信大捷安

保密级别	公开
版 本 号	V1.0
文档编号	

移动警务 Android 应用

开发指南

目录

1. 前言 1
   1.1. 术语说明 1
   1.2. 示例代码 1
2. 源文件基础 1
   2.1. 文件编码 1
   2.2. 特殊字符 1
      2.2.1. 空白字符 1
      2.2.2. 特殊转义序列 2
      2.2.3. 非 ASCII 字符 2
3. 源文件结构 2
   3.1. 许可证或版权信息 3
   3.2. package 语句 3
   3.3. import 语句 3
   3.4. 类声明 3
4. 术语格式 5
   4.1. 大括号 5
      4.1.1. 使用大括号 5
      4.1.2. 非空块 K&R 风格 5
      4.1.3. 空块：可以简洁版本 6
   4.2. 块缩进 7
   4.3. 一行一个语句 7
   4.4. 列限制 7
   4.5. 自动换行 7
   4.6. 空白 8
      4.6.1. 垂直空白 8
      4.6.2. 水平空白 9
      4.6.3. 水平对齐：不做要求 9

4.7. 用小括号来限定组：推荐 10
4.8. 具体结构 10
4.8.1. 枚举类 10
4.8.2. 变量声明 11
4.8.3. 数组 11
4.8.4. switch 语句 12
4.8.5. 注解(Annotations) 13
4.8.6. 注释 14
5. 命名约定 16
5.1. 对所有标识符都通用的规则 16
5.2. 标识符类型的规则 16
5.2.1. 包名 16
5.2.2. 类名 17
5.2.3. 方法名 18
5.2.4. 非常量字段命名 19
5.2.5. 常量命名 20
5.2.6. 参数名 21
5.2.7. 局部变量名 21
5.2.8. 异常命名 21
5.2.9. 资源命名 21
6. 编程实践 24
6.1. 规约 24
6.1.1. 方法 24
6.1.2. 参数和返回值 25
6.1.3. 神秘的数 25
6.1.4. 控制语句 25
6.1.5. 异常的捕捉处理 26
6.1.6. 访问控制 26

6.1.7. 分支语句.....27
6.1.8. 循环语句.....27
6.1.9. 日志信息.....28
6.2. 约定俗成.....28
6.2.1. 变量赋值.....28
6.2.2. 圆括号.....29
6.2.3. 返回值.....29
6.2.4. 条件运算符"?"前的表达式.....30
6.3. 21 种代码的坏味道.....30
1. Duplicated Code（重复代码）.....30
2. Long method（过长函数）.....30
3. Large Class（过大的类）.....30
4. Divergent Change（发散式变化）.....31
5. Shotgun Surgery（霰弹式修改）.....31
6. Feature Envy（依恋情结）.....31
7. Data Clumps（数据泥团）.....31
8. Primitive Obsession（基本类型偏执）.....31
9. Switch Statement（switch 惊悚显示）.....32
10. Parallel Inheritance Hierarchies（平行继承体系）.....32
11. Lazy Class（冗赘类）.....32
12. Speculative Generality（夸夸其谈未来性）.....32
13. Temporary Field（令人迷惑的暂时字段）.....32
14. Message Chain（过度耦合的消息链）.....32
15. Middle Man（中间人）.....32
16. Inappropriate Intimacy（狎昵关系）.....33
17. Alternative Classes with Different Interfaces（异曲同工的类）.....33
18. Incomplete Library Class（不完美的类库）.....33
19. Data Class（纯稚的数据类）.....33

20. Refused Bequest（被拒绝的遗赠）.....33
21. Comments（过多的注释）.....33
6.4. Android 建议.....33
附录.....i
表 1 UI 控件缩写表.....i
表 2 常见的英文单词缩写:.....i

文档记录

版本编号	变化状态	简要说明	撰写/变更人	日期
V1.0	C	创建	樊建东	2018-10-19

*变化状态：C—创建、A—增加、M—修改（+修改说明）、D—删除（+删除说明）

1. 前言

本文档参考了《Google Java 编程风格规范》和 《Google 官方 Android 编码风格规范》。与其它的编程风格指南一样，这里所讨论的不仅仅是编码格式美不美观的问题， 同时也讨论一些约定及编码标准，该文档仅供参考，志在建立统一的编码规范。

1.1. 术语说明

在本文档中，除非另有说明:

1. 术语 class 可表示一个普通类，枚举类，接口或是 annotation 类型(@interface)
2. 术语 comment 只用来指代实现的注释(implementation comments)，我们不使 用”documentation comments”一词，而是用 Javadoc。其他的术语说明会偶尔在后面的文档出现。

1.2. 示例代码

1. 指南说明本文档中的示例代码并不作为规范，仅供参考。
2. 基本格式方面使用 Android Studio 默认模板即可（使用格式化快捷键处理后基本符合）。

2. 源文件基础

2.1. 文件编码

源文件编码格式为 UTF-8。

2.2. 特殊字符

2.2.1. 空白字符

除了行结束符序列，ASCII 水平空格字符(0 × 20，即空格)是源文件中唯一允许出现的空白

字符，这意味着:

1. 所有其它字符串中的空白字符都要进行转义。
2. 制表符不用于缩进（可以在 IDE 中 Tab 键设置为若干个空格）。

2.2.2. 特殊转义序列

对于具有特殊转义序列的任何字符(\b, \t, \n, \f, \r, \”, \’ 及)，我们使用它的转义序列，而不是相应的八进制(比如\012)或 Unicode(比如\u000a)转义。

2.2.3. 非 ASCII 字符

对于剩余的非 ASCII 字符，是使用实际的 Unicode 字符(比如∞)，还是使用等价的 Unicode 转义符(比如\u221e)，取决于哪个能让代码更易于阅读和理解。

Tip:在使用 Unicode 转义符或是一些实际的 Unicode 字符时，建议做些注释给出解释，这有助于别人阅读和理解。

例如:

String unitAbbrev = " μ s"; | 赞，即使没有注释也非常清晰

String unitAbbrev = "\u03bcs"; // " μ s" | 允许，但没有理由要这样做

String unitAbbrev = "\u03bcs"; // Greek letter mu, "s" | 允许，但这样做显得笨拙还容易出错

String unitAbbrev = "\u03bcs"; | 很糟，读者根本看不出这是什么

return '\ufeff' + content; // byte order mark | Good，对于非打印字符，使用转义，并在必要时写上注释

Tip: 永远不要由于害怕某些程序可能无法正确处理非 ASCII 字符而让你的代码可读性变差。当程序无法正确处理非 ASCII 字符时，它自然无法正确运行， 你就会去 fix 这些问题的了。(言下之意就是大胆去用非 ASCII 字符，如果真的有需要的话)

3. 源文件结构

一个源文件包含(按顺序地):

1. 许可证或版权信息(如有需要)
2. package 语句
3. import 语句
4. 一个顶级类(只有一个)以上每个部分之间用一个空行隔开。

以上每个部分之间用一个空行隔开。

3.1. 许可证或版权信息

如果一个文件包含许可证或版权信息，那么它应当被放在文件最前面。

3.2. package 语句

package 语句不换行 (即 package 语句写在一行里)

3.3. import 语句

1. import 不要使用通配符

即，不要出现类似这样的 import 语句：import java.util.*;

2. 不要换行

import 语句不换行，列限制(4.4 节)并不适用于 import 语句。(每个 import 语句独立成行)

3. 顺序和间距

import 语句可分为以下几组，按照这个顺序，每组由一个空行分隔:

- 所有的静态导入独立成组
- com.google imports(仅当这个源文件是在 com.google 包下)
- 第三方的包。每个顶级包为一组，字典序。例如：android, com, junit, org, sun
- java imports5.javax imports 组内不空行，按字典序排列。

3.4. 类声明

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)。

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

2. 类成员顺序

类的成员顺序对易学性有很大的影响，但这也不存在唯一的通用法则。不同的类对成员的排序可能是不同的。

最重要的一点，每个类应该以某种逻辑去排序它的成员，维护者应该要能解释这种排序逻辑。比如， 新的方法不能总是习惯性地添加到类的结尾，因为这样就是按时间顺序而非某种逻辑来排序的。

2.1. 区块划分

建议使用注释将源文件分为明显的区块，区块划分如下

- 常量声明区
- UI 控件成员变量声明区
- 普通成员变量声明区
- 内部接口声明区
- 初始化相关方法区
- 事件响应方法区
- 普通逻辑方法区
- 重载的逻辑方法区
- 发起异步任务方法区
- 异步任务回调方法区
- 生命周期回调方法区（出去 onCreate()方法）
- 内部类声明区

2.2.类成员排列通用规则

- 按照发生的先后顺序排列
- 常量按照使用先后排列
- UI 控件成员变量按照 layout 文件中的先后顺序排列

- 普通成员变量按照使用的先后顺序排列
- 方法基本上都按照调用的先后顺序在各自区块中排列
- 相关功能作为小区块放在一起（或者封装掉）

2.3.重载：永不分离

当一个类有多个构造函数，或是多个同名方法，这些函数/方法应该按顺序出现在一起，中间不要放进其它函数/方法。

4. 术语格式

说明：块状结构(block-like construct)指的是一个类，方法或构造函数的主体。需要注意的是，数组初始化中的初始值可被选择性地视为块状结构(第 4.8.3.1 节)。

4.1. 大括号

4.1.1. 使用大括号

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。

4.1.2. 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):

- 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

实例:

```
return new MyClass() {
    @Override public void method() {
```

```
        if (condition()) {
            try {
                something();
            } catch (ProblemException e) {
                recover();
            }
        }
    }
};
```

第 4.8.1 节给出了 enum 类的一些例外。

当一个表达式超过指定行宽时，根据这些通用原则进行处理：

- 在逗号后换行。
- 在操作符前换行。

方法调用换行示例：

```
public void function(int a, int b,int c)
{
 }
```

算术表达式换行示例：

```
String sql = "execsql1" + "execsql2"
           + "execsql3"
           + "execsql4";
```

4.1.3. 空块：可以简洁版本

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。例如：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() {}
```

4.2. 块缩进

每当开始一个新的块，缩进增加 4 个空格，当块结束时，缩进返回先前的缩进级别。缩进级别适用于代码和注释。

4.3. 一行一个语句

每个语句后要换行。

4.4. 列限制

80 或 100

无特别规定，因为现在显示器都比较大，所有推荐使用 100 进行设置。

例如:

1. 不可能满足列限制的行(例如，Javadoc 中的一个长 URL，或是一个长的 JSNI 方法参考)。
2. package 和 import 语句(见 3.2 节和 3.3 节)。
3. 注释中那些可能被剪切并粘贴到 shell 中的命令行

4.5. 自动换行

术语说明：一般情况下，一行长代码为了避免超出列限制(80 或 100 个字符)而被分为多行，我们称之为自动换行(line-wrapping)。我们并没有全面，确定性的准则来决定在每一种情况下如何自动换行。很多时候，对于同一段代码会有好几种有效的自动换行方式。

Tip:提取方法或局部变量可以在不换行的情况下解决代码过长的问题(是合理缩短命名长度吧)

1. 断开位置

自动换行的基本准则是：更倾向于在更高的语法级别处断开。

- 如果在非赋值运算符处断开，那么在该符号前断开(比如+，它将位于下一行)。注意：这一点与 Google 其它语言的编程风格不同(如 C++ 和 JavaScript)。

- 这条规则也适用于以下”类运算符”符号：点分隔符(.)，类型界限中的 &（），catch 块中的管道符号(catch (FooException | BarException e)
- 如果在赋值运算符处断开，通常的做法是在该符号后断开(比如=，它与前面的内容留在同一行)。这条规则也适用于 foreach 语句中的分号。
- 方法名或构造函数名与左括号留在同一行。
- 逗号(,)与其前面的内容留在同一行。

2. 自动换行缩进

自动换行时，第一行后的每一行至少比第一行多缩进 8 个空格(注意：制表符不用于缩进。见 2.3.1 节)。当存在连续自动换行时，缩进可能会多缩进不只 8 个空格(语法元素存在多级时)。一般而言，两个连续行使用相同的缩进当且仅当它们开始于同级语法元素。

第 4.6.3 水平对齐一节中指出，不鼓励使用可变数目的空格来对齐前面行的符号。

4.6. 空白

4.6.1. 垂直空白

以下情况需要使用一个空行:

- 类内连续的成员之间：字段，构造函数，方法，嵌套类，静态初始化块，实例初始化块。 例外： 两个连续字段之间的空行是可选的，用于字段的空行主要用来对字段进行逻辑分组。
- 在函数体内，语句的逻辑分组间使用空行。
- 类内的第一个成员前或最后一个成员后的空行是可选的(既不鼓励也不反对这样做，视个人喜好而定)。
- 要满足本文档中其他节的空行要求(比如 3.3 节：import 语句)
- 多个连续的空行是允许的，但没有必要这样做(我们也不鼓励这样做)。

4.6.2. 水平空白

除了语言需求和其它规则，并且除了文字，注释和 Javadoc 用到单个空格，单个 ASCII 空格也出现在以下几个地方:

1. 分隔任何保留字与紧随其后的左括号(()(如 if, for catch 等)。
2. 分隔任何保留字与其前面的右大括号(})(如 else, catch)。
3. 在任何左大括号前({)，两个例外:
   - @SomeAnnotation({a, b})(不使用空格)。
   - String[][] x = foo;(大括号间没有空格，见下面的 Note)。
4. 在任何二元或三元运算符的两侧。这也适用于以下"类运算符"符号:
   - 类型界限中的&()。
   - catch 块中的管道符号(catch (FooException | BarException e)。
   - foreach 语句中的分号。
5. 在,:;及右括号())后
6. 如果在一条语句后做注释，则双斜杠(//)两边都要空格。这里可以允许多个空格，但没有必要。
7. 类型和变量之间: List list。
8. 数组初始化中，大括号内的空格是可选的，即 new int[] {5, 6}和 new int[] { 5, 6 }都是可以的。

Note: 这个规则并不要求或禁止一行的开关或结尾需要额外的空格，只对内部空格做要求。

4.6.3. 水平对齐: 不做要求

术语说明: 水平对齐指的是通过增加可变数量的空格来使某一行的字符与上一行的相应字符对齐。这是允许的(而且在不少地方可以看到这样的代码)，但 Google 编程风格对此不做要求。即使对于已经使用水平对齐的代码，我们也不需要去保持这种风格。

以下示例先展示未对齐的代码，然后是对齐的代码:

```
private int x; // this is fine
```

```
private Color color; // this too
```

```
private int       x;              // permitted, but future edits
private Color     color;          // may leave it unaligned
```

Tip: 对齐可增加代码可读性，但它为日后的维护带来问题。考虑未来某个时候，我们需要修改一堆对齐的代码中的一行。这可能导致原本很漂亮的对齐代码变得错位。很可能它会提示你调整周围代码的空白来使这一堆代码重新水平对齐(比如程序员想保持这种水平对齐的风格)。

这就会让你做许多的无用功，增加了 reviewer 的工作并且可能导致更多的合并冲突。

4.7. 用小括号来限定组：推荐

除非作者和 reviewer 都认为去掉小括号也不会使代码被误解，或是去掉小括号能让代码更易于阅读，否则我们不应该去掉小括号。

我们没有理由假设读者能记住整个 Java 运算符优先级表。

4.8. 具体结构

4.8.1. 枚举类

枚举常量间用逗号隔开，换行可选。

没有方法和文档的枚举类可写成数组初始化的格式:

```
    private enum Suit { CLUBS, HEARTS, SPADES, DIAMONDS }
```

由于枚举类也是一个类，因此所有适用于其它类的格式规则也适用于枚举类。

注: Android 官方不建议使用枚举。

4.8.2. 变量声明

4.8.2.1. 每次只声明一个变量

不要使用组合声明，比如 int a, b;。

4.8.2.2. 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。

4.8.3. 数组

4.8.3.1. 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}
```

4.8.3.2. 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[]。

4.8.4. 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。

4.8.4.1. 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。

4.8.4.2. 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);
}
```

4.8.4.3. 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。

4.8.5. 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() { ... }
```

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。

4.8.6. 注释

4.8.6.1. 文件注释

所有的源文件都应该在开头有一个注释，其中列出类名、版本信息、日期和版权声明。

如下:

```
/*
* 文件名
* 包含类名列表
* 版本信息，版本号
* 创建日期。
* 版权声明
*/
```

4.8.6.2. 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。

```
/**
* 实现的主要功能。
* @作者 作者名称
* @创建日期
* <p>修改者，修改日期，修改内容。</p>
*/
```

4.8.6.3. 4.8.6.3 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>
 * @作者 wzg
 * @创建时间 2014-7-15 下午4:08:58
 * @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
 */
```

4.8.6.4. 4.8.6.4 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

```
/**
* XXXX含义
*/
```

注：但对于可见属性定义为private的变量，可使用如下形式注释

```
//XXXX 含义
```

4.8.6.5. 4.8.6.5 xml 注释

规约：如果当前 layout 或资源需要被多处调用，或为公共使用的 layout（若 list_item），则需要在 xml 写明注释。要求注释清晰易懂

4.8.6.6. 4.8.6.6 其他注释

方法内部的注释 如果需要多行 使用/*…… */形式，如果为单行是用//……形式的注

释。不要再方法内部使用 java doc 形式的注释 “/**……**/”，简单的区分方法是，java doc 形式的注释在 eclipse 中为蓝色，普通注释为绿色

5. 命名约定

5.1. 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

5.2. 标识符类型的规则

5.2.1. 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。

采用反域名命名规则，全部使用小写字母。一级包名为 com，edu，gov，mil，net，org，二级包名为 xx，三级包名根据应用进行命名，四级包名为模块名或层级名。

例如：com.xdja.test

包名	此包中包含
com.xx.应用名称缩写.activity	页面用到的 Activity 类 (activitie 层级名用户界面层)
com.xx.应用名称缩写.base	基础共享的类
com.xx.应用名称缩写.adapter	页面用到的 Adapter 类 (适配器的类)
com.xx.应用名称缩写.util	此包中包含：公共工具方法类（util 模块名）
com.xx.应用名称缩写.bean	下面可分：vo、po、dto 此包中包含：JavaBean 类
com.xx.应用名称缩写.model	此包中包含：模型类
com.xx.应用名称缩写.db	数据库操作类
com.xx.应用名称缩写.view (或者 com.xx.应用名称缩写.widget)	自定义的 View 类等
com.xx.应用名称缩写.service	Service 服务
com.xx.应用名称缩写.receiver	BroadcastReceiver 服务

注意:

如果项目采用 MVP，所有 M、V、P 抽取出来的接口都放置在相应模块的 i 包下，所有的实现都放置在相应模块的 impl 下。

5.2.2. 类名

类名都以 UpperCamelCase 风格编写。

- 命名规则：类名是一个名词，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；若是业务接口，一般在名字前加上“I”前缀。

  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。

类	描述	例如
Activity 类 Fragment	Activity 为后缀标识	启动页面 WelcomeActivity
Service 类	以 Service 为后缀标识	时间服务 TimeServiceBroadcast
Receiver 类	以 Receiver 为后缀标识	推送接收 JPushReceiver
ContentProvider	以 Provider 为后缀标识	PhoneBookProvider
自定义的共享基础类	以 Base 开头	BaseActivity,BaseFragment
Adapter 类	Adapter 为后缀标识	新闻详情数据适配器 NewsDetailAdapter
解析类	Parser 为后缀标识	首页解析类 HomePosterParser
工具方法类	Util 或 Manager 为后缀标识（与系统或第三方的 Utils 区分）或功能+Util	线程池管理类: ThreadPoolManager 日志工具类: LogUtil（Logger 也可） 打印工具类: PrinterUtil

数据库类	以 DBHelper 后缀标识	新闻数据库：NewDBHelper

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。

接口（interface）：命名规则与类一样采用大驼峰命名法，多以 able 或 ible 结尾，如 interface Runnable ;interface Accessible。

注意:

如果项目采用 MVP，所有 Model、View、Presenter 的接口都以 I 为前缀，不加后缀，其他的接口采用上述命名规则。

5.2.3. 方法名

方法名都以 LowerCamelCase 风格编写。

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：

方法	说明
initXX()	初始化相关方法,使用 init 为前缀标识，如初始化布局 initView()
isXX() checkXX()	方法返回值为 boolean 型的请使用 is 或 check 为前缀标识
getXX()	返回某个值的方法，使用 get 为前缀标识
handleXX()	对数据进行处理的方法，尽量使用 handle 为前缀标识
displayXX() showXX()	弹出提示框和提示信息，使用 display/show 为前缀标识
saveXX()	与保存数据相关的，使用 save 为前缀标识
resetXX()	对数据重组的，使用 reset 前缀标识
clearXX()	清除数据相关的
removeXXX()	清除数据相关的

drawXXX()	绘制数据或效果相关的，使用 draw 前缀标识

- 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。
- 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().
- 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。
- 类的普通方法一般采用完整的英文描述说明成员方法功能，第一个单词尽可能采用动词，首字母小写，如 openFile ( )，addCount ( )。
- 构造方法应该用递增的方式写。(参数多的写在后面)。

5.2.4. 非常量字段命名

非常量字段名以 LowerCamelCase 风格的基础上改造为如下风格:

基本结构为 scopeVariableNameType

scope: 范围

1. 非公有，非静态字段命名以 m 开头。
2. 静态字段命名以 s 开头。
3. 公有非静态字段命名以 p 开头。
4. 公有静态字段（全局变量）命名以 g 开头。
5. public static final 字段(常量) 全部大写，并用下划线连起来。

```
public class MyClass {

      public static final int SOME_CONSTANT = 42;

      public int pField;

      private static MyClass sSingleton;

      int mPackagePrivate;
```

```
    private int mPrivate;

    protected int mProtected;

    public static int gField;

}
```

Type：类型

考虑到 Android 中使用很多 UI 控件，为避免控件和普通成员变量混淆以及更好达意，所有用来表示控件的成员变量统一加上控件缩写作为后缀。

对于普通变量一般不添加类型后缀，如果统一添加类型后缀。

用统一的量词通过在结尾处放置一个量词，就可创建更加统一的变量，它们更容易理解，也更容易搜索。

注意:

如果项目中使用 ButterKnife，则不添加 m 前缀，以 LowerCamelCase 风格命名。

例如，请使用 mCustomerStrFirst 和 mCustomerStrLast，而不要使用 mFirstCustomerStr 和 mLastCustomerStr。

量词列表：量词后缀说明

First 一组变量中的第一个

Last 一组变量中的最后一个

Next 一组变量中的下一个变量

Prev 一组变量中的上一个

Cur 一组变量中的当前变量。

5.2.5. 常量命名

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。

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;

5.2.6. 参数名

参数名以 LowerCamelCase 风格编写。

5.2.7. 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。

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。 如： for (int i = 0; i < len ; i++)，并且它和第一个单词间没有空格。

5.2.8. 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。

5.2.9. 资源命名

1. 资源布局文件（XML 文件（layout 布局文件））:

   全部小写，采用下划线命名法

   1) contentview 命名

      必须以全部单词小写，单词间以下划线分割，使用名词或名词词组。

      所有 Activity 或 Fragment 的 contentView 必须与其类名对应，对应规则为：将所有字母都转为小写，将类型和功能调换（也就是后缀变前缀）。例如：activity_main.xml

   2) Dialog 命名：dialog_描述.xml

      例如：dialog_hint.xml

3) PopupWindow 命名：ppw_描述.xml

例如：ppw_info.xml

4) 列表项命名：item_描述.xml

例如：item_city.xml

5) 包含项命名：模块_(位置)描述.xml

例如：activity_main_head.xml、activity_main_bottom.xml

注意：通用的包含项命名采用：项目名称缩写_描述.xml

例如：xxxx_title.xml

2. 资源文件（图片 drawable 文件夹下）:

全部小写，采用下划线命名法，加前缀区分

命名模式：可加后缀 _small 表示小图, _big 表示大图，逻辑名称可由多个单词加下划线组成，采用以下规则：

- 用途_模块名_逻辑名称
- 用途_模块名_颜色
- 用途_逻辑名称
- 用途_颜色

说明：用途也指控件类型（具体见 UI 控件缩写表）

例如:

btn_main_home.png 按键

divider_maket_white.png 分割线

ic_edit.png 图标

bg_main.png 背景

btn_red.png 红色按键

btn_red_big.png 红色大按键

ic_head_small.png 小头像

bg_input.png 输入框背景

divider_white.png 白色分割线

如果有多种形态如按钮等除外如 btn_xx.xml（selector）

名称	功能
btn_xx	按钮图片使用 btn_整体效果（selector）
btn_xx_normal	按钮图片使用 btn_正常情况效果
btn_xx_pressed	按钮图片使用 btn_点击时候效果
btn_xx_focused	state_focused 聚焦效果
btn_xx_disabled	state_enabled (false)不可用效果
btn_xx_checked	state_checked 选中效果
btn_xx_selected	state_selected 选中效果
btn_xx_hovered	state_hovered 悬停效果
btn_xx_checkable	state_checkable 可选效果
btn_xx_activated	state_activated 激活的
btn_xx_windowfocused	state_window_focused
bg_head	背景图片使用 bg_功能_说明
def_search_cell	默认图片使用 def_功能_说明
ic_more_help	图标图片使用 ic_功能_说明
seg_list_line	具有分隔特征的图片使用 seg_功能_说明
sel_ok	选择图标使用 sel_功能_说明

注意：

使用 AndroidStudio 的插件 SelectorChapek 可以快速生成 selector，前提是命名要规范。

3. 动画文件（anim 文件夹下）

全部小写，采用下划线命名法，加前缀区分。

具体动画采用以下规则:

- 模块名_逻辑名称
- 逻辑名称

refresh_progress.xml

4. values 中 name 命名

类别	命名	示例
strings	strings 的 name 命名使用下划线命名法，采用以下规则：模块名+逻辑名称	main_menu_about 主菜单按键文字 friend_title 好友模块标题栏 friend_dialog_del 好友删除提示 login_check_email 登录验证 dialog_title 弹出框标题 button_ok 确认键 loading 加载文字
colors	colors 的 name 命名使用下划线命名法，采用以下规则:	friend_info_bg friend_bg transparent gray

	模块名+逻辑名称 颜色	
styles	styles 的 name 命名使用 Camel 命名法，采用以下规则：模块名+逻辑名称	main_tabBottom

5. layout 布局中 id 命名

命名模式为：view 缩写_view 的逻辑名称

使用 AndroidStudio 的插件 ButterKnife Zelezny，生成注解非常方便。

如果不使用 ButterKnife Zelezny，则建议使用 view 缩写做后缀，如：username_tv（展示用户名的 TextView）

6. 编程实践

6.1. 规约

6.1.1. 方法

- 一个方法尽量不要超过 80 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。
- 要确保函数只做一件事，函数中的语句都要在同一抽象层次上。
- 避免输出参数。即避免函数通过参数传递结果。函数参数应该是函数的输入。
- 函数要么做什么事，要么回答什么事，但二者不可兼得。函数应该修改某对象的状态，或是返回该对象的有关信息。两者都做常会导致混乱。
- 某个函数调用了另一个，就应该把它们放到一起，而且调用者应该尽可能放在被调用者上面。这样就能轻易找到被调用的函数，极大地增强了整个模块的可读性。
- 准确地确定成员变量、成员函数的存取控制符号，不是必须使用 public 属性的，请使用 protected，不是必须使用 protected，请使用 private。
- 不要使用 try catch 处理业务逻辑!!!!

注：Eclipse 中把代码块封装方法的快捷键： Alt+ Shitf + M

6.1.2. 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()

6.1.3. 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!（for 语句中除外）

6.1.4. 控制语句

If 语句嵌套应该小于两层，如果超过，则应考虑把代码块封装为方法。

判断中如有常量，则应将常量置于判断式的右侧。如:

```
if ( true == isAdmin()) ...
```

尽量不使用三目条件的嵌套。

所有 if 语句必须用{}包括起来,即便是只有一句:

```
if (true){
//do something......
}
if (true)
i = 0; //不要使用这种
```

对于循环:

//不推荐方式_____

```
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
}
//推荐方式_____________________________________________
//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){
}
```

6.1.5. 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。
- 不要捕捉了异常却什么也不做，除非有特殊需求。如果隐藏了一个异常，将永远不知道异常到底发生了没有。
- 异常捕获后，如果不对该异常进行处理，则应该记录日志或 e.printStackTrace()。但不应捕获了异常却什么都不做。

6.1.6. 访问控制

- 若没有足够理由，不要把实例或类变量声明为公有。通常，实例变量无需显式的设置(setter)和获取(getter)，通常这作为方法调用的边缘效应 (side effect)而产生。
- 一个具有公有实例变量的恰当例子，是类仅作为数据结构，没有行为。亦即，若你要使用一个结构(struct)而非一个类(如果 java 支持结构的话)，那么把类的实例变量声明为公

有是合适的。

6.1.7. 分支语句

- 每个 case 语句的结尾不要忘了加 break，否则将导致多个分支重叠（除非有意使多个分支重叠）。
- 不要忘记最后那个 default 分支，即使程序真的不需要 default 处理，也应该保留语句 default 。
- 不要在 case 分支中定义变量；如果定义变量，应在所有 case 分支之前定义。
- Case 语句的处理代码，如果超过三行，应该把处理代码封装为方法。

6.1.8. 循环语句

- 在多重循环中，如果有可能，应当将最长的循环放在最内层，最短的循环放在最外层，以减少 CPU 跨切循环层的次数。如:

```
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 50; j++) {


    }
}
```

- 不可在 for 循环体内修改循环变量，防止 for 循环失去控制 。
- for 循环的数组大小应在外部声明变量，避免在循环中再次获取数组长度。
- for 语句的循环控制变量的取值采用“半开半闭区间”写法,即假设我们有一个变量 i 要循环 50 次，则我们要求采取以下的 for 语句写法:

```
for (int j = 0; j < 50; j++) {


}
```

而不要采取以下的写法:

```
for (int j = 0; j <= 49; j++) {
```

```
}
```

6.1.9. 日志信息

- 正常代码执行便于分析问题日志信息打印级别控制在 v、d 和 i 级别日志信息，如果程序异常捕获或者由于某种现象出现的导致较为严重的逻辑，可以打印 w 和 e 级别的日志信息。
- 便于分析完的问题，问题解决以后，应该移除为调试问题而打印的日志。
- 如果是发布后也需打印的日志信息，则日志 Tag 请使用应用统一标签。
- 日志打印中的 TAG 不能出现敏感数据

  反例:Log.e(“zhangsan”,””+a);
- release 版本不要打印 debug 时期的日志数据
- 建议可打印所在的方法和行号，便于开发人员查看 Log 信息

6.2. 约定俗成

6.2.1. 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;
```

6.2.2. 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
```

6.2.3. 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);
```

6.2.4. 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`

6.3. 21 种代码的坏味道

应该在编程中尽量避免这 21 种“坏味道”。

1. Duplicated Code（重复代码）

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。

2. Long method（过长函数）

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。

3. Large Class（过大的类）

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。

4. Divergent Change（发散式变化）

一个类里面的内容变化率不同。某些状态一个小时变一次，某些则几个月一年才变一次；某些状态因为这方面的原因发生变化，而另一些则因为其他方面的原因变一次。面向对象的抽象就是把相对不变的和相对变化相隔离。把问题变化的一方面和另一方面相隔离。这使得这些相对不变的可以重用。问题变化的每个方面都可以单独重用。这种相异变化的共存使得重用非常困难。

5. Shotgun Surgery（霰弹式修改）

这正好和上面相反。对系统一个地方的改变涉及到其他许多地方的相关改变。这些变化率和变化内容相似的状态和行为通常应当放在同一个类中。

6. Feature Envy（依恋情结）

对象的目的就是封装状态以及与这些状态紧密相关的行为。如果一个类的方法频繁用 get 方法存取其他类的状态进行计算，那么你要考虑把行为移到涉及状态数目最多的那个类。

7. Data Clumps（数据泥团）

某些数据通常像孩子一样成群玩耍：一起出现在很多类的成员变量中，一起出现在许多方法的参数中……，这些数据或许应该自己独立形成对象。

8. Primitive Obsession（基本类型偏执）

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。

9. Switch Statement（switch 惊悚显示）

基于常量的开关语句是 OO 的大敌，你应当把他变为子类、state 或 strategy。

10. Parallel Inheritance Hierarchies（平行继承体系）

并行的继承层次是 shotgun surgery 的特殊情况。因为当你改变一个层次中的某一个类时，你必须同时改变另外一个层次的并行子类。

11. Lazy Class（冗赘类）

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。

12. Speculative Generality（夸夸其谈未来性）

一个类实现了从未用到的功能和通用性。通常这样的类或方法唯一的用户是 testcase。不要犹豫，删除它。

13. Temporary Field（令人迷惑的暂时字段）

一个对象的属性可能只在某些情况下才有意义。这样的代码将难以理解。专门建立一个对象来持有这样的孤儿属性，把只和他相关的行为移到该类。最常见的是一个特定的算法需要某些只有该算法才有用的变量。

14. Message Chain（过度耦合的消息链）

消息链发生于当一个客户向一个对象要求另一个对象，然后客户又向这另一对象要求另一个对象，再向这另一个对象要求另一个对象，如此如此。这时，你需要隐藏分派。

15. Middle Man（中间人）

对象的基本特性之一就是封装，而你经常会通过分派去实现封装。但是这一步不能走得

太远，如果你发现一个类接口的一大半方法都在做分派，你可能需要移去这个中间人。

16. Inappropriate Intimacy（狎昵关系）

某些类相互之间太亲密，它们花费了太多的时间去砖研别人的私有部分。对人类而言，我们也许不应该太假正经，但我们应当让自己的类严格遵守禁欲主义。

17. Alternative Classes with Different Interfaces（异曲同工的类）

做相同事情的方法有不同的函数 signature，一致把它们往类层次上移，直至协议一致。

18. Incomplete Library Class（不完美的类库）

要建立一个好的类库非常困难。我们大量的程序工作都基于类库实现。然而，如此广泛而又相异的目标对库构建者提出了苛刻的要求。库构建者也不是万能的。有时候我们会发现库类无法实现我们需要的功能。而直接对库类的修改有非常困难。这时候就需要用各种手段进行 Refactoring。

19. Data Class（纯稚的数据类）

对象包括状态和行为。如果一个类只有状态没有行为，那么肯定有什么地方出问题了。

20. Refused Bequest（被拒绝的遗赠）

超类传下来很多行为和状态，而子类只是用了其中的很小一部分。这通常意味着你的类层次有问题。

21. Comments（过多的注释）

经常觉得要写很多注释表示你的代码难以理解。如果这种感觉太多，表示你需要 Refactoring。

6.4. Android 建议

1. 慎用枚举

使用 enum 很方便,但是会比使用静态变量产生多于两倍的内存消耗,所以 Android 官方强烈建议不要在 Android 程序里面使用到 enum.

2. 引用第三方库要慎重，避免应用大容量的第三方库，导致客户端包非常大
3. @Override：能用则用
4. 捕获的异常：不能忽视
5. 静态成员：使用类进行调用
6. Finalizers：禁用

附录

表 1 UI 控件缩写表

控件	缩写	例子
LinearLayout	ll	llFriend 或者 mFriendLL
RelativeLayout	rl	rlMessage 或 mMessageRL
FrameLayout	fl	flCart 或 mCartFL
TableLayout	tl	tlTab 或 mTabTL
Button	btn	btnHome 或 mHomeBtn
ImageButton	ibtn	btnPlay 或 mPlayIBtn
TextView	tv	tvName 或 mNameTV
EditText	et	etName 或 mNameET
ListView	lv	lvCart 或 mCartLV
ImageView	iv	ivHead 或 mHeadIV
GridView	gv	gvPhoto 或 mPhotoGV

表 2 常见的英文单词缩写:

名称	缩写
icon	ic （主要用在 app 的图标）
color	cl（主要用于颜色值）
divider	di（主要用于分隔线，不仅包括 Listview 中的 divider，还包括普通布局中的线）
selector	sl（主要用于某一 view 多种状态，不仅包括 Listview 中的 selector，还包括按钮的 selector）
average	avg
background	bg（主要用于布局和子布局的背景）

buffer	buf
control	ctrl
delete	del
document	doc
error	err
escape	esc
increment	inc
infomation	info
initial	init
image	img
Internationalization	I18N
length	len
library	lib
message	msg
password	pwd
position	pos
server	srv
string	str
temp	tmp
window	wnd(win)