秒杀和限流

# 1 基于redis的秒杀方案

抢库存

发消息

下订单

## 1 缓存秒杀模型

Watch保证数量的一致性

执行exec

扣减incrBy

事务multi

Redis中的商品数量

|  |
| --- |
| redis是单线程的，所以在redis中所有命令都是原子操作。而当要多条redis命令同时执行而不被打断时，则需要使用redis的事务了。 |

|  |
| --- |
| **- MULTI**  **- EXEC**  **- DISCARD**  **- WATCH**  MULTI命令  用于开启一个事务，它总是返回OK。MULTI执行之后,客户端可以继续向服务器发送任意多条命令， 这些命令不会立即被执行，而是被放到一个队列中，当 EXEC命令被调用时， 所有队列中的命令才会被执行。  EXEC命令  负责触发并执行事务中的所有命令：  如果客户端成功开启事务后执行EXEC，那么事务中的所有命令都会被执行。  如果客户端在使用MULTI开启了事务后，却因为断线而没有成功执行EXEC,那么事务中的所有命令都不会被执行。  需要特别注意的是：即使事务中有某条/某些命令执行失败了，事务队列中的其他命令仍然会继续执行——Redis不会停止执行事务中的命令，而不会像我们通常使用的关系型数据库一样进行回滚。  DISCARD命令  当执行 DISCARD 命令时， 事务会被放弃， 事务队列会被清空，并且客户端会从事务状态中退出。  WATCH 命令  可以为Redis事务提供 check-and-set （CAS）行为。被WATCH的键会被监视，并会发觉这些键是否被改动过了。 如果有至少一个被监视的键在 EXEC 执行之前被修改了， 那么整个事务都会被取消， EXEC 返回nil-reply来表示事务已经失败。 |

## 2 实现方案(jedis或者redisson)

1 基于jedis的实现

|  |
| --- |
| jedis.watch(productKey);//保证一致性Transaction tx = jedis.multi();//开启事务tx.incrBy(productKey, -1);//扣减库存List<Object> list = tx.exec();//执行事务  mq.send(order);//发出订单 |

2 基于redisson的实现

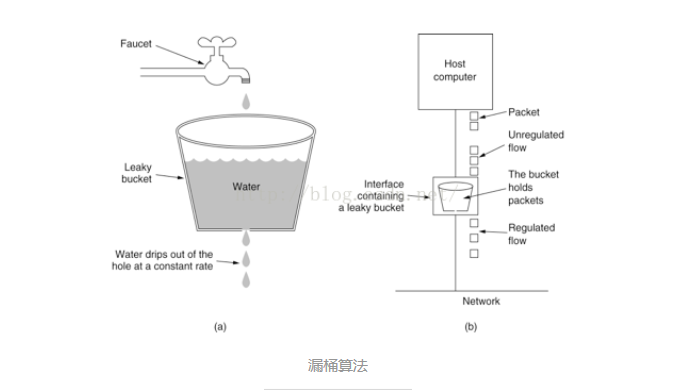
|  |
| --- |
| RSemaphore semaphore = redissonClient.getSemaphore("SEC\_KILL" + skuId + "");  boolean b = semaphore.tryAcquire();  //setnxex(user) //控制频率，规定时间内只能秒一个 |

# 2限流（算法介绍）



## 1 漏桶算法

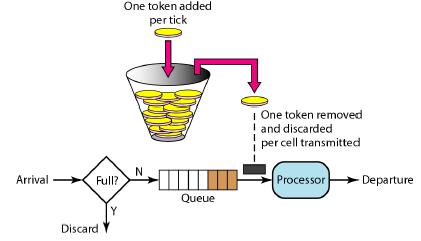
漏桶算法很好的解决了时间边界处理不够**平滑**的问题，在每次请求进桶前都将执行“漏水”的操作，然后再计算当前水量，即不以时间为界限，而以流量为界限进行计算，回避了**时间边界**的问题。



|  |
| --- |
| long timeStamp = getNowTime();  int capacity = 10000;// 桶的容量，即最大承载值  int rate = 1;//水漏出的速度，即服务器的处理请求的能力  int water = 100;//当前水量，即当前的即时请求压力  //当前请求线程进入漏桶方法，true则不被拒绝，false则说明当前服务器负载水量不足，则被拒绝  public static bool control() {  long now = getNowTime();//当前请求时间  //先执行漏水代码  //rate是固定的代表服务器的处理能力，所以可以认为“时间间隔\*rate”即为漏出的水量  water = Math.max(0, water - (now - timeStamp) \* rate);//请求时间-上次请求时间=时间间隔  timeStamp = now;//更新时间，为下次请求计算间隔做准备  if (water < capacity) { // 执行漏水代码后，发现漏桶未满，则可以继续加水，即没有到服务器可以承担的上线  water ++;  return true;  } else {  return false;//水满，拒绝加水，到服务器可以承担的上线，拒绝请求  }  } |

## 2 令牌桶算法

令牌桶算法的原理是系统会以一个恒定的速度往桶里放入令牌，而如果请求需要被处理，则需要先从桶里获取一个令牌，当桶里没有令牌可取时，则拒绝服务。



|  |
| --- |
| long timeStamp=getNowTime();  int capacity; // 桶的容量  int rate ;//令牌放入速度  int tokens;//当前水量  bool control() {  //先执行添加令牌的操作  long now = getNowTime();  tokens = max(capacity, tokens+ (now - timeStamp)\*rate);  timeStamp = now;  if(tokens<1){  return false; //令牌已用完，拒绝访问  }else{  tokens--;  retun true; //还有令牌，领取令牌  }  } |