**ZAB协议**

# ZAB

Zookeeper原子广播协议，即Zookeeper Atomic Broadcast，ZAB协议是分布式协调服务Zookeeper专门设计的一种崩溃恢复和原子广播协议。

## 1.1.两种基本模式

ZAB协议包括两种基本模式，分别是崩溃恢复模式（选主和数据同步）和消息广播模式（事务操作）

## 1.2. ZXID

ZXID是一个长度64位数字，其中低32位是按照数字递增的，即每次客户端发起一个proposal，低32位数字就简单+1，高32位是leader周期epoch编号，每当选出一个新的leader，新的leader就从本地事务日志中取出ZXID，然后解析出高32位的epoch编号，加一后低32位全部置0。

## 1.3. 节点状态

* Looking，竞选状态
* Following，跟随状态，同步leader状态，参与投票
* Observing，观察状态，同步leader状态，不参与投票
* Leading，领导者状态

节点分为以下角色：

* Leader：恢复数据，维持与follower心跳，并接受follower请求，follower的消息类型有ping消息、request消息、ack消息、revalidate消息，ping消息是指follower的心跳信息，request消息是follower发送提议信息，包括写请求和同步请求，ack消息是follower对提议的回复，超过半数follower通过，commit该提议，revalidate消息是用来延长session有效时间。
* Follower：向leader发送请求（ping消息、request、ack消息、revalidate消息），接受leader消息并处理，接收client的请求，如果是写请求，发送给leader，follower会处理来自leader的几种消息：ping消息（心跳信息）、proposal消息（leader发起的提案）、commit消息、uptodate消息（表明同步成功）、sync消息---用来强制得到最新数据

## 1.4.触发选主

ZAB使用fastLeaderElection选主算法，

* 在集群初始化启动
* 对于follower来说，leader发生网络分区或是宕机而无法与follower保持联系，触发选举
* 对于leader而言，Leader发现未收到过半的心跳报文回复（leader与follower节点之间产生网络分区）

而当一台机器进入leader选举流程时，当前集群可能处于一下两种状态：

* 集群中本来就存在leader（如新节点加入）
* 集群中确实不存在leader（leader宕机或集群初始化）

### 1.4.1.选举

选举阶段，集群间投票vote主要包括两个维度信息（ID，ZXID）

* ID：集群中每个节点都有一个唯一标识
* ZXID：从节点的内存中取得，每个事务都有一个唯一的标识

选举算法有三个线程：

* 选举线程，调用lookForLeader方法，通过阻塞队列sendqueue和recvqueue与其他两个线程协作
* workerReceive线程：选票接收器，不断获取其他服务器发送的选举消息，筛选后放入recvqueue队列，workerReceive线程一直处于运行状态，即使当前节点处于leading状态或是following状态

1. looking状态，将外部选票信息转交给选举线程
2. 非looking状态，收到处于looking状态节点的投票信息，当前节点向集群中广播最新的内存vote(ID,ZXID)，落后节点收到后，注册到leader上，完成数据同步，跟上节奏，提供正常服务

* workerSender线程：从sendqueue队列中获取待发送选票，广播到集群

选主算法的ZXID是从内存数据库中取的最新事务ID，而事务如果在内存中，则肯定是接收到commit消息后应用到内存中，所以选主只是选出了内存中数据最新的节点，

需要注意的是，ZAB在达成多数派选出leader时，此时的leader只是暂时，并不会接受客户端的请求，处于加读锁状态，只有与follower数据同步之后，leader才真正成为leader接受client的请求。

与raft不同之处：

1. Raft算法中一轮投票中每个节点只能投一票，而ZAB中节点根据投票请求的ZXID、SID决定是否重新投票，然后将自己的投票结果发送给其他节点。
2. Raft是一旦follower在election timeout时间内未收到leader的心跳报文，则发起选举，但不一定获胜，因为其最后一个日志可能比其他节点更旧，则可能有两种状况：

* leader没有宕机，只是网络分区导致leader和follower不能通信，follower带着更大的term发起选举，收到选举信息的follower节点更新自己的term，leader发送RPC请求被忽略，election timeout时间后触发重新选主，如果该节点日志是最新的，则可能成为leader，否则不可能成为leader，只是扰动整个集群。
* Leader宕机导致follower在election timeout时间内未收到心跳报文，按照正常程序选主。

1. ZAB则是leader和follower各自检测超时方式，leader检测是否过半follower心跳报文回复了，follower检测leader是否发送心跳报文。

* 一旦leader检测失败，leader进入looking状态，其他follower一段时间内收不到leader心跳也会进入looking状态，从而触发新的选举。
* 一旦follower检测失败，该follower进入looking状态，此时如果其他节点与leader保持良好通信，则follower仍然学习上述leader的投票，而不是触发新一轮的leader选举

## 1.5.一致性保证

ZAB协议保证在leader选举过程中，已经被commit的数据不会丢失，未被commit的数据对客户不可见。

### 1.5.1 commit过的数据不会丢失

假设一个事务在leader服务器上被提交了，并且已经得到了多数follower服务器ack的反馈，但是在将commit消息发送给所有follower之前，leader服务器挂了，如何保证一致性？

确保的方式是在重新选举leader的时候只挑ZXID最大的follower，因为至少半数follower回复ack说明重新选举时ZXID最大的follower应该是当初回复ack但尚未提交中的一个。

### 1.5.2 丢弃那些只在leader上提出的数据

假设leader在提出一个事务之后就崩溃退出，导致集群中其他节点都没有收到这个事务，进入崩溃恢复阶段，其他follower选主成功之后这个宕机的leader以follower的身份加入，此时他有一个多余的日志，与其他节点不一致。

确保的方式是新的leader通过分析follower节点发送的

## 1.6.日志同步

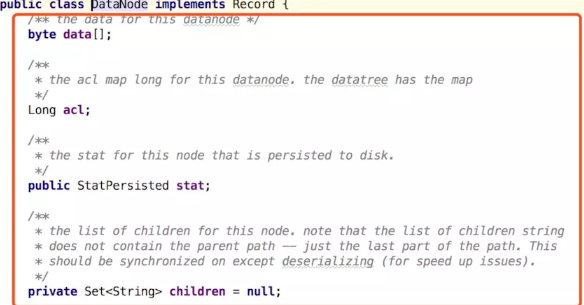
Raft的日志必须是连续的，而且是由leader流向follower，选举的leader拥有最多最新的日志，ZAB本身不要求日志连续，日志也可以由follower流向leader，但fastLeaderElection要求日志连续，follower根据leader的日志内容更新自己的日志，fastLeaderElection是ZAB默认的选主算法。

# zookeeper架构原理

zookeeper是一个分布式服务框架，主要用来解决分布式集群中一致性的问题，它能提供类似文件系统的目录节点树方式存储数据，但是zookeeper不是用来存储数据，它的作用主要是用来维护和监控存储数据的状态变化。

## 2.1.zookeeper数据结构

Zookeeper核心数据结构是一个DataTree，其内部是ConcurrentHashMap<String, DataNode> nodes，其中key是path，DataNode是真正保存数据的类型。



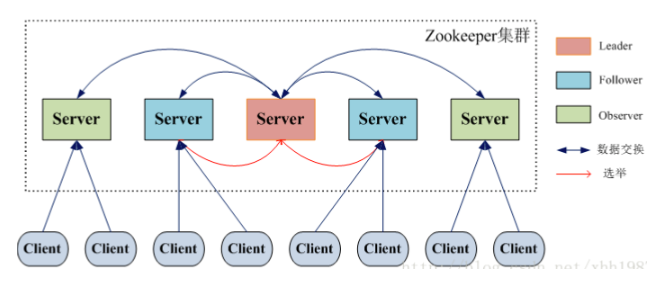
Byte data[]保存节点内容，Set<String> children保存该DataNode节点下path。

## 2.2.zookeeper数据模型

Zookeeper会维护一个具有层次关系的数据结构，非常类似于一个标准的文件系统，每个节点都叫数据节点。

* znode可以有子节点目录，每个咋弄的都可以存储数据，注意临时节点ephemeral类型的目录节点不能有子节点目录
* znode是有版本的，每个znode存储的数据可以有多个版本，
* znode可以是临时节点，一旦创建这个znode的client与server失去联系，这个znode自动删除，zookeeper的client与server通信采用长连接方式，每个client和server通过心跳来保持连接，这个状态称为session，如果znode是临时节点，这个session失效，znode自动删除
* znode的目录名可以自动编号，如APP1已经存在，再创建的话，就会自动命名为app2
* znode可以被监控，包括这个目录节点的数据的修改，子节点目录的变化，一旦变化可以通知设置监控的client

## 2.3.zookeeper的架构



### 2.3.1.zookeeper集群角色介绍

* leader：leader不直接接受client的请求，但接受由其他follower和observer转发过来的client请求，此外，leader还负责投票发起和决议，即时更新数据与状态
* follower：follower接受client请求并返回结果，参与leader发起的投票和选举，写请求将转发给leader节点
* observer：observer接受client连接，将写操作转给leader，但observer不参与投票，只同步leader节点状态

### 2.3.2.zookeeper的保证

* 更新请求顺序进行，来自同一个client的更新请求按其发送顺序依次执行
* 数据更新原子性，依次数据更新要么成功，要么失败
* 全局唯一数据视图，client无论连接到哪个server，数据视图都是一致的
* 实时性，在一定时间范围内，client都能读到最新数据

## 2.4.应用场景

### 2.4.1.统一命名服务

Zookeeper的命名服务有两个应用方向：

* Zookeeper提供类似JNDI的服务，都能够帮助应用系统通过一个资源引用方式实现对资源的定位与使用
* 利用zookeeper顺序节点的特性，提供分布式的ID生成器

### 2.4.2.配置管理

利用zookeeper的发布、订阅和watch功能，把相关的配置数据写到zookeeper的某个指定的节点下，应用服务监听这个节点的数据变化，一旦节点数据发生了变化，应用服务就会收到zookeeper的通知，然后client就可以从zookeeper获取新的配置信息。

### 2.4.3.集群管理

多台server组成一个服务集群，需要知道集群中每台机器的服务状态，一旦有机器不能提供服务，集群中其他机器必须知道，从而做出调整重新分配服务策略。

### 2.4.4.分布式锁

可以通过在zookeeper中创建一个临时节点，然后调用getchildren方法获取当前目录节点列表中最小的目录节点，最小目录节点获得锁，其他节点监听比自己小的最大节点，当其监听的节点释放锁，则当前节点获得锁。

### 2.4.5.队列管理

Zookeeper可以处理两种类型的队列：

* 当一个队列的成员都聚齐，这个队列才可用，否则等待所有成员达到，即同步队列
* 队列按照FIFO方式进行入队和出队操作

#### 2.4.5.1.同步队列

创建一个父目录/synchronized，每个成员都监控标志位目录/synchronized/start是否存在，然后每个成员都加入这个队列，加入队列的方式就是创建/synchronized/member\_i，临时目录节点，然后每个成员获取/synchronized目录下所有节点，如果个数等于成员个数，创建/synchronized/start。

#### 2.4.5.2.FIFO队列

在特定的目录下创建sequential类型的子目录/queue\_i，这样就能保证所有成员加入队列都是由编号的，出队时通过getchildren方法返回当前所有队列中的元素，然后消费其中最小的一个。