# 前言

根据Zookeeper的使用场景和运行原理，我们预先选择出几个探究方向：

1. zookeeper的leader选举；
2. zookeeper的数据存储和读取，数据同步；
3. zookeeper的使用场景；
4. zookeeper保证了顺序一致性、原子性、单一视图、可靠性和实时性，这些特性是如何实现的；
5. zookeeper的监听机制；
6. zookeeper的节点类型；

# zookeeper的leader选举

zookeeper是一种master/slave的结构，zookeeper顾名思义，动物园管理员，多个节点中，只有一个节点作为管理员。但管理员有可能会发生宕机或其它情况，导致zookeeper的管理员“死掉”，Leader节点崩溃，这时zookeeper的“非观察者”节点会重新选举出一个节点，在探究选举的原理和细节前，我们先了解一下zookeeper服务端的角色。

下面是zookeeper服务端的结构：



SERVER根据角色划分，分为：Leader（领导者）、Learner（学习者），而learner根据作用又可以分为Follower（跟随者）和Observer（观察者），Follower和Observer的区别是Follower参与Leader选举的投票，而Observer不参与投票，而是接收客户端的写请求，然后把请求转发给Leader。

以上说明，使用图示如下：



好了，了解了zookeeper 的角色划分，我们来探究zookeeper是如何进行Leader选举的。

Zookeeper使用了paxos（分布式一致性算法，多个节点对某个值达到一致性）算法协议作为选举策略基础，但是paxos是一种协议概念，并非拥有固定的代码实现，根据需要，可以实现自身的paxos算法。

在这里可以举两个简单的例子：

例子1：每个节点使用随机数给自己投票，并接收其它节点的投票，比较投票的大小，优先考虑数字较小的，把最小的作为下一个leader；这是一种非常简单考虑的算法；

例子2：类似拍卖，每个节点给出自身的报价，把报价发送给其它节点，当接收到其它节点报价比自身报价高，那么可以根据节点自身极限提高报价，把提高后的报价发送给其它节点，以此原则，直至大家都无法反抗一个价格（最高价格）时，那么该报价的节点就成为leader；

Zookeeper提供了三种选举策略：

* LeaderElection
* AuthFastLeaderElection
* FastLeaderElection（最快选举策略）

其中，FastLeaderElection（最快选举策略）是默认的选举策略；下面对FastLeaderElection过程进行介绍（参考：<https://www.cnblogs.com/scuwangjun/p/9063149.html>）

FastLeaderElection（最快选举策略）策略基础概念：

* Sid：节点SERVER的id；
* Zxid：事务id，数据越新，事务id越大；
* Epoch：投票轮次，每进行新一轮投票，epoch自增1；
* SERVER状态：looking（观望，想成为leader的状态）、leading（成为了领导者的状态）、following（成为跟随者的状态）、observing（成为观察者的状态）；

选举步骤：



1. 状态变更：当leader崩溃或宕机后，非观察者的server会把自身的状态改为looking，此时，大家都想成为leader；
2. 发起投票：投自己一票，把自己的（sid、zxid、epoch、SERVER状态）当作选票发送给其它的SERVER；
3. 接收并检查投票：接收来自其它SERVER的投票，检查是否是本轮的投票和投票的SERVER状态是否是looking；
4. 处理投票：检查完其它选票后，按以下原则选择下一轮的投票：

* 对接收到的投票进行比较，优先选择zxid较大的选票；
* 如果zxid相同，那么选择sid较大的选票；

根据以上原则选择了合适的投票后，进入下一轮选票；

1. 统计投票：如果接收到的投票中，有过半的投票都指向了同一个SERVER，那么可以确认该SERVER将作为leader；
2. 确认leader后，非观察者的SERVER把自身的状态更新为leading或following；

# zookeeper的数据存储和读取



以上是数据的大概交互，但是并没有说明的很细，例如：

* 跟随者把请求交由给leader后如何处理？
* follower和observer的职责如何区分？
* leader和follower的数据如何同步？