Zookeeper

> zookeeper是一个高性能、开源的分布式应用协调服务。Zookeeper 致力于为那些高吞吐的大型分布式系统提供一个高性能、高可用、且具有严格顺序访问控制能力的分布式协调服务。具有以下特性：
>
> **顺序一致性**：从一个客户端发起的事务请求，最终都会严格按照其发起顺序被应用到 Zookeeper 中；
>
> **原子性**：所有事务请求的处理结果在整个集群中所有机器上都是一致的；不存在部分机器应用了该事务，而另一部分没有应用的情况；
>
> **单一视图**：所有客户端看到的服务端数据模型都是一致的；
>
> **可靠性**：一旦服务端成功应用了一个事务，则其引起的改变会一直保留，直到被另外一个事务所更改；
>
> **实时性**：一旦一个事务被成功应用后，Zookeeper 可以保证客户端立即可以读取到这个事务变更后的最新状态的数据。

Zookeeper 概念

Znode节点信息

每个 ZNode 节点在存储数据的同时，都会维护一个叫做 `Stat` 的数据结构，里面存储了关于该节点的全部状态信息。

> | 状态属性 | 说明 |
> | --- | --- |
> | czxid | 数据节点创建时的事务 ID |
> | ctime | 数据节点创建时的时间 |
> | mzxid | 数据节点最后一次更新时的事务 ID |
> | mtime | 数据节点最后一次更新时的时间 |
> | pzxid | 数据节点的子节点最后一次被修改时的事务 ID |
> | cversion | 子节点的更改次数 |
> | version | 节点数据的更改次数 |
> | aversion | 节点的 ACL 的更改次数 |

会话

1. session是客户端与zk服务端之间建立的长连接。
2. zk在一个会话中进行心跳检测来感知客户端链接的存活。
3. zk客户端在一个会话中接收来自服务端的watch事件通知。
4. zk可以给会话设置超时时间。

zk中的版本

zk中由三种类型的版本 1、version:代表当前Znode的版本 2、Cversion：代表当前Znode的子节点的版本，子节点发生变化时会增加该版本号的值 3、Aversion：代表当前Znode的ACL（访问控制）的版本，修改节点的访问控制权限会增加该版本号的值。

zk中的ACL（访问控制）

类似Linux/Unix的权限控制，有以下几种控制访问权限： 1、CREATE：创建子节点的权限 2、DELETE：删除子节点的权限 3、READ：获取节点数据和子节点列表的权限 4、WRITE：更新节点数据的权限 5、ADMIN：设置节点ACL的权限

Znode的访问控制

zk中ACL由三部分组成，即Scheme🆔permission。其中： 1、scheme是验证过程中使用的检验策略 2、id是权限被赋予的对象，比如ip或某个用户 3、permission为可以操作的权限

权限检验策略即scheme有五种类型：world/auth/digest/IP/super

ZXID

在Zookeeper 中，事务是指能够改变 Zookeeper 服务器状态的操作，我们也称之为事务操作或更新操作，一般包括数据节点创建与删除、数据节点内容更新和客户端会话创建与失效等操作。对于每一个事务请求，Zookeeper 都会为其分配一个全局唯一的事务ID，用 ZXID 来表示，通常是一个 64 位的数字。每一个 ZXID 对应一次更新操作，从这些 ZXID 中可以间接地识别出 Zookeeper 处理这些更新操作请求的全局顺序。

ZXID 是一个 64 位的数字，其中低 32 位可以看作是一个简单的单调递增的计数器，针对客户端的每一个事务请求，Leader 服务器在产生一个新的事务 Proposal 的时候，都会对该计数器进行加 1 操作；而高 32 位则代表了 Leader 周期 epoch 的编号，每当选举产生一个新的 Leader 服务器，就会从这个 Leader 服务器上取出其本地日志中最大事务 Proposal 的 ZXID，并从该 ZXID 中解析出对应的 epoch 值，然后再对其进行加 1 操作，之后就会以此编号作为新的 epoch，并将低 32 位置 0 来开始生成新的 ZXID。

服务器状态

- looking：寻找leader状态。当服务器处于该状态时，它会认为当前集群中没有Leader，因此需要进入 Leader 选举流程。
- leading：领导者状态。表明当前服务器角色是leader。
- following：跟随者状态。表明当前服务器角色是follower。
- observing：观察者状态。表明当前服务器角色是observer。

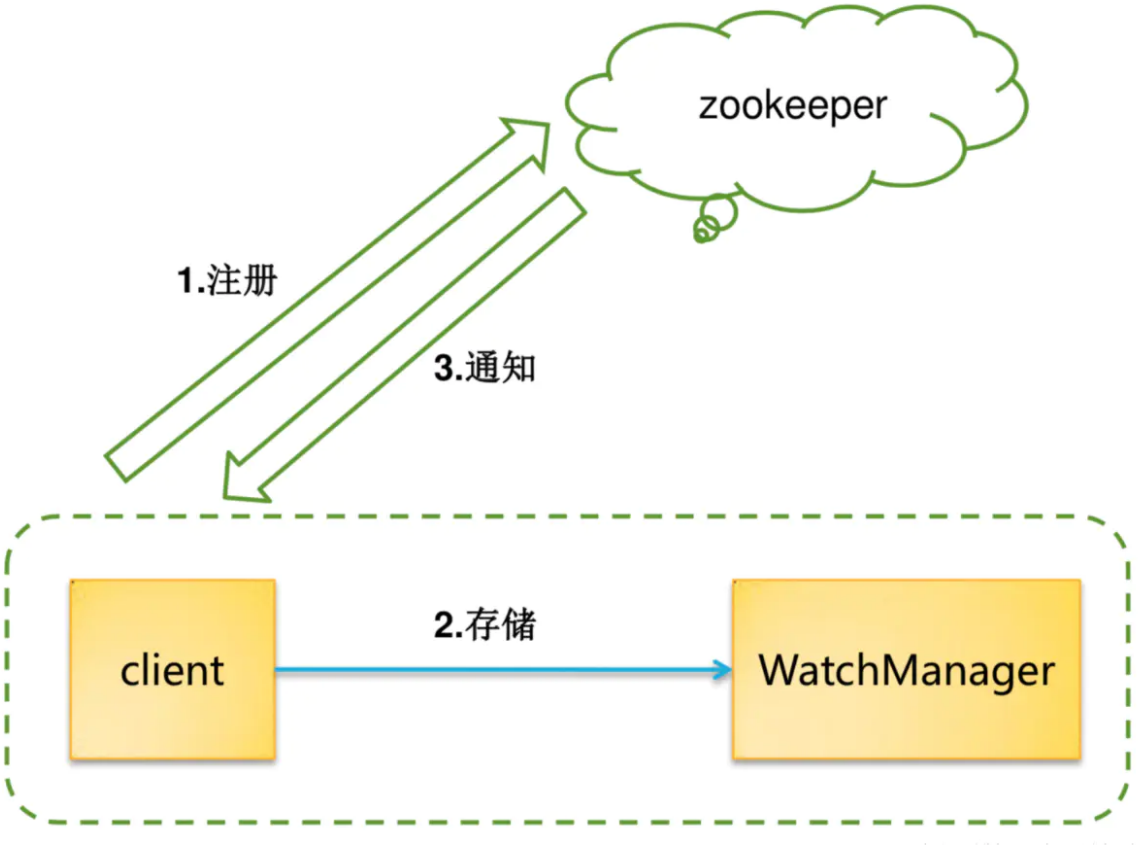
处理投票

针对每一个投票，服务器都需要将别人的投票和自己的投票进行pk，pk规则如下

- 优先检查zxid。zxid比较大的服务器优先作为leader。
- 如果zxid相同，那么就比较myid。myid较大的服务器作为leader服务器

watcher基本原理

zk实现watcher需要三个部分：分别是zk服务端，zk客户端和客户端的watchManager。

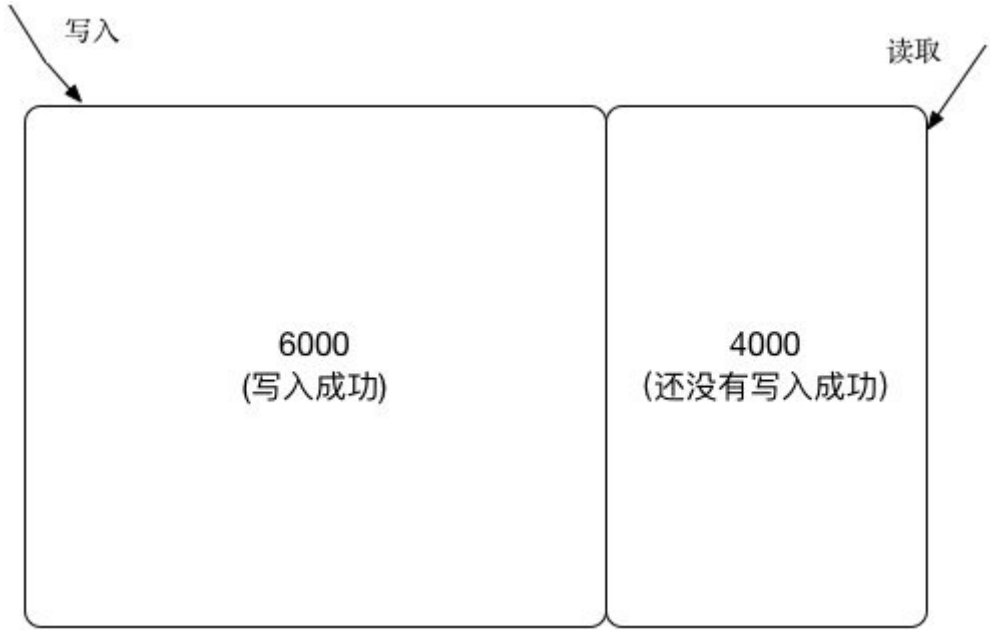


客户端向zk注册watcher的同时，会将客户端的watcher对象存储在客户端的watchManager中。zk服务器触发watch事件后，会向客户端发送通知，客户端线程从watchManager中取出对应watcher执行。Watcher是一次性的

Watcher特性

特性	说明
一次性	Watcher是一次性的，一旦被触发就会移除，再次使用时需要重新注册
客户端顺序回调	Watcher回调是顺序串行化执行的，只有回调后客户端才能看到最新的数据状态。一个Watcher回调逻辑不应该太多，以免影响别的watcher执行
轻量级	WatchEvent是最小的通信单元，结构上只包含通知状态、事件类型和节点路径，并不会告诉数据节点变化前后的具体内容；
时效性	Watcher只有在当前session彻底失效时才会无效，若在session有效期内快速重连成功，则watcher依然存在，仍可接收到通知；

Zookeeper并不保证读取的是最新数据



ZooKeeper并不保证在每个实例中，两个不同的客户端将具有相同的ZooKeeper数据的视图。由于诸如网络延迟的因素，一个客户端可以在另一客户端被通知该改变之前执行更新，考虑两个客户端A和B的场景。如果客户端A将znode / a的值从0设置为1，则告诉客户端B读取/ a，则客户端B可以读取旧值0，这取决于它连接到的服务器。如果客户端A和客户端B读取相同的值很重要，则客户端B应该在执行读取之前从ZooKeeper API方法调用**sync()**方法。

对于zookeeper来说，它实现了A可用性、P分区容错性、C中的写入强一致性，丧失的是C中的读取一致性。

参考： https://juejin.cn/post/6844904177815011341#heading-9