


京东到家库存系统架构设计

京东技术 2018-04-12

文章转载自公众号  达达京东到家技术，作者 柳志崇

 来这里找志同道合的小伙伴！

目前，京东到家库存系统经历两年多的线上考验与技术迭代，现服务着万级商家十万级店铺的规模，需求的变更与技术演进，我们是如何做到系统的稳定性与高可用呢，下图会给你揭晓答案（通过强大的基础服务平台让应用、JVM、Docker、物理机所有健康指标一目了然，7*24小时智能监控告警让开发无须一直盯着监控，另外数据与业务相辅相成，用数据验证业务需求，迭代业务需求，让业务需求都尽可能的收益最大化，库存系统的开发同学只需要关注业务需求，大版本上线前相应的测试同学会跟进帮你压测，防止上线后潜在的性能瓶颈）。

京东到家-库存系统技术架构





库存系统的架构很有意思，从上图来看功能上其实并不复杂，但是他面临的技术复杂度却是相当高的，比如秒杀品在高并发的情况下如何防止超卖，另外库存系统还不是一个纯技术的系统，需要结合用户的行为特点来考虑，比如下文中提到什么时间进行库存的扣减最合适，我们先抛出几个问题和大家一起探讨下，如有不妥不处，欢迎大家拍砖。

库存什么时候进行预占(或者扣减)呢

商家销售的商品数量是有限的，用户下单后商品会被扣减，我们可以怎么实现呢？

举个例子：一件商品有1000个库存，现在有1000个用户，每个用户计划同时购买1000个。

- （实现方案1）如果用户加入购物车时进行库存预占，那么将只能有1个用户将1000个商品加入购物车。
- （实现方案2）如果用户提交订单时进行库存预占，那么将也只能有1个用户将1000个商品提单成功，其它的人均提示“库存不足，提单失败”。
- （实现方案3）如果用户提交订单&支付成功时进行库存预占，那么这1000个人都能生成订单，但是只有1个人可以支付成功，其它的订单均会被自动取消。

京东到家目前采用的是方案2，理由：

- 用户可能只是暂时加入购物车，并不表示用户最终会提单并支付。
- 所以在购物车进行库存校验并预占，会造成其它真正想买的用户不能加入购物车的情况，但是之前加车的用户一直不付款，最终损失的是公司。
- 方案3会造成生成1000个订单，无论是在支付前校验库存还是在支付成功后再检验库存，都会造成用户准备好支付条件后却会出现99.9%的系统取消订单的概率，也就是说会给99.9%的用户体验到不爽的感觉。
- 数据表明用户提交订单不支付的占比是非常小的（相对于加入购物车不购买的行为），目前京东到家给用户预留的最长支付时间是30分钟，超过30分钟订单自动取消，预占的库存自动释放

综上所述，方案2也可能由于用户下单预占库存但最终未支付，造成库存30分钟后才能被其它用户使用的情况，但是相较于方案1，方案3无疑是折中的最好方案。

重复提交订单的问题？

重复提交订单造成的库存重复扣减的后果是比较严重的。比如商家设置有1000件商品，而实际情况可能卖了900件就提示用户无货了，给商家造成无形的损失

可能出现重复提交订单的情况：

- （1、用户善意行为）app上用户单击“提交订单”按钮后由于后端接口没有返回，用户以为没有操作成功会再次单击“提交订单”按钮
- （2、用户恶意行为）黑客直接刷提单接口，绕过App端防重提交功能
- （3、提单系统重试）比如提单系统为了提高系统的可用性，在第一次调用库存系统扣减接口超时后会重试再次提交扣减请求

好了，既然问题根源缕清楚了，我们一一对症下药

- （1、用户善意行为）app侧在用户第一次单击“提交订单”按钮后对按钮进行置灰，禁止再次提交订单
- （2、用户恶意行为）采用令牌机制，用户每次进入结算页，提单系统会颁发一个令牌ID（全局唯一），当用户点击“提交订单”按钮时发起的网络请求中会带上这个令牌

ID,这个时候提单系统会优先进行令牌ID验证, 令牌ID存在&令牌ID访问次数=1的话才会放行处理后续逻辑, 否则直接返回

- (3、提单系统重试) 这种情况则需要后端系统(比如库存系统)来保证接口的幂等性, 每次调用库存系统时均带上订单号, 库存系统会基于订单号增加一个分布式事务锁, 伪代码如下:

```
1. int ret=redis.incr(orderId);
2. redis.expire(orderId,5,TimeUnit.MINUTES);
3. if(ret==1){//添加成功, 说明之前没有处理过这个订单号或者5分钟之前处理过了
4.     boolean alreadySuccess=alreadySuccessDoOrder(orderProductRequest);
5.     if(!alreadySuccess){
6.         doOrder(orderProductRequest);
7.     }else{
8.         return "操作失败, 原因: 重复提交";
9.     }
10. }else{
11.     return "操作失败, 原因: 重复提交";
12. }
```

库存数据的回滚机制如何做

需要库存回滚的场景也是比较多的, 比如:

- (1、用户未支付) 用户下单后后悔了
- (2、用户支付后取消) 用户下单&支付后后悔了
- (3、风控取消) 风控识别到异常行为, 强制取消订单
- (4、耦合系统故障) 比如提交订单时提单系统T1同时会调用积分扣减系统X1、库存扣减系统X2、优惠券系统X3, 假如X1,X2成功后, 调用X3失败, 需要回滚用户积分与商家库存。

其中场景1, 2, 3比较类似, 都会造成订单取消, 订单中心取消后会发送mq出来, 各个系统保证自己能正确消费订单取消MQ即可。而场景4订单其实尚未生成, 相对来说要复杂些, 如上面提到的, 提单系统T1需要主动发起库存系统X2、优惠券系统X3的回滚请求(入参必须带上订单号), X2、X3回滚接口需要支持幂等性。

其实针对场景4, 还存在一种极端情况, 如果提单系统T1准备回滚时自身也宕机了, 那么库存系统X2、优惠券系统X3就必须依靠自己来完成回滚操作了, 也就是说具备自我数据健康检查的能力, 具体怎么说实现呢?

可以利用当前订单号所属的订单尚未生成的特点, 可以通过worker机制, 每次捞取40分钟(这里的40一定要大于容忍用户的支付时间)前的订单, 调用订单中心查询订单的状态, 确保不是已取消的, 否则进行自我数据的回滚。

多人同时购买1件商品, 如何安全地库存扣减

现实中同一件商品可能会出现多人同时购买的情况，我们可以如何做到并发安全呢？

伪代码片段1：

```
1. synchronized(this){
2.     long stockNum = getProductStockNum(productId);
3.     if(stockNum>requestBuyNum) {
4.         int ret=updateSQL("update stock_main set stockNum=stockNum-
5.             if(ret==1){
6.                 return "扣减成功";
7.             }else {
8.                 return "扣减失败";
9.             }
10.    }
11. }
```

伪代码片段1的设计思想是所有的请求过来之后首先加锁，强制其串行化处理，可见其效率一定不高，

伪代码片段2：

```
1. int ret=updateSQL("update stock_main set stockNum=stockNum-"+requ
2. if(ret==1){
3.     return "扣减成功";
4. }else {
5.     return "扣减失败";
6. }
```

这段代码只是在where条件里增加了and stockNum>="+requestBuyNum即可防止超卖的行为，达到了与上述伪代码1的功能

如果商品是促销品（比如参与了秒杀的商品）并发扣减的机率会更高，那么数据库的压力会更高，这个时候还可以怎么做呢 海量的用户秒杀请求,本质上是一个排序,先到先得.但是如此之多的请求，注定了有些人是抢不到的，可以在进入上述伪代码Dao层之前增加一个计数器进行控制，比如有50%的流量将直接告诉其抢购失败，伪代码如下：

```
1. public class SeckillServiceImpl{
2.     private long count=0;
3.
4.     public String buy(User user,int productId,int productNum){
5.         count++;
6.         if(count%2=1){
7.             Thread.sleep(1000);
8.             return "抢购失败";
9.         }else{
10.             return doBuy(user,productId,productNum);
11.         }
12.     }
13. }
```

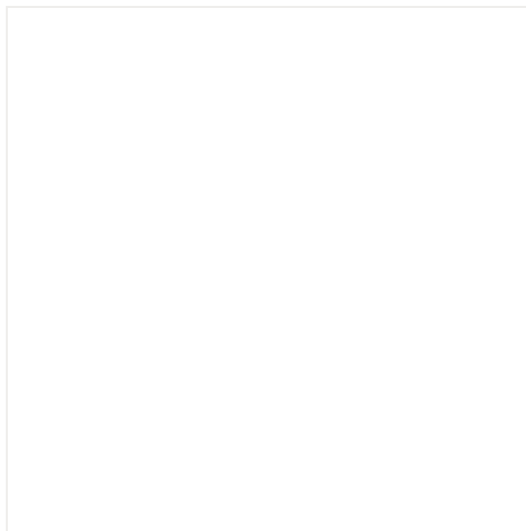
```
l1.      }  
l2.      }  
l3. }
```

另外同一个用户，不允许多次抢购同一件商品，我们又该如何做呢

```
1. public String doBuy(user,productId,productNum){  
2.     //用户除了第一次进入值为1，其它时候均大于1  
3.     int tmp=redis.incr(user.getUid()+productId);  
4.     if(tmp==1){  
5.         redis.expire(user.getUid()+productId,3600); //1小时后key自  
6.         doBuy1(user,productId,productNum);  
7.     }else{  
8.         return "抢购失败";  
9.     }  
10. }
```

怎么样，看了上述的介绍是不是觉得库存系统很有意思，而且总会有你意想不到的高并发问题等你来日挑战，当然了，也非常欢迎你加入我们（目前北京、上海均在招Java高级工程师，可以加微信【北京的同学加**lzc_java**，上海的同学加**25252937**】进一步了解职位详情，等你来约）

京东技术 | 关注技术的公众号



长按，识别二维码，加关注