从零开始学iOS7开发系列3-我的地盘我做主-Cha4

原文及示例代码来自raywenderlich store中的iOS Apprentice 系列3教程，经过翻译和改编。

今晚累得够呛，不过只要有人来我还是欢迎~欢迎继续我们的学习。

好吧，这一课的主题还是——Objective-C的理论知识。如果你对这些理论知识一点都没兴趣，也可以完全跳过去再说。

上一课的内容中我们复习了条件语句，接下来先看看循环loop
在之前的教程中我们已经接触了for循环语句。

```
for (ChecklistItem *item in self.items) {
if (!item.checked) {
count += 1; }
}
```

使用上面的循环语句，会遍历self.items这个数组中的每个对象。这种for循环语句又被称为fast enumeration（快速枚举），当然快速枚举这种for循环方式只适用于某些集合对象，比如NSArray和NSDictionary。
当然，如果你之前学过其它编程语言，而这种编程语言又没有提供快速枚举，那么你可能对下面的这种语法结构更为熟悉~

```
for (int i = start; i < end; i++) {
// statements
}
```

对于for后面圆括号中的内容，可以读作：(起始条件；终止条件；递增操作）
比如，当你看到下面的代码：

```
for (int i = 0; i < 4; i++) {
 NSLog(@"%d", i);
}
```

这就表示我们有了一个for 循环，而且它可以：
1.有一个循环计数器i,从0开始计数
2.当i不再小于4时循环结束（也就是等于4的时候就结束）
3.每次iteration（迭代）后i的数值会自动加1

对于上面的代码，会在Debug调试区显示下面的内容：

```
0
1
2
3
```

虽然看起来有点奇怪，不过for (int i=0;i<X;i++)是最常用的表达方式，也就是说循环会重复X次

注意这里的i++起始就是i+=1的意思，如果我们只想显示偶数，那么上面的代码就可以改成：

```
for (int i = 0; i < 4; i += 2) {
NSLog(@"%d", i);
}
```

注意变量i只会在这个循环体中生存！我们不能在循环体语句外使用i这个变量，所以它的生命周期其实要比本地变量还要短。
如果我们采用这种标准的方式来遍历ChecklistItems数值，那么代码就可以改写成：

```
for (int i = 0; i < [self.items count]; i++) {
ChecklistItem *item = [self.items objectAtIndex:i];
```

```
if (!item.checked) {
count += 1; }
}
```

这种方式首先增加了代码量，其次其运行速度也比快速枚举的方式要慢的多。
所以在iOS开发中我们经常使用快速枚举替代C语言中常见的标准for循环。

当然，for语句不是实现循环的唯一方式。另一种非常有用的循环结构是while语句：

```
while (something is true) {
// statements
}
```

while循环会重复执行其中的语句，直到圆括号中的条件判断变成false（在iOS中就是NO)。
我们还可以换种写法：

```
do {
// statements
} while (something is true);
```

和之前那种while循环的唯一区别是，虽然这里也会对条件进行判断，但无论如何都会执行执行花括号中的语句一次。

你可以把前一种while循环看做是麦当劳肯德基式的快餐，要吃KFC等先掏钱。那么后面的do while循环则是一般的酒楼饭店，一般是先点菜进餐后买单。假如你想吃霸王餐（练就了金钟罩铁布衫），显然就得选择KFC，否则连机会都不给。

说到KFC我就伤心，貌似从去年起我最爱的原味鸡就被取消了，从此相见不相识，假装和它是路人！

不作死就不会死，既然你执意要死我就只有成全你了好吗？！

大多数循环结构体都基本相似，即便看起来有些微妙的差异。不管是哪种循环体，都会重复执行一系列的语句，直到终止条件得到了满足。其实for,while或者do-while这几种循环语句的差异真不是那么大，当然在具体的情境下某种方式可能看起来更容易理解一些。

之前提到过，我们可以用return语句从某个方法中直接退出，在循环中也可以使用break语句直接退出，即便for圆括号里面的退出条件还没有被满足：

```
BOOL found = NO;
for (NSString *string in _array) {
        if ([string isEqualToString:text])
        {
                found = YES;
                break;
        }
}
```

在上面的例子中，会遍历数组中的字符串，如果某个字符串和特定的text数值相等（注意是相等而不是相同，注意用的是isEqualToString而不是等号=），就直接使用break;跳出循环。因为我们已经找到了自己想要的东西，继续执行下去也没有意义了。

当然，和break相反，还有一个continue语句。它不会直接跳出循环体本身，但是会结束本次循环而调到下一次循环。

关于对象

对象无处不在。对象不是GF或者BF，但对象是代码世界中真正的生命体。
对象会把数据（实例变量）和函数（方法）整合到一起以便重用。前提是，你设计对象的方式比较合理！

在使用Objective-C编程的过程中，我们用到了大量由苹果官方提供的对象，比如NSString,NSArray和UITableView，当然我们也会使用自定义的对象。

为了创建一个新的对象，我们需要一个包含了@interface的.h文件：

```
@interface MyObject : NSObject

@property (nonatomic, strong) NSString *text;

- (void)myMethod;

@end
```

然后还需要一个包含了@implementation的.m文件：

```
#import “MyObject.h"

@implementation MyObject {
int _anInstanceVariable;

}
- (id)init {
if ((self = [super init])) {
// statements
}
return self;
}
```

现在讲这些或许你大概能懂一点了，因为我们已经接触过很多次.h和.m文件。

所以，先感性认识，再理性认识，或许比起我朝的先一大堆理论知识，再给很少的实践要更容易上手些。

.h文件又被术语党称作header file（头文件），其中包含了所有你想让其它对象看到的属性和方法声明。而.m文件则包含了对象的实现细节。简单来说，.h文件描述了这个对象能做什么，而.m则描述了它是如何来完成的。

当然，实例变量因为是当前对象的方法实现代码中感兴趣的，所以通常会“hidden隐藏”在.m文件中，而不是像以往那样放在.h中。
.m的实现部分还需要包含在.h中所声明的方法的实现细节。有些方法只想在对象内部使用，而不想被其它对象所调用。
所以可以把.m文件看做对象的私有部分。

在之前版本的Objective-C编译器中，规定必须在.h头文件的@interface部分声明实例变量，但现在不是这样的了。当然，如果你看过其它老版本的教程，或者是你的前任程序猿遗留下来的应用，或许还会用老的编码习惯。但是最新的最佳实践是把实例变量的声明放到.m的@implementation部分。

再说方法声明，如果你想让别人知道你会开车，会打麻将，会战炉石，那么可以在.h中声明，因为这样别人就可以邀请你做这些事情。但对于你会下载tokyo hot，会黑入中情局高级账户这些事情，虽然你这么D,但也不想随便让别人知道的吧?

假定我们有另外的一个对象，比如说是一个视图控制器，它想要使用MyObject，那么就必须导入.h文件来了解MyObject有哪些方法和属性。

```
#import "MyObject.h"

@implementation SomeViewController

- (void)someMethod {
        MyObject *myObject = [[MyObject alloc] init];
        myObject.text = @"Hello, world";
        [myObject myMethod];

}
```

如果这些东西你是在.m中声明的，那么很遗憾程序自然会毫无悬念的崩溃掉。

为了创建一个新的对象，我们需要首先使用alloc来为它的数据分配所需的内存空间，然后接着用init让对象可以立即投入使用（也就是所谓的初始化，或者叫创建并初始化）。

init方法和任何其它方法一样，我们可以创建私人定制的初始化方法。

```
@implementation MyObject

- (id)init {
if ((self = [super init])) {
// perform your own initialization here

self.text = @"Hello, world";
}
return self;
}
```

当然，对于编写init初始化方法，还是有一些规定的。首先，我们必须调用super的init方法，并将结果返回给self。只有当self不是nil的时候才会继续对象的初始化工作。一旦完成这个初始化，我们就可以向调用该方法的对象返回self的值。
看起来颇有些奇怪，不过这就是Objective-C的方式。
通常一个对象可以有多个init方法。虽然我们只能也只需要初始化对象一次，但具体采用哪种初始化方法却可以根据具体情况而定。

比如对于UITableViewController表视图控制器来说，如果是从storyboard中自动加载的，就需要调用initWithCoder方法，如果是手动从nib文件中创建的，就需要使用initWithNibName方法，如果是手写代码（不用storyboard或nib）生成，就需要使用initWithStyle方法。
具体使用哪种初始化方法要因地制宜。

很多对象还提供了convenience constructors(有人翻译成便利构造器，我对此表示无语，先凑合用着吧），可以把alloc和init方法整合到一起：

```
// using alloc and init:
NSString *text = [[NSString alloc] initWithFormat:@"Item-%d", index];

// shorter version:
NSString *text = [NSString stringWithFormat:@"Item-%d", index];
```

好了，关于对象的初始化，就先了解这些吧。

接下来是属性

对一个对象来说，方法提供了它的功能函数，而实例变量则包含了它的数据。那么属性(properties)的作用何在？好吧，它们的主要作用就是为了让代码易读易写。

比如说MyObject有一个名为text的属性，那么就意味着我们可以写下面的代码：

```
MyObject *myObject = [[MyObject alloc] init];
...
myObject.text = @"Hello, world";
```

这就是所谓的dot syntax（点表示法），因为它使用小圆点把不同的元素分隔开。看到这熟悉的点表示法，来自Java,C++的朋友表示无比欢喜。
不过别高兴的太早，在Objective-C中，点表示法的使用仅限于属性，而不能像它语言那样用点表示法调用方法或函数。

比如MyObject是另一个对象（假如是一个视图控制器）的属性，那么就可以使用下面的方法来更改它的文本内容：

```
someViewController.myObject.text = @"How's the weather?";
```

如果没有属性，以上的操作在Objective-C中会变得很繁琐。

我们需要更改MyObject的定义，然后编写类似下面的代码：

```
@interface MyObject : NSObject

- (void)setText:(NSString *)newText;

- (NSString *)text;

@end
```

上面的setText方法就是所谓的setter方法，因为它会改变对象中的数值；而texxt方法就是所谓的getter方法，因为它会从对象中读取一个数值。

此时对以上方法的实现就会如下面的代码所示。注意我们还得显式的声明一个实例变量来保存text文本。

```
@implementation MyObject {
NSString *_text;
}
```

```
- (void)setText:(NSString *)newText {
    if (newText != _text) {
    _text = newText;
    }
}

- (NSString *)text
{
return _text;
}
@end
```

如果我们需要修改text的数值，就必须这样来做：

```
MyObject *myObject = [[MyObject alloc] init];
 ...
[myObject setText:@"Hello, world"];
```

或者当MyObject是视图控制器一部分的时候就必须这么做：

```
[[someViewController myObject] setText:@"How's the weather?"];
```

好吧，感谢你还有耐心看到这里。我知道很多从其它编程语言转职过来的朋友对方括号调用方法的语法非常厌恶。如果没有属性变量和点表示法，恐怕你的代码中会充满了这种方括号。

其实我个人也很讨厌方括号，虽然按照苹果官方的说法它会让代码的结构更加清晰易读，这点我并不否认。
但是！！！！在键盘上敲TMD方括号比起原点要费时间好不好，而且方括号很容易漏掉一个两个好不好！！！而且方括号的位置放错了（如果是用自动提示的话很容易受伤~）代码就是垃圾好不好！！！

可惜这是苹果的地盘，既然选择了苹果，选择了Objective-C，你就得接受它的方括号表示法~

就好比你生在天朝，就默认选择了地狱模式。你抱怨没有用，有干爹或者有本事就移民出国，否则就凭自己的努力吧。
虽然输方括号时间久了会让手抽筋，不过如果能换来数钱数的手抽筋貌似也还是可以接受的~

好了，YY无用，说了上面的一大堆废话，无非就是说属性变量可以代替setter和getter方法，让编译器帮我们处理内部的机制（虽然我们没写上面的那堆代码，但编译器暗地里替我们做了类似的处理，感谢万能的编译器~）

这就意味着我们可以用下面的代码

```
@property (nonatomic, strong) NSString *text;
```

编译器看到这行代码就会自动为我们的对象添加setText和text方法，以及用来保存该属性变量的实例变量。而我们要做的就是使用点表示法来直接读写text的数值。

当然，属性的伟大力量不仅仅在于省事。如果我们想让别人了解对象所拥有的数据，也必须在@interface部分声明属性变量。因为我们使用了属性变量，其它对象就没机会直接访问对象内部的实例变量，而是首先来使用getter和setter方法。

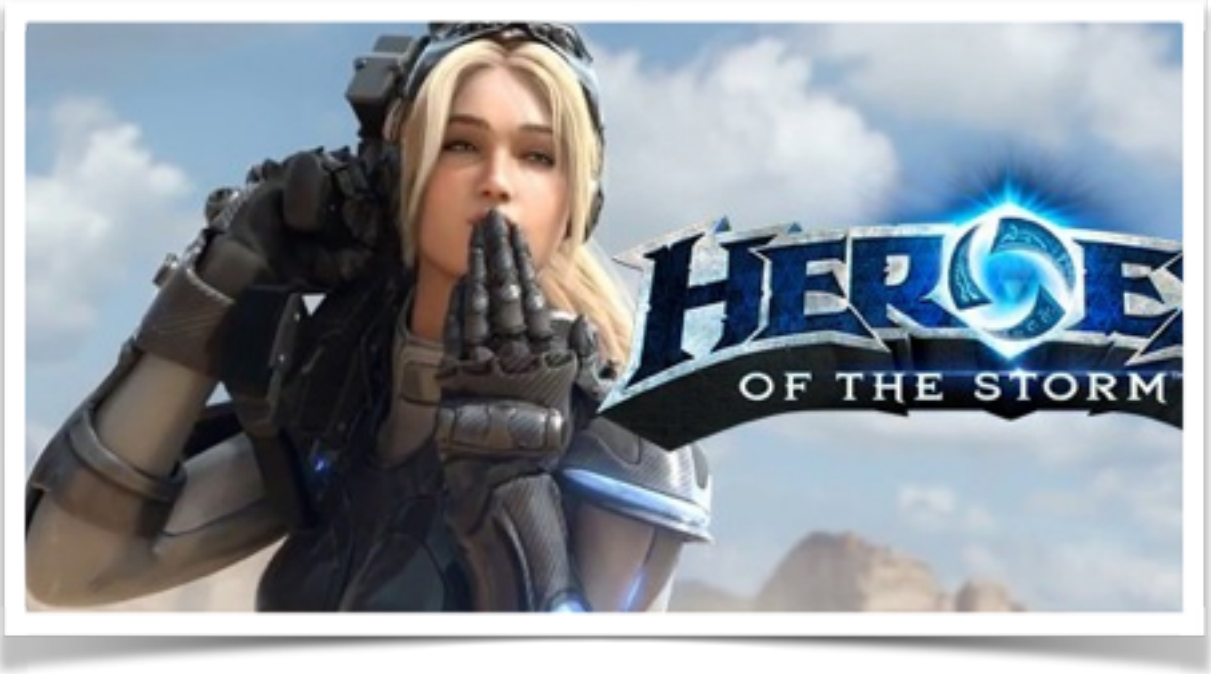
注意，关键词IBOutlet是告诉编译器该属性变量和Interface Builder里面的某个控件关联 在一起。
此外在声明属性变量的时候还会用到类似nonatomic和strong之类的关键字，关于它们的作用在后面会详细解释。
在老版本的Xcode中，对某个属性变量都必须synthesize，这就意味着我们必须在.m文件中为每个属性添加一行代码：

```
@synthesize text = _text;
```

幸运的是Objective-C的设计人员认为这个事情应该交给编译器来完成，程序猿还是要把主要精力放到创造新事物上面。当然，不可否认还有些情况下我们不得不使用@synthesize语句，不过对目前来说，大多数情况下可以忽略这行代码了。

好了，我也想尽快结束理论知识复习，不过考虑到我们在之前的学习中一直在实战，对于这些基础知识缺课了不少，现在看看还是有帮助的。
不过还是那句话，如果你不喜欢看理论知识，可以继续skip to my lou~

据说风暴英雄要开始测试了，期待马上测试！



福利送上