1 编码规范

文档属性

目录

- 添加必要的注释与日志
- 程序结构主次分明
- 对方法输入参数进行合法性检查
- 对方法调用返回结果进行判断
- 分支判断要闭环
- 对异常情况进行处理
- 合理进行事务控制
- 合理进行并发控制
- 接口设计可扩展
- 数据库变更需谨慎

Version	Editor	Update Date Time	Detail
v1.0	eric.zhang	2021-07-03	初版发布

添加必要的注释与日志

具体文档模板与代码风格模板见 IDE代码风格模板与规范

注释

【强制】所有的类都必须添加创建者和创建日期。

【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。

日志打印

【强制】必须打印跨系统调用请求参数、响应结果等；业务日志必须包含关键业务字段，例如业务单号、userId等；以便于问题排查；

【强制】非系统预期或者业务预期的情况需要打印日志并上报监控，例如：调用中间件（缓存、消息队列）出现未知异常或访问数据库返回空；

程序结构主次分明

【推荐】程序结构应逻辑清晰，主次分明。可类比书本的结构，主方法里面是骨干逻辑类似目录；具体实现细节在子方法里面，类似目录结构中的各子章节；

【推荐】单个程序、单个方法职责单一；不可将多个逻辑耦合在一个程序或者方法里面；

【推荐】单个方法包括方法签名、左右大括号、方法内代码、空行、回车及任何不可见字符的总行数不超过80行（不包含注释）。

正例：代码链接，代码逻辑清晰，主方法里面只有主体逻辑，具体处理逻辑封装为子方法，起到化繁为简的作用。

反例：代码链接，代码逻辑没有主次，所有处理逻辑全部写在一个大方法里面，不便于维护；另外单个方法太大，复杂度提升，容易出现bug。

对方法输入参数进行合法性检查

【强制】编写方法首先对传入的所有参数进行合法性检查。例如账号非空检查；金额范围检查；币种合法性检查等。

【强制】参数检查逻辑放在业务处理逻辑之前，遵循“先检查，再处理”的原则。

反例：代码链接，对金额、账号等进行合法性检查，一边检查一边处理业务逻辑。正确的做法是把检查逻辑抽成一个方法，在此方法里面进行业务合法性检查；主方法针对失败情况先处理，然后做正常业务的处理。

对方法调用返回结果进行判断

【强制】调用本地方法或远程服务必须对返回结果进行判断，根据调用结果进行相应处理；远程方法调用尤其需要考虑失败或者状态未知的情况。

正例：代码链接，业务中心调用支付，对支付结果按成功、失败、未知分别做对应的处理；

正例：代码链接，更新还款流水表判断affectRows；当affectRows为0时抛出异常回滚事务；

反例：代码链接，更新还款单据未判断affectRows；当affectRows为0时未考虑，直接按正常逻辑处理；

分支判断要闭环

【强制】分支逻辑判断需要对各种可能情况考虑全面，不可只考虑正常分支而忽略异常分支。

【强制】在一个switch块内，每个case要么通过continue/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。

说明：注意break是退出switch语句块，而return是退出方法体。

反例：代码链接，还款类型有7种枚举值，分支判断只处理了其中的5种情况；另外2种情况未做任何处理。

正例：代码链接，process方法按operationType分别处理，switch块包含default语句兜底。

对异常情况进行处理

【强制】对外提供的开放api必须使用“错误码”；跨应用间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”；而应用内部推荐抛出异常，不允许直接抛出RuntimeException，更不允许抛出Exception或者Throwable，必须是有业务含义的自定义异常；

【强制】catch异常尽可能区分异常类型，再做对应的异常处理；在调用二方包、或动态生成类的相关方法时，catch异常必须使用Throwable类来进行拦截；最外层的服务必须catch住Throwable类型异常，并将其转化为用户可以理解的内容；

说明： 通过反射机制来调用方法，如果找不到方法抛出NoSuchMethodException。什么情况会抛出NoSuchMethodError呢？二方包在类冲突时，仲裁机制可能导致引入非预期的版本使类的方法签名不匹配，或者在字节码修改框架（比如： ASM）动态创建或修改类时，修改了相应的方法签名。这些情况，即使代码编译期是正确的，但在代码运行期时会抛NoSuchMethodError。因此catch这类异常需要用Throwable类来进行拦截。

合理进行事务控制

【强制】事务作用范围要明确；对数据库的操作必须在事务作用范围内。强制使用编程式事务。事务注解坑比较多，稍有不慎事务失效，详细见@Transactional注解容易产生的问题。

【推荐】尽量不要在事务中调用远程服务；一方面，可能出现远程服务调用成功本地事务失败回滚从而导致数据不一致；另一方面，远程服务可能耗时较长导致本地数据库连接不能及时释放进而可能耗尽数据库连接资源。

【强制】调用远程服务前必须先保存本地单据并提交事务；远程调用的流水号必须从数据库流水号字段获取；进行业务重试原则上使用原单据流水号发起重试。

正例：代码链接，具体见linkageBind方法，数据库操作通过事务管理起来，并且事务内只有本地数据库操作；TransactionExecutor封装了事务提交与回滚。

反例：代码链接，事务中包含远程调用invokeCoreQuery，则可能因远程调用耗时久引起本地数据库连接释放缓慢进而引起数据库连接不足；如果远程方法是写接口，则可能导致本地数据库和远程数据库的数据不一致。

反例：代码链接，支付调用核心记账时每次生成新的资金流水号，存在重复记账风险。

合理进行并发控制

【强制】数据库表必须增加业务唯一键；

反例：收单支付流水表缺少业务唯一键，存在重复记账风险；增加BANK_REF + TRAN_TYPE 作为唯一键；BBW往账acct_info表没有业务唯一键，存在重复记账风险；增加ref_no + settle_type + settle_step作为唯一键。

【强制】通过乐观锁或悲观锁防止数据库操作并发；乐观锁建议通过版本号字段控制；悲观锁必须先开启事务，“一锁二判断三更新”；

正例：代码链接，具体见updateStatus方法：先开启事务，selectLockByInternalKey悲观锁锁住单据（具体sql如下）；接着判断业务状态是否合法；然后更新业务单据，提交事务。

```
select *
```

```
from PT_INFO_TRAN_MSG

where INTERNAL_KEY = #{internalKey,jdbcType=DECIMAL}

for update;
```

【强制】通过java锁机制控制同一应用实例多线程并发处理；通过分布式锁（zookeeper或redis）控制同一集群多线程并发处理；跨集群并发控制建议通过数据库唯一键控制并发处理（因为多集群场景下zookeeper或redis可能独立部署，无法保证全局唯一）；

正例：代码链接，具体见execute方法；该方法被一个批量调度程序调用，为了避免同一个任务被多个pod处理，通过zookeeper控制集群内并发。

接口设计可扩展

【强制】提供给其他系统调用的接口，方法参数应可扩展；建议封装为DTO，避免新增参数后方法签名发生变化；

【强制】修改接口尽量增加字段，而不是删除原有字段或者修改原有字段；

数据库变更需谨慎

【强制】禁止删除数据库表、列；禁止修改数据库表名、列名；

【强制】批量数据订正要灰度；分成多个批次，第一批验证无误之后才进行后续批次订正；

反例：银行核心在v2.1_0702版本发布时rename一个正在使用的表，导致线上13笔转账异常。