抢购、秒杀是如今很常见的一个应用场景，主要需要解决的问题有两个：

1 高并发对数据库产生的压力

2 竞争状态下如何解决库存的正确减少（"超卖"问题）

先重点解决问题2：

常规写法：

|  |
| --- |
| 1. <?php 2. $conn=mysql\_connect("localhost","big","123456"); 3. **if**(!$conn){ 4. echo "connect failed"; 5. exit; 6. } 7. mysql\_select\_db("big",$conn); 8. mysql\_query("set names utf8"); 10. $price=10; 11. $user\_id=1; 12. $goods\_id=1; 13. $sku\_id=11; 14. $number=1; 16. //生成唯一订单 17. **function** build\_order\_no(){ 18. **return** date('ymd').substr(implode(NULL, array\_map('ord', str\_split(substr(uniqid(), 7, 13), 1))), 0, 8); 19. } 20. //记录日志 21. **function** insertLog($event,$type=0){ 22. **global** $conn; 23. $sql="insert into ih\_log(event,type) 24. values('$event','$type')"; 25. mysql\_query($sql,$conn); 26. } 28. //模拟下单操作 29. //库存是否大于0 30. $sql="select number from ih\_store where goods\_id='$goods\_id' and sku\_id='$sku\_id'";//解锁 此时ih\_store数据中goods\_id='$goods\_id' and sku\_id='$sku\_id' 的数据被锁住(注3)，其它事务必须等待此次事务 提交后才能执行 31. $rs=mysql\_query($sql,$conn); 32. $row=mysql\_fetch\_assoc($rs); 33. **if**($row['number']>0){//高并发下会导致超卖 34. $order\_sn=build\_order\_no(); 35. //生成订单 36. $sql="insert into ih\_order(order\_sn,user\_id,goods\_id,sku\_id,price) 37. values('$order\_sn','$user\_id','$goods\_id','$sku\_id','$price')"; 38. $order\_rs=mysql\_query($sql,$conn); 40. //库存减少 41. $sql="update ih\_store set number=number-{$number} where sku\_id='$sku\_id'"; 42. $store\_rs=mysql\_query($sql,$conn); 43. **if**(mysql\_affected\_rows()){ 44. insertLog('库存减少成功'); 45. }**else**{ 46. insertLog('库存减少失败'); 47. } 48. }**else**{ 49. insertLog('库存不够'); 50. } 51. ?> |

优化方案1：将库存字段number字段设为unsigned，当库存为0时，因为字段不能为负数，将会返回false

|  |
| --- |
| 1. //库存减少 2. $sql="update ih\_store set number=number-{$number} where sku\_id='$sku\_id' and number>0"; 3. $store\_rs=mysql\_query($sql,$conn); 4. **if**(mysql\_affected\_rows()){ 5. insertLog('库存减少成功'); 6. } |

优化方案2：Mysql的事务，锁住操作的行

|  |
| --- |
| 1. <?php 2. $conn=mysql\_connect("localhost","big","123456"); 3. **if**(!$conn){ 4. echo "connect failed"; 5. exit; 6. } 7. mysql\_select\_db("big",$conn); 8. mysql\_query("set names utf8"); 10. $price=10; 11. $user\_id=1; 12. $goods\_id=1; 13. $sku\_id=11; 14. $number=1; 16. //生成唯一订单号 17. **function** build\_order\_no(){ 18. **return** date('ymd').substr(implode(NULL, array\_map('ord', str\_split(substr(uniqid(), 7, 13), 1))), 0, 8); 19. } 20. //记录日志 21. **function** insertLog($event,$type=0){ 22. **global** $conn; 23. $sql="insert into ih\_log(event,type) 24. values('$event','$type')"; 25. mysql\_query($sql,$conn); 26. } 28. //模拟下单操作 29. //库存是否大于0 30. mysql\_query("BEGIN");   //开始事务 31. $sql="select number from ih\_store where goods\_id='$goods\_id' and sku\_id='$sku\_id' FOR UPDATE";//此时这条记录被锁住,其它事务必须等待此次事务提交后才能执行 32. $rs=mysql\_query($sql,$conn); 33. $row=mysql\_fetch\_assoc($rs); 34. **if**($row['number']>0){ 35. //生成订单 36. $order\_sn=build\_order\_no(); 37. $sql="insert into ih\_order(order\_sn,user\_id,goods\_id,sku\_id,price) 38. values('$order\_sn','$user\_id','$goods\_id','$sku\_id','$price')"; 39. $order\_rs=mysql\_query($sql,$conn); 41. //库存减少 42. $sql="update ih\_store set number=number-{$number} where sku\_id='$sku\_id'"; 43. $store\_rs=mysql\_query($sql,$conn); 44. **if**(mysql\_affected\_rows()){ 45. insertLog('库存减少成功'); 46. mysql\_query("COMMIT");//事务提交即解锁 47. }**else**{ 48. insertLog('库存减少失败'); 49. } 50. }**else**{ 51. insertLog('库存不够'); 52. mysql\_query("ROLLBACK"); 53. } 54. ?> |

优化方案3：使用非阻塞的文件排它锁

|  |
| --- |
| 1. <?php 2. $conn=mysql\_connect("localhost","root","123456"); 3. **if**(!$conn){ 4. echo "connect failed"; 5. exit; 6. } 7. mysql\_select\_db("big-bak",$conn); 8. mysql\_query("set names utf8"); 10. $price=10; 11. $user\_id=1; 12. $goods\_id=1; 13. $sku\_id=11; 14. $number=1; 16. //生成唯一订单号 17. **function** build\_order\_no(){ 18. **return** date('ymd').substr(implode(NULL, array\_map('ord', str\_split(substr(uniqid(), 7, 13), 1))), 0, 8); 19. } 20. //记录日志 21. **function** insertLog($event,$type=0){ 22. **global** $conn; 23. $sql="insert into ih\_log(event,type) 24. values('$event','$type')"; 25. mysql\_query($sql,$conn); 26. } 28. $fp = fopen("lock.txt", "w+"); 29. **if**(!flock($fp,LOCK\_EX | LOCK\_NB)){ //若不能给文件上锁，说明文件正被占用，被别人加锁了 30. echo "系统繁忙，请稍后再试"; 31. **return**; 32. } 33. //下单 34. $sql="select number from ih\_store where goods\_id='$goods\_id' and sku\_id='$sku\_id'"; 35. $rs=mysql\_query($sql,$conn); 36. $row=mysql\_fetch\_assoc($rs); 37. **if**($row['number']>0){//库存是否大于0 38. //模拟下单操作 39. $order\_sn=build\_order\_no(); 40. $sql="insert into ih\_order(order\_sn,user\_id,goods\_id,sku\_id,price) 41. values('$order\_sn','$user\_id','$goods\_id','$sku\_id','$price')"; 42. $order\_rs=mysql\_query($sql,$conn); 44. //库存减少 45. $sql="update ih\_store set number=number-{$number} where sku\_id='$sku\_id'"; 46. $store\_rs=mysql\_query($sql,$conn); 47. **if**(mysql\_affected\_rows()){ 48. insertLog('库存减少成功'); 49. flock($fp,LOCK\_UN);//释放锁 50. }**else**{ 51. insertLog('库存减少失败'); 52. } 53. }**else**{ 54. insertLog('库存不够'); 55. } 56. fclose($fp); |

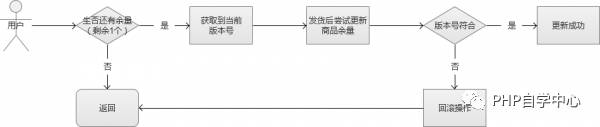
优化方案4：redis队列，因为pop操作是原子的，即使有很多用户同时到达，也是依次执行（推荐，在高并发下，mysql事务和文件锁性能下降很厉害）

先将商品库存存入队列，$store是总库存，‘goods\_store’是redis队列，队列里可能还有数据，所以用总库存减去队列里剩余库存，将差值存入redis。

|  |
| --- |
| 1. <?php 2. $store=1000; 3. $redis=**new** Redis(); 4. $result=$redis->connect('127.0.0.1',6379); 5. $res=$redis->llen('goods\_store'); 6. echo $res; 7. $count=$store-$res; 8. **for**($i=0;$i<$count;$i++){ 9. $redis->lpush('goods\_store',1); 10. } 11. echo $redis->llen('goods\_store'); 12. ?> |

|  |
| --- |
| 1. <?php 2. $conn=mysql\_connect("localhost","big","123456"); 3. **if**(!$conn){ 4. echo "connect failed"; 5. exit; 6. } 7. mysql\_select\_db("big",$conn); 8. mysql\_query("set names utf8"); 10. $price=10; 11. $user\_id=1; 12. $goods\_id=1; 13. $sku\_id=11; 14. $number=1; 16. //生成唯一订单号 17. **function** build\_order\_no(){ 18. **return** date('ymd').substr(implode(NULL, array\_map('ord', str\_split(substr(uniqid(), 7, 13), 1))), 0, 8); 19. } 20. //记录日志 21. **function** insertLog($event,$type=0){ 22. **global** $conn; 23. $sql="insert into ih\_log(event,type) 24. values('$event','$type')"; 25. mysql\_query($sql,$conn); 26. } 28. //模拟下单操作 29. //下单前判断redis队列库存量 30. $redis=**new** Redis(); 31. $result=$redis->connect('127.0.0.1',6379); 32. $count=$redis->lpop('goods\_store');  //弹出一个库存 33. **if**(!$count){ 34. insertLog('error:no store redis'); 35. **return**; 36. } 38. //生成订单 39. $order\_sn=build\_order\_no(); 40. $sql="insert into ih\_order(order\_sn,user\_id,goods\_id,sku\_id,price) 41. values('$order\_sn','$user\_id','$goods\_id','$sku\_id','$price')"; 42. $order\_rs=mysql\_query($sql,$conn); 44. //库存减少 45. $sql="update ih\_store set number=number-{$number} where sku\_id='$sku\_id'"; 46. $store\_rs=mysql\_query($sql,$conn); 47. **if**(mysql\_affected\_rows()){ 48. insertLog('库存减少成功'); 49. }**else**{ 50. insertLog('库存减少失败'); 51. } |

优化方案5：乐观锁



这个数据所有请求都有资格去修改，但会获得一个该数据的版本号，只有版本号符合的才能更新成功，其他的返回抢购失败。

优化方案6：Redis的 watch

类似于mysql的乐观锁；

监控一个KEY是否被其他的客户端修改。如果KEY在调用watch()和exec()之间被修改，那么批量处理最终的exec()执行将失败。

但是，通过一些实验，这个函数的效果其实并没有那么好，或者说不能够准确的去监控。



超卖现象：

若用原来的：

1. **if**($row['number']>0){//库存是否大于0

当库存剩1时，同时进来n个人同时抢，if条件都满足，则会出现超卖的现象！

上面会出现一个用户抢多个的情况，解决：设一个排队队列，一个已抢队列，高并发时，先将用户进入排队队列，取出一个用户，判断用户是否在已抢队列中，如果在，则不能再抢，不在，则库存减一，写数据库，将其添加到已抢队列；

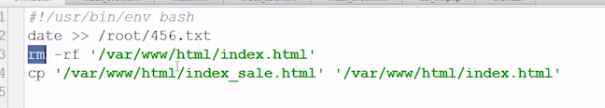
最终优化之后的实例：

|  |
| --- |
| $uid = $\_SESSION['uid'];  $redis->lPush('wait\_list',$uid);//将用户id压入队列  $count = $redis->rPop('num');//弹出一个库存（）  if(!$count){// 判断库存是否充足  echo '库存不足';  }  $user = $redis->rPop('wait\_list');//从排队队列弹出一个用户  $bought\_list = $redis->lRange('bought\_list',0,-1);//已抢购用户队列  if($user && in\_array($user,$bought\_list)){//判断该用户是否存在并且已经抢过  echo '你已经抢过了';  }else{  //生成订单等数据库读写操作  $redis->lPush('bought\_list',$user);//将该用户压入已抢购队列；  } |

具体实现：

秒杀等待页面->秒杀进行页面->秒杀结束页面

第一步：等待页面到秒杀进行页面的切换，使用shell脚本（如果用js倒计时来控制页面，可能会有人篡改前端代码，跳过js）



然后通过linux定时任务，在秒杀开始时执行以上脚本；将页面进行替换。

第二步：秒杀进行到秒杀结束页面的切换，用php解决

