

## 01 | 创建和更新订单时，如何保证数据准确无误？

2020-02-26 李玥

后端存储实战课

[进入课程 >](#)



讲述：李玥

时长 15:31 大小 12.44M



你好，我是李玥。

订单系统是整个电商系统中最重要的一個子系统，订单数据也就是电商企业最重要的数据资产。今天这节课，我来和你说一下，在设计和实现一个订单系统的存储过程中，有哪些问题是要特别考虑的。

一个合格的订单系统，最基本的要求是什么？**数据不能错。**



一个购物流程，从下单开始、支付、发货，直到收货，这么长的一个流程中，每一个环节，都少不了更新订单数据，每一次更新操作又需要同时更新好几张表。这些操作可能被随机分布到很多台服务器上执行，服务器有可能故障，网络有可能出问题。

在这么复杂的情况下，保证订单数据一笔都不能错，是不是很难？实际上，只要掌握了方法，其实并不难。

首先，你的代码必须是正确没 Bug 的，如果说是因为代码 Bug 导致的数据错误，那谁也救不了你。

然后，你要会正确地使用数据库的事务。比如，你在创建订单的时候，同时要在订单表和订单商品表中插入数据，那这些插入数据的 INSERT 必须在一个数据库事务中执行，数据库的事务可以确保：执行这些 INSERT 语句，要么一起都成功，要么一起都失败。

我相信这些“基本操作”对于你来说，应该不是问题。

但是，还有一些情况下会引起数据错误，我们一起来看一下。不过在此之前，我们要明白，对于一个订单系统而言，它的核心功能和数据结构是怎样的。

因为，任何一个电商，它的订单系统的功能都是独一无二的，基于它的业务，有非常多的功能，并且都很复杂。我们在讨论订单系统的存储问题时，必须得化繁为简，只聚焦那些最核心的、共通的业务和功能上，并且以这个为基础来讨论存储技术问题。

## 订单系统的核心功能和数据

我先和你简单梳理一下一个订单系统必备的功能，它包含但远远不限于：

1. 创建订单；
2. 随着购物流程更新订单状态；
3. 查询订单，包括用订单数据生成各种报表。

为了支撑这些必备功能，在数据库中，我们至少需要有这样几张表：

1. 订单主表：也叫订单表，保存订单的基本信息。
2. 订单商品表：保存订单中的商品信息。
3. 订单支付表：保存订单的支付和退款信息。
4. 订单优惠表：保存订单使用的所有优惠信息。

这几个表之间的关系是这样的：订单主表和后面的几个子表都是一对多的关系，关联的外键就是订单主表的主键，也就是订单号。

绝大部分订单系统的核心功能和数据结构都是这样的。

## 如何避免重复下单？

接下来我们来看一个场景。一个订单系统，提供创建订单的 HTTP 接口，用户在浏览器页面上点击“提交订单”按钮的时候，浏览器就会给订单系统发一个创建订单的请求，订单系统的后端服务，在收到请求之后，往数据库的订单表插入一条订单数据，创建订单成功。

假如说，用户点击“创建订单”的按钮时手一抖，点了两下，浏览器发了两个 HTTP 请求，结果是什么？创建了两条一模一样的订单。这样肯定不行，需要做防重。

有的同学会说，前端页面上应该防止用户重复提交表单，你说的没错。但是，网络错误会导致重传，很多 RPC 框架、网关都会有自动重试机制，所以对于订单服务来说，重复请求这个事儿，你是没办法完全避免的。

解决办法是，**让你的订单服务具备幂等性**。什么是幂等呢？一个幂等操作的特点是，其任意多次执行所产生的影响均与一次执行的影响相同。也就是说，一个幂等的方法，使用同样的参数，对它进行调用多次和调用一次，对系统产生的影响是一样的。所以，对于幂等的方法，不用担心重复执行会对系统造成任何改变。一个幂等的创建订单服务，无论创建订单的请求发送多少次，正确的结果是，数据库只有一条新创建的订单记录。

这里面有一个不太好解决的问题：对于订单服务来说，它怎么知道发过来的创建订单请求是不是重复请求呢？

在插入订单数据之前，先查询一下订单表里面有没有重复的订单，行不行？不太行，因为你很难用 SQL 的条件来定义“重复的订单”，订单用户一样、商品一样、价格一样，就认为是重复订单么？不一定，万一用户就是连续下了两个一模一样的订单呢？所以这个方法说起来容易，实际上很难实现。

很多电商解决这个问题的思路是这样的。在数据库的最佳实践中有一条就是，数据库的每个表都要有主键，绝大部分数据表都遵循这个最佳实践。一般来说，我们在往数据库插入一条

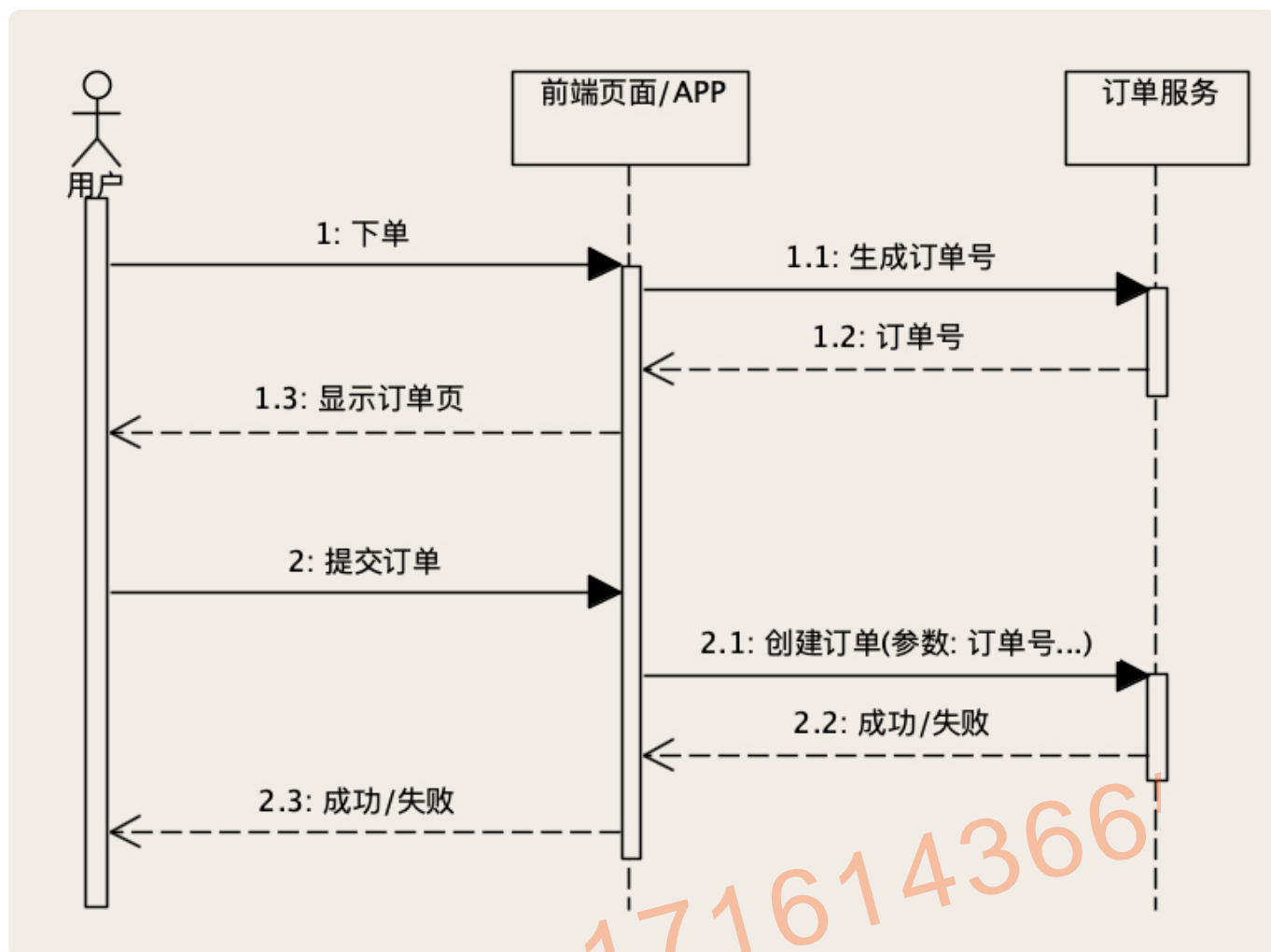
记录的时候，都不提供主键，由数据库在插入的同时自动生成一个主键。这样重复的请求就会导致插入重复数据。

我们知道，表的主键自带唯一约束，如果我们在一条 INSERT 语句中提供了主键，并且这个主键的值在表中已经存在，那这条 INSERT 会执行失败，数据也不会被写入表中。**我们可以利用数据库的这种“主键唯一约束”特性，在插入数据的时候带上主键，来解决创建订单服务的幂等性问题。**

具体的做法是这样的，我们给订单系统增加一个“生成订单号”的服务，这个服务没有参数，返回值就是一个新的、全局唯一的订单号。在用户进入创建订单的页面时，前端页面先调用这个生成订单号服务得到一个订单号，在用户提交订单的时候，在创建订单的请求中带着这个订单号。

这个订单号也是我们订单表的主键，这样，无论是用户手抖，还是各种情况导致的重试，这些重复请求中带的都是同一个订单号。订单服务在订单表中插入数据的时候，执行的这些重复 INSERT 语句中的主键，也都是同一个订单号。数据库的唯一约束就可以保证，只有一次 INSERT 语句是执行成功的，这样就实现了创建订单服务幂等性。

为了便于你理解，我把上面这个幂等创建订单的流程，绘制成了时序图供你参考：



还有一点需要注意的是，如果是因为重复订单导致插入订单表失败，订单服务不要把这个错误返回给前端页面。否则，就有可能出现这样的情况：用户点击创建订单按钮后，页面提示创建订单失败，而实际上订单却创建成功了。正确的做法是，遇到这种情况，订单服务直接返回订单创建成功就可以了。

## 如何解决 ABA 问题？

同样，订单系统各种更新订单的服务一样也要具备幂等性。

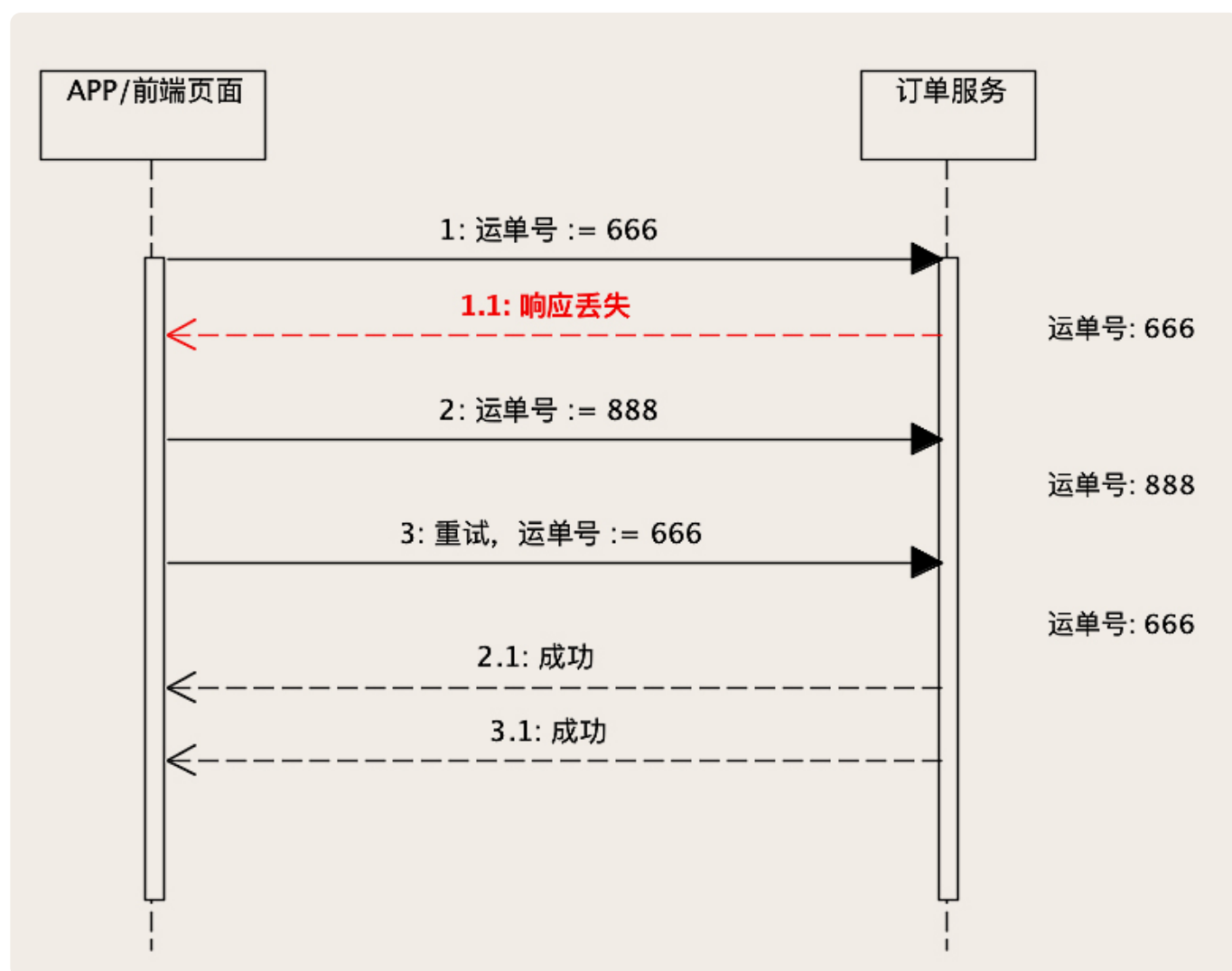
这些更新订单服务，比如说支付、发货等等这些步骤中的更新订单操作，最终落到订单库上，都是对订单主表的 UPDATE 操作。数据库的更新操作，本身就具备天然的幂等性，比如说，你把订单状态，从未支付更新成已支付，执行一次和重复执行多次，订单状态都是已支付，不用我们做任何额外的逻辑，这就是天然幂等。

那在实现这些更新订单服务时，还有什么问题需要特别注意的吗？还真有，在并发环境下，你需要注意 ABA 问题。

什么是 ABA 问题呢？我举个例子你就明白了。比如说，订单支付之后，小二要发货，发货完成后要填个快递单号。假设说，小二填了一个单号 666，刚填完，发现填错了，赶紧再修改成 888。对订单服务来说，这就是 2 个更新订单的请求。

正常情况下，订单中的快递单号会先更新成 666，再更新成 888，这是没问题的。那不正常情况呢？666 请求到了，单号更新成 666，然后 888 请求到了，单号又更新成 888，但是 666 更新成功的响应丢了，调用方没收到成功响应，自动重试，再次发起 666 请求，单号又被更新成 666 了，这数据显然就错了。这就是非常有名的 ABA 问题。

具体的时序你可以参考下面这张时序图：




ABA 问题怎么解决？这里给你提供一个比较通用的解决方法。给你的订单主表增加一列，列名可以叫 version，也即是“版本号”的意思。每次查询订单的时候，版本号需要随着订单数据返回给页面。页面在更新数据的请求中，需要把这个版本号作为更新请求的参数，再带回给订单更新服务。



订单服务在更新数据的时候，需要比较订单当前数据的版本号，是否和消息中的版本号一致，如果不一致就拒绝更新数据。如果版本号一致，还需要再更新数据的同时，把版本号 +1。“比较版本号、更新数据和版本号 +1”，这个过程必须在同一个事务里面执行。

具体的 SQL 可以这样来写：

 复制代码

```
1 UPDATE orders set tracking_number = 666, version = version + 1
2 WHERE version = 8;
```

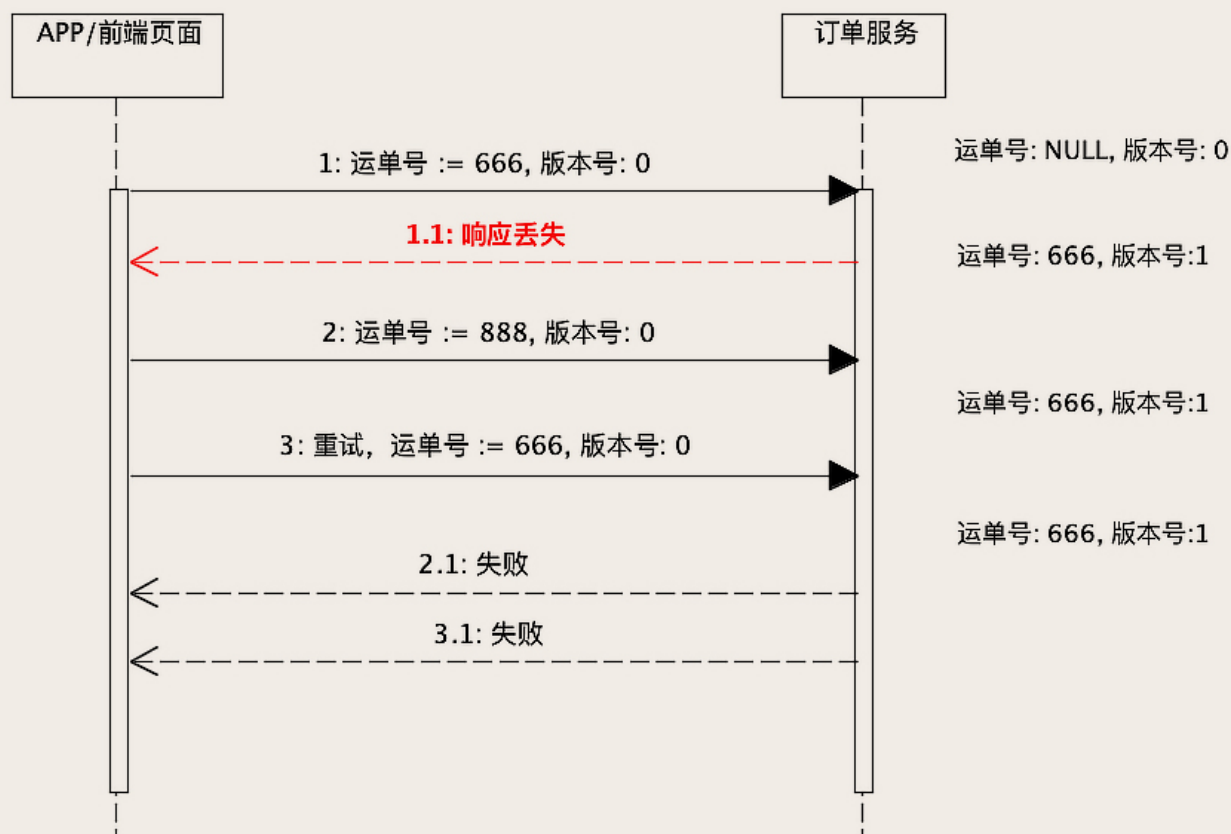
在这条 SQL 的 WHERE 条件中，version 的值需要页面在更新的时候通过请求传进来。

通过这个版本号，就可以保证，从我打开这条订单记录开始，一直到我更新这条订单记录成功，这个期间没有其他人修改过这条订单数据。因为，如果有其他人修改过，数据库中的版本号就会改变，那我的更新操作就不会执行成功。我只能重新查询新版本的订单数据，然后再尝试更新。

有了这个版本号，再回头看一下我们上面那个 ABA 问题的例子，会出现什么结果？可能出现两种情况：

1. 第一种情况，把运单号更新为 666 的操作成功了，更新为 888 的请求带着旧版本号，那就会更新失败，页面提示用户更新 888 失败。
2. 第二种情况，666 更新成功后，888 带着新的版本号，888 更新成功。这时候即使重试的 666 请求再来，因为它和上一条 666 请求带着相同的版本号，上一条请求更新成功后，这个版本号已经变了，所以重试请求的更新必然失败。

无论哪种情况，数据库中的数据与页面上给用户的反馈都是一致的。这样就可以实现幂等更新并且避免了 ABA 问题。下图展示的是第一种情况，第二种情况也是差不多的：



## 小结

我们把今天这节课的内容做一个总结。今天这节课，实际上就讲了一个事儿，也就是，实现订单操作的幂等的方法。

因为网络、服务器等等这些不确定的因素，重试请求是普遍存在并且不可避免的。具有幂等性的服务可以完美地克服重试导致的数据错误。

对于创建订单服务来说，可以通过预先生成订单号，然后利用数据库中订单号的唯一约束这个特性，避免重复写入订单，实现创建订单服务的幂等性。对于更新订单服务，可以通过一个版本号机制，每次更新数据前校验版本号，更新数据同时自增版本号，这样的方式，来解决 ABA 问题，确保更新订单服务的幂等性。

通过这样两种幂等的实现方法，就可以保证，无论请求是不是重复，订单表中的数据都是正确的。当然，上面讲到的实现订单幂等的方法，你完全可以套用在其他需要实现幂等的服务中，只需要这个服务操作的数据保存在数据库中，并且有一张带有主键的数据表就可以了。

## 思考题



实现服务幂等的方法，远不止我们这节课上介绍的这两种，课后请你想一下，在你负责开发的业务系统中，能不能用这节课中讲到的方法来实现幂等？除了这两种方法以外，还有哪些实现服务幂等的方法？欢迎你在留言区与我交流互动。

感谢你的阅读，如果你觉得今天的内容对你有所帮助，也欢迎把它分享给你的朋友。

# 后端存储实战课

## 类电商平台存储技术应用指南

李玥

京东零售计算存储平台部资深架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 课前加餐| 电商系统是如何设计的？

下一篇 02 | 流量大、数据多的商品详情页系统该如何设计？

### 精选留言 (23)

 写留言



李玥 置顶

2020-02-26

hello，我是李玥。之后我都会在留言板上同步上节课思考题的答案，欢迎你跟我一起学习讨论。

在课前加餐这节课里，我给你留了道思考题，让你作为公司的CTO，想一想上节课我们提到的电商系统，它的技术选型应该是什么样的？...

展开 ▾

💬 1

👍 11



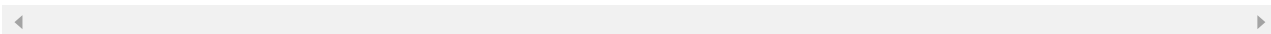
川杰

2020-02-26

请问，生成订单号 服务的一般逻辑会是怎样的？想来想去，如果要想这个ID全局唯一，只能带上时间，可是如果带上时间，像那种，不小心点了两次按钮的情况，必然是两个不同的订单号；请问这个问题怎么解决？

作者回复: 如果单纯是生成GUID（全局唯一ID）方法有很多，比如小规模系统完全可以用MySQL的Sequence或者Redis来生成。大规模系统也可以采用类似雪花算法之类的方式分布式生成GUID。

但是订单号这个东西又有点儿特殊要求，比如在订单号中最好包含一些品类、时间等信息，便于业务处理，再比如，订单号它不能是一个单纯自增的ID，否则别人很容易根据订单号计算出你大致的销量，所以订单号的生产算法在保证不重复的前提下，一般都会加入很多业务规则在里面，这个每家都不一样，算是商业秘密吧。



💬 8

👍 8



业余爱好者

2020-02-26

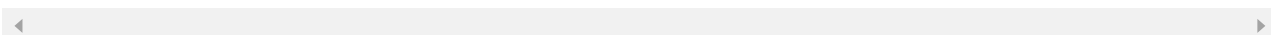
每次请求之前必须先生成一个唯一的请求id,服务端将该id暂时放入redis。客户端请求时必须携带上这个id，借口会首先到redis中查询，如何有的话就继续后续的处理逻辑，同时删除该id,又有的话就退出，返回不能重复请求的错误到客户端。

一句话总结：每次处理必须对应一个一次性的token。

展开 ▾

作者回复: 这个思路非常好，并且可以适用很多的场景。

如果是超大规模的系统，可能用一个Redis实例来生成和验证这个GUID就忙不过来了，大家也可以想一下有没有什么性能上更好的方式？



💬 1

👍 8



鸠摩智

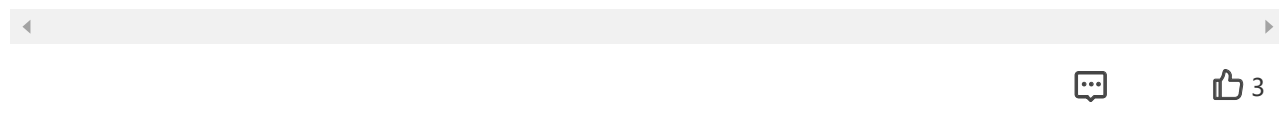
2020-02-28

老师，生成全局唯一订单号如果不是自增的，插入mysql innodb表的时候，底层的B+树

索引是不是会发生页分裂等问题，影响插入性能？如果遇到大促，短时间生成大量订单，写入会成为瓶颈不？

作者回复: 这确实是一个需要考虑的性能问题。

所以很多业务在设计订单号规则的时候都不是完全随机的，一般都是递增的。这种情况下，页分裂就不会特别严重。



约书亚

2020-02-27

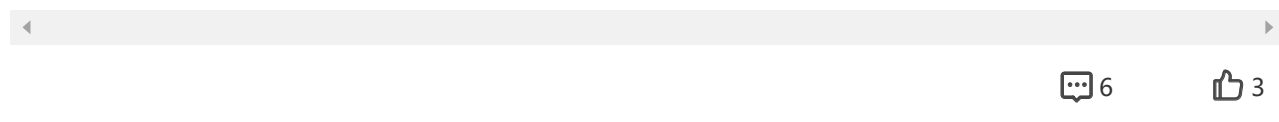
请教您一个实际问题：

无论采用哪种生成订单id的方式，短时间内都可能出现靠前的id后提交（生成订单的问题），这样一定程度上数据库接收到的id不是严格按时间递增的。而页面浏览的场景，尤甚。

请问在您做过的系统里，这会带来一定程度上的性能下降嘛？

展开 ▾

作者回复: 你是指在数据库中插入数据的时候，写入索引的性能下降吗？



每天晒白牙

2020-02-28

学习收获

1.创建订单如何解决重复下单？

通过提前预先生成订单号【这里生成订单号可以走单独的id生成器，如何设计一个好的id生成器需要学习】，利用数据库订单号唯一约束【订单号是主键，但主键不是由数据库自增，而是由外部传入】来避免重复入库，实现幂等...

展开 ▾



aoe

2020-02-27

老师，您在“幂等创建订单的流程”的时序图中：1. 下单；1.1. 生成订单号 ....., 按这个逻辑是不能防止重复提交。

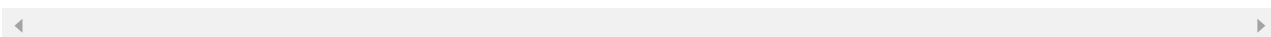
因为：下单后会走一个完整的生成订单流程  
留言中” 业余爱好者 “的思路是正确的

展开 ▾

作者回复: “业余爱好者” 同学相比课中的方案, 多了一步在Redis中验证订单号, 我理解, 这种设计一般是出于安全的考虑, 确保提交的订单号确实是我们生成的, 而不是来自外部的攻击。

真正的幂等或者说是防重, 还是要依靠数据库的唯一约束。

也欢迎你说一下你的理解。



💬 3

👍 1



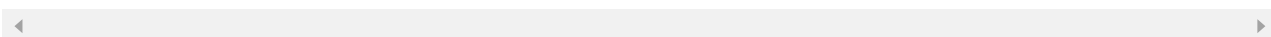
**Spring coming**

2020-02-27

如何重现aba问题呢, 或者说怎么样模拟网络重试?

展开 ▾

作者回复: 用并发的压测工具 (比如LoadRunner) 结合脚本就可以模拟出来。



💬

👍 1



**樱花落花**

2020-03-01

我们公司的解决方案是: 一个用户30s内只能下一单, 这样不仅能够限流还能解决重复问题, 使用redis分布式锁判断, 感觉老师说的前端请求订单号生成服务有点奇怪, 我们的订单号是时间+服务器号+随机生成码之类算法出来的一个自增码, 这样也能用作mysql主键。

展开 ▾

💬

👍

Sticker

**Sticker**

2020-03-01

我有个疑问, 在提交订单的时候, 一种是先拿着订单号去查库, 让业务代码校验是否存在, 另一种是直接利用库表主键唯一约束抛异常, 这两种处理方式哪种性能更好?

💬

👍

Sticker

**Sticker**

2020-03-01

最喜欢看评论了, 总能找到解决我疑问的答案。

展开 ▾



不记年

2020-03-01

订单ID也可以在前端生成吧，可以减少一次网络调用，不过会不会有ID生产算法暴露的风险



陈迪

2020-02-29

第一个防重复创建订单的解决方案中有个疑点，唯一的订单号是在进入“创建订单页面”时创建，用户如果对这个页面不小心同时开了多个tab页，每次打开一个tab页都会生成一个新订单号，那就可以提交多个“一模一样”的订单了，而且用户本意应该是一个订单。

老师怎么看待这个问题？

展开

作者回复: 这种情况我们确实没提到。如果用户打开了多个tab页，只会产生多个订单号，这个时候还没有生成新订单呢。

只有他在每个tab也都点“下单”按钮，才会产生多个订单。这种情况不存在“误操作”的可能了。



GeekAml

2020-02-29

请问老师，服务与服务之间调用的幂等性如何做?总不能要求A服务调用B之前，先请求一个唯一ID吧

作者回复: 你可以参考一下我在《消息队列高手课》中的[这节课]([https://time.geekbang.org/column/article/111552?utm\\_term=zeus7N1US&utm\\_source=geektime&utm\\_medium=app](https://time.geekbang.org/column/article/111552?utm_term=zeus7N1US&utm_source=geektime&utm_medium=app))，里面讲到了一些更通用的实现幂等等方法。

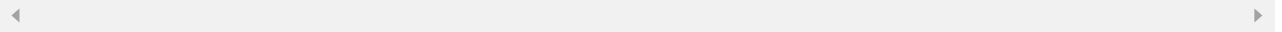


听雨

2020-02-28

老师讲的通透，另外，留胡子可是太帅了，如果把海报也换一下，应该能吸不少女粉

作者回复: 极客有女粉? 我不信。



**电光火石**

2020-02-28

总结:

网络的不稳定会带来重试, 如果保证接口的幂等性:

1. 插入的时候, 通过业务主键来保持数据库的唯一性, 进而保证接口的幂等性
2. 更新的时候, 通过乐观锁来保证操作的幂等性

...

展开 ∨



**观弈道人**

2020-02-28

用心了, 李老师这是要打造爆款的节奏

展开 ∨



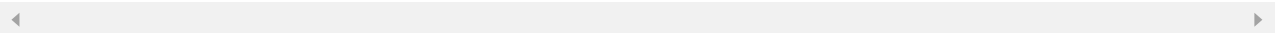
**Spring coming**

2020-02-27

谢谢老师的回答, 除了压测外, 还有什么方式是比较容易模拟的呢? 我对文章中的"但是 666 更新成功的响应丢了, 调用方没收到成功响应, 自动重试, 再次发起 666 请求"有点没理解, 这里边的"调用方"是指的前端业务实现的超时重试还是http协议中有重试机制?

展开 ∨

作者回复: 这里面的调用方有可能APP, 也有可能是某个后端服务, 我们不用去纠结这个, 很多RPC框架或者网关都有这种“超时或失败重试”的机制。



**刘楠**

2020-02-27

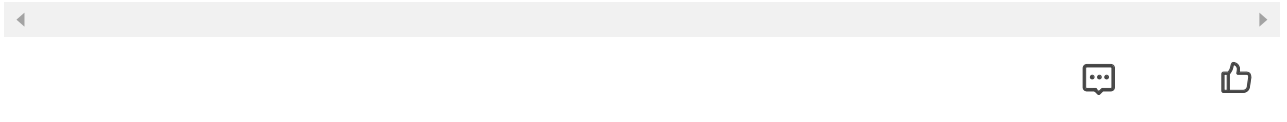
创建订单服务时, 在用户进入创建订单的页面时, 客户端先生成一个单号, 提交时, 把穿上单号提交上来, 同时保存在数据库, 生成订单成功后, 客户端生成单号放缓存-给个短的过期时间, 第二次提交上来, 先查缓存, 有就直接返回, 没有就去生成订单?

展开 ∨



作者回复: 订单号一定是后端服务来生成, 新生成的订单号不用保存在任何地方, 就是返回给前端页面。

提交订单就正常用新的单号插入数据库, 如果用户没提交订单关了页面, 这个单号废了也没关系。不需要做任何清理。



飞翔

2020-02-27

老师 能不能 给这几个表的字段和给个具体例子  
展开 ▾

