Objective-C语言代码风格指导

良好代码风格是一种艺术

作者：woodjobber

博客：woodjobber.github.io

简介

代码规范目的不是提供死板的规定，仅仅是建议，提供给开发者之间降低阅读代码的障碍。优秀的代码应该具备：**代码简洁简明，代码自我翻译等特点。我一直认为--良好代码规范优先于性能**。

编程规范

点语法

对于`getter`,`setter`属性尽量使用点（`.`）语法(除了在`init`和`dealloc`方法中，应该直接使用`实例变量`，至于原因稍后解释)。

Preferred:

```
view.backgroundColor = [UIColor redColor];
[UIApplication sharedApplication].delegate;
```

Not Preferred:

```
[view setBackgroundColor:[UIColor redColor]];
UIApplication.sharedApplication.delegate;
```

条件判断

条件语句体应该总是被花括号`{}`包裹，包括仅仅只有一句代码，且都不能省略，`if`与左圆括弧`(`之间有一个`空格`，右圆括弧`)`与左花括号`{`之间有一个`空格`。多条件判断的时候，可以先判断否定的情况。

Preferred:

```
if (!isVaild) {
    return NO;
}

if (!isVaild) {return NO;}
```

Not Preferred:

```
if (!isVaild)
return NO;
```

Or

```
if (!isVaild) return NO;
```

Preferred:

```
- (BOOL)isValidUser:(User *)user {
    if (!user.name) {return NO;}
    if (!user.password) {return NO;}
    if (!user.email) {return NO;}

    return YES;
}
```

Or

```
- (BOOL)isValidUser:(User *)user {
    BOOL isValidForName = user.name;
    BOOL isValidForPassword = user.password;
    BOOL isValidForEmail = user.email;
    BOOL isInValid = !isValidForName || !isValidForPassword || !isValidForEmail;
    if (isInValid) {
        return NO;
    }
    //其他条件
    return YES;
}
```

Not Preferred:

```
- (BOOL)isValidUser:(User *)user {
    BOOL isValid = NO;
    if (user.name)
    {
        if (user.password)
        {

            if (usr.email) {
                isValid = YES;
            }
        }

    }

    retrun isValid;
}
```

尤达表达式

最好不要使用`尤达表达式`。`尤达表达式`是指，`比较者`与`被比较者`之间的前后关系，`不要用``比较者`与`被比较者`相比较 `而是` 应该用`被比较者`与`比较者`相比较。简单说，就是`用` `变量`与 `参照物`相比较，而`不是``参照物`与`变量`相比较。

Preferred:

```
id value = ?;//? 代表某一个对象
if ([value isEqual:@12]) {
    //code
}
```

Not preferred:

```
if (@12 isEqual:value) {
    //code
}
```

三元运算符

当三元运算符的第二个参数的返回值与检查对象一致时，这样使用：

Preferred:

```
result = obj?:[self callOtherMethod];
```

Not Preferred:

```
result = obj? obj : [self callOtherMethod];
```

字面值

对于字面值我们应该使用`NSString`,`NSDictionary`,`NSArray`，`NSNumber`的简写形式来创建对象。

Preferred:

```
NSArray *cities = @[@"成都"，@"北京",@"上海"];
NSDictionary *productPrice = @{@"iPhone5":@"4300",@"iPhone6":@"4800"};
NSNumber *age = @23;
```

Not Preferred:

```
NSArray *cities = [NSArray arrayWithObjects:@"成都",@"北京",@"上海",nil];
NSDicitionary *productPrice = [NSDictionary dictionaryWithObjectsAndKeys:@"4300",@"iPhone5",@"4800",@"iPhone"];
```

方法

对于方法应该遵循苹果的规范，(`-/+`)应该与左圆括弧`(`有一个`空格`。

Preferred:

```
- (void)sendMessage;
```

Not Preferred:

```
-(void)sendMessage;
```

变量

Preferred:

```
@interface Person: NSObject

@property (nonatomic,copy) NSString *name;

@end
```

Not Preferred:

```
@interface Person: NSObject {

    NSString *name;//或NSString *_name;
}
```

init 和 dealloc

`dealloc`方法应该放在实现文件的顶部，直接放在`@sythesize` 和 `@dynamic`的后面，`init`方法应该放在`dealloc`方法的后面。注意`init`方法里面永远不应该调用`setter`、`getter`方法，你应该直接访问`实例变量`。至于什么原因？在于一个对象只有在`init`返回之后，才完成初始化状态，只有这样才能说明创建的对象已经处于稳定的状态。同样，在`dealloc`中，也不能这样做。简单的一句话总结：总是使用`self.propery`这种形式(出于对内存管理考虑)，`切记除了在` `init`和`dealloc`方法中，应该直接访问`实例变量`。原因：

If you're trying to be as defensive as possible, then you have to operate under the assumption that someone may have subclassed the class and overridden your public methods. These overrides may be assuming that when the method is invoked, all of the properties and ivars and everything are in a consistent and fully-initialized state.

So, if your initializer or dealloc method causes one of these overridden methods to be executed, then the logic of the method may do something incorrect. The best case scenario is that it works as intended. The worst case scenario is that you do something like dereference a NULL pointer or index beyond the bounds of an array or cause some data to become corrupted, etc.`

While you're inside an initializer or dealloc method, your object is in an inconsistent state. Thus, if you're trying to be as defensive as possible, you should avoid invoking methods that might be making assumptions about the state of the object.

请参考

- Ivars should be directly accessed in init'ers #6
- Don't Use Accessor Methods in Initializer Methods and dealloc

Pereferred:

```
@implement
@sythesize age;
@dynamic name;
- (void)dealloc {

}
- (id)init {
    if (self = [super init]) {

    }
    return self;
}

@end
```

Pereferred:

```
- (id)init {
    self = [super init];
    if (self) {
        _name = @"jumei";
        _age = @"4";
    }
    return self;
}
```

Not Pereferred:

```
- (id)init {
    if (self = [super init]) {
        self.name = @"jumei";
        self.age = @"4";
    }
    return self;
}
```

顺便提一句，不要在`init`方法中，初始化耗时的代码，`init`方法中应该尽可能快速完成初始化并返回。

BOOL 和 nil

`变量`一定不要直接与`YES`作比较，至于什么原因?请参考BOOL，你真的了解吗？

Pereferred:

```
if (!obj) {

}

if (obj == nil) {
}

if (!isValid) {
}
if (isvalid == NO) {
}
```

Not Pereferred:

```
if (isValid == YES) {

}
```

instancetype & id

在`方便构造器`中，一定要使用 `intancetype`，而且在`init`方法应该多使用`instancetype`。请参考为什么应该用intancetype替代id

pereferred:

```
- (instancetype)initWithName:(NSString)name {

    if (self = [super init]) {

    }

    return self;
}

+ (instancetype)initWithName:(NSString)name {

    if (self = [super init]) {

    }

    return self;
}
```

Block的循环引用

当使用`block`时，一定要注意避免引用循环。可以使用`@weakify`/`@strongify`。

对于只调用一个方法：

Pereferred:

```
__weak __typeof(self)weakSelf = self;
[self executeBlock:^(NSString *str) {
    [weakSelf performSomethingWithString:str];
}];
```

Not Pereferred:

```
[self executeBlock:^(NSString *str) {
    [self performSomethingWithString:str];
}];
```

对于调用多个方法：

Pereferred:

```
__weak __typeof(self)weakSelf = self;
[self executeBlock:^(NSString *str) {
    __strong _typeof(weakSelf) strongSelf = weakSelf;
    if (strongSelf) {
        [strongSelf performSomethingWithString:str];
        [strongSelf performOtherSomethingWithString:str];
    }

}];
```

NOt Pereferred:

```
__weak __typeof(self)weakSelf = self;
[self executeBlock:^(NSString *str) {
    [weakSelf performSomethingWithString:str];
    [weakSelf performOtherSomethingWithString:str];
}];
```

命名规范

命名注意两点：**可读性**与**命名冲突**。

类命名

以`大写字母`作为前缀，可以使用`公司缩写`或者`项目名称缩写`。例如以`WJUserEnity`。形式：`大写字母`+`类名`;命名要清晰，能够描述该类的功能。

常量命名

常量命名应该以`驼峰命名`，并以相关的类名作为前缀。形式：`类名`+`属性`。

Pereferred:

```
static const CGFloat WJOrderListTableViewCellHeight = 44.0f;

NSString *const WJOrderTableViewCellIdentifier = @"WJOrderTableViewCellIdentifier";
```

变量命名

变量命名遵守尽可能能描述的清楚,但也不要画蛇添足。形式：修饰词+[类型]。

Pereferred:

```
NSString *title;
NSAttributedString *titleAttributedString;
NSDate *lastModifiedDate;
```

Not Pereferred:

```
NSString *titleString;
NSAtttributedString *titleAttributedStr;
NSDate *lastModified;
```

#define命名

以`k`开头+`类名`+`属性`的形式。

Pereferred:

```
#define kWJOrderListTableViewCellHeight = 22.0f;
```

枚举命名

枚举命名要加相关类名前缀并且枚举值要有枚举类型前缀，最好用`NS_ENUM`

Pereferred:

```
typedef NS_ENUM(NSInteger,UIViewAnimationTransition) {
    UIViewAnimationTransitionNone,
    UIViewAnimationTransitonFlipFromLeft,
}
```

方法命名

方法命名主体采用`驼峰命名`。方法名前不要有下划线(`_`)，我比较推荐的一种写法是如果是`私有方法`,用`字母`+`下划线`的形式；可以使用情态动词（`can`,`should`,`will`等）来阐明含义，不要使用`do`。

Pereferred:

```
- (void)requestNetwork;
- (void)shouldRequstNetwork;
私有方法
- (void)wj_requestNetwork;
```

Not Pereferred:

```
- (void)doRequstNetork;
- (void)_requstNetwork;
```

带多参数方法命名

方法返回类型与（`-/+`）符合之间应该有一个`空格`。尽可能少用`"and"`这个词。遵循苹果风格,参数前应该有一个描述性`关键词`。

pereferred:

```
-(void)setProductName:(NSString *)name;

- (void)sendMessage:(NSString *)message :(NSString *)client;

- (instancetype)initWithWidth:(CGFloat)width andHeight:(CGFloat)height;
```

Not Pereferred:

```
-(void)setProductName:(NSString *)name;

- (void)sendMessage:(NSString *)message :(NSString *)client;

- (instancetype)initWithWidth:(CGFloat)width andHeight:(CGFloat)height;
```

类目命名

`类目命名`必须描述该类目的功能，且`首字母`必须`大写`。

Pereferred:

```
@interface UIViewController (JMAudioPlaying)
```

Not Pereferred:

```
@interface UIViewController (audioPlaying)
```

类目方法命名

`类目方法命名`的格式：`小写字母`开头+`下划线`+`功能描述`;

Pereferred:

```
- (id)jm_itemAtIndex:(NSUInteger)index;
```

Not Pereferred:

```
- (id)itemAtIndex:(NSUInteger)index;
```

这种格式是苹果官方推荐的，请参考Avoid Category Method Name Clashes

协议命名

协议命名格式：类名+功能描述

```
@protocol JMPickerViewDelegate <NSObject>

@end

@protocol JMMessageData <NSObject>

@end
```

代理方法命名

协议方法命名形式：某类某个对象做了什么。

Pereferred:

```
@protocol JMPickerViewDelegate <NSObject>
- (void)pickerView:(JMPickerView *)pickerView didTapRightButton:(UIButton *)button;
@end
```

Not Pereferred:

```
@protocol JMPickerViewDelegate <NSObject>
- (void)didTapRightButton:(UIButton *)button;
@end
```