平安好房iOS代码规范

- 使用驼峰命名
- 不允许直接声明和使用实例变量，应当使用属性变量

```
错误：

@interface object{
    NSString *_name;
}
@end

正确：

@interface object

@property (nonatomic, strong) NSString *name;

@end
```

1.1 UIViewController

1.1.1 统一代码布局

严格按照以下顺序组织viewController中的代码：

- #pragma mark - life cycle
- #pragma mark - public methods
- #pragma mark - private methods
- #pragma mark - event response
- #pragma mark - [系统控件的Protocol]
- #pragma mark - [自定义控件的Protocol]
- #pragma mark - getters and setters

其中，`[系统控件的Delegate]` 和 `[自定义控件的Delegate]` 需要替换成对应的真实Delegate名字。这样在Xcode中按住command键点击对应的pragma，就能跳转到Delegate的定义处。

1.1.2 管理好头文件引用

1. 通过换行来将引用的头文件归类(比如UI相关的、功能相关的、第三方)

2. 禁止引用不使用的头文件
3. 第三方Pod的头文件引用全部用 <>

```
//
#import "BLGoodsDetailViewController.h"

#import "BLGoodsDetailCellProtocol.h"
#import "BLGoodsDetailTableViewDataSource.h"
#import "BLGoodsDetailTableViewDataSource+Record.h"
#import "BLGoodsDetailTableViewDataSource+FuLi.h"
#import "BLGoodsDetailTableViewDataSource+Basket.h"
#import "BLGoodsDetailCellEventProxy.h"
#import "BLGoodsDetailAPICenter.h"
#import "BLGoodsDetailNavigationBarViewModel.h"

#import "BLGoodsDetailAllCells.h"
#import "BLGoodsDetailNavigationBar.h"
#import "BLNavigationMiddleSwichView.h"
#import "IBLGoodsDetailSelfPickupActionView.h"
#import "GoodsDetailSegementHeaderView.h"
#import "BLGoodsDetailBottomBar.h"
#import "BLActionSheetView.h"
#import "BLGoodsDetailOffShelfView.h"
#import "BLGoodsDetailGiftGifView.h"
#import "BLGoodsDetailNavigationSegmView.h"

#import <BLCategories/UIResponder+Router.h>
#import <BLCategories/UIColor+Hex.h>
#import <IBLPopDownMenu/IBLPopDownMenu.h>
#import <IBLProgressHud/IBLProgressHud.h>
#import <BLSafeFetchDataFunctions/BLSafeFetchDataFunctions.h>
#import <BLCategories/IBLEmptyDataView.h>
#import <BLCategories/UIViewController+DismissKeyBoard.h>
#import <BLImage/iBLImage.h>

#import <ReactiveCocoa/RACEXTScope.h>
#import <ReactiveCocoa/ReactiveCocoa.h>
#import <IQKeyboardManager/IQKeyboardManager.h>

```

1.2 命名规范

1.2.1 变量名、函数名

1.2.1.1 变量名

- 使用驼峰命名法
- 不用知道上下文，光看变量名就能知道这个变量是干什么的
- 不用知道上下文，光看变量名就能知道这个变量是什么类型
    - UIView系列的全部以View结尾

- UIButton系列的全部以Button结尾
- UIGestureRecognizer系列的全部以Recognizer结尾
- UIViewController系列的全部以ViewController结尾
- 以此类推

1.2.1.2 函数名

- 不用知道上下文，光看函数名就能知道这个函数是干什么的
- 不用知道上下文，光看函数名就能知道这个函数会在什么时候被调用
- -/+ [空格] [返回类型] [函数名]

```
错误：
-(void)functionName;
-(void) functionName;
- (void) functionName;

正确：
- (void)functionName;
```

1.2.2 Notification名

- k + 发送者名 + 事件名 + Notification

```
k UIApplication WillTerminate Notification
k BLLoginManager DidLoggedIn Notification
```

1.2.3 Delegate函数名

delegate方法四要素：

1. 返回类型
2. 自己
3. 事件
4. 反馈参数

```
- (void)manager:(XXManager *)manager didFailedWithErrorCode:(NSString)errorCode er
rorMessage:(NSString *)errorMessage;

- (void)managerTaskDidFinished:(XXManager *)manager;
```

delegate方法第一个参数永远都应该是自己。

1.2.4 enum/option名

`enum/option类型名：`

类名 + XXX Type/Style

```
PAAPIErrorType
```

`enum/option 值名：`

enum/option类型名 + 具体的内容

```
PAAPIErrorTypeNoContent
PAAPIErrorTypeParamsError
PAAPIErrorTypeTimeout
```

1.2.5 事件响应函数名

1.2.5.1 Notification响应函数名

统一使用 `didReceive` 开头，后面跟Notification的名字。

```
- (void)didReceive[Notification名]:(NSNotification *)notification;
```

1.2.5.2 UIButton响应函数名

统一使用 `didTapped` 开头，后面跟Button的名字。

```
- (void)didTapped[Button名]:(UIButton *)button;
```

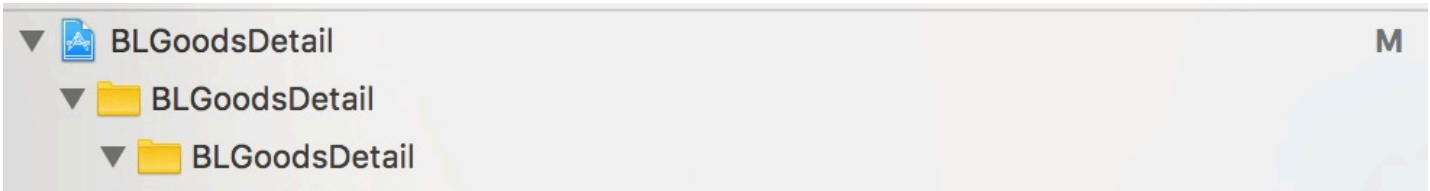
1.2.5.3 UIGestureRecognizer响应函数名

统一使用 `didRecognized` 开头，后面跟recognizer的名字。recognizer的名字需要体现出具体的手势，例如滑动手势就应该命名为 `[页面名]SwipeGestureRecognizer`

```
- (void)didRecognized[GestureRecognizer名]:(UIGestureRecognizer *)recognizer;
```

1.3 文件目录结构规范

文件目录结构样例：



1.3.1 每一个Group都有对应的文件夹

Group需要和文件路径上一致，这样子便于代码文件的迁移。同时，在项目工程中也比较容易定位。

1.3.2 目录层级关系可以表达调用关系或依赖关系

这么做便于了解某一个对象或者模块，都需要哪些辅助文件。也便于后续做代码复用和拆分。

1.3.3 单个文件夹下最多只能有一个对象的源文件

这一个对象必定是这个目录下的主要对象，这样才能使得文件结构主次分明。

1.4 符号使用规范

1.4.1 运算符号两边都要有空格

```
错误：
a+b
a?b:c

正确：
a + b
a ? b : c
```

1.4.2 使用()来表达优先级

```
错误：
a || !b && c

正确：
( a || b ) && ( c || d )
```

1.4.3 {}的使用规范

函数中 `{}` 换行

```
错误：
- (void)foo {
    ...
}

正确：
- (void)foo
{
    ...
}
```

if-else中 `{}` 不换行，else中的 `{}` 不换行：

```
错误：
if (foo)
{
    ...
}
else
{
    ...
}

if (foo) {
    ...
}else{
    ...
}

正确：
if (foo) {
    ...
} else {
    ...
}
```

1.4.4 用换行分隔意群

有的时候一个函数里面代码量比较大，但是这部分代码量又没有大到必须拆成多个函数的情况，就需要使用换行去分隔意群。所谓意群就是做同一件事情的几行代码。

在大函数中，应该尽可能把做同一件事情的几行代码写在一起，然后作为一个意群，用空行分隔开。

```objc
#pragma mark - life cycle
- (void)viewDidLoad
{
    [super viewDidLoad];
    [self setupForDismissKeyboard];
    self.view.backgroundColor = [UIColor whiteColor];

    [self.view addSubview:self.tableView];
    [self.view addSubview:self.navigationBar];
    [self.view addSubview:self.segmentHeaderView];
    [self.view addSubview:self.offShelfView];
    [self.view addSubview:self.scrollTopButton];
    [self.view addSubview:self.salesEmptyTipsLabel];
    [self.view addSubview:self.bottomBar];
    [self.view addSubview:self.gifView];

    [IBLProgressHud showInView:self.view isCanBack:YES];
    [IBLProgressHud showInView:self.view isCanBack:YES];

    [self.apiCenter viewDidLoad_loadAPI];

    [self.dataSource reloadBottomBarStyle];
    [self.dataSource reloadBottomBarState];

    [self bindSignals];
}
```

```objc
- (UITableView *)tableView
{
    if (_tableView == nil) {
        _tableView = [[UITableView alloc] initWithFrame:CGRectZero style:UITableViewStyleGrouped];
        _tableView.delegate = self;
        _tableView.dataSource = self.dataSource;
        _tableView.separatorStyle = UITableViewCellSeparatorStyleNone;
        _tableView.backgroundColor = [UIColor colorWithHex:0xF6F6F6];
        _tableView.estimatedRowHeight = 0;
        _tableView.estimatedSectionHeaderHeight = 0;
        _tableView.estimatedSectionFooterHeight = 0;

        if (@available(iOS 11.0, *)) {
            _tableView.contentInsetAdjustmentBehavior = UIScrollViewContentInsetAdjustmentAutomatic;
        }else{
            self.automaticallyAdjustsScrollViewInsets = NO;
        }
    }
    return _tableView;
}
```

1.5 注释规范(最终目的)

我们的注释规范就是尽最大可能把代码写好，然后不写注释！

> 在好的代码和设计中，不需要写注释、写文档

1. 代码就是最好的文档（头文件、实现文件）
2. 尝试把你的注释全部删掉，把代码设计成不用注释也能够看懂的样子（命名、流程设计）

3. 函数名长点儿没关系，表意清楚最重要

> 在好的工程中，不允许有被注释的代码

1. 通过版本管理操作来消灭调试开发过程中的注释代码
2. 手脚干净地调试项目，调试完后及时删除，不拖泥带水提交注释代码入库