分布式事务协议

事务的四大特性——ACID：

1. 原子性(Atomicity)
2. 一致性(Consistency)
3. 隔离性(Isolation)
4. 持久性(Durability)

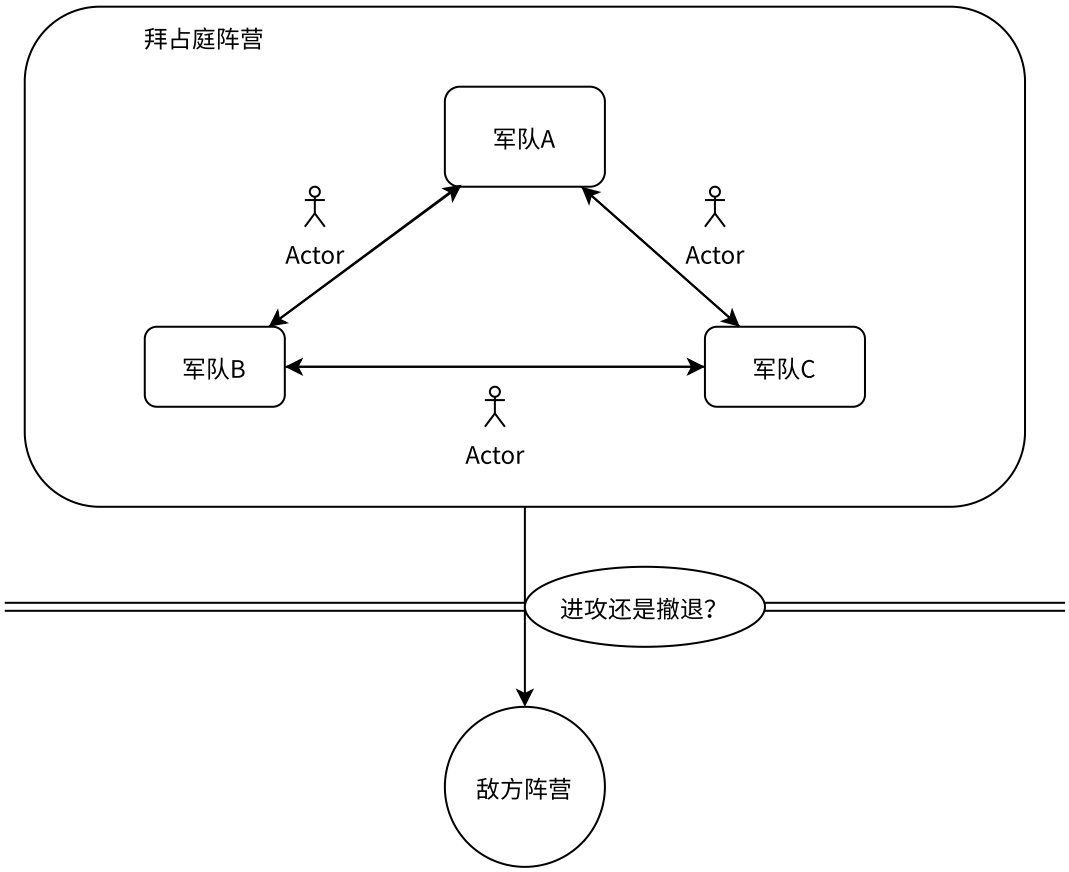
单节点系统的事务还是比较好实现的，但是一旦到了分布式系统的事务就有点让人头疼了；因为它涉及到的操作是跨系统的。

分布式事务协议：二阶段提交协议和 TCC（Try-Confirm-Cancel）。

二阶段提交协议

假设这么一个场景，三个将领ABC，共同决定是进攻还是撤退！

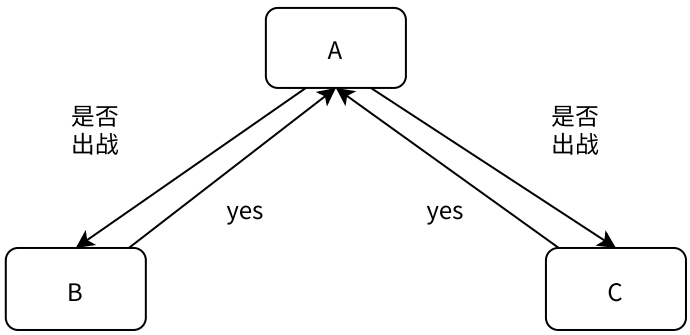
要求：商量好一致的作战方案之后，任何一方临时无法满足作战方案，则三支军队都同时恢复到初始状态；比如协商的一致性方案是进攻，但是在进攻的当天 B 因为某种原因无法出战，则所有军队的状态都恢复到协商前。



第一阶段：（投票阶段）

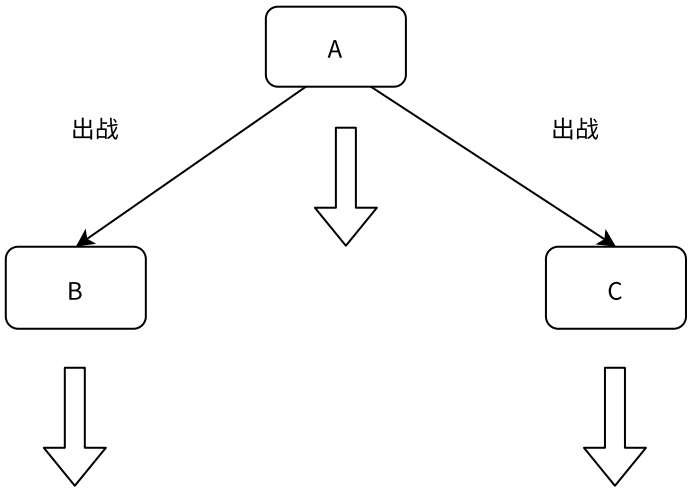
A先试探性地询问一下BC的意见：你们是否可以进攻?

如果BC都可以进攻，则会返回yes，接着去做好作战准备；否则回复 no



第二阶段：

根据收到的回信，A会发现BC都可以出战，所以A会发出真正出战的指令



XA协议

MySQL 在事务提交的时候，会使用内部 XA 两阶段提交协议来处理 Redo log 和 bing log 两个协议。

因为事务提交前，redo log 和 bing log 都是有所更新的，假设不采用二阶段提交，则可能发生：

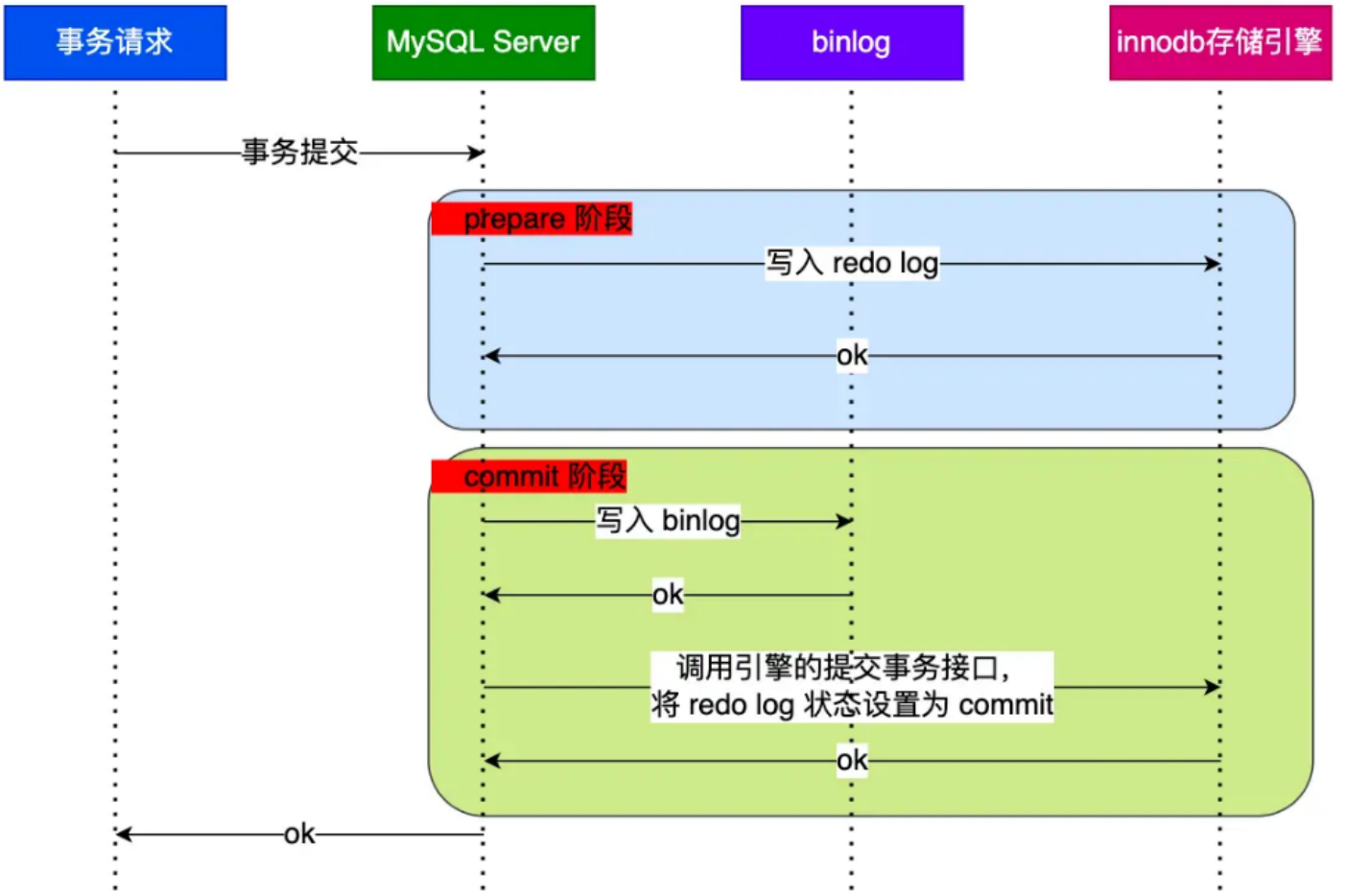
> 假设先写 redo log，后写 bing log，但是在写 bing log的时候宕机了；这样 redo log 中就有当前事务的操作记录，而 bing log 中则没有；而主库使用 redo log 恢复，从库使用 bing log 同步数据，最终会导致主从不一致。
>
> 假设先写 bing log，后写 redo log，但是在写 redo log的时候宕机了；这样 redo log 中就没有有当前事务的操作记录，而 bing log 中则有；而主库使用 redo log 恢复，从库使用 bing log 同步数据，最终会导致主从不一致。

因此，MySQL 在这里是使用二阶段提交协议来解决这个问题的。

MySQL为什么需要两阶段提交

第一阶段——准备阶段：将数据写入 redo log，并设置其状态为 prepare

第二阶段——提交阶段：将数据写入 bing log，并回去将 redo log 的状态设置为 committed

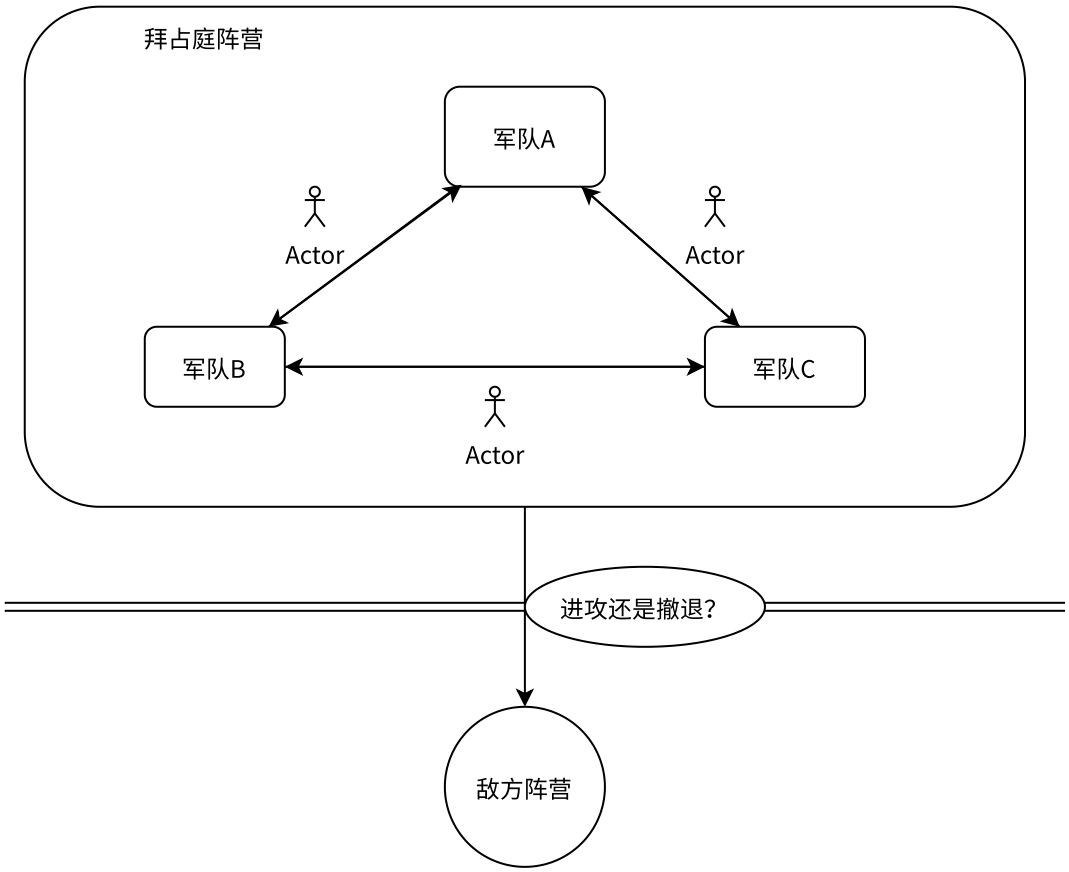


TCC（Try-Confirm-Cancel）

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。

假设这么一个场景，三个将领ABC，共同决定是进攻还是撤退！

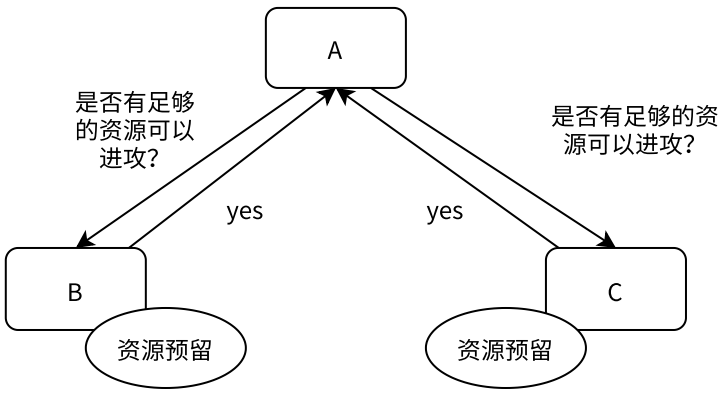
要求：商量好一致的作战方案之后，任何一方临时无法满足作战方案，则三支军队都同时恢复到初始状态；比如协商的一致性方案是进攻，但是在进攻的当天 B 因为某种原因无法出战，则所有军队的状态都恢复到协商前。



第一阶段：（预留阶段）

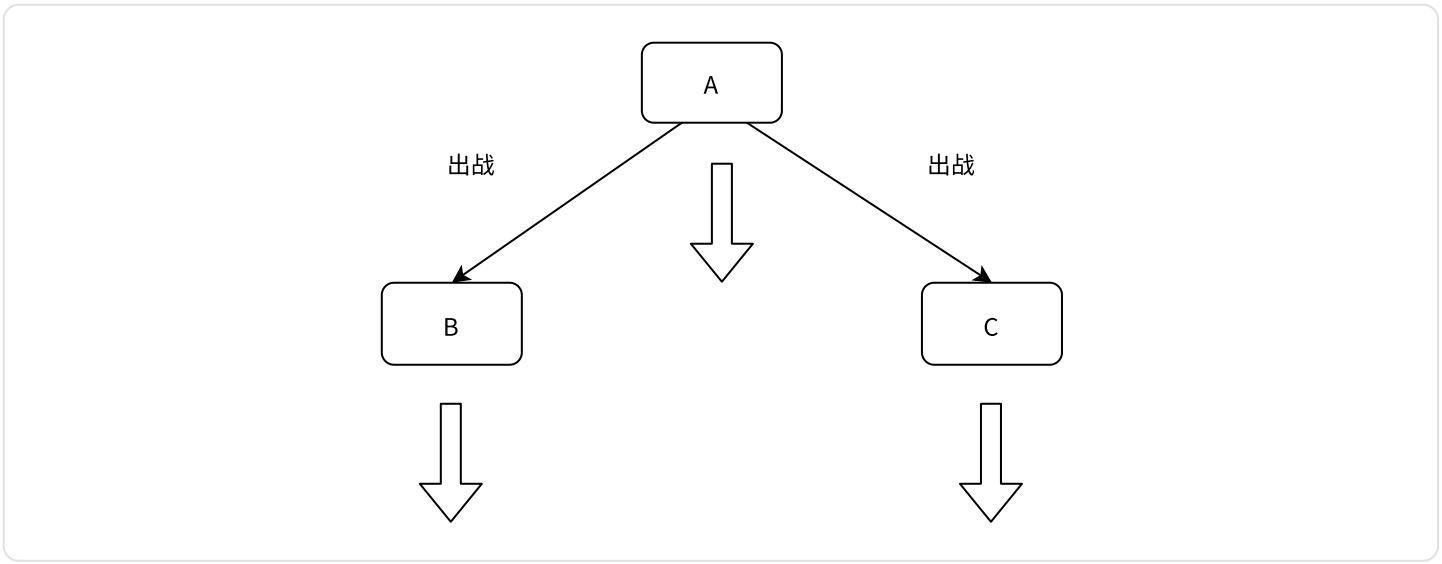
A先试探性地询问一下BC的意见：你们是否有足够的资源可以进攻?

如果BC都有足够的资源可以进攻，则会返回yes，接着去做好资源预留；否则回复 no



第二阶段：（确认阶段）

根据收到的回信，A会发现BC都可以出战，所以A会发出真正出战的指令



第二阶段：（补偿阶段）（撤销阶段）

如果A会发现B或者C不可以出战，所以A会发出撤销的指令