# Zookeeper入门

## 概述

定义：Zookeeper是一个开源的分布式的应用程序协调服务。它包含一个简单的原语集，分布式系统可以基于它实现同步服务、配置服务和命名服务等，它是集群的管理者，监视着集群中各个节点的状态，将简单易用的接口和性能高效、功能稳定的系统提供给用户

Zookeeper工作机制：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，它负责存储和管理数据，然后客户端会在上面进行注册，一旦这些数据的状态发生变化，zookeeper就将负责通知已经在zookeeper上注册的客户端做出相应的响应

|  |
| --- |
|  |

## 特点

|  |
| --- |
|  |

1. zookeeper是由一个领导者(leader)和多个跟随者(follower)组成的集群
2. zookeeper中只要有半数以上的节点存活(eg:4个节点中有2个节点挂掉，集群将不能正常工作，5个节点中挂两个可以正常工作)，zookeeper集群就能正常服务
3. 全局数据一致：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. 更新请求数据进行，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，一次数据更新要么全部成功，要么全部失败
6. 实时性，在一定时间范围内，client都能读到最新数据

## 数据结构

|  |
| --- |
|  |

Zookeeper的数据结构整体上可以看作一棵树，每个节点称作一个ZNode，每个ZNode默认都能够存储1MB的数据，每个ZNode都可以通过其路径唯一标识

## 应用场景

提供的服务包括：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/服务进行统一命名服务，便于识别。

eg：ip不容易记住，而域名容易记住

|  |
| --- |
|  |

### 统一配置管理

1. 分布式环境下，配置文件同步非常常见

一般要求一个集群中，所有节点的配置信息都是一致的，eg：kafka集群；

对配置文件修改后，希望能够快速同步到各个节点上

1. 配置管理可交由zookeeper实现

可将配置信息写入zookeeper的一个ZNode上；

各个客户端监听这个ZNode；

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

1. 分布式环境中，实时掌握每个节点的状态是必要的

可根据节点的实时状态作出一些调整

1. Zookeeper可以实现实时监控节点状态变化

可将节点信息写入zookeeper的一个ZNode上

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

客户端能够实时察觉到服务器上下线的变化

|  |
| --- |
|  |

### 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

|  |
| --- |
|  |

# Zookeeper的安装

## 本地单机模式安装部署

### 安装前准备

以下是linux上面安装zookeeper，windows下安装参考dubbo使用文档

1. 安装jdk
2. 拷贝zookeeper安装包到linux系统下的soft目录下，rz -be
3. 解压到指定目录，tar -zxvf apache-zookeeper-3.5.6.tar.gz -C /data/soft

|  |
| --- |
|  |

### 修改配置文件

1. 将/data/soft/apache-zookeeper-3.5.6/conf路径下的zoo\_sample.cfg修改为zoo.cfg

|  |
| --- |
|  |

1. 打开zoo.cfg，修改dataDir路径，用于保存zk数据

dataDir=/data/soft/apache-zookeeper-3.5.6/zkData

其中zkData这个文件夹需要手动创建：mkdir zkData

### 操作zookeeper

1. 启动zookeeper：启动bin目录下的zkServer.sh bin/zkServer.sh start

|  |
| --- |
|  |

1. 查看进程是否启动jps

|  |
| --- |
|  |

1. 查看状态bin/zkServer.sh status

|  |
| --- |
|  |

1. 启动客户端bin/zkCli.sh

|  |
| --- |
|  |

1. 退出客户端quit

|  |
| --- |
|  |

1. 停止zookeeper：bin/zkServer.sh stop

|  |
| --- |
|  |

## zookeeper中zoo.cfg中配置参数的解读

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

zookeeper使用的基本时间，服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每隔tickTime就会发送一个心跳

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

1. initLimit=10：LF初始通信时限

集群中的follower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Follower之间的最大响应时间单位，假如响应时间超过syncLimit\*tickTime，Leader认为Follower挂掉，从服务器列表中删除Follower

1. dataDir：数据文件目录+数据持久化路径

主要用于保存zookeeper中的路径

1. clientPort=2181：客户端连接端口

监听客户端连接的端口

# zookeeper实战

## 分布式安装部署

集群规划：在三台机器上部署安装zookeeper

1. 在三台机器上分别解压zookeeper安装包到/data/zookeeper目录下

|  |
| --- |
|  |

1. 3台配置服务器编号

在/data/zookeeper/apache-zookeeper-3.5.6这个目录下创建zkData文件夹

|  |
| --- |
|  |

在zkData这个文件夹下面创建myid文件 vim myid

|  |
| --- |
|  |

编辑myid文件，在文件中添加与之对应的server编号，比如2，3，4

1. 修改每台服务器zoo\_sample.cfg文件改名为zoo.cfg文件

|  |
| --- |
|  |

修改dataDir的地址为zkData

|  |
| --- |
|  |

dataDir=/data/zookeeper/apache-zookeeper-3.5.6/zkData

增加如下配置

server.2=ip:2888:3888  
server.3=ip:2888:3888  
server.4=ip:2888:3888

|  |
| --- |
|  |

其中2/3/4是一个数字，表示这是第几号服务器

集群模式下配置一个文件myid，这个文件在dataDir目录下，这个里面有一个数据就是对应的2/3/4，zookeeper启动时读取此文件，拿到里面的数据与zoo.cfg里面的配置信息比较从而判断到底哪个时server

ip是这台服务器对应的ip地址

2888：是这个服务器follower与leader服务器交换信息的端口

3888：是万一集群中的leader服务器挂了，需要一个端口来重新选举，选出一个新的leader，而这个端口就是用来执行选举时服务器相互通信的端口

## 操作集群

1. 分别启动3台服务器zookeeper：bin/zkServer.sh start

|  |
| --- |
|  |

使用jps查看是否多了一个zookeeper进程

|  |
| --- |
|  |

使用bin/zkServer.sh status

|  |
| --- |
|  |

此时查看服务器状态是error状态，因为是集群状态，需要启动每台服务器的zookeeper或者是启动一半以上的服务器

|  |
| --- |
|  |

1. 当三台服务器全部启动时，查看服务器状态bin/zkServer.sh status 可以看到有一个leader和两个follower

|  |
| --- |
|  |

## 客户端命令行操作

1. 启动客户端：bin/zkCli.sh

|  |
| --- |
|  |

1. 查看当前ZNode中所包含的内容（节点）：ls /

|  |
| --- |
|  |

1. 显示客户端所有操作命令：help

|  |
| --- |
|  |

1. 查看当前节点详细信息：ls2 /

|  |
| --- |
|  |

1. 分别创建两个普通节点

注意：创建节点的同时需要向里面写数据，否则创建失败

|  |
| --- |
|  |

1. 获得节点的值

|  |
| --- |
|  |

1. 创建短暂节点

|  |
| --- |
|  |

* 1. 在当前客户端时可以看到的

|  |
| --- |
|  |

* 1. 退出当前客户端然后再重启客户端

|  |
| --- |
|  |

* 1. 再次查看根目录下短暂节点已经删除

|  |
| --- |
|  |

1. 创建带序号的节点
   1. 先创建一个普通的根节点 create /sanguo/weiguo “caocao”

|  |
| --- |
|  |

* 1. 创建带序号的节点：create -s /sanguo/weiguo/xiaoqiao “bb”

如果原来没有序号节点，序号从0开始依次递增。如果原节点下已有2个节点，则再排序时从2开始，依次类推

|  |
| --- |
|  |

1. 修改节点数据值：set /sanguo/weiguo “aa”

|  |
| --- |
|  |

1. 节点的值变化监听，启动所有服务器
   1. 在第三台机器上注册监听/snaguo节点数据变化：get /sanguo watch

|  |
| --- |
|  |

* 1. 在第一台机器上修改/sanguo节点的数据

|  |
| --- |
|  |

* 1. 观察到第三台机器收到数据变化的监听

|  |
| --- |
|  |

1. 节点的子节点变化监听（路径变化）
   1. 在第三台机器上注册监听/sanguo节点的子节点变化

|  |
| --- |
|  |

* 1. 在第一台机器/sanguo节点上创建子节点

|  |
| --- |
|  |

* 1. 观察第三台机器收到子节点变化的监听

|  |
| --- |
|  |

1. 删除节点，节点下面有子节点不允许直接删除，可以用递归删除rmr

|  |
| --- |
|  |

1. 递归删除节点

|  |
| --- |
|  |

1. 查看节点状态

|  |
| --- |
|  |

## API应用

* + 1. 创建maven工程
    2. 添加pom文件

|  |
| --- |
| <dependencies> <dependency> <groupId>junit</groupId> <artifactId>junit</artifactId> <version>RELEASE</version> </dependency> <dependency> <groupId>org.apache.logging.log4j</groupId> <artifactId>log4j-core</artifactId> <version>2.8.2</version> </dependency> <!-- https://mvnrepository.com/artifact/org.apache.zookeeper/zook eeper --> <dependency> <groupId>org.apache.zookeeper</groupId> <artifactId>zookeeper</artifactId> <version>3.4.10</version> </dependency> </dependencies> |

* + 1. 添加记录日志配置文件log4j.properties

|  |
| --- |
| 1. log4j.rootLogger=INFO, stdout log4j.appender.stdout=org.apache.log4j.ConsoleAppender log4j.appender.stdout.layout=org.apache.log4j.PatternLayout log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n log4j.appender.logfile=org.apache.log4j.FileAppender log4j.appender.logfile.File=target/spring.log log4j.appender.logfile.layout=org.apache.log4j.PatternLayout log4j.appender.logfile.layout.ConversionPattern=%d %p [%c] - %m%n |

3.4.4 创建zookeeper客户端

（1）测试zookeeper客户端连接服务器

|  |
| --- |
|  |

1. 创建子节点

|  |
| --- |
|  |

1. 获取节点

|  |
| --- |
|  |

1. 判断节点是否存在

|  |
| --- |
|  |

## 3.5 监听服务器节点动态上下线案例

（1）需求：

某分布式环境中，主节点可以有多台，可以动态上下线，任意一台客户端都能实时感知到主节点服务器的上下线