专题18：一致性协议 （史上最全、定期更新）

本文版本说明：V2

> 此文的格式，由markdown 通过程序转成而来，由于很多表格，没有来的及调整，出现一个格式问题，尼恩在此给大家道歉啦。
>
> 由于社群很多小伙伴，在面试，不断的交流最新的面试难题，所以，《Java面试红宝书》， 后面会不断升级，迭代。
>
> 本专题，作为 《Java面试红宝书》专题之一， 《Java面试红宝书》一共**30个面试专题，后续还会增加**

《Java面试红宝书》升级的规划为：

后续基本上，**每一个月，都会发布一次**，最新版本，可以扫描扫架构师尼恩微信， 发送 “领取电子书” 获取。

尼恩的微信二维码在哪里呢？ 请参见文末

面试问题交流说明：

如果遇到面试难题，或者职业发展问题， 或者中年危机问题，都可以来 疯狂创客圈社群交流，

入交流群，加尼恩微信即可， 发送**“入群”**

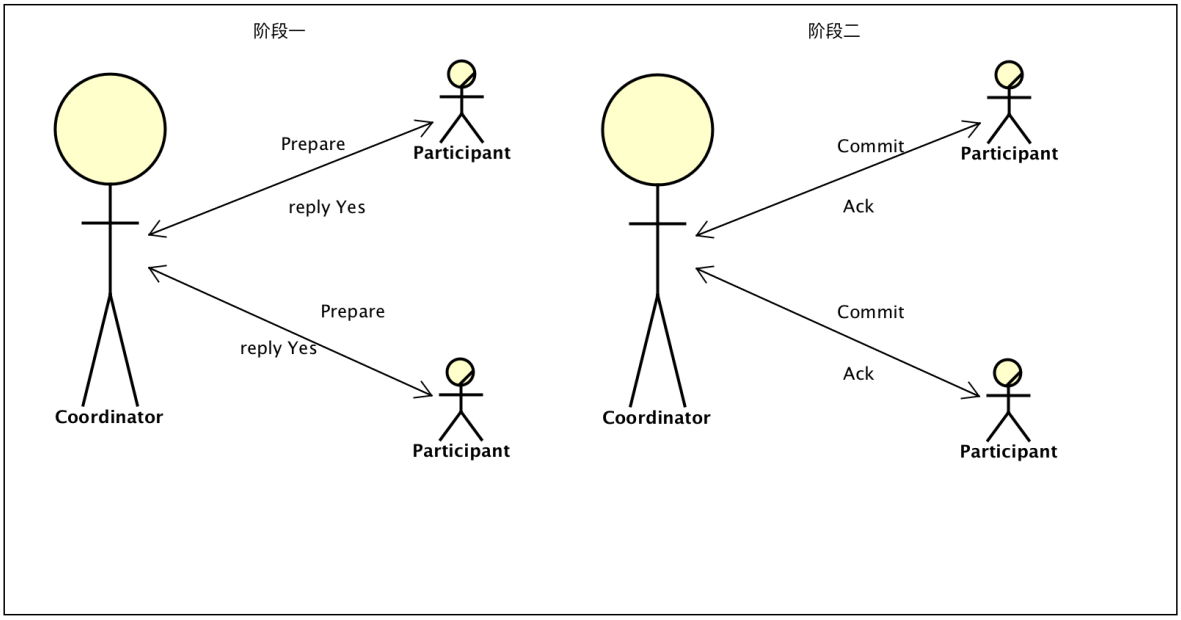
尼恩的微信二维码在哪里呢？ 请参见文末

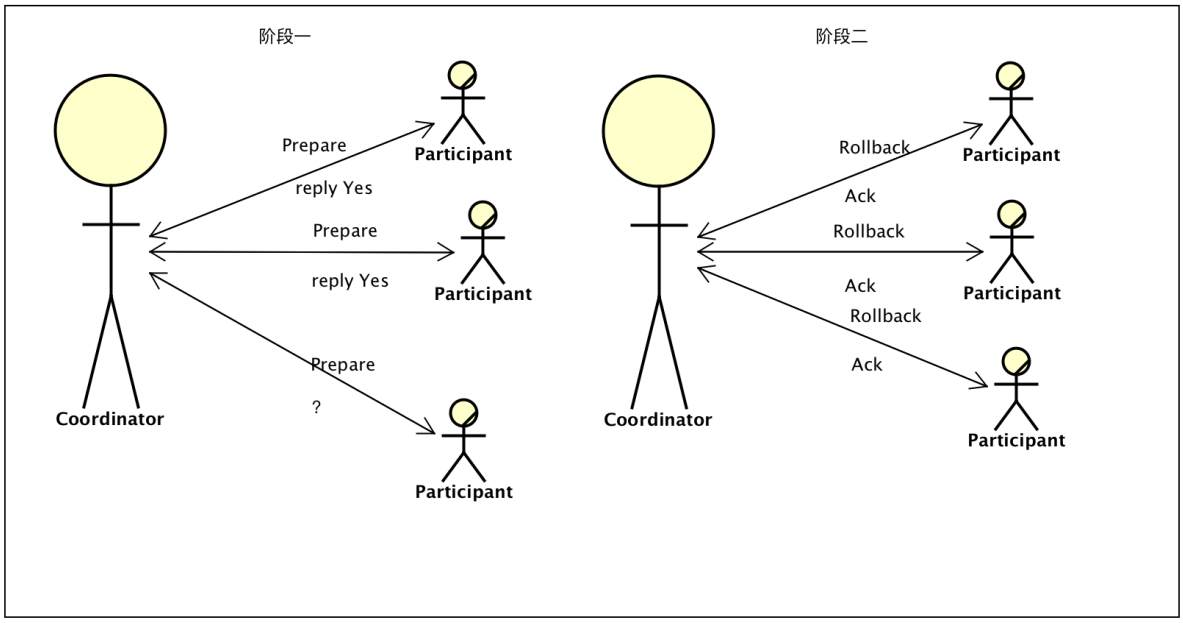
一致性协议

常见的一致性协议 有二阶段提交（2PC）、三阶段提交（3PC）、Paxos、Raft等算法，在本文将介绍他们中的一部分。

2PC

2PC即Two-Phase Commit，二阶段提交。广泛应用在数据库领域，为了使得基于分布式架构的所有节点可以在进行事务处理时能够保持原子性和一致性。绝大部分关系型数据库，都是基于2PC完成分布式的事务处理。
顾名思义，2PC分为两个阶段处理，





阶段一：提交事务请求

1. 事务询问。协调者向所有参与者发送事务内容，询问是否可以执行提交操作，并开始等待各参与者进行响应；
2. 执行事务。各参与者节点，执行事务操作，并将Undo和Redo操作计入本机事务日志；
3. 各参与者向协调者反馈事务问询的响应。成功执行返回Yes，否则返回No。

阶段二：执行事务提交

协调者在阶段二决定是否最终执行事务提交操作。这一阶段包含两种情形：

执行事务提交
所有参与者reply Yes，那么执行事务提交。

1. 发送提交请求。协调者向所有参与者发送Commit请求；
2. 事务提交。参与者收到Commit请求后，会**正式执行事务提交操作**，并在完成提交操作之后，释放在整个事务执行期间占用的资源；
3. 反馈事务提交结果。参与者在完成事务提交后，写协调者发送Ack消息确认；
4. 完成事务。协调者在收到所有参与者的Ack后，完成事务。

中断事务

事情总会出现意外，当存在某一参与者向协调者发送No响应，或者等待超时。协调者只要无法收到所有参与者的Yes响应，就会中断事务。

1. 发送回滚请求。协调者向所有参与者发送Rollback请求；
2. 回滚。参与者收到请求后，利用本机Undo信息，执行Rollback操作。并在回滚结束后释放该事务所占用的系统资源；
3. 反馈回滚结果。参与者在完成回滚操作后，向协调者发送Ack消息；
4. 中断事务。协调者收到所有参与者的回滚Ack消息后，完成事务中断。

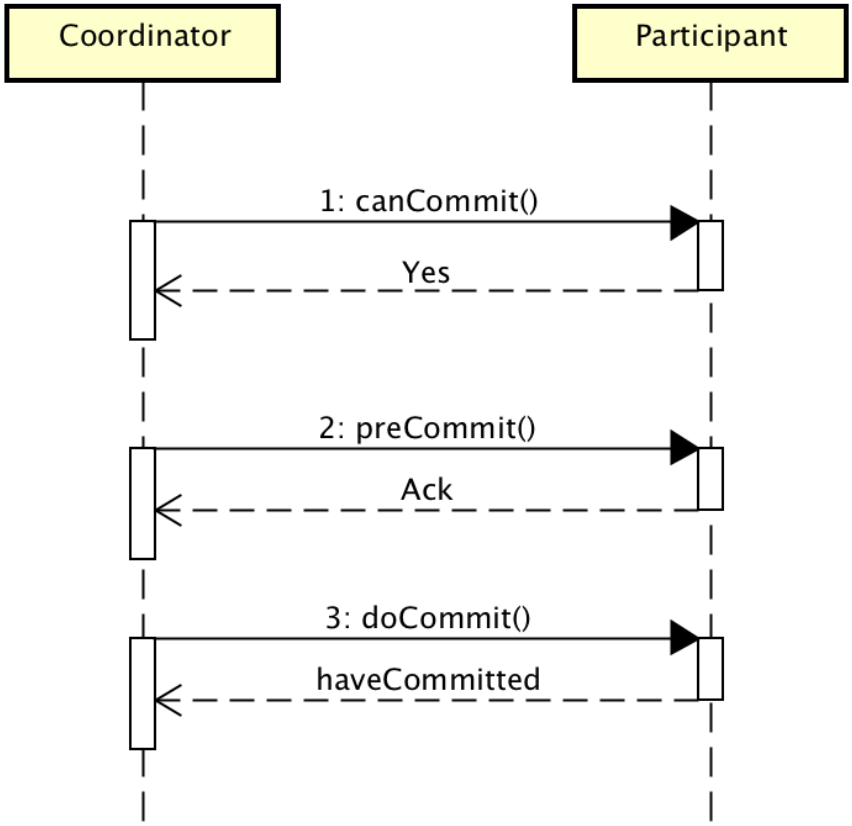
2PC具有明显的优缺点：

优点主要体现在实现原理简单；
缺点比较多：

- 2PC的提交在执行过程中，所有参与事务操作的逻辑都处于阻塞状态，也就是说，各个参与者都在等待其他参与者响应，无法进行其他操作；
- 协调者是个单点，一旦出现问题，其他参与者将无法释放事务资源，也无法完成事务操作；
- 数据不一致。当执行事务提交过程中，如果协调者向所有参与者发送Commit请求后，发生局部网络异常或者协调者在尚未发送完Commit请求，即出现崩溃，最终导致只有部分参与者收到、执行请求。于是整个系统将会出现数据不一致的情形；
- 保守。2PC没有完善的容错机制，当参与者出现故障时，协调者无法快速得知这一失败，只能严格依赖超时设置来决定是否进一步的执行提交还是中断事务。

3PC

针对2PC的缺点，研究者提出了3PC，即Three-Phase Commit。作为2PC的改进版，3PC将原有的两阶段过程，重新划分为CanCommit、PreCommit和do Commit三个阶段。



阶段一：CanCommit

1. 事务询问。协调者向所有参与者发送包含事务内容的canCommit的请求，询问是否可以执行事务提交，并等待应答；
2. 各参与者反馈事务询问。正常情况下，如果参与者认为可以顺利执行事务，则返回Yes，否则返回No。

阶段二：PreCommit

在本阶段，协调者会根据上一阶段的反馈情况来决定是否可以执行事务的PreCommit操作。有以下两种可能：

执行事务预提交

1. 发送预提交请求。协调者向所有节点发出PreCommit请求，并进入prepared阶段；
2. 事务预提交。参与者收到PreCommit请求后，会执行事务操作，并将Undo和Redo日志写入本机事务日志；
3. 各参与者成功执行事务操作，同时将反馈以Ack响应形式发送给协调者，同事等待最终的Commit或Abort指令。

中断事务
加入任意一个参与者向协调者发送No响应，或者等待超时，协调者在没有得到所有参与者响应时，即可以中断事务：

1. 发送中断请求。 协调者向所有参与者发送Abort请求；
2. 中断事务。无论是收到协调者的Abort请求，还是等待协调者请求过程中出现超时，参与者都会中断事务；

阶段三：doCommit

在这个阶段，会真正的进行事务提交，同样存在两种可能。

执行提交

1. 发送提交请求。假如协调者收到了所有参与者的Ack响应，那么将从预提交转换到提交状态，并向所有参与者，发送doCommit请求；
2. 事务提交。参与者收到doCommit请求后，会正式执行事务提交操作，并在完成提交操作后释放占用资源；
3. 反馈事务提交结果。参与者将在完成事务提交后，向协调者发送Ack消息；
4. 完成事务。协调者接收到所有参与者的Ack消息后，完成事务。

中断事务
在该阶段，假设正常状态的协调者接收到任一个参与者发送的No响应，或在超时时间内，仍旧没收到反馈消息，就会中断事务：

1. 发送中断请求。协调者向所有的参与者发送abort请求；
2. 事务回滚。参与者收到abort请求后，会利用阶段二中的Undo消息执行事务回滚，并在完成回滚后释放占用资源；
3. 反馈事务回滚结果。参与者在完成回滚后向协调者发送Ack消息；
4. 中端事务。协调者接收到所有参与者反馈的Ack消息后，完成事务中断。

3PC的优缺点：

3PC有效降低了2PC带来的参与者阻塞范围，并且能够在出现单点故障后继续达成一致；
但3PC带来了新的问题，在参与者收到preCommit消息后，如果网络出现分区，协调者和参与者无法进行后续的通信，这种情况下，参与者在等待超时后，依旧会执行事务提交，这样会导致数据的不一致。

Paxos协议

Paxos协议 解决了什么问题

像 2PC 和 3PC 都需要引入一个协调者的角色，当协调者 down 掉之后，整个事务都无法提交，参与者的资源都出于锁定的状态，对于系统的影响是灾难性的，而且出现网络分区的情况，很有可能会出现数据不一致的情况。有没有不需要协调者角色，每个参与者来协调事务呢，在网络分区的情况下，又能最大程度保证一致性的解决方案呢。此时 Paxos 出现了。

Paxos 算法是 Lamport 于 1990 年提出的一种基于消息传递的一致性算法。由于算法难以理解起初并没有引起人们的重视，Lamport在八年后重新发表，即便如此Paxos算法还是没有得到重视。2006 年 Google 的三篇论文石破天惊，其中的 chubby 锁服务使用Paxos 作为 chubbycell 中的一致性，后来才得到关注。

Paxos 协议是一个解决分布式系统中，多个节点之间就某个值（提案）达成一致（决议）的通信协议。它能够处理在少数节点离线的情况下，剩余的多数节点仍然能够达成一致。**即每个节点，既是参与者，也是决策者**

Paxos 协议的角色（可以是同一台机器）

> 由于 Paxos 和下文提到的 zookeeper 使用的 ZAB 协议过于相似，详细讲解参照下文， ZAB 协议部分

分布式系统中的节点通信存在两种模型：**共享内存（Shared memory）**和**消息传递（Messages passing）**。

基于消息传递通信模型的分布式系统，不可避免的会发生以下错误：进程可能会慢、被杀死或者重启，消息可能会延迟、丢失、重复，在基础Paxos场景中，先不考虑可能出现消息篡改,即拜占庭错误的情况。(网络环境一般为自建内网，消息安全相对高)

Paxos算法解决的问题是在一个可能发生上述异常的分布式系统中如何就某个值达成一致，保证不论发生以上任何异常，都不会破坏决议的一致性。

Paxos 协议的角色 主要有三类节点：

- 提议者（Proposer）：提议一个值；
- 接受者（Acceptor）：对每个提议进行投票；

- 告知者（Learner）：被告知投票的结果，不参与投票过程。

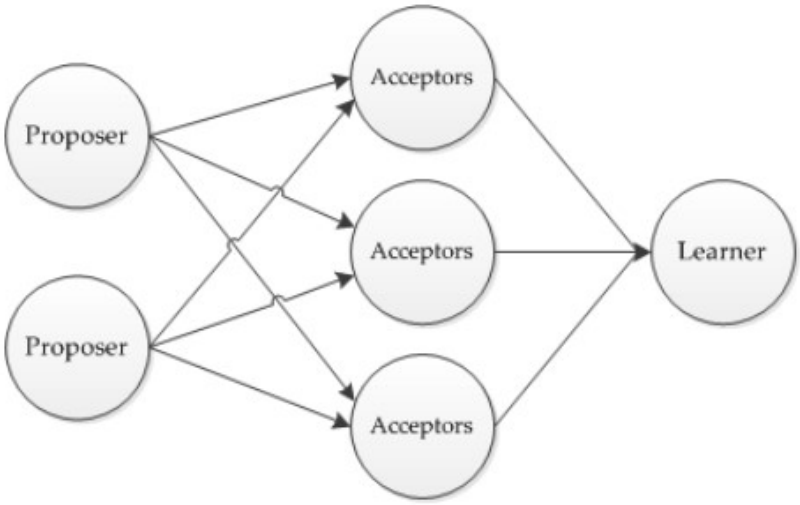


Figure 1: Basic Paxos architecture. A number of proposers make proposals to acceptors. When an acceptor accepts a value it sends the result to learner nodes.

过程：

规定一个提议包含两个字段：[n, v]，其中 n 为序号（具有唯一性），v 为提议值。

下图演示了两个 Proposer 和三个 Acceptor 的系统中运行该算法的初始过程，每个 Proposer 都会向所有 Acceptor 发送提议请求。

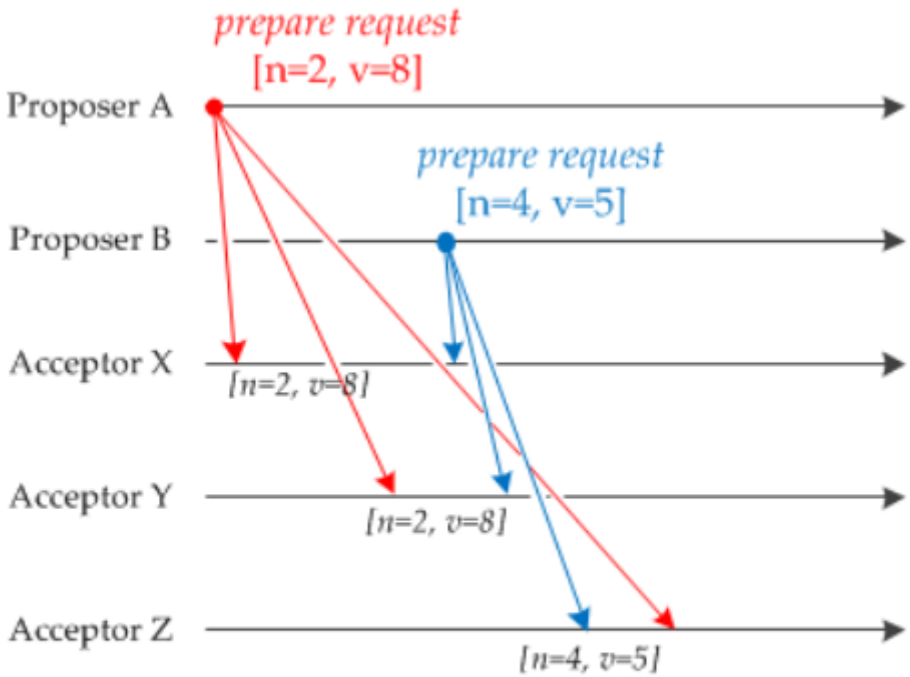


Figure 2: Paxos. Proposers A and B each send prepare requests to every acceptor. In this example proposer A's request reaches acceptors X and Y first, and proposer B's request reaches acceptor Z first.

当 Acceptor 接收到一个提议请求，包含的提议为 [n1, v1]，并且之前还未接收过提议请求，那么发送一个提议响应，设置当前接收到的提议为 [n1, v1]，并且保证以后不会再接受序号小于 n1 的提议。

如下图，Acceptor X 在收到 [n=2, v=8] 的提议请求时，由于之前没有接收过提议，因此就发送一个 [no previous] 的提议响应，并且设置当前接收到的提议为 [n=2, v=8]，并且保证以后不会再接受序号小于 2 的提议。其它的 Acceptor 类似。

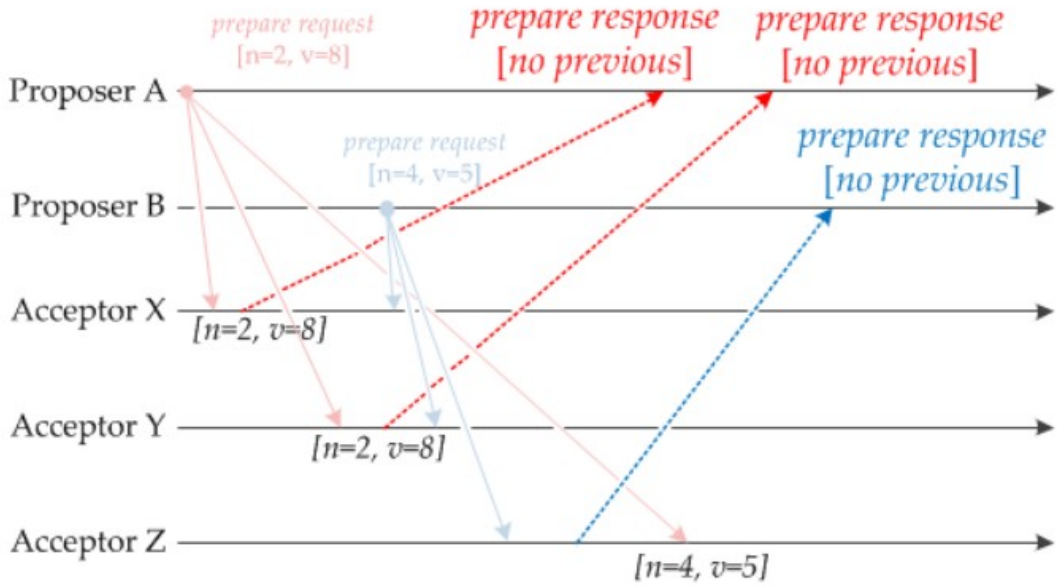


Figure 3: Paxos. Each acceptor responds to the first prepare request message that it receives.

如果 Acceptor 接受到一个提议请求，包含的提议为 [n2, v2]，并且之前已经接收过提议 [n1, v1]。如果 n1 > n2，那么就丢弃该提议请求；否则，发送提议响应，该提议响应包含之前已经接收过的提议 [n1, v1]，设置当前接收到的提议为 [n2, v2]，并且保证以后不会再接受序号小于 n2 的提议。

如下图，Acceptor Z 收到 Proposer A 发来的 [n=2, v=8] 的提议请求，由于之前已经接收过 [n=4, v=5] 的提议，并且 n > 2，因此就抛弃该提议请求；Acceptor X 收到 Proposer B 发来的 [n=4, v=5] 的提议请求，因为之前接收到的提议为 [n=2, v=8]，并且 2 <= 4，因此就发送 [n=2, v=8] 的提议响应，设置当前接收到的提议为 [n=4, v=5]，并且保证以后不会再接受序号小于 4 的提议。Acceptor Y 类似。

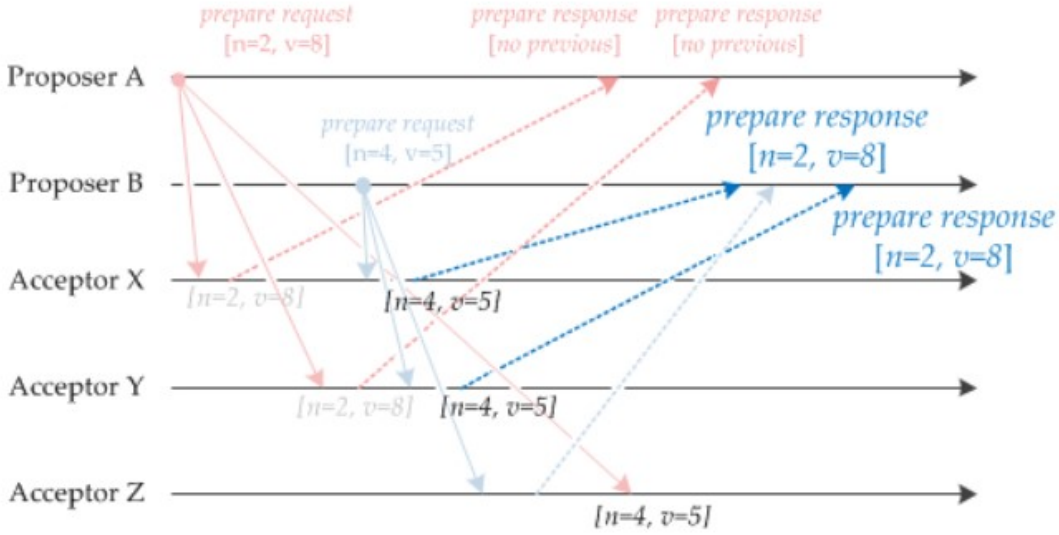


Figure 4: Paxos. Acceptor Z ignores proposer A's request because it has already seen a higher numbered proposal (4> 2). Acceptors X and Y respond to proposer B's request with the previous highest request that they acknowledged, and a promise to ignore any lower numbered proposals.

当一个 Proposer 接收到超过一半 Acceptor 的提议响应时，就可以发送接受请求。

Proposer A 接受到两个提议响应之后，就发送 [n=2, v=8] 接受请求。该接受请求会被所有 Acceptor 丢弃，因为此时所有 Acceptor 都保证不接受序号小于 4 的提议。

Proposer B 过后也收到了两个提议响应，因此也开始发送接受请求。需要注意的是，接受请求的 v 需要取它收到的最大 v 值，也就是 8。因此它发送 [n=4, v=8] 的接受请求。

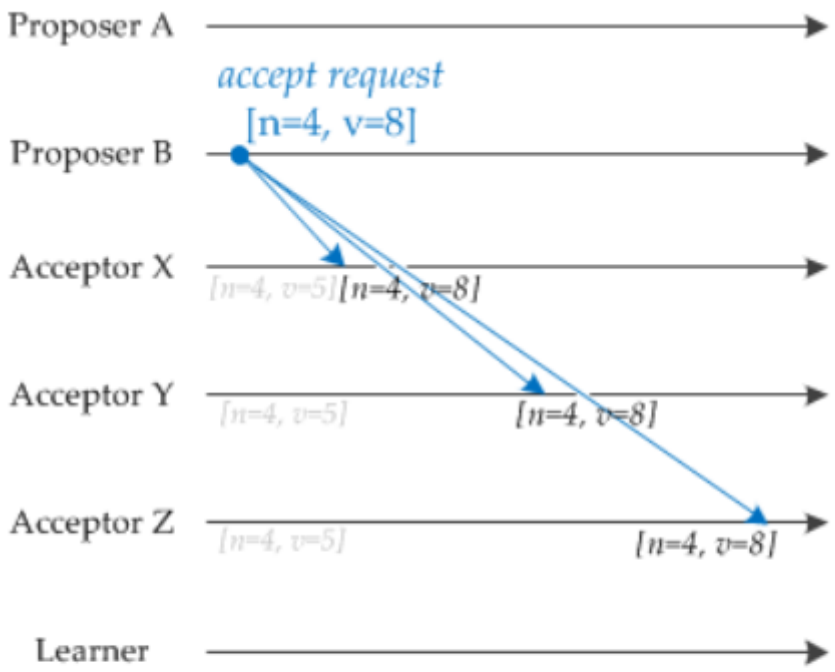
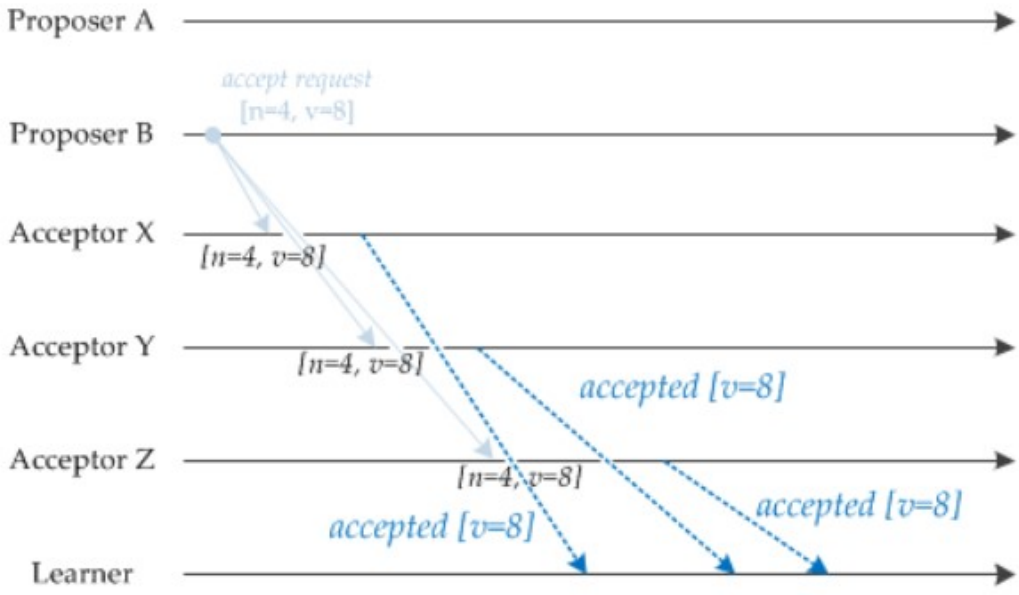


Figure 5: Paxos. Proposer B sends an accept request to each acceptor, with its previous proposal number (4), and the value of the highest numbered proposal it has seen (8, from [n=2, v=8

Acceptor 接收到接受请求时，如果序号大于等于该 Acceptor 承诺的最小序号，那么就发送通知给所有的 Learner。当 Learner 发现有大多数的 Acceptor 接收了某个提议，那么该提议的提议值就被 Paxos 选择出来。



Raft协议

Paxos 是论证了一致性协议的可行性，但是论证的过程据说晦涩难懂，缺少必要的实现细节，而且工程实现难度比较高广为人知实现只有 zk 的实现 zab 协议。

Paxos协议的出现为分布式强一致性提供了很好的理论基础，但是Paxos协议理解起来较为困难，实现比较复杂。

然后斯坦福大学RamCloud项目中提出了易实现，易理解的分布式一致性复制协议 Raft。Java，C++，Go 等都有其对应的实现

之后出现的Raft相对要简洁很多。
引入主节点，通过竞选。
节点类型：Follower、Candidate 和 Leader

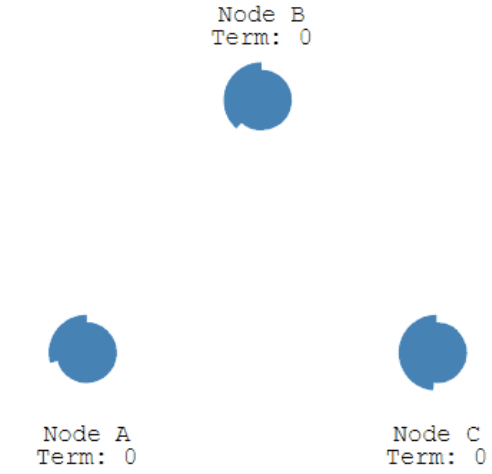
Leader 会周期性的发送心跳包给 Follower。每个 Follower 都设置了一个随机的竞选超时时间，一般为 150ms~300ms，如果在这个时间内没有收到 Leader 的心跳包，就会变成 Candidate，进入竞选阶段。

基本名词

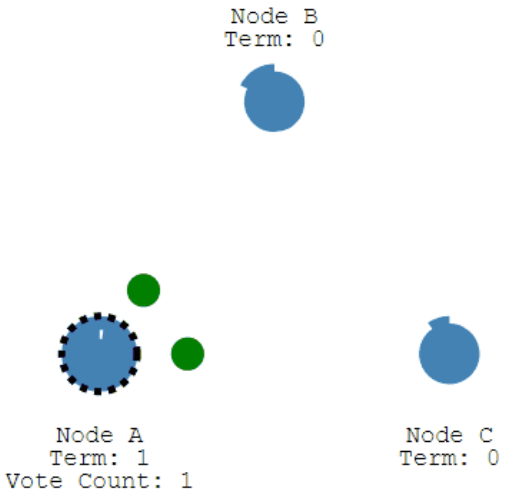
- 节点状态
  - Leader（主节点）：接受 client 更新请求，写入本地后，然后同步到其他副本中
  - Follower（从节点）：从 Leader 中接受更新请求，然后写入本地日志文件。对客户端提供读请求
  - Candidate（候选节点）：如果 follower 在一段时间内未收到 leader 心跳。则判断 leader 可能故障，发起选主提议。节点状态从 Follower 变为 Candidate 状态，直到选主结束
- termId：任期号，时间被划分成一个个任期，每次选举后都会产生一个新的 termId，一个任期内只有一个 leader。termId 相当于 paxos 的 proposalId。
- RequestVote：请求投票，candidate 在选举过程中发起，收到 quorum (多数派）响应后，成为 leader。
- AppendEntries：附加日志，leader 发送日志和心跳的机制
- election timeout：选举超时，如果 follower 在一段时间内没有收到任何消息(追加日志或者心跳)，就是选举超时。

竞选阶段流程

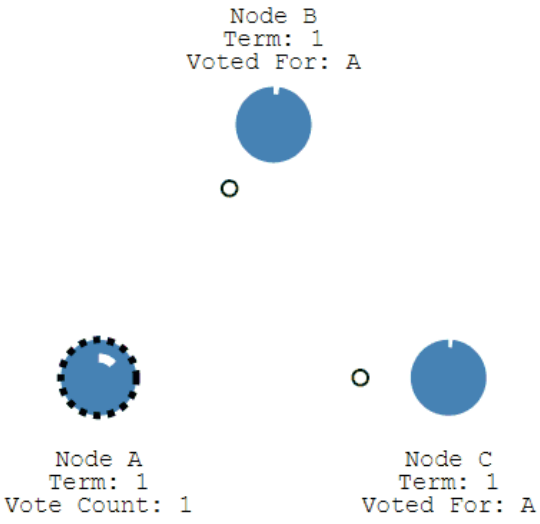
① 下图表示一个分布式系统的最初阶段，此时只有 Follower，没有 Leader。Follower A 等待一个随机的竞选超时时间之后，没收到 Leader 发来的心跳包，因此进入竞选阶段。



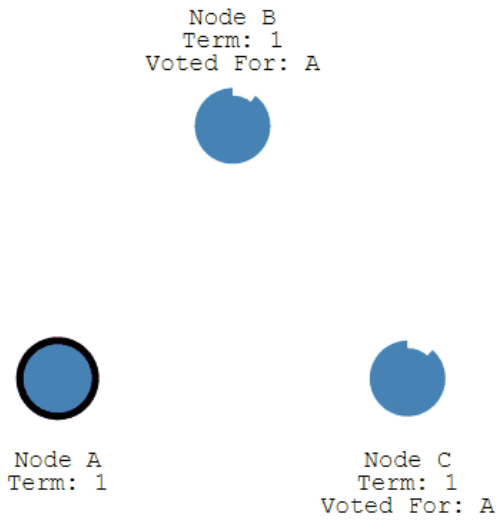
② 此时 A 发送投票请求给其它所有节点。



③ 其它节点会对请求进行回复，如果超过一半的节点回复了，那么该 Candidate 就会变成 Leader。

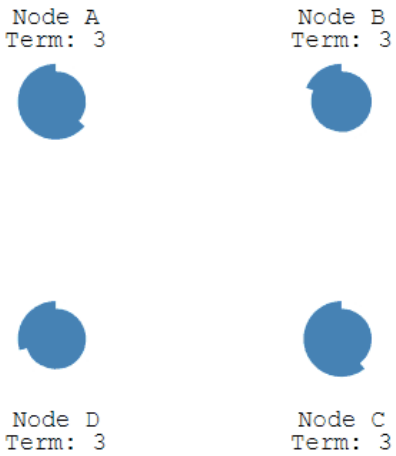


④ 之后 Leader 会周期性地发送心跳包给 Follower，Follower 接收到心跳包，会重新开始计时。



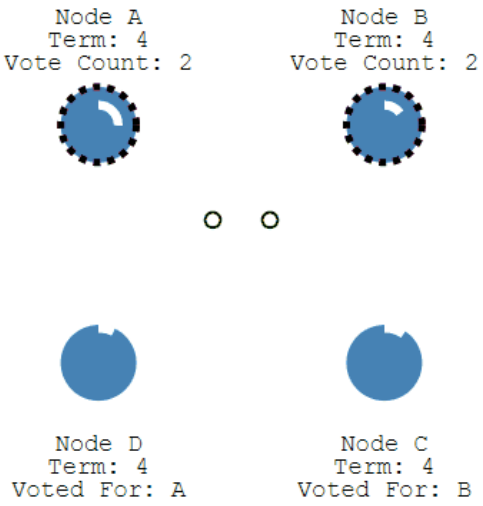
多个 Candidate 竞选

① 如果有多个 Follower 成为 Candidate，并且所获得票数相同，那么就需要重新开始投票，例如下图中 Candidate B 和 Candidate D 都获得两票，因此需要重新开始投票。



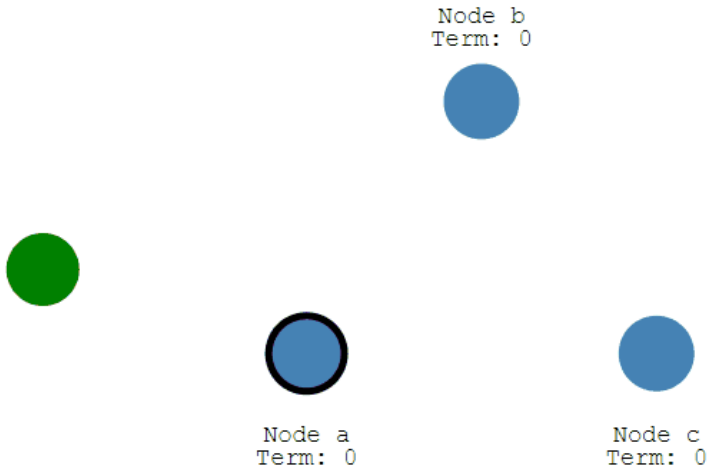
② 当重新开始投票时，由于每个节点设置的随机竞选超时时间不同，因此能下一次再次出现多个

Candidate 并获得同样票数的概率很低。

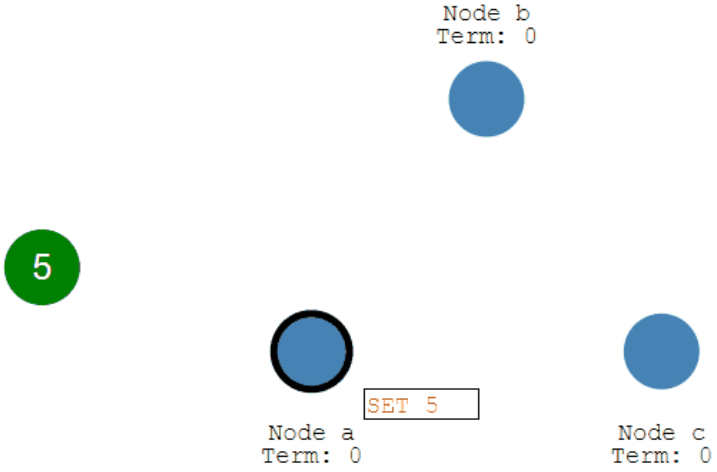


日志复制

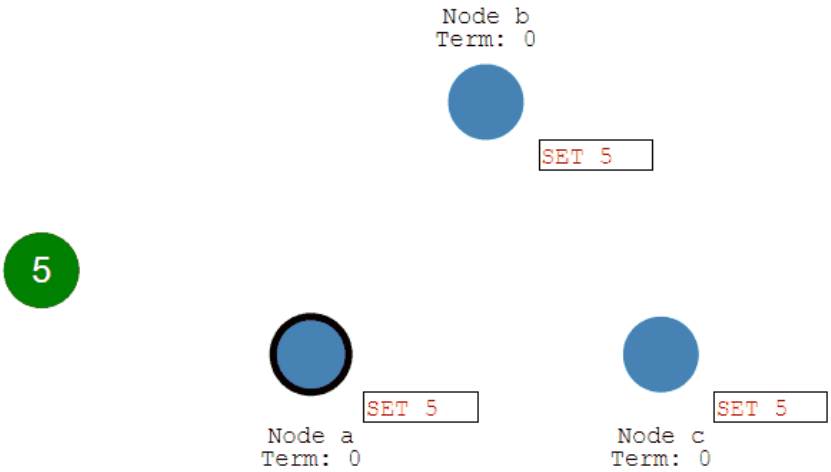
① 来自客户端的修改都会被传入 Leader。注意该修改还未被提交，只是写入日志中。



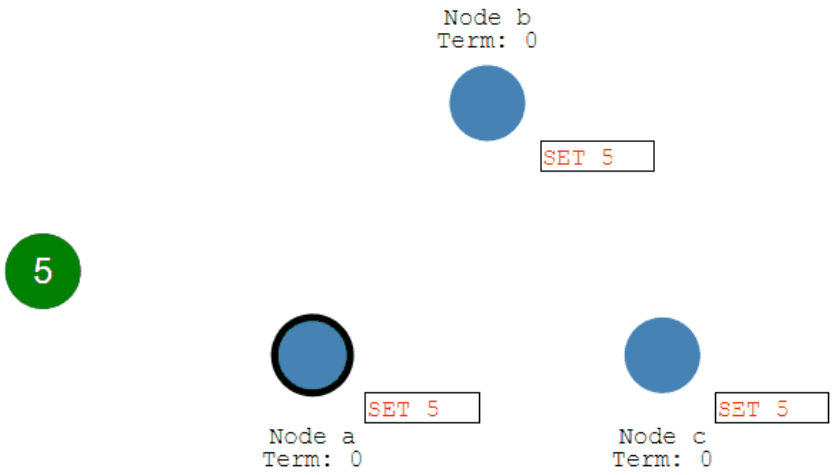
② Leader 会把修改复制到所有 Follower。



③ Leader 会等待大多数的 Follower 也进行了修改，然后才将修改提交。



④ 此时 Leader 会通知的所有 Follower 让它们也提交修改，此时所有节点的值达成一致。



ZAB协议

ZAB协议 概述

Google 的粗粒度锁服务 Chubby 的设计开发者 Burrows 曾经说过：“所有一致性协议本质上要么是 Paxos 要么是其变体”。Paxos 虽然解决了分布式系统中，多个节点就某个值达成一致性的通信协议。但是还是引入了其他的问题。由于其每个节点，都可以提议提案，也可以批准提案。当有三个及以上的 proposer 在发送 prepare 请求后，很难有一个 proposer 收到半数以上的回复而不断地执行第一阶段的协议，**在这种竞争下，会导致选举速度变慢。**

所以 zookeeper 在 paxos 的基础上，提出了 ZAB 协议，本质上是，只有一台机器能提议提案（Proposer），而这台机器的名称称之为 Leader 角色。其他参与者扮演 Acceptor 角色。为了保证 Leader 的健壮性，引入了 Leader 选举机制。

ZAB协议还解决了这些问题

1. 在半数以下节点宕机，依然能对台提供服务
2. 客户端所有的写请求，交由 Leader 来处理。写入成功后，需要同步给所有的 follower 和 observer
3. leader 宕机，或者集群重启。需要确保已经再 Leader 提交的事务最终都能被服务器提交，并且确保集群能快速回复到故障前的状态

ZAB协议 基本概念

- 基本名词
  - 数据节点（dataNode）：zk 数据模型中的最小数据单元，数据模型是一棵树，由斜杠（/）分割的路径名唯一标识，数据节点可以存储数据内容及一系列属性信息，同时还可以挂载子节点，构成一个层次化的命名空间。
  - 事务及 zxid：事务是指能够改变 Zookeeper 服务器状态的操作，一般包括数据节点的创建与删除、数据节点内容更新和客户端会话创建与失效等操作。对于每个事务请求，zk 都会为其分配一个全局唯一的事务 ID，即 zxid，是一个 64 位的数字，高 32 位表示该事务发生的集群选举周期（集群每发生一次 leader 选举，值加 1），低 32 位表示该事务在当前选择周期

内的递增次序（leader 每处理一个事务请求，值加 1，发生一次 leader 选择，低 32 位要清 0）。
    - 事务日志：所有事务操作都是需要记录到日志文件中的，可通过 dataLogDir 配置文件目录，文件是以写入的第一条事务 zxid 为后缀，方便后续的定位查找。zk 会采取“磁盘空间预分配”的策略，来避免磁盘 Seek 频率，提升 zk 服务器对事务请求的影响能力。默认设置下，每次事务日志写入操作都会实时刷入磁盘，也可以设置成非实时（写到内存文件流，定时批量写入磁盘），但那样断电时会带来丢失数据的风险。
    - 事务快照：数据快照是 zk 数据存储中另一个非常核心的运行机制。数据快照用来记录 zk 服务器上某一时刻的全量内存数据内容，并将其写入到指定的磁盘文件中，可通过 dataDir 配置文件目录。可配置参数 snapCount，设置两次快照之间的事务操作个数，zk 节点记录完事务日志时，会统计判断是否需要做数据快照（距离上次快照，事务操作次数等于 snapCount/2~snapCount 中的某个值时，会触发快照生成操作，随机值是为了避免所有节点同时生成快照，导致集群影响缓慢）。
- 核心角色
    - leader：系统刚启动时或者 Leader 崩溃后正处于选举状态；
    - follower：Follower 节点所处的状态，Follower 与 Leader 处于数据同步阶段；
    - observer：Leader 所处状态，当前集群中有一个 Leader 为主进程。
- 节点状态
    - LOOKING：节点正处于选主状态，不对外提供服务，直至选主结束；
    - FOLLOWING：作为系统的从节点，接受主节点的更新并写入本地日志；
    - LEADING：作为系统主节点，接受客户端更新，写入本地日志并复制到从节点

ZAB协议 常见的误区

- 写入节点后的数据，立马就能被读到，这是错误的。** zk 写入是必须通过 leader 串行的写入，而且只要一半以上的节点写入成功即可。而任何节点都可提供读取服务**。例如：zk，有 1~5 个节点，写入了一个最新的数据，最新数据写入到节点 1~3，会返回成功。然后读取请求过来要读取最新的节点数据，请求可能被分配到节点 4~5 。而此时最新数据还没有同步到节点4~5。会读取不到最近的数据。**如果想要读取到最新的数据，可以在读取前使用 sync 命令**。
- zk启动节点不能偶数台，这也是错误的。zk 是需要一半以上节点才能正常工作的。例如创建 4 个节点，半数以上正常节点数是 3。也就是最多只允许一台机器 down 掉。而 3 台节点，半数以上正常节点数是 2，也是最多允许一台机器 down 掉。4 个节点，多了一台机器的成本，但是健壮性和 3 个节点的集群一样。基于成本的考虑是不推荐的

ZAB协议 选举同步过程

发起投票的契机

1. 节点启动
2. 节点运行期间无法与 Leader 保持连接，
3. Leader 失去一半以上节点的连接

如何保证事务

ZAB 协议类似于两阶段提交，客户端有一个写请求过来，例如设置 `/my/test` 值为 1，Leader 会生成对应的事务提议（proposal）（当前 zxid为 0x5000010 提议的 zxid 为Ox5000011），现将`set /my/test 1`（此处为伪代码）写入本地事务日志，然后`set /my/test 1`日志同步到所有的 follower。follower收到事务 proposal ，将 proposal 写入到事务日志。如果收到半数以上 follower 的回应，那么广播发起 commit 请求。follower 收到 commit 请求后。会将文件中的 zxid ox5000011 应用到内存中。

上面说的是正常的情况。有两种情况。第一种 Leader 写入本地事务日志后，没有发送同步请求，就 down 了。即使选主之后又作为 follower 启动。此时这种还是会日志会丢掉（原因是选出的 leader 无此日志，无法进行同步）。第二种 Leader 发出同步请求，但是还没有 commit 就 down 了。此时这个日志不会丢掉，会同步提交到其他节点中。

服务器启动过程中的投票过程

现在 5 台 zk 机器依次编号 1~5

1. 节点 1 启动，发出去的请求没有响应，此时是 Looking 的状态
2. 节点 2 启动，与节点 1 进行通信，交换选举结果。由于两者没有历史数据，即 zxid 无法比较，此时 id 值较大的节点 2 胜出，但是由于还没有超过半数的节点，所以 1 和 2 都保持 looking 的状态
3. 节点 3 启动，根据上面的分析，id 值最大的节点 3 胜出，而且超过半数的节点都参与了选举。节点 3 胜出成为了 Leader
4. 节点 4 启动，和 1~3 个节点通信，得知最新的 leader 为节点 3，而此时 zxid 也小于节点 3，所以承认了节点 3 的 leader 的角色
5. 节点 5 启动，和节点 4 一样，选取承认节点 3 的 leader 的角色

服务器运行过程中选主过程

1.节点 1 发起投票，**第一轮投票先投自己**，然后进入 Looking 等待的状态 2.其他的节点（如节点 2 ）收到对方的投票信息。节点 2 在 Looking 状态，则将自己的投票结果广播出去（此时走的是上图中左侧的 Looking 分支）；如果不在 Looking 状态，则直接告诉节点 1 当前的 Leader 是谁，就不要瞎折腾选举了（此时走的是上图右侧的 Leading/following 分支） 3.此时节点 1，收到了节点 2 的选举结果。如果节点 2 的 zxid 更大，那么清空投票箱，建立新的投票箱，广播自己最新的投票结果。在同一次选举中，如果在收到所有节点的投票结果后，如果投票箱中有一半以上的节点选出了某个节点，那么证明 leader 已经选出来了，投票也就终止了。否则一直循环

zookeeper 的选举，优先比较大 zxid，zxid 最大的节点代表拥有最新的数据。如果没有 zxid，如系统刚刚启动的时候，则比较机器的编号，优先选择编号大的

同步的过程

在选出 Leader 之后，zk 就进入状态同步的过程。其实就是把最新的 zxid 对应的日志数据，应用到其他的节点中。此 zxid 包含 follower 中写入日志但是未提交的 zxid 。称之为服务器提议缓存队列 committedLog 中的 zxid。

同步会完成三个 zxid 值的初始化。

`peerLastZxid`：该 learner 服务器最后处理的 zxid。 `minCommittedLog`：leader服务器提议缓存队列 committedLog 中的最小 zxid。 `maxCommittedLog`：leader服务器提议缓存队列 committedLog 中的最大 zxid。 系统会根据 learner 的`peerLastZxid`和 leader 的`minCommittedLog`，`maxCommittedLog`做出比较后做出不同的同步策略

直接差异化同步

场景：`peerLastZxid`介于`minCommittedLogZxid`和`maxCommittedLogZxid`间

此种场景出现在，上文提到过的，Leader 发出了同步请求，但是还没有 commit 就 down 了。 leader 会发送 Proposal 数据包，以及 commit 指令数据包。新选出的 leader 继续完成上一任 leader 未完成的工作。

例如此刻Leader提议的缓存队列为 0x20001，0x20002，0x20003，0x20004，此处learn的peerLastZxid为0x20002，Leader会将0x20003和0x20004两个提议同步给learner

先回滚在差异化同步/仅回滚同步

此种场景出现在，上文提到过的，Leader写入本地事务日志后，还没发出同步请求，就down了，然后在同步日志的时候作为learner出现。

例如即将要 down 掉的 leader 节点 1，已经处理了 0x20001，0x20002，在处理 0x20003 时还没发出提议就 down 了。后来节点 2 当选为新 leader，同步数据的时候，节点 1 又神奇复活。如果新 leader 还没有处理新事务，新 leader 的队列为，0x20001, 0x20002，那么仅让节点 1 回滚到 0x20002 节点处，0x20003 日志废弃，称之为仅回滚同步。如果新 leader 已经处理 0x30001 , 0x30002 事务，那么新 leader 此处队列为0x20001，0x20002，0x30001，0x30002，那么让节点 1 先回滚，到 0x20002 处，再差异化同步0x30001，0x30002。

全量同步

`peerLastZxid`小于`minCommittedLogZxid`或者leader上面没有缓存队列。leader直接使用SNAP命令进行全量同步

参考文献：

https://www.cnblogs.com/zhang-qc/p/8688258.html

https://blog.csdn.net/weixin_33725272/article/details/87947998

http://ifeve.com/raft/

硬核推荐：尼恩Java硬核架构班

又名疯狂创客圈社群 VIP

详情：

https://www.cnblogs.com/crazymakercircle/p/9904544.html

架构班（社群 VIP）的起源：

最初的视频，主要是给读者加餐。很多的读者，需要一些高质量的实操、理论视频，所以，我就围绕书，和底层，做了几个实操、理论视频，然后效果还不错，后面就做成迭代模式了。

架构班（社群 VIP）的功能：

提供高质量实操项目整刀真枪的架构指导、快速提升大家的:

- 开发水平
- 设计水平
- 架构水平

弥补业务中 CRUD 开发短板，帮助大家尽早脱离具备 3 高能力，掌握：

- 高性能
- 高并发
- 高可用

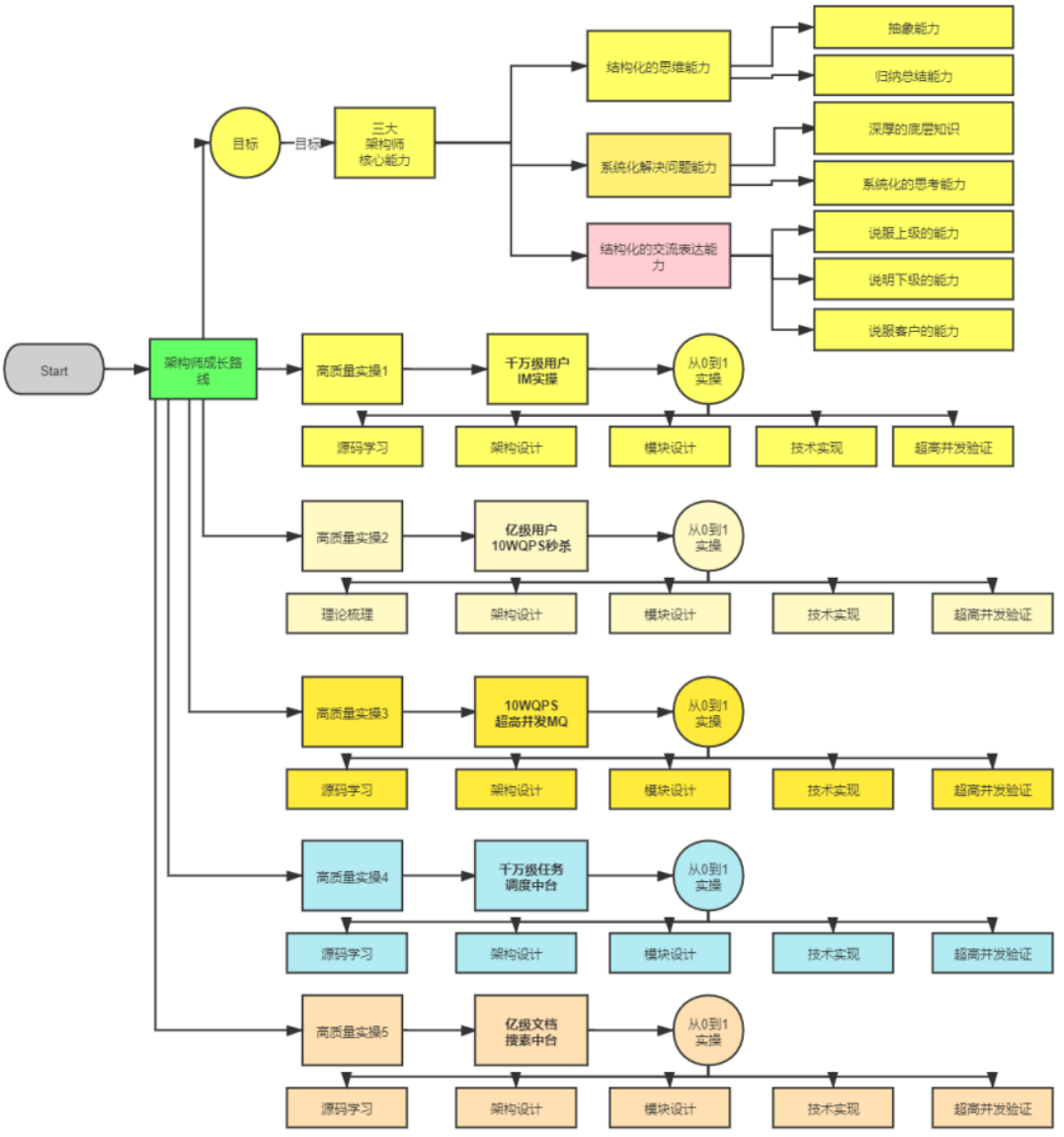
作为一个高质量的架构师成长、人脉社群，把所有的卷王聚焦起来，一起卷：

- 卷高并发实操
- 卷底层原理
- 卷架构理论、架构哲学
- 最终成为顶级架构师，实现人生理想，走向人生巅峰

架构班（社群 VIP）的目的：

- 高质量的实操，大大提升简历的含金量，吸引力，增强面试的召唤率
- 为大家提供九阳真经、葵花宝典，快速提升水平
- 进大厂、拿高薪
- 一路陪伴，提供助学视频和指导，辅导大家成为架构师
- 自学为主，和其他卷王一起，卷高并发实操，卷底层原理、卷大厂面试题，争取狠卷 3 月成高手，狠卷 3 年成为顶级架构师

N 个超高并发实操项目：简历压轴、个顶个精彩



【样章】第 17 章:横扫全网 Rocketmq 视频第 2 部曲: 工业级 rocketmq 高可用(HA)底层原理和实操

工业级 rocketmq 高可用底层原理，包含：消息消费、同步消息、异步消息、单向消息等不同消息的底层原理和源码实现；消息队列非常底层的主从复制、高可用、同步刷盘、异步刷盘等底层原理。

工业级 rocketmq 高可用底层原理和搭建实操，包含：高可用集群的搭建。

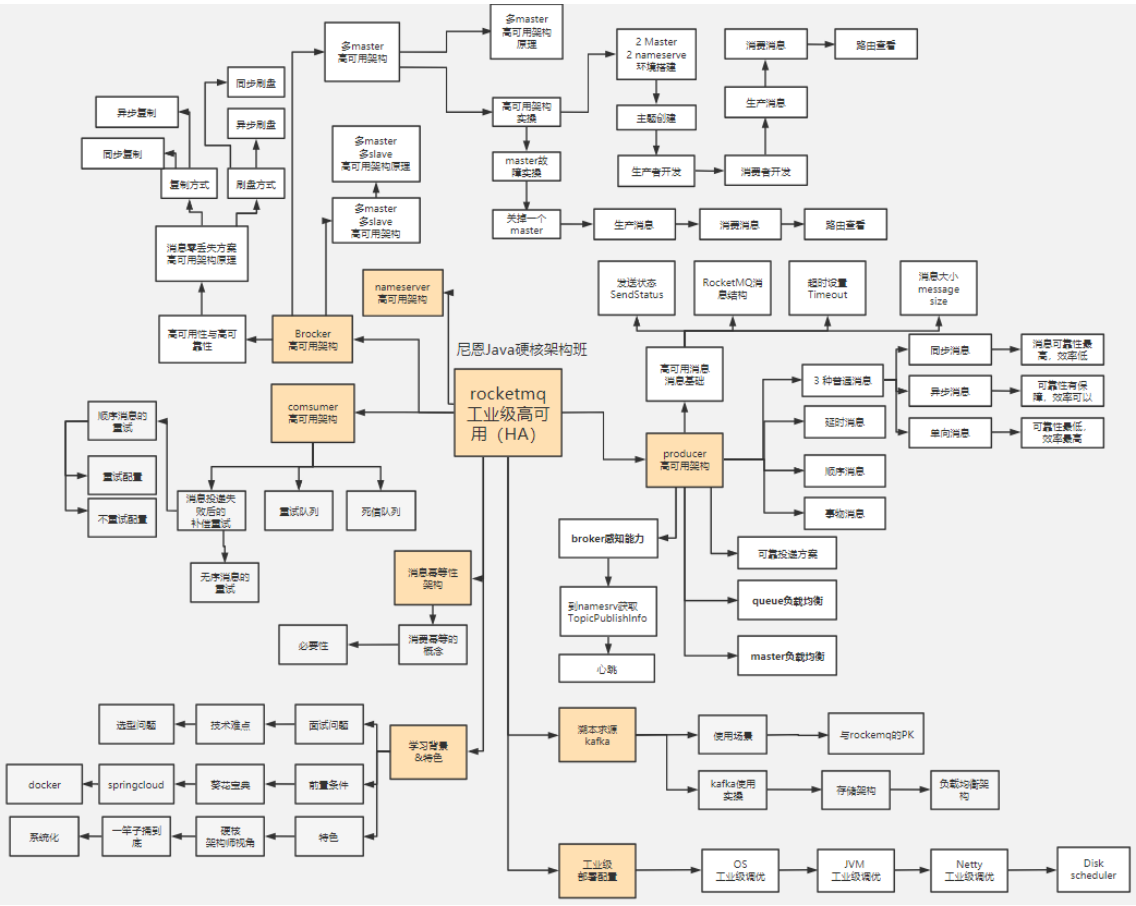
解决以下难题：

1、技术难题：RocketMQ 如何最大限度的保证消息不丢失的呢？RocketMQ 消息如何做到高可靠投递？

2、技术难题：基于消息的分布式事务，核心原理不理解

3、选型难题： kafka or rocketmq ，该娶谁？

下图链接：https://www.processon.com/view/6178e8ae0e3e7416bde9da19



成功案例：2 年翻 3 倍，35 岁卷王成功转型为架构师

详情：http://topcoder.cloud/forum.php?mod=forumdisplay&fid=43&page=1

最新 最后发表 热门 精华

- 成功案例：[1057号卷王] 3年小伙拿到外企offer，薪酬涨了200% — 卷王1号 超级版主 前天 17:41
- 成功案例：[645号卷王] 4年经验卷王逆袭，被毕业后，反涨24W — 卷王1号 超级版主 2022-9-21
- 成功案例：[878号卷王] 小伙8年经验，年薪60W — 卷王1号 超级版主 2022-8-13
- 年薪70W案例：通过尼恩的指导，小伙伴年薪从40W涨到70W — 卷王1号 超级版主 2022-2-11
- 成功案例：[493号卷王] 5年小伙拿满意offer，就业寒冬季逆涨30% — 卷王1号 超级版主 前天 17:43
- 成功案例：[250号卷王] 就业极寒时代，收offer 涨25% — 卷王1号 超级版主 前天 17:38
- 成功案例：[612号卷王] 就业极寒时代，从外包到自研 — 卷王1号 超级版主 前天 17:15
- 成功案例：[913号卷王] 热烈祝贺6年经验卷王，年薪40W — 卷王1号 超级版主 2022-9-21
- 成功案例：[959号卷王] 4年经验卷王，喜获百度、Boss直聘等N个优质offer，最高涨100% — 卷王1号 超级版主 2022-9-21
- 成功案例：[529号卷王] 5年经验卷王喜收2大offer，最高涨5K — 卷王1号 超级版主 2022-9-21
- 成功案例：[811号卷王] 热烈祝贺7年经验卷王，薪酬涨30% — 卷王1号 超级版主 2022-9-21
- 成功案例：[287号卷王] 不惧大寒潮，卷王逆市收4 offer，涨30%，可喜可贺 — 卷王1号 超级版主 2022-5-30
- 成功案例：[1002号卷王] 5月份"被毕业"，改简历后，斩获顶级央企Offer，涨薪7000+ — 卷王1号 超级版主 2022-7-5
- 成功案例：[7号卷王] 热烈祝贺小伙伴涨薪120% — 卷王1号 超级版主 2022-8-13
- 成功案例：[134号卷王] 大三小伙卷1年，斩获顶级央企Offer，成功逆袭 — 卷王1号 超级版主 2022-7-6
- 成功案例：[1008号卷王] 5年经验卷王收42W offer，月涨8000，可喜可贺 — 卷王1号 超级版主 2022-5-30
- 成功案例：[453号卷王] 非全日制 6年卷王喜提3 offer，年薪30W，可喜可贺 — 卷王1号 超级版主 2022-5-21
- 成功案例：[924号卷王] 6年卷王喜提4 offer，最高涨薪9000，可喜可贺 — 卷王1号 超级版主 2022-5-21
- 成功案例：[15号卷王] 4年卷王入职 微软，涨薪50%，可喜可贺 — 卷王1号 超级版主 2022-5-12
- 成功案例：[527号卷王] 4年卷王喜提2 offer，涨薪50%，可喜可贺 — 卷王1号 超级版主 2022-5-13
- 成功案例：[788号卷王] 3年卷王喜提优质Offer，涨薪60% — 卷王1号 超级版主 2022-5-11
- 成功案例：热烈祝贺：非全日制卷王，喜提2个心仪offer，面3家过2家 — 卷王1号 超级版主 2022-4-21

最新 最后发表 热门 精华

- 成功案例：[693号卷王] 二线城市6年卷王喜提4大优质Offer，含央企offer，最高薪酬35W — 卷王1号 超级版主 2022-4-16
- 成功案例：[85号卷王] 双非2本小伙，春招大捷，喜提9个offer，最高薪酬近30万 — 卷王1号 超级版主 2022-4-14
- 成功案例：[741号卷王] 卷王逆袭！6年小伙从很少面试机会到搞定35K*14薪Offer — 卷王1号 超级版主 2022-4-12
- 成功案例：[642号卷王] 热烈祝贺，6年卷王喜提优质国企offer — 卷王1号 超级版主 2022-4-7
- 成功案例：[796号卷王] 热烈祝贺，36岁卷王喜提52万优质offer — 卷王1号 超级版主 2022-3-25
- 成功案例：[15号卷王] 小伙卷1年，涨薪9K+，喜收ebay等多个优质offer — 卷王1号 超级版主 2022-3-24
- 成功案例：[821号卷王] 小伙狠卷3个月，喜提10多个offer — 卷王1号 超级版主 2022-3-21
- 成功案例：[736号卷王] 3年半经验收22k offer，但是小伙志存高远，冲击25k+ — 卷王1号 超级版主 2022-3-20
- 成功案例：热烈祝贺1群小卷王offer拿到手软，甚至拒了阿里offer — 卷王1号 超级版主 2022-3-16
- 简历案例：简历一改，腾讯的邀请就来了！热烈祝贺，小伙收到一大堆面试邀请 — 卷王1号 超级版主 2022-3-10
- 成功案例：祝贺我圈两大超级卷王，一个过了阿里HR面，一个过了阿里2面 — 卷王1号 超级版主 2022-3-10
- 成功案例：小伙伴php转Java，卷1.5年Java，涨薪50%，喜收多个优质offer — 卷王1号 超级版主 2022-3-10
- 成功案例：4年小伙狠卷半年，拿到 移动、京东 两大顶级offer — 尼恩 超级版主 2022-3-5
- 成功案例：[267号卷王] 助力3年经验卷王，拿到蜂巢的17k x 14薪的offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[143号卷王] 二本院校00后卷神，毕业没到一年跳到宇节，年薪45W — 卷王1号 超级版主 2022-2-27
- 成功案例：[494号卷王] 尼恩分布式事务助力卷王拿到 中信银行offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[76号卷王] 2线城市卷王，狠卷1.5年，喜收22K offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[429号卷王] 小伙伴在社群卷5个月，涨8k+ — 卷王1号 超级版主 2022-2-27
- 成功案例：[154号卷王] 双非学校毕业卷王，连拿 京东到家&滴滴 两个大厂Offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[232号卷王] 涨薪10K，继续卷向食物链顶端 — 卷王1号 超级版主 2022-2-27
- 成功案例：狠卷1年技术，喜收 腾讯、阿里、微软三大Offer，最高年薪56W — 卷王1号 超级版主 2022-2-27
- 成功案例：[449号卷王] 应届毕业卷王喜收 滴滴offer，年薪33W — 卷王1号 超级版主 2022-2-27

- 成功案例：[732号卷王] 尼恩助力3年经验卷王收获 京东offer，年薪35W — 卷王1号 超级版主 2022-2-27
- 成功案例：[558号卷王] 2年经验卷王，喜收 网易和阿里子公司两个优质offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[569号卷王] 双非应届生卷王，喜收字节跳动实习offer — 卷王1号 超级版主 2022-2-25
- 成功案例：[420号卷王] 狠卷1年，卷王涨薪80%，涨薪12000元！ — 卷王1号 超级版主 2022-2-25
- 成功案例：[76号卷王] 通过尼恩1年半的指导，专科学历小伙伴从0.8K涨到22K — 卷王1号 超级版主 2022-2-10

- 成功案例：[551号卷王] 小伙伴学完后，成功进入大厂，并且推荐自己的朋友加VIP学习 — 卷王1号 超级版主 2022-2-10
- 成功案例：[214号卷王] 助力2年经验卷王，成功拿到17K月薪 — 卷王1号 超级版主 2022-2-10
- 成功案例：[92号卷王] 课程实操助力社群小伙伴喜收 喜马拉雅Offer — 卷王1号 超级版主 2022-2-10
- 成功案例：社群卷王小伙伴成功过了滴滴三面 获滴滴Offer — 卷王1号 超级版主 2022-2-10
- [612号卷王]滴滴小伙伴，蹲点考察半年，觉得靠谱后加入 疯狂创客圈 — 卷王1号 超级版主 2022-2-10

简历优化后的成功涨薪案例（VIP 含免费简历优化）

修改简历找尼恩（资深简历优化专家）

- 如果面试表达不好，尼恩会提供 简历优化指导
- 如果项目没有亮点，尼恩会提供 项目亮点指导
- 如果面试表达不好，尼恩会提供 面试表达指导

作为 40 岁老架构师，尼恩长期承担技术面试官的角色：

- **从业以来，“阅历”无数，对简历有着点石成金、改头换面、脱胎换骨**的指导能力。
- **尼恩指导过刚刚就业的小白，也指导过 P8 级的老专家，都指导他们上岸。**

如何联系尼恩。尼恩微信，请参考下面的地址：

语雀：https://www.yuque.com/crazymakercircle/gkkw8s/khigna
码云：https://gitee.com/crazymaker/SimpleCrayIM/blob/master/疯狂创客圈总目录.md