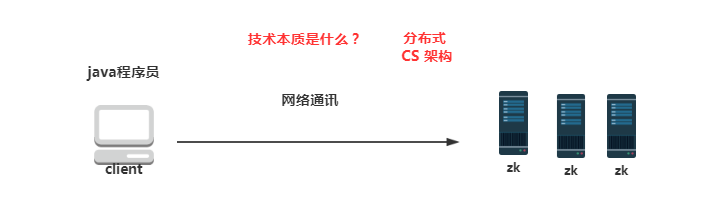
这节课我们是zk的最后一节课。之前的课我们了解了zk的特性，并且可以用它们能有些事情，那其根本，本质，底层是如何做到的，这节课我们将为大家来解密。

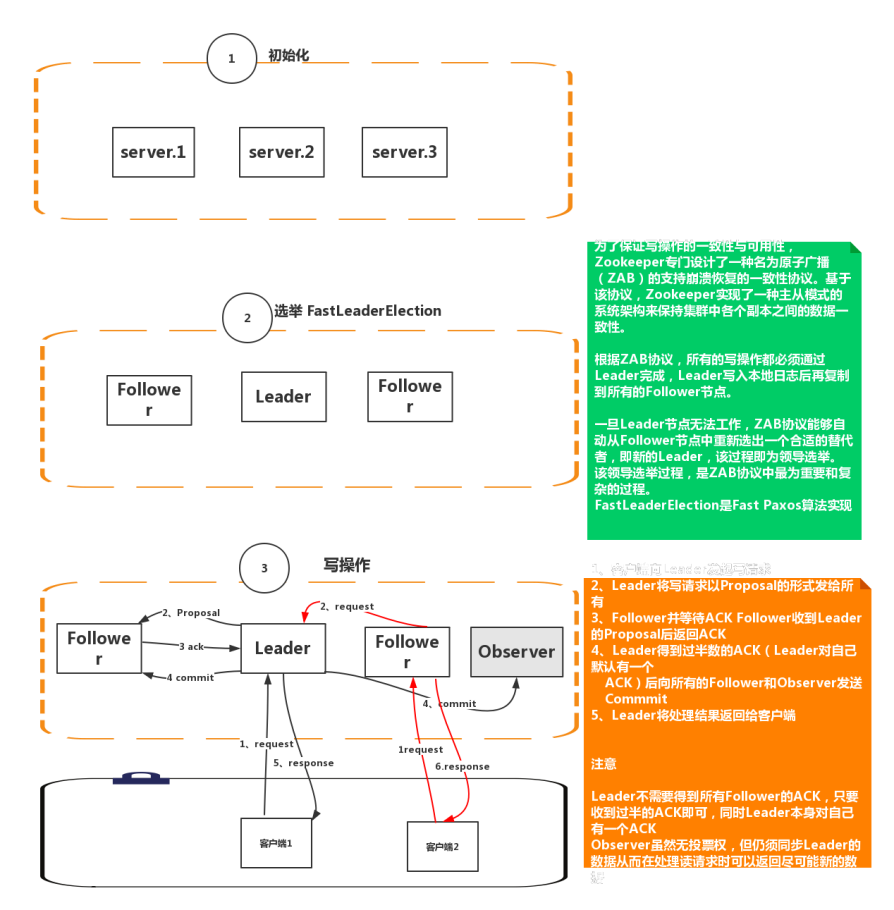
这节课会讲到：服务端源码原理、客户端源码原理、选举算法、序列化、运维、容灾等问题。

# ZK原理：



# 源码分析部分：

### 服务端：



服务启动：

|  |
| --- |
| **public static void** main(String[] args) {  QuorumPeerMain main = **new** QuorumPeerMain();  **try** {  main.initializeAndRun(args);//点这看  } **catch** (IllegalArgumentException e) {  ***LOG***.error(**"Invalid arguments, exiting abnormally"**, e);  ***LOG***.info(***USAGE***);  System.***err***.println(***USAGE***);  System.*exit*(2);  } **catch** (ConfigException e) {  ***LOG***.error(**"Invalid config, exiting abnormally"**, e);  System.***err***.println(**"Invalid config, exiting abnormally"**);  System.*exit*(2);  } **catch** (Exception e) {  ***LOG***.error(**"Unexpected exception, exiting abnormally"**, e);  System.*exit*(1);  }  ***LOG***.info(**"Exiting normally"**);  System.*exit*(0); } |

org.apache.zookeeper.server.quorum.QuorumPeerMain#initializeAndRun

|  |
| --- |
| **protected void** initializeAndRun(String[] args)  **throws** ConfigException, IOException { //读取zoo.cfg配置参数  QuorumPeerConfig config = **new** QuorumPeerConfig();  **if** (args.**length** == 1) {  config.parse(args[0]);  }   *// Start and schedule the the purge task*  *//启动日志清除任务* DatadirCleanupManager purgeMgr = **new** DatadirCleanupManager(config  .getDataDir(), config.getDataLogDir(), config  .getSnapRetainCount(), config.getPurgeInterval());  purgeMgr.start();   **if** (args.**length** == 1 && config.**servers**.size() > 0) {  runFromConfig(config); //读取到的配置进行搞事xxoo 哈哈   } **else** {  ***LOG***.warn(**"Either no config or no quorum defined in config, running "** + **" in standalone mode"**);  *// there is only server in the quorum -- run as standalone* ZooKeeperServerMain.*main*(args);  } } |

org.apache.zookeeper.server.quorum.QuorumPeerMain#runFromConfig

|  |
| --- |
| **public void** runFromConfig(QuorumPeerConfig config) **throws** IOException {  **try** {  ManagedUtil.*registerLog4jMBeans*();  } **catch** (JMException e) {  ***LOG***.warn(**"Unable to register log4j JMX control"**, e);  }   ***LOG***.info(**"Starting quorum peer"**);  **try** {  ServerCnxnFactory cnxnFactory = ServerCnxnFactory.*createFactory*();  cnxnFactory.configure(config.getClientPortAddress(),  config.getMaxClientCnxns());//创建服务端的Socket实列   **quorumPeer** = **new** QuorumPeer();//confg读取到的zoo.cfg赋值  **quorumPeer**.setClientPortAddress(config.getClientPortAddress());  **quorumPeer**.setTxnFactory(**new** FileTxnSnapLog(  **new** File(config.getDataLogDir()),  **new** File(config.getDataDir())));  **quorumPeer**.setQuorumPeers(config.getServers());  **quorumPeer**.setElectionType(config.getElectionAlg());  **quorumPeer**.setMyid(config.getServerId());  **quorumPeer**.setTickTime(config.getTickTime());  **quorumPeer**.setMinSessionTimeout(config.getMinSessionTimeout());  **quorumPeer**.setMaxSessionTimeout(config.getMaxSessionTimeout());  **quorumPeer**.setInitLimit(config.getInitLimit());  **quorumPeer**.setSyncLimit(config.getSyncLimit());  **quorumPeer**.setQuorumVerifier(config.getQuorumVerifier());  **quorumPeer**.setCnxnFactory(cnxnFactory);  **quorumPeer**.setZKDatabase(**new** ZKDatabase(**quorumPeer**.getTxnFactory()));  **quorumPeer**.setLearnerType(config.getPeerType());  **quorumPeer**.setSyncEnabled(config.getSyncEnabled());  **quorumPeer**.setQuorumListenOnAllIPs(config.getQuorumListenOnAllIPs());   **quorumPeer**.start(); //调用start方法 注意这不是调用线程的start方法  **quorumPeer**.join();  } **catch** (InterruptedException e) {  *// warn, but generally this is ok* ***LOG***.warn(**"Quorum Peer interrupted"**, e);  } } |

org.apache.zookeeper.server.quorum.QuorumPeer#start

|  |
| --- |
| @Override **public synchronized void** start() {  loadDataBase();//先从内存中恢复数据写到文件中  **cnxnFactory**.start(); //启动服务器端Socket实现   startLeaderElection();//开始选举  **super**.start();//这才真正调用线程的start方法也就会执行run方法 } |

org.apache.zookeeper.server.NIOServerCnxnFactory#run 服务端建立链接

|  |
| --- |
| **public void** run() {  **while** (!**ss**.socket().isClosed()) {  **try** {  **selector**.select(1000);  Set<SelectionKey> selected;  **synchronized** (**this**) {  selected = **selector**.selectedKeys();  }  ArrayList<SelectionKey> selectedList = **new** ArrayList<SelectionKey>(  selected);  Collections.*shuffle*(selectedList)；//乱序  **for** (SelectionKey k : selectedList) {  **if** ((k.readyOps() & SelectionKey.***OP\_ACCEPT***) != 0) {  SocketChannel sc = ((ServerSocketChannel) k  .channel()).accept();  InetAddress ia = sc.socket().getInetAddress();  **int** cnxncount = getClientCnxnCount(ia);  //调用zoo.cfg配置的客户端连接数是否超过了  **if** (**maxClientCnxns** > 0 && cnxncount >= **maxClientCnxns**){  ***LOG***.warn(**"Too many connections from "** + ia  + **" - max is "** + **maxClientCnxns** );  sc.close();  } **else** {  ***LOG***.info(**"Accepted socket connection from "** + sc.socket().getRemoteSocketAddress());  sc.configureBlocking(**false**);  //监听read事件  SelectionKey sk = sc.register(**selector**,  SelectionKey.***OP\_READ***);  //创建内部  NIOServerCnxn cnxn = createConnection(sc, sk);  sk.attach(cnxn);  addCnxn(cnxn);  }  } **else if** ((k.readyOps() & (SelectionKey.***OP\_READ*** | SelectionKey.***OP\_WRITE***)) != 0) {//处理读和写事件操作  NIOServerCnxn c = (NIOServerCnxn) k.attachment();  c.doIO(k);//不建议跟下去了  } **else** {  **if** (***LOG***.isDebugEnabled()) {  ***LOG***.debug(**"Unexpected ops in select "** + k.readyOps());  }  }  }  //清除 下次之需  selected.clear();  } **catch** (RuntimeException e) {  ***LOG***.warn(**"Ignoring unexpected runtime exception"**, e);  } **catch** (Exception e) {  ***LOG***.warn(**"Ignoring exception"**, e);  }  }  closeAll();  ***LOG***.info(**"NIOServerCnxn factory exited run method"**); } |

org.apache.zookeeper.server.quorum.QuorumPeer#startLeaderElection 选举开始

|  |
| --- |
| **synchronized public void** startLeaderElection() {  **try** {  **currentVote** = **new** Vote(**myid**, getLastLoggedZxid(), getCurrentEpoch());  //投票给自己  } **catch**(IOException e) {  RuntimeException re = **new** RuntimeException(e.getMessage());  re.setStackTrace(e.getStackTrace());  **throw** re;  }  //从配置中拿自己的选举地址  **for** (QuorumServer p : getView().values()) {  **if** (p.**id** == **myid**) {  **myQuorumAddr** = p.**addr**;  **break**;  }  }  **if** (**myQuorumAddr** == **null**) {  **throw new** RuntimeException(**"My id "** + **myid** + **" not in the peer list"**);  }  **if** (**electionType** == 0) {  **try** {  **udpSocket** = **new** DatagramSocket(**myQuorumAddr**.getPort());  **responder** = **new** ResponderThread();  **responder**.start();  } **catch** (SocketException e) {  **throw new** RuntimeException(e);  }  }  **this**.**electionAlg** = createElectionAlgorithm(**electionType**); //这是选举的开始 } |

org.apache.zookeeper.server.quorum.FastLeaderElection#starter 选举初始化

|  |
| --- |
| **private void** starter(QuorumPeer self, QuorumCnxManager manager) {  **this**.**self** = self;  **proposedLeader** = -1;  **proposedZxid** = -1;   **sendqueue** = **new** LinkedBlockingQueue<ToSend>();  **recvqueue** = **new** LinkedBlockingQueue<Notification>();  **this**.**messenger** = **new** Messenger(manager); } |

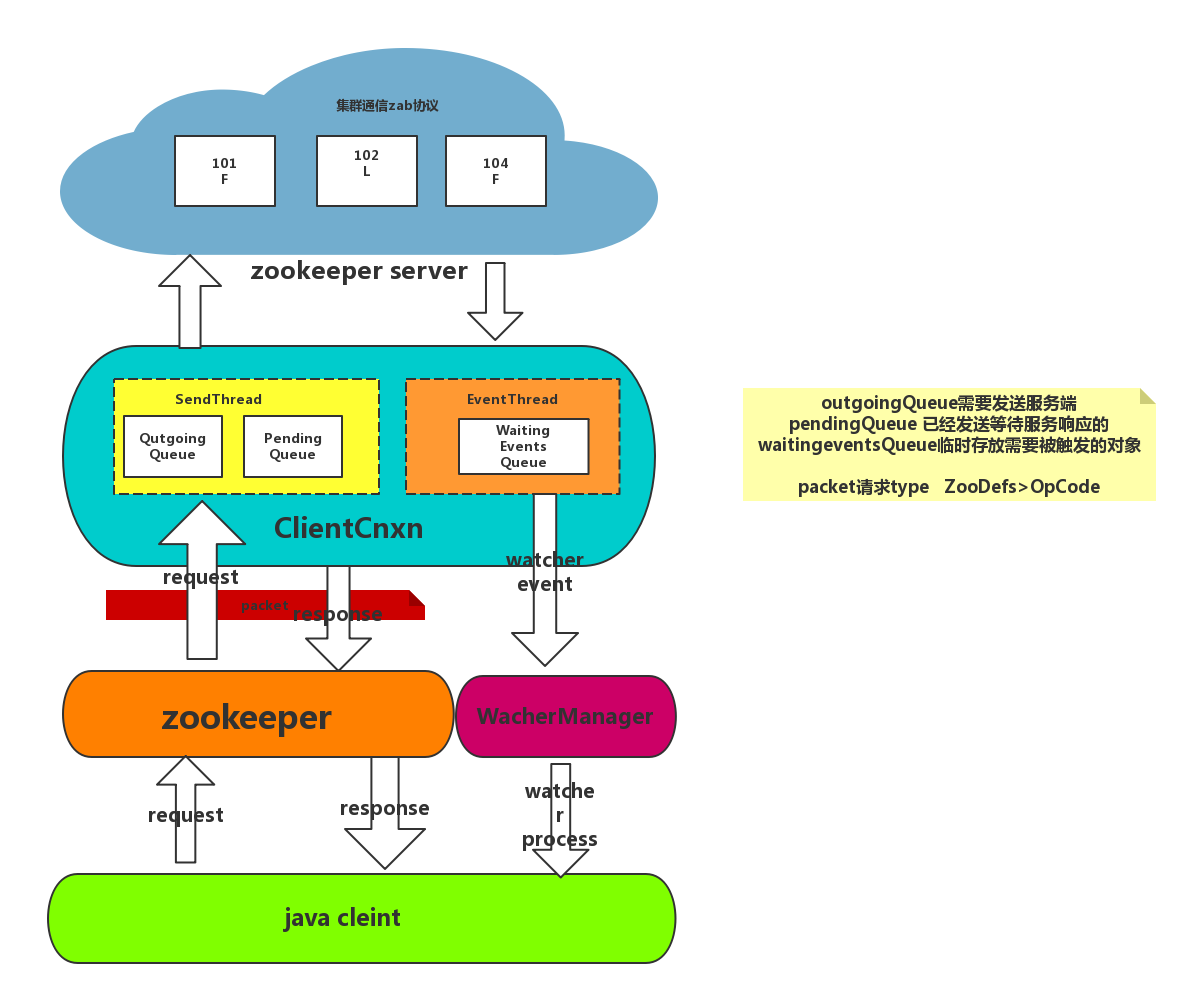
org.apache.zookeeper.server.quorum.QuorumPeer#run选举开始

这就不贴代码了

其次可以看看

FastLeaderElection中的lookForLeader方法 在这个run方法中会调用它 产生leader和follower

## 客户端：



# 客户端：

|  |
| --- |
| **public** ZooKeeper(String connectString, **int** sessionTimeout, Watcher watcher,  **boolean** canBeReadOnly)  **throws** IOException {  ***LOG***.info(**"Initiating client connection, connectString="** + connectString  + **" sessionTimeout="** + sessionTimeout + **" watcher="** + watcher);   **watchManager**.**defaultWatcher** = watcher;   ConnectStringParser connectStringParser = **new** ConnectStringParser(  connectString);  HostProvider hostProvider = **new** StaticHostProvider(  connectStringParser.getServerAddresses());//拿到ip端口号  **cnxn** = **new** ClientCnxn(connectStringParser.getChrootPath(),  hostProvider, sessionTimeout, **this**, **watchManager**,  *getClientCnxnSocket*(), canBeReadOnly);//创建ClientCnxn对象  **cnxn**.start();//非thread线程启动 } |

org.apache.zookeeper.ClientCnxn#ClientCnxn初始化 启动了两个线程 send和event

|  |
| --- |
| **public** ClientCnxn(String chrootPath, HostProvider hostProvider, **int** sessionTimeout, ZooKeeper zooKeeper,  ClientWatchManager watcher, ClientCnxnSocket clientCnxnSocket,  **long** sessionId, **byte**[] sessionPasswd, **boolean** canBeReadOnly) {  **this**.**zooKeeper** = zooKeeper;  **this**.**watcher** = watcher;  **this**.**sessionId** = sessionId;  **this**.**sessionPasswd** = sessionPasswd;  **this**.**sessionTimeout** = sessionTimeout;  **this**.**hostProvider** = hostProvider;  **this**.**chrootPath** = chrootPath;   **connectTimeout** = sessionTimeout / hostProvider.size();  **readTimeout** = sessionTimeout \* 2 / 3;  **readOnly** = canBeReadOnly;   **sendThread** = **new** SendThread(clientCnxnSocket);  **eventThread** = **new** EventThread();  } |

org.apache.zookeeper.ClientCnxn#start连个线程start>run方法

|  |
| --- |
| **public void** start() {  **sendThread**.start();  **eventThread**.start(); } |

org.apache.zookeeper.ClientCnxn.SendThread#run

|  |
| --- |
| @Override **public void** run() {  **clientCnxnSocket**.introduce(**this**,**sessionId**);  **clientCnxnSocket**.updateNow();  **clientCnxnSocket**.updateLastSendAndHeard(); //客户端和服务端链接的socket更新  **int** to;  **long** lastPingRwServer = System.*currentTimeMillis*();  **final int** MAX\_SEND\_PING\_INTERVAL = 10000; *//10 seconds* **while** (**state**.isAlive()) {  **try** {  **if** (!**clientCnxnSocket**.isConnected()) {  **if**(!**isFirstConnect**){  **try** {  Thread.*sleep*(**r**.nextInt(1000));  } **catch** (InterruptedException e) {  ***LOG***.warn(**"Unexpected exception"**, e);  }  }  *// don't re-establish connection if we are closing* **if** (**closing** || !**state**.isAlive()) {  **break**;  }  startConnect();  **clientCnxnSocket**.updateLastSendAndHeard();  }   **if** (**state**.isConnected()) {  *// determine whether we need to send an AuthFailed event.* **if** (**zooKeeperSaslClient** != **null**) {  **boolean** sendAuthEvent = **false**;  **if** (**zooKeeperSaslClient**.getSaslState() == ZooKeeperSaslClient.SaslState.***INITIAL***) {  **try** {  **zooKeeperSaslClient**.initialize(ClientCnxn.**this**);  } **catch** (SaslException e) {  ***LOG***.error(**"SASL authentication with Zookeeper Quorum member failed: "** + e);  **state** = States.***AUTH\_FAILED***;  sendAuthEvent = **true**;  }  }  KeeperState authState = **zooKeeperSaslClient**.getKeeperState();  **if** (authState != **null**) {  **if** (authState == KeeperState.***AuthFailed***) {  *// An authentication error occurred during authentication with the Zookeeper Server.* **state** = States.***AUTH\_FAILED***;  sendAuthEvent = **true**;  } **else** {  **if** (authState == KeeperState.***SaslAuthenticated***) {  sendAuthEvent = **true**;  }  }  }   **if** (sendAuthEvent == **true**) {  **eventThread**.queueEvent(**new** WatchedEvent(  Watcher.Event.EventType.***None***,  authState,**null**));  }  }  to = **readTimeout** - **clientCnxnSocket**.getIdleRecv();  } **else** {  to = **connectTimeout** - **clientCnxnSocket**.getIdleRecv();  }    **if** (to <= 0) {  String warnInfo;  warnInfo = **"Client session timed out, have not heard from server in "** + **clientCnxnSocket**.getIdleRecv()  + **"ms"** + **" for sessionid 0x"** + Long.*toHexString*(**sessionId**);  ***LOG***.warn(warnInfo);  **throw new** SessionTimeoutException(warnInfo);  }  **if** (**state**.isConnected()) {   *//1000(1 second) is to prevent race condition missing to send the second ping  //also make sure not to send too many pings when readTimeout is small* **int** timeToNextPing = **readTimeout** / 2 - **clientCnxnSocket**.getIdleSend() -   ((**clientCnxnSocket**.getIdleSend() > 1000) ? 1000 : 0);  *//send a ping request either time is due or no packet sent out within MAX\_SEND\_PING\_INTERVAL* **if** (timeToNextPing <= 0 || **clientCnxnSocket**.getIdleSend() > MAX\_SEND\_PING\_INTERVAL) {  sendPing();//发送心跳  **clientCnxnSocket**.updateLastSend();  } **else** {  **if** (timeToNextPing < to) {  to = timeToNextPing;  }  }  }   *// If we are in read-only mode, seek for read/write server* **if** (**state** == States.***CONNECTEDREADONLY***) {  **long** now = System.*currentTimeMillis*();  **int** idlePingRwServer = (**int**) (now - lastPingRwServer);  **if** (idlePingRwServer >= **pingRwTimeout**) {  lastPingRwServer = now;  idlePingRwServer = 0;  **pingRwTimeout** =  Math.*min*(2\***pingRwTimeout**, ***maxPingRwTimeout***);  pingRwServer();  }  to = Math.*min*(to, **pingRwTimeout** - idlePingRwServer);  }   **clientCnxnSocket**.doTransport(to, **pendingQueue**, **outgoingQueue**, ClientCnxn.**this**);  //这个方法比较长 重点看这  } **catch** (Throwable e) {  **if** (**closing**) {  **if** (***LOG***.isDebugEnabled()) {  *// closing so this is expected* ***LOG***.debug(**"An exception was thrown while closing send thread for session 0x"** + Long.*toHexString*(getSessionId())  + **" : "** + e.getMessage());  }  **break**;  } **else** {  *// this is ugly, you have a better way speak up* **if** (e **instanceof** SessionExpiredException) {  ***LOG***.info(e.getMessage() + **", closing socket connection"**);  } **else if** (e **instanceof** SessionTimeoutException) {  ***LOG***.info(e.getMessage() + ***RETRY\_CONN\_MSG***);  } **else if** (e **instanceof** EndOfStreamException) {  ***LOG***.info(e.getMessage() + ***RETRY\_CONN\_MSG***);  } **else if** (e **instanceof** RWServerFoundException) {  ***LOG***.info(e.getMessage());  } **else** {  ***LOG***.warn(  **"Session 0x"** + Long.*toHexString*(getSessionId())  + **" for server "** + **clientCnxnSocket**.getRemoteSocketAddress()  + **", unexpected error"** + ***RETRY\_CONN\_MSG***, e);  }  cleanup();  **if** (**state**.isAlive()) {  **eventThread**.queueEvent(**new** WatchedEvent(  Event.EventType.***None***,  Event.KeeperState.***Disconnected***,  **null**));  }  **clientCnxnSocket**.updateNow();  **clientCnxnSocket**.updateLastSendAndHeard();  }  }  }  cleanup();  **clientCnxnSocket**.close();  **if** (**state**.isAlive()) {  **eventThread**.queueEvent(**new** WatchedEvent(Event.EventType.***None***,  Event.KeeperState.***Disconnected***, **null**));  }  ZooTrace.*logTraceMessage*(***LOG***, ZooTrace.*getTextTraceLevel*(),  **"SendThread exited loop for session: 0x"** + Long.*toHexString*(getSessionId())); } |

org.apache.zookeeper.ClientCnxnSocketNIO#doTransport 真正干事的

|  |
| --- |
| @Override **void** doTransport(**int** waitTimeOut, List<Packet> pendingQueue, LinkedList<Packet> outgoingQueue,  ClientCnxn cnxn)  **throws** IOException, InterruptedException {  **selector**.select(waitTimeOut);  Set<SelectionKey> selected;  **synchronized** (**this**) {  selected = **selector**.selectedKeys();  }  *// Everything below and until we get back to the select is  // non blocking, so time is effectively a constant. That is  // Why we just have to do this once, here* updateNow();  **for** (SelectionKey k : selected) {  SocketChannel sc = ((SocketChannel) k.channel());  **if** ((k.readyOps() & SelectionKey.***OP\_CONNECT***) != 0) {  **if** (sc.finishConnect()) {  updateLastSendAndHeard();  **sendThread**.primeConnection();  }  } **else if** ((k.readyOps() & (SelectionKey.***OP\_READ*** | SelectionKey.***OP\_WRITE***)) != 0) {  doIO(pendingQueue, outgoingQueue, cnxn);//这是处理客户端往服务端发送的数据 链接之后会处理读和写操作 这不往下跟代码了 }  }  **if** (**sendThread**.getZkState().isConnected()) {  **synchronized**(outgoingQueue) {  **if** (findSendablePacket(outgoingQueue,  cnxn.**sendThread**.clientTunneledAuthenticationInProgress()) != **null**) {  enableWrite();  }  }  }  selected.clear(); } |

如果是回调函数怎么处理了

org.apache.zookeeper.ClientCnxn.SendThread#run

里面调用了org.apache.zookeeper.ClientCnxn.EventThread#queueEvent 这个是往event队列放数据的。

org.apache.zookeeper.ClientCnxn.EventThread#run 这个就是从队列里面取数据了

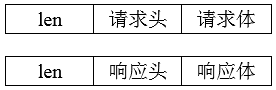
|  |
| --- |
| **public void** run() {  **try** {  **isRunning** = **true**;  **while** (**true**) {  Object event = **waitingEvents**.take();  **if** (event == **eventOfDeath**) {  **wasKilled** = **true**;  } **else** {  processEvent(event);  }  **if** (**wasKilled**)  **synchronized** (**waitingEvents**) {  **if** (**waitingEvents**.isEmpty()) {  **isRunning** = **false**;  **break**;  }  }  }  } **catch** (InterruptedException e) {  ***LOG***.error(**"Event thread exiting due to interruption"**, e);  }   ***LOG***.info(**"EventThread shut down for session: 0x{}"**,  Long.*toHexString*(getSessionId())); } |

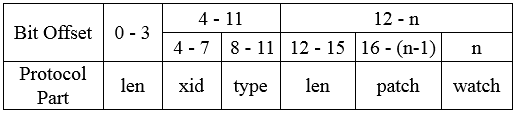
# ZK序列化Jute详解：

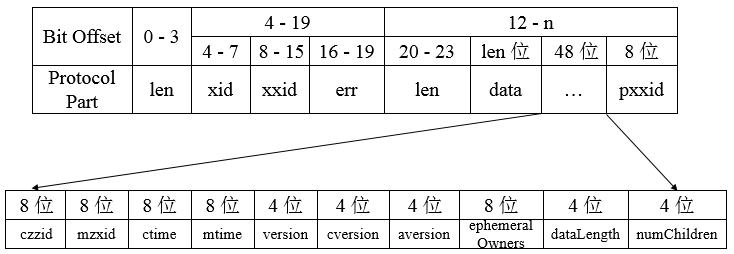
一般序列化框架：Apache Avro，Thrift，Protobuf等序列化组件，

但是Zk并没有用它们，而是用的Jute，主要还是由于考虑到新老版本序列化组件的兼容性，另一方面Jute并没有成为Zookeeper的瓶颈所在；

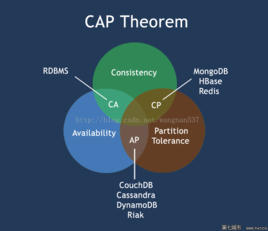
OutputArchive和InputArchive分别是Jute底层的序列化器和反序列化器。







# 分布式CAP理论：



CAP原则又称CAP定理，指的是在一个分布式系统中，Consistency（一致性）、 Availability（可用性）、Partition tolerance（分区容错性），三者不可兼得。

在课上把这块和选举结合起来讲。

## 运维：

四字命令：

主要如下：

| **ZooKeeper四字命令** | **功能描述** |
| --- | --- |
| conf | 3.3.0版本引入的。打印出服务相关配置的详细信息。 |
| cons | 3.3.0版本引入的。列出所有连接到这台服务器的客户端全部连接/会话详细信息。包括"接受/发送"的包数量、会话id、操作延迟、最后的操作执行等等信息。 |
| crst | 3.3.0版本引入的。重置所有连接的连接和会话统计信息。 |
| dump | 列出那些比较重要的会话和临时节点。这个命令只能在leader节点上有用。 |
| envi | 打印出服务环境的详细信息。 |
| reqs | 列出未经处理的请求 |
| ruok | 测试服务是否处于正确状态。如果确实如此，那么服务返回"imok"，否则不做任何相应。 |
| stat | 输出关于性能和连接的客户端的列表。 |
| srst | 重置服务器的统计。 |
| srvr | 3.3.0版本引入的。列出连接服务器的详细信息 |
| wchs | 3.3.0版本引入的。列出服务器watch的详细信息。 |
| wchc | 3.3.0版本引入的。通过session列出服务器watch的详细信息，它的输出是一个与watch相关的会话的列表。 |
| wchp | 3.3.0版本引入的。通过路径列出服务器watch的详细信息。它输出一个与session相关的路径。 |
| mntr | 3.4.0版本引入的。输出可用于检测集群健康状态的变量列表 |

# 容灾

如何保证高可用，跨机房如何部署，部署策略是什么？

课上说