**Android**

**开发代码规范**

**【PRD】**

# 命名规范

## 类命名

除了符合Java的类命名规范之外，像XxxActivity，XxxService，XxxFragment，XxxDialog, XxxView，XxxRequest等表示属于某种特定类型的类，一般把表示类型的名词放到最后。

例如：LoginActivity,MusicPlayerService,UserCenterFragment,

ProgressDialog 等

## 接口命名

一般以I开头，但是Listener除外，如点击事件的处理接口OnClickListener。

例如：IUserInfo, IMediaPlayer 等

## 变量命名

1. 非公有，非静态的变量以 m开头。

int mPackagePrivate;

private int mPrivateField;

protected int mProtectedField;

public int publicField;

1. 静态变量以s开头。

private static sField;

1. 其他变量以小写字母开头。

int number;

1. public static final 常量全大写并用下划线“\_”分隔。

public static final int ACK\_CODE = 42;

## Layout命名

### layout 文件中的控件 id命名

范围：应用于在res/layout目录下的文件中的控件 id命名。

规范：[控件缩写\_描述]

layout 中所使用的控件 id必须以全部单词小写，单词间以下

划线分割，尽可能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。例如:

@+id/tv\_name @+id/et\_user\_name @+id/rl\_user\_info

|  |  |  |  |
| --- | --- | --- | --- |
| **组件名称** | **简写** | **组件名称** | **简写** |
| **Button** | Btn(btn) | **RadioButton** | Rb(rb) |
| **ImageButton** | Ib(ib) | **TextView** | Tv(tv) |
| **ImageView** | Iv (iv) | **ListView** | Lv(lv) |
| **ProgressBar** | Pb (pb) | **EditText** | Et(et) |
| **ScrollView** | Sv(sv) | **CheckBox** | Ck(ck) |
| **RelativeLayout** | Rl(rl) | **LinearLayout** | Ll(ll) |
| **TableLayout** | Tl(tl) | **AbsoluteLayout** | Al(al) |
| **FrameLayout** | Fl(fl) |  |  |

**表1 组件缩写**

### layout 文件中的资源命名

范围：应用于在res/layout目录下的文件中的资源命名。

规范：[模块名\_用途]

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。

只能使用a-z,0-9,和“\_”组成，并且不能单独使用数字作为资源名称。

## 图片命名

### 静态图片

范围：res/drawable目录下的静态图片，静态图片是指只有一种状

态的图片，如背景图片、图标。

规范：[前缀\_模块]、[前缀\_模块\_描述]、[描述]

前缀：类别的缩写（参照表 2 – 图片前缀）

模块：使用该图片的模块

描述：描述该图片的功能

例如：bg\_main.png ic\_feedback\_send.png menuitem\_save.png

|  |  |
| --- | --- |
| **前缀** | **说明** |
| **ic -- icon** | 主要用于布局和子布局的图标 |
| **bg -- background** | 主要用于布局和子布局的背景 |
| **div -- divider** | 主要用于分隔线，不仅包括Listview中的还包括普通布局中的线 |
| **xx** | 主要用于组件的表示，例如btn表示按钮，详细信息请查看表1 |

**表2 图片前缀**

### 动态图片

范围：res/drawable目录下的动态图片，动态图片是指有多种状态的

图片，如按钮（button）、状态图片、选择按钮（radio button）状态图片。

规范：[前缀\_描述\_状态]、[前缀\_模块\_描述\_状态]

前缀：文件类别的缩写（参照表 2 – 图片前缀）

模块：使用该图片的模块

描述：描述该图片的功能

状态：该图片所表示的状态（参考表3 – 图片后缀）

例如：btn\_login\_normal.png / btn\_login\_pressed.png、

rb\_film\_unselected.png / rb\_film\_selected.png

|  |  |
| --- | --- |
| **后缀/状态** | **说明** |
| **unit** | 在使用xml的tilemode（多个重复平铺）来配图片时，element图片使用此后缀 |
| **normal** | 图片的状态，代表普通状态 |
| **highlight** | 图片的状态，代表高亮状态 |
| **pressed** | 图片的状态，代表按下状态 |
| **selected** | 图片的状态，代表其所占的view被选中 |
| **unselected** | 图片的状态，代表其所占的view没有被选中 |

**表3 图片后缀**

# Android中的Java语言规范

## 异常的捕获

### 捕获的异常：不能忽视

必须在代码中以一些原则性的方式处理每一个异常，具体的处理方式取决于不同的情况。例如：

**void** setServerPort(String value) {  
 **try** {  
 **serverPort** = Integer.*parseInt*(value);  
 } **catch** (NumberFormatException e) {  
 }  
}

推荐的处理方式：

在方法的调用者中抛出异常。

**void** setServerPort(String value) **throws** NumberFormatException {  
 **serverPort** = Integer.*parseInt*(value);  
}

处理错误：

**void** setServerPort(String value) {  
 **try** {  
 **serverPort** = Integer.*parseInt*(value);  
 } **catch** (NumberFormatException e) {  
 **serverPort** = 80; *// default port for server* }  
}

### 不要总是捕获一般异常

**try** {  
 someComplicatedIOFunction(); *// may throw IOException* someComplicatedParsingFunction(); *// may throw ParsingException* someComplicatedSecurityFunction(); *// may throw SecurityException   
 // phew, made it all the way*} **catch** (Exception e) { *// I'll just catch all exceptions* handleError(); *// with one generic handler!*}

这是非常不推荐的，它掩盖了你的代码故障处理的性能。 还意味着，如果有人在你调用的代码中添加了一个新类型的异常，编译器并不会意识到你需要以不同的方式处理这个错误。 而且在大多数情况下，不管怎么样你都不应该以同样的方式处理不同类型的异常。

推荐的处理方式：

分别捕获每一个异常作为单独的catch块。

## Import不使用通配符

例如：当你想要用foo包中的Bar类时，有两种方法去import这个类:

import foo.\*;

优点: 减少潜在import语句的数量。

import foo.Bar;

优点: 可以很明显地看出哪些类确实被使用了，使得代码对于维护者来说，可读性更强。

## 方法要尽量简短

如果一个方法超过40行左右，应考虑是否可以分成几部分同时又不损害程序的结构。保证每个方法只做一件事。将方法分隔开来。

## 在核实的地方定义字段

字段应该定义在该文件的顶部，或者紧接在使用它们的方法之前进行定义。方便管理查看。

## 限制变量的作用域

### 类内部方法中局部变量

局部变量的范围应保持在最小范围。 通过这样做，可以增加代码的可读性和可维护性，降低出错的可能性。

例如：一般，循环变量应该在for语句本身中声明:

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

doSomething(i);

}

### 类全局变量

变量的前缀尽量使用private如果没有外部调用的必要性。

类继承的前缀使用protected，

静态的变量名，前面加final，防止恶性篡改。

。

# Android代码风格规范

## import的顺序

import声明排序如下:

Android imports

第三方的Imports (com, junit, net, org)

java 和 javax

每个分组中按字母顺序排列，大写字母在小写字母之前

## 使用代码整理

Android Studio快捷键为Ctrl +Alt+L，让代码更加的清晰整洁。

## 限制变量的作用域

## 注释

Log所打印出来的信息应具有可读性，不要打印只有自己能看懂的。log使用规范的log分类，例如调试的信息使用DEBUG，错误的信息使用ERROR。

注意发布版本的log中不能打印敏感信息，防止隐私泄漏。

可以在Application文件中使用以下代码，防止正式发布的版本有日志敏感信息泄露。

LogUtil.*setAllowAllLog*(isApkDebugable(getApplicationContext()));

*/\* 判断是否是debug状态的包\*/***public boolean** isApkDebugable(Context context) {  
 **try** {  
 ApplicationInfo info = context.getApplicationInfo();  
 **return** (info.**flags** & ApplicationInfo.***FLAG\_DEBUGGABLE***) != 0;  
 } **catch** (Exception e) {  
  
 }  
 **return false**;  
}