西北師範大學

Android 开法编码规范

题　　目：Android 开法编码规范

学　　院：计算机科学与工程学院

专　　业：计算机科学与技术

班　　级：2019 级卓越工程师班

团队名称：奋起上进组

团队成员：曹永兴 李斌 尚洁 张蓉星

目录

1 第一章 绪论.....3
1.1 概述.....3
1.2 目的.....3
1.3 适用范围.....3
2.第二章 命名注释规范.....3
2.2.1 命名规范.....3
2.2 注释规范.....7
3 第三章 编码规范.....11
3.1 对象释放.....11
3.2 异常处理.....12
3.3 魔鬼数字.....12
3.4 控制台输出.....13
3.5 类的定义.....13
3.6 判断语句.....14
3.7 向上转型.....14
3.8 重复代码.....15
3.9 代码复杂度.....15
3.10 枚举的使用.....16
3.11 参数和返回值.....16
4 第四章 性能与安全.....16
4.1 基本原则.....16
4.2 String 与 StringBuffer.....17
4.3 集合.....17
4.4 对象.....17
4.5 同步.....18
4.6 避免使用 Static 对象.....18
4.7 layout 嵌套层次.....18

1 第一章 绪论

1.1 概述

编码规范对于程序员而言尤为重要，有以下几个原因：

原因 1：一个软件的生命周期中，80%的花费在于维护。

原因 2: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。

原因 3：编码规范可以改善软件的可读性。

1.2 目的

统一规范 Eclipse/studio 编辑环境下 android 的编码风格和标准。此为最基本的编码要求规范，包括文件、注释、命名规范，必须完全遵守。

1.3 适用范围

适用于安卓手机 APP 项目。

2. 第二章 命名注释规范

2.2.1 命名规范

2.1.1 包命名

命名规则：包名采用域后缀倒置的加上自定义的包名，采用小写字母，都应该以 com.*(公司名)开头(不包括一些特殊原因)。在部门内部应该规划好包名的范围，防止产生冲突。部门内部产品使用部门的名称加上模块名称。产品线的产品使用产品的名称加上模块的名称。

说明：除特殊原因包结构都必须以 com.*开头,已有项目包结构不做调整.

格式：com.公司名.产品名.模块名称

2.1.2 类和接口命名

规则一：命名必须使用驼峰规则，即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，类名和接口使用类意义完整的英文描述，不允许出现无意义的单词，如（FirstActivity），应该为（LauncherActivity）。

示例：ChatActivity, LogManager, LogConfig

规则二：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 Application 结尾——MTApplication

Activity 类型的，命名以 Activity 结尾——LoginActivity

fragment 类型的，建议命名以 fragment 结尾——CourseFragment

adapter 类型的，建议命名以 adapter 结尾——ContactDetailAdapter

bean 类型的，请求体，建议命名以 Req 结尾-GetMMSListReq，消息返回提，建议以 Resp 结尾-BackupProgressResp。

2.1.3 方法命名

规则一：方法名是一个动词，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，并且方法名使用类意义完整的英文描述。

示例：

public void addNewOrder();

规则二：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，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。

格式：

set + 属性名()

get + 非布尔属性名()

动词()

动词 + 宾语()

is + 布尔属性名()

示例：

public void setVisible(boolean);

public String getType();

public void show();

public void addKeyListener(Listener);

public boolean isFinished();

规则三：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()

2.1.4 属性名

规则一：属性名使用意义完整的英文描述，变量名应简短且富于描述，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。尽量避免单个字符的变量名，除非是一次性的临时变量。

示例：

private customerName;

private orderNumber;

private smpSession;

规则二：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

2.1.5 常量名

规则一：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 static final 修饰。

示例：

public static final int MAX_VALUE = 1000;

public static final String DEFAULT_START_DATE = “2001-12-08”;

2.1.6 layout 命名

规则一：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：

person_login.xml

2.1.7 id 命名

规则一：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，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_name_show

@+id/book_name_edit

2.1.8 资源命名

规约：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。如果为公共资源，如分割线等，则直接用用途来命名

示例：

home_icon_username.png

home_icon_confirm.png

line.png 或 default_image.png

Home_register_sumbit_bg_selector

2.2 注释规范

2.2.1 基本原则

源程序注释量必须在 30%以上。由于每个文件的代码注释不一定都可以达到 30%，建议以一个系统内部模块作为单位进行检查。

2.2.2 基本概念

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。

2.2.3 文件注释

规则一：包的注释：写入一个名为 package.html 的 HTML 格式的说明文件放入包所在路径。包的注释内容：简述本包的作用、详细描述本包的内容、产品模块名称和版本、公司版权。（目前不涉及到对外开放源码，可以暂时不添加）
说明：方便 JavaDoc 收集，方便对包的了解
示例：
/*

文件名
包含类名列表
版本信息，版本号
创建日期。
版权声明
*/

2.2.4 类注释

规则一：类和接口的注释放在 class 或者 interface 关键字之前，import 关键字之后。注释主要是一句话功能简述与功能详细描述。类注释使用“/** */”注释方式

说明：方便 JavaDoc 收集,没有 import 可放在 package 之后。注释可根据需要列出：作者、内容、功能、与其它类的关系等。功能详细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。

```
/**

〈功能详细描述〉
@version [版本号, 创建时间]
@author [作者]（必须）
@see [相关类/方法]（可选）
@since [产品/模块版本] （必须）
*/
```

示例：

```
/**
* <推送相关字段>
*
* @author caoyinfei
* @version [版本号, 2016/5/23]
* @see [相关类/方法]
* @since [V39]
*/
```

2.2.5 方法注释

规则一：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,返回参数的含义。

示例：

```
/**

<判断对象是否为 null>
@param 需要判断的对象
@return 为 null 返回 true,否则返回 false
*/
public static boolean isNull(Object obj)
{
return (null == obj)    true : false;
}
```

2.2.6 类变量注释

规则一：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，注释方式为“/** /”。

示例：

```
/*

密码输入框
*/
```

2.2.7 其他注释

规则一：方法内部的注释 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“/……/”。

关键跳转需要在代码上方加上注释

示例：

```
@Override
public void onClick(View v)
{
switch (v.getId())
{
/
* 响应返回按钮
/
case R.id.title_back_layout:
// 关闭当前页面
finish();
break;/
* 响应登陆按钮/
case R.id.login:
break;/
* 响应注册按钮
/
case R.id.title_call_layout:
break;/
* 响应忘记密码按钮
*/
case R.id.forget_password:
break;
default:
break;
}
}
```

3 第三章 编码规范

3.1 对象释放

规则一：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()，如果有多个 IO 对象需要 close()，需要分别对每个对象的 close()方法进行 try-catch,防止一个 IO 对象关闭失败其他 IO 对象都未关闭。

示例：

```
try
{
// … …
}
catch(IOException ioe)
{
//… …
}
finally
{
try
{
out.close();
}
catch (IOException ioe)
{
//… …
}

try
{
```

```
in.close();
}
catch (IOException ioe)
{
//… …
}
}
```

规则二：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引用，虚拟机的垃圾收集器就不会回收。

3.2 异常处理

规则一：系统非正常运行产生的异常捕获后，如果不对该异常进行处理，则应该记录日志。

说明：此规则指通常的系统非正常运行产生的异常，不包括一些基于异常的设计。若有特殊原因必须用注释加以说明。

示例：

```
try
{
//… …
}
catch (IOException ioe)
{
logger.error(ioe);
}
```

3.3 魔鬼数字

规则一：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或者枚举来代替。

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
}
```

3.4 控制台输出

规则一：不要使用 System.out 和系统 logger 进行控制台打印，应该使用统一封装的工具类(如：公共日志工具)进行统一记录或者打印。

说明：代码发布的时候可以统一关闭控制台打印，代码调试的时候又可以打开控制台打印，方便调试。

规则二：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。

3.5 类的定义

规则一：一个文件不要定义两个类(并非指内部类)。

说明：方便程序的阅读与代码的维护

3.6 判断语句

规则一：判断语句不要使用”* == true”来判断为真

说明：方便阅读，减少没有必要的计算

以下错误：

```
if (ok == true)
{
……
}
```

以下正确：

```
if (ok)
{
……
}
```

规则二：常量放在 equals 的前面，防止空指针异常。

示例：

```
if (“abc”.equals(bean.getName()))
{
……
}
```

说明：如果为 bean.getName().equals(“abc”),当 bean.getName()为 null 时，则会抛出空指针异常。

3.7 向上转型

规则一：不要写没有必要的向上强制转型。

说明：没必要写的向上强制转型会浪费性能，增加代码阅读难度

示例：以下错误：

```
FileInputStream fis = new FileInputStream(f);
InputStream is = (InputStream)fis;
```

3.8 重复代码

规则一：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。

3.9 代码复杂度

规则一：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设计与算法。

规则二：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。

规则三：类中成员函数的圈复杂度不大于 15。

规则四：一个方法只完成一项功能，在定义系统的公用接口方法外的方法应尽可能的缩小其可见性。避免用一个类是实例去访问其静态变量和方法。避免在一个较长的方法里提供多个出口：

示例：

```
//不要使用这钟方式，当处理程序段很长时将很难找到出口点
if(condition)
{
return A;
}
else
{
return B;
}
//建议使用如下方式
```

```
String result = null;
if(condition)
{
result = A;
}
else
{
result = B;
}
return result;
```

3.10 枚举的使用

规则一：2.3 之前的枚举效率很低，后面 google 优化了，枚举可以放心使用，效率没有太大差别。

3.11 参数和返回值

规则一：避免过多的参数列表，尽量控制在 5 个以内，若需要传递多个参数时，当使用一个容纳这些参数的对象进行传递，以提高程序的可读性和可扩展性。参数类型和返回值尽量接口化，以屏蔽具体的实现细节，提高系统的可扩展性，例如：

```
public void joinGroup(List userList){}
public List listAllUsers(){}
```

4 第四章 性能与安全

4.1 基本原则

性能的提升并不是一蹴而就的，而是由良好的编程积累的，虽然任何良好的习惯和经验所提升的性能都十分有限，甚至微乎其微，但良好的系统性能却是

由这些习惯等积累而成，android 性能优化涉及到大篇幅的介绍跟讲解，这边只是简单列举出几个。

4.2 String 与 StringBuffer

不要使用如下 String 初始化方法： String str = new String(“abcdef”);这将产生两个对象，应当直接赋值： String str = “abcdef”;

在处理可变 String 的时候要尽量使用 StringBuffer 类，StringBuffer 类是构成 String 类的基础。String 类将 StringBuffer 类封装了起来，（以花费更多时间为代价）为开发人员提供了一个安全的接口。当我们在构造字符串的时候，我们应该用 StringBuffer 来实现大部分的工作，当工作完成后将 StringBuffer 对象再转换为需要的 String 对象。比如：如果有一个字符串必须不断地在其后添加许多字符来完成构造，那么我们应该使用 StringBuffer 对象和她的 append() 方法。如果我们用 String 对象代替 StringBuffer 对象的话，将会花费许多不必要的创建和释放对象的 CPU 时间。

4.3 集合

避免使用 Vector 和 HashTable 等旧的集合实现，这些实现的存在仅是为了与旧的系统兼容，而且由于这些实现是同步的，故而在大量操作时会带来不必要的性能损失。在新的系统设计中不当出现这些实现，使用 ArrayList 代替 Vector，使用 HashMap 代替 HashTable。

若却是需要使用同步集合类，当使用如下方式获得同步集合实例：

Map map = Collections.synchronizedMap(new HashMap());

由于数组、ArrayList 与 Vector 之间的性能差异巨大，故在能使用数组时不要使用 ArrayList，尽量避免使用 Vector。

我们经常会使用到 HashMap 这个容器，它非常好用，但是却很占用内存，Android 提供了内存效率更高的 ArrayMap。

4.4 对象

避免在循环中频繁构建和释放对象。不再使用的对象应及时销毁。如无必要，不要序列化对象。

4.5 同步

在不需要同步操作时避免使用同步操作类，如能使用 ArrayList 时不要使用 Vector。尽量少用同步方法，避免使用太多的 synchronized 关键字。尽量将同步最小化，即将同步作用到最需要的地方，避免大块的同步块或方法等。

4.6 避免使用 Static 对象

因为 static 的生命周期过长，使用不当很可能导致 leak，在 Android 中应该尽量避免使用 static 对象。

4.7 layout 嵌套层次

你可以在手机打开 设置—->开发者选项—->显示 GPU 过度绘制，这个开关的作用是按不同颜色值来显示布局的过度绘制，绘制的层次从最优到最差：蓝，绿，淡红，红。一般布局层次不宜过多，一般保持在绿色为比较优的界面布局。

减少布局层次可以从以下几个方面：

1.实现复杂的布局时，尽量使用 RelativeLayout ，由于 RelativeLayout 所需要的嵌套层次少，控件平铺是最终努力的方向，这样性能会好一些。

2.ViewStub 的使用，viewStub 是一个轻量级的页面，我们通常使用它来做预加载处理，来改善页面加载速度和提高流畅性，ViewStub 本身不会占用层级，它最终会被它指定的层级取代。 看到页面布局里面很多 gone 的控件，即使是将某一个控件的 visibility 属性设置为不可见的 gone，在整个页面加载过程中还是会加载此控件的。