移动开发经典丛书

学习和使用Objective-C编程语言的便利实践参考

# Objective-C 开发范例代码大全

Objective-C Recipes: A Problem-Solution Approach

[美] Matthew Campbell 著
景 丽 译

Apress®

清华大学出版社

移动开发经典丛书

# Objective-C 开发范例代码大全

[美] Matthew Campbell 著

景 丽 译



清华大学出版社

北 京

Matthew Campbell
Objective-C Recipes: A Problem-Solution Approach
EISBN：978-1-4302-43717


**图书在版编目(CIP)数据**

Objective-C 开发范例代码大全/(美)坎贝尔(Campbell, M.) 著；景丽 译. —北京：清华大学出版社，2013.2
(移动开发经典丛书)
书名原文：Objective-C Recipes: A Problem-Solution Approach
ISBN 978-7-302-31364-9
Ⅰ. ①O… Ⅱ. ①坎… ②景… Ⅲ. ①C 语言—程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2013)第 013985 号

**责任编辑**：王　军　李维杰
**装帧设计**：牛静敏
**责任校对**：邱晓玉
**责任印制**：宋　林

**出版发行**：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
**印 装 者**：清华大学印刷厂
**经　　销**：全国新华书店
**开　　本**：185mm×260mm　　**印　　张**：21.75　　**字　　数**：529 千字
**版　　次**：2013 年 2 月第 1 版　　**印　　次**：2013 年 2 月第1次印刷
**印　　数**：1～4000
**定　　价**：49.80 元

产品编号：049604-01

# 作 者 简 介

Matthew Campbell 曾在 Mobile App Mastery Institute 与 iOS Code Camp 上培训过 800 多位 iOS 开发新手。他还开发了 Tasting Notes，这是一款面向葡萄酒爱好者的通用应用。Matthew 是 http://HowToMakeiPhoneApps.com 的首席博主，这是一个关于如何创建 iPhone 应用的博客。

# 技术审校者简介

Anselm Bradford 是新西兰奥克兰科技大学的数字媒体讲师，他在那里研究交互式媒体、Web 媒体与可视化通信。他已经是多本 iOS 图书的技术审校者，也是 *HTML5 Mastery* 的主要作者以及 *CSS3 Solutions* 的合著者。可以通过 Twitter 账号@anselmbradford 找到他，他偶尔也会在 AnselmBradford.com 上发表博文。

# 致　谢

如果说这样一本书是封面上作者一个人的功劳，那简直是太棒了。当然，这是不可能的，如果没有 Apress 编辑的支持，这本书是不可能面世的。

特别地，我要感谢 Louise Corrigan，他的批注遍布我们的共享文档，鼓励我完成每个章节的写作。我还要感谢本书的技术审校者 Anselm Bradford，他的帮助保证了书中代码的正确性。

我要感谢 Corbin Collins，他使我们一直走在正确的轨道上。如果没有他偶尔的提醒，我很容易就错过了截止期限，Corbin 帮了我的大忙。

最后，我要感谢 http://HowToMakeiPhoneApps.com 博客的所有读者以及 Mobile App Mastery Institute 的学生们。本书之所以能出版都是因为他们多年来慷慨的支持与关心。如果没有他们的反馈与校验，我是不可能完成全书写作的。

# 前　言

时至今日，学习编程其实是在学习如何塑造我们的世界。Objective-C 程序员正处在独一无二的位置之上，他们创建的应用会被全世界的人们在每天的生活中使用。

使用 Objective-C 是一件乐事，而其他编程语言有时会略显笨拙，Objective-C 会向你展现出它的能量与优雅。其他编程语言中的棘手难题在 Objective-C 中都变成了小菜一碟。

本书的核心是在语言的自然环境中探索 Objective-C。Objective-C 的代码是关于计算机科学的，能以优雅的方式解决问题。

# 目　录


第 1 章　应用开发 ……1
1.1　创建终端应用 ……1
1.2　输出到控制台 ……3
1.3　创建新的自定义类 ……5
1.4　编写属性访问器 ……7
1.5　使用@synthesize 编写属性访问器 ……10
1.6　向自定义类中添加类方法 ……12
1.7　向自定义类中添加实例方法 ……14
1.8　使用类别对类进行扩展 ……15
1.9　从终端创建基于窗口的 Mac 应用 ……17
1.10　向 Mac 应用添加用户控件 ……20
1.11　从 Xcode 创建基于窗口的 Mac 应用 ……23
1.12　从 Xcode 创建 iOS 应用 ……25
1.13　使用目标-动作向 iOS 应用添加用户控件 ……29
1.14　使用委托向 iOS 应用添加用户控件 ……33

第 2 章　使用字符串与数字 ……37
2.1　创建字符串对象 ……37
2.2　在 Mac 上从文件读取字符串 ……39
2.3　在 iOS 上从文件读取字符串 ……41
2.4　在 Mac 上将字符串写到文件中 ……43
2.5　在 iOS 上将字符串写到文件中 ……45
2.6　比较字符串 ……48
2.7　操纵字符串 ……50
2.8　搜索字符串 ……53
2.9　本地化字符串 ……54
2.10　将数字转换为字符串 ……56
2.11　将字符串转换为数字 ……58
2.12　格式化数字 ……59

第 3 章　使用对象集合 ……61
3.1　创建数组 ……62
3.2　引用数组中的对象 ……63
3.3　获取数组中元素的数量 ……65
3.4　遍历数组 ……66
3.5　排序数组 ……68
3.6　查询数组 ……72
3.7　操纵数组内容 ……75
3.8　将数组保存到文件系统中 ……78
3.9　从文件系统读取数组 ……80
3.10　创建字典 ……81
3.11　引用数组中的对象 ……83
3.12　获取字典中元素的数量 ……84
3.13　遍历字典 ……85
3.14　操纵字典内容 ……87
3.15　将字典保存到文件系统中 ……89
3.16　从文件系统读取字典 ……90
3.17　创建集合 ……92
3.18　获取集合中元素的数量 ……93
3.19　比较集合 ……94
3.20　遍历集合 ……96
3.21　操纵集合内容 ……97

第 4 章　文件系统 ……101
4.1　引用并使用文件管理器 ……101
4.2　获得指向 Mac 系统目录的引用 ……103
4.3　获得指向关键 iOS 目录的引用 ……105
4.4　获取文件属性 ……107

4.5 获得目录下的文件与子目录列表 ····109
4.6 管理目录 ····111
4.7 管理文件 ····114
4.8 查看文件状态 ····117
4.9 修改文件属性 ····119
4.10 使用 NSFileManager 委托 ····121
4.11 使用 NSData 处理数据 ····127
4.12 使用 NSCache 缓存内容 ····131

**第 5 章 使用日期、时间与定时器 ····137**
5.1 创建表示今天的日期对象 ····137
5.2 通过 Component 创建自定义日期 ····138
5.3 比较两个日期 ····140
5.4 将字符串转换为日期 ····143
5.5 格式化日期以便显示 ····144
5.6 加减日期 ····146
5.7 使用定时器调度并重复执行任务 ····147

**第 6 章 异步处理 ····151**
6.1 在新线程中执行处理 ····151
6.2 主线程与后台线程之间的通信 ····156
6.3 使用 NSLock 锁定线程 ····163
6.4 使用@synchronized 锁定线程 ····167
6.5 使用 Grand Central Dispatch(GCD) 进行异步处理 ····171
6.6 在 GCD 中使用顺序队列 ····177
6.7 使用 NSOperationQueue 实现异步处理 ····182

**第 7 章 使用 Web 服务 ····187**
7.1 下载文件 ····187
7.2 通过 XML 使用 Web 服务 ····189
7.3 通过 JSON 使用 Web 服务 ····195
7.4 异步地使用 Web 服务 ····198

**第 8 章 内存管理 ····201**
8.1 理解内存管理 ····201
8.2 创建不使用 ARC 的应用 ····203
8.3 使用引用计数管理内存 ····205
8.4 为自定义类添加内存管理 ····207
8.5 使用 autorelease 消息 ····210
8.6 为 Mac 应用启用垃圾收集 ····215

**第 9 章 使用对象图 ····217**
9.1 创建对象图 ····218
9.2 使用键-值编码 ····229
9.3 在对象图中使用键路径 ····236
9.4 使用键路径聚合信息 ····241
9.5 实现观察者模式 ····247
9.6 探查类与对象 ····252
9.7 归档对象图 ····257

**第 10 章 Core Data ····267**
10.1 向应用添加 Core Data 支持 ····267
10.2 添加实体描述 ····274
10.3 向应用添加托管对象 ····276
10.4 向 Core Data 添加托管对象 ····280
10.5 从数据存储中检索对象 ····285
10.6 将变更发回数据存储 ····290
10.7 使用 Core Data 管理一对一关联关系 ····296
10.8 使用 Core Data 管理一对多关联关系 ····304
10.9 管理数据存储的版本 ····315

**第 11 章 Objective-C：超越 Mac 与 iOS ····325**
11.1 在 Windows 上安装 GNUstep ····325
11.2 Windows 上的 Objective-C 程序 Hello World ····327
11.3 下载 Objective-J 以进行 Web 应用开发 ····330
11.4 编写 Objective-J 应用 Hello World ····331
11.5 向 Objective-J 应用添加按钮 ····336

# 第 1 章

# 应 用 开 发

本章将会介绍从命令行和 Xcode 创建 Objective-C 应用所需的一些基础知识，还将介绍如何编写命令行 Mac 桌面应用以及如何为 iPhone 和 iPad 创建 iOS 应用。

**本章内容：**

- 从命令行编译 Objective-C 程序
- 使用属性和方法编写自定义类
- 实现实例方法与类方法
- 使用类别扩展现有的类
- 编写与编译 Mac 命令行应用
- 使用 Xcode 创建 Mac 应用
- 使用 Xcode 创建 iOS 应用
- 使用委托与目标-动作模式向应用添加用户控件

**注意：**

本书的大部分内容假设你使用的是 Mac 并且安装有 Xcode 4.2，可以从 Mac App Store 获取 Xcode，网址为 www.apple.com/mac/app-store/。

## 1.1 创建终端应用

### 问题

你想要通过终端构建简单的 Objective-C 程序，Objective-C 程序并不依赖于 Xcode 携带的额外特性。程序会使用 Objective-C 向 Mac 的终端控制台窗口输出一条消息。

## 解决方案

使用喜欢的文本编辑器在主目录下创建文件，主目录位于/Users/[yourusername]/。可以使用终端的文本编辑器 vi，也可以使用 Mac 自带的 GUI 程序 TextEdit。如果使用 TextEdit，那么确保将文件保存为纯文本格式。

在文件中，需要添加 main 函数(与 C 语言中的 main 函数非常像)、导入 Foundation 框架、编写 Objective-C 代码以向控制台输出一条 Hello World 消息。

为了编译程序，需要使用名为 clang 的工具来创建可执行文件，接下来就可以通过终端屏幕来运行程序了。

## 说明

Objective-C 的启动代码总是位于名为 main 的函数中，main 函数接收一些参数并返回一个整数值。在第一行代码中导入 Foundation 框架，这是使用 Objective-C 对象所必需的一个框架。

在 main 函数中，需要创建自动释放池，Objective-C 会通过它来管理内存。接下来，你就可以使用 NSString 类来创建 Hello World 字符串并使用 NSLog 将该字符串写到控制台屏幕。

用于编译代码的终端命令叫做 clang，它会编译 Objective-C 程序。下面是在使用 clang 编译 Objective-C 程序时可能会用到的一些选项：

- –fobj：表示 Objective-C 是编程语言。
- –arc：指定自动引用计数。
- –framework：用于链接到 Foundation 框架。
- –o：指定将要创建的可执行文件的名称。

**注意：**

如果 Mac 运行的是 OSX 10.7 或更高版本，那么你就可以使用自动引用计数(Automatic Reference Counting，ARC)。ARC 是 OSX 10.7 中用于内存管理的新特性，你可以向编译程序的语句中添加-arc 来启用 ARC 功能。如果不确定使用的是哪个 OSX 版本，那么现在可以忽略掉-arc。请参见第 8 章以深入了解 ARC 与内存管理。

## 代码

代码如下所示：

```
#import <Foundation/Foundation.h>
int main (int argc, const char * argv[]){
        @autoreleasepool {
                NSString *helloString = @"Hello World";
                NSLog(@"%@", helloString);
        }
        return 0;
}
```

### 使用

打开终端并输入如下命令来编译代码(确保在编译前转到代码文件的存放位置):

```
clang -fobjc -framework Foundation main.m -o maccommandlineapp
```

对于该例来说，假设代码位于名为 main.m 的文件中，输出文件名为 maccommandlineapp。

按回车键编译代码。程序编译完毕后，输入 open maccommandlineapp 并按回车键来运行和测试程序。

这时会打开另一个窗口，输出如下所示:

```
Hello World
logout

[Process completed]
```

## 1.2 输出到控制台

### 问题

在测试代码时，你可能想要将值输出到控制台窗口中。对象与原生类型值都可以输出，但它们都需要特定的字符串格式化器并搭配 NSLog 才可以。

### 解决方案

将对象与原生类型值代入 NSLog 中以将这些变量的值输出到控制台屏幕。

### 说明

可以使用 NSLog 将对象与原生类型值输出到控制台。每种类型都有不同的指示符，并且指示符必须用作值的占位符。当在输出控制台中显示字符串时，需要将指示符放在你希望值出现的位置。你可以放置多个指示符，但必须确保在调用 NSLog 时包含每个值。

比如，存在名为 myInteger 的整型变量和名为 myCharacter 的字符变量，你想将这些值输出到控制台，那么可以这样做:

```
NSLog(@"myCharacter = %c and myInteger = %i", myCharacter, myInteger);
```

**警告:**

**NSLog 字符串中的每个指示符都必须在右边以逗号分隔的列表中有对应的值，否则编译器会在编译期间抛出错误，指出'%'转换要比数据参数多。**

你可能会用到更多的指示符。表 1-1 列出了常用的格式指示符的列表。

表 1-1 NSLog 常用的指示符的列表

| 指示符 | 数据类型 |
| --- | --- |
| %@ | Objective-C 对象(参见 description 方法) |
| %d、%D、%i | int(有符号 32 位整数) |
| %u、%U | unsigned int(无符号 32 位整数) |
| %f | double(64 位浮点数) |
| %e | double(以科学计数法表示的 64 位浮点数) |
| %c | unsigned char(无符号 8 位字符) |
| %C | unichar(16 位字符) |
| %p | 指针(以十六进制打印) |
| %% | 转义字符，可以打印出%符号 |

## 代码

下面的代码展示了如何使用 NSLog 将各种变量的值打印到控制台：

```
#import <Foundation/Foundation.h>
int main (int argc, const char * argv[]){
      @autoreleasepool {

            //To print out primitive types:
            int myInteger = 1;
            NSLog(@"myInteger = %i", myInteger);

            float myFloatingPointNumber = 2;
            NSLog(@"myFloatingPointNumber = %f", myFloatingPointNumber);
            NSLog(@"myFloatingPointNumber in scientific notation = %e", ➥
                                                myFloatingPointNumber);


            char myCharacter = 'A';
            NSLog(@"myCharacter = %c", myCharacter);

            //To print out the % symbol
            NSLog(@"Percent Sign looks like %%");

            //To print out Objective-C objects:
            NSString *myString = @"My String";
            NSLog(@"myString = %@", myString);
            NSLog(@"myString's pointer = %p", myString);

            //To print out a series of values
            NSLog(@"myCharacter = %c and myInteger = %i", myCharacter,
                myInteger);
```

```
        }
        return 0;
}
```

### 使用

要想测试代码，请使用前面介绍的 clang 编译文件，如下所示：

```
clang -fobjc -framework Foundation main.m -o maccommandlineapp
```

在终端窗口中输入 open maccommandlineapp 来运行应用，输出如下所示：

```
myInteger = 1
myFloatingPointNumber = 2.000000
myFloatingPointNumber in scientific notation = 2.000000e+00
myCharacter = A
Percent Sign looks like %
myString = My String
myString's pointer = 0x105880110
myCharacter = A and myInteger = 1
logout

[Process completed]
```

**注意：**
**在你的输出中，myString 的指针值与作者的是不同的。**

## 1.3 创建新的自定义类

### 问题

面向对象的程序员喜欢将功能封装到对象中。为了做到这一点，需要使用特性与行为来自定义类。

### 解决方案

Objective-C 中的类需要接口与实现定义。虽然并非完全必要，但通常会将接口与实现放在单独的文件中。接口文件与类本身同名，但文件扩展名是.h。实现文件也与类同名，但文件扩展名是.m。

要想使用自定义类，需要将类的头文件导入到使用类的代码文件中。接下来就可以通过类，实例化对象并使用类中封装的功能了。

### 说明

第一步是添加两个文件来存放编写的自定义类代码，可以使用任何文本编辑器来编

写。假设需要用于表示汽车的类。在该例中，只需要添加两个新文件即可：Car.h 与 Car.m。将这两个文件放到与 main.m 文件相同的目录中，这样稍后在编译时就会轻松一些(参见程序清单 1-1～1-3)。

在 Car.h 文件中，找到 Car 类的接口部分。类的接口必须以@interface 关键字开头，以@end 关键字结束。这两个关键字之间的部分定义了类的属性与方法。下面展示了定义 Car 类的必要代码：

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

@interface Car : NSObject
@end
```

**注意：**

**在 Car 类的定义中再次导入了 Foundation 框架，并且在 Car 名字后紧跟着“NSObject”，这表示你定义的 Car 类是 NSObject 的子类。事实上，NSObject 是 Objective-C 中的根类，所有其他对象要么是 NSObject 的子类，要么是 NSObject 子类的子类。**

Car.m 文件类似于 Car.h 文件。首先导入 Car.h 文件，接下来使用@implementation 关键字声明你要实现自定义类。用于实现的所有代码都位于实现 Car 类的声明代码行之后。下述代码展示了到目前为止 Car 类的实现：

```
#import "Car.h"

@implementation Car
@end
```

为了使用 Car 类，需要导入 Car.h 并根据类来实例化对象。为了实例化对象，需要发送两条消息：alloc 与 init。这两条消息都来自于 NSObject 父类。

```
Car *car = [[Car alloc] init];
```

## 代码

**程序清单 1-1** Car.h

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

@interface Car : NSObject
@end
```

**程序清单 1-2** Car.m

```
#import "Car.h"

@implementation Car
@end
```

**程序清单 1-3** main.m

```
#import "Car.h"

int main (int argc, const char * argv[]){
        @autoreleasepool {
                Car *car = [[Car alloc] init];
                NSLog(@"car is %@", car);

        }
        return 0;
}
```

### 使用

要想使用上述代码，请按照之前的方式编译文件，只不过除了 main.m 代码文件外，还需要将 Car 类的文件包含进来：

```
clang -fobjc -framework Foundation Car.m main.m -o maccommandlineapp
```

在命令文本中，需要将 Car.m 文件放在 main.m 文件前。在打开 maccommandlineapp 时，会看到如下输出：

```
car is <Car: 0x10c411cd0>
logout

[Process completed]
```

当然，只有在添加了自定义的属性与方法后，Car 类才有意义，接下来的攻略就将介绍这一点。

## 1.4 编写属性访问器

### 问题

自定义类需要用于表示它们所建模实体的属性，你需要知道如何在 Objective-C 中通过定义与实现属性来做到这一点。

### 解决方案

要想实现自定义类的属性，需要在类接口中声明属性并在类的实现中实现这些属性。实现属性后，就可以在需要时通过访问这些属性在其他代码文件中使用它们。

### 说明

在向类中添加属性时，首先需要与自定义类的头文件打交道。需要两个东西：持有属

性值的局部实例变量和属性声明。接口看起来如下所示：

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

@interface Car : NSObject{
@private
    NSString *name_;
}

@property(strong) NSString *name;
@end
```

这里的局部实例叫做 name_，属性声明以关键字@property 开头。注意，属性声明在类名前的圆括号中有个单词 strong。这个单词叫做属性特性，strong 是众多属性描述符中的一个。表 1-2 列出了属性特性的列表。

表 1-2 属性特性

| 特 性 | 说 明 |
| --- | --- |
| readwrite | 属性需要 getter 与 setter(默认值) |
| readonly | 属性只需要 getter(对象无法设置属性) |
| strong | 属性具有强关系(对象将被保持) |
| weak | 当目标对象解除分配时，属性将被设为 nil |
| assign | 属性只使用赋值(用于原生类型) |
| copy | 属性返回一个副本，并且必须实现 NSCopying 协议 |
| retain | 在 setter 方法中会发送一条 retain 消息 |
| nonatomic | 表示属性不是原子的(在访问时不会被锁定) |

接下来要处理的是实现，在该例中，实现位于 Car.m 中。需要在这里编写所谓的 getter 与 setter：

```
#import "Car.h"

@implementation Car

-(void)setName:(NSString *)name{
    name_ = name;
}

-(NSString *) name{
    return name_;
}

@end
```

可以像下面这样通过点符号来设置与获取属性值：

```
car.name = @"Sports Car";
```

```
NSLog(@"car is %@", car.name);
```

也可以通过标准的 Objective-C 消息来使用属性：

```
[car setName:@"New Car Name"];
NSLog(@"car.name is %@", [car name]);
```

随着阅读的 Objective-C 代码越来越多，你会看到访问属性的这两种方式的更多应用。点符号(第 1 个示例)是相对较新的 Objective-C 特性，是在 Objective-C 2.0 中被引入进来的。注意，对于熟悉其他编程语言(点符号是标准做法)的程序员来说，他们会更加熟悉点符号。第 2 个示例使用了常规的 Objective-C 消息，这种方式现在仍然会被经常用到。到底选择哪种方式取决于个人偏好。参见程序清单 1-4～1-6。

## 代码

**程序清单 1-4** Car.h

```
import <Foundation/Foundation.h>

@interface Car : NSObject{
@private
    NSString *name_;
}

@property(strong) NSString *name;

@end
```

**程序清单 1-5** Car.m

```
#import "Car.h"

@implementation Car

-(void)setName:(NSString *)name{
    name_ = name;
}

-(NSString *) name{
    return name_;
}

@end
```

**程序清单 1-6** main.m

```
#import "Car.h"
int main (int argc, const char * argv[]){
        @autoreleasepool {
                Car *car = [[Car alloc] init];
```