# MemCache作为分布式锁的实际运用

## Jar包引用

|  |
| --- |
| <dependency>  <groupId>com.googlecode.xmemcached</groupId>  <artifactId>xmemcached</artifactId>  <version>2.4.6</version>  </dependency> |

## 接口类

|  |
| --- |
| package start.memcache.lock.tool;  public interface DistributedLock {  /\*\*  \* Try distributed lock, only returns true is get lock success  \*  \* @param lockExpireInSeconds Lock expire time in seconds  \* @return result  \* @throws DistributedLockException  \*/  boolean tryLock(int lockExpireInSeconds) throws DistributedLockException;  /\*\*  \* Try distributed lock, only returns true is get lock success  \*  \* @return result  \* @throws DistributedLockException  \*/  boolean tryLock() throws DistributedLockException;  /\*\*  \* Release distributed lock  \*/  void unLock();  } |

## 工厂类

|  |
| --- |
| package start.memcache.lock.tool;  public interface DistributedLockFactory {  DistributedLock createDistributedLock(String lockId);  } |

## 工厂实现类

|  |
| --- |
| package start.memcache.lock.tool.impl.factory;  import start.memcache.lock.tool.DistributedLock;  import start.memcache.lock.tool.DistributedLockException;  import start.memcache.lock.tool.DistributedLockFactory;  import net.rubyeye.xmemcached.MemcachedClient;  /\*\*  \* XMemcached distributed lock factory implement  \*/  public class XMemcachedDistributedLockFactory implements DistributedLockFactory {  private static final String XMEMCACHED\_DISTRIBUTED\_LOCK\_FACTORY\_PREFIX = "\_\_xmemcached\_distributed\_lock\_factory\_prefix:";  private static final String XMEMCACHED\_DISTRIBUTED\_LOCK\_FACTORY\_DEFAULT\_VALUE = "x";  private int defaultLockExpireInSeconds = 10 \* 60;  private MemcachedClient client;  public XMemcachedDistributedLockFactory(MemcachedClient client) {  this.client = client;  }  public XMemcachedDistributedLockFactory(MemcachedClient client, int defaultLockExpireInSeconds) {  this.client = client;  this.defaultLockExpireInSeconds = defaultLockExpireInSeconds;  }  @Override  public DistributedLock createDistributedLock(String lockId) {  final String theLockId = XMEMCACHED\_DISTRIBUTED\_LOCK\_FACTORY\_PREFIX + lockId;  return new DistributedLock() {  @Override  public boolean tryLock(int lockExpireInSeconds) {  try {  return client.add(theLockId, lockExpireInSeconds, XMEMCACHED\_DISTRIBUTED\_LOCK\_FACTORY\_DEFAULT\_VALUE);  } catch (Exception e) {  e.printStackTrace();  return false;  }  }  @Override  public boolean tryLock() throws DistributedLockException {  return tryLock(defaultLockExpireInSeconds);  }  @Override  public void unLock() {  try {  client.delete(theLockId);  } catch (Exception e) {  e.printStackTrace();  }  }  };  }  } |

## 测试类

|  |
| --- |
| package study2021.memcachelock;  import lombok.extern.slf4j.Slf4j;  import net.rubyeye.xmemcached.MemcachedClient;  import net.rubyeye.xmemcached.XMemcachedClient;  import start.memcache.lock.tool.DistributedLock;  import start.memcache.lock.tool.DistributedLockFactory;  import start.memcache.lock.tool.impl.factory.XMemcachedDistributedLockFactory;  import java.util.concurrent.ExecutorService;  import java.util.concurrent.Executors;  /\*\*  \* 前提：本地启动MemCache服务  \* 教程安装包：https://www.runoob.com/memcached/window-install-memcached.html  \* 使用管理员权限执行以下命令，install安装命令执行一次即可  \* 安装命令：D:\memcache\memcached-win64-1.4.4-14\memcached\memcached.exe -d install  \* 启动命令：D:\memcache\memcached-win64-1.4.4-14\memcached\memcached.exe -d start  \* 停止命令：D:\memcache\memcached-win64-1.4.4-14\memcached\memcached.exe -d stop  \*/  @Slf4j  public class XMemcachedDistributedLockTest {  public static final String MEM\_CACHE\_ADDRESS = "127.0.0.1";  public static final int MEM\_CACHE\_PORT = 11211;  public static void main(String[] args) throws Exception {  // 默认本机端口：11211  MemcachedClient client = new XMemcachedClient(MEM\_CACHE\_ADDRESS, MEM\_CACHE\_PORT);  // 分布式锁工厂类  DistributedLockFactory factory = new XMemcachedDistributedLockFactory(client);  // 定义线程池  ExecutorService executorService = Executors.newFixedThreadPool(5);  // 循环线程  for (int i = 0; i < 5; i++) {  log.debug("Enter for loop");  executorService.submit(() -> {  DistributedLock distributedLock = factory.createDistributedLock("a");  while (true) {  System.out.println("while loop");  if (distributedLock.tryLock()) {  System.out.println("lock success in :" + System.currentTimeMillis());  try {  method();  Thread.sleep(10L);  } catch (InterruptedException e) {  e.printStackTrace();  } finally {  System.out.println("lock success out:" + System.currentTimeMillis());  distributedLock.unLock();  break;  }  } else {  try {  Thread.sleep(1000L);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  });  }  executorService.shutdown();  }  /\*\*  \* 执行业务逻辑  \*/  private static void method() {  // TODO  System.out.println("执行业务逻辑");  }  } |