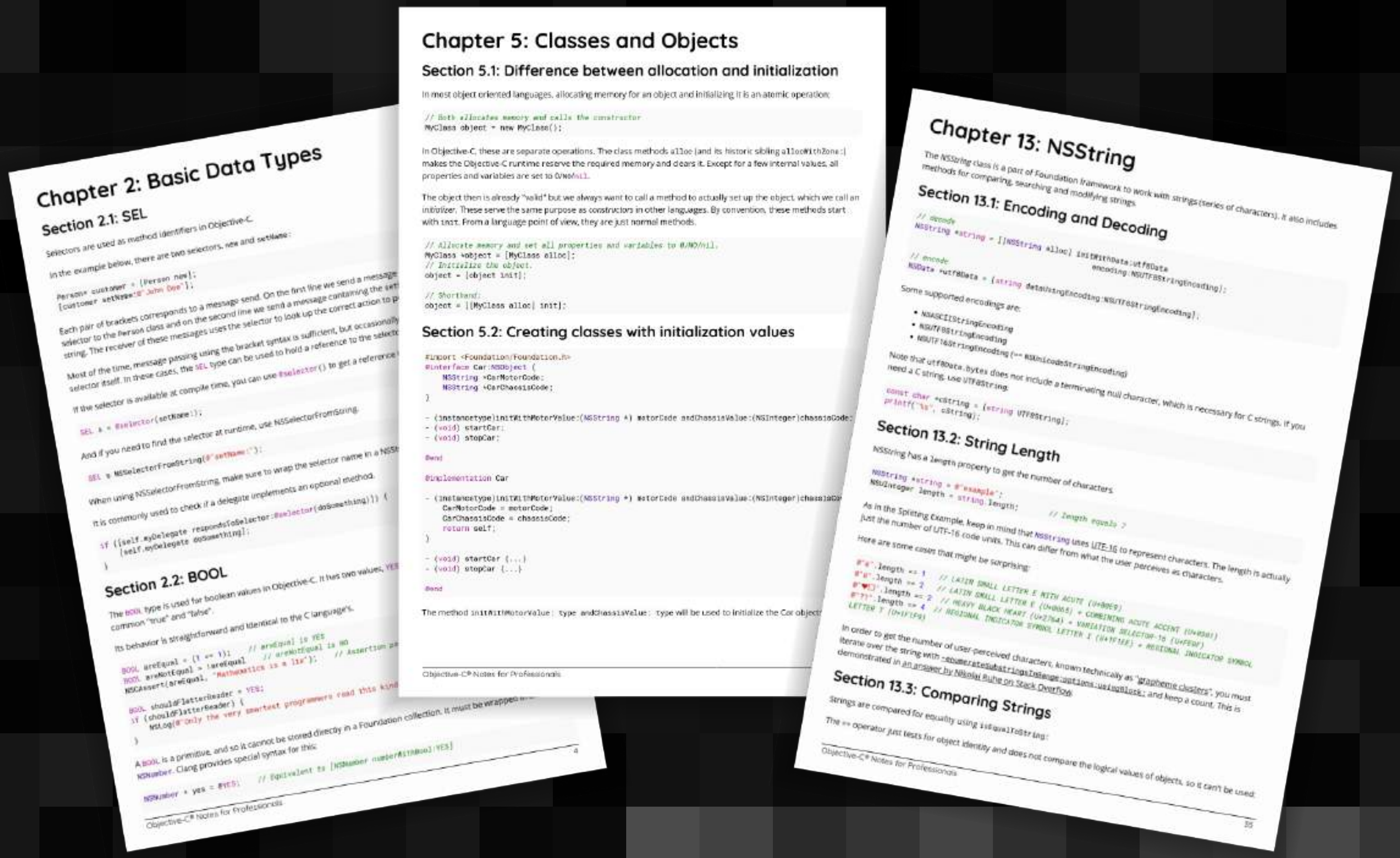
专业人士笔记

Objective-C®

专业人士笔记


Chapter 2: Basic Data Types

Section 2.1: SEL

Section 2.2: BOOL

Chapter 5: Classes and Objects

Section 5.1: Difference between allocation and initialization

Section 5.2: Creating classes with initialization values

Chapter 13: NSString

Section 13.1: Encoding and Decoding

Section 13.2: String Length

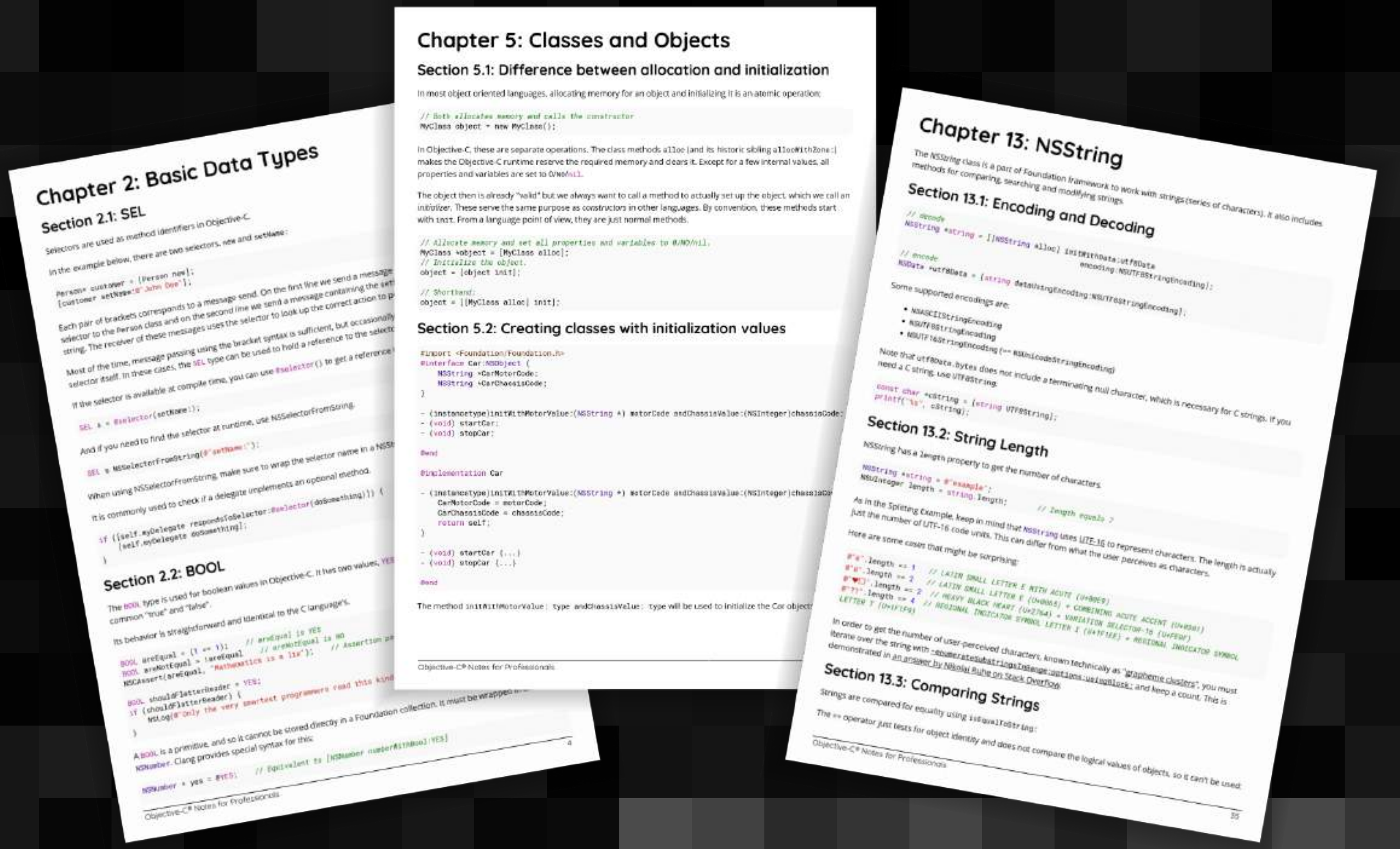
Section 13.3: Comparing Strings

100多页

专业提示和技巧

Objective-C®

Notes for Professionals


Chapter 2: Basic Data Types

Section 2.1: SEL

Section 2.2: BOOL

Chapter 5: Classes and Objects

Section 5.1: Difference between allocation and initialization

Section 5.2: Creating classes with initialization values

Chapter 13: NSString

Section 13.1: Encoding and Decoding

Section 13.2: String Length

Section 13.3: Comparing Strings

100+ pages

of professional hints and tricks

目录

Contents

关于

About

第1章：Objective-C语言入门

版本	发布日期
1.0	1983-01-01
2.0	2007-10-27
现代	2014-03-10

第1.1节：你好，世界

该程序将输出“你好，世界！”

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

int main(int argc, char * argv[]) {
    NSLog(@"你好，世界！");
}
```

#import 是一个预处理指令，表示我们想要导入或包含该文件中的信息到程序中。在本例中，编译器会将Foundation.h文件的内容复制到Foundation框架的文件顶部。#import和#include的主要区别在于，#import足够“智能”，不会

重新处理已经被其他#include包含过的文件。

C语言文档解释了main函数。

NSLog()函数会将提供的字符串打印到控制台，同时附带一些调试信息。在本例中，我们使用了Objective-C字符串字面量：@"你好，世界！"。在C语言中，你会写成"Hello World!"，然而，苹果的Foundation框架增加了NSString类，提供了许多有用的功能，并被NSLog使用。创建NSString实例的最简单方法是：@">这里是字符串内容"。

> 从技术上讲，NSLog() 是苹果 Foundation 框架的一部分，实际上并不是 Objective-C 语言的一部分。然而，Foundation 框架在整个 Objective-C 编程中无处不在。由于 Foundation 框架不是开源的，且不能在苹果开发环境之外使用，因此存在与 OPENStep 和 GNUStep 相关的开源替代框架。

编译程序

假设我们要编译我们的“Hello World”程序，该程序由一个hello.m文件组成，编译可执行文件的命令是：

```
clang -framework Foundation hello.m -o hello
```

然后你可以运行它：

```
./hello
```

这将输出：

```
Hello World!
```

Chapter 1: Getting started with Objective-C Language

Version	Release Date
1.0	1983-01-01
2.0	2007-10-27
Modern	2014-03-10

Section 1.1: Hello World

This program will output "Hello World!"

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

int main(int argc, char * argv[]) {
    NSLog(@"Hello World!");
}
```

#import is a pre-processor directive, which indicates we want to *import* or include the information from that file into the program. In this case, the compiler will copy the contents of Foundation.h in the Foundation framework to the top of the file. The main difference between #import and #include is that #import is "smart" enough to not reprocess files that have already been included in other #includes.

The C Language documentation explains the main function.

The NSLog() function will print the string provided to the console, along with some debugging information. In this case, we use an Objective-C string literal: @"Hello World!". In C, you would write this as "Hello World!", however, Apple's Foundation Framework adds the NSString class which provides a lot of useful functionality, and is used by NSLog. The simplest way to create an instance of NSString is like this: @">*string content here*".

> Technically, NSLog() is part of Apple's Foundation Framework and is not actually part of the Objective-C language. However, the Foundation Framework is ubiquitous throughout Objective-C programming. Since the Foundation Framework is not open-source and cannot be used outside of Apple development, there are open-source alternatives to the framework which are associated with OPENStep and GNUStep.

Compiling the program

Assuming we want to compile our Hello World program, which consist of a single hello.m file, the command to compile the executable is:

```
clang -framework Foundation hello.m -o hello
```

Then you can run it:

```
./hello
```

This will output:

```
Hello World!
```

选项说明如下：

- `-framework`：指定用于编译程序的框架。由于该程序使用了Foundation，我们包含了Foundation框架。
- -o: 该选项用于指定我们希望将程序输出到哪个文件。在本例中为 hello。如果未指定，默认值为 a.out。

The options are:

- `-framework`: Specifies a framework to use to compile the program. Since this program uses Foundation, we include the Foundation framework.
- `-o`: This option indicate to which file we'd like to output our program. In our case `hello`. If not specified, the default value is `a.out`.

第二章：基本数据类型

第2.1节：SEL

选择器在Objective-C中用作方法标识符。

下面的示例中，有两个选择器。new 和 setName：

```
Person* customer = [Person new];
[customer setName:@"John Doe"];
```

每对方括号对应一次消息发送。第一行我们向Person类发送包含new选择器的消息，第二行我们发送包含setName :选择器和一个字符串的消息。消息的接收者使用选择器查找要执行的正确操作。

大多数情况下，使用方括号语法进行消息传递已足够，但有时需要直接操作选择器本身。在这些情况下，可以使用SEL类型来保存对选择器的引用。

如果选择器在编译时可用，可以使用@selector()来获取其引用。

```
SEL s = @selector(setName:);
```

如果你需要在运行时查找选择器，请使用 NSSelectorFromString。

```
SEL s NSSelectorFromString(@"setName:");
```

使用 NSSelectorFromString 时，确保将选择器名称包装在 NSString 中。

它通常用于检查代理是否实现了可选方法。

```
if ([self.myDelegate respondsToSelector:@selector(doSomething)]) {
    [self.myDelegate doSomething];
}
```

第 2.2 节：BOOL

在 Objective-C 中，BOOL 类型用于布尔值。它有两个值，YES 和 NO，与更常见的“true”和“false”不同。

它的行为简单明了，与 C 语言完全相同。

```
BOOL areEqual = (1 == 1);    // areEqual 是 YES
BOOL areNotEqual = !areEqual    // areNotEqual 是 NO
NSCAssert(areEqual, "Mathematics is a lie");    // 断言通过

BOOL shouldFlatterReader = YES;
if (shouldFlatterReader) {
NSLog(@"只有最聪明的程序员才会阅读这类材料。");
}
```

A BOOL 是一种原始类型，因此不能直接存储在 Foundation 集合中。它必须被包装在一个 NSNumber 中。Clang 提供了特殊的语法：

```
NSNumber * yes = @YES;    // 等同于 [NSNumber numberWithBool:YES]
```

Chapter 2: Basic Data Types

Section 2.1: SEL

Selectors are used as method identifiers in Objective-C.

In the example below, there are two selectors. `new` and `setName:`

```
Person* customer = [Person new];
[customer setName:@"John Doe"];
```

Each pair of brackets corresponds to a message send. On the first line we send a message containing the `new` selector to the `Person` class and on the second line we send a message containing the `setName:` selector and a string. The receiver of these messages uses the selector to look up the correct action to perform.

Most of the time, message passing using the bracket syntax is sufficient, but occasionally you need to work with the selector itself. In these cases, the `SEL` type can be used to hold a reference to the selector.

If the selector is available at compile time, you can use `@selector()` to get a reference to it.

```
SEL s = @selector(setName:);
```

And if you need to find the selector at runtime, use NSSelectorFromString.

```
SEL s NSSelectorFromString(@"setName:");
```

When using NSSelectorFromString, make sure to wrap the selector name in a NSString.

It is commonly used to check if a delegate implements an optional method.

```
if ([self.myDelegate respondsToSelector:@selector(doSomething)]) {
    [self.myDelegate doSomething];
}
```

Section 2.2: BOOL

The `BOOL` type is used for boolean values in Objective-C. It has two values, `YES`, and `NO`, in contrast to the more common "true" and "false".

Its behavior is straightforward and identical to the C language's.

```
BOOL areEqual = (1 == 1);    // areEqual is YES
BOOL areNotEqual = !areEqual    // areNotEqual is NO
NSCAssert(areEqual, "Mathematics is a lie");    // Assertion passes

BOOL shouldFlatterReader = YES;
if (shouldFlatterReader) {
    NSLog(@"Only the very smartest programmers read this kind of material.");
}
```

A `BOOL` is a primitive, and so it cannot be stored directly in a Foundation collection. It must be wrapped in an `NSNumber`. Clang provides special syntax for this:

```
NSNumber * yes = @YES;    // Equivalent to [NSNumber numberWithBool:YES]
NSNumber * no = @NO;    // 等同于 [NSNumber numberWithBool:NO]
```

BOOL 的实现直接基于 C 语言，它是 C99 标准类型 bool 的一个 typedef。 YES 和 NO 值分别定义为 __objc_yes 和 __objc_no。这些特殊值是由 Clang 引入的编译器内置，转换为 (BOOL)1 和 (BOOL)0。如果不可用，YES 和 NO 则直接定义为强制类型转换的整数形式。这些定义可以在 Objective-C 运行时头文件 objc.h 中找到

第 2.3 节：id

id 是通用的对象指针，是 Objective-C 中表示“任何对象”的类型。任何 Objective-C 类的实例都可以存储在 id 变量中。 id 和任何其他类类型之间可以相互赋值，无需强制转换：

```
id anonymousSurname = @"Doe";
NSString * surname = anonymousSurname;
id anonymousFullName = [NSString stringWithFormat:@"%@, John", surname];
```

当从集合中检索对象时，这一点变得相关。像objectAtIndex这样的方法的返回类型：正是出于这个原因使用id。

```
DataRecord * record = [records objectAtIndex:anIndex];
```

这也意味着类型为id的方法或函数参数可以接受任何对象。

当一个对象被标记为id时，可以向它发送任何已知消息：方法分发不依赖于编译时类型。

```
NSString * extinctBirdMaybe =
               [anonymousSurname stringByAppendingString:anonymousSurname];
```

当然，发送一个对象实际上不响应的消息仍然会在运行时引发异常。

```
NSDate * nope = [anonymousSurname addTimeInterval:10];
// 引发“对象不响应选择器”异常
```

防止异常发生。

```
NSDate * nope;
if([anonymousSurname isKindOfClass:[NSDate class]]){
    nope = [anonymousSurname addTimeInterval:10];
}
```

id类型定义在objc.h中

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

第2.4节：IMP（实现指针）

IMP是一个C类型，指向方法的实现，也称为实现指针。它是指向方法实现起始位置的指针。

语法：

```
NSNumber * no = @NO;    // Equivalent to [NSNumber numberWithBool:NO]
```

The BOOL implementation is directly based on C's, in that it is a typedef of the C99 standard type bool. The YES and NO values are defined to __objc_yes and __objc_no, respectively. These special values are compiler builtins introduced by Clang, which are translated to (BOOL)1 and (BOOL)0. If they are not available, YES and NO are defined directly as the cast-integer form. The definitions are found in the Objective-C runtime header objc.h

Section 2.3: id

id is the generic object pointer, an Objective-C type representing "any object". An instance of any Objective-C class can be stored in an id variable. An id and any other class type can be assigned back and forth without casting:

```
id anonymousSurname = @"Doe";
NSString * surname = anonymousSurname;
id anonymousFullName = [NSString stringWithFormat:@"%@, John", surname];
```

This becomes relevant when retrieving objects from a collection. The return types of methods like objectAtIndex: are id for exactly this reason.

```
DataRecord * record = [records objectAtIndex:anIndex];
```

It also means that a method or function parameter typed as id can accept any object.

When an object is typed as id, any known message can be passed to it: method dispatch does not depend on the compile-time type.

```
NSString * extinctBirdMaybe =
               [anonymousSurname stringByAppendingString:anonymousSurname];
```

A message that the object does not actually respond to will still cause an exception at runtime, of course.

```
NSDate * nope = [anonymousSurname addTimeInterval:10];
// Raises "Does not respond to selector" exception
```

Guarding against exception.

```
NSDate * nope;
if([anonymousSurname isKindOfClass:[NSDate class]]){
    nope = [anonymousSurname addTimeInterval:10];
}
```

The id type is defined in objc.h

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

Section 2.4: IMP (implementation pointer)

IMP is a C type referring to the implementation of a method, also known as an implementation pointer. It is a pointer to the start of a method implementation.

Syntax:

```
id (*IMP)(id, SEL, ...)
```

IMP的定义为：

```
typedef id (*IMP)(id self,SEL _cmd,...);
```

要访问此IMP，可以使用消息“methodForSelector”。

示例1：

```
IMP ImpDoSomething = [myObject methodForSelector:@selector(doSomething)];
```

可以通过解引用IMP来调用IMP所指向的方法。

```
ImpDoSomething(myObject, @selector(doSomething));
```

因此，这些调用是等价的：

```
myImpDoSomething(myObject, @selector(doSomething));
[myObject doSomething]
[myObject performSelector:mySelector]
[myObject performSelector:@selector(doSomething)]
[myObject performSelector:NSSelectorFromString(@"doSomething")];
```

示例2：

```
SEL otherWaySelector = NSSelectorFromString(@“methodWithFirst:andSecond:andThird:");

IMP methodImplementation  = [self methodForSelector:otherWaySelector];

result = methodImplementation(self,
                              betterWaySelector,
                              first,
second,
                              third);

NSLog(@"methodForSelector : %@", result);
```

这里，我们调用了[NSObject methodForSelector]，它返回一个指向实际实现该方法的C函数的指针，随后我们可以直接调用该函数。

第2.5节：NSInteger和NSUInteger

NSInteger只是根据架构定义为int或long的typedef。NSUInteger同理，是无符号类型的typedef。如果你查看NS Integer，会看到如下内容：

```
#if __LP64__ || (TARGET_OS_EMBEDDED && !TARGET_OS_IPHONE) || TARGET_OS_WIN32 || NS_BUILD_32_LIKE_64
typedef long NSInteger;
typedef unsigned long NSUInteger;
#else
typedef int NSInteger;
typedef unsigned int NSUInteger;
#endif
```

有符号和无符号的int或long的区别在于，有符号的int或long可以包含负数

```
id (*IMP)(id, SEL, ...)
```

IMP is defined by:

```
typedef id (*IMP)(id self,SEL _cmd,...);
```

To access this IMP, the message “**methodForSelector**” can be used.

Example 1:

```
IMP ImpDoSomething = [myObject methodForSelector:@selector(doSomething)];
```

The method addressed by the IMP can be called by dereferencing the IMP.

```
ImpDoSomething(myObject, @selector(doSomething));
```

So these calls are equal:

```
 myImpDoSomething(myObject, @selector(doSomething));
[myObject doSomething]
[myObject performSelector:mySelector]
[myObject performSelector:@selector(doSomething)]
[myObject performSelector:NSSelectorFromString(@"doSomething")];
```

Example 2:

```
SEL otherWaySelector = NSSelectorFromString(@“methodWithFirst:andSecond:andThird:");

IMP methodImplementation  = [self methodForSelector:otherWaySelector];

result = methodImplementation( self,
                              betterWaySelector,
                              first,
                              second,
                              third );

NSLog(@"methodForSelector : %@", result);
```

Here, we call [NSObject methodForSelector which returns us a pointer to the C function that actually implements the method, which we can the subsequently call directly.

Section 2.5: NSInteger and NSUInteger

The NSInteger is just a typedef for either an int or a long depending on the architecture. The same goes for a NSUInteger which is a typedef for the unsigned variants. If you check the NSInteger you will see the following:

```
#if __LP64__ || (TARGET_OS_EMBEDDED && !TARGET_OS_IPHONE) || TARGET_OS_WIN32 || NS_BUILD_32_LIKE_64
typedef long NSInteger;
typedef unsigned long NSUInteger;
#else
typedef int NSInteger;
typedef unsigned int NSUInteger;
#endif
```

The difference between an signed and an unsigned int or long is that a signed int or long can contain negative

值。int 的范围是 -2 147 483 648 到 2 147 483 647，而 unsigned int 的范围是 0 到 4 294 967295。值翻倍是因为第一个位不再用于表示值是否为负数。对于 64 位架构上的 long 和 NSInteger，范围更广。

苹果提供的大多数方法返回的是 NS(U)Integer 而不是普通的 int。如果你尝试将其强制转换为普通的 int，会收到警告，因为在 64 位架构上会丢失精度。虽然在大多数情况下这无关紧要，但使用 NS(U)Integer 会更方便。例如，数组的 count 方法会返回一个 NSUInteger。

```
NSNumber *iAmNumber = @0;

NSInteger iAmSigned = [iAmNumber integerValue];
NSUInteger iAmUnsigned = [iAmNumber unsignedIntegerValue];

NSLog(@"%ld", iAmSigned); // 打印 NSInteger 的方式。
NSLog(@"%lu", iAmUnsigned); // 打印 NSUInteger 的方式。
```

就像 BOOL 一样，NS(U)Integer 是一种原始数据类型，所以有时你需要将其包装在 NSNumber 中，可以像上面那样在整数前加 @ 来转换，并使用下面的方法来获取。但要转换成
NSNumber，你也可以使用以下方法：

```
[NSNumber numberWithInteger:0];
[NSNumber numberWithUnsignedInteger:0];
```

values. The range of the int is -2 147 483 648 to 2 147 483 647 while the unsigned int has a range of 0 to 4 294 967 295. The value is doubled because the first bit isn't used anymore to say the value is negative or not. For a long and NSInteger on 64-bit architectures, the range is much wider.

Most methods Apple provides are returning an NS(U)Integer over the normal int. You'll get a warning if you try to cast it to a normal int because you will lose precision if you are running on a 64-bit architecture. Not that it would matter in most cases, but it is easier to use NS(U)Integer. For example, the count method on an array will return an NSUInteger.

```
NSNumber *iAmNumber = @0;

NSInteger iAmSigned = [iAmNumber integerValue];
NSUInteger iAmUnsigned = [iAmNumber unsignedIntegerValue];

NSLog(@"%ld", iAmSigned); // The way to print a NSInteger.
NSLog(@"%lu", iAmUnsigned); // The way to print a NSUInteger.
```

Just like a BOOL, the NS(U)Integer is a primitive datatype, so you sometimes need to wrap it in a NSNumber you can use the @ before the integer to cast it like above and retrieve it using the methods below. But to cast it to NSNumber, you could also use the following methods:

```
[NSNumber numberWithInteger:0];
[NSNumber numberWithUnsignedInteger:0];
```

第3章：枚举

第3.1节：Objective-C 中的 typedef enum 声明

枚举声明了一组有序的值——typedef 只是为其添加了一个方便的名称。第一个元素是 0，依此类推。

```
typedef enum {
        星期一=1,
        星期二,
        星期三

    }  工作日;

工作日 today = 星期一;//值 1
```

第3.2节：将C++ std::vector<Enum>转换为 Objective-C数组

许多C++库使用枚举，并使用包含枚举的向量返回/接收数据。由于C枚举不是Objective-C对象，Objective-C集合不能直接用于C枚举。下面的示例通过结合NSArray和泛型以及数组的包装对象来处理这个问题。这样，集合可以明确数据类型，并且不必担心使用C数组时可能出现的内存泄漏，因为使用了Objective-C对象。

以下是C枚举和Objective-C等效对象：

```
typedef enum
{
错误0 = 0,
错误1 = 1,
错误2 = 2
} MyError;

@interface ErrorEnumObj : NSObject

@property (nonatomic) int intValue;

+ (instancetype) objWithEnum:(MyError) myError;
- (MyError) getEnumValue;

@end

@implementation ErrorEnumObj

+ (instancetype) objWithEnum:(MyError) error
{
ErrorEnumObj * obj = [ErrorEnumObj new];
    obj.intValue = (int)error;
    return obj;
}

- (MyError) getEnumValue
{
    return (MyError)self.intValue;
}

@end
```

Chapter 3: Enums

Section 3.1: typedef enum declaration in Objective-C

A enum declares a set of ordered values - the typedef just adds a handy name to this. The 1st element is 0 etc.

```
typedef enum {
       Monday=1,
       Tuesday,
       Wednesday

   } WORKDAYS;

   WORKDAYS today = Monday;//value 1
```

Section 3.2: Converting C++ std::vector<Enum> to an Objective-C Array

Many C++ libraries use enums and return/receive data using vectors that contain enums. As C enums are not Objective-C objects, Objective-C collections cannot be used directly with C enums. The example below deals with this by using a combination of an NSArray and generics and a wrapper object for the array. This way, the collection can be explicit about the data type and there is no worry about possible memory leaks with C arrays Objective-C objects are used.

Here is the C enum & Objective-C equivalent object:

```
typedef enum
{
  Error0 = 0,
  Error1 = 1,
  Error2 = 2
} MyError;

@interface ErrorEnumObj : NSObject

@property (nonatomic) int intValue;

+ (instancetype) objWithEnum:(MyError) myError;
- (MyError) getEnumValue;

@end

@implementation ErrorEnumObj

+ (instancetype) objWithEnum:(MyError) error
{
    ErrorEnumObj * obj = [ErrorEnumObj new];
    obj.intValue = (int)error;
    return obj;
}

- (MyError) getEnumValue
{
    return (MyError)self.intValue;
}

@end
```

下面是在Objective-C++中可能的用法（生成的NSArray只能在Objective-C文件中使用，因为没有使用C++）。

```
class ListenerImpl : public Listener
{
public:
ListenerImpl(Listener* listener) : _listener(listener) {}
    void onError(std::vector<MyError> errors) override
    {
NSMutableArray<ErrorEnumObj *> * array = [NSMutableArray<ErrorEnumObj *> new];
        for (auto&& myError : errors)
        {
            [array addObject:[ErrorEnumObj objWithEnum:myError]];
        }
        [_listener onError:array];
    }

private:
__weak Listener* _listener;
}
```

如果这种解决方案要用于多个枚举，EnumObj（声明和实现）的创建
可以使用宏来完成（以创建类似模板的解决方案）。

第3.3节：定义枚举

枚举通过上述语法定义。

```
typedef NS_ENUM(NSUInteger, MyEnum) {
   MyEnumValueA,
MyEnumValueB,
   MyEnumValueC,
};
```

你也可以为枚举类型设置自定义的原始值。

```
typedef NS_ENUM(NSUInteger, MyEnum) {
    MyEnumValueA = 0,
MyEnumValueB = 5,
    MyEnumValueC = 10,
};
```

你也可以只指定第一个值，后续的值将自动递增：

```
typedef NS_ENUM(NSUInteger, MyEnum) {
    MyEnumValueA = 0,
MyEnumValueB,
   MyEnumValueC,
};
```

该枚举的变量可以通过MyEnum enumVar = MyEnumValueA创建。

And here is a possible use of it in Objective-C++ (the resulting NSArray can be used in Objective-C only files as no C++ is used).

```
class ListenerImpl : public Listener
{
public:
    ListenerImpl(Listener* listener) : _listener(listener) {}
    void onError(std::vector<MyError> errors) override
    {
        NSMutableArray<ErrorEnumObj *> * array = [NSMutableArray<ErrorEnumObj *> new];
        for (auto&& myError : errors)
        {
            [array addObject:[ErrorEnumObj objWithEnum:myError]];
        }
        [_listener onError:array];
    }

private:
    __weak Listener* _listener;
}
```

If this kind of solution is to be used on multiple enums, the creation of the EnumObj (declaration & implementation) can be done using a macro (to create a template like solution).

Section 3.3: Defining an enum

Enums are defined by the following the syntax above.

```
typedef NS_ENUM(NSUInteger, MyEnum) {
    MyEnumValueA,
    MyEnumValueB,
    MyEnumValueC,
};
```

You also can set your own raw-values to the enumeration types.

```
typedef NS_ENUM(NSUInteger, MyEnum) {
    MyEnumValueA = 0,
    MyEnumValueB = 5,
    MyEnumValueC = 10,
};
```

You can also specify on the first value and all the following will use it with increment:

```
typedef NS_ENUM(NSUInteger, MyEnum) {
    MyEnumValueA = 0,
    MyEnumValueB,
    MyEnumValueC,
};
```

Variables of this enum can be created by `MyEnum enumVar = MyEnumValueA`.

第4章：结构体

第4.1节：定义结构体及访问结构体成员

struct语句的格式如下：

```
struct [结构体标签]
{
成员定义；
   成员定义；
   ...
成员定义；
} [一个或多个结构体变量];
```

示例：声明ThreeFloats结构体：

```
   typedef struct {
    float x, y, z;
} ThreeFloats;

@interface MyClass
- (void)setThreeFloats:(ThreeFloats)threeFloats;
- (ThreeFloats)threeFloats;
@end
```

向MyClass的实例发送带参数@"threeFloats"的消息valueForKey:，将调用MyClass的threeFloats方法，并返回封装在NSValue中的结果。

第4.2节：CGPoint

一个非常好的结构体示例是CGPoint；它是一个表示二维点的简单值。它有两个属性，x和y，可以写成

```
typedef struct {
    CGFloat x;
    CGFloat y;
} CGPoint;
```

如果你以前使用过Mac或iOS应用开发的Objective-C，几乎肯定遇到过CGPoint；CGPoints保存屏幕上几乎所有事物的位置，从视图和控件到游戏中的对象，再到渐变的变化。这意味着CGPoint被大量使用。在性能要求很高的游戏中尤其如此；这些游戏通常有很多对象，所有这些对象都需要位置。这些位置通常是CG Point，或者是传达点的其他类型结构体（例如3D游戏中的三维点）。

像CGPoint这样的点也可以很容易地表示为对象，比如

```
@interface CGPoint {
    CGFloat x;
CGFloat y;
}

... //与点相关的方法（例如add，isEqualToPoint等）
```

Chapter 4: Structs

Section 4.1: Defining a Structure and Accessing Structure Members

The format of the struct statement is this:

```
struct [structure tag]
{
   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

Example: declare the ThreeFloats structure:

```
   typedef struct {
    float x, y, z;
} ThreeFloats;

@interface MyClass
- (void)setThreeFloats:(ThreeFloats)threeFloats;
- (ThreeFloats)threeFloats;
@end
```

Sending an instance of MyClass the message valueForKey: with the parameter @"threeFloats" will invoke the MyClass method threeFloats and return the result wrapped in an NSValue.

Section 4.2: CGPoint

One really good example of a struct is `CGPoint`; it's a simple value that represents a 2-dimensional point. It has 2 properties, `x` and `y`, and can be written as

```
typedef struct {
    CGFloat x;
    CGFloat y;
} CGPoint;
```

If you used Objective-C for Mac or iOS app development before, you've almost certainly come across `CGPoint`; `CGPoints` hold the position of pretty much everything on screen, from views and controls to objects in a game to changes in a gradient. This means that `CGPoints` are used a lot. This is even more true with really performance-heavy games; these games tend to have a lot of objects, and all of these objects need positions. These positions are often either `CGPoints`, or some other type of struct that conveys a point (such as a 3-dimensional point for 3d games).

Points like `CGPoint` could easily be represented as objects, like

```
@interface CGPoint {
    CGFloat x;
    CGFloat y;
}

... //Point-related methods (e.g. add, isEqualToPoint, etc.)
```

```
@property(nonatomic, assign)CGFloat x;
@property(nonatomic, assign)CGFloat y;

@end

@implementation CGPoint

@synthesize x, y;

...

@end
```

但是，如果以这种方式使用CGPoint，创建和操作点将花费更多时间。在较小且更快的程序中，这实际上不会造成差异，在这些情况下使用对象点是可以的，甚至可能更好。但在大量使用点的大型程序中，使用对象作为点会严重影响性能，使程序变慢，同时浪费内存，可能导致程序崩溃。

```
@property(nonatomic, assign)CGFloat x;
@property(nonatomic, assign)CGFloat y;

@end

@implementation CGPoint

@synthesize x, y;

...

@end
```

However, if `CGPoint` was used in this way it would take a lot longer to create and manipulate points. In smaller, faster programs this wouldn't really cause a difference, and in those cases it would be OK or maybe even better to use object points. But in large programs where points are be used a lot, using objects as points can really hurt performance, making the program slower, and also waste memory, which could force the program to crash.

第5章：类和对象

第5.1节：分配和初始化的区别

在大多数面向对象语言中，为对象分配内存和初始化是一个原子操作：

```
// 同时分配内存并调用构造函数
MyClass object = new MyClass();
```

在Objective-C中，这些是分开的操作。类方法alloc（及其历史上的同类方法allocWithZone:）使Objective-C运行时保留所需内存并清零。除了一些内部值外，所有属性和变量都被设置为0/NO/nil。

对象此时已经"有效"，但我们总是希望调用一个方法来实际设置对象，这个方法称为initializer。这些方法的作用与其他语言中的constructors相同。按照惯例，这些方法以init开头。从语言角度看，它们只是普通方法。

```
// 分配内存并将所有属性和变量设置为0/否/nil。
MyClass *对象 = [MyClass 分配];
// 初始化对象。
对象 = [对象 初始化];

// 简写：
对象 = [[MyClass 分配] 初始化];
```

第5.2节：使用初始化值创建类

```
#import <Foundation/Foundation.h>
@interface 车:NSObject {
    NSString *车发动机代码;
    NSString *车底盘代码;
}

          - (instancetype)initWithMotorValue:(NSString *) 发动机代码 andChassisValue:(NSInteger)底盘代码;
- (void) 启动车辆;
- (void) stopCar;

@end

@implementation Car

- (instancetype)initWithMotorValue:(NSString *) motorCode andChassisValue:(NSInteger)chassisCode{
    CarMotorCode = motorCode;
CarChassisCode = chassisCode;
    return self;
}

- (void) startCar {...}
- (void) stopCar {...}

@end
```

方法initWithMotorValue:类型和ChassisValue:类型将用于初始化Car对象。

Chapter 5: Classes and Objects

Section 5.1: Difference between allocation and initialization

In most object oriented languages, allocating memory for an object and initializing it is an atomic operation:

```
// Both allocates memory and calls the constructor
MyClass object = new MyClass();
```

In Objective-C, these are separate operations. The class methods `alloc` (and its historic sibling `allocWithZone:`) makes the Objective-C runtime reserve the required memory and clears it. Except for a few internal values, all properties and variables are set to 0/`NO`/`nil`.

The object then is already "valid" but we always want to call a method to actually set up the object, which we call an *initializer*. These serve the same purpose as *constructors* in other languages. By convention, these methods start with `init`. From a language point of view, they are just normal methods.

```
// Allocate memory and set all properties and variables to 0/NO/nil.
MyClass *object = [MyClass alloc];
// Initialize the object.
object = [object init];

// Shorthand:
object = [[MyClass alloc] init];
```

Section 5.2: Creating classes with initialization values

```
#import <Foundation/Foundation.h>
@interface Car:NSObject {
    NSString *CarMotorCode;
    NSString *CarChassisCode;
}

- (instancetype)initWithMotorValue:(NSString *) motorCode andChassisValue:(NSInteger)chassisCode;
- (void) startCar;
- (void) stopCar;

@end

@implementation Car

- (instancetype)initWithMotorValue:(NSString *) motorCode andChassisValue:(NSInteger)chassisCode{
    CarMotorCode = motorCode;
    CarChassisCode = chassisCode;
    return self;
}

- (void) startCar {...}
- (void) stopCar {...}

@end
```

The method `initWithMotorValue: type andChassisValue: type` will be used to initialize the Car objects.

第5.3节：指定泛型

你可以像NSArray或NSDictionary一样为自己的类增强泛型功能。

```
@interface MyClass<__covariant T>

@property (nonnull, nonatomic, strong, readonly) NSArray<T>* allObjects;

- (void) addObject:(nonnull T)obj;

@end
```

第5.4节：单例类

什么是单例类？

单例类无论应用请求多少次，都会返回同一个实例。与普通类不同，单例对象为其类的资源提供了一个全局访问点。

何时使用单例类？

单例通常用于需要单一控制点的场景，例如提供某些通用服务或资源的类。

如何创建单例类

首先，创建一个新文件并继承自NSObject。命名随意，这里我们使用CommonClass。Xcode将为你生成CommonClass.h和CommonClass.m文件。

在你的CommonClass.h文件中：

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

@interface CommonClass : NSObject {
}
+ (CommonClass *)sharedObject;
@property NSString *commonString;
@end
```

在你的CommonClass.m文件中：

```
#import "CommonClass.h"

@implementation CommonClass

+ (CommonClass *)sharedObject {
    static CommonClass *sharedClass = nil;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        sharedClass = [[self alloc] init];
    });
    return sharedClass;
}

- (id)init {
    if (self = [super init]) {
        self.commonString = @"this is string";
    }
```

Section 5.3: Specifying Generics

You can enhance your own classes with *generics* just like NSArray or NSDictionary.

```
@interface MyClass<__covariant T>

@property (nonnull, nonatomic, strong, readonly) NSArray<T>* allObjects;

- (void) addObject:(nonnull T)obj;

@end
```

Section 5.4: Singleton Class

What is a Singleton Class?

A singleton class returns the same instance no matter how many times an application requests it. Unlike a regular class, A singleton object provides a global point of access to the resources of its class.

When to Use Singleton Classes?

Singletons are used in situations where this single point of control is desirable, such as with classes that offer some general service or resource.

How to Create Singleton Classes

First, create a New file and subclass it from NSObject. Name it anything, we will use CommonClass here. Xcode will now generate CommonClass.h and CommonClass.m files for you.

In your CommonClass.h file:

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

@interface CommonClass : NSObject {
}
+ (CommonClass *)sharedObject;
@property NSString *commonString;
@end
```

In your CommonClass.m File:

```
#import "CommonClass.h"

@implementation CommonClass

+ (CommonClass *)sharedObject {
    static CommonClass *sharedClass = nil;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        sharedClass = [[self alloc] init];
    });
    return sharedClass;
}

- (id)init {
    if (self = [super init]) {
        self.commonString = @"this is string";
    }
    return self;
}

@end
```

如何使用单例类

我们之前创建的单例类，只要在相关模块中导入了CommonClass.h文件，就可以在项目的任何地方访问。要修改和访问单例类中的共享数据，必须通过该类的共享对象访问，可以使用sharedObject方法，如下所示：

```
[CommonClass sharedObject]
```

要读取或修改共享类中的元素，请执行以下操作：

```
NSString *commonString = [[CommonClass sharedObject].commonString; //读取单例类中的字符串


NSString *newString = @"New String";
[CommonClass sharedObject].commonString = newString;//修改单例类中的字符串
```

第5.5节："instancetype"返回类型

Objective-C支持一种特殊类型`instancetype`，只能用作方法的返回类型。它的值为接收对象的类。

考虑以下类层次结构：

```
@interface Foo : NSObject

- (instancetype)initWithString:(NSString *)string;

@end

@interface Bar : Foo
@end
```

当`[[Foo alloc] initWithString:@"abc"]`被调用时，编译器可以推断返回类型是`Foo *`。`Bar`类继承自Foo但没有重写初始化方法的声明。然而，得益于`instancetype`，编译器可以推断`[[Bar alloc] initWithString:@"xyz"]`返回的值类型是`Bar *`。

考虑将`-[Foo initWithString:]`的返回类型设为`Foo *`：如果你调用`[[Bar alloc]` initWithString:]，编译器会推断返回的是Foo *，而不是开发者意图的Bar *。instancetype解决了这个问题。

在引入instancetype之前，初始化方法、单例访问器等静态方法以及其他想返回接收类实例的方法都需要返回id。问题是id表示"任意类型的对象"。编译器因此无法检测出NSString *wrong = [[Foo alloc]initWithString:@"abc"];赋值给了类型错误的变量。

由于这个问题，初始化方法应始终使用instancetype而非id作为返回值。

```
    return self;
}

@end
```

How to Use Singleton Classes

The Singleton Class that we created earlier will be accessible from anywhere in the project as long as you have imported `CommonClass.h` file in the relevant module. To modify and access the shared data in Singleton Class, you will have to access the shared Object of that class which can be accessed by using `sharedObject` method like following:

```
[CommonClass sharedObject]
```

To read or modify the elements in Shared Class, do the following:

```
NSString *commonString = [[CommonClass sharedObject].commonString; //Read the string in singleton
class

NSString *newString = @"New String";
[CommonClass sharedObject].commonString = newString;//Modified the string in singleton class
```

Section 5.5: The "instancetype" return type

Objective-C supports a special type called `instancetype that can only be used as type returned by a method. It evaluates to the class of the receiving object.

Consider the following class hierarchy:

```
@interface Foo : NSObject

- (instancetype)initWithString:(NSString *)string;

@end

@interface Bar : Foo
@end
```

When `[[Foo alloc] initWithString:@"abc"]` is called, the compiler can infer that the return type is `Foo *`. The `Bar` class derived from `Foo` but did not override the declaration of the initializer. Yet, thanks to `instancetype`, the compiler can infer that `[[Bar alloc] initWithString:@"xyz"]` returns a value of type `Bar *`.

Consider the return type of `-[Foo initWithString:]` being `Foo *` instead: if you would call `[[Bar alloc] initWithString:]`, the compiler would infer that a `Foo *` is returned, not a `Bar *` as is the intention of the developer. The `instancetype` solved this issue.

Before the introduction of `instancetype`, initializers, static methods like singleton accessors and other methods that want to return an instance of the receiving class needed to return an `id`. The problem is that `id` means *"an object of any type"*. The compiler is thus not able to detect that `NSString *wrong = [[Foo alloc] initWithString:@"abc"];` is assigning to a variable with an incorrect type.

Due to this issue, **initializers should always use `instancetype` instead of `id`** as the return value.

第6章：继承

第6.1节：Car继承自Vehicle

考虑如下基类Vehicle及其派生类Car：

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

@interface Vehicle : NSObject

{
    NSString *vehicleName;
   NSInteger vehicleModelNo;
}

- (id)initWithName:(NSString *)name andModel:(NSInteger)modelno;
-  (void) print;
@end

@implementation Vehicle

- (id)initWithName:(NSString *)name andModel:(NSInteger)modelno{
    vehicleName = name;
vehicleModelNo = modelno;
    return self;
}

- (void)print{
NSLog(@"Name: %@", vehicleName);
    NSLog(@"Model: %ld", vehicleModelNo);
}

@end

@interface Car : Vehicle

{
    NSString *carCompanyName;
}

- (id)initWithName:(NSString *)name andModel:(NSInteger)modelno
  andCompanyName:(NSString *)companyname;
-  (void) print;

@end


@implementation Car

- (id)initWithName:(NSString *)name andModel:(NSInteger) modelno
  andCompanyName: (NSString *) companyname
  {
vehicleName = name;
    vehicleModelNo = modelno;
    carCompanyName = companyname;
    return self;
}
- (void)print
{
NSLog(@"名称: %@", vehicleName);
```

Chapter 6: Inheritance

Section 6.1: Car is inherited from Vehicle

Consider a base class **Vehicle** and its derived class **Car** as follows:

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

@interface Vehicle : NSObject

{
    NSString *vehicleName;
    NSInteger vehicleModelNo;
}

- (id)initWithName:(NSString *)name andModel:(NSInteger)modelno;
- (void)print;
@end

@implementation Vehicle

- (id)initWithName:(NSString *)name andModel:(NSInteger)modelno{
    vehicleName = name;
    vehicleModelNo = modelno;
    return self;
}

- (void)print{
    NSLog(@"Name: %@", vehicleName);
    NSLog(@"Model: %ld", vehicleModelNo);
}

@end

@interface Car : Vehicle

{
    NSString *carCompanyName;
}

- (id)initWithName:(NSString *)name andModel:(NSInteger)modelno
  andCompanyName:(NSString *)companyname;
- (void)print;

@end


@implementation Car

- (id)initWithName:(NSString *)name andModel:(NSInteger) modelno
  andCompanyName: (NSString *) companyname
  {
    vehicleName = name;
    vehicleModelNo = modelno;
    carCompanyName = companyname;
    return self;
}
- (void)print
{
    NSLog(@"Name: %@", vehicleName);
NSLog(@"型号: %ld", vehicleModelNo);
    NSLog(@"公司: %@", carCompanyName);
}

@end

int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSLog(@"基类车辆对象");
    Vehicle *vehicle = [[Vehicle alloc]initWithName:@"四轮车" andModel:1234];
    [vehicle print];
NSLog(@"继承类汽车对象");
    Car *car = [[Car alloc]initWithName:@"S级"
    andModel:7777 andCompanyName:@"奔驰"];
    [car print];
    [pool drain];
    return 0;
}
```

当上述代码被编译并执行时，产生以下结果：

```
2016-09-29 18:21:03.561 继承[349:303] 基类 车辆对象

2016-09-29 18:21:03.563 继承[349:303] 名称：四轮车

2016-09-29 18:21:03.563 Inheritance[349:303] 型号：1234

2016-09-29 18:21:03.564 Inheritance[349:303] 继承类汽车对象

2016-09-29 18:21:03.564 Inheritance[349:303] 名称：S级

2016-09-29 18:21:03.565 Inheritance[349:303] 型号：7777

2016-09-29 18:21:03.565 Inheritance[349:303] 公司：奔驰
```

```
    NSLog(@"Model: %ld", vehicleModelNo);
    NSLog(@"Company: %@", carCompanyName);
}

@end

int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSLog(@"Base class Vehicle Object");
    Vehicle *vehicle = [[Vehicle alloc]initWithName:@"4Wheeler" andModel:1234];
    [vehicle print];
    NSLog(@"Inherited Class Car Object");
    Car *car = [[Car alloc]initWithName:@"S-Class"
    andModel:7777 andCompanyName:@"Benz"];
    [car print];
    [pool drain];
    return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
2016-09-29 18:21:03.561 Inheritance[349:303] Base class Vehicle Object

2016-09-29 18:21:03.563 Inheritance[349:303] Name: 4Wheeler

2016-09-29 18:21:03.563 Inheritance[349:303] Model: 1234

2016-09-29 18:21:03.564 Inheritance[349:303] Inherited Class Car Object

2016-09-29 18:21:03.564 Inheritance[349:303] Name: S-Class

2016-09-29 18:21:03.565 Inheritance[349:303] Model: 7777

2016-09-29 18:21:03.565 Inheritance[349:303] Company: Benz
```

第7章：方法

第7.1节：类方法

类方法是在方法所属的类上调用的，而不是在其实例上调用。这是可能的，因为
Objective-C类本身也是对象。要将方法标记为类方法，将-改为+：

```
+ (void)hello {
  NSLog(@"Hello World");
}
```

第7.2节：值传递参数传递

在方法的值传递参数传递中，实际参数值会被复制给形式参数值。因此
调用函数返回后，实际参数值不会改变。

```
@interface SwapClass : NSObject

-(void) swap:(NSInteger)num1 andNum2:(NSInteger)num2;

@end

@implementation SwapClass

-(void) num:(NSInteger)num1 andNum2:(NSInteger)num2{
    int temp;
temp = num1;
    num1 = num2;
    num2 = temp;
}
@end
```

调用方法：

```
NSInteger a = 10, b =20;
SwapClass *swap = [[SwapClass alloc]init];
NSLog(@"调用 swap 之前: a=%d,b=%d",a,b);
[swap num:a andNum2:b];
NSLog(@"调用 swap 之后: a=%d,b=%d",a,b);
```

输出：

```
2016-07-30 23:55:41.870 Test[5214:81162] 调用 swap 之前: a=10,b=20
2016-07-30 23:55:41.871 Test[5214:81162] 调用 swap 之后: a=10,b=20
```

第7.3节：通过引用传递参数

在通过引用传递参数给方法时，实际参数的地址被传递给形式参数。
因此，实际参数的值将在被调用函数返回后发生变化。

```
@interface SwapClass : NSObject

-(void) swap:(int)num1 和 Num2:(int)num2;
```

Chapter 7: Methods

Section 7.1: Class methods

A class method is called on the class the method belongs to, not an instance of it. This is possible because Objective-C classes are also objects. To denote a method as a class method, change the - to a +:

```
+ (void)hello {
  NSLog(@"Hello World");
}
```

Section 7.2: Pass by value parameter passing

In pass by value of parameter passing to a method, actual parameter value is copied to formal parameter value. So actual parameter value will not change after returning from called function.

```
@interface SwapClass : NSObject

-(void) swap:(NSInteger)num1 andNum2:(NSInteger)num2;

@end

@implementation SwapClass

-(void) num:(NSInteger)num1 andNum2:(NSInteger)num2{
    int temp;
    temp = num1;
    num1 = num2;
    num2 = temp;
}
@end
```

Calling the methods:

```
NSInteger a = 10, b =20;
SwapClass *swap = [[SwapClass alloc]init];
NSLog(@"Before calling swap: a=%d,b=%d",a,b);
[swap num:a andNum2:b];
NSLog(@"After calling swap: a=%d,b=%d",a,b);
```

Output:

```
2016-07-30 23:55:41.870 Test[5214:81162] Before calling swap: a=10,b=20
2016-07-30 23:55:41.871 Test[5214:81162] After calling swap: a=10,b=20
```

Section 7.3: Pass by reference parameter passing

In pass by reference of parameter passing to a method, address of actual parameter is passed to formal parameter. So actual parameter value will be changed after returning from called function.

```
@interface SwapClass : NSObject

-(void) swap:(int)num1 andNum2:(int)num2;
```

```
@end

@implementation SwapClass

-(void) num:(int*)num1 和 Num2:(int*)num2{
     int temp;
temp = *num1;
     *num1 = *num2;
     *num2 = temp;
}
@end
```

调用方法：

```
int a = 10, b =20;
SwapClass *swap = [[SwapClass alloc]init];
NSLog(@"调用 swap 之前: a=%d,b=%d",a,b);
[swap num:&a 和 Num2:&b];
NSLog(@"调用 swap 之后: a=%d,b=%d",a,b);
```

输出：

```
2016-07-31 00:01:47.067 Test[5260:83491] 调用 swap 之前: a=10,b=20
2016-07-31 00:01:47.070 Test[5260:83491] 调用 swap 之后: a=20,b=10
```

第7.4节：方法参数

如果你想在调用方法时传入值，可以使用参数：

```
- (int)addInt:(int)intOne 加上 Int:(int)intTwo {
   return intOne + intTwo;
}
```

冒号（:）将参数与方法名分开。

参数类型写在括号内(int)。

参数名写在参数类型之后。

第7.5节：创建一个基本方法

这是创建一个在控制台打印“Hello World”的基本方法的方式：

```
- (void)hello {
   NSLog(@"Hello World");
}
```

开头的-表示这是一个实例方法。

(void)表示返回类型。此方法不返回任何值，所以写void。

‘hello’是方法的名称。

{}中的所有内容都是调用该方法时执行的代码。

```
@end

@implementation SwapClass

-(void) num:(int*)num1 andNum2:(int*)num2{
    int temp;
    temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
@end
```

Calling the methods:

```
int a = 10, b =20;
SwapClass *swap = [[SwapClass alloc]init];
NSLog(@"Before calling swap: a=%d,b=%d",a,b);
[swap num:&a andNum2:&b];
NSLog(@"After calling swap: a=%d,b=%d",a,b);
```

Output:

```
2016-07-31 00:01:47.067 Test[5260:83491] Before calling swap: a=10,b=20
2016-07-31 00:01:47.070 Test[5260:83491] After calling swap: a=20,b=10
```

Section 7.4: Method parameters

If you want to pass in values to a method when it is called, you use parameters:

```
- (int)addInt:(int)intOne toInt:(int)intTwo {
  return intOne + intTwo;
}
```

The colon (`:`) separates the parameter from the method name.

The parameter type goes in the parentheses (`int`).

The parameter name goes after the parameter type.

Section 7.5: Create a basic method

This is how to create a basic method that logs 'Hello World" to the console:

```
- (void)hello {
  NSLog(@"Hello World");
}
```

The - at the beginning denotes this method as an instance method.

The `(void)` denotes the return type. This method doesn't return anything, so you enter `void`.

The 'hello' is the name of the method.

Everything in the `{}` is the code run when the method is called.

第7.6节：返回值

当你想从一个方法返回一个值时，你需要在第一对括号中写出你想返回的类型。

```
- (NSString)returnHello {
  return @"Hello World";
}
```

你想返回的值写在return关键字后面；

第7.7节：调用方法

调用实例方法：

```
[classInstance hello];

 @interface Sample
 -(void)hello; // 暴露类的实例方法
 @end

 @implementation Sample
     -(void)hello{
        NSLog(@"hello");
      }
 @end
```

在当前实例上调用实例方法：

```
[self hello];

@implementation Sample

     -(void)otherMethod{
       [self hello];
     }

     -(void)hello{
        NSLog(@"hello");
      }
 @end
```

调用带参数的方法：

```
[classInstance addInt:1 toInt:2];

 @implementation Sample
     -(void)add:(NSInteger)add to:(NSInteger)to
        NSLog(@"sum = %d",(add+to));
      }
 @end
```

调用类方法：

```
[Class hello];

 @interface Sample
 +(void)hello; // 暴露类方法
 @end
```

Section 7.6: Return values

When you want to return a value from a method, you put the type you want to return in the first set of parentheses.

```
- (NSString)returnHello {
  return @"Hello World";
}
```

The value you want to return goes after the `return` keyword;

Section 7.7: Calling methods

Calling an instance method:

```
[classInstance hello];

 @interface Sample
 -(void)hello; // exposing the class Instance method
 @end

 @implementation Sample
     -(void)hello{
        NSLog(@"hello");
      }
 @end
```

Calling an instance method on the current instance:

```
[self hello];

@implementation Sample

     -(void)otherMethod{
       [self hello];
     }

     -(void)hello{
        NSLog(@"hello");
      }
 @end
```

Calling a method that takes arguments:

```
[classInstance addInt:1 toInt:2];

 @implementation Sample
     -(void)add:(NSInteger)add to:(NSInteger)to
        NSLog(@"sum = %d",(add+to));
      }
 @end
```

Calling a class method:

```
[Class hello];

 @interface Sample
 +(void)hello; // exposing the class method
 @end
```

```
@implementation Sample
    +(void)hello{
       NSLog(@"hello");
     }
@end
```

第7.8节：实例方法

实例方法是在类的特定实例被实例化后可用的方法：

```
MyClass *实例 = [MyClass new];
[实例 someInstanceMethod];
```

以下是定义方法的方式：

```
@interface MyClass : NSObject

- (void)someInstanceMethod; // "-" 表示实例方法

@end

@implementation MyClass

- (void)someInstanceMethod {
NSLog(@"是谁想出要有一个叫做 \"someInstanceMethod\" 的方法？");
}

@end
```

```
@implementation Sample
    +(void)hello{
       NSLog(@"hello");
     }
@end
```

Section 7.8: Instance methods

An instance method is a method that's available on a particular instance of a class, after the instance has been instantiated:

```
MyClass *instance = [MyClass new];
[instance someInstanceMethod];
```

Here's how you define one:

```
@interface MyClass : NSObject

- (void)someInstanceMethod; // "-" denotes an instance method

@end

@implementation MyClass

- (void)someInstanceMethod {
    NSLog(@"Whose idea was it to have a method called \"someInstanceMethod\"?");
}

@end
```

第8章：属性

属性	描述
原子性	隐式。 使合成的访问器方法支持同步。
非原子性	禁用合成访问器方法中的同步。
可读写	隐式。 合成getter、setter和支持变量（ivar）。
只读	仅合成getter方法和支持变量（ivar），支持变量可以直接赋值。
getter=名称	指定getter方法的名称，隐式默认为propertyName。
setter=name	指定setter方法的名称，隐式名称为setPropertyName:。冒号:必须是名称的一部分。
`strong`	ARC下对象的隐式属性。使用__strong合成支持变量，防止被引用对象被释放。
`retain`	是strong的同义词。
`copy`	与strong相同，但合成的setter还会对新值调用-copy方法。
`unsafe_unretained`	*隐式属性，ARC下对象除外。支持变量使用* __unsafe_unretained合成，对于对象来说，一旦被引用对象释放，会导致悬空指针。
`assign`	unsafe_unretained的同义词。适用于非对象类型。
`weak`	使用__weak合成备份实例变量，因此一旦被引用的对象被释放，值将被置为null。
`class`	属性访问器被合成为类方法，而非实例方法。不合成备份存储。
`nullable`	属性接受nil值。主要用于Swift桥接。
`nonnull`	属性不接受nil值。主要用于Swift桥接。
`null_resettable`	该属性在设置器中接受无值（nil），但从获取器中永远不会返回无值（nil）。您自定义的获取器或设置器实现必须确保此行为。主要用于Swift桥接。

null_unspecified隐式的。该属性不指定对无值（nil）的处理方式。主要用于Swift桥接。

第8.1节：自定义获取器和设置器

默认的属性获取器和设置器可以被重写：

```
@interface 测试类（TestClass）

@property NSString *someString;

@end

@implementation 测试类（TestClass）

// 重写设置器以打印一条消息
- (void)setSomeString:(NSString *)newString {
    NSLog(@"将 someString 设置为 %@", newString);
     // 确保访问实例变量（默认是属性名，前面带有 _）// 因为调用 self.someString 会再次调
     用同一方法，导致无限递归

_someString = newString;
}

- (void) doSomething {
     // 下一行将调用 setSomeString: 方法
self.someString = @"Test";
}
```

Chapter 8: Properties

Attribute	Description
`atomic`	*Implicit.* Enables synchronization in synthesized accessor methods.
`nonatomic`	Disables synchronization in the synthesized accessor methods.
`readwrite`	*Implicit.* Synthesizes getter, setter and backing ivar.
`readonly`	Synthesizes only the getter method and backing ivar, which can be assigned directly.
`getter=`*`name`*	Specifies the name of getter method, implicit is `propertyName`.
`setter=`*`name`*	Specifies the name of setter method, implicity is `setPropertyName:`. Colon `:` must be a part of the name.
`strong`	*Implicit for objects under ARC.* The backing ivar is synthesized using `__strong`, which prevents deallocation of referenced object.
`retain`	Synonym for `strong`.
`copy`	Same as `strong`, but the synthesized setter also calls `-copy` on the new value.
`unsafe_unretained`	*Implicit, except for objects under ARC.* The backing ivar is synthesized using `__unsafe_unretained`, which (for objects) results in dangling pointer once the referenced object deallocates.
`assign`	Synonym for `unsafe_unretained`. Suitable for non-object types.
`weak`	Backing ivar is synthesized using `__weak`, so the value will be nullified once the referenced object is deallocated.
`class`	Property accessors are synthesized as class methods, instead of instance methods. No backing storage is synthesized.
`nullable`	The property accepts `nil` values. Mainly used for Swift bridging.
`nonnull`	The property doesn't accept `nil` values. Mainly used for Swift bridging.
`null_resettable`	The property accepts `nil` values in setter, but never returns `nil` values from getter. Your custom implementation of getter or setter must ensure this behavior. Mainly used for Swift bridging.
`null_unspecified`	*Implicit.* The property doesn't specify handling of `nil` values. Mainly used for Swift bridging.

Section 8.1: Custom getters and setters

The default property getters and setters can be overridden:

```
@interface TestClass

@property NSString *someString;

@end

@implementation TestClass

// override the setter to print a message
- (void)setSomeString:(NSString *)newString {
    NSLog(@"Setting someString to %@", newString);
    // Make sure to access the ivar (default is the property name with a _
    // at the beginning) because calling self.someString would call the same
    // method again leading to an infinite recursion
    _someString = newString;
}

- (void)doSomething {
    // The next line will call the setSomeString: method
    self.someString = @"Test";
}
```

```
@end
```

这对于提供例如延迟初始化非常有用（通过重写 getter 方法，在尚未设置初始值时进行设置）：

```
- (NSString *) someString {
    if (_someString == nil) {
_someString = [self getInitialValueForSomeString];
    }
    return _someString;
}
```

你也可以创建一个在 getter 中计算其值的属性：

```
@interface Circle : NSObject

@property CGPoint origin;
@property CGFloat radius;
@property (readonly) CGFloat area;

@end

@implementation Circle

- (CGFloat)area {
    return M_PI * pow(self.radius, 2);
}

@end
```

第8.2节：导致更新的属性

该对象Shape有一个属性image，依赖于numberOfSides和sideWidth。如果其中任一属性被设置，则需要重新计算image。但重新计算可能耗时，且如果两个属性都被设置，只需计算一次，因此Shape提供了一种方法，可以同时设置两个属性并只计算一次。这是通过直接设置属性实例变量实现的。

在Shape.h

```
@interface Shape {
    NSUInteger numberOfSides;
    CGFloat sideWidth;

UIImage * image;
}

// 接收属性初始值的初始化方法。
- (instancetype)initWithNumberOfSides:(NSUInteger)numberOfSides withWidth:(CGFloat)width;

// 允许一次调用设置两个属性的方法。
// 如果设置这些属性会产生昂贵的副作用，这个方法很有用。
// 使用一个方法同时设置两个值，可以让副作用只执行一次。

- (void)setNumberOfSides:(NSUInteger)numberOfSides andWidth:(CGFloat)width;

// 使用默认属性的属性。
@property NSUInteger numberOfSides;
@property CGFloat sideWidth;
```

```
@end
```

This can be useful to provide, for example, lazy initialization (by overriding the getter to set the initial value if it has not yet been set):

```
- (NSString *)someString {
    if (_someString == nil) {
        _someString = [self getInitialValueForSomeString];
    }
    return _someString;
}
```

You can also make a property that computes its value in the getter:

```
@interface Circle : NSObject

@property CGPoint origin;
@property CGFloat radius;
@property (readonly) CGFloat area;

@end

@implementation Circle

- (CGFloat)area {
    return M_PI * pow(self.radius, 2);
}

@end
```

Section 8.2: Properties that cause updates

This object, `Shape` has a property `image` that depends on `numberOfSides` and `sideWidth`. If either one of them is set, than the `image` has to be recalculated. But recalculation is presumably long, and only needs to be done once if both properties are set, so the `Shape` provides a way to set both properties and only recalculate once. This is done by setting the property ivars directly.

In `Shape.h`

```
@interface Shape {
    NSUInteger numberOfSides;
    CGFloat sideWidth;

    UIImage * image;
}

// Initializer that takes initial values for the properties.
- (instancetype)initWithNumberOfSides:(NSUInteger)numberOfSides withWidth:(CGFloat)width;

// Method that allows to set both properties in once call.
// This is useful if setting these properties has expensive side-effects.
// Using a method to set both values at once allows you to have the side-
// effect executed only once.
- (void)setNumberOfSides:(NSUInteger)numberOfSides andWidth:(CGFloat)width;

// Properties using default attributes.
@property NSUInteger numberOfSides;
@property CGFloat sideWidth;
```

```
// 使用显式属性的属性。
@property(strong, readonly) UIImage * image;

@end
```

在Shape.m

```
@implementation AnObject

// 编译器自动生成的属性变量名默认是属性名加下划线前缀，// 例如"_propertyName"。你可
以使用以下语句更改这个默认变量名：


// @synthesize propertyName = customVariableName;

- (id)initWithNumberOfSides:(NSUInteger)numberOfSides withWidth:(CGFloat)width {
    if ((self = [self init])) {
       [self setNumberOfSides:numberOfSides andWidth:width];
    }

    return self;
}

- (void)setNumberOfSides:(NSUInteger)numberOfSides {
    _numberOfSides = numberOfSides;

    [self updateImage];
}

- (void)setSideWidth:(CGFloat)sideWidth {
    _sideWidth = sideWidth;

    [self updateImage];
}

- (void)setNumberOfSides:(NSUInteger)numberOfSides andWidth:(CGFloat)sideWidth {
    _numberOfSides = numberOfSides;
_sideWidth = sideWidth;

    [self updateImage];
}

// 当任一属性被更新后执行一些后处理的方法。

- (void)updateImage {
    …
}

@end
```

当属性被赋值（使用 object.property = value）时，会调用对应的 setter 方法 setProperty:。即使该 setter 由 @synthesize 提供，也可以被重写，正如本例中对 numberOfSides 和 sideWidth 所做的那样。但是，如果你直接设置属性的实例变量（通过property如果对象是self，或者object->property），它不会调用getter或setter，这样你就可以做一些操作，比如多次设置属性只调用一次更新，或者绕过setter引起的副作用。

第8.3节：什么是属性？

下面是一个示例类，它有几个实例变量，没有使用属性：

```
// Property using explicit attributes.
@property(strong, readonly) UIImage * image;

@end
```

In Shape.m

```
@implementation AnObject

// The variable name of a property that is auto-generated by the compiler
// defaults to being the property name prefixed with an underscore, for
// example "_propertyName". You can change this default variable name using
// the following statement:
// @synthesize propertyName = customVariableName;

- (id)initWithNumberOfSides:(NSUInteger)numberOfSides withWidth:(CGFloat)width {
    if ((self = [self init])) {
       [self setNumberOfSides:numberOfSides andWidth:width];
    }

    return self;
}

- (void)setNumberOfSides:(NSUInteger)numberOfSides {
    _numberOfSides = numberOfSides;

    [self updateImage];
}

- (void)setSideWidth:(CGFloat)sideWidth {
    _sideWidth = sideWidth;

    [self updateImage];
}

- (void)setNumberOfSides:(NSUInteger)numberOfSides andWidth:(CGFloat)sideWidth {
    _numberOfSides = numberOfSides;
    _sideWidth = sideWidth;

    [self updateImage];
}

// Method that does some post-processing once either of the properties has
// been updated.
- (void)updateImage {
    ...
}

@end
```

When properties are assigned to (using `object.property = value`), the setter method `setProperty:` is called. This setter, even if provided by `@synthesize`, can be overridden, as it is in this case for `numberOfSides` and `sideWidth`. However, if you set an property's ivar directly (through `property` if the object is self, or `object->property`), it doesn't call the getter or setter, allowing you to do things like multiple property sets that only call one update or bypass side-effects caused by the setter.

Section 8.3: What are properties?

Here is an example class which has a couple of instance variables, without using properties:

```
@interface 测试类 : NSObject {
    NSString *_someString;
    int _someInt;
}

-(NSString *)someString;
-(void)setSomeString:(NSString *)newString;

-(int)someInt;
-(void)setSomeInt:(NSString *)newInt;

@end


@implementation 测试类（TestClass）

-(NSString *)someString {
    return _someString;
}

-(void)setSomeString:(NSString *)newString {
    _someString = newString;
}

-(int)someInt {
    return _someInt;
}

-(void)setSomeInt:(int)newInt {
    _someInt = newInt;
}

@end
```

创建一个简单的实例变量需要相当多的样板代码。你必须创建实例变量并创建访问器方法，这些方法除了设置或返回实例变量外什么也不做。因此，在 Objective-C 2.0 中，苹果引入了属性（properties），它们会自动生成部分或全部样板代码。

下面是使用属性重写的上述类：

```
@interface TestClass

@property NSString *someString;
@property int someInt;

@end


@implementation testClass

@end
```

属性是与自动生成的 getter 和 setter 配对的实例变量。对于名为 someString 的属性，getter 和 setter 分别称为 someString 和 setSomeString:。实例变量的名称默认是属性名称前加下划线（因此 someString 的实例变量名为_someString），但这可以通过在 @implementation 部分使用 @synthesize 指令来覆盖：

```
@synthesize someString=foo;    //将实例变量命名为 "foo"
@synthesize someString;    //将其命名为 "someString"
```

```
@interface TestClass : NSObject {
    NSString *_someString;
    int _someInt;
}

-(NSString *)someString;
-(void)setSomeString:(NSString *)newString;

-(int)someInt;
-(void)setSomeInt:(NSString *)newInt;

@end


@implementation TestClass

-(NSString *)someString {
    return _someString;
}

-(void)setSomeString:(NSString *)newString {
    _someString = newString;
}

-(int)someInt {
    return _someInt;
}

-(void)setSomeInt:(int)newInt {
    _someInt = newInt;
}

@end
```

This is quite a lot of boilerplate code to create a simple instance variable. You have to create the instance variable & create accessor methods which do nothing except set or return the instance variable. So with Objective-C 2.0, Apple introduced properties, which auto-generate some or all of the boilerplate code.

Here is the above class rewritten with properties:

```
@interface TestClass

@property NSString *someString;
@property int someInt;

@end


@implementation testClass

@end
```

A property is an instance variable paired with auto-generated getters and setters. For a property called `someString`, the getter and setter are called `someString` and `setSomeString:` respectively. The name of the instance variable is, by default, the name of the property prefixed with an underscore (so the instance variable for `someString` is called `_someString`, but this can be overridden with an `@synthesize` directive in the `@implementation` section:

```
@synthesize someString=foo;    //names the instance variable "foo"
@synthesize someString;    //names it "someString"
```

```
@synthesize someString=_someString;         //将其命名为 "_someString"；如果没有 @synthesize 指令
                                                ，则为默认值
```

可以通过调用 getter 和 setter 来访问属性：

```
[testObject setSomeString:@"Foo"];
NSLog(@"someInt 是 %d", [testObject someInt]);
```

也可以使用点语法访问：

```
testObject.someString = @"Foo";
NSLog(@"someInt 是 %d", testObject.someInt);
```

```
@synthesize someString=_someString;        //names it "_someString"; the default if
                                           //there is no @synthesize directive
```

Properties can be accessed by calling the getters and setters:

```
[testObject setSomeString:@"Foo"];
NSLog(@"someInt is %d", [testObject someInt]);
```

They can also be accessed using dot notation:

```
testObject.someString = @"Foo";
NSLog(@"someInt is %d", testObject.someInt);
```

第9章：随机整数

第9.1节：基本随机整数

arc4random_uniform() 函数是获取高质量随机整数的最简单方法。根据手册说明：

> arc4random_uniform(upper_bound) 将返回一个小于
>
> upper_bound 的均匀分布随机数。
>
> 推荐使用 arc4random_uniform() 而非类似 "arc4random() % upper_bound" 的写法，因为它避免了当上限不是2的幂时的“模偏差”。

```
uint32_t randomInteger = arc4random_uniform(5); // 0 到 4 之间的随机整数
```

第9.2节：范围内的随机整数

下面的代码演示了如何使用arc4random_uniform()生成一个介于3到12之间的随机整数：

```
uint32_t randomIntegerWithinRange = arc4random_uniform(10) + 3; // 生成一个介于3和
12
```

这之所以能创建一个范围，是因为`arc4random_uniform(10)`返回一个介于0到9之间的整数。给这个随机整数加上3，就产生了一个介于`0 + 3`和`9 + 3`之间的范围。

Chapter 9: Random Integer

Section 9.1: Basic Random Integer

The `arc4random_uniform()` function is the simplest way to get high-quality random integers. As per the manual:

> arc4random_uniform(upper_bound) will return a uniformly distributed random number less than upper_bound.
>
> arc4random_uniform() is recommended over constructions like "arc4random() % upper_bound" as it avoids "modulo bias" when the upper bound is not a power of two.

```
uint32_t randomInteger = arc4random_uniform(5); // A random integer between 0 and 4
```

Section 9.2: Random Integer within a Range

The following code demonstrates usage of `arc4random_uniform()` to generate a random integer between 3 and 12:

```
uint32_t randomIntegerWithinRange = arc4random_uniform(10) + 3; // A random integer between 3 and
12
```

This works to create a range because `arc4random_uniform(10)` returns an integer between 0 and 9. Adding 3 to this random integer produces a range between `0 + 3` and `9 + 3`.

第10章：BOOL / bool / Boolean / NSCFBoolean

第10.1节：BOOL/Boolean/bool/NSCFBoolean

1.bool是C99中定义的数据类型。
2.Boolean值用于条件语句，如if或while语句，用于有条件地执行逻辑或重复执行。当评估条件语句时，值0被视为"假"，而任何其他值被视为"真"。由于NULL和nil被定义为0，对这些

不存在的值的条件语句也被评估为"假"。
3.BOOL是Objective-C中定义的类型，定义为有符号字符，并使用宏YES和NO来表示真和假

来自 objc.h 中的定义：

```
#if (TARGET_OS_IPHONE && __LP64__)  ||  TARGET_OS_WATCH
typedef bool BOOL;
#else
typedef signed char BOOL;
// BOOL 明确为有符号，因此 @encode(BOOL) == "c" 而不是 "C"
// 即使使用了 -funsigned-char。
#endif

#define YES ((BOOL)1)
#define NO  ((BOOL)0)
```

4.NSCFBoolean 是 NSNumber 类簇中的一个私有类。它是 CFBooleanRef 类型的桥接，用于包装 Core Foundation 属性列表和集合中的布尔值。CFBoolean 定义了常量 kCFBooleanTrue 和 kCFBooleanFalse。由于 CFNumberRef 和 CFBooleanRef 在 Core Foundation 中是不同的类型，因此它们由 NSNumber 中不同的桥接类表示是合理的。
NSNumber。

第10.2节：BOOL 与 Boolean

BOOL

- 苹果的 Objective-C 框架和大多数 Objective-C/Cocoa 代码使用
  BOOL。
- 在 Objective-C 中使用 BOOL，处理任何 CoreFoundation API 时

布尔值

- Boolean 是一个旧的 Carbon 关键字，定义为无符号字符

Chapter 10: BOOL / bool / Boolean / NSCFBoolean

Section 10.1: BOOL/Boolean/bool/NSCFBoolean

1. bool is a datatype defined in C99.
2. Boolean values are used in conditionals, such as if or while statements, to conditionally perform logic or repeat execution. When evaluating a conditional statement, the value 0 is considered "false", while any other value is considered "true". Because NULL and nil are defined as 0, conditional statements on these nonexistent values are also evaluated as "false".
3. BOOL is an Objective-C type defined as signed char with the macros YES and NO to represent true and false

From the definition in objc.h:

```
#if (TARGET_OS_IPHONE && __LP64__)  ||  TARGET_OS_WATCH
typedef bool BOOL;
#else
typedef signed char BOOL;
// BOOL is explicitly signed so @encode(BOOL) == "c" rather than "C"
// even if -funsigned-char is used.
#endif

#define YES ((BOOL)1)
#define NO  ((BOOL)0)
```

4. NSCFBoolean is a private class in the NSNumber class cluster. It is a bridge to the CFBooleanRef type, which is used to wrap boolean values for Core Foundation property lists and collections. CFBoolean defines the constants kCFBooleanTrue and kCFBooleanFalse. Because CFNumberRef and CFBooleanRef are different types in Core Foundation, it makes sense that they are represented by different bridging classes in NSNumber.

Section 10.2: BOOL VS Boolean

BOOL

- Apple's Objective-C frameworks and most Objective-C/Cocoa code uses
  BOOL.
- Use BOOL in objective-C, when dealing with any CoreFoundation APIs

Boolean

- Boolean is an old Carbon keyword , defined as an unsigned char

第11章：继续和跳出！

第11.1节：continue 和 break 语句

Objective-C 编程语言中的 continue 语句的作用有点类似于 break 语句。然而，continue 并不强制终止，而是强制执行下一次循环迭代，跳过中间的任何代码。

对于 for 循环，continue 语句会导致循环的条件测试和增量部分执行。对于 while 和 do...while 循环，continue 语句会使程序控制流转到条件测试部分。

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

int main ()
{
   /* 局部变量定义 */
   int a = 10;

   /* 执行 do 循环 */
   do
   {
      if( a == 15)
      {
         /* 跳过本次迭代 */
a = a + 1;
         continue;
      }
NSLog(@"value of a: %d", a);       a++;


   }while( a < 20 );

   return 0;
}
```

输出：

```
2013-09-07 22:20:35.647 demo[29998] value of a: 10
2013-09-07 22:20:35.647 demo[29998] value of a: 11
2013-09-07 22:20:35.647 demo[29998] value of a: 12
2013-09-07 22:20:35.647 demo[29998] value of a: 13
2013-09-07 22:20:35.647 demo[29998] value of a: 14
2013-09-07 22:20:35.647 demo[29998] value of a: 16
2013-09-07 22:20:35.647 demo[29998] value of a: 17
2013-09-07 22:20:35.647 demo[29998] value of a: 18
2013-09-07 22:20:35.647 demo[29998] value of a: 19
```

有关更多信息，请参阅此链接。

Chapter 11: Continue and Break!

Section 11.1: Continue and Break Statement

The continue statement in Objective-C programming language works somewhat like the break statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.

For the for loop, continue statement causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, continue statement causes the program control pass to the conditional tests.

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

int main ()
{
   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do
   {
      if( a == 15)
      {
         /* skip the iteration */
         a = a + 1;
         continue;
      }
      NSLog(@"value of a: %d\n", a);
      a++;

   }while( a < 20 );

   return 0;
}
```

Output:

```
2013-09-07 22:20:35.647 demo[29998] value of a: 10
2013-09-07 22:20:35.647 demo[29998] value of a: 11
2013-09-07 22:20:35.647 demo[29998] value of a: 12
2013-09-07 22:20:35.647 demo[29998] value of a: 13
2013-09-07 22:20:35.647 demo[29998] value of a: 14
2013-09-07 22:20:35.647 demo[29998] value of a: 16
2013-09-07 22:20:35.647 demo[29998] value of a: 17
2013-09-07 22:20:35.647 demo[29998] value of a: 18
2013-09-07 22:20:35.647 demo[29998] value of a: 19
```

Refer to this link for more information.

第12章：键值编码 / 键值观察

第12.1节：最常见的现实生活键值编码示例

键值编码通过NSObject集成，使用NSKeyValueCoding协议。

这意味着什么？

这意味着任何id对象都可以调用valueForKey方法及其各种变体，如valueForKeyPath
等。

这也意味着任何id对象都可以调用setValue方法及其各种变体。

示例：

```
id obj = [[MyClass alloc] init];
id value = [obj valueForKey:@"myNumber"];

int myNumberAsInt = [value intValue];
myNumberAsInt = 53;
[obj setValue:@(myNumberAsInt) forKey:@"myNumber"];
```

异常情况：

上述示例假设MyClass有一个名为myNumber的NSNumber属性。如果myNumber未出现在MyClass接口定义中
，可能会在第2行和第5行引发NSUndefinedKeyException异常——通常称为：

```
此类不支持键 myNumber 的键值编码。
```

为什么这如此强大：

您可以编写代码动态访问类的属性，而无需该类的接口。这意味着表视图可以显示任何继承自 NSObject 的对象的属性值，前提是其属性名称在运行时动态提供。

在上述示例中，即使 MyClass 不可用，且调用代码中只有 id 类型的 obj，该代码仍然可以正常工作。

第12.2节：查询 KVC 数据

```
if ([[dataObject objectForKey:@"yourVariable"] isEqualToString:"Hello World"]) {
    return YES;
} else {
    return NO;
}
```

您可以快速且轻松地查询使用 KVC 存储的值，无需将其检索或转换为局部变量。

Chapter 12: Key Value Coding / Key Value Observing

Section 12.1: Most Common Real Life Key Value Coding Example

Key Value Coding is integrated into **NSObject** using **NSKeyValueCoding** protocol.

What this means?

It means that any id object is capable of calling valueForKey method and its various variants like valueForKeyPath etc. '

It also means that any id object can invoke setValue method and its various variants too.

Example:

```
id obj = [[MyClass alloc] init];
id value = [obj valueForKey:@"myNumber"];

int myNumberAsInt = [value intValue];
myNumberAsInt = 53;
[obj setValue:@(myNumberAsInt) forKey:@"myNumber"];
```

Exceptions:

Above example assumes that MyClass has an NSNumber Property called myNumber. If myNumber does not appear in MyClass interface definition, an NSUndefinedKeyException can be raised at possibly both lines 2 and 5 - popularly known as:

```
this class is not key value coding-compliant for the key myNumber.
```

Why this is SO powerful:

You can write code that can access properties of a class dynamically, without needing interface for that class. This means that a table view can display values from any properties of an NSObject derived object, provided its property names are supplied dynamically at runtime.

In the example above, the code can as well work without MyClass being available and id type obj being available to calling code.

Section 12.2: Querying KVC Data

```
if ([[dataObject objectForKey:@"yourVariable"] isEqualToString:"Hello World"]) {
    return YES;
} else {
    return NO;
}
```

You can query values stored using KVC quickly and easily, without needing to retrieve or cast these as local variables.

第12.3节：集合操作符

集合操作符可以在 KVC 键路径中用于对“集合类型”属性执行操作（例如
NSArray、NSSet及类似集合。例如，一个常见的操作是统计集合中的对象数量。为此，你可以使用@count集合操作符：

```
self.array = @[@5, @4, @3, @2, @1];
NSNumber *count = [self.array valueForKeyPath:@"@count"];
NSNumber *countAlt = [self valueForKeyPath:@"array.@count"];
// count == countAlt == 5
```

虽然这里完全是多余的（我们本可以直接访问count属性），但有时确实有用，尽管很少必要。然而，有一些集合操作符更为有用，即@max、@min、@sum、@avg和@unionOf系列。需要注意的是，这些操作符也需要在操作符后面跟一个单独的键路径才能正确工作。以下是它们及其适用数据类型的列表：

操作符	数据类型
@count	(无)
@max	NSNumber, NSDate, int（及相关类型）等
@min	NSNumber, NSDate, int（及相关类型）等
@sum	NSNumber, int（及相关类型），double（及相关类型）等
@avg	NSNumber, int（及相关类型），double（及相关类型）等
@unionOfObjects	NSArray, NSSet等
@distinctUnionOfObjects	NSArray, NSSet等
@unionOfArrays	NSArray<NSArray*>
@distinctUnionOfArrays	NSArray<NSArray*>
@distinctUnionOfSets	NSSet<NSSet*>

@max 和 @min 分别返回集合中对象某属性的最高值或最低值。例如，查看以下代码：

```
// “Point” 类用于我们的集合
@interface Point : NSObject

@property NSInteger x, y;

+ (instancetype)pointWithX:(NSInteger)x y:(NSInteger)y;

@end

...

self.points = @[[Point pointWithX:0 y:0],
                [Point pointWithX:1 y:-1],
                [Point pointWithX:5 y:-6],
                [Point pointWithX:3 y:0],
                [Point pointWithX:8 y:-4],
];

NSNumber *maxX = [self valueForKeyPath:@"points.@max.x"]];
NSNumber *minX = [self valueForKeyPath:@"points.@min.x"]];
NSNumber *maxY = [self valueForKeyPath:@"points.@max.y"]];
NSNumber *minY = [self valueForKeyPath:@"points.@min.y"]];
```

Section 12.3: Collection Operators

Collection Operators can be used in a KVC key path to perform an operation on a “collection-type” property (i.e. NSArray, NSSet and similar). For example, a common operation to perform is to count the objects in a collection. To achieve this, you use the @count *collection operator*:

```
self.array = @[@5, @4, @3, @2, @1];
NSNumber *count = [self.array valueForKeyPath:@"@count"];
NSNumber *countAlt = [self valueForKeyPath:@"array.@count"];
// count == countAlt == 5
```

While this is completely redundant here (we could have just accessed the count property), it *can* be useful on occasion, though it is rarely necessary. There are, however, some collection operators that are much more useful, namely @max, @min, @sum, @avg and the @unionOf family. It is important to note that these operators *also* require a separate key path *following* the operator to function correctly. Here's a list of them and the type of data they work with:

Operator	Data Type
@count	(none)
@max	NSNumber, NSDate, int (and related), etc.
@min	NSNumber, NSDate, int (and related), etc.
@sum	NSNumber, int (and related), double (and related), etc.
@avg	NSNumber, int (and related), double (and related), etc.
@unionOfObjects	NSArray, NSSet, etc.
@distinctUnionOfObjects	NSArray, NSSet, etc.
@unionOfArrays	NSArray<NSArray*>
@distinctUnionOfArrays	NSArray<NSArray*>
@distinctUnionOfSets	NSSet<NSSet*>

@max and @min will return the highest or lowest value, respectively, of a property of objects in the collection. For example, look at the following code:

```
// “Point” class used in our collection
@interface Point : NSObject

@property NSInteger x, y;

+ (instancetype)pointWithX:(NSInteger)x y:(NSInteger)y;

@end

...

self.points = @[[Point pointWithX:0 y:0],
                [Point pointWithX:1 y:-1],
                [Point pointWithX:5 y:-6],
                [Point pointWithX:3 y:0],
                [Point pointWithX:8 y:-4],
];

NSNumber *maxX = [self valueForKeyPath:@"points.@max.x"];
NSNumber *minX = [self valueForKeyPath:@"points.@min.x"];
NSNumber *maxY = [self valueForKeyPath:@"points.@max.y"];
NSNumber *minY = [self valueForKeyPath:@"points.@min.y"];
```

```
NSArray<NSNumber*> *所有点的边界 = @[最大X, 最小X, 最大Y, 最小Y];

...
```

仅用4行代码和纯Foundation，借助键值编码集合操作符的强大功能，我们就能提取出一个包含数组中所有点的矩形。

需要注意的是，这些比较是通过调用对象的compare:方法来完成的，因此如果你想让自己的类兼容这些操作符，必须实现该方法。

@sum将会，如你所料，计算某个属性的所有值的总和。

```
@interface费用: NSObject

@property NSNumber *价格;

+ (instancetype)带价格的费用:(NSNumber *)价格;

@end

...

self.expenses = @[[费用 带价格:@1.50],
                  [费用 带价格:@9.99],
                  [费用 带价格:@2.78],
                  [费用 带价格:@9.99],
                  [费用 带价格:@24.95]
];

NSNumber *totalExpenses = [self valueForKeyPath:@"expenses.@sum.price"];
```

这里，我们使用@sum来计算数组中所有费用的总价。如果我们想要计算每项费用的平均价格，可以使用@avg：

```
NSNumber *averagePrice = [self valueForKeyPath:@"expenses.@avg.price"];
```

最后，是@unionOf系列。这个系列有五个不同的操作符，但它们的工作方式大致相同，仅有细微差别。首先是@unionOfObjects，它会返回数组中对象属性的数组：

```
// 参见上面的“expenses”数组

NSArray<NSNumber*> *allPrices = [self valueForKeyPath:
    @"expenses.@unionOfObjects.price"];

// 等同于 @[ @1.50, @9.99, @2.78, @9.99, @24.95 ]
```

@distinctUnionOfObjects的功能与@unionOfObjects相同，但会去除重复项：

```
NSArray<NSNumber*> *differentPrices = [self valueForKeyPath:
    @"expenses.@distinctUnionOfObjects.price"];

// 等同于 @[ @1.50, @9.99, @2.78, @24.95 ]
```

最后，@unionOf 系列中的最后三个操作符将更进一步，返回一个数组，数组中包含双重嵌套数组内某个属性的值：

```
NSArray<NSNumber*> *boundsOfAllPoints = @[maxX, minX, maxY, minY];

...
```

In just a 4 lines of code and pure Foundation, with the power of Key-Value Coding collection operators we were able to extract a rectangle that encapsulates all of the points in our array.

It is important to note that these comparisons are made by invoking the `compare:` method on the objects, so if you ever want to make your own class compatible with these operators, you must implement this method.

`@sum` will, as you can probably guess, add up all the values of a property.

```
@interface Expense : NSObject

@property NSNumber *price;

+ (instancetype)expenseWithPrice:(NSNumber *)price;

@end

...

self.expenses = @[[Expense expenseWithPrice:@1.50],
                  [Expense expenseWithPrice:@9.99],
                  [Expense expenseWithPrice:@2.78],
                  [Expense expenseWithPrice:@9.99],
                  [Expense expenseWithPrice:@24.95]
];

NSNumber *totalExpenses = [self valueForKeyPath:@"expenses.@sum.price"];
```

Here, we used `@sum` to find the total price of all the expenses in the array. If we instead wanted to find the average price we're paying for each expense, we can use `@avg`:

```
NSNumber *averagePrice = [self valueForKeyPath:@"expenses.@avg.price"];
```

Finally, there's the `@unionOf` family. There are five different operators in this family, but they all work mostly the same, with only small differences between each. First, there's `@unionOfObjects` which will return an array of the properties of objects in an array:

```
// See "expenses" array above

NSArray<NSNumber*> *allPrices = [self valueForKeyPath:
    @"expenses.@unionOfObjects.price"];

// Equal to @[ @1.50, @9.99, @2.78, @9.99, @24.95 ]
```

`@distinctUnionOfObjects` functions the same as `@unionOfObjects`, but it removes duplicates:

```
NSArray<NSNumber*> *differentPrices = [self valueForKeyPath:
    @"expenses.@distinctUnionOfObjects.price"];

// Equal to @[ @1.50, @9.99, @2.78, @24.95 ]
```

And finally, the last 3 operators in the `@unionOf` family will go one step deeper and return an array of values found for a property contained inside dually-nested arrays:

```
NSArray<NSArray<Expense*,Expense*>*> *arrayOfArrays =
    @[
@[ [Expense expenseWithPrice:@19.99],
           [Expense expenseWithPrice:@14.95],
           [Expense expenseWithPrice:@4.50],
           [Expense expenseWithPrice:@19.99]
         ],

@[ [Expense expenseWithPrice:@3.75],
           [Expense expenseWithPrice:@14.95]
         ]
     ];

// @unionOfArrays
NSArray<NSNumber*> allPrices = [arrayOfArrays valueForKeyPath:
    @"@unionOfArrays.price"];
// 等同于 @[ @19.99, @14.95, @4.50, @19.99, @3.75, @14.95 ];

// @distinctUnionOfArrays
NSArray<NSNumber*> allPrices = [arrayOfArrays valueForKeyPath:
    @"@distinctUnionOfArrays.price"];
// 等同于 @[ @19.99, @14.95, @4.50, @3.75 ];
```

这个例子中缺少的是@distinctUnionOfSets，不过这个函数的作用和@distinctUnionOfArrays完全相同，但它处理并返回的是NSSet（没有非distinct版本，因为在集合中，每个对象本来就必须是唯一的）。

就是这样！如果正确使用，集合操作符可以非常强大，并且有助于避免不必要的循环遍历。

最后一点：你也可以在NSNumber数组上使用标准的集合操作符（无需额外的属性访问）。为此，你可以访问返回对象本身的self伪属性：

```
NSArray<NSNumber*> *数字 = @[@0, @1, @5, @27, @1337, @2048];

NSNumber *最大值 = [数字 valueForKeyPath:@"@max.self"];
NSNumber *最小值 = [数字 valueForKeyPath:@"@min.self"];
NSNumber *总和 = [数字 valueForKeyPath:@"@sum.self"];
NSNumber *平均值 = [数字 valueForKeyPath:@"@avg.self"];
```

第12.4节：键值观察

设置键值观察。

在这种情况下，我们想观察观察者所拥有对象的contentOffset属性

```
//
// 观察的类
//
@interface XYZScrollView: NSObject
@property (nonatomic, assign) CGPoint contentOffset;
@end

@implementation XYZScrollView
@end


//
// 观察变化的类
```

```
NSArray<NSArray<Expense*,Expense*>*> *arrayOfArrays =
    @[
        @[ [Expense expenseWithPrice:@19.99],
           [Expense expenseWithPrice:@14.95],
           [Expense expenseWithPrice:@4.50],
           [Expense expenseWithPrice:@19.99]
         ],

        @[ [Expense expenseWithPrice:@3.75],
           [Expense expenseWithPrice:@14.95]
         ]
     ];

// @unionOfArrays
NSArray<NSNumber*> allPrices = [arrayOfArrays valueForKeyPath:
    @"@unionOfArrays.price"];
// Equal to @[ @19.99, @14.95, @4.50, @19.99, @3.75, @14.95 ];

// @distinctUnionOfArrays
NSArray<NSNumber*> allPrices = [arrayOfArrays valueForKeyPath:
    @"@distinctUnionOfArrays.price"];
// Equal to @[ @19.99, @14.95, @4.50, @3.75 ];
```

The one missing from this example is `@distinctUnionOfSets`, however this functions exactly the same as `@distinctUnionOfArrays`, but works with and returns `NSSet`s instead (there is no non-`distinct` version because in a set, every object must be distinct anyway).

And that's it! Collection operators can be really powerful if used correctly, and can help to avoid having to loop through stuff unnecessarily.

One last note: you can also use the standard collection operators on arrays of `NSNumber`s (without additional property access). To do this, you access the `self` pseudo-property that just returns the object:

```
NSArray<NSNumber*> *numbers = @[@0, @1, @5, @27, @1337, @2048];

NSNumber *largest = [numbers valueForKeyPath:@"@max.self"];
NSNumber *smallest = [numbers valueForKeyPath:@"@min.self"];
NSNumber *total = [numbers valueForKeyPath:@"@sum.self"];
NSNumber *average = [numbers valueForKeyPath:@"@avg.self"];
```

Section 12.4: Key Value Observing

Setting up key value observing.

In this case, we want to observe the `contentOffset` on an object that our observer owns

```
//
// Class to observe
//
@interface XYZScrollView: NSObject
@property (nonatomic, assign) CGPoint contentOffset;
@end

@implementation XYZScrollView
@end


//
// Class that will observe changes
```

```
//
@interface XYZObserver: NSObject
@property (nonatomic, strong) XYZScrollView *scrollView;
@end

@implementation XYZObserver

// 创建 KVO 上下文的简单方法
static void *XYZObserverContext = &XYZObserverContext;


// 辅助方法，将自身添加为
// scrollView 的 contentOffset 属性的观察者
- (void)addObserver {

    // NSKeyValueObservingOptions
    //
    // - NSKeyValueObservingOptionNew
    // - NSKeyValueObservingOptionOld
    // - NSKeyValueObservingOptionInitial
    // - NSKeyValueObservingOptionPrior
    //
    // 可以组合使用：
    // (NSKeyValueObservingOptionNew | NSKeyValueObservingOptionOld)

    NSString *keyPath = NSStringFromSelector(@selector(contentOffset));
    NSKeyValueObservingOptions options = NSKeyValueObservingOptionNew;

    [self.scrollView addObserver: self
                      forKeyPath: keyPath
                         options: options
                         context: XYZObserverContext];
}

- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
change:(NSDictionary<NSString *,id> *)change context:(void *)context {

    if (context == XYZObserverContext) { // 检查上下文

        // 检查 keyPath 是否是所需的 keyPath 之一。
        // 你可以观察多个 keyPath
        if ([keyPath isEqualToString: NSStringFromSelector(@selector(contentOffset))]) {

            // change 字典的键：
            // - NSKeyValueChangeKindKey
            // - NSKeyValueChangeNewKey
            // - NSKeyValueChangeOldKey
            // - NSKeyValueChangeIndexesKey
            // - NSKeyValueChangeNotificationIsPriorKey

            // 这里用于CGPoint观察的变化字典将
            // 返回一个NSPoint，因此我们可以取它的CGPointValue。
CGPoint point = [change[NSKeyValueChangeNewKey] CGPointValue];

            // 处理point
        }

    } else {

        // 如果上下文与我们当前对象的上下文不匹配
        // 我们希望将观察参数传递给super
        [super observeValueForKeyPath: keyPath
```

```
//
@interface XYZObserver: NSObject
@property (nonatomic, strong) XYZScrollView *scrollView;
@end

@implementation XYZObserver

// simple way to create a KVO context
static void *XYZObserverContext = &XYZObserverContext;


// Helper method to add self as an observer to
// the scrollView's contentOffset property
- (void)addObserver {

    // NSKeyValueObservingOptions
    //
    // - NSKeyValueObservingOptionNew
    // - NSKeyValueObservingOptionOld
    // - NSKeyValueObservingOptionInitial
    // - NSKeyValueObservingOptionPrior
    //
    // can be combined:
    // (NSKeyValueObservingOptionNew | NSKeyValueObservingOptionOld)

    NSString *keyPath = NSStringFromSelector(@selector(contentOffset));
    NSKeyValueObservingOptions options = NSKeyValueObservingOptionNew;

    [self.scrollView addObserver: self
                      forKeyPath: keyPath
                         options: options
                         context: XYZObserverContext];
}

- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
change:(NSDictionary<NSString *,id> *)change context:(void *)context {

    if (context == XYZObserverContext) { // check the context

        // check the keyPath to see if it's any of the desired keyPath's.
        // You can observe multiple keyPath's
        if ([keyPath isEqualToString: NSStringFromSelector(@selector(contentOffset))]) {

            // change dictionary keys:
            // - NSKeyValueChangeKindKey
            // - NSKeyValueChangeNewKey
            // - NSKeyValueChangeOldKey
            // - NSKeyValueChangeIndexesKey
            // - NSKeyValueChangeNotificationIsPriorKey

            // the change dictionary here for a CGPoint observation will
            // return an NSPoint, so we can take the CGPointValue of it.
            CGPoint point = [change[NSKeyValueChangeNewKey] CGPointValue];

            // handle point
        }

    } else {

        // if the context doesn't match our current object's context
        // we want to pass the observation parameters to super
        [super observeValueForKeyPath: keyPath
ofObject: 对象
                                  change: 变化
                                 context: 上下文];
    }
}

// 如果对象在被释放前未移除为观察者，程序可能会崩溃

//
// 辅助方法，用于移除自身作为scrollView的contentOffset属性的观察者

- (void)removeObserver {
    NSString *keyPath = NSStringFromSelector(@selector(contentOffset));
    [self.scrollView removeObserver: self forKeyPath: keyPath];
}

@end
```

```
                             ofObject: object
                               change: change
                              context: context];
    }
}

// The program can crash if an object is not removed as observer
// before it is dealloc'd
//
// Helper method to remove self as an observer of the scrollView's
// contentOffset property
- (void)removeObserver {
    NSString *keyPath = NSStringFromSelector(@selector(contentOffset));
    [self.scrollView removeObserver: self forKeyPath: keyPath];
}

@end
```

第13章：NSString

NSString类是Foundation框架的一部分，用于处理字符串（字符序列）。它还包括比较、搜索和修改字符串的方法
。

第13.1节：编码与解码

```
// 解码
NSString *string = [[NSString alloc] initWithData:utf8Data
                                         encoding:NSUTF8StringEncoding];

// 编码
NSData *utf8Data = [string dataUsingEncoding:NSUTF8StringEncoding];
```

一些支持的编码包括：

- NSASCIIStringEncoding
- NSUTF8StringEncoding
- NSUTF16StringEncoding (== NSUnicodeStringEncoding)

注意，utf8Data.bytes 不包含终止的空字符，而这是 C 字符串所必需的。如果你
需要一个 C 字符串，请使用 UTF8String：

```
const char *cString = [string UTF8String];
printf("%s", cString);
```

第13.2节：字符串长度

NSString 有一个 length 属性用于获取字符数。

```
NSString *string = @"example";
NSUInteger length = string.length;       // 长度等于7
```

如同拆分示例中所示，请记住NSString使用UTF-16来表示字符。长度实际上
只是UTF-16代码单元的数量。这可能与用户感知的字符数不同。

以下是一些可能令人惊讶的情况：

```
@"é".length == 1   // 带重音符号的拉丁小写字母e (U+00E9)
@"é".length == 2   // 拉丁小写字母e (U+0065) + 组合重音符号 (U+0301)
@"❤".length == 2  // 粗黑心形符号 (U+2764) + 变体选择符-16 (U+FE0F)
@"                    ".length == 4  // 区域指示符号字母I (U+1F1EE) + 区域指示符号
字母T (U+1F1F9)
```

为了获取用户感知的字符数，技术上称为"字形簇"，必须使用-enumerateSubstringsInRange:options:usingBlock
:遍历字符串并计数。这个方法在Stack Overflow上由Nikolai Ruhe的回答中有演示。

第13.3节：字符串比较

字符串通过isEqualToString:方法进行相等性比较

==操作符仅测试对象身份，不比较对象的逻辑值，因此不能使用：

Chapter 13: NSString

The *NSString* class is a part of Foundation framework to work with strings (series of characters). It also includes methods for comparing, searching and modifying strings.

Section 13.1: Encoding and Decoding

```
// decode
NSString *string = [[NSString alloc] initWithData:utf8Data
                                         encoding:NSUTF8StringEncoding];

// encode
NSData *utf8Data = [string dataUsingEncoding:NSUTF8StringEncoding];
```

Some supported encodings are:

- NSASCIIStringEncoding
- NSUTF8StringEncoding
- NSUTF16StringEncoding (== NSUnicodeStringEncoding)

Note that utf8Data.bytes does not include a terminating null character, which is necessary for C strings. If you need a C string, use UTF8String:

```
const char *cString = [string UTF8String];
printf("%s", cString);
```

Section 13.2: String Length

NSString has a length property to get the number of characters.

```
NSString *string = @"example";
NSUInteger length = string.length;       // length equals 7
```

As in the Splitting Example, keep in mind that NSString uses UTF-16 to represent characters. The length is actually just the number of UTF-16 code units. This can differ from what the user perceives as characters.

Here are some cases that might be surprising:

```
@"é".length == 1   // LATIN SMALL LETTER E WITH ACUTE (U+00E9)
@"é".length == 2   // LATIN SMALL LETTER E (U+0065) + COMBINING ACUTE ACCENT (U+0301)
@"❤️".length == 2  // HEAVY BLACK HEART (U+2764) + VARIATION SELECTOR-16 (U+FE0F)
@"□□".length == 4  // REGIONAL INDICATOR SYMBOL LETTER I (U+1F1EE) + REGIONAL INDICATOR SYMBOL
LETTER T (U+1F1F9)
```

In order to get the number of user-perceived characters, known technically as "grapheme clusters", you must iterate over the string with -enumerateSubstringsInRange:options:usingBlock: and keep a count. This is demonstrated in an answer by Nikolai Ruhe on Stack Overflow.

Section 13.3: Comparing Strings

Strings are compared for equality using isEqualToString:

The == operator just tests for object identity and does not compare the logical values of objects, so it can't be used:

```
NSString *stringOne = @"example";
NSString *stringTwo = [stringOne mutableCopy];

BOOL objectsAreIdentical = (stringOne == stringTwo);          // 否
BOOL stringsAreEqual = [stringOne isEqualToString:stringTwo]; // 是
```

表达式(`stringOne == stringTwo`)用于测试两个字符串的内存地址是否相同，
这通常不是我们想要的。

如果字符串变量可能为 nil，你也必须处理这种情况：

```
BOOL equalValues = stringOne == stringTwo || [stringOne isEqualToString:stringTwo];
```

当字符串值相等或两者均为 nil时，该条件返回 YES。

要按字母顺序排列两个字符串，使用 compare:。

```
NSComparisonResult result = [firstString compare:secondString];
```

NSComparisonResult 可以是：

- NSOrderedAscending: 第一个字符串排在第二个字符串之前。
- NSOrderedSame: 两个字符串相等。
- NSOrderedDescending: 第二个字符串排在第一个字符串之前。

要比较两个字符串是否相等，使用isEqualToString:。

```
BOOL result = [firstString isEqualToString:secondString];
```

与空字符串（@""）比较时，最好使用length。

```
BOOL result = string.length == 0;
```

第13.4节：拆分

你可以将字符串拆分成由分隔符字符分割的多个部分数组。

```
NSString * yourString = @"Stack,Exchange,Network";
NSArray * yourWords = [yourString componentsSeparatedByString:@","];
// 输出: @[@"Stack", @"Exchange", @"Network"]
```

如果需要根据一组**多个不同的分隔符**拆分，使用`-[NSString componentsSeparatedByCharactersInSet:]`。

```
NSString * yourString = @"Stack Overflow+Documentation/Objective-C";
NSArray * yourWords = [yourString componentsSeparatedByCharactersInSet:
                       [NSCharacterSet characterSetWithCharactersInString:@"+/"]];
// Output: @[@"Stack Overflow", @"Documentation", @"Objective-C"]`
```

如果你需要将字符串拆分成**单个字符**，可以遍历字符串的长度，将每个
字符转换成新的字符串。

```
NSMutableArray * characters = [[NSMutableArray alloc] initWithCapacity:[yourString length]];
for (int i = 0; i < [myString length]; i++) {
    [characters addObject: [NSString stringWithFormat:@"%C",
```

```
NSString *stringOne = @"example";
NSString *stringTwo = [stringOne mutableCopy];

BOOL objectsAreIdentical = (stringOne == stringTwo);          // NO
BOOL stringsAreEqual = [stringOne isEqualToString:stringTwo]; // YES
```

The expression (`stringOne == stringTwo`) tests to see if the memory addresses of the two strings are the same, which is usually not what we want.

If the string variables can be `nil` you have to take care about this case as well:

```
BOOL equalValues = stringOne == stringTwo || [stringOne isEqualToString:stringTwo];
```

This condition returns `YES` when strings have equal values or both are `nil`.

To order two strings alphabetically, use `compare:`.

```
NSComparisonResult result = [firstString compare:secondString];
```

`NSComparisonResult` can be:

- `NSOrderedAscending`: The first string comes before the second string.
- `NSOrderedSame`: The strings are equal.
- `NSOrderedDescending`: The second string comes before the first string.

To compare two strings equality, use `isEqualToString:`.

```
BOOL result = [firstString isEqualToString:secondString];
```

To compare with the empty string (`@""`), better use `length`.

```
BOOL result = string.length == 0;
```

Section 13.4: Splitting

You can split a string into an array of parts, divided by **a separator character**.

```
NSString * yourString = @"Stack,Exchange,Network";
NSArray * yourWords = [yourString componentsSeparatedByString:@","];
// Output: @[@"Stack", @"Exchange", @"Network"]
```

If you need to split on a set of **several different delimiters**, use `-[NSString componentsSeparatedByCharactersInSet:]`.

```
NSString * yourString = @"Stack Overflow+Documentation/Objective-C";
NSArray * yourWords = [yourString componentsSeparatedByCharactersInSet:
                       [NSCharacterSet characterSetWithCharactersInString:@"+/"]];
// Output: @[@"Stack Overflow", @"Documentation", @"Objective-C"]`
```

If you need to break a string into its **individual characters**, loop over the length of the string and convert each character into a new string.

```
NSMutableArray * characters = [[NSMutableArray alloc] initWithCapacity:[yourString length]];
for (int i = 0; i < [myString length]; i++) {
    [characters addObject: [NSString stringWithFormat:@"%C",
                                        [yourString characterAtIndex:i]];
}
```

如同长度示例中所述，请记住这里的“字符”是UTF-16编码单元，不一定是用户
看到的字符。如果你用这个循环处理@"　　　　"，你会看到它被拆分成四部分。

为了获取用户感知的字符列表，使用-enumerateSubstringsInRange:options:usingBlock:方法。

```
NSMutableArray * characters = [NSMutableArray array];
[yourString enumerateSubstringsInRange:(NSRange){0, [yourString length]}
                                options:NSStringEnumerationByComposedCharacterSequences
                             usingBlock:^(NSString * substring, NSRange r, NSRange s, BOOL * b){
                                 [characters addObject:substring];
                             }];
```

这将意大利国旗等字素簇保留为单个子字符串。

第13.5节：搜索子字符串

要搜索字符串是否包含子字符串，请执行以下操作：

```
NSString *myString = @"This is for checking substrings";
NSString *subString = @"checking";

BOOL doesContainSubstring = [myString containsString:subString];  // YES
```

如果目标是iOS 7或OS X 10.9（或更早版本）：

```
BOOL doesContainSubstring = ([myString rangeOfString:subString].location != NSNotFound);  // YES
```

第13.6节：创建

简单：

```
NSString *newString = @"My String";
```

从多个字符串创建：

```
NSString *stringOne = @"Hello";
NSString *stringTwo = @"world";
NSString *newString = [NSString stringWithFormat:@"我的信息：%@ %@",
                      stringOne, stringTwo];
```

使用可变字符串

```
NSString *stringOne = @"Hello";
NSString *stringTwo = @"World";
NSMutableString *mutableString = [NSMutableString new];
[mutableString appendString:stringOne];
[mutableString appendString:stringTwo];
```

来自 NSData：

从NSData初始化时，必须提供明确的编码，因为NSString无法猜测原始数据流中字符的表示方式。如今最常用
的编码是UTF-8，这甚至是一种

```
                                        [yourString characterAtIndex:i]];
}
```

As in the Length Example, keep in mind that a "character" here is a UTF-16 code unit, not necessarily what the user sees as a character. If you use this loop with @"□□", you'll see that it's split into four pieces.

In order to get a list of the user-perceived characters, use -enumerateSubstringsInRange:options:usingBlock:.

```
NSMutableArray * characters = [NSMutableArray array];
[yourString enumerateSubstringsInRange:(NSRange){0, [yourString length]}
                                options:NSStringEnumerationByComposedCharacterSequences
                             usingBlock:^(NSString * substring, NSRange r, NSRange s, BOOL * b){
                                 [characters addObject:substring];
                             }];
```

This preserves grapheme clusters like the Italian flag as a single substring.

Section 13.5: Searching for a Substring

To search if a String contains a substring, do the following:

```
NSString *myString = @"This is for checking substrings";
NSString *subString = @"checking";

BOOL doesContainSubstring = [myString containsString:subString];  // YES
```

If targeting iOS 7 or OS X 10.9 (or earlier):

```
BOOL doesContainSubstring = ([myString rangeOfString:subString].location != NSNotFound);  // YES
```

Section 13.6: Creation

Simple:

```
NSString *newString = @"My String";
```

From multiple strings:

```
NSString *stringOne = @"Hello";
NSString *stringTwo = @"world";
NSString *newString = [NSString stringWithFormat:@"My message: %@ %@",
                      stringOne, stringTwo];
```

Using Mutable String

```
NSString *stringOne = @"Hello";
NSString *stringTwo = @"World";
NSMutableString *mutableString = [NSMutableString new];
[mutableString appendString:stringOne];
[mutableString appendString:stringTwo];
```

From NSData:

When initializing from NSData, an explicit encoding must be provided as NSString is not able to guess how characters are represented in the raw data stream. The most common encoding nowadays is UTF-8, which is even a

对某些数据（如 JSON）的要求。

避免使用+[NSString stringWithUTF8String:]，因为它期望一个明确以NULL结尾的C字符串，而-[NSData bytes]并不提供。

```
NSString *newString = [[NSString alloc] initWithData:myData encoding:NSUTF8StringEncoding];
```

来自 NSArray：

```
NSArray *myArray = [NSArray arrayWithObjects:@"Apple", @"Banana", @"Strawberry", @"Kiwi", nil];
NSString *newString = [myArray componentsJoinedByString:@" "];
```

第13.7节：改变大小写

要将字符串转换为大写，使用 uppercaseString：

```
NSString *myString = @"Emphasize this";
NSLog(@"%@", [myString uppercaseString]; // @"EMPHASIZE THIS"
```

要将字符串转换为小写，使用 lowercaseString：

```
NSString *myString = @"NORMALIZE this";
NSLog(@"%@", [myString lowercaseString]; // @"normalize this"
```

要将字符串中每个单词的首字母大写，使用 capitalizedString：

```
NSString *myString = @"firstname lastname";
NSLog(@"%@", [myString capitalizedString]); // @"Firstname Lastname"
```

第13.8节：去除首尾空白字符

```
NSString *someString = @"   Objective-C Language  ";NSString *
trimmedString = [someString stringByTrimmingCharactersInSet:[NSCharacterSetwhitespaceAndNewlineCh
aracterSet]];
//输出结果将是 - "Objective-C Language"
```

方法 stringByTrimmingCharactersInSet 返回一个新字符串，该字符串通过移除原字符串两端包含在指定字符集中的字符生成。

我们也可以只移除空白字符或换行符

```
// 只移除空白字符
NSString *trimmedWhiteSpace = [someString stringByTrimmingCharactersInSet:[NSCharacterSet
whitespaceCharacterSet]];
//输出结果将是 - "Objective-C Language  "// 只移除换

行符
NSString *trimmedNewLine = [someString stringByTrimmingCharactersInSet:[NSCharacterSet
newlineCharacterSet]];
//输出结果将是 - "   Objective-C Language  "
```

第13.9节：连接字符串数组

将一个NSArray的NSString合并成一个新的NSString：

requirement for certain data like JSON.

Avoid using +[NSString stringWithUTF8String:] since it expects an explicitly NULL-terminated C-string, which -[NSData bytes] does *not* provide.

```
NSString *newString = [[NSString alloc] initWithData:myData encoding:NSUTF8StringEncoding];
```

From NSArray:

```
NSArray *myArray = [NSArray arrayWithObjects:@"Apple", @"Banana", @"Strawberry", @"Kiwi", nil];
NSString *newString = [myArray componentsJoinedByString:@" "];
```

Section 13.7: Changing Case

To convert a String to uppercase, use uppercaseString:

```
NSString *myString = @"Emphasize this";
NSLog(@"%@", [myString uppercaseString]; // @"EMPHASIZE THIS"
```

To convert a String to lowercase, use lowercaseString:

```
NSString *myString = @"NORMALIZE this";
NSLog(@"%@", [myString lowercaseString]; // @"normalize this"
```

To capitalize the first letter character of each word in a string, use capitalizedString:

```
NSString *myString = @"firstname lastname";
NSLog(@"%@", [myString capitalizedString]); // @"Firstname Lastname"
```

Section 13.8: Removing Leading and Trailing Whitespace

```
NSString *someString = @"   Objective-C Language  \n";
NSString *trimmedString = [someString stringByTrimmingCharactersInSet:[NSCharacterSet
whitespaceAndNewlineCharacterSet]];
//Output will be - "Objective-C Language"
```

Method stringByTrimmingCharactersInSet returns a new string made by removing from both ends of the String characters contained in a given character set.

We can also just remove only whitespace or newline

```
// Removing only WhiteSpace
NSString *trimmedWhiteSpace = [someString stringByTrimmingCharactersInSet:[NSCharacterSet
whitespaceCharacterSet]];
//Output will be - "Objective-C Language  \n"

// Removing only NewLine
NSString *trimmedNewLine = [someString stringByTrimmingCharactersInSet:[NSCharacterSet
newlineCharacterSet]];
//Output will be - "   Objective-C Language  "
```

Section 13.9: Joining an Array of Strings

To combine an NSArray of NSString into a new NSString:

```
NSArray *yourWords = @[@"Objective-C", @"is", @"just", @"awesome"];
NSString *sentence = [yourWords componentsJoinedByString:@" "];

// 句子现在是: @"Objective-C is just awesome"
```

第13.10节：格式化

NSString的格式化支持所有printf ANSI-C函数中可用的格式字符串。语言唯一新增的是用于格式化所有Objective-C对象的%@符号。

可以格式化整数

```
int myAge = 21;
NSString *formattedAge = [NSString stringWithFormat:@"I am %d years old", my_age];
```

或者任何继承自NSObject的对象

```
NSDate *now = [NSDate date];
NSString *formattedDate = [NSString stringWithFormat:@"The time right now is: %@", now];
```

有关格式说明符的完整列表，请参见：Objective-C，格式说明符，语法

第13.11节：处理C字符串

要将NSString转换为const char，请使用-[NSString UTF8String]：

```
NSString *myNSString = @"Some string";
const char *cString = [myNSString UTF8String];
```

如果您的字符串使用的编码不是UTF-8，也可以使用-[NSString cStringUsingEncoding:]。

反向转换请使用-[NSString stringWithUTF8String:]：

```
const *char cString = "Some string";
NSString *myNSString = [NSString stringWithUTF8String:cString];
myNSString = @(cString); // 与上述等效。
```

一旦获得了const char *，就可以像操作char数组一样使用它：

```
printf("%c", cString[5]);
```

如果想修改字符串，请先复制一份：

```
char *cpy = calloc(strlen(cString)+1, 1);
strncpy(cpy, cString, strlen(cString));
// 对 cpy 进行操作
free(cpy);
```

第13.12节：反转 NSString Objective-C

```
// myString 是 "hi"
NSMutableString * reversedString = [NSMutableString string];
NSInteger charIndex = [myString length];
while (charIndex > 0) {
```

```
NSArray *yourWords = @[@"Objective-C", @"is", @"just", @"awesome"];
NSString *sentence = [yourWords componentsJoinedByString:@" "];

// Sentence is now: @"Objective-C is just awesome"
```

Section 13.10: Formatting

The NSString formatting supports all the format strings available on the printf ANSI-C function. The only addition made by the language is the %@ symbol used for formatting all the Objective-C objects.

It is possible to format integers

```
int myAge = 21;
NSString *formattedAge = [NSString stringWithFormat:@"I am %d years old", my_age];
```

Or any object subclassed from NSObject

```
NSDate *now = [NSDate date];
NSString *formattedDate = [NSString stringWithFormat:@"The time right now is: %@", now];
```

For a complete list of Format Specifiers, please see: Objective-C, Format Specifiers, Syntax

Section 13.11: Working with C Strings

To convert NSString to const char use -[NSString UTF8String]:

```
NSString *myNSString = @"Some string";
const char *cString = [myNSString UTF8String];
```

You could also use -[NSString cStringUsingEncoding:] if your string is encoded with something other than UTF-8.

For the reverse path use -[NSString stringWithUTF8String:]:

```
const *char cString = "Some string";
NSString *myNSString = [NSString stringWithUTF8String:cString];
myNSString = @(cString); // Equivalent to the above.
```

Once you have the const char *, you can work with it similarly to an array of chars:

```
printf("%c\n", cString[5]);
```

If you want to modify the string, make a copy:

```
char *cpy = calloc(strlen(cString)+1, 1);
strncpy(cpy, cString, strlen(cString));
// Do stuff with cpy
free(cpy);
```

Section 13.12: Reversing a NSString Objective-C

```
// myString is "hi"
NSMutableString *reversedString = [NSMutableString string];
NSInteger charIndex = [myString length];
while (charIndex > 0) {
```

```
charIndex--;
    NSRange subStrRange = NSMakeRange(charIndex, 1);
    [reversedString appendString:[myString substringWithRange:subStrRange]];
}
NSLog(@"%@", reversedString); // 输出 "ih"
```

```
    charIndex--;
    NSRange subStrRange = NSMakeRange(charIndex, 1);
    [reversedString appendString:[myString substringWithRange:subStrRange]];
}
NSLog(@"%@", reversedString); // outputs "ih"
```

第14章：NSArray

第14.1节：创建数组

创建不可变数组：

```
NSArray *myColors = [NSArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];

// 使用数组字面量语法：
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];
```

有关可变数组，请参见 NSMutableArray。

第14.2节：访问元素

```
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];
// 上述是 [NSArray arrayWithObjects:...] 的推荐等价写法
```

获取单个元素

objectAtIndex 方法提供单个对象。NSArray 中的第一个对象索引为 0。由于 NSArray 可以是同质的（包含不同类型的对象），返回类型为 id（“任意对象”）。(id 可以赋值给任何其他对象类型的变量。) 重要的是，NSArray 只能包含对象，不能包含像 int 这样的值。

```
NSUInteger idx = 2;
NSString *color = [myColors objectAtIndex:idx];
// color 现在指向字符串 @"Green"
```

Clang 提供了更好的下标语法，作为其数组字面量功能的一部分：

```
NSString *color = myColors[idx];
```

如果传入的索引小于 0 或大于 count - 1，以上两种写法都会抛出异常。

第一个和最后一个元素

```
NSString *firstColor = myColors.firstObject;
NSString *lastColor = myColors.lastObject;
```

firstObject 和 lastObject 是计算属性，空数组时返回 nil 而不是崩溃。
对于单元素数组，它们返回相同的对象。虽然 firstObject 方法直到 iOS 4.0 才引入到 NSArray 中。

```
NSArray *empty = @[]
id notAnObject = empty.firstObject;    // 返回 `nil`
id kaboom = empty[0];    // 崩溃；索引越界
```

第14.3节：使用泛型

为了增加安全性，我们可以定义数组中包含的对象类型：

```
NSArray<NSString *> *colors = @[@"Red", @"Green", @"Blue", @"Yellow"];
NSMutableArray<NSString *> *myColors = [NSMutableArray arrayWithArray:colors];
```

Chapter 14: NSArray

Section 14.1: Creating Arrays

Creating immutable arrays:

```
NSArray *myColors = [NSArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];

// Using the array literal syntax:
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];
```

For mutable arrays, see NSMutableArray.

Section 14.2: Accessing elements

```
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];
// Preceding is the preferred equivalent to [NSArray arrayWithObjects:...]
```

Getting a single item

The objectAtIndex: method provides a single object. The first object in an NSArray is index 0. Since an NSArray can be homogenous (holding different types of objects), the return type is id ("any object"). (An id can be assigned to a variable of any other object type.) Importantly, NSArrays can only contain objects. They cannot contain values like int.

```
NSUInteger idx = 2;
NSString *color = [myColors objectAtIndex:idx];
// color now points to the string @"Green"
```

Clang provides a better subscript syntax as part of its array literals functionality:

```
NSString *color = myColors[idx];
```

Both of these throw an exception if the passed index is less than 0 or greater than count - 1.

First and Last Item

```
NSString *firstColor = myColors.firstObject;
NSString *lastColor = myColors.lastObject;
```

The firstObject and lastObject are computed properties and return nil rather than crashing for empty arrays. For single element arrays they return the same object. Although, the firstObject method was not introduced to NSArray until iOS 4.0.

```
NSArray *empty = @[]
id notAnObject = empty.firstObject;    // Returns `nil`
id kaboom = empty[0];    // Crashes; index out of bounds
```

Section 14.3: Using Generics

For added safety we can define the type of object that the array contains:

```
NSArray<NSString *> *colors = @[@"Red", @"Green", @"Blue", @"Yellow"];
NSMutableArray<NSString *> *myColors = [NSMutableArray arrayWithArray:colors];
```

```
[myColors addObject:@"Orange"]; // 可以
[myColors addObject:[UIColor purpleColor]]; // “不兼容的指针类型”警告
```

需要注意的是，这仅在编译时进行检查。

第14.4节：反转数组

```
NSArray *reversedArray = [myArray.reverseObjectEnumerator allObjects];
```

第14.5节：集合与数组之间的转换

```
NSSet *set = [NSSet set];
NSArray *array = [NSArray array];

NSArray *fromSet = [set allObjects];
NSSet *fromArray = [NSSet setWithArray:array];
```

第14.6节：将NSArray转换为NSMutableArray以允许修改

```
NSArray *myColors = [NSArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];

// 将myColors转换为可变数组
NSMutableArray *myColorsMutable = [myColors mutableCopy];
```

第14.7节：循环遍历

```
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];

// 快速枚举
// myColors在循环内不能被修改
for (NSString *color in myColors) {
    NSLog(@"元素 %@", color);
}

// 使用索引
for (NSUInteger i = 0; i < myColors.count; i++) {
    NSLog(@"元素 %d = %@", i, myColors[i]);
}

// 使用块枚举
[myColors enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL * stop) {
    NSLog(@"元素 %d = %@", idx, obj);

     // 要中止使用：
     *stop = YES
}];

// 使用带选项的块枚举
[myColors enumerateObjectsWithOptions:NSEnumerationReverse usingBlock:^(id obj, NSUInteger idx,
BOOL * stop) {
NSLog(@"元素 %d = %@", idx, obj);
}];
```

```
[myColors addObject:@"Orange"]; // OK
[myColors addObject:[UIColor purpleColor]]; // "Incompatible pointer type" warning
```

It should be noted that this is checked during compilation time only.

Section 14.4: Reverse an Array

```
NSArray *reversedArray = [myArray.reverseObjectEnumerator allObjects];
```

Section 14.5: Converting between Sets and Arrays

```
NSSet *set = [NSSet set];
NSArray *array = [NSArray array];

NSArray *fromSet = [set allObjects];
NSSet *fromArray = [NSSet setWithArray:array];
```

Section 14.6: Converting NSArray to NSMutableArray to allow modification

```
NSArray *myColors = [NSArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];

// Convert myColors to mutable
NSMutableArray *myColorsMutable = [myColors mutableCopy];
```

Section 14.7: Looping through

```
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];

// Fast enumeration
// myColors cannot be modified inside the loop
for (NSString *color in myColors) {
    NSLog(@"Element %@", color);
}

// Using indices
for (NSUInteger i = 0; i < myColors.count; i++) {
    NSLog(@"Element %d = %@", i, myColors[i]);
}

// Using block enumeration
[myColors enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL * stop) {
    NSLog(@"Element %d = %@", idx, obj);

    // To abort use:
    *stop = YES
}];

// Using block enumeration with options
[myColors enumerateObjectsWithOptions:NSEnumerationReverse usingBlock:^(id obj, NSUInteger idx,
BOOL * stop) {
    NSLog(@"Element %d = %@", idx, obj);
}];
```

第14.8节：使用块进行枚举

```
NSArray *myColors = @[@"红色", @"绿色", @"蓝色", @"黄色"];
[myColors enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
    NSLog(@"枚举对象 %@ 在索引 %lu", obj, idx);
}];
```

通过将 stop 参数设置为 YES，您可以表示不需要继续枚举。要做到这一点，只需设置 &停止= 是。

NSEnumerationOptions
您可以反向和/或并发枚举数组：

```
[myColors enumerateObjectsWithOptions:NSEnumerationConcurrent | NSEnumerationReverse
                               usingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
                                   NSLog(@"enumerating object %@ at index %lu", obj, idx);
                               }];
```

枚举数组的子集

```
NSIndexSet *indexSet = [NSIndexSet indexSetWithIndexesInRange:NSMakeRange(1, 1)];
[myColors enumerateObjectsAtIndexes:indexSet
options:kNilOptions
                          usingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
                               NSLog(@"enumerating object %@ at index %lu", obj, idx);
                           }];
```

第14.9节：比较数组

数组可以使用恰当命名的 isEqualToArray: 方法进行相等性比较，当两个数组元素数量相同且每对元素通过 isEqual: 比较时，该方法返回 YES。

```
NSArray *germanMakes = @[@"梅赛德斯-奔驰", @"宝马", @"保时捷",
                         @"欧宝", @"大众", @"奥迪"];
NSArray *sameGermanMakes = [NSArray arrayWithObjects:@"梅赛德斯-奔驰",
                            @"宝马", @"保时捷", @"欧宝",
                            @“大众汽车”, @“奥迪”, nil];

if ([germanMakes isEqualToArray:sameGermanMakes]) {
    NSLog(@“哦，好，字面数组和NSArray是相同的”);
}
```

重要的是每一对都必须通过isEqual:测试。对于自定义对象，应实现此方法。它存在于NSObject协议中。

第14.10节：使用谓词过滤数组

```
NSArray *array = [NSArray arrayWithObjects:@“Nick”, @“Ben”, @“Adam”, @“Melissa”, nil];

NSPredicate *aPredicate = [NSPredicate predicateWithFormat:@“SELF beginswith[c] 'a'”];
NSArray *beginWithA = [array filteredArrayUsingPredicate:bPredicate];
     // beginWithA 包含 { @“Adam” }。

NSPredicate *ePredicate = [NSPredicate predicateWithFormat:@“SELF contains[c] 'e'”];
[array filterUsingPredicate:ePredicate];
    // array 现在包含 { @“Ben”, @“Melissa” }
```

Section 14.8: Enumerating using blocks

```
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];
[myColors enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
    NSLog(@"enumerating object %@ at index %lu", obj, idx);
}];
```

By setting the `stop` parameter to `YES` you can indicate that further enumeration is not needed. to do this simply set `&stop = YES`.

NSEnumerationOptions
You can enumerate the array in reverse and / or concurrently :

```
[myColors enumerateObjectsWithOptions:NSEnumerationConcurrent | NSEnumerationReverse
                               usingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
                                   NSLog(@"enumerating object %@ at index %lu", obj, idx);
                               }];
```

Enumerating subset of array

```
NSIndexSet *indexSet = [NSIndexSet indexSetWithIndexesInRange:NSMakeRange(1, 1)];
[myColors enumerateObjectsAtIndexes:indexSet
                            options:kNilOptions
                         usingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
                              NSLog(@"enumerating object %@ at index %lu", obj, idx);
                          }];
```

Section 14.9: Comparing arrays

Arrays can be compared for equality with the aptly named **isEqualToArray:** method, which returns **YES** when both arrays have the same number of elements and every pair pass an **isEqual:** comparison.

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                         @"Opel", @"Volkswagen", @"Audi"];
NSArray *sameGermanMakes = [NSArray arrayWithObjects:@"Mercedes-Benz",
                            @"BMW", @"Porsche", @"Opel",
                            @"Volkswagen", @"Audi", nil];

if ([germanMakes isEqualToArray:sameGermanMakes]) {
    NSLog(@"Oh good, literal arrays are the same as NSArrays");
}
```

The important thing is every pair must pass the isEqual: test. For custom objects this method should be implemented. It exists in the NSObject protocol.

Section 14.10: Filtering Arrays With Predicates

```
NSArray *array = [NSArray arrayWithObjects:@"Nick", @"Ben", @"Adam", @"Melissa", nil];

NSPredicate *aPredicate = [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'a'"];
NSArray *beginWithA = [array filteredArrayUsingPredicate:bPredicate];
     // beginWithA contains { @"Adam" }.

NSPredicate *ePredicate = [NSPredicate predicateWithFormat:@"SELF contains[c] 'e'"];
[array filterUsingPredicate:ePredicate];
    // array now contains { @"Ben", @"Melissa" }
```

更多关于

NSPredicate的信息：

苹果文档：NSPredicate

第14.11节：使用自定义对象排序数组

比较方法

你可以为你的对象实现一个比较方法：

```
- (NSComparisonResult)compare:(Person *)otherObject {
    return [self.birthDate compare:otherObject.birthDate];
}

NSArray *sortedArray = [drinkDetails sortedArrayUsingSelector:@selector(compare:)];
```

NSSortDescriptor（排序描述符）

```
NSSortDescriptor *sortDescriptor;
sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"birthDate"
                                              ascending:YES];
NSArray *sortDescriptors = [NSArray arrayWithObject:sortDescriptor];
NSArray *sortedArray = [drinkDetails sortedArrayUsingDescriptors:sortDescriptors];
```

你可以通过向数组中添加多个排序描述符，轻松实现多键排序。也可以使用自定义比较器方法。请查看文档了解详情。

块（Blocks）

```
NSArray *排序数组;
排序数组 = [drinkDetails 使用比较器排序数组:^NSComparisonResult(id a, id b) {
    NSDate *第一个 = [(Person*)a 出生日期];
    NSDate *第二个 = [(Person*)b 出生日期];
    返回 [第一个 比较:第二个];
}];
```

性能

一般来说，-compare: 和基于块的方法会比使用 NSSortDescriptor 快得多，因为后者依赖于 KVC。 NSSortDescriptor 方法的主要优点是它提供了一种使用数据而非代码来定义排序顺序的方式，这使得例如用户可以通过点击表头行来排序 NSTableView 变得简单。

第14.12节：数组排序

排序数组最灵活的方法是使用 sortedArrayUsingComparator: 方法。该方法接受一个 **^NSComparisonResult(id obj1, id obj2) 块。**

```
返回值              描述
 NSOrderedAscending       obj1 在 obj2 之前
 NSOrderedSame             obj1 和 obj2 无顺序
 NSOrderedDescending      obj1 在 obj2 之后
```

示例：

```
   NSArray *类别数组 = @[@"应用", @"音乐", @"歌曲",
```

More about

NSPredicate:

Apple doc : NSPredicate

Section 14.11: Sorting array with custom objects

Compare method

Either you implement a compare-method for your object:

```
- (NSComparisonResult)compare:(Person *)otherObject {
    return [self.birthDate compare:otherObject.birthDate];
}

NSArray *sortedArray = [drinkDetails sortedArrayUsingSelector:@selector(compare:)];
```

NSSortDescriptor

```
NSSortDescriptor *sortDescriptor;
sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"birthDate"
                                              ascending:YES];
NSArray *sortDescriptors = [NSArray arrayWithObject:sortDescriptor];
NSArray *sortedArray = [drinkDetails sortedArrayUsingDescriptors:sortDescriptors];
```

You can easily sort by multiple keys by adding more than one to the array. Using custom comparator-methods is possible as well. Have a look at the documentation.

Blocks

```
NSArray *sortedArray;
sortedArray = [drinkDetails sortedArrayUsingComparator:^NSComparisonResult(id a, id b) {
    NSDate *first = [(Person*)a birthDate];
    NSDate *second = [(Person*)b birthDate];
    return [first compare:second];
}];
```

Performance

The -compare: and block-based methods will be quite a bit faster, in general, than using NSSortDescriptor as the latter relies on KVC. The primary advantage of the NSSortDescriptor method is that it provides a way to define your sort order using data, rather than code, which makes it easy to e.g. set things up so users can sort an NSTableView by clicking on the header row.

Section 14.12: Sorting Arrays

The most flexible ways to sort an array is with the sortedArrayUsingComparator: method. This accepts an **^NSComparisonResult(id obj1, id obj2) block**.

```
 Return Value           Description
 NSOrderedAscending     obj1 comes before obj2
 NSOrderedSame          obj1 and obj2 have no order
 NSOrderedDescending    obj1 comes after obj2
```

Example:

```
   NSArray *categoryArray = @[@"Apps", @"Music", @"Songs",
                      @"iTunes", @"Books", @"Videos"];

    NSArray *排序数组 = [类别数组 使用比较器排序:
^NSComparisonResult(id 对象1, id 对象2) {
    如果 ([对象1 长度] < [对象2 长度]) {
        返回 NSOrderedAscending;
    } 否则如果 ([对象1 长度] > [对象2 长度]) {
        返回 NSOrderedDescending;
    } else {
        返回 NSOrderedSame;
    }
  }];

NSLog(@"%@", 排序数组);
```

第14.13节：过滤NSArray和NSMutableArray

```
NSMutableArray *数组 =
    [NSMutableArray arrayWithObjects:@"Ken", @"Tim", @"Chris", @"Steve",@"Charlie",@"Melissa",
无];

NSPredicate *谓词 =
    [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'c'"];
NSArray *以C开头的数组 =
    [array filteredArrayUsingPredicate:bPredicate];
// 以 "C" 开头，包含 { @"Chris", @"Charlie" }。

NSPredicate *sPredicate =
    [NSPredicate predicateWithFormat:@"SELF contains[c] 'a'"];
[array filterUsingPredicate:sPredicate];
// 数组现在包含 { @"Charlie", @"Melissa" }
```

第14.14节：向NSArray添加对象

```
NSArray *a = @[@1];
a = [a arrayByAddingObject:@2];
a = [a arrayByAddingObjectsFromArray:@[@3, @4, @5]];
```

这些方法经过优化，可以非常高效地重新创建新数组，通常无需销毁
原始数组或甚至分配更多内存。

第14.15节：查找数组中的元素数量

```
NSArray *myColors = [NSArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];
NSLog (@"Number of elements in array = %lu", [myColors count]);
```

第14.16节：创建NSArray实例

```
NSArray *array1 = [NSArray arrayWithObjects:@"one", @"two", @"three", nil];
NSArray *array2 = @[@"one", @"two", @"three"];
```

```
                      @"iTunes", @"Books", @"Videos"];

    NSArray *sortedArray = [categoryArray sortedArrayUsingComparator:
^NSComparisonResult(id obj1, id obj2) {
    if ([obj1 length] < [obj2 length]) {
        return NSOrderedAscending;
    } else if ([obj1 length] > [obj2 length]) {
        return NSOrderedDescending;
    } else {
        return NSOrderedSame;
    }
  }];

 NSLog(@"%@", sortedArray);
```

Section 14.13: Filter NSArray and NSMutableArray

```
NSMutableArray *array =
    [NSMutableArray arrayWithObjects:@"Ken", @"Tim", @"Chris", @"Steve",@"Charlie",@"Melissa",
nil];

NSPredicate *bPredicate =
    [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'c'"];
NSArray *beginWithB =
    [array filteredArrayUsingPredicate:bPredicate];
// beginWith "C" contains { @"Chris", @"Charlie" }.

NSPredicate *sPredicate =
    [NSPredicate predicateWithFormat:@"SELF contains[c] 'a'"];
[array filterUsingPredicate:sPredicate];
// array now contains { @"Charlie", @"Melissa" }
```

Section 14.14: Add objects to NSArray

```
NSArray *a = @[@1];
a = [a arrayByAddingObject:@2];
a = [a arrayByAddingObjectsFromArray:@[@3, @4, @5]];
```

These methods are optimized to recreate the new array very efficiently, usually without having to destroy the original array or even allocate more memory.

Section 14.15: Finding out the Number of Elements in an Array

```
NSArray *myColors = [NSArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];
NSLog (@"Number of elements in array = %lu", [myColors count]);
```

Section 14.16: Creating NSArray instances

```
NSArray *array1 = [NSArray arrayWithObjects:@"one", @"two", @"three", nil];
NSArray *array2 = @[@"one", @"two", @"three"];
```

第15章：NSMutableArray

第15.1节：数组排序

```
NSMutableArray *myColors = [NSMutableArray arrayWithObjects: @"red", @"green", @"blue", @"yellow",
nil];
NSArray *排序数组;
sortedArray = [myColors sortedArrayUsingSelector:@selector(localizedCaseInsensitiveCompare:)];
```

第15.2节：创建NSMutableArray

NSMutableArray 可以这样初始化为空数组：

```
NSMutableArray *array = [[NSMutableArray alloc] init];
// 或者
NSMutableArray *array2 = @[].mutableCopy;
// 或者
NSMutableArray *array3 = [NSMutableArray array];
```

NSMutableArray 可以这样用另一个数组初始化：

```
NSMutableArray *array4 = [[NSMutableArray alloc] initWithArray:anotherArray];
// 或者
NSMutableArray *array5 = anotherArray.mutableCopy;
```

第15.3节：添加元素

```
NSMutableArray *myColors;
myColors = [NSMutableArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];
[myColors addObject: @"Indigo"];
[myColors addObject: @"Violet"];

//从NSArray添加对象
NSArray *myArray = @[@"Purple",@"Orange"];
[myColors addObjectsFromArray:myArray];
```

第15.4节：插入元素

```
NSMutableArray *myColors;
int i;
int count;
myColors = [NSMutableArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];
[myColors insertObject: @"Indigo" atIndex: 1];
[myColors insertObject: @"Violet" atIndex: 3];
```

第15.5节：删除元素

在特定索引处删除：

```
[myColors removeObjectAtIndex: 3];
```

删除特定对象的第一个实例：

```
[myColors removeObject: @"Red"];
```

Chapter 15: NSMutableArray

Section 15.1: Sorting Arrays

```
NSMutableArray *myColors = [NSMutableArray arrayWithObjects: @"red", @"green", @"blue", @"yellow",
nil];
NSArray *sortedArray;
sortedArray = [myColors sortedArrayUsingSelector:@selector(localizedCaseInsensitiveCompare:)];
```

Section 15.2: Creating an NSMutableArray

NSMutableArray can be initialized as an empty array like this:

```
NSMutableArray *array = [[NSMutableArray alloc] init];
// or
NSMutableArray *array2 = @[].mutableCopy;
// or
NSMutableArray *array3 = [NSMutableArray array];
```

NSMutableArray can be initialized with another array like this:

```
NSMutableArray *array4 = [[NSMutableArray alloc] initWithArray:anotherArray];
// or
NSMutableArray *array5 = anotherArray.mutableCopy;
```

Section 15.3: Adding elements

```
NSMutableArray *myColors;
myColors = [NSMutableArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];
[myColors addObject: @"Indigo"];
[myColors addObject: @"Violet"];

//Add objects from an NSArray
NSArray *myArray = @[@"Purple",@"Orange"];
[myColors addObjectsFromArray:myArray];
```

Section 15.4: Insert Elements

```
NSMutableArray *myColors;
int i;
int count;
myColors = [NSMutableArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];
[myColors insertObject: @"Indigo" atIndex: 1];
[myColors insertObject: @"Violet" atIndex: 3];
```

Section 15.5: Deleting Elements

Remove at specific index:

```
[myColors removeObjectAtIndex: 3];
```

Remove the first instance of a specific object:

```
[myColors removeObject: @"Red"];
```

移除所有指定对象的实例：

```
[myColors removeObjectIdenticalTo: @"Red"];
```

移除所有对象：

```
[myColors removeAllObjects];
```

移除最后一个对象：

```
[myColors removeLastObject];
```

第15.6节：将对象移动到另一个索引

将Blue移动到数组开头：

```
NSMutableArray *myColors = [NSMutableArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow",
nil];

NSUInteger fromIndex = 2;
NSUInteger toIndex = 0;

id blue = [[[self.array objectAtIndex:fromIndex] retain] autorelease];
[self.array removeObjectAtIndex:fromIndex];
[self.array insertObject:blue atIndex:toIndex];
```

myColors 现在是 [@"Blue", @"Red", @"Green", @"Yellow"]。

第15.7节：使用谓词过滤数组内容

使用 filterUsingPredicate: 该方法对数组内容应用给定的谓词，并返回匹配的对象。

示例：

```
      NSMutableArray *array = [NSMutableArray array];
      [array setArray:@[@"iOS",@"macOS",@"tvOS"]];
      NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'i'"];
      NSArray *resultArray = [array filteredArrayUsingPredicate:predicate];
NSLog(@"%@",resultArray);
```

Remove all instances of a specific object:

```
[myColors removeObjectIdenticalTo: @"Red"];
```

Remove all objects:

```
[myColors removeAllObjects];
```

Remove last object:

```
[myColors removeLastObject];
```

Section 15.6: Move object to another index

Move *Blue* to the beginning of the array:

```
NSMutableArray *myColors = [NSMutableArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow",
nil];

NSUInteger fromIndex = 2;
NSUInteger toIndex = 0;

id blue = [[[self.array objectAtIndex:fromIndex] retain] autorelease];
[self.array removeObjectAtIndex:fromIndex];
[self.array insertObject:blue atIndex:toIndex];
```

myColors is now [@"Blue", @"Red", @"Green", @"Yellow"].

Section 15.7: Filtering Array content with Predicate

Using **filterUsingPredicate:** This Evaluates a given predicate against the arrays content and return objects that match.

Example:

```
      NSMutableArray *array = [NSMutableArray array];
      [array setArray:@[@"iOS",@"macOS",@"tvOS"]];
      NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'i'"];
      NSArray *resultArray = [array filteredArrayUsingPredicate:predicate];
      NSLog(@"%@",resultArray);
```

第16章：NSDictionary

第16.1节：创建

```
NSDictionary *dict = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1", @"value2",
@"key2", nil];
```

或者

```
NSArray *keys = [NSArray arrayWithObjects:@"key1", @"key2", nil];
NSArray *objects = [NSArray arrayWithObjects:@"value1", @"value2", nil];
NSDictionary *dictionary = [NSDictionary dictionaryWithObjects:objects
                                                       forKeys:keys];
```

或者使用合适的字面量语法

```
NSDictionary *dict = @{@"key": @"value", @"nextKey": @"nextValue"};
```

第16.2节：快速枚举

NSDictionary 可以像其他集合类型一样使用快速枚举：

```
NSDictionary stockSymbolsDictionary = @{
                                     @"AAPL": @"苹果",
                                     @"GOOGL": @"字母表",
                                     @"MSFT": @"微软",
                                     @"AMZN": @"亚马逊"
                                   };

for (id key in stockSymbolsDictionary)
{
    id value = dictionary[key];
    NSLog(@"Key: %@, Value: %@", key, value);
}
```

由于NSDictionary本质上是无序的，for循环中键的顺序无法保证。

第16.3节：使用字面量创建

```
NSDictionary *库存 = @{
    @"梅赛德斯-奔驰 SLK250" : @(13),
    @"宝马 M3 Coupe" : @(self.BMWM3CoupeInventory.count),
    @"最后更新" : @"2016年7月21日",
    @"下次更新"  : self.nextInventoryUpdateString
};
```

第16.4节：使用 dictionaryWithObjectsAndKeys: 创建

```
NSDictionary *库存 = [NSDictionary dictionaryWithObjectsAndKeys:
    [NSNumber numberWithInt:13], @"梅赛德斯-奔驰 SLK250",
    [NSNumber numberWithInt:22], @"梅赛德斯-奔驰 E350",
    [NSNumber numberWithInt:19], @"宝马 M3 Coupe",
    [NSNumber numberWithInt:16], @"宝马 X6",
    nil];
```

Chapter 16: NSDictionary

Section 16.1: Create

```
NSDictionary *dict = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1", @"value2",
@"key2", nil];
```

or

```
NSArray *keys = [NSArray arrayWithObjects:@"key1", @"key2", nil];
NSArray *objects = [NSArray arrayWithObjects:@"value1", @"value2", nil];
NSDictionary *dictionary = [NSDictionary dictionaryWithObjects:objects
                                                       forKeys:keys];
```

or using appropriate literal syntax

```
NSDictionary *dict = @{@"key": @"value", @"nextKey": @"nextValue"};
```

Section 16.2: Fast Enumeration

NSDictionary can be enumerated using fast enumeration, just like other collection types:

```
NSDictionary stockSymbolsDictionary = @{
                                     @"AAPL": @"Apple",
                                     @"GOOGL": @"Alphabet",
                                     @"MSFT": @"Microsoft",
                                     @"AMZN": @"Amazon"
                                   };

for (id key in stockSymbolsDictionary)
{
    id value = dictionary[key];
    NSLog(@"Key: %@, Value: %@", key, value);
}
```

Because NSDictionary is inherently unordered, the order of keys that in the for loop is not guaranteed.

Section 16.3: Creating using literals

```
NSDictionary *inventory = @{
    @"Mercedes-Benz SLK250" : @(13),
    @"BMW M3 Coupe" : @(self.BMWM3CoupeInventory.count),
    @"Last Updated" : @"Jul 21, 2016",
    @"Next Update"  : self.nextInventoryUpdateString
};
```

Section 16.4: Creating using dictionaryWithObjectsAndKeys:

```
NSDictionary *inventory = [NSDictionary dictionaryWithObjectsAndKeys:
    [NSNumber numberWithInt:13], @"Mercedes-Benz SLK250",
    [NSNumber numberWithInt:22], @"Mercedes-Benz E350",
    [NSNumber numberWithInt:19], @"BMW M3 Coupe",
    [NSNumber numberWithInt:16], @"BMW X6",
    nil];
```

nil 必须作为最后一个参数传递，作为结束的哨兵标志。

重要的是要记住，以这种方式实例化字典时，值先于键。在上面的例子中，字符串是键，数字是值。方法名也反映了这一点：
dictionaryWithObjectsAndKeys。虽然这不是错误的，但更现代的实例化字典方式（使用字面量）更受推荐。

第16.5节：NSDictionary 转 NSArray

```
NSDictionary *myDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];
NSArray *copiedArray = myDictionary.copy;
```

获取键：

```
NSArray *keys = [myDictionary allKeys];
```

获取值：

```
NSArray *values = [myDictionary allValues];
```

第16.6节：NSDictionary 转 NSData

```
NSDictionary *myDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];
NSData *myData = [NSKeyedArchiver archivedDataWithRootObject:myDictionary];
```

保留路径：

```
NSDictionary *myDictionary = (NSDictionary*) [NSKeyedUnarchiver unarchiveObjectWithData:myData];
```

第16.7节：NSDictionary 转 JSON

```
NSDictionary *myDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];

NSMutableDictionary *mutableDictionary = [myDictionary mutableCopy];
NSData *data = [NSJSONSerialization dataWithJSONObject:myDictionary
options:NSJSONWritingPrettyPrinted error:nil];
NSString *jsonString = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
```

第16.8节：使用plist创建

```
NSString *pathToPlist = [[NSBundle mainBundle] pathForResource:@"plistName"
    ofType:@"plist"];
NSDictionary *plistDict = [[NSDictionary alloc] initWithContentsOfFile:pathToPlist];
```

第16.9节：在NSDictionary中设置值

设置NSDictionary中键对应对象有多种方式，类似于获取值的方式。例如，向汽车列表中添加一辆兰博基尼

标准

nil must be passed as the last parameter as a sentinel signifying the end.

It's important to remember that when instantiating dictionaries this way the values go first and the keys second. In the example above the strings are the keys and the numbers are the values. The method's name reflects this too: dictionaryWithObjectsAndKeys. While this is not incorrect, the more modern way of instantiating dictionaries (with literals) is preferred.

Section 16.5: NSDictionary to NSArray

```
NSDictionary *myDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];
NSArray *copiedArray = myDictionary.copy;
```

Get keys:

```
NSArray *keys = [myDictionary allKeys];
```

Get values:

```
NSArray *values = [myDictionary allValues];
```

Section 16.6: NSDictionary to NSData

```
NSDictionary *myDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];
NSData *myData = [NSKeyedArchiver archivedDataWithRootObject:myDictionary];
```

Reserve path:

```
NSDictionary *myDictionary = (NSDictionary*) [NSKeyedUnarchiver unarchiveObjectWithData:myData];
```

Section 16.7: NSDictionary to JSON

```
NSDictionary *myDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];

NSMutableDictionary *mutableDictionary = [myDictionary mutableCopy];
NSData *data = [NSJSONSerialization dataWithJSONObject:myDictionary
options:NSJSONWritingPrettyPrinted error:nil];
NSString *jsonString = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
```

Section 16.8: Creating using plists

```
NSString *pathToPlist = [[NSBundle mainBundle] pathForResource:@"plistName"
    ofType:@"plist"];
NSDictionary *plistDict = [[NSDictionary alloc] initWithContentsOfFile:pathToPlist];
```

Section 16.9: Setting a Value in NSDictionary

There are multiple ways to set a key's object in an NSDictionary, corresponding to the ways you get a value. For instance, to add a Lamborghini to a list of cars

Standard

```
[cars setObject:lamborghini forKey:@"Lamborghini"];
```

就像其他对象一样，调用NSDictionary中用于设置键对应对象的方法objectForKey:。注意不要将其与setValue:forKey:混淆；后者是完全不同的东西，键值编码（Key Value Coding）

简写

```
cars[@"Lamborghini"] = lamborghini;
```

这是大多数其他语言（如C#、Java和JavaScript）中使用字典的语法。它比标准语法方便得多，而且可以说更易读（尤其是如果你使用这些其他语言编程），但当然，它不是标准语法。它也只在较新版本的Objective-C中可用

第16.10节：从NSDictionary获取值

有多种方法可以通过键从NSDictionary中获取对象。例如，要从汽车列表中获取一辆兰博基尼

标准

```
Car * lamborghini = [cars objectForKey:@"Lamborghini"];
```

就像其他对象一样，调用NSDictionary中用于获取键对应对象的方法objectForKey:。注意不要将其与valueForKey:混淆；后者是完全不同的东西，键值编码（Key Value Coding）

简写

```
汽车*兰博基尼=汽车[@"Lamborghini"];
```

这是大多数其他语言（如C#、Java和JavaScript）中使用字典的语法。它比标准语法方便得多，而且可以说更易读（尤其是如果你使用这些其他语言编程），但当然，它不是标准语法。它也只在较新版本的Objective-C中可用

第16.11节：检查NSDictionary是否已经包含某个键

Objective-C：

```
   //这是你开始使用的字典。
 NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:@"name1", @"Sam",@"name2",
@"Sanju",nil];

//检查字典是否包含你要修改的键。在本例中为@"Sam"
if (dict[@"name1"] != nil) {
    //键name1有对应的条目
}
else {
    //键name1没有对应的条目
}
```

第16.12节：基于Block的枚举

枚举字典允许你使用该方法对每个字典的键值对运行一段代码块
enumerateKeysAndObjectsUsingBlock:(void (^)(id key, id obj, BOOL *stop))block

示例：

```
NSDictionary stockSymbolsDictionary = @{
```

```
[cars setObject:lamborghini forKey:@"Lamborghini"];
```

Just like any other object, call the method of NSDictionary that sets an object of a key, `objectForKey:`. Be careful not to confuse this with `setValue:forKey:`; that's for a completely different thing, Key Value Coding

Shorthand

```
cars[@"Lamborghini"] = lamborghini;
```

This is the syntax that you use for dictionaries in most other languages, such as C#, Java, and JavaScript. It's much more convenient than the standard syntax, and arguably more readable (especially if you code in these other languages), but of course, it isn't *standard*. It's also only available in newer versions of Objective-C

Section 16.10: Getting a Value from NSDictionary

There are multiple ways to get an object from an NSDictionary with a key. For instance, to get a lamborghini from a list of cars

Standard

```
Car * lamborghini = [cars objectForKey:@"Lamborghini"];
```

Just like any other object, call the method of NSDictionary that gives you an object for a key, `objectForKey:`. Be careful not to confuse this with `valueForKey:`; that's for a completely different thing, Key Value Coding

Shorthand

```
Car * lamborghini = cars[@"Lamborghini"];
```

This is the syntax that you use for dictionaries in most other languages, such as C#, Java, and JavaScript. It's much more convenient than the standard syntax, and arguably more readable (especially if you code in these other languages), but of course, it isn't *standard*. It's also only available in newer versions of Objective-C

Section 16.11: Check if NSDictionary already has a key or not

Objective-C:

```
   //this is the dictionary you start with.
 NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:@"name1", @"Sam",@"name2",
@"Sanju",nil];

//check if the dictionary contains the key you are going to modify.  In this example, @"Sam"
if (dict[@"name1"] != nil) {
    //there is an entry for Key name1
}
else {
    //There is no entry for name1
}
```

Section 16.12: Block Based Enumeration

Enumerating dictionaries allows you to run a block of code on each dictionary key-value pair using the method `enumerateKeysAndObjectsUsingBlock:(void (^)(id key, id obj, BOOL *stop))block`

Example:

```
NSDictionary stockSymbolsDictionary = @{
```

```
                                         @"AAPL": @"Apple",
                                         @"GOOGL": @"Alphabet",
                                         @"MSFT": @"Microsoft",
                                         @"AMZN": @"Amazon"
                                       };
NSLog(@"通过枚举打印字典内容");
[stockSymbolsDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    NSLog(@"键: %@, 值: %@", key, obj);
}];
```

```
                                         @"AAPL": @"Apple",
                                         @"GOOGL": @"Alphabet",
                                         @"MSFT": @"Microsoft",
                                         @"AMZN": @"Amazon"
                                       };
NSLog(@"Printing contents of dictionary via enumeration");
[stockSymbolsDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    NSLog(@"Key: %@, Value: %@", key, obj);
}];
```

第17章：NSMutableDictionary

对象	键
包含新字典值的数组。	Cell 一个包含新字典键的数组。每个键都会被复制，复制品会被添加到字典中。

第17.1节：NSMutableDictionary示例

+ dictionaryWithCapacity:

创建并返回一个可变字典，初始时为其分配足够的内存以容纳给定数量的条目。

```
NSMutableDictionary *dict =  [NSMutableDictionary dictionaryWithCapacity:1];
NSLog(@"%@",dict);
```

- init

初始化一个新分配的可变字典。

```
NSMutableDictionary *dict =  [[NSMutableDictionary alloc] init];
NSLog(@"%@",dict);
```

+ dictionaryWithSharedKeySet:

创建一个针对已知键集合进行优化的可变字典。

```
id sharedKeySet = [NSDictionary sharedKeySetForKeys:@[@"key1", @"key2"]]; // 返回 NSSharedKeySet
NSMutableDictionary *dict = [NSMutableDictionary dictionaryWithSharedKeySet:sharedKeySet];
dict[@"key1"] = @"Easy";
dict[@"key2"] = @"Tutorials";
//我们可以使用一个不在共享键集中的对象
dict[@"key3"] = @"Website";
NSLog(@"%@",dict);
```

> 输出

```
{
key1 = Easy;
key2 = Tutorials;
key3 = Website;
}
```

向可变字典添加条目

- setObject:forKey:

向字典中添加给定的键值对。

```
NSMutableDictionary *dict =  [NSMutableDictionary dictionary];
[dict setObject:@"Easy" forKey:@"Key1"];
NSLog(@"%@",dict);
```

Chapter 17: NSMutableDictionary

objects	keys
An array containing the values for the new dictionary.	CellAn array containing the keys for the new dictionary. Each key is copied and the copy is added to the dictionary.

Section 17.1: NSMutableDictionary Example

+ dictionaryWithCapacity:

Creates and returns a mutable dictionary, initially giving it enough allocated memory to hold a given number of entries.

```
NSMutableDictionary *dict =  [NSMutableDictionary dictionaryWithCapacity:1];
NSLog(@"%@",dict);
```

- init

Initializes a newly allocated mutable dictionary.

```
NSMutableDictionary *dict =  [[NSMutableDictionary alloc] init];
NSLog(@"%@",dict);
```

+ dictionaryWithSharedKeySet:

Creates a mutable dictionary which is optimized for dealing with a known set of keys.

```
id sharedKeySet = [NSDictionary sharedKeySetForKeys:@[@"key1", @"key2"]]; // returns NSSharedKeySet
NSMutableDictionary *dict = [NSMutableDictionary dictionaryWithSharedKeySet:sharedKeySet];
dict[@"key1"] = @"Easy";
dict[@"key2"] = @"Tutorials";
//We can an object that is not in the shared keyset
dict[@"key3"] = @"Website";
NSLog(@"%@",dict);
```

> Output

```
{
key1 = Easy;
key2 = Tutorials;
key3 = Website;
}
```

Adding Entries to a Mutable Dictionary

- setObject:forKey:

Adds a given key-value pair to the dictionary.

```
NSMutableDictionary *dict =  [NSMutableDictionary dictionary];
[dict setObject:@"Easy" forKey:@"Key1"];
NSLog(@"%@",dict);
```

> 输出

```
{
Key1 = Eezy;
}
```

- setObject:forKeyedSubscript:

向字典中添加给定的键值对。

```
NSMutableDictionary *dict = [NSMutableDictionary dictionary];
[dict setObject:@"Easy" forKeyedSubscript:@"Key1"];
NSLog(@"%@",dict);
```

> 输出
>
> ```
> {
> Key1 = Easy;
> }
> ```

第17.2节：从可变字典中移除条目

- removeObjectForKey:

从字典中移除指定的键及其关联的值。

```
NSMutableDictionary *dict =  [NSMutableDictionary
dictionaryWithDictionary:@{@"key1":@"Eezy",@"key2": @"Tutorials"}];
[dict removeObjectForKey:@"key1"];
NSLog(@"%@",dict);
```

> 输出

```
{
key2 = Tutorials;
}
```

- removeAllObjects

清空字典中的所有条目。

```
NSMutableDictionary *dict =  [NSMutableDictionary
dictionaryWithDictionary:@{@"key1":@"Eezy",@"key2": @"Tutorials"}];
[dict removeAllObjects];
NSLog(@"%@",dict);
```

> 输出

> Output

```
{
Key1 = Eezy;
}
```

- setObject:forKeyedSubscript:

Adds a given key-value pair to the dictionary.

```
NSMutableDictionary *dict = [NSMutableDictionary dictionary];
[dict setObject:@"Easy" forKeyedSubscript:@"Key1"];
NSLog(@"%@",dict);
```

> Output
>
> ```
> {
> Key1 = Easy;
> }
> ```

Section 17.2: Removing Entries From a Mutable Dictionary

- removeObjectForKey:

Removes a given key and its associated value from the dictionary.

```
NSMutableDictionary *dict =  [NSMutableDictionary
dictionaryWithDictionary:@{@"key1":@"Eezy",@"key2": @"Tutorials"}];
[dict removeObjectForKey:@"key1"];
NSLog(@"%@",dict);
```

> OUTPUT

```
{
    key2 = Tutorials;
}
```

- removeAllObjects

Empties the dictionary of its entries.

```
NSMutableDictionary *dict =  [NSMutableDictionary
dictionaryWithDictionary:@{@"key1":@"Eezy",@"key2": @"Tutorials"}];
[dict removeAllObjects];
NSLog(@"%@",dict);
```

> OUTPUT

```
                                                                    }];

[dict removeObjectsForKeys:@[@"key1"]];
NSLog(@"%@",dict);
```

> 输出

```
{
key2 = Tutorials;
}
```

```
{
}
```

- removeObjectsForKeys:

Removes from the dictionary entries specified by elements in a given array.

```
NSMutableDictionary *dict =  [NSMutableDictionary
dictionaryWithDictionary:@{@"key1":@"Easy",@"key2": @"Tutorials"}];
[dict removeObjectsForKeys:@[@"key1"]];
NSLog(@"%@",dict);
```

> OUTPUT

```
{
    key2 = Tutorials;
}
```

第18章：NSDate

第18.1节：将仅由小时和
分钟组成的NSDate转换为完整的NSDate

在很多情况下，人们只用小时和分钟格式创建了一个NSDate，例如：08:12

这种情况下的缺点是你的NSDate几乎是“裸露”的，你需要做的是
创建：日、月、年、秒和时区，以使该对象能够与其他NSDate类型“协同工作”。

举例来说，假设hourAndMinute是由小时和
分钟格式组成的NSDate类型：

```
NSDateComponents *hourAndMintuteComponents = [calendar components:NSCalendarUnitHour |
NSCalendarUnitMinute
fromDate:hourAndMinute];
NSDateComponents *componentsOfDate = [[NSCalendar currentCalendar] components:NSCalendarUnitDay |
NSCalendarUnitMonth | NSCalendarUnitYear
fromDate:[NSDate date]];

NSDateComponents *components = [[NSDateComponents alloc] init];
[components setDay: componentsOfDate.day];
[components setMonth: componentsOfDate.month];
[components setYear: componentsOfDate.year];
[components setHour: [hourAndMintuteComponents hour]];
[components setMinute: [hourAndMintuteComponents minute]];
[components setSecond: 0];
[calendar setTimeZone: [NSTimeZone defaultTimeZone]];

NSDate *yourFullNSDateObject = [calendar dateFromComponents:components];
```

现在你的对象完全不是“裸”的了。

第18.2节：将NSDate转换为NSString

如果我们有NSDate对象，并且想将其转换为NSString。有不同类型的日期字符串。我们如何做到这一点？非常简单。
只需3个步骤。

1. 创建NSDateFormatter对象

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

2. 设置你想要的字符串日期格式。

```
dateFormatter.dateFormat = @"yyyy-MM-dd 'at' HH:mm";
```

3. 现在，获取格式化后的字符串

```
NSDate *date = [NSDate date]; // 你的 NSDate 对象
NSString *dateString = [dateFormatter stringFromDate:date];
```

这将输出类似如下内容：`2001-01-02 at 13:00`

注意：

Chapter 18: NSDate

Section 18.1: Convert NSDate that is composed from hour and minute (only) to a full NSDate

There are many cases when one has created an NSDate from only an hour and minute format, i.e: 08:12

The downside for this situation is that your NSDate is almost completely "naked" and what you need to do is to create: day, month, year, second and time zone in order to this object to "play along" with other NSDate types.

For the sake of the example let's say that hourAndMinute is the NSDate type that is composed from hour and minute format:

```
NSDateComponents *hourAndMintuteComponents = [calendar components:NSCalendarUnitHour |
NSCalendarUnitMinute
                                                         fromDate:hourAndMinute];
NSDateComponents *componentsOfDate = [[NSCalendar currentCalendar] components:NSCalendarUnitDay |
NSCalendarUnitMonth | NSCalendarUnitYear
                                                                     fromDate:[NSDate date]];

NSDateComponents *components = [[NSDateComponents alloc] init];
[components setDay: componentsOfDate.day];
[components setMonth: componentsOfDate.month];
[components setYear: componentsOfDate.year];
[components setHour: [hourAndMintuteComponents hour]];
[components setMinute: [hourAndMintuteComponents minute]];
[components setSecond: 0];
[calendar setTimeZone: [NSTimeZone defaultTimeZone]];

NSDate *yourFullNSDateObject = [calendar dateFromComponents:components];
```

Now your object is the total opposite of being "naked".

Section 18.2: Converting NSDate to NSString

If ww have NSDate object, and we want to convert it into NSString. There are different types of Date strings. How we can do that?, It is very simple. Just 3 steps.

1. Create NSDateFormatter Object

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

2. Set the date format in which you want your string.

```
dateFormatter.dateFormat = @"yyyy-MM-dd 'at' HH:mm";
```

3. Now, get the formatted string

```
NSDate *date = [NSDate date]; // your NSDate object
NSString *dateString = [dateFormatter stringFromDate:date];
```

This will give output something like this: `2001-01-02 at 13:00`

Note:

创建一个 NSDateFormatter 实例是一个开销较大的操作，因此建议尽可能只创建一次并重复使用
。

第18.3节：创建一个 NSDate

NSDate 类提供了创建对应于给定日期和时间的 NSDate 对象的方法。一个 NSDate
可以使用指定的初始化方法初始化，该方法：

> 返回一个 NSDate 对象，该对象相对于2001年1月1日00:00:00 UTC，通过给定的秒数进行初始化。

```
NSDate *date = [[NSDate alloc] initWithTimeIntervalSinceReferenceDate:100.0];
```

NSDate 还提供了一种简单的方法来创建等于当前日期和时间的 NSDate 对象：

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

也可以创建一个NSDate，表示从当前日期和时间起经过指定秒数的时间点：

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

第18.4节：日期比较

在Objective-C中，有4种方法用于比较NSDate：

- - (BOOL)isEqualToDate:(NSDate *)anotherDate
- - (NSDate *)earlierDate:(NSDate *)anotherDate
- - (NSDate *)laterDate:(NSDate *)anotherDate
- - (NSComparisonResult)compare:(NSDate *)anotherDate

考虑以下使用两个日期的示例，NSDate date1 = 2016年7月7日，NSDate date2 = 2016年7月2日：

```
NSDateComponents *comps1 = [[NSDateComponents alloc]init];
comps.year = 2016;
comps.month = 7;
comps.day = 7;

NSDateComponents *comps2 = [[NSDateComponents alloc]init];
    comps.year = 2016;
comps.month = 7;
comps.day = 2;

NSDate* date1 = [calendar dateFromComponents:comps1]; //初始化为2016年7月7日
NSDate* date2 = [calendar dateFromComponents:comps2]; //初始化为2016年7月2日
```

现在NSDate对象已经创建，可以进行比较：

```
if ([date1 isEqualToDate:date2]) {
    //这里返回false，因为两个日期不相等
}
```

我们也可以使用NSDate类的earlierDate:和laterDate:方法：

```
NSDate *earlierDate = [date1 earlierDate:date2];//返回两个日期中较早的日期。这里是earlierDate
```

Creating an NSDateFormatter instance is an expensive operation, so it is recommended to create it once and reuse when possible.

Section 18.3: Creating an NSDate

The NSDate class provides methods for creating NSDate objects corresponding to a given date and time. An NSDate can be initialized using the designated initializer, which:

> Returns an NSDate object initialized relative to 00:00:00 UTC on 1 January 2001 by a given number of seconds.

```
NSDate *date = [[NSDate alloc] initWithTimeIntervalSinceReferenceDate:100.0];
```

NSDate also provides an easy way to create an NSDate equal to the current date and time:

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

It is also possible to create an NSDate a given amount of seconds from the current date and time:

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

Section 18.4: Date Comparison

There are 4 methods for comparing NSDates in Objective-C:

- - (BOOL)isEqualToDate:(NSDate *)anotherDate
- - (NSDate *)earlierDate:(NSDate *)anotherDate
- - (NSDate *)laterDate:(NSDate *)anotherDate
- - (NSComparisonResult)compare:(NSDate *)anotherDate

Consider the following example using 2 dates, NSDate date1 = July 7, 2016 and NSDate date2 = July 2, 2016:

```
NSDateComponents *comps1 = [[NSDateComponents alloc]init];
comps.year = 2016;
comps.month = 7;
comps.day = 7;

NSDateComponents *comps2 = [[NSDateComponents alloc]init];
    comps.year = 2016;
    comps.month = 7;
    comps.day = 2;

NSDate* date1 = [calendar dateFromComponents:comps1]; //Initialized as July 7, 2016
NSDate* date2 = [calendar dateFromComponents:comps2]; //Initialized as July 2, 2016
```

Now that the NSDates are created, they can be compared:

```
if ([date1 isEqualToDate:date2]) {
    //Here it returns false, as both dates are not equal
}
```

We can also use the earlierDate: and laterDate: methods of the NSDate class:

```
NSDate *earlierDate = [date1 earlierDate:date2];//Returns the earlier of 2 dates. Here earlierDate
```

```
将等于 date2。
NSDate *laterDate = [date1 laterDate:date2];// 返回两个日期中较晚的一个。这里 laterDate 将等于 date1。
```

最后，我们可以使用 NSDate 的 compare: 方法：

```
NSComparisonResult result = [date1 compare:date2];
    if (result == NSOrderedAscending) {
        // 失败
        // 如果 date1 早于 date2，则执行这里。在我们的例子中不会执行这里。
    } else if (result == NSOrderedSame){
        // 失败
        // 如果 date1 与 date2 相同，则执行这里。在我们的例子中不会执行这里。
    } else{// NSOrderedDescending

        // 成功
        //如果date1晚于date2，则会执行这里的代码。在我们的例子中会执行这里
    }
```

```
will equal date2.
NSDate *laterDate = [date1 laterDate:date2];//Returns the later of 2 dates. Here laterDate will
equal date1.
```

Lastly, we can use NSDate's compare: method:

```
NSComparisonResult result = [date1 compare:date2];
    if (result == NSOrderedAscending) {
        //Fails
        //Comes here if date1 is earlier than date2. In our case it will not come here.
    }else if (result == NSOrderedSame){
        //Fails
        //Comes here if date1 is the same as date2. In our case it will not come here.
    }else{//NSOrderedDescending

        //Succeeds
        //Comes here if date1 is later than date2. In our case it will come here
    }
```

第19章：NSURL

第19.1节：创建

从NSString：

```
NSString *urlString = @"https://www.stackoverflow.com";
NSURL *myUrl = [NSURL URLWithString: urlString];
```

你也可以使用以下方法：

```
-initWithString:
+URLWithString:relativeToURL:
-initWithString:relativeToURL:
+fileURLWithPath:是否为目录:
-initFileURLWithPath:是否为目录:
+fileURLWithPath:
-initFileURLWithPath:
指定初始化方法
+fileURLWithPathComponents:
+通过解析别名文件的URL获取URL:选项:错误:
+通过解析书签数据获取URL:选项:相对URL:书签数据是否过时:错误:
-通过解析书签数据初始化:选项:相对URL:书签数据是否过时:错误:
+fileURLWithFileSystemRepresentation:是否为目录:相对URL:
-getFileSystemRepresentation:maxLength:
-initFileURLWithFileSystemRepresentation:isDirectory:relativeToURL:
```

第19.2节：比较NSURL

```
NSString *urlString = @"https://www.stackoverflow.com";

NSURL *myUrl = [NSURL URLWithString: urlString];
NSURL *myUrl2 = [NSURL URLWithString: urlString];

if ([myUrl isEqual:myUrl2]) return YES;
```

第19.3节：修改和转换文件URL，删除和追加路径

1. URLByDeletingPathExtension:

如果接收者表示根路径，则该属性包含原始URL的副本。如果URL有多个路径扩展名，则只删除最后一个。

2. URLByAppendingPathExtension:

返回一个通过向原始URL追加路径扩展名而生成的新URL。

示例：

```
NSUInteger count = 0;
        NSString *filePath = nil;
        do {
            NSString *extension = ( NSString *)UTTypeCopyPreferredTagWithClass((
CFStringRef)AVFileTypeQuickTimeMovie, kUTTagClassFilenameExtension);
            NSString *fileNameNoExtension = [[asset.defaultRepresentation.url
```

Chapter 19: NSURL

Section 19.1: Create

From NSString:

```
NSString *urlString = @"https://www.stackoverflow.com";
NSURL *myUrl = [NSURL URLWithString: urlString];
```

You can also use the following methods:

```
- initWithString:
+ URLWithString:relativeToURL:
- initWithString:relativeToURL:
+ fileURLWithPath:isDirectory:
- initFileURLWithPath:isDirectory:
+ fileURLWithPath:
- initFileURLWithPath:
 Designated Initializer
+ fileURLWithPathComponents:
+ URLByResolvingAliasFileAtURL:options:error:
+ URLByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error:
- initByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error:
+ fileURLWithFileSystemRepresentation:isDirectory:relativeToURL:
- getFileSystemRepresentation:maxLength:
- initFileURLWithFileSystemRepresentation:isDirectory:relativeToURL:
```

Section 19.2: Compare NSURL

```
NSString *urlString = @"https://www.stackoverflow.com";

NSURL *myUrl = [NSURL URLWithString: urlString];
NSURL *myUrl2 = [NSURL URLWithString: urlString];

if ([myUrl isEqual:myUrl2]) return YES;
```

Section 19.3: Modifying and Converting a File URL with removing and appending path

1. URLByDeletingPathExtension:

If the receiver represents the root path, this property contains a copy of the original URL. If the URL has multiple path extensions, only the last one is removed.

2. URLByAppendingPathExtension:

Returns a new URL made by appending a path extension to the original URL.

Example:

```
    NSUInteger count = 0;
        NSString *filePath = nil;
        do {
            NSString *extension = ( NSString *)UTTypeCopyPreferredTagWithClass((
CFStringRef)AVFileTypeQuickTimeMovie, kUTTagClassFilenameExtension);
            NSString *fileNameNoExtension = [[asset.defaultRepresentation.url
URLByDeletingPathExtension] lastPathComponent];//Delete 用法
                NSString *fileName = [NSString stringWithFormat:@"%@-%@-%u",fileNameNoExtension ,
AVAssetExportPresetLowQuality, count];
filePath = NSTemporaryDirectory();
filePath = [filePath stringByAppendingPathComponent:fileName];//Appending 用法
             filePath = [filePath stringByAppendingPathExtension:extension];
count++;

        } while ([[NSFileManager defaultManager] fileExistsAtPath:filePath]);

        NSURL *outputURL = [NSURL fileURLWithPath:filePath];
```

```
URLByDeletingPathExtension] lastPathComponent];//Delete is used
            NSString *fileName = [NSString stringWithFormat:@"%@-%@-%u",fileNameNoExtension ,
AVAssetExportPresetLowQuality, count];
            filePath = NSTemporaryDirectory();
            filePath = [filePath stringByAppendingPathComponent:fileName];//Appending is used
            filePath = [filePath stringByAppendingPathExtension:extension];
            count++;

        } while ([[NSFileManager defaultManager] fileExistsAtPath:filePath]);

        NSURL *outputURL = [NSURL fileURLWithPath:filePath];
```

第20章：NSUrl 发送 POST 请求

第20.1节：简单的 POST 请求

```
// 创建请求。
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL
URLWithString:@"http://google.com"]];

// 指定这是一个POST请求
request.HTTPMethod = @"POST";

// 这是设置请求头字段的方式
[request setValue:@"application/xml; charset=utf-8" forHTTPHeaderField:@"Content-Type"];

// 转换数据并设置请求的HTTPBody属性
NSString *stringData = @"some data";
NSData *requestBodyData = [stringData dataUsingEncoding:NSUTF8StringEncoding];
request.HTTPBody = requestBodyData;

// 创建URL连接并发送请求
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```

第20.2节：带超时的简单POST请求

```
// 创建请求。
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL
URLWithString:@"http://google.com"]];

// 指定这是一个POST请求
request.HTTPMethod = @"POST";

// 设置超时
request.timeoutInterval = 20.0;
// 这是设置请求头字段的方式
[request setValue:@"application/xml; charset=utf-8" forHTTPHeaderField:@"Content-Type"];

// 转换数据并设置请求的HTTPBody属性
NSString *stringData = @"some data";
NSData *requestBodyData = [stringData dataUsingEncoding:NSUTF8StringEncoding];
request.HTTPBody = requestBodyData;

// 创建URL连接并发送请求
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```

Chapter 20: NSUrl send a post request

Section 20.1: Simple POST request

```
// Create the request.
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL
URLWithString:@"http://google.com"]];

// Specify that it will be a POST request
request.HTTPMethod = @"POST";

// This is how we set header fields
[request setValue:@"application/xml; charset=utf-8" forHTTPHeaderField:@"Content-Type"];

// Convert your data and set your request's HTTPBody property
NSString *stringData = @"some data";
NSData *requestBodyData = [stringData dataUsingEncoding:NSUTF8StringEncoding];
request.HTTPBody = requestBodyData;

// Create url connection and fire request
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```

Section 20.2: Simple Post Request With Timeout

```
// Create the request.
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL
URLWithString:@"http://google.com"]];

// Specify that it will be a POST request
request.HTTPMethod = @"POST";

// Setting a timeout
request.timeoutInterval = 20.0;
// This is how we set header fields
[request setValue:@"application/xml; charset=utf-8" forHTTPHeaderField:@"Content-Type"];

// Convert your data and set your request's HTTPBody property
NSString *stringData = @"some data";
NSData *requestBodyData = [stringData dataUsingEncoding:NSUTF8StringEncoding];
request.HTTPBody = requestBodyData;

// Create url connection and fire request
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```

第21章：NSData

第21.1节：创建

从NSString：

```
NSString *str = @"Hello world";
NSData *data = [str dataUsingEncoding:NSUTF8StringEncoding];
```

从整数：

```
int i = 1;
NSData *data = [NSData dataWithBytes: &i length: sizeof(i)];
```

你也可以使用以下方法：

```
+dataWithContentsOfURL:
+dataWithContentsOfURL:options:error:
+dataWithData:
-initWithBase64EncodedData:options:
-initWithBase64EncodedString:options:
-initWithBase64Encoding:
-initWithBytesNoCopy:length:
-initWithBytesNoCopy:length:deallocator:
-initWithBytesNoCopy:length:freeWhenDone:
-initWithContentsOfFile:
-initWithContentsOfFile:options:error:
-initWithContentsOfMappedFile:
-initWithContentsOfURL:
-initWithContentsOfURL:options:error:
-initWithData:
```

第21.2节：NSData与十六进制字符串

从十六进制字符串获取NSData

```
+ (NSData *)dataFromHexString:(NSString *)string
{
    string = [string lowercaseString];
    NSMutableData *data= [NSMutableData new];
    unsigned char whole_byte;
    char byte_chars[3] = {'\0','\0','\0'};
    int i = 0;
    int length = (int) string.length;
    while (i < length-1) {
        char c = [string characterAtIndex:i++];
        if (c < '0' || (c > '9' && c < 'a') || c > 'f')
            continue;
byte_chars[0] = c;
byte_chars[1] = [string characterAtIndex:i++];
        whole_byte = strtol(byte_chars, NULL, 16);
        [data appendBytes:&whole_byte length:1];
    }
    return data;
}
```

从数据获取十六进制字符串：

Chapter 21: NSData

Section 21.1: Create

From NSString:

```
NSString *str = @"Hello world";
NSData *data = [str dataUsingEncoding:NSUTF8StringEncoding];
```

From Int:

```
int i = 1;
NSData *data = [NSData dataWithBytes: &i length: sizeof(i)];
```

You can also use the following methods:

```
+ dataWithContentsOfURL:
+ dataWithContentsOfURL:options:error:
+ dataWithData:
- initWithBase64EncodedData:options:
- initWithBase64EncodedString:options:
- initWithBase64Encoding:
- initWithBytesNoCopy:length:
- initWithBytesNoCopy:length:deallocator:
- initWithBytesNoCopy:length:freeWhenDone:
- initWithContentsOfFile:
- initWithContentsOfFile:options:error:
- initWithContentsOfMappedFile:
- initWithContentsOfURL:
- initWithContentsOfURL:options:error:
- initWithData:
```

Section 21.2: NSData and Hexadecimal String

Get NSData from Hexadecimal String

```
+ (NSData *)dataFromHexString:(NSString *)string
{
    string = [string lowercaseString];
    NSMutableData *data= [NSMutableData new];
    unsigned char whole_byte;
    char byte_chars[3] = {'\0','\0','\0'};
    int i = 0;
    int length = (int) string.length;
    while (i < length-1) {
        char c = [string characterAtIndex:i++];
        if (c < '0' || (c > '9' && c < 'a') || c > 'f')
            continue;
        byte_chars[0] = c;
        byte_chars[1] = [string characterAtIndex:i++];
        whole_byte = strtol(byte_chars, NULL, 16);
        [data appendBytes:&whole_byte length:1];
    }
    return data;
}
```

Get Hexadecimal String from data:

```
+ (NSString *)hexStringForData:(NSData *)data
{
    if (data == nil) {
        return nil;
    }

    NSMutableString *hexString = [NSMutableString string];

    const unsigned char *p = [data bytes];

    for (int i=0; i < [data length]; i++) {
        [hexString appendFormat:@"%02x", *p++];
    }

    return hexString;
}
```

第21.3节：获取NSData长度

```
NSString *filePath = [[NSFileManager defaultManager] pathForRessorce: @"data" ofType:@"txt"];
NSData *data = [NSData dataWithContentsOfFile:filePath];
int len = [data length];
```

第21.4节：使用NSData进行字符串的编码和解码Base64

编码

```
 // 创建一个Base64编码的NSString对象
      NSData *nsdata = [@"iOS Developer Tips encoded in Base64" dataUsingEncoding:NSUTF8StringEncoding];

 // 从NSData对象获取Base64编码的NSString
 NSString *base64Encoded = [nsdata base64EncodedStringWithOptions:0];
 // 打印Base64编码的字符串
NSLog(@"Encoded: %@", base64Encoded);
```

解码：

```
// 从Base64编码字符串获取NSData
NSData *nsdataFromBase64String = [[NSData alloc]initWithBase64EncodedString:base64Encoded
options:0];

// 从 NSData 解码的 NSString
NSString *base64Decoded = [[NSString alloc] initWithData:nsdataFromBase64String
encoding:NSUTF8StringEncoding];
NSLog(@"解码结果: %@", base64Decoded);
```

```
+ (NSString *)hexStringForData:(NSData *)data
{
    if (data == nil) {
        return nil;
    }

    NSMutableString *hexString = [NSMutableString string];

    const unsigned char *p = [data bytes];

    for (int i=0; i < [data length]; i++) {
        [hexString appendFormat:@"%02x", *p++];
    }

    return hexString;
}
```

Section 21.3: Get NSData length

```
NSString *filePath = [[NSFileManager defaultManager] pathForRessorce: @"data" ofType:@"txt"];
NSData *data = [NSData dataWithContentsOfFile:filePath];
int len = [data length];
```

Section 21.4: Encoding and decoding a string using NSData Base64

Encoding

```
 //Create a Base64 Encoded NSString Object
 NSData *nsdata = [@"iOS Developer Tips encoded in Base64" dataUsingEncoding:NSUTF8StringEncoding];

 // Get NSString from NSData object in Base64
 NSString *base64Encoded = [nsdata base64EncodedStringWithOptions:0];
 // Print the Base64 encoded string
 NSLog(@"Encoded: %@", base64Encoded);
```

Decoding:

```
// NSData from the Base64 encoded str
NSData *nsdataFromBase64String = [[NSData alloc]initWithBase64EncodedString:base64Encoded
options:0];

// Decoded NSString from the NSData
NSString *base64Decoded = [[NSString alloc] initWithData:nsdataFromBase64String
encoding:NSUTF8StringEncoding];
NSLog(@"Decoded: %@", base64Decoded);
```

第22章：NSPredicate

第22.1节：按名称过滤

```
NSArray *array = @[
@{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB71",
                        @"title": @"成龙动作电影",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
@{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB72",
                        @"title": @"福尔摩斯",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @0
                    },
@{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB73",
                        @"title": @"泰坦尼克号",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
@{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB74",
                        @"title": @"星球大战",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
@{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB75",
                        @"title": @"宝可梦",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @0
                    },
@{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76",
                        @"title": @"阿凡达",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
@{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB77",
                        @"title": @"大力水手",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
@{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB78",
                        @"title": @"猫和老鼠",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
```

Chapter 22: NSPredicate

Section 22.1: Filter By Name

```
NSArray *array = @[
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB71",
                        @"title": @"Jackie Chan Strike Movie",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB72",
                        @"title": @"Sherlock homes",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @0
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB73",
                        @"title": @"Titanic",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB74",
                        @"title": @"Star Wars",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB75",
                        @"title": @"Pokemon",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @0
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76",
                        @"title": @"Avatar",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB77",
                        @"title": @"Popey",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB78",
                        @"title": @"Tom and Jerry",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                        },
@{
                            @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB79",
                            @"title": @"狼",
                            @"url": @"http://abc.com/playback.m3u8",
                            @"thumbnailURL": @"http://abc.com/thumbnail.png",
                            @"isMovie" : @1
                        }
                        ];

// *** 不区分大小写且完全匹配标题 ***
NSPredicate *filterByNameCIS = [NSPredicate predicateWithFormat:@"self.title LIKE[cd] %@",@"Tom and
Jerry"];
NSLog(@"按名称过滤（不区分大小写）：%@",[array filteredArrayUsingPredicate:filterByNameCIS]);
```

第22.2节：查找除指定ID外的电影

```
// *** 查找除指定ID外的电影 ***
NSPredicate *filterByNotInIds = [NSPredicate predicateWithFormat:@"NOT (self.id IN
%@)",@[@"7CDF6D22-8D36-49C2-84FE-E31EECCECB79", @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76"]];
NSLog(@"过滤除指定ID外的电影: %@",[array filteredArrayUsingPredicate:filterByNotInIds]);
```

第22.3节：查找所有类型为电影的对象

```
// *** 查找所有类型为电影的对象，两种语法均有效 ***
NSPredicate *filterByMovieType = [NSPredicate predicateWithFormat:@"self.isMovie = %@",@1];
// 或者
//NSPredicate *filterByMovieType = [NSPredicate predicateWithFormat:@"self.isMovie = %@",[NSNumber
numberWithBool:YES]];
NSLog(@"按电影类型筛选：%@",[array filteredArrayUsingPredicate:filterByMovieType]);
```

第22.4节：查找数组中不同的对象ID

```
// *** 查找数组中不同的对象ID ***
NSLog(@"不同的ID：%@",[array valueForKeyPath:@"@distinctUnionOfObjects.id"]);
```

第22.5节：查找具有特定ID的电影

```
// *** 查找具有特定ID的电影 ***
NSPredicate *filterByIds = [NSPredicate predicateWithFormat:@"self.id IN
%@",@[@"7CDF6D22-8D36-49C2-84FE-E31EECCECB79", @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76"]];
NSLog(@"按ID筛选：%@",[array filteredArrayUsingPredicate:filterByIds]);
```

第22.6节：不区分大小写的精确标题匹配

```
// *** 不区分大小写且完全匹配标题 ***
NSPredicate *filterByNameCIS = [NSPredicate predicateWithFormat:@"self.title LIKE[cd] %@",@"Tom and
Jerry"];
NSLog(@"按名称过滤（不区分大小写）：%@",[array filteredArrayUsingPredicate:filterByNameCIS]);
```

第22.7节：区分大小写的精确标题匹配

```
// *** 区分大小写的精确标题匹配 ***
NSPredicate *filterByNameCS = [NSPredicate predicateWithFormat:@"self.title = %@",@"Tom and
```

```
                        },
                        @{
                            @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB79",
                            @"title": @"The wolf",
                            @"url": @"http://abc.com/playback.m3u8",
                            @"thumbnailURL": @"http://abc.com/thumbnail.png",
                            @"isMovie" : @1
                        }
                        ];

// *** Case Insensitive comparison with exact title match ***
NSPredicate *filterByNameCIS = [NSPredicate predicateWithFormat:@"self.title LIKE[cd] %@",@"Tom and
Jerry"];
NSLog(@"Filter By Name(CIS) : %@",[array filteredArrayUsingPredicate:filterByNameCIS]);
```

Section 22.2: Find movies except given ids

```
// *** Find movies except given ids ***
NSPredicate *filterByNotInIds = [NSPredicate predicateWithFormat:@"NOT (self.id IN
%@)",@[@"7CDF6D22-8D36-49C2-84FE-E31EECCECB79", @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76"]];
NSLog(@"Filter movies except given Ids : %@",[array filteredArrayUsingPredicate:filterByNotInIds]);
```

Section 22.3: Find all the objects which is of type movie

```
// *** Find all the objects which is of type movie, Both the syntax are valid ***
NSPredicate *filterByMovieType = [NSPredicate predicateWithFormat:@"self.isMovie = %@",@1];
// OR
//NSPredicate *filterByMovieType = [NSPredicate predicateWithFormat:@"self.isMovie = %@",[NSNumber
numberWithBool:YES]];
NSLog(@"Filter By Movie Type : %@",[array filteredArrayUsingPredicate:filterByMovieType]);
```

Section 22.4: Find Distinct object ids of array

```
// *** Find Distinct object ids of array ***
NSLog(@"Distinct id : %@",[array valueForKeyPath:@"@distinctUnionOfObjects.id"]);
```

Section 22.5: Find movies with specific ids

```
// *** Find movies with specific ids ***
NSPredicate *filterByIds = [NSPredicate predicateWithFormat:@"self.id IN
%@",@[@"7CDF6D22-8D36-49C2-84FE-E31EECCECB79", @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76"]];
NSLog(@"Filter By Ids : %@",[array filteredArrayUsingPredicate:filterByIds]);
```

Section 22.6: Case Insensitive comparison with exact title match

```
// *** Case Insensitive comparison with exact title match ***
NSPredicate *filterByNameCIS = [NSPredicate predicateWithFormat:@"self.title LIKE[cd] %@",@"Tom and
Jerry"];
NSLog(@"Filter By Name(CIS) : %@",[array filteredArrayUsingPredicate:filterByNameCIS]);
```

Section 22.7: Case sensitive with exact title match

```
// *** Case sensitive with exact title match ***
NSPredicate *filterByNameCS = [NSPredicate predicateWithFormat:@"self.title = %@",@"Tom and
Jerry"];
NSLog(@"按名称过滤（区分大小写）：%@",[array filteredArrayUsingPredicate:filterByNameCS]);
```

第22.8节：不区分大小写的比较与匹配子集

```
// *** 不区分大小写的比较与匹配子集 ***
NSPredicate *filterByName = [NSPredicate predicateWithFormat:@"self.title CONTAINS[cd] %@",@"Tom"];
NSLog(@"按包含名称过滤：%@",[array filteredArrayUsingPredicate:filterByName]);
```

```
Jerry"];
NSLog(@"Filter By Name(CS) : %@",[array filteredArrayUsingPredicate:filterByNameCS]);
```

Section 22.8: Case Insensitive comparison with matching subset

```
// *** Case Insensitive comparison with matching subset ***
NSPredicate *filterByName = [NSPredicate predicateWithFormat:@"self.title CONTAINS[cd] %@",@"Tom"];
NSLog(@"Filter By Containing Name : %@",[array filteredArrayUsingPredicate:filterByName]);
```

第23章：NSRegularExpression

第23.1节：检查字符串是否匹配某个模式

```
NSString *testString1 = @"(555) 123-5678";
NSString *testString2 = @"不是电话号码";

NSError *error = nil;
NSRegularExpression *regex = [NSRegularExpression regularExpressionWithPattern:@"^\\(\\d{3}\\)
\\d{3}\\-\\d{4}$"

options:NSRegularExpressionCaseInsensitive error:&error];

NSInteger result1 = [regex numberOfMatchesInString:testString1 options:0 range:NSMakeRange(0,
testString1.length)];
NSInteger result2 = [regex numberOfMatchesInString:testString2 options:0 range:NSMakeRange(0,
testString2.length)];

NSLog(@"字符串1是电话号码吗？ %@", result1 > 0 ? @"是" : @"否");
NSLog(@"字符串2是电话号码吗？ %@", result2 > 0 ? @"是" : @"否");
```

输出将显示第一个字符串是电话号码，第二个不是。

第23.2节：查找字符串中的所有数字

```
NSString *testString = @"有42只羊和8672头牛。";
NSError *error = nil;
NSRegularExpression *regex = [NSRegularExpression regularExpressionWithPattern:@"(\\d+)"

 options:NSRegularExpressionCaseInsensitive
error:&error];

NSArray *matches = [regex matchesInString:testString
options:0
                                    range:NSMakeRange(0, testString.length)];

for (NSTextCheckingResult *matchResult in matches) {
    NSString* match = [testString substringWithRange:matchResult.range];
    NSLog(@"匹配项: %@", match);
}
```

输出将是 匹配项: 42 和 匹配项: 8672。

Chapter 23: NSRegularExpression

Section 23.1: Check whether a string matches a pattern

```
NSString *testString1 = @"(555) 123-5678";
NSString *testString2 = @"not a phone number";

NSError *error = nil;
NSRegularExpression *regex = [NSRegularExpression regularExpressionWithPattern:@"^\\(\\d{3}\\)
\\d{3}\\-\\d{4}$"

 options:NSRegularExpressionCaseInsensitive error:&error];

NSInteger result1 = [regex numberOfMatchesInString:testString1 options:0 range:NSMakeRange(0,
testString1.length)];
NSInteger result2 = [regex numberOfMatchesInString:testString2 options:0 range:NSMakeRange(0,
testString2.length)];

NSLog(@"Is string 1 a phone number? %@", result1 > 0 ? @"YES" : @"NO");
NSLog(@"Is string 2 a phone number? %@", result2 > 0 ? @"YES" : @"NO");
```

The output will show that the first string is a phone number and the second one isn't.

Section 23.2: Find all the numbers in a string

```
NSString *testString = @"There are 42 sheep and 8672 cows.";
NSError *error = nil;
NSRegularExpression *regex = [NSRegularExpression regularExpressionWithPattern:@"(\\d+)"

 options:NSRegularExpressionCaseInsensitive
                                                                         error:&error];

NSArray *matches = [regex matchesInString:testString
                                  options:0
                                    range:NSMakeRange(0, testString.length)];

for (NSTextCheckingResult *matchResult in matches) {
    NSString* match = [testString substringWithRange:matchResult.range];
    NSLog(@"match: %@", match);
}
```

The output will be `match: 42` and `match: 8672`.

第24章：NSJSONSerialization

操作符	描述
数据	包含JSON数据的数据对象
选项	用于读取JSON数据和创建Foundation对象的选项。
错误	如果发生错误，返回时包含描述问题的NSError对象。

第24.1节：使用NSJSONSerialization解析JSON Objective-C

```
NSError *e = nil;
NSString *jsonString = @"[{\"id\": \"1\", \"name\":\"sam\"}]";
NSData *data = [jsonString dataUsingEncoding:NSUTF8StringEncoding];

NSArray *jsonArray = [NSJSONSerialization JSONObjectWithData: data options:
NSJSONReadingMutableContainers 错误: &e];

if (!jsonArray) {
NSLog(@"解析JSON时出错: %@", e);
} else {
    for(NSDictionary *item in jsonArray) {
        NSLog(@"项目: %@", item);
    }
}
```

输出：

```
项目: {
id = 1;
name = sam;
}
```

示例2：使用URL内容：

```
//解析：

NSData *data = [NSData dataWithContentsOfURL:@"URL HERE"];
NSError *error;
NSDictionary *json = [NSJSONSerialization JSONObjectWithData:data options:kNilOptions
error:&error];
NSLog(@"json :%@",json);
```

示例响应：

```
json: {
MESSAGE = "测试信息";
    RESPONSE =(
                {
email = "test@gmail.com";
            id = 15;
phone = 1234567890;
            name = Staffy;
        }
    );
STATUS = SUCCESS;
```

Chapter 24: NSJSONSerialization

Operator	Description
data	A data object containing JSON data
opt	Options for reading the JSON data and creating the Foundation objects.
error	If an error occurs, upon return contains an NSError object that describes the problem.

Section 24.1: JSON Parsing using NSJSONSerialization Objective-C

```
NSError *e = nil;
NSString *jsonString = @"[{\"id\": \"1\", \"name\":\"sam\"}]";
NSData *data = [jsonString dataUsingEncoding:NSUTF8StringEncoding];

NSArray *jsonArray = [NSJSONSerialization JSONObjectWithData: data options:
 NSJSONReadingMutableContainers error: &e];

if (!jsonArray) {
    NSLog(@"Error parsing JSON: %@", e);
} else {
    for(NSDictionary *item in jsonArray) {
        NSLog(@"Item: %@", item);
    }
}
```

Output:

```
Item: {
id = 1;
name = sam;
}
```

Example 2: Using contents of url:

```
//Parsing:

NSData *data = [NSData dataWithContentsOfURL:@"URL HERE"];
NSError *error;
NSDictionary *json = [NSJSONSerialization JSONObjectWithData:data options:kNilOptions
error:&error];
NSLog(@"json :%@",json);
```

Sample response:

```
json: {
    MESSAGE = "Test Message";
    RESPONSE =(
                {
            email = "test@gmail.com";
            id = 15;
            phone = 1234567890;
            name = Staffy;
        }
    );
    STATUS = SUCCESS;
}


NSMutableDictionary *response = [[[json valueForKey:@"RESPONSE"] objectAtIndex:0]mutableCopy];
NSString *nameStr = [response valueForKey:@"name"];
NSString *emailIdStr = [response valueForKey:@"email"];
```

```
}


NSMutableDictionary *response = [[[json valueForKey:@"RESPONSE"] objectAtIndex:0]mutableCopy];
NSString *nameStr = [response valueForKey:@"name"];
NSString *emailIdStr = [response valueForKey:@"email"];
```

第25章：NSCalendar

第25.1节：系统区域信息

+currentCalendar 返回当前用户的逻辑日历。

```
    NSCalendar *calendar = [NSCalendar currentCalendar];
    NSLog(@"%@",calendar);
```

+autoupdatingCurrentCalendar 返回当前用户的当前逻辑日历。

```
NSCalendar *calendar = [NSCalendar autoupdatingCurrentCalendar];
NSLog(@"%@",calendar);
```

第25.2节：初始化日历

-initWithCalendarIdentifier: 初始化一个新分配的NSCalendar对象，使用指定的日历标识符。
给定的标识符。

```
NSCalendar *calender = [[NSCalendar alloc]initWithCalendarIdentifier:@"gregorian"];
NSLog(@"%@",calender);
```

-setFirstWeekday: 设置接收者的第一天星期的索引。

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setFirstWeekday:1];
NSLog(@"%d",[calender firstWeekday]);
```

-setLocale: 设置接收者的区域设置。

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setLocale:[NSLocale currentLocale]];
NSLog(@"%@",[calender locale]);
```

-setMinimumDaysInFirstWeek: 设置接收者第一周的最小天数。

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setMinimumDaysInFirstWeek:7];
NSLog(@"%d",[calender minimumDaysInFirstWeek]);
```

-setTimeZone: 设置接收者的时区。

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
NSLog(@"%@",[calender timeZone]);
```

第25.3节：历法计算

- components:fromDate: 返回一个NSDateComponents对象，包含分解为指定部分的给定日期
组件

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
NSLog(@"%@",[calender components:NSCalendarUnitDay fromDate:[NSDate date]]);
```

Chapter 25: NSCalendar

Section 25.1: System Locale Information

+currentCalendar returns the logical calendar for the current user.

```
    NSCalendar *calendar = [NSCalendar currentCalendar];
    NSLog(@"%@",calendar);
```

+autoupdatingCurrentCalendar returns the current logical calendar for the current user.

```
NSCalendar *calendar = [NSCalendar autoupdatingCurrentCalendar];
NSLog(@"%@",calendar);
```

Section 25.2: Initializing a Calendar

- initWithCalendarIdentifier: Initializes a newly-allocated NSCalendar object for the calendar specified by a given identifier.

```
NSCalendar *calender = [[NSCalendar alloc]initWithCalendarIdentifier:@"gregorian"];
NSLog(@"%@",calender);
```

- setFirstWeekday: Sets the index of the first weekday for the receiver.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setFirstWeekday:1];
NSLog(@"%d",[calender firstWeekday]);
```

- setLocale: Sets the locale for the receiver.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setLocale:[NSLocale currentLocale]];
NSLog(@"%@",[calender locale]);
```

- setMinimumDaysInFirstWeek: Sets the minimum number of days in the first week of the receiver.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setMinimumDaysInFirstWeek:7];
NSLog(@"%d",[calender minimumDaysInFirstWeek]);
```

- setTimeZone: Sets the time zone for the receiver.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
NSLog(@"%@",[calender timeZone]);
```

Section 25.3: Calendrical Calculations

- components:fromDate: Returns a NSDateComponents object containing a given date decomposed into specified components

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
NSLog(@"%@",[calender components:NSCalendarUnitDay fromDate:[NSDate date]]);
NSLog(@"%@",[calender components:NSCalendarUnitYear fromDate:[NSDate date]]);
NSLog(@"%@",[calender components:NSCalendarUnitMonth fromDate:[NSDate date]]);
```

- components:fromDate:toDate:options: 返回一个 NSDateComponents 对象，使用指定的组件，两个提供的日期之间的差异。

```
 NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
NSLog(@"%@",[calender components:NSCalendarUnitYear fromDate:[NSDate
dateWithTimeIntervalSince1970:0] toDate:[NSDate dateWithTimeIntervalSinceNow:18000]
options:NSCalendarWrapComponents]);
```

-dateByAddingComponents:toDate:options: 返回一个新的 NSDate 对象，表示通过将给定组件加到给定日期上计算出的绝对时间。calculated by adding given components to a given date.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
NSDateComponents *dateComponent = [[NSDateComponents alloc]init];
[dateComponent setYear:10];
NSLog(@"%@",[calender dateByAddingComponents:dateComponent toDate:[NSDate
dateWithTimeIntervalSinceNow:0] options:NSCalendarWrapComponents] );
```

-dateFromComponents: 返回一个新的NSDate对象，表示根据给定组件计算的绝对时间。

```
NSCalendar *日历 = [NSCalendar 自动更新当前日历];
NSDateComponents *日期组件 = [[NSDateComponents 分配]初始化];
[日期组件 设置年份:2020];
NSLog(@"%@",[日历 dateFromComponents:日期组件]);
```

```
NSLog(@"%@",[calender components:NSCalendarUnitYear fromDate:[NSDate date]]);
NSLog(@"%@",[calender components:NSCalendarUnitMonth fromDate:[NSDate date]]);
```

- `components:fromDate:toDate:options:` Returns, as an NSDateComponents object using specified components, the difference between two supplied dates.

```
 NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
NSLog(@"%@",[calender components:NSCalendarUnitYear fromDate:[NSDate
dateWithTimeIntervalSince1970:0] toDate:[NSDate dateWithTimeIntervalSinceNow:18000]
options:NSCalendarWrapComponents]);
```

- `dateByAddingComponents:toDate:options:` Returns a new NSDate object representing the absolute time calculated by adding given components to a given date.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
NSDateComponents *dateComponent = [[NSDateComponents alloc]init];
[dateComponent setYear:10];
NSLog(@"%@",[calender dateByAddingComponents:dateComponent toDate:[NSDate
dateWithTimeIntervalSinceNow:0] options:NSCalendarWrapComponents] );
```

- `dateFromComponents:` Returns a new NSDate object representing the absolute time calculated from given components.

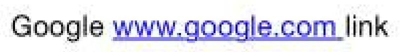
```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
NSDateComponents *dateComponent = [[NSDateComponents alloc]init];
[dateComponent setYear:2020];
NSLog(@"%@",[calender dateFromComponents:dateComponent]);
```

第26章：NSAttributedString

第26.1节：使用枚举遍历字符串中的属性并为字符串部分添加下划线

```
 NSMutableDictionary *属性字典 = [NSMutableDictionary 字典];
 [属性字典 设置对象:[UIFont 系统字体大小:14] 对应键:NSFontAttributeName];
 //[属性字典 设置对象:[UIColor 红色] 对应键:NSForegroundColorAttributeName];
 NSMutableAttributedString *属性字符串 = [[NSMutableAttributedString
分配]用字符串初始化:@"Google www.google.com 链接" 属性:属性字典];

 [属性字符串 枚举属性:(NSString *) NSFontAttributeName
                         范围:NSMakeRange(0, [属性字符串 长度])
                         选项:NSAttributedStringEnumerationLongestEffectiveRangeNotRequired
                       使用块:^(id 值, NSRange 范围, BOOL *停止) {
NSLog(@"属性: %@, %@", value, NSStringFromRange(range));
                              }];

  NSMutableAttributedString *attributedStr = [[NSMutableAttributedString alloc]
initWithString:@"www.google.com "];

  [attributedString addAttribute:NSUnderlineStyleAttributeName
                          value:[NSNumber numberWithInt:NSUnderlineStyleDouble]
                          range:NSMakeRange(7, attributedStr.length)];

  [attributedString addAttribute:NSForegroundColorAttributeName
                          value:[UIColor blueColor]
range:NSMakeRange(6,attributedStr.length)];

  _attriLbl.attributedText = attributedString;//_attriLbl (类型为 UILabel) 在故事板中添加
```

输出：


Google www.google.com link

第26.2节：创建具有自定义字距（字母间距）的字符串 editshare

NSAttributedString（及其可变子类 NSMutableAttributedString）允许你创建在外观上对用户来说复杂的字符串。

一个常见的应用是用它来显示字符串并添加自定义字距/字母间距。

实现方法如下（label 是一个 UILabel），为单词“kerning”设置不同的字距

```
NSMutableAttributedString *attributedString;
attributedString = [[NSMutableAttributedString alloc] initWithString:@"Apply kerning"];
[attributedString addAttribute:NSKernAttributeName value:@5 range:NSMakeRange(6, 7)];
[label setAttributedText:attributedString];
```

第26.3节：创建带有删除线的字符串

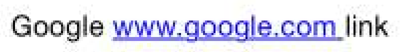
```
NSMutableAttributedString *attributeString = [[NSMutableAttributedString alloc]
initWithString:@"Your String here"];
[attributeString addAttribute:NSStrikethroughStyleAttributeName
```

Chapter 26: NSAttributedString

Section 26.1: Using Enumerating over Attributes in a String and underline part of string

```
 NSMutableDictionary *attributesDictionary = [NSMutableDictionary dictionary];
 [attributesDictionary setObject:[UIFont systemFontOfSize:14] forKey:NSFontAttributeName];
 //[attributesDictionary setObject:[UIColor redColor] forKey:NSForegroundColorAttributeName];
 NSMutableAttributedString *attributedString = [[NSMutableAttributedString
alloc]initWithString:@"Google www.google.com link" attributes:attributesDictionary];

 [attributedString enumerateAttribute:(NSString *) NSFontAttributeName
                              inRange:NSMakeRange(0, [attributedString length])
                              options:NSAttributedStringEnumerationLongestEffectiveRangeNotRequired
                           usingBlock:^(id value, NSRange range, BOOL *stop) {
                               NSLog(@"Attribute: %@, %@", value, NSStringFromRange(range));
                              }];

  NSMutableAttributedString *attributedStr = [[NSMutableAttributedString alloc]
initWithString:@"www.google.com "];

  [attributedString addAttribute:NSUnderlineStyleAttributeName
                          value:[NSNumber numberWithInt:NSUnderlineStyleDouble]
                          range:NSMakeRange(7, attributedStr.length)];

  [attributedString addAttribute:NSForegroundColorAttributeName
                          value:[UIColor blueColor]
                          range:NSMakeRange(6,attributedStr.length)];

  _attriLbl.attributedText = attributedString;//_attriLbl (of type UILabel) added in storyboard
```

Output:


Google www.google.com link

Section 26.2: Creating a string that has custom kerning (letter spacing) editshare

NSAttributedString (and its mutable sibling NSMutableAttributedString) allows you to create strings that are complex in their appearance to the user.

A common application is to use this to display a string and adding custom kerning / letter-spacing.

This would be achieved as follows (where label is a UILabel), giving a different kerning for the word "kerning"

```
NSMutableAttributedString *attributedString;
attributedString = [[NSMutableAttributedString alloc] initWithString:@"Apply kerning"];
[attributedString addAttribute:NSKernAttributeName value:@5 range:NSMakeRange(6, 7)];
[label setAttributedText:attributedString];
```

Section 26.3: Create a string with text struck through

```
NSMutableAttributedString *attributeString = [[NSMutableAttributedString alloc]
initWithString:@"Your String here"];
[attributeString addAttribute:NSStrikethroughStyleAttributeName
value:@2
range:NSMakeRange(0, [attributeString  length])];
```

第26.4节：如何创建三色属性字符串

```
 NSMutableAttributedString * string = [[NSMutableAttributedString alloc]
initWithString:@"firstsecondthird"];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor redColor]
range:NSMakeRange(0,5)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor greenColor]
range:NSMakeRange(5,6)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor blueColor]
range:NSMakeRange(11,5)];
```

范围：起始到结束字符串

这里我们有 firstsecondthird 字符串，所以在 first 中我们设置了范围 (0,5)，因此从第一个字符到第五个字符将以绿色文本颜色显示。

```
                              value:@2
                              range:NSMakeRange(0, [attributeString length])];
```

Section 26.4: How you create a tri-color attributed string

```
 NSMutableAttributedString * string = [[NSMutableAttributedString alloc]
initWithString:@"firstsecondthird"];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor redColor]
range:NSMakeRange(0,5)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor greenColor]
range:NSMakeRange(5,6)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor blueColor]
range:NSMakeRange(11,5)];
```

Range : start to end string

Here we have firstsecondthird string so in first we have set range (0,5) so from starting first character to fifth character it will display in green text color.

第27章：NSTimer

第27.1节：在定时器中存储信息

创建定时器时，可以设置userInfo参数，以包含你想传递给定时器调用的函数的信息。

通过在该函数中将定时器作为参数传入，可以访问userInfo属性。

```
NSDictionary *字典 = @{
                                   @“Message”：@“你好，世界！”
                                 }；//这个字典包含一条消息
[NSTimer scheduledTimerWithTimeInterval:5.0
    target:self
selector:@selector(doSomething)
     userInfo:字典
repeats:NO]; //定时器包含该字典，稍后调用该函数

...

- (void) doSomething:(NSTimer*)定时器{
    //该函数从定时器中获取消息
NSLog("%@", timer.userInfo["Message"]);
}
```

第27.2节：创建定时器

这将创建一个定时器，在5.0秒后调用self的doSomething方法。

```
[NSTimer scheduledTimerWithTimeInterval:5.0
         target:self
selector:@selector(doSomething)
         userInfo:nil
repeats:NO];
```

将repeats参数设置为false/NO表示我们希望定时器只触发一次。如果设置为true/YES，则定时器会每五秒触发一次，直到手动失效。

第27.3节：使定时器失效

```
[timer invalidate];
timer = nil;
```

这将停止计时器触发。必须从创建计时器的线程中调用，请参阅苹果的说明：

> 您必须从安装计时器的线程发送此消息。如果您从另一个线程发送此消息，计时器关联的输入源可能不会从其运行循环中移除，这可能会导致线程无法正常退出。

将 nil 设置为下一步检查它是否正在运行会有所帮助。

```
if(计时器) {
    [计时器 失效];
   计时器 = nil;
```

Chapter 27: NSTimer

Section 27.1: Storing information in the Timer

When creating a timer, you can set the userInfo parameter to include information that you want to pass to the function you call with the timer.

By taking a timer as a parameter in said function, you can access the userInfo property.

```
NSDictionary *dictionary = @{
                             @"Message" : @"Hello, world!"
                            }; //this dictionary contains a message
[NSTimer scheduledTimerWithTimeInterval:5.0
     target:self
     selector:@selector(doSomething)
     userInfo:dictionary
     repeats:NO]; //the timer contains the dictionary and later calls the function

...

- (void) doSomething:(NSTimer*)timer{
    //the function retrieves the message from the timer
    NSLog("%@", timer.userInfo["Message"]);
}
```

Section 27.2: Creating a Timer

This will create a timer to call the doSomething method on self in 5.0 seconds.

```
[NSTimer scheduledTimerWithTimeInterval:5.0
        target:self
        selector:@selector(doSomething)
        userInfo:nil
        repeats:NO];
```

Setting the repeats parameter to false/NO indicates that we want the timer to fire only once. If we set this to true/YES, it would fire every five seconds until manually invalidated.

Section 27.3: Invalidating a timer

```
[timer invalidate];
timer = nil;
```

This will stop the timer from firing. **Must be called from the thread the timer was created in,** see Apple's notes:

> You must send this message from the thread on which the timer was installed. If you send this message from another thread, the input source associated with the timer may not be removed from its run loop, which could prevent the thread from exiting properly.

Setting nil will help you next to check whether it's running or not.

```
if(timer) {
    [timer invalidate];
    timer = nil;
}

//现在再次设置计时器。
```

第27.4节：手动触发计时器

```
[计时器 触发];
```

调用触发方法会使NSTimer执行它通常按计划执行的任务。

对于非重复计时器，这将自动使计时器失效。也就是说，在时间间隔结束前调用触发将只导致一次调用。

对于重复计时器，这将仅调用动作而不会中断正常的计划。

```
}

//Now set a timer again.
```

Section 27.4: Manually firing a timer

```
[timer fire];
```

Calling the `fire` method causes an NSTimer to perform the task it would have usually performed on a schedule.

In a **non-repeating timer**, this will automatically invalidate the timer. That is, calling `fire` before the time interval is up will result in only one invocation.

In a **repeating timer**, this will simply invoke the action without interrupting the usual schedule.

第28章：NSObject

NSObject是Cocoa的根类，然而Objective-C语言本身并不定义任何根类这是由苹果的框架Cocoa定义的。这个根类是大多数Objective-C类层次结构的根基，子类从中继承了对运行时系统的基本接口以及作为Objective-C对象的行为能力。

该类具有所有Objective'C类对象的基本属性，例如：

self。

class（类名）。

superclass（当前类的父类）。

第28.1节：NSObject

@interface NSString : NSObject（NSObject是NSString类的基类）。

你可以使用以下方法来分配字符串类：

```
- (instancetype)init

+ (instancetype)new

+ (instancetype)alloc
```

复制任意对象：

```
- (id)copy;

- (id)mutableCopy;
```

比较对象：

```
- (BOOL)isEqual:(id)object
```

获取当前类的父类：

```
superclass
```

检查这是什么类型的类？

```
- (BOOL)isKindOfClass:(Class)aClass
```

NON-ARC 类的一些属性：

```
- (instancetype)retain OBJC_ARC_UNAVAILABLE;

- (oneway void)release OBJC_ARC_UNAVAILABLE;

- (instancetype)autorelease OBJC_ARC_UNAVAILABLE;

- (NSUInteger)retainCount
```

Chapter 28: NSObject

NSObject is the root class of Cocoa, however the Objective-C language itself does not define any root classes at all its define by Cocoa, Apple's Framework. This root class of most Objective-C class hierarchies, from which subclasses inherit a basic interface to the runtime system and the ability to behave as Objective-C objects.

This class have all basic property of Objective'C class object like:

self.

class (name of the class).

superclass (superclass of current class).

Section 28.1: NSObject

@interface NSString : NSObject (NSObject is a base class of NSString class).

You can use below methods for allocation of string class:

```
- (instancetype)init

+ (instancetype)new

+ (instancetype)alloc
```

For Copy any object :

```
- (id)copy;

- (id)mutableCopy;
```

For compare objects :

```
- (BOOL)isEqual:(id)object
```

To get superclass of current class :

```
superclass
```

To check which kind of class is this ?

```
- (BOOL)isKindOfClass:(Class)aClass
```

Some property of NON-ARC classes:

```
- (instancetype)retain OBJC_ARC_UNAVAILABLE;

- (oneway void)release OBJC_ARC_UNAVAILABLE;

- (instancetype)autorelease OBJC_ARC_UNAVAILABLE;

- (NSUInteger)retainCount
```

第29章：NSSortDescriptor

第29.1节：通过 NSSortDescriptor 组合排序

```
NSArray *aryFName = @[ @"Alice", @"Bob", @"Charlie", @"Quentin" ];
NSArray *aryLName = @[ @"Smith", @"Jones", @"Smith", @"Alberts" ];
NSArray *aryAge = @[ @24, @27, @33, @31 ];

// 创建一个自定义类，包含 firstName 和 lastName 属性，类型为 NSString *,
// 以及 age，类型为 NSUInteger。

NSMutableArray *aryPerson = [NSMutableArray array];
[firstNames enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
    Person *person = [[Person alloc] init];
person.firstName = [aryFName objectAtIndex:idx];
    person.lastName = [aryLName objectAtIndex:idx];
    person.age = [aryAge objectAtIndex:idx];
    [aryPerson 添加对象:person];
}];

NSSortDescriptor *firstNameSortDescriptor = [NSSortDescriptor 使用键:@"firstName"
                                     升序:YES
选择器:@selector(localizedStandardCompare:)];

NSSortDescriptor *lastNameSortDescriptor = [NSSortDescriptor 使用键:@"lastName"
                                     升序:YES
选择器:@selector(localizedStandardCompare:)];

NSSortDescriptor *ageSortDescriptor = [NSSortDescriptor 使用键:@"age"
降序:NO];

NSLog(@"按年龄排序: %@", [aryPerson 使用排序描述符排序:@[ageSortDescriptor]]);
// "查理·史密斯", "昆廷·阿尔伯茨", "鲍勃·琼斯", "爱丽丝·史密斯"


NSLog(@"按名字排序: %@", [aryPerson 使用排序描述符排序:@[firstNameSortDescriptor]]);
// "爱丽丝·史密斯", "鲍勃·琼斯", "查理·史密斯", "昆廷·阿尔伯茨"


NSLog(@"按姓氏和名字排序: %@", [aryPerson
使用排序描述符排序:@[lastNameSortDescriptor, firstNameSortDescriptor]]);
// "昆廷·阿尔伯茨", "鲍勃·琼斯", "爱丽丝·史密斯", "查理·史密斯"
```

Chapter 29: NSSortDescriptor

Section 29.1: Sorted by combinations of NSSortDescriptor

```
NSArray *aryFName = @[ @"Alice", @"Bob", @"Charlie", @"Quentin" ];
NSArray *aryLName = @[ @"Smith", @"Jones", @"Smith", @"Alberts" ];
NSArray *aryAge = @[ @24, @27, @33, @31 ];

//Create a Custom class with properties for firstName & lastName of type NSString *,
//and age, which is an NSUInteger.

NSMutableArray *aryPerson = [NSMutableArray array];
[firstNames enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
    Person *person = [[Person alloc] init];
    person.firstName = [aryFName objectAtIndex:idx];
    person.lastName = [aryLName objectAtIndex:idx];
    person.age = [aryAge objectAtIndex:idx];
    [aryPerson addObject:person];
}];

NSSortDescriptor *firstNameSortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"firstName"
                                             ascending:YES
                                             selector:@selector(localizedStandardCompare:)];

NSSortDescriptor *lastNameSortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"lastName"
                                            ascending:YES
                                            selector:@selector(localizedStandardCompare:)];

NSSortDescriptor *ageSortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"age"
                                       ascending:NO];

NSLog(@"By age: %@", [aryPerson sortedArrayUsingDescriptors:@[ageSortDescriptor]]);
// "Charlie Smith", "Quentin Alberts", "Bob Jones", "Alice Smith"


NSLog(@"By first name: %@", [aryPerson sortedArrayUsingDescriptors:@[firstNameSortDescriptor]]);
// "Alice Smith", "Bob Jones", "Charlie Smith", "Quentin Alberts"


NSLog(@"By last name, first name: %@", [aryPerson
sortedArrayUsingDescriptors:@[lastNameSortDescriptor, firstNameSortDescriptor]]);
// "Quentin Alberts", "Bob Jones", "Alice Smith", "Charlie Smith"
```

第30章：NSTextAttachment

第30.1节：NSTextAttachment示例

```
NSTextAttachment *attachment = [[NSTextAttachment alloc] init];
attachment.image = [UIImage imageNamed:@"imageName"];
attachment.bounds = CGRectMake(0, 0, 35, 35);
NSAttributedString *attachmentString = [NSAttributedString
attributedStringWithAttachment:attachment];
```

Chapter 30: NSTextAttachment

Section 30.1: NSTextAttachment Example

```
NSTextAttachment *attachment = [[NSTextAttachment alloc] init];
attachment.image = [UIImage imageNamed:@"imageName"];
attachment.bounds = CGRectMake(0, 0, 35, 35);
NSAttributedString *attachmentString = [NSAttributedString
attributedStringWithAttachment:attachment];
```

第31章：NSCache

第31.1节：NSCache

你使用它的方式与使用NSMutableDictionary相同。不同之处在于，当NSCache检测到
内存压力过大（即缓存了太多值）时，它会释放其中一些值以腾出空间。

如果你可以在运行时重新创建这些值（通过从互联网下载、进行计算等）
那么NSCache可能适合你的需求。如果数据无法重新创建（例如，它是用户输入的，具有时效性等），那么
你不应该将其存储在NSCache中，因为它会被销毁。

Chapter 31: NSCache

Section 31.1: NSCache

You use it the same way you would use NSMutableDictionary. The difference is that when NSCache detects excessive memory pressure (i.e. it's caching too many values) it will release some of those values to make room.

If you can recreate those values at runtime (by downloading from the Internet, by doing calculations, whatever) then NSCache may suit your needs. If the data cannot be recreated (e.g. it's user input, it is time-sensitive, etc.) then you should not store it in an NSCache because it will be destroyed there.

第32章：NSUserDefaults

第32.1节：简单示例

例如：

用于保存：

```
NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];

// 保存一个 NSString
[prefs setObject:txtUsername.text forKey:@"userName"];
[prefs setObject:txtPassword.text forKey:@"password"];

[prefs synchronize];
```

用于获取

```
NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];

// 获取一个 NSString
NSString *savedUsername = [prefs stringForKey:@"userName"];
NSString *savedPassword = [prefs stringForKey:@"password"];
```

第32.2节：清除 NSUserDefaults

```
NSString *appDomain = [[NSBundle mainBundle] bundleIdentifier];
[[NSUserDefaults standardUserDefaults] removePersistentDomainForName:appDomain];
```

Chapter 32: NSUserDefaults

Section 32.1: Simple example

For example:

FOR SAVING:

```
NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];

// saving an NSString
[prefs setObject:txtUsername.text forKey:@"userName"];
[prefs setObject:txtPassword.text forKey:@"password"];

[prefs synchronize];
```

FOR RETRIEVING

```
NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];

// getting an NSString
NSString *savedUsername = [prefs stringForKey:@"userName"];
NSString *savedPassword = [prefs stringForKey:@"password"];
```

Section 32.2: Clear NSUserDefaults

```
NSString *appDomain = [[NSBundle mainBundle] bundleIdentifier];
[[NSUserDefaults standardUserDefaults] removePersistentDomainForName:appDomain];
```

第33章：下标

第33.1节：NSArray的下标

下标可以用来简化数组中元素的获取和设置。给定以下数组

```
NSArray *fruit = @[@"Apples", @"Bananas", @"Cherries"];
```

这行代码

```
[fruit objectAtIndex: 1];
```

可以被替换为

```
fruit[1];
```

下标也可以用来设置可变数组中的元素。

```
NSMutableArray *fruit = [@[@"Apples", @"Bananas", @"Cherries"] mutableCopy];
fruit[1] = @"Blueberries";
NSLog(@"%@", fruit[1]); //Blueberries
```

如果下标的索引等于数组的长度，该元素将被追加到数组中。

可以使用重复的下标来访问嵌套数组的元素。

```
NSArray *fruit = @[@"苹果", @"香蕉", @"樱桃"];
NSArray *vegetables = @[@"鳄梨", @"豆类", @"胡萝卜"];
NSArray *produce = @[fruit, vegetables];

NSLog(@"%@", produce[0][1]); //香蕉
```

第33.2节：自定义下标

你可以通过实现所需的方法为自己的类添加下标功能。

对于索引下标（类似数组）：

```
- (id)objectAtIndexedSubscript:(NSUInteger)idx
- (void)setObject:(id)obj atIndexedSubscript:(NSUInteger)idx
```

对于键控下标（类似字典）：

```
- (id)objectForKeyedSubscript:(id)key
- (void)setObject:(id)obj forKeyedSubscript:(id <NSCopying>)key
```

第33.3节：使用NSDictionary的下标

下标也可以用于NSDictionary和NSMutableDictionary。以下代码：

```
NSMutableDictionary *myDictionary = [@{@"Foo": @"Bar"} mutableCopy];
[myDictionary setObject:@"Baz" forKey:@"Foo"];
NSLog(@"%@", [myDictionary objectForKey:@"Foo"]); // Baz
```

Chapter 33: Subscripting

Section 33.1: Subscripts with NSArray

Subscripts can be used to simplify retrieving and setting elements in an array. Given the following array

```
NSArray *fruit = @[@"Apples", @"Bananas", @"Cherries"];
```

This line

```
[fruit objectAtIndex: 1];
```

Can be replaced by

```
fruit[1];
```

They can also be used to set an element in a mutable array.

```
NSMutableArray *fruit = [@[@"Apples", @"Bananas", @"Cherries"] mutableCopy];
fruit[1] = @"Blueberries";
NSLog(@"%@", fruit[1]); //Blueberries
```

If the index of the subscript equals the count of the array, the element will be appended to the array.

Repeated subscripts may be used to access elements of nested arrays.

```
NSArray *fruit = @[@"Apples", @"Bananas", @"Cherries"];
NSArray *vegetables = @[@"Avocado", @"Beans", @"Carrots"];
NSArray *produce = @[fruit, vegetables];

NSLog(@"%@", produce[0][1]); //Bananas
```

Section 33.2: Custom Subscripting

You can add subscripting to your own classes by implementing the required methods.

For indexed subscripting (like arrays):

```
- (id)objectAtIndexedSubscript:(NSUInteger)idx
- (void)setObject:(id)obj atIndexedSubscript:(NSUInteger)idx
```

For keyed subscripting (like dictionaries):

```
- (id)objectForKeyedSubscript:(id)key
- (void)setObject:(id)obj forKeyedSubscript:(id <NSCopying>)key
```

Section 33.3: Subscripts with NSDictionary

Subscripts can also be used with NSDictionary and NSMutableDictionary. The following code:

```
NSMutableDictionary *myDictionary = [@{@"Foo": @"Bar"} mutableCopy];
[myDictionary setObject:@"Baz" forKey:@"Foo"];
NSLog(@"%@", [myDictionary objectForKey:@"Foo"]); // Baz
```

可以简化为：

```
NSMutableDictionary *myDictionary = [@{@"Foo": @"Bar"} mutableCopy];
myDictionary[@"Foo"] = @"Baz";
NSLog(@"%@", myDictionary[@"Foo"]); // Baz
```

Can be shortened to:

```
NSMutableDictionary *myDictionary = [@{@"Foo": @"Bar"} mutableCopy];
myDictionary[@"Foo"] = @"Baz";
NSLog(@"%@", myDictionary[@"Foo"]); // Baz
```

第34章：低级运行时环境

第34.1节：使用方法交换增强方法

Objective-C 运行时允许你在运行时更改方法的实现。这被称为方法*交换*，通常用于交换两个方法的实现。例如，如果方法foo和bar被交换，发送消息foo将执行bar的实现，反之亦然。

该技术可用于增强或"修补"你无法直接编辑的现有方法，例如系统提供类的方法。

在以下示例中，-[NSUserDefaults synchronize]方法被增强，以打印原始实现的执行时间。

重要提示：许多人尝试使用method_exchangeImplementations进行交换。然而，如果你需要调用被替换的方法，这种方法是危险的，因为你将使用与其预期接收不同的选择器调用它。结果，你的代码可能会以奇怪且意想不到的方式崩溃——特别是当多个方以这种方式交换同一个对象时。相反，你应始终结合使用setImplementation和 C 函数进行交换，这样可以让你使用原始选择器调用该方法。

```
#import "NSUserDefaults+Timing.h"
#import <objc/runtime.h> // 需要用于方法交换

static IMP old_synchronize = NULL;

static void new_synchronize(id self, SEL _cmd);

@implementation NSUserDefaults(Timing)

+ (void)load
{
Method originalMethod = class_getInstanceMethod([self class], @selector(synchronize:));
    IMP swizzleImp = (IMP)new_synchronize;
old_synchronize = method_setImplementation(originalMethod, swizzleImp);
}
@end

static void new_synchronize(id self, SEL _cmd);
{
    NSDate *started;
    BOOL returnValue;

started = [NSDate date];

    // 调用原始实现，传递与此函数调用时相同的参数
    // 包括选择器。
returnValue = old_synchronize(self, _cmd);


NSLog(@"写入用户默认设置耗时 %f 秒。", [[NSDate date]
timeIntervalSinceDate:started]);

    return returnValue;
}
```

Chapter 34: Low-level Runtime Environment

Section 34.1: Augmenting methods using Method Swizzling

The Objective-C runtime allows you to change the implementation of a method at runtime. This is called *method swizzling* and is often used to exchange the implementations of two methods. For example, if the methods foo and bar are exchanged, sending the message foo will now execute the implementation of bar and vice versa.

This technique can be used to augment or "patch" existing methods which you cannot edit directly, such as methods of system-provided classes.

In the following example, the -[NSUserDefaults synchronize] method is augmented to print the execution time of the original implementation.

IMPORTANT: Many people try to do swizzling using method_exchangeImplementations. However, this approach is dangerous if you need to call the method you're replacing, because you'll be calling it using a different selector than it is expecting to receive. As a result, your code can break in strange and unexpected ways—particularly if multiple parties swizzle an object in this way. Instead, you should always do swizzling using setImplementation in conjunction with a C function, allowing you to call the method with the original selector.

```
#import "NSUserDefaults+Timing.h"
#import <objc/runtime.h> // Needed for method swizzling

static IMP old_synchronize = NULL;

static void new_synchronize(id self, SEL _cmd);

@implementation NSUserDefaults(Timing)

+ (void)load
{
    Method originalMethod = class_getInstanceMethod([self class], @selector(synchronize:));
    IMP swizzleImp = (IMP)new_synchronize;
    old_synchronize = method_setImplementation(originalMethod, swizzleImp);
}
@end

static void new_synchronize(id self, SEL _cmd);
{
    NSDate *started;
    BOOL returnValue;

    started = [NSDate date];

    // Call the original implementation, passing the same parameters
    // that this function was called with, including the selector.
    returnValue = old_synchronize(self, _cmd);


    NSLog(@"Writing user defaults took %f seconds.", [[NSDate date]
timeIntervalSinceDate:started]);

    return returnValue;
}
```

```
@end
```

如果需要替换一个带参数的方法，只需将它们作为函数的额外参数添加即可。
例如：

```
static IMP old_viewWillAppear_animated = NULL;
static void new_viewWillAppear_animated(id self, SEL _cmd, BOOL animated);

…

Method originalMethod = class_getClassMethod([UIViewController class], @selector(viewWillAppear:));
IMP swizzleImp = (IMP)new_viewWillAppear_animated;
old_viewWillAppear_animated = method_setImplementation(originalMethod, swizzleImp);

…

static void new_viewWillAppear_animated(id self, SEL _cmd, BOOL animated)
{
    ...

old_viewWillAppear_animated(self, _cmd, animated);

   …
}
```

第34.2节：将对象附加到另一个已存在的对象（关联）

可以将一个对象附加到一个已存在的对象上，就好像新增了一个属性一样。这称为关联，允许扩展现有对象。它可以用于在通过类扩展添加属性时提供存储，或以其他方式向现有对象添加额外信息。

一旦目标对象被释放，关联的对象会由运行时自动释放。

```
#import <objc/runtime.h>

// 关联的“键”。其值从未被使用且无关紧要。
// 这个全局静态变量的唯一目的是
// 在运行时提供一个保证唯一的值：没有两个不同的
// 全局变量可以共享相同的地址。
static char key;

id target = …;
id payload = …;
objc_setAssociateObject(目标, &键, 载荷, OBJC_ASSOCIATION_RETAIN);
// 其他有用的值有 OBJC_ASSOCIATION_COPY
// 和 OBJ_ASSOCIATION_ASSIGN

id 查询载荷 = objc_getAssociatedObject(目标, &键);
```

第34.3节：直接调用方法

如果你需要从C代码调用Objective-C方法，有两种方式：使用`objc_msgSend`，或者获取`IMP`（方法实现函数指针）并调用它。

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

@implementation 示例
```

```
@end
```

If you need to swizzle a method that takes parameters, you just add them as additional parameters to the function. For example:

```
static IMP old_viewWillAppear_animated = NULL;
static void new_viewWillAppear_animated(id self, SEL _cmd, BOOL animated);

...

Method originalMethod = class_getClassMethod([UIViewController class], @selector(viewWillAppear:));
IMP swizzleImp = (IMP)new_viewWillAppear_animated;
old_viewWillAppear_animated = method_setImplementation(originalMethod, swizzleImp);

...

static void new_viewWillAppear_animated(id self, SEL _cmd, BOOL animated)
{
    ...

    old_viewWillAppear_animated(self, _cmd, animated);

    ...
}
```

Section 34.2: Attach object to another existing object (association)

It's possible to attach an object to an existing object as if there was a new property. This is called *association* and allows one to extend existing objects. It can be used to provide storage when adding a property via a class extension or otherwise add additional information to an existing object.

The associated object is automatically released by the runtime once the target object is deallocated.

```
#import <objc/runtime.h>

// "Key" for association. Its value is never used and doesn't
// matter. The only purpose of this global static variable is to
// provide a guaranteed unique value at runtime: no two distinct
// global variables can share the same address.
static char key;

id target = ...;
id payload = ...;
objc_setAssociateObject(target, &key, payload, OBJC_ASSOCIATION_RETAIN);
// Other useful values are OBJC_ASSOCIATION_COPY
// and OBJ_ASSOCIATION_ASSIGN

id queryPayload = objc_getAssociatedObject(target, &key);
```

Section 34.3: Calling methods directly

If you need to call an Objective-C method from C code, you have two ways: using `objc_msgSend`, or obtaining the `IMP` (method implementation function pointer) and calling that.

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

@implementation Example
```

```
- (double)取反:(double)值 {
    return -值;
}

- (double)倒数:(double)值 {
    return 1 / 值;
}

@end

// 在对象上调用选择器。期望该方法有一个double参数并返回一个double。

double performSelectorWithMsgSend(id object, SEL selector, double value) {
    // 我们声明指向函数的指针，并将 `objc_msgSend` 强制转换为预期的签名。
    // 警告：这一步非常重要！否则你可能会得到意想不到的结果！
    double (*msgSend)(id, SEL, double) = (typeof(msgSend)) &objc_msgSend;

    // 除了任何显式参数外，还需要传递隐式参数 self 和 _cmd。

    return msgSend(object, selector, value);
}

// 功能与上述函数相同，但通过获取方法的 IMP 实现。
double performSelectorWithIMP(id object, SEL selector, double value) {
    // 获取方法的实现。
    IMP imp = class_getMethodImplementation([self class], selector);

    // 强制转换类型以便类型已知且 ARC 能正确工作。
    double (*callableImp)(id, SEL, double) = (typeof(callableImp)) imp;

    // 再次说明，你需要显式的参数。
    return callableImp(object, selector, value);
}

int main() {
Example *e = [Example new];

    // 调用取反，结果是 -4
    double x = performSelectorWithMsgSend(e, @selector(negate:), 4);

    // 调用取倒数，结果是 0.25
    double y = performSelectorWithIMP(e, @selector(invert:), 4);
}
```

`objc_msgSend` 的工作原理是获取方法的 IMP 并调用它。最近调用的几个方法的 `IMP` 被缓存，所以如果你在一个非常紧密的循环中发送 Objective-C 消息，可以获得可接受的性能
性能。在某些情况下，手动缓存IMP可以带来略微更好的性能，尽管这是一种最后的优化手段。

```
- (double)negate:(double)value {
    return -value;
}

- (double)invert:(double)value {
    return 1 / value;
}

@end

// Calls the selector on the object. Expects the method to have one double argument and return a
double.
double performSelectorWithMsgSend(id object, SEL selector, double value) {
    // We declare pointer to function and cast `objc_msgSend` to expected signature.
    // WARNING: This step is important! Otherwise you may get unexpected results!
    double (*msgSend)(id, SEL, double) = (typeof(msgSend)) &objc_msgSend;

    // The implicit arguments of self and _cmd need to be passed in addition to any explicit
arguments.
    return msgSend(object, selector, value);
}

// Does the same as the above function, but by obtaining the method's IMP.
double performSelectorWithIMP(id object, SEL selector, double value) {
    // Get the method's implementation.
    IMP imp = class_getMethodImplementation([self class], selector);

    // Cast it so the types are known and ARC can work correctly.
    double (*callableImp)(id, SEL, double) = (typeof(callableImp)) imp;

    // Again, you need the explicit arguments.
    return callableImp(object, selector, value);
}

int main() {
    Example *e = [Example new];

    // Invoke negation, result is -4
    double x = performSelectorWithMsgSend(e, @selector(negate:), 4);

    // Invoke inversion, result is 0.25
    double y = performSelectorWithIMP(e, @selector(invert:), 4);
}
```

`objc_msgSend` works by obtaining the IMP for the method and calling that. The `IMP`s for the last several methods called are cached, so if you're sending an Objective-C message in a very tight loop you can get acceptable performance. In some cases, manually caching the IMP can give slightly better performance, although this is a last resort optimization.

第35章：快速枚举

第35.1节：带索引的NSArray快速枚举

本例展示了如何使用快速枚举遍历NSArray。通过这种方式，您还可以在遍历时跟踪当前对象的索引。

假设您有一个数组，

```
NSArray *weekDays = @[@"星期一", @"星期二", @"星期三", @"星期四", @"星期五", @"星期六",
@"星期日"];
```

现在您可以像下面这样遍历数组，

```
[weekDays enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {

    //... 在这里执行您的常规操作

obj  // 这是当前对象
    idx  // 这是当前对象的索引
    stop // 如果想停止遍历，将此设置为true

}];
```

第35.2节：NSArray的快速枚举

本示例展示了如何使用快速枚举遍历 NSArray。

当你有一个数组，例如

```
NSArray *collection = @[@"fast", @"enumeration", @"in objc"];
```

你可以使用 for ... in 语法遍历数组中的每个元素，自动从索引 0 的第一个元素开始，直到最后一个元素结束：

```
for (NSString *item in collection) {
    NSLog(@"item: %@", item);
}
```

在此示例中，生成的输出将如下所示

```
// item: fast
// item: enumeration
// item: in objc
```

Chapter 35: Fast Enumeration

Section 35.1: Fast enumeration of an NSArray with index

This example shows how to use fast enumeration in order to traverse through an NSArray. With this way you can also track current object's index while traversing.

Suppose you have an array,

```
NSArray *weekDays = @[@"Monday", @"Tuesday", @"Wednesday", @"Thursday", @"Friday", @"Saturday",
@"Sunday"];
```

Now you can traverse through the array like below,

```
[weekDays enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {

    //... Do your usual stuff here

    obj  // This is the current object
    idx  // This is the index of the current object
    stop // Set this to true if you want to stop

}];
```

Section 35.2: Fast enumeration of an NSArray

This example shows how to use fast enumeration in order to traverse through an NSArray.

When you have an array, such as

```
NSArray *collection = @[@"fast", @"enumeration", @"in objc"];
```

You can use the `for ... in` syntax to go through each item of the array, automatically starting with the first at index `0` and stopping with the last item:

```
for (NSString *item in collection) {
    NSLog(@"item: %@", item);
}
```

In this example, the output generated would look like

```
// item: fast
// item: enumeration
// item: in objc
```

第36章：类别

第36.1节：遵循协议

你可以向标准类添加协议以扩展它们的功能：

```
@protocol EncodableToString <NSObject>
- (NSString *)toString;
@end

@interface NSDictionary (XYZExtended) <EncodableToString>
@end

@implementation NSDictionary (XYZExtended)
- (NSString *)toString {
    return self.description;
}
@end
```

其中 XYZ 是你项目的前缀

第36.2节：简单类别

NSArray的一个简单类别的接口和实现，名为Filter，包含一个用于过滤数字的方法。

给方法添加前缀（PF）是个好习惯，以确保不会覆盖未来的NSArray方法。

```
@interface NSArray (PFFilter)

- (NSArray *)pf_filterSmaller:(double)number;

@end

@implementation NSArray (PFFilter)

- (NSArray *)pf_filterSmaller:(double)number
{
    NSMutableArray *result = [NSMutableArray array];
    for (id val in self)
    {
        if ([val isKindOfClass:[NSNumber class]] && [val doubleValue] >= number)
        {
            [result addObject:val];
        }
    }
    return [result copy];
}

@end
```

第36.3节：声明类方法

头文件 UIColor+XYZPalette.h:

```
@interface UIColor (XYZPalette)
```

Chapter 36: Categories

Section 36.1: Conforming to protocol

You can add protocols to standard classes to extends their functionality:

```
@protocol EncodableToString <NSObject>
- (NSString *)toString;
@end

@interface NSDictionary (XYZExtended) <EncodableToString>
@end

@implementation NSDictionary (XYZExtended)
- (NSString *)toString {
    return self.description;
}
@end
```

where XYZ your project's prefix

Section 36.2: Simple Category

Interface and implementation of a simple category on NSArray, named Filter, with a single method that filters numbers.

It is good practice to add a prefix (PF) to the method to ensure we don't overwrite any future NSArray methods.

```
@interface NSArray (PFFilter)

- (NSArray *)pf_filterSmaller:(double)number;

@end

@implementation NSArray (PFFilter)

- (NSArray *)pf_filterSmaller:(double)number
{
    NSMutableArray *result = [NSMutableArray array];
    for (id val in self)
    {
        if ([val isKindOfClass:[NSNumber class]] && [val doubleValue] >= number)
        {
            [result addObject:val];
        }
    }
    return [result copy];
}

@end
```

Section 36.3: Declaring a class method

Header file UIColor+XYZPalette.h:

```
@interface UIColor (XYZPalette)
```

```
+(UIColor *)xyz_indigoColor;

@end
```

和实现 UIColor+XYZPalette.m:

```
@implementation UIColor (XYZPalette)

+(UIColor *)xyz_indigoColor
{
    return [UIColor colorWithRed:75/255.0f green:0/255.0f blue:130/255.0f alpha:1.0f];
}

@end
```

第36.4节：使用类别添加属性

可以使用Objective-C运行时的关联对象功能，通过类别添加属性。

注意，属性声明中的retain，nonatomic与objc_setAssociatedObject的最后一个参数相匹配
。详见“将对象附加到另一个现有对象”的说明。

```
#import <objc/runtime.h>

@interface UIViewController (ScreenName)

@property (retain, nonatomic) NSString *screenName;

@end

@implementation UIViewController (ScreenName)

@dynamic screenName;

- (NSString *)screenName {
    return objc_getAssociatedObject(self, @selector(screenName));
}

- (void)setScreenName:(NSString *)screenName {
    objc_setAssociatedObject(self, @selector(screenName), screenName,
OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

@end
```

第36.5节：在XCode中创建分类

分类提供了在不进行子类化或更改实际对象的情况下，为对象添加额外功能的能力。

例如，我们想设置一些自定义字体。让我们创建一个为UIFont类添加功能的分类。打开你的XCode项目，点击File -> New -> File，选择Objective-C文件，点击下一步，输入你的分类名称，比如“CustomFont”，选择文件类型为分类，类为UIFont，然后点击“下一步”，最后点击“创建”。

```
+(UIColor *)xyz_indigoColor;

@end
```

and implementation `UIColor+XYZPalette.m`:

```
@implementation UIColor (XYZPalette)

+(UIColor *)xyz_indigoColor
{
    return [UIColor colorWithRed:75/255.0f green:0/255.0f blue:130/255.0f alpha:1.0f];
}

@end
```

Section 36.4: Adding a property with a category

Properties can be added with categories using associated objects, a feature of the Objective-C runtime.

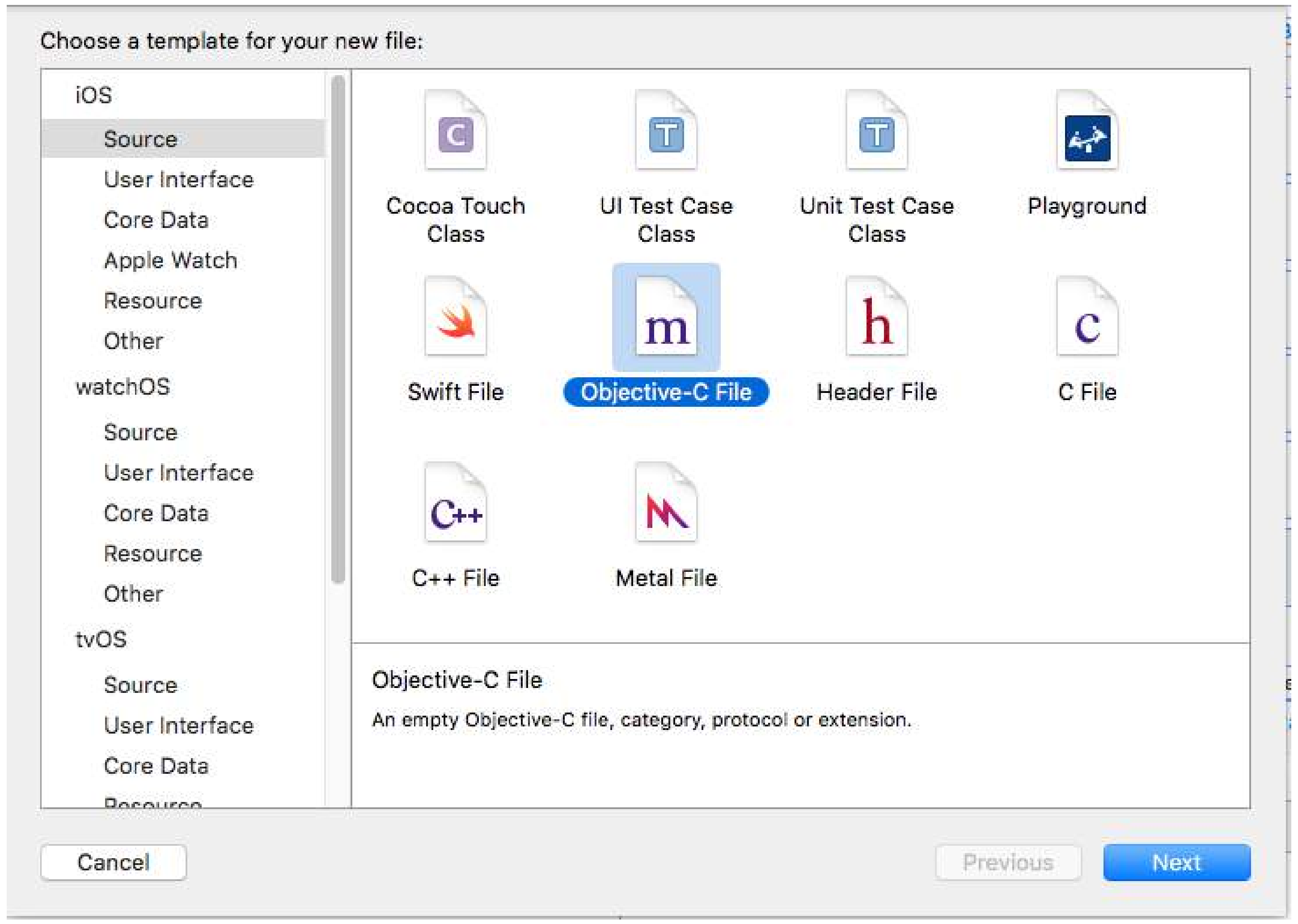
Note that the property declaration of `retain, nonatomic` matches the last argument to `objc_setAssociatedObject`. See Attach object to another existing object for explanations.

```
#import <objc/runtime.h>

@interface UIViewController (ScreenName)

@property (retain, nonatomic) NSString *screenName;

@end

@implementation UIViewController (ScreenName)

@dynamic screenName;

- (NSString *)screenName {
    return objc_getAssociatedObject(self, @selector(screenName));
}

- (void)setScreenName:(NSString *)screenName {
    objc_setAssociatedObject(self, @selector(screenName), screenName,
OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

@end
```

Section 36.5: Create a Category on XCode

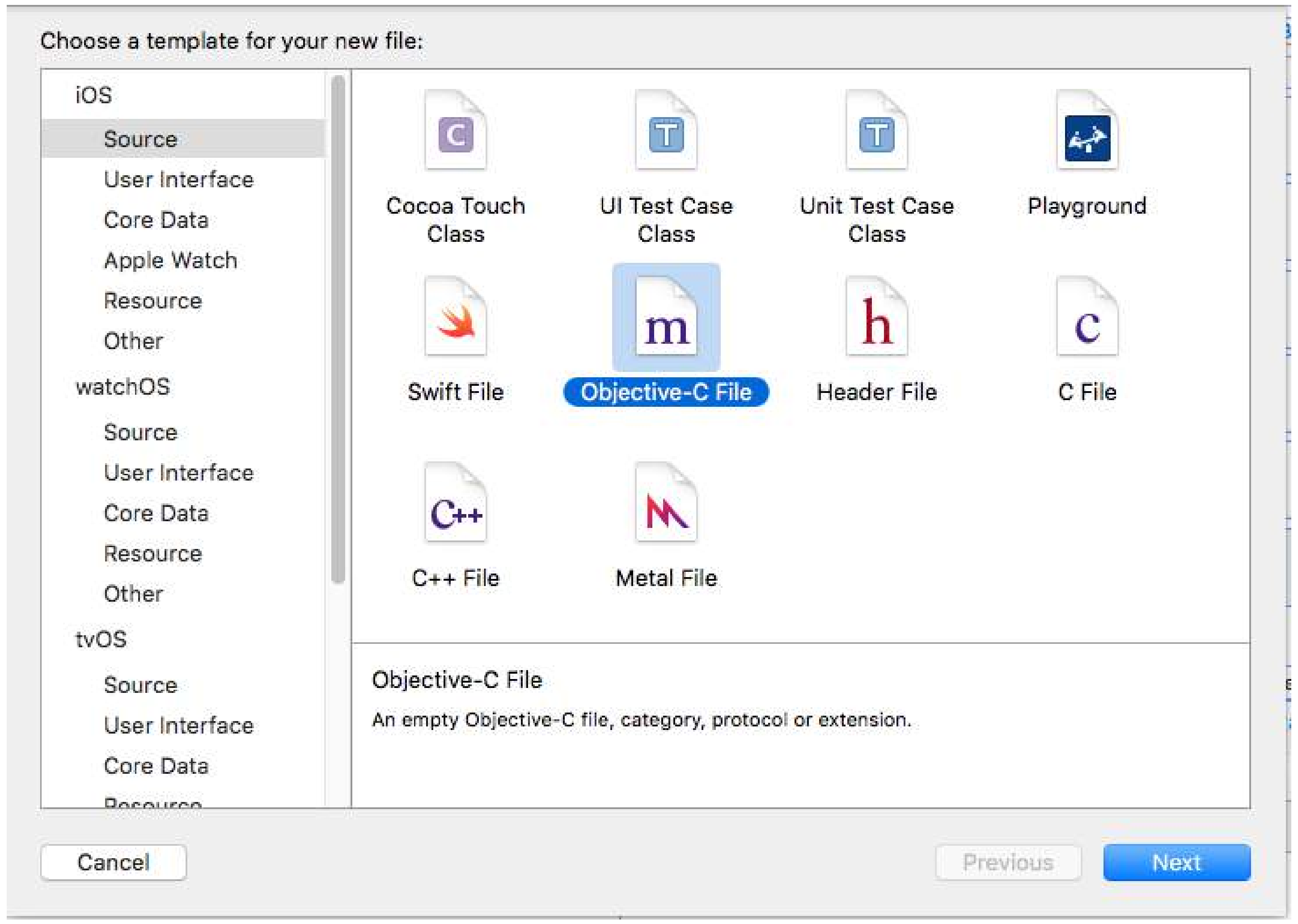
Categories provide the ability to add some extra functionality to an object without subclassing or changing the actual object.

For example we want to set some custom fonts. Let's create a category that add functionality to `UIFont` class. Open your XCode project, click on `File` -> `New` -> `File` and choose `Objective-C file`, click Next enter your category name say "CustomFont" choose file type as Category and Class as UIFont then Click "Next" followed by "Create."

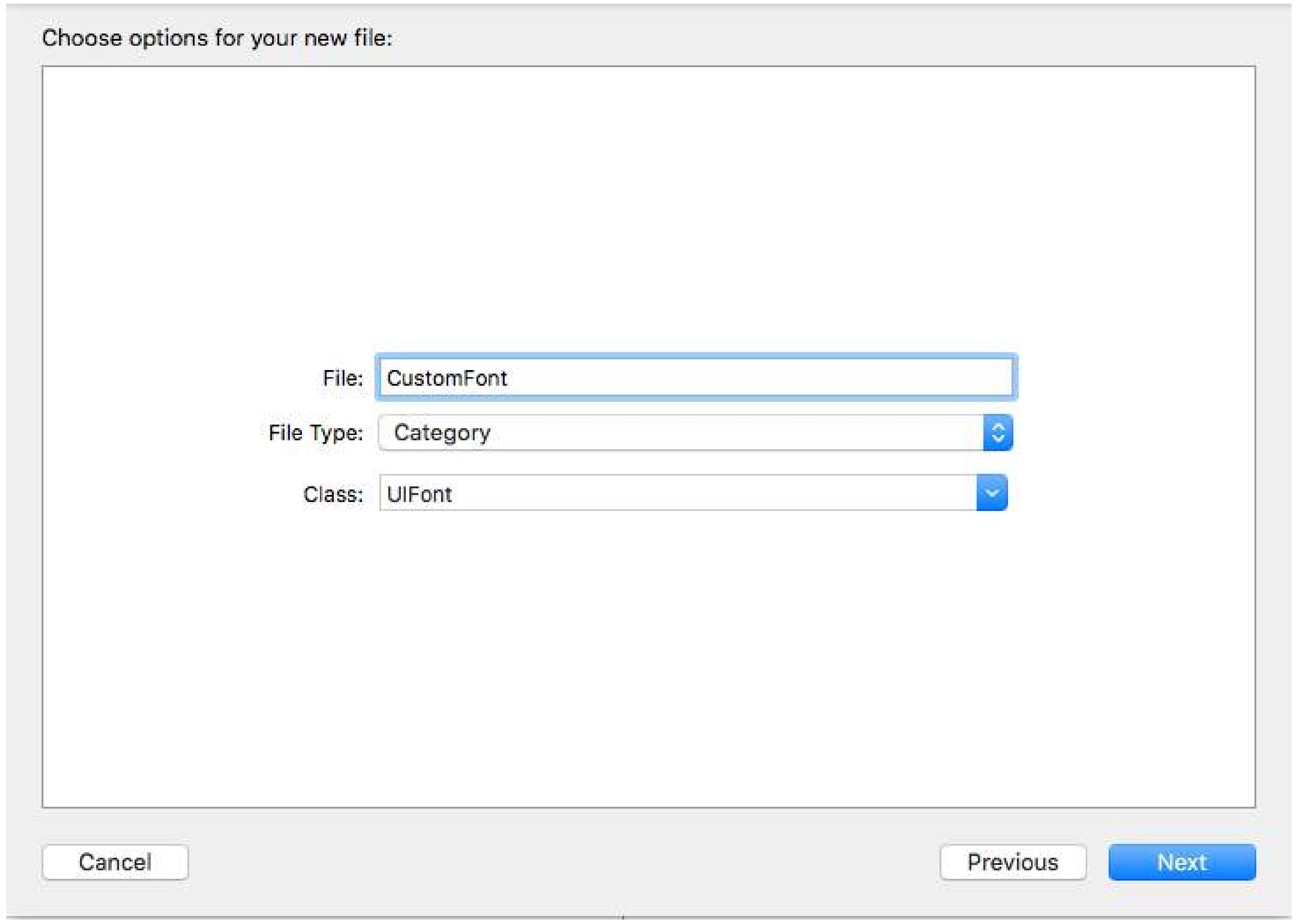


Objective-C File

An empty Objective-C file, category, protocol or extension.


Objective-C File

An empty Objective-C file, category, protocol or extension.



Choose options for your new file:

File: CustomFont

File Type: Category

Class: UIFont

声明分类方法：

点击“UIFont+CustomFonts.h”查看新分类的头文件。向接口中添加以下代码以声明该方法。

```
@interface UIFont (CustomFonts)

+(UIFont *)productSansRegularFontWithSize:(CGFloat)size;

@end
```

现在实施类别法：

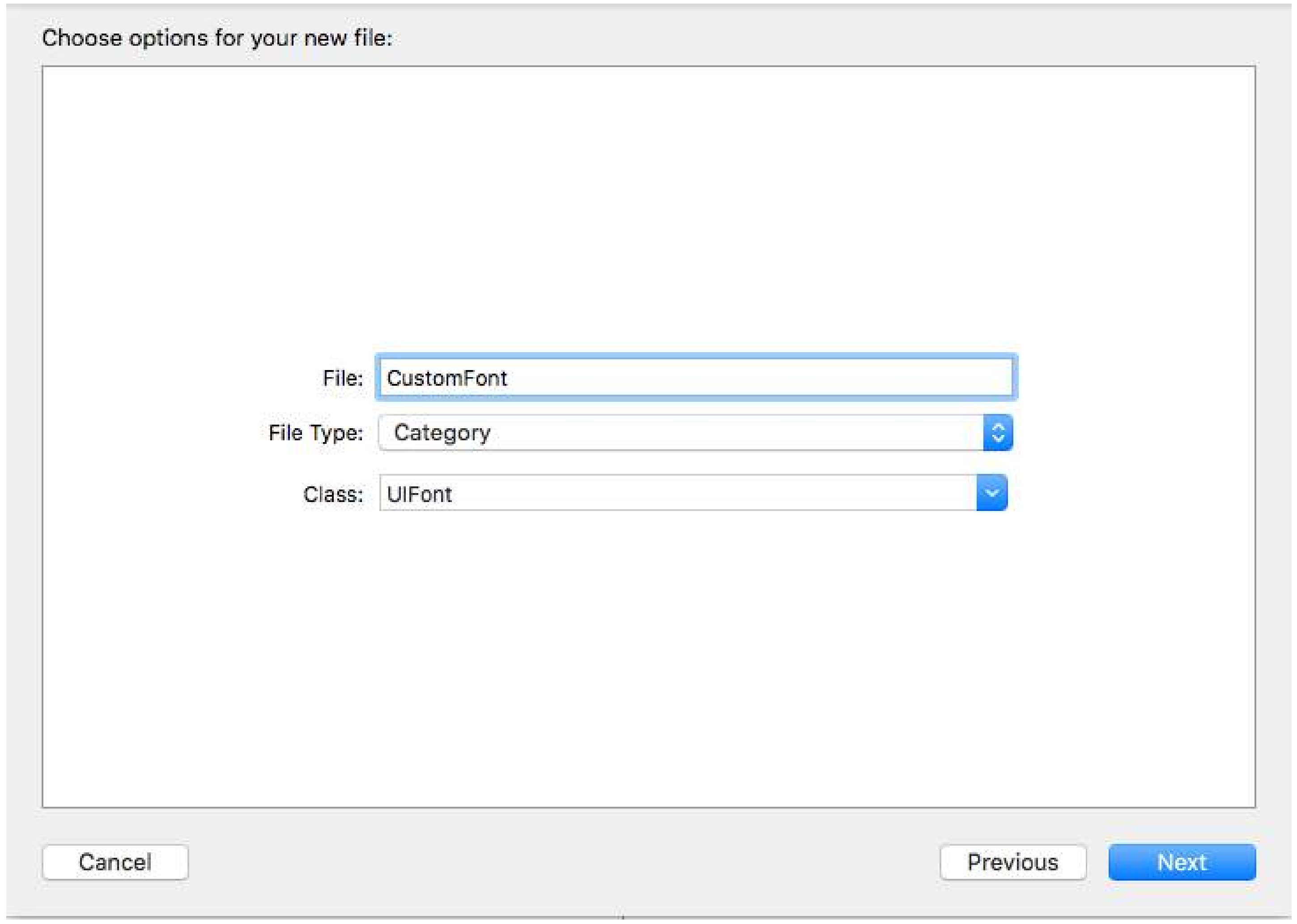
点击 "UIFont+CustomFonts.m" 查看该分类的实现文件。添加以下代码以创建一个设置 ProductSansRegular 字体的方法。

```
+(UIFont *)productSansRegularFontWithSize:(CGFloat)size{

    return [UIFont fontWithName:@"ProductSans-Regular" size:size];

}
```

导入你的分类

```
#import "UIFont+CustomFonts.h"
```

现在设置标签字体



Choose options for your new file:

File: CustomFont

File Type: Category

Class: UIFont

Declare the Category Method:

Click "UIFont+CustomFonts.h" to view the new category's header file. Add the following code to the interface to declare the method.

```
@interface UIFont (CustomFonts)

+(UIFont *)productSansRegularFontWithSize:(CGFloat)size;

@end
```

Now Implement the Category Method:

Click "UIFont+CustomFonts.m" to view the category's implementation file. Add the following code to create a method that will set ProductSansRegular Font.

```
+(UIFont *)productSansRegularFontWithSize:(CGFloat)size{

    return [UIFont fontWithName:@"ProductSans-Regular" size:size];

}
```

Import your category

```
#import "UIFont+CustomFonts.h"
```

Now set the Label font

```
[self.label setFont:[UIFont productSansRegularFontWithSize:16.0]];
```

```
[self.label setFont:[UIFont productSansRegularFontWithSize:16.0]];
```

第37章：协议

第37.1节：可选和必需方法

默认情况下，协议中声明的所有方法都是必需的。这意味着任何遵循该协议的类都必须实现这些方法。

也可以声明可选方法。这些方法只有在需要时才可以实现。

你可以用@optional指令标记可选方法。

```
@protocol NewProtocol
- (void)protocolMethod:(id)argument;
@optional
- (id)anotherMethod;
@end
```

在这种情况下，只有anotherMethod被标记为可选；没有@optional指令的方法被视为必需。

@optional指令适用于其后的方法，直到协议定义结束或遇到另一个指令为止。

```
@protocol NewProtocol
- (void)protocolMethod:(id)argument;
@optional
- (id)anotherMethod;
- (void)andAnotherMethod:(id)argument;
@required
- (void)lastProtocolMethod;
@end
```

最后这个例子定义了一个包含两个可选方法和两个必需方法的协议。

第37.2节：检查可选方法实现的存在性

```
if ([object respondsToSelector:@selector(someOptionalMethodInProtocol:)])
{
    [object someOptionalMethodInProtocol:argument];
}
```

第37.3节：前置声明

可以声明协议名而不指定方法：

```
@protocol Person;
```

在代码中使用它（类定义等）：

```
@interface World : NSObject
@property (strong, nonatomic) NSArray<id<some>> *employees;
@end
```

Chapter 37: Protocols

Section 37.1: Optional and required methods

By default, all the methods declared in a protocol are required. This means that any class that conforms to this protocol must implement those methods.

It is also possible to declare *optional* methods. These method can be implemented only if needed.

You mark optional methods with the `@optional` directive.

```
@protocol NewProtocol
- (void)protocolMethod:(id)argument;
@optional
- (id)anotherMethod;
@end
```

In this case, only `anotherMethod` is marked as optional; the methods without the `@optional` directive are assumed to be required.

The `@optional` directive applies to methods that follow, until the end of the protocol definition or, until another directive is found.

```
@protocol NewProtocol
- (void)protocolMethod:(id)argument;
@optional
- (id)anotherMethod;
- (void)andAnotherMethod:(id)argument;
@required
- (void)lastProtocolMethod;
@end
```

This last example defines a protocol with two optional methods and two required methods.

Section 37.2: Checking existence of optional method implementations

```
if ([object respondsToSelector:@selector(someOptionalMethodInProtocol:)])
{
    [object someOptionalMethodInProtocol:argument];
}
```

Section 37.3: Forward Declarations

It's possible to declare protocol name without methods:

```
@protocol Person;
```

use it your code (class definition, etc):

```
@interface World : NSObject
@property (strong, nonatomic) NSArray<id<some>> *employees;
@end
```

然后在代码的某处定义协议的方法：

```
@protocol Person
- (NSString *)gender;
- (NSString *)name;
@end
```

当你不需要知道协议的细节，直到导入包含协议定义的文件时，这非常有用。因此，你的类头文件保持简洁，只包含类的细节。

第37.4节：遵循协议

以下语法表示一个类采用协议，使用尖括号。

```
@interface NewClass : NSObject <NewProtocol>
...
@end
```

这意味着NewClass的任何实例不仅会响应其接口中声明的方法，还会为NewProtocol的所有必需方法提供实现。

一个类也可以遵循多个协议，协议之间用逗号分隔。

```
@interface NewClass : NSObject <NewProtocol, AnotherProtocol, MyProtocol>
...
@end
```

就像遵循单一协议时，类必须实现每个协议的所有必需方法，以及你选择实现的每个可选方法。

第37.5节：基本协议定义

定义一个新协议：

```
@protocol NewProtocol

- (void)protocolMethod:(id)argument;

- (id)anotherMethod;

@end
```

第37.6节：检查是否遵循协议

返回一个布尔值，指示类是否遵循该协议：

```
[MyClass conformsToProtocol:@protocol(MyProtocol)];
```

and later define protocol's method somewhere in your code:

```
@protocol Person
- (NSString *)gender;
- (NSString *)name;
@end
```

It's useful when you don't need to know protocols details until you import that file with protocol definition. So, your class header file stays clear and contains details of the class only.

Section 37.4: Conforming to Protocols

The following syntax indicate that a class adopts a protocol, using angle brackets.

```
@interface NewClass : NSObject <NewProtocol>
...
@end
```

This means that any instance of NewClass will respond to methods declared in its interface but also it will provide an implementation for all the required methods of `NewProtocol`.

It is also possible for a class to conform to multiple protocols, by separating them with comma.

```
@interface NewClass : NSObject <NewProtocol, AnotherProtocol, MyProtocol>
...
@end
```

Like when conforming to a single protocol, the class must implement each required method of each protocols, and each optional method you choose to implement.

Section 37.5: Basic Protocol Definition

Defining a new protocol:

```
@protocol NewProtocol

- (void)protocolMethod:(id)argument;

- (id)anotherMethod;

@end
```

Section 37.6: Check conforms Protocol

Returns a Boolean indicating if the class conform the protocol:

```
[MyClass conformsToProtocol:@protocol(MyProtocol)];
```

第38章：协议与代理

第38.1节：协议和代理机制的实现

假设你有两个视图ViewA和ViewB

在ViewA内部创建了ViewB的实例，因此ViewA可以向ViewB的实例发送消息，但要实现反向通信，我们需要实现委托（这样通过代理，ViewB的实例可以向ViewA发送消息）按照以下步骤实现委托

1. 在ViewB中创建协议，如下

```
@protocol ViewBDelegate

-(void) exampleDelegateMethod;

@end
```

2. 在发送者类中声明代理

```
@interface ViewB : UIView
@property (nonatomic, weak) id< ViewBDelegate > delegate;
@end
```

3. 在类 ViewA 中采用该协议

```
@interface ViewA: UIView < ViewBDelegate >
```

4. 设置代理

```
-(void) anyFunction
{
    // 创建类 ViewB 的实例并设置代理
    [viewB setDelegate:self];
}
```

5. 在类 ViewA 中实现代理方法

```
-(void) exampleDelegateMethod
{
    // 将由类 ViewB 的实例调用
}
```

6. 在类 ViewB 中使用该方法调用代理方法，如下

```
-(void) callDelegateMethod
{
    [delegate exampleDelegateMethod];
    //假设代理已分配，否则会出错
}
```

Chapter 38: Protocols and Delegates

Section 38.1: Implementation of Protocols and Delegation mechanism

Suppose you have two views ViewA and ViewB

Instance of ViewB is created inside ViewA, so ViewA can send message to ViewB's instance, but for the reverse to happen we need to implement delegation (so that using delegate ViewB's instance could send message to ViewA)

Follow these steps to implement the delegation

1. In ViewB create protocol as

```
@protocol ViewBDelegate

-(void) exampleDelegateMethod;

@end
```

2. Declare the delegate in the sender class

```
@interface ViewB : UIView
@property (nonatomic, weak) id< ViewBDelegate > delegate;
@end
```

3. Adopt the protocol in Class ViewA

```
@interfac ViewA: UIView < ViewBDelegate >
```

4. Set the delegate

```
-(void) anyFunction
{
    // create Class ViewB's instance and set the delegate
    [viewB setDelegate:self];
}
```

5. Implement the delegate method in class ViewA

```
-(void) exampleDelegateMethod
{
    // will be called by Class ViewB's instance
}
```

6. Use the method in class ViewB to call the delegate method as

```
-(void) callDelegateMethod
{
    [delegate exampleDelegateMethod];
    //assuming the delegate is assigned otherwise error
}
```

第39章：块（Blocks）

第39.1节：块类型定义（Block Typedefs）

```
typedef double (^Operation)(double first, double second);
```

如果你将块类型声明为typedef，那么你可以使用新的类型名来代替完整的参数和返回值描述。这定义了Operation为一个接受两个double并返回double的块。

该类型可以用作方法的参数：

```
- (double)doWithOperation:(Operation)operation
                    first:(double)first
second:(double)second;
```

或者作为变量类型：

```
Operation addition = ^double(double first, double second){
    return first + second;
};

// 返回 3.0
[self doWithOperation:addition
                first:1.0
               second:2.0];
```

没有 typedef 的话，这会更乱：

```
- (double)doWithOperation:(double (^)(double, double))operation
                    first:(double)first
second:(double)second;

double (^addition)(double, double) = // ...
```

第 39.2 节：作为属性的 Block

```
@interface MyObject : MySuperclass

@property (copy) void (^blockProperty)(NSString *string);

@end
```

赋值时，由于 self 会持有 blockProperty，block 不应包含对 self 的强引用。这些相互的强引用称为“保留循环”，会阻止任一对象的释放。

```
__weak __typeof(self) weakSelf = self;
self.blockProperty = ^(NSString *string) {
    // 这里只引用 weakSelf。self 会导致保留循环
};
```

虽然极不可能，但self可能会在block内部的某处执行过程中被释放。在这种情况下weakSelf会变成nil，所有对它的消息调用都不会产生预期效果。这可能会导致应用处于未知状态。可以通过在block执行期间用__strong实例变量保留weakSelf，并在之后进行清理来避免这种情况。

```
__weak __typeof(self) weakSelf = self;
```

Chapter 39: Blocks

Section 39.1: Block Typedefs

```
typedef double (^Operation)(double first, double second);
```

If you declare a block type as a typedef, you can then use the new type name instead of the full description of the arguments and return values. This defines `Operation` as a block that takes two doubles and returns a double.

The type can be used for the parameter of a method:

```
- (double)doWithOperation:(Operation)operation
                    first:(double)first
                   second:(double)second;
```

or as a variable type:

```
Operation addition = ^double(double first, double second){
    return first + second;
};

// Returns 3.0
[self doWithOperation:addition
                first:1.0
               second:2.0];
```

Without the typedef, this is much messier:

```
- (double)doWithOperation:(double (^)(double, double))operation
                    first:(double)first
                   second:(double)second;

double (^addition)(double, double) = // ...
```

Section 39.2: Blocks as Properties

```
@interface MyObject : MySuperclass

@property (copy) void (^blockProperty)(NSString *string);

@end
```

When assigning, since `self` retains `blockProperty`, block should not contain a strong reference to self. Those mutual strong references are called a "retain cycle" and will prevent the release of either object.

```
__weak __typeof(self) weakSelf = self;
self.blockProperty = ^(NSString *string) {
    // refer only to weakSelf here.  self will cause a retain cycle
};
```

It is highly unlikely, but `self` might be deallocated inside the block, somewhere during the execution. In this case `weakSelf` becomes `nil` and all messages to it have no desired effect. This might leave the app in an unknown state. This can be avoided by retaining `weakSelf` with a `__strong` ivar during block execution and clean up afterward.

```
__weak __typeof(self) weakSelf = self;
```

```
self.blockProperty = ^(NSString *string) {
    __strong __typeof(weakSelf) strongSelf = weakSelf;
    // 这里仅引用strongSelf。
    // ...
    // 执行结束时，清理引用
strongSelf = nil;
};
```

第39.3节：作为局部变量的Block

```
returnType (^blockName)(parameterType1, parameterType2, ...) = ^returnType(argument1, argument2,
...) {...};

float (^square)(float) = ^(float x) {return x*x;};

square(5); // 结果为 25
square(-7); // 结果为49
```

这是一个没有返回值且无参数的示例：

```
NSMutableDictionary *localStatus;
void (^logStatus)() = ^(void){ [MYUniversalLogger logCurrentStatus:localStatus]};

// 插入一些代码以向localStatus字典添加有用的状态信息


logStatus(); // 这将调用带有当前localStatus的block
```

第39.4节：作为方法参数的Block

```
- (void)methodWithBlock:(returnType (^)(paramType1, paramType2, ...))name;
```

第39.5节：定义与赋值

一个执行两个双精度数相加的block，赋值给变量addition：

```
double (^addition)(double, double) = ^double(double first, double second){
    return first + second;
};
```

该代码块随后可以这样调用：

```
double result = addition(1.0, 2.0); // result == 3.0
```

```
self.blockProperty = ^(NSString *string) {
    __strong __typeof(weakSelf) strongSelf = weakSelf;
    // refer only to strongSelf here.
    // ...
    // At the end of execution, clean up the reference
    strongSelf = nil;
};
```

Section 39.3: Blocks as local variables

```
returnType (^blockName)(parameterType1, parameterType2, ...) = ^returnType(argument1, argument2,
...) {...};

float (^square)(float) = ^(float x) {return x*x;};

square(5); // resolves to 25
square(-7); // resolves to 49
```

Here's an example with no return and no parameters:

```
NSMutableDictionary *localStatus;
void (^logStatus)() = ^(void){ [MYUniversalLogger logCurrentStatus:localStatus]};

// Insert some code to add useful status information
// to localStatus dictionary

logStatus(); // this will call the block with the current localStatus
```

Section 39.4: Blocks as Method Parameters

```
- (void)methodWithBlock:(returnType (^)(paramType1, paramType2, ...))name;
```

Section 39.5: Defining and Assigning

A block that performs addition of two double precision numbers, assigned to variable `addition`:

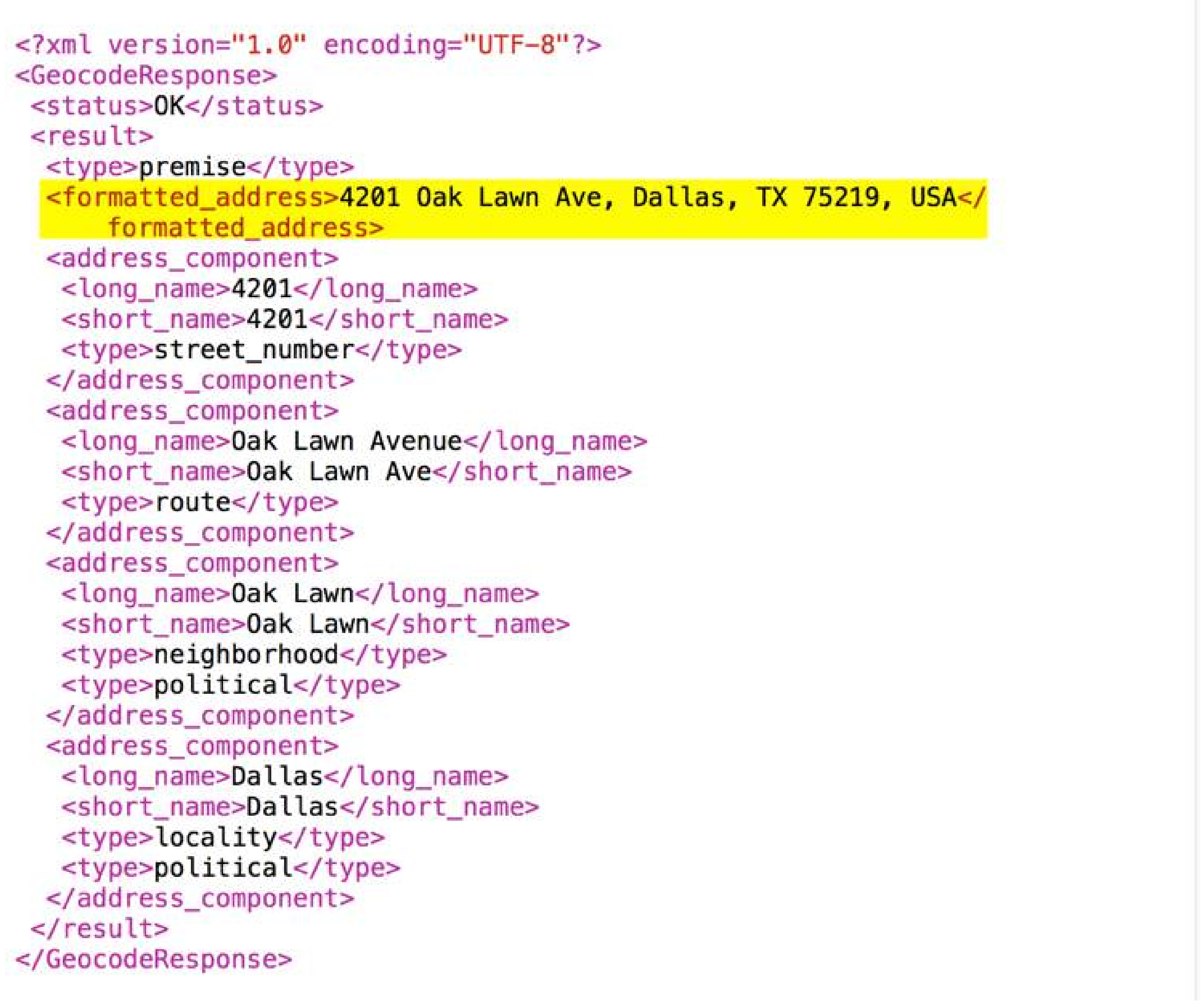
```
double (^addition)(double, double) = ^double(double first, double second){
    return first + second;
};
```

The block can be subsequently called like so:

```
double result = addition(1.0, 2.0); // result == 3.0
```

第40章：XML解析

第40.1节：XML解析



```
<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
 <status>OK</status>
 <result>
  <type>premise</type>
  <formatted_address>4201 Oak Lawn Ave, Dallas, TX 75219, USA</
      formatted_address>
  <address_component>
   <long_name>4201</long_name>
   <short_name>4201</short_name>
   <type>street_number</type>
  </address_component>
  <address_component>
   <long_name>Oak Lawn Avenue</long_name>
   <short_name>Oak Lawn Ave</short_name>
   <type>route</type>
  </address_component>
  <address_component>
   <long_name>Oak Lawn</long_name>
   <short_name>Oak Lawn</short_name>
   <type>neighborhood</type>
   <type>political</type>
  </address_component>
  <address_component>
   <long_name>Dallas</long_name>
   <short_name>Dallas</short_name>
   <type>locality</type>
   <type>political</type>
  </address_component>
 </result>
</GeocodeResponse>
```

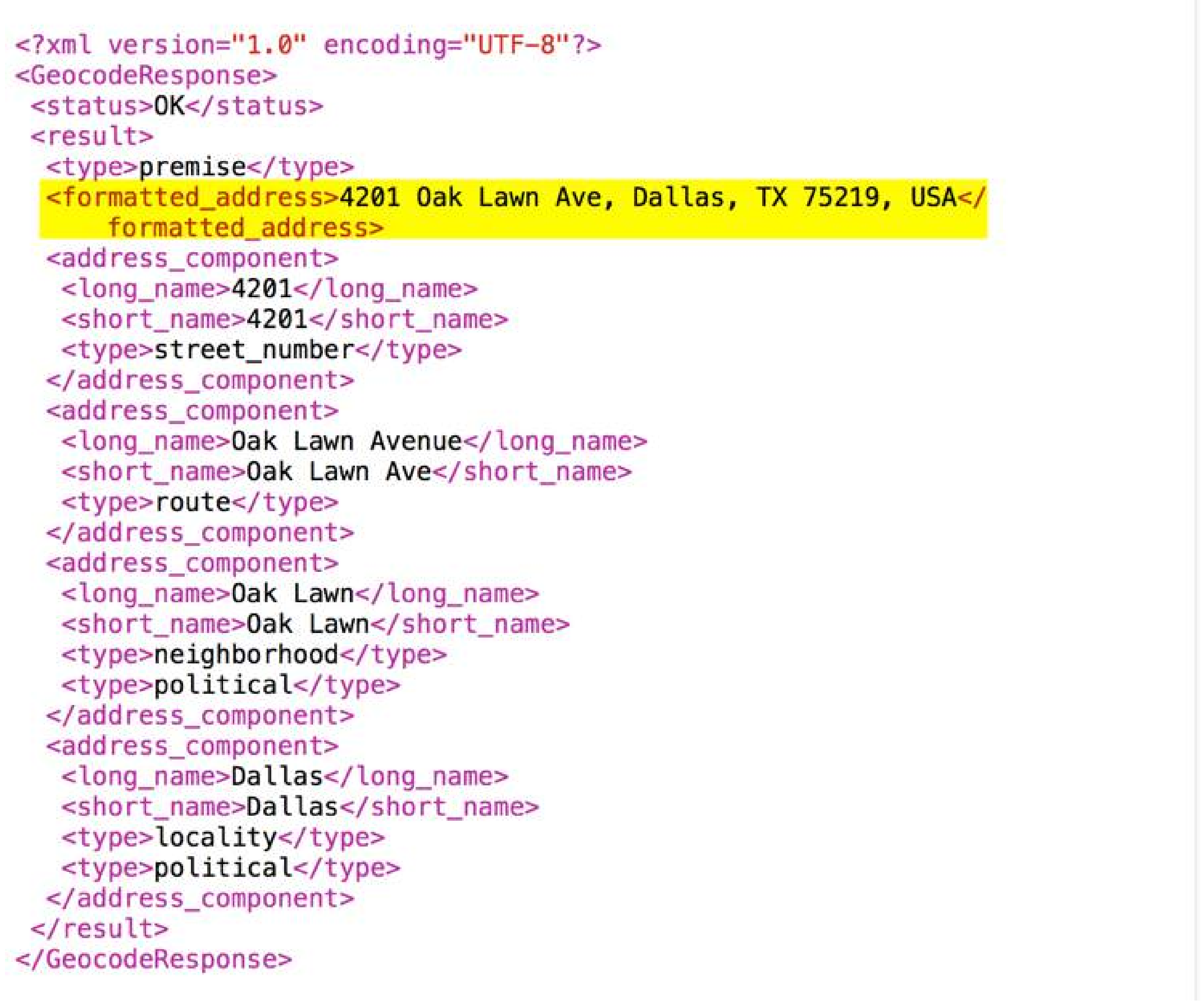
我们将通过NSXMLParser解析高亮标签数据我们声明了以下几

个属性

```
@property(nonatomic, strong)NSMutableArray *results;
@property(nonatomic, strong)NSMutableString *parsedString;
@property(nonatomic, strong)NSXMLParser *xmlParser;

//获取xml数据
NSURLSession *session=[NSURLSession sessionWithConfiguration:[NSURLSessionConfiguration
defaultSessionConfiguration]];

NSURLSessionDataTask *task=[session dataTaskWithRequest:[NSURLRequest requestWithURL:[NSURL
URLWithString:YOUR_XMLURL]] completionHandler:^(NSData * _Nullable data, NSURLResponse * _Nullable
response, NSError * _Nullable error) {

self.xmlParser=[[NSXMLParser 分配] 使用数据:data]初始化；
self.xmlParser.delegate=self;
```

Chapter 40: XML parsing

Section 40.1: XML Parsing



```
<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
 <status>OK</status>
 <result>
  <type>premise</type>
  <formatted_address>4201 Oak Lawn Ave, Dallas, TX 75219, USA</
      formatted_address>
  <address_component>
   <long_name>4201</long_name>
   <short_name>4201</short_name>
   <type>street_number</type>
  </address_component>
  <address_component>
   <long_name>Oak Lawn Avenue</long_name>
   <short_name>Oak Lawn Ave</short_name>
   <type>route</type>
  </address_component>
  <address_component>
   <long_name>Oak Lawn</long_name>
   <short_name>Oak Lawn</short_name>
   <type>neighborhood</type>
   <type>political</type>
  </address_component>
  <address_component>
   <long_name>Dallas</long_name>
   <short_name>Dallas</short_name>
   <type>locality</type>
   <type>political</type>
  </address_component>
 </result>
</GeocodeResponse>
```

We will parse the highlighted tag data through `NSXMLParser`

We have declared few properties as follows

```
@property(nonatomic, strong)NSMutableArray *results;
@property(nonatomic, strong)NSMutableString *parsedString;
@property(nonatomic, strong)NSXMLParser *xmlParser;

//Fetch xml data
NSURLSession *session=[NSURLSession sessionWithConfiguration:[NSURLSessionConfiguration
defaultSessionConfiguration]];

NSURLSessionDataTask *task=[session dataTaskWithRequest:[NSURLRequest requestWithURL:[NSURL
URLWithString:YOUR_XMLURL]] completionHandler:^(NSData * _Nullable data, NSURLResponse * _Nullable
response, NSError * _Nullable error) {

self.xmlParser=[[NSXMLParser alloc] initWithData:data];
self.xmlParser.delegate=self;
if([self.xmlParser parse]){
   //如果解析成功完成

NSLog(@"%@",self.results);

}

}];

[task resume];
```

然后我们定义NSXMLParserDelegate

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName namespaceURI:(nullable
NSString *)namespaceURI qualifiedName:(nullable NSString *)qName attributes:(NSDictionary<NSString
*, NSString *> *)attributeDict{

    if([elementName isEqualToString:@"GeocodeResponse"]){
        self.results=[[NSMutableArray alloc] init];
    }

    if([elementName isEqualToString:@"formatted_address"]){
        self.parsedString=[[NSMutableString alloc] init];
    }

}


- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string{

    if(self.parsedString){
        [self.parsedString appendString:[string stringByTrimmingCharactersInSet:[NSCharacterSet
whitespaceAndNewlineCharacterSet]]];
    }


}

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName namespaceURI:(nullable
NSString *)namespaceURI qualifiedName:(nullable NSString *)qName{

    if([elementName isEqualToString:@"formatted_address"]){
        [self.results addObject:self.parsedString];

self.parsedString=nil;
    }

}
```

```
if([self.xmlParser parse]){
   //If parsing completed successfully

        NSLog(@"%@",self.results);

}

}];

[task resume];
```

Then we define the `NSXMLParserDelegate`

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName namespaceURI:(nullable
NSString *)namespaceURI qualifiedName:(nullable NSString *)qName attributes:(NSDictionary<NSString
*, NSString *> *)attributeDict{

    if([elementName isEqualToString:@"GeocodeResponse"]){
        self.results=[[NSMutableArray alloc] init];
    }

    if([elementName isEqualToString:@"formatted_address"]){
        self.parsedString=[[NSMutableString alloc] init];
    }

}


- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string{

    if(self.parsedString){
        [self.parsedString appendString:[string stringByTrimmingCharactersInSet:[NSCharacterSet
whitespaceAndNewlineCharacterSet]]];
    }


}

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName namespaceURI:(nullable
NSString *)namespaceURI qualifiedName:(nullable NSString *)qName{

    if([elementName isEqualToString:@"formatted_address"]){
        [self.results addObject:self.parsedString];

        self.parsedString=nil;
    }

}
```

第41章：声明类方法和实例方法

实例方法是特定于某个类的方法。实例方法的声明和定义以 -（减号）符号开头。

类方法可以通过类名本身调用。类方法的声明和定义使用 +（加号）符号。

第41.1节：如何声明类方法和实例方法

实例方法使用类的实例。

```
@interface MyTestClass : NSObject

- (void)测试实例方法;

@end
```

它们可以这样使用：

```
MyTestClass *对象 = [[MyTestClass 分配] 初始化];
[对象 测试实例方法];
```

类方法可以仅用类名调用。

```
@interface MyClass : NSObject

+ (void)类方法;

@end
```

它们可以这样使用：

```
[MyClass 类方法];
```

类方法是许多Foundation类的便捷方法，比如[NSString的 +stringWithFormat:]或NSArray的+arrayWithArray

Chapter 41: Declare class method and instance method

Instance method are methods that are specific to particular classes. Instance methods are declared and defined followed by - (minus) symbol.

Class methods can be called by class name itself .Class methods are declared and defined by using + (plus)sign .

Section 41.1: How to declare class method and instance method

instance methods use an instance of a class.

```
@interface MyTestClass : NSObject

- (void)testInstanceMethod;

@end
```

They could then be used like so:

```
MyTestClass *object = [[MyTestClass alloc] init];
[object testInstanceMethod];
```

Class method can be used with just the class name.

```
@interface MyClass : NSObject

+ (void)aClassMethod;

@end
```

They could then be used like so:

```
[MyClass aClassMethod];
```

class methods are the convenience methods on many Foundation classes like [NSString's +stringWithFormat:] or NSArray's +arrayWithArray

第42章：预定义宏

ANSI C定义了许多宏。虽然每个宏都可以在编程中使用，但预定义宏不应被直接修改。

第42.1节：预定义宏

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

int main()
{
NSLog(@"文件 :%s", __FILE__ );    NSLog(@"
日期 :%s", __DATE__ );    NSLog(@"时间 :%s
", __TIME__ );    NSLog(@"行号 :%d", __LIN
E__ );    NSLog(@"ANSI :%d", __STDC__ );


   return 0;
}
```

当上述代码保存在 main.m 文件中并编译执行时，会产生以下结果：

```
2013-09-14 04:46:14.859 demo[20683] 文件 :main.m
2013-09-14 04:46:14.859 demo[20683] 日期 :Sep 14 2013
2013-09-14 04:46:14.859 demo[20683] 时间 :04:46:14
2013-09-14 04:46:14.859 demo[20683] 行号 :8
2013-09-14 04:46:14.859 demo[20683] ANSI :1
```

Chapter 42: Predefined Macros

ANSI C defines a number of macros. Although each one is available for your use in programming, the predefined macros should not be directly modified.

Section 42.1: Predefined Macros

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

int main()
{
   NSLog(@"File :%s\n", __FILE__ );
   NSLog(@"Date :%s\n", __DATE__ );
   NSLog(@"Time :%s\n", __TIME__ );
   NSLog(@"Line :%d\n", __LINE__ );
   NSLog(@"ANSI :%d\n", __STDC__ );

   return 0;
}
```

When the above code in a file main.m is compiled and executed, it produces the following result:

```
2013-09-14 04:46:14.859 demo[20683] File :main.m
2013-09-14 04:46:14.859 demo[20683] Date :Sep 14 2013
2013-09-14 04:46:14.859 demo[20683] Time :04:46:14
2013-09-14 04:46:14.859 demo[20683] Line :8
2013-09-14 04:46:14.859 demo[20683] ANSI :1
```

第43章：大中央调度

大中央调度（Grand Central Dispatch，GCD） 在 iOS 中，苹果提供了两种多任务处理方式：大中央调度（GCD）和 NSOperationQueue 框架。这里我们将讨论 GCD。GCD 是一种轻量级方式，用于表示将要并发执行的工作单元。你不需要为这些工作单元安排调度；系统会为你处理调度。为代码块添加依赖关系可能会很麻烦。取消或挂起代码块会给开发者带来额外的工作！

第43.1节：什么是Grand Central Dispatch

什么是并发？

- 同时做多件事情。
- 利用多核CPU中可用的核心数量。
- 并行运行多个程序。

并发的目标

- 在后台运行程序而不占用过多CPU资源。
- 定义任务，定义规则，让系统负责执行它们。
- 通过确保主线程可以自由响应用户事件来提高响应能力。

调度队列

Grand Central Dispatch–调度队列允许我们异步或同步执行任意代码块。所有调度队列均为先进先出。所有添加到调度队列的任务都按添加到调度队列的顺序开始执行。

Chapter 43: Grand Central Dispatch

Grand Central Dispatch (GCD) In iOS, Apple provides two ways to do multitasking: The Grand Central Dispatch (GCD) and NSOperationQueue frameworks. We will discuss here about GCD. GCD is a lightweight way to represent units of work that are going to be executed concurrently You don't schedule these units of work; the system takes care of scheduling for you. Adding dependency among blocks can be a headache. Canceling or suspending a block creates extra work for you as a developer!

Section 43.1: What is Grand central dispatch

What is Concurrency?

- Doing multiple things at the same time.
- Taking advantage of number of cores available in multicore CPUs.
- Running multiple programs in parallel.

Objectives of Concurrency

- Running program in background without hogging CPU.
- Define Tasks, Define Rules and let the system take the responsibility of performing them.
- Improve responsiveness by ensuring that the main thread is free to respond to user events.

DISPATCH QUEUES

Grand central dispatch – dispatch queues allows us to execute arbitrary blocks of code either asynchronously or synchronously All Dispatch Queues are first in – first out All the tasks added to dispatch queue are started in the order they were added to the dispatch queue.

第44章：格式说明符

格式说明符用于在Objective-C中将对象值嵌入字符串。

第44.1节：整数示例 - %i

```
int highScore = 57;
NSString *scoreBoard = [NSString stringWithFormat:@"HighScore: %i", (int)highScore];

NSLog(scoreBoard);//输出 "HighScore: 57"
```

Chapter 44: Format-Specifiers

Format-Specifiers are used in Objective-C to implant object-values into a string.

Section 44.1: Integer Example - %i

```
int highScore = 57;
NSString *scoreBoard = [NSString stringWithFormat:@"HighScore: %i", (int)highScore];

NSLog(scoreBoard);//logs "HighScore: 57"
```

第45章：日志记录

第45.1节：日志记录

```
NSLog(@"日志信息！");
NSLog(@"NSString 值: %@", stringValue);
NSLog(@"整数值: %d", intValue);
```

NSLog的第一个参数是包含日志消息格式的NSString。其余参数用作替换格式说明符的值。

格式化的工作方式与printf完全相同，除了额外的格式说明符%@用于任意的Objective-C对象。如下：

```
NSLog(@"%@", object);
```

等同于：

```
NSLog(@"%s", [object description].UTF8String);
```

第45.2节：NSLog输出格式

```
NSLog(@"NSLog消息");
```

调用NSLog打印的消息在Console.app中显示时格式如下：

日期	时间	程序名	进程ID	线程ID	消息
2016-07-16	08:58:04.681	测试	[46259	: 1244773]	NSLog 消息

第45.3节：从发布版本中移除日志语句

通过NSLog打印的消息即使在应用的发布版本中也会显示在 Console.app 上，这对于仅用于调试的打印信息来说没有意义。为了解决这个问题，你可以使用这个宏来进行调试日志记录，替代NSLog。

```
#ifdef DEBUG
#define DLog(...) NSLog(__VA_ARGS__)
#else
#define DLog(...)
#endif
```

使用方法：

```
NSString *value = @"value 1";
DLog(@"value = %@", value);
// 鲜为人知的事实：程序员会在 Console.app 中寻找招聘信息
NSLog(@"我们正在招聘！");
```

在调试版本中，DLog 会调用 NSLog。在发布版本中，DLog 不会执行任何操作。

第45.4节：记录变量值

你不应该像这样调用 NSLog 而不带字面格式字符串：

Chapter 45: Logging

Section 45.1: Logging

```
NSLog(@"Log Message!");
NSLog(@"NSString value: %@", stringValue);
NSLog(@"Integer value: %d", intValue);
```

The first argument of `NSLog` is an `NSString` containing the log message format. The rest of the parameters are used as values to substitute in place of the format specifiers.

The formatting works exactly the same as `printf`, except for the additional format specifier `%@` for an arbitrary Objective-C object. This:

```
NSLog(@"%@", object);
```

is equivalent to:

```
NSLog(@"%s", [object description].UTF8String);
```

Section 45.2: NSLog Output Format

```
NSLog(@"NSLog message");
```

The message that gets printed by calling `NSLog` has the following format when viewed in Console.app:

Date	Time	Program name	Process ID	Thread ID	Message
2016-07-16	08:58:04.681	test	[46259	: 1244773]	NSLog message

Section 45.3: Removing Log Statements from Release Builds

Messages printed from `NSLog` are displayed on Console.app even in the release build of your app, which doesn't make sense for printouts that are only useful for debugging. To fix this, you can use this macro for debug logging instead of `NSLog`.

```
#ifdef DEBUG
#define DLog(...) NSLog(__VA_ARGS__)
#else
#define DLog(...)
#endif
```

To use:

```
NSString *value = @"value 1";
DLog(@"value = %@", value);
// little known fact: programmers look for job postings in Console.app
NSLog(@"We're hiring!");
```

In debug builds, `DLog` will call `NSLog`. In release builds, `DLog` will do nothing.

Section 45.4: Logging Variable Values

You shouldn't call `NSLog` without a literal format string like this:

```
NSLog(variable);     // 危险代码！
```

如果变量不是 NSString，程序将崩溃，因为 NSLog 期望的是一个 NSString。

如果变量是 NSString，除非你的字符串包含 %，否则它会正常工作。 NSLog 会将 % 序列解析为格式说明符，然后从栈中读取一个垃圾值，导致崩溃甚至执行任意代码。

相反，始终将第一个参数设为格式说明符，如下所示：

```
NSLog(@"%@", anObjectVariable);
NSLog(@"%d", anIntegerVariable);
```

第45.5节：空消息不打印

当 NSLog 被要求打印空字符串时，它会完全省略该日志。

```
NSString *name = @"";
NSLog(@"%@", name);  // 解析为 @""
```

上述代码将打印空内容。

在日志前加标签是一种良好习惯：

```
NSString *name = @"";
NSLog(@"Name: %@", name);  // 解析为 @"Name: "
```

上述代码将打印：

```
2016-07-21 14:20:28.623 App[87711:6153103] Name:
```

第45.6节：使用 __FUNCTION __

```
NSLog(@"%s %@",__FUNCTION__, @"etc etc");
```

将类名和方法名插入输出：

```
2016-07-22 12:51:30.099 loggingExample[18132:2971471] -[ViewController viewDidLoad] 等等等等
```

第45.7节：NSLog与printf

```
NSLog(@"NSLog消息");printf("
printf消息");
```

输出：

```
2016-07-16 08:58:04.681 test[46259:1244773] NSLog消息
printf消息
```

NSLog除了日志消息外，还会输出日期、时间、进程名、进程ID和线程ID。printf仅输出消息。

```
NSLog(variable);    // Dangerous code!
```

If the variable is not an NSString, the program will crash, because NSLog expects an NSString.

If the variable is an NSString, it will work unless your string contains a %. NSLog will parse the % sequence as a format specifier and then read a garbage value off the stack, causing a crash or even executing arbitrary code.

Instead, always make the first argument a format specifier, like this:

```
NSLog(@"%@", anObjectVariable);
NSLog(@"%d", anIntegerVariable);
```

Section 45.5: Empty message is not printed

When NSLog is asked to print empty string, it omits the log completely.

```
NSString *name = @"";
NSLog(@"%@", name);  // Resolves to @""
```

The above code will print **nothing**.

It is a good practice to prefix logs with labels:

```
NSString *name = @"";
NSLog(@"Name: %@", name);  // Resolves to @"Name: "
```

The above code will print:

```
2016-07-21 14:20:28.623 App[87711:6153103] Name:
```

Section 45.6: Using __FUNCTION __

```
NSLog(@"%s %@",__FUNCTION__, @"etc etc");
```

Inserts the class and method name into the output:

```
2016-07-22 12:51:30.099 loggingExample[18132:2971471] -[ViewController viewDidLoad] etc etc
```

Section 45.7: NSLog vs printf

```
NSLog(@"NSLog message");
printf("printf message\n");
```

Output:

```
2016-07-16 08:58:04.681 test[46259:1244773] NSLog message
printf message
```

NSLog outputs the date, time, process name, process ID, and thread ID in addition to the log message. printf just outputs the message.

NSLog需要一个NSString，并且会自动在末尾添加换行符。printf需要一个C字符串，且不会自动添加换行符。

NSLog将输出发送到stderr，printf将输出发送到stdout。

printf和NSLog中的某些格式说明符不同。例如，当包含嵌套字符串时，会产生以下差异：

```
NSLog(@"我的字符串：%@", (NSString *)myString);
printf("我的字符串：%s", [(NSString *)myString UTF8String]);
```

第45.8节：记录NSLog元数据

```
NSLog(@"%s %d %s, yourVariable: %@", __FILE__, __LINE__, __PRETTY_FUNCTION__, yourVariable);
```

将记录文件、行号和函数数据，以及您想要记录的任何变量。这可能会使日志行变得更长，尤其是当文件名和方法名较长时，但它有助于加快错误诊断速度。

您也可以将其封装在宏中（将其存储在单例中或您最需要的地方）；

```
#define ALog(fmt, ...) NSLog((@"%s [Line %d] " fmt), __PRETTY_FUNCTION__, __LINE__, ##__VA_ARGS__);
```

然后当您想要记录日志时，只需调用

```
ALog(@"name: %@", firstName);
```

这将给您类似于以下内容；

```
 -[AppDelegate application:didFinishLaunchingWithOptions:] [Line 27] name: John
```

第45.9节：NSLog和BOOL类型

NSLog没有格式说明符来打印布尔类型。打印布尔值的一种方法是将其转换为字符串。

```
BOOL boolValue = YES;
NSLog(@"布尔值 %@", boolValue ? @"是" : @"否");
```

输出：

```
2016-07-30 22:53:18.269 Test[4445:64129] 布尔值 YES
```

另一种打印布尔值的方法是将其强制转换为整数，从而实现二进制输出（1=是，0=否）。

```
BOOL boolValue = YES;
NSLog(@"布尔值 %i", boolValue);
```

输出：

```
2016-07-30 22:53:18.269 Test[4445:64129] 布尔值 1
```

NSLog requires an NSString and automatically adds a newline at the end. printf requires a C string and does not automatically add a newline.

NSLog sends output to stderr, printf sends output to stdout.

Some format-specifiers in printf vs NSLog are different. For example when including a nested string, the following differences incur:

```
NSLog(@"My string: %@", (NSString *)myString);
printf("My string: %s", [(NSString *)myString UTF8String]);
```

Section 45.8: Logging NSLog meta data

```
NSLog(@"%s %d %s, yourVariable: %@", __FILE__, __LINE__, __PRETTY_FUNCTION__, yourVariable);
```

Will log the file, line number and function data along with any variables you want to log. This can make the log lines much longer, particularly with verbose file and method names, however it can help to speed up error diagnostics.

You can also wrap this in a Macro (store this in a Singleton or where you'll need it most);

```
#define ALog(fmt, ...) NSLog((@"%s [Line %d] " fmt), __PRETTY_FUNCTION__, __LINE__, ##__VA_ARGS__);
```

Then when you want to log, simply call

```
ALog(@"name: %@", firstName);
```

Which will give you something like;

```
 -[AppDelegate application:didFinishLaunchingWithOptions:] [Line 27] name: John
```

Section 45.9: NSLog and BOOL type

There is no format specifier to print boolean type using NSLog. One way to print boolean value is to convert it to a string.

```
BOOL boolValue = YES;
NSLog(@"Bool value %@", boolValue ? @"YES" : @"NO");
```

Output:

```
2016-07-30 22:53:18.269 Test[4445:64129] Bool value YES
```

Another way to print boolean value is to cast it to integer, achieving a binary output (1=yes, 0=no).

```
BOOL boolValue = YES;
NSLog(@"Bool value %i", boolValue);
```

Output:

```
2016-07-30 22:53:18.269 Test[4445:64129] Bool value 1
```

第45.10节：通过追加到文件进行日志记录

NSLog 很好，但你也可以通过追加到文件来记录日志，使用如下代码：

```
NSFileHandle* fh = [NSFileHandle fileHandleForWritingAtPath:path];
if ( !fh ) {
    [[NSFileManager defaultManager] createFileAtPath:path contents:nil attributes:nil];
    fh = [NSFileHandle fileHandleForWritingAtPath:path];
}
if ( fh ) {
    @try {
        [fh seekToEndOfFile];
        [fh writeData:[self dataUsingEncoding:enc]];
    }
    @catch (...) {
    }
    [fh closeFile];
}
```

Section 45.10: Logging by Appending to a File

NSLog is good, but you can also log by appending to a file instead, using code like:

```
NSFileHandle* fh = [NSFileHandle fileHandleForWritingAtPath:path];
if ( !fh ) {
    [[NSFileManager defaultManager] createFileAtPath:path contents:nil attributes:nil];
    fh = [NSFileHandle fileHandleForWritingAtPath:path];
}
if ( fh ) {
    @try {
        [fh seekToEndOfFile];
        [fh writeData:[self dataUsingEncoding:enc]];
    }
    @catch (...) {
    }
    [fh closeFile];
}
```

第46章：错误处理

第46.1节：使用try catch块进行错误和异常处理

异常表示程序员级别的错误，比如尝试访问一个不存在的数组元素。

错误是用户级别的问题，比如尝试加载一个不存在的文件。因为错误是在程序正常执行过程中预期会发生的。

示例：

```
    NSArray *库存 = @[@"Sam",
                              @"John",
                              @"Sanju"];
    int 选中索引 = 3;
    @try {
        NSString * name = inventory[selectedIndex];
        NSLog(@"选中的名称是: %@", name);
    } @catch(NSException *theException) {
NSLog(@"发生异常: %@", theException.name);
        NSLog(@"以下是一些详细信息: %@", theException.reason);
    } @finally {
NSLog(@"执行 finally 块");
    }
```

输出：

> 发生异常：NSRangeException

> 以下是一些详细信息：*** -[__NSArrayI objectAtIndex:]: 索引 3 超出范围 [0 .. 2]

> 执行 finally 块

第46.2节：断言

```
@implementation 三角形

...

-(void)setAngles:(NSArray *)_angles {
    self.angles = _angles;

NSAssert((self.angles.count == 3), @"三角形必须有3个角。数组 '%@' 有 %i 个",
self.angles, (int)self.angles.count);

CGFloat angleA = [self.angles[0] floatValue];
    CGFloat angleB = [self.angles[1] floatValue];
    CGFloat angleC = [self.angles[2] floatValue];
    CGFloat sum = (angleA + angleB + angleC);
NSAssert((sum == M_PI), @"三角形的角度必须加起来等于π弧度（180°）。该三角形的
角度加起来为 %f 弧度（%f°）", (float)sum, (float)(sum * (180.0f / M_PI)));
```

Chapter 46: Error Handling

Section 46.1: Error & Exception handling with try catch block

Exceptions represent programmer-level bugs like trying to access an array element that doesn't exist.

Errors are user-level issues like trying load a file that doesn't exist. Because errors are expected during the normal execution of a program.

Example:

```
    NSArray *inventory = @[@"Sam",
                           @"John",
                           @"Sanju"];
    int selectedIndex = 3;
    @try {
        NSString * name = inventory[selectedIndex];
        NSLog(@"The selected Name is: %@", name);
    } @catch(NSException *theException) {
        NSLog(@"An exception occurred: %@", theException.name);
        NSLog(@"Here are some details: %@", theException.reason);
    } @finally {
        NSLog(@"Executing finally block");
    }
```

Output:

> An exception occurred: NSRangeException

> Here are some details: *** -[__NSArrayI objectAtIndex:]: index 3 beyond bounds [0 .. 2]

> Executing finally block

Section 46.2: Asserting

```
@implementation Triangle

...

-(void)setAngles:(NSArray *)_angles {
    self.angles = _angles;

    NSAssert((self.angles.count == 3), @"Triangles must have 3 angles. Array '%@' has %i",
self.angles, (int)self.angles.count);

    CGFloat angleA = [self.angles[0] floatValue];
    CGFloat angleB = [self.angles[1] floatValue];
    CGFloat angleC = [self.angles[2] floatValue];
    CGFloat sum = (angleA + angleB + angleC);
    NSAssert((sum == M_PI), @"Triangles' angles must add up to pi radians (180°). This triangle's
angles add up to %f radians (%f°)", (float)sum, (float)(sum * (180.0f / M_PI)));
}
```

这些断言确保你不会给三角形错误的角度，如果给出错误角度则会抛出异常。如果不抛出异常，那么这个三角形实际上就不是真正的三角形，可能会导致后续代码中的一些错误。

```
}
```

These assertions make sure that you don't give a triangle incorrect angles, by throwing an exception if you do. If they didn't throw an exception than the triangle, not being a true triangle at all, might cause some bugs in later code.

第47章：现代Objective-C

第47.1节：字面量

现代Objective-C提供了减少初始化某些常见类型所需代码量的方法。这种新方式与使用常量字符串初始化NSString对象非常相似。

NSNumber

旧方法：

```
NSNumber *数字 = [NSNumber numberWithInt:25];
```

现代方法：

```
NSNumber *number = @25;
```

注意：你也可以使用@YES、@NO或@(someBoolValue)将BOOL值存储在NSNumber对象中；

NSArray

旧方法：

```
NSArray *array = [[NSArray alloc] initWithObjects:@"One", @"Two", [NSNumber numberWithInt:3],
@"Four", nil];
```

现代方法：

```
NSArray *array = @[@"One", @"Two", @3, @"Four"];
```

NSDictionary

旧方法：

```
NSDictionary *dictionary = [NSDictionary dictionaryWithObjectsAndKeys: array, @"Object", [NSNumber
numberWithFloat:1.5], @"Value", @"ObjectiveC", @"Language", nil];
```

现代方法：

```
NSDictionary *dictionary = @{@"Object": array, @"Value": @1.5, @"Language": @"ObjectiveC"};
```

第47.2节：容器下标

在现代Objective-C语法中，你可以使用容器下标从NSArray和NSDictionary容器中获取值。

旧方法：

```
NSObject *object1 = [array objectAtIndex:1];
NSObject *object2 = [dictionary objectForKey:@"Value"];
```

现代方法：

Chapter 47: Modern Objective-C

Section 47.1: Literals

Modern Objective-C provides ways to reduce amount of code you need to initialize some common types. This new way is very similar to how NSString objects are initialized with constant strings.

NSNumber

Old way:

```
NSNumber *number = [NSNumber numberWithInt:25];
```

Modern way:

```
NSNumber *number = @25;
```

Note: you can also store `BOOL` values in `NSNumber` objects using `@YES`, `@NO` or `@(someBoolValue)`;

NSArray

Old way:

```
NSArray *array = [[NSArray alloc] initWithObjects:@"One", @"Two", [NSNumber numberWithInt:3],
@"Four", nil];
```

Modern way:

```
NSArray *array = @[@"One", @"Two", @3, @"Four"];
```

NSDictionary

Old way:

```
NSDictionary *dictionary = [NSDictionary dictionaryWithObjectsAndKeys: array, @"Object", [NSNumber
numberWithFloat:1.5], @"Value", @"ObjectiveC", @"Language", nil];
```

Modern way:

```
NSDictionary *dictionary = @{@"Object": array, @"Value": @1.5, @"Language": @"ObjectiveC"};
```

Section 47.2: Container subscripting

In modern Objective-C syntax you can get values from `NSArray` and `NSDictionary` containers using container subscripting.

Old way:

```
NSObject *object1 = [array objectAtIndex:1];
NSObject *object2 = [dictionary objectForKey:@"Value"];
```

Modern way:

```
NSObject *object1 = array[1];
NSObject *object2 = dictionary[@"Value"];
```

你也可以用更简洁的方式将对象插入数组或为字典设置键对应的对象：

旧方法：

```
// 在特定索引替换
[mutableArray replaceObjectAtIndex:1 withObject:@"NewValue"];
// 在末尾添加新值
[mutableArray addObject:@"NewValue"];

[mutableDictionary setObject:@"NewValue" forKey:@"NewKey"];
```

现代方法：

```
mutableArray[1] = @"NewValue";
mutableArray[[mutableArray count]] = @"NewValue";

mutableDictionary[@"NewKey"] = @"NewValue";
```

```
NSObject *object1 = array[1];
NSObject *object2 = dictionary[@"Value"];
```

You can also insert objects into arrays and set objects for keys in dictionaries in a cleaner way:

Old way:

```
// replacing at specific index
[mutableArray replaceObjectAtIndex:1 withObject:@"NewValue"];
// adding a new value to the end
[mutableArray addObject:@"NewValue"];

[mutableDictionary setObject:@"NewValue" forKey:@"NewKey"];
```

Modern way:

```
mutableArray[1] = @"NewValue";
mutableArray[[mutableArray count]] = @"NewValue";

mutableDictionary[@"NewKey"] = @"NewValue";
```

第48章：单例模式

使用之前，请务必阅读这篇帖子（单例模式有什么坏处？）。

第48.1节：使用Grand Central Dispatch（GCD）

GCD 将保证您的单例只被实例化一次，即使从多个线程调用也是如此。将此插入到任何类中，以创建一个名为shared的单例实例。

```
+ (instancetype)shared {

    // 指向单例实例的变量。`static` 修饰符使其表现得像全局变量：赋给它的值会“存活”于
    方法调用之后。

    static id _shared;

    static dispatch_once_t _onceToken;
    dispatch_once(&_onceToken, ^{

        // 该代码块只会执行一次，且线程安全。
        // 创建实例并赋值给静态变量。
_shared = [self new];
    });

    return _shared;
}
```

第48.2节：创建单例并防止通过alloc/init、new创建多个实例

```
//MySingletonClass.h
@interface MySingletonClass : NSObject

+ (instancetype)sharedInstance;

-(instancetype)init NS_UNAVAILABLE;

-(instancetype)new NS_UNAVAILABLE;

@end

//MySingletonClass.m

@implementation MySingletonClass

+ (instancetype)sharedInstance
{
    static MySingletonClass *_sharedInstance = nil;
    static dispatch_once_t oncePredicate;
dispatch_once(&oncePredicate, ^{
        _sharedInstance = [[self alloc]init];
    });

    return _sharedInstance;
}
-(instancetype)init
{
self = [super init];
```

Chapter 48: Singletons

Just make sure you read this thread (What is so bad about singletons?) before using it.

Section 48.1: Using Grand Central Dispatch (GCD)

GCD will guarantee that your singleton only gets instantiated once, even if called from multiple threads. Insert this into any class for a singleton instance called `shared`.

```
+ (instancetype)shared {

    // Variable that will point to the singleton instance. The `static`
    // modifier makes it behave like a global variable: the value assigned
    // to it will "survive" the method call.
    static id _shared;

    static dispatch_once_t _onceToken;
    dispatch_once(&_onceToken, ^{

        // This block is only executed once, in a thread-safe way.
        // Create the instance and assign it to the static variable.
        _shared = [self new];
    });

    return _shared;
}
```

Section 48.2: Creating Singleton and also preventing it from having multiple instance using alloc/init, new

```
//MySingletonClass.h
@interface MySingletonClass : NSObject

+ (instancetype)sharedInstance;

-(instancetype)init NS_UNAVAILABLE;

-(instancetype)new NS_UNAVAILABLE;

@end

//MySingletonClass.m

@implementation MySingletonClass

+ (instancetype)sharedInstance
{
    static MySingletonClass *_sharedInstance = nil;
    static dispatch_once_t oncePredicate;
    dispatch_once(&oncePredicate, ^{
        _sharedInstance = [[self alloc]init];
    });

    return _sharedInstance;
}
-(instancetype)init
{
    self = [super init];
    if(self)
    {
        //如果需要，执行任何额外的初始化操作
    }
    return self;
}
@end
```

第48.3节：创建单例类并防止其通过alloc/init创建多个实例

我们可以以一种方式创建单例类，强制开发者使用共享实例（单例对象），而不是创建自己的实例。

```
@implementation MySingletonClass

+ (instancetype)sharedInstance
{
    static MySingletonClass *_sharedInstance = nil;
    static dispatch_once_t oncePredicate;
dispatch_once(&oncePredicate, ^{
        _sharedInstance = [[self alloc] initClass];
    });

    return _sharedInstance;
}

-(instancetype)initClass
{
self = [super init];
    if(self)
    {
        //如果需要，执行任何额外的初始化操作
    }
    return self;
}

- (instancetype)init
{
    @throw [NSException exceptionWithName:@"非指定初始化方法"
                                   reason:@"请使用 [MySingletonClass sharedInstance]"
                            userInfo:nil];
    return nil;
}
@end



/*以下代码行在尝试直接使用 alloc/init 而非使用 s
haredInstance 时  会抛出异常，异常原因是："请使用 [MySingletonClas
s sharedInstance]"*/
MySingletonClass *mySingletonClass = [[MySingletonClass alloc] init];
```

```
    if(self)
    {
        //Do any additional initialization if required
    }
    return self;
}
@end
```

Section 48.3: Creating Singleton class and also preventing it from having multiple instances using alloc/init

We can create Singleton class in such a way that developers are forced to use the shared instance (singleton object) instead of creating their own instances.

```
@implementation MySingletonClass

+ (instancetype)sharedInstance
{
    static MySingletonClass *_sharedInstance = nil;
    static dispatch_once_t oncePredicate;
    dispatch_once(&oncePredicate, ^{
        _sharedInstance = [[self alloc] initClass];
    });

    return _sharedInstance;
}

-(instancetype)initClass
{
    self = [super init];
    if(self)
    {
        //Do any additional initialization if required
    }
    return self;
}

- (instancetype)init
{
    @throw [NSException exceptionWithName:@"Not designated initializer"
                                   reason:@"Use [MySingletonClass sharedInstance]"
                                 userInfo:nil];
    return nil;
}
@end



/*Following line will throw an exception
  with the Reason:"Use [MySingletonClass sharedInstance]"
  when tried to alloc/init directly instead of using sharedInstance */
MySingletonClass *mySingletonClass = [[MySingletonClass alloc] init];
```

第49章：多线程

第49.1节：创建一个简单线程

创建线程最简单的方法是调用一个选择器“在后台”执行。这意味着会创建一个新线程来执行该选择器。接收对象可以是任何对象，不仅仅是self，但它需要响应给定的选择器。

```
- (void)createThread {
    [self performSelectorInBackground:@selector(threadMainWithOptionalArgument:)
                           withObject:someObject];
}

- (void)threadMainWithOptionalArgument:(id)argument {
    // 为避免内存泄漏，线程方法首先需要做的是
    // 创建一个新的自动释放池，可以手动创建或使用 "@autoreleasepool"。
@autoreleasepool {
        // 线程代码应写在这里。
    }
}
```

第49.2节：创建更复杂的线程

使用NSThread的子类可以实现更复杂的线程（例如，允许传递更多
参数或将所有相关的辅助方法封装在一个类中）。此外，NSThread实例可以
保存在属性或变量中，并且可以查询其当前状态（是否仍在运行）。

NSThread类支持一个名为cancel的方法，可以从任何线程调用，该方法以线程安全的方式将
cancelled属性设置为YES。线程实现可以查询（和/或观察）
cancelled属性并退出其main方法。这可以用于优雅地关闭工作线程。

```
// 创建一个新的NSThread子类
@interface MyThread : NSThread

// 为需要从调用者传递到新线程的值添加属性。线程启动后，调用者不得修改这些值，以避免// 线程问
题（或者必须使用锁使属性线程安全）。

@property NSInteger someProperty;

@end

@implementation MyThread

- (void)main
{
@autoreleasepool {
        // 主线程方法写在这里
NSLog(@"新线程。某个属性: %ld", (long)self.someProperty);
    }
}

@end


MyThread *thread = [[MyThread alloc] init];
thread.someProperty = 42;
[thread start];
```

Chapter 49: Multi-Threading

Section 49.1: Creating a simple thread

The most simple way to create a thread is by calling a selector "in the background". This means a new thread is created to execute the selector. The receiving object can be any object, not just `self`, but it needs to respond to the given selector.

```
- (void)createThread {
    [self performSelectorInBackground:@selector(threadMainWithOptionalArgument:)
                           withObject:someObject];
}

- (void)threadMainWithOptionalArgument:(id)argument {
    // To avoid memory leaks, the first thing a thread method needs to do is
    // create a new autorelease pool, either manually or via "@autoreleasepool".
    @autoreleasepool {
        // The thread code should be here.
    }
}
```

Section 49.2: Create more complex thread

Using a subclass of `NSThread` allows implementation of more complex threads (for example, to allow passing more arguments or to encapsulate all related helper methods in one class). Additionally, the `NSThread` instance can be saved in a property or variable and can be queried about its current state (whether it's still running).

The `NSThread` class supports a method called `cancel` that can be called from any thread, which then sets the `cancelled` property to `YES` in a thread-safe way. The thread implementation can query (and/or observe) the `cancelled` property and exit its `main` method. This can be used to gracefully shut down a worker thread.

```
// Create a new NSThread subclass
@interface MyThread : NSThread

// Add properties for values that need to be passed from the caller to the new
// thread. Caller must not modify these once the thread is started to avoid
// threading issues (or the properties must be made thread-safe using locks).
@property NSInteger someProperty;

@end

@implementation MyThread

- (void)main
{
    @autoreleasepool {
        // The main thread method goes here
        NSLog(@"New thread. Some property: %ld", (long)self.someProperty);
    }
}

@end


MyThread *thread = [[MyThread alloc] init];
thread.someProperty = 42;
[thread start];
```

第49.3节：线程局部存储

每个线程都可以访问一个属于当前线程的可变字典。这允许以简单的方式缓存信息而无需加锁，因为每个线程都有自己的专用可变字典：

```
NSMutableDictionary *localStorage = [NSThread currentThread].threadDictionary;
localStorage[someKey] = someValue;
```

线程终止时，该字典会自动释放。

Section 49.3: Thread-local storage

Every thread has access to a mutable dictionary that is local to the current thread. This allows to cache information in an easy way without the need for locking, as each thread has its own dedicated mutable dictionary:

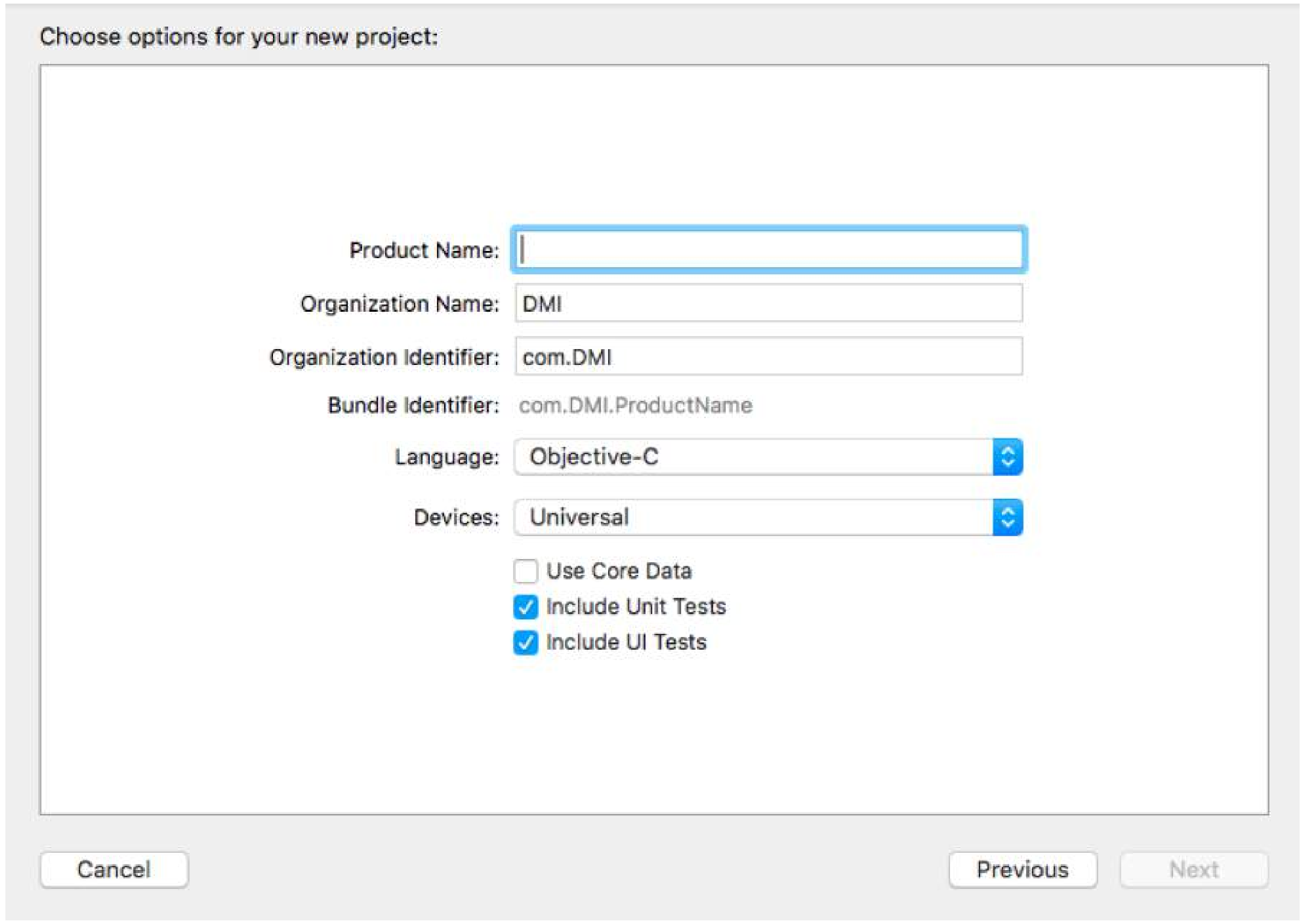
```
NSMutableDictionary *localStorage = [NSThread currentThread].threadDictionary;
localStorage[someKey] = someValue;
```

The dictionary is automatically released when the thread terminates.

第50章：使用Xcode进行单元测试

第50.1节：注：

确保在创建新项目时勾选包含单元测试用例框，如下所示：


Choose options for your new project:

Product Name:
Organization Name: DMI
Organization Identifier: com.DMI
Bundle Identifier: com.DMI.ProductName
Language: Objective-C
Devices: Universal

☐ Use Core Data
☑ Include Unit Tests
☑ Include UI Tests

第50.2节：测试一段代码块或某个方法：

- 导入包含待测试方法的类。
- 使用虚拟数据执行操作。
- 现在将操作结果与预期结果进行比较。

```
- (void)testReverseString{
NSString *originalString = @"hi_my_name_is_siddharth";
NSString *reversedString = [self.someObject reverseString:originalString];
NSString *expectedReversedString = @"htrahddis_si_eman_ym_ih";
XCTAssertEqualObjects(expectedReversedString, reversedString, @"反转字符串与预期反转不匹配");

}
```

> **如有需要，将虚拟数据传入被测试方法，然后比较预期结果与实际结果。**

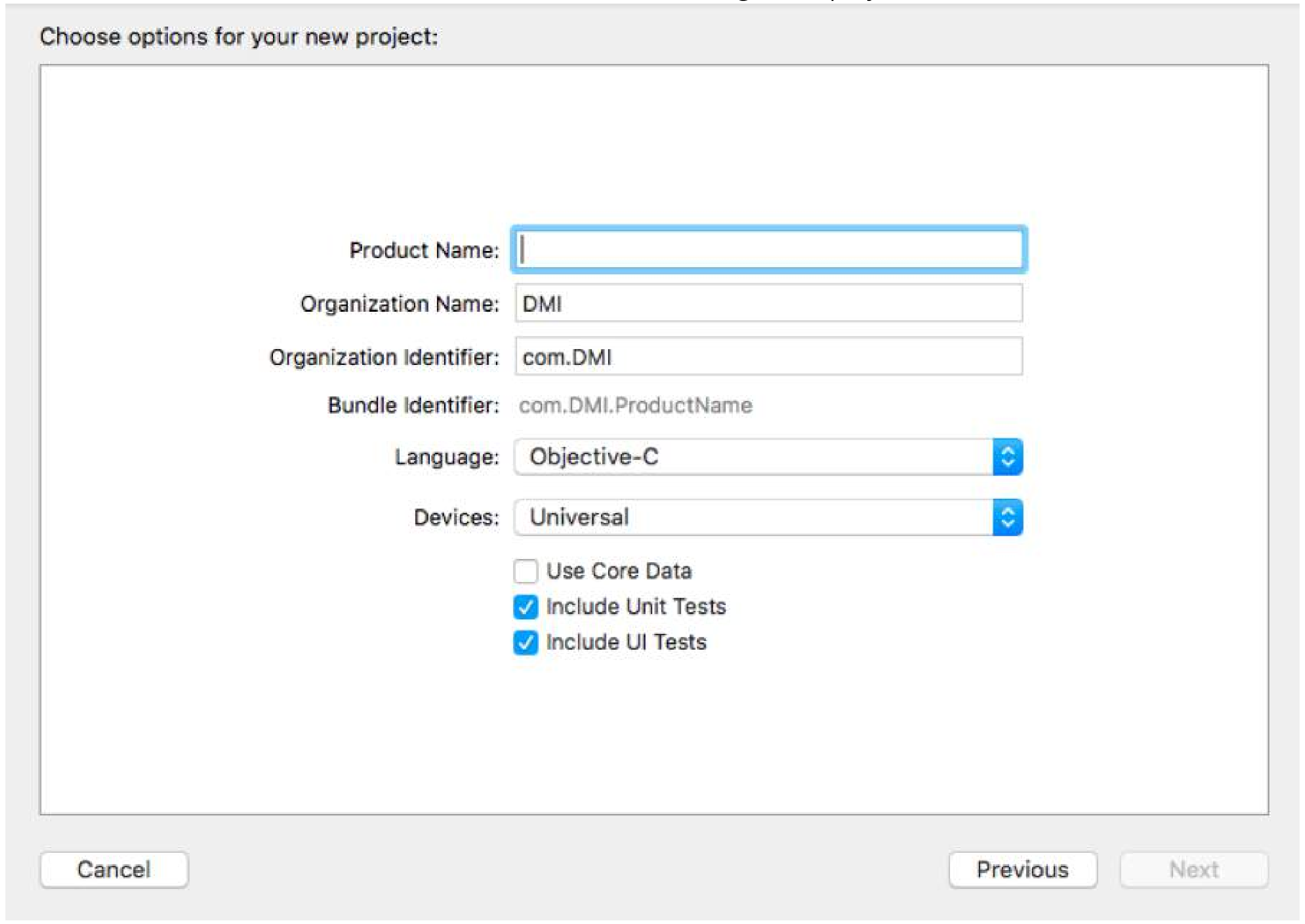
第50.3节：测试异步代码块：

```
- (void)testDoSomethingThatTakesSomeTime{
XCTestExpectation *completionExpectation = [self expectationWithDescription:@"Long method"];
```

Chapter 50: Unit testing using Xcode

Section 50.1: Note:

Make sure that include unit test case box is checked when creating a new project as shown below:


Choose options for your new project:

Product Name:
Organization Name: DMI
Organization Identifier: com.DMI
Bundle Identifier: com.DMI.ProductName
Language: Objective-C
Devices: Universal

☐ Use Core Data
☑ Include Unit Tests
☑ Include UI Tests

Section 50.2: Testing a block of code or some method:

- Import the class, which contains the method to be tested.
- Perform the operation with dummy data.
- Now compare the result of operation with expected result.

```
- (void)testReverseString{
NSString *originalString = @"hi_my_name_is_siddharth";
NSString *reversedString = [self.someObject reverseString:originalString];
NSString *expectedReversedString = @"htrahddis_si_eman_ym_ih";
XCTAssertEqualObjects(expectedReversedString, reversedString, @"The reversed string did not match
the expected reverse");
}
```

> **Feed the dummy data to the method under test if required & then compare the expected & actual results.**

Section 50.3: Testing asynchronous block of code:

```
- (void)testDoSomethingThatTakesSomeTime{
XCTestExpectation *completionExpectation = [self expectationWithDescription:@"Long method"];
[self.someObject doSomethingThatTakesSomeTimesWithCompletionBlock:^(NSString *result) {
    XCTAssertEqualObjects(@"result", result, @"Result was not correct!");
    [completionExpectation fulfill];
}];
[self waitForExpectationsWithTimeout:5.0 handler:nil];
}
```

- 如果需要，向被测试的方法提供虚拟数据。
- 测试将在此处暂停，运行运行循环，直到超时或所有期望都被满足。
- 超时是异步块响应的预期时间。

第50.4节：测量代码块的性能：

1. 对于同步方法：

```
- (void)testPerformanceReverseString {
    NSString *originalString = @"hi_my_name_is_siddharth";
    [self measureBlock:^{
        [self.someObject reverseString:originalString];
    }];
}
```

2. 异步方法：

```
- (void)testPerformanceOfAsynchronousBlock {
   [self measureMetrics:@[XCTPerformanceMetric_WallClockTime] automaticallyStartMeasuring:YES
forBlock:^{

XCTestExpectation *expectation = [self
expectationWithDescription:@"performanceTestWithResponse"];

    [self.someObject doSomethingThatTakesSomeTimesWithCompletionBlock:^(NSString *result) {
        [expectation fulfill];
    }];
    [self waitForExpectationsWithTimeout:5.0 handler:^(NSError *error) {
    }];
}];
}
```

- 这些性能测量代码块连续执行10次，然后计算平均值，
  基于此平均值生成性能结果，并接受该基线用于后续评估。
- 性能结果会与之前的测试结果和基线进行比较，允许自定义最大标准差。

第50.5节：运行测试套件：

通过选择产品 > 测试来运行所有测试。点击测试导航器图标以查看测试的状态和结果。你可以通过点击测试导航器左下角的添加（加号）按钮，将测试目标添加到项目中（或向测试中添加类）。要查看特定测试的源代码，请从测试列表中选择它。文件将在源代码编辑器中打开。

```
[self.someObject doSomethingThatTakesSomeTimesWithCompletionBlock:^(NSString *result) {
    XCTAssertEqualObjects(@"result", result, @"Result was not correct!");
    [completionExpectation fulfill];
}];
[self waitForExpectationsWithTimeout:5.0 handler:nil];
}
```

- Feed the dummy data to the method under test if required.
- The test will pause here, running the run loop, until the timeout is hit or all expectations are fulfilled.
- Timeout is the expected time for the asynchronous block to response.

Section 50.4: Measuring Performance of a block of code:

1. For Synchronous methods :

```
- (void)testPerformanceReverseString {
    NSString *originalString = @"hi_my_name_is_siddharth";
    [self measureBlock:^{
        [self.someObject reverseString:originalString];
    }];
}
```

2. For Asynchronous methods :

```
- (void)testPerformanceOfAsynchronousBlock {
   [self measureMetrics:@[XCTPerformanceMetric_WallClockTime] automaticallyStartMeasuring:YES
forBlock:^{

    XCTestExpectation *expectation = [self
expectationWithDescription:@"performanceTestWithResponse"];

    [self.someObject doSomethingThatTakesSomeTimesWithCompletionBlock:^(NSString *result) {
        [expectation fulfill];
    }];
    [self waitForExpectationsWithTimeout:5.0 handler:^(NSError *error) {
    }];
}];
}
```

- These performance measure block gets executed for 10 times consecutively & then the average is calculated, & on the basis of this average performance result gets created & baseline is accepted for further evaluation.
- The performance result is compared with the previous test results & baseline with a customizable max standard deviation.

Section 50.5: Running Test Suits:

Run all tests by choosing Product > Test. Click the Test Navigator icon to view the status and results of the tests. You can add a test target to a project (or add a class to a test) by clicking the Add (plus) button in the bottom-left corner of the test navigator. To view the source code for a particular test, select it from the test list. The file opens in the source code editor.

第51章：内存管理

第51.1节：使用手动引用计数时的内存管理规则

这些规则仅适用于你使用手动引用计数的情况！

1.你拥有你创建的任何对象

通过调用名称以alloc、new、copy或mutableCopy开头的方法。例如：

```
NSObject *object1 = [[NSObject alloc] init];
NSObject *object2 = [NSObject new];
NSObject *object3 = [object2 copy];
```

这意味着当你使用完这些对象后，你有责任释放它们。

2.你可以使用 retain 来获取对象的所有权

要对一个对象负责，你需要调用 retain 方法。

例如：

```
NSObject *object = [NSObject new]; // 对象的 retain 计数已经是 1
[object retain]; // 现在 retain 计数是 2
```

这只在一些罕见的情况下才有意义。

例如，当你实现一个访问器或初始化方法以取得所有权时：

```
- (void)setStringValue:(NSString *)stringValue {
    [_privateStringValue release]; // 释放旧值，你不再需要它[stringValue retain]; // 确保该对象不会
    在你的作用域之外被释放。

_privateStringValue = stringValue;
}
```

3.当你不再需要它时，必须放弃你拥有的对象的所有权

```
NSObject* object = [NSObject new]; // retain 计数现在是 1
[object performAction1]; // 现在我们已经用完了该对象
[object release]; // 释放该对象
```

4.你不能放弃你不拥有的对象的所有权

这意味着当你没有取得对象的所有权时，你不能释放它。

5.自动释放池

自动释放池是一段代码块，会释放该代码块中所有接收到自动释放消息的对象。

Chapter 51: Memory Management

Section 51.1: Memory management rules when using manual reference counting

These rules apply only if you use manual reference counting!

1. **You own any object you create**

   By calling a method whose name begins with `alloc`, `new`, `copy` or `mutableCopy`. For example:

```
NSObject *object1 = [[NSObject alloc] init];
NSObject *object2 = [NSObject new];
NSObject *object3 = [object2 copy];
```

   That means that you are responsible for releasing these objects when you are done with them.

2. **You can take ownership of an object using retain**

   To take ownership for an object you call the retain method.

   For example:

```
NSObject *object = [NSObject new]; // object already has a retain count of 1
[object retain]; // retain count is now 2
```

   This makes only sense in some rare situations.

   For example when you implement an accessor or an init method to take ownership:

```
- (void)setStringValue:(NSString *)stringValue {
    [_privateStringValue release]; // Release the old value, you no longer need it
    [stringValue retain]; // You make sure that this object does not get deallocated outside
of your scope.
    _privateStringValue = stringValue;
}
```

3. **When you no longer need it, you must relinquish ownership of an object you own**

```
NSObject* object = [NSObject new]; // The retain count is now 1
[object performAction1]; // Now we are done with the object
[object release]; // Release the object
```

4. **You must not relinquish ownership of an object you do not own**

   That means when you didn't take ownership of an object you don't release it.

5. **Autoreleasepool**

   The autoreleasepool is a block of code that releases every object in the block that received an autorelease message.

示例：

```
@autoreleasepool {
    NSString* string = [NSString stringWithString:@"我们不拥有这个对象"];
}
```

我们创建了一个不拥有所有权的字符串。NSString 方法 stringWithString: 必须确保字符串在不再需要时被正确释放。在该方法返回之前，新创建的字符串会调用 autorelease 方法，因此不必拥有该字符串的所有权。

stringWithString: 的实现方式如下：

```
+ (NSString *)stringWithString:(NSString *)string {
    NSString *createdString = [[NSString alloc] initWithString:string];
    [createdString autorelease];
    return createdString;
}
```

使用 autoreleasepool 块是必要的，因为有时你会遇到不拥有所有权的对象（第四条规则并不总是适用）。

自动引用计数会自动处理这些规则，因此你不必手动管理。

第51.2节：自动引用计数

使用自动引用计数（ARC）时，编译器会在需要的地方插入retain、release和autorelease语句，因此你不必自己编写它们。它还会为你编写dealloc方法。

手动内存管理中的示例程序在使用ARC时如下所示：

```
@interface MyObject : NSObject {
    NSString *_property;
}
@end

@implementation MyObject
@synthesize property = _property;

- (id)initWithProperty:(NSString *)property {
    if (self = [super init]) {
_property = property;
    }
    return self;
}

- (NSString *)property {
    return property;
}

- (void)setProperty:(NSString *)property {
    _property = property;
}

@end
```

```
int main() {
MyObject *obj = [[MyObject alloc] init];
```

Example:

```
@autoreleasepool {
    NSString* string = [NSString stringWithString:@"We don't own this object"];
}
```

We have created a string without taking ownership. The `NSString` method `stringWithString:` has to make sure that the string is correctly deallocated after it is no longer needed. Before the method returns the newly created string calls the autorelease method so it does not have to take ownership of the string.

This is how the `stringWithString:` is implemented:

```
+ (NSString *)stringWithString:(NSString *)string {
    NSString *createdString = [[NSString alloc] initWithString:string];
    [createdString autorelease];
    return createdString;
}
```

It is necessary to use autoreleasepool blocks because you sometimes have objects that you don't own (the fourth rules does not always apply).

Automatic reference counting takes automatically care of the rules so you don't have to.

Section 51.2: Automatic Reference Counting

With automatic reference counting (ARC), the compiler inserts `retain`, `release`, and `autorelease` statements where they are needed, so you don't have to write them yourself. It also writes `dealloc` methods for you.

The sample program from Manual Memory Management looks like this with ARC:

```
@interface MyObject : NSObject {
    NSString *_property;
}
@end

@implementation MyObject
@synthesize property = _property;

- (id)initWithProperty:(NSString *)property {
    if (self = [super init]) {
        _property = property;
    }
    return self;
}

- (NSString *)property {
    return property;
}

- (void)setProperty:(NSString *)property {
    _property = property;
}

@end

int main() {
    MyObject *obj = [[MyObject alloc] init];
    NSString *value = [[NSString alloc] initWithString:@"value"];
    [obj setProperty:value];

    [obj setProperty:@"value"];
}
```

你仍然可以重写dealloc方法来清理ARC未处理的资源。与使用手动内存管理时不同，你不需要调用[super dealloc]
。

```
-(void)dealloc {
   //清理
}
```

第51.3节：强引用和弱引用

版本 = 现代

弱引用看起来像下面这样：

```
@property (weak) NSString *property;
NSString *__weak variable;
```

如果你对一个对象有弱引用，那么在底层：

- 你并没有保留它。
- 当它被释放时，所有对它的引用将自动被设置为nil

对象引用默认总是强引用。但你可以显式指定它们是强引用：

```
@property (strong) NSString *property;
NSString *__strong variable;
```

强引用意味着只要该引用存在，你就会保留该对象。

第51.4节：手动内存管理

这是一个使用手动内存管理编写的程序示例。除非因为某些原因不能使用ARC（例如需要支持32位），否则你真的不应该这样写代码。该示例避免了
@property 语法，以说明你过去如何编写getter和setter。

```
@interface MyObject : NSObject {
    NSString *_property;
}
@end

@implementation MyObject
@synthesize property = _property;

- (id)initWithProperty:(NSString *)property {
    if (self = [super init]) {
        // 获取对属性的引用以确保它不会消失。
        // 该引用在dealloc中被释放。
_property = [property retain];
    }
    return self;
```

```
    NSString *value = [[NSString alloc] initWithString:@"value"];
    [obj setProperty:value];

    [obj setProperty:@"value"];
}
```

You are still able to override the dealloc method to clean up resources not handled by ARC. Unlike when using manual memory management you do not call [super dealloc].

```
-(void)dealloc {
   //clean up
}
```

Section 51.3: Strong and weak references

Version = Modern

A weak reference looks like one of these:

```
@property (weak) NSString *property;
NSString *__weak variable;
```

If you have a weak reference to an object, then under the hood:

- You're not retaining it.
- When it gets deallocated, every reference to it will automatically be set to nil

Object references are always strong by default. But you can explicitly specify that they're strong:

```
@property (strong) NSString *property;
NSString *__strong variable;
```

A strong reference means that while that reference exists, you are retaining the object.

Section 51.4: Manual Memory Management

This is an example of a program written with manual memory management. You really shouldn't write your code like this, unless for some reason you can't use ARC (like if you need to support 32-bit). The example avoids @property notation to illustrate how you used to have to write getters and setters.

```
@interface MyObject : NSObject {
    NSString *_property;
}
@end

@implementation MyObject
@synthesize property = _property;

- (id)initWithProperty:(NSString *)property {
    if (self = [super init]) {
        // Grab a reference to property to make sure it doesn't go away.
        // The reference is released in dealloc.
        _property = [property retain];
    }
    return self;
}

- (NSString *)property {
    return [[property retain] autorelease];
}

- (void)setProperty:(NSString *)property {
    // 保留，然后释放。所以设置为相同的值不会丢失引用。
    [property retain];
    [_property release];
    _property = property;
}

- (void)dealloc {
    [_property release];
    [super dealloc]; // 别忘了！
}

@end
```

```
int main() {
    // 创建对象
    // obj 是一个需要释放的引用
MyObject *obj = [[MyObject alloc] init];

    // 我们必须释放值，因为我们创建了它。
    NSString *value = [[NSString alloc] initWithString:@"value"];
    [obj setProperty:value];
    [值释放];

    // 但是，字符串常量永远不需要被释放。
    [obj setProperty:@"value"];
    [obj release];
}
```

```
}

- (NSString *)property {
    return [[property retain] autorelease];
}

- (void)setProperty:(NSString *)property {
    // Retain, then release. So setting it to the same value won't lose the reference.
    [property retain];
    [_property release];
    _property = property;
}

- (void)dealloc {
    [_property release];
    [super dealloc]; // Don't forget!
}

@end
```

```
int main() {
    // create object
    // obj is a reference that we need to release
    MyObject *obj = [[MyObject alloc] init];

    // We have to release value because we created it.
    NSString *value = [[NSString alloc] initWithString:@"value"];
    [obj setProperty:value];
    [value release];

    // However, string constants never need to be released.
    [obj setProperty:@"value"];
    [obj release];
}
```

鸣谢

非常感谢所有来自Stack Overflow Documentation的人员帮助提供这些内容，
更多更改可以发送至web@petercv.com，以发布或更新新内容

阿德里安娜·卡雷利	第32章
阿尔伯特·伦肖	第13、44和45章
阿里·里亚希普尔	第1章
阿米特·卡尔加特吉	第48章
aniket.ghode	第15章
AnthoPak	第14章
Arc676	第27章
atroutt	第36章
BB9z	第39章
巴拉特	第14章
比宾·K·奥南昆朱	第6章和第30章
作者：吉万	第25章
卡勒布·克莱维特	第7章
克里斯·普林斯	第45章
代码更改者	第28章
康纳	第2、33和45章
科里·威利特	第12章
丹	第39章
暗尘	第5、8、13、34、36、45、48和49章
达尔山·昆贾迪亚	第13章
DavidA	第15章和第45章
dgatwood	第34章
迪彭·潘查萨拉	第22章
文档	第3章、第4章、第8章和第46章
多隆·雅科夫列夫	第3章
埃克塔·帕达利亚	第14章
范蒂尼	第16章和第39章
法兰·加尼	第36章
ff10	第35章
霍沃德	第36章和第37章
HCarrasko	第5章、第14章、第15章和第45章
赫曼	第27章
邪恶的火腿博士	第13章
insys	第1、12、16、36、37和39章
iphonic	第40章
J F	第1、8和39章
j.f.	第14章
詹姆斯·P	第13、14和51章
杰森·麦克德莫特	第12章和第45章
杰夫·沃尔斯基	第1章、第13章、第14章和第39章
延斯·梅德尔	第5章
约翰内斯·法伦克鲁格	第14章和第23章
约翰尼·罗克斯	第13章
乔恩·施奈德	第13章
乔斯特	第15章
乔什·布朗	第1章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,
more changes can be sent to web@petercv.com for new content to be published or updated

Adriana Carelli	Chapter 32
Albert Renshaw	Chapters 13, 44 and 45
Ali Riahipour	Chapter 1
Amit Kalghatgi	Chapter 48
aniket.ghode	Chapter 15
AnthoPak	Chapter 14
Arc676	Chapter 27
atroutt	Chapter 36
BB9z	Chapter 39
Bharath	Chapter 14
BIBIN K ONANKUNJU	Chapters 6 and 30
byJeevan	Chapter 25
Caleb Kleveter	Chapter 7
Chris Prince	Chapter 45
CodeChanger	Chapter 28
connor	Chapters 2, 33 and 45
Cory Wilhite	Chapter 12
danh	Chapter 39
DarkDust	Chapters 5, 8, 13, 34, 36, 45, 48 and 49
Darshan Kunjadiya	Chapter 13
DavidA	Chapters 15 and 45
dgatwood	Chapter 34
Dipen Panchasara	Chapter 22
Doc	Chapters 3, 4, 8 and 46
Doron Yakovlev	Chapter 3
Ekta Padaliya	Chapter 14
Fantini	Chapters 16 and 39
Faran Ghani	Chapter 36
ff10	Chapter 35
Håvard	Chapters 36 and 37
HCarrasko	Chapters 5, 14, 15 and 45
Hemang	Chapter 27
il Malvagio Dottor Prosciutto	Chapter 13
insys	Chapters 1, 12, 16, 36, 37 and 39
iphonic	Chapter 40
J F	Chapters 1, 8 and 39
j.f.	Chapter 14
James P	Chapters 13, 14 and 51
Jason McDermott	Chapters 12 and 45
Jeff Wolski	Chapters 1, 13, 14 and 39
Jens Meder	Chapter 5
Johannes Fahrenkrug	Chapters 14 and 23
Johnny Rockex	Chapter 13
Jon Schneider	Chapter 13
Joost	Chapter 15
Josh Brown	Chapter 1

乔什·卡斯韦尔	第2、13、14和39章
约书亚	第7、13和14章
jsondwyer	第7、9、14和18章
科特	第39章
洛西奥瓦蒂	第14章
运输途中丢失	第3章
易卜拉欣·哈桑	第10、11、13和20章
米哈伊尔·拉里奥诺夫	第3章
mrtnf	第7和14章
mszaro	第14章
穆罕默德·佐海布·埃赫桑	第2和14章
米科拉·德尼苏克	第36章和第37章
Nef10	第8章
尼古拉斯·米亚里	第45章
尼古拉·鲁赫	第13章和第18章
尼拉夫·巴特	第12章
恩朱里	第14章
无名氏	第8章和第45章
NS新手	第5章和第13章
ok404	第51章
奥兰多	第13章和第36章
帕特里克	第13、16、18、19、21、26、27和37章
保罗·菲耶罗	第14章和第36章
pckill	第47章
彼得·德维斯	第48章
彼得·N·刘易斯	第45章
phi	第36章
拉胡尔	第29章
拉面厨师	第13章
拉维·多拉吉亚	第17章
regetskcob	第3章
桑杰·莫哈尼	第38章
舒沃	第35章
悉达特·苏尼尔	第50章
西茨	第2章
斯派迪	第15章
斯蒂芬·莱皮克	第40章
斯特阿布兹	第1章和第37章
苏贾尼亚	第2、4、7、15、16、19、21、24、26和46章
苏尼尔·夏尔马	第13章
塔马罗斯	第51章
塔潘·普拉卡什	第7章和第45章
tbodt	第7、13、14、15、34、39、45和51章
ThatsJustCheesy	第12章
托马斯·坦佩尔曼	第45章
Tricertops	第8、13、34和45章
user1374	第3、10、26、31、41、42和43章
william205	第15章
叶夫亨·杜比宁	第39章

Josh Caswell	Chapters 2, 13, 14 and 39
Joshua	Chapters 7, 13 and 14
jsondwyer	Chapters 7, 9, 14 and 18
Kote	Chapter 39
Losiowaty	Chapter 14
lostInTransit	Chapter 3
Md. Ibrahim Hassan	Chapters 10, 11, 13 and 20
Mikhail Larionov	Chapter 3
mrtnf	Chapters 7 and 14
mszaro	Chapter 14
Muhammad Zohaib Ehsan	Chapters 2 and 14
Mykola Denysyuk	Chapters 36 and 37
Nef10	Chapter 8
Nicolas Miari	Chapter 45
Nikolai Ruhe	Chapters 13 and 18
Nirav Bhatt	Chapter 12
njuri	Chapter 14
NobodyNada	Chapters 8 and 45
NSNoob	Chapters 5 and 13
ok404	Chapter 51
Orlando	Chapters 13 and 36
Patrick	Chapters 13, 16, 18, 19, 21, 26, 27 and 37
Paulo Fierro	Chapters 14 and 36
pckill	Chapter 47
Peter DeWeese	Chapter 48
Peter N Lewis	Chapter 45
phi	Chapter 36
Rahul	Chapter 29
RamenChef	Chapter 13
Ravi Dhorajiya	Chapter 17
regetskcob	Chapter 3
Sanjay Mohnani	Chapter 38
shuvo	Chapter 35
Siddharth Sunil	Chapter 50
Sietse	Chapter 2
Spidy	Chapter 15
Stephen Leppik	Chapter 40
StrAbZ	Chapters 1 and 37
Sujania	Chapters 2, 4, 7, 15, 16, 19, 21, 24, 26 and 46
Sunil Sharma	Chapter 13
Tamarous	Chapter 51
Tapan Prakash	Chapters 7 and 45
tbodt	Chapters 7, 13, 14, 15, 34, 39, 45 and 51
ThatsJustCheesy	Chapter 12
Thomas Tempelmann	Chapter 45
Tricertops	Chapters 8, 13, 34 and 45
user1374	Chapters 3, 10, 26, 31, 41, 42 and 43
william205	Chapter 15
Yevhen Dubinin	Chapter 39