黄欣健:20:32:30

RecvWorker

黄欣健:20:36:36

有一些可能是旧的 epoch？

黄欣健:20:37:58

变更投票

Ghost:20:41:10

2

杨腾飞:20:41:12

7？

黄欣健:20:41:12

78

答：准确来说是 8 7

黄欣健:20:43:17

leader 已确定

Ghost:20:43:57

想不清楚 我都大于一半了 怎么还会有比我还大的

黄欣健:20:44:49

因为有可能你初始化投出去的，后面你在比较的时候，发现别人比你大，你就会变更重投

李鹏建:20:44:54

有更适合的，投票会变化

李鹏建:20:47:54

嗯嗯

Ghost:20:50:19

就是这个合适的判断不是 大于一半的判断 而是 zxid 和年代号的判断 适合

答: 对的,就是这个意思

杨腾飞:20:51:32

剩余的省去了过半判断

答:剩余的没有省去,还要重新 put 回去.

黄欣健:20:52:42

其实每一次最外层的 while 循环的这一步逻辑也会执行，只不过之前是 tremPredicate 方法进不去

答: 对的

杨腾飞:20:53:34

嗯

黄欣健:20:58:50

那这个 endvote 就只是用来记录日志的

解答: 要看是谁调用了它

大胜.彼得:20:59:02

所有被推荐者 只有一个会选票过半 对吧？

黄欣健:20:59:41

哦哦，看漏了

黄欣健:21:01:47

7

黄欣健:21:01:52

吧

大胜.彼得:21:02:22

是的话 就不进来了

湮汐:21:03:31

能结束

杨腾飞:21:03:50

9

余爽:21:03:50

7

大胜.彼得:21:03:52

7

湮汐:21:03:54

7

没有这首歌。:21:03:57

8

大胜.彼得:21:05:13

空

余爽:21:05:16

1

大胜.彼得:21:05:17

1

Ghost:21:05:22

1

xxzx_4684222:21:05:25

1

xxzx_4684222:21:06:43

空了

黄欣健:21:06:44

空了

余爽:21:08:55

不能

余爽:21:09:03

还没更新票箱

余爽:21:09:22

不是

湮汐:21:10:59

7

xxzx_4684222:21:11:02

我们

任献良:21:11:04

推荐的

湮汐:21:11:29

有

Ghost:21:11:35

有

王助教:21:11:38

有

Ghost:21:11:42

原来的 8 吧

任献良:21:11:42

8 票有可能比我们大

余爽:21:12:26

没可能

任献良:21:13:54

那全票过了

王助教:21:14:33

有点像找出多个数字中的最大值

Free3610:21:16:12

晕了

Free3610:21:17:14

老雷可以用画图描述一下这个代码吗

答:这个逻辑不好画图,得自己 debug 慢慢跟才行

余爽:21:17:15

一轮过后，推荐都是最合适的

答: 对的

任献良:21:17:19

8 个 List> 里面找 一个最大数

Ghost:21:17:39

它这个通知 发出去 就还在这个方法中就接到了其他服务通知 还能放到队列里？？？ 这是咋子实现的呀？jmx？

答:这个过程不是我们讲的这段代码里的,这里讲的将更新过的提案广播发送出去,也就是 sendNotifications();方法. 具体的发送逻辑不在这里,这里也仅仅只是把消息放到队列里 sendqueue.offer(notmsg);

余爽:21:18:13

第一轮，大家都是推荐的自己，所以总会有最合适的

任献良:21:19:51

那两轮就搞定了 这样

黄欣健:21:19:55

雷哥,有没有这样一种情况,就是服务器 A 一开始毛遂自荐,然后发现后面接二连三的选票都比自己推荐的大，那 A 就会一直发出新的投票吗

答:是的

黄欣健:21:20:58

如果是这样，那别的服务器也会一直收到 A 的新选票的话，会不会存在了多个过半

答: 不会

任献良:21:21:06

zxid 最小的 最苦逼，最累了

答: 一般情况下 zxid 都是相等的,最苦逼最累的是 myid

湮汐:21:21:22

健哥，不会的。因为它是把票放到 hashmap 里面

大胜.彼得:21:22:55

不会多个 因为是 map 会覆盖

任献良:21:23:45

这个队列没顺序吗？

答: 顺序跟网络有关系,比如高速链路有可能就是最新接收到的.

任献良:21:24:52

1

任献良:21:25:10

还的拼人品

大胜.彼得:21:25:19

感觉 队列顺序 会影响投票的轮数

答: 对的

大胜.彼得:21:26:58

临时工

任献良:21:27:11

不可能有这种情况

小孟-北京-4 年:21:27:31

ob 没有选举权

任献良:21:29:18

写的严谨怕出 bug

黄欣健:21:29:22

如果杀掉进程修改状态重新启动咧

石将从:21:29:42

为了代码逻辑的完整性吧

湮汐:21:29:46

我觉得作者并不是一遍就把这个逻辑写通过的，这些复杂的

逻辑，也是写完后不断调试的，所以之前肯定有些无效代码在这里。

杨腾飞:21:29:50

更多是语义上的考虑吧

任献良:21:30:24

也有可能是当时出了 bug 后面上线这块没去掉

黄欣健:21:30:57

如果杀掉进程修改状态重新启动，连接时，这个 id 会更新在验证里面的集合

黄欣健:21:31:02

吗

答：状态是不能随便更新的,一开始写在配置文件里面的

xxzx_4684222:21:33:10

写在配置文件里的，角色

黄欣健:21:33:21

对哦，每个都要一起配置

李鹏建:21:35:08

只能说 if 语句进不去，但是 state 里面要有，保证统一性吧

李鹏建:21:35:21

我猜的

大胜.彼得:21:41:06

大佬们，现在 zk 外面用的多吗

湮汐:21:41:55

比较多，有写组件必须依赖 ZK

湮汐:21:42:01

比如 kafka

湮汐:21:42:36

当当的 elastic job 也是必须依赖 zk

任献良:21:42:53

kafka consumer 没有数据 也有很大流量 好奇怪

攀登:21:46:48

ack

攀登:21:47:15

我收到，就会回复

Ghost:21:47:28

；礼貌

攀登:21:50:27

observer 只负责转发，收到 observer 信息的节点会忽略掉，不回复。

答: 不是的,发的时候就不给 observer 发.

大胜.彼得:21:51:01

选举时候 ob 不参与

答: 对的

溷汐:21:51:47

老师，每个 Server 接到了其他的通知就会向那个 server 发送自己的通知。这个过程会不会一直发下去：你发给我，我回给你；你又接到了我的，又发给我，一直递归下去？

答: 过程是这样,但是 zk 的代码肯定是会避免递归的问题的.

攀登:21:51:56

对对，这个是选举 leader 类

黄欣健:21:52:46

这个在选举结束的时候就可以停止了吧？

余爽:21:52:53

投票通过之后

xxzx_4684222:21:55:55

我们搭建集群的时候三台 server，第三个 server 启动的时候，前两个 server 已经选出 leader 了，是不是就是这种场景

黄欣健:21:57:29

不知道意思是他们的循环还没跑完的意思吗

湮汐:22:00:40

感觉 zk 源码比 spring 难好多啊

Ghost:22:01:04

更恐怖的是 我们只看了一个方法

小孟-北京-4 年:22:02:39

2 台，咋选出 leader 啊，过半了吗

答: 两台都选一个

黄欣健:22:03:13

2 台启动的时候，配置文件配置了三台

大胜.彼得:22:03:13

2 台先启的

Ghost:22:03:15

都说用 zk 做高可用 咋子做高可用？在后面吗

答: 集群环境下,宕机了几台,又不影响剩余的几台 zk 工作,这

就是高可用了.

黄欣健:22:03:59

zk 不是 cp 吗，不能满足 a

答: zk 是 cp 的,但是 cap 中的 a 跟 Ghost 提到的高可用指的不是同一个东西.

小孟-北京-4 年:22:05:05

zk 选举的时候要暂停对外服务

答: 对的

大胜.彼得:22:05:11

容灾能力

任献良:22:09:29

leader 都出来了，直接更新不就好了

答: 不知道 leader 出来了,没办法更新,现在在做的就是设法知道 leader 出来了.

大胜.彼得:22:09:43

它不知道出来了

任献良:22:09:55

不是有 leading 状态

杜庆奎:22:10:20

投石问路

黄欣健:22:11:23

状态

溧汐:22:11:45

leading 状态的 server 发出来的票 难道有可能不是 leader 吗？

杨腾飞:22:12:18

不是没有可能，只是还没结束

杨腾飞:22:19:14

发现 leading，TRUE

黄欣健:22:19:54

上一轮选的

任献良:22:27:24

参数传一个 outofelection 不就行了，为啥传两个

答: oopPredicate 方法就是要处理两个集合的

黄海-成都:22:27:36

什么情况下，场景 1 的第一个条件会不满足？

答: 第一票的时候进来,这时候没过半,就不满足

xxzx_4684222:22:28:47

第一次肯定不满足

大胜.彼得:22:30:02

循环一半之前 肯定不过半

黄海-成都:22:30:12

不哦

任献良:22:30:59

懵了

xxzx_4684222:22:31:03

逻辑要细品，绕的很

黄海-成都:22:31:17

场景 1，新加入节点，就是收到 foller，推荐 1 是 leader，难道这还有假？

Free3610:22:31:17

老雷，我彻底蒙圈了

大胜.彼得:22:31:20

场景一 self 的 id 不能是 lead 把

答: 是的,只是那个方法需要做这么一个逻辑判断而已

余爽:22:31:21

有不过半的可能，比如脑裂了

Ghost:22:31:36

我现在的情况是 都不知道我懂没懂

大胜.彼得:22:31:51

1071 行

黄欣健:22:32:09

leader 还不知道自己是 leader

答: 这是场景二的情况.

大胜.彼得:22:32:47

那 1071 那个 3 元运算 就不能是 lead

杜庆奎:22:32:47

如果场景二收到 leader 的反馈了，已经明确 Leder 了为什么还要判断票数过半呢？

答: 那个 leader 是其它人推荐的 leader,而且有可能不在同一轮.

黄欣健:22:33:57

不过如果它已经收到了 leader 的选票，直接可以得到 leader

的状态就可以直接退出了吧？

答: 不是的,没办法确定是 leader 还是 following 发出的.还是需要通过 ooePredicate 方法判断.

余爽:22:34:58

这个 leader 是你们选的，我不一定认可

黄欣健:22:35:36

例如在 ooePredicate 方法的短路与方法那里，里面 new vote 有个 n.state

答: 这么判断是可以,还是会出现不是同一轮的情况.

黄欣健:22:35:52

先判断 n.state 不可以的吗

黄海-成都:22:36:00

就是人家已经选出来 leader 了，但是还是啥要自己判断下，虽然判断结果肯定是一样的。

答: 大多数情况下结果是一样的.

大胜.彼得:22:36:13

不需要区分吧，当前 n.leader 就是

杜庆奎:22:37:06

也就是说我自己已经放弃竞选了，只是在找我的老大是谁吧

任献良:22:37:19

如果我的更适合的话，是不是 leading 状态还要再修改？

大胜.彼得:22:37:32

可能还是为了合法性，它不相信别人的 n 信息，总想通过合法选票判断谁是 leader

余爽:22:38:06

比如出现脑裂？

湮汐:22:38:15

我觉得有可能是 一个刚刚当上 leader,然后马上死了

杜庆奎:22:38:16

没有

Ghost:22:38:46

死了 心跳没有了 就都是 look 了吧

黄欣健:22:38:53

猫哥，它完全可以在 ooePredicate 里面的短路与时，先判断 n.state 的吧，因为场景二的第一个条件就是判断 epoch，这样就不用判断是否过半了吧？

答: 有可能两种场景融到一个方法里面去判断了.

黄欣健:22:39:09

打错，雷哥

黄欣健:22:39:16

哈哈

大胜.彼得:22:39:22

好尴尬

张中强:22:39:35

拉出去砍了

黄欣健:22:40:05

如果能提高性能，分开两个方法不是更好吗

湮汐:22:41:35

有可能，zk 选举压测也不好做

湮汐:22:41:45

否则自己改下代码 压测看看会不会出现

杜庆奎:22:43:00

且听下回分解

黄欣健:22:43:03

压测俺也不会

Ghost:22:43:58

zk 还有两节课。。。。

柱子-深圳:22:44:15

懵啦

Ro-深圳-3 年 Java:22:44:23

一脸懵逼

Ghost:22:44:25

搞不搞的完呦

黄欣健:22:44:27

似懂非懂，得靠自己把流程说出来可能才算懂

xxzx_4684222:22:44:28

打脑壳

湮汐:22:44:31

zk 发送消息这里会讲吗

答: 没有

Ghost:22:44:35

183 脸懵

湮汐:22:45:27

老师能不能提供几个 zk 主要逻辑的源码类和方法作为入口，我们线下自己看？

答: 下次课会说到一些.

任献良:22:45:41

再来两边估计差不多

黄欣健:22:46:06

那个哥们就是这个哥们

湮汐:22:46:24

不知道怎么找入口

湮汐:22:46:39

还行

任献良:22:46:41

懵

石将从:22:46:44

懵

杜庆奎:22:46:45

高潮迭起

Ghost:22:46:56

讲的是挺好 听的是。。。

湮汐同学单独提出的问题:

1、 zk 除了选举，还有哪些代码是比较关键的，可以值得去读的（包括老雷不会讲，但是属于 zk 非常重要的逻辑的代码），这些老雷老师能整理给我们吗？

答: 在 zk 的面试中一般会问到的问题集中在理论上，例如，paxos 算法、zab 协议。但也会涉及到源码中的一些问题，例

如，client 在启动时连接的是 zk 集群中的哪台主机？再如，zk 对于 client 连接空闲超时管理采用一种称为“分桶策略”的方式，其原理是什么？像这两个问题是涉及源码问的最多的。当然，我们都会讲的。

如果学员还想再多读些 zk 源码，可以再研究一下 client 是如何实现对节点的创建、删除、修改等操作的。但这些在面试中较少会问到。

2、 读代码的时候，如果读到某个接口的方法调用，而这个接口有多个实现类，如果不断点调试（zk 这种就不太容易调试），怎么判断调用的是哪个实现类？

答：至于第二个问题，这个就不好回答了。因为，所有在框架中出现的接口实现类都有其对应的应用场景，不同场景会调用不同实现类。若其不会执行该类，也不会出现在框架中。只要出现，一定有用。

问题一：

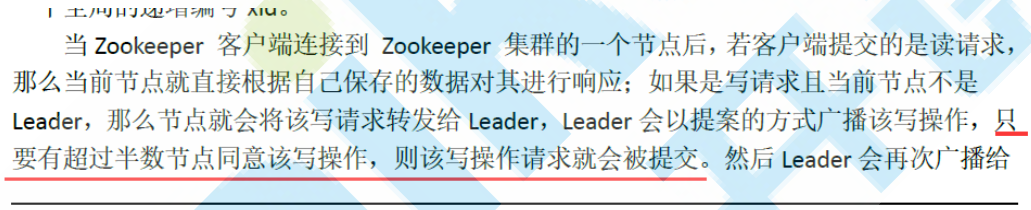
Zookeeper 使用一个单一主进程来接收并处理客户端的所有事务请求，即写请求。当服务器数据的状态发生变更后，集群采用 ZAB 原子广播协议，以事务提案 Proposal 的形式广播到所有的副本进程上。ZAB 协议能够保证一个全局的变更序列，即可以为每一个事务分配一个全局的递增编号 xid。

Paxos 算法是先 prepare，再 accept（这个阶段发送 Proposal），最后 commit。

“ZAB 协议以事务提案 Proposal 的形式广播到所有的副本进程上”，这里的意思是 ZAB 协议它没有 prepare 阶段吗，ZAB 是直接进入 accept 阶段？

答: ZAB 协议是 3PC 的，当然有 prepare。讲义中的意思是在通过了 prepare 阶段后再进入 accept 阶段，将 Proposal 广播给副本进程。当本阶段也通过了再发送 commit，让副本进程将 Proposal 同步到本地。

问题二：



文章中超过半数，写操作请求就会被提交。

所以根据我的理解，剩下的没有“同意”的，或者说“没来得及同意”的 follower，这时候状态就不是 following，它不能对外提供服务，必须通过后续的心跳来同步数据。这种理解是否正确？

答: 这里不存在 Following 状态，只有当 leader 选举时才会有 following 状态。这里是对提案进行表决。在进行事务同步时，集群也是对外提供服务的，只不过客户端访问不到还未完成同步的最新数据。

问题三：

当 Leader 收到超过半数 Follower 的 ACKs 后，就向各个 Follower 广播 COMMIT 消息，批准各个 Server 执行该写操作事务。当各个 Server 在接收到 Leader 的 COMMIT 消息后就会在本地执行该写操作，然后会向客户端响应写操作成功。

写请求的全流程是怎样？

是半数 follower 写成功，就响应给 client；还是要全部 follower 写成功才响应？

响应流程是 leader 直接响应 client，还是根据请求流程原路返回（leader 先响应 client 连接的 learner，由 learner 再响应给 client）？

答：让学员听“zookeeper-015-Observer 的数量问题.avi”录播视频可以解决该问题。

响应是原路返回的。

整个 ZAB 协议，分为两个阶段

1、消息广播

2、崩溃恢复

1、消息广播：根据网上所说，其实是 2PC 不是 3PC，没有 prepare 阶段，上来就是 accept 阶段，直接提交 proposal（我没看源码，所以我也不清楚老雷的回答和网上所说哪个更对）

2、崩溃恢复：它又包含两个阶段

2.1、选举阶段

2.2、数据同步阶段

2.1、选举阶段：就是老雷给我们讲的源码，跟 2PC，3PC 没啥关系，也和 PAXOS 没啥关系。其过程就是首先“我选我”，然后不断的从队列里面 poll 选票，判断是否需要更新自己的选票，统计选票是否过半。不管选票是 looking,following 或者是 leading 状态的，我都会去统计选票，看看选票是否过半，最终返回选出的 leader

数据同步阶段这个没讲，自己也没去分析，就不表态了。

答:ZAB 中的消息广播即非纯粹的 2PC 与非纯粹的 3PC，因为 2PC 是存在很多问题的，例如同步阻塞、单点等。但 3PC 不存在这些问题。若单从 leader 发起次数来说是发起了 2 次，但其并不是简单的 2PC，实际是对 2PC 与 3PC 的综合。
另外，Leader 选举是典型的 paxos 算法：所有参与者均可提交提案，由所有表决者对该提案进行表决。它是典型的 2PC：提交提案（首先是我选我）后若大家同意，则提交 commit（leader 广播 epoch 等信息）。