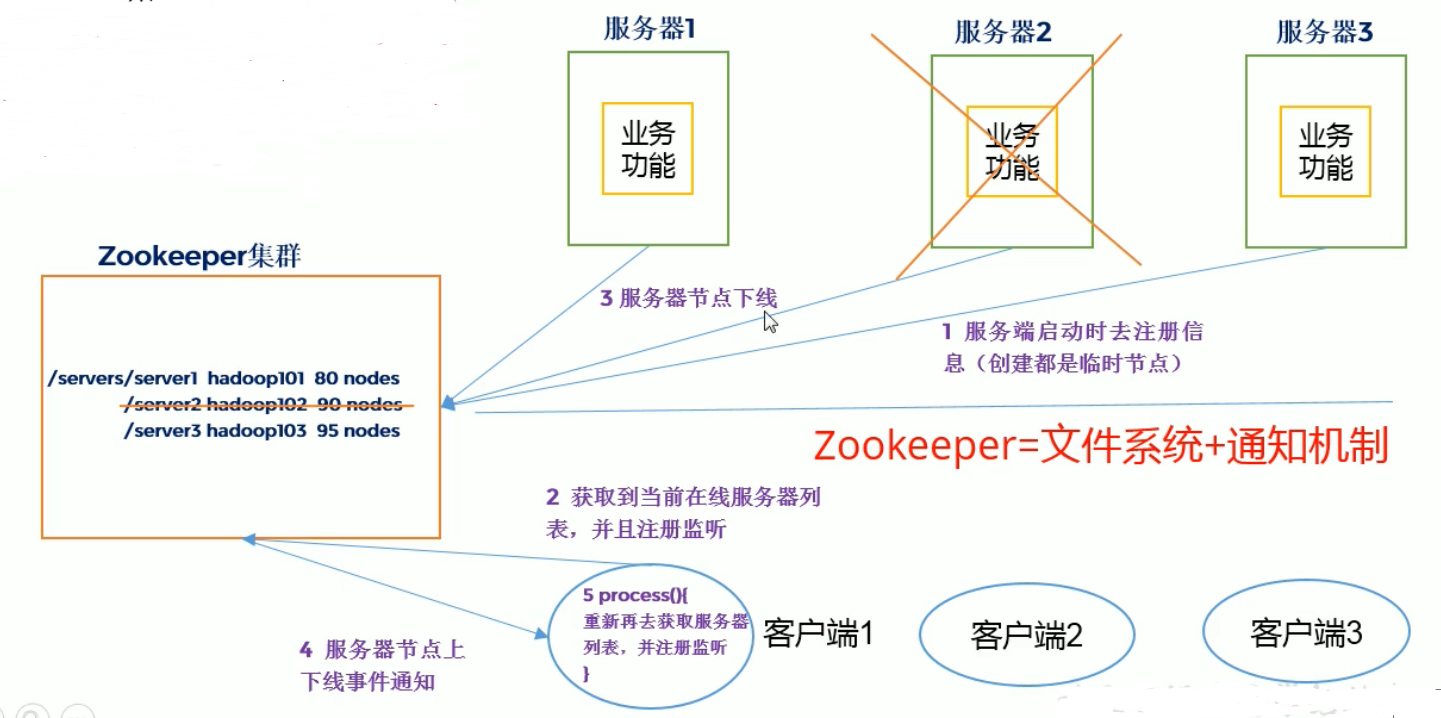
# 概述

Zookeeper是由雅虎研究院开发，是Google Chubby的开源实现，后者托管到Apache。

## 概述

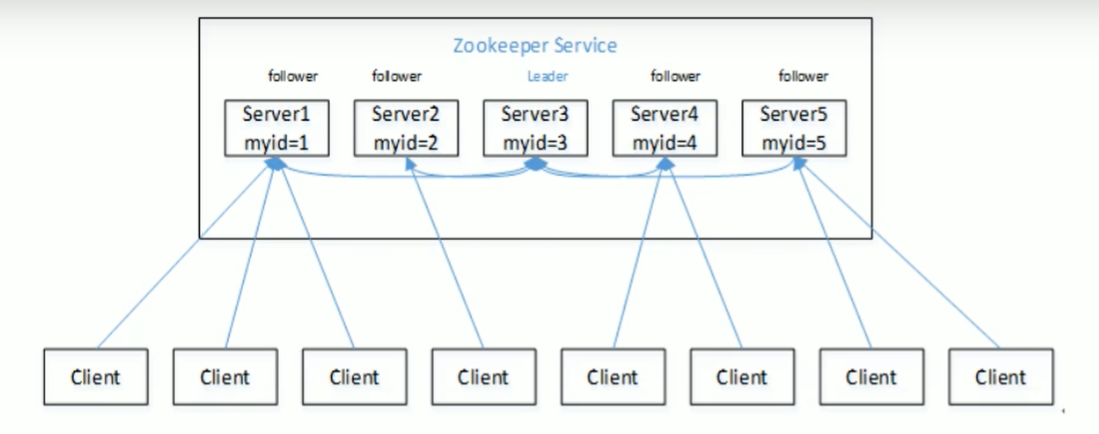
Zookeeper=文件系统+监听通知机制。

客户端注册监听它关心的目录节点，当目录节点发生变化（数据改变、被删除、子目录节点增加删除）时，zookeeper会通知客户端。



Zookeeper从设计模式角度来理解：是一个基于观察者模式设计的分布式服务管理框架，它负责存储和管理大家都关心的数据，然后观察者的注册，一旦这些数据的状态发生变化，zookeeper就将负责通知已经在zookeeper上注册的那些观察者做出相应的反应。

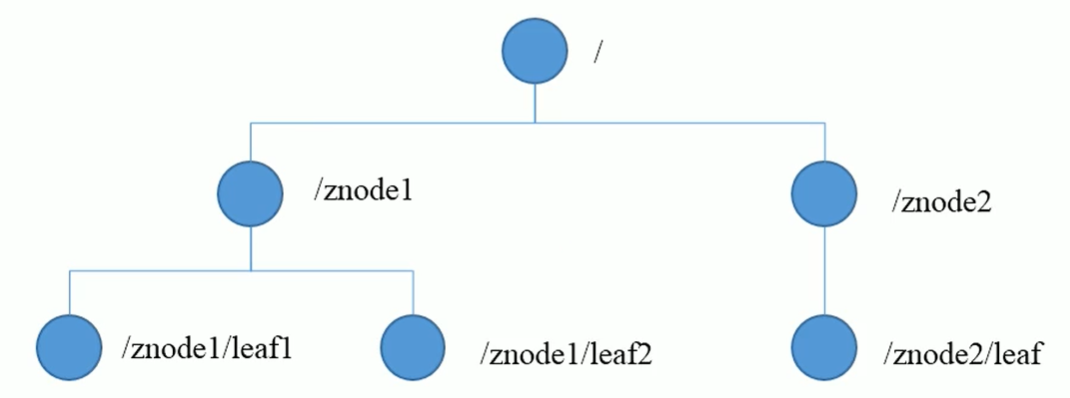
## 特点



1. zookeeper：一个领导者（leader），多个跟随者（follower）组成的集群；
2. 集群只要有半数以上节点存活，zookeeper集群就能正常服务；
3. 全局数据一致性：每个server保存一份相同的数据副本，client无论连接到哪个server，数据都是一致的；
4. 更新请求顺序进行，来自同一个client的更新请求按其发送顺序依次执行；
5. 数据更新原子性，一次数据更新要么成功，要么失败；
6. 实时性，在一定时间范围内，client能读到最新数据。

## 数据结构

Zookeeper数据模型的结构与Unix文件系统很相似，整体上可以看作是一棵树，每个节点称作一个ZNode。**每一个ZNode默认能够存储1MB的数据，每个ZNode都可以通过其路径唯一标识**。



## 设计目标

Zookeeper致力于为分布式应用提供一个高性能、高可用，并且具有严格顺序访问控制能力的分布式协调服务。

1. 高性能

Zookeeper将全量数据存储在内存中，并直接服务于客户端的所有非事务请求，尤其适用于以读为主的应用场景。

1. 高可用

Zookeeper一般以集群的方式对外提供服务，一般3~5台机器就可以组成一个可用的zookeeper集群，没台机器都会在内存中维护当前的服务器状态，并且每台机器之间都相互保持着通信。只要集群中超过一半的机器都能够正常工作，那么整个集群就能够正常对外服务。

1. 严格顺序访问

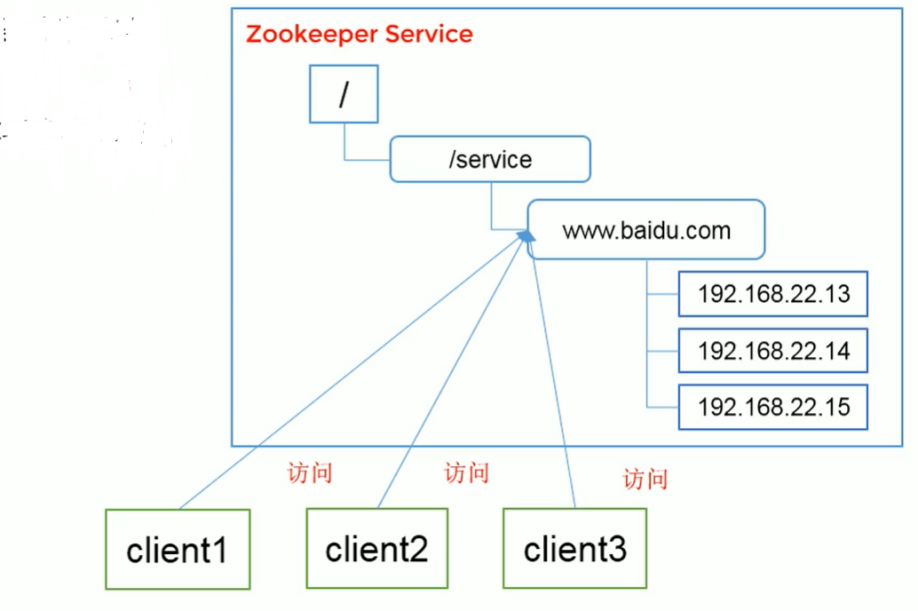
对于来自客户端的每个更新请求，zookeeper都会分配一个全局唯一的递增编号，这个编号反映了所有事务操作的先后顺序。

## 应用场景

提供服务包括：统一命名服务、统一配置管理、统一集群管理、服务器节点动态上下线、软负载均衡、分布式锁、分布式唯一ID等。

### 统一命名服务

在分布式环境下，经常需要对应用/服务进行统一命名，便于识别。例如：IP不容易记住，而域名容易记住。



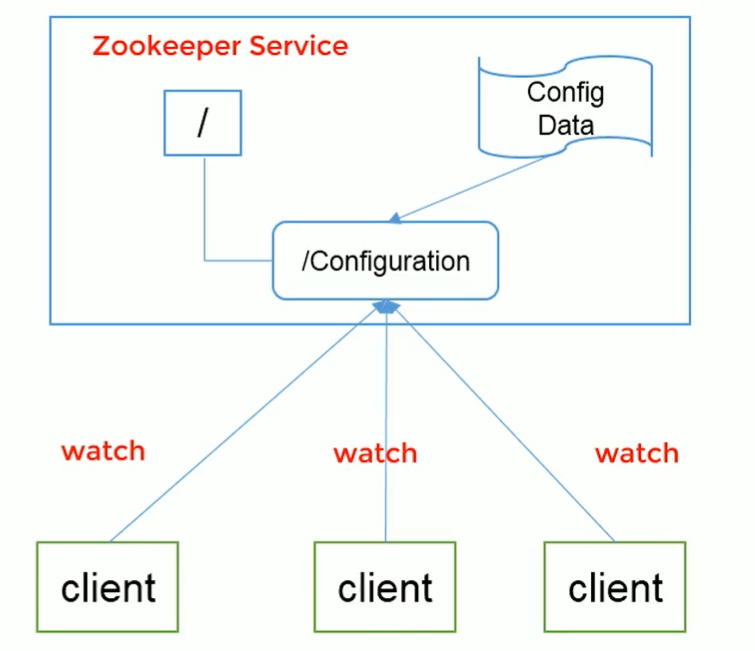
### 统一配置管理

在分布式环境下，配置文件同步非常常见：

1. 一般要求一个集群中，所有节点的配置信息是一致的，比如kafka集群；
2. 对配置文件修改后，希望能够快速同步到各个节点上；

配置管理可交由zookeeper实现：

1. 可将配置信息写入zookeeper上的一个ZNode；
2. 各个客户端服务器监听这个ZNode；
3. 一旦ZNode中的数据被修改，zookeeper将通知各个客户端服务器。

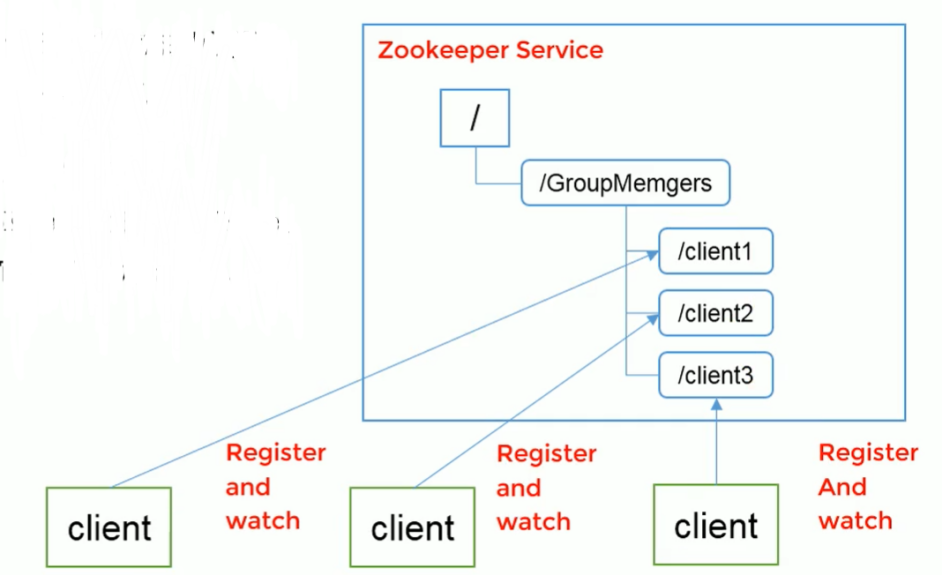


### 统一集群管理

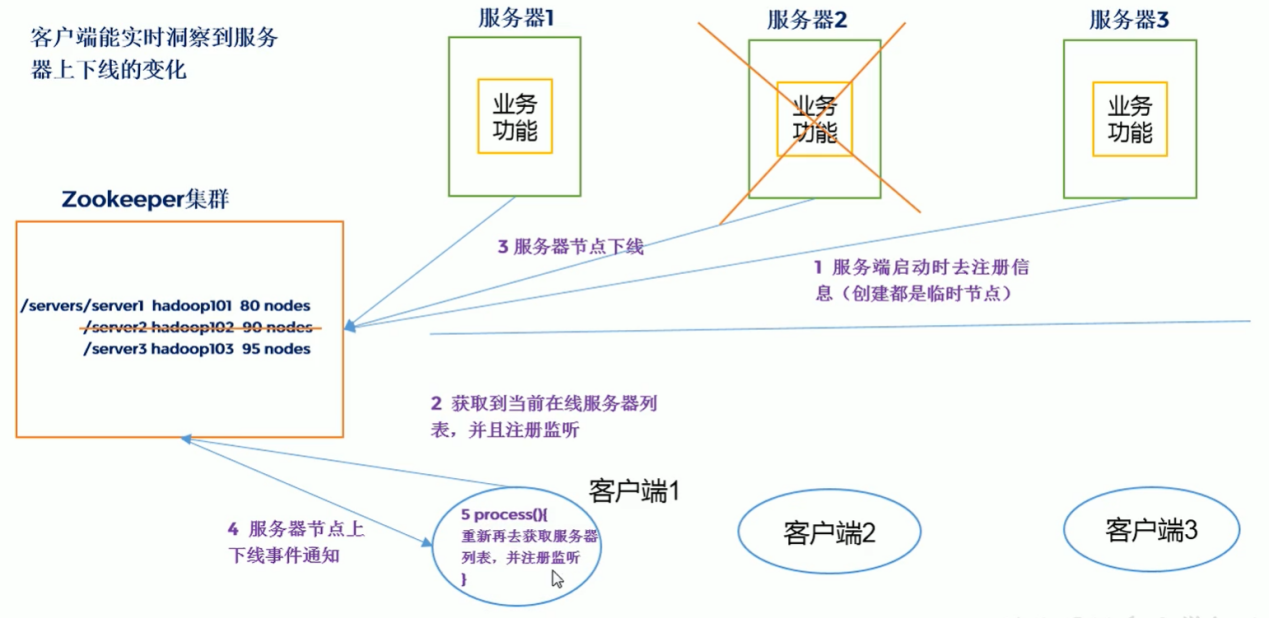
在分布式环境中，实时掌握每个节点的状态是必要的，可以根据节点实时状态做出一些调整。

Zookeeper可以实现实时监控节点状态变化：

1. 可将节点信息写入zookeeper上的一个ZNode；
2. 监听这个ZNode可获取它的实时状态变化。

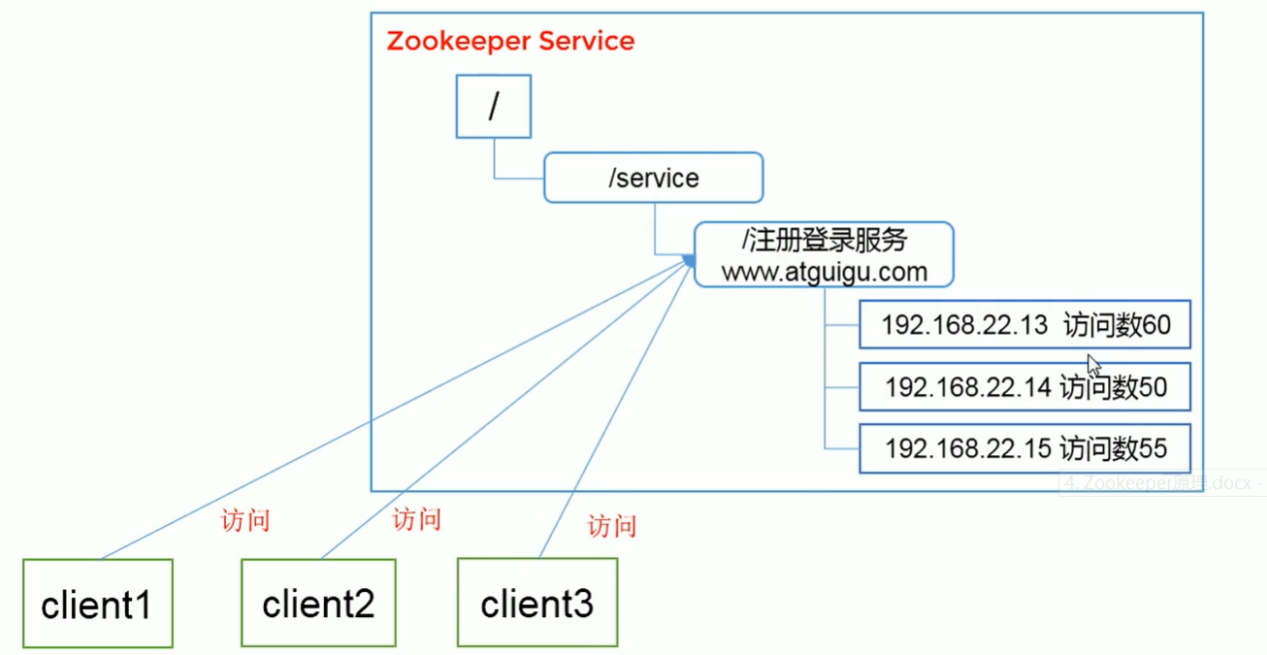


### 服务器节点动态上下线



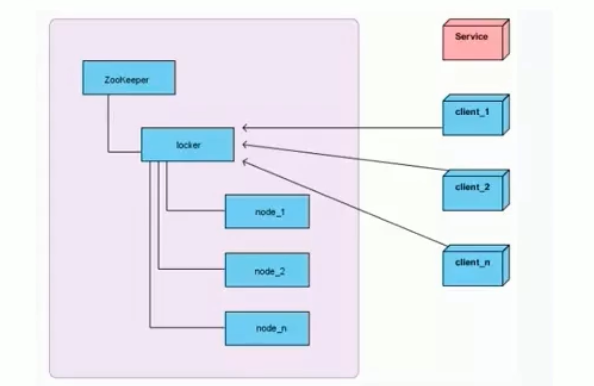
### 软负载均衡

在zookeeper中记录每台服务器的访问数，让访问数最少的服务器去处理最新的客户端请求。



### 分布式锁

一个集群是一个分布式系统，由堕胎服务器组成。为了提高并发度和可靠性，多台服务器上运行着同一服务。当多个服务在运行时就需要协调各个服务的进度，有时候需要保证某个服务在运行某个操作时，其他的服务都不能进行该操作，即对该操作进行加锁，如果当前机器挂掉后，释放锁并fail over到其他的机器继续执行该服务。



### 分布式唯一ID

在过去的单库单表型系统中，通常可以使用数据库字段自带的suto\_increment属性来自动为每条记录生成一个唯一的ID。但是分库分表后，就无法依靠数据库的auto\_increment属性来唯一标识一条记录了。此时，我们就可以用zookeeper的分布式环境下生成全局唯一ID。

做法如下：每次要生成一个新的ID时，创建一个持久顺序节点，创建操作返回的节点序号，即为新ID，然后把比自己节点小的删除即可。

# 安装配置

## 本地模式安装部署

1. 安装前准备
2. 安装JDK
3. 拷贝zookeeper安装包到Linux目录，并解压到指定目录
4. 配置修改
5. 将/opt/mudule/zookeeper-\*/conf路径下的zoo\_sample.cfg修改为zoo.cfg
6. 打开zoo.cfg，修改dataDir路径

dataDir=/opt/module/zookeeper-\*/zkData

1. 在/opt/mudule/zookeeper-\*目录下创建zkData文件夹
2. 操作zookeeper
3. 启动zookeeper：bin/zkServer.sh start
4. 查看进程是否启动：jps
5. 查看状态：bin/zkServer.sh status
6. 启动客户端：bin/zkCli.sh
7. 退出客户端：quit
8. 停止zookeeper：bin/zkServer.sh stop

## 配置参数

Zookeeper中的配置文件zoo.cfg中参数含义：

1. tickTime=2000：通信心跳数，zookeeper服务器与客户端心跳时间，单位毫秒

Zookeeper使用的基本时间，服务器之间或者客户端与服务器之间维持心跳的时间间隔，也就是每个tickTime时间就会发送一次心跳，时间单位为毫秒。

它用于心跳机制，并且设置最小的session超时时间为两倍心跳时间（session的最小超时时间是2\*tickTime）。

1. initLimit=10：LF初始通信时限

集群中的Fellower跟随者服务器与Leader领导者服务器之间初始连接时能容忍的最多心跳数（tickTime的数量），用它来限定集群中的zookeeper服务器连接到Leader的时限。

1. syncLimit=5：LF同步通信时限

集群中Leader与Fellower之间的最大响应时间单位，假如响应时间超过syncLimit\*tickTime，Leader认为Fellower死掉，从服务器列表中删除Fellower。

1. dataDir：数据文件目录+数据持久化路径

主要用于保存zookeeper中的数据。

1. clientPort=2181：客户端连接端口

监听客户端连接的端口。

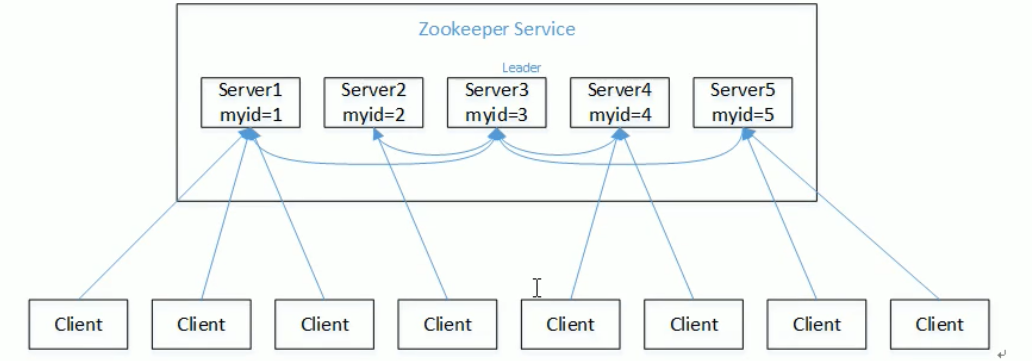
# 原理

## 选举机制

1. 半数机制：集群中半数以上机器存活，集群可用，所以zookeeper适合安装奇数台服务器。
2. Zookeeper虽然在配置文件中并没有制定master和slave，但是zookeeper工作时，是有一个节点为Leader，其他则为Fellower，Leader是通过内部的选举机制临时产生的。

举例：

假设有五台服务器组成的zookeeper集群，它们的id从1~5，同时它们都是最新启动的，也就是没有历史数据，在存放数据量这一点上，都是一样的。假设这些服务器依次启动，观察会发生什么：



1. 服务器1启动，此时只有它一台服务器启动了，它发出去的报文没有任何响应，所以它的选举状态一直是LOOKING状态。
2. 服务器2启动，它与最开始启动的服务器1进行通信，互相交换自己的选举结果，由于两者都没有历史数据，所以id值较大的服务器2胜出，但是由于没有达到超过半数以上的投票。
3. 服务器3启动，同样投票给自己，id值较大的服务器3胜出，此时达到半数以上的投票，产生Leader。
4. 服务器4启动，但是此时已经产生Leader，作为Fellower。
5. 服务器5启动，同样为Fellower。

## 数据模型

### 节点类型

1. PERSISTENT：持久化目录节点

客户端与zookeeper断开连接后，该节点依旧存在

1. PERSISTENT\_SEQUENTIAL：持久阿虎顺序编号目录节点

客户端与zookee断开连接后，该节点依旧存在，只是zookeeper给该节点名称进行顺序编号

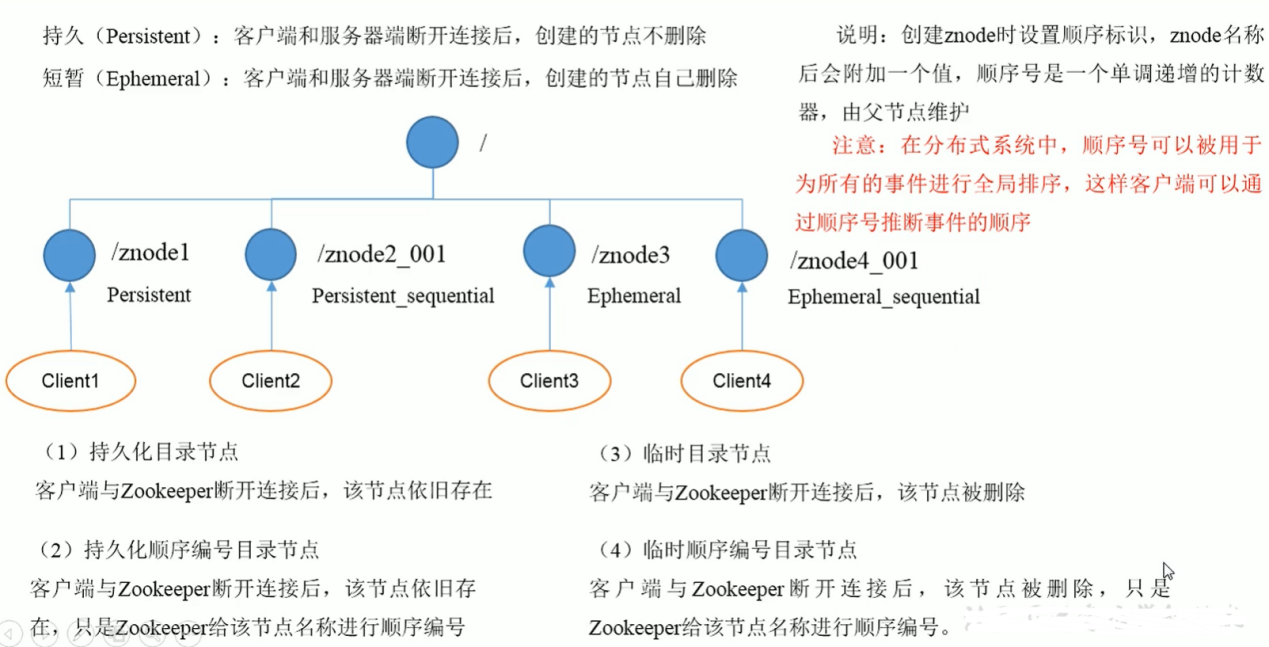
1. EPHEMERAL：临时目录节点

客户端与zookeeper断开连接后，该节点被删除

1. EPHEMERAL\_SEQUENTIAL：临时顺序编号目录节点

客户端与zookeeper断开连接后，该节点被删除，只是zookeeper给该节点名称进行顺序编号

注：对于持久节点和临时节点，同一个znode下，节点的名称是唯一的。

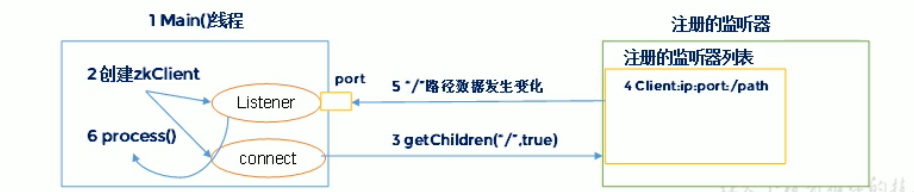


### Stat结构体

1. czxid - 引起这个znode创建的zxid，创建节点的事务的zxid
2. ctime - znode被创建的毫秒数(从1970年开始)
3. mzxid - znode最后更新的zxid
4. mtime - znode最后修改的毫秒数(从1970年开始)
5. pZxid-znode - 最后更新的子节点zxid
6. cversion - znode子节点变化号，znode子节点修改次数
7. dataversion - znode数据变化号
8. aclVersion - znode访问控制列表的变化号
9. ephemeralOwner - 如果是临时节点，这个是znode拥有者的session id。如果不是临时节点则是0
10. dataLength - znode的数据长度
11. numChildren - znode子节点数量

## 监听器/watcher

**原理：**



1. **监听原理：**
2. 首先要有一个main线程；
3. 在main线程中创建zookeeper客户端，这时就会创建两个线程，一个负责网络连接通信（connect），一个负责监听（listener）；
4. 通过connect线程将注册的监听事件发送给zookeeper；
5. 在zookeeper的注册监听器列表中将注册的监听事件添加到列表中；
6. Zookeeper监听到有效数据或路径变化，就会将这个消息发送给listener线程；
7. listener线程内部调用了process()方法。
8. **常见监听：**
9. 监听节点数据的变化：

get path [watch]

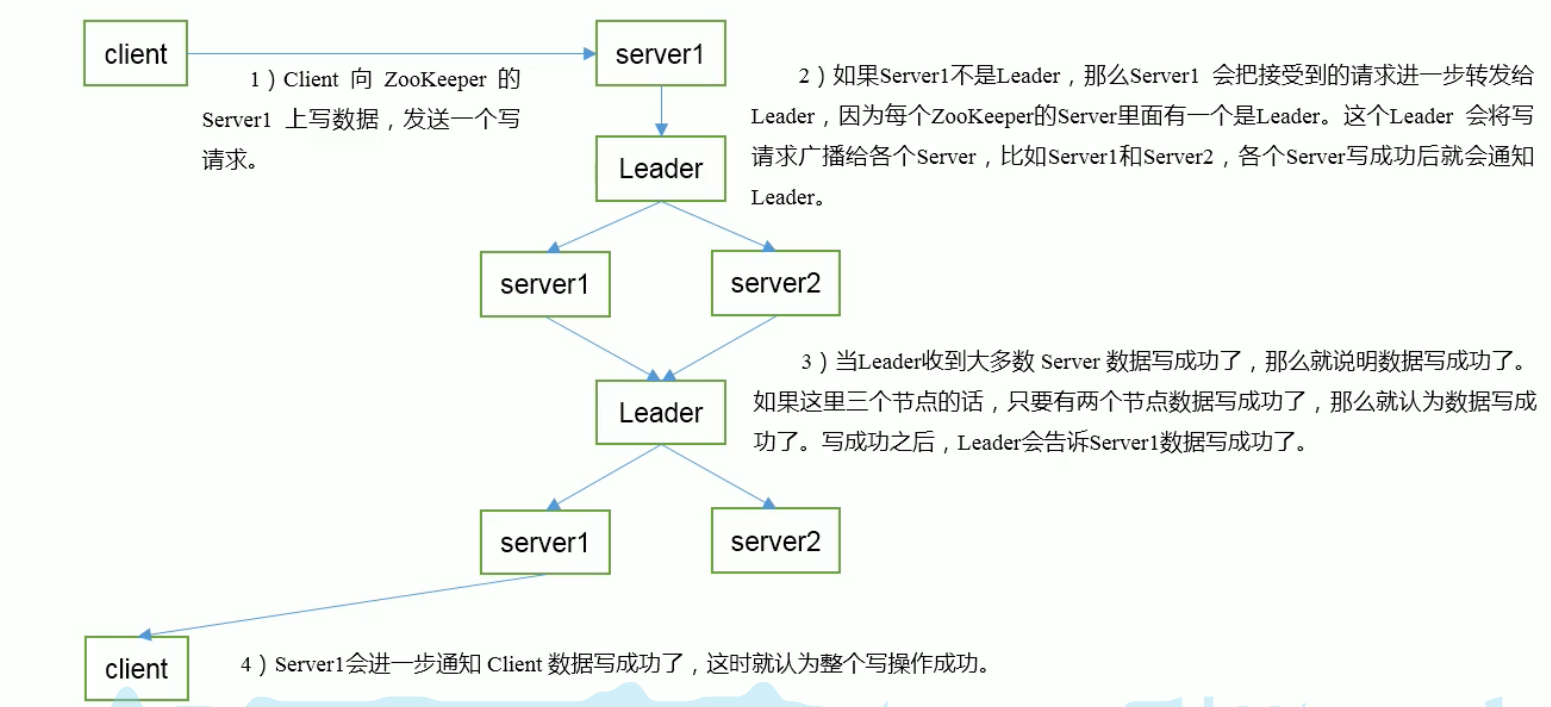
1. 监听子节点增减的变化：

ls path [watch]

**特性：**



## 写数据流程



# 部署方式

# ACL权限控制

## 概述

Zookeeper类似文件系统，client可以创建节点、更新节点、删除节点，那么如何做到节点的权限 控制呢？Zookeeper的access control list访问控制列表可以做到这一点。

ACL权限控制，使用scheme:id:permission来标识，主要涵盖三个方面：

权限模式（scheme）：授权的策略

授权对象（id）：授权的对象

权限（permission）：授予的权限

其特征如下：

1. zookeeper的权限控制是基于每个znode节点的，需要对每个节点设置权限
2. 每个znode支持设置多种权限控制方案和多个权限
3. 子节点不会继承父节点的权限，客户端无权访问某节点，但可能可以访问它的子节点

## 权限模式



### world授权模式

命令：setAcl <path> world:anyone:<acl>

### ip授权模式

命令：setAcl <path> ip:<ip>:<acl>

### auth授权模式

命令：

addauth digest <user>:<password> #添加认证用户

setAcl <path> auth:<user>:<acl>

### digest授权模式

命令：setAcl <path> digest:<user>:<password>:<acl>

### 多种授权模式

### ACL超级管理员

## 授权对象

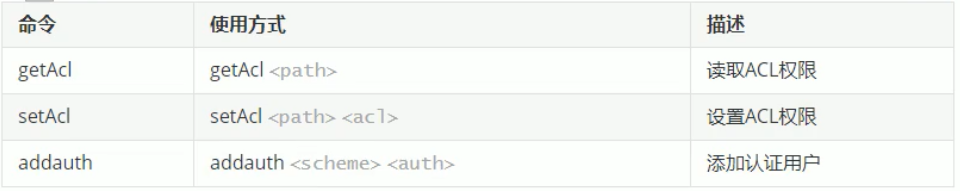
给谁授予权限，授权对象ID是指，权限赋予的实体，例如：IP地址或用户。

## 授予权限

create、delete、read、write、admin也就是增、删、改、查、管理权限，这5种权限简写为cdrma，注意，这5种权限中，delete是指对子节点的删除权限，其他4种权限指对自身节点的操作权限。



## 相关命令



# ZAB协议

# 常用指令

## 创建zookeeper客户端

## 新增节点

通过客户端对象的create方法创建节点。

## 修改节点

## 删除节点

## 查看节点和状态

## 返回节点链表

# 客户端

## 开源客户端curator

## 图形化客户端ZooInspector