从零开始学iOS7开发系列3-我的地盘我做主-Cha13

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。

最近的天气好冷，这就是传说中的倒春寒吧？大家还是要多注意身体，小心感冒哦~

欢迎继续我们的学习。

让我们把理论知识进行到底吧。

首先来了解下Overriding methods（方法覆盖/重写）的概念。

为什么会有这样一个概念的存在呢？这个其实和我们上一课提到的类继承机制有关。当我们通过类继承的方式创建一个自定义类时，这个自定义的类也会继承来自父类的方法。
比如我们创建了一个新类MyClass:

```
@interface MyClass : NSObject
@end
```

那么在后面的代码中就可以这样来分配并初始化一个MyClass类的实例对象：

```
MyClass *myObject = [[MyClass alloc] init];
```

即便我们的MyClass类中没有显示声明一个alloc方法或一个init方法，上面的代码仍然是有效的。因为MyClass类继承自NSObject类，而NSObject类具有这样的方法，这就行了。也就是说MyClass类会免费拥有NSObject类的所有方法。这充分说明了作为一个可以继承大量财富的富二代是多么的令人羡慕。土豪，我们做朋友吧！

继承来的财富不一定让土豪满意，比如祖上虽然有庞大的产业，但主要收入来源是东莞的五星级酒店和娱乐场所。作为一个有良知尊重妇女的富二代，你可能会考虑弃暗投明，升级产业转而搞比较有节操的手机游戏。比如可以考虑设计一个疯狂美少女游戏，充分利用旗下的美女资源，把她们的形象卡通化。玩家把美少女架在树上，可以攻击那些猪一样的猿类。弹指一挥间，强撸灰飞烟灭。

因此，我们的MyClass类可能想要拥有自己的init方法：

```
@implementation MyClass

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

@end
```

从此之后，如果后面在代码中调用[[MyClass alloc]init]，就会触发这个新的init方法。

当然，在这个过程中我们还是需要调用NSObject版本的init方法，这样父类才可以初始化自己，这也是为神马在上面的自定义init方法的if条件语句中有[super init]。

想想也是，如果没有父辈的启动资金，你也没机会搞神马产业升级。

当然，我们无需在interface部分单独声明init方法：

```
@interface MyClass : NSObject
- (id)init; // not needed
@end
```

当然，如果你这么做也没有坏处，不过会让代码显得多余，因为在NSObject类的interface中已经声明了init方法。既然MyClass类继承自NSObject类，那么它自然就知道自己拥有一个init方法。

当然，如果我们使用其它名称创建了一个初始化方法，那么就需要在interface部分添加一个方法声明，否则其它类就无法使用它。

```
@interface MyClass: NSObject

-(id)initWithText:(NSString *)text;

@end
```

方法可以用于父类和其子类之间的通讯，这样子类就可以在特定的情况下执行特定的行为。这也是诸如viewDidLoad和viewWillAppear:方法的目的。这些方法在UIViewController中声明和实现，而在我们自己创建的自定义视图控制器子类中则可以override（覆盖/重写）。

例如，当用户界面将要呈现时，UIViewController类会调用[self viewWillAppear:YES]。通常情况下这个方法会触发来自父类的UIViewController的viewWillAppear:方法，不过我们也可以提供自定义子类的同名方法，这样就会在实际运行时触发这个新的方法。通过overriding viewWillAppear:方法，我们就可以在调用父类方法处理这个事件。

```
@implementation MyViewController

-(void)viewWillAppear:(BOOL)animated{

//don’t forget to call this!

[super viewWillAppear:animated];

//添加自己的代码


}
```

通过类似上面的方式，我们可以充分利用父类的威力。一个设计优雅的子类可以提供这样的”hooks(关联）来对特定事件作出响应。当然，别忘了调用父类的同名方法。如果你忽略了这一点，那么父类就无法得到自己的消息通知，从而会发生一些可怕的事情~

那么，一个子类是否需要使用来自父类的所有方法？答案当然是-否定的。以UIViewController类为例，在它的interface里面有多个方法，不过很多在实现中都隐藏了。即便是子类，也只能访问其父类的公开可见的方法。理论上来说，你可以调用这些隐藏的方法（假如你知道这些方法名称的话）。但是即便如此，你的应用也很可能会被App Store的审核人员拒之门外。

当然，你可能在学习其它编程语言（JAVA,C#)的时候听到过overloading（重载）的概念。overloading和overriding完全是两回事。在类似Java或C#之类的语言中，所谓的方法重载是在类中创建多个方法，它们的方法名称相同，但参数的个数/类型不同。调用方法时根据传递的参数个数和类型来决定使用哪种方法，也就是所谓的多态。

在Objective-C中相对比较少用到overloading这个概念，不过如果你实在对此感兴趣，可以参考这里：
http://stackoverflow.com/questions/11374032/objective-c-method-overriding-overloading-confusion

下一个概念— Protected instance variables（受保护实例变量）

子类可以从其父类继承所有的属性和实例变量。在子类中，我们可以使用父类的所有属性（至少是interface中所声明的属性变量），但如果要直接访问父类的实例变量则有点难度。

我们之前已经提到过，一个对象的interface中的几乎所有东西都可以被其它类的对象访问，而implementation实现部分的所有细节则是对外不开放的。不过如果这些其它对象是子类的对象呢？好吧，这就出现了一个新的概念，protected（受保护）。在Objective-C中，苹果提供了四种实例变量的范围类型，分别是@private, @protected, @public和@package。

其中@private这种范围类型的实例变量最严格，只能被声明它的类访问，即便是继承自它的子类也不行。

@public这种实例变量最宽松，任何类（当然也包括子类）都可以访问。

@protected这种实例变量居于二者之间，可以被声明它的类和子类访问。默认情况下（也就是不加这个前缀的情况下），在interface部分声明的实例变量都属于@protected这种范围类型。

当然还有一个不常用的@package范围类型，它和C语言中的private_extern比较类似，任何在实现类image镜像之外的代码如果要使用它都会引发link error。
通常情况下这个类型最常用于框架类(framework)的实例变量。

在声明实例变量的时候，我们可以通过显示的使用关键字@protected让implementation部分声明的实例变量可以被子类访问。

```
@implementation IceCream{

@protected
        float _percentageSugar;

}

@end
```

此时IceCream类的子类就可以访问_percentageSugar这个实例变量了。

```
@interface FrozenYoghurt: IceCream
@end

@implementation FrozenYoghurt
-(void)eatIt{

NSLog(@“Sugar: %f”,_percentageSugar);

}

@end
```

在其它编程语言中，使用@protected类型的实例变量是很普遍的事情。不过在Objective-C中基本上很少用到它。通常我们会把需要让其它类访问的变量声明成属性变量，而把实例变量隐藏。

不过了解上面的这些知识会方便你看懂苹果官方的API文档和定义。

下一个术语是啥?

Casts（转换？）

说实话习惯了看e文文档后，再看中文的文档真的很头大，很多术语翻译成天朝文字后都是让人云里雾里，感觉自己纯粹是个SB，其实一看e文就豁然开朗。

不多说废话了，来解释下这个名词吧。

现在三体人都知道NSObject是iOS应用中所有类的积累，因此我们可以把自己的对象当做NSObject的实例对象一样来引用。这句话看起来有点费劲，不过代码就一目了然了：

```
NSString *text = @“Hello, Trisomy ”;
NSObject *o = text;
NSLog(@“The text is: %@”,o);
```

上面的代码是可以跑起来的，因为NSString说到底也就是NSObject的扩展。我们甚至可以这样来做：

```
NSObject *o = @“Hello, Trisomy”;
NSLog(“The text is: %@”,o);
```

不过，下面的这句代码就不行了：

```
NSNumber *n = @“Hello, Trisomy”;
```

这是因为NSString不是NSNumber的父类，所以你不可能让NSNumber类型的变量像一个字符串一样来使用。

好吧，上面的代码多少让你有点害怕了，擦，竟然还可以这样来用父类?
不过，即便在Objective-C中可以这样做，有这个必要吗?
当然有，比如我们正在学习的这个系列教程就用到了这一点。

这个应用中用到了一个UITabBarController，其中有三个tab选项，而每一个都用一个视图控制器来呈现。第一个tab选项的视图控制器就是CurrentLocationViewController。而后面我们还将添加两个，分别是对应第二个tab的LocationsViewController和对应第三个tab的MapViewController。

当然，苹果设计UITabBarController这个控件的黄马甲们对这三个视图控制器肯定是一无所知了。唯一可以确定的就是，每个tab都会对应一个视图控制器，而每个视图控制器毫无例外都会继承自UIViewController。

因此UITabBarController无需直接和CurrentLocationViewController这个类来交流（设计的时候谁知道这个类会是神马样子的），而是只需要了解其父类UIViewController。作为一个UITabBarController，它只需要知道自己有三个UIViewController实例就好了，至于每个视图控制器的细节，就无所谓了。

同样，对于UINavigationController也是如此。对于导航控制器来说，只需要了解放到导航堆栈上的新视图控制器都是UIViewController的实例，不需要知道更多。

话虽这么说，有时候还是有点恼火的。比如当你向导航控制器请求获取它堆栈上的某个视图控制器时，它会返回到一个UIViewController对象的指针，即便这个对象的完整数据类型不是UIViewController（通常是它的子类）。如果我们想要像处理自己的视图控制器子类一样处理这个对象，就必须将它cast（转换）到一个合适的类型。

比如在上一个系列教程中我们在prepareForSegue中使用了类似下面的代码：

```
UINavigationController *navigationController = segue.destinationViewController;
ItemDetailViewController *controller =
(ItemDetailViewController*)navigationController.topViewController;

controller.delegate = self;
```

在上面的代码中，我们希望从导航堆栈中获取最上面的视图控制器，它是ItemDetailViewController的一个实例。然后再设置它的delegate属性。

遗憾的是，UINavigationController的topViewController属性返回的不是ItemDetailViewController类型的对象，而是简单的UIViewController对象，显然它没有所需的delegate属性。

如果我们这样写这行代码：

ItemDetailViewController *controller = navigationController.topViewController;

此时Xcode会给出警告：Incompatible pointer types

显然，我们不能直接把任何UIViewController对象赋给一个ItemDetailViewController类型的变量。即便所有的ItemDetailViewController都是UIViewController，但并非所有的UIViewController都是ItemDetailViewController。

比如你认识的一个美女苍老师喜欢穿黑丝，不代表所有穿黑丝的美女都是苍老师！不然，你就属于最近三个月的重点整顿对象了~

为了解决这个问题，我们需要将对象cast（转换）成合适的类型。这里我们已经知道该对象就是一个ItemDetailViewController了，因此只需要使用cast操作符()来通知编译器，我希望你像对待ItemDetailViewController一样对待这个对象。

此时，刚才的这行代码就变成：

ItemDetailViewController *controller = (ItemDetailViewController *)navigationController.topViewController;

好吧，现在我们就可以像处理一个ItemDetailViewController对象一样来处理这个从topViewController属性中获取的对象了。

不过，编译器并不会检查你所casting的对象是否真的就是这种类型的！！！
因此，如果你自己一时糊涂，应用就会在某个时刻突然崩溃。

cast（转换）并不是像吉安娜的变羊术一样把一个类型的对象转变成另一个类型。我们无法将一个NSNumber对象转换成NSString。通常情况下，我们会用cast来将某个父类对象cast成一个更具体的子类对象，而且这两种类型必须是兼容的。

好了，今天的学习到此结束，不过瘾的话你可以等凑够几章一起看，或者直接买原作者的e文版（强烈推荐！）

又到了发放福利的时间，老是发美女照片真的很没创意，该怎么办呢？这一期先凑合着吧。