# iOS编程规范

2016.11.25（1.0版本）

目录

iOS编程规范 1

0.1前言 3

0.2代码总体原则 3

0.3术语定义 3

1.Cocoa和Objective-c特性头文件引用 3

2.命名 9

3.注释 12

4.格式 13

## 0.1前言

为􏰀高产品代码质量,开发人员编写出简洁、可维护、可靠、可测试、高效 、 可移植的代码,、规范的解释以及参考材料(what &why)。列出了一些业界比较优秀的编程规范,作为延伸阅读参考材料。

## 0.2代码总体原则

1、清晰第一清晰性是易于维护 、易于重构的程序必需具备的特征 。代码首先是给人读的,好的代码应当可以像文章一样发声朗诵出来。一般情况下,代码的可阅读性高于性能,只有确定性能是瓶颈时,才应该主动优化。

2、简洁为美简洁就是易于理解并且易于实现。代码越长越难以看懂,也就越容易在修改时引入错误。写的代码越多,意味着出错的地方越多,也就意味着代码的可靠性越低 。因此,提倡大家通过编写简洁明了的代码来提升代码可靠性。废弃的代码(没有被调用的函数和全局变量)要及时清除,重复代码应该尽可能提炼成函数 。

3、选择合适的风格,与代码原有风格保持一致产品所有人共同分享同一种风格所带来的好处,远远超出为了统一而付出的代价。

## 0.3术语定义

原则:编程时必须坚持的指导思想。

规则:编程时强制必须遵守的约定。

建议:编程时必须加以考虑的约定。说明:对此原则/规则/建议进行必要的解释。

示例:对此原则/规则/建议从正、反两个方面给出例子。

### 1.Cocoa和Objective-c特性头文件引用

1.1规则:#import引用Objective-C/Objective-C++头文件; #include引用C/C++头文件

1.2规则:使用#include时要注意#define头保护。

1.3建议:#import头文件按模块分类。

说明:对头文件按模块分类,使代码易读,也便于理解。

1.4建议:#import的顺序,应该是先引用自定义类,再引用系统类/框架。

说明:将系统类放在最后引入,可避免和检验自定义类的头文件引用不完整 漏

洞

1.5建议:#import根框架而不是单独的零散文件

说明:当你试图从框架(如Cocoa或者Foundation)中包含若干零散的系统头

文件时,实际上包 含顶层根框架的话,编译器要做的工作更少。根框架通常已

经经过预编译,加载更快。

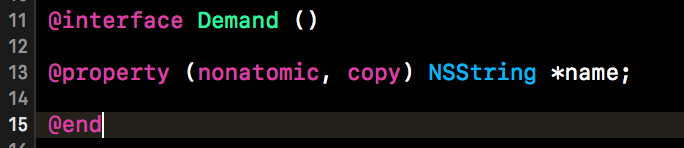
1.6建议:@class与#import的区别使用

说明:如需要继承类或执行协议,可以在.h中进行#import类或协议;其他情况下,在.h中声明用@classs声明此类即可。这样可以减少因头文件依赖引起重复编译,提高编译速度。

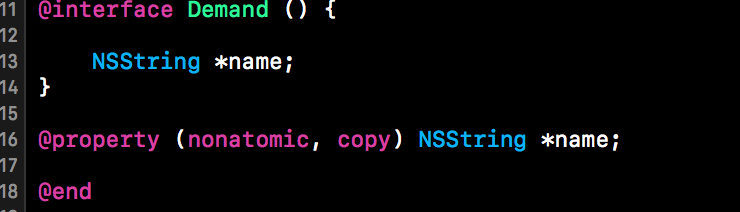
1.7规则:禁止对属性做出错误的内部成员变量声明

说明: xcode4.4开始会自动@synthesize ,为@ property声明一个前缀下划线的内部成员变量。因 此,不需要为属性重复做出内部成员变量声明。同时,禁止对属性做出错误的内部成员变量声明 。

示例：良好的风格：



错误的声明：



1.9建议:使用常类型变量,而不是内嵌的字符串字面值或数字

说明:常类型变量便于复用常用的变量值(如π),同时可以快速地修改值而无

需查找替换。

1.10建议:不用在init方法中,将成员变量初始化为0或者nil

说明:刚分配的对象,默认值都是0,除了isa指针(译者注: NSObject的isa识对象的类型)。所以不用在初始化器里面写一堆将成员初始化为0或者nil的代码。

1.12建议:字符串应使用copy属性(Attribute)

说明:应该用copy属性(attribute)声明NSString属性(property)。 从逻辑上,确保遵守NSString的setter必须使用copy而不是retain的原则。

1.13建议:谨慎声明属性的原子性

说明:一定要注意属性(property)的开销。缺省情况下,所有synthesize的setter和getter都是原子的。这会给每个get或者set带来一定的同步开销。将属性( property )声明为nonatomic,除非你需要原子性。

1.14建议:点引用只用于简单的属性set、get操作

说明:点引用只用于简单的属性set、get操作,但不应该用它来调用对象的其

它操作。

1.15建议:使用nil来检查应用程序的逻辑流程,而不是避免崩溃

说明:Objective-C运行时会处理向nil对象发送消息的情况。如果方法没有返回值,就没关系。 如果有返回值,可能由于运行时架构、返回值类型以及OS X版本的不同而不同,参见Apple’s documentation。注意,这和C/C++中检查指针是否为’NULL’很不一样,C/C++运行时不做任何检查,从而导致 应用程序崩溃。因此你仍然需要保证你不会对一个C/C++的空指针解引用

1.17建议:BOOL的陷阱,不要直接把整形转换成BOOL

说明:Ojbective-C中把BOOL定义成无符号字符型,不要直接把整形转换成BOOL。常见的错误 包括将数组的大小、指针值及位运算的结果直接转换成BOOL ,取决于整型结果的最后一个字节,很可能会产生一个NO值。当转换整形至BOOL时,使用三目操作符来返回YES或者NO。

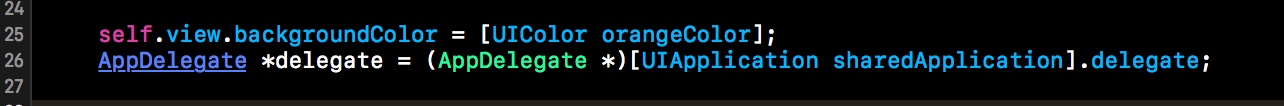
1.18建议:尽量使用NSInteger,而不是int

说明:当需要使用int类型的变量的时候,可以像写C的程序一样,用int,也可以用NSInteger,但 更推荐使用NSInteger,因为这样就不用考虑设备是32位的还是64位的.

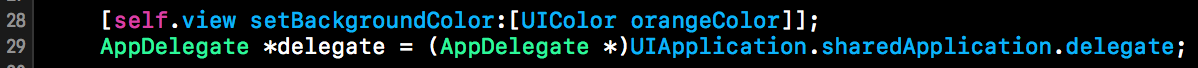
1.19点标记语法

说明:属性和幂等方法(多次调用和一次调用返回的结果相同)使用点标记语法访问,其他的情况 使用方括号标记语法。

示例：良好的风格



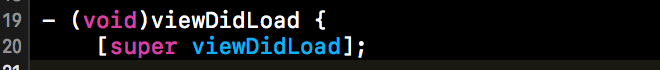
不良的风格



1.20规则:当子类需要使用init函数时,确保已经重载了父类的构造函数。

说明:当子类需要使用init函数时,确保已经重载了父类的构造函数 。否则,子类的初始化不会调 用,这会导致很多很难定位的bug。

示例：



1.21规则:出参与入参需对应

说明:在32位系统中,int和long都是32位的,但是在64位系统中,long则是64位的。将long赋值给int,在64位系统中,会出现truncate的隐患。

1.22建议:明确指定构造函数

说明:对于需要继承你的类的人来说,明确指定构造函数十分重要 。这样他们就可以只重写一个构 造函数(可能是几个)来保证他们的子类的构造函数会被调用 。这也有助于将来别人调试你的类时,理解初始化代码的工作流程。

1.23建议:避免重载+new

说明:不要调用NSObject的类方法new,或在子类中重载new方法。使用new方法会使关于内存分配 的代码审核变得很困难。使用alloc和init来创建并初始化对象。

1.24建议:保持公共API简单

说明:保持类简单;避免“厨房水槽( kitchen-sink)”式的API。如果一个函数压根没必要公开,就不要这么做。用私有类别保证公共头文件整洁。与C++不同, Objective-C没有方法来区分公共的方法和私有的方法–所有的方法都是公共的(译者注:这取决于Objective-C运行时的方法调用的消息机制)。因此,除非客户端的代码期望 使用某个方法,不要把这个方法放进公共API中。尽可能的避免了你不希望被调用的方法却被调 用到。这包括重载父类的方法 。对于内部实现所需要的方法,在实现的文件中定义一个类别,而不是把它们放进公有的头文件中。

1.25建议:大的@implementation用categories拆分成更容易理解的小块。

说明:便于理解的同时,还可以为最适合的类添加新的、特定应用程序的功能。例如,当添加一个“middle truncation”方法时,创建一个NSString的新类别并把方法放在里面,要比创建任意 的一个新类把方法放进里面好得多。

1.26规则:用宏定义表达式时,要使用完备的括号。

示例:如下定义的宏都存在一定的风险。

#define RECTANGLE\_AREA( a, b )

正确的定义应为:

#define RECTANGLE\_AREA ((a) \* (b))

1.27建议:委托模式

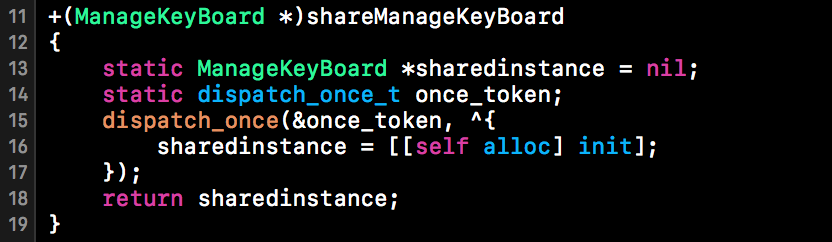
说明:委托对象不应该被retain实现委托模式的类应:1.拥有一个名为\_delegate的实例变量来引用委托。2.因此,访问器方法应该命名为delegate和setDelegate:。3.\_delegate对象不应该被retain。

1.28建议:模型/视图/控制器(MVC)

说明:分离模型与视图。分离控制器与视图、模型。•分离模型与视图:不要假设模型或者数据源的表示方法 。保持数据源与表示层之间的接口抽象。视图不需要了解模型的逻辑(主要的规则是问问你自己,对于数据源的一个实例,有没有可能有多种不同状态的表示方法)。•分离控制器与模型、视图:不要把所有的“业务逻辑”放进跟视图有关的类中。这使代码非常难以复用 。使用控制器类来处理这些代码,但保证控制器不需要了解太多表示层的 逻辑。

1.29建议:应该使用线程安全的模式创建共享的单例实例示例

示例：



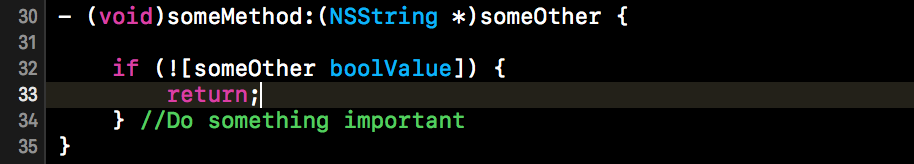
1.30建议:尽量使用不透明视图

说明:不透明的视图可以极大地􏰀高渲染的速度。因此如非必要,可以将tablecell及其子视图的opaque属性设为YES(默认值)。

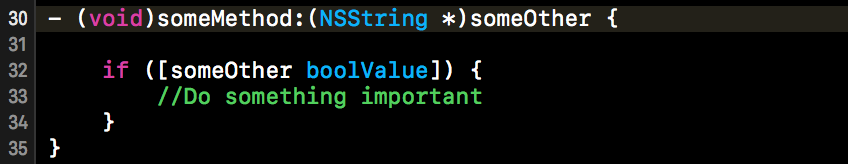
1.34建议:做条件判断时,应使用Golden Path模式

说明：在使用条件语句编程时，代码的左边距应该是一条“黄金”或者“快乐”的大道。 也就是说，不要嵌套 if语句。使用多个 return 可以避免增加循环的复杂度，并提高代码的可读性。因为方法的重要部分没有嵌套在分支里面，并且你可以很清楚地找到相关的代码

示例：推荐：



不推荐：



1.35建议:异常和错误处理

不要在流控制语句中使用异常( NSException )。异常仅用于表明程序员的错误。为了表明一个错误,使用NSError \*。 当一个方法通过引用返回一个错误参数,应该检测返回值的状态,而不是错误参数的状态。

### 2.命名

2.1建议:关于命名和命名风格

说明:对于易维护的代码而言,命名规则非常重要。Objective-C的方法名往往十分长,但代码块 读起来就像散文一样,不需要太多的代码注释。当编写纯粹的Objective-C代码时,我们基本遵守标准的Objective-C namingrules,这 些命名规则可能与C++风格指南中的大相径庭。例如, Google的C++风格指南中推荐使用下划线分隔的单词作为变量名,而(苹果的)风格指南则使用驼峰命名法,这在Objective-C社区中非 常普遍。命名可适当使用缩略词。但必须确保,这些缩略词是久经历史,认识性比较高的缩略词。如 下:缩略词。不要使用,你自己认为大家可能会理解的缩略词 。因为,跳出语言环境之后,这个缩 略词就变得难懂了。当编写Objective-C++代码时,事情就不这么简单了。许多项目需要实现跨平台的C++ API,并混合一些Objective-C、Cocoa代码,或者直接以C++为后端,前端用本地Cocoa代码。 这就导致了两种命名方式直接不统一。我们的解决方案是:编码风格取决于方法/函数以哪种语言 实现。如果在一个@implementation语句中,就使用Objective-C的风格。如果实现一个C++的类,就使用C++的风格。这样避免了一个函数里面实例变量和局部变量命名规则混乱,严重影响 可读性。

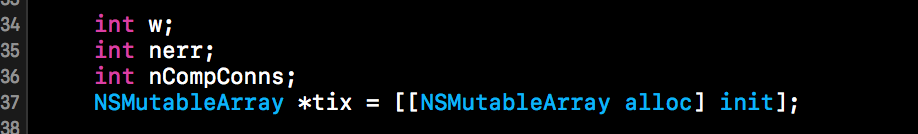
2.2建议:命名尽量简洁,但不能因为简洁而使命名难以理解

2.3建议:关于变量名

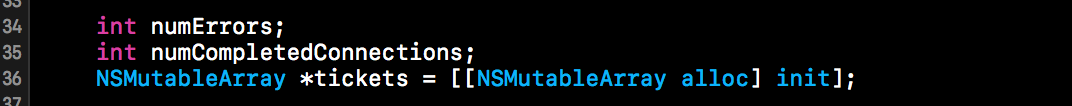
说明:变量名应该以小写字母开头,并使用驼峰格式 。尽量为变量起一个可述性的名字 。不要担心 浪费列宽,因为让新的代码阅读者立即理解你的代码更重要。

示例：

错误的命名:



正确的命名:



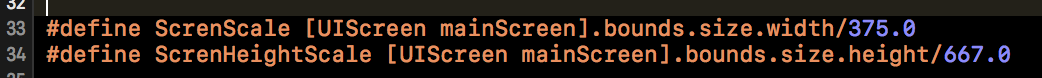
2.4建议:实例变量

说明:实例变量应该混合大小写,并以下划线作为前缀,如\_usernameTextField。然而,如果不能使用Objective-C 2.0(操作系统版本的限制),并且使用了KVO/KVC绑定成员变量时,我们允许例外(译者注: KVO=Key ValueObserving,KVC=Key Value Coding)。这种情况下,可以以一 个下划线作为成员变量名字的前缀,这是苹果所接受的键/值命名惯例。如果可以使用Objective-C 2.0,@property以及@synthesize􏰀供了遵从这一命名规则的解决方案。

2.5建议:常量

常量名(如宏定义、枚举、静态局部变量等)应该以小写字母k开头,使用驼峰格式分隔单词,

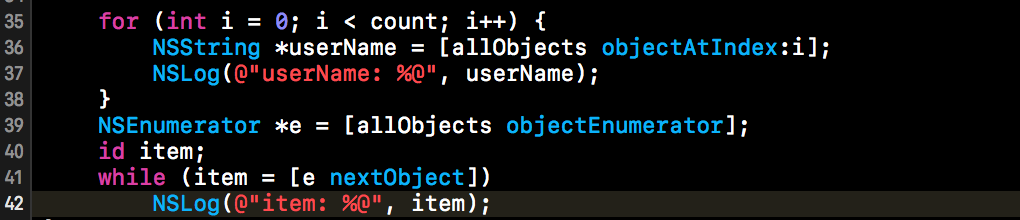
示例：



2.6建议:不要使用单个字符来定义变量名。

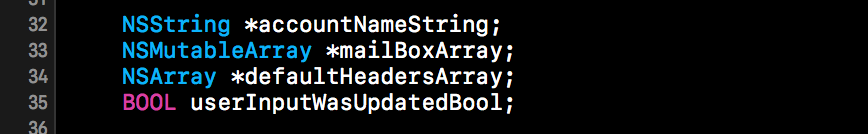
说明:即时变量只是一个index,为它取名index,而不是i、j、k。 一般循环语句

的当前对象的命名前缀包括“ one ”、“ a/an ”。对于简单的单个对象使用“ item ”命名。示例：良好的风格：

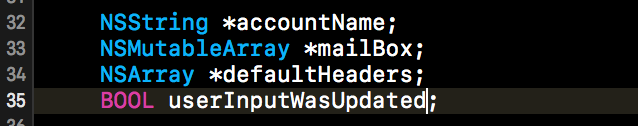


2.7建议:对于NSString、NSArray、NSNumber或BOOL类型,变量的命名一般不需要表明其 类型。

良好的风格:



不好的风格：



如果变量不是以上基本常用类型,则变量的命名就应该反映出自身的类型 。但有时仅需要某 些类的一个实例的情况下,那么只需要基于类名进行命名。示例：NSImage \*previewPaneImage ; NSProgressIndicator \*uploadIndicator ;NSFontManager \* fontManager ; //基于类名命名大部分情况下, NSArray或NSSet类型的变量只需要使用单词复数形式(比如mailboxes ),不必在命名中包含“ mutable ”。如果复数变量不是NSArray或NSSet类型,则 需要指定其类型。良好的风格:NSDictionary \* keyed AccountNames; NSDictionary \* messageDictionary ; NSIndexSet \* selectedMailboxes IndexSet ;

2.8规则:不要使用下划线前缀来声明私有方法。 说明:苹果保留这一风格。

2.9规则:方法的每个参数必须有参数声明,尽可能对参数做出描述,不能为空

2.10建议:关于类名

说明:类名(以及类别、协议名)应首字母大写,并以驼峰格式分割单词。应用层的代码,应该尽 量避免不必要的前缀 。为每个类都添加相同的前缀无助于可读性 。当编写的代码期望在不同应用程 序间复用时,应使用前缀(如:GTMSendMessage )。

2.11建议:关于类别名

说明: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 。类别名应 该包含它所扩展的类的名字。比如我们要基于NSString创建一个用于解析的类别,我们将把类别放在一个名为GTMNSString+Parsing.h的文件中。类别本身命名为GTMStringParsingAdditions (是的,我们 知道类别名和文件名不一样,但是这个文件中可能存在多个不同的与解析有关类别)。类别中的方 法应该以gtm\_myCategoryMethodOnAString:为前缀以避免命名冲突,因为Objective-C只有一 个名字空间 。如果代码不会分享出去,也不会运行在不同的地址空间中,方法名字就不那么重要了 。类名与包含类别名的括号之间,应该以一个空格分隔。

2.12建议:关于方法名

说明:一个方法的命名首先描述返回什么,接着是什么情况下被返回 。方法签名中冒号的前面描述传入参数的类型。方法名应该以小写字母开头,并混合驼峰格式。每个具名参数也应该以小写字母开头。 方法名应尽量读起来就像句子,这表示你应该选择与方法名连在一起读起来通顺的参数名。(例如,convertPoint:fromRect:或replaceCharactersInRange:withString:)。访问器方法应该与他们 要获取的成员变量的名字一样,但不应该以get作为前缀。

2.13建议:关于方法的参数名

说明:方法参数名一般使用的前缀包括“ the ”、“ an ”、“ new ”。

2.14建议:方法的参数名不要使用“and”做链接。

说明:当参数较多时,代码会显得冗余。有一个例外,当参数􏰁述的是两个不同的action时,可以 用“and”做链接。

2.15建议:框架类模块中,在类名和常类型变量名前添加一个由三个大写的字母组成的前缀(如RNC )。

说明:由于Objective-C不支持名字空间,为了防止出现命名空间的冲突,在类名和常类型变量 名前添加一个由三个大写的字母组成的前缀(如RNC ),对于Core Data实体名则可以忽略此规 则。如果你子类化了标准的Cocoa类,将前缀和父类名合并是一个很好的做法。如继承UITableView的类可命名为RNCTableView。

2.16建议:关于文件名

说明:文件名须反映出其实现了什么类–包括大小写。遵循你所参与项目的约定。 文件的扩展名应该如下:类别的文件名应该包含被扩展的类名,如: GTMNSString+Utils.h或``GTMNSTextView+Autocomplete.h``。

### 3.注释

3.1原则:文件/类/框架的注释尽量完整。变量/方法的注释尽量简洁。

说明:尽管注释很重要,但最好的代码应该自成文档。 并且注释的更新往往没有代码更新及时 。与 其写一段复杂的注释,不如直接起一个有意义的名字 。更新代码时一定要更新注释,防止对代码造 成误解。

3.2规则:文件注释

说明:每个文件的开头以文件内容的简要􏰁述起始,紧接着是作者,最后是版权声明和 样板。版权信息及作者每个文件应该按顺序包括如下项:/或许可证•文件内容的简要􏰁述•代码作者•版权信息声明(如:Copyright 2008 GoogleInc.)•必要的话,加上许可证样板。为项目选择一个合适的授权样板(例如,Apache 2.0, BSD, LGPL,GPL)。如果你对其他人的原始代码作出重大的修改,请把你自己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。

3.3建议:使用javadoc-style注释风格。

说明:注释的第一行是对注释API的总结,随后的注释行是对代码更多细节的解释。

3.4建议:声明部分的注释

说明:每个接口、类别以及协议应辅以注释,以描述它的目的及与整个项目的关系。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。 如果有的话,为类的线程安全性作注释。如果类的实例可以被多个线程访问,记得注释多线程条件下的使用规则。

3.5建议:实现部分的注释

说明: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。

3.6建议:程序中变量、方法命名尽量能以字面意思表示功能,对于需要用注释来解释的部分代码,注释以如下格式表述:

示例:/\*\*

\*方法或变量说明\* @param参数1说明(针对方法)\* @param参数2说明(针对方法)\* @return若方法有返回值则对返回值作说明

\*/

3.7建议:每一个方法之前都有一个99字符宽的注释行,注释行相对于使用空行更能􏰀高代码的 辨识度,当一行代码很长的时候,注释行也起到了越界检测的作用。

注释行: ///////////////////////////////////////////////////////////////////////////////////// //////////////

### 4.格式

4.1建议:尽量让你的代码保持在100列之内。

说明:我们深知Objective-C是一门繁冗的语言,在某些情况下略超100列可能有助于􏰀高可读 性,但这也只能是特例而已,不能成为开脱 。对于超长的代码,我们要怀疑代码逻辑的简练性。 设置方法: Xcode > Preferences > TextEditing > Show page guide。

4.2规则:指针变量的星号指示符应该紧靠变量。示例:NSString \*text,而不是

NSString\* text或NSString \* text。

4.3规则:方法声明中,-/ +和返回类型之间须使用一个空格。

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

4.4规则:运算符间距

说明:二元运算符和参数之间需要放置一个空格,一元运算符、强制类型转换和参数之间不放置空 格。关键字之后圆括号之前,需要放置一个空格。数组和字典类型的字面值的方括号两边 各放置一个空格。NSArray \*theShit = @[ @1 , @2, @3 ] ;字典字面值的键和冒号之间没有空格,冒号和值之间有一个空格。NSDictionary \*keyedShit = @{ GHDidCreateStyleGuide: @YES };C函数声明中,左括号的前面 不保留空格,并且函数名应该像类一样带有命名空间标识

4.5建议:只使用空格,不使用制表符。一次缩进两个空格。

说明:我们使用空格缩进,不在代码中使用制表符。在跨平台开发中,不同IDE的tab对应缩进格式 不同。应该将编辑器设置成自动将制表符替换成空格。

4.6建议:类型标识符和尖括号内的协议名之间,不能有任何空格。

这条建议适用于类声明、实例变量以及方法声明。

4.7建议:@public和@private访问修饰符应该以一个空格缩进。