链滴

《阿里巴巴 Java 开发手册 》二、异常日志

作者：junjiecheng

原文链接：https://ld246.com/article/1534226495960

来源网站：链滴

(一)异常处理

1.【强制】Java 类库中定义的一类 RuntimeException可以通过预先检查进行规避，而不应该通过 catch 来处理

比如：IndexOutOfBoundsException，NullPointerException等等。

说明：无法通过预检查的异常除外，如在解析一个外部传来的字符串形式数字时，通过 catch NumberFormatException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，因为异常的处理效率比条件分支低。

3.【强制】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.【强制】捕获异常是为了处理它

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.【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

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 null值。调用方需要进行 null判断防止 NPE问题。

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 JDK8的 Optional类来防止 NPE问题。

11.【推荐】定义时区分 unchecked/checked 异常

定义时区分 unchecked/checked 异常,避免直接抛出 newRuntimeException()，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】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。

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);

2.【强制】日志文件推荐至少保存 15天，因为有些异常具备以“周”为频次发生的特点。

3.【强制】应用中的扩展日志（如打点、临时监控、访问日志等）命名方式：

appName_logType_logName.log。

logType:日志类型，推荐分类有 stats/desc/monitor/visit 等；

logName:日志描述。这种命名的好处：通过文件名就可知道日志文件属于什么应用，什么类型，什目的，也有利于归类查找。

正例：mppserver应用中单独监控时区转换异常，如：

mppserver_monitor_timeZoneConvert.log

说明：推荐对日志进行分类，如将错误日志和业务日志分开存放，便于开发人员查看，也便于通过日对系统进行及时监控。

4.【强制】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）

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, symbol);

5.【强制】避免重复打印日志，浪费磁盘空间，务必在 log4j.xml中设置 additivity=false。

正例：

6.【强制】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);

7.【推荐】谨慎地记录日志。

生产环境禁止输出 debug日志；有选择地输出 info日志；如果使用 warn来记录刚上线时的业务行为息，一定要注意日志输出量的问题，避免把服务器磁盘 撑爆，并记得及时删除这些观察日志。

说明：大量地输出无效日志，不利于系统性能提升，也不利于快速定位错误点。记录日志时请

思考：这些日志真的有人看吗？看到这条日志你能做什么？能不能给问题排查带来好处？

8.【参考】可以使用 warn日志级别来记录用户输入参数错误的情况，避免用户投诉时，无所适从。

注意日志输出的级别，error级别只记录系统逻辑出错、异常等重要的错误信息。如非必要，请不要在场景打出 error级别。