# Android代码规范

——合作终端组

## 为什么需要代码规范？

**[便于维护]**

@ 一个软件的生命周期中，80%的花费在于维护。

**[提高代码质量]**

@ 良好的编码习惯和规范的编码规约可以提高程序的鲁棒性。

**[团队协作]**

@ 风格统一，规范一致的代码有助于团队内成员快速上手，降低交接成本，阅读成本。

**[程序员个人能力]**

@ 是的，我们常常从一个程序员写的代码本身看出很多东西。

## 本文档结构

本文分为[**[命名规范]**](#_一、命名规范)**、**[**[代码风格]**](#_二、代码风格)和[**[编码规约]**](#_三、编码规约)和三个部分：

[**[命名规范]**](#_一、命名规范)

@ 规范代码中变量、常量、id以及文件名的命名方式，统一命名风格。

[**[代码风格]**](#_二、代码风格)

@ 规范代码的排版、编码、存储和注释等的风格，统一代码文件的风格。

[**[编码规约]**](#_三、编码规约)

@ 规范代码的实现，避免常见错误和编码陋习，优化代码实现方式。

## 参考链接

[[面向贡献者的 AOSP 代码样式指南](https://source.android.com/source/code-style)]

[[Google Java编程风格指南](http://www.hawstein.com/posts/google-java-style.html)]

[[阿里巴巴Java开发手册](https://github.com/alibaba/p3c)]

[[Android Java编码规范(adroplv)](http://km.oa.com/group/ep/articles/show/332047)]

[[阿里巴巴Android开发手册](http://www.wanandroid.com/blogimgs/a2609aed-1000-4039-93c3-7541aaa2013b.pdf)]

## 一、命名规范

**【使用场景】**编程过程中所有需要自定义名称的场合

**【目 的】**规范命名便于本人及团队其他成员理解代码，降低维护成本

**【说明】**

@ 大驼峰（UpperCamelCase）：每个词首字母大写，其他字母小写

@ 小驼峰（lowerCamelCase）：第一个词全部小写，其他词首字母大写，其他字母小写

**【正文】**

1 [通用]-[强制]

- 代码中命名不使用下划线或美元符开始和结尾。

- 代码中的命名禁止使用拼音与英文混合的方式。

- 代码中的命名拼写错误必须检查，避免拼写错误（专有名词/通用缩写除外）。









@ IDE会把拼写不正确的单词标波浪线，并提示“typo: In word”。每出现这种情况时，必须检查是否需要修正（1.添加下划线或者遵循驼峰规则，大写单词的第一个字母2.修正拼写错误）。

- 代码中命名不使用下划线或美元符开始和结尾。

- 代码中的命名禁止使用拼音与英文混合的方式。

- 代码中的命名拼写错误必须检查，避免拼写错误（专有名词/通用缩写除外）。

- 代码中的命名严禁使用不规范的缩写（望文不知义）。对英文缩写拿不准时使用完整单词，命名长好过难以理解。

2 [包名]-[强制]

- 包名统一使用小写，点分隔之前有且仅有一个自然语义单词，不使用下滑线。

@ 一级包名为顶级域名：com/edu/gov/net/org

@ 二级包名为公司/个人名

@ 常见分包：db/base/util/activity/adapter/manager/test/common/model等等

3 [类名]-[强制]

- 类名使用大驼峰（UpperCamelCase）风格。

4 [类名]-[建议]

- 类名命名时建议根据其功能使用通用的前后缀。

@ 示例

|  |  |
| --- | --- |
| 抽象类 | 使用 Abstract 或 Base 开头 |
| 异常类 | 使用 Exception 结尾 |
| 测试类 | 以它要测试的类名开始，以 Test 结尾 |
| 解析类 | Paser结尾 |
| 自定义共享基础类 | 以Base开头 |

- 使用到设计模式的类命名时，尽量体现出具体的模式，有助于快速理解。

@ 示例：XXXProxy，XXXFactory，XXXDelegate

- 类名通用缩写不需全部大写。

@ 示例：HtmlActivity、 XmlFileDecoder、HttpRequestSender。(Android代码风格，jdk中专有缩写全部大写)

5 [方法名]-[强制]

- 方法名统一使用小驼峰(lowerCamelCase)风格。

@ 方法名尽量能体现方法功能

@ 方法命名也有常用的前缀：init/is/check/handle/clear/remove… [建议]命名时根据方法的实际功能选择常用的前缀命名。

6 [变量]-[强制]

- 局部变量名统一使用小驼峰(lowerCamelCase)风格。

- 类成员变量名统一使用小驼峰(lowerCamelCase)风格，private变量以小写m开头。

- 静态字段名统一使用小驼峰(lowerCamelCase)风格，**[建议]**以小写s(private)/g(public)开头。

7 [常量]-[强制]

- 常量命名统一使用大写字母加下划线格式。\*\*\*

8 [控件ID]-[建议]

- xml文件内控件id的命名使用小写字母加下划线格式，建议使用“view缩写\_描述”的方式命名。

@ 常见控件缩写：

|  |  |
| --- | --- |
| LinearLayout | ll |
| RelativeLayout | rl |
| FrameLayout | fl |
| RecyclerView | rv |
| ListView | lv |
| GridView | gv |
| ImageView | igv |
| TextView | txv |
| Button | btn |

@ 示例：btn\_title\_back、iv\_donwload\_manager、fl\_content

9 [控件变量名]-[建议]

- 代码中控件变量的命名参照[13]，变量以控件类型的缩写结尾，大块背景控件以RootView或Layout作为变量名称或结尾词（ll,fl,rl不太好理解）

@ 示例：mNameTxv、mNumberEdt、mPhotoIgv、mRootView、mCtrlLayout、mAppIcon

10 [layout文件名]-[强制]

- 以小写字母加下划线命名。\*\*\*

@ Activity/Fragement/Dialog以activity\_/fragment\_/dialog\_作为前缀

@ ListView/RecyclerView/GridView的子项以list\_item/recycler\_item/grid\_item作为前缀

@ 自定义View以view\_作为前缀

11 [xml drawable文件名]-[强制]

- 使用小写字母加下划线命名方式，以”模块名称+drawable后缀”的形式命名。\*

@ 以\_selector/\_shape/\_layer\_list作为后缀

12 [图片资源]-[强制]

- 以小写字母加下划线命名。\*\*\*

@ 图标统一以icon\_作为前缀（icon\_back.png）

@ 背景图以bg\_作为前缀（bg\_default\_banner.png）

@ 普通通用图片pic\_作为前缀（pic\_red\_dot.png）

13 [anim文件名]-[强制]

- 以小写字母加下划线命名。\*\*\*

@ 以动画名称命名：fade\_in.xml、slip \_out.xml

14 [color/strings/dimens]-[强制]

- 以小写字母加下划线命名。\*\*\*

@ 使用模块名+描述的方式：

<color name=”home\_navigation\_btn\_tx\_color”>#112233</color>

<string name=”home\_navigation”>导航</ string>

<dimens name=”home\_navigation\_btn\_height”>56dp</ dimens>

15 [declare-styleable/style][强制]

- 命名方式：大驼峰（UpperCamelCase）风格。\*\*\*

@ 示例：

<style name="BigRatingBar" parent="@android:style/Widget.RatingBar">

<item name="android:progressDrawable">@drawable/layer\_big\_ratingbar</item>

<item name="android:minHeight">27dip</item>

</style>

<declare-styleable name="ExpandableTextView">

<attr name="maxCollapsedLines" format="integer"/>

<attr name="animDuration" format="integer"/>

<attr name="animAlphaStart" format="float"/>

<attr name="expandDrawable" format="reference"/>

<attr name="collapseDrawable" format="reference"/>

</declare-styleable>

每当不确定某种类型的命名方式时，参考Android源码中相应内容的命名方式。

## 二、代码风格

**【使用场景】**无关于功能实现的代码编写

**【目 的】**统一文件的编写方式，方便团队内成员阅读

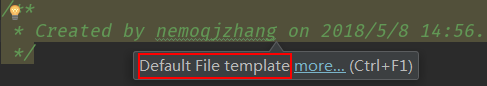
**【正文】**

1 [注释]-[强制]

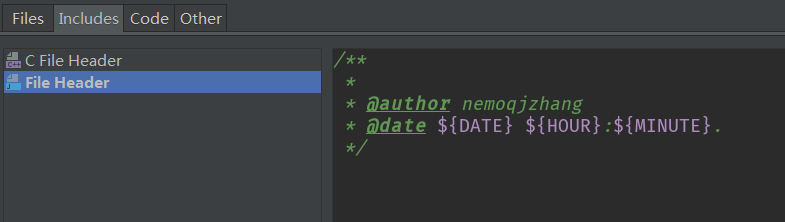
- 类、类成员、类方法的注释必须使用Javadoc规范，即“/\*\*内容\*/”格式，不使用“//内容”方式。\*\*\*

- 所有的类都必须添加创建者和创建日期。\*\*\*

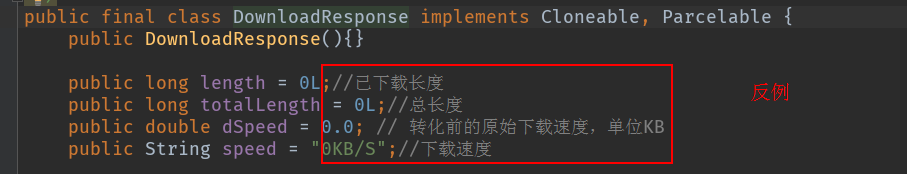
@ 默认的文件头注释IDE会提示格式问题：



@ 建议自定义文件头模板。File->Settings->Editor->File and Code Templates->Includes->File Header可以设置新建文件的注释模板。



- 不使用行尾注释。



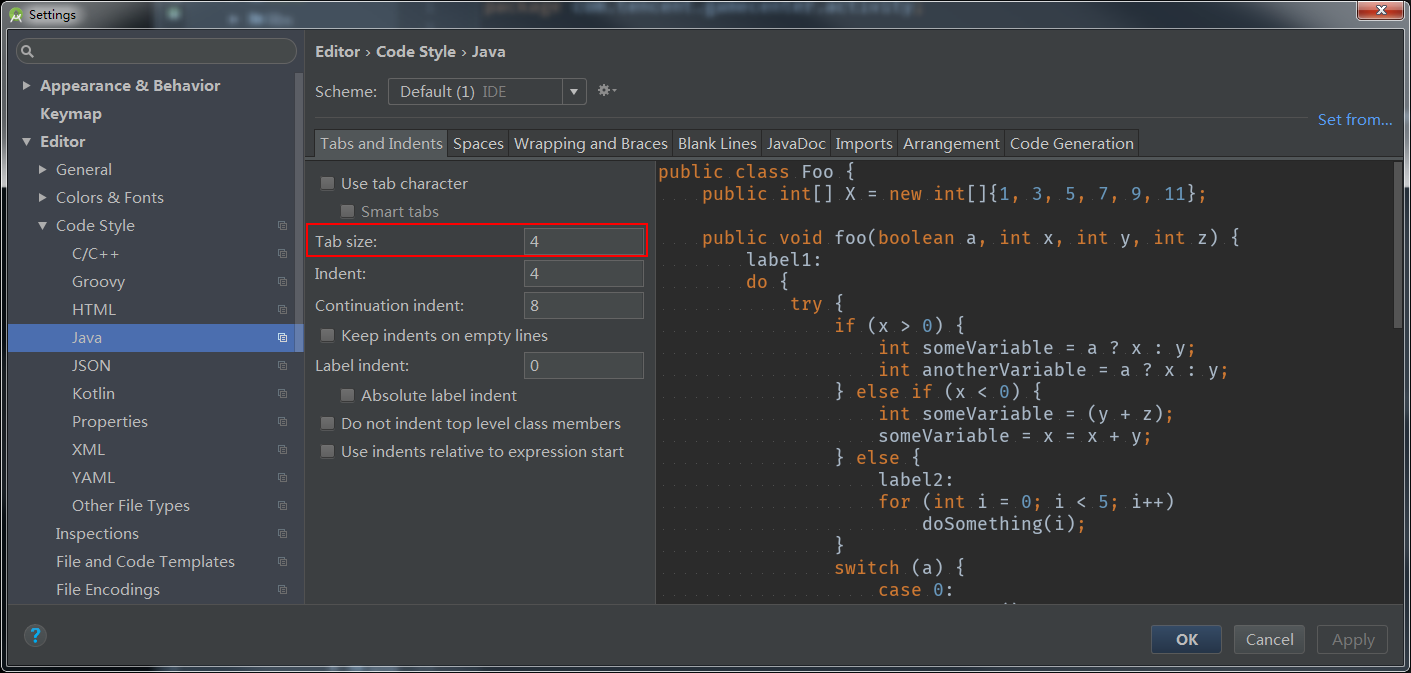
2 [注释]-[建议]

- 注释掉的代码，要么说明保留的原因，要么删除掉（历史代码有版本控制工具可以保存）。

- 抽象方法、接口中的方法须添加注释说明（参数、返回值、完成的功能等）。

3 [缩进]-[建议]

- 使用4个空格作为缩进，建议把tab设置为4空格大小



@ 关于缩进应该使用tab还是space的问题，普遍观点是：都行，关键在于统一。

@ 据说使用空格缩进的薪酬普遍更高…：<https://www.zhihu.com/question/19960028>

4 [大括号]-[强制]

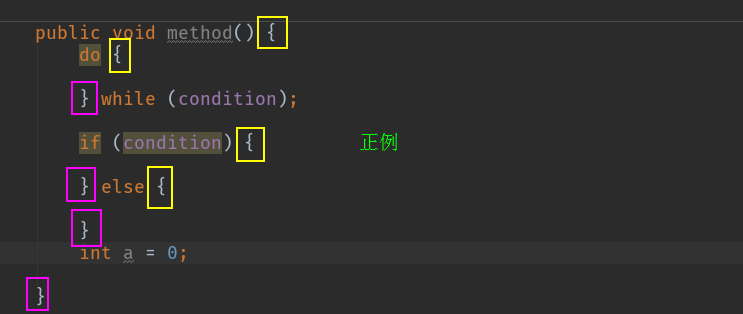
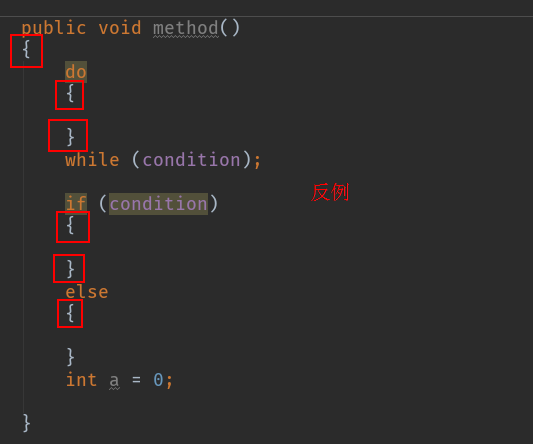
- 大括号内容非空时：

@ 左大括号前不换行

@ 左大括号后换行

@ 右大括号前换行

@ 右大括号后还有else/while等代码时不换行，表示块终止时必须换行

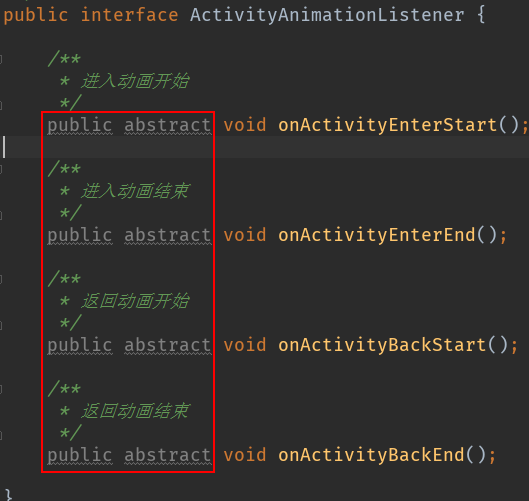
5 [大括号]-[建议]

- 大括号内容为空简写为“{ }”

6 [接口]-[强制]

- 接口内的变量、常量、方法不添加任何修饰符。\*\*

@ 接口中所有的方法都是public && abstract；所有的成员都是public && final && static。接口定义时没必要再写多余的修饰符，保持代码简洁。



**7** [方法摆放][建议]

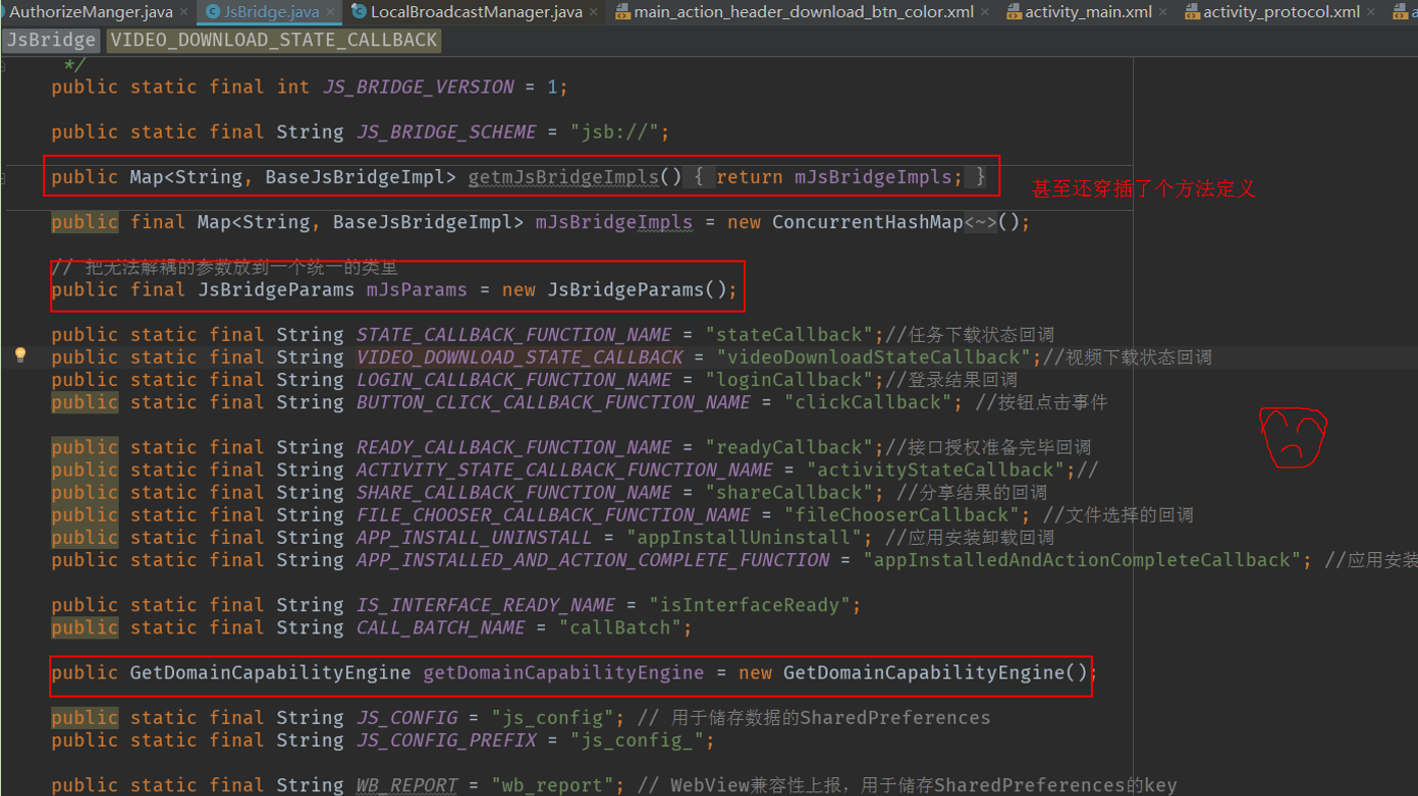
-类内方法摆放顺序，从上至下依次按照：public/protected > private>getter/setter的顺序摆放。

**8** [方法摆放][强制]

**-** 类内同名方法放在一起（优先级高于[7]）。\*\*\*

**9** [变量摆放][建议]

- 类内变量按照：常量>静态变量>成员变量分块放置，同时每个块内按照：public/protected > private的顺序放置，便于阅读。



**10** [文件编码][强制]

**-** 工程内文本文件编码方式统一设置为UTF-8。

**11** [换行][建议]

- 每一行文本的长度都应该不超过 100 个字符（超过IDE的行宽提示线，就可以考虑换行）

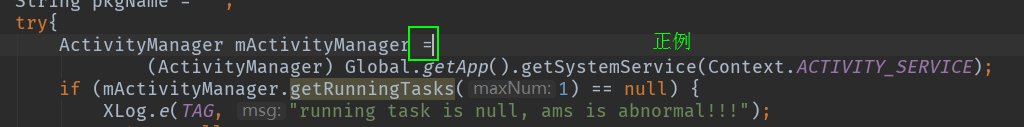
**-** 换行时的断开规则

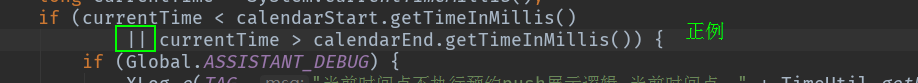
@ 在非赋值运算符处换行，建议在该符号前断开(比如+，它将位于下一行)

@ 在赋值运算符处换行，建议在该符号后断开(比如=，它与前面的内容留在同一行)

@ 逗号、foreach语句中的分号同赋值运算符

@ 大括号换行规则参见[4]





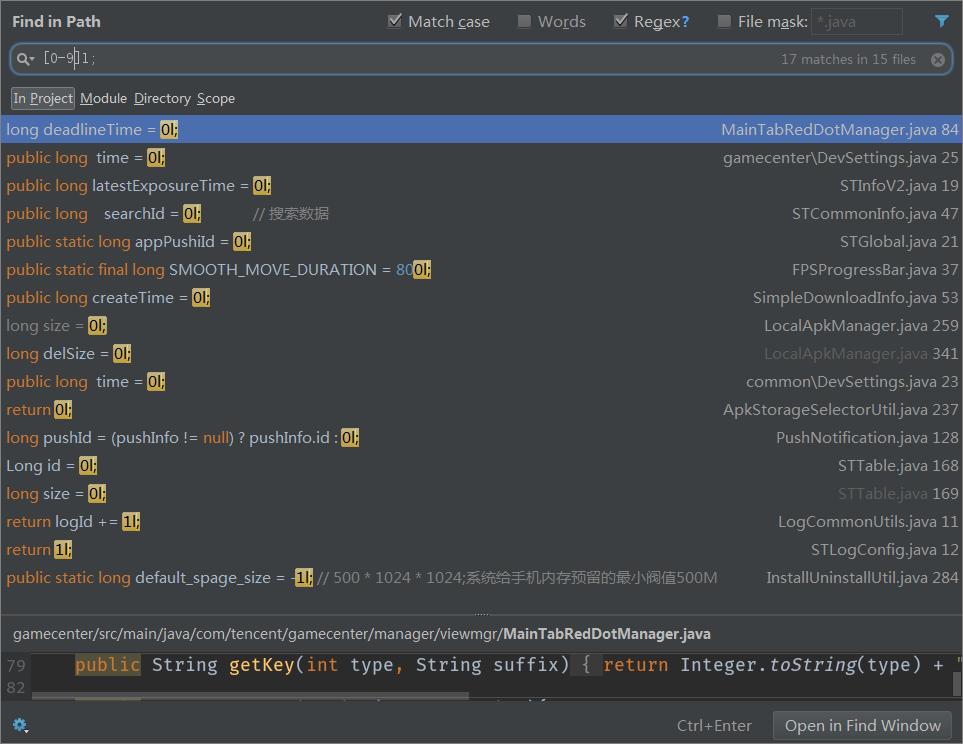
**-** 布局文件中控件标签名后统一换行：





12 [Long][强制]

- long或者Long初始赋值时，使用大写的L，不能是小写的l。\*



13 [数组类型][强制]

- 定义数组类型时中括号是类型的一部分，与类名放在一起：String[] args 而非 String args[]。

14 [日志打印][建议]

- 类打印日志定义private final static String TAG，并使用类名（适当结合前缀）的字符串常量命名。

@ 正式版本不要打印包含敏感信息的日志

@ 普通日志打印使用i ，错误日志使用e

@ 同一功能模块的日志TAG使用相同的前缀，方便过滤该模块所有日志。

15 [代码格式化][强制]

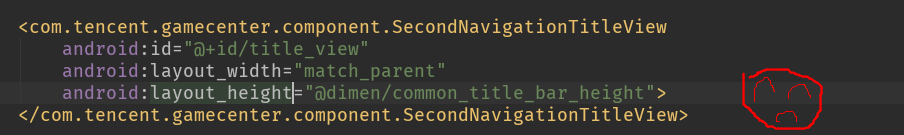
- 新代码提交前需要格式化（Formatter）代码。

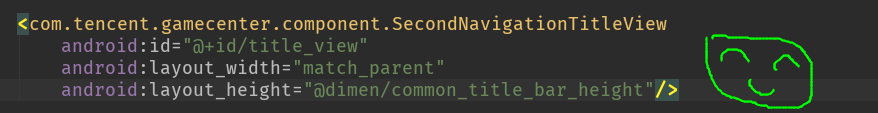
@ 格式化代码（Ctrl+ A && Ctrl+Shift+F）||（Ctrl+ A && Ctrl+Alt+L）

@ 清理导包（Ctrl+ Shift + O）

16 [布局控件][建议]

- 非ViewGroup控件定义时建议使用<TextView />格式，不建议写成<TextView ></TextView>





## 三、编码规约

**【使用场景】**代码实现。

**【目 的】**规范代码实现，减少程序出错的可能，提高代码健壮性。

**【Java篇】**

1 [finalize][强制]

**-** 不要重载finalize方法。

2 [finally ][强制]

**-** 不要在 finally 块中使用 return。

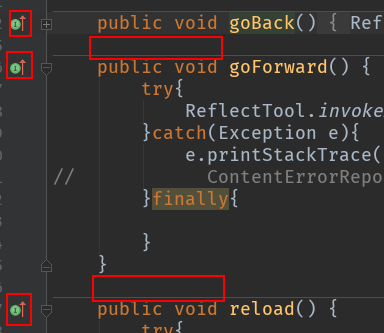
3 [switch][建议]

- switch语句最好包含一个default，即使什么也不做。

4 [复写][强制]

**-** 复写父类方法一定要使用@Override标注。

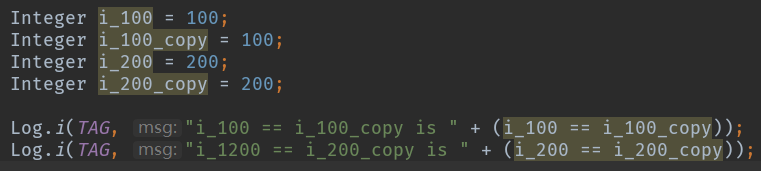
@ 标明Override有助于读者快速了解方法所有信息。



5 [值比较][强制]

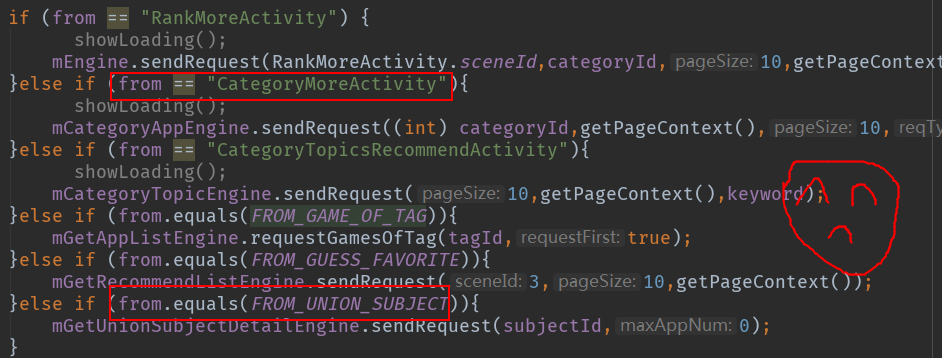
**-** 非基本类型的值比较统一使用equals。

@ 你清楚下边这两个比较的结果吗？



**-** equals方法尽量用确认有值的对象调用。\*\*

@ 使用"1".equals(selfLink)而不是selfLink.equals("1")。



6 [字符串拼接][强制]

**-** 循环体内字符串拼接使用StringBuilder的append（并发场景用StringBuffer）。\*

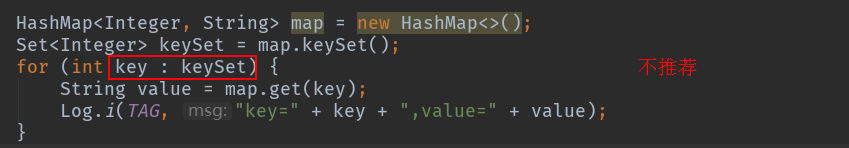
@ 用javap 查看字节码可以发现，字符串的“+”操作最终会被优化为StringBuilder对象的append，而在循环体内字符串的“+”操作意味着StringBuilder对象的反复创建，影响性能。

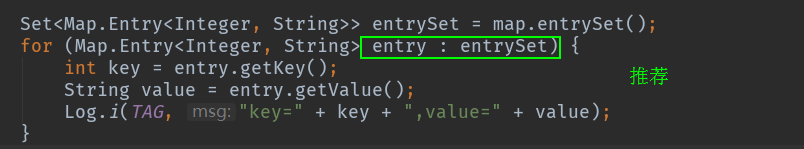
- “+”和append的详细分析参考:<https://segmentfault.com/a/1190000007099818>。



7 [Map][建议]

- 使用entrySet遍历Map，而不是 keySet 方式进行遍历。





@ keySet 其实是遍历了 2 次，一次是转为 Iterator 对象，另一次是从 hashMap 中取出key 所对应的 value。而 entrySet 只是遍历了一次就把 key 和 value 都放到了 entry 中，效率更高。

**-** 以Integer作为Map的key时，数据量不大的情况下建议使用SparseArray代替HashMap。

@ SparseArray 仅能存储以int值为key的数据。内部把数据按key的大小存储，同时对数据的删除和存储机制有优化，使得它内存占用更小，性能更好。

@ 详细分析：<https://juejin.im/entry/57c3e8c48ac24700634bd3cf>

8 [线程创建][强制]

- 线程创建时必须指定有意义的线程名。

@ 方便出问题时找到它的主人，便于快速定位。

9 [线程创建][建议]

- 尽量避免直接创建线程，建议使用线程池操作线程。

@ 频繁使用子线程的场景一定要使用线程池。控制线程创建开销，统一管理子线程。

10 [加锁][强制]

- 对多个资源、数据库表、对象同时加锁时，需要保持一致的加锁顺序。\*\*

@ 线程一获取ABC锁之后才能更新数据，线程二加锁顺序也必须是ABC，否则可能出现死锁。

11 [异常处理][强制]

- Java类库中定义的可以通过预检查方式规避的RuntimeException异常不应该通过catch的方式来处理，比如：NullPointerException， IndexOutOfBoundsException 等等。

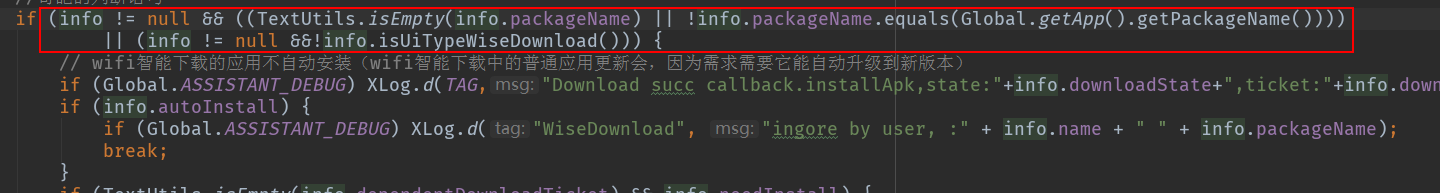
@ 无法通过预检查的异常除外。比如，在解析字符串形式的数字时，不得不通过 catch NumberFormatException来实现。

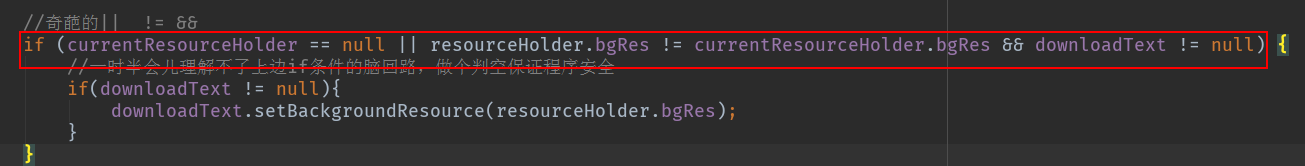
**-** 捕获异常是为了处理它，不要捕获了却什么都不处理。catch尽可能区分异常类型，再做对应的处理。

12 [编码习惯][强制]

- 异常不要用来做流程/条件控制。

- 判断语句的条件如果逻辑很复杂，需要适当分组拆分，并注释说明。





13 [编码习惯][建议]

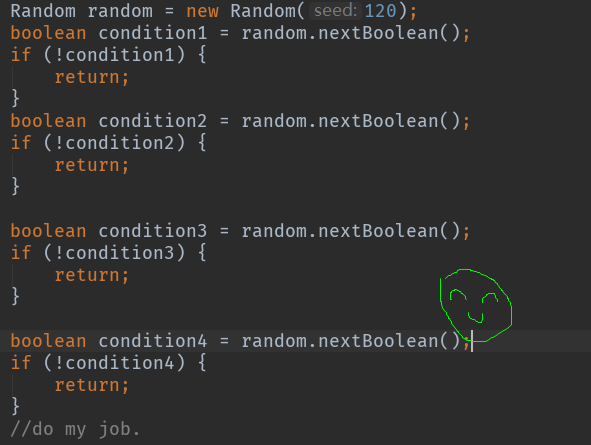
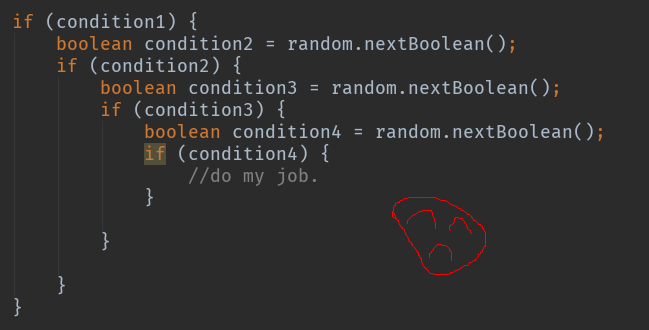
- 避免通过一个类的对象引用访问此类的静态变量或静态方法。

**-** 不要使用组合声明，比如int a, b。

**-** 编写Comparator时需处理相等的情况。

@ 不处理相等的情况时，使用该comparator比较时会出现a > b 和 b > a同时成立的情况。

- 表达异常的分支时，少用 if-else ，这种方式可以改写成卫语句，或者根据实际场景选择策略模式、状态模式等优化代码结构）。



**【Android 篇】**

1 [组件生命周期][强制]

- 避免在UI线程执行耗时操作。

- Activity或者Fragment中动态注册BroadCastReceiver时，registerReceiver()和unregisterReceiver()要成对出现。

- Activity的onPause方法中不要进行耗时操作。页面跳转时，当前Activity的onPause执行完之后才会执行下个Activity的onCreate，过于耗时会影响到跳转时的效率。

2 [组件生命周期][建议]

- 不建议在Activity#onDestroy()内执行释放资源的工作，例如一些工作线程的销毁和停止，因为 onDestroy()执行的时机可能较晚。可根据实际需要，在 Activity#onPause()/onStop()中结合isFinishing()的判断来执行。

3 [UI布局][强制]

- 不使用 ScrollView 包裹 RecyclerView、ListView、GridView、ExpandableListVIew。

@ 存在各种滑动上的冲突，需要增加滑动处理成本

@ SrollView为了计算子控件的高度（宽度）需要把列表控件的所有 Item 都绘制出来，会消耗巨大的内存和cpu，影响性能

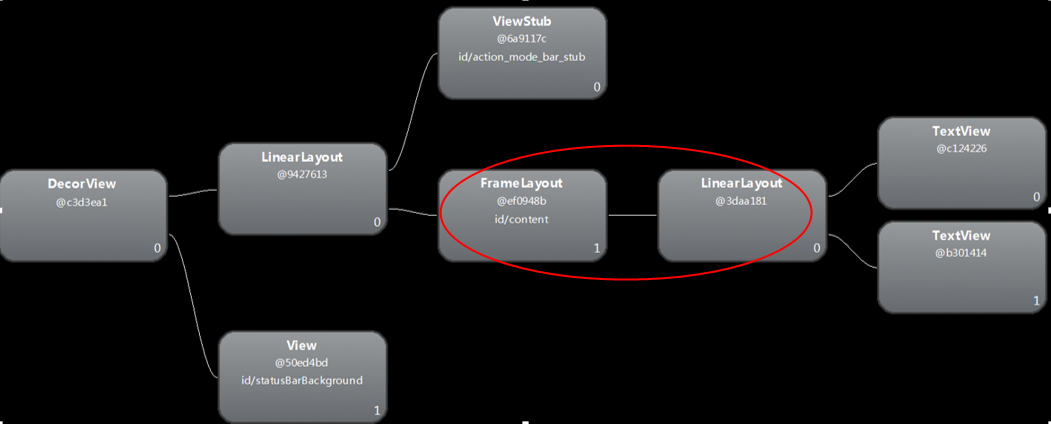
4 [UI布局][建议]

- 文本大小使用 dp单位，view 大小使用 dp单位。

**-** TextView在文字大小确定的情况下建议使用wrap\_content布局，手动设置TextView控件精确宽高可能导致文字显示不全。

**-** 页面布局内层级尽可能的少，可以使用merge/viewstub优化布局，减少视图树种的节点个数、延迟加载某些非必须控件。

@ 使用DDMS中的Hierarchy View查看页面的视图树。



**-** 使用styles复用样式定义，减少重复代码。

5 [通信][强制]

- 不要通过 Intent 在 Android 基础组件之间传递大数据（binder transaction 缓存为 1MB）。

**-** 子线程中不能更新界面，更新界面必须在主线程中进行，网络操作不能在主线程中调用。

**-** 禁止在多进程之间用SharedPreferences共享数据， (MODE\_MULTI\_PROCESS)模式已过时，并不能保证多进程情况下的数据一致。

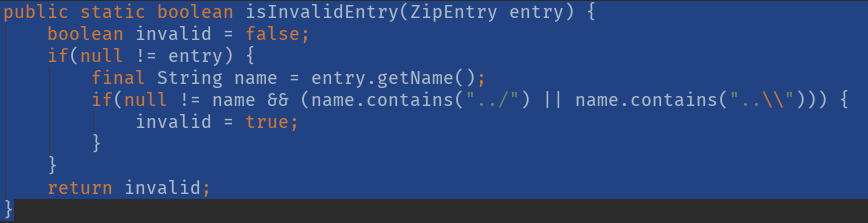
6 [文件操作][强制]

**-** 禁止硬编码文件路径，使用Android文件系统API访问文件系统。

**-** zip解压时需要对可能的../../file这样的路径做过滤，解压含有“../”的文件路径的文件，可能覆盖已有文件，造成攻击。

@ 当 zip 压缩包中允许存在"../"的字符串，攻击者可以利用多个"../"在解压时改变 zip 文件存放的位置，当文件已经存在时就会进行覆盖，如果覆盖掉的文件是 so、 dex或者 odex文件，就有可能造成严重的安全问题

@ 判断代码：



7 [文件操作][建议]

**-** 应用间共享文件时应使用FileProvider。

@ 背景：对于面向 Android 7.0 的应用，Android 框架执行的 StrictMode API 政策禁止在您的应用外部公开 file:// URI。如果一项包含文件 URI 的 intent 离开应用，则应用出现故障，并出现 FileUriExposedException 异常，导致应用崩溃。

@ 常见场景：Call起安装、调用系统拍照、调用系统裁剪等

@ 适配FileProvider：<https://blog.csdn.net/lmj623565791/article/details/72859156>

- SharedPreference提交数据时，如果仅是为了保存这次修改，并没有立即读取操作时，建议使用apply而非commit。（apply提交内存，commit写入磁盘）

8 [资源][建议]

- 大分辨率图片统一放置在xxhdpi目录下，并给低端机型适配低分辨率图。否则可能出现低端机型加载高分辨率图片导致。

@ 为了支持多种屏幕尺寸和密度，Android 为多种屏幕提供不同的资源目录进行适配。为不同屏幕密度提供不同的位图可绘制对象，可用于密度特定资源的配置限定符（在下面详述） 包括 ldpi（低）、 mdpi（中）、 hdpi（高）、 xhdpi（超高）、 xxhdpi （超超高）和 xxxhdpi（超超超高）。例如，高密度屏幕的位图应使用 drawable-hdpi/。根据当前的设备屏幕尺寸和密度，将会寻找最匹配的资源，如果将高分辨率图片放入低密度目录，将会造成低端机加载过大图片资源，又可能造成 OOM，同时也是资源浪费，没有必要在低端机使用大图。

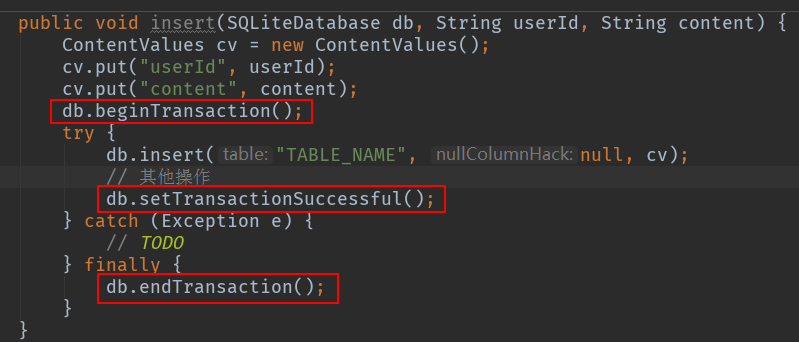
9 [广播][建议]

**-** 对于只用于应用（进程）内的广播，优先使用 LocalBroadcastManager 来进行注册和发送，LocalBroadcastManager 安全性更好，同时拥有更高的运行效率。

@ 避免全局广播中的敏感信息被其他应用恶意解析。

10 [数据库][强制]

-多线程操作写入数据库时，需要使用事务操作，以免出现同步问题。



@ 使用事务操作数据库可以保证数据的统一性和完整性，同时也可以提高效率。事务操作说白了就是把对数据库的一组增删改查操作打包，一起写入数据库（如果失败一起回滚）。

@ Android数据库事务简述：<https://www.cnblogs.com/wuyudong/p/5571168.html>

@ Room框架自动生成的数据库写入代码均采用了上述事务操作模式。

11 [内存][建议]

**-** bitmap使用结束后，在 2.3.3 及以下需要调用 recycle()函数，在 2.3.3 以上 GC 会自动管理，把bitmap的引用置空即可，除非明确不需要再用。

**-** 能用Application Context替代的地方，尽量不要使用Activity、Service的Context。

**-** 慎用static变量引用资源消耗大的实例，比如某个非全局Context。

12 [安全][建议]

**-** 将android:allowbackup属性设置为false，防止adb backup导出数据。