Zookeeper

- 节点类型
  - 临时节点
  - 临时顺序节点
  - 持久节点
  - 持久顺序节点
- 定义
  - 一个分布式的，开放源码的分布式应用程序协调服务。为分布式应用提供一致性服务
  - 发布订阅（配置中心）
  - 负载均衡
  - 命名服务
  - 分布式协调/通知
  - 集群管理
  - Master选举
  - 分布式锁
  - 分布式队列
- 应用
  - Canal
  - Hadoop
  - HBase
  - Kafka
  - Dubbo
- ZAB协议
  - 数据同步
  - ZXID
  - 两种模式
    - 消息广播
      - 客户端将写入数据申请发送到任意Follower
      - Follower将写入请求转发给Leader
      - Leader采用两阶段提交方式，先发送Prapose（带上最新的ZXID）广播给各Follower
      - Follower接收到Prapose后将该请求加入到消息历史
        队列(history queue)，并返回ACK消息给Leader
      - 当leader收到大多数follower（超过法定数量）的ack消息，leader会发送commit请求
      - 当follower收到commit请求时，会判断该事务的ZXID是不是比历史队列中的任何事务的ZXID都小，如果是则提交，如果不是则等待比它更小的事务的commit
    - 崩溃恢复
      - 基本特性
        - 确保那些已经在Leader服务器上提交的事务最终被所有服务器都提交
        - 确保丢弃那些只在Leader服务器上被提交的议案
      - 四个阶段
        - 选主
        - 发现
        - 同步
          - 1. 重新加载本地磁盘上的数据快照至内存，并从日志文件中取出快照之后的
            所有事务操作，逐条应用至内存，并添加到已提交事务缓存commitedProposals。
            这样能保证日志文件中的事务操作，必定会应用到leader的内存数据库中。
          - 2. 获取learner发送的FOLLOWERINFO/OBSERVERINFO信息，并与自身commitedProposals比对，确定采用哪种同步方式，不同的learner可能采用
            不同同步方式（DIFF同步、TRUNC+DIFF同步、SNAP同步）
            - DIFF同步
            - TRUNC+DIFF同步
            - SNAP同步
          - leader主动向所有learner发送同步数据消息，每个learner有自己的发送队列，互不干扰。
            同步结束时，leader会向learner发送NEWLEADER指令，同时learner会反馈一个ACK。当leader接收到来自learner的ACK消息后，就认为当前learner已经完成了数据同步，同时进入"过半策略"等待阶段。当leader统计到收到了一半已上的ACK时，会向所有已经完成数据同步的learner发送一个UPTODATE指令，用来通知learner集群已经完成了数据同步，可以对外服务了。
        - 广播
- 崩溃恢复过程
  - 1. 各个节点向其他节点发起投票，投票当中包含自己的服务器ID及最新事务ID（ZXID），此时处于Looking状态
  - 2. 节点会用自身的ZXID 与其他节点的ZXID做比较，如果发现其他节点的ZXID比较大，则重新发起投票（ZXID相同则取ID最大的），投票给目前已知最大的ZXID所属节点
  - 3. 每次投票后，服务器都会统计投票数量，判断是否有某个节点得到半数以上的投票，该节点将会成为准leader, 状态变为 Leading, 其他节点为Following
  - 4. 发现阶段，检查是否有多个leading状态的节点，即leading接收Followler发来的各自的最新的epoch值，leading从中选出最大的epoch值，基于此值加一，生成新的epoch分发给各个Follower,各个Follower 收到最新epoch后返回ACK给leading 节点，带上各自最大的ZXID和历史事务日志
  - 5. 同步阶段，把leader 刚才收集到的最新历史事务日志，同步给集群中所有的Follower, 只有当过半数的Follower同步成功，这个leading状态的节点才会成为正式的Leader
- 数据顺序一致性
- 集群角色
  - Leader
  - Follower
  - Observer
- 设计目标
  - 1. 简单的数据模型
  - 2. 可以构建集群
  - 3. 顺序访问
  - 4. 高性能