03 | ACID理论：CAP的酸，追求一致性

2020-02-17 韩健

讲述：于航
时长 12:32 大小 10.06M

你好，我是韩健。

提到 ACID，我想你并不陌生，很多同学也会觉得它容易理解，在单机上实现 ACID 也不难，比如可以通过锁、时间序列等机制保障操作的顺序执行，让系统实现 ACID 特性。但是，一说要实现分布式系统的 ACID 特性，很多同学就犯难了。那么问题来了，为什么分布式系统的 ACID 特性在实现上，比较难掌握呢？

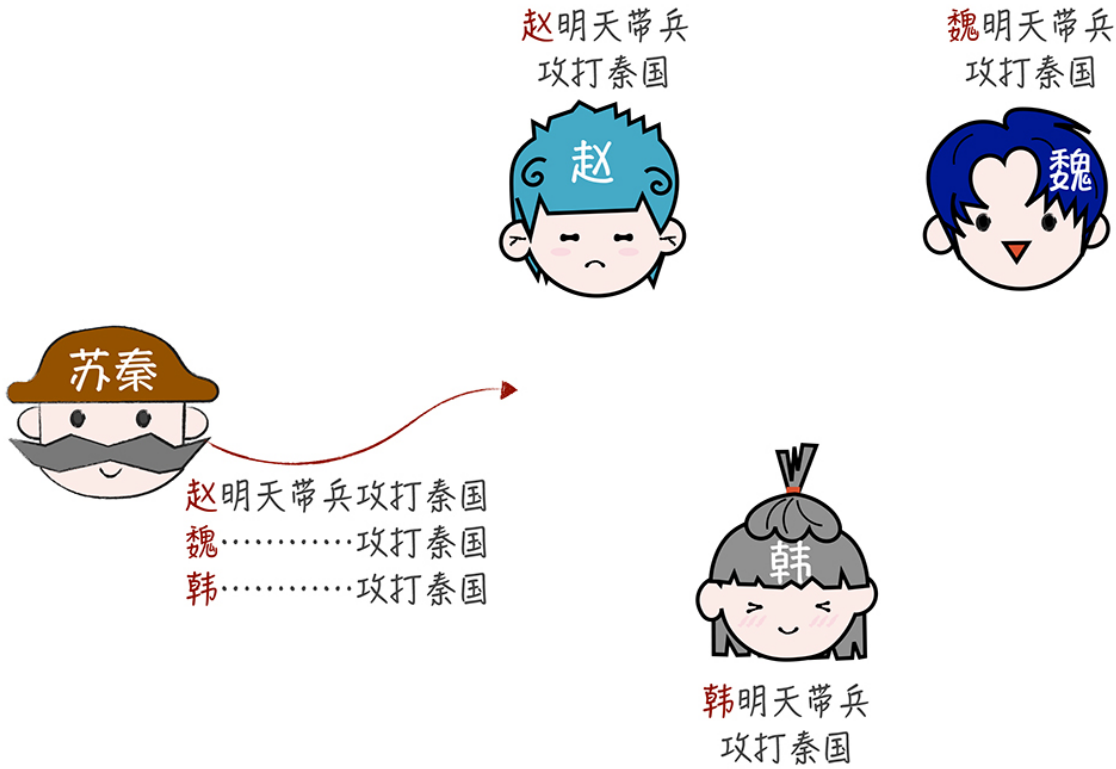
在我看来，ACID 理论是对事务特性的抽象和总结，方便我们实现事务。你可以理解成，如果实现了操作的 ACID 特性，那么就实现了事务。而大多数人觉得比较难，是因为分布式系统涉及多个节点间的操作。加锁、时间序列等机制，只能保证单个节点上操作的 ACID 特性，无法保证节点间操作的 ACID 特性。

那么怎么做才会让实现不那么难呢？答案是你要掌握分布式事务协议，比如二阶段提交协议和 TCC（Try-Confirm-Cancel）。这也是我接下来重点和你分享的内容。

不过在带你了解二阶段提交协议和 TCC 之前，咱们先继续看看苏秦的故事，看这回苏秦又遇到了什么事儿。

最近呢，秦国按捺不住自己躁动的心，开始骚扰魏国边境，魏王头疼，向苏秦求助，苏秦认为“三晋一家亲”，建议魏王联合赵、韩一起对抗秦国。但是这三个国家实力都很弱，需要大家都同意联合，一致行动，如果有任何一方不方便行动，就取消整个计划。

根据侦查情况，明天发动反攻胜算比较大。苏秦想协调赵、魏、韩，明天一起行动。**那么对苏秦来说，他面临的问题是，如何高效协同赵、魏、韩一起行动，并且保证当有一方不方便行动时，取消整个计划。**



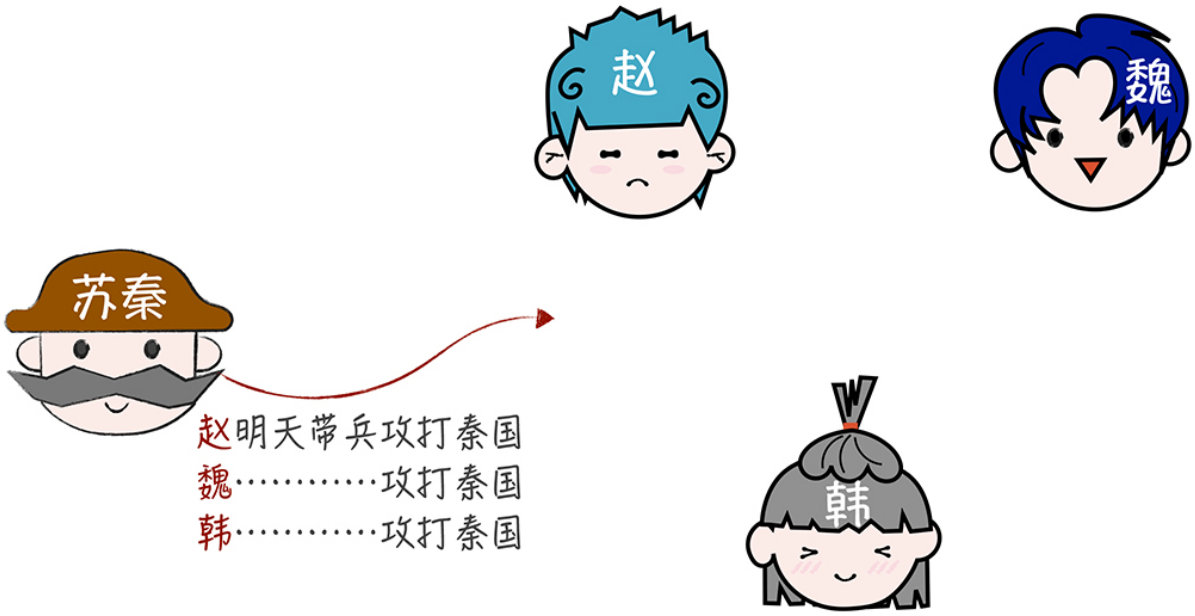
苏秦面对的这个新问题，就是典型的如何实现分布式事务的问题，**赵、魏、韩明天攻打秦国，这三个操作组成一个分布式事务，要么全部执行，要么全部不执行。**

了解了这个问题之后，我们看看如何通过二阶段提交协议和 TCC，来帮助苏秦解决这个难题。

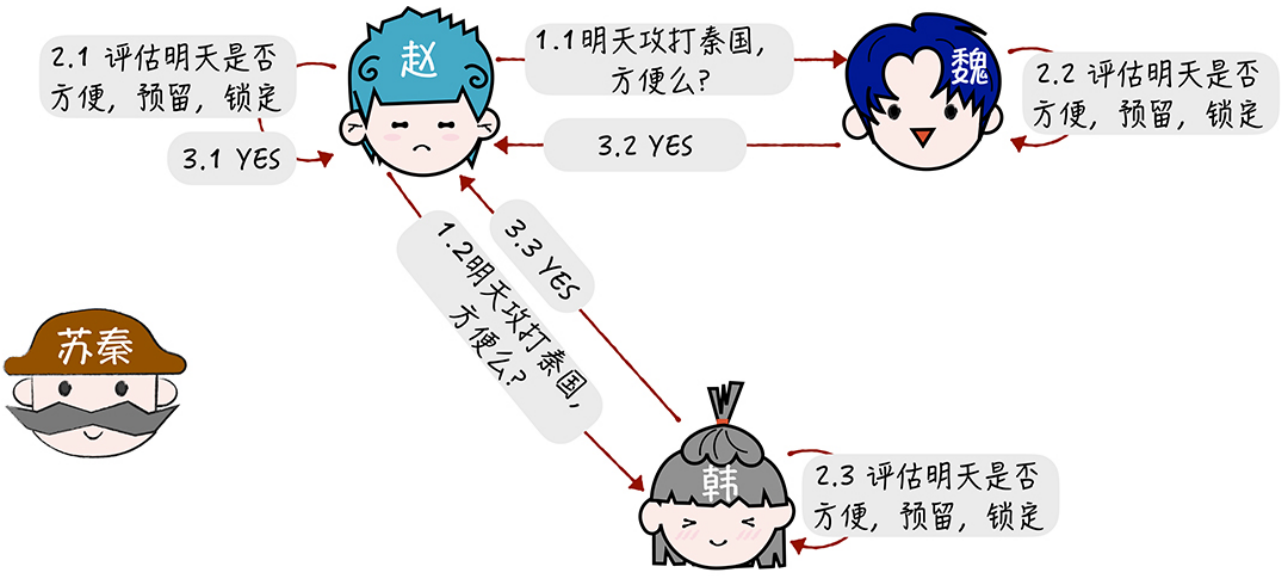
二阶段提交协议

二阶段提交协议，顾名思义，就是通过二阶段的协商来完成一个提交操作，那么具体是怎么操作的呢？

首先，苏秦发消息给赵，赵接收到消息后就扮演协调者（Coordinator）的身份，由赵联系魏和韩，发起二阶段提交：



赵发起二阶段提交后，先进入**提交请求阶段（又称投票阶段）。** 为了方便演示，我们先假设赵、魏、韩明天都能去攻打秦国：

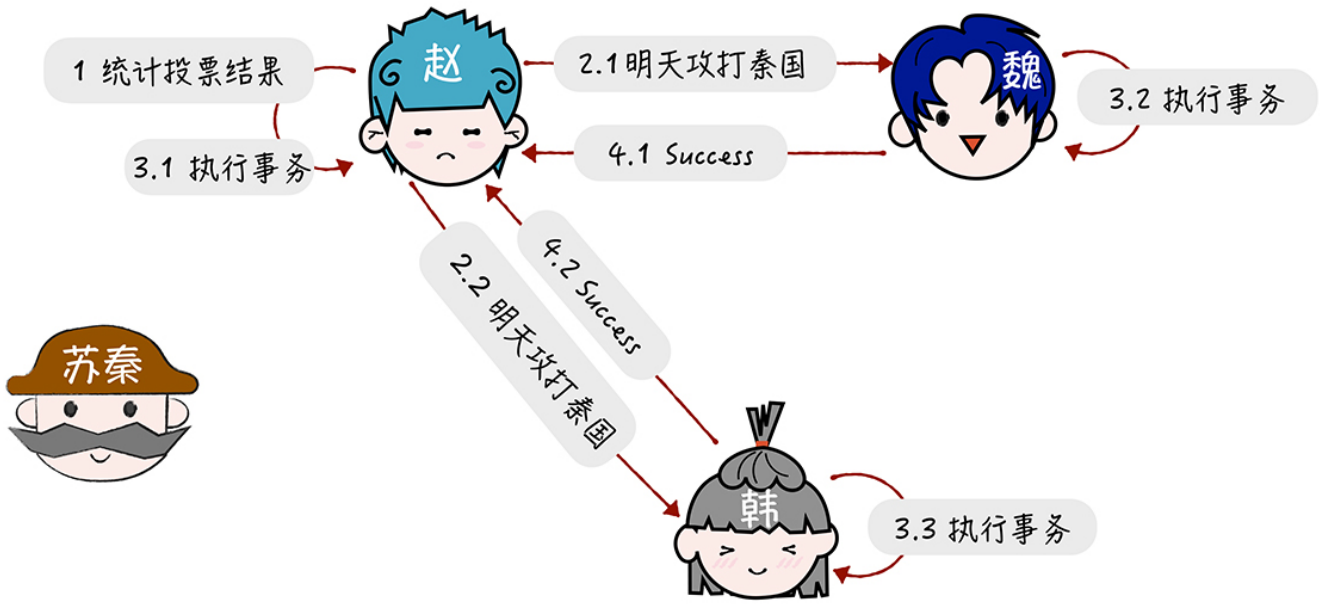


也就是说，第一步，赵分别向魏、韩发送消息：“明天攻打秦国，方便吗？”

第二步，赵、魏、韩，分别评估明天能否去攻打秦国，如果能，就预留时间并锁定，不再安排其他军事活动。

第三步，赵得到全部的回复结果（包括他自己的评估结果），都是 YES。

赵收到所有回复后，进入**提交执行阶段（又称完成阶段）**，也就是具体执行操作了，大致步骤如下：



首先，赵按照“要么全部执行，要么放弃”的原则，统计投票结果，因为所有的回复结果都是 YES，所以赵决定执行分布式事务，明天攻打秦国。

然后，赵通知魏、韩：“明天攻打秦国。”

接到通知之后，魏、韩执行事务，明天攻打秦国。

最后，魏、韩将执行事务的结果返回给赵。

这样一来，赵就将事务执行的结果（也就是赵、魏、韩明天一起攻打秦国），返回给苏秦，那么，这时苏秦就解决了问题，协调好了明天的作战计划。

在这里，赵采用的方法就是二阶段提交协议。在这个协议中：

你可以将“赵明天攻打秦国、魏明天攻打秦国、韩明天攻打秦国”，理解成一个分布式事务操作；

将赵、魏、韩理解为分布式系统的三个节点，其中，赵是协调者（Coordinator），将苏秦理解为业务，也就是客户端；

将消息理解为网络消息；

将“明天能否攻打秦国，预留时间”，理解为评估事务中需要操作的对象和对象状态，是否准备好，能否提交新操作。

需要注意的是，在第一个阶段，每个参与者投票表决事务是放弃还是提交。一旦参与者投票要求提交事务，那么就不允许放弃事务。也就是说，**在一个参与者投票要求提交事务之前，它必须保证能够执行提交协议中它自己那一部分，即使参与者出现故障或者中途被替换掉。**这个特性，是我们需要在代码实现时保障的。

还需要你注意的是，在第二个阶段，事务的每个参与者执行最终统一的决定，提交事务或者放弃事务。这个约定，是为了实现 ACID 中的原子性。

二阶段提交协议最早是用来实现数据库的分布式事务的，不过现在最常用的协议是 XA 协议。这个协议是 X/Open 国际联盟基于二阶段提交协议提出的，也叫作 X/Open Distributed Transaction Processing（DTP）模型，比如 MySQL 就是通过 MySQL XA 实现了分布式事务。

但是不管是原始的二阶段提交协议，还是 XA 协议，都存在一些问题：

在提交请求阶段，需要预留资源，在资源预留期间，其他人不能操作（比如，XA 在第一阶段会将相关资源锁定）；

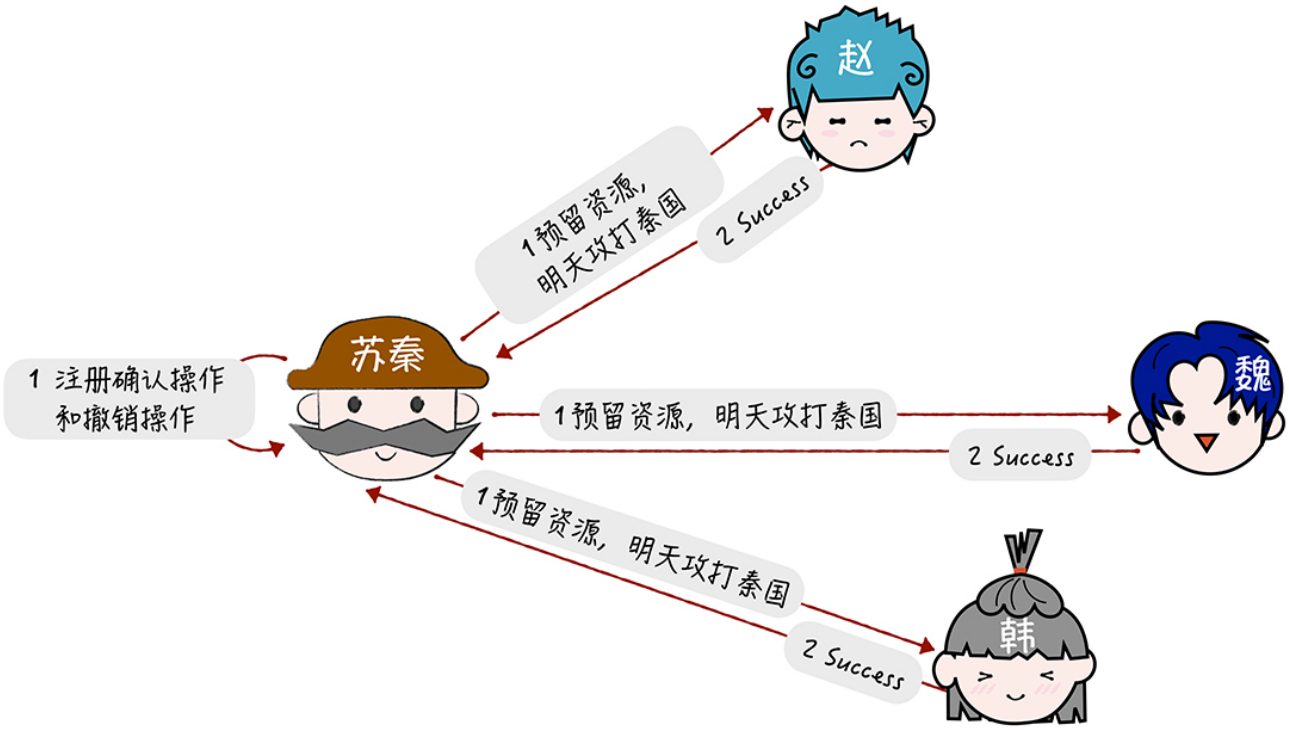
数据库是独立的系统。

因为上面这两点，我们无法根据业务特点弹性地调整锁的粒度，而这些都会影响数据库的并发性能。那用什么办法可以解决这些问题呢？答案就是 TCC。

TCC（Try-Confirm-Cancel）

TCC 是 Try（预留）、Confirm（确认）、Cancel（撤销） 3 个操作的简称，它包含了预留、确认或撤销这 2 个阶段。那么你如何使用 TCC 协议，解决苏秦面临的问题呢?

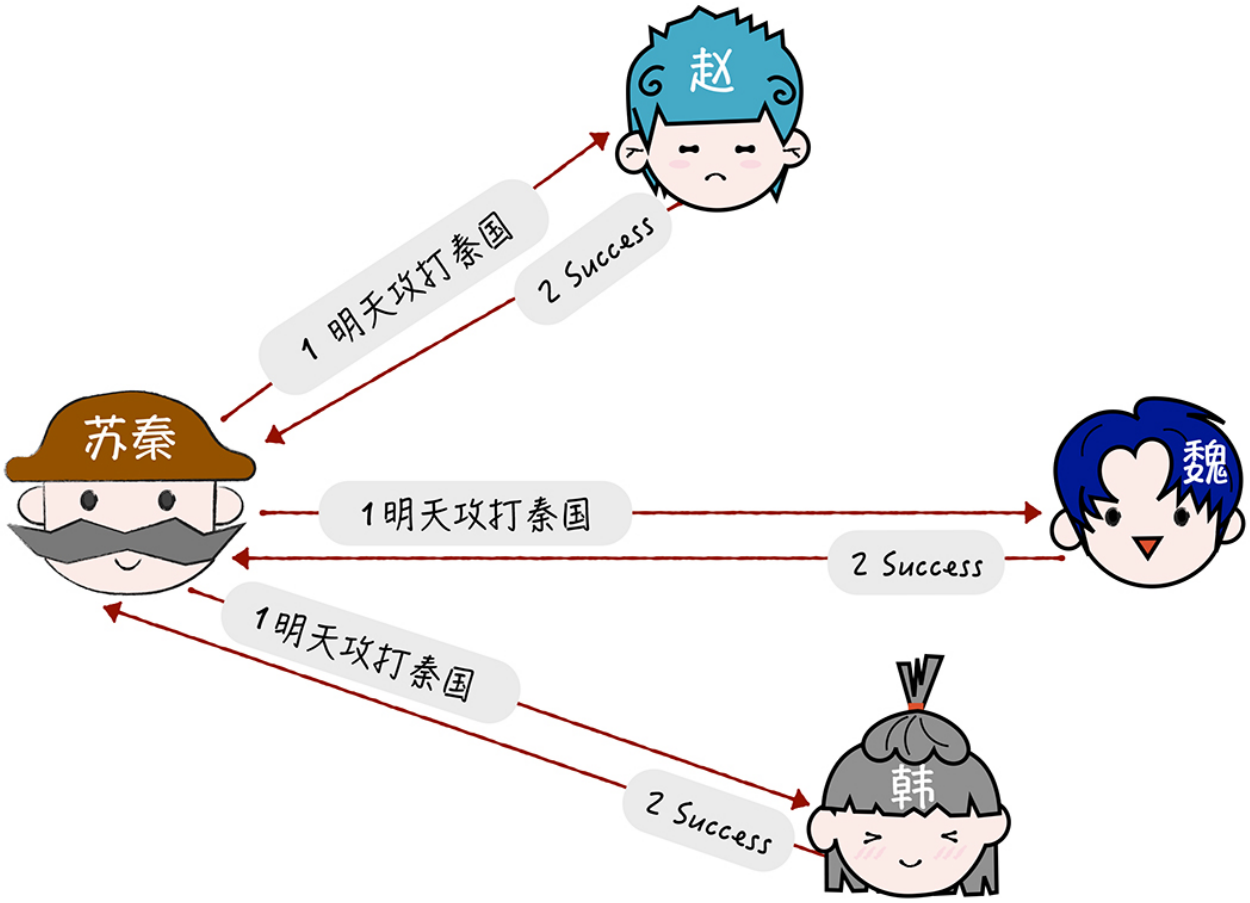
首先，我们先**进入到预留阶段**，大致的步骤如下：



第一步，苏秦分别发送消息通知赵、魏、韩，让他们预留明天的时间和相关资源。然后苏秦实现确认操作（明天攻打秦国），和撤销操作（取消明天攻打秦国）。

第二步，苏秦收到赵、魏、韩的预留答复，都是 OK。

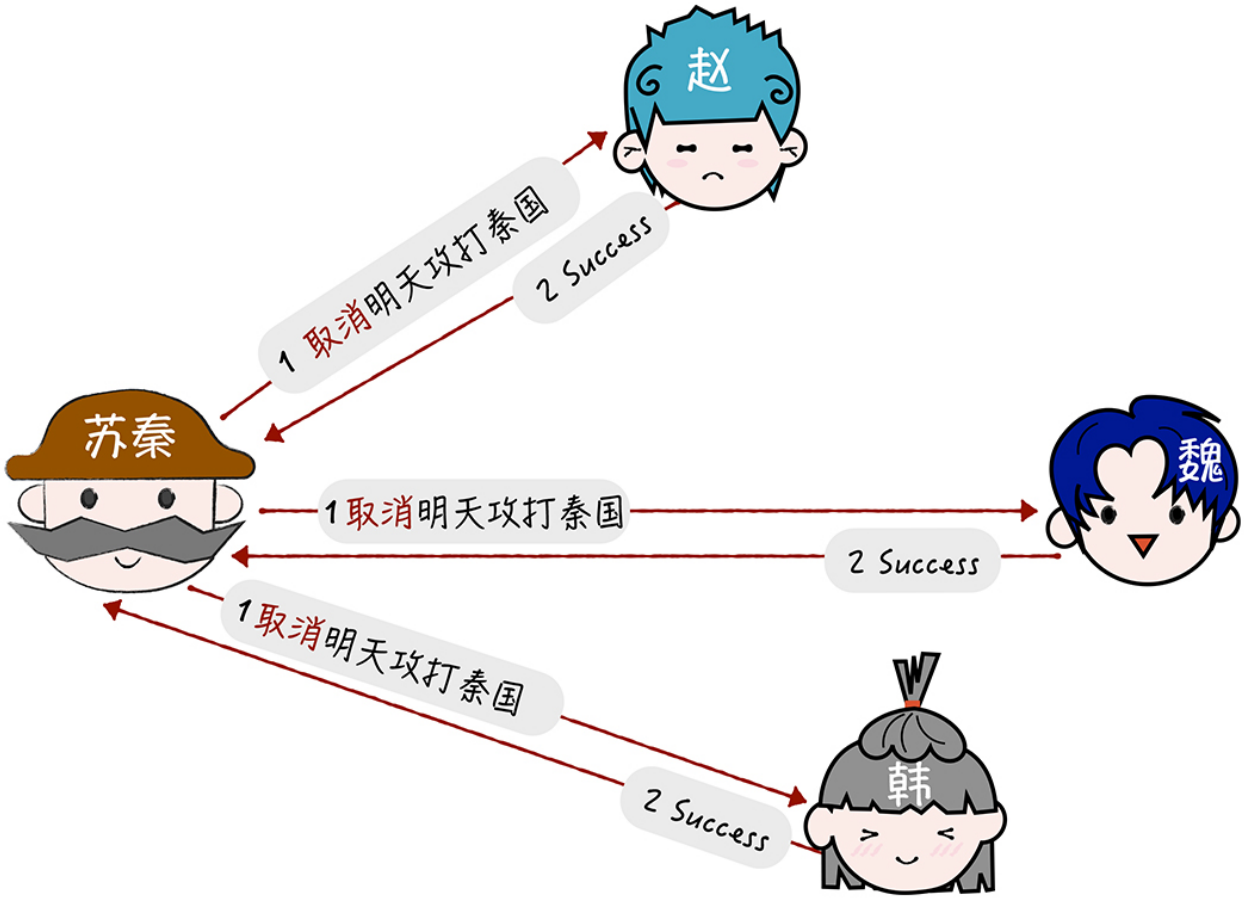
如果预留阶段的执行都没有问题，就进入**确认阶段**，大致步骤如下：



第一步，苏秦执行确认操作，通知赵、魏、韩明天攻打秦国。

第二步，收到确认操作的响应，完成分布式事务。

如果预留阶段执行出错，比如赵的一部分军队还在赶来的路上，无法出兵，那么就进入撤销**阶段**，大致步骤如下：



第一步，苏秦执行撤销操作，通知赵、魏、韩取消明天攻打秦国的计划。

第二步，收到撤销操作的响应。

你看，在经过了预留和确认（或撤销）2 阶段的协商，苏秦实现这个分布式事务：赵、魏、韩三国，要么明天一起进攻，要么明天都按兵不动。

其实在我看来，TCC 本质上是补偿事务，**它的核心思想是针对每个操作都要注册一个与其对应的确认操作和补偿操作（也就是撤销操作）。** 它是一个业务层面的协议，你也可以将 TCC 理解为编程模型，TCC 的 3 个操作是需要在业务代码中编码实现的，为了实现一致性，确认操作和补偿操作必须是等幂的，因为这 2 个操作可能会失败重试。

另外，TCC 不依赖于数据库的事务，而是在业务中实现了分布式事务，这样能减轻数据库的压力，但对业务代码的入侵性也更强，实现的复杂度也更高。所以，我推荐在需要分布式事务能力时，优先考虑现成的事务型数据库（比如 MySQL XA），当现有的事务型数据库不能满足业务的需求时，再考虑基于 TCC 实现分布式事务。

内容小结

本节课我主要带你了解了实现分布式系统 ACID 特性的方法，二阶段提交协议和 TCC，我希望你明确这样几个重点。

1. 二阶段提交协议，不仅仅是协议，也是一种非常经典的思想。二阶段提交在达成提交操作共识的算法中应用广泛，比如 XA 协议、TCC、Paxos、Raft 等。我希望你不仅能理解二阶段提交协议，更能理解协议背后的二阶段提交的思想，当后续需要时，能灵活地根据二阶段提交思想，设计新的事务或一致性协议。
2. 幂等性，是指同一操作对同一系统的任意多次执行，所产生的影响均与一次执行的影响相同，不会因为多次执行而产生副作用。常见的实现方法有 Token、索引等。它的本质是通过唯一标识，标记同一操作的方式，来消除多次执行的副作用。
3. Paxos、Raft 等强一致性算法，也采用了二阶段提交操作，在“提交请求阶段”，只要大多数节点确认就可以，而具有 ACID 特性的事务，则要求全部节点确认可以。所以可以将具有 ACID 特性的操作，理解为最强的一致性。

另外，我想补充一下，三阶段提交协议，虽然针对二阶段提交协议的“协调者故障，参与者长期锁定资源”的痛点，通过引入了询问阶段和超时机制，来减少资源被长时间锁定的情况，不过这会导致集群各节点在正常运行的情况下，使用更多的消息进行协商，增加系统负载和响应延迟。也正是因为这些问题，三阶段提交协议很少被使用，所以，你只要知道有这么个协议就可以了，但如果你想继续研究，可以参考《Concurrency Control and Recovery in Database Systems》来学习。

最后我想强调的是，你可以将 ACID 特性理解为 CAP 中一致性的边界，最强的一致性，也就是 CAP 的酸（Acid）。根据 CAP 理论，如果在分布式系统中实现了一致性，可用性必然受到影响。比如，如果出现一个节点故障，则整个分布式事务的执行都是失败的。实际上，绝大部分场景对一致性要求没那么高，短暂的不一致是能接受的，另外，也基于可用性和并发性能的考虑，**建议在开发实现分布式系统，如果不是必须，尽量不要实现事务，可以考虑采用强一致性或最终一致性**。

课堂思考

既然我提了一些实现分布式事务的方法，比如二阶段提交协议、TCC 等，那么你不妨思考一下，事务型分布式系统有哪些优点，哪些缺点呢？欢迎在留言区分享你的看法，与我一同讨论。

最后，感谢你的阅读，如果这篇文章让你有所收获，也欢迎你将它分享给更多的朋友。

上一篇　02 | CAP理论：分布式系统的PH试纸，用它来测酸碱度

下一篇　04 | BASE理论：CAP的碱，追求可用性

精选留言 (22)

写留言

Joe Black
2020-02-17

在两阶段提交协议中，如果一个节点在第一阶段确认可以提交，然后崩溃了怎么办？第二阶段它实际没法真正应用自己那部分事务。这个看起来没法处理啊

展开

作者回复: 需要将提交相关信息保存到持久存储上，新进程启动后，恢复到之前的状态。

7　7

沉淀的梦想
2020-02-17

老师在"二阶段提交存在的问题"中说 "数据库是独立的系统"，是表达的什么意思？

作者回复: 相比业务，数据库是独立的，也就是说，数据库是独立的第三方软件，咱们可以编程或修改业务代码，但，很少会修改数据库核心代码，更不会根据业务需求，修改实现不同的数据库代码逻辑。

3 4

峰
2020-02-17

一直有个问题，两阶段协议解决的是分布式事务问题，而raft 这些解决的是分布式数据共识问题，即数据在主从副本的条件下保持数据对外始终一样。为什么这两个总混在一起说是解决一致性的呢？？？

作者回复: 常被误解的一个概念，在中文语义中，Consensus和Consistency都被翻译为了一致性。后面，会做个加餐篇，具体说说:)。

3 4

一步
2020-02-17

文中说到 两阶段提交的弊端：在提交请求阶段，需要预留资源，在资源预留期间，其他人不能操作

利用 TCC 可以解决，这里不是很明白，在 TCC中不是也有预留请求，同样预留资源的，难道在这期间其他事务可以使用这部分资源呢？ 如果能使用预留资源，那么在执行Confir...
展开

作者回复: TCC是在业务服务中实现的，更灵活。

3

张高
2020-02-17

文中只说明了预留阶段出错了如何撤销，却没有说明：如果确认阶段出错了，该怎么处理。

作者回复: 通过预留阶段的确认，保证确认阶段执行不会出错。

1 2

Undefined
2020-02-20

真正理解2PC和TCC的使用场景之后就不会分不清这两个东西，老师可以增加使用真实场景，希望老师在后面的讲解中加上相关分布式协议思想的真实场景，要不仅凭叙述理解起来确实麻烦
2PC用在集群间一致性数据同步，所有参与者完成的是同一件事，可以理解为它们在一个start transaction--commit里面，具有强一致性...

展开

作者回复: 在TCC中，数据是最终一致。分布式事务和各算法场景，会在加餐篇做补充。

1

高志强
2020-02-19

想问一下老师，分布式事物，在不同节点不基于数据库的实现，是不是难度很大。如果使用mysql实现事物，多个节点是不是就无法使用了

展开

作者回复: 一般而言，是用数据库就可以了，自己实现的复杂度比较高，如果没有很硬的需求，不推荐自己实现。事务的原子性，要求要么全部提交、要么全部不提交，也就是说，当有分区错误或节点故障时，都无法提交，需要是注意的是，旧数据是一致的，是能读的，只是新数据无法写入。

1

小晏子
2020-02-17

事务型分布式系统优点就是能解决分布式系统中分布式事务的问题，但是缺点也很明显，就是实现复杂，业务侵入性强，比如TCC还要给每个操作写一个cancel操作，而且因为实现复杂，极易引入bug，维护麻烦，遇到问题不好定位，所以能不用尽量不用。

展开

作者回复: 加一颗星:)

1

一步
2020-02-17

感觉 TCC 也采用了两阶段提交的思想，是不是 TCC 是两阶段提交思想的一种实现？
在看文中的图，是代表 TCC 中，是客户端直接通知所有的节点吗？没有所谓的协作者了？

作者回复: 加一颗星:)，客户端在扮演协调者的角色。

1

洛奇
2020-02-17

老师，幂等性的操作能举几个例子吗？

展开

作者回复: 比如，可以通过避免重复执行来实现冥等性，使用id标识操作，每次执行时，检查该id对应的操作是否执行过来了，避免重复执行。这个问题比较共性，后续会出个加餐篇:)。

1 1

施耀南
2020-02-17

谢谢韩老师的讲解，通过学习我的理解是：事务型分布式系统
优点：就是最强一致性，保证所有节点都一致。
缺点：复杂带来的开销，节点增加业务规模增加可能导致延迟。

展开

作者回复: 加一颗星:)

1 1

发条橙子 。
2020-02-23

老师 两阶段可以通过数据库XA线性实现 TCC可以业务代码实现 那现有开源的分布式事务框架是否可以考虑引入 或者是在什么阶段可以引入

展开

作者回复: 数据库分布式事务不能满足时，或者业务比较复杂时，涉及到跨库的数据一致性，考虑TCC。我想补充一点，在绝大部分场景，是不需要分布式事务的，比如互联网后台台，游戏后台等等，推荐在设计架构时，尽量对业务进行拆分，尽量采用最终一致性的方案。


云学
2020-02-22

非常喜欢作者的写作风格，在关键处能够提到代码的实现要点，这正是理论到实践所需要的桥梁！


云学
2020-02-22

分享下我的理解，望指正：二阶段提交是保证不同系统/模块的逻辑结果一致，要么都成功，要么都失败，这些系统的物理数据是完全不相关的； 而Raft是保证多个系统的数据物理一致，举例1: 对于3副本存储系统，上传数据时肯定要保证这3个副本的数据物理一致，举例2:对于一笔交易，需要通过2PC保证订单系统和库存系统的逻辑结果一致，但订单系统和库存系统的物理数据是完全不同的； 如果把2PC协商过程应用在同一个程序的不同进...

展开


zmysang
2020-02-22

事务型分布式系统的数据一致性比较高，在采用acid的情况下可以保证所有节点一致，其他情况也可以保证至少大部分节点一致。
二阶段提交协议和tcc协议的区别在于，tcc是在业务层面的实现，可以更好地根据业务需求去适配，调整预留资源锁粒度的问题。
另外，有一个问题是，tcc为什么不像二阶段提交一样使用一个节点作为协调者，而是使...

展开


时彬斌
2020-02-21

关于一致性有些疑问，强一致性、最终一致性的区别不是很清晰，之前学习raft的时候看到，raft的leader在收到集群内一半以上的数据节点确认操作后就认为事务完成，这时有些节点的数据并没有更新完成，此时理解应该为最终一致性，为什么说raft是强一致性呢？raft强一致性难道仅仅体现在所有请求的处理都是在leader节点处理吗？

展开


姜川
2020-02-19

有个疑问，二阶段提交有资源预留的问题，TCC也需要资源预留，这么看来TCC并没有解决二阶段提交的问题呀，还是不太清楚TCC针对二阶段提交的那种思想在哪里做了提升，除了将协调者转移到非业务节点上。

作者回复: 都是二阶段提交，TCC在业务中实现，更灵活。

陈
2020-02-19

事务型分布式系统的缺点是在并发情况下的资源锁定，优点是保证一致性。

作者回复: 加一颗星:)

fcb的鱼
2020-02-18

问下，在"二阶段提交存在的问题"中说 "数据库是独立的系统"，想传递什么意思呢。

作者回复: 相比业务，数据库是独立的，也就是说，数据库是独立的第三方软件，咱们可以编程或修改业务代码，但，很少会修改数据库核心代码，更不会根据业务需求，修改实现不同的数据库代码逻辑。

超能力先生
2020-02-17

TCC try阶段的时候应该是直接commit了事务，confirm做确认，不然就撤销。这样才能达到减少资源锁定时间。感觉图里第一部预留资源的说法有点歧义。

展开

作者回复: 在confirm阶段执行的:)，TCC也是二阶段提交。

1