ZooKeeper

特性

- 最终一致性 —— 客户端连接任何节点，都会收到同一份状态
- 原子性 —— 更新只能成功或者失败，没有中间状态
- 等待无关（wait-free） —— 慢的或者失效的 client 不得干预快速的 client 的请求，使得每个 client 都能有效的等待
- 顺序性
  - 全局有序 —— 在一台 server 上消息 a 在消息 b 前发布，则在所有 Server 上消息 a 都将在消息 b 前被发布
  - 偏序 —— 消息 b 在消息 a 后被同一个发送者发布，a 必将排在 b 前面
- 可靠性 —— 如果消息被到一台 server 接受，那么它将被所有的 server 接受
- 实时性 —— 保证 client 将在一定时间间隔内获得 server 的更新信息，或者 server 失效的信息

数据模型

- 层次模型
  - 树形结构便于表达数据间的层次关系
  - 树形结构便于为不同的应用分配独立的命名空间
- 节点类型
  - 持久性 znode —— 在创建之后即使发生 ZooKeeper 集群宕机或者 client 宕机也不会丢失
  - 临时性 znode —— client 宕机或在指定的 timeout 时间内没有给 ZooKeeper 集群发消息，节点就会消失
  - 持久顺序性 znode
  - 临时顺序性 znode
  - container —— 目的在于下挂子节点（当其所有子节点被删除之后，ZooKeeper 会删除掉该节点）
- Data tree API
  - 使用 UNIX 风格路径名来定位 znode，如 /A/X
  - znode 的数据只支持全量写入和读取
  - 所有 API 都是 wait-free 的，执行中的 API 调用不会影响其他 API 的完成
  - 不直接提供锁这样的分布式协同机制，但是可以用来实现多种分布式协同机制

集群架构

- 角色
  - 领导者（Leader）
    - 不直接接受 Client 请求，接受由 Follower 和 Observer 转发过来的 Client 请求
    - 负责进行投票的发起和决议，更新系统状态
  - 学习者（Learner）
    - 跟随者（Follower）
      - 接受客户端请求并返回结果
      - 参与 Leader 发起的投票和选举
      - 不具有写操作的权限
    - 观察者（Observer）
      - 接受客户端连接，将写操作转给 Leader
      - 不参与投票，只同步 Leader 节点的状态
- 节点数 —— 大于 1，且为单数，目的在于容错、防脑裂

应用场景

- 命名服务
- 数据发布/订阅
  - 分类
    - 配置中心
    - 服务注册与发现
  - 思路
    - server 给注册了 watch 的 client 推送通知
    - client 获得通知后，主动拉取 server 最新的数据
- 分布式锁
  - 思路 —— 使用临时顺序 znode 来表示获取锁的请求，创建最小后缀数字 znode 的用户成功拿到锁
  - 避免羊群效应 —— 每个锁请求者 watch 它前面的锁请求者。每次锁被释放，只会有一个锁请求者会被通知到
- Master 选举
  - 思路 1
    - 谁先创建 master 临时节点，谁就是 master
    - 当 master 挂掉（master 节点消失）后，其它节点会监听到，并继续创建新的 master 节点
  - 思路 2 —— 在 master 下面创建临时有序节点，那个节点最小，那个就是 master
- 分布式协调/通知 —— 思路
  - 发送者创建一个节点，并将数据写入该节点
  - 接受者对该节点注册 watch，等待通知的到来
- 集群管理 —— 思路
  - client 对数据节点注册 Watcher 监听，当该节点的内容或子节点列表发生变更时，就会收到通知
  - 对于临时节点，一旦 client 与 server 间的会话失效，那么该节点就会被自动清除
- 分布式队列 —— 不建议使用