

昵称：47号Gamer
园龄：2年10个月
粉丝：4
关注：3
+加关注

<2021年12月>						
日	一	二	三	四	五	六
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类

Java8(11)
Java基础(31)
JVM(7)
MyCat(6)
MySQL(21)
Netty(5)
Spring(24)
SpringBoot(29)
zookeeper(17)
分布式(12)
其他(15)
设计模式(22)
数据结构(6)
算法(5)
线程(27)
消息中间件(10)
转载(1)

随笔档案

2021年8月(2)
2021年7月(7)
2021年4月(1)
2021年3月(3)
2021年2月(6)
2021年1月(5)
2020年12月(11)
2020年11月(17)
2020年10月(59)
2020年9月(63)
2020年8月(27)
2020年7月(13)
2020年6月(35)

zookeeper原理之选举流程分析

前面分析这么多，还没有正式分析到 leader 选举的核心流程，前期准备工作做好了以后，接下来就开始正式分析 leader 选举的过程：

```
1 public synchronized void start() {  
2     loadDataBase();  
3     cnxnFactory.start();  
4     startLeaderElection();  
5     super.start(); //启动线程  
6 }
```

很明显，super.start() 表示当前类 QuorumPeer 继承了线程，线程必须要重写 run 方法，所以我们可以 在 QuorumPeer 中找到一个 run 方法。

QuorumPeer.run

这段代码的逻辑比较长。粗略看一下结构，好像也不难

PeerState 有几种状态，分别是

1. LOOKING，竞选状态。
2. FOLLOWING，随从状态，同步 leader 状态，参与投票。
3. OBSERVING，观察状态,同步 leader 状态，不参与投票。
4. LEADING，领导者状态。

对于选举来说，默认都是 LOOKING 状态，只有 LOOKING 状态才会去执行选举算法。每个服务器在启动时都会选择自己做为领导，然后将投票信息发送出去，循环一直到选举出领导为止。

```
1 @Override  
2 public void run() {  
3     setName("QuorumPeer" + "[myid=" + getId() + "]" + cnxnFactory.getLocalAddress());  
4     // ... 根据选举状态, 选择不同的处理方式  
5     while (running) {  
6         switch (getPeerState()) {  
7             case LOOKING:  
8                 LOG.info("LOOKING");  
9                 // 判断是否为只读模式,通过"readonlymode.enabled"开 启  
10                if (Boolean.getBoolean("readonlymode.enabled")) {  
11                    // 只读模式的启动流程  
12                } else {  
13                    try {  
14                        setBCVote(null);  
15                        // 设置当前的投票, 通过策略模式来决定当前用哪个选举算法来进行领导选举  
16  
17                        setCurrentVote(makeLEStrategy().lookForLeader());  
18                    } catch (Exception e) {  
19                        LOG.warn("Unexpected exception", e);  
20                        setPeerState(ServerState.LOOKING);  
21                    }  
22                }  
23                break;  
24                // ...后续逻辑暂时不用管  
25            }  
26        }  
27    }
```

FastLeaderElection.lookForLeader

开始发起投票流程

阅读排行榜

1. java8 Stream对List<Map>的分组合并操作(5614)
2. Hash表之ASL和不成功ASL的计算 (平均查找长度)(2954)
3. Java中的CompletableFuture超时使用(2904)
4. RabbitMQ消息最终一致性解决方案 (TCC方式) (2295)
5. Java8 中map中删除元素的简单方法(2069)

评论排行榜

1. Spring IOC基于注解容器的初始化(1)

推荐排行榜

1. Kettle大量数据快速导出的解决方案 (利用SQL导出百万级数据, 挺快的) (1)
2. 内部类与静态内部类详解(1)
3. Java8的CompletableFuture在方法内使用不当, 导致局部变量出现线程安全问题(1)
4. Java8 中map中删除元素的简单方法(1)
5. ArrayBlockingQueue原理分析讲解(1)

最新评论

1. Re:Spring IOC基于注解容器的初始化
前面基于注解的初始化一直说着好好的, 小结变成了基于Xml的初始化。

--吴小破

```
1 public Vote lookForLeader() throws InterruptedException {
2     try {
3         HashMap<Long, Vote> recvset = new HashMap<Long, Vote>();
4         HashMap<Long, Vote> outofelection = new HashMap<Long, Vote>();
5         int notTimeout = finalizeWait;
6         synchronized(this){
7             logicalclock.incrementAndGet();//更新逻辑时钟, 用来判断是否在同一轮选举周期
8             //初始化选票数据:这里其实就是把当前节点的 myid, zxid, epoch 更新到本地的成员属性
9             updateProposal(getInitId(), getInitLastLoggedZxid(), getPeerEpoch());
10        }
11        LOG.info("New election. My id = " + self.getId() + ", proposed zxid=0x" + Long.toHexString(proposedZxid));
12        sendNotifications();//异步发送选举信息
13        /*
14         * Loop in which we exchange notifications until we find a leader
15         */
16        //这里就是不断循环, 根据投票信息进行进行 leader 选举
17        while ((self.getPeerState() == ServerState.LOOKING) && (!stop)){
18            /*
19             * Remove next notification from queue, timeout after 2 times
20             * the termination time
21             */
22            //从 recvqueue 中获取消息
23            Notification n = recvqueue.poll(notTimeout, TimeUnit.MILLISECONDS);
24            /*
25             * Sends more notifications if haven't received enough.
26             * Otherwise processes new notification.
27             */
28            if(n == null){ //如果没有获取到外部的投票, 有可能是集群之间的节点没有真正连接上
29                if(manager.haveDelivered()){//判断发送队列是否有数据, 如果发送队列为空, 再发一次自己的选票
30                    sendNotifications();
31                } else { //在此发起集群节点之间的连接
32                    manager.connectAll();
33                }
34            }
35            /*
36             * Exponential backoff
37             */
38            int tmpTimeOut = notTimeout*2;
39            notTimeout = (tmpTimeOut < maxNotificationInterval? tmpTimeOut : maxNotificationInterval);
40            LOG.info("Notification time out: " + notTimeout);
41        }
42    }
43 }
```

选票的判断逻辑 (核心代码)

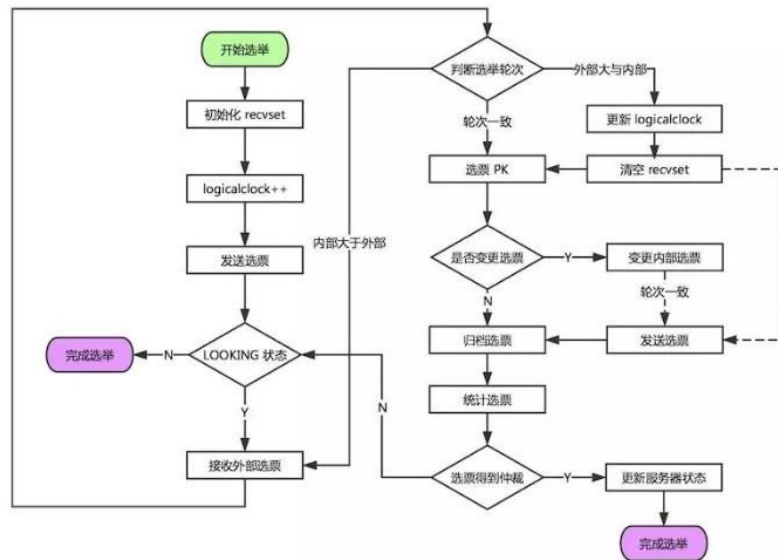
```
1 // 判断收到的选票中的 sid 和选举的 leader 的 sid 是否存在于我们集群所配置的 myid 范围
2 if (validVoter(n.sid) && validVoter(n.leader)) {
3     // 判断接收到的投票者的状态, 默认是 LOOKING 状态, 说明当前发起投票的服务器也是在找 leader
4     switch (n.state) {
5     case LOOKING: // 说明当前发起投票的服务器也是在找 leader
6         // 如果收到的投票的逻辑时钟大于当前的节点的逻辑时钟
7         if (n.electionEpoch > logicalclock.get()) {
8             logicalclock.set(n.electionEpoch);// 更新成新一轮的逻辑时钟
9             recvset.clear();
10            // 比较接收到的投票和当前节点的信息进行比较, 比较的顺序epoch, zxid, myid, 如果返回
11            // true, 则更新当前节点的票据(sid, zxid, epoch), 那么下次再发起投票的时候, 就不再是选自己了
12            if (totalOrderPredicate(n.leader, n.zxid, n.peerEpoch, getInitId(), getInitLastLoggedZxid(),
13                getPeerEpoch())) {
14                updateProposal(n.leader, n.zxid, n.peerEpoch);
15            } else { // 否则, 说明当前节点的票据优先级更高, 再次更新自己的票据
16                updateProposal(getInitId(), getInitLastLoggedZxid(), getPeerEpoch());
17            }
18            sendNotifications();// 再次发送消息把当前的票据发出去
```

```

19     } else if (n.electionEpoch < logicalclock.get()) { // 如果小于, 说明收到的票据已经过期了, 直接把这张票去掉
20         if (LOG.isDebugEnabled()) {
21             LOG.debug("Notification election epoch is smaller than logicalclock. n.electionEpoch = 0x"
22                 + Long.toHexString(n.electionEpoch) + ", logicalclock=0x"
23                 + Long.toHexString(logicalclock.get()));
24         }
25         break; // 这个判断表示收到的票据的 epoch 是相同的, 那么按照 epoch.zxid.myid
26             // 顺序进行比较比较成功以后, 把对方的票据信息更新到自己的节点
27     } else if (totalOrderPredicate(n.leader, n.zxid, n.peerEpoch, proposedLeader, proposedZxid,
28         proposedEpoch)) {
29         updateProposal(n.leader, n.zxid, n.peerEpoch);
30         sendNotifications();// 把收到的票据再发出去告诉大家我要选 n.leader 为 leader
31     }
32     if (LOG.isDebugEnabled()) {
33         LOG.debug("Adding vote: from=" + n.sid + ", proposed leader=" + n.leader + ", proposed zxid=0x"
34             + Long.toHexString(n.zxid) + ", proposed election epoch=0x"
35             + Long.toHexString(n.electionEpoch));
36     }
37     // 将收到的投票信息放入投票的集合 recvset 中, 用来作为最终的 "过半原则" 判断
38     recvset.put(n.sid, new Vote(n.leader, n.zxid, n.electionEpoch, n.peerEpoch));
39     // 判断选举是否结束
40     if (termPredicate(recvset, new Vote(proposedLeader, proposedZxid, logicalclock.get(), proposedEpoch)) {
41         // 进入这个判断, 说明选票达到了 leader 选举的要求
42         // 在更新状态之前, 服务器会等待 finalizeWait 毫秒时间来接收新的选票, 以防止漏下关键选票如果收到可能改变
43         // Leader 的新选票, 则重新进行计票
44         while ((n = recvqueue.poll(finalizeWait, TimeUnit.MILLISECONDS)) != null) {
45             if (totalOrderPredicate(n.leader, n.zxid, n.peerEpoch, proposedLeader, proposedZxid,
46                 proposedEpoch)) {
47                 recvqueue.put(n);
48                 break;
49             }
50         }
51         // 如果 notification 为空, 说明 Leader 节点是可以确定好了
52         if (n == null) {
53             // 设置当前节点的状态(判断 leader 节点是不是我自己, 如果是, 直接更新当前节点的状态为
54             // LEADING)否则, 根据当前节点的特性进行判断, 决定是FOLLOWING 还是 OBSERVING
55             self.setPeerState((proposedLeader == self.getId()) ? ServerState.LEADING : learningState());
56             // 组装生成这次 Leader 选举最终的投票的结果
57             Vote endVote = new Vote(proposedLeader, proposedZxid, logicalclock.get(), proposedEpoch);
58             leaveInstance(endVote);// 清空
59             return endVote; // 返回最终的票据
60         }
61     }
62     break;
63     case OBSERVING:// OBSERVING 不参与 leader 选举
64         LOG.debug("Notification from observer: " + n.sid);
65         break;
66     case FOLLOWING:
67     case LEADING:
68         /*
69          * Consider all notifications from the same epoch together.
70          */
71         if (n.electionEpoch == logicalclock.get()) {
72             recvset.put(n.sid, new Vote(n.leader, n.zxid, n.electionEpoch, n.peerEpoch));
73
74             if (ooePredicate(recvset, outofelection, n)) {
75                 self.setPeerState((n.leader == self.getId()) ? ServerState.LEADING : learningState());
76                 Vote endVote = new Vote(n.leader, n.zxid, n.electionEpoch, n.peerEpoch);
77                 leaveInstance(endVote);
78                 return endVote;
79             }
80         }
81         /*
82          * Before joining an established ensemble, verify a majority is

```

投票处理的流程图



这个方法是使用过半原则来判断选举是否结束，如果返回 true，说明能够选出 leader 服务器 votes 表示收到的外部选票的集合 vote 表示当前服务器的选票。

QuorumMaj. containsQuorum

```
1 public boolean containsQuorum(Set<Long> set){
2     return (set.size() > half);
3 }
```

这个 half 的值是多少呢？

可以在 QuorumPeerConfig.parseProperties 这个方法中，找到如下代码。

```
398 LOG.info("Defaulting to majority quorums");
399 quorumVerifier = new QuorumMaj(servers.size());
```

也就是说，在构建 QuorumMaj 的时候，传递了当前集群节点的数量，这里是 3 那么，half=3/2=1

```
1 public QuorumMaj(int n){
2     this.half = n/2;
3 }
```

那么 set.size()>1. 意味着至少要有两个节点的票据是选择你当 leader，否则，还得继续投。

分类: [zookeeper](#)

[好文置顶](#)[关注我](#)[收藏该文](#)



47号Gamer\

关注 - 3

粉丝 - 4

[+加关注](#)

« 上一篇: [SpringBoot 过滤器，监听器，拦截器详解](#)

» 下一篇: [zookeeper原理之投票的网络通信流程](#)

0


 推荐

0

 反对

posted @ 2020-08-20 22:52 47号Gamer\ 阅读(138) 评论(0) 编辑 收藏 举报

[刷新评论](#) [刷新页面](#) [返回顶部](#)

 登录后才能查看或发表评论，立即 [登录](#) 或者 [逛逛](#) [博客园首页](#)

编辑推荐：

- 带团队后的日常思考（六）
- 聊聊云原生和微服务架构
- 异或运算的巧用：不用额外的变量，如何交换两个变量的值？
- OAuth 2.1 带来了哪些变化
- 理解微前端技术原理



最新资讯：

- 夺走 20 多亿用户「唱反调」的声音，全球最大的视频网站凭什么？（2021-12-02 17:33）
 - vivo T1 体验：高能实力派，还有颜，妥妥的C位（2021-12-02 17:22）
 - 2021互联网文学盘点：年轻人集体发疯，中年人爱凡尔赛（2021-12-02 17:11）
 - Meta开源全新移动端AI生成神器PyTorch Live，造个AI应用5分钟，安卓iOS都支持（2021-12-02 17:00）
 - 和「咸鱼」做邻居！从虹宇宙看国内元宇宙应用的真相（2021-12-02 16:53）
- » 更多新闻...

