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

北京市版权局著作权合同登记号　图字：01-2012-8973

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：

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

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3章

使用对象集合

本章介绍如何通过 Foundation 框架使用 Objective-C 处理数组与字典。

本章内容：

- 使用 NSArray 与 NSMutableArray 创建数组
- 在数组中添加、删除与插入对象
- 数组的搜索与排序
- 通过不同方式遍历数组
- 将数组的内容保存到文件系统中
- 使用 NSDictionary 与 NSMutableDictionary 创建字典
- 在字典中添加与删除对象
- 通过不同方式遍历字典
- 将字典的内容保存到文件系统中
- 使用 NSSet 与 NSMutableSet 创建集合
- 根据对象内容比较集合
- 通过不同方式遍历集合
- 在集合中添加与删除对象

注意：

在 Objective-C 中有 3 类对象集合：数组、字典与集合。到底使用哪种集合取决于应用的需要。

- 数组在列表中组织对象，列表通过整数进行索引。
- 字典通过键组织对象；字典中的每个对象都与某个键相关联，可以通过键检索对象。
- 集合包含对象，但不要假定这些对象以一定的顺序排列。集合中的对象也必须是唯一的(不能重复)。从集合中检索对象是非常快的，因为集合没有索引带来的系统开销。因此，你会看到在需要考虑性能的场合下会用到集合。

3.1 创建数组

问题

应用需要在列表中组织对象。

解决方案

Objective-C 提供了两个 Foundation 类来创建对象列表，它们分别是 NSArray 与 NSMutableArray。如果列表不需要改变，那么请使用 NSArray 类；如果要向数组添加和删除对象，那么请使用 NSMutableArray 类。

说明

在 Objective-C 中，数组的创建与其他对象类似：使用 alloc 与 init 构造函数，或者使用诸如 arrayWithObjects:之类的便捷函数创建数组。如果使用 NSArray 创建数组，那么一旦数组创建完毕，就无法再进行修改。使用 NSMutableArray 创建的数组可以在后面修改。

下面的示例展示了如何创建字符串数组：

```
NSArray *listOfLetters = [NSArray arrayWithObjects:@"A", @"B", @"C", nil];
```

在使用 arrayWithObjects:创建数组时，需要传递使用逗号分隔的对象列表并以 nil 结束。该例使用了 NSString 对象，但对于 NSArray 与 NSMutableArray 来说，可以使用任何对象，包括从自定义类实例化得来的对象。

如果使用 NSMutableArray，那么可以通过相同的构造函数来创建数组(NSMutableArray 是 NSArray 的子类)。如果后面还要向数组中添加对象，那么还可以通过 alloc 与 init 来创建 NSMutableArray 对象。表 3-1 列出了 NSArray 与 NSMutableArray 类的完整构造函数列表，程序清单 3-1 列出了相关的代码。

表 3-1 NSArray 与 NSMutableArray 类的构造函数

构 造 函 数	说 明
- (id)initWithObjects:(const id [])objects count:(NSUInteger)cnt;	使用指定的对象与数量初始化数组
- (id)initWithObjects:(id)firstObj, ... NS_REQUIRES_NIL_TERMINATION;	使用指定的以 nil 结束的对象列表初始化数组
- (id)initWithArray:(NSArray *)array;	使用另一个数组初始化数组
- (id)initWithArray:(NSArray *)array copyItems:(BOOL)flag;	使用另一个数组初始化数组，为数组中的每个对象创建新的副本
- (id)initWithContentsOfFile:(NSString *)path;	使用本地文件的内容初始化数组
- (id)initWithContentsOfURL:(NSURL *)url;	使用 URL 的内容初始化数组

代码

程序清单 3-1 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
      NSArray *listOfLetters1 = [NSArray arrayWithObjects:@"A", @"B", @"C", nil];
        NSLog(@"listOfLetters1 = %@", listOfLetters1);

        NSNumber *number1 = [NSNumber numberWithInt:1];
        NSNumber *number2 = [NSNumber numberWithInt:2];
        NSNumber *number3 = [NSNumber numberWithInt:3];

        NSMutableArray *listOfNumbers1 = [[NSMutableArray alloc] ➥
           initWithObjects:number1, number2, number3, nil];

        NSLog(@"listOfNumbers1 = %@", listOfNumbers1);

        id list[3];
        list[0] = @"D";
        list[1] = @"E";
        list[2] = @"F";

        NSMutableArray *listOfLetters2 = [[NSMutableArray alloc] ➥
           initWithObjects:list
                     count:3];

        NSLog(@"listOfLetters2 = %@", listOfLetters2);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。通过日志查看每个数组的内容。接下来的攻略将会介绍如何引用这些数组元素，这样就可以将它们的内容打印到日志或是在程序的其他地方使用它们了。

3.2　引用数组中的对象

问题

你想要获得指向数组中对象的引用以访问它们的属性或是向对象发送消息。

解决方案

可以使用 objectAtIndex:方法获取数组中位于某个整数位置的对象引用，还可以通过 lastObject 函数获取数组中最后一个对象的引用。

说明

NSArray 在列表中组织对象，列表通过以 0 开始的整数进行索引。如果你想要获得数组中某个对象的引用并且知道这个对象的位置，那么可以通过 objectAtIndex:方法获得指向这个对象的引用：

```
NSString *stringObject1 = [listOfLetters objectAtIndex:0];
```

通过函数 lastObject 可以快速获得数组中最后一个对象的引用：

```
NSString *stringObject2 = [listOfLetters lastObject];
```

通常情况下，你并不知晓对象在数组中的位置。如果已经获得对象的引用，那么可以通过 indexOfObject:函数并且将对象引用作为参数来获得对象在数组中的位置：

```
NSUInteger position = [listOfLetters indexOfObject:@"B"];
```

参见程序清单 3-2。

代码

程序清单 3-2 main.m

```
import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSMutableArray *listOfLetters = [NSMutableArray➥
          arrayWithObjects:@"A", @"B", @"C", nil];
        NSString *stringObject1 = [listOfLetters objectAtIndex:0];
        NSLog(@"stringObject1 = %@", stringObject1);
        NSString *stringObject2 = [listOfLetters lastObject];
        NSLog(@"stringObject2 = %@", stringObject2);
        NSUInteger position = [listOfLetters indexOfObject:@"B"];
        NSLog(@"position = %lu", position);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。从控制台输出可以看到，对象

已被成功引用：

```
stringObject1 = A
stringObject2 = C
position = 1
```

3.3　获取数组中元素的数量

问题

应用使用数组中的内容，你需要知道数组中有多少个元素以便恰当地展现内容。

解决方案

NSArray 对象提供了 count 属性，可以通过这个属性获得数组中元素的数量。

说明

要想使用 count 属性，可以对任何数组对象使用点符号(listOfLetters.count)或是发送 count 消息([listOfLetters count])以获得数组中元素的数量。参见程序清单 3-3。

代码

程序清单 3-3 main.m

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

int main (int argc, const char * argv[])
{

    @autoreleasepool {
        NSMutableArray *listOfLetters = [NSMutableArray➥
            arrayWithObjects:@"A", @"B", @"C", nil];
        NSLog(@"listOfLetters has %lu elements", listOfLetters.count);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。日志消息会显示出元素的数量：

```
listOfLetters has 3 elements
```

3.4 遍历数组

问题

你拥有对象数组，想要向数组中的每个对象发送同样的消息或是访问同样的属性。

解决方案

NSArray 对象提供了 3 种内置方式来遍历对象列表。很多人都使用 for-each 循环遍历数组中的每个元素。通过这种结构，可以使用相同的代码来遍历数组中的每个元素。

还可以使用 makeObjectsPerformSelector:withObject:方法，在这种情况下，可以传递希望每个对象都执行的方法名和一个参数。

最后，还可以通过 enumerateObjectsUsingBlock:方法将代码块作为参数应用到数组中的每个对象上。该方法的作用与 for-each 循环一样，但无须为循环本身编写代码，并且可以通过参数追踪当前元素的索引。

说明

由于该攻略需要一些示例，因此创建 3 个 NSMutableString 对象并放到新的数组中：

```
NSMutableString *string1 = [NSMutableString stringWithString:@"A"];
NSMutableString *string2 = [NSMutableString stringWithString:@"B"];
NSMutableString *string3 = [NSMutableString stringWithString:@"C"];

NSArray *listOfObjects = [NSArray arrayWithObjects:string1, string2, ➥
  string3, nil];
```

要想使用 for-each 循环遍历 NSArray 数组，需要创建循环并提供局部变量，局部变量可以引用数组中的当前对象。你还需要添加代码供列表中的每个对象执行：

```
for(NSMutableString *s in listOfObjects){
   NSLog(@"This string in lowercase is %@", [s lowercaseString]);
}
```

for-each 循环遍历数组中的每个可变字符串并将消息写到日志中。lowercaseString 是 NSString 类的成员函数，能以小写形式返回字符串。

无需通过 for-each 循环就可以向数组中的每个对象发送消息，方式是发送 makeObjectsPerformSelector:withObject:消息。需要将@selector 关键字传递到方法中并将方法名放到圆括号中。在属性修饰 withObject:的后面传递参数值：

```
[listOfObjects makeObjectsPerformSelector:@selector(appendString:) ➥
    withObject:@"-MORE"];
```

这里所做的是向数组中的每个可变字符串发送 appendString:消息以及字符串参数

@"-MORE"。现在，每个字符串最终都会被修改，其中包含了额外的字符。

警告：

确保数组中的对象可以响应你所发出的消息。如果向数组中的对象发送它们无法响应的消息，那么程序就会崩溃，你会收到一条错误消息——“unrecognized selector sent to instance”。

可以使用块定义应用到数组中的每个对象的代码块。块是一种代码封装方式，这样一来，代码就可以被当作对象，并以参数形式传递给其他对象。借助于 NSArray 方法 enumerateObjectsUsingBlock:，就可以对数组中的每个元素执行代码块：

```
[listOfObjects enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop)
{
    NSLog(@"object(%lu)'s description is %@",idx, [obj description]);
}];
```

这与 for-each 循环的作用一样，但你只需要使用一行代码即可。此外，块还为需要引用的对象提供了内置参数和索引以帮助你追踪当前在数组中的位置。该例通过块并使用对象的 description 函数与 index 参数，针对列表中的每个对象向日志中写入了一条消息。参见程序清单 3-4。

代码

程序清单 3-4 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSMutableString *string1 = [NSMutableString stringWithString:@"A"];
        NSMutableString *string2 = [NSMutableString stringWithString:@"B"];
        NSMutableString *string3 = [NSMutableString stringWithString:@"C"];

        NSArray *listOfObjects = [NSArray arrayWithObjects:string1, string2, ➥
           string3,  nil];

        for(NSMutableString *s in listOfObjects){
            NSLog(@"This string in lowercase is %@", [s lowercaseString]);
        }

        [listOfObjects makeObjectsPerformSelector:@selector(appendString:)
             withObject:@"-MORE"];
       [listOfObjects enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, ➥
 BOOL *stop) {
            NSLog(@"object(%lu)'s description is %@",idx, [obj description]);
        }];
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。日志消息会显示出对之前所创建数组应用各种遍历方式后的结果：

```
This string in lowercase is a
This string in lowercase is b
This string in lowercase is c
object(0)'s description is A-MORE
object(1)'s description is B-MORE
object(2)'s description is C-MORE
```

3.5 排序数组

问题

你使用数组来组织自定义对象，希望对象按照各自属性值的顺序出现在列表中。

解决方案

为用于数组排序的每个属性创建 NSSortDescriptor 对象，将所有这些 NSSortDescriptor 对象放到一个数组中，该数组将会在后面用作参数。使用 NSArray 类的 sortedArrayUsing-Descriptors:方法并将 NSSortDescriptor 对象数组作为参数传递进去，结果会返回一个数组，这个数组中的对象已根据你指定的属性排好序。

说明

本攻略会使用到 Person 对象。程序清单 3-5 展示了 Person 对象的类定义。虽然可以通过该攻略对字符串或数字对象排序，但只有当你使用 NSSortDescriptor 处理自定义对象时，才会真正感受到它的强大功能。

自定义类叫做 Person，它有 3 个属性，分别是 firstName、lastName 与 age。Person 类还有两个方法，分别是 reportState 与 initWithFirstName:lastName:andAge:，后者是自定义构造函数。

首先，创建 Person 对象数组：

```
//Instantiate Person objects and add them all to an array:
Person *p1 = [[Person alloc] initWithFirstName:@"Rebecca"
                                      lastName:@"Smith"
                                        andAge:33];
Person *p2 = [[Person alloc] initWithFirstName:@"Albert"
                                      lastName:@"Case"
                                        andAge:24];
Person *p3 = [[Person alloc] initWithFirstName:@"Anton"
                                      lastName:@"Belfey"
```

```
                                    andAge:45];
Person *p4 = [[Person alloc] initWithFirstName:@"Tom"
                                    lastName:@"Gun"
                                      andAge:17];
Person *p5 = [[Person alloc] initWithFirstName:@"Cindy"
                                    lastName:@"Lou"
                                      andAge:6];
Person *p6 = [[Person alloc] initWithFirstName:@"Yanno"
                                    lastName:@"Dirst"
                                      andAge:76];

NSArray *listOfObjects = [NSArray arrayWithObjects:p1, p2, p3, p4, p5, p6,  nil];
```

如果打印数组中的每个元素，对象就会以你将它们放到数组中时的顺序出现。如果想要根据每个人的年龄、姓与名排序数组，那么可以使用 NSSortDescriptor 对象。需要为用于排序的每个 Person 属性提供排序描述符：

```
//Create three sort descriptors and add to an array:
NSSortDescriptor *sd1 = [NSSortDescriptor sortDescriptorWithKey:@"age"
                                                      ascending:YES];

NSSortDescriptor *sd2 = [NSSortDescriptor sortDescriptorWithKey:@"lastName"
                                                    ascending:YES];

NSSortDescriptor *sd3 = [NSSortDescriptor sortDescriptorWithKey:@"firstName"
                                                    ascending:YES];

NSArray *sdArray1 = [NSArray arrayWithObjects:sd1, sd2, sd3, nil];
```

需要将属性名以字符串的形式传递进去并指定根据属性进行升序还是降序排序。最后，所有排序描述符需要放在数组中。

数组中每个排序描述符的位置决定了对象的排序顺序。因此，如果希望数组先根据年龄，然后根据姓进行排序，那么确保先将与年龄对应的排序描述符添加进来，然后再将与姓对应的排序描述符添加进来。

要想获得排好序的数组，请向待排序的数组发送 sortedArrayUsingDescriptors:消息并将排序描述符数组作为参数传递进来：

```
NSArray  *sortedArray1 = [listOfObjects sortedArrayUsingDescriptors:sdArray1];
```

要想查看结果，请使用 makeObjectsPerformSelector:方法将排好序的数组中的每个对象的状态输出到日志中：

```
[sortedArray1 makeObjectsPerformSelector:@selector(reportState)];
```

这会以排序描述符(年龄、姓、名)指定的顺序将每个 Person 对象的详细信息打印到日志中。参见程序清单 3-5～3-7。

代码

程序清单 3-5 Person.h

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

@interface Person : NSObject
@property(strong) NSString *firstName;
@property(strong) NSString *lastName;
@property(assign) int age;

-(id)initWithFirstName:(NSString *)fName lastName:(NSString *)lName
   andAge:(int)a;
-(void)reportState;

@end
```

程序清单 3-6 Person.m

```
#import "Person.h"

@implementation Person
@synthesize firstName, lastName, age;

-(id)initWithFirstName:(NSString *)fName lastName:(NSString *)lName
   andAge:(int)a{
    self = [super init];
    if (self) {
        self.firstName = fName;
        self.lastName = lName;
        self.age = a;
    }
    return self;
}

-(void)reportState{
    NSLog(@"This person's name is %@ %@ who is %i years old", firstName,
       lastName, age);
}

@end
```

程序清单 3-7 main.m

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

int main (int argc, const char * argv[])
{

    @autoreleasepool {
```

```
        //Instantiate Person objects and add them all to an array:
        Person *p1 = [[Person alloc] initWithFirstName:@"Rebecca"
                                              lastName:@"Smith"
                                                andAge:33];
        Person *p2 = [[Person alloc] initWithFirstName:@"Albert"
                                              lastName:@"Case"
                                                andAge:24];
        Person *p3 = [[Person alloc] initWithFirstName:@"Anton"
                                              lastName:@"Belfey"
                                                andAge:45];
        Person *p4 = [[Person alloc] initWithFirstName:@"Tom"
                                              lastName:@"Gun"
                                                andAge:17];
        Person *p5 = [[Person alloc] initWithFirstName:@"Cindy"
                                              lastName:@"Lou"
                                                andAge:6];
        Person *p6 = [[Person alloc] initWithFirstName:@"Yanno"
                                              lastName:@"Dirst"
                                                andAge:76];
        NSArray *listOfObjects = [NSArray arrayWithObjects:p1, p2, p3, p4, ➥
          p5, p6, nil];
        NSLog(@"PRINT OUT ARRAY UNSORTED");
        [listOfObjects makeObjectsPerformSelector:@selector(reportState)];

        //Create three sort descriptors and add to an array:
        NSSortDescriptor *sd1 = [NSSortDescriptor ➥
           sortDescriptorWithKey:@"age"ascending:YES];

        NSSortDescriptor *sd2 = [NSSortDescriptor ➥
           sortDescriptorWithKey:@"lastName"ascending:YES];

        NSSortDescriptor *sd3 = [NSSortDescriptor ➥
           sortDescriptorWithKey:@"firstName"ascending:YES];

        NSArray *sdArray1 = [NSArray arrayWithObjects:sd1, sd2, sd3, nil];

        NSLog(@"PRINT OUT SORTED ARRAY (AGE,LASTNAME,FIRSTNAME)");
        NSArray *sortedArray1 = [listOfObjects➥
            sortedArrayUsingDescriptors:sdArray1];
        [sortedArray1 makeObjectsPerformSelector:@selector(reportState)];
        NSArray *sdArray2 = [NSArray arrayWithObjects:sd2, sd1, sd3, nil];
        NSArray *sortedArray2 = [listOfObjects➥
            sortedArrayUsingDescriptors:sdArray2];
        NSLog(@"PRINT OUT SORTED ARRAY (LASTNAME,FIRSTNAME,AGE)");
        [sortedArray2 makeObjectsPerformSelector:@selector(reportState)];

    }
    return 0;
}
```

使用

要想使用上述代码，需要为 Person 类创建文件。这是一个 Objective-C 类，可以使用 Xcode 文件模板创建。Person 类必须导入到 main.m 文件的代码中(对于 Mac 命令行应用)。构建并运行项目，然后通过控制台日志查看数组的排序结果：

```
PRINT OUT ARRAY UNSORTED
This person's name is Rebecca Smith who is 33 years old
This person's name is Albert Case who is 24 years old
This person's name is Anton Belfey who is 45 years old
This person's name is Tom Gun who is 17 years old
This person's name is Cindy Lou who is 6 years old
This person's name is Yanno Dirst who is 76 years old
PRINT OUT SORTED ARRAY (AGE,LASTNAME,FIRSTNAME)
This person's name is Cindy Lou who is 6 years old
This person's name is Tom Gun who is 17 years old
This person's name is Albert Case who is 24 years old
This person's name is Rebecca Smith who is 33 years old
This person's name is Anton Belfey who is 45 years old
This person's name is Yanno Dirst who is 76 years old
PRINT OUT SORTED ARRAY (LASTNAME,FIRSTNAME,AGE)
This person's name is Anton Belfey who is 45 years old
This person's name is Albert Case who is 24 years old
This person's name is Yanno Dirst who is 76 years old
This person's name is Tom Gun who is 17 years old
This person's name is Cindy Lou who is 6 years old
This person's name is Rebecca Smith who is 33 years old
```

3.6 查询数组

问题

你拥有填满了对象的数组，想要根据某些条件(这些条件可以通过 iOS 应用表格中的搜索栏等类似控件进行输入)找出数组的某个子集。

解决方案

你首先需要的是 NSPredicate 对象，NSPredicate 用于定义搜索查询。接下来，可以使用原始数组的 filteredArrayUsingPredicate:函数并根据定义的 NSPredicate 对象规范返回数组的子集。

说明

该攻略会用到 3.5 节中使用的 Person 对象。你要定义如下判断，让返回的数组只包含年龄大于 30 的 Person 对象：

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"age > 30"];
```

判断需要对数组中的对象应用逻辑表达式。这里的 age 必须是待查询数组中对象的某个属性。使用的比较运算符与编程时使用的相同(表 3-2 完整列出了 NSPredicate 比较运算符)。

表 3-2　NSPredicate 比较运算符

运 算 符		说　　明
基本比较	=、==	左侧等于右侧
	>=、=>	左侧大于等于右侧
	<=、=<	左侧小于等于右侧
	>	左侧大于右侧
	<	左侧小于右侧
	!=、<>	左侧不等于右侧
	BETWEEN	左侧介于右侧指定的值之间
复合判断	AND、&&	逻辑与
	OR、||	逻辑或
	NOT、!	逻辑非
字符串比较	BEGINSWITH	左侧以右侧的表达式开始
	CONTAINS	左侧包含右侧的表达式
	ENDSWITH	左侧以右侧的表达式结束
	LIKE	左侧等于右侧(?与*是通配符)
	MATCHES	左侧匹配右侧的正则表达式
聚合运算符	ANY、SOME	指定右侧表达式中的任意元素
	ALL	指定右侧表达式中的所有元素
	NONE	不指定右侧表达式中的任何元素
	IN	左侧必须位于右侧指定的集合中

创建好判断后，你只需要使用 filteredArrayUsingPredicate:函数并将 NSPredicate 对象作为参数传递进去，即可获得数组的子集：

```
NSArray *arraySubset = [listOfObjects filteredArrayUsingPredicate:predicate];
```

你会获得另一个数组，该数组只包含与 NSPredicate 对象中所定义规范相匹配的那些对象。参见程序清单 3-8～3-10。

代码

程序清单 3-8 `Person.h`

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

```
@interface Person : NSObject
@property(strong) NSString *firstName;
@property(strong) NSString *lastName;
@property(assign) int age;

-(id)initWithFirstName:(NSString *)fName lastName:(NSString *)lName
   andAge:(int)a;
-(void)reportState;

@end
```

程序清单 3-9 Person.m

```
#import "Person.h"

@implementation Person
@synthesize firstName, lastName, age;

-(id)initWithFirstName:(NSString *)fName lastName:(NSString *)lName
  andAge:(int)a{
    self = [super init];
    if (self) {
        self.firstName = fName;
        self.lastName = lName;
        self.age = a;
    }
    return self;
}

-(void)reportState{
    NSLog(@"This person's name is %@ %@ who is %i years old", firstName,
      lastName, age);
}

@end
```

程序清单 3-10 main.m

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

int main (int argc, const char * argv[])
{

    @autoreleasepool {
        //Instantiate Person objects and add them all to an array:
        Person *p1 = [[Person alloc] initWithFirstName:@"Rebecca"
                                        lastName:@"Smith"
                                          andAge:33];
        Person *p2 = [[Person alloc] initWithFirstName:@"Albert"
                                        lastName:@"Case"
                                          andAge:24];
        Person *p3 = [[Person alloc] initWithFirstName:@"Anton"
                                        lastName:@"Belfey"
```

```
                                        andAge:45];
        Person *p4 = [[Person alloc] initWithFirstName:@"Tom"
                                      lastName:@"Gun"
                                        andAge:17];
        Person *p5 = [[Person alloc] initWithFirstName:@"Cindy"
                                      lastName:@"Lou"
                                        andAge:6];
        Person *p6 = [[Person alloc] initWithFirstName:@"Yanno"
                                      lastName:@"Dirst"
                                        andAge:76];

        NSArray *listOfObjects = [NSArray arrayWithObjects:p1, p2, p3, p4, ➥
          p5, p6,  nil];
        NSPredicate *predicate = [NSPredicate predicateWithFormat:@"age > 30"];

        NSArray *arraySubset = [listOfObjects ➥
           filteredArrayUsingPredicate:predicate];
        NSLog(@"PRINT OUT ARRAY SUBSET");
        [arraySubset makeObjectsPerformSelector:@selector(reportState)];

    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。通过控制台查看根据 NSPredicate 对象的查询结果。

3.7　操纵数组内容

问题

你希望数组内容能够更具动态性，这样用户或你就可以在数组中添加、删除和插入对象了。然而，NSArray 是不可变类，因此一旦创建 NSArray 对象，你就无法再对其中内容进行任何修改。

解决方案

如果你认为所用的数组需要是动态的，那么请使用 NSMutableArray。NSMutableArray 是 NSArray 的子类，这样就可以像 NSArray 那样使用 NSMutableArray 了。但 NSMutableArray 提供了额外的一些方法，可以通过这些方法在数组列表中添加、删除和插入对象。

说明

首先实例化 NSMutableArray 类，可以使用任何构造函数进行实例化。要想创建新的空

的 NSMutableArray 对象，只需要使用 alloc 与 init 即可：

```
NSMutableArray *listOfLetters = [[NSMutableArray alloc] init];
```

要想向数组中添加对象，需要向数组发送 addObject:消息并且将想要添加到数组中的对象作为参数传递进去：

```
[listOfLetters addObject:@"A"];
[listOfLetters addObject:@"B"];
[listOfLetters addObject:@"C"];
```

在使用 addObject:时，总是会将对象添加到数组列表的末尾。如果想要将对象插入到数组中的其他位置，那么需要使用 insertObject:atIndex:方法：

```
[listOfLetters insertObject:@"a"
                    atIndex:0];
```

这会将对象插入到数组的首个位置。

如果想要将某个索引位置的对象替换为另一个对象，那么可以使用 replaceObjectAtIndex:withObject:方法。下面的代码展示了如何将字符 C 替换为小写字母 c：

```
[listOfLetters replaceObjectAtIndex:2
                         withObject:@"c"];
```

要想交换数组中两个对象的位置，可以使用 exchangeObjectAtIndex:withObjectAtIndex:方法：

```
[listOfLetters exchangeObjectAtIndex:0
                   withObjectAtIndex:2];
```

当需要删除数组中的对象时，可以选择几种不同的方法。可以删除特定索引位置的对象，可以删除数组中的最后一个对象，还可以删除列表中的全部对象。如果拥有某个对象的引用，那么还可以通过对象引用删除数组中的这个对象。如下代码展示了删除对象的各种方式：

```
[listOfLetters removeObject:@"A"];
[listOfLetters removeObjectAtIndex:1];
[listOfLetters removeLastObject];
[listOfLetters removeAllObjects];
```

参见程序清单 3-11。

代码

程序清单 3-11 main.m

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

int main (int argc, const char * argv[])
{
```

```
    @autoreleasepool {

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

        [listOfLetters addObject:@"A"];
        [listOfLetters addObject:@"B"];
        [listOfLetters addObject:@"C"];

        NSLog(@"OBJECTS ADDED TO ARRAY: %@", listOfLetters);
        [listOfLetters insertObject:@"a"
                            atIndex:0];
        NSLog(@"OBJECT 'a' INSERTED INTO ARRAY: %@", listOfLetters);
        [listOfLetters replaceObjectAtIndex:2
                                 withObject:@"c"];
        NSLog(@"OBJECT 'c' REPLACED 'C' IN ARRAY: %@", listOfLetters);
        [listOfLetters exchangeObjectAtIndex:0
                           withObjectAtIndex:2];

        NSLog(@"OBJECT AT INDEX 1 EXCHANGED WITH OBJECT AT INDEX 2 IN ARRAY: ➥
          %@", listOfLetters);

        [listOfLetters removeObject:@"A"];
         NSLog(@"OBJECT 'A' REMOVED IN ARRAY: %@", listOfLetters);
        [listOfLetters removeObjectAtIndex:1];
         NSLog(@"OBJECT AT INDEX 1 REMOVED IN ARRAY: %@", listOfLetters);
        [listOfLetters removeLastObject];
         NSLog(@"LAST OBJECT REMOVED IN ARRAY: %@", listOfLetters);
        [listOfLetters removeAllObjects];
         NSLog(@"ALL OBJECTS REMOVED IN ARRAY: %@", listOfLetters);

    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。通过控制台查看在使用每个操作后数组都发生了什么变化：

```
OBJECTS ADDED TO ARRAY: (
    A,
    B,
    C
)
OBJECT 'a' INSERTED INTO ARRAY: (
    a,
    A,
    B,
    C
)
OBJECT 'c' REPLACED 'C' IN ARRAY: (
    a,
    A,
    B,
```

```
    c
)
OBJECT AT INDEX 1 EXCHANGED WITH OBJECT AT INDEX 2 IN ARRAY: (
    B,
    A,
    a,
    c
)
OBJECT 'A' REMOVED IN ARRAY: (
    B,
    a,
    c
)
OBJECT AT INDEX 1 REMOVED IN ARRAY: (
    B,
    c
)
LAST OBJECT REMOVED IN ARRAY: (
    B
)
ALL OBJECTS REMOVED IN ARRAY: (
)
```

3.8 将数组保存到文件系统中

问题

你想将数组中的对象保存到文件系统中以供其他程序使用。

解决方案

如果数组包含了数字或字符串对象的列表，那么可以将所有这些对象保存到文件系统中以供后续使用，使用 writeToFile:atomically:方法可以做到这一点。注意，该方法无法处理自定义对象。自定义对象需要使用 NSCoding 协议并使用归档类(参见第 9 章)或 Core Data(参见第 10 章)。

说明

该攻略需要创建如下由字符串与数字组成的数组：

```
NSArray *listOfObjects = [NSArray arrayWithObjects:@"A", @"B", @"C",
[NSNumber numberWithInt:1], [NSNumber numberWithInt:2], [NSNumber numberWithInt:3],
nil];
```

要想将数组保存到文件系统中，首先需要如下文件引用：

```
NSString *filePathName = @"/Users/Shared/array.txt";
```

说明：

本攻略假设你使用的是 Mac 应用。iOS 文件引用的工作方式与此不同，参见 2.5 节以了解如何获得 iOS 文件引用。

现在可以使用 writeToFile:atomically:方法将数组内容写到 Mac 文件系统中：

```
[listOfObjects writeToFile:filePathName
                atomically:YES];
```

参见程序清单 3-12。

代码

程序清单 3-12 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
       NSArray *listOfObjects = [NSArray arrayWithObjects:@"A", @"B", @"C",
         [NSNumber numberWithInt:1], [NSNumber numberWithInt:2], [NSNumber
         numberWithInt:3], nil];

        NSString *filePathName = @"/Users/Shared/array.txt";
        [listOfObjects writeToFile:filePathName
                        atomically:YES];

    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。使用 Finder 找到创建的文件，位于/Users/Shared/array.txt。下面列出文本文件的内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<array>
       <string>A</string>
       <string>B</string>
       <string>C</string>
       <integer>1</integer>
       <integer>2</integer>
       <integer>3</integer>
</array>
```

```
</plist>
```

数据以 XML 格式组织为属性列表(一种存储键数据的 Objective-C 格式)。

3.9 从文件系统读取数组

问题

有一些应用可以访问到的文件，其中包含了以数组形式组织的内容，你想在应用中使用这些内容。

解决方案

如果你有通过数组创建的文件(使用 writeToFile:atomically:方法保存)，那么可以使用 initWithContentsOfFile:构造函数实例化新的数组，新数组是由文件内容构成的。

说明

本攻略使用 3.8 节中的文件(已将数组的内容保存到文件系统中)。因此，这里会使用相同的文件路径名：

```
NSString *filePathName = @"/Users/Shared/array.txt";
```

注意：

本攻略假设你使用的是 Mac 应用。iOS 文件引用的工作方式与此不同，参见 2.5 节以了解如何获得 iOS 文件引用。

接下来就可以使用 initWithContentsOfFile:构造函数创建由文件内容构成的新数组：

```
NSArray *listOfObjects = [[NSArray alloc] initWithContentsOfFile:filePathName];
```

参见程序清单 3-13。

代码

程序清单 3-13 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSString *filePathName = @"/Users/Shared/array.txt";
        NSArray *listOfObjects = [[NSArray alloc] ➥
            initWithContentsOfFile:filePathName];
        NSLog(@"%@", listOfObjects);
```

```
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。通过日志可以查看数组内容：

```
(
    A,
    B,
    C,
    1,
    2,
    3
)
```

3.10　创建字典

问题

应用需要对列表中的对象进行分组，你想要通过键来引用对象。

解决方案

可以通过两个 Objective-C Foundation 类——NSDictionary 与 NSMutableDictionary 来创建带有键的对象列表。如果不需要修改列表，就使用 NSDictionary；如果在后面要向字典中添加或删除对象，就使用 NSMutableDictionary。

说明

在 Objective-C 中，字典的创建与其他对象一样：可以使用 alloc 与 init 构造函数，或者使用 dictionaryWithObjects:forKeys:等便捷函数创建字典。如果使用 NSDictionary 创建字典，那么字典一旦创建完毕后，就无法再进行修改。使用 NSMutableDictionary 创建的字典可以在后面进行修改。

在下面的示例中，创建的字典包含了不同语言版本的 Hello World。该短语的每个版本都会被录入到相应的语言中：

```
NSArray *listOfObjects = [NSArray arrayWithObjects:@"Hello World",
    @"Bonjour tout le monde", @"Hola Mundo", nil];
NSArray *listOfKeys = [NSArray arrayWithObjects:@"english", @"french",
    @"spanish", nil];
NSDictionary *dictionary2 = [NSDictionary
    dictionaryWithObjects:listOfObjects forKeys:listOfKeys];
```

NSDictionary 构造函数 arrayWithObjects:forKeys:需要两个数组作为参数。第一个数组必须包含待存储的对象，第二个数组必须包含与这些对象关联的键。

如果使用 NSMutableDictionary，那么可以通过相同的构造函数创建字典(NSMutableDictionary 是 NSDictionary 的子类)。如果后面还要向字典中添加对象，那么还可以通过 alloc 与 init 来创建 NSMutableDictionary 对象。表 3-3 完整列出了 NSDictionary 与 NSMutableDictionary 类的构造函数，程序清单 3-14 列出了相关代码。

表 3-3　NSDictionary 与 NSMutableDictionary 类的构造函数

构 造 函 数	说　明
- (id)initWithObjects:(const id [])objects forKeys:(const id [])keys count:(NSUInteger)cnt;	使用指定的对象、键与数量初始化字典
- (id)initWithObjectsAndKeys:(id)firstObject, NS_REQUIRES_NIL_TERMINATION;	使用指定的以 nil 结尾的对象与键对列表初始化字典
- (id)initWithDictionary:(NSDictionary *)otherDictionary;	使用另一个字典初始化字典
- (id)initWithDictionary:(NSDictionary *)otherDictionary copyItems:(BOOL)flag;	使用另一个字典初始化字典，还可以为每个对象创建新的副本
- (id)initWithObjects:(NSArray *)objects forKeys:(NSArray *)keys;	使用指定的对象与键初始化字典
- (id)initWithContentsOfFile:(NSString *)path;	使用本地文件的内容初始化字典
- (id)initWithContentsOfURL:(NSURL *)url;	使用 URL 的内容初始化字典

代码

程序清单 3-14 main.m

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

int main (int argc, const char * argv[])
{

    @autoreleasepool {
       NSDictionary *dictionary1 = [[NSDictionary alloc] init];
       NSArray *listOfObjects = [NSArray arrayWithObjects:@"Hello World",
          @"Bonjour tout le monde",  @"Hola Mundo", nil];
       NSArray *listOfKeys = [NSArray arrayWithObjects:@"english",
          @"french", @"spanish", nil];
        NSDictionary *dictionary2 = [NSDictionary
           dictionaryWithObjects:listOfObject forKeys:listOfKeys];

        NSLog(@"dictionary2 = %@", dictionary2);
```

```
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。可以设置断点并使用 Xcode 调试器查看这些字典的内容。3.11 节将会介绍如何引用每个字典元素，这样就可以将它们的内容打印到日志中或是在程序的其他地方使用它们了。你会看到字典的全部内容被打印到日志中：

```
dictionary2 = {
    english = "Hello World";
    french = "Bonjour tout le monde";
    spanish = "Hola Mundo";
}
```

3.11　引用数组中的对象

问题

你想要获得指向字典中对象的引用以便访问它们的属性或是向对象发送消息。

解决方案

使用 objectForKey:方法可以获得与你提供的键相对应的对象引用。

说明

NSDictionary 根据你提供的键来组织对象。这样一来，查找对象就变得非常简单和快速。只需要使用 objectForKey:并提供与想要查找的对象相对应的键，就可以获得对象引用：

```
NSString *helloWorld = [dictionary objectForKey:@"english"];
```

参见程序清单 3-15。

代码

程序清单 3-15 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSArray *listOfObjects = [NSArray arrayWithObjects:@"Hello World", ➥
```

```
            @"Bonjour tout le monde", @"Hola Mundo", nil];
        NSArray *listOfKeys = [NSArray arrayWithObjects:@"english",
            @"french", @"spanish", nil];
        NSDictionary *dictionary = [NSDictionary
            dictionaryWithObjects:listOfObjects forKeys:listOfKeys];
        NSString *helloWorld = [dictionary objectForKey:@"english"];
        NSLog(@"%@", helloWorld);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。打印出的 hello world 消息是与 English 键对应的对象：

```
Hello World
```

要想查看法语版本的 hello world 消息，请将下述代码添加到应用中：

```
helloWorld = [dictionary objectForKey:@"french"];
NSLog(@"%@", helloWorld);
```

再次运行应用，检查最后一条控制台消息，你会看到法语版本的 hello world 消息。可以针对西班牙语进行同样操作。

3.12 获取字典中元素的数量

问题

应用使用了字典中的内容，你需要知道字典中有多少元素以便能恰当地显示出内容。

解决方案

NSDictionary 对象提供了 count 属性，可以通过该属性获得字典中元素的数量。

说明

要想使用 count 属性，可以对字典对象使用点符号(dictionary.count)或是发送 count 消息([dictionary count])，从而获得字典中元素的数量。参见程序清单 3-16。

代码

程序清单 3-16 `main.m`

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

```
int main (int argc, const char * argv[])
{
    @autoreleasepool {
       NSArray *listOfObjects = [NSArray arrayWithObjects:@"Hello World", ➥
          @"Bonjour tout le monde", @"Hola Mundo", nil];
       NSArray *listOfKeys = [NSArray arrayWithObjects:@"english",➥
          @"french", @"spanish", nil];
        NSDictionary *dictionary = [NSDictionary➥
          dictionaryWithObjects:listOfObjects forKeys:listOfKeys];
        NSUInteger count = dictionary.count;
        NSLog(@"The dictionary contains %lu items", count);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。日志消息会显示出元素的数量：

```
The dictionary contains 3 items
```

3.13　遍历字典

问题

你有了对象字典，想要对字典中的每个对象发送相同的消息或是访问同样的属性。

解决方案

使用 NSDictionary 函数 allValues 将字典转换为数组，接下来就可以使用 for-each 循环了，此外还可以使用 enumerateKeysAndObjectsUsingBlock:方法来处理字典中的每个对象。

说明

NSDictionary 对象提供了内置方式来遍历对象列表。然而，如果想要使用 3.4 节中介绍的方式，那么可以临时将字典的键与对象内容转换为数组。例如，要想使用 for-each 循环遍历字典中的对象，可以通过如下代码实现：

```
for (NSString *s in [dictionary allValues]) {
    NSLog(@"value: %@", s);
}
```

NSDictionary 函数 allValues 会返回以数组而非字典形式组织的对象。函数 allKeys 会将键值作为数组返回：

```
for (NSString *s in [dictionary allKeys]) {
   NSLog(@"key: %@", s);
}
```

还可以使用块，通过 enumerateKeysAndObjectsUsingBlock:方法针对字典中的每个对象执行代码。可以用来定义代码块，然后应用到字典中的每个对象，同时又不必创建 for-each 循环或是获得数组版本的字典引用：

```
[dictionary enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
   NSLog(@"key = %@ and obj = %@", key, obj);
}];
```

参见程序清单 3-17。

代码

程序清单 3-17 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
       NSArray *listOfObjects = [NSArray arrayWithObjects:@"Hello World", ➥
          @"Bonjour tout le monde", @"Hola Mundo", nil];
       NSArray *listOfKeys = [NSArray arrayWithObjects:@"english",➥
          @"french", @"spanish", nil];
       NSDictionary *dictionary = [NSDictionary ➥
         dictionaryWithObjects:listOfObjects forKeys:listOfKeys];
       for (NSString *s in [dictionary allValues]) {
          NSLog(@"value: %@", s);
       }

       for (NSString *s in [dictionary allKeys]) {
          NSLog(@"key: %@", s);
       }

       [dictionary enumerateKeysAndObjectsUsingBlock:^(id key, id obj, ➥
       BOOL *stop) {
          NSLog(@"key = %@ and obj = %@", key, obj);
       }];

    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。日志消息会显示出使用各种方

式遍历字典后的结果：

```
value: Hello World
value: Bonjour tout le monde
value: Hola Mundo
key: english
key: french
key: spanish
key = english and obj = Hello World
key = french and obj = Bonjour tout le monde
key = spanish and obj = Hola Mundo
```

3.14　操纵字典内容

问题

你希望字典内容能够更具动态性，这样用户或你就可以在字典中添加、删除和插入对象了。然而，NSDictionary 是不可变类，因此一旦创建 NSDictionary 对象，你就无法再修改其中的内容。

解决方案

如果知道字典会动态变化，那么请使用 NSMutableDictionary。NSMutableDictionary 是 NSDictionary 的子类，这意味着可以像使用 NSDictionary 那样使用 NSMutableDictionary。但 NSMutableDictionary 提供了额外的一些方法，可以在字典中添加、删除和插入对象。

说明

首先需要实例化 NSMutableDictionary 类，可以通过任何构造函数做到这一点。要想创建新的、空的 NSMutableDictionary 对象，只需要使用 alloc 与 init 即可：

```
NSMutableDictionary *dictionary = [[NSMutableDictionary alloc] init];
```

要想向字典中添加对象，需要向字典发送 setObject:forKey:消息，同时带上要添加的对象以及与对象关联的键：

```
[dictionary setObject:@"Hello World"
              forKey:@"english"];
[dictionary setObject:@"Bonjour tout le monde"
              forKey:@"french"];
[dictionary setObject:@"Hola Mundo
              forKey:@"spanish"];
```

在使用 setObject:forKey:时，你向字典中添加的对象一定是由你提供的键索引的。

要想从字典中删除对象，就必须拥有与对象匹配的键。如果拥有键，那么可以通过

removeObjectForKey:方法删除对象：

```
[dictionary removeObjectForKey:@"french"];
```

最后，可以通过 removeAllObjects:方法一次性删除字典中的全部对象。参见程序清单 3-18。

代码

程序清单 3-18 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSMutableDictionary *dictionary = [[NSMutableDictionary alloc] init];
        [dictionary setObject:@"Hello World"
                       forKey:@"english"];
        [dictionary setObject:@"Bonjour tout le monde"
                       forKey:@"french"];
        [dictionary setObject:@"Hola Mundo"
                       forKey:@"spanish"];
        NSLog(@"OBJECTS ADDED TO DICTIONARY: %@", dictionary);
        [dictionary removeObjectForKey:@"french"];
        NSLog(@"OBJECT REMOVED FROM DICTIONARY: %@", dictionary);
        [dictionary removeAllObjects];
        NSLog(@"ALL OBJECTS REMOVED FROM DICTIONARY: %@", dictionary);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。通过日志控制台，可以查看在应用每个操作后字典都发生了哪些变化：

```
OBJECTS ADDED TO DICTIONARY: {
    english = "Hello World";
    french = "Bonjour tout le monde";
    spanish = "Hola Mundo";
}
OBJECT REMOVED FROM DICTIONARY: {
    english = "Hello World";
    spanish = "Hola Mundo";
}
ALL OBJECTS REMOVED FROM DICTIONARY: {
}
```

3.15　将字典保存到文件系统中

问题

你想将字典中的对象保存到文件系统中以供日后或是其他程序使用。

解决方案

如果字典中包含数字或字符串对象的列表，那么可以将所有这些对象保存到文件系统中以供后续使用，使用 writeToFile:atomically:方法可以做到这一点。注意，该方法无法处理自定义对象。

说明

该攻略需要创建如下字典，其中的内容是与键匹配的短语：

```
NSArray *listOfObjects = [NSArray arrayWithObjects:@"Hello World", ➥
   @"Bonjour tout le monde", @"Hola Mundo", nil];
NSArray *listOfKeys = [NSArray arrayWithObjects:@"english", @"french",➥
   @"spanish", nil];
NSDictionary *dictionary = [NSDictionary dictionaryWithObjects:listOfObjects
                                                       forKeys:listOfKeys];
```

要想将字典保存到文件系统中，首先需要获得如下文件引用：

```
NSString *filePathName = @"/Users/Shared/dictionary.txt";
```

注意：

本攻略假设你使用的是 Mac 应用。iOS 文件引用的工作方式与此不同，参见 2.5 节以了解如何获得 iOS 文件引用。

现在可以使用 writeToFile:atomically:方法将字典内容写到 Mac 文件系统中：

```
[dictionary writeToFile:filePathName
             atomically:YES];
```

参见程序清单 3-19。

代码

程序清单 3-19 main.m

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

int main (int argc, const char * argv[])
{
```

```
    @autoreleasepool {
        NSArray *listOfObjects = [NSArray arrayWithObjects:@"Hello World",
            @"Bonjour tout le monde", @"Hola Mundo", nil];
        NSArray *listOfKeys = [NSArray arrayWithObjects:@"english",
            @"french", @"spanish", nil];
        NSDictionary *dictionary = [NSDictionary
            dictionaryWithObjects:listOfObjects forKeys:listOfKeys];
        NSString *filePathName = @"/Users/Shared/dictionary.txt";
        [dictionary writeToFile:filePathName
                    atomically:YES];
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。使用 Finder 找到创建的文件，位于/Users/Shared/dictionary.txt。下面列出文本文件的内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
       <key>english</key>
       <string>Hello World</string>
       <key>french</key>
       <string>Bonjour tout le monde</string>
       <key>spanish</key>
       <string>Hola Mundo</string>
</dict>
</plist>
```

数据以 XML 格式组织为属性列表(一种存储键数据的 Objective-C 格式)。

3.16 从文件系统读取字典

问题

有一些应用可以访问到的文件，其中包含了以字典形式组织的内容，你想在应用中使用这些内容。

解决方案

如果有了通过字典创建的文件(使用 writeToFile:atomically:方法保存)，那么可以使用

initWithContentsOfFile:构造函数实例化新的字典，新字典是由文件内容构成的。

说明

本攻略使用 3.15 节中的文件(已将字典的内容保存到文件系统中)。因此，这里会使用相同的文件路径名：

```
NSString *filePathName = @"/Users/Shared/dictionary.txt";
```

注意：

本攻略假设你使用的是 Mac 应用。iOS 文件引用的工作方式与此不同，参见 2.5 节以了解如何获得 iOS 文件引用。

接下来就可以使用 initWithContentsOfFile:构造函数创建由文件内容构成的新字典：

```
NSDictionary *dictionary = [[NSDictionary alloc] ➥
    initWithContentsOfFile:filePathName];
```

参见程序清单 3-20。

代码

程序清单 3-20 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSString *filePathName = @"/Users/Shared/dictionary.txt";
        NSDictionary *dictionary = [[NSDictionary alloc] ➥
            initWithContentsOfFile:filePathName];
        NSLog(@"dictionary: %@", dictionary);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。可通过日志查看字典的内容：

```
dictionary: {
    english = "Hello World";
    french = "Bonjour tout le monde";
    spanish = "Hola Mundo";
}
```

3.17 创建集合

问题

应用需要在未排序的集合中存储对象。

解决方案

可以通过两个 Objective-C Foundation 类——NSSet 与 NSMutableSet 来创建集合。如果不需要修改集合，就使用 NSSet；如果在后面要向集合中添加或删除对象，就使用 NSMutableSet。

说明

在 Objective-C 中，集合的创建与其他对象一样：可以使用 alloc 与 init 构造函数，或者使用 setWithObjects:等便捷函数创建集合。如果使用 NSSet 创建集合，那么集合一旦创建完毕后，就无法再进行修改。使用 NSMutableSet 创建的集合可以在后面进行修改。

在下面的示例中，创建的集合包含了不同语言版本的 Hello World：

```
NSSet *set = [NSSet setWithObjects:@"Hello World", @"Bonjour tout le monde",
   @"Hola Mundo", nil];
```

NSSet 构造函数 setWithObjects:需要使用以 nil 结尾的数组，数组里面的对象会出现在集合中。

如果使用 NSMutableSet，那么可以通过相同的构造函数创建集合(NSMutableSet 是 NSSet 的子类)。如果后面还要向集合中添加对象，那么还可以通过 alloc 与 init 创建 NSMutableSet。表 3-4 完整列出了 NSSet 与 NSMutableSet 类的构造函数，程序清单 3-21 列出了相关代码。

表 3-4 NSSet 与 NSMutableSet 类的构造函数

构 造 函 数	说 明
`- (id)initWithObjects:(const id *)objects count:(NSUInteger)cnt;`	使用指定的对象与数量创建集合
`- (id)initWithObjects:(id)firstObj, ... NS_REQUIRES_NIL_TERMINATION;`	使用指定的以 nil 结尾的对象列表创建集合
`- (id)initWithSet:(NSSet *)set;`	使用另一个集合创建集合
`- (id)initWithSet:(NSSet *)set copyItems:(BOOL)flag;`	使用另一个集合创建集合，还可以创建每个对象的副本
`- (id)initWithArray:(NSArray *)array;`	使用指定的对象创建集合

代码

程序清单 3-21 main.m

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

int main (int argc, const char * argv[])
{

    @autoreleasepool {
        NSSet *set = [NSSet setWithObjects:@"Hello World", @"Bonjour tout le
           monde", @"Hola Mundo", nil];
        NSLog(@"set: %@",set);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。可以通过日志查看打印出的集合的全部内容：

```
set: {(
    "Bonjour tout le monde",
    "Hello World",
    "Hola Mundo"
)}
```

3.18　获取集合中元素的数量

问题

应用使用了集合中的内容，你需要知道集合中有多少元素以便能恰当地显示出内容。

解决方案

NSSet 对象提供了 count 属性，可以通过该属性获得集合中元素的数量。

说明

要想使用 count 属性，可以对集合对象使用点符号(set.count)或是发送 count 消息([set count])，从而获得集合中元素的数量。参见程序清单 3-22。

代码

程序清单 3-22 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSSet *set = [NSSet setWithObjects:@"Hello World", @"Bonjour tout le
           monde", @"Hola Mundo", nil];
        NSUInteger count = set.count;
        NSLog(@"The set contains %lu items", count);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。日志消息会显示出元素的数量：

```
The set contains 3 items
```

3.19 比较集合

问题

应用中使用了很多集合，你想要了解关于每个集合的更多信息以及每个集合中都有哪些对象。

解决方案

NSSet 提供了用于比较集合的某些内置方法。可以判断两个集合是否相交(它们有一些共同元素)，可以判断某个集合是否是另一个集合的子集(构成这个集合的对象全部属于另一个集合)，还可以判断两个集合是否相等以及某个对象是否位于集合中。

说明

本攻略需要用到两个集合，这两个集合只包含由字母表中的字母构成的字符串对象：

```
NSSet *set1 = [NSSet setWithObjects:@"A", @"B", @"C", @"D", @"E", nil];
NSSet *set2 = [NSSet setWithObjects:@"D", @"E", @"F", @"G", @"H", nil];
```

如果想判断这两个集合是否包含重叠的对象(集合相交)，那么可以使用 intersectsSet:函数来返回布尔值：

```
BOOL setsIntersect = [set1 intersectsSet:set2];
```

要想判断某个集合包含的对象是否全部位于另一个集合中，可以使用 isSubsetOfSet:函数：

```
BOOL set2IsSubset = [set2 isSubsetOfSet:set1];
```

要想测试两个集合是否相等，可以使用 isEqualToSet:函数：

```
BOOL set1IsEqualToSet2 = [set1 isEqualToSet:set2];
```

最后，如果想知道某个对象是否位于集合中，可以使用 containsObject:函数：

```
BOOL set1ContainsD = [set1 containsObject:@"D"];
```

参见程序清单 3-23。

代码

程序清单 3-23 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSSet *set1 = [NSSet setWithObjects:@"A", @"B", @"C", @"D", @"E", nil];
        NSSet *set2 = [NSSet setWithObjects:@"D", @"E", @"F", @"G", @"H", nil];
        NSLog(@"set1 contains:%@", set1);
        NSLog(@"set2 contains:%@", set2);
        BOOL setsIntersect = [set1 intersectsSet:set2];
        BOOL set2IsSubset = [set2 isSubsetOfSet:set1];
        BOOL set1IsEqualToSet2 = [set1 isEqualToSet:set2];
        BOOL set1ContainsD = [set1 containsObject:@"D"];
        NSLog(@"setsIntersect = %i, set2IsSubset = %i, set1IsEqualToSet2 = %i,
          set1ContainsD = %i", setsIntersect, set2IsSubset,
                    set1IsEqualToSet2, set1ContainsD);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。日志消息会显示出集合与各种测试的结果。如果布尔值是 YES，日志会打印出 1；如果布尔值是 NO，日志会打印出 0：

```
set1 contains:{(
    A,
    D,
    B,
```

```
        E,
        C
    )}
    set2 contains:{(
        H,
        F,
        D,
        G,
        E
    )}
    setsIntersect = 1, set2IsSubset = 0, set1IsEqualToSet2 = 0, set1ContainsD = 1
```

3.20 遍历集合

问题

你有了对象集合，现在想要对集合中的每个对象发送相同的消息或是访问同样的属性。

解决方案

使用 NSSet 函数 allObjects:将集合转换为数组，接下来就可以使用 for-each 循环了，此外还可以使用 enumerateObjectsUsingBlock:来处理集合中的每个对象。NSSet 还支持 makeObjectsPerformSelector:，如果你希望每个对象只执行一个方法，那么这个方法会非常适合。

说明

如果想要使用 3.4 节中介绍的方式，那么可以临时将集合内容转换为数组。例如，要想使用 for-each 循环遍历集合中的对象，可以通过如下代码实现：

```
for (NSString *s in [set allObjects]) {
    NSLog(@"value: %@", s);
}
```

还可以使用块，通过 enumerateObjectsUsingBlock:方法针对集合中的每个对象执行代码。可以用来定义代码块，然后应用到集合中的每个对象，同时又不必创建 for-each 循环或是获得数组版本的集合引用：

```
[set enumerateObjectsUsingBlock:^(id obj, BOOL *stop) {
    NSLog(@"obj = %@", obj);
}];
```

如果只想对每个对象执行单个动作(动作是方法，位于对象的类定义中)，那么可以使用 makeObjectsPerformSelector:方法：

```
[set makeObjectsPerformSelector:@selector(description)];
```

参见程序清单 3-24。

代码

程序清单 3-24 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSSet *set = [NSSet setWithObjects:@"Hello World", @"Bonjour tout le
           monde", @"Hola Mundo", nil];
        for (NSString *s in [set allObjects]) {
            NSLog(@"value: %@", s);
        }

        [set enumerateObjectsUsingBlock:^(id obj, BOOL *stop) {
            NSLog(@"obj = %@", obj);
        }];
        [set makeObjectsPerformSelector:@selector(description)];
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。日志消息会显示出使用各种方式遍历集合后的结果：

```
value: Bonjour tout le monde
value: Hello World
value: Hola Mundo
obj = Bonjour tout le monde
obj = Hello World
obj = Hola Mundo
```

3.21　操纵集合内容

问题

你希望集合内容能够更具动态性，这样用户或你就可以在集合中添加和删除对象了。然而，NSSet 是不可变类，因此一旦创建 NSSet 对象，就无法再修改其中的内容。

解决方案

如果知道集合会动态变化，那么请使用 NSMutableSet。NSMutableSet 是 NSSet 的子类，这意味着可以像使用 NSSet 那样使用 NSMutableSet，但 NSMutableSet 提供了额外的一些方法，可以在集合中添加和删除对象。

说明

首先需要实例化 NSMutableSet 类，可以通过任何构造函数做到这一点。要想创建新的、空的 NSMutableSet 对象，只需要使用 alloc 与 init 即可：

```
NSMutableSet *set = [[NSMutableSet alloc] init];
```

要想向集合中添加对象，需要向集合发送 addObject:消息，同时将要添加的对象作为参数传递进去：

```
[set addObject:@"Hello World"];
[set addObject:@"Bonjour tout le monde"];
[set addObject:@"Hola Mundo"];
```

要想从集合中删除对象，就必须拥有对象引用。如果拥有对象引用，那么可以通过 removeObject:方法删除对象：

```
[set removeObject:@"Bonjour tout le monde"];
```

最后，可以通过 removeAllObjects:方法一次性删除集合中的全部对象。参见程序清单 3-25。

代码

程序清单 3-25 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSMutableSet *set = [[NSMutableSet alloc] init];
        [set addObject:@"Hello World"];
        [set addObject:@"Bonjour tout le monde"];
        [set addObject:@"Hola Mundo"];
        NSLog(@"Objects added to set:%@", set);
        [set removeObject:@"Bonjour tout le monde"];
        NSLog(@"Object removed from set:%@", set);
        [set removeAllObjects];
        NSLog(@"All objects removed from set:%@", set);
    }
```

```
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。通过控制台，可以查看在应用每个操作后集合都发生了哪些变化：

```
Objects added to set:{(
    "Bonjour tout le monde",
    "Hello World",
    "Hola Mundo"
)}
Object removed from set:{(
    "Hello World",
    "Hola Mundo"
)}
All objects removed from set:{(
)}
```

第5章

使用日期、时间与定时器

本章介绍如何通过 Foundation 框架使用 Objective-C 处理日期与定时器。

本章内容：

- 使用 NSDate 创建今天的日期
- 使用 NSDateComponents 创建自定义日期
- 比较日期
- 将字符串转换为日期
- 格式化日期以在用户界面上显示
- 加减日期
- 使用定时器调度重复与非重复的代码

5.1 创建表示今天的日期对象

问题

应用中需要表示今天的日期。

解决方案

使用 NSDate 类的 date 方法创建表示当前日期的日期对象。

说明

NSDate 类通常与其他类搭配使用(接下来的攻略将会介绍这一点)。NSDate 本身能获取今天的日期，可以将日期打印到控制台或是呈现给用户。要想获得今天的日期，请使用 date

方法并将结果赋给 NSDate 对象：

```
NSDate *todaysDate = [NSDate date];
```

参见程序清单 5-1。

代码

程序清单 5-1 main.m

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

int main (int argc, const char * argv[])
{
   @autoreleasepool {
        NSDate *todaysDate = [NSDate date];
        NSLog(@"Today's date is %@", todaysDate);
}
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。通过日志查看打印的今天的日期：

```
Today's date is 2012-06-27 13:14:30 +0000
```

5.2 通过 Component 创建自定义日期

问题

除了当前日期外，应用还需要引用其他日期。

解决方案

使用 NSDateComponents 定义具体日期，然后使用 NSCalendar 和日期组件返回 NSDate 对象引用，在应用中可以使用 NSDate 对象引用。

说明

要想创建自定义日期，需要使用 3 个 Foundation 类：NSDate、NSDateComponents 与 NSCalendar。这里，NSDate 作为表示日期的最基本的类。

NSDateComponents 类代表构成日期与时间的细节信息：天、月、年与小时。NSDate-Components 提供了很多关于日期与时间的细节信息，可以设置这些信息来自定义日期。

NSCalendar 类表示真实世界的日历，用于管理与日历相关的复杂操作。可以指定使用

的日历或是获取用户系统中采用的日历。通常情况下，可以假定使用的是 Gregorian 日历，但也可以指定其他日历，比如 Hebrew 或 Islamic 日历。

要想创建自定义日期，首先需要新建 NSDateComponents 实例：

```
NSDateComponents *dateComponents = [[NSDateComponents alloc] init];
```

接下来需要设置自定义日期中需要的所有属性。本攻略会设置用于表示美国加利福尼亚州 iPhone 发布起始日期的所有组件：

```
dateComponents.year = 2007;
dateComponents.month = 6;
dateComponents.day = 29;
dateComponents.hour = 12;
dateComponents.minute = 01;
dateComponents.second = 31;
dateComponents.timeZone = [NSTimeZone timeZoneWithAbbreviation:@"PDT"];
```

你只需要使用点符号设置感兴趣的日期属性。上面的最后那个属性需要特殊的 NSTimeZone 对象，可以指定所需的任何时区，也可以不设置该属性以使用系统时区。

最后，要想创建 NSDate 对象，需要有指向日历的引用(通常是当前的系统日历)，可通过 currentCalendar 消息[NSCalendar currentCalendar]获得日历引用。接下来，使用 dateWithComponents:函数获得与设置相对应的日期对象：

```
SDate *iPhoneReleaseDate = [[NSCalendar currentCalendar] ➥
    dateFromComponents:dateComponents];
```

参见程序清单 5-2。

代码

程序清单 5-2 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSDateComponents *dateComponents = [[NSDateComponents alloc] init];
        dateComponents.year = 2007;
        dateComponents.month = 6;
        dateComponents.day = 29;
        dateComponents.hour = 12;
        dateComponents.minute = 01;
        dateComponents.second = 31;
        dateComponents.timeZone = [NSTimeZone timeZoneWithAbbreviation:@"PDT"];

        NSDate *iPhoneReleaseDate = [[NSCalendar currentCalendar] ➥
            dateFromComponents:dateComponents];
```

```
        NSLog(@"The original iPhone went on sale: %@", iPhoneReleaseDate);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。可以通过日志看到打印出的 iPhone 发布日期：

```
The original iPhone went on sale: 2007-06-29 19:01:31 +0000
```

5.3 比较两个日期

问题

应用中至少有两个日期，你想知道它们之间的关系。例如，某个日期是否在另一个日期之前？两个日期之间相差多少天？

解决方案

对于简单的比较来说，可以使用内置的 NSDate 比较函数。要想知道从某个日期开始经过了多少天，需要有指向系统日历的引用和两个日期。

说明

本攻略假定你使用 5.2 节中设置的 iPhone 发布日期，并将之与今天的日期进行比较。可以通过 NSDate 的 date 函数来获得今天的日期。

首先比较 iPhone 发布日期是否是今天，请使用 isEqualToDate:函数并将需要比较的日期作为参数传递进来，该函数返回一个布尔值：

```
NSDate *todaysDate = [NSDate date];

if([todaysDate isEqualToDate:iPhoneReleaseDate])
    NSLog(@"The iPhone was released today!");
else
    NSLog(@"The iPhone was released on some other date");
```

要想判断某个日期是否早于另一个日期，请使用 earlierDate:函数并将另一个日期作为参数传递进来，该函数返回更早的那个日期：

```
NSDate *earlierDateIs = [todaysDate earlierDate:iPhoneReleaseDate];
```

还可以进行相反比较，判断哪个日期更晚：

```
NSDate *laterDateIs = [todaysDate laterDate:iPhoneReleaseDate];
```

要想获得两个日期之间相差的秒数，请使用 timeIntervalSinceDate:函数并将第二个日期作为参数传递进来。你会得到一个双精度浮点数，其值为两个日期间相差的秒数。这是个名为 NSTimeInterval(你会发现其他的日期方法使用了 NSTimeInterval)的 typedef 声明。

可以通过系统日历与 NSDateComponents 类获得关于日期比较的更详细信息，这会以你需要的格式返回两个日期间相差的时间。因此，如果想要知道天数、小时数、分钟数、年数、月数或是其中的组合，那么可以从中受益。

首先需要获得指向用户系统日历的引用：

```
NSCalendar *systemCalendar = [NSCalendar currentCalendar];
```

接下来，通过对 NSCalendar 常量进行按位或运算来指定采用的时间单位：

```
unsigned int unitFlags = NSYearCalendarUnit | NSMonthCalendarUnit |
   NSDayCalendarUnit;
```

注意：

位操作是处理底层二进制信息的一种方式。如你所知，计算机以一系列 1 和 0 来表示信息(比如 00000011 表示数字 3)。位操作会比较两部分信息的二进制表示并根据比较得到结果。按位或表示如果两个信息中有一个为 1，那么结果就为 1。

换句话说，我想知道两个日期间相差的年数、月数与天数。表 5-1 列出了这里可以使用的常量。

表 5-1　NSCalendar 常量

常　量	说　明
NSEraCalendarUnit	指定纪元
NSYearCalendarUnit	指定年
NSMonthCalendarUnit	指定月
NSDayCalendarUnit	指定天
NSHourCalendarUnit	指定小时
NSMinuteCalendarUnit	指定分钟
NSSecondCalendarUnit	指定秒
NSWeekCalendarUnit	指定周
NSWeekdayCalendarUnit	指定工作日

可以通过 NSCalendar 的 components:fromDate:toDate:options:函数来返回 NSDateComponents 对象，对象中的数据描述了根据指定的 NSCalendar 常量，计算得出的两个日期间的时间差：

```
NSDateComponents *dateComparisonComponents = [systemCalendar
           components:unitFlags
            fromDate:iPhoneReleaseDate
              toDate:todaysDate
  options:NSWrapCalendarComponents];
```

可以访问相应的属性以获得所需的信息。例如，可以通过 dateComparisonComponents.year 属性获得年数。参见程序清单 5-3。

代码

程序清单 5-3 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSDateComponents *dateComponents = [[NSDateComponents alloc] init];
        dateComponents.year = 2007;
        dateComponents.month = 6;
        dateComponents.day = 29;
        dateComponents.hour = 12;
        dateComponents.minute = 01;
        dateComponents.second = 31;
        dateComponents.timeZone = [NSTimeZone ➥
           timeZoneWithAbbreviation:@"PDT"];

        NSDate *iPhoneReleaseDate = [[NSCalendar currentCalendar] ➥
            dateFromComponents:dateComponents];
        NSLog(@"The original iPhone went on sale: %@", iPhoneReleaseDate);
        NSDate *todaysDate = [NSDate date];
        NSLog(@"Today's date is: %@", todaysDate);

        if([todaysDate isEqualToDate:iPhoneReleaseDate])
           NSLog(@"The iPhone was released today!");
        else
           NSLog(@"The iPhone was released on some other date");
        NSDate *earlierDateIs = [todaysDate earlierDate:iPhoneReleaseDate];
        NSLog(@"The earlier date is: %@", earlierDateIs);
        NSDate *laterDateIs = [todaysDate laterDate:iPhoneReleaseDate];

        NSLog(@"The later date is: %@", laterDateIs);
        NSTimeInterval timeBetweenDates = [todaysDate➥
            timeIntervalSinceDate:iPhoneReleaseDate];
        NSLog(@"The iPhone was released %f seconds ago", timeBetweenDates);
        NSCalendar *systemCalendar = [NSCalendar currentCalendar];
        unsigned int unitFlags = NSYearCalendarUnit | NSMonthCalendarUnit|➥
             NSDayCalendarUnit;
        NSDateComponents *dateComparisonComponents = ➥
        [systemCalendar components:unitFlags
                          fromDate:iPhoneReleaseDate
                            toDate:todaysDate
                           options:NSWrapCalendarComponents];
```

```
        NSLog(@"The iPhone was released %ld years, %ld months and %ld days ago",
              dateComparisonComponents.year,
              dateComparisonComponents.month,
              dateComparisonComponents.day
              );

    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。通过日志消息查看日期与它们之间的比较结果：

```
The original iPhone went on sale: 2007-06-29 19:01:31 +0000
Today's date is: 2012-06-27 20:54:56 +0000
The earlier date is: 2007-06-29 19:01:31 +0000
The later date is: 2012-06-27 20:54:56 +0000
The iPhone was released on some other date
The iPhone was released 143776405.074785 seconds ago
The iPhone was released 4 years, 6 months and 20 days ago
```

注意，你的输出消息与这里显示的不同，因为你运行上述代码的时间和我的不同。

5.4　将字符串转换为日期

问题

你通过字符串文件获得了包含日期信息的字符串，想要以日期对象的形式使用日期信息。

解决方案

使用 NSDateFormatter 指定字符串格式并创建新的日期对象。

说明

本攻略假设你有以字符串形式存储的日期信息：

```
NSString *dateString = @"02/14/2012";
```

你首先需要有日期格式化器，可以使用 NSDateFormatter 类进行创建：

```
NSDateFormatter *df = [[NSDateFormatter alloc] init];
```

接下来，使用字符串格式设置 dateFormat 属性：

```
df.dateFormat = @"MM/dd/yyyy";
```

注意：

日期格式化器使用了 Unicode 日期格式。参见 http://unicode.org/reports/tr35/-tr35-10.html#Date_Format_Patterns 以了解可用日期格式的完整列表。

要想创建日期对象，请使用 dateFromString:日期格式化函数：

```
NSDate *valentinesDay = [df dateFromString:dateString];
```

参见程序清单 5-4。

代码

程序清单 5-4 main.m

```
import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSString *dateString = @"02/14/2012";
        NSDateFormatter *df = [[NSDateFormatter alloc] init];
        df.dateFormat = @"MM/dd/yyyy";
        NSDate *valentinesDay = [df dateFromString:dateString];
        NSLog(@"Valentine's Day = %@", valentinesDay);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。通过日志消息查看从字符串创建的日期对象：

```
Valentine's Day = 2012-02-14 05:02:00 +0000
```

根据本地时区的不同，你的输出结果可能与我的也会不同。

5.5 格式化日期以便显示

问题

你希望能以用户可以识别并且在用户界面上呈现良好的方式显示日期对象。

解决方案

使用 NSDateFormatter 创建日期格式化器并将日期对象格式化为字符串以显示给用户。

说明

使用 5.4 节中介绍的 Unicode 日期格式指定日期格式化器。因此，如果已经有了 5.4 节中的日期，但却想要以不同的格式展现给用户，那么可以像下面这样设置日期格式化器：

```
df.dateFormat = @"EEEE, MMMM d";
```

这会显示出日期所处的工作日名称、月份名称与天数。
请使用 NSDateFormatter 的 stringFromDate:函数查看结果：

```
SLog(@"Another Formatted Valentine's Day = %@", [df➥
     stringFromDate:valentinesDay]);
```

这会以下面的形式显示出日期：

```
Tuesday, February 14
```

参见程序清单 5-5。

代码

程序清单 5-5 main.m

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

int main (int argc, const char * argv[])
{

    @autoreleasepool {
        NSString *dateString = @"02/14/2012";
        NSDateFormatter *df = [[NSDateFormatter alloc] init];
        df.dateFormat = @"MM/dd/yyyy";
        NSDate *valentinesDay = [df dateFromString:dateString];
        NSLog(@"Unformatted Valentine's Day = %@", valentinesDay);
        NSLog(@"Formatted Valentine's Day = %@", [df➥
            stringFromDate:valentinesDay]);
        df.dateFormat = @"EEEE, MMMM d";
        NSLog(@"Another Formatted Valentine's Day = %@", ➥
            [df stringFromDate:valentinesDay]);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。通过日志消息查看格式化之后的日期对象：

```
Unformatted Valentine's Day = 2012-02-14 05:00:00 +0000
Formatted Valentine's Day = 02/14/2012
Another Formatted Valentine's Day = Tuesday, February 14
```

5.6 加减日期

问题

你想要加减应用中的日期。

解决方案

使用 NSDateComponents、NSCalendar 类以及日期对象来加减日期。NSDateComponents 指定了时间长度(一天、一周或是其他时间间隔)。而借助 NSCalendar 提供的方法，我们可以通过用户的日历与创建日期组件对象时指定的信息来新建日期。

说明

我们继续使用 5.4 节中创建的情人节日期。对于该例来说，你想要获得情人节前一周的日期(或许作为购物时间)。

你首先需要有 NSDateComponents 对象。alloc 与 init 构造函数用于创建这种对象：

```
NSDateComponents *weekBeforeDateComponents = [[NSDateComponents alloc] init];
```

要想使用时间间隔，可以设置所需的任何属性。对于该例来说，你只想减掉一周，因此将 NSDateComponents 对象的 week 属性设为−1：

```
weekBeforeDateComponents.week = -1;
```

现在可以通过用户的日历与 dateByAddingComponents:toDate:options:函数获得前一周的日期：

```
NSDate *vDayShoppingDay = [[NSCalendar currentCalendar] ➥
  dateByAddingComponents:weekBeforeDateComponents
                  toDate:valentinesDay
                 options:0];
```

该函数返回代表前一周的新日期。此外，注意为了减掉日期，你在使用该函数时用到了负整数(并不存在 dateBySubtractingComponents 函数)，参见程序清单 5-6。

代码

程序清单 5-6 main.m

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

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSString *dateString = @"02/14/2012";
        NSDateFormatter *df = [[NSDateFormatter alloc] init];
        df.dateFormat = @"MM/dd/yyyy";
        NSDate *valentinesDay = [df dateFromString:dateString];
        NSLog(@"Valentine's Day = %@", valentinesDay);
        NSDateComponents *weekBeforeDateComponents = [[NSDateComponents
            alloc] init];
        weekBeforeDateComponents.week = -1;
        NSDate *vDayShoppingDay = [[NSCalendar currentCalendar]
        dateByAddingComponents:weekBeforeDateComponents
                        toDate:valentinesDay
                       options:0];
        NSLog(@"Shop for Valentine's Day by %@", vDayShoppingDay);
    }
    return 0;
}
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。通过控制台查看日期相减之后的结果：

```
Valentine's Day = 2012-02-14 05:00:00 +0000
Shop for Valentine's Day by 2012-02-07 05:00:00 +0000
```

5.7　使用定时器调度并重复执行任务

问题

应用需要调度代码以在特定的时间执行。此外，你还想要重复执行任务。

解决方案

使用 NSTimer 调度代码以在特定的时间执行。为了使用 NSTimer，你需要有日期对象与指向应用的运行循环的引用。

注意：

NSTimer 需要有运行循环，如果想在 Mac 或 iOS 应用中使用定时器，就必须有运行循环。本攻略需要应用带有运行循环。1.11 与 1.12 节分别介绍了创建 Mac 与 iOS 应用的步骤。

说明

本攻略的代码位于应用委托中。通常情况下，定时器会放在自定义类或是应用控制器中。

定时器会从特定的日期与时间开始向对象发送消息。如果应用需要重复，那么定时器可能会间隔一段时间后再发送消息。你首先需要有日期对象，用来表示定时器开始向对象发送消息的日期与时间：

```
NSDate *scheduledTime = [NSDate dateWithTimeIntervalSinceNow:10.0];
```

上述调度时间位于上面这一行代码执行后的 10 秒钟。可以在这里使用任何日期。

接下来，通过 initWithFireDate:interval:target:selector:userInfo:repeats:构造函数创建定时器：

```
SString *customUserObject = @"To demo userInfo";

NSTimer *timer = [[NSTimer alloc] initWithFireDate:scheduledTime
                                          interval:2
                                            target:self
                                          selector:@selector(task)
                                          userInfo:customUserObject
                                           repeats:YES];
```

这里有些内容需要说明一下。第 1 个参数是日期对象，指定了定时器何时变成活动状态。接下来是间隔时间，是定时器再次发送消息前所需等待的秒数。之后是目标参数描述符，目标是方法所处的对象。selector 参数是位于圆括号中的方法名，前面是@selector 关键字。由于方法与定时器一样都位于应用委托中，因此可以使用 self 关键字。

userInfo 是定时器使用的自定义内容。可以使用任何对象，并且可以获得正在执行的消息中的对象引用(上面的 selector 参数)。这里使用了字符串，但通常会使用字典或其他集合以支持更加复杂的活动。

repeats 参数表示定时器是发送一次消息，还是根据第 2 个参数指定的时间间隔重复发送。

接下来需要指向运行循环的引用。可以通过 NSRunLoop 的 currentRunLoop 函数获得该引用：

```
NSRunLoop *runLoop = [NSRunLoop currentRunLoop];
```

现在，只需要将定时器添加到运行循环中即可：

```
[runLoop addTimer:timer forMode:NSDefaultRunLoopMode];
```

10 秒钟后，定时器将会开始每隔两秒钟向应用发送 task 消息。

启动之后，如果想停止定时器，可以向定时器发送 invalidate 消息。这会从运行循环中

删除定时器，代码如下所示：

```
[timer invalidate];
```

参见程序清单 5-7。

代码

程序清单 5-7 main.m

```
#import "AppDelegate.h"

@implementation AppDelegate
@synthesize window = _window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification{
    NSDate *scheduledTime = [NSDate dateWithTimeIntervalSinceNow:10.0];
    NSString *customUserObject = @"To demo userInfo";
    NSTimer *timer = [[NSTimer alloc] initWithFireDate:scheduledTime
                                              interval:2
                                                target:self
                                              selector:@selector(task)
                                              userInfo:customUserObject
                                               repeats:YES];
    NSRunLoop *runLoop = [NSRunLoop currentRunLoop];
    [runLoop addTimer:timer forMode:NSDefaultRunLoopMode];
}

-(void)task:(id)sender{
    NSTimer *localTimer = (NSTimer *)sender;
    NSLog(@"Schedule task has executed with this user info: %@", [localTimer
        userInfo]);
}

@end
```

使用

要想使用上述代码，请从 Xcode 构建并运行 Mac 应用。注意控制台窗口以及消息何时开始打印到日志中。此处打印出了时间戳，这样就可以看到时间间隔的运作方式了：

```
2012-01-19 15:23:28.651 Timer[31067:707] Schedule task has executed with
this user info: To demo userInfo
2012-01-19 15:23:30.651 Timer[31067:707] Schedule task has executed with
this user info: To demo userInfo
2012-01-19 15:23:32.651 Timer[31067:707] Schedule task has executed with
this user info: To demo userInfo
```

第 7 章

使用 Web 服务

本章将介绍如何通过 Objective-C 使用 Web 服务。

本章内容：

- 使用 NSURL 下载文件
- 通过 XML 与 JSON 使用 Web 服务
- 解析 XML 与 JSON 数据
- 通过 NSURLConnection 异步地使用 Web 服务

7.1 下载文件

问题

你想从网上下载文件。

解决方案

使用 NSURL 指定文件的 URL，然后使用 NSData 将文件的内容下载到文件系统中。

注意：

URL 代表的是统一资源定位符(Uniform Resource Locator)。URL 是字符串，指定了网上资源的位置。NSURL 是 Foundation 类，可以通过它在 Objective-C 中使用 URL。

说明

对于本攻略来说，网上必须要有可供下载的文件存在。我在自己的博客上放置了一个文本文件，该文件的 URL 是：

```
http://howtomakeiphoneapps.com/wp-content/uploads/2012/03/objective-c-recipes-
example-file.txt
```

首先要通过待下载资源的 URL 新建 NSURL 对象，使用方才提供 URL，代码如下所示：

```
NSURL *remoteTextFileURL = [NSURL
URLWithString:@"http://howtomakeiphoneapps.com/wp-
content/uploads/2012/03/objective-c-recipes-example-file.txt"];
```

接下来，使用 NSURL 对象的内容新建如下 NSData 对象：

```
NSData *remoteTextFileData = [NSData dataWithContentsOfURL:remoteTextFileURL];
```

可以在 Objective-C 程序中使用 NSData 对象(参见 4.11 节以了解如何使用 NSData)或是将其保存到文件系统中，代码如下所示：

```
[remoteTextFileData writeToFile:@"/Users/Shared/objective-c-recipes-example-file.txt"
                atomically:YES];
```

参见程序清单 7-1。

代码

程序清单 7-1 main.m

```
#import <Foundation/Foundation.h>
int main (int argc, const char * argv[]){
    @autoreleasepool {
        NSURL *remoteTextFileURL = [NSURL URLWithString:
            @"http://howtomakeiphoneapps.com/wp-content/uploads/2012/03/objective-c-
            recipes-example-file.txt"];
        NSData *remoteTextFileData = [NSData dataWithContentsOfURL:remoteTextFileURL];
        [remoteTextFileData writeToFile:@"/Users/Shared/objective-c-recipes-
         example-file.txt" atomically:YES];
    }
    return 0;
}
```

使用

要想运行上述代码，创建 Mac 命令行 Xcode 项目并将 main.m 文件的内容改为程序清单 7-1 中的代码。构建并运行该命令行应用，将文件下载到自己的 Mac 上：

```
/Users/Shared/objective-c-recipes-example-file.txt
```

如果下载成功，找到并打开文件，查看其中的内容。

7.2　通过 XML 使用 Web 服务

问题

你想将使用了 XML 数据的 Web 服务添加到应用中。

注意：

互联网公司发布 Web 服务，这样开发者就可以将他们的服务包含到自己的应用中。Web 服务的工作方式类似于 Web 浏览器。在 Web 浏览器中，你输入 Web 地址(请求)，按下回车键，等待网上远程计算机的响应。当响应返回时，Web 浏览器会使用响应中的规则与内容显示网页。Web 服务的工作方式与之相同，只不过由应用发送请求并接收响应。

互联网公司会尽自己最大努力通过标准格式来规划 Web 服务的请求与响应，使应用能够更轻松地使用他们的服务。Web 请求是字符串(类似于 Web 地址)，Web 响应是格式化为 XML 或 JSON 的字符串。后面会详细介绍 XML 与 JSON。

解决方案

根据 Web 服务发布者提供的文档规划请求字符串。根据请求字符串创建 NSURL 对象与 NSData，从 Web 服务下载响应。使用 NSXMLParser 遍历返回的 XML 文档。

说明

本攻略将会介绍如何使用 bitly 公司提供的 Web 服务。bitly 公司发布了用于缩短 URL 的 Web 服务。你只需要向 bitly 公司的 Web 服务发送请求，同时传递长 URL 及其所要求格式的 bitly 认证信息即可；bitly 会返回一个 XML 文件，其中的内容包含了缩短后的 URL。

注意：

你需要创建免费的 bitly 账户并获得自己的 API 键与用户名。请访问 https://bit.ly 以申请账户。

我打算通过基于命令行的 Mac 应用来讲解本攻略，但可以使用任何类型的项目。由于 NSXMLParser 使用了委托设计模式，因此你使用的类需要采用协议并支持委托。通过菜单 File | New File | Objective-C class 向项目中添加新类，将新类命名为 LinkShortener。

LinkShortener 的接口需要包含名为 recorderString 的 NSMutableString 前置声明，进而记录从 Web 服务获得的数据。LinkShortener 还需要有字符串以记录 XML 解析器当前查找的 XML 文件区域。将那个变量声明为 currentElement，并将 currentElement 与 recorderString 设为私有。此外，需要为用于缩短 URL 的函数定义前置声明，将函数声明为 getTheShortUrlVersionOfThisLongURL。LinkShortener 的接口如下所示：

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

```
@interface LinkShortener : NSObject{
    @private
    NSMutableString *recorderString;
    NSString *currentElement;
}
-(NSString *)getTheShortURLVersionOfThisLongURL:(NSString *)longURL;

@end
```

XML 的解析方式

首先介绍一下什么是 XML 以及 NSXMLParser 如何读取 XML 文档。XML 表示可扩展标记语言(EXtensible Markup Language)，用于存储和传输数据。XML 的使用需要通过开始和结束标签来包含数据。开始标签是通过字符<与>包围的字符，结束标签是通过字符</与>包围的字符。标签与数据总称为元素。

例如，假设存在 Person 这种 XML 数据类型，那么开始标签是<Person>，结束标签是</Person>，中间的字符就是数据。总体看起来代码如下所示：

```
<Person>Matthew J. Campbell</Person>
```

XML 标签具有描述作用，因此标签的含义是显而易见的。整个文档会有多个带有数据的标签，这些标签以层次化的结构组织。可以在其他标签数据中放置标签数据，如下所示：

```
<Person>
       <Name>Matthew J. Campbell</Name>
       <Gender>Male</Gender>
</Person>
```

NSXMLParser 会从头读取 XML 文档，一个元素接一个元素地读取，直到文档结束为止。如果 NSXMLParser 读取上面的 XML 文档，那么会从 Person 元素开始，然后是 Name 元素，接下来是 Gender 元素。这种用来解析 XML 文档的方式叫做 Simple API for XML(SAX)。

我们将会使用委托来解析从 bitly 得到的 XML 数据。由于解析器查询文档中的每个元素，因此会向委托对象 LinkShortener 发送消息，这样就可以从每个元素中提取数据了。

LinkShortener 需要作为 NSXMLParser 的委托，这意味着 LinkShortener 需要采用 NSXMLParserDelegate 协议。请将协议名放在父类 NSObject 之后：

```
#import <Foundation/Foundation.h>
@interface LinkShortener : NSObject<NSXMLParserDelegate>{
    @private
    NSMutableString *recorderString;
    NSString *currentElement;
}
-(NSString *)getTheShortURLVersionOfThisLongURL:(NSString *)longURL;

@end
```

既然采用了 NSXML<ParserDelegate>协议，你至少需要实现如下两个委托方法：parser:

didStartElement:namespaceURI:qualifiedName:attributes:与 parser:foundCharacters:。

由于应用只需要 XML 文件中某个元素的内容，因此这两个委托方法足矣。但如果有必要，可以实现 parser:didEndElement:namespaceURI:qualifiedName:方法。这样一来，当解析器遇到 XML 数据的结束标签时，就会通知你。

打开 LinkShortener.m，实现第一个委托方法：

```
- (void) parser:(NSXMLParser *)parser
didStartElement:(NSString *)elementName
   namespaceURI:(NSString *)namespaceURI
  qualifiedName:(NSString *)qName
     attributes:(NSDictionary *)attributeDict{
        currentElement = [elementName copy];
        if ([elementName isEqualToString:@"shortUrl"]) {
               recorderString = [[NSMutableString alloc] init];
        }
}
```

该委托方法会在读到每个新的 XML 元素时执行一次(大多数 XML 文件都有多个元素)。出于这个原因，我们将参数 elementName 复制一份并赋给 currentElement。你希望在其他委托方法执行时能够追踪到当前的元素。

上述代码中的另一个重要之处是 if 语句，你在这里测试当前元素是否是 shortUrl。bitly 会将缩短后的 URL 字符串放到该元素所在位置。如果遇到 shortUrl 元素，就会创建并初始化新的 NSMutableString 以记录元素中的内容。

现在可以实现下一个委托方法了。该委托方法会在每次遇到文件中 XML 元素的数据时执行。可以测试 XML 解析器是否位于 shortUrl 元素中，如果是，就将遇到的字符附加到 NSMutableString 对象 recorderString 的后面：

```
- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string{
        if ([currentElement isEqualToString:@"shortUrl"])
               [recorderString appendString:string];
}
```

以上做法会让 LinkShortener 准备开始接收 XML 数据。现在可以准备请求并将请求发送给 Web 服务器以下载响应。所有这些都位于函数 getTheShortUrlVersionOfThisLongURL:中，可以开始在 LinkShortener 实现文件中进行编码了：

```
-(NSString *)getTheShortUrlVersionOfThisLongURL:(NSString *)longURL{
}
```

首先构造请求字符串。需要根据 bitly 需要的请求字符串、longURL 参数和自己的 API 键与 API 登录来构建字符串：

```
#warning Get your API Login from https://bitly.com/a/your_api_key and put➥
it here before running
NSString *APILogin = @"[YOUR API LOGIN]";
#warning Get your API key from https://bitly.com/a/your_api_key and put it➥
here before running
```

```
NSString *APIKey = @"[YOUR API KEY]";
NSString *requestString = [[NSString alloc] initWithFormat: ➥
@"http://api.bit.ly/shorten?version=2.0.1&longUrl=%@&login=%@&apiKey=%@&format=xml",➥
 longURL, APILogin, APIKey];
```

我已经在示例代码中写了警告，这样你就知道此处应该使用自己的认证信息。此外，如果仔细查看请求字符串的第一部分，你就会发现有个 format 参数，其值是 xml(format=xml)。通过 format 参数，告诉 bitly 响应要以 XML 的格式返回。

接下来需要创建 NSURL 对象，可以根据请求字符串来创建 NSURL 对象：

```
NSURL *requestURL = [NSURL URLWithString:requestString];
```

要想下载数据，请按照 7.1 节那样使用 NSData：

```
recorderString = nil;
NSData *responseData = [NSData dataWithContentsOfURL:requestURL];
```

这里将 recorderString 设为 nil 是为了防止该函数在对象的生命周期前被使用。既然下载了数据，因而可以使用 NSXMLParser 遍历 XML 并获取到 shortUrl 元素中的数据。

为了做到这一点，使用下载的 NSData 对象实例化新的 NSXMLParser，将 NSXMLParser 对象的 delegate 设为 self，然后向 NSXMLParser 对象发送 parse 消息：

```
NSXMLParser *xmlParser = [[NSXMLParser alloc] initWithData:responseData];
xmlParser.delegate = self;
[xmlParser parse];
```

现在，XML 解析器会遍历数据并使用之前编写的委托方法获取有意义的内容。XML 解析器找到的所有内容都会记录在 recorderString 中。一旦 XML 解析器遍历完毕，你就可以将结果返回给调用者：

```
if(recorderString)
    return [recorderString copy];
else
    return nil;
```

如果找到数据，就可以通过 if 语句发送 recorderString 的副本。

最后，要想使用程序另一部分的功能，需要导入 LinkShortener 头文件，实例化 LinkShortener 对象，然后使用带有长 URL 的函数版本。下面的 main.m 展示了如何实现：

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

int main(int argc, const char * argv[]){
   @autoreleasepool {
      NSString *longURL = @"http://howtomakeiphoneapps.com/how-to-➥
        asynchronously-add-web-content-to-uitableview-in-ios/1732/";
      LinkShortener *linkShortener = [[LinkShortener alloc] init];
      NSString *shortURL = [linkShortener ➥
        getTheShortURLVersionOfThisLongURL:longURL];
```

```
        NSLog(@"shortURL = %@", shortURL);
    }
    return 0;
}
```

参见程序清单 7-2～7-4。

代码

程序清单 7-2 LinkShortener.h

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

@interface LinkShortener : NSObject<NSXMLParserDelegate>{
    @private
    NSMutableString *recorderString;
    NSString *currentElement;
}
-(NSString *)getTheShortURLVersionOfThisLongURL:(NSString *)longURL;

@end
```

程序清单 7-3 LinkShortener.m

```
#import "LinkShortener.h"

@implementation LinkShortener

-(NSString *)getTheShortURLVersionOfThisLongURL:(NSString *)longURL{
    #warning Get your API Login from https://bitly.com/ and put it here before running
    NSString *APILogin = @"[YOUR API LOGIN]";
    #warning Get your API key from https://bitly.com/ and put it here before running
    NSString *APIKey = @"[YOUR API KEY]";

    NSString *requestString = [[NSString alloc] initWithFormat: ➥
      @"http://api.bit.ly/shorten?version=2.0.1➥
      &longUrl=%@&login=%@&apiKey=%@&format=xml",➥
      longURL, APILogin, APIKey];
    NSURL *requestURL = [NSURL URLWithString:requestString];
    recorderString = nil;
    NSData *responseData = [NSData dataWithContentsOfURL:requestURL];
    NSXMLParser *xmlParser = [[NSXMLParser alloc] initWithData:responseData];
    xmlParser.delegate = self;
    [xmlParser parse];

    if(recorderString)
        return [recorderString copy];
    else
        return nil;;
}
```

```
- (void) parser:(NSXMLParser *)parser
   didStartElement:(NSString *)elementName
   namespaceURI:(NSString *)namespaceURI
   qualifiedName:(NSString *)qName
     attributes:(NSDictionary *)attributeDict{
        currentElement = [elementName copy];
        if ([elementName isEqualToString:@"shortUrl"]) {
               recorderString = [[NSMutableString alloc] init];
        }
}

- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string{
        if ([currentElement isEqualToString:@"shortUrl"])
               [recorderString appendString:string];
}

@end
```

程序清单 7-4 main.m

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

int main(int argc, const char * argv[]){
    @autoreleasepool {
        NSString *longURL = @"http://howtomakeiphoneapps.com/
          how-to-asynchronously-add-web-content-to-uitableview-in-ios/1732/";
        LinkShortener *linkShortener = [[LinkShortener alloc] init];
        NSString *shortURL = [linkShortener getTheShortURLVersionOfThisLongURL:longURL];
        NSLog(@"shortURL = %@", shortURL);
    }
    return 0;
}
```

代码

要想使用上述 URL 缩短功能，请将头文件导入到使用该功能的类中。接下来从 LinkShortener 类实例化 LinkShortener 对象。最后，要想使用该功能，请向 LinkShortener 对象发送消息 getTheShortURLVersionOfThisLongURL:并将长 URL 作为参数。如果 Web 请求成功，那么该函数会返回缩短后的 URL；如果失败，会返回 nil。下面是在控制台日志中打印出的内容：

```
shortURL = http://bit.ly/yFmJFh
```

注意：
通过测试可以发现，你从 bitly 接收到的缩短后的 URL 可能与我接收到的不太一样。

7.3　通过 JSON 使用 Web 服务

问题

你想将使用了 JSON 数据的 Web 服务添加到应用中。

注意：

JSON 是 XML 的替代方案，很多互联网公司在实现 Web 服务时都会使用 JSON。JSON 代表 JavaScript Object Notation，用于数据的存储与传输。以 REST(REpresentational State Transfer)方式实现的 Web 服务会提供 XML 与 JSON 响应数据。其他类型的 Web 服务可能只会提供其中的一种。

解决方案

与 7.2 节一样，根据 Web 服务发布者提供的文档规划请求字符串。根据请求字符串创建 NSURL 对象并使用 NSData 从 Web 服务下载响应。使用 NSJSONSerialization 解析返回的 JSON 数据。

注意：

NSJSONSerialization 在 Mac OSX 10.7 和 iOS 5.0 中才被引入进来。

说明

本攻略使用同样的 bitly Web 服务，与 7.2 节介绍的过程一样，请求并下载结果。然而，NSJSONSerialization 并没有使用委托，因此无须添加新的文件或类来实现 NSJSONSerialization 的 JSON 解析。

由于无需单独的类，因此可以将用于构建请求字符串的代码放在应用中需要使用 Web 服务的地方。如果继续使用命令行 Mac 应用，那么可以将这些代码放在 main.m 文件中。下述代码展示了如何构建请求字符串：

```
NSString *longURL = @"http://howtomakeiphoneapps.com/➥
how-to-asynchronously-add-web-content-to-uitableview-in-ios/1732/";

#warning Get your API Login from https://bitly.com/ and put it here before running
NSString *APILogin = @"[YOUR API LOGIN]";
#warning Get your API key from https://bitly.com/ and put it here before running
NSString *APIKey = @"[YOUR API KEY]";

NSString *requestString = [[NSString alloc] initWithFormat: ➥
@"http://api.bit.ly/shorten?version=2.0.1➥
&longUrl=%@&login=%@&apiKey=%@&format=json", longURL, APILogin, APIKey];
```

然后，可以使用 NSURL 与 NSData 根据请求字符串获得响应：

```
NSURL *requestURL = [NSURL URLWithString:requestString];
NSData *responseData = [NSData dataWithContentsOfURL:requestURL];
```

JSON 解析

接下来介绍如何解析 JSON。相对于 XML 使用的标签数据模式，JSON 根据两种结构来组织内容：名-值对的集合与有序的值列表。JSON 的名-值对集合遵循与 Objective-C NSDictionary 相同的模式，而 JSON 的有序值列表则对应于 Objective-C NSArray。

如果查看 JSON 文件，就会看到这些类型的结构是通过花括号而非 XML 文件的标签数据组织的。例如，7.2 节中介绍的 Person 元素的 JSON 版本如下所示：

```
{"Person":"Matthew J. Campbell","Gender":"Male"}
```

由于 JSON 数据是通过这种方式输入的，而且字典与数组结构能很自然地映射到编程语言，因此通常情况下 JSON 更易于解析。稍后就会看到，有个 Foundation 函数可以轻松将 JSON 响应的内容转换为 NSDictionary。

首先需要有 NSError 对象，在向 NSJSONSerialization 发送 JSONObjectWithData:options:error:消息时需要传递 NSError 对象：

```
NSError *error = nil;
NSDictionary *bitlyJSON = [NSJSONSerialization JSONObjectWithData:responseData
                                                          options:0
                                                            error:&error];
```

将上述函数的返回结果赋给 NSDictionary 对象，这个对象持有 JSON 数据的内容。要想获得所需内容，只需要访问字典中的各个对象即可。通常情况下，这需要引用嵌套的字典、数组与对象。需要检查响应数据以找到所需内容。下述代码展示了如何从响应数据得到 bitly 的短 URL：

```
if(!error){
    NSDictionary *results = [bitlyJSON objectForKey:@"results"];
    NSDictionary *resultsForLongURL = [results objectForKey:longURL];
    NSString *shortURL = [resultsForLongURL objectForKey:@"shortUrl"];
    NSLog(@"shortURL = %@", shortURL);
}
else{
    NSLog(@"There was an error parsing the JSON");
}
```

参见程序清单 7-5。

代码

程序清单 7-5 main.m

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

```
    @autoreleasepool {
        NSString *longURL = @"http://howtomakeiphoneapps.com/how-to-
         asynchronously-add-web-content-to-uitableview-in-ios/1732/";

         #warning Get your API Login from https://bitly.com/ and put it here
         before running
         NSString *APILogin = @"[YOUR API LOGIN]";
         #warning Get your API key from https://bitly.com/ and put it here
         before running
         NSString *APIKey = @"[YOUR API KEY]";
         NSString *requestString = [[NSString alloc] initWithFormat:
              @"http://api.bit.ly/shorten?version=2.0.1
              &longUrl=%@&login=%@&apiKey=%@&format=json",
              longURL, APILogin, APIKey];
        NSURL *requestURL = [NSURL URLWithString:requestString];
        NSData *responseData = [NSData dataWithContentsOfURL:requestURL];
        NSError *error = nil;
        NSDictionary *bitlyJSON = [NSJSONSerialization
             JSONObjectWithData:responseData
                        options:0
                          error:&error];
        if(!error){
            NSDictionary *results = [bitlyJSON objectForKey:@"results"];
            NSDictionary *resultsForLongURL = [results
               objectForKey:longURL];
            NSString *shortURL = [resultsForLongURL
                objectForKey:@"shortUrl"];
            NSLog(@"shortURL = %@", shortURL);
        }
        else{
            NSLog(@"There was an error parsing the JSON");
        }
    }
    return 0;
}
```

使用

可以在应用中的任何地方使用上述代码。如果使用 Mac 命令行应用进行测试，那么只需要将上述代码放到 main.m 文件中即可，但需要从 bitly 获得 API 键与登录。通过控制台日志窗口可以查看 Web 服务的请求结果，如下所示：

```
shortURL = http://bit.ly/yFmJFh
```

注意：

JSON 的解析步骤要比 XML 少很多，是处理 Web 服务的首选(如果可用)。然而请注意，Web 服务并非总是使用 JSON，并且只有 Mac OSX 10.7、iOS 5 以及它们的更高版本才能使用 JSON。

7.4 异步地使用 Web 服务

问题

你想以后台处理的方式使用 Web 服务，这样网络活动就不会影响到用户界面了。

解决方案

如果想要异步地使用网络或是对网络连接和 Web 请求的使用进行更多控制，那么请使用 NSURLConnection 和 NSURLRequest。

说明

首先需要有发送给 Web 服务器的请求字符串。可以使用 Web 服务的请求字符串(就像你在 7.2 与 7.3 节中所做的那样)，甚至还可以放在网页中。对于该例来说，我使用自己的博客的 RSS 源，因为我知道这里能提供一些 XML 数据供下载。

注意：

RSS 表示 Really Simple Syndication。RSS 用于发布诸如博客和播客之类的内容。RSS 文件通常都是基于 XML 的大文件，而且遵循一些附加规范，这些规范对于发布内容会起到帮助作用。向应用添加 RSS 源是向用户发布信息的一种简单方式，因为大多数博客软件都内置了 RSS 特性。

本攻略会使用 Mac Cocoa 应用，也可以使用 iOS 应用。但如果使用命令行 Mac 应用，就会遇到问题。这是因为 NSURLConnection 使用的异步方法的执行时间比 Mac 命令行应用的生命周期还要长，所以无法在控制台日志中看到结果。

代码位于 Mac Cocoa 应用的 AppDelegate.m 文件的 applicationDidFinishLaunching:委托方法中。首先需要的是请求字符串(对于该例来说，也就是笔者的博客的 RSS 源)：

```
NSString *requestString = @"http://www.howtomakeiphoneapps.com/feed/";
```

接下来，根据请求字符串构造 NSURL 对象：

```
NSURL *requestURL = [NSURL URLWithString:requestString];
```

使用 requestURL 对象实例化新的 NSURLRequest：

```
NSURLRequest *request = [[NSURLRequest alloc] initWithURL:requestURL
                    cachePolicy:NSURLRequestReload IgnoringLocalCacheData ➥
                timeoutInterval:10];
```

还可以在这里指定请求如何处理缓存与超时间隔。接下来需要创建 NSOperationQueue，用来与 NSURLConnection 一起执行 Web 请求：

```
NSOperationQueue *backgroundQueue =[[NSOperationQueue alloc] init];
```

最后，使用类方法异步地执行 Web 请求。这里需要 3 个参数：NSURLRequest 对象、NSOperationQueue 对象与代码块。NSURLConnection 通过代码块可以知道，在获取到数据后该执行哪些代码：

```
[NSURLConnection sendAsynchronousRequest:request
                                   queue:backgroundQueue
    completionHandler:^(NSURLResponse *response, NSData *data, NSError *error) {

    if(!error){
        NSString *requestResults = [[NSString alloc] initWithData:data
                        encoding:NSStringEncodingConversionAllowLossy];
        NSLog(@"requestResults=%@", requestResults);
    }
    else
        NSLog(@"error=%@", error);

          }];
```

注意：
该特性在 Mac OSX 10.7 与 iOS 5.0 中才被引入进来。

仔细查看 completionHandler 块，看看如何处理返回的 NSData 对象。通常情况下，测试 NSError 对象以在处理数据之前判断是否一切正常。这里所做的是将整个 RSS 源写到控制台日志中。但如果想要处理 RSS 源，那么可以创建类似 7.2 节中的 XML 解析器。参见程序清单 7-6～7-8。

代码

程序清单 7-6 main.m

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

int main(int argc, char *argv[]){
    return NSApplicationMain(argc, (const char **)argv);
}
```

程序清单 7-7 AppDelegate.h

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

@interface AppDelegate : NSObject <NSApplicationDelegate>
@property (assign) IBOutlet NSWindow *window;

@end
```

程序清单 7-8 AppDelegate.m

```
#import "AppDelegate.h"
```

```
@implementation AppDelegate
@synthesize window = _window;
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification{
    NSString *requestString = @"http://www.howtomakeiphoneapps.com/feed/";
    NSURL *requestURL = [NSURL URLWithString:requestString];
    NSURLRequest *request = [[NSURLRequest alloc] initWithURL: requestURL
          cachePolicy:NSURLRequestReloadIgnoringLocalCacheData timeoutInterval:10];
    NSOperationQueue *backgroundQueue =[[NSOperationQueue alloc] init];
    [NSURLConnection sendAsynchronousRequest:request
                                       queue:backgroundQueue
     completionHandler:^(NSURLResponse *response, NSData *data,
          NSError *error) {
                 if(!error){
         NSString *requestResults = [[NSString alloc] initWithData:data
encoding:NSStringEncodingConversionAllowLossy];
                     NSLog(@"requestResults=%@", requestResults);
                          }
                          else{
                              NSLog(@"error=%@", error);
                          }
             }];
}

@end
```

使用

要想尝试本攻略，请将上述代码放到 Mac 或 iOS 应用的应用委托中。通过日志查看 Web 下载的数据。要想测试错误处理，请将 Mac 断网并通过日志查看错误对象报告。如果 Web 请求成功，控制台日志会打印出如下内容：

```
requestResults=<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0"
       xmlns:content="http://purl.org/rss/1.0/modules/content/"
       xmlns:wfw="http://wellformedweb.org/CommentAPI/"
       xmlns:dc="http://purl.org/dc/elements/1.1/"
       xmlns:atom="http://www.w3.org/2005/Atom"
       xmlns:sy="http://purl.org/rss/1.0/modules/syndication/"
       xmlns:slash="http://purl.org/rss/1.0/modules/slash/"
       >

<channel>
       <title>How to Make iPhone Apps</title>
       <atom:link href="http://howtomakeiphoneapps.com/feed/" rel="self"
                 type="application/rss+xml" />
…
```

可以在应用中的任何地方使用上述代码，但不要妨碍或阻塞用户界面等其他处理。