WATCH乐观锁

只会在数据被其他客户端抢先修改的情况下通知执行了这个命令的客户端，而不会阻止其他客户端对数据进行修改

一般用 watch、multi、exec结构包裹

SETNX悲观锁

这个命令只有在键不存在的情况下为键设置值，将一个随机生成的值设置为键的值，并使用这个值防止锁被其他进程取得。

程序尝试获取锁失败后会不断重试，直到成功或者超过给定时限

细粒度锁

在商品交易时前两种锁锁住的是整个市场，而我们关心的是某件商品，通过只锁住商品二不是整个市场，可以减少锁竞争，提升程序性能

为了给锁加上超时限制特性，程序将在取得锁之后，调用EXPIRE命令来为锁设置过期时间，使得Redis可以自动删除超时的锁。为了确保锁在客户端已经崩溃（在执行SETNX和EXPIRE之间崩溃最极端）的情况下仍然能够自动释放，客户端在尝试获取锁失败后，检查锁的超时时间，并为未设置超时时间的锁设置超时时间。Redis2.6.12版本后 直接 使用SET参数设置 过期时间 set key value NX EX ‘时间’

锁不正确运行的原因

1. 持有锁的进程因为操作时间长而导致锁被自动释放，但进程不知晓，甚至还释放掉其他进程持有的锁
2. 一个持有锁并打算执行长时间操作的进程已经崩溃，但其他想获取锁的进程无法检测，白白浪费时间等待锁被释放
3. 在一个进程持有锁过期后，其他多个进程同时获取锁，并且都获得了锁

ThinkPHP6 测试代码：

<?php

/\*\*

\* +----------------------------------------------------------------------

\* | Redis 互斥锁

\* +----------------------------------------------------------------------

\*/

namespace app\common;

use think\facade\Cache;

class RedisLock {

private $\_redis;

public function \_\_construct()

{

//获取redis handler资源句柄

$this->\_redis = Cache::handler();

}

/\*\*

\* 获取锁

\* @param String $key 锁标识

\* @param Int $expire 锁过期时间

\* @param Int $random 随机值

\* @return Boolean

\*/

public function acquire\_lock($key, $random, $expire)

{

//设置锁的超时时间，避免释放锁失败，del()操作失败，产生死锁。

$ret = $this->\_redis->set($key, $random, ['nx', 'ex' => $expire]);

return $ret;

}

/\*\*

\* 释放锁

\* @param String $key 锁标识

\* @param Int $random 随机值

\* @return Boolean

\*/

public function release\_lock($key, $random)

{

//防止操作时间过长，超过了锁的有效时间，导致其他请求拿到了锁,误删其他请求创建的锁

if ($this->\_redis->get($key) == $random) {

return $this->\_redis->del($key);

}else {

return 2;

}

}

}