ZooKeeper的三种角色

上周有人和我说，有篇讲怎么调zk API的水文都上了csdn首页，我表示相当无奈，毕竟大多数人看一门技术就是想知道怎么调API，很少有人去了解内部机制。而懂不懂原理，恰恰是码农和架构师的最大区别。我肯定也会讲API，不过是下下篇的内容，最近仍然要讲原理。

这次说的是ZooKeeper的三种角色，也就是ZooKeeper服务器的三种节点类型（需要区分这里的“节点”和名字空间的“节点”，完全不是同一个意思，曾经有一家公司的面试官问我zk里有几种节点，结果被我反问了，场面相当尴尬）：群首（leader），追随者（follower），观察者（observer）。

Leader

Leader作为整个ZooKeeper集群的主节点，负责响应所有对ZooKeeper状态变更的请求。它会将每个状态更新请求进行排序和编号，以便保证整个集群内部消息处理的FIFO。

这里补充一下ZooKeeper的请求类型。对于exists，getData，getChildren等只读请求，收到该请求的zk服务器将会在本地处理，因为由第一讲的ZAB理论可知，每个服务器看到的名字空间内容都是一致的，无所谓在哪台机器上读取数据，因此如果ZooKeeper集群的负载是读多写少，并且读请求分布得均衡的话，效率是很高的。对于create，setData，delete等有写操作的请求，则需要统一转发给leader处理，leader需要决定编号、执行操作，这个过程称为一个事务（transaction）。

事务的编号就不说了，ZAB一章中已经把zxid的格式说得很清楚，已经忘了的可以回头查阅，重点来说说事务的执行。ZooKeeper事务和关系型数据库事务相似之处是都具备原子性，即整个事务（编号+执行）要么一起成功要么一起失败。另外事务还具备幂等性，即对一个事务执行多次，结果永远都是一致的。但ZooKeeper事务不具备关系型数据库事务的回滚机制，原因是不需要，因为ZAB协议已经保证消息是严格FIFO的，并且只有一个leader实际处理事务。（回忆两阶段提交2PC，之所以需要2PC的原因，归根结底是有不止一个“主”，必须保证这么多“主”看到的结果都是一致的）

另一个重要话题是leader选举，ZAB一章中已经提到有三种选举算法，目前默认的版本是FastLeaderElection，另两种已经被标记为deprecated。其过程如下：

- 数据恢复阶段。首先，每个ZooKeeper服务器先读取当前保存在磁盘的事务数据，从而得知当前自己能看到的最大zxid
- 首次发送自己的投票值。在读取数据之后，每个ZooKeeper服务器发送自己提议的leader，这个协议中包含了以下几部分的数据：

  1)所选举leader的id，在初始阶段，每台服务器的这个值都是自己的id
  2)服务器的最大zxid，因为FIFO原则，这个值越大说明该服务器离主越近
  3)逻辑时钟的值，也就是epoch，每次选举leader这个值会加1
  4)本机在当前选举过程中的状态，有以下几种：LOOKING，FOLLOWING，OBSERVING，LEADING

- 每台服务器将自己的上两种数据发送到集群中的其他服务器，同时也会接收来自其他服务器的这两种数据，此时如果该服务器的状态是在选举阶段(LOOKING状态)，那么首先要判断逻辑时钟值，分为以下三种情况：

1) 如果发送过来的逻辑时钟大于目前的逻辑时钟，那么说明这个提议比本机发起的提议更接近最新数据，此时需要更新本机的逻辑时钟值，同时将之前收集到的来自其他服务器的提议清空，因为这些数据已经不再有效了。然后判断是否需要更新当前自己的提议，这里是根据提议的leader id和保存的最大zxid来进行判断的，判断逻辑是：首先看zxid，大者胜出；其次再判断leader id，大者胜出。然后再将自身最新的提议数据广播给其他服务器

2) 发送过来的逻辑时钟小于本机的逻辑时钟，说明对方在一个相对较早的选举进程中，这时只需要将本机的提议发送过去即可

3) 两边的逻辑时钟相同，此时只是调用totalOrderPredicate函数判断是否需要更新本机的数据，如果更新了再将自己最新的提议广播出去即可

- 以上三种情况处理完毕之后，再执行两个判断：

1)判断是不是已经收集到了所有服务器的提议状态，如果是那么根据选举结果设置自己的角色（FOLLOWING还是LEADER），然后退出选举过程

2)即使没有收集到所有服务器的提议状态，也可以判断一下根据以上过程之后最新的选举leader是不是得到了超过半数以上法定人数的支持，如果是，那么尝试在200ms内（默认）接收一下数据，如果没有新的数据到来，说明大家都已经默认了这个结果，同样也设置角色退出选举过程

- 如果所接收服务器不在选举状态，也就是在FOLLOWING或者LEADING状态，做以下两个判断：

1) 如果逻辑时钟相同，将该数据保存到recvset，如果接收服务器处于LEADING状态，那么将判断是不是有半数以上的服务器选举它，如果是则设置选举状态，退出选举过程

2) 否则这是一条与当前逻辑时钟不符合的消息，说明在另一个选举过程中已经有了选举结果，于是将该选举结果加入到outofelection集合中，再根据outofelection来判断是否可以结束选举，如果可以也保存逻辑时钟，设置选举状态，退出选举过程

Follower

Follower的逻辑就比较简单了。除了响应本服务器上的读请求外，follower还要处理leader的提议，并在leader提交该提议时在本地也进行提交。Follower处理提议的过程已经在ZAB一章中描述过了。

另外需要注意的是，leader和follower构成ZooKeeper集群的法定人数，也就是说，只有他们才参与新leader的选举、响应leader的提议。

Observer

如果ZooKeeper集群的读取负载很高，或者客户端多到跨机房，可以设置一些observer服务器，以提高读取的吞吐量。Observer和Follower比较相似，只有一些小区别：首先observer不属于法定人数，即不参加选举也不响应提议；其次是observer不需要将事务持久化到磁盘，一旦observer被重启，需要从leader重新同步整个名字空间。