# 分布式一致性协议

# 先是给所有人发一个proposal,半数成功之后就是给所有人发commit

zookeeper用的是zab协议而不是paxos算法.

## paxos算法

<https://www.zhihu.com/question/19787937>

<https://zhuanlan.zhihu.com/p/31780743>

Proposer: 提出提案 (Proposal)。Proposal信息包括提案编号 (Proposal ID) 和提议的值 (Value)。

Acceptor：参与决策，回应Proposers的提案。收到Proposal后可以接受提案，若Proposal获得多数Acceptors的接受，则称该Proposal被批准。

Learner：不参与决策，从Proposers/Acceptors学习最新达成一致的提案（Value）

第一阶段：Prepare阶段。Proposer向Acceptors发出Prepare请求，Acceptors针对收到的Prepare请求进行Promise承诺。

第二阶段：Accept阶段。Proposer收到多数Acceptors承诺的Promise后，向Acceptors发出Propose请求，Acceptors针对收到的Propose请求进行Accept处理。

第三阶段：Learn阶段。Proposer在收到多数Acceptors的Accept之后，标志着本次Accept成功，决议形成，将形成的决议发送给所有Learners。

谁的proposal大谁吊

Prepare: Proposer生成全局唯一且递增的Proposal ID (可使用时间戳加Server ID)，向所有Acceptors发送Prepare请求，这里无需携带提案内容，只携带Proposal ID即可。

Promise: Acceptors收到Prepare请求后，做出“两个承诺，一个应答”。

两个承诺：

1. 不再接受Proposal ID小于等于（注意：这里是<= ）当前请求的Prepare请求。

2. 不再接受Proposal ID小于（注意：这里是< ）当前请求的Propose请求。

一个应答：

不违背以前作出的承诺下，回复已经Accept过的提案中Proposal ID最大的那个提案的Value和Proposal ID，没有则返回空值。

Propose: Proposer 收到多数Acceptors的Promise应答后，从应答中选择Proposal ID最大的提案的Value，作为本次要发起的提案。如果所有应答的提案Value均为空值，则可以自己随意决定提案Value。然后携带当前Proposal ID，向所有Acceptors发送Propose请求。

Accept: Acceptor收到Propose请求后，在不违背自己之前作出的承诺下，接受并持久化当前Proposal ID和提案Value。

Learn: Proposer收到多数Acceptors的Accept后，决议形成，将形成的决议发送给所有Learners。

1 获取一个Proposal ID n，为了保证Proposal ID唯一，可采用时间戳+Server ID生成；

Proposer向所有Acceptors广播Prepare(n)请求；

2 Acceptor比较n和minProposal，如果n>minProposal，minProposal=n，并且将 acceptedProposal 和 acceptedValue 返回；

3 Proposer接收到过半数回复后，如果发现有acceptedValue返回，将所有回复中acceptedProposal最大的acceptedValue作为本次提案的value，否则可以任意决定本次提案的value；

4 到这里可以进入第二阶段，广播Accept (n,value) 到所有节点；

Acceptor比较n和minProposal，如果n>=minProposal，则acceptedProposal=minProposal=n，acceptedValue=value，本地持久化后，返回；否则，返回minProposal。

提议者接收到过半数请求后，如果发现有返回值result >n，表示有更新的提议，跳转到1；否则value达成一致。

一是promser收到prepare的返回后,返现有value了,就会取pid最大的value当作提议value

这是一点.

二是在accept的返回后,发现accept又返回了更大的pid,(说明在我走accept的时候又有人给他发prepare了)那么就再次回到1,并且携带这个大的value.

反正就是发现还有别人,别人的pid还比我大,我就信别人的,这样就能尽快的大家确定一个方案出来.

生米煮成熟饭的意思是我的accpet的响应都被proposer收到了,你再来发prepare就没用了.

## raft协议

Leader：接受客户端请求，并向Follower同步请求日志，当日志同步到大多数节点上后告诉Follower提交日志。

Follower：接受并持久化Leader同步的日志，在Leader告之日志可以提交之后，提交日志。

Candidate：Leader选举过程中的临时角色。

可以看出,消息广播的时候也是leader先告诉follower,收到大部分反馈的时候,触发后面的commit消息,follower提交.

也会有选举之类的

Paxos和raft都是一旦一个entries（raft协议叫日志，paxos叫提案，叫法而已）得到多数派的赞成，这个entries就会定下来，不丢失，值不更改，最终所有节点都会赞成它。Paxos中称为提案被决定，Raft,ZAB,VR称为日志被提交，这只是说法问题**。一个日志一旦被提交(或者决定），就不会丢失，也不可能更改，这一点这4个协议都是一致的**。**Multi-paxos和Raft都用一个数字来标识leader的合法性**，multi-paxos中叫proposer-id，Raft叫term，意义是一样的，**multi-paxos proposer-id最大的Leader提出的决议才是有效的，raft协议中term最大的leader才是合法的**。

## Consul raft产物

基于raft算法.

单独的进程,下载安装.

可以配置consul集群.

服务注册就是把自己的psm和ip端口注册上去呗,访问consul的接口.

## Etcd raft产物

etcd 是一个高可用的 Key/Value 存储系统，主要用于分享配置和服务发现。etcd 的灵感来自于 ZooKeeper 和 Doozer，侧重于：

简单：支持 curl 方式的用户 API (HTTP+JSON)

安全：可选 SSL 客户端证书认证

快速：单实例可达每秒 10000 次写操作

可靠：使用 Raft 实现分布式

golang写的.

## zab协议

zookeeper用的.

Zab协议是为分布式协调服务Zookeeper专门设计的一种 支持崩溃恢复 的 原子广播协议 ，是Zookeeper保证数据一致性的核心算法。Zab借鉴了Paxos算法，但又不像Paxos那样，是一种通用的分布式一致性算法。它是特别为Zookeeper设计的支持崩溃恢复的原子广播协议。

在Zookeeper中主要依赖Zab协议来实现数据一致性，基于该协议，zk实现了一种主备模型（即Leader和Follower模型）的系统架构来保证集群中各个副本之间数据的一致性。  
这里的主备系统架构模型，就是指只有一台客户端（Leader）负责处理外部的写事务请求，然后Leader客户端将数据同步到其他Follower节点。

不就主从读写分离么

· 发现：要求zookeeper集群必须选举出一个 Leader 进程，同时 Leader 会维护一个 Follower 可用客户端列表。将来客户端可以和这些 Follower节点进行通信。

· · 同步：Leader 要负责将本身的数据与 Follower 完成同步，做到多副本存储。这样也是提现了CAP中的高可用和分区容错。Follower将队列中未处理完的请求消费完成后，写入本地事务日志中。

· · 广播：Leader 可以接受客户端新的事务Proposal请求，将新的Proposal请求广播给所有的 Follower。

分发之后Leader服务器需要等待所有Follower服务器的反馈（Ack请求），在Zab协议中，只要超过半数的Follower服务器进行了正确的反馈后（也就是收到半数以上的Follower的Ack请求），那么 Leader 就会再次向所有的 Follower服务器发送 Commit 消息，要求其将上一个 事务proposal 进行提交。

先是给所有人发一个proposal,半数成功之后就是给所有人发commit

Zab 协议包括两种基本的模式：**崩溃恢复** 和 **消息广播**

### 协议过程

当整个集群启动过程中，或者当 Leader 服务器出现网络中弄断、崩溃退出或重启等异常时，Zab协议就会 **进入崩溃恢复模式**，选举产生新的Leader。

当选举产生了新的 Leader，同时集群中有过半的机器与该 Leader 服务器完成了状态同步（即数据同步）之后，Zab协议就会退出崩溃恢复模式，**进入消息广播模式**。

这时，如果有一台遵守Zab协议的服务器加入集群，因为此时集群中已经存在一个Leader服务器在广播消息，那么该新加入的服务器自动进入恢复模式：找到Leader服务器，并且完成数据同步。同步完成后，作为新的Follower一起参与到消息广播流程中。

### 消息广播流程 1）客户端发起一个写操作请求。

2）Leader 服务器将客户端的请求转化为事务 Proposal 提案，同时为每个 Proposal 分配一个全局的ID，即zxid。

3）Leader 服务器为每个 Follower 服务器分配一个单独的队列，然后将需要广播的 Proposal 依次放到队列中取，并且根据 FIFO 策略进行消息发送。

4）Follower 接收到 Proposal 后，会首先将其以事务日志的方式写入本地磁盘中，写入成功后向 Leader 反馈一个 Ack 响应消息。

5）Leader 接收到超过半数以上 Follower 的 Ack 响应消息后，即认为消息发送成功，可以发送 commit 消息。

6）Leader 向所有 Follower 广播 commit 消息，同时自身也会完成事务提交。Follower 接收到 commit 消息后，会将上一条事务提交。