标识符必须在苹果公司的开发人员计划网站创建。从有利的一面来说,无论何时将设备插入到其主计算机中,沙盒都能让所有程序数据保持同步。从不利的一面来说,苹果公司此时并未明确如何从Windows或Macintosh桌面应用程序访问这些同步数据。(第7章将讨论如何从iTunes及其Mobile Devices框架创建的mdbackup文件恢复数据。)

说明 沙盒规范文件(使用扩展名.sb)存储在/var/mobile/Applications中,其中还包括一些实际的沙盒文件夹。这些文件用于控制权限,如对文件系统中各数据位的读写访问权。如果出现这种可能性,则不要直接编辑此文件,否则应用程序有可能受损而无法使用。示例沙盒文件通常位于/usr/share/sandbox文件夹中。

1.4 平台限制

在讨论iPhone之类的移动平台时,人们会非常关心存储、交互限制和电池寿命之类的问题。移动平台不能提供与桌面系统匹敌的磁盘空间。而除了存储限制外,界面交互和电量消耗也会严重限制开发人员的技能。

不能使用iPhone针对大屏幕、鼠标、物理键盘,甚至物理常开A/C电源进行设计。开发必须根据平台实体因地制宜。所幸的是,苹果公司出奇制胜地设计了一个新平台,它极尽所能地在存储、交互控制和电池寿命的局限下展现了灵活性。

1.4.1 存储限制

iPhone提供强大但精简的OS X安装。虽然整个iPhone OS文件只有数百兆字节大小(对于如

今风行一时的大操作系统安装来说，这简直微不足道），但它提供了丰富的框架库。这些预编译例程的框架支持iPhone用户运行众多精简的应用程序，如从电话到音频播放，从电子邮件到Web浏览。iPhone提供了足够的编程支持来创建灵活的界面，同时保持系统文件足够小，以符合严格的存储限制。

1.4.2 数据访问限制

所有iPhone应用程序都采用沙盒机制。也就是说，应用程序受到文件系统的严格限制。不能从其他应用程序和其他外部文件夹（包括设备上的iTune库）访问程序。但是，如果iPhone建立了网络连接，程序可以访问因特网上任何免费可用的数据。

1.4.3 内存限制

在iPhone上，内存管理非常重要。iPhone不支持基于磁盘交换的虚拟内存。当内存耗尽时，iPhone将会重新启动——如苹果公司所说，意外重启并不是你所期望的用户体验。由于没有交换文件，你必须小心地管理内存需求，并时刻准备着iPhone OS会因内存不足而终止应用程序。你还需要注意应用程序所使用的资源。过多高分辨率图像或音频文件会导致应用程序自动终止。

说明 Xcode使用SDK随带的pngcrush实用工具自动优化PNG图像。（该程序位于iPhone OS平台文件夹/Developer中。在命令行中带-iphone选项运行该工具，将标准PNG文件转换为iPhone支持的格式。）鉴于此，应尽可能在iPhone应用程序中使用PNG图像作为首选的图像格式。

1.4.4 交互限制

缺少物理输入设备和只能小屏幕操作并不意味着缺乏交互灵活性。借助多触摸（multitouch），可以构建不受限制的用户界面。iPhone的触摸技术可以使用比实际物理设备更大的虚拟屏幕，来设计具备文本输入和指针控制的应用程序。

智能自动纠错屏幕键盘和探测方向的加速计提供了两项关键技术，使iPhone明显地有别于其他移动计算设备。当然，这表示需要对文本输入和窗口滚动等方面加以控制。

应努力设计易输入式界面，而不是类似于桌面程序的界面。记住，一次只能使用一个窗口，这与桌面应用程序可任意使用多个窗口的方式不同。

说明 iPhone屏幕最多支持同时5次触摸操作，但很少有应用程序需要同时使用3次以上的触摸操作。

1.4.5 电量限制

对于移动平台，不能忽略电量限制。苹果公司的SDK特性将帮助你设计应用程序，以限制CPU占用并避免用完电池电量。灵活应用技术（如适当地挂起程序）可以让应用程序在iPhone上正常

运行，同时控制软件不会使设备过度发热。一些持续运行的程序会产生相当高的热量，这会让电话变得非常烫手并快速耗完电量。Camera应用程序就是一个明显的例子。

1.4.6 应用程序限制

苹果公司提出了一项强有力的“一次一个应用程序”策略。这表示，第三方开发人员不能开发在后台运行的应用程序，如苹果公司的Mail和Phone实用工具。程序每次运行时，它必须清空一切是非，然后才将控制传递给用户选择的下一个应用程序。不能让检查新消息或定期发送更新的后台程序一直运行下去。一些爱好者创建的“Open SDK”忽略了此限制，但是使用这些工具创建的应用程序无法添加到iPhone App Store中，因而也不能在其上销售。

另一方面，苹果公司不支持推送（push）来自Web服务的数据。已注册的服务可以将标记编号和消息推送给用户，让他们知道数据正在这些服务器上等待。

说明 根据iPhone的服务条款，不能为iPhone应用程序创建外部框架，也不能在提交给App Store的应用程序中使用Cocoa的插件架构。

1.4.7 用户行为限制

这不是物理设备上的限制，不过也必须考虑到iPhone用户会时不时使用基于电话的应用程序的事实。他们进入程序，很快使用完毕，然后马上离开。设备的手持特性意味着必须围绕短期交互来设计应用程序，并让程序在用户将电话放回口袋时自动中止。要保存会话间的应用程序状态，每次会话开始时迅速重新启动并大致进入用户上次使用程序的状态。

1.5 SDK 限制

你可能已经预料到，构建iPhone应用程序与构建Macintosh应用程序类似，也需要使用Objective-C 2.0，也通过链接到各种框架进行编译。换句话说，iPhone SDK是受限的。下面是需要记住的一些要点。

- ❑ 垃圾收集仍不可用，并且可能始终如此。苹果公司的内部人员表示平台限制不允许以正常实用的方式实现垃圾收集。你需要负责保留和释放内存中的对象。
- ❑ 许多库只得到了部分实现。Core Animation只能通过Quartz Core框架得以部分利用，但许多类和方法实际上仍然不可用。这是因为你使用的是早期发行的软件。绕过缺失的部分并将遇到的bug提交给苹果公司，希望他们做好需要使用的部分。注意，苹果公司故意限制了对某些专用类的访问。
- ❑ 公共SDK框架并不像私有框架那么多样化。在最初的iPhone开放SDK世界中，可以调用iTunes Store框架来搜索移动商店，调用Celestial框架来进行类似QuickTime的音频/视频播放。随着官方SDK的面市，它们已不再公开可用，并且苹果公司已经将第三方开发严格限制为一个公共框架子集。

1.6 编程范型

iPhone编程中心采用两种重要的范型：面向对象范型和模型-视图-控制器(MVC)设计模式。iPhone SDK在设计时考虑了在开发人员构建的程序中支持这些概念。为此，它引入了委托(控制器)、数据源方法(模型)和自定义的视图类(视图)。下面简要介绍本书所使用的一些重要的iPhone/Cocoa Touch设计术语。

1.6.1 面向对象编程

Objective-C主要以Smalltalk为基础，后者曾经是最为重要的一种面向对象语言。面向对象编程使用封装和继承的概念来构建可重用的类，这些类使用已经发布的外部接口和私有的内部实现。应用程序将通过一些能够像乐高积木玩具一样整合在一起的具体类来构建，通过类声明很明确哪些部分可以组合在一起。

伪多重继承(通过调用转移)是Objective-C面向对象编程方法中的一个重要特性。iPhone类可以从多个父类继承行为和数据类型。以UITextView类为例，它既是文本，同时也是视图。与其他视图类相似，它可以出现在屏幕上。它有边界和指定的不透明度。同时，它还继承了一些特定于文本的行为。你可以方便地修改其显示字体、颜色或文本大小。Objective-C和Cocoa Touch将这些行为结合在一个易于使用的类中。

1.6.2 模型-视图-控制器

MVC将屏幕对象的外观和行为分开。屏幕按钮(视图)没有内在含义，它只是用户可以按下的按钮。此视图的控制器充当桥梁，将用户交互(如按钮触击)与应用程序中的目标方法(即模型)联系在一起。应用程序提供和保存有意义的数据，并通过生成某种有用的结果来响应按钮触击等交互。

各MVC元素彼此独立运作。举例来说，可以将单击按钮换成切换开关，而无需修改模型或控制器。程序会继续像原来一样运行，但GUI现在会拥有不同的外观。或者可以保留界面不变，而在模型中修改应用程序按钮的触发方式。通过分开这些元素，可以构建可维护、可独立更新的程序组件。

iPhone上的MVC范型划分为以下类别。

- 视图。视图组件由UIView类的子类以及与其相关的(命名有些不妥的)UIViewController类提供。
- 控制器。控制器行为通过3种关键技术实现：委托、目标操作和通知。
- 模型。模型方法通过数据源和数据含义等协议提供数据，需要实现由控制器触发的回调方法。

这3种元素是MVC编程范型的主要部分。让我们详细了解一下iPhone MVC设计模式的这些元素。以下部分将介绍各元素及其支持类。

1. 视图类

iPhone在构建视图时以两个重要的类为基础：UIView和UIViewController。这两个类负责定义和放置所有屏幕元素。

视图在屏幕上绘制事物时，UIView表示最抽象的视图类。几乎所有用户界面类都继承自UIView或其父类UIResponder。视图提供组成应用程序的所有可视元素。重要的UIView类包括UITextView、UIImageView和UIAlertViews等。UIWindow类（一种UIView）提供到应用程序的视口（viewport），是显示的根本要素。

由于其屏幕特性，所有视图都创建了某种类型的框架。这是一个屏幕上的矩形框，它定义了各视图所占的空间。该矩形框根据视图的原点和宽度建立。

视图是分层结构，由子视图树组成。要显示某个视图，可以将它设置为主窗口的内容视图，也可以将它添加到另一个视图中，方法是使用addSubview方法为父视图分配一个子视图。可以将视图看成是向屏幕添加几层透明薄膜，各视图都有各自的绘图。最后添加的视图就是当前看到的视图。较早添加的视图可能被其后加上的视图遮住而变得模糊。

严格地说，UIViewController类并不是MVC概念中的控制器，这与它的名称不符。它们负责对屏幕中各项进行布局，以及使较复杂的布局细节变得模糊。苹果公司的术语并非总是与学校里计算机课程所教授的MVC范型相一致。

最重要的是，视图控制器可以简化开发人员的工作。它们负责根据用户握持iPhone的方式来旋转屏幕显示。在使用导航栏或工具栏时，它们将重新调整视图的大小以使其适合边界。它们处理所有界面事务，并将直接管理交互元素所涉及的复杂性隐藏在底层。开发人员在设计和构建iPhone应用程序时甚至不必再使用UIViewController或其子类，但为什么还要用呢？因为该类提供了极大的便利性，编写应用程序时不使用它是得不偿失的。

除了支持基控制器的方向和视图重新调整之外，还有UINavigationController和UITabBarController这两个特殊的控制器可以神奇地处理视图切换。前者支持在各视图间平滑地切换显示。它会记住视图显示的顺序，并提供完整的“返回”按钮浏览路径，因此不必额外编程就可返回之前的视图。

后者可以使用选项卡式显示方便地在视图控制器实例之间来回切换。因此，如果应用程序有10大项目列表、游戏窗口和帮助表，那么可以添加一个含有3个按钮的选项卡栏，用于在这些视图之间迅速切换，而不需要额外编程。

每个UIViewController子类都实现了自己的loadView方法。该方法对控制器的子视图进行布局，并建立所有的触发、回调和委托。因此，仅就这方面而言，UIViewController算得上是一个控制器，它提供了各种界面外观下的交互解释方式。并且，由于你几乎总是要向UIViewController本身发送回调，因此除了主要充当所创建的和要显示的视图的控制器之外，它通常还充当应用程序的模型。特别要指出，它并不是MVC，但它非常方便并且易于编程实现。

2. 控制器

苹果公司在设计滑块和表之类的交互式元素时，他们不知道用户会如何使用这些元素。经过

精心设计，类具有通用性。通过MVC，行选择或按钮单击并没有与之相关的编程含义。这需要开发人员提供模型来添加含义。iPhone为预构建的Cocoa Touch类提供了一些与自定义类交流的方式。下面介绍最重要的3种方式：委托、目标操作和通知。

● 委托

许多UIKit类都使用委托移交响应用户交互的责任。在设置某个对象的委托时，让对象传递交互消息，而让该委托负责处理这些事务。UITableViews就是一个很好的例子。当用户触击某个表行时，UITableView没有响应该触击操作的内置方式，而是会请示委托（通常是一个视图控制器类或主应用程序委托）并通过一个委托方法传递选择更改。

UITableView委托方法tableView: didSelectRowAtIndexPath:就是一个典型的例子。模型将控制此方法并实现它对行更改的响应方式。可以显示一个菜单或导航到一个子视图，或在当前选项旁边放置一个选中标记。响应完全依赖于怎样去实现委托的选择更改方法。

要设置某个对象的委托，可以使用setDelegate:方法的一些变体。这可以指示应用程序将交互回调重定向到委托。通过在类声明中添加它所实现的委托协议说明，可以让Xcode知道对象实现了委托调用。该说明位于类继承右侧的尖括号中。代码清单1-1显示了一种UIViewController，它实现了UITableView视图的委托方法。因此，MergedTableController类负责实现所有需要的表委托方法。

委托并不仅限于苹果公司提供的类。你可以方便地在类中添加自己的协议声明，并使用它们定义回调词汇。代码清单1-1创建了FTPHostDelegate协议，该协议声明了ftpHost实例变量。在使用时，对象必须实现协议中声明的所有3种方法。

说明 如果应用程序是围绕中心表视图构建的，可以使用UITableViewController实例简化表的创建和使用。

代码清单1-1 在类定义中定义和添加委托协议声明

```
@protocol FTPHostDelegate <NSObject>
- (void) percentDone: (NSString *) percent;
- (void) downloadDone: (id) sender;
- (void) uploadDone: (id) sender;
@end

@interface MergedTableController : UIViewController <UITableViewDelegate,
    UITableViewDataSource>
{
    UIView                *contentView;
    UITableView            *subView;
    UIButton              *button;
    id <FTPHostDelegate> *ftpHost;
    SEL                   finishedAction;
}
@end
```


● 目标操作

目标操作 (target-action) 是重定向用户交互的一种较为低级的方式。基本上只有在实现 `UIControl` 类的子类时，你会经常遇到它们。借助于目标操作，可以通知控件在特定用户事件发生时联系哪些对象。例如，指定用户单击按钮时要联系的对象。

代码清单1-2给出了一个典型的例子。此代码段定义了一个 `UIBarButtonItem` 实例，即 iPhone 工具栏中使用的一个类似于按钮的典型控件。它将项目的目标设定为 `self`，并将操作设定为 `@selector(setHelvetica:)`。按下该按钮时，它将触发调用定义发送 `setHelvetica:` 消息的对象。

代码清单1-2 使用目标操作向控件添加响应

```
UIBarButtonItem *helvItem = [[UIBarButtonItem alloc]
initWithTitle:@"Helvetica" style:UIBarButtonItemStyleBordered
target:self action:@selector(setHelvetica:)] autorelease];
```

可以看到，方法的名称 (`setHelvetica:`) 完全是任意的。目标操作并未像委托那样依赖已建立的方法词汇。但是，在使用过程中，它们的工作方式完全相同。用户执行某些操作，在本例中为按下按钮，目标将实现选择器以提供有意义的响应。

定义此 `UIBarButtonItem` 实例的任何对象都必须实现一个 `setHelvetica:` 方法。如果未实现此方法，程序会在运行时遇到未定义的方法调用错误，并因此而崩溃。

标准的目标操作对始终传递一个参数，即交互对象。在本例中，该对象为按下的 `UIBarButtonItem` 实例。这种自身引用（触发的对象包括在调用中）支持构建更加通用的操作代码。不必为 `setHelvetica:`、`setGeneva:` 和 `setCourier:` 分别构建单独的方法，可以创建一个 `setFontFace:` 方法来根据用户按下的按钮更新字体。

要将目标操作构建到自己的类中，需添加一个 `id` 类型的目标变量（任何对象类）和一个 `SEL` 类型的操作变量（方法选择器）。

● 通知

除了委托和目标操作之外，iPhone 还通过另一种方式传递模型与视图（必要时还有其他事件）之间的用户交互。**通知** 支持应用程序中的对象相互交流，以及与系统上的其他应用程序通信。通过广播信息，通知支持对象发送状态消息：“我已更改”、“我已开始某任务”或“我已完成”。

其他对象可能在监听这些广播，也可能没有。为了“听到”通知，对象必须向通知中心注册并开始监听消息。iPhone 至少实现了4种通知中心。

❑ **NSNotificationCenter**。这是应用程序内部通知的黄金标准。通过这种通知中心，可以订阅任何一个或所有通知，并监听对象之间的通信。通知得到了完全实现，并且可以携带数据和通知名称。这种“名称+数据”的实现方式提供了极大的灵活性，可以使用此中心执行复杂的消息传递。

❑ **NSDistributedNotificationCenter**。此中心用于实现应用程序间的通知。它在 iPhone 并未得到完全实现，应避免使用它。

- ❑ **DarwinNotificationCenter**。iPhone依赖Darwin通知中心实现应用程序间消息传递。它受到了限制，只支持广播通知，并且不允许通过这些通知发送数据。因此，可以宣布一些项目已被更改，但不能发送关于被修改项目的信息。尽管如此，Darwin通知仍然可以信赖且非常健壮。消息可以可靠地送达。Darwin通知是使用标准BSD通知构建的（例如`notify_post()`、`notify_register_mach_port()`等）。
- ❑ **TelephonyNotificationCenter**。电话通知是私有且未发布的。遗憾的是，苹果公司并未开放此中心，但如果你悄悄监听这个具有特殊用途的中心，则可以知道电话呼叫和SMS消息何时到达。

订阅通知中心非常简单。将应用程序委托（更加具体来说，即`UIViewController`）添加为已注册的观察者。提供一个任意的选择器，以在通知到达时调用，在本例中为`trackNotifications:`。该方法接受一个参数，即`NSNotification`。将名称和对象参数设置为`nil`，确保回调方法将听到所有应用程序通知。

所有通知都包含3个数据元素：通知名称、相关对象和用户信息字典。如果不确定应用程序的UIKit对象生成了哪些通知，则指定回调打印出接收到的所有通知的名称——例如，`NSLog(@"%@", [notification name])`。苹果公司针对通知协议提供的文档说明并没有委托和数据源协议文档那么丰富细致。

通知的种类因当前正在执行的任务而异。例如，旋转应用程序时的通知包括`UIApplicationWillChangeStatusBarOrientationNotification`和`UIDeviceOrientationDidChangeNotification`。在某些情况下，这些通知与已有的委托方法严格对应，但在有些情况下，却并非如此，这时需要监听通知，并编写程序查找苹果公司委托协议实现中的缺口。监听方式如下：

```
[[NSNotificationCenter defaultCenter] addObserver:self
    selector:@selector(trackNotifications:) name:nil object:nil];
```

说明 本书中的代码通常使用的是`printf`，而不是`NSLog()`，因为前者在SDK Beta时期运行更加稳定。

3. 模型

开发人员负责构建所有应用程序语义——任何MVC应用程序的模型部分。你要创建应用程序控制器触发的回调方法，并提供所需的任何委托协议的实现。但是，iPhone SDK助了你一臂之力，它提供了数据源。通过数据源，可以在UIKit对象中填充自定义内容。

● 数据源

数据源指任何向另一个对象提供所需数据的对象。一些UI对象是无任何本机内容的容器。将另一个对象设置为它的数据源时（通常通过`[uiobject setDataSource:applicationobject]`调用），开发人员可用UI对象（视图）在数据源（模型）中查询数据，如给定UITableView的表单元。通常，数据源从文件（如本地数据库）、Web服务（如XML提要）或经过扫描的源（如本地检测的WiFi热点）提取数据源。UITableView和UIPickerView是支持或需要数据源的少见的两个Cocoa Touch类。

数据源类似于委托，必须在另一个对象中实现它们的方法，这一个对象通常是拥有表的 `UITableViewController`。它与委托的不同之处在于创建/提供对象而不是针对用户交互做出反应。

代码清单1-3显示了一个典型的数据源方法，该方法返回特定行的表单元格。与其他数据源方法类似，可以用它将填充给定视图的实现语义从苹果公司提供的构建视图容器的功能中分离出来。

实现数据源协议的对象必须声明自身，这与委托协议相同。代码清单1-1显示了一个支持 `UITableViews` 的委托和数据源协议的类声明。苹果公司的文档全面地说明了数据源协议。

代码清单1-3 数据源方法提供信息，有关于在视图中填充有意义的内容

```
// Return a cell for the ith row, labeled with its number
- (UITableViewCell *)tableView:(UITableView *)tableView
  cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:
        @"any-cell"];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
            reuseIdentifier:@"any-cell"] autorelease];
    }
    // Set up the cell
    cell.text = [tableTitles objectAtIndex:indexPath.row];
    cell.editingStyle = UITableViewCellEditingStyleDelete;
    return cell;
}
```

● UIApplication对象

从理论上说，你会假设iPhone“模型”组件以 `UIApplication` 类为中心。在实践中，它并非如此，至少在MVC概念的模型方面不是这样。在苹果公司SDK中，每个程序都刚好包含一个 `UIApplication` 实例，你可以通过 `[UIApplication sharedApplication]` 来引用它。

大多数情况下，除非你需要在Safari中打开URL，恢复重要窗口或调整状态栏的外观，否则完全可以忽略 `UIApplication`。我们围绕自定义应用程序委托类来构建程序，这个类负责在应用程序启动时进行设置，以及在应用程序终止时关闭进程。否则，将其余模型职责交给自定义 `UIViewController` 类中的方法，或交给自定义模型类。

说明 使用 `[[UIApplication sharedApplication] keyWindow]` 查找应用程序的主窗口对象。

● 公开数据源和委托方法

除了监听通知之外，消息跟踪还是一个宝贵的工具。将以下代码段添加到类定义中，公开类所响应的所有方法（提供文档说明和未提供文档说明的）、数据源和委托：

```
- (BOOL) respondsToSelector:(SEL)aSelector {
    printf("SELECTOR: %s\n", [NSStringFromSelector(aSelector) UTF8String]);
    return [super respondsToSelector:aSelector];
}
```


1.7 构建 iPhone 应用程序主干

你构建的每一个iPhone应用程序几乎都将包含一些关键组件。下面是这些组件的简要介绍、使用频率和作用。

- ❑ **main()函数（始终使用）**。每个基于C的程序都需要使用main()函数。对于iPhone而言，此函数主要用于内存管理以及启动应用程序事件循环。
- ❑ **applicationDidFinishLaunching:方法（始终使用）**。此方法是程序中触发的第一个方法。在其中创建基本窗口、设定内容并指定它作为应用程序的关键响应者。
- ❑ **applicationWillTerminate:方法（经常使用）**。此方法支持在将控制交回SpringBoard之前，处理任何状态最终化。此方法用来保存默认值、更新数据以及关闭文件。
- ❑ **loadView方法（始终使用）**。假定你已经围绕UIViewController构建了自己的应用程序（并且通常没有理由不这样做），强制的loadView方法将设置屏幕并布局任何子视图。要确保从UITableViewController或UITabBarController等专门的子类继承时调用[super loadView]。
- ❑ **shouldAutorotateToInterfaceOrientation:方法（经常使用）**。除非有不可抗拒的原因要强制用户保持在纵向模式方向，否则应添加should-autorotate方法以允许UIViewController方法根据iPhone的方向自动匹配屏幕。

还有许多其他方法需要反复使用。每次创建新应用程序时，都需要考虑这5个方法提及的要点。

1.8 Hello World 应用程序

代码清单1-4提供了一个基本的图像形式的Hello World应用程序。该应用程序是使用UINavigationController和自定义UIViewController(此处称为HelloController)构建的，用于显示一幅图像。顾名思义，该应用程序的作用就是打开自己并在屏幕上显示。因此，你在此处接触到的全是基础知识。它们是应用程序主体所依赖的主要特性，只有掌握它们，才能开始编写具有更多意义的代码。

代码清单1-4 Hello World

```
#import <UIKit/UIKit.h>
@class UIImageView;

// Custom UIViewController displays a simple image
@interface HelloController : UIViewController
{
    UIImageView *contentView;
}
@end

@implementation HelloController
- (id)init
```

```
{
    if (self = [super init])
    {
        self.title = [[[NSBundle mainBundle] infoDictionary]
objectForKey:@"CFBundleName"];
        // place any further initialization here
    }
    return self;
}

- (void)loadView
{
    // Load an application image and set it as the primary view
    contentView = [[UIImageView alloc] initWithFrame:[UIScreen mainScreen]
➤applicationFrame]];
    [contentView setImage:[UIImage imageNamed:@"helloworld.png"]];
    self.view = contentView;
    [contentView release]; // reduce retain count by one

    // Provide support for auto-rotation and resizing
    contentView.autoresizesSubviews = YES;
    contentView.autoresizingMask = (UIViewAutoresizingFlexibleWidth |
➤UIViewAutoresizingFlexibleHeight);
}

// Allow the view to respond to iPhone Orientation changes
- (BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation
{
    return YES;
}

- (void) dealloc
{
    // add any further clean-up here
    [contentView release];
    [super dealloc];
}
@end

// Application Delegate handles application start-up and shut-down
@interface SampleAppDelegate : NSObject <UIApplicationDelegate> {
}
@end

@implementation SampleAppDelegate

// On launch, create a basic window
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    UIWindow *window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen]
```

```
    [window addSubview:nav.view];
    [window makeKeyAndVisible];
}

- (void)applicationWillTerminate:(UIApplication *)application {
    // handle any final state matters here
}

- (void)dealloc {
    [super dealloc];
}

@end

int main(int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, @"SampleAppDelegate");
    [pool release];
    return retVal;
}
```

1.8.1 类

导航控制器负责创建蓝灰色标题栏，如图1-3所示。虽然这个简单演示中的标题栏并没有太多作用，但它在实际的应用程序中却非常重要。打开iPhone Mail和Safari应用程序时，可以看到导航栏的实际效果。UINavigationController类可以方便地将这些特性添加到应用程序中。



图1-3 UINavigationController控制器可以通过缩放子视图来自动处理视图方向更改。在操作图像时，你可能希望掌握方向更改并切换为适合横向的模式，而不是接受此处显示的压缩画面

此处，视图控制器加载一幅图像并根据方向更改对它进行调整。图1-3显示了此源代码针对纵向和横向模式构建的界面。显然，在处理图像时，你可能希望掌握方向更改，并在横向模式与纵向模式之间切换。在本例中，并未选择此方法有两个原因。首先，这样你会清楚地看到自动图像调整发生的过程。其次，有助于理解这种图像调整方式的强大之处——应用程序能完全靠自己来处理方向更改，几乎不需要编写任何代码。

1.8.2 代码

此程序包含两个类定义（HelloController类和SampleAppDelegate类）以及main()函数。与本书中的所有其他代码一样，此代码以单个文件的形式存在。通常它需要分成5个单独的文件，但书中合并了全部代码来确保可读性。强烈建议在实际应用程序中使用标准的多文件系统。

main()函数由Xcode自动生成，基本未经过开发人员手动更改。它调用UIApplicationMain()，向它传递主应用程序委托的名称，在本例中为SampleAppDelegate。如无其他原因，我将本书中的所有示例代码应用程序委托都称为SampleAppDelegate。你可以根据需要为自己的委托指定任意名称。

随后调用的是applicationDidFinishLaunching[colon]方法。此处，代码使用标准mainScreen几何函数创建了一个新窗口。它构建了一个导航控制器，并分配了一个新的HelloController(UIViewController子类)作为其根视图控制器。它指定HelloController视图作为导航控制器的主要子视图。此操作可以让程序支持完整的导航栏特性。

要完成设置窗口，此程序设置nav.view（该导航栏上的整个视图）作为窗口的子视图，并调出该窗口。

在HelloController类中，loadView将创建主应用程序视图（在本例中为加载helloworld.png内容的UIImageView）。使用self.view = contentView将图像视图分配给控制器。

为自动旋转设置一些标记，并添加shouldAutorotateToInterfaceOrientation:以确保应用程序能响应iPhone方向更改。

1.8.3 关于示例代码和内存管理的注意事项

下面的Hello World代码是本书中的第一个示例。本书撰写于从2008年3月开始公共的SDK Beta版时期。在这段时期内，苹果公司发布了若干个SDK版本，且对SDK在类定义和内存管理方面的更改一直没有停止过。书中所有示例代码都经过编写再编写、测试再测试，以确保能继续适用于各SDK修订版。

在本书的代码中，我没有过多地考虑内存管理，只为使示例能稳定运转。你在阅读代码时可能会发现一些似乎无关的驻留内容和各版本缺乏匹配，都是出于这个原因。可以根据需要为代码添加适当的内存管理限制。

本书中的所有示例都使用相同的基本项目（如Hello World）。这将确保你的设备不会被形形色色的图标和名称淹没。可以在我的网站ericasadun.com找到最新的示例代码、Xcode项目和视频。

1.9 构建 Hello World 应用程序

以下部分将使用Xcode和iPhone SDK创建Hello World应用程序。你将创建一个新的iPhone项目，对它稍加定制，然后添加如代码清单1-4所示的Hello World示例代码。

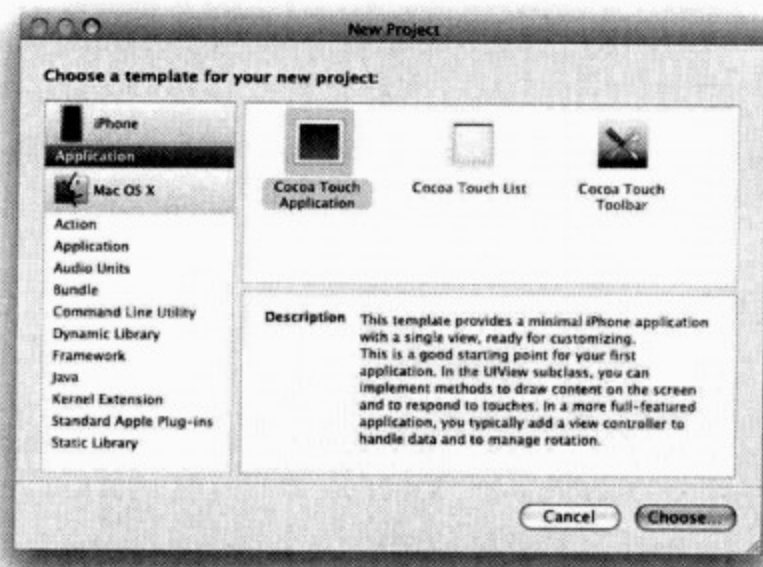


图1-4 New Project窗口包含3个iPhone应用程序模板，可用于构建程序

1.9.1 创建 iPhone 项目

要创建Hello World，需要在Xcode中新建一个项目。选择File, New Project(Command-Shift-N)。此时将打开New Project窗口（如图1-4所示）。选择iPhone, Application, Cocoa Touch Application（较新的Xcode版本中改为了View Based Application），然后单击Choose。使用标准Leopard控件转到要保存新应用程序的位置，输入名称（Hello World）并单击Save。Xcode将打开一个新的iPhone项目窗口，并在其中填充构建第一个iPhone应用程序所需的所有基本元素和框架。你将在此项目中看到以下几项。

- ❑ **Foundation和Core Graphics框架。**这些基本框架支持使用与在Macintosh中类似的基础类和调用来构建iPhone应用程序。
- ❑ **UIKit框架。**此框架提供特定于iPhone的用户界面元素，对于开发在iPhone屏幕上可见且可交互的应用程序非常重要。
- ❑ **HelloWorld.app。**该占位符显示为红色，用于存储已完成的应用程序。与在Macintosh上类似，iPhone应用程序以束的形式存在，它由中央文件夹中的许多项组成。
- ❑ **Info.plist。**此文件向iPhone系统描述应用程序，并支持你指定其可执行文件、应用程序标识符和其他关键特性。其运行方式与Mac上的Info.plist文件相同。
- ❑ **mainWindow.xib。**此Interface Builder文件将创建一个未被填充的窗口。在这次基本练习中将不会使用它。
- ❑ **main.m、HelloWorldAppDelegate.h和HelloWorldAppDelegate.m等。**这些文件包含一个粗略的主干，可以通过定制和扩展它们来创建应用程序。

1.9.2 运行主干

如果愿意，可以选择Run, Run。程序将显示一个全黑的屏幕，此时可以通过单击模拟的Home键退出程序。标准项目主干的作用仅仅是启动一个空白窗口。默认情况下，项目的编译和启动将始终在仿真器中完成。

可以选择Project, Set Active SDK，指定Xcode是为基于Leopard的仿真器还是为iPhone构建应用程序。在撰写本书时，未经认证的开发人员不能将软件加载到iPhone中。iPhone开发人员计划的基本费用是99美元/年，可以访问on-iPhone App Store。要加入该计划，请访问<http://developer.apple.com/iphone/program>。

1.9.3 定制 iPhone 项目

以下步骤可将标准主干转换为Hello World项目。

(1) 删除类。选择左侧Groups & Files列中的Classes文件夹，然后按Delete。Xcode会要求进行确认。单击Also Move to Trash。

(2) 删除.xib文件。在本项目中，不需要使用它。

(3) 从Info.plist文件中删除两行。它们分别是带NSMainNibFile键的行及MainWindow字符串后面的一行。

(4) 添加一个Images文件夹。右键单击Groups & Files列中的Hello World并选择Add, New Group。将新文件夹命名为Pictures。

(5) 在项目中添加图像。从本书提供的Chapter One项目文件夹中将Icon.png、Default.png和helloworld.png这3个图像拖到Pictures文件夹中。选中Copy Items into Destination Group's Folder（如果需要）并单击Add。

(6) 替换main.m。打开main.m并将该文件中的所有文本替换为Chapter One项目文件夹中main.m文件的内容。选择File, Save (Command-S) 保存更改，然后关闭main.m窗口。

(7) 选择iPhone仿真器。选择Project, Set Active SDK, iPhone Simulator。

(8) 运行程序。选择Build, Build & Run (Command-R)。程序将编译并在iPhone模拟窗口中启动。

初次运行模拟iPhone应用程序时可能需要了解以下操作。

❑ 旋转屏幕。当Hello World应用程序运行时，可以使用Command-Left箭头和Command-Right箭头旋转仿真器。

❑ 返回主屏幕。按下Home按钮 (Command-Shift-H) 可离开应用程序并返回SpringBoard主屏幕。

❑ 锁定“电话”。选择Hardware, Lock (Command-L) 可模拟锁定电话。

❑ 查看控制台。在Xcode中，选择Run, Console (Command-Shift-R) 查看Xcode控制台窗口。printf、NSLog和CFShow消息默认都在此窗口中发送。在NSLog消息未出现的情况（极少发生）下，使用Xcode组织程序 (Window, Organizer, Console) 或打开/Applications/

Utilities/Console查看NSLog消息。

- ❑ **运行检测。**使用Run, Start with Performance Tool选择仿真器要使用的检测 (Instruments) 模板 (如Object Allocations或CPU Sampler)。检测在应用程序运行期间监视内存和CUP使用。

说明 虽然Organizer窗口可直接访问故障日志,但你可能希望同时启用Developer Mode for Crash Reporter。这就允许及时查看跟踪反馈,而不需要总是单击Report。为此,打开/Developer/Applications/Utilities/CrashReporterPrefs,选择Developer,然后退出。(此技巧由infinite-labs.net的Emanuele Vulcano提供。)

1.9.4 编辑标识信息

默认Info.plist文件支持创建工作程序,但默认状态无法正确处理iPhone设置 (provisioning),后者是电话应用程序测试和分发的必要前提条件,本章将在稍后讨论此内容。iPhones要求一个有符号的标识,可以在苹果公司的iPhone开发人员程序网站 (<http://developer.apple.com/iphone/program>) 生成该标识。对于仅限于仿真器的应用,可以双击Info.plist文件在新窗口中打开它。此处,你将看到程序的应用程序标识符,只可以编辑公司名称条目。

Info.plist文件并不是定制工作的最后一项。关闭Info并双击InfoPlist.strings。此文件与许多其他项目文件列在一处。编辑"©__MyCompanyName__, 2008"为程序使用实际的版权声明。再次选择File, Save (Command-S) 保存更改。

1.9.5 使用调试器

Xcode集成的调试器为iPhone应用程序开发提供了一个宝贵的工具。以下内容将介绍调试器的位置,以及在程序中使用调试器的基本方法。在这些步骤中,你将探索如何设置断点,以及如何使用调试器控制台检查程序详细信息。

1. 设置断点

在Hello World程序中找到loadView方法。单击最左侧右括号左边的Xcode窗口列。将出现一个蓝色的断点指示符 (参见图1-5)。深蓝色表示断点为活动状态。单击一次取消活动状态 (断点将变为淡蓝色),再单击一次将重新激活。可以通过将断点拖离屏幕来删除它们,通过单击任何代码行旁边的列来添加断点。

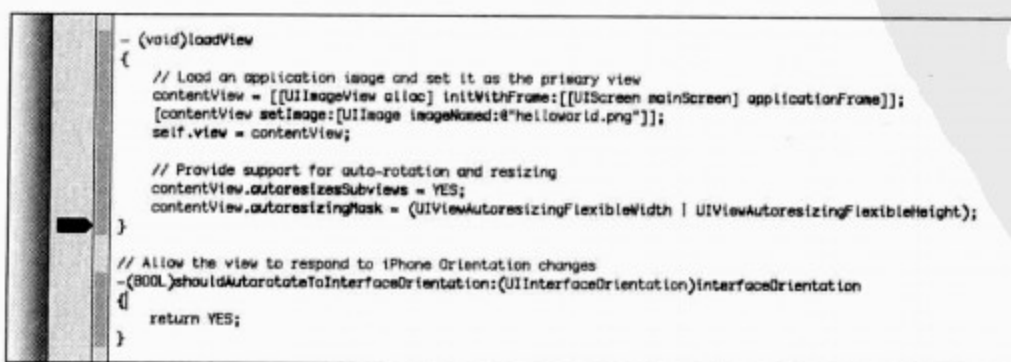


图1-5 Xcode窗口最左侧列中出现的蓝色断点指示符

2. 运行程序

确保断点为深蓝色，且Xcode屏幕顶部的按钮显示为“Deactivate”（这表示断点处于活动状态），然后运行程序。程序在运行到断点处时会自动停止，蓝色箭头上方将出现一个红点。

3. 打开Debugger窗口

选择Run, Debugger (Command-Shift-Y) 打开主调试器，并选择Run, Console (Command-Shift-R) 打开控制台窗口。图1-6显示了这两个窗口。调试器提供一个用于检查程序对象的图形前端。控制台提供一个标准Gnu Debugger (gdb) 命令行。

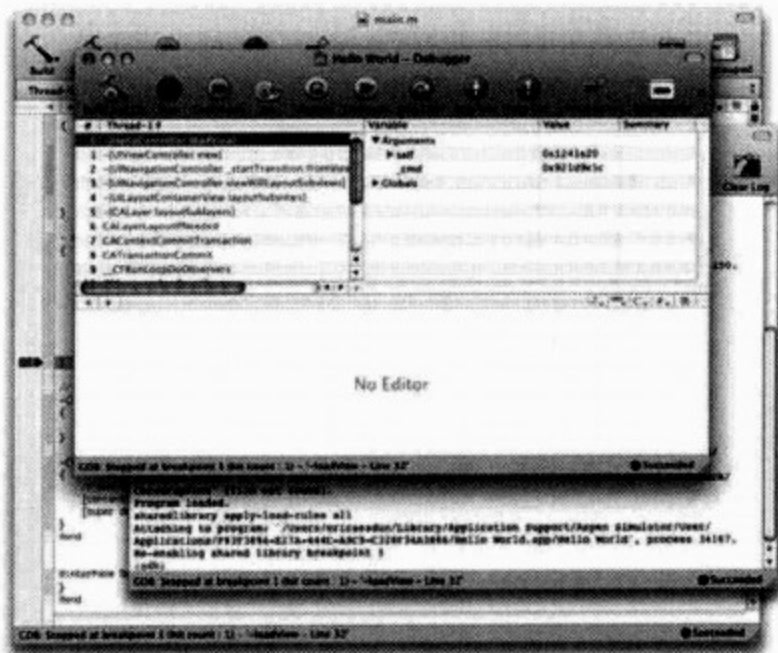


图1-6 Xcode的图形调试器支持以交互的方式检查程序状态。命令行版本的gdb将在控制台窗口中并发运行，如（gdb）提示符所示。活动断点处将显示一个红色箭头

4. 检查

在断点处停止之后，可以使用交互式调试器或gdb命令行检查程序中的对象。对于本例，依次查看Arguments、self、content View、UIView和_viewFlags，确认已正确设置了autoresizeMask。假设苹果公司未更改其常量，此掩码应为18，对应的灵活宽度和高度分别为2和16。

5. 考虑其他断点选项

Xcode窗口顶部的断点菜单支持添加自定义断点操作来代替在执行中使用的一般断点。图1-7显示了这些选项。要了解关于Xcode调试器及其特性的更多信息，请参阅苹果公司的Xcode User Guide: Debugging Programs。

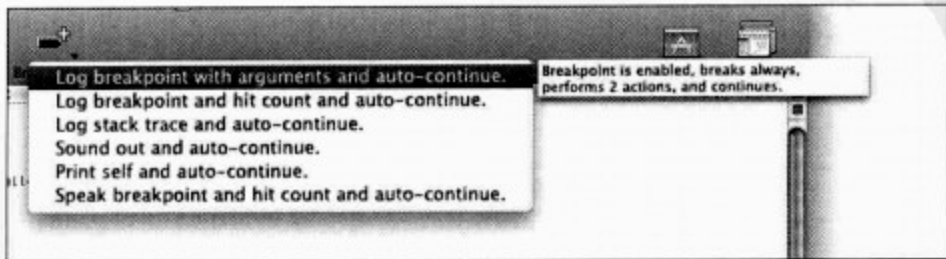


图1-7 使用预定义的断点操作自动执行调试器操作

1.10 苹果公司的 iPhone 开发人员计划

不能直接在iPhone上运行程序，除非加入了苹果公司（需付费）的iPhone开发人员计划。加入该计划的最低费用是99美元，可以作为个人（99美元/年）、公司（99美元/年）或企业内部（299美元/年）开发iPhone应用程序。如需申请，请访问<http://developer.apple.com/iphone/program>。

iPhone使用代码签名。只有正确签名的应用程序才能在iPhone上运行。加入此计划之后，你将能够创建一个将你授权为开发人员的签名证书，并获得iPhone或iPod touch单元进行测试。关于此计划、临时分发（针对小部分用户）、一般设置流程和对苹果公司iPhone App Store访问的详细信息尚未完全确定。请随时关注苹果公司提供的最新信息。

1.10.1 开发电话

开发电话和iPod必须向苹果公司注册，并通过Xcode进行设置。成功加入开发人员计划并付费之后，你将能够使用苹果公司的程序门户获取认证并注册你的单元。这个过程有点复杂。建议在创建开发和分发认证以及为应用程序构建设置配置文件时，按照苹果公司的说明来操作。

首次使用标准USB线将iPhone连接到计算机时，Xcode将检查你的单元（如图1-8所示）。如果希望使用你的设备进行开发，需确认该设备；否则单击Ignore。开发单元只能专用于开发。使用设备作为开发单元意味着它受机载数据更改的影响，并且可能不再是可靠的单元。

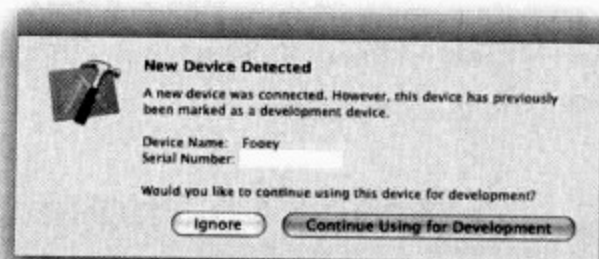


图1-8 Xcode自动检测新iPhone和iPod touch，并能将它们添加为开发设备

1.10.2 应用程序标识符

苹果公司使用两类设置：开发设置和分发设置。通过开发设置，可以将软件下载到iPhone中进行测试。分发设置可以创建适合App Store或临时分发的应用程序。iPhone将其活动设置列在Settings, General, Profile中。

要分发iPhone上的应用程序，苹果公司要求你在他们的站点注册Application ID。为此，你必须是注册过的团队管理员。使用苹果公司iPhone Dev Center上的Developer Program门户。使用Program Portal, App IDs输入新ID，然后通过Program, Provisioning创建一个设置配置文件。

要开发应用程序，可以使用特定的应用程序标识符在手机上进行测试。更好的方式是创建一个反向域的通配符com.mydomain.*应用程序标识符完成所有设置，以进行测试。此通配符设置必须匹配Info.plist文件中的各应用程序ID。我使用的是com.sadun.*。选取一个反向域来描述你的单位或个人身份。

在程序门户中创建通配符设置之后，下载新开发设置配置文件。返回Xcode，执行以下操作。

- (1) 将mobileprovision文件拖到组织程序中。或者将它拖到Xcode应用程序图标上。
- (2) 从项目窗口中选择Debug或Release。
- (3) 在项目窗口的Groups & Files列表中，打开Targets并双击应用程序名称。这将打开一个

Target Info对话框。

(4) 在Target Info窗口中选择Debug或Release，匹配在项目窗口中选择的内容。

(5) 单击Properties选项卡。在Identifier字段中输入新应用程序ID。如果设置了通配符，则输入实际的应用程序名称，而不是通配符（例如，应该是com.mydomain.myapplication，而不是com.mydomain.*）。

(6) 单击Build选项卡。将Code Signing Identity设置为Any iPhone OS Device和iPhone Developer。此阶段用于匹配密钥链中的开发证书。要检查密钥链，打开/Applications/Utilities/Keychain Access。My Certificates类别应该包含至少一个证书，该证书的名称中应有iPhone Developer。如果你已经针对分发进行了签名，则还可能有一个iPhone分发证书。

(7) 将Code Signing Provision Profile设置为Any iPhone OS Device，然后在弹出菜单中指定刚才为应用程序ID创建的设置的名称。（不要为Code Signing Identity选择Default Provisioning Profile。应用程序既不会加载，也不会iPhone上运行。）

完成这些步骤之后，你应该能够针对iPhone编译应用程序（Project，Set Active SDK，Device—iPhone OS），安装它并在设备中运行它。如果遇到任何问题，请参阅苹果公司的文档或咨询开发人员技术支持。

1.11 从 Xcode 到 iPhone: Organizer 界面

将项目添加到Xcode中之后，可以使用Xcode的Organizer工具（Window，Organizer或Control-Command-O）来管理开发。此窗口（如图1-9所示）形成了开发计算机与被测试的iPhone或iPod之间的访问控制中心。可以通过此窗口添加和删除应用程序，查看测试时的控制台结果，检查故障日志，以及在测试代码时截取测试单元的屏幕快照。Organizer控制台提供的主要特性如下所示。

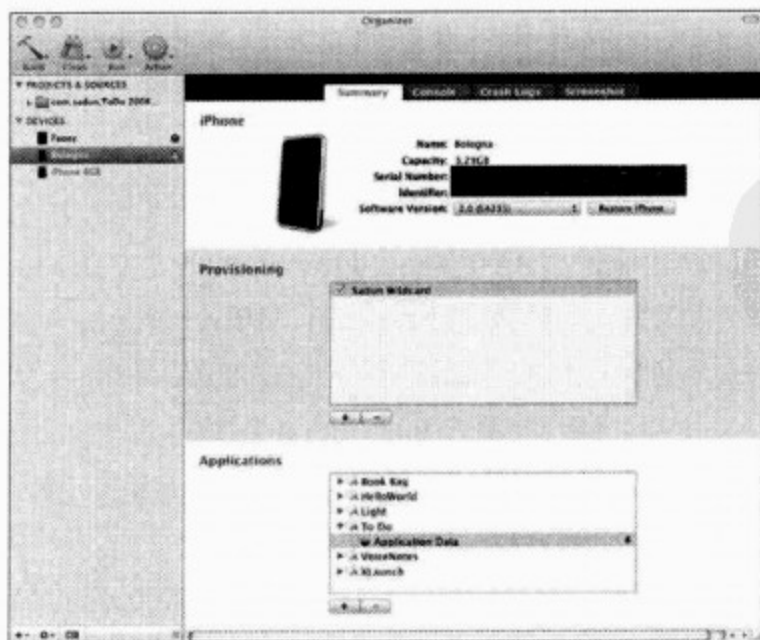


图1-9 基于Xcode的iPhone Organizer窗口（Window，Organizer）提供了一个控制中心来满足大部分应用程序测试需求。此处，可以加载固件，安装和删除应用程序，读取故障日志以及截取基于设备的屏幕快照等

1.11.1 PROJECTS & SOURCES 列表

将当前项目拖到Organizer控制台中，这样可以便于管理它们。添加之后，双击项目名称打开相应项目。可以添加各个源文件以及完整的项目。使用Organizer窗口顶部的Build、Clean、Run和Action选项可以直接在Organizer中执行更多开发任务。

1.11.2 DEVICES 列表

DEVICES列表显示已授权作为开发平台的设备的名称和状态。各名称右侧的指示符显示是否已添加了设备（绿灯表示是，红灯表示否）。

1.11.3 Summary 选项卡

Summary选项卡显示iPhone或iPod touch的名称、容量、序列号和标识符。在该选项卡中，可以设置单元（即准许它运行在Xcode中构建的项目），添加和删除应用程序，以及加载最新的固件。

每个开发人员许可证都允许在测试时最多设置5个iPhones/iPod touch。Provisioning列表显示你的单元可用的应用程序设置列表。你需要选中希望使用的设置。设置将确定哪些应用程序可以而哪些应用程序不可以在设备上运行。根据规则，分发设置将列于此处（还有开发设置，但它们显示为灰色）。

Summary选项卡底部显示已安装的应用程序列表。可以使用+和-按钮添加或删除应用程序。打开各应用程序名称旁边的三角形以显示与该应用程序相关的数据（Documents和Library文件夹）。

说明 苹果公司在其iPhone程序门户页面上提供了如何设置iPhone和创建个人签名证书的完整步骤。请参阅<http://developer.apple.com/iphone/program>。

1.11.4 Console 选项卡

使用控制台查看已连接单元发出的系统消息。此屏幕显示了在连线的iPhone上运行软件时的NSLog()调用。除了调试添加到iPhone应用程序的消息之外，还可以查看苹果系统软件的系统通知、设备信息和调试调用。这里汇聚了各种文本信息。Xcode调试控制台（Run, Console）还显示日志数据以及printf输出。

1.11.5 Crash Logs 选项卡

从屏幕左侧的列中选择特定的故障（标有iPhone应用程序名称以及故障发生的日期和时间）可以直接访问故障日志。该列右侧的文本视图中显示各种故障详细信息，包括线程信息、异常类型等。

1.11.6 Screenshot 选项卡

单击Screenshot选项卡中的Capture按钮可以截取iPhone的屏幕。屏幕快照特性可截取iPhone

的运行屏幕，而不管是否打开了应用程序。因此，可以访问苹果公司内置软件以及iPhone上运行的任何其他应用程序的快照。

截取快照之后，可以将快照图像拖到桌面上，或将它们保存为某公开项目的新Default.png图像。历史快照出现在窗口左侧的库中。选择快照并按下Delete键可永久删除它。

说明 屏幕快照存储在Developer/Shared/Xcode/Screenshots的Application Support文件夹内。

1.11.7 线缆连接

目前，苹果公司尚未提供任何通过无线方式传输、调试或监控应用程序的方法。这表示你要通过一个标准的iPhone USB线来执行几乎所有的任务。设备受线缆限制的调试有时会带来各种问题。其原因包括以下方面。

- ❑ 拔下线缆时，所有交互式调试、控制台和屏幕快照特性都将关闭。因此，需要始终插入该线缆。
- ❑ 插上线缆的iPhone使用不太方便。线缆连接很稳定，但在这种情况下（例如，iPhone只能以75°角放置）一边测试界面一边触摸屏幕非常不便。
- ❑ 线缆接口位于单元的底部而不是顶部，这意味着非常容易碰到线缆，将iPhone摔到地板上。

显然，无线测试可以极大改善这其中的诸多问题。遗憾的是，苹果公司尚未引入该选项。如果愿意，你可以通过一些复杂的方法迂回解决这些问题。一种方法是在iPhone背面贴一枚Velcro粘扣带（使底部接口不用），用来将iPhone固定在桌子上。这样很影响美观，但它能确保iPhone不会总是从桌面上掉下来。

1.11.8 在iPhone上测试应用程序

能够在iPhone上测试应用程序之前，必须首先针对iPhone的ARM处理器编译源代码，并将该束复制到自己的单元中。所幸，Xcode允许方便地选择是针对仿真器编译iPhone应用程序，还是针对连接设备编译iPhone应用程序。步骤如下。

(1) 将经过设置的iPhone连接到Macintosh。

(2) 选择Project, Set Active SDK, Device，并像在仿真器中的操作那样编译应用程序。确保已经设置好应用程序(Target, Build, Code Signing Identity和Target, Build, Code Signing Provisioning Profile)，并将匹配的项目标识符添加到Target, Properties, Identifier。

(3) 编译项目并通过Run, Go或Run, Run运行它。这些菜单选项将编译程序，并为iPhone准备好应用程序。然后，它们将安装并运行程序。

(4) 等待。编译、签名、沙盒以及在iPhone上安装应用程序需要花费一些时间，特别是当项目文件较大且iPhone上已经有了若干个应用程序时。在加载程序时可以休息片刻。

安装好应用程序之后，它将自动启动，如果已经使用调试器进行了编译(Project, Set Active Build Configuration, Debug)，则Gnu Debugger (gdb)将在Xcode调试控制台窗口中打开。调试功能在退出应用程序之前仍然可用。可以通过Xcode（通常通过编译新版本的程序并准备安装它）

或按下iPhone上的Home键退出应用程序。

在iPhone上进行测试很重要。在仿真器上测试简单而方便，但它远远算不上是完整的iPhone测试体验。在撰写本书时，尚不能使用仿真器测试iPhone上的照相机（参见第7章）、密钥链访问（参见第10章）或加速计反馈（参见第9章）。并且由于iPhone是目标平台，因此在本机上运行软件获得的效果要比在仿真器上更佳。iPhone本身提供完全受引导、未被削弱的测试平台，至少允许完全受引导、读写受限的沙盒化应用程序。

说明 SpringBoard通过两种系统通知了解新的和已删除的应用程序：`com.apple.mobile.application_installed`和`com.apple.mobile.application_uninstalled`。Xcode将这些通知发送给Darwin通知中心，然后SpringBoard开始监听。

1.11.9 编译以分发

向App Store提交应用程序需要针对分发（不是开发）编译程序。简要步骤如下。

(1) 在iPhone程序门户网站创建一个分发证书。将其下载到Macintosh，然后添加到密钥链。只需执行一次此操作。创建开发或分发认证之后，无论构建了多少个应用程序，它都将保留在密钥链中。

(2) 如果尚未注册，则在程序门户网站注册一个通配符应用程序标识符。

(3) 为该通配符应用程序标识符创建一个分发设置配置文件。将它命名为易于标识的名称（例如My Wildcard Distribution Profile）。下载它，然后添加到Xcode中，方法是将其拖到Xcode应用程序图标上或Organizer窗口中。

(4) 打开Project Info窗口（Command-I）。在Configurations选项卡中选择Release。单击Duplicate并将新副本命名为Distribution。这个新的分发配置将存储所有必需的信息，用以创建兼容App Store的版本。完成创建之后，可以在Debug、Release和Distribution配置文件之间进行切换，它们将记住自己的设置。记住只需要在两个位置设置配置：主项目窗口和Target Info窗口。

(5) 关闭Project Info窗口。在主项目窗口中，从Active Build Configuration弹出菜单中选择Distribution（图1-1）。这是第一个需要设置构建配置的地方。

(6) 打开Target Info窗口（在主项目窗口中双击Targets, *Project Name*）并选择Build选项卡。

(7) 在Target Info, Build窗口中，从Configuration弹出菜单中选择Distribution。这是第二个需要配置的地方。

(8) 在Code Signing Identity下，将iPhone Developer修改为iPhone Distribution。

(9) 从Code Signing Provisioning Profile弹出菜单中选择新通配符分发配置文件。

(10) 构建项目。然后，选择Groups & Files, *Project Name*, Products, *Application Name*, Reveal in Finder，查找最新创建的应用程序束。

通过这些步骤，可以构建程序的分发版本。构建项目之后，可以压缩新束并将它提交到App Store。

1.12 使用文档中未记录的 API 调用

按官方说法，苹果公司禁止使用文档中未记录的API调用。如果希望应用程序符合App Store限制，应不使用任何未发布的东西。而非正式的情况下，苹果公司的一贯策略是可以使用公共框架中任何API，只要你愿意承担程序可能在苹果公司发布新操作系统更新时无法运行的风险。这种非正式的策略目前已延伸到iPhone。只要公共框架中有，你就可以使用，但是，你必须自己承担可能出现的后果。

本书多处采用了这种开放式访问的方式。文档中未记录的调用在本书中随处可见，尤其是第4章。每一处应用都有标记。所有使用文档中未记录的调用的地方都提供了明确的说明。

同时，本书还是有自己的规则的。本书示例的规则如下所示。

- ❑ 私有框架我们不碰。只链接和访问出现在Apple-sanctioned Frameworks文件夹中的公共项。
- ❑ 不必运行守护进程。所有代码都在主应用程序中执行。
- ❑ 必须在iPhone表示拒绝时退出。代码遵守苹果公司一段时间只运行一个应用程序的策略。
- ❑ 不能使用插件或执行任意代码。苹果公司和App Store支持所有代码。
- ❑ 沙盒就是沙盒。本书从未提过如何访问私有区域，如机载iTunes Media文件夹。

遵章守纪的计算并不意味着大受局限的计算。如果你更愿意避免文档中未记录的API调用(我知道大多数读者都会这样)，本书介绍的绝大多数技巧仍然可以为iPhone开发体验增加乐趣和新鲜感。

1.13 临时分发

苹果公司允许通过临时(Ad Hoc)分发给App Store外部分发应用程序。通过临时分发，可以将应用程序发送到最多100个注册设备，并使用一种特殊的移动设置(允许应用程序在iPhone的FairPlay限制下执行)来运行这些应用程序。临时分发对于Beta测试以及向新闻和刊物网站提交审查应用程序极为有用。

首先注册设备。使用iPhone Developer Program Portal注册设备标识符(Program Portal, Devices)。直接从iPhone(使用第9章中的UIDevice调用)、iTunes(单击iPhone的Summary选项卡上的Serial Number)或System Profiler(选择USB, iPhone, Serial Number)恢复这些标识符。输入标识符和唯一的用户名。

接下来，构建设置。要构建移动设置，选择Program Portal, Provisioning, Distribution。单击Add Profile，然后选择Ad Hoc，输入配置文件名称、标准通配符应用程序标识符(如com.yourname.*)，并选择要部署的一个或多个设备。单击Submit并等待苹果公司构建新移动设置。然后下载设置文件，将它拖放到Xcode中。你将使用它构建应用程序。

需要在Ad Hoc项目中添加一个特殊的权限文件。在Xcode中，选择File, New File, Code Signing, Entitlements。单击Next。新建一个名称为dist.plist的权限文件。单击Finish。权限属性列表将出现在Products文件夹中。双击打开该列表并取消选中get-task-allow(也就是将它设置为FALSE Boolean值)。

设置权限之后，需要将它添加到目标设置中。从项目窗口中选择Distribution配置。双击项目目标（Targets > 项目名称），此时将打开Target Info窗口。在Build选项卡中，再次将配置设置为Distribution（如果尚未设置）。双击Code Signing Entitlements。在Code Signing Entitlementment中添加文件名dist.plist并单击OK。

现在，你已经可以开始构建应用程序了。确保已将Code Signing Identity设置为iPhone Distribution。从Any iPhone OS Device弹出菜单中选择新的临时移动设置。选择Build, Clean（Command-Shift-K），然后选择Build, Build（Command-B）构建项目。从项目窗口的Products文件夹中选择最新编译的产品。右键单击它并选择Reveal in Finder。将打开一个Finder窗口，其中显示了编译后的项目。

分发此应用程序的一个副本，刚才你已通过移动临时设置以及从苹果公司下载的设置编译了该应用程序。用户可以将设置和应用程序拖到iTunes中，然后再让应用程序保持与自己的iPhone同步。应用程序将仅在这些注册过的电话上运行，这提供了一种直接向用户分发应用程序的安全方式。

1.14 小结

本章介绍了iPhone应用程序，如何构建它以及如何通过少许步骤构建基本的Hello World程序。最后，我们来回顾一下本章的要点。

- ❑ 下载和安装SDK以及构建和测试第一个应用程序应能在数小时内完成（具体时间由下载速度决定）。SDK大小超过1G，下载之后便可轻松地创建第一个应用程序。
- ❑ iPhone应用程序束比它在Macintosh上的对等应用程序更加简单，但它们有许多共同特性，包括Info.plist和lproj文件夹。
- ❑ iPhone SDK要求只在有有线缆连接的设备上进行测试。WiFi测试尚不可用。尽管如此，Xcode中的Organizer提供了极好的组织功能，可通过其控制台、故障日志和屏幕快照测试应用程序。
- ❑ 具有Cocoa经验的开发人员就足以创建iPhone应用程序。熟悉Objective-C和Cocoa最佳实践的开发人员会具备良好的开发基础。



几乎所有出现在iPhone屏幕上的内容都是视图。视图的作用类似于一块画布，可以在上面绘制各种颜色、图片和按钮。可以在屏幕中拖动它们，也可以重新调整它们的大小，还可以对它们分层。本章将探讨如何使用Cocoa Touch和UIView来设计和构建屏幕内容。你将了解视图层次结构、几何学和动画，并掌握如何将UITouches中的事件反馈结合到有意义的UIView响应中。UIView的功能非常强大，仅仅一章内容不足以全面讲解整个类的所有方面。因此，本章只介绍它的基本功能，让读者可以顺利开始自己的UIView探索之旅。

2.1 UIView 和 UIWindow

一个窗口，多个视图，iPhone的运行规则大体如此。如果谨记此规则，则iPhone界面设计场景将得到简化。打个比方，UIWindow是电视机，UIView是你喜爱的节目中的演员。演员们可以在屏幕上走动、出现或消失，并且可以随着时间的变化改变自己的外观和行为。

而电视机通常是保持静止的。它有固定的屏幕大小，并不因为你看到的无限的虚拟世界而发生变化。甚至可以在一间房内放置多台电视机（就像可以在同一应用程序中创建若干UIWindow实例一样），但同一时间只能看一台电视。

UIView是GUI构建块。它们提供在屏幕上显示的可视元素，并发起用户交互。每个iPhone用户界面都是通过在一个UIWindow中显示的UIView构建的。UIWindow本身就是一种特殊的UIView。窗口充当容器，它是显示层次结构的根。它保存所有可见的应用程序组件。以下部分将介绍如何控制和操作视图、视图的层次结构及其几何特性。

2.1.1 层次结构

基于树的层次结构对iPhone屏幕上的可见内容进行排序。从主窗口开始，视图以特殊的分层方式布置。所有视图都可以有子视图。每个视图，包括根窗口，都拥有各自子视图的排序列表。视图可能拥有许多子视图，也可能没有子视图。应用程序将决定视图的布局方式和从属关系。

子视图按顺序在屏幕上显示内容，通常从后向前显示。并且，由于iPhone支持视图透明度，因此这种方式与动画单元栈极为类似——动画单元是用于创建动画的透明表单。只有表单中已绘制的部分才会显示出来，而未绘制的部分允许用户看到表单后面的可视元素。

图2-1显示了典型窗口中使用的分层效果。此处，可以看到该窗口拥有一个基于UINavigationController的窗口。该窗口（由最右侧未绘制的元素表示）拥有一个Navigation子视图，而Navigation子视图拥有两个子视图按钮（一左一右）和一个表单。这些项堆积在一起共同构成了GUI。

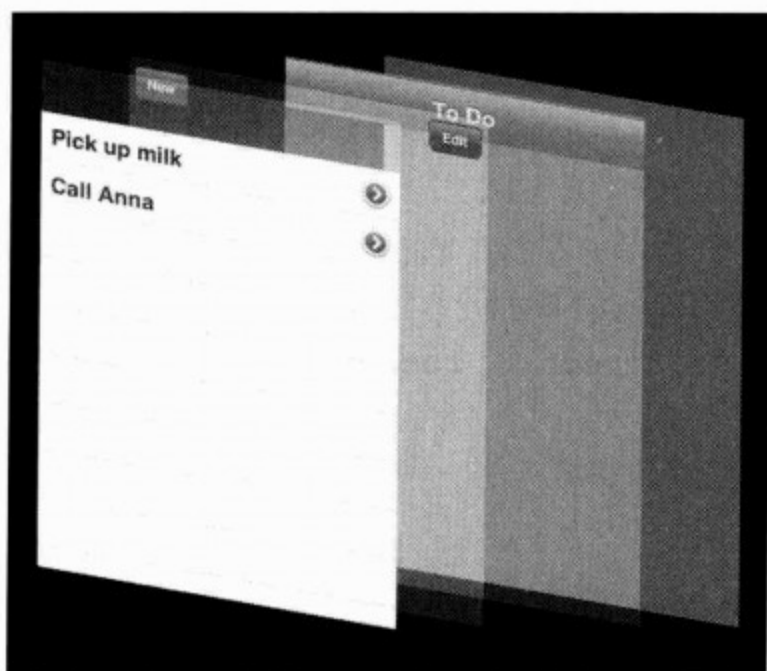


图2-1 添加子视图层次结构后，就可以构建复杂的GUI

注意按钮在导航栏上的显示方式，以及表单是如何调整大小而不影响按钮或标题栏的显示的。按钮框较小，在屏幕上只占用很少的空间。表框则较大，占用了屏幕大部分空间。可以用下面的方法在程序中管理子视图。

- ❑ 要添加子视图，可以调用`[parentView addSubview:child]`。新添加的子视图通常显示在屏幕的最前面。
- ❑ 查询某视图的全部子视图的方法是调用其`[parentView subviews]`。这将返回一组视图，由后向前排序。
- ❑ 通过`[childView removeFromSuperview]`删除视图中的子视图。
- ❑ 使用`[parentView exchangeSubviewAtIndex:i withSubviewAtIndex:j]`对子视图重新排序。使用`bringSubviewToFront:`或`sendSubviewToBack:`将子视图向前或向后移动。
- ❑ 使用`setTag:`给子视图添加标记。通过对子视图编号，可以标识它们。对父视图调用`viewWithTag:`可以从层次结构中检索该视图。

说明 可以为UIView的任何子实例添加标记，包括窗口和控件。因此，举例来说，如果有许多屏幕按钮和开关，添加标记有助于在用户触发它们时对它们进行区分。

2.1.2 几何特征

每个视图都使用一个边框定义其界限。这个边框指定了视图的轮廓：其位置、宽度和高度。

我们使用Core Graphics结构来定义边框矩形。对于边框，这通常表示由原点（CGPoint，x和y）和大小（CGSize，宽度和高度）组成的CGRect矩形。下面是这些类型的简要描述。

1. CGRect

CGRect结构在屏幕上定义了一个矩形。它包含原点（rect.origin）和大小（rect.size）。需要了解以下CGRect函数。

- ❑ **CGRectMake (origin.x, origin.y, size.width, size.height)**在代码中定义矩形。
- ❑ **NSStringFromCGRect (someCGRect)**把CGRect结构转变为格式化字符串。
- ❑ **CGRectFromString (aString)**由字符串表示恢复出矩形。
- ❑ **CGRectInset (aRect)**用来创建较小或较大的矩形（中心点相同）。使用正的内嵌值表示较小的矩形，使用负的内嵌值表示较大的矩形。
- ❑ **CGRectIntersectsRect (rect 1, rect 2)**可以判断矩形结构是否交叉，两个矩形对象是否重叠。
- ❑ **CGRectZero**是一个高度和宽度为零、位于(0,0)的矩形常量。需要创建边框但还不确定边框大小或位置时，可以使用此常量。

2. CGPoint和CGSize

前者通过x和y坐标定义，后者包括宽度和高度。使用CGPointMake (x, y)创建点。CGSizeMake (width, height)创建大小。虽然这两个结构看上去相同（两个浮点值），但iPhone SDK对二者有所区别。点表示位置值，大小表示程度值。不能将myFrame.origin设置为大小。

与矩形相同，可以将它们与字符串进行相互转换，可用如下函数：NSStringFromCGPoint()、NSStringFromCGSize()、CGSizeFromString()和CGPointFromString()。

3. 定义位置

可以通过设置中心点（CGPoint）或边界（CGRect）定义视图的位置。与边框不同，视图的边界使用它自己的坐标系来反映视图的边框。在实际应用中，这表示边界的原点为(0.0,0.0)，大小是它的宽度和高度。

当希望移动视图或重新调整视图大小时，可以更新其边框的原点、中心点或大小。不需要担心显示或隐藏的矩形部分。iPhone将负责完成重新绘图。这允许将视图作为切实的对象对待，并将呈现问题委托给Cocoa Touch。例如[myView setFrame:CGRectMake(0.0f, 50.0f, mywidth, myheight)];。

4. 转换

标准Core Graphics调用会实时转换视图。举例来说，可以应用剪辑、旋转或其他平面几何效果。Cocoa Touch支持一整套仿射转换（转换、旋转、缩放和倾斜等）。任何UIView子类的drawRect:方法都提供通过低级Core Graphics调用进行绘图的入口点。

说明 在调用Core Graphics函数时，记住Quartz从左下方建立坐标系，而UIViews的原点位于左上方。

5. 其他视图特征

除了物理屏幕布局之外，还可以设置以下视图特征。

- ❑ 每个视图都有一个半透明要素（alpha），其值介于不透明和透明之间。可以通过[myView setAlpha:value]来调整它，其alpha值介于0.0（完全透明）和1.0（完全不透明）之间。
- ❑ 可以为视图背景指定一个颜色。[myView setBackgroundColor:[UIColor redColor]]可将视图设置为红色。

6. 视图布局

图2-2显示了普通iPhone应用程序屏幕的布局。对于当前发布的iPhone，屏幕大小在纵向模式下为320×480，在横向模式下为480×320。无论使用横向还是纵向模式，位于屏幕顶部的标准状态栏都占用20像素的高度。要查询状态栏边框，可以调用[[UIApplication sharedApplication] statusBarFrame]。

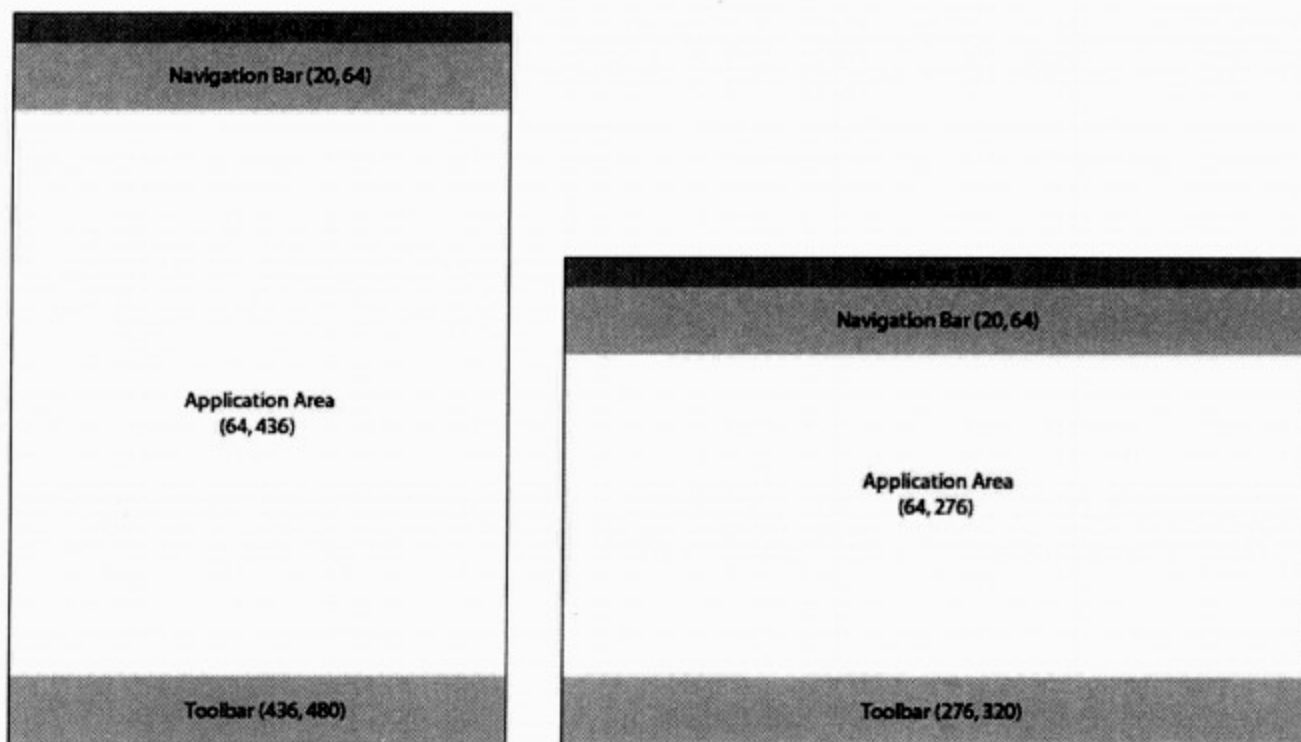


图2-2 在当前的iPhone中，状态栏的高度为20像素，其下方通常是一个高度为44像素的导航栏。如果在屏幕底部使用工具栏，那么它将占用44像素的高度。它有助于使用Photoshop或某些其他图像布局程序来设计屏幕，同时将这些几何因素考虑在内

如果更愿意释放这20像素的屏幕空间，则可以完全隐藏状态栏。使用此UIApplication调用：[[UIApplication sharedApplication] setStatusBarHidden:YES animated:NO]。或者，在应用程序的Info.plist文件中将UIStatusBarHidden设置为<true/>。

要仅在横向模式下运行应用程序，可以将状态栏方向设置为横向。如果计划隐藏状态栏，也应执行此操作（即[[UIApplication sharedApplication] setStatusBarOrientation:UIInterfaceOrientationLandscapeRight]）。这将强制窗口并排显示并生成正确的横向键盘。

UIScreen对象代表iPhone的物理屏幕（[UIScreen mainScreen]）。此屏幕对象建立视图布局边界与像素空间之间的映射。它返回全屏大小（bounds）或只是应用程序占用的矩形

(applicationFrame)。后面会考虑状态栏以及任何工具栏/导航栏（如果已使用）的大小。

默认情况下，UINavigationController、UIToolbar和UITabBar对象的高度均为44像素。使用这些数字计算iPhone屏幕上的可用空间，在未使用Interface Builder的布局工具时对应用程序进行布局。

2.1.3 手势

视图截获用户触摸操作。界面与操作的这种集成允许添加对轻击、拖动等操作的响应。在视图图中添加touchesBegan: withEvent:等触摸处理程序时允许截获用户触摸操作，确定触摸操作的阶段（等价于鼠标按下、鼠标拖动和鼠标弹起），并根据这些触摸生成反馈。

UITouch类通知你事件发生的位置(locationInView:)和轻击数(tapCount)，这对于区分单击和双击操作非常重要。本章中的一些秘诀将演示如何使用这些手势响应，以及如何将视图几何特性和层次结构集成到应用程序中，以实现漂亮、分层、直接操作的界面。

2.2 秘诀：添加递进式子视图

调用addSubview:展开视图层次结构。这将在某个视图添加一个子视图。秘诀2-1展示了一个简单的UIViewController的loadView方法，该方法定义了一系列递进式子视图。它演示了分配、设计和添加视图的基本过程。

这些子视图并未相互嵌套，因为它们都属于相同的父视图。之所以缩排它们是为了方便查看。这种缩排结构使用非常方便的CGRectInset()函数。向它传递一个矩形（使用CGRect结构）和两个内嵌（垂直和水平），它将返回居中的矩形。此处，每个子视图都采用其父视图的内嵌，或者每条边使用32像素的边框。

就其最简单的形式而言，视图可以说就是透明占位符。在缺少有意义的内容的情况下，给视图背景着色可以区分各个视图（如图2-3所示）。在提交实际设计之前测试布局时，这是一个非常实用的技巧。

始终需要考虑坐标系。在操作视图层次时，你必须在其父视图的坐标系中定义视图的边框。秘诀2-1中的示例请求应用程序边框布局主视图，然后将其原点重置为(0,0)。重置原点会将边框从屏幕更新到主视图的坐标系。这种重置形成了后面视图布局的基础。

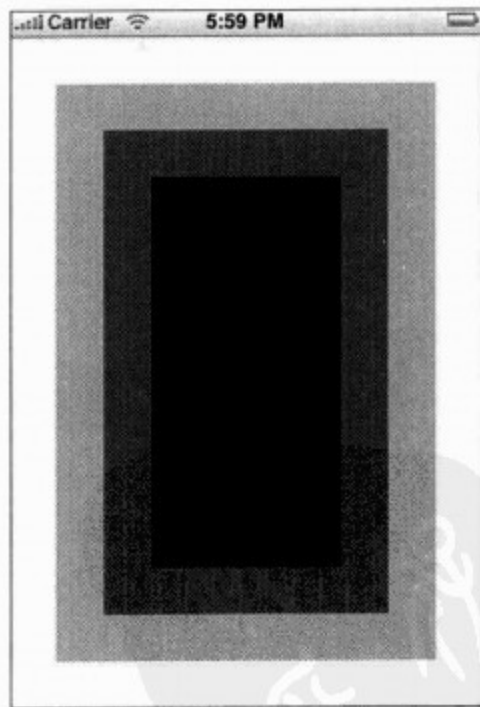


图2-3 秘诀2-1中的代码定义了UIViewController的主视图，包括3个有颜色的内嵌子视图

秘诀2-1 添加内嵌子视图

```
- (void)loadView
{
```

```
// Create the main view
CGRect appRect = [[UIScreen mainScreen] applicationFrame];
contentView = [[UIView alloc] initWithFrame:appRect];
contentView.backgroundColor = [UIColor whiteColor];

// Provide support for autorotation and resizing
contentView.autoresizesSubviews = YES;
contentView.autoresizingMask = (UIViewAutoresizingFlexibleWidth |
    UIViewAutoresizingFlexibleHeight);
self.view = contentView;
[contentView release];

// reset the origin point for subviews. The new origin is 0,0
appRect.origin = CGPointMake(0.0f, 0.0f);

// Add the subviews, each stepped by 32 pixels on each side
UIView *subview = [[UIView alloc] initWithFrame:CGRectInset(appRect, 32.0f,
    32.0f)];
subview.backgroundColor = [UIColor lightGrayColor];
[contentView addSubview:subview];
[subview release];

subview = [[UIView alloc] initWithFrame:CGRectInset(appRect, 64.0f, 64.0f)];
subview.backgroundColor = [UIColor darkGrayColor];
[contentView addSubview:subview];
[subview release];

subview = [[UIView alloc] initWithFrame:CGRectInset(appRect, 96.0f, 96.0f)];
subview.backgroundColor = [UIColor blackColor];
[contentView addSubview:subview];
[subview release];
}
```

重置方向

在秘诀2-1的基础上添加方向更改要花一些心思。不能根据自己的想法仅交换各子视图的高度和宽度。这是因为iPhone上水平和垂直应用程序的形状使用不同的纵横比。假设有一个20像素的状态栏，纵向视图区域为320像素宽×460像素高；横向视图区域为480像素宽×300像素高（参见图2-2）。这种差异使界面不能仅通过旋转实现方向重置。

要旋转此示例，需要添加代码来区分横向方向与纵向方向，并相应地调整边框。如秘诀2-2所示。

避免使用依赖切换的方向重设方案（例如，“我刚才处于纵向模式，如果方向发生更改，我必须进入横向模式。”）。从左横向模式进入右横向模式，而不单击中间的纵向状态是完全可能的。方向重设主要涉及传感器和反馈，并且iPhone不能保证捕获两种方向之间的中间状态。好在UIKit提供了UIViewController回调，可向用户提醒新方向并指定是哪个方向。

秘诀2-2 向之前的子视图示例添加方向重设支持

```

- (void)willRotateToInterfaceOrientation:
    (UIInterfaceOrientation)orientation
    duration:(NSTimeInterval)duration {

    CGRect apprect;
    apprect.origin = CGPointMake(0.0f, 0.0f);

    // adjust the frame size based on actual orientation
    if ((orientation == UIInterfaceOrientationLandscapeLeft) ||
        (orientation == UIInterfaceOrientationLandscapeRight))
        apprect.size = CGSizeMake(480.0f, 300.0f);
    else
        apprect.size = CGSizeMake(320.0f, 460.0f);

    // resize each subview accordingly
    float offset = 32.0f;
    for (UIView *subview in [contentView subviews]) {
        CGRect frame = CGRectInset(apprect, offset, offset);
        [subview setFrame:frame];
        offset += 32.0f;
    }
}

// Allow the view to respond to iPhone Orientation changes
- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation
{
    return YES;
}

```

2.3 秘诀：拖动视图

Cocoa Touch简化了直接视图操作。在处理许多屏幕视图时，iPhone将负责决定用户触摸的是哪个视图，并将所有触摸事件传递给正确的视图。这有助于编写具体的直接操作界面，用户可以触摸、拖动屏幕对象并与之交互。

秘诀2-3侧重实际的触摸。此示例将为UIImageView创建一个名为DragView的子视图，支持用户在iPhone屏幕上拖动视图。对于图像视图，支持用户交互非常重要（通过[dragger setUserInteractionEnabled:YES]）。这对于背景和直接交互视图同样适用。在直接操作界面中操作UIImageView时，确保支持交互，无论其在视图层次中是什么角色。通过图像视图，用户交互切换将影响所有视图的子视图以及视图本身。

当用户初次触摸任何DragView时（参见图2-4中的花），对象将开始位置存储为距视图原点的偏移值。当用户拖动对象时，视图将与手指一同移动——始终维持相同的原点偏移值，以便让移动感觉更加自然。

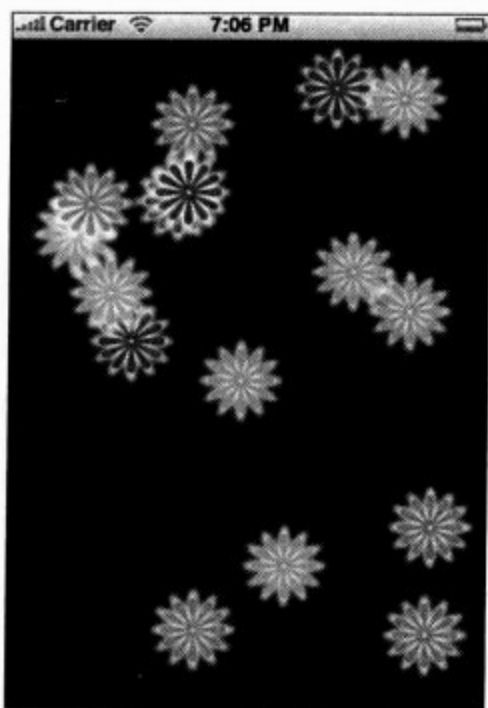


图2-4 秘诀2-3中的代码将创建一个包含16朵花界面，用户可以在iPhone屏幕上拖动这些花

说明 在图2-4的示例中，可以使用多个手指在屏幕上一次拖动多个花。它本质上并非多点触摸，因为每朵花（UIView）一次只响应一个触摸操作。本章将在稍后讨论真正的多点触摸交互。

在本代码中，触摸某个对象还会执行另外一项任务：它会将该对象弹出到父视图的前端。这意味着任何拖动的对象始终悬浮在其他屏幕对象之上。为此，通知视图的父视图（它的superview）将视图呈现到前端。

2.3.1 UITouch

UITouch类定义手指在iPhone屏幕上移动的方式。调用标准的开始、移动和结束处理程序时会发送触摸。还可以查询（UIEvent类的）用户事件以返回触摸，通过touchesForView:和touchesForWindow:影响特定的视图。这些调用将返回一组无序的触摸（NSSet）。

说明 将allObjects发送给任何NSSet以返回由这些对象组成的数组。

触摸反映了以下几项内容：触摸发生的位置（包括当前位置和最近的历史位置），触摸的阶段（主要包括鼠标按下、鼠标移动和鼠标弹起），轻击数量（例如，单击/双击），触摸发生的时间（通过时间戳），等等。

非多触点交互样式假定任何时候都只处理一次触摸。秘诀2-3中的代码通过在返回的触摸集上调用anyObject来恢复每个事件第一次可用的触摸。

秘诀2-3 构建多个可拖动视图

```
/*
 * DragView: Draggable views
```

```
*/

@interface DragView : UIImageView
{
    CGPoint startLocation;
}
@end

@implementation DragView

// Note the touch point and bring the touched view to the front
- (void) touchesBegan:(NSSet*)touches withEvent:(UIEvent*)event
{
    CGPoint pt = [[touches anyObject] locationInView:self];
    startLocation = pt;
    [[self superview] bringSubviewToFront:self];
}

// As the user drags, move the flower with the touch
- (void) touchesMoved:(NSSet*)touches withEvent:(UIEvent*)event
{
    CGPoint pt = [[touches anyObject] locationInView:self];
    CGRect frame = [self frame];

    frame.origin.x += pt.x - startLocation.x;
    frame.origin.y += pt.y - startLocation.y;
    [self setFrame:frame];
}
@end

/*
 * Hello Controller: The primary view controller
 */

@interface HelloController : UIViewController
{
    UIView *contentView;
}
@end

@implementation HelloController

#define MAXFLOWERS 16

CGPoint randomPoint() {return CGPointMake(random() % 256, random() % 396);}

- (void)loadView
{
    // Create the main view with a black background
```



```
CGRect apprect = [[UIScreen mainScreen] applicationFrame];
contentView = [[UIView alloc] initWithFrame:apprect];
contentView.backgroundColor = [UIColor blackColor];
self.view = contentView;
[contentView release];

// Add the flowers to random points on the screen
for (int i = 0; i < MAXFLOWERS; i++)
{
    CGRect dragRect = CGRectMake(0.0f, 0.0f, 64.0f, 64.0f);
    dragRect.origin = randomPoint();
    DragView *dragger = [[DragView alloc] initWithFrame:dragRect];
    [dragger setUserInteractionEnabled:YES];

    // select random flower color
    NSString *whichFlower = [[NSArray arrayWithObjects:@"blueFlower.png",
    @"pinkFlower.png", @"orangeFlower.png", nil] objectAtIndex:(random() %
    3)];
    [dragger setImage:[UIImage imageNamed:whichFlower]];

    // add the new subview
    [contentView addSubview:dragger];
    [dragger release];
}
}

-(void) dealloc
{
    [contentView release];
    [super dealloc];
}
@end
```

2.3.2 添加持久性

持久性表示一个关键的iPhone设计触摸点。当用户离开程序时,苹果公司强烈建议他们尽量返回与离开时相同的状态。在本示例中添加持久性包括以下步骤:

- (1) 存储数据;
- (2) 从已保存会话恢复;
- (3) 提供与最后会话相匹配的启动图像。

1. 存储状态

每个视图都知道其位置,因为可以查询其frame。这允许恢复和存储屏幕上各朵花的位置。花的类型(绿色、粉红或蓝色)则是另外一个话题。要让各视图报告其当前花朵,DragView类也必须存储该值。通过添加字符串实例变量,视图可以返回所使用的图像名称。代码清单2-1显示了扩展后的DragView类定义。

代码清单2-1 更新后的DragView类添加了一个字符串来存储花的类型

```

@interface DragView : UIImageView
{
    CGPoint startLocation;
    NSString *whichFlower;
}
@property (nonatomic, retain) NSString *whichFlower;
@end

```

通过另外添加这个变量，HelloController类能够将颜色列表和位置列表存储在其默认文件中。一个简单的循环收集每个可拖动视图中的这两个值，然后保存它们。代码清单2-2展示了一个updateDefaults方法，其定义在HelloController中。该方法将当前状态保存到磁盘中。它应该在程序快结束之前在应用程序委托的applicationWillTerminate:方法中被调用。

注意此处所使用的NSStringFromCGRect()。它提供了一种严谨的方式将边框信息存储为字符串。要恢复矩形，请调用NSStringFromCGRect()。每个调用接受一个参数：第一个例子中为CGRect，第二个例子中为NSString*。UIKit框架提供了一些调用，可以从/向字符串中转换点、大小以及矩形。

可以看到，Defaults类似于一个字典。为键分配一个对象，iPhone会“自动神奇地”更新与应用程序ID相关的首选项文件。应用程序ID在Info.plist中定义。Defaults存储在应用程序沙盒内部的Library/Preferences中。调用synchronize函数将立即更新这些默认值，而不是等待程序终止。

代码清单2-2 通过用户Defaults存储花朵位置

```

// Collect all the colors and locations and save them for the next use
- (void) updateDefaults
{
    NSMutableArray *colors = [[NSMutableArray alloc] init];
    NSMutableArray *locs = [[NSMutableArray alloc] init];

    for (DragView *dv in [contentView subviews]) {
        [colors addObject:[dv whichFlower]];
        [locs addObject:NSStringFromCGRect([dv frame])];
    }

    [[NSUserDefaults standardUserDefaults] setObject:colors forKey:@"colors"];
    [[NSUserDefaults standardUserDefaults] setObject:locs forKey:@"locs"];
    [[NSUserDefaults standardUserDefaults] synchronize];
    [colors release];
    [locs release];
}

```

2. 恢复状态

持久性感知通常位于视图控制器的init或loadView中（例如，在视图实际出现之前）。这些方法应该查找之前的所有状态信息，并且（在本例中）匹配花朵与状态。查询用户默认值时，此

代码将检查状态数据是否不可用(例如, 返回值为nil)。如果缺少状态值, 方法将在随机位置创建随机花朵。代码清单2-3显示了一个状态感知版本的LoadView。

说明 在操作大型数据源时, 可能希望在UITableViewController的init方法中初始化并填充保存的对象数组, 然后在loadView中拖动它们。如果可能, 在操作多个对象时使用线程避免阻塞。

代码清单2-3 检查之前的状态

```
- (void)loadView
{
    // Create the main view
    CGRect apprect = [[UIScreen mainScreen] applicationFrame];
    contentView = [[UIView alloc] initWithFrame:apprect];
    contentView.backgroundColor = [UIColor blackColor];
    self.view = contentView;

    // Attempt to read in previous colors and locations
    NSMutableArray *colors, *locs;
    colors = [[NSUserDefaults standardUserDefaults] objectForKey:@"colors"];
    locs = [[NSUserDefaults standardUserDefaults] objectForKey:@"locs"];

    for (int i = 0; i < MAXFLOWERS; i++)
    {
        // Use a random point unless there's a previous location
        CGRect dragRect = CGRectMake(0.0f, 0.0f, 64.0f, 64.0f);
        dragRect.origin = randomPoint();
        if (locs && ([locs count] == MAXFLOWERS))
            dragRect = CGRectFromString([locs objectAtIndex:i]);
        DragView *dragger = [[DragView alloc] initWithFrame:dragRect];
        [dragger setUserInteractionEnabled:YES];

        // Use a random color unless there's a previous color
        NSString *whichFlower = [[NSArray arrayWithObjects:@"blueFlower.png",
            @"pinkFlower.png", @"orangeFlower.png", nil] objectAtIndex:(random()
            % 3)];
        if (colors && ([colors count] == MAXFLOWERS))
            whichFlower = [colors objectAtIndex:i];
        [dragger setWhichFlower:whichFlower];
        [dragger setImage:[UIImage imageNamed:whichFlower]];
        // Add the subview
        [contentView addSubview:dragger];
        [dragger release];
    }
}
```

3. 启动图像

苹果公司并未将持久性屏幕快照功能并入其官方SDK发行版中, 但该功能已作为一个未在文

档中记录的调用在UIKit框架中部分可用。要访问如代码清单2-4所示的_writeApplication-Snapshot特性，必须将其手动添加到UIApplicationClass界面中。完成添加之后，可以在结束应用程序之前构建缓存的屏幕快照。

说明 参阅第1章关于在程序中使用文档中未记录的调用和特性的说明。

启动图像的方法是：当离开应用程序时，截取当前屏幕的图像。然后，当应用程序启动时（假定返回你离开时的状态），缓存图像将充当Default.png图像，让你感觉像是直接跳回该状态，而没有任何启动序列。

苹果公司已经在iPhone SDK中支持此特性，并且在撰写本书时，应用程序不能登入以查找已更新的屏幕快照。但愿苹果公司在未来固件发行版中提供此功能。

代码清单2-4 应用程序终止前的屏幕快照

```
@interface UIApplication (Extended)
- (void) _writeApplicationSnapshot;
@end

[[UIApplication sharedApplication] _writeApplicationSnapshot];
```

2.4 秘诀：剪辑视图

在使用直接操作界面时，你可能不希望只使用矩形视图。我们可以方便地将软边框、圆角和其他可视增强添加到UIView实例中。

剪辑将创建仅部分填充视图框的视图形状。通过Core Graphics，可以使用UIView对象的drawRect:方法生成剪辑，这与Macintosh上的操作类似。借助Core Graphics，可以从各种源构建路径，包括点、线、标准形状（如椭圆）和贝塞尔曲线。将视图剪辑到这些路径中可以实现非矩形屏幕对象的感觉。图2-5显示了一系列圆形剪辑视图，它们相互之间存在明显的重叠。这些视图由代码清单2-5中的代码创建。此代码将创建一个路径，执行剪辑，然后将其拖到剪辑视图中。

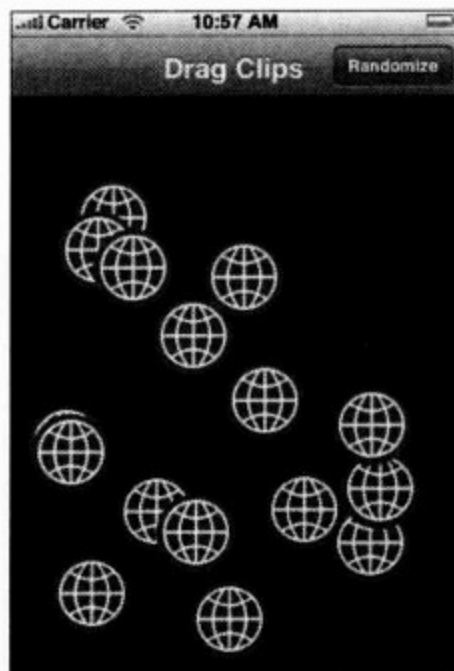


图2-5 通过剪辑，可以使用UIView边框从矩形源数据创建非矩形屏幕视图

代码清单2-5 将视图剪辑为圆形路径

```
- (void) drawRect: (CGRect) aRect
{
    CGRect bounds = CGRectMake(0.0f, 0.0f, SIDELENGTH, SIDELENGTH);
```

```
// Create a new path
CGContextRef context = UIGraphicsGetCurrentContext();
CGMutablePathRef path = CGPathCreateMutable();

// Add circle to path
CGPathAddEllipseInRect(path, NULL, bounds);
CGContextAddPath(context, path);
// Clip to the circle and draw the logo
CGContextClip(context);
[logo drawInRect:bounds];
CFRelease(path);
}
```

2.4.1 通过剪辑平衡触摸

可视剪辑不会影响UIView对触摸的响应。iPhone会感觉到用户在整个视图框中的轻击操作。这包括未绘图区域，如图2-5中的实际圆形框外部的区域，这几乎占据了剪辑演示的大部分空间。这意味着，除非添加某种单击测试，否则用户可能会尝试轻击被UIView边框清晰部分“遮住”的视图。

代码清单2-6将在剪辑视图添加一个简单的单击测试，以确定触摸是否发生在剪辑路径内。作为最简单的例子，实现了一个圆形剪辑和剪辑单击测试。然后使用任何可计算的测试方法确定用户触摸是否与视图相交。将pointInside:withEvent:添加到UIView子类，当触摸发生在视图内部时返回YES，否则返回NO。

代码清单2-6 检查圆形视图上的触摸

```
- (BOOL) pointInside:(CGPoint)point withEvent:(UIEvent *)event
{
    CGPoint pt;
    float HALFSIDE = SIDELENGTH / 2.0f;

    // normalize with centered origin
    pt.x = (point.x - HALFSIDE) / HALFSIDE;
    pt.y = (point.y - HALFSIDE) / HALFSIDE;

    // x^2 + y^2 = hypoteneus length
    float xsquared = pt.x * pt.x;
    float ysquared = pt.y * pt.y;

    // If the length < 1, the point is within the clipped circle
    if ((xsquared + ysquared) < 1.0) return YES;
    return NO;
}
```

2.4.2 访问逐个像素值

可以采用多种方法测试用户对视图的触摸。代码清单2-6计算触摸是否发生在某个圆中。通

过点击蒙板和透明度可变的图像，可以测试某一点的alpha值。透明度控制着是否会触发响应。代码清单2-7扩展UIImageView类，添加了图像的位图表示。它测试位图中触摸处的alpha值（逐点测试）。alpha值小于0.5的像素将不会响应使用本代码的触摸。

说明 代码清单2-7中的代码将返回一个位图上下文，并且其位图数据以苹果公司的示例代码为基础。

代码清单2-7 测试位图上的触摸

```
// Return a bitmap context using alpha/red/green/blue byte values
CGContextRef CreateARGBBitmapContext (CGImageRef inImage)
{
    CGContextRef    context = NULL;
    CGColorSpaceRef colorSpace;
    void *          bitmapData;
    int             bitmapByteCount;
    int             bitmapBytesPerRow;

    size_t pixelsWide = CGImageGetWidth(inImage);
    size_t pixelsHigh = CGImageGetHeight(inImage);
    bitmapBytesPerRow = (pixelsWide * 4);
    bitmapByteCount = (bitmapBytesPerRow * pixelsHigh);
    colorSpace = CGColorSpaceCreateDeviceRGB();

    if (colorSpace == NULL)
    {
        fprintf(stderr, "Error allocating color space\n");
        return NULL;
    }

    // allocate the bitmap & create context
    bitmapData = malloc( bitmapByteCount );
    if (bitmapData == NULL)
    {
        fprintf (stderr, "Memory not allocated!");
        CGColorSpaceRelease( colorSpace );
        return NULL;
    }
    context = CGBitmapContextCreate (bitmapData, pixelsWide, pixelsHigh, 8,
    bitmapBytesPerRow, colorSpace, kCGImageAlphaPremultipliedFirst);
    if (context == NULL)
    {
        free (bitmapData);
        fprintf (stderr, "Context not created!");
    }
    CGColorSpaceRelease( colorSpace );
    return context;
}
```



```

// Return Image Pixel data as an ARGB bitmap
unsigned char *RequestImagePixelData(UIImage *inImage)
{
    CGImageRef img = [inImage CGImage];
    CGSize size = [inImage size];

    CGContextRef cgctx = CreateARGBBitmapContext(img, size);
    if (cgctx == NULL) return NULL;

    CGRect rect = {{0,0},{size.width, size.height}};
    CGContextDrawImage(cgctx, rect, img);
    unsigned char *data = CGBitmapContextGetData (cgctx);
    CGContextRelease(cgctx);

    return data;
}

// Create an Image View that stores a copy of its image as an addressable bitmap
@interface BitMapView : UIImageView
{
    unsigned char *bitmap;
    CGSize size;
    UIView *colorView;
}
@end
@implementation BitMapView

// Hit test relies on the alpha level of the touched pixel
- (BOOL) pointInside:(CGPoint)point withEvent:(UIEvent *)event
{
    long startByte = (int)((point.y * size.width) + point.x) * 4;
    int alpha = (unsigned char) bitmap[startByte];
    return (alpha > 0.5);
}

- (void) setImage:(UIImage *) anImage
{
    [super setImage:anImage];
    bitmap = RequestImagePixelData(anImage);
    size = [anImage size];
}
@end

```

2.5 秘诀：检查多点触摸

通过在UIView中支持多点触摸交互，iPhone允许恢复和响应多手指交互。秘诀2-4演示如何向iPhone应用程序添加多点触摸（如图2-6所示）。

首先，将multipleTouchEnabled设置为YES或覆盖视图的isMultipleTouchEnabled。这

将通知应用程序一次检测多个UITouch。现在，当调用touchesForView:时，返回集可能包含若干个触摸。使用NSSet的allObjects方法将该集转换为可寻址的NSArray。当数组的计数超过1时，处理的就是多点触摸。

理论上，iPhone可以支持任意数量的触摸。实际中，多点触摸一次仅限于5个手指。甚至一次5个手指都已经超出了大多数开发人员的需要。同时用5个手指完成的有意义的手势并不多。当你一只手持着iPhone，而另一只手触摸时，就更不可能同时使用5个手指。但值得庆幸的是，如果需要使用多点触摸，iPhone也内置支持。遗憾的是，当一次使用3点或多点触摸时，屏幕会比较容易丢失对其中一个或多个手指的跟踪。当触摸的手指超过2时，编程跟踪连贯的手势会难以实现。

触摸没有分组。例如，如果你每个手用两个手指触摸屏幕，则无法确定哪个触摸来自哪只手。触摸顺序是任意的。虽然分组触摸在一个触摸事件（按下，移动，弹起）中保留相同的手指顺序，但其顺序可能会在用户下次触摸屏幕时发生更改。当你需要区分触摸时，可以构建一个按触摸对象索引的触摸字典。

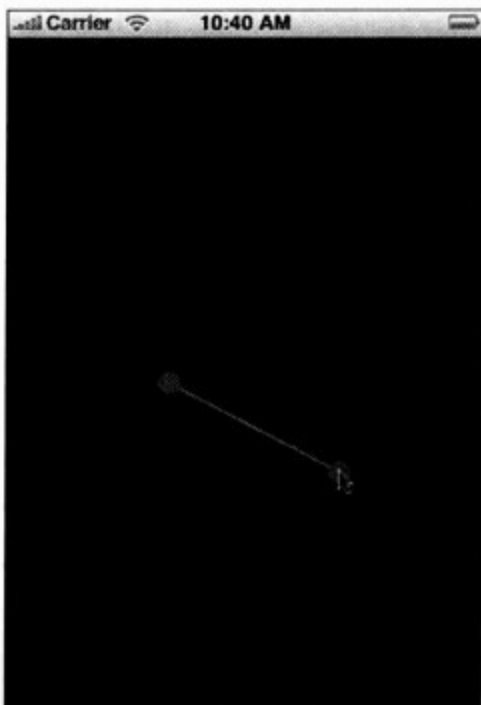


图2-6 iPhone支持捕获多点触摸事件和单点触摸事件。本例中将使用两个圆标记用户已经触摸的屏幕点

说明 秘诀2-4中的drawRect:例程将在每次调用时明确其上下文。这将移除之前的圆和线。如果要查看事件轨迹，请注释掉此行。

秘诀2-4 可视化多点触摸

```
@interface MultiTouchView : UIView
{
    CGPoint loc1, loc2;
}
@property (nonatomic) CGPoint loc1;
@property (nonatomic) CGPoint loc2;
@end

@implementation MultiTouchView
@synthesize loc1;
@synthesize loc2;

- (BOOL) isMultipleTouchEnabled {return YES;}

- (void) touchesBegan:(NSSet*)touches withEvent:(UIEvent*)event
{
    NSArray *allTouches = [touches allObjects];
    int count = [allTouches count];
```



```
    if (count > 0) loc1 = [[allTouches objectAtIndex:0] locationInView:self];
    if (count > 1) loc2 = [[allTouches objectAtIndex:1] locationInView:self];
    [self setNeedsDisplay];
}

// React to moved touches the same as to "began"
- (void) touchesMoved:(NSSet*)touches withEvent:(UIEvent*)event
{
    [self touchesBegan:touches withEvent:event];
}

- (void) drawRect:(CGRect) aRect
{
    // Get the current context
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextClearRect(context, aRect);

    // Set up the stroke and fill characteristics
    CGContextSetLineWidth(context, 3.0f);
    CGFloat gray[4] = {0.5f, 0.5f, 0.5f, 1.0f};
    CGContextSetStrokeColor(context, gray);
    CGFloat red[4] = {0.75f, 0.25f, 0.25f, 1.0f};
    CGContextSetFillColor(context, red);

    // Draw a line between the two location points
    CGContextMoveToPoint(context, loc1.x, loc1.y);
    CGContextAddLineToPoint(context, loc2.x, loc2.y);
    CGContextStrokePath(context);

    CGRect p1box = CGRectMake(loc1.x, loc1.y, 0.0f, 0.0f);
    CGRect p2box = CGRectMake(loc2.x, loc2.y, 0.0f, 0.0f);
    float offset = -8.0f;

    // circle point 1
    CGMutablePathRef path = CGPathCreateMutable();
    CGPathAddEllipseInRect(path, NULL, CGRectInset(p1box, offset, offset));
    CGContextAddPath(context, path);
    CGContextFillPath(context);
    CFRelease(path);

    // circle point 2
    path = CGPathCreateMutable();
    CGPathAddEllipseInRect(path, NULL, CGRectInset(p2box, offset, offset));
    CGContextAddPath(context, path);
    CGContextFillPath(context);
    CFRelease(path);
}

@end
```


说明 苹果公司在其开发人员网站上提供了许多Core Graphics/Quartz 2D资源。虽然这些论坛、邮件列表和源代码示例不是iPhone所特有的，但它们为你扩展iPhone Core Graphics知识提供了宝贵的资源。

2.6 UIView 动画

UIView动画提供了一种新颖别致的操作iPhone开发人员平台的方式。它支持你在更新视图，生成平滑动画结果（用于增强用户体验）时放慢变化。最出色的一点是，所有这些都不需要你做太多工作。

UIView动画对于在视图的当前状态和更改后的状态之间构建可视桥梁非常理想。借助它们，可以加强可视变化，并创建将这些变化链接在一起的动画。动画变化包括以下方面：

- ❑ 位置变化——在屏幕上移动视图；
- ❑ 大小变化——更新视图边框；
- ❑ 透明度变化——提示视图的alpha值；
- ❑ 对视图应用的旋转或其他仿射转换的变化。

构建 UIView 动画块

UIView动画以块的形式工作，即每次执行一项完整的事务。块以beginAnimations:context:调用开始，以commitAnimations结束。这些类方法将被发送给UIView，而不是各个视图。在位于这两个调用之间的块中，定义动画运行的方式并执行实际的视图更新。你将使用的动画控件包括以下几个。

- ❑ **beginAnimations:context**。标记动画块的开始。
- ❑ **setAnimationCurve**。定义动画加速及减速的方式。除非有不可抗拒的理由，否则应使用渐入/渐出（UIViewAnimationCurveEaseInOut），而不是选择其他曲线。其他曲线类型包括渐入（加速进入动画）、线性（无动画加速）和渐出（加速移出动画）。渐入/渐出提供感觉最自然的动画样式。
- ❑ **setAnimationDuration**。以秒为单位指定动画的长度。这一点非常好。你可以将动画扩展为自己需要的长度。但一定要考虑到用户的耐心是有限的，保持动画长度不超过1秒或2秒。
- ❑ **commitAnimations**。标记动画块的结束。

将实际视图更改命令置于动画细节设置之后和动画结束之前。代码清单2-8显示了实际的UIView动画，它设置了动画曲线和动画时长（此处为1秒）。实际动画是一个透明度更新。内容视图的alpha值变为0，使它不可见。与视图直接消失不同，动画块减慢了变化速度并让它逐渐淡出视线。

说明 苹果公司通常使用两个连续的动画块在动画中添加反弹效果。例如，他们可能会过多放大某个视图，然后使用第二个动画将放大的视图缩回其最终大小。使用“反弹”可以丰富动画块应确保动画没有重叠，还要添加延时让第二个动画直到第一个动画结束时才启动（performSelector: withObject: afterDelay:），或分配一个动画委托回调（animationDidStop: finished:）来捕获第一个动画的结束并启动第二个动画。

代码清单2-8 使用UIView动画调用

```
[UIView beginAnimations:nil context:context];
[UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
[UIView setAnimationDuration:1.0];
[contentView setAlpha:0.0f];
[UIView commitAnimations];
```

2.7 秘诀：淡入和淡出视图

有时，你希望在屏幕上添加一些信息覆盖在视图上，但它们本身不执行任何操作。例如，你可以显示一个积分表或一些指令，或者提供一个上下文感知的工具提示。秘诀2-5演示了如何使用UIView动画块缓慢淡出或淡入非交互式覆盖视图。

其实现方法是创建一个自定义ToggleView。根据此代码的定义，ToggleView是带有一个子图像视图的UIView。轻击它时，动画块会切换alpha设置（从关到开，或从开到关）。其要点如下。

- ❑ 确保子视图不查找交互事件。Cocoa Touch不允许透明视图捕获触摸。因此，必须允许父视图ToggleView处理所有用户交互。在创建子视图时，方法会将子视图的userInteractionEnabled属性设置为NO。
- ❑ 确保仅捕获鼠标按下事件。对于简单的开关式切换，仅捕获和响应按压（press），实现最自然的用户反馈。否则，用户轻击将隐藏图像视图，然后再次立即显示。
- ❑ 选取一个合理的动画时间。如果动画时间超过了用户的意愿，则结果是在第一个动画完成之前处理新轻击事件。此处所使用的1秒时长的动画应该是最大的限度了。对于常见的界面更改，半秒或1/4秒比较合适。

秘诀2-5 使用UIView动画实现透明度更改

```
@interface ToggleView: UIView
{
    BOOL isVisible;
    UIImageView *imgView;
}
@end

@implementation ToggleView
- (id) initWithFrame: (CGRect) aFrame;
{
```

```

self = [super initWithFrame:aFrame];
isVisible = YES;
imageView = [[UIImageView alloc] initWithFrame:[UIScreen mainScreen]
➡applicationFrame]];
[imageView setImage:[UIImage imageNamed:@"alphablend.png"]];
imageView.userInteractionEnabled = NO;
[self addSubview:imageView];
[imageView release];
return self;
}

- (void) touchesBegan:(NSSet*)touches withEvent:(UIEvent*)event
{
    // only respond to mouse down events
    UITouch *touch = [touches anyObject];
    if ([touch phase] != UITouchPhaseBegan) return;

    isVisible = !isVisible;

    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];
    [imageView setAlpha:(float)isVisible];
    [UIView commitAnimations];
}

- (void) dealloc
{
    [imageView release];
    [super dealloc];
}

@end

```

2.8 秘诀：交换视图

UIView动画块并未限制为单一更改。秘诀2-6结合边框大小更新与透明度更改以创建更具吸引力的动画。其方法是在动画块中一次添加若干指令。秘诀2-6一次执行4个操作。它放大并淡入一个视图，同时缩小并淡出另一个视图。图2-7提供了此动画的效果预览。

秘诀2-6 在动画块中结合多个视图更改

```

@interface ToggleView: UIView
{
    BOOL isOne;
    UIImageView *imageView1, *imageView2;
}

@end

```



```
@implementation ToggleView

#define BIGRECT CGRectMake(0.0f, 0.0f, 320.0f, 435.0f)
#define SMALLRECT CGRectMake(130.0f, 187.0f, 60.0f, 60.0f)

- (id) initWithFrame: (CGRect) aFrame;
{
    self = [super initWithFrame:aFrame];

    // Load both views, make them noninteractive
    UIImageView *imgView1 = [[UIImageView alloc] initWithFrame:BIGRECT];
    UIImageView *imgView2 = [[UIImageView alloc] initWithFrame:SMALLRECT];
    [imgView1 setImage:[UIImage imageNamed:@"one.png"]];
    [imgView2 setImage:[UIImage imageNamed:@"two.png"]];
    imgView1.userInteractionEnabled = NO;
    imgView2.userInteractionEnabled = NO;

    // image 1 is in front of image 2 to begin
    [self addSubview:imgView2];
    [self addSubview:imgView1];
    isOne = YES;
    [imgView1 release];
    [imgView2 release];
    return self;
}

- (void) touchesBegan: (NSSet*)touches withEvent: (UIEvent*)event
{
    // Determine which view occupies which role
    UIImageView *big = isOne ? imgView1 : imgView2;
    UIImageView *little = isOne ? imgView2 : imgView1;
    isOne = !isOne;

    // Pack all the changes into the animation block
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];

    [big setFrame:SMALLRECT];
    [big setAlpha:0.5];
    [little setFrame:BIGRECT];
    [little setAlpha:1.0];

    [UIView commitAnimations];

    // Hide the shrunken "big" image.
    [big setAlpha:0.0f];
    [[big superview] bringSubviewToFront:big];
}
```

解
乎
難

```
-(void) dealloc
{
    [imgView1 release];
    [imgView2 release];
    [super dealloc];
}
@end
```

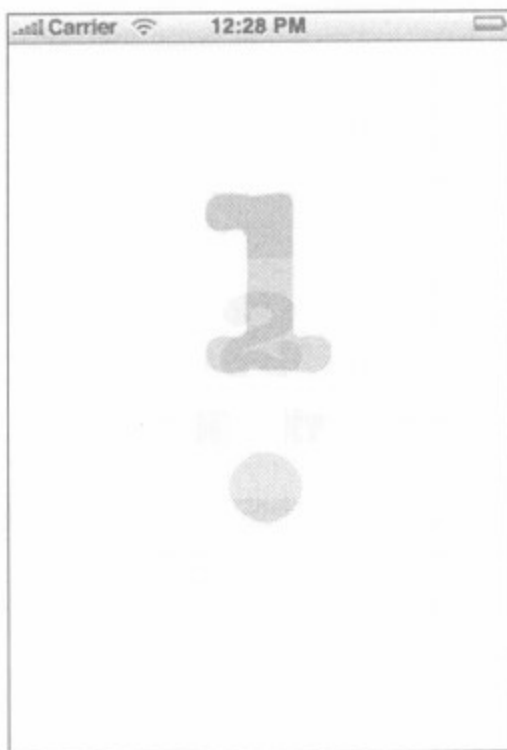


图2-7 在一个UIView动画块内部发起若干视图更改可以创建复杂的可视效果

2.9 秘诀：翻转视图

可以通过转换扩展UIView动画块，以添加更具可视性的功能。UIViewAnimationTransitionFlipFromLeft和UIViewAnimationTransitionFlipFromRight这两个转换的作用正如其名，使用它们可以将视图向左或向右翻转。它们是UIView中仅有的两个可用的官方转换。

说明 在SDK Beta版期间，苹果公司承诺的额外动画从未实现，特别是UIViewAnimationTransitionCurlUp和UIViewAnimationTransitionCurlDown。这些额外的动画可能会在未来得以实现。

要在UIView动画块中使用转换，需要做两个工作。首先，必须将转换作为块参数添加。使用setAnimationTransition:将转换分配给封闭的UIView动画块。其次，应该在块内部重新安排视图顺序。最好使用exchangeSubviewAtIndex:withSubviewAtIndex:来完成这项工作。秘诀2-7演示了如何使用这些技巧创建一个简单的翻转视图。轻击时，视图将使用动画从一端翻转到另一端，如图2-8所示。

不要混淆UIView动画块与Core Animation中的CATransition类。遗憾的是,不能向UIView动画分配CATransition。要使用CATransition,必须将它应用到某个UIView层,详见下一秘诀。

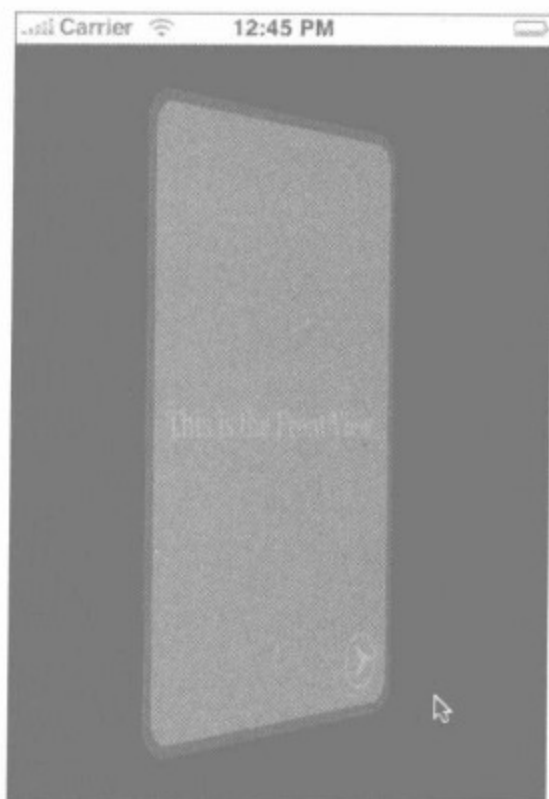


图2-8 使用UIView的内置转换动画翻转视图

秘诀2-7 在UIView动画块中使用转换

```
@interface FlipView : UIImageView
@end

@implementation FlipView
- (void) touchesEnded:(NSSet*)touches withEvent:(UIEvent*)event
{
    // Start Animation Block
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationTransition: UIViewAnimationTransitionFlipFromLeft
    ➡forView:[self superview] cache:YES];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];

    // Animations
    [[self superview] exchangeSubviewAtIndex:0 withSubviewAtIndex:1];

    // Commit Animation Block
    [UIView commitAnimations];
}
@end
```


2.10 秘诀：将 CATransition 应用于层

Core Animation Transition与UIView动画相似，仅在实现方面稍有不同。CATransition作用于层，而非视图。层是与各UIView相关的Core Animation呈现平面。在操作Core Animation时，将CATransition应用于某个视图的默认层（[myView layer]），而不是视图本身。

不用像操作UIView动画那样通过UIView设置参数。首先创建一个Core Animation对象，设置其参数，然后将参数化的转换添加到层中。代码清单2-9显示了一个简单的pushFromLeft方法，可以使用它替换秘诀2-7中的flip方法。

动画使用类型和子类型。类型定义所使用的转换类型，子类型设置其方向。类型和子类型共同指定当你将动画应用于视图时，视图应如何反应。

Core Animation Transition有别于之前秘诀中所讨论的两个UIView翻转。Cocoa Touch提供了4种类型的Core Animation。这些可用类型包括交叉淡入淡出、推送（在代码清单2-9中使用）、展现（一个视图滑出另一个视图）和覆盖（一个视图滑入到另一个视图上）。最后3种类型支持你通过子类型指定转换运动的方向。显然，交叉淡入淡出没有方向，也不使用子类型。

Core Animation是Quartz Core框架的一部分。要使用此示例，必须将Quartz Core框架添加到项目中，并将<QuartzCore/QuartzCore.h>导入到代码中。

说明 苹果公司的Core Animation的一大特色是围绕Objective-C类构建的2D和3D例程。这些类为iPhone和Macintosh应用程序提供图形呈现和动画。Core Animation避免了许多低级开发细节（例如，与直接OpenGL相关的细节），同时保留了操作分层视图的简洁性。

代码清单2-9 向UIView层添加Core Animation Transition

```
@implementation PushView
- (void) touchesEnded: (NSSet*) touches withEvent: (UIEvent*) event
{
    CATransition *animation = [CATransition animation];
    [animation setDelegate:self];
    [animation setDuration:1.0f];
    [animation setTimingFunction:UIViewAnimationCurveEaseInOut];
    [animation setType: kCATransitionPush];
    [animation setSubtype: kCATransitionFromLeft];

    [[self superview] exchangeSubviewAtIndex:0 withSubviewAtIndex:1];
    [[[self superview] layer] addAnimation:animation
    forKey:@"transitionViewAnimation"];
}
@end
```

2.10.1 文档中未记录的动画类型

iPhone实际实现的动画类型比官方文档叙述的要多。如代码清单2-10所示，iPhone完全具备

在Google Maps应用程序中处理地图曲线的能力。在iPhone而非仿真器上运行的此代码需要从UIKit二进制框架文件提取动画名称。

与所有文档中未记录的调用类似,这也有风险。苹果公司可以随时更改或删除这些动画。其他动画类型包括pageCurl、pageUnCurl、suckEffect、spewEffect、cameraIris(来自Photos应用程序)、cameraIrisHollowOpen、cameraIrisHollowClose、genieEffect(通常用于删除垃圾)、unGenieEffect、rippleEffect、twist、tubey、swirl、charminUltra、zoomyIn、zoomyOut和oglFlip。

注意setRemovedOnCompletion: NO的使用。这将在动画结束时冻结它,允许翻卷的地图保持可见,如图2-9所示。

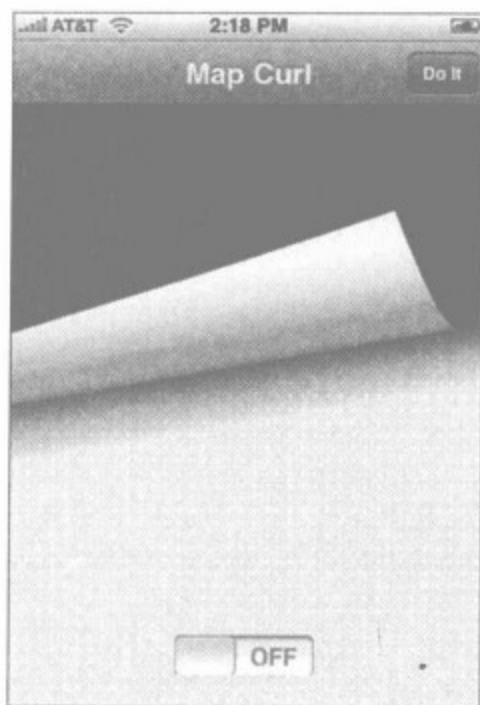


图2-9 这种引人注目的效果使用文档中未记录的Core Animation, 名为mapCurl

代码清单2-10 调用文档中未记录的动画类型

```
- (void) performCurl
{
    // Curl the image up or down
    CATransition *animation = [CATransition animation];
    [animation setDelegate:self];
    [animation setDuration:1.0f];
    [animation setTimingFunction:UIViewAnimationCurveEaseInOut];
    [animation setType:(notCurled ? @"mapCurl" : @"mapUnCurl")];
    [animation setRemovedOnCompletion:NO];
    [animation setFillMode: @"extended"];
    [animation setRemovedOnCompletion: NO];
    notCurled = !notCurled;

    [[topView layer] addAnimation:animation forKey:@"pageFlipAnimation"];
}
```

2.10.2 通用 Core Animation 调用

iPhone提供了对Core Animation调用的部分支持。部分的意思是许多标准类在实际应用中都缺失了。CIFilter就是其中的一个类。它并未包括在Cocoa Touch中,但CALayer和CATransition都是过滤器感知的。如果你愿意解决这些限制,则可以在程序中随意使用标准Core Animation调用。

代码清单2-11显示了iPhone本机Core Animation代码,该代码基于Lucas Newman (<http://lucasnewman.com>) 提供的一个示例。在运行时,该方法会缩小并淡出UIImageView的内容。源将在视图后面添加一个透明反射层。

实际上,此代码从Mac OS X示例(代码以之为基础)摘录过来就没有修改过。较复杂的Core Animation示例可能会带来迁移方面的挑战,但对于简单的反射、阴影和转换,需要的所有功能在本机iPhone层面上都存在。

代码清单2-11 本机iPhone Core Animation调用

```
// Adapted from http://lucasnewman.com/animationsamples.zip
- (void) scaleAndFade
{
    // create the reflection layer
    CALayer *reflectionLayer = [CALayer layer];

    // share the contents image with the screen layer
    reflectionLayer.contents = [contentView layer].contents;
    reflectionLayer.opacity = 0.4;
    reflectionLayer.frame = CGRectOffset([contentView layer].frame, 0.5,
    ➤ 416.0f + 0.5);
    // flip the y-axis
    reflectionLayer.transform = CATransform3DMakeScale(1.0, -1.0, 1.0);
    reflectionLayer.sublayerTransform = reflectionLayer.transform;
    [[contentView layer] addSublayer:reflectionLayer];

#define ANIMATION_DURATION (4.0)

    [CATransaction begin];
    [CATransaction setValue:[NSNumber numberWithFloat:ANIMATION_DURATION]
    ➤ forKey:kCATransactionAnimationDuration];

    // scale it down
    CABasicAnimation *shrinkAnimation = [CABasicAnimation
    ➤ animationWithKeyPath:@"transform.scale"];
    shrinkAnimation.timingFunction = [CAMediaTimingFunction
    ➤ functionWithName:kCAMediaTimingFunctionEaseIn];
    shrinkAnimation.toValue = [NSNumber numberWithFloat:0.0];
    [[contentView layer] addAnimation:shrinkAnimation forKey:@"shrinkAnimation"];

    // fade it out
    CABasicAnimation *fadeAnimation = [CABasicAnimation
    ➤ animationWithKeyPath:@"opacity"];
```



```

fadeAnimation.toValue = [NSNumber numberWithFloat:0.0];
fadeAnimation.timingFunction = [CAMediaTimingFunction functionWithName:
    kCAMediaTimingFunctionEaseIn];
[[contentView layer] addAnimation:fadeAnimation forKey:@"fadeAnimation"];

[CATransaction commit];
}

```

2.11 秘诀：滑动视图

滑动 (swipe) 是一个便利但经常被忽略的 iPhone 交互样式。当用户的手指快速在屏幕上拖动时，针对该手势返回的 `UITouch` 对象将包括一个 `info` 属性。该属性定义用户滑过屏幕的方向（如上、下、左或右）。此行为在 iPhone 的 Photos 应用程序中经常可见，用户通过向左或向右滑动在相册图片之间来回移动。

早期版本的 iPhone SDK 将滑动检测作为 `UITouch` 对象的一个标准部分提供，但较新发行版都摒弃了此功能。而苹果公司是在其 “iPhone Developers Guide” 中提供了替代的解决代码。秘诀 2-8 以该代码为基础。它定义了一个移动安全区，确保用户手指在区内仍朝一个方向移动。如果用户在对角线方向上越过路径超过 6 像素，则滑动操作将被取消。沿路径至少移动 12 像素才算一次成功的滑动操作。

秘诀 2-8 对滑动操作的成功完成应用了一个 `Core Animation Transition`。它使用滑动方向设置动画的子类型。子类型在 `Core Animation` 中用于指定动画的总体移动，包括向上、向下和斜向。

此示例模拟用于浏览 Photos 中的相册图片的交互样式，但允许向上、向下、向左和向右移动。如果注释掉 `kCATransitionPush` 动画类型，并在其后的线条中将它替换为文档中未记录的 `oglFlip`，你会惊喜地发现情况变得更加完美。翻转不再仅限于两个核心方向，虽然苹果公司并未将其中一个包括在公共 SDK 中，iPhone 实际上支持完整的四向翻转样式。

说明 在早期版本的 iPhone SDK 中，滑动无法在仿真器中生效。在随后的版本中，它们才得到了支持。如果你在开发时遇到平台限制（如在操作 Camera 时），可以轻松通过测试平台添加解决方案。在源代码中添加编译器指令，如 `#if defined(TARGET_IPHONE_SIMULATOR)`。

秘诀 2-8 检测和响应视图中的用户滑动

```

- (CATransition *) getAnimation:(NSString *) direction
{
    CATransition *animation = [CATransition animation];
    [animation setDelegate:self];
    [animation setType:kCATransitionPush];
    // [animation setType:@"oglFlip"];
    [animation setSubtype:direction];
    [animation setDuration:1.0f];
    [animation setTimingFunction:[CAMediaTimingFunction
        functionWithName:kCAMediaTimingFunctionEaseInEaseOut]];
}

```

```
        return animation;
    }

#define HORIZ_SWIPE_DRAG_MIN 12
#define VERT_SWIPE_DRAG_MAX 4

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    startTouchPosition = [touch locationInView:self];
    dirString = NULL;
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = touches.anyObject;
    CGPoint currentTouchPosition = [touch locationInView:self];

    if (fabsf(startTouchPosition.x - currentTouchPosition.x) >=
        HORIZ_SWIPE_DRAG_MIN &&
        fabsf(startTouchPosition.y - currentTouchPosition.y) <=
        VERT_SWIPE_DRAG_MAX)
    {
        // Horizontal Swipe
        if (startTouchPosition.x < currentTouchPosition.x) {
            dirString = kCATransitionFromLeft;
        }
        else
            dirString = kCATransitionFromRight;
    }
    else if (fabsf(startTouchPosition.y - currentTouchPosition.y) >=
        HORIZ_SWIPE_DRAG_MIN &&
        fabsf(startTouchPosition.x - currentTouchPosition.x) <=
        VERT_SWIPE_DRAG_MAX)
    {
        // Vertical Swipe
        if (startTouchPosition.y < currentTouchPosition.y)
            dirString = kCATransitionFromBottom;
        else
            dirString = kCATransitionFromTop;
    }
    else
    {
        // Process a non-swipe event.
        // dirString = NULL;
    }
}

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

```
{
    if (dirString)
    {
        CATransition *animation = [self getAnimation:dirString];
        [[self superview] exchangeSubviewAtIndex:0 withSubviewAtIndex:1];
        [[[self superview] layer] addAnimation:animation forKey:kAnimationKey];
    }
}
```

2.12 秘诀：转换视图

仿射转换支持更改某个对象的几何特征，方法是将该对象从一个视图坐标系映射到另一个视图坐标系。iPhone SDK完全支持标准的仿射2D转换。借助它们，可以缩放、转换、旋转和倾斜视图，而与应用程序的需求无关。

转换在Core Graphics中定义，包括CGAffineTransformMakeRotation和CGAffineTransformScale等调用。它们构建并修改 3×3 转换矩阵。构建之后，使用UIView的setTransform:调用将2D仿射转换应用于UIView对象。

秘诀2-9演示了如何构建和应用UIView的仿射转换。在创建示例时，我尽量保持简单。我构建了一个NSTimer，每隔1/30秒计时一次。在计时过程中，视图在余弦曲线上旋转 $\pi/100$ 。考虑到两个原因，我使用的是余弦的绝对值。一是视图始终保持可见，NSTimer在缩放更改检测时提供了良好的反弹效果。二是这将生成一种旋转和无阻尼的反弹动画。

在这些示例中，这是最适合构建也是最适合查看的代码。你将能够更好地看到handleTimer:方法如何与当前的可见效果相关联。

秘诀2-9 UIView仿射转换示例

```
#import "math.h"
#define PI 3.14159265

@interface HelloController : UIViewController
{
    UIView *contentView;
    UIImageView *rotateView;
    int theta;
}
@end

@implementation HelloController
- (id)init
{
    if (self = [super init]) self.title = @"Affine Demo";
    return self;
}
```



```
- (void) handleTimer: (NSTimer *) timer
{
    // Rotate each iteration by 1% of PI
    float angle = theta * (PI / 100);
    CGAffineTransform transform = CGAffineTransformMakeRotation(angle);
    theta = (theta + 1) % 200;
    // For fun, scale by the absolute value of the cosine
    float degree = cos(angle);
    if (degree < 0.0) degree *= -1.0f;
    degree += 0.5f;
    CGAffineTransform scaled = CGAffineTransformScale(transform, degree, degree);

    // Apply the affine transform
    [rotateView setTransform:scaled];
}

- (void)loadView
{
    theta = 0;

    contentView = [[UIView alloc] initWithFrame:[UIScreen mainScreen]
    ➡applicationFrame];
    rotateView = [[UIImageView alloc] initWithFrame:CGRectMake(0.0f, 0.0f, 240.0f,
    ➡240.0f)];
    [rotateView setImage:[UIImage imageNamed:@"rotateart.png"]];
    [rotateView setCenter:CGPointMake(160.0f, 208.0f)];
    [contentView addSubview:rotateView];
    [rotateView release];

    self.view = contentView;
    [contentView release];

    [NSTimer scheduledTimerWithTimeInterval: 0.03f target: self selector:
    ➡@selector(handleTimer:)
    userInfo: nil repeats: YES];
}

// Allow the view to respond to iPhone Orientation changes
- (BOOL)shouldAutorotateToInterfaceOrientation: (UIInterfaceOrientation)
interfaceOrientation
{
    return NO;
}

- (void) dealloc
{
    [contentView release];
    [rotateView release];
    [super dealloc];
}
```

```

}
@end

```

居中放置横向视图

使用相同的仿射转换方法居中放置横向的视图。代码清单2-12将创建一个 480×320 像素的视图，并将其居中放置到 $[160, 240]$ （使用纵向视图坐标），然后再将其旋转到位。 $\pi/2$ 对应于90度，即创建横向向右旋转。居中可保证整个视图位于屏幕上。所有子视图，包括文本字段、标签、开关等，都将与父视图一并旋转。

如果你希望操作此视图的横向键盘，应确保调用`[[UIApplication sharedApplication] setStatusBarOrientation:UIInterfaceOrientationLandscapeRight]`。这将设置状态栏方向，用于控制键盘，而不管状态栏为隐藏还是显示。

代码清单2-12 将横向视图旋转到位

```

@implementation HelloController
- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:CGRectMake(0.0f, 0.0f, 480.0f,
        ➡320.0f)];
    [contentView setCenter:CGPointMake(160.0f, 240.0f)];
    [contentView setBackgroundColor:[UIColor blackColor]];
    [contentView setTransform:CGAffineTransformMakeRotation(3.141592f / 2.0f)];
    self.view = contentView;
    [contentView release];
}

- (void) dealloc
{
    [contentView release];
    [super dealloc];
}
@end

```

2.13 小结

UIView提供用户可以看到并能与之交互的屏幕组件。如本章所述，即使处于最基本的形式，它们也都提供了难以置信的灵活性和强大的功能。你了解了如何使用视图在屏幕上构建元素，创建多个交互对象，并引入引人瞩目的动画。在继续往下阅读本书之前，你可能希望回顾一下本章秘诀所呈现的要点。

- 在操作多个屏幕视图时，应始终优先考虑分层结构——这不是开玩笑。使用视图分层词汇（`bringSubviewToFront:`、`sendSubviewToBack:`、`exchangeSubviewAtIndex:withSubviewAtIndex:`）来处理视图，并始终将正确的可视内容呈现给用户。

- ❑ 并非仅限于使用矩形。使用UIView/Core Graphics剪辑可以创建富有吸引力的、没有直角的交互对象。
- ❑ 创造真实感。iPhone拥有绝佳的触摸屏幕。为何不让你的用户用手指在屏幕上拖动项目呢？这可以增加真实感和平台的交互性。
- ❑ 将所有事物制作成动画。动画不需要过于喧闹、华丽，也不能设计得太差。iPhone提供的对动画的强有力支持允许在用户任务之间添加平滑的转换。简洁、平滑、专注的更改是iPhone的根本特征。
- ❑ 用户每只手都有5个手指。有必要将交互扩展到多点触摸领域时，不要将界面限制在只能使用一个手指。
- ❑ Quartz图形和Core Animation的坚实基础对你大有帮助。使用drawRect:，可以构建任何种类的自定义UIView演示，包括文本、贝塞尔曲线和涂鸦等。
- ❑ 学会探索。本章仅涉及在应用程序中使用UIView的少量知识。将本章作为起点去学习完整UIView类的知识吧。



iPhone屏幕并不大，它的长宽都只有几英寸。这种小型屏幕限制了用户的物理视角，从而对程序造成了限制。虽然你的虚拟世界实际上非常大，但小型物理屏幕意味着iPhone类必须智能地扩展它们的实际交互空间。本章将探索使用视图控制器创建虚拟演示的多种方法，以扩大应用程序的物理空间。读者可以了解如何使用视图控制器构建可导航的世界，以及如何使用Interface Builder连接到这些类。

3.1 视图管理

在iPhone中，视图控制器采用集中化的基本视图管理。通过链接第2章中介绍的视图功能与设备实体，它们提供了实际的实用工具。视图控制器处理方向重设事件（例如，当用户倾斜iPhone进入横向模式时）和导航调用（例如，当用户需要在视图间转移时）。以下部分将介绍iPhone SDK中使用的各种视图控制器。你将看到视图控制器范型内使用了哪些类，包括核心类和专用类。

3.1.1 核心类

iPhone SDK提供了许多视图控制器及其子类。其范围涉及通用类和专用类。其中只有以下3个表示核心控制器类。

- ❑ **UIViewController**。这是视图控制器的父类，用于明确链接到一个视图。它服务于视图控制器，并且需要花费大量时间对其进行定制。最重要的是，UIViewController处理所有视图方向重设任务，支持你轻松地针对横向和纵向进行编程。
- ❑ **UINavigationController**。使用导航控制器在基于树的视图层次结构中上下浏览（例如数据钻取）。顾名思义，它们创建位于标准iPhone应用程序顶部的蓝色或灰色的导航栏。它们支持通过使用Navigation Items添加按钮，并自动生成“返回”按钮，显示调用视图控制器的标题。所有导航控制器都使用“根”视图控制器建立导航树的顶部。
- ❑ **UITabBarController**。并行视图类似于广播站。通过选项卡栏，用户可以选择“调频到”哪个UIViewController，而不必存在特定的导航层次。可以在YouTube和iPod等应用程序上看到它，用户选择是否查看“前25”列或查看相册及播放列表。当TabBar实例一次

提供超过5种视图控制器选择时，用户可以通过More > Edit屏幕来定制它们。More > Edit屏幕允许用户将自己喜爱的控制器拖到屏幕底部的按钮栏下方。

3.1.2 专用类

除了以上介绍的这3种通用类之外，苹果公司还提供了许多高度适用的控制器。这些专用项目利用对iPhone核心数据库及函数的访问。可以将这些类的实例抛入应用程序，并像任何其他UIViewController那样使用它们，但它们预先加载了高级功能。SDK中可用的一些专用视图控制器包括以下几个。

- ❑ **UIImagePickerController**。此实用工具控制器支持用户从机载相册中选择图像，或通过iPhone照相机拍照。苹果公司添加了一个高级图像选择界面。控制器甚至允许用户在结束之前翻转和缩放图像。
- ❑ **ABNewPersonViewController**。此控制器支持用户为他们的Address Book创建新联系人。其他Address Book控制器样式包括ABPersonViewController、ABUnknownPersonViewController和ABPeoplePickerNavigationController。
- ❑ **UITableViewController**。表视图控制器仅使用iPhone项目中的表。此控制器类提供了一个已经连接的标准UITableView实例，并自动将委托和数据源设置为指向自己。只需提供这些委托和数据源方法，在表中填充数据并响应用户轻击。我们将在第5章中深入讨论UITableViewController。
- ❑ **MPMoviePlayerController**。当用户需要看电影或听音频时，此控制器就派上了用场。只需向它提供一个到媒体源的路径，并展开其显示。控制器将为用户提供一个Done按钮，或在播放完成时自动返回委托调用。

从某种方式上说，这些专用控制器是一把双刃剑。一方面，它们引入了众多功能，基本上没有额外的编程负担。另一方面，它们专用性太强，因此经常隐藏开发人员所需的核心特性。

举例来说，并没有简单的音频播放类。要么通过AudioQueue从最底层入手，处理采样率和编码样式，要么通过MPMoviePlayerController在最高层进行处理。无法访问通用scrubber栏或暂停/播放按钮。它们实际上是隐藏在Media Player框架中的特性，但是却没有公开定义的方法或类。它们以前是名为Celestial的公共框架，向音频可视控制器和便捷的重放特性提供类似于QuickTime的界面，但苹果公司在1.1.4版本后的固件中将此框架移到了Private Frameworks层次结构中。

3.1.3 创建 UIViewController

UIViewController实例提供了极佳的核心视图管理。它们只是没有提供实际视图。可以根据自己的需要创建一个，或者通过LoadView方法将它安装到视图控制器中。每一个UIViewController和子类都实现了此方法，可在其中将控制器连接到视图实例。需要执行两项任务：初始化视图，然后将它连接到控制器。

- ❑ 初始化内容视图，并调整其大小，方法是调用initWithFrame:并使用[[UIScreen

`mainScreen]` `applicationFrame]` 作为其参数。这将返回应用程序的标准范围。数学方法会自动考虑任何屏幕导航栏和选项卡栏。第1章中的Hello World示例提供了一个简单的实际布局。

- ❑ 将视图连接到控制器，方法是指定一个`self.view`给UIVIew实例。在`LoadView`方法中完成此操作（例如，`self.view = contentView`）。

在操作专用控制器时，要始终确保从子视图中调用子类版本的方法（即`[super loadView]`）。这将确保子视图继承父视图的布局行为以及类方法和变量。其他关键的`UIViewController`方法包括以下几个。

1. `init`

使用类的`init`方法设置视图控制器的标题和其他需要在实际使用类之前设置的特性。要使标题与应用程序名称保持一致，可以为`[[NSBundle mainBundle] infoDictionary]` `objectForKey:@"CFBundleName"]`指定`self.title`。这将返回在束的`Info.plist`文件中定义的名称。

与`UIView`不同，`UIViewController`实例是一个抽象的对象，不在物理屏幕上显示。不要像操作普通的`UIView`那样设置一个边框。`UIViews`使用`initWithFrame:`，`UIViewController`s使用`init`。在正确命名的`loadView`方法中设置任何视图边框。

2. `shouldAutorotateToInterfaceOrientation:`

此方法返回`YES`或者`NO`，这取决于是否希望在程序中使用自动旋转。返回`YES`时，视图控制器使用若干个标记确定自动旋转的方式。代码清单3-1显示了`LoadView`方法中一段典型的代码，它指定子视图应该自动在垂直和水平方向重新调整大小。

代码清单3-1 设置自动旋转标记

```
contentView.autoresizesSubviews = YES;
contentView.autoresizingMask = (UIViewAutoresizingFlexibleWidth |
    UIViewAutoresizingFlexibleHeight);
```

3. `viewDidAppear:`和`viewDidDisappear:`

每次`UIViewController`视图进入屏幕或离开屏幕时都会调用这两个方法——通常因为视图之间的导航。使用这些方法刷新视图，以反映当前数据状态并更新存储的数据。通过这些方法，可以声明或放弃文本元素的首要响应者状态，根据需要删除文本选择，或执行任何其他需要在将控制返回给用户之前完成的簿记。

4. `didRotateFromInterfaceOrientation:`

通过在`UIViewController`子类中实现此方法可以获取方向更改。通过此方法，可以响应新的方向并传递之前的方向（如果你希望根据所涉及的更改执行特定的动画类型）。从一个方向移动到另一个方向，而不经预期的中间阶段，这是完全可能的，因此也避免了假设。我们使用传递给此方法的实际数据。

说明 苹果公司已承诺在未来某个时刻在SDK中分别提供对横向和纵向视图的支持。在撰写本书时，此功能尚未实现。当方向发生改变时，需要重新调整屏幕。要强制应用程序进入横向模式，就对 `shouldAutorotateToInterfaceOrientation:` 返回 `NO`，调用 `applicationDidFinishLaunching:` 方法中的 `[[UIApplication sharedApplication] setStatusBarOrientation:UIInterfaceOrientationLandscapeRight]`，并在 `Info.plist` 文件中将 `UIInterfaceOrientation` 设置为 `UIInterfaceOrientationLandscapeRight`。

3.2 使用 Interface Builder 为 UIViewController 构建视图

借助其交互式GUI布局工具，Interface Builder可帮助创建UIView内容。使用它能够添加交互式控件，在屏幕上用手移动这些控件以设置独特的界面。Interface Builder在UIView布局方面极为出色。它未提供对UIViewController行为的直接支持(或者很少)。这是因为UIViewController是一个抽象类，而UIView是一个具体类。

考虑到这种限制，可以使用Interface Builder将UIView嵌入到UIViewController实例以及其他视图控制器类中。通过下面介绍的工作流可以结合Interface Builder设置工具的强大功能和视图控制器类的便利性。

3.2.1 温度转换器示例

此示例将结合Xcode与Interface Builder创建一个典型的华氏-摄氏温度转换器。UIView子类定义转换行为。此视图连接按钮和文本字段，并通过添加`convert:`函数执行数学方法来为它们赋予含义。

视图嵌入在一个普通的UIViewController子类中。该控制器通常负责接管所有自动旋转和视图导航任务。在本示例中，它实际上仅演示了如何使用Interface Builder工具创建视图以及将视图添加到控制器中。

在下面的步骤中，将创建一个新的Xcode项目，使用转换源代码填充它，然后使用Interface Builder的工具来布局GUI。

(1) 创建一个新的Xcode项目。

在Xcode中，选择File, New Project。创建一个新的Cocoa Touch Application View-Based项目，并将它命名为需要的名称。名称应尽量简短。Xcode和Interface Builder都使用项目名称定义应用程序委托，因此名称越简单越好。

(2) 清理项目并替换代码。

选择Classes子文件夹并删除它。这将保留`main.m`作为唯一的源文件。打开它并将其内容替换为代码清单3-2中的内容。更改应用程序委托的名称(此处为`WalkThroughAppDelegate`)以匹配项目名称。注意两个IBOutlet文本字段和IBAction返回类型。IBOutlet关键字指示对象要在Interface Builder中定义。IBAction会通知Interface Builder将方法视为目标/操作对中的操作。IBAction功能等价于(void)。

代码清单3-2 转换程序源代码

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

// Custom view that converts temperatures
@interface MyView : UIView {
    IBOutlet UITextField *infield;
    IBOutlet UITextField *outfield;
}
@end

@implementation MyView
- (IBAction) convert: (id) sender
{
    float inval = [[infield text] floatValue];
    float outval = (inval - 32.0f) * 5.0f / 9.0f;
    [outfield setText:[NSString stringWithFormat:@"%3.2f", outval]];
}
@end

// View controller that autorotates
@interface MyViewController : UIViewController
@end

@implementation MyViewController
- (void) loadView
{
    [super loadView];
    self.view.autoresizesSubviews = YES;
    self.view.autoresizingMask = (UIViewAutoresizingFlexibleWidth |
    UIViewAutoresizingFlexibleHeight);
}
- (BOOL) shouldAutorotateToInterfaceOrientation: (UIInterfaceOrientation)
interfaceOrientation
{
    return YES;
}
@end

// Application delegate that uses NIB objects
@interface WalkThroughAppDelegate : NSObject {
    IBOutlet UIWindow *window;
    IBOutlet MyView *contentView;
}
@end

@implementation WalkThroughAppDelegate
- (void) applicationDidFinishLaunching: (UIApplication *) application {
    [window addSubview:contentView];
}
```

```

        [window makeKeyAndVisible];
    }
- (void)dealloc {
    [contentView release];
    [window release];
    [super dealloc];
}
@end

int main(int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}

```

(3) 启动Interface Builder。

在项目中双击MainWindow.xib文件（参见图3-1）。这将在Interface Builder中打开它。

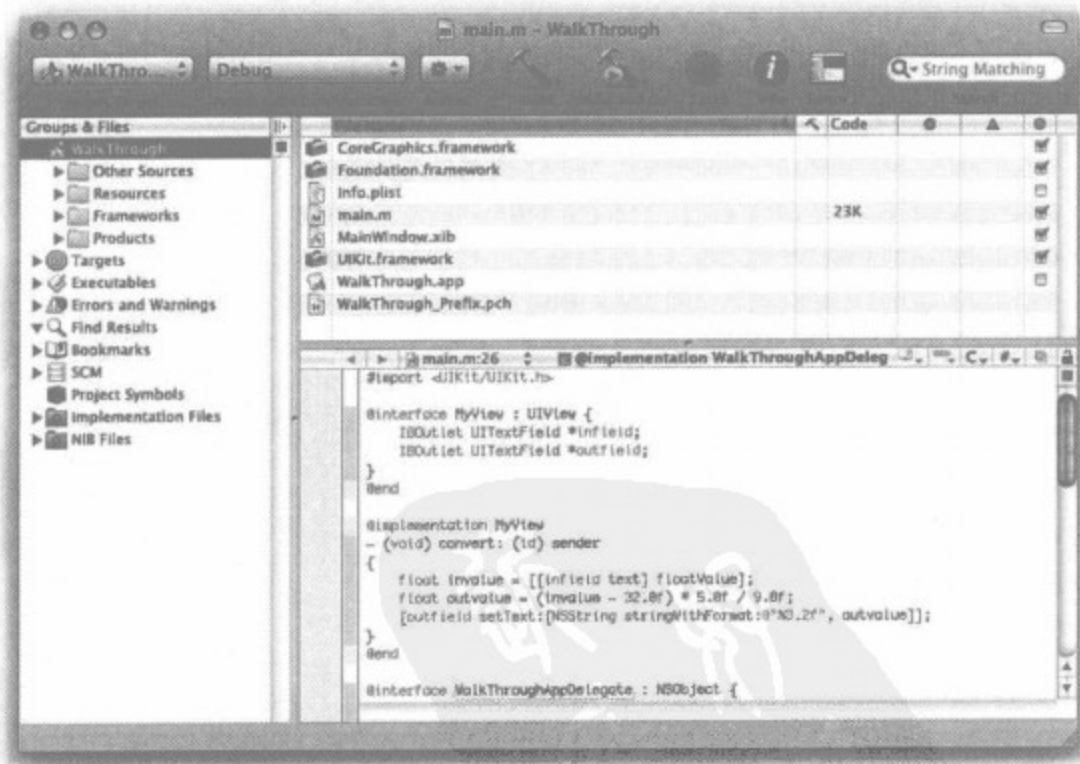


图3-1 Xcode将自动在项目中创建一个MainWindow.xib。双击该文件启动Interface Builder

(4) 设置窗口。

当Interface Builder启动时，MainWindow.nib项目包含5个对象：一个文件所有者、一个应用程序委托对象（使用项目名命名）、一个First Responder对象、一个视图控制器和一个窗口（参见图3-2）。

(5) 打开Inspector和Library。

选择Tools, Inspector (Command-Shift-I) 和Tools, Library (Command-Shift-L)。这两个窗口提供Interface Builder中的两种关键工具。

- ❑ Library提供一组可重用的Cocoa Touch对象。可以从Library窗口中将这些对象拖到应用程序中。
- ❑ 可以通过Inspector查看视图并在项目中定制界面对象。使用Inspector查看对象之间的连接关系, 以及它们所链接的类。

(6) 设置应用程序代理。

在主XIB窗口中轻击应用程序委托对象(并非文件的所有者或窗口)。然后, 轻击Inspector中最右侧的选项卡, 或者按下Command-4。这两个操作都将打开Identity Inspector。在Identity Inspector中, 在窗口顶部找到类名(参见图3-3)。

3

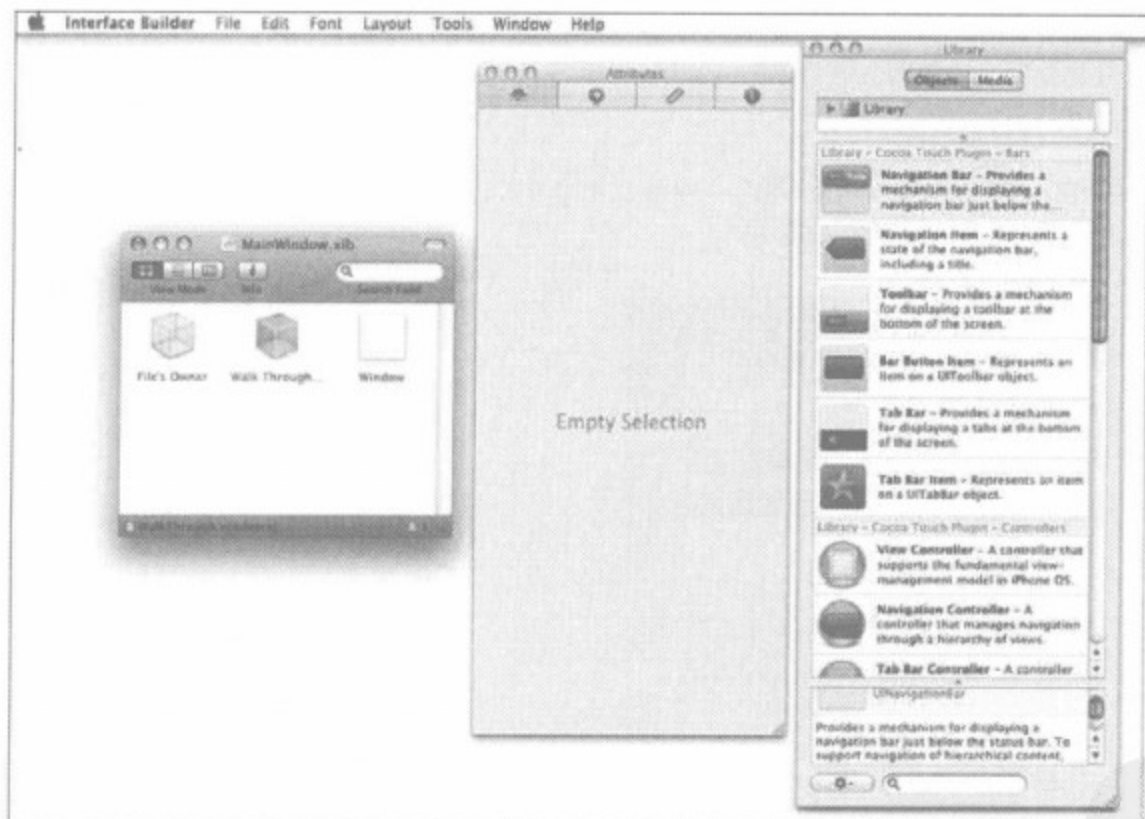


图3-2 默认Xcode XIB文件包含一个文件所有者、一个应用程序委托和一个窗口

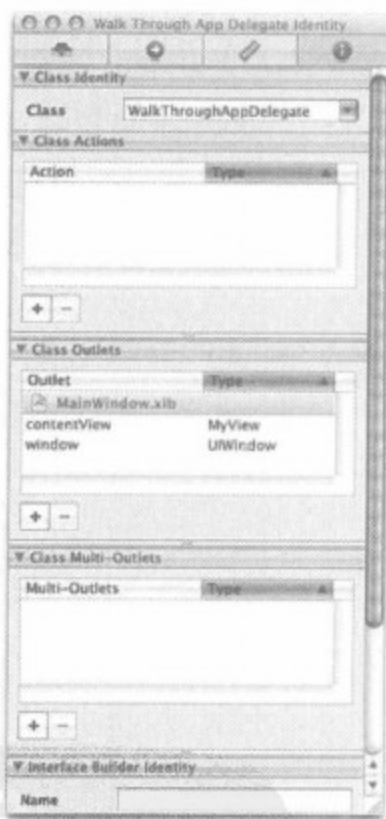


图3-3 使用Identity Inspector添加输出或操作, 并更新与Interface Builder对象相关的类名

如果你希望更改应用程序委托所使用的类名, 可以在此处更新它。

所有者、委托和窗口对象映射标准应用程序中使用的类。文件的所有者对象定义“拥有”XIB文件的类(即共享的UIApplication实例)。应用程序委托对象将连接到委托, 委托负责处理applicationDidFinishLaunching:回调, 设置主窗口和指定其显示方式。

(7) 向委托添加输出口。

添加两个输出口: window和contentView。将窗口类从id改为UIView, 并将视图类改为

MyView。键入这些输出口会向IB设置添加一层控制。在建立连接时，它们仅连接到适当类的对象。并不需要添加类类型（稍后将介绍不键入输出口如何实现），但开发人员应该知道这一点。

(8) 向项目窗口添加对象。

在Library中，找到UIViewController（位于库的顶部）和UIView（接近库的底部）。分别拖一个到项目窗口中。图3-4显示了添加对象后的项目窗口。这两个项目都使用此对象。这两个项目用于表示在MyView和MyViewController代码中定义的MyView类的实例。

(9) 更改View Controller类。

轻击新View Controller。在Identity Inspector（Command-4）中，将其类改为MyViewController。View Controller在项目窗口中的名称将更新为My View Controller。

(10) 更改视图类并添加其输出口和操作。

轻击新视图。使用Identity Inspector将其类改为MyView。在视图添加一个操作（convert:）和两个输出口（infield和outfield），如图3-5所示。



图3-4 从Library中将对象拖到XIB主窗口中，将这些类的实例添加到项目中

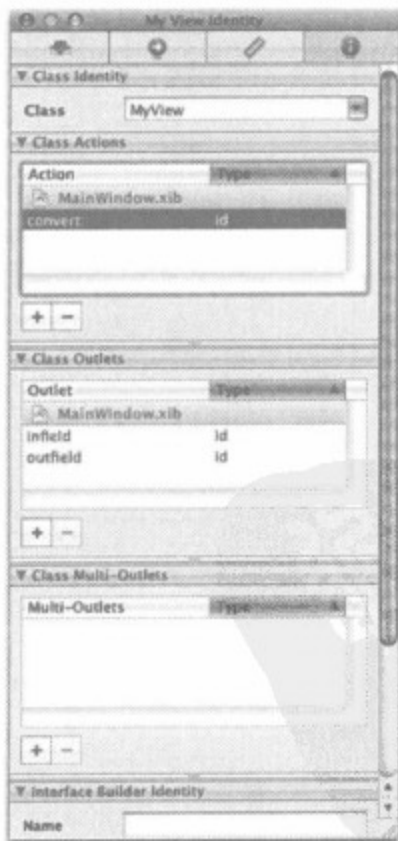


图3-5 可以通过Identity Inspector向对象添加输出口和操作。它们将响应源代码中的实例变量和实例方法

操作响应源代码中的实例方法，输出口响应实例变量。回顾步骤(2)中的代码，可以看到为MyView类定义的两个IBOutlet实例和一个用于执行转换的方法。在接下来的步骤中，你将了解

如何将屏幕对象连接到这些实例变量和方法。

(11) 编辑视图。

在项目窗口中双击My View，打开其编辑器。从Library中拖出两个标签、一个按钮和两个字段，如图3-6所示。

(12) 编辑视图。

模仿图3-7所示的布局调整视图中的元素。将项目向屏幕顶部放置，在底部留下足够的空间供键盘使用。

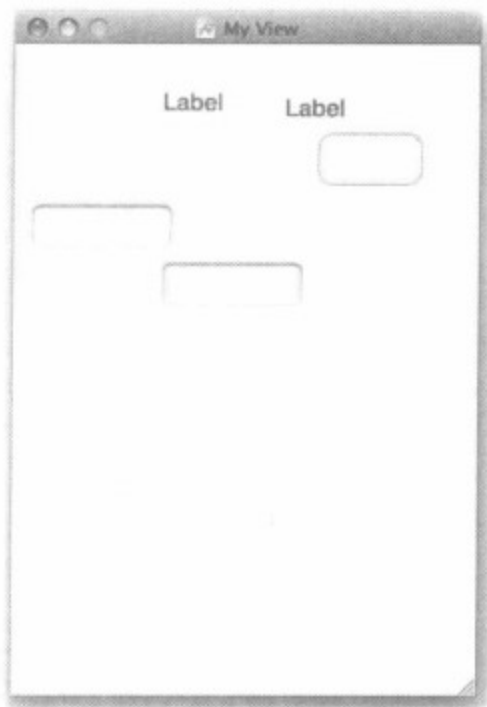


图3-6 从Interface Builder Library (Tools, Library) 中将界面元素拖动到视图上，开始布局其GUI

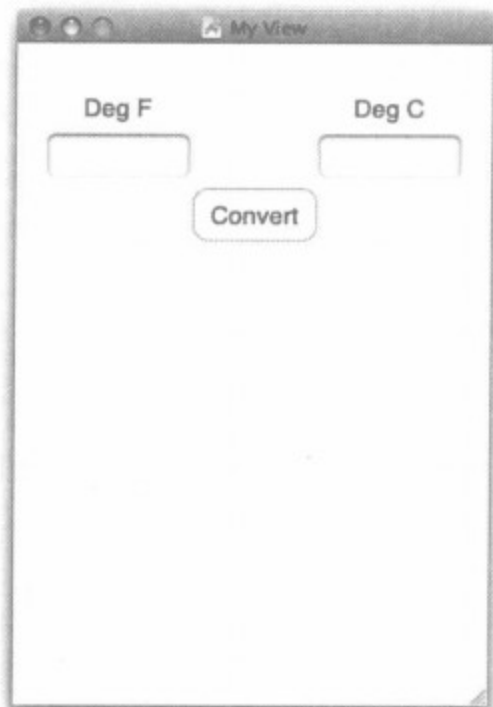


图3-7 将界面元素向屏幕顶部放置，将底部的空间留给键盘

3

如果愿意，使用内置的水平和垂直参考线 (Layout, Add Horizontal Guide and Layout, Add Vertical Guide) 来简化布局。这些参考线类似于Photoshop中的参考线。它们仅仅是虚拟的位置帮助工具。它们不会出现在最终应用程序中。如果选择使用它们，应注意它们在移动时显示的左右数值指示器。通过它们可以了解距其左侧及右侧有多少像素。160 | 160垂直参考线表示处于屏幕中央。

(13) 编辑左字段特性。

选择左 (输入) 字段并使用Attributes Inspector (Command-I) 将其键盘类型设置为数值和标点符号。可以自己花时间研究它的其他特性。

(14) 将应用程序委托连接到其窗口和视图。

在按下Control键的同时，从项目窗口中的应用程序委托对象拖到Window对象。从弹出菜单中选择Window。这会将Window对象分配给应用程序委托的window实例变量。

接下来，按下Control键的同时从应用程序委托拖到My View对象。从弹出菜单中选择contentView。

这些操作会设置应用程序委托的实例变量，在代码中声明的IBOutlet实例之间创建关联，并在Interface Builder中布局实际窗口和视图。

(15) 将视图连接到其输出口。

打开My View窗口使其可见。按下Control键的同时连接My View对象与左文本字段。从弹出的菜单中选择infield。然后，按下Control键的同时从视图拖到右文本字段，然后选择outfield。这次仍然是为实例变量分配对象。

(16) 将按钮连接到视图。

按下Control键的同时从在自定义视图中添加的Convert按钮拖到My View对象。此时将出现一个自定义弹出菜单。向下滚动弹出的菜单，找到Touch Up Inside，在XIB项目窗口中从圆向右拖到其右侧的MyView实例。从弹出的菜单中选择convert:。

此操作将按钮内部的触摸连接到在对象中定义的转换操作。再次右击Convert按钮，检查连接。弹出菜单将确认此连接，如图3-8所示。

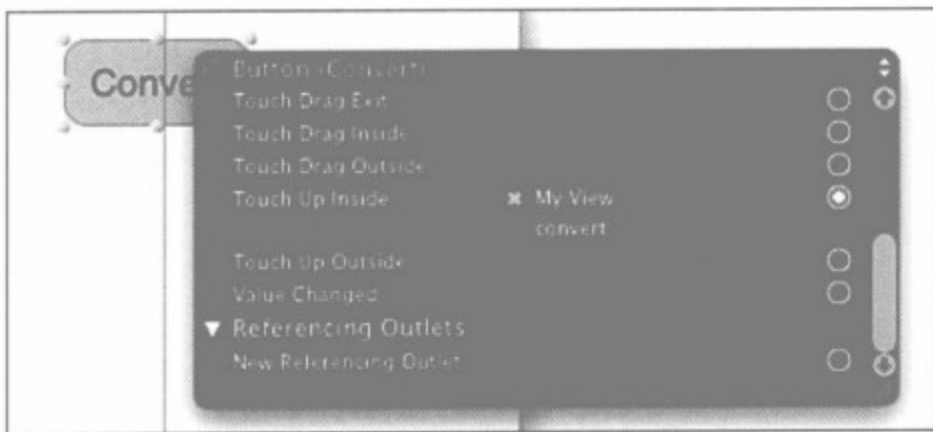


图3-8 完成连接之后，可以查看与触摸事件相关的目标和操作

(17) 将视图添加到其控制器。

双击My View Controller打开其编辑器。将项目窗口中的My View实例拖到此编辑器中。它会自动跳转到位。注意View Controller编辑器右上方的右向箭头。

这将允许在横向模式以及纵向模式中预览视图。继续操作，根据需要进行调整以提供满意的横向外观。

通过将视图分配给控制器，可以支持该视图控制器自动处理应用程序中的视图。不需要在Interface Builder中进行任何进一步的连接。

(18) 保存。

在Interface Builder中保存更改并返回Xcode。

(19) 运行。

在Xcode中，将活动SDK设置为设备或仿真器，然后运行已完成的应用程序。图3-9显示了最终的视图，视图中的元素在Interface Builder中完整定义，并连接到设计的一个简单的UIViewController。

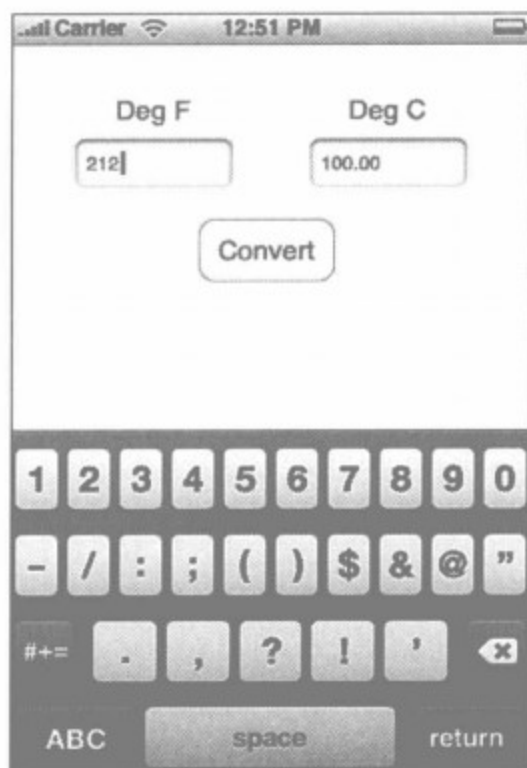


图3-9 完成后的应用程序将定制构建的视图绑定到指定的UIViewController

3.2.2 直接加载 XIB 文件

Cocoa Touch允许通过调用loadNibNamed: owner: options:从任何XIB文件恢复对象。这将返回在NIB/XIB束中初始化的一个对象数组，然后可以在程序中捕捉和使用它。

```
NSArray *niblets = [[NSBundle mainBundle] loadNibNamed:@"sample" owner:self
➡options:NULL];
for (id theObject in niblets) {
    if ([theObject isKindOfClass:[UIViewController class]])
        [self.navigationController pushViewController:theObject
➡animated:YES];
}
```

3.3 导航控制器

仅需要少量编程，UINavigationController类就可以提供基于UINavigationController的所有优点。借助导航控制器，可以使用内置动画顺利地在视图之间移动，还可以获得历史控制。控制器处理所有Back按钮功能。这意味着父视图标题会自动作为Back按钮出现，并且可以“弹出栈”，也就是说不需要任何编程。

如果以上优点还不足以满足你的要求，导航控制器还提供了一个简单的菜单栏。可以在栏中添加按钮（甚至更加复杂的控件），将操作构建到应用程序中。在导航、历史和菜单这3种特性之间，导航控制器为易于编写的包中增添了许多光彩。

随后的秘诀将介绍这些核心导航控制器特性，包括构建菜单来构建历史栈。在这些示例中，你将了解如何使用UINavigationController类创建各种不同的新特性和实用的界面。

3.3.1 设置导航控制器

无论你是否计划使用导航控制器简化视图的切换（其最初用途），都应该了解导航控制器的工作方式。从最简单的层次来说，导航控制器管理视图控制器栈。

每个导航控制器都拥有一个根视图控制器。该控制器形成了栈的基础。可以将其他控制器推入栈中。这会扩展导航浏览路径记录，并自动构建一个Back按钮。可以通过轻击其中一个Back按钮将控制器弹出栈。还可以继续执行出栈操作，直到到达根控制器。出栈操作到此为止，不能将根控制器弹出栈。

这种基于栈的设计比较耗时，即便计划只使用一个视图控制器。例如，你可能希望利用UINavigationController的内置导航栏来构建一个含有两个按钮的菜单。这将忽视栈的任何导航优势。仍然需要通过initWithRootViewController:将该控制器设置为根。

代码清单3-3显示了在普通applicationDidFinishLaunching:方法中创建和初始化导航控制器的过程。初始化时，将为该控制器分配一个根视图，然后通过其view属性将其添加到主窗口。

代码清单3-3 初始化导航控制器

```
- (void)applicationDidFinishLaunching:(UIApplication *)application
{
    UIWindow *window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen]
    ➤ bounds]];
    UINavigationController *nav = [[UINavigationController alloc]
    ➤ initWithRootViewController: [[HelloController alloc] init]];
    [window addSubview:nav.view];
    [window makeKeyAndVisible];
}
```

3.3.2 推入和弹出视图控制器

通过使用pushViewController:animated:推入新控制器，可以将新项目添加到导航栈中。将此调用发送给任何UIViewController的父导航控制器（如[self navigationController]）。推入后，新的控制器将滑入屏幕（假设你将animated设置为YES）。此时将出现一个左向Back按钮，使用它仅需一步就能返回栈中。Back按钮使用之前视图控制器的名称。

可能有许多原因促使你推入新视图。通常，这需要导航到子视图（如详细信息视图）或向下展开文件结构。一般会将控制器推入导航控制器栈中，然后再让用户轻击按钮、表项目或信息公开附件。

在UIViewController子类中执行推入请求和其他导航栏定制。没有理由也不需要继承UINavigationController类。可以直接访问导航栏及其特性。下面给出了loadView方法中的一行代码，用于使用一系列属性定制导航栏样式：

```
self.navigationController.navigationBar.barStyle = UIBarStyleBlackTranslucent;
```


模式表示

模式表示提供了另一种呈现视图控制器的方法。发送`presentModalViewController:animated:`消息之后，一个新的视图控制器将滑入屏幕并接过控制权，直到被`dismissModalViewControllerAnimated:`取消。这使你能在应用程序中添加特殊用途的对话框。通常，模式控制器用于从Address Book中提取联系人等数据，或从Library中提取相片。第7章和第9章展示了实际的模式视图控制器。

3.3.3 导航项类

填充到导航栏中的对象将通过`UINavigationController`类放置到合适的位置。该类允许将按钮、文本和其他UI对象附加到3个关键位置：导航栏的左侧、中间和右侧。通常，普通按钮位于右侧，一些文本（通常为`UIViewController`的标题）位于中间，而左侧是一个Back样式的按钮。但是，并非仅限于此布局。可以在这3个位置添加自定义控件。因此，可以构建带搜索字段、分段控件、工具栏和图像的导航栏。

通过`SetCustomLeftItem:animated:`、`SetCustomRightItem:animated:`并更新`customTitleView`属性指向所选的视图来分配这些项目。代码清单3-4展示了如何在右导航项目视图中添加自定义按钮。这是标准的导航样式`UIButton (UIButtonTypeNavigation)`。在本例中，在推入时会调用`randomize`方法。

代码清单3-4 在`UINavigationController`中添加自定义按钮

```
// Add a randomize button
UIBarButtonItem *randomButton = [[[UIBarButtonItem alloc]
    initWithTitle:@"Randomize"
    style:UIBarButtonItemStylePlain
    target:self
    action:@selector(randomize)] autorelease];
self.navigationItem.rightBarButtonItem = randomButton;
```

在标题空间添加自定义的视图非常简单。无需添加控件，而是分配一个视图。本示例（来自第5章的第一个秘诀）将添加一个自定义的`UILabel`：

```
self.navigationItem.titleView = [[UILabel alloc] initWithFrame:CGRectMake(0.0f,
    4.0f, 320.0f, 36.0f)];
```

3.4 秘诀：构建简单的双项菜单

虽然许多应用程序都要求具备完整的用户界面，但通常不必过于复杂。具有一个或两个按钮的简单菜单能够完成许多任务。使用以下步骤为简单的实用工具创建基本界面。

- (1) 创建一个`UIViewController`子类，作为主要交互空间。
- (2) 分配一个导航控制器，并为其根视图分配一个自定义视图控制器实例。
- (3) 在自定义视图控制器中，创建一个或两个按钮并将它们添加到视图的导航项中。
- (4) 构建回调例程，用于在用户轻击按钮时触发。

秘诀3-1演示了这些步骤。它将创建一个简单的视图控制器，名为HelloController，并将其指定为UINavigationController的根视图。在loadView方法中，两个按钮将填充视图导航项的左侧及右侧自定义槽。当轻击时，它们将更新主视图的背景颜色。此秘诀并未提供丰富的特性，但它提供了一个易于构建的由两个项组成的菜单。图3-10显示了实际的界面。

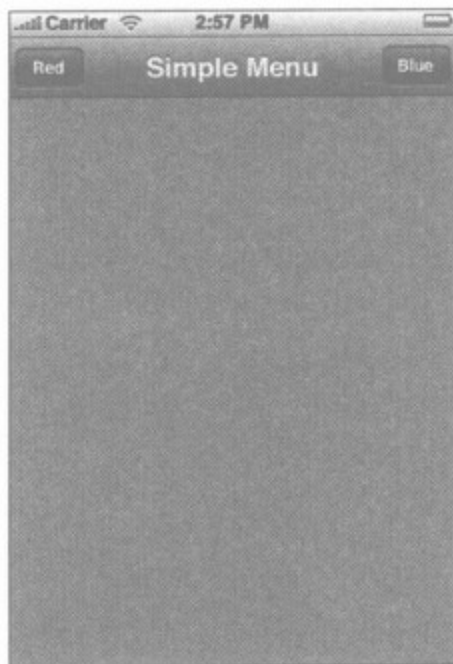


图3-10 通过在UINavigationController中添加一个自定义按钮，为简单的iPhone应用程序创建一个由两个按钮组成的基本菜单

如果你需要的导航项多于两项，可以考虑让按钮触发UIActionSheet菜单。操作表（将在第4章中讨论）支持从一个简短的选项列表（通常为2~5个选项）中选择操作，它们在Photos和Mail应用程序中可用于共享和存储数据。

说明 除了文本之外，还可以将图像添加到在导航栏中使用的UIButton中。

秘诀3-1 使用导航控制器创建一个包含两个项的菜单

```
- (void) goRed
{
    contentView.backgroundColor = [UIColor colorWithRed: 1.0f green:0.45f
➤blue:0.45f alpha:1.0f];
}

- (void) goBlue
{
    contentView.backgroundColor = [UIColor colorWithRed: 0.45f green:0.45f
➤blue:1.0f alpha:1.0f];
}

- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:[UIScreen mainScreen]
```

```
➡applicationFrame]]];
self.view = contentView;
contentView.backgroundColor = [UIColor whiteColor];
[contentView release];

// Add a left button
self.navigationItem.leftBarButtonItem = [[[UIBarButtonItem alloc]
initWithTitle:@"Red"
style:UIBarButtonItemStylePlain
target:self
action:@selector(goRed)]
autorelease];

// Add a right button
self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
initWithTitle:@"Blue"
style:UIBarButtonItemStylePlain
target:self
action:@selector(goBlue)]
autorelease];

}
```

3.5 秘诀：添加分段控件

之前的秘诀展示了如何使用导航栏中的两个可用按钮槽来构建微型菜单。秘诀3-2扩展了此想法，它引入一个包含4个项的UISegmentedControl，并将它添加到导航栏的自定义标题视图中，如图3-11所示。在轻击时，每个项都将使用其数值更新主视图。

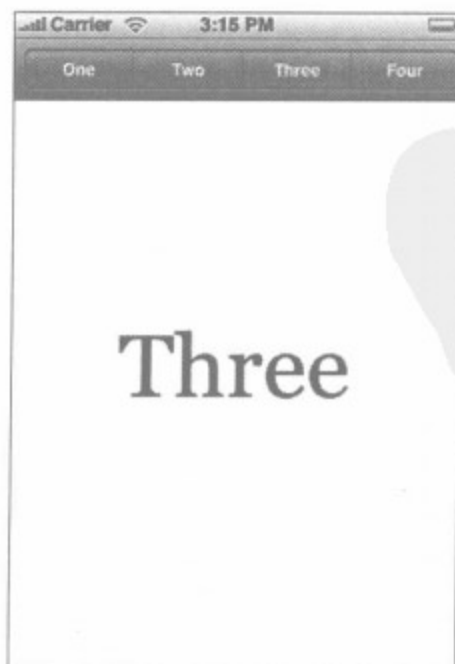


图3-11 通过在自定义标题视图中添加分段控件，可以构建一个多项菜单。注意，即便在某操作发生之后，也没有项目继续处于突出显示状态（在本例中，按下的是第3个按钮）

此秘诀中需要注意的一个地方是分配给分段控件的momentary属性。它将界面从单选按钮样式转换为实际的菜单，该菜单中的项可被独立地进行多次选择。因此，例如，当轻击第3项之后，还可以再次轻击。这对于菜单交互来说是一个重要的行为。

与秘诀3-1不同，分段控件中的所有项都将触发相同的操作（在本例中为segmentAction:）。通过查询控件的selectedSegmentIndex并匹配该值与合适的行为，确定要采用哪种操作。

秘诀3-2 在导航栏中添加分段控件

```
- (void)loadView
{
    // Set up the content view
    contentView = [[UITextView alloc] initWithFrame:[UIScreen mainScreen]
    ➤applicationFrame]];
    contentView.autoresizesSubviews = YES;
    contentView.textAlignment = NSTextAlignmentCenter;
    contentView.font = [UIFont fontWithName:@"Georgia" size:64.0f];
    contentView.autoresizingMask = (UIViewAutoresizingFlexibleWidth |
    ➤UIViewAutoresizingFlexibleHeight);
    [contentView setEditable:NO];
    self.view = contentView;
    [contentView release];
    // Build the Segmented Control
    NSArray *buttonNames = [NSArray arrayWithObjects:@"One", @"Two", @"Three",
    ➤@"Four", nil];
    UISegmentedControl* segmentedControl = [[UISegmentedControl alloc]
    ➤initWithItems:buttonNames];
    segmentedControl.momentary = YES;
    [contentView setText:@""];

    // Customize the Segmented Control
    segmentedControl.autoresizingMask = UIViewAutoresizingFlexibleWidth;
    segmentedControl.segmentedControlStyle = UISegmentedControlStyleBar;
    segmentedControl.frame = CGRectMake(0, 0, 400, 30);
    [segmentedControl addTarget:self action:@selector(segmentAction:)
    ➤forControlEvents:UIControlEventValueChanged];

    // Add the control to the navigation bar
    self.navigationItem.titleView = segmentedControl;
    [segmentedControl release];
}

- (void) segmentAction: (id) sender
{
    switch([sender selectedSegmentIndex] + 1)
    {
        case 1: [contentView setText:@"\n\nOne"]; break;
        case 2: [contentView setText:@"\n\nTwo"]; break;
        case 3: [contentView setText:@"\n\nThree"]; break;
        case 4: [contentView setText:@"\n\nFour"]; break;
    }
}
```



```

        default: break;
    }
}

```

3.6 秘诀：在导航栏中添加 UIToolbar

下一个秘诀构建于秘诀3-2的基础之上。秘诀3-2在导航栏中使用了一个分段控件，并执行了进一步操作。秘诀3-3将在导航栏项的自定义标题视图中添加一个UIToolbar，并将它添加到秘诀3-1的基本界面中。这将完成两件事情：

- 它添加了强大的工具栏，同时未占用屏幕底部的空间；
- 它不需要干预导航栏的两个核心按钮槽。

在使用此方法时，集群是应该考虑的一个问题。图3-12显示了更新后的界面，包括菜单按钮和工具栏，同时维护了干净简洁的外观。工具栏使用箭头左侧及右侧的灵活空间，以居中控件并维持简单的外观。还应注意横向外观。工具栏高度（可完全适应纵向导航栏）可能无法匹配较新SDK版本中的横向栏。

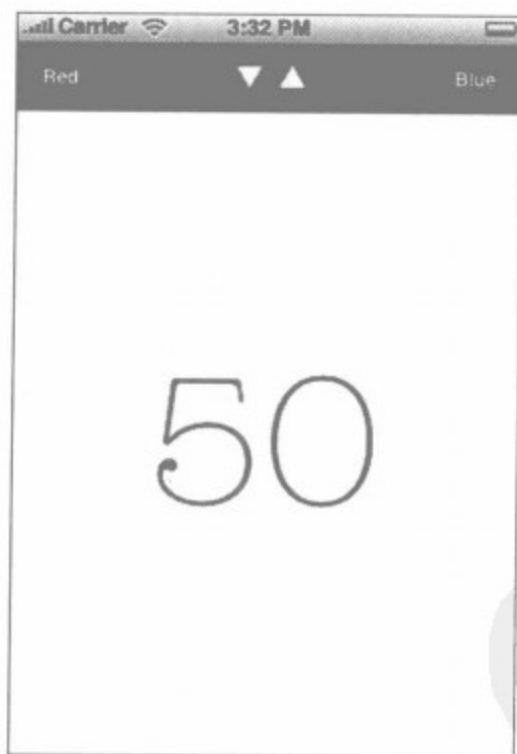


图3-12 设计良好的自定义工具栏利用了导航栏按钮之间的空间，同时释放了屏幕底部的空间

说明 两个箭头使用自定义图标up.png和down.png，它们将在运行时从应用程序束中加载并被添加到工具栏按钮中。它们由Photoshop创建，是基本的PNG图像，在干净的背景上采用白色显示并使用透明度进行保存。

秘诀3-3 在自定义标题视图中添加工具栏

```

- (void)loadView
{

```

```
// Set up the text view to show the current value
contentView = [[UITextView alloc] initWithFrame:[UIScreen mainScreen]
applicationFrame]];
contentView.autoresizesSubviews = YES;
contentView.autoresizingMask = (UIViewAutoresizingFlexibleWidth |
UIViewAutoresizingFlexibleHeight);
contentView.editable = NO;
contentView.textAlignment = UITextAlignmentCenter;
contentView.font = [UIFont fontWithName:@"American Typewriter" size:120];
self.view = contentView;
[contentView release];

// Initialize at 50
[contentView setText:@"\n50"];
value = 50;

// The app style is black
self.navigationController.navigationBar.barStyle = UIBarStyleBlackTranslucent;

NSMutableArray *buttons = [[NSMutableArray alloc] init];

UIBarButtonItem *flexibleSpaceItem;
flexibleSpaceItem = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:
UIBarButtonSystemItemFlexibleSpace
target:nil action:NULL];
[buttons addObject:flexibleSpaceItem];
[flexibleSpaceItem release];

// Add the up and down items
UIBarButtonItem *item;
item = [[UIBarButtonItem alloc]
initWithImage:[UIImage imageNamed:@"down.png"]
style:UIBarButtonItemStylePlain
target:self
action:@selector(decrement:)];
[buttons addObject:item];
[item release];

item = [[UIBarButtonItem alloc]
initWithImage:[UIImage imageNamed:@"up.png"]
style:UIBarButtonItemStylePlain target:self
action:@selector(increment:)];
[buttons addObject:item];
[item release];

flexibleSpaceItem = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:
UIBarButtonSystemItemFlexibleSpace
```

```

        target:nil action:NULL];
[buttons addObject:flexibleSpaceItem];
[flexibleSpaceItem release];

UIToolbar *toolbar = [[UIToolbar alloc] init];
toolbar.barStyle = UIBarStyleBlackOpaque;
[toolbar setItems:buttons animated:YES];
[toolbar sizeToFit];
self.navigationItem.titleView = toolbar;
[toolbar release];

// Add a left button
self.navigationItem.leftBarButtonItem = [[UIBarButtonItem alloc]
                                         initWithTitle:@"Red"
                                         style:UIBarButtonItemStylePlain
                                         target:self
                                         action:@selector(goRed)]
                                         autorelease];

// Add a right button
self.navigationItem.rightBarButtonItem = [[UIBarButtonItem alloc]
                                         initWithTitle:@"Blue"
                                         style:UIBarButtonItemStylePlain
                                         target:self
                                         action:@selector(goBlue)]
                                         autorelease];
}

-(void) increment: (id) sender
{
    [contentView setText:[NSString stringWithFormat:@"%d", ++value]];
}

-(void) decrement: (id) sender
{
    [contentView setText:[NSString stringWithFormat:@"%d", --value]];
}

```

3.7 秘诀：在视图控制器之间导航

除了提供漂亮的菜单之外，导航控制器还有其本职工作：当用户在视图间导航时管理层次结构。秘诀3-4将导航控制器作为实际的导航控制器引入，从而将视图推入栈中。这些视图由3个 `UIViewController` 子类构成，每个子类都有可以识别的颜色：石灰色、粉红色和橙色。在实际使用中，你会使用更加有意义的视图控制器。本示例从事物最简单的层面进行演示。在本秘诀中，需要记住以下一些要点。

- 导航控制器将自动创建如图3-13所示的Back按钮（Lime，石灰色），其效果是从（根）Lime

控制器推入新的（Orange，橙色）控制器。最右侧的按钮（Pink，粉红色）通过调用 `pushViewController:animated:` 触发导航到下一个控制器。在推入时，下一个Back按钮将显示为Orange。

- 界面可同时适应在横向和纵向方向。在测试时，你可能希望尝试这两种风格。在仿真器中，依次选择Hardware、Rotate Left（Command-左箭头）或Hardware、Rotate Right（Command-右箭头）查看实际效果。

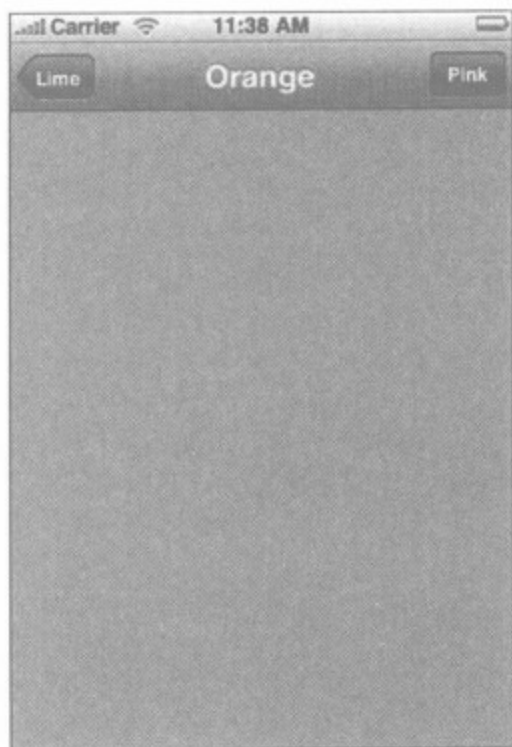


图3-13 导航控制器将自动创建Back和Forward按钮

秘诀3-4 通过UINavigationController向下展开视图

```
@implementation OrangeController
- (id) init
{
    // The view controller title sets the name for back buttons
    // once it's been pushed
    if (self = [super init]) self.title = @"Orange";
    return self;
}

- (void) loadButton
{
    // Create the button that activates the switch
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
        initWithTitle:@"Pink"
        style:UIBarButtonItemStylePlain
        target:self
        action:@selector(switch:)] autorelease];
}
```



```
- (void) loadView
{
    [super loadView];
    contentView.backgroundColor = ORANGE;
}

- (void) switch: (id) sender
{
    // On a button press, push the next controller into place
    [[self navigationController] pushViewController:[[PinkController alloc] init]
    animated:YES];
}
@end
```

3.7.1 返回根

虽然你通常希望通过点击Back按钮来返回之前的视图控制器，但应该明白有时需要弹出整个栈。举例来说，可能你有一个交互式测验，或者某个博物馆参观者完成了他的行程。对于这两种情况，一次返回一屏幕冗长复杂的树不太合适。而应该使用`popToRootViewControllerAnimated:`直接返回根，或使用`popToViewController:animated:`停在靠近根的位置。它们将弹出视图控制器，直到根控制器或指定的控制器成为顶部控制器，并相应地更新显示。

3.7.2 加载视图控制器数组

还可以创建一个由UINavigationController对象组成的NSArray数组，并将其分配给UINavigationController的viewControllers属性。顶部（即活动）的viewController将占用该数组中的最后一个位置（ $n-1$ ），而根对象的下标仍然为0。在概念树中跳转时，数组是非常便捷的。例如，你可能正在目录间导航，然后需要通过某符号链接跳转到别处。通过设置整个数组，可以避免弹出和推入栈的细节工作。

视图控制器数组还可以帮助在退出后恢复之前的状态，然后返回到某个应用程序。将状态列表存储到本地默认设置，然后在启动时重新创建相同的数组。将重新创建的数组分配给viewControllers，并让用户返回到其上次退出时的那个位置。

3.8 选项卡栏

通过UITabBarController类，用户可以在多个视图控制器之间移动，并定制屏幕底部的栏。这在YouTube和iPod应用程序中较常见。两者都支持通过单击操作访问不同的视图，且都提供了一个More按钮，用户可通过该按钮选择和编辑底部栏。

借助选项卡栏，不必像操作导航栏那样推入视图。而是组建一系列控制器（它们各自可以是UIViewController、UINavigationController或任何其他种类的视图控制器），并将它们添加到选项卡栏中。仅需设置栏的viewControllers属性，实际上就是这么简单。Cocoa Touch将

为你完成所有其余工作。将allowsCustomizing设置为YES，以支持用户对栏重新排序。

秘诀3-5将创建11个简单的BrightnessController类的视图控制器。该类将构建一个UIView，并将其背景设置为指定的灰度，在本例中为0%~100%，以10%为最小刻度。这11个视图控制器存储在一个数组中，它们充当可供用户导航和选择的选项，如图3-14所示。



图3-14 通过选项卡栏控制器，用户可以从屏幕底部的栏中选择视图控制器（图的左侧），并通过一个可用视图控制器的列表对栏进行定制（图的右侧）

注意，此秘诀将这11个控制器添加了两次。第一次是将它们分配给对于用户可用的视图控制器列表：

```
tbarController.viewControllers = controllers;
```

第二次是指定用户在以交互式的方式定制底部选项卡栏时，可以从整个列表中进行选择：

```
tbarController.customizableViewControllers = controllers;
```

第二行代码是可选的，而第一行代码是必不可少的。设置视图控制器之后，可以将所有或部分视图控制器添加到可自定义的列表中。如果没有这样做，你也能够使用More按钮查看其他视图控制器，但用户不能随需将它们包括在主选项卡中。

注意，图标将以相反的颜色显示在More屏幕上。根据苹果公司的说法，这是预期的也是正确的行为。苹果公司并未计划修改它。当纯白色的手表以纯黑色出现在该屏幕上时，将出现有趣的视图反差效果。

秘诀3-5 创建选项卡视图控制器

```
@interface BrightnessController : UIViewController
{
    UIView *contentView;
    int brightness;
}
@end
```

```

@implementation BrightnessController

-(BrightnessController *) initWithBrightness: (int) aBrightness
{
    self = [super init];
    brightness = aBrightness;
    self.title = [NSString stringWithFormat:@"%d%%", brightness * 10];
    [self.tabBarItem initWithTitle:self.title image:createImage(((float)
    ➤ brightness / 10.0f)) tag:NULL];
    return self;
}

-(void)loadView
{
    // Create each view with the given brightness
    contentView = [[UIView alloc] init];
    float percent = brightness * 0.1;
    contentView.backgroundColor = [UIColor colorWithRed:percent green:percent
    ➤ blue:percent alpha:1.0];
    contentView.autoresizesSubviews = YES;
    contentView.autoresizingMask = (UIViewAutoresizingFlexibleWidth |
    ➤ UIViewAutoresizingFlexibleHeight);
    self.view = contentView;
    [contentView release];
}

-(void) dealloc
{
    [contentView release];
    [super dealloc];
}

@end

@interface SampleAppDelegate : NSObject <UIApplicationDelegate,
➤ UITabBarControllerDelegate> {
}

@end

@implementation SampleAppDelegate

-(void)applicationDidFinishLaunching:(UIApplication *)application {

    UIWindow *window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen]
    ➤ bounds];

    // Create the array of UIViewControllers
    NSMutableArray *controllers = [[NSMutableArray alloc] init];

    for (int i = 0; i < 11; i++) {

```

```

    BrightnessController *bControl = [[BrightnessController alloc]
                                       initWithBrightness:i];
    UINavigationController *nav = [[UINavigationController alloc]
                                   initWithRootViewController:bControl];
    nav.navigationBar.barStyle = UIBarStyleBlackTranslucent;
    [controllers addObject:nav];
    [bControl release];
    [nav release];
}

UITabBarController *tbarController = [[UITabBarController alloc] init];
tbarController.viewControllers = controllers;
tbarController.customizableViewControllers = controllers;
tbarController.delegate = self;
tbarController.allowsCustomizing = YES;
[window addSubview:tbarController.view];
[window makeKeyAndVisible];
[controllers release];
}

- (void)dealloc {
    [super dealloc];
}

@end

```

3.9 小结

本章介绍了UIViewController、UINavigationController和UITabBarController类，并展示了如何使用它们。这些类处理视图管理和用户导航。开发人员可以在它们之间扩展虚拟交互空间，并根据其应用程序的需要创建多个界面。

通过详细步骤和秘诀，你学习了如何将视图控制器连接到在Interface Builder中设计的界面，以及如何完全在Xcode中创建界面。在继续阅读下一章之前，需要掌握以下关于视图控制器的要点。

- ❑ Interface Builder在布局UIView实例的内容方面极为出色。使用其工具将这些实例连接到程序中的视图控制器，并使用Interface Builder改进可见即所得样式的界面，例如本章中的温度转换器示例。
- ❑ 了解Interface Builder何时不是正确的解决方案。当通过最少的窗口设计构建选项卡栏和导航控制器时（例如在基于表或基于文本的应用程序中），并非特别需要IB的视图布局工具。如果略过了IB，应确保从项目中删除XIB文件并从Info.plist中删除主NIB窗口。
- ❑ 使用导航树构建分层界面。它们非常适合查看文件结构或构建设置树。当打算制作“信息公开视图”或“首选项”时，可以考虑将一个新的控制器推入到导航栈中。
- ❑ 不要害怕以非传统的方式使用传统UI元素。本章中涵盖的大部分内容都以创新的方式使用了UINavigationController，未涉及任何导航任务。工具是死的，但人应是活的。

有时，你需要引起用户的注意。例如接收到新消息了，或者系统状态发生了更改。你可能希望提醒用户在执行某些操作之前等待一段时间，或者在等待结束之后引起用户的注意。iPhone提供了许多提醒用户的方法，包括警告和进度条，以及语音提示和状态栏更新。在本章中，你将探索如何在应用程序中构建这些指示并扩展用户警告词汇。你将看到演示这些类的真实示例，并了解如何确保在适当的时间引起用户注意。

4.1 通过警告直接与用户对话

可以通过UIActionSheet和UIAlertView对象与用户对话。它们通过弹出或在其他视图上方滚动来发送消息。这些轻量级类向应用程序中添加双向对话。警告直观地与用户“对话”并且可以提示用户回答。应用程序在屏幕上显示警告，获取用户确认，然后关闭警告继续进行其他任务。

如果你认为警告只不过是附带OK按钮的消息（如图4-1所示），那么这样的结论值得反思。UIAlertSheet对象提供了丰富的功能（假设苹果公司继续让你访问这些丰富的功能）。借助警告表，你可以实际构建菜单、文本输入、查询以及更多内容。遗憾的是，此行为多半都被归类为文档中未记录或几乎未记录的类别。在本章的秘诀中，你将了解如何创建可以在自己程序中使用的各种实用的警告。

本章中涉及的大部分功能已从正式的SDK中删除，但在公共框架中仍然存在。由于本章更多地依赖于“非正式”调用，因此应该找到可靠性和功能之间的最佳平衡点。苹果公司不鼓励开发人员使用私有例程，因为开发人员会随意更改这些例程。但其本质与Mac OS X相同。如果你知道如何在公共架构中使用私有例程的方法，则可以使用它们。在应用程序中访问私有框架则是错误的做法。

说明 在早期版本的iPhone固件中，UIActionSheet和UIAlertView实际上由相同的UIAlertSheet类实现。该类不仅提供弹出警告，而且还提供菜单功能。然后苹果公司将警告表替换为UIModalView，并对来自该基类的这些新对象进行子类化。之后，苹果公司删除了UIModalView，在新版本的SDK中，UIActionSheet和UIAlertView不再由此类派生。（它们都来自于UIView。）与其前身一样，它们在行为方面仍然属于同类，并且使用相似的底层技术在屏幕上呈现自己。

4.1.1 记录结果

由于printf非常简单，因此大部分秘诀都使用printf以可以查看的格式输出它们的结果。在最初的SDK Beta版本期间，printf提供了检查内部程序数据的最可靠的方法。现在，在最新的iPhone SDK版本中，NSLog()得到了完全实现，并且非常稳定。此外，下面给出了几个实用的技巧。

1. 重定向stderr

标准的freopen()函数将stderr输出重定向到你选择的日志文件，为其指定一个本地路径。此处，logPath是一个标准的NSString(如@"tmp/mylog")，它已被转换为本地文件系统表示。完成此调用之后，所有stderr结果(包括NSLog)将直接进入你的日志文件。当你的程序运行时，使用"w"(写)而不是使用"a"(附加)来重新启动日志：

```
freopen([logPath fileSystemRepresentation], "a", stderr);
```

2. 构建自定义日志函数

构建你自己的日志函数，可以添加除标准NSLog之外的功能。下面是模拟NSLog功能的一个基本日志函数。通过此函数，可以方便地生成警告通知或文本视图更新，而不必输出到stderr：

```
#include <stdarg.h>
void dolog(id formatstring,...)
{
    va_list arglist;
    if (formatstring)
    {
        va_start(arglist, formatstring);
        id outstring = [[NSString alloc] initWithFormat:formatstring
        arguments:arglist];
        fprintf(stderr, "%s\n", [outstring UTF8String]);
        [outstring release];
        va_end(arglist);
    }
}
```

4.1.2 构建警告

若要创建警告表，请分配一个UIAlertView对象。通过初始化为其分配一个标题、一个按钮数组和若干其他选项。该标题是一个NSString，按钮数组包含多个NSString，其中每个字符串表示一个按钮。代码清单4-1创建了一个非常简单的警告，如图4-1所示。

使用警告时，通常空间非常珍贵。将所有按钮放在一行上可以最大程度地减小警告高度，并且这也是默认表示。当定义更多按钮时，它们会一个挨一个地添加到该行中。因此，任何时候都要将添加的按钮数量限制为不超过3个或4个。按钮越少，效果越好，最好只有一两个。

UIAlertView对象没有提供可视的“默认”按钮突出显示。在iPhone上下文中，按钮突出显示不但不能提供帮助，而且还容易造成混淆。(例如，用户应该按浅色按钮还是深色按钮?)没有办法使界面符合用户期望，因为此表示不明确。因此，与OS X不同的是，苹果公司在较新的iPhone固件中删除了突出显示。

说明 Cancel按钮出现在菜单的底部，并且位于弹出警告的左侧。

代码清单4-1 创建基本警告

```
UIAlertView *baseAlert = [[UIAlertView alloc]
                           initWithTitle:@"Alert" message:@" "
                           delegate:self cancelButtonTitle:nil
                           otherButtonTitles:@"OK", nil];
```

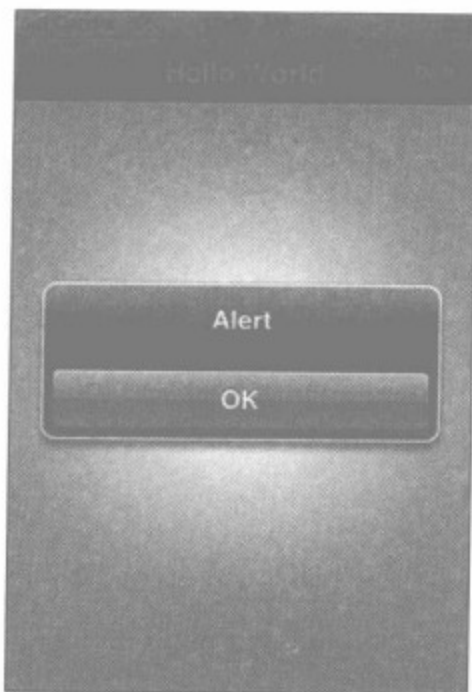


图4-1 使用UIAlertView类构建简单的弹出警告

除非万不得已，否则应将委托设置为UIViewController主对象。委托实现了UIAlertView-Delegate协议。UIAlertView实例至少需要此委托支持才能处理按钮轻击。

通过委托方法，你可以自定义当按下不同的按钮时的响应。实际上，如果仅需要通过一个OK按钮抛出一些消息，则可以忽略此委托支持。

用户看到警告并与之交互之后，他们会发起以下委托方法调用：alertView:clickedButtonAtIndex:。通过此调用，你可以确定按下了哪个按钮并且恢复在此表上设置的任何选项。按钮编号从零开始。

4.1.3 显示警告

使用show方法通知警告显示在屏幕上（即[myAlert show]）。显示时，警告采用模态方式工作。即，它使其后的屏幕暗淡并阻止用户与应用程序交互。此模式交互将一直继续，直到用户按钮轻击（通常是选择OK或Cancel）确认该警告为止。

说明 创建警告表之后，你可以通过更新其message来自定义警告。该可选文本位于警告标题下方，警告按钮下方。其他可自定义的属性包括title、delegate和visible。

4.2 秘诀：创建多行按钮显示

默认情况下，iPhone在一行上显示其按钮。图4-2在左侧显示了这个标准的外观。你可以使用文档中未记录的调用覆盖一行显示。此调用创建一个多行警告，如右侧所示。使用 `setNumberOfRows:` 构建单行按钮警告，如代码清单4-2所示。其中提供一个参数和一个整数，用于指定所请求的行数。

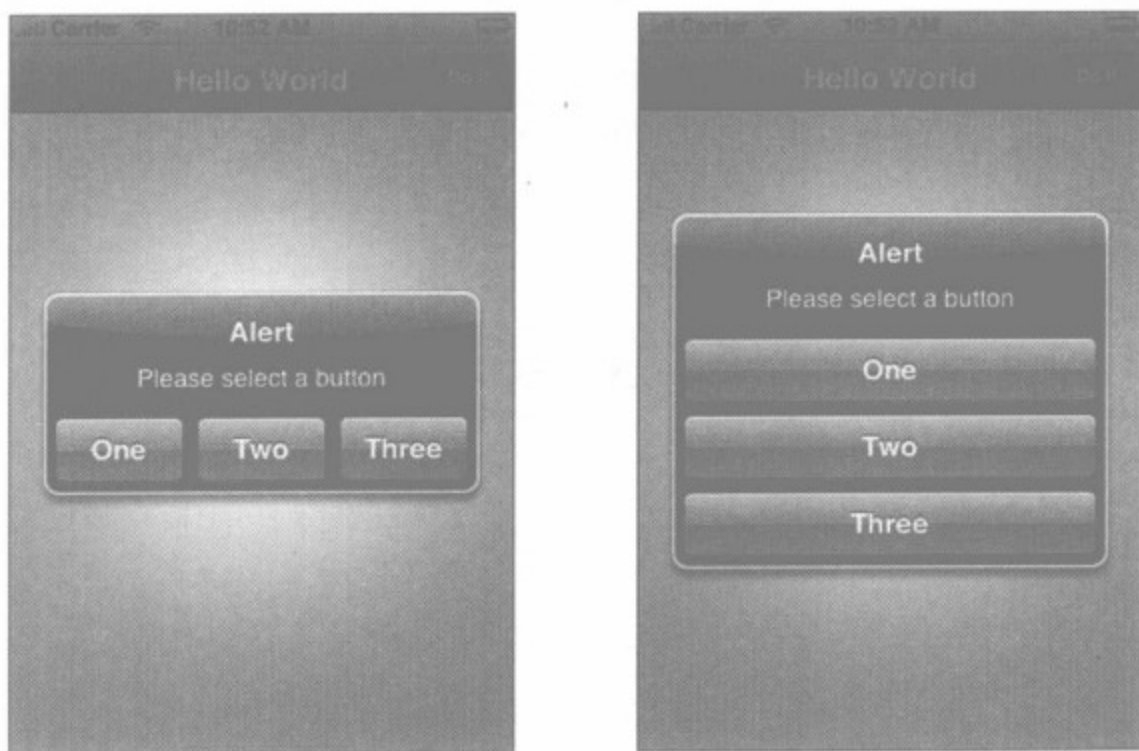


图4-2 通过文档中未记录的UIKit调用，你可以选择是将按钮放置在一行还是一列

说明 有关在程序中使用文档中未记录的调用和特性的详细讨论，请参见第1章。

早期版本iPhone软件的工作方式恰好相反。最初的标准采用多行模式，并且必须使用 `setNumberOfRows:` 让电话的按钮移动到单行上。

秘诀4-1创建的示例在屏幕中央弹出其警告，其周围的屏幕将变暗。由于默认情况下它采用模态方式运行，因此在用户轻击OK按钮之前，任何事情都处于等待状态。轻击之后，警告关闭，然后将控制传递给 `modalView: clickedButtonAtIndex:` 委托方法。该方法正如此处所定义的，除了打印轻击按钮的编号之外，什么也没有做。

说明 你可以使用 `setDimsBackground:NO` 禁止背景变暗淡。这是另一个文档中未记录的调用。我建议不要将它用作常规用途。它违背了苹果公司的人机界面标准，并且可能使用户弄不清楚是否可以在警告生存期内轻击背景。

秘诀4-1 创建带单行按钮的UIAlertView

```
- (void) presentSheet
{
```



```

UIAlertView *baseAlert = [[UIAlertView alloc]
                           initWithTitle:@"Alert"
                           message:@"Please select a button"
                           delegate:self cancelButtonTitle:nil
                           otherButtonTitles:@"One", @"Two", @"Three", nil];
[baseAlert setNumberOfRows:3];
[baseAlert show];
}
- (void)alertView:(UIAlertView *)alertView
  clickedButtonAtIndex:(NSInteger)buttonIndex
{
    printf("User Pressed Button %d\n", buttonIndex + 1);
    [alertView release];
}

```

4.3 秘诀：自动计时的无按钮警告

4

无按钮警告提出了一个特殊的挑战，原因是它们无法正确地回调委托方法。它们无法自动关闭，即使在轻击时也是如此。图4-3显示了一个简单的无按钮警告。从此屏幕快照中可以看出，删除按钮会使警告看起来比较笨拙。你可以通过调整消息文本来改进此显示。添加回车(@"\n")，通过顶部的空间来平衡底部（正常情况下是按钮所在的位置）。



图4-3 使用文档中未记录的dismiss方法在显示一个无按钮的UIAlertView之后，将控制返回你的程序。在message属性中添加回车有助于平衡底部因缺少按钮而多出来的空间

由于警告以模态方式运行，因此你需要通过一项安全措施来确保警告在特定时间点消失，并且用户可以继续使用他们的iPhone。一个简单的NSTimer可在一段时间（必须将控制返回到标准GUI的时间段）之后关闭警告。秘诀4-2创建了一个警告并使用计时器将其关闭，要么使用文档中未记录的dismiss方法，要么使用UIAlertView类的dismissWithClickedButtonIndex:animated:方法。

秘诀4-2 创建带计时器自动防故障装置的无按钮UIAlertView

```

- (void) performDismiss: (NSTimer *)timer
{
    [baseAlert dismissWithClickedButtonIndex:0 animated:NO];
}

- (void) presentSheet
{
    baseAlert = [[UIAlertView alloc]
                  initWithTitle:@"Alert"
                  message:@"Message to user with asynchronous
                           information"
                  delegate:self cancelButtonTitle:nil
                  otherButtonTitles: nil];
    [NSTimer scheduledTimerWithTimeInterval:3.0f target:self selector:
    ➡@selector(performDismiss:)
                           userInfo:nil repeats:NO];

    [baseAlert show];
}

```

4.4 秘诀：请求用户的文本输入

暂且将正式SDK调用放在一边，警告视图还提供了一个非常简单的方法，用于提示用户输入文本。UIAlertView类提供显示和关闭关联键盘的全部命令。你只需要添加文本输入字段，告诉警告如何处理它们，以及让它注意其他事项。

若要添加文本字段，请使用文档中未记录的addTextFieldWithValue:label:调用。将默认文本作为第一个参数，将otherwiseempty字段中显示的文本作为第二个参数传递给该调用。图4-4显示了已填充字段和空字段。



图4-4 细致地管理空间并忽略标题和正文文本，你一次最多可向UIAlertView中添加3个文本输入字段。你可能希望将AlertViews限制为1个或2个文本字段

你可以通过调用[alert textFieldAtIndex:0]，然后从带text属性的文本字段获取文本来恢复每个字段。由于每个字段都是可寻址的，因此你可以在显示警告之前设置其属性。自定义自动标题、自动纠正以及首选的键盘，如秘诀4-3所示。

空间非常重要。如果要细致地管理空间，你可以将警告放在屏幕上，同时留出足够的空间用于键盘显示。通过跳过标题和正文文本，你可以一次在屏幕上放置3个（而不是4个）文本输入字段。两个字段是符合实际的最大值，如图4-4所示。

秘诀4-3 使用UIAlertView请求用户的文本

```
- (void)alertView:(UIAlertView *)alertView
clickedButtonAtIndex:(NSInteger)buttonIndex
{
    printf("User Pressed Button %d\n", buttonIndex + 1);
    printf("Text Field 1: %s\n", [[[alertView textFieldAtIndex:0] text]
    ↪cStringUsingEncoding:NSUTF8StringEncoding]);
    printf("Text Field 2: %s\n", [[[alertView textFieldAtIndex:1] text]
    ↪cStringUsingEncoding:1]);
    [alertView release];
}

- (void) presentSheet
{
    UIAlertView *alert = [[UIAlertView alloc]
        initWithTitle: @"Enter Information"
        message:@"Specify the Name and URL"
        delegate:self
        cancelButtonTitle:@"Cancel"
        otherButtonTitles:@"OK", nil];
    [alert addTextFieldWithValue:@" " label:@"Enter Name"];
    [alert addTextFieldWithValue:@"http://" label:@"Enter URL"];

    // Name field
    UITextField *tf = [alert textFieldAtIndex:0];
    tf.clearButtonMode = UITextFieldViewModeWhileEditing;
    tf.keyboardType = UIKeyboardTypeAlphabet;
    tf.keyboardAppearance = UIKeyboardAppearanceAlert;
    tf.autocapitalizationType = UITextAutocapitalizationTypeWords;
    tf.autocorrectionType = UITextAutocorrectionTypeNo;

    // URL field
    tf = [alert textFieldAtIndex:1];
    tf.clearButtonMode = UITextFieldViewModeWhileEditing;
    tf.keyboardType = UIKeyboardTypeURL;
    tf.keyboardAppearance = UIKeyboardAppearanceAlert;
    tf.autocapitalizationType = UITextAutocapitalizationTypeNone;
    tf.autocorrectionType = UITextAutocorrectionTypeNo;

    [alert show];
}
```

4.5 秘诀：显示简单菜单

谈到菜单，UIActionSheet实例可满足iPhone的需要。它们将选择对象（主要是表示可能操作的按钮列表）滑动到屏幕上并且等待用户响应。操作页不同于弹出菜单。弹出菜单与界面相分离，并且比较适合用于请求注意。菜单滑入视图中，并且可以更好地与当前应用程序工作集成。Cocoa Touch提供了两种显示菜单的方法。

- ❑ `showInView`。在视图中显示表是使用菜单的理想方式，此处使用的就是这个方法。该方法将视图底部的菜单向上滑动（参见图4-5）。
- ❑ `showFromToolBar`和`showFromTabBar`。在使用工具栏、选项卡栏或提供水平分组按钮的任何其他栏（在很多应用程序的底部你可以看到这些栏）时，这些方法将该菜单与栏顶部对齐并且完全滑出它原来所在的位置。

秘诀4-4显示了初始化和显示简单的UIActionSheet实例的方法。它的初始化方法引入了一个UIAlertView：破坏性按钮中缺少的概念。红色的破坏性按钮表示没有返回的操作，如永久删除某个文件（参见图4-5）。明亮的红色向用户发出关于这个选择的警告。很明显，该选项应该谨慎使用。

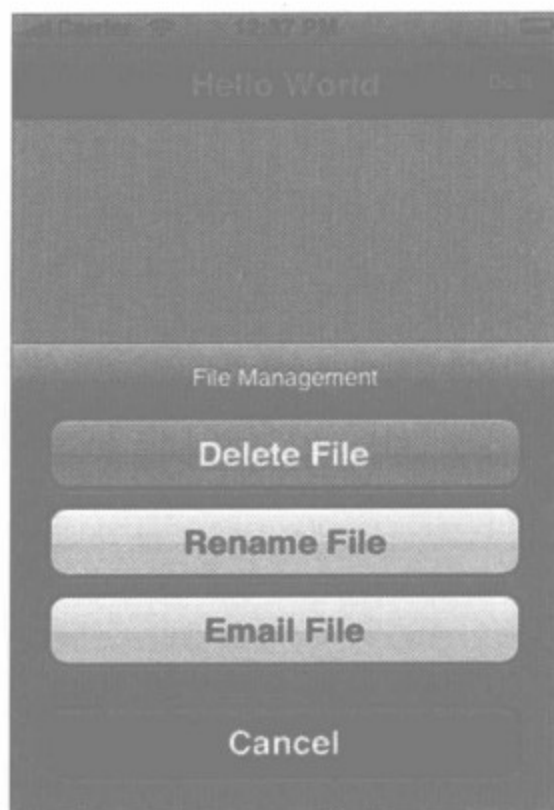


图4-5 使用`showInView`：创建简单的菜单演示。菜单滑动到视图的底部。尽管此处Delete File菜单按钮显示为灰色，但它在iPhone上显示为红色，表示永久的操作，这些操作可能对用户造成不利影响

说明 操作表缺少的就是UIAlertView的消息（至少对于正式SDK是如此）。除了显示标题之外，操作表实例未正式支持添加消息。实际上，消息虽然得到了实现，但效果却不是非常理想。使用文档中未记录的`setMessage:`调用向菜单中添加消息。使用时，消息显示在表标题下方，文本稍微有点大。

秘诀4-4 显示菜单

```
-(void)actionSheet:(UIActionSheet *)actionSheet
  clickedButtonAtIndex:(NSInteger)buttonIndex
{
    printf("User Pressed Button %d\n", buttonIndex + 1);
    [actionSheet release];
}

- (void) presentSheet
{
    UIActionSheet *menu = [[UIActionSheet alloc]
```



```
initWithTitle: @"File Management"
delegate:self
cancelButtonTitle:@"Cancel"
destructiveButtonTitle:@"Delete File"
otherButtonTitles:@"Rename File", @"Email File", nil];
[menu showInView:contentView];
}
```

4.6 “请稍候”：向用户显示进度

等待是计算体验中固有的一部分，在可预见的将来仍然如此。作为一名开发人员，将这个事实告诉用户是你的工作。Cocoa Touch提供了许多类，用于通知用户等待进程完成。这些进度指示器采用两种形式：进度轮，它在其演示期间一直存在；进度条，在整个进程中，它从左侧向右侧填充。提供这些指示的类如下所示。

- ❑ **UIActivityIndicatorView**。该进度指示器提供一个旋转的圆，通知用户等待，而不提供有关其完成程度的具体信息。iPhone活动指示器非常小，但它的活动动画会吸引用户的眼球并且最适合用于正常应用程序中的快速中断。
- ❑ **UIProgressView**。该视图显示一个进度条。该条提供有关已经完成的工作量以及剩余但仍然占用相对较小数量的屏幕空间的工作量的具体反馈。它显示为一个狭长、水平的矩形，随着进度的前进它从左向右填充。这种经典的用户界面元素对于长时间延迟来说效果最佳，因为在这期间用户想知道作业已经完成了多少。

从文档中未记录的特性的角度来看，你拥有以下内容。

- ❑ **UIProgressHUD**。文档中未记录的“小心显示”版本的进度指示器浮在窗口中的所有其他视图上方，并且向基本进度指示器添加一个状态消息。HUD在进度条和简单指示器之间提供一个中间立场。你不需要采用进度条的方式量化你的进度，但可以利用关联文本描绘进度。当你想指出伴随某个进程所发生的情况时，进度HUD工作特别好（例如，“联系服务器”、“身份验证”、“请求数据”等）。

4.7 秘诀：调用基本的文档中未记录的 UIProgressHUD

UIProgressHUD类使用一个状态对话框来覆盖窗口。该对话框包含一个旋转的进度指示器和一个可指定的简短消息。就编程而言，你只需要实例化对象，使用主窗口初始化它，并通过发送show:YES告诉它执行的操作。图4-6显示了一个实际运行的UIProgressHUD显示，秘诀4-5显示了创建该显示的代码。

通过UIProgressHUD，你可以根据需要更新其文本。随时都可发送一个新的setText:消息。完成后，使用show:NO关闭它。如果需要，你可以通过setFontSize:调整字体大小，但是我建议你确保消息简短扼要。

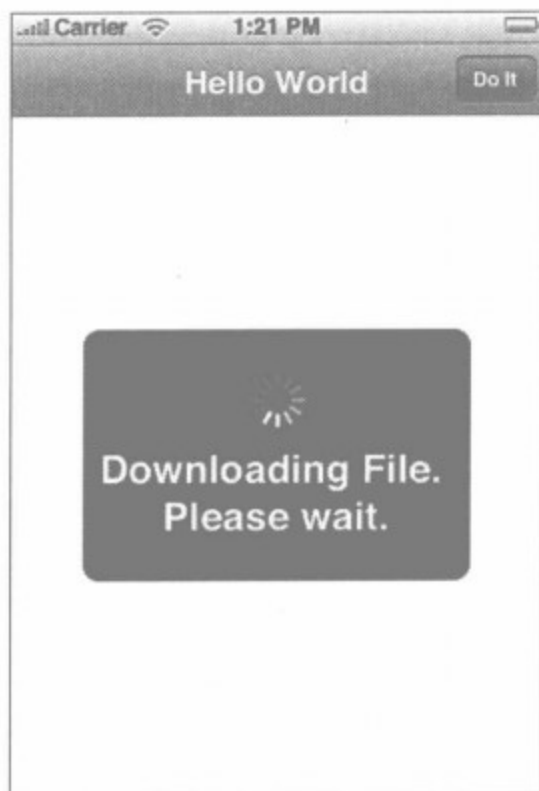


图4-6 UIPressHUD的名称表示它是一个HUD。它添加了一个覆盖在窗口之上的进度条。你告诉它什么时候显示，什么时候隐藏以及内容是什么

说明 令人失望的是，苹果公司并没有将它包含在标准SDK类中。这里，我已经提出了解决办法。到这本书出版时，我真心地希望苹果公司将它添加回Cocoa Touch。

使用带数据密集型代码的HUD显示时，你会希望让它在其自己的线程中运行，以便UI更新不会阻止。为此，可以使用NSThread的detatchThreadSelector: toTarget: withObject:方法。使用单独的线程使其启动，更新HUD的文本并在完成后关闭HUD。

秘诀4-5 为窗口创建一个UIProgressHUD

```
@interface UIPressHUD : NSObject
- (void) show: (BOOL) yesOrNo;
- (UIProgressHUD *) initWithWindow: (UIView *) window;
- (void) setText: (NSString *) theText;
@end

- (void) killHUD: (id) aHUD
{
    [aHUD show:NO];
    [aHUD release];
}

- (void) presentSheet
{
    id HUD = [[UIProgressHUD alloc] initWithWindow:[contentView superview]];
    [HUD setText:@"Downloading File. Please wait."];
    [HUD show:YES];
}
```

```
[self performSelector:@selector(killHUD:)
    withObject:HUD afterDelay:5.0];
}
```

4.8 秘诀：使用 UIActivityIndicatorView

UIActivityIndicatorView实例提供轻型视图，这些视图显示一个标准的旋转进度轮。当使用这些视图时，最重要的一个关键词是小。20×20像素是大多数指示器样式获得最清楚显示效果的大小。只要稍大一点，指示器都会变得模糊。图4-7显示了一个40像素的版本。

你需要在屏幕上将该指示器居中。将其放置在最方便操作的位置。作为背面清晰的视图，指示器将混合位于其后的背景视图。该背景的主要颜色帮助选择要使用的指示器样式。

对于一般用途，只需将活动指示器作为子视图添加到你要覆盖的窗口、视图、工具栏或导航栏。分配该指示器并使用一个框来初始化它，最好位于当前所使用的父视图中央。

通过将指示器的animating属性更新为YES来启动它。若要停止，将该属性设置为NO。Cocoa Touch会负责完成其余工作，在视图不使用时隐藏视图。

iPhone提供了几种不同样式的UIActivityIndicatorView类。UIActivityIndicatorViewStyleWhite和UIActivityIndicatorViewStyleGray是最简洁的。黑色背景下最适合白色版本的外观，白色背景最适合灰色外观（如图4-7所示）。它非常瘦小，而且采用夏普风格。选择白色还是灰色时要格外注意。全白显示在白色背景下将不能显示任何内容。而UIActivityIndicatorViewStyleWhiteLarge只能用于深色背景。它提供最大、最清晰的指示器。秘诀4-6显示了创建这些简单的UIActivityIndicatorView实例的代码。



图4-7 UIActivityIndicatorView类提供了一个简单的旋转轮，这意味着将以相对较小的尺寸显示

秘诀4-6 向程序中添加UIActivityIndicatorView

```

- (void) performAction
{
    if (progressShowing) [activityIndicator stopAnimating];
    else [activityIndicator startAnimating];
    progressShowing = !progressShowing;
}

- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen]
➡applicationFrame]];
    self.view = contentView;
    contentView.backgroundColor = [UIColor whiteColor];
    [contentView release];
    // Add an action button
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                                initWithTitle:@"Do It"
                                                style:UIBarButtonItemStylePlain
                                                target:self
                                                action:@selector(performAction)]
                                                autorelease];

    // Add the progress indicator but do not start it
    progressShowing = NO;
    activityIndicator = [[UIActivityIndicatorView alloc]
➡initWithFrame:CGRectMake(0.0f, 0.0f, 32.0f, 32.0f)];
    [activityIndicator setCenter:CGPointMake(160.0f, 208.0f)];
    [activityIndicator
➡setActivityIndicatorViewStyle:UIActivityIndicatorViewStyleGray];
    [contentView addSubview:activityIndicator];
    [activityIndicator release];
}

```

4.9 秘诀：构建 UIProgressView

进度视图可以让用户在执行任务时了解任务进度，而不仅仅是告诉用户“请稍候”。会显示一条从左向右填充的进度条，表示任务完成的程度。进度条最适合长时间的等待，它提供状态反馈，从而使用户能够保持控制感。

若要创建进度视图，请分配该视图并设置它的框。若要使用进度条，请调用`setProgress:`。它只带一个参数，一个范围介于0.0（无进度）和1.0（已完成）之间的浮点数字。进度视图条有两种样式：基本白色或浅灰色。可以通过`setStyle:`方法选择你喜欢的种类。

与其他种类的进度指示器不同的是，显示和隐藏进度条的视图将完全由你来决定。它没有`setVisible:`方法。我喜欢向操作表中添加进度条。这样既可以将它们带到屏幕上，又能关闭它们。另一个好处是当显示警告表时，屏幕的其他部分会变暗。这样便可在你的任务进行时强制采用模式显示。用户无法与GUI交互，直到关闭警告为止。秘诀4-7显示了一个UIActionSheet/

UIProgressView示例, 该示例生成的显示如图4-8所示。

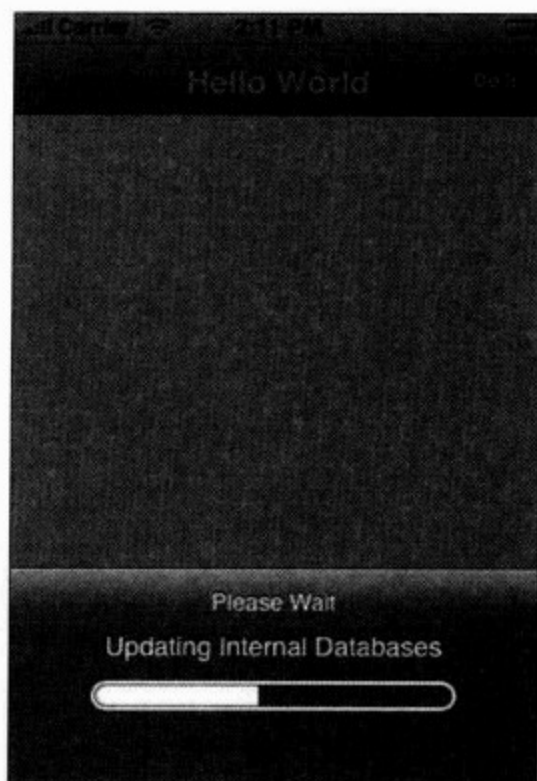


图4-8 使用UIProgressView实例跟踪长时间延迟的进度。将它们添加到一个UIActionSheet简化它们的显示和关闭

该秘诀使用文档中未记录的setNumberOfRows:调用。若要停留在SDK标准之内, 就向标题中添加几个新行(\n)。

说明 在操作表中嵌入进度条时, 你需要实现UIActionSheet委托协议。添加一个按钮单击方法符合该协议所需的方法, 即使该方法本身未执行任何操作。

秘诀4-7 向警告表中添加进度条

```
@interface UIActionSheet (extended)
- (void) setNumberOfRows: (NSInteger) rows;
@end

@implementation HelloController

// This callback fakes progress via setProgress:
- (void) incrementBar: (id) timer
{
    amountDone += 1.0f;
    [progbar setProgress: (amountDone / 20.0)];
    if (amountDone > 20.0) {[baseSheet dismissWithClickedButtonIndex:0
        animated:YES]; [timer invalidate];}
}

// Dismiss the action sheet backdrop used here
```


4.10 秘诀：添加自定义、可轻击的覆盖层

当与用户交流时，你需要限制使用标准的（甚至是文档中未记录的）UIKit对象。通过在视图上添加覆盖层，你可以创造无限的可能。你可以抛出文本、图像、动画以及你所拥有的任何内容。可以让覆盖层具有交互性（如此秘诀中所示），或者允许通过轻击进入下层界面。可以添加简单的状态信息（如透明背景上白色、灰色或黑色的文本），或者可以构建详细的显示。这些选择由你自己决定。

图4-9显示了在秘诀4-8中构建的自定义“请稍候”覆盖层。添加UIView动画，如第2章中所述，使覆盖层的入口变得平滑，并退出显示。通过在覆盖层上轻击来关闭它，从而返回到下层界面。可以看到，该覆盖层非常繁忙，并且其作用主要是提供一个干净、明亮的背景。若要使用这种类型的覆盖层以及复杂的界面，你可能需要用半透明的白色来填充透明区域，这样会部分隐藏界面并添加一个仍然能看见任何消息的背景。

如果你想添加一个类似于此的覆盖层并提供与UIProgressHUD相似的背景，应尽量与苹果公司的整体图形设计一致。在自定义界面元素中，用户对错误尤为敏感。



图4-9 结合透明度和动画，创建可弹出的视图覆盖层

秘诀4-8 创建自定义动画UIView覆盖层

```
@interface PleaseWaitView : UIImageView
@end

@implementation PleaseWaitView
- (PleaseWaitView *) initWithFrame: (CGRect) rect
{
    self = [super initWithFrame:rect];
    [self setImage:[UIImage imageNamed:@"PlsWait.png"]];

    UIImageView *imgView = [[UIImageView alloc] initWithFrame:CGRectMake(40.0f,
    300.0f, 60.0f, 60.0f)];
    [self addSubview:imgView];
    [imgView release];

    // load in the animation cells for the butterfly
    NSMutableArray *bflies = [[NSMutableArray alloc] init];
    for (int i = 1; i <= 17; i++) {
        NSString *cname = [NSString stringWithFormat:@"bf_%d.png", i];
        UIImage *img = [UIImage imageNamed:cname];
```

```
        if (img) [bflies addObject:img];
    }

    // begin the animation
    [imgView setAnimationImages:bflies];
    imgView.animationDuration = 0.75f;
    [imgView startAnimating];
    [bflies release];

    return self;
}

- (void)touchesBegan: (NSSet *) touches withEvent:(UIEvent *)event
{
    // fade away the overlay
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];

    [self setAlpha:0.0f];

    // Complete the animation
    [UIView commitAnimations];
    [pleaseWait setInteractionEnabled:YES];
}

@end

@interface HelloController : UIViewController
{
    UIView *contentView;
    PleaseWaitView *pleaseWait;
}
@end

@implementation HelloController

CGPoint randomPoint() { return CGPointMake(random() % 256, random() % 256); }

- (void) moveButterfly: (NSTimer *) timer
{
    if ([pleaseWait alpha] == 0.0f) {[timer invalidate]; return;}

    // Create an animation block around moving the butterfly to a new origin
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];
```



```
// Move the butterfly
UIImageView *iv = [[pleaseWait subviews] objectAtIndex:0];
CGRect frame = [iv frame];
frame.origin = randomPoint();
[iv setFrame:frame];

// Complete the animation
[UIView commitAnimations];
}

- (void) presentSheet
{
    if ([pleaseWait alpha] != 0.0f) return;

    // Create an animation block around fading in the overlay
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];

    [pleaseWait setAlpha:1.0f];

    // Complete the animation
    [UIView commitAnimations];

    // Give life to the butterfly
    [NSTimer scheduledTimerWithTimeInterval: 4.0f target: self selector:
    ➡@selector(moveButterfly:)
    userInfo: nil repeats: YES];
}

- (id) init
{
    if (self = [super init]) self.title = @"Hello World";
    return self;
}

- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen]
    ➡applicationFrame]];
    self.view = contentView;
    contentView.backgroundColor = [UIColor whiteColor];
    [contentView release]; // reduce retain count by one

    // Add an action button
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
```

```

initWithTitle:@"Do It"
style:UIBarButtonItemStylePlain
target:self
action:@selector(presentSheet)
autorelease];

// Create and hide the "Please Wait" view
pleaseWait = [[PleaseWaitView alloc] initWithFrame:[UIScreen mainScreen]
➡applicationFrame]];
[contentView addSubview:pleaseWait];
[pleaseWait setAlpha:0.0f];
}

-(void) dealloc
{
    [pleaseWait release];
    [contentView release];
    [super dealloc];
}
@end

```

4.11 秘诀：构建下滑式警告

在构建之前介绍的动画块时，你可以采用相同的方法来创建一个比较传统的下滑式警告。秘诀4-9让某视图框的位置有了动画效果，从而使视图看上去具有滚动的屏幕。该框刚开始不在屏幕中，然后以动画的方式显示在屏幕上，如图4-10所示。当用户单击OK时，进程反向。你可以使用该方法构建任何种类的警告。

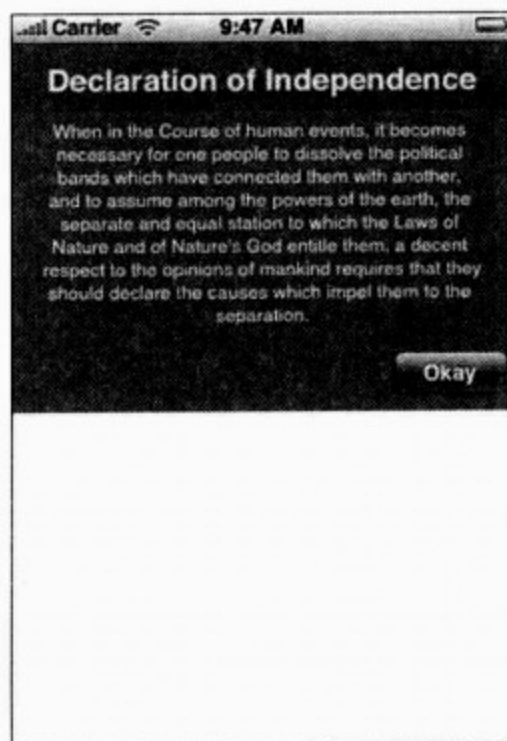


图4-10 使用UIViewAnimation创建自定义下滑式警告表

该秘诀中的视图包含一个标题、一个文本块以及一个简单的按钮。向你自己的实现中添加任何所需的UIView元素或控件。第5章中的一个示例就采用了本秘诀的方法，它在向下滑动的视图中添加用户可以选择的滚动表。

秘诀4-9 构建自定义下滑式警告

```
// Create a color of blue that mimics the official gray highlighting
UIColor *sysBlueColor(float percent) {
    float red = percent * 255.0f;
    float green = (red + 20.0f) / 255.0f;
    float blue = (red + 45.0f) / 255.0f;
    if (green > 1.0) green = 1.0f;
    if (blue > 1.0f) blue = 1.0f;

    return [UIColor colorWithRed:percent green:green blue:blue alpha:1.0f];
}

@interface TopAlert : UIView
{
    UILabel *title, *message;
}
- (void) setTitle: (NSString *)titleText;
- (void) setMessage: (NSString *)messageText;
@end

@implementation TopAlert

- (void) setTitle: (NSString *)titleText
{
    [title setText:titleText];
}
- (void) setMessage: (NSString *)messageText
{
    [message setText:messageText];
}

- (TopAlert *) initWithFrame: (CGRect) rect
{
    rect.origin.y = 20.0f - rect.size.height; // Place above status bar
    self = [super initWithFrame:rect];

    [self setAlpha:0.9];
    [self setBackgroundColor: sysBlueColor(0.4f)];

    // Add button
    UIButton *button = [[UIButton buttonWithType:UIButtonTypeCustom]
        initWithFrame:CGRectMake(220.0f, 200.0f, 80.0f, 32.0f)];
    [button setBackgroundImage:[UIImage imageNamed:@"whiteButton.png"]
        forState:UIControlStateNormal];
    [button setTitle:@"Okay" forState: UIControlStateHighlighted];
```

```
[button setTitle:@"Okay" forState: UIControlStateNormal];
[button setFont:[UIFont boldSystemFontOfSize:14.0f]];
[button addTarget:self action:@selector(removeView)
➤forControlEvents:UIControlEventsTouchUpInside];
[self addSubview:button];

// Add title
title = [[UILabel alloc] initWithFrame:CGRectMake(0.0f, 8.0f, 320.0f, 32.0f)];
title.text = @"Declaration of Independence";
title.textAlignment = NSTextAlignmentCenter;
title.textColor = [UIColor whiteColor];
title.backgroundColor = [UIColor clearColor];
// Alternative framed title:
// title.backgroundColor = [UIColor colorWithRed:1.0f green:1.0f blue:1.0f
➤alpha:0.25f];
title.font = [UIFont boldSystemFontOfSize:20.0f];
[self addSubview:title];
[title release];

// Add message
message = [[UILabel alloc] initWithFrame:CGRectMake(20.0f, 40.0f, 280.0f,
➤200.0f - 48.0f)];
message.text = @"When in the Course of human events, it becomes necessary for
➤one people to dissolve the political bands which have connected them with
➤another, and to assume among the powers of the earth, the separate and equal
➤which the Laws of Nature and of Nature's God entitle them, a decent respect
➤to the opinions of mankind requires that they should declare the causes
➤which impel them to the separation.";
message.textAlignment = NSTextAlignmentCenter;
message.numberOfLines = 999;
message.textColor = [UIColor whiteColor];
message.backgroundColor = [UIColor clearColor];
message.lineBreakMode = UILineBreakModeWordWrap;
message.font = [UIFont systemFontOfSize:[UIFont smallSystemFontSize]];
[self addSubview:message];
[message release];

return self;
}

- (void) removeView
{
    // Scroll away the overlay
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:0.5];

    CGRect rect = [self frame];
    rect.origin.y = -10.0f - rect.size.height;
```



```

        [self setFrame:rect];

        // Complete the animation
        [UIView commitAnimations];
    }

    - (void) presentViewController
    {
        // Scroll in the overlay
        CGContextRef context = UIGraphicsGetCurrentContext();
        [UIView beginAnimations:nil context:context];
        [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
        [UIView setAnimationDuration:0.5];

        CGRect rect = [self frame];
        rect.origin.y = 0.0f;
        [self setFrame:rect];

        // Complete the animation
        [UIView commitAnimations];
    }

    - (void) dealloc
    {
        [title release];
        [message release];
        [super dealloc];
    }

@end

```

4.12 秘诀：添加状态栏图像

位于iPhone顶部的栏提供了丰富的状态信息。这些应用程序触发的状态符号位于时间显示的右侧。比较常用的图标包括暂停/播放指示器、电池使用等。UIApplication实例可以使用文档中未记录的方法向状态栏中添加现有图像，但是我无法让状态栏调用使用来自应用程序包自身的数据。所有状态栏图像都存储在SpringBoard.app中的/System/Library/CoreServices/SpringBoard.app下。因此不难发现，这非常符合App Store协议的边界条件。

此限制有两个意义深远的结果。首先，你能使用的图像非常有限（它们只是标准的PNG，使用透明度）；其次，需要向用户的System文件夹中添加自定义图像，这会带来安全性问题并影响操作系统的整体性能。这种方式不好，而且不安全，并且开发人员和苹果公司之间签有协议和沙盒。

秘诀4-10显示了如何加载和删除这些图像。若要设置图标，请调用addStatusBarImageNamed:，并传递一个参数（不带扩展的图标名）。作为一项规则，每个图像有两个版本，一个用于黑色的状态栏（每个名称都以FSO_开头），一个用于白色/灰色状态栏（名称以Default_开头）。所有更改在退出程序之后都会持续存在。这也表明你应该谨慎地使用状态栏更新。

例如，在传递时，@"Airplane"（飞机模式图标）、SpringBoard显示Default_Airplane.png（针对黑色状态栏）。你没有传递扩展名或前缀。使用removeStatusBarImageNamed:删除图标。图4-11显示了飞机图标，如图所示。

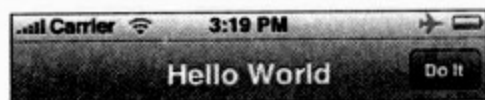


图4-11 使用状态栏图标，可以与用户交流模式信息

在应用程序结束之后，状态栏图像会仍然存在。在状态栏中添加一个飞机，然后退出，飞机仍然存在。如果你决定使用状态栏图像，则考虑在挂起或终止程序之前清除状态栏。按照定义，这些图像的目的是指出超过任何应用程序会话范围的行为。由于苹果公司坚决要求“一次一个应用程序”，因此在该限制之内应尽量少用这种技术。

秘诀4-10 添加和删除状态栏图标

```
@interface UIApplication (extended)
- (void) addStatusBarImageNamed:(NSString *)aName;
- (void) removeStatusBarImageNamed:(NSString *)aName;
@end

- (void) performAction
{
    isShowing = !isShowing;

    if (isShowing)
        [[UIApplication sharedApplication] addStatusBarImageNamed:@"Airplane"];
    else
        [[UIApplication sharedApplication] removeStatusBarImageNamed:@"Airplane"];
}
```

4.13 添加应用程序标记

如果曾经使用过iPhone或iPod touch，你可能在主屏幕的应用程序上面看到过红色的小标记（badge）。这可能表示自从用户上次打开电话或电子邮件以来累计的未接电话数量或未读的电子邮件数量。

实际上，可采用3种方法来标记应用程序：一种是非常简单的UIApplication调用；另一种有点复杂，需要在UIKit中建立通道；最后一种是使用从Web服务器发布的SDK push功能。若要在应用程序内部设置标记，则可以使用文档中未记录的setApplicationBadge:调用或设置正式的applicationIconBadgeNumber属性。将NSString作为常规标记参数传递给该调用，将字符串大小限制为4或5（最多）个字符，或将标记数量属性限制为NSInteger。例如，你可以用当前月份的3个字母缩写来标记应用程序，如代码清单4-2所示。

说明 若要隐藏标记，将applicationIconBadgeNumber设置为0（数字零），或将setApplicationBadge:设置为@""（空字符串）。

代码清单4-2 使用当前月份标记应用程序

```

NSDate *now = [NSDate dateWithTimeIntervalSinceNow:0];
NSString *caldate = [[now
    dateWithCalendarFormat:@"%b"
    timeZone:nil] description];
[[UIApplication sharedApplication] setApplicationBadge:caldate];

```

若要删除应用程序标记，请传递空字符串（如@" "），这将从图标中删除任何现有标志。如果想获得一个“空”标记，则传递一个空格字符串@" "。

UIApplication方法的问题在于，若要使用它，必须将你的请求直接放置在应用程序下，然后只能以数值表示。你可能希望标记图标来自应用程序外部。这是开发人员将希望了解的事情，尤其是你不打算通过应用程序存储分发某些应用程序时。代码清单4-3演示了如何使用公用UIKit框架的动态链接来标记应用程序。该代码不要求应用程序自身完成标记设置。而是调用SpringBoard来执行琐碎的工作。

此函数依赖动态链接。对UIKit框架进行反向工程展示了UIApplication的setApplicationBadge的工作原理。它调用具有一个字符串和一个应用程序标识符的SBSetApplicationBadge。通常，在日常编程中，我不赞成使用动态链接方法。图4-12显示了使用该功能标记几个图标的结果。

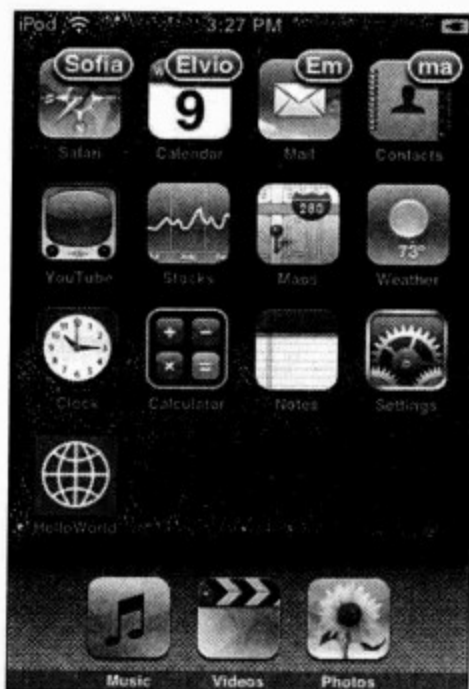


图4-12 应用程序标记显示为位于应用程序图标右上角的红色小容器

注意，最新的iPhone固件版本行为与标记不一致。第三方标志继续可靠地为/Applications文件夹中的项工作，但不能为位于沙盒中的那些应用程序可靠地工作。

说明 就App Store提交而言，仍然不确定SpringBoard调用是属于“文档中未记录但可以接受”类别还是属于“文档中未记录且不受支持”类别。应谨慎使用动态链接。

代码清单4-3 通过反向工程和动态链接实现常规标记

```

#include <dlfcn.h>
void badge(char *appid, char *badge)
{
    // link to UIKit
    void *uikit = dlopen("/System/Library/Frameworks/UIKit.framework/UIKit",
                        RTLD_LAZY);

    // Recover the SpringBoard Port
    int (*SBSSpringBoardServerPort)() =
        dlsym(uikit, "SBSSpringBoardServerPort");

    mach_port_t *p;
    p = SBSSpringBoardServerPort();

    // Perform the badging
    int (*doBadge)(mach_port_t* port, char* x, char *y) =
        dlsym(uikit, "SBSetApplicationBadge");

    doBadge(p, appid, badge);
    dlclose(uikit);
}

```

4.14 秘诀：简单的音频警告

音频警告直接对用户“说话”。它们产生即时反馈（假设用户的听力没有问题）。幸运的是，苹果公司通过系统音频服务将基本声音播放构建到Cocoa Touch SDK中。它的工作方式与Macintosh上的系统音频非常相似。或者也可以使用Audio Queue调用来创建音频警告，但价格非常高昂。Audio Queue播放对于程序来说“非常昂贵”，它涉及的内容比所需的简单的警告声音更加复杂。相比之下，你可以下载系统音频，然后仅使用几行代码来播放它。

苹果公司表示，让警告声音保持简短，两秒或更少时间最合适。虽是这么说，但我使用长度高达一分钟的声音进行了测试，它们使用系统音频服务播放起来非常不错。系统音频只播放PCM和IMA音频。这意味着将你的声音限制为AIFF、WAV和CAF格式。

若要构建系统声音，请通过一个指向声音文件的文件URL来调用AudioServicesCreateSystemSoundID。该调用返回一个初始化的系统声音对象，然后你可以随意播放该对象。只需借助声音对象来调用AudioServicesPlaySystemSound。此单一调用完成了所有工作。

你可以添加一个可选的系统声音完成回调，在声音完成播放（调用AudioServicesAddSystemSoundCompletion）之后通知程序，但实际上，你真的不需要这样做，尤其在使用苹果公司认可的简短声音时更是如此。

若要清除声音，请在使用它们之后，抛弃它们。可以使用正在讨论的声音对象调用AudioServicesDisposeSystemSoundID。

说明 若要使用这些系统声音服务，请确保在代码中包含Audio Toolbox/AudioServices.h，并且链接到音频工具箱框架。

震动

与音频警告相同，震动会立即引起用户的注意。而且，震动几乎适合所有用户，包括听力或视力受损的用户。使用秘诀4-11中的系统音频服务，你可以在播放声音的同时震动。你只需要通过下面这行调用来实现它：

```
AudioServicesPlaySystemSound (kSystemSoundID_Vibrate);
```

你无法改变震动参数。每个调用都会产生一个简短的一到两秒的嗡嗡声。在不支持震动的平台上（如iPod touch），该调用不执行任何操作，但也不会产生错误。

秘诀4-11 播放简单的音频警告

```
@interface HelloController : UIViewController
{
    UIView *contentView;
    SystemSoundID pmph;
}
@end

@implementation HelloController

- (void) playSound
{
    AudioServicesPlaySystemSound (pmph);
}

- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:[UIScreen mainScreen]
    ➤applicationFrame]];
    self.view = contentView;
    contentView.backgroundColor = [UIColor whiteColor];
    [contentView release];

    // Add an action button
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
        initWithTitle:@"Do It"
        style:UIBarButtonItemStylePlain
        target:self
        action:@selector(playSound)]
        autorelease];

    // Load the sound
    id sndpath = [[NSBundle mainBundle] pathForResource:@"pmph1" ofType:@"wav"
    ➤inDirectory:@""];
    CFURLRef baseUrl = (CFURLRef)[[NSURL alloc] initWithURLWithPath:sndpath];
    AudioServicesCreateSystemSoundID (baseUrl, &pmph);
}
```

```
-(void) dealloc
{
    if (pmpH) AudioServicesDisposeSystemSoundID(pmpH);
    [contentView release];
    [super dealloc];
}
@end
```

4.15 小结

本章介绍了直接与用户交互的方式。你已经学习了如何构建警告（视觉、听觉和触觉）以引起用户的注意，并且可以请求立即反馈。使用这些示例增强程序的交互性，并利用仅iPhone支持的独特特性。下面给出了本章的一些要点。

- ❑ 本章中的很多秘诀都依赖于文档中未记录的UIKit调用或动态链接（较差的方式）。我已经将这些秘诀包含在一本书中，该书侧重于正式的SDK开发，因为它们的功能不仅实用，而且经过了大部分iPhone开发人员的测试。本章中的所有调用都仅限于公用的SDK框架，并且没有违背苹果公司所声明的策略（至少理论上是这样）。
- ❑ 无论何时何任务占用大量时间时，一定要对用户有礼貌并且显示一种进度反馈。iPhone提供了许多可完成此操作的方法，例如弹出显示和状态栏指示器。
- ❑ 警告立刻引起用户注意。它们的作用是在交流信息时引起响应。而且，正如本章所述，它们具有极佳的可定制性。可以围绕简单的UIAlertSheet构建整个应用程序。
- ❑ 标记和状态栏图标都允许将应用程序状态扩展到当前应用程序之外。保守地使用这些特性可以提供大量的信息。
- ❑ 音频反馈就像单击和嘟嘟声一样，可以改进程序并丰富交互性。使用系统声音调用意味着使用iPod功能可以出色地播放声音，不会破坏当前的听觉体验。



表格提供一个交互类，该类尤其适合在小型的狭窄设备上工作。iPhone和iPod touch附带的许多甚至大多数应用程序都以表格为中心，包括Settings、iPod、YouTube、Stocks和Weather。iPhone的屏幕大小有限，因此支持滚动以及选择各个项的表格以简单、易于操作的形式提供信息和内容，这是一种理想方式。本章将探索iPhone表格的工作方式，研究开发人员可以使用哪种表格，以及如何在自己的程序中使用表格特性。

5.1 UITableView和UITableViewController简介

标准的iPhone表格由各个单元格的简单滚动列表组成，它提供交互式数据索引。用户可以上下滚动或换页，直到找到要交互的项为止。然后，用户可以独立于其他行来操作该项目。在iPhone上，表格无处不在。几乎每个标准软件包都使用它们，并且它们也形成了许多第三方应用程序的核心。在本节中，你将探索表格的工作原理，以及需要将哪些元素放到一起才能创建自己的表格。

iPhone SDK支持多种表格，所有这些表格都作为UITableView类实现。除了标准的单元格滚动列表（它提供大多数常规表格实现）之外，你可以创建若干专用的表格，包括你在首选项应用程序中看见的表格种类，它们具有灰色的背景以及圆形单元格边缘。表格具有许多部分以及一个类似在Contacts应用程序中用过的索引，一个具有滚轮式表格的UITableView子类，和用来设置约会日期和警报的那些表格一样。第6章将介绍这些专用表格样式的基础知识。不管你使用哪种类型的表格，它们都采用相同的常规工作方式。它们包含从数据源提供的一列单元格，通过调用定义明确的方法来响应用户的交互操作。

UITableViewController是UIViewController的一个子类。与其父类一样，它们使你能够通过最少的代码，以最便捷的方式构建屏幕上的视图。你将从本节中了解到，UITableViewController类大大简化了创建UITableView的过程，减少或取消了直接使用表格实例所需的重复性步骤。UITableViewController处理视图布局的繁琐细节，并且通过添加一个本地tableView实例变量以及对委托和数据源的自动表格协议支持，来提供特定于表格的便利。

5.1.1 创建表格

若要实现表格，必须定义3个关键元素：表格的布局方式、用来填充表格的内容种类以及表

格对用户交互操作做出反应的方式。若要指定这些元素，请向应用程序中添加描述和方法。当构建视图时，便创建了视觉布局，定义了根据要求提供表格单元的数据源，实现了响应用户交互操作（如行选择更改）的委托方法。

1. 对视图进行布局

顾名思义，UITableView就是视图。它们在iPhone屏幕上呈现交互式表格。UITableView类是从UIScrollView类继承的。这种继承提供了表格使用的上下滚动功能。和其他视图一样，UITableView实例通过框来定义其边界，它们可以是其他视图的子视图或父视图。若要创建表格视图，请对其进行分配（alloc），使用一个框来初始化它，就像任何其他视图一样（initWithFrame:），然后通过指定数据源和委托对象来添加所有簿记细节。

UITableViewController替你关注布局工作。它创建一个标准的UIViewController，并用一个UITableView来填充它，设置其框以便为任何导航栏或工具栏留出空间。你可以通过tableView实例变量来访问该表格视图。

一个重要说明是：当创建自定义UITableViewController子类时，应确保从子类的layoutView方法调用[super layoutView]。这样便可以在控制器内正确设置视图，并且能够在子类中添加自定义特性，如导航项目按钮。

2. 指定数据源

表格不直接拥有或复制它们使用的单元格。表格单元的存在独立于每个表格。UITableView实例依赖外部源根据需要提供新的或现有的表格单元。此外部源称为“数据源”，是指负责向表格的查询返回单元格的对象。

数据源基于索引路径提供表格单元。索引路径是NSIndexPath类的成员，它描述通过数据树到特殊节点的路径。数据源的工作是将该路径连接到具体的UITableViewCell实例，并根据要求返回该单元格。

iPhone SDK提供了一个用于重用表格单元的内在机制。你可以标记单元格以便重用，然后根据需要将它们弹出栈。这样便节省了内存，并且当用户快速滚动长列表屏幕时，提供了一种快速有效的反馈单元格的方法。

使用setDataSource:向表格指定一个对象作为其数据源。该对象必须实现UITableViewDataSource协议。最常见的情况是，拥有表格视图的UITableViewController充当该视图的数据源。当使用UITableViewController子类时，你需要指定<UITableViewDataSource>协议。UITableViewController类隐式支持该协议。

指定数据源之后，你可以继续操作并加载表格，直到具有它的单元格。调用表格的reloadData方法，表格开始查询其数据源，将实际的屏幕单元格加载到你的表格中。

3. 指定委托

和许多其他交互对象一样，UITableView实例使用委托来响应用户交互操作并执行有意义的响应。你的表格委托可以响应事件，如表格滚动或行选择更改。委托让表格将响应这些交互操作的责任移交给指定的对象，通常是拥有表格视图的UITableViewController对象。如果你直接使用UITableView，则使用标准的setDelegate:方法来设置表格的委托。该委托必须实现

UITableViewDelegate协议。当使用UITableViewController时，忽略setDelegate:方法和协议指定。该类会自动处理这些。苹果公司的SDK文档中列出了所有的委托方法。

说明 UITableView实例除了提供委托方法调用之外，还提供通知。通知能够通过广播更新使应用程序的不同线程彼此进行通信。你可以使你的应用程序订阅这些通知（使用标准的NSNotificationCenter调用）以确定表格状态发生更改的时间。

5.1.2 UITableViewController 的作用

UITableViewController类将一个UITableView嵌入到管理其表格视图的UIViewController对象中。它的视图名为tableView，你可以通过调用self.tableView来直接访问该视图。它自动设置数据源并且将表格视图的方法委托给自己。而且当通过setEditing:animated:调用向其父导航栏中添加Edit|Done按钮时，它还会自动实现编辑模式。

5.2 秘诀：创建简单的列表表格

许多字符串数组都可以用来设置和填充表格。本秘诀利用UIFont类的功能来列出可用的系统字体。[UIFontfamilyNames]调用返回一个用这些字体名称填充的NSArray，便于设置表格数据源。秘诀5-1基于这些字体名称创建一个基本表格。图5-1显示了该代码在iPhone仿真器上运行时产生的界面。（在仿真器上运行时会产生一个不同的字体集，因为它们基于运行SDK的Macintosh中可用的字体，而不是基于iPhone自身的字体。）



图5-1 基于任何字符串数组很容易用单元格填充UITableView。此处列出了UIFont类中的字体系列列表。当轻击时，所选择的字体将更新标题栏字体

5.2.1 数据源函数

若要显示表格，每个表格数据源（UITableViewController）必须提供3个核心方法。以下列表显示了这些方法，并在简单的列表视图中提供了有关其用法的注释，如图5-1所示。

- ❑ **numberOfSectionsInTableView**。表格必须确定它们是以各个部分显示其数据还是以一
个列表显示数据。对于简单的表格来说，此方法始终返回1，因为是在表格的一个部分中。
- ❑ **tableView: numberOfRowsInSectionSection**。由于简单列表只包含一个部分，因此这个方法
返回整个表的行数。对于比较复杂的列表，你会希望提供一种为每个部分返回报告的方法。
- ❑ **tableView: cellForRowAtIndexPath**。这是实际返回单元格的方法。使用索引路径的
row来确定要提供的单元格。

5.2.2 重用单元格

iPhone节省内存的一种方法是重用单元格。你可以为每个单元格指定一个ID字符串。这将指定单元格的类型。对于不同种类的单元格使用不同的ID。对于简单的表格，一个标识符即可完成工作。在秘诀5-1中，标识符为@"any-cell"。字符串是任意的。你可以按照你喜欢的方式定义它们，但是在使用多个单元格类型时，应该保证它们的名称有意义。

在分配新的单元格之前，都应检查是否可以使用可重用的单元格。如果表格的
dequeueReusableCellWithIdentifier:请求返回NULL，则需要为其分配新单元格。如果它返回一个单元格，则使用对于当前行索引有意义的信息自定义该单元格。没有必要将单元格添加到重用队列中。Cocoa Touch会替你处理所有这些细节。

5.2.3 字体表格示例

秘诀5-1演示了如何构建简单的基于列表的表格。它创建一个表格并用所有可用的字体系列填充该表格。轻击时，视图控制器将该字体分配给屏幕顶部蓝色导航栏中的标签，并将该系列的可用字体列表打印到调试器控制台。此行为定义在tableView: didSelectRowAtIndexPath:委托方法中。

此处使用UITableViewController作为委托是最好的选择，因为该表格的用户交互操作影响其视图。如果你不使用应用程序委托，则可以通过查询应用程序：[[UIApplication sharedApplication] delegate]来调用 setDelegate: 及该对象。这会覆盖标准的UITableViewController设置。

说明 表格可以用于通过选择蓝色或灰色覆盖为所选单元格设置颜色。将selectionStyle属性设置为 UITableViewCellSelectionStyleBlue 或 UITableViewCellSelectionStyleGray。如果你不想显示选择内容，则使用 UITableViewCellSelectionStyleNone。仍然可以选择单元格，但是不会显示覆盖。

秘诀5-1 创建简单的基于列表的表格

```
@interface HelloController : UITableViewController
@end

@implementation HelloController

- (HelloController *) init
{
    if (self = [super init]) self.title = @"Fonts";
    return self;
}

#pragma mark UITableViewDataSource Methods

// Only one section in this table
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}

// Return the number of rows in the table
- (NSInteger)tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section
{
    return [[UIFont familyNames] count];
}

// Return a cell for the specified index path
- (UITableViewCell *)tableView:(UITableView *)tableView
  cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView
  dequeueReusableCellWithIdentifier:@"any-cell"];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
  reuseIdentifier:@"any-cell"] autorelease];
    }
    // Set up the cell
    cell.text = [[UIFont familyNames] objectAtIndex:[indexPath row]];
    return cell;
}

#pragma mark UITableViewDelegateMethods

// Respond to user selection
- (void)tableView:(UITableView *)tableView
  didSelectRowAtIndexPath:(NSIndexPath *)newIndexPath
{
    printf("User selected row %d\n", [newIndexPath row] + 1);
    NSString *fontName = [[UIFont familyNames] objectAtIndex:[newIndexPath row]];
```

```

        CFShow([UIFont fontNamesForFamilyName:fontName]);
        [(UILabel *)self.navigationItem.titleView setFont:[UIFont fontWithName:
        ➤fontName size:[UIFont systemFontOfSize]]];
    }

#pragma mark Controller's loadView method
- (void)loadView
{
    [super loadView];

    // Add the custom title bar label
    self.navigationItem.titleView = [[UILabel alloc]
    ➤initWithFrame:CGRectMake(0.0f, 4.0f, 320.0f, 36.0f)];
    [(UILabel *)self.navigationItem.titleView setText:@"Font Families"];
    [(UILabel *)self.navigationItem.titleView setBackgroundColor:[UIColor
    ➤clearColor]];
    [(UILabel *)self.navigationItem.titleView setTextColor:[UIColor whiteColor]];
    [(UILabel *)self.navigationItem.titleView
    ➤setTextAlignment:UITextAlignmentCenter];
    [(UILabel *)self.navigationItem.titleView setFont:[UIFont
    ➤boldSystemFontOfSize:[UIFont systemFontOfSize]]];
}

- (void) dealloc
{
    [super dealloc];
}

@end

```

5.3 秘诀：创建基于表格的选择表

上述秘诀显示了如何创建非常简单的字体表系列。此秘诀深化了这一思想，它将该表格嵌入到视图中，并根据用户选择来呈现它（参见图5-2）。表格可以出色地为用户提供类似选项的选择内容。Mobile Safari已经在UIPickerTableView显示上实现了标准化，但是我认为一个简单的平面滚动列表提供了许多内容。Picker视图使用的类似金属的外观非常不和谐，但是我的设计技巧可能非常有限，这个UITableView选择应该能够在许多应用程序中找到家。

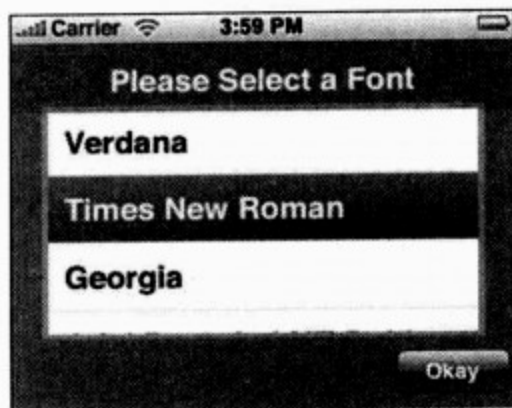


图5-2 将表格嵌入到其他视图中，为用户提供另一种在列表中进行选择的方法

注意，此秘诀并未使用UITableViewController。该解决方案围绕一个自定义视图，该视图具有一个表格和一个按钮，用于指出选择内容。这不在UITableViewController的范围之内，并且通过动画使UIView子类进入该位置来呈现。

秘诀5-2创建一个自定义视图FontPickerSheet，并使用一个UIView动画块（参见第2章）根据需要将其滑到屏幕上。用户选择存储在本地的selection属性中，可以从UITableViewController查询该属性。值为NULL表示用户无法进行选择。否则，它返回一个NSString，即所选择字体的名称。

此处我没有包含位于表和其下视图之间的半透明的UIView，这可能是你希望考虑的内容。该视图将发挥两个作用。首先，它将模糊你的主应用程序视图；其次，它将阻止与该视图的交互。这样便在将控制返回主应用程序之前强制模式演示，它是一种将你的用户限制为一个视图的交互样式。

秘诀5-2 将表格嵌入到表中并恢复选择

```
@interface FontPickerSheet : UIView <UITableViewDataSource, UITableViewDelegate>
{
    UILabel *title;
    UITableView *tableView;
    NSString *selection;
}
@property (nonatomic, retain) NSString *selection;
@end

@implementation FontPickerSheet
@synthesize selection;

#pragma mark UITableViewDataSource Methods

// Only one section in this table
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}

// One row for each family font name
- (NSInteger)tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section
{
    return [[UIFont familyNames] count];
}

// Return a cell for the ith row
- (UITableViewCell *)tableView:(UITableView *)tView
  cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tView dequeueReusableCellWithIdentifier:@"any-cell"];
```

```

    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
            reuseIdentifier:@"any-cell"] autorelease];
    }
    // Set up the cell
    cell.text = [[UIFont familyNames] objectAtIndex:[indexPath row]];
    return cell;
}

#pragma mark UITableViewDelegateMethods

// Respond to user selection
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)
    newIndexPath
{
    selection = [[[UIFont familyNames] objectAtIndex:[newIndexPath row]] retain];
}

#pragma mark FontPickerSheet customization

- (FontPickerSheet *) initWithFrame: (CGRect) rect
{
    rect.origin.y = 0.0f - rect.size.height; // Place above status bar
    self = [super initWithFrame:rect];
    [self setAlpha:0.9];
    [self setBackgroundColor:sysBlueColor(0.4f)];

    // Add button
    UIButton *button = [[UIButton buttonWithType:UIButtonTypeCustom]
        initWithFrame:CGRectMake(220.0f, 200.0f, 80.0f, 32.0f)];
    [button setBackgroundImage:[UIImage imageNamed:@"whiteButton.png"]
        forState:UIControlStateNormal];
    [button setTitle:@"Okay" forState: UIControlStateHighlighted];
    [button setTitle:@"Okay" forState: UIControlStateNormal];
    [button setFont:[UIFont boldSystemFontOfSize:14.0f]];
    [button addTarget:self action:@selector(removeView)
        forControlEvents:UIControlEventTouchUpInside];
    [self addSubview:button];

    // Add title
    title = [[UILabel alloc] initWithFrame:CGRectMake(0.0f, 8.0f, 320.0f, 32.0f)];
    title.text = @"Please Select a Font";
    title.textAlignment = UITextAlignmentCenter;
    title.textColor = [UIColor whiteColor];
    title.backgroundColor = [UIColor clearColor];
    title.font = [UIFont boldSystemFontOfSize:20.0f];
    [self addSubview:title];
    [title release];

    // Add border for the table
    CGRect bounds = CGRectMake(20.0f, 40.0f, 280.0f, 200.0f - 48.0f);
    UIView *borderView = [[UIView alloc] initWithFrame:bounds];

```

```
[borderView setBackgroundColor:sysBlueColor(0.55f)];
[self addSubview:borderView];
[borderView release];

// Add table
tableView = [[UITableView alloc] initWithFrame:CGRectInset(bounds, 4.0f, 4.0f)
               style:UITableViewStylePlain];
tableView.backgroundColor = [UIColor whiteColor];
tableView.delegate = self;
tableView.dataSource = self;
[tableView reloadData];
[self addSubview:tableView];
[tableView release];

return self;
}

- (void) removeView
{
    [tableView deselectRowAtIndexPath:[tableView indexPathForSelectedRow]
     animated:NO];

    // Scroll away the overlay
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:0.5];

    CGRect rect = [self frame];
    rect.origin.y = 0.0f - rect.size.height;
    [self setFrame:rect];

    // Complete the animation
    [UIView commitAnimations];
}

- (void) presentView
{
    selection = NULL;

    // Scroll in the overlay
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:0.5];

    CGRect rect = [self frame];
    rect.origin.y = 0.0f;
    [self setFrame:rect];

    // Complete the animation
    [UIView commitAnimations];
}
```

```

-(void) dealloc
{
    [title release];
    [tableView release];
    [super dealloc];
}
@end

```

5.4 秘诀：将图像加载到表格单元中

通过将图像分配给各个单元格，将图像添加到表格。秘诀5-3中，UITableViewController类使你能够将小型缩略图放置到任何单元格的左侧（如图5-3所示），而不需要任何特定的编程，只需将单元格的image属性设置为UIImage。该图像不会干扰该单元格的正常text属性。任何文本都出现在图像的右侧，并会自动考虑空格。

你必须知道你使用的图像的大致尺寸，并且相应调整表格视图的rowHeight属性。该示例将行高设置为64，留有许多空间以容纳57像素的方形应用程序图标。

该示例在你的iPhone上扫描应用程序图标。它使用NSFileManager来访问/Applications文件夹，并扫描各应用程序的icon.png。找到之后，它使用该图标标记每个单元格。如果没有找到，则为其自己的应用程序添加该图标。你可能希望看到大多数应用程序的图标，包括Calculator和Maps，但两个MobileSlideshow应用程序（Photos和Camera）除外，因为它们不提供单个的icon.png文件。



图5-3 UITableViewController使得向表格和文本标记中添加缩略图更加容易

说明 如果你想启动所选择的程序，请调用文档中未记录的UIApplication方法launchApplicationWithIdentifier:suspended:，将其传递给应用程序ID（如com.apple.Calculator）。但是，应用程序存储协议会严格禁止启动应用程序。

秘诀5-3 向表格中添加图像缩略图

```

// Return a cell for the ith row
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"any-cell"];

```



```

if (cell == nil) {
    cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
        reuseIdentifier:@"any-cell"] autorelease];
}
// Set up the cell
NSString *appname = [[[NSFileManager defaultManager]
    directoryContentsAtPath:@"Applications"] objectAtIndex:indexPath.row];
cell.text = appname;

NSString *imagePath = [NSString stringWithFormat:@"%@/icon.png",
    appname];
printf("app: %s\n", [imagePath UTF8String]);
if ([[NSFileManager defaultManager] fileExistsAtPath:imagePath])
    cell.image = [UIImage imageWithContentsOfFile:imagePath];
else
    cell.image = [UIImage imageWithContentsOfFile:[NSBundle mainBundle]
        pathForResource:@"Icon" ofType:@"png" inDirectory:@"/"]];
return cell;
}

- (void)loadView
{
    [super loadView];
    [self.tableView setRowHeight:64];
}

```

5.5 秘诀：设置单元格的文本特性

每个UITableViewCell都提供全面的文本特性支持。你可以设置单元格标签的颜色、字体、大小和对齐方式等。秘诀5-4显示了创建简单的由3个项目组成的表格的数据源方法，为每个单元格指定独特的颜色和字体。尽管这会造成视觉上的混乱，如图5-4所示，但是你可以看到如何通过指定视觉线索，将表格行与其同伴区分开来。

你可以使用不同的文本样式来指出哪些表格元素“繁忙”且无法访问，哪些项目尚未检查。在之前的一个iPhone实用程序（SendFile，参见ericasadun.com）中，我使用颜色来告诉用户哪个文件正在通过Bonjour的免费配置的网络共享。若要指定文本特性，请使用标准的文本属性，如textColor和font。

秘诀5-4任意指定了这些种类的特性，以阐明概念。它证明每个单元格都可以拥有其自己的文本样式。

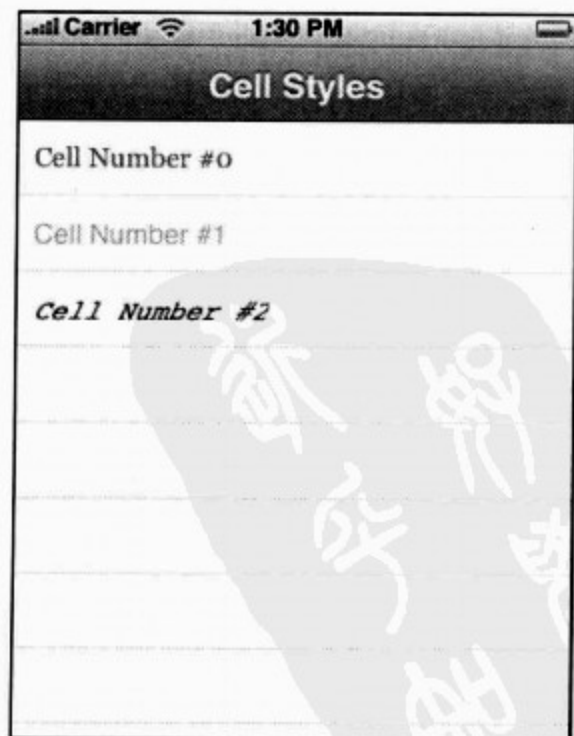


图5-4 使用UITableViewCell可以直接访问其标签的文本特性

秘诀5-4 为表格单元指定文本特性

```

// Return a cell for the ith row
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"any-cell"];
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithFrame:CGRectZero
            reuseIdentifier:@"any-cell"] autorelease];
    }
    // Set up the cell
    cell.text = [NSString stringWithFormat:@"Cell Number %d", [indexPath row]];
    cell.font = [UIFont fontWithName:[NSArray arrayWithObjects:@"Georgia",
        @"Helvetica", @"Courier", nil]
        objectAtIndex:[indexPath row]] size:16];
    cell.textColor = [[NSArray arrayWithObjects:
        [UIColor redColor],
        [UIColor lightGrayColor],
        [UIColor blueColor], nil]
        objectAtIndex:[indexPath row]];

    return cell;
}

```

5.6 秘诀：删除单元格选择

我对UITableView类（并通过关联UITableViewController）不满意的地方就是它对单元格重选的历史态度。根据iPhone SDK Beta程序的说明，UITableView实例将不允许你重新选择当前选择的单元格并触发委托方法调用，因为现在你可以在最新的SDK版本中执行该操作。你始终都至少有一个方法来获取表格单元，以允许它被选择两次（通过删除当前选择以便该表格准备接受下一次轻击来完成该操作）。

当使用委托方法时，你不能只告诉所选择单元格来取消对它自己的选择。没有[aCell setSelected:NO];方法。但是你必须与表格交互并让其取消选择正被讨论的单元格，如代码清单5-1所示。此处，半秒之后，每个用户选择都会触发一个延迟的取消选择（deselect方法）。

既然单元格支持多次轻击，因此没有使用取消选择功能，而是从语义上实现取消选择。具有选定项的表格可能与没有选择的表格提供的选项不同。例如，在一个音频应用程序中，当用户从列表选择一个声音时，可能只显示Play或Pause按钮。

代码清单5-1 删除表格的单元格选择

```

- (void) deselect: (id) sender
{
    [tableView deselectRowAtIndexPath:[tableView indexPathForSelectedRow]
        animated:YES];
}

```

```
// Respond to user selection
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)
newIndexPath
{
    printf("User selected row %d\n", [newIndexPath row] + 1);
    [self performSelector:@selector(deselect:) withObject:nil afterDelay:0.5f];
}
```

当在视图之间移动时（通常作为表格选择的结果发生），延迟该方法，直到包含该表格的视图被完全重绘为止。在表格的父UIViewController的viewDidAppear方法中执行取消选择。

5.7 秘诀：创建复杂的单元格

表格单元并不仅限于实现带可选图像的文本标签。表格单元比这个更加灵活和复杂。但它们最终还是视图。同样，你可以采用自定义任何其他视图的方法（使用任意的框大小和孙子视图）来自定义它们。

秘诀5-5演示了如何创建一个具有子对象的自定义表格单元。在本例中，我修改了标准的UITableViewCell。单元构造函数（newCell:url:note:）创建一个320×100像素的单元格，并用几个具有独特文本特性的UILabel填充它（参见图5-5）。如图5-5所示，我的设计强项受到了限制。知道单元格可以是画布之后，具有出色开发图形技巧的开发人员将能够做大量工作。第6章将演示多个表格单元自定义的示例。

请注意，该示例并未使用继承自UITableViewCell的子类。它自定义标准单元格。还要注意，与标准的表格单元格不同的是，该演示并未将选择上的黑色字体转化为白色。如果你不喜欢标准的蓝黑显示，则必须使用你的单元格选择行为。这需要捕获选择，还原任何之前所选择的单元格的顏色，转换或更改新选择的单元格中标签的文本颜色。

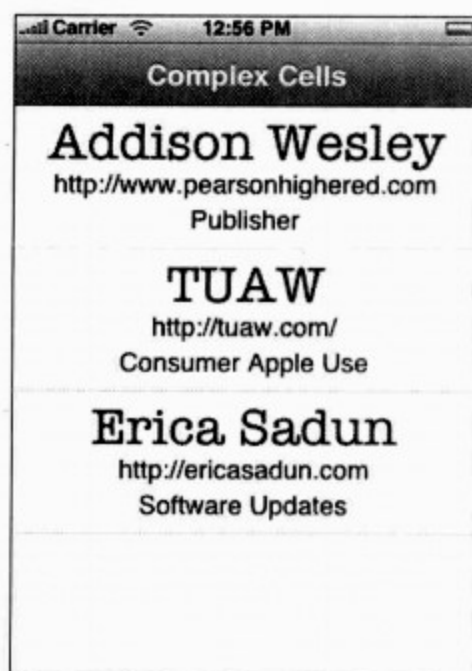


图5-5 通过向UITableViewCell中添加子视图来创建复杂的单元格类型

秘诀5-5 创建自定义的UITableViewCell

```
- (void) modCell:(UITableViewCell *)aCell withTitle:(NSString *)title
url: (NSString *) url note: (NSString *) comment
{
    // Title
    CGRect tRect1 = CGRectMake(0.0f, 5.0f, 320.0f, 40.0f);
    id title1 = [[UILabel alloc] initWithFrame:tRect1];
    [title1 setText:title];
    [title1 setTextAlignment:UITextAlignmentCenter];
    [title1 setFont: [UIFont fontWithName:@"American Typewriter" size:36.0f]];
    [title1 setBackgroundColor:[UIColor clearColor]];
}
```



```
// URL
CGRect tRect2 = CGRectMake(0.0f, 45.0f, 320.0f, 20.0f);
id title2 = [[UILabel alloc] initWithFrame:tRect2];
[title2 setText:url];
[title2 setTextAlignment:UITextAlignmentCenter];
[title2 setFont: [UIFont fontWithName:@"Helvetica" size:18.0f]];
[title2 setBackgroundColor:[UIColor clearColor]];

// Comment
CGRect tRect3 = CGRectMake(0.0f, 70.0f, 320.0f, 20.0f);
id title3 = [[UILabel alloc] initWithFrame:tRect3];
[title3 setText:comment];
[title3 setTextAlignment:UITextAlignmentCenter];
[title3 setFont: [UIFont fontWithName:@"Helvetica" size:18.0f]];
[title3 setBackgroundColor:[UIColor clearColor]];

// Add to cell
[aCell addSubview:title1];
[aCell addSubview:title2];
[aCell addSubview:title3];

[title1 release];
[title2 release];
[title3 release];
}

// Return a cell for the ith row
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
reuseIdentifier:@"any-cell"] autorelease];
}
// Set up the cell
int row = [indexPath row];
switch (row)
{
    case 0:
        [self modCell:cell withTitle:@"Addison Wesley"
        url: @"http://pearsonhighered.com/"
        note: @"Publisher Issues"];
        break;
    case 1:
        [self modCell:cell withTitle:@"TUAW"
        url: @"http://tuaw.com/"
        note: @"Consumer Apple Use"];
        break;
    case 2:
```



```

        [self modCell:cell withTitle:@"Erica Sadun"
          url: @"http://ericasadun.com"
          note: @"Software Updates"];
        break;
    }
    return cell;
}

```

5.8 秘诀：创建选中的选择

使用附件视图，你可以扩展常规的UITableViewController功能。最常用的附件是Delete按钮以及用于重新排序的拖动条，但是你还可以添加复选标记以创建交互式的“多选一”或“多选多”选择。借助这种选择，你可以让用户选取他们举行宴会所需的内容或者选择他们要更新的项。这种类型的单选按钮/复选框行为提供了丰富的表格交互。秘诀5-6演示了如何创建这类表格。

图5-6显示了实际的复选标记：一个具有附属单元格的标准UITableView。复选标记出现在所选项的旁边。轻击时，复选标记打开或关闭。通过单元格的迭代来查询你的表格。选中项目使用了UITableViewCellAccessoryCheckmark附件类型。



图5-6 复选标记附件提供一种从列表中进行“多选一”或“多选多”选择的简便方法

注意，要非常仔细地记录此处被选中的单元格，而不是与该单元格关联的逻辑项。当你滚动表格时，单元格会被重用。当执行该操作时，重用的单元格仍然保持选中状态或未选中状态。在下一章中，秘诀6-6会演示如何存储和更新与数据关联的状态信息。

你控制选择项的方式。若要简化切换项目的开关，则跟踪当前的accessoryType设置。若要创建“多选一”选择，则从旧索引路径中删除该附件，并将其添加到新索引路径中的单元格。

秘诀5-6 实现表格单元格选中

```
// Respond to user selection
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
➡*)indexPath
{
    printf("User selected row %d\n", [indexPath row] + 1);
    if ([[tableView cellForRowAtIndexPath:indexPath] accessoryType] ==
➡UITableViewCellAccessoryCheckmark)
        [[tableView cellForRowAtIndexPath:indexPath]
➡setAccessoryType:UITableViewCellAccessoryNone];
    else
        [[tableView cellForRowAtIndexPath:indexPath]
➡setAccessoryType:UITableViewCellAccessoryCheckmark];

    [self performSelector:@selector(deselect) withObject:nil afterDelay:0.5f];
}
```

5.9 秘诀：删除单元格

iPhone删除附件使得表格的管理更加简单和出色。每个iPhone用户都能迅速熟悉红色的小圆，这些小圆使他们能够从表格中删除单元格，并且许多用户选取基本的滑动-删除（swipe-to-delete）功能。秘诀5-7显示了如何构建有意义地响应单元格删除的表格。在本例中，用户可以通过轻击Add按钮来添加新的单元格，通过滑动或进入编辑模式并使用红色的删除控件（参见图5-7）来删除单元格。



图5-7 使用红色的删除控件，用户可以从交互式表格中交互地删除项目

秘诀5-7 动态删除单元格

```
// Initialize and set up the table title array
```

```
- (HelloController *) init
{
    if (![self = [super init]]) return self;

    self.title = @"Table Edits";
    tableTitles = [[NSMutableArray alloc] init];
    ithTitle = NCELLS;
    for (int i = 1; i <= NCELLS; i++)
        [tableTitles addObject:[NSString stringWithFormat:@"Table Cell #%d", i]];

    return self;
}

// Add a row to the table
- (void) add
{
    [tableTitles addObject:[NSString stringWithFormat:@"Table Cell #%d",
        ++ithTitle]];
    [self.tableView reloadData];
}

// Finished with edit mode, so restore the buttons and commit changes
- (void) leaveEditMode
{
    // Add the edit button
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
        initWithTitle:@"Edit"
        style:UIBarButtonItemStylePlain
        target:self
        action:@selector(enterEditMode)]
        autorelease];

    [self.tableView endUpdates];
    [self.tableView setEditing:NO animated:YES];
}

// Handle deletion requests
- (void) tableView:(UITableView *) tableView
    commitEditingStyle:(UITableViewCellEditingStyle) editingStyle
    forRowAtIndexPath:(NSIndexPath *) indexPath
{
    printf("About to delete item %d\n", [indexPath row]);
    [tableTitles removeObjectAtIndex:[indexPath row]];
    [tableView reloadData];
}

// Enter edit mode by changing "Edit" to "Done"
- (void) enterEditMode
{
    // Add the edit button
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
```

```

initWithTitle:@"Done"
style:UIBarButtonItemStylePlain
target:self
action:@selector(leaveEditMode)]
autorelease];

[self.tableView setEditing:YES animated:YES];
[self.tableView beginUpdates];
}

// Initialize the view with its buttons
- (void)loadView
{
    [super loadView];

    // Add an add button
    self.navigationItem.leftBarButtonItem = [[[UIBarButtonItem alloc]
                                              initWithTitle:@"New"
                                              style:UIBarButtonItemStylePlain
                                              target:self
                                              action:@selector(add)] autorelease];

    // Add the edit button
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                              initWithTitle:@"Edit"
                                              style:UIBarButtonItemStylePlain
                                              target:self
                                              action:@selector(enterEditMode)]
                                              autorelease];
}

```

删除控件属于UIRemoveControl类，它不是一个公用类。与在秘诀5-6中所讨论的复选标记一样，删除控件是一种附件视图。这些附件使你能够向UITableViewController中添加用户可以轻击的元素。附件类型包括此处的删除控件、添加控件、复选标记以及公开。

5.9.1 创建和显示删除控件

实际上，当向单元格中添加删除控件时，你需要采取两个步骤。创建单元格时，你必须指定它使用的附件视图种类（本例中为删除控件）。你通过将单元格的editingStyle指定为UITableViewCellEditingStyleDelete来执行该操作。在tableView:cellForRowAtIndexPath:数据源例程中实现该操作。

除了创建控件之外，你必须告诉表格进入编辑模式。通过两个调用来执行该操作。首先，使表格进入编辑模式，在每个单元格标题旁边显示编辑控件，并进行缩进以便为控件留出空间。为此，调用[tableView setEditing:YES animated:YES]。如果由于某些原因你没有进行此更改，则可以传递NO而不是YES，但是我劝你不要这样做。只要可能，将动画添加到你的iPhone用户界面，以将用户从一个状态带到下一个状态，这样他们就准备好在屏幕上进行模式更改了。

接下来，调用[tableView beginUpdates]。这将告诉你的表格进入一个模式，你能够交

互地更改它的内容。这意味着使用删除控件来删除单元格或使用插入控件添加单元格。（后者需要UITableVfiewCellEditingStyleInsert的EditingStyle。）

在秘诀5-7中，我将这些近来的操作组合到一个enterEditMode方法中。当用户轻击导航栏的Edit按钮时，将调用该方法。该方法还将标题从Edit转换为Done，并更新该按钮的目标操作设置以调用leaveEditMode。

5.9.2 关闭删除控件

当你完成编辑模式并且想返回正常表显示时，采用相反的顺序操作。使用[tableView endUpdates]终止表格的更新。然后关闭控件([tableView setEditing:NO animated:YES])。你可以随意再次进入编辑模式，而无需更新单元格属性或执行任何其他工作。为单元格设置了编辑样式之后，它便可以通过UITableView调用严格显示和隐藏它的删除控件。

5.9.3 处理删除请求

在删除行时，表格通过发出tableView: commitEditingStyle: forRowAtIndexPath:回调与你的应用程序进行通信。该方法允许你更新数据源并响应用户刚才的行删除操作。

下面是你从数据结构中删除项的实际位置，该数据结构提供数据源方法（在本秘诀中，通过一个由标题组成的NSMutableArray数组）并处理任何实际的操作，如删除因此而出现的文件。在此示例代码中，单元格消失，但对删除没有什么实际的后果。该示例不是以真实模型为根据的。标题列表只是丢失了有特殊编号的单元格标题。

5.9.4 滑动单元格

滑动提供一种从UITableView中删除项的出色方法。你无需执行任何操作就能够支持滑动以及设置单元格的EditingStyle。表格将处理其他事务。执行滑动时，用户需要从单元格的左侧轻轻拖到右侧。矩形的删除确认出现在单元格的右侧，但单元格不显示位于左侧的圆形删除控件。

用户滑动并确认之后，tableView: commitEditingStyle: forRowAtIndexPath:方法便处理数据更新，就像删除发生在编辑模式下一样。

5.9.5 添加单元格

秘诀5-7引入了一个Add按钮。该按钮用于添加新的表格单元。为完成该操作，add方法在tableTitles数组后面附加一个新的单元格标题，然后告诉表格使用reloadData更新数据源。因此，它实际上不创建新的单元格，这使得正常的表格机制能够使用这个更新后的数据源来检查数据并重新创建表格视图。

5.10 秘诀：对单元格重新排序

若使用户能够直接对表格的单元格重新排序，他们的能力将得到大大提升。他们可以应用该

交互操作按优先级对待处理项进行排序，或者选择应该位于播放列表第一个的歌曲。iPhone附带了内在的重新排序支持，可以轻松将该支持添加到应用程序中，如秘诀5-8所示。要将单元格的showsReorderControl:设置为YES。

若要正确地对单元格进行重新排序（如图5-8所示），除了设置showsReorderControl属性：tableView: moveRowAtIndexPath: toIndexPath:之外，还需要实现另一个方法。该方法将数据源与屏幕更改同步，和提交使用单元格删除的编辑一样。该数据源方法使你能够将对象从原始路径移动到更新后的路径。在本例中，这意味着将单元格标题移动到tableTitlesmutableArray中更新后的位置。

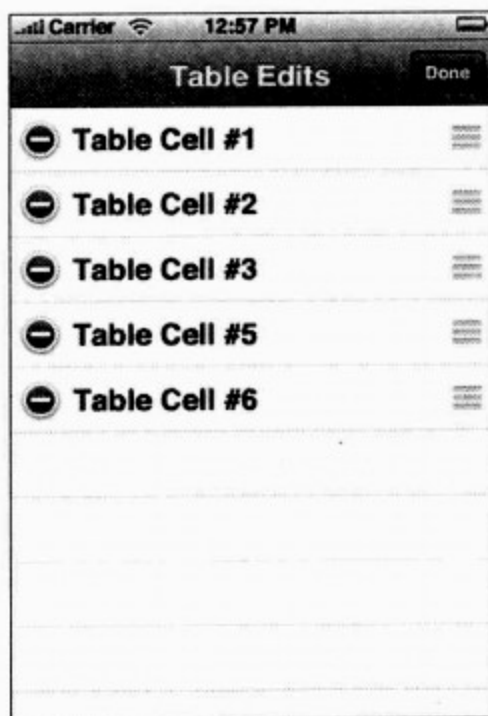


图5-8 在编辑模式期间对出现在每个单元格右侧的控件重新排序。它们显示为3个堆叠的灰色行

秘诀5-8 对表格单元格重新排序

```
// Perform the reorder
-(void) tableView: (UITableView *) tableView moveRowAtIndexPath: (NSIndexPath *)
    oldPath toIndexPath: (NSIndexPath *) newPath
{
    NSString *title = [tableTitles objectAtIndex:[oldPath row]];
    [tableTitles removeObjectAtIndex:[oldPath row]];
    [tableTitles insertObject:title atIndex:[newPath row]];
}
```

5.11 秘诀：使用公开

公开（disclosure）是指位于表格单元格右侧的蓝色或灰色面向右侧的V形小符号。公开使你能够从单元格进入到支持该单元格的视图。例如，在Contacts列表和Calendar应用程序中，这些V形小符号链接到某些屏幕，这些屏幕使你能够自定义联系人信息和约会。图5-9显示了两个UITable

示例，在这些示例中，每个单元格都显示一个公开控件。



图5-9 指向右侧的V形小符号表示公开控件，它们允许将各个表格项链接到其他视图

蓝色和灰色的V形小符号有两个作用。蓝色的UITableViewControllerAccessoryDetailDisclosureButton版本实际上是按钮。它们响应触摸，并且应该表明该按钮导致一个完全交互式的详细视图。灰色的UITableViewControllerAccessoryDisclosureIndicator并不跟踪触摸，并且应该将用户带到进一步选项的视图，主要是有关该选择的选项。

你可以在Settings应用程序中看到这两个附件的实际应用。在Wi-Fi Networks屏幕中，细节公开会进入有关每个WiFi网络的特定详细信息：它的IP地址、子网掩码、路由器、DNS等。“Other”的公开指示器使你能够通过向上滚动屏幕以便输入网络信息来添加新的网络。然后新网络显示自己的详细公开。

当一个屏幕带你到一个相关的子菜单时，你还会看到公开指示器。当使用子菜单时，严格执行简单的灰色V形小符号。经验是这样的：子菜单使用灰色的V形小符号，对象自定义使用蓝色的V形小符号。

秘诀5-9演示了如何在应用程序使用公开。该代码将每个单元格的accessoryType设置为UITableViewControllerAccessoryDetailDisclosureButton。重要的是，它还将hidesAccessoryWhenEditing设置为YES。当删除或重新排序控件出现时，公开V形小符号将会隐藏，从而使用户能够自由地编辑，而不会意外跑到新视图上。

为了处理用户在公开上的轻击，tableView: accessoryButtonTappedForRowWithIndexPath:方法使你能够检查轻击的行并且执行某些适当的响应。该示例只是推出了一个新的UIViewController，它显示应用程序的Default.png图像。在现实生活中，你已经移动到另一个视图，该视图解释有关所选择项的详细信息或者使你能够从附加选项中进行选择。

秘诀5-9 添加公开视图

```
// Return a cell for the ith row
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"any-cell"];
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithFrame:CGRectZero
            reuseIdentifier:@"any-cell"] autorelease];
    }
    // Set up the cell
    cell.text = [tableTitles objectAtIndex:indexPath.row];
    cell.editingStyle = UITableViewCellEditingStyleDelete;
    cell.accessoryType = UITableViewCellAccessoryDetailDisclosureButton;
    cell.hidesAccessoryWhenEditing = YES;
    return cell;
}

- (void)tableView:(UITableView *)tableView
accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath
{
    [[self navigationController] pushViewController:[[ImageController alloc] init]
        animated:YES];
}
```

5.12 小结

本章介绍了所有基本的iPhone表格特性：从简单的表格到编辑，再到重新排序。本章中介绍的这些技巧应该能让你为iPhone和iPod touch构建很多基于基本表格的应用程序。下面是本章的一些关键点。

- ❑ 当谈到对表格的理解时，你应该提前知道数据源和委托方法之间的差别。数据源用有意义的单元格填充表格。委托方法响应用户交互操作。
- ❑ 不仅对表格编程而言，而且对所有UI元素编程而言，通知是非常有用的一部分，不要忽略通知。借助这些表格，通知有时可以使你突破某些SDK的内在限制。
- ❑ UITableViewController简化了围绕中央的UITableView构建的应用程序。但是，如果你的应用程序需要，请不要犹豫，直接使用UITableView实例。只需确保明确支持UITableViewDelegate和UITableViewDataSource协议。
- ❑ 编辑或删除单元格不止一种方法。如果你宁可引入“edit/done”模式开关，则可以通过设置单元格的editingStyle来支持滑动-删除。

iPhone表格并不仅限于简单的滚动列表。你可以构建由各部分构成的表格，每个部分都有自己的标题，其中具有多个滚动列以及更多内容。你可以添加开关等控件，创建半透明的单元格背景，并包含自定义字体。你可以显示一个快速访问的索引，如在Address Book中使用的索引，或者构建Settings中那样的首选项屏幕。本章以第5章为基础，介绍在你的iPhone程序中使用的高级表格秘诀。通过探究这些秘诀，你将了解设计和实现更复杂表格所需执行的操作。

6.1 秘诀：对表格选择进行分组

在iPhone上，表格分为两种格式：分组的表格和普通表格列表。你已经看到了后者。第5章已经侧重介绍了如何创建它们。它们使用一个部分，而没有对内在的元素进行分组。但是你确实也已经看到了前者。iPhone上的Settings应用程序提供了实际的分组列表。这些列表显示在蓝灰色的背景上，并且每个子部分都出现在略微有些圆角的矩形中。图6-1显示了由秘诀6-1构建的分组列表。



图6-1 分组列表使你能够创建在蓝灰背景上滚动的单元格块

秘诀6-1 创建分组表格

```

@interface HelloController : UITableViewController
{
    NSMutableArray *sectionArray;
    int fullCount;
}
@end

@implementation HelloController
#define ALPHA @"ABCDEFGHIJKLMNOPQRSTUVWXYZ"

// Initialize the table view controller with the grouped style
- (HelloController *) init
{
    if (self = [super initWithStyle:UITableViewStyleGrouped]) self.title =
        @"Crayon Colors";
    return self;
}

// One section for each alphabet member
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return [sectionArray count];
}

// Each row array object contains the members for that section
- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section
{
    return [[sectionArray objectAtIndex:section] count];
}

// Convert a 6-character hex color to a UIColor object
- (UIColor *) getColor: (NSString *) hexColor
{
    unsigned int red, green, blue;
    NSRange range;
    range.length = 2;

    range.location = 0;
    [[NSScanner scannerWithString:[hexColor substringWithRange:range]]
     scanHexInt:&red];
    range.location = 2;
    [[NSScanner scannerWithString:[hexColor substringWithRange:range]]
     scanHexInt:&green];
    range.location = 4;
    [[NSScanner scannerWithString:[hexColor substringWithRange:range]]
     scanHexInt:&blue];
}

```

```

        return [UIColor colorWithRed:(float)(red/255.0f) green:(float)(green/255.0f)
        ↪blue:(float)(blue/255.0f) alpha:1.0f];
    }

    // Return a cell for the row and section
    - (UITableViewCell *)tableView:(UITableView *)tableView
    ↪cellForRowAtIndexPath:(NSIndexPath *)indexPath
    {
        NSInteger row = [indexPath row];
        NSInteger section = [indexPath section];

        // Create a cell if one is not already available
        UITableViewCell *cell = [self.tableView
        ↪dequeueReusableCellWithIdentifier:@"any-cell"];
        if (cell == nil)
            cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
            ↪reuseIdentifier:@"any-cell"] autorelease];

        // Set up the cell by coloring its text
        NSArray *crayon = [[[sectionArray objectAtIndex:section] objectAtIndex:row]
        ↪componentsSeparatedByString:@"#"];
        cell.text = [crayon objectAtIndex:0];
        cell.textColor = [self getColor:[crayon objectAtIndex:1]];
        return cell;
    }

    // Remove the current table row selection
    - (void) deselect
    {
        [self.tableView deselectRowAtIndexPath:[self.tableView
        ↪indexPathForSelectedRow] animated:YES];
    }

    // Respond to user selection by coloring the navigation bar
    - (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:
    ↪(NSIndexPath *)newIndexPath
    {
        // Retrieve named color
        int row = [newIndexPath row];
        int section = [newIndexPath section];
        NSArray *crayon = [[[sectionArray objectAtIndex:section] objectAtIndex:row]
        ↪componentsSeparatedByString:@"#"];

        // Update the nav bar color
        self.navigationController.navigationBar.tintColor = [self getColor:[crayon
        ↪objectAtIndex:1]];

        // Deselect
        [self performSelector:@selector(deselect) withObject:NULL afterDelay:0.5];
    }

```

```

// Build a section/row list from the alphabetically ordered word list
- (void) createSectionList: (id) wordArray
{
    // Build an array with 26 sub-array sections
    sectionArray = [[NSMutableArray alloc] init];
    for (int i = 0; i < 26; i++) [sectionArray addObject:[NSMutableArray alloc]
        init]];

    // Add each word to its alphabetical section
    for (NSString *word in wordArray)
    {
        if ([word length] == 0) continue;

        // determine which letter starts the name
        NSRange range = [ALPHA rangeOfString:[word substringToIndex:1]
            uppercaseString]];

        // Add the name to the proper array
        [[sectionArray objectAtIndex:range.location] addObject:word];
    }
}

// Prepare the Table View
- (void)loadView
{
    [super loadView];

    // Retrieve the text and colors from file
    NSString *pathname = [[NSBundle mainBundle] pathForResource:@"crayons"
        ofType:@"txt" inDirectory:@""];
    NSString *wordstring = [NSString stringWithContentsOfFile:pathname];
    NSArray *wordArray = [wordstring componentsSeparatedByString:@"\n"];

    // Build the sorted section array
    [self createSectionList:wordArray];
}

// Clean up
- (void) dealloc
{
    [sectionArray release];
    [super dealloc];
}

@end

```

分组列表需要正确地设置表格的样式。该操作在对象初始化期间完成。但之后无法返回原样式，并且无法将表格的样式从一种改为另一种，因此你必须三思而后行。决定要构建的列表类型，然后按照如下方式将其添加到视图控制器的init或loadView例程中。

借助列表，你可以操作表格视图控制器（如本例），或直接操作UITableView实例。对于表

格视图控制器，使用initWithStyle:将表格的样式设置为UITableViewStyleGrouped。对于UITableView实例，使用initWithFrame: style:。目前，SDK仅支持上面提到的这两种样式：分组样式和普通样式UITableViewStylePlain（默认设置）。

说明 苹果公司似乎正在从“普通”术语向“索引化”转移，因此这些名称可能会发生更改，恕不另行通知。

样式初始化是可选的。如果你不使用默认设置，只使用init或initWithFrame:创建表格。这与使用普通样式初始化表格的结果相同。

6.1.1 构建基于部分的数据源

在使用分组和部分时，要从两方面进行考虑。使用部分数组，可以存储和访问每个部分的成员。秘诀6-1通过创建一系列数组实现了该操作。部分数组为每个部分存储一个数组，该数组包含标题，在本例中包含每个单元格的顏色。

本秘诀通过createSectionList方法实现了此目的。该方法获取一个排序的单词数组并通过它们进行迭代。当它进行迭代时，会根据该单词第一个字母的索引将每个名称添加到数组中。

要让它正常工作，此特定的实现依赖于两件事情：首先，单词已经进行排序，每个子部分都按照它们在数组中的顺序添加单词；其次，部分与单词匹配。以标点符号或数字开头的条目将会使此循环失败。你可以正常添加一个“other”部分以关注这些情况，但这个（简单的）示例忽略了这些情况。

尽管字母部分非常有用并且可能是最常用的分组，但是你可以使用你喜欢的任何种类的分组结构。例如，你可以按照部门对人分组，按等级对宝石分组，按日期对约会分组。无论你选择哪种分组，都有一系列数组提供与分为几个部分的表格最匹配的表格视图数据源。

分组表格除了需要标准的UITableView数据源例程之外，还需要两个关键的数据源方法，如下所示。

❑ **numberOfSectionsInTableView**。该方法指定出现在表格中的部分数量。这样便可以确定要显示的分组数量。使用部分数组时，此处建议返回该部分数组中的项目数量（即[mySectionArray count]）。

❑ **tableView:numberOfRowsInSection**。通过一个部分编号来调用该方法。然后指定出现在该部分中的行的数量。对于推荐的数据结构，只需返回第 n 个子数组上的项数：
[[mySectionArray objectAtIndex:sectionNumber] count]。

除了这些数据源方法之外，务必使用行和部分信息来查找所讨论的单元格数据。第5章中的秘诀使用了一个带行号索引的二维平面数组。本章中的秘诀必须查找部分数组，然后在该部分中查找索引中的行。

6.1.2 添加部分标题

我们只需执行很少的操作便可向分组表格中添加部分标题。代码清单6-1显示了可选的

tableView: titleForHeaderInSection:方法, 该方法给每个部分添加标题。它传递一个整数。反过来, 你需要向它提供一个标题。

代码清单6-1 向部分添加标题

```
#define ALPHA_ARRAY [NSArray arrayWithObjects: @"A", @"B", @"C", @"D", @"E", @"F",
➤ @"G", @"H", @"I", @"J", @"K", @"L", @"M", @"N", @"O", @"P", @"Q", @"R", @"S",
➤ @"T", @"U", @"V", @"W", @"X", @"Y", @"Z", nil]

// Add a title for each section
- (NSString *)tableView:(UITableView *)tableView
➤titleForHeaderInSection:(NSInteger)section
{
    return [NSString stringWithFormat:@"Crayon names starting with '%@'",
    ➤[ALPHA_ARRAY objectAtIndex:section]];
}
```

如图6-2(左图)所示, 当你提供这个(可选)方法时, 会自动出现标题。标题名称出现在每个组的上方, 位于蓝灰背景上。如图6-2(右图)所示, 这些标题显示你是否提供部分成员。你可能希望通过删除标题和部分数据来删除空白部分, 以避免此处出现的双重标题。

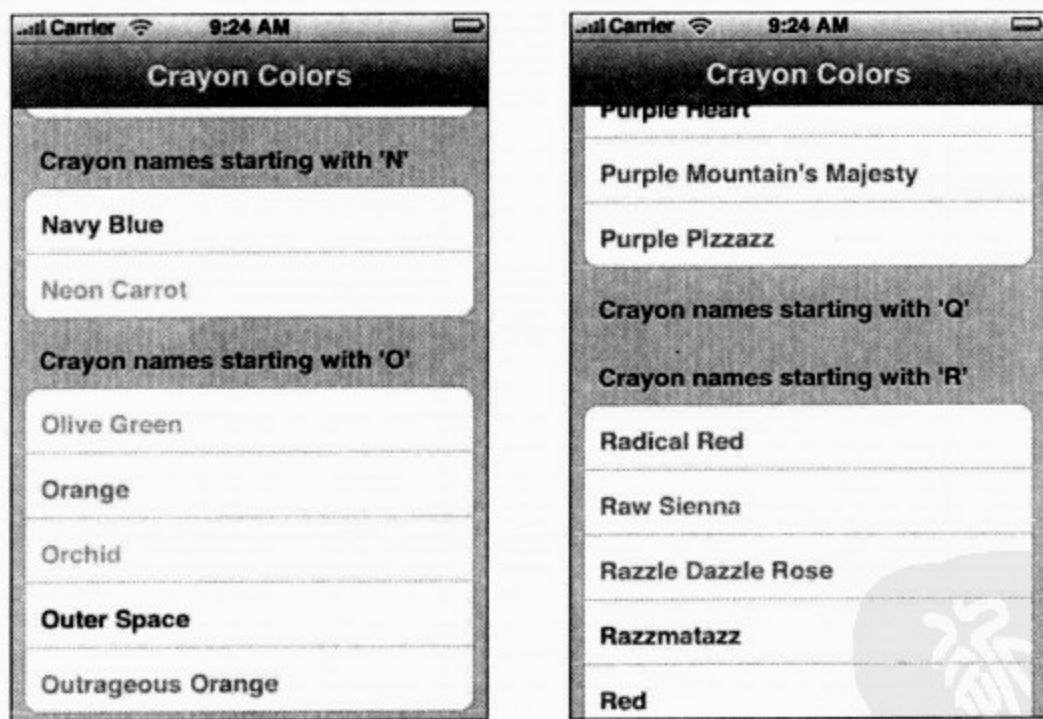


图6-2 (左图) 部分标题出现在每个部分的上方, 位于灰蓝背景上。
(右图) 当部分没有成员时, 标题会堆叠起来

6.2 秘诀: 构建带索引的部分表格

部分列表不需要分组。实际上, 如果你希望使用索引, 如图6-3(左)所示, 则应该避免使用分组列表。也就是说, 完全可以使用具有分组的索引。只是它看起来不太美观。

创建部分列表与创建分组表格的方式完全相同, 只是在初始化时使用普通表格样式, 并添加一个方法以提供索引标题数组。该数据源方法称为sectionIndexTitlesForTableView。使用

时, 如在秘诀6-2中, 它告诉表格为索引构建一个新的UIControl, 并将其添加到你的表格。

该方法返回一个字符串数组。对于秘诀6-2, 它是一个单一字符的列表, 每个字母一个字符, 但不要让它限制你的设计。如图6-3 (右图) 所示, 你可以返回单词和字符。尽管外观不美观, 但可能是你的设计所需要的。

秘诀6-2 构建带索引的部分列表

```
- (HelloController *) init
{
    if (self = [super initWithStyle:UITableViewStylePlain]) self.title = @"Crayon
    Colors";
    return self;
}

#define ALPHA_ARRAY [NSArray arrayWithObjects: @"A", @"B", @"C", @"D", @"E", @"F",
    @"G", @"H", @"I", @"J", @"K", @"L", @"M", @"N", @"O", @"P", @"Q", @"R", @"S",
    @"T", @"U", @"V", @"W", @"X", @"Y", @"Z", nil]

// Adding a section index here
- (NSArray *)sectionIndexTitlesForTableView:(UITableView *)tableView
{
    return ALPHA_ARRAY;
}
```



图6-3 (左图) 使用带索引的列表, 你可以通过轻击其标题直接跳到某些部分。
(右图) 索引不限于单个字符。如果需要, 可以使用整个单词来创建索引对象

6.3 秘诀: 定制单元格背景

标准的 UITableView 类包含带白色背景的单元格列表。你可以通过设置它的 backgroundColor 属性来更改表格的总体背景颜色。将其设置为某种颜色, 则单元格呈现该背

景颜色。将其设置为透明（clear），则表格背后的任何内容都可以看见。

这是苹果公司构建这些类的方法，但正因如此你的表格才能够可见。传播这种行为并不难，但却很重要。尽管不是特别直观或者简单，但是你可以通过进入表格中使用的UIView层次结构来定制你的表格、它们的单元格以及任何索引控件。本节中介绍如何在你自己的应用程序中构建这种效果。图6-4显示了最终结果：一个自定义的UITableView，每个单元格具有不同的颜色。

从此图中可以看出，该方法并不影响部分标题。当进行这些修改时，你可能希望从表格中忽略这些更改。（没有部分标题看起来很不错，在iPhone上运行更加平稳，但是由于秘诀6-3是直接从秘诀6-2构建的，因此考虑到连续性，我保留了它们。）

更改背景需要遵循3个基本步骤。

(1) 访问索引并清除其背景。

位于表格右侧的索引控件是表格视图的一个子视图。它的类为UITableViewIndex。若要定制，你可以从表格中单独设置其背景颜色。否则，它将继承表格使用的颜色。

若要允许背景透明，则可以使用[UIColor clearColor]。好在你只需执行该操作一次。但在loadView方法返回之后，才会将索引放置到表格中。因此，你必须延迟修改它。

在秘诀6-3中，在创建表格期间，我一直等待，直到调用tableView: cellForRowAtIndexPath:。然后我使用布尔变量clearedIndex确保该修改只运行一次。

若要执行修改，你需要查找实际的索引控件对象。通过tableView的子视图搜索索引对象之后，我将索引的背景颜色设置为透明。如果愿意，你还可以执行其他索引修改。例如，你可以根据需要将其字体更新为另一种样式。

(2) 为单元格创建自定义标签视图。

标准的UITableView单元格由一个单元格背景视图组成，其上添加了一个自定义UILabel。然而，即使你可以通过浏览子视图来访问这个标准的UILabel，但是当请求时，它并不会正确更新其特性。

你可以尝试清除其背景颜色，但是自定义行为会阻止完成该操作，从而使你在文本之后保留一种固定的颜色，该颜色与表格的背景颜色相匹配。你需要添加一个标准的UILabel并将该标签用于单元格标签。该方法不仅仅是一个替代的解决方案，而且它还具备很好的可靠性和一致性。

这个替代的解决方案需要在实际创建单元格时添加新的UILabel标签。这样便为它们提供了一个全新的UILabel，而不会出现与UITableViewCell的版本有关的任何行为问题。如果你正确初始化你的单元格，则它们将会在退出重用栈的队列时仍然保留这些标签。

秘诀6-3的工作类似于此。在创建单元格期间，它分配那些新标签并将其作为子视图添加到

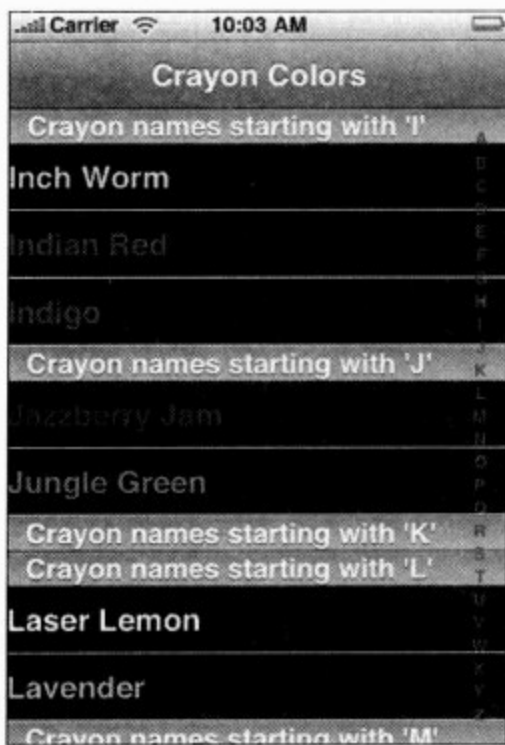


图6-4 为自定义的背景定制单元格和索引

单元格。然后，它对标签进行标记，以便可以检索它的设置文本。

这与标准的UITableViewCell结构稍有不同，其中标签通常是背景的子对象，而不是单元格本身的直接子对象。

(3) 标记单元格背景。

UITableViewCell的第一个子对象就是它的背景视图。秘诀6-3对该视图进行了标记，以便可以根据需要对其进行恢复和变色。若要给背景上色，请调整它的backgroundColor属性。若要添加自定义图像，则创建一个UIImageView子视图。

这3个步骤确保表格元素（即索引、单元格标题以及单元格背景）可以自定义。

秘诀6-3 创建自定义单元格背景

```
// Return a cell on demand
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // if the Index background has not yet been cleared, clear it
    if (!clearedIndex) {
        for (UIView *view in [tableView subviews])
            if ([[view class] description]
                isEqualToString:@"UITableViewIndex"]) {
                [view setBackgroundColor:[UIColor clearColor]];
            }
        clearedIndex = YES;
    }

    // Recover section and row info
    NSInteger row = [indexPath row];
    NSInteger section = [indexPath section];

    // Pull the cell
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"any-
    cell"];
    UILabel *labelView = NULL;

    // If there's a new cell needed, add a custom label
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
            reuseIdentifier:@"any-cell"] autorelease];
        cell.selectionStyle = UITableViewCellSelectionStyleGray;

        // Tag the background view
        [[[cell subviews] objectAtIndex:0] setTag:111];
        // Add and tag the label view
        labelView = [[UILabel alloc] initWithFrame: CGRectMake(0, 0, 300, 44)];
        [labelView setBackgroundColor:[UIColor clearColor]];
        [labelView setShadowColor:[UIColor whiteColor]];
        [labelView setFont:[UIFont boldSystemFontOfSize:20]];
```

```

        [labelView setTag:222];
        [cell addSubview:labelView];
        [labelView release];
    }

    // Set up the cell
    NSArray *crayon = [[[sectionArray objectAtIndex:section] objectAtIndex:row]
        ➤componentsSeparatedByString:@"#"];

    // Recover the label
    UILabel *labelView = (UILabel *)[cell viewWithTag:222];
    [labelView setText:[crayon objectAtIndex:0]];

    // recover the background
    UIView *bgView = [cell viewWithTag:111];
    [bgView setBackgroundColor:[self getColor:[crayon objectAtIndex:1]]];

    return cell;
}

```

定制表格视图

通过使用标准的UIViewController并对表格视图进行解耦可以进一步定制你的表格。图6-5显示了一个位于UIImageView中间的半透明表格。代码清单6-2显示了表格的创建。它构建了一个小的框架，设置的alpha级别刚好为0.35，并且已作为子视图添加到背景中。这种类型的修改允许你摆脱“我的应用程序就是表格，我的表格就是应用程序”思想的束缚，可以更加灵活地使用滚动表格，而且成本是可以计算的。确保在iPhone上进行测试以捕获自定义引发的任何性能问题。

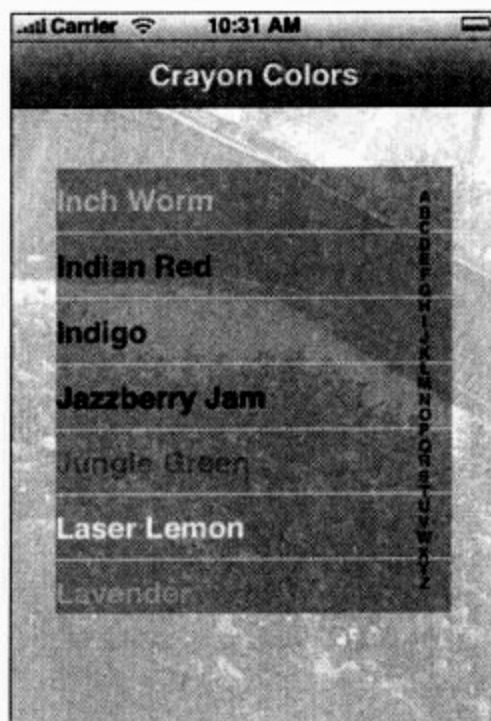


图6-5 表格可以使用半透明，如这个高度自定义的表格视图所示

代码清单6-2 创建一个浮动的半透明表格

```

- (void)loadView
{
    // Add a background
    CGRect frame = [[UIScreen mainScreen] applicationFrame];
    self.view = [[[UIImageView alloc] initWithImage:[UIImage
    imageNamed:@"bg.png"]] autorelease];
    self.view.userInteractionEnabled = YES;
    [self.view setFrame:frame];

    // Set up the table -- allowing for customizations
    frame = CGRectMake(30.0f, 40.0f, 260.0f, 300.0f);
    tableView = [[UITableView alloc] initWithFrame:frame style:
    UITableViewStylePlain];
    [tableView setDelegate:self];
    [tableView setDataSource:self];
    [tableView setBackgroundColor:[UIColor colorWithRed:1.0f green:1.0f
    blue:1.0f alpha:0.35f]];
    [self.view addSubview:tableView];
    [tableView release];

    // Retrieve the text and colors from file
    NSString *pathname = [[NSBundle mainBundle] pathForResource:@"crayons"
    ofType:@"txt" inDirectory:@""];
    NSString *wordstring = [NSString stringWithContentsOfFile:pathname];
    NSArray *wordArray = [wordstring componentsSeparatedByString:@"\n"];

    // Build the sorted section array
    [self createSectionList:wordArray];

    clearedIndex = NO;
}

```

6.4 秘诀：创建蓝白交替的单元格

尽管蓝白单元格交替是常见的且非常需要的表格特性，但是苹果公司在其iPhone SDK中并未包含这种样式。最后几个秘诀中所示的自定义单元格技术使你能够构建白/蓝交替的单元格结构，如图6-6所示。秘诀6-4提供了创建单元格的方法。

与之前的秘诀一样，该代码使用一个自定义的UILabel将文本混合在使用的背景上。简单的奇/偶检查($\text{row} \% 2$)指定是用蓝色还是白色填充背景。由于这个表格只使用一个部分，因此它大大简化了数学计算。蓝/白交替单元格最适合视觉和程序上非分组、非部分的表格。



图6-6 使用自定义背景创建蓝白交替的单元格

秘诀6-4 创建蓝白交替的单元格

```

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSInteger row = [indexPath row];

    // Pull the cell
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"any-cell"];
    UILabel *labelView = NULL;

    // If there's a new cell needed, add a custom label
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
➤reuseIdentifier:@"any-cell"] autorelease];
        cell.selectionStyle = UITableViewCellSelectionStyleGray;
        [[[cell subviews] objectAtIndex:0] setTag:111];

        labelView = [[UILabel alloc] initWithFrame: CGRectMake(8, 0, 300, 44)];
        [labelView setBackgroundColor:[UIColor clearColor]];
        [labelView setTag:222];
        [labelView setFont:[UIFont boldSystemFontOfSize:20]];
        [cell addSubview:labelView];
        [labelView release];
    }

    UIView *cellView = [cell viewWithTag:111];
    if (row % 2)
        [cellView setBackgroundColor:[UIColor whiteColor]];
    else
        [cellView setBackgroundColor:[UIColor colorWithRed:0.90f green:0.95f
➤blue:1.0f alpha:1.0f]];

    // recover labelView from the cell
    [(UILabel *) [cell viewWithTag:222] setText:[[[sectionArray objectAtIndex:row]
➤componentsSeparatedByString:@"#"] objectAtIndex:0]];

    return cell;
}

```

6.5 秘诀：设置表格边框

带边框的表格是SDK中未提供的另一种常用的表格特性。与分组表格使用的圆角部分不同，带边框的表格使用户能够在圆角的边框中滚动单元格。图6-7显示了带边框的表格。此处，圆形边界保持静止。所有单元格移动都发生在该边界之内和之后。

若要使边框工作，你的表格必须是一个独立的视图。这会限制你将UITableView添加到UIViewController，而不是直接使用UITableViewController类。反过来，边框只是添加到UIImageView中的UIImage图片（参见图6-7，右图）。该视图直接添加在UITableView的顶部，

并且通过 `userInteractionEnabled = NO` 设置为非交互。这意味着所有用户的触摸都会重定向到下面的表格。覆盖提供图片，而不会干扰交互操作。

秘诀6-5显示了如何设置表格并使用图6-7中所示的图片添加其边框。调整表格的大小以便与边框开口相匹配，并且图片位于表格的顶部。

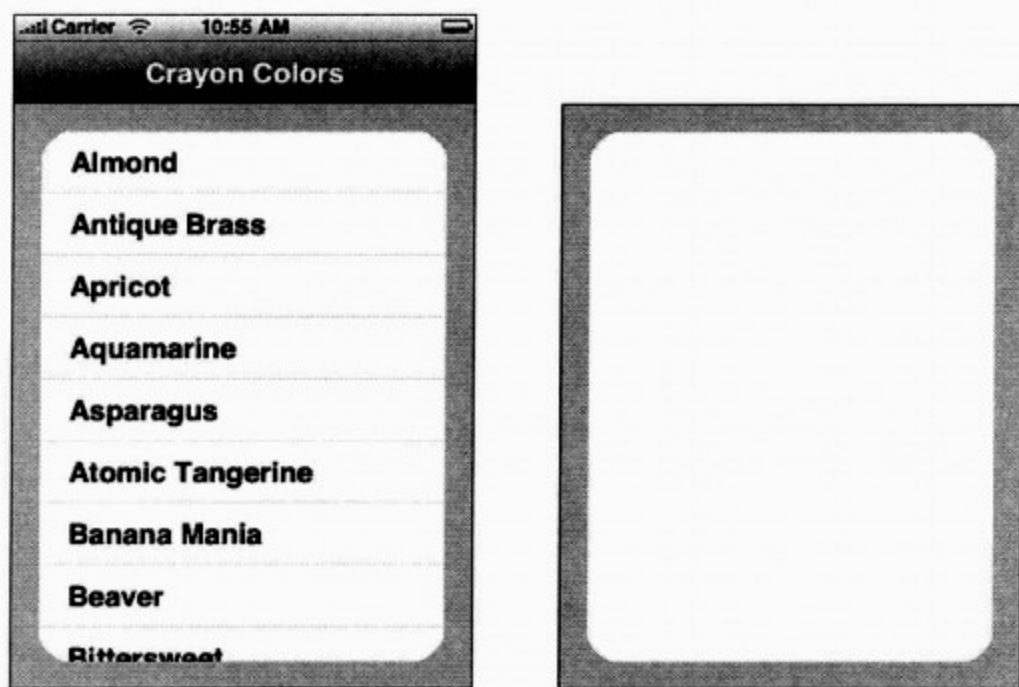


图6-7（左图）带边框的表格向选择中添加一个iPhone样式组件，该选择的目的是不是占满整个屏幕。（右图）是用来覆盖表格以产生所需边框的图片

秘诀6-5 设置列表视图的边框

```
- (void)loadView
{
    // Add a background for the main content view
    UIImageView *bg = [[[UIImageView alloc] initWithImage:[UIImage
➡imageNamed:@"BG.png"]] autorelease];
    bg.userInteractionEnabled = YES;
    self.view = bg;
    [bg release];

    // Build the table frame and add the table as a subview
    CGRect frame = CGRectMake(10.0f, 20.0f, 300.0f, 440.0f);
    tableView = [[UITableView alloc] initWithFrame:frame style:
➡UITableViewStylePlain];
    [tableView setDelegate:self];
    [tableView setDataSource:self];
    [bg addSubview:tableView];
    [tableView release];

    // Load the frame art, add it as a subview and set it
    // to be non-interactive
    UIImageView *fg = [[[UIImageView alloc] initWithImage:[UIImage
➡imageNamed:@"cutbg.png"]] autorelease];
```

```

    fg.userInteractionEnabled = NO;
    [bg addSubview:fg];
    [fg release];
}

```

6.6 秘诀：添加耦合的单元格控件

你可以通过添加UIControl项目来增加单元格的交互性。图6-8显示了具有on/off开关的表格。根据两个方面来额外添加这种单元格。首先，你需要仔细管理单元格重用。其次，你必须扩展数据源实现以包括控件的可寻址的状态信息。



图6-8 使用单元格重用创建具有自定义控件的高效利用内存的表格

和上几个秘诀中使用的自定义标签一样，秘诀6-6使用了一个单元格队列。它添加控件作为新单元格创建，并从取消队列的单元格中恢复单元格。这样便将所使用的实际控件项目数限制为每个程序最多10个。

关于状态信息，它保存在一个简单的易变字典 (boolDict) 中，并且通过单元格名称进行索引。使用时向字典中添加单元格，以便字典仍然稀疏，并且在指定之前，认为控件处于“off”状态。在可重用的单元格队列和稀疏字典之间，该秘诀通过少量的内存占用产生丰富的可视效果。

秘诀6-6 构建具有自定义控件的表格单元格

```

// respond to user switches by updating the Boolean dictionary
- (void) switchEm: (UIControl *) sender
{
    UITableViewCell *cell = (UITableViewCell *)[sender superview];
    NSString *whichCell = [cell text];

```

```

NSNumber *value = [boolDict objectForKey:whichCell];
if (!value) // no value assigned, false
{
    [boolDict setObject:[NSNumber numberWithBool:YES] forKey:whichCell];
} else {
    value = [NSNumber numberWithBool:![value boolValue]]; // toggle
    [boolDict setObject:value forKey:whichCell];
}
}

// Return a cell on demand
- (UITableViewCell *)tableView:(UITableView *)tableView
  cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // Recover section and row info
    NSInteger row = [indexPath row];
    NSInteger section = [indexPath section];
    // Pull the cell
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"any-
    cell"];
    UISwitch *switchView = NULL;

    // If there's a new cell needed, add a custom switch
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
        reuseIdentifier:@"any-cell"] autorelease];
        cell.selectionStyle = UITableViewCellSelectionStyleGray;
        cell.indentationLevel = 5;

        switchView = [[UISwitch alloc] initWithFrame: CGRectMake(4.0f, 16.0f,
        100.0f, 28.0f)];
        [switchView setTag:997];
        [cell addSubview:switchView];
        [switchView release];
    }

    // recover switchView from the cell
    switchView = [cell viewWithTag:997];
    [switchView addTarget:self action:@selector(switchEm:)
    forControlEvents:UIControlEventValueChanged];

    // Set up the cell
    NSArray *crayon = [[[sectionArray objectAtIndex:section] objectAtIndex:row]
    componentsSeparatedByString:@"#"];
    [cell setText:[crayon objectAtIndex:0]];
    // Retrieve the switch state and apply it
    NSNumber *value = [boolDict objectForKey:[crayon objectAtIndex:0]];
    if (!value) value = [NSNumber numberWithBool:NO];
    [switchView setOn:[value boolValue] animated:NO];

```

```

    return cell;
}

```

6.7 秘诀：构建多滚轮表格

有时你希望用户在多个列表中进行选择。这就是UIPickerView真正出色的地方。UIPickerView产生单独提供滚轮的表格，如图6-9所示。用户与每个滚轮进行交互以建立其选择。

尽管从表面上看这些表格与标准的UITableView非常类似，但是它们却使用不同的数据和委托协议。具体而言，在编写时，苹果公司根本未指定数据源协议。所有选取器视图支持都通过UIPickerViewDelegate协议进行处理，包括通常与数据源关联的调用。下面是需要记住的有关选取器视图与表格的区别。

❑ 没有UIPickerViewController。

UIPickerView实例充当其他视图的子视图。它们并不是应用程序视图的中心焦点。

❑ 选取器视图使用数字，而不使用对象。

组件，也就是所说的滚轮，通过数字进行索引，而不是通过NSIndexPath实例进行索引。总之，它是一个比UITableView更加正式的类。

❑ 选取器的视图高度是静态的。

你可以通过操作边框来调整标准UITableView的大小，但你不能采用同样的方式调整选取器的大小。所有选取器都是216像素高，你可以通过将视图与屏幕的底部对齐来测试该数字。任何其他边框高度看起来都会产生扭曲或被省略。

创建 UIPickerView

只要高度是216像素，宽度是320像素，你就可以对UIPickerView使用任何边框大小。也就是说，你可以将表格浮动到屏幕上任何所需的位置。并且你将必须进行浮动。选取器并未正确响应iPhone方向调整，因此你需要使其居中，并将其保留在屏幕上以防用户旋转其设备。

创建选取器时，有两个关键点需要记住。首先，你要启用选择指示器。这是一个浮动在已选项目上的蓝色栏。因此将showsSelectionIndicator设置为YES。其次，不要忘记指定委托。没有委托支持，你无法向视图添加数据或定义其特性。

为UIPickerView实现3个主要的数据源方法以使其正常工作。这些方法如下所示，你的应用程序可用来控制列数和组成这些列的行数。

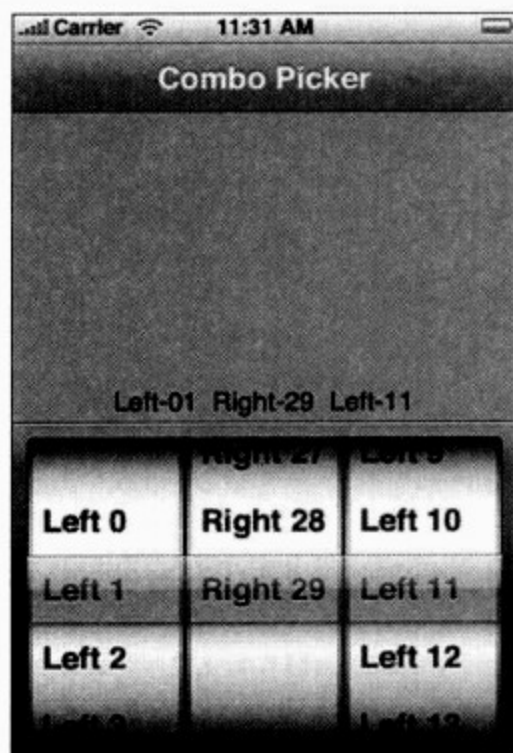


图6-9 UIPickerView实例使用户能够从独立的滚轮中进行选择

- ❑ **numberOfComponentsInPickerView**。返回一个整数, 即列数。
- ❑ **pickerView: numberOfRowsInComponent**。返回一个整数, 即每个滚轮的最大行数。这些数字不必相同。你可以让一个滚轮拥有很多行, 让另一个滚轮只拥有几行。
- ❑ **pickerView: titleForRow: forComponent**。该方法指定用来标记给定组件上的某一行的文本。它返回一个**NSString**。

除了这些数据源方法之外, 你还希望提供一个委托方法。该方法通过滚轮选择响应用户的交互操作。

- ❑ **pickerView: didSelectRow: inComponent**。向该方法中添加任何特定于应用程序的行为。如果需要, 你可以查询**pickerView**以对视图中的任何滚轮都返回**selectedRowInComponent**。

说明 文档中未记录的**setSoundsEnabled**: 控制用户能否听见与滚轮关联的单击效果。

秘诀6-7演示了如何使用**UIPickerView**实现多列选择。

秘诀6-7 使用**UIPickerView**实现多列选择

```

@interface HelloController : UIViewController <UIPickerViewDelegate>
{
    UIPickerView *pickerView;
    UILabel *comboBox;
}
@end

@implementation HelloController

- (HelloController *) init
{
    if (self = [super init]) self.title = @"Combo Picker";
    return self;
}

// Number of wheels
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView
➡ { return 3; }

// Number of rows per wheel
- (NSInteger)pickerView:(UIPickerView *)pView numberOfRowsInComponent:
➡ (NSInteger) component { return 30; }

// Return the title of each cell by row and component
- (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row
➡ forComponent:(NSInteger)component
{
    return [NSString stringWithFormat:@"%d", (component != 1) ? @"Left" :
    ➡ @"Right", row];
}

```

```

// Respond to user selection
- (void)pickerView:(UIPickerView *)aPickerView didSelectRow:(NSInteger)row
inComponent:(NSInteger)component
{
    printf("User selected row %d for component %d\n", row, component);

    int first = [pickerView selectedRowInComponent:0];
    int second = [pickerView selectedRowInComponent:1];
    int third = [pickerView selectedRowInComponent:2];

    NSString *results = [NSString stringWithFormat:@"Left-%%d Right-%%d
    Left-%%d",
                        (first < 10) ? @"0" : @"",
                        first,
                        (second < 10) ? @"0" : @"",
                        second,
                        (third < 10) ? @"0" : @"",
                        third];
    [comboBox setText:results];
}

- (void)loadView
{
    self.view = [[UIView alloc] initWithFrame:[UIScreen mainScreen]
    applicationFrame];
    self.view.backgroundColor = [UIColor lightGrayColor];

    // Add the picker
    float height = 216.0f;
    pickerView = [[UIPickerView alloc] initWithFrame:CGRectMake(0.0f,
    416.0f - height, 320.0f, height)];
    pickerView.delegate = self;
    pickerView.showsSelectionIndicator = YES;
    [self.view addSubview:pickerView];
    [pickerView release];

    // Add a results label
    comboBox = [[UILabel alloc] initWithFrame:CGRectMake(0.0f, 170.0f, 320.0f,
    32.0f)];
    [comboBox setBackgroundColor:[UIColor clearColor]];
    [comboBox setTextAlignment:UITextAlignmentCenter];
    [comboBox setText:@"Left-00 Right-00 Left-00"];
    [self.view addSubview:comboBox];
    [comboBox release];
}

- (void) dealloc
{

```

```

[pickerView release];
[comboBox release];
[super dealloc];
}
@end

```

6.8 秘诀: 使用 UIDatePicker

当你想让用户输入日期信息时, 苹果公司提供了一个恰当的UIPickerView的子类, 以处理多种类型的时间输入。图6-10显示了你可以从中进行选择的4种内置样式的UIDatePicker。范围从选择时间、选择日期到选择两者组合。

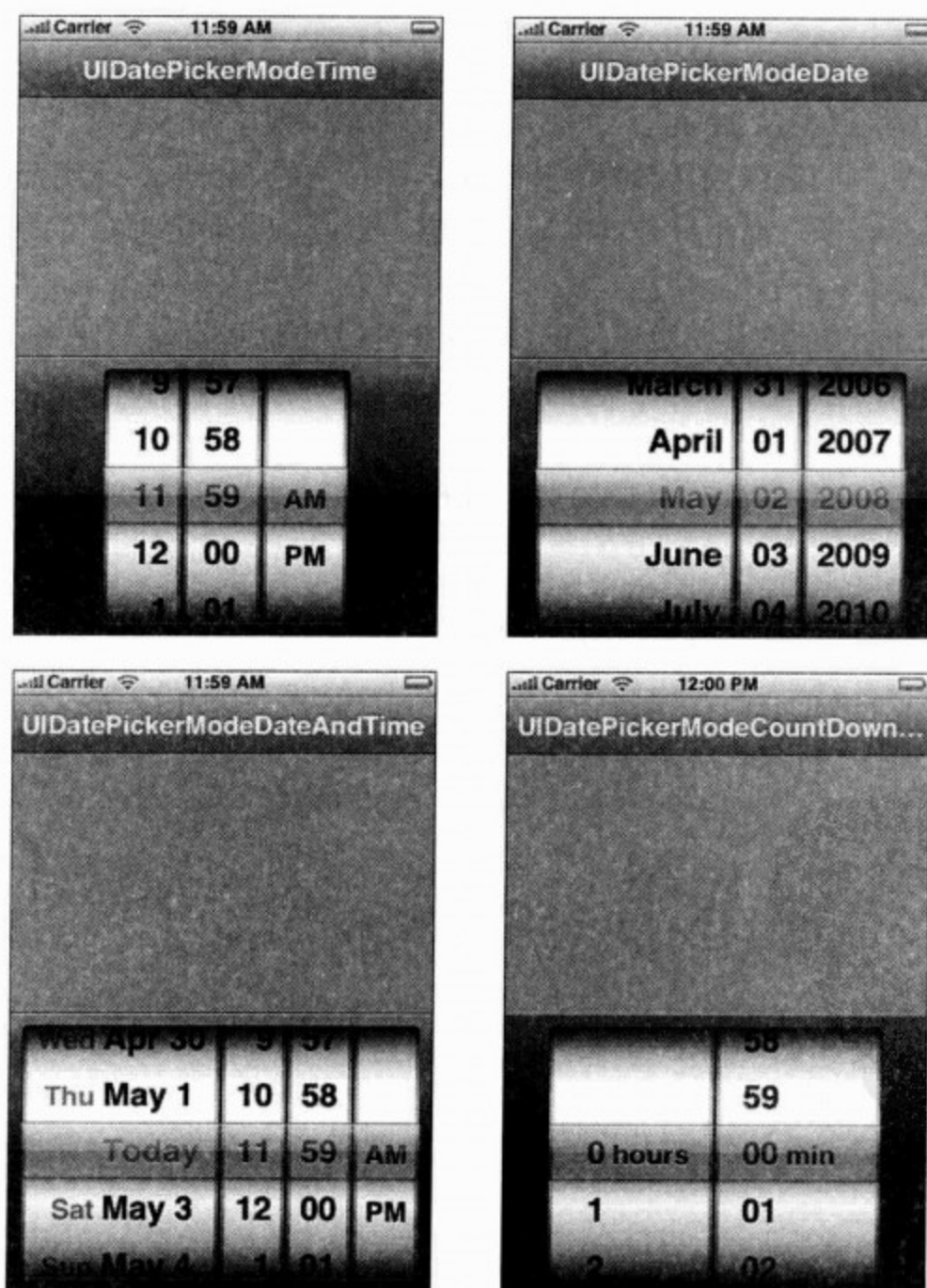


图6-10 iPhone提供了4种常备的日期选取器模型。使用datePickerMode属性可选择要在应用程序中使用的选取器

创建日期选取器

摆放日期选取器，使之与UIPickerView完全相同。创建一个320像素宽、216像素高的边框。之后，所有事情都变得更加容易。你不需要设置委托或定义数据源方法。但是，需要指定一个日期选取器模式（从UIDatePickerModeTime、UIDatePickerModeDate、UIDatePickerModeDateAndTime和UIDatePickerModeCountDownTimer中进行选择），然后当选择发生更改（UIControlEventValueChanged）时添加一个目标。

所有剩下的事情就是创建使用valuechanged事件触发的回调方法。完成之后，实现UIDatePicker对象就非常简单了，如秘诀6-8所示。

下面是你将在UIDatePicker类中利用的几个属性。

❑ date

设置日期属性以初始化选取器。

❑ maximumDate和minimumDate

这些属性为日期和时间选取设置边界。指定每一个标准的NSDate。借助这些，你可以限制用户从下一年中选取某个日期，而不只是输入日期，然后检查该日期是否在可接受的时间范围之内。

❑ minuteInterval

有时，你希望在你的选择上使用5、10、15或30分钟的时间间隔。使用minuteInterval属性指定该值。无论你传递的什么数字，它都必须均匀地分成60份。

❑ countDownDuration

使用该属性可为倒计时的计时器设置可用的最大值，这个设置可高达23小时59分钟（即，86 399秒）。

秘诀6-8 使用UIDatePicker选择日期和时间

```
// Respond to a user when the date has changed
- (void) changedDate: (UIDatePicker *) picker
{
    NSString *caldate = [[[picker date] dateWithCalendarFormat:@"%B %d, %Y"
    ➤timeZone:nil] description];
    [resultsView setText:caldate];
}

- (void)loadView
{
    self.view = [[UIView alloc] initWithFrame:[[UIScreen mainScreen]
    ➤ applicationFrame]];
    self.view.backgroundColor = [UIColor lightGrayColor];

    // Add the picker
    float height = 216.0f;
    pickerView = [[UIDatePicker alloc] initWithFrame:CGRectMake(0.0f, 416.0f -
    ➤height, 320.0f, height)];
    pickerView.datePickerMode = UIDatePickerModeDate;
```



```

[pickerView addTarget:self action:@selector(changedDate:)
➡forControlEvents:UIControlEventValueChanged];
[self.view addSubview:pickerView];
[pickerView release];

// Add a results label
resultsView = [[UILabel alloc] initWithFrame:CGRectMake(0.0f, 170.0f, 320.0f,
➡32.0f)];
[resultsView setBackgroundColor:[UIColor clearColor]];
[resultsView setTextAlignment:UITextAlignmentCenter];
[self.view addSubview:resultsView];
[resultsView release];
}

```

6.9 秘诀：创建完全自定义的分组表格

如果字母的部分列表表格是iPhone表格世界的M. C. Eschers，每个部分块都与列表中的其他部分精确地契合在一起，然后形式自由的分组表格就是Marc Chagalls。每个部分都是形式自由的手工艺品。

就本章而言，你掌握了此诀窍之后，创建之前的所有表格就会变得相当容易。但是美化分组表格（通常，iPhone爱好者称之为“首选表格”，因为这种类型的表格在Settings应用程序中得到了使用）仍然是幻想。构建分组表格就是进行拼贴，一片一片地手工制作外观。

像Interface Builder这样的工具用来将布局直接放置到开发人员的手中，而不是尝试通常编程来解决这些问题。然而，存在一个问题：Interface Builder至少此时没有创建首选表格。因此，由你决定通过编程将所有这些材料放到一起。仍然有标准化的方法，你可以用来构建这些表格。所有这些需要用到大量数据源方法以及很多很多的switch调用。

当你在控制之下获得基本知识之后，首选表格就变成了一个项目，你可以对其进行浇铸和塑形。图6-11显示了一个由两个组组成的简单的首选表格：一个包含4个项的清单和一个具有文本和文本输入字段的块。秘诀6-9演示了提供这样一个创建（creation）的所有这些工作。

不过，添加新项或更新旧项需要很多非常细的工作。该工作无法集中。你必须查看每个数据源方法，并用新的或精确的项进行更新。下一部分介绍这些数据源方法以及这些方法的作用。

创建分组首选表格

为首选表格安排一个新的UITableViewController不涉及任何特殊内容。你可以分配该表

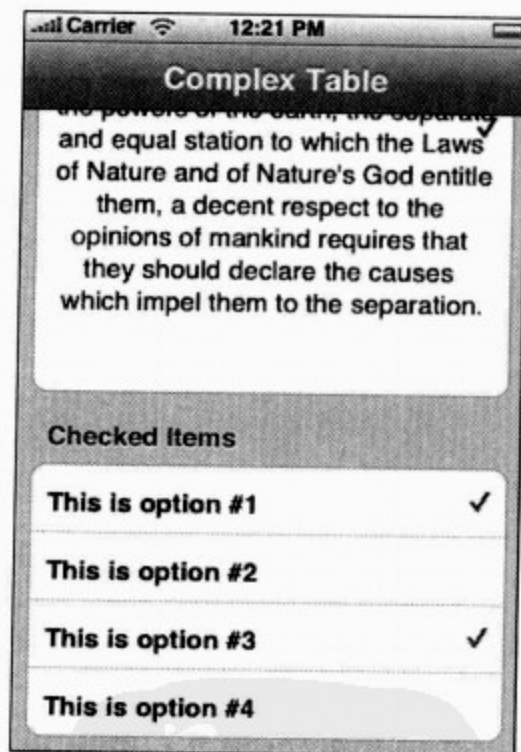


图6-11 首选表格必须手动布局，通过数据源方法指定每一行和组

格，并使用组合的表格样式对其进行初始化。这几乎已经接近了它的结尾。所有工作都已完成并且清扫干净。只是数据源和委托方法提出了挑战。下面是你需要定义的方法。

❑ `numberOfSectionsInTableView:`

所有首选表格都包含项分组。每组都包含在其中一个可爱的圆角矩形之中。以整数的形式返回你要定义的分组数。

❑ `tableView: titleForHeaderInSection:`

为该方法中的每个部分添加标题。返回一个具有请求的部分名称的NSString。

❑ `tableView: numberOfRowsInSection:`

每个部分都可以包含任意数量的单元格。让该方法返回一个表示该分组行数（即单元格）的整数。

❑ `tableView: heightForRowAtIndexPath:`

使用行高灵活的表格，计算强度更大。如果你需要使用可变高度（秘诀6-9就是这样），则实现该可选方法以指定这些高度。

❑ `tableView: cellForRowAtIndexPath:`

这是你在本章中看到的标准的cell-for-row方法。不同的是它的实现。秘诀6-9并未使用一种类型的单元格，而是为每种单元格类型构建了不同类型的可重用单元格（具有各种不同的重用标记）。如此秘诀所示，当使用多种单元格类型时，事情变得更加复杂。确保仔细管理你的重用队列。

❑ `tableView: didSelectRowAtIndexPath:`

根据所选择的单元格类型，在此委托方法中提供对单元格选择的不同反应。

秘诀6-9 简单的分组首选样式表格

```
// Number of groups
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 2;
}

// Section Titles
- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section
{
    switch (section)
    {
        case 0:
            return @"Civics";
        case 1:
            return @"Checked Items";
        default:
            return @"";
    }
}
```

```
}

// Number of rows per section
- (NSInteger)tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section
{
    switch (section)
    {
        case 0:
            return 2;
        case 1:
            return 4;
        default:
            return 0;
    }
}

// Heights per row
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    int section = [indexPath section];
    int row = [indexPath row];

    switch (section)
    {
        case 0:
            if (row == 0) return 80.0f;
            return 350.0f;
        case 1:
            return 44.0f;
        default:
            return 44.0f;
    }
}

// Produce cells
- (UITableViewCell *)tableView:(UITableView *)tableView
  cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSInteger row = [indexPath row];
    NSInteger section = [indexPath section];
    UITableViewCell *cell;

    switch (section)
    {
        case 0:
            if (row == 0) {
                // Add a text field to the cell
            }
        case 1:
            // ...
        default:
            // ...
    }
}
```

```

cell = [tableView dequeueReusableCellWithIdentifier:@"textCell"];
if (!cell) {
    cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
➤reuseIdentifier:@"textCell"] autorelease];
    UILabel *label = [[UILabel alloc]
➤initWithFrame:CGRectMake(20.0f, 10.0f, 100.0f, 20.0f)];
    [label setText:@"Name:"];
    [cell addSubview:label];
    [label release];
    [cell addSubview:[UITextField alloc]
➤initWithFrame:CGRectMake(20.0f, 40.0f, 280.0f, 30.0f)]];
}
UITextField *tf = [[cell subviews] lastObject];
tf.placeholder = @"Enter Your Name Here";
tf.delegate = self;
tf.borderStyle = UITextBorderStyleBezel;
return cell;
}
if (row == 1) {
    // Create a big word-wrapped UILabel
    cell = [tableView
➤dequeueReusableCellWithIdentifier:@"libertyCell"];
    if (!cell) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
➤reuseIdentifier:@"libertyCell"] autorelease];
        [cell addSubview:[UILabel alloc]
➤initWithFrame:CGRectMake(20.0f, 10.0f, 280.0f, 330.0f)]];
    }
    UILabel *sv = [[cell subviews] lastObject];
    sv.text = @"When in the Course of human events, it becomes
➤ necessary for one people to dissolve the political bands which have connected them
➤ with another, and to assume among the powers of the earth, the separate and equal
➤ station to which the Laws of Nature and of Nature's God entitle them, a decent
➤ respect to the opinions of mankind requires that they should declare the causes
➤ which impel them to the separation.";
    sv.textAlignment = UITextAlignmentCenter;
    sv.lineBreakMode = UILineBreakModeWordWrap;
    sv.numberOfLines = 9999;
    return cell;
}
break;
case 1:
    // Create cells with accessory checking
    cell = [tableView dequeueReusableCellWithIdentifier:@"checkCell"];
    if (!cell) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
➤reuseIdentifier:@"checkCell"] autorelease];
        cell.accessoryType = UITableViewCellAccessoryNone;
    }
    cell.text = [NSString stringWithFormat:@"This is option #%d",
➤row + 1];

```



```

        return cell;
        break;
    default:
        break;
    }

    // Return a generic cell if all else fails
    cell = [tableView dequeueReusableCellWithIdentifier:@"any-cell"];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
            reuseIdentifier:@"any-cell"] autorelease];
    }

    return cell;
}

- (void) notify: (NSString *) aMessage
{
    UIAlertView *baseAlert = [[UIAlertView alloc]
        initWithTitle:@"Alert" message:aMessage
        delegate:self cancelButtonTitle:nil
        otherButtonTitles:@"Okay", nil];

    [baseAlert show];
}

// TextField delegate handles return events and dismisses keyboard
- (BOOL)textFieldShouldReturn:(UITextField *)textField
{
    [textField resignFirstResponder];
    [self notify:[NSString stringWithFormat:@"Hello %@", [textField text]]];
    return YES;
}

// Respond to user selection based on the cell type
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)
newIndexPath
{
    int section = [newIndexPath section];
    int row = [newIndexPath row];
    UITableViewCell *cell = [self.tableView cellForRowAtIndexPath:newIndexPath];

    switch (section)
    {
    case 0:
        if (row == 0)
        {
            [self notify:[NSString stringWithFormat:@"Hello %@",
                [[[cell subviews] lastObject] text]]];
        }
    }
}

```

```
        case 1:
            if (cell.accessoryType == UITableViewCellAccessoryNone)
                cell.accessoryType = UITableViewCellAccessoryCheckmark;
            else
                cell.accessoryType = UITableViewCellAccessoryNone;
            break;
            break;
        default:
            break;
    }

    [self performSelector:@selector(deselect) withObject:NULL afterDelay:0.5];
}
```

6.10 小结

本章介绍了各种高级iPhone表格，包括选取器元素、带索引的字母列表和首选表格。本章涉及的技巧应该使你能够为iPhone和iPod touch构建丰富的基于表格的应用程序。下面列出了本章中的一些要点。

- ❑ 苹果公司设计UITableView来采用某种特定的方式工作。不要拘泥于苹果公司，不要将你自己局限于那些设计途径（它们也许不能满足你的需要）。本章介绍了访问单元格和表格视图以及根据需要管理这些视图的很多备选方法。
- ❑ 索引控件提供了一种通过大量排好序的列表快速导航的出色方法。在使用表格时利用它们的功能，否则它们会变得无法导航。
- ❑ 日期选取器非常专业，并且非常擅长于它的任务：请求用户输入日期和时间。选取器视图提供了一个不太专业的解决方案，需要你做更多工作以完善它们。
- ❑ 由于首选表格对于编程来说比较复杂并且令人烦恼，因此谈到iPhone编程时，首选表格是一项最需要的特性。你可以从第三方软件看到它们几乎无处不在。它们允许你将很多类型的交互式输入组合到一个外观漂亮且易于使用的滚动页面上。当你成功完成所有这些繁琐的工作之后，它们便成为你的编程库中一个功能强大的工具。

如你所愿，iPhone可以加载和显示多种格式的媒体。它可以加载和显示音乐、电影。还能处理图像和Web页面。还可以显示PDF文档和相册，以及更多格式的内容。在本章中，你将探索如何用多种方法将数据导入或下载到应用程序，并使用iPhone的多点触摸（multitouch）界面显示那些数据。本章介绍可以为用户呈现大量媒体类型的秘诀。你将了解如何构建这些数据类型的查看器，以及如何加载来自iPhone本身和远程位置的内容的数据类型。

7.1 秘诀：按照文件类型浏览 Documents 文件夹

Documents文件夹的作用正如它的名称所示。可以将文档保存到此文件夹中，并从中访问这些文档。苹果公司建议你存储由自己的程序创建的或者从自己的程序中浏览的文件数据。

Documents文件夹位于应用程序沙盒（sandbox）中。它是除应用程序束之外，在沙盒中自动创建的3个目录之一。其他两个是Library和tmp。Library为程序存储用户默认设置和其他状态信息。tmp文件夹是动态创建临时文件的地方。Documents和Library文件夹中的文件不是临时的。只要iPhone进行同步，iTunes就备份Documents和Library文件夹中的所有文件。当iPhone重启时，它将丢弃所有的tmp文件。

这里缺少的一个标准文件夹是Application Support文件夹。在Mac上，Application Support有特别的作用。它保存不包含于应用程序束中的文件、插件和支持数据，它们在应用程序运行时提供支持，不过并不是应用程序正常执行所必需的。在iPhone上，像这样的数据应该被存储在Library文件夹（更倾向于存储静态数据）或Documents文件夹（更倾向于存储可变数据，如用户会更改和更新的SQL数据）。

通常，处理媒体时，你会希望浏览文档文件夹，以查找用户创建或下载的文件。通过浏览此文件夹，可以生成一个用户可选择的可用文件列表。

秘诀7-1展示了如何创建一个表，并用来自Documents文件夹的文件名填充该表。按照路径扩展名选择这些名称。此代码收集了匹配到一个路径扩展名数组的所有文件。图7-1实际地显示了这个过程，列出了以txt、c、h或m结尾的文档文件。

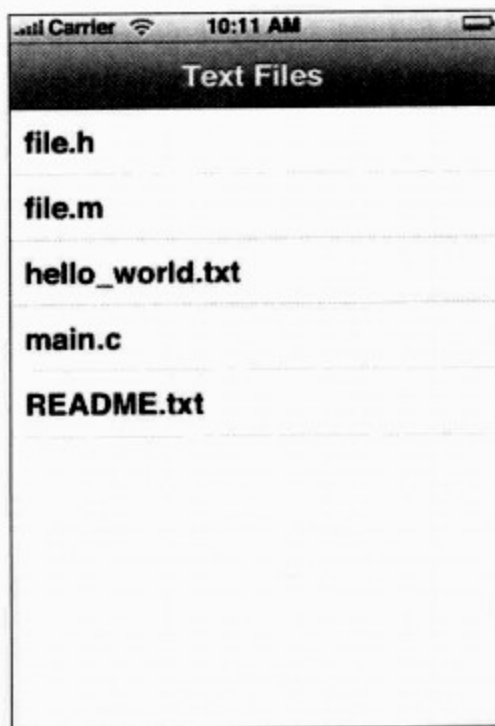


图7-1 通过在Documents文件夹中按照扩展名类型进行浏览，
可以为用户呈现一个匹配文件名的列表

查找 Documents

在Macintosh上，你可以使用标准文件位置和非标准文件位置。而iPhone及其沙盒则更具结构性——苹果公司规定必须这样做。其文件出现在更明确的地方。在Mac上，查找文档文件夹意味着搜索用户域。这是查找文档文件夹的标准方法。

```
NSArray *paths =
    ➤ NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
        NSUserDomainMask, YES);
return [paths lastObject];
```

iPhone更加受约束。可以通过调用主目录函数安全地定位到顶级沙盒文件夹。这允许你向下导航一级到Documents文件夹，理所当然，这会完全保证你达到正确的目的地：

```
#define DOCUMENTS_FOLDER [NSHomeDirectory()
    ➤ stringByAppendingPathComponent:@"Documents"]
```

找到Documents文件夹后，通过调用NSFileManager的directoryContentsAtPath:方法获取一个文件数组。NSArray的pathsMatchingExtensions:方法检索那些你想要使用的文件类型。秘诀7-1使用这些方法创建一个文本文件的文件列表，并使用这些文本文件作为数据源填充一个表。

秘诀7-1 构建一个用户可选择的Documents列表

```
#define DOCUMENTS_FOLDER [NSHomeDirectory()
    ➤ stringByAppendingPathComponent:@"Documents"]

- (HelloController *) init
{
    if (self = [super init]) self.title = @"Text Files";
```



```

    // initialize the file list with any text documents
    fileList = (NSMutableArray *) [[[NSFileManager defaultManager]
    ➤directoryContentsAtPath:DOCUMENTS_FOLDER] pathsMatchingExtensions:[NSArray
    ➤arrayWithObjects:@"txt", @"c", @"h", @"m", nil]] retain];
    return self;
}

- (void) loadView
{
    [super loadView];

    // Reveal the documents folder - so you can pre-fill it for the simulator
    printf("Documents folder is %s\n", [DOCUMENTS_FOLDER UTF8String]);

    self.tableView.autoresizesSubviews = YES;
    self.tableView.autoresizingMask = (UIViewAutoresizingFlexibleWidth |
    ➤UIViewAutoresizingFlexibleHeight);
}

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

- (NSInteger)tableView:(UITableView *)tableView
    ➤numberOfRowsInSection:(NSInteger)section
{
    return [fileList count];
}

// Return a cell for the ith row
- (UITableViewCell *)tableView:(UITableView *)tableView
    ➤cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"any-
    ➤cell"];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
        ➤reuseIdentifier:@"any-cell"] autorelease];
    }

    cell.text = [fileList objectAtIndex:indexPath.row];
    return cell;
}

```

7.2 加载和查看图像

处理图像时，并没有一个适合所有问题的解决方案。在下面的几节中，你会看到处理多种类

型的图像和图像源的秘诀。这些秘诀涉及本地显示和从万维网实时下载的图像。每个秘诀都为你认为可能是很简单的问题提供一个不同视角。

图像显示都是有关内存的操作。要区别对待小图像和大图像的显示。对于小图像，也就是小于半兆字节的图像，直接把它们加载到UIImageView并放到屏幕上。甚至可以把它们放置在滚动视图上（参见秘诀7-2），这样用户可以直接调整其大小和位置。

对于大图像，像UIWebView这样的类能够很容易地处理强烈依赖于内存的（memory-intense）数据（如秘诀7-3中所示）。WebView对象提供了一个不太令人满意的界面，因为对于PDF文件和Web页面来说，它是最佳选择，但是当涉及大型文件时，它就不起作用了。你可以获得你所需要的调整大小和显示的能力，不过最初的呈现是非常紧凑的。在编写此书时，苹果公司还没有提供“调整到屏幕大小”选项，不能以最佳的比例开始呈现UIWebView图像外观。

从机载相册中查看和选择图像时，仅通过UIImagePickerController类就足够了。它简化了在照相机相册和同步相片簿中浏览图像的过程，还简化了挑选图像、剪裁图像和添加新图片的过程。秘诀7-4到秘诀7-6展示了其功能范围。

你将看到，每一种秘诀都介绍了显示iPhone图像和与iPhone图像进行交互的不同方法。

说明 iPhone支持以下图像类型：PNG、JPG、THM、JPEG、TIF、TIFF、GIF、BMP、BMPF、ICO、CUR、XBM和PDF。

7.3 秘诀：显示小图像

显示小图像（大小小于半兆字节的图像）有些无关紧要。当从应用程序束中加载小图像时，就证实了这一点。使用UIImage imageNamed:方法加载带有应用程序图像内容的UIImageView。imageName:返回的结果被缓存，因此文件检索非常有效：

```
contentView = [[UIImageView alloc] initWithFrame:[UIScreen mainScreen]
    applicationFrame];
[contentView setImage:[UIImage imageNamed:@"helloworld.png"]];
```

你不会始终把图像存储在应用程序束中。要从其他位置加载图像，应使用imageWithContentsOfFile:方法，把到图像的路径传递给它。UIImage将处理其余工作，从文件中加载图像：

```
NSString *path = [DOCUMENTS_FOLDER
    stringByAppendingString:@"myImage.png"];
[contentView setImage:[UIImage imageWithContentsOfFile:path]];
```

还可以从正在运行的主机加载图像数据，而不是从文件中获取。秘诀7-2展示了如何初始化来自基于Web的URL内容的图像数据。在这里，图像数据是来自www.weather.com的最新气象图。此代码把该数据加载到UIImageView对象，并把该图像视图放到一个UIScrollView实例上。这就生成了图7-2所示的完全交互式的界面。用户可以滚动气象图，并放大和缩小。

在此秘诀中，滚动视图使其内容大小与播报图像大小（[img size]）相符合，最大和最小

比例分别是实际大小的两倍和一半。通过这些设置,用户能够与图像进行完全交互,不过该交互只限制在不会给iPhone内存增加负担的合理范围内。

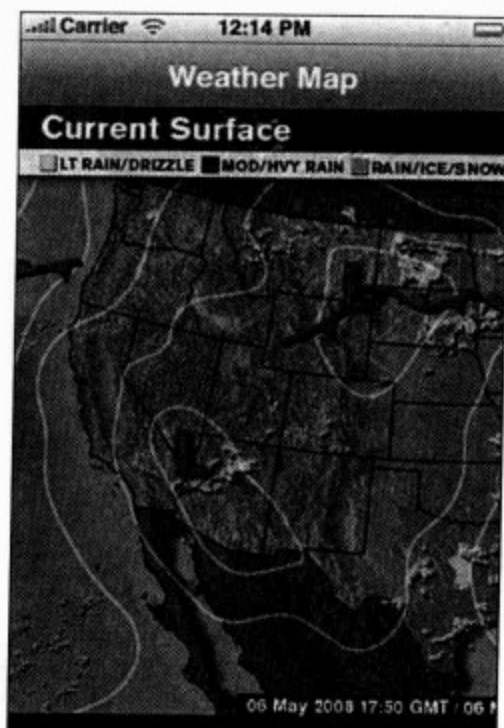


图7-2 这个气象图是从万维网URL上下载的,它被放置到滚动视图上,以使用户能够缩放和拖动图像

说明 UIImage View类可以显示单个图像或移动幻灯片。秘诀4-8的蝴蝶展示了如何制作一个图像显示动画。

秘诀7-2 下载图像数据,在UIScrollView上显示

```
@interface HelloController : UIViewController <UIScrollViewDelegate>
{
    UIScrollView *contentView;
    UIImageView *imgView;
}
@end

@implementation HelloController

- (id)init
{
    if (self = [super init]) self.title = @"Weather Map";
    return self;
}

// Return the UIImageView instance that gets zoomed by the scroller
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)scrollView
{

```

```

    return imageView;
}

- (void)scrollViewDidEndZooming:(UIScrollView *)scrollView withView:(UIView *)view
  atScale:(float)scale
{
    // perform any post-scaling tasks here like re-centering
}

- (void)loadView
{
    // Download the 600x405 weather map and load it into the image view
    NSURL *url = [NSURL URLWithString:@"http://image.weather.com/images/maps/
    ➡current/curwx_600x405.jpg"];
    UIImage *img = [UIImage imageWithData:[NSData dataWithContentsOfURL:url]];
    UIImageView *imageView = [[UIImageView alloc] initWithImage:img];
    [imageView setUserInteractionEnabled:NO];

    // Set up the main scroller
    contentView = [[UIScrollView alloc] initWithFrame: [[UIScreen mainScreen]
    ➡applicationFrame]];
    [contentView setScrollEnabled:YES];
    [contentView setContentSize:[img size]];
    [contentView addSubview:imageView];
    [imageView release];

    [contentView setMaximumZoomScale: 2.0f];
    [contentView setMinimumZoomScale: 0.5f];
    [contentView setDelegate:self];

    self.view = contentView;
    [contentView release]; // reduce retain count by one
    // Provide support for auto-rotation and resizing
    contentView.autoresizesSubviews = YES;
    contentView.autoresizingMask = (UIViewAutoresizingFlexibleWidth |
    ➡UIViewAutoresizingFlexibleHeight);
}

// Allow the view to respond to iPhone Orientation changes
- (BOOL)shouldAutorotateToInterfaceOrientation:
  (UIInterfaceOrientation)interfaceOrientation
{
    return YES;
}

- (void) dealloc
{
    [imageView release];
    [contentView release];
}

```



```

    [super dealloc] ;
}
@end

```

7.4 秘诀：使用 UIWebView 显示图像

随着图像大小的增加，可以丢掉滚动视图/图像视图方法，而使用UIWebView类。UIWebView类是一个奇迹。秘诀7-3展示了如何使用这个类加载和显示图像。它使用来自秘诀7-1的文档浏览器，被定制为拥有PNG、JPEG、TIFF和PDF图像选项。不需要任何程序就能支持这些文件类型，它们都是内置的，包括图7-3中所示的PDF翻页支持（左图）和HTML/CSS支持。

说明 把大的PDF数据文件加载到UIWebView实例中时，会触发applicationDidReceiveMemoryWarning:回调到应用程序委托。一般可以忽略这些警告，因为苹果公司提供的UIWebView类适当地减少了内存负载。如果有更多要求，可以使用工具配置应用程序。

前面提到过，其缺点是不能直接控制图像的大小。如图7-3（右图）所示，它以极高的分辨率（pixel per inch, ppi）显示图像。可以通过生成HTML源来设置图像宽度和高度属性，并生成所需的准确呈现。

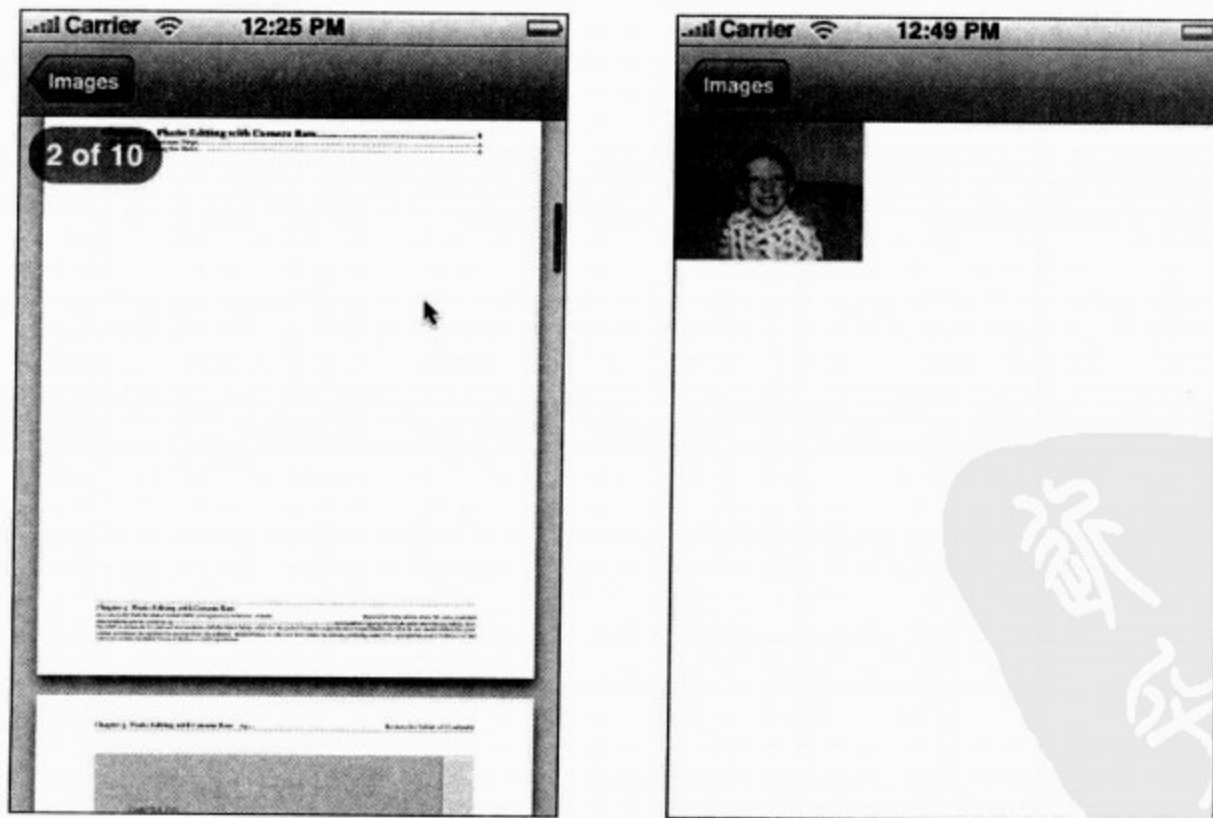


图7-3 （左图）UIWebView对象带有内置的PDF翻页支持，以及对所有iPhone支持图像类型的完全兼容。（右图）使用UIWebView时，默认的图片显示大小和理想大小差距甚远

秘诀7-3 使用UIWebView类显示图像

```

#define DOCUMENTS_FOLDER [NSHomeDirectory()
➡ stringByAppendingPathComponent:@"Documents"]

```

```
@interface WebController : UIViewController
{
    UIWebView *webView;
}
@end

@implementation WebController

- (id)initWithFile: filename
{
    // Initialize a URL pointing to the image file to display
    NSString *path = [DOCUMENTS_FOLDER stringByAppendingPathComponent:filename];
    NSURL *fileURL = [[NSURL alloc] initWithFileURLWithPath:path];
    NSURLRequest *req = [NSURLRequest requestWithURL:fileURL];
    // Initialize the Web view
    webView = [[UIWebView alloc] initWithFrame:[UIScreen mainScreen]
    applicationFrame];
    // allow user interaction
    [webView setScalesPageToFit:YES];

    // Load the Web view with the contents of the image file URL
    [webView loadRequest:req];
    return self;
}

- (void)loadView
{
    self.view = webView;
    [webView release];
}

- (void) dealloc
{
    [webView release];
    [super dealloc];
}
@end
```

使用 UIWebView 显示 Web 页面

要显示Web页面（而非图像），需要在秘诀7-3中做一些小的更改。这些更改如下所示。

- (1) 在主界面中，把扩展名转换为搜索HTML文件而非图像文件。
- (2) 使用代码清单7-1的内容代替WebController的initWithFile:方法。

这两个简单的替换操作提供了用于本地HTML源的一个具有丰富功能的Web浏览器。你能够连接到基于Web的资源，并通过点击链接进行导航。UIWebView在一个易于使用的类中提供了功能全面的Safari-esque浏览器。

代码清单7-1 把HTML源加载到UIWebView对象中

```

- (id)initWithFile: filename
{
    // Read in the HTML source
    NSString *fileText = [NSString stringWithContentsOfFile:filename
    ➤encoding:NSUTF8StringEncoding error:NULL];

    // Initialize the Web view
    webView = [[UIWebView alloc] initWithFrame:[[UIScreen mainScreen]
    ➤applicationFrame]];

    // Load it with the HTML source
    if (!fileText) [webView loadHTMLString: [error localizedDescription]
    ➤baseURL:fileURL]; else [webView loadHTMLString:fileText baseURL:fileURL];

    // Allow direct user interaction
    [webView setScalesPageToFit:YES];
    return self;
}

```

7.5 秘诀：浏览图像库

UIImagePickerController类是一个奇怪的类。它有一些相对少的公共方法和一些稍显古怪的方法。例如，它有自己的内置导航控制器。如果将它加入到一个现有的基于导航控制器的视图方案中，它会在第一个导航栏下面添加第二个导航栏。也就是说，虽然可以与标记栏一起使用它，虽然可以把它用作一个独立的视图系统，你也不能真正地把它放置到现有的导航栈中并使它正常显示。它将要被用作一个模式控制器。你将在下一个秘诀中看到此内容，这个秘诀只介绍处理其内置导航控制器的一个更直接的方法。

UIImagePickerController的导航控制器直接被集成在选取器对象中。这里不会按照处理普通视图栈的方式（即[self.navigationController popToRootViewControllerAnimated:YES]）返回到根视图。而是直接返回到控制器（即[self. popToRootViewControllerAnimated:YES]）。这就把控制权移动到了相册屏幕。为什么要这么做？通过在每个选择完成之后出栈，可以设置秘诀7-4中使用的一个循环，这就允许用户一个接一个地挑选相片，直到离开程序。

秘诀7-4显示了选取器的最简单模式，“从相册中选择”模式，如图7-4所示。设置选取器以使用任何合法的源类型。这3种类型的源如下所示：

- ❑ UIImagePickerControllerSourceTypePhotoLibrary，所有同步到iPhone的图像加上任意相册（这里使用的）；
- ❑ UIImagePickerControllerSourceTypeSavedPhotosAlbum，只是相册；
- ❑ UIImagePickerControllerSourceTypeCamera，让你能够使用iPhone内置的照相机拍照。

这个秘诀遵循基本的工作步骤。选择一个相册，选择一个图像，然后重复。这个简单的工作流程起作用是因为不包含编辑的图像。因为allowsImageEditing属性默认情况下为NO。因此，

任意选择（实际上就是任意的图像点击）都会把对UIImagePickerControllerDelegate对象的控制重定向到已完成图像选择的方法。图像编辑被启用时，回调流程会改变，你最好使用秘诀7-5中介绍的模式交互类型。

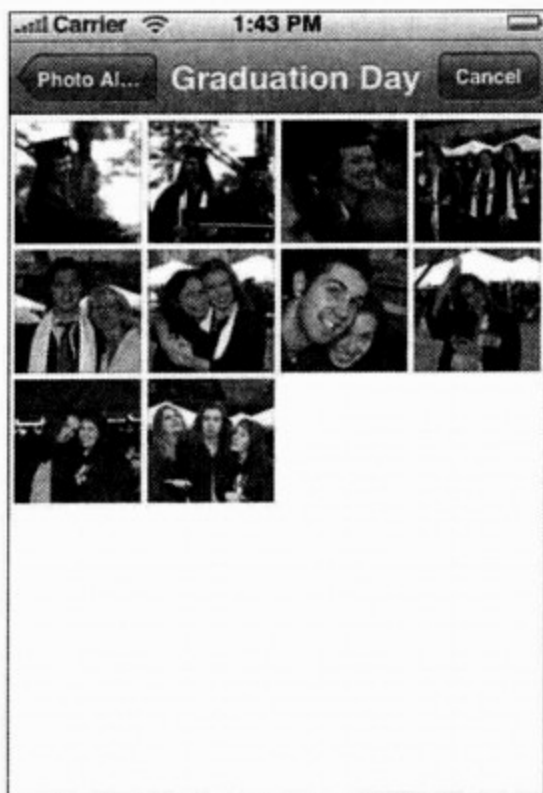


图7-4 苹果公司提供了几个内置相册，包括这个Graduation Day，用来进行仿真器测试

说明 秘诀7-4把选中图像的副本保存到Documents文件夹中。它获取图像的PNG表示（这返回NSData *）并写入文件。

秘诀7-4 选择一个图像并把它写入到Documents文件夹

```
@interface HelloController : UIImagePickerController
    <UIImagePickerControllerDelegate>
@end

@implementation HelloController

#define SOURCETYPE UIImagePickerControllerSourceTypePhotoLibrary
#define DOCSFOLDER [NSHomeDirectory() stringByAppendingPathComponent:@"Documents"]

- (id) init
{
    if (self = [super init]) self.title = @"Hello World";
    if ([UIImagePickerController isSourceTypeAvailable:SOURCETYPE])
        self.sourceType = SOURCETYPE;
    self.delegate = self;
    return self;
}
```



```

- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingImage:(UIImage *)image editingInfo:(NSDictionary *)editingInfo
{
    // Find a unique path name so files are not overwritten
    int i = 0;
    NSString *uniquePath = [DOCSFOLDER
    stringByAppendingPathComponent:@"selectedImage.png"];
    while ([[NSFileManager defaultManager] fileExistsAtPath:uniquePath])
        uniquePath = [NSString stringWithFormat:@"%s/%s-%d.%s", DOCSFOLDER,
        @"selectedImage", ++i, @"png"];

    printf("Writing selected image to Documents folder\n");
    [UIImagePNGRepresentation(image) writeToFile: uniquePath atomically:YES];
    [self popToRootViewControllerAnimated:YES];
}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker
{
    printf("User cancelled\n");
    [self popToRootViewControllerAnimated:YES];
}

@end

```

7.6 秘诀：选择和定制相册中的图像

此秘诀将扩展图像选取器交互，添加图像编辑功能。使用编辑或多或少地影响到模式交互。模式外观把用户的交互选项在某一段时间内限制在单个视图中，在这里是 `UIImagePickerController` 表示的视图。第4章介绍了用于警告视图和动作表对象的模式交互。秘诀7-5结合图像选取器使用模式。

模式外观对每个 `UIViewController` 都是内置的。现在介绍它们的工作方式。当父控制器调用 `presentModalViewController:animated:` 时，其子视图控制器在屏幕上从窗口的底部开始滑动。用户与模式控制器进行交互，完成工作，然后你就调用 `dismissModalViewControllerAnimated:` 方法。这会使控制器撤回。模式外观允许你暂时把控制重定向到主视图结构之外，并当子任务完成时再返回。

要在 `UIImagePickerController` 中启用图像编辑，应该把 `allowsImageEditing` 属性设置为 `YES`。这使用户

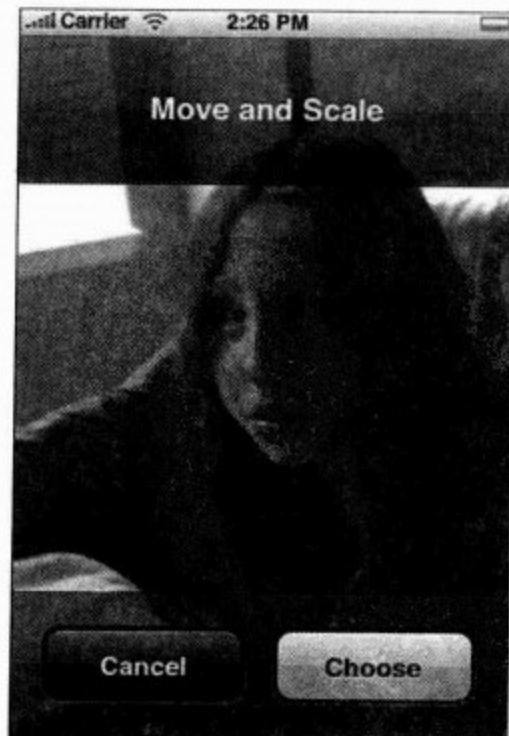


图7-5 使用交互式图像编辑器进行移动、缩放和选择最终外观

能够在选择图像之后，或者在摄像时拍下一张照片之后，调整图像的大小和位置。在iPhone上，这种行为在Settings的Set Wallpaper特性得到了使用。图7-5显示了后选择（post-selection）编辑器。

这个窗口提示用户按自己的期望移动和缩放图像。当用户点击Choose时，控制权移动到选取器委托。点击Cancel时，控制又回到相册视图。

委托方法UIImagePickerController: didFinishPickingImage:editingInfo:返回图像的已编辑版本作为第二个参数。此图像反映了用户指定的缩放和移动。返回的UIImage比较小，其大小被调整为正好适合iPhone屏幕。

现在，第三个参数editingInfo字典非常重要。它包含一个源图像的副本和一个表示图像裁剪的矩形。这些项被设置为字典的对象，当需要时可以重新使用它们。

说明 要填充iPhone仿真器上的相册（camera roll），要在~/Library/Application Support/iPhone Simulator/User中找到用户文件系统。把100APPLE文件夹从iPhone真机中复制到Media/DCIM文件夹中。要确保同时复制了JPG图像和小的THM略缩图文件。

秘诀7-5 将用户编辑添加到图像选择

```
@interface HelloController : UIImagePickerController
    <UIImagePickerControllerDelegate>
@end

@implementation HelloController

#define SOURCETYPE UIImagePickerControllerSourceTypeSavedPhotosAlbum
// #define SOURCETYPE UIImagePickerControllerSourceTypePhotoLibrary
#define DOCSFOLDER [NSHomeDirectory() stringByAppendingPathComponent:@"Documents"]

- (id) init
{
    if (!(self = [super init])) return self;

    // Set up to use the photo roll aka "Saved Photos"
    if ([UIImagePickerController isSourceTypeAvailable:SOURCETYPE])
        self.sourceType = SOURCETYPE;
    self.allowsImageEditing = YES;
    self.delegate = self;

    return self;
}

- (void)imagePickerController:(UIImagePickerController *)picker
    didFinishPickingImage:(UIImage *)image editingInfo:(NSDictionary *)editingInfo
{
    // Find a unique path name so files are not overwritten
    int i = 0;
    NSString *uniquePath = [DOCSFOLDER
        stringByAppendingPathComponent:@"selectedImage.png"];

```

```

while ([[NSFileManager defaultManager] fileExistsAtPath:uniquePath])
    uniquePath = [NSString stringWithFormat:@"%s/%s-%d.%s", DOCSFOLDER,
        @"selectedImage", ++i, @"png"];

printf("Writing selected image to Documents folder\n");
[UIImagePNGRepresentation(image) writeToFile: uniquePath atomically:YES];
// Show the editing changes
printf("Dictionary: %s\n", [[editingInfo description] UTF8String]);

// Pop after saving the edited file
[[self parentViewController] dismissModalViewControllerAnimated:YES];
}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker
{
    [[self parentViewController] dismissModalViewControllerAnimated:YES];
}

- (void) dealloc
{
    [super dealloc];
}

@end

@interface PrimaryController : UIViewController
@end

@implementation PrimaryController
- (void) snap
{
    [self presentModalViewController:[HelloController alloc] init] animated:YES];
}

- (void) loadView
{
    [super loadView];

    UIView *contentView = [[UIView alloc] initWithFrame:[UIScreen mainScreen]
        applicationFrame];
    contentView.backgroundColor = [UIColor whiteColor];
    self.view = contentView;
    [contentView release];
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
        initWithTitle:@"Select"
        style:UIBarButtonItemStylePlain
        target:self
        action:@selector(snap)] autorelease];
}

@end

```

7.7 秘诀：使用 iPhone 照相机拍照

秘诀7-4和秘诀7-5介绍了3个UIImagePickerController模式中的两个：从整个照片集中进行选择 and 只从相册中选择。秘诀7-6介绍了第3种模式，使用iPhone内置照相机拍照。图7-6实际地显示了此界面。此特性在仿真器中无效。它必须从iPhone上运行才能工作。

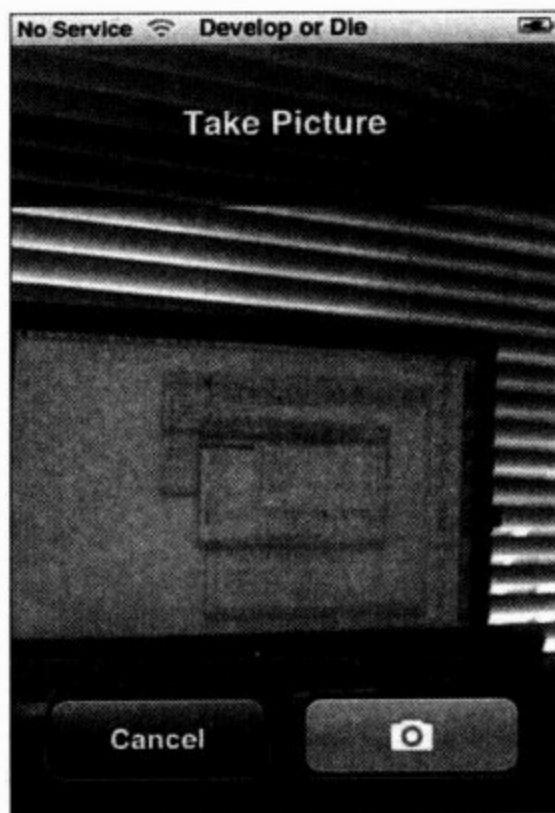


图7-6 通过iPhone照相机界面，可以拍照，调整其大小和缩放，如有必要还可以重拍

与其他模式一样，可以允许或禁止图像编辑作为照片捕获过程的一部分。照相机交互带来的无与伦比的一个特性是预览（Preview）屏幕，在用户点击绿色的照相机“拍照（snap）”按钮之后就显示该屏幕。通过预览屏幕，你能够补拍相片或照原来的样子使用照片。一旦点击Use Photo，控制权就传递到下一个阶段。如果启动了图像编辑，控制权的传递会在下一步发生；如果没有启动图像编辑，控制权就移动到标准的“确实完成选择”方法。

与编辑一样，为拍照使用一个模式外观。如果没有，你就必须自己解决把控制权返回到主程序的问题。把照相机界面用作一个模式视图控制器会使呈现和对照相机的响应变得简单。

说明 按住Sleep/Wake按钮并点击Home键拍摄屏幕快照，如图7-6所示。屏幕会闪一下白光，表示快照已被添加到DCIM文件夹。换句话说，我们在Xcode中使用iPhone Organizer（Control-Command-O）来拍摄屏幕的活动画面。

秘诀7-6 访问iPhone照相机

```
@implementation CameraController
```

```
#define SOURCETYPE UIImagePickerControllerSourceTypeCamera
```



```

#define DOCSFOLDER [NSHomeDirectory() stringByAppendingPathComponent:@"Documents"]

- (id) init
{
    if (!(self = [super init])) return self;

    // Set up as the camera source
    if ([UIImagePickerController isSourceTypeAvailable:SOURCETYPE])
        self.sourceType = SOURCETYPE;

    // Just snap here. Although you can easily add image editing if you want
    self.allowsImageEditing = NO;

    self.delegate = self;
    return self;
}

- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingImage:(UIImage *)image editingInfo:(NSDictionary *)editingInfo
{
    // Find a unique path name so files are not overwritten
    int i = 0;
    NSString *uniquePath = [DOCSFOLDER
        stringByAppendingPathComponent:@"selectedImage.png"];
    while ([[NSFileManager defaultManager] fileExistsAtPath:uniquePath])
        uniquePath = [NSString stringWithFormat:@"%d/%d-%d.%d", DOCSFOLDER,
            @"selectedImage", ++i, @"png"];

    printf("Writing selected image to Documents folder\n");
    [UIImagePNGRepresentation(image) writeToFile: uniquePath atomically:YES];

    // Thread of control ends here
    [[self parentViewController] dismissModalViewControllerAnimated:YES];
    [picker release];
}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker
{
    [[self parentViewController] dismissModalViewControllerAnimated:YES];
    [picker release];
}

@end

```

7.8 处理 iPhone 音频

和你希望的一样，iPhone及其基于iPod的产品是具有强大音频功能的设备。在程序中使用iPhone音频简单而直接。虽然苹果公司曾经有一个像QuickTime一样的公共类Celestial，在Firmware 2.0中，它将这个类移到了其PrivateFrameworks文件夹中。这就使开发人员接触不到许

多美丽、优雅而易于使用的类和例程，正是它们使得iPhone音频重放并记录美丽的东西。使用iPhone SDK，开发人员只能使用两个方法，System Audio和Audio Queues。

- ❑ **System Audio**服务使你能够回放短的声音，同时提供了报警和振动。秘诀4-11介绍了这些调用，并显示了如何在程序中使用它们。
- ❑ **Audio Queues**服务为iPhone的Audio Toolbox框架提供了一个基于C的界面。iPhone SDK官方没有为常规音频重放提供任何简单和实用的Objective-C类。苹果公司提供的程序示例很复杂，而且有些小漏洞。

不过，如果你乐意钻研，还有两个可用的优雅和常规的重放解决方法。这些解决方法隐藏在Media Player框架中。秘诀7-7和秘诀7-8分别介绍了每个方法。第一种方法为音频重放获取对Celestial框架的幕后访问。第二种方法使用MPMoviePlayer类重放音频和电影。

说明 iPhone沙盒不提供对机载iTunes Music Library的访问。你不能在Library中查看或重放任何文件，甚至不能从“授权”程序中获取列表或请求重放。你只能重放那些可以访问的文件，通常只限于在应用程序束和Documents文件夹中的文件。

7.9 秘诀：使用 Celestial 播放音频

正式情况下，Celestial媒体框架没有限制。非正式情况下，Media Player框架为音频重放提供了几个内置Celestial特性。那是因为iPhone的媒体播放器基于Celestial，而且它公开了几个类。你可以通过无正式文档的API调用来访问这些类。以下类能用于播放几乎所有类型音频。

- ❑ **MPItem**。在Celestial中，AVItem对象提供来自iPhone上电影和音频文件的数据。MPItem提供标准Celestial AVItem的一个媒体播放器版本。使用当前媒体的路径初始化MPItem实例。
- ❑ **MPAVController**。这个类提供了一个集中的视听控制器，以处理条目重放和队列管理的问题。MPAVController的一个共享实例在任意时候都保持可用。
- ❑ **MPArrayQueueFeeder**。这个音频队列存储需要重放的挂起媒体(pending medio)的列表，像在播放列表中一样。使用一个路径数组对它进行初始化，其中每个路径指向希望重放的每个条目。
- ❑ **AVController**。这个类是标准的Celestial AV控制器，它构成了MPAVController对象的部分内容。可以通过共享Media Player控制器对象访问这个基于Celestial的控制器，并要求它重放MPItem数据。

其中每个类都包含在Media Player框架中，并且是可链接的。可以通过提供头文件来在应用程序中使用它们。代码清单7-2提供了该头文件(Playback.h)，它显示了这些类以及它们最基本的界面。

说明 有关在程序中使用文档中未记录的调用和特性，第1章进行了更深入的讨论。

代码清单7-2 Playback.h

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

// Celestial's AV Item
@interface AVItem : NSObject
@end

// Media Player's version of AVItem
@interface MPItem : AVItem
- (id)initWithPath:(NSString *)path error:(id *)anError;
@end

// Media Player's primary playback controller
@interface MPAVController : NSObject
+ (id) sharedInstance;
- (id) avController;
@end

// Celestial's AVController
@interface AVController : NSObject
- (void) play: (id *) whatever;
- (void) pause;
- (void) setQueueFeeder:(id)fp8;
@end

// Celestial's AV Queue Feeder
@interface AVQueueFeeder : NSObject
@end

// Media Player's version of Celestial's feeder
@interface MPQueueFeeder : AVQueueFeeder
@end

// The Media Player array-based feeder
@interface MPArrayQueueFeeder : MPQueueFeeder
- (id) initWithPaths:(id)fp8;
@end
```

秘诀7-7呈现了一个极其朴实无华的播放器，它从应用程序束加载Martin Luther King的“I have a dream”音频副本并进行重放。此方法分配了一个新的MPItem，并使用到音频文件的路径对它进行了初始化。共享音频控制器加载该MPItem并开始重放。在大约5个调用中，你可以获得逐页对AudioQueue编程的价格功能。

使用Celestial的优势有很多。你不需要处理复杂的AudioQueue调用。Celestial的AVController类几乎可以处理你指定的所有音频格式。你可以通过少许编程得到很好的性能。

不过也有缺点。Celestial重放会干扰任何正在进行的iPod重放。开始播放音频时会打断正在播放的歌曲。相比之下，System Audio服务重放没有这样的干扰。它会用你需要播放的短报警覆

盖任何iPod音频。

要停止重放，需要调用`[avc pause]`。但在此后不能“停止暂停”以继续重放。

`AVController`广播你希望了解的若干个通知。其中最重要的两个是`AVController_ItemReadyToPlay`和`AVController_ItemPlaybackDidEnd`。前者在队列完成数据加载时会告知程序，这会花费一两秒的时间。后者在重放实际结束时会让你知道，这与用户点击Pause或Stop按钮相反。`AVController`也发送有关队列选择更改等更多内容的通知。你可以订阅所有通知（详细信息请参见第1章），以监听`AVController`的状态广播。

秘诀7-7 使用Celestial的AVController重放音频

```
- (void) startPlayback : (id) sender
{
    // Recover the AV Controller
    avc = [[MPAVController sharedInstance] avController];

    // Set up a queue to feed the controller
    id feeder = [[MPArrayQueueFeeder alloc] initWithPaths:[NSArray
    arrayWithObject:path]];
    [avc setQueueFeeder:feeder];
    [feeder release];

    // Start playback
    [avc play:nil];
}
```

7.10 秘诀：使用媒体播放器实现音频和视频重放

如图7-7所示，全屏媒体播放器界面提供了拖动、播放/暂停等控件。这些特性可用于音频和视频重放。其区别在于提供的文件URL是指向一个音频文件还是一个视频文件。此秘诀会将视频URL换为音频URL，并最终为你展示一个全屏电影。



图7-7 全屏媒体播放器界面为用户提供对音频重放的全面控制

与本章中的其他示例一样，MPMoviePlayerController按照自己的规则播放。不要把它放入导航栈，不要模式化地调用它，而是创建它然后告知它进行播放。它控制屏幕，提供图7-7中所示的重放控制。要重新获得控制权，就把应用程序订阅到MPMoviePlayerPlaybackDidFinishNotification通知。在两种情况下发送此通知：当重放实际结束时或当用户点击Done时。

秘诀7-8由两个方法组成。一个方法启动重放，另一个方法在重放结束时进行清理。此精确代码可用于重放音频或重放视频。所有需要提供的是一个电影文件，而不是音频文件。所支持的文件类型包括MOV、MP4、MPV、M4V和3GP，以及MP3、AIFF、M4A，等等。

秘诀7-8 使用MPMoviePlayerController重放音频或视频

```

- (void)myMovieFinishedCallback: (NSNotification*) aNotification
{
    // Remove and release the observer
    MPMoviePlayerController* theMovie=[aNotification object];
    [[NSNotificationCenter defaultCenter] removeObserver:self
    ➤name:MPMoviePlayerPlaybackDidFinishNotification object:theMovie];
    [theMovie release];

    // restore the status bar to portrait orientation
    [[UIApplication sharedApplication]
    ➤setStatusOrientation:UIInterfaceOrientationPortrait animated:YES];
}

- (void) startPlayback : (id) sender
{
    // create a new player and initialize it with the path
    NSString *path = [[NSBundle mainBundle] pathForResource:@"dream"
    ➤ ofType:@"m4a"];
    MPMoviePlayerController* theMovie=[[MPMoviePlayerController alloc]
    ➤ initWithContentURL:[NSURL URLWithString:path]];
    theMovie.scalingMode=MPMovieScalingModeAspectFill;
    // Add an observer to catch when playback ends
    [[NSNotificationCenter defaultCenter] addObserver:self
    ➤ selector:@selector(myMovieFinishedCallback:)
    ➤ name:MPMoviePlayerPlaybackDidFinishNotification object:theMovie];

    // Start playback
    [theMovie play];
}

```

7.11 秘诀：录制音频

不幸的是，Celestial中愚蠢的易于使用的音频录制器对象好像没有从1.1.4转换到2.0。这就把Audio Queue录制留给了开发人员。秘诀7-9提供了一个包装在Objective-C类中的录制梗概。注意类是如何推迟一秒钟结束录制，以便完全清除挂起缓冲区的。此样例完全基于苹果公司的示例代码，还要感谢trailsinthesand.com网站的出色教程。

秘诀7-9 基本Audio Queue录制

```

#import <UIKit/UIKit.h>
#import <AudioToolbox/AudioQueue.h>
#import <AudioToolbox/AudioFile.h>

#define NUM_BUFFERS 3
#define kAudioConverterPropertyMaximumOutputPacketSize 'xops'
#define DOCUMENTS_FOLDER [NSHomeDirectory()
    stringByAppendingPathComponent:@"Documents"]

// Custom recording state storage
typedef struct
{
    AudioFileID          audioFile;
    AudioStreamBasicDescription dataFormat;
    AudioQueueRef         queue;
    AudioQueueBufferRef   buffers[NUM_BUFFERS];
    UInt32                bufferSize;
    SInt64                currentPacket;
    BOOL                  recording;
} RecordState;

// Objective-C wrapper for the Audio Queue functionality
@interface Recorder : NSObject {
    RecordState recordState;
    CFURLRef fileURL;
}

- (BOOL)    isRecording;

// Automatically generate a dated file in Documents
- (void)    toggleRecording;

// Manual recording
- (void)    startRecording: (NSString *) filePath;
- (void)    stopRecording;

@end

// Derive the Buffer Size. I punt with the max buffer size.
void DeriveBufferSize (
    AudioQueueRef          audioQueue,
    AudioStreamBasicDescription ASBDescription,
    Float64                seconds,
    UInt32                  *outBufferSize
) {
    static const int maxBufferSize = 0x50000;

```

```

int maxPacketSize = ASBDescription.mBytesPerPacket;
if (maxPacketSize == 0) {
    UInt32 maxVBRPacketSize = sizeof(maxPacketSize);
    AudioQueueGetProperty (
        audioQueue,
        kAudioConverterPropertyMaximumOutputPacketSize,
        &maxPacketSize,
        &maxVBRPacketSize
    );
}

Float64 numBytesForTime =
ASBDescription.mSampleRate * maxPacketSize * seconds;
*outBufferSize = (UInt32) ((numBytesForTime < maxBufferSize) ?
numBytesForTime : maxBufferSize);
}

// Forward Declaration to Input Buffer Handler
static void HandleInputBuffer (
    void                                *aqData,
    AudioQueueRef                       inAQ,
    AudioQueueBufferRef                 inBuffer,
    const AudioTimeStamp                *inStartTime,
    UInt32                              inNumPackets,
    const AudioStreamPacketDescription *inPacketDesc
);

// Assign the file metadata
OSStatus SetMagicCookieForFile (
    AudioQueueRef inQueue,
    AudioFileID   inFile
) {
    OSStatus result = noErr;
    UInt32 cookieSize;

    if (
        AudioQueueGetPropertySize (
            inQueue,
            kAudioQueueProperty_MagicCookie,
            &cookieSize
        ) == noErr
    ) {
        char* magicCookie =
        (char *) malloc (cookieSize);
        if (
            AudioQueueGetProperty (
                inQueue,
                kAudioQueueProperty_MagicCookie,

```

```

        magicCookie,
        &cookieSize
    ) == noErr
    )
    result = AudioFileSetProperty (
        inFile,
        kAudioFilePropertyMagicCookieData,
        cookieSize,
        magicCookie
    );

    free (magicCookie);
}
return result;
}

// Write out current packets
static void HandleInputBuffer (
    void                                *aqData,
    AudioQueueRef                       inAQ,
    AudioQueueBufferRef                 inBuffer,
    const AudioTimeStamp                *inStartTime,
    UInt32                              inNumPackets,
    const AudioStreamPacketDescription *inPacketDesc
) {
    RecordState *pAqData = (RecordState *) aqData;

    if (inNumPackets == 0 &&
        pAqData->dataFormat.mBytesPerPacket != 0)
        inNumPackets =
            inBuffer->mAudioDataByteSize / pAqData->dataFormat.mBytesPerPacket;

    if (AudioFileWritePackets (
        pAqData->audioFile,
        NO,
        inBuffer->mAudioDataByteSize,
        inPacketDesc,
        pAqData->currentPacket,
        &inNumPackets,
        inBuffer->mAudioData
    ) == noErr) {
        pAqData->currentPacket += inNumPackets;

        if (pAqData->recording == 0)
            return;

        AudioQueueEnqueueBuffer (
            pAqData->queue,
            inBuffer,
            0,

```



```

        NULL
    };
}

// Write the buffers out
void AudioInputCallback(
    void *inUserData,
    AudioQueueRef inAQ,
    AudioQueueBufferRef inBuffer,
    const AudioTimeStamp *inStartTime,
    UInt32 inNumberPacketDescriptions,
    const AudioStreamPacketDescription *inPacketDescs)
{
    RecordState* recordState = (RecordState*)inUserData;
    if(!recordState->recording) { printf("Not recording, returning\n"); return;
}

    printf("Writing buffer %d\n", recordState->currentPacket);
    OSStatus status = AudioFileWritePackets(recordState->audioFile,
                                            NO,
                                            inBuffer->mAudioDataByteSize,
                                            inPacketDescs,
                                            recordState->currentPacket,
                                            &inNumberPacketDescriptions,
                                            inBuffer->mAudioData);

    if(status == 0) // success
    {
        recordState->currentPacket += inNumberPacketDescriptions;
    }

    AudioQueueEnqueueBuffer(recordState->queue, inBuffer, 0, NULL);
}

// For use with the file naming, which is automatic
@interface NSDate (extended)
-(NSDate *) dateWithCalendarFormat:(NSString *)format timeZone: (NSTimeZone *)
    timeZone;
@end

@implementation Recorder

// Initialize the recorder
- (id) init
{
    self = [super init];
    recordState.recording = NO;
    return self;
}

```

```

// Set up the recording format as low-quality mono AIFF
- (void)setupAudioFormat:(AudioStreamBasicDescription*)format
{
    format->mSampleRate = 8000.0;
    format->mFormatID = kAudioFormatLinearPCM;
    format->mFormatFlags = kLinearPCMFormatFlagIsBigEndian |
        kLinearPCMFormatFlagIsSignedInteger | kLinearPCMFormatFlagIsPacked;

    format->mChannelsPerFrame = 1; // mono
    format->mBitsPerChannel = 16;
    format->mFramesPerPacket = 1;
    format->mBytesPerPacket = 2;
    format->mBytesPerFrame = 2; // not used, apparently required
    format->mReserved = 0;
}

// Begin recording
- (void)startRecording:(NSString *)filePath
{
    [self setupAudioFormat:&recordState.dataFormat];
    fileURL = CFURLCreateFromFileSystemRepresentation(NULL, (const UInt8 *)
        [filePath UTF8String], [filePath length], NO);
    recordState.currentPacket = 0;

    OSStatus status;
    status = AudioQueueNewInput(&recordState.dataFormat,
                                HandleInputBuffer,
                                &recordState,
                                CFRunLoopGetCurrent(),
                                kCFRunLoopCommonModes,
                                0,
                                &recordState.queue);

    if (status != 0) {
        printf("Could not establish new queue\n");
        return;
    }

    status = AudioFileCreateWithURL(fileURL,
                                    kAudioFileAIFFType,
                                    &recordState.dataFormat,
                                    kAudioFileFlags_EraseFile,
                                    &recordState.audioFile);

    if (status != 0)
    {
        printf("Could not create file to record audio\n");
        return;
    }
}

```

```

DeriveBufferSize (
    recordState.queue,
    recordState.dataFormat,
    0.5,
    &recordState.bufferByteSize
);
// printf("Buffer size: %d\n", recordState.bufferByteSize);
for(int i = 0; i < NUM_BUFFERS; i++)
{
    status = AudioQueueAllocateBuffer(recordState.queue,
                                      recordState.bufferByteSize,
                                      &recordState.buffers[i]);

    if (status) {
        printf("Error allocating buffer %d\n", i);
        return;
    }
    status = AudioQueueEnqueueBuffer(recordState.queue,
    ↪recordState.buffers[i], 0, NULL);
    if (status) {
        printf("Error enqueueing buffer %d\n", i);
        return;
    }
}

status = SetMagicCookieForFile(recordState.queue, recordState.audioFile);
if (status != 0)
{
    printf("Magic cookie failed\n");
    return;
}

status = AudioQueueStart(recordState.queue, NULL);
if (status != 0)
{
    printf("Could not start Audio Queue\n");
    return;
}

recordState.currentPacket = 0;
recordState.recording = YES;
return;
}

// There's generally about a one-second delay before the buffers fully empty
- (void) reallyStopRecording
{
    AudioQueueFlush(recordState.queue);
    AudioQueueStop(recordState.queue, NO);
    recordState.recording = NO;
}

```

```

recordState.recording = NO;
SetMagicCookieForFile(recordState.queue, recordState.audioFile);
for(int i = 0; i < NUM_BUFFERS; i++)
{
    AudioQueueFreeBuffer(recordState.queue,
                          recordState.buffers[i]);
}

AudioQueueDispose(recordState.queue, YES);
AudioFileClose(recordState.audioFile);
}

// Stop the recording after waiting just a second
- (void) stopRecording
{
    [self performSelector:@selector(reallyStopRecording) withObject:NULL
     afterDelay:1.0f];
}

// Automatically create a file in Documents and start/stop recording to it
- (void) toggleRecording
{
    // Create a new dated file
    NSDate *now = [NSDate dateWithTimeIntervalSinceNow:0];
    NSString *caldate = [[now dateWithCalendarFormat:@"%b_%d_%H_%M_%S"
     timeZone:nil] description]; // no : 's.
    NSString *filePath = [NSString stringWithFormat:@"%s/%s.aiff",
     DOCUMENTS_FOLDER, caldate];

    if(!recordState.recording)
    {
        printf("Starting recording\n");
        [self startRecording: filePath];
    }
    else
    {
        printf("Stopping recording\n");
        [self stopRecording];
    }
}

- (BOOL) isRecording
{
    return recordState.recording;
}

@end

```


7.12 读入文本数据

在所有iPhone媒体中，文本数据被证明（理所当然地）是最容易读入和写出的。NSString类可以轻松地从iPhone文件加载文本数据：

```
NSString *path = [[NSBundle mainBundle] pathForResource:@"myText"
ofType:@"txt"];
NSString *fileText = [NSString stringWithContentsOfFile:path encoding:
NSUTF8StringEncoding error:nil];
```

从文件内容加载一个字符串之后，可以把它添加到一个文本视图中（textView.text = fileText），把它分隔成多个字符串以填充，例如，一个表（[fileText componentsSeparatedByString:@"\n"]），或者可能把它用作文本警告的主体（参见第4章）。

要输出文本，就请求字符串把自身写出到一个文件：

```
NSError *error;
[myString writeToFile:path atomically:YES encoding:NSUTF8StringEncoding
error:&error];
```

显示属性列表

属性列表，即苹果公司用来存储默认设置和其他类型系统数据的文件，确切地说并不是“媒体”。另一方面，你可能希望从磁盘读取并在屏幕上显示这些东西。下面介绍如何读入一个属性列表（在这里是SpringBoard首选项，它是可读的，虽然有沙盒限制），以及如何生成一个适合在你最爱的UITextView或UILabel中显示的字符串描述。此方法适用于那些带有strings扩展名和plist扩展名的文件：

```
#define PLISTPATH @"/private/var/mobile/Library/Preferences/
com.apple.springboard.plist"

NSDictionary *plist = [NSMutableDictionary
dictionaryWithContentsOfFile:PLISTPATH];
if (plist) textView.text = [plist description];
else textView.text = @"Could not read property list from file";
```

大多数复杂结构，包括设置、数组、字典等，都会按需应变地返回一个NSString description。它是将一个属性列表转换为可见格式的一种好方法。另一种方法是，将NSString XML属性列表表示转换为一个属性列表，使用propertyList方法（如idplist = [myXMLString propertyList]）。

说明 属性列表和字典也可以被串行化为一个XML字符串。第10章显示了如何实现这一目的，以及如何从其串行化格式还原数据。

7.13 从备份文件还原媒体

当iTunes备份iPhone时，它会存储来自每个应用程序的Documents和Library文件夹的数据。这

些文件被编码和总体检验，并被添加到一个中心备份清单。在Macintosh上，各同步iPhone的备份文件夹位于~/Library/Application Support/MobileSync/Backup；在Windows中，位于C:\Documents and Settings\UserName\Application Data\Apple Computer\SyncServices\Local。在每个文件夹中，都有一个Info.plist文件、一个Manifest.plist文件和许多mbackup文件。

- Info.plist文件描述设备、其名称、序列号和iTunes首选项，以及其他特定于设备的信息。
- Manifest.plist文件存储所有备份文件内容的一个表，这些文件的修改时间和数据散列。这样做可以达到两个目的。它保证了数据与备份时的数据保持一致，并阻止第三方开发人员在恢复数据时将自定义数据加载回iPhone。前者是一件好事，后者就不是那么重要了。
- Mbackup文件包含备份到计算机的每个文件的串行化版本。基于Macintosh的mdhelper命令行实用程序 (<http://ericasadun.com/ftp/Macintosh>) 全面扫描这些移动设备备份文件，并从中恢复备份数据。

串行化数据位于编码块中。访问此块并以其原来的形式把它存储到磁盘是很简单的，只需要3个步骤。

(1) 把mbackup文件读入到一个字典中：

```
NSDictionary *mddict = [NSDictionary
➤ dictionaryWithContentsOfFile:backupPath];
```

(2) 提取编码数据块：

```
NSData *data = [mddict objectForKey:@"Data"];
```

(3) 存储数据到磁盘：

```
[data writeToFile:testPath atomically:YES];
```

通过这3个步骤，可以把那些令人费解的数据块转换为原文件，不管是为清单还是为单独的备份数据。秘诀7-10显示了mdhelper例程，它从备份文件夹中提取清单。

秘诀7-10 从iTunes备份恢复清单数据

```
// Recover all manifests from available folders
void getManifests()
{
    for (NSString *path in backupFolders())
    {
        NSString *fullPath = [backupDirPath()
➤ stringByAppendingPathComponent:path];

        // Get the path to the Manifest.plist file
        NSString *manifestPath = [fullPath
➤ stringByAppendingPathComponent:@"Manifest.plist"];

        // Get the path to the Info.plist file
        NSString *infoPath = [fullPath
➤ stringByAppendingPathComponent:@"Info.plist"];

        // Read in the manifest and info dictionaries
```



```

NSMutableDictionary *manifestDict = [NSMutableDictionary
➤dictionaryWithContentsOfFile:manifestPath];
NSMutableDictionary *infoDict = [NSMutableDictionary
➤dictionaryWithContentsOfFile:infoPath];

// Read in the device name from Info.plist
NSString *deviceName = [infoDict objectForKey:@"Device Name"];

// Read in the manifests's data glob
NSData *data = [manifestDict objectForKey:@"Data"];

// Output the manifest's data glob to disk
NSString *outfile = [[recoveryFolderPath()
➤stringByAppendingPathComponent:deviceName] stringByAppendingString:@"-
➤Manifest.plist"];
if (data) printf("Extracting manifest for device %s\n", [deviceName
➤UTF8String]);
[data writeToFile:outfile atomically:YES];

[data release];
[outfile release];
[deviceName release];
[infoPath release];
[manifestPath release];
[fullPath release];
}
}

```

7.14 小结

7

本章介绍了处理媒体的多种方法，包括加载、存储、选择和提取文件。你已经看到了处理音频、视频、图像和文本数据的秘诀。在继续阅读本书之前，先看看有关这些秘诀的一些要点。

- ❑ 不管处理图像还是音频，选择一个最能与当前文件的数据匹配的表示方法。对于iPhone来说，没有对媒体一劳永逸的解决方法。
- ❑ 虽然苹果公司没有把Celestial AVController类作为公共类，但可以在公共框架中使用它。你只需要提供头文件，苹果公司负责提供需要的功能。
- ❑ Audio Queue提供强大的低级音频例程，但它们不是为胆小鬼和那些只想要一个快速解决方案的人准备的。如果你需要Audio Queue带来的细化音频控制类型，苹果公司提供了实现此目的所需的扩展文件。
- ❑ 许多类都提供比说明更多的功能：使用UIWebView对象显示HTML、PDF文件和图像，使用MPMoviePlayerController重放电影和音频文件。

UIControl类为iPhone的许多交互式元素提供了基础，包括按钮、文本字段、滑块和开关。与它们的祖先类相比，这些控件有更多共同点。控件都使用相似的布局和目标操作（target-action）方法。本章介绍控件及其用法，探索如何采用多种方式构建和定制控件。你将了解已在文档中细致记录的SDK调用和未在文档中细致记录的SDK调用。你还会学习如何通过分解UIControl视图结构（hierarchy）来访问和修改控件元素。从简单到复杂，本章介绍了大量可以在程序中重新使用的控件秘诀。

8.1 秘诀：构建简单的按钮

iPhone支持若干种按钮类型。两个最常使用的按钮是从头开始创建的独立的UIButton实例，和专门在栏（包括导航栏、标签栏和工具栏）上使用的UIBarButtonItem实例。

UIBarButtonItem提供了一个极其简单的方法，用最少的代码向程序添加按钮操作功能。代码清单8-1显示了一个典型的调用，你可以在UIViewController loadView方法中找到该调用。使用一个标题、一种样式、一个目标和一个操作来创建按钮。父视图，不管是导航栏、标签栏还是工具栏，都会为你处理按钮的位置和大小问题。

代码清单8-1 向导航栏添加UIBarButtonItem

```
// Add a randomize button
UIBarButtonItem *randomButton = [[[UIBarButtonItem alloc]
    initWithTitle:@"Randomize"
    style:UIBarButtonItemStylePlain
    target:self
    action:@selector(randomize)] autorelease];
self.navigationItem.rightBarButtonItem = randomButton;
```

UIBarButtonItem使用目标操作。与所有控件相同，当用户与它们进行交互时，它们允许添加至少一个目标和回调操作。栏按钮条目（bar button item）不指定相关事件的类型，它唯一关注的事件是按钮点击。

使用栏按钮条目的缺点是不能把它们随意地放置在屏幕上。它们不是视图。因此，不能设置它们的结构并把它们添加到其他视图中。拥有它们的栏会构建和管理它们的外观。要把按钮放置

在任意视图中，需要使用UIButton类。

说明 按钮由其父视图构建完成之后，就可以访问栏按钮条目的真实视图了。调用文档中未记录的UIButton view方法。在没有遇到极其复杂问题的情况下，不能将该视图从其父视图中移除并把它看成一个独立的对象。而应使用视图添加像（不受文档支持）UIToolbarButtonBadge类这样的子视图，如本章后面的内容所示。

8.1.1 UIButton 类

UIButton实例向iPhone应用程序添加交互式按钮。它们提供更传统的控件，包括用于目标操作调用的边框（frame）和事件规范。点击这些按钮时，它们向指定的目标发送消息。iPhone提供了构建UIButton的两种方法。你可以使用一个预置按钮类型或从头开始构建一个自定义按钮。

当前的iPhone SDK提供了下面这些预置按钮类型。和你看到的一样，按钮类型并不是十分普遍。它们被添加到SDK主要是为了方便苹果公司，而不是方便你。虽然如此，你也可以在需要时在程序中使用它们。

- ❑ **Detail Disclosure**。当你向表单元格添加细节展开附件时，会看到圆形的、蓝色的带V形的圆圈。
- ❑ **Info Light**和**Info Dark**。这两个按钮提供了一个小的圆形的*i*，和你在Macintosh的Dashboard小配件中看到的一样。这些按钮在Weather和Stocks应用程序中使用，用来将视图从一边翻到另一边。
- ❑ **Contact Add**。这个圆形的蓝色圆圈中心有一个白色的“+”，在联系人应用程序中向地址簿添加新联系人时可以看到它。
- ❑ **Rounded Rectangle**。这个按钮提供了一个简单的环绕按钮文本的圆角矩形。它并不是特别有魅力的按钮（就是说，它并不十分像“苹果风格”的外观），不过在应用程序中编写和使用它非常简单。

要使用预置按钮，需要分配它、设置其边框并添加一个目标。不要为添加自定义样式或创建按钮的整体外观而担心，因为SDK负责所有这些工作。代码清单8-2展示了如何构建一个简单的圆角矩形按钮。要构建其他标准按钮类型，请忽略标题行。圆角矩形是唯一使用标题的预置按钮类型。

说明 使用UIControlEventTouchUpInside事件获取用户对按钮的点击。

代码清单8-2 构建“预置”圆角矩形按钮

```
UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[button setFrame:CGRectMake(0.0f, 0.0f, 80.0f, 30.0f)];
[button setCenter:CGPointMake(160.0f, 208.0f)];
[button setTitle:@"Beep" forState:UIControlStateNormal];
[button addTarget:self action:@selector(playSound:)
forControlEvents:UIControlEventTouchUpInside];
[contentView addSubview:button];
```

8.1.2 构建自定义按钮

使用UIButtonTypeCustom类型时，你需要提供所有的按钮样式。图像的数量取决于你希望按钮如何工作。对于一个简单的按钮，可以添加一个背景图像，并在按钮被按下时把标签颜色改为突出显示。对于一个开关类型的按钮，需要使用4个图像：正常显示下的“关”状态，突出显示下的“关”状态（即按钮按下时），还有“开”状态对应的两个图像。由你自己来选择和设计交互细节。

秘诀8-1构建了一个切换开关按钮，展示了有关构建自定义按钮的详细信息。点击按钮时，按钮会把它的样式从绿色切换到红色或者从红色切换到绿色。这样，你的用户（非色盲）能立刻识别当前状态。图8-1（左图）显示了使用此秘诀创建的按钮。

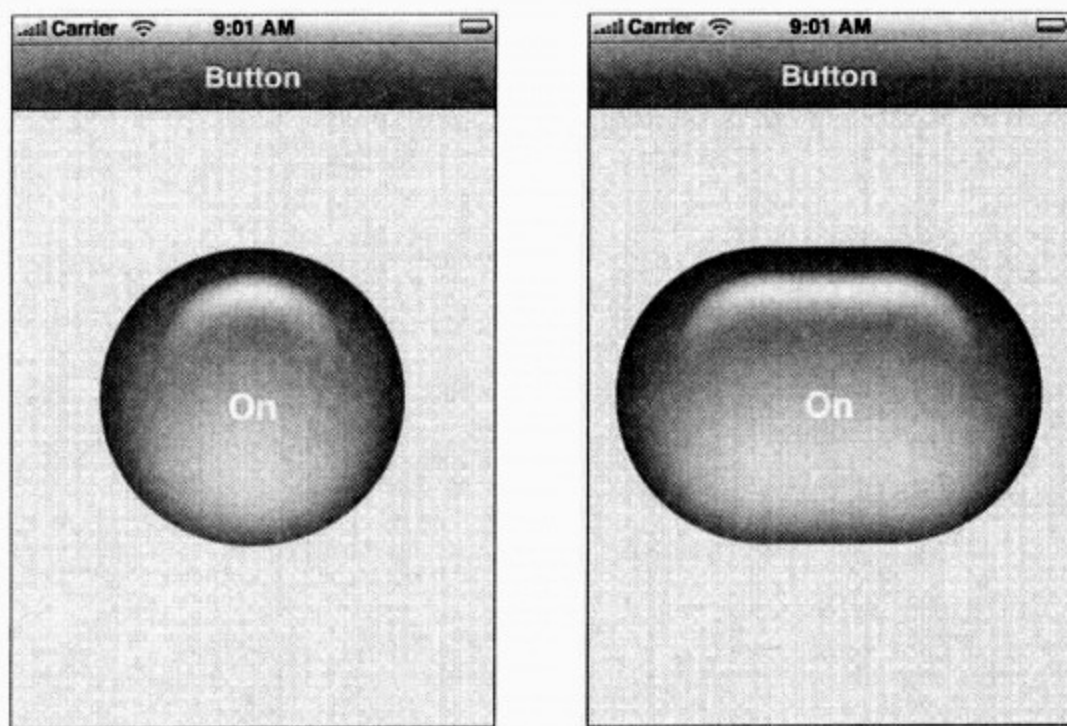


图8-1 使用UIImage拉伸来调整任意按钮宽度。设置左边罩（cap）的宽度以便指定从哪个位置进行拉伸

此秘诀中的UIImage可拉伸图像调用在按钮创建过程中起着重要作用。通过将圆形样式转换为菱形按钮，可拉伸图像使你能够创建任意宽度的按钮。你将指定任意一个边缘的罩（cap）（即不应该被拉伸的部分）。在本例中，罩为110像素宽。如果你打算将此秘诀中使用的220像素按钮宽度改为300，就在罩结束的那一点中间的位置拉伸按钮，如图8-1所示（右图）。

秘诀8-1 构建切换开关的UIButton

```
@interface HelloController : UIViewController
{
    UIImageView          *contentView;
    BOOL                 isOn;
}
@end
```

```

@implementation HelloController
- (UIButton *) buildButton: (NSString *) aTitle
{
    // Create a button sized to our art
    UIButton *button = [[UIButton alloc] initWithFrame:CGRectMake(0.0f, 0.0f,
        ➡220.0f, 233.0f)];

    // The cap width indicates the location where the stretch occurs
    [button setBackgroundImage:[UIImage imageNamed:@"green.png"]
        ➡stretchableImageWithLeftCapWidth:110.0 topCapHeight:0.0]
        ➡forState:UIControlStateNormal];
    [button setBackgroundImage:[UIImage imageNamed:@"green2.png"]
        ➡stretchableImageWithLeftCapWidth:110.0 topCapHeight:0.0]
        ➡forState:UIControlStateHighlighted];

    // Set up the button alignment properties
    button.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
    button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentCenter;

    // Set the title, font and color. The color switches from
    // white to gray for highlights
    [button setTitle:aTitle forState:UIControlStateNormal];
    [button setTitle:aTitle forState:UIControlStateHighlighted];

    [button setTitleColor:[UIColor whiteColor] forState:UIControlStateNormal];
    [button setTitleColor:[UIColor lightGrayColor]
        ➡forState:UIControlStateHighlighted];

    [button setFont:[UIFont boldSystemFontOfSize:24.0f]];

    isOn = NO;
    return [button autorelease];
}

- (void) toggleButton: (UIButton *) button
{
    // Swap the art when the state changes
    if (isOn = !isOn)
    {
        [button setTitle:@"On" forState:UIControlStateNormal];
        [button setTitle:@"On" forState:UIControlStateHighlighted];
        [button setBackgroundImage:[UIImage imageNamed:@"green.png"]
            ➡stretchableImageWithLeftCapWidth:110.0 topCapHeight:0.0]
            ➡forState:UIControlStateNormal];
        [button setBackgroundImage:[UIImage imageNamed:@"green2.png"]
            ➡stretchableImageWithLeftCapWidth:110.0 topCapHeight:0.0]
            ➡forState:UIControlStateHighlighted];
    }
    else
    {
        [button setTitle:@"Off" forState:UIControlStateNormal];
        [button setTitle:@"Off" forState:UIControlStateHighlighted];
        [button setBackgroundImage:[UIImage imageNamed:@"red.png"]
            ➡stretchableImageWithLeftCapWidth:110.0 topCapHeight:0.0]
    }
}

```



```

        ➡forState:UIControlStateNormal];
        [button setBackgroundImage:[UIImage imageNamed:@"red2.png"]
        ➡stretchableImageWithLeftCapWidth:110.0 topCapHeight:0.0]
        ➡forState:UIControlStateHighlighted];
    }
}
- (void)loadView
{
    // Load an application image and set it as the primary view
    contentView = [[UIImageView alloc] initWithFrame:[UIScreen mainScreen]
    ➡applicationFrame]];
    [contentView setImage:[UIImage imageNamed:@"bluedots.png"]];
    [contentView setUserInteractionEnabled:YES];
    self.view = contentView;
    [contentView release];

    // Add the button
    UIButton *button = [self buildButton:@"On"];
    [button setCenter:CGPointMake(160.0f, 200.0f)];
    [contentView addSubview: button];
    [button addTarget:self action:@selector(toggleButton:)
    ➡forControlEvents: UIControlEventTouchUpInside];

    isOn = YES;
}
- (void) dealloc
{
    [contentView release];
    [super dealloc];
}
@end

```

8.1.3 玻璃按钮 (glass button)

在SDK Beta阶段，苹果公司取消了对简单UIGlassButton类的官方支持。这个类提供了比圆角矩形更漂亮的外观，并支持对控件进行着色。在编写本书时，UIGlassButton类依然存在于模拟器的UIKit框架中，不过苹果公司并没有在iPhone的UIKit框架中完全实现它。

8.2 秘诀：向按钮添加动画元素

你不能向按钮添加子视图，不过可以创造性地在按钮的前面或后面放置样式。使用标准UIView结构来达到此目的，要确保对任何视图禁用用户交互，否则会让按钮变得模糊难懂 (setUserInteractionEnabled:NO)。图8-2显示了当你把半透明按钮样式和它后面的动画UIImage结合起来后会发生什么。图像视图内容“泄漏”给观察者，使你能够向按钮添加动画元素。

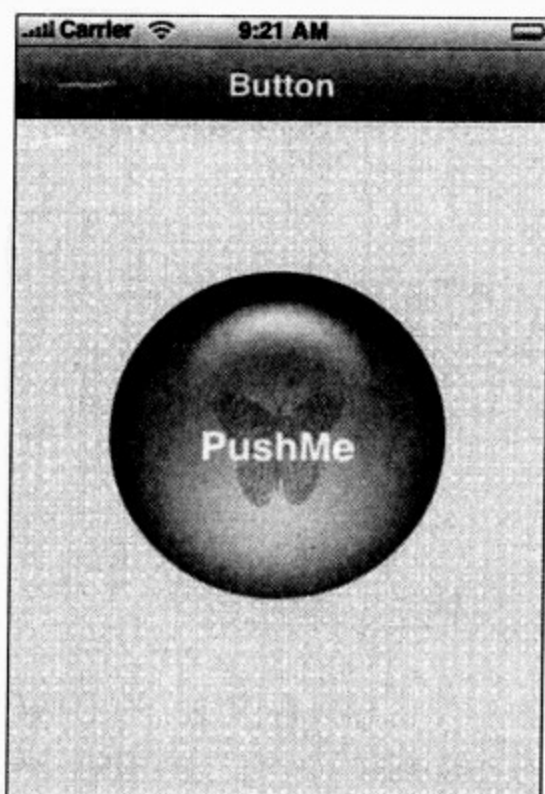


图8-2 将半透明的按钮样式和动画UIImageViews结合起来，以构建引人注目UI元素。在秘诀8-2中，蝴蝶在按钮“内部”拍动翅膀

秘诀8-2 在UIButton后面添加动画元素

```
- (void)loadView
{
    // Load an application image and set it as the primary view
    contentView = [[UIImageView alloc] initWithFrame:[UIScreen mainScreen]
    ➔applicationFrame]];
    [contentView setImage:[UIImage imageNamed:@"bluedots.png"]];
    [contentView setUserInteractionEnabled:YES];
    self.view = contentView;
    [contentView release];

    // load in the animation cells for the butterfly
    NSMutableArray *bflies = [[NSMutableArray alloc] init];
    for (int i = 1; i <= 17; i++) {
        NSString *cname = [NSString stringWithFormat:@"bf_%d.png", i];
        UIImage *img = [UIImage imageNamed:cname];
        if (img) [bflies addObject:img];
    }

    // create the image view and add it
    UIImageView *imageView = [[UIImageView alloc] initWithFrame:CGRectMake(0.0f,
    ➔0.0f, 80.0f, 80.0f)];
    [imageView setAnimationImages:bflies];
    [imageView setAnimationDuration:1.2f];
    [imageView startAnimating];
    [bflies release];
}
```

```

[contentView addSubview:imageView];
[imageView release];
[imageView setCenter:CGPointMake(160.0f, 200.0f)];
[imageView setUserInteractionEnabled:NO];

// Add the button and set it to play the sound
UIButton *button = [self buildButton:@"PushMe"];
[button setCenter:CGPointMake(160.0f, 200.0f)];
[contentView addSubview: button];
[button addTarget:self action:@selector(playSound:) forControlEvents:
    UIControlEventTouchDown];
}

```

8.3 秘诀：为按钮响应制作动画效果

UIControl除了边框和目标操作外还有更多内容。所有的控件都继承自UIView类。这就意味着当你处理控件时，可以像处理标准视图时一样使用UIView动画块。秘诀8-3构建了一个切换开关，它使用UIViewAnimationTransitionFlipFromLeft旋转按钮，同时更改状态。

与秘诀8-1不同，此代码不转换样式，而是转换按钮。共有两个按钮：一个开按钮和一个关按钮，它们都基于清晰的UIView。通过为这两个按钮提供一个透明的父视图，你能够仅为这些按钮应用翻转，而不包含用户界面的其余元素。忽略清晰的背景，并最终旋转整个窗口——这并不是好的UI选择。

由于此秘诀使用了和上一个秘诀相同的半透明样式，因此任何时候在屏幕上只出现一个按钮是很重要的。要达到此目的，在动画块（animation block）中时，当前按钮应该把自己从该清晰的超视图中移除。相反状态的按钮会取代它的位置。图8-3显示了翻转过程中的翻转按钮。

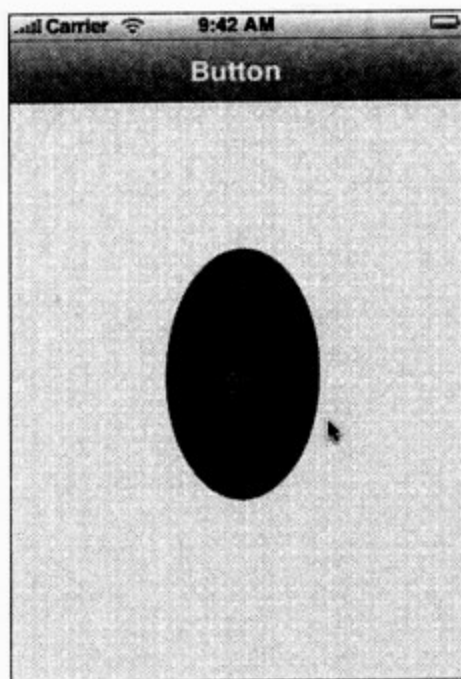


图8-3 使用UIView动画块在控件状态之间翻转。这里，按钮正在旋转，在关状态与开状态之间进行切换

秘诀8-3 将UIView动画块添加到控件

```

- (void) toggleButton: (UIButton *) button
{
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:1.0f];
    [UIView setAnimationTransition:
        UIViewAnimationTransitionFlipFromLeft forView:clearView cache:YES];

    // swap out the button views during the flip
    [button removeFromSuperview];
    if (isOn = !isOn) [clearView addSubview:onButton];
    else [clearView addSubview:offButton];

    [UIView commitAnimations];
}

```

8.4 秘诀：定制开关

重要的UIControl对象通常基于一系列子视图构建。通过导航某个控件的子视图树，你可以公开和定制通常不能从标准SDK中访问的对象。以开关为例，SDK只提供一种类型的开关：一个开/关按钮。要编辑该按钮使其转换为如图8-4所示（左图）的Yes/No按钮，只需要编写几行代码。

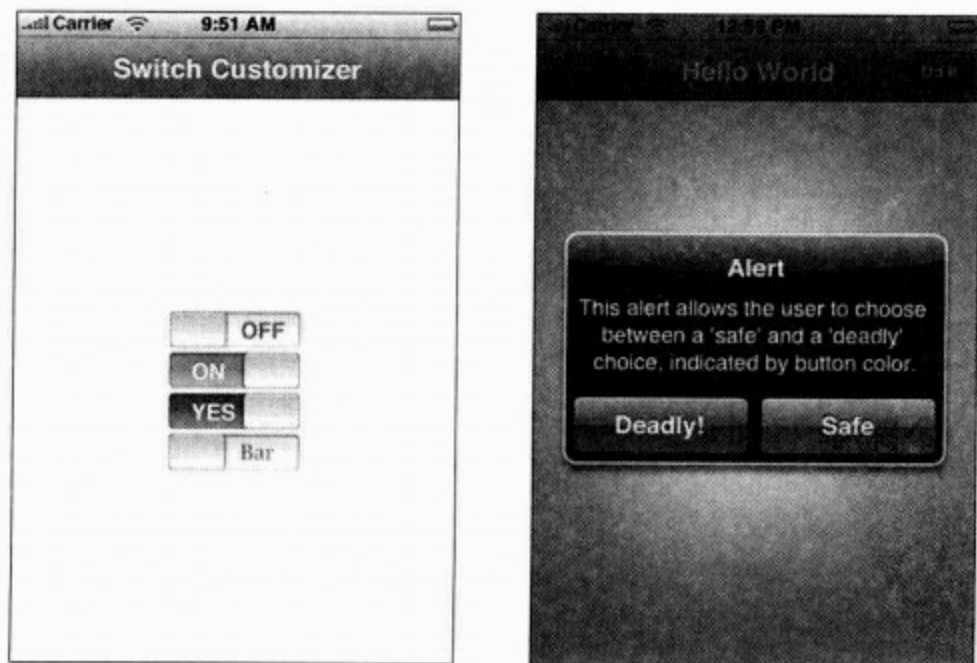


图8-4 （左图）通过向下导航对象（dumpster diving），你能够公开和自定义底层UISwitch视图。
（右图）通过访问警告的子视图来为UIAlertView按钮着色

这种定制依赖于对控件视图树的理解。代码清单8-3显示了如何探索该结构。向explode:level:发送一个控件对象，初始级别为0。该方法以递归的方式遍历树，依次呈现每个子视图。

代码清单8-3 探索UIControl视图结构

```

#define outstring(anObject) [[anObject description] UTF8String]

- (void) explode: (id) aView level: (int) aLevel
{
    // indent to show the current level
    for (int i = 0; i < aLevel; i++) printf("-", aLevel);

    // show the class and superclass for the current object
    printf("%s:%s\n", outstring([aView class]), outstring([aView superclass]));

    // recurse for all subviews
    for (UIView *subview in [aView subviews])
        [self explode:subview level:(aLevel + 1)];
}

```

发送开关时，explode会在Xcode控制台中创建以下结果。正如你所看到的，一个开关有一个子视图，即一个自定义UISwitchSlider对象。这里有4个子视图。3个图像用于创建背景和开关样式。一个视图拥有开关上使用的两个标签。因为这些标签是标准的UILabel实例，因此可以允许它们使用你提供的任何文本：

```

UISwitch:UIControl
--_UISwitchSlider:UISlider
----UIImageView:UIView
----UIImageView:UIView
----UIView:UIResponder
-----UILabel:UIView
-----UILabel:UIView
----UIImageView:UIView

```

知道如何设置视图后，就能够构建一个访问这些条目的UISwitch子类。秘诀8-4显示了添加标签文本定制的UICustomSwitch类实现。

说明 秘诀8-4中扩展的UISwitch类定义还显示了文档中未记录的setAlternateColors:调用，使你能够使用桔黄色背景而不是蓝色背景。有关在应用程序中使用文档中未记录的特性的讨论，请参见第1章。

秘诀8-4 向iPhone开关添加文本定制

```

// Add the undocumented alternate colors
@interface UISwitch (extended)
- (void) setAlternateColors:(BOOL) boolean;
@end

// Expose the _UISwitchSlider class
@interface _UISwitchSlider : UIView
@end

```



```
// UICustomSwitch allows you to set the left and right label text
@interface UICustomSwitch : UISwitch
- (void) setLeftLabelText: (NSString *) labelText;
- (void) setRightLabelText: (NSString *) labelText;
@end

@implementation UICustomSwitch
- (_UISwitchSlider *) slider {
    return [[self subviews] lastObject];
}
- (UIView *) textHolder {
    return [[[self slider] subviews] objectAtIndex:2];
}
- (UILabel *) leftLabel {
    return [[[self textHolder] subviews] objectAtIndex:0];
}
- (UILabel *) rightLabel {
    return [[[self textHolder] subviews] objectAtIndex:1];
}
- (void) setLeftLabelText: (NSString *) labelText {
    [[self leftLabel] setText:labelText];
}
- (void) setRightLabelText: (NSString *) labelText {
    [[self rightLabel] setText:labelText];
}
@end
```

定制 UIAlertView 按钮

通过访问子视图,你可以定制许多类。图8-4(右图)显示了一个带有自定义按钮的UIAlertView。被着色为红色时,它指示可以在UIActionSheet破坏性按钮中找到的相同类型的危险。这是危险信号。

着色过程是不一致的。因为UIAlertView是一个文档中未记录的按钮类,它不响应标准的tintColor调用。你必须为按钮设置背景颜色。这就意味着颜色会渗过按钮边缘。幸运的是,渗漏很轻微,只有挂饰(pendant)才需要避免此方法。这里的教训是使用非标准SDK类时要小心。它们中的许多类都可以使用drawRect:调用来呈现,但不允许对自身的完全定制。

代码清单8-4使用了willPresentAlertView:委托方法。此方法使警告能够在定制按钮之前完成自身的构建。警告视图和动作表(以及大多数栏类)不会完全地构建它们的组件视图,直到它们出现在屏幕上之前。等待使得子视图能够在被修改之前和在屏幕上被呈现给用户之前完成对自身的构建。

代码清单8-4 为UIActionSheet按钮着色

```
- (void)willPresentAlertView:(UIAlertView *)alertView
{
```



```

CGContextRef    context = NULL;
CGColorSpaceRef colorSpace;
void *          bitmapData;
int             bitmapByteCount;
int             bitmapBytesPerRow;

bitmapBytesPerRow = (pixelsWide * 4);
bitmapByteCount   = (bitmapBytesPerRow * pixelsHigh);

colorSpace = CGColorSpaceCreateDeviceRGB();
bitmapData = malloc( bitmapByteCount );
if (bitmapData == NULL)
{
    fprintf (stderr, "Memory not allocated!");
    return NULL;
}
context = CGBitmapContextCreate (bitmapData,
                                pixelsWide,
                                pixelsHigh,
                                8,
                                bitmapBytesPerRow,
                                colorSpace,
                                kCGImageAlphaPremultipliedLast);

if (context== NULL)
{
    free (bitmapData);
    fprintf (stderr, "Context not created!");
    return NULL;
}
CGColorSpaceRelease( colorSpace );

return context;
}

// Build a thumb image based on the grayscale percentage
id createImage(float percentage)
{
    CGRect aRect = CGRectMake(0.0f, 0.0f, 48.0f, 48.0f);
    CGContextRef context = MyCreateBitmapContext(48, 48);
    CGContextClearRect(context, aRect);

    // Outer gray circle
    CGContextSetFillColorWithColor(context, [[UIColor lightGrayColor] CGColor]);
    CGContextFillEllipseInRect(context, aRect);

    // Inner circle with feedback levels
    CGContextSetFillColorWithColor(context, [[UIColor colorWithRed:percentage
    ➤green:0.0f blue:0.0f alpha:1.0f] CGColor]);
    CGContextFillEllipseInRect(context, CGRectInset(aRect, 4.0f, 4.0f));

```

```

    // Inner gray circle
    CGContextSetFillColorWithColor(context, [[UIColor lightGrayColor] CGColor]);
    CGContextFillEllipseInRect(context, CGRectInset(aRect, 16.0f, 16.0f));

    CGImageRef myRef = CGBitmapContextCreateImage(context);
    free(CGBitmapContextGetData(context));
    CGContextRelease(context);

    return [UIImage imageWithCGImage:myRef];
}

@interface HelloController : UIViewController
{
    UIImageView *contentView;
    UILabel *valueLabel;
    UIImage *knob;
}
@end

@implementation HelloController
- (void) updateValue: (UISlider *) slider
{
    [valueLabel setText:[NSString stringWithFormat:@"%3.1f", [slider value]]];

    CGImageRef oldknobref = [knob CGImage];
    UIImage *knob = createImage([slider value] / 100.0f);
    [slider setThumbImage:knob forState:UIControlStateNormal];
    [slider setThumbImage:knob forState:UIControlStateHighlighted];
    CFRelease(oldknobref);
}

- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen]
    ➡applicationFrame]];
    [contentView setImage:[UIImage imageNamed:@"bluedots.png"]];
    [contentView setUserInteractionEnabled:YES];
    self.view = contentView;
    [contentView release];

    // Add a text label to show the current slider scale
    valueLabel = [[UILabel alloc] initWithFrame:CGRectMake(50.0f, 100.0f, 220.0f,
    ➡40.0f)];
    valueLabel.textAlignment = UITextAlignmentCenter;
    valueLabel.text = @"50.0";
    valueLabel.backgroundColor = [UIColor clearColor];
    [contentView addSubview:valueLabel];
    [valueLabel release];

    // Build the slider

```



```

UISlider *slider = [[UISlider alloc] initWithFrame:CGRectMake(50.0f, 160.0f,
    ➤220.0f, 40.0f)];
slider.backgroundColor = [UIColor clearColor];
slider.minimumValue = 0.0f;
slider.maximumValue = 100.0f;
slider.continuous = YES;
slider.value = 50.0f;

// Add the custom knob
knob = createImage(0.5);
[slider setThumbImage:knob forState:UIControlStateNormal];
[slider setThumbImage:knob forState:UIControlStateHighlighted];

[slider addTarget:self action:@selector(updateValue:)
    ➤forControlEvents:UIControlEventValueChanged];

[contentView addSubview: slider];
[slider release];
}
@end

```

向滑块添加文本

不幸的是，添加文本的代价相对来说比较高。可以向滑块添加文本，如代码清单8-5所示，不过对于普通的交互都需要付出大量成本。此代码清单展示了如何将标签放入到核心图形上下文中。最好预先计算这些小图像，并从那些预先计算的图像中加载它们，而不是立即生成它们。虽然此例程在Macintosh上模拟得很好，但对于处理那些文本处理给实时图像创建带来的额外负担来说，iPhone的芯片有些功率不足。相比之下，预先计算的图像比较便宜，无论是在计算方面还是在内存存储量方面。

记住，Quartz坐标系从左下角开始，而不是左上角。将文本添加到图像中的一个点使用了Quartz系统。

代码清单8-5 添加略缩文本

```

id createImage(float percentage)
{
    CGRect aRect = CGRectMake(0.0f, 0.0f, 48.0f, 48.0f);
    CGContextRef context = MyCreateBitmapContext(48, 48);
    CGContextClearRect(context, aRect);

    CGContextSetFillColorWithColor(context, [[UIColor lightGrayColor] CGColor]);
    CGContextFillEllipseInRect(context, aRect);

    CGContextSetFillColorWithColor(context, [[UIColor colorWithRed:percentage
    ➤green:0.0f blue:0.0f alpha:1.0f] CGColor]);
    CGContextFillEllipseInRect(context, CGRectInset(aRect, 4.0f, 4.0f));

    CGContextSetFillColorWithColor(context, [[UIColor whiteColor] CGColor]);

```

```

CGContextSelectFont(context, "Georgia", 14.0f, kCGEncodingMacRoman);
CGContextSetTextDrawingMode(context, kCGTextFill);
CGContextSetShouldAntialias(context, true);
NSString *outString = [NSString stringWithFormat:@"%4.2f", percentage];
CGContextShowTextAtPoint(context, 10, 20, [outString UTF8String], [outString
    ➔length]);

CGImageRef myRef = CGBitmapContextCreateImage(context);
free(CGBitmapContextGetData(context));
CGContextRelease(context);

return [UIImage imageWithCGImage:myRef];
}

```

8.6 秘诀：关闭 UITextField 键盘

有关UITextField控件最常问到的一个问题是“如何关闭键盘？”没有内置方法可以自动解决这个问题。当用户结束对UITextField内容的编辑时，键盘就应该自动消失。

所幸，响应编辑结束还需要一些工作。通过监视Return键，可以放弃首要响应者状态。这会把键盘移出视线之外，如秘诀8-6所示。下面是有关上述操作的一些要点。



图8-6 捕获按下Return键的动作，以了解你的用户什么时候结束编辑一个文本字段，这样你就可以关闭键盘

- ❑ 将Return键类型设置为UIReturnKeyDone。使用一个“Done”类型的Return键告知用户如何完成编辑。图8-6显示了一个使用Done键类型的键盘。
- ❑ 实现textFieldShouldReturn:。此方法捕获所有按下回车键的动作——不管它们是如何被命名的。使用此方法放弃首要响应者。这会隐藏键盘，直到用户触摸另一个文本字段或文本视图。

□ 确保你的视图控制器实现了UITextFieldDelegate协议，并确保将字段委托设置为指向控制器。

代码需要处理这些要点，以便为UITextField实例创建平滑的交互过程。图8-6实际地显示了UITextField控件，以及字段编辑过程中位于适当位置的键盘。

秘诀8-6 使用Done键关闭文本字段键盘

```
// This method catches the Return action
- (BOOL)textFieldShouldReturn:(UITextField *)textField
{
    [textField resignFirstResponder];
    return YES;
}

- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:[UIScreen mainScreen]
    ➡applicationFrame];
    contentView.backgroundColor = [UIColor whiteColor];
    self.view = contentView;
    [contentView release];

    // Create a field with a Done return key
    namefield = [[[UITextField alloc] initWithFrame:CGRectMake(120.0f, 40.0f,
    ➡150.0f, 30.0f)] retain];
    [namefield setBorderStyle:UITextBorderStyleRoundedRect];

    namefield.placeholder = @"name";
    namefield.returnKeyType = UIReturnKeyDone;
    namefield.clearButtonMode = UITextFieldViewModeWhileEditing;
    namefield.delegate = self;
    [contentView addSubview:namefield];
    [namefield release];
}
```

8.7 秘诀：关闭 UITextView 键盘

当关闭键盘时，UITextView实例需要的方法和UITextField实例稍微有些不同。用户应该能够在文本视图中点击Return，在不关闭键盘的情况下添加回车。当文本视图有效时，向普通界面添加一个Done按钮，如图8-7所示。当用户完成他或她的编辑时，可以使用这个键放弃首要响应者状态。

为了理解文本视图的活动，视图控制器必须实现UITextViewDelegate协议，它必须被设置为文本视图的委托。当用户点击视图时，就触发textViewDidBeginEditing:委托方法。通过查明此问题，你能够添加或启用Done按钮。然后，用户可以在完成编辑之后点击Done按钮。Done按钮为完成编辑和关闭键盘提供了一个明了的方法。

秘诀8-7展示了如何在委托方法调用中添加导航条目按钮，以及当用户完成编辑时如何移除它。当视图变为有效时就显示Done按钮，当放弃视图的首要响应者状态时就隐藏它。

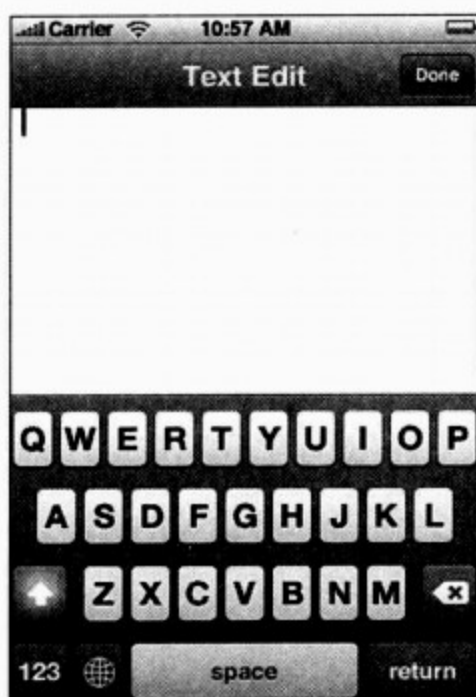


图8-7 当用户开始与文本视图进行交互时，就向导航栏添加一个Done键。这为用户完成编辑和关闭键盘提供了一个明了的方法

秘诀8-7 向有效的UITextView会话添加一个Done按钮

```
@interface HelloController : UIViewController <UITextViewDelegate>
{
    UITextView *contentView;
}
@end

@implementation HelloController

// Reveal a done button when editing starts
- (void) textViewDidBeginEditing: (UITextView *) textView
{
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                                initWithTitle:@"Done"
                                                style:UIBarButtonItemStyleDone
                                                target:self
                                                action:@selector(doneEditing:)]
                                                autorelease];
}

// The done button forces the text view to resign its first-responder status
- (void) doneEditing: (id) sender
{
    [contentView resignFirstResponder];
    self.navigationItem.rightBarButtonItem = NULL;
}

- (void)loadView
{
    // Load an application image and set it as the primary view
}
```



```

    contentView = [[UITextView alloc] initWithFrame:[UIScreen mainScreen]
    ➔applicationFrame]];
    [contentView setDelegate:self];
    self.view = contentView;
    [contentView release];
}

-(void) dealloc
{
    [contentView release];
    [super dealloc];
}
@end

```

8.8 秘诀：向文本视图添加一个撤销（Undo）按钮

秘诀8-7显示了如何在一个文本视图内部捕获用户交互。UITextViewDelegate协议也可以理解用户对显示文本所做的更改。当用户在视图中键入或删除文本时，就会触发textView:shouldChangeTextInRange: replacementText:方法。通过返回YES并显示一个撤销按钮，可以创建支持恢复到最近保存状态的基本结构。

秘诀8-8向秘诀8-7中构建的基本结构添加了持久性数据。现在，当用户点击Done按钮以取消键盘时，应用程序会把文本的一个副本保存到一个文件中。在编辑过程中，如果用户点击了Undo按钮，程序会用该文件的内容代替文本视图的内容，并重新隐藏Undo按钮。

这些更改确保了只有当可见文本相比存储文本有了变化时才显示Undo按钮。图8-8显示了使用此秘诀构建的界面，即用户文本更改触发Undo按钮时的界面。

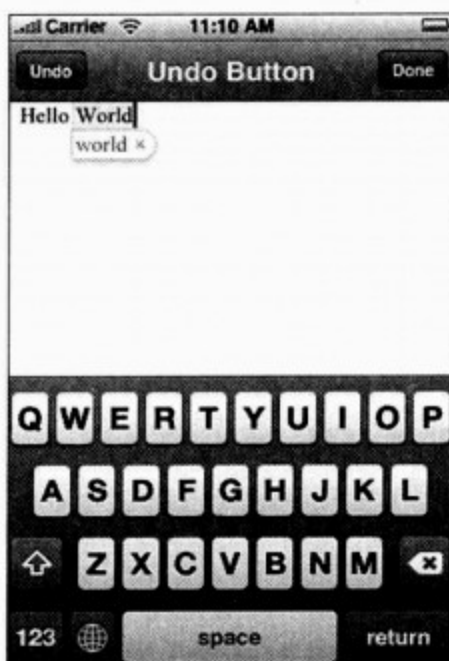


图8-8 屏幕左上角的Undo按钮只有在用户更改了文本视图中最近存储状态下的文本时才出现。点击Done按钮会把所有更改保存到文件中（并隐藏Undo按钮）。点击Undo按钮会把视图内容恢复到最近存储的状态

秘訣8-8 向文本视图添加撤销支持

```

@interface HelloController : UIViewController <UITextViewDelegate>
{
    UITextView *contentView;
    BOOL readyToUndo;
}
@end

@implementation HelloController

#define FILEPATH [NSHomeDirectory()
➡stringByAppendingPathComponent:@"Documents/MyText.txt"]
// Reveal a Done button when starting to edit
- (void) textViewDidBeginEditing: (UITextView *) textView
{
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                                initWithTitle:@"Done"
                                                style:UIBarButtonItemStyleDone
                                                target:self
                                                action:@selector(doneEditing:)]
                                                autorelease];

    readyToUndo = YES;
}

// Only reveal the undo after text has actually been typed or deleted
- (BOOL)textView:(UITextView *)textView shouldChangeTextInRange:(NSRange)range
➡replacementText:(NSString *)text
{
    if (readyToUndo)
        self.navigationItem.leftBarButtonItem = [[[UIBarButtonItem alloc]
                                                    initWithTitle:@"Undo"
                                                    style:UIBarButtonItemStylePlain
                                                    target:self
                                                    action:@selector(undoEditing:)]
                                                    autorelease];

    readyToUndo = NO;
    return YES;
}

// The Done button forces the text view to resign its first-responder status and
➡saves changes
- (void) doneEditing: (id) sender
{
    NSError *error;

    [contentView resignFirstResponder];
    self.navigationItem.rightBarButtonItem = NULL;
    self.navigationItem.leftBarButtonItem = NULL;
    [[contentView text] writeToFile:FILEPATH atomically: YES
    ➡encoding:NSUTF8StringEncoding error: &error];
}

```

```
// Undo means revert to last saved state
- (void) undoEditing: (id) sender
{
    if ([[NSFileManager defaultManager] fileExistsAtPath:FILEPATH])
        [contentView setText:[NSString stringWithContentsOfFile:FILEPATH]];
    else
        [contentView setText:@""];

    readyToUndo = YES;
    self.navigationItem.leftBarButtonItem = NULL;
}

- (void)loadView
{
    // Load an application image and set it as the primary view
    contentView = [[UITextView alloc] initWithFrame:[UIScreen mainScreen]
        applicationFrame];
    [contentView setDelegate:self];
    self.view = contentView;
    [contentView release];

    // Preload with text if any has been saved
    if ([[NSFileManager defaultManager] fileExistsAtPath:FILEPATH])
        [contentView setText:[NSString stringWithContentsOfFile:FILEPATH]];
}
@end
```

8.9 秘诀：创建一个基于文本视图的 HTML 编辑器

虽然苹果公司正式从UITextView类中删除了HTML支持，但它仍然隐藏在UIKit框架中。当希望添加简单的富文本扩展时，可以访问这项文档中未记录的特性。显然，苹果公司希望你使用UIWebView而不是UITextView来进行HTML显示，不过UITextView提供了更吸引人的特性。

要访问此HTML显示，需要声明setContentToHTMLString:方法，如秘诀8-9中所示。这个文档中未记录的UITextView方法告知文本视图将一个字符串解释为HTML源。使用UITextView有以下两个好处。

- ❑ UITextView可以被编辑。你可以使用HTML文本初始化视图，并允许用户编辑结果。文本会选择周围元素的属性作为自己的属性。例如，如果向一个粗体标题行添加文本，新的文本也是粗体的。
- ❑ 第二，UITextView可以很容易地被重新加载。例如，如果你允许用户在基于文本的源的模式中编辑文本，可以在HTML模式中把这些更改重新加载到相同的视图中。

秘诀8-9显示了如何构建一个简单的HTML编辑器。图8-9显示了相同的UITextView中的文本源和HTML表示。这并不是技术的通常用法（或实际用法），不过它突出了一项以后会用到的功能。通常，你只希望用一些富文本功能（如粗体的标题行）来初始化文本，然后允许用户直接编

辑该文本。

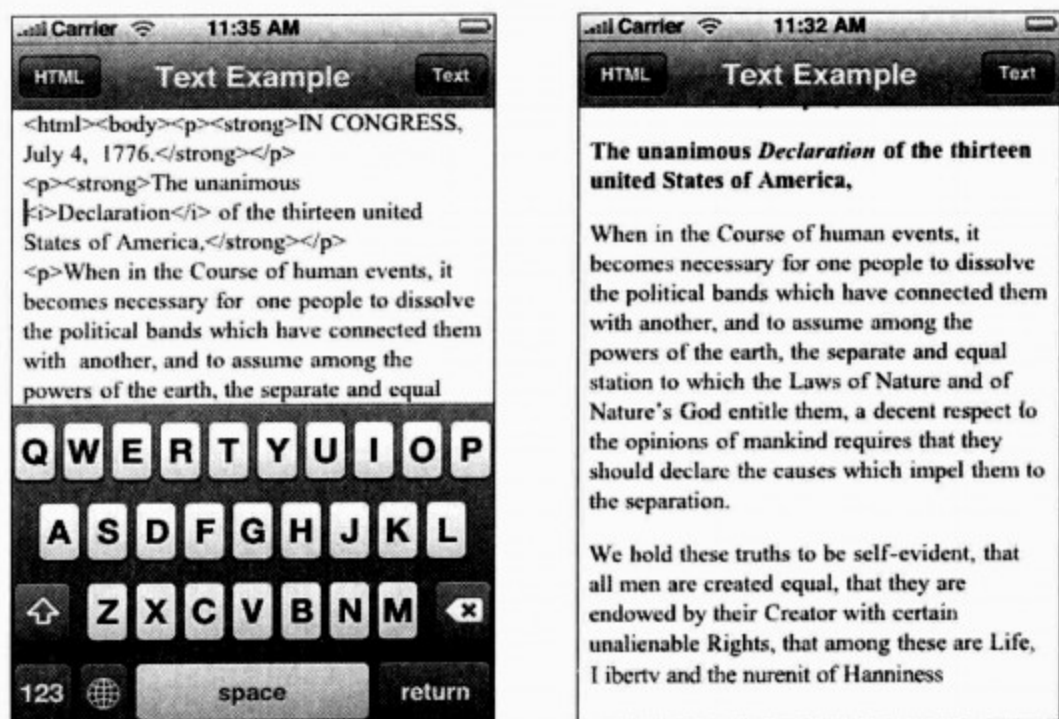


图8-9 这个基于UITextView的HTML编辑器允许用户在同一视图中编辑源和查看结果

说明 可以使用NSLocalizedString使文本国际化。向它传递来自语言文件夹的Localizable.strings文件（例如，从French.lproj或English.lproj文件）的任意键。NSLocalizedString (@“Title”, nil)在strings文件中查找Title键，并返回与该键匹配的本地化字符串。通过常规国际设置，iPhone知道使用哪种语言，也知道要访问哪个lproj文件夹。为了使应用程序名国际化，需要向每个lproj文件夹添加一个InfoPlist.strings文件。在这些文件中设置CFBundleDisplayName键。SDK发布之后，有关最新的详细信息可以参考苹果公司的文档。

秘诀8-9 使用文档中未记录的UITextView HTML特性

```
- (void) presentText: (id) sender
{
    // dismiss keyboard if it exists
    [contentView resignFirstResponder];
    // Store text only if previous editor was text-based
    if ([contentView tag]) self.sourceText = [contentView text];

    // Text-based Editor
    [contentView setText:self.sourceText];
    [contentView setEditable:YES];
    [contentView setTag:YES];
}

- (void) presentHTML: (id) sender
{

```



```

// dismiss keyboard if it exists
[contentView resignFirstResponder];

// Store text only if previous editor was text-based
if ([contentView tag]) self.sourceText = [contentView text];

// HTML-based Presentation-only
[contentView setContentToHTMLString:self.sourceText];
[contentView setEditable:NO];
[contentView setTag:NO];
}

- (void)loadView
{
    // Load an application image and set it as the primary view
    contentView = [[UITextView alloc] initWithFrame:[UIScreen mainScreen]
        applicationFrame];
    [contentView setAutocorrectionType:UITextAutocorrectionTypeNo];
    [contentView setTag:NO];
    [contentView setEditable:NO];
    self.view = contentView;
    [contentView release];

    // Load the HTML source
    self.sourceText = [NSString stringWithContentsOfFile:[NSBundle mainBundle]
        pathForResource:@"decl" ofType:@"html"];
    [contentView setContentToHTMLString:self.sourceText];

    // Add Text and HTML buttons
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
        initWithTitle:@"Text"
        style:UIBarButtonItemStylePlain
        target:self
        action:@selector(presentText:)]
        autorelease];

    self.navigationItem.leftBarButtonItem = [[[UIBarButtonItem alloc]
        initWithTitle:@"HTML"
        style:UIBarButtonItemStylePlain
        target:self
        action:@selector(presentHTML:)]
        autorelease];
}

```

8.10 秘诀：构建一个交互搜索栏

UISearchBar类提供了一个简单的搜索主题文本字段，它显示在应用程序的导航栏中正好合适。图8-10实际地显示了一个搜索栏，在颜色画笔名中搜索匹配的项。搜索是实时的。当用户键

入文本时，下面的列表会自动更新。

秘诀8-10使用searchbar:textDidChange:方法捕获搜索栏文本更改。在用户键入新字符时，此方法重新构建用于填充屏幕列表的数组。它使搜索字段的文本与数组中的条目相匹配。

注意图8-10中使用的Return键换出。通常，搜索栏会延迟搜索，直到用户键入了一个词语。这就是它们为默认的Return键显示Search的原因。秘诀8-10的实时交互不使用上述搜索模型。取消键盘后，用户通过Done键可以检验他或她已经执行的当前搜索的结果。

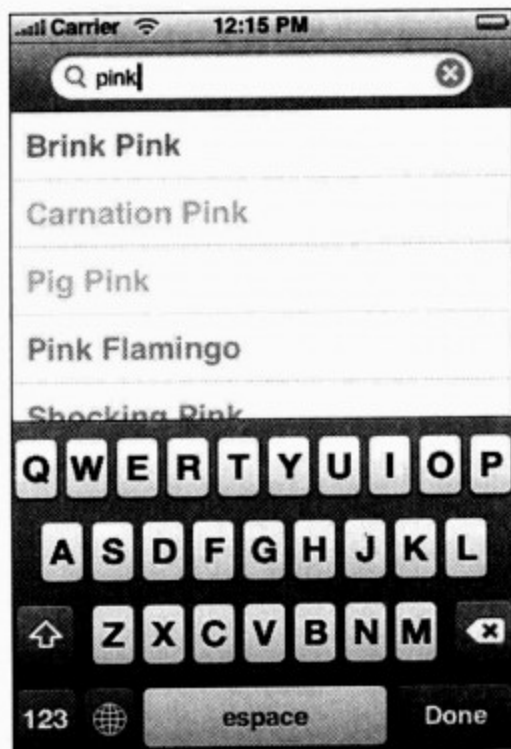


图8-10 搜索栏提供了出色的交互控件，用于在冗长的列表中放置条目

要代替Return键，需要访问UISearchBar实例的搜索字段子视图，并为该字段设置文本特性。这时，文本视图是最前端（即最后一个）视图。如果在以后SDK发布中更改了此内容，应对应地调整秘诀代码。

秘诀8-10 使用带有实时反馈的搜索栏

```
// create an array by matching to the search string
- (void) buildSearchArrayFrom: (NSString *) matchString
{
    NSString *upString = [[matchString uppercaseString] autorelease];
    if (searchArray) [searchArray release];

    searchArray = [[NSMutableArray alloc] init];
    for (NSString *word in colorArray)
    {
        if ([matchString length] == 0)
        {
            [searchArray addObject:word];
            continue;
        }
    }
}
```

```

        NSRange range = [[[word componentsSeparatedByString:@" #"]
        ➡objectAtIndex:0] uppercaseString] rangeOfString:@"upString"];
        if (range.location != NSNotFound) [searchArray addObject:word];
    }

    [self.tableView reloadData];
}

// When the search text changes, update the array
- (void)searchBar:(UISearchBar *)searchBar textDidChange:(NSString *)searchText
{
    [self buildSearchArrayFrom:searchText];
    if ([searchText length] == 0) [searchBar resignFirstResponder];
}

// When the search ("done") button is clicked, hide the keyboard
- (void)searchBarSearchButtonClicked:(UISearchBar *)searchBar
{
    [searchBar resignFirstResponder];
}

// Prepare the Table View
- (void)loadView
{
    [super loadView];

    // Retrieve the text and colors from file
    NSString *pathname = [[NSBundle mainBundle] pathForResource:@"crayons"
    ➡ ofType:@"txt" inDirectory:@""];
    NSString *wordstring = [NSString stringWithContentsOfFile:pathname];
    colorArray = [[wordstring componentsSeparatedByString:@"\n"] retain];
    [self buildSearchArrayFrom:@""];

    search = [[UISearchBar alloc] initWithFrame:CGRectMake(0.0f, 0.0f, 280.0f,
    ➡ 44.0f)];
    search.delegate = self;
    search.placeholder = @"color match text";
    search.autocorrectionType = UITextAutocorrectionTypeNo;
    search.autocapitalizationType = UITextAutocapitalizationTypeNone;
    self.navigationItem.titleView = search;
    [search release];

    // "Search" is the wrong key usage here. Replacing it with "Done"
    UITextField *searchField = [[search subviews] lastObject];
    [searchField setReturnKeyType:UIReturnKeyDone];
}

```

8.11 秘诀：添加标注（callout）视图

如果你在iPhone上使用过Google Map，你可能在实战中见过UICalloutView实例。虽然它们

的名称如此，但是它们是一种UIControl实例。它们是文档中未记录的，但在UIKit框架中可用。

标注视图指向屏幕上的某些内容。它们在使用附加的扩展按钮移动到另一个消息之前，可以显示一个临时消息。图8-11显示了带有几个标注视图的屏幕，其中有些标注视图显示它们的源（临时）消息，其他标注视图显示最终消息。秘诀8-11提供了构建此屏幕的代码。

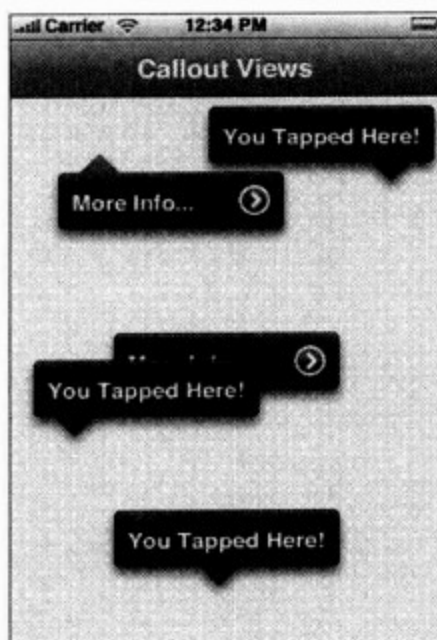


图8-11 UICalloutView实例指向某些内容。它们可以显示一个初始消息（这里是“**You Tapped Here!**”）和一个最终消息（这里是“**More Info...**”），以及一个附加的扩展按钮。苹果公司的Google Map应用程序中使用了标注

由于类是文档中未记录的，因此需要定义头文件。代码清单8-6显示了UICalloutView类定义。它还不完整，不过它提供了足够的功能使你能够在程序中有效地使用这个类。

代码清单8-6 定义UICalloutView界面

```
@class UIImageView, UILabel, UIButton;

@interface UICalloutView : UIControl
- (UICalloutView *)initWithFrame:(struct CGRect)aFrame;
- (void)fadeOutWithDuration:(float)duration;
- (void)setTemporaryTitle:(NSString *)fp8;
- (NSString *)temporaryTitle;
- (void)setTitle:(NSString *)fp8;
- (NSString *)title;
- (void)addTarget:(id)target action:(SEL)selector;
- (void)removeTarget:(id)target;
- (void)setAnchorPoint:(struct CGPoint)aPoint boundaryRect:(struct CGRect)aRect
  ➤animate:(BOOL)yorn;
@end
```

要创建一个标注，需要传递一个定位点（即标注指向的坐标）和一个广义（generalized）边界矩形。据我所知，这些限制完全被忽略了。

标注使用两个标题：一个临时标题和一个普通标题。当添加一个临时标题时，此标题显示大

约一秒钟的时间, 然后会变为普通标题, 即带有扩展按钮的标题。还可以向标注添加子标题, 不过那样看起来很可怕, 这里不推荐使用。

通过调用 `addTarget: action:` 方法, 为扩展按钮设置目标。当扩展按钮被点击时, 就调用传递的选择程序。这里不为标注上的用户点击计数, iPhone 忽略了它们。因此, 如果希望在出现一个新标注时除去现有标注, 应该由你来消除它。可以快速地使用 `fadeOutWithDuration:` 方法并指定渐没时间。

秘诀8-11 使用UICalloutView

```
@interface TapView : UIImageView
@end

@implementation TapView
- (void) hideDisclosure: (UIButton *) calloutButton
{
    UICalloutView *callout = (UICalloutView *) [calloutButton superview];
    [callout fadeOutWithDuration:1.0f];
}

- (UICalloutView *) buildDisclosure
{
    UICalloutView *callout = [[[UICalloutView alloc] initWithFrame:CGRectZero]
    ➤autorelease];
    callout.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
    callout.contentHorizontalAlignment =
    ➤UIControlContentHorizontalAlignmentCenter;

    [callout setTemporaryTitle:@"You Tapped Here!"];
    [callout setTitle:@"More Info..."];
    [callout addTarget:self action:@selector(hideDisclosure)];

    return callout;
}

- (void) showDisclosureAt:(CGPoint) aPoint
{
    UICalloutView *callout = [self buildDisclosure];
    [self addSubview:callout];
    [callout setAnchorPoint:aPoint boundaryRect:CGRectMake(0.0f, 0.0f, 320.0f,
    ➤100.0f) animate:YES];
}

- (void) touchesBegan:(NSSet*)touches withEvent:(UIEvent*)event
{
    [self showDisclosureAt:[touches anyObject] locationInView:self];
}

@end
```

8.12 添加一个页面指示器控件

令人伤心的是，UIPageControl类并不能真正满足我们的期望。它和在SpringBoard中看到的出色类不一样。它们看起来相同，但执行起来不同。官方SDK中可用的UIPageControl类很难使用，通常会给用户带来许多烦恼。因此，在使用它时，要确保添加了可选择的导航选项（比如第2章中介绍的滑动），以便页面控件更像一个指示器，而不是一个控件。

图8-12实际地显示了一个页面控件。点击当前颜色鲜艳的页面指示器的左边或右边，会触发UIControlEventValueChanged事件，并启动设置为控件动作的任何方法。你可以通过调用currentPage查询控件的新值，并通过调整numberOfPage属性设置可用的页面数。

秘诀8-12显示了图像的5个独特页面，不过实际上，它只使用了两个图像视图。在任意给定的时间上，显示一个视图，另一个视图准备加载下一个图像——无论在数值上该图像在当前图像之前还是之后。当用户移动到一个新页面时，视图控制器将新的图像加载到未使用的图像视图中，并调用UIViewAnimation块以便在两个图像视图之间进行交换。两个“页面”之间的数值关系确定了交换会把图像从右边推到左边还是从左边推到右边。

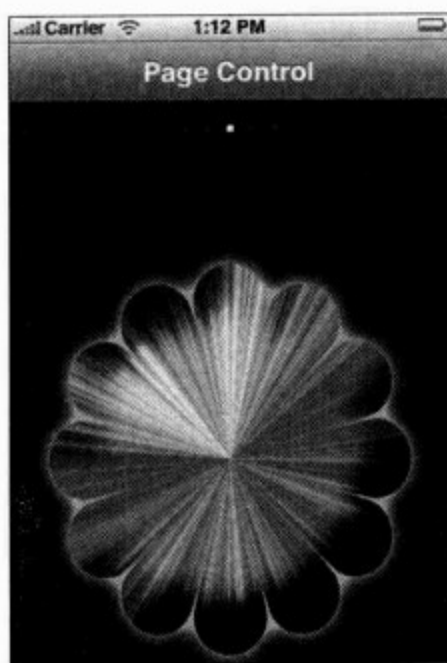


图8-12 UIPageControl类提供了一个用于多页面表示的交互指示器。通过点击活动点的左边或右边，用户能够选择新的页面。至少理论上是这样。页面控件很难点击，而且其响应性能也很差

秘诀8-12 使用UIPageControl指示器

```
@interface HelloController : UIViewController
{
    SwipeView *contentView;
    int currentPage;
}
@end

@implementation HelloController
```

```

- (id)init
{
    if (!(self = [super init])) return self;
    self.title = @"Page Control";
    return self;
}

// Build and return an animation with the given direction
- (CATransition *) getAnimation:(NSString *) direction
{
    CATransition *animation = [CATransition animation];
    [animation setDelegate:self];
    [animation setType:kCATransitionPush];
    [animation setSubtype:direction];
    [animation setDuration:1.0f];
    [animation setTimingFunction:[CAMediaTimingFunction
    ➤functionWithName:kCAMediaTimingFunctionEaseInEaseOut]];
    return animation;
}

// Perform the page turn animation, in this case a push
- (void) pageTurn: (UIPageControl *) pageControl
{
    CATransition *transition;
    int secondPage = [pageControl currentPage];
    if ((secondPage - currentPage) > 0)
        transition = [self getAnimation:@"fromRight"];
    else
        transition = [self getAnimation:@"fromLeft"];
    UIImageView *newView = (UIImageView *) [[contentView subviews]
    ➤objectAtIndex:0];
    [newView setImage:[UIImage imageNamed:[NSString stringWithFormat:@"%d.png",
    ➤secondPage + 1]]];
    [contentView exchangeSubviewAtIndex:0 withSubviewAtIndex:1];
    [[contentView layer] addAnimation:transition
    ➤forKey:@"transitionViewAnimation"];

    currentPage = [pageControl currentPage];
}

- (void)loadView
{
    UIView *baseView = [[[UIView alloc] initWithFrame:[UIScreen mainScreen]
    ➤applicationFrame]] autorelease];

    // Add the content view with its two images
    contentView = [[[SwipeView alloc] initWithFrame:[UIScreen mainScreen]
    ➤applicationFrame]] autorelease];
    [contentView setHost:self];

```

```

[contentView addSubview:[[UIImageView alloc] initWithFrame:[UIScreen
➤ mainScreen] applicationFrame]]];
[contentView addSubview:[[UIImageView alloc] initWithFrame:[UIScreen
➤ mainScreen] applicationFrame]]];
for (UIView *view in [contentView subviews]) [view
➤ setUserInteractionEnabled:NO];

// initialize with "page" 3
[[[contentView subviews] lastObject] setImage:[UIImage imageNamed:@"f3.png"]];

[baseView addSubview:contentView];

// Add the page control
UIPageControl *pageControl = [[UIPageControl alloc]
➤ initWithFrame:CGRectMake(0.0f, 10.0f, 320.0f, 20.0f)];
[pageControl setNumberOfPages:5];
[pageControl setCurrentPage:(currentPage = 2)];
[pageControl addTarget:self action:@selector(pageTurn:)
➤ forControlEvents:UIControlEventValueChanged];
[baseView addSubview:pageControl];
self.view = baseView;
}

-(void) dealloc
{
    [contentView release];
    [super dealloc];
}
@end

```

8.13 秘诀：定制工具栏

UITabBar类拥有对定制的官方支持，而UIToolBar类没有。你可以把视图控制器从它的More屏幕拖到或拖出标签栏（tab bar）。而对于工具栏，你不能这么做。至少不能正式地这么做。

非正式情况下，UIToolbar提供了一个相似的（不过是文档中未记录的）交互类型。虽然不能添加新的条目或换出条目，但可以对工具栏重新排序。它并不是非常灵活的交互类型，不过你可以使用它让用户定制工具栏的显示外观或让用户重新排列Scrabble信件。用法由你来决定。和公共框架中的所有未记录的调用一样，你可以在程序中自由地使用它们，不过它们会在不预先通知的情况下做出更改。图8-13实际地显示了工具栏定制屏幕。

工具栏中的所有对象都出现在定制屏幕中。也就是说，如果你添加固定或可变空间，它们会像图标一样在那里显示。必须为完成编辑工具栏添加一个外部方法，因为一旦进入定制模式，触发的动作在工具栏按钮上就不起作用了。这正是屏幕右上角Done按钮的任务。

秘诀8-13显示了如何添加工具栏定制。它扩展了UIToolbar类，以显示文档中未记录的开始和结束定制方法。

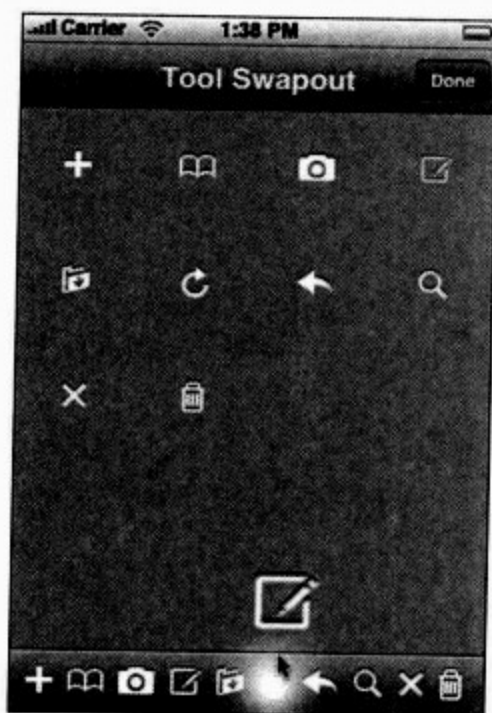


图8-13 文档中未记录的UIToolbar定制屏幕允许用户对工具栏条目重新排列

秘诀8-13 定制工具栏

```
// Expose the customizing calls
@interface UIToolbar (extended)
- (void)beginCustomizingItems:(id)fp8;
- (BOOL)endCustomizingAnimated:(BOOL)fp8;
@end;

@interface HelloController : UIViewController
{
    UIImageView *contentView;
    UIToolbar *toolbar;
    NSMutableArray *allitems;
}
@end

@implementation HelloController

// When finished customizing, remove the Done button
- (void) endEditing
{
    [toolbar endCustomizingAnimated:YES];
    self.navigationItem.rightBarButtonItem = NULL;
}

// Reveal the Done button and begin customizing
- (void) doAction: (id) sender
{
    [toolbar beginCustomizingItems:allitems];
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
```

```

initWithTitle:@"Done"
style:UIBarButtonItemStylePlain
target:self
action:@selector(endEditing)]
autorelease];

}

- (void)loadView
{
    // Load an application image and set it as the primary view
    contentView = [[UIView alloc] initWithFrame:[UIScreen mainScreen]
    ➤applicationFrame]];
    [contentView setImage:[UIImage imageNamed:@"bluedots.png"]];
    [contentView setUserInteractionEnabled:YES];
    self.view = contentView;
    [contentView release];

    toolbar = [[UIToolbar alloc] initWithFrame:CGRectMake(0.0f, 416.0f - 44.0f,
    ➤320.0f, 44.0f)];

    // Add a whole bunch of arbitrary system items with icons
    allitems = [[NSMutableArray alloc] init];
    [allitems addObject:[UIBarButtonItem alloc]
    ➤initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
    ➤target:self action:@selector(doAction:)] autorelease];
    [allitems addObject:[UIBarButtonItem alloc]
    ➤initWithBarButtonSystemItem:UIBarButtonSystemItemBookmarks
    ➤target:self action:@selector(doAction:)] autorelease];
    [allitems addObject:[UIBarButtonItem alloc]
    ➤initWithBarButtonSystemItem:UIBarButtonSystemItemCamera
    ➤target:self action:@selector(doAction:)] autorelease];
    [allitems addObject:[UIBarButtonItem alloc]
    ➤initWithBarButtonSystemItem:UIBarButtonSystemItemCompose
    ➤target:self action:@selector(doAction:)] autorelease];
    [allitems addObject:[UIBarButtonItem alloc]
    ➤initWithBarButtonSystemItem:UIBarButtonSystemItemOrganize
    ➤target:self action:@selector(doAction:)] autorelease];
    [allitems addObject:[UIBarButtonItem alloc]
    ➤initWithBarButtonSystemItem:UIBarButtonSystemItemRefresh
    ➤target:self action:@selector(doAction:)] autorelease];
    [allitems addObject:[UIBarButtonItem alloc]
    ➤initWithBarButtonSystemItem:UIBarButtonSystemItemReply
    ➤target:self action:@selector(doAction:)] autorelease];
    [allitems addObject:[UIBarButtonItem alloc]
    ➤initWithBarButtonSystemItem:UIBarButtonSystemItemSearch
    ➤target:self action:@selector(doAction:)] autorelease];
    [allitems addObject:[UIBarButtonItem alloc]
    ➤initWithBarButtonSystemItem:UIBarButtonSystemItemStop
    ➤target:self action:@selector(doAction:)] autorelease];
    [allitems addObject:[UIBarButtonItem alloc]
    ➤initWithBarButtonSystemItem:UIBarButtonSystemItemTrash
    ➤target:self action:@selector(doAction:)] autorelease];

```

```

        [toolbar setItems:allitems];
        [contentView addSubview:toolbar];
        [toolbar release];
    }

    -(void) dealloc
    {
        [toolbar release];
        [allitems release];
        [contentView release];
        [super dealloc];
    }
@end

```

工具栏提示

使用工具栏时，下面是一些将来会用到的技巧。

□ 固定空间可以拥有宽度。

在所有UIBarButtonItem中，只有UIBarButtonItemFixedSpace条目可以被分配一个宽度。因此，创建空间条目，设置其宽度，然后只把它添加到条目数组中。

```

UIBarButtonItem *spacer = [[[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemFixedSpace
target:NULL action:NULL] autorelease];
spacer.width = 20;
[items addObject:spacer];

```

□ 使用一个灵活空间进行左对齐或右对齐。

在条目列表开始添加一个UIBarButtonItemFlexibleSpace会使所有剩余条目右对齐。在末尾添加一个UIBarButtonItemFlexibleSpace会使所有剩余条目左对齐。使用两个UIBarButtonItemFlexibleSpace，一个添加在开头，一个添加在末尾，会使剩余条目居中对齐。

□ 考虑遗漏的条目。

根据上下文隐藏栏按钮条目时，不要只使用灵活的空间分配来除去它们。而是使用一个与条目原来大小相匹配的固定宽度的空间代替该条目。那样做会在条目消失之前和消失之后保存布局并把所有其他图标放在原来的位置。

□ 栏按钮条目不是视图。

如果你需要向一个栏按钮条目添加子视图，应该为它提供一个合适的父视图并让该父视图（不管是工具栏、标签栏还是导航栏）在屏幕上构建它。然后，可以使用文档中未记录的view调用来恢复栏按钮条目的视图。

8.14 小结

本章介绍了许多与控件进行交互和在程序中使控件发挥最大作用的非标准（有些是非官方和

文档中未记录的)方法。在继续下一章的内容之前,先回顾一下本章的一些要点。

- ❑ 不要害怕查看控件,你会发现它们是如何组织在一起的。公开了像标签和按钮这样的标准SDK类之后,就可以根据自己的规范和期望来定制它们。
- ❑ 如果只是一个条目属于UIControl类,这并不意味着你可以把它当作一个UIView来对待。可以为它提供子视图,调整其大小,为它制作动画,在屏幕周围移动它或在以后点击它。
- ❑ 利用Core Graphic可以根据需要构建可见元素。使用SDK类的舒适感和随时发出的叫好声会为外观呈现增加砝码。
- ❑ 和搜索栏键盘示例一样,不要让苹果公司有关应该如何使用控件的预想概念妨碍你以自己希望的方式使用它。如果“search”没有描述在交互结束时用户需要做什么,就把Return按钮转换为一个能完成此任务的短语。
- ❑ 标注视图只是UIKit载有的许多未在文档中记录的类中的一个,它丰富了应用程序。
- ❑ 本章介绍了许多文档中未记录的类和调用。和所有未在文档中记录的非官方的资料一样,在软件中应该小心地使用它们。



除了在任意计算机上都能看到的标准用户界面控件和媒体组件，iPhone SDK还提供了大量特定于iPhone和iPod touch交付的专门开发人员解决方案。这些类和例程只能访问有限的iPhone核心功能。其中最有用的包括Address Book访问（人物）、Core Location（地点）和传感器（事件）。每一项功能都使iPhone更加人性化：你在哪里，你认识什么人，以及你如何持有这部电话。本章介绍了多种秘诀，向你展示如何在自己的开发工作中使用这些技术。

9.1 地址簿框架

iPhone SDK提供了两个地址簿框架：AddressBook.framework和AddressBookUI.framework。顾名思义，它们在iPhone SDK中占有两个截然不同的领域。Address Book包含基于C的低级结构和例程，用于从iPhone的机载SQLite数据库中访问联系人信息。Address Book UI提供呈现给用户的基于Objective-C的高级UIViewController浏览器对象。两个框架都比较小。它们只提供少数的类和类型。

在iPhone上，联系人数据位于主Library文件夹。在Macintosh仿真器上，可以在~/Library/Application Support/iPhone Simulator/User/Library中随意访问这些文件。AddressBook/AddressBook.sqlitedb和AddressBook/AddressBookImages.sqlitedb这两个文件都使用标准SQLite3来存储联系人信息和可选的联系人图像。在iPhone上，可以直接访问这些文件。它们位于/var/mobile/Library/AddressBook中。你必须使用这两个Address Book框架或者摆脱沙盒的束缚。后者会使任意应用程序的App Store资格失效。

9.1.1 Address Book UI

Address Book UI框架提供了一些与机载联系人数据库进行交互的预置视图，包括一个普通的人员选取器、一个个人联系人显示器和一个联系人编辑器。设置一个委托，然后把这些控制器都放入到导航器栈中或模式地显示它们，如本章中的秘诀所示。

和在第7章中所看到的Image Picker和Camera控制器一样，Address Book UI控制器也不是很灵活。苹果公司希望你根据说明来使用它们，只允开发人员进行小部分定制或不允许定制。另外，它们需要某种程度的低级编程功能。你会看到，这些类以迂回的方式与底层Address Book进行交互。

9.1.2 Address Book

在基于C的Address Book框架中，ABPerson类型提供核心联系人结构。此记录存储每个联系人的所有信息，包括姓名、电子邮件、电话号码，等等。由于每个人可能拥有与他或她的联系人相关的多个电子邮件、电话号码和重要日期，因此ABPerson使用一个复杂的“多值”结构来存储这些信息。在一般编程过程中，这意味着要使用一个数组。在Address Book中，这会包含少部分额外编程。

1. 处理人员选取器调用

因为日期、字符串和数字都使用不同的键入类型，因此Address Book提供了一些函数来帮助你从联系人数据文件中存储或恢复混合类型的数据。如果你正在寻找完整的Objective-C或Core Foundation调用，那么再看看吧。

通过AddressBookUI人员选取器响应GUI交互时，通常会恢复无类型的多值属性。也就是说，你从ABPerson记录复制了一个值，但不会准确地知道你复制的是什么内容。你必须请求属性类型，然后为每个类型提供单独的处理过程，从其数组中提取实际的值。例如：

```
id theProperty = [(id)ABRecordCopyValue(person, property) autorelease];
int propertyType = ABPersonGetTypeOfProperty(property);
if (propertyType == kABMultiStringPropertyType)
    NSString *result = [(NSString *)[(NSArray
    ➤*)ABMultiValueCopyArrayOfAllValues(theProperty)
    ➤objectAtIndex:identifier] autorelease];
```

这里，复制值函数提取一个属性，但不能获取属性类型。通过从所有已知类型中检验此属性类型，你可以为提取的属性标识一个类并提供已知类型的结果对象。

幸而，这种复杂的无类型处理过程仅限于交互地选择像电话号码、电子邮件地址或街道地址这样的属性。让用户一次性选择完整的联系人信息然后从该信息中恢复一般信息，这样做更容易一些。

2. 查询Address Book及其记录

直接进行数据库搜索显然比响应GUI容易一些。要请求所有可用联系人（企业和个人）的一个列表，请调用“所有人”（all people）函数。这将返回一个所有可用ABPerson的数组。这里看到的任意命名的create函数返回一个充满来自整个联系人数据库的数据的Address Book引用。实际上，你不是在创建一个新的Address Book，只是提供了带有到现有Address Book链接的复制数组函数。

```
peopleArray = (NSMutableArray
    ➤*)ABAddressBookCopyArrayOfAllPeople(ABAddressBookCreate());
```

查询ABPerson记录以找出任意和所有属性。这里有相当多的属性，如果我数得没错的话，应该是17个。它们的范围从生日和中间名字（middle name）到电子邮件地址和职称。一部分返回一个字符串或一个整数，另一部分返回一个字典或一个日期。一部分返回单个值（很少有人拥有多个名字），另一部分返回数组（人们通常拥有许多电子邮件地址）。

下面是从ABPerson记录中找出联系人的名字（first name）、姓氏（last name）和公司的方法。

每一个语句都返回一个字符串。注意ABRecordCopyValue函数是如何返回一个无类型的值的。要事先知道这些属性的类型或调用ABPersonGetTypeOfProperty()来请求类型：

```
NSString *firstName = [(NSString *)ABRecordCopyValue(person,
    kABPersonFirstNameProperty) autorelease];
NSString *lastName = [(NSString *)ABRecordCopyValue(person,
    kABPersonLastNameProperty) autorelease];
NSString *biz = [(NSString *)ABRecordCopyValue(person,
    kABPersonOrganizationProperty) autorelease];
```

9.2 秘诀：访问地址簿图像数据

一些Address Book联系人会使用图像数据，另一些则不会。为使联系人和面部图像匹配并提高iPhone的用户体验，相关图像提供了一个简洁的方法。使用图像数据时，图像被存储在AddressBookImage.sqlitedb数据库中。要检验一个图像是否可用，可以调用ABPersonHasImageData(person)，其中person是一条ABPerson记录。这会返回YES或NO，表示是否能够请求图像数据。

当返回YES时，就调用ABPersonCopyImageData(person)函数来提供NSData引用，你可以使用该引用填充UIImage对象。下面是将图像数据添加到UITableCell的典型调用：

```
if (ABPersonHasImageData(person)) {
    UIImage *img = [UIImage imageWithData:(NSData
        *)ABPersonCopyImageData(person)];
    [cell setImage:img];
} else { [cell setImage:NULL]; }
```

然而，图像不遵循任何本地尺寸。图像可以只有指甲那么小，也可以占据整个屏幕。秘诀9-1展示了如何访问联系人图像，并调整大小使其正好适合指定的标准略缩图大小，这里是60×60像素。此秘诀返回填充了地址信息和（如果可能）60×60像素的联系人图像的表单元。要实现这一目的，它会创建一个自定义位图上下文，并用成比例的图像填充该上下文。

说明 此代码调用MyCreateBitmapContext函数，在秘诀8-5中可以找到它。

秘诀9-1 访问联系人图像并调整其大小

```
#define IMG_SIZE 60

// Draw the image into a 60x60 bitmap and use that bitmap to
// create a new UIImage
id createImage(CGImageRef image)
{
    // Set the size of the output image
    CGRect aRect = CGRectMake(0.0f, 0.0f, IMG_SIZE, IMG_SIZE);

    // Create a bitmap context to store the new thumbnail
    CGContextRef context = MyCreateBitmapContext(IMG_SIZE, IMG_SIZE);
```



```

// Clear the context and draw the image into the rectangle
CGContextClearRect(context, aRect);
CGContextDrawImage(context, aRect, image);

// Return a UIImage populated with the new thumbnail
CGImageRef myRef = CGBitmapContextCreateImage(context);
free(CGBitmapContextGetData(context));
CGContextRelease(context);

return [UIImage imageWithCGImage:myRef];
}

// Return the string that best represents the contact name
// or organization
- (NSString *) getName: (ABRecordRef) person
{
    NSString *firstName = [(NSString *)ABRecordCopyValue(person,
        kABPersonFirstNameProperty) autorelease];
    NSString *lastName = [(NSString *)ABRecordCopyValue(person,
        kABPersonLastNameProperty) autorelease];
    NSString *biz = [(NSString *)ABRecordCopyValue(person,
        kABPersonOrganizationProperty) autorelease];

    if ((!firstName) && !(lastName))
    {
        if (biz) return biz;
        return @"[No name supplied]";
    }

    if (!lastName) lastName = @"";
    if (!firstName) firstName = @"";

    return [NSString stringWithFormat:@"%@ %@", firstName, lastName];
}

// Provide a table cell that contains the contact name and,
// where possible, a contact image
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSInteger row = [indexPath row];
    UITableViewCell *cell = [self.tableView
        dequeueReusableCellWithIdentifier:@"any-cell"];
    if (cell == nil)
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
            reuseIdentifier:@"any-cell"] autorelease];

    id person = [searchArray objectAtIndex:row];
    cell.text = [self getName:person];
}

```



```

// Set the cell image when image data is available
if (ABPersonHasImageData(person))
{
    UIImage *img = createImage([[UIImage imageWithData:(NSData *)
    ➡ABPersonCopyImageData(person)] CGImage]);
    [cell setImage:img];
}
else [cell setImage:NULL];

return cell;
}

```

9.3 秘诀：显示地址簿信息

AddressBookUI框架简化了联系人信息的显示。使用ABPersonViewController实例来呈现存储在ABPerson记录中的数据，如图9-1中所示。AddressBookUI除去了所有的设计细节，创建了完美的布局。如秘诀9-2所示，只需要分配控制器，将displayPerson设置为一个ABPerson记录，然后把视图控制器放入正确的位置即可。

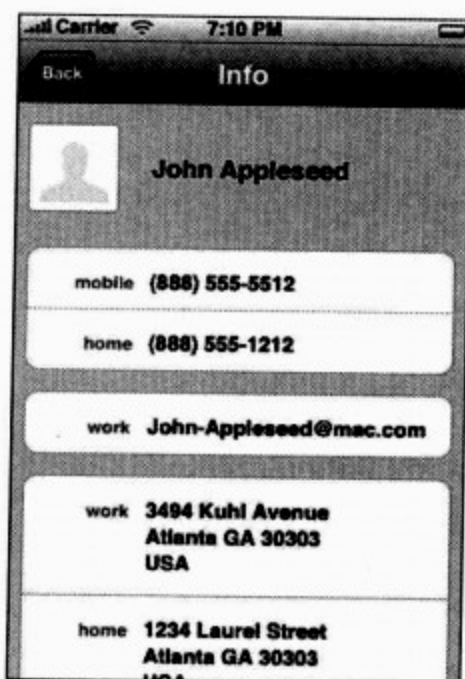


图9-1 初始化并显示AddressBookPersonViewController实例，以呈现来自任何ABPerson记录的数据

秘诀9-2 使用ABPersonViewController

```

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
➡*)newIndexPath
{
    ABPersonViewController *pvc = [[ABPersonViewController alloc] init];
    pvc.displayedPerson = [searchArray objectAtIndex:[newIndexPath row]];
    [[self navigationController] pushViewController:pvc animated:YES];
}

```

9.4 秘诀：浏览地址簿

AddressBookUI框架提供了一些可运行的控制器。浏览整个联系人列表的操作就像显示各个联系人屏幕一样容易完成。使用ABPeoplePickerNavigationController类来呈现交互式浏览器，如图9-2所示。下面是创建该显示的代码：

```
ABPeoplePickerNavigationController *ab = [[ABPeoplePickerNavigationController
➡alloc] init];
[ab setPeoplePickerDelegate:self];
[self presentViewController:ab animated:YES];
```

决定何时跳出浏览器以及如何恢复任意选中的数据仍然是很棘手的问题。使用3个委托方法控制你的选取器如何对用户交互做出反应。你可以指定用户点击一个联系人或任意联系人属性后，或者当用户点击Cancel时做出的反应：

- peoplePickerNavigationController:
 - ➡shouldContinueAfterSelectingPerson:

当用户点击某联系人时，你有两个选择。可以作为最后选择接受此联系人并清除模式视图控制器，或者导航到在图9-1中看到的个人信息显示界面。如果只选择联系人，此方法必须返回NO。如果要继续导航到个人界面，则返回YES。第二个参数包含选中的联系人，以防在选择任意ABPerson记录后想要停止。

- peoplePickerNavigationController:
 - ➡shouldContinueAfterSelectingPerson: property: identifier:

直到用户进入到各个联系人信息显示屏幕（像图9-2中所示的屏幕一样）时，才会调用这个可选方法。然后，是返回程序（返回NO）还是继续（返回YES），由你来决定。秘诀9-3展示了如何确定点击了哪一个属性以及如何恢复它的值。但这样做很别扭而且很复杂。

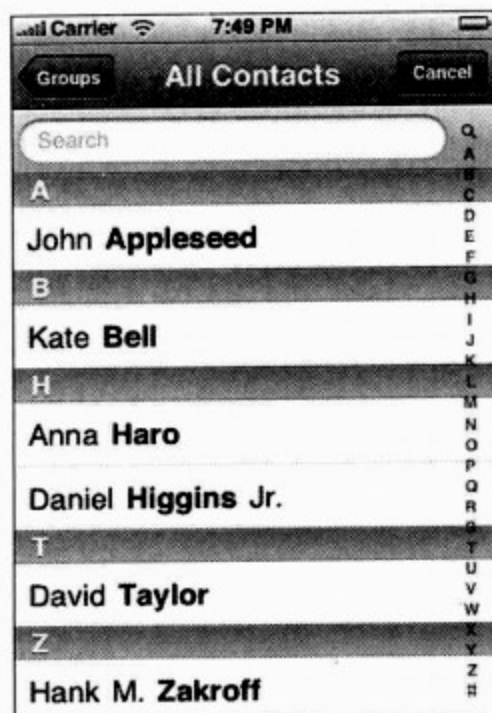


图9-2 iPhone人员选取器导航控制使用户能够在联系人数据库中进行搜索并选择一个人或组织

- peoplePickerNavigationControllerDidCancel:

用户点击Cancel之后, 需要清除模式视图。通过实现此委托方法来获取取消事件, 并使用它来执行清除。

秘诀9-3 处理人员选取器事件

```
// Handle the tap on a username

- (BOOL) peoplePickerNavigationController:(ABPeoplePickerNavigationController
➡*)peoplePicker shouldContinueAfterSelectingPerson:(ABRecordRef)person
{
    self.title = [self getName:person];
    return YES; // continue to secondary screen
}

// Handle the tap on a user property
- (BOOL) peoplePickerNavigationController:(ABPeoplePickerNavigationController
➡*)peoplePicker shouldContinueAfterSelectingPerson:(ABRecordRef)person
➡property:(ABPropertyID)property identifier:(ABMultiValueIdentifier)identifier
{
    // Get the selected property and its property type
    id theProperty = (id)ABRecordCopyValue(person, property);
    int propertyType = ABPersonGetTypeOfProperty(property);

    // determine the kind of property passed and
    // print out its value to the debugging console
    if (propertyType == kABStringPropertyType) {
        printf("%s\n", [theProperty UTF8String]);
    } else if (propertyType == kABIntegerPropertyType) {
        printf("%d\n", [theProperty integerValue]);
    } else if (propertyType == kABRealPropertyType) {
        printf("%d\n", [theProperty floatValue]);
    } else if (propertyType == kABDateTimePropertyType) {
        printf("%s\n", [[theProperty description] UTF8String]);
    } else if (propertyType == kABMultiStringPropertyType) {
        printf("%s\n",
            [[(NSArray *)ABMultiValueCopyArrayOfAllValues(theProperty)
            objectAtIndex:identifier] UTF8String]);
    } else if (propertyType == kABMultiIntegerPropertyType) {
        printf("%d\n",
            [[(NSArray *)ABMultiValueCopyArrayOfAllValues(theProperty)
            objectAtIndex:identifier] integerValue]);
    } else if (propertyType == kABMultiRealPropertyType) {
        printf("%f\n",
            [[(NSArray *)ABMultiValueCopyArrayOfAllValues(theProperty)
            objectAtIndex:identifier] floatValue]);
    } else if (propertyType == kABMultiDateTimePropertyType) {
        printf("%s\n",
            [[[NSArray *)ABMultiValueCopyArrayOfAllValues(theProperty)
            objectAtIndex:identifier] description] UTF8String]);
    }
}
```

```

    } else if (propertyType == kABMultiDictionaryPropertyType) {
        printf("%s\n",
            [[[NSArray *)ABMultiValueCopyArrayOfAllValues(theProperty)
              objectAtIndex:identifier] description] UTF8String]);
    }

    [self dismissModalViewControllerAnimated:YES];
    CFRelease(theProperty);
    [peoplePicker release];
    return NO; // leave the picker after printing property
}

// Handle a user's Cancel tap
- (void) peoplePickerNavigationControllerDidCancel:
    (ABPeoplePickerNavigationController *)peoplePicker
{
    [self dismissModalViewControllerAnimated:YES];
    [peoplePicker release];
}

```

9.4.1 (只) 浏览电子邮件地址

当你希望用户选择(例如)一个电子邮件地址时,你不会希望电子邮件地址和联系人的街道地址或传真号一同显示。你可以把人员选取器的显示属性限制在只显示用户从中选择的条目。要实现此目的,通过向控制器提交一个属性类型数组来选择显示的属性:

```

ABPeoplePickerNavigationController *ab = [[ABPeoplePickerNavigationController
    alloc] init];
[ab setDisplayedProperties:[NSArray arrayWithObject:[NSNumber
    numberWithInt:kABPersonEmailProperty]]];

```

9.4.2 添加新的联系人

ABNewPersonViewController类提供了一种交互式方法,用于向地址簿(Address Book)添加新联系人。但在编写本书时,苹果公司还没有完全地实现这个类。下面展示了这个类在未来将如何工作。这里显示的方法预先设置了联系人的姓和名字,不过这并不是强制性的:

```

- (void) presentSheet
{
    ABNewPersonViewController *np = [[ABNewPersonViewController
    alloc] init];
    np.addressBook = ABAddressBookCreate();
    ABRecordRef aRecord = ABPersonCreate();
    CFErrorRef anError;

    ABRecordSetValue(aRecord, kABPersonFirstNameProperty,
        CFSTR("Katie"), &anError);
    ABRecordSetValue(aRecord, kABPersonLastNameProperty,
        CFSTR("Bell"), &anError);
}

```



```
np.displayedPerson = aRecord;
CFRelease(aRecord);

np.newPersonViewDelegate = self;
[self presentViewController:np animated:YES];
}
```

9.5 Core Location

Core Location是iPhone SDK中最强大的特性之一。这项服务为不带内置GPS的iPhone和iPod touch设备返回（近似）立即响应的带有经度和纬度的地理定位，并为带有GPS的设备提供（精确的）定位功能。向应用程序添加定位功能为编程展开了全新的一页，你的计算机变得和你要计算的内容一样重要。你可以与朋友不期而遇，可以搜索本地资源，或者可以提供基于位置的个人信息流。

Core Location使应用程序能够连接到位置感知（location-aware）的社交网络和数据存储网站，如fireeagle.com、outside.in、upcoming.org、twitter.com和flicker.com。通过它可以提供按地理位置划分的内容或请求本地资源，如餐馆和事件列表。使用立即响应的地理定位，移动计算技术具有了新的意义，并为Web 2.0打开了更广阔的天地。

Core Location 是如何工作的

iPhone使用3个（还有可能是4个）方法来进行定位。这些技术依赖于几家提供商，包括Skyhook Wireless（<http://skyhookwireless.com>，即<http://loki.com>）、Google Maps（<http://maps.google.com>）和U.S. Department of Defense Global Positioning System。

1. GPS定位技术

如果你拥有一台新型3G iPhone，机载的GPS系统会使用美国国防部提供的一系列地球轨道卫星来跟踪你的移动位置。这些人造卫星发出微波信号，iPhone会接收这些信号并使用它们以高准确性来确定你的位置。和任何GPS系统相同，你和卫星之间的路径必须很通畅，因此最好在户外和远离树木的地方进行定位。

当前，在第一代2.5G iPhone或iPod touch上不能使用GPS定位系统。

2. SkyHook WiFi定位技术

Core Location首选的伪GPS地理定位方法依赖于SkyHook Wireless公司。SkyHook提供极其精确的WiFi定位。iPhone找到附近的WiFi和WiMax路由器后，它使用MAC地址搜索SkyHook数据库，根据该数据进行定位。

SkyHook WiFi数据集的工作方式如下。SkyHook将驱动程序向下发送到它所覆盖区域的城市街道，包括大多数美国城市区域。车辆扫描WiFi热点，找到后它们会记录下使用传统GPS定位技术匹配到WiFi MAC地址的位置。

在WiFi路由器静止不动时，这会工作得很好。当人们把他们的WiFi路由器打包并带着它们移动到（如）肯德基时，会工作得很糟糕。所以，SkyHook数据需要得到更新。它提供准确的定位

技术，通常能够在实际位置的几百英尺以内进行定位，即使人们和他们的路由器会继续移动到肯德基或其他地方。

3. Google Maps蜂窝基站（cell tower）定位技术

不太精确的方法包括Google Maps和蜂窝基站（cell tower）定位技术。这里，iPhone使用天线找到距离最近的4个或5个基站，然后根据蜂窝基站信号的强度对位置进行三角（triangulate）定位。你可能实际地见过蜂窝基站定位，它会显示距离你所站位置半英里远的地方（假定你不会正好站在一个蜂窝基站旁边）。

说也奇怪，SkyHook也提供了蜂窝基站定位技术，不过我上一次浏览iPhone框架时，调用会转到Google。这会根据苹果公司的专利使用权转让协议而有所改变。Yahoo! ZoneTag也提供了免费的蜂窝基站定位技术API，它比较新而且没有Google的那么精确，不过它是开放的，带有公共API。

iPod touch设备不能使用蜂窝基站定位技术，但第一代和第二代iPhone都能够使用。

4. SkyHook因特网服务提供商定位技术

SkyHook实际上提供了第三种定位方法，不过我从未见过iPhone使用它。还有，我住在大都市，并没有好好尝试过使用它。这个最后的方法使用因特网提供商定位来查找最近的因特网提供商的中心站。这是最后一招解决方案。返回的数据通常距离你的实际位置几英里（除非你正巧在拜访你的因特网提供商）。

5. 混合使用这些方法

iPhone分阶段处理定位问题。它根据请求的准确性级别使用一个回退方法。如果不能使用GPS或SkyHook WiFi映射准确地定位，就回退到Google Maps的蜂窝基站定位系统。如果那样做也没有效果，可能会进一步回退到SkyHook因特网提供商定位系统。如果那还不行，最终定位失败。SDK的最新版本为每个回退方法提供了多个（异步的）成功的回调。无论何时，你都会得到3个或4个结果。

知道iPhone如何完成这些任务很重要。这是因为在第一代iPhone上任意尝试10次来获取你的位置，最终可能只有3个或4个WiFi的成功结果，剩下的都回退到蜂窝基站定位系统。虽然你可以把自己需要的位置准确性设置为最高（也就是kCLLocationAccuracyBest，我经常这样做），除非你发起多个请求，否则会错过获取最准确位置的机会。

这样做的代价是耗费时间。一个定位请求会花费10或15秒的时间。处理多个请求，计算平均值和选取最好的结果最好在远离GUI的后台下完成。如果有可能，请避免让你的用户等待程序完成位置查询。

说明 苹果公司要求用户批准所有的位置请求。另外，不能根据App Store项创建交替应用程序。开发应用程序时，要紧记这些限制。

9.6 秘诀：Core Location 简介

Core Location易于使用。可以按照以下简单的步骤使用它。这些步骤详细地介绍了程序设置

过程以请求经度和纬度。

(1) 添加Core Location框架。把它拖到Xcode项目中, 然后把它添加到Group & Files栏中的Frameworks文件夹。

(2) 使用合适的包含文件。需要包含CoreLocation.h和CLLocationManagerDelegate.h:

```
#import <CoreLocation/CoreLocation.h>
#import <CoreLocation/CLLocationManagerDelegate.h>
```

(3) 分配一个位置管理器。把它的委托设置为主视图控制器, 并把它设置为使用可用的最佳准确性。

```
locmanager = [[CLLocationManager alloc] init];
[locmanager setDelegate:self];
[locmanager setDesiredAccuracy:kCLLocationAccuracyBest];
```

(4) 开始进行定位。告诉位置管理器开始更新位置。找到位置时, 委托回调会通知你。这会花费很多秒甚至1分钟的时间:

```
[locmanager startUpdatingLocation];
```

(5) 处理定位事件。你将处理两种类型的回调: 返回CLLocation数据的成功回调和不返回CLLocation数据的失败回调。添加处理每种情况的委托方法。在秘诀9-4中, 成功的回调恢复经度和纬度信息并运行Google Maps来显示该位置。根据使用的多种定位方法和请求的准确性, 你会获得3个或4个定位回调, 因此要考虑到这个非线性特性。

(6) 重复。如果你花得起时间(每次的定位请求会花费10~20秒钟的时间), 那么考虑重复定位几次以得到最准确的位置。定位信息返回水平和垂直的准确尺寸, 你可以使用它们估算出该定位技术有多准确。

秘诀9-4 使用Core Location检索经度和纬度

```
#import <CoreLocation/CoreLocation.h>
#import <CoreLocation/CLLocationManagerDelegate.h>

@interface HelloController : UIViewController <CLLocationManagerDelegate>
{
    UITextView          *contentView;
    CLLocationManager    *locmanager;
    BOOL                isLocating;
    BOOL                wasFound;
}
@end

@implementation HelloController

// Perform a location request - either start or stop locating
- (void) doIt
{
    if (isLocating)
```



```
        style:UIBarButtonItemStylePlain
        target:self
        action:@selector(doIt)]
        autorelease];

    [contentView setText:@"Location search failed"];
    isLocating = NO;
}

- (void)loadView
{
    contentView = [[UITextView alloc] initWithFrame:[[UIScreen mainScreen]
        applicationFrame]];
    [contentView setEditable:NO];
    self.view = contentView;
    [contentView release];

    locationManager = [[CLLocationManager alloc] init];
    [locationManager setDelegate:self];
    [locationManager setDesiredAccuracy:kCLLocationAccuracyBest];
    isLocating = NO;

    if (!locationManager.locationServicesEnabled)
    {
        [contentView setText:@"Location Services Are Not Enabled"];
        return;
    }

    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
        initWithTitle:@"Find Me"
        style:UIBarButtonItemStylePlain
        target:self
        action:@selector(doIt)]
        autorelease];
}

- (void) dealloc
{
    [contentView release];
    [super dealloc];
}

@end
```

9.7 秘诀：将地理编码转化为地址

转换地理编码（reverse geocoding）意味着把经度和纬度信息转换为人类可识别的地址信息。Yahoo的ZoneTag服务提供了免费的、受限的、非商业性用途的地理编码转换。首先需要在<http://developer.yahoo.com/yrb/zonetag/>上注册一个开发人员ID。

拥有自己的ID（也就是应用程序标记）之后，可以用经度和纬度调用Yahoo! 服务并获取该位置所属的国家、州、领域和邮政编码。这些信息并不具有很高的精确度（不会得到街道级别的地址），不过它是免费和开放的API。秘诀9-5展示了如何下载ZoneTag XML数据并使用iPhone的NSXMLParser类来获取位置信息。

记住，Yahoo! API是可以更改的。因此要不断检查ZoneTag，过一段时间就看看是否更新了某些细节。在编写本书时，ZoneTag开始在XML文本之前返回开发人员ID。你将在秘诀9-5中看到，这使得客户在到达实际的XML之前跳过ID标记的长度。

说明 ZoneTag也返回“标志（tags）”，有关本地事件、餐馆等的信息。我发现这些标志用途有限，因为该服务还相当新颖且没有流行起来。在本书出版时，Yahoo! 发布了Developer API中的几个重要更改。ZoneTag将被加入到Yahoo! 新的FireEagle服务中，API也将会相应地改变。

秘诀9-5 使用ZoneTag转换地理编码

```
@interface XMLParser : NSObject
{
    NSString          *current; // the current tag
    NSMutableString   *outstring; // results
}
@end

@implementation XMLParser

// Entry point for requesting the data from the URL
- (NSString *)parseXMLFile: (NSURL *) url
{
    outstring = [[NSMutableString alloc] init];

    NSXMLParser *parser = [[NSXMLParser alloc] initWithContentsOfURL:url];
    [parser setDelegate:self];
    [parser parse];
    [parser release];

    return [outstring autorelease];
}

// Entry point for parsing XML data directly
- (NSString *)parseXMLData: (NSData *) data
{
    outstring = [[NSMutableString alloc] init];

    NSXMLParser *parser = [[NSXMLParser alloc] initWithData:data];
    [parser setDelegate:self];
    [parser parse];
    [parser release];

    return [outstring autorelease];
}
```

```

}

// Parse an open XML Tag
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
  namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
  attributes:(NSDictionary *)attributeDict
{
    if (elementName)
        current = [NSString stringWithString:elementName];
}

// Parse the close XML tag
- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
  namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
{
    if (current) [current release];
    current = NULL;
}

// On finding data, test to see if the container is recognized
- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string
{
    if (!current) return;
    if ([current isEqualToString:@"country"] ||
        [current isEqualToString:@"state"] ||
        [current isEqualToString:@"zipcode"] ||
        [current isEqualToString:@"neighbourhood"])
        [outstring appendFormat:@"%s: %@\n", current, string];
}

- (void) dealloc
{
    [current release];
    [outstring release];
    [super dealloc];
}

@end

#define APPTOKEN @"YOUR_ZONE_TAG_DEVELOPER_ID_HERE"

// ZoneTag method from the calling view controller
- (void)locationManager:(CLLocationManager *)manager
  didUpdateToLocation:(CLLocation *)newLocation fromLocation:(CLLocation
  *)oldLocation
{
    // Respond only to the first location
    if (wasFound) return;
    wasFound = YES;
    self.navigationItem.rightBarButtonItem = NULL;

    CLLocationCoordinate2D loc = [newLocation coordinate];

```

```

[contentView setText:@"Location found. Looking up information via Yahoo."];
// Please use your own Yahoo Developer Token
if ([APPTOKEN isEqualToString:@"YOUR TOKEN HERE"])
{
    [contentView setText:@"You need to supply an application token to use
    ➡Yahoo services"];
    return;
}

// Contact Yahoo! and make a local info request
NSString *requestString = [NSString
➡stringWithFormat:@"http://zonetag.research.yahooapis.com/services/rest/V1/
➡suggestedTags.php?apptoken=%@&latitude=%f&longitude=%f&output=xml",
➡APPTOKEN, loc.latitude, loc.longitude];
NSURL *url = [NSURL URLWithString:requestString];
NSString *string = [[NSString stringWithContentsOfURL:url]
➡substringFromIndex:[APPTOKEN length]];
[contentView setText:[XMLParser sharedInstance] parseXMLData:[string
➡dataUsingEncoding:NSUTF8StringEncoding]];
}

```

9.8 秘诀：使用 Core Location 数据访问地图

除了ZoneTag的地理编码转换，Yahoo! 还提供了对开发人员友好的方法来生成和下载在应用程序中使用的地图。秘诀9-6使用了Yahoo的Maps服务下载Core Location返回的坐标上的地图。然后把该图像加载到一个Web视图上并显示它，使用户能够交互地在地图中进行拖动和缩放，如图9-3中所示。

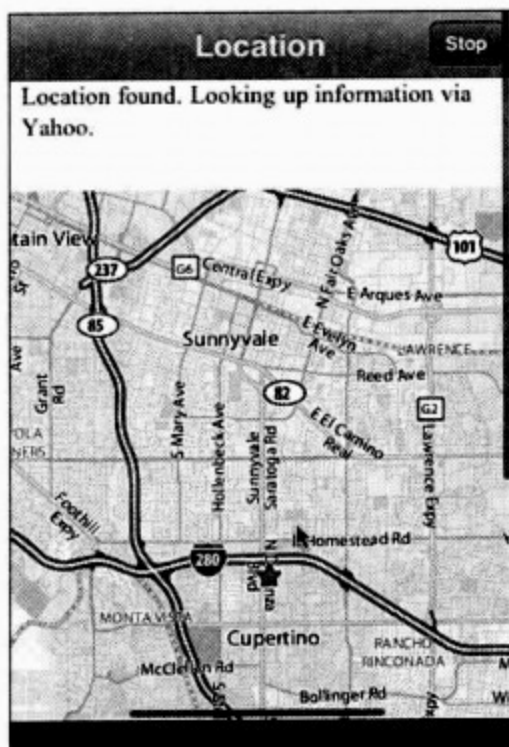


图9-3 这个Yahoo! 地图是根据Core Location返回的经度和纬度值而生成的。在仿真器上，你的位置始终默认为1 Infinite Loop，位于美国加利福尼亚州库比提诺的苹果公司总部

此代码使用了一个两层方法。第一个请求返回包含请求地图URL的XML。NSScanner实例帮助在简单的XML中导航, 以找到地图的URL。

第二个请求将地图数据下载到NSData对象, 该对象被存储到文件并用于填充UIWebView。

如果你希望将数据保存到iPhone照片簿中, 而不是在Web视图中显示它们, 首先要使用[UIImage initWithData:imgData]创建一个图像, 然后调用UIImageWriteToSavedPhotosAlbum(image, NULL, NULL, NULL)。如果你希望把数据写为一个JPEG图像而不是写到Documents文件夹中, 请使用UIImageJPEGRepresentation(image, 1.0f)创建JPEG格式的NSData版本, 然后将NSData保存到文件中。

说明 NSScanner类提供了扫描字符串的便利方法。虽然不像基于Perl的解决方法那样简洁, 但利用它可以通过把你的字符串和子字符串相匹配来获取数据。

秘诀9-6 从Yahoo下载地图数据

```
#define APPTOKEN @"YOUR_ZONE_TAG_DEVELOPER_ID_HERE"
- (void)locationManager:(CLLocationManager *)manager
didUpdateToLocation:(CLLocation *)newLocation fromLocation:(CLLocation
*)oldLocation
{
    // Respond only to the first location
    if (wasFound) return;
    wasFound = YES;
    self.navigationItem.rightBarButtonItem = NULL;

    CLLocationCoordinate2D loc = [newLocation coordinate];
    [contentView setText:@"Location found. Looking up information via Yahoo."];
    if ([APPTOKEN isEqualToString:@"YOUR_TOKEN_HERE"])
    {
        [contentView setText:@"You need to supply an application token to use
        Yahoo services"];
        return;
    }

    // Contact Yahoo! and make a local info request
    NSString *requestString = [NSString
    stringWithFormat:@"http://local.yahooapis.com/MapsService/V1/
    mapImage?appid=%@&latitude=%f&longitude=%f", APPTOKEN, loc.latitude,
    loc.longitude];

    // Retrieve the direct map location from the request URL
    NSURL *url = [NSURL URLWithString:requestString];
    NSString *directLocation = [NSString stringWithContentsOfURL:url];

    // Scan through XML for the map URL
    NSMutableString *mapURLString;
    NSScanner *scanner = [NSScanner scannerWithString:directLocation];
    [scanner scanUpToString:@">http://" intoString:NULL]; // first
```

```

[scanner scanUpToString:@"http://" intoString:NULL]; // second
[scanner scanUpToString:@"<" intoString:&mapURLString];

// Retrieve the image
url = [NSURL URLWithString:mapURLString];
UIImage *img = [UIImage imageWithData:[NSData dataWithContentsOfURL:url]];
NSData *imgData = UIImageJPEGRepresentation(img, 1.0f);

// Display it on a Web view
UIWebView *wv = [[UIWebView alloc] initWithFrame:[UIScreen mainScreen]
➡applicationFrame]];
[contentView addSubview:wv];
[wv loadData:imgData MIMEType:@"image/jpeg" textEncodingName:NULL
➡baseURL:NULL];
[wv setScalesPageToFit:YES];
self.view = wv;
}

```

9.9 秘诀：访问核心设备信息

利用UIDevice类能够获取特定于设备的关键值，包括正在使用的iPhone或iPod touch模型、设备名称、操作系统名称和版本。如秘诀9-7所示，它是获取系统详细信息的一站式解决方法。然而，它并不是一个完整而详尽的列表，苹果公司禁止了对iPhone的IORegistry的直接访问，虽然IOKit仍然是一个“公共”（而不是私有）框架。这会阻止你访问苹果公司认为比较敏感的信息，如iPhone的IMEI，用于为载体网络标识该电话机的国际化移动设备标识。

说明 使用iPhone独特的标识符登记设备，以实现临时应用程序分发。

秘诀9-7 使用UIDevice类

```

- (void) getDeviceInfo
{
    results = [[NSMutableString alloc] init];

    [results appendFormat:@"Identifier:\n %@\n", [[UIDevice currentDevice]
➡uniqueIdentifier]];
    [results appendFormat:@"Model: %@\n", [[UIDevice currentDevice] model]];
    [results appendFormat:@"Localized Model: %@\n", [[UIDevice currentDevice]
➡localizedModel]];
    [results appendFormat:@"Name: %@\n", [[UIDevice currentDevice] name]];
    [results appendFormat:@"System Name: %@\n", [[UIDevice currentDevice]
➡systemName]];
    [results appendFormat:@"System Version: %@\n", [[UIDevice currentDevice]
➡systemVersion]];

    [contentView setText:results];
}

```

9.10 秘诀：启用和禁用近程传感器

如果贴近身体部位放置iPhone，则近程传感器发挥不了多大作用。启用近程传感器时，它只做一件事。它检测前方是否存在一个大型对象，如果存在，就把屏幕关掉。将阻塞对象移开后，屏幕会重新打开。

秘诀9-8展示了如何在iPhone上启用近程感应系统。它将setProximitySensingEnabled:消息发送给共享的UIApplication实例。设置YES以启用此功能，设置NO禁用此功能。

秘诀9-8 启用近程感应系统

```
- (void) enable
{
    [[UIApplication sharedApplication] setProximitySensingEnabled:YES];
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                             initWithTitle:@"Disable Proximity"
                                             style:UIBarButtonItemStylePlain
                                             target:self
                                             action:@selector(disable)]
                                             autorelease];
}
```

9.11 秘诀：使用加速度将方向定位到“向上”

iPhone提供了3个机载传感器，它们沿iPhone的垂直坐标轴测量加速度，即左/右（X）、上/下（Y）和前/后（Z）。这些值表示影响iPhone的力，即同时来自重力和用户移动时的受力。

说明 通过在头顶转动（向心力）iPhone或者从高层建筑上丢掉它（自由下落），你可以得到反作用力。不过，你的iPhone快要被淘汰时不能恢复该数据。

要订阅一个iPhone的定向通知的申请，需要从某个视图控制器中调用`[[UIAccelerometer sharedAccelerometer] set Delegate:self]`。之后，委托将接收`accelerometer: didAccelerate:`消息，可以跟踪并响应这些消息。

发送到此方法的UIAcceleration对象为X、Y和Z轴返回一个浮点值。这个值的范围在-1.0到1.0之间。秘诀9-9使用这些值通过返回x和y矢量的反正切来确定哪一个方向是“向上”的。处理新的加速度消息时，此方法旋转带有箭头图像的UIImageView，如图9-4

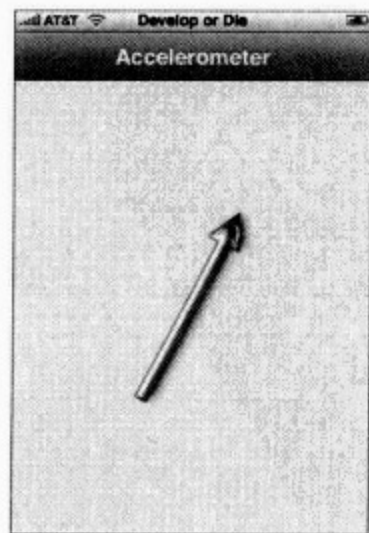


图9-4 通过使用x和y力矢量执行反正切函数可以恢复“向上”的方向。在此示例中，不管用户如何调整iPhone的方向，箭头始终指向上方

中所示。对用户操作的实时响应确保了不管用户如何调整电话机的方向，箭头都会继续指向上方。

秘诀9-9 获取加速度事件

```
#import <math.h>

@interface HelloController : UIViewController <UIAccelerometerDelegate>
{
    UIImageView *contentView;
    UIImageView *arrowView;
}
@end

@implementation HelloController
- (void)accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration
{
    float xx = -[acceleration x];
    float yy = [acceleration y];
    float angle = atan2(yy, xx);

    [arrowView setTransform:CGAffineTransformMakeRotation(angle)];
}

- (void)loadView
{
    contentView = [[UIImageView alloc] initWithFrame:[UIScreen mainScreen]
    applicationFrame];
    [contentView setImage:[UIImage imageNamed:@"bluedots.png"]];
    self.view = contentView;
    [contentView release];

    arrowView = [[UIImageView alloc] initWithImage:[UIImage
    imageNamed:@"arrow.png"]];
    [arrowView setCenter:CGPointMake(160.0f, 208.0f)];
    [contentView addSubview:arrowView];
    [arrowView release];

    [[UIAccelerometer sharedAccelerometer] setDelegate:self];
}

- (void) dealloc
{
    [contentView release];
    [super dealloc];
}
@end
```


9.12 秘诀：使用加速度移动屏幕上的对象

如果在编程中使用一些技巧，iPhone的机载加速计可以使对象在屏幕周围“移动”，实时地响应用户倾斜电话机的方式。秘诀9-10构建了一个蝴蝶动画，用户可以把它沿屏幕滑动。图9-5显示了应用程序运行时屏幕的样子，用户可以交互地控制蝴蝶的位置。

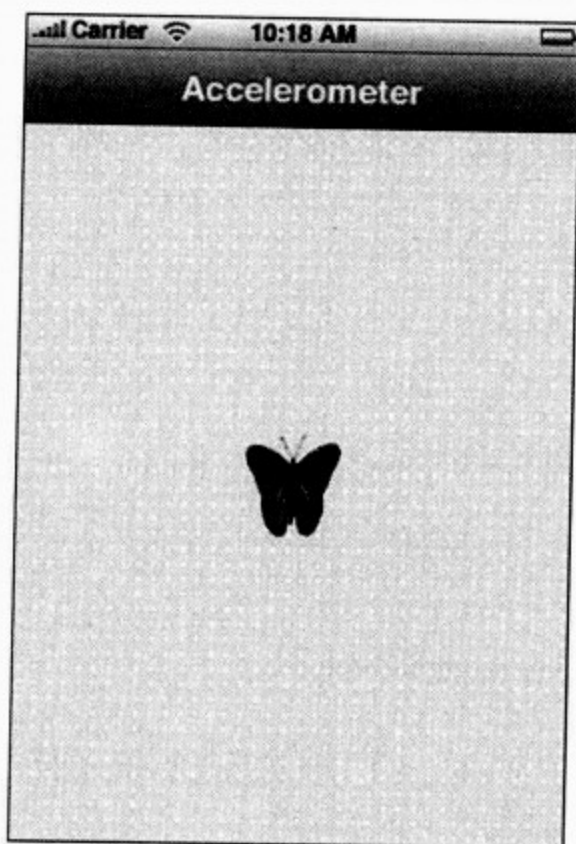


图9-5 在秘诀9-10中，通过向上、向下和向旁边倾斜iPhone，可以绕屏幕滑动蝴蝶

实现此目的的秘诀在于向程序添加一个“物理计时器”。和秘诀9-9中响应加速度的更改不同，加速计回调只测量当前的受力。由计时器例程负责把这些受力应用于蝴蝶。

- ❑ 只要所受的力保持同一个方向，蝴蝶就会加速。它的速度会按照X或Y方向上加速度的程度相应地增加。
- ❑ 计时器调用的tick例程通过向蝴蝶原点添加速度矢量来移动蝴蝶。
- ❑ 蝴蝶的移动有范围约束。当它碰到边缘时，就停止向那个方向移动。这使得蝴蝶一直显示在屏幕上。

秘诀9-10 根据加速计反馈滑动一个对象

```
#import <math.h>
@interface HelloController : UIViewController <UIAccelerometerDelegate>
{
    UIImageView *contentView;
    UIImageView *butterfly;
    float xaccel, yaccel, xvelocity, yvelocity;
}
@end
```

```

@implementation HelloController
- (void) tick
{
    // Move the butterfly according to the current velocity vector
    CGRect rect = [butterfly frame];
    rect.origin.x += xvelocity;
    rect.origin.y += yvelocity;

    // Check for boundary conditions so the butterfly stays on-screen
    if (rect.origin.x > 260.0f) rect.origin.x = 260.0f;
    if (rect.origin.x < 0.0f) rect.origin.x = 0.0f;
    if (rect.origin.y > 356.0f) rect.origin.y = 356.0f;
    if (rect.origin.y < 0.0f) rect.origin.y = 0.0f;

    [butterfly setFrame:rect];
}

#define SIGN(x) ((x < 0.0f) ? -1.0f : 1.0f)

- (void) accelerometer:(UIAccelerometer *) accelerometer
didAccelerate:(UIAcceleration *) acceleration
{
    // extract the acceleration components
    float xx = -[acceleration x];
    float yy = [acceleration y];

    // Has the direction changed?
    float accelDirX = SIGN(xvelocity) * -1.0f;
    float newDirX = SIGN(xx);
    float accelDirY = SIGN(yvelocity) * -1.0f;
    float newDirY = SIGN(yy);

    // Accelerate. To increase viscosity, lower the values below 1.0f
    if (accelDirX == newDirX)
        xaccel = (abs(xaccel) + 0.85f) * SIGN(xaccel);
    if (accelDirY == newDirY)
        yaccel = (abs(yaccel) + 0.85f) * SIGN(yaccel);

    // Apply acceleration changes to the current velocity
    xvelocity = -xaccel * xx;
    yvelocity = -yaccel * yy;
}

- (void) loadView
{
    contentView = [[UIImageView alloc] initWithFrame:[UIScreen mainScreen]
    applicationFrame];
    [contentView setImage:[UIImage imageNamed:@"bluedots.png"]];
    self.view = contentView;
}

```

```
[contentView release];

// Create the butterfly
butterfly = [[UIImageView alloc] initWithFrame:CGRectMake(40.0f, 300.0f,
    60.0f, 60.0f)];
[contentView addSubview:butterfly];
[butterfly release];

// Load the animation cells
NSMutableArray *bflies = [[NSMutableArray alloc] init];
for (int i = 1; i <= 17; i++) {
    NSString *cname = [NSString stringWithFormat:@"bf_%d.png", i];
    UIImage *img = [UIImage imageNamed:cname];
    if (img) [bflies addObject:img];
    [cname release];
}

// Begin the animation
[butterfly setAnimationImages:bflies];
butterfly.animationDuration = 0.75f;
[butterfly startAnimating];
[bflies release];

// Center the butterfly
[butterfly setCenter:CGPointMake(160.0f, 208.0f)];

// Set the butterfly's initial speed and acceleration
xaccel = 2.0f;
yaccel = 2.0f;
xvelocity = 0.0f;
yvelocity = 0.0f;

// Activate the accelerometer
[[UIAccelerometer sharedAccelerometer] setDelegate:self];

// Start the physics timer
[NSTimer scheduledTimerWithTimeInterval: 0.03f
    target: self
    selector: @selector(tick)
    userInfo: nil
    repeats: YES];
}

-(void) dealloc
{
    [contentView release];
    [super dealloc];
}

@end
```

9.13 小结

本章介绍了一些核心功能，包括Address Book、Core Location和机载传感器。这些特性把iPhone和iPod touch和当前市场上大多数可用的其他手持平台区别开来。下面是有关上述框架的一些要点。

- ❑ 由于苹果公司薄弱的文档管理以及处理“多值”数据时的复杂方式，低级的Address Book功能虽然极其有用，但被证明是很难使用的。它有助于慢慢理解这些例程并直接引用框架头文件，以补充有关多种类型和记录如何工作方面的知识。
- ❑ 使你的iPhone保持绝对静止时，根据多种定位算法的成败，调用Core Location仍然会得到不同的结果。检验你的位置并不像检验时间一样，其中有更大的变数。
- ❑ 和Google相比，Yahoo！对于社交网络和定位领域都是崭新的，其开发人员资源反映了Yahoo！在该领域中的构建承诺。不要小看它的开发人员API和新的位置感知服务，比如FireEagle和Upcoming.org。
- ❑ iPhone的加速计很好地补充其触摸界面。使用加速度数据可以扩展“触摸这里”的基本用户交互，并介绍倾斜感知的反馈。



作为一种能连接因特网的设备，iPhone非常适合用于订阅基于Web的服务。苹果公司通过其在各类网络计算方面的坚实基础以及支持技术丰富了这种平台。iPhone SDK处理套接字、密码密钥链、SQL访问、XML处理等。本章将探讨常用的网络计算技术，并提供能简化日常工作的方法。

10.1 秘诀：添加自定义设置束

任何iPhone应用程序均可在Settings主屏幕中添加首选项（参见图10-1）。自定义设置列于系统设置之后，但其他的外观和操作均与苹果公司在系统中预先加载的设置相同。正如这些屏幕快照所示，自定义首选项提供了多种数据交互风格，包括文本字段、开关和滑块。

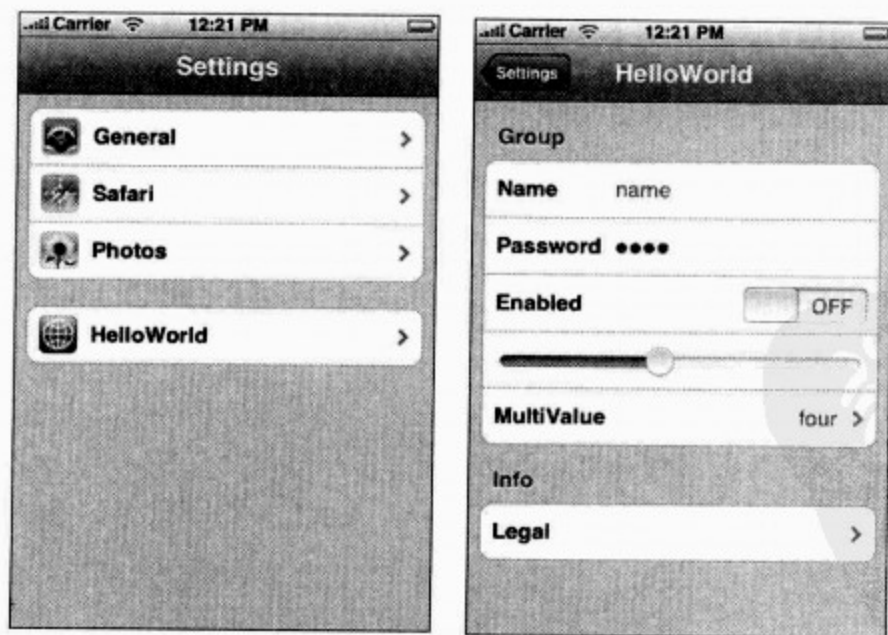


图10-1 （左）自定义设置束显示在Setting屏幕上，位于列表末端的内置设置之后。在iPhone上，可能需要向下滚动才能看到这些设置。（右）开发人员定义的首选项元素可包含文本字段（包括普通的和安全的）、开关、滑块、多值选项、组标题和子窗格

基于应用程序的设置将创建标准NSUserDefaults项目，你可以在程序中查询它们。使用设置可存储不敏感的账户首选项，例如用户名和选项切换开关。尽管密码通常是使用圆点隐藏的，但是以明文（clear text）的形式存储在应用程序沙盒之中。在处理更加敏感的信息时，应考虑使

用iPhone的安全密钥链。本章稍后将提供有关密钥链的秘诀。

声明应用程序设置

在/Developer/Platforms/iPhoneOS.platform/Developer/Library/Xcode/Plug-ins/iPhoneSettingsPlistStructDefs.xcodeplugin的Developer文件夹中可以找到设置模式的副本。Xcode使用此文件来检查属性列表语法。在该文件中，可以看到所有定义和用于指定自定义首选项的必需的和可选的属性。

每个设置窗格都对应于一个属性列表文件。秘诀10-1显示了图10-1（右侧）所示窗格的源代码。它显示了每种SDK设置类型及其定义。类型包括文本字段（字符串）、滑块（浮点数）、开关（布尔值）和多选项（n选1）。此外还可以组织项目并链接到子窗格。

要添加新设置，需要建立一个字典，并将其添加到PreferencesSpecifiers数组。每个独立的首选项字段都至少需要一个类型和一个标题。某些设置（如PSGroupSpecifier组项目）无需更多内容。而有些设置（如文本字段）则需要一些属性。你可能还希望指定大写和自动更正操作，以及键盘类型和密码安全特性是否隐藏文本，如秘诀10-1所示。

要为程序添加设置束，请按以下步骤操作。

- (1) 创建各属性列表，一个属性列表对应一个屏幕。主属性列表必须命名为Root.plist。
- (2) 创建一个新文件夹并添加属性列表。
- (3) 将文件夹重命名为Setting.bundle。OS X会为你提供关于名称的警告，继续操作确认重命名。文件夹会转换为一个束。（要查看新束的内容，只需右击[Control-click]并从弹出的上下文菜单中选择Show Package Contents即可。）
- (4) 将束拖到Xcode项目的Groups & Files列中（参见图10-2）。

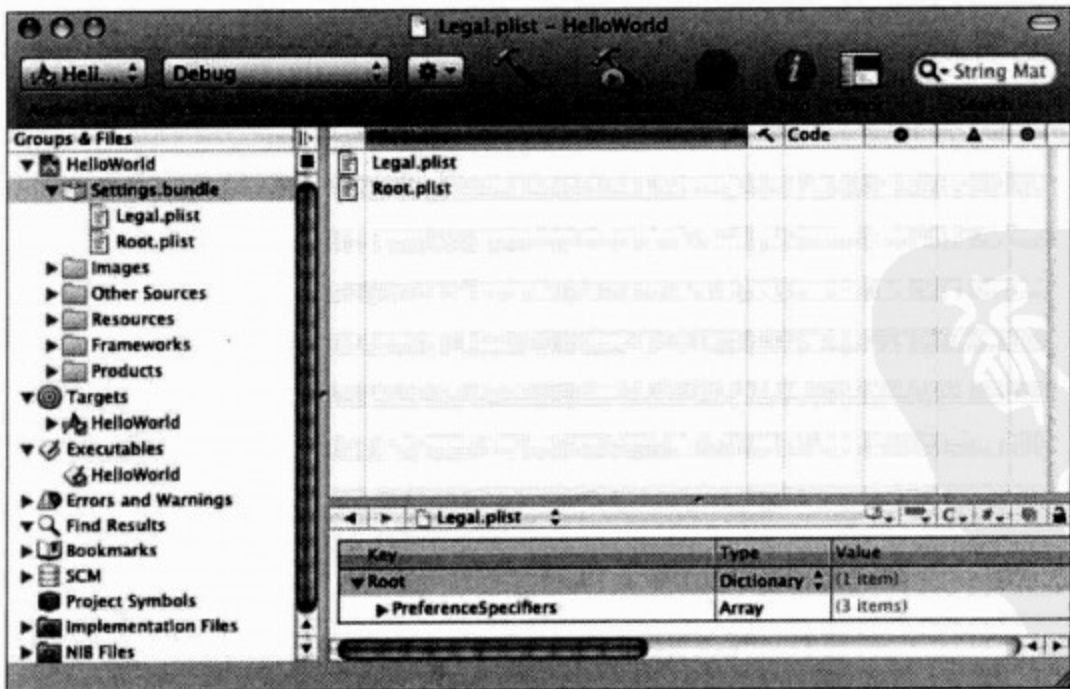


图10-2 将Setting.bundle添加到项目的Groups & Files列表中。
右击属性列表以便在Xcode中进一步编辑

在下一次运行程序时，设置束会安装并可供Settings应用程序使用。如果你的源代码存在语

法错误，你将看到空白屏幕，而不是预期的设置。分阶段构建设置能帮助避免这种情况。

Xcode提供了能进行一定交互、关注语法的编辑窗口。打开属性列表，随后选择View、Property List Type、iPhone Setting plist。我发现，在独立的属性列表编辑器或TextEdit中手动编辑更为简单。你也可以选择其他不同的工具。

说明 在秘诀10-1中，File属性未使用任何扩展名，默认使用.plist扩展名。

秘诀10-1 创建自定义设置窗格

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Title</key>
    <string>YOUR_PROJECT_NAME</string>
    <key>StringsTable</key>
<string>Root</string>
<key>PreferenceSpecifiers</key>
<array>
    <dict>
        <key>Type</key>
        <string>PSGroupSpecifier</string>
        <key>Title</key>
        <string>Group</string>
    </dict>
    <dict>
        <key>Type</key>
        <string>PSTextFieldSpecifier</string>
        <key>Title</key>
        <string>Name</string>
        <key>Key</key>
        <string>name_preference</string>
        <key>DefaultValue</key>
        <string></string>
        <key>IsSecure</key>
        <false/>
        <key>KeyboardType</key>
        <string>Alphabet</string>
        <key>AutocapitalizationType</key>
        <string>None</string>
        <key>AutocorrectionType</key>
        <string>No</string>
    </dict>
    <dict>
        <key>Type</key>
        <string>PSTextFieldSpecifier</string>
        <key>Title</key>
```

```
<string>Password</string>
<key>Key</key>
<string>prefs_preference</string>
<key>DefaultValue</key>
<string></string>
<key>IsSecure</key>
<true/>
<key>KeyboardType</key>
<string>Alphabet</string>
<key>AutocapitalizationType</key>
<string>None</string>
<key>AutocorrectionType</key>
<string>No</string>
</dict>
<dict>
  <key>Type</key>
  <string>PSToggleSwitchSpecifier</string>
  <key>Title</key>
  <string>Enabled</string>
  <key>Key</key>
  <string>enabled_preference</string>
  <key>DefaultValue</key>
  <true/>
  <key>TrueValue</key>
  <string>YES</string>
  <key>FalseValue</key>
  <string>NO</string>
</dict>
<dict>
  <key>Type</key>
  <string>PSSliderSpecifier</string>
  <key>Key</key>
  <string>slider_preference</string>
  <key>DefaultValue</key>
  <real>0.5</real>
  <key>MinimumValue</key>
  <integer>0</integer>
  <key>MaximumValue</key>
  <integer>1</integer>
  <key>MinimumValueImage</key>
  <string></string>
  <key>MaximumValueImage</key>
  <string></string>
</dict>
<dict>
  <key>Type</key>
  <string>PSMultiValueSpecifier</string>
  <key>Key</key>
```



```
<string>multi_preference</string>
<key>DefaultValue</key>
<string>One</string>
<key>Title</key>
<string>MultiValue</string>
<key>Titles</key>
<array>
  <string>one</string>
  <string>two</string>
  <string>three</string>
  <string>four</string>
</array>
<key>Values</key>
<array>
  <string>one</string>
  <string>two</string>
  <string>three</string>
  <string>four</string>
</array>
</dict>

<dict>
  <key>Type</key>
  <string>PSGroupSpecifier</string>
  <key>Title</key>
  <string>Info</string>
</dict>
<dict>
  <key>Type</key>
  <string>PSChildPaneSpecifier</string>
  <key>Title</key>
  <string>Legal</string>
  <key>File</key>
  <string>Legal</string>
</dict>
</array>
</dict>
</plist>
```

10.2 秘诀：使应用程序支持自定义 URL 模式

利用自定义模式可以在 Mobile Safari（或其他应用程序）打开此类 URL 时启用应用程序。URL 模式表示在冒号前出现的 URL 的第一个部分，如 http 和 ftp。例如，如果你的应用程序注册了 xyz，则任何 xyz:// 链接都将直接转入你的应用程序以便处理，它们会传入可选的 application: handleOpenURL: 方法。

无论你是否定义了处理方法，自定义模式都将启动应用程序。如果你只希望运行应用程序，

可添加模式并打开URL，这支持交叉应用程序启动。

处理程序可扩展启动，从而允许应用程序对传入的URL进行一些处理。它们可能会打开特定数据文件，检索特定名称，显示某些图像或以其他方式处理调用中包含的信息。

代码清单10-1显示了一个简单的处理程序，它将传入的URL存储在应用程序的首选项中，以便供应用程序的视图控制器随后检索。

代码清单10-1 定义一个典型的URL处理程序例程

```
- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url
{
    if (!url) { return NO; }

    NSString *URLString = [url absoluteString];
    [[NSUserDefaults standardUserDefaults] setObject:URLString forKey:@"url"];
    [[NSUserDefaults standardUserDefaults] synchronize];
    return YES;
}
```

在应用程序的Info.plist文件中定制自定义URL模式。秘诀10-2显示了一个属性列表，注册xyz模式。它通过在CFBundleURLTypes数组中指定两个项目来完成此目标。第一个是CFBundleURLName，其中包含一个随机标识符，通常使用苹果公司的标准反转域名样式。第二个包含一个CFBundleURLSchemes数组，声明为字符串。该数组包含一个项目，即xyz模式。

当iPhone安装应用程序时，Info.plist列表通知SpringBoard将应用程序与xyz模式相关联。此后，在SpringBoard遇到匹配的模式时，就会启动恰当的应用程序来处理URL。

秘诀10-2 注册自定义URL模式

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CFBundleDevelopmentRegion</key>
    <string>en</string>
    <key>CFBundleDisplayName</key>
    <string>${PRODUCT_NAME}</string>
    <key>CFBundleExecutable</key>
    <string>${EXECUTABLE_NAME}</string>
    <key>CFBundleIconFile</key>
    <string>Icon.png</string>
    <key>CFBundleIdentifier</key>
    <string>com.sadun.${PRODUCT_NAME:identifier}</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>6.0</string>
    <key>CFBundleName</key>
    <string>${PRODUCT_NAME}</string>
    <key>CFBundlePackageType</key>
```



```

<string>APPL</string>
<key>CFBundleSignature</key>
<string>????</string>
<key>CFBundleVersion</key>
<string>1.0</string>
<key>CFBundleURLTypes</key>
<array>
  <dict>
    <key>CFBundleURLName</key>
    <string>com.sadun.demonstration</string>
    <key>CFBundleURLSchemes</key>
    <array>
      <string>xyz</string>
    </array>
  </dict>
</array>
</dict>
</plist>

```

10.3 秘诀：检查网络状态

对于网络，可以检查连接状态并恢复关于iPhone在网络中的角色的信息。下面的几个代码清单展示了如何测试网络状态，恢复iPhone的IP地址，查找站点的IP地址以及确定站点是否可以访问。

10.3.1 测试网络状态

系统配置(System Configuration)框架提供了许多支持联网的函数。其中的SCNetworkReachabilityCreateWithAddress用于检查IP地址是否可访问。传入清零地址（例如，0.0.0.0）来查询网络状态。代码清单10-2显示了此测试的实例。苹果公司的示例代码支持这种方法，它将在网络可用时返回YES，在不可用时返回NO。所返回的标记必须表明网络可访问(kSCNetworkFlagsReachable)或无需更多连接(kSCNetworkFlagsConnectionRequired)。可使用的其他标记如下。

- ❑ kSCNetworkReachabilityFlagsIsWWAN测试你的用户使用的是运营商的网络还是本地WiFi。设置为YES时，网络可通过EDGE、GPRS或其他单元连接来访问。
- ❑ kSCNetworkReachabilityFlagsDirect告诉你网络流量是通过网关到达还是直接到达。

说明 对飞行模式进行开或关可以测试此代码在网络启用和禁用情况下的表现。

代码清单10-2 检查iPhone的网络连接状态

```

- (BOOL) connectedToNetwork
{
    // Create zero addy

```

```

struct sockaddr_in zeroAddress;
bzero(&zeroAddress, sizeof(zeroAddress));
zeroAddress.sin_len = sizeof(zeroAddress);
zeroAddress.sin_family = AF_INET;

// Recover reachability flags
SCNetworkReachabilityRef defaultRouteReachability =
    SCNetworkReachabilityCreateWithAddress(NULL, (struct sockaddr
    *)&zeroAddress);
SCNetworkReachabilityFlags flags;

BOOL didRetrieveFlags =
    SCNetworkReachabilityGetFlags(defaultRouteReachability, &flags);
CFRelease(defaultRouteReachability);

if (!didRetrieveFlags)
{
    printf("Error. Could not recover network reachability flags\n");
    return 0;
}

BOOL isReachable = flags & kSCNetworkFlagsReachable;
BOOL needsConnection = flags & kSCNetworkFlagsConnectionRequired;
return (isReachable && !needsConnection) ? YES : NO;
}

```

10.3.2 恢复本地 IP 地址

在早期,开发人员James Cuff (core)和Jay Freeman (saurik)发现,iPhone的主机名查找特性不合常规。尽管较新的iPhone固件版本已经修订了许多bug,但在恢复iPhone的主机名时,iPhone依然表现奇特。与仿真器及其Mac OS X主机不同,iPhone不会自动为iPhone的主机名添加local后缀。

代码清单10-3展示了如何为iPhone检索本地IP地址。在通过iPhone建立联网服务时(例如FTP服务器或HTTP守护进程),这是一个方便的例程。该代码附加了丢失的local后缀,在纠正名称时,使用gethostbyname来返回有效的主机实体。此后inet_ntoa()从该主机提取IP地址。

说明 此代码在仿真器上无法使用(仿真器会正确地自动添加.local后缀)。我们仅可在iPhone和特定的iPod touch代码中运行这些代码。

代码清单10-3 获取iPhone的本地IP地址

```

// Return the localized IP address
- (NSString *) localIPAddress
{
    char baseHostName[255];

```



```

gethostname(baseHostName, 255);

char hn[255];
// This adjusts for iPhone by adding .local to the host name
sprintf(hn, "%s.local", baseHostName);
struct hostent *host = gethostbyname(hn);
if (host == NULL)
{
    perror("resolve");
    return NULL;
}
else {
    struct in_addr **list = host->h_addr_list;
    return [NSString stringWithCString:inet_ntoa(*list[0])];
}

return NULL;
}

```

10.3.3 查询站点的 IP 地址

iPhone支持使用标准的gethostbyname调用来查找远程站点的IP地址。代码清单10-4中显示的方法接受一个站点名称(如yahoo.com或www.google.com),并返回该站点的第一个可用IP地址。如你所见,完成此任务的例程是标准例程。不存在iPhone特有的古怪内容。

代码清单10-4 按主机名查找IP地址

```

- (NSString *) getIPAddressForHost: (NSString *) theHost
{
    struct hostent *host = gethostbyname([theHost UTF8String]);

    if (host == NULL) {
        perror("resolve");
        return NULL;
    }

    struct in_addr **list = (struct in_addr **)host->h_addr_list;
    NSString *addressString = [NSString stringWithCString:inet_ntoa(*list[0])];
    return addressString;
}

```

10.3.4 检查站点可用性

恢复了一个站点的IP地址后,使用检查站点是否可访问的SCNetworkReachabilityCreateWithAddress函数即可检查站点可用性。传入填充了站点IP地址的sockaddr记录(感谢苹果公司提供了实现此任务的方法),随后在函数返回时,检查kSCNetworkFlagsReachable标记。秘诀10-3显示了检查站点的hostAvailable:方法。它将返回YES或NO。

秘诀10-3 检查站点是否可访问

```

- (BOOL) hostAvailable: (NSString *) theHost
{
    NSString *addressString = [self getIPAddressForHost:theHost];
    if (!addressString)
    {
        printf("Error recovering IP address from host name\n");
        return NO;
    }

    struct sockaddr_in address;
    BOOL gotAddress = [self addressFromString:addressString address:&address];

    if (!gotAddress)
    {
        printf("Error recovering sockaddr address from %s\n", [addressString
            ➡UTF8String]);
        return NO;
    }
    SCNetworkReachabilityRef defaultRouteReachability =
        ➡SCNetworkReachabilityCreateWithAddress(NULL, (struct sockaddr *)&address);
    SCNetworkReachabilityFlags flags;

    BOOL didRetrieveFlags =
        ➡SCNetworkReachabilityGetFlags(defaultRouteReachability, &flags);
    CFRelease(defaultRouteReachability);

    if (!didRetrieveFlags)
    {
        printf("Error. Could not recover network reachability flags\n");
        return NO;
    }

    BOOL isReachable = flags & kSCNetworkFlagsReachable;
    return isReachable ? YES : NO;
}

// Populating a sockaddr_in record.
// Direct from Apple. Thank you Apple
- (BOOL)addressFromString:(NSString *)IPaddress address:(struct sockaddr_in *)
➡address
{
    if (!IPaddress || ![IPaddress length]) {
        return NO;
    }

    memset((char *) address, sizeof(struct sockaddr_in), 0);
    address->sin_family = AF_INET;
    address->sin_len = sizeof(struct sockaddr_in);

```

```

int conversionResult = inet_aton([IPAddress UTF8String], &address->sin_addr);
if (conversionResult == 0) {
    NSLog(@"Failed to convert the IP address string
    into a sockaddr_in: %@", IPAddress);
    return NO;
}

return YES;
}

```

10.4 秘诀：与 iPhone 数据库交互

数据库存储为网络访问提供了另一种支持机制。尽管iPhone提供了全面的sqlite3交互，但我发现苹果公司提供的库比较麻烦。代码清单10-5展示了来自苹果公司的SQLiteBooks代码的示例SQLite访问例程。这是一个非常典型的方法。请注意所使用的例程。它们集中关注基于C语言的libsql调用，还略带Cocoa风格。例程在数据库中查询SELECT author, copyright FROM book WHERE pk=?。

代码清单10-5 来自苹果公司的SQLiteBooks示例的示例SQL访问代码

```

// Brings the rest of the object data into memory. If already in memory, no action
is taken (harmless no-op).
- (void)hydrate {
    // Check if action is necessary.
    if (hydrated) return;
    // Compile the hydration statement, if needed.
    if (hydrate_statement == nil) {
        const char *sql = "SELECT author, copyright FROM book WHERE pk=?";
        if (sqlite3_prepare_v2(database, sql, -1, &hydrate_statement, NULL) !=
            SQLITE_OK) {
            NSLog(@"Error: failed to prepare statement with message '%s'.",
                sqlite3_errmsg(database));
        }
    }
    // Bind the primary key variable.
    sqlite3_bind_int(hydrate_statement, 1, primaryKey);
    // Execute the query.
    int success = sqlite3_step(hydrate_statement);
    if (success == SQLITE_ROW) {
        char *str = (char *)sqlite3_column_text(hydrate_statement, 0);
        self.author = (str) ? [NSString stringWithUTF8String:str] : @"";
        self.copyright = [NSDate dateWithTimeIntervalSince1970:sqlite3_column_
            double(hydrate_statement, 1)];
    } else {
        // The query did not return
        self.author = @"Unknown";
        self.copyright = [NSDate date];
    }
}

```

```

// Reset the query for the next use.
sqlite3_reset(hydrate_statement);
// Update object state with respect to hydration.
hydrated = YES;
}

```

我更喜欢使用Gus Mueller的FMDB Cocoa包装(wrapper)来满足数据库需求(<http://gusmueller.com/x/fmdb.zip>)。FMDB提供了sqlite绑定,可简化iPhone数据库的创建、访问和更新。秘诀10-3展示了FMDB实例,它将一个简单的SQLite调用以NSString查询的形式发送([db executeQuery:@"select * from call"])

秘诀10-4所显示的方法将列举最呼叫历史。将这段相对简单的代码与代码清单10-5相比较,两者实现了类似的数据库整体读取操作。

在秘诀10-4中,关注点依然是高级数据,而非低级访问调用,这为SQL访问提供了一种Objective-C方法。

要在程序中使用FMDB,请从Mueller的网站下载代码,为程序添加类,并将iPhone版本的libsqlite3.0.dylib拖到Xcode项目中。dylib文件与其他可链接的iPhone库处于相同位置,即/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS2.0.sdk/usr/lib/libsqlite3.0.dylib。

说明 在编写本书时,iPhone不会通过读取呼叫历史为你提供沙盒功能。如果这项策略更改,使用同样的FMDB方法即可访问其他任何可读取的SQL数据库。

秘诀10-4 使用FMDB访问iPhone呼叫历史

```

- (void) fetchCalls
{
    // Open the iPhone call history database
    NSString *dbpath = @"/private/var/mobile/Library/CallHistory/call_history.db";
    FMDatabase* db = [FMDatabase databaseWithPath:dbpath];
    if (![db open]) {
        printf("Could not open db.\n");
        return;
    }

    FMResultSet *rs = [db executeQuery:@"select * from call"];

    int i = 1; // counter
    NSMutableString *outstring = [[NSMutableString alloc] init];

    while ([rs next]) {

        // Recover the call time
        int calltime = [rs intForColumn:@"date"];
        NSDate *cdate = [NSDate dateWithTimeIntervalSince1970:calltime];
        NSString *caldate = [[cdate dateWithCalendarFormat:@"%Y-%m-%d %H:%M:%S"
        ➤timeZone:nil] description];
    }
}

```



```

// Duration in seconds
int secs = [[rs stringForColumn:@"duration"] intValue];
int minutes = (secs / 60) + 1;

// Phone number
NSString *phoneNumber = [rs stringForColumn:@"address"];
[outstring appendFormat:@"%3d: %@ %@ - %3d minutes\n", i++, caldate,
    *phoneNumber, minutes];
}

[db close];
[contentView setText:outstring];
[outstring release];
}

```

10.5 秘诀：将 XML 转换为树

iPhone SDK中提供的NSXMLParser类允许你扫描XML，在新元素被处理并完成时创建回调。在第9章中，秘诀9-5在其最简单的窗体中引入了NSXMLParser，查找叶元素并跟踪当前元素。如果你正下载简单的数据提要，希望获得一些相关的信息，那么该类非常适用。进行依赖于错误检查、状态信息和双向握手的生产类工作时，这个类并无优势。

遗憾的是，iPhone SDK依靠NSXMLParser来完成核心XML处理。因此需要由你来填补NSXMLParser和基于树的解析结果之间的空缺。不过你可以轻松构建一个基于NSXMLParser的类来返回更加标准的基于树的数据。这需要简单而标准的树节点，如代码清单10-6所示。简单的双向链树节点提供了其父节点的指针、其子节点的数组、元素容器的键值，对于叶节点，还提供了存储容器内XML文本数据的叶值。

代码清单10-6 为XML支持定义简单的树节点

```

@interface TreeNode : NSObject
{
    TreeNode      *parent;
    NSMutableArray *children;
    NSString      *key;
    NSString      *leafvalue;
}
@property (nonatomic, retain) TreeNode      *parent;
@property (nonatomic, retain) NSMutableArray *children;
@property (nonatomic, retain) NSString      *key;
@property (nonatomic, retain) NSString      *leafvalue;

@end

```

使用该树节点数据类型，将解析树构建为处理XML源代码的NSXMLParser类。3种标准的NSXML例程（开始元素、结束元素和查找特征）对树执行递归的深度优先遍历。

在到达新元素 (parser: didStartElement: qualifiedName: attributes:) 时添加新节点, 在遇到文本时 (parser: foundCharacters:) 添加叶值。由于XML允许具有相同树深度的同级节点, 因而此代码使用栈来跟踪当前到树根的路径。在parser: didEndElement: 中, 同级节点总是会弹回相同的父节点, 使之添加到正确的级别。

完成XML扫描之后, parseXMLFile: 方法将返回根节点。

秘诀10-5展示了如何将XML转换为解析树。

秘诀10-5 将XML转换为解析树

```
@implementation XMLParser

static XMLParser *sharedInstance = nil;

// Use just one parser instance at any time
+(XMLParser *) sharedInstance
{
    if(!sharedInstance) {
        sharedInstance = [[self alloc] init];
    }
    return sharedInstance;
}

// Public parser returns the tree root
- (TreeNode *)parseXMLFile: (NSURL *) url
{
    stack = [[NSMutableArray alloc] init];

    root = [[TreeNode alloc] init];
    root.parent = NULL;
    root.leafvalue = NULL;
    root.children = [[NSMutableArray alloc] init];

    [stack addObject:root];
    [root release];

    NSXMLParser *parser = [[NSXMLParser alloc] initWithContentsOfURL:url];
    [parser setDelegate:self];
    [parser parse];
    [parser release];

    // pop down to real root
    return [[root children] lastObject];
}

// Descend to a new element
- (void)parser: (NSXMLParser *)parser didStartElement: (NSString *)elementName
    namespaceURI: (NSString *)namespaceURI qualifiedName: (NSString *)qName
    attributes: (NSDictionary *)attributeDict
{

```

```

    if (qName) elementName = qName;

    TreeNode *leaf = [[TreeNode alloc] init];
    leaf.parent = [stack lastObject];
    [(NSMutableArray *)[[stack lastObject] children] addObject:leaf];

    leaf.key = [NSString stringWithString:elementName];
    leaf.leafvalue = NULL;
    leaf.children = [[NSMutableArray alloc] init];

    [stack addObject:leaf];
    [leaf release];
}

// Pop after finishing element
- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
  namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
{
    [stack removeLastObject];
}

// Reached a leaf
- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string
{
    [[stack lastObject] setLeafvalue:[NSString stringWithString:string]];
}
@end

```

10.6 秘诀：存储和检索密钥链项

本章前面提到过，“安全”的密码文本字段并非总是那么安全。尽管文本在屏幕上被隐藏，但仍然以明文方式存储在首选项文件中。苹果公司通过其安全性框架，提供了一种基于密钥链的高度安全的密码服务。

秘诀10-6展示了如何利用这种技术，使用了4种可用的安全性调用。这些调用可以添加、更新、查询和删除密钥链中的密码。具体如下。

- ❑ **methodSecItemAdd()**。使用SecItemAdd在密钥链中添加新项。添加完成后，就无法直接修改。必须更新项，或在密钥链中删除项后重新添加。
- ❑ **SecItemUpdate()**方法。更新调用可以修改现有密钥链项。无法更新不存在的项，因此在更改项之前，一定要检查该项是否存在。
- ❑ **SecItemCopyMatching()**方法。复制匹配提供了重要的“查找”功能。它使你能够查询密钥链，以确定能否找到匹配项。
- ❑ **SecItemDelete()**方法。删除功能或许与你猜想的并不完全一致，它将从密钥链中删除项。

使密钥链的使用变得复杂的因素在于，它依靠创建字典来支持你的所有请求。上述4种函数

调用均使用字典在程序和密钥链间传递数据、属性和其他参数。搜索查询、更新请求和数据存储均需要不同的字典设置。

秘诀10-6中的代码提供了简单可行的单密码密钥链包装。利用它，你就可以设置密码（setpassword:），也可以检索密码（fetchPassword）。其余工作已抽象出来。内部方法会构建字典并调用Secutiry函数。

说明 安全性框架仅可用于iPhone设备。无法在模拟器上编译或测试其功能。

秘诀10-6 创建安全的密钥链密码保持包装

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

// Password Keeper securely saves just one password
@interface PasswordKeeper: NSObject
+ (PasswordKeeper *) sharedInstance;
- (void) setPassword: (NSString *) password;
- (NSString *) fetchPassword;
@end

@implementation PasswordKeeper

static PasswordKeeper *sharedInstance = nil;

+ (PasswordKeeper *) sharedInstance {
    if(!sharedInstance) {
        sharedInstance = [[self alloc] init];
    }
    return sharedInstance;
}

// Translate status messages into return strings
- (NSString *) fetchStatus : (OSStatus) status
{
    if (status == 0) return(@"Success!");
    else if (status == errSecNotAvailable) return(@"No trust results are
    available.");
    else if (status == errSecItemNotFound) return(@"The item cannot be found.");
    else if (status == errSecParam) return(@"Parameter error.");
    else if (status == errSecAllocate) return(@"Memory allocation error. Failed to
    allocate memory.");
    else if (status == errSecInteractionNotAllowed) return(@"User interaction is
    not allowed.");
    else if (status == errSecUnimplemented) return(@"Function is not implemented");
    else if (status == errSecDuplicateItem) return(@"The item already exists.");
    else if (status == errSecDecode) return(@"Unable to decode the provided
    data.");
    else
        return([NSString stringWithFormat:@"Function returned: %d", status]);
}
```



```
#define ACCOUNT @"iPhone Developer Cookbook Test Account"
#define SERVICE @"iPhone Developer Cookbook Test Service"
#define PWKEY @"iPhone Developer Cookbook Test Password Data"
#define DEBUG YES

// Return a base dictionary
- (NSMutableDictionary *) baseDictionary
{
    NSMutableDictionary *md = [[NSMutableDictionary alloc] init];

    // Password identification keys
    NSData *identifier = [PWKEY dataUsingEncoding:NSUTF8StringEncoding];
    [md setObject:identifier forKey:(id)kSecAttrGeneric];
    [md setObject:ACCOUNT forKey:(id)kSecAttrAccount];
    [md setObject:SERVICE forKey:(id)kSecAttrService];
    [md setObject:(id)kSecClassGenericPassword forKey:(id)kSecClass];

    return [md autorelease];
}

// Return a keychain-style dictionary populated with the password
- (NSMutableDictionary *) buildDictForPassword:(NSString *) password
{
    NSMutableDictionary *passwordDict = [self baseDictionary];
    // Add the password
    NSData *passwordData = [password dataUsingEncoding:NSUTF8StringEncoding];
    [passwordDict setObject:passwordData forKey:(id)kSecValueData]; // password

    return passwordDict;
}

// Build a search query based
- (NSMutableDictionary *) buildSearchQuery
{
    NSMutableDictionary *genericPasswordQuery = [self baseDictionary];

    // Add the search constraints: One match returning both
    // data and attributes
    [genericPasswordQuery setObject:(id)kSecMatchLimitOne
                                forKey:(id)kSecMatchLimit];
    [genericPasswordQuery setObject:(id)kCFBooleanTrue
                                forKey:(id)kSecReturnAttributes];
    [genericPasswordQuery setObject:(id)kCFBooleanTrue
                                forKey:(id)kSecReturnData];

    return genericPasswordQuery;
}
```

```

// retrieve data dictionary from the keychain
- (NSMutableDictionary *) fetchDictionary
{
    NSMutableDictionary *genericPasswordQuery = [self buildSearchQuery];

    NSMutableDictionary *outDictionary = nil;
    OSStatus status = SecItemCopyMatching((CFDictionaryRef)genericPasswordQuery,
    (CFTypeRef *)&outDictionary);
    if (DEBUG) printf("FETCH: %s\n", [[self fetchStatus:status] UTF8String]);

    if (status == errSecItemNotFound) return NULL;
    return outDictionary;
}

// create a new keychain entry
- (BOOL) createKeychainValue:(NSString *) password
{
    NSMutableDictionary *md = [self buildDictForPassword:password];
    OSStatus status = SecItemAdd((CFDictionaryRef)md, NULL);
    if (DEBUG) printf("CREATE: %s\n", [[self fetchStatus:status] UTF8String]);

    if (status == noErr) return YES; else return NO;
}

// remove a keychain entry
- (void) clearKeychain
{
    NSMutableDictionary *genericPasswordQuery = [self baseDictionary];

    OSStatus status = SecItemDelete((CFDictionaryRef) genericPasswordQuery);
    if (DEBUG) printf("DELETE: %s\n", [[self fetchStatus:status] UTF8String]);
}

// update a keychain entry
- (BOOL) updateKeychainValue:(NSString *)password
{
    NSMutableDictionary *genericPasswordQuery = [self baseDictionary];

    NSMutableDictionary *attributesToUpdate = [[NSMutableDictionary alloc] init];
    NSData *passwordData = [password dataUsingEncoding:NSUTF8StringEncoding];
    [attributesToUpdate setObject:passwordData forKey:(id)kSecValueData];

    OSStatus status = SecItemUpdate((CFDictionaryRef)genericPasswordQuery,
    (CFDictionaryRef)attributesToUpdate);
    if (DEBUG) printf("UPDATE: %s\n", [[self fetchStatus:status] UTF8String]);

    if (status == 0) return YES; else return NO;
}

// fetch a keychain value

```

```

- (NSString *) fetchPassword
{
    NSMutableDictionary *outDictionary = [self fetchDictionary];

    if (outDictionary)
    {
        NSString *password = [[NSString alloc] initWithData:[outDictionary
            ➤objectForKey:(id)kSecValueData] encoding:NSUTF8StringEncoding];
        return [password autorelease];
    } else return NULL;
}

- (void) setPassword: (NSString *) thePassword
{
    if (![self createKeychainValue:thePassword])
        [self updateKeychainValue:thePassword];
}
@end

```

10.6.1 存储多个密钥链值

使用密钥链时，我发现最简单的方法就是在处理多个密码时坚持使用单项输入方法。为此，应存储串行化的字典而非NSString。串行化会将字典转换为一个NSData对象。可以通过反串行化来恢复字典。代码清单10-7显示了如何在字典和数据之间进行转换。

代码清单10-7 使用串行化在字典和数据间进行转换

```

// Serialize to data
- (NSData *) dataFromDictionary: (NSMutableDictionary *) dict
{
    NSString *errorString;
    NSData *outData = [NSPropertyListSerialization dataFromPropertyList:dict
        ➤format:NSPropertyListBinaryFormat_v1_0 errorDescription:&errorString];
    return outData;
}

// Deserialize from data
- (NSMutableDictionary *) dictionaryFromData: (NSData *) data
{
    NSString *errorString;
    NSMutableDictionary *outDict = [NSPropertyListSerialization
        ➤propertyListFromData:data
        ➤mutabilityOption:kCFPropertyListMutableContainersAndLeaves format:NULL
        ➤errorDescription:&errorString];
    return outDict;
}

```

串行化会使你的密钥链方法变得简单。可以使用相同的方法来存储和检索秘诀10-6中的调

用。同时，使用字典可以扩展密码保持接口，以提供多键值存储。代码清单10-8显示了多密码密钥链包装的接口和对象调用。它构建在与秘诀10-6相同的数据输入代码之上。

代码清单10-8 扩展单项输入密钥链以支持多个密码

```
@interface PasswordKeeper: NSObject
+ (PasswordKeeper *) sharedInstance;
- (void) setObject: (id) anObject forKey: (NSString *) aKey;
- (void) removeObjectForKey: (NSString *) aKey;
- (id) objectForKey: (NSString *) aKey;
- (NSMutableDictionary *) passwordDict;
@end

// Simple Keychain Access Protocol Methods
- (void) setObject: (id) anObject forKey: (NSString *) aKey
{
    NSMutableDictionary *dict = [self fetchKeychainValue];
    if (dict)
    {
        // Keychain already has object
        [dict setObject:anObject forKey:aKey];
        [self updateKeychainValue:dict];
        return;
    }

    // Dictionary not found so create it
    dict = [[NSMutableDictionary alloc] init];
    [dict setObject:anObject forKey:aKey];
    if (![self createKeychainValue:dict]) [self updateKeychainValue:dict];
}

- (void) removeObjectForKey: (NSString *) aKey
{
    NSMutableDictionary *dict = [self fetchKeychainValue];
    if (dict)
    {
        // Keychain has object
        [dict removeObjectForKey:aKey];
        [self updateKeychainValue:dict];
        return;
    }
}

- (id) objectForKey: (NSString *) aKey
{
    NSMutableDictionary *dict = [self fetchKeychainValue];
    return [dict objectForKey:aKey];
}

- (NSMutableDictionary *) passwordDict
```



```
{
    return [self fetchKeychainValue];
}
```

10.6.2 密钥链持久化

即便在卸载应用程序之后，密钥链数据也会持续存在。与苹果公司有开发合作关系的Maurice Cusseaux写道：“由任何应用程序创建的密钥链项都将在卸载后保持，因为密钥链的存储并非在应用程序束之内，没有任何工具能通知系统在某些程序卸载时也卸载所有相关的密钥链项。要么因恶意卸载丢失敏感密码，要么保持敏感密码不受侵害并保证用户能安全地使用密码，这两者之间的权衡带来了一个策略问题。”

通过这种行为可以利用密钥链来维护持久的iPhone信息。你可跟踪用户的注册情况或限制演示模式的使用。持久化意味着在固件重装时（无备份存储库）不会清除数据。

10.7 发送和接收文件

在iPhone和其他主机之间来回发送文件可能是一项挑战。在撰写本书时，苹果公司尚未提供任何标准方法能以开放的方式与iPhone同步数据（第7章讨论了从iTune的mdbackup文件中恢复数据的有关内容）。下面列举了一些技术，你可能希望将其添加到你的应用程序中。

- **Bonjour**。iPhone的CFNetwork框架允许使用Bonjour的零配置网络与计算机之间传递数据。Bonjour支持自动服务发现，使你轻松连接同一个网络中的两个系统，以传输数据。苹果公司的WiTap样本代码演示了如何在你的应用程序中利用Bonjour消息传递。利用Bonjour，你可在主机之间发送短消息或整个文件。
- **FTP**。文件传输协议十分有用，因为它在多种平台上都是标准协议。苹果公司的Core Foundation、FTP示例代码很容易使用，你可在<http://developer.apple.com/samplecode/CFFTPSample/listing1.html>中找到源代码。你必须进行一些处理才能使上传和下载功能生效。存储了用户名、密码和主机信息之后，即可轻松自动传递FTP数据。
- **电子邮件**。苹果公司不再支持使用附件的形式为外发的电子邮件附加本地文件。因此，如果希望使用电子邮件，就必须构建你自己的客户端。只需进行简单的套接字编程，即可构建一个外发SMTP或SMTPAUTH实用工具。可通过Google查找示例代码。可以找到大量示例，大多数是使用标准C语言编写的。
- **Web服务**。你可使用iPhone的机载因特网连接，与万维网之间发送和接收数据。在本章和本书介绍的各个方法中，你已经看到了如何从URL下载数据。你还可以使用NSURLConnection类来处理GET和POST的同步与异步请求，从而将iPhone的数据上传到Web。在Google Code中可以找到Sean Heber的iAppADay源代码（<http://code.google.com/p/iappaday/>），这是学习数据发布的出色起点。
- **Web服务器**。无需实现整个Web服务器将数据从iPhone提供到浏览器。只需几个函数（提

供索引和一些文件)就可以了。如果你的目标是公开数据,那么创建一个基本的服务器将允许你使用Safari和Firefox等标准软件从iPhone检索文件。

10.8 秘诀: 构建一个简单的基于 Web 的服务器

Web服务器提供了将iPhone中的数据发送到其他计算机的最简便的方法。无需特殊的客户端软件。任何浏览器都可列举和访问基于Web的文件。图10-3展示了iPhone为Web浏览器提供内容的示例,浏览器屏幕上叠加的是iPhone屏幕快照。

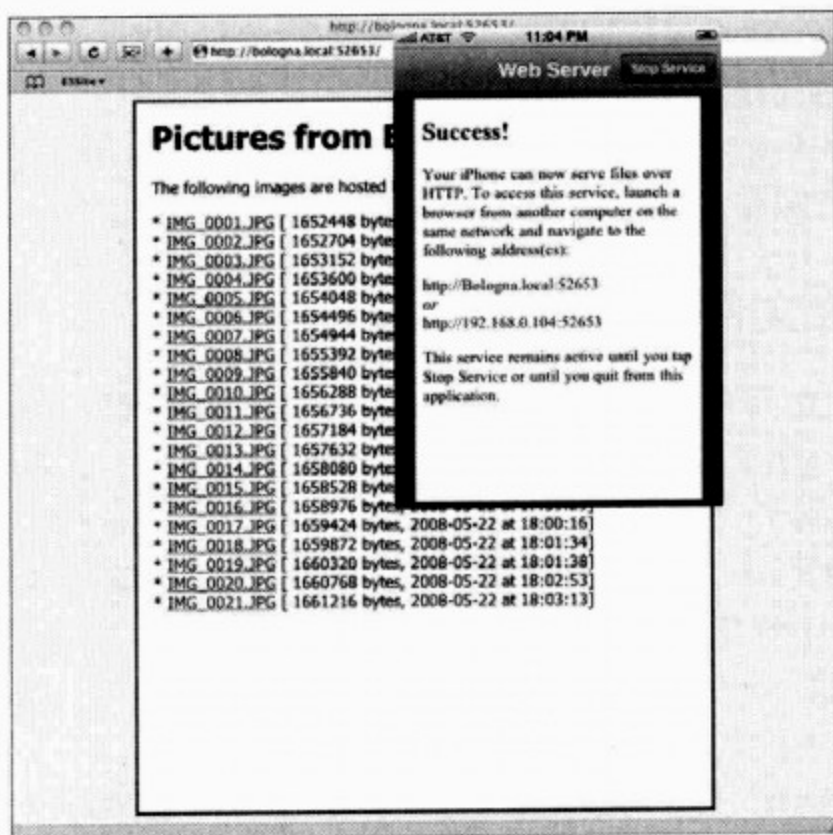


图10-3 仅需简单编程,即可设置iPhone,将文件发送到联网的计算机

Web服务器仅需几个关键例程。必须建立服务,创建一个侦听请求的循环(startServer),随后将请求传递给处理程序(handleWebRequest:),该处理程序将以所请求的数据作为响应。还需要一种方法(stopService)来处理服务关闭。秘诀10-7展示了建立和控制Web服务的3种核心方法。

循环例程使用低级套接字编程来建立一个侦听端口并捕捉客户端请求。当客户端发出GET命令时,服务器会截获该请求,并将其传递给Web请求处理程序。处理程序将分解请求以查找所需数据文件的名称,随后在文件夹中查找该文件,如果找到了文件,则读取并提供所请求的文件数据。

秘诀10-7 通过Web服务提供iPhone文件

```
// Serve files to GET requests
- (void) handleWebRequest:(NSNumber *)fdNum
{
```

```

NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
int fd = [fdNum intValue];
static char buffer[BUFSIZE+1];
int len = read(fd, buffer, BUFSIZE);
buffer[len] = '\0';

NSArray *reqs = [[NSString stringWithCString:buffer]
    ➤componentsSeparatedByString:@"\n"];
NSString *getreq = [[reqs objectAtIndex:0] substringFromIndex:4];
NSRange range = [getreq rangeOfString:@"HTTP/"];
if (range.location == NSNotFound)
{
    printf("Error: GET request was improperly formed\n");
    close(fd);
    return;
}

NSString *filereq = [[getreq substringToIndex:range.location]
    ➤stringByTrimmingCharactersInSet:[NSCharacterSet whitespaceCharacterSet]];

// Serve an index.html file to "GET /" requests
if ([filereq isEqualToString:@" /"])
{
    NSString *outcontent = [NSString stringWithFormat:@"HTTP/1.0 200
    ➤OK\r\nContent-Type: text/html\r\n\r\n"];
    write(fd, [outcontent UTF8String], [outcontent length]);

    NSString *outdata = [self createindex];
    write(fd, [outdata UTF8String], [outdata length]);
    close(fd);
    return;
}

// Check extensions against known MIME types
NSString *mime = [self mimeForExt:[filereq pathExtension]];
if (!mime)
{
    printf("Error recovering mime type.\n");
    NSString *outcontent = [NSString stringWithFormat:@"HTTP/1.0 200
    ➤OK\r\nContent-Type: text/html\r\n\r\n"];
    write (fd, [outcontent UTF8String], [outcontent length]);
    outcontent = @"<p>Sorry. This file type is not supported</p>\n";
    write (fd, [outcontent UTF8String], [outcontent length]);
    close(fd);
    return;
}

// Read and serve the file
NSString *outcontent = [NSString stringWithFormat:@"HTTP/1.0 200
    ➤OK\r\nContent-Type: %@\r\n\r\n", mime];

```

```

write (fd, [outcontent UTF8String], [outcontent length]);
filereq = [filereq
    stringByReplacingPercentEscapesUsingEncoding:NSUTF8StringEncoding];
NSString *fullpath = [PICS_FOLDER stringByAppendingString:filereq];
NSData *data = [NSData dataWithContentsOfFile:fullpath];
if (!data)
{
    printf("Error: file not found.\n");
    NSString *outcontent = [NSString stringWithFormat:@"HTTP/1.0 200
    OK\r\nContent-Type: text/html\r\n\r\n"];
    write (fd, [outcontent UTF8String], [outcontent length]);
    outcontent = @"<p>File was not found.</p>\n";
    write (fd, [outcontent UTF8String], [outcontent length]);
    close(fd);
    return;
}

printf("%d bytes read from file\n", [data length]);
write(fd, [data bytes], [data length]);
close(fd);

[pool release];
}

// Begin serving data
- (void) startServer
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    int sockfd;
    socklen_t length;
    static struct sockaddr_in cli_addr;
    static struct sockaddr_in serv_addr;

    // Set up socket
    if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        isServing = NO;
        return;
    }
    // Serve to a random port
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = 0;

    // Bind
    if(bind(listenfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        isServing = NO;
    }
}

```



```
        return;
    }

    // Find out what port number was chosen.
    int namelen = sizeof(serv_addr);
    if (getsockname(listenfd, (struct sockaddr *)&serv_addr, (void *)
        &namelen) < 0) {
        close(listenfd);
        isServing = NO;
        return;
    }
    chosenPort = ntohs(serv_addr.sin_port);

    // Listen
    if(listen(listenfd, 64) < 0)
    {
        isServing = NO;
        return;
    }

    // Service has now successfully started
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                                initWithTitle:@"Stop Service"
                                                style:UIBarButtonItemStylePlain
                                                target:self
                                                action:@selector(stopService)]
                                                autorelease];

    // Respond to requests until the service shuts down
    while (1 > 0) {
        length = sizeof(cli_addr);
        if((socketfd = accept(listenfd, (struct sockaddr *)&cli_addr, &length)) < 0)
        {
            isServing = NO;
            return;
        }
        [self handleWebRequest:[NSNumber numberWithInt:socketfd]];
    }

    [pool release];
}

// Shut down service
- (void) stopService
{
    printf("Shutting down service\n");
    isServing = NO;
    close(listenfd);
    [self stoppingService];
}
```

```
self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                         initWithTitle:@"Start Service"
                                         style:UIBarButtonItemStylePlain
                                         target:self
                                         action:@selector(startService)]
                                         autorelease];
}
```

10.9 通知推送

尽管苹果公司已经正式发布了iPhone正式的通知推送服务，但在撰写本书时，该服务尚未引入。通知推送服务允许你的应用程序在Web服务器上提供了可下载的新数据时通知用户。通知推送包括徽章（badge）（显示在SpringBoard图标上的红色小圈，参见代码清单4-2和代码清单4-3）、基于文本的警报（代码清单4-1）和自定义声音（秘诀4-11）。

要为应用程序添加此功能，你的服务器需要向iPhone的机载通知推送服务发送通知。在服务器端提供希望通知的应用程序的束标识符、iPhone的唯一设备ID、服务的代码名称，通知的长度不得超过1 K（包括苹果公司的所有DTD定义），通知可指定以下内容。

- ☐ 短消息；
- ☐ 启动应用程序的按钮（通常是“查看”）；
- ☐ 徽章号（正整数）；
- ☐ 指定播放哪个应用程序声音文件的警报声音（或UIDefaultSound）。

无需发送任何数据。为了保持推送机制有效并遵循1 K的限制，通知仅提供状态更改。应用程序在需要时检索数据。

在iPhone端，应用程序注册一项服务（从而为该服务提供唯一的iPhone标识符），并启用或禁用通知。应用程序的通知推送到一个设备时，你的应用程序通过即将发布的UIApplicationDelegate方法进行响应。（参见代码清单10-1，了解该方法的大致形式，使用通知处理取代URL处理。）

委托方法可选择处理通知或忽略通知。通常的方法是更新“可用数据”列表或下载新数据。例如，如果用户订阅了基于PDF的新闻稿，推送系统将允许用户了解新版本在何时可用。用户随后可启动应用程序来下载并阅读这些主题内容。

10.10 小结

本章介绍了许多网络支持技术。你了解了如何添加首选项窗格，构建自定义URL模式，检查网络连接，使用密钥链等。在本章结束时，请牢记以下几点。

- ☐ 密钥链提供了高度的程序安全性，但无法集成到易于使用的自定义设置窗格中。应谨慎考虑隐藏密码的最低安全保障方法是否足以保护你的程序。若如此，可绕开密钥链，利用iPhone上最优秀的开发人员集成特性之一。
- ☐ 尽管自定义URL模式提供了受限的应用程序间通信模式，但你也可以在一个应用程序中启动

另一个应用程序，只需调用（文档中未记录的）`launchApplicationWithIdentifier:suspended:`方法即可。

- ❑ 苹果公司的大多数网络支持都是通过极为低级的C例程提供的。如果你发现友好的Objective-C包装可简化编程工作，就应该考虑使用它。唯一的不足是，在你特别需要在应用程序的最基本层面进行严密的网络控制时，这不适用。
- ❑ 虽然苹果公司像NSXMLParser一样也提供了Objective-C包装，但它并不总是应该选择的类。调整类是iPhone编程中的一个重要部分。

虽然Cover Flow并未正式包含在iPhone SDK中，但它仍然是iPhone体验中最优秀的特性之一。使用Cover Flow能为用户提供极为出色的视觉选择，这一点令标准的滚动列表望尘莫及。本章将介绍Cover Flow，并展示如何在应用程序中使用它。

11.1 UICoverFlowLayer 类

虽然正式的SDK中未包含UICoverFlowLayer，但它仍然是标准的UIKit。通过Steve Nygard的类转储（class-dump）（www.codethecode.com/projects/class-dump/），能从UIKit框架中提取UICoverFlowLayer头文件，如代码清单11-1所示。在Mach-O文件的Objective-C程序中生成类声明便可启动类转储。

代码清单11-1 UICoverFlowLayer.h类头文件

```
/*
 *   Generated by class-dump 3.1.1.
 *
 *   class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2006 by Steve
 *   ☛Nygard.
 */

@interface UICoverFlowLayer : NSObject
{
    void *_private;
}

- (id)initWithFrame:(struct CGRect)fp8 numberOfCovers:(unsigned int)fp24;
- (void)dealloc;
- (void)setDelegate:(id)fp8;
- (void)setPlaceholderImage:(void *)fp8;
- (void)_prefetch:(unsigned int)fp8 atIndex:(unsigned int)fp12;
- (void)_requestBatch;
- (void)addSublayer:(id)fp8;
- (void)_requestImageAtIndex:(int)fp8 quality:(unsigned int)fp12;
- (void)_requestImageAtIndex:(int)fp8;
```



```

- (void)_notifySelectionDidChange;
- (void)startHeartbeat:(SEL) aSelector inRunLoopMode: (id)whatever2;
- (void)transitionIn:(float)fp8;
- (void)transitionOut:(float)fp8;
- (void)transition:(unsigned int)fp8 withCoverFrame:(struct CGRect)fp12;
- (void)transitionIn:(float)fp8 fromFrame:(struct CGRect)fp12;
- (void)transitionOut:(float)fp8 toFrame:(struct CGRect)fp12;
- (void)setDisplayedOrientation:(int)fp8 animate:(BOOL)fp12;
- (void)setInfoLayer:(id)fp8;
- (void)setImage:(void *)fp8 atIndex:(unsigned int)fp12 type:(unsigned int)fp16;
- (void)setImage:(void *)fp8 atIndex:(unsigned int)fp12 type:(unsigned int)fp16
  imageSubRect:(struct _NSRect)fp20;
- (void)setImage:(void *)fp8 atIndex:(unsigned int)fp12;
- (unsigned int)indexOfSelectedCover;
- (unsigned int)_coverAtScreenPosition:(struct CGPoint)fp8;
- (void)_recycleLayer:(int)fp8 to:(int)fp12;
- (void)_setNewSelectedIndex:(int)fp8;
- (void)_updateTick;
- (void)displayTick;
- (void)dragFlow:(unsigned int)fp8 atPoint:(struct CGPoint)fp12;
- (void)selectCoverAtIndex:(unsigned int)fp8;
- (void)selectCoverAtOffset:(int)fp8;
- (unsigned int)coverIndexAtPosition:(float)fp8;
- (void)_setupFlippedCoverLayer:(id)fp8;
- (void)flipSelectedCover;
- (int)benchmarkTick;
- (void)benchmarkHeartbeatLongScrub;
- (void)benchmarkHeartbeatShortScrub;
- (void)benchmarkHeartbeatScrubAndWait;
- (void)benchmarkTightLoop;
- (void)benchmarkTightLoopScrub;
- (BOOL)benchmarkLoadScrub;
- (BOOL)benchmarkImageManager:(void *)fp8;
- (void)benchmarkSetEnv;
- (void)benchmarkMode:(int)fp8;
- (void)benchmarkTickMode:(int)fp8;
- (void)benchmarkImageMode:(int)fp8;
- (void)benchmarkPerformanceLog:(BOOL)fp8;
- (void)benchmarkTightLoopTime:(unsigned int)fp8;
- (void)benchmarkLongScrubSpeed:(float)fp8;
- (void)benchmarkSkipImageLoad:(BOOL)fp8;
@end

```

以上代码说明，UICoverFlowLayer类定义的是图层，而不是视图。图层是抽象呈现类，存在于每个UIView对象中，它还能为这些视图提供内部画布。因此，如果你要使用此类，需要将它包含在可用的视图中。以下部分具体说明其实现方法。

11.2 构建 Cover Flow 视图

在开始处理Cover Flow之前，需要在UIView中插入UICoverFlowLayer实例。把图层放入视图就使Cover Flow技术成为了标准的UIKit术语，然后便可在本书中使用UIView类进行开发了。为此，你仅需创建视图和分配图层。然而，为了让视图正常地显示和工作，你还需要额外执行一些步骤。

这些步骤定制用户与Cover Flow交互的方式。用户在视图中触摸、拖动、双击时，Cover Flow视图需要对此做出正确的响应。为此，视图必须发送dragFlow: atPoint:消息，以处理与Cover Flow图层的触摸和拖动的交互过程。

双击处理提出了另一个难题。Cover Flow并未针对双击实现任何标准的响应，然而，你却希望应用程序能识别此交互过程，并对此执行一些合理的响应。按照我设计的UIView类，你就可以设置遵守CoverFlowHost协议的主应用程序。当视图截取双击操作时，就调用主应用程序的doubleTapCallback方法。这样，用户在所选择的封面上执行双击操作时，包含Cover Flow视图的视图控制器就能对此做出响应。

封面翻转（cover flipping）是指Cover Flow图像翻开以显示第二个视图（flipSelectedCover）时而采取的一种交互方式。通常，你可以使用双击反馈，或者使用封面翻转。但是，请确定只选取一种方式，如果同时实现两种方式，就会引发两种交互方式的无谓竞争，同时，你也许无法控制调用哪种响应方式。实现方法如代码清单11-2所示。

说明 本书示例代码中有关本章的完整项目，请参考<http://ericasadun.com>。

代码清单11-2 构建Cover Flow视图

```
#import <UIKit/UIKit.h>
#import "UICoverFlowLayer.h"
@protocol CoverFlowHost <NSObject>
- (void) doubleTapCallback;
@end

@interface CoverFlowView : UIView {
    id <CoverFlowHost> host;
    id info;
    UICoverFlowLayer *cfLayer;
    UILabel *label;
}
- (CoverFlowView *) initWithFrame: (CGRect) aFrame andCount: (int) aCount;
- (void) tick;
- (void) setHost: (id)anObject;
- (void) flipSelectedCover;
- (UILabel *) label;
- (UICoverFlowLayer *) cfLayer;
@end
@implementation CoverFlowView
```

```

- (CoverFlowView *) initWithFrame: (CGRect) aRect andCount: (int) count
{
    self = [super initWithFrame:aRect];
    cfLayer = [[UICoverFlowLayer alloc] initWithFrame:[UIScreen mainScreen]
        bounds] numberOfCovers:count];
    [[self layer] addSublayer:cfLayer];

    // Add the placeholder (image stand-in) layer
    CGRect phrect = CGRectMake(0.0f, 0.0f, 200.0f, 200.0f);
    UIImageView *phimg = [[UIImageView alloc] initWithFrame:phrect];
    [cfLayer setPlaceholderImage: [phimg layer]];

    // Add its info (label) layer
    label = [[UILabel alloc] init];
    [label setTextAlignment:UITextAlignmentCenter];
    [label setFont:[UIFont boldSystemFontOfSize:20.0f]];
    [label setBackgroundColor:[UIColor colorWithRed:0.0f green:0.0f blue:0.0f
        alpha:0.0f]];
    [label setTextColor:[UIColor colorWithRed:1.0f green:1.0f blue:1.0f
        alpha:0.75f]];
    [label setNumberOfLines:2];
    [label setLineBreakMode:UILineBreakModeWordWrap];
    [cfLayer setInfoLayer:[label layer]];

    return self;
}

- (void) touchesBegan: (NSSet*) touches withEvent: (UIEvent*) event
{
    UITouch *touch = [touches anyObject];
    CGPoint pt = [touch locationInView:self];
    [cfLayer dragFlow:0 atPoint:pt];
}

- (void) touchesMoved: (NSSet*) touches withEvent: (UIEvent*) event
{
    UITouch *touch = [touches anyObject];
    CGPoint pt = [touch locationInView:self];
    [cfLayer dragFlow:1 atPoint:pt];
}

- (void) touchesEnded: (NSSet*) touches withEvent: (UIEvent*) event
{
    UITouch *touch = [touches anyObject];
    CGPoint pt = [touch locationInView:self];

    if ([touch tapCount] == 2)
    {
        if (host) [host doubleTapCallback];
        return;
    }
}

```

```

        [cfLayer dragFlow:2 atPoint:pt];
    }
    - (void) flipSelectedCover
    {
        [cfLayer flipSelectedCover];
    }

    - (BOOL) ignoresMouseEvents {return NO;}
    - (void) tick {[cfLayer displayTick];}
    - (void) setHost: (id)anObject {host = anObject;}
    - (UILabel *) label {return label;}
    - (UICoverFlowLayer *) cfLayer {return cfLayer;}
@end

```

11.3 构建 Cover Flow 视图控制器

建立CoverFlowView类之后，你就可以向任何视图控制器中添加此视图。让一切正常运行的诀窍在于核对几个计划框。为了让Cover Flow呈现最佳的工作状态，你需要采取以下措施。

- ❑ **隐藏应用程序状态栏。**Cover Flow应用程序应该全屏运行，而不应出现令人分心的状态栏。隐藏状态栏的方式是调用[[UIApplication sharedApplication] setStatusBarHidden:YES]。
- ❑ **分配和初始化视图。**你必须提前知道将要使用的封面数。准备好封面和标题字符串，这些能为视图提供数据源。随后，封面数将被传给初始化程序。
- ❑ **等待。**等待loadView方法将Cover Flow设置为主视图，然后开始当前的运行循环(run loop)。
- ❑ **提供委托和数据源方法。**确保视图控制器能处理一些必要的Cover Flow委托和数据源回调。以下部分将详细介绍这些必要的和可选的方法。

当你完成上述这些目标后，应该会很快出现并运行有效的Cover Flow演示界面。图11-1显示通过视图控制器在代码清单11-3中建立的Cover Flow界面。在该Cover Flow中，蜡笔的不同颜色以一连串圆形样板的形式展现给用户。轻击翻转样板，将显示代表该颜色的6个字符的十六进制代码。

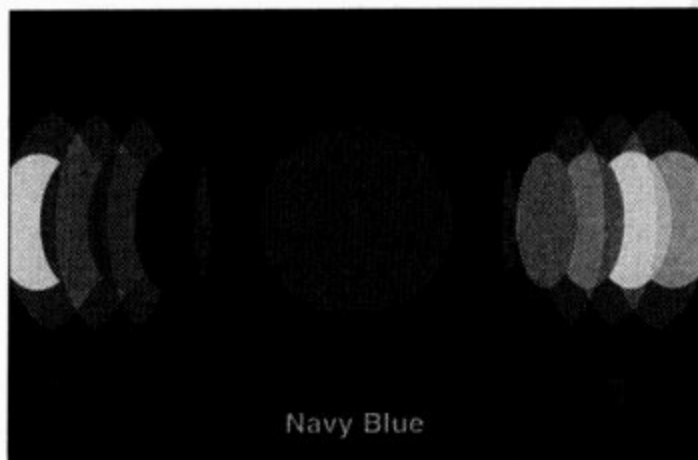


图11-1 在应用程序中使用文档中未记录的UIKit Cover Flow图层层类，构建引人注目的基于图像的Cover Flow演示程序

11.3.1 Cover Flow 数据源方法

对于标准的Cover Flow操作，视图控制器为coverFlow:requestImageAtIndex:quality方法提供了UIImage。利用所提供的索引确定要返回的图像。此方法对于数据源的正常工作的非常必要的。为了更快地载入图像，可以使用松散的图像生成方式，而不要使用以下代码清单中的数组方式来创建运行中需要的图像。你可以将这些图像存储到稀疏目录下，而只为那些将要使用的图像构建Cover Flow数据源方法。

翻转图层是可选的。用户轻击它们时，通过图层就能集中封面进行浏览。使用翻转图层时，向视图控制器中添加coverFlow:requestFlipLayerAtIndex:方法。此方法返回图像图层以显示翻转项目的后退路径。

11.3.2 Cover Flow 委托方法

如代码清单11-3所示，用户选取新页面时，Cover Flow视图产生selectionDidChange:回调方法。使用此方法能更新显示的标签文本，并跟踪最近所选项目的路径。

另一种委托方法是coverFlowFlipDidEnd:，它能通知你翻转动画何时完成。调用这种方法能实现从前向后翻转或者从后向前翻转，因此跟踪了对应状态的路径。请注意翻转视图是显示的还是隐藏的。

代码清单11-3 在视图控制器中构建Cover Flow视图

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

@interface CoverViewController : UIViewController
{
    CoverFlowView      *cfView;
    UICoverFlowLayer   *cfLayer;
    UILabel             *label;
    NSMutableArray      *covers;
    NSMutableArray      *titles;
    int                 whichItem;
    id                  target;
    SEL                 selector;

    NSMutableDictionary *colorDict;
    UILabel             *flippedView;
    BOOL                flipOut;
}
@end

// *****
// Cover Flow View Controller
//
@implementation CoverViewController
```

```

- (CoverViewController *) init
{
    if (!(self = [super init])) return self;
    // Read in the crayons and set up the arrays and dictionaries
    NSArray *crayons = [[NSString stringWithContentsOfFile:[NSBundle mainBundle]
    ➤pathForResource:@"crayons" ofType:@"txt"]]
    ➤componentsSeparatedByString:@"\n"];
    covers = [[NSMutableArray alloc] init];
    titles = [[NSMutableArray alloc] init];
    colorDict = [[NSMutableDictionary alloc] init];

    // Create the title and cover arrays
    for (NSString *crayon in crayons)
    {
        NSArray *theCrayon = [crayon componentsSeparatedByString:@"#"];
        if ([theCrayon count] != 2) continue;
        [titles addObject:[theCrayon objectAtIndex:0]];
        [covers addObject:createImage([theCrayon objectAtIndex:1])];
        [colorDict setObject:[theCrayon objectAtIndex:1] forKey:[theCrayon
        ➤objectAtIndex:0]];
    }

    // Create the flip object
    CGRect fliprect = CGRectMake(0.0f, 0.0f, 480.0f, 480.0f);
    flippedView = [[FlipView alloc] initWithFrame:fliprect];
    [flippedView setTransform:CGAffineTransformMakeRotation(3.141592f / 2.0f)];
    [flippedView setUserInteractionEnabled:YES];

    // Initialize
    cfView = [[CoverFlowView alloc] initWithFrame:[UIScreen mainScreen]
    ➤applicationFrame] andCount:[titles count]];
    [cfView setUserInteractionEnabled:YES];
    [cfView setHost:self];
    cfLayer = [cfView cfLayer];
    label = [cfView label];

    // Finish setting up the Cover Flow layer
    whichItem = [titles count] / 2;
    [cfLayer selectCoverAtIndex:whichItem];
    [cfLayer setDelegate:self];

    selector = NULL;
    target = NULL;

    return self;
}

// *****
// Cover Flow delegate methods
//

```

```

- (void) coverFlow: (id) coverFlow selectionDidChange: (int) index
{
    whichItem = index;
    [label setText:[titles objectAtIndex:index]];
}

// Detect the end of the flip - both on reveal and hide
- (void) coverFlowFlipDidEnd: (UICoverFlowLayer *)coverFlow
{
    if (flipOut)
        [[[UIApplication sharedApplication] keyWindow] addSubview:flippedView];
    else
        [flippedView removeFromSuperview];
}

// *****
// Cover Flow datasource methods
//

- (void) coverFlow:(id)coverFlow requestImageAtIndex: (int)index quality:
    (int)quality
{
    UIImage *whichImg = [covers objectAtIndex:index];
    [coverFlow setImage:[whichImg CGImage] atIndex:index type:quality];
}

// Return a flip layer, one that preferably integrates into the flip presentation
- (id) coverFlow: (UICoverFlowLayer *)coverFlow requestFlipLayerAtIndex: (int)
    index
{
    if (flipOut) [flippedView removeFromSuperview];
    flipOut = !flipOut;

    // Prepare the flip text
    [flippedView setText:[NSString stringWithFormat:@"%s\n",
        [titles objectAtIndex:index], [colorDict
        objectForKey:[titles objectAtIndex:index]]];
    // Flip with a simple blank square
    UIView *view = [[UIView alloc] initWithFrame:CGRectMake(0.0f, 0.0f, 140.0f,
        140.0f)];
    [view setBackgroundColor:[UIColor clearColor]];

    return [view layer];
}

// *****
// Utility methods
//

- (CoverFlowView *) cfView {return cfView; }
- (UICoverFlowLayer *) cfLayer {return cfLayer; }

```

```
- (int) selectedItem {return whichItem; }

- (void) start
{
    [cfView startHeartbeat: @selector(tick) inRunLoopMode:
    ➡(id)kCFRunLoopDefaultMode];
    [cfLayer transitionIn:1.0f];
}

- (void) stop
{
    [cfView stopHeartbeat: @selector(tick)];
}

- (void) loadView
{
    [super loadView];
    self.view = cfView;
    [self start];
}

// *****
// Callback method for double tap
//
- (void) doubleTapCallback
{
}

@end
```

11.4 小结

通过阅读本章可以看出，iPhone编程中一些非常精致的细节都包含在公共的iPhone框架中，而不在正式的SDK中。苹果公司这种非正式策略的意图很明显：你可以在程序中使用这些项目，但是这样做存在一定的风险。在每个固件版本中，你的代码都可能会中断。这就意味着你需要在风险和回报之间寻找最佳的平衡点。