Objective-C Style Guide

Based on Google Objective-C Style Guide

Revision 1.1.1

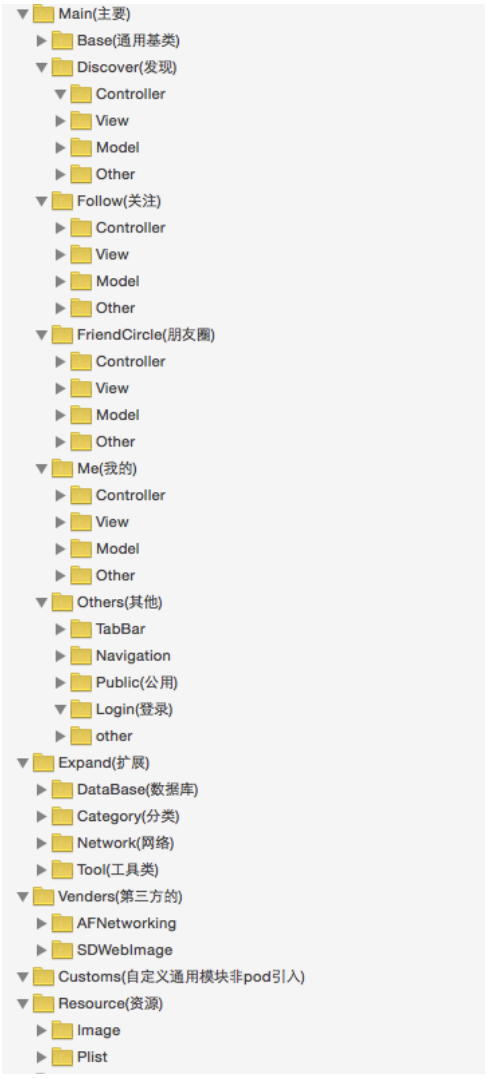
Xiamen Meitu Technology Co., Ltd.

工程索引与目录规范

- **工程中目录和文件命名统一使用英文**
- **App主目录按模块分类，内目录按业务分类**

  具体工程目录分层结构如下：

  `实际项目中索引文件夹名称括号内的汉字不需要，此处只能说明使用；功能模块文件夹命名仅供参考`



  说明：通用宏定义、打包环境标识，单独使用文件管理。（建议放在AppDelegate同一目录）

- 资源文件也按模块分类(上图Resource中Image里也按模块分类)

编码风格规范

import引入头文件按分类排列顺序:系统头文件>第三方头文件>自定义模块头文件。

import引入头文件按分类排列顺序事例：“//“的注释部分非必须存在部分。

```
#import "MYClass.h"

// 系统库头文件
#import <Foundation/Foundation.h>
#import <CoreFoundation/CoreFoundation.h>

// 第三方库头文件
#import "GPUImage.h"
#import "AFNetWorking.h"

// 自定义模块的头文件
#import "AppSettingsUtil.h"
```

import引入自定义文件部分建议按:ViewController > View > control/model

空格与制表符 Spaces vs. Tabs

- 1.只允许使用空格,并且每次缩进四个空格。将编辑器设置为Tab = 4 Spaces。
- 2.类中成员变量类型申明时，@public, @private indent 缩进2个空格
- 3.行宽 Line Length代码中的每行最多不超过100个字符。 设置“Xcode => Preferences => TextEditing => Show Page Guide”显示单页辅助线

指针“＊” 的位置

代码中“＊”与变量类型间空格隔开，与变量名相连。如 NSString *varName;

方法声明与定义 Method Declarations and Definitions

- or + 和返回类型之间留1个空格。参数列表中,参数之间留1个空格。如:

```
- (void)doSomethingWithString:(NSString *)theString {
...
}
```

当参数过长，每个参数占用一行，以冒号对其。 如:

```
- (void)doSomethingWith:(GTMFoo *)theFoo
                   rect:(NSRect)theRect
               interval:(float)theInterval {
   ...
}
```

当方法名比参数名短，每个参数占用一行，至少缩进4个空格，并垂直对齐(非冒号对齐)。 如:

```
- (void)short:(GTMFoo *)theFoo
    longKeyword:(NSRect)theRect
    evenLongerKeyword:(float)theInterval {
    ...
}
```

方法调用 Method Invocations

1. 方法调用与方法声明的格式一致。

   例外：如果源文件已经遵从一定的格式，则修改的代码要与源代码一致。

2. 方法调用时,所有参数应该在同一行;或者每行一个参数,并以冒号对齐.

   ```
   // ...be great!
   //所有参数在同一行
   [myObject doFooWith:arg1 name:arg2 error:arg3];
   //每行一个参数,并以冒号对齐:
   [myObject doFooWith:arg1
                        name:arg2
                       error:arg3];
   ```

   不要使用以下缩进风格:

```
// ...be forbidden!
[myObject doFooWith:arg1 name:arg2  // some lines with >1 arg
             error:arg3];

[myObject doFooWith:arg1
              name:arg2 error:arg3];

[myObject doFooWith:arg1
                    name:arg2  // aligning keywords instead of colons
                    error:arg3];
```

3. 方法定义与方法声明一样,当关键字的长度不足以以冒号对齐时,下一行都要以四个空格进行缩进。

```
[myObj short:arg1
    longKeyword:arg2
    evenLongerKeyword:arg3];
```

协议 Protocols

1. **尖括号所包括的协议名称与前面的类型标识之间不应该有空格，尖括号与类型名之间隔开一个空格。**

   这条规则也同样适用于类声明、成员变量以及方法声明。例如:

```
@interface MyProtocoledClass : NSObject <NSWindowDelegate> {
}
- (void)setDelegate:(id<MyFancyDelegate>)aDelegate;
@end
```

2. **如果类声明中包含多个protocol，每个protocol占用一行，缩进4个字符（空格）。** 如:

```
@interface CustomViewController : ViewController <
     AbcDelegate,
     DefDelegate>
{
    ...
}
```

“{”和“}”匹配对其规范

1. 接口实现部分和if while，”{”和“}”的匹配风格如下:

```
// 接口实现{}匹配风格
- (void)doSomething {
    ...
}

// if{}匹配风格
if (condition) {
    ...
} else {
    ...
}
```

2. if、while 嵌套不能超过三层，超过时提取成方法，再进行调用。
3. if、while的“{}”中代码较长（超过50行）需根据实际情况考虑是否能封装成method，用于提高代码可读性；接口实现部分尽量不超过100行（xcode一页显示范围），超过之后需要提取为method。

代码段

1. 接口内部代码段之间使用回车空一行隔开

```
代码段1...

代码段2....
```

块 Blocks

1. block 的循环引用(retain cycle)

   > 解决方案：将引用的一方变成weak，从而避免循环引用。

2. block实现部分较长时，需要定义成block变量。（原因:提高可读性，同时可以控制接口代码长度）

其他 Others

1. init 和 dealloc 是最常用的方法，放在类实现的开始位置。
2. 使用空格将相同的变量、属性对齐，使用换行分组。
3. 类定义、属性声明、 接口声明、协议声明之间使用换行隔开；属性声明中使用换行将不同组的属性隔开。

命名规范

文件名 File Names

1. 文件名应反映它们所包含的类实现的名称及实现的逻辑，并遵循项目当前风格。

UI资源文件命名

具体格式：

```
type_location_identifier_state@2x.png
```

例如：

```
icon_home_new_normal@2x.png/icon_home_new_normal@3x.png

type:
    icon: icon
    UIButton: btn
    background: bg

state: 采用系统UIButton的状态
    normal-正常状态
    highlight-高亮状态
    disable-禁按状态
```

UI资源文件命名规则说明参考下图：

Naming Systems

Coming up with a standard naming system for your assets and components on a project eases the transition from design to development and will make everyone happy! Although different people and studios will have their own individual way of doing things a good approach is to base your naming on a hierarchical system, which starts off with a broad identification of the component and then progressively adds more levels of detail. So you might use a structure like this:

type_location_identifier_state

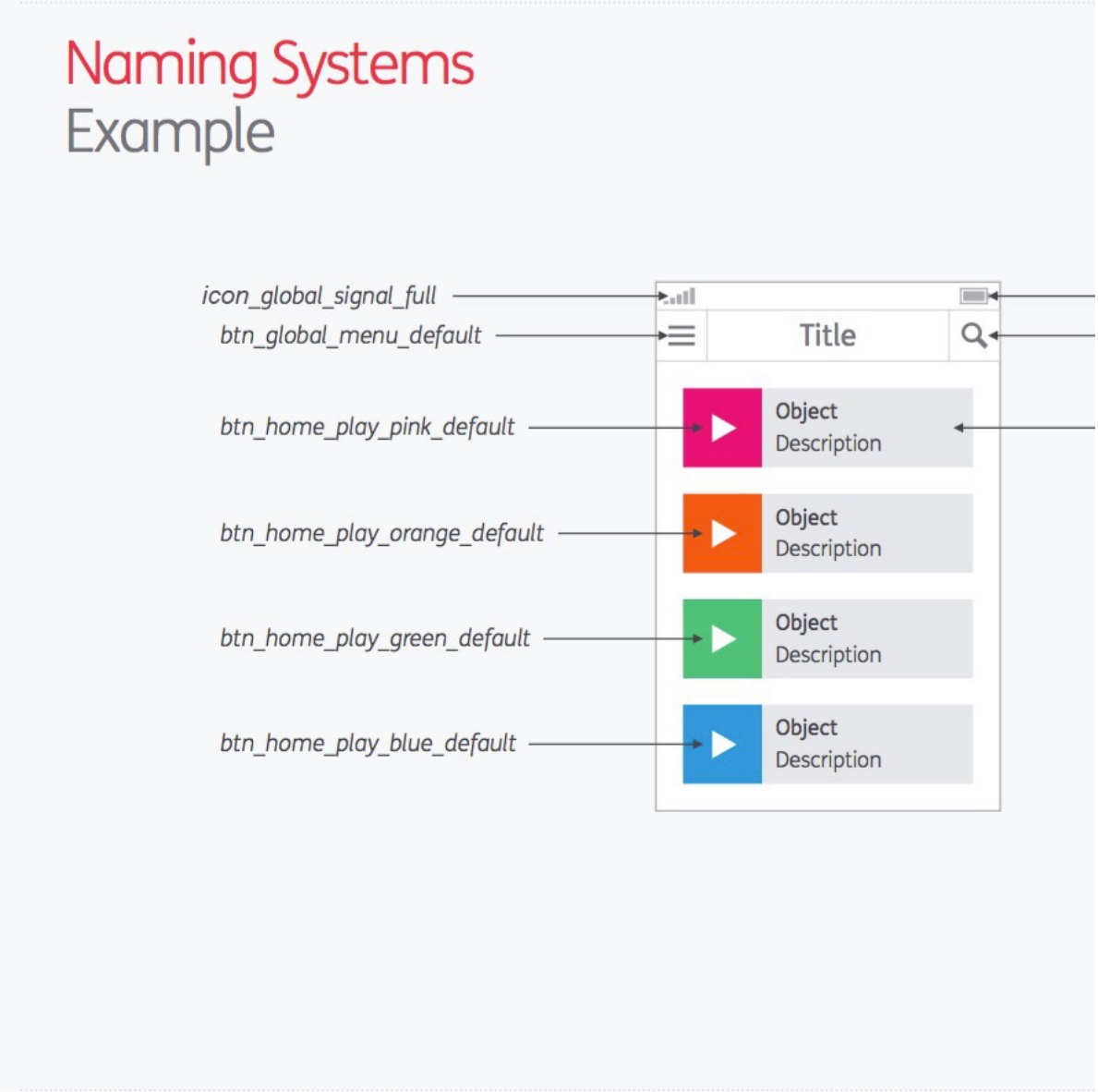
The *type* refers to what category the component belongs to, such as:

bg (background) *btn* (button) *icon* (icon)

The next step is to add the screen or location where this component appears:

bg_help *btn_home* *icon_global*

(*global* refers to components that appear across multiple screens or sections)



扩展以及说明:

1. 不同设备使用不同UI资源时时，在state前添加_屏幕寸数大小.

```
icon_home_new_3_5_normal.png  (iphone4和iphone4s)
icon_home_new_4_0_normal.png  (iphone5~iphone5s)
icon_home_new_4_7_normal.png  (iphone6 iphone6s)
icon_home_new_5_5_normal.png  (iphone6plus iphone6splus)
icon_home_new_9_7_normal.png  (ipad)
```

2. 常规UI资源统一只提供@2x和@3x的图片，接在state后面。（plus系列设备使用@3x，非plus设备使用@2x）

Objective-C++

当编写源代码时, Objective-C++应该采用当前的代码风格。

- 为了最小化Cocoa/Objective-C与C++之间的命名风格的冲突。 在实现 @implementation 语句块时,使用Objective-C的命名规则;如果实现 C++ class, 则使用 C++ 命名规则。

类命名 Class Names

类名(包括 category,protocol)首字母大写，并使用首字母大写的形式分割单词。

1. 在面向特定应用的代码中，类名应尽量避免使用前缀，每个类都添加前缀影响可读性。

2. 在面向多应用的代码中，推荐使用前缀。（尽量使用三个大写字母作为前缀，原因：2个大写字母做前缀容易与系统冲突） 如：MTUSendMessage

分类命名 Category Names

类别名应该有两三个字母的前缀以表示类别是项目的一部分或者该类别是通用的。类别应该包含它所扩展的类的名字。

1. 分类的文件名应该包含被扩展的类的名字

```
如：NSString+MTUtils.h 或 NSTextView+MTAutocomplete.h
```

2. 分类接口或属性需要添加相关的前缀

```
如：- (void)mt_myCategoryMethod;
```

类名与包含类别名的括号之间,应该以一个空格分隔。

方法命名 Objective-C Method Names

1. 方法名的首字母小写，且使用首字母大写的形式分割单词。方法的参数使用相同的规则。
2. 方法名+参数应尽量读起来像一句话. 参见Apple's Guide to Naming Methods

```
例如：
    1.convertPoint:fromRect:
    2.replaceCharactersInRange:withString:
```

3. getter的方法名和变量名应相同。不允许使用“get”前缀。本规则仅针对Objective-C代码，C++代码使用C++的习惯

```
例如：
    - (id)getDelegate;  // AVOID（c++文件除外，OC文件见一次打一次的节奏啊）
    - (id)delegate;    // GOOD
```

变量命名 Variable Names

1. 变量名的首字母小写，且使用首字母大写的形式分割单词。

```
例如：myVariable
```

2. 类的成员变量加下划线前缀。

```
例如：NSString *_myInstanceVariable;
```

普通变量名 Common Variable Names

对于静态的类别,如int以及指针等,不要使用匈牙利命名法。要为变量起一个描述性的名字。不要担心浪费列宽,因为让新的代码阅读者立即理解你的代码更重要。

例如:

```
// ...be great!
int numErrors;
int numCompletedConnections;
tickets = [[NSMutableArray alloc] init];
userInfo = [someObject object];
port = [network port];
```

错误示例

```
// ...be forbidden!
int w;
int nerr;
int nCompConns;
tix = [[NSMutableArray alloc] init];
obj = [someObject object];
p = [network port];
```

成员变量 Instance Variables

成员变量应该混合大小写,并以下划线作为前缀。

```
例如 _usernameTextField。
```

成员变量使用“*”作为前缀（如：“NSString *varName;”。虽然这与苹果的标准（使用“*”作为后缀）相冲突，但基于使用“*”作为前缀，更容易在有代码自动补全功能的IDE中区分“属性(self.userInfo)”和“成员变量(*userInfo)”，仍使用“*”作为前缀。

常量 Constants

常量名(如宏、静态局部变量等)应该以小写字母k开头,使用混合大小写的格式来分隔单词。

```
例如：kInvalidHandle, kWritePerm.
```

const的介绍与使用见文档底部参考链接 ***iOS开发：正确使用const,static,extern***

宏定义

宏定义多有字母使用大写，分段部分用“_”隔开。 例如：

```
#define IS_TEST_PACKAGE 1
```

注释 Comments

> *虽然写起来很痛苦,但注释是保证代码可读性的关键。下面的规则给出了你应该在何时、何地进行注释。记住:尽管注释很重要,但好代码应该自成文档。与其给类型及变量起一个晦涩难懂的名字,再为它写注释,不如直接起有意义的名字。 当你写注释的时候,记得你是在给你的听众写,即下一个需要阅读你的代码的开发人员。大方一点,下一个阅读代码的人可能就是你!*

注释内容包括如下几点

```
1.文件注释 - xcode创建文件自带，需xcode设置中修改作者和版权信息
2.每个interface, category, protocol 的声明都应该有配套的注释说明。
3.每个public interface中的方法需要在 .h 文件中有详尽的注释，包括功能，参数，返回值，提醒之类的
4.每个类成员变量需要有相应的注释，说明其用途，包括private的那些
5.枚举类型定义、结构体定义都需要注释说明
6.内部代码段
```

文件注释 File Comments

xcode创建文件时自带的文件注释格式，修改作者和版权即可。修改作者和版权：在 Address Book 里修改自己名片里的Company名称和姓名。

```
//
//  AppDelegate.m
//  LauchImageTest
//
//  Created by ph on 16/2/2.
//  Copyright © 2016年 ph. All rights reserved.
//
```

如果你对其他人的原始代码作出重大的修改,请把你的名字添加到作者里面。当另外一个代码贡献者对文件有问题时,他需要知道怎么联系你,这十分有用。

声明注释 Declaration Comments

每个接口、类别以及协议应该注释,以描述它的目的及作用。

```
// A delegate for NSApplication to handle notifications about app
// launch and shutdown. Owned by the main app controller.
@interface MyAppDelegate : NSObject {
  ...
}
@end
```

如果你已经在文件头部详细描述了接口,可以直接说明“完整的描述请参见文件头部”,但是一定要有这部分注释。

另外,公共接口的每个方法,都应该有注释来解释它的作用、参数、返回值以及其它影响。

如果类的实例可以被多个线程访问，需要为类的线程安全性作注释，描述多线程条件下的使用规则。

实现注释 Implementation Comments

使用| 来引用注释中的变量名及符号名而不是使用引号。

这会避免二义性,尤其是当符号是一个常用词汇,这使用语句读起来很糟糕。例如,对于符 号"count":

```
// Sometimes we need |count| to be less than zero.
```

或者当引用已经包含引号的符号:

```
// Remember to call |StringWithoutSpaces("foo bar baz")|
```

注释风格 The Style Of Comments

注释风格使用javaDoc风格；注释工具使用VVDocumenter-Xcode

需要注释的点：结构体的定义，类的定义、属性、接口，代码段等。

1. 数据结构体的注释

```
/**
 *  自定义坐标点结构体（简单描述）
 */
 typedef struct {
    CGFloat             x;       /**< 坐标点在x轴上的位置 */
    CGFloat             y;       /**< 坐标点在y轴上的位置 */
 } MTPoint;
```

2. 枚举数据类型的注释

```
/**
 *   枚举类型的简单描述
 */
typedef NS_ENUM(NSInteger,AssetState) {
    AssetsState_Normal,              /**< 初始状态 */
    AssetsState_Downloading,         /**< 正在下载状态 */
    AssetsState_Downloaded,          /**< 下载完成状态 */
};
```

   说明：枚举和结构体的注释按"/*< /"形式便于文档快速查看，编写较复杂，可在xcode中设置代码块。

3. 类定义的注释

```
/**
 *  美图秀秀主界面  (类的简单介绍，介绍较长时可另起一行)
 */
@interface HomeViewController : UIViewController
```

4. 类接口的注释

```
/**
 *   美化图片功能  (接口功能简单描述)
 *
 *   @param     sender  美化图片按钮（参数描述）
 *
 *   @return  无  (返回值描述)
 */
 - (IBAction)beauty:(id)sender;
```

5. 成员变量（属性）的注释

注释与成员变量同一行时:

```
UIImage *image;   // image的描述信息
```

注释在成员变量上一行时:

```
// 刷新自由拼图控件的图片位置信息以及背景
UIImage *image;
```

1. 代码段的注释

注释在段代码上一行,且与其他代码或注释行间隔一行，注释格式如下。

```
... (代码段1)

/** 刷新自由拼图控件的图片位置信息以及背景(此处为代码段2的注释) */
... (代码段2)
```

1. 逻辑条件代码的注释

当if，while 等逻辑代码嵌套出现，切“{”和“ }”之前相隔较远时，需要在“}”后加 注释。

注释风格如下：

```
if (condition1) {
  …
  while(condition2) {
          if (condiTion3) {
           …
          } else if (…) {

          }
  }   // end while(condition2)
  …
}  // end if (condition1)
```

1. #pragma mark - 的使用

   在头文件和.m文件中，可以使用#pragma mark – 对接口进行分类和说明，保证 文件结构清晰，易读。

   分类可以根据接口类型和接口功能进行分类。

编码注意事项

1.重载初始化方法 Override Designated Initializer

```
当编写子类实现时，需要init...方法，记得重载父类指定的初始化方法。
```

2.保持接口简单 Keep the Public API Simple

```
尽量避免定义重复功能的公有接口
```

3.#import and #include

```
使用 #import 引入Ojbective-C和Ojbective-C++头文件，使用 #include 引入C和C++头文件
```

4.Use Root Frameworks

```
虽然有时我们仅需要框架（如Cocoa 或  Foundation）的某几个头文件，但引入根文件编译器会运行的更快。因为根框架（root frameworks）一般会预编译，所以加载会更快。

例如:
#import <Foundation/Foundation.h>       // 推荐使用
#import <Foundation/NSArray.h>          // 禁止使用
#import <Foundation/NSString.h>         // 禁止使用
...
```

再次强调：**使用 #import 而非 #include 来引入Objective-C框架。**

5.永远不要仅仅retain一个字符串。这可以避免调用者在你不知道的情况下对字符串作出了修改。不要假设你接受的对象是NSString而不是NSMutableString。

```
@property (nonatomic, copy) NSString *aString;
```

6.Delegate委托对象不能被retained。

7.当三元运算符的第二个参数返回值是条件语句中已经检查的对象时，表达方式如下：

```
result = object ? object : [self createObject];

//不推荐这种表达方式
result = object ? : [self createObject];
```

7.尽量避免同一段代码在多处粘贴拷贝，应尽量抽离出来。

8.如果要定义常量使用static const优于宏定义，前者会进行类型检查

9.删除代码逻辑时应该将相应的不需要的头文件引入、变量等一并删掉。

10.遇到警告要及时处置，不要等累积多了处置就很麻烦了。

BOOL值注意事项

将int值转换为BOOL时应特别小心。避免直接和YES比较。

Objective-C中，BOOL被定义为unsigned char，这意味着除了 YES (1) 和 NO (0)外它还可以是其他值。

常见的错误包括：将数组的大小、指针值或位运算符的结果强制转换为BOOL，此时该BOOL值的结果取决于整型值的最后一个字节的值，仍有可能出现 NO 的值。当进行这类强制转换时，需要使用三元运算符返回 YES / NO 。

BOOL、_Bool和bool之间的转换是安全的，但是BOOL和Boolean间的转换不是安全的，所以必须将Boolean看成整型值如上讨论的。

使用与布尔逻辑运算符(&&,| | and ！)也是有效的，返回值可以安全转换为BOOL值而不需要使用三元运算符。

如：

```
// ...be great!
- (BOOL)isBold {
  return ([self fontTraits] & NSFontBoldTrait) ? YES : NO;
}
- (BOOL)isValid {
  return [self stringValue] != nil;
}
- (BOOL)isEnabled {
  return [self isValid] && [self isBold];
}
```

错误示例

```
// ...be forbidden
- (BOOL)isBold {
  return [self fontTraits] & NSFontBoldTrait;
}
- (BOOL)isValid {
  return [self stringValue];
}
```

禁止直接将BOOL和YES/NO比较，如：

```
// ...be forbidden
BOOL great = [foo isGreat];
if (great == YES)
```

```
// ...be great!
BOOL great = [foo isGreat];
if (great)
```

MVC Model/View/Controller

分离模型与视图。

分离控制器与视图、模型。

其他 Others

宏定义软件正式包和测试包的标记位的规范

软件开发过程中会有测试版（内测）和正式版，不同版本会从不同的网络地址请求网络数据。所以需要宏定义标志伟来方便测试版本和正式版本间的切换。

正式／测试包的标识定义为 0 / 1 (这样可以进行预编译处理；保证代码逻辑清晰)

例如：

```
在AppConfiguration.h 中
#define IS_TEST_PACKAGE 0   // 0:正式包 1:测试包
```

在需要根据IS_TEST_ PACKAGE的宏定义值的预处理结果，对某个宏定义取不同的值的头文件中，添加需要代码：

```
#ifndef IS_TEST_PACKAGE
#if DEBUG
#define IS_TEST_PACKAGE 1
#else
#define IS_TEST_PACKAGE 0
#endif
#endif

#if IS_TEST_PACKAGE
#define CHECK_VERSIONUPDATE_URL @"www.meitutest.com"
#else
#define CHECK_VERSIONUPDATE_URL @"www.meitu.com"
#endif
```

内存泄露和查找

对于项目工程新建的模块文件，需要进行编译警告查找和处理以及内存泄露检测。

在Xcode中先使用Analyze编译条件进行编译，检测一些简单的逻辑错误（可以检测出一些简单的逻辑上的内存泄露），再使用Profile中的InStruments进行调试，查找内存泄露。

规范执行

iOS规范正式发布后开始执行。

1. 新开项目或组件按规范文档执行。
2. 老项目中新建模块、重构模块需按规范文档执行；旧文件涉及修改或扩展时按规范进行整理。
3. 旧项目的旧代码部分各位开发大神可根据实际情况安排整理计划（此条非强制）

参考文献说明：本文档主要参考 Objective-C Style Guidepdf, Cocoa_编码指南.pdf

参考链接:

分享我设计的IOS项目目录结构

iOS 项目的目录结构能看出你的开发经验

google OC style guide R2.59

iOS开发：正确使用const,static,extern