04 | 如何利用事务消息实现分布式事务?

李玥 · 消息队列高手课

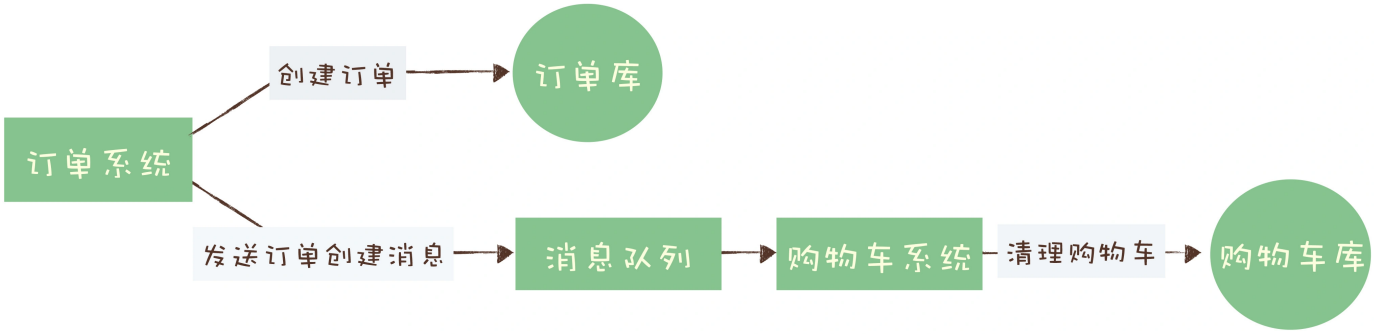
你好，我是李玥，今天我们来聊一聊消息和事务。

一说起事务，你可能自然会联想到数据库。的确，我们日常使用事务的场景，绝大部分都是在操作数据库的时候。像 MySQL、Oracle 这些主流的关系型数据库，也都提供了完整的事务实现。那消息队列为什么也需要事务呢?

其实很多场景下，我们“发消息”这个过程，目的往往是通知另外一个系统或者模块去更新数据，**消息队列中的“事务”，主要解决的是消息生产者和消息消费者的数据一致性问题。**

依然拿我们熟悉的电商来举个例子。一般来说，用户在电商 APP 上购物时，先把商品加到购物车里，然后几件商品一起下单，最后支付，完成购物流程，就可以愉快地等待收货了。

这个过程中有一个需要用到消息队列的步骤，订单系统创建订单后，发消息给购物车系统，将已下单的商品从购物车中删除。因为从购物车删除已下单商品这个步骤，并不是用户下单支付这个主要流程中必需的步骤，使用消息队列来异步清理购物车是更加合理的设计。



对于订单系统来说，它创建订单的过程中实际上执行了 2 个步骤的操作：

1. 在订单库中插入一条订单数据，创建订单；
2. 发消息给消息队列，消息的内容就是刚刚创建的订单。

购物车系统订阅相应的主题，接收订单创建的消息，然后清理购物车，在购物车中删除订单中的商品。

在分布式系统中，上面提到的这些步骤，任何一个步骤都有可能失败，如果不做任何处理，那就有可能出现订单数据与购物车数据不一致的情况，比如说：

创建了订单，没有清理购物车；

订单没创建成功，购物车里面的商品却被清掉了。

那我们需要解决的问题可以总结为：在上述任意步骤都有可能失败的情况下，还要保证订单库和购物车库这两个库的数据一致性。

对于购物车系统收到订单创建成功消息清理购物车这个操作来说，失败的处理比较简单，只要成功执行购物车清理后再提交消费确认即可，如果失败，由于没有提交消费确认，消息队列会自动重试。

问题的关键点集中在订单系统，创建订单和发送消息这两个步骤要么都操作成功，要么都操作失败，不允许一个成功而另一个失败的情况出现。

这就是事务需要解决的问题。

什么是分布式事务?

那什么是事务呢？如果我们需要对若干数据进行更新操作，为了保证这些数据的完整性和一致性，我们希望这些更新操作**要么都成功，要么都失败。**至于更新的数据，不只局限于数据库中的数据，可以是磁盘上的一个文件，也可以是远端的一个服务，或者以其他形式存储的数据。

这就是通常我们理解的事务。其实这段对事务的描述不是太准确也不完整，但是，它更易于理解，大体上也是正确的。所以我还是倾向于这样来讲“事务”这个比较抽象的概念。

一个严格意义的事务实现，应该具有 4 个属性：原子性、一致性、隔离性、持久性。这四个属性通常称为 ACID 特性。

原子性，是指一个事务操作不可分割，要么成功，要么失败，不能有一半成功一半失败的情况。

一致性，是指这些数据在事务执行完成这个时间点之前，读到的一定是更新前的数据，之后读到的一定是更新后的数据，不应该存在一个时刻，让用户读到更新过程中的数据。

隔离性，是指一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对正在进行的其他事务是隔离的，并发执行的各个事务之间不能互相干扰，这个有点儿像我们打网游中的副本，我们在副本中打的怪和掉的装备，与其他副本没有任何关联也不会互相影响。

持久性，是指一个事务一旦完成提交，后续的其他操作和故障都不会对事务的结果产生任何影响。

大部分传统的单体关系型数据库都完整的实现了 ACID，但是，对于分布式系统来说，严格的实现 ACID 这四个特性几乎是不可能的，或者说实现的代价太大，大到我们无法接受。

分布式事务就是要在分布式系统中的实现事务。在分布式系统中，在保证可用性和不严重牺牲性能的前提下，光是要实现数据的一致性就已经非常困难了，所以出现了很多“残血版”的一致性，比如顺序一致性、最终一致性等等。

显然实现严格的分布式事务是更加不可能完成的任务。所以，目前大家所说的分布式事务，更多情况下，是在分布式系统中事务的不完整实现。在不同的应用场景中，有不同的实现，目的都是通过一些妥协来解决实际问题。

在实际应用中，比较常见的分布式事务实现有 2PC（Two-phase Commit，也叫二阶段提交）、TCC(Try-Confirm-Cancel) 和事务消息。每一种实现都有其特定的使用场景，也有各自的问题，都不是完美的解决方案。

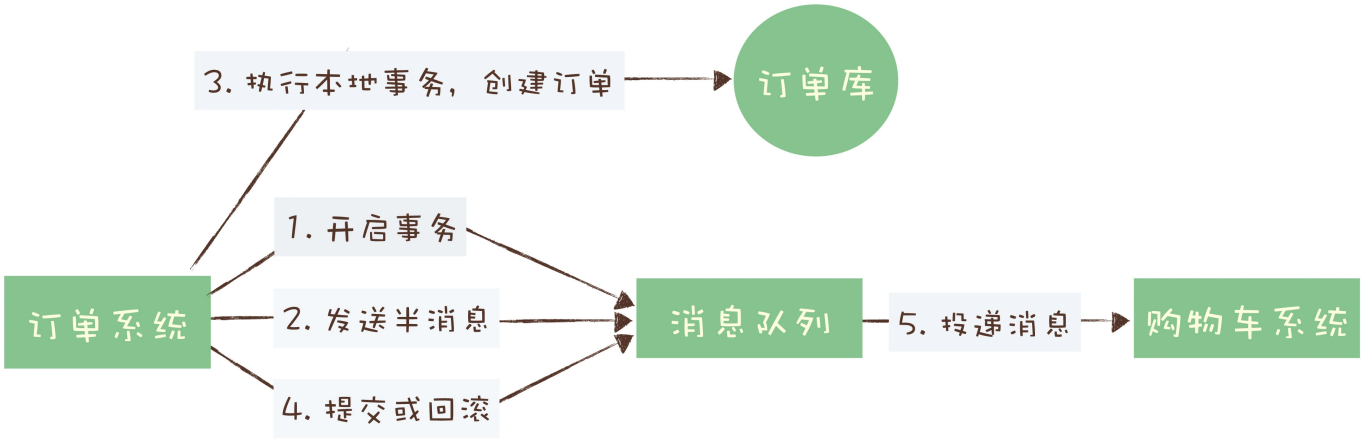
事务消息适用的场景主要是那些需要异步更新数据，并且对数据实时性要求不太高的场景。比如我们在开始时提到的那个例子，在创建订单后，如果出现短暂的几秒，购物车里的商品没有被及时清空，也不是完全不可接受的，只要最终购物车的数据和订单数据保持一致就可以了。

2PC 和 TCC 不是我们本次课程讨论的内容，就不展开讲了，感兴趣的同学可以自行学习。

消息队列是如何实现分布式事务的?

事务消息需要消息队列提供相应的功能才能实现，Kafka 和 RocketMQ 都提供了事务相关功能。

回到订单和购物车这个例子，我们一起来看下如何用消息队列来实现分布式事务。



首先，订单系统在消息队列上开启一个事务。然后订单系统给消息服务器发送一个“半消息”，这个半消息不是说消息内容不完整，它包含的内容就是完整的消息内容，半消息和普通消息的唯一区别是，在事务提交之前，对于消费者来说，这个消息是不可见的。

半消息发送成功后，订单系统就可以执行本地事务了，在订单库中创建一条订单记录，并提交订单库的数据库事务。然后根据本地事务的执行结果决定提交或者回滚事务消息。如果订单创建成功，那就提交事务消息，购物车系统就可以消费到这条消息继续后续的流程。如果订单创建失败，那就回滚事务消息，购物车系统就不会收到这条消息。这样就基本实现了“要么都成功，要么都失败”的一致性要求。

如果你足够细心，可能已经发现了，这个实现过程中，有一个问题是没有解决的。如果在第四步提交事务消息时失败了怎么办？对于这个问题，Kafka 和 RocketMQ 给出了 2 种不同的解决方案。

Kafka 的解决方案比较简单粗暴，直接抛出异常，让用户自行处理。我们可以在业务代码中反复重试提交，直到提交成功，或者删除之前创建的订单进行补偿。RocketMQ 则给出了另外一种解决方案。

RocketMQ 中的分布式事务实现

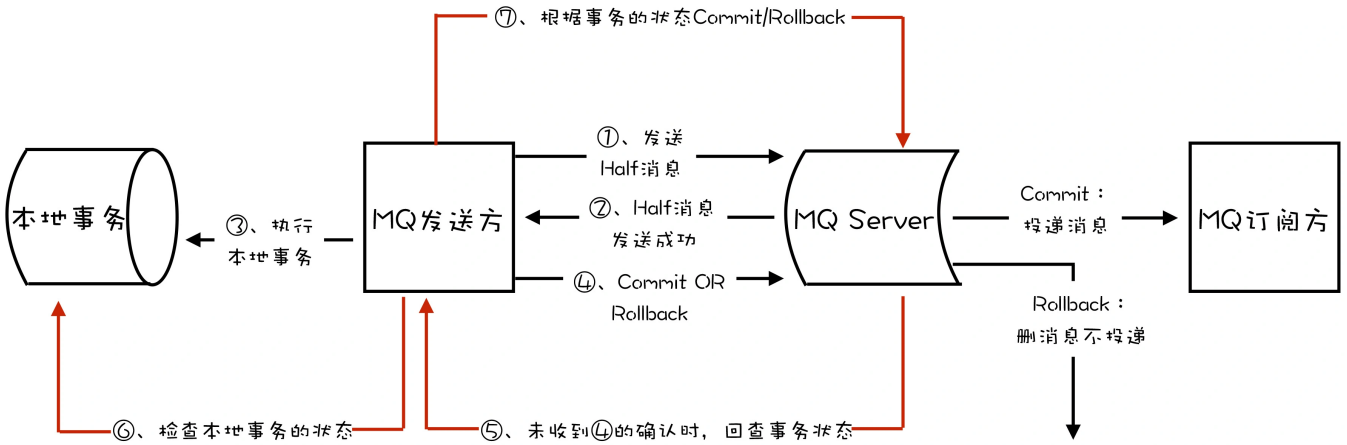
在 RocketMQ 中的事务实现中，增加了事务反查的机制来解决事务消息提交失败的问题。如果 Producer 也就是订单系统，在提交或者回滚事务消息时发生网络异常，RocketMQ 的 Broker 没有收到提交或者回滚的请求，Broker 会定期去 Producer 上反查这个事务对应的本地事务的状态，然后根据反查结果决定提交或者回滚这个事务。

为了支撑这个事务反查机制，我们的业务代码需要实现一个反查本地事务状态的接口，告知 RocketMQ 本地事务是成功还是失败。

在我们这个例子中，反查本地事务的逻辑也很简单，我们只要根据消息中的订单 ID，在订单库中查询这个订单是否存在即可，如果订单存在则返回成功，否则返回失败。RocketMQ 会自动根据事务反查的结果提交或者回滚事务消息。

这个反查本地事务的实现，并不依赖消息的发送方，也就是订单服务的某个实例节点上的任何数据。这种情况下，即使是发送事务消息的那个订单服务节点宕机了，RocketMQ 依然可以通过其他订单服务的节点来执行反查，确保事务的完整性。

综合上面讲的通用事务消息的实现和 RocketMQ 的事务反查机制，使用 RocketMQ 事务消息功能实现分布式事务的流程如下图：



小结

我们通过一个订单购物车的例子，学习了事务的 ACID 四个特性，以及如何使用消息队列来实现分布式事务。

然后我们给出了现有的几种分布式事务的解决方案，包括事务消息，但是这几种方案都不能解决分布式系统中的所有问题，每一种方案都有局限性和特定的适用场景。

最后，我们一起学习了 RocketMQ 的事务反查机制，这种机制通过定期反查事务状态，来补偿提交事务消息可能出现的通信失败。在 Kafka 的事务功能中，并没有类似的反查机制，需要用户自行去解决这个问题。

但是，这不代表 RocketMQ 的事务功能比 Kafka 更好，只能说在我们这个例子的场景下，更适合使用 RocketMQ。Kafka 对于事务的定义、实现和适用场景，和 RocketMQ 有比较大的差异，后面的课程中，我们会专门讲到 Kafka 的事务的实现原理。

思考题

课后，我建议你最好能通过写代码的方式，把我们这节课中的订单购物车的例子，用 RocketMQ 完整地实现出来。然后再思考一下，RocketMQ 的这种事务消息是不是完整地实

现了事务的 ACID 四个特性？如果不是，哪些特性没有实现？欢迎在留言区与我分享讨论。

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。

精选留言 (184)

一步
2019-07-30

对于上面订单的例子，为什么不等待订单创建成功再向消息队列发送订单数据呢？这样串行的话，对性能影响应该也不大，也不用考虑订单创建失败而发送消息的情况了。

作者回复: 考虑这样一种情况：订单创建成功了，还没来得及发消息，这个节点突然断电了。

共 44 条评论>

200

微微一笑
2019-07-30

老师您好，下面是我对思考题的一些看法：
A:本地事物的操作1，与往消息队列中生产消息的操作2，是两个分离的操作，不符合对原子性的定义；
C:由于操作消息队列属于异步操作，在数据一致性上，只能保证数据的最终一致性。若对于时效性要求很高的系统来说，事物消息不是数据一致的；但对于时效性要求不高的系统来说，他就是数据一致的。我认为，用不同的业务视角来看问题，会有不同的答案；
I：隔离性上，由于事物消息是分两步操作的，本地事物提交后，别的事物消息就已经可以看到提交的消息了。所以，不符合隔离性的定义；
D：持久性上，rocketMq上支持事物的反查机制，但我不太清楚“半消息”是存储在磁盘中，还是内存里。若存储在磁盘中，那就支持持久性，即使事物消息提交后，发生服务突然宕机也不受影响；若存储在内存中，则无法保证持久性。
以上是我的理解，望老师指点~

作者回复: 这个总结非常到位，给你点赞！

共 11 条评论>

136

ly
2019-07-30

实现订单下单场景：
1. 首先通过producer.sendMessageInTransaction()方法发送一个半消息给MQ.
2. 此时会在TransactionListener中的executeLocalTransaction()方法阻塞，然后在这个方法里面进行订单创建并提交本地事务，如果commit成功，则返回COMMIT状态，否则是ROLLBACK状态，如果正常返回COMMIT或者ROLLBACK的话，不会存在第3步的反查情况。
3. 如果上面的本地事务提交成功以后，此节点突然断电，那么checkLocalTransaction()反查方法就会在某个时候被MQ调用，此方法会根据消息中的订单号去数据库确认订单是否存在，存在就返回COMMIT状态，否则是ROLLBACK状态。
4. 购物车在另外一个项目中，反正只要收到MQ的消息就将本次订单的商品从购物车中删除即可。

以上是通过代码的进行步骤写的，老师看有没有什么问题。

作者回复: 非常好，完全正确！

共 13 条评论>

77

Calix
2019-08-06

这个半消息，和生活中的“交定金”有点类似。

作者回复: 其实是交全款，不发货。

共 4 条评论>

49

君莫笑
2019-08-26

老师，我回头重新看的时候看到这一章有一点疑问，消息队列的手动确认模式是可以保证分布式事务的最终一致性，那么如果生产者在处理完自己的业务之后将消息放入消息队列中（通过生产者确认方式可以确保消息送达Broker），然后消费者消费这个消息的时候出了问题，假设是消息体本身的原因导致消费该消息一定会抛出异常，这种情况下怎么通知生产者回滚该消息所处理的业务数据呢?

作者回复: 这种情况下是没有办法回滚的，也不应该回滚。
因为对于消息队列来说，它的一个重要功能就是解耦。
消费者的任何行为，不应该影响生产者。

对于你说的“坏消息”，反复消费都不能成功，有的MQ会把这种消息放到一个单独的特殊队列中，等着后续人工处理，避免卡死队列。

共 9 条评论>

36

oscarwin
2019-07-31

先开启本地事务，然后创建订单，订单创建成功后再发消息，根据发消息是否成功来决定提交还是回滚本地事务。这样不需要事务消息也能解决这个场景的问题了？还是说我考虑的不够全面。

作者回复: 如果本地事务提交失败，已经发出去的消息是无法撤回的，会导致数据不一致。

共 16 条评论>

34

Geek
2019-08-21

老师，有几个问题没有太理解，可以解答一下么？
1.kafka在commit/rollback的时候如果发送失败了就会抛出异常，会不会存在已经发送成功了但是超时了的情况呢，这个时候broker已经收到数据了。但是上游业务却回滚了
2.RocketMq反查时有没有可能本地事务还没提交呢，导致broker取消了事务造成了不一致
3.RocketMq在反查时如果订单服务异常了，导致broker取消了会不会导致事务造成了不一致
谢谢老师

作者回复: 第一个问题，我们后面还有专门的一节课来讲事务是如何实现的，这里面会有你想要的答案。

第二第三个问题，RocketMQ给出的解决方案是，反查的结果返回的状态中，不仅有成功和失败，还有一个“不确定”的状态，意思就是“我现在不知道本地事务是不是成功了，将来它可能会成功，也可能会失败”，像你提的这两种情况，在实现反查接口的时候，都应该返回不确定的状态，RocketMQ在收到这个状态后，会定时多次进行反查，直到得到成功、失败的状态或者事务超时才结束。

共 2 条评论>

30

linqw
2019-07-30

使用rocketmq实现分布式事务的理解和疑问，老师有空帮忙解答下哦
1、rocketmq实现分布式事务，使用的是两阶段提交，和mysql写redo log和binlog日志的两阶段提交类似，以上面订单的为例，提交订单消息到mq中，等待mq回复ack，消息提交成功，但是此时的消息对消费组不可见，即half消息，此阶段像mysql的引擎层写redo log的prepare阶段，执行本地事务，执行本地事务成功，此阶段像mysql的service层写binlog的阶段，写binlog成功，最后提交或者回滚队列事务，rocketmq为了防止commit和rollback超时或者失败，采取回查的补偿机制，回查次数默认15次（感觉这个会不会导致服务超时了），超过会rollback，有点像mysql宕机重启根据redo log中的xid找binlog的xid事务，如果binlog日志也已经写成功，mysql这个事务也会提交，因为redo log和binlog这个事务都写完整。
2、消息对消费者不可见，将其消息的主题topic和队列id修改为half topic，原先的主题和队列id也做为消息的属性，如果事务提交或者回滚会将其消息的队列改为原先的队列。rocketMq开启任务，从half topic中获取消息，调用其中的生产者的监听进行回查是否提交回滚。
3、rocketmq采用commitlog存放消息，消费者使用consumeQueue二级索引从commitlog获取消息实体内容，不太理解Index File：索引文件？回查借助OP topic进行获取到Half消息进行后续的回查操作，感觉整体流程还是没有串通，老师能否帮忙解答下么？

作者回复: indexFile的作用就是给commitlog做的索引，提升读取消息时的查询效率。

另外，关于事务的实现流程，总结的很到位，你还有哪些具体的问题不清楚，可以继续留言提出来。

共 9 条评论> 28

Yize Li
2019-09-22

看了之前的一个留言 认为本地数据库和消息系统是两个系统所以违反了原子性 我是有些疑惑的。
我认为 原子性破坏与否取决于是否存在数据库中订单成功但是在购物车中商品没有取消的情况 通过rocketmq的半消息模式是可以保证该情况不出现。所以原子性没有破坏 但是由于消息系统的异步性 导致我们可以观察到事物执行过程中或回滚中的中间状态 这意味着强一致性被破坏 只剩下了最终一致性

作者回复: 是这样的。

共 5 条评论> 25

yan
2019-09-26

如果订单ID是要创建完订单才会有的，那消息中就没有订单ID，那反查本地事务要根据什么查?

作者回复: 所以几乎所有的类似系统都会事先生成订单ID，而不是在插入数据库的时候才生成。

共 3 条评论>

17

DC
2019-08-01

rocket mq 事务消息参考文档：https://rocketmq.apache.org/docs/transaction-example/

作者回复: 👍👍👍

17

朱海昆
2019-07-30

老师，如果消息队列不支持半消息，是否有其他的解决方案？我了解到一种解决方法是利用数据库的事务消息表的方案。把消息信息的快照和对业务数据的操作作为数据库事务操作数据库，操作成功后从数据库读取消息信息发送给broker，收到发送成功的回执后删除数据库中的消息快照。我个人觉得这种方案在不支持半消息的队列方案里也是一种选择，不知道您觉得这种实现方案有没有什么问题。

作者回复: 如果有一个生产者和消费者都可以访问的，并且性能还不错数据库，肯定是使用这个数据库来实现事务比较好。

大部分事务消息使用的场景是，没有这样的数据库的。或者由于设计、安全或者网络原因，生产者消费者不能共享数据库，或者是数据库的性能达不到要求。

共 4 条评论>

10

史双龙
2019-07-30

我jiao着吧，如果先创建订单，当前服务由于不可抗拒因素不能正常工作了，没有给购物车系统发送消息，这种情况加就会出现 订单已经创建并且购物车没有清空的情况。然鹅发送半消息这种情况，可以通过定期查询事务的状态然后根据然后具体的业务回滚操作或者重新发送消息（保持业务的幂等性）。技术渣理解的有可能不到位 谅解

作者回复: 到位。

共 2 条评论>

9

佳明
2019-08-01

消费端做幂等处理来保障消息不会重复消费:1. 可以采用状态机的方式。2.消息数据唯一键+redis setnx来保障。3.本地消息表，要确保插入本地消息表和执行消息消费业务在同一事务里。

8

Cha
2019-12-09

使用事务消息的话，如果此时mq服务挂了，是否订单就没办法下单了？

作者回复: 一般MQ的Broker都有高可用方案，不会出现一个节点宕机就无法发消息的问题。

所以不用担心这个问题。

7

长期规划
2019-09-18

老师，kafka对提交消息队列事务失败时的处理方法，我感觉和不用事务没什么区别吧？如果不用事务，先完成数据库事务，再发消息，消失失败也是用户自己处理，比如重试。这跟kafka用分布式事务有什么区别吗

作者回复: 把数据库事务放到Kafka事务中的好处是，在发消息和执行SQL阶段，无论哪个操作失败了，都可以自动回滚。
只有“提交Kafka事务失败了”这个情况才需要手动处理。

而要是不用Kafka事务，先执行数据库事务，再发普通消息，如果发消息失败就要手动回滚数据库。

你要明白，提交事务只是设置一个状态，失败的概率要远远小于发消息的概率。

虽然理论上这两种情况失败都需手动处理，但是失败的概率差别很大，所以实际上还是非常有用的。

共 5 条评论>

7

多襄丸

2019-09-18

老师 那要是写主库 查从库 rocketmq 去反查的时候查的是从库， 而从库的数据还没有被同步进去。 会不会有问题哇?

作者回复: 不会，你要注意，写反查逻辑的时候，如果查不到，不要反悔失败，而应该返回UNKNOWN

共 7 条评论>

7

晴天

2019-08-06

你好，我想问一下，比如订单创建这些都成功了，也发送了消息，但是清空购物车的时候失败了，这个时候我看到做法就是消息重试，但是会不会有一种场景，下游决定上游需要回滚，也就是需要回滚订单创建的事务，这种可能存在吗？怎么处理呢

作者回复: 这个问题消息队列解决不了，可能需要用其他的分布式事务解决方案。

共 3 条评论>

7

陈天柱

2019-10-11

老师，您好。在这个分布式事务例子里，是涉及两个分布式服务，假如遇到跨三个或者三个以上的分布式该如何解决？假设通过服务划分或者业务改造，都免不了需要跨三个以上的分布式服务。

作者回复: 这种你就需要使用一些通用的分布式事务算法的实现了，比如二阶段提交。

共 3 条评论>

6

YAO

2019-12-17

rabbitmq 支持事务吗

作者回复: 支持的，但是每种MQ支持的“事务”都不太一样，使用前你需要看一下文档是不是你需要

的那种“事务”


4