ZooKeeper服务的内部通信基于**Zab（ZooKeeper Atomic Broadcast）协议**，Zab是在**Paxos算法**基础上扩展改造而来的。

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。

选举出Leader节点后Zab进入原子广播阶段，这时Leader为和自己同步的每个节点Follower创建一个操作序列，一个时期一个Follower只能和一个Leader保持同步，Leader节点与Follower节点使用心跳检测来感知对方的存在。当Leader节点在超时时间内收到来自Follower的心跳检测，那么Follower节点会一直与该节点保持连接，若超时时间内Leader没有接收到来自过半Follower节点的心跳检测或TCP连接断开，那Leader会结束当前周期的领导，切换到Looking状态，所有Follower节点也会放弃该Leader节点切换到Looking状态，然后开始新一轮选举。

每个Server在工作过程中有四种状态：

l   LOOKING：当前Server不知道Leader是谁，正在搜寻

l   LEADING：当前Server即为选举出来的Leader

l   FOLLOWING：Leader已经选举出来，当前Server与之同步

l   OBSERVING：Observer的行为在大多数情况下与Follower完全一致，但是他们不参加选举和投票，而仅仅接受(observing)选举和投票的结果。

**3.2.1 选举流程**

当Leader崩溃或者Leader失去大多数的Follower，这时候ZooKeeper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。ZooKeeper的选举算法有两种：一种是基于basic paxos实现的，另外一种是基于fast paxos算法实现的。系统默认的选举算法为fast paxos。

fast paxos流程是在选举过程中，某Server首先向所有Server提议自己要成为Leader，当其它Server收到提议以后，解决epoch和 zxid的冲突，并接受对方的提议，然后向对方发送接受提议完成的消息，重复这个流程，最后一定能选举出Leader。

用文字描述就是：

（1）全天下我最牛，在我没有发现比我牛的推荐人的情况下，我就一直推举我当Leader。第一次投票那必须推举我自己当Leader。

（2）每当我接收到其它的被推举者，我都要回馈一个信息，表明我还是不是推举我自己。如果被推举者没我大，我就一直推举我当Leader，是我是我还是我！

（3）我有一个票箱，和我属于同一轮的投票情况都在这个票箱里面。一人一票 重复的或者过期的票，我都不接受。

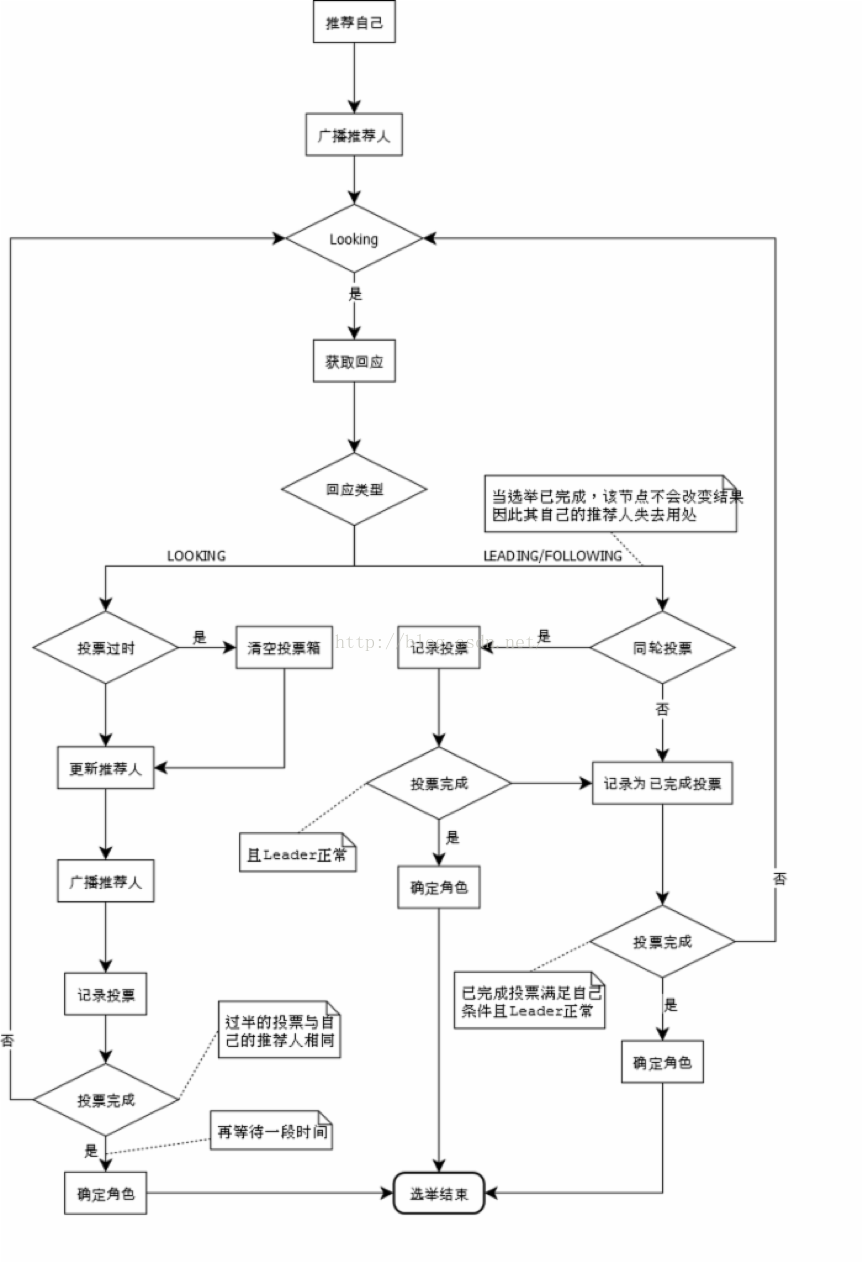
（4）一旦我不再推举我自己了（这时我发现别人推举的人比我推荐的更牛），我就把我的票箱清空，重新发起一轮投票（这时我的票箱一定有两票了，都是选的我认为最牛的人）。

（5）一旦我发现收到的推举信息中投票轮要高于我的投票轮，我也要清空我的票箱。并且还是投当初我觉得最牛的那个人（除非当前的人比我最初的推荐牛，我就顺带更新我的推荐）。

（6）不断的重复上面的过程，不断的告诉别人“我的投票是第几轮”、“我推举的人是谁”。直到我的票箱中“我推举的最牛的人”收到了不少于 N /2 + 1的推举投票。

（7）这时我就可以决定我是Follower还是Leader了（如果至始至终都是我最牛，那我就是Leader，其它情况就是Follower）。并且不论随后收到谁的投票，都向它直接反馈“我的结果”。

其流程图如下所示：



**3.2.2 同步流程**

选完Leader以后，ZooKeeper就进入状态同步过程。

1、Leader等待Server连接；

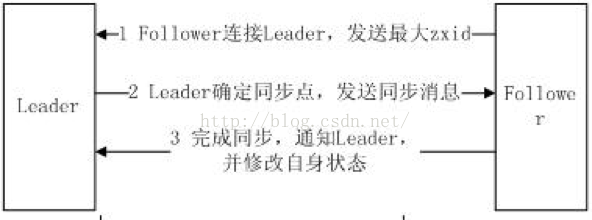
2、Follower连接Leader，将最大的Zxid发送给Leader；

3、Leader根据Follower的Zxid确定同步点；

4、完成同步后通知Follower 已经成为uptodate状态；

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

流程图如下所示：



**3.2.3 Leader工作流程**

Leader主要有三个功能：

1、恢复数据；

2、维持与Learner的心跳，接收Learner请求并判断Learner的请求消息类型；

3、Learner的消息类型主要有**PING**消息、**REQUEST**消息、**ACK**消息、**REVALIDATE**消息，根据不同的消息类型，进行不同的处理。

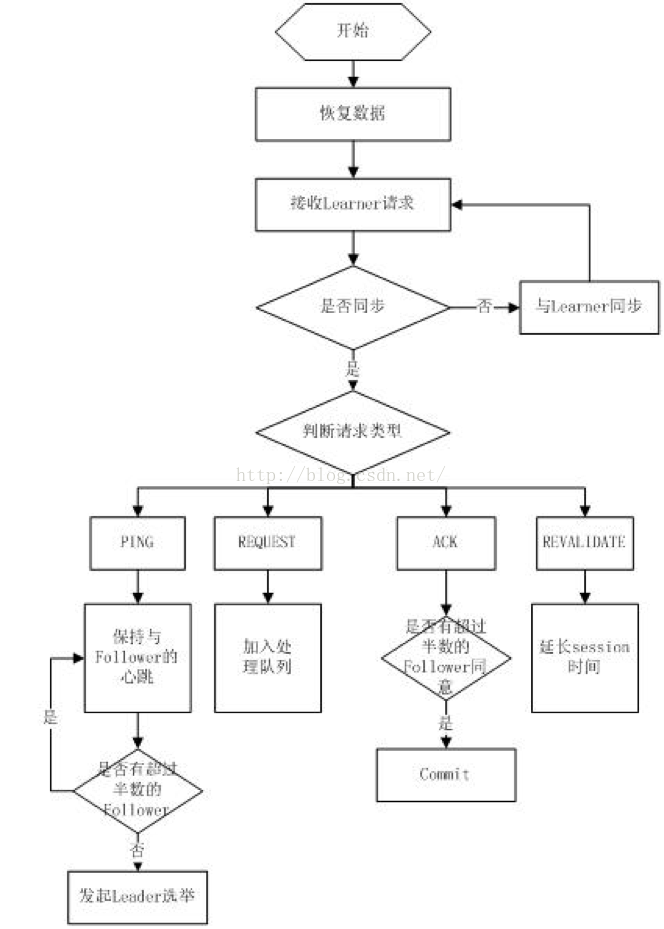
l   PING消息是指Learner的心跳信息；

l   REQUEST消息是Follower发送的提议信息，包括写请求及同步请求；

l   ACK消息是 Follower的对提议的回复，超过半数的Follower通过，则commit该提议；

l   REVALIDATE消息是用来延长SESSION有效时间。

Leader的工作流程简图如下所示：



**3.2.4 Follower工作流程**

Follower主要有四个功能：

1、向Leader发送请求（PING消息、REQUEST消息、ACK消息、REVALIDATE消息）；

2、接收Leader消息并进行处理；

3、接收Client的请求，如果为写请求，发送给Leader进行投票；

4、返回Client结果。

Follower的消息循环处理如下几种来自Leader的消息：

l   **PING**消息： 心跳消息；

l   **PROPOSAL**消息：Leader发起的提案，要求Follower投票；

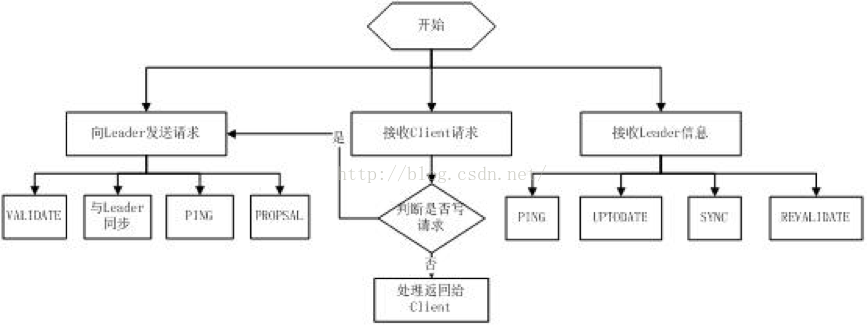
l   **COMMIT**消息：服务器端最新一次提案的信息；

l   **UPTODATE**消息：表明同步完成；

l   **REVALIDATE**消息：根据Leader的REVALIDATE结果，关闭待revalidate的session还是允许其接受消息；

l   **SYNC**消息：返回SYNC结果到客户端，这个消息最初由客户端发起，用来强制得到最新的更新。

Follower的工作流程简图如下所示：



对于Observer的流程不再叙述，Observer流程和Follower的唯一不同的地方就是observer不会参加Leader发起的投票。

**3.2.5 两段式提交**

ZooKeeper集群是通过“**两段提交协议**”（a two-phase commit）来发起意向提议。

第一阶段：Leader给所有的Follower发送一个PROPOSAL消息，一个Follower接收到这次PROPOSAL消息，写到磁盘，发送给Leader一个ACK消息，告知已经收到。

第二阶段：当Leader收到法定人数（quorum）的Follower的ACK时候，发送commit消息执行。

为了保证事务的顺序一致性，ZooKeeper采用了递增的事务id号（Zxid）来标识事务。所有的提议（Proposal）都在被提出的时候加上 了Zxid。实现中Zxid是一个64位的数字，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，标识当前属于那个Leader的统治时期。低32位用于递增计数。

Zab协议保证：

（1）如果Leader以T1和T2的顺序广播，那么所有的Server必须先执行T1，再执行T2。

（2）如果任意一个Server以T1、T2的顺序commit执行，其他所有的Server也必须以T1、T2的顺序执行。

“两段提交协议”最大的问题是如果Leader发送了PROPOSAL消息后crash或暂时失去连接，会导致整个集群处在一种不确定的状态（Follower不知道该放弃这次提交还是执行提交）。ZooKeeper这时会选出新的Leader，请求处理也会移到新的Leader上，不同的Leader由不同的epoch标识。切换Leader时，需要解决下面两个问题：

（1）Never forget delivered messages

Leader在COMMIT投递到任何一台Follower之前crash，只有它自己commit了。新Leader必须保证这个事务也必须commit。

（2）Let go of messages that are skipped

Leader产生某个proposal，但是在crash之前，没有Follower看到这个proposal。该server恢复时，必须丢弃这个proposal。