Java语言编码规范

目录

[Java语言编码规范 1](#_Toc38617795)

[一、编程规约 3](#_Toc38617796)

[(一)命名规约 3](#_Toc38617797)

[(二)常量定义 5](#_Toc38617798)

[(三)格式规约 6](#_Toc38617799)

[(四)00P规约 8](#_Toc38617800)

[(五)控制语句 11](#_Toc38617801)

[二、异常日志 14](#_Toc38617802)

[(一)异常处理 14](#_Toc38617803)

## 一、编程规约

### (一)命名规约

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/$0bject/name\_/name$/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。

反例:MoShiChange[模式改变]/getJianZhuByName()[获取建筑]/int变量=0

正例:alibaba/taobao/youku/hangzhou等国际通用的名称，可视同英文。

3.类名使用UpperCamelCase风格，必须遵从驼峰形式，但以下情形例外:(领域模型的相关命名)DO/BO/DTO/Vo等。

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:请勿使用Stringargs[]的方式来定义。

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P0J0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:P0J0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IPE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: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PO]O类时，不要设定任何属性默认值。

反例:P0]O类的gmtCreate默认值为newDate();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>generatetoString时，如果继承了另一个POJ0类，注意在前面加一下super.toString。

说明:在方法执行拋出异常时，可以直接调用P0]0的toString()方法打印其属性值，便于排查问题。

13.接口和实现类的命名有两套规则:

1. 对于Service和DA0类，基于S0A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。
2. 正例:CacheServiceImpl实现CacheService接口。
3. 如果是形容能力的接口名称,取对应的形容词做接口名(通常是-able的形式)。

正例:AbstractTranslator实现Translatable。

14.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常量类，且构造方法被默认强制是私有。

正例:枚举名字:DealStatusEnum，成员名称:SUCCESS/UNKOWN\_REASON。

15.各层命名规约:

1. Service/DA0层方法命名规约
2. 获取单个对象的方法用get做前缀。
3. 获取多个对象的方法用list做前缀。
4. 获取统计值的方法用count做前缀。
5. 插入的方法用save(推荐)或insert做前缀。
6. 删除的方法用remove，(推荐)或delete做前缀。
7. 修改的方法用update做前缀。
8. 领域模型命名规约
   1. 数据对象:xxxD0，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:xxxV0，xxx一-般为网页名称。
   4. P0J0是DO/DTO/B0/V0的统称，禁止命名成xxxP0J0。

### (二)常量定义

1.不允许出现任何魔法值(即未经定义的常量)直接出现在代码中。

反例:Stringkey="Id#taobao\_"+tradeId:cache.put(key,value);

2.long或者Long初始赋值时，必须使用大写的L，不能是小写的1，小写容易跟数字

1混淆，造成误解。

说明:Longa=21;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，应该按常量功能进行归类,分开维护。如:缓存相关的常量放在类:CacheConsts下:系统配置相关的常量放在类:ConfigConsts下。

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。

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:放置在一方库的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:publicstaticfinalStringYES="yes";类B中:publicstaticfinalStringYES="y";A.YES.equals(B.YES)，预期是true，但实际返回为false,导致产生线上问题。

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privatestaticfinal定义。

5.如果变量值仅在-一个范围内变化用Enum类。如果还带有名称之外的延伸属性，必须使用Enum类，下面正例中的数字就是延伸信息，表示星期几。

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

### (三)格式规约

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:表示终止右大括号后必须换行。

2．左括号和后-一个字符之间不出现空格:同样，右括号和前一个字符之间也不出现空格。详见第5条下方正例提示。

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

4.任何运算符左右必须加一个空格。

说明:运算符包括赋值运算符=、逻辑运算符&、加减乘除符号、三目运行符等。

5.缩进采用4个空格，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Usetabcharacter;而在eclipse中，必须勾选insertspacesfortabs.

正例:

(涉及1-5点)

publicstaticvoidmain(Stringargs[]){

//缩进4个空格

Stringsay="hello";

//运算符的左右必须有一个空格

intflag=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.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-起换行。

4)在多个参数超长，逗号后进行换行。

5)在括号前不要换行，见反例。

正例:

StringBuffersb=newStringBuffer();

//超过120个字符的情况下，换行缩进4个空格，并且方法前的点符号--起换行

sb.append("zi").append("xin")...

.append("huang")...

.append("huang")...

.append("huang");

反例:

StringBuffersb=newStringBuffer();

//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);

7.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");

8.IDE的textfileencoding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。

9.没有必要增加若干空格来使某一行的字符与上一行的相应字符对齐。

正例:

inta=3;

longb=4L;

floatc=5F;

StringBuffersb=newStringBuffer();

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，是一种累赘的事情。

10.方法体内的执行语句组、变量的定义语句组、不同的业务逻辑之间或者不同的语义之间插入-一个空行。相同业务逻辑和语义之间不需要插入空行。

说明:没有必要插入多行空格进行隔开。

### (四)00P规约

1.避免通过--个类的对象引用访间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。

反例:get0bject()与getObject()的问题。--个是字母的0，一个是数字的0，加@verride.

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:publicUsergetUsers(Stringtype,Integer...ids)

4.对外暴露的接口签名，原则.上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(StringencodeStr)这个方法已经过时，应

该使用双参数decode(Stringsource,Stringencode)。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.objectst#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Integervar=?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.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P0]0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: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PO]O类时，不要设定任何属性默认值。

反例:PO]O类的gmtCreate默认值为newDate();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>generatetoString时，如果继承了另一个P0J0类，注意在前面加一下super.toString。

说明:在方法执行拋出异常时，可以直接调用P0]0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，否则会有拋IndexOutOfBoundsException的风险。

说明:

Stringstr="a,b,c,,";

String[]ary=str.split(",");

//预期大于3，结果是3

System.out.println(ary.length);

14.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。

15.类内方法定义顺序依次是: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:因为方法信息价值较低，所有Service和DAO的getter/setter方法放在类体最后。

16.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，尽量不要增加业务逻辑，增加排查问题的难度。

反例:publicIntegergetData(){

if(true){

returndata+100;

}else{

returndata-100;

}

17.循环体内，字符串的联接方式，使用StringBuilder的append方法进行扩展。

反例

Stringstr="start";

for(inti=0;i<100;i++){

str=str+"hello";

说明: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。

18.final可提高程序响应效率，声明成final的情况:

1)不需要重新赋值的变量，包括类属性、局部变量。

2)对象参数前加final,表示不允许修改引用的指向。

3)类方法确定不允许被重写。

19.慎用object的clone万法米拷贝对家。

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方法实现属性对象的拷贝。

20.类成员与方法访问控制从严:

1)如果不允许外部直接通过new来创建对象，那么构造方法必须是private。

2)工具类不允许有public或default构造方法。

3)类非static成员变量并且与子类共享，必须是protected。

4)类非static成员变量并且仅在本类使用，必须是private。

5)类static成员变量如果仅在本类使用，必须是private。

6)若是static成员变量，必须考虑是否为final。

7)类成员方法只供类内部调用，必须是private。

8)类成员方法只对继承类公开，那么限制为protected。

说明:任何类、方法、参数、变量，严控访问范围。过宽泛的访问范围，不利于模块解耦。思考:如果是一个private的方法，想删除就删除，可是一个public的Service方法，或者一个public的成员变量，删除一下，不得手心冒点汗吗?变量像自己的小孩，尽量在自己的视线内，变量作用域太大，如果无限制的到处跑，那么你会担心的。

### (五)控制语句

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，即使它什么代码也没有。

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，避免使用

下面的形式:if(condition)statements;

3.推荐尽量少用else，if-else的方式可以改写成:

if(condition){

returnobj;

}

//接着写else的业务逻辑代码;

说明:如果非得使用if()...elseif()...else...方式表达逻辑，请勿超过3层，

超过请使用状态设计模式。

正例:逻辑上超过3层的if-else代码可以使用卫语句，或者状态模式来实现。

4.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

booleanexisted=(file.open(fileName,"w")!=null)&(...I1(...);

if(existed){

反例:if((file.open(fileName,"w")!=null)&&(...)11(...)){

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.接口入参保护，这种场景常见的是用于做批量操作的接口。

7.方法中需要进行参数校验的场景:

1)调用频次低的方法。

2)执行时间开销很大的方法，参数校验时间几乎可以忽略不计，但如果因为参数错误导致中间执行回退，或者错误，那得不偿失。

3)需要极高稳定性和可用性的方法。

4)对外提供的开放接口，不管是RPC/API/HTTP接口。

5)敏感权限入口。

8.方法中不需要参数校验的场景:

1)极有可能被循环调用的方法，不建议对参数进行校验。但在方法说明里必须注明外部参数检查。

2)底层的方法调用频度都比较高，-般不校验。毕竟是像纯净水过滤的最后--道，参数错误不太可能到底层才会暴露问题。--般DAO层与Service层都在同一个应用中，部署在同一台服务器中，所以DAO的参数校验，可以省略。

3)被声明成private只会被自己代码所调用的方法，如果能够确定调用方法的代码传入参数已经做过检查或者肯定不会有问题，此时可以不校验参数。

(八)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--并说明。

3.所有的类都必须添加创建者信息。

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.注释掉的代码尽量要配合说明，而不是简单的注释掉。

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，对于注释的要求:第--、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.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:

//putelephantintofridge

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。

11.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(TODO):(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。

## 二、异常日志

### (一)异常处理

1.不要捕获Java类库中定义的继承自RuntimeException的运行时异常类，如：

IndexOutOfBoundsException/NullPointerException，这类异常由程序员预检查来规避，保证程序健壮性。

正例：if(obj!=null){...}

反例：try{obj.method()}catch(NullPointerExceptione){...}

2.异常不要用来做流程控制，条件控制，因为异常的处理效率比条件分支低。

3.对大段代码进行try-catch，这是不负责任的表现。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。

4.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.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.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，可以使用try-with-resources方式。

7.不能在finally块中使用return，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.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.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。调用方需要进行null判断防止NPE问题。

10.说明：本规约明确防止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，运行时异常等场景返回null的情况。防止NPE，是程序员的基本修养，注意NPE产生的场景：

1. 返回类型为包装数据类型，有可能是null，返回int值时注意判空。反例：publicintf(){returnInteger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，一律要求进行NPE判断。
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。

11.在代码中使用“抛异常”还是“返回错误码”，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；跨应用间RPC调用优先考虑使用Result方式，封装isSuccess、“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：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。如果不加栈信息，只是new自定义异常，加入自己的理解的errormessage，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。

12.定义时区分unchecked/checked异常，避免直接使用RuntimeException抛出，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。

13.避免出现重复的代码（Don’tRepeatYourself），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，甚至是共用模块。

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：privatebooleancheckParam(DTOdto){...}