

# 1. ZooKeeper是什么,有什么功能

---

Zookeeper 是一个典型的分布式数据一致性的解决方案。

**Zookeeper的典型应用场景:**

- 数据发布/订阅
- 负载均衡
- 命名服务
- 分布式协调/通知
- 集群管理
- Master
- 分布式锁
- 分布式队列

## 2. 讲一下leader 选举过程

---

这里选取3台机器组成的服务器集群为例。在集群初始化阶段，当有一台服务器Server1启动时，其单独无法进行和完成Leader选举，当第二台服务器Server2启动时，此时两台机器可以相互通信，每台机器都试图找到Leader，于是进入Leader选举过程。选举过程如下：

**(1) 每个Server发出一个投票。**由于是初始情况，Server1和Server2都会将自己作为Leader服务器来进行投票，每次投票会包含所推举的服务器的myid和ZXID，使用(myid, ZXID)来表示，此时Server1的投票为(1, 0)，Server2的投票为(2, 0)，然后各自将这个投票发给集群中其他机器。

**(2) 接受来自各个服务器的投票。**集群的每个服务器收到投票后，首先判断该投票的有效性，如检查是否是本轮投票、是否来自LOOKING状态的服务器。

**(3) 处理投票。**针对每一个投票，服务器都需要将别人的投票和自己的投票进行PK，PK规则如下

- **优先检查ZXID。** ZXID比较大的服务器优先作为Leader。
- **如果ZXID相同，那么就比较myid。** myid较大的服务器作为Leader服务器。

对于Server1而言，它的投票是(1, 0)，接收Server2的投票为(2, 0)，首先会比较两者的ZXID，均为0，再比较myid，此时Server2的myid最大，于是更新自己的投票为(2, 0)，然后重新投票，对于Server2而言，其无须更新自己的投票，只是再次向集群中所有机器发出上一次投票信息即可。

**(4) 统计投票。**每次投票后，服务器都会统计投票信息，判断是否已经有过半机器接受到相同的投票信息，对于Server1、Server2而言，都统计出集群中已经有两台机器接受了(2, 0)的投票信息，此时便认为已经选出了Leader。

**(5) 改变服务器状态。**一旦确定了Leader，每个服务器就会更新自己的状态，如果是Follower，那么就变更为FOLLOWING，如果是Leader，就变更为LEADING。

## Leader 选取算法分析

在3.4.0后的Zookeeper的版本只保留了TCP版本的FastLeaderElection选举算法。当一台机器进入Leader选举时，当前集群可能会处于以下两种状态

- 集群中已经存在Leader。
- 集群中不存在Leader。

对于集群中已经存在Leader而言，此种情况一般都是某台机器启动得较晚，在其启动之前，集群已经在正常工作，对这种情况，该机器试图去选举Leader时，会被告知当前服务器的Leader信息，对于该机器而言，仅仅需要和Leader机器建立起连接，并进行状态同步即可。而在集群中不存在Leader情况下则会相对复杂，其步骤如下：

(1) **第一次投票**。无论哪种导致进行Leader选举，集群的所有机器都处于试图选举出一个Leader的状态，即LOOKING状态，LOOKING机器会向所有其他机器发送消息，该消息称为投票。投票中包含了SID（服务器的唯一标识）和ZXID（事务ID），(SID, ZXID)形式来标识一次投票信息。假定Zookeeper由5台机器组成，SID分别为1、2、3、4、5，ZXID分别为9、9、9、8、8，并且此时SID为2的机器是Leader机器，某一时刻，1、2所在机器出现故障，因此集群开始进行Leader选举。在第一次投票时，每台机器都会将自己作为投票对象，于是SID为3、4、5的机器投票情况分别为(3, 9)，(4, 8)，(5, 8)。

(2) **变更投票**。每台机器发出投票后，也会收到其他机器的投票，每台机器会根据一定规则来处理收到的其他机器的投票，并以此来决定是否需要变更自己的投票，这个规则也是整个Leader选举算法的核心所在，其中术语描述如下

- **vote\_sid**：接收到的投票中所推举Leader服务器的SID。
- **vote\_zxid**：接收到的投票中所推举Leader服务器的ZXID。
- **self\_sid**：当前服务器自己的SID。
- **self\_zxid**：当前服务器自己的ZXID。

每次对收到的投票的处理，都是对(vote\_sid, vote\_zxid)和(self\_sid, self\_zxid)对比的过程。

规则一：如果vote\_zxid大于self\_zxid，就认可当前收到的投票，并再次将该投票发送出去。

规则二：如果vote\_zxid小于self\_zxid，那么坚持自己的投票，不做任何变更。

规则三：如果vote\_zxid等于self\_zxid，那么就对比两者的SID，如果vote\_sid大于self\_sid，那么就认可当前收到的投票，并再次将该投票发送出去。

规则四：如果vote\_zxid等于self\_zxid，并且vote\_sid小于self\_sid，那么坚持自己的投票，不做任何变更。

结合上面规则，给出下面的集群变更过程。



**zk的作用主要有如下几点:**

1. kafka的元数据都存放在zk上面,由zk来管理
2. 0.8之前版本的kafka, consumer的消费状态, group的管理以及 offset的值都是由zk管理的,现在offset会保存在本地topic文件里
3. 负责borker的lead选举和管理

## 4. 说一下ZooKeeper的通知机制

---

客户端会对某个 znode 建立一个 watcher 事件, 当该 znode 发生变化时, 这些客户端会收到 zookeeper 的通知, 然后客户端可以根据 znode 变化来做出业务上的改变

## 5. ZooKeeper的分布式锁实现方式

---

使用zookeeper实现分布式锁的算法流程, 假设锁空间的根节点为/lock:

1. 客户端连接zookeeper, 并在/lock下创建临时的且有序的子节点, 第一个客户端对应的子节点为/lock/lock-0000000000, 第二个为/lock/lock-0000000001, 以此类推。
2. 客户端获取/lock下的子节点列表, 判断自己创建的子节点是否为当前子节点列表中序号最小的子节点, 如果是则认为获得锁, 否则监听刚好在自己之前一位的子节点删除消息, 获得子节点变更通知后重复此步骤直至获得锁;
3. 执行业务代码;
4. 完成业务流程后, 删除对应的子节点释放锁。

## 6. ZooKeeper是怎样保证主从节点的状态同步

---

zookeeper 的核心是原子广播, 这个机制保证了各个 server 之间的同步。实现这个机制的协议叫做 zab 协议。zab 协议有两种模式, 分别是恢复模式(选主)和广播模式(同步)。当服务启动或者在领导者崩溃后, zab 就进入了恢复模式, 当领导者被选举出来, 且大多数 server 完成了和 leader 的状态同步以后, 恢复模式就结束了。状态同步保证了 leader 和 server 具有相同的系统状态。

## 7. ZooKeeper有几种部署模式

---

zookeeper有两种运行模式: 集群模式和单机模式,还有一种伪集群模式,在单机模式下模拟集群的zookeeper服务

## 8. ZooKeeper采用的哪种分布式一致性协议? 还有哪些分布式一致性协议

---

常见的分布式一致性协议有: 两阶段提交协议, 三阶段提交协议, 向量时钟, RWN协议, paxos协议, Raft协议. zk采用的是paxos协议。

- **两阶段提交协议(2PC)**

两阶段提交协议，简称2PC，是比较常用的解决分布式事务问题的方式，要么所有参与进程都提交事务，要么都取消事务，即实现ACID中的原子性(A)的常用手段。

- **三阶段提交协议(3PC)**

3PC就是在2PC基础上将2PC的提交阶段细分位两个阶段：预提交阶段和提交阶段

- **向量时钟**

通过向量空间祖先继承的关系比较, 使数据保持最终一致性,这就是向量时钟的基本定义。

- **NWR协议**

NWR是一种在分布式存储系统中用于控制一致性级别的一种策略。在Amazon的Dynamo云存储系统中，就应用NWR来控制一致性。

让我们先来看看这三个字母的含义：

N：在分布式存储系统中，有多少份备份数据

W：代表一次成功的更新操作要求至少有w份数据写入成功

R：代表一次成功的读数据操作要求至少有R份数据成功读取

NWR值的不同组合会产生不同的一致性效果，当 $W+R>N$ 的时候，整个系统对于客户端来讲能保证强一致性。

当W+R 以常见的 $N=3$ 、 $W=2$ 、 $R=2$ 为例：

$N=3$ ，表示，任何一个对象都必须有三个副本（Replica）， $W=2$ 表示，对数据的修改操作（Write）只需要在3个Replica中的2个上面完成就返回， $R=2$ 表示，从三个对象中要读取到2个数据对象，才能返回。

在分布式系统中，数据的单点是不允许存在的。即线上正常存在的Replica数量是1的情况是非常危险的，因为一旦这个Replica再次错误，就可能发生数据的永久性错误。假如我们把N设置成为2，那么，只要有一个存储节点发生损坏，就会有单点的存在。所以N必须大于2。N约高，系统的维护和整体 成本就越高。工业界通常把N设置为3。

当W是2、R是2的时候， $W+R>N$ ，这种情况对于客户端就是强一致性的。

- **paxos协议**

[架构师需要了解的Paxos原理、历程及实践](#)

- **Raft协议**

[Raft协议的动画](#)

注：资料来源于网络。