Java 开发手册

前言

《Java 开发手册》是 Java 社区爱好者的集体智慧结晶和经验总结，经历了多次大规模一线实战的检验及不断完善，整理成册后，众多社区开发者踊跃参与打磨完善，系统化地整理成册，当前的最新版本是黄山版。现代软件行业的高速发展对开发者的综合素质要求越来越高，因为不仅是编程知识点，其它维度的知识点也会影响到软件的最终交付质量。比如：五花八门的错误码会人为地增加排查问题的难度；数据库的表结构和索引设计缺陷带来的系统架构缺陷或性能风险；工程结构混乱导致后续项目维护艰难；没有鉴权的漏洞代码容易被黑客攻击等。所以本手册以 Java 开发者为中心视角，划分为编程规约、异常日志、单元测试、安全规约、MySQL 数据库、工程结构、设计规约七个维度，再根据内容特征，细分成若干二级子目录。此外，依据约束力强弱及故障敏感性，规约依次分为【强制】、【推荐】、【参考】三大类。在延伸的信息中，“说明”对规约做了适当扩展和解释；“正例”提倡什么样的编码和实现方式；“反例”说明需要提防的雷区，以及真实的错误案例。

手册的愿景是码出高效，码出质量。现代软件架构的复杂性需要协同开发完成，如何高效地协同呢？无规矩不成方圆，无规范难以协同，比如，制定交通法规表面上是要限制行车权，实际上是保障公众的人身安全，试想如果没有限速，没有红绿灯，谁还敢上路行驶？对软件来说，适当的规范和标准绝不是消灭代码内容的创造性、优雅性，而是限制过度个性化，以一种普遍认可的统一方式一起做事，提升协作效率，降低沟通成本。代码的字里行间流淌的是软件系统的血液，代码质量的提升是尽可能少踩坑，杜绝踩重复的坑，切实提升系统稳定性，码出质量。

2017 年杭州云栖大会上发布了配套的 Java 开发规约 IDE 插件，下载量已达到 275 万人次，阿里云效也集成了代码规约扫描引擎。2018 年 9 月在云栖厅发布 36 万字的配套详解图书《码出高效》，秉持“图胜于表，表胜于言”的理念，深入浅出地将计算机基础、面向对象思想、数据结构与集合、JVM 探源与内存分析、并发与多线程、单元测试等知识丰富立体地呈现出来。本书紧扣学以致用、学以精进的目标，结合一线开发的实践经验和故障案例，与底层源码解析融会贯通，娓娓道来。《码出高效》和《Java 开发手册（第 2 版）》稿费所得收入均捐赠公益事情，希望用技术情怀帮助到更多的人。

目录

版本号	制定团队	更新日期	备注
1.7.1	全球 Java 社区开发者	2022.02.03	黄山版，新增 11 条新规约。

一、编程规约

(一) 命名风格

1. 【强制】所有编程相关的命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。
   反例：_name / __name / $Object / name_ / name$ / Object$
2. 【强制】所有编程相关的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。
   说明：正确的英文拼写和语法可以让阅读者易于理解，避免歧义。注意，即使纯拼音命名方式也要避免采用。
   正例：ali / alibaba / taobao / kaikeba / aliyun / youku / hangzhou 等国际通用的名称，可视同英文。
   反例：DaZhePromotion【打折】/ getPingfenByName()【评分】 / String fw【福娃】/ int 变量名 = 3
3. 【强制】代码和注释中都要避免使用任何人类语言中的种族歧视性或侮辱性词语。
   正例：blockList / allowList / secondary
   反例：blackList / whiteList / slave / SB / WTF
4. 【强制】类名使用 UpperCamelCase 风格，以下情形例外：DO / PO / DTO / BO / VO / UID 等。
   正例：ForceCode / UserDO / HtmlDTO / XmlService / TcpUdpDeal / TaPromotion
   反例：forcecode / UserDo / HTMLDto / XMLService / TCPUDPDeal / TAPromotion
5. 【强制】方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase 风格。
   正例：localValue / getHttpMessage() / inputUserId
6. 【强制】常量命名应该全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。
   正例：MAX_STOCK_COUNT / CACHE_EXPIRED_TIME
   反例：MAX_COUNT / EXPIRED_TIME
7. 【强制】抽象类命名使用 Abstract 或 Base 开头；异常类命名使用 Exception 结尾，测试类命名以它要测试的类的名称开始，以 Test 结尾。
8. 【强制】类型与中括号紧挨相连来定义数组。
   正例：定义整形数组 int[] arrayDemo。
   反例：在 main 参数中，使用 String args[] 来定义。
9. 【强制】POJO 类中的任何布尔类型的变量，都不要加 is 前缀，否则部分框架解析会引起序列化错误。
   说明：本文 MySQL 规约中的建表约定第 1 条，表达是与否的变量采用 is_xxx 的命名方式，所以需要在<resultMap>设置从 is_xxx 到 xxx 的映射关系。
   反例：定义为布尔类型 Boolean isDeleted 的字段，它的 getter 方法也是 isDeleted()，部分框架在反向解析时，“误以为”对应的字段名称是 deleted，导致字段获取不到，得到意料之外的结果或抛出异常。
10. 【强制】包名统一使用小写，点分隔符之间有且仅有一个自然语义的英语单词。包名统一使用单数形式，但是类名如果有复数含义，类名可以使用复数形式。
    正例：应用工具类包名为 com.alibaba.ei.kunlun.aap.util；类名为 MessageUtils（此规则参考 spring 的框架结构）。
11. 【强制】避免在子父类的成员变量之间、或者不同代码块的局部变量之间采用完全相同的命名，使可理解性降低。

说明：子类、父类成员变量名相同，即使是 public 也是能够通过编译，而局部变量在同一方法内的不同代码块中同名也是合法的，但是要避免使用。对于非 setter / getter 的参数名称也要避免与成员变量名称相同。

反例：

```
public class ConfusingName {
    protected int stock;
    protected String alibaba;
    // 非 setter/getter 的参数名称，不允许与本类成员变量同名
    public void access(String alibaba) {
        if (condition) {
            final int money = 666;
            // ...
        }
        for (int i = 0; i < 10; i++) {
            // 在同一方法体中，不允许与其它代码块中的 money 命名相同
            final int money = 15978;
            // ...
        }
    }
}
class Son extends ConfusingName {
    // 不允许与父类的成员变量名称相同
    private int stock;
}
```

12. 【强制】杜绝完全不规范的英文缩写，避免望文不知义。

反例：AbstractClass“缩写”成 AbsClass；condition“缩写”成 condi；Function“缩写”成 Fu，此类随意缩写严重降低了代码的可阅读性。

13. 【推荐】为了达到代码自解释的目标，任何自定义编程元素在命名时，使用完整的单词组合来表达。

正例：在 JDK 中，对某个对象引用的 volatile 字段进行原子更新的类名为 AtomicReferenceFieldUpdater。

反例：常见的方法内变量为 int a; 的定义方式。

14. 【推荐】在常量与变量命名时，表示类型的名词放在词尾，以提升辨识度。

正例：startTime / workQueue / nameList / TERMINATED_THREAD_COUNT

反例：startedAt / QueueOfWork / listName / COUNT_TERMINATED_THREAD

15. 【推荐】如果模块、接口、类、方法使用了设计模式，在命名时要体现出具体模式。

说明：将设计模式体现在名字中，有利于阅读者快速理解架构设计思想。

正例：public class OrderFactory;
public class LoginProxy;
public class ResourceObserver;

16. 【推荐】接口类中的方法和属性不要加任何修饰符号（public 也不要加），保持代码的简洁性，并加上有效的 Javadoc 注释。尽量不要在接口里定义常量，如果一定要定义，最好确定该常量与接口的方法相关，并且是整个应用的基础常量。

正例：接口方法签名 void commit();
接口基础常量 String COMPANY = "alibaba";

反例：接口方法定义 public abstract void commit();

说明：JDK8 中接口允许有默认实现，那么这个 default 方法，是对所有实现类都有价值的默认实现。

17.接口和实现类的命名有两套规则：

1）【强制】对于 Service 和 DAO 类，基于 SOA 的理念，暴露出来的服务一定是接口，内部的实现类用 Impl 的后缀与接口区别。

正例：CacheServiceImpl 实现 CacheService 接口。

2）【推荐】如果是形容能力的接口名称，取对应的形容词为接口名（通常是 –able 结尾的形容词）。

正例：AbstractTranslator 实现 Translatable。

18.【参考】枚举类名带上 Enum 后缀，枚举成员名称需要全大写，单词间用下划线隔开。

说明：枚举其实就是特殊的常量类，且构造方法被默认强制是私有。

正例：枚举名字为 ProcessStatusEnum 的成员名称：SUCCESS / UNKNOWN_REASON

19.【参考】各层命名规约：

A）Service / DAO 层方法命名规约：

1）获取单个对象的方法用 get 做前缀。

2）获取多个对象的方法用 list 做前缀，复数结尾，如：listObjects

3）获取统计值的方法用 count 做前缀。

4）插入的方法用 save / insert 做前缀。

5）删除的方法用 remove / delete 做前缀。

6）修改的方法用 update 做前缀。

B）领域模型命名规约：

1）数据对象：xxxDO，xxx 即为数据表名。

2）数据传输对象：xxxDTO，xxx 为业务领域相关的名称。

3）展示对象：xxxVO，xxx 一般为网页名称。

4）POJO 是 DO / DTO / BO / VO 的统称，禁止命名成 xxxPOJO。

(二) 常量定义

1.【强制】不允许任何魔法值（即未经预先定义的常量）直接出现在代码中。

反例：

```
// 开发者 A 定义了缓存的 key。
String key = "Id#taobao_" + tradeId;
cache.put(key, value);
// 开发者 B 使用缓存时直接复制少了下划线，即 key 是"Id#taobao" + tradeId，导致出现故障。
String key = "Id#taobao" + tradeId;
cache.get(key);
```

2.【强制】long 或 Long 赋值时，数值后使用大写 L，不能是小写 l，小写容易跟数字混淆，造成误解。

说明：public static final Long NUM = 2l; 写的是数字的 21，还是 Long 型的 2?

3.【强制】浮点数类型的数值后缀统一为大写的 D 或 F。

正例：public static final double HEIGHT = 175.5D;

public static final float WEIGHT = 150.3F;

4.【推荐】不要使用一个常量类维护所有常量，要按常量功能进行归类，分开维护。

说明：大而全的常量类，杂乱无章，使用查找功能才能定位到要修改的常量，不利于理解，也不利于维护。

正例：缓存相关常量放在类 CacheConsts 下；系统配置相关常量放在类 SystemConfigConsts 下。

5.【推荐】常量的复用层次有五层：跨应用共享常量、应用内共享常量、子工程内共享常量、包内共享常量、类内共享常量。

1）跨应用共享常量：放置在二方库中，通常是 client.jar 中的 constant 目录下。

2）应用内共享常量：放置在一方库中，通常是子模块中的 constant 目录下。

反例：易懂常量也要统一定义成应用内共享常量，两个程序员在两个类中分别定义了表示“是”的常量：
类 A 中：public static final String YES = "yes";
类 B 中：public static final String YES = "y";
A.YES.equals(B.YES)，预期是 true，但实际返回为 false，导致线上问题。

3）子工程内部共享常量：即在当前子工程的 constant 目录下。
4）包内共享常量：即在当前包下单独的 constant 目录下。
5）类内共享常量：直接在类内部 private static final 定义。

6. 【推荐】如果变量值仅在一个固定范围内变化用 enum 类型来定义。
说明：如果存在名称之外的延伸属性应使用 enum 类型，下面正例中的数字就是延伸信息，表示一年中的第几个季节。
正例：

```
public enum SeasonEnum {
    SPRING(1), SUMMER(2), AUTUMN(3), WINTER(4);
    private int seq;
    SeasonEnum(int seq) {
        this.seq = seq;
    }
    public int getSeq() {
        return seq;
    }
}
```

(三) 代码格式

1. 【强制】如果大括号内为空，简洁地写成{}即可，大括号中间无需换行和空格；如果是非空代码块，则：
1）左大括号前不换行。
2）左大括号后换行。
3）右大括号前换行。
4）右大括号后还有 else 等代码则不换行；表示终止的右大括号后必须换行。

2. 【强制】左小括号和右边相邻字符之间不需要空格；右小括号和左边相邻字符之间也不需要空格；而左大括号前需要加空格。详见第 5 条下方正例提示。
反例：if(空格 a == b 空格)

3. 【强制】if / for / while / switch / do 等保留字与左右括号之间都必须加空格。

4. 【强制】任何二目、三目运算符的左右两边都需要加一个空格。
说明：包括赋值运算符 =、逻辑运算符 &&、加减乘除符号等。

5. 【强制】采用 4 个空格缩进，禁止使用 Tab 字符。
说明：如使用 Tab 缩进，必须设置 1 个 Tab 为 4 个空格。IDEA 设置 Tab 为 4 个空格时，请勿勾选 Use tab character；而在 Eclipse 中，找到 tab policy 设置为 Spaces only，Tab size：4，最后必须勾选 insert spaces for tabs
正例：（涉及上述中的 1-5 点）

```
public static void main(String[] args) {
    // 缩进 4 个空格
    String say = "hello";
    // 运算符的左右必须有一个空格
    int flag = 0;
    // 关键词 if 与括号之间必须有一个空格，括号内的 f 与左括号，0 与右括号不需要空格
    if (flag == 0) {
        System.out.println(say);
```

```
        }
        // 左大括号前加空格且不换行；左大括号后换行
        if (flag == 1) {
            System.out.println("world");
            // 右大括号前换行，右大括号后有 else，不用换行
        } else {
            System.out.println("ok");
            // 在右大括号后直接结束，则必须换行
        }
    }
```

6. 【强制】注释的双斜线与注释内容之间有且仅有一个空格。

正例：

```
// 这是示例注释，请注意在双斜线之后有一个空格
String commentString = new String("demo");
```

7. 【强制】在进行类型强制转换时，右括号与强制转换值之间不需要任何空格隔开。

正例：

```
double first = 3.2D;
int second = (int)first + 2;
```

8. 【强制】单行字符数限制不超过 120 个，超出需要换行，换行时遵循如下原则：

1）第二行相对第一行缩进 4 个空格，从第三行开始，不再继续缩进，参考示例。

2）运算符与下文一起换行。

3）方法调用的点符号与下文一起换行。

4）方法调用中的多个参数需要换行时，在逗号后进行。

5）在括号前不要换行，见反例。

正例：

```
StringBuilder builder = new StringBuilder();
// 超过 120 个字符的情况下，换行缩进 4 个空格，并且方法前的点号一起换行
builder.append("yang").append("hao")...
    .append("chen")...
    .append("chen")...
    .append("chen");
```

反例：

```
StringBuilder builder = new StringBuilder();
// 超过 120 个字符的情况下，不要在括号前换行
builder.append("you").append("are")...append
    ("lucky");
// 参数很多的方法调用可能超过 120 个字符，逗号后才是换行处
method(args1, args2, args3, ...
    , argsX);
```

9. 【强制】方法参数在定义和传入时，多个参数逗号后面必须加空格。

正例：下例中实参的 args1 逗号后边必须要有一个空格。

```
method(args1, args2, args3);
```

10. 【强制】IDE 的 text file encoding 设置为 UTF-8；IDE 中文件的换行符使用 Unix 格式，不要使用 Windows 格式。

11. 【推荐】单个方法的总行数不超过 80 行。

说明：除注释之外的方法签名、左右大括号、方法内代码、空行、回车及任何不可见字符的总行数不超过 80 行。

正例：代码逻辑分清红花和绿叶，个性和共性，绿叶逻辑单独出来成为额外方法，使主干代码更加晰；共性逻辑抽取成为共性方法，便于复用和维护。

12.【推荐】没有必要增加若干空格来使变量的赋值等号与上一行对应位置的等号对齐。

正例：

```
int one = 1;
long two = 2L;
float three = 3F;
StringBuilder builder = new StringBuilder();
```

说明：增加 builder 这个变量，如果需要对齐，则给 one、two、three 都要增加几个空格，在变量比较多的情况下，是非常累赘的事情。

13.【推荐】不同逻辑、不同语义、不同业务的代码之间插入一个空行，分隔开来以提升可读性。

说明：任何情形，没有必要插入多个空行进行隔开。

(四) OOP 规约

1.【强制】避免通过一个类的对象引用访问此类的静态变量或静态方法，无谓增加编译器解析成本，直接用类名来访问即可。

2.【强制】所有的覆写方法，必须加 @Override 注解。

说明：getObject() 与 get0bject() 的问题。一个是字母的 O，一个是数字的 0，加 @Override 可以准确判断是否覆盖成功。另外，如果在抽象类中对方法签名进行修改，其实现类会马上编译报错。

3.【强制】相同参数类型，相同业务含义，才可以使用的可变参数，参数类型避免定义为 Object。

说明：可变参数必须放置在参数列表的最后。（建议开发者尽量不用可变参数编程）

正例：public List<User> listUsers(String type, Long... ids) {...}

4.【强制】外部正在调用的接口或者二方库依赖的接口，不允许修改方法签名，避免对接口调用方产生影响。接口过时必须加 @Deprecated 注解，并清晰地说明采用的新接口或者新服务是什么。

5.【强制】不能使用过时的类或方法。

说明：java.net.URLDecoder 中的方法 decode(String encodeStr) 这个方法已经过时，应该使用双参数 decode(String source, String encode)。接口提供方既然明确是过时接口，那么有义务同时提供新的接口；作为调用方来说，有义务去考证过时方法的新实现是什么。

6.【强制】Object 的 equals 方法容易抛空指针异常，应使用常量或确定有值的对象来调用 equals。

正例："test".equals(param);

反例：param.equals("test");

说明：推荐使用 JDK7 引入的工具类 java.util.Objects#equals(Object a, Object b)

7.【强制】所有整型包装类对象之间值的比较，全部使用 equals 方法比较。

说明：对于 Integer var = ? 在 -128 至 127 之间的赋值，Integer 对象是在 IntegerCache.cache 产生，会复用已有对象，这个区间内的 Integer 值可以直接使用 == 进行判断，但是这个区间之外的所有数据，都会在堆上产生，并不会复用已有对象，这是一个大坑，推荐使用 equals 方法进行判断。

8.【强制】任何货币金额，均以最小货币单位且为整型类型进行存储。

9.【强制】浮点数之间的等值判断，基本数据类型不能使用 == 进行比较，包装数据类型不能使用 equals 进行判断。

说明：浮点数采用“尾数+阶码”的编码方式，类似于科学计数法的“有效数字+指数”的表示方式。二进制无法精确表示大部分的十进制小数，具体原理参考《码出高效》。

反例：

```
float a = 1.0F - 0.9F;
float b = 0.9F - 0.8F;
if (a == b) {
    // 预期进入此代码块，执行其它业务逻辑
    // 但事实上 a == b 的结果为 false
}
Float x = Float.valueOf(a);
Float y = Float.valueOf(b);
if (x.equals(y)) {
    // 预期进入此代码块，执行其它业务逻辑
    // 但事实上 equals 的结果为 false
}
```

正例：

(1)指定一个误差范围，两个浮点数的差值在此范围之内，则认为是相等的。

```
float a = 1.0F - 0.9F;
float b = 0.9F - 0.8F;
float diff = 1e-6F;
if (Math.abs(a - b) < diff) {
    System.out.println("true");
}
```

(2)使用 BigDecimal 来定义值，再进行浮点数的运算操作。

```
BigDecimal a = new BigDecimal("1.0");
BigDecimal b = new BigDecimal("0.9");
BigDecimal c = new BigDecimal("0.8");
BigDecimal x = a.subtract(b);
BigDecimal y = b.subtract(c);

if (x.compareTo(y) == 0) {
    System.out.println("true");
}
```

10. 【强制】BigDecimal 的等值比较应使用 compareTo() 方法，而不是 equals() 方法。

说明：equals() 方法会比较值和精度（1.0 与 1.00 返回结果为 false），而 compareTo() 则会忽略精度。

11. 【强制】定义数据对象 DO 类时，属性类型要与数据库字段类型相匹配。

正例：数据库字段的 bigint 必须与类属性的 Long 类型相对应。

反例：某业务的数据库表 id 字段定义类型为 bigint unsigned，实际类对象属性为 Integer，随着 id 越来越大，超过 Integer 的表示范围而溢出成为负数，此时数据库 id 不支持存入负数抛出异常产生线上故障。

12. 【强制】禁止使用构造方法 BigDecimal(double) 的方式把 double 值转化为 BigDecimal 对象。

说明：BigDecimal(double) 存在精度损失风险，在精确计算或值比较的场景中可能会导致业务逻辑异常。如：BigDecimal g = new BigDecimal(0.1F)；实际的存储值为：0.100000001490116119384765625

正例：优先推荐入参为 String 的构造方法，或使用 BigDecimal 的 valueOf 方法，此方法内部其实执行了 Double 的 toString，而 Double 的 toString 按 double 的实际能表达的精度对尾数进行了截断。

```
BigDecimal recommend1 = new BigDecimal("0.1");
BigDecimal recommend2 = BigDecimal.valueOf(0.1);
```

13.关于基本数据类型与包装数据类型的使用标准如下：

1）【强制】所有的 POJO 类属性必须使用包装数据类型。

2）【强制】RPC 方法的返回值和参数必须使用包装数据类型。

3）【推荐】所有的局部变量使用基本数据类型。