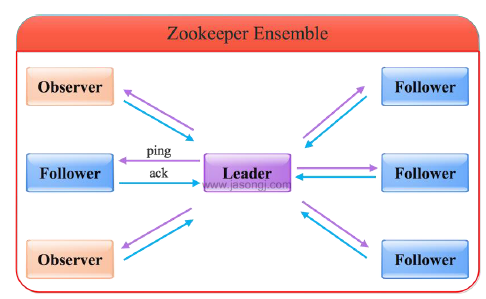
# 11. Zookeeper

## 11.1.1. Zookeeper概念

Zookeeper是一个分布式协调服务，可用于服务发现，分布式锁，分布式领导选举，配置管理等。Zookeeper提供了一个类似于Linux文件系统的树形结构（可认为是轻量级的内存文件系统，但只适合存少量信息，完全不适合存储大量文件或者大文件），同时提供了对于每个节点的监控与通知机制。

## 11.1.2. Zookeeper角色

Zookeeper集群是一个基于主从复制的高可用集群，每个服务器承担如下三种角色中的一种



### 11.1.2.1. Leader

1. 一个Zookeeper集群同一时间只会有一个实际工作的Leader，它会发起并维护与各Follwer及Observer间的心跳。

2. 所有的写操作必须要通过Leader完成再由Leader将写操作广播给其它服务器。只要有超过半数节点（不包括observer节点）写入成功，该写请求就会被提交（类 2PC 协议）。

### 11.1.2.2. Follower

1. 一个Zookeeper集群可能同时存在多个Follower，它会响应Leader的心跳，

2. Follower可直接处理并返回客户端的读请求，同时会将写请求转发给Leader处理，

3. 并且负责在Leader处理写请求时对请求进行投票。

### 11.1.2.3. Observer

角色与Follower类似，但是无投票权。Zookeeper需保证高可用和强一致性，为了支持更多的客户端，需要增加更多Server；Server增多，投票阶段延迟增大，影响性能；引入Observer，Observer不参与投票； Observers接受客户端的连接，并将写请求转发给leader节点； 加入更多Observer节点，提高伸缩性，同时不影响吞吐率。

### 11.1.2.4. ZAB协议

ZAB ( ZooKeeper Atomic Broadcast)是指 ZooKeeper 原子消息广播协议。

***事务编号 Zxid（事务请求计数器+ epoch）***

在 ZAB协议的事务编号 Zxid 设计中，Zxid 是一个 64 位的数字，其中低 32 位是一个简单的单调递增的计数器，针对客户端每一个事务请求，计数器加 1；而高 32 位则代表 Leader 周期 epoch 的编号，每个当选产生一个新的 Leader 服务器，就会从这个 Leader 服务器上取出其本地日志中最大事务的ZXID，并从中读取 epoch 值，然后加 1，以此作为新的 epoch，并将低 32 位从 0 开始计数。

Zxid（Transaction id）类似于RDBMS中的事务ID，用于标识一次更新操作的Proposal（提议） ID。为了保证顺序性，该zkid必须单调递增。

***epoch***

epoch：可以理解为当前集群所处的年代或者周期，每个 leader 就像皇帝，都有自己的年号，所以每次改朝换代，leader 变更之后，都会在前一个年代的基础上加 1。这样就算旧的 leader 崩溃恢复之后，也没有人听他的了，因为 follower 只听从当前年代的 leader 的命令。

#### Zab协议有两种模式-恢复模式（选主）、广播模式（同步）

Zab协议有两种模式，它们分别是恢复模式（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。

#### ZAB协议4阶段

***Leader election（选举阶段-选出准Leader）***

1. Leader election（选举阶段）：节点在一开始都处于选举阶段，只要有一个节点得到超半数节点的票数，它就可以当选准 leader。只有到达 广播阶段（broadcast） 准 leader 才会成为真正的 leader。这一阶段的目的是就是为了选出一个准 leader，然后进入下一个阶段。

***Discovery（发现阶段-接受提议、生成epoch、接受epoch）***

2. Discovery（发现阶段）：在这个阶段，followers 跟准 leader 进行通信，同步 followers 最近接收的事务提议。这个一阶段的主要目的是发现当前大多数节点接收的最新提议，并且准 leader 生成新的 epoch，让 followers 接受，更新它们的 accepted Epoch

一个 follower 只会连接一个 leader，如果有一个节点 f 认为另一个 follower p 是 leader，f 在尝试连接 p 时会被拒绝，f 被拒绝之后，就会进入重新选举阶段。

***Synchronization（同步阶段-同步follower副本）***

3. Synchronization（同步阶段）：同步阶段主要是利用 leader 前一阶段获得的最新提议历史，同步集群中所有的副本。只有当 大多数节点都同步完成，准 leader 才会成为真正的 leader。follower 只会接收 zxid 比自己的 lastZxid 大的提议。

***Broadcast（广播阶段-leader消息广播）***

4. Broadcast（广播阶段）：到了这个阶段，Zookeeper 集群才能正式对外提供事务服务，并且 leader 可以进行消息广播。同时如果有新的节点加入，还需要对新节点进行同步。

ZAB 提交事务并不像 2PC 一样需要全部 follower 都 ACK，只需要得到超过半数的节点的 ACK 就可以了。

***ZAB协议JAVA实现（FLE-发现阶段和同步合并为 Recovery Phase（恢复阶段））***

协议的 Java 版本实现跟上面的定义有些不同，选举阶段使用的是 Fast Leader Election（FLE），它包含了 选举的发现职责。因为 FLE 会选举拥有最新提议历史的节点作为 leader，这样就省去了发现最新提议的步骤。实际的实现将 发现阶段 和 同步合并为 Recovery Phase（恢复阶段）。所以，ZAB 的实现只有三个阶段：Fast Leader Election；Recovery Phase；Broadcast Phase。

### 11.1.2.5. 投票机制

**每个sever首先给自己投票，然后用自己的选票和其他sever选票对比，权重大的胜出，使用权重较大的更新自身选票箱。**具体选举过程如下：

1. 每个Server启动以后都询问其它的Server它要投票给谁。对于其他server的询问，server每次根据自己的状态都回复自己推荐的leader的id和上一次处理事务的zxid（系统启动时每个server都会推荐自己）

2. 收到所有Server回复以后，就计算出zxid最大的哪个Server，并将这个Server相关信息设置成下一次要投票的Server。

3. 计算这过程中获得票数最多的的sever为获胜者，如果获胜者的票数超过半数，则改server被选为leader。否则，继续这个过程，直到leader被选举出来

4. leader就会开始等待server连接

5. Follower连接leader，将最大的zxid发送给leader

6. Leader根据follower的zxid确定同步点，至此选举阶段完成。

7. 选举阶段完成Leader同步后通知follower 已经成为uptodate状态

8. Follower收到uptodate消息后，又可以重新接受client的请求进行服务了

目前有5台服务器，每台服务器均没有数据，它们的编号分别是1,2,3,4,5,按编号依次启动，它们的选择举过程如下：

1. 服务器1启动，给自己投票，然后发投票信息，由于其它机器还没有启动所以它收不到反馈信息，服务器1的状态一直属于Looking。

2. 服务器2启动，给自己投票，同时与之前启动的服务器1交换结果，由于服务器2的编号大所以服务器2胜出，但此时投票数没有大于半数，所以两个服务器的状态依然是LOOKING。

3. 服务器3启动，给自己投票，同时与之前启动的服务器1,2交换信息，由于服务器3的编号最大所以服务器3胜出，此时投票数正好大于半数，所以服务器3成为领导者，服务器1,2成为小弟。

4. 服务器4启动，给自己投票，同时与之前启动的服务器1,2,3交换信息，尽管服务器4的编号大，但之前服务器3已经胜出，所以服务器4只能成为小弟。

5. 服务器5启动，后面的逻辑同服务器4成为小弟。

## 11.1.2. Zookeeper工作原理（原子广播）

1. **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，它们分别是恢复模式和广播模式。

2. 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的完成了和leader的状态同步以后，恢复模式就结束了。

3. 状态同步保证了leader和server具有相同的系统状态

4. **一旦leader已经和多数的follower进行了状态同步后，他就可以开始广播消息了**，即进入广播状态。这时候当一个server加入zookeeper服务中，它会在恢复模式下启动，发现leader，并和leader进行状态同步。待到同步结束，它也参与消息广播。Zookeeper服务一直维持在Broadcast状态，直到leader崩溃了或者leader失去了大部分的followers支持。

5. 广播模式需要保证proposal被按顺序处理，因此zk采用了递增的事务id号(zxid)来保证。所有的提议(proposal)都在被提出的时候加上了zxid。

6. 实现中zxid是一个64为的数字，它高32位是epoch用来标识leader关系是否改变，每次一个leader被选出来，它都会有一个新的epoch。低32位是个递增计数。

7. 当leader崩溃或者leader失去大多数的follower，这时候zk进入恢复模式，恢复模式需要重新选举出一个新的leader，让所有的server都恢复到一个正确的状态。

## 11.1.3. Znode有四种形式的目录节点

1. PERSISTENT：持久的节点。

2. EPHEMERAL：暂时的节点。

3. PERSISTENT\_SEQUENTIAL：持久化顺序编号目录节点。

4. EPHEMERAL\_SEQUENTIAL：暂时化顺序编号目录节点。