1. 前言

这份文档参考了 Google Java 编程风格规范和 Google 官方 Android 编码风格规范。该文档仅供参考，只要形成一个统一的风格，见量知其意就可。

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 指南说明本文档中的示例代码并不作为规范，仅供参考。

基本格式方面使用 AndroidStudio 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 文件名

源文件以其最顶层的类名来命名，大小写敏感，文件扩展名为.java。

2.2 文件编码：UTF-8

源文件编码格式为 UTF-8。

2.3 特殊字符

2.3.1 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）。

2.3.2 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。

2.3.3 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 = "μs"; | 赞，即使没有注释也非常清晰

String unitAbbrev = "\u03bcs"; // "μs" | 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。

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，列限制(4.4 节)并不适用于 package 语句。(即 package 语句写在一行里)

3.3 import 语句

3.3.1 import 不要使用通配符

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

3.3.2 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)

3.3.3 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：

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

3.4 类声明

3.4.1 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。

3.4.2 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。

3.4.2.1 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

1. 常量声明区
2. UI 控件成员变量声明区
3. 普通成员变量声明区
4. 内部接口声明区
5. 初始化相关方法区
6. 事件响应方法区
7. 普通逻辑方法区
8. 重载的逻辑方法区

9. 发起异步任务方法区
10. 异步任务回调方法区
11. 生命周期回调方法区（出去 onCreate()方法）
12. 内部类声明区

3.4.2.2 类成员排列通用规则

1. 按照发生的先后顺序排列
2. 常量按照使用先后排列
3. UI 控件成员变量按照 layout 文件中的先后顺序排列
4. 普通成员变量按照使用的先后顺序排列
5. 方法基本上都按照调用的先后顺序在各自区块中排列
6. 相关功能作为小区块放在一起（或者封装掉）

3.4.2.3 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. 格式术语

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(4.8.3.1 节)。

4.1 大括号

4.1.1 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。

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。

示例：

```
return new MyClass() {

    @Override public void method() {

        if (condition()) {
```

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

4.8.1 节给出了 enum 类的一些例外。

4.1.3 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() {}
```

4.2 块缩进：4 个空格

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。(见 4.1.2 节中的代码示例)

4.3 一行一个语句

每个语句后要换行。

4.4 列限制：80 或 100

一个项目可以选择一行 80 个字符或 100 个字符的列限制，除了下述例外，任何一行如果超过这个字符数限制，必须自动换行。

例外：

- •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)。
- •package 和 import 语句(见 3.2 节和 3.3 节)。
- 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)

4.5.1 从哪里断开

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。

1. 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)。
2. 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)
3. 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。
4. 方法名或构造函数名与左括号留在同一行。
5. 逗号(,)与其前面的内容留在同一行。

4.5.2 自动换行时缩进至少+8 个空格

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：

1. 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。
2. 在函数体内，语句的逻辑分组间使用空行。
3. 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)。
4. 要满足本文档中其他节的空行要求(比如 3.3 节：import 语句)
5. 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。

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：

```
@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。

4.8.6 注释

4.8.6.1 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/ ... /。

4.8.7 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

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+。

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，二级包名为 xx（可以是公司或则个人的随便），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.jiashuangkuaizi.kitchen

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

包名	此包中包含
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。

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。

名词，采用大驼峰命名法，尽量避免缩写，除非该缩写是众所周知的，比如 HTML,URL，如果类名称中包含单词缩写，则单词缩写的每个字母均应大写。

类	描述	例如
Activity 类	Activity 为后缀标识	欢迎页面类 WelcomeActivity
Adapter 类	Adapter 为后缀标识	新闻详情适配器 NewDetailAdapter
解析类	Parser 为后缀标识	首页解析类 HomePosterParser

类	描述	例如
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
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	
自定义的共享基础类	以 Base 开头	BaseActivity,BaseFragment

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。

方法名通常是动词或动词短语。

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

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_，例如 testPop_emptyStack。

并不存在唯一正确的方式来命名测试方法。

5.2.4 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 ImmutableListNAMES = ImmutableList.of("Ed", "
Ann");

static final Joiner COMMA_JOINER = Joiner.on(','); // beca
use 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 SetmutableCollection = new HashSet();

static final ImmutableSetmutableElements = ImmutableSet.of
(mutable);
```

```
static final Logger logger = Logger.getLogger(MyClass.getN
ame());

static final String[] nonEmptyArray = {"these", "can", "ch
ange"};
```

这些名字通常是名词或名词短语。

5.2.5 非常量字段名

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

非公有，非静态字段命名以 m 开头。

静态字段命名以 s 开头。

公有非静态字段命名以 p 开头。

公有静态字段（全局变量）命名以 g 开头。

public static final 字段(常量) 全部大写，并用下划线连起来。

例子：

```
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;
}
```

使用 1 字符前缀来表示作用范围，1 个字符的前缀必须小写，前缀后面是由表意性强的一个单词或多个单词组成的名字，而且每个单词的首写字母大写，其它字母小写，这样保证了对变量名能够进行正确的断句。

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（文末附有缩写表）。

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，请参考文末的缩写表。

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。

说明：

集合添加如下后缀：List、Map、Set

数组添加如下后缀：Arr

注意：所有的 VO（值对象）统一采用标准的 lowerCamelCase 风格编写，所有的 DTO（数据传输对象）就按照接口文档中定义的字段名编写。

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 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)。

5.2.9 资源文件命名规范

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

名称	功能
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

market_cart_add.xml

market_cart_remove.xml

普通的 tween 动画采用如下表格中的命名方式

// 前面为动画的类型，后面为方向

动画命名例子	规范写法
fade_in	淡入
fade_out	淡出
push_down_in	从下方推入
push_down_out	从下方推出
push_left	推向左方
slide_in_from_top	从头部滑动进入
zoom_enter	变形进入
slide_in	滑动进入
shrink_to_middle	中间缩小

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

类别	命名	示例
		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：模块名+逻辑名称 颜色	friend_info_bg friend_bg transparent gray
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 中的 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 @Override：能用则用

只要是合法的，就把@Override 注解给用上。

6.2 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();
} 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) {
```

```
}
```

6.3 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
```

6.4 Finalizers: 禁用

极少会去重载 Object.finalize。

Tip：

不要使用 finalize。如果你非要使用它，请先仔细阅读和理解 Effective Java 第 7 条款：” Avoid Finalizers” ，然后不要使用它。

7. Javadoc

7.1 格式

7.1.1 一般形式

Javadoc 块的基本格式如下所示：

```
/**
* Multiple lines of Javadoc text are written here,
* wrapped normally...
*/
 public int method(String p1) { ... }
```

或者是以下单行形式：

```
/** An especially short bit of Javadoc. */
```

基本格式总是 OK 的。当整个 Javadoc 块能容纳于一行时(且没有 Javadoc 标记@XXX)，可以使用单行形式。

7.1.2 段落

空行(即，只包含最左侧星号的行)会出现在段落之间和 Javadoc 标记(@XXX)之前(如果有的话)。

除了第一个段落，每个段落第一个单词前都有标签`<p>`，并且它和第一个单词间没有空格。

7.1.3 Javadoc 标记

标准的 Javadoc 标记按以下顺序出现：@param, @return, @throws, @deprecated,

前面这 4 种标记如果出现，描述都不能为空。 当描述无法在一行中容纳，连续行需要至少再缩进 4 个空格。

7.2 摘要片段

每个类或成员的 Javadoc 以一个简短的摘要片段开始。这个片段是非常重要的，在某些情况下，它是唯一出现的文本，比如在类和方法索引中。

这只是一个小片段，可以是一个名词短语或动词短语，但不是一个完整的句子。它不会以 A {@code Foo} is a...或 This method returns...开头,它也不会是一个完整的祈使句，如 Save the record...。然而，由于开头大写及被加了标点，它看起来就像是个完整的句子。

Tip：

一个常见的错误是把简单的 Javadoc 写成 `/** @return the customer ID */`，这是不正确的。它应该写成`/** Returns the customer ID. */`。

7.3 哪里需要使用 Javadoc

至少在每个 public 类及它的每个 public 和 protected 成员处使用 Javadoc，以下是一些例外：

7.3.1 例外：不言自明的方法

对于简单明显的方法如 getFoo，Javadoc 是可选的(即，是可以不写的)。这种情况下除了写“Returns the foo”，确实也没有什么值得写了。

单元测试类中的测试方法可能是不言自明的最常见例子了，我们通常可以从这些方法的描述性命名中知道它是干什么的，因此不需要额外的文档说明。

> Tip：
>
> 如果有一些相关信息是需要读者了解的，那么以上的例外不应作为忽视这些信息的理由。例如，对于方法名 getCanonicalName，

就不应该忽视文档说明，因为读者很可能不知道词语 canonical name 指的是什么。

7.3.2 例外：重载

如果一个方法重载了超类中的方法，那么 Javadoc 并非必需的。

7.3.3 可选的 Javadoc

对于包外不可见的类和方法，如有需要，也是要使用 Javadoc 的。

如果一个注释是用来定义一个类，方法，字段的整体目的或行为，那么这个注释应该写成 Javadoc，这样更统一更友好。

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）

名称	缩写
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)

程序中使用单词缩写原则：不要用缩写，除非该缩写是约定俗成的。