# Objective-C编码规范

## 应该使用美式英语来命名

应该:

1. UIColor \*myColor = [UIColor whiteColor];

不应该

1. UIColor \*myColour = [UIColor whiteColor];

## 使用函数和protocol/delegate视线中使用#paragma mark- 来分类方法：

示例：

1. #pragma mark - Lifecycle
2. - (instancetype)init {}
3. - (**void**)dealloc {}
4. - (**void**)viewDidLoad {}
5. - (**void**)viewWillAppear:(**BOOL**)animated {}
6. - (**void**)didReceiveMemoryWarning {}
7. #pragma mark - Custom Accessors
8. - (**void**)setCustomProperty:(id)value {}
9. - (id)customProperty {}
10. #pragma mark - IBActions
11. - (IBAction)submitData:(id)sender {}
12. #pragma mark - Public
13. - (**void**)publicMethod {}
14. #pragma mark - Private
15. - (**void**)privateMethod {}
16. #pragma mark - Protocol conformance
17. #pragma mark - UITextFieldDelegate
18. #pragma mark - UITableViewDataSource
19. #pragma mark - UITableViewDelegate
20. #pragma mark - NSCopying
21. - (id)copyWithZone:(NSZone \*)zone {}
22. #pragma mark - NSObject
23. - (NSString \*)description {}

## 空格

缩进使用4个空格，可在Xcode偏好设置来设置（Xcode默认4个空格）

方法大括号和if/else/switch/while总在与语句同一行打开，在新行关闭

应该：

1. **if** (user.isHappy) {
2. //Do something
3. } **else** {
4. //Do something else
5. }

不应该：

1. **if** (user.isHappy)
2. {
3. //Do something
4. }
5. **else** {
6. //Do something else
7. }

## 注释

当需要注释时，注释应该能够解释代码的行为，注释必须保持最新。

一般都避免使用块注释，因为代码尽可能做到自解释，只有当断断续续或几行代码时才需要注释。例外：这不应用在生成文档的注释。

## 命名

Apple命名长的，描述性的方法和变量命名是好的。

应该：

1. UIButton \*settingsButton;

不应该：

1. UIButton \*setBut;

在类名和常量命名前加三个字符前缀，但在Core Data的实体名中应该忽略。

属性应该使用驼峰式，但首单词的首字母小写。对属性应该使用auto-synthesis,而不是手动编写@synthesize语句，除非你有好的理由：

应该：

@property (strong, nonatomic) NSString \*descriptiveVariableName;

不应该：

id varnm;

## 下划线

当使用属性时，实例变量应该使用self.来访问和改变。这意味着所有的属性访问前面需加self.

但有一个特例：在初始化方法里，实例变量(例如，\_variableName)应该直接被使用来避免getters/setters潜在的副作用。

局部变量不应该包含下划线。

## 方法

在方法签名中，应该在(-/+)符号之后有一个空格，方法各个段之间应该有一个控件，参数之前应该包含描述性的关键字来描述参数。

应该：

1. - (**void**)setExampleText:(NSString \*)text image:(UIImage \*)image;
2. - (**void**)sendAction:(SEL)aSelector to:(id)anObject forAllCells:(**BOOL**)flag;
3. - (id)viewWithTag:(NSInteger)tag;
4. - (instancetype)initWithWidth:(CGFloat)width height:(CGFloat)height;

不应该：

1. -(**void**)setT:(NSString \*)text i:(UIImage \*)image;
2. - (**void**)sendAction:(SEL)aSelector :(id)anObject :(**BOOL**)flag;
3. - (id)taggedView:(NSInteger)tag;
4. - (instancetype)initWithWidth:(CGFloat)width andHeight:(CGFloat)height;
5. - (instancetype)initWith:(**int**)width and:(**int**)height;  // Never do this.

## 变量

变量尽量以描述性的方式来命名。单个字符的变量命名应该尽量避免，除了在for()循环。

星号表示变量是指针。例如， NSString \*text 既不是 NSString\* text 也不是 NSString \* text，除了一些特殊情况下常量。

私有变量应该尽可能替代实例变量的使用。尽管实例变量是一种有效的方式，但更偏向于使用属性来保持代码一致性。

通过使用\_variable来直接访问实例变量应该尽量避免，除了在初始化方法(init,initWith),dealloc方法和自定义getters和setters方法。

应该：

1. @interface RWTTutorial : NSObject
2. @property (strong, nonatomic) NSString \*tutorialName;
3. @end

不应该：

1. @interface RWTTutorial : NSObject {
2. NSString \*tutorialName;
3. }

## 属性特性

所有的属性特性应该显式的列出来，有助于新手阅读代码。属性特性的顺序应该是storage、atomicity，与在Interface Builder连接UI元素时自动生成代码一致。

应该：

1. @property (weak, nonatomic) IBOutlet UIView \*containerView;
2. @property (strong, nonatomic) NSString \*tutorialName;

不应该：

1. @property (nonatomic, weak) IBOutlet UIView \*containerView;
2. @property (nonatomic) NSString \*tutorialName;

NSString应该使用copy而不是strong的属性特性。

为什么？即使你声明一个NSString的属性，有人可能传入一个NSMutableString的实例，然后你再没有注意的情况下修改它：

应该：

1. @property (copy, nonatomic) NSString \*tutorialName;

不应该：

@property (strong, nonatomic) NSString \*tutorialName;

## 点符号语法

点语法是一种很方便的封装访问方法调用的方式。当你使用点语法时，实际上是通过使用getter和setter方法，属性仍然被访问和修改。

应该：

1. NSInteger arrayCount = [self.array count];
2. view.backgroundColor = [UIColor orangeColor];
3. [UIApplication sharedApplication].delegate;

不应该：

1. NSInteger arrayCount = self.array.count;
2. [view setBackgroundColor:[UIColor orangeColor]];
3. UIApplication.sharedApplication.delegate;

## 字面值

NSString、NSDictionary、NSArray和NSNumber的字面值应该在创建这些类的不可变实例时使用。注意nil值不能传入NSArry和NSDictionary字面值，因为这样会导致crash.

应该：

1. NSArray \*names = @[@"Brian", @"Matt", @"Chris", @"Alex", @"Steve", @"Paul"];
2. NSDictionary \*productManagers = @{@"iPhone": @"Kate", @"iPad": @"Kamal", @"Mobile Web": @"Bill"};
3. NSNumber \*shouldUseLiterals = @YES;
4. NSNumber \*buildingStreetNumber = @10018;

不应该：

1. NSArray \*names = [NSArray arrayWithObjects:@"Brian", @"Matt", @"Chris", @"Alex", @"Steve", @"Paul", nil];
2. NSDictionary \*productManagers = [NSDictionary dictionaryWithObjectsAndKeys: @"Kate", @"iPhone", @"Kamal", @"iPad", @"Bill", @"Mobile Web", nil];
3. NSNumber \*shouldUseLiterals = [NSNumber numberWithBool:YES];
4. NSNumber \*buildingStreetNumber = [NSNumber numberWithInteger:10018];

## 常量

常量时容易重复被使用和无需通过查找和替代就能快速修改值。常量应该通过static声明，而不是#define,除非显式使用宏。

应该：

1. **static** NSString \* **const** RWTAboutViewControllerCompanyName = @"RayWenderlich.com";
2. **static** CGFloat **const** RWTImageThumbnailHeight = 50.0;

不应该

1. #define CompanyName @"RayWenderlich.com"
2. #define thumbnailHeight 2

## 枚举类型

当使用enum时，推荐使用新的固定基本类型，因为它有更强的类型检查和代码补全。现在SDK有一个宏NS\_ENUM()来帮助和鼓励你使用固定的基本类型。

例如：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/3)

1. **typedef** NS\_ENUM(NSInteger, RWTLeftMenuTopItemType) {
2. RWTLeftMenuTopItemMain,
3. RWTLeftMenuTopItemShows,
4. RWTLeftMenuTopItemSchedule
5. };

你也可以显式地赋值(展示旧的k-style常量定义)：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/3)

1. **typedef** NS\_ENUM(NSInteger, RWTGlobalConstants) {
2. RWTPinSizeMin = 1,
3. RWTPinSizeMax = 5,
4. RWTPinCountMin = 100,
5. RWTPinCountMax = 500,
6. };

旧的k-style常量定义应该避免除非编写Core Foundation C的代码。

不应该：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/3)

1. **enum** GlobalConstants {
2. kMaxPinSize = 5,
3. kMaxPinCount = 500,
4. };

## Case语句

大括号在case语句中不是必须的，除非编译器有强制要求。当一个case语句包含多行代码时，大括号应该加上。

1. **switch** (condition) {
2. **case** 1:
3. // ...
4. **break**;
5. **case** 2: {
6. // ...
7. // Multi-line example using braces
8. **break**;
9. }
10. **case** 3:
11. // ...
12. **break**;
13. **default**:
14. // ...
15. **break**;
16. }

当相同代码被多个case使用时，一个fall-through应该被使用。

1. **switch** (condition) {
2. **case** 1:
3. // \*\* fall-through! \*\*
4. **case** 2:
5. // code executed for values 1 and 2
6. **break**;
7. **default**:
8. // ...
9. **break**;
10. }

## 私有属性

私有属性应该在类的实现文件中的类拓展中声明。

1. @interface RWTDetailViewController ()
2. @property (strong, nonatomic) GADBannerView \*googleAdView;
3. @property (strong, nonatomic) ADBannerView \*iAdView;
4. @property (strong, nonatomic) UIWebView \*adXWebView;
5. @end

## 布尔值

Objective-C使用YES和NO。因为true和false只在Core Foundation, C或C++代码中使用。因为nil被解释成NO，所以没有必要再条件语句中比较。不要拿某样东西直接与YES比较，因为YES被定义为1和一个BOOL能被设置为8位。

这是为了在不同文件保持一致性和在视觉上更加简洁而考虑。

应该：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/4)

1. **if** (someObject) {}
2. **if** (![anotherObject boolValue]) {}

不应该：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/4)

1. **if** (someObject == nil) {}
2. **if** ([anotherObject boolValue] == NO) {}
3. **if** (isAwesome == YES) {} // Never do this.
4. **if** (isAwesome == **true**) {} // Never do this.

如果BOOL属性的名字是一个形容词，属性就能忽略"is"前缀，但要指定get访问器的惯用名称。例如：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/4)

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

## 条件语句

条件语句主体为了防止出错应该使用大括号包围，即使条件语句主体能够不用大括号编写(如，只用一行代码)。

应该：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/4)

1. **if** (!error) {
2. **return** success;
3. }

不应该：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/4)

1. **if** (!error)
2. **return** success;

或

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/4)

1. **if** (!error) **return** success;

## 三元操作符

当需要提高代码的清晰性和简洁性时，三元操作符?:才会使用。单个条件求值常常需要它。多个条件求值时，如果使用if语句或重构成实例变量时，代码会更加易读。一般来说，最好使用三元操作符是在根据条件来赋值的情况下。

Non-boolean的变量与某东西比较，加上括号()会提高可读性。如果被比较的变量是boolean类型，那么就不需要括号。

应该：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/4)

1. NSInteger value = 5;
2. result = (value != 0) ? x : y;
3. **BOOL** isHorizontal = YES;
4. result = isHorizontal ? x : y;

不应该：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/4)

1. result = a > b ? x = c > d ? c : d : y;

## init方法

Init方法应该遵循Apple生成代码模板的命名规则，返回类型应该使用instancetype而不是id。

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/4)

1. - (instancetype)init {
2. self = [super init];
3. **if** (self) {
4. // ...
5. }
6. **return** self;
7. }

## 类构造函数

当类构造方法被使用时，它应该返回类型是instancetype而不是id。这样确保编译器正确地推断结果类型。

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/4)

1. @interface Airplane
2. + (instancetype)airplaneWithType:(RWTAirplaneType)type;
3. @end

## 黄金路径

当使用条件语句编码时，左手边的代码应该是"golden" 或 "happy"路径。也就是不要嵌套if语句，多个返回语句也是OK。

应该：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/5)

1. - (**void**)someMethod {
2. **if** (![someOther boolValue]) {
3. **return**;
4. }
5. //Do something important
6. }

不应该：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/5)

1. - (**void**)someMethod {
2. **if** ([someOther boolValue]) {
3. //Do something important
4. }
5. }

## 错误处理

当方法通过引用来返回一个错误参数，判断返回值而不是错误变量。

应该：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/5)

1. NSError \*error;
2. **if** (![self trySomethingWithError:&error]) {
3. // Handle Error
4. }

不应该：

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/5)

1. NSError \*error;
2. [self trySomethingWithError:&error];
3. **if** (error) {
4. // Handle Error
5. }

在成功的情况下，有些Apple的APIs记录垃圾值(garbage values)到错误参数(如果non-NULL)，那么判断错误值会导致false负值和crash。

## 单例模式

单例对象应该使用线程安全模式来创建共享实例。

**[cpp]** [view plaincopy](http://www.csdn.net/article/2015-06-01/2824818-objective-c-style-guide/5)

1. + (instancetype)sharedInstance {
2. **static** id sharedInstance = nil;
3. **static** dispatch\_once\_t onceToken;
4. dispatch\_once(&onceToken, ^{
5. sharedInstance = [[self alloc] init];
6. });
7. **return** sharedInstance;
8. }

这会防止[possible and sometimes prolific crashes](http://cocoasamurai.blogspot.com/2011/04/singletons-your-doing-them-wrong.html)。

## XCode工程

物理文件应该与Xcode工程文件保持同步来避免文件扩张。任何Xcode分组的创建应该在文件系统的文件体现。代码不仅是根据类型来分组，而且还可以根据功能来分组，这样代码更加清晰。

尽可能在target的Build Settings打开"Treat Warnings as Errors，和启用以下[additional warnings](http://boredzo.org/blog/archives/2009-11-07/warnings)。如果你需要忽略特殊的警告，使用[Clang's pragma feature](http://clang.llvm.org/docs/UsersManual.html#controlling-diagnostics-via-pragmas)。