

Android编码规范

基于Android的校园社交及活动管理平台App

项 目 承 担 组： G10

撰 写 人（签名）：盛轶群 刘坤 钱金港

完 成 日 期：2017/10/20

评审负责人（签名）：

评 审 日 期：不作评审

文件修改控制页

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **更新时间** | **更新人** | **更新摘要** |
| V0.1 | 2017-12-13 | 盛轶群 | 初稿完成 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

文件修改控制页 2

1 类声明 4

1.1 只有一个顶级类声明 4

2.1大括号 4

2.1.1 使用大括号(即使是可选的) 4

2.1.2 非空块：K & R 风格 4

2.1.3 空块：可以用简洁版本 5

2.2 一行一个语句 5

2.3.1 变量声明 5

2.3.2 数组 5

2.3.3 switch语句 6

2.3.4 注解(Annotations) 7

3.命名约定 8

3.1 对所有标识符都通用的规则 8

3.2 标识符类型的规则 8

3.2.1 包名 8

3.2.2 类名 8

3.2.3 方法名 9

3.2.4 常量名 9

3.2.5 非常量字段名 10

3.2.6 参数名 10

3.2.7 局部变量名 10

3.2.8 类型变量名 10

3.3 驼峰式命名法(CamelCase) 10

4.编程实践 11

4.1 @Override：能用则用 11

4.2 捕获的异常：不能忽视 11

4.3 静态成员：使用类进行调用 12

# G10 Android编码规范（参考Google Java编码规范）

# 1 类声明

## 1.1 只有一个顶级类声明

每个顶级类都在一个与它同名的源文件中(当然，还包含.java后缀)。

例外：package-info.java，该文件中可没有package-info类。

## 2.1大括号

### 2.1.1 使用大括号(即使是可选的)

大括号与if, else, for, do, while语句一起使用，即使只有一条语句(或是空)，也应该把大括号写上。

### 2.1.2 非空块：K & R 风格

对于非空块和块状结构，大括号遵循Kernighan和Ritchie风格 ([Egyptian brackets](http://www.codinghorror.com/blog/2012/07/new-programming-jargon.html)):

左大括号前不换行

左大括号后换行

右大括号前换行

如果右大括号是一个语句、函数体或类的终止，则右大括号后换行; 否则不换行。例如，如果右大括号后面是else或逗号，则不换行。

示例：

return new MyClass() {

@Override public void method() {

if (condition()) {

try {

something();

} catch (ProblemException e) {

recover();

}

}

}

};

### 2.1.3 空块：可以用简洁版本

一个空的块状结构里什么也不包含，大括号可以简洁地写成{}，不需要换行。例外：如果它是一个多块语句的一部分(if/else 或 try/catch/finally) ，即使大括号内没内容，右大括号也要换行。

示例：

void doNothing() {}

## 2.2 一行一个语句

每个语句后要换行。

### 2.3.1 变量声明

#### 2.3.1.1 每次只声明一个变量

不要使用组合声明，比如int a, b;。

#### 2.3.1.2 需要时才声明，并尽快进行初始化

不要在一个代码块的开头把局部变量一次性都声明了(这是c语言的做法)，而是在第一次需要使用它时才声明。 局部变量在声明时最好就进行初始化，或者声明后尽快进行初始化。

### 2.3.2 数组

#### 2.3.2.1 数组初始化：可写成块状结构

数组初始化可以写成块状结构，比如，下面的写法都是OK的：

new int[] {

0, 1, 2, 3

}

new int[] {

0,

1,

2,

3

}

new int[] {

0, 1,

2, 3

}

new int[]

{0, 1, 2, 3}

#### 2.3.2.2 非C风格的数组声明

中括号是类型的一部分：String[] args， 而非String args[]。

### 2.3.3 switch语句

**术语说明**：switch块的大括号内是一个或多个语句组。每个语句组包含一个或多个switch标签(case FOO:或default:)，后面跟着一条或多条语句。

#### 2.3.3.1 缩进

与其它块状结构一致，switch块中的内容缩进为2个空格。

每个switch标签后新起一行，再缩进2个空格，写下一条或多条语句。

#### 2.3.3.2 Fall-through：注释

在一个switch块内，每个语句组要么通过break, continue, return或抛出异常来终止，要么通过一条注释来说明程序将继续执行到下一个语句组， 任何能表达这个意思的注释都是OK的(典型的是用// fall through)。这个特殊的注释并不需要在最后一个语句组(一般是default)中出现。示例：

switch (input) {

case 1:

case 2:

prepareOneOrTwo();

// fall through

case 3:

handleOneTwoOrThree();

break;

default:

handleLargeNumber(input);

}

#### 2.3.3.3 default的情况要写出来

每个switch语句都包含一个default语句组，即使它什么代码也不包含。

### 2.3.4 注解(Annotations)

注解紧跟在文档块后面，应用于类、方法和构造函数，一个注解独占一行。这些换行不属于自动换行(第4.5节，自动换行)，因此缩进级别不变。例如：

@Override

@Nullable

public String getNameIfPresent() { ... }

**例外**：单个的注解可以和签名的第一行出现在同一行。例如：

@Override public int hashCode() { ... }

应用于字段的注解紧随文档块出现，应用于字段的多个注解允许与字段出现在同一行。例如：

@Partial @Mock DataLoader loader;

参数和局部变量注解没有特定规则。

2.3.5 注释

2.3.5.1 块注释风格

块注释与其周围的代码在同一缩进级别。它们可以是/\* ... \*/风格，也可以是// ...风格。对于多行的/\* ... \*/注释，后续行必须从\*开始， 并且与前一行的\*对齐。以下示例注释都是OK的。

/\*

\* This is // And so /\* Or you can

\* okay. // is this. \* even do this. \*/

\*/

注释不要封闭在由星号或其它字符绘制的框架里。

Tip：在写多行注释时，如果你希望在必要时能重新换行(即注释像段落风格一样)，那么使用/\* ... \*/。

2.3.6 Modifiers

类和成员的modifiers如果存在，则按Java语言规范中推荐的顺序出现。

public protected private abstract static final transient volatile synchronized native strictfp

# 3.命名约定

## 3.1 对所有标识符都通用的规则

标识符只能使用ASCII字母和数字，因此每个有效的标识符名称都能匹配正则表达式\w+。

在Google其它编程语言风格中使用的特殊前缀或后缀，如name\_, mName, s\_name和kName，在Java编程风格中都不再使用。

## 3.2 标识符类型的规则

### 3.2.1 包名

包名全部小写，连续的单词只是简单地连接起来，不使用下划线。

### 3.2.2 类名

类名都以UpperCamelCase风格编写。

类名通常是名词或名词短语，接口名称有时可能是形容词或形容词短语。现在还没有特定的规则或行之有效的约定来命名注解类型。

测试类的命名以它要测试的类的名称开始，以Test结束。例如，HashTest或HashIntegrationTest。

### 3.2.3 方法名

方法名都以lowerCamelCase风格编写。

方法名通常是动词或动词短语。

下划线可能出现在JUnit测试方法名称中用以分隔名称的逻辑组件。一个典型的模式是：test<MethodUnderTest>\_<state>，例如testPop\_emptyStack。 并不存在唯一正确的方式来命名测试方法。

### 3.2.4 常量名

常量名命名模式为CONSTANT\_CASE，全部字母大写，用下划线分隔单词。那，到底什么算是一个常量？

每个常量都是一个静态final字段，但不是所有静态final字段都是常量。在决定一个字段是否是一个常量时， 考虑它是否真的感觉像是一个常量。例如，如果任何一个该实例的观测状态是可变的，则它几乎肯定不会是一个常量。 只是永远不打算改变对象一般是不够的，它要真的一直不变才能将它示为常量。

// Constants

static final int NUMBER = 5;

static final ImmutableList<String> NAMES = ImmutableList.of("Ed", "Ann");

static final Joiner COMMA\_JOINER = Joiner.on(','); // because Joiner is immutable

static final SomeMutableType[] EMPTY\_ARRAY = {};

enum SomeEnum { ENUM\_CONSTANT }

// Not constants

static String nonFinal = "non-final";

final String nonStatic = "non-static";

static final Set<String> mutableCollection = new HashSet<String>();

static final ImmutableSet<SomeMutableType> mutableElements = ImmutableSet.of(mutable);

static final Logger logger = Logger.getLogger(MyClass.getName());

static final String[] nonEmptyArray = {"these", "can", "change"};

这些名字通常是名词或名词短语。

### 3.2.5 非常量字段名

非常量字段名以lowerCamelCase风格编写。

这些名字通常是名词或名词短语。

### 3.2.6 参数名

参数名以lowerCamelCase风格编写。

参数应该避免用单个字符命名。

### 3.2.7 局部变量名

局部变量名以lowerCamelCase风格编写，比起其它类型的名称，局部变量名可以有更为宽松的缩写。

虽然缩写更宽松，但还是要避免用单字符进行命名，除了临时变量和循环变量。

即使局部变量是final和不可改变的，也不应该把它示为常量，自然也不能用常量的规则去命名它。

### 3.2.8 类型变量名

类型变量可用以下两种风格之一进行命名：

单个的大写字母，后面可以跟一个数字(如：E, T, X, T2)。

以类命名方式(5.2.2节)，后面加个大写的T(如：RequestT, FooBarT)。

## 3.3 驼峰式命名法(CamelCase)

驼峰式命名法分大驼峰式命名法(UpperCamelCase)和小驼峰式命名法(lowerCamelCase)。 有时，我们有不只一种合理的方式将一个英语词组转换成驼峰形式，如缩略语或不寻常的结构(例如"IPv6"或"iOS")。Google指定了以下的转换方案。

名字从散文形式(prose form)开始:

把短语转换为纯ASCII码，并且移除任何单引号。例如："Müller’s algorithm"将变成"Muellers algorithm"。

把这个结果切分成单词，在空格或其它标点符号(通常是连字符)处分割开。

推荐：如果某个单词已经有了常用的驼峰表示形式，按它的组成将它分割开(如"AdWords"将分割成"ad words")。 需要注意的是"iOS"并不是一个真正的驼峰表示形式，因此该推荐对它并不适用。

现在将所有字母都小写(包括缩写)，然后将单词的第一个字母大写：

每个单词的第一个字母都大写，来得到大驼峰式命名。

除了第一个单词，每个单词的第一个字母都大写，来得到小驼峰式命名。

最后将所有的单词连接起来得到一个标识符。

# 4.编程实践

## 4.1 @Override：能用则用

只要是合法的，就把@Override注解给用上。

## 4.2 捕获的异常：不能忽视

除了下面的例子，对捕获的异常不做响应是极少正确的。(典型的响应方式是打印日志，或者如果它被认为是不可能的，则把它当作一个AssertionError重新抛出。)

如果它确实是不需要在catch块中做任何响应，需要做注释加以说明(如下面的例子)。

try {

int i = Integer.parseInt(response);

return handleNumericResponse(i);

} catch (NumberFormatException ok) {

// it's not numeric; that's fine, just continue

}

return handleTextResponse(response);

**例外**：在测试中，如果一个捕获的异常被命名为expected，则它可以被不加注释地忽略。下面是一种非常常见的情形，用以确保所测试的方法会抛出一个期望中的异常， 因此在这里就没有必要加注释。

try {

emptyStack.pop();

fail();

} catch (NoSuchElementException expected) {

}

## 4.3 静态成员：使用类进行调用

使用类名调用静态的类成员，而不是具体某个对象或表达式。

Foo aFoo = ...;

Foo.aStaticMethod(); // good

aFoo.aStaticMethod(); // bad

somethingThatYieldsAFoo().aStaticMethod(); // very bad