# 两阶段提交

系统中有两类节点：协调者和事务参与者，协议中假设每个节点都会记录写前日志并持久性存储，即使节点发生故障日志也不会丢失。

## 1.1.两阶段执行

* 请求阶段：

协调者将通知事务参与者准备提交事务，然后进入表决过程，事务参与者告知协调者自己的决策。

* 提交阶段

协调者基于第一阶段的投票结果进行决策：提交或是取消。当其仅当所有事务参与者同意提交，协调者参会通知所有参与者提交事务，否则通知所欲参与者回滚

## 1.2.缺点

* 同步阻塞问题

执行过程中，所有参与节点都是事务阻塞的，当参与者占有公共资源时，其他第三方节点访问公共资源不得不处于阻塞状态

* 单点故障问题

一旦协调者发生故障，参与者会一直阻塞下去，尤其是在第二阶段，协调者发生故障，所有参与者都还处于锁定事务资源的状态中，无法继续完成事务操作

* 数据不一致

在第二阶段中，协调者向所有参与者发送commit请求之后，发生了局部网络异常或者是发送commit之后协调者发生了故障，这会导致一部分参与者收到了commit请求，其他未收到commit请求的无法执行事务提交

## 1.3.无法解决的问题

* 协调者出错，参与者也出错，两阶段无法保证事务执行完整性
* 协调者发送commit消息之后宕机，唯一收到这条消息的参与者也宕机，那么及时协调者通过选举协议产生了新的协调者，这条事务的状态也是不确定的，没人知道事务是否已经被提交

# 三阶段提交

三阶段提交协议在协调者和参与者中都引入了超时机制，并且把两阶段提交的第一阶段分成两步：询问，然后锁住资源，最后真正提交。

## 2.1.三阶段的执行

* canCommit：与2PC的准备阶段很像，协调者像参与者发送commit请求，参与者如果可以提交就返回yes响应，否则返回no响应
* preCommit：协调者根据参与者的反应情况决定是否可以继续事务的preCommit操作

1. 加入收到的反馈都是yes，那么就会进行事务的预执行：发送预提交请求，事务预提交，响应反馈。
2. 如果有一个参与者发送了no响应，或是等待超时之后，协调者就会中断事务，发送中断请求，参与者会中断事务（收到中断请求，或是超时未收到协调者的请求）

* doCommit：该阶段进行真正的事务提交，可以分为两种情况：

1. 执行提交：首先协调者发送提交请求，参与者收到提交请求后提交事务，并响应反馈给协调者，协调者收到所有参与者的响应，完成事务
2. 中断事务：协调者没有收到参与者的ack响应（或是超时，或是收到的不是ack响应），则中断事务

注意：进入阶段3后，无论协调者出现问题，或者协调者与参与者网络出现问题，都会导致参与者无法接收到协调者发出的doCommit请求或abort请求，此时，参与者都会在等待超时之后，继续执行事务提交（这是基于概率决定的，当进入第三阶段时，说明参与者在第二阶段已经收到了preCommit请求，那么协调者产生preCommit的请求的前提条件是第二阶段开始前，收到所有参与者的回复都是yes，所以当进入第三阶段时，由于网络超时原因，虽然参与者没有收到commit或是abort请求，他有理由相信，成功提交的几率很大）。

## 2.2.三阶段的优点与缺点

* 降低了阻塞范围，等待超时后，参与者会主动提交事务而不是一直持有事务的资源
* 避免了协调者单点故障，协调者发生故障后，在协调者恢复之前，参与者之间可以互相通信确定事务是应该提交还是应该终止
* 脑裂问题依然存在，即在参与者收到preCommit请求后等待最终指令，如果此时协调者无法与参与者正常通信，协调者发送abort请求没有被参与者即时接收到，那么参与者会等待超时之后执行commit操作，这就和其他收到abort命令并执行回滚的参与者之间存在数据不一致的情况
* 虽然能避免阻塞状态，但需要更多的通讯次数，实现比较复杂，因此实际应用较少，大多使用两阶段提交