



内容全面，系统讲解开发企业级iOS应用所需掌握的各项核心技术，以及各种工具和框架的用法，包含大量技巧和最佳实践

实战性强，不仅为各个知识点精心设计了能辅助读者理解的小案例，而且还有能指导读者完整实践的大案例，具备极强的可操作性



杨宏焱 著

*Enterprise Application Development for iOS*

# 企业级iOS应用 开发实战



附光盘



机械工业出版社  
China Machine Press

# 企业级 iOS 应用开发实战

杨宏焱 著



## 图书在版编目 (CIP) 数据

企业级 iOS 应用开发实战 / 杨宏焱著. —北京：机械工业出版社，2012.12

ISBN 978-7-111-40459-0

I. 企… II. 杨… III. 移动电话机—应用程序—程序设计 IV. TN929.53

中国版本图书馆 CIP 数据核字 (2012) 第 274687 号

### 版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问 北京市展达律师事务所

本书内容全面，它不仅详细讲解了开发企业级 iOS 应用所需掌握的各项核心技术，以及各种工具和框架的用法，而且还系统讲解了企业级 iOS 应用开发的流程和方法；实战性强，不仅为各个知识点精心设计了能辅助读者理解的小案例，而且还有能指导读者进行完整实践的大案例，具备极强的可操作性。除此之外，本书还包含大量的开发技巧和最佳实践。

本书分为三部分：基础篇（1~6 章），首先介绍了传统企业级应用与 iOS 企业级应用的区别、iOS 企业级应用程序的架构以及发布方法，然后详细讲解了 iOS 的开发框架、Objective-C 语法的核心要素、Xcode 集成开发环境、Interface Builder 和高级图形界面；核心技术篇（7~17 章），系统深入地讲解了网络、XML 和 JSON、用户数据保存、安全、多媒体、绘图、动画、多点触摸和手势、GPS、重力感应、本地化、多线程、并行编程、通知、通讯簿等与企业级应用相关的核心技术特性，同时也讲解了开源框架 CorePlot；实战篇（18~19 章）以迭代的方式讲解了两个综合案例的完整实现过程，既融合了前面的理论知识，又展现了企业级 iOS 应用开发的流程和方法。

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）

责任编辑：吴 怡

印刷

2013 年 1 月第 1 版第 1 次印刷

186mm×240mm · 26 印张

标准书号：ISBN 978-7-111-40459-0

ISBN 978-7-89433-713-9 (光盘)

定 价：69.00 元 (附光盘)

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88378991 88361066

投稿热线：(010) 88379604

购书热线：(010) 68326294 88379649 68995259

读者信箱：hzjsj@hzbook.com



## 为什么写这本书

随着我国 3G 网络和移动互联网的兴起，许多传统的企业应用正在从桌面向移动终端扩展，移动办公、移动营销、移动作业等需求日渐强烈。

有迹象表明，传统的互联网正在向移动互联网发展。根据摩根士丹利发布的全球互联网发展趋势报告（Mary Meeker 2010）显示：全球互联网发展趋势正在由 PC-Internet 向 Mobile-Internet 转变，手机在某种意义上已经主导着互联网的发展，新兴的下一代互联网，即 Mobile 2.0 正在崛起，这完全得益于移动通信技术的迅猛发展。这不仅仅是一场由最新数码科技与网络技术导致的变革，还是传统企业应用从 Internet 向移动互联网转移的前提和诱因。

与 PC 相比，移动终端的这种全新用户体验是不可替代的，在一定程度上吸引了人们从桌面向移动终端的转移。而且，移动互联网也凭借着其出色的业务吸引力和资费吸引力，成为人们生活中不可或缺的一部分。

然而，机遇与挑战并存。对于企业而言，能否将自己的企业应用向移动互联网扩展，仍然存在着巨大的风险。诸如：用户体验改变、企业信息安全和企业机密泄露、移动应用开发中存在的技术风险等。

以苹果 iOS 为代表的移动应用开发正方兴未艾。iPhone 和 iPad 正式进入中国的时间其实还不到 3 年（iPhone 于 2010 年 7 月正式在中国香港上市），国内开发者在苹果商店上淘金的时间比这要早些，但绝对不会超过 4 年。实际上，App Store 拥有的历史还不到 5 年（App Store 正式上线时间是 2008 年 7 月 11 日）。所以说，iPhone 应用开发仍然有着无限的潜力，称为“历

史”恐怕为时尚早。因此，作者选择了以 iOS 为目标平台的企业移动应用开发作为本书讲述的主题。

作者于 2009 年起开始接触 iOS 开发。对于一个多年奋战在企业应用开发第一线的开发人员来说，iOS 企业开发是一个全新的领域。完全陌生的 Mac OS X 操作系统，别扭的 Xcode IDE 和 Interface Builder，古怪的 iOS……由于不知道什么是开发证书和代码签名，甚至在第一次调试我的 3.5 英寸屏幕的 iTouch 时，都是充满了坎坷。

当我历尽千辛万苦，终于逐渐步入 iOS 开发这座大门。然而，却又面临新的问题：“企业应用是否能向 iOS 平台进行迁移？”

从我第一天接到任务起，这样的怀疑就在我心中存在。iOS 实在是太封闭了，相对于 Android 这样的开放平台，iOS 平台对企业开发人员的限制实在是太多。而且，App 商店始终只是游戏开发者的天堂，90%以上的个人开发者把自己的目光盯在了游戏、娱乐等个人应用领域，企业应用根本无法登上 App 商店的大雅之堂。原因也很好理解，企业应用不会为苹果公司带来可观的利润分成，App 商店的盈利模式是基于应用下载量的。一个企业应用会有多少用户？几百万？几千万？不，大多数企业应用的用户量不会超过 5 位数。没有庞大的用户群，就不会给 App 商店带来丰厚的利润分成。实际上，我第一次向 App 商店提交一个企业应用时，被无情地拒绝了。苹果公司给的理由很直白：“你的应用只针对有限的用户群”，换句话说“苹果将无利可图”。

幸好苹果公司提供了“企业开发程序”（企业版 IDP），虽然购买“企业开发程序”需要 299 美金一年，但对于一个真正决心将企业应用向 iOS 移动终端扩展的企业来说，还是负担得起的。“企业开发程序”不需要苹果公司审核，使用“企业开发程序”部署应用不需要经过 App 商店，企业可以任意分发给自己的用户，苹果公司也不会找你要一分钱。

可以预料，在将来一段时间内，国内的 iOS 个人应用开发者将不断向企业开发领域转移。实际上，苹果商店中个人应用的数量已经饱和，开发者的生存空间将逐渐变得狭小。大量的同质竞争直接导致了 App 商店的生态环境恶化，同类应用竞争激烈。为了保持 App 商店的竞争力和盈利模式不趋于低质化，苹果公司今后对商店应用的审核将越来越趋于严格，开发者想在商店中获利和生存的难度将越来越高。国内的公司及个人开发者会逐渐将目光转移到位于 App 商店生态圈之外的企业应用，iOS 企业应用将成为今后新的利润增长点。

最近，企业移动应用开发出现了一些新的趋势。最新的企业移动应用，有从 Native App（即本地代码）向云发展的趋势，比如 HTML5、虚拟桌面。HTML 5 充分利用移动终端的浏览器（如 Safari 或 IE）和网络连接能力，来访问企业服务，并实现“一次开发，跨平台共享”的目的。而虚拟桌面则利用“数据中心”进行桌面的扩展，将客户端的数据、资源和图像放到了“云”上，iOS 客户端则通过网络访问个人桌面。虚拟桌面一般作为企业的“云计算”解决方案进行实施，市场上比较成熟的产品主要来自 VMWare、Citrix、Microsoft 和 Oracle 等几大厂商，它跟开发人员没有太大的关系。还有一种趋势就是“服务器配置+中间代码+本地代码”。开发人员在服务器上以配置的方式产生出中间代码，然后服务器将中间代码编译为多个平台的本地代码（iOS、Android、Sybian、Windows），然后分别部署，以此实现跨平台的目的。

这些趋势的出现，从一定程度上试图解决当前本地代码开发（iOS 开发、Android 开发）的弊端，但效果还难以令人满意，比如都有性能下降、用户体验差、网络带宽占用大等缺点。因此，就目前来说，移动应用开发仍然是以本书介绍的本地代码开发为主流。

## 本书特色

本书是作者多年开发经验的总结，很多内容来自作者在 CSDN 上的博客，书中不少内容是经验之谈。本书根据 iOS 操作系统更新频繁的实际情况，针对新的 SDK 版本进行了内容上的调整（本书内容适用于 SDK 4.0~5.0，本书所有代码在 Xcode4.3 下编译通过）。在介绍每一种 SDK 框架的同时，注重扩展，在继承的基础上进行创新，而不是一味复制、粘贴代码。

本书具有如下特点：

- 主题明确，以“iOS”和“企业开发”为主题，但并没有将二者割裂开来，而是将二者紧密联系、互相呼应。首先由浅入深介绍了整个 SDK 框架层次，包括 Objective-C 语言简介、SDK 的构成、Foundation 框架、UIKit 框架、QuartzCore、CoreAnimation 以及其他第三方扩展框架，然后对在企业应用中一些需要特别讨论的方面（如安全、网络、APN、多线程等内容）进行专门的论述。撇开企业开发的特色不谈，本书也完全可以作为一本 iOS 开发的经典教材。
- 理论和技术兼顾。许多 iOS 开发书籍，轻理论，重技术，往往只告诉你怎么做，而不告诉你为什么要这样做，难以让读者在理解的基础上加深记忆。而本书以理论为纲，以技术为体，从基本理论到具体使用的技术都一一道来，不仅告诉你怎么做，而且将每一种技术的来龙去脉阐述清楚。在讲解具体技术的同时，不时穿插着小的知识点，让读者进一步拓宽相关的背景知识。
- 详细分析代码，实用性强。作为编程类书籍，免不了有大量的代码。但本书对多数代码都进行了阐释，重点内容还会有专门的标注，如“提示”、“注意”等，以提醒读者注意，或者及时回顾前面的知识点。本书中的每一个示例程序，都收录到本书的随书光盘中。所有的程序都经过作者认真调试，可以直接运行。

## 合适阅读本书的人

本书适用于以下读者：

- 从未接触过 Objective-C、从其他语言转向 iOS 开发、有一定面向对象编程基础的程序员。
- 正准备转向企业移动应用开发的 iOS 应用程序开发人员。

## 如何阅读本书

这是一本讲述 iOS 和企业应用开发相结合的书，介绍如何在 iOS 上进行企业应用的开发

及分发、部署。本书从一个企业应用开发者的角度出发，以实现企业移动办公和 3G 应用为宗旨，介绍如何充分发挥苹果新一代操作系统 iOS 的优势和 iPhone 手机的软、硬件特性将企业应用扩展到 iOS 平台。在最后一章以 step by step 的形式介绍了一个实战项目，以达到理论与实践结合的目的。本书也对苹果 Cocoa 框架和其他第三方开源框架进行了深入介绍。

针对本书面对的两种主要读者，我们建议如下：

对于本书第一类读者，即“从未接触过 Objective-C、从其他语言转向 iOS 开发、有一定面向对象编程基础的程序员”，本书提供了一个快捷的 Objective -C 语言入门，以及一个简单易读而又务实详尽的 iOS SDK 入门教程；本书的全部章节都将有助于读者尽快对 Objective -C 及 iOS SDK 有一个全面的了解，并迅速跨入 iOS 开发的大门。

对于本书第二类读者，即“正准备转向企业移动应用开发的 iOS 应用程序开发人员”，可以省略阅读本书部分章节，例如第 2、3、4、5、6 各章，但本书其他一些章节具备了良好的参考价值，例如：第 10 章以后的各章，在这些章节中，有部分内容是其他参考书中难以见到的，可以有选择地阅读相应章节。

本书共分 19 章，主要内容如下：

### 基础篇

第 1 章介绍了企业应用的概念，什么是 iOS 企业应用，iOS 企业应用的框架及构成，特别是对于苹果 iOS 企业证书申请和 iOS 企业应用程序的部署方式（In-House、Ad-Hoc、OTA）进行了详细的介绍。

第 2 章介绍 iOS SDK，包括其框架和构成。iOS SDK 是 iOS 开发中最为重要的工具和武器，每个 iOS 开发人员都必须熟悉并深刻理解它。

第 3 章介绍 iOS 开发语言 Objective-C。对于没有接触过这种语言的读者，将在本章对 Objective-C 有一个全面的理解。本章从两个方面对 Objective-C 进行了介绍，即 Objective-C 的 C 语言特性和面向对象特性。也对 Objective-C 的一些现代语言特性，如块编程（函数式编程中的主要内容）、反射（运行时支持）和可变参数也进行了介绍，这些内容在其他书籍中是比较罕见的。

第 4 章介绍 Xcode IDE。从 Xcode 4.0 开始，苹果对其功能和界面进行了全新的设计，把 Interface Builder 完全整合到 Xcode 中，使程序员的开发效率更高。

第 5 章单独对 Xcode 中的 Interface Builder 进行了进一步介绍，特别是 Assistant Editor 的出现，与之前的版本相比，大大简化了开发人员进行各种连接（IBOutlet 和 IBAction）的操作。

第 6 章介绍 UIKit 以及 UIKit 中包含的一系列最基本的 UI 组件，此外，介绍了如何在 UIKit 的基础上进行扩充，创建自己的自定义组件库。

### 企业应用篇

第 7 章到第 10 章，依次从网络、XML/Json、数据存储、安全这几个方面进行介绍。这些内容中，有相当一部分是企业开发人员早已熟知的领域（如网络、XML/Json、数据存储和安全）。这些章节结合 iOS 自身的特点进行详细的阐述，包含安全沙箱、嵌入式数据库以及 iOS

安全框架等内容。

第 11 章介绍 Cocoa 的多媒体、Quartz 2D 和 Core Animation 框架。

第 12 章介绍 Cocoa Touch 特有的多点触摸和手势识别。

第 13 章介绍如何利用 iPhone 的多语言支持实现应用程序的国际化。

第 14 章涉及两个方面：传统的线程编程和并行编程 GCD ( Grand Central Dispatch )。在企业应用中，免不了要使用多线程。前者是传统的异步编程技术，直接与操作系统底层的线程打交道；后者是 iOS 4.0 以后新的异步编程技术，以一种函数式编程的方式，达到让系统自动进行线程管理的目的，从而避开了线程编程的复杂性。

第 15 章介绍通知、本地通知和远程通知。通知是多个对象间进行对话的机制，但耦合性低于直接的方法调用。本地通知和远程通知是两种不同的进程唤醒技术，前者由系统来唤醒，后者通过 RPC ( Remote Process Calling ) 唤醒。

第 16 章介绍开源框架 Core Plot。Core Plot 是著名的 2D 图形框架，用于绘制散点图、柱状图和饼图等图表。

第 17 章针对 iOS 特有的硬件特性进行介绍，如通讯簿、相机、加速计和 GPS。

### 实战篇

第 18 章，介绍“企业 APN”在企业中的应用，以及使用“企业 APN”网络对 iOS 客户端的一些特殊要求。该章实际上包含了一个实战项目，即一个简单的 APN 切换工具（同时也提供了简单的网络状态检测）。在这个实战项目中，涉及了广泛的内容和前面诸多章节中介绍的知识，诸如后台任务、配置描述文件、BSD Socket 编程、网络检测、Safari 阻塞和并行编程 GCD。

第 19 章以案例导航的方式介绍了一个实战项目，指导读者从用户的需求出发，结合本书中讲述过的理论知识和技术，开发一个完整的 iOS 邮件客户端，使读者对企业应用的开发有直观的认识。

## 结语

本书从开始选题、写作至最终定稿，总共花费了作者 14 个月以来所有的业余时间。在本书写作的过程中，得到了许多人的无私帮助和支持。

首先要感谢的是吴怡编辑。她负责本书所有文字内容、图片及格式，她总是不厌其烦地向作者指出本书中的每一处错误。直至最终定稿前，她仍然和作者对书中的内容进行讨论，甚至是逐字逐句地讨论，以使本书不至于出现太多谬误或引发读者产生歧义。

其次要感谢我的同事们。在他们身上，我学习到了许多。我在工作中取得的每一点进步，都离不开他们的支持和指导，他们敬业的态度和对我的无私帮助，是本书得以成书的动力。

还要感谢我的家人。本书是在他们的关怀和支持下才得以面世的，尤其是我的妻子，因为她的理解和默默奉献，使作者从家庭琐事中解脱出来，全身心投入本书的写作当中。

我要感谢始终关注作者博客、不断给作者以鼓励的网友们。虽然直至本书出书之时，我都向他们隐瞒了本书的消息，但他们留下的只言片语，仍然激励着我在知识的海洋中不断前行、探索和攀登。

最后，感谢所有在本书写作过程中，给本书以参考的文献作者和论坛作者。本书参考了大量文献，尤其是苹果文档参考库、苹果开发者论坛和 stackoverflow.com，它们对本书的写作起到了重要的作用。

由于作者的时间、精力及自身水平有限，书中错误在所难免，欢迎广大读者一一指正。作者的联系方式如下：

博客：[blog.csdn.net/kmyhy](http://blog.csdn.net/kmyhy)

邮箱：kmyhy@126.com。



## 前言

## 基础篇

### 第 1 章 企业应用的话题 / 2

- 1.1 什么是企业应用 / 2
  - 1.1.1 传统意义的企业应用 / 2
  - 1.1.2 iOS 企业应用 / 3
- 1.2 iOS 企业应用程序的架构 / 3
  - 1.2.1 服务端 / 4
  - 1.2.2 iOS 客户端 / 4
- 1.3 iOS 企业应用程序的发布 / 5
  - 1.3.1 iOS 应用程序发布与 App Store / 5
  - 1.3.2 Ad-Hoc 与 In-House 发布 / 6
  - 1.3.3 OTA 无线部署 / 21

### 第 2 章 iOS 开发框架简介 / 24

- 2.1 苹果 iOS 简介 / 24
- 2.2 iOS 框架介绍 / 25
- 2.3 Cocoa Touch 框架简介 / 25

- 2.4 搭建 iOS 开发环境 / 27
  - 2.4.1 安装 Mac OS X 操作系统 / 27
  - 2.4.2 下载安装 SDK / 33
- 2.5 写一个 iPhone 程序 / 33
- 2.6 在模拟器上运行应用程序 / 39
- 2.7 在 iPhone 上运行应用程序 / 39

## 第 3 章 Objective-C 语法简介 / 42

- 3.1 Objective-C 的 C 语言特性 / 42
  - 3.1.1 一个简单的 Hello World / 42
  - 3.1.2 Objective-C 是另一种 C / 43
  - 3.1.3 数据类型 / 44
  - 3.1.4 常量、变量和宏 / 50
  - 3.1.5 #include 和#import / 51
  - 3.1.6 函数 / 51
  - 3.1.7 分支和循环 / 51
- 3.2 面向对象的 C / 51
  - 3.2.1 类和对象 / 51
  - 3.2.2 消息机制 / 54
  - 3.2.3 Objective-C 的内存管理 / 55
  - 3.2.4 类别和协议 / 57
  - 3.2.5 反射机制 / 59
  - 3.2.6 谓词 / 62
- 3.3 MVC 模式 / 65
- 3.4 KVO 模型 / 65
  - 3.4.1 注册 KVO / 66
  - 3.4.2 接收变更通知 / 67
  - 3.4.3 发送变更通知 / 67
- 3.5 块编程 / 68
  - 3.5.1 块的特点 / 68
  - 3.5.2 Objective-C 中的块 / 69
- 3.6 可变参数 / 71
- 3.7 本章小结 / 73

## 第 4 章 Xcode 集成开发环境 / 74

- 4.1 创建第一个 Xcode 应用程序 / 74
- 4.2 构成应用程序的那些东西 / 76
  - 4.2.1 Info.plist 和 pch 文件 / 76
  - 4.2.2 Xib 文件 / 77

4.2.3	资源文件 / 77
4.2.4	源代码文件 / 77
4.2.5	项目和目标 / 77
4.2.6	Frameworks / 80
4.2.7	应用程序的文档目录和临时文件夹 / 81
4.3	了解 Xcode 为我们做了些什么 / 83
4.3.1	main.m / 83
4.3.2	应用程序委托 / 84
4.4	在 Xcode 中添加 View Controller / 84
4.5	在 Xcode 中添加框架 / 89
4.6	Xcode 使用技巧 / 90
4.6.1	自动完成 / 90
4.6.2	查找和替换 / 91
4.6.3	快速帮助 / 91
4.6.4	快照 / 91
4.6.5	书签 / 91
4.6.6	使用导航条 / 92
4.7	本章小结 / 92

## 第 5 章 Interface Builder / 93

5.1	IB 和 xib、nib 文件 / 93
5.2	初识 IB / 94
5.3	使用 IB 创建图形界面 / 95
5.3.1	控制器和视图 / 95
5.3.2	基本控件介绍 / 99
5.4	连接 / 100
5.4.1	IBOutlet 连接 / 100
5.4.2	IBAction 连接 / 102
5.4.3	委托连接 / 103
5.4.4	使用 Assistant Editor 创建连接 / 105
5.5	本章小结 / 106

## 第 6 章 高级图形界面 / 107

6.1	应用程序多视图的导航 / 107
6.1.1	UITabBarController / 107
6.1.2	UINavigationController / 110
6.1.3	窗体导航应用实例 / 114
6.2	表视图 UITableView 的应用及其扩展 / 116
6.2.1	简单的表视图控制器 / 116

6.2.2	UITableView 的数据源和委托 / 117
6.2.3	分组表视图 / 119
6.2.4	可折叠的分组表视图 / 121
6.3	扩展 UIKit / 131
6.3.1	扩展日期挑选控件 / 131
6.3.2	扩展单选按钮和复选按钮 / 133
6.3.3	扩展下拉列表框 / 135
6.3.4	封装自己的控件库 / 137
6.4	翻页控件和翻页控制器 / 142
6.4.1	UIPageControl / 143
6.4.2	UIPageViewController / 147
6.5	本章小结 / 152

## 企业应用篇

### 第 7 章 网络 / 154

7.1	使用 NSURLConnection 获得网络数据 / 154
7.2	使用 NSOperation 进行异步请求 / 158
7.3	与网络相关的示例 / 163
7.4	ASIHTTPRequest 框架介绍 / 166
7.4.1	发送同步请求 / 167
7.4.2	发送异步请求 / 168
7.4.3	文件上传 / 169
7.4.4	文件下载 / 172
7.4.5	Cookies 和 Sessions / 176
7.5	编写自己的网络模块类 / 179
7.5.1	PostRequest 类 / 179
7.5.2	NetworkModule 类 / 181
7.5.3	测试 NetworkModule / 185
7.6	本章小结 / 186

### 第 8 章 XML 和 Json / 188

8.1	Cocoa 与 XML 解析 / 188
8.1.1	NSXMLParser / 188
8.1.2	NSXMLParserDelegate / 189
8.2	TBXML / 190
8.3	libxml / 191

8.3.1 在项目中使用 libxml / 192
8.3.2 libxml 应用实例 / 192
8.4 GDataXML / 202
8.5 Json 和 SBJson / 218
8.5.1 在项目使用 SBJson / 218
8.5.2 SBJson 使用示例 / 218
8.6 本章小结 / 219

## 第 9 章 保存用户数据 / 220

9.1 文件的持久化 / 220
9.1.1 保存到 plist 文件 / 220
9.1.2 NSUserDefaults / 221
9.1.3 归档 / 224
9.2 数据库 / 226
9.2.1 嵌入式数据库 SQLite3 / 226
9.2.2 使用 Core Data / 228
9.2.3 使用 PLDatabase 访问数据库 / 232
9.3 本章小结 / 236

## 第 10 章 安全 / 237

10.1 iOS 安全框架简介 / 237
10.1.1 证书、密钥和信任服务 / 237
10.1.2 在 iPhone 中使用 X.509 证书 / 238
10.2 使用 SSL 和服务器通信 / 244
10.3 OpenSSL / 245
10.3.1 在 iOS 中使用 OpenSSL 库 / 245
10.3.2 OpenSSL 应用实例——使用 OpenSSL 进行 MD5 加密 / 248
10.4 CommonCrypto / 250
10.5 本章小结 / 252

## 第 11 章 多媒体、绘图及动画 / 253

11.1 播放视频 / 253
11.2 播放音频 / 254
11.3 Quartz 2D / 255
11.3.1 图形上下文 / 255
11.3.2 路径 / 256
11.3.3 变换 / 257
11.3.4 图案 / 261
11.3.5 阴影 / 262

- 11.3.6 演变 / 263
- 11.3.7 透明图层 / 264
- 11.3.8 位图及遮罩 / 264
- 11.4 Core Animation / 267
  - 11.4.1 隐式动画 / 267
  - 11.4.2 显式动画 / 268
- 11.5 本章小结 / 269

## 第 12 章 多点触摸及手势 / 270

- 12.1 手势识别器: UIGestureRecognizer 类 / 270
- 12.2 创建手势识别器 / 272
- 12.3 实现图片的拖动及缩放 / 276
- 12.4 本章小结 / 279

## 第 13 章 本地化 / 280

- 13.1 iPhone 的本地化支持 / 280
  - 13.1.1 国家代码和语言代码 / 280
  - 13.1.2 本地化文件夹的匹配 / 281
- 13.2 本地化应用程序 / 281
  - 13.2.1 使用 NSLocalizedString 本地化字符串 / 281
  - 13.2.2 本地化图像 / 285
  - 13.2.3 本地化 xib 文件 / 285
  - 13.2.4 本地化应用程序名称 / 285
- 13.3 示例 / 285
- 13.4 本章小结 / 289

## 第 14 章 iOS 多线程和并行编程 / 290

- 14.1 多线程 / 290
  - 14.1.1 NSThread / 291
  - 14.1.2 RunLoop / 293
- 14.2 并行编程 / 296
  - 14.2.1 Dispatch Queue / 296
  - 14.2.2 将任务加入 Dispatch Queue / 297
  - 14.2.3 Dispatch 源 / 298
- 14.3 后台任务 / 301
- 14.4 本章小结 / 303

## 第 15 章 通知、本地通知和远程通知 / 304

- 15.1 通知 / 304
- 15.2 本地通知 / 307

- 15.3 远程通知 / 315
  - 15.3.1 Apple Push 简介 / 316
  - 15.3.2 准备使用 APNs / 316
  - 15.3.3 准备接收推送通知 / 320
  - 15.3.4 创建 Push Notification Provider / 322
- 15.4 本章小结 / 325

## 第 16 章 开源框架 Core Plot / 327

- 16.1 编译 Core Plot 框架 / 327
- 16.2 使用 Core Plot SDK / 327
- 16.3 安装 Core Plot 帮助文档 / 328
- 16.4 图表的构成 / 329
- 16.5 类图 / 330
- 16.6 使用 Core Plot 绘制折线图 / 331
- 16.7 使用 Core Plot 绘制柱状图 / 335
  - 16.7.1 绘制基本的柱状图 / 335
  - 16.7.2 固定坐标轴 / 336
  - 16.7.3 显示数据点的值 / 338
  - 16.7.4 显示网格线 / 339
- 16.8 使用 Core Plot 绘制饼图 / 339
  - 16.8.1 饼图的绘制 / 340
  - 16.8.2 显示每个扇形的比例 / 341
  - 16.8.3 剥离扇形 / 341
  - 16.8.4 显示图例 / 342
  - 16.8.5 响应事件 / 343
- 16.9 自定义 Core Plot 主题 / 343
- 16.10 本章小结 / 346

## 第 17 章 通讯簿、GPS 和重力感应 / 347

- 17.1 通讯簿 / 347
  - 17.1.1 Address Book UI / 347
  - 17.1.2 Address Book / 348
  - 17.1.3 联系人中文姓氏排序 / 350
- 17.2 GPS 和 CoreLocation / 351
- 17.3 重力感应 / 353
- 17.4 地理编码 / 355
- 17.5 本章小结 / 356

## 实战篇

### 第 18 章 企业 APN / 358

- 18.1 企业 APN 的建设 / 358
- 18.2 iPhone 与 APN / 359
- 18.3 配置描述文件 / 360
- 18.4 在 iPhone 上实现一个 HTTP 服务器 / 362
- 18.5 后台任务与无限后台任务 / 365
- 18.6 实现 APN 切换 / 368
- 18.7 检测网络状况 / 369
- 18.8 Safari 阻塞 / 373
- 18.9 本章小结 / 377

### 第 19 章 iOS 企业应用实战 / 378

- 19.1 应用场景与功能概述 / 378
- 19.2 应用程序架构 / 378
- 19.3 服务器端 / 378
  - 19.3.1 环境搭建 / 378
  - 19.3.2 实现登录接口 / 379
  - 19.3.3 实现企业通讯簿接口 / 379
  - 19.3.4 实现收件箱接口 / 380
  - 19.3.5 实现附件上传接口 / 380
  - 19.3.6 实现附件下载接口 / 380
- 19.4 iPhone 客户端 / 381
  - 19.4.1 实现登录 / 381
  - 19.4.2 查看收件箱 / 383
  - 19.4.3 邮件浏览 / 387
  - 19.4.4 新建邮件 / 389
  - 19.4.5 正文输入界面 / 391
  - 19.4.6 通讯簿 / 392
  - 19.4.7 附件文件的上传 / 397
- 19.5 本章小结 / 399

# 第 1 章 企业应用的话题

本书是一本关于 iOS 企业应用开发的书。在本书开篇，首先讨论一下企业应用的话题。包括：什么是企业应用、iOS 企业应用、iOS 企业应用中所使用的应用程序发布方式 Ad-Hoc 和 In-House，以及 iOS 4.0 以后新增的无线部署功能。

## 1.1 什么是企业应用

iPhone 开发是一个新兴的话题，对于“企业应用”和“非企业应用”，它并没有很清晰的划分。这里借用了传统意义上的企业应用概念，试图阐述清楚如何区分 iOS 企业应用，以及 iOS 企业应用的定义。

### 1.1.1 传统意义的企业应用

据 IDC 统计，在过去的 10 年中，全球企业在信息系统上一共投资 18 万亿美元。巨大的投资为企业建立了众多信息系统，以帮助企业进行内外部业务的处理和管理工作。根据 METAGroup 的统计，一家典型的大型企业平均拥有 49 个应用系统。虽然迄今为止，“企业应用”都没有一个明确的定义，笔者认为企业应用是企业环境中的特定系统，例如：

ECS（电子商务系统），企业通过实施电子商务实现企业经营目标，电子商务系统提供了网上交易和管理等全过程的服务，如网上订购、网上支付、电子账户、服务传递、意见征询、业务管理等各项功能。

ERP（企业资源规划）系统，指建立在信息技术基础上，以系统化的管理思想，为企业决策层及员工提供决策运行手段的管理平台。企业通过企业资源规划系统能实现企业供应链管理、精益制造、敏捷制造以及整个生产过程的计划、控制、采购、销售、成本核算的管理目标。

CRM（客户关系管理）系统，企业利用信息技术（IT）和互联网技术实现对客户的整合营销，是以客户为核心的企业营销的技术实现和管理实现。客户关系管理注重的是与客户的交流，企业的经营是以客户为中心，而不是传统的以产品或以市场为中心。

OA（办公自动化）系统，它是利用计算机技术提高办公效率，实现办公自动化处理的系统。它采用 Internet/Intranet 技术，和工作流的概念，使企业内部人员能方便快捷地共享信息，协同工作，提升日常办公的工作效率，并为企业的管理和决策提供帮助。

DBS（数据库系统），是企业信息化的核心，负责整个企业在经营过程中的数据储存、共享和处理，为其他信息系统提供支撑。

DW（数据仓库）是在数据库已经大量存在的情况下，为了进一步挖掘数据资源、为了决

策需要而建设的数据仓库。数据仓库系统是一个信息处理平台，它从业务处理系统获得数据，并处理数据，从而获得战略信息。

由此可见，只要是在企业信息化环境中运行的应用软件，都可以称为企业应用。

### 1.1.2 iOS 企业应用

根据摩根士丹利发布的全球互联网发展趋势报告（Mary Meeker 2010）显示：全球互联网发展趋势正在由 PC-Internet 向 Mobile-Internet 转变，手机在某种意义上已经主导着互联网的发展，移动互联网将带来很多新的商业机会。新兴的下一代互联网，即 Mobile 2.0 正在崛起，同样带来令人刺激的软件行业商业机会。

此外，2008年底中国3G牌照正式发放，标志着移动通信3G时代终于来临。移动通信网络由2G/2.5G向3G的过渡，为移动互联网绑上了高速发展的助推器。对国内软件开发商而言，这意味着新的机遇和挑战产生了。

根据工业和信息化部网站发布的数据（中国工业和信息化部 2010），随着中国电信3G用户数达到1000万、TD用户数达到1698万、中国联通3G用户数达到1166万，目前我国三家电信企业的3G用户数均过千万。截至10月底，我国3G用户数累计达到3864万，环比增长10.4%，同比增长295.7%，比2009年年底增长2538万，10月新增用户364.6万户。TD用户在3G用户中的占比达到43.9%。

与传统的2G和2.5G网络相比，3G网络带宽已高达300~600kb/s，比之512kb/s的ADSL也相差无几，因此诸多应用不再受到带宽限制，诸如：移动办公、个人应用、移动金融、GPS导航、视频通话，甚至是传统的企业应用CRM、ERP，也可能运行在手机上。

2007年1月的Macworld年度大会上，苹果公司发布了令人期待已久的iPhone手机。iPhone将创新的移动电话、可触摸宽屏iPod以及具有桌面级电子邮件、网页浏览、搜索和地图功能的突破性因特网通信设备这三种产品完美地融为一体，引入了基于大型多触点显示屏和领先性新软件的全新用户界面，让用户用手指即可控制iPhone。iPhone还开创了移动设备软件尖端功能的新纪元，重新定义了移动电话的功能。

全球互联网向移动互联网的迁移，3G网络的兴起，新一代智能手机产品尤其是iPhone在全球市场中受到热烈追捧，导致企业应用正呈现由传统internet/intranet向移动网络/手机终端扩张的趋势。iOS正是苹果公司为其创新性产品iPhone开发的新一代手机操作系统，iOS企业应用的概念，也因此衍生而来。

## 1.2 iOS 企业应用程序的架构

本书把iOS企业应用定义为传统企业应用向iOS手机终端的顺延和扩张。在此定义下，iOS企业应用由服务端和iOS客户端构成，二者间通过3G移动互联网（CDMA/TD/WCDMA）连接或通信。

### 1.2.1 服务端

服务端（企业网络或 Web 服务）实际上为 iOS 企业应用提供企业数据和服务。如果把 iOS 客户端看做是前端应用，则服务端就是后台服务。服务端向前端提供一系列访问传统企业应用的接口，也可以为前端提供企业数据库和业务系统的访问。因此，iOS 企业应用的服务端可能有两层或多层：接口、企业应用、企业数据库。

本书的核心内容是介绍 iOS 开发技术，不会对企业开发技术做过多的介绍。因此服务端代码（企业应用和企业数据库）的开发细节不会在本书中出现，但对于本书中涉及的接口，会提供必要的代码给读者学习。此外，本书中的接口代码是以 Java 编写的，需要读者对 Java 语言有一定的了解。

### 1.2.2 iOS 客户端

iOS 客户端是一个标准的 iOS 应用，当然它也具备一些企业应用所特有的特点。但无论如何，它不应当是在浏览器中运行的 Web 网页。如果你想找一本介绍如何开发在 iPhone 浏览器上运行的 Web 网页应用程序的书，那么不应该是本书。

本书大部分内容旨在教你开发标准的 iOS 应用程序，这与市面上大部分介绍 iPhone 开发的书籍是一致的，但有一些例外。

首先，作为运行在手机上的 iOS 企业应用来说，安全是尤其需要注意的问题。因为 iPhone 等手持式移动终端所特有的一些特点，比如随身携带、随处可用，不需要登录，容易丢失等等，稍有不慎，就有可能导致企业机密的泄漏。

其次，对于企业应用来说，访问网络的需要，尤其是访问企业网络内部资源，如服务器、数据库等，永远是必不可少的重要内容。无论在任何情况下，网络带宽永远是企业的稀缺资源，对于企业应用尤其如此，因此，必须在节省带宽和提高用户体验中进行平衡。本书使用了很大的篇幅来介绍网络访问技术，此外，企业网络的类型（例如 APN 网络）会给 iOS 访问企业数据带来麻烦。由于 iOS 本身的限制，iPhone 在切换 APN 网络时显得不太灵活——iPhone 只能通过.mobileconfig 描述文件切换 APN。你可以在 App Store 上找到一堆的应用，专门用于给 iPhone 提供 APN 切换的功能。因此，本书也会介绍如何在自己的项目中实现一个简单 APN 切换器。

另外，与 App Store 中占据主要份额的游戏应用不同，iOS 企业应用有使用数据库技术的迫切需要——作为企业开发人员，习惯于把业务数据保存在关系数据库中的这一顽疾早已根深蒂固——哪怕我们在客户端使用数据库的目的仅仅是出于把服务端数据缓存到本地的需要。

最后，还需要介绍一下文档和报表的显示。企业办公环境中离不开各种文档：文本、图片、视频和声音，尤其 Microsoft 的 Office 文档俨然已成为了企业办公中公文流转的标准格式。如果在 iPhone 手机上竟然无法打开这些最为常见的企业办公文档，这绝对是一场悲剧。而报表和图表，是企业管理中最为常见的数据表现形式和数据分析手段，把企业运营数据以报表图表

的形式进行展示，显然是 iOS 企业应用中应该提供的基本功能。

综上所述，企业开发人员必须充分认识到 iOS 企业应用的特点，结合企业的实际需要，才能开发出一个优秀的 iOS 企业应用。

## 1.3 iOS 企业应用程序的发布

除了上述特点，iOS 企业应用还有一个显著的特点，就是应用程序的发布方式。iOS 企业应用具有两种发布方式：In-House 和 Ad-Hoc，它们并不经过苹果公司的 App Store 进行发布，而只是在企业内部进行发布。换句话说，不经过苹果商店的应用程序审核程序。

### 1.3.1 iOS 应用程序发布与 App Store

2008 年 3 月 6 日苹果公司推出了 iPhone 的应用程序开发包 (iPhone SDK)，吸引了全世界的开发者。2008 年 7 月 11 日，App Store 正式上线，从而开辟了一种前所未有的应用程序销售模式。起初，只有少数的开发者（主要是大型程序开发商）掌握 iPhone 的开发语言，应用程序商店出现的都是一些精品程序。例如，在 2008 年 TIOBE 发布的全球编程语言排名中，iPhone 等使用的 Objective C 一直都在 40 位左右徘徊，使用率为 0.134%；排名第一的为 Java，使用率约为 19%。《连线》杂志评出的 2008 年最佳 iPhone 程序中，排名第一的是来自谷歌的地图程序 Google Earth，Twitter 的 iPhone 手机客户端排名第 7。

然而，随着更多开发者尤其是个人开发者的进入，一些开发者开始从 App Store 中获益，并被媒体大肆报道。例如，Freeverse 公司开发的《Skee-ball》游戏在一个月内就赚得 18.1 万美元，而 Freeverse 开发和部署该游戏仅花费了两个月的时间。国内开发者 139.ME 团队的水族箱程序第一天的下载就实现了 300 美元的收入。

此后，App Store 中应用程序的数量呈现出爆炸性增长。2010 年 8 月，Objective-C 已经进入了 TIOBE 的前十大编程语言排行榜，使用率达到 3.15%，并且是增长率最快的语言。App Store 中应用程序的数量，也从 2008 年底的 1 万个，增加到 2009 年底的 10 万个。截至 2010 年 10 月，App Store 已经增加到 30 万个应用程序，下载量更是突破 50 亿。

这些惊人的数字，只能用奇迹来形容，这也让人不难理解：为什么 iPhone 开发会这样火爆。

然而，如果你现在正准备成为或者已经是一个 iPhone 开发人员，你也许知道，要想下载 iPhone SDK，必须注册成为苹果公司的 iPhone Developer (iPhone 开发人员)——这并不是免费的，你需要购买苹果公司的 iPhone Development Program；否则你只能下载一个功能有限的 iPhone SDK。

除此之外，由于苹果公司一贯坚持的“精品应用”策略，苹果公司没有开放 iPhone 的操作系统。实际上，在 App Store 上线以前，人们甚至无法在 iPhone 上安装程序——苹果公司通过 App Store 控制着 iPhone 应用程序的发布。如果你想让自己开发的程序安装在 iPhone 上，你必须购买 iPhone Developer Program。

iPhone Development Program 有两个版本：标准版和企业版。

标准版程序价格为 99 美元/年，它提供大量开发工具、资源和技术支持，支持可以通过苹果公司 App Store 发布应用程序。同时也支持在 iPhone/iPod Touch/iPad（不仅仅是在模拟器）上调试代码。

企业版程序价格为 299 美元/年，支持开发企业专用的，在内部发布的企业应用程序。它不支持在苹果公司 App Store 销售和发布应用程序，但它支持不经苹果公司审核的应用程序发布方式。

标准版和企业版这两者的区别很像是 IT 界中“做产品”和“做项目”的提法。做产品依靠销售产品拷贝盈利，卖的拷贝数越多则赚的就越多，标准版程序也是如此，依靠下载数量盈利。而做项目并不需要销售产品拷贝，它为企业提供解决方案，做的企业越多则赚的越多，即企业版程序，不管有多少用户在下载（安装）和使用，每做一个项目（企业）收取的开发费用总是相对固定的。

网络资料中，介绍标准版 IDP 比较多，介绍企业版 IDP 相对较少，而 iOS 企业应用是本书的主题，与企业版程序有着直接关系，因此接下来详细介绍企业版程序（企业版 IDP）。

### 1.3.2 Ad-Hoc 与 In-House 发布

企业版 IDP 只支持两种应用程序发布方式，Ad-Hoc 和 In-House 发布。尤其是 In-House 发布，是企业版 IDP 所独有的。它使用一种叫做“In House Distribution Provisioning Profile”的文件进行发布，不能发布到 App 商店进行销售，也不需要 Apple 的评审。你可以把 In-House 应用通过任何方式发布给你的企业员工、用户及你认可的其他任何人，尤其适合于企业应用的开发。

#### 1. 申请企业版 IDP

首先，你需要有一个 Apple ID，如果没有则需要事先申请一个。其次，你的企业需要拥有邓白氏编码。如果没有则需要进行注册。

邓白氏编码是美国联邦政府推荐使用的企业机构编码。可以看成是美国版的“组织机构代码”，只不过已经得到了联合国、澳大利亚政府、欧盟及美国政府的承认，成为了全球企业标准。

申请邓白氏编码在 D&B 公司的网站（英文）：<http://www.dunsregistered.com/>，或者“华夏邓白氏”网站（中文）：<http://dnbregistered.com.cn/>申请，在网站上提交注册申请后，等待 1~2 天，对方人员会跟你联系（Email）。如果英文沟通有问题，你可以在华夏邓白氏进行申请，他们会安排中籍文员跟你联系。

邓白氏注册服务有几个版本，收费情况也不一样。笔者一开始收到的邮件是“实地核实”的版本，报价 15200/2 年。后来经与北京苹果公司联系，只需要购买最基本的“标准版”即可，报价 8600 元/2 年，有网友说 2000~3000 元/年，现在看来是不可能的。联系时一定要强调是购买标准版服务（最便宜），否则你可能会花冤枉钱。

收到邮件后，把申请表、协议打印出来，填好并加盖公章，然后加上企业营业执照副本、扫描为电子的，发给对方邮箱。其实还有一个就是汇款水单（小票），需要发送给对方——这一步其实可以省略，笔者申请时并没有 Email 汇款水单，只要对方确认汇款到账即可。

大约 5~7 天后，对方发来第 2 封邮件，告诉你贵公司的编码。此外还可以在你的公司网站上安装一个邓白氏电子标识——在网页上嵌入指定脚本，则会在页面上显示一个 D&B 图标，点击图标自动链接到 D&B 的网站并呈现你们公司的电子注册信息。

现在，可以申请企业版 IDP 了。登录苹果公司开发者网站 <http://developer.apple.com/iphone/>，申请 Apple Developer Program，注意要选择 iOS Enterprise Program 链接（在页面底部）。

点击 Apply Now 按钮，进入下一页，点击 Continue 按钮，再进入下一页，选择“Use an existing Apple ID”，点击 Continue。进入下一页，输入你的 Apple ID、密码，登录。

后面就是确认注册协议和填写你的公司资料了（英文）。内容最好同邓白氏申请时一样，否则对方会打电话来确认，要你更改。填写完公司资料，还要填写委托人联系资料。注意委托人应该有权代表公司签字（需要贵公司认可，他们会在电话里确认）。

提交资料后，会在注册的联系邮箱里收到苹果公司的邮件，内容大概是感谢你提交了申请，申请的编号是多少，公司名称、邮箱地址等，如果你想看评审流程，可以登录 Member Center。接下来就是等待苹果公司的电话了。这个过程大概要 2~3 天，对方会安排懂中文的人员来电话，如果没什么问题，接下来（电话之后几分钟）会收到苹果公司的第 2 封邮件，大意是要你点击邮件中的链接，查看一个协议。

同意协议后，显示一个页面，大意是你所申请的国家不支持在线购买苹果公司产品（在线支付），需要你下载一个 PDF 格式的 Purchase Form（见图 1-1）。

将它打印出来，根据要求填好，然后传真给苹果公司。

注意，国内信用卡支持美元支付的一般是 Visa 卡（如招行）和 Master 卡（如交行），一定要找那种卡上印有“Visa”或“Master”标志的信用卡。

Cvc2 code 是指信用卡背面的那串数字（7 位）的末 3 位。

信用卡地址写申请信用卡时登记的地址。

如果传真机发送国际传真有麻烦，可将 Purchase Form 扫描后用 Email 发给亚洲苹果公司 [chinadev@asia.apple.com](mailto:chinadev@asia.apple.com)，请其转交给 Billing 团队。亚洲苹果公司几分钟后自动回复了一封邮件，并在信中附了一个业务流水号：Follow-Up: 149653xxx。下次再给亚洲苹果公司联系时，可以附上这个业务流水号。

然后 3~5 个工作日后，如果信用卡办理了账户余额变动短信提醒功能，则会收到扣费成功短信（注意美国和中国有时差，很可能是在半夜发送的）。登录邮箱后，果然收到了苹果公司的 2 封 Email，1 封是发票，上面有你的发票号码，单位报账的时候把这封邮件内容打印出来就可以了。另 1 封是激活邮件，告诉你现在你的 IDP 账号已经生效了，你点击那个 login now 按钮可以登录到 member center，这时可以看到你的 developer program overview 的状态已经改变。同时，Peoples 中会包含一个成员，这个成员就是你注册 IDP 时所绑定的开发者账号（Apple ID），同时也是该 IDP 的 Agent（超级管理员，具有发布权限）。

**Purchase Form**  
Apple Developer Programs

To complete the purchase process, fill in all the sections of this form clearly, sign, and fax to +1(408) 862-7602. You will receive an activation email once your order has been processed.

Fax Number: +1 (408) 862-7602  
Attention: Apple Developer Programs Billing

**1. Select the program you wish to purchase.**

<input type="checkbox"/> iOS Developer Program Standard	USD \$99*
<input type="checkbox"/> iOS Developer Program Enterprise	USD \$299*
<input type="checkbox"/> Mac Developer Program	USD \$99*

\* Your Order will be charged in US dollars

**2. Enter your account information.**

Full Name: \_\_\_\_\_  
 Company Name (if applicable): \_\_\_\_\_  
(\*You must include your Company Name if this information needs to reflect on your purchase invoice)  
 Apple Developer Program Enrollment ID: \_\_\_\_\_ Person ID: \_\_\_\_\_

**3. Enter your billing information.**

Amex     Visa     MasterCard     Discover

Credit card number: \_\_\_\_\_  
 Expiration date (MM/YY): \_\_\_\_\_ CVV/CVC2 Code: \_\_\_\_\_  
 Name on card: \_\_\_\_\_  
(Please ensure you provide your name exactly as it appears on your credit card)

Street/House number: \_\_\_\_\_  
 City: \_\_\_\_\_  
 State/Province: \_\_\_\_\_ Country: \_\_\_\_\_  
 Postal code : \_\_\_\_\_

**4. Cardholder Signature:** \_\_\_\_\_  
(Your signature is required for us to process your purchase)

**5. Email address to send activation code:** \_\_\_\_\_  
Once your order has been processed, your activation code will be sent to the email address provided above.  
 Follow the instructions within the email to activate your Apple Developer Program.

图 1-1 Purchase Form

## 2. 制作开发者证书

### (1) 在本机生成证书请求 CSR

从 Dock 栏的“应用程序”→“实用工具”中，打开“钥匙串”应用程序，修改偏好设置如图 1-2 所示。



图 1-2 修改钥匙串偏好设置

选择菜单“钥匙串访问→证书助理→从证书颁发机构求证书”，如图 1-3 所示。

注意，如果此时密钥中的某个私钥处于选中状态，则菜单会变为“钥匙串访问→证书助理→用<私钥>从证书颁发机构求证书”，这样制作出来的 CSR 是无效的。

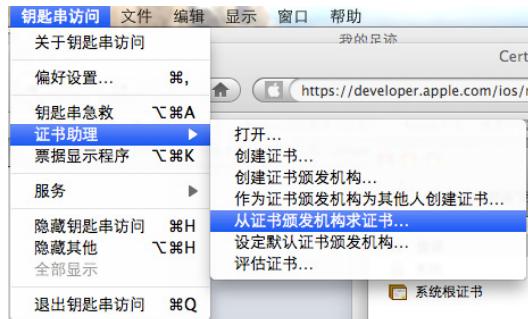


图 1-3 使用证书助理请求证书

输入你的 Email 地址和名字，确保 Email 地址和名字与你注册为 iOS 开发者时登记的一致。

选中 Saved to Disk (保存到磁盘) 单选按钮并勾选 Let me specify key pair information (指定密钥对信息) 复选框，然后点击 Continue 按钮，如图 1-4 所示。



图 1-4 输入 CSR 证书信息

当选择了 Let me specify key pair 复选框之后，会要求你指定文件保存位置。接下来按图 1-5 所示指定密钥对信息。

点击 Continue 按钮，即可生成 CSR 文件。一旦生成 CSR，在“登录”钥匙串中会生成一对密钥对（一个私钥，一个公钥）。你可以在钥匙串的密钥栏中查看。



图 1-5 指定密钥对信息

## (2) 提交 CSR 文件

用企业版 IDP 绑定的 Apple ID ( 跟制作 CSR 时要求输的可能不一致, 这里是注册企业版时绑定的 iOS 开发者账号, 即 Agent ) 登录 iOS Provision Portal 。

在 Provision Portal 页面中, 依次点击 “Certificates→Development 中的 Add Certificate” , 进入图 1-6 所示的页面。

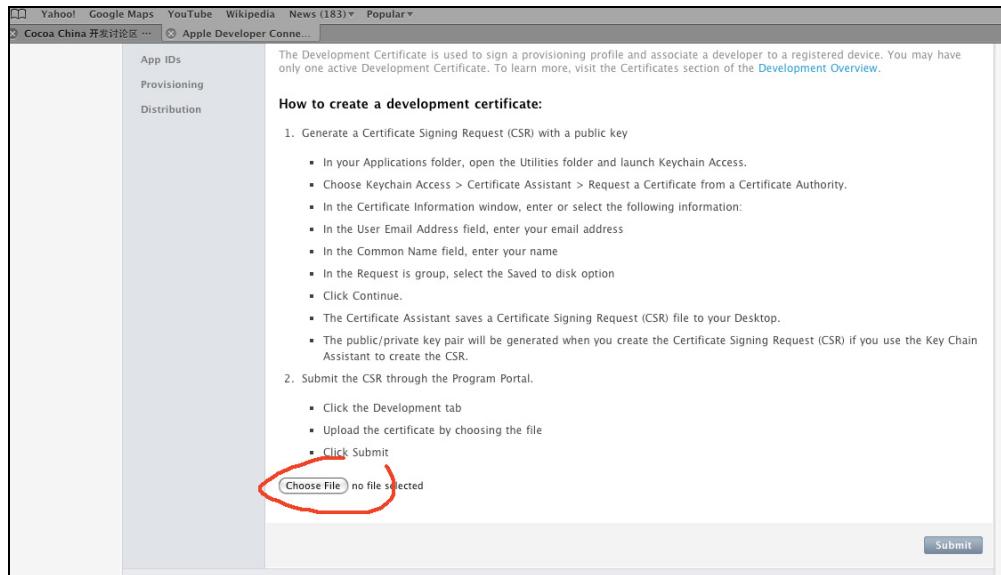


图 1-6 提交 CSR

接下来点击左下角的 Choose File 按钮, 选择所生成的 CSR 文件, 然后点击 Submit 按钮。如果密钥长度未设置为 2048, Portal 会拒绝 CSR。

提交 CSR 后，Team 管理员（Agent）会收到一封提醒邮件，主题为 Certificate Request Requires Your Approval，提示你需要去同意该 CSR。此时 Agent 需要登录 Portal 去同意该 CSR。但实际上，Agent 也可能根本不需要点击“同意”按钮，Portal 几秒钟后就自动同意了——笔者遇到的情况就是这样的（可能笔者是用 Agent 提交 CSR 的）。

### （3）下载并安装开发者证书

用提交 CSR 的 Apple 账号登录 Portal。如果机器上未安装 WWDR 证书，请点击“Certificate → Distribution”中的链接“Saved Linked File to Downloads”，以下载 WWDR 证书，并通过双击 WWDR 证书文件进行安装。

在“Certificate→Development”中，在 Your Certificate 下会列出当前有效的开发者证书。点击 Download 按钮，即可下载到本机。下载后双击，即可安装到本机。可以在钥匙串“证书”一栏中查看到导入的开发证书。

Team 成员只能下载自己的 iOS 开发证书。Team 管理员有权下载所有成员的公有证书。苹果公司不接受 CSR 中的私钥。私钥仅对创建者有效，并且必须存储在系统钥匙串里。

### （4）保存私钥并迁移到其他系统

如果你在多台电脑上进行开发或者重装系统，那么把私钥存储在安全的地方是一件很重要的事情。如果没有私钥，你无法在 Xcode 中签名代码并进行真机调试。

钥匙串在生成操作系统 CSR 时，就会在“登录”钥匙串中创建一个私钥。该私钥和你的用户账号绑定，如果重装操作系统导致该私钥遗失，则该私钥无法再次生成。如果你想在多台电脑上开发和调试，你必须将私钥导入到每一台机器上：

在钥匙串访问程序中，选择登录钥匙串的“密钥”。可以看到有许多密钥对，选择与你的开发者证书相对应的私钥（还记得创建 CSR 时要你输入的邮箱地址和名字吗？那个名字会显示在私钥的名字上）。然后选择菜单“文件→导出项目...”，将私钥保存为.p12 格式（Personal Information Exchange）。当提示输入密码时，设置一个密码并记住它，它会在导入.p12 文件时使用。现在，你可以把.p12 文件拷贝到其他机器上并双击它进行安装，这时会提示你输入导出私钥时设置的密码。

这个私钥是重要的。如果你机器重装系统了（或者你想把开发环境迁移到另一台机器上），那么很可能需要重新安装开发环境，包括导入开发者证书。每个开发者证书都是和申请证书（提交 CSR）时的私钥（私钥可以保存在.p12 文件中）是绑定的。如果仅仅是导入了开发者证书文件而机器上没有对应的私钥，则这个开发者证书对这台机器是无效的。

## 3. 设备 ID

所谓设备 ID（device ID 又称 UDID）是 Apple 设备上的 40 位十六进制码，每台 Apple 设备的设备 ID 都是唯一的，Apple 以此来识别每一台 iOS 设备。

我们通过在 Provision Portal 中录入设备的设备 ID，可以允许开发者在指定真实设备上进行调试。在 Provision Portal 中最多允许输入 100 个设备 ID。

因此，录入设备 ID 是后续制作 Provision Profile 的必需步骤（而 Provision Profile 又是真机调试的必需步骤）。

### (1) 获取设备 ID

有两种获取设备 ID 的方式：

把 Apple 设备 (iPhone 或 iPod) 连接电脑，打开 Xcode (以 4.2 版本为例) 的 Organizer，如图 1-7 所示。

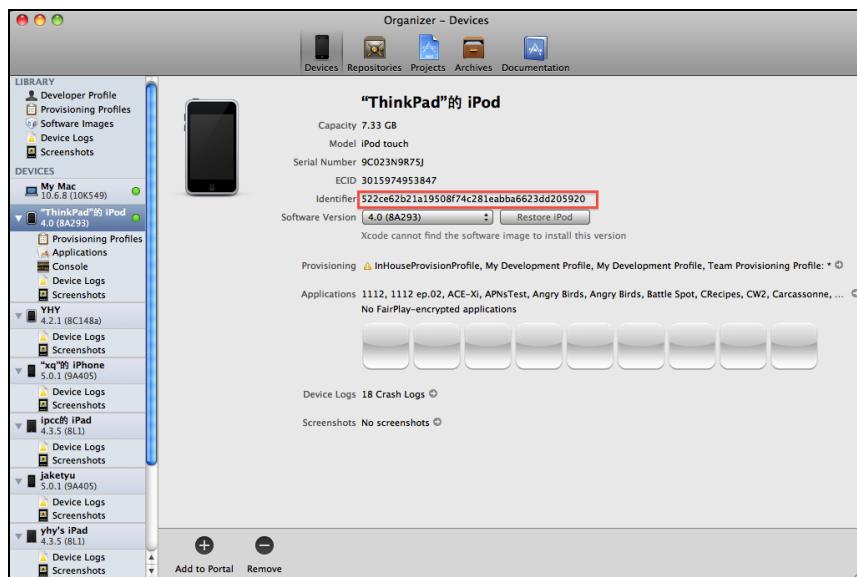


图 1-7 在 Organizer 中查看设备 ID

或者，把 Apple 设备 (iPhone 或 iPod) 连接电脑，打开 iTunes，如图 1-8 所示。



图 1-8 在 iTunes 中查看设备 ID

那个40位16进制的数字就是设备ID。

### (2) 添加设备ID

以Team管理员登录Provision Portal，点击Devices页面中的“Add Device”按钮，如图1-9所示，在其中进行以下设置：

Device Name：设备名称，输入一个描述该设备的名字。

UD ID：即设备ID。

点击Submit按钮即可。

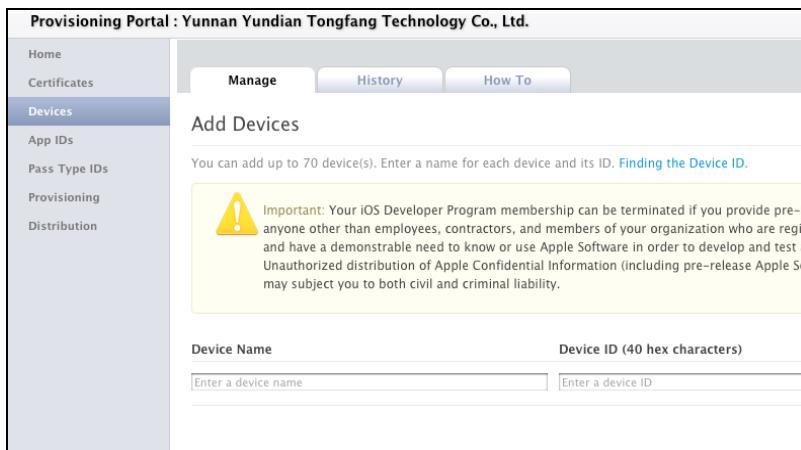


图1-9 在Provision Portal中添加设备ID

## 4. 创建App ID

App ID是识别不同应用程序的唯一编码。如果你的程序要连接Apple Push Notification服务（一种push通知），需要用到App ID。如果应用程序之间要共享钥匙串数据，也会用到App ID。总之，App ID在iOS开发中很重要。在这里App ID的最大用处是制作真机调试用的Provision Profile（对代码进行签名，它需要提供一个App ID）。

一个App ID由两部分构成：一个10位字符的Bundle Seed ID前缀，这个Bundle Seed ID由Apple分配，全球唯一，保证不会重复；一个Bundle Identifier后缀，这个Bundle Identifier由Team管理员指派，Apple建议用反域名规则命名这个Bundle Identifier。例如：8E549T7128.com.apple.AddressBook。其中，8E549T7128是Bundle Seed ID，com.apple.AddressBook是Bundle Identifier。

如果你写了一系列应用程序，它们共用相同的钥匙串（如共用密码），或者根本就不使用钥匙串访问，你可以只创建一个App ID，所有的应用程序都使用以星号结尾的App ID。这个星号就是通配符，只能用于App ID最后一个字符。例如，这个App ID可以是：R2T24EVAEE.com.domainname.\*或者R2T24EVAEE.\*。

以Agent或Team管理员登录Provision Portal，点击“App ID”页面中的“New App ID”按钮。如图1-10所示，在其中修改如下信息：

- Description: 给这个 App ID 一个名字。如果存在多个 App ID，每个 App ID 需要一个易于识别的名称。
- Bundle Identifier: 如前面所述，Bundle Seed ID 是 Apple 分配的，其实这里只需要你输入 Bundle Identifier。可以使用统配符\*。

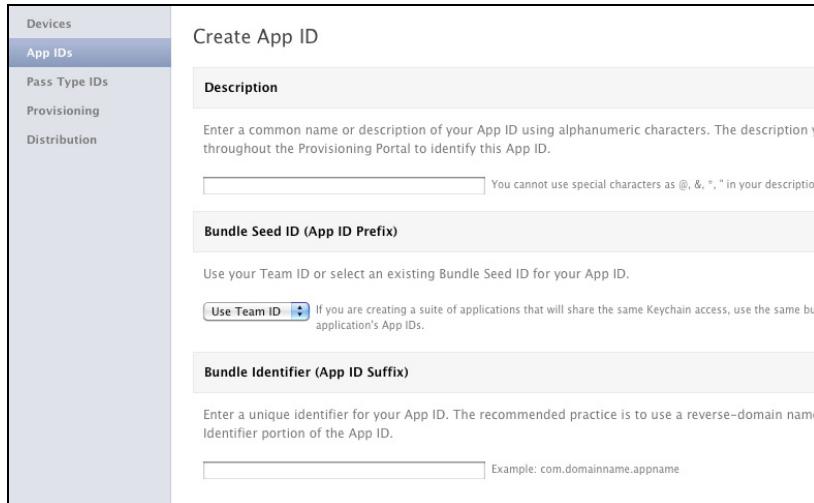


图 1-10 创建 App ID

## 5. 制作开发者 Provisioning Profile

拥有了开发者证书（Development Certificate），只是表明你有权利在电脑上进行开发，在模拟器上运行程序，但你还不能在 iPhone 上运行你开发的程序。其实如果你只是在模拟器上调试程序的话，要不要开发者证书都无所谓，因为证书只是用来代码签名（Code Sign）的，如果在模拟器上运行的话，你可以选择不签名（don't code sign）。

如果要在真机上调试就不一样了，没有这个 Provision Profile，苹果设备无法安装、运行你开发的程序（这个 Provision Profile 也将随程序一同安装到 iPhone 上）。这个 Provision Profile 中记录了一些信息：开发者证书、开发者 Apple ID、一系列设备 ID（开发者可以使用哪几部设备进行调试——这些设备的设备 ID 要登录到 Portal 上）。

### （1）创建开发者 Provision Profile

登录 Provision Portal，点击“Provisioning→Development”，点击 New Profile 按钮，修改以下信息：

- Profile Name: 输入 Profile 的名字，随意。
- Certificate: 选择开发者证书。
- App ID: 选择一个 App ID。
- Devices: 设备 ID 列表。

点击 Submit 按钮，即会生成 Development Provisioning Profile，如图 1-11 所示。

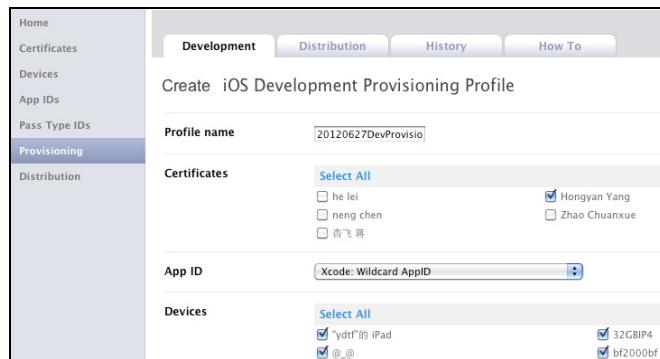


图 1-11 生成开发者 Provision Profile

### (2) 安装 Development Provision Profile

所有 Team 成员都可以下载 Development Provision Profile。但只有 Profile 中记录了设备 ID 的设备以及 iOS 开发者证书所指定的开发者能够使用这个 Profile。

在 Portal 的“Provisioning→Development”中，点击某个 profile 右边的“download”按钮。下载 profile 后，将下载到的文件拖曳到桌面 Dock 面板的 Xcode 图标上（或者直接拖到 Xcode 的 Organizer 中）。这会将 profile 文件拷贝到~/Library/MobileDevice/Provisioning Profiles 目录。

### (3) 签名并调试

这需要用到两个文件：证书用于给代码签名，Provisioning Profile 用于真机调试。

在 Xcode（以 4.2 版本为例）中打开项目，选中 Target，打开 info 窗口，在 Build Settings 面板中找到“Code Signing Identity”，打开并点击“Debug”下面的“Any iOS Device”将弹出一个签名文档（即 Provisioning Profile）选择列表，如图 1-12 所示。

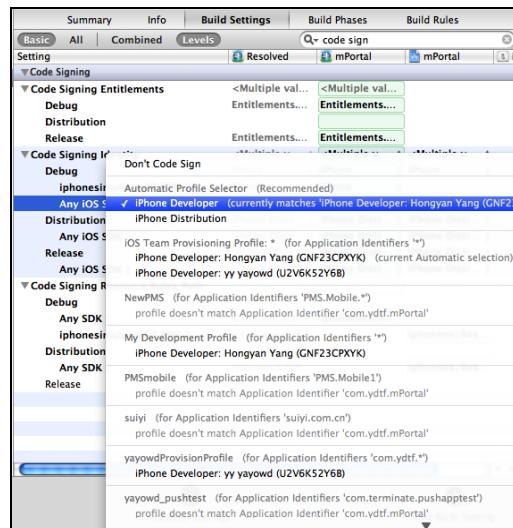


图 1-12 使用 Provision Profile 进行签名

---

**提示：**除了可以对“Debug”进行签名，我们还可以对“Release”、“Distribution”等进行签名。这里“Debug”、“Release”、“Distribution”指的是不同的编译版本，在Xcode里也叫做Schema。一个Schema是一种编译方案，代表了一个项目在编译时所采用的编译选项，包括编译器选项参数、环境变量、签名文档和编译脚本等。默认情况下，一个Xcode项目只有“Debug”和“Release”两种Schema，分别代表了调试时和发布时的不同编译选项。在Xcode里，这两种Schema是不一样的。因为程序员在开发调试过程中，对代码进行编译是非常频繁的，这种情况下，应该对编译速度进行一些优化，以节省编译时间，但同时运行效率就会较差一些。但对于发布版本就相反了，这时的编译器应当优先考虑在运行速度上进行优化，而编译速度就会有所下降。此外，你也可以自己加入一些定制的Schema，比如“Distribution”，从而对编译选项进行一些调优。

---

在弹出菜单中选择你要用于签名的 Provision Profile（即先前在Portal中制作的 Provision Profile），该签名应当和一个开发者证书对应。这个Profile就是前面安装的 Development Provision Profile。

注意，有时候Xcode会自动根据当前连接的设备的设备ID选择一个有该设备调试权限的签名，比如一般是位于Automatic Profile Selector（灰色）条目下面的iPhone Developer项。这在大部分时候是适用的，但有时候Xcode的选择并不符合你的意愿。此时就需要手动修改签名。

例如，在图1-12中，Xcode自动选择了“Automatic Profile Selector”下面的“iPhone Developer (current matches 'iPhone Developer:Hongyan Yang...')”进行签名。这表明将用iPhone Developer“Hongyan Yang”的数字证书进行签名。一个证书可以绑定多个Provision Profile。在图1-12的例子中，“Hongyan Yang”的证书就会存在于许多Provision Profile中：My Development Profile、iOS Team Provisioning Profile。在图1-12中，有的证书是灰色的，表明不能用于当前签名（可能是Provision Profile的IDs列表中没有包含调试设备，或者App ID不匹配，或者证书和私钥不匹配等原因）。

在Target的info面板（其实就是info.plist中的内容）中，还需要设置的Bundle Identifier。如果你的App ID是A1B2C3D4E5.com.domainname.applicationname（我们在前面创建的App ID），那么Bundle Identifier可以是com.domainname.applicationname（不需要填写Bundle Seed ID）。如果App ID使用了通配符，比如A1B2C3D4E5.com.domainname.\*，则Bundle Identifier可以是com.domainname.<任意字符>。如图1-13所示（以Xcode 4.2为例）。

签名完成，你就可以在真机上运行程序了。点击Xcode工具栏左上角Scheme下拉按钮，从中选择Device→Debug，然后点击Build and Debug按钮，编译并在真机上运行程序。

#### （4）发布应用程序

发布应用程序需要使用发布证书（Distribution Certificate）。发布证书的制作，跟制作开发者证书的步骤是一样的，只不过使用的是Provision Portal的“Certificates→Distribution”功能。

把制作好的发布证书下载、安装到本机。

Custom iOS Target Properties		
Key	Type	Value
Bundle identifier	String	com.ydtf.mPortal
InfoDictionary version	String	6.0
Application supports iTunes file sharing	Boolean	YES
Bundle version	String	0.2
Executable file	String	\$(EXECUTABLE_NAME)
Application requires iPhone environment	Boolean	YES
Java classpaths	Array	(1 item)
Bundle display name	String	南网电力通
Cocoa Java application	String	YES
Bundle OS Type code	String	APPL
Icon file	String	app.png
Bundle creator OS Type code	String	????
Java root directory	String	Contents/Resources
Localization native development region	String	China
Bundle name	String	\$(PRODUCT_NAME)
▶ Document Types (0)		
▶ Exported UTIs (0)		
▶ Imported UTIs (0)		
▶ URL Types (0)		

图 1-13 为程序分配 App ID

发布应用程序时使用的是“发布证书”，就如同开发时要使用“开发证书”一样。同理，发布时用的签名文档（即 Provision Profile）也与开发时使用的不太一样。

企业版 IDP 有两种发布方式：In-House 和 Ad-Hoc。两种 Profile 制作步骤稍有区别。而前者（In-House 方式发布）正是企业版 IDP 真正区别于其他版本的 IDP 所在。我们重点介绍 In-House 方式的发布。

## 6. 制作 In-House 发布的签名文档

以 Team Admin (Agent) 登录 Provision Portal，打开“Provisioning Distribution”页面，如图 1-14 所示。

Provisioning Portal : Yunnan Yundian Tongfang Technology Co., Ltd.

Home Certificates Devices App IDs **Provisioning** Distribution History How To

Create iOS Distribution Provisioning Profile

Generate provisioning profiles here. All fields are required unless otherwise noted. To learn more, visit the [How To](#) section.

Distribution Method  In House  Ad Hoc

Profile Name

Distribution Certificate  Yunnan Yundian Tongfang Technology Co., Ltd. (expiring on May 3, 2012)

App ID

Devices (optional) Select up to 100 devices for distributing the final application; the final application will run only on these selected devices.  
 yhy's iPhone3GS

图 1-14 创建 In-House 发布证书

进行如下配置：

- Distribution Method：发布方式，选择 In House。
- Profile Name：Profile 名称，用于区别多个 Profile。
- Distribution Certificate：选择要在 Profile 中绑定的发布证书。
- App ID：指定一个已有的 AppID。
- Devices (optional)：要绑定的 device ID。因为 In-House 方式可以在任何 Apple 设备上发布，所以不需要设定 Devices，这一项不可用。

点击 Submit 按钮，生成 Profile。将 Profile 下载到本地进行安装。方法：把 Profile 文件拖曳到 Dock 上的 Xcode 图标。

## 7. 制作 Ad-Hoc 发布的签名文档

以 Admin 或 Agent 登录 Provision Portal。打开“Provisioning Distribution”页面，如图 1-15 所示。

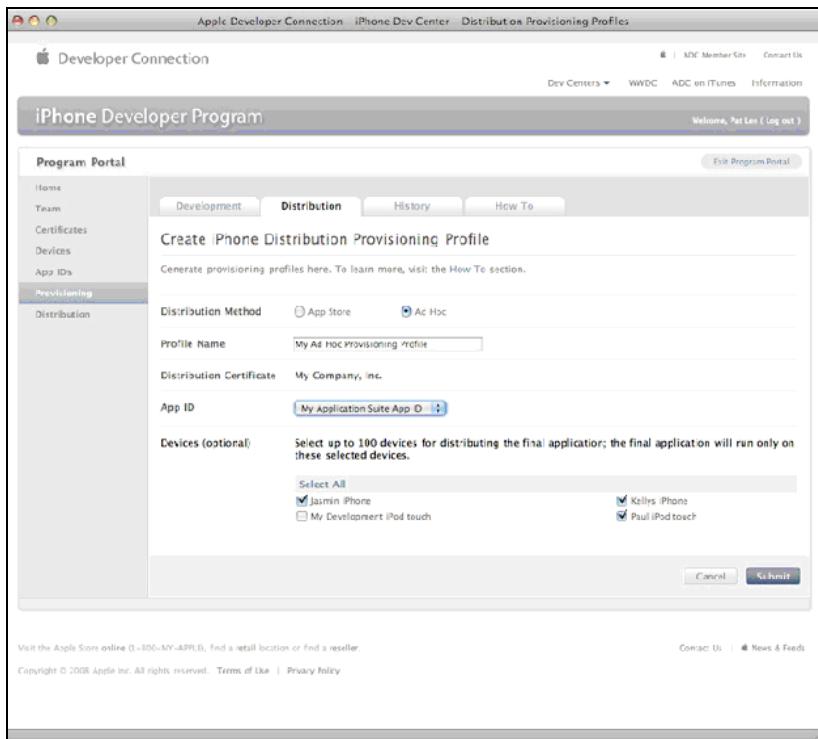


图 1-15 创建 Ad-Hoc 发布证书

与 In-House 方式大同小异，只不过发布方式选择 Ad-Hoc，同时在 Devices ( optional ) 栏勾选要绑定的 device ID，最多可选择 100 个。

点击 Submit 按钮，生成 Profile，将 Profile 下载到本地进行安装。

## 8. 编译发布版本

打开你的项目。在 Target 的 Builder Settings 面板中，找到 Code Signing Identity 下面的 Release 项。将 Any iOS SDK 指定为你的发布证书（Distribution Certificate），如图 1-16 所示。



图 1-16 在 Build Settings 中为发布版本进行代码签名

切换到 Info 面板，在 Identifier 栏输入 Bundle Identifier。该 Bundle Identifier 应根据 App ID 填写。

选择菜单“Project→Edit Scheme”，在 Profile 的 Info 窗口中，将 Build Configuration 选择为 Release，如图 1-17 所示。

在菜单栏选择“Product→Archive”。

如果 Archive 是灰色的，请连接设备，在 Scheme 按钮中选择所连接的 iPhone，如图 1-18 所示。

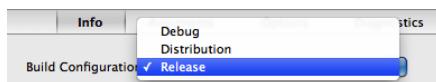


图 1-17 编辑 Scheme

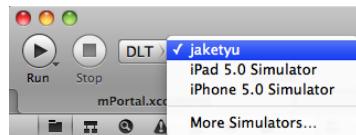


图 1-18 Scheme 请选择真机

这时再次选择“Product→Archive”，Archive 就变成可用的了。

点击“Product→Archive”，编译成功后，会弹出 Organizer 窗口，如图 1-19 所示。

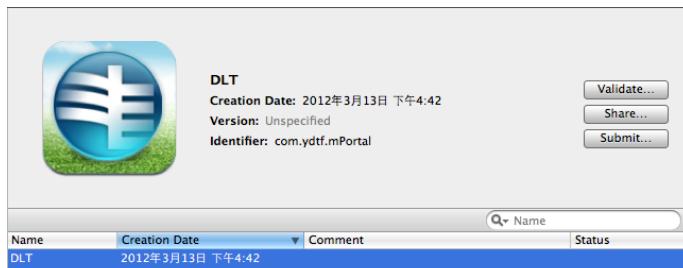


图 1-19 Archive 成功后的 Organizer 窗口

点击 Share 按钮，又会弹出另一个窗口，如图 1-20 所示。

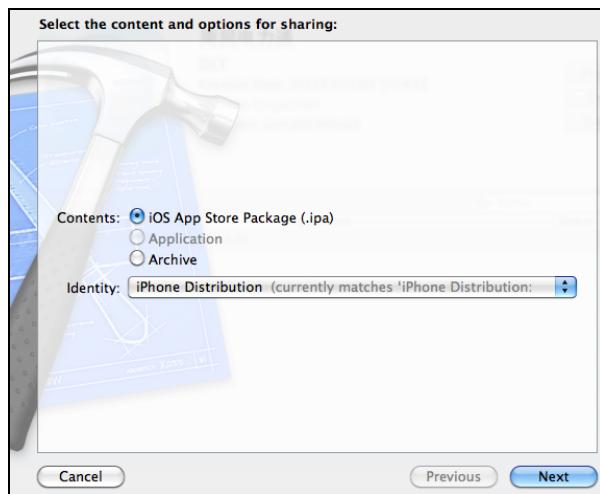


图 1-20 选择 Content 类型

然后，点击 Next 按钮，会弹出新窗口选择 Archive 保存路径，如图 1-21 所示。注意，不要勾选“Save for Enterprise Distribution”选项。

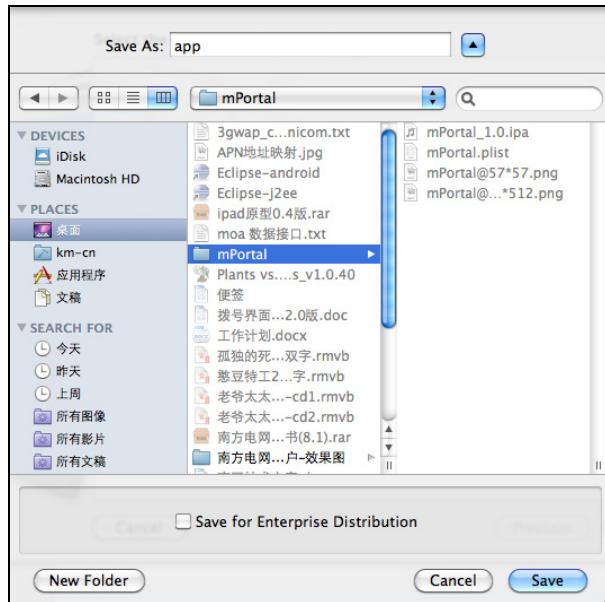


图 1-21 选择保存路径

填入文件名，选择保存路径，点击 Save 按钮，文件会保存在指定地方。打开 Finder，你可以在指定目录下看到生成的.ipa 文件。

## 9. 安装应用程序

以 Ad-Hoc 或 In-House 方式发布的应用程序，可以将.ipa 文件直接发送给用户。用户可以用两种方式安装：使用 iTunes，或者使用 iPhone 配置实用工具。

### (1) 使用 iTunes

用户将压缩包中的.ipa 文件拖到 iTunes 的“资料库→应用程序”下，然后和 iPhone/iPod 进行同步。

### (2) 使用 iPhone 配置实用工具

iPhone 配置工具是完全免费的，你可以从这里下载：

[http://support.apple.com/kb/DL926?viewlocale=zh\\_CN](http://support.apple.com/kb/DL926?viewlocale=zh_CN)

安装后会在“应用程序/实用工具”中生成一个快捷方式“iPhone 配置实用工具”。

同样，将 iPhone/iPod 连上电脑，打开“iPhone 配置实用工具”，将.ipa 文件拖放到“iPhone 配置实用工具”的“资料库→应用程序”下，然后选中你的 iPhone/iPod，在右边“安装或删除应用程序列表”中，点击某个应用程序右边的“安装”按钮进行安装。

## 10. 问题及错误

如果 Xcode 出现 Code sign 错误：

Code Sign Errors: profile doesn't match any valid certificate/private key pair in the default keychain

同时在 Organizer 中出现下列提示：

A valid signing identity matching this profile could not be found in your keychain

则需要把钥匙串中的所有证书和密钥删除，然后重新请求证书、修复 provision profile、下载并安装，一般可以得到解决。

### 1.3.3 OTA 无线部署

所谓 OTA( Over The Air )，是苹果公司在 iOS 4.0 中加入的一种新的企业部署方式，即 iOS 4 的无线部署。无线部署是完全脱离 iTunes 的发布程序的一种方式。苹果公司扩展了 iOS 的 Safari 的功能，使得 iOS 企业应用可以通过 Safari 浏览器进行部署。Safari 对 URL 地址中的特殊协议 itms-services 进行识别并自动下载 ipa 安装包，并在下载完成后调用 iOS 操作系统的应用程序安装界面（iTunes）进行安装。

“无线部署”专用于企业部署，包括 Ad-Hoc 和 In-House 部署，所以这里你必须使用这两种 provision profile 文件。下面介绍如何在 Xcode（以 4.2 版本为例）中进行 OTA 部署。

OTA 部署仍然使用“Product→Archive”菜单的功能（正如 1.3.2 节下面的“8. 编译发布版本”节所描述）。但在选择保存路径这一步（如图 1-21 所示步骤），注意勾选“Save for Enterprise Distribution”选项。这样将出现 OTA 部署选项，如图 1-22 所示。

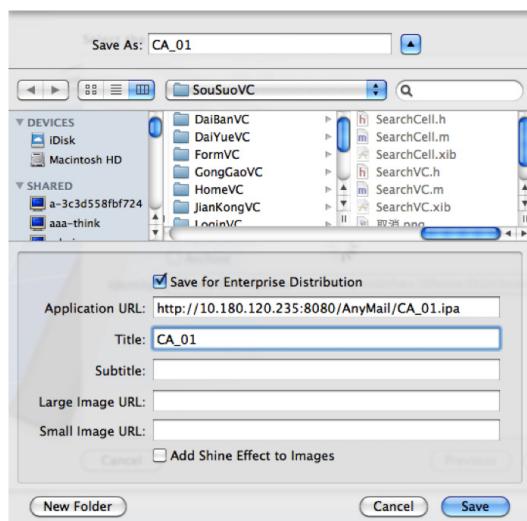


图 1-22 勾选 Save for Enterprise Distribution 将出现 OTA 部署选项

需要填写的各项设置意义如下：

- Application URL：你的.ipa 文件将部署到服务器的哪个位置。当用户通过 Safari 安装你的企业应用时，这个 URL 应指向.ipa 文件所在的地址，必填。
- Title：企业应用名称，必填。
- Subtitle：子标题，可选。
- Large Image URL：大图标（512×512）所在的 URL。大图标用于在 iPad 上安装时显示应用程序图标，可选。
- Small Image URL：小图标（57×57）所在的 URL。大图标用于在 iPhone 上安装时显示应用程序图标，可选。
- Add Shine Effect to Image：会在大/小图标上加上一个光照效果，可选。

填写完后，点击 Save 按钮。Xcode 会在硬盘上产生一个.ipa 文件和一个.plist 文件。这两个文件都 OTA 部署所必需的。其中.plist 文件是一个 XML 文件，保存了你刚才填入的那些配置选项。

将上述两个文件和图标文件（如果有的话），放到 Web 服务器上。注意，.ipa 文件和图标文件所放的位置应该和你在图 1-22 界面中配置的内容一致。至于.plist 文件，你可以让它和.ipa 文件放在一起，也可以单独放在另外一个地方，比如：<http://10.180.120.235:8080/yourappname.plist>。

另外制作一个 html 文件，如 install.html，内容如下：

```
<html>
<head><title>TextGlowDemo</title></head>
<body>
```

```

<ul>
  <li>
    <a href="http://10.180.120.235:8080/AnyMail/InHouseProvisionProfile.
      mobileprovision"> Provisioning File</a>
  </li>
  <li>
    <a href="itms-services://?action=download-manifest&url=http://10.180.120.235:8080/
      AnyMail/GlowDemo.plist">
      install GlowDemo</a>
    </li>
</ul>
</body>
</html>

```

主要是两个[标签](#)。第一个[标签](#)是 Provision Profile 文件的超级链接，因为在 iPhone 上安装应用程序之前，必须先安装 Provision Profile 文件，所以这个链接是有必要的。第二个[是.plist 的超级链接。](#)

注意：在第 2 个[标签](#)中，[属性的“itms-services://?action=download-manifest&url=”](#)是固定的（后面的跟着.plist 文件的真实 URL 地址）。“itms-services://?action=download-manifest&url=” 不能改成其他内容，否则 Safari 不会调用安装程序。

把这个 HTML 文件放在 Web 服务器上，然后在 iPhone 上用 Safari 访问这个 HTML 文件地址（URL，比如：<http://10.180.120.235:8080/install.html>），你将看到如图 1-23 所示的页面。



图 1-23 在 iPhone 上访问 install.html

首先点击 Provisioning File 链接，iOS 提示你要安装该预置描述文件，根据提示安装就可。

然后点击 install GlowDemo 链接，Safari 会开始下载.ipa 包，当下载完成会立即启动系统安装程序进行安装。

---

**注意：**如果 iPhone 上已经安装有该应用程序，则会进行覆盖安装。

---

这样，通过 OTA 部署，不再需要用户将 iPhone 和 PC 同步的繁琐步骤（利用 iTunes）。OTA 部署直接利用 iPhone 的无线通信功能（WiFi 网络或移动网络）进行应用程序部署（或者在线更新），显得更加灵活和方便。

## 第 2 章 iOS 开发框架简介

本章首先介绍苹果操作系统 iOS 的起源、发展及构成，然后对 iOS 开发框架 Cocoa Touch 进行介绍。Cocoa Touch（或 Cocoa）是多个开发框架的集合，由多个层级的子框架构成。最后介绍苹果开发工具包 iOS SDK 及开发环境的搭建。

### 2.1 苹果 iOS 简介

苹果 iPhone 手机自发布之日以来就给人们带来了全新的感觉和操作体验。一是因为 iPhone 更为优秀的硬件性能，二是因为苹果手机跨时代的操作系统——苹果 iOS。

iOS 即 iPhone OS，是苹果公司针对其 iPhone、iPod Touch 和 iPad 产品开发的基于 UNIX 架构的苹果专属操作系统。原本这个系统名为 iPhone OS，直到 2010 年 6 月 7 日 WWDC 大会上宣布改名为 iOS。iOS 分为 iPhone、iPod Touch 和 iPad 三个版本，但三个版本往往同步更新。

iOS 的发展历史：

- 2007 年 6 月 29 日，苹果发布了 iPhone OS 1.0 固件。
- 2008 年 7 月 11 日苹果发布了 iOS 2.0。
- iOS 3.0 固件在 2009 年 6 月 17 日发布。
- 2010 年 6 月 21 日，苹果发布第四代 iPhone 操作系统——iOS 4.0，在这个版本中，苹果加入了人们期待已久的多任务处理功能。
- 目前最新的 iOS 版本为 2012 年 7 月发布的 iOS 6 版本，这只是 Beta 3 版，正式版预计稍后推出。

虽然 iOS 是一个相对封闭、苹果专属的操作系统，其他品牌厂商的产品无法使用，但其优秀的运行性能和杰出的操作体验，仍然是许多操作系统无法比拟的。作者在同时使用了一段时间的 iPhone 和 Android 手机后发现，无论是系统性能还是稳定性上而言，iOS 4.0 的表现都要远远优于 Android 2.x 系统。

其次，从应用程序的丰富程度上看，iOS 应用也远远超过了 Android 应用。很显然，App Store 能让应用开发商盈利这一点，是 iOS 开发平台比 Android 更胜一筹的主要原因。

可以预料到的是，在未来相当长一段时间内，三大智能手机操作系统中以 iOS 处于领先地位的事实不会改变。iOS 在 iPad 上获得的成功也从侧面印证了这一点。

搭载 iOS 系统的苹果 iPad 获得了巨大的成功，其良好的用户体验和完善丰富的应用软件是成功的重要因素，这一成功与 iOS 系统在 iPhone 智能手机上的积累和完善密不可分。在应用到 iPad 产品之前，iOS 经过三年多的改进，在 iPhone 上已经相当成熟，同时其积累的应用程序和开发经验也可以顺利转移到平板电脑上。另一方面，iPhone 形成的良好口碑和用户经验

也可以顺利转移到 iPad 平板电脑系统上。

RichRelevance 于 2011 年 12 月 25 日的调查数据显示，iOS 设备在移动商务市场上的份额由 4 月份的 88% 上升到 12 月份的逾 92%，Chitika Insights 发布的数据显示，至 2012 年 2 月份，iOS 设备的网络流量市场份额甚至超过了苹果自己的 Mac OS。

## 2.2 iOS 框架介绍

iOS 衍生自 Mac OS X 的成熟内核，但 iOS 操作系统更紧凑和高效，支持 iPhone 和 iPod Touch 的硬件。iOS 继承了 Mac OS X 的风格，包括：统一的 OS X 内核，针对网络的 BSD 套接字，以及 Objective-C 和 C/C++ 编译器。

iOS 框架分为 Cocoa Touch、Media、Core Service、Core OS 四个层次，如图 2-1 所示。



图 2-1 iOS 框架

这 4 个层次从上到下排列，位置越高说明层次越抽象，距离硬件底层越远，其特点如下：

- 层次最高的是 Cocoa Touch 框架，是我们使用得最多的框架，每个 iOS 应用都要使用，其中包括：UIKit 和 Foundation ( NS )，下一节会更详细地介绍这一层。
- Media 框架是对 iPhone 音频和视频协议的封装，例如，OpenGL ES、EAGL、Quartz、Core Animation、Core Audio、Open Audio Library 和 Media Player。
- Core Services 框架提供了一些核心框架，诸如 Address Book 和 Core Foundation，后者包含了基本的数据类型定义，如数组和集合。
- Core OS 框架包含系统内核级服务，如线程、文件、I/O、内存和网络。

## 2.3 Cocoa Touch 框架简介

Cocoa Touch 框架是进行 iPhone 应用程序开发工作的主要框架，主要包括 UIKit 和 Foundation( NS )框架，这些库统称为 Cocoa Touch 框架。该框架完全是面向对象的，它是 Cocoa 框架的子集。

注意：Cocoa 框架早先是用于 Mac OS X 上的一个面向对象的应用程序快速开发（Rapid Application Development, RAD）框架，包含了 Foundation 和 App Kit 框架，可用于开发 Mac OS X 系统的应用程序。而随后苹果又在 Cocoa 中加入了对 iOS 的支持，即 UIKit 框架。习惯上，把 UIKit 框架、Foundation 框架及一些附属框架合称为 Cocoa Touch 框架，如图 2-2 所示。



图 2-2 Cocoa 框架和 Cocoa Touch 框架

注意，App Kit 用于 Mac OS X。而 UIKit 用于 iOS（它参考了 App Kit 的实现）。Foundation 框架和附属框架则是二者所共有。

Cocoa Touch 是 iOS 上关于用户交互的可编程框架。采用源自 Cocoa 和强大的 Mac 桌面的技术，Cocoa Touch 和 iOS 针对多点触控进行了重新设计。由于其小巧的外形，iPhone 上的按钮、表格表单、页面过渡以及触摸手势都是独特的，而这些界面功能，都可以通过 Cocoa Touch 框架实现。

Cocoa 框架采用“模型-视图-控制器”（MVC）设计模式。“模型”封装应用程序的数据，“视图”显示和编辑数据，“控制器”处理前两者之间的逻辑关系。这种分工负责的方式使得程序易于设计，实现和维护，如图 2-3 所示。

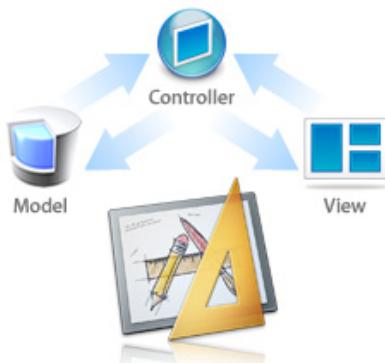


图 2-3 Cocoa 框架中的 MVC 设计模式

## 2.4 搭建 iOS 开发环境

迄今为止，iOS 只支持在苹果的 Mac OSX 操作系统下进行开发。因此，对于大部分开发者而言，一台基于 Intel 的苹果电脑仍然是必需的——无论是 Mac Book 还是 Mac Mini 都能满足开发的需要。当然，也可以在非苹果的电脑上安装 Mac OS X，正如下面介绍的，借助于硬件虚拟化技术的支持，可以在虚拟机中安装 Mac OS X。

此外，需要下载并安装苹果的 iOS 开发工具包（Software Development Kit, SDK）。这是一个应用程序集合，包括了用于创建 iOS 应用程序所必需的 IDE、API 库及实用工具。

最后，你可能需要在苹果官方网站进行注册。虽然这不是必需的，但如果这样做的话，你可能无法将你的程序安装到设备上运行。

### 2.4.1 安装 Mac OS X 操作系统

自从 2007 年年底苹果公司正式发布代号为 Leopard 的 Mac OS X 10.5 开始，一种叫做“Hacked Apple”——把 Mac OS 安装到 PC 上的技术就成为了现实。仅仅在 Leopard 正式上市后的第二天就有高手将其成功破解，使用几个补丁文件便能让 Leopard 安装到普通的电脑上。

由于 Mac OS X 本身对 PC 硬件的支持十分有限，在普通 PC 和笔记本电脑上安装 Hacked Apple 极其不易。尽管网络上存在有各种破解补丁、硬件驱动，甚至破解好的镜像文件，要想在一台非苹果电脑上“啃”一嘴苹果仍然是被戏称为“拼人品”，网上有着无数失败的先例。

有鉴于此，笔者并不建议初学者在非苹果 PC 上安装 Mac OS X 操作系统，与浪费了的无数精力和时间相比，所获得的好处实在不足以称道。如果实在是无法接受苹果电脑的高端价格，那么你可以尝试另一种在 PC 上安装 Mac 系统的方式——在虚拟机中安装——幸好我们还有虚拟机，无论是 VMWare，还是 VirtualBox。

在虚拟机中安装 Mac 拥有以下好处：在 Windows 系统和 Mac 系统间切换不需要重启；在虚拟机中安装避免了硬件驱动不支持的问题，因为不需要安装硬件驱动程序；使用虚拟机安装有更高的成功率。

以下以笔者的华硕 X42J 笔记本为例，演示如何在 VirtualBox 中安装 Mac Snow Leopard OS X 10.6.5（支持 i3/i5/i7）。

#### 1. 推荐硬件配置

原则上，CPU 必须支持 SSE2/SSE3 和硬件虚拟技术。如果不能确定 CPU 是否支持硬件虚拟，可以运行 SecurAble 进行测试，出现如图 2-4 所示的对话框即为支持。

以下列出笔者的笔记本硬件配置，以供参考：

- 电脑型号——华硕 K42JE 笔记本电脑
- 处理器——英特尔 Core i3 M350 @ 2.27GHz 笔记本处理器
- 主板——华硕 K42JE（英特尔 HM55 芯片组）
- 内存——2GB（海力士 DDR3 1333MHz）



图 2-4 用 SecurAble 进行测试

- 主硬盘——希捷 ST9320423AS ( 320 GB / 7200 转/分 )
- 显卡——ATI Mobility Radeon HD 5470 ( 512 MB )
- 光驱——日立-LG DVDRAM GT32N DVD 刻录机
- 声卡——瑞昱 ALC269 @ 英特尔 5 Series/3400 Series Chipset
- 网卡——智微 JMC25X PCI Express Gigabit Ethernet Adapter

## 2. 准备使用的工具

虚拟机 Virtual Box 的下载地址：<http://u.115.com/file/t54cd05734>。

破解版的 Mac OS X, iAntares OSx86 10.6.5 v3.2 繁简英整合版( 2010 年 12 月 12 日更新 ),  
下载地址：<http://www.ed2000.com>ShowFile.asp?FileID=255645>。

## 3. 安装过程

打开 Virtual Box, 点击工具栏上的“新建”按钮, 弹出“新建虚拟电脑”向导, 选择操作系统类型为 Mac OS X 及 Mac OS X Server, 并为虚拟机设置一个名称 ( 比如 Snow Leopard ), 如图 2-5 所示。



图 2-5 “新建虚拟电脑”向导

点击“下一步”按钮, 设置虚拟机使用的物理内存, 请至少选择 1GB ( 如图 2-6 所示 )。



图 2-6 分配虚拟机使用的内存大小

在选择虚拟磁盘时，选择“创建新的虚拟硬盘”。为了取得更好的性能，虚拟硬盘类型选择“固定大小”（如图 2-7 所示）。



图 2-7 设置虚拟硬盘类型

虚拟硬盘容量至少设定为 30GB，并保证文件存放位置的可用空间是足够的（如图 2-8 所示）。



图 2-8 设定虚拟硬盘空间大小

点击“下一步”按钮，直至安装结束。

选择刚才创建的虚拟机 Snow Leopard，点击工具栏中的“设置”按钮，在弹出的虚拟机设置窗口左侧面板中选中“系统”，“启动顺序”选择“光驱、硬盘”，然后取消“启用 EFI”选项，如图 2-9 所示。

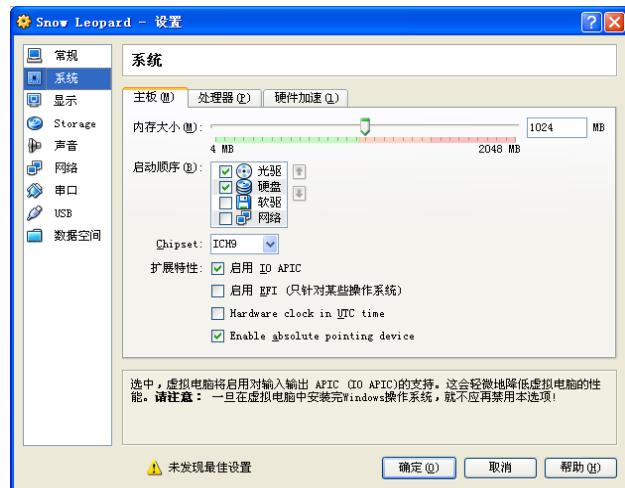


图 2-9 主板设置

选择左面板中“显示”项，将“显存大小”调为最大，然后选择“启动 3D 加速”选项（如图 2-10 所示）。

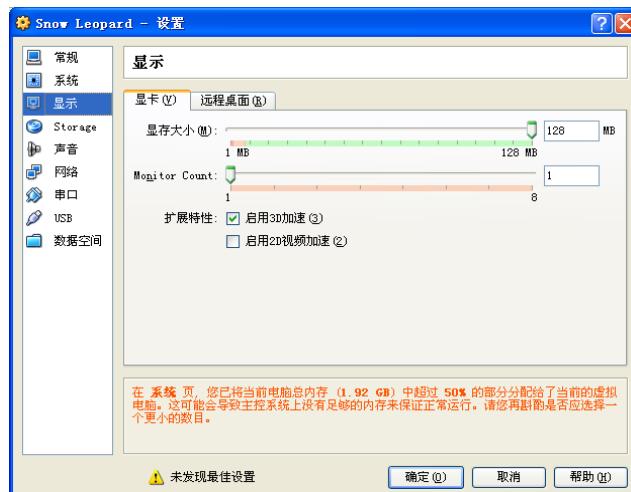


图 2-10 显卡设置

好的，虚拟机配置完成了，下面的步骤是安装 Snow Leopard。

在虚拟机设置窗口中，选择 Storage，在 IDE 控制器中添加一个虚拟光驱，然后为这个虚拟光驱添加一个盘片，把 iAntares OSx86 10.6.5 v3.2 的 iso 文件镜像加载进去(如图 2-11 所示)。

关闭设置窗口，双击虚拟机 Snow Leopard 启动虚拟机。如果顺利，虚拟机会用 iAntares\_v3.iso 进行引导，并进入 Snow Leopard 的安装界面。选取中文作为安装语言，然后从菜单“实用工具”中打开“磁盘工具”。

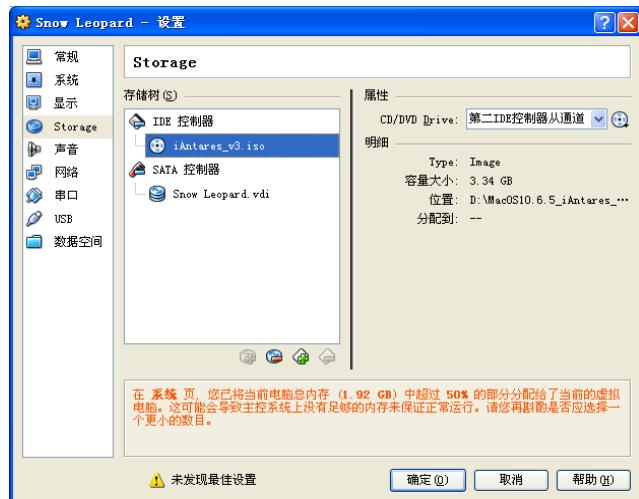


图 2-11 加载光盘镜像

在磁盘工具点击标签栏的“抹掉”，对磁盘进行格式化。文件系统格式为 Mac OS 扩展（日志式），然后点击按钮“抹掉”按钮（如图 2-12 所示）。

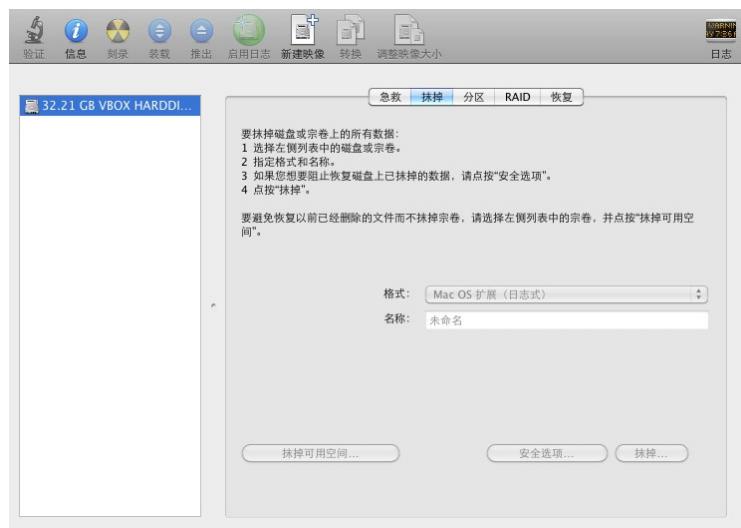


图 2-12 格式化磁盘

格式化完成后，选择格式化的磁盘作为安装目标，同时点击“自定”按钮。

在接下来的自定义安装界面中，“启动选项”除了后面3项以外全部选中，硬件驱动全部不需要选（虚拟机已经带硬件驱动），引导器选择变色龙 RC4 r684 而不是 RC5 r653，其余选项随意设置或保持默认值（如图 2-13 所示）。

这个步骤是整个安装中最重要的步骤，也许需要尝试很多次才知道最适合机器的设置。这个过程中需要不断地修改启动选项并重启，甚至可能会出现几次蓝屏。但在虚拟机中安装的好处就在于，除了出现蓝屏以外，都不需要按电源或 Reset 键，虚拟机重启的速度比硬启动要快许多。

这个步骤完成后就是缓慢的安装进度了，这需要一些时间，请耐心等候。

安装完成后，可能会出现“安装失败”的提示，不必惊慌，重启虚拟机后，会发现虚拟机引导菜单上多了一个 snow leopard 的引导选项，这个就是我们安装成功的 Mac OS 操作系统，另一个是安装光盘（如图 2-14 所示）。

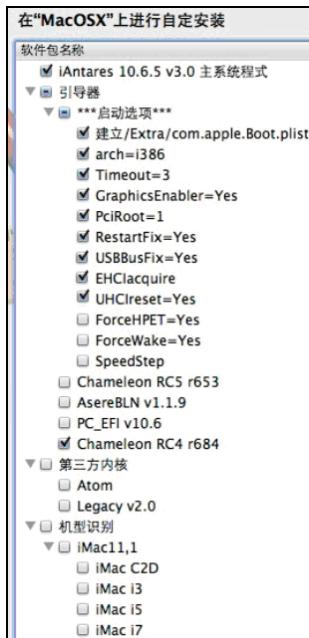


图 2-13 设置自定义选项



图 2-14 选择从新安装的 Mac OS 系统引导

用方向键把光标移动到 snow leopard 上，回车，变色龙开始从 Mac OS 进行引导。

启动后进入 Snow Leopard 桌面。由于某些 Bug，在这个桌面工具栏上会有 3 个图标显示为问号（如图 2-15 所示），当然如果为了美观，完全可以删除它们。

需要注意的是，如果 Mac 提示安装版本更新，请不要轻易更新系统，否则你可能进不了系统。因为破解的 Mac OS X 系统对系统内核进行了修改，如果升级的话有可能导致系统文件再次被覆盖，导致系统无法正常引导。

接下来需要下载 iOS SDK，并在 Mac 下进行安装。

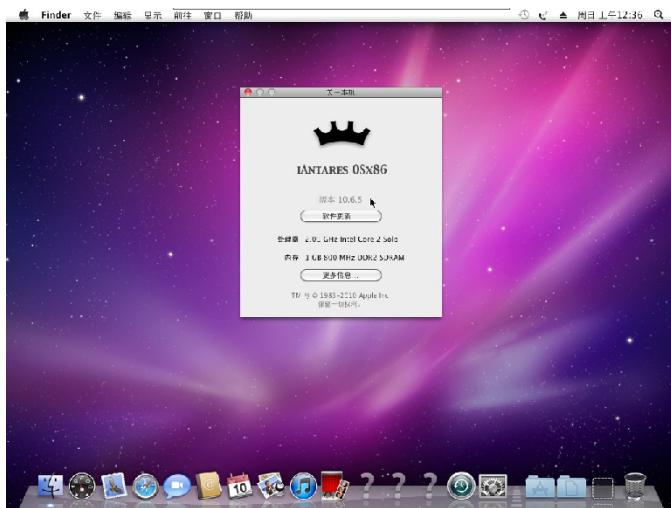


图 2-15 Snow Leopard 桌面

## 2.4.2 下载安装 SDK

每一个在苹果网站上注册了 iPhone 开发人员计划的程序员，都可以登录到以下地址下载最新版本的 iOS SDK：

<https://developer.apple.com/devcenter/ios/login.action>

这是一个几个 GB（根据版本不同）的 Mac 安装镜像文件，里面包括如下内容：

- Xcode 集成在 SDK 中一起发布，它支持苹果的 Objective-C 语言，也支持 C 和 C++ 代码。我们将在第 4 章介绍它的使用。
- Interface Builder 用于创建程序的 GUI，它和 Xcode 集成在一起，也可以单独启动。在本书很多地方仍然使用了它，第 5 章将对 Interface Builder 进行介绍。
- iPhone 模拟器可以在 Mac 中调试 iOS 应用程序，它的外观和真实的 iPhone/iPad 设备一模一样。使用它调试程序，比在真实设备中更方便快捷。在后面的章节中，会大量使用这个工具调试程序。
- Dashcode 也是/Developer/Applications 中的一部分，它是用于创建 Web 应用的优秀、极为精巧的图形开发环境，本书中不会使用到它。

双击下载后的文件，把 SDK 安装到 Mac 上。

接下来，创建我们的第一个 iOS 应用程序，以此检验我们的开发环境已配置成功。

## 2.5 写一个 iPhone 程序

点击桌面上的 Xcode 图标，启动 Xcode。选择菜单“File→New Project”，显示新建项目模

板向导（如图 2-16 所示）。

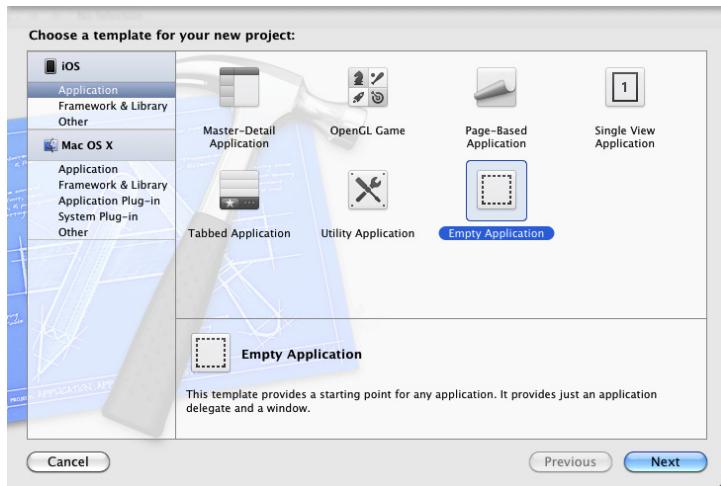


图 2-16 新建项目向导

在左边栏中列出了 Xcode 支持的两种项目类型：iOS 和 Mac OS X 项目，选择 iOS 下方的 Application，然后选择 Empty Application 类型的项目。点击 Next 按钮，进入新项目设置界面，如图 2-17 所示。

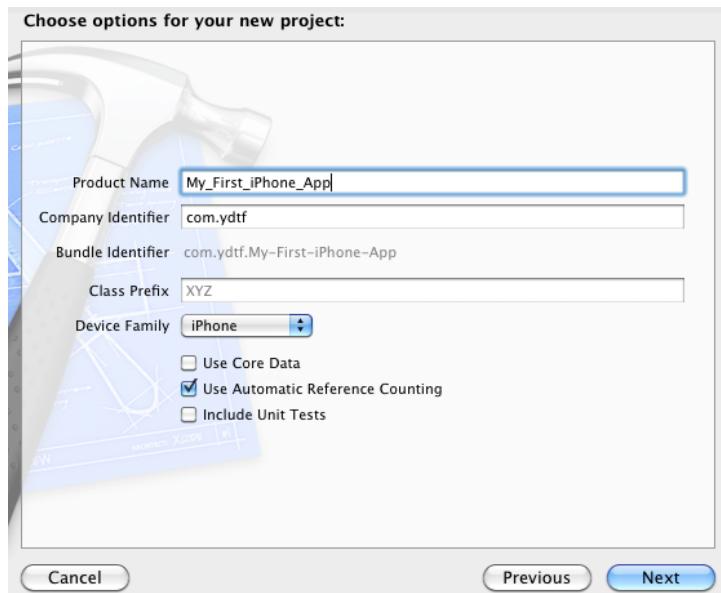


图 2-17 新项目设置界面

在新项目设置界面中，在 Product Name 栏填写项目名称，比如 My\_First\_iPhone\_App。在

Company Identifier 栏，填写公司名前缀，比如 com.ydtf。在 Device Family 栏填写所开发目标平台，比如 iPhone ( Universal 则表示 iPhone/iPad “二合一”版本)。然后点击 Next 按钮。

接下来是指定项目保存路径界面，如图 2-18 所示。

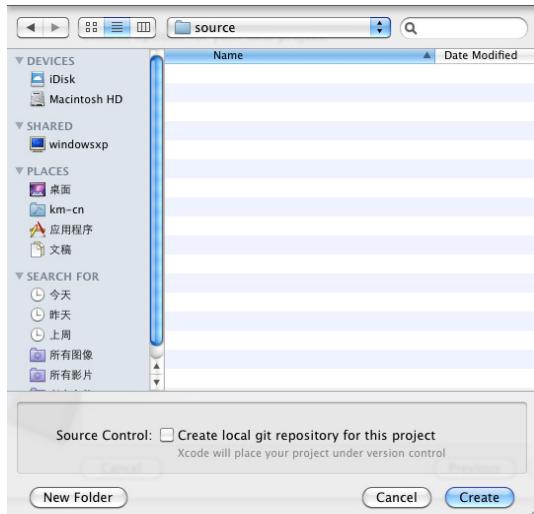


图 2-18 项目保存路径对话框

选择一个合适的项目保存路径，然后点击 Create 按钮。

这样，一个 iPhone 应用程序就创建好了。如图 2-19 所示是 My\_First\_iPhone\_App 项目的项目编辑界面，由于图太大，这里只显示了窗口的一部分。

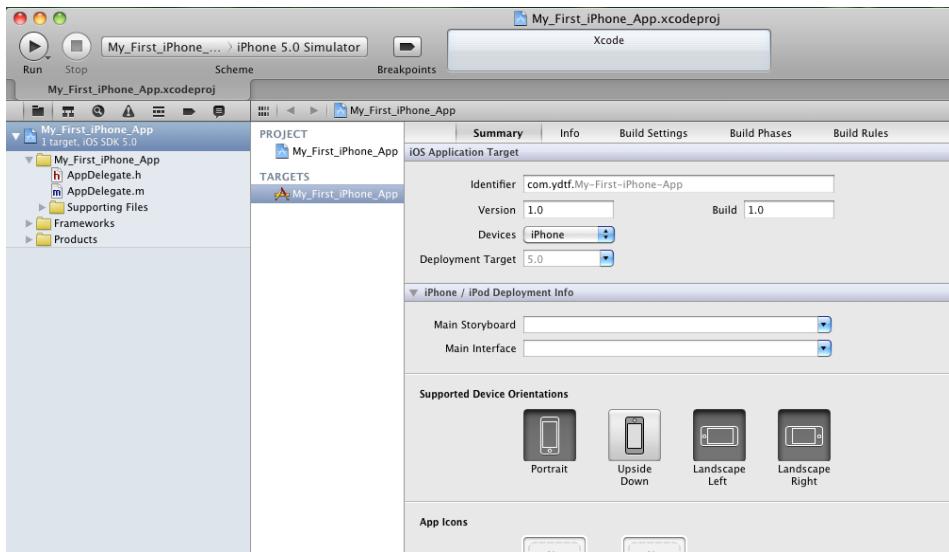


图 2-19 项目编辑窗口

界面的左侧是 Project Navigator 窗口，它列出了项目的所有资源，包括源文件、.xib、.plist、框架/库、二进制和图片等。右侧是指定资源（文件）的 Info 窗口或编辑窗口，我们主要的编辑工作都在这里完成。

---

**提示：**如果你看不到 Project Navigator 窗口，可以通过菜单“View→Navigators→Show Project Navigator”来重现它。

---

在 Project Navigator 中选择 My\_First\_iPhone\_App 文件夹，单击右键，选择“New File”菜单，弹出新建文件模板向导，如图 2-20 所示。

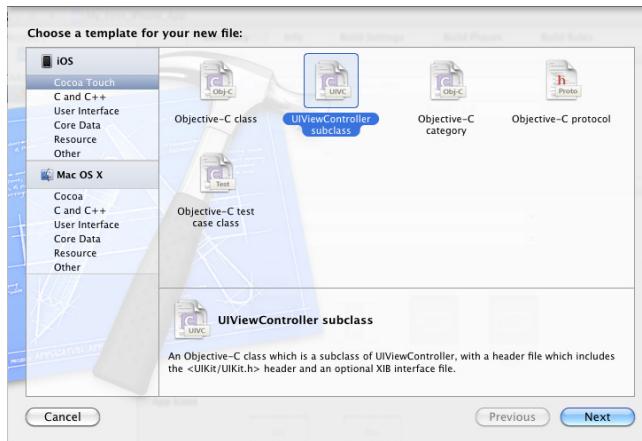


图 2-20 新建文件模板选择向导

Xcode 4.2 能创建各种各样的文件。我们选择 iOS/Cocoa Touch 下的“UIViewController subclass”，然后点击 Next 按钮，将弹出如图 2-21 所示的新文件设置向导窗口。

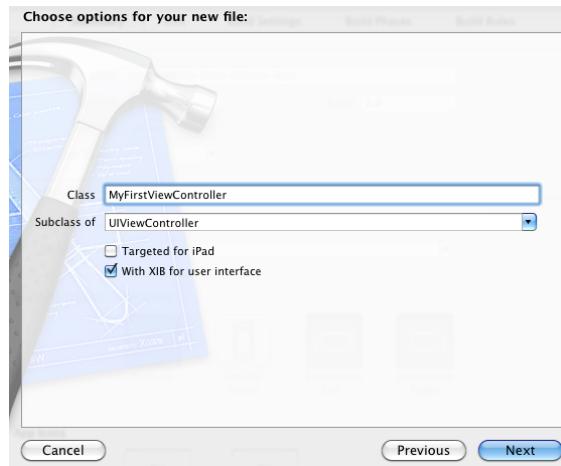


图 2-21 新文件设置向导

在该窗口中，输入类的名称，如 MyFirstViewController。勾选“With XIB for user interface”选项，点击 Next 按钮，进入文件保存路径窗口，如图 2-22 所示。

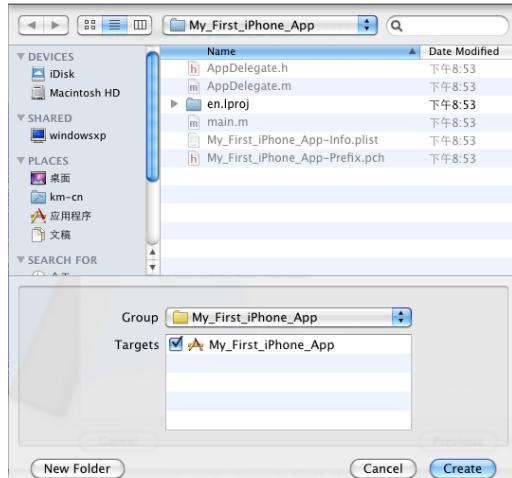


图 2-22 新文件保存路径界面

点击 Create 按钮。默认情况下，将转入 MyFirstViewController.xib 文件的编辑界面（即 Interface Builder 界面），如图 2-23 所示。

---

**提示：**与 Xcode 3.2 不同，在 Xcode 4.2 中，Interface Builder 是真正集成在 Xcode 的 IDE 中，而不再单独存在。

---

此时，在 Interface Builder 的右侧（用于全屏太大，图 2-23 不能显示出来），可以找到如图 2-24 所示的 Object Library 窗口。

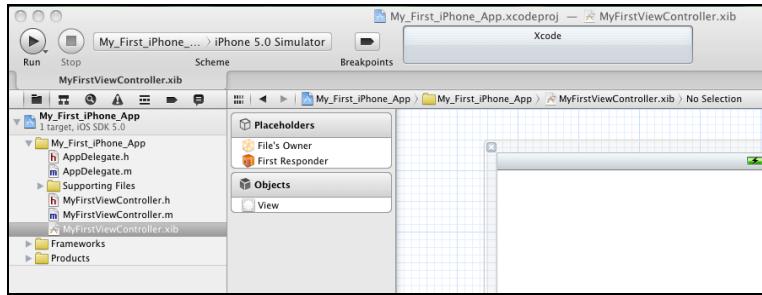


图 2-23 Xcode 中集成的 Interface Builder 界面

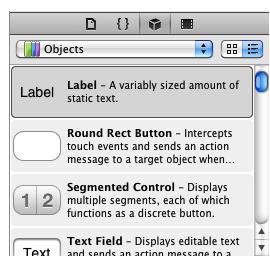


图 2-24 Object Library 窗口

我们从 Object Library 窗口中找到一个 Label 对象（就在 Object Library 窗口的第 1 行），然后按住它不放，直接把它拖放到 MyFirstViewController 的编辑窗口中（Interface Builder 中），结果如图 2-25 所示。



图 2-25 在 MyFirstViewController 中加入一个 Label

然后双击图 2-25 中的 Label 对象，将它的文本修改为“嗨，这是我的第 1 个 iPhone App！”，如图 2-26 所示。



图 2-26 修改 Label 的显示文本

你可以任意拖动标签控件改变它在窗口中的位置。保存在 Interface Builder 中所做的更改（快捷键 **⌘ + S**）。

---

**提示：**对于 Windows 键盘，win 键对应苹果键盘中的苹果键**⌘**。

---

在 Project Navigator 窗口中找到源文件 AppDelegate.m，选中它，我们将对其进行一些编码工作。在 AppDelegate.m 的编辑窗口的顶部#import “AppDelegate.h” 一行后换行，增加以下代码：

```
#import "MyFirstViewController"
```

找到方法：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
    (NSDictionary *)launchOptions
```

在其中 “self.window.backgroundColor = [UIColor whiteColor];” 一行后增加以下两行代码：

```
MyFirstViewController* vc=[[MyFirstViewController alloc] init];
self.window.rootViewController=vc;
```

接下来要运行这个程序，看看它最终实现的效果。

## 2.6 在模拟器上运行应用程序

在 Xcode 4.2 的顶部工具栏中，找到 Scheme 按钮。从该按钮的下拉菜单中选择“iPhone 5.0 Simulator”，如图 2-27 所示。



图 2-27 修改项目的 Scheme

---

**提示：**如果 Xcode 4.2 的工具栏未显示，请选择菜单“view→Show Toolbar”。

---

然后点击工具栏中的 Run 按钮（快捷键  $\text{⌘+R}$ ），会启动 iPhone 模拟器界面，并通过 iPhone 模拟器来运行我们的 My\_First\_iPhone\_App 应程序（如图 2-28 所示）。



图 2-28 在模拟器中运行应用程序

## 2.7 在 iPhone 上运行应用程序

如果要在 iPhone 手机上运行程序则没有那么容易了。

正如第 1 章所述，在开始开发 iPhone 应用程序之前，你需要注册成为 iPhone 开发人员。只有这样，苹果公司才会允许你使用“完全的”的 SDK，否则你只能下载一个有功能限制的免费 SDK。

注册页面位于 <http://developer.apple.com/iphone>。苹果将该注册程序称为苹果开发者计划（Apple Developer Plan），其中针对 iPhone 开发人员的称作 iOS 开发者程序。在该页面的底部提供了苹果支持的所有注册程序（见图 2-29）。

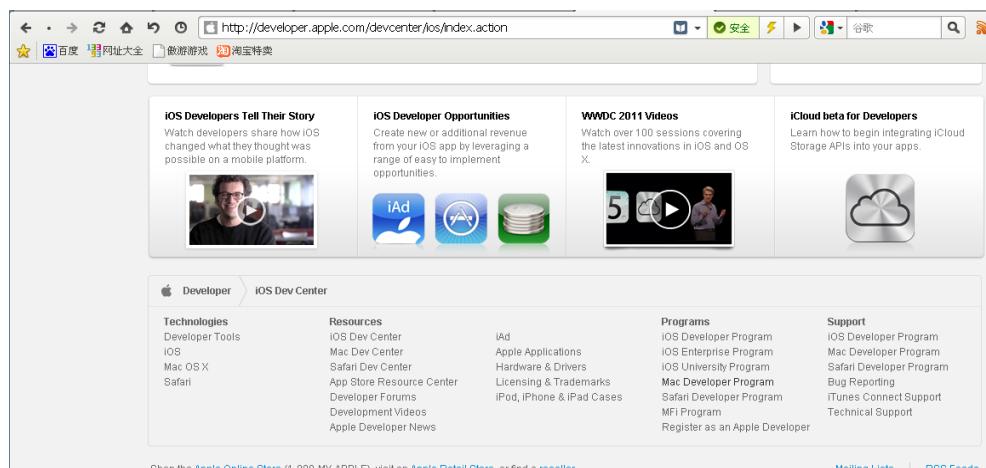


图 2-29 苹果开发者计划

在该页面底部的 Programs 列表中，列出了所有的 iOS 开发者程序类型（排在列表的头 3 项）：

- iOS Developer Program
- iOS Enterprise Program
- iOS University Program

iOS University Program 程序是免费的，面向科研和教学人员。它也提供了完整的 Xcode 和 iPhone 模拟器，但不支持将应用程序在真实的 iPhone（iPod Touch 或 iPad）中运行，而且也不支持通过 App Store 发布应用程序。

iOS Developer Program 程序是开发者们最常用的版本，即标准版 IDP，它的价格是 99 美元/年。它提供了一个 Xcode，一个 iPhone 模拟器——支持在 Mac 上运行绝大多数 iPhone 程序。标准版 IDP 支持通过苹果 App Store 分发应用程序，并可在 iPhone 上（不是模拟器上）调试应用程序。

iOS Enterprise Program 程序即企业版 IDP，在第 1 章中已经做了详细的介绍，它的价格是 299 美元/年。

当你拥有了标准版或企业版的 IDP 证书，还需要通过 Provision Portal 制作相应的 Provisioning Profile，并下载到你的电脑上。然后使用 IDP 对程序进行代码签名，才能在 iPhone 上运行你开发的程序，这个过程正如第 1 章所述。如果你已经仔细看完第 1 章，那么我们可以假设你已经完成了这些步骤。接下来就可以在真机上运行（调试）程序了。

将 iPhone 连接到电脑，Xcode 会自动识别出你的 iPhone。将项目的 Scheme 修改为你的 iPhone 名字，例如作者的 iPhone 名为“YHY's iPhone”，如图 2-30 所示。然后点击 Run 按钮，Xcode 将会在你 iPhone 上运行 My\_First\_iPhone\_App 程序了。



图 2-30 在设备上运行程序

---

提示：如果是第一次调试这个设备，则 Xcode 会提示要在该设备上安装一个 Provisioning Profile，请选择同意安装。

---

# 第 3 章 Objective-C 语法简介

本书不是一本关于 Objective-C 编程语言的专著，但仍然会介绍一些 Objective-C 语言的语法基础和有趣特性。这对于刚刚接触到 iPhone 编程的人来说，会是一个很好的开始。

Objective-C 兼具 C 语言和面向对象语言的特性。因此本章会从两个方面来介绍 Objective-C (简称 O-C)。

首先，由于 Objective-C 具有 C 语言背景 (它仍然是一种 C)，它从 C 语言中继承了一些 C 语言特性，例如：Objective-C 全面支持 C 的数据类型 (包括简单类型和复合类型)、常量/变量/宏、包含头文件 (import/include 指令)、函数、条件和循环控制语句。

其次，Objective-C 在 C 语言的基础上进行了扩展，加入了面向对象的内容。例如：对象和类 (方法及属性的集合)、Objective-C 的消息机制、内存管理 (包括对象生命周期和对象的创建和释放)、类别和协议、反射、谓词，以及 Cocoa 对一些常见模式 (Pattern) 的 Objective-C 实现 (如 MVC、KVO)。

在本书后续的一些章节中，使用了块语法。因此本章最后单独对 Objective-C 的块编程进行了介绍。块是现代 C 语言标准的一部分，类似 C++ 中的 inline 函数。Objective-C 以面向对象的方式对 C 的块进行了封装。

你可以有针对性地阅读这些内容。比如对于有 C 基础的程序员，应该跳过基本语法部分 (主要是 C 语言特性)，重点阅读 Objective-C 在 C 语言基础上进行的扩展 (如 NS 类、类别和协议)，以及 Cocoa 提供的一些优秀特性 (比如 MVC、KVO)。

## 3.1 Objective-C 的 C 语言特性

Objective-C 源自 C，它是 C 语言特性和 Smalltalk 语法的集合。从 20 世纪 80 年代开始，Objective-C 对 C 语言进行了大量的扩展，直至 30 年后的今天，Objective-C 已经发展成为当下最流行的编程语言之一。

Objective-C 全面支持 C99 标准。对于 C 这种程序员早已熟知熟悉的经典语言，作者在此并不准备多做介绍，你可以阅读大学计算机课程中使用的 C 语言教程，或者 Dave Mark 编写经典专著《Learn C on the Mac》。本章只简单介绍 Objective-C 中一些主要的 C 语言特性，比如 Objective-C 支持的 C 语言数据类型、常量/变量和函数、条件和循环等，以及 Objective-C 的一些非 C 语言特性 (面向对象特性)。

### 3.1.1 一个简单的 Hello World

了解一种新语言是从一个最简单的程序开始，Hello World 程序无疑是最简单的，让我们

从它开始。

打开 Xcode，使用菜单“New Project”打开新建项目向导，在项目模板中选择“Mac OS X → Application → Command Line Utility → Foundation Tool”，并可以将项目命名为 HelloWorld。为了能更清晰地了解 Objective-C 这门语言，我们特意选择了最简单的命令行项目，这种类型的项目并不能在 iPhone 上运行，它是在 Mac 命令终端环境中运行的。要知道，原本 Objective-C 就是用来设计 Mac 程序的，后来才扩展使用到 iPhone 程序。因此，我们避免了使用 Cocoa 框架带来的一些复杂性。

Xcode 为我们生成了一系列文件，打开其中 HelloWorld.m，查看 Xcode 自动产生的代码如下所示：

```
#import <Foundation/Foundation.h>
int main (int argc, const char * argv[]) {
    pool = [[NSAutoreleasePool alloc] init];
    //插入代码
    NSLog(@"%@", @"Hello, World!");
    [pool drain];
    return 0;
}
```

整个代码包括注释不超过 10 行。然而对于从未接触过这门语言的人来说，可以从中学习到许多东西。

### 3.1.2 Objective-C 是另一种 C

C 语言作为传统的计算机教学语言，对许多接受过大学理工科教育的人来说，并不陌生。从上面的代码中，我们看到了许多 C 语言的特征，比如语句用分号结束，程序开头同样需要导入头文件，还有一模一样的 main 函数作为程序的入口，等等。实际上，Objective-C 完全兼容标准 C，它同样采用 gcc 编译器，可以在 Objective-C 程序中随意嵌入 C 代码。对于许多推崇标准 C 的 iPhone 程序员来说，他们更倾向于使用 C 代码而不是 Objective-C 代码来写 iOS 应用程序。

因此，虽然 Hello World 是一个 Objective-C 程序，但是我们却可以把它变成 C 程序，将代码修改为：

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

编译运行程序，黑黑的控制台窗口输出了同样的文字内容。两段代码的区别仅在于：HelloWorld 的 C 语言版没有使用任何框架，而 Objective-C 版本则使用了 Foundation 框架。

同 C 一样，Objective-C 源代码也分成了两部分：接口和实现。Objective-C 的接口文件采

用.h 作为后缀名；实现文件的后缀则采用.m，而不是.c（C）或者.cc（C++）。

### 3.1.3 数据类型

在第 2 章的 Cocoa 框架介绍中，提到过 Foundation 框架是 Cocoa 框架的基本框架之一（见图 2-2 Cocoa 框架）。而 Foundation 框架中就定义了数量众多的数据类型和底层类，例如 NSString、NSArray、NSEnumerator 和 NSNumber。我们不可能全部介绍它们，整个 Foundation 框架定义了上百个类。但我们会讨论其中最有用的一些。

前面讨论的 HelloWorld 程序是很重要的，起码对于这一章来说。因为我们后面学习的代码都会基于这个 main 函数，只需在这个函数中加入少量的代码，程序就可以运行。

#### 1. C 基本数据类型

由于 Objective-C 和 C 兼容，C 语言中的 5 种基本类型：char、int、float、double、void 在 Objective-C 中仍然可用，并且这些基本类型可以使用的修饰符 unsigned、signed、short、long 进行修饰。

#### 2. C 复合类型

C 语言中的复合类型：数组、指针、结构、联合、枚举仍然得到了支持。此外，Objective-C 中特有的几种复合类型，就是 id、SEL、nil 和 NULL。

id 是指向任意对象的指针，类似 C 语言中的 void\*。这里的任意对象是指 Objective-C 中任何继承自 NSObject 类的实例。在 Cocoa 中，NSObject 是所有类的根类。所以，可以用 id 指向任何类型的 Objective-C 对象。

选择器 SEL 实际上被定义为 const char\*，在 Objective-C 中，用它来指向任何方法的定义，等同于 C 语言中的函数指针。要创建 SEL 类型有两种方式，一种是使用@selector()关键字，并在括号中传递一个方法签名作为参数，通过这种方式你可以调用一个 Objective-C 对象的指定方法：

```
[object performSelector:@selecootr(doSomething)];
```

另一种就是通过 NSSelectorFromString()函数，把方法签名以字符串的方式作为参数传递，这种方式类似于 Objective-C 的反射（关于选择器，后面还会介绍）。

nil 和 NULL 都代表了空指针。nil 用于 Objective-C 对象，而 NULL 则用于指针类型，并且二者不可互换。任何 Objective-C 对象都是 NSObject 类的子类，它们都可以用 id 来指向，因此可以使用 nil 指针来初始化为空。而 C 指针类型指向的数据是结构体，而不是对象，因此不能用 nil 指针初始化为空，但可以用 NULL 指针初始化为空。此外，NSNull 用于指向集合对象，表示集合为空，没有任何有效的对象。

#### 3. 布尔类型

由于 C 没有布尔类型，你可以使用 C++ 中的 BOOL 类型。但 Objective-C 提供了自己的布尔类型 BOOL，它提供了两个表示真和假的名义值：YES 和 NO，类似于 True 和 False。让人

奇怪的是，它虽然属于 Cocoa，却没有使用 NS 作为前缀（如后所述）。

然而使用 BOOL 类型时要小心，否则可能会导致出人意料的事情发生。

因为 BOOL 的定义其实是带符号字符，占 1 个字节的存储空间。我们实际上可以把一个字节的数据放到 BOOL 中去，如果把一个超过 1 字节的整数（如 short 或 int）放进去，那么只有低位字节会被放进 BOOL 中去。在其他语言中，0 定义为假，非 0 定义为真。而在 Objective-C 中不同，1 定义为 YES，0 定义为 NO。0 和 1 以外的数据即不是 YES，也不是 NO。例如，我们习惯于这样的语句：

```
if(isNotEquals(a,b)==YES) { // 如果 a,b 不相等
    ...
} else{ // 如果 a,b 相等
    ...
}
```

`isNotEquals (a,b)`是一个函数，无论它的返回结果是什么，在 if 的条件表达式中，它首先会被转换为 BOOL 类型。但是即使 `isNotEquals (a,b)`函数真的被定义为 BOOL 类型，它也不一定只返回 YES 和 NO (1 和 0)。原因如前面所述，如果在 `isNotEquals` 函数中返回其他类型，比如 int，仍然可以正常转换为 BOOL，Objective-C 并不会检查出错误。实际上，在 `isNotEquals` 函数中很可能是这样返回的：

```
return a-b;
```

如果 a 和 b 相等，那么返回的实际是 NO，即 0。如果 a 为 4，b 为 3，则返回的是 1，即 YES。这都没有什么问题。但如果 a 为 5，b 为 2，那么实际上返回的这个布尔值即不是 YES，也不是 NO，而是整型 3。那么接下来的 if 语句中，3 自然不等于 YES ( YES=1 )，你会发现程序的结果是 a 与 b 相等。问题是，你觉得 5 和 2 应该相等吗？

解决问题的方式有两种，要么你要确保每一个 BOOL 值返回的除了 YES 和 NO 外不能有其他值，比如上面的返回语句应该修改为：

```
Return a-b!=0;
```

要么，在条件表达式中，永远只和 NO 进行比较，而不要和 YES 进行比较，比如：

```
If(isNotEquals(a,b)==NO) { // 如果 a,b 相等
    ...
} else{ // 如果 a,b 不相等
    ...
}
```

喜欢使用哪种方式，完全由你个人的喜好来定。

#### 4. NS 结构体

Cocoa 的 Foundation 框架中定义了大量的结构体，如 `NSRange`、`NSPoint`、`NSSize` 等，这些结构体统一用 NS 作为前缀，在后面的章节中你还会见到一些其他的 NS 结构体。

下面介绍一些有用的 NS 结构体。

### (1) NSRange

NSRange 的定义是：

```
typedef struct _NSRange{
    unsigned int location;
    unsigned int length;
}NSRange;
```

它有两个有用的成员，location 表示范围从哪个数字开始，默认为 0，length 表示范围的大小，默认值是 NSNotFound，表示不存在的范围。比如：

```
NSRange range={3,6};
```

这个语句用 C 语言的风格创建了一个 NSRange 变量，这个 NSRange 表示了从 3 开始、总长度为 6 的数值范围。

或者用给结构成员赋值的方式初始化 NSRange：

```
NSRange range;
range.location=3;
range.length=6;
```

但是 Cocoa 还为所有的结构提供了便利函数 NSMakeRange：

```
NSRange range=NSMakeRange(3,6);
```

可以很方便地创建一个 NSRange。

### (2) NSPoint、NSSize 和 NSRect

结构体可能是最方便表述几何图形的数据类型了，比如 NSPoint、NSSize 和 NSRect。这 3 个结构体构成了一个二维平面中最基本的数据类型。

NSPoint 表示物体在二维坐标中的 x 位置和 y 位置：

```
typedef struct _NSPoint{
    float x;
    float y;
}NSPoint;
```

NSSize 用于表示一个形状的大小，任何占据一定二维空间的形状，总是离不开宽和高这两个属性：

```
typedef struct _NSSize{
    float width;
    float height;
}NSSize;
```

NSRect 用于表示一个矩形，它由矩形的坐标位置和大小共同构成，很显然位置可以用 NSPoint 表示，而大小就用 NSSize 表示：

```
typedef struct _NSRect{
    NSPoint origin;
    NSSize size;
}NSRect;
```

## 5. NS 类

既然 Objective-C 是“面向对象的 C”，那么除了对 C 的基本数据类型进行全面支持外，当然也对这些类型进行了面向对象的封装和扩展。Cocoa 框架把所有的类都加了“NS”前缀——来自于 Cocoa 前身 NextSTEP 的缩写，以显示和 Core Foundation 类的区别（以 CF 为前缀），比如 NSString、NSNumber、NSArray、NSDictionary。本书中会大量使用这些 NS 类。

与 C 语言寥寥数种的基本数据类型不同，NS 类家族可谓种类繁多，本书不可能逐一讨论。在此，我们将只介绍几种最常用的 NS 类。

### (1) NSString 和 NSMutableString

在 HelloWorld 程序中有一个打印语句：

```
 NSLog(@"%@", @"Hello, World!");
```

其中`@"Hello, World!"`就是一个字符串，即 NSString。注意，使用@和双引号来括住字符串，而不是像 C++一样只使用双引号来括住字符串，这样就是一个 NSString 类型的字符串常量。

NSString 有许多封装的特性，比如统计字符串的长度，将自身与其他字符串进行比较，方便地将自身转换为整数或浮点数。

出于性能上的考虑，你不能改变 NSString 的值，比如在 NSString 对象中临时插入其他字符或者删除某些内容，这跟 Java 语言中的 String 类型是一样的。当然当你确实需要这样做的时候，可以使用 NSString 的可变版本，NSMutableString。后者和前者几乎一模一样，除了可以对其储存的字符串进行改变操作。

创建一个 NSMutableString 可以用以下代码：

```
NSMutableString *mutableStr=[NSMutableString stringWithString:@"Following me"];
```

现在，我们可以任意替换其中的字符内容：

```
NSRange range={10,2};
[mutableStr replaceCharactersInRange:range withString:@"you"];
```

于是，这个字符串的内容变成了“Following you”。如果想把这个字符串中的“me”删除，可以用 NSString 的 deleteCharatersInRange 方法：

```
[ mutableStr deleteCharactersInRange:range];
```

当然 NSMutableString 类最常用的方法还是 appendString 方法，在字符串的末尾加上另一个字符串：

```
- (void) appendString:(NSString*)nsstring;
```

### (2) NSNumber

NSNumber 用于处理所有与数字相关的数据类型，从整型到浮点型的数据，也包括前面提到的 BOOL 类型。我们可以通过一系列的方法把基本数据类型包装为 NSNumber：

```
+ (NSNumber*) numberWithChar:(char)value;
+ (NSNumber*) numberWithInt:(int)value;
+ (NSNumber*) numberWithFloat:(float)value;
+ (NSNumber*) numberWithBool:(BOOL)value;
```

类似的方法还有很多，包括各种 unsigned 和 long 型数据的版本，我们不能一一列举。创建了 NSNumber 对象之后，我们就可以把它存储到 NSArray 和 NSDictionary 集合中去，而此前这两种集合对象是无法存放基本类型数据的。当我们从集合中把 NSNumber 取出来后，又可以通过以下方法将其还原为原来的基本类型：

```
- (char)charValue;
- (int)intValue;
- (float)floatValue;
- (BOOL)boolValue;
...
```

很显然，这些方法是和创建 NSNumber 的方法一一对应的。

### (3) NSArray 和 NSMutableArray

NSArray 与 C 语言中的数组是类似的，但是它还封装了很多 C 数组不具备的特性。比如，它不需要像数组一样，随时随地都需要程序员对数组下标越界进行检查和判断。它其实更像 Java 语言中的 ArrayList。

但是正如前面提到过的，NSArray 只能存储 Objective-C 对象，而不能存放基本数据类型，如 int、float、char、struct、enum 等。

**提示：** NSDictionary 也有同样的限制。

此外，NSArray 也不能存放空值 nil，因为 nil 用作 NSArray 的结束符。NSArray 的最后一个元素永远是 nil 对象。

NSArray 有一个便利的创建方法，可以一次性创建一个包括了许多元素的 NSArray 对象，如下所示：

```
NSArray* array=[NSArray arrayWithObjects:@"one",@"two",@"three",nil];
```

这个方法提供了数目不定的参数作为其初始的创建时就包含的元素。牢记前面的两个限制：1) 其元素只能是对象；2) 最终要以一个 nil 结束。

要获得 NSArray 的元素数目很容易，访问其 count 属性即可。

使用以下方法访问指定索引的对象：

```
- (id)objectAtIndex:(unsigned int)index;
```

如果索引大于数组元素中的个数，Cocoa 会抛出 NSRangeException 异常。

然而，NSArray 是不可变的对象数组，一旦创建就不可能增加和删除其中包含的对象——当然对象自身还是可以改变的。但是数组的元素个数不会改变，其中的元素也不能替换。

如果你需要一个在创建后仍然可以任意增加和删除元素的数组，可以使用 NSArray 的可变版本 NSMutableArray。

通过类方法 arrayWithCapacity 和 arrayWithArray 来创建一个可变数组：

```
+ (id)arrayWithCapacity: (unsigned)numItems;
+ (id)arrayWithArray: (NSArray*)array;
...
```

创建 NSMutableArray 的类方法有很多，但最常见的是这两个。创建之后就可以用 addObject 方法从数组末尾添加对象：

```
- (void)addObject: (id)anObject;
```

同样，获取某个索引处的对象使用继承自 NSArray 的 objectAtIndex 方法。

NSMutableArray 还可以删除某个索引处的对象：

```
- (void)removeObjectAtIndex: (unsigned)index;
```

同 C 数组一样，NSArray 和 NSMutableArray 也是从 0 开始索引的。此外，可变数组可以在特定索引处插入/替换某个对象，进行数组排序等。

#### (4) NSDictionary 和 NSMutableDictionary

字典也称散列表( Java 语言中 )或者关联数组( C 语言中 )，在 Objective-C 中用 NSDictionary 表示字典。创建一个 NSDictionary 通常是使用类方法：

```
+ (id)dictionaryWithObjectsAndKeys: (id)firstObject, ...;
```

...代表了可变参数数组，一系列个数不能确定的同类型参数的集合。跟 NSArray 一样，一旦创建并初始化了一个 NSDictionary，其包含的对象就不能改变：

```
NSDictionary* d=[NSDictionary dictionaryWithObjectsAndKeys:
    obj1, @"first object", obj2, @"second object", nil];
```

同样，在字典的最后以 nil 标志结束。在初始化方法的参数列表中，除了 nil 外，所有的参数都是成对的，比如：对象 obj1 和字符串 “first object” 是一对，对象 obj2 和字符串 “second object” 是一对。每对都代表字典集合中所存储的一个对象和它对应的索引键，当然，索引键在字典中是唯一的。我们可以通过索引键来检索字典中存放的对象：

```
- (id)objectForKey: (id)aKey;
```

这样，通过键 “first object” 查找字典 d 中存储的对象 obj1：

```
id something=[d objectForKey:@"first object"];
```

NSMutableDictionary 是可变字典。假设需要在字典 d 中增加新的对象，我们可以创建

NSMutableDictionary:

```
NSMutableDictionary* mutableD=[NSMutableDictionary
    dictionaryWithDictionary:d];
[d setObject:obj3 forKey:@"third object"];
```

如果新加入的对象所采用的索引键与字典中已存在的键相同，则原来的对象会被覆盖。这就是为什么方法名采用了 `setObject` 而不是 `addObject` 的原因。

要删除字典中的对象，可采用如下方法：

```
- (void)removeObjectForKey:(id)aKey;
```

#### (5) NSDate 和 NSDateFormatter

日期的显示非常重要，我们最常用到日期的地方就是获取当前时间，这也是为什么可以使用 iPhone 手机用来取代手表，并用它来做烦人的闹钟。

`NSDate` 的 `date` 类方法创建了一个临时的 `NSDate` 对象，它代表了当前时间：

```
NSLog(@"%@", [NSDate date]);
```

这会在控制台输出如下内容：

当前时间：2011-7-30 15:37:02 -0570

`NSDateFormatter` 类用于和 `NSDate` 配合，将各种格式的字符串转换为 `NSDate` 或从 `NSDate` 转换为指定格式的文本。

日期转换为文本：

```
NSDateFormatter* df=[[NSDateFormatter alloc] init];
[df setDateFormat:@"yyyy/MM/dd"];
NSLog(@"%@", [df stringFromDate:[NSDate date]]);
```

从文本转换为日期使用：

```
NSDate *date=[df dateFromString:@"2011-7-30"];
```

### 3.1.4 常量、变量和宏

Objective-C 的变量声明和 C 完全相同：类型<变量名>。如果是声明对象的话，需要使用指针符号“\*”，例如前面曾经见到过的许多代码：

```
NSDate* date;
NSString *string;
NSArray* array;
...
```

常量可以使用 `static` 关键字声明：

```
static NSString *string=@"abc";
```

也可以通过预定义宏来声明：

```
#define PI 3.14
#define PATH @“images”;
```

### 3.1.5 #include 和#import

C 和 Objective-C 都使用头文件来声明类型、结构体、符号常量和函数原型等，因此需要通知编译器去头文件中查找相应的定义。为了实现这个目的，你可以使用#include 和#import 语句，二者的区别仅在于#import 是 GCC 编译器提供的，它可保证同一个头文件无论被包含多少次始终只导入一次。由于这个特性，你完全可以在 Objective-C 程序中用#import 取代#include。

HelloWorld 程序中的#import <Foundation/Foundation.h> 文件语句中的第一个 Foundation 并不代表目录，而是指我们在第 2 章中提到过的 Foundation 框架。

---

**提示：**包含头文件时，框架和库中的头文件用尖括号引住，项目中的头文件用双引号引住。如#import <Foundation/Foundation.h> 和 #import “ViewController.h”。

---

### 3.1.6 函数

Objective-C 支持函数的声明和定义。但函数不是类的一部分。函数可以在.h 头文件中进行声明，也可以直接在.m 文件中定义（调用前）。关于函数的声明和定义，请遵循标准 C 的语法规规范。

### 3.1.7 分支和循环

Objective-C 支持 C 的所有分支语句和循环语句：switch 语句、判断、while 循环、do...while 循环和 for 循环。除此之外，从 Objective-C 2.0 开始，支持快速迭代，类似 VB 中的 For...each 语法。使用快速迭代，可以简单地遍历数组元素：

```
For(NSString *each in array) {
    NSLog(@"%@", each);
}
```

其中，array 是一个字符串数组。显然，这样的语法显得更加简洁明快。快速迭代不仅仅可用于数组，也可以用于遍历任何集合对象。

## 3.2 面向对象的 C

从现在开始，我们开始介绍期待已久的 Objective-C 的面向对象特性。

### 3.2.1 类和对象

面向对象最重要的概念就是类。通过类，我们可以实现面向对象的两个主要特性：继承和聚合。

在 Cocoa 框架中，`NSObject` 是所有类的根类，其他所有类从此开始继承。

### 1. 类的定义

类的定义在接口.h 文件中进行，典型的类定义如下面的代码所示：

```
@interface MyClass: NSObject
{
    NSString *name;
    NSArray* array;
    ...
}
@property(nonatomic,retain) NSString *name;
@property(nonatomic,retain) NSArray* array;
-(id)initWithName:(NSString*)string;
...
@end
```

从上面的例子中，我们可以看出以下几点：

- 类的定义由`@interface` 开始，到`@end` 结束；
- 类名后面紧跟冒号和所继承的父类名；
- 花括号中定义类的成员变量；
- 属性由`@property` 关键字声明；
- 方法声明位于成员声明之后，`@end` 之前。

### 2. 方法

方法类似于函数，虽然形式上有区别，但仍然由返回值、方法名和参数列表构成。其中，返回值和参数的类型说明使用圆括号括住，方法名分为多个部分，有几个参数，方法名就被分成几个部分，每个部分都有一个冒号分隔，冒号后面才是参数类型和参数名。此外，方法一般由一个方法类型符（+号或者-号）修饰，+号表示方法为类方法，-号表示方法为实例方法。例如：

```
- (id)initWithName:(NSString*)string withArray:(NSArray*)arr;
```

方法开头的-号表示这是一个实例方法。实例方法和类方法不同，实例方法属于实例对象所拥有，必须通过类的实例来调用；而类方法属于类所有，它直接通过类来调用，甚至不需要创建实例对象（等同于 C++/Java 中的 static 方法）。

这个方法有两个参数（数一数参数列表中冒号的个数，一个冒号代表一个参数），因此方法名由两部分组成，两个带冒号的单词：`initWithName:` 和 `withArray:`。

记住，方法名实际上由“方法名+参数名”两部分构成。而参数名是后面带冒号的单词，如果该方法没有参数，则方法名仅包含第 1 部分（即只有方法名，而没有参数名）。而且方法的第 1 个参数的参数名就是方法名。因此有多少个参数，方法名就会由多少个带冒号的单词组成。记住这一点，因为这跟现存的许多语言都不一样。在 Objective-C 中，总是用 `initWithName:withArray:` 这样的形式表示一个方法名，而不是像 Java 一样用 `init` 来表示方法名。

Objective-C 吸收了 C 和 C++的一些令人称道的优点，比如可变参数。如果你在使用方法中总是拿不定总共需要多少个参数，那么你可以使用可变参数，例如 `NSString` 的 `stringWithFormat:` 方法：

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

这个方法的第一个参数是 `NSString` 类型的参数 `format`，但第二个参数就是一个可变参数。这样的参数不仅没有确切的参数数目，而且也无法得知其具体类型。对于 `stringWithFormat:` 方法而言，这些不确定因素只能通过已确定的参数 `format` 来确定。因此这个方法需要通过 `format` 字符串中的`%@`、`%d` 等字符串来确定第 2 个可变参数的数目和类型。如果可变参数的类型和数目与 `format` 参数中的格式字符串不相匹配，`stringWithFormat:` 方法将无法正常工作。这样的例子还有 Objective-C 程序中经常用到的  `NSLog` 函数。

### 3. 属性

属性用于封装对成员变量的访问，是面向对象语言中的一个重要特性，Objective-C 也不例外。也许你已经在 C++ 和 Java 中熟悉了属性的概念，那么我们不再对此进行过多强调。

在将实例变量声明为属性的过程中，首先需要声明实例变量，然后使用`@property` 关键字。例如前面的代码中，首先用：

```
NSString *name;
```

声明了实例变量 `name`，然后用`@property` 将 `name` 声明为属性：

```
@property(nonatomic,retain) NSString *name;
```

---

**提示：** 实例变量的声明不是必须的。如果省略实例变量声明的话，编译器会自动提供一个与属性名同名的实例变量。

---

然后在.m 文件的 `implementation` 语句后使用`@synthesize<属性名>;` 语句。

当你这样做完之后，编译器会自动声明变量 `name` 的 `get`、`set` 方法，`get` 方法就是变量名，而 `set` 方法是单词 `set` 加上首字母被大写的变量名，例如：

```
- (NSString*) name;
- (void)setName: (NSString)newValue;
```

虽然这些声明在源代码中不可见，但确实是存在的，由编译器在编译时产生。

在其中比较复杂的部分是 `nonatomic` 和 `retain` 关键字的真实含义。在声明属性时，我们可以在`@property` 后面的圆括号中使用多个关键字修饰属性，以指定属性的可访问性、线程管理、内存管理等。由编译器根据这些修饰词产生不同的 `get`、`set` 方法。

`nonatomic` 关键字的意思是非原子的，意指对属性进行存取操作时是线程不安全的，如果在多线程环境下，该属性很可能是不同步的，一个线程读取属性值时，另一个属性却修改了属性值，这样两个线程对同一个属性进行操作的情况下，属性的值是不一致的。我们在属性中使用 `nonatomic` 的原因是，该属性不会在多线程环境下使用，使用非原子特性能得到较好的性能。而在 iOS 编程中，性能始终是程序员首先要考虑的问题。

retain 关键字是我们用得最多的属性修饰符之一，它和属性的内存管理有关。这样在对这个属性进行赋值操作时，在编译器自动生成（如果你使用了@synthesize 关键字的话）的 Set 方法代码中，会对实例变量进行 retain 操作。对于 Objective-C 对象类型的实例变量而言，使用 retain 操作使得属性在赋值后一直到对象被销毁之前始终可用。如果实例变量或属性并不是 Objective-C 对象类型，而是一个简单类型，如 BOOL、int、id、float，则用 assign 关键字替换 retain 关键字。这样，属性在赋值时不会被持有，这样导致的直接后果是：刚对一个属性赋值后，再访问这个属性，这个属性就变成空了。关于 Objective-C 的内存管理，更多内容会在后面介绍。

属性还可以用 readonly、readwrite 进行修饰，分别用于创建只读的属性和可读可写属性。

属性声明之后，接下来就是在实现文件 (.m 文件) 中实现属性的 get、set 方法。这些方法你可以选择自己去实现，也可以让 Objective-C 替你实现，只需要使用@synthesize 或者@dynamic 编译器指令。

也许你已经习惯书写传统 C++ 和 Java 的 get、set 方法，虽然实际上那是一件相当枯燥的事情。在 Objective-C 中，你完全不需要这样做，除非你真的需要。在实现的.m 文件中，使用@synthesize 关键字，可以自动产生属性的 get、set 方法代码。而这一切不是在源代码中进行的，而是编译器在编译时产生的，因此你不会在源代码中看到自动产生的代码。@dynamic 指令则是在运行时才产生这些代码。不管使用@synthesize 还是@dynamic，这些自动产生的代码均不可见，但它们是存在的。

### 3.2.2 消息机制

消息是 Objective-C 区别于其他语言的最大的地方，它其实是 Objective-C 特有的方法调用语法。同 C++ 和 Java 的方法调用一样：

- 1) 消息能够接受参数；
- 2) 消息可以嵌套调用；
- 3) 消息既可以发送 Objective-C 对象，也可以发送给类。

#### 1. 消息的参数

一个没有参数的消息是这样的：

```
[object methodWithoutParameter];
```

如果要调用的方法带有参数，则这个消息是这样的：

```
[object methodWithOneParameter: value];
```

注意，当方法没有参数时，其方法名后没有冒号。当方法有多个参数时，其方法名如我们前面所述，会有多个带冒号的参数：

```
[object methodWithFirstParameter:value1 withSecondParameter:value2];
```

#### 2. 消息的嵌套

当要进行一个嵌套的方法调用时，会使用嵌套消息：

```
[textView setTextColor:[UIColor whiteColor]];
```

内层的消息[UIColor whiteColor]首先调用并返回一个 UIColor，然后将这个临时的 UIColor 对象作为外层消息的参数传入。

即第 1 个消息的输出作为第 2 个消息的输入。

### 3. 接收者

如前面所举的例子中，消息的第一个元素 object 和 textView 都是消息的接收者，即我们要将消息发给的对象。

除了接收者可以是对象以外，接收者还可以是一个类。因为在方法的种类中，不仅仅有实例方法，还有类方法。使用类作为接收者，就能调用类方法：

```
NSMutableString *mstring=[NSMutableString stringWithString:@""];
```

## 3.2.3 Objective-C 的内存管理

内存管理是资源管理的重要部分，在 iPhone 中尤其如此（因为其内存有限）。相对于 Java 语言的垃圾回收机制而言，Cocoa 的内存管理相当粗糙。这意味着对 Objective-C 程序员而言，需要做更多的工作，即使是相当有经验的程序员，在这方面也很难不犯错，尤其是不使用 ARC 的情况下。

本书不讲述 ARC（即 iOS SDK 5.0 以后的自动引用计数），因为在 iOS 企业应用中会大量使用第三方框架，这些框架大部分不使用 ARC，这给我们在项目中应用 ARC 带来许多困难——要么你要单独设置每个.m 文件的 ARC 选项，要么你得放弃使用该框架，这在很多时候给我们带来困惑。因此，本书不推荐使用 ARC。

### 1. 对象的生命周期

每个对象都有生命周期，程序员要知道一个对象的生命周期是什么，就需要仔细回忆一下这个对象是什么时候创建的。每当对象创建出来，它的生命就已经开始了，一直到操作系统释放了该对象，对象的生命才结束。但操作系统怎么知道一个对象应该释放了呢？Objective-C 采用了两种内存管理模式：基于计数器的内存管理和基于自动释放池的内存管理。这两种内存管理模式会采用不同的方式来通知操作系统，何时应该释放一个内存对象。此外，Objective-C 2.0 以后也支持基于垃圾回收的内存管理，但垃圾回收只限于 Mac，而无法在 iOS 中使用。

基于计数器的内存管理和基于自动释放池的内存管理我们在下面介绍。而基于垃圾回收的内存管理则不属于本书讨论的范围。

### 2. 基于计数器的内存管理

在这种模式下，iOS 对内存对象的生命周期管理是通过引用计数来进行的。每个对象都有一个引用计数器，它记录了对象被使用的情况。如果计数器的值为 0，表示该对象不再被使用，对象的 dealloc 方法被调用，dealloc 方法是对对象的销毁方法，于是对象被销毁。

要知道一个对象的计数器值是多少，可以发送 retainCount 方法：

```
NSLog(@"%@", [object retainCount]);
```

当使用 alloc、copy、new 三种方法之中的任一种方法创建对象时，对象计数器会被自动设置为 1。此外，如果向对象发送 retain 消息，对象计数器会自动加 1。而向对象发送 release 消息，对象计数器会自动减 1。

因此，在使用计数器情况下的内存管理应当遵循以下原则：

- alloc、copy 或者 new 总是和 release 配套使用。如果你通过 alloc、copy 或者 new 方法创建了对象，那么必然跟随着一个 release。如果这个对象不是使用这 3 种方法之一创建的，则不需要一个配套的 release。
- retain 总是和 release 配套使用。如果对象创建后发送了 1 个 retain 消息，则需要配套 1 个 release 消息。
- release 的使用时机。对于临时对象，如果使用过 alloc、copy、new 以及 retain 之后，当不再使用该临时对象时，就可以使用 release 消息。如果该对象不是临时对象，而是类的成员，则在类的 dealloc 方法中 release 对象。
- 对于不是使用 alloc、copy、new 方法创建的对象，如果该对象是一个临时对象，则不要对该对象进行 retain 操作，也不要对该对象进行 release；如果该对象是类的成员，则需要在获取该对象时 retain，在 dealloc 方法中 release。

### 3. 基于自动释放池的内存管理

如前面讲述，使用基于计数器的内存管理十分麻烦，程序员必须对自己创建的每一个对象是否使用了 alloc、copy、new、retain 方法了如指掌，同时依据一系列规则适时地并在恰当的地方使用 release 消息。而且有时候要明白一个对象什么时候不再使用并不是一件容易的事情。

如果你不想使用对象计数器进行内存管理，则自动释放池是你的另外一个选择。

Cocoa 引入了自动释放池（autorelease pool）的概念。自动释放池实际是一个对象集合或对象容器的概念。如果你把所有的对象在创建时，都放到这个池中，则自动释放池被销毁时，池中的对象都会接收到 release 消息。

要把对象加入到自动释放池中，只需在创建对象的同时调用 autorelease 方法。该方法由 NSObject 提供：

```
- (id) autorelease;
```

例如下面的代码，当对象 string 一创建就被放到了自动释放池里：

```
NSString *string=[[NSString alloc]initWithString:@"I am a string"] autorelease];
```

那么，这个自动释放池是什么时候创建的？我们并没有创建它，它是 Foundation 框架自动为我们创建的。当我们使用 Xcode 新建任何一个 iPhone 应用程序时，Foundation 框架创建的模板代码中总是有这样的代码，你可以在应用程序委托的实现文件中找到它：

```
NSAutoreleasePool *pool;
pool=[[NSAutoreleasePool alloc] init];
...
[pool release];
```

其实我们的 HelloWorld 程序中也有类似的代码。原来，应用程序一开始就创建了这个自

动释放池，并立即成为当前活动的池。我们使用对象的 autorelease 消息就是把对象放入这个池中。当程序退出时，最后的[pool release]或[pool drain]代码会向池中对象发送 release 消息。

这样，对象会从创建开始，一直存在到应用程序结束。这样无疑会造成一定的内存浪费，整个应用程序运行期间，随着对象的不断创建，内存只见增加不见减少，容易造成内存资源的紧张。很显然，并不适合内存使用率较高的应用程序。

本书的建议，程序员主要使用基于计数器的内存管理模式，自动释放池模式的使用则适可而止。

---

**提示：**[pool release]方法适用于 Mac OS 的所有版本，而[pool drain]方法只适用于 Mac OS 10.4 (Tiger) 以上版本。

---

### 3.2.4 类别和协议

#### 1. 类别

类别是另一种为现有类添加新行为的方法。不同于子类，类别实际上是 Objective-C 的一种动态行为，它利用了运行时分配机制。因此，类别甚至不需要拥有原类的源代码。此外，类别不能向现有的类中添加实例变量。

要声明一个类别，你需要使用类声明时使用的@interface 关键字，如：

```
@interface NSString (NumberConvenience)
- (NSNumber*) lengthAsNumber;
@end
```

注意，这很像是一个类的声明。首先它使用@interface 类名开始，以@end 结束。但实际上，NSString 的类名加上其后的圆括号，表明这是一个对现有类 NSString 的补充或扩充。圆括号说明扩充方式为通过类别扩充而非通过继承来扩充。圆括号中的单词是类别的名称，本例中的类别名称是 NumberConvenience。

另外，第 2 行的方法声明语句也很像是 NSString 的成员方法。但实际上 NSString 类中根本没有这个方法。这个方法是类别为 Cocoa 类 NSString 新增加的。

此外，由于类别不能扩充实例变量这一限制，类别的声明中不会出现用于声明实例变量的{}符号。

类别主要是定义对现有类的扩充方法，因此类别的实现中 (.m 文件) 就需要对这些新方法一一进行实现：

```
@implementation NSString (NumberConvenience)
- (NSNumber*) lengthAsNumber{
    ...
}
```

这个实现文件跟类的实现文件没有多大区别，除了圆括号中的类别名外。接下来，你可以像这样调用这个新方法 lengthAsNumber (假设 NumberConvenience 类别声明是定义在

NSStringCategory.h 头文件里):

```
#import "NSStringCategory.h"
...
NSNumber* number=[@“hello” lengthAsNumber];
NSLog(@“length by NSNumber: %d”, [number intValue]);
```

就好像 lengthAsNumber 本来就是 NSString 中已经存在的方法一样。

类别有许多好处，比如把类的实现分散在多个 implementation 文件里（如果使用类，你做不到这点，这就好像 C# 中的分部类），或者用于创建非正式协议（如下面有关协议的内容所述）。

但是，类别仍然有一些限制，在类别的使用中一定要注意以下两点：

- 类别仅仅是一堆方法的集合，你无法为类添加实例变量。
- 类别不能解决命名冲突，如果类别中的方法与类原有的方法重名，则类中的方法被覆盖。

## 2. 协议

Objective-C 的协议等同于 Java 中接口的概念。下面我们来讨论协议的声明：

```
@protocol NSCopying
-(id)copyWithZone: (NSZone*) zone;
...
@end
```

协议看起来就像类别的声明，还是一堆方法声明的集合。但 @protocol 关键字的出现说明了这是一份正式协议。

正式协议的意思是，每个采用（Java 中用实现 implements 这个词）这份协议的类必须实现这份协议中的所有方法。这种说法一直持续到 Objective-C 2.0。

从 Objective-C 2.0 开始，协议中的方法可以有选择地由类实现。对于需要实现的方法，使用 @required 关键字修饰，对于可选择性地实现的方法，使用 @optional 关键字修饰，比如：

```
@protocol TheProtocol
@required
-(void)firstMethod;
-(void)secondMethod;
@optional
-(void)thirdMethod;
@end
```

这个协议规定，第 1、2 个方法是必须实现的，而第 3 个方法可以实现，但不要求一定实现。如果一个类要采用（或实现）这个协议，则需要在 @interface 中这样声明：

```
@interface MyClass: NSObject<TheProtocol>
```

而 MyClass 类的实现中，则应当实现该协议规定的 2 个必选方法及 1 个可选的方法。

除了正式协议外，我们也可以采用非正式协议。非正式协议不需要采用 @protocol 关键字

声明，但需要创建一个类别，例如 MyCategory：

```
@interface NSObject (MyCategory)
- (void)doSomething;
...
@end
```

由于 NSObject 是所有 Cocoa 类的根类，这个类别实际上指明了任何类都可以实现这些方法。从而可以向任何对象发送这些消息，而不需要在类的声明中做任何特别的说明。

### 3.2.5 反射机制

同其他高级语言一样，Objective-C 也提供了运行时支持——即 Java 所谓的反射机制，这充分体现了 Objective-C 的动态性特征。尤其在 Mac OS X 10.5 以后，苹果对 Objective-C 运行时 API 进行了重要的升级，以提供对 64 位模式的支持。

Objective-C 运行时 API 包含了众多函数和结构体的定义，然而，我们不准备在此介绍庞大的 Objective-C 运行时库。如果要全面了解这些 API，请翻阅苹果运行时库参考。

通常，我们不需要直接调用 Objective-C 运行时 API，Cocoa 框架已经把它的许多有用函数进行了封装，从而使我们更容易调用。

#### 1. 获取类信息

首先，我们知道 NSArray 等集合对象中不限制所存储的对象类型，只要它是一个 NSObject 就行。但有时候，我们想知道我们刚刚放进去的一个对象到底是什么类型（程序员总是那么健忘），是一个字符串？还是别的什么？

我们可以发送 class 消息：

```
id class=[[array objectAtIndex:0] class];
```

id 类型（即对象）其实是一个 objc\_object 结构：

```
typedef struct objc_object {
    Class isa;
} *id;
```

NSObject 的 class 方法实际上是返回了这个结构中的 isa 成员。isa 指向了一个 Class 对象，因此除了发送 class 消息外，我们也可以使用 isa 成员来访问 Class 对象。Class 对象指向了该对象所属的类（即“类对象”，Objective-C 把类也看成是对象，以贯彻“万物皆对象”的原则）。

Class 对象实际上是一个 objc\_class 结构：

```
typedef struct objc_class *Class;
struct objc_class {
    Class isa;
    Class super_class;
    /* followed by runtime specific details... */
};
```

---

**注意：**这个结构很象是 `objc_object`，但多了一个指向父类的 `super_class` 成员。也就是说，对象跟类的区别仅在于，对象中不保存继承关系，而类（或“类对象”）保存了继承关系。因此我们是通过类而不是对象来追溯类的“父类”、“祖父类”等。

---

我们有时需要比较一个对象是否属于某个类，可以使用类似的代码：

```
if (obj->isa == [NSString class]){
    ...
}
```

而 `super_class` 成员指向了父类对象，通过它我们可以访问父类的信息。

在 Cocoa 中还有一个“元类”的概念，即“类对象的类”。因此完整的类信息称为“类对”，即由“类对象”和“元类”构成。

如果向一个对象发送 `class` 方法，我们可以得到“类对象”，而如果再向“类对象”发送 `class` 消息，则返回的就是“元类”（meta class）。每个类只能有一个元类，其包含了类方法列表。而类对象不同，它包含了对象的方法列表。

## 2. 选择器

选择器实际上是一个方法名称，用`@selector` 关键字来指定一个选择器，选择器用于查询对象的某个方法，例如：

```
@selector(setResponse:)
```

通过 `NSObject` 的 `responseToSelector:` 方法，我们可以动态地查询某个对象是否存在指定的方法。该方法需要指定一个选择器参数：

```
if ([object respondsToSelector:@selector(setResponse:)]) {
    ...
}
```

如果 `object` 对象能够响应 `setResponse:` 方法，则返回 YES，否则返回 NO。

如果确定某对象能响应指定方法，则可以通过 `performSelector:` 方法进行调用：

```
[anObject performSelector:@selector(method)];
```

如果该方法带有参数，则使用 `performSelector: withObject:` 方法传递参数：

```
[anObject performSelector:@selector(method) withObject:obj];
```

如果有 2 个参数，则使用 `performSelector: withObject: withObject:`。更多的参数或者有返回值，则使用 `NSInvocation`。

除了向对象发送 `performSelector:` 消息之外，Objective-C 还提供了 `objc_msgSend` 函数，你可以用它向任何对象发送一条消息：

```
objc_msgSend(anObject,@selector(method),obj);
```

要使用 `objc_msgSend` 函数，需要导入`<objc/message.h>`头文件，其完整定义如下：

```
id objc_msgSend(id theReceiver, SEL theSelector, ...)
```

函数的第 1 个参数指向消息的接收者（即该方法的对象），第 2 个参数是一个选择器（即方法），第 3 个参数是一个可变参数，是该方法的 1 个或多个参数，如果该方法没有参数，用一个 `nil` 代替。

方法的返回值通过函数的返回值返回。

### 3. 类的动态创建

要在代码中创建类，而不是通过静态的.h 和.m 文件定义类，可以使用 Objective C 运行时库 API（需要`#include <objc/runtime.h>`）：

```
Class newClz=objc_allocateClassPair([NSError class], "RuntimeErrorSubclass", 0);
class_addMethod(newClz, @selector(report), (IMP)ReportFunction, "v@:");
objc_registerClassPair(newClz);
```

首先，使用 `objc_allocateClassPair` 动态函数创建了一个类，并在参数中指明该类的父类和类名。用 `class_addMethod` 函数为该类增加了一个方法 `report`，这个方法是由函数 `ReportFunction` 实现的，由于该函数至少应包含两个参数 `self` 和 `_cmd`，因此定义了该方法有 3 个参数，类型分别为 `v`、`@`、`:`（一个返回值，`self`，`_cmd`）。

`v` 代表 `void`，指定了方法的返回值；`@` 表示了 `id` 类型（对象），指定了方法的固定参数 `self`；`:` 表示选择器类型（SEL），指定了固定参数 `_cmd`。因此函数 `ReportFunction` 应当实现为：

```
void ReportFunction(id self, SEL _cmd)
{
    // 实现代码
}
```

最后，新类被注册为类对（Class Pair）。

类对注册后，即可在代码中这样使用类 `newClz`：

```
id obj =[[newClz alloc] init];
[obj performSelector:@selector(report)];
[obj release];
```

### 4. 类的动态加载

Cocoa 的 Foundation 框架提供的 `NSClassFromString` 函数类似于 Java 的 `Class.forName()` 方法：

```
Class cls=NSClassFromString(@"MyClass");
```

返回对象为“类对象” Class。通过这种方法，我们从字符串构建类实例就不再是什么问题：

```
id obj=[[cls alloc]init]
```

或者

```
id obj=[[NSClassFromString(@"MyClass") alloc] init];
```

这使得我们可以在运行时而不是编译时加载类，同时，不需要#import “MyClass”。

## 5. 方法的动态调用

通过@selector 关键字我们已经可以在一定程度上实现方法的动态调用。然而更动态的方式是通过 Fundation 框架的 NSSelectorFromString 函数，它可以直接从字符串获得一个选择器：

```
SEL sel = NSSelectorFromString(@"doSomethingMethod:")
if([object respondsToSelector:sel]) {
    [object performSelector:sel withObject:color];
}
```

## 3.2.6 谓词

Cocoa 提供 NSPredicate 类，用于描述条件和计算对象是否匹配指定条件。类似于 SQL 语句，通过 NSPredicate，可以将条件的计算从代码中分离出来，从而在比较对象时避免使用硬编码。

### 1. 创建谓词

首先需要创建一个 NSPredicate 对象：

```
NSPredicate *predicate=[NSPredicate predicateWithFormat:@"name=='Herbie'"];
```

类方法 predicateWithFormat 常用于创建一个 NSPredicate 对象。仅需要提供一个代表计算条件的字符串。字符串 @"name=='Herbie'" 代表了需要进行计算的 C 条件表达式。事实上，除了==运算符外，谓词的条件表达式支持括号和 C 语言的比较运算符和逻辑运算符。

下面我们来看看谓词是如何进行计算的。

### 2. 谓词的计算

要对谓词进行计算，需要将谓词应用在某个对象身上。例如：

```
BOOL match=[predicate evaluateWithObject: car];
```

谓词 predicate 的 evaluateWithObject 方法实际上对 car 对象进行了计算，并返回 BOOL 值 YES 或 NO。前面的条件表达式 @"name=='Herbie'"，表明将用 car 的 name 属性与字符串 Herbie 进行比较，如果相等，则返回 YES，否则返回 NO。

谓词可以用任何对象作为参数进行计算，包括数组或集合。如果 persons 是一个包含了多个 person 实例的 NSArray 数组，那么我们可以用谓词计算或过滤出其中 gender（性别）属性为男性的对象：

```
NSPredicate* predicate=[NSPredicate predicateWithFormat: @"gender=='M'"];
NSArray *result=[persons filteredArrayUsingPredicate: predicate];
```

谓词将对 persons 中的所有对象进行条件计算，把 gender 等于 M 的对象加到结果集数组中返回。

### 3. 在条件下使用变量

有时候，在构建谓词时，使用变量组成条件表达式将更方便。这就类似于 JDBC 中的预编译语句，Where 条件子句中的表达式可以使用可变参数。只需在 SQL 语句中使用问号？来代表可变参数即可。这样的好处是显而易见的，我们就可以重用预编译语句，节省数据库编译上 SQL 语句的时间。我们只需在执行预编译语句时，提供不同的可变参数值，即可执行不同的 SQL 查询。

同样，我们也可以在条件表达式中使用格式化字符串或占位符，来作为条件表达式中的可变部分。例如：

```
NSPredicate* predicate=[NSPredicate predicateWithFormat:@"name==%@", @"Herbie"];
```

这和原来没有任何区别。占位符%@表示这里将被一个对象（id 类型）所替换。如果想在表达式左边使用变量，可以使用%K 作为占位符，例如：

```
predicate=[NSPredicate predicateWithFormat:@"%@=%@", @"name", @"Herbie"];
```

这与前面一条语句是等效的。

**提示：**%K 中的字母 K 代表的是 keypath（键路径）。在后面介绍。

除此之外，我们还可以使用 NSLog 中使用的各种占位符，如%d 等。当然，不使用占位符，还可以使用命名的键-值对（NSDictionary）作为条件变量。例如：

```
predicate=[NSPredicate predicateWithFormat:@"name==$NAME"];
```

这里\$NAME 就代表了一个变量（而不是对象），这个变量值未指定。如果要指定这个变量的值，需要在 NSDictionary 中加入一个 Key 为 Name 的对象，同时用这个 NSDictionary（Cocoa 将这个 NSDictionary 称为环境变量 SubstitutionVariances）创建新的 predicate：

```
NSPredicate* predicate=[NSPredicate predicateWithFormat: @"name=$NAME"];
predicate=[predicate predicateWithSubstitutionVariances: dic];
```

然后使用新的 predicate 去进行计算。也就是说，通过这种方式，我们提交一个 NSDictionary 作为谓词变量。

### 4. BETWEEN 和 IN

BETWEEN 运算符用于表示位于某个区间中的取值。例如：

```
predicate=[NSPredicate predicateWithFormat:@"age BETWEEN{16,50}"];
```

该条件表达式表示 age 属性值在 16 到 50 之间的对象。

IN 用于测试某个值处于一个集合范围内的情况。例如：

```
predicate=[NSPredicate predicateWithFormat:
    @"/name IN {'Herbie','Elvis','Phoenix'}"];
```

该条件表达式表示 name 属性等于这三者之一的对象。

## 5. 字符串匹配

谓词常用于判断字符串的匹配，除使用比较运算符外，谓词表达式还支持多种对字符串进行计算的运算符：

- **BEGINSWITH** 用于判断一个字符串的开头是否包含另一个字符串。
- **ENDSWITH** 用于判断一个字符串的结尾是否包含另一个字符串。
- **CONTAINS** 用于判断一个字符串中是否包含另一个字符串。
- **LIKE** 运算符用于进行字符串的匹配。如同 SQL 语法一样，谓词表达式中的 LIKE 运算符支持两种通配符：\*和?，\*号匹配任意个数的任意字符，?号匹配 1 个任意字符。

此外，[c]选项用于表示忽略大小写，[d]选项用于表示忽略发音符号,[cd]选项用于表示同时忽略大小写和发音符号。例如：

```
predicate=[NSPredicate predicateWithFormat: @"/name BEGINSWITH[c] 'HER'"];
```

---

**提示：**如果你想在谓词表达式中使用正则式，则需要使用 MATCHES 运算符。

---

## 6. KeyPath 和 SELF

Cocoa 中经常引用 KeyPath 的概念。键路径 KeyPath 实际上和文件系统中路径概念无关，KeyPath 是面向对象语言如 Java 中“.”语法的概念，比如：

```
indexPath.row
```

这就是一个 KeyPath。

用前面的例子，我们想匹配 persons 数组中名称长度超过 10 的对象，那么在谓词表达式中不能使用 name 作为主语了（假设我们把谓词表达式分为“主语+运算符+宾语”结构，位于运算符左边的操作数是主语，而位于运算符右边的操作数就是宾语）。因为 name（具体地说，指数组中某个对象的 name 属性）只是一个字符串，它的长度应该用 name.length 表示。如前面所说，name.length 就是一个 KeyPath，用它作为表达式的主语是恰当的。那么，这个谓词应该这样构建：

```
predicate=[NSPredicate predicateWithFormat:@"/name.length > 10"];
```

然后再对 persons 数组中的每个对象进行计算：

```
NSArray *result=[persons filteredArrayUsingPredicate: predicate];
```

其实 KeyPath “name.length” 还可以是 “SELF.name.length”，这二者是完全等效的。SELF 用来表示应用了谓词的对象。SELF 经常可以省略，但在一种情况下不能被省略。让我们来假设这样的情况：

```
predicate=[NSPredicate predicateWithFormat:@"SELF ==@'Herbie'];
BOOL match=[predicate evaluateWithObject:@"Rubby"];
```

先看第2句，我们把谓词应用到一个字符串对象@"Rubby"。那么第1句中的SELF就不能省略了。该谓词的KeyPath使用SELF，表示我们要用它（而不是它的属性）直接和另一个字符串@"Herbie"进行比较，这是可以理解的。在这种情况下，你不能省略SELF，否则谓词表达式就不完整了，因为它缺乏了“主语”这个关键的语法元素。

### 3.3 MVC 模式

MVC模型是应用程序设计者们普遍采用的一种设计模式，在第2章介绍Cocoa Touch框架时曾简单介绍了MVC。MVC模式把应用程序GUI代码根据功能拆分为不同的类或组件：

- “模型”：用于封装应用程序的数据；
- “视图”：负责显示和编辑数据；
- “控制器”：负责处理前两者之间的逻辑关系。

它们之间的逻辑关系参考第2章的图2-3。

Cocoa Touch本身也遵循MVC模型原则。在MVC模型下，3个层次都由截然不同的类来实现，编写任何类的代码都应当明显地归为其中一类，并且其大部分功能代码不应当属于另外两类。这种分工负责的方式使得程序易于设计、实现和维护。

一般情况下，我们在Interface Builder中创建视图组件（关于Interface Builder的使用，我们在第5章中进行介绍）。或者，使用Xcode通过代码的方式继承已有的视图和控件。

模型负责保存应用程序数据，通常我们使用Objective-C对象或者Core Data来构建模型组件。

控制器组件可以使用UIKit控件中的ViewController及其子类，或者是完全由程序员自己定制的类。

从第4章开始，我们将开始在iOS应用程序开发中逐步应用MVC模型的基本理论来构建应用程序框架。始终遵循MVC模型的基本理论，将有助于你创建出更加简洁、易于维护的代码。

### 3.4 KVO模型

KVO(key-value observer，“键-值”观察)模型是Cocoa绑定技术中常用的一种编程模型，它可以使一个对象在属性值发生变化时主动通知另一个对象并触发相应的方法。与NSNotification不同，KVO没有所谓的中心对象来为所有观察者提供变化通知。当“被观察者”对象状态发生变化时，通知被直接发送至“观察者”对象，如图3-1所示。

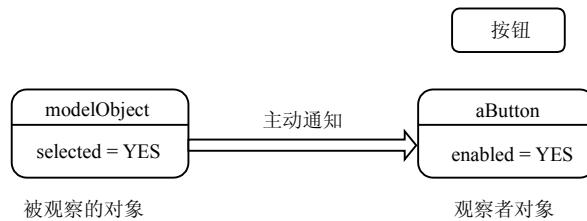


图 3-1 KVO 模型

在图 3-1 中，观察者是 aButton，被观察者是 modelObject。aButton 是一个按钮控件，它是一个 UI 对象，有一个 BOOL 型的 enabled 属性，表示按钮是否可被点击。modelObject 对象是一个模型对象，它没有可以呈现给用户的界面，同样，它有一个 BOOL 属性 selected。通过 KVO 模型，modelObject 的 selected 属性可以绑定到 UIButton 的 enabled 属性。即当 modelObject 的 selected 属性发生变化时（这是可以被编程的），KVO 会主动通知 aButton 这种改变，因此按钮的外观随之可发生相应的呈现。比如由不可点击的灰色改变为可点击的着色状态。

KVO 是一种很有用的绑定技术（Cocoa 还提供另外一种绑定技术：Dynamic binding）。而且它是由被观察的对象主动通知观察者的，并不需要经过一个统一的通知中心（如后面章节介绍的通知技术所述），它的执行效率和适用场景要更佳。

为了实现 KVO，你需要进行如下操作：

- 注册观察者。所谓观察者即对象状态变化时需要通知的对象。
- 接收变更通知。接收变更通知主要是让观察者实现指定方法，在指定方法中，你可以接收到对象状态变更的消息，并在方法中进行处理。
- 取消所注册的观察者。观察者处理完状态变更消息之后，需要取消原先的注册状态。

### 3.4.1 注册 KVO

对象要将自己注册为观察者，必须发送一个 addObserver:forKeyPath:options:context:消息至被观察对象：

```
[account addObserver:inspector
    forKeyPath:@"name"
    options:(NSKeyValueObservingOptionNew |
    NSKeyValueObservingOptionOld)
    context:NULL];
```

以上例子将 inspector 对象注册为 account 对象的观察者，并表明观察者将对名为“name”的属性变更感兴趣。

forKeyPath 参数“name”注明了需要观察的属性的关键路径 KeyPath。关键路径 KeyPath 实际是一个字符串，用于表示某个属性，你可以直接用属性名。但如果某属性是一个对象，则 KeyPath 可以用“.”语法的形式表示对象成员，如“account.name”。

options 参数注明了对该属性的何种状态感兴趣。`NSKeyValueObservingOptionNew` 表示属性在变更后的新值, `NSKeyValueObservingOptionOld` 表示属性未改变之前的值。以上例子中的 option 参数设置表明, 当 name 属性变更时, 会将这两个值以 `NSDictionary` 的方式 (即 change 参数) 提交给观察者, 观察者可以从 `NSDictionary` 中以键—值对的方式检索到这两个值。

`context` 参数用于传递一个对象, 该对象 (或指针) 会在属性变化时通过变更通知传递给观察者 (通过 `context` 参数)。

移除观察者的注册, 使用方法 `removeObserver forKeyPath:`:

```
[subject removeObserver:observer forKeyPath:@"name"];
```

### 3.4.2 接收变更通知

观察者要想收到对象的属性变更通知, 需要实现方法 `observeValueForKeyPath:ofObject:change:context:`, 并在其中进行通知的处理。例如:

```
- (void)observeValueForKeyPath:(NSString *)keyPath
    ofObject:(id)object
    change:(NSDictionary *)change
    context:(void *)context
{
    NSLog(@"%@", keyPath);
    if ([keyPath isEqualToString:@"name"]) {
        NSLog(@"name is changed:%@",
              [change objectForKey:NSKeyValueChangeNewKey]);
    } else
        [super observeValueForKeyPath:keyPath
            ofObject:object
            change:change
            context:context];
}
```

### 3.4.3 发送变更通知

`NSObject` 支持两种属性变更通知, 一种是自动变更通知, 一种是手动变更通知。一般情况下, 使用自动变更通知则更为简单, 因此我们主要介绍自动变更通知。

#### 1. 自动变更通知

要使用自动变更通知, 需要实现被观察者的 `automaticallyNotifiesObserversForKey` 方法, 在此方法中明确说明需要使用自动变更通知的属性。对于需要使用自动变更通知的属性, 返回 YES, 如下代码所示:

```
+ (BOOL) automaticallyNotifiesObserversForKey:(NSString*)key
{
    // 对于属性 name , 使用自动通知
```

```

if ([key isEqualToString:@"name"])
{
    return YES;
}
// 确保调用了父类的 automaticallyNotifiesObserversForKey 方法
return [super automaticallyNotifiesObserversForKey:key];
}

```

然后，在 name 属性发生变化的时候通知观察者，比如调用以下语句之一：

```

subject.name=newName;
[subject setValue:newName forKey:@"name"];
[subject setValue:newName forKeyPath:@"name"];

```

如果属性是集合类型，则可以使用方法 mutableSetValueForKey 来支持以下集合方法导致的自动变更通知：

- 添加： insertObject:InKey:或者 insertObjectAtIndex:
- 替换： replaceObject:InKey:或者 replaceObjectAtIndex:
- 删除： removeObjectFromKey:或者 removeObjectAtIndex:

## 2. 手动变更通知

对于手动变更通知，除了需要在 automaticallyNotifiesObserversForKey: 方法中将要使用的手动变更通知返回 NO 外，还需要 在改变值之前调用 willChangeValueForKey: 并在更改它之后调用 didChangeValueForKey:。

为了便于你理解 KVO 模型，我做了一个示例程序，放在光盘“source/第 3 章/TestKVO”文件夹，它使用了本节所介绍的知识点，可供参考和学习。

## 3.5 块编程

C 语言的运行时特性中包括了块，标准 C 工作组的 N1370: Apple's Extensions to C 中(其中也包括垃圾回收)对块进行了定义。作为 C 语言的扩展，Objective-C 在 OSX 10.6 及 iOS 4.0 以后支持块语法。块运行时也会被集成到 LLVM 的 compiler-rt 子项目存储库中。

### 3.5.1 块的特点

一些面向对象的动态语言如 ruby、groovy，都提供了对块的支持（在 groovy 中，块被称为闭包 “closure”）。块是用一对 {} 括号括起来的多个语句的集合。类似于函数，但不同于函数，可以把块作为表达式或变量的一部分，或者作为参数传递。在作为参数传递块时，代码被作为数据的一部分进行传递。

块具有以下特征：

- 同函数一样，有类型化参数列表。

- 有返回结果或者要申明返回类型。
- 能获取同一作用域（与块所在同一作用域）内的状态。
- 可以修改同一作用域的状态（变量）。
- 与同一范围内的其他块同享变量。
- 在作用域释放后能继续共享和改变同一范围内的变量。

除以上特点外，甚至可以复制块并传递到其他后续执行的线程，编译器和运行时负责把所有块引用的变量保护在所有块的拷贝的生命周期内。当然，这已经超出了本章的范围，可以参考苹果官方文档来了解这些内容。

### 3.5.2 Objective-C 中的块

对于 C 和 C++，块是变量，但对于 Objective-C，块仍然是对象。下面简单介绍 Objective-C 中的块。

#### 1. 块变量声明

用`^`操作符声明一个块变量的开始，分号表示块结束，如下代码所示：

```
int multiplier = 7;
int (^myBlock)(int) = ^(int num) {
    return num * multiplier;
};
```

块语法比较奇怪，块变量声明的解释如图 3-2 所示。

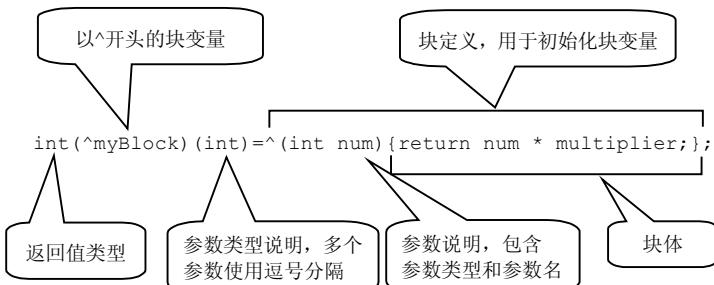


图 3-2 块变量声明的解释

块变量的声明语句从前至后分为了几部分：

- 返回值类型，如 `int`、`double`，如果未显式地声明块的返回值类型，可能会自动从块代码中推断返回类型（通过 `return` 语句）。
- 块变量名用括号括住，块变量名前加`^`符号。
- 参数类型用括号括住，多个参数以逗号分隔，如果参数列表为 `void`，而且返回类型依靠推断，可以省略参数列表的 `void`。
- 等号，将后面的块赋值给前面的块变量（即 `myBlock`）。

- 以^开头并以;结束的块定义。

块定义中又分为以下两个部分（除去开头的^和结尾的;外）：

- 参数列表，同函数的参数列表。

- 块体，同函数体。

值得注意的是，块可以使用同一作用域内定义的变量，而函数不行。

一旦声明了块，你可以像使用函数一样调用它：

```
int multiplier = 7;
int (^myBlock)(int) = ^(int num) {
    return num * multiplier;
};
printf (" %d", myBlock(3));
```

## 2. 行内块

有时候，你不准备重复使用某个块，因此你不必为它想一个名称。那你可以使用行内块而不用声明为块变量。以下代码来自苹果文档：

```
// qsort_b 类似标准的 qsort_r 函数，但它最后一个参数是一个块。
char *myCharacters[3] = { "TomJohn", "George", "Charles Condomine" };
qsort_b(myCharacters, 3, sizeof(char *), ^(const void *l, const void *r) {
    char *left = * (char **)l;
    char *right = * (char **)r;
    return strncmp(left, right, 1);
});
// myCharacters 现在是 { "Charles Condomine", "George", TomJohn" }
```

在 qsort\_b 方法调用中，第 4 个参数就是一个匿名的块（行内块）。匿名块跟块变量不同，它没有变量名，因此你无法重用匿名块。下次调用这个块时，必须把整个块定义的代码再复制一遍。

## 3. \_\_block 关键字

块允许访问本地变量。这很重要。它使得我们在线程间共享变量变得简单，而且，你可以规定一个本地变量是否可以写，这可通过使用\_\_block 关键字，这是一种类类似 register、auto 和 static 存储类型修饰符。

用\_\_block 修饰的变量，可以在所有同一作用域内的块，以及块复制之间共享数据。在指定作用域内的多个块能同时使用共享变量。

如同块，\_\_block 变量也使用栈存储。如果使用 block\_copy 拷贝块（或者向块发送 copy 消息），变量被拷贝到堆里。而且，\_\_block 变量的地址随后就会改变。

\_\_block 变量有两个限制：不能是可变长度的数组，也不能是包含 C99 可变长度数组的结构体。

下面显示了\_\_block 变量的使用：

```
__block int x = 123; // x 是块可写的
void (^printXAndY)(int) = ^(int y) {
    x = x + y;
    printf("%d %d\n", x, y);
};
printXAndY(456); // 打印出: 579 456
// x 现在的值是: 579
```

下面显示了在块中使用多种类型的变量：

```
extern NSInteger CounterGlobal;
static NSInteger CounterStatic;
{
    NSInteger localCounter = 42;
    __block char localCharacter;
    void (^aBlock)(void) = ^(void) {
        ++CounterGlobal;
        ++CounterStatic;
        CounterGlobal = localCounter;
        localCharacter = 'a';
    };
    ++localCounter;
    localCharacter = 'b';
    aBlock();
}
```

## 3.6 可变参数

我们知道，C 和 C++语言支持可变参数的函数，例如我们常用的 NSLog 和 printf 函数。Objective-C 作为 C 语言的超集，当然毫无例外地也支持可变参数。迄今为止，我们至少用过了一种使用可变参数的方法，即 NSString 的 stringWithFormat:方法。

C 语言通过 stdarg.h 库支持可变参数，Objective-C 也不例外。在 C 语言中，如果你要使用可变参数，必须包含头文件 stdarg.h，但在 Cocoa 中却不必，因为苹果已经在 NSObjC Runtime.h 中包含了 stdarg.h。

stdarg.h 的定义如下：

```
typedef __void * va_list;
#define va_start(ap, param) __builtin_va_start(ap, param)
#define va_end(ap)          __builtin_va_end(ap)
#define va_arg(ap, type)    __builtin_va_arg(ap, type)
```

首先定义了一个 va\_list 类型，其实就是一个 void\*，即可以指向任何类型的指针。你可以把它看成是 char\*，因为 char\*实际上也可以指向任何内存单元的地址。

然后是 3 个预定义宏，va\_start、va\_end 和 va\_arg。可以看出 stdarg.h 完全是以“预定义宏”

这种“古老”的方式来支持可变参数的。

接下来我们看一个例子，该方法使用了一个可变参数，并将这些可变参数进行了累加，然后返回一个 NSNumber：

```
- (NSNumber *) addValues:(int) count, ... {
    va_list args;
    va_start(args, count);
    NSNumber *value;
    double retval;
    for( int i = 0; i < count; i++ ) {
        value = va_arg(args, NSNumber *);
        retval += [value doubleValue];
    }
    va_end(args);
    return [NSNumber numberWithDouble:retval];
}
```

**代码说明：**

第 1 行是方法定义，该定义应当加到头文件中。省略号...表明方法接收一系列数目不定的参数，在...前面至少需要指定一个任意类型的参数。有时我们必须知道参数的个数以防止出现无效的引用，但在某种情况下，参数个数是可以通过其他参数推断出来的（例如 NSLog 或 printf 函数可以通过计算%号的个数推断可变参数的个数），或者对于 NSMutableArray 来说，它总是以 nil 终止。

如果是最后一种方法，我们可以把方法重新定义为：

```
- (NSNumber *) addValues:(NSNumber *) firstNumber, ...
```

这样，我们就可以用以下调用方式代替“addValues:3,num1,num2,num3”：

```
addValues:num1,num2,num3;
```

这样，我们就可以省略第 1 个表示可变参数个数的 int 参数。

第 2 行中的 va\_list 是 void \*类型，因此它实际上是一个可变的对象数组。

第 3 行用 args 来存放可变参数列表，而 count 则表示函数最后一个参数（即第一个“固定参数”）。这将使编译器把 args 指向第 1 个参数后的位置（通过 count 地址加上 count 变量的长度）。

很奇怪吗？可变参数中第 1 个参数的位址为什么是“count 地址+count 的长度”？因为对于大多数 C 编译器，函数栈中参数的存放顺序是从右到左的，也就是说先放入可变参数的最后一个参数，再放可变参数的倒数第 2 个参数……，然后放可变参数的第一个参数，最后是固定参数 count。而与此同时，栈的方向是向下的，即先入栈的数据位于高地址，后入栈的数据则位于栈的起始地址。这样，实际上最后放入的固定参数 count 的地址变成了栈的起始地址。而紧随 count 之后的地址则是可变参数的第一个参数地址，即“count 地址+count 的长度”，因

此编译器要能找到第 1 个可变参数的地址，只要知道 1 个参数：count 就够了，由 count 取得函数栈的起始地址，加上 sizeof(count)，得到第 1 个可变参数的地址。va\_start 的第 1 个参数 args 是一个输出参数，经过 va\_start 调用之后，args 将等于 arg\_start 计算出来的第 1 个可变参数的地址。

第 6 行是一个 for 循环，因为我们无法通过 args 自身推断 args 的大小，因此必须显式地用 count 来指定 args 的大小。或者可以使用 nil 终止的列表来检索可变参数。

如果你使用 nil 终止的数组作为可变参数，则应该用下面一行来代替第 6~7 行：

```
while( value = va_arg( args, NSNumber * ) )
```

第 7 行将 args 中的下一个参数放入 value，并显式地转为 NSNumber\*（如果不知道类型，可以用 id）。

第 10 行表明，一旦使用完 args 列表，就关闭它。

---

**提示：**如果你使用 va\_arg(args, double)（或者 float 等其他原始类型），那么当你试图传递一系列整数作为参数时（例如：addValues:4,4,3,2,1），可能会出现一些古怪的结果。而如果你显式地将这些参数说明为 double（例如，double num1,double num2,double num3,double num4）则不会有什么问题。

这是因为，如果编译器看到一个方法有一个 double 参数但你却传递了一个整数给这个方法时，它会进行类型转换。但如果方法使用了可变参数，编译器无法知道参数所使用的类型，因此编译器只会简单地把参数作为整型处理。

---

## 3.7 本章小结

Objective-C 是 C 语言的扩展和超集。本章重点从 C 语言特性和面向对象特性两个方面对 Objective-C 的语言特性进行论述，包括基本语法（数据类型、常量 / 变量、分支与循环）、运行时特性（反射支持）和一些特有属性（NS 类、类别和协议、消息、KVO 和块）。通过这些介绍，我们对 Objective-C 的一些重要特性有了最基本的了解，从而为后续的学习打下坚实的基础。接下来，本书将陆续介绍 iOS SDK 中的一些重要框架，如 UIKit、Core Animation、Quartz Core，以及其他一些第三方的开源框架。



如何结合iOS操作系统、iOS SDK，以及诸多相关开源框架的特点，开发出满足企业需求的iOS应用，这是每个企业级应用开发人员需要考虑的问题，本书给出了详细解决方案。本书不仅介绍了iOS开发中的一些基础的、具有共性的技术点，还重点讲解了企业级iOS开发中特有的难点，如Objective-C内存管理、iOS的多任务、企业APN、多线程和GCD等。此外，还讲解了在云计算环境下，如何实现企业级iOS应用的安全和网络等方面的需求。

特别地，本书对iOS的多任务、多线程概念进行了详细的解读，给出了全新的编程模型范例。本书的最后是两个综合案例：“企业APN”和“网络应用实战”，对书中的理论给出了实战解析。随书附赠光盘包含全部案例代码。

### 本书主要内容：

- 介绍iOS企业开发中的两个典型特征：企业IDP和无线部署。
- Objective-C语言中的新特性，如：块、可变参数、运行时、并行编程等。
- 全面介绍了Xcode IDE和Interface Builder的使用、快捷键及连接技巧。
- 如何使用ASIHTTPRequest简化网络编程。
- 如何使用PLDatabase访问数据库。
- 详细讲解对Cocoa的两种进程唤醒技术：本地通知和远程通知。
- 对UIKit进行扩展：自定义组件和静态库。
- 介绍了SDK的Security框架及两个安全算法库：OpenSSL和CommonCrypto。
- 如何使用图表框架CorePlot在应用程序中绘制图表。
- 介绍Cocoa touch新增的多点触摸和手势支持。
- 如何利用iPhone的多语言支持实现应用程序的国际化。
- 介绍iOS对iPhone特有硬件特性的支持，如通讯簿、相机、加速计和GPS。



附光盘

客服热线：(010) 88378991, 88361066  
购书热线：(010) 68326294, 88379649, 68995259  
投稿热线：(010) 88379604

读者信箱：[hzjsj@hzbook.com](mailto:hzjsj@hzbook.com)  
华章网站：[www.hzbook.com](http://www.hzbook.com)  
网上购书：[www.china-pub.com](http://www.china-pub.com)

上架指导：程序设计/移动开发

ISBN 978-7-111-40459-0



9 787111 404590 >

定价：69.00元 (附光盘)