# Zookeeper

# 什么是Zookeeper

Apache ZooKeeper是由集群（节点组）使用的一种服务，用于在自身之间协调，并通过稳健的同步技术维护共享数据。ZooKeeper本身是一个分布式应用程序，为写入分布式应用程序提供服务，有效的降低分布式系统开发过程中节点间数据协调问题开发难度。

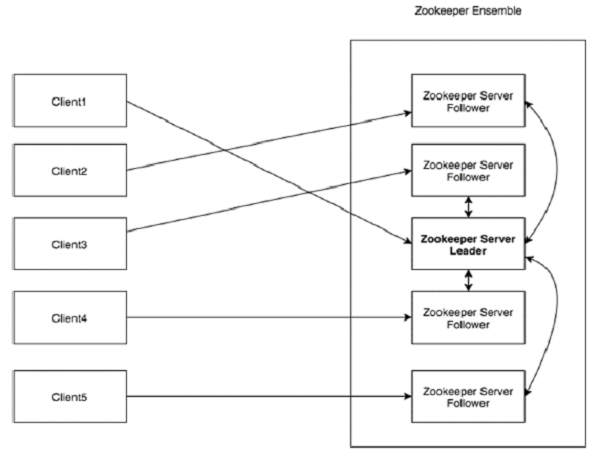
Zookeeper提供的服务功能有：

1. 数据注册功能；
2. 数据查询功能；
3. 数据监听功能；

Zookeeper能用来做什么：

* 1. 分布式系统中主从协调
  2. 分布式系统中配置信息同步
  3. 分布式系统中分布式共享锁（Master选举）
  4. 分布式系统中负载均衡
  5. 分布式系统中明名称服务

# Zookeeper基础



作为ZooKeeper架构每个组件:

|  |  |
| --- | --- |
| **部分** | **描述** |
| Client | 请求发起方。 |
| Server | 服务器，我们的ZooKeeper总体中的一个节点，为客户端提供所有的服务。 |
| Leader | 服务器中某个节点，在服务启动时被选举。主要负责处理数据和更新等操作 |
| Follower | 执行leader指令及选举leader的服务器节点。 |

## 2.1 Zookeeper的数据组织形式：

Zookeeper的数据以key-value形式存储，但是key类型比较特殊，类似路径的形式”/keyname /aa/bb”, 我们称之为znode, value值可以是任意的二进制数据byte[].

## 2.2 Znode的类型

Znode被分为持久（persistent）节点，顺序（sequential）节点和临时（ephemeral）节点。

**永久节点**: 一经创建，除非客户端主动删除，否则一直存储。

**顺序节点**：客户端在创建这样的节点时，服务端会在该节点后拼接上一个序号，类似/aa0000000001，/aa0000000002...

**临时节点**：创建这类节点的客户端必须和zookeeper服务端保持心跳(不能断开)，否则服务器端会自动将该客户端创建的临时节点删除。

其中还可以将这三种节点进行组合：

永久节点+顺序节点；

临时节点+顺序节点；

## 2.3 Leader选举

Zookeeper集群安装节点数量最好是奇数台(3,5,7......)，因为zookeeper集中中没有固定的leader， leader是动态选举出来的，而选举的数量必须要超过集群的半数以上。

选举规则：谁的票数多，谁当选leader.

leader选举过程(假设3台节点, 每台节点都配置了id)：

* 初次启动的投票过程：

1. 集群中N1节点启动，它会从网络中寻找leader, 没有leader，则开始在网络中广播投票(第一次投自己);
2. 集群中N2节点启动，它会从网络中寻找leader, 没有leader，但是收到N1的投票，N2也开始在网络中广播投票，由于N2节点id>N1节点id，所以N2投自己一票， N1收到N2的投票后，得知N2的id，则N1投N2一票;
3. 此时，N2的2票，票数过半，则当选为leader, N2将自己的状态切换为leader, N1则将自己的投票状态切换为follower;
4. 集群中N3节点启动，他会从网络中寻找leader, 找到N2为leader，则N3直接进入follower状态。

* 运行过程中leader宕机了，则重新进行leader选举

投票规则：谁的数据版本最新则选举谁为leader, 如果数据版本一致，则谁的id选举谁为leader.

# Zookeeper安装

## 3.1下载ZooKeeper

访问： http://zookeeper.apache.org/releases.html，下载最新版本的ZooKeeper。

解压：

|  |
| --- |
| $ tar -zxf zookeeper-3.4.6.tar.gz  $ cd zookeeper-3.4.6  $ mkdir data |

## 3.2 修改配置

进入到解压好的zookeeper-3.4.6/conf目录，重命名zoo\_sample.cfg为zoo.cfg

|  |
| --- |
| $ mv zoo\_sample.cfg zoo.cfg  $ vi zoo.cfg  *tickTime = 2000*  *dataDir = /zookeeper-3.4.6/data*  *clientPort = 2181*  *initLimit = 5*  *syncLimit = 2*  *server.1=hdp-nd-01:2888:3888*  *server.2=hdp-nd-02:2888:3888*  *server.3=hdp-nd-03:2888:3888* |

将zookeeper-3.4.6拷贝到其他2台服务器节点：

|  |
| --- |
| $ scp -r zookeeper-3.4.6 hdp-nd-02:$PWD  $ scp -r zookeeper-3.4.6 hdp-nd-03:$PWD |

分别在三台机器上执行如下命令：

hdp-nd-01

|  |
| --- |
| $ echo 1 > zookeeper-3.4.6/data/myid |

hdp-nd-02

|  |
| --- |
| $ echo 2 > zookeeper-3.4.6/data/myid |

hdp-nd-03

|  |
| --- |
| $ echo 3 > zookeeper-3.4.6/data/myid |

## 3.3 启动ZooKeeper服务器

执行以下命令

$ bin/zkServer.sh start conf/zoo.cfg

执行此命令后，你将收到以下响应

$ JMX enabled by default

$ Using config: /Users/../zookeeper-3.4.6/bin/../conf/zoo.cfg

$ Starting zookeeper ... STARTED

$ bin/zkServer.sh status #查看节点状态

## 3.4 停止ZooKeeper服务器

连接服务器并执行所有操作后，可以使用以下命令停止zookeeper服务器。

$ bin/zkServer.sh stop

# 命令行客户端

Ls 查看节点子节点

Create 创建节点

Set 修改节点数据

Watch 监听节点事件

Rmr 删除节点

监听分为两种监听：znode = key value

1. 监听节点数据变化

/k 节点数据（value）发生变化的时候，我们会收到一个事件通知，收到通知的前提条件，就是你要注册这个节点的数据变化事件（注册监听）

监听事件只能一次有效，如果要想重复监听需要再注册监听

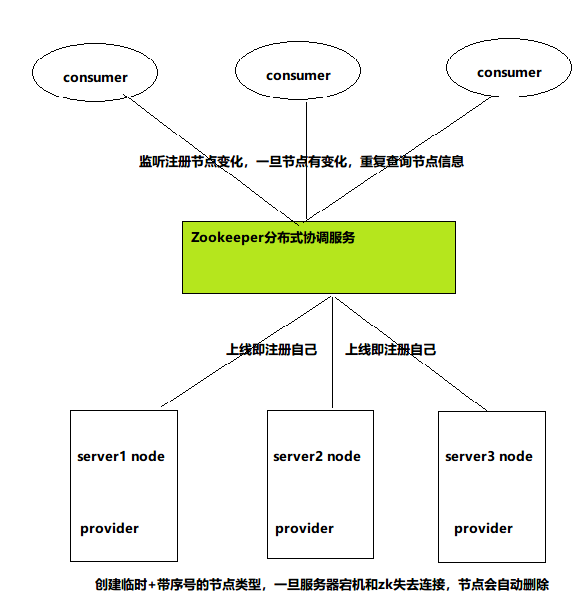
1. 监听节点变化

/k 如果/k这个节点下面新增或者删除节点的话，会收到一个变化事件，前提就是你的注册了这个节点监听

# Java API

|  |
| --- |
| import org.apache.zookeeper.\*; import org.junit.Test;  import java.io.IOException; import java.util.List;  */\*\*  \* author: sheep.Old  \* qq: 64341393  \* Created 2018/5/29  \*/* public class ZKClientApi {   private static final String *ZK\_SERVER* = "hdp-nd-01:2181,hdp-nd-02:2181,hdp-nd-03:2181";  private static final int *TIMEOUT* = 2000;   ZooKeeper zk = null;    */\*\*  \* 增加节点 create  \*/* @Test  public void testCreate() throws Exception {   zk = new ZooKeeper(*ZK\_SERVER*, *TIMEOUT*, null);  zk.create("/sheep", "1".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*PERSISTENT*);   zk.close();  }    */\*\*  \* 修改节点数据  \*/* @Test  public void testSet() throws Exception {  zk = new ZooKeeper(*ZK\_SERVER*, *TIMEOUT*, null);  zk.setData("/sheep", "laoyang1".getBytes(), -1);  zk.close();  }   */\*\*  \* 删除节点  \*/* @Test  public void testdelete() throws IOException, KeeperException, InterruptedException {  zk = new ZooKeeper(*ZK\_SERVER*, *TIMEOUT*, null);  zk.delete("/sheep", -1);  zk.close();  }    */\*\*  \* 查询节点  \*/* @Test  public void testList() throws IOException, KeeperException, InterruptedException {  zk = new ZooKeeper(*ZK\_SERVER*, *TIMEOUT*, null);  List<String> children = zk.getChildren("/", false);   for (String child : children) {  System.*out*.println("child = " + child);  }   zk.close();  }    */\*\*  \* 监听节点数据变化事件  \*/* @Test  public void testWatchDataChange() throws Exception {  zk = new ZooKeeper(*ZK\_SERVER*, *TIMEOUT*, event -> {  try {  System.*out*.print("event = " + event);  byte[] data = zk.getData("/sheep", true, null);   System.*out*.println(" data = " + new String(data));  } catch (KeeperException e) {  e.printStackTrace();  } catch (InterruptedException e) {  e.printStackTrace();  }  });    Thread.*sleep*(Long.*MAX\_VALUE*);  }    */\*\*  \* 监听节点变化事件  \*/* @Test  public void testWatchNodeChange() throws Exception {   zk = new ZooKeeper(*ZK\_SERVER*, *TIMEOUT*, new Watcher() {  @Override  public void process(WatchedEvent event) {  try {  System.*out*.println("event = " + event);  zk.getChildren("/sheep", true);  } catch (KeeperException e) {  e.printStackTrace();  } catch (InterruptedException e) {  e.printStackTrace();  }  }  });  zk.getChildren("/sheep", true);   Thread.*sleep*(Long.*MAX\_VALUE*);  zk.close();   }    */\*\*  \* 递归删除节点  \*/* @Test  public void testrmr() throws IOException, KeeperException, InterruptedException {  zk = new ZooKeeper(*ZK\_SERVER*, *TIMEOUT*, null);  *rmr*("/sheep", zk);  zk.close();  }   */\*\*  \* 递归删除所有节点信息  \** ***@param*** *path  \** ***@param*** *zk  \** ***@throws*** *KeeperException  \** ***@throws*** *InterruptedException  \*/* public static void rmr(String path, ZooKeeper zk) throws KeeperException, InterruptedException {   List<String> zkChildren = zk.getChildren(path, false);  for (String zkChild : zkChildren) {  *rmr*(path + "/" + zkChild, zk);  }  zk.delete(path, -1);  } } |

# 服务器动态上下线感知解决案例



* Provider

|  |
| --- |
| */\*\*  \* 服务提供者  \* 启动服务时，向zk服务器注册自己的主机名及服务端口号  \* 下线时，删除自己注册的节点信息  \* <p>  \* author: sheep.Old  \* qq: 64341393  \* Created 2018/5/29  \*/* public class Provider {   ZooKeeper zk = null;   */\*\*  \* 所有服务节点上线都往改节点下注册自己  \*/* final static String *pznode* = "/servers";   final static String *ZKSERVER* = "hdp-nd-01:2181,hd-nd-02:2181";  final static int *TIMEOUT* = 2000;   final static int *port* = 8082;    public static void main(String[] args) throws Exception {   Provider provider = new Provider();   // 连接zk  provider.connZookeeper();   // 向zk注册  provider.registerSelf();   // 等待调用  provider.handlerService();   }   */\*\*  \* 等待接受服务  \*/* private void handlerService() throws IOException {   ServerSocket socket = new ServerSocket(*port*);  while (true) {   Socket accept = socket.accept();  // *TODO* }   }   private void registerSelf() throws Exception {   // 注册的父节点还不存在，则创建父节点  if (null == zk.exists(*pznode*, false)) {  // 父节点为永久节点  zk.create(*pznode*, "online servers".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*PERSISTENT*);  }   // 开始注册 主机名及端口号  String address = InetAddress.*getLocalHost*().getHostName();   String result = zk.create(*pznode* + "/server", (address + ":" + *port*).getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL\_SEQUENTIAL*);   System.*out*.println("成功注册自己 = " + result);   }    public void connZookeeper() throws IOException {  zk = new ZooKeeper(*ZKSERVER*, *TIMEOUT*, null);  }  } |

* Consumer

|  |
| --- |
| import org.apache.zookeeper.KeeperException; import org.apache.zookeeper.WatchedEvent; import org.apache.zookeeper.Watcher; import org.apache.zookeeper.ZooKeeper;  import java.io.IOException; import java.util.ArrayList; import java.util.List; import java.util.Random;  */\*\*  \* author: sheep.Old  \* qq: 64341393  \* Created 2018/5/29  \*/* public class Consumer {    ZooKeeper zk = null;   */\*\*  \* 所有服务节点上线都往改节点下注册自己  \*/* final static String *pznode* = "/servers";   final static String *ZKSERVER* = "hdp-nd-01:2181,hd-nd-02:2181";  final static int *TIMEOUT* = 2000;   // 定义一个集合存放在线的服务器列表  ArrayList<String> onlineServers = null;   public static void main(String[] args) throws IOException, InterruptedException {   Consumer consumer = new Consumer();  // 连接zk  consumer.connectZK();  // 随机挑选服务  consumer.sendRequest();   }   private void sendRequest() throws InterruptedException {  while (true) {  Thread.*sleep*(1000);   if (onlineServers != null) {  Random random = new Random();  int love = random.nextInt(onlineServers.size());  System.*out*.println("本次挑中了 = " + onlineServers.get(love));  } else {  System.*err*.println("当前没有在线的服务器, 等待中...");  }  }  }   private void connectZK() throws IOException {   zk = new ZooKeeper(*ZKSERVER*, *TIMEOUT*, new Watcher() {  @Override  public void process(WatchedEvent event) {   try {   // 监听到事件，重新查询节点信息  getOnlineServer(zk);   } catch (KeeperException e) {  e.printStackTrace();  } catch (InterruptedException e) {  e.printStackTrace();  }  }  });  }   private void getOnlineServer(ZooKeeper zk) throws KeeperException, InterruptedException {  List<String> zkChildren = zk.getChildren(*pznode*, true);   ArrayList<String> currentOnlineServers = new ArrayList<>();  for (String zkChild : zkChildren) {  byte[] data = zk.getData(*pznode* + "/" + zkChild, false, null);  currentOnlineServers.add(new String(data));  }   onlineServers = currentOnlineServers;  System.*out*.println("节点变化，服务器当前在线列表："+onlineServers);  }  } |