HZ BOOKS
PEARSON

开发人员专业技术丛书

最新SDK快速上手iPhone应用开发

iPhone SDK 3 Visual QuickStart Guide

iPhone SDK 3 开发快速上手

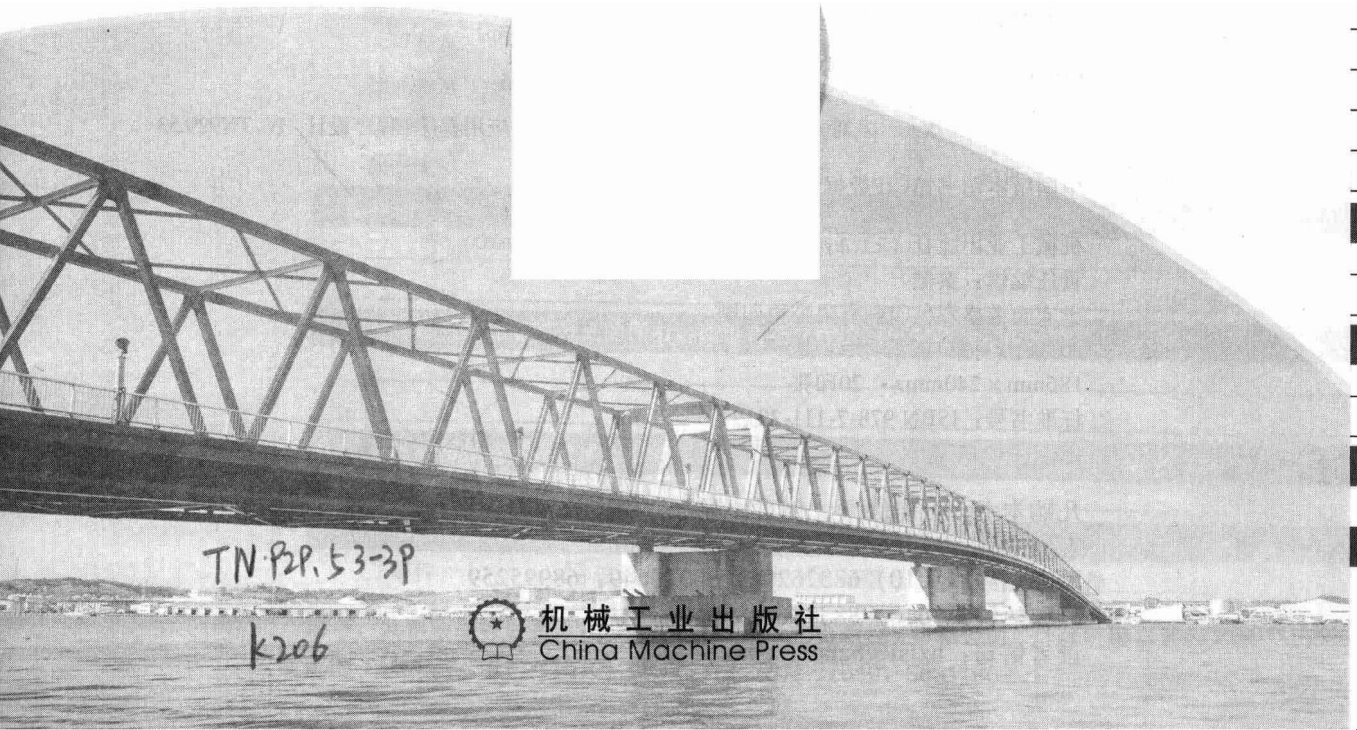
（美）Duncan Campbell 著

刘红伟 等译

机械工业出版社
China Machine Press

iPhone SDK 3 Visual QuickStart Guide

iPhone SDK 3 开发快速上手



机械工业出版社
China Machine Press

本书基于iPhone SDK 3，介绍用来创建iPhone应用程序的工具。从最常用的任务和UI元素开始，讲解如何使用标签页和表格、文件和网络、单多触点显示，以及内建的GPS硬件等。其中着重介绍开发iPhone应用程序的一些常见技术。

无论是iPhone开发新手还是老手，本书都将提供有益的帮助。

本书版权登记号：图字：01-2009-6673

图书在版编目（CIP）数据

iPhone SDK 3开发快速上手/（美）坎贝尔（Campbell, D.）著；刘红伟等译. —北京：机械工业出版社，2010.1

（开发人员专业技术丛书）

书名原文：iPhone SDK 3 Visual QuickStart Guide

ISBN 978-7-111-29191-6

Ⅰ.i… Ⅱ.①坎… ②刘… Ⅲ.移动通信—携带电话机—应用程序—程序设计 Ⅳ.TN929.53

中国版本图书馆CIP数据核字（2009）第222144号

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：秦健

北京市荣盛彩色印刷有限公司印刷

2010年1月第1版第1次印刷

186mm×240mm · 20印张

标准书号：ISBN 978-7-111-29191-6

定价：49.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991；88361066

购书热线：（010）68326294；88379649；68995259

投稿热线：（010）88379604

读者信箱：hzjsj@hzbook.com

译 者 序

iPhone不仅是一部成功行销全球的手机，而且已经成为时下最流行的移动开发平台之一。这一结果与Apple公司的战略和推动是分不开的。Apple不仅仅把iPhone当做一款产品，而且为它打造了强大的软件开发工具包iPhone SDK，从而为开发人员提供了有力的支持。更令人惊叹的是，Apple以其Apple Store的模式，成功地打通了开发、应用、通信等产业链。这一模式带来的成效是令人瞩目的，几乎给移动开发、通信等相关行业带来了颠覆性的变化。

对开发者来说，这种成功的模式带来了更大、更灵活的商业空间。你完全可以把自己的Apple应用变成钱，只需开发出自己的应用，放到Apple Store上去销售就行了。而要开发自己的应用，iPhone SDK则是不必可少的工具。它是Apple为iPhone应用开发量身打造的开发工具包，包括Xcode、iPhone模拟器、Interface Builder、文档，以及开发iPhone应用程序时所需的其他众多工具。

本书基于iPhone SDK 3来讲解，为iPhone应用开发初学者量身打造，是为初学者准备的快车道。充足实用的案例、按部就班的图释，可以帮助你快速掌握iPhone应用开发的必备技能。

本书由刘红伟、关志兴、王建勇、毛立涛、闫柳青、姜巧生、沈海峰等翻译。读者在阅读学习过程中，有任何疑问，可通过liuhongwei198004@sina.com与译者交流。

前　言

欢迎来到令人兴奋、吃惊，神秘的iPhone开发世界。

它并非真的神秘，但是，如果你是从非Macintosh的开发环境中转到iPhone的话，事情可能看上去很奇怪，甚至可能令你吃惊，但我希望它会令你兴奋。特别是，Xcode和Interface Builder，它们与很多其他的集成开发环境（integrated development environment，IDE）不同，Objective-C有着奇怪的语法，并且Cocoa框架规模庞大。我希望本书能够对你的学习过程有帮助，并且很快你将会发现，事情并非都是与你已经知道的那些截然不同，它们只是以一种不同但可能更好的方式去实现。

本书主要面向那些iPhone开发新手，但是，开始之前你应该有一些基于C语言的知识，并且熟悉面向对象（object-oriented，OO）的概念。要全面介绍iPhone SDK（software development kit），所需的篇幅可能是本书的数倍，因此，我着重介绍在开发你自己的iPhone应用程序的时候应该知道的一些较常见的和有趣的主题。

如何使用本书

我发现自己通过学习例子总是能够学习得更好，因此，在本书中展示概念的时候，我创建了独立的应用程序。这么做的目的是给你足够的信息来帮助你开始编程（并构建一些有用的应用），然后，我为你指出文档中相关的位置以便获取更多的信息。你应该能够直接进入某一章并开始编程，而不需要阅读前面的各章。

本书有很多图片，帮助你了解在构建自己的iPhone应用程序的时候会在计算机屏幕上看到什么。大多数例子的界面是直接在代码中创建的，而不是使用Interface Builder创建的。我觉得让你一开始就了解到背后究竟发生了什么是很重要的，这样一来，当事情不像预料的那样，你可以很容易搞清楚要到哪里去查找问题。

遗憾的是，并不是我编写的所有内容都放入了本书之中，但这些内容都没有丢掉。Peachpit出版社的热心人把这些额外的章节放到了他们的Web站点上，供读者免费下载：

www.peachpit.com/iphonesdkvqs

在那里，你会找到如下一些内容：使用iPhone的多媒体功能来播放和录制音频，使用iPhone相机来拍照和拍视频，使用你自己的应用程序访问iPod库、查询Address Book、发送E-mail等。

本书中所有示例的源代码都可以通过访问我的Web站点来下载：

http://objective-d.com/iphonebook/

好了，让我们启程吧！

目　录

第1章

Objective-C和Cocoa

Objective-C是iPhone开发最常用的语言。它是ANSI-C的一个超集，拥有Smalltalk式的语法。如果你曾经使用任何现代语言（例如C++、Java或者甚至PHP）编写过程序，应该能够较快地熟悉Objective-C。

Cocoa是Apple为OS X和iPhone开发所提供的框架的总称。在本书中，Cocoa表示特定于iPhone的API。

在本章中，我们将简单地概述Objective-C代码的结构是什么样的，以及如何构建自己的类。然后，我们将学习如何管理内存，并且学习一些较为常用的Cocoa类。最后，我们将学习在整个Cocoa框架中使用的一些设计模式。

1.1 框架

iPhone OS为iPhone开发提供了一组框架。像UIKit、Core Location、Map Kit、Address Book和Media Player这样的框架，是用来帮助我们实现特定技术的类的一个集合。

为项目添加一个框架，就可以使用该框架中包含的类。Apple按照功能把这些框架组织为四组（参见表1-1）。

表1-1 iPhone OS框架的分组

框架分组	说 明
Cocoa Touch	该框架用来处理所有触摸和事件驱动编程，并可以访问系统范围的接口组件（如Address Book、浏览器、地图、消息和大多数的用户接口组件）
Media	该框架用来播放和录制音频和视频，并且提供对动画和2D及3D图形的支持
Core Services	该框架用来访问很多iPhone的低层级功能，例如文件、网络、定位服务、对应用内购买（in-app purchase）的支持，以及像网络可用性这样的配置信息
Core OS	该框架提供对iPhone的内存、文件系统、低层级网络以及硬件的访问

向项目中添加一个框架的方法如下：

1）在Groups & Files面板中，展开Targets部分，鼠标右键点击你的应用程序目标，并且选择Get Info。

2）确保选中General标签，点击Linked Libraries列表底部的Add (+)，然后从可用列表中添加框架（如图1-1所示）。

3）在类的头文件中，导入该框架。

程序清单1-1示例如何把CoreAudio框架的引用添加到一个类中。

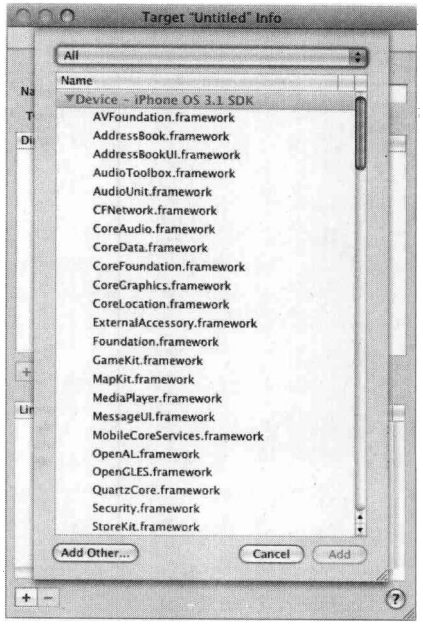


图1-1 为项目添加一个框架

程序清单1-1 在代码中引用一个框架

```
Code
//
//  UntitledViewController.h
//  Untitled
//

#import <UIKit/UIKit.h>
#import <CoreAudio/CoreAudioTypes.h>

@interface UntitledViewController : UIViewController
{
}

@end
```

1.2 类

和常规的C一样，Objective-C把类划分到两个文件中：头文件和实现文件。头（.h）文件是类的公共接口，它包含了属性、实例变量和可用方法的声明。图1-2展示一个接口文件的主要部分。

- #import指示符，和C语言的#include语句类似，它允许我们在源代码中包含头文件。然而，#import确保同一文件不会包含多次。
- @interface行声明类名及其超类，即该类所继承的父类。该类所实现的任何协议都附加在最后的尖括号之中（<和>）。
- 接下来，在花括号中（{和}），我们定义类所使用的任何实例变量。
- 最后，定义类的方法和属性声明，然后，用@end指示符结束接口文件。

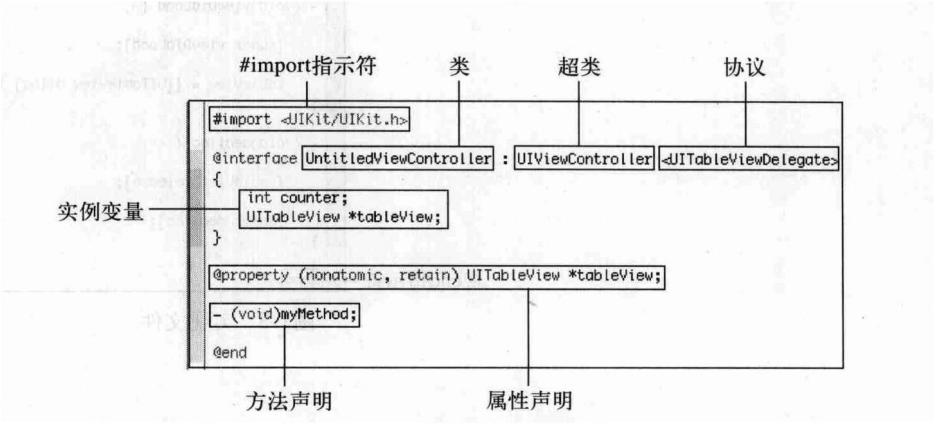


图1-2 接口文件

实现文件(.m)放置的是头文件中所定义的方法的实现代码。也可以在这里实现私有的方法，使用你的类的任何人不会看到这些方法。图1-3展示了实现文件的主要部分。

- 再次用到#import指示符，这次是导入接口声明。
- @implementation行是为类编写的代码部分的开始。
- 接下来使用@synthesize指示符来为类的属性生成setter和getter方法。注意，它们在同一行出现，要用逗点隔开。
- 最后，编写代码，实现接口文件中定义的方法，然后，用@end指示符结束实现。

1.2.1 方法

Objective-C中的方法在对象上执行一个动作。方法用方括号括起来：

```
[myObject foo];
```

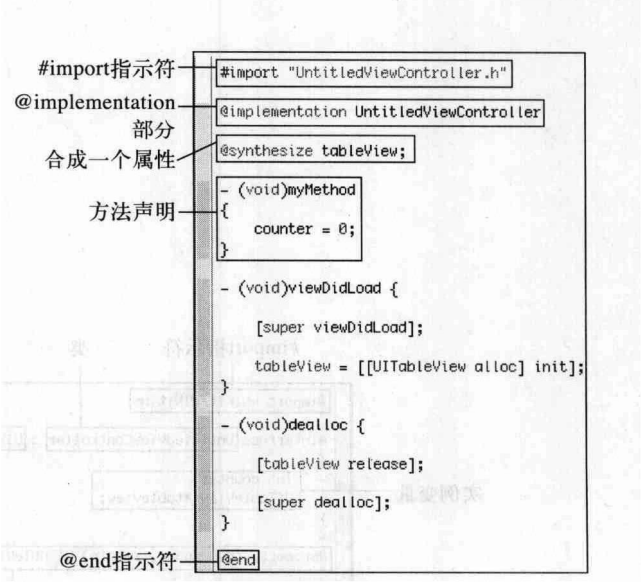


图1-3 实现文件

在这里，我们调用myObject对象上的一个名为foo的方法。调用一个方法的过程叫做发送消息（messaging），消息（message）就是方法的签名，其中包括所传递的任何参数。

Objective-C是一种冗长的语言，带有很长的、详细的方法名和参数名。方法名和参数名组合在一起，形成一个短语，说明了方法的动作。这里采用了骆驼命名法的一种变体，第一个单词通常是小写，后续的每个单词的首字母大写，并且单词之间没有空格：

```
[myObject performSomeAction];
```

这会调用myObject的performSomeAction方法。

当向方法中传递一个值的时候，如果类型重要的话，参数名通常也会说明数据类型：

```
[myObject saveInteger:10];
```

这将会调用myObject对象上的saveInteger方法，把值10作为第一个参数传递。

有多个参数的时候，指定每一个参数，并且在参数的帮助下形成描述方法的目的的短语。例如，创建一个分数并返回结果的C函数，可能如下所示：

```
fraction = MakeFraction(10,20);
```

用Objective-C的方法来实现，它可能如下所示：

```
fraction = [Fraction fractionWith
→Numerator:10 denominator:20];
```

这里，我们调用Fraction上的fractionWithNumerator:denominator:方法，传递两个

参数，并且把返回值存储到变量fraction中。

调用方法和定义方法的语法非常类似。例如，可以如下这样定义前面的方法：

```
-(double)fractionWithNumerator:
→(int)num denominator:(int)denom;
```

很多类提供所谓的类方法(class method)，我们可以直接在类自身上调用一个方法，而不用创建一个对象然后在对象上调用方法。

按照惯例，类方法（除了+new和+alloc）通常返回自动释放的对象（参见本章后面的1.3节“内存管理”）。

定义类方法的时候，在方法类型标识符前加一个加号（+）：

```
+ (MyClass *)classWithInteger:
→(int)iValue
```

程序清单1-2给出一些常用的类方法的例子。

编写方法的快捷方式：从方法签名中删去数据类型和参数名称，保留一个冒号（:）来表示一个参数。例如下面的方法：

```
-(NSString *)appendString:(NSString *)
→string1 toString:(NSString *)string2
```

可以简写为下面的形式：

```
appendString:toString:
```

程序清单1-2 一些常用的类方法

Code

```
- (void)viewDidLoad {

    NSString *myString = [NSString stringWithString:@"foobar"];
    NSMutableDictionary *dict = [NSMutableDictionary dictionaryWithObject:nil];
    NSMutableArray *arr = [NSMutableArray arrayWithObject:nil];

    UIButton *myButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    UIImage *img = [UIImage imageNamed:@"apple.png"];
    UIFont *font = [UIFont systemFontOfSize:14.0];
}
```

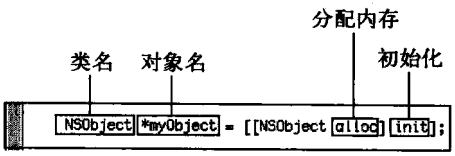


图1-4 创建对象

1.2.2 创建对象

一般来讲，在Objective-C中创建一个对象，我们需要按照下面步骤进行：

- 定义对象的类型，并且给它一个名字。
- 用alloc类方法分配内存。
- 用init方法初始化该对象。

例如：

```
NSObject *myObject;
myObject = [NSObject alloc];
[myObject init];
```

这通常写成下面的一条语句：

```
NSObject*myObject=[[NSObject alloc]
→init];
```

图1-4给出这条语句的分解。注意，两个示例中的方括号的数目是相同的。这种方法调用的嵌套在Objective-C中很常见，并且在贯穿本书的例子中都将见到。

很多类提供了额外的初始化方法，允许你在单独一次方法调用中执行多个步骤。例如，要创建一个NSString并为其赋值，可以使用下面的语句：

```
NSString *myString=[[NSString alloc]
→initWithString:@"some value"];
```

也可以使用类方法：

```
NSString *myString = [NSString
→stringWithString:@"some value"];
```

(NSString包含了很多这样的初始化方法，我们将在本章后面的1.4节“常用类”中介绍。)

1.2.3 属性

属性为我们提供一种方便的方法在对象上获取和设置实例变量，而不必定义或使用存取方法（accessor method，通常叫做getter和setter）。

- 例如，想要创建一个新的UIView对象可以像下面这样编写：

```
UIView *myView = [[UIView alloc]
→init];
```

- 然后，可以设置backgroundColor属性：

```
myView.backgroundColor = [UIColor
→redColor];
```

- 也可以获取同一个属性的值：

```
UIColor *bgColor =
→myView.backgroundColor;
```

正如前面提到的，使用关键字@property在类头(.h)文件中定义属性（见程序清单1-3）。

注意，将属性定义为只读的（readonly），以及设置存取方法的实现方式如下：直接分配的（assignment，默认）、持有的（retain），或者是作为用于分配的对象的一个副本（copy）。

在类实现(.m)文件中，使用@synthesize关键字将会自动产生getter和setter方法（见程序清单1-4）。

提示

- 要了解关于属性的详细信息，参考开发人员文档中的《The Objective-C 2.0 Programming Language Guide》的“Declared Properties”部分。

程序清单1-3 定义属性

Code

```
@interface UntitledViewController : UIViewController
{
    int counter;
    NSString *username;
    NSString *language;
    NSNumber *age;
}

@property int counter;
@property (copy, readwrite) NSString *username;
@property (readonly, assign) NSString *language;
@property (retain) NSNumber *age;
```

程序清单1-4 合成属性

Code

```
@implementation UntitledViewController

@synthesize counter;
@synthesize username,language;
@synthesize age;
```

1.3 内存管理

在本章前面的1.2.2小节“创建对象”中，我们创建了一个对象：

```
NSObject *myObject=[[NSObject alloc]
→init];
```

Objective-C采用一种叫做引用计数(reference counting)的过程来管理内存。当创建一个对象的时候（在这个例子中，通过调用alloc方法），它包含一个引用计数，也叫做持有计数（retain count），该计数为1。从现在开始，每次任何人引用该对象的时候（通过调用其retain方法），引用计数增加1。当使用完该对象，我们调用其release方法，引用计数就会减1。当引用计数达到0的时候，对象的内存就会从系统中释放。

持有之后的对象如果不释放的话，将会导致内存泄漏，因此，确保在用完对象的时候总是释放它们，这一点很重要。相反，我们需要知道何时应该持有其他人所创建的一个对象，如果我们还在使用某个对象的话，当然不希望释放它；并且，我们也不希望释放一个自己没有持有的对象。

在使用对象的时候，一个有用的习惯是，尽可能早地释放它们。

考虑如下的代码：

```
UILabel *myLabel = [[UILabel alloc]
→init];
myLabel.text = @"some text";
[myView addSubview:myLabel];
```

```
[myLabel release];
```

首先创建一个标签，这会将其持有计数设置为1。在设置文本之后，我们把这个标签添加到一个视图中。这会把持有计数增加到2（当标签添加为一个子视图的时候，该视图在该标签上调用retain）。我们现在不再需要标签（视图现在拥有它），因此，在下一行代码中释放它。我们现在可以安全地忘掉管理标签的内存，我们已经平衡retain/release调用，并且视图将会自己释放其子视图（并且由此释放标签）。

只要用完对象就释放它（而不是等到在代码的最后才做），这是很好的方法，并且有助于减少发生内存泄漏的可能性。

自动释放池

为了便于控制，Objective-C提供了自动释放池。

考虑下面的例子：

```
-(NSString *)makeUserName
{
NSString *name = [[NSString alloc]
→initWithString:@"new name"];
return name;
}
```

在这里，我们创建一个新的字符串并且要从一个方法返回它。遗憾的是，使用这个方法的某人无法知道我们期望他们在将要返回的字符串上调用release，因此，将会造成内存泄漏。显然，我们无法在方法内调用release，因为这会把持有计数设置为0并且没有可返回的内容。