|  |  |
| --- | --- |
| Document number 文档编号 | Confidentiality level 密级 |
|  | 内部公开 |
| Document version 文档版本 | Total pages 共 页 |
| V1.00 |

iOS编码规范

Swift版

|  |  |  |  |
| --- | --- | --- | --- |
| **Prepared by 拟制** |  | **Date 日期** |  |
| **Reviewed by 评审人** |  | **Date 日期** |  |
| **Approved by 批准** |  | **Date 日期** |  |

Revision Record 修订记录

| Date  日期 | Version  修订版本 | Sec No.  修改章节 | Description  修改描述 | Author  作者 |
| --- | --- | --- | --- | --- |
| 2015-03-18 | V1.0.0 |  | 文档初稿 | 许航宇 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Table of Contents 目录**

[1. 范围 4](#_Toc414460209)

[2. 规范性引用文件 4](#_Toc414460210)

[3. 术语和定义 4](#_Toc414460211)

[4. 版式规范 5](#_Toc414460212)

[4.1. 空格 5](#_Toc414460213)

[4.2. 代码行 6](#_Toc414460214)

[4.3. 空行 7](#_Toc414460215)

[4.4. 代码结构 8](#_Toc414460216)

[4.5. 分号 8](#_Toc414460217)

[5. 注释规范 9](#_Toc414460218)

[5.1. 通用 9](#_Toc414460219)

[5.2. 文件的注释 11](#_Toc414460220)

[5.3. 类和接口的注释 11](#_Toc414460221)

[5.4. 函数和方法的注释 12](#_Toc414460222)

[5.5. 变量和常量的注释 13](#_Toc414460223)

[5.6. 块注释 14](#_Toc414460224)

[6. 命名规范 16](#_Toc414460225)

[6.1. 通用 16](#_Toc414460226)

[6.2. 包命名 16](#_Toc414460227)

[6.3. 文件的命名 17](#_Toc414460228)

[6.4. 类和接口的命名 17](#_Toc414460229)

[6.5. 函数和方法的命名 18](#_Toc414460230)

[6.6. 变量和常量的命名 18](#_Toc414460231)

[6.7. 控件和资源文件的命名 20](#_Toc414460232)

[7. 备用知识点一 21](#_Toc414460233)

[7.1. 规则 21](#_Toc414460234)

[7.2. 建议 26](#_Toc414460235)

[8. 备用知识点二 29](#_Toc414460236)

[8.1. 规则 29](#_Toc414460237)

[8.2. 建议 31](#_Toc414460238)

# 范围

本规范规定了使用Swift语言编程时排版、注释、命名、编码和JTEST的规则和建议。

本规范适用于使用Swift语言编程的产品和项目。

# 规范性引用文件

下列文件中的条款通过本规范的引用而成为本规范的条款。凡是注日期的引用文件，其随后所有的修改单（不包括勘误的内容）或修订版均不适用于本规范，然而，鼓励根据本规范达成协议的各方研究是否可使用这些文件的最新版本。凡是不注日期的引用文件，其最新版本适用于本规范。

|  |  |  |
| --- | --- | --- |
| 序号 | 编号 | 名称 |
| 1 |  | 《php语言编程规范》 |
| 2 |  | 《android编程规范》 |

# 术语和定义

**【规则】：**编程时强制必须遵守的原则。

**【建议】：**编程时必须加以考虑的原则。

**【格式】：**对此规范格式的说明。

**【说明】：**对此规范或建议进行必要的解释。

**【示例】：**对此规范或建议从正、反两个方面给出例子。

# 版式规范

## 空格

### **【规则】**程序块要采用缩进风格编写，缩进的空格数为4个。

**【说明】**对于由开发工具自动生成的代码可以有不一致。

### **【规则】**对齐只使用空格键，不使用TAB键。

**【说明】**以免用不同的编辑器阅读程序时，因TAB键所设置的空格数目不同而造成程序布局不整齐。

### **【规则】**在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格；进行非对等操作时，如果是关系密切的立即操作符（如.），后不应加空格。

【说明】采用这种松散方式编写代码的目的是使代码更加清晰。

由于留空格所产生的清晰性是相对的，所以，在已经非常清晰的语句中没有必要再留空格，如果语句已足够清晰则括号内侧(即左括号后面和右括号前面)不需要加空格，多重括号间不必加空格，因为在Swift语言中括号已经是最清晰的标志了。

在长语句中，如果需要加的空格非常多，那么应该保持整体清晰，而在局部不加空格。给操作符留空格时不要连续留两个以上空格。

**【示例】**

1. 逗号、分号只在后面加空格。

timedAction(delay: 1.0, perform: someOtherAction)

1. 比较操作符, 赋值操作符"="、 "+="，算术操作符"+"、"%"，逻辑操作符"&&"，等双目操作符的前后加空格。

if (current\_time >= MAX\_TIME\_VALUE)

a = b + c;

a += 2;

a = b ^ 2;

1. "!"、"~"、"++"、"--"、"&"（地址运算符）等单目操作符前后不加空格。

flag = !isEmpty; // 非操作"!"与内容之间

i++; // "++","--"与内容之间

1. "."前后不加空格。

p.id = pid; // "."前后不加空格

1. if、for、while、switch等与后面的括号间应加空格，使if等关键字更为突出、明显。

if (a >= b && c > d)

## 代码行

### **【规则】** if、for、do、while、case、switch、default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{}。这样可以防止书写失误。

### **【规则】**对于“{}”来说，作”{”必须与父语句同行，不能另起一行。

**【示例】**如下例子不符合规范：

if (width < height) dosomething();

for initialization; condition; increment{dosomething();}

other();

应如下书写：

if (width < height) {

dosomething();

}

for initialization; condition; increment {

dosomething();

}

other();

### **【规则】**较长的语句、表达式或参数（>80字符）要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读。

【示例】

func reticulateSplines(spline: [Double], adjustmentFactor: Double,

translateConstant: Int, comment: String) -> Bool {

// reticulate code goes here

}

### **【规则】**请不要在一行中写上用分号隔开的多条语句（这条规则的唯一例外就是for-conditional-increment结构，该结构中分号是必需的。不过请尽量用for-in结构代替它们）。

**【示例】**如下例子不符合规范：

LogFilename now = null;LogFilename that = null;

应如下书写：

LogFilename now = null;

LogFilename that = null;

## 空行

### **【规则】**相对独立的程序块之间、变量说明之后必须加空行。

**【示例】**如下例子不符合规范：

if(log.getLevel() < LogConfig.getRecordLevel())

{

doSomething();

}

otherThing();

应如下书写：

if(log.getLevel() < LogConfig.getRecordLevel()) {

doSomething();;

}

otherThing();

## 代码结构

### **【建议】**类属性和类方法不要交叉放置，不同存取范围的属性或者方法也尽量不要交叉放置。

【格式】

类定义

{

类的公有属性定义

。。。

类的保护属性定义

。。。

类的私有属性定义

。。。

类的公有方法定义

。。。

类的保护方法定义

。。。

类的私有方法定义

。。。

}

## 分号

### **【规则】**Swift不要求每条语句后加分号，实际开发时应参照项目组其他成员的编码习惯，确定出是否加分号，以便后期维护。

# 注释规范

## 通用

### **【规则】**一般情况下，源程序有效注释量必须在20％以上。

**【说明】**注释的原则是有助于对程序的阅读理解，在该加的地方都加了，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。可以用注释统计工具来统计。

### **【规则】**边写代码边注释，修改代码同时修改相应的注释，保证注释与代码的一致性。不再有用的注释要及时删除。

### **【规则】**注释应当准确、易懂，防止注释有二义性。错误的注释不但无益反而有害。

### **【规则】**注释与所描述内容进行同样的缩排。

### **【示例】**如下例子不符合规范：

func example( ){

// 注释

CodeBlock One

// 注释

CodeBlock Two

}

应如下书写：

func example( ){

// 注释

CodeBlock One

// 注释

CodeBlock Two

}

### **【规则】**对变量的定义和分支语句（条件分支、循环语句等）必须编写注释。

**【说明】**这些语句往往是程序实现某一特定功能的关键，对于维护人员来说，良好的注释帮助更好的理解程序，有时甚至优于看设计文档。

### **【规则】**对于switch语句下的case语句，如果因为特殊情况需要处理完一个case后进入下一个case处理，必须在该case语句处理完、下一个case语句前加上明确的注释。

**【说明】**这样比较清楚程序编写者的意图，有效防止无故遗漏break语句。

### **【建议】**通过对函数或过程、变量、结构等正确的命名以及合理地组织代码的结构，使代码成为自注释的。

**【说明】**清晰准确的函数、变量等的命名，可增加代码可读性，并减少不必要的注释。

### **【建议】**在代码的功能、意图层次上进行注释，提供有用、额外的信息。

**【说明】**注释的目的是解释代码的目的、功能和采用的方法，提供代码以外的信息，帮助读者理解代码，防止没必要的重复注释信息。

**【示例】**如下注释意义不大：

// 如果 receiveFlag 为真

if (receiveFlag)

而如下的注释则给出了额外有用的信息：

// 如果从服务器收到消息

if (receiveFlag)

### **【建议】**注释应考虑程序易读及外观排版的因素，使用的语言若是中、英兼有的，建议多使用中文，除非能用非常流利准确的英文表达。

**【说明】**注释语言不统一，影响程序易读性和外观排版，出于维护的考虑，建议使用中文。

### **【建议】**注释尽量使用中文注释和中文标点。方法和类描述的第一句话尽量使用简洁明了的话概括一下功能，然后加以句号。接下来的部分可以详细描述。

## 文件的注释

多行注释建议使用VVDocumenter插件

### **【规则】**文件注释写入文件头部，包名之前的位置。

### **【规则】**文件注释内容：版权说明、描述信息、生成日期、修改历史。

### **【格式】**格式如下，可直接粘贴至模板中：

/\*\*(描述该文件开发的目的，技术选型，实现功能等)

\* ${tags}

\*

\* @name ${file\_name}

\* @author ${user} on ${date}

\* @version v1.0.0

\* @status Enable/Disable

\* @copyright Copyright(c) 2015,山东智慧生活数据系统有限公司. All rights reserved.

\*

\* @editor

\* name: ${user} on ${date}<br>

\* version: v1.0.1<br>

\* type: new/update/delete/repair<br>

\* content: (简述修改内容，具体的修改内容要在模块或者方法的注释中写清楚)

\*

\*/

**【说明】**每次修改后在文件头部写明修改信息，提交代码的时候可以直接把editor信息粘贴到注释上。

## 类和接口的注释

### **【规则】**类和接口的注释放在 import 关键字之后，class 关键字之前。

### **【规则】**类的注释主要是一句话功能简述、功能详细描述。

**【说明】**可根据需要列出：版本号、生成日期、作者、内容、功能、与其它类的关系等。 如果一个类存在Bug，请如实说明这些Bug。

### **【格式】**格式如下，可直接粘贴至模板中：

/\*\*(一句话简述功能流程)

\* (详细描述功能流程等)

\* ${tags}

\* @name ${file\_name}

\* @author ${user} on ${date}

\* @version v1.0.0

\* @status Enable/Disable

\* @see (相关类/方法)

\* @since (产品/模块版本)

\*

\*/

**【说明】**描述部分说明该类或者接口的功能、作用、使用方法和注意事项，每次修改后增加作者和更新版本号和日期，@since 表示从那个版本开始就有这个类或者接口，@deprecated 表示不建议使用该类或者接口。

## 函数和方法的注释

### **【规则】**函数和方法注释写在函数和方法上方。

### **【规则】**函数和方法注释内容包括功能简述、功能详细描述、输入参数、输出参数、返回值、异常、修改历史等。

### **【格式】**格式如下，可直接粘贴至模板中：

/\*\*(描述该方法的功能，实现方式，业务流程等)

\* ${tags}

\*

\* @name ${enclosing\_method}

\* @author ${user} on ${date}

\* @status Enable/Disable

\* @see (参考文档)

\*

\* @editor

\* name: ${user} on ${date}<br>

\* type: update/repair<br>

\* content: (修改内容)

\*

\*/

**【说明】**描述部分说明方法的功能、作用、使用方法和注意事项，每次修改后增加作者和日期，并详细标注修改内容。若编辑器不能自动生成参数及异常注释的话，需要手动填写。

## 变量和常量的注释

### **【规则】**变量和常量的注释内容：变量和常量的意义、目的、功能，可能被用到的地方。

### **【规则】**全局变量要有较详细的注释，包括对其功能、取值范围、哪些函数存取它以及存取时注意事项等的说明。

### **【建议】**最好能避免使用全局变量。

【示例】

var userName:NSString = “”;//用以保存用户名信息

var userToken:NSString = ””//用户token信息，从服务器获取，提交请求时需要用到

var DOWNLOAD\_STATUS:Int = 1;//文件下载的状态：1：正在下载；2：下载完毕

let SERVER\_IP:NSString = “192.168.100.161”;//服务器的IP地址，广泛用于提交请求

## 块注释

### **【规则】**注释的位置应与被描述的代码相邻，可以放在代码的上方或右方，但不可放在下方。如放于上方则需与其上面的代码用空行隔开。

### **【建议】**方法内的注释使用//。

### **【建议】**避免在一行代码或表达式的中间插入注释。

**【说明】**除非必要，不应在代码或表达中间插入注释，否则容易使代码可理解性变差。

### **【建议】**在程序块的结束行右方加注释标记，以表明某程序块的结束。

**【说明】**当代码段较长，特别是多重嵌套时，这样做可以使代码更清晰，更便于阅读。

**【示例】**参见如下例子：

if (...)

{

program code1

while (index < MAX\_INDEX)

{

program code2

} // end of while (index < MAX\_INDEX) // 指明该条while语句结束

} // end of if (...) // 指明是哪条if语句结束

### **【建议】**顺序实现流程的说明使用1、2、3、4在每个实现步骤部分的代码前面进行注释。

**【示例】**如下是对设置属性的流程注释：

//1、 判断输入参数是否有效。

。。。。。

/ 2、设置本地变量。

。。。。。。

# 命名规范

## 通用

### **【规则】**标示符命名可用驼峰命名法（除首单词外其余单词首字母均大写，例mPwd）、帕斯卡命名法（所有单词首字母均大写,例LoginViewController）、下划线命名法（单词间用下划线做间隔，例SERVER\_IP）和匈牙利命名法（以帕斯卡命名法的变量前附加小写序列说明该变量的类型，例label\_UserName）。

### **【规则】**只能有字母，数字，下划线”\_”,美元符号”$”,人民币符号”¥”组成，且首字母不能是数字。

### **【规则】**不能把关键字和保留字作为标识符（如String，int，class…）。

### **【建议】**含有集合意义的属性命名，尽量包含其复数的意义。

### **【建议】**用正确的反义词组命名具有互斥意义的变量或相反动作的函数等。

**【说明】**下面是一些在软件中常用的反义词组：

|  |  |  |  |
| --- | --- | --- | --- |
| add / remove | begin / end | create / destroy | insert/delete |
| first / last | get / set | open / close | min / max |
| send / receive | source / target | show / hide | up / down |
| visible / invisible | old / new | Initial/finalize | next / previous |

### **【建议】**含有集合意义的属性命名，尽量包含其复数的意义。

## 包命名

### **【规则】**包名若无特殊要求必须遵循 “com.itelife.[应用名]”的格式。

**【说明】**iOS代码结构中无严格意义的包名存在，此处包名指APP的Bundle ID。

### **【规则】**若是做内部环境测试生成的版本，则只对应用名进行标示。

### **【规则】**必须确保同一项目中各端的包名相同，在包名不统一的前提下，应当先确定出同一的包名；若包名不统一，不允许注册任何第三方平台。

## 文件的命名

### **【规则】**如果无特殊要求，文件的命名应与类和接口名一致。

### **【规则】**功能相近或者关联的文件应放到一个文件夹中，方便管理。

## 类和接口的命名

### **【规则】**类和接口的命名必须遵循帕斯卡命名法，分为前缀和后缀两部分。

**【说明】**前缀代表类或者接口的功能，后缀达标类或接口的类型。

### **【规则】**前缀命名如无特殊要求选用该类或接口功能的英文单词，原则上不超过3个单词。

### **【规则】**后缀命名根据类型决定，可参考下表：

|  |  |
| --- | --- |
| 类型 | 后缀 |
| ViewController类 | XXXViewController |
| 私有方法类 | XXXAction |
| 数据库交互类 | XXXDAO |
| 对象类 | XXXBean |
| 公共方法类 | XXXUtil或者XXXManager |
| 服务类 | XXXService |
| 自定义控件类 | XXX[控件名] |

### **【建议】**iOS端和Android端的类和接口名尽量保持一致。

## 函数和方法的命名

### **【规则】**方法的命名必须遵循驼峰命名法。

* + 1. **【规则】**方法中，存取属性的方法采用set和 get方法，动作方法采用动词和动宾结构。
    2. **【格式】**参考格式如下：

get + 非布尔属性名()

is + 布尔属性名()

set + 属性名()

动词()

动词 + 宾语()

【示例】

func getType();

func isFinished(translateConstant: Int) –> Bool;

func show();

func addKeyListener(Listener: String);

## 变量和常量的命名

### **【规则】**全局变量和常量名使用全大写的英文描述，英文单词之间用下划线分隔开，原则上不要超过3个单词。

### **【规则】**局部变量使用驼峰或者匈牙利命名法，原则上不要超过3个单词。

### **【规则】**属性名可以和公有方法参数相同，不能和局部变量相同，引用非静态成员变量时使用self引用，引用静态成员变量时使用类名引用。

### **【建议】**用统一的量词通过在结尾处放置一个量词，就可以创建更加统一的变量，他们更易于理解，也更容易搜索，如使用strUserFirst和strUserLast，而不要使用strFirstUser和strLastUser.。

### **【建议】**控件类型的变量名称最好与控件名相同。

**【示例】**

class Circle: Shape {

var x: Int, y: Int

var radius: Double

var diameter: Double {

get {

return radius \* 2

}

set {

radius = newValue / 2

}

}

init(x: Int, y: Int, radius: Double) {

self.x = x

self.y = y

self.radius = radius

}

convenience init(x: Int, y: Int, diameter: Double) {

self.init(x: x, y: y, radius: diameter / 2)

}

func describe() -> String {

return "I am a circle at (\(x),\(y)) with an area of \(computeArea())"

}

func computeArea() -> Double {

return M\_PI \* radius \* radius

}

## 控件和资源文件的命名

### **【规则】**控件和资源文件的命名使用匈牙利命名法，分为前缀、关系说明和后缀，格式为”[前缀]\_[关系说明]\_[后缀]”。

### **【规则】**前缀为控件和资源文件的名称缩写，如UIButton简写为btn、UITableView简写为tv、图片简写为img等。

### **【规则】**后缀为控件和资源文件功能的英文描述，原则上使用一个单词进行描述，如果内容较复杂可对后缀使用帕斯卡命名法。

### **【规则】**关系说明为该资源文件的父控件后缀，表明与父控件的层次关系，如果超过两层，则用匈牙利命名法继续添加。