**Android编码规范**

# 概要

## 目的

本规范的目的是通过设定统一的编码基准，不仅可以为代码编写提供指导，而且可以保证其一贯性和易维护性，提高代码的可读性，减少编码中的缺陷隐患。

## 范围

本规范规定了使用Android Java语言编程时命名、排版、注释、编码的规则和建议。本规范适用于使用Android Java语言编程的产品。

## 总则

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。

# 命名规范

## 规则

### 包名

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.ect888.模块名.小模块名

示例：

IM包名 com.ect888.im

IM下的View包名 com.ect888.im.view

### 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。

示例：OrderInformation, CustomerList, LogManager, LogConfig, SmpTransaction。

### 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。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。

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。

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：第一个单词必须是控件的缩写、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 tvName;  
private edtPassword;

|  |  |
| --- | --- |
| 常用控件名称 | 缩写 |
| TextView | tv |
| EditText | edt |
| Button | btn |
| ImageButton | ib |
| ImageView | iv |
| CheckBox | cb |
| RadioButton | rb |
| ProgressBar | pb |
| SeekBar | sb |
| WebView | wv |
| ScollView | sv |
| ListView | lv |
| MapView | mv |

### 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 MAX\_VALUE = 1000;  
public final static String DEFAULT\_START\_DATE = "2001-12-08";

### layout 命名

layout xml 的命名必须以 全部单词小写，单词间以下划线分割，并且使用名词或名词词组，即使用 模块名\_功能名称 来命名。

示例： activity\_main.xml

fragment\_risk.xml

### id 命名

layout 中所使用的id必须以全部单词小写，单词间以下划线分割，并且使用名词或名词词组，即使用 模块名\_用途 来命名。并且要求能够通过id直接理解当前组件要实现的功能。

示例：@+id/book\_show

@+id/book\_edit

### 资源命名

layout中所使用的所有资源（如drawable,style等）命名必须以全部单词小写，单词间以下划线分割，并且尽可能的使用名词或名词组，即使用 模块名\_用途 来命名。

示例： menu\_icon\_nav.png

## 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：Activity 类型的，命名以Activity结尾——MainActivity Bean 类型的，建议命名以Bean 结尾——UserBean View 类型的，建议命名以View 结尾——SearchView。

### 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()。

### 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(), 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。

# 排版规范

## 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。

### 分界符 ‘{’和‘}’

分界符 ‘{’不独占一行，和前面一个语句于同一列。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：如下例子不符合规范。

for (...)

{  
 ... // program code

}  
if (...)

{  
 ... // program code  
 }

void example\_fun( void )

{  
 ... // program code  
 }

应如下书写。

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

void example\_fun( void ){  
 ... // program code  
}

### 较长的语句、表达式或参数

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：

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

if (filename != null

&& new File(logPath + filename).length() < LogConfig.getFileSize())

{

... // program code

}

### 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;

### 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。在编辑环境里支持行首TAB替换成空格，应将该选项打开。

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

一般情况下，源程序有效注释量必须在30％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。

### 文件注释

文件注释写入文件头部，包名之前的位置。文件名需要使用全路径名，而不是主文件名。说明：注意以 /\* 开始避免被 JavaDoc 收集示例：

/\*

\* 注释内容

\*/

### 文件注释内容

文件注释内容包括版权说明、描述信息、生成日期、修改历史。其中文件名是可选的。如果有文件名的话，文件名需要是全路径名而不是不带路径的文件名。提交到SVN上的时候需要提交log，内容和每次修改记录是一致的，只要再添加修改文件名。格式如下：

/\*

\* 文件名：[文件名]

\* 版权：〈版权〉

\* 描述：〈描述〉

\* 修改人：〈修改人〉

\* 修改时间：YYYY-MM-DD

\* 跟踪单号：〈跟踪单号〉

\* 修改单号：〈修改单号〉

\* 修改内容：〈修改内容〉  
\*/

示例：

/\*

\* 文件名：LogManager.java

\* 版权：Copyright 2000-2001 ECT Co. Ltd. All Rights Reserved.

\* 描述： WIN V200R002 WEBSMAP 通用日志系统

\* 修改人：张三

\* 修改时间：2001-02-16

\* 修改内容：新增

\* 修改人：李四

\* 修改时间：2001-02-26

\* 跟踪单号：D20103

\* 修改单号：WSS368

\* 修改内容：。。。。。。

\* 修改人：王五

\* 修改时间：2001-03-25

\* 跟踪单号：D27153

\* 修改单号：WSS498

\* 修改内容：。。。。。。

### 类和接口注释

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.ect888.im.view;

/\*\*

\* 注释内容

\*/  
public class CommManager

### 类和接口的注释内容

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。

格式如下：

/\*\*

\* 〈一句话功能简述〉

\* 〈功能详细描述〉

\* @author [作者]

\* @version [版本号, YYYY-MM-DD]

\* @see [相关类/方法]

\* @since [产品/模块版本]

\* @deprecated

\*/

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。

示例：

/\*\*

\* LogManager 类集中控制对日志读写的操作。

\* 全部为静态变量和静态方法，对外提供统一接口。分配对应日志类型的读写器，

\* 读取或写入符合条件的日志纪录。

\* @author 张三，李四，王五

\* @version 1.2, 2001-03-25

\* @see LogIteraotor

\* @see BasicLog

\* @since CommonLog1.0

\*/

### 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

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：  
/\*\*

\* 〈一句话功能简述〉

\* 〈功能详细描述〉

\* @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]

\* @see [类、类#方法、类#成员]

\* @deprecated

\*/  
说明：@since 表示从那个版本开始就有这个方法；@exception或throws 列出可能仍出的异常；@deprecated 表示不建议使用该方法。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

\* @since CommonLog1.0

\*/

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

### 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。

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。

### 注释的管理

边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。

### 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。

### 注释中的缩写

避免在注释中使用缩写，特别是不常用缩写。说明：在使用缩写时或之前，应对缩写进行必要的说明。

## 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。

### 代码行其实也是注释

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。

### 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。  
// 如果从连结收到消息  
if (receiveFlag)

### 方法内的单行注释使用 //

说明：调试程序的时候可以方便的使用 /\* 。。。\*/ 注释掉一长段程序。

### 注释应该简洁明了

注释使用英文注释英文标点。方法和类描述的第一句话尽量使用简洁明了语句概括功能，然后加以句号。接下来的部分可以详细描述。

说明：JavaDoc工具收集简介的时候使用选取第一句话。

### 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。

### 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。

# 编码规范

## 规则

### 明确方法功能，精确（而不是近似）地实现方法设计。

一个函数仅完成一件功能，即使简单功能也应该编写方法实现。说明：虽然为仅用一两行就可完成的功能去编方法好象没有必要，但用方法可使功能明确化，增加程序可读性，亦可方便维护、测试。

### 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。

### 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  
{

//.... ...  
}  
catch (IOException ioe)  
{  
 ioe.printStackTrace ();  
}

### 自己抛出的异常必须要填写详细的描述信息

说明：便于问题定位。示例：

throw new IOException("Writing data error! Data: " + data.toString());

### 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。

还有一种异常是Error 继承而来的，这种异常由虚拟机自己维护，表示发生了致命错误，程序无法继续运行例如内存不足。我们自己的程序不应该捕获这种异常，并且也不应该创建该种类型的异常。

### 异常处理和使用错误的取舍

在程序中使用异常处理还是使用错误返回码处理，根据是否有利于程序结构来确定，并且异常和错误码不应该混合使用，推荐使用异常。说明：一个系统或者模块应该统一规划异常类型和返回码的含义。但是不能用异常来做一般流程处理的方式，不要过多地使用异常，异常的处理效率比条件分支低，而且异常的跳转流程难以预测。

### 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)

if ((a | b) && (a & c)) (2)

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

如果书写为

high << 8 | low

a | b && a & c

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 == 0) {

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。

### DEBUG测试开关

用调测开关来切换软件的DEBUG版和正式版，而不要同时存在正式版本和DEBUG版本的不同源文件，以减少维护的难度。

### 魔鬼数字

魔鬼数字尽量不要出现在代码中，尽量使用静态的常量来定义魔鬼数字。这样可以提高代码的可维护性。

## 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。

### 重复代码段函数实现

如果多段代码重复做同一件事情，那么在方法的划分上可能存在问题。

说明：若此段代码各语句之间有实质性关联并且是完成同一件功能的，那么可考虑把此段代码构造成一个新的方法。

### 集合数据管理应用及时释放

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。