Android平台Java编程规范

（文档编号 / IPD-CMMI V1.3 / 仅供内部使用）

|  |  |  |  |
| --- | --- | --- | --- |
| 拟制: | 吴晓红 00126135 | 日期： | 2013-8-12 |
| 审核: |  | 日期： |  |
| 批准: |  | 日期： |  |



Huawei Technologies Co., Ltd.

华为技术有限公司

All rights reserved

版权所有 侵权必究

Revision record 修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Date  日期 | Revision version  修订版本 | CR ID / Defect ID CR号 | Section Number 修改 章节 | Description  描述 | Author  作者 |
| 2013-8-10 |  |  |  | 初稿 | 吴晓红 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目录

[前言 5](#_Toc364327303)

[1 范围 6](#_Toc364327304)

[2 规范性引用文件 6](#_Toc364327305)

[3 术语和定义 6](#_Toc364327306)

[4 排版规范 6](#_Toc364327307)

[4.1. 规则 7](#_Toc364327308)

[4.1.1 \*缩进风格 7](#_Toc364327309)

[4.1.2 \*对齐尽量使用空格键，也可以使用TAB键。 7](#_Toc364327310)

[4.1.3 分界符 ‘{’和‘}’ 7](#_Toc364327311)

[4.1.4 \*较长的语句、表达式或参数 8](#_Toc364327312)

[4.1.5 文件长度 8](#_Toc364327313)

[4.1.6 \*不允许把多个短语句写在一行中 8](#_Toc364327314)

[4.1.7 \*if, for, do, while, case, switch, default 等语句自占一行， 9](#_Toc364327315)

[4.1.8 \*相对独立的程序块之间、变量说明之后必须加空行。 9](#_Toc364327316)

[4.1.9 \*关键字、变量、常量进行对等操作 10](#_Toc364327317)

[4.2 建议 10](#_Toc364327318)

[4.2.1 交叉放置 10](#_Toc364327319)

[5 注释规范 11](#_Toc364327320)

[5.1 规则 11](#_Toc364327321)

[5.1.1 注释量 11](#_Toc364327322)

[5.1.2 注释风格 11](#_Toc364327323)

[5.1.3 类和接口注释 11](#_Toc364327324)

[5.1.4 类属性、公有和保护方法注释 12](#_Toc364327325)

[5.1.5 异常抛出 14](#_Toc364327326)

[5.1.6 \*注释与代码放置 15](#_Toc364327327)

[5.1.7 \*注释与描述内容的缩排 15](#_Toc364327328)

[5.1.8 \* 将注释与其上面的代码用空行隔开 16](#_Toc364327329)

[5.1.9 \* 对变量的定义和**关键分支语句（条件分支、循环语句等）必须编写注释** 16](#_Toc364327330)

[5.1.10 \* switch语句下的case语句 16](#_Toc364327331)

[5.1.11 \*\*注释的内容 17](#_Toc364327332)

[5.1.12 注释语言 17](#_Toc364327333)

[5.1.13 删除代码的注释 17](#_Toc364327334)

[5.2 建议 18](#_Toc364327335)

[5.2.1 \*注释不应该出现在代码行中 18](#_Toc364327336)

[5.2.2 \*简洁明了。 18](#_Toc364327337)

[5.2.3 \*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。 18](#_Toc364327338)

[5.2.4 一些复杂的代码需要说明 19](#_Toc364327339)

[6 命名规范 19](#_Toc364327340)

[6.1 规则 19](#_Toc364327341)

[6.1.1 包名 19](#_Toc364327342)

[6.1.2 类名和接口名 20](#_Toc364327343)

[6.1.3 方法名 21](#_Toc364327344)

[6.1.4 方法get和set 21](#_Toc364327345)

[6.1.5 属性名 21](#_Toc364327346)

[6.1.6 常量名 22](#_Toc364327347)

[6.1.7 属性名约定 22](#_Toc364327348)

[6.1.8 函数的命名 22](#_Toc364327349)

[6.1.9 图片&xml文件的命名 23](#_Toc364327350)

[6.1.10 控件名 23](#_Toc364327351)

[6.2 建议 23](#_Toc364327352)

[6.2.1 常用组件类的命名以组件名加上组件类型名结尾。 23](#_Toc364327353)

[**6.2.2** **函数名过长的处理** 23](#_Toc364327354)

[6.2.3 准确地确定成员函数的存取控制符号 24](#_Toc364327355)

[6.2.4 含有集合意义的属性命名，尽量包含其复数的意义 24](#_Toc364327356)

[7 编码规范 24](#_Toc364327357)

[7.1 规则 24](#_Toc364327358)

[7.1.1 消除告警 24](#_Toc364327359)

[7.1.2 比较语句常量在前面 24](#_Toc364327360)

[7.1.3 函数参数合法性检查 24](#_Toc364327361)

[7.1.4 类功能明确，类实现精确 25](#_Toc364327362)

[7.1.5 数据库操作、IO操作等需要使用结束close()的对象必须在try -catch-finally 的finally中close()。 25](#_Toc364327363)

[7.1.6 异常捕获后，对不处理的该异常进纪录日志或者ex.printStackTrace() 。 26](#_Toc364327364)

[7.1.7 运行期异常RuntimeException 26](#_Toc364327365)

[7.1.8 \*运算符的优先级 27](#_Toc364327366)

[7.1.9 \*避免使用不易理解的数字，用有意义的标识来替代。 27](#_Toc364327367)

[7.1.10 数组声明的时候使用 int[] index ，而不要使用 int index[] 。 28](#_Toc364327368)

[7.1.11 代码的调试Log 28](#_Toc364327369)

[7.2 建议 29](#_Toc364327370)

[7.2.1 记录异常的显示 29](#_Toc364327371)

[7.2.2 一个方法不应抛出太多类型的异常 29](#_Toc364327372)

[7.2.3 集合数据管理应用及时 29](#_Toc364327373)

[7.2.4 \*源程序中关系较为紧密的代码应尽可能相邻。 29](#_Toc364327374)

[7.2.5 \*不要使用难懂的技巧性很高的语句，除非很有必要时。 29](#_Toc364327375)

前言

本规范是参考公司的《软件编程规范总则》，SUN公司《Java Coding Style Guide》和公司的《Java编程规范》在原有《U8220 Android编程规范》基础上修改形成《Android平台Java编程规范》。本规范没有涉及到的相关部分，请参见《软件编程规范总则》和公司的《Java编程规范》。

Android平台Java编程规范

# 范围

本规范规定了使用Android Java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Android平台 Java语言编程的部门和产品。

# 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 |  | 《软件编程规范总则》同总则的部分使用\*号 |

# 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。

# 排版规范

直接用Format.xml文件。  
注意， format只允许格式化自己模块的文件， 不允许格式化其它模块，避免格式化launch模块，提交代码只能提交自己模块。

## 规则

### \*缩进风格

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。使用范围包括java, xml 文件。

### \*对齐尽量使用空格键，也可以使用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。JBuilder、UltraEdit等编辑环境，支持行首TAB替换成空格，应将该选项打开。

如果要用tab，设置eclipse的tab为4个空格数。

### 分界符 ‘{’和‘}’

应与所修改文件或新建文件所在工程或模块保持一致。**分界符 ‘{’独占一行，和前面一个语句于不同一列**。分界符‘}’独占一行，引用它们的语句左对齐。在函数体的开始、类和接口的定义、以及if、for、do、while、switch、case语句中的程序都要采用如上的缩进方式。

示例：如下例子符合规范。

for (...)

{  
 ... // program code

}

不应如下书写。

for (...) {  
 ... // program code  
}

### \*较长的语句、表达式或参数

较长的语句、表达式或参数（**不超过80字符**）要分成多行书写，但不是只有超过80才分行，多逻辑表达式建议添加(),不是靠本身的优先级判断，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：

if ( (filename != null && filename.length() < LogConfig.getFileSize())

|| list !=null){

... // program code

}

意义相同的考虑上下对齐，如下参数，不受缩进规则影响

public static LogIterator read(String logType, Date startTime, Date endTime, int logLevel, String userName, int bufferNum)

方法里面的代码长度最大80行代码。

### 文件长度

Java类长度不要超过1000行。

### \*不允许把多个短语句写在一行中

不允许把多个短语句写在一行中，即一行只写一条语句示例：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;

### \*if, for, do, while, case, switch, default 等语句自占一行，

if, for, do, while, case, switch, default 等语句自占一行，且if, for, do, while等语句的执行语句无论多少都要加括号{}。

示例：如下例子不符合规范。  
if(writeToFile) writeFileThread.interrupt();

应如下书写：

if(writeToFile)

{  
 writeFileThread.interrupt();  
}

### \*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())  
 {

return;  
 }  
 LogWriter writer;

应如下书写：

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

return;  
 }

LogWriter writer;  
 int index;

### \*关键字、变量、常量进行对等操作

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Java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：

* 逗号、分号只在后面加空格。  
  int a, b, c;
* 比较操作符, 赋值操作符"="、 "+="， 算术操作符"+"、"%"， 逻辑操作符"&&"、

"&"，位域操作符"<<"、"^"等双目操作符的前后加空格。if ( MAX\_TIME\_VALUE <= current\_time ) a = b + c; a \*= 2; a = b ^ 2;

* "!"、"~"、"++"、"--"、"&"（ 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
* "."前后不加空格。  
  p.id = pid; // "."前后不加空格
* 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。

## 建议

### 交叉放置

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。

类格式定义：

{

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

}

# 注释规范

## 规则

### 注释量

源程序有效注释量控制在一定比例，不追求绝对数字，按需确定。

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。

注释应解释代码的意图，而不是描述代码怎么做的。

### 注释风格

单行注释用//，块注释用/\* \*/，JavaDoc注释用/\*\* \*/

### 类和接口注释

文件头的不添加版权注释，代码不开源。

类和接口注释放在 package 关键字之后，class 或者 interface 关键字之前。这样可以方便JavaDoc收集。示例如下：

package com.huawei.android.mms.ui;

/\*\*

\* 注释内容

\*/  
public class CommManager

类的注释主要是一句话功能描述，说明：可根据需要列出：生成日期、作者、内容、功能、与其它类的关系等。

格式如下：

/\*\*

\* @function: 功能描述

\* @author: [作者]

\* @version: [版本号, YYYY-MM-DD]

\*/

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。

示例：

/\*\*

\* function: 集中控制对日志读写的操作。

\* @author: 张三，李四，王五

\* @version: 2001-03-25

\*/

### 类属性、公有和保护方法注释

类属性、公有和保护方法注释写在类属性、公有和保护方法上面。示例：

/\*\*

\* 注释内容

\*/  
private String logType;

/\*\*

\* 注释内容

\*/  
public void write()

成员变量注释内容包括成员变量的意义、目的、功能，可能被用到的地方。 公有和保护方法注释内容：列出方法的一句话功能简述、**功能详细描述、输入参数、输出参数、返回值**、违例等。

格式：  
/\*\*

\* 〈功能描述〉

\* @param [参数1] [参数1说明]

\* @param [参数2] [参数2说明]

\* @return [返回类型说明]

\* @exception/throws [违例类型] [违例说明]

\*/

示例：

/\*\*

\* 根据日志类型和时间读取日志。

\* 分配对应日志类型的LogReader， 指定类型、查询时间段、条件和反复器缓冲数，

\* 读取日志记录。查询条件为null或0的表示没有限制，反复器缓冲数为0读不到日志。

\* 查询时间为左包含原则，即 [startTime, endTime) 。

\* @param logTypeName 日志类型名（在配置文件中定义的）

\* @param startTime 查询日志的开始时间

\* @param endTime 查询日志的结束时间

\* @param logLevel 查询日志的级别

\* @param userName 查询该用户的日志

\* @param bufferNum 日志反复器缓冲记录数

\* @return 结果集，日志反复器

\*/

public static LogIterator read(String logType, Date startTime, Date endTime, int logLevel, String userName, int bufferNum)

**排除：对于实体类的get, set方法， 不用加注释， 其它类的get, set等简单方法也可以考虑不加注释。**

### 异常抛出

对于方法内部用throw语句抛出的异常，必须在方法的注释中标明，对于所调用的其他方法所抛出的异常，选择主要的在注释中说明。对于非RuntimeException ，即throws子句声明会抛出的异常，必须在方法的注释中标明。

说明：异常注释用@exception或@throws表示，在JavaDoc中两者等价，但推荐用@exception标注Runtime 异常，@throws标注非Runtime 异常。**异常的注释**必须说明该异常的含义及什么条件下抛出该异常。

### \*注释与代码放置

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。

### \*注释与描述内容的缩排

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。示例：如下例子，排版不整齐，阅读稍感不方便。

public void example( ){

// 注释

CodeBlock One

// 注释  
 CodeBlock Two  
}  
应改为如下布局。

public void example( )

{  
 // 注释  
 CodeBlock One

// 注释  
CodeBlock Two

}

### \* 将注释与其上面的代码用空行隔开

示例：如下例子，显得代码过于紧凑。

//注释program code one

//注释program code two

应如下书写：  
//注释  
program code one

//注释  
program code two

### \* 对变量的定义和**关键分支语句（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。

### \* switch语句下的case语句

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。 如关键呼叫分支。

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。

**由于format.xml文件限制，switch 下面注释写在case后面， 如下**。

//进行联系人搜索

**switch** (range)

{

**case** *SEARCH\_RANGE\_ALL*: // 搜索所有

setSearchRangeIndicate(R.string.*dialpad\_all*);

**break**;

**case** *SEARCH\_RANGE\_FAVORITE*: //搜索爱好

setSearchRangeIndicate(R.string.*dialpad\_favorete\_number*);

**break**;

**default**:

**break**;

}

不允许如下

**switch** (range)

{

// 搜索所有

**case** *SEARCH\_RANGE\_ALL*:

setSearchRangeIndicate(R.string.*dialpad\_all*);

**break**;

//搜索爱好

**case** *SEARCH\_RANGE\_FAVORITE*:

setSearchRangeIndicate(R.string.*dialpad\_favorete\_number*);

**break**;

**default**:

**break**;

}

### \*注释的内容

注释的内容要清楚、明了，含义准确，防止注释二义性。说明：错误的注释不但无益反而有害。

### 注释语言

注释应该**使用中文（英语看项目组能力）和半角标点符号**，否则编译中会出现大量乱码，影响编译问题定位。

Eclipse工程用utf-8编码。

### 删除代码的注释

**不要用注释保留废弃代码**，现代的配置管理工具能恢复任意历史时刻的代码。

可以通过注释说明修改信息。

## 建议

### \*注释不应该出现在代码行中

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。

### \*简洁明了。

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。  
// 如果从连结收到消息  
if (receiveFlag)

### \*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

### 一些复杂的代码需要说明

示例：这里主要是对闰年算法的说明。

//1. 如果能被4整除，是闰年；

//2. 如果能被100整除，不是闰年；

//3. 如果能被400整除，是闰年。

# 命名规范

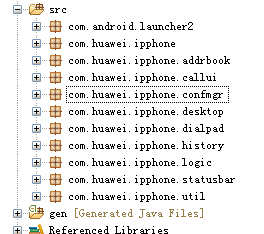
## 规则

### 包名

完全自己开发的包，包名采用域后缀倒置的加上自定义的包名，采用小写字母。包命名格式需要采取如下格式：com.huawei.ipphone.模块名

com.huawei.ipphone.模块名.小模块名

对于修改的代码，一律保持原先的包名不变化。



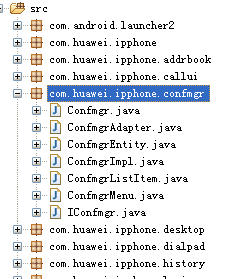
Setting - > apk

### 类名和接口名

使用类意义完整的英文描述，每个英文单词的首字母使用大写、其余字母使用小写的大小写混合法，尽量按模块名开头。示例：OrderInformation, CustomerList, LogManager, LogConfig, SmpTransaction。

对于具体业务类，有以下约定

|  |  |
| --- | --- |
| 类型 | 命名规定 |
| 模块的主体类，包含事件定义的类 | ＆模块名相同 |
| 接口 | I＋类名 |
| 实现类 | Impl结尾 |
| 适配类 | Adapter结尾 |
| 实体类 | Entity结尾 |
| 抽象类 | Abs开始 |



### 方法名

使用意义完整的英文描述：第一个单词的字母使用小写、剩余单词首字母大写其余字母小写的大小写混合法。

示例：  
private void calculateRate();  
public void addNewOrder();

### 方法get和set

存取属性的方法采用setter 和 getter方法，动作方法采用动词和动宾结构。

取消android默认的m前缀变量命名。

格式：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：

public String getType();

public boolean isFinished();

public void setVisible(boolean);

public void show();

public void addKeyListener(Listener);

### 属性名

属性名使用意义完整的英文描述：第一个单词必须是小写字母、剩余单词首字母大写其余字母小写的大小写混合法。属性名不能与方法名相同。命名规范示例：

Private String customerName;  
private String orderNumber;

循环变量统一用i , j , k 三个. 一重循环用i, 二重循环用j, 三重循环用k.

### 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，并且使用 final static 修饰

示例：

public final static int MENU\_MAX\_VALUE = 1000;  
public final static String DEFAULT\_START\_DATE = "2001-12-08";

### 属性名约定

属性名不可以和公有方法参数相同，如果相同，需要使用this来表示成员变量。属性名不能和局部变量相同。引用静态成员变量时使用类名引用。即使是在类内部使用，静态成员变量也需要使用类名进行引用。

示例如下：  
public class Person {  
 private String Name;  
 private static List PROPERTIES;

public void setName (String name){  
 Name = name;  
 }  
 public void setProperties (List properties){  
 Person. PROPERTIES = properties;  
 }  
}

### 函数的命名

函数的命名需要采用“动宾”结构，动词说明需要做的操作如Add，Delete。宾语表示动作作用的对象。如addContact，deleteMmsItem。

函数的命名最好能做到“见名思意”。这样可以提高代码的可读性。

函数实现功能需要单一，函数名字和函数实现的功能需要保持一致。如addContact。那么只是执行添加一个联系人信息到通讯录中。而没有必要执行更新界面的操作。

### 图片&xml文件的命名

统一按模块名字开头，进行分类，其中图片格式必须为png格式，图片， XML的存放路径分别放在唯一路径,如下。

图片： drawable-sw800dp-mdpi

XML: layout-sw800dp

Drawable: 按钮的点击文件定义以pressed结尾，默认是default结尾。

### 控件名

控件id 常用控件ID命名 前缀+对应的英文字义

TextView txt txtUsername

Button btn btnLogin

Imagebuton ibtn

ListView lv

EditText edt

ScrollView sv

Imageview iv

Gallery gallery

LinearLayout lt

RelativeLayout rl

AbsoluteLayout al

Tablelayout tl

framelayout fl

## 建议

### 常用组件类的命名以组件名加上组件类型名结尾。

示例：Application 类型的，命名以App 结尾——MainApp Frame 类型的，命名以Frame 结尾——TopoFrame Panel 类型的，建议命名以Panel 结尾——CreateCircuitPanel Bean 类型的，建议命名以Bean 结尾——DataAccessBean EJB 类型的，建议命名以EJB 结尾——DBProxyEJB Applet 类型的，建议命名以Applet 结尾——PictureShowApplet。

### **函数名过长的处理**

如果函数名**超过15 个字母**，可采用以去掉元音字母的方法或者以行业内约定俗成的缩写方式缩写函数名。示例：getCustomerInformation() 改为 getCustomerInfo()。

如简写对照

Information -> info, application -> app, windows -> wnd, description ->des

### 准确地确定成员函数的存取控制符号

不是必须使用 public 属性的，请使用 protected，不是必须使用 protected, 请使用 private。示例： protected void setUserName(),默认， private void calculateRate()。

### 含有集合意义的属性命名，尽量包含其复数的意义

示例：customers, orderItems。

# 编码规范

## 规则

### 消除告警

所有告警尽量清除。

### 比较语句常量在前面

对于if之类判断语句，==, != 运算符， 常量放在左边，如

If( MIN\_VALUE == value)

其它运算符， 变量放左边， 如

If( value >= MIN\_VALUE)

### 函数参数合法性检查

应明确规定对接口方法参数的合法性检查应由方法的调用者负责还是由接口方法本身负责，缺省是由方法调用者负责。

说明：对于模块间接口方法的参数的合法性检查这一问题，往往有两个极端现象，即：要么是调用者和被调用者对参数均不作合法性检查，结果就遗漏了合法性检查这一必要的处理过程，造成问题隐患；要么就是调用者和被调用者均对参数进行合法性检查，这种情况虽不会造成问题，但产生了冗余代码，降低了效率。

### Xml文件使用规定

Layout\_marginleft, layout\_marginright 用 Layout\_marginstart, Layout\_marginend代替， 目的解决阿拉伯语的对齐

Paddingleft, paddingright 也用paddingstart, paddingend代替。

### 类功能明确，类实现精确

明确类的功能，精确（而不是近似）地实现类的设计。一个类仅实现一组相近的功能。说明：划分类的时候，应该尽量把逻辑处理、数据和显示分离，实现类功能的单一性。示例：数据类不能包含数据处理的逻辑。通信类不能包含显示处理的逻辑。

### 数据库操作、IO操作等需要使用结束close()的对象必须在try -catch-finally 的finally中close()。

示例：

try  
{  
2002-1-9, 15:57:02   
// ... ...  
}  
catch(IOException ioe)  
{  
 //... ...  
}  
finally  
{  
 try  
 {  
 out.close();  
 }  
 catch (IOException ioe)  
 {  
 //... ...  
 }  
}

### 异常捕获后，对不处理的该异常进纪录日志或者ex.printStackTrace() 。

说明：若有特殊原因必须用注释加以说明。示例：

try  
{

//.... ...  
}

//asldjljkasdklsdf

catch (IOException ioe) //asldjljkasdklsdf  
{

//asldjljkasdklsdf  
 ioe.printStackTrace ();  
}

Catch(Throwable e){

}

### 运行期异常RuntimeException

运行期异常使用RuntimeException 的子类来表示，不用在可能抛出异常的方法声明上加throws子句。非运行期异常是从Exception继承而来的，必须在方法声明上加throws子句。

说明：非运行期异常是由外界运行环境决定异常抛出条件的异常，例如文件操作，可能受权限、磁盘空间大小的影响而失败，这种异常是程序本身无法避免的，需要调用者明确考虑该异常出现时该如何处理方法，因此非运行期异常必须有throws子句标出，不标出或者调用者不捕获该类型异常都会导致编译失败，从而防止程序员本身疏忽。运行期异常是程序在运行过程中本身考虑不周导致的异常，例如传入错误的参数等。抛出运行期异常的目的是防止异常扩散，导致定位困难。因此在做异常体系设计时要根据错误的性质合理选择自定义异常的继承关系。

所有异常都需要捕获，不能将原始信息直接抛给客户端用户呈现，以免信息安全。

### \*运算符的优先级

注意运算符的优先级，并用括号明确表达式的操作顺序，避免使用默认优先级。说明：防止阅读程序时产生误解，防止因默认的优先级与设计思想不符而导致程序出错。示例：

下列语句中的表达式

word = (high << 8) | low (1)

boolean changingPages = (whichPage != mCurrentPage); (2)

if ((a | b) < (c & d)) (3)

如果书写为

high << 8 | low

boolean changingPages = whichPage != mCurrentPage;

a | b < c & d

（1）（2）虽然不会出错，但语句不易理解；（3）造成了判断条件出错。

### \*避免使用不易理解的数字，用有意义的标识来替代。

涉及物理状态或者含有物理意义的常量，**不应直接使用魔鬼数字**，必须用有意义的静态变量来代替。

示例：如下的程序可读性差。

if (state == 1) {

state = 1;

... // program code

}

应改为如下形式：

private final static int TRUNK\_IDLE = 0;  
private final static int TRUNK\_BUSY = 1;  
private final static int TRUNK\_UNKNOWN = -1;

if (state == TRUNK\_IDLE){  
 state = TRUNK\_BUSY;  
 ... // program code

}

### 数组声明的时候使用 int[] index ，而不要使用 int index[] 。

说明：使用int index[] 格式使程序的可读性较差示例：如下程序可读性差：

public int getIndex()[]{  
 ....  
}

如下程序可读性好：

public int[] getIndex(){  
 ....  
}

### 代码的调试Log

调试代码的时候，不要使用 System.out 和 System.err 进行打印，应该使用一个包含统一开关的测试类进行统一打印。

说明：代码发布的时候可以统一关闭调试代码，定位问题的时候又可以打开开关。

|  |  |
| --- | --- |
| Log.error | 出错日志，影响后面方法运行 |
| Log.info | 正常日志 |
| Log.warn | 告警，但不影响方法运行 |
| Log.debug | 调试日志 |
| Log.operation | 操作日志 |

## 建议

### 记录异常的显示

记录异常不要保存exception.getMessage()，而要记录exception.toString()。示例：NullPointException抛出时常常描述为空，这样往往看不出是出了什么错。

### 一个方法不应抛出太多类型的异常

编码规范说明：如果程序中需要分类处理，则将异常根据分类组织成继承关系。如果确实有很多异常类型首先考虑用异常描述来区别，throws/exception子句标明的异常最好不要超过三个。

异常捕获尽量不要直接 catch (Exception ex) ，应该把异常细分处理。对于比较复杂的情况，建议捕获全部细分的异常后在末尾增加catch (Exception)，避免一些不易发生的运行时异常未被捕获。

### 集合数据管理应用及时

集合中的数据如果不使用了应该及时释放，尤其是可重复使用的集合。说明：由于集合保存了对象的句柄，虚拟机的垃圾收集器就不会回收。

### \*源程序中关系较为紧密的代码应尽可能相邻。

说明：便于程序阅读和查找。

示例：矩形的长与宽关系较密切，放在一起。

rect.length = 10;  
rect.width = 5;

### \*不要使用难懂的技巧性很高的语句，除非很有必要时。

说明：高技巧语句不等于高效率的程序，实际上程序的效率关键在于算法。