# **<http://www.cnblogs.com/fxwl/p/6060617.html>**

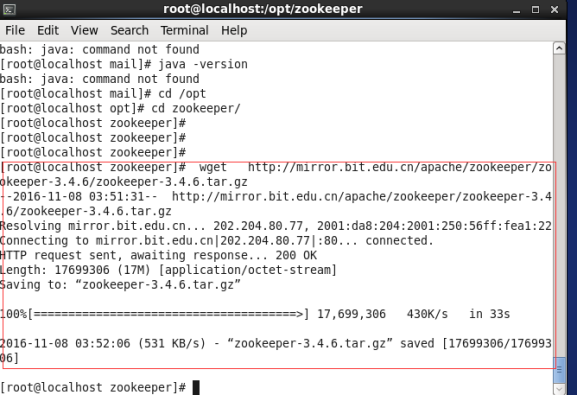
# **[zookeeper的安装及共享锁的应用](http://www.cnblogs.com/fxwl/p/6060617.html)**

# ****Zookeeper的安装及共享锁的应用****

## **1.**zookeeper的安装****

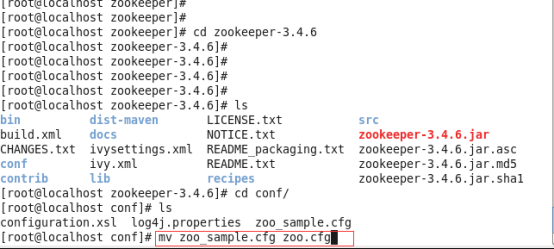
  1.1  下载安装包

     Wget <http://mirror.bit.edu.cn/apache/zookeeper/zookeeper-3.4.6/zookeeper-3.4.6.tar.gz>



  1.2 .解压缩安装包 tar -zxvf zookeeper-3.4.6.tar.gz 这里我们的操作都是在一个新建目录/opt/zookeeper 文件夹下完成的，自己可以根据实际需要创建文件夹

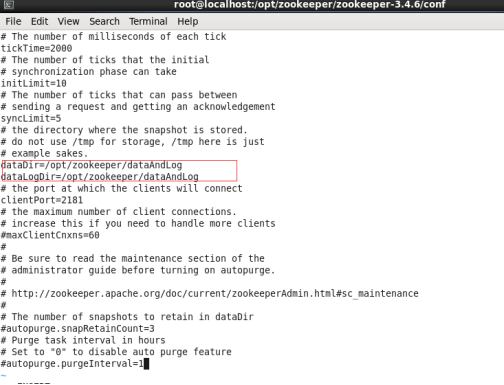
  1.3 进入/opt/zookeeper/zookeeper-3.4.6/conf文件夹，将zoo\_sample.cfg 文件重命名为 zoo.cfg ，命令:mv  zoo\_sample.cfg  zoo.cfg



 1.4 修改zoo.cfg配置文件 命令vi zoo.cfg ,将dataDir，dataLogDar参数进行配置，文件夹可以自己定义，在这里我配置成如下：

    　dataDir=/opt/zookeeper/dataAndLog

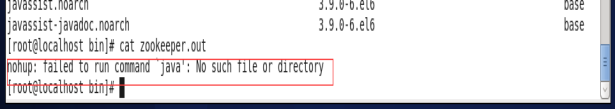
 dataLogDir=/opt/zookeeper/dataAndLog



1.5 进入/opt/zookeeper/zookeeper-3.4.6/bin目录，启动zookeeper,命令为: ./zkServer.sh start

1.6 查看zookeeper 运行状态 ./zkServer.sh status，或者输入ps aux|grep zookeeper ,这里我们发现zookeeper 并没有正常启动，这里我们可以输入cat zookeeper.out 来查看启动信息

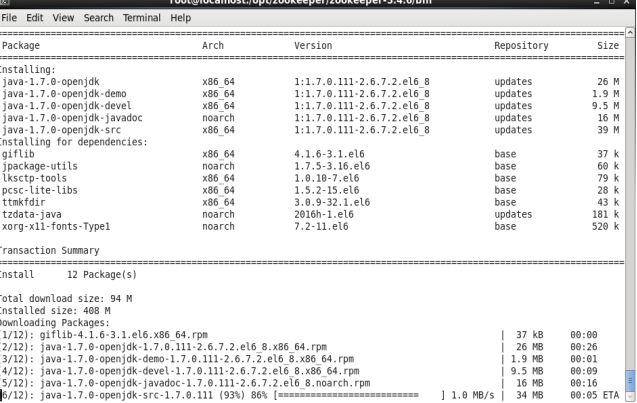


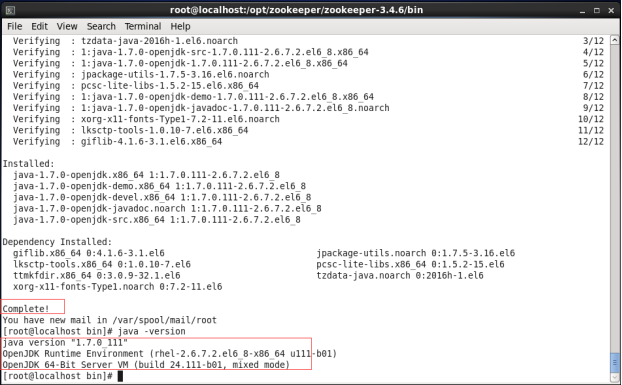


这里我们发现没有安装jdk

1.7 安装jdk

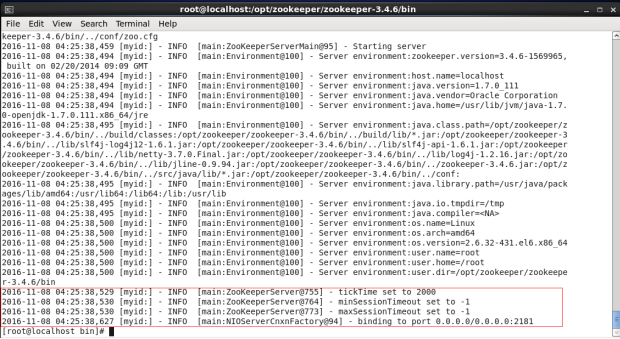
通过yum 安装openjdk1.7 yum -y install java-1.7.0-openjdk\* ，jdk安装完成后输入 java -version  来查看jdk安装情况

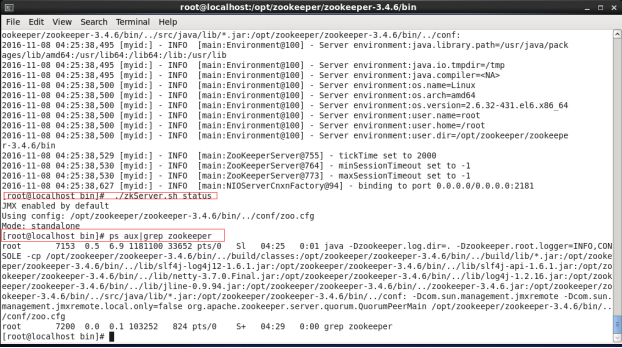




1.8 再次启动zookeeper

   输入 ./zkServer.sh start 重新启动zookeeper ,然后再次通过 ./zkServer.sh status  cat zookeeper.out  ps aux|grep zookeeper  来查看启动状态启动完成.





1.9 配置防火墙端口(不配置防火墙 外网注册不了服务)

   Zookeeper 需要用到3个端口  2181 ，2888 ，3888 ，第一个是客户端连接的端口，后两个是做zookeeper集群时所用的端口。

修改防火墙配置文件  vi /etc/sysconfig/iptables 加入下面内容

  -A INPUT -p tcp -m tcp --dport 2181 -j ACCEPT

  -A INPUT -p tcp -m tcp --dport 2888 -j ACCEPT

  -A INPUT -p tcp -m tcp --dport 3888 -j ACCEPT

 然后 重启防火墙

/etc/init.d/iptables restart

2.0 集群的配置

   在生产环境中，要配置成分布式模式，才能发挥威力。

    ZooKeeper集群一般被称为ZooKeeper ensemble，或者 quorum.

准备3台机器

假设有三台机器，hostname和ip对应关系是：

192.168.1.101 hadoop101

192.168.1.102 hadoop102

192.168.1.103 hadoop103

ZooKeeper不存在明显的master/slave关系，各个节点都是服务器，leader挂了，会立马从follower中选举一个出来作为leader.

由于没有主从关系，也不用配置SSH无密码登录了，各个zk服务器是自己启动的，互相之间通过TCP端口来交换数据。

修改hadoop101的配置文件conf/zoo.cfg

....

....

clientPort=2181

server.1=hadoop101:2888:3888 /server.1=192.168.1.101:2888:3888

server.2=hadoop102:2888:3888/server.1=192.168.1.102:2888:3888

server.3=hadoop103:2888:3888/server.1=192.168.1.103:2888:3888

修改完后拷贝到 hadoop102,以及 hadoop103上.

然后启动每台机器，因为3个节点的启动是有顺序的，所以在陆续启动三个节点的时候，前面先启动的节点连接未启动的节点的时候会报出一些错误。可以忽略。

可以使用java客户端连接ZooKeeper集群中的任意一台服务器了。

## **2.**分布式共享锁的应用****

 2.1  这里讲解一下使用java客户端来获取共享锁

      这里有一个客户端类 DistributedLock.java 代码如下：

[IMG_265](http://www.cnblogs.com/fxwl/p/javascript:void(0);" \o "复制代码)

package com.daat.front.activity.web.actions.tools;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.concurrent.CountDownLatch;

import java.util.concurrent.TimeUnit;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import org.apache.log4j.Logger;

import org.apache.zookeeper.CreateMode;

import org.apache.zookeeper.KeeperException;

import org.apache.zookeeper.WatchedEvent;

import org.apache.zookeeper.Watcher;

import org.apache.zookeeper.ZooDefs;

import org.apache.zookeeper.ZooKeeper;

import org.apache.zookeeper.data.Stat;

import com.daat.front.activity.service.bo.impl.BoActivity;

/\*\*

\* 基于zoopkeeper的分布式共享锁

\* @author yank

\*

\*/

public class DistributedLock implements Lock, Watcher {

private ZooKeeper zk;

private String root = "/locks";

private String lockName;

private String waitNode;

private String myZnode;

private CountDownLatch latch;

private int sessionTimeout = 30000;

private List<Exception> exception = new ArrayList<Exception>();

private static Logger log = Logger.getLogger(DistributedLock.class);

public DistributedLock(String config, String lockName) {

this.lockName = lockName;

try {

zk = new ZooKeeper(config, sessionTimeout, this);

Stat stat = zk.exists(root, false);

if (stat == null) {

zk.create(root, new byte[0], ZooDefs.Ids.OPEN\_ACL\_UNSAFE,CreateMode.PERSISTENT);

}

} catch (IOException e) {

e.printStackTrace();

exception.add(e);

} catch (KeeperException e) {

e.printStackTrace();

exception.add(e);

} catch (InterruptedException e) {

e.printStackTrace();

exception.add(e);

}

}

public void process(WatchedEvent event) {

if (this.latch != null) {

this.latch.countDown();

}

}

public void lock() {

if (exception.size() > 0) {

throw new LockException(exception.get(0));

}

try {

if (this.tryLock()) {

System.out.println("Thread " + Thread.currentThread().getId()+ " " + myZnode + " get lock true");

return;

} else {

waitForLock(waitNode, sessionTimeout);

}

} catch (KeeperException e) {

e.printStackTrace();

throw new LockException(e);

} catch (InterruptedException e) {

e.printStackTrace();

throw new LockException(e);

}

}

public boolean tryLock() {

try {

String splitStr = "\_lock\_";

if (lockName.contains(splitStr))

throw new LockException("lockName can not contains \\u000B");

myZnode = zk.create(root + "/" + lockName + splitStr, new byte[0],ZooDefs.Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL);

System.out.println(myZnode + " is created ");

List<String> subNodes = zk.getChildren(root, false);

List<String> lockObjNodes = new ArrayList<String>();

for (String node : subNodes) {

String \_node = node.split(splitStr)[0];

if (\_node.equals(lockName)) {

lockObjNodes.add(node);

}

}

Collections.sort(lockObjNodes);

System.out.println(myZnode + "==" + lockObjNodes.get(0));

if (myZnode.equals(root + "/" + lockObjNodes.get(0))) {

return true;

}

String subMyZnode = myZnode.substring(myZnode.lastIndexOf("/") + 1);

waitNode = lockObjNodes.get(Collections.binarySearch(lockObjNodes,subMyZnode) - 1);

} catch (KeeperException e) {

e.printStackTrace();

throw new LockException(e);

} catch (InterruptedException e) {

e.printStackTrace();

throw new LockException(e);

}

return false;

}

public boolean tryLock(long time, TimeUnit unit) {

try {

if (this.tryLock()) {

return true;

}

return waitForLock(waitNode, time);

} catch (Exception e) {

e.printStackTrace();

}

return false;

}

private boolean waitForLock(String lower, long waitTime)

throws InterruptedException, KeeperException {

Stat stat = zk.exists(root + "/" + lower, true);

if (stat != null) {

System.out.println("Thread " + Thread.currentThread().getId()+ " waiting for " + root + "/" + lower);

this.latch = new CountDownLatch(1);

this.latch.await(waitTime, TimeUnit.MILLISECONDS);

this.latch = null;

}

return true;

}

public void unlock() {

try {

System.out.println("unlock " + myZnode);

zk.delete(myZnode, -1);

myZnode = null;

zk.close();

} catch (InterruptedException e) {

e.printStackTrace();

} catch (KeeperException e) {

e.printStackTrace();

}

}

public void lockInterruptibly() throws InterruptedException {

this.lock();

}

public Condition newCondition() {

return null;

}

public class LockException extends RuntimeException {

private static final long serialVersionUID = 1L;

public LockException(String e) {

super(e);

}

public LockException(Exception e) {

super(e);

}

}

}

[IMG_266](http://www.cnblogs.com/fxwl/p/javascript:void(0);" \o "复制代码)

2.2 再写一个测试类 TestLock.java 代码如下：

[IMG_267](http://www.cnblogs.com/fxwl/p/javascript:void(0);" \o "复制代码)

package com.daat.front.base.web.tools;

import java.util.Date;

public class TestLock implements Runnable{

public static void main(String[] args) {

for(int i=0;i<10;i++){

Thread ad=new Thread(new TestLock());

ad.start();

}

}

@Override

public void run() {

DistributedLock lock = null;

try {

System.out.println("线程开启:"+Thread.currentThread().getId());

//多个锁用“，” 分隔开

//lock = new DistributedLock("10.168.128.113:2181,10.168.173.159:2181", "amount");

Thread.sleep(1000);

lock = new DistributedLock("192.168.122.129:2181", "testLock");

lock.lock();

System.out.println("线程："+Thread.currentThread().getId()+"在"+new Date().getTime()+"时间获得锁");

Thread.sleep(4000);

System.out.println("===Thread "+ Thread.currentThread().getId() + " running");

} catch (Exception e) {

e.printStackTrace();

}finally{

if (lock != null){

System.out.println("线程："+Thread.currentThread().getId()+"在"+new Date().getTime()+"释放锁");

lock.unlock();

}

}

}

}

[IMG_268](http://www.cnblogs.com/fxwl/p/javascript:void(0);" \o "复制代码)

测试类是开启10个线程同时去做锁竞争操作，看看是否可以控制

获取锁---->释放锁  的操作，测试结果如下：

第一次测试：

线程开启:9

线程开启:11

线程开启:10

线程开启:12

线程开启:15

线程开启:14

线程开启:13

线程开启:16

线程开启:17

线程开启:18

线程：11在1478591252342时间获得锁

线程：11在1478591257342释放锁

线程：14在1478591257348时间获得锁

线程：14在1478591262349释放锁

线程：15在1478591262352时间获得锁

线程：15在1478591267352释放锁

线程：10在1478591267360时间获得锁

线程：10在1478591272360释放锁

线程：12在1478591272364时间获得锁

线程：12在1478591277364释放锁

线程：16在1478591277367时间获得锁

线程：16在1478591282368释放锁

线程：9在1478591282371时间获得锁

线程：9在1478591287371释放锁

线程：18在1478591285302时间获得锁

线程：18在1478591290302释放锁

线程：13在1478591285302时间获得锁

线程：13在1478591290302释放锁

线程：17在1478591285303时间获得锁

线程：17在1478591290303释放锁

第二次测试：

线程开启:9

线程开启:11

线程开启:12

线程开启:10

线程开启:13

线程开启:14

线程开启:15

线程开启:16

线程开启:18

线程开启:17

线程：17在1478591690760时间获得锁

线程：17在1478591694760释放锁

线程：18在1478591694767时间获得锁

线程：18在1478591698767释放锁

线程：14在1478591698770时间获得锁

线程：14在1478591702771释放锁

线程：15在1478591702775时间获得锁

线程：15在1478591706775释放锁

线程：12在1478591706778时间获得锁

线程：12在1478591710778释放锁

线程：11在1478591710783时间获得锁

线程：11在1478591714783释放锁

线程：9在1478591714786时间获得锁

线程：9在1478591718787释放锁

线程：10在1478591718790时间获得锁

线程：10在1478591722790释放锁

线程：16在1478591720773时间获得锁

线程：16在1478591724773释放锁

线程：13在1478591720774时间获得锁

线程：13在1478591724774释放锁

通过测试。