06 | 分布式事务：All or nothing

2019-10-04 聂鹏程

分布式技术原理与算法解析 进入课程 >

讲述：聂鹏程
时长 21:25　大小 19.62M

你好，我是聂鹏程。今天，我来继续带你打卡分布式核心技术。

对于网上购物的每一笔订单来说，电商平台一般都会有两个核心步骤：一是订单业务采取下订单操作，二是库存业务采取减库存操作。

通常，这两个业务会运行在不同的机器上，甚至是运行在不同区域的机器上。针对同一笔订单，当且仅当订单操作和减库存操作一致时，才能保证交易的正确性。也就是说一笔订单，只有这两个操作都完成，才能算做处理成功，否则处理失败，充分体现了“All or nothing”的思想。

在分布式领域中，这个问题就是分布式事务问题。那么今天，我们就一起打卡分布式事务吧。

什么是分布式事务?

要想理解分布式事务，我们首先来看一下什么是事务。

事务，其实是包含一系列操作的、一个有边界的工作序列，有明确的开始和结束标志，且要么被完全执行，要么完全失败，即 all or nothing。通常情况下，我们所说的事务指的都是本地事务，也就是在单机上的事务。

而**分布式事务，就是在分布式系统中运行的事务，由多个本地事务组合而成。**在分布式场景下，对事务的处理操作可能来自不同的机器，甚至是来自不同的操作系统。文章开头提到的电商处理订单问题，就是典型的分布式事务。

要深入理解分布式事务，我们首先需要了解它的特征。分布式事务是多个事务的组合，那么事务的特征 ACID，也是分布式事务的基本特征，其中 ACID 具体含义如下：

原子性（Atomicity），即事务最终的状态只有两种，全部执行成功和全部不执行。若处理事务的任何一项操作不成功，就会导致整个事务失败。一旦操作失败，所有操作都会被取消（即回滚），使得事务仿佛没有被执行过一样。

一致性（Consistency），是指事务操作前和操作后，数据的完整性保持一致或满足完整性约束。比如，用户 A 和用户 B 在银行分别有 800 元和 600 元，总共 1400 元，用户 A 给用户 B 转账 200 元，分为两个步骤，从 A 的账户扣除 200 元和对 B 的账户增加 200 元；一致性就是要求上述步骤操作后，最后的结果是用户 A 还有 600 元，用户 B 有 800 元，总共 1400 元，而不会出现用户 A 扣除了 200 元，但用户 B 未增加的情况 (该情况，用户 A 和 B 均为 600 元，总共 1200 元)。

隔离性（Isolation），是指当系统内有多个事务并发执行时，多个事务不会相互干扰，即一个事务内部的操作及使用的数据，对其他并发事务是隔离的。

持久性（Durability），也被称为永久性，是指一个事务完成了，那么它对数据库所做的更新就被永久保存下来了。即使发生系统崩溃或宕机等故障，只要数据库能够重新被访问，那么一定能够将其恢复到事务完成时的状态。

分布式事务基本能够满足 ACID，其中的 C 是强一致性，也就是所有操作均执行成功，才提交最终结果，以保证数据一致性或完整性。但随着分布式系统规模不断扩大，复杂度急剧上升，达成强一致性所需时间周期较长，限定了复杂业务的处理。为了适应复杂业务，出现了 BASE 理论，该理论的一个关键点就是采用最终一致性代替强一致性。我会在“知识扩展”模块与你详细展开 BASE 理论这部分内容。

介绍完什么是分布式事务，以及事务的基本特征后，就进入“怎么做”的阶段啦。所以接下来，我们就看看如何实现分布式事务吧。

如何实现分布式事务?

实际上，分布式事务主要是解决在分布式环境下，组合事务的一致性问题。实现分布式事务有以下 3 种基本方法：

基于 XA 协议的二阶段提交协议方法；

三阶段提交协议方法；

基于消息的最终一致性方法。

其中，基于 XA 协议的二阶段提交协议方法和三阶段提交协议方法，采用了强一致性，遵从 ACID，基于消息的最终一致性方法，采用了最终一致性，遵从 BASE 理论。

基于 XA 协议的二阶段提交方法

XA 是一个分布式事务协议，规定了事务管理器和资源管理器接口。因此，XA 协议可以分为两部分，即事务管理器和本地资源管理器。

XA 实现分布式事务的原理，就类似于我在第 3 篇文章中与你介绍的集中式算法：事务管理器作为协调者，负责各个本地资源的提交和回滚；而资源管理器就是分布式事务的参与者，通常由数据库实现，比如 Oracle、DB2 等商业数据库都实现了 XA 接口。

基于 XA 协议的二阶段提交方法中，二阶段提交协议（The two-phase commit protocol，2PC），用于保证分布式系统中事务提交时的数据一致性，是 XA 在全局事务中用于协调多个资源的机制。

那么，**两阶段提交协议如何保证分布在不同节点上的分布式事务的一致性呢**？为了保证它们的一致性，我们需要引入一个协调者来管理所有的节点，并确保这些节点正确提交操作结果，若提交失败则放弃事务。接下来，我们看看两阶段提交协议的具体过程。

两阶段提交协议的执行过程，分为投票（voting）和提交（commit）两个阶段。

投票为第一阶段，协调者（Coordinator，即事务管理器）会向事务的参与者（Cohort，即本地资源管理器）发起执行操作的 CanCommit 请求，并等待参与者的响应。参与者接收到请求后，会执行请求中的事务操作，记录日志信息但不提交，待参与者执行成功，则向协调者发送“Yes”消息，表示同意操作；若不成功，则发送“No”消息，表示终止操作。

当所有的参与者都返回了操作结果（Yes 或 No 消息）后，**系统进入了提交阶段**。在提交阶段，协调者会根据所有参与者返回的信息向参与者发送 DoCommit 或 DoAbort 指令：

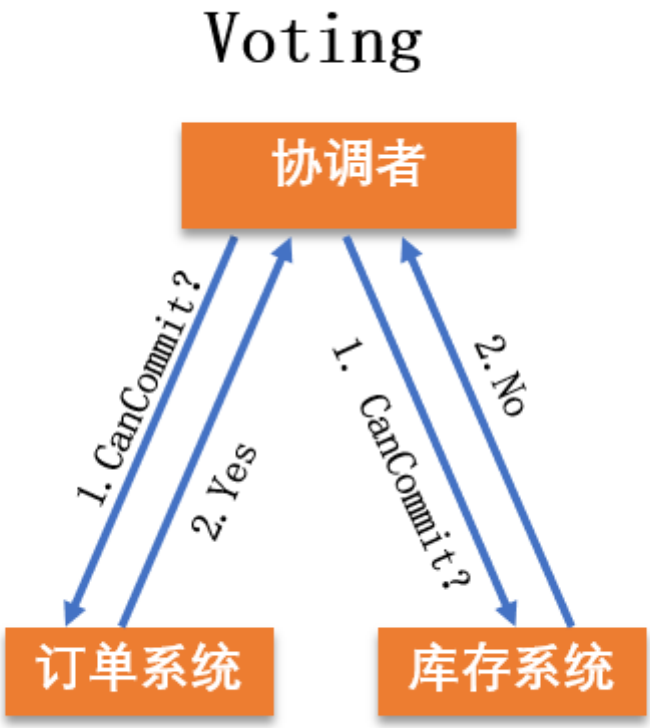
若协调者收到的都是“Yes”消息，则向参与者发送“DoCommit”消息，参与者会完成剩余的操作并释放资源，然后向协调者返回“HaveCommitted”消息；

如果协调者收到的消息中包含“No”消息，则向所有参与者发送“DoAbort”消息，此时发送“Yes”的参与者则会根据之前执行操作时的回滚日志对操作进行回滚，然后所有参与者会向协调者发送“HaveCommitted”消息；

协调者接收到“HaveCommitted”消息，就意味着整个事务结束了。

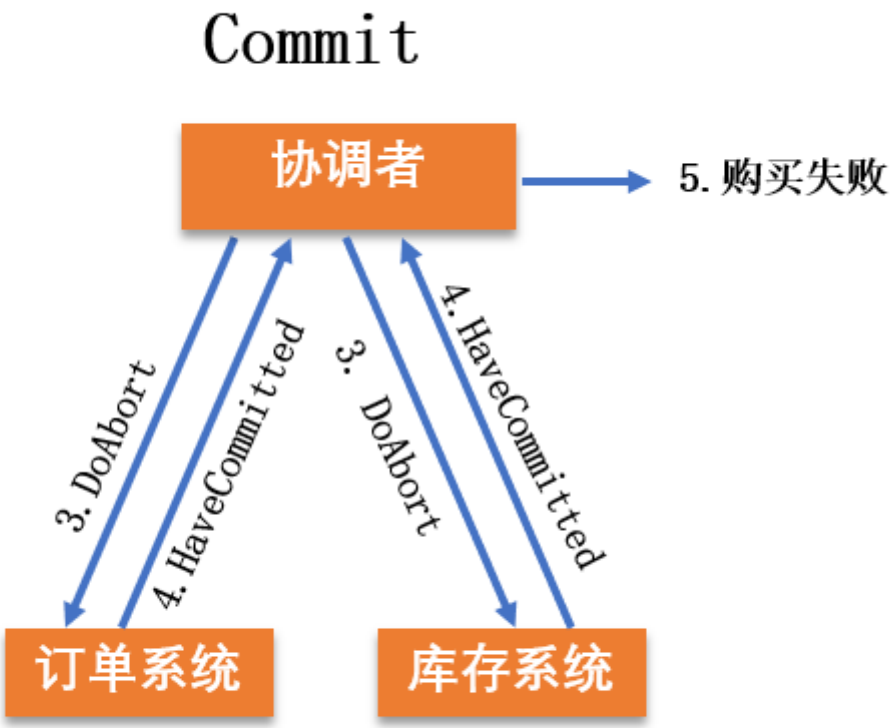
接下来，**我以用户 A 要在网上下单购买 100 件 T 恤为例，重点与你介绍下单操作和减库存操作这两个操作**，帮助你加深对二阶段提交协议的理解。

第一阶段：订单系统中将与用户 A 有关的订单数据库锁住，准备好增加一条关于用户 A 购买 100 件 T 恤的信息，并将同意消息“Yes”回复给协调者。而库存系统由于 T 恤库存不足，出货失败，因此向协调者回复了一个终止消息“No”。



第二阶段：由于库存系统操作不成功，因此，协调者就会向订单系统和库存系统发送“DoAbort”消息。订单系统接收到“DoAbort”消息后，将系统内的数据退回到没有用户 A 购买 100 件 T 恤的版本，并释放锁住的数据库资源。订单系统和库存系统完成操作后，向协调者发送“HaveCommitted”消息，表示完成了事务的撤销操作。

至此，用户 A 购买 100 件 T 恤这一事务已经结束，用户 A 购买失败。



由上述流程可以看出，**二阶段提交的算法思路可以概括为**：协调者下发请求事务操作，参与者将操作结果通知协调者，协调者根据所有参与者的反馈结果决定各参与者是要提交操作还是撤销操作。

虽然基于 XA 的二阶段提交算法基本满足了事务的 ACID 特性，但依然有些不足。

同步阻塞问题：二阶段提交算法在执行过程中，所有参与节点都是事务阻塞型的。也就是说，当本地资源管理器占有临界资源时，其他资源管理器如果要访问同一临界资源，会处于阻塞状态。

单点故障问题：基于 XA 的二阶段提交算法类似于集中式算法，一旦事务管理器发生故障，整个系统都处于停滞状态。尤其是在提交阶段，一旦事务管理器发生故障，资源管理器会由于等待管理器的消息，而一直锁定事务资源，导致整个系统被阻塞。

数据不一致问题：在提交阶段，当协调者向参与者发送 DoCommit 请求之后，如果发生了局部网络异常，或者在发送提交请求的过程中协调者发生了故障，就会导致只有一部分参与者接收到了提交请求并执行提交操作，但其他未接到提交请求的那部分参与者则无法执行事务提交。于是整个分布式系统便出现了数据不一致的问题。

三阶段提交方法

三阶段提交协议（Three-phase commit protocol，3PC），是对二阶段提交（2PC）的改进。为了解决两阶段提交的同步阻塞和数据不一致问题，**三阶段提交引入了超时机制和准备阶段**。

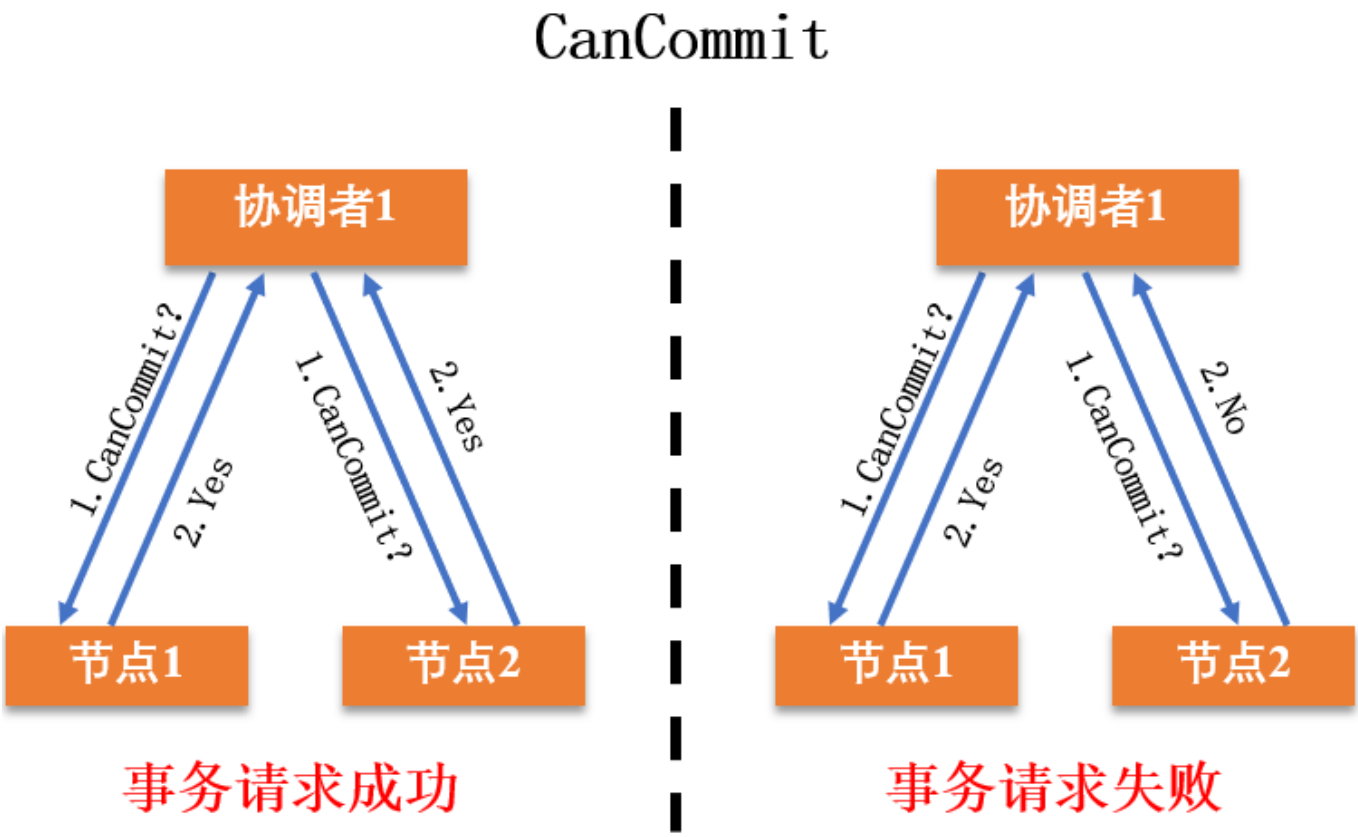
> 同时在协调者和参与者中引入超时机制。如果协调者或参与者在规定的时间内没有接收到来自其他节点的响应，就会根据当前的状态选择提交或者终止整个事务。
>
> 在第一阶段和第二阶段中间引入了一个准备阶段，也就是在提交阶段之前，加入了一个预提交阶段。在预提交阶段排除一些不一致的情况，保证在最后提交之前各参与节点的状态是一致的。

也就是说，除了引入超时机制之外，3PC 把 2PC 的提交阶段一分为二，这样三阶段提交协议就有 CanCommit、PreCommit、DoCommit 三个阶段。

第一，CanCommit 阶段。

CanCommit 阶段与 2PC 的投票阶段类似：协调者向参与者发送请求操作（CanCommit 请求），询问参与者是否可以执行事务提交操作，然后等待参与者的响应；参与者收到 CanCommit 请求之后，回复 Yes，表示可以顺利执行事务；否则回复 No。

CanCommit 阶段不同节点之间的事务请求成功和失败的流程，如下所示。



第二，PreCommit 阶段。

协调者根据参与者的回复情况，来决定是否可以进行 PreCommit 操作。

如果所有参与者回复的都是“Yes”，那么协调者就会执行事务的预执行：

发送预提交请求。协调者向参与者发送 PreCommit 请求，进入预提交阶段。

事务预提交。参与者接收到 PreCommit 请求后执行事务操作，并将 Undo 和 Redo 信息记录到事务日志中。

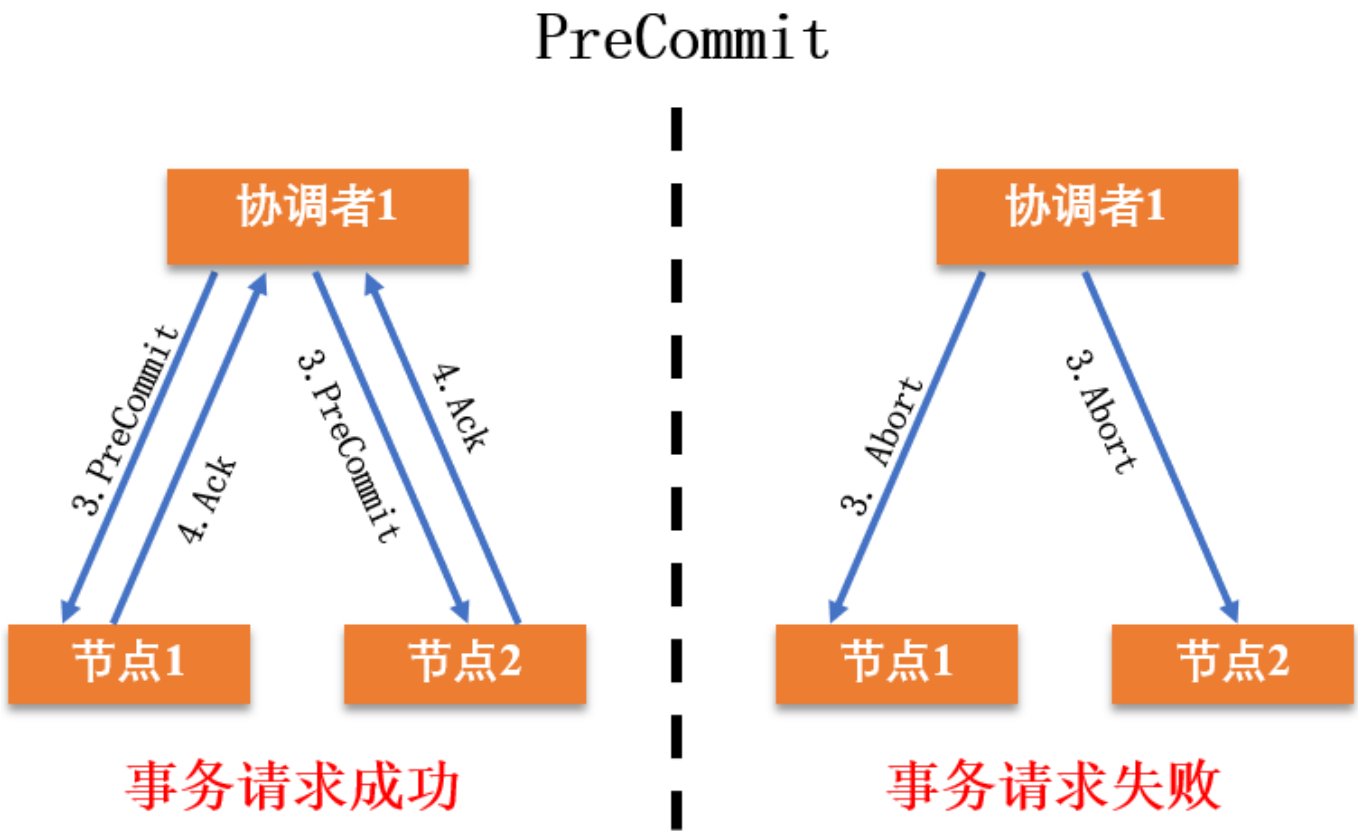
响应反馈。如果参与者成功执行了事务操作，则返回 ACK 响应，同时开始等待最终指令。

假如任何一个参与者向协调者发送了“No”消息，或者等待超时之后，协调者都没有收到参与者的响应，就执行中断事务的操作：

发送中断请求。协调者向所有参与者发送“Abort”消息。

终断事务。参与者收到“Abort”消息之后，或超时后仍未收到协调者的消息，执行事务的终断操作。

预执行阶段，不同节点上事务执行成功和失败的流程，如下所示。



第三，DoCommit 阶段。

DoCmmit 阶段进行真正的事务提交，根据 PreCommit 阶段协调者发送的消息，进入执行提交阶段或事务中断阶段。

执行提交阶段：

发送提交请求。协调者接收到所有参与者发送的 Ack 响应，从预提交状态进入到提交状态，并向所有参与者发送 DoCommit 消息。

事务提交。参与者接收到 DoCommit 消息之后，正式提交事务。完成事务提交之后，释放所有锁住的资源。

响应反馈。参与者提交完事务之后，向协调者发送 Ack 响应。

完成事务。协调者接收到所有参与者的 Ack 响应之后，完成事务。

事务中断阶段：

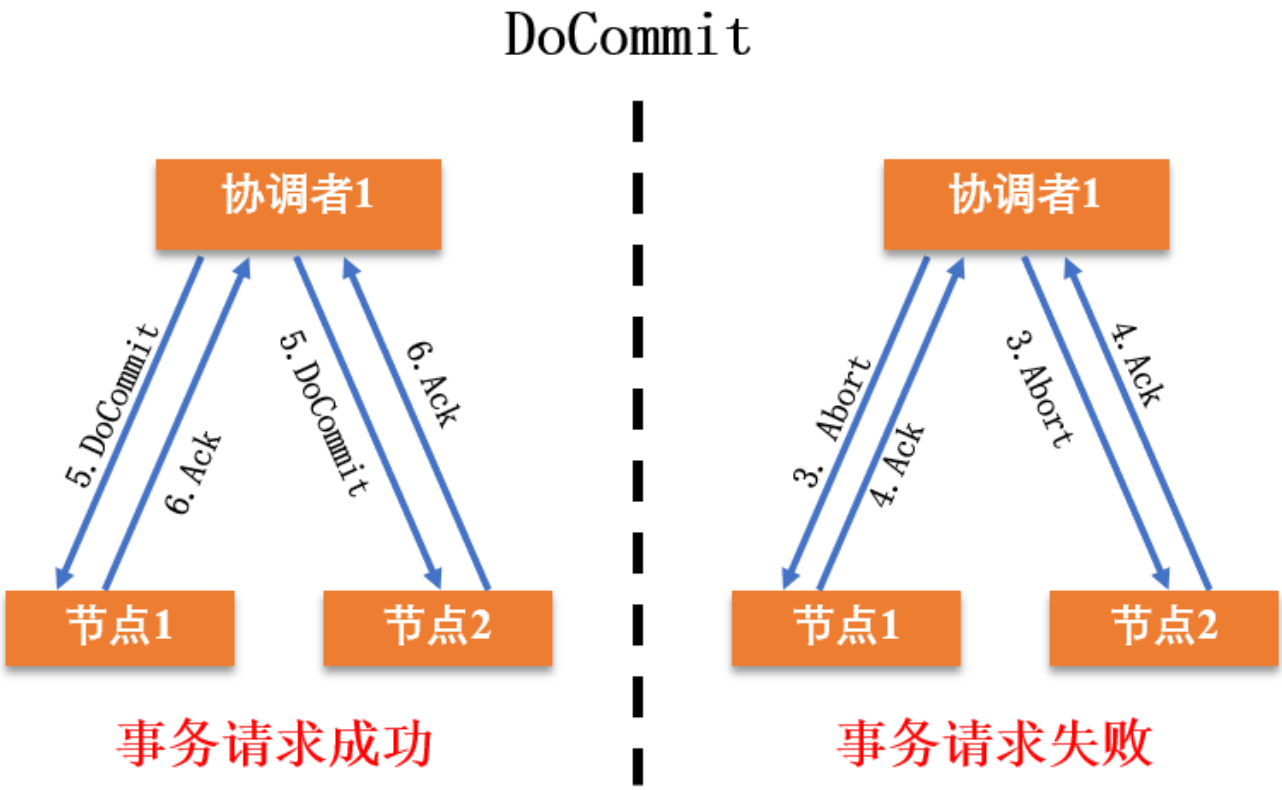
发送中断请求。协调者向所有参与者发送 Abort 请求。

事务回滚。参与者接收到 Abort 消息之后，利用其在 PreCommit 阶段记录的 Undo 信息执行事务的回滚操作，并释放所有锁住的资源。

反馈结果。参与者完成事务回滚之后，向协调者发送 Ack 消息。

终断事务。协调者接收到参与者反馈的 Ack 消息之后，执行事务的终断，并结束事务。

执行阶段不同节点上事务执行成功和失败 (事务终断) 的流程，如下所示。



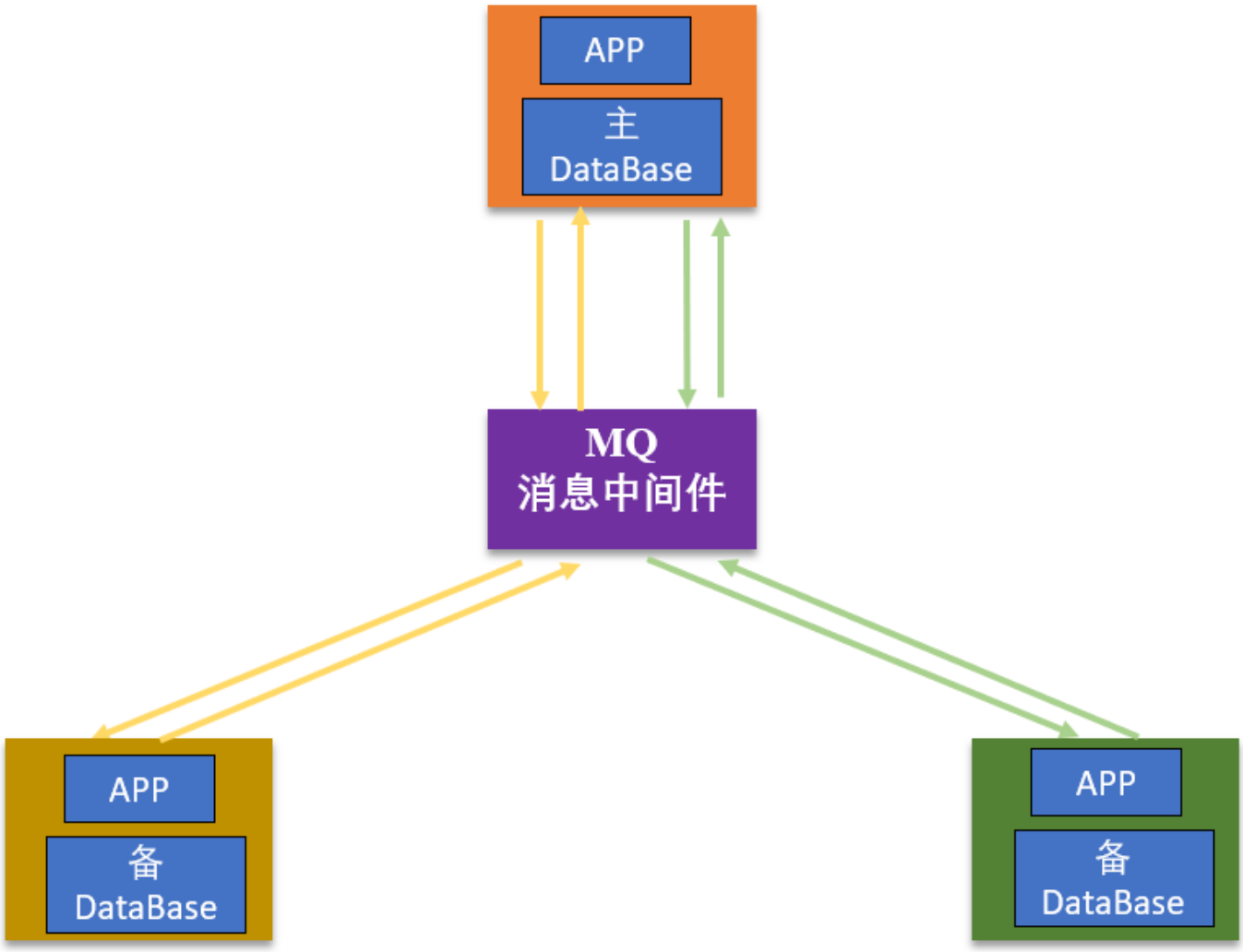
在 DoCommit 阶段，当参与者向协调者发送 Ack 消息后，如果长时间没有得到协调者的响应，在默认情况下，参与者会自动将超时的事务进行提交，不会像两阶段提交那样被阻塞住。

基于分布式消息的最终一致性方案

2PC 和 3PC 这两种方法，有两个共同的缺点，一是都需要锁定资源，降低系统性能；二是，没有解决数据不一致的问题。因此，便有了通过分布式消息来确保事务最终一致性的方案。

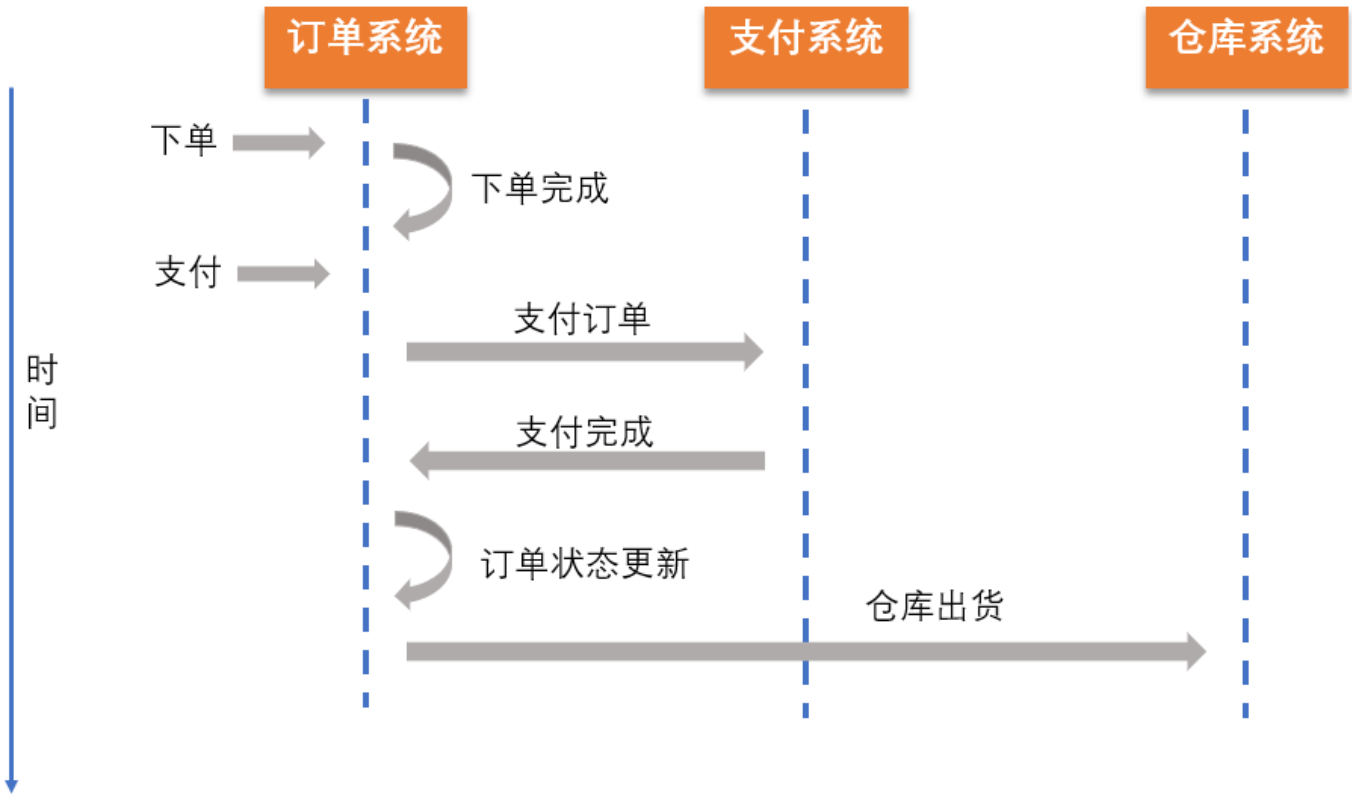
在 eBay 的分布式系统架构中，架构师解决一致性问题的核心思想就是：将需要分布式处理的事务通过消息或者日志的方式异步执行，消息或日志可以存到本地文件、数据库或消息队列中，再通过业务规则进行失败重试。这个案例，就是使用**基于分布式消息的最终一致性方案**解决了分布式事务的问题。

基于分布式消息的最终一致性方案的事务处理，引入了一个消息中间件（Message Queue，MQ），用于在多个应用之间进行消息传递。基于消息中间件协商多个节点分布式事务执行操作的示意图，如下所示。

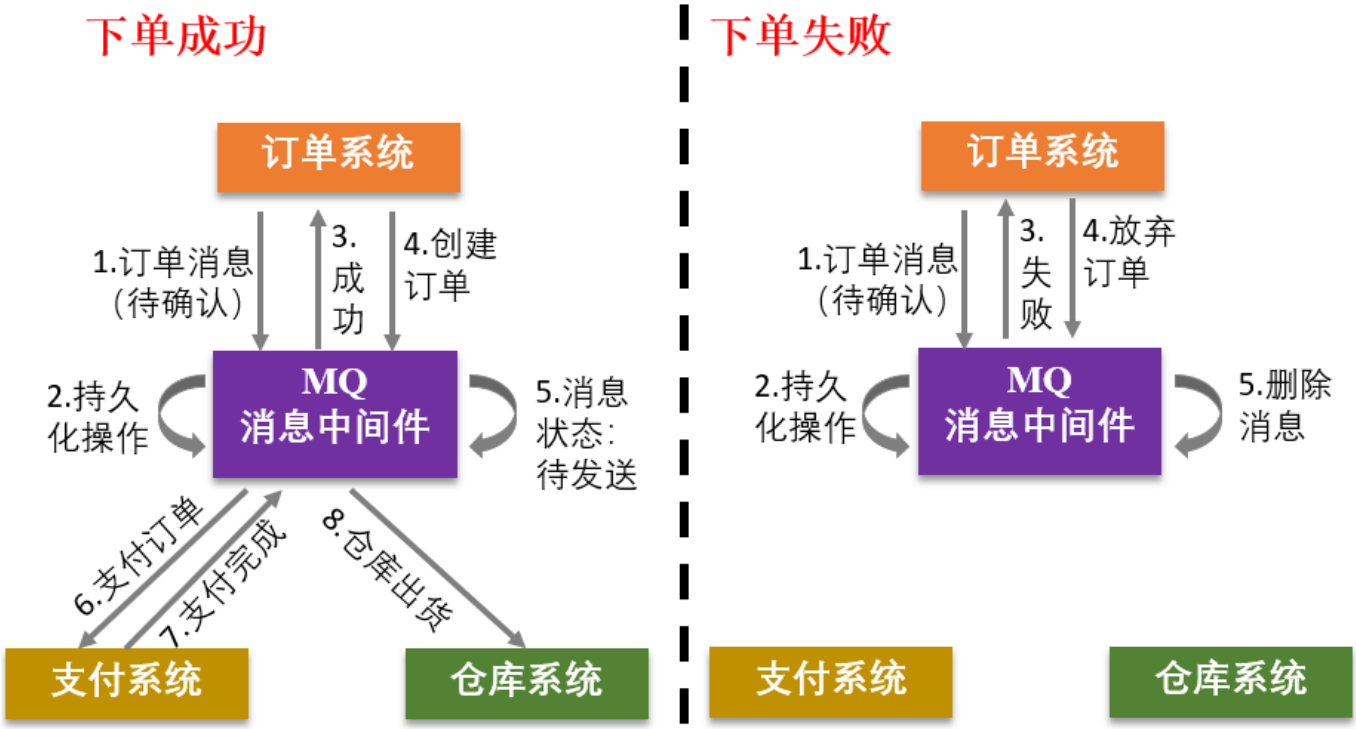


仍然以网上购物为例。假设用户 A 在某电商平台下了一个订单，需要支付 50 元，发现自己的账户余额共 150 元，就使用余额支付，支付成功之后，订单状态修改为支付成功，然后通知仓库发货。

在该事件中，涉及到了订单系统、支付系统、仓库系统，这三个系统是相互独立的应用，通过远程服务进行调用。



根据基于分布式消息的最终一致性方案，用户 A 通过终端手机首先在订单系统上操作，然后整个购物的流程如下所示。



1. 订单系统把订单消息发给消息中间件，消息状态标记为“待确认”。
2. 消息中间件收到消息后，进行消息持久化操作，即在消息存储系统中新增一条状态为“待发送”的消息。

3. 消息中间件返回消息持久化结果（成功 / 失败），订单系统根据返回结果判断如何进行业务操作。失败，放弃订单，结束（必要时向上层返回失败结果）；成功，则创建订单。
4. 订单操作完成后，把操作结果（成功 / 失败）发送给消息中间件。
5. 消息中间件收到业务操作结果后，根据结果进行处理：失败，删除消息存储中的消息，结束；成功，则更新消息存储中的消息状态为“待发送（可发送）”，并执行消息投递。
6. 如果消息状态为“可发送”，则 MQ 会将消息发送给支付系统，表示已经创建好订单，需要对订单进行支付。支付系统也按照上述方式进行订单支付操作。
7. 订单系统支付完成后，会将支付消息返回给消息中间件，中间件将消息传送给订单系统。订单系统再调用库存系统，进行出货操作。

可以看出，分布式事务中，当且仅当所有的事务均成功时整个流程才成功。所以，**分布式事务的一致性是实现分布式事务的关键问题，目前来看还没有一种很简单、完美的方案可以应对所有场景。**

三种实现方式对比

现在，为了方便你理解并记忆这三种方法，我总结了一张表格，从算法一致性、执行方式、性能等角度进行了对比：

	基于XA的二阶段提交协议	三阶段提交协议	基于分布式消息的最终一致性方案
算法一致性类别	强一致性	强一致性	最终一致性
执行方式	同步执行	同步执行	异步执行
同步阻塞问题	有	无	无
单点故障问题	有	无	无
系统并发度	基于XA的二阶段提交方法 < 三阶段提交方法 < 基于分布式消息的最终一致性方案		
分布式事务系统性能	基于XA的二阶段提交方法 < 三阶段提交方法 < 基于分布式消息的最终一致性方案		

知识扩展：刚性事务与柔性事务

在讨论事务的时候，我们经常会提到刚性事务与柔性事务，但却很难区分这两种事务。所以，今天的知识扩展内容，我就来和你说说什么是刚性事务、柔性事务，以及两者之间有何区别？

刚性事务，遵循 ACID 原则，具有强一致性。比如，数据库事务。

柔性事务，其实就是根据不同的业务场景使用不同的方法实现最终一致性，也就是说我们可以根据业务的特性做部分取舍，容忍一定时间内的数据不一致。

总结来讲，与刚性事务不同，柔性事务允许一定时间内，不同节点的数据不一致，但要求最终一致。而柔性事务的最终一致性，遵循的是 BASE 理论。

那，**什么是 BASE 理论**呢？

BASE 理论包括基本可用（Basically Available）、柔性状态（Soft State）和最终一致性（Eventual Consistency）。

基本可用：分布式系统出现故障的时候，允许损失一部分功能的可用性。比如，某些电商 618 大促的时候，会对一些非核心链路的功能进行降级处理。

柔性状态：在柔性事务中，允许系统存在中间状态，且这个中间状态不会影响系统整体可用性。比如，数据库读写分离，写库同步到读库（主库同步到从库）会有一个延时，其实就是一种柔性状态。

最终一致性：事务在操作过程中可能会由于同步延迟等问题导致不一致，但最终状态下，数据都是一致的。

可见，BASE 理论为了支持大型分布式系统，通过牺牲强一致性，保证最终一致性，来获得高可用性，是对 ACID 原则的弱化。具体到今天的三种分布式事务实现方式，二阶段提交、三阶段提交方法，遵循的是 ACID 原则，而消息最终一致性方案遵循的就是 BASE 理论。

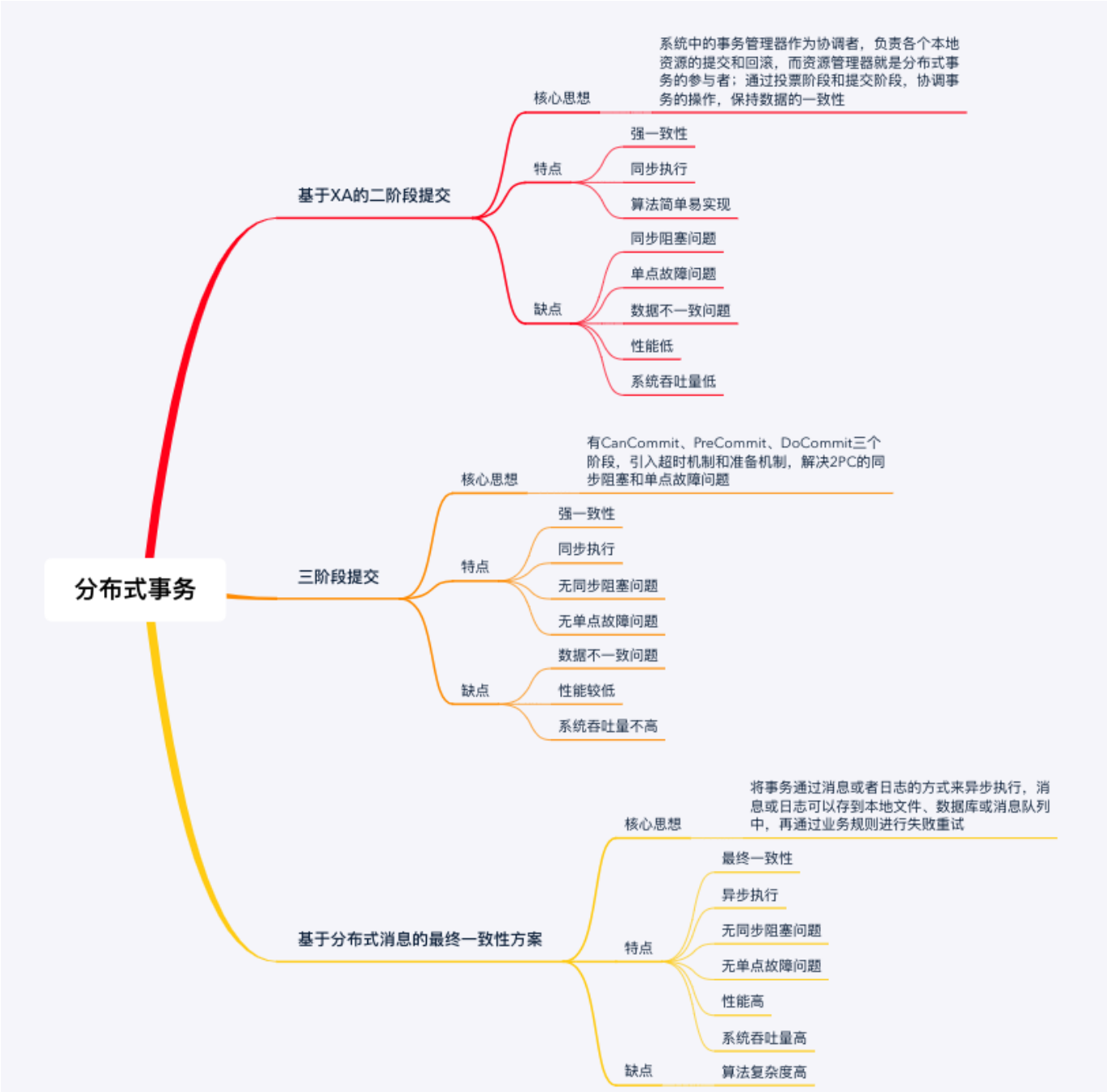
总结

我从事务的 ACID 特性出发，介绍了分布式事务的概念、特征，以及如何实现分布式事务。在关于如何实现分布式的部分，我以网购为例，与你介绍了常见的三种实现方式，即基于 XA 协议的二阶段提交方法，三阶段方法以及基于分布式消息的最终一致性方法。

二阶段和三阶段方法是维护强一致性的算法，它们针对刚性事务，实现的是事务的 ACID 特性。而基于分布式消息的最终一致性方案更适用于大规模分布式系统，它维护的是事务的最终一致性，遵循的是 BASE 理论，因此适用于柔性事务。

在分布式系统的设计与实现中，分布式事务是不可或缺的一部分。可以说，没有实现分布式事务的分布式系统，不是一个完整的分布式系统。分布式事务的实现过程看似复杂，但将方法分解剖析后，你就会发现分布式事务的实现是有章可循的。

我将实现分布式事务常用的三个算法整理为了一张思维导图，以帮助你加深理解与记忆。



思考题

你觉得分布式互斥与分布式事务之间的关系是什么呢？

我是聂鹏程，感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎你把这篇文章分享给更多的朋友一起阅读。我们下期再会！

上一篇　05 | 分布式共识：存异求同

下一篇　07 | 分布式锁：关键重地，非请勿入

精选留言 (10)

写留言

约书亚
2019-10-04

疑问不少
1. 2pc和3pc的第一步到底是不是“类似”？从本文中看，2pc corhort 收到CanCommit已经开始执行事务但不提交，3pc则写着在PreCommit阶段开始预提交。文中说二者第一步“类似”，但其实是非常不类似的吧？...

展开

Geek_54edc1
2019-10-04

个人感觉应该是分布式事务在并发时会用到分布式互斥，比如基于XA协议，在多个分布式事务并发时，哪个会占用事务管理器资源，这时就会用到分布式互斥;基于分布式消息，消息中间件是不用互斥占用的，但是与消息中间件交互的其他资源管理模块（比如订单模块，支付模块等）在多个事务并发的情况下，就需要有互斥机制了

展开

1

忆水寒
2019-10-04

分布式互斥是访问某一个共享资源防止产生不一致现象。分布式事务，是保证一组命令的完整执行或完全不执行。过程来看，都是保证数据一致性的手段。但是两者又不一样，互斥不可以回退（除非发起反向新操作），事务可以回退。

展开

1

A:春哥大魔王
2019-10-06

可靠消息这种方式必须采用mq吗？使用db是不是也可以，看起来只是一个事务状态的存储和管理，是多个两阶段提交的组合啊！

展开

boglond
2019-10-06

CAP理论老师没有讲一讲。

展开

随心而至
2019-10-05

老师能不能放一些参考链接放出来，或者一些引申的链接

tt
2019-10-05

老师，分布式事物，可以用于工作流系统么？

比如工作流上总体有3个节点，A,B,C，每个节点的操作均分为经办和复合操作，当作业顺序经历A→B→C后，一个事务才算完成。

...

展开

啦啦啦

2019-10-04

第一阶段：订单系统中将与用户 A 有关的订单数据库锁住，准备好增加一条关于用户 A 购买 100 件 T 恤的信息，并将同意消息“Yes”回复给协调者。老师这个地方是没有进行插入操作的吧？应该只是锁住了相关数据的吧

展开

Jxin

2019-10-04

1.事务是实现原子性操作的手段，互斥是保障一个线程唯一持有一个竞量的手段。
2.事务的实现需要互斥，互斥的存在不一定是为了实现事务。

Geekki

2019-10-04

分布式事务在访问临界资源时会用到分布式互斥算法?

展开