大数据技术之Zookeeper

# 第1章 Zookeeper入门

## 1.1 概述

Zookeeper是一个开源的分布式的，为分布式应用提供协调服务的Apache项目。

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，然后接受观察者的注册，一旦这些数据的状态发生了变化，Zookeeper就负责通知已经在Zookeeper上注册的那些观察者做出相应的反应.

**Zookeeper = 文件系统 + 通知机制**



## 1.2 特点



## 1.3 数据结构



## 1.4 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。**







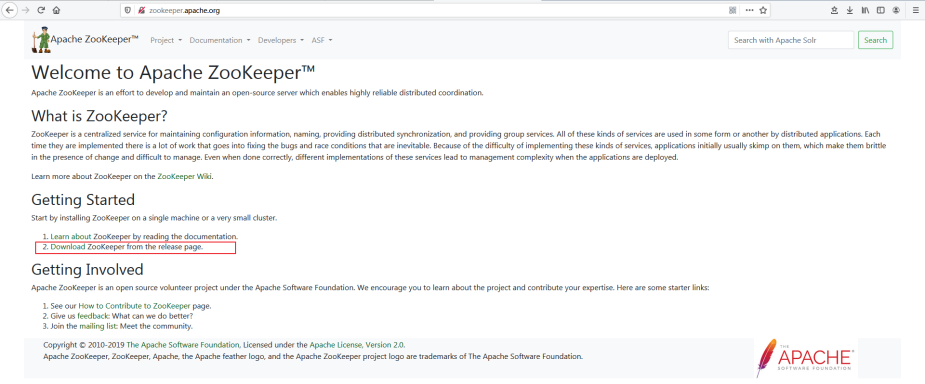


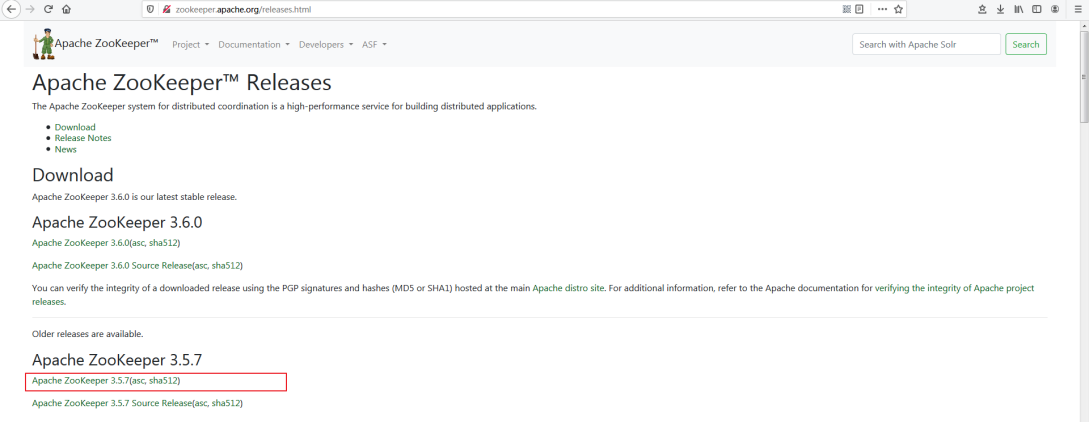
## 1.5 下载地址

**1）官网首页：**

https://zookeeper.apache.org/

**2）下载截图**





# 第2章 Zookeeper安装

## 2.1 本地模式安装部署(了解)

### 1）安装前准备

（1）安装Jdk

（2）拷贝Zookeeper安装包到Linux系统下

（3）解压到指定目录

[hadoop@hadoop102 software]$ tar -zxvf zookeeper-3.5.7.tar.gz -C /opt/module/

### 2）配置修改

（1）将/opt/module/zookeeper-3.5.7/conf这个路径下的zoo\_sample.cfg修改为zoo.cfg；

[hadoop@hadoop102 conf]$ mv zoo\_sample.cfg **zoo.cfg**

（2）打开zoo.cfg文件，修改dataDir路径：

[hadoop@hadoop102 zookeeper-3.5.7]$ vim zoo.cfg

修改如下内容：

dataDir=/opt/module/zookeeper-3.5.7/zkData

（3）在/opt/module/zookeeper-3.5.7/这个目录上创建zkData文件夹

[hadoop@hadoop102 zookeeper-3.5.7]$ mkdir zkData

### 3）操作Zookeeper

（1）启动Zookeeper

[hadoop@hadoop102 zookeeper-3.5.7]$ bin/zkServer.sh start

（2）查看进程是否启动

[hadoop@hadoop102 zookeeper-3.5.7]$ jps

4020 Jps

4001 QuorumPeerMain

（3）查看状态：

[hadoop@hadoop102 zookeeper-3.5.7]$ bin/zkServer.sh status

ZooKeeper JMX enabled by default

Using config: /opt/module/zookeeper-3.5.7/bin/../conf/zoo.cfg

Mode: standalone

（4）启动客户端：

[hadoop@hadoop102 zookeeper-3.5.7]$ bin/zkCli.sh

（5）退出客户端：

[zk: localhost:2181(CONNECTED) 0] quit

（6）停止Zookeeper

[hadoop@hadoop102 zookeeper-3.5.7]$ bin/zkServer.sh stop

## 2.2 配置参数解读

Zookeeper中的配置文件zoo.cfg中参数含义解读如下：

**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时间就会发送一个心跳，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)

**2）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。

**3）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，假如响应超过syncLimit \* tickTime，Leader认为Follwer死掉，从服务器列表中删除Follwer。

**4）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。

**5）clientPort =2181：客户端连接端口**

监听客户端连接的端口。

# 第3章 Zookeeper实战（开发重点）

## 3.1 分布式安装部署

### 1）集群规划

在hadoop102、hadoop103和hadoop104三个节点上部署Zookeeper。

### 2）解压安装

（1）解压Zookeeper安装包到/opt/module/目录下

[hadoop@hadoop102 software]$ tar -zxvf zookeeper-3.5.7.tar.gz -C /opt/module/

（2）同步/opt/module/zookeeper-3.5.7目录内容到hadoop103、hadoop104

[hadoop@hadoop102 module]$ xsync zookeeper-3.5.7/

### 3）配置服务器编号

（1）在/opt/module/zookeeper-3.5.7/这个目录下创建zkData

[hadoop@hadoop102 zookeeper-3.5.7]$ mkdir -p zkData

（2）在/opt/module/zookeeper-3.5.7/zkData目录下创建一个myid的文件

[hadoop@hadoop102 zkData]$ touch myid

添加myid文件，注意一定要在linux里面创建，在notepad++里面很可能乱码

（3）编辑myid文件

[hadoop@hadoop102 zkData]$ vi myid

在文件中添加与server对应的编号：

2

（4）拷贝配置好的zookeeper到其他机器上

[hadoop@hadoop102 zkData]$ xsync myid

并分别在hadoop103、hadoop104上修改myid文件中内容为3、4

### 4）配置zoo.cfg文件

（1）重命名/opt/module/zookeeper-3.5.7/conf这个目录下的zoo\_sample.cfg为zoo.cfg

[hadoop@hadoop102 conf]$ mv zoo\_sample.cfg zoo.cfg

（2）打开zoo.cfg文件

[hadoop@hadoop102 conf]$ vim zoo.cfg

修改数据存储路径配置

dataDir=/opt/module/zookeeper-3.5.7/zkData

增加如下配置

#######################cluster##########################

server.2=hadoop102:2888:3888

server.3=hadoop103:2888:3888

server.4=hadoop104:2888:3888

（3）同步zoo.cfg配置文件

[hadoop@hadoop102 conf]$ xsync zoo.cfg

**（4）配置参数解读**

**server.A=B:C:D。**

**A**是一个数字，表示这个是第几号服务器；

集群模式下配置一个文件myid，这个文件在dataDir目录下，这个文件里面有一个数据就是A的值，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是哪个server。

**B**是这个服务器的地址；

**C**是这个服务器Follower与集群中的Leader服务器交换信息的端口；

**D**是万一集群中的Leader服务器挂了，需要一个端口来重新进行选举，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。

**5）集群操作**

（1）分别启动Zookeeper

[hadoop@hadoop102 zookeeper-3.5.7]$ bin/zkServer.sh start

[hadoop@hadoop103 zookeeper-3.5.7]$ bin/zkServer.sh start

[hadoop@hadoop104 zookeeper-3.5.7]$ bin/zkServer.sh start

（2）查看状态

[hadoop@hadoop102 zookeeper-3.5.7]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.5.7/bin/../conf/zoo.cfg

Mode: follower

[hadoop@hadoop103 zookeeper-3.5.7]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.5.7/bin/../conf/zoo.cfg

Mode: leader

[hadoop@hadoop104 zookeeper-3.5.7]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.5.7/bin/../conf/zoo.cfg

Mode: follower

## 3.2 客户端命令行操作

|  |  |
| --- | --- |
| 命令基本语法 | 功能描述 |
| help | 显示所有操作命令 |
| ls path | 使用 ls 命令来查看当前znode的子节点  watch 监听子节点变化  ls2 附加次级信息 |
| create | 普通创建  -s 含有序列  -e 临时（重启或者超时消失） |
| get path | 获得节点的值  watch 监听节点内容变化 |
| set | 设置节点的具体值 |
| stat | 查看节点状态 |
| delete | 删除节点 |
| rmr | 递归删除节点 |

### 1）启动客户端

[hadoop@hadoop103 zookeeper-3.5.7]$ bin/zkCli.sh

### 2）显示所有操作命令

[zk: localhost:2181(CONNECTED) 1] help

### 3）查看当前znode中所包含的内容

[zk: localhost:2181(CONNECTED) 0] ls /

[zookeeper]

### 4）查看当前节点详细数据

[zk: localhost:2181(CONNECTED) 1] ls -s /

[zookeeper]

cZxid = 0x0

ctime = Thu Jan 01 08:00:00 CST 1970

mZxid = 0x0

mtime = Thu Jan 01 08:00:00 CST 1970

pZxid = 0x0

cversion = -1

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 0

numChildren = 1

### 5）分别创建2个普通节点

[zk: localhost:2181(CONNECTED) 3] create /sanguo "diaochan"

Created /sanguo

[zk: localhost:2181(CONNECTED) 4] create /sanguo/shuguo "liubei"

Created /sanguo/shuguo

### 6）获得节点的值

[zk: localhost:2181(CONNECTED) 5] get /sanguo

diaochan

[zk: localhost:2181(CONNECTED) 6] get -s /sanguo

diaochan

cZxid = 0x100000003

ctime = Wed Aug 29 00:03:23 CST 2018

mZxid = 0x100000003

mtime = Wed Aug 29 00:03:23 CST 2018

pZxid = 0x100000004

cversion = 1

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 7

numChildren = 1

[zk: localhost:2181(CONNECTED) 7]

[zk: localhost:2181(CONNECTED) 7] get -s /sanguo/shuguo

liubei

cZxid = 0x100000004

ctime = Wed Aug 29 00:04:35 CST 2018

mZxid = 0x100000004

mtime = Wed Aug 29 00:04:35 CST 2018

pZxid = 0x100000004

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 6

numChildren = 0

### 7）创建临时节点

[zk: localhost:2181(CONNECTED) 7] create -e /sanguo/wuguo "zhouyu"

Created /sanguo/wuguo

（1）在当前客户端是能查看到的

[zk: localhost:2181(CONNECTED) 3] ls /sanguo

[wuguo, shuguo]

（2）退出当前客户端然后再重启客户端

[zk: localhost:2181(CONNECTED) 12] quit

[hadoop@hadoop104 zookeeper-3.5.7]$ bin/zkCli.sh

（3）再次查看根目录下短暂节点已经删除

[zk: localhost:2181(CONNECTED) 0] ls /sanguo

[shuguo]

### 8）创建带序号的节点

（1）先创建一个普通的根节点/sanguo/weiguo

[zk: localhost:2181(CONNECTED) 1] create /sanguo/weiguo "caocao"

Created /sanguo/weiguo

（2）创建带序号的节点

[zk: localhost:2181(CONNECTED) 2] create /sanguo/weiguo "caocao"

Node already exists: /sanguo/weiguo

[zk: localhost:2181(CONNECTED) 3] create -s /sanguo/weiguo "caocao"

Created /sanguo/weiguo0000000000

[zk: localhost:2181(CONNECTED) 4] create -s /sanguo/weiguo "caocao"

Created /sanguo/weiguo0000000001

[zk: localhost:2181(CONNECTED) 5] create -s /sanguo/weiguo "caocao"

Created /sanguo/weiguo0000000002

[zk: localhost:2181(CONNECTED) 6] ls /sanguo

[shuguo, weiguo, weiguo0000000000, weiguo0000000001, weiguo0000000002, wuguo]

[zk: localhost:2181(CONNECTED) 6]

如果节点下原来没有子节点，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，以此类推。

### 9）修改节点数据值

[zk: localhost:2181(CONNECTED) 6] set /sanguo/weiguo "caopi"

### 10）节点的值变化监听

（1）在hadoop104主机上注册监听/sanguo节点数据变化

[zk: localhost:2181(CONNECTED) 26] [zk: localhost:2181(CONNECTED) 8] get -w /sanguo

（2）在hadoop103主机上修改/sanguo节点的数据

[zk: localhost:2181(CONNECTED) 1] set /sanguo "xishi"

（3）观察hadoop104主机收到数据变化的监听

WATCHER::

WatchedEvent state:SyncConnected type:NodeDataChanged path:/sanguo

### 11）节点的子节点变化监听（路径变化）

（1）在hadoop104主机上注册监听/sanguo节点的子节点变化

[zk: localhost:2181(CONNECTED) 1] ls -w /sanguo

[aa0000000001, server101]

（2）在hadoop103主机/sanguo节点上创建子节点

[zk: localhost:2181(CONNECTED) 2] create /sanguo/jin "simayi"

Created /sanguo/jin

（3）观察hadoop104主机收到子节点变化的监听

WATCHER::

WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/sanguo

### 12）删除节点

[zk: localhost:2181(CONNECTED) 4] delete /sanguo/jin

### 13）递归删除节点

[zk: localhost:2181(CONNECTED) 15] deleteall /sanguo/shuguo

### 14）查看节点状态

[zk: localhost:2181(CONNECTED) 17] stat /sanguo

cZxid = 0x100000003

ctime = Wed Aug 29 00:03:23 CST 2018

mZxid = 0x100000011

mtime = Wed Aug 29 00:21:23 CST 2018

pZxid = 0x100000014

cversion = 9

dataVersion = 1

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 4

numChildren = 1

## 3.3 API应用(了解)

### 3.3.1 IDEA环境搭建

**1）创建一个Maven Module**

**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>

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

<version>3.5.7</version>

</dependency>

</dependencies>

**3）拷贝log4j.properties文件到项目根目录**

需要在项目的src/main/resources目录下，新建一个文件，命名为“log4j.properties”，在文件中填入。

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

### 3.3.2 初始化ZooKeeper客户端

public class MyZookeeper {

private String connectString;

private int sessionTimeout;

private ZooKeeper zkClient;

@Before //获取客户端对象

public void init() throws IOException {

connectString = "hadoop102:2181,hadoop103:2181,hadoop104:2181";

int sessionTimeout = 10000;

//参数解读 1集群连接字符串 2连接超时时间 单位:毫秒 3当前客户端默认的监控器

zkClient = new ZooKeeper(connectString, sessionTimeout, new Watcher() {

@Override

public void process(WatchedEvent event) {

}

});

}

@After //关闭客户端对象

public void close() throws InterruptedException {

zkClient.close();

}

}

### 3.3.3 获取子节点列表,不监听

@Test

public void ls() throws IOException, KeeperException, InterruptedException {

//用客户端对象做各种操作

List<String> children = zkClient.getChildren("/", false);

System.out.println(children);

}

### 3.3.4 获取子节点列表,并监听

@Test

public void lsAndWatch() throws KeeperException, InterruptedException {

List<String> children = zkClient.getChildren("/hadoop", new Watcher() {

@Override

public void process(WatchedEvent event) {

System.out.println(event);

}

});

System.out.println(children);

//因为设置了监听,所以当前线程不能结束

Thread.sleep(Long.MAX\_VALUE);

}

### 3.3.5 创建子节点

@Test

public void create() throws KeeperException, InterruptedException {

//参数解读 1节点路径 2节点存储的数据

//3节点的权限(使用Ids选个OPEN即可) 4节点类型 短暂 持久 短暂带序号 持久带序号

String path = zkClient.create("/hadoop", "guiguzi".getBytes(), ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);

//创建临时节点

//String path = zkClient.create("/hadoop2", "hadoop".getBytes(), ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.EPHEMERAL);

System.out.println(path);

//创建临时节点的话,需要线程阻塞

//Thread.sleep(10000);

}

### 3.3.6 判断Znode是否存在

@Test

public void exist() throws Exception {

Stat stat = zkClient.exists("/hadoop", false);

System.out.println(stat == null ? "not exist" : "exist");

}

### 3.3.7 获取子节点存储的数据,不监听

@Test

public void get() throws KeeperException, InterruptedException {

//判断节点是否存在

Stat stat = zkClient.exists("/hadoop", false);

if (stat == null) {

System.out.println("节点不存在...");

return;

}

byte[] data = zkClient.getData("/hadoop", false, stat);

System.out.println(new String(data));

}

### 3.3.8 获取子节点存储的数据,并监听

@Test

public void getAndWatch() throws KeeperException, InterruptedException {

//判断节点是否存在

Stat stat = zkClient.exists("/hadoop", false);

if (stat == null) {

System.out.println("节点不存在...");

return;

}

byte[] data = zkClient.getData("/hadoop", new Watcher() {

@Override

public void process(WatchedEvent event) {

System.out.println(event);

}

}, stat);

System.out.println(new String(data));

//线程阻塞

Thread.sleep(Long.MAX\_VALUE);

}

### 3.3.9 设置节点的值

@Test

public void set() throws KeeperException, InterruptedException {

//判断节点是否存在

Stat stat = zkClient.exists("/hadoop", false);

if (stat == null) {

System.out.println("节点不存在...");

return;

}

//参数解读 1节点路径 2节点的值 3版本号

zkClient.setData("/hadoop", "mlf".getBytes(), stat.getVersion());

}

### 3.3.10 删除空节点

@Test

public void delete() throws KeeperException, InterruptedException {

//判断节点是否存在

Stat stat = zkClient.exists("/aaa", false);

if (stat == null) {

System.out.println("节点不存在...");

return;

}

zkClient.delete("/aaa", stat.getVersion());

}

### 3.3.11 删除非空节点,递归实现

//封装一个方法,方便递归调用

public void deleteAll(String path, ZooKeeper zk) throws KeeperException, InterruptedException {

//判断节点是否存在

Stat stat = zkClient.exists(path, false);

if (stat == null) {

System.out.println("节点不存在...");

return;

}

//先获取当前传入节点下的所有子节点

List<String> children = zk.getChildren(path, false);

if (children.isEmpty()) {

//说明传入的节点没有子节点,可以直接删除

zk.delete(path, stat.getVersion());

} else {

//如果传入的节点有子节点,循环所有子节点

for (String child : children) {

//删除子节点,但是不知道子节点下面还有没有子节点,所以递归调用

deleteAll(path + "/" + child, zk);

}

//删除完所有子节点以后,记得删除传入的节点

zk.delete(path, stat.getVersion());

}

}

//测试deleteAll

@Test

public void testDeleteAll() throws KeeperException, InterruptedException {

deleteAll("/hadoop",zkClient);

}

# 第4章 Zookeeper内部原理

## 4.1 节点类型



## 4.2 Stat结构体

（1）czxid-创建节点的事务zxid

每次修改ZooKeeper状态都会收到一个zxid形式的时间戳，也就是ZooKeeper事务ID。

事务ID是ZooKeeper中所有修改总的次序。每个修改都有唯一的zxid，如果zxid1小于zxid2，那么zxid1在zxid2之前发生。

（2）ctime - znode被创建的毫秒数(从1970年开始)

（3）mzxid - znode最后更新的事务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

## 4.3 监听器原理（面试重点）



## 4.4 选举机制（面试重点）

**（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指定Master和Slave。但是，Zookeeper工作时，是有一个节点为Leader，其他则为Follower，Leader是通过内部的选举机制临时产生的。

（3）以一个简单的例子来说明整个选举的过程。

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。假设这些服务器依序启动，来看看会发生什么。



**Zookeeper的选举机制**

（1）服务器1启动，发起一次选举。服务器1投自己一票。此时服务器1票数一票，不够半数以上（3票），选举无法完成，服务器1状态保持为LOOKING；

（2）服务器2启动，再发起一次选举。服务器1和2分别投自己一票并交换选票信息：此时服务器1发现服务器2的ID比自己目前投票推举的（服务器1）大，更改选票为推举服务器2。此时服务器1票数0票，服务器2票数2票，没有半数以上结果，选举无法完成，服务器1，2状态保持LOOKING

（3）服务器3启动，发起一次选举。此时服务器1和2都会更改选票为服务器3。此次投票结果：服务器1为0票，服务器2为0票，服务器3为3票。此时服务器3的票数已经超过半数，服务器3当选Leader。服务器1，2更改状态为FOLLOWING，服务器3更改状态为LEADING；

（4）服务器4启动，发起一次选举。此时服务器1，2，3已经不是LOOKING状态，不会更改选票信息。交换选票信息结果：服务器3为3票，服务器4为1票。此时服务器4服从多数，更改选票信息为服务器3，并更改状态为FOLLOWING；

（5）服务器5启动，同4一样当小弟。

## 4.5 写数据流程



# 第5章 企业面试真题

## 5.1 请简述ZooKeeper的选举机制

**1. 选举机制总原则**：集群中的每台机器都参与投票，通过交换选票得到每台机器的最终得票， 一旦出现得票数超过机器总数一半以上数量，当前机器即为leader。

**2.举例说明1**

场景：以5台机器为例，集群的机器顺时启动，当前集群中没有任何数据。

①. server1 启动，首先server1给自己投一票，然后看当前票数是否超过半数，结果没有超过，这时候leader就没选出来，当前选举状态是Locking状态。

②. server2 启动，首先server2先给自己投一票，因为当前集群已经有两台机器已启动，所以server1server2会交换选票，交换后发现各自有一票，接下来比较myid 发现server2的myid值 > server2的myid值此时server2胜出，最后server2有两票。最后再看当前票数是否半，发现未过半，集群的选举状态集训保持locking状态。

③. server3启动， 首先自己投自己一票，server1和server2也会投自己一票，然后交换选票发现都一样，接着比较myid 最后server3胜出，此时server3就有3票，同时server3的票数超过半数。所以server3成为 leader。

④. server4启动，发现当前集群已经有leader 它自己自动成为follower

⑤. server5启动，发现当前集群已经有leader 它自己自动成为follower

**3. 举例说明2**

场景：以5台机器为例，当前集群正在使用（有数据/没数据），leader突然宕机的情况。

当集群中的leader挂掉，集群会重新选出一个leader，此时首先会比较每一台机器的czxid, czxid最大的被选为leader。极端情况，czxid都相等的情况，那么就会直接比较myid。

## 5.2 ZooKeeper的监听原理是什么？

详见4.3。

## 5.3 ZooKeeper的部署方式有哪几种？集群中的角色有哪些？集群最少需要几台机器？

（1）部署方式单机模式、集群模式

（2）角色：Leader和Follower

（3）集群最少需要机器数：3

## 5.4 ZooKeeper的常用命令

ls create get delete set…