**Android编码规范**

# 一、为什么需要编码规范?

编码规范对于程序员而言尤为重要，有以下几个原因：

一个软件的生命周期中，80%的花费在于维护

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

编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码

如果你将源码作为产品发布，就需要确任它是否被很好的打包并且清晰无误，一如你已构建的其它任何产品

# 二、命名

## 包命名

命名规则：一个唯一包名的前缀总是全部小写的ASCII 字母并且是一个顶级域名，通常是com，edu，gov，mil，net，org。包名的后续部分根据不同机构各自内部的命名规范而不尽相同。这类命名规范可能以特定目录名的组成来区分部门 (department) ，项目(project)，机器(machine)，或注册名(login names)。

例如： com.kaokao.topsales.activities

规约：包命名必须以com. kaokao开始，后面跟有项目名称（或者缩写）,再后面为模块名或层级名称。

如：com. kaokao.项目缩写.模块名 com. kaokao. topsales.bookmark

如：com. kaokao.项目缩写.层级名 com. kaokao. topsales.activities

## 类和接口 命名

命名规则：类名是个一名词，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，避免缩写词(除非该缩写词被更广泛使用，像 URL，HTML)

接口一般要使用able、ible、er 等后缀

例如： class Raster; class ImageSprite;

规约：类名必须使用驼峰规则，即首字母必须大写，如果为词组，则每个单词的首字母也必须要大写，类名必须使用名词，或名词词组。要求类名简单，不允许出现无意义的单词（如 class XXXActivity）。

以下为几种常用类的命名：

1. activity类，命名以Activity为后缀，如：LoginActivity

2. fragment类，命名以Fragment为后缀，如：ShareDialogFragment

3. service类，命名以Service为后缀，如：DownloadService

4. adapter类，命名以Adapter为后缀，如：CouponListAdapter

5. 工具类，命名以Util为后缀，如：EncryptUtil

6. 实体类，命名以Bean为后缀，如：CouponBean

7. 接口实现类，命名以Impl为后缀，如：ApiImpl

## 方法的命名

命名规则：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。

例如： public void run(); public String getBookName();

类中常用方法的命名：

1. 构造方法应该用递增的方式写。（参数多的写在后面）。

2. 初始化方法，命名以init开头，例：initView

3. 类的获取方法（一般具有返回值）一般要求在被访问的字段名前加上get，如getFirstName()，getLastName()。一般来说，get前缀方法返回的是单个值，find前缀的方法返回的是列表值。

4. 类的设置方法（一般返回类型为void）：被访问字段名的前面加上前缀 set，如 setFirstName(),setLastName().

5. 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，如isPersistent()，isString()。或者使用具有逻辑意义的单词，例如equal 或equals。

6. 按钮点击方法，命名以to开头，例：toLogin

7. 通过异步加载数据的方法，命名以load开头，例：loadData

## 变量命名

命名规则：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，尽管这在语法上是允许的。变量名应简短且富于描述。变量名的选用应该易于记忆，即，能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量。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。

例如：String bookName;

规则：变量命名也必须使用驼峰规则，但是首字母必须小写，变量名尽可能的使用名 词或名词词组。同样要求简单易懂，不允许出现无意义的单词。

如：String bookName; //正确

如：String bookNameString; //错误！

* **成员变量命名**

同变量命名，基本要求

私有变量以m开头 private String mUserNmae;

静态变量以s开头 private static String sTag;

注：Activity中控件命名规则，命名应保持和xml文件中的一致。如xml中 定义一个Button的ID为btn\_commit\_auth，则Activity中对应的成员变量声明 应为private Button btn\_commit\_auth。

* **常量命名**

命名规则：类常量的声明，应该全部大写，单词间用下划线隔开。

例如：static final int MIN\_WIDTH = 4;

例如：static final int MAX\_WIDTH = 999;

例如：static final int GET\_THE\_CPU = 1;

* **异常命名**

自定义异常的命名必须以Exception为结尾。已明确标示为一个异常。

## layout 命名

组件类型\_{范围\_}功能，范围可选，只在有明确定义的范围内才需要加上。

以下为几种常用的组件类型命名：

1. activity\_{范围\_}功能，为Activity的命名格式

2. fragment\_{范围\_}功能，为Fragment的命名格式

3. dialog\_{范围\_}功能，为Dialog的命名格式

4. item\_list\_{范围\_}功能，为ListView的item命名格式

5. item\_grid\_{范围\_}功能，为GridView的item命名格式

6. header\_list\_{范围\_}功能，为ListView的HeaderView命名格式

7. footer\_list\_{范围\_}功能，为ListView的FooterView命名格式

## 控件id 命名

规约：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。

命名为：控件缩写\_{范围\_}意义，范围可选，只在有明确定义的范围内才需要加上。

如：某TextView @+id/textbookname 错误 !应为 @+id/tv\_book\_name\_show

如：某EditText @+id/textbookname 错误 !应为 @+id/et\_book\_name\_edit

各种控件的缩写如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 控件 | 缩写 | 控件 | 缩写 |
| TextView | tv | EditText | et |
| Button | btn | ImageButton | btn |
| ImageView | img | ListView | list |
| RadioGroup | group | RadioButton | radio |
| ProgressBar | progress | SeekBar | seek |
| CheckBox | chk | Spinner | spinner |
| TableLayout | table | TableRow | row |
| LinearLayout | linear | RelativeLayout | relative |
| ScrollView | scroll | SearchView | search |
| TabHost | host | TabWidget | widget |

## Strings命名

类型\_{范围\_}功能，范围可选。

以下为几种常用的命名：

1.页面标题，命名格式为：title\_页面

2.按钮文字，命名格式为：btn\_按钮事件

3.标签文字，命名格式为：label\_标签文字

4.选项卡文字，命名格式为：tab\_选项卡文字

5.消息框文字，命名格式为：toast\_消息

6.编辑框的提示文字，命名格式为：hint\_提示信息

7.图片的描述文字，命名格式为：desc\_图片文字

8.对话框的文字，命名格式为：dialog\_文字

9.menu的item文字，命名格式为：action\_文字

## Colors命名

前缀{\_控件}{\_范围}{\_后缀}，控件、范围、后缀可选，但控件和范围至少要有一个。

背景颜色，添加bg前缀

文本颜色，添加txt前缀

分割线颜色，添加line前缀

区分状态时，默认状态的颜色，添加normal后缀

区分状态时，按下时的颜色，添加pressed后缀

区分状态时，选中时的颜色，添加selected后缀

区分状态时，不可用时的颜色，添加disable后缀

## drawable命名

前缀{\_控件}{\_范围}{\_后缀}，控件、范围、后缀可选，但控件和范围至少要有一个。

图标类，添加icon前缀

背景类，添加bg前缀

分隔类，添加line前缀

默认类，添加default前缀

区分状态时，默认状态，添加normal后缀

区分状态时，按下时的状态，添加pressed后缀

区分状态时，选中时的状态，添加selected后缀

区分状态时，不可用时的状态，添加disable后缀

多种状态的，添加selector后缀（一般为ListView的selector或按钮的selector）

## 动画命名

{范围\_}动画类型\_动画方向。范围可选。

xxx\_activity\_fade\_in，淡入

fade\_out，淡出

push\_down\_in，从下方推入

push\_down\_out，从下方推出

slide\_in\_from\_top，从头部滑动进入

zoom\_enter，变形进入

shrink\_to\_middle，中间缩小

## 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。

/\*\*

\* @author ${USER} jyb\_96@sina.com

\* @version V1.0

\* @Description:请添加描述

\* @date ${TIME}

\*/

## 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

\* <p>方法详述（简单方法可不必详述）</p>

\* @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

\* \*/

## 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

private String mAuthor // 作者名字

# 代码风格

## 缩进

使用IDE默认缩进，直接格式化代码即可

## 空行

空行将逻辑相关的代码段分隔开，以提高可读性。

下列情况应该总是使用空行：

一个源文件的两个片段(section)之间

类声明和接口声明之间

两个方法之间

方法内的局部变量和方法的第一条语句之间

一个方法内的两个逻辑段之间，用以提高可读性

规约：通常在 变量声明区域之后要用空行分隔，常量声明区域之后要有空行 分隔，方法声明之前要有空行分隔。

## 空格使用

int result = a + b; //Good, = 和 + 两边各用一个空格隔开

int result=a+b; //Bad,=和+两边没用空格隔开

编译器默认格式化，即可达到效果，不用刻意去空格操作。

## 定义变量

一行声明一个变量，不要一行声明多个变量，这样有利于写注释。

private String param1; // 参数1

private String param2; // 参数2

# 规约

## 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。

不要使用 try catch 处理业务逻辑！！！！

## 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()

## 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以及枚举！（for语句中除外）

## 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：

1、if ( true == isAdmin())...

2、尽量不使用三目条件的嵌套。

3、所有if 语句必须用{}包括起来,即便是只有一句：

避免这样

if (true)

i = 0; //

正确的方式

if (true){

//do something......

}

## 对于循环

避免如下

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

}

应该如下

//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){

}

## 字符串的比较中

避免如下

if(user.getName().equals(“”))

应该如下

if(“”.equals(user.getName()))

避免空指针异常

## 异常的捕捉处理

通常的思想是只对错误采用异常处理：逻辑和编程错误，设置错误，被破坏的数据，资源耗尽，等等。

通常的法则是系统在正常状态下以及无重载和硬件失效状态下，不应产生任何异常。

最小化从一个给定的抽象类中导出的异常的个数。对于经常发生的可预计事件不要采用异常。不要使用异常实现控制结构。

若有finally 子句，则不要在try 块中直接返回，亦不要在finally 中直接返回。

## 范围型的常量用枚举类定义，而不要直接用整型或字符，减少范围值的有效性检查。

// 用枚举类定义，Good

public enum CouponType {

// 现金券

@SerializedName("1")

CASH,

// 抵用券

@SerializedName("2")

DEBIT,

// 折扣券

@SerializedName("3")

DISCOUNT

}

// 用整型定义，Bad

public static final int TYPE\_CASH = 1; // 现金券

public static final int TYPE\_DEBIT = 2; // 抵扣券

public static final int TYPE\_DISCOUNT = 3; // 折扣券

## 文字大小的单位统一用sp，元素大小的单位统一用dp。

## 资源引用

资源的引用（包括文字，颜色）统一通过安卓的标准引用方式，不允许出现直接的硬编码。

# 约定俗成

## 变量赋值

避免在一个语句中给多个变量赋相同的值。它很难读懂。例如：

fooBar.fChar = barFoo.lchar = 'c';

不要将赋值运算符用在容易与相等关系运算符混淆的地方。例如：

if (c++ = d++) { // AVOID! (Java disallows)

...

}

应该写成

if ((c++ = d++) != 0) {

...

}

不要使用内嵌(embedded)赋值运算符试图提高运行时的效率，这是编译器的工作。例如：

d = (a = b + c) + r; // AVOID!

应该写成

a = b + c;

d = a + r;

## 圆括号

一般而言，在含有多种运算符的表达式中使用圆括号来避免运算符优先级问题，是个好方法。

即使运算符的优先级对你而言可能很清楚，但对其他人未必如此。你不能假设别的程序员和你一样清楚运算符的优先级。

if (a == b && c == d) // AVOID!

if ((a == b) && (c == d)) // RIGHT

## 返回值

设法让你的程序结构符合目的。例如：

if (booleanExpression) {

return true;

} else {

return false;

}

应该代之以如下方法：

return booleanExpression

类似地：

if (condition) {

return x;

}

return y;

应该写做：

return (condition ? x : y);

条件运算符"?"前的表达式

如果一个包含二元运算符的表达式出现在三元运算符" ? : "的"?"之前，那么应该给表达式添上一对圆括号。例如：

(x >= 0) ? x : -x

# 框架约定

## 所有调试显示信息，必须通过tools属性去设置

例如：

<TextView

android:id="@+id/tv\_to\_account\_fen"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_alignParentRight="true"

android:layout\_marginRight="15dp"

android:textSize="16sp"

tools:text="第六期" />

tools数据只此预览模式下显式，在运行模式下不显示，方便直接预览布局。

## 时间格式处理，统一调用StdTimeUtils进行操作

转换为时间对象 public static Date parseStringToDate(String date) {}，然后调用相关方法完成业务需要。禁止直接通过截取等方式进行处理。

## 严格遵守框架的模块方法，来书写相应功能代码

@Override

protected void loadViewLayout() {

//加载布局

}

@Override

protected void initView() {

//初始化控件（如果通过注入实现，可以不用写）

}

@Override

protected void initData() {

//初始化数据

}

@Override

protected void setListener() {

//设置监听

}

@Override

protected void processLogic() {

//处理业务逻辑，一般为发送网络请求

}

@Override

public void onReceive(BaseResponse message) {

//消息处理

}

@Override

public boolean handleMessage(Message msg) {

//异步网络数据处理

return false;

}

@Override

public void onClick(View v) {

//点击事件监听

}

# 需要避免的代码行为

## Duplicated Code

代码重复几乎是最常见的异味了。他也是Refactoring 的主要目标之一。代码重复往

往来自于copy-and-paste 的编程风格。

## Long method

它是传统结构化的“遗毒“。一个方法应当具有自我独立的意图，不要把几个意图

放在一起。

## Large Class

大类就是你把太多的责任交给了一个类。这里的规则是One Class One Responsibility。

## Primitive Obsession

面向对象的新手通常习惯使用几个原始类型的数据来表示一个概念。譬如对于范围，他们会使用两个数字。对于Money，他们会用一个浮点数来表示。因为你没有使用对象来表达问题中存在的概念，这使得代码变的难以理解，解决问题的难度大大增加。好的习惯是扩充语言所能提供原始类型，用小对象来表示范围、金额、转化率、邮政编码等等。

## Switch Statement

基于常量的开关语句是代码阅读的障碍，你应当把他变为子类、state 或strategy。

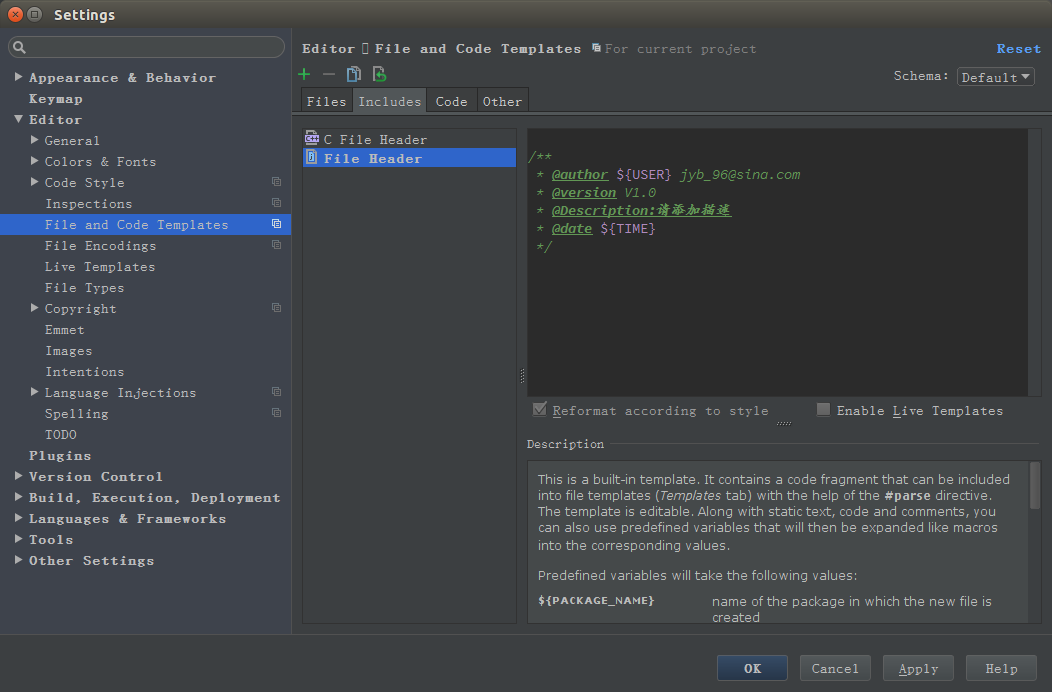
## Lazy Class

一个干活不多的类。类的维护需要额外的开销，如果一个类承担了太少的责任，应当消除它。

# Android Studio配置方法

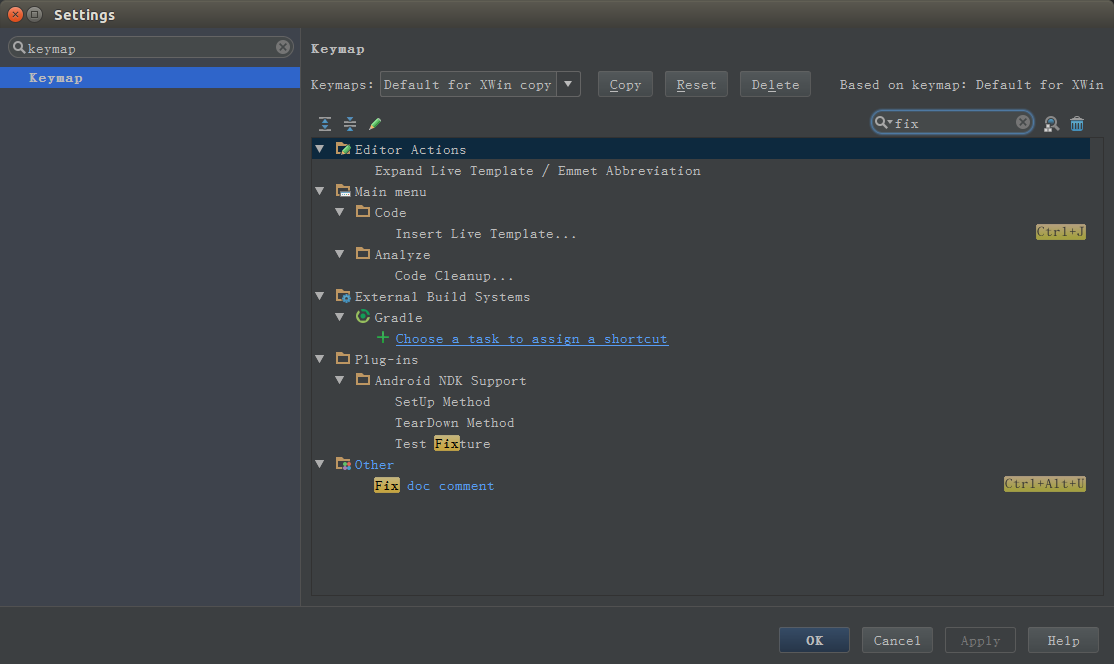
## 注释模板

1. setting – File and Code Templates



## 添加方法注释方法

1. setting – Keymap – Fix dox comment 设置相应的热键即可



编辑：金雅博 殷胜义

2016年4月15日15:49:00

版本历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 编辑人 | 时间 | 备注 |
| V1 | 金雅博 | 2016年4月14日 | 初版 |
| V2 | 殷胜义 | 2016年4月15日 | 内容补充 |
| V3 | 金雅博 | 2016年4月15日 | 补充框架约束 |

Reference

<http://keeganlee.me/post/android/20150709>

<http://google-styleguide.googlecode.com/svn/trunk/javaguide.html>

<http://www.360sdn.com/2013/android_0517/144.html>