iOS编码规范

1.指导原则

1.1 首先软件代码要易读、易维护。

说明：这是软件开发的基本要点，软件的生命周期贯穿产品的开发、测试、生产、用户使用、版本升级和后期维护等长期过程，只有易读、易维护的软件代码才具有生命力。

1.2 编程时要保持代码简明清晰，避免过分的编程技巧。

说明：简单是最美。保持代码的简单化是软件工程化的基本要求。不要过分追求技巧，否则会降低程序的可读性。

1.3 编程时首先达到正确性，其次考虑效率。

说明：编程首先考虑的是满足正确性、健壮性、可维护性、可移植性等质量因素，最后才考虑程序的效率和资源占用。

1.4 编程时要考虑到代码的可测试性。

说明：不可以测试的代码是无法保障质量的，开发人员要牢记这一点来设计、编码。实现设计功能的同时，要提供可以测试、验证的方法。

1.5 方法是为一特定功能而编写，不是万能工具箱。

说明：方法是一个处理单元，是有特定功能的，所以应该很好地规划方法，不能是所有东西都放在一个方法里实现。

1.6 多加注释。

2.布局

程序布局的目的是显示出程序良好的逻辑结构，提高程序的准确性、连续性、可读性、可维护性。更重要的是，统一的程序布局和编程风格，有助于提高整个项目的开发质量，提高开发效率，降低开发成本。同时，对于普通程序员来说，养成良好的编程习惯有助于提高自己的编程水平，提高编程效率。因此，统一的、良好的程序布局和编程风格不仅仅是主观美学上的或是形式上的问题，而且会涉及到产品质量，涉及到个人编程能力的提高，必须引起大家重视。

2.1 文件布局

2.1.1 遵循统一的布局顺序来书写头文件。

说明：以下内容如果某些节不需要，可以忽略。但是其它节要保持该次序。

```
头文件布局：
文件头
#import （依次为标准库头文件、非标准库头文件）
全局宏
常量定义
全局数据类型
类定义
```

正例:

```
/****************** 文件引用 ******************/
/****************** 类引用 ******************/
/****************** 宏定义 ******************/
/****************** 常量 ******************/
/****************** 类型定义 ******************/
/****************** 类定义******************/
```

2.1.2 遵循统一的布局顺序来书写实现文件.

说明：以下内容如果某些节不需要，可以忽略。但是其它节要保持该次序。

```
实现文件布局：
文件头（参见“注释”一节）
#import （依次为标准库头文件、非标准库头文件）
文件内部使用的宏
常量定义
文件内部使用的数据类型
全局变量
本地变量（即静态全局变量）
类的实现
```

正例：

```
/****************** 文件引用 ******************/
/****************** 宏定义 ******************/
/****************** 常量 ******************/
/****************** 类型定义 ******************/
/****************** 全局变量 ******************/
```

2.1.3 包含标准库头文件用尖括号 < >，包含非标准库头文件用双引号 “ ”。

正例：

```
#import <UIKit/UIKit.h>
#import “MyTableviewController.h”
```

2.2 基本格式

2.2.1 if、else、else if、for、while、do 等语句自占一行，执行语句不得紧跟其后。不论执行语句有多少都要加 { }。而且 if、else、else if、for、while、do 等语句和 ( )，以及 ( ) 和 { } 之间要有空格。

说明：这样可以防止书写失误，也易于阅读。

正例：

```
if (varible1 < varible2) {
      varible1 = varible2;
}
```

反例：下面的代码执行语句紧跟if的条件之后，而且没有加{}，违反规则。

```
if (varible1 < varible2)
varible1 = varible2;
```

【建议2.2.1】源程序中关系较为紧密的代码应尽可能相邻。

说明：这样便于程序阅读和查找。

正例：

```
//矩形的长与宽关系较密切，放在一起。
iLength = 10;
iWidth = 5;
StrCaption = “Test”;
```

反例：

```
iLength = 10;
strCaption = “Test”;
iWidth = 5;
```

2.2.2 定义指针类型的变量，*应放在变量前。

正例：

```
float *pfBuffer;
```

反例：

```
float* pfBuffer;
```

2.3 空格和对齐

2.3.1 property属性括号两边各空一格，属性关键字以逗号加空格分开。

```
@property(nonatomic, retain)NSString *foo;   (不合适，括号两边各空一格)
@property (nonatomic,retain) NSString *foo;  (不合适，属性关键字以逗号加空格分开)
@property (nonatomic, retain) NSString *foo; (合适)
```

2.3.2 赋值操作运算符两边有一个空格

```
_myName = name; (合适)
_myName=name; (不合适，= 两边没有空格)
```

2.3.3 头文件中，**#import**作为一块， **@protocol** 和 **@class** 声明作为一块, **@interface** 作为一块,其中属性之间不空格，方法之间空一格； **@interface** 和 **@protocol** 之间空一行, **@protocol** 如果要在头文件中声明，统一放在 **@interface** 下面，各模块空格和注释参考如下实现：

```
#import <Foundation/Foundation.h>
#import "MyClass.h"

@protocol FooDelegate;
@class IDFDocument;

@interface IDFFoo : NSObject

@property (nonatomic, copy) NSStrin *foo;
@property (nonatomic, strong) id<FooDelegate> delegate;

+ (id)fooWithString:(NSString *) string;
//initialization
+ (id)init;
//init with a string
+ (id)initWithString:(NSString *) string;

@end

@protocol FooDelegate <NSObject>

- (void)iDFFooAskForReloadData:(IDFFoo *)foo;
- (void)iDFFoo:(IDFFoo *)foo requestWithData:(NSData *)data
 error:(NSError *)error;

@end
```

2.3.4 头文件中不出现私有变量申明，私有变量可在**.m**文件中以扩展的方式申明，如下**MyClass.m**文件中的私有变量申明：

```
@interface MyClass()

@property (nonatomic ,assign) NSString *privateParam;

@end


@implemention MyClass

@end
```

2.3.5 实现文件中，**#import**作为一块，**@synthesize**作为一块，**@synthesize**和**@implementation**之间空一行，且采用**@synthesize varName = _varName** 的方式；**#pragma mark - XXX** 写在一行，且和其他函数之间有空行并且对方法进行分组，举例如下:

```
#import "IDFFoo.h"
#import "MyClass.h"

@implementation IDFFoo

@synthesize foo = _foo;
@synthesize bar = _bar;
@synthesize delegate = _delegate;

#pragma mark - LifeCycle

- (void)dealloc {
    [_foo release];_foo = nil;  //OR self.foo = nil;
    [_bar release];_bar = nil;  //OR self.bar = nil;
    [super dealloc];
}

#pragma mark - UITableViewDataSource Methods

#pragma mark - UIScrollViewDelegate Methods

#pragma mark - UITextFieldDelegate Methods

#pragma mark - UITextViewDelegate Methods
```

2.3.6 实现文件中，采用类似如下的方式对函数分组，确保每个函数都在合适的组内:

```
#pragma mark - LifeCycle (类的生命周期相关函数)

- (instancetype)init;

- (void)viewDidLoad;

- (void)dealloc;

- (void)setName;

- (NSString *)name;


#pragma mark - Private (私有函数，比如算法，工具函数等)

- (void)helperMethod;

- (void)aPrivateMethod;


#pragma mark - Public (头文件中公开的函数)

- (void)doSomething; // API in h file


#pragma mark - Actions （UIControl的响应函数，比如按钮事件）

- (void)voiceValuedDidChanged:(id)sender;

- (void)startUpload:(id)sender;


#pragma mark - Notification (通知的响应函数)

- (void)deviceDidConnected:(NSNotification *)notification;


#pragma mark - DelegateName (各种代理的实现方法)

- (void)delegateCallback:(id)sender;
```

2.3.7 函数之间空一行，函数内部逻辑相关的代码块空一行, 函数的 ”{" 建议放在函数同一行

```
- (NSString *)foo {
    return _foo;
}

- (void)setFoo:(NSString *)newFoo {
    if (_foo != newFoo) {
        [_foo release];
        _foo = [newFoo retain];
}

//some other code block

}
```

2.3.8 权限控制符**@public** 和**@private** 缩进一个空格.

```
@interface MyClass : NSObject {

 @public
 ...

 @private
 ...
}

@end
```

2.3.9 在**-**或**+**和返回类型之间留一个空格,在*前留一个空格,其余地方不留空格.

```
- (void)doSomething; (合适)
- (void)doSomething; (不合适， -或+和返回类型之间要留一个空格)
- (void) doSomething; (不合适，“)”和 doSomething之间不空格)
```

2.3.10 多个参数(>=3)的对齐使用XCode的默认对齐方式

3.注释

注释有助于理解代码，有效的注释是指在代码的功能、意图层次上进行注释，提供有用、额外的信息，而不是代码表面意义的简单重复。

3.1 多行注释采用“/* … */”，单行注释采用“// …”。

建议: 不管多行还是单行，采用注释符“/* … */”。

3.2 一般情况下，源程序有效注释量必须在30%以。

说明: 注释的原则是有助于对程序的阅读理解，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。有效的注释是指在代码的功能、意图层次上进行注释，提供有用、额外的信息。

3.3 文件头部必须进行注释

3.4 方法头部应进行注释，列出：方法的目的/功能、输入参数、输出参数、返回值、访问和修改的表、修改信息等，除了方法名称和功能描述必须描述外，其它部分建议写描述。

```
/**
* 功能描述:  // 方法功能、性能等的描述
* 输入参数:  // 输入参数说明，包括每个参数      的作用、取值说明及参数
间关系
* 返 回 值:  // 方法返回值的说明
* 其它说明:  // 其它说明
**/
```

3.5 注释应与其描述的代码相近，对代码的注释应放在其上方或右方（对单条语句的注释）相邻位置，不可放在下面，如放于上方则需与其上面的代码用空行隔开。

说明: 在使用缩写时或之前，应对缩写进行必要的说明。

正例：如下书写比较结构清晰

```
/* 获得子系统索引 */
iSubSysIndex = aData[iIndex].iSysIndex;

/* 代码段1注释 */
[ 代码段1 ]

/* 代码段2注释 */
[ 代码段2 ]
```

反例1：如下例子注释与描述的代码相隔太远。

```
/* 获得子系统索引 */
iSubSysIndex = aData[iIndex].iSysIndex;
```

反例2：如下例子注释不应放在所描述的代码下面。

```
iSubSysIndex = aData[iIndex].iSysIndex;
/* 获得子系统索引 */
```

反例3：如下例子，显得代码与注释过于紧凑。

```
/* 代码段1注释 */
[ 代码段1 ]
/* 代码段2注释 */
[ 代码段2 ]
```

3.6 注释与所描述内容进行同样的缩排。

说明: 可使程序排版整齐，并方便注释的阅读与理解。

正例：如下注释结构比较清晰

```
- (int)doSomething
{
    /* 代码段1注释 */
    [ 代码段1 ]

    /* 代码段2注释 */
    [ 代码段2 ]
}
```

反例： 如下例子，排版不整齐，阅读不方便;

```
- (int)doSomething
{
    /* 代码段1注释 */
    [ 代码段1 ]

    /* 代码段2注释 */
    [ 代码段2 ]
}
```

3.7 注释必须列出：版权信息、文件标识、内容摘要、版本号、作者、完成日期、修改信息等。修改记录部分建议在代码做了大修改之后添加修改记录。备注：文件名称，内容摘要，作者等部分一定要写清楚。

3.8 当if语句的判断条件复杂时，需要用注释说明判断内容

3.9 接口类的头文件每个方法前都应该注明方法的作用。

```
#import <Foundation/Foundation.h>
#include <UIKit/UIKit.h>

// 轮播图
@interface TZRollPictureBy : NSObject

@property (nonatomic, strong) NSMutableArray *images; //图片
链接数组

// 初始化类方法
+ (instancetype)shareInstance;

// 轮播方法
- (void)createPictureByWithSuperView:(UIView *)superView He
ight:(CGFloat)height;
```

@end

4.命名规则

好的命名规则能极大地增加可读性和可维护性。同时，对于一个有上百个人共同完成的大项目来说，统一命名约定也是一项必不可少的内容

4.1 方法命名

4.1.1 方法名首字母小写,其他首字母大写

```
doSomethingWithParam: (合适)
DosomethingWithParam: (不合适，首字母要小写，之后要使用首字母大写)
```

4.1.2 简短清晰的同时要保证语意的清晰

```
insertObject: atIndex: (合适)
insert: at: (不合适，表达不清晰；例如什么被插入？at具体指什么？)

removeObjectAtIndex: (合适)
removeObject: (不合适，因为移除了对象相关的主题)
```

4.1.3 大多数时候使用全拼而不是缩写，即使他们很长。

```
destinationSelection (合适)
destSel (不合适，因为表达不清晰)
setBackgroundColor: (合适)
setBkgdColor: (不合适，因为表达不清晰)
```

4.1.4 避免产生歧义的命名

```
sendPort (不合适，是要发送端口？还是从端口返回？)
displayName (不合适，是要显示一个名字？还是返回从用户接口接收到的标题)
```

4.2 变量命名

4.2.1 变量名首字母小写,后续首字母大写，”*"和变量名之间不空格。

```
@property (atomatic, copy) NSString *myDevice (合适)
@property (atomatic, copy) NSString *MyDevice (不合适，首字母要小写，之后要使用首字母大写)
@property (atomatic, copy) NSString * myDevice (不合适，”*”和变量名之间不空格)

NSString *userName; (合适)
NSString *VarName; (不合适，首字母要小写，之后要使用首字母大写)
NSString * userName; (不合适，”*”和变量名之间不空格)
```

4.2.2 成员变量使用””*作为前缀，临时变量不需要*””前缀。

```
NSString *_varName; (当作为成员变量时 合适)
NSString *varName; (当作为临时变量时，合适，不需要“_”作为前缀)
NSString *varName; (当作为成员变量时，不合适，应使用“_”作为前缀)
```

4.2.3 成员变量名不使用类型前缀或者“m”前缀（该规则同样适用于临时变量），比如定义一个“设备数组”的成员变量：

```
NSArray *_deviceArray; (合适，从名字可看出类型，符合变量的用途：“设备数组”)
NSArray *_arrDevice; (不合适，变量名不使用类型前缀)
NSString *_mDevice; (不合适，成员变量名不使用“m”前缀)
```

4.3 类命名

4.3.1 类名首字母以及后续单词首字母大写。

```
homepageViewController (不合适，首字母需要大写)
VSDevice （合适）
```

4.3.2 类名可使用简短的前缀，比如：项目名缩写，公司名缩写 (立项时，团队

成员可共同拟定)

```
KLMyClass (合适， 前缀使用了KLStudio的前两个字母)
VSHomePageViewController (合适，VS是项目名称缩写)
```

4.4 文件命名

4.4.1 类文件名称所有首字母大写，命名需要和它所实现的类的名字相同，同时需要体现出它的角色，比如控制器需要添加**ViewController**后缀，视图一般情况需要有**View**后缀（模型类不需要添加**Model**作为后缀）。

```
STBDevice (合适，它是一个模型角色)
STBDeviceModel (不合适，它本身代表一个设备，不需要多余“Model”后缀)

HomePageView (合适，它是一个视图角色)
HomePageViewController (合适，它是一个控制器角色)
homePageViewController  (不合适，所有首字母需要大写)
HomePage (不合适，从名字不能看出它是一个控制器还是一个视图)
```

4.4.2 类别的文件名应该使用“类别名 **+** 类别功能说明”。

```
NSString+Utils.h (为NSString添加了一些工具方法的集合)
NSString+MD5.h (为NSString添加了MD5支持)
```

4.4.3 协议的文件名应该使用“协议名 **+ Delegate**” **ReplyViewDelegate**。

4.5 宏，枚举，**UI**控件命名

4.5.1#define的宏作为**key (NSDictionary,NSUserDefault,NSCoding)**，前面加”**k"**关键字；其他宏采用首字母大写，通知需要添加**Notification**后缀。

```
#define CanvasView_Tag 10001
#define ItunesURL   @"http://www.itunesconnect.com"
#define kUserName  @"UserName" (作为字典,NSUserDefault,NSCod
ing的key)
#define DeviceDidConnectedNotification    @“DeviceDidConnec
tedNotification"
```

4.5.2 enums的枚举变量首字母大写，枚举变量带上有意义的枚举名作为前缀，之后是下划线，最后是参数名,枚举数值等号对齐。

```
typedef enum {
    KLMood_Happy = 0,
    KLMood_Grin  = 1,
    KLMood_Sad   = 2,
    KLMood_Mad   = 3
}KLMood; (合适)

typedef NS_ENUM(NSInteger, IDFMood) {
    happyMood = 0,
    grinMood,
    sadMood,
    madMood
}; (不合适，首写字母没有大写 参数名前置不方便阅读)

typedef NS_ENUM(NSInteger, IDFMood) {
    KLMood_Happy = 1<<0,
    KLMood_Grin = 1<<1,
    KLMood_Sad = 1<<2,
    KLMood_Mad = 1<<3
}; (不合适，枚举数值应从等号对齐)
```

4.5.3 使用整形表示某种状态或类型时，需要采用枚举值；使用常量字符串需要在特定文件或实现文件的统一位置声明，也可以采用宏定义的方式声明常量字符串，避免代码出现“魔数”“魔字符串”

4.5.4 界面控件采用 “功能 + 控件名”的命名方式，且控件名使用完整拼写，举例

如下：

```
registerButton
nameLabel
```

4.6 图片命名

4.6.1 采用单词全拼，或者大家公认无岐义的缩写(比如：**nav**，**bg**，**btn**等)。

4.6.2 采用“模块+功能”命名法，模块分为公共模块、私有模块。公共模块主要包括统一的背景，导航条，标签，公共的按钮背景，公共的默认图等等；私有模块主要根据**app**的业务功能模块划分，比如用户中心，消息中心等。

```
tabbar_compose_shooting.png
tabbar_compose_shooting@2x.png
tabbar_compose_shooting@3x.png
tabbar_compose_voice.png
tabbar_compose_voice@2x.png
tabbar_compose_voice@3x.png
tabbar_compose_video.png
tabbar_compose_video@2x.png
tabbar_compose_video@3x.png
```

5.方法和接口

5.1 在**.h**文件中留出与外部程序交互的接口，对**.m**中的实现进行封装和不可视。

5.2 方法尽量具体，功能单一。不要代码进行冗余。

6.约定

6.1 除非有不能删除的说明，注释的代码块在可用的模块中不应该出现

6.2 一个方法只做一件事，一个方法的实现不超过一屏

6.3 if 条件判断推荐使用 **if (true)** 写法

推荐写法：

```
if (true) {
   //Do something...
}
```

不推荐写法：

```
if (false) {
   return;
}//Do something...
```

7.其他补充

7.1 操作符前后都要加空格

7.2 避免相同的代码段在多个地方出现

7.3 语句嵌套层次不得超过**3**层

7.4 每个实现文件建议在**500**行以内，不能超过**1000**行，超过之后应考虑通过抽象类对代码进行重构

7.5 及时删除或注释掉无用的代码

7.6 UITableViewCell里面的network client都要委托出来

7.7 点击按钮之后需要切换按钮图片，当这两张图片没有关联时（例如一张图片相比另一张图片有选中效果），不应该设置为UIControlSelected

7.8 控件布局使用相对坐标

7.9 确定不使用的代码应该删除

7.10 在新建或已有的工程配置Organization Name栏填入公司英文名称，在偏好设置的帐户设置中使用自己的姓名全拼或昵称

7.11 使用开源代码时，保留版权信息

```
// TZTextFieldViewController.h
// XJHD_Customer
//
// Created by 糯米团子 on 15/9/1.
//  Copyright (c) 2015年 com.ync. All rights reserved.
```