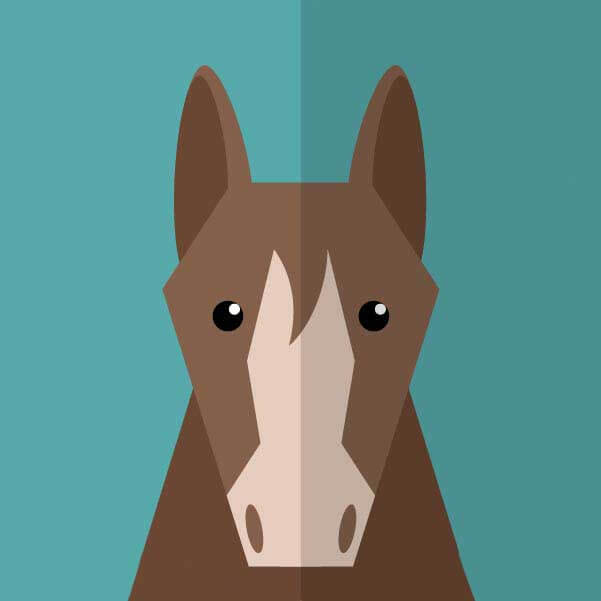
https://www.jianshu.com/p/31335efec309

**ZooKeeper + Curator 实现分布式锁**

[](https://www.jianshu.com/u/381dcaf1555a)

[jnil](https://www.jianshu.com/u/381dcaf1555a) 关注

2017.09.02 20:50\* 字数 872 阅读 1090评论 0喜欢 2

在 JDK 的 java.util.concurrent.locks 中, 为我们提供了可重入锁, 读写锁, 及超时获取锁的方法. 为我们提供了完好的支持, 但是在分布式系统中, 当多个应用需要共同操作某一个资源时. 我么就无法使用 JDK 来实现了, 这时就需要使用一个外部服务来为此进行支持, 现在我们选用 [ZooKeeper](https://link.jianshu.com/?t=https://zookeeper.apache.org" \t "_blank) + [Curator](https://link.jianshu.com/?t=http://curator.apache.org) 来完成分布式锁

**项目环境**

* ZooKeeper 3.5.3-beta
* Curator 4.0.0

如果 ZooKeeper 版本为 3.4.x, 请进行[兼容处理](https://link.jianshu.com/?t=http://curator.apache.org/zk-compatibility.html" \t "_blank)

**准备工作**

下载、安装、启动 ZooKeeper, 可以查看这篇博文[ZooKeeper 的安装、配置、启动和使用（一）——单机模式](https://link.jianshu.com/?t=http://blog.csdn.net/gaohuanjie/article/details/37736939" \t "_blank)  
如果想跳过这一步的话请参考最下面的[便捷测试](# 便捷测试)  
创建一个 Maven 工程, 然后引入所需资源

<dependency>

<groupId>org.apache.curator</groupId>

<artifactId>curator-recipes</artifactId>

<version>4.0.0</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

在 src/test/java 下创建一个 DistributedLockDemo 类  
基本代码如下

public class DistributedLockDemo {

*// ZooKeeper 锁节点路径, 分布式锁的相关操作都是在这个节点上进行*

private final String lockPath = "/distributed-lock";

*// ZooKeeper 服务地址, 单机格式为:(127.0.0.1:2181), 集群格式为:(127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183)*

private String connectString;

*// Curator 客户端重试策略*

private RetryPolicy retry;

*// Curator 客户端对象*

private CuratorFramework client;

*// client2 用户模拟其他客户端*

private CuratorFramework client2;

*// 初始化资源*

@Before

public void init() throws Exception {

*// 设置 ZooKeeper 服务地址为本机的 2181 端口*

connectString = "127.0.0.1:2181";

*// 重试策略*

*// 初始休眠时间为 1000ms, 最大重试次数为 3*

retry = new ExponentialBackoffRetry(1000, 3);

*// 创建一个客户端, 60000(ms)为 session 超时时间, 15000(ms)为链接超时时间*

client = CuratorFrameworkFactory.newClient(connectString, 60000, 15000, retry);

client2 = CuratorFrameworkFactory.newClient(connectString, 60000, 15000, retry);

*// 创建会话*

client.start();

client2.start();

}

*// 释放资源*

@After

public void close() {

CloseableUtils.closeQuietly(client);

}

}

[**共享锁**](https://link.jianshu.com/?t=http://curator.apache.org/curator-recipes/shared-lock.html)

@Test

public void sharedLock() throws Exception {

*// 创建共享锁*

InterProcessLock lock = new InterProcessSemaphoreMutex(client, lockPath);

*// lock2 用于模拟其他客户端*

InterProcessLock lock2 = new InterProcessSemaphoreMutex(client2, lockPath);

*// 获取锁对象*

lock.acquire();

*// 测试是否可以重入*

*// 超时获取锁对象(第一个参数为时间, 第二个参数为时间单位), 因为锁已经被获取, 所以返回 false*

Assert.assertFalse(lock.acquire(2, TimeUnit.SECONDS));

*// 释放锁*

lock.release();

*// lock2 尝试获取锁成功, 因为锁已经被释放*

Assert.assertTrue(lock2.acquire(2, TimeUnit.SECONDS));

lock2.release();

}

[**共享可重入锁**](https://link.jianshu.com/?t=http://curator.apache.org/curator-recipes/shared-reentrant-lock.html)

public void sharedReentrantLock() throws Exception {

*// 创建可重入锁*

InterProcessLock lock = new InterProcessMutex(client, lockPath);

*// lock2 用于模拟其他客户端*

InterProcessLock lock2 = new InterProcessMutex(client2, lockPath);

*// lock 获取锁*

lock.acquire();

try {

*// lock 第二次获取锁*

lock.acquire();

try {

*// lock2 超时获取锁, 因为锁已经被 lock 客户端占用, 所以获取失败, 需要等 lock 释放*

Assert.assertFalse(lock2.acquire(2, TimeUnit.SECONDS));

} finally {

lock.release();

}

} finally {

*// 重入锁获取与释放需要一一对应, 如果获取 2 次, 释放 1 次, 那么该锁依然是被占用, 如果将下面这行代码注释, 那么会发现下面的 lock2 获取锁失败*

lock.release();

}

*// 在 lock 释放后, lock2 能够获取锁*

Assert.assertTrue(lock2.acquire(2, TimeUnit.SECONDS));

lock2.release();

}

[**共享可重入读写锁**](https://link.jianshu.com/?t=http://curator.apache.org/curator-recipes/shared-reentrant-read-write-lock.html)

@Test

public void sharedReentrantReadWriteLock() throws Exception {

*// 创建读写锁对象, Curator 以公平锁的方式进行实现*

InterProcessReadWriteLock lock = new InterProcessReadWriteLock(client, lockPath);

*// lock2 用于模拟其他客户端*

InterProcessReadWriteLock lock2 = new InterProcessReadWriteLock(client2, lockPath);

*// 使用 lock 模拟读操作*

*// 使用 lock2 模拟写操作*

*// 获取读锁(使用 InterProcessMutex 实现, 所以是可以重入的)*

InterProcessLock readLock = lock.readLock();

*// 获取写锁(使用 InterProcessMutex 实现, 所以是可以重入的)*

InterProcessLock writeLock = lock2.writeLock();

*/\*\**

*\* 读写锁测试对象*

*\*/*

class ReadWriteLockTest {

*// 测试数据变更字段*

private Integer testData = 0;

private Set<Thread> threadSet = new HashSet<>();

*// 写入数据*

private void write() throws Exception {

writeLock.acquire();

try {

Thread.sleep(10);

testData++;

System.out.println("写入数据 \ t" + testData);

} finally {

writeLock.release();

}

}

*// 读取数据*

private void read() throws Exception {

readLock.acquire();

try {

Thread.sleep(10);

System.out.println("读取数据 \ t" + testData);

} finally {

readLock.release();

}

}

*// 等待线程结束, 防止 test 方法调用完成后, 当前线程直接退出, 导致控制台无法输出信息*

public void waitThread() throws InterruptedException {

for (Thread thread : threadSet) {

thread.join();

}

}

*// 创建线程方法*

private void createThread(int type) {

Thread thread = new Thread(new Runnable() {

@Override

public void run() {

try {

if (type == 1) {

write();

} else {

read();

}

} catch (Exception e) {

e.printStackTrace();

}

}

});

threadSet.add(thread);

thread.start();

}

*// 测试方法*

public void test() {

for (int i = 0; i < 5; i++) {

createThread(1);

}

for (int i = 0; i < 5; i++) {

createThread(2);

}

}

}

ReadWriteLockTest readWriteLockTest = new ReadWriteLockTest();

readWriteLockTest.test();

readWriteLockTest.waitThread();

}

测试结果如下:

写入数据 1  
写入数据 2  
读取数据 2  
写入数据 3  
读取数据 3  
写入数据 4  
读取数据 4  
读取数据 4  
写入数据 5  
读取数据 5

读取数据线程总是能看到最新写入的数据

[**共享信号量**](https://link.jianshu.com/?t=http://curator.apache.org/curator-recipes/shared-semaphore.html)

@Test

public void semaphore() throws Exception {

*// 创建一个信号量, Curator 以公平锁的方式进行实现*

InterProcessSemaphoreV2 semaphore = new InterProcessSemaphoreV2(client, lockPath, 6);

*// semaphore2 用于模拟其他客户端*

InterProcessSemaphoreV2 semaphore2 = new InterProcessSemaphoreV2(client2, lockPath, 6);

*// 获取一个许可*

Lease lease = semaphore.acquire();

Assert.assertNotNull(lease);

*// semaphore.getParticipantNodes() 会返回当前参与信号量的节点列表, 俩个客户端所获取的信息相同*

Assert.assertEquals(semaphore.getParticipantNodes(), semaphore2.getParticipantNodes());

*// 超时获取一个许可*

Lease lease2 = semaphore2.acquire(2, TimeUnit.SECONDS);

Assert.assertNotNull(lease2);

Assert.assertEquals(semaphore.getParticipantNodes(), semaphore2.getParticipantNodes());

*// 获取多个许可, 参数为许可数量*

Collection<Lease> leases = semaphore.acquire(2);

Assert.assertTrue(leases.size() == 2);

Assert.assertEquals(semaphore.getParticipantNodes(), semaphore2.getParticipantNodes());

*// 超时获取多个许可, 第一个参数为许可数量*

Collection<Lease> leases2 = semaphore2.acquire(2, 2, TimeUnit.SECONDS);

Assert.assertTrue(leases2.size() == 2);

Assert.assertEquals(semaphore.getParticipantNodes(), semaphore2.getParticipantNodes());

*// 目前 semaphore 已经获取 3 个许可, semaphore2 也获取 3 个许可, 加起来为 6 个, 所以他们无法再进行许可获取*

Assert.assertNull(semaphore.acquire(2, TimeUnit.SECONDS));

Assert.assertNull(semaphore2.acquire(2, TimeUnit.SECONDS));

*// 释放一个许可*

semaphore.returnLease(lease);

semaphore2.returnLease(lease2);

*// 释放多个许可*

semaphore.returnAll(leases);

semaphore2.returnAll(leases2);

}

[**多重共享锁**](https://link.jianshu.com/?t=http://curator.apache.org/curator-recipes/multi-shared-lock.html)

@Test

public void multiLock() throws Exception {

*// 可重入锁*

InterProcessLock interProcessLock1 = new InterProcessMutex(client, lockPath);

*// 不可重入锁*

InterProcessLock interProcessLock2 = new InterProcessSemaphoreMutex(client2, lockPath);

*// 创建多重锁对象*

InterProcessLock lock = new InterProcessMultiLock(Arrays.asList(interProcessLock1, interProcessLock2));

*// 获取参数集合中的所有锁*

lock.acquire();

*// 因为存在一个不可重入锁, 所以整个 InterProcessMultiLock 不可重入*

Assert.assertFalse(lock.acquire(2, TimeUnit.SECONDS));

*// interProcessLock1 是可重入锁, 所以可以继续获取锁*

Assert.assertTrue(interProcessLock1.acquire(2, TimeUnit.SECONDS));

*// interProcessLock2 是不可重入锁, 所以获取锁失败*

Assert.assertFalse(interProcessLock2.acquire(2, TimeUnit.SECONDS));

*// 释放参数集合中的所有锁*

lock.release();

*// interProcessLock2 中的锁已经释放, 所以可以获取*

Assert.assertTrue(interProcessLock2.acquire(2, TimeUnit.SECONDS));

}

**Curator 分布式锁解决的问题**

分布式锁服务宕机, ZooKeeper 一般是以集群部署, 如果出现 ZooKeeper 宕机, 那么只要当前正常的服务器超过集群的半数, 依然可以正常提供服务  
持有锁资源服务器宕机, 假如一台服务器获取锁之后就宕机了, 那么就会导致其他服务器无法再获取该锁. 就会造成 **死锁** 问题, 在 Curator 中, 锁的信息都是保存在临时节点上, 如果持有锁资源的服务器宕机, 那么 ZooKeeper 就会移除它的信息, 这时其他服务器就能进行获取锁操作

**便捷测试**

为了测试上面的代码, 我们需要下载、安装、启动一个 ZooKeeper 服务, 然后将该服务地址配置为 connectString. 如果更换环境的话又需要重新安装, 未免麻烦了点. Curator 为我们提供一个专门用于开发、测试的便捷方法, 让我们更加专注于编写与 ZooKeeper 相关的程序.  
首先需要导入 curator-test 测试包

<dependency>

<groupId>org.apache.curator</groupId>

<artifactId>curator-test</artifactId>

<version>4.0.0</version>

<scope>test</scope>

</dependency>

在这个包中为我们提供了一个 TestingServer 类, 主要用法如下  
构造方法有多个, 但是主要使用到的有这两个

TestingServer()

TestingServer(int port, File tempDirectory)

port 为端口  
tempDirectory 为临时的 dataDir 目录  
如果调用 TestingServer() 方法构造, 会获取一个空闲端口, 同时在 java.io.tmpdir 创建一个临时目录当作本次的 dataDir 目录  
然后使用以下方法创建客户端

TestingServer server=new TestingServer();

*// server.getConnectString() 方法会返回可用的服务链接地址, 如: 127.0.0.1:2181*

CuratorFramework client=CuratorFrameworkFactory.newClient(server.getConnectString(), retry);

另外在测试完成记得进行资源释放

@After

public void close() {

CloseableUtils.closeQuietly(client);

CloseableUtils.closeQuietly(server);

}

TestingServer 能为我们简单的启动一个 ZooKeeper 服务器, 但是如果需要进行集群测试呢?  
这个时候我们可以使用 TestingCluster 启动 ZooKeeper 集群  
TestingCluster 同样提供多个构造器, 但是主要使用以下两个

TestingCluster(int instanceQty)

TestingCluster(InstanceSpec... specs)

instanceQty 是集群的数量  
specs 是 InstanceSpec 的变长参数  
InstanceSpec 的创建方法可以参考 TestingServer 的构造方法实现  
然后创建客户端使用以下方法

TestingCluster server=new TestingCluster(3);

*// server.getConnectString() 方法会返回可用的服务链接地址, 如: 127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183*

CuratorFramework client=CuratorFrameworkFactory.newClient(server.getConnectString(), retry);

同样请记得释放资源

[测试源码](https://link.jianshu.com/?t=https://github.com/ghthou/learning-distributed-lock)

|  |
| --- |
| https://github.com/ghthou/learning-distributed-lock |