应用系统体系架构 — 作业2

# 学号： 522031910213 姓名： 朱涵 得分：

## A.

## 在我的addOrder方法中应有四个需要使用事务保持原子性的子方法 分别是修改书籍库存和销售量的saveBook、写入order的saveOrder和写入orderItem的saveOrderItem、和删除对应购物车项的deleteCartItem。

## 使用@Transaction对这三个方法进行注解时 都选择**默认的REQUIRED**传播行为 这样使得子方法都可以加入父方法的事务之中 做到发生回滚就全部方法都回滚 如果设置为REQUIRED\_NEW 某一个子方法内发生了回滚 子方法外的其他子方法和父方法都不会回滚 这是不符合设计原则的 比如saveOrder内部发生回滚 订单没有成功写入 其他数据（比如book和order）却成功写入了 导致了数据不一致

## 下表是实际测试出的结果。由于orderItem需要从属的orderId 所以必须先写入order后再写入orderItems。

## 唯一与课件中有区别的是当后者设置为REQUIRED\_NEW时 会出现异常 推测是因为saveOrder方法加入了父方法的事务 在整个方法结束前不会提交 而saveOrderItems新建了一个事务 在父方法的事务未提交前就进行了保存操作 此时数据库内的order表里缺少对应的记录 并且新的事务也读取不到未提交的数据 导致orderItems的外键无法正确建立 于是无法进行下去 则报错了

## 另外有一个小疑惑 如果助教方便的话可以在canvas的评价里解答一下 非常感谢：

## 我原本的实现是往Order对象里添加OrderItems 然后只调用OrderDAO对order进行保存 由于设置了级联 会一并把orderItems也保存到数据库里 似乎没有必要分开调用两个不同的DAO来分别保存订单和订单项 不知道这两者有什么区别

|  |  |  |  |
| --- | --- | --- | --- |
| addOrder | saveOrder | saveOrderItems | result |
| 正常 | 正常 | 正常 | 订单和订单项都成功保存 |
| result = 10 / 0 | 正常 | 正常 | 整个事务回滚 |
| 正常 | result = 10 / 0 | 正常 | 整个事务回滚 |
| 正常 | 正常 | result = 10 / 0 | 整个事务回滚 |
| 正常 | 正常 | 正常 REQUIRES\_NEW | 订单项回滚 订单等其他数据成功保存 |
| result = 10 / 0，在saveOrderItems之前 | 正常 | 正常 REQUIRES\_NEW | 整个事务回滚 |
| result = 10 / 0，在saveOrderItems之后 | 正常 | 正常 REQUIRES\_NEW | 订单项成功保存 订单等其他数据回滚 |
| 正常 | result = 10 / 0 | 正常 REQUIRES\_NEW | 整个事务回滚 |
| 正常 | 正常 | result = 10 / 0REQUIRES\_NEW | 订单项数据回滚 订单等其他数据成功保存 |

## B.

## 选用第三种方式返回响应。

## 实现思路：

## 在后端配置一个websocket的端点 这个端点不会收到消息 只需要封装一个sendToUser函数用于给前端的用户推送消息即可 sessionMap采用uid做键

## 实现端点后 在finishOrder的监听器中向用户发送消息 如果处理成功 发送“订单成功处理” 否则发送“订单处理失败+{异常信息}”

## 前端在购物车页面加入websocket 使得进入购物车页面后与ws端点建立连接 关闭页面后关闭连接 避免不必要的资源浪费 为onmessage方法配置一个简单的alert 用户将会在收到消息时看到一个提醒框

## 消息格式：

## 简单的字符串 “订单成功处理”或者“订单处理失败+{异常信息}” 前端只需要直接呈现即可 后续若需要更详细的信息 可以封装为自定义的Result 前端解析为json后进行呈现

## Topic配置项：

## 两个topic AddOrder负责接受controller发送的消息 然后调用service层的addOrder方法进行业务处理 然后把结果封装好发送给FinishOrder话题 FinishOrder监听器会根据结果的状态码调用websocket的发送消息方法给前端

## 订单处理结果呈现：

## 目前简单的使用alert方法告知用户是否成功 以及订单处理失败的原因

## 3种处理方式的优劣：

## 1 使用nodejs前端直接连接kafka

## 优点：无需其他中间件 直接通过前端连接消息队列获取对应topic的消息 比较方便

## 缺点：需要手动刷新 对于用户不友好 不知道何时处理完

## 2 使用ajax请求

## 优点：编写代码简单 直接在前端调用get的方法即可

## 缺点：需要定时发送请求 不知道何时处理完 轮询会浪费带宽资源

## 3 使用websocket推送

## 优点：后端处理完业务后自动推送给前端 用户只需要等待结果即可 无需任何操作 即时反馈 且建立连接后使用心跳维持 数据传输开销比较小

## 缺点：需要编写websocket端点 增加代码复杂性 某些浏览器可能不支持ws 兼容性不好

## 右图是订单处理成功后的响应结果：