Zookeeper

-子慕

目录

Contents

Zookeeper基础及高级应用精讲

Zookeeper

概述

ZooKeeper 是 Apache 软件基金会的一个软件项目，它为大型分布式计算提供开源的**分布式配置服务、同步服务和命名注册**。

ZooKeeper 的架构通过冗余服务实现高可用性（CP）。

Zookeeper 的设计目标是将那些复杂且容易出错的分布式一致性服务封装起来，构成一个高效可靠的原语集，并以一系列简单易用的接口提供给用户使用。

一个典型的分布式数据一致性的解决方案，分布式应用程序可以基于它实现诸如数据发布/订阅、负载均衡、命名服务、分布式协调/通知、集群管理、Master 选举、分布式锁和分布式队列等功能

Apache ZooKeeper™ Project Documentation Developers ASF

Welcome to Apache ZooKeeper™

Apache ZooKeeper is an effort to develop and maintain an open-source server which enables highly reliable distributed coordination.

What is ZooKeeper?

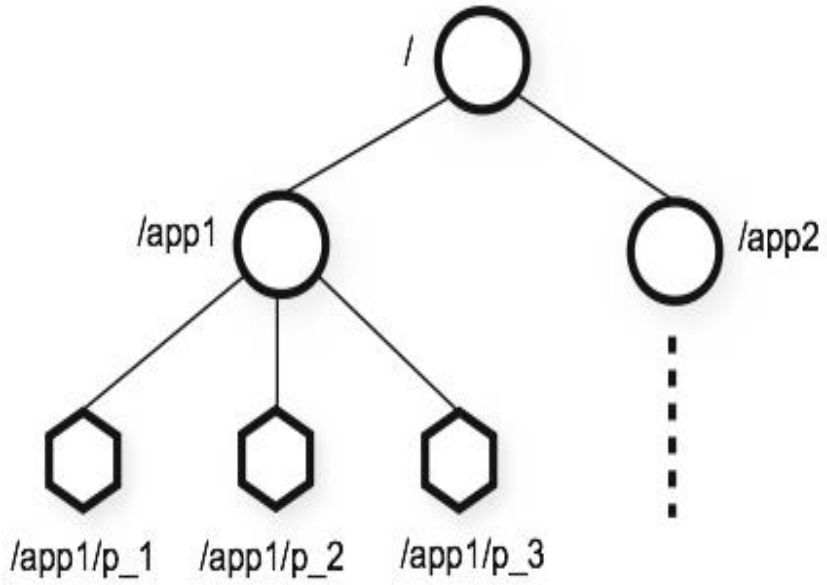
ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

Learn more about ZooKeeper on the ZooKeeper Wiki.

zookeeper数据结构

数据结构

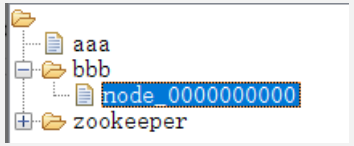
zookeeper本身是一个**树形目录服务**（名称空间），非常类似于标准文件系统，key-value 的形式存储。名称 key 由斜线 / 分割的一系列路径元素，zookeeper 名称空间中的每个节点都是由一个路径来标识的。



- 每个路径下的节点key(完整路径，名称)是唯一的，即同一级节点 key 名称是唯一的
- 每个节点中存储了节点value和对应的状态属性（多个）

Zookeeper节点操作

zookeeper节点类型

类型	描述
PERSISTENT	持久节点，默认
PERSISTENT_SEQUENTIAL	持久顺序节点，创建时zookeeper 会在路径上加上序号作为后缀，非常适合用于分布式锁、分布式选举等场景，创建时添加 -s 参数，如图： 
EPHEMERAL	临时节点(不可在拥有子节点)，跟连接会话绑定，临时节点会在客户端会话断开后由zk服务端自动删除。适用于心跳，服务发现等场景，创建时添加 -e 参数
EPHEMERAL_SEQUENTIAL	临时顺序节点(不可在拥有子节点)，与持久顺序节点类似，不同之处在于EPHEMERAL_SEQUENTIAL是临时的，会在会话断开后删除，创建时添加 -e -s 参数
CONTAINER	容器节点，当子节点都被删除后，Container 也随即删除，创建时添加 -c 参数
PERSISTENT_WITH_TTL	TTL节点，客户端断开连接后不会自动删除Znode，如果该Znode没有子Znode且在给定TTL时间内无修改，该Znode将会被删除；单位是毫秒；创建时添加 -t 参数

zookeeper节点操作的客户端基础命令-常见

命令	描述	用法示例	说明
ls	查看某个路径下目录列表	ls [-s] [-w] [-R] path	path：代表路径，完整路径 -s：返回状态信息 -w：监听节点变化 -R:递归查看某路径下目录列表
create	创建节点并赋值	create [-s] [-e] [-c] [-t ttl] path [data] [acl]	[-s] [-e]：-s 和 -e 都是可选的，-s 代表顺序节点， -e 代表临时节点，注意其中 -s 和 -e 可以同时使用的，并且临时节点不能再创建子节点 path：指定要创建节点的路径，比如 /runoob data：要在此节点存储的数据 acl：访问权限相关，默认是 world，相当于全世界都能访问
set	修改节点存储的数据	set [-s] [-v version] path data	path：节点路径。 data：需要存储的数据。 [version]：可选项，版本号(可用作乐观锁)
get	获取节点数据和状态信息	get [-s] [-w] path	-s：返回结果带上状态信息 -w:返回数据并对对节点进行事件监听
stat	查看节点状态信息	stat [-w] path	path：代表路径 -w:对节点进行事件监听
delete /deleteall	删除某节点	delete [-v version] path deleteall path [-b batch size]	如果某节点不为空，则不能用delete命令删除

数据模型 znode 结构详解

状态属性	描述
cZxid	创建节点时的事务ID
ctime	创建节点时的时间
mZxid	最后修改节点时的事务ID
mtime	最后修改节点时的时间
pZxid	表示该节点的子节点列表最后一次修改的事务ID，添加子节点或删除子节点就会影响子节点列表，但是修改子节点的数据内容则不影响该ID（注意，只有子节点列表变更了才会变更pzxid，子节点内容变更不会影响pzxid）
cversion	子节点版本号，子节点每次修改版本号加1
dataversion	数据版本号，数据每次修改该版本号加1
aclversion	权限版本号，权限每次修改该版本号加1
ephemeralOwner	创建该临时节点的会话的sessionID。（**如果该节点是持久节点，那么这个属性值为0）**
dataLength	该节点的数据长度
numChildren	该节点拥有子节点的数量（只统计直接子节点的数量）

zookeeper节点操作的客户端基础命令-其他


```
er -server host:port -client-configuration properties-file cmd args
addWatch [-m mode] path # optional mode is one of [PERSISTENT, PERSISTENT_RECURSIVE] - default is PERSISTENT_RECURSIVE
addauth scheme auth
close
config [-c] [-w] [-s]
connect host:port
create [-s] [-e] [-c] [-t ttl] path [data] [acl]
delete [-v version] path
deleteall path [-b batch size]
delquota [-n|-b|-N|-B] path
get [-s] [-w] path
getAcl [-s] path
getAllChildrenNumber path
getEphemerals path
history
listquota path
ls [-s] [-w] [-R] path
printwatches on|off
quit
reconfig [-s] [-v version] [[-file path] | [-members serverID=host:port1:port2;port3[,...]*]] | [-add serverId=host:port1:port2;port3[,...]]* [-remove server

redo cmdno
removewatches path [-c|-d|-a] [-l]
set [-s] [-v version] path data
setAcl [-s] [-v version] [-R] path acl
setquota -n|-b|-N|-B val path
stat [-w] path
sync path
version
whoami
```

zookeeper java客户端

java

1、zookeeper 原生API

2、Curator

3、zkClient

What is Curator?

Curator *n* *ˈkyoorˌātər*: a keeper or custodian of a museum or other collection - A ZooKeeper Keeper.

Apache Curator is a Java/JVM client library for Apache ZooKeeper, a distributed coordination service. It includes a highlevel API framework and utilities to make using Apache ZooKeeper much easier and more reliable. It also includes recipes for common use cases and extensions such as service discovery and a Java 8 asynchronous DSL.

"Guava is to Java what Curator is to ZooKeeper"
Patrick Hunt, ZooKeeper committer

sgroschupf / zkclient

Code | Issues 30 | Pull requests | Actions | Projects | Wiki | Security | Insights

master | 1 branch | 10 tags | Go to file | Add file | Code

高级应用原理

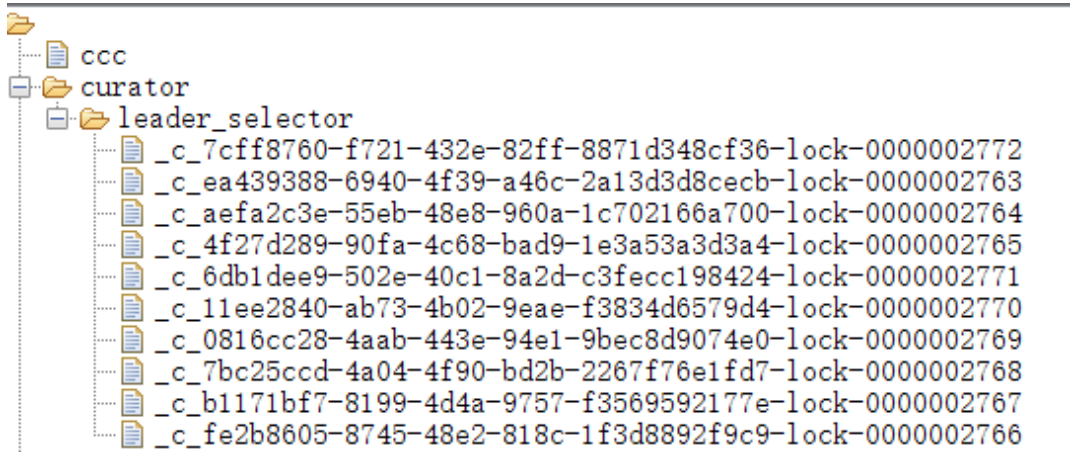
分布式锁

zk实现

分布式锁，指在分布式环境下，保护跨进程、跨主机、跨网络的共享资源，实现互斥访问，保证一致性；在zk中，锁就是一个数据节点

普通实现：注册临时节点，谁注册成功谁获取锁，其他监听该节点的删除事件，一旦被删除，通知其他客户端，再次重复该流程；此为最简单的实现，但容易引发一些问题

羊群效应

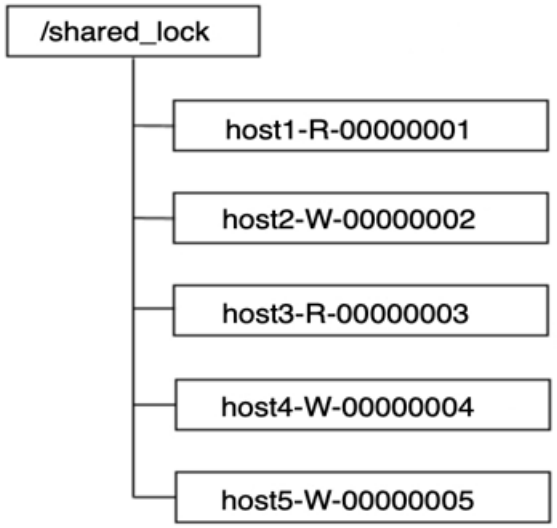


- 所有服务要获取锁时都去zookeeper中注册一个临时顺序节点，并将基本信息写入临时节点
- 所有服务获取节点列表并判断自己的节点是否是最小的那个，谁最小谁就获取了锁
- 未获取锁的客户端添加对前一个节点删除事件的监听
- 锁释放/持有锁的客户端宕机 后，节点被删除
- 下一个节点的客户端收到通知，重复上述流程

分布式锁-读写锁

共享锁，排它锁

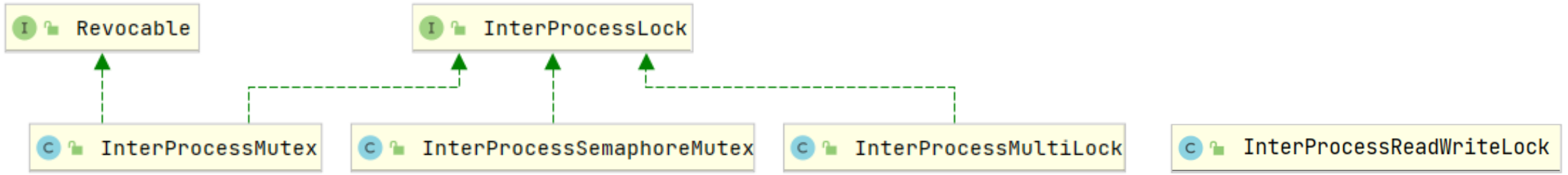
- 共享锁：又称读锁。如果事务T1对数据对象O1加上了共享锁，那么当前事务只能对O1进行读取操作，其他事务也只能对这个数据对象加共享锁，直到该数据对象上的所有共享锁都被释放
- 排它锁：又称写锁或独占锁。如果事务T1对数据对象O1加上了排他锁，那么在整个加锁期间，只允许事务T1对O1进行读取或更新操作，其他任务事务都不能对这个数据对象进行任何操作，直到T1释放了排他锁



可以将临时有序节点分为**读锁**节点和**写锁**节点

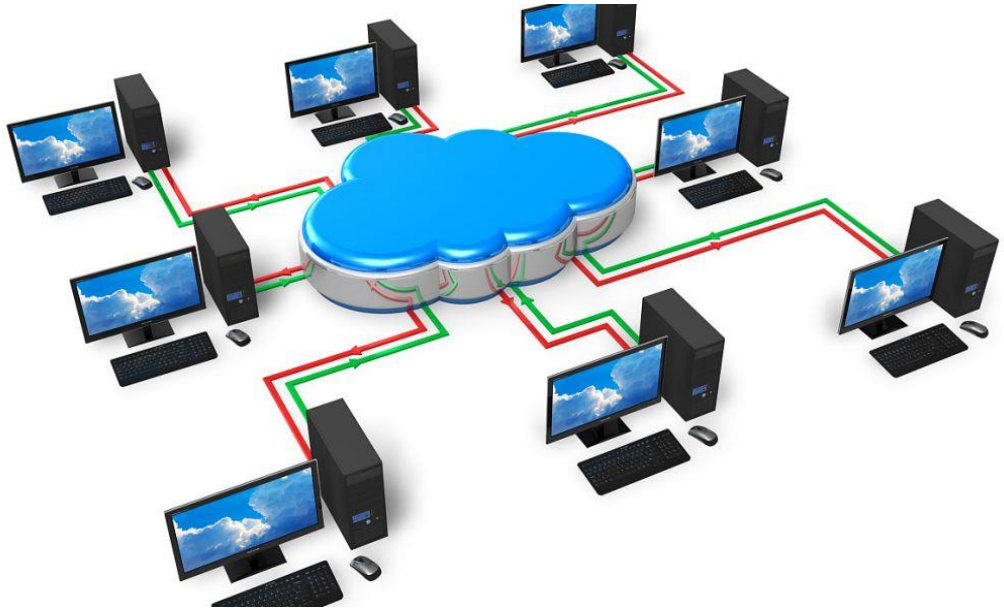
- 对于读锁节点而言，其只需要关心前一个写锁节点的释放。如果前一个写锁释放了，则多个读锁节点对应的线程可以并发地读取数据
- 对于写锁节点而言，其只需要关心前一个节点的释放，而不需要关心前一个节点是写锁节点还是读锁节点。因为为了保证有序性，写操作必须要等待前面的读操作或者写操作执行完成

分布式锁-Curator实现



- InterProcessMutex：分布式可重入排它锁（可重入可以借助LocalMap存计数器）
- InterProcessSemaphoreMutex：分布式排它锁
- InterProcessMultiLock：将多个锁作为单个实体管理的容器
- InterProcessReadWriteLock：分布式读写锁

Zookeeper选举策略



zookeeper 集群

zookeeper 集群节点个数配置

为什么zookeeper节点推荐配奇数?

- **容错率**：需要保证集群能够有半数进行投票

2台服务器，至少2台正常运行才行（2的半数为1，半数以上最少为2），正常运行1台服务器都不允许挂掉，但是相对于 单节点服务器，2台服务器还有两个单点故障，所以直接排除了。

3台服务器，至少2台正常运行才行（3的半数为1.5，半数以上最少为2），正常运行可以允许1台服务器挂掉

4台服务器，至少3台正常运行才行（4的半数为2，半数以上最少为3），正常运行可以允许1台服务器挂掉

5台服务器，至少3台正常运行才行（5的半数为2.5，半数以上最少为3），正常运行可以允许2台服务器挂掉

- **防脑裂**：脑裂集群的脑裂通常是发生在节点之间通信不可达的情况下，集群会分裂成不同的小集群，小集群各自选出自己的leader节点，导致原有的集群出现多个leader节点的情况，这就是脑裂

3台服务器，投票选举半数为1.5，一台服务裂开，和另外两台服务器无法通行，这时候2台服务器的集群（2票大于半数1.5票），所以可以选举出leader，而 1 台服务器的集群无法选举。

4台服务器，投票选举半数为2，可以分成 1,3两个集群或者2,2两个集群，对于 1,3集群，3集群可以选举；对于2,2集群，则不能选择，造成没有leader节点。

5台服务器，投票选举半数为2.5，可以分成1,4两个集群，或者2,3两集群，这两个集群分别都只能选举一个集群，满足zookeeper集群搭建数目。

以上分析，我们从容错率以及防止脑裂两方面说明了3台服务器是搭建集群的最少数目，4台发生脑裂时会造成没有leader节点的错误

zookeeper 选举

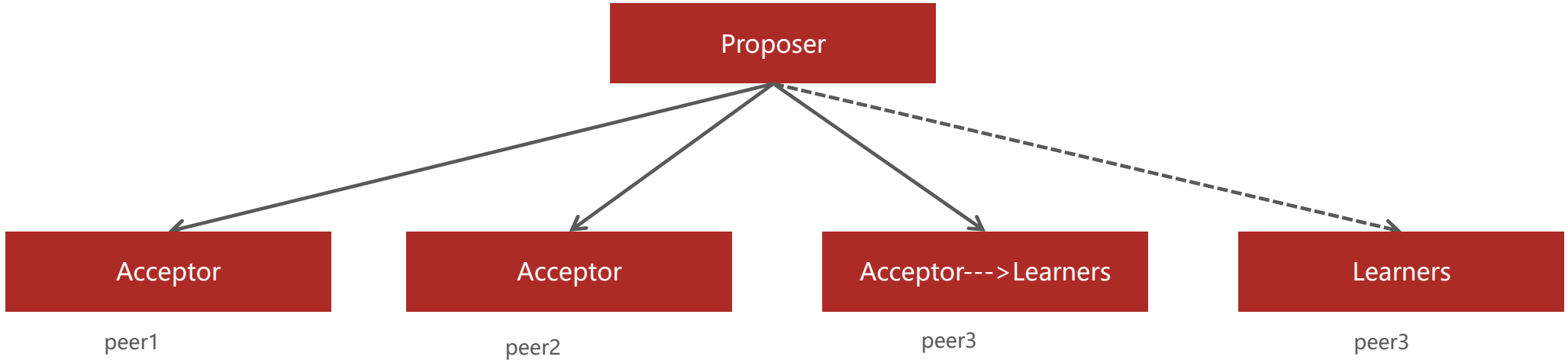
算法模型

Paxos算法是基于消息传递且具有高度容错特性的一致性算法，是目前公认的解决分布式一致性问题最有效的算法之一，其解决的问题就是在分布式系统中如何就某个值（决议）达成一致，paxos是一个分布式选举算法

该算法定义了三种角色

- Proposer：提案(决议)发起者
- Acceptor：提案接收者，可同意或不同意
- Learners：虽然不同意提案，但也只能被动接收学习；或者是后来的，只能被动接受

提案遵循少数服从多数的原则，过半原则。



zookeeper 选举

ZAB协议

ZooKeeper使用的是ZAB协议作为数据一致性的算法， ZAB（ZooKeeper Atomic Broadcast ） 全称为：原子消息广播协议。在Paxos算法基础上进行了扩展改造而来的，ZAB协议设计了支持原子广播、崩溃恢复，ZAB协议保证Leader广播的变更序列被顺序的处理。

四种状态，其中三种跟选举有关

- LOOKING：系统刚启动时或者Leader崩溃后正处于选举状态
- FOLLOWING：Follower节点所处的状态，同步leader状态，参与投票
- LEADING：Leader所处状态
- OBSERVING，观察状态,同步leader状态，不参与投票

选举时也是半数以上通过才算通过

zookeeper 选举

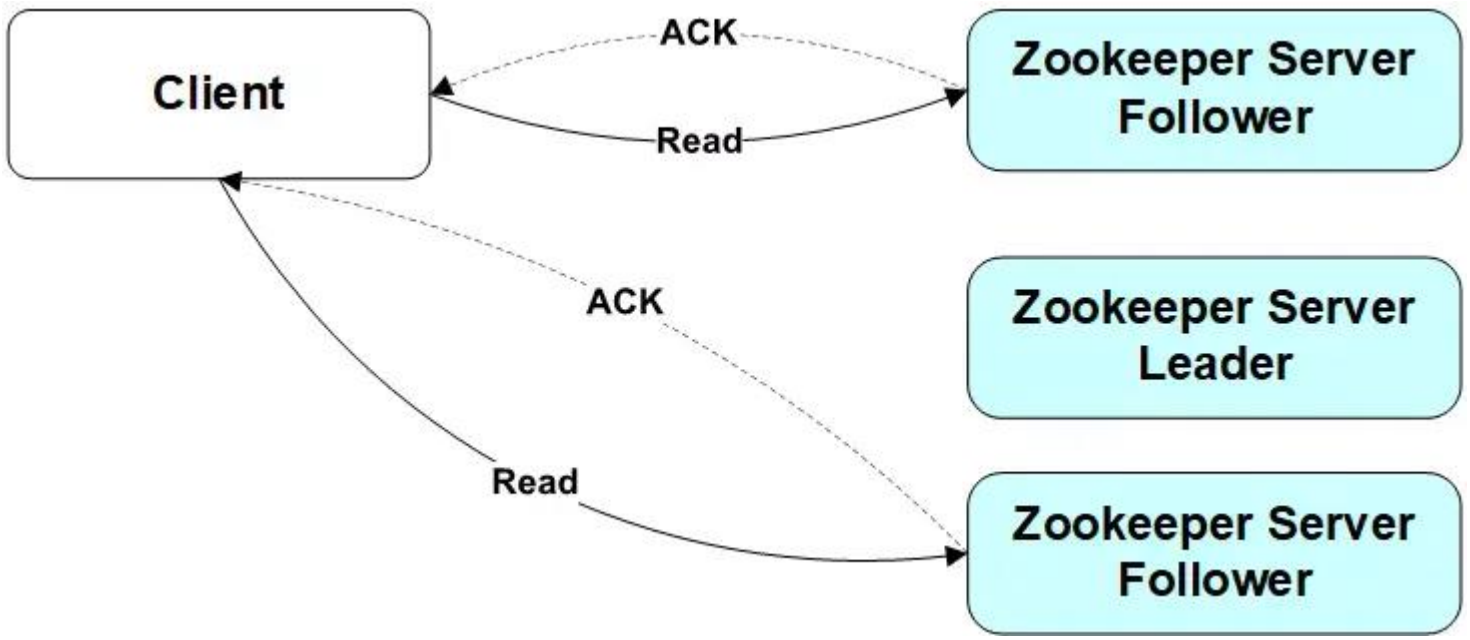
FastLeaderElection

```
protected boolean totalOrderPredicate(long newId, long newZxid, long newEpoch, long curId, long curZxid, long curEpoch) {
    LOG.debug(
        "id: {}, proposed id: {}, zxid: 0x{}, proposed zxid: 0x{}",
        newId,
        curId,
        Long.toHexString(newZxid),
        Long.toHexString(curZxid));

    if (self.getQuorumVerifier().getWeight(newId) == 0) {
        return false;
    }

    /*
     * We return true if one of the following three cases hold:
     * 1- New epoch is higher
     * 2- New epoch is the same as current epoch, but new zxid is higher
     * 3- New epoch is the same as current epoch, new zxid is the same
     *  as current zxid, but server id is higher.
     */
    return ((newEpoch > curEpoch)
            || ((newEpoch == curEpoch)
                && ((newZxid > curZxid)
                    || ((newZxid == curZxid)
                        && (newId > curId)))));
}
```

zookeeper 集群数据读写

读请求

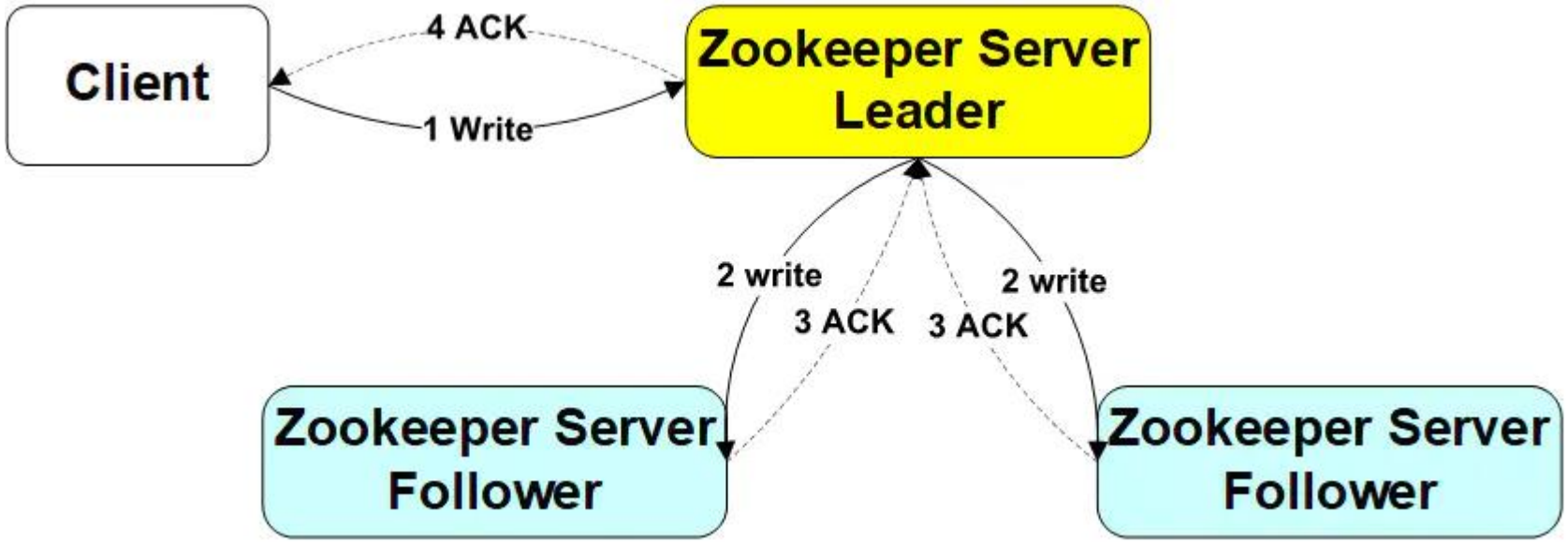
当Client向zookeeper发出读请求时，无论是Leader还是Follower，都直接返回查询结果



zookeeper 集群数据读写

写请求-leader

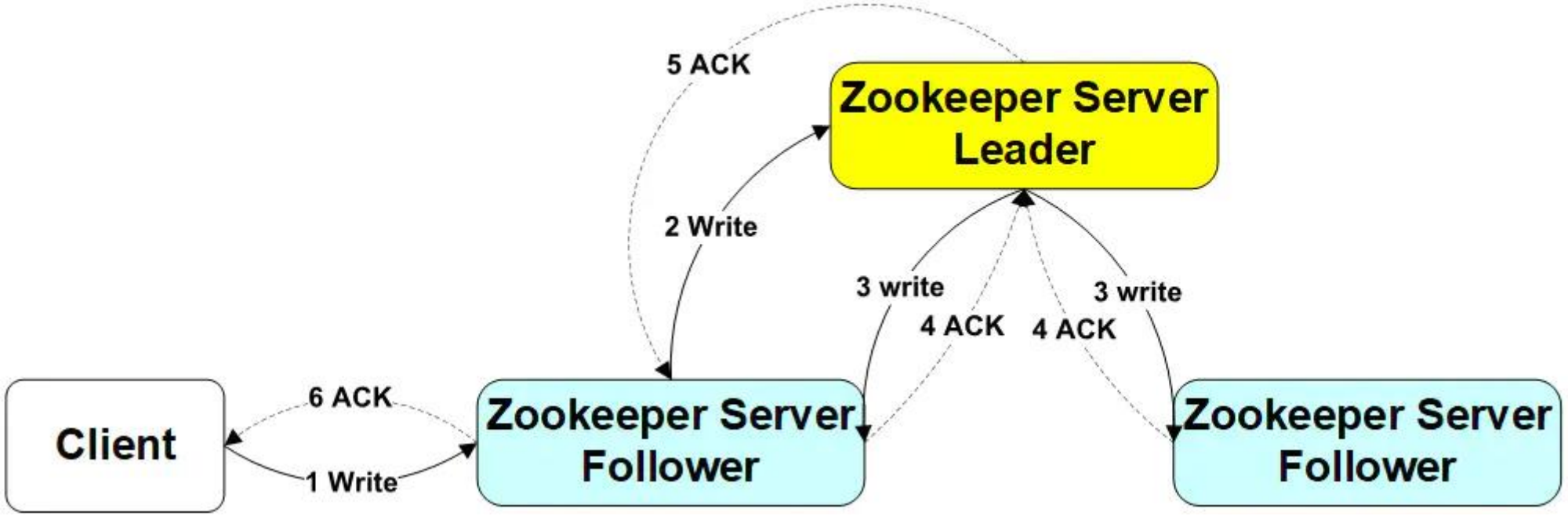
Client向Leader发出写请求，Leader将数据写入到本节点，并将数据发送到所有的Follower节点，等待Follower节点返回，当Leader接收到一半以上节点(包含自己)返回写成功的信息之后，返回写入成功消息给client



zookeeper 集群数据读写

写请求-follwer

Client向Follower发出写请求,Follower节点将请求转发给Leader,Leader将数据写入到本节点，并将数据发送到所有的Follower节点,等待Follower节点返回,当Leader接收到一半以上节点(包含自己)返回写成功的信息之后，返回写入成功消息给原来的Follower,原来的Follower返回写入成功消息给Client



传智教育旗下高端IT教育品牌