**Android开发规范**

|  |  |
| --- | --- |
| 文档编号： |  |
| 编 写 人： | 王兴才 |
| 审 批 人： | [经理] |
| 生效日期： | [发布日期] |

**惠通团队**

文档历史

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本 | 修订者 | 修订内容 | 修订日期 | 审批者 | 审批日期 |
| 0.1.0 | 王兴才 | 文档初稿 | 2014.12.16 |  |  |
| 1.0.0 | 王兴才 | 对格式进行修改 | 2014.12.17 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

*填写说明：*

* *版本号规则：版本号采用三级生成规则，即0.0.0，第一位表示主版本号，每一次正式发布，主版本号加1，第二位表示从版本号，每一次对文档进行较大范围修订（增、删章节，修改内容主干等），从版本号加1；第三位为修订版本号，在不改变文档主体内容结构与思想，只做文字上的修饰与调整，则修订版本加1。版本号不采用进制策略，按规则累加，当上层版本变化时，次级版本归0。例如0.0.92，当从版本号升为1时，修订版本号归0，成为0.1.0。*
* *修订内容：详细描述修订及修订内容所在章节*

目录

[1 引言 7](#_Toc406595660)

[1.1编写目的 7](#_Toc406595661)

[1.2书写原则 7](#_Toc406595662)

[1.3 参考资料 7](#_Toc406595663)

[2.标识符命名 7](#_Toc406595664)

[2.1标识符命名规则： 7](#_Toc406595665)

[2.1.1标识符组成： 7](#_Toc406595666)

[2.1.2标识符与关键字： 7](#_Toc406595667)

[2.1.3标识符大小写： 7](#_Toc406595668)

[2.2标识符命名法： 7](#_Toc406595669)

[2.2..1驼峰(Camel)命名法： 7](#_Toc406595670)

[2.2.2帕斯卡（Pascal）命名法: 7](#_Toc406595671)

[2.2.3下划线命名法： 7](#_Toc406595672)

[2.2.4匈牙利命名法： 7](#_Toc406595673)

[2.3英文缩写原则： 8](#_Toc406595674)

[2.3.1短单词缩写原则： 8](#_Toc406595675)

[2.3.2.长单词缩写原则： 8](#_Toc406595676)

[2.3.3.常见缩写： 8](#_Toc406595677)

[2.4 分类命名规范 9](#_Toc406595678)

[2.4.1:包(packages)的命名 9](#_Toc406595679)

[2.4.2 类(classes)的命名 10](#_Toc406595680)

[2.4.3 接口(interface)的命名 11](#_Toc406595681)

[2.4.4 方法(methods)的命名 11](#_Toc406595682)

[2.4.5 变量(variables)的命名 11](#_Toc406595683)

[2.4.6 常量(constants)的命名 12](#_Toc406595684)

[2.4.7 XML文件(布局文件)的命名 12](#_Toc406595685)

[2.4.8 资源文件(图片) 的命名 12](#_Toc406595686)

[2.4.9 动画文件(anim文件夹)的命名 13](#_Toc406595687)

[2.4.10 layout中的id命名 13](#_Toc406595688)

[2.4.11 string.xml中的id命名 15](#_Toc406595689)

[2.4.12 drwable中的图片命名 15](#_Toc406595690)

[3.注释规范 15](#_Toc406595691)

[3.1 头文件注释 15](#_Toc406595692)

[3.2 类或接口的注释 16](#_Toc406595693)

[3.3 方法注释 16](#_Toc406595694)

[3.4重写方法的注释 16](#_Toc406595695)

[3.5块注释 17](#_Toc406595696)

[3.6 变量注释 17](#_Toc406595697)

[3.6.1成员变量注释 17](#_Toc406595698)

[3.6.2 局部变量注释 17](#_Toc406595699)

[3.7关键语句注释 18](#_Toc406595700)

[3.8总结 18](#_Toc406595701)

[4.编程规范 18](#_Toc406595702)

[4.1 方法 18](#_Toc406595703)

[4.2 参数和返回值 19](#_Toc406595704)

[4.2.1 参数限制 19](#_Toc406595705)

[4.2.2 返回异常 19](#_Toc406595706)

[4.2.3 神秘的数： 19](#_Toc406595707)

[4.3空格的使用 19](#_Toc406595708)

[4.3.1 关键字 19](#_Toc406595709)

[4.3.2 逗号 19](#_Toc406595710)

[4.3.3 运算符 20](#_Toc406595711)

[4.3.4 for语句 20](#_Toc406595712)

[4.3.5 强制转型 20](#_Toc406595713)

[4. 4 if语句 20](#_Toc406595714)

[4. 4.1 常量判断 20](#_Toc406595715)

[4. 4.2 {}包括 20](#_Toc406595716)

[4. 5 switch语句 20](#_Toc406595717)

[4. 5.1 case之间空行 20](#_Toc406595718)

[4. 5.2 标注 21](#_Toc406595719)

[4. 5.3 default语句 21](#_Toc406595720)

[4. 6循环表达式 21](#_Toc406595721)

[4. 6.1 用for each 代替for 21](#_Toc406595722)

[4. 6.2 循环终止条件 21](#_Toc406595723)

[4. 6.3 外短内长 21](#_Toc406595724)

[4. 6.4 嵌套限制 21](#_Toc406595725)

[4. 7 异常捕获原则 22](#_Toc406595726)

[4. 7.1 try catch捕获异常： 22](#_Toc406595727)

[4. 7.2 throws抛出异常 22](#_Toc406595728)

[4. 8 多异常捕获或抛出情况 23](#_Toc406595729)

[4. 8.1用try catch正常捕获异常 23](#_Toc406595730)

[4. 8.2使用throws正常抛出异常 23](#_Toc406595731)

[4. 8.3使用try catch 错误捕获异常(即直接捕获Exception) 23](#_Toc406595732)

[4. 8.4使用throws错误抛出异常 23](#_Toc406595733)

[4. 9 单行定律 24](#_Toc406595734)

[4. 10 空格缩进： 24](#_Toc406595735)

[4. 11 慎用Log 24](#_Toc406595736)

[4. 11.1 error： 24](#_Toc406595737)

[4. 11.2 warning： 24](#_Toc406595738)

[4. 11.3 informative： 24](#_Toc406595739)

[4. 11.4 debug ： 24](#_Toc406595740)

[4. 11.5 verbose： 25](#_Toc406595741)

[5.Android性能优化： 25](#_Toc406595742)

[5.1 网络编程： 25](#_Toc406595743)

[5.2 ListView 性能优化 25](#_Toc406595744)

[5.2.1 复用convertView 25](#_Toc406595745)

[5.2.2 异步加载图片 25](#_Toc406595746)

[5.2.3 .list中异步加载的图片 25](#_Toc406595747)

[5.3 BaseAdapter避免内存溢出： 25](#_Toc406595748)

[5.4 异步任务 25](#_Toc406595749)

[5.4.1网络处理 26](#_Toc406595750)

[5.4.2 数据库处理 26](#_Toc406595751)

[5.4.3文件处理 26](#_Toc406595752)

[5.5 不同变量的使用： 26](#_Toc406595753)

[5.6 static变量 26](#_Toc406595754)

[5.7 final声明 26](#_Toc406595755)

[5.7 finalize函数 26](#_Toc406595756)

[5.8 软引用(SoftReference)缓存 26](#_Toc406595757)

[5.9 Bitmap 26](#_Toc406595758)

[5.10 两种Context 27](#_Toc406595759)

# 1 引言

## 1.1编写目的

编写本规范的目的是为了进一步规范Android软件编程风格，提高代码的可读性，可靠性，可重用性。确保在开发成员或开发团队之间的工作可以顺利交接，不必花很大力气便能理解代码，以便维护和改进以前的工作，不断的提高软件源程序的质量和可维护性，减少软件维护成本。

## 1.2书写原则

正确性，简约型，条理性

## 1.3 参考资料

参考网络资源1.标识符命名

# 2.标识符命名

## 2.1标识符命名规则：

### 2.1.1标识符组成：

只能有字母，数字，下划线”\_”,美元符号”$”,人民币符号”¥”组成，且首字母不能是数字。

### 2.1.2标识符与关键字：

不能把关键字和保留字作为标识符（如public，string，int，class…）。

### 2.1.3标识符大小写：

android标识符对大小写敏感。

## 2.2标识符命名法：

### 2.2..1驼峰(Camel)命名法：

也称小驼峰命名法，除首单词外，其余所有单词的首字母大写。

### 2.2.2帕斯卡（Pascal）命名法:

也称大驼峰命名法，即所有单词首字母大写。

### 2.2.3下划线命名法：

单词与单词之间用下划线”\_”做间隔。

### 2.2.4匈牙利命名法：

广泛应用于微软编程环境中，在以Pascal命名法的变量前附加小写序列说明该变量的类型。变量取名方式为<scope\_>+<prefix\_>+<qualifier\_>范围前缀，类型前缀，限定词

## 2.3英文缩写原则：

### 2.3.1短单词缩写原则：

较短单词可通过去掉”元音”形成缩写。

### 2.3.2.长单词缩写原则：

较长的单词可取单词的头几个字母形成缩写。

### 2.3.3.常见缩写：

一些约定俗成的英文单词缩写，如下（**程序中不要使用单词缩写除非该缩写是约定俗成的**）,下面为常见英文单词缩写

|  |  |
| --- | --- |
| **名称** | **缩写** |
| icon | ic （主要用在app的图标） |
| color | cl（主要用于颜色值） |
| divider | di（主要用于分隔线，不仅包括Listview中的divider，还包括普通布局中的线） |
| selector | sl（主要用于某一view多种状态，不仅包括Listview中的selector，还包括按钮的selector） |
| average | avg |
| background | Bg（主要用于布局和子布局的背景） |
| buffer | buf |
| control | ctrl |
| delete | del |
| document | doc |
| error | err |
| escape | esc |
| increment | inc |
| infomation | info |
| initial | init |
| image | img |
| Internationalization | I18N |
| length | len |
| library | lib |
| message | msg |
| password- | pwd |
| position | pos |
| server | srv |
| string | str |
| temp | tmp |
| window | wnd(win) |

## **2.4 分类命名规范**

### **2.4.1:包(packages)的命名**

Android包的命名采用反域名命名规则，全部使用小写字母，一级包名为com,二级包名为xx(可以是公司或按个人喜好),三级包名根据应用进行命名，四级包名为模块名或层级名。

|  |  |
| --- | --- |
| **包名** | **此包中包含** |
| com.xx.应用名称缩写.activities | 页面用到的Activity类 (activities层级名用户界面层) |
| com.xx.应用名称缩写.base | 页面中每个Activity类共享的可以写成一个i额BaseActivity类 (基础共享的类) |
| com.xx.应用名称缩写.adapter | 页面用到的Adapter类 (适配器的类) |
| com.xx.应用名称缩写.tools | 此包中包含：公共工具方法类（tools模块名） |
| com.xx.应用名称缩写.bean  (或则 com.xx.应用名称缩写.unity ) | 此包中包含：元素类 |
| com.xx.应用名称缩写.db | 数据库操作类 |
| com.xx.应用名称缩写.view  (或则 com.xx.应用名称缩写.ui ) | 自定义的View类等 |
| com.xx.应用名称缩写.service | Service服务 |
| com.xx.应用名称缩写.broadcast | Broadcast服务 |

### **2.4.2 类(classes)的命名**

类的名称需使用Pascal命名法，即如果类名由单个单词构成，则首字母大写，其余字母小写，若类名由多个字母组成，则每个单词首字母均要被大写，如MainActivity；如果类名中包含单词缩写(应尽量避免缩写，除非该缩写是众所周知的，如HTML,URL)，则该缩写词的每个字母均应大写，如HTMLExample

而且由于类名是代表对象的（java的面向对象特性），所以命名类名的时候尽量使用名词。

|  |  |  |
| --- | --- | --- |
| **类** | **描述** | **例如** |
| activity 类 | Aty或者Activity为后缀标识 | 欢迎页面类WelcomeAty.或者WelcomeActivity |
| Adapter类 | Adp或者Adapte为后缀标识 | 新闻详情适配器NewtDetailAdp或则直接NewDetailAdapter |
| 解析类 | Hlr为后缀标识 | 首页解析类HomePosterHlr |
| 公共方法类 | Tools或Manager为后缀标识 | 线程池管理类：ThreadPoolManager日志工具类：LogTools |
| 数据库类 | 以DBHelper后缀标识 | 新闻数据库：NewDBHelper |
| Service类 | 以Service为后缀标识 | 时间服务TimeService |
| BroadcastReceive类 | 以Broadcast为后缀标识 | 时间通知TimeBroadcast |
| ContentProvider | 以Provider为后缀标识 |  |
| 直接写的共享基础类 | 以Base开头 | BaseActivity,BaseFragment |

### **2.4.3 接口(interface)的命名**

接口命名规则与类相同，均采用帕斯卡尔命名法，多以able或ible结尾，如interface Runnable;interface Aceessible;

### **2.4.4 方法(methods)的命名**

方法的命名规则是采用小驼峰命名法，如onCreate(),run();

|  |  |
| --- | --- |
| **方法** | **说明** |
| initXX() | 初始化相关方法,使用init为前缀标识，如初始化布局initView() |
| isXX() | checkXX()方法返回值为boolean型的请使用is或check为前缀标识 |
| getXX() | 返回某个值的方法，使用get为前缀标识 |
| processXX() | 对数据进行处理的方法，尽量使用process为前缀标识 |
| displayXX() | 弹出提示框和提示信息，使用display为前缀标识 |
| saveXX() | 与保存数据相关的，使用save为前缀标识 |
| resetXX() | 对数据重组的，使用reset前缀标识 |
| clearXX() | 清除数据相关的 |
| removeXXX() | 清除数据相关的 |
| drawXXX() | 绘制数据或效果相关的，使用draw前缀标识 |

### **2.4.5 变量(variables)的命名**

采用小驼峰命名法，类中控件的名称必须与xml布局中的id保持一致。

用统一的量词通过在结尾处放置一个量词，就可以创建更加统一的变量，他们更易于理解，也更容易搜索，如使用strUserFirst和strUserLast,而不要使用strFirstUser和strLastUser.

量词列表： 量词后缀说明

First 一组变量中的第一个

Last 一组变量中的最后一个

Next 一组变量中的下一个变量

Prew 一组变量中的上一个

Cur 一组变量中的当前变量

### **2.4.6 常量(constants)的命名**

常量命名需要全部大写，采用下划线命名法，如MAX\_WIDTH;

### **2.4.7 XML文件(布局文件)的命名**

全部小写，采用下划线命名法

#### **2.4.7.1 Activity默认布局**

以去掉后缀的Activity命名，是不加后缀的

#### **2.4.7.2 Activity子布局**

以父布局名称开始，追加item和子布局功能说明

如Activity默认布局layoutlist.xml,子布局为layoutlist\_item\_content.xml

### **2.4.8 资源文件(图片)** 的命名

全部小写，采用下划线命名法，附加前缀以区分

|  |  |
| --- | --- |
| **名称** | **功能** |
| btn\_xx | 按钮图片使用btn\_整体效果**（selector）** |
| btn\_xx\_normal | 按钮图片使用btn\_正常情况效果 |
| btn\_xx\_press | 按钮图片使用btn\_点击时候效果 |
| bg\_head | 背景图片使用bg\_功能\_说明 |
| def\_search\_cell | 默认图片使用def\_功能\_说明 |
| icon\_more\_help | 图标图片使用icon\_功能\_说明 |
| seg\_list\_line | 具有分隔特征的图片使用seg\_功能\_说明 |
| sel\_ok | 选择图标使用sel\_功能\_说明 |

命名后缀的意义：

|  |  |
| --- | --- |
| **后缀** | **说明** |
| unit | 在使用xml的tilemode来配图片时，element图片使用此后缀 |
| nor | 图片的状态，代表普通状态 |
| hl | 图片的状态，代表高亮状态 |
| press | 图片的状态，代表按下状态 |
| select | 图片的状态，代表其所占的view被选中 |
| unselect | 图片的状态，代表其所占的view没有被选中 |

### **2.4.9 动画文件(anim文件夹)的命名**

全部小写，采用下划线命名法，附加前缀以区分

|  |  |  |
| --- | --- | --- |
| **动画命名例子** | **规范写法** | **备注** |
| fade\_in | 淡入 |  |
| fade\_out | 淡出 |  |
| push\_down\_in | 从下方推入 |  |
| push\_down\_out | 从下方推出 |  |
| push\_left | 推像左方 |  |
| slide\_in\_from\_top | 从头部滑动进入 |  |
| zoom\_enter | 变形进入 |  |
| slide\_in | 滑动进入 |  |
| shrink\_to\_middle | 中间缩小 |  |

### **2.4.10 layout中的id命名**

命名模式为：view缩写\_模块名称\_view的逻辑名称

View的缩写如下：

|  |  |
| --- | --- |
| **控件** | **缩写** |
| LayoutView | lv |
| RelativeView | rv |
| TextView | tv |
| Button | btn |
| ImageButton | imgBtn |
| ImageView | mgView 或则 iv |
| CheckBox | chk |
| RadioButton | rdoBtn |
| analogClock | anaClk |
| DigtalClock | dgtClk |
| DatePicker | dtPk |
| EditText | edtTxt |
| TimePicker | tmPk |
| toggleButton | tglBtn |
| ProgressBar | proBar |
| SeekBar | skBar |
| AutoCompleteTextView | autoTxt |
| ZoomControls | zmCtl |
| VideoView | vdoVi |
| WdbView | webVi |
| RantingBar | ratBar |
| Tab | tab |
| Spinner | spn |
| Chronometer | cmt |
| ScollView | sclVi |
| TextSwitch | txtSwt |
| ImageSwitch | imgSwt |
| listView | lVi 或则lv |
| ExpandableList | epdLt |
| MapView | mapVi |

### 2.4.11 string.xml中的id命名

命名模式：activity名称\_功能模块名称\_逻辑名称/activity名称\_逻辑名称

### 2.4.12 drwable中的图片命名

命名模式：activity名称\_逻辑名称/common\_逻辑名称

# 3.注释规范

注释包括实现注释(implementation comments)和文档注释(document comments).实现注释是使用/\*…\*/和//界定的注释；文档注释由/\*\*…\*/界定的注释，而且可以通过JavaDoc工具转换成HTML文件。

在一个类文件中，注视主要分为头文件注释，类注释，变量注视，方法注释

## **3.1 头文件注释**

每一个源文件的头部都需要有一个注释，其中列出文件的版权声明，文件名，功能描述，版本，作者，创建时间等如：

/\*

\* Copyright(c) 2014,山东艾欧特智慧生活数据有限公司 All rights reserved

\* File Name :Medicine App

\* Version:V1.0

\* Authors:chaos

\* Date:2014.12.15

\* Description:design for students for they to gain information from school

\*/

## **3.2 类或接口的注释**

主要对类或接口的功能，创建时间，作者进行说明，如下

/\*

\* @ClassName: MainActivity

\* @Description:

\* @author: chaos

\* @date: 2014.12.15

\*

## **3.3 方法注释**

主要对一个方法的方法名，参数，返回值，异常信息，主要功能，参考文档进行描述，如下

/\*

\* @name 方法名：btn\_login(View v)

\* @params 参数说明：每个参数各占一行，需进行初始化

\* @return 返回值：注释出成功，失败，错误，异常时的返回情况

\* @exception 异常，标明什么条件下会引发什么样的异常

\* @Description：点击登录

\* @see 参考的JavaDoc

android.app.Activity#onCreateOptionsMenu(android.view.Menu)

\*/

## **3.4重写方法的注释**

包括重写方法的方法名，功能，参数，返回值，参考文档

/\*

\* @重写父类的onTouchEvent(MotionEvent event) Name:onTouchEvent(MotionEvent

\* event)

\* @Function:触屏事件

\* @params：MotionEvent event

\* @return :boolean-是否撤销该事件

\* @see 参考的JavaDoc android.app.Activity#onCreate(android.os.Bundle)

\*/

## 3.5块注释

在实现一段阶段性功能的代码上面做块注释，主要描述代码块的功能

//校验输入的用户名和密码，如果通过的话跳转到另一个页面，否则给出提示

**if** (name.equals("user") && pwd.equals("123")) {

Intent intent = **new** Intent(MainActivity.**this**, NewsActivity.**class**);

startActivity(intent);

} **else** {

Toast.*makeText*(MainActivity.**this**, "对不起，您输入的用户名或密码错误，请重新输入！",

Toast.*LENGTH\_LONG*).show();

}

## 3.6 变量注释

所有成员变量和大多数局部变量在声明时都需要为其做功能注释

### **3.6.1成员变量注释**

如下：

**public** **static** **final** String *NAME* = **null**;// 用户输入的用户名

**public** **static** **final** String *PWD* = **null**;// 用户输入的密码

### **3.6.2 局部变量注释**

如下：

**public** **void** login() {

String name = **null**;// 用户输入的用户名

String pwd = **null**;// 用户输入的密码

//校验输入的用户名和密码，如果通过的话跳转到另一个页面，否则给出提示

**if** (name.equals("user") && pwd.equals("123")) {

Intent intent = **new** Intent(MainActivity.**this**, NewsActivity.**class**);

startActivity(intent);

} **else** {

Toast.*makeText*(MainActivity.**this**, "对不起，您输入的用户名或密码错误，请重新输入！",

Toast.*LENGTH\_LONG*).show();

}

}

## 3.7关键语句注释

在关键语句的右侧要做单条语句注释，如下：

LoginResult loginResult = **new** LoginResult();// 定义用户登录返回实体类

String Pwd = MD5.*generatePassword*(UserPwd).toLowerCase();// 获取加密密码

HttpClient httpclient = **new** DefaultHttpClient();//创建httpclient

httpclient.getParams().setIntParameter(HttpConnectionParams.*SO\_TIMEOUT*,10000);// 设置连接超时

## 3.8总结

文件和方法的注释应该使用/\*\*/形式；变量注释和语句注释应该使用//形式；块注释应尽量使用//形式，当文字较长时，也可以使用/\*\*/形式。

注释量必须不少于代码总量的三分之一。

# 4.编程规范

## 4.1 方法

一个方法只完成一个功能，如果业务逻辑非常复杂的话，需要进行拆分封装，减少代码间的耦合度；且一个方法尽量不要超过30行。

## 4.2 参数和返回值

### **4.2.1 参数限制**

一个方法的参数尽量不要超过4个

### **4.2.2 返回异常**

如果一个方法返回的是一个错误码，需要使用异常，尽可能不要使用null替代异常，如返回List可以使用Collections.emptyList()

### **4.2.3 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，可将它们封装成静态常量(for语句除外).

**if** (**null** == name && **null** == pwd) {

Toast.*makeText*(MainActivity.**this**, "对不起，用户名和密码不能为空，请重新输入！",Toast.*LENGTH\_LONG*).show();

} **else** {

Intent intent = **new** Intent(MainActivity.**this**, NewsActivity.**class**);

startActivity(intent);

}

}

## **4.3**空格的使用

### **4.3**.1 关键字

一个紧跟着括号的关键字应该被括号分开。例如：

While (true) {

…

}

### **4.3.2 逗号**

空格应该位于参数列表逗号的后面。例如：

public void info(String name, String age, String address, String tel){}

### **4.3**.3 运算符

所有二元运算符，除了”.”,都应该使用空格将之与操作数分开。一元操作符和操作数之间不应该加空格，比如自加“++“，自减”—“。例如：

a = c + d;

a = (a + b) / (c \* d);

### **4.3**.4 for语句

For语句中的表达式应该被空格隔开。例如：

for (int I = 0; I < 20; i++){}

### **4.3.5 强制转型**

强制转型后应该跟一个空格。例如：

myMethod((byte) aNum, (Object) x);

myMethod((int) (cp + 5), ((int) (I + 3) + 1)

## 4. 4 if语句

### **4.** 4**.1 常量判断**

判断中如果有常量的话，则应将常量置于判断的左侧，如：

### **4.** 4**.2 {}包括**

if语句必须用{}包括起来，即便只用一句，如上图

## 4. 5 switch语句

### **4.** 5**.1 case之间空行**

switch语句中，break与下一条的case之间要空一行

### **4.** 5**.2 标注**

对于不需要的break语句，使用/\*Falls through \*/标注

### **4.** 5**.3 default语句**

保证switch的完整性，写上default语句

如：

**switch** (v.getId()) {

**case** 1:

**break**;

此处空置一行

**case** 2: /\* Falls through \*/

**break**;

**case** 3:

**break**;

**default**:

**break**;

}

## 4. 6循环表达式

### **4.** 6**.1 用for each 代替for**

尽量使用for each 语句代替原始的for语句

### **4.** 6**.2 循环终止条件**

循环是很浪费资源的工作，必须有终止循环的条件或语句，避免死循环

### **4.** 6**.3 外短内长**

循环要尽可能的短，把长循环放到内部

### **4.** 6**.4 嵌套限制**

嵌套次数不应超过3层，过多的话代码可读性较差。如：

**public** **void** turn() {

**for** (**int** i = 0; i < 100; i++) {

**for** (**int** j = 0; j < 200; i++) {

**for** (**int** z = 0; z < 300; z++) {

System.*out*.println("i=" + i + "j=" + j + "z=" + z);

}

}

}

}

## 4. 7 异常捕获原则

### **4.** 7**.1 try catch捕获异常：**

使用try，catch捕获异常，try中为可能抛出异常的代码段，catch进行异常捕获，切记，catch语句不能为空。如下：

**try**{

HttpResponse response = httpClient.execute(request);

HttpEntity entity = response.getEntity();

}**catch**(Exception e){

}

或许你认为你的代码永远不会出问题，所谓的异常处理并不重要，可类似这种忽视异常就会在代码中埋下一颗炸弹，好的代码不仅要好用，还要具有可发展性，可扩展性。这就要求程序员必须用严谨的态度对待程序，不给程序留下漏洞。

正确的应该如下：

**try**{

HttpResponse response = httpClient.execute(request);

HttpEntity entity = response.getEntity();

}**catch**(Exception e){

e.printStackTrace();

}

### **4.** 7**.2 throws抛出异常**

使用throws抛出异常

Throws JSONException,ClientProtocolException,IOException{}

## 4. 8 多异常捕获或抛出情况

当有多种异常抛出时，不要用顶级的Exception或Throwable，这样会造成错误处理不清晰，将许多不必考虑的异常都卷了进来

### 4. 8.1用try catch正常捕获异常

**catch** (ClientProtocolException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (JSONException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

### 4. 8.2使用throws正常抛出异常

Throws JSONException,ClientProtocolException,IOException{}

### **4.** 8**.3使用try catch 错误捕获异常**(即直接捕获Exception)

Catch(Exception e){e.printStackTrace()}

### **4.** 8**.4使用throws错误抛出异常**

throws Throwable{}

这都是不合适的，因为这意味着将一些本来不用考虑的异常卷入到应用程序的错误处理中来，这会使得错误变得模糊不清，所以，不要为了方便而使得代码质量下降。

## **4. 9 单行定律**

定义变量或常量时，每一行确保只有一个句子，多个语句分行编写。

private int i = 1;

private int j = 2;

而不要使用一下方式：

private int i = 1; private int j = 2;

## 4. 10 空格缩进：

一般代码块缩进的时候使用4个空格(换行后会缩进8个空格)而不要使用Tab，因为空格在所有编辑器里都只占一个字符，但Tab在不同的编辑器里字符会发生变化的，会影响代码的可视性。

## 4. 11 慎用Log

记录日志会对性能产生显著的负面影响，如果日志内容不够简练的话，很快会丧失可用性，日志功能支持五种不同的级别。一下列出了各个界别及其使用的场合和方式。

### **4.** 11**.1 error：**

该级别日志在发生致命错误时使用，即错误的后果能被用户看到，但是不明确删除部分数据，卸载程序，清除数据区或重新刷机等就无法恢复。该级别总是记录日志。需要记录error级别日志的事件一般都应该向统计信息收集(statistics-gathering)服务器报告

### **4.** 11**.2 warning：**

该级别日志用于重大的意外的事件，即错误的后果能被看到，但是不采取明确的动作可能就无法恢复，从等待或重启应用开始，直至重新下载新版程序或重启设备。该级别总是记录日志，需记录warning级别日志的事件也可以考虑向统计信心收集服务器报告

### **4.** 11**.3 informative：**

该级别日志应用于记录大部分人感兴趣的事件，即如果检测到事件的影响面可能很广，但不一定是错误，应该只有那些拥有本区域内最高级别身份认证的模块才能记录这些日志，该级别总是记录日志

### **4.** 11**.4 debug ：**

该级别的日志应该用于进一步记录有关调查，调试意外现象的设备事件。应该只记录那些有关控件运行所必需的信息，如果debug日志占用了太多的日志卡农关键，那就应该使用详细级别日志(verbose)才更合适

### **4.** 11**.5 verbose**：

该级别日志应用于所有其余的事件(除以上四种情况之外的所有事件).该级别仅会在调试版本下记录日志，并且需用if(LOCAL\_LOGV)语句块或等效语句包裹，这样该部分代码就不会编译进发行版本中了，所有构建日志文字的代码将会在发行版本中剥离出去，并且须包含在if(LOCAL\_LOGV)语句块中

# 5.Android性能优化：

## **5.1 网络编程**：

网络编程时，设置连接超时和响应超时时间

## 5.2 ListView 性能优化

### **5.2.1 复用convertView**

在适配器的getItemView中，判断convertView是否为空，如果不为空，可复用。如果convertView中的view需要添加listener，代码一定要卸载if(convertView==null){}之外。具体可参考：

<http://mzh3344258.blog.51cto.com/1823534/889879>

### **5.2.2 异步加载图片**

加载网络图片之类的是很浪费资源的事，不能在主线程中加载，需要另起子线程或使用异步类加载。

### **5.2.3 .list中异步加载的图片**

当不在可视范围内，按照一定的算法及时回收(如在当前可视范围的上下8条item以外的图片进行回收，或者将图片进行缓存，设置一个大小，按照最少使用原则对超过部分进行回收)

## **5.3 BaseAdapter避免内存溢出**：

如果BaseAdapter的实体类有属性非常消耗内存，可以保存到文件；为提高性能，可以进行缓存，并限制缓存大小。

## **5.4 异步任务**

分为核心任务和普通任务，只有核心人物中出现的系统级错误才会报错，异步任务的ui更新操作需要判断原activity是否处于激活状态

### **5.4.1网络处理**

主线程不要进行网络处理

### **5.4.2 数据库处理**

主线程不要进行数据库处理

### **5.4.3文件处理**

主线程不要进行文件处理

## **5.5 不同变量的使用**：

尽可能的少使用普通全局变量，static全局变量，尽量用局部变量代替。全部变量都是静态存储方式，只是作用域不同。其中普通的全局变量作用域整个源程序，而静态的全局变量作用于被定义的源文件中，即其他源文件无法访问，把局部变量改为静态变量更改了它的存储方式即生命周期，把局部变量改为静态变量改变了它的作用域，所以原则能用局部不用全局。

## **5.6 static变量**

Static变量在程序运行期间内存是共享的，且只有一份，所以有时将一些变量声明为static，可以节约内存空间。但是由于静态变量的生命周期较长，和应用程序同步，不易被系统回收，很占用资源，所以静态变量的使用要谨慎，不要盲目使用。

## **5.7 final声明**

类内私有常量尽量用final声明，程序中供全局使用的常量可以声明为static final。

## **5.7 finalize函数**

尽量少用finalize函数，finalize函数是Java提供给程序员释 放对象或资源的机会。但它会加大GC工作量，所以尽量少用finalize释放资源，依 靠系统的自动回收基本上足矣。

## **5.8 软引用(SoftReference)缓存**

在使用图片资源过多时，使用软引用缓存。使用软引用定义的资源在内存不够时会自动被系统回收。可以最大限度保证不产生OutOfMemory异常，一般用于缓存的实现，但缺点是软引用的对象初始化比较耗时，而且还要负责关于对象回收后的重建工作，较为麻烦。有关引用的方面的知识，可参考：<http://www.apkbus.com/forum.php?mod=viewthread&tid=13534>

## **5.9 Bitmap**

对象在不使用时应该先调用recycle()方法释放内存，再将对象赋值为null。

## **5.10 两种Context**

区分Application Context和Activity Context，Application Context的声明周期与应用程序同步，Activity Context的生命周期与类同步，不让生命周期长的对象引用Activity Context，可以避免Activity不能及时被销毁。