1. 上课须知

每次课前的约定：

20:00 准时开始上课！ 20:00 准时开始上课！ 20:00 准时开始上课！

能听到音乐，能看到画面的小伙伴，请在直播间评论栏扣 666， 如果有其他小伙伴扣 666 证明我的直播环境是OK 的，然后听不见，或者看不到的小伙伴赶紧调整自己的上课环境。

一个晚上的上课时间是：20:00 - 23:00，中间会找机会休息一次。10分钟左右。

2. 环境准备

不需要过多的准备，准备一个 IDE，从官网下载源码包，然后直接用 IDE 打开即可！

```
1、准备一个IDE：IDEA
2、从官网下载源码包，IDEA去导入这个源码项目即可
3、稍微等待一下，maven去下载一些依赖jar
```

```
zookeeper-3.4.14.tar.gz    安装包 源码包
```

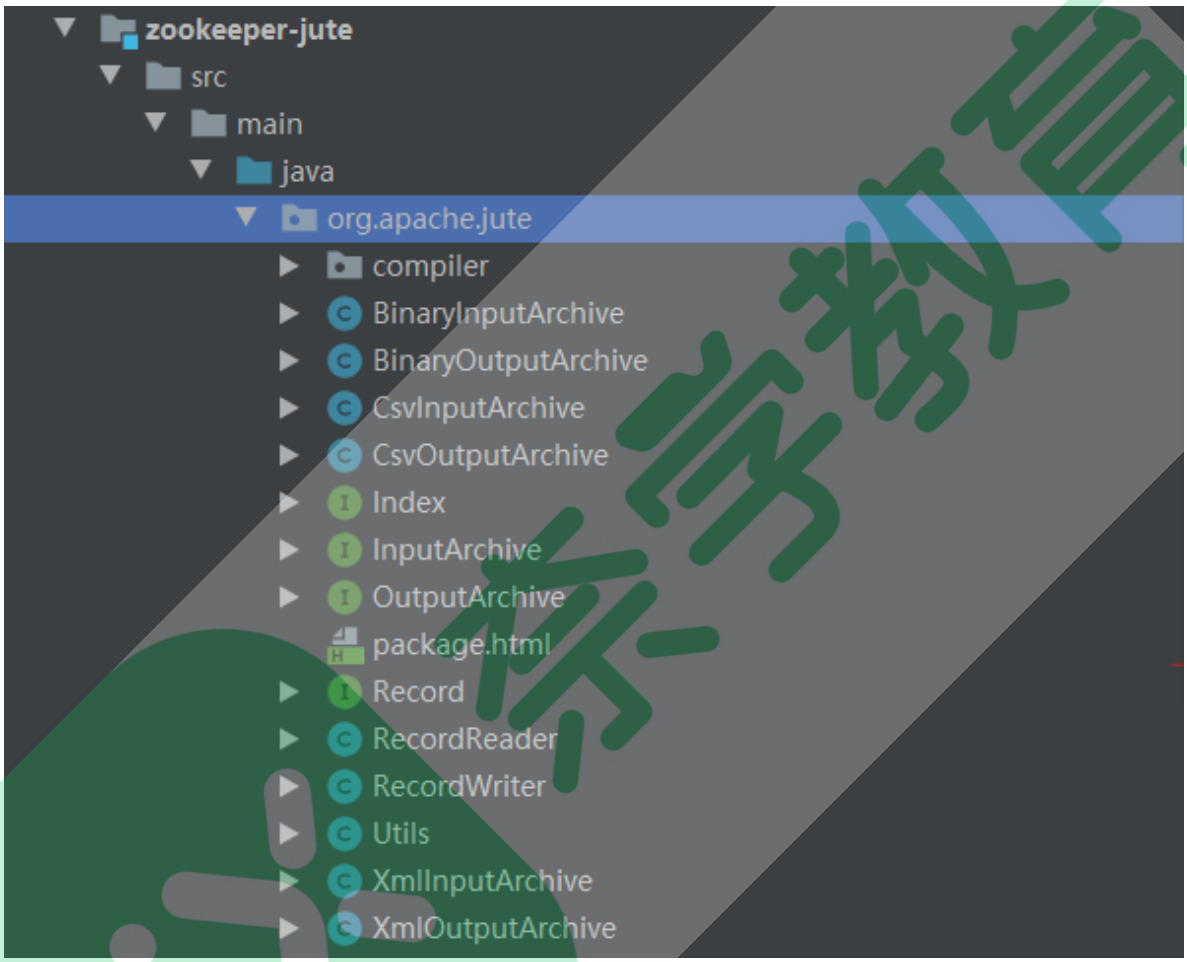
```
从 github 去拉取！
```

3. ZooKeeper序列化和网络通信协议详解

任何一个分布式系统的底层，都必然会有网络通信，这就必然要提供一个分布式通信框架和序列化机制。所以我们在看ZooKeeper源码之前，先搞定 ZooKeeper 网络通信和序列化

3.1. 序列化

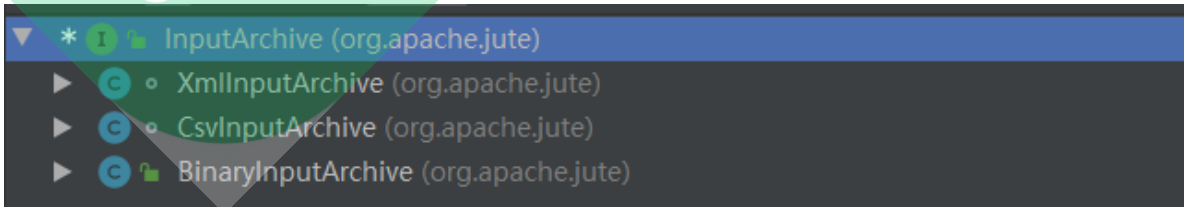
序列化的 API 主要在 zookeeper-jute 模块中。



重点API：

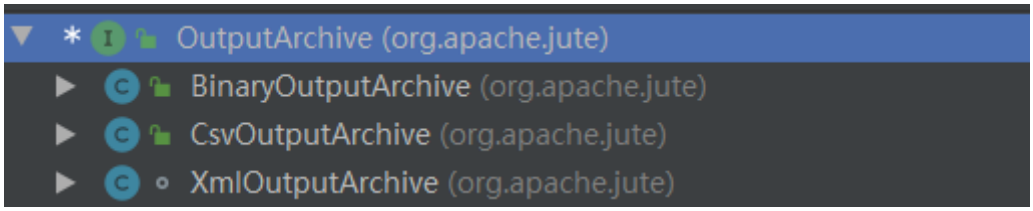
org.apache.jute.InputArchive：反序列化需要实现的接口，其中各种read开头的方法，都是反序列化方法

实现类有:



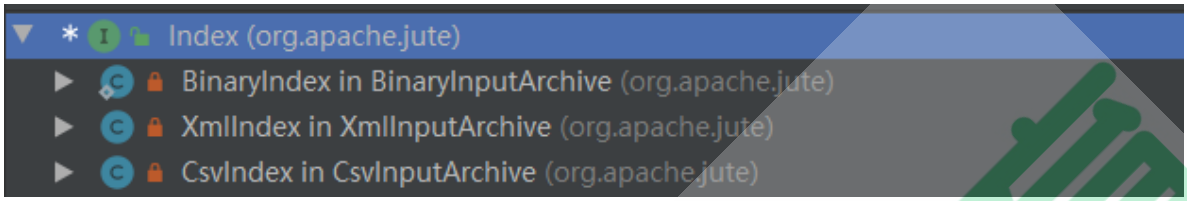
org.apache.jute.OutputArchive：所有进行序列化操作的都是实现这个接口，其中各种write开头的方法都是序列化方法。

实现类有：



org.apache.jute.Index：用于迭代数据进行反序列化的迭代器

实现类有：

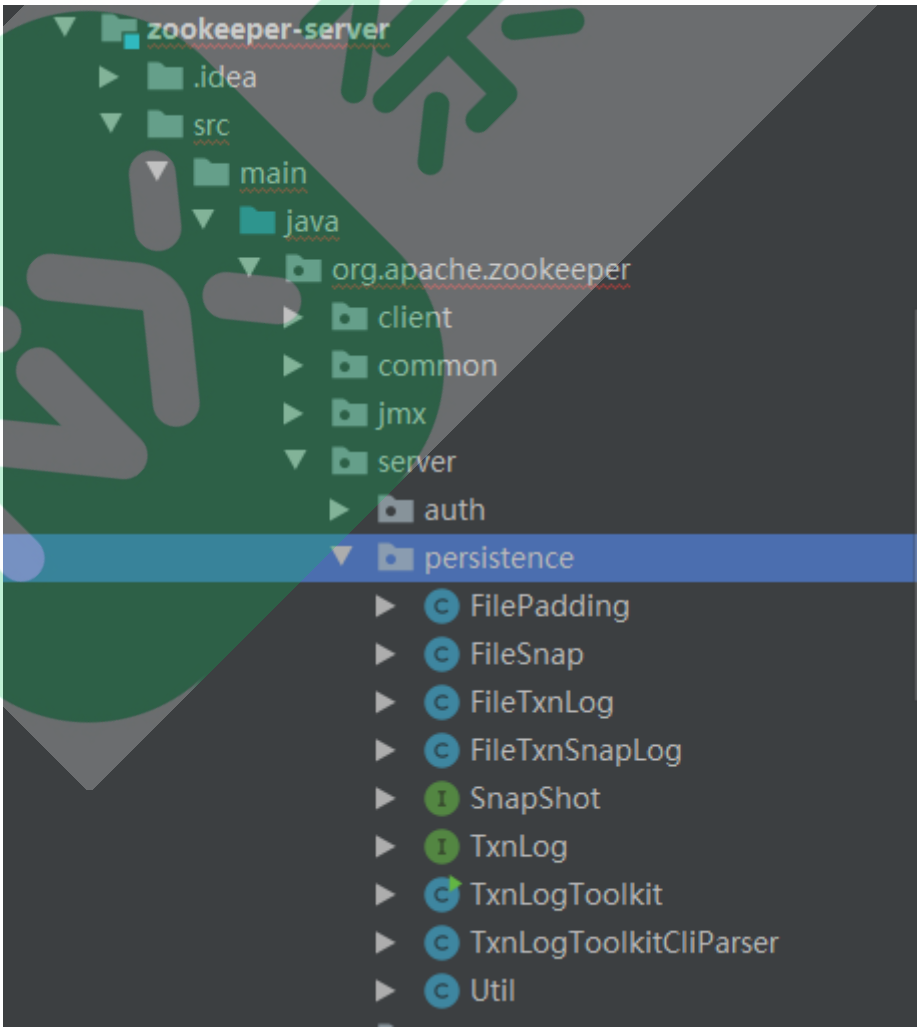


org.apache.jute.Record：在 ZooKeeper 要进行网络通信的对象，都需要实现这个接口。里面有序列化和反序列化两个重要的方法

3.2. ZooKeeper的持久化机制

```
zookeeper本身是一个 leader, follower 对等架构（内部选举 leader）
每个节点上都保存了整个系统的所有数据
每个节点上的都把数据放在磁盘一份，放在内存一份
```

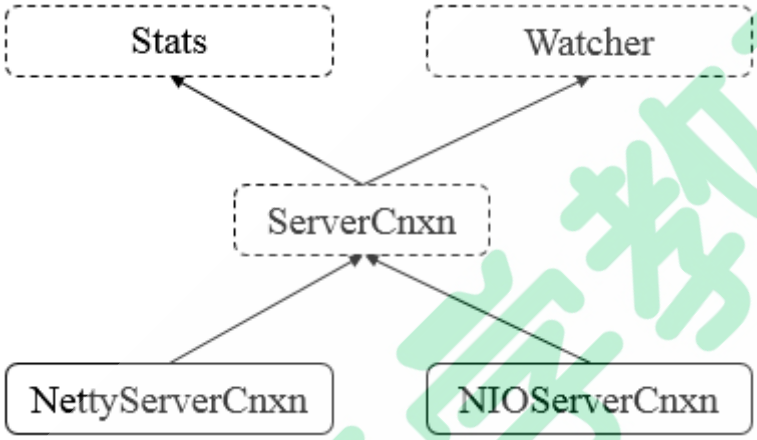
ZooKeeper 的持久化的一些操作接口，都在：org.apache.zookeeper.server.persistence 包中。



主要的类的介绍：

```
TxnLog，接口，读取事务性日志的接口。
FileTxnLog，实现TxnLog接口，添加了访问该事务性日志的API。
Snapshot，接口类型，持久层快照接口。
FileSnap，实现Snapshot接口，负责存储、序列化、反序列化、访问快照。
FileTxnSnapLog，封装了TxnLog和SnapShot。
Util，工具类，提供持久化所需的API。
```

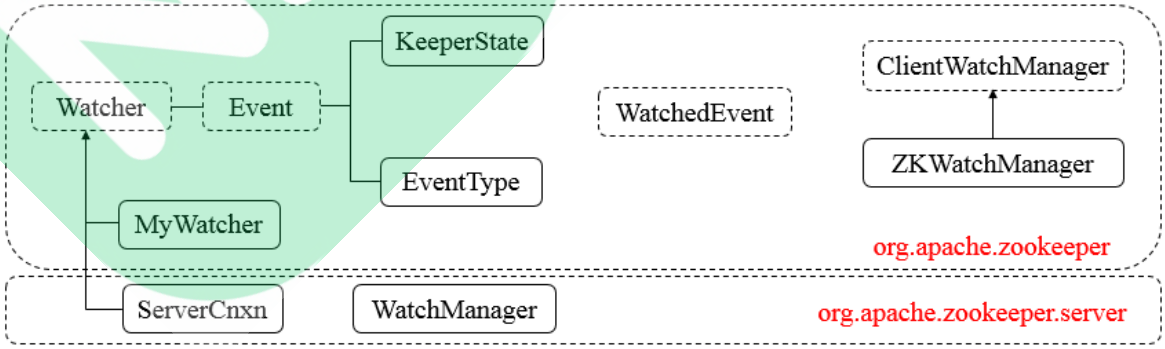
3.3. 网络通信框架



详细说明：

```
Stats，表示ServerCnxn上的统计数据。
Watcher，表示时间处理器。
ServerCnxn，表示服务器连接，表示一个从客户端到服务器的连接。
NettyServerCnxn，基于Netty的连接的具体实现。
NIOServerCnxn，基于NIO的连接的具体实现。
```

4. Watcher监听机制



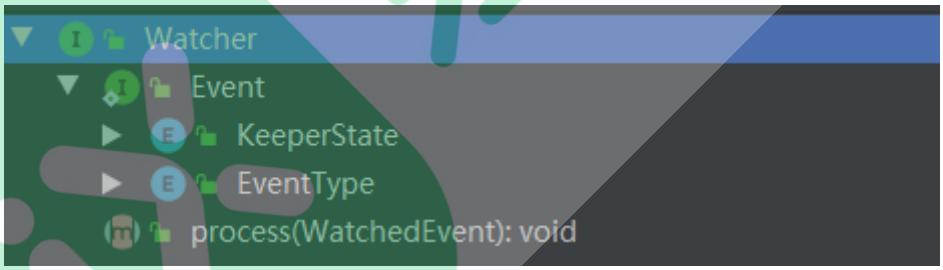

```
interface Watcher{
    class Event{

        class KeeperState
```

```
        class EventType
    }

    class WatchedEvent{

        KeeperState state     会话连接的状态信息
        String path           znode节点的绝对路劲
        EventType type        事件的类型
    }

    void process(WatchedEvent event)
}
```

组件说明：

```
Watcher，接口类型，其定义了process方法，需子类实现。
Event，接口类型，Watcher的内部类，无任何方法。
KeeperState，枚举类型，Event的内部类，表示Zookeeper所处的状态。
EventType，枚举类型，Event的内部类，表示Zookeeper中发生的事件类型。
WatchedEvent，表示对ZooKeeper上发生变化后的反馈，包含了KeeperState和EventType。
ClientWatchManager，接口类型，表示客户端的Watcher管理者，其定义了materialized方法，需子类实现。
ZKWatchManager，Zookeeper的内部类，继承ClientWatchManager。
MyWatcher，ZooKeeperMain的内部类，继承Watcher。
ServerCnxn，接口类型，继承Watcher，表示客户端与服务端的一个连接。
WatchManager，管理Watcher。
```

Watcher类组成：



WatchedEvent构成：

```java
/**
 *  A WatchedEvent represents a change on the ZooKeeper that a Watcher
 *  is able to respond to.  The WatchedEvent includes exactly what happened,
 *  the current state of the ZooKeeper, and the path of the znode that
 *  was involved in the event.
 */
@InterfaceAudience.Public
public class WatchedEvent {

    // 链接信息
    final private KeeperState keeperState;
    // 事件类型
    final private EventType eventType;
    // 事件发生的znode节点
    private String path;
```

5. ZooKeeper的选举机制

新启动节点的状态为LOOKING，在节点的主线程启动后（QuorumPeer.run调用后），会调用Election的lookForLeader获取Leader信息。

当FastLeaderElection要发送数据时，会通过向sendqueue发送数据来异步调用WorkerSender.process。

在QuorumCnxManager.toSend的实现中，若到发送目标节点的连接不存在，则会主动建立连接。

QuorumCnxManager通过加数据添加到发送列表来异步调用SendWorker.send。

在发送端建立连接时，首先会发送一个头信息，包括版本信息，当前节点的id，以及ip地址信息，详细信息详见QuorumCnxManager.initiateConnection。

在接收端收到新连接时，首先会解析发送方发送的头信息。为了保证两个节点之间只有一个连接（即不会出现A建立连接到B，同时B也建立连接到A），zk会检查建立连接的节点的id。若发起连接的节点id小于当前节点，zk会断开这连接，并且主动建立一个到对方节点的连接。

RecvWorker在收到数据后，调用addToRecvQueue将数据添加到队列，WorkerReceiver从队列取数据处理。WorkerReceiver处理逻辑如下:

- 检查对方节点是否参与选举，不参与选举的话直接返回当前的节点的结果。
- 若当前节点也在LOOKING状态，则将对方节点选举的结果添加到接收队列。若对方节点处于LOOKING状态，切状态落后于当前节点，则发送当前节点的选举信息到对方节点。
- 若当前节点处于选举成功状态(不是LOOKING状态)，则发送当前节点的选举结果到对方节点。

6. Leader 选举算法源码解读

Election为选举基类，lookForLeader会阻塞一直到成功获取Leader节点。

在整个选举过程中，最终的两个类是： FastLeaderElection 和 QuorumCnxManager

QuorumCnxManager负责选举中的连接管理，该类包含三个内部类:

- Listener 负责监接收新连接，并且管理连接，在单独的线程运行。
- SendWorker 负责发送数据到其他节点，所发送的数据由FastLeaderElection提供，在单独的线程运行。
- RecvWorker 负责从其他节点接收数据，并且将数据发送给FastLeaderElection，在单独的线程运行。

FastLeaderElection 负责选举的算法处理，该类包含如下主要inner class。

- Messenger 发送和接收数据的管理类。注意，此处的发送和接收数据更关注的是打包和解包相关的逻辑，而不和网络相关，网络相关的部分由QuorumCnxManager负责。
- WorkerSender 负责打包选举数据，发送到其他节点，在单独的线程运行。
- WorkerReceiver 负责解析接收到数据，并根据结果反馈对应的ACK，在单独的线程运行。

FastLeaderElection 启动后：

- 创建QuorumCnxManager实例，并且开始监听其他节点新建的连接。
- 创建FastLeaderElection实例，并且开启发送和接收数据的线程。（Messenger类会启动WorkerSender和WorkReceiver的线程）

7. 数据同步源码解读

在节点选举完成后，就会进入FOLLOWING或LEADING状态。在其能对外提供服务之前，需要对各个服务器之间状态进行同步，使服务器的数据保持一致。

7.1. 同步数据

在创建Leader对象时，会创建Leader监听的socket，在LeanerCnxAcceptor中会accept该连接，并且创建LearnerHandler的实例，该实例会新建线程，并处理和该新建的连接通信。

7.2. 同步数据详细流程

以 Follower 节点为例介绍 Learner 节点和 Leader 同步。同步数据的相关类以及接口为上两节的时序图所示，具体的流程如下。

- Follower节点在建立到Leader的连接后，立即发送FollowerInfo, 数据中包括自身节点类型，zxid，协议号(0x10000)，以及节点ID(sid)。
- Leader 回复 LeaderInfo 信息，信息包括 zxid, 以及版本号
- Follower 回复 ACKEPOCH, 信息包括 lastLoggedZxid，epoch 信息。
- Leader 等待超过半数节点回复的 ACKEPOCH
- Leader 会根据 Follower 发送的 zxid 信息，判断采用何种方式和 Follower 同步数据，并将同步数据的 Request 添加到队列。注意，此时消息并未发送，只是添加到队列。而具体的同步细节会在后续讲解操作日志以及 Snapshot 文件时讨论。
- 将 NEWLEADER Request 添加到队列。
- 发送队列的消息，因为同步数据的队列在前，NEWLEADER 在后，所以会先和 Follower 同步数据，再发送 NEWLEADER。
- Follower 收到 NEWLEADER 后回复 ACK。
- Leader 等待大部分节点回复 ACK
- Leader 发送 UPTODATE

在上述步骤完成后，集群状态已经稳定，可以对外提供服务。

8. 源码阅读万变不离其宗大法

8.1. 源码阅读

```
1、了解大概原理
```

大致的启动过程
大致的心跳机制流程
大致的任务提交过程

2、场景驱动
hdfs dfs -put /a /b
启动流程？
数据读写流程？
任务执行机制？

3、找入口
启动集群的命令 start-dfs.sh
hadoop-daemon.sh start namenode
NameNode.main()
hadoop-daemon.sh start datanode
提交任务的命令

4、理主线
namenode启动httpserver
namenode加载元数据
namenode启动RPCserver

5、看源码注释
类注释
成员变量注释
成员方法注释

6、代码结构：
1、参数解析 和 权限控制， 总之为核心业务做准备
2、try catch， 一般来说，核心方法，都藏于try中，异常以及容错处理，都藏于 catch 中
3、状态处理， 一般用来处理核心业务的结果数据
4、收尾，回收资源相关

7、作图
啥都没有图好使！自从画了图，一辈子忘不了。