Report

张磊 2017K8009922027

1

大家好，今天我给大家汇报 ZooKeeper 源码分析的第二部分，Leader 选举的核心流程设计。

3

这里，是今天有关于我今天要汇报的内容的 ZooKeeper 的类图，由于整个 ZooKeeper 里的类太多，类与类之间的关系太复杂，所以我只截取了最重要的一部分。

在 FastLeaderElection 这个类里，包含有 manager，sendqueue，recvqueue 这些属性，以及 sendnotification，totalOrderpredicate，updateproposal，lookforleader 这些方法，当然，还有其他几个类的一些属性和暴露的方法，这些都会在我接下来的汇报中讲到。

4

此外，我们还会讲到一些关于 ZooKeeper 启动的逻辑，这里是一些和我待会的汇报有关的类的类图。

一台 ZooKeeper 服务器的启动，就是从启动一个 QuorumPeer 对象开始的。

5

其实我今天的汇报内容，就是这么一张简单的图，当我从空白开始，把这张图画完，我的汇报也就基本结束了。

6

这张图里的东西可能有点多，那么我们概括一下，这张图就更简单了。**ZooKeeper** 中的 **Leader** 选举逻辑可以分为 **2** 个层次，一个是产生投票的应用层，另一个则是负责发送和接收投票的传输层。

7

在 **QuorumPeerMain** 中有一个 **main** 方法，调用这个 **main** 方法，我们就可以启动一台 **ZooKeeper** 服务器。

Main 方法，主要的功能就是调用 **initialize** 方法，根据我们事先设置好的配置文件，初始化服务器的基本属性，这个 **args** 参数，其实就是我们的配置文件的路径，右边的这张图就是一台 **ZooKeeper** 服务器的配置文件，里边定义了 **tickTime**，数据存储的目录，以及一些端口信息，服务器的角色信息**(Obeserver)**。

8

当配置文件中服务器的个数超过 **0** 时，服务器就会以集群模式启动。单机模式是不会存在 **Leader** 选举的情况的。

9

在 runFromConfig 中做的就是根据配置文件进行初始化，设置好 QuorumPeer 中的一些属性，然后调用 start 方法启动这个类（线程）。

10

在 start 方法中，服务器首先会加载数据库，把我们要用的数据都加载到服务器中，保证我们能够使用，然后调用 cnxFactory 的 start 方法。

11

这个 cnxFactory 的 start 方法会启动一个用于接收客户端请求的线程，但是由于现在还没有选出 Leader，服务器的角色都还没有确定，服务器还不能对外提供服务。所以，就算这个时候有客服端发送请求过来，服务器也会把它拒绝掉。

12

然后就开始我们今天的主题，Leader 选举。

13

开始 Leader 选举之前，首先要初始化 Leader 选举的中需要用到的连接，以为后续的选举投票服务。我们之前说过，Leader 选举可以分为应用层和传输层两个层次。这里的 StartLeaderElection 做的其实就是初始化我们的传输层,建立服务器之间的用于传输投票的连接，选择一个选举算法。

14

每台服务器初始化的时候都生成一个给自己的投票，然后选择一个选举算法，算法也可以在配置文件中指定。

15

这里虽然有 4 个选举算法，但是其实前 3 个都过期了，现在使用的基本都是 FastLeaderElection，如果没有在配置文件中指定，那么默认采用的也是 FastLeaderElection。

采用 FastLeaderElection 时，服务器会再创建并启动一个 listener 线程，用于监听其它的服务器。后续会进一步完善。

16

现在，我们要画的图是这样的。

17

然后我们调用 createCnxnManager 方法，创建这个 listner。

18

这里我们会启动一个 SendWork 线程，用于发送选票。这里采用了 hashMap 的结构，是为了向各台服务器发送消息时检索的能够快一些。queueSendMap 用于存放要给各台服务器发送的消息。

recvQueue 用于存放本服务接收到的消息。

19

创建完成后，我们就返回选择选举算法那里，listener.start 会启动这个 listner 线程。这个线程会创建一个 Socket 连接，这个 Socket 就是用来完成 Leader 选举过程中的选票的发送和接收的。这个连接会和服务器之间的监听端口绑定，比如之前配置文件里的 3888 端口。

20

这个 listener 线程启动之后，我们的图就变成这样了。

虽然每个服务器都建立了 Socket 连接，并且都开始监听，但是服务器和服务器之间还没有连接上。

21

选择完选举算法后，我们还要去初始化一下这个算法需要用到的应用层的用于传输选票的东西。

22

在 FastLeaderElection 的 starter 方法中，LinkedBlockingQueue <ToSend>会创建一个应用层的发送选票线 Worker Sender，并将其与 sendqueue 绑定;

而 LinkedBlockingQueue<Notification>会创建一个应用层的接收选票线程 WorkRecvier，并将其与 recvqueue 绑定;

23

现在我们的图已经基本成型了，只是这个 Socket 还没有完全建立好，那我们接着看。

24

完成了选举需要的东西的初始化后，我们就可以开始选举了。也就是这里的 super.start 方法。

25

Super.start 其实就是把这个 QuorumPeer 跑起来，在这个服务器的 run 方法里有一个主循环，之后服务器就都是在这个主循环里运行了。

初始化的时候，参与选举的每台服务器（也就是除了配置文件中指定为 Obeserver 的服务器），的状态都是 LOOKING。所以会进到这个 Case(LOOKING)。

setCurrentVote 就是设置本台服务器的投票，这里他设置的投票就是函数 makeLEStrategy.lookForLeader 返回的。选举可能要花费一段时间，所以主循环会被阻塞在这里，直到选举完成。

26

原代码太长，看伪代码就简单多了，每个状态都干些什么事。

27

ServerState 有 4 个枚举值，分别对应 4 种集群角色的状态。

28

Leader 选举，我们要干 5 件事。一，先给自己投一票；二，接收其他服务器的投票；三，PK 竞选；四，可能需要该票；五，统计，完成最后的投票。

29

这里的 recvset 就是投票箱，用于存放选票。

updateProposal 就是提出投票提议，在这里，其实就是给自己投票的意思。

然后通过 sendNotification，发出自己的投票。

30

sendNotification 做的，就是将投票的信息封装成一个 ToSend notmsg，发送给别的服务器。这里的 sid 就表示要把这个消息发送给哪台服务器。

Sendqueue.offer 会选票放到 sendqueue 里，然后就会有相应的线程 WorkSender 负责把这个投票传送出去；

31

就是用红圈圈起来这里。

32

在发送选票的时候，如果发现是自己发给自己的投票，那么就直接放到自己的 recvqueue 里就可以了，否则，放到 sendqueeu 里。

33

然后，服务器就会进入一个 PK 选票的循环。不停的从 recvqueue 里取出选票，然后和自己当先投的服务器比较，从中选择一个自己认为更好的服务器。

第一次从 recvqueue 中拿选票时会拿到投给自己的选票，但是由于服务器之间的 Socket 连接还没有建立好，所以再次拿票时，拿到的票就会变成 null。

这里 manager.connectAll 就会真正完成我们的 Socket 的连接，将各台服务器之间的传输通路搭建好，这样我们就可以收到其他服务器发送过来的选票了。

34

下面我们来看一下 PK 的逻辑。

这里会分为 3 种情况，收到的选票的届号比自己大，收到的选票届号比自己小，以及届号相等。

如果收到选票的届号比自己的大，那么说明自己落后了，这时，会清空自己的投票箱，然后和自己比比看，根据比较结果更新投票的提议。

35

如果收到的选票的届号比自己的小，那么就直接忽略。

如果届号相等，那么就和自己的提议 PK，然后根据 PK 结果，如果收到的选票投的服务器更好，那么就更新投票提议，再把自己的新投票发送出去。

36

PK 的逻辑是这样的：

首先比较届号，收到选票的届号更高，返回 true，这一步在刚在判断代码里就已经完成了；

届号相同，但是收到选票的 zxid 更大，返回 true；

届号，zxid 都相同的话，收到的选票的 sid 更大，返回 true；

37

完成选票的 PK 后，就把自己的得出的选票放到投票箱里。

然后看一下，能否通过过半机制，从投票箱中选出 Leader。

如果可以选出 Leader 了，并且 recvqueue 中也没有其他服务器发出的选票，那么就说明 Leader 已经选择出来了。否则，如果 recvqueue 中还有其他服务器发出的选票，那么就还得接着 PK，直到没有其他服务器发出选票，此时，才能真正确定选出的 Leader。

38

过半机制很简单，其实就是判断这台服务器获得的选票是不是过半了。N 就是服务器的个数。

39

来到这里，说明 PK 已经完全结束了。这时，只要根据本机最终得到的选票，设置本机的角色状态，返回本机的最终投票就可以了。

40

到这里，所有的服务器，最终选出的 Leader 都是一样的，这里 Leader 选举就完成了。

41

最后，从 lookForLeader 中退出后，setCurrentVote 会将自己的 currentVote 设置为最终选出的 Leader。下一次循环，服务器就会根据自己的状态角色，进入不同的 CASE，初始化本机对应角色的事务，然后就可以开始对外提供服务了。

42

最后，（如果还有时间的话）我们一起来看一下简短的视频。

这个是 RAFT 分布式一致性算法的 Leader 选举的视频，它和 ZooKeeper 的很像。

我今天的汇报到此结束，谢谢大家。