# 分布式协调服务-zookeeper

# 分布式环境的特点

## 分布性

## 并发性

程序运行过程中，并发性操作是很常见的。比如同一个分布式系统中的多个节点，同时访问一个共享资源。数据库、分布式存储

## 无序性

进程之间的消息通信，会出现顺序不一致问题

# 分布式环境下面临的问题

## 网络通信

网络本身的不可靠性，因此会涉及到一些网络通信问题

## 网络分区(脑裂)

当网络发生异常导致分布式系统中部分节点之间的网络延时不断增大，最终导致组成分布式架构的所有节点，只有部分节点能够正常通信

## 三态

在分布式架构里面，除了成功、失败、超时

## 分布式事务

ACID(原子性、一致性、隔离性、持久性)

# 中心化和去中心化

冷备或者热备

分布式架构里面，很多的架构思想采用的是：当集群发生故障的时候，集群中的人群会自动“选举”出一个新的领导。

最典型的是： zookeeper / etcd

# 经典的CAP/BASE理论

## CAP

C（一致性 Consistency）: 所有节点上的数据，时刻保持一致

可用性（Availability）：每个请求都能够收到一个响应，无论响应成功或者失败

**分区容错 （Partition-tolerance）：表示系统出现脑裂以后，可能导致某些server与集群中的其他机器失去联系**

CP / AP

CAP理论仅适用于原子读写的Nosql场景，不适用于数据库系统

## BASE

基于CAP理论，CAP理论并不适用于数据库事务（因为更新一些错误的数据而导致数据出现紊乱，无论什么样的数据库高可用方案都是

徒劳） ，虽然XA事务可以保证数据库在分布式系统下的ACID特性，但是会带来性能方面的影响；

eBay尝试了一种完全不同的套路，放宽了对事务ACID的要求。提出了BASE理论

Basically available ： 数据库采用分片模式， 把100W的用户数据分布在5个实例上。如果破坏了其中一个实例，仍然可以保证

80%的用户可用

soft-state： 在基于client-server模式的系统中，server端是否有状态，决定了系统是否具备良好的水平扩展、负载均衡、故障恢复等特性。

Server端承诺会维护client端状态数据，这个状态仅仅维持一小段时间, 这段时间以后，server端就会丢弃这个状态，恢复正常状态

Eventually consistent：数据的最终一致性

# 初步认识zookeeper

zookeeper是一个开源的分布式协调服务，是由雅虎创建的，基于google chubby。

## zookeeper是什么

分布式数据一致性的解决方案

## zookeeper能做什么

数据的发布/订阅（配置中心:disconf） 、 负载均衡（dubbo利用了zookeeper机制实现负载均衡） 、命名服务、

master选举(kafka、hadoop、hbase)、分布式队列、分布式锁

## zookeeper的特性

### 顺序一致性

从同一个客户端发起的事务请求，最终会严格按照顺序被应用到zookeeper中

### 原子性

所有的事务请求的处理结果在整个集群中的所有机器上的应用情况是一致的，也就是说，要么整个集群中的所有机器都成功应用了某一事务、

要么全都不应用

### 可靠性

一旦服务器成功应用了某一个事务数据，并且对客户端做了响应，那么这个数据在整个集群中一定是同步并且保留下来的

### 实时性

一旦一个事务被成功应用，客户端就能够立即从服务器端读取到事务变更后的最新数据状态；（zookeeper仅仅保证在一定时间内，近实时）

# zookeeper安装

## 单机环境安装

1. 下载zookeeper的安装包

http://apache.fayea.com/zookeeper/stable/zookeeper-3.4.10.tar.gz

1. 解压zookeeper

tar -zxvf zookeeper-3.4.10.tar.gz

1. cd 到 ZK\_HOME/conf , copy一份zoo.cfg

cp zoo\_sample.cfg zoo.cfg

1. sh zkServer.sh

{start|start-foreground|stop|restart|status|upgrade|print-cmd}

1. sh zkCli.sh -server ip:port

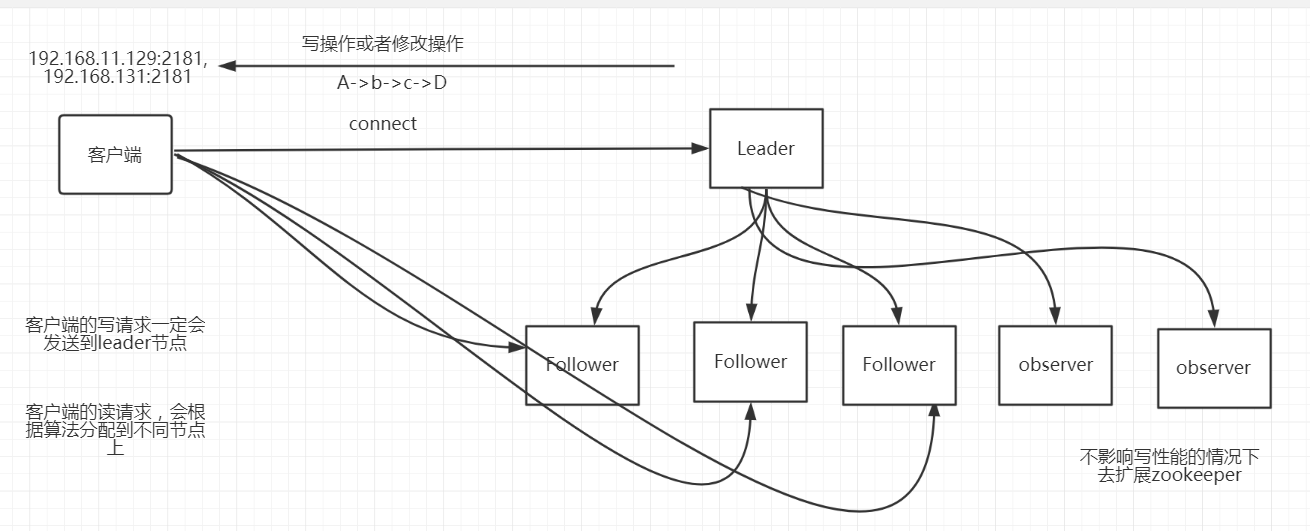
## 集群环境

zookeeper集群, 包含三种角色: leader / follower /**observer**

observer

observer 是一种特殊的zookeeper节点。可以帮助解决zookeeper的扩展性（如果大量客户端访问我们zookeeper集群，需要增加zookeeper集群机器数量。从而增加zookeeper集群的性能。 导致zookeeper写性能下降， zookeeper的数据变更需要半数以上服务器投票通过。造成网络消耗增加投票成本）

1. observer不参与投票。 只接收投票结果。
2. 不属于zookeeper的关键部位。



1. 在zoo.cfg里面增加

peerType=observer

server.1=192.168.11.129:2181:3181:observer

server.2=192.168.11.131:2181:3181

server.3=192.168.11.135:2181:3181

第一步： 修改配置文件

server.id=host:port:port

id的取值范围： 1~255； 用id来标识该机器在集群中的机器序号

2181是zookeeper的端口； //3306

3181表示leader选举的端口

server.1=192.168.11.129:2181:3181

server.2=192.168.11.131:2181:3181

server.3=192.168.11.135:2181:3181

第二步：创建myid

在每一个服务器的dataDir目录下创建一个myid的文件，文件就一行数据，数据内容是每台机器对应的server ID的数字

第三步：启动zookeeper

192.168.11.129

192.168.11.131

192.168.11.135

记得录屏

1. 分布式系统里面的特点
2. 分布式系统架构存在的问题
3. 中心化和去中心化
4. CAP和BASE
5. zookeeper的安装 单机环境安装/集群环境安装
6. zookeeper的特性

今天的内容

1. zookeeper的客户端使用
2. zoo.cfg里面配置信息的讲解
3. zookeeper的一些常见概念模型
4. zookeeper java客户端的使用

集群的角色： leader 、follower、 observer

# 集群的搭建

1. 修改zoo.cfg

129/135/136

server.id=ip:port:port

server.1=192.168.11.129:2888:3181 2888表示follower节点与leader节点交换信息的端口号 3181 如果leader节点挂掉了, 需要一个端口来重新选举。

server.2=192.168.11.135:2888:3181

server.3=192.168.111.136:2888:3181

1. zoo.cfg中有一个dataDir = /tmp/zookeeper

$dataDir/myid 添加一个myid文件。

1. 启动服务

如果需要增加observer节点

zoo.cfg中 增加 ;peerType=observer

server.1=192.168.11.129:2888:3181

server.2=192.168.11.135:2888:3181

server.3=192.168.111.136:2888:3181:observer

# zoo.cfg配置文件分析

tickTime=2000 zookeeper中最小的时间单位长度 （ms）

initLimit=10 follower节点启动后与leader节点完成数据同步的时间

syncLimit=5 leader节点和follower节点进行心跳检测的最大延时时间

dataDir=/tmp/zookeeper 表示zookeeper服务器存储快照文件的目录

dataLogDir 表示配置 zookeeper事务日志的存储路径，默认指定在dataDir目录下

clientPort 表示客户端和服务端建立连接的端口号： 2181

# zookeeper中的一些概念

## 数据模型

zookeeper的数据模型和文件系统类似，每一个节点称为：znode. 是zookeeper中的最小数据单元。每一个znode上都可以

保存数据和挂载子节点。 从而构成一个层次化的属性结构

节点特性

持久化节点 ： 节点创建后会一直存在zookeeper服务器上，直到主动删除

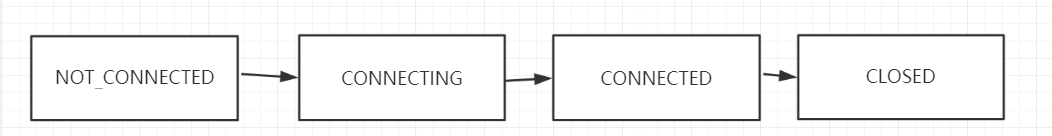
持久化有序节点 ：每个节点都会为它的一级子节点维护一个顺序

临时节点 ： 临时节点的生命周期和客户端的**会话**保持一致。当客户端会话失效，该节点自动清理

临时有序节点 ： 在临时节点上多勒一个顺序性特性

## 会话

**参考ppt**



## Watcher

**zookeeper提供了分布式数据发布/订阅,zookeeper允许客户端向服务器注册一个watcher监听。当服务器端的节点触发指定事件的时候**

**会触发watcher。服务端会向客户端发送一个事件通知  
watcher的通知是一次性，一旦触发一次通知后，该watcher就失效**

## ACL

zookeeper提供控制节点访问权限的功能，用于有效的保证zookeeper中数据的安全性。避免误操作而导致系统出现重大事故。

CREATE /READ/WRITE/DELETE/ADMIN

# zookeeper的命令操作

## 1. create [-s] [-e] path data acl

-s 表示节点是否有序

-e 表示是否为临时节点

默认情况下，是持久化节点

## 2. get path [watch]

获得指定 path的信息

## 3.set path data [version]

修改节点 path对应的data

乐观锁的概念

数据库里面有一个 version 字段去控制数据行的版本号

## 4.delete path [version]

删除节点

# stat信息

cversion = 0 子节点的版本号

aclVersion = 0 表示acl的版本号，修改节点权限

dataVersion = 1 表示的是当前节点数据的版本号

czxid 节点被创建时的事务ID

mzxid 节点最后一次被更新的事务ID

pzxid 当前节点下的子节点最后一次被修改时的事务ID

ctime = Sat Aug 05 20:48:26 CST 2017

mtime = Sat Aug 05 20:48:50 CST 2017

cZxid = 0x500000015

ctime = Sat Aug 05 20:48:26 CST 2017

mZxid = 0x500000016

mtime = Sat Aug 05 20:48:50 CST 2017

pZxid = 0x500000015

cversion = 0

dataVersion = 1

aclVersion = 0

ephemeralOwner = 0x0 创建临时节点的时候，会有一个sessionId 。 该值存储的就是这个sessionid

dataLength = 3 数据值长度

numChildren = 0 子节点数

# java API的使用

1. 导入jar包

<**dependency**>  
 <**groupId**>org.apache.zookeeper</**groupId**>  
 <**artifactId**>zookeeper</**artifactId**>  
 <**version**>3.4.8</**version**>  
</**dependency**>

1. 具体见代码

# zkclient

# curator

记得录屏

1. 集群的部署回顾
2. 数据模型（节点的特性）
3. 会话的概念
4. 数据节点的信息（stat）
5. create/delete/set/get 操作zookeeper的数据节点
6. java api的使用（事件响应，watcher）

# java api的使用

权限控制模式

schema：授权对象

ip : 192.168.1.1

Digest : username:password

world : 开放式的权限控制模式，数据节点的访问权限对所有用户开放。 world:anyone

super ：超级用户，可以对zookeeper上的数据节点进行操作

## 连接状态

KeeperStat.Expired 在一定时间内客户端没有收到服务器的通知， 则认为当前的会话已经过期了。

KeeperStat.Disconnected 断开连接的状态

KeeperStat.SyncConnected 客户端和服务器端在某一个节点上建立连接，并且完成一次version、zxid同步

KeeperStat.authFailed 授权失败

## 事件类型

NodeCreated 当节点被创建的时候，触发

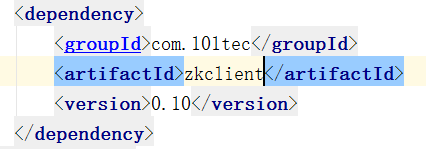
NodeChildrenChanged 表示子节点被创建、被删除、子节点数据发生变化

NodeDataChanged 节点数据发生变化

NodeDeleted 节点被删除

None 客户端和服务器端连接状态发生变化的时候，事件类型就是None

# zkclient



# curator

Curator本身是Netflix公司开源的zookeeper客户端；

curator提供了各种应用场景的实现封装

curator-framework 提供了fluent风格api

curator-replice 提供了实现封装

## curator连接的重试策略

ExponentialBackoffRetry() 衰减重试

RetryNTimes 指定最大重试次数

RetryOneTime 仅重试一次

RetryUnitilElapsed 一直重试知道规定的时间

# zookeeper的实际应用场景

zookeeper能够实现哪些场景

订阅发布

watcher机制

统一配置管理（disconf）

分布式锁

redis

zookeeper

数据库

负载均衡

ID生成器

分布式队列

统一命名服务

master选举

分布式锁

master选举

记得录屏

# 数据发布订阅/ 配置中心

实现配置信息的集中式管理和数据的动态更新

实现配置中心有两种模式：push 、pull。

长轮训

zookeeper采用的是推拉相结合的方式。 客户端向服务器端注册自己需要关注的节点。一旦节点数据发生变化，那么服务器端就会向客户端

发送watcher事件通知。客户端收到通知后，主动到服务器端获取更新后的数据

1. 数据量比较小
2. 数据内容在运行时会发生动态变更
3. 集群中的各个机器共享配置

# 负载均衡

请求/数据分摊多个计算机单元上

# 分布式锁

通常实现分布式锁有几种方式

1. redis。 setNX 存在则会返回0， 不存在
2. 数据方式去实现

创建一个表， 通过索引唯一的方式

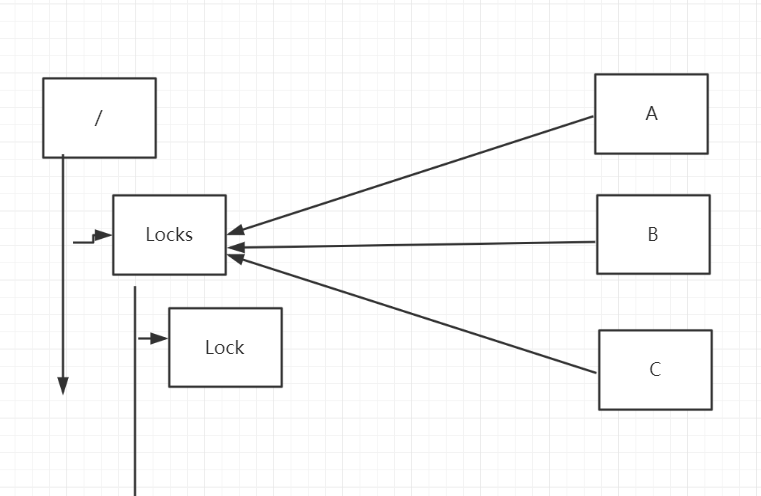
create table (id , methodname …) methodname增加唯一索引

insert 一条数据XXX delete 语句删除这条记录

mysql for update

1. zookeeper实现

排他锁



共享锁（读锁）

实现共享锁，使用java api的方式

命名服务

master选举

7\*24小时可用， 99.999%可用

master-slave模式

使用zookeeper解决

下堂课的内容：per实现原理讲解

**分布式队列**

1. **作业**

**master选举改成多线程(多进程)模型（master-slave） 创建三个工程，while去抢**

1. **分布式队列**

**activeMQ、kafka、….**

先进先出队列

1. 通过getChildren获取指定根节点下的所有子节点，子节点就是任务
2. 确定自己节点在子节点中的顺序
3. 如果自己不是最小的子节点，那么监控比自己小的上一个子节点，否则处于等待
4. 接收watcher通知，重复流程

Barrier模式

zookeeper数据模型

**临时节点（有序）、 持久化节点（有序）**

zookeeper是一个开源的分布式协调框架; 数据发布订阅、负载均衡、集群、master选举。。。

原子性： 要么同时成功、要么同时失败 （分布式事务）

单一视图： 无论客户端连接到哪个服务器，所看到的模型都是一样

可靠性：一旦服务器端提交了一个事务并且获得了服务器端返回成功的标识，那么这个事务所引起的服务器端的变更会一直保留

实时性： 近实时

zookeeper并不是用来存储数据的，通过监控数据状态的变化，达到基于数据的集群管理。

# 集群配置

1. 修改zoo.cfg

server.id=ip:port:port 第一个Port 数据同步通信、 第二个port ：leader选举（3181）

id=myid (myid 参与leader选举、 在整个集群中表示唯一服务器的标识)

1. dataDir目录下 创建一个myid的文件 ， 内容： server.id对应当前服务器的id号
2. 如果增加observer

需要在第一步中， server.id=ip:port:port:observer ; peerType=observer

# 会话

NOT\_CONNECTED - > CONNECTING ->CONNECTED ->ClOSE

# 数据模型

数据模型是一个树形结构，最小的数据单元是ZNODE

临时节点和持久化节点

临时有序节点

持久化有序节点

# 状态信息

Stat

cZxid = 0xb0000000f

ctime = Sun Aug 13 20:24:03 CST 2017

mZxid = 0xb0000000f

mtime = Sun Aug 13 20:24:03 CST 2017

pZxid = 0xb0000000f

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x15dda30f72f0000

dataLength = 2

numChildren = 0

zab协议 ： 如果客户端发了一个事务请求给到leader， 而leader发送给各个follower以后，并且收到了ack，leader已经commit。 在准备ack给各个follower节点comit的时候，leader挂了，怎么处理的。

1. 选举新的leader（zxid的最大值）
2. 同步给其他的folower

# watcher

EventyType

None 客户端与服务器端成功建立会话

NodeCreated 节点创建

NodeDeleted 节点删除

NodeDataChanged 数据变更：数据内容

NodeChildrenChanged 子节点发生变更： 子节点删除、新增的时候，才会触发

watcher的特性  
一次性触发： 事件被处理一次后，会被移除，如果需要永久监听，则需要反复注册

zkClient （ 永久监听的封装）

curator

java api的话， zk.exists , zk.getData 创建一个watcher监听

zookeeper序列化使用的是Jute

# Acl权限的操作

保证存储在zookeeper上的数据安全性问题

schema(ip/**Digest**/world/super)  
授权对象（192.168.1.1/11 , root:root / world:anyone/ super）

# 数据存储

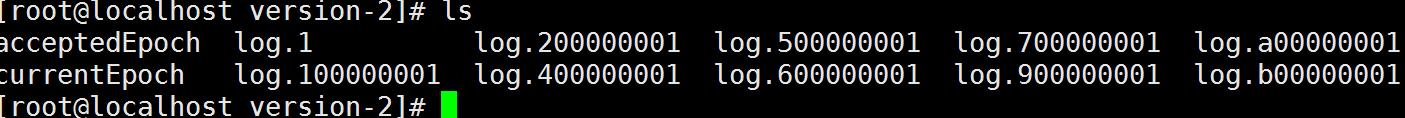
内存数据和磁盘数据

zookeeper会定时把数据存储在磁盘上。

DataDir = 存储的是数据的快照

快照： 存储某一个时刻全量的内存数据内容

DataLogDir 存储事务日志



log.zxid

查看事务日志的命令

java -cp :/mic/data/program/zookeeper-3.4.10/lib/slf4j-api-1.6.1.jar:/mic/data/program/zookeeper-3.4.10/zookeeper-3.4.10.jar org.apache.zookeeper.server.LogFormatter log.200000001

zookeeper 有三种日志

zookeeper.out //运行日志

快照 存储某一时刻的全量数据

事务日志 事务操作的日志记录

全剧终~

# Dubbo

dubbo.io

dubbo+spring boot +docker

## dubbo能解决什么问题

1. 怎么去维护url

通过注册中心去维护url（zookeeper、redis、memcache…）

1. F5硬件负载均衡器的单点压力比较大

软负载均衡

1. 怎么去整理出服务之间的依赖关系。

自动去整理各个服务之间的依赖

1. 如果服务器的调用量越来越大，服务器的容量问题怎么去评估，扩容的指标

需要一个监控平台，可以监控调用量、响应时间

## Dubbo是什么

dubbo是一个分布式的服务框架，提供高性能的以及透明化的RPC远程服务调用解决方法，以及SOA服务治理方案。

Dubbo的核心部分：

远程通信

集群容错

服务的自动发现

负载均衡

# Dubbo的架构

核心角色

Provider

Consumer

Registry

Monitor

Container

一台高性能的刀片机

32G内存 16核心

虚拟化

4G 2核心 – 8台

PAAS（platform-as-a-service）/IAAS(infrastucturre-as-a-service)/SAAS(软件即服务)

kvm vm window server

、记得录屏

curator 提供应用场景的封装

curator-reciples

master/leader选举

分布式锁（读锁、写锁）

分布式队列

…

LeaderLatch

写一个master

LeaderSelector

每一个应用都写一个临时有序节点，根据最小的节点来获得优先权

zookeeper的几个原理分析

# zookeeper集群角色

## leader

leader是zookeeper集群的核心。

1. 事务请求的唯一调度者和处理者，保证集群事务处理的顺序性
2. 集群内部各个服务器的调度者

## follower

1. 处理客户端非事务请求，以及转发事务请求给leader服务器
2. 参与事务请求提议（proposal）的投票（客户端的一个事务请求，需要半数服务器投票通过以后才能通知leader commit； leader会发起一个提案，要求follower投票）
3. 参与leader选举的投票

## observer

观察zookeeper集群中最新状态的变化并将这些状态同步到observer服务器上

增加observer不影响集群中事务处理能力，同时还能提升集群的非事务处理能力

# zookeeper的集群组成

zookeeper一般是由 2n+1台服务器组成

# leader选举

leaderElection/AuthFastLeaderElection/FastLeaderElection

QuorumPeer startLeaderElection

源码地址：<https://github.com/apache/zookeeper.git>

需要的条件： jdk 1.7以上 、ant 、idea

FastLeaderElection

serverid : 在配置server集群的时候，给定服务器的标识id（myid）

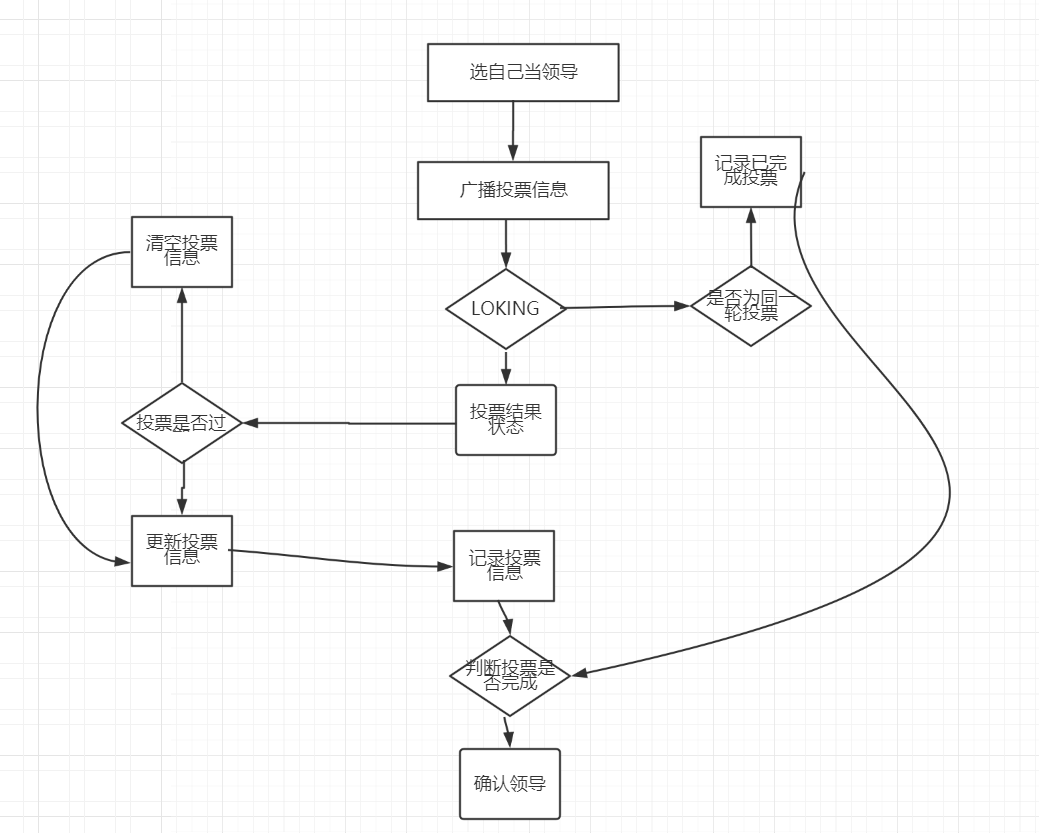
zxid : 服务器在运行时产生的数据ID， zxid的值越大，表示数据越新

Epoch: 选举的轮数

server的状态：Looking、 Following、Observering、Leading

第一次初始化启动的时候： LOOKING

1. 所有在集群中的server都会推荐自己为leader，然后把（myid、zxid、epoch）作为广播信息，广播给集群中的其他server, 然后等待其他服务器返回
2. 每个服务器都会接收来自集群中的其他服务器的投票。集群中的每个服务器在接受到投票后，开始判断投票的有效性
   1. 判断逻辑时钟(Epoch) ，如果Epoch大于自己当前的Epoch，说明自己保存的Epoch是过期。更新Epoch，同时clear其他服务器发送过来的选举数据。判断是否需要更新当前自己的选举情况
   2. 如果Epoch小于目前的Epoch，说明对方的epoch过期了，也就意味着对方服务器的选举轮数是过期的。这个时候，只需要讲自己的信息发送给对方



# ZAB协议

拜占庭问题

paxos协议主要就是如何保证在分布式环网络环境下，各个服务器如何达成一致最终保证数据的一致性问题

ZAB协议，基于paxos协议的一个改进。

zab协议为分布式协调服务zookeeper专门设计的一种支持崩溃恢复的原子广播协议

zookeeper并没有完全采用paxos算法， 而是采用zab Zookeeper atomic broadcast

## zab协议的原理

1. 在zookeeper 的主备模式下，通过zab协议来保证集群中各个副本数据的一致性
2. zookeeper使用的是单一的主进程来接收并处理所有的事务请求，并采用zab协议，

把数据的状态变更以事务请求的形式广播到其他的节点

1. zab协议在主备模型架构中，保证了同一时刻只能有一个主进程来广播服务器的状态变更
2. 所有的事务请求必须由全局唯一的服务器来协调处理，这个的服务器叫leader，其他的叫follower

leader节点主要负责把客户端的事务请求转化成一个事务提议（proposal），并分发给集群中的所有follower节点

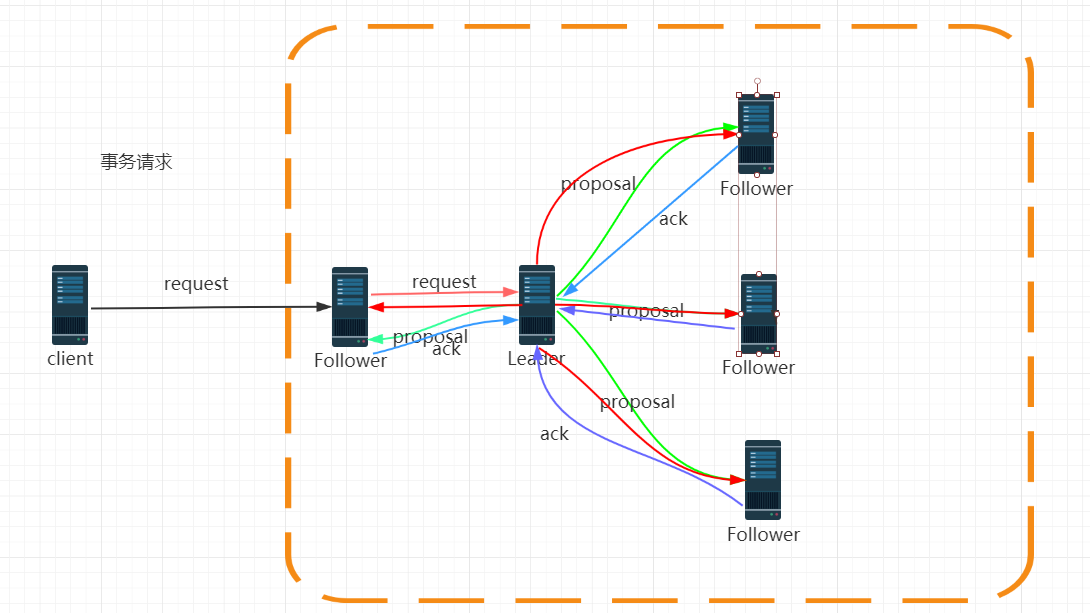
再等待所有follower节点的反馈。一旦超过半数服务器进行了正确的反馈，那么leader就会commit这条消息

崩溃恢复

原子广播

# zab协议的工作原理

1. 什么情况下zab协议会进入崩溃恢复模式
2. 当服务器启动时
3. 当leader服务器出现网络中断、崩溃或者重启的情况
4. 集群中已经不存在过半的服务器与该leader保持正常通信
5. zab协议进入崩溃恢复模式会做什么
6. 当leader出现问题，zab协议进入崩溃恢复模式，并且选举出新的leader。当新的leader选举出来以后，如果集群中已经有过半机器完成了leader服务器的状态同（数据同步），退出崩溃恢复，进入消息广播模式
7. 当新的机器加入到集群中的时候，如果已经存在leader服务器，那么新加入的服务器就会自觉进入数据恢复模式，找到leader进行数据同步



# 问题

假设一个事务在leader服务器被提交了，并且已经有过半的follower返回了ack。 在leader节点把commit消息发送给folower机器之前

leader服务器挂了怎么办

zab协议，一定需要保证已经被leader提交的事务也能够被所有follower提交

zab协议需要保证，在崩溃恢复过程中跳过哪些已经被丢弃的事务

# watcher的原理