# Zookeeper操作手册

## 1.Zookeeper的搭建方式

Zookeeper安装方式有三种，**单机模式**和**集群模式**以及**伪集群模式**。

■ 单机模式：Zookeeper只运行在一台服务器上，适合测试环境；  
■ 伪集群模式：就是在一台物理机上运行多个Zookeeper 实例；  
■ 集群模式：Zookeeper运行于一个集群上，适合生产环境，这个计算机集群被称为一个“集合体”（ensemble）

Zookeeper通过复制来实现高可用性，只要集合体中半数以上的机器处于可用状态，它就能够保证服务继续。**为什么一定要超过半数呢**？这跟Zookeeper的复制策略有关：zookeeper确保对znode树的每一个修改都会被复制到集合体中超过半数的机器上。

### 1.1Zookeeper的单机模式搭建

**下载ZooKeeper。**

**配置文件**：在conf目录下删除zoo\_sample.cfg文件，创建一个配置文件zoo.cfg。

tickTime=2000  
dataDir=D:\\zookeeper\\data

dataLogDir=D:\\zookeeper\\log     
clientPort=2181

**配置环境变量**：为了今后操作方便，我们需要对Zookeeper的环境变量进行配置。

添加变量：

ZOOKEEPER\_HOME ： D:\zookeeper\zookeeper-3.4.8

再path中添加，%ZOOKEEPER\_HOME%\bin

启动：zkServer

### 1.2 Zookeeper的伪集群模式搭建

Zookeeper不但可以在单机上运行单机模式Zookeeper，而且可以在单机模拟集群模式 Zookeeper的运行，也就是将不同节点运行在同一台机器。我们知道伪分布模式下Hadoop的操作和分布式模式下有着很大的不同，但是在集群为分布 式模式下对Zookeeper的操作却和集群模式下没有本质的区别。显然，集群伪分布式模式为我们体验Zookeeper和做一些尝试性的实验提供了很大 的便利。比如，我们在实验的时候，可以先使用少量数据在集群伪分布模式下进行测试。当测试可行的时候，再将数据移植到集群模式进行真实的数据实验。这样不 但保证了它的可行性，同时大大提高了实验的效率。这种搭建方式，比较简便，成本比较低，适合测试和学习，如果你的手头机器不足，就可以在一台机器上部署了 3个server。

具体操作如下：

在一台机器上部署了3个server，需要注意的是在集群为分布式模式下我们使用的每个配置文档模拟一台机器，也就是说单台机器及上运行多个Zookeeper实例。但是，必须保证每个配置文档的各个端口号不能冲突，除了clientPort不同之外，dataDir也不同。另外，还要在dataDir所对应的目录中创建myid文件来指定对应的Zookeeper服务器实例。

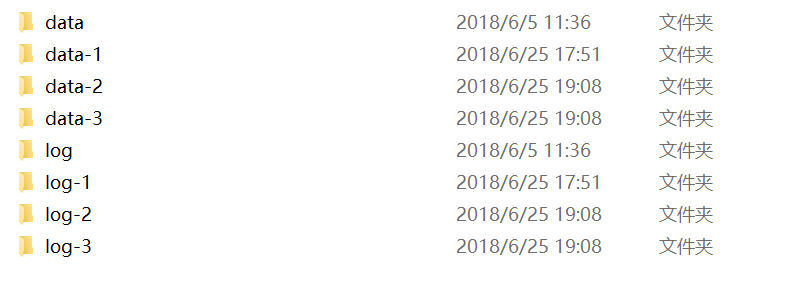
■ clientPort端口：如果在1台机器上部署多个server，那么每台机器都要不同的 clientPort，比如 server1是2181,server2是2182，server3是2183

■ dataDir和dataLogDir：dataDir和dataLogDir也需要区分下，将数据文件和日志文件分开存放，同时每个server的这两变量所对应的路径都是不同的

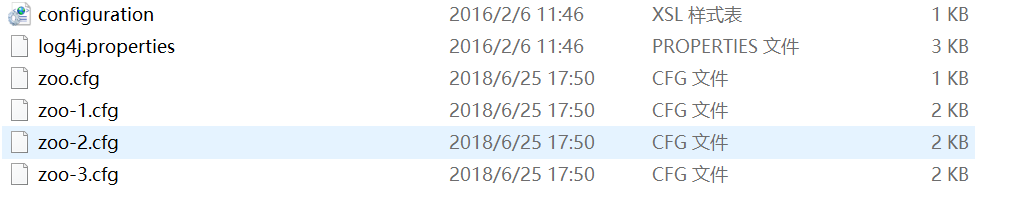
■ server.X和myid： server.X 这个数字就是对应，data/myid中的数字。在3个server的myid文件中分别写入了1，2，3，那么每个server中的zoo.cfg都配 server.1 server.2,server.3就行了。因为在同一台机器上，后面连着的2个端口，3个server都不要一样，否则端口冲突

操作如下：

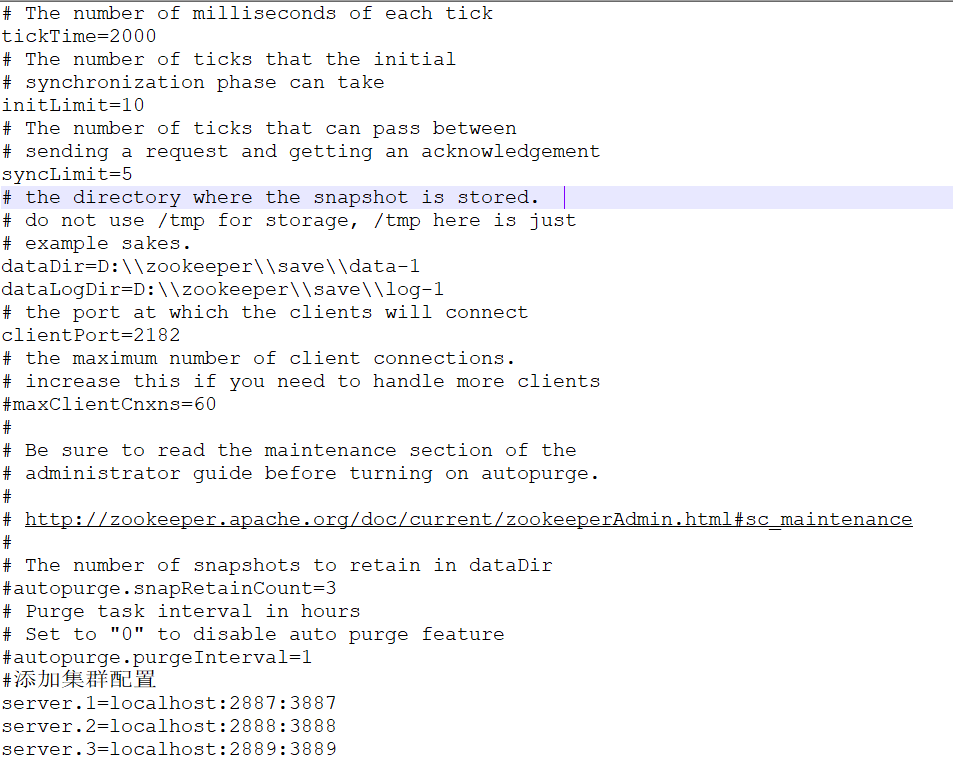
1. Save目录下建立data1、data2、data3和logs1、logs2、logs3，这些文件目录可以自定义在任何目录。



1. 将zoo.cfg复制3份



Zoo-1.cfg修改相应配置，如下：



对不同的zoo文件，只需制定相应的数据文件以及日志文件即可。

同时添加

server.1=localhost:2887:3887  
server.2=localhost:2888:3888

 server.3=localhost:2889:3889

        server.num=ip/domain：Port1：Port2

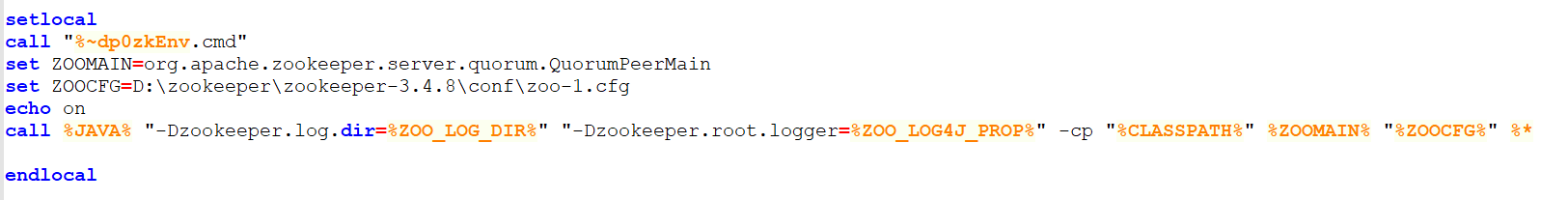
        其中**num：**表示数字表示第几号服务器；**ip/domain** ：是服务器域名或者ip地址。**Port1**：表示这个服务器和集群中的Leader服务器交换信息的端口；**Port2：**表示万一集群中的Leader服务器挂了，需要一个端口重新进行选举，选出一个新的Leader，这个端口就是用来执行选举时服务器相互通信的端口。

        由于我们是伪集群，所以ip或者域名是一样的，所以要分配不同的端口号。注意三个节点的端口不能相同

1. 修改命令，加载不同的配置。复制zkServer.cmd并且重命名为，zkServer-1.cmd，zkServer-2.cmd，zkServer-3.cmd 分别修改这三个cmd如下所示。



1. zkServer-1.cmd



1. 分别启动zkServer-1 .cmd、zkServer-2.cmd zkServer-3.cmd

注意单独启动时会报错，当三个节点全部启动才会正常。

## Zookeeper配置

### 2.1 基本配置

下面是在最低配置要求中必须配置的参数：

**(1) client**：监听客户端连接的端口。  
**(2) tickTime**：基本事件单元，这个时间是作为Zookeeper服务器之间或客户端与服务器之间维持心跳的时间间隔，每隔tickTime时间就会发送一个心跳；最小 的session过期时间为2倍tickTime 　　  
**dataDir**：存储内存中数据库快照的位置，如果不设置参数，更新食物的日志将被存储到默认位置。

### 2.2 高级配置

**(1) dataLogdDir**

这个操作让管理机器把事务日志写入“dataLogDir”所指定的目录中，而不是“dataDir”所指定的目录。这将允许使用一个专用的日志设备，帮助我们避免日志和快照的竞争

**(2)** **maxClientCnxns**

这个操作将限制连接到Zookeeper的客户端数量，并限制并发连接的数量，通过IP来区分不同的客户端。此配置选项可以阻止某些类别的Dos攻击。将他设置为零或忽略不进行设置将会取消对并发连接的限制。

例如，此时我们将maxClientCnxns的值设为1，如下所示：

# set maxClientCnxns  
   maxClientCnxns=1

启动Zookeeper之后，首先用一个客户端连接到Zookeeper服务器上。之后如果有第二个客户端尝试对Zookeeper进行连接，或者有某些隐式的对客户端的连接操作，将会触发Zookeeper的上述配置。

**(3)** **minSessionTimeout**和**maxSessionTimeout**

即最小的会话超时和最大的会话超时时间。在默认情况下，minSession=2\*tickTime；maxSession=20\*tickTime。

### 2.3 集群配置

**(1)** **initLimit**

此配置表示，允许follower(相对于Leaderer言的“客户端”)连接并同步到Leader的初始化连接时间，以tickTime为单位。当初始化连接时间超过该值，则表示连接失败。

**(2)** **syncLimit**

此配置项表示Leader与Follower之间发送消息时，请求和应答时间长度。如果follower在设置时间内不能与leader通信，那么此follower将会被丢弃。

**(3)** **server.A=B：C：D**

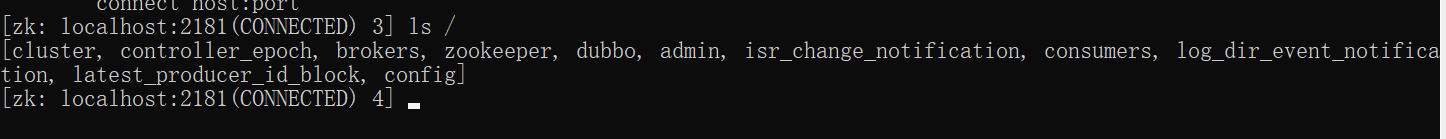
A：其中 A 是一个数字，表示这个是服务器的编号；  
B：是这个服务器的 ip 地址；  
C：Leader选举的端口；  
D：Zookeeper服务器之间的通信端口。

**(4) myid**和**zoo.cfg**

除了修改 zoo.cfg 配置文件，集群模式下还要配置一个文件 myid，这个文件在 dataDir 目录下，这个文件里面就有一个数据就是 A 的值，Zookeeper 启动时会读取这个文件，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那个 server。

## Zookeeper基本操作

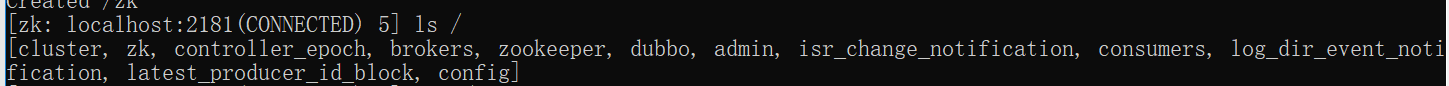
1. 启动zkServer
2. 启动zkCli链接只服务端zkCli -server local host：2181，链接成功后，再次按下回车即可输入命令。
3. 使用ls命令查看当前Zookeeper中所包含的内容：ls /



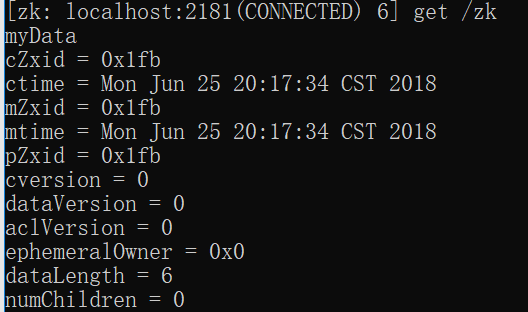
1. 创建一个新的Znode节点"zk"，以及和它相关字符，执行命令：create /zk myData



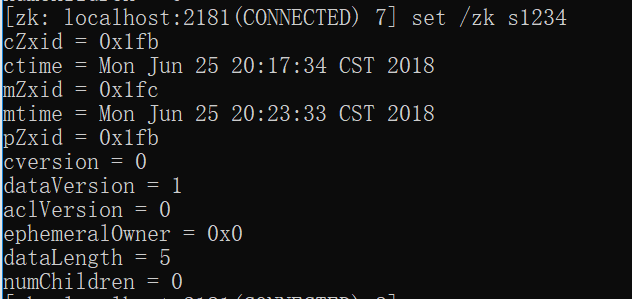
(3) 再次使用ls命令来查看现在Zookeeper的中所包含的内容：ls /



（4）使用get命令来确认第二步中所创建的Znode是否包含我们创建的字符串，执行命令：get /zk



(5) 接下来通过set命令来对zk所关联的字符串进行设置，执行命令：set /zk s1234



(7) 下面我们将刚才创建的Znode删除，执行命令：delete /zk

## Api使用

先打开zkserver。

### 创建组

新建源文件。

import org.apache.zookeeper.\*;  
  
import java.io.IOException;  
import java.util.concurrent.CountDownLatch;  
  
public class CreateGroup implements Watcher {  
 private static final int *SESSION\_TIMEOUT*=5000;  
  
 private ZooKeeper zk;  
 private CountDownLatch connectedSignal=new CountDownLatch(1);  
 public void process(WatchedEvent watchedEvent) {  
 if(watchedEvent.getState()== Event.KeeperState.*SyncConnected*){  
 connectedSignal.countDown();  
 }  
 }  
 private void close() throws InterruptedException {  
 zk.close();  
 }  
  
 private void create(String groupName) throws KeeperException, InterruptedException {  
 String path="/"+groupName;  
 if(zk.exists(path, false)== null){  
 zk.create(path, null/\*data\*/, ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*PERSISTENT*);  
 }  
 System.*out*.println("Created:"+path);  
 }  
  
 private void connect(String hosts) throws IOException, InterruptedException {  
 zk = new ZooKeeper(hosts, *SESSION\_TIMEOUT*, this);  
 connectedSignal.await();  
 }  
  
 public static void main(String[] args) throws IOException, InterruptedException, KeeperException {  
 CreateGroup createGroup = new CreateGroup();  
 createGroup.connect("localhost:2181");  
 createGroup.create("zoo");  
 createGroup.close();  
 }  
}

在上面代码中，main()方法执行时，创建了一个**CreateGroup**的**实例**并且调用了这个实例的connect()方法。connect方法实例化了一个新的ZooKeeper类的对象，这个类是客户端API中的主要类，并且负责维护客户端和ZooKeeper服务之间的连接。ZooKeeper类的构造函数有三个参数：  
　　第一个是：ZooKeeper服务的主机地址，可指定端口，默认端口是2181。  
　　第二个是：以毫秒为单位的会话超时参数，这里我们设成5秒。  
　　第三个是：参数是一个Watcher对象的实例。（Watcher对象接收来自于ZooKeeper的回调，以获得各种事件的通知。在这个例子中，CreateGroup是一个Watcher对象，因此我们将它传递给ZooKeeper的构造函数）。

当一个ZooKeeper的实例被创建时，会启动一个线程连接到ZooKeeper服务。由于对构造函数的调用是立即返回的，因此在使用新建的ZooKeeper对象之前一定要等待其与ZooKeeper服务之间的连接建立成功。我们使用Java的CountDownLatch类来阻止使用新建的ZooKeeper对象，直到这个ZooKeeper对象已经准备就绪。这就是Watcher类的用途。

客户端已经与ZooKeeper建立连接后，Watcher的process()方法会被调用，参数是一个表示该连接的事件。在接收到一个连接事件（由Watcher.Event.KeeperState的枚举型值SyncConnected来表示）时，我们通过调用CountDownLatch的countDown()方法来递减它的计数器。锁存器(CountDownLatc)被创建时带有一个值为1的计数器，用于表示在它释放所有等待线程之前需要发生的事件数。在调用一欢countDown()方法之后，计数器的值变为0，则await()方法返回。现在connect()方法已经返回，下一个执行的是CreateGroup的create()方法。

在这个方法中，我们使用ZooKeeper实例中的create()方法来创建一个新的ZooKeeper的znode。所需的参数包括：

路径：用字符串表示。  
　　　　znode的内容：字节数组，本例中使用空值。  
　　　　访问控制列表：简称ACL，本例中使用了完全开放的ACL，允许任何客户端对znode进行读写。  
　　　　创建znode的类型：有两种类型的znode：短暂的和持久的。

create()方法的返回值是ZooKeeper所创建的路径，我们用这个返回值来打印一条表示路径成功创建的消息。当我们查看“顺序znode”(sequential znode)时．会发现create()方法返回的路径与传递给该方法的路径不同。

### 加入组

下面的这一段程序用于注册组的成员。每个组成员将作为一个程序运行，并且加入到组中。当程序退出时，这个组成员应当从组中被删除。为了实现这一点，我们在ZooKeeper的命名空间中使用短暂znode来代表一个组成员。在基类ConnectionWatcher中，对创建和连接ZooKeeper实例的程序逻辑进行了重构，

import org.apache.zookeeper.\*;  
  
import java.util.concurrent.CountDownLatch;  
  
public class ConnectionWatcher implements Watcher {  
 private static final int *SESSION\_TIMEOUT* = 5000;  
  
 protected ZooKeeper zk;  
 private CountDownLatch connectedSignal = new CountDownLatch(1);  
  
 public void connect(String hosts) throws Exception{  
 //传入: 主机地址; 会话超时参数; Watcher对象,用于接收来自zookeeper的回调,获得各种事件的通知.  
 //内部会创建一个线程连接到服务器;  
 zk = new ZooKeeper(hosts, *SESSION\_TIMEOUT*, this);  
 //等待连接成功的响应.  
 connectedSignal.await();  
 }  
 // 当客户端与服务端建立成功之后, watcher 的 process()方法会被调用. 参数是一个用于表示该连接的事件  
 public void process(WatchedEvent watchedEvent) {  
 //返回响应, 表示连接已经成功  
 if(watchedEvent.getState() == Event.KeeperState.*SyncConnected*){  
 connectedSignal.countDown();  
 }  
 }  
  
 */\*\*  
 \* 创建一个新的 Zookeeper 的 znode,  
 \** ***@param*** *groupName  
 \** ***@throws*** *Exception  
 \*/* public void create(String groupName) throws Exception{  
 String path = "/" + groupName;  
  
 String createdPath = zk.create(path, null, ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*PERSISTENT*);  
  
 System.*out*.println("created " + createdPath);  
 }  
  
 public void close() throws Exception{  
 zk.close();  
 }  
  
}

import org.apache.zookeeper.CreateMode;  
import org.apache.zookeeper.KeeperException;  
import org.apache.zookeeper.ZooDefs;  
  
import java.io.IOException;  
  
public class JoinGroup extends ConnectionWatcher {  
  
 public void join(String groupName,String memberName) throws KeeperException, InterruptedException{  
 String path="/"+groupName+"/"+memberName;  
 String createdPath=zk.create(path, null, ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL*);  
 System.*out*.println("Created:"+createdPath);  
 }  
  
 public static void main(String[] args) throws Exception{  
 JoinGroup joinGroup = new JoinGroup();  
 joinGroup.connect("localhost:2181");  
 joinGroup.join("zoo", "cow");  
  
 //stay alive until process is killed or thread is interrupted  
 Thread.*sleep*(Long.*MAX\_VALUE*);  
 }  
}

JoinGroup的代码与CreateGroup非常相似，在它的join()方法中，创建短暂znode，作为组znode的子节点，然后通过休眠来模拟正在做某种工作，直到该进程被强行终止。接着，你会看到随着进程终止，这个短暂znode被ZooKeeper删除。

### 列出组成员

import java.util.List;  
  
public class ListGroup extends ConnectionWatcher {  
 public void list(String groupNmae) throws KeeperException, InterruptedException{  
 String path ="/"+groupNmae;  
 try {  
 List<String> children = zk.getChildren(path, false);  
 if(children.isEmpty()){  
 System.*out*.printf("No memebers in group %s\n",groupNmae);  
 System.*exit*(1);  
 }  
 for(String child:children){  
 System.*out*.println(child);  
 }  
 } catch (KeeperException.NoNodeException e) {  
 System.*out*.printf("Group %s does not exist \n", groupNmae);  
 System.*exit*(1);  
 }  
 }  
 public static void main(String[] args) throws Exception{  
 ListGroup listGroup = new ListGroup();  
 listGroup.connect("localhost:2181");  
 listGroup.list("zoo");  
 listGroup.close();  
 }  
}

在list()方法中，我们调用了getChildren()方法来检索并打印输出一个znode的子节点列表，调用参数为：该znode的路径和设为false的观察标志。如果在一znode上设置了观察标志，那么一旦该znode的状态改变，关联的观察(Watcher)会被触发。虽然在这里我们可以不使用观察，但在查看一个znode的子节点时，也可以设置观察，让应用程序接收到组成员加入、退出和组被删除的有关通知。

### 删除组

import java.util.List;  
  
public class DeleteGroup extends ConnectionWatcher {  
 public void delete(String groupName) throws InterruptedException, KeeperException {  
 String path="/"+groupName;  
 List<String> children;  
 try {  
 children = zk.getChildren(path, false);  
 for(String child:children){  
 zk.delete(path+"/"+child, -1);  
 }  
 zk.delete(path, -1);  
 } catch (KeeperException.NoNodeException e) {  
 System.*out*.printf("Group %s does not exist\n", groupName);  
 System.*exit*(1);  
 }  
 }  
 public static void main(String[] args) throws Exception{  
 DeleteGroup deleteGroup = new DeleteGroup();  
 deleteGroup.connect("localhost:2181");  
 deleteGroup.delete("zoo");  
 deleteGroup.close();  
 }  
}

下面来看如何删除一个组。ZooKeeper类提供了一个delete()方法，该方法有两个参数：

1. 路径

2. 版本号

如果所提供的版本号与znode的版本号一致，ZooKeeper会删除这个znode。这是一种乐观的加锁机制，使客户端能够检测出对znode的修改冲突。通过将版本号设置为-1，可以绕过这个版本检测机制，不管znode的版本号是什么而直接将其删除。ZooKeeper不支持递归的删除操作，因此在删除父节点之前必须先删除子节点。