**Android编码规范**

**为什么需要编码规范**

* 一个软件的生命周期中，80%的花费用于维护
* 没办法保证开发人员和维护人员是同一个人
* 如果将源码作为作品发布，他不但是给机器理解的，更需要给人理解
* 编码规范可以改善软件的可理解性，并保持软件的清晰无误

**原则**

* 编码前想清楚代码的逻辑结构，必要时可借助图形图表来帮助思考
* 切勿简单的Copy-Paste编码
* 随手重构有“坏味道”的代码
* 保持代码的简单清晰

**格式布局**

* 缩进排版，4个空格作为一个缩进的单位
* 行长度，尽量避免一行的长度超过120个字符
* 换行，当一个表达式无法在一行内书写时，可以依据如下规则断行：
* 在一个逗号后面断开
* 在一个操作符号前面断开
* 宁可选择高级别的断行，而非低级别的断行
* 新的一行应该与上一行同级别的表达式开头对齐
* 如果以上规则导致你的代码混乱或者使你的代码堆积在右边，就代之缩进2个单位（8空格）
* 语句块的“{”不要另起新行
* 尽量使用圆括号来避免运算符优先级问题
* 代码编码使用utf-8

例如：

**privat**e **stati**c synchronized horkingLongMethodName(int anArg,

Object anotherArg, String yetAnotherArg,

Object andStillAnother) {

...

}

**注释**

* 代码本身应该包含软件的大部分信息，通过代码便可以理解它的意思
* 好的注释为了说明软件的意图或者总结程序的功能，而不是简单的重复代码，他解释的是“为什么”而不是“是什么”
* 对使人困惑的代码进行重写而不是注释
* 对过期的注释要进行更新修正，删除冗余和自相矛盾的注释，确保注释清晰正确
* 对以下情况须注释：
* 每个类的功能
* 方法的功能（存取方法外）
* 类变量的功能
* “令人惊异”的代码
* 冗长或复杂的控制结构
* 输入数据的限制
* 去掉自动生成的“// TODO Auto-generated method stub”
* 待改进的代码用“//FIXME:”进行注释

例如：

/\*\*

\* @author sanping.li

*\* AST节点接口*

\*

**\***/

***public*** *interfa*ce Node {

/\*\*

*\* 节*点**创建完成之后**，准备添加子节点

\*/

public void open();

...

}

**命名**

* 命名采用驼峰命名法
* 变量名或函数名首字母小写，后面单词的首字母大写
* 常量名全部大写
* 类名首字母大写
* 包名全部小写
* 命名是为了区分标识，如果相同级别所有的元素都有相同的名字“装饰”，就可以去掉（比如布局文件，类名前面的Alipay）
* 变量名要准确描述变量的含义，不要使用没意义的名字，不要随意使用缩写，特别是有歧义的缩写
* 命名不要过长也不要过短，不要超过4个单词

例如：

**package com.alipay.android.core*;//*包**

publ**i**c cla**ss ActivityShel**l exten**ds RootActivity *{/*/**类

priv**ate int mTy*pe;/*/**变量

pri**vate boolean checkRequisit*e();*/**/方法

**p**u**blic s**t**atic** final int STATE\_NOMARL *= 0*;//常量

**变量**

* 推荐每行只声明一个变量
* 不要将不同类型的变量声明放在一块
* 变量声明可以使用制表符对齐
* 显式的声明变量的作用域
* 尽量缩小变量的作用域，加强封装性。若没足够理由，不要使用public类的变量

例如：

private int mState;

private String mName;

private Map<String,Object> mCache;

**语句**

* 每行只包含一条语句
* 避免在一个语句中给多个变量赋值
* 语句块尽管只有一行语句，也尽量加上“{}”
* switch语句的每个case后面需要有break或者return语句
* switch语句必须要default语句

例如：

**i**f (mType == TYPE\_XML){

**...**

}

switch (mSt**ate)** {

**case STATE\_PA**USE:

**...**

break;

**case STATE**\_INST**AL**LING:

...

br**e**ak;

**def**ault:

throw new Exception("error msg");

}

**方法**

* 每个方法只做一件事并做好
* 每个方法的长度不要超过47行
* 方法的参数个数尽量少，不要超过5个
* 逻辑嵌套层次不要大于三层
* 一个方法只有一个出口

例如：

**publi**c int doSomething(int arg1,int arg2){

int retVal = 0;

...

return retVal;

}

**类**

* 一个类不但要封装数据，还要封装操作
* 使用常量来代替“魔法数字”
* 每个类的长度不要超过1000行
* 不要使用对象来访问静态变量和方法

例如：

publi**c** clas**s Sample{**

**priv**a**te sta**t**ic fi**nal int TYPE\_XML = 0;

...

Processor.parser(arg*ument);//不要new Processor().parser(argument)访问*静态方法和**变量**

...

if(mType == *TYPE\_XML){//用常量来替换“魔法数字”，而不是if(m*Type == 0){

}

**包**

* 包不要划分的过粗也不要过细
* 包的划分按段来区分功能及子功能的模块

**异常处理**

* 尽量减小try块的体积，不要嵌套try块。
* 捕获了异常就需要对它进行适当的处理，而不是丢弃，否则应该向上抛
* 在catch语句中尽可能指定具体的异常类型，必要时使用多个catch，不要试图处理所有可能出现的异常。
* 充分运用finally，保证所有资源都被正确释放
* 不要在finally中出现return语句
* 在异常处理模块中提供适量的错误原因信息，组织错误信息使其易于理解和阅读。
* 全面考虑可能出现的异常以及这些异常对执行流程的影响，不要使用异常来做流程控制。
* 异常捕获不是万能的，不要出现问题全套进来

例如：

**tr**y {

**...**

**} c**atch (FileNotFoundException fe) {

**Log.e(TAG,fe.getMessa*ge());//*组织错**误信息

} catch (IOException ie) {

Log.e(TAG,ie.getMe*ssage())*;//**组织错误信**息

*} catch (Excepti*on e){

*.*../**/不处理应该向**上抛

} finally {

...

}

**资源**

* 尽量使用资源而不是代码来完成功能
* 布局文件要尽量简洁，且模块清晰
* 尽量使用样式来控制布局元素的外观
* 字符常量要放到strings.xml
* 9png的四个区域最好画全
* 一定不要忘记关闭流
* 游标使用方切记关闭游标

**生命周期**

* 应用程序生命周期的对象关联的应该是Application Context
* Activity生命周期的数据关联的才是Activity Context
* 要考虑Activity被回收时候的数据保存和恢复，不单是Activity的数据还有Application的数据

**内存管理**

* 同样的数据只保存一个副本
* 列表类的数据尽量使用AdapterView来展示
* 使用Adapter的时候注意convertView的处理，并可以考虑ViewHolder
* 尽量不要去处理（合成，拷贝）bitmap
* bitmap对象过多的时候考虑使用软引用
* 尽量使用android的activity而非view
* 尽量少用静态变量，否则需要显示的去释放它所占的内存

例如：

publi**c View getView(int position, View convertView, ViewGroup** paren**t) {**

if (convertView **== n**ull) {

...*//创建视图*

}

...//做数据处理

}

**操作习惯**

* 常格式化代码（eclipse使用ctrl+shift+F）
* 去除不必要的包导入（eclipse使用ctrl+o）以及不用的变量和方法
* 代码提交
* 提交代码前用版本控制工具查看所做的修改
* 提交代码前要先做update
* 提交代码的需要写备注，本次修改的详细信息，如果是fix bug，需要给出bug的headline或链接