Learn Objective-C

原文地址http://cocoadevcentral.com/d/learn_objectivec/

译者前言

在网上看到这篇文章觉得写的很不错，但是貌似没有人翻译成中文，我就大胆翻译了。本人做软件开发6年了，但是大多数是在Windows平台上，最近才在我的机器上装了一个leopard，在Objective－C上也是一个新手，对于本文翻译的是否到位心里还在打鼓。如果有觉得翻译不对不好的地方，请与我联系，我的油箱是：cchenhao at gmail dot com。

cc很好☺，谢谢大家。

Objective-C

Objective-C是开发Mac软件的主要编程语言。如果你了解一些面向对象的基本概念和C语言，那么会对你学习Objective-C有很多帮助，如果你不了解C，那么建议你先读一下C 指南。

这篇指南由Scott Stevenson编写并做图。

方法调用

为了尽快开始，让我们先看一些例子。

调用一个对象的方法的基本语法是这样的：

```
[object method];
[object methodWithInput:input];
```

方法可以有返回值：

```
output = [object methodWithOutput];
output = [object methodWithInputAndOutput:input];
```

你还可以调用类的方法，这也是创建一个对象的办法。在下面的例子里，我们调用了NSSting类的string方法，用来返回一个新的NSString类的对象。

```
id myObject = [NSString string];
```

id类型意味着变量myObject可以是任意类型的对象。所以，当你编译这段代码时，它的实际类型以及它所实现的方法编译器是不知道的。

在我们的例子里，很显然对象的类型是NSString，所以我们可以改变对象的类型声明：

```
NSString* myString = [NSString string];
```

现在，这就是一个NSString类型的变量了，如果我们在这个对象上调用NSString类型对象不支持的方法，编译器就会发出警告。

注意：在对象类型的右面有一个星号（*），在Objective-C中，所有的对象变量都是指针类型。id类型已经被预定义为指针类型，所以不需要加一个星号。

嵌套调用

在许多编程语言中，嵌套的方法或函数调用像是这样的：

```
function1(function2());
```

function2的返回值做为输入参数传递给function1。在Objective-C中，嵌套调用看上去像是这样的：

```
[NSString stringWithFormat:[prefs format]];
```

要尽量避免在一行语句中进行两层以上的嵌套，这样会使代码的可读性降低。

多输入参数的方法

一些方法需要多个输入参数。在Objective-C中，一个方法的名字可以被拆分成几段，在头文件中，多输入参数的方法声明看上去像是这样的：

```
-(BOOL)writeToFile:(NSString *)path
atomically:(BOOL)useAuxiliaryFile;
```

你可以这样调用这个方法：

```
BOOL result = [myData writeToFile:@"/tmp/log.txt"
atomically:NO];
```

这些不是命名参数。在运行时环境中，该方法的名字实际上是

```
writeToFile:atomically:
```

访问器

在Objective-C中，所有的实例变量默认都是私有的，所以，在大多数情况下，你应该使用访问器来获取或设置这些变量的值。现在有两种语法。下面的是传统的1.x语法：

```
[photo setCation:@"Day at the Beach"];
output = [photo caption];
```

第2行代码不是直接读取实例变量。实际上它是在调用名为caption的方法。在Objective-C中，大多数情况你不用在获取器（getter）前面添加一个"get"前缀

在任何情况下，在方括号中代码都意味着你是在给一个对象或者一个类型发送一个消息（即一个方法调用）。

点操作符

在Mac OS X 10.5中，Objective-C 2.0新增了点操作符的设置器（setter）和获取器（getter）：

```
photo.caption = @”Day at the Beach”;
output = photo.caption;
```

两种语法你可以使用任何一种，但是在一个项目中最好只使用一种。同时，点语法只能使用在设置器（setter）和获取器（getter）上，而不能用于普通方法。

创建对象

创建对象有两种主要的办法。第一个是之前你看到的：

```
NSString* myString = [NSString string];
```

这是一种更加方便自然的方式。通过这种方法，你创建了一个**自动释放(autoreleased)**的对象，这一点我们会在后面看到更多的细节。尽管如此，在许多地方，你可能需要通过手工创建的方式来创建一个对象，如下：

```
NSString* myString = [[NSString alloc] init];
```

这是一个嵌套的方法调用。第一个是NSString类本身的alloc方法调用。这是一个相对低层的调用，它的作用是分配内存及实例化一个对象。

第二个是调用新创建对象的**init**方法。init方法通常做对象的初始化设置工作，比如创建实例变量。作为一个类的使用者，你无法知道这些方法的实现细节。

在某些情况下，你可以使用**init**方法的另外一种版本，这些版本带有输入参数：

```
NSNumber* value = [[NSNumber alloc] initWithFloat:1.0];
```

内存管理基础

当你为Mac OS X编写应用程序时，你可以选择允许垃圾回收。这意味着如果不是在特别复杂的情况下，你不用考虑内存管理。

然而，你并不会总是工作在支持垃圾回收的环境中。这样的话，你就需要知道一些基本概念。

如果你通过手工alloc的方式创建一个对象，之后你需要release这个对象。同样，你也不能手工释放(release)一个能自动释放(autoreleased)的对象，因为这将会使你的应用程序崩溃。

以下是两个例子：

```
//string1 将被自动释放
NSString* string1 = [NSString string];

//必须在用完后手工释放
NSString* string2 = [[NSString alloc] init];
[string2 release];
```

在这里，你可以认为自动释放对象会在当前函数结束的时候被自动释放。

关于内存管理要学的东西很多，但是我们先了解一下其他的概念，这样我们会有更多的认识。

设计类接口

在Objective-C的语法中，创建一个类是非常简单的。一个类通常分为两部分。

类的接口（interface）通常存放在类似**ClassName.h**的文件中，在这里，我们定义实例变量和公用（public）方法。

类的实现存放在**ClassName.m**这样的文件中，它包含了这些方法的实际实现代码。它通常还定义了客户类不能访问的私有（private）方法。

一个接口文件看上去像以下这样的。这个类名字叫做Photo，所以接口文件名是**Photo.h**：

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

```
@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}
@end
```

首先，我们导入了**Cocoa.h**，目的是将Cocoa应用程序的基本类添加进来。**#import**指令会自动防止将同一个文件导入多次。

@interface表明这是类**Photo**的声明。冒号后面指定父类(superclass)，这里父类是NSObject。

在花括号里面声明了两个实例变量：**caption**和**photographer**。都是NSString类型，实例变量可以是任何对象类型，包括id类型。

最后，**@end**符号结束类的声明。

添加方法

让我们给实力变量添加一些获取器（getter）

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

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}

- caption;
- photographer;

@end
```

记住，Objective-C语言中通常省略方法的“get”前缀。方法名字前面的单个减号(-)表明该方法是一个实例方法。如果方法名字前面是一个加号(+)，则表明该方法是一个类（static）方法。

编译器会默认一个方法的返回值是一个id类型的对象，所有的输入参数也默认是id类型。上述代码在技术上是正确的，但是我们一般不这样写，我们需要给这些方法指定返回值类型。

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

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}

- (NSString*)caption;
```

```
- (NSString*)photographer;

@end
```

现在，我们来添加设置器(setter)：

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

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}

- (NSString*)caption;
- (NSString*)photographer;

- (void) setCaption: (NSString*)input;
- (void) setPhotographer: (NSString*)input;

@end
```

设置器不需要有返回值，所以我们指定返回值是**void**。

类实现

现在,我们从获取器(getter)开始，来创建一个类的实现。

```
#import "Photo.h"

@implementation Photo

- (NSString*) caption {
    return caption;
}

- (NSString*) photographer {
    return photographer;
}

@end
```

这段代码以@implementation和类的名字开始，并且像接口一样，有一个@end。所有的方法必须写在这两条语句之间。

如果你写过代码，就会觉得上面的获取器看上去很熟悉，所以我们还是来看一看设置器，它们需要多一点解释。

```
- (void) setCaption: (NSString*)input
{
    [caption autorelease];
    caption = [input retain];
}

- (void) setPhotographer: (NSString*)input
{
    [photographer autorelease];
    photographer = [input retain];
}
```

每一个设置器都要处理两个变量，第一个是当前引用的对象，第二个是新输入的对象。在带有垃圾回收机制的环境中，我们可以直接设置成新的值。

```
- (void) setCaption: (NSString*)input
{
    caption = input;
}
```

但是，如果你不能使用垃圾回收，你需要**release**旧的对象，并且**retain**新的对象。

释放一个对象的引用实际上有两种方法：**release** 和 **autorelease**。标准的release会立刻释放对象的引用。autorelease会等一会儿才释放，但是引用实际上会一直存在，直到当前方法结束（除非你添加自定义的代码来明确的改变它）。

在设置器里面使用autorelease方法会更加安全一些，因为要改变的变量的新旧两个值可能指向的是同一个对象。而你可能不希望立刻释放实际上你要保留的对象。

现在，这看上去有点让人迷惑，但是随着你的不断学习，你就会有更多的认识。所以，现在不必彻底的理解这些。

Init

我们可以创建一个init方法用来给我们的实例变量设置初始化值：

```
- (id) init
{
    if ( self = [super init] )
    {
        [self setCaption:@"Default Caption"];
        [self setPhotographer:@"Default Photographer"];
    }
    return self;
}
```

这段代码是完全不需要加以说明的，尽管第二行看上去有点不常见。它是一个单个的等号（=），作用是将**[super init]**的结果赋值给**self**。

这实际上是要求父类做（父类的）初始化操作。if语句的作用是在尝试设置（本对象的）缺省值之前验证父类是否初始化成功。

Dealloc

dealloc方法在一个对象从内存中删除时被调用。通常在这个方法里面释放所有对象里的实例变量。

```
- (void) dealloc
{
    [caption release];
    [photographer release];
    [super dealloc];
}
```

在前两行，我们直接调用了实例变量的release方法。在这里，我们不需要使用autorelease，因为标准的release更快一些。

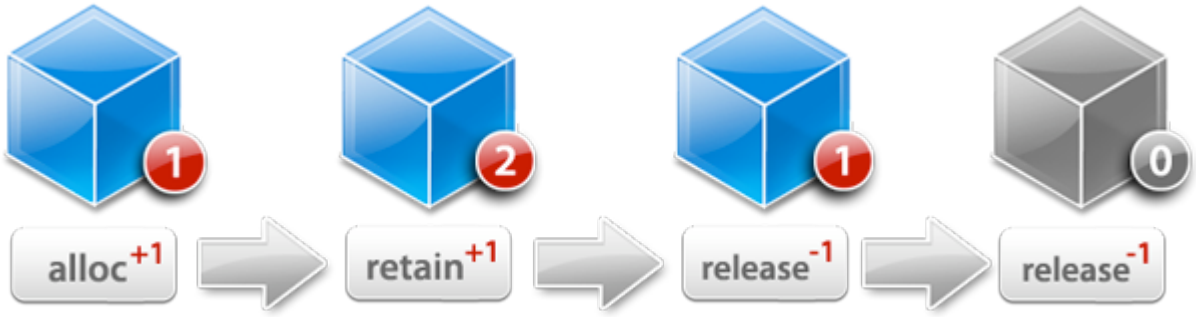
最后一行非常重要，我们发送了一个[super dealloc]消息，要求父类做清理工作。如果我们不做的话，该对象就不会被从内存中删除，这就造成了内存泄露。

当启用垃圾回收机制时，对象的dealloc方法不会被调用。此时，你可以实现一个finalize方法来代替它。

内存管理

Objective-C的内存管理是基于引用计数的。你要做的事情只是关注你的引用，而释放内存的工作实际上由运行环境完成。

在最简单的情形中，你分配的（alloc）对象，或者是保留（retain）在一些地方的对象，都需要给他们发送一个release消息。这也意味着，如果你使用了一次alloc，然后又retain了一次，那么你需要release两次才能释放该对象的内存。



这就是引用计数的理论。在实际应用中，通常只有两个原因我们才会创建一个对象：

1．作为一个实例变量保留。

2．在函数内部作为临时变量使用。

大多数情况下，一个实例变量的设置器（setter）会自动释放（autorelease）原来引用的对象，同时保留（retain）新的。你只需要保证在dealloc函数中释放（release）了它就行了。

那么，我们实际要做的工作就只有管理函数内部的本地引用了。在这里只有一条规则：如果过你通过alloc或者copy创建了一个对象，在函数结尾的地方给它发送一个

release或者autorelease消息就行了。如果你是通过其它方式创建的对象，就什么也别做。

下面是第一个例子，管理实例变量：

```
- (void) setTotalAmount: (NSNumber*)input
{
    [totalAmount autorelease];
    totalAmount = [input retain];
}

- (void) dealloc
{
    [totalAmount release];
    [super dealloc];
}
```

下面是另外一个例子，关于本地引用。我们只需要释放通过alloc创建的对象就行了：

```
NSNumber* value1 = [[NSNumber alloc] initWithFloat:8.75];
NSNumber* value2 = [NSNumber numberWithFloat:14.78];

// only release value1, not value2
[value1 release];
```

下面是一个组合例子，将一个本地引用设置给实例变量：

```
NSNumber* value1 = [[NSNumber alloc] initWithFloat:8.75];
[self setTotal:value1];

NSNumber* value2 = [NSNumber numberWithFloat:14.78];
[self setTotal:value2];

[value1 release];
```

注意，不论你是不是把本地引用当成实例变量一样赋值，管理它们都是完全相同的。你不必考虑设置器（setter）是如何实现的。

如果你理解了这些，你就理解了关于Objective-C内存管理中90%你需要知道的内容。

日志记录

在Objective-C中，将日志信息输出到控制台是非常简单的。实际上NSLog()函数很像C语言里面的printf()函数，除了要用一个%@符号代表一个对象。

```
NSLog ( @"The current date and time is: %@", [NSDate date] );
```

你可以将一个对象的信息作为日志在控制台输出。NSLog函数调用该对象的description方法，并且将这个方法返回的NSString打印到控制台。你可以在你的类中重写这个方法以返回你自定义的字符串。

属性（Properties）

前面我们写了caption和author的访问方法，你可能也注意到了，那些代码很简单，应该可以写成具有更普遍意义的形式。

属性是Objective-C的一个特性，它允许我们自动生成访问器，同时还有其它方面的好处。我们用属性来改写一下Photo类。

之前的代码看上去是这样的：

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

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}
- (NSString*) caption;
- (NSString*) photographer;

- (void) setCaption: (NSString*)input;
- (void) setPhotographer: (NSString*)input;

@end
```

用属性改写后的代码看上去是这样的：

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

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}
@property (retain) NSString* caption;
@property (retain) NSString* photographer;

@end
```

@property是Objective-C语言的一个指令，通过它声明属性。带括号的”retain”指示设置器（setter）要保留输入值，该行后面的是指定属性的类型以及名称。

下面我们看看这个类的实现部分：

```
#import "Photo.h"
```

```
@implementation Photo

@synthesize caption;
@synthesize photographer;

- (void) dealloc
{
    [caption release];
    [photographer release];
    [super dealloc];
}

@end
```

@synthesize指令为我们主动生成了setter和getter，所以我们必须要做的就只有实现dealloc方法了。

只有当访问器不存在的时候，@synthesize才会自动生成访问器，所以，即使是使用@synthesize声明了一个属性，你仍然可以实现自定义的getter和setter。编译器只会自动生成你没有自定义的方法。

关于属性的声明还有很多选项，但是它们超出了本指南的范围。

在Nil上调用方法

在Objective-C中，nil对象的作用等同于很多其他语言的NULL指针。不同的地方在于，在nil上调用方法不会导致程序崩溃或抛出异常。

这种技术被用在很多地方，但是对于我们来讲，最主要的就是我们不用在调用一个对象的方法之前检查该对象是否为空。如果你调用了一个nil对象的方法并且该方法有返回值的话，你会得到一个nil返回值。

我们也可以用它来稍微改进一下我们的dealloc方法：

```
- (void) dealloc
{
    self.caption = nil;
    self.photographer = nil;
    [super dealloc];
}
```

可以这样做是因为当我们将nil赋值给一个实例变量，设置器(setter)会释放旧对象并且保留（retain）nil对象。这种做法对于dealloc来说更好一些，因为这样做避免了让变量指向一个随机的数据，而这个数据又恰好是另外一个对象。

注意，我们在这里使用了self.<var>语法，这表示我们使用的是setter，它会进行内存管理。如果我们仅仅是直接设置值，像下面这样，那就会产生内存泄露：

```
// incorrect. causes a memory leak.
// use self.caption to go through setter
caption = nil;
```

类目（Category）

类目是Objective-C中最有用的一个特性。实质上，类目允许你为一个已存在的类添加一些方法而不用子类化该类，也不需要你了解该类的实现细节。

这是特别有用的，因为你可以给一个内建的对象添加方法。当你想在你的应用程序里面给所有NSString类型的实例添加一个方法，你只需要添加一个类目，而不需要通过定义一个子类来添加该方法。

比如，我想给NSString添加一个方法以判断它是不是一个URL，写法就像这样：

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

@interface NSString (Utilities)
- (BOOL) isURL;
@end
```

这很像一个类的声明。不同的地方在于后面没有列出父类，并且在括号里面写了类目的名称。类目的名字可以随便取，但是最好能表达出你在类目中包含的方法所要做的事。

下面是一个实现。切记，这不是很好的检查URL的方法。我们只是为了说清楚类目的概念。

```
#import "NSString-Utilities.h"

@implementation NSString (Utilities)

- (BOOL) isURL
{
    if ( [self hasPrefix:@"http://"] )
        return YES;
    else
        return NO;
}

@end
```

现在，你可以使用NSString的这个方法了，下面的代码会在控制台打印“string1 is a URL”：

```
NSString* string1 = @"http://pixar.com/";
NSString* string2 = @"Pixar";

if ( [string1 isURL] )
    NSLog (@"string1 is a URL");

if ( [string2 isURL] )
    NSLog (@"string2 is a URL");
```

与子类不同，你不能通过类目来添加实例变量。但是你能通过类目重写（override）类中已经存在的方法，当然，重写的时候要特别小心。

记住，当你通过类目更改一个类的时候，这个更改会影响你这个应用程序中所有这个类的实例。

总结

本文介绍了Objective-C基本知识。就像你看到的一样，这个语言学习起来非常简单。它没有很多特殊的语法，并且许多规范也在Cocoa中被多次使用。

如果你喜欢上面那些例子，你可以下载后面的工程然后察看它的源代码。

LearnObjectiveC Xcode 3.0 Project (56k)
http://cocoadevcentral.com/downloads/LearnObjectiveC-20080414a.zip

谢谢。