西北師範大學

背包问题知识社区系统

编码规范

版本：1.0

团队名称：待宰的高羊

团队成员：何晨泽

高杨

阿丽米拉

谢家俊

指导教师：代祖华

完成时间：2022年6月19日

修改记录

* A-新增 D-删除 M-修改

修改号	日期	影响的范围	A/D/M *	概要描述
001	2022-6-19	全文	A	编撰版本文档 1.0

目录

一、命名风格

1．代码中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．代码中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：alibaba / taobao / youku / hangzhou 等国际通用的名称，可视同英文。

反例：DaZhePromotion [打折] / getPingfenByName() [评分] / int 某变量 = 3

3．类名使用 UpperCamelCase 风格，必须遵从驼峰形式，但以下情形例外：DO / BO / DTO / VO / AO。

正例：MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion

反例：macroPolo / UserDo / XMLService / TCPUDPDeal / TAPromotion

4．方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase 风格，必须遵从驼峰形式。

正例： localValue / getHttpMessage() / inputUserId

5．常量命名全部大写，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

反例：MAX_COUNT

6．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。

7．中括号是数组类型的一部分，数组定义如下：String[] args;

反例：使用 String args[]的方式来定义。

8．POJO 类中布尔类型的变量，都不要加 is，否则部分框架解析会引起序列化错误。

反例：定义为基本数据类型 Boolean isDeleted；的属性，它的方法也是 isDeleted()，RPC 框架在反向解析的时候，“以为”对应的属性名称是 deleted，导致属性获取不到，进而抛出异常。

9．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.open.util、类名为 MessageUtils（此规则参考 spring 的框架结构）。

二、常量定义

1．不允许任何魔法值（即未经定义的常量）直接出现在代码中。

反例：`String key = "Id#taobao_" + tradeId; cache.put(key, value);`

2．long 或者 Long 初始赋值时，使用大写的 L，不能是小写的 l，小写容易跟数字 1 混淆，造成误解。

说明：`Long a = 2l;` 写的是数字的 21，还是 Long 型的 2

3．不要使用一个常量类维护所有常量，按常量功能进行归类，分开维护。

说明：大而全的常量类，非得使用查找功能才能定位到修改的常量，不利于理解和维护。

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

4．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 `modules` 中的 `constant` 目录下。 反例：易懂变量也要统一定义成应用内共享常量，两位攻城师在两个类中分别定义了表示“是”的变量：

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。

5．如果变量值仅在一个范围内变化，且带有名称之外的延伸属性，定义为枚举类。下面 正例中的数字就是延伸信息，表示星期几。

正例：public Enum { MONDAY(1), TUESDAY(2), WEDNESDAY(3), THURSDAY(4), FRIDAY(5), SATURDAY(6), SUNDAY(7);}

三、代码格式

1．大括号的使用约定。如果是大括号内为空，则简洁地写成{}即可，不需要换行；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．左小括号和字符之间不出现空格；同样，右小括号和字符之间也不出现空格。

反例：if (空格 a == b 空格)

3．if/for/while/switch/do 等保留字与括号之间都必须加空格。

4．任何二目、三目运算符的左右两边都需要加一个空格。

说明：运算符包括赋值运算符=、逻辑运算符&&、加减乘除符号等。

5．采用 4 个空格缩进，禁止使用 tab 字符。

说明：如果使用 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，必须勾选 insert spaces for tabs。

6．注释的双斜线与注释内容之间有且仅有一个空格。

正例：// 注释内容，注意在//和注释内容之间有一个空格。

7．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） 方法调用时，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。

8．方法参数在定义和传入时，多个参数逗号后边必须加空格。

正例：下例中实参的"a",后边必须要有一个空格。 method("a", "b", "c");

四、OOP 规约

1．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．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．相同参数类型，相同业务含义，才可以使用 Java 的可变参数，避免使用 Object。

说明：可变参数必须放置在参数列表的最后。（提倡同学们尽量不用可变参数编程）正例:public User getUsers(String type, Integer... ids) {...}

4．外部正在调用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．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. 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(object); 反例：object.equals("test");
说明：推荐使用 java.util.Objects#equals（JDK7 引入的工具类）

7．所有的相同类型的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 范围内的赋值，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．关于基本数据类型与包装数据类型的使用标准如下：

1）所有的 POJO 类属性必须使用包装数据类型。

2）RPC 方法的返回值和参数必须使用包装数据类型。

3）所有的局部变量使用基本数据类型。

说明：POJO 类属性没有初值是提醒使用者在需要使用时，必须自己显式地进行赋值，任何 NPE 问题，或者入库检查，都由使用者来保证。

正例：数据库的查询结果可能是 null，因为自动拆箱，用基本数据类型接收有 NPE 风险。

反例：比如显示成交总额涨跌情况，即正负 x%，x 为基本数据类型，调用的 RPC 服务，调用 不成功时，返回的是默认值，页面显示为 0%，这是不合理的，应该显示成中划线。所以包装 数据类型的 null 值，能够表示额外的信息，如：远程调用失败，异常退出。

9．定义 DO/DTO/VO 等 POJO 类时，不要设定任何属性默认值。

反例：POJO 类的 gmtCreate 默认值为 new Date();但是这个属性在数据提取时并没有置入具 体值，在更新其它字段时又附带更新了此字段，导致创建时间被修改成当前时间。

10．序列化类新增属性时，请不要修改 serialVersionUID 字段，避免反序列失败；如 果完全不兼容升级，避免反序列化混乱，那么请修改 serialVersionUID 值。

说明：注意 serialVersionUID 不一致会抛出序列化运行时异常。

11. 构造方法里面禁止加入任何业务逻辑，如果有初始化逻辑，请放在 init 方法中。

12. POJO 类必须写 toString 方法。使用 IDE 的中工具：source> generate toString 时，如果继承了另一个 POJO 类，注意在前面加一下 super.toString。

说明：在方法执行抛出异常时，可以直接调用 POJO 的 toString()方法打印其属性值，便于排 查问题。

13. 使用索引访问用 String 的 split 方法得到的数组时，需做最后一个分隔符后有无 内容的检查，否则会有抛 IndexOutOfBoundsException 的风险。

五、集合处理

1．关于 hashCode 和 equals 的处理，遵循如下规则：

1） 只要重写 equals，就必须重写 hashCode。

2） 因为 Set 存储的是不重复的对象，依据 hashCode 和 equals 进行判断，所以 Set 存储的 对象必须重写这两个方法。

3）如果自定义对象做为 Map 的键,那么必须重写 hashCode 和 equals。

说明：String 重写了 hashCode 和 equals 方法，所以我们可以非常愉快地使用 String 对象 作为 key 来使用。

2． ArrayList 的 subList 结果不可强转成 ArrayList，否则会抛出 ClassCastException 异常，即 java.util.RandomAccessSubList cannot be cast to java.util.ArrayList.

说明：subList 返回的是 ArrayList 的内部类 SubList，并不是 ArrayList ，而是 ArrayList 的一个视图，对于 SubList 子列表的所有操作最终会反映到原列表上。

3．在 subList 场景中，高度注意对原集合元素个数的修改，会导致子列表的遍历、增加、 删除均会产生 ConcurrentModificationException 异常。

4．使用集合转数组的方法，必须使用集合的 toArray(T[] array)，传入的是类型完全 一样的数组，大小就是 list.size()。

说明：使用 toArray 带参方法，入参分配的数组空间不够大时，toArray 方法内部将重新分配 内存空间，并返回新数组地址；如果数组元素大于实际所需，下标为[list.size()]的数组 元素将被置为 null，其它数组元素保持原值，因此最好将方法入参数组大小定义与集合元素 个数一致。

5． 使用工具类 Arrays.asList()把数组转换成集合时，不能使用其修改集合相关的方 法，它的 add/remove/clear 方法会抛出

UnsupportedOperationException 异常。

说明：asList 的返回对象是一个 Arrays 内部类，并没有实现集合的修改方法。Arrays.asList 体现的是适配器模式，只是转换接口，后台的数据仍是数组。

6． 泛型通配符来接收返回的数据，此写法的泛型集合不能使用 add 方法，而不能使用 get 方法，做为接口调用赋值时易出错。

说明：扩展说一下 PECS(Producer Extends Consumer Super)原则：第一、频繁往外读取内 容的，适合用。第二、经常往里插入的，适合用。

7．不要在 foreach 循环里进行元素的 remove/add 操作。remove 元素请使用 Iterator 方式，如果并发操作，需要对 Iterator 对象加锁。

8． 在 JDK7 版本及以上，Comparator 要满足如下三个条件，不然 Arrays.sort， Collections.sort 会报 IllegalArgumentException 异常。

9．集合初始化时，指定集合初始值大小。

说明：HashMap 使用 HashMap(int initialCapacity) 初始化， 正例：initialCapacity = (需要存储的元素个数 / 负载因子) + 1。注意负载因子（即 loader factor）默认为 0.75，如果暂时无法确定初始值大小，请设置为 16（即默认值）。 反例：HashMap 需要放置 1024 个元素，由于没有设置容量初始大小，随着元素不断增加，容 量 7 次被迫扩大，resize 需要重建 hash 表，严重影响性能。

10．使用 entrySet 遍历 Map 类集合 KV，而不是 keySet 方式进行遍历。

说明：keySet 其实是遍历了 2 次，一次是转为 Iterator 对象，另一次是从 hashMap 中取出 key 所对应的 value。而 entrySet 只是遍历了一次就把 key 和 value 都放到了 entry 中，效 率更高。如果是 JDK8，使用 Map.foreach 方法。

正例：values()返回的是 V 值集合，是一个 list 集合对象；keySet()返回的是 K 值集合，是 一个 Set 集合对象；entrySet()返回的是 K-V 值组合集合。

11．高度注意 Map 类集合 K/V 能不能存储 null 值的情况。

12．合理利用好集合的有序性(sort)和稳定性(order)，避免集合的无序性(unsort)和 不稳定性(unorder)带来的负面影响。

13．利用 Set 元素唯一的特性，可以快速对一个集合进行去重操作，避免使用 List 的 contains 方法进行遍历、对比、去重操作。

六、注释规约

1．类、类属性、类方法的注释必须使用 Javadoc 规范，使用/内容*/格式，不得使用 // xxx 方式。**

说明：在 IDE 编辑窗口中，Javadoc 方式会提示相关注释，生成 Javadoc 可以正确输出相应注 释；在 IDE 中，工程调用方法时，不进入方法即可悬浮提示方法、参数、返回值的意义，提高 阅读效率。

2．所有的抽象方法（包括接口中的方法）必须要用 Javadoc 注释、除了返回值、参数、 异常说明外，还必须指出该方法做什么事情，实现什么功能。

说明：对子类的实现要求，或者调用注意事项，请一并说明。

3．所有的类都必须添加创建者和创建日期。

4．方法内部单行注释，在被注释语句上方另起一行，使用//注释。方法内部多行注释 使用/* */注释，注意与代码对齐。

5．所有的枚举类型字段必须要有注释，说明每个数据项的用途。

6．与其“半吊子”英文来注释，不如用中文注释把问题说清楚。专有名词与关键字保持 英文原文即可。

反例：“TCP 连接超时”解释成“传输控制协议连接超时”，理解反而费脑筋。

7．代码修改的同时，注释也要进行相应的修改，尤其是参数、返回值、异常、核心逻辑等的修改。

说明：代码与注释更新不同步，就像路网与导航软件更新不同步一样，如果导航软件严重滞后， 就失去了导航的意义。

8． 谨慎注释掉代码。在上方详细说明，而不是简单地注释掉。如果无用，则删除。

说明：代码被注释掉有两种可能性：1）后续会恢复此段代码逻辑。2）永久

不用。前者如果没 有备注信息，难以知晓注释动机。后者建议直接删掉（代码仓库保存了历史代码）。

9．好的命名、代码结构是自解释的，注释力求精简准确、表达到位。避免出现注释的 一个极端：过多过滥的注释，代码的逻辑一旦修改，修改注释是相当大的负担。