jedis超时重试机制注意事项

jedis客户端在创建连接时会设置一个超时，并且会有重试机制。

问题起源

在使用jedis客户端的时候，我测试了一下”incr”命令，该命令在执行过程中是原子的，所以理论上不会出现问题（不论单线程还是多线程）

代码如下：

/\*\*

\* incrf:

\* 将 key 中储存的数字值增一。

如果 key 不存在，那么 key 的值会先被初始化为 0 ，然后再执行 INCR 操作。

如果值包含错误的类型，或字符串类型的值不能表示为数字，那么返回一个错误。

本操作的值限制在 64 位(bit)有符号数字表示之内。

这是一个针对字符串的操作，因为 Redis 没有专用的整数类型，所以 key 内储存的字符串被解释为十进制 64 位有符号整数来执行 INCR 操作。

返回值： 执行 INCR 命令之后 key 的值。

这里有问题，最终数据结果大于10000

这是因为设置的超时时间太小了，他去重试了，所以最终结果大于10000

\*/

@Test

public void incrTest() throws InterruptedException {

/\*\*

\* 测试线程安全

\*/

jedisCluster.del("incrNum");

final AtomicInteger atomicInteger = new AtomicInteger(0);

final CountDownLatch countDownLatch = new CountDownLatch(10);

ExecutorService executorService = Executors.newFixedThreadPool(10);

for(int i = 0 ; i < 10 ; i ++){

executorService.submit(new Runnable() {

@Override

public void run() {

//每个线程增加1000次，每次加1

for(int j = 0 ; j < 1000 ; j ++){

atomicInteger.incrementAndGet();

jedisCluster.incr("incrNum");

}

countDownLatch.countDown();

}

});

}

countDownLatch.await();

System.out.println(jedisCluster.get("incrNum"));

System.out.println(atomicInteger);

}

这是个多线程的demo，不管多线程还是单线程，我最终得到的结果都是比10000大。这就奇怪了，”incr”是原子的，理论上不应该会这样。我起初以为是jedis客户端的BUG，后来仔细一看才发现，这不是BUG，是我在创建jedisCluster的时候设置的超时时间太短了，导致其超时重试。

我使用的是这个构造方法：

public JedisCluster(Set<HostAndPort> jedisClusterNode, int connectionTimeout, int soTimeout,

int maxAttempts, String password, final GenericObjectPoolConfig poolConfig) {

super(jedisClusterNode, connectionTimeout, soTimeout, maxAttempts, password, poolConfig);

}

* soTimeout：返回值的超时时间
* maxAttempts: 出现异常最大重试次数

解决办法

将超时时间设置得大一些。

我们在使用这些开源框架的时候，一定要全面了解其运作原理，这样才能事半功倍。另外需要注意，像httpclient、dubbo等RPC框架都会有超时重试机制，在使用的时候要注意。