钱被扣走了，但是订单却未成功！支付掉单异常最全解决方案

作者：9526xu
原文链接：https://ld246.com/article/1602462772843
来源网站：链滴

<h2 id="小黑碎碎念">小黑碎碎念</h2>
Hello，大家好，我是楼下小黑哥~
这个长假不知道大家玩的开心不？有没有出去玩就被堵在路上的？
哈哈，这个长假，我过的还是有点累(□ ^ □)。
长假前两天都在加班迁移旧系统的历史数据，由于这个依赖我同事那边的迁移进度，所以第一天实还好，就是划划水，优化一下程序，晚上还出去看了下『姜子牙』。
第二天的时候，又划了一天的，等到晚上的时候，同事终于把他那边的迁移搞定，我终于可以开迁移。
可是这个时候甲方爸爸竟然开始加各种需求了，没办法，只能满足他们，临时再开始修改程序，后再加上监控，一直搞到了 12 点。

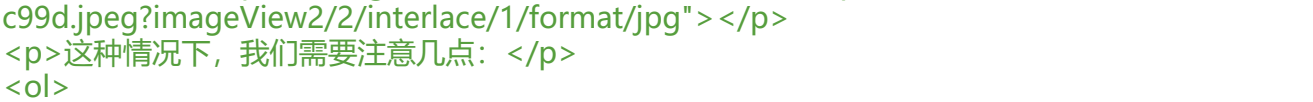
搞完这下，本来以为没啥幺蛾子了，长假只要每天看看迁移进度就可以了。
哎，没想到设计的时候忘记考虑性能的问题，等到数据量增长到千万的时候，查询速度奇慢，无只好优化程序，想办法让查询加快，最后一直搞到了凌晨四点 ┬┬﹏┬┬
真的是设计不规范，加班两行泪啊~后面有机会跟大家复盘一下这次大数据量迁移。
<h2 id="前言">前言</h2>
好了，回归到今天的主题，今天分享一下支付系统中异常一些处理方式。
其实这些处理方式并不只是局限于支付系统，也可以适用于其他系统，大家可以借鉴，应用到自系统中，提高自己系统的健壮性。
异常是系统运行不可避免会发生的问题，如果一切都正常，我们的系统设计将会相当简单。
但是可惜没有人能做到这一点，所以为了处理异常可能导致的问题，我们不得不需要加上很多额的设计，用来应对这些异常。
可以说系统设计中，异常处理需要我们着重思考，将会占据我们大部分的精力。
下面我们先来看下支付系统中最常见的异常：掉单
<h2 id="掉单异常">掉单异常</h2>
一个最常见的支付平台架构关系如下所示：

<blockquote>
上图我们是站在第三方支付公司支付角度，如果是自己公司的内部支付系统，那么外部商户这一其实就是公司内部一些系统，比如说订单系统，而外部支付渠道其实就是第三方支付公司
</blockquote>
我们以携程为例，在其上面发起一笔订单支付，将会经过三个系统：

携程创建订单，向第三方支付公司发起支付请求
第三方支付公司创建订单，并向工行发起支付请求
工行完成扣款操作，返回第三方支付公司
第三方支付完成订单更新并返回携程
携程变更订单状态

上面的流程，简单如下图所示：

在这个过程就可能会碰到，用户工行卡已经扣款，但是携程订单却还是待支付，我们通常将这种况称为掉单。
上述掉单的场景，多数是因为 ③、⑤ 环节信息丢失导致，这种掉单我们将称为外部掉单。
还有一种极少数的情况，收到 ③、⑤ 环节返回信息，但是在 ④⑥ 环节内部系统更新订单状态失败，从而导致丢失支付成功的信息，这类掉单由于是内部题，我们通常将其称之为内部掉单。

<h2 id="外部掉单">外部掉单</h2>
外部掉单是因为没有收到对端返回信息，这种情况极有可能是网络问题，也有可能对端处理逻辑慢，导致我方请求超时，直接断开了网络请求。
<h3 id="增加超时时间">增加超时时间</h3>
对于这种情况，第一个最简单的解决办法，适当的增加超时时间。
不过这里需要注意了，在我们增加网络超时时间之后，我们可能还需要调整整个链路的超时时间不然有可能导致整个链路内部差事从而引起内部掉单。
<blockquote>
画外音：对接外部渠道，一定要设置网络连接超时时间与读取超时时间。
</blockquote>
<h3 id="接收异步通知">接收异步通知</h3>
第二个办法，接收渠道异步回执通知信息。
一般来说，现在支付渠道接口我们都可以上送一个异步回调地址，当渠道端处理成功，将会把成信息通知到这个回调地址上。
这种情况下，我们只需要接收通知信息，然后解析，再更新内部订单状态。


这种情况下，我们需要注意几点：

对于异步请求信息，一定需要对通知内容进行签名验证，并校验返回的订单金额是否与商户侧的单金额一致，防止数据泄漏导致出现“假通知”，造成资金损失。
异步通知将会发送多次，所以异步通知处理需要幂等。

<h3 id="掉单查询">掉单查询</h3>
有的渠道可能没有提供异步通知的功能，只提供了订单查询的接口，这种情况下，我们只能使用三种解决办法，定时掉单查询。
我们可以将这类超时未知的订单的单独保存到掉单表，然后定时向渠道端查询订单的状态。
若查询成功或者明确失败（比如订单不存在等），可以更新订单状态，并且删除掉单表记录。</
>
若查询依旧未知，这时我们需要等待下次查询的结果。


这里我们需要注意了，有些情况下，有可能无法查询返回订单的状态，所以我们需要设置订单查的最大次数，防止无限查询浪费性能。
<h3 id="对账">对账</h3>
最后，极少数的情况下，订单查询与异步通知都无法获取的支付结果，这就还剩下最后一种兜底解决办法，对账。
如果第二天渠道端给的对账文件有这一笔支付结果，那么我们可以根据这个记录更新直接更新我内部支付记录。
之前小黑哥写过一篇对账文章，感兴趣的可以再看一下：<a href="https://ld246.com/forward
goto=https%3A%2F%2Fmp.weixin.qq.com%2Fs%2F47s0YdRM6u1JNngNe6yghg" target="_bl
nk" rel="nofollow ugc">聊聊对账系统的设计方案
<blockquote>
画外音：稳妥一点，可以先发起查询，然后根据查询结果更新订单记录。
不过有些极端情况，查询无法获取结果，那么直接更新内部记录即可。
</blockquote>
那如果第二天也没有这笔记录的结果，这种情况下，我们可以认为这笔是失败的。如果用户被扣，渠道端内部将会发起退款，将支付金额返回给用户。所以这种情况可以无需处理。
<h2 id="内部掉单异常">内部掉单异常</h2>
<h3 id="支付公司内部订单关系">支付公司内部订单关系</h3>
接下来我们讲下内部掉单异常，首先我们来看下为什么会发生内部掉单的异常，这其实跟我们系

架构有关。
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/10/solofetchupload6665525560621205337-db83a605.jpeg?imageView2/2/interlace/1/format/jpg">
如上图随所示，第三方支付公司内部表通常为支付订单与渠道订单这样一种 1 比 N 的关系。

支付订单保存着外部商户系统的订单号，代表第三方支付公司内部订单与外部商户的订单的关系
而渠道订单代表着第三方支付公司与外部渠道的关系，其实对于外部渠道系统来讲，第三方支付司就是一个外部商户。
为什么需要设计这种关系那？而不是使用下面这种 1 对 1 关系的那？
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/10/solofetchupload7032470071746054623-cfd8e28b.jpeg?imageView2/2/interface/1/format/jpg">
如果我们使用上图 1 对 1 的订单关系，如果第一次支付支付失败，外部商户可能会再次使用相订单号对第三方支付公司发起支付。
这时如果第三方支付公司也拿相同的内部订单去请求外部渠道系统，有可能外部渠道系统并不支同一订单号再次请求。
那其实我们也有其他办法，生成一个新的内部单号，更新原有支付订单上内部记录，然后去请求部渠道系统。但是这样的话就会丢失上次支付失败记录，这就不利于我们做一些事后统计了。
那其实第三方支付公司也可以不支持相同的订单号再次发起请求，但是这样的话，就需要外部商重新生成的新的订单号。
这样的话，第三方支付公司是系统是简单了，全部复杂度都交给了外部商户。
但是现实的情况，很多外部商户并不是那么容易更换生成新的订单号，所以一般第三方支付公司需要支持同一外部商户订单号在未成功的情况下，支持重复支付。
在这种情况下，就需要我们上面的 1:N 的订单关系图了。
<h3 id="内部掉单异常的原因">内部掉单异常的原因</h3>
当我们收到外部渠道系统的成功的返回信息，成功更新了渠道订单表的记录。但是由于渠道订单与支付订单表可能不是同一个数据库，也有可能两者并不在同一个应用中，这就有可能导致更新支付单表的更新失败。
<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile.com/file/2020/10/solofetchupload7786765336782894392-a063a440.jpeg?imageView2/2/interface/1/format/jpg">
由于支付订单是表保存着外部商户订单与内部订单关系，支付订单未成功，所以外部商户也无法询得到成功的支付结果。
此时渠道订单表已经成功，所以上面外部掉单的方法并不适用内部掉单。
<h3 id="内部掉单异常解决办法">内部掉单异常解决办法</h3>
第一种解决办法，分布式事务。
内部掉单异常，说白就是因为支付订单表与渠道订单表无法使用数据库事务保证两者同时更新成或失败。
那么这种情况下，我们其实就需要使用分布式事务了。
不过我们没有采用这种分布式事务，一是因为之前开发的时候市面上并没有开源成熟分布式事务架，第二自己自己开发难度又很大。
所以对于分布式事务这一块，并没有什么使用经验。如果有使用分布式事务解决这类的问题同学留言去可以评论一下。
第二种解决办法，异步补偿更新。
当发生内部掉单的情况，即更新支付订单失败等情况，可以将这里支付订单保存到一张内部掉单。
但是这里可能会有一个问题，我们无法保证保存到内部掉单表这一步骤也一定成功。
所以说，我们还需要定时查询，查询一段时间内支付订单未成功，而渠道订单表已成功的支付订记录，然后也将其插入到内部掉单表。
另一个系统应用，只需要定时扫描内部掉单表，将支付订单成功，然后再删除内部掉单记录即可
这里需要注意了，当支付订单表数据量很大之后，定时查询可能会慢，为了防止影响主库，所以

类查询可以在备库进行。
<h2 id="总结">总结</h2>
今天主要介绍了支付系统中的掉单异常，这类异常往往会导致用户实际已经被扣钱，但是商户订还是等待支付的情况。
这个异常如果没有很好处理，将会导致客户用户体验很不好，还有可能收到客户的投诉。
掉单的异常，通常可以外部系统与内部系统。而大部分的掉单都是因为外部系统导致，我们可以加超时时间，掉单查询，以及接受异步通知解决 99% 的问题，剩下 1% 的掉单只能通过次日的对账兜底。
内部系统导致掉单异常是典型的分布式环境数据一致性的问题，这类问题我们可以不需要追求强致性，只要保证最终一致性即可。我们可以使用分布式事务解决这类问题，也可以定时扫描状态不一的订单，然后在做批量更新。
最后，这次只是介绍支付系统中一类掉单异常，下一篇文章中，再给大家介绍一下支付系统的其异常，敬请期待！
<h2 id="参考资料">参考资料</h2>

知乎 @ 天顺 谈谈异常（一）
