大数据技术之Zookeeper

# **一 Zookeeper概述**

## **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。Hadoop和Hbase的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

## **1.2 特点**

1）Zookeeper：一个领导者（leader），多个跟随者（follower）组成的集群。

2）Leader负责进行投票的发起和决议，更新系统状态。

3）Follower用于接收客户请求并向客户端返回结果，在选举Leader过程中参与投票。

4）集群中奇数台服务器只要有半数以上节点存活，Zookeeper集群就能正常服务。

5）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。

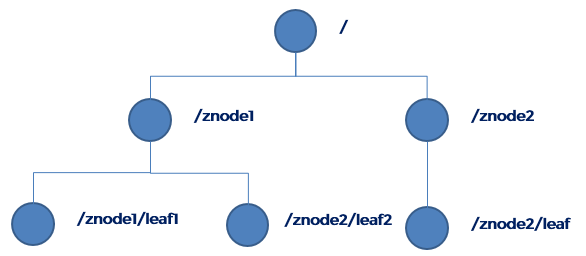
6）更新请求顺序进行，来自同一个client的更新请求按其发送顺序依次执行。

7）数据更新原子性，一次数据更新要么成功，要么失败。

8）实时性，在一定时间范围内，client能读到最新数据。

## **1.3 数据结构**

ZooKeeper数据模型的结构与Unix文件系统很类似，整体上可以看作是一棵树，每个节点称做一个ZNode。每一个ZNode默认能够存储1MB的元数据，每个ZNode都可以通过其路径唯一标识。

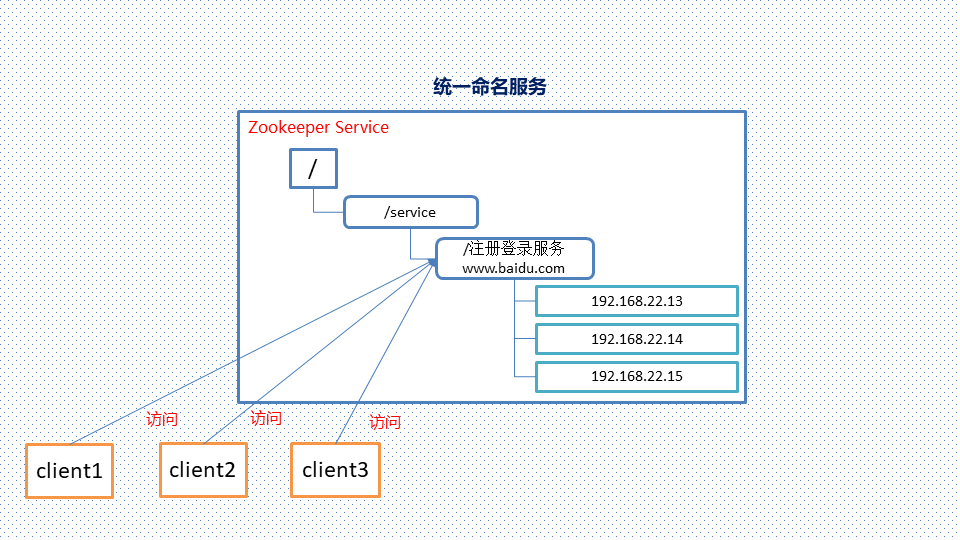


数据结构图

## **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。

### **1.4.1 统一命名服务**



### **1.4.2 统一配置管理**

1、分布式环境下，配置文件管理和同步是一个常见问题

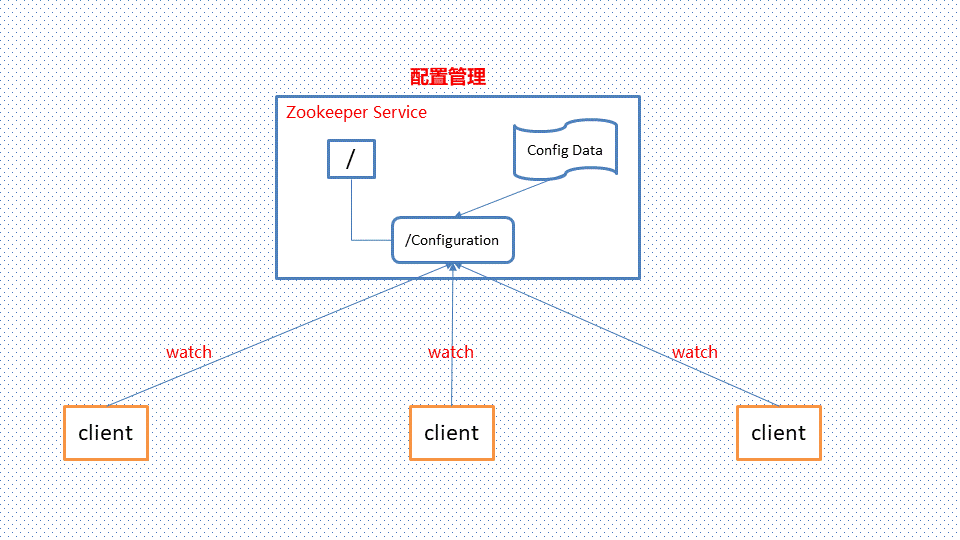
1. 一个集群中，所有节点的配置信息是一致的，比如hadoop集群
2. 对配置文件修改后，希望能够快速同步到各个节点上

2、配置管理可交由ZK实现

（1）可配置信息写入ZK上的一个Znode

（2）各个节点监听这个ZNode

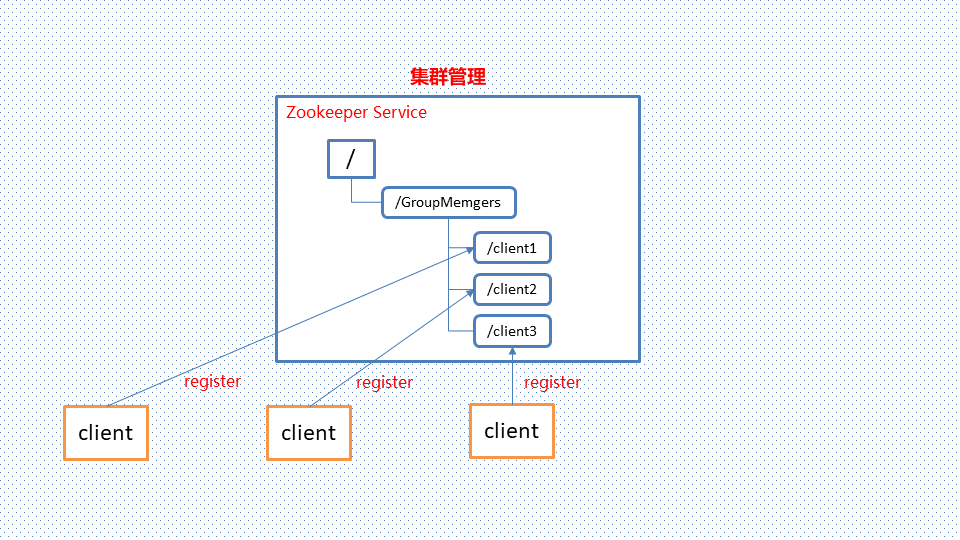
（3）一旦Znode中的数据被修改，ZK将通知各个节点



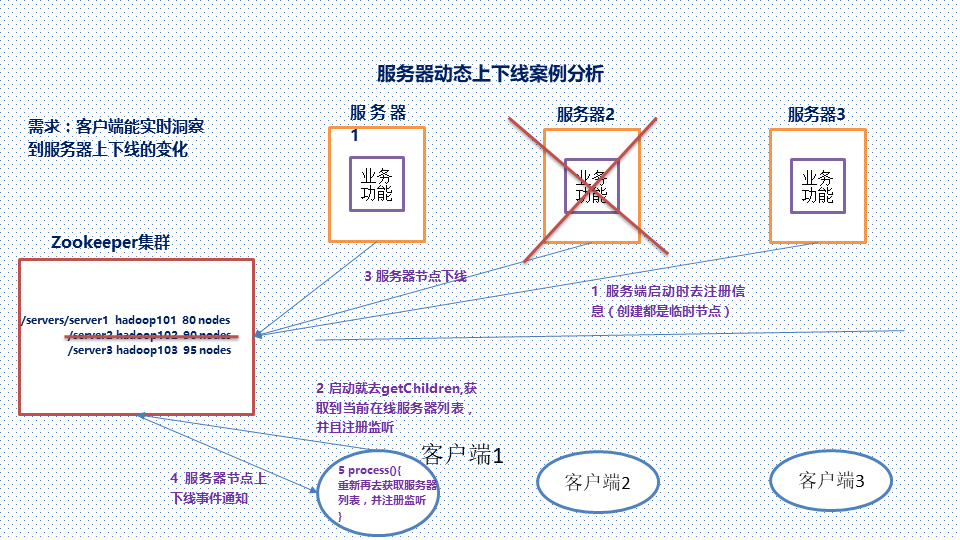
### **1.4.3 统一集群管理**

集群管理结构图如下所示。

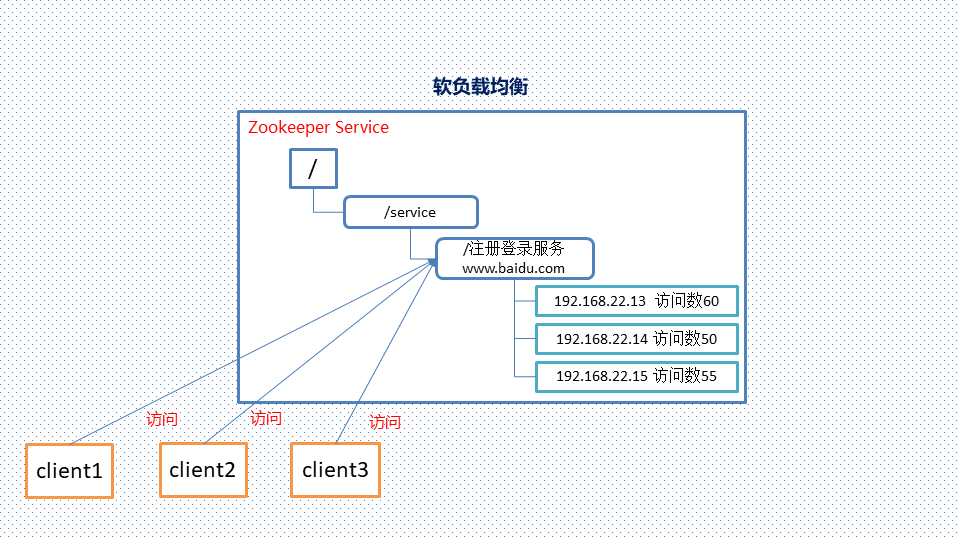
1. 分布式环境中，实时掌握每个节点的状态是必要的。
2. 可根据节点实时做出一些调整
3. 可交由Zk实现
4. 可将节点信息写入ZK上的一个ZNode
5. 监听这个Znode可获取它的实时状态变化
6. 典型应用
7. HBase中Master状态监控与选举



### **1.4.4 服务器节点动态上下线**



### **1.4.5 软负载均衡**

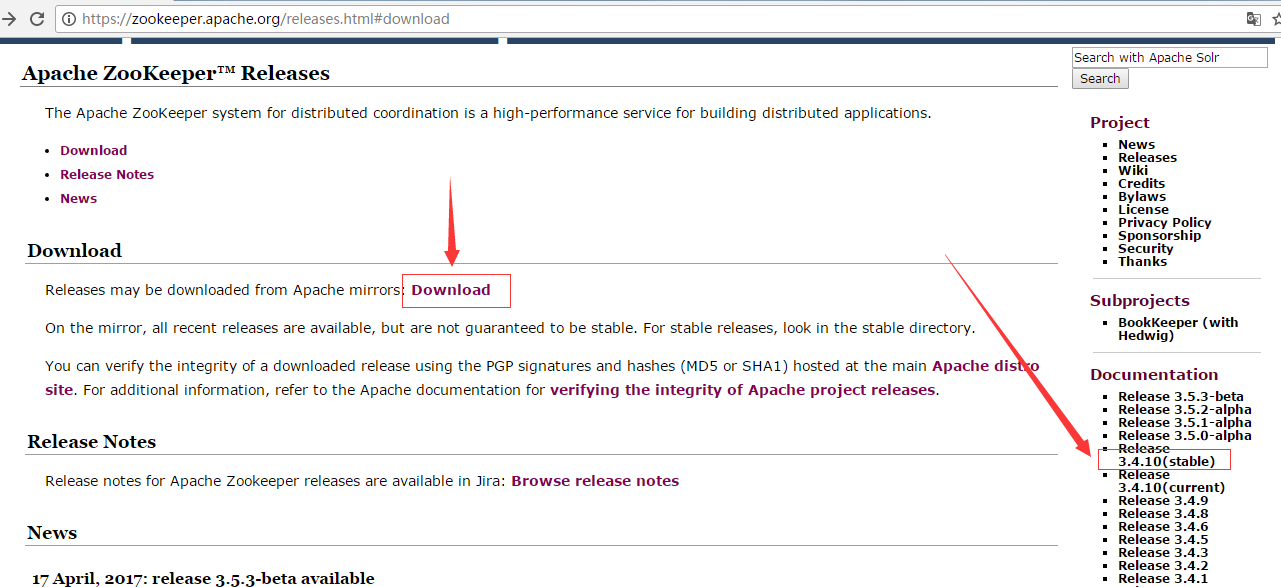


## **1.5 下载地址**

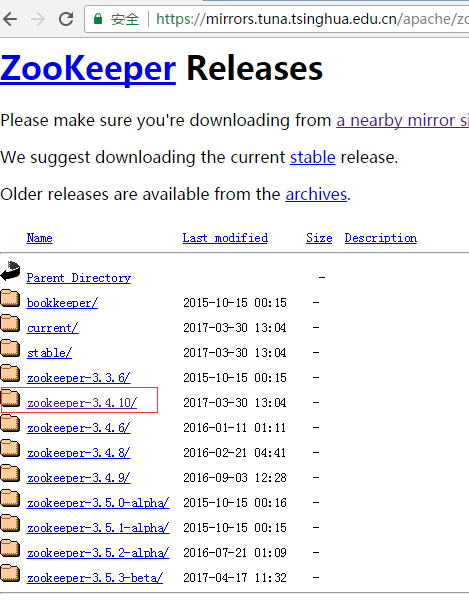
1）官网首页：

https://zookeeper.apache.org/

2）下载截图







# **二 Zookeeper安装**

## **2.1 本地模式安装部署**

1）安装前准备：

（1）安装jdk

（2）上传zookeeper到linux系统下

（3）修改tar包权限

[itstar@bigdata111 software]$ chmod u+x zookeeper-3.4.10.tar.gz

（4）解压到指定目录

[itstar@bigdata111 software]$ tar -zxvf zookeeper-3.4.10.tar.gz -C /opt/module/

（5）配置环境变量

[root@bigdata111 software]$ vi /etc/profile

export ZOOKEEPER\_HOME=/opt/module/zookeeper-3.4.10

export PATH=$PATH:$ZOOKEEPER\_HOME/bin

2）配置修改

将/opt/module/zookeeper-3.4.10/conf这个路径下的zoo\_sample.cfg修改为zoo.cfg；

[itstar@bigdata111 zookeeper-3.4.10]$mv zoo\_sample.cfg zoo.cfg

进入zoo.cfg文件：vim zoo.cfg

修改dataDir路径为

dataDir=/opt/module/zookeeper-3.4.10/zkData

在/opt/module/zookeeper-3.4.10/这个目录上创建zkData文件夹

[itstar@bigdata111 zookeeper-3.4.10]$ mkdir zkData

3）操作zookeeper

（1）启动zookeeper

[itstar@bigdata111 zookeeper-3.4.10]$ bin/zkServer.sh start

（2）查看进程是否启动

[itstar@bigdata111 zookeeper-3.4.10]$ jps

4020 Jps

4001 QuorumPeerMain

（3）查看状态：

[itstar@bigdata111 zookeeper-3.4.10]$ bin/zkServer.sh status

ZooKeeper JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: standalone

（4）启动客户端：

[itstar@bigdata111 zookeeper-3.4.10]$ bin/zkCli.sh

（5）退出客户端：

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

（6）停止zookeeper

[itstar@bigdata111 zookeeper-3.4.10]$ bin/zkServer.sh stop

## **2.2 配置参数解读**

解读zoo.cfg文件中参数含义

1）tickTime=2000：通信心跳数，Zookeeper服务器心跳时间，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：Leader和Follower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。

投票选举新leader的初始化时间

Follower在启动过程中，会从Leader同步所有最新数据，然后确定自己能够对外服务的起始状态。

Leader允许Follower在initLimit时间内完成这个工作。

3）syncLimit=5：Leader和Follower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。

在运行过程中，Leader负责与ZK集群中所有机器进行通信，例如通过一些心跳检测机制，来检测机器的存活状态。

如果L发出心跳包在syncLimit之后，还没有从F那收到响应，那么就认为这个F已经不在线了。

4）dataDir：数据文件目录+数据持久化路径

保存内存数据库快照信息的位置，如果没有其他说明，更新的事务日志也保存到数据库。

5）clientPort=2181：客户端连接端口

监听客户端连接的端口

# **三 Zookeeper内部原理**

## **3.1 选举机制**

**Server ID： myid（权重越大）**

**Zxid：数据ID（先一数据低进行选择）**

1）半数机制（Paxos 协议）：集群中半数以上机器存活，集群可用。所以zookeeper适合装在奇数台机器上。

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。

（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服务器都同意选举它(这个例子中的半数以上是3)，所以服务器1、2还是继续保持LOOKING状态。

（3）服务器3启动，根据前面的理论分析，服务器3成为服务器1、2、3中的老大，而与上面不同的是，此时有三台服务器选举了它，所以它成为了这次选举的leader。

（4）服务器4启动，根据前面的分析，理论上服务器4应该是服务器1、2、3、4中最大的，但是由于前面已经有半数以上的服务器选举了服务器3，所以它只能接收当小弟的命了。

（5）服务器5启动，同4一样当小弟。

## **3.2 节点类型**

1）Znode有两种类型：

短暂（ephemeral）：客户端和服务器端断开连接后，创建的节点自己删除

持久（persistent）：客户端和服务器端断开连接后，创建的节点不删除

2）Znode有四种形式的目录节点（默认是persistent ）

（1）持久化目录节点（PERSISTENT）(小写：persistent)

客户端与zookeeper断开连接后，该节点依旧存在。

（2）持久化顺序编号目录节点（PERSISTENT\_SEQUENTIAL）

（小写：persistent\_sequential）

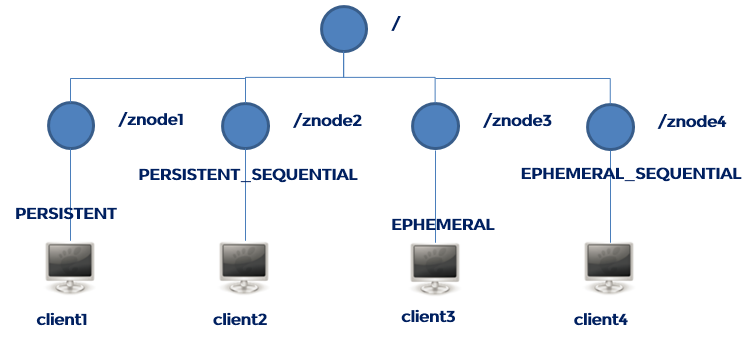
客户端与zookeeper断开连接后，该节点依旧存在，只是Zookeeper给该节点名称进行顺序编号。

（3）临时目录节点（EPHEMERAL）(ephemeral)

客户端与zookeeper断开连接后，该节点被删除。

（4）临时顺序编号目录节点（EPHEMERAL\_SEQUENTIAL）(ephemeral\_sequential)

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。



3）创建znode时设置顺序标识，znode名称后会附加一个值，顺序号是一个单调递增的计数器，由父节点维护

4）在分布式系统中，顺序号可以被用于为所有的事件进行全局排序，这样客户端可以通过顺序号推断事件的顺序

## **3.3 stat结构体**

1）czxid- 引起这个znode创建的zxid，创建节点的事务的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

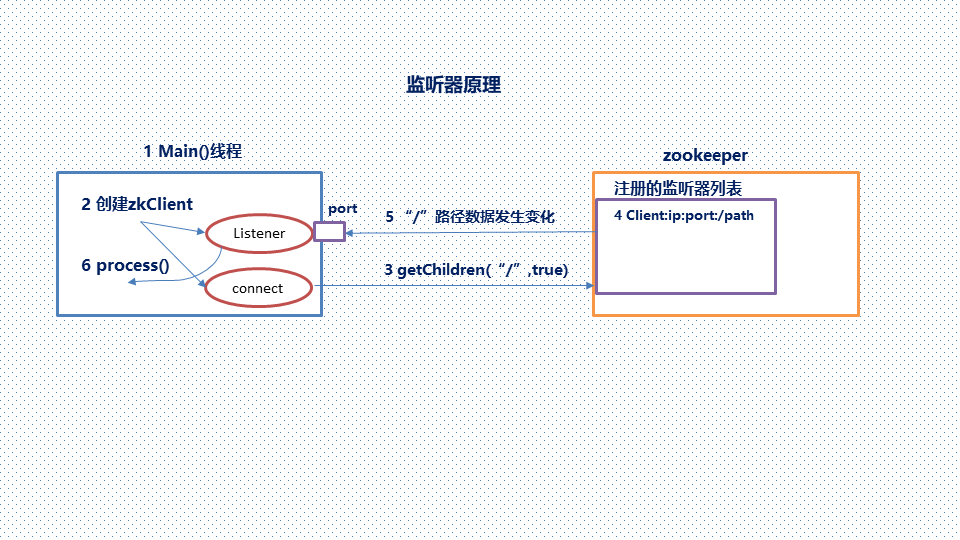
9）ephemeralOwner- 如果是临时节点，这个是znode拥有者的session id。如果不是临时节点则是0。

10）dataLength- znode的数据长度

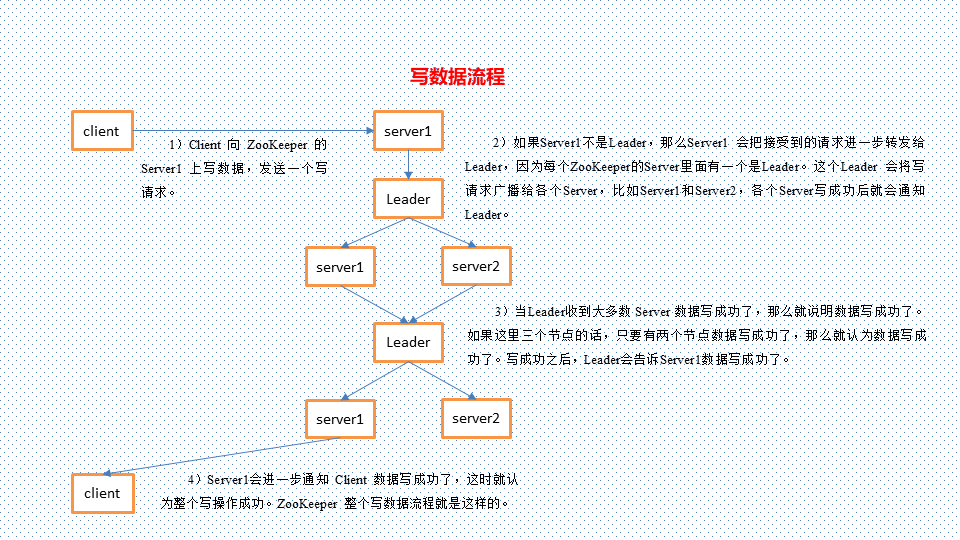
11）numChildren - znode子节点数量

## **0.4 监听器原理**

1. 监听原理详解：
   1. 首先要有一个main（）线程
   2. 在main线程中创建ZK客户端，这是会创建两个线程，一个负责网络连接通信（connect）,一个负责监听（listener）
   3. 通过connect线程将注册的监听事件发送给ZK
   4. 在ZK的注册监听器列表中将注册的监听事件添加到列表中
   5. ZK监听到有数据或路径发生变化时，就会将这个消息发送给listener线程
   6. Listener线程内部调用process()方法
2. 常见的监听
   1. 监听节点数据的变化
      1. Get path [watch]
   2. 监听子节点增减的变化
      1. Ls path [watch]



## **3.5 写数据流程**



读是局部性的，即client只需要从与它相连的server上读取数据即可；而client有写请求的话，与之相连的server会通知leader，然后leader会把写操作分发给所有server。所以定要比读慢很多。

# **四 Zookeeper实战**

## **4.1 分布式安装部署**

0）集群规划

在bigdata111、bigdata112和bigdata113三个节点上部署Zookeeper。

1）解压安装

（1）解压zookeeper安装包到/opt/module/目录下

[itstar@bigdata111 software]$ tar -zxvf zookeeper-3.4.10.tar.gz -C /opt/module/

（2）在/opt/module/zookeeper-3.4.10/这个目录下创建zkData

mkdir -p zkData

（3）重命名/opt/module/zookeeper-3.4.10/conf这个目录下的zoo\_sample.cfg为zoo.cfg

mv zoo\_sample.cfg zoo.cfg

2）配置zoo.cfg文件

（1）具体配置

dataDir=/opt/module/zookeeper-3.4.10/zkData

增加如下配置

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

server.1=bigdata111:2888:3888

server.2=bigdata112:2888:3888

server.3=bigdata113:2888:3888

（2）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；

B是这个服务器的ip地址；

C是这个服务器与集群中的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。

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。

3）集群操作

（1）在/opt/module/zookeeper-3.4.10/zkData目录下创建一个myid的文件

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

（2）编辑myid文件

vi myid

在文件中添加与server对应的编号：如2

（3）拷贝配置好的zookeeper到其他机器上

scp -r zookeeper-3.4.10/ [root@bigdata112.itstar.com:/opt/app/](oop103.)

scp -r zookeeper-3.4.10/ [root@bigdata113.itstar.com:/opt/app/](mailto:root@hadoop104.atguigu.com:/opt/app/)

并分别修改myid文件中内容为3、4

（4）分别启动zookeeper

[root@bigdata111 zookeeper-3.4.10]# bin/zkServer.sh start

[root@bigdata112 zookeeper-3.4.10]# bin/zkServer.sh start

[root@bigdata113 zookeeper-3.4.10]# bin/zkServer.sh start

（5）查看状态

[root@bigdata111 zookeeper-3.4.10]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: follower

[root@bigdata112 zookeeper-3.4.10]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: leader

[root@bigdata113 zookeeper-3.4.5]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: follower

## **4.2 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 [watch] | 使用 ls 命令来查看当前znode中所包含的内容 |
| ls2 path [watch] | 查看当前节点数据并能看到更新次数等数据 |
| create | 普通创建(永久节点)  -s 含有序列  -e 临时（重启或者超时消失） |
| get path [watch] | 获得节点的值 |
| 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

[itstar@bigdata112 zookeeper-3.4.10]$ 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）查看当前节点数据并能看到更新次数等数据

[zk: localhost:2181(CONNECTED) 1] ls2 /

[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）创建普通节点

[zk: localhost:2181(CONNECTED) 2] create /app1 "hello app1"

Created /app1

[zk: localhost:2181(CONNECTED) 4] create /app1/server101 "192.168.1.101"

Created /app1/server101

6）获得节点的值

[zk: localhost:2181(CONNECTED) 6] get /app1

hello app1

cZxid = 0x20000000a

ctime = Mon Jul 17 16:08:35 CST 2017

mZxid = 0x20000000a

mtime = Mon Jul 17 16:08:35 CST 2017

pZxid = 0x20000000b

cversion = 1

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 10

numChildren = 1

[zk: localhost:2181(CONNECTED) 8] get /app1/server101

192.168.1.101

cZxid = 0x20000000b

ctime = Mon Jul 17 16:11:04 CST 2017

mZxid = 0x20000000b

mtime = Mon Jul 17 16:11:04 CST 2017

pZxid = 0x20000000b

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 13

numChildren = 0

7）创建短暂节点

[zk: localhost:2181(CONNECTED) 9] create -e /app-emphemeral 8888

（1）在当前客户端是能查看到的

[zk: localhost:2181(CONNECTED) 10] ls /

[app1, app-emphemeral, zookeeper]

（2）退出当前客户端然后再重启客户端

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

[itstar@bigdata113 zookeeper-3.4.10]$ 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 /

[app1, zookeeper]

8）创建带序号的节点

（1）先创建一个普通的根节点app2

[zk: localhost:2181(CONNECTED) 11] create /app2 "app2"

（2）创建带序号的节点

[zk: localhost:2181(CONNECTED) 13] create -s /app2/aa 888

Created /app2/aa0000000000

[zk: localhost:2181(CONNECTED) 14] create -s /app2/bb 888

Created /app2/bb0000000001

[zk: localhost:2181(CONNECTED) 15] create -s /app2/cc 888

Created /app2/cc0000000002

如果原节点下有1个节点，则再排序时从1开始，以此类推。

[zk: localhost:2181(CONNECTED) 16] create -s /app1/aa 888

Created /app1/aa0000000001

9）修改节点数据值

[zk: localhost:2181(CONNECTED) 2] set /app1 999

10）节点的值变化监听

（1）在104主机上注册监听/app1节点数据变化

[zk: localhost:2181(CONNECTED) 26] get /app1 watch

（2）在103主机上修改/app1节点的数据

[zk: localhost:2181(CONNECTED) 5] set /app1 777

（3）观察104主机收到数据变化的监听

WATCHER::

WatchedEvent state:SyncConnected type:NodeDataChanged path:/app1

11）节点的子节点变化监听（路径变化）

（1）在104主机上注册监听/app1节点的子节点变化

[zk: localhost:2181(CONNECTED) 1] ls /app1 watch

[aa0000000001, server101]

（2）在103主机/app1节点上创建子节点

[zk: localhost:2181(CONNECTED) 6] create /app1/bb 666

Created /app1/bb

（3）观察104主机收到子节点变化的监听

WATCHER::

WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/app1

12）删除节点

[zk: localhost:2181(CONNECTED) 4] delete /app1/bb

13）递归删除节点

[zk: localhost:2181(CONNECTED) 7] rmr /app2

14）查看节点状态

[zk: localhost:2181(CONNECTED) 12] stat /app1

cZxid = 0x20000000a

ctime = Mon Jul 17 16:08:35 CST 2017

mZxid = 0x200000018

mtime = Mon Jul 17 16:54:38 CST 2017

pZxid = 0x20000001c

cversion = 4

dataVersion = 2

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 3

numChildren = 2

## **4.3 API应用**

### **4.3.1 IDEA环境搭建**

1）创建一个工程

Pom.xml

|  |
| --- |
| <dependencies>  <dependency>  <groupId>org.apache.zookeeper</groupId>  <artifactId>zookeeper</artifactId>  <version>3.4.10</version>  </dependency>  </dependencies> |

### **4.3.2 创建ZooKeeper客户端**

|  |
| --- |
| **private** **static** String *connectString* = "bigdata111:2181,bigdata112:2181,bigdata113:2181";  **private** **static** **int** *sessionTimeout* = 2000;  **private** ZooKeeper zkClient = **null**;  @Before  **public** **void** init() **throws** Exception {  zkClient = **new** ZooKeeper(*connectString*, *sessionTimeout*, **new** Watcher() {  @Override  **public** **void** process(WatchedEvent event) {  // 收到事件通知后的回调函数（用户的业务逻辑）  System.***out***.println(event.getType() + "--" + event.getPath());  // 再次启动监听  **try** {  zkClient.getChildren("/", **true**);  } **catch** (Exception e) {  e.printStackTrace();  }  }  });  } |

### **4.3.3 创建子节点**

|  |
| --- |
| // 创建子节点  @Test  **public** **void** create() **throws** Exception {  // 数据的增删改查  // 参数1：要创建的节点的路径； 参数2：节点数据 ； 参数3：节点权限 ；参数4：节点的类型  String nodeCreated = zkClient.create("/eclipse", "hello zk".getBytes(), Ids.***OPEN\_ACL\_UNSAFE***,CreateMode.***PERSISTENT***);  } |

### **4.3.4 获取子节点并监听**

|  |
| --- |
| // 获取子节点  @Test  **public** **void** getChildren() **throws** Exception {  List<String> children = zkClient.getChildren("/", **true**);  **for** (String child : children) {  System.***out***.println(child);  }  // 延时阻塞  Thread.*sleep*(Long.***MAX\_VALUE***);  } |

### **4.3.5 判断znode是否存在**

|  |
| --- |
| // 判断znode是否存在  @Test  **public** **void** exist() **throws** Exception {  Stat stat = zkClient.exists("/eclipse", **false**);  System.***out***.println(stat == **null** ? "not exist" : "exist");  } |

## **4.4 案例实战 分布式秒杀**

Zk所需要的Pom依赖

|  |
| --- |
| <!-- zookeeper所需依赖 -->  <dependency>  <groupId>org.apache.curator</groupId>  <artifactId>curator-framework</artifactId>  <version>4.0.0</version>  </dependency>  <dependency>  <groupId>org.apache.curator</groupId>  <artifactId>curator-recipes</artifactId>  <version>4.0.0</version>  </dependency>  <dependency>  <groupId>org.apache.curator</groupId>  <artifactId>curator-client</artifactId>  <version>4.0.0</version>  </dependency>  <dependency>  <groupId>com.google.guava</groupId>  <artifactId>guava</artifactId>  <version>16.0.1</version>  </dependency> |

TestDistributedLock

|  |
| --- |
| import org.apache.curator.RetryPolicy;  import org.apache.curator.framework.CuratorFramework;  import org.apache.curator.framework.CuratorFrameworkFactory;  import org.apache.curator.framework.recipes.locks.InterProcessMutex;  import org.apache.curator.retry.ExponentialBackoffRetry;  public class TestDistributedLock {  //定义共享资源  private static int count = 10;  private static void printCountNumber() {  System.out.println("\*\*\*\*\*\*\*\*\*\*\*" + Thread.currentThread().getName() + "\*\*\*\*\*\*\*\*\*\*");  System.out.println("当前值：" + count);  count--;  //睡2秒  try {  Thread.sleep(500);  } catch (InterruptedException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  System.out.println("\*\*\*\*\*\*\*\*\*\*\*" + Thread.currentThread().getName() + "\*\*\*\*\*\*\*\*\*\*");  }  public static void main(String[] args) {  //定义客户端重试的策略  RetryPolicy policy = new ExponentialBackoffRetry(1000, //每次等待的时间  10); //最大重试的次数  //定义ZK的一个客户端  CuratorFramework client = CuratorFrameworkFactory.builder()  .connectString("bigdata111:2181")  .retryPolicy(policy)  .build();  //在ZK生成锁 ---> 就是ZK的目录  client.start();  final InterProcessMutex lock = new InterProcessMutex(client, "/mylock");  // 启动10个线程去访问共享资源  for (int i = 0; i < 10; i++) {  new Thread(new Runnable() {  public void run() {  try {  //请求得到锁  lock.acquire();  //访问共享资源  printCountNumber();  } catch (Exception ex) {  ex.printStackTrace();  } finally {  //释放锁  try {  lock.release();  } catch (Exception e) {  e.printStackTrace();  }  }  }  }).start();  }  }  } |