Java后台代码开发规范(综合工具项目组)

本规范参考阿里巴巴java代码规范

# 1.编程规约

## 命名风格

1.代码中的命名风格不能以下划线或美元符号开始也不能以此结束

反例：\_name / name\_ $name / name$

2.代码中严禁使用拼音英文混合使用

反例：getxingmingById

3.类名采用UpperCamelCase(大驼峰式)写法

例子：UserManage

4.常量命名全部大写

例子 public static final String MAX\_INT=9999

5.方法名,属性名，参数名，成员变量统一使用小驼峰写法

6.异常类定义由Exception结尾,测试类定义由Test结尾

7.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。

8. POJO 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.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.hoperun.util、类名为MessageUtils（此规则参考spring 的框架结构）

10.接口和实现类的命名规则：

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。

11.【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。

12.java代码需要增加javadoc注释，新建的实体类必须增加作者

## 代码规范

* + - 1. 不允许出现魔鬼数字

反例：return(“你好”)

* + - 1. 【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。

* + - 1. 【推荐】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，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。
      2. 【强制】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。

正例：

List<String> list = new ArrayList<>();

list.add("1");

list.add("2");

Iterator<String> iterator = list.iterator();

while (iterator.hasNext()) {

String item = iterator.next();

if (删除元素的条件) {

iterator.remove();

} }

反例

for(String item: mylist){

mylist.remove();

}

所有方法都要加javadoc注释，并写出方法用途。

所有类名都要加作者

# 2.异常处理

运行时异常runtimException 不需要通过catch来处理比如空指针异常，越界异常等

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. 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. 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。

# 3.日志规范

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SLF4J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);

日志命名规则

日志按类型可分为接口日志和运行时日志，运行日志统一存放在 /模块名/run.log ,接口日志定义为 /模块名/access.log

对trace/debug/info级别的日志输出，必须使用条件输出形式或者使用占位符的方式。 说明：logger.debug("Processing trade with id: " + id + " and symbol: " + symbol); 如果日志级别是warn，上述日志不会打印，但是会执行字符串拼接操作，如果symbol是对象，会执行toString()方法，浪费了系统资源，执行了上述操作，最终日志却没有打印。 正例：（条件）建设采用如下方式

if (logger.isDebugEnabled()) {

logger.debug("Processing trade with id: " + id + " and symbol: " + symbol);

}

正例：（占位符）

logger.debug("Processing trade with id: {} and symbol : {} ", id, symb)

异常信息应该包括两类信息：案发现场信息和异常堆栈信息。如果不处理，那么通过关键字throws往上抛出。 正例：logger.error(各类参数或者对象toString() + "\_" + e.getMessage(), e);