# Zookeeper基础

## 基础数据参数介绍：

znode的4中类型：

节点生命周期中，主要有4种状态

* **持久节点（persistent）:**

这类节点被创建后，就会一直存在于Zk服务器上。直到手动删除。

* **持久顺序节点（persistent\_sequential）:**

它的基本特性同持久节点，不同在于增加了顺序性。父节点会维护一个自增整性数字，用于子节点的创建的先后顺序。

* **临时节点（ephemeral）**:  
   临时节点的生命周期与客户端的会话绑定，一旦客户端会话失效（非TCP连接断开），那么这个节点就会被自动清理掉。zk规定临时节点只能作为叶子节点。
* **临时顺序节点（ephemeral\_sequential）:**

基本特性同临时节点，添加了顺序的特性。

Znode

Todo :这些节点在zookeeper中的作用？

## zookeeper特性：

Zookeeper的分布式特性：

顺序一致性:

原子性：

单一视图：

## Zookeeper选举机制

### **3.1 选举中概念：**

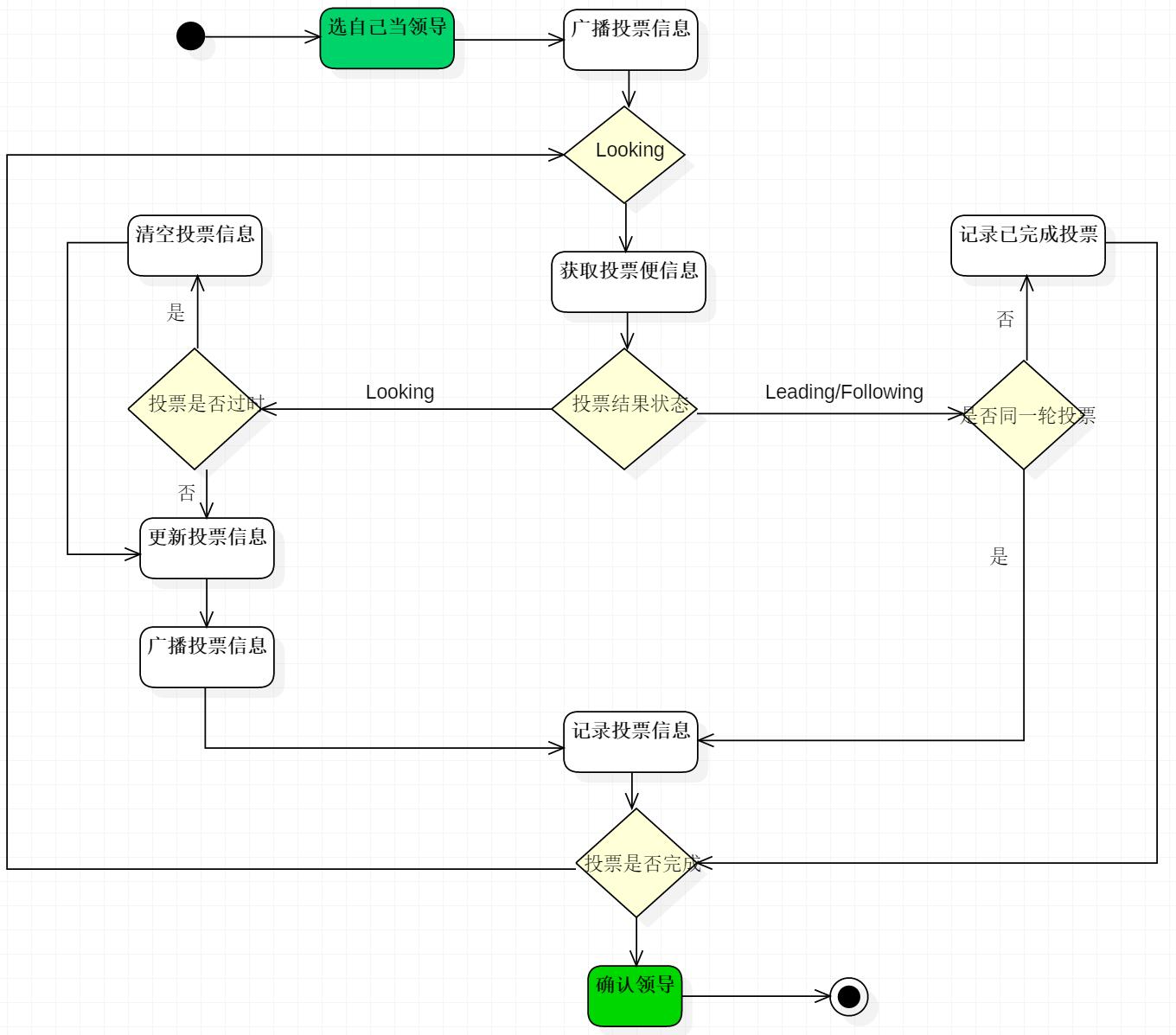
* ServiceId:服务器ID,编号越大，算法权重越大，myID;
* Zxid：数据Id,服务器汇总存放的最大的数据Id,值越大说明数据越新，在选举算法数据越新，权重越大；
* Epcho；逻辑时钟，同一轮投票过程中逻辑时钟的值是相同的。每次投完票数都会增加一，然后与接收到其他服务器返回的投票信息的数值相同，根据不同的值做出不同的判断；
* Service状态:选举状态：  
   Looking状态：竞选状态，同步leader状态，参与投票

Following状态：随从状态，同步leader状态，不参与投票；

Observing状态：观察状态，同步leader不参与投票；

Leading状态：领导者状态；

投票流程图:



### 3.2 集群启动选举方式

1. 服务器1启动,此时只有它一台服务器启动了,它发出去的报没有任何响应,所以它的**选举状态一直是LOOKING状态**
2. 服务器2启动,它与最开始启动的服务器1进行通信,互相交换自己的选举结果,由于两者都没有历史数据,所以**id值较大的服务器2胜出**,但是由于没有达到超过半数以上的服务器都同意选举它(这个例子中的半数以上是3),所以服务器1,2还是**继续保持LOOKING状态**.
3. 服务器3启动,根据前面的理论分析,服务器3成为服务器1,2,3中的老大,而与上面不同的是,此时有三台服务器选举了它,所以它成为了这次选举的leader.
4. 服务器4启动,根据前面的分析,理论上服务器4应该是服务器1,2,3,4中最大的,但是由于前面已经有半数以上的服务器选举了服务器3,所以它只能接收当小弟的命了.
5. 服务器5启动,同4一样,当小弟.

综上所述：

1. id大的数据自动选举为准leader；
2. 当启动的集群中的zk节点选举的超过一半的节点超过，后面新加的节点自动变为follower；

### 3.3 当leader宕机，选举机制：

这里需要引入一些概念：

数据id：数据新的id就大，数据**每次更新都会更新id**。

Leader id：就是我们配置的**myid中的值**，每个机器一个。

逻辑时钟：这个值从0开始递增,每次选举对应一个值,也就是说:  如果在同一次选举中,那么这个值应该是一致的 ; **逻辑时钟值越大,说明这一次选举leader的进程更新**.

选举规则：

1、逻辑时钟小的选举结果被忽略，重新投票

2、统一逻辑时钟后，数据id大的胜出

3、数据id相同的情况下，leader id大的胜出

#### 详细流程：默认的选举算法是fast paxos字段

（1）选举线程由当前Server发起**选举的线程担任**，其主要功能是对投票结果进行统计，并选出推荐的Server；

（2）选举线程首先向所有Server发起一次询问(包括自己)；

（3）选举线程收到回复后，验证是否是自己发起的询问(验证zxid是否一致)，然后获取对方的id(myid)，并存储到当前询问对象列表中，最后获取对方提议的leader相关信息(id,zxid)，并将这些信息存储到当次选举的投票记录表中；

（4）收到所有Server回复以后，就计算出zxid最大的那个Server，并将这个Server相关信息设置成下一次要投票的Server；

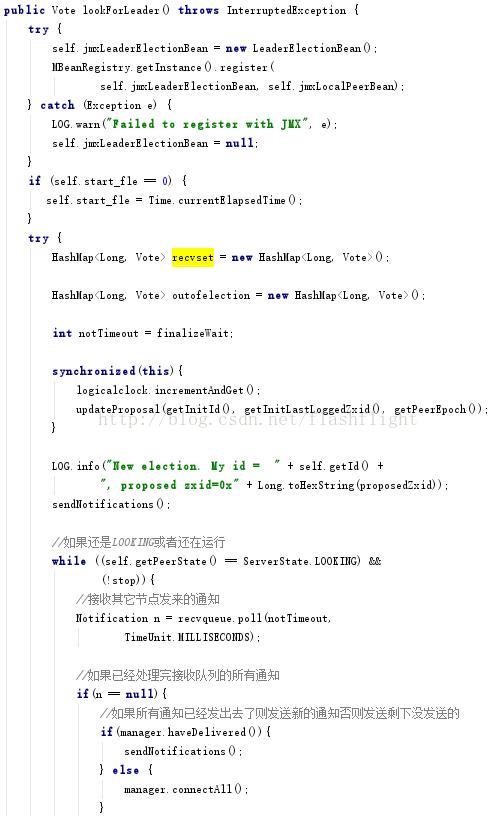
（5）线程将当前zxid最大的Server设置为当前Server要推荐的Leader，如果此时获胜的Server获得n/2 + 1的Server票数，设置当前推荐的leader为获胜的Server，将根据获胜的Server相关信息设置自己的状态，否则，继续这个过程，直到leader被选举出来。通过流程分析我们可以得出：要使Leader获得多数Server的支持，则Server总数必须是奇数2n+1，且存活的Server的数目不得少于n+1. 每个Server启动后都会重复以上流程。在恢复模式下，如果是刚从崩溃状态恢复的或者刚启动的server还会从磁盘快照中恢复数据和会话信息，zk会记录事务日志并定期进行快照，方便在恢复时进行状态恢复。

## 4.选举算法

### FastLeaderElection（默认算法）：

FastLeaderElection：当前启动时发送到所有的集群节点，每个参与的节点的初始状态是LOOKING状态，所以会选择自己当leader并返回自己的zxid，然后QuorumPeer会选择zxid大的节点作为leader,但是并不能保证zxid一定是所有的服务器中最大的；

此算法，LeaderElection不同的是它不会像后者那样在每轮投票中要搜集到所有结果后才统计投票结果。而是每次都会统计一次结果，这样效率会更加的高效；





### LeaderElection算法

### AuthLeaderElection算法：

与fastLeaderElection算法类似，只是加入了认证信息；