## Jedis使用总结

**作者：** [Teaey](mailto:masfay@163.com)

**微博：**<http://weibo.com/teaey>

前段时间细节的了解了Jedis的使用，Jedis是redis的java版本的客户端实现。  
本文做个总结，主要分享如下内容：

【pipeline】【分布式的id生成器】【分布式锁【watch】【multi】】【redis分布式】  
好了，一个一个来。  
一、 Pipeline  
官方的说明是：starts a pipeline,which is a very efficient way to send lots of command and read all the responses when you finish sending them。简单点说pipeline适用于批处理。当有大量的操作需要一次性执行的时候，可以用管道。  
示例：

Jedis jedis = new Jedis(String, int);  
Pipeline p = jedis.pipelined();  
p.set(key,value);//每个操作都发送请求给redis-server  
p.get(key,value);  
http://www.blogjava.net/Images/dot.gif  
p.sync();//这段代码获取所有的response

这里我进行了20w次连续操作（10w读，10w写），不用pipeline耗时：187242ms，用pipeline耗时：1188ms，可见使用管道后的性能上了一个台阶。看了代码了解到，管道通过一次性写入请求，然后一次性读取响应。也就是说jedis是：request response，request response，...；pipeline则是：request request... response response的方式。这样无需每次请求都等待server端的响应。  
  
二、 跨jvm的id生成器   
谈到这个话题，首先要知道redis-server端是单线程来处理client端的请求的。  
这样来实现一个id生成器就非常简单了，只要简单的调用jdeis.incr(key);就搞定了。  
你或许会问，incr是原子操作吗，能保证不会出现并发问题吗，不是说了吗，server端是单线程处理请求的。  
  
三、 【跨jvm的锁实现【watch】【multi】】  
首先说下这个问题的使用场景，有些时候我们业务逻辑是在不同的jvm进程甚至是不同的物理机上的jvm处理的。这样如何来实现不同jvm上的同步问题呢，其实我们可以基于redis来实现一个锁。  
具体事务和监听请参考文章：[**redis学习笔记之事务**](http://www.cnblogs.com/xhan/archive/2011/02/04/1949151.html)  
 暂时找到三种实现方式：  
1. 通过jedis.setnx(key,value)实现  
     import java.util.Random;

import org.apache.commons.pool.impl.GenericObjectPool.Config;  
  
import redis.clients.jedis.Jedis;  
import redis.clients.jedis.JedisPool;  
import redis.clients.jedis.Transaction;  
  
/\*\*  
 \* @author Teaey  
 \*/  
public class RedisLock {  
    //加锁标志  
    public static final String LOCKED = "TRUE";  
    public static final long ONE\_MILLI\_NANOS = 1000000L;  
    //默认超时时间（毫秒）  
    public static final long DEFAULT\_TIME\_OUT = 3000;  
    public static JedisPool pool;  
    public static final Random r = new Random();  
    //锁的超时时间（秒），过期删除  
    public static final int EXPIRE = 5 \* 60;  
    static {  
        pool = new JedisPool(new Config(), "host", 6379);  
    }  
    private Jedis jedis;  
    private String key;  
    //锁状态标志  
    private boolean locked = false;  
  
    public RedisLock(String key) {  
        this.key = key;  
        this.jedis = pool.getResource();  
    }  
  
    public boolean lock(long timeout) {  
        long nano = System.nanoTime();  
        timeout \*= ONE\_MILLI\_NANOS;  
        try {  
            while ((System.nanoTime() - nano) < timeout) {  
                if (jedis.setnx(key, LOCKED) == 1) {  
                    jedis.expire(key, EXPIRE);  
                    locked = true;  
                    return locked;  
                }  
                // 短暂休眠，nano避免出现活锁  
                Thread.sleep(3, r.nextInt(500));  
            }  
        } catch (Exception e) {  
        }  
        return false;  
    }  
    public boolean lock() {  
        return lock(DEFAULT\_TIME\_OUT);  
    }  
  
    // 无论是否加锁成功，必须调用  
    public void unlock() {  
        try {  
            if (locked)  
                jedis.del(key);  
        } finally {  
            pool.returnResource(jedis);  
        }  
    }  
}

2. 通过事务(multi)实现  
由于采纳第一张方法，第二种跟第三种实现只贴了关键代码，望谅解。^\_^  
     public boolean lock\_2(long timeout) {

        long nano = System.nanoTime();  
        timeout \*= ONE\_MILLI\_NANOS;  
        try {  
            while ((System.nanoTime() - nano) < timeout) {  
                Transaction t = jedis.multi();  
                // 开启事务，当server端收到multi指令  
                // 会将该client的命令放入一个队列，然后依次执行，知道收到exec指令  
                t.getSet(key, LOCKED);  
                t.expire(key, EXPIRE);  
                String ret = (String) t.exec().get(0);  
                if (ret == null || ret.equals("UNLOCK")) {  
                    return true;  
                }  
                // 短暂休眠，nano避免出现活锁  
                Thread.sleep(3, r.nextInt(500));  
            }  
        } catch (Exception e) {  
        }  
        return false;  
    }

3. 通过事务+监听实现  
    public boolean lock\_3(long timeout) {

        long nano = System.nanoTime();  
        timeout \*= ONE\_MILLI\_NANOS;  
        try {  
            while ((System.nanoTime() - nano) < timeout) {  
                jedis.watch(key);  
                // 开启watch之后，如果key的值被修改，则事务失败，exec方法返回null  
                String value = jedis.get(key);  
                if (value == null || value.equals("UNLOCK")) {  
                    Transaction t = jedis.multi();  
                    t.setex(key, EXPIRE, LOCKED);  
                    if (t.exec() != null) {  
                        return true;  
                    }  
                }  
                jedis.unwatch();  
                // 短暂休眠，nano避免出现活锁  
                Thread.sleep(3, r.nextInt(500));  
            }  
        } catch (Exception e) {  
        }  
        return false;  
    }

最终采用第一种实现，因为加锁只需发送一个请求，效率最高。  
四、 【redis分布式】  
    最后一个话题，jedis的分布式。在jedis的源码里发现了两种hash算法（MD5，MURMUR Hash(默认）），也可以自己实现redis.clients.util.Hashing接口扩展。  
  List<JedisShardInfo> hosts = new ArrayList<JedisShardInfo>();

        //server1  
        JedisShardInfo host1 = new JedisShardInfo("", 6380, 2000);  
        //server2  
        JedisShardInfo host2 = new JedisShardInfo("", 6381, 2000);  
        hosts.add(host1);  
        hosts.add(host2);  
        ShardedJedis jedis = new ShardedJedis(hosts);  
        jedis.set("key", "");

另外写博客真费力。。。尤其是demo不在外网机