47号Gamer丶

昵称： 47号Gamer丶
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

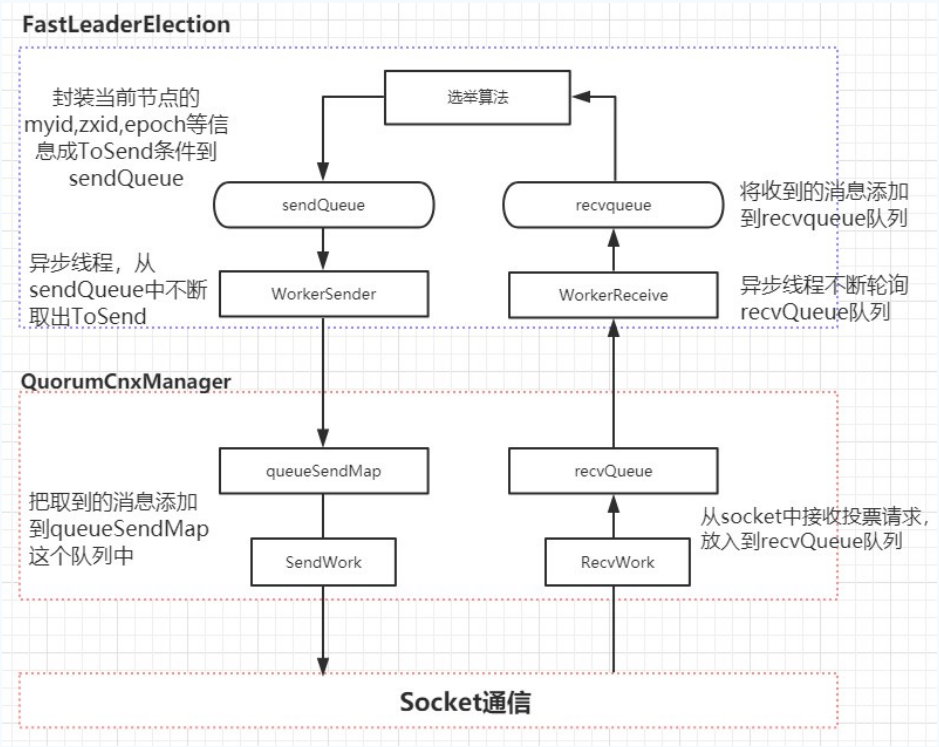
常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签

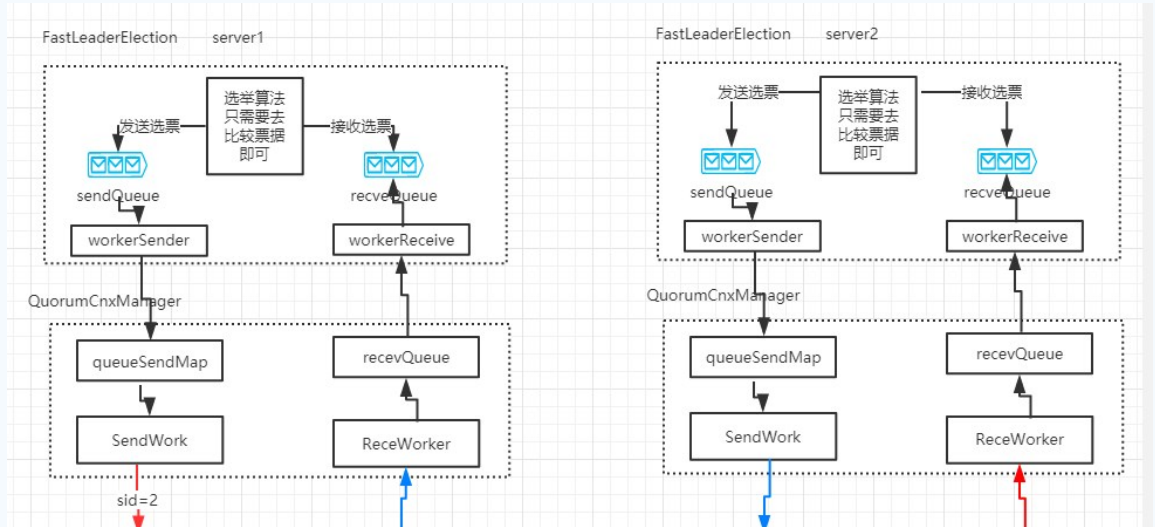
随笔分类
Java8(11)
Java基础(31)
JVM(7)
MyCat(6)
MySQL(21)
Netty(5)
Spring(24)
SpringBoot(29)
zookeeper(17)
分布式(12)
其他(15)
设计模式(22)
数据结构(6)
算法(5)
线程(27)
消息中间件(10)
转载(1)

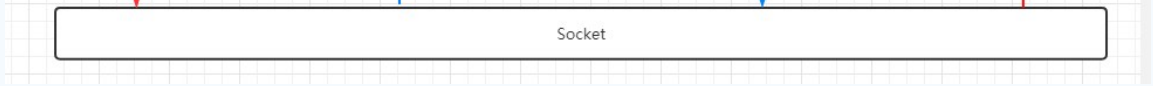
随笔档案
2021年8月(2)
2021年7月(7)
2021年4月(1)
2021年3月(3)
2021年2月(6)
2021年1月(5)
2020年12月(11)
2020年11月(17)
2020年10月(59)
2020年9月(63)
2020年8月(27)
2020年7月(13)
2020年6月(35)

zookeeper原理之投票的网络通信流程

通信流程图：







接收数据 Notification 和发送 ToSend

ToSender	Notification
leader； 被推荐的服务器 sid zxid; 被推荐的服务器当前最新的事务 id peerEpoch; 被推荐的服务器当前所处的 epoch electionepoch； 当前服务器所处的 epoch stat 当前服务器状态 sid 接收消息的服务器 sid（myid）	leader； //被推荐的服务器 sid zxid； 被推荐的服务器最新事务 id peerEpoch; 被推荐的服务器当前所处的 epoch electionEpoch 选举服务器所处的 epoch stat； 选举服务器当前的状态 sid； 选举服务器的 sid

通信过程源码分析

每个 zk 服务启动后创建 socket 监听

```
protected Election createElectionAlgorithm(int electionAlgorithm){
    //….
    case 3:
        qcm = createCnxnManager();
        QuorumCnxManager.Listener listener =
        qcm.listener;
        if(listener != null){
            listener.start();
        }
    // 启动监听listener 实现了线程，所以在 run 方法中可以看到构建ServerSocket 的请求，这里专门用来接收其他zkServer
    // 的投票请求
    // 这块后续再分析
    @Override
    public void run() {
            int numRetries = 0;
            InetSocketAddress addr;
            while((!shutdown) && (numRetries < 3)){
                try {
                    ss = new ServerSocket();
                }
            }
        }
    }
```

FastLeaderElection.lookForLeader

这个方法在前面分析过，里面会调用 sendNotifications 来发送投票请求

```
public Vote lookForLeader() throws InterruptedException {
     //省略部分代码
     sendNotifications(); //这个方法，会把当前zk 服务器的信息添加到 sendqueue
     /*
     * Loop in which we exchange
    notifications until we find a leader
     */
     while ((self.getPeerState() == ServerState.LOOKING) &&
     //省略部分代码
}
```

FastLeaderElection.sendqueue

sendQueue 这个队列的数据，是通过 WorkerSender 来进行获取并发送的。而这个 WorkerSender 线程，在构建 fastLeaderElection 时，会启动

```
class WorkerSender extends ZooKeeperThread {
    public void run() {
        while (!stop) {
            try {//从队列中获取 ToSend 对象
                ToSend m = sendqueue.poll(3000, TimeUnit.MILLISECONDS);
                if(m == null) continue;
                process(m);
```

```
                //省略部分代码
                void process(ToSend m) {
                    ByteBuffer requestBuffer = buildMsg(m.state.ordinal(),
                    m.leader, m.zxid,
                    m.electionEpoch,
                    m.peerEpoch);
                    manager.toSend(m.sid, requestBuffer); // 这里就是调用 QuorumCnxManager
                                                          // 进行消息发送
                }
            }
        }
    }
}
```

QuorumCnxManager.toSend

```
public void toSend(Long sid, ByteBuffer b) {

        if (this.mySid == sid) { // 如果接受者是自己，直接放置到接收队列
            b.position(0);
            addToRecvQueue(new Message(b.duplicate(), sid));
        } else { // 否则发送到对应的发送队列上
            ArrayBlockingQueue<ByteBuffer> bq = new ArrayBlockingQueue<ByteBuffer>(SEND_CAPACITY);
            // 判断当前的 sid 是否已经存在于发送队列，如果是，则直接把已经存在的数据发送出去
            ArrayBlockingQueue<ByteBuffer> bqExisting = queueSendMap.putIfAbsent(sid, bq);
            if (bqExisting != null) {
                addToSendQueue(bqExisting, b);
            } else {
                addToSendQueue(bq, b);
            }
            connectOne(sid); // 连接申请调用链 connectOne-->initiateConnection-
                             // ->startConnection ， startConnection 就是发送方启动入口
        }
    }
```

startConnection

```
private boolean startConnection(Socket sock, Long sid) {
        // 省略部分代码
        if (sid > this.mySid) {
            // 为了防止重复建立连接，只允许 sid 大的主动连接 sid 小的
            closeSocket(sock);
        } else {
            // 构建一个发送线程和接收线程，负责针对当前连接的数据传递，后续的逻辑比较简单，就不做分析
            SendWorker sw = new SendWorker(sock, sid);
            RecvWorker rw = new RecvWorker(sock, din, sid, sw);
            sw.setRecv(rw);
        }
    }
```

SendWorker 会监听对应 sid 的阻塞队列，启动的时候回如果队列为空时会重新发送一次最前最后的消息，以防上一次处理是服务器异常退出，造成上一条消息未处理成功；然后就是不停监听队里，发现有消息时调用send 方法RecvWorker：RecvWorker 不停监听 socket 的 inputstream，读取消息放到消息接收队列中,消息放入队列中，qcm 的流程就完毕了。

QuorumCnxManager.Listener

listener 监听到客户端请求之后，开始处理消息

```
public void run() {
        // 省略部分代码
        while (!shutdown) {
            Socket client = ss.accept();
            setSockOpts(client);
            LOG.info("Received connection request" + client.getRemoteSocketAddress());
            if (quorumSaslAuthEnabled) {
                receiveConnectionAsync(client);
            } else {
                receiveConnection(client); // 接收客户端请求
            }
```

```
        }
    }
```

QuorumCnxManager.receiveConnection

```
public void receiveConnection(final Socket sock) {
        DataInputStream din = null;
        try {
            // 获取客户端的数据包
            din = new DataInputStream(new BufferedInputStream(sock.getInputStream()));
            handleConnection(sock, din);// 调用 handle 进行处理
        } catch (IOException e) {
            LOG.error("Exception handling connection, addr: {}, closing server connection",
                    sock.getRemoteSocketAddress());
            closeSocket(sock);
        }
    }
```

handleConnection

```
private void handleConnection(Socket sock, DataInputStream din)throws IOException {
             Long sid = null;
             try {
                 //获取客户端的 sid, 也就是 myid
                 sid = din.readLong();
                if (sid < 0) {
                    sid = din.readLong();
                    if (sid < this.mySid) {
                        //为了防止重复建立连接, 只允许 sid 大的主动连接 sid 小的
                        SendWorker sw = senderWorkerMap.get(sid);
                        if (sw != null) {
                            sw.finish();//关闭连接
                        }
                        LOG.debug("Create new connection to server: " + sid);
                        closeSocket(sock);//关闭连接
                        connectOne(sid);//向 sid 发起连接
                    } else {//同样, 构建一个 SendWorker 和RecvWorker 进行发送和接收数据
                        SendWorker sw = new
                        SendWorker(sock, sid);
                        RecvWorker rw = new
                        RecvWorker(sock, din, sid, sw);
                        sw.setRecv(rw);
                    }
                }
             }
        }
```

分类: zookeeper

47号Gamer丶
关注 - 3
粉丝 - 4
+加关注

« 上一篇： zookeeper原理之选举流程分析
» 下一篇： zookeeper原理之Leader选举完成之后的处理逻辑

posted @ 2020-08-21 23:03 47号Gamer丶 阅读(172) 评论(0) 编辑 收藏 举报

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

最新新闻：
· 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...