开发规范

1. 编程规约

(一)命名风格

1．【强制】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．【强制】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．【强制】类名使用upperCamelCase风格，但以下情形例外: Do / Bo / DTo / vo / A0 /

PO / UID等。

正例: MarcoPolo / userDo / xmlService / TcpUdpDeal / TaPromotion反例:macroPolo / UserDo / xMLService / TCPUDPDeal / TAPromotion

4．【强制】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() l inputUserId

5.【强制】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．【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。7．【强制】类型与中括号紧挨相连来表示数组。

7。【强制】类型与中括号紧挨相连来表示数组。

正例:定义整形数组int[] arrayDemo;

反例:在main参数中，使用String args[]来定义。

正例:定义整形数组int[] arrayDemo;

反例:在 main参数中，使用String args[]来定义。

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

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，“误以为”对应的属性名称是deleted，导致属性获取不到，进而抛出异常。

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

10.【强制】杜绝完全不规范的缩写，避免望文不知义。

反例:Abstractclass“缩写”命名成Absclass; condition“缩写”命名成condi，此类随意缩写严重降低了代码的可阅读性。

11.【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，使用尽量完整的单词

组合来表达其意。

正例:在JDK中，表达原子更新的类名为: AtomicReferenceFieldUpdater。反例:变量int a的随意命名方式。

12．【推荐】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;

13．【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变量，如果一定要定义变量，肯定是与接口方法相关，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 f( );

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。

14.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的形式)。正例: AbstractTranslator实现 Translatable接口。

15.【参考】枚举类名建议带上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类，域成员均为常量，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。16.【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。

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:xxxDT0，xxx为业务领域相关的名称。3）展示对象: xxxVO，xx×一般为网页名称。

4)POJ0是Do/DT0/Bo/vo的统称，禁止命名成xxXPOJ0。

(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:String key = "Id#taobao\_" + tradeId;

cache. put(key ,value) ;

2．【强制】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。

说明: Long a = 2l;写的是数字的21，还是Long型的2?

3．【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．【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．【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;

(三)代码格式

1．【强制】大括号的使用约定。如果是大括号内为空，则简洁地写成{手即可，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．【强制】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。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:运算符包括赋值运算符=、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。

正例:(涉及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" ) ;

1/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:

l/这是示例注释，请注意在双斜线之后有一个空格string ygb = new String( ) ;

7．【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:

stringBuffer sb = new StringBuffer( );

//超过120个字符的情况下，换行缩进4个空格，点号和方法名称一起换行sb.append( "zi" ).append( "xin " )...

.append( "huang" )...

.append( " huang" ) ....append( " huang" );反例:

stringBuffer sb = new stringBuffer( );//超过120个字符的情况下，不要在括号前换行sb.append( "zi" ).append( "xin" )...append

( " huang" );

//参数很多的方法调用可能超过120个字符，不要在逗号前换行method(args1,args2,args3, ...

, argsX);

8．【强制】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args1，后边必须要有一个空格。

method(args1,args2，args3);

9．【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。

10．【推荐】单个方法的总行数不超过80行。

说明:包括方法签名、结束右大括号、方法内代码、注释、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，使主干代码更加清晰;共性逻辑抽取成为共性方法，便于复用和维护。

11．【推荐】没有必要增加若干空格来使某一行的字符与上一行对应位置的字符对齐。

正例:

int one = 1;long two = 2L;float three = 3F;

stringBuffer sb = new stringBuffer();

说明:增加sb这个变量，如果需要对齐，则给a、b、c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。

12.【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)注释规约

1．【强制】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

l/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．【强制】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.【强制】所有的类都必须添加创建者和创建日期。

4．【强制】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．【强制】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．【推荐】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:“TC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.【推荐】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．【参考】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.【参考】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。

10.【参考】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:

l / put elephant into fridgeput(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。

11.【参考】特殊注释标记，请注明标记人与标记时间。注意及时处理这些标记，通过标记扫描，

经常清理此类标记。线上故障有时候就是来源于这些标记处的代码。

1)待办事宜(TOD0):(标记人，标记时间，[预计处理时间]>

表示需要实现,但目前还未实现的功能。这实际上是一个Javadoc的标签,目前的Javadoc还没有实现，但已经被广泛使用。只能应用于类，接口和方法(因为它是一个Javadoc标签)。2)错误，不能工作(FIXME):(标记人，标记时间，[预计处理时间])

在注释中用FIXME标记某代码是错误的，而且不能工作，需要及时纠正的情况。

(五)其它

1．【强制】在使用正则表达式时，利用好其预编译功能，可以有效加快正则匹配速度。

说明:不要在方法体内定义: Pattern pattern = Pattern.compile(“规则");

2．【强制】velocity调用Po0类的属性时，建议直接使用属性名取值即可，模板引擎会自动按

规范调用PO0的getXxx()，如果是boolean基本数据类型变量(boolean命名不需要加is前缀），会自动调用isxxx()方法。

说明:注意如果是Boolean包装类对象，优先调用getxxx()的方法。

3．【强制】后台输送给页面的变量必须加$ ! {var}——中间的感叹号。

说明:如果var等于null或者不存在，那么${var}会直接显示在页面上。

4．【强制】注意Math.random()这个方法返回是double类型，注意取值的范围0sx<1(能够

取到零值，注意除零异常)，如果想获取整数类型的随机数，不要将x放大10的若干倍然后取整，直接使用Random对象的nextInt或者nextLong方法。

5．【强制】获取当前毫秒数 system.currentTimeMillis();而不是new Date().getTime( );

说明:如果想获取更加精确的纳秒级时间值，使用system.nanoTime()的方式。在JDK8中，针对统计时间等场景，推荐使用Instant类。

6．【推荐】不要在视图模板中加入任何复杂的逻辑。

说明:根据MVC理论，视图的职责是展示，不要抢模型和控制器的活。

1. 【推荐】任何数据结构的构造或初始化，都应指定大小，避免数据结构无限增长吃光内存。
2. 8．【推荐】及时清理不再使用的代码段或配置信息。

说明:对于垃圾代码或过时配置，坚决清理干净，避免程序过度臃肿，代码冗余。

正例:对于暂时被注释掉，后续可能恢复使用的代码片断，在注释代码上方，统一规定使用三个斜杠(///)来说明注释掉代码的理由。

1. 异常日志

(一)异常处理

1. 【强制】Java类库中定义的可以通过预检查方式规避的RuntimeException异常不应该通过

catch 的方式来处理，比如:NullPointerException，IndexoutOfBoundsException等等。说明:无法通过预检查的异常除外，比如，在解析字符串形式的数字时，不得不通过catchNumberFormatException来实现。

正例: if(obj != null) ...})

反例: try { obj.method(); } catch (NullPointerException e) {...}

2.【强制】异常不要用来做流程控制，条件控制。

说明:异常设计的初衷是解决程序运行中的各种意外情况，且异常的处理效率比条件判断方式要低很多。

3． 【强制】catch时请分清稳定代码和非稳定代码，稳定代码指的是无论如何不会出错的代码。

对于非稳定代码的catch尽可能进行区分异常类型，再做对应的异常处理。

说明:对大段代码进行try-catch，使程序无法根据不同的异常做出正确的应激反应，也不利于定位问题，这是一种不负责任的表现。

正例:用户注册的场景中，如果用户输入非法字符，或用户名称已存在，或用户输入密码过于简单，在程序上作出分门别类的判断，并提示给用户。

4．【强制】捕获异常是为了处理它，不要捕获了却什么都不处理而抛弃之，如果不想处理它，请

将该异常抛给它的调用者。最外层的业务使用者，必须处理异常，将其转化为用户可以理解的内容。

5．【强制】有try块放到了事务代码中，catch异常后，如果需要回滚事务，一定要注意手动回

滚事务。

6．【强制】finally块必须对资源对象、流对象进行关闭，有异常也要做try-catch。

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

7．【强制】不要在finally块中使用return。

说明: finally块中的return返回后方法结束执行，不会再执行try块中的return语句。

8．【强制】捕获异常与抛异常，必须是完全匹配，或者捕获异常是抛异常的父类。

说明:如果预期对方抛的是绣球，实际接到的是铅球，就会产生意外情况。

9．【推荐】方法的返回值可以为null，不强制返回空集合，或者空对象等，必须添加注释充分

说明什么情况下会返回null值。

说明:本手册明确防止NPE是调用者的责任。即使被调用方法返回空集合或者空对象，对调用

者来说，也并非高枕无忧，必须考虑到远程调用失败、序列化失败、运行时异常等场景返回null的情况。

10．【推荐】防止NPE，是程序员的基本修养，注意NPE产生的场景:

1）返回类型为基本数据类型，return包装数据类型的对象时，自动拆箱有可能产生NPE。

反例: public int f() { return Integer对象}，如果为null，自动解箱抛NPE。2)数据库的查询结果可能为null。

3)集合里的元素即使isNotEmpty，取出的数据元素也可能为null。4）远程调用返回对象时，一律要求进行空指针判断，防止NPE。5)对于Session中获取的数据，建议NPE检查，避免空指针。

6)级联调用obj.getA( ) .getB( ).getc():一连串调用，易产生NPE。

正例:使用DK8的 optional类来防止NPE问题。

11．【推荐】定义时区分unchecked / checked异常，避免直接抛出 new RuntimeException()，

更不允许抛出Exception或者Throwable，应使用有业务含义的自定义异常。推荐业界已定义过的自定义异常，如:DAOException / serviceException等。

12.【参考】对于公司外的http/api开放接口必须使用“错误码";而应用内部推荐异常抛出;

跨应用问RPC调用优先考虑使用Result方式，封装issuccess()方法、“错误码”、“错误简短信息”。

说明:关于RPC方法返回方式使用Result方式的理由:

1）使用抛异常返回方式，调用方如果没有捕获到就会产生运行时错误。2〉如果不加栈信息，只是new自定义异常，加入自己的理解的error message，对于调用端解决问题的帮助不会太多。如果加了栈信息，在频繁调用出错的情况下，数据序列化和传输的性能损耗也是问题。

13.【参考】避免出现重复的代码(Don't Repeat Yourself)，即 DRY原则。

说明:随意复制和粘贴代码，必然会导致代码的重复，在以后需要修改时，需要修改所有的副本，容易遗漏。必要时抽取共性方法，或者抽象公共类，甚至是组件化。

正例:一个类中有多个public方法,都需要进行数行相同的参数校验操作，这个时候请抽取:

private boolean checkParam(DTo dto) {...}

(二)日志规约

1.【强制】应用中不可直接使用日志系统〈Log4j、Logback)中的API，而应依赖使用日志框架

SLF4]中的API，使用门面模式的日志框架，有利于维护和各个类的日志处理方式统一。

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

private static final Logger logger = LoggerFactory.getLogger(Abc.class);

20/38

阿里巴巴Java开发手册