Redis构建分布式锁

构建合适的锁可以在高并发下能够保持数据的一致性，即客户端在执行连贯的命令时上锁的数据不会被别的客户端的更改而发生错误。同时还能够保证命令执行的成功率。

事务：在redis中，通过特殊的命令MULTI为开始，之后用户传入一连贯的命令，最后EXEC为结束。进一步分析，redis事务中的命令会先推入队列，等到EXEC命令出现的时候才会将一条条命令执行。假若watch监控的key发生改变，这个事务将会失败。这也就说明Redis事务中不存在锁，其他客户端可以修改正在执行事务中的有关数据，这也就为什么在多个客户端同时处理相同的数据时事务往往会发生错误。

并发测试：

1、在redis中设置一个字符串count，运用程序将其取出来加+1，再存储回去，一直循环十万次

2、在两个浏览器上同时执行这个代码

3、将count取出来，查看结果

|  |
| --- |
| test.php文件  <?php  $redis=new Redis();  $redis->connect('127.0.0.1','6379');  for ($i=0; $i < 100000; $i++)  {  $count=$redis->get('count');  $count=$count+1;  $redis->set('count',$count);  }  echo $redis->get('count');  ?> |

最终结果：112826

总共执行两次，count原本应该是二十万才对的，但实际上count等于十三万多，远远小于二十万，这是为什么呢？

原因：由前面的内容可知，redis是采用单线程IO多路复用模型的。因此我们使用两个浏览器即为两个会话（A、B），取出、加1、存入这三个命令并不是原子操作，并且在执行取出、存入这两个redis命令时是哪个客户端先到就先执行。

例如：

1、此时count=120

2、A取出count=120，紧接着B的取出命令流到了，也将count=120取出

3、A取出后立即加1，并将count=121存回去

4、此时B也紧跟着，也将count=121存进去了

注意：

1、设置循环次数尽量大一点，太小的话，当在第一个浏览器执行完毕，第二个浏览器还没开始进行呢

2、必须要两个浏览器同时执行。假若在一个浏览器中同时执行两次test.php文件，不管是否同时执行，最终结果就是count=200000。因为在同一个浏览器中执行，都是属于同一个会话（所有命令都在同一个通道通过），所以redis会让先执行的十万次执行完，再接着执行其他的十万次。

解决：

1. 事务（不可行）

|  |
| --- |
| for ($i=0; $i < 100000; $i++)  {  $redis->multi();  $count=$redis->get('count');  $count=$count+1;  $redis->set('count',$count);  $redis->exec();  }  echo $redis->get('count'); |

结果出错：我们都知道当redis开启时，事务中的命令是不执行的，而是先将命令压入队列，然后当出现exec命令的时候，才会阻塞式的将所有的命令一个接一个的执行。

所以当使用PHP中的Redis类进行redis事务的时候，所有有关redis的命令都不会真正的执行，而仅仅是将命令发送到redis中进行存储起来。

所以上面绿色的$count是个对象，并不是inter值；

1. 原子性操作incr

|  |
| --- |
| for ($i=0; $i < 100000; $i++)  {  $count=$redis->incr('count');  }  echo $redis->get('count'); |

可以解决，但实际生产中逻辑比incr复杂，不通用；

1. 构建分布式锁

我们构造锁的目的就是在高并发下消除选择竞争、保持数据一致性

构造锁的时候，我们需要注意几个问题：

1、预防处理持有锁在执行操作的时候进程奔溃，导致死锁，其他进程一直得不到此锁

2、持有锁进程因为操作时间长而导致锁自动释放，但本身进程并不知道，最后错误的释放其他进程的锁

3、一个进程锁过期后，其他多个进程同时尝试获取锁，并且都成功获得锁

代码见：RedisLock.class.php