从零开始学iOS7开发系列3-我的地盘我做主-Cha3

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。

在火炉边找个位置坐下吧，欢迎继续我们的学习。

这一课的主题还是关于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。

关于方法(methods)

现在我们已经知道对象是构成iOS应用的基础，而对象又由数据和功能构成。数据通常保存在变量中，而功能则是由methods（方法）来提供的。
说实话，无论是用方法还是函数来表示这个概念其实都不算很贴切。这里的方法不是哲学里面的方法论，也不是我们常说的解决方法，而是指一个对象能做的事情。

下面是方法的例子：

```
// 没有任何参数，也不返回数值的方法
- (void)viewDidLoad;

//只有一个参数，但并不返回数值的方法(IBAction 和 void其实是同义词，只有Xcode才懂)
- (IBAction)showAlert:(id)sender;

// 有一个参数，而且会返回一个布尔类型数值的方法

- (BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation;

// 带有两个参数tableView 和 section,

// 而且会返回一个NSInteger类型变量的方法

- (NSInteger)tableView:(UITableView *)tableView

numberOfRowsInSection:(NSInteger)section;

// 带有两个参数tableView 和 indexPath,

// 而且会返回一个NSIndexPath对象的方法

- (NSIndexPath *)tableView:(UITableView *)tableView

willSelectRowAtIndexPath:(NSIndexPath *)indexPath;
```

在Objective-C中，方法可能会有很长的名称。以上方法的全称是如下这样的：

```
viewDidLoad
showAlert: shouldAutorotateToInterfaceOrientation:
```

```
tableView:numberOfRowsInSection:
tableView:willSelectRowAtIndexPath:
```

注意这里的冒号:也是方法名称的一部分。viewDidLoad方法没有参数，所有方法名称中没有冒号，但showAlert:有一个参数，因此需要冒号。之前我们曾简单提到过selectors（选择器）和动态方法命名，但无论在何种情况下，使用恰当的方法名称都是有必要的。如果使用了错误的方法名称，应用就会崩溃。

比如：

```
// this refers to a method with no parameters

@selector(showAlert)

// this refers to a method with one parameter

@selector(showAlert:)
```

关于selectors和这一类型的错误，后面还会详细解释。

当我们需要对一个对象调用方法的时候，要用到[]括号：

```
[self loadChecklists];

[lists addObject:checklist];

[self configureTextForCell:cell withChecklistItem:item];
```

在这种代码结构中，括号中的第一个单词表示要调用方法的对象。
这里的self是一个特殊的关键字，它表示会对自身调用该方法。
当我们调用一个方法的时候，应用会跳转到方法的实现代码，按顺序执行其中的所有语句，然后根据需要向调用方法的对象返回一个数值：

```
- (int)performUselessCalculation:(int)a {

int b = arc4random_uniform(100); int c = a / 2;
return (a + b) * c;

}
```

因为Objective-C是源于C语言的，因此它支持C语言的主要特性。比如说我们也可以使用C语言中的函数(functions)，比如arc4random_uniform()。函数和方法的作用是完全相同的，它们都是把具备某种功能的代码块放在一个可以重用的小单元中。但区别在于，函数这个生命体不依赖于某个特定的对象。如果你是从C语言转过来的，很可能还习惯于使用函数来解决一切问题，但在面向对象的编程法则中，方法的使用更受青睐。

通常来说，大多数方法都有一到多个参数，这样我们就可以利用来自不同地方的多个数据源来完成某件任务。如果某个方法和固定的数据源关系紧密，那么可重用性就大打折扣。

例如，下面的方法是没有参数的：

```
- (int)addValuesFromArray {
```

```
int total = 0;
for (NSNumber *number in _array) {

total += [number intValue]; }

return total; }
```

这里的_array变量是个实例变量。那么此时的addValuesFromArray方法就和这个实例变量紧紧绑定在一起了，如果没有这个实例变量的存在，这个方法也就没有存在的意义了。

假定我们需要在应用中添加另一个数组，而且想进行类似的计算。那么你就只能选择拷贝粘贴以上的方法代码，然后改变其中的变量名称了。当然，如果你真这么做，那只能说你是劳模命，或者是标准的码农行为。
真正的程序猿会给这个方法添加一个参数，而这个参数就是我们要执行计算的数组。
这样一来这个方法就不会依赖于实例变量的存在了：

```
- (int)addValuesFromArray:(NSArray *)array

{
int total = 0;
for (NSNumber *number in array) {

total += [number intValue]; }

return total; }
```

现在我们可以在调用该方法时使用任何NSArray对象作为其参数。
这里举的例子并不是说我们不能在方法体中直接使用实例变量，但通过给方法增加参数的形式如果可以让这个方法变得更加通用和独立，那么何乐而不为之呢？

在方法中我们可以做下面这些事情：
（1）创建本地变量
（2）使用+,-,*,/,%等数学运算符进行简单的计算
（3）将新的数值保存在变量中（包括本地变量和实例变量）。
（4）调用其它方法。
（5）通过使用if或switch等条件判断语句来做决策
（6）通过使用for或while等语句来执行重复的操作
（7）向调用该方法的对象返回一个数值。

下面让我们来仔细看看if和for 语句。

自动决策机

如果你是个犹豫不决的人（比如买彩票不知道买什么号，在超市买东西不知道排哪一队，买手机不知道挑多大容量的，等等，总之就是决策困难症严重患者吧），那么现在有一个超级神器可以帮你解决这个需求了-史上第一台全人工智能助理-伟大的自动决策机出现了！
你只需要输入几个定制化的条件，然后再输入几种可选方案，我们的自动决策机会帮你解决一切！
比如：

```
if(money <= 799){
        text = @“买个毛线的手机，还是回家吃泡面吧，记得找你妹借买泡面的钱”;
}else if(money < 999){
        text = @“不用多想了，就买荣耀3C吧”;
}else if(money < 5299){
```

```
        text = @“传说中的米粉就是这样炼成的”;
}else if(money < 10000){
        text = @“土豪金，我为你代言！”;
}
```

圆括号()里面的语句就是所谓的condition（条件语句），如果一个条件语句的结果是true（在Objective-C里面就是YES），那么就会执行{}花括号里面的语句。如果不是，则会继续else部分的条件语句判断。

在Objective-C中，我们使用relational operators（关系操作符）来比较两个数值的大小：
== 等于
!= 不等于
> 大于
>= 大于或等于
< 小于
<= 小于或等于

我们还可以使用logical （逻辑）操作符来整合两个表达式：
a&&b 只有当a和b都是true的时候才是true，也就是所谓的逻辑与
a||b 当a或者b有一个是true的时候就会是true,也就是所谓的逻辑或
当然，还有逻辑非!，它会颠倒是非黑白~

说到逻辑操作符，可以顺便扯一扯数字电路。虽然和编程关系不大，但如果哪天你不小心穿越回古代了，这点小知识说不定能让你荫爵封王呢。
数字电路对于没上过大学理工科的朋友可能有点恼火，不过相信小学、中学里面都有物理课，而里面都会学到电路的概念。其实中学物理里面的电路基本上属于模拟电路（至少我当年学的时候，现在不知道了），经常让你计算什么电压、电流、电阻神马的。特别是用交流电来供电的时候，电压是一个波形的，和直流电完全不同。而既然是数字电路，那么显然就是0和1的世界。怎么把连续波动的模拟电路转换成只有0和1两种状态的数字电路呢？看起来很复杂，但真正核心的道理却很简单。
首先要设置一个标准，比如3v，那么如果在某个周期内检测到的平均电压大于或者等于这个值，就说它是高电平，也就是1，反之就是0。虽然实际的A/D,D/A转换电路没这么简单，但我们只要知道这一点就行了。
感兴趣的朋友可以自行去了解下数字电路和模拟电路这门课，包括如何设计A/D,D/A转换电路之类的东东，这里就不多废话了。

虽然说我们这个系列的主旨是学习iOS开发，不过作为一个软件开发人员（或者对软件开发感兴趣的人），适当了解下计算机硬件的原理绝无坏处（以后穿越时候的必杀技）。

好了，继续我们的条件判断语句。
我们还可以使用()圆括号把表达语句分隔开：

```
if (((this && that) || (such && so)) && !other) {

// statements

}
```

这样就相对比较好理解了：

```
if (((this and that) or (such and so)) and not other) {

 // statements

}
```

如果你觉得上面的表述有点非人类，还可以用下面这种方式来理解：

```
if (
(this and that)

or (such and so)

) and (

(not other) )
```

当然，这个语句写的越复杂，你就越难理解自己究竟要干啥，或者说你的极品前任程序猿究竟想干啥。
注意，当我们使用==操作符的时候，通常是比较所谓的字面量。如果用==来比较对象，就表示在比较两个真实的对象（不仅仅是它们所包含的内容，还包括它们自身的身份！！！）：

```
- (BOOL)compare:(NSString *)a with:(NSString *)b {

return (a == b); }
```

只有当a和b实际上是同一个对象时结果才是YES。
比如：

```
NSString *a = @"Hello, world"; NSString *b = a;
if ([self compare:a with:b])

{

NSLog(@"They are equal!");

}
```

在上面的这种情况下a和b才是完全相同的。
而在下面这种情况下虽然它们的内容完全相同，但却不是同一个对象！！！

```
NSString *a = @"Hello, world";
NSString *b = [NSString stringWithFormat:

@"Hello, %@", @"world"];

if ([self compare:a with:b]) {

NSLog(@"They are equal!"); // 不会有结果的~

}
```

使用==将两个对象进行比较是初学iOS开发的童鞋最常犯的错误之一。
如果我们只是想比较两个对象中所保存的值，而不是它们自身是否占用了同样的内存，那么就应该用isEqual或者isEqualToString：
(BOOL)compare:(NSString *)a with:(NSString *)b {

```
return [a isEqualToString:b];
}
```
除了if语句，还有个使用频率相对低一些的条件判断语句switch：
```
switch (condition) {
case value1:
// statements
break;
case value2:
// statements
break;
case value3:
// statements
break;
default:
// statements
break;
}
```

它的工作原理和if是相同的，只不过更适合使用单一数值判断的情况，而不是使用范围判断的情况。
比如上面的语句也可以用if语句改写成：
```
if (condition == value1) {
// statements
} else if (condition == value2) {
// statements
} else if (condition == value3) {
// statements
} else {
// statements
}
```

switch语句在某些情况下更方便，而且它的执行速度显然要比if-else更高，因为只需要对条件进行一次判断，但灵活性相比if-else更差，因为在case中的数值只能是int类型的常数。
因此你不要指望可以用下面的语句帮到你：
```
NSString *text = ...;
switch (text) {
case @"A":
// statements
break;

case @"B":
// statements
break;
}
```
上面的用法是错误的。
还有一个初学者常犯的错误，在使用switch 的时候要在每一种情况后面加上break;否则，你就等着哭吧~
此外，使用if和return的组合还可以从方法体中提前跳出，而不必担心某些意外情况的发生。
```
- (int)divide:(int)a by:(int)b {
if (b == 0) {
NSLog(@"You really shouldn't divide by zero");
return 0;
}
return a / b;
}
```
上面的方法避免了当除数是0的时候会发生的意外。

此外，if和return还可以在不返回数值的方法中使用，比如：
```
- (void)performDifficultCalculation:(NSArray *)list
{
if ([list count] < 2) {
```

```
NSLog(@"Too few items in list");
return;
}
// perform the very difficult calculation
}
```
在这种情况下，return的意思是说，“这个方法就到此结束吧，别费劲了”return后的所有语句都将被忽略，程序会将操控权交回调用它的对象。

当然，用下面的写法也是可以的：
```
- (void)performDifficultCalculation:(NSArray *)list {
if ([list count] < 2) { NSLog(@"Too few items in list");
} else {
// perform the very difficult calculation
}
}
```
有时候我们会看到一环套一环的if语句：

```
- (void)someMethod {
if (condition1) {
        if (condition2) {
                if (condition3) {
                        if (condition4) {
                        // statements
                        } else {
                        // statements
                        }
                } else {
                // statements
                }
                } else {
                // statements
                }
                } else {

                // statements
                }
}
```
虽然写的时候很省事，但对于这种极品前程序猿，只有两个字送给他们：滚粗！
个人推荐下面的写法：
```
- (void)someMethod {
if (!condition1) {
 // statements return;
}
if (!condition2) {
 // statements return;
}
if (!condition3) {
// statements return;
}
if (!condition4) {
 // statements return;
}
// statements
}
```
虽然两种写法的结果是相同的，但前一种写法显然是反人类的。记住，真正的高手写的程序不仅仅是机器容易看懂的，也是人可以看懂的。现在的项目都是团队合作型的，即便是one-man studio，你也不是一个人在战斗。因为你需要去阅读开源代码，去参考官方示例，即便你全部自己搞定，也需要穿越时空问3年前的自己当时写的那段代码是TMD神马意思，为神马现在根本看不懂？
理论？又是理论？
是的，不但这一课是理论，下一课还是。如果你看不下去了，可以先去LOL或者炉石玩两把。

送上今日福利（2014年1月20日）

春节快到了，买到回家的火车票了吗?

光有车票还不行，还得花个一百万租个女友回家过年，还得是博士或者处女，一般人还不够格。