[1、简介 3](#_Toc426102152)

[1.1背景 3](#_Toc426102153)

[1.2定义 3](#_Toc426102154)

[2、编码规范 4](#_Toc426102155)

[2.1代码风格 4](#_Toc426102156)

[2.2基本命名规范 4](#_Toc426102157)

[2.3控件命名规范 5](#_Toc426102158)

[2.4资源文件命名规范 5](#_Toc426102159)

[2.5注释 6](#_Toc426102160)

[2.5.1文件注释 6](#_Toc426102161)

[2.5.2类注释 7](#_Toc426102162)

[2.5.3方法注释 8](#_Toc426102163)

[2.5.4类成员变量和常量注释 8](#_Toc426102164)

[2.5.5其他注释 9](#_Toc426102165)

[2.5.6 XML注释 9](#_Toc426102166)

[2.6编程规范 9](#_Toc426102167)

[2.6.1 方法 9](#_Toc426102168)

[2.5.2 参数和返回值 9](#_Toc426102169)

[2.5.3 神秘的数 9](#_Toc426102170)

[2.5.4 控制语句 9](#_Toc426102171)

[2.5.5 访问控制 10](#_Toc426102172)

[3、工程结构 10](#_Toc426102173)

[4、架构机制 11](#_Toc426102174)

[4.1异常处理机制 11](#_Toc426102175)

[4.2多线程机制 12](#_Toc426102176)

[4.3用户验证机制 12](#_Toc426102177)

[4.4缓存处理机制 12](#_Toc426102178)

[4.5静态和单例机制 13](#_Toc426102179)

[5、Activity类规范 13](#_Toc426102180)

[6、Strings.xml 规范 14](#_Toc426102181)

[7、代码审查 14](#_Toc426102182)

[8、产品版本命名规范 14](#_Toc426102183)

[8、部署规范 14](#_Toc426102184)

[9、參考 15](#_Toc426102185)

# 1、简介

## 1.1背景

随着软件开发项目的日益庞大，随着项目参与人员的逐渐增加，项目管理、人员合作和文档交流等诸多方面都面临的巨大挑战。为了更好地管理项目资源；为了方便参与人员之间的团队协作；为了高效地交流项目文档；为了形成统一的JAVA编码风格；以保障项目代码的易维护性和编码安全性，特制定《Android项目开发管理标准和规范》（下称《规范》）

## 1.2定义

|  |  |
| --- | --- |
| **术语或缩写词** | **说明性定义** |
| JavaDoc | 来自Sun 微系统用于生成API 文档为来自Java 源代码的HTML 格式的一种计算机软件工具。 |
| 9Patch | NinePatchDrawable 绘画的是一个可以伸缩的位图图像，Android会自动调整大小来容纳显示的内容。 |

# 2、编码规范

## 2.1代码风格

代码内的空行、缩进、对齐、换行、间隔等代码风格

Eclipse： Eclipse2.1的标准。

Android Studio参照默认的标准即可。

配置方式请自行解决。

## 2.2基本命名规范

|  |  |  |
| --- | --- | --- |
| **类别** | **说明** | **例子** |
| 包 | 包名一律小写, 少用缩写和长名；采用以下规则：[com].[组织名].[项目名] 后面再跟后面为模块名或层级名称。  这里需要根据项目架构做不同的划分，比如MVP架构就使用MVP划分形式划分，MVC就是要MVC形式划分。 | com.aadhk.项目缩写.模块名  com. aadhk.restpos.dao  如：com.aadhk.项目缩写.层级名  com. aadhk.restpos.activities |
| 类 | 类或接口名是个一名词，采用大小写混合的方式，每个单词的首字母大写。尽量使你的类名简洁而富于描述。使用完整单词，避免用缩写词(除非该缩写词被更广泛使用，像URL，HTML) | LoginActivity(Activity类)  ShareFragment(Fragment类)  DownloadService(Service类)  EncryptUtil(工具类)  ApiImpl(接口实现类) |
| 方法 | 方法名是动词[名词]结构，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； | InitView()  toLogin()  goLogin()  loadData()  Get/Set方法请用工具自动生成 |
| 变量 | 只在代码块的开始处声明变量；  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；  变量名不应以下划线或美元符号开头；  尽量避免单个字符的变量名；  组件或部件变量使用其类型名或类型名缩写作其后缀。;  集合类型变量，例如数组和矢量，应采用复数命名或使用表示该集合的名词做后缀。 | String bookName |
| 成员变量 | 使用m开头，其它和变量的规则一致, | boolean mSuccessful |
| 临时变量 | 临时变量通常被取名为i，j，k，m和n，它们一般用于整型；c，d，e，它们一般用于字符型。 | for (int i=0;i<iLen;i++) |
| 常量 | 全部采用大写，单词间用下划线隔开 | Static final String MSG\_GATEWAY\_NOT\_RESPONSE; |

## 2.3控件命名规范

|  |  |  |
| --- | --- | --- |
| **类别** | **说明（前缀）** | **例子** |
| LinearLayout | lay | layFriend |
| RelativeLayout | lay | layMsg |
| FrameLayout | lay | layCart |
| TableLayout | lay | layTab |
| Button | btn | btnHome |
| ImageButton | ibtn | ibtnPlay |
| TextView | tv | tvName |
| EditText | et | etName |
| ListView | lv | lvCart |
| ImageView | iv | ivHead |
| GridView | gv | gvPhoto |

## 2.4资源文件命名规范

|  |  |  |
| --- | --- | --- |
| **类别** | **说明** | **例子** |
| layout文件 | 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。common表示多模块共用的，activity表示Activity....  view表示自定义的View,inc表示嵌入布局 | activity\_common.xml  activity\_login.xml  fragment\_login.xml  adapter\_user.xml  view\_error.xml  Inc\_button.xml |
| drawable | 全部单词小写，单词间以下划线分割。因为AS中有mdpi, hdpi,xhdpi,xxhdpi.... 所对应的大小以dp为单位，如果是XML为背景的话那么是功能+位置的形式表示 | Ic\_add\_24dp.png  Ic\_search\_36dp.png  bg\_toolbar\_search.xml |
| anim | 全部单词小写，单词间以下划线分割，采用以下规则：  模块名\_逻辑名称  逻辑名称 | refresh\_progress.xml  market\_cart\_add.xml  market\_cart\_remove.xml |
| strings | strings的name命名使用参照Android Studio默认推荐 | user\_login用户登录  toolbar\_save 确认  ordering\_confirm 下单 |
| colors | 有关联的，组件式颜色确定规则如下，如Toolbar下各种颜色colorToolbar.text, colorToolbar.bg  单独的,无关联的采用常规模块+含义  常规颜色可单独定义，如black，transparent等 | colorItemAdapter.text  colorItemAdapter.price  colorBtnConfirm  commonBg  divider  light\_white  blue |
| styles | colors的name命名使用  Camel命名法，采用以下规则：  模块名+逻辑名称  逻辑名称 | MainTabBottom  BlueBigButton  GraySmallButton  FriendList |

## 2.5 Import

导入导出请自行解决。

## 2.5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。

### 2.5.1文件注释

所有的源文件都应该在开头有一个注释，其中列出类名、版本信息、日期和版权声明。

如下：

/\*

\* 文件名

\* 包含类名列表

\* 版本信息，版本号

\* 创建日期

\* 版权声明

\*/

需要配置的注释格式如下：

/\*\*

\* Copyright (C) Hong Kong Android Technology Co.

\* All right reserved.

\*/

配置注释格式请自行解决。

### 2.5.2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。

/\*\*

\* @name 类名

\* @author 原作者

\* @date创建日期

\* 修改者，修改日期，修改内容。

\* @description实现的主要功能。

\*/

/\*\*

\* @name ${type\_name}

\* @author ${user}

\* @date ${date}

\* @description

\*

\* ${tags}

\*/

配置方式请自行解决。

### 2.5.3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

\* 方法详述（简单方法可不必详述）

\* @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

\*/

### 2.5.4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

/\*\*

\* XXXX含义

\*/

### 2.5.5其他注释

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。不要在方法内部使用 java doc 形式的注释“/\*\*……\*\*/”，简单的区分方法是，java doc形式的注释在 eclipse中为蓝色，普通注释为绿色。

### 2.5.6 XML注释

如果当前layout 或资源需要被多处调用，或为公共使用的layout（若common\_list\_item），则需要在xml写明注释。要求注释清晰易懂。

## 2.6编程规范

### 2.6.1 方法

 一个方法尽量不要超过30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。

### 2.5.2 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()

### 2.5.3 神秘的数

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）

### 2.5.4 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){

//do something......

}

if (true)

i = 0; //不要使用这种

对于循环：

//不推荐方式\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

while(index < products.getCount()){

//每此都会执行一次getCount()方法，

//若此方法耗时则会影响执行效率

//而且可能带来同步问题，若有同步需求，请使用同步块或同步方法

}

//推荐方式\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//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){

}

### 2.5.5 访问控制

若没有足够理由，不要把实例或类变量声明为公有。通常，实例变量无需显式的设置(set)和获取(gotten)，通常这作为方法调用的边缘效应 (side effect)而产生。

一个具有公有实例变量的恰当例子，是类仅作为数据结构，没有行为。亦即，若你要使用一个结构(struct)而非一个类(如果java 支持结构的话)，那么把类的实例变量声明为公有是合适的。

# 3、工程结构

Android开发工程中初始化的时候如何在初期我们就能搭建一个好的架构，从开发的角度，看到整齐的代码、统一的风格、清晰的分层，对后面工程的开发和扩展非常重要。关于Android架构，目前有MVC,MVP,MVVM，不同的架构会有不同的工程架构，参考github上较为成熟的即可。

# 4、架构机制

## 4.1异常处理机制

* 所有导致异常的代码，都应尽可能提供一个不用引发异常就能检查是否成功的方法。例如，有些对象可以返回为null,如果不加判断就去操作就会报异常。尽管这并不总是可能的，但其目标是在正常执行情况下不应有异常发生；
* 在每个捕捉到的异常中，使用用户可以理解的语言描述异常，而不要将系统异常的信息直接显示给用户；
* 发生系统异常时，系统应能记录该异常，并提供给系统管理员、开发人员等查看。(如使用bugsense把错误发到服务端)；
* 对于正常或预期的错误，或者对于正常的控制流，不要使用异常；
* 业务流程中发生因业务不完整而返回的情况时，请勿使用异常，而采用return false;ref int ResultCode的方式返回业务错误代码。

## 4.2多线程机制

* 主线程负责创建、显示、更新UI控件，启动、停止子线程。用子线程完成逻辑运算并向主线程发出Message消息让主线程更新UI，使用RxAndroid实现子线程和主线程之间的消息传递；
* Android对UI线程的反应时间要求高，超过5秒钟会报ANR错误。确保长时间操作在子线程运行；
* 开发方式参考http://gank.io/post/560e15be2dca930e00da1083，这种新的方式引发了新一波开发热潮，无疑是目前来讲最清晰的，易维护的，最强大的线程框架，该方案的优势和劣势有：
  1. 清晰明了，易维护
  2. 主流网络请求框架 Retrofit+ Okhttp的支持
  3. 结合RxBus将优势发挥到极致
  4. 使用门槛较高
* 简单调用例子



## 4.3用户验证机制

* 用户远程验证，比如：HTTPCLIENT,WEBSERVICE,SOCKET等，Android客户端负责加密，服务端负责解密，为了安全性着想，数据在传输的过程中必须进行加密;

常见有Token机制，session机制，hash校验，加盐处理等，当然还可以HTTPS

* 数据格式验证，对用户输入的数据进行格式验证，比如：邮箱，数字、手机号码、电话，出生年月、时间等等进行严格的格式、长度、为空验证;

项目Util包的RegexUtil正则表达式工具类提供了常见的数据格式验证方法。

## 4.4缓存处理机制

* 缓存管理的原理：通过时间的设置来判断是否读取缓存还是重新下载；
* 我们采用文件缓存法处理，使用File.lastModified()方法得到文件的最后修改时间，与当前时间判断是否过期，从而实现缓存效果；
* 根据数据的更新频率和数据的重要性设置缓存时间

1. 无网络环境下，我们只能读取缓存文件，哪怕缓存早就过期。WiFi网络环境下，缓存时间可以设置短一点，一是网速较快，而是流量不要费用。移动数据流量环境下，缓存时间可以设置长一点，节省流量，就是节省金钱，而且用户体验也更好；
2. 根据文件的类型设置缓存时间，如图片文件可以永远读取缓存。

* 刷新时间设置，如果数据更新时间为1天，则缓存时间设置为4~8小时比较合适，如果是资讯类应用，再减少，2~4小时，如果觉得数据比较重要或者比较受欢迎，用户会经常把玩，再减少，1~2小时，依次类推。为了保险起见，可能需要毫无理由的再次缩减一下。
* 实现代码参考：  
  http://www.cnblogs.com/qianxudetianxia/archive/2012/02/20/2112128.html

## 4.5静态和单例机制

* 静态变量统一在config包下的locSession类中定义，减少使用静态变量；
* 单例模式可以避免一个类构造出多个实例，减少内存消耗，但是单例模式本身就包含静态变量，会常驻系统内存，要减少使用单例模式；

# 5、Activity类规范

* JDK统一使用JDK1.8
* 继承统一的BaseActivity类；
* 在获取参数和对象时，一定要有缺省值，防止获取不动参数和对象时，程序出现异常。

# 7、代码审查

* Inspect Code
* Android Lint

# 8、产品版本命名规范

版本命名采用V\*\*.\*.\*.\*\*\*\*的格式。按下表规定来命名：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| V | \*\* | . | \* | . | \*\* | . | \* |
|  | 主版本号，初始值为01，每十次子版本发布，进行1的递增。 |  | 子版本号,初始值为1，子版本号的更新，必须进行了相应的生产环境的布署。 |  | 修正版本号，初始值为0,针对严重级别Bug或功能缺陷，发布的版本,或是增量Bug的累集修复，递增1 。修正版本号必须对应有生产环境的布署动作。 |  | 编译版本号，初始值为0000,每发布一个测试版本，递增1。每个编译版本的发布，对应一个。每个编译版本的发布，对应一个测试环境布署的工作。 |

# 8、部署规范

* 环境部署：需要统一安装目录、安装路径、安装相同版本的IDE；
* 发布环境系统配置：在AndroidManifest.xml中配置好versionCode和versionName,versionCode是用来版本升级用的，数据类型为整形，versionName则是用来显示给用户看；
* 发布步骤：
  1. 使用统一的签名部署，以便确保代码的安全性和版本的安全升级，各个APP都使用公司相同的密钥签名；
  2. 必须使用合适的私钥生成的数字签名证书来给程序签名，不能使用ADT插件或者ANT工具生成调试证书来发布；
  3. 数字证书都是有有效期的，Android只是在应用程序安装的时候查验证书的有效期，如果已经安装，即使证书过期也不会影响正常功能；
  4. 签名后需使用zipalign优化程序，以来提供手机CPU对代码的访问速度；
  5. 使用Eclipse工具来打包，具体步骤参照eclipse工具操作。

# 9、參考

<https://source.android.com/source/code-style.html>

https://google.github.io/styleguide/javaguide.html

<http://developer.android.com/guide/topics/resources/string-resource.html>