重庆紫光华山智安科技有限公司技术规范

安防产品编程规范之 Java 代码

目 录

1 介绍 1-2

2 命名规范 2-1

2.1 概述 2-1

2.1.1 统一 2-1

2.1.2 达意 2-1

2.1.3 简洁 2-1

2.1.4 骆驼法则 2-1

2.1.5 英文 2-1

2.2 包名 2-1

2.3 类名和接口名 2-2

2.3.1 基本原则 2-2

2.3.2 常用类后缀 2-2

2.3.3 特殊类 2-3

2.4 方法名 2-3

2.4.1 基本原则 2-3

2.4.2 常用方法前缀说明 2-3

2.4.3 命名规约 2-4

2.5 域名 2-5

2.5.1 常量 2-5

2.5.2 变量 2-5

3 代码风格 3-1

3.1 源文件编码 3-1

3.2 缩进风格 3-1

3.2.1 大括号的使用 3-1

3.2.2 空格的使用 3-1

3.2.3 空行的使用 3-1

4 控制语句 4-1

4.1 基本原则 4-1

4.2 推荐使用 4-2

5 注释规范 5-3

5.1 基本原则 5-3

5.2 推荐使用 5-3

6 OOP 规范 ····· 6-5
6.1 基本原则 ····· 6-5

1 介绍

本 Java 代码规范以 SUN 的标准 Java 代码规范为基础，结合业界最佳实践以及我司实际需要做了一些调整。本文档中没有说明的地方，请参看 SUN Java 标准代码规范。

2 命名规范

2.1 概述

标识符的命名力求做到统一、达意和简洁。尽量做到每个人按照规范来，多人开发如一人开发一样。

2.1.1 统一

统一是指，对于同一概念，在程序中用同一种表示方法。比如对于供应商，既可以用 supplier，也可以用 provider，但是我们只能选定一个使用，至少在一个 Java 项目中保持统一。统一是作为重要的，如果对同一的概念有不同的表示方法，会使代码混乱难以理解。即使不能取得好的名称，但是只要统一，阅读起来也不会太困难，因为阅读者只要理解一次。

2.1.2 达意

达意是指，标识符能准确的表达出它所代表的意义，比如：newSupplier,OrderPaymentGatewayService 等；而 supplier1,service2,idtts 等则不是好的命名方式。准确有两层含义：一是正确，不要有歧义；二是丰富。如果给一个代表供应商的变量起名是 order，显然没有正确表达。同样的，supplier 远没有 targetSupplier 意义丰富。

2.1.3 简洁

简洁是指，在统一和达意的前提下，用尽量少的标识符。如果不能达意，宁愿不要简洁。比如：theOrderNameOfTheTargetSupplierWitchIsTransfered 太长，transferedTargetSupplierOrderName 则较好，但是 transTgtSplOrdNm 就不好了。省略元音的缩写方式不要使用，缩写的使用原则是只使用业界通用、约定俗成的缩写，比如 dao、impl。

2.1.4 骆驼法则

Java 中，除了包名，静态常量等特殊情况，大部分情况下标识符使用骆驼法则，即单词之间不使用特殊符号分割，而是通过首字母大写来分割。比如：supplierName,addNewContract，而不是 supplier_name,add_new_contract。

2.1.5 英文

尽量使用通俗易懂的英文单词，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。

2.2 包名

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。命名格式如下：“倒置的公司域后缀 部门名 小组名 模块名”，可以根据产品划分为多级模块。

正例：package com.h3c.bigdata.panda.dataengine.server

2.3 类名和接口名

2.3.1 基本原则

1）类名使用 UpperCamelCase 风格，但以下情形例外：DO / BO / DTO / VO / AO / PO / UID 等。

正例：MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion

反例：macroPolo / UserDo / XMLService / TCPUDPDeal / TAPromotion

2.3.2 常用类后缀

类名往往用不同的后缀表达额外的意思，如**错误!未找到引用源。**。

表2-1 类后缀说明

后缀名	意义	举例
Service	表明这个类是个服务类，里面包含了给其他类提同业务服务的方法	PaymentOrderService
Impl	这个类是一个实现类，而不是接口	PaymentOrderServiceImpl
Inter	这个类是一个接口	LifeCycleInter
Job	某种按时间运行的任务	PayMentOrderCancelJob
Util	Java工具类	SecurityUtil
Mapper	Mybatis生成的数据访问方法类	UserMapper
Dao	这个类封装了数据访问方法	PaymentOrderDao
Action	直接处理页面请求，管理页面逻辑类	UpdateOrderListAction
Listener	响应某种事件的类	PaymentSuccseeListener
Event	这个类代表了某种事件	PaymentSuccessEvent
Servlet	一个Servlet	PaymentCallbackServlet
Factory	生成某种对象工厂的类	PaymentOrderFactory
Adapter	用来连接某种以前不被支持的对象的类	DatabaseLogAdapter
Wrapper	这是一个包装类，为了给某个类提供没有的能力	SelectableOrListWrapper
EJB	表示这个类为EJB类	LCIssueInfoManagerEJB
Bean	这是一个POJO	MenuStateBean

2.3.3 特殊类

1）抽象类命名使用 Abstract 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。

2）枚举类名建议带上 Enum 后缀。

*说明：枚举其实就是特殊的类，域成员均为常量，*名称需要全大写，单词间用下划线隔开，*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。

3）如果模块、接口、类、方法使用了设计模式，在命名时需体现出具体模式。

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理念。

正例：public class OrderFactory;

```
public class LoginProxy;
public class ResourceObserver;
```

2.4 方法名

2.4.1 基本原则

1）方法命名一般采用动宾结构，动词在前，首字母小写，必须遵从驼峰形式。

正例：addOrder() ;

反例：AddOrder()/orderAdd() ;

2）查询方法要查询的内容在前，条件在后。

正例：getUserNameById()

2.4.2 常用方法前缀说明

动词前缀往往表达特定的含义，如下**错误!未找到引用源。**

表2-2 方法前缀说明

前缀名	意义	举例
create	创建	createOrder()
delete	删除	deleteOrder()
add	创建，暗示新创建的对象属于某个集合	addPaidOrder()
remove	删除	removeOrder()
init或则initialize	初始化，暗示会做些诸如获取资源等特殊动作	initializeObjectPool
destroy	销毁，暗示会做些诸如释放资源的特殊动作	destroyObjectPool
open	打开	openConnection()
close	关闭	closeConnection()
read	读取	readUserName()
write	写入	writeUserName()
get	获得	getName()
set	设置	setName()
prepare	准备	prepareOrderList()
copy	复制	copyCustomerList()
modity	修改	modifyActualTotalAmount()
calculate	数值计算	calculateCommission()
do	执行某个过程或流程	doOrderCancelJob()
dispatch	判断程序流程转向	dispatchUserRequest()
start	开始	startOrderProcessing()
stop	结束	stopOrderProcessing()
send	发送某个消息或事件	sendOrderPaidMessage()
receive	接受消息或时间	receiveOrderPaidMessage()
respond	响应用户动作	responseOrderListItemClicked()
find或select	查找对象	findNewSupplier()
update	更新对象	updateCommission()

2.4.3 命名规约

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，复数形式结尾如：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。

2.5 域名

2.5.1 常量

1）常量名使用全大写的英文描述，英文单词之间用下划线分割，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

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

3）在 long 或者 Long 赋值时，数值后使用大写的 L，不能是小写的 l，小写容易跟数字 1 混淆，造成误解。

4）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。

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。

2.5.2 变量

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/$name/name_/name$/name__

2） 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。

3 代码风格

3.1 源文件编码

源文件使用 utf-8 编码，结尾用 unix \n 分割。

3.2 缩进风格

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。

3.2.1 大括号的使用

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。

3.2.2 空格的使用

1）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；而左大括号前需要空格。

反例：if (空格 a == b 空格)

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

3）任何二目、三目运算符的左右两边都需要加一个空格。

4）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 ygb = new String();
```

5）尽量避免一行的长度超过 200 个字符。

6）方法参数在定义和传入时，多个参数逗号后边必须加空格。

3.2.3 空行的使用

空行可以表达代码在语义上的分割，注释的作用范围，等等。将类似操作，或一组操作放在一起不用空行隔开，而用空行隔开不同组的代码。

1）连续两行的空行代表更大的语义分割

2）方法之间用空行分割（尽量用一行空行）

3）域之间用空行分割

4 控制语句

4.1 基本原则

1）在一个 switch 块内，每个 case 要么通过 break/return 等来终止，要么注释说明程序将继续执行到哪一个 case 为止；在一个 switch 块内，都必须包含一个 default 语句并且放在最后，即使空代码

2）在 if/else/for/while/do 语句中必须使用大括号。即使只有一行代码，避免采用单行的编码方式：反例：if (condition) statements

3）在高并发场景中，避免使用”等于”判断作为中断或退出的条件。

说明：如果并发控制没有处理好，容易产生等值判断被“击穿”的情况，使用大于或小于的区间判断条件来代替。

反例：判断剩余奖品数量等于 0 时，终止发放奖品，但因为并发处理错误导致奖品数量瞬间变成了负数，这样的话，活动无法终止。

4）表达异常的分支时，少用 if-else 方式，这种方式可以改写成：

```
if (condition) {
    ...
    return obj;
}
// 接着写 else 的业务逻辑代码;
```

说明：如果非得使用 if()...else if()...else...方式表达逻辑，避免后续代码维护困难，请勿超过 3 层。

正例：超过 3 层的 if-else 的逻辑判断代码可以使用卫语句、策略模式、状态模式等来实现，其中卫语句示例如下：

```
public void today() {
    if (isBusy()) {
        System.out.println(“change time.”);
        return;
    }

    if (isFree()) {
        System.out.println(“go to travel.”);
        return;
    }

    System.out.println(“stay at home to learn Alibaba Java Coding Guidelines.”);
    return;
}
```

5）循环体中的语句要考量性能，以下操作尽量移至循环体外处理，如定义对象、变量、获取数据库连接，进行不必要的 try-catch 操作（这个 try-catch 是否可以移至循环体外）

6）针对 try-catch 的函数，规定必须写 finally 逻辑，可以将一些文件的关闭，数据库的断开连接等在 finally 中执行。

说明：如果 JDK7 及以上，可以使用 try-with-resources 方式。

4.2 推荐使用

1）除常用方法（如 getXxx/isXxx）等外，不要在条件判断中执行其它复杂的语句，将复杂逻辑判断的结果赋值给一个有意义的布尔变量名，以提高可读性。

说明：很多 if 语句内的逻辑相当复杂，阅读者需要分析条件表达式的最终结果，才能明确什么样的条件执行什么样的语句，那么，如果阅读者分析逻辑表达式错误呢？

正例：

```
// 伪代码如下
final boolean existed = (file.open(fileName, "w") != null) && (...) || (...);
if (existed) {
    ...
}
```

反例：

```
if ((file.open(fileName, "w") != null) && (...) || (...)) {
    ...
}
```

2）避免采用取反逻辑运算符。说明：取反逻辑不利于快速理解，并且取反逻辑写法必然存在对应的正向逻辑写法。

正例：使用 if (x < 628) 来表达x 小于628。

反例：使用if (!(x >= 628)) 来表达x 小于628。

3）接口入参保护，这种场景常见的是用作批量操作的接口。

5 注释规范

5.1 基本原则

1）类、类属性、类方法的注释必须使用 Javadoc 规范。

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。

类格式：

类属性格式

方法格式：

内容格式：使用 /**内容*/ 格式，不得使用 // xxx 方式

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。

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。

6）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。

5.2 推荐使用

1）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。

2）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）。

3）对于注释的要求：第一、能够准确反应设计思想和代码逻辑；第二、能够描述业务含义，使别的程序员能够迅速了解到代码背后的信息。完全没有注释的大段代码对于阅读者形同天书，注释是给自己看的，即使隔很长时间，也能清晰理解当时的思路；注释也是给继任者看的，使其能够快速接替自己的工作。

4）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。

反例：

```
// put elephant into fridge
put(elephant, fridge);
```

方法名 put，加上两个有意义的变量名 elephant 和 fridge，已经说明了这是在干什么，语义清晰的代码不需要额外的注释。

6 OOP 规范

6.1 基本原则

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 List<User> listUsers(String type, Long... 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：

所有的 POJO 类属性必须使用包装数据类型。

RPC 方法的返回值和参数必须使用包装数据类型。

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 中的工具：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） 禁止在 POJO 类中，同时存在对应属性 xxx 的 isXxx()和 getXxx()方法。

说明：框架在调用属性 xxx 的提取方法时，并不能确定哪个方法一定是被优先调用到。

14） 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。

说明：

```
String str = "a,b,c,,";
String[] ary = str.split(",");
// 预期大于 3，结果是 3
System.out.println(ary.length);
```

15） 当一个类有多个构造方法，或者多个同名方法，这些方法应该按顺序放置在一起，便于阅读，此条规则优先于第 16 条规则。

16） 类内方法定义的顺序依次是：公有方法或保护方法 > 私有方法 > getter/setter 方法。

说明：公有方法是类的调用者和维护者最关心的方法，首屏展示最好；保护方法虽然只是子类关心，也可能是“模板设计模式”下的核心方法；而私有方法外部一般不需要特别关心，是一个黑盒实现；因为承载的信息价值较低，所有 Service 和 DAO 的 getter/setter 方法放在类体最后。

17） setter 方法中，参数名称与类成员变量名称一致，this.成员名 = 参数名。在 getter/setter 方法中，不要增加业务逻辑，增加排查问题的难度。

反例：

```
public Integer getData() {
    if (condition) {
        return this.data + 100;
    } else {
        return this.data - 100;
    }
}
```

18） 循环体内，字符串的连接方式，使用 StringBuilder 的 append 方法进行扩展。说明：下例中，反编译出的字节码文件显示每次循环都会 new 出一个 StringBuilder 对象，然后进行 append 操作，最后通过 toString 方法返回 String 对象，造成内存资源浪费。

反例：

```
String str = "start";
for (int i = 0; i < 100; i++) {
    str = str + "hello";
}
```

19） final 可以声明类、成员变量、方法、以及本地变量，下列情况使用 final 关键字：

a) 不允许被继承的类，如：String 类。

b) 不允许修改引用的域对象。

c) 不允许被重写的方法，如：POJO 类的 setter 方法。

d) 不允许运行过程中重新赋值的局部变量。

e) 避免上下文重复使用一个变量，使用 final 描述可以强制重新定义一个变量，方便更好地进行重构。

20） 慎用 Object 的 clone 方法来拷贝对象。

说明：对象的 clone 方法默认是浅拷贝，若想实现深拷贝需要重写 clone 方法实现域对象的深度遍历式拷贝。