从零开始学iOS7开发系列3-我的地盘我做主-Cha12

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。

这可是你自找的！

好了，在法师女神吉安娜的招呼声中，我们开启了新一轮的战斗，哦不，新一课的学习。

在前一部分（1-11课）的内容中，我们学习了如何使用Core Location获取用户所在的地理位置信息，并进行反向地理编码。同时，我们还了解了防御型编程的概念，也就是从用户体验和产品健壮度出发，把各种意外的情况处理都考虑在内。

总的来说，再次强调一点，这个教程的主要目的不是教你学iOS开发，而是带你进入iOS的世界溜一圈，教你如何学习编程，了解程序猿的思维和工作方式，学会如何学习新的知识。

如果你是为了找个好工作来看该系列教程，建议不如报个绿翔技校或者西大神鸟神马的来得快。

之所以有这些废话，是因为接下来——又是让你惨不忍睹不忍继续看下去的—-理论知识了！！！

不管你看不看，我就继续了。至于不看的后果吗，嘿嘿，有女神吉安娜镇楼，小心我的魔法会把你撕成碎片哦~

到目前为止，我们的教程中把几乎所有的代码生物都叫做对象（object）。其实这样的叫法是不严谨的。神马？为毛到现在才告诉我？不知道我都跟小伙伴们炫耀好久了吗嗷嗷嗷！女神说了，这是你自找的，装B遭雷劈哦，它才不管你是否能渡劫成功。其实，为了更精确的贯彻面向对象编程的理念，我们要把对象(object)和它所属的类(class）区分开来。

当我们使用类似下面的代码时，

```
@interface ChecklistItem : NSObject
…

@end
```

实际上我们是在定义一个名为ChecklistItem的class（类），而不是一个object（对象）。

而当我们使用类似如下的代码时才是创建一个对象。

```
ChecklistItem *item = [[ChecklistItem alloc] init];
```

现在item这个变量包含了ChecklistItem类的一个对象。只要你看到该死的*星号，就知道它是一个对象了。你可以这样理解：item这个变量包含了ChecklistItem类的一个实例。在这里，对象和实例是一回事。不过对象和实例变量可不是一回事哦。

换句话说，ChecklistItem类的对象其实是item这个变量的datatype（数据类型）。
Objective-C语言中提供了一大堆内置的数据类型，当然我们也可以通过创建自定义类的方式来添加属于自己的数据类型。

为了更好说明类和实例/对象的区别，还是用个例子来说明吧。

咱两都是标准的吃货，还都是哈根达斯的爱好者（会不会有点伪娘的感觉啊？）。最近土豪大哥给发了5元钱的微信红包，于是哥两商量好干脆去大吃一顿哈根达斯。哈根达斯(Haeagen-Dasz)就是我们要吃的事物的类，为了简单起见，就用Dazs这个单词吧。

让我们创建一个哈根达斯类如下：

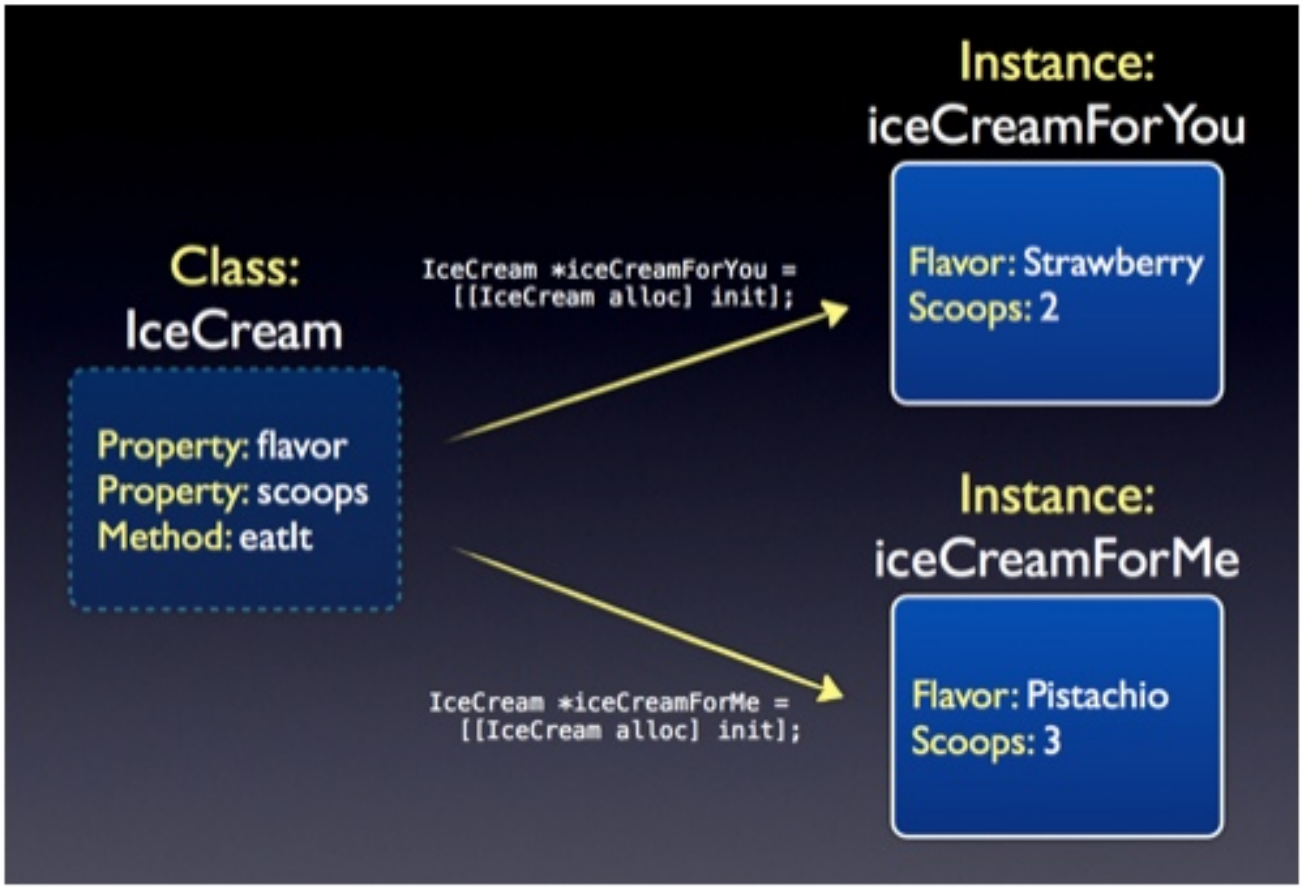
```
@interface Dazs : NSObject

@property (nonatomic, strong) NSString *flavor;
@property (nonatomic, assign) int scoops;

- (void)eatIt;

@end
```

咱两一起去了哈根达斯店，向小妹要了两个。
买完后，漂亮小妹递过来一个对你说，这是你的哈根达斯。你害羞的笑笑，指着旁边的我，不，弄错了，这是他的哈根达斯。

```
//你的那个

Dazs *iceCreamForYou = [[Dazs alloc] init];
dazsForYou.flavor = @"Strawberry";
dazsForYou.scoops = 2;


//我的那个

Dazs *iceCreamForMe = [[Dazs alloc] init];
dazsForMe.flavor = @“Pistachio”;
dazsForMe.scoops = 3;
```

好吧，貌似我的份量大点，这主要是因为RP比你好。当然不排除妹纸看我顺眼多加了点。

现在应用中有两个Dazs的实例了。其中一个是你的，另外一个是我的。不过这里只有一个类来描述我们在吃的食物-哈根达斯，但却有两个不同的对象。你的对象是草莓味的，我的是开心果味的。



Dazs类就好比一个模板或者模具，只要是这个类型的对象都有两个属性：flavor和scoops（它们的数值保存在同名的实例变量中），同时还有一个名为eatIt的方法。任何从这个模板新创建的实例都会具有这些属性，方法，不过却生活在属于自己的计算机内存中，并拥有自己的数值。

Inheritance（继承）

继续下一个重要的概念，inheritance（继承）。作为一个DS，每天在小面馆就着大蒜吸溜着二两炸酱面的时候，你会YY自己有一个目前定居在火星上的三爹的四舅的五叔的二爸的妹夫的干爹的女儿。她去韩国旅游回来后发现自己得了可怕的相思病然后很快不治，在临终前她翻看facebook上的一张老照片突然想起自己还有你这么个二货亲戚，于是决定把自己在南极的一个小岛转让给你。停停停停停，这样的梦我每天都在做，期待这种事情的发生和期待自己中双色球头奖一样渺茫。不过如果真有这种事，土豪，我们做个朋友吧。

这里要说的是class inheritance（类继承），它也是面向对象编程的一个重要思想。

通过继承，可以在之前的类的基础上创建一个新的类。新的类会继承父类的所有数据、功能，同时还可以在此基础上添加自己的新数据和功能。

以刚才的Dazs类为例，它是基于NSObject这个类创建的，而NSObject这个类是Objective-C中所有类的父类。大家可能已经习惯了苹果官方提供的类都是NS开头的，NS其实NextStep的缩写。关于这个历史渊源，可以参考系列1的内容，乔帮主是个怀（记）旧（仇）的人啊！

在@interface这一行中会这样写：

@interface Dazs : NSObject

这就意味着Dazs这个类是NSObject类的子类，不过有自己额外添加的属性和方法而已。

再重复下，NSObject是Objective-C语言中所有类的base class（基类，父类）。在iOS开发中，不管你是直接继承NSObject类，还是继承其它类，归根到底还是基于NSObject类。

比如之前我们看到过类似下面的代码：

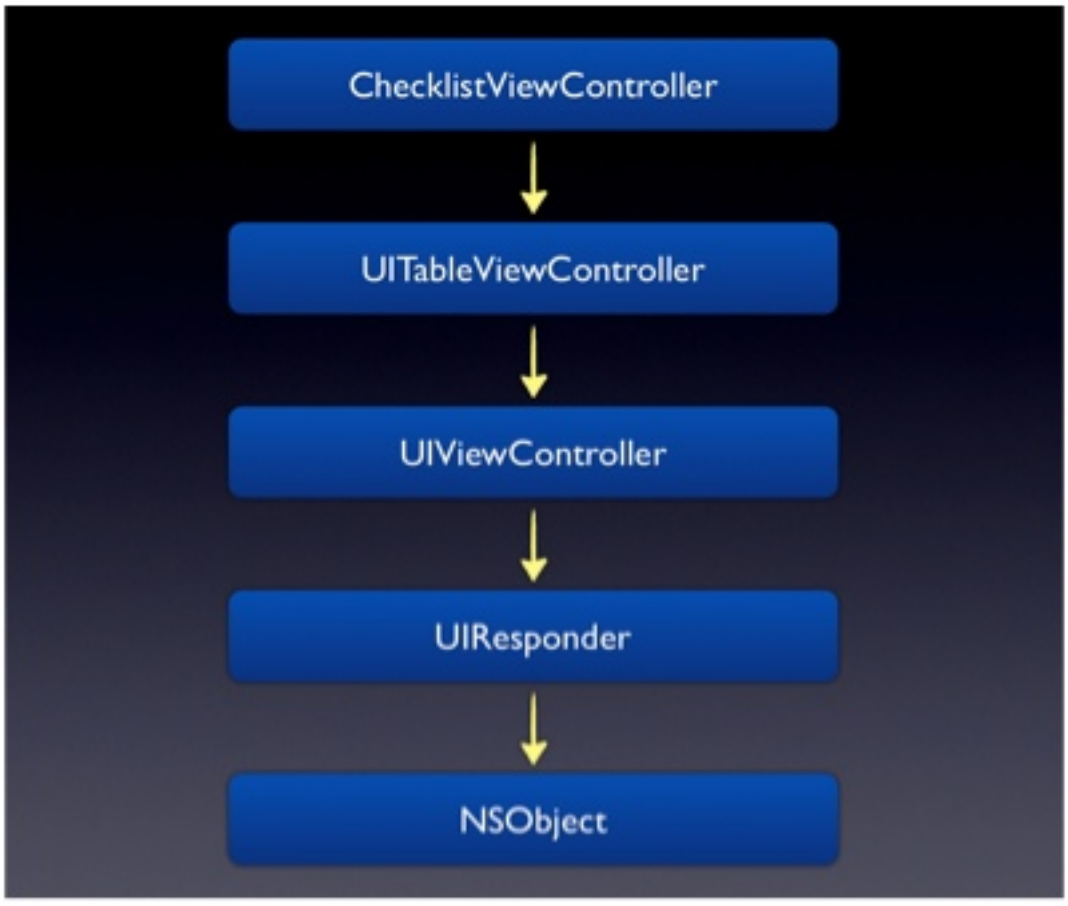
@interface ChecklistViewController : UITableViewController

ChecklistViewController类实际上是UITableViewController的子类，它可以实现一个UITableViewController类可以做的一切事情，同时还可以有自己的新数据和功能。

这种类继承机制非常管用。因为UITableViewController这个类本身就已经可以做很多事情了。它有自己的table view表视图，知道如何处理prototype cell和static cell，知道如何处理用户的上下滚动，还知道很多其它的事情。当我们基于它创建了自己的类时，只需要添加定制化的属性和方法就可以了。

如果所有的一切都需要从零开始创建，恐怕到现在为止你还在系列1里面晃悠。类继承机制让开发者可以轻松使用现有的众多代码功能。

当然，UITableViewController这个类又是基于UIViewController类的，而UIViewController类又是基于UIResponder类的，而最终都是基于NSObject类，这就是所谓的类继承树。



All classes stand on the shoulders of NSObject

看到这棵树，我想到了老子说过的话（不错，就是老子，你没看错！）：
道生一，一生二，二生三，三生万物。

在Objective-C的代码世界中，道是什么呢？

当然，对于类继承树中的节点来说，位于上面的节点（子类）比位于下面的节点（父类、基类）要更丰富多彩一些。

比如Objective-C的万物之母NSObject类只提供了所有对象都需要具备的基本功能。比如，它拥有创世用的alloc方法，使用这个神技可以为对象的实例变量预留内存空间，当然还有基本的init初始化方法。

UIViewController是所有视图控制器的基类。如果我们想创建属于自己的视图控制器，当然就是在UIViewController类的基础上扩展就好。在java等语言中使用extend关键字表示某个类继承自另一个类。当然常用的术语还有derive from或者base on。不管用哪个词，意思都是一样。

苹果的黄马甲们比较偏好:冒号，因此在创建继承自其它类的自定义类时也用的是冒号。

@interface ChecklistViewController : UITableViewController

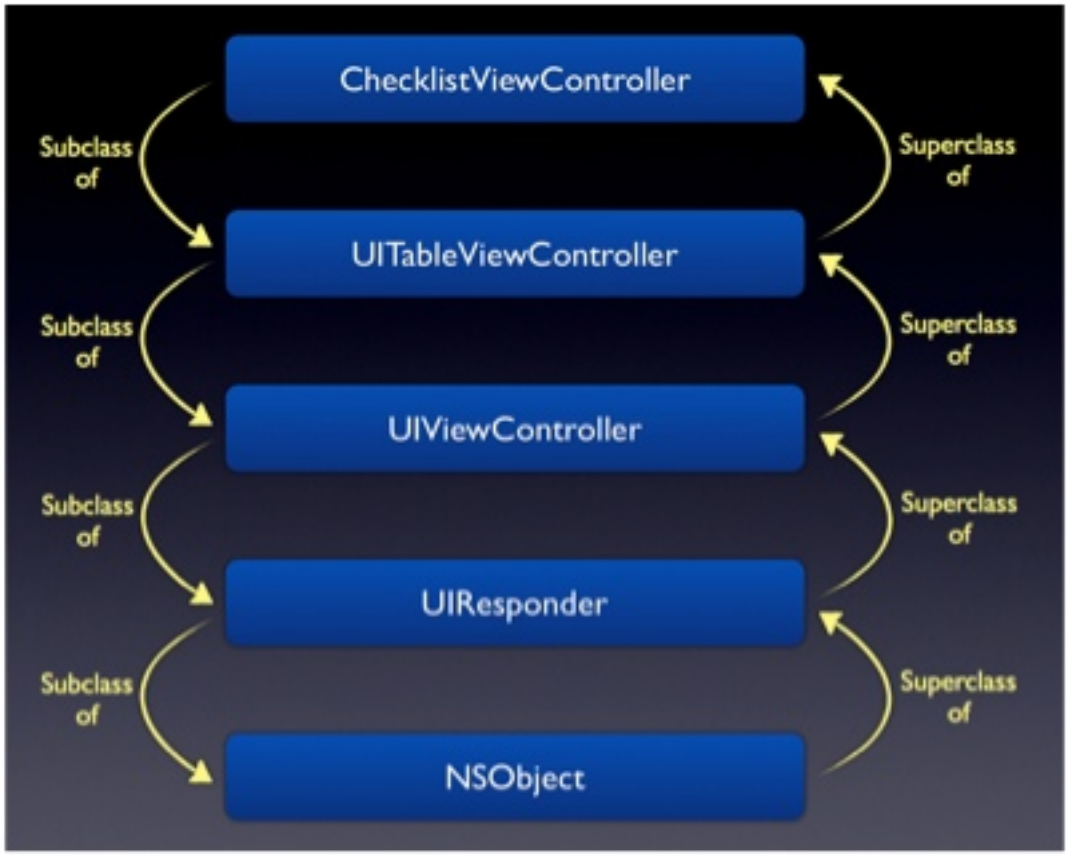
如果你看惯了其它语言的extend关键字，可能对此略有些不爽。不过习惯了就好，这不是你我的力量可以改变的。

作为一个开发者，最重要的是基于已有的语言、工具、框架来创建对用户有价值的产品。当然，如果你是个研究狂人，打算开发自己的编程语言，就当我这句话没说。
普通开发者肯定不想自己编写所有的界面和视图处理代码。这种事是由比我们聪明得多的Apple黄马甲们来搞定，而他们会把这些功能整合到UIViewController中，然后在每次iOS升级时更新API。我们要做的是创建一个继承自UIViewController类的自定义类，然后就可以免费获得苹果提供的所有功能了。接下来只需要添加自己的数据和逻辑就好。

如果你的应用界面主要用于处理table view表视图，那么就直接继承自UITableViewController就更好。这个类不但可以实现UIViewController类所做的一切（因为它继承自UIViewController类），还拥有专门处理表视图的神技。当然，如果你不嫌麻烦的话自然可以手动编写相关代码，但在你做这些事情的时候用户已经冲着你怒吼咆哮了—接受正义的制裁吧！

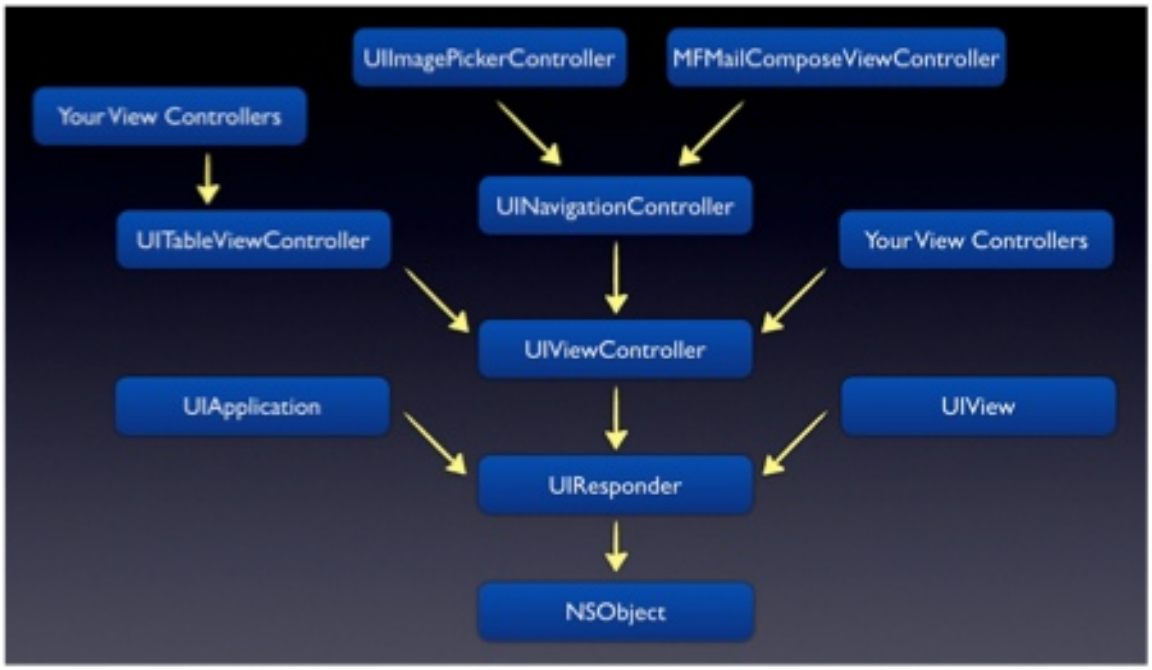
类继承机制可以帮你节省大量的时间和精力，让你专注于用创造性的思维解决用户的问题！

当程序猿们谈论继承的时候，通常还会用到superclass（超类，父类）和subclass（子类）这两个名词。在上面的例子中，UITableViewController是ChecklistViewController的直接父类，而ChecklistViewController则是UITableViewController的子类。



Superclass and subclass

在Objective-C中，一个类可能有很多子类，但只能有一个父类。也就是说，在Objective-C中，不能提供C++的多重继承机制。不然谁知道你是不是喜当爹啊?
有很多类继承自UIViewController类:



A small portion of the UIKit inheritance tree

考虑到在Objective-C中几乎所有的类都是继承自NSObject，因此它们形成了一个巨大的层级体系。
了解Objective-C中的类层级体系非常重要，因为这样我们才知道什么时候该继承哪个类来创建自己的自定义类。当然，类层级体系只是编程中众多层级体系的一个而已。
因为程序猿特殊的大脑构造，这个另类的生物群体对层级体系非常中意。

除了类，你会注意到协议也有类似的行为：

```
@protocol MyProtocol <NSObject>
...
@end
```

当一个@interface语句的后面出现尖括号<>时，就意味着这个类要遵从一个或多个协议。
而协议本身也可以这么做。
在上面的 例子中，MyProtocol这个协议本身要遵从NSObject协议。

当然，这种协议遵从协议的方式和类继承是两回事，不过原理是类似的。
这样一来，所有遵从MyProtocol协议的类也必须同时实现NSObject协议中的方法。

好吧，你可能要问，NSObject不是一个类吗？肿么还有一个NSObject协议？
这两看起来同名，不过因为属性的不同却是完全不同的两个生物。
NSObject类是iOS中所有类的基类，而NSObject协议则是iOS中所有协议所必须遵从的协议。希望你没有被这个绕口令搞糊涂。

如果你实在看不懂也没关系，对有些东西的理解是需要时间和经验的。
目前来说，作为开发者只需要知道，在iOS开发中所有的类最终都继承自NSObject类，而所有的自定义协议都至少需要遵从NSObject协议。

好了，今天的学习到此结束了。

无码的一天，是不是稍微有点无趣?

不过还是要提醒你，下一课仍然是无码的理论课哦。如果你不爽了，赶紧投奔绿翔吧~
在这下雪天，还是看看绿色比较养眼。

三次元的看多了，偶尔可以看看二次元的萌妹子