**蓝色互动**

**Android开发规范**

上海科匠信息科技有限公司

Android开发组编制

2014年3月1日

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修订内容 | 作者 | 时间 |
| 1.0.0 | 初稿 | 张一 | 2014-3-1 |
| 1.0.1 | 修定 删减部分繁琐细节 | 李国涛 | 2014-3-5 |
| 1.0.2 | 修定 删减部分繁琐细节 | 潘城 | 2014-3-5 |
| 1.0.3 | 添加版本号要求,添加对编码格式的要求 | 张一 | 2014-3-20 |
| 1.0.4 | 添加对注释有效行数的要求 | 张一 | 2014-4-1 |
| 1.0.5 | 板式调整 | 张一 | 2014-4-7 |
|  |  |  |  |
|  |  |  |  |

## 为什么需要编码注释规范?

编码规范对于程序员而言尤为重要，有以下几个原因：

* 一个软件的生命周期中，80%的花费在于维护
* 几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护
* 编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码
* 如果你将源码作为产品发布，就需要确任它是否被很好的打包并且清晰无误，一如你已构建的其它任何产品

## 命名

### 包命名

**规约：包命名必须以com.bm开始，后面跟有项目名称（或者缩写）,再后面为模块名或层级名称。**

如：com.bm.psb.adapter

com.bm.psb.activity

### 类和接口命名

**规约：类名必须使用驼峰规则，即首字母必须大写，如果为词组，则每个单词的首字 母也必须要大写,尽量使用词义明确的英文单词.**

如：class BookMarkAdd{}

### 方法的命名

**规约：方法名采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。**

例如： public void run();

public String getBookName();

类中常用方法的命名：

1. 类的获取方法（一般具有返回值）一般要求在被访问的字段名前加上get，如

getFirstName()，getLastName()。一般来说，get前缀方法返回的是单个值，find前缀的方法返回的是列表值。

1. 类的设置方法（一般返回类型为void）：被访问字段名的前面加上前缀 set，如

setFirstName(),setLastName().

1. 类的布尔型的判断方法一般要求方法名使用单词 is或has 做前缀，如isPersistent()，isString()。或者使用具有逻辑意义的单词，例如equal 或equals。
2. 类的普通方法一般采用完整的英文描述说明成员方法功能，第一个单词尽可能采用动词，首字母小写，如openFile（），addCount（）。
3. 构造方法应该用递增的方式写。（参数多的写在后面）。

### 变量命名

**规约：第一个单词的首字母小写，其后单词的首字母大写。尽量使用词义明确的英文单词.**

例如：String bookName;

### 成员变量命名

**规约：同变量命名，但不要在私有变量前添加m字样！**

### 常量命名

**规约：类常量的声明，应该全部大写，单词间用下划线隔开。**

例如：static final int SCREEN\_WIDTH = 480;

### layout 命名

**规约：layout xml 的命名必须以全部单词小写，单词间以下划线分割，以对应类缩写开头,并且使用名词或名词词组，即使用 类缩写\_模块名 来命名。**

如：ac\_knowledge\_gained\_home.xml

类缩写:

|  |  |
| --- | --- |
| Activity | ac |
| Fragment | fg |
| Dialog | dg |
| Popupwindow | pw |
| Adapter | item |
| 自定义View | v |
| 自定义layout | l |

### id 命名

**规约：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，使用名词或名词词组,以控件缩写开头,要求能够通过id直接理解当前组件要实现的功能。**

如：某TextView 为 @+id/tv\_book\_name\_show

常见控件缩写:

|  |  |
| --- | --- |
| LinearLayout | ll |
| RelativeLayout | rl |
| FrameLayout | fl |
| TableLayout | tl |
| SrollView | sv |
| Listview | lv |
| GridView | gv |
| ViewPager | vp |
| Textview | tv |
| EditText | et |
| Button | btn |
| Imageview | iv |
| ImageButton | ibtn |
| CheckBox | cb |
| ProgressBar | pb |
| RadioButton | rb |
| RadioGroup | rg |

### 资源命名

**规约：layout中所使用的所有资源（如drawable,string,color,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 用途\_模块名 来命名。如果为公共资源，直接用用途来命名**

如: btn\_back,bg\_app;

icon\_menu,

## 注释

Java 程序有两类注释：实现注释(implementation comments)和文档注释(document comments)。实现注释是使用/\*...\*/和//界定的注释。文档注释(被称为"doc comments") 由/\*\*...\*/界定。文档注释可以通过javadoc 工具转换成HTML 文件。

**规约: 单个类中,注释有效行数必须超过当前类总行数的20%**

### 1. 类注释

每一个类都要包含如下格式的注释，以说明当前类的功能等。

/\*\*

\* 类名

\* **@author** 作者 创建日期

\* 实现的主要功能。

\* **@author** 修改者，修改日期

\* 修改内容。

\*/

### 2. 方法注释

每一个方法都要包含 如下格式的注释 包括当前方法的用途，当前方法参数的含义，当前方法返回值的内容和抛出异常的列表。

/\*\*

\* 方法的一句话概述

\* **@param** s 说明参数含义

\* **@return** 说明返回值含义

\* **@throws** IOException 说明发生此异常的条件

\* **@throws** NullPointerException 说明发生此异常的条件

\*/

### 3. 类成员变量和常量注释

成员变量和常量需要使用java doc形式的注释，以说明当前变量或常量的含义

/\*\*

\* 常量含义

\*/

### 4. 方法内部注释

**规约:重要或复杂代码必须进行内部注释!**

如果需要多行使用/\*…… \*/形式，如果为单行是用//……形式的注释。不要再方法内部使用 java doc 形式的注释“/\*\*……\*\*/”.

### 5. XML注释

**规约：如果当前layout 或资源需要被多处调用，或为公共使用的layout（若list\_item），则需要在xml写明注释。要求注释清晰易懂。必须使用英文或拼音**

## 版本号

### 1. versionName采用a.b.c方式。

1. **修改bug和需求微动c加1;**
2. **需求变动b加1;**
3. **版本升级a加1.**

### 2. versionCode为versionName去掉半角句号之后的int型.

例如:初始versionName为1.0.0,则versionCode为101。

1. 修改bug或者需求微动后，versionName变为1.0.1,versionCode变为101；
2. 需求变动后，versionName变为1.1.1,versionCode变为111；
3. 版本升级后，versionName变为2.1.1,versionCode变为211；

## 编码格式

1. **编码格式强制为UTF-8.**
2. **第三方代码必须先转换成UTF-8格式后再放到工程中。**

## 代码风格(建议)

### 缩进

**规约：不允许使用Tab进行缩进，使用空格进行缩进，推荐缩进为2空格。**

### 空行

**规约：通常在 变量声明区域之后要用空行分隔，常量声明区域之后要有空行 分隔，方法声明之前要有空行分隔。**

空行将逻辑相关的代码段分隔开，以提高可读性。

下列情况应该总是使用空行：

* 一个源文件的两个片段(section)之间
* 类声明和接口声明之间
* 两个方法之间
* 方法内的局部变量和方法的第一条语句之间
* 一个方法内的两个逻辑段之间，用以提高可读性

### 行宽

无特别规定，因为现在的显示器都比较大，所以推荐使用120进行设置。

## 其他

### 方法

* 一个方法尽量不要超过15行，如果方法太长，说明当前方法业务逻辑已经非常复杂，那么就需要进行方法拆分，保证每个方法只作一件事。
* 不要使用 try catch 处理业务逻辑！

### 参数和返回值

* 一个方法的参数尽可能的不要超过4个！
* 如果一个方法返回的是一个错误码，请使用异常！！
* 尽可能不要使用null， 替代为异常 或者使用空变量 如返回 List 则可以使用Collections.emptyList()

### 神秘的数

**规约:代码中不允许出现单独的数字，字符！如果需要使用数字或字符，则将它们按照 含义 封装为静态常量！（for语句中除外）**

### 控制语句

* 判断中如有常量，则应将常量置于判断式的右侧。如：

if ( true == isAdmin()){

...

}

* 尽量不使用三目条件的嵌套。
* 所有if 语句必须用{}包括起来,即便是只有一句：

if (true){

...

}

* 对于循环：

将操作结构保存在临时变量里，减少方法调用次数

**final** **int** count = products.getCount();

**while**(index < count){

...

}

### 异常的捕捉处理

* 通常的思想是只对错误采用异常处理：逻辑和编程错误，设置错误，被破坏的数据，资源耗尽，等等。
* 通常的法则是系统在正常状态下以及无重载和硬件失效状态下，不应产生任何异常。
* 最小化从一个给定的抽象类中导出的异常的个数。对于经常发生的可预计事件不要采用异常。不要使用异常实现控制结构。
* 若有finally 子句，则不要在try 块中直接返回，亦不要在finally 中直接返回。