JAVA 编码规范

魏水锋

版 本 历 史

版本/状态	责任人	起止日期	备注
V0.1/	魏水锋	06Sep17	创建文档，起草大纲。

目 录

1. 说明

为统一 Java 编码规范，提高可读性，特制定本规范。

本规范只包含最基本的 Java 编码规范，尽可能排除有歧义、操作性不强的规范，也不包含编程实践，编程实践由独立的文档给出。

本规范适用于通用 Java 开发、Java Fx 开发、JEE 开发和 Android 开发。

本规范主要参考Google Java Style。

1.1 术语

除非特别预定，我们适用以下术语：

- 类用于指代所有的类(class）、枚举(enum class)、接口(interface)以及注解(annotation)。
- 注释仅用于指代码行间注释，Javadoc 用于指代文档注释。

2. 源文件规范

2.1 文件名

源文件名必须和它包含的顶层类名保持一致，包括大小写，并以.java 作为后缀名。

2.2 文件编码

所有源文件编码必须是 UTF-8。

2.3 特殊字符

2.3.1 空格

除了换行符之外，ASCII 空格（0x20）是唯一合法的空格字符。这意味着：

- 所有在源代码中（包括字符、字符串）出现的其他空格字符需要转义,例如 Tab 用\t 表示。
- 缩进必须使用空格而不是 Tab。

2.3.2 特殊转义字符

对于有特殊转义表示的字符（\b, \t, \n, \f, \r, \", \', \\），禁止使用其它等价转义方式。例如\012 或者\u00a 表示。

2.3.3 非 ASCII 字符

对于非 ASCII 字符，可以使用实际字符（如∞）或者它的 Unicode 转义（如\u221e），取决于哪种写法的可读性更好。

建议：使用注释有助于增强可读性。

示例	说明

String unitAbbrev = “μs”;	最佳写法，无需注释就可以理解
String unitAbbrev = “\u03bcs”; // μs	合法，但是没有必要
String unitAbbrev = “μs”; // Greek letter mu, “s”	合法，但是不容易理解
return ‘\ufeff’ + content; // byte order mark	很好的写法，用 Unicode 转义标示非打印字符，并有很好的注释

3. 源文件组织结构

源文件必须按顺序由以下部分组成：

1. 许可证（License）或版权声明（Copyright）

2. package 语句

3. import 语句

4. 唯一的顶层类

每两部分之间用一个空行分隔。

3.1 许可证（License）或版权声明（Copyright）

如果文件有许可证（License）或版权声明（Copyright），放在最开头。如果没有的话，此部分可以忽略。

3.2 package 语句

package 语句占据单独一行不换行，允许超出 120 字符列宽限制。

3.3 import 语句

3.3.1 禁止通配符 import

无论是 static 还是非 static imports，均禁止使用通配符 import。

3.3.2 不换行

每条 import 语句占据单独一行不换行，允许超出 120 字符列宽限制。

3.3.3 顺序

import 语句需按照一定的逻辑顺序组织。

可参考以下的组织形式，不强制：

按以下顺序进行分组，每两组之间用一个空行分隔。

1. 所有的 static import 语句

2. com.tplink import 语句（仅当源文件属于 com.tplink 时适用）

3. 第三方包。每个顶层包独立一组，按 ASCII 顺序排列。例如：android, com, junit, org, sun

4. java import 语句

5. javax import 语句

组内不包含空行，按照所 import 的包名的 ASCII 码顺序排列。

3.3.4 删除未使用的 import 语句

所有未使用的 import 语句应该被删除。

3.4 类声明

3.4.1 唯一的顶层类

每个源文件只允许包含唯一一个顶层类。

以下写法虽然符合 Java 语法，但禁止使用

```
public class A {
}
class B {
}
```

必须把 B 放在一个独立的源文件中

3.4.2 类成员顺序：有一定的逻辑顺序即可

不规定严格的类成员顺序，但需要遵循一定逻辑规律，让阅读者容易理解。例如不建议把新添加的方法一律放在最后，而是应该插入到合适的地方。

3.4.2.1 重载（Overload）方法必须放在一起

重载的方法（同名的构造函数或方法）应该按序排列，之间禁止插入其他成员。

4. 代码格式

4.1 花括号

4.1.1 不得省略花括号

在 if、else、for、do 和 while 语句中，即使没有语句或者只有一行，也不得省略花括号。

4.1.2 非空块中花括号的使用

在非空代码块中使用花括号时要遵循 K&R 风格（Kernighan and Ritchie Style）：

1. 左花括号（{）前不能换行，在其后换行。

2. 在右花括号（}）前要有换行。

3. 如果右花括号是一句语句、一个方法、构造函数或非匿名类的结尾，其后需要换行。

```
return new MyClass() { // 左花括号前不能换行，在其后换行
    @Override
    public void method() {
        if (condition()) {
```

```
            try {
                do {
                    something();
                } while (!stop()); // do-while 中间的右花括号后
            } catch (ProblemException e) { // try-catch 中间的右花括号后
无需换行
                recover();
            } // try-catch 结束，右花括号后需要换行
        } else { // if-else 中间的右花括号后无需换行
            doSomethingElese();
        } // if-else 结束，右花括号后需要换行
    }
}; // 匿名类结尾的右花括号后无需换行
```

4.1.3 空代码块中花括号的使用

如果一个代码块是空的，可以直接使用{}。除了 if/else-if/else 和 try/catch/finally 这样的多块语句：

```
EmptyConstructor() {}
```

4.2 块缩进：4 个空格

每次开始书写一个新的代码块时，使用 4 个空格进行缩进，在代码块结束时，恢复之前的缩进级别。

4.3 每行只写一条语句

每条语句之后都要换行。

4.4 列宽：120 字符

列宽必须为 120 字符，以下情况可以不遵守列宽限制：

1. 无法限制宽度的内容，比如注释里的长 URL
2. package 和 import 语句
3. 注释中需要被粘贴到 Shell 里去执行的命令

4.5 换行

换行指的是一行语句由于某些限制（例如列宽）需要分为多行的情况。

4.5.1 何处换行

换行原则：尽量在高层次的语法元素处换行。此外：

1. 在非赋值操作符前换行，该原则也适用于这些符号：点（.），泛型类型绑定中的与符号（<T extends Foo & Bar>），catch 中的竖线（catch (FooException | BarException e)）
2. 在赋值操作符（=）后换行，该原则也适用于 foreach 语句中的:。例外：在给注解（Annotation）中的数组类型参数赋值时，应参考数组初始化形式，优先选择在左花括号（{）后面换行，详见下面的例子。

3. 方法、构造函数名字和左圆括号（(）之间不换行

4. 逗号（,）紧跟前面的内容不换行

4.5.2 至少用 8 个以上的空格缩进连续的行

在连续换行时，第二行要比上一行多缩进 8 个空格。从第三行开始，可以视情况在上一行缩进基础上增加更多空格。一个原则是如果同一条语句的两行采用同样的缩进，那么它们打头的语法元素必须在语法树上处于同一级。详见下面最后一个例子。

```
public class Example {
    @ExampleAnnotation(stringArrayValue = {
        "value1", "value2", "value3", "value4", "value5", "value6", "v
alue7", "value8"
    })
    public void exampleMethod() {
        // 在非赋值操作符前换行
        method1("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklm"
                + "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklm");
        // 该原则也适用于这些符号：点（.）
        method2("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklm")
                .method1("0123456789ABCDEFGHIJKLMNOPQRSTU");
        // 在赋值操作符（=）后换行
        String varWithVeryVeryVeryVeryVeryVeryVeryVeryVeryLongName =
                "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklm";
        // 该原则也适用于 foreach 语句中的:
        for (int anotherVarWithVeryVeryVeryVeryVeryVeryLongName :
                new int[] {0, 1, 2, 4, 5, 6, 7, 8, 9}) {
        }
        // 方法、构造函数名字和(之间不换行
        methodWithVeryVeryVeryVeryVeryVeryVeryVeryVeryVeryLongName(
                "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklm");
        // 逗号,紧跟前面的内容不换行
        methodWithTwoParameter("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ",
                "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ");
        // 至少用 8 个以上的空格缩进连续的行
        methodWithMultipleParameter("0123456789ABCDEFGHIJKLM",
                "如果同一条语句的两行采用同样的缩进，那么它们打头的语法元素必须
在语法树上处于同一级"
                    + "如果有多行需要换行可以按照需要调整为更多空格",
                "如果同一条语句的两行采用同样的缩进，那么它们打头的语法元素必须
在语法树上处于同一级");
    }
}
```

4.6 空白

4.6.1 空行

在以下情况下增加空行：

1. 在类的不同的成员间增加空行，包括：成员变量、构造函数、方法、内部类、静态初始化块、实例初始化块等。

   a) 两个成员变量声明之间可以不加空行。空行通常用于对成员变量进行逻辑分组。

2. 方法体内，按需增加空行，以便从逻辑上对语句进行分组禁止使用连续的空行。

4.6.2 空格

除了语法要求，字符串内的空格，以及 JavaDoc 里的空格，需要在下列情况里使用空格

1. 保留字（比如 if、for 和 catch）和随后的左圆括号（(）之间要有一个空格
2. 保留字（比如 else 和 catch）和之前的右花括号（}）之间要有一个空格
3. 在任意左花括号（{）之前要有一个空格。@SomeAnnotation({a, b})和 String[][] x = {{"foo"}}; 这两种情况除外。
4. 在二目和三目运算符两边各要有一个空格。该规则同样适用于<T extends Foo & Bar>中的&，catch (FooException | BarException e)中的|，以及 foreach 中的:
5. 在逗号（,）”、冒号（:）和类型转换用的右圆括号（)）后要有一个空格
6. 在用于行末注释的//前后各要有一个空格
7. 在声明语句的类型和名称之间要有一个空格，比如 List<String> list
8. 以下两种初始化数组的形式均可接受：

```
new int[] {5, 6}
new int[] { 5, 6 }
```

9. 除行首缩进、注释和字符串内的空格以外，禁止使用连续的空格

```
public class Example {
    public List<Element> getAllValidElements(String[] nodes) throws Pe
rmissionDeniedException {
        List<Element> result;
        for (int i = 0; i < nodes.length; ++i) {
            if (checkPermission(nodes[i])) {
                for (int id : getAllElementIdList(nodes[i])) {
                    Element e = (Element) getElement(id);
                    if (isValidElement(e)) {
                        result.add(e);
                    }
                }
            } else {
                throw new PermissionDeniedException("Can not access no
de " + nodes[i]);
            }
        }
        return result;
    }
}
```

4.7 表达式圆括号

由于不能保证所有人都清楚 Java 操作符优先级，因此推荐在表达式中增加圆括号用来明确其中运算的优先级。不做强制要求。

4.8 其他说明

4.8.1 枚举类

用逗号(,)分隔每个枚举常量，也可以用换行。

```
enum Color {
    RED,
    GREEN,
    YELLOW
}
enum Action {
    CREATE, READ, UPDATE, DELETE
}
```

4.8.2 变量声明

1. 一行只声明一个变量

2. 在需要时声明变量，声明后尽快初始化

4.8.3 数组

初始化换行原则：数组初始化通常不换行。在需要换行时，原则和普通代码块保持一致

例如以下示例均合法

```
new int[] {
    0, 1, 2, 3
}
new int[] {
    0, 1,
    2, 3
}
new int[] {
    0,
    1,
    2,
    3,
}
```

不要使用 C 的数组声明风格：必须用 String[] args 的方式来声明数组，而非 String args[]。

4.8.4 switch 语句

说明：一个 switch 块包含了一个或多个语句组。每组都以一个或多个标签开头（如 case 0:或者 default:），后面跟着一个或多个语句。

1. 缩进

    a) switch 块内的内容，使用 4 个空格缩进。

    b) 在每个标签后换行，换行后缩进 4 个空格，下一个标签恢复原缩进。

2. case 贯穿：禁止

    a) case 必须被终止（break、return 和抛出异常）。不允许使用 case 贯穿，这种风格容易导致 bug。如有实际需要的场景，应使用 if-else 来实现。

3. default

a) 每个 switch 中都要有 default。

b) default 也必须被终止（break、return 和抛出异常）。

c) 在 default 预期不可能被执行的时候，应执行抛出异常等可以有效发现 bug 的动作。

```
switch (dayOfWeek) {
    case MONDAY:
        ...
        break;
    case TUESDAY:
        ...
        break;
    ...
    default:
        throw new RuntimeException("Unhandled case " + dayOfWeek);
}
```

4.8.5 注解（Annotation）

添加在类、方法、构造函数、成员属性上的注解（Annotation）直接写在注释块之后，每个注解独占一行。

```
@Override
@Nullable
public String getNameIfPresent() {
    ...
}
```

4.8.6 注释

块注释的缩进与其上下文保持一致，可以使用/* ... */或者// ...的风格。多行的块注释中*必须对齐。

```
/*
 * 可以
 * 这样
 */
// 也可以
// 这样
/* 或者甚至
 * 这样 */
```

4.8.7 修饰符

当同时使用多个修饰符时，按照下列顺序：

public protected private abstract static final transient volatile synchronized native strictfp

4.8.8 数字

长整型数字必须使用大写字母 L 结尾，不能使用小写字母 l，以便和数字 1 进行区分。例如使用 3000000000L 而不是 3000000000l

5. 命名

5.1 通用命名规范

所有的标识符只允许使用 ASCII 字符和数字。合法的标识符命名必须能够匹配正则表达式：\w。

禁止使用一些特定的前缀和后缀，比如：name_、mName、m_name。Android 的成员变量命名允许例外，详见5.2.5变量命名。

不建议使用中文拼音来命名。例外：

1. 得到广泛认可的中文产品名，像贴吧、凤巢等

2. 以个人命名的算法或数据结构的实现

5.2 特定类型命名规范

5.2.1 包命名

包名只允许使用小写字母和数字，并且单词直接连接（不允许使用下划线）。比如：com.tplink.somebusiness，而 com.tplink.someBusiness 或 com.tplink.some_business 是不允许的。

5.2.2 类命名

类名必须遵循大写字母开头的驼峰式命名方式（UpperCamelCase）。

类名通常使用名词或名词短语，比如：Customer、ImmutableList。接口名也可以是名词或名词短语（比如：List），但很多时候会使用形容词代替（比如：Runnable）。

对于注解（Annotation）命名没有特定的规定。

单测类的命名规则必须以被测试类名开始，Test 单词结束。比如：BubbleSortTest。

5.2.3 方法命名

方法必须遵循小写字母开头的驼峰式命名方式（lowerCamelCase）。

方法名通常使用动词或动词短语，比如：sendMessage、stop。

在单测方法中可以含有下划线，通常用来指定特定场景：test<测试方法名>_<场景>，比如：testPop_emptyStack。

对于测试方法的命名没有强制性要求，也可以采用其他的命名方式。

5.2.4 常量命名

常量必须为大写单词，下划线分隔的命名方式。但什么是常量则没有特别精确的定义，需要主观判断。

首先常量一定是 static final 的字段（但是不是所有的 static final 字段都是常量）。一个字段是否是常量需要按照语义进行判断。举例：如果一个变量的状态是可以改变的，那么几乎可以肯定它不是一个常量。

仅仅看对象不可改变是不够的。

```
// 常量
static final int NUMBER = 5;
static final ImmutableList<String> NAMES = ImmutableList.of("Ed", "Ann
");
static final Joiner COMMA_JOINER = Joiner.on(',');
static final SomeMutableType[] EMPTY_ARRAY = {};
enum SomeEnum {
    ENUM_CONSTANT
}
// 非常量
static String nonFinal = "non-final";
final String nonStatic = "non-static";
static final Set<String> mutableCollection = new HashSet<String>();
static final ImmutableSet<SomeMutableType> mutableElements = Immutable
Set.of(mutable);
static final Logger logger = Logger.getLogger(MyClass.getName());
//常量通常使用名词或名词短语。
static final String[] nonEmptyArray = {"these", "can", "change"};
```

5.2.5 变量命名

非常量的变量（类变量和实例成员变量）名必须采用小写单词驼峰命名方式（lowerCamelCase）。

变量命名通常使用名词和名词短语。举例：computedValue、index。

对于 Android，允许一些命名例外：

非 public 非 static 的变量可以使用 m 开头

非常量的 static 变量可以使用 s 开头

5.2.6 参数命名

参数名必须采用小写单词驼峰命名方式（lowerCamelCase）。

严格限制使用单字符命名参数。某些约定俗成不会发生理解偏差的场景可以允许使用单字符参数，例如使用 x，y 表示坐标的时候。

5.2.7 局部变量命名

局部变量名通常采用小写单词驼峰命名方式（lowerCamelCase），可以自由缩写。

除循环变量和临时变量外，不允许使用单字符命名。

局部变量禁止使用常量命名方式，不管是否标识为 final。

5.2.8 泛型类型变量命名

泛型类型变量名必须遵循以下两种方式之一：

1. 单独一个大写字母，有时后面再跟一个数字。（例如，E、T2）。

2. 类命名的最后接一个大写字母。（例如，RequestT）。

5.3 驼峰命名方式定义

通常有多种方式将短语组织成驼峰方式，像一些缩写词：IPv6、iOS 等。为了统一，必须遵循以下几点规则。

1. 将字符全部转换为 ASCII 字符，并且去掉’等符号。例如，Müller's algorithm 被转换为 Muellers algorithm
2. 在空格和标点符号处对上一步的结果进行切分，组成一个词组。

   推荐：一些已经是驼峰命名的词语，也应该在这个时候被拆分。（例如 AdWords 被拆分为 ad words）。但是例如 iOS 之类的词语，它其实不是一个驼峰形式的词语，而是人们惯例使用的一个词语，因此不用做拆分。
3. 经过上面两步后，先将所有的字母转换为小写，再把每个词语的第一个字母转换为大写。
4. 最后，将所有词语连在一起，形成一个标识符。

注意：词语原来的大小写规则，应该被完全忽略。以下是一些例子：

原始短语	正确写法	非法写法
"XML HTTP request"	XmlHttpRequest	XMLHTTPRequest
"new customer ID"	newCustomerId	newCustomerID
"inner stopwatch"	innerStopwatch	innerStopWatch
"supports IPv6 on iOS?"	supportsIpv6OnIos	supportsIPv6OnIOS
"YouTube importer"	YouTubeImporter or YoutubeImporter[1]	

[1]号表示可以接受，但是不建议使用。

注意：有些词语在英文中，可以用[-]连接使用，也可以不使用[-]直接使用。例如“nonempty”和“non-empty”都可以。因此方法名字为 checkNonempty 或者 checkNonEmpty 都是合法的写法。

6. 编程实践

6.1 使用@Override

在允许的场景下推荐使用@Override 来标注方法。包括方法覆盖（Override）了父类的方法、方法实现了接口方法、还有接口方法覆盖（Override）了父接口的方法定义。

存在某些不允许的场景，例如 Java SE 5 不允许用@Override 标注方法对接口的实现。

例外情况：覆盖（Override）的目标方法标识为@Deprecated 的情况除外。

6.2 异常捕获：不推荐 Ignore

实际编码中仅仅在小部分场景下可以忽略异常不做任何处理。（通常是打印日志，或者重新包装抛出一个新异常）

如果真的不需要处理异常，那么需要在注释中解释。

举例：

```
try {
    int i = Integer.parseInt(response);
    return handleNumericResponse(i);
} catch (NumberFormatException ok) {
    // it's not numeric; that's fine, just continue
}
return handleTextResponse(response);
```

例外情况：在测试 Case 中，如果这个 Case 是验证预想中的异常分支，那么可以省略注释解释。

```
try {
    emptyStack.pop();
    fail();
} catch (NoSuchElementException expected) {
}
```

补充说明：针对上面这个场景，更推荐下面这种写法

```
@Test(expected = NoSuchElementException.class)
public void testPopEmptyStack() throws Exception {
    emptyStack.pop();
}
```

6.3 静态成员：

当访问一个静态成员的时候，正确的用法是使用类名引用，而不是使用对象或者对象表达式来引用。

```
Foo aFoo = ...;
Foo.aStaticMethod(); // good
aFoo.aStaticMethod(); // bad
somethingThatYieldsAFoo().aStaticMethod(); // very bad
```

6.4 Finalizer：禁用

禁止覆盖（Override）Object.finalize 方法。

7. Javadoc

7.1 格式

7.1.1 通用

基础的 Javadoc 格式如下例：

```
/**
 * Multiple lines of Javadoc text are written here,
 * wrapped normally...
 */
```

```
public int method(String p1) {
    ...
}
```

7.1.2 段落

除了第一段之外，后续段落描述应以<p>标签开始。段落首个文字可以紧接着<p>标签，也可以换行重新开始。

```
/**
 * Represents ....
 *
 * <p>More details here
 */
public class XXX {
    ...
}
/**
 * Represents ....
 *
 * <p>
 * New line is allowed.
 */
public class XXX {
    ...
}
```

7.1.3 @条目相关

在使用标准的@条目时按照@param，@return，@throws，@deprecated 的顺序进行排列。这四个条目在使用时描述不能为空。其余条目顺序可以随意。 每个@条目必须占据独立的行。当一行不够需要换行时，使用 8 个以上的空格缩进。

类的 Javadoc 注释中必须加@auth 和@date 标明作者和创建时间。

7.2 摘要

每个类和成员的 Javadoc 需要以摘要片段来开头。这非常重要，因为 Javadoc 会自动抽取第一句话显示在索引和方法概述表中。写摘要片段的原则是简洁清晰。它不是一句完整的话，例如 This is a ... ， 而 是 更 接 近 短 语 ， 如 A mutable sequence of characters. 或 Thrown when an exceptional arithmetic condition hasoccurred.。但在书写时采用类似句子的写法，以大写字母开头，并用.结尾。

7.3 Javadoc 应该应用在哪些场合

至少 Javadoc 应该应用于所有的 public 类、public 和 protected 的成员变量和方法，除了后面条款列举的例外情况。

除此之外的类和成员有时候也需要 Javadoc。例如在需要总体解释一个类、方法、成员的具体实现逻辑的候，通常会使用 Javadoc 替代注释。

7.3.1 例外情况：方法本身已经足够说明用途

对于那些非常显而易见的方法可以不写 Javadoc，比如 getFoo，写一个 Returns the foo 的意义不大。

注意：有时候不应该应用此例外来省略一些用户需要知道的信息。例如：getCannicalName。当大部分代码阅读者不知道 canonical name 是什么意思时，不应该省略 Javadoc（或者只写 Returns the canonical name 也是不行的，需要详细阐述什么是 canonical name）。

7.3.2 例外情况：覆盖（Override）方法

大多数情况下覆盖（Override）方法不需要 Javadoc。