17. 支付新手高频易发故障及应对

大家好，我是隐墨星辰，专注境内/跨境支付架构设计十余年。

一路走来，我犯过很多错，也引发过很多线上故障。在带团队后，也不断看到团队内的新人频繁出现各种线上故障。这些故障或多或少都对公司的业务和新人的心理造成了一定的负面影响。

这里汇总一些新手高频易发的故障，以及应对策略，少点故障，安心过年。主要包括：渠道返回码，乱序，三高（高并发、高性能、高可用），幂等，多币种等。

1. 对外部渠道的返回码理解一知半解

无论是四方还是三方，大部分情况下都是调用外部渠道或银行做真实的扣款，而渠道对接是新手触发故障的高风险区。

首先就是渠道返回码的映射。**最常见的就是把渠道返回“系统异常”、“订单不存在”、或调用超时，推进到失败。**

我现在还记得我接入的第一个外部渠道，吭哧吭哧前后忙活了1个多月，终于上线，开量，稳定运行，非常开心。

有天财务小妹找到我，说我接的那个渠道有笔支付订单对账出了长款，我们的订单是失败的，但是渠道给的对账文件是成功的。我不敢大意，赶紧去查日志和数据库的数据，原来是渠道支付接口超时，查询接口返回“订单不存在”，被映射到了“失败”。

当时被老板被狠批一顿：“你怎么犯这种低级错误！怎么能把订单不存在映射成失败呢？”那时年轻，毫不客气地回敬道：“这是银行的问题哪，他都告诉我订单不存在了，我咋还不能推进到失败呢？我不推进到失败我推进到哪里？”

虽然嘴硬，心还是虚的。在网络环境下，数据包走不同的路由分发，银行内部多个系统之间有延时，最后导致查询比支付先到，或者查询时数据还没有落库，银行返回订单不存在，很正常的。

梳理出大致如下几条准则：

1. 只有明确成功，才能推进成功。
2. 只有明确失败，才能推进失败。
3. 结果不明确，默认全部是“未知”。
4. 支付场景谨慎推进到成功，退款、提现、商家打款等场景谨慎推进到失败。
5. 典型的结果不明确场景有：订单不存在，系统异常，内部错误等。

这部分在“渠道返回码映射设计与最佳实践”的那篇文章里写得更清楚明确，有兴趣的可以历史发文里面找一找。

2. 缺少乱序的概念

人的大脑是顺序思维，但互联网世界是乱序的。

比如以前发生一个线上问题，财务小妹说和建行的对账对出了长款：**银行支付成功，我方不是成功**。一查，原来是渠道异步通知接口先返回了“成功”，单据先被推进到了成功，然后同步接口才返回“系统异常”，单据又被推进到了支付失败。

这里有两个问题：1）不应该把系统异常映射到失败，这个在上面的返回码映射已经说得非常清楚。2）成功是终态，不应该再变更。

第2个问题的解法非常简单，只需要设计好状态机就行。比如终态不可变，只有符合要求的初始状态+指定事件，才能推进到指定的状态。有兴趣的，可去找公号历史发文里面的“支付交易订单状态机设计与最佳实践”，写得非常详细。

不仅仅是外部渠道对接，现在互联网应用都是微服务架构，微服务之间除了api接口，还通过消息来传递数据，而大部分消息是不保证顺序性的。

我时不时听见一些新人质问消息发送方：“为什么你的数据没有顺序，你需要保证你的数据要有顺序，不然我处理不了！”。我只能哑然一笑。

群里也有小伙伴说某某消息中间件可以保证有序性。可是我们如果设计一个系统，不依赖中间件来保证顺序性，不是更好么？极端场景下，就算消息中间件能保证顺序性，但在分布式场景下，我们也无法保证一笔业务只在一台服务器上处理。

3. 缺少三高的概念

“高并发、高可用、高性能”虽然很虚，却又很实在。不同数量级的并发，对系统的要求是截然不同的。“压死骆驼的最后一根稻草”在大规模交易场景下，不是一句笑话，而是真实存在的。

以前很长一段时间我分辨不出“高并发”和“高性能”之间区别，以为两者是一回事。事实不是，高并发是指同时能接多少请求，比如同时处理1百个请求，还是1万个请求，就是并发的概念。高性能是指一个请求需要处理多少时间，比如平均每个请求处理时间是500毫秒，性能就远好于平均处理时间1000毫秒。

举几个例子：

1. **高性能**：用户一般可以使用多个支付方式，比如余额、红包、满减、外部银行等，分别保存在不同的子应用中，没有经验的新人，就按**顺序**去各个应用中查数据。有经验的老人就知道，使用多线程同时去各子应用中查数据。这就是提高性能的方式。
2. **高可用**：以前分享过一个故障，后台操作出现慢查询，在线业务去缓存查数据，击穿到数据库，数据库宕机，进而导致线上业务宕机。最后分析下来，在线业务对那个缓存数据是弱依赖，没有也能部分有损地处理业务。这就是高可用的部分。有兴趣的可以看看公号里的历史发文中的一些故障分享。
3. **高并发**：高并发引发的问题太多了。比如代码new 一个大对象，小流量没事，流量一大，就会出现频繁full gc。比如数据库连接没有按预期释放，小流量也没事，流量一大，连接不够用，就会出现频繁报错。

   这三块在网上有很多资料，边看边学边实践，没什么好办法。

4. 缺少幂等概念

在支付系统中，幂等极其重要，一旦出现幂等失败，经常导致资损，不是平台资损，就是用户资损。

常见有三种：

1. **对外提供的接口升级，导致幂等失效**。现在互联网应用对外的接口都会声明幂等字段，最怕就是内部做升级，不小心把幂等组合修改掉，比如原来使用订单号，后面使用订单号+日期，上游订单号没有变，日期变了，本应该是一笔订单，因为幂等失败，就成了两笔订单。
2. **调用下游接口，把下游依赖的幂等字段内容修改了**。比如下游依赖订单号+创建日期做幂等，上游在架构升级中，传入了订单号+当前日期，在跨零点的交易中，就可能出现幂等失败。
3. **使用redis实现的分布式锁做幂等**。redis的可靠性和持久存储是比不上关系数据库的，最起码需要使用关系数据库的唯一索引做幂等。

幂等在网上有很多资料，值得好好看看。更高阶的，还需要考虑异地多活的幂等场景，A机房挂了，去到B机房，会不会幂等失败。

5. 缺少多币种或金额单位换算概念

在跨境交易中，必然是多币种的，不同的币种，最小单位是不一样的。人民币、美元都很标准，有元有分，但是日元最小单位就是元，印尼盾虽然最小单位是分，但是日常只会用到元。

在微服务框架中，如果一些子应用中使用分，一些子应用使用元，会出现什么情况?

新人还容易犯的一个错误就是**手动加减乘除，容易出现金额放大缩小100倍，或者出现舍入不正确**（舍入有：向下取整，向上取整，四舍五入、银行家算法等）。尤其是渠道对接时，以为内部使用的分，外部渠道接口要求是元，直接**乘**100，但实际上内部应用使用的是元，妥妥资损100倍。

解决方案在公号以前的发文“支付系统金额处理规范及最佳实践”中有详细说明，有兴趣可以去翻翻。

6. 结束语

上面只是举了一些常见的例子，无法覆盖全部。公号里发过一篇“支付系统资损防控指南（精华版）”，是线上故障血淋淋的教训之后总结出来的避坑指南，覆盖绝大部分容易出问题的场景，值得一看。

基本所有的新人都是踩着线上故障在成长。我们能做的，只是提供一些规范或案例，让新人能踩在前人的故障上成长，而不全是踩着自己的线上故障在成长。很多时候一不小心因为一个大故障而被清理门户，成长虽有，代价却过大，得不偿失。

这是《支付通识》专栏系列文章中的第（17）篇。

深耕境内/跨境支付架构设计十余年，欢迎关注并星标公众号“隐墨星辰”，和我一起深入解码支付系统的方方面面。

专栏系列文章PDF合集不定时更新，欢迎关注我的公众号“隐墨星辰”，留言“PDF”获取。

隐墨星辰 公众号

10年顶尖境内/跨境支付公司架构经验

著有《图解支付系统设计与实现》

和我一起解码支付系统方方面面

有个支付系统设计与实现讨论群，添加个人微信（yinmon_sc）备注666进入。

隐墨星辰 个人微信

10年顶尖境内/跨境支付公司架构经验

著有《图解支付系统设计与实现》

备注666进支付讨论群