# 01-创建和更新订单时，如何保证数据准确无误？

你好，我是李玥。

订单系统是整个电商系统中最重要的一个子系统，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。

绝大部分订单系统它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。

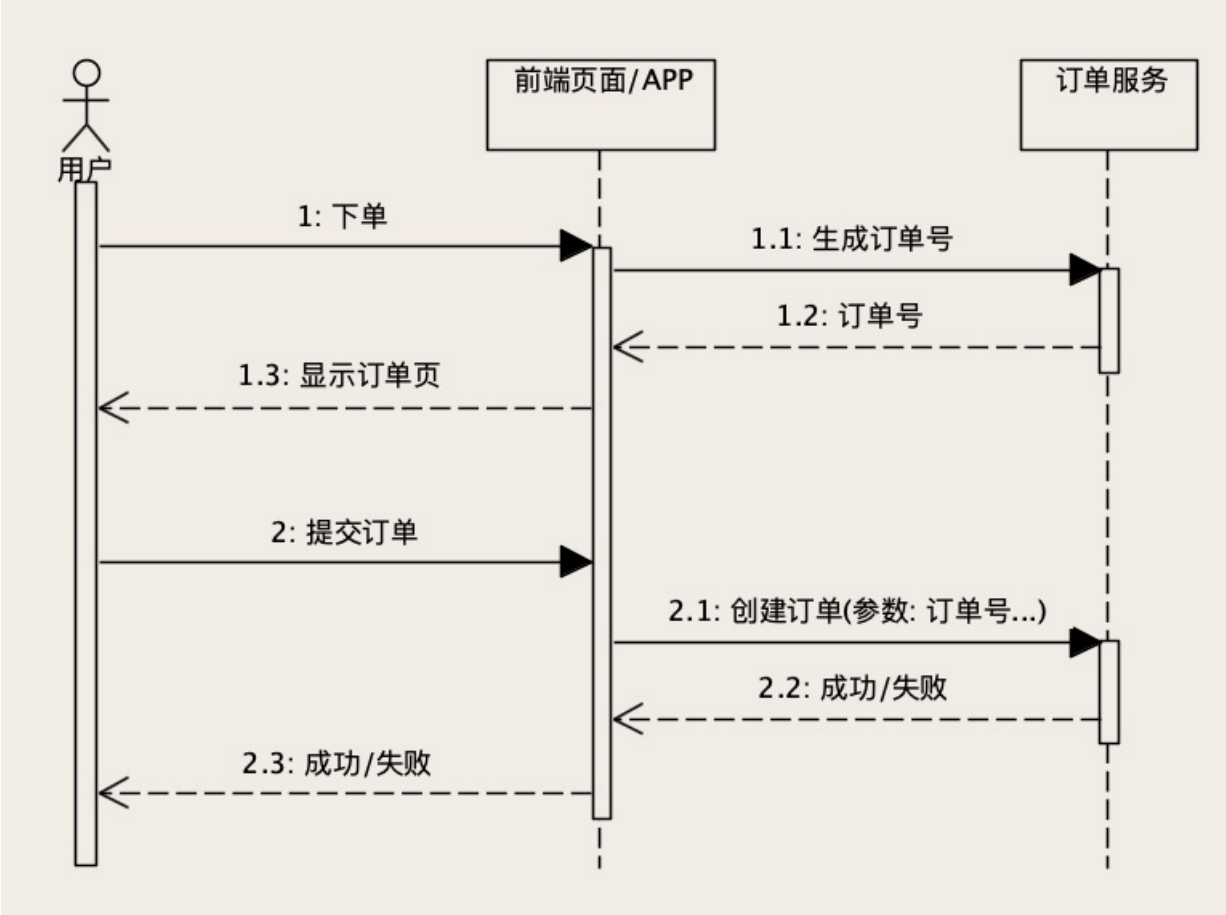
很多电商解决这个问题的思路是这样的。在数据库的最佳实践中有一条就是，数据库的每个表都要有主键，绝大部分数据表都遵循这个最佳实践。一般来说，我们在往数据库插入一条记录的时候，都不提供主键，由数据库在插入的同时自动生成一个主键。这样重复的请求就会导致插入重复数据。

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。

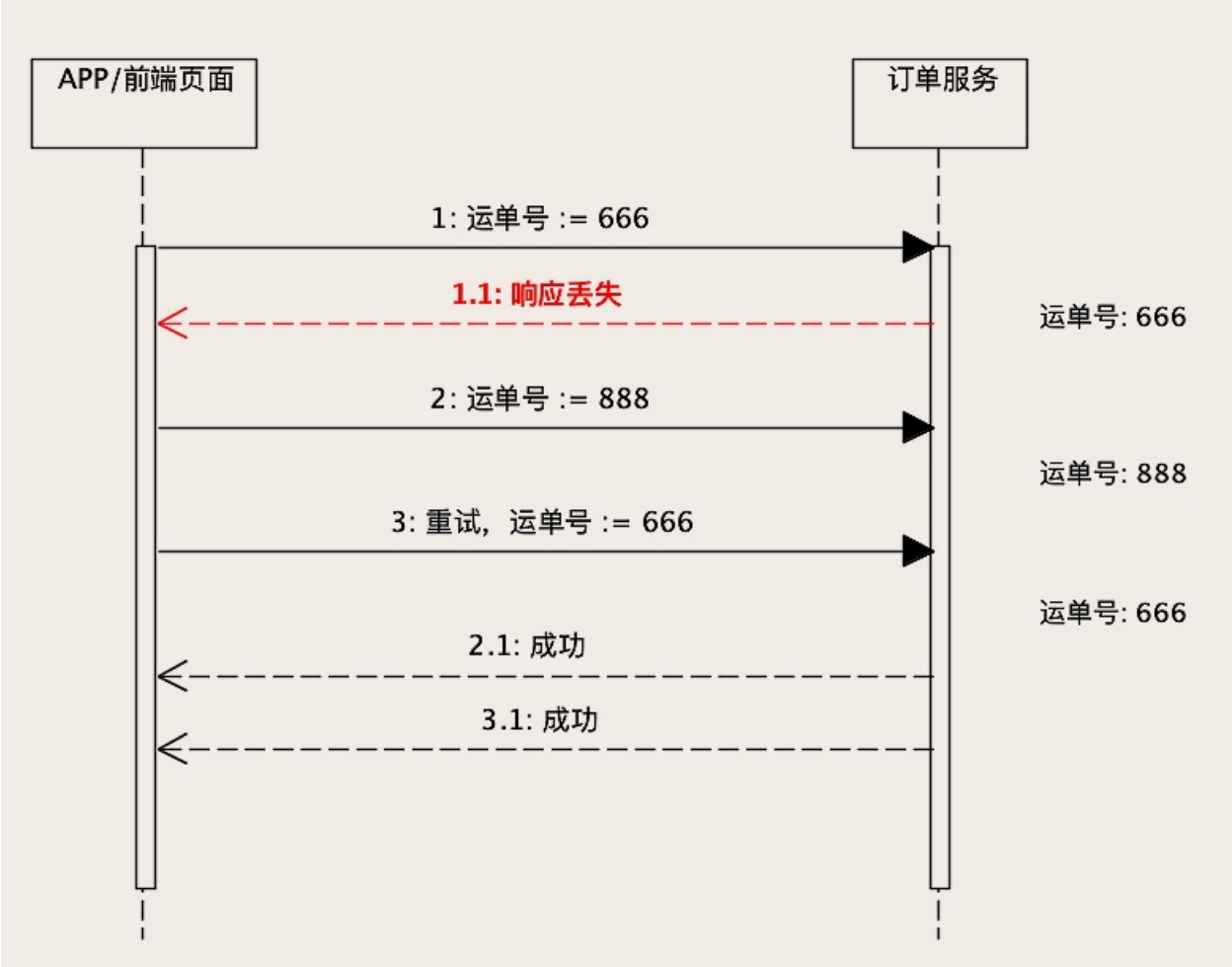
这些更新订单服务，比如说支付、发货等等这些步骤中的更新订单操作，最终落到订单库上，都是对订单主表的UPDATE操作。数据库的更新操作，本身就具备天然的幂等性，比如说，你把订单状态，从未支付更新成已支付，执行一次和重复执行多次，订单状态都是已支付，不用我们做任何额外的逻辑，这就是天然幂等。

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：

```
UPDATE orders set tracking_number = 666, version = version + 1
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。

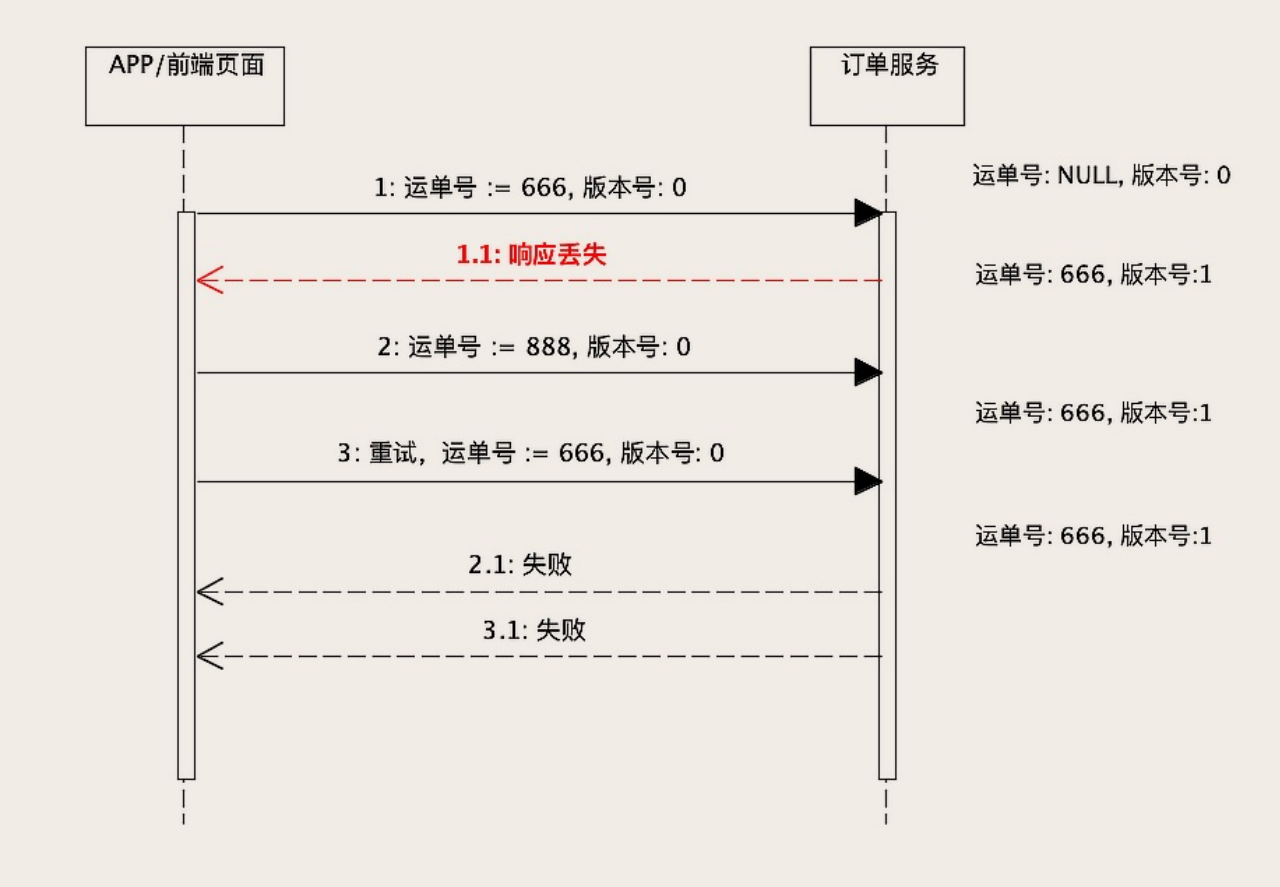
有了这个版本号，再回头看一下我们上面那个ABA问题的例子，会出现什么结果？可能出现两种情况：

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。

## 精选留言：

- 李玥 2020-02-26 18:01:56
  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？

  关于这个问题，我是这么理解的。

  技术选型本身没有好与坏，更多的是选择“合适”的技术。对于编程语言和技术栈的选择，我认为需要从两方面考虑，一方面就是团队的人员配置，尽量选择大家熟悉的技术，第二个方面就是要考察选择的技术它的生态是不是够完善。这两个原则在选择编程语言、技术栈、云服务和存储的时候都是适用的。

  如何根据业务来选择合适的存储系统，这是个很大的话题，我会在后面的课程中陆续穿插的来和你讲，什么场景下应该选择什么样的存储，敬请期待。 [4赞]

- 川杰 2020-02-26 19:49:49
  请问，生成订单号 服务的一般逻辑会是怎样的？思来想去，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？ [1赞]

- 业余爱好者 2020-02-26 18:33:53
  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,灭有的话就退出，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。

- 铁生 2020-02-26 17:49:41
  老师不说，头像要按吴彦祖来嘛，这也没换啊，哈哈哈。

- 李东勇 2020-02-26 17:46:26
  我们目前对重复下单不做处理， 过期不支付的话就把该订单关闭了， 加上我们目前不会把订单展示给用户， 似乎没啥问题（不过有隐患）