Objective-C代码规范

类结构

公共接口

保持public接口的简洁，即在.h文件中只放置需外部可见的方法和property。

内部接口

- 内部实现需要使用的私有propery和方法，在.m文件使用匿名category方法声明。
- 类内部实现的、无需外部可见的协议（ `protocol` ）也在.m文件中的匿名category中声明。

例如:

```
@interface RWTDetailViewController ()<UITableViewDataSource,UITableViewDelegate>

@property (strong, nonatomic) GADBannerView *googleAdView;
@property (strong, nonatomic) ADBannerView *iAdView;
@property (strong, nonatomic) UIWebView *adXWebView;

@end
```

代码组织

- 使用 `#pragma mark -` 对方法按照功能、 `protocol` 实现进行分组。
- `protocol` 的实现使用对应protocol名字作为分组标识。
- ViewControl的有关View生命周期的方法(如 `-init` , `-dealloc` , `-viewDidLoad` , `-viewWillAppear:`)在.m文件的最上方，统一使用分组名为 `View lifecycle` 。

例如:

```
#pragma mark - View lifecycle

- (instancetype)init {}
- (void)dealloc {}
- (void)viewDidLoad {}
- (void)viewWillAppear:(BOOL)animated {}
- (void)didReceiveMemoryWarning {}

#pragma mark - Custom Accessors

- (void)setCustomProperty:(id)value {}
- (id)customProperty {}

#pragma mark - IBActions

- (IBAction)submitData:(id)sender {}

#pragma mark - Public

- (void)publicMethod {}

#pragma mark - Private
```

```
- (void)privateMethod {}

#pragma mark - Protocol conformance
#pragma mark - UITextFieldDelegate
#pragma mark - UITableViewDataSource
#pragma mark - UITableViewDelegate

#pragma mark - NSCopying

- (id)copyWithZone:(NSZone *)zone {}

#pragma mark - NSObject

- (NSString *)description {}
```

property

- 明确指定propery的内存管理属性(Attributes),例如 `strong` 、 `assign` 、 `weak` 、 `copy` 等

  推荐方式

  ```
  @property (nonatomic, weak) IBOutlet UIView *containerView;
  @property (nonatomic,strong) NSString *tutorialName;
  ```

  不推荐方式

  ```
  @property (nonatomic, weak) IBOutlet UIView *containerView;
  @property (nonatomic) NSString *tutorialName;
  ```

- 读取和修改property一律使用dot(.)操作符,调用方法使用[]操作符.

  推荐方式:

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

  不推荐方式:

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

- 类实现中访问property关联的实例变量(Instance Variable)一律使用 `self.` 语法，不使用 `_` 开头直接访问方法（ `-init` , `-dealloc` ,定制的 `getter` 和 `setter` 方法除外）。

  原则上不需要在 `property` 之外，声明单独的实例变量。

构造方法

类构造方法的返回值为 `instancetype` ,而不是 `id` 。

```
@interface Airplane
+ (instancetype)airplaneWithType:(RWTAirplaneType)type;
@end
```

dealloc方法

注意在 `dealloc` 方法中移除observer,KVO等

```
- (void)dealloc{
    [[NSNotificationCenter defaultCenter] removeObserver:self];
}
```

命名

- 使用驼峰式命名法。
- 使用清晰、描述性的文字命名，不使用非通用的缩写。

  推荐方式:

  ```
  UIButton *settingsButton;
  ```

  不推荐方式:

  ```
  UIButton *setBut;
  ```

- 统一前缀

  对于类(`Class`)名、协议（ `Protocol` ）名、常量、typedef中的枚举变量名可在前面加上2-3个字符的大写前缀，表示命名空间；如 `NL` 可代表 `Netease Lottery` 。

类名

- 类名加前缀，彩票为 `NL` 。
- `UIViewController` 子类的命名格式为: `[前缀]+[类描述]+ViewController` ， 例如 `NLPushNotificationSettingViewController` 。

变量/property名

- 单字符的变量名(`i` ，`j` ，`k` 等)只允许出现在 `for()` 循环中。
- 星号 `*` 需靠近到变量一侧，而不是在类型一侧。

  推荐方式:

  ```
  NSString *text;
  ```

  不推荐方式:

  ```
  NSString* text;
  NSString * text;
  ```

- `collection` 类型(如 `NSArray` , `NSDictionary` , `NSSet` 及其变体)的实例变量名字后面加标示类型的后缀，如 `xxxxArray` , `xxxxDic` , `xxxxSet` 等。

方法名

- 在方法名的 `-/+` 符号后，保留一空格。
- 方法的每个参数段(method segments)中间保留一空格。
- 每个形参之前使用关键字对该参数进行描述。
- 避免方法名中包含 `and` ，除非在一个方法中完成两个不同的操作.

推荐方式:

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

不推荐方式:

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

- 表示某个Action的方法，以action命名，不以触发方式命名.

推荐方式:

```
- (IBAction)showDetailViewController:(id)sender
```

不推荐方式:

```
- (void)detailButtonTapped:(id)sender
```

代理方法

- 代理方法的首个参数应为发送消息的对象

```
- (BOOL)tableView:(NSTableView *)tableView shouldSelectRow:(int)row;
- (BOOL)application:(NSApplication *)sender openFile:(NSString *)filename;
```

- 代理方法中使用 `did` 和 `will` 来表示某些事已经发生或即将发生;使用 `shoud` 询问某事是否允许发生

```
- (void)browserDidScroll:(NSBrowser *)sender;
- (NSUndoManager *)windowWillReturnUndoManager:(NSWindow *)window;
- (BOOL)windowShouldClose:(id)sender;
```

通知(Notification)名称

通知名称格式为

```
[Name of associated class] + [Did | Will] + [UniquePartOfName] + Notification
```

例如:

- UIApplicationDidBecomeActiveNotification
- UIKeyboardWillShowNotification

图片资源名

图片资源文件的命名的组成建议为[图片类别]*[图片对应的功能]*[颜色、状态、方位等]；比如button_background_refresh_highlighted.png，表示这是一个用于refresh(功能)的button(类别)在highlight(状态)下的background(类别)。

类别(Category)名

类名 `Category` 需要按照其提供的功能组命名，不要在一个 `Category` 里包含不相关的内容。

控制结构

if/else

- 除单行语句外， `if` 和 `else` 的内容都需要使用 `{}` 包含。
- 左括号 `{` 跟在在判断状态的同一行，不另起新行。（ `switch` , `while` ）相同。
- `else` 应该与上一个判断状态的 `}` 在同一行，除非需要注释。

例如:

```
if (somethingIsBad) return;

if (something == nil) {
    // do stuff
} else if{
    // do other stuff
}
// Comment explaining the alternative
else {
    // Do other stuff
}
```

异常处理

- `Exception` 只用于捕获代码错误，不能用于控制代码流程。
- 使用 `NSError**` 参数来传递错误。

数据类型及定义

Literals语法

创建不可变的 NSString 、 NSDictionary 、 NSArray 、 NSNumber 实例时应该使用 Literals 语法。

推荐方式:

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

不推荐方式:

```
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 *buildingStreetNumber = [NSNumber numberWithInteger:10018];
```

常量定义

- 常量使用 static 方式定义，而不是 #define 定义宏的方式,特殊情况下可使用 #define 。

推荐:

```
static NSString * const RWTAboutViewControllerCompanyName = @"RayWenderlich.com";

static CGFloat const RWTImageThumbnailHeight = 50.0;
```

不推荐:

```
#define CompanyName @"RayWenderlich.com"

#define thumbnailHeight 2
```

- 代码中不能出现直接使用 Magic Number ,而使用有意义常量定义名.

推荐方式:

```
if ([pin length] > RBKPinSizeMax)
```

不推荐方式:

```
if ([pin length] > 5)
```

变量类型

- 使用 NSInteger 与 NSUInteger 代替 int 、 long .
- 使用 CGFloat 代替 float .

枚举类型

- 定义枚举类型，使用 NS_ENUM 宏，可以指定变量类型。

例如:

```
typedef NS_ENUM(NSInteger, RWTLeftMenuTopItemType) {
      RWTLeftMenuTopItemMain,
     RWTLeftMenuTopItemShows,
      RWTLeftMenuTopItemSchedule
};
```

- 按位的枚举类型，使用 NS_OPTIONS 宏定义

例如:

```
typedef NS_OPTIONS(NSUInteger, NYTAdCategory) {
      NYTAdCategoryAutos      = 1 << 0,
      NYTAdCategoryJobs       = 1 << 1,
      NYTAdCategoryRealState  = 1 << 2,
      NYTAdCategoryTechnology = 1 << 3
};
```

其他

单例

- 使用线程安全模式创建单例。
- 单例方法名统一为 sharedXXXX 。

```
+ (instancetype)sharedInstance {
      static id sharedInstance = nil;

      static dispatch_once_t onceToken;
      dispatch_once(&onceToken, ^{
        sharedInstance = [[self alloc] init];
      });

      return sharedInstance;
}
```

CGRect方法

访问CGRect的 x,y,width,height时使用 CGGeomety 的函数，避免直接使用 rect.size.width 方式访问。

推荐方式:

```
CGRect frame = self.view.frame;
```

```
CGFloat x = CGRectGetMinX(frame);
CGFloat y = CGRectGetMinY(frame);
CGFloat width = CGRectGetWidth(frame);
CGFloat height = CGRectGetHeight(frame);
CGRect frame = CGRectMake(0.0, 0.0, width, height);
```

不推荐方式:

```
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;
CGRect frame = (CGRect){ .origin = CGPointZero, .size = frame.size };
```

Window

- 需要对主window操作时，使用 `[UIApplication sharedApplication].delegate.window` 引用window。
- 禁止使用 `[[UIApplication sharedApplication] keyWindow]` ，`keyWindow` 为当前接收用户点击事件的window,在有 `UIAlertView` 或键盘显示的时候， `keyWindow` 就变成了对应的alertView和键盘所在的window,而非主界面所在的window。

UIViewController业务类规范

1. 不允许对ViewController做涉及业务的继承(即只能继承自UIViewController,UITableViewController等)
   - 多个ViewController需要实现类似的方法，使用Protocol声明
   - 多个ViewController需要类似的界面展示，则对View进行封装
   - 目前通过继承实现的ViewController类继承层级不能超过2层。
2. 对ViewController类的实现进行简化

   可参考Lighter View Controllers,或对应的翻译版。

   基本思路是:
   - 将业务逻辑、网络请求逻辑移到Model层
   - 将View代码移动View层

参考文档

- Github objective-c-conventions
- raywenderlich objective-c-style-guide
- NYTimes objective-c-style-guide
- Google Objective-C Style Guide
- Coding Guidelines for Cocoa
- Cocoa代码规范官方指南（要点与简译）