**ApacheCurator“菜谱”下的高级特性:分布式锁与Leader选举**

2016-12-15 09:24:00      [**0**](javascript:void(0);)[个评论](https://www.2cto.com/net/201612/574874.html#SOHUCS)    来源：[tengxing007](http://blog.csdn.net/tengxing007)

[收藏](javascript:;)   [我要投稿](https://www.2cto.com/index.php?m=member&c=content&a=publish&modid=1&siteid=1)

**ApacheCurator“菜谱”下的高级特性:分布式锁与Leader选举：**网上说Curator框架是最好用，最流行的zookeeper的客户端。  
  
优点:

提供了一套非常友好的操作API; 提供一些高级特性（包括但不仅限于前篇文章中提到的）的封装 易测试

基本的API见前一篇  
https://blog.csdn.net/tengxing007/article/details/53572224  
为什么说“菜谱”？

[?](https://www.2cto.com/net/201612/574874.html)

<dependency>

    <groupid>org.apache.curator</groupid>

    curator-recipes</artifactid>

    <version>2.7.1</version>

</dependency>

“菜谱”如下：  
- 锁：包括共享锁、共享可重入锁、读写锁等。  
- 选举：Leader选举算法。  
- Barrier：阻止分布式计算直至某个条件被满足的“栅栏”，可以看做JDK Concurrent包中Barrier的分布式实现。  
- 缓存：见前一篇https://blog.csdn.net/tengxing007/article/details/53572224  
- 持久化结点：连接或Session终止后仍然在Zookeeper中存在的结点。  
这里主要学习分布式锁与Leader选举

**分布式锁**

说到锁其实我并不了解，应为以前使用spring开发，没有太在意这个。今天取看了看java 锁，先记一下。  
一段synchronized的代码被一个线程执行之前，他要先拿到执行这段代码的权限，在[Java](https://www.2cto.com/kf/ware/Java/)里边就是拿到某个同步对象的锁（一个对象只有一把锁）； 如果这个时候同步对象的锁被其他线程拿走了，他（这个线程）就只能等了（线程阻塞在锁池等待队列中）。 取到锁后，他就开始执行同步代码(被synchronized修饰的代码）；线程执行完同步代码后马上就把锁还给同步对象，其他在锁池中等待的某个线程就可以拿到锁执行同步代码了。这样就保证了同步代码在统一时刻只有一个线程在执行。  
简单点说：我们来举一个Dirty的例子：某餐厅的卫生间很小，几乎只能容纳一个人如厕。为了保证不受干扰，如厕的人进入卫生间，就要锁上房门。我们可以把卫生间想 象成是共享的资源，而众多需要如厕的人可以被视作多个线程。假如卫生间当前有人占用，那么其他人必须等待，直到这个人如厕完毕，打开房门走出来为止。这就 好比多个线程共享一个资源的时候，是一定要分出先来后到的。

有人说：那如果我没有这道门会怎样呢？让两个线程相互竞争，谁抢先了，谁就可以先干活，这样多好阿？但是我们知道：如果厕所没有门的话，如厕的人一起涌向 厕所，那么必然会发生争执，正常的如厕步骤就会被打乱，很有可能会发生意想不到的结果，例如某些人可能只好被迫在不正确的地方施肥……

正是因为有这道门，任何一个单独进入如厕的人都可以顺利的完成他们的如厕过程，而不会被干扰，甚至发生以外的结果。这就是说，如厕的时候要讲究先来后到。  
所以说：***多线程的线程同步机制实际上是靠锁来实现的***  
详情见另外一篇：  
——————————  
进入正题：  
下面的程序会启动两个线程t1和t2去争夺锁，拿到锁的线程会占用5秒。运行多次可以观察到，有时是t1先拿到锁而t2等待，有时又会反过来。Curator会用我们提供的lock路径的结点作为全局锁，这个结点的数据类似这种格式：[\_c\_64e0811f-9475-44ca-aa36-c1db65ae5350-lock-0000000005]，每次获得锁时会生成这种串，释放锁时清空数据。

[?](https://www.2cto.com/net/201612/574874.html)

import org.apache.curator.framework.CuratorFramework;

import org.apache.curator.framework.CuratorFrameworkFactory;

import org.apache.curator.framework.recipes.locks.InterProcessMultiLock;

import org.apache.curator.framework.recipes.locks.InterProcessMutex;

import org.apache.curator.retry.RetryNTimes;

import java.util.concurrent.TimeUnit;

/\*\*

 \* 锁测试

 \* Created by tengxing on 16-12-11.

 \*/

public class CuratorDistrLockDemo {

    /\*\*

     \* Zookeeper info

     \*/

    private static final String ZK\_ADDRESS = "192.168.1.100:2181";

    private static final String ZK\_LOCK\_PATH = "/tengxing";

    public static void main(String []args){

       /\* 1.connect to zk\*/

        CuratorFramework client= CuratorFrameworkFactory.newClient(

                ZK\_ADDRESS,

                new RetryNTimes(10,5000)

        );

        client.start();

        System.out.println("zk client start successfully!");

        Thread t1=new Thread(() ->{

            doWithLock(client);

        },"t1");

        Thread t2=new Thread(()->{

            doWithLock(client);

        },"t2");

    }

    private static void doWithLock(CuratorFramework client){

        InterProcessMutex lock = new InterProcessMutex(client,ZK\_LOCK\_PATH);

        try{

            if (lock.acquire(10 \* 1000,TimeUnit.SECONDS)) {

                System.out.println(Thread.currentThread().getName() + " hold lock");

                Thread.sleep(5000L);

                System.out.println(Thread.currentThread().getName() + " release lock");

            }

        }catch (Exception e){

            e.printStackTrace();

        }finally {

            try {

                lock.release();

            } catch (Exception e) {

                e.printStackTrace();

            }

        }

    }

}

**Leader选举**

当集群里的某个服务down机时，我们可能要从slave结点里选出一个作为新的master，这时就需要一套能在分布式环境中自动协调的Leader选举方法。Curator提供了LeaderSelector监听器实现Leader选举功能。同一时刻，只有一个Listener会进入takeLeadership()方法，说明它是当前的Leader。注意：当Listener从takeLeadership()退出时就说明它放弃了“Leader身份”，这时Curator会利用Zookeeper再从剩余的Listener中选出一个新的Leader。autoRequeue()方法使放弃Leadership的Listener有机会重新获得Leadership，如果不设置的话放弃了的Listener是不会再变成Leader的。

[?](https://www.2cto.com/net/201612/574874.html)

import org.apache.curator.framework.CuratorFramework;

import org.apache.curator.framework.CuratorFrameworkFactory;

import org.apache.curator.framework.recipes.leader.LeaderSelector;

import org.apache.curator.framework.recipes.leader.LeaderSelectorListener;

import org.apache.curator.framework.state.ConnectionState;

import org.apache.curator.retry.RetryNTimes;

import org.apache.curator.utils.EnsurePath;

/\*\*

 \* Curator framework's leader election test.

 \* Output:

 \*  LeaderSelector-2 take leadership!

 \*  LeaderSelector-2 relinquish leadership!

 \*  LeaderSelector-1 take leadership!

 \*  LeaderSelector-1 relinquish leadership!

 \*  LeaderSelector-0 take leadership!

 \*  LeaderSelector-0 relinquish leadership!

 \*      ...

 \*/

public class CuratorLeaderTest {

    /\*\* Zookeeper info \*/

    private static final String ZK\_ADDRESS = "192.168.1.100:2181";

    private static final String ZK\_PATH = "/zktest";

    public static void main(String[] args) throws InterruptedException {

        LeaderSelectorListener listener = new LeaderSelectorListener() {

            @Override

            public void takeLeadership(CuratorFramework client) throws Exception {

                System.out.println(Thread.currentThread().getName() + " take leadership!");

                // takeLeadership() method should only return when leadership is being relinquished.

                Thread.sleep(5000L);

                System.out.println(Thread.currentThread().getName() + " relinquish leadership!");

            }

            @Override

            public void stateChanged(CuratorFramework client, ConnectionState state) {

            }

        };

        new Thread(() -> {

            registerListener(listener);

        }).start();

        new Thread(() -> {

            registerListener(listener);

        }).start();

        new Thread(() -> {

            registerListener(listener);

        }).start();

        Thread.sleep(Integer.MAX\_VALUE);

    }

    private static void registerListener(LeaderSelectorListener listener) {

        // 1.Connect to zk

        CuratorFramework client = CuratorFrameworkFactory.newClient(

                ZK\_ADDRESS,

                new RetryNTimes(10, 5000)

        );

        client.start();

        // 2.Ensure path

        try {

            new EnsurePath(ZK\_PATH).ensure(client.getZookeeperClient());

        } catch (Exception e) {

            e.printStackTrace();

        }

        // 3.Register listener

        LeaderSelector selector = new LeaderSelector(client, ZK\_PATH, listener);

        selector.autoRequeue();

        selector.start();

    }

}