public class ZooKeeperDistributedLock implements Watcher{

private ZooKeeper zk;

private String locksRoot= "/locks";

private String productId;

private String waitNode;

private String lockNode;

private CountDownLatch latch;

private CountDownLatch connectedLatch = new CountDownLatch(1);

private int sessionTimeout = 30000;

public ZooKeeperDistributedLock(String productId){

this.productId = productId;

try {

String address = "192.168.31.187:2181,192.168.31.19:2181,192.168.31.227:2181";

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

connectedLatch.await();

} catch (IOException e) {

throw new LockException(e);

} catch (KeeperException e) {

throw new LockException(e);

} catch (InterruptedException e) {

throw new LockException(e);

}

}

public void process(WatchedEvent event) {

if(event.getState()==KeeperState.SyncConnected){

connectedLatch.countDown();

return;

}

if(this.latch != null) {

this.latch.countDown();

}

}

public void acquireDistributedLock() {

try {

if(this.tryLock()){

return;

}

else{

waitForLock(waitNode, sessionTimeout);

}

} catch (KeeperException e) {

throw new LockException(e);

} catch (InterruptedException e) {

throw new LockException(e);

}

}

public boolean tryLock() {

try {

// 传入进去的locksRoot + “/” + productId

// 假设productId代表了一个商品id，比如说1

// locksRoot = locks

// /locks/10000000000，/locks/10000000001，/locks/10000000002

lockNode = zk.create(locksRoot + "/" + productId, new byte[0], ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.EPHEMERAL\_SEQUENTIAL);

// 看看刚创建的节点是不是最小的节点

// locks：10000000000，10000000001，10000000002

List<String> locks = zk.getChildren(locksRoot, false);

Collections.sort(locks);

if(lockNode.equals(locksRoot+"/"+ locks.get(0))){

//如果是最小的节点,则表示取得锁

return true;

}

//如果不是最小的节点，找到比自己小1的节点

int previousLockIndex = -1;

for(int i = 0; i < locks.size(); i++) {

if(lockNode.equals(locksRoot + “/” + locks.get(i))) {

previousLockIndex = i - 1;

break;

}

}

this.waitNode = locks.get(previousLockIndex);

} catch (KeeperException e) {

throw new LockException(e);

} catch (InterruptedException e) {

throw new LockException(e);

}

return false;

}

private boolean waitForLock(String waitNode, long waitTime) throws InterruptedException, KeeperException {

Stat stat = zk.exists(locksRoot + "/" + waitNode, true);

if(stat != null){

this.latch = new CountDownLatch(1);

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

}

return true;

}

public void unlock() {

try {

// 删除/locks/10000000000节点

// 删除/locks/10000000001节点

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

zk.delete(lockNode,-1);

lockNode = null;

zk.close();

} catch (InterruptedException e) {

e.printStackTrace();

} catch (KeeperException e) {

e.printStackTrace();

}

}

public class LockException extends RuntimeException {

private static final long serialVersionUID = 1L;

public LockException(String e){

super(e);

}

public LockException(Exception e){

super(e);

}

}

// 如果有一把锁，被多个人给竞争，此时多个人会排队，第一个拿到锁的人会执行，然后释放锁，后面的每个人都会去监听排在自己前面的那个人创建的node上，一旦某个人释放了锁，排在自己后面的人就会被zookeeper给通知，一旦被通知了之后，就ok了，自己就获取到了锁，就可以执行代码了

}