# 大银幕代码设计规范 (Android)

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 2015/11/11 | 1.0 | 创建文档 | 梁窠金 |

目录

[大银幕代码设计规范 (Android) 1](#_Toc435002761)

[代码格式规范 1](#_Toc435002762)

[命名规范 2](#_Toc435002763)

[代码的注释 3](#_Toc435002764)

[代码的新增修改更新 3](#_Toc435002765)

[代码提交 4](#_Toc435002766)

## 代码格式规范

代码的风格采用传统的Android代码风格，不同的是，为了代码的可读性考虑，我们规定所有的类结构，接口，方法的 {} 都需要独占一行, 其他代码结构使用传统的Unix风格（即{}错开放置）， 对于if {} else {}语句， try{} catch{} finally {} 语句， 建议使用

if (a == 3) {

}

else {

}

try {

}

catch (Exception e) {

}

finally{

}

这个方式， 控制语句最好都使用 {} 括起来，避免出错和增加代码可读性

示例:

@Annotation(param1 = "value1", param2 = "value2")

@SuppressWarnings({"ALL"})

public class Foo<T extends Bar & Abba, U>

{

int[] X = new int[]{1, 3, 5, 6, 7, 87, 1213, 2};

int[] empty = new int[]{};

public void foo(int x, int y)

{

Runnable r = () -> {

};

Runnable r1 = this::bar;

for (int i = 0; i < x; i++) {

y += (y ^ 0x123) << 2;

}

do {

try (MyResource r1 = getResource(); MyResource r2 = null) {

if (0 < x && x < 10) {

while (x != y) {

x = f(x \* 3 + 5);

}

}

else {

synchronized (this) {

switch (e.getCode()) {

//...

}

}

}

}

catch (MyException e) {

}

finally {

int[] arr = (int[]) g(y);

x = y >= 0 ? arr[y] : -1;

Map<String, String> sMap = new HashMap<String, String>();

Bar.<String, Integer>mess(null);

}

} while (true);

}

void bar()

{

{

return;

}

}

}

class Bar

{

static <U, T> U mess(T t)

{

return null;

}

}

interface Abba

{

}

## 命名规范

* 1. Java代码的命名规则

|  |  |
| --- | --- |
| **名字** | **规则** |
| 类名，接口名 | 大驼峰式命名（首字母大写，每个单词的首字母大写） |
| 方法名 | 小驼峰式命名（首字母小写, 其他每个单词的首字母大写） |
| 常量 | 所有字母全部大写, 单词之间用下划线分开 |
| 类变量 | 所有的类变量都需要使用m开头，其他每个单词首字母大写 |
| 局部变量, 参数 | 尽量使用小驼峰式命名 |

* 1. Drawable文件命名规则

|  |  |
| --- | --- |
| **用途** | **规则 (所有的Drawable文件名每个单词都以下划线分开)** |
| 用于button | 以btn为前缀 |
| 用于icon | 以ic为前缀 |
| 用于image | 以img为前缀 |
| 用于background | 以bg为前缀 |
| 对于自定义的xml的drawable 都应该放置在drawable/目录下，其他drawable目录下放置适配不同分辨率的图片资源文件 | |

* 1. Layout文件命名规则

|  |  |
| --- | --- |
| **用途** | **规则 (所有的layout文件名每个单词都以下划线分开)** |
| 用于activity | 以activity为前缀 |
| 用于fragment | 以fragment为前缀 |
| 用于dialog | 以dialog为前缀 |
| 其他 | 以layout为前缀 |

## 代码的注释

如果你写的是一个公用的API接口，则需要对接口的公开类和方法进行完整的说明和注释，方便他人使用你的接口, 对于一般类和方法的注释，尽量使用

/\*\*

\*

\*/

方式进行注释, 如果有需要产生 Javadoc 的地方，需要对参数进行解释

## 代码的新增修改更新

* 1. 新增

如果你新增了一个.java 或者.xml文件, 则需要添加你的姓名和日期的注释，并需要提供这个文件的基本用途, 示例:

*/\*\*  
 \* Author: xoracle ( Liang Ke Jin )  
 \* Date: 2015/11/10  
 \*/*

* 1. 修改

如果你修改了别人的代码，则需要在这个文件中标记你所修改的区域，并加上你的信息和修改的目的, 示例:

// Modified By Liang Ke Jin 2015/11/11

// 修改数据数组链表为LinkedList  
mListData = new LinkedList<>();  
// End

如果是修复了一个bug, 则需要加上所修复bug的ID, 和基本的bug描述

// Repaired BUG\_001 By Liang Ke Jin 2015/11/11 直接返回  
return;  
// End

## 代码提交

代码使用git进行版本控制，主要有两个分支 master 和 develop分支， develop分支为开发分支，master 为稳定分支，所以本地代码不能直接提交到 master分支上， 只有当develop分支完成了功能开发并稳定后才能合并到 master 分支上!

当需要尝试新功能开发时，需要创建新的分支，避免污染原有代码，当功能开发完成，合并到 develop 分支, 最好合并到 master 分支