### 点表示法

应“仅”用于获取和改变属性。括号表示法用于所有其它实例。

例如：

恰当用法：

view.backgroundColor = [UIColor orangeColor];

[UIApplication sharedApplication].delegate;

不当用法：

[view setBackgroundColor:[UIColor orangeColor]]; UIApplication.sharedApplication.delegate;

（对象的属性统一通过 .来访问即 object.property object.setProperty，函数通过[object method]的方式）

### 空格

（1）行缩进使用4个空格。禁止使用Tab键来缩进。请在Xcode偏好设置中进行设置。(主要意思是禁止使用制表符来缩进，在Xcode中配置好后，可以用Tab键缩进，它会将制表符用空格来填充)

（2）方法大括号和其它大括号（比如if/else/switch/while等等）应在语句的同一行开始，而在新的一行关闭。

例如：

if (user.isHappy) {

//Do something

}

else {

//Do something else

}

（

建议统一使用

if (user.isHappy) {

//Do something

} else {

//Do something else

}

）

（3）为保证视觉上的整洁和代码组织，在方法之间应提供且仅提供一行空白。方法中的空白应用于区分功能，但空白行最好用于区分两个不同方法。

### 条件语句

为避免错误，条件语句体必须使用大括号，即便语句体中的语句可以不必使用大括号（比如只有一行语句）。常见的错误包括在不使用大括号的情况下添加第二行语句，以为它属于if语句的一部分。此外，更可怕的事情是，如果条件语句中的代码行被注释，则本不术语条件语句的下一行代码将变成条件语句的一部分。此外，这种编码风格和所有其它条件语句均保持一致。

例如：

恰当用法：=====

if (!error) {

return success;

}

不当用法：

if (!error)

return success;

不当用法2：

if (!error) return success;

### 方法

在方法声明中，在(-/+)符号之后应加上一个空格。此外，在方法段之间应添加一个空格。

例如：

- (void)setExampleText:(NSString \*)text image:(UIImage \*)image;

在方法声明中，在(-/+)符号之后应加上一个空格。此外，在方法段之间应另起一行，且分号要对齐，参数名和方法的参数名保持一致.(不强制要求一定要另起一行，但当参数比较多的时候，每个字段一行比较易读)

例如：

- (void)setExampleText:(NSString \*)text

image:(UIImage \*)image;

调用者方式：

[object setExampleText:text

image:image];

or

[object setExampleText:text image:image];

（方法函数编写规范补充：

1 最理想的参数数量是零（零参数函数），其次是一（单参数函数），再次是二（双参数函数），应尽量避免三（三参数函数）。有足够特殊的理由才能用三个以上参数（多参数函数）---所以无论如何也不要这么做。主要目的是为了代码清晰。

2 通过返回值从函数中输出，不太期望信息通过参数输出，所以，输出参数往往让人苦思之后才恍然大悟。

3 如果函数看来需要两个，三个或三个以上参数，就说明其中一些参数应该封装为类了。

4 暴露给外面的函数和属性写在h文件中，内部使用的写在m文件的类扩展里。

### 变量

变量的命名应尽可能具有自解释性。除了在for()循环语句中(目前项目是在for循环里使用NSInteger index)，应避免使用单个字母变量名称。

除非是常量，星号应紧贴变量名称表示指向变量的指针，比如：

正确用法：

NSString \*text;

不当用法：

NSString\* text;

NSString \* text;

应尽可能使用属性定义替代单一的实例变量。避免在初始化方法,dealloc方法和自定义的setter和getter方法中直接读取实例变量参数（init,initWithCoder:，等等）。

（

这里是说避免在init和dealloc中使用self.property的方式操作属性

苹果官方文档有说**Don’t Use Accessor Methods in Initializer Methods and dealloc**

https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/MemoryMgmt/Articles/mmPractical.html#//apple\_ref/doc/uid/TP40004447-SW6

StackOverFlow上的解释

http://stackoverflow.com/questions/5932677/initializing-a-property-dot-notation/5932733#5932733

几种属性使用方式的讨论

http://stackoverflow.com/questions/7842641/best-way-to-set-a-retained-property-to-a-newly-created-object/7842811#7842811

解决方案

self.myProperty =[[[MyClass alloc] init] autorelease];这种属性操作方式比较优雅，也不容易出错，尽量采用这种方式，包括在init和dealloc。

从本人实验的结果来看，虽然apple官方说避免用这种方式，但是除了在StackOverFlow的解释中因为在子类中重写了父类的property方法，这种情况下会发生leaks，其他情况下是安全的.

所以考虑到本项目中子类重写父类property方法这种情况基本很少，而采用self.property所带来的好处，建议统一采用这种self.property的方式。

）

例如：

恰当用法：

@interface NYTSection: NSObject

@property (nonatomic) NSString \*headline;

@end

不当用法：

@interface NYTSection : NSObject {

NSString \*headline;

}

@property补充

括号内的属性用逗号+一个空格隔开

现项目中的属性声明，只需要@property 这一句就可以了，公开的放在h，私有的放在m的类扩展中，有copy和retain的记得在dealloc中调用self.property = nil，arc下则不需要;

恰当用法：

@property (nonatomic, copy) NSString \*headline;

不当用法：

@property(nonatomic, copy) NSString \*headline;

@property (nonatomic, copy)NSString \*headline;

@property (nonatomic,copy) NSString \*headline;

### 命名规范

苹果的命名规范应尽可能符合内存管理法则(NARC）memory management rules

在Objective-C中鼓励使用长的描述性的方法和变量名称。

(长的描述性只是手段，目的是者变量要让读这代码的人容易懂，帮助理解代码，当然不是说变量越长越好，保持清晰的基础上取个精炼简洁的好名字。

基本规则：

1函数内部的局部变量可以采用较短的命名，当然也不能牺牲描述性，由于上下文一目了然，

较长的变量名反而会显得冗余。

2 变量的作用域越大，变量名包含的信息应该越多。

)

btnSettings

lblName;

labelName;

imgName;

imgViewName

例如：

恰当用法：UIButton \*settingButton;

不当用法：

UIButton \*setBut;

### **关于 pragma**

在 <http://raptureinvenice.com/pragmas-arent-just-for-marks/> 有很好的关于 pragma 的讨论了，在这边我们再做部分说明。

大多数 iOS 开发者平时并没有和很多编译器选项打交道。一些选项是对控制严格检查（或者不检查）你的代码或者错误的。有时候，你想要用 pragma 直接产生一个异常，临时打断编译器的行为。

当你使用ARC的时候，编译器帮你插入了内存管理相关的调用。但是这样可能产生一些烦人的事情。比如你使用NSSelectorFromString 来动态地产生一个 selector 调用的时候，ARC不知道这个方法是哪个并且不知道应该用那种内存管理方法，你会被提示 performSelector may cause a leak because its selector is unknown（执行 selector 可能导致泄漏，因为这个 selector 是未知的）.

如果你知道你的代码不会导致内存泄露，你可以通过加入这些代码忽略这些警告

#pragma clang diagnostic push

#pragma clang diagnostic ignored "-Warc-performSelector-leaks"

[myObj performSelector:mySelector withObject:name];

#pragma clang diagnostic pop

注意我们是如何在相关代码上下文中用 pragma 停用 -Warc-performSelector-leaks 检查的。这确保我们没有全局禁用。如果全局禁用，可能会导致错误。

### **忽略没用使用变量的编译警告**

告诉你申明的变量它将不会被使用，这种做法很有用。大多数情况下，你希望移除这些引用来（稍微地）提高性能，但是有时候你希望保留它们。为什么？或许它们以后有用，或者有些特性只是暂时移除。无论如何，一个消除这些警告的好方法是用相关语句进行注解，使用 #pragma unused():

- (NSInteger)giveMeFive

{

NSString \*foo;

#pragma unused (foo)

return 5;

}

现在你的代码不用任何编译警告了。注意你的 pragma 需要标记到问题代码之下。

## **明确编译器警告和错误**

编译器是一个机器人，它会标记你代码中被 Clang 规则定义为错误的地方。但是，你总是比 Clang 更聪明。通常，你会发现一些讨厌的代码会导致这个问题，但是暂时却解决不了。你可以这样明确一个错误：

- (NSInteger)divide:(NSInteger)dividend by:(NSInteger)divisor

{

#error Whoa, buddy, you need to check for zero here!

return (dividend / divisor);

}

类似的，你可以这样标明一个警告

- (float)divide:(float)dividend by:(float)divisor

{

#warning Dude, don't compare floating point numbers like this!

if (divisor != 0.0) {

return (dividend / divisor);

}

else {

return NAN;

}

}

### 注释

在需要注释的地方，应使用注释来解释某一块特定的代码的功能。所有的代码注释必须是最新的，要吗就删掉。（对的，没用的注释和代码全部删掉，这些都在svn上保留着，放心删）

应尽量使用行注释，而避免使用块注释。之所以这样是因为代码自身需要是自文档化的，因此只需要零散添加一些行注释。当然，对于用于生成文档的注释，该原则并不适用。

（关于注释，建议统一采用VVDocument插件来生成doxygen注释，可以通过doxygen和graphviz生成项目的文档和代码的类图，非常方便

注释需要注意的地方

1 类的h头文件中，需要加入注释说明这个类的主要用途，写的时候请思考如果是一个新人，

他如何通过注释快速了解这个类以及相关的用途。

2 不强制要求每个属性和函数都要注释，能自说明的变量名和函数名就是最好的注释。

3 较复杂的函数和逻辑，可以写一下处理的思路和主要代码逻辑

4 代码注释必须是最新的，如果代码的意图有变，则修改注释或者干脆删除注释。切勿注释

对不上代码意图。

）

### Literals字面量

在创建NSString,NSDictionary,NSArray和NSNumber等对象的immutable实例时，应使用字面量。需要注意的是，不应将nil传递给NSArray和NSDictionary字面量，否则会引起程序崩溃。

例如：

恰当用法：

NSArray \*names = @[@"Brian", @"Matt", @"Chris", @"Alex", @"Steve", @"Paul"];

NSDictionary \*productManagers = @{@"iPhone" : @"Kate", @"iPad" : @"Kamal", @"Mobile Web" : @"Bill"};

NSNumber \*shouldUseLiterals = @YES;

NSNumber \*buildingZIPCode = @10018;

不当用法：

NSArray \*names = [NSArray arrayWithObjects:@"Brian", @"Matt", @"Chris", @"Alex", @"Steve", @"Paul", nil];

NSDictionary \*productManagers = [NSDictionary dictionaryWithObjectsAndKeys: @"Kate", @"iPhone", @"Kamal", @"iPad", @"Bill", @"Mobile Web", nil];

NSNumber \*shouldUseLiterals = [NSNumber numberWithBool:YES];

NSNumber \*ZIPCode = [NSNumber numberWithInteger:10018];

(Modern Objective-C 的语法糖，建议统一采用恰当用法)

**避免使用 new**

不要调用NSObject类的类方法new，也不要在子类重写。相反，你应该使用alloc和init方法来创建并初始化一个对象。

现代的Ojbective-C代码通过调用alloc和init方法来创建并且retain一个对象。由于类方法new很少被使用，这使得有关内存分配的代码审查更困难。

### CGRect函数

当需要获取一个CGRect矩形的x,y,width,height属性时，应使用CGGeometry函数，而非直接访问结构体成员。

例如：

恰当用法：

CGRect frame = self.view.frame;

CGFloat x = CGRectGetMinX(frame);

CGFloat y = CGRectGetMinY(frame);

CGFloat width = CGRectGetWidth(frame);

CGFloat height = CGRectGetHeight(frame);

不当用法：

CGRect frame = self.view.frame;

CGFloat x = frame.origin.x;

CGFloat y = frame.origin.y;

CGFloat width = frame.size.width;

CGFloat height = frame.size.height;

### 常量

相对字符串字面量或数字，我们更推荐适用常量。应使用static方式声明常量，而非使用#define的方式来定义宏。

例如：

恰当用法：

static NSString \* const NYTAboutViewControllerCompanyName = @"The New York Times Company";

static const CGFloat NYTImageThumbnailHeight = 50.0;

不当用法：

#define CompanyName @"The New York Times Company"

#define thumbnailHeight 2

(宏有适用的地方，不要滥用就是了，一些常量还是建议用static或者extern const的方式参考)

### 枚举类型

在使用enum的时候，推荐适用最新的fixed underlying type(WWDC 2012 session 405- Modern Objective-C)规范，因为它具备更强的类型检查和代码完成功能。

例如：

typedef NS\_ENUM(NSInteger, NYTAdRequestState){

NYTAdRequestStateInactive,

NYTAdRequestStateLoading

};

### 私有属性

私有属性应在类实现文件的类扩展（匿名分类）中进行声明。应避免使用命名分类（比如NYTPrivate或private）。

例如：

@interface NYTAdvertisement ()

@property (nonatomic, strong) GADBannerView \*googleAdView;

@property (nonatomic, strong) ADBannerView \*iAdView;

@property (nonatomic, strong) UIWebView \*adXWebView;

@end

### 图片名称

在命名图片名称的时候，应保持一致性，从而让开发团队和成员可以明白其含义。图片名称的第一个单词应描述其用途，并使用camel-case风格，然后是不带前缀的所属类名称或属性，最后是色彩、位置和状态。

例如：

RefreshBarButtonItem / RefreshBarButtonItem@2x and RefreshBarButtonItemSelected / RefreshBarButtonItemSelected@2x ArticleNavigationBarWhite / ArticleNavigationBarWhite@2x and ArticleNavigationBarBlackSelected / ArticleNavigationBarBlackSelected@2x.

### 布尔变量

因为nil将被解析为NO，因此没有必要在条件语句中进行比较。永远不要将任何东西和YES进行直接比较，因为YES被定义为1，而一个BOOL变量可以有8个字节。

例如：

恰当用法：

if (someObject) {

}

if (someEnum == Normal) {

}

不当用法：

if (someObject == nil) { }

if (nil == someObject) { }

if (YES == someObject) { }

if (someObject == NULL) {}

以下是BOOL变量的使用：

恰当用法：

if (isAwesome)

if (![someObject boolValue])

不当用法：

if ([someObject boolValue] == NO)

if (isAwesome == YES) // Never do this.

如果一个BOOL属性使用形容词来表达，属性将忽略’is’前缀，但会强调惯用名称。

例如：

@property (assign, getter=isEditable) BOOL editable;

### 单例

在创建单例对象的共享实例时，应使用线程安全模式。

例如：

- (instancetype)sharedInstance {

static id sharedInstance = nil;

static dispatch\_once\_t onceToken;

dispatch\_once(&onceToken, ^{

sharedInstance = [[self alloc] init];

});

return sharedInstance;

}

Xcode项目

为避免文件混乱，实际的物理文件应和Xcode项目保持一直。在Xcode中所创建的任何group都应有文件系统中相对应的文件夹。不应仅根据文件类型来进行分组，还需要考虑到其作用。

在Xcode的target的Build Setting中，中尽量开启”Treat Warnings as Errors“，同时尽量开启其他的警告[additional warnings](http://boredzo.org/blog/archives/2009-11-07/warnings" \t "_blank)。如果需要忽略某个特定的警告，可以使用[Clang's pragma feature](http://clang.llvm.org/docs/UsersManual.html" \l "controlling-diagnostics-via-pragmas" \t "_blank)

（在项目的维护性上比较重要，Treat Warnings as Errors，这个目前可能不太现实，但是项目本身的warnings要处理掉，第三方的由于目前没有分Library，所以会一直存在，直到第三方代码的维护者解决了这个问题）

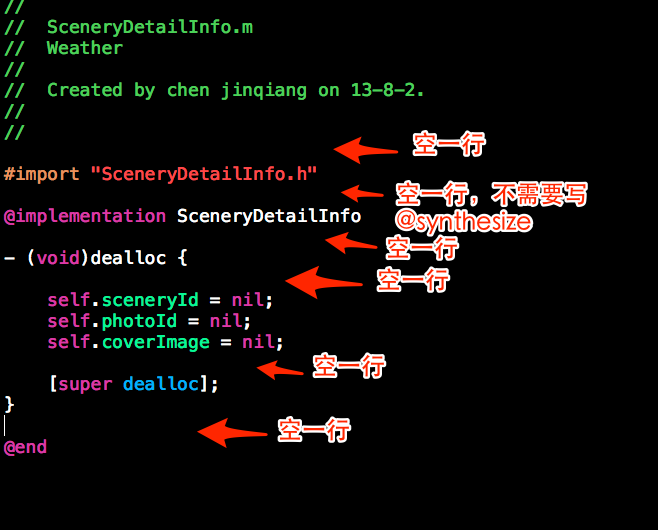
类的函数排版

1. Model类.h文件代码书写规范建议



1. 312

2、21.'wen'j﷽﷽﷽﷽﷽﷽﷽﷽明未CGRectZero];Model类.m文件代码书写规范建议：

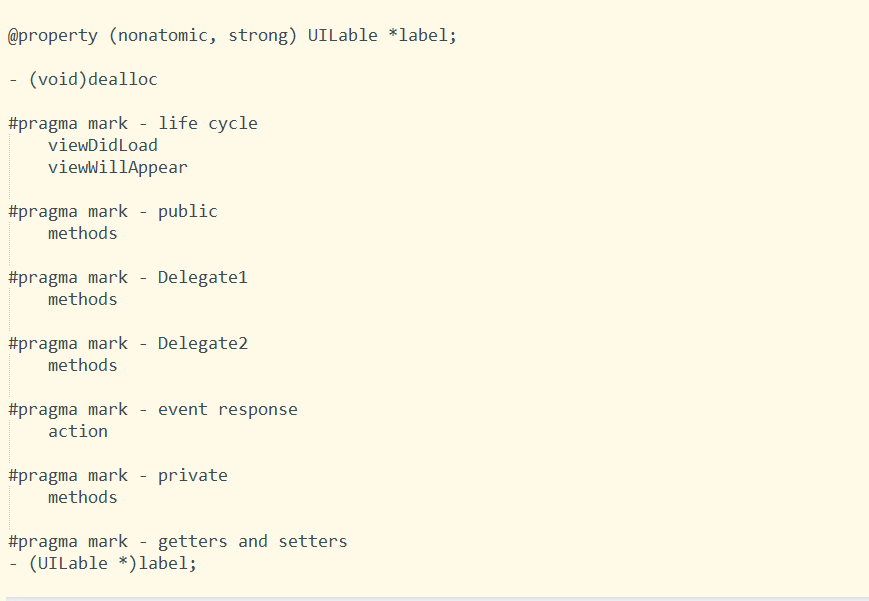


1. 1.'wen'j﷽﷽﷽﷽﷽﷽﷽﷽明未CGRectZero];Concsdfsd Controller 类 .h文件代码书写规范建议：



1. Controller类.m文件代码书写规范建议：



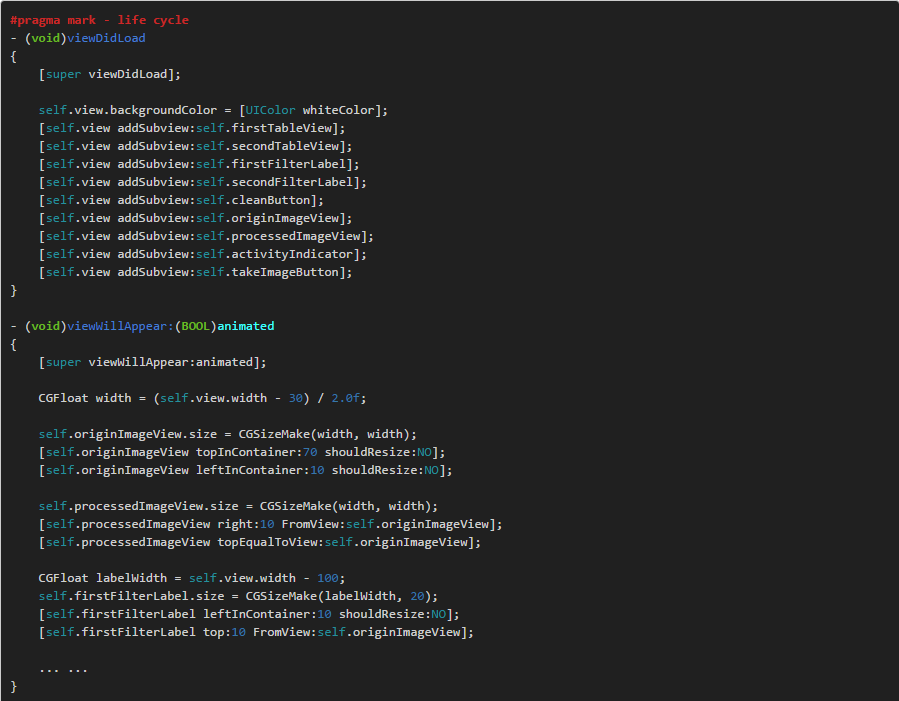


要点：

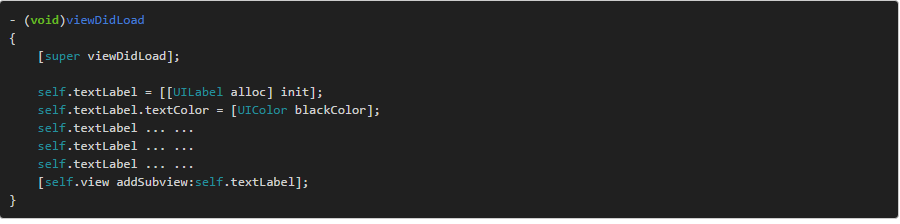
所有的属性都使用getter和setter

不要在viewDidLoad里面初始化你的view然后再add，这样代码就很难看。在viewDidload里面只做addSubview的事情，然后在viewWillAppear里面做布局的事情，最后在viewDidAppear里面做Notification的监听之类的事情。至于属性的初始化，则交给getter去做。

比如这样：



这样即便在属性非常多的情况下，还是能够保持代码整齐，view的初始化都交给getter去做了。总之就是尽量不要出现以下的情况：



这种做法就不够干净，都扔到getter里面去就好了。

getter和setter全部都放在最后

因为一个ViewController很有可能会有非常多的view，就像上面给出的代码样例一样，如果getter和setter写在前面，就会把主要逻辑扯到后面去，其他人看的时候就要先划过一长串getter和setter，这样不太好。然后要求业务工程师写代码的时候按照顺序来分配代码块的位置，先是life cycle，然后是Delegate方法实现，然后是event response，然后才是getters and setters。这样后来者阅读代码时就能省力很多。

1. 1.'wen'j﷽﷽﷽﷽﷽﷽﷽﷽明未CGRectZero];view.h文件代码书写规范建议：



1. 1.'wen'j﷽﷽﷽﷽﷽﷽﷽﷽明未CGRectZero];view类.m文件代码书写规范建议：





