**[PHP商品秒杀问题解决方案实例详解【mysql与redis】](https://www.cnblogs.com/freespider/p/11324048.html)**

本文实例讲述了PHP商品秒杀问题解决方案。分享给大家供大家参考，具体如下：

**引言**

假设num是存储在数据库中的字段，保存了被秒杀产品的剩余数量。

|  |  |
| --- | --- |
| 1  2  3  4 | if($num > 0){    //用户抢购成功，记录用户信息    $num--;  } |

假设在一个并发量较高的场景，数据库中num的值为1时，可能同时会有多个进程读取到num为1，程序判断符合条件，抢购成功，num减一。这样会导致商品超发的情况，本来只有10件可以抢购的商品，可能会有超过10个人抢到，此时num在抢购完成之后为负值。

解决该问题的方案由很多，可以简单分为基于mysql和redis的解决方案，redis的性能要由于mysql，因此可以承载更高的并发量，不过下面介绍的方案都是基于单台mysql和redis的，更高的并发量需要分布式的解决方案，本文没有涉及。

**基于mysql的解决方案**

商品表 goods

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | CREATE TABLE `goods` (   `id` int(11) NOT NULL,   `num` int(11) DEFAULT NULL,   `version` int(11) DEFAULT NULL,   PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 |

抢购结果表 log

|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE TABLE `log` (   `id` int(11) NOT NULL AUTO\_INCREMENT,   `good\_id` int(11) DEFAULT NULL,   PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 |

**悲观锁**

悲观锁的方案采用的是排他读，也就是同时只能有一个进程读取到num的值。事务在提交或回滚之后，锁会释放，其他的进程才能读取。该方案最简单易懂，在对性能要求不高时，可以直接采用该方案。要注意的是，SELECT … FOR UPDATE要尽可能的使用索引，以便锁定尽可能少的行数；排他锁是在事务执行结束之后才释放的，不是读取完成之后就释放，因此使用的事务应该尽可能的早些提交或回滚，以便早些释放排它锁。

<?php  
$this->mysqli->begin\_transaction();  
$result = $this->mysqli->query("SELECT num FROM goods WHERE id=1 LIMIT 1 FOR UPDATE");  
$row = $result->fetch\_assoc();  
$num = intval($row['num']);  
if($num > 0){  
 usleep(100);  
 $this->mysqli->query("UPDATE goods SET num=num-1");  
 $affected\_rows = $this->mysqli->affected\_rows;  
 if($affected\_rows == 1){  
 $this->mysqli->query("INSERT INTO log(good\_id) VALUES({$num})");  
 $affected\_rows = $this->mysqli->affected\_rows;  
 if($affected\_rows == 1){  
 $this->mysqli->commit();  
 echo "success:".$num;  
 }else{  
 $this->mysqli->rollback();  
 echo "fail1:".$num;  
 }  
 }else{  
 $this->mysqli->rollback();  
 echo "fail2:".$num;  
 }  
}else{  
 $this->mysqli->commit();  
 echo "fail3:".$num;  
}

**乐观锁**

乐观锁的方案在读取数据是并没有加排他锁，而是通过一个每次更新都会自增的version字段来解决，多个进程读取到相同num，然后都能更新成功的问题。在每个进程读取num的同时，也读取version的值，并且在更新num的同时也更新version，并在更新时加上对version的等值判断。假设有10个进程都读取到了num的值为1，version值为9，则这10个进程执行的更新语句都是UPDATE goods SET num=num-1,version=version+1 WHERE version=9，然而当其中一个进程执行成功之后，数据库中version的值就会变为10，剩余的9个进程都不会执行成功，这样保证了商品不会超发，num的值不会小于0，但这也导致了一个问题，那就是发出抢购请求较早的用户可能抢不到，反而被后来的请求抢到了。

<?php  
$result = $this->mysqli->query("SELECT num,version FROM goods WHERE id=1 LIMIT 1");  
$row = $result->fetch\_assoc();  
$num = intval($row['num']);  
$version = intval($row['version']);  
if($num > 0){  
 usleep(100);  
 $this->mysqli->begin\_transaction();  
 $this->mysqli->query("UPDATE goods SET num=num-1,version=version+1 WHERE version={$version}");  
 $affected\_rows = $this->mysqli->affected\_rows;  
 if($affected\_rows == 1){  
 $this->mysqli->query("INSERT INTO log(good\_id) VALUES({$num})");  
 $affected\_rows = $this->mysqli->affected\_rows;  
 if($affected\_rows == 1){  
 $this->mysqli->commit();  
 echo "success:".$num;  
 }else{  
 $this->mysqli->rollback();  
 echo "fail1:".$num;  
 }  
 }else{  
 $this->mysqli->rollback();  
 echo "fail2:".$num;  
 }  
}else{  
 echo "fail3:".$num;  
}

**where条件（原子操作）**

悲观锁的方案保证了数据库中num的值在同一时间只能被一个进程读取并处理，也就是并发的读取进程到这里要排队依次执行。乐观锁的方案虽然num的值可以被多个进程同时读取到，但是更新操作中version的等值判断可以保证并发的更新操作在同一时间只能有一个更新成功。

还有一种更简单的方案，只在更新操作时加上num>0的条件限制即可。通过where条件限制的方案虽然看似和乐观锁方案类似，都能够防止超发问题的出现，但在num较大时的表现还是有很大区别的。假如此时num为10，同时有5个进程读取到了num=10，对于乐观锁的方案由于version字段的等值判断，这5个进程只会有一个更新成功，这5个进程执行完成之后num为9；对于where条件判断的方案，只要num>0都能够更新成功，这5个进程执行完成之后num为5。

<?php  
$result = $this->mysqli->query("SELECT num FROM goods WHERE id=1 LIMIT 1");  
$row = $result->fetch\_assoc();  
$num = intval($row['num']);  
if($num > 0){  
 usleep(100);  
 $this->mysqli->begin\_transaction();  
 $this->mysqli->query("UPDATE goods SET num=num-1 WHERE num>0");  
 $affected\_rows = $this->mysqli->affected\_rows;  
 if($affected\_rows == 1){  
 $this->mysqli->query("INSERT INTO log(good\_id) VALUES({$num})");  
 $affected\_rows = $this->mysqli->affected\_rows;  
 if($affected\_rows == 1){  
 $this->mysqli->commit();  
 echo "success:".$num;  
 }else{  
 $this->mysqli->rollback();  
 echo "fail1:".$num;  
 }  
 }else{  
 $this->mysqli->rollback();  
 echo "fail2:".$num;  
 }  
}else{  
 echo "fail3:".$num;  
}

**基于redis的解决方案**

**基于watch的乐观锁方案**

watch用于监视一个(或多个) key ，如果在事务执行之前这个(或这些) key 被其他命令所改动，那么事务将被打断。这种方案跟mysql中的乐观锁方案类似，具体表现也是一样的。

<?php  
$num = $this->redis->get('num');  
if($num > 0) {  
 $this->redis->watch('num');  
 usleep(100);  
 $res = $this->redis->multi()->decr('num')->lPush('result',$num)->exec();  
 if($res == false){  
 echo "fail1";  
 }else{  
 echo "success:".$num;  
 }  
}else{  
 echo "fail2";  
}

**基于list的队列方案**

基于队列的方案利用了redis出队操作的原子性，抢购开始之前首先将商品编号放入响应的队列中，在抢购时依次从队列中弹出操作，这样可以保证每个商品只能被一个进程获取并操作，不存在超发的情况。该方案的优点是理解和实现起来都比较简单，缺点是当商品数量较多是，需要将大量的数据存入到队列中，并且不同的商品需要存入到不同的消息队列中。

<?php  
public function init(){  
 $this->redis->del('goods');  
 for($i=1;$i<=10;$i++){  
 $this->redis->lPush('goods',$i);  
 }  
 $this->redis->del('result');  
 echo 'init done';  
}  
public function run(){  
 $goods\_id = $this->redis->rPop('goods');  
 usleep(100);  
 if($goods\_id == false) {  
 echo "fail1";  
 }else{  
 $res = $this->redis->lPush('result',$goods\_id);  
 if($res == false){  
 echo "writelog:".$goods\_id;  
 }else{  
 echo "success".$goods\_id;  
 }  
 }  
}

**基于decr返回值的方案**

如果我们将剩余量num设置为一个键值类型，每次先get之后判断，然后再decr是不能解决超发问题的。但是redis中的decr操作会返回执行后的结果，可以解决超发问题。我们首先get到num的值进行第一步判断，避免每次都去更新num的值，然后再对num执行decr操作，并判断decr的返回值，如果返回值不小于0，这说明decr之前是大于0的，用户抢购成功。

<?php  
public function run(){  
 $num = $this->redis->get('num');  
 if($num > 0) {  
 usleep(100);  
 $retNum = $this->redis->decr('num');  
 if($retNum >= 0){  
 $res = $this->redis->lPush('result',$retNum);  
 if($res == false){  
 echo "writeLog:".$retNum;  
 }else{  
 echo "success:".$retNum;  
 }  
 }else{  
 echo "fail1";  
 }  
 }else{  
 echo "fail2";  
 }  
}

**基于setnx的排它锁方案**

redis没有像mysql中的排它锁，但是可以通过一些方式实现排它锁的功能，就类似php使用文件锁实现排它锁一样。

setnx实现了exists和set两个指令的功能，若给定的key已存在，则setnx不做任何动作，返回0；若key不存在，则执行类似set的操作，返回1。我们设置一个超时时间timeout，每隔一定时间尝试setnx操作，如果设置成功就是获得了相应的锁，执行num的decr操作，操作完成删除相应的key，模拟释放锁的操作。

<?php  
public function run(){  
 do {  
 $res = $this->redis->setnx("numKey",1);  
 $this->timeout -= 100;  
 usleep(100);  
 }while($res == 0 && $this->timeout>0);  
 if($res == 0){  
 echo 'fail1';  
 }else{  
 $num = $this->redis->get('num');  
 if($num > 0) {  
 $this->redis->decr('num');  
 usleep(100);  
 $res = $this->redis->lPush('result',$num);  
 if($res == false){  
 echo "fail2";  
 }else{  
 echo "success:".$num;  
 }  
 }else{  
 echo "fail3";  
 }  
 $this->redis->del("numKey");  
 }  
}