# 课时11 逻辑漏洞（下）

## 本章学习目标

- 学习支付安全相关漏洞
  - 溢出
  - 精度问题
  - 负数问题
  - 修改参数
  - 支付接口问题
- 条件竞争漏洞
- 重放攻击漏洞

上节课讲了一些常见的逻辑漏洞，今天接着来介绍一个比较经典的逻辑漏洞场景叫支付安全。支付安全也是现在网上电商，还有一些网银这种比较经典的一个场景，可能每个人都会在网上去买东西购物，然后也会有一些小的平台做的不是那么好的，就会存在一些安全问题。

支付安全里面我们举几个例子，溢出精度问题 负数问题 修改参数 支付接口问题，然后除了这个以外，再讲另外两个，就是跟支付安全关系稍微比较紧密的漏洞，一个是条件竞争漏洞，一个是重放攻击漏洞。

# 支付安全概述

**支付**场景是网站应用的一个重要的业务场景。围绕支付场景下的漏洞层出不穷，这就引出了对支付安全的讨论。

**支付安全**是讨论围绕支付以及其相关功能安全性的一个话题，凡是与金钱相关的问题都不是小问题，所以在整个网站体系中保障支付安全显得尤为重要。

**支付安全**方面的漏洞很多，其大致包括了：**溢出**、**四舍五入**、**负数问题**、**修改参数**、**支付接口问题**、**条件竞争问题**、**重放攻击问题**等。其中**条件竞争问题**、**重放攻击问题**还经常出现在许多非支付安全的场景所以我们将它们单独讨论。

我们先来看一下什么是支付场景。支付场景主要是讨论支付购买东西的一个场景。在这样一个单独的场景下面，去看它可能会有哪些安全问题。因为支付他跟金钱相关，所以支付安全方面整个业务安全里面讨论的一个重心。曾经历史上曝光过很多漏洞，有溢出、四舍五入、负数问题、修改参数等列到的这些问题。
然后还有一些是除了支付安全以外，其他的也有的，比如说条件竞争问题，重放攻击问题，这些是在不光支付领域，其他领域也会有，我们把放在这个里面来一起介绍一下。

# 一个典型的支付场景


然后介绍一个经典的支付的一个场景。一个在线购物商城，这里面商品有三件，都是京东卡。

假如我们来到了这样一个商城，第一步先登录一下，然后购买。这时候会访问到这样几个数据，价格一般都是写死的，可以选个数，填收货地址，选择是否使用优惠券，然后提交。这个订单就生成了，然后我们去支付，点完支付按钮可能跳到微信支付宝，或者是跳到网银银联，在线支付，发货收码，订单完成。
在这个场景下它有哪些逻辑漏洞，我们今天主要就讨论这个话题。

## 支付安全：1.溢出

溢出：杯子里的水满了，如果继续加水，水会溢出出来。相同地，在许多计算机语言的数据类型中，整型变量拥有一个最大值和一个最小值，当变量已到达最大值/最小值，继续增加/减少会造成整型变量溢出。溢出攻击最早出现于二进制安全领域中。

超过最大值的溢出称为上溢，超过最小值的溢出称为下溢。

有符号整型： signed int 取值范围：（2^（n–1））~ +（2^（n–1）– 1）

无符号整型： unsigned int 取值范围：0 ~ （2^n – 1）

除了整型数据以外，浮点数、double等类型也会出现溢出问题。当变量发生溢出时，往往不能按照预先的逻辑继续执行，这时就可能会存在漏洞，例如：
购买商品时，想办法造成价格溢出，从而花较少的金额购买到大量商品
也可以想办法造成余额下溢，这种情况往往在使用余额成功购买到超过用户当前余额的商品时会出现

首先我们来看一下溢出，溢出是一个很早就有的漏洞，这种漏洞最早是在二进制中出现的。它是什么意思？假如说我们现在有一个杯子，然后用它来装水，水满了我们还往里面继续在倒水的话，这时候水就会从杯子口蔓延出来，就像就溢出了对吧？就是一个东西它已经满了，然后我们如果再继续增加，它就会溢出。相同的在计算机里面我们知道很多编程语言都有一个叫数据类型的东西，它会有不同的变量，每个变量有自己的数据类型，比如说C语言有int整型，int整型有一个最大值和一个最小值。当这个变量已经到达了最大值我们还继续的增加，它会继续减少，它就会超过它最大值，当它一旦超过最大值的时候，这就造成了一次溢出，然后这溢出可能会让它从最大值重新变回为最小值，或者是从最小值重新变回最大值，这个过程就是溢出。
为什么会产生这样的变化带来安全问题呢？是因为在程序员写代码的时候很容易忽略一些递增或者递减的场景，没有考虑到这个东西一天一天增加，可能会叠加到一个最大值场景，也没有考虑到，如果它减小到一个不能再减小的一个地方，如果这时候通过一些技术手段让他继续减小，会不会引发一些新的问题。
这里是给了一个这种范围，一般来说如果是32位的这种就是二的32为32次方，如果是一个有符号的就有正有负的，一般来说整数的最大值就是二的32-1次方，再减一一般是它上面最大的一个边界，下面边界是-2的31次方。除了整形数据以外，浮点数还有double这种类型都会出现问题，然后当溢出发生的时候，他就不能按照正常逻辑去执行了，这时候有可能会出现漏洞。

想一下购买商品的时候，我们能不能想办法让价格造成一个溢出，我们买很多件商品，然后这商品乘以它的价格之后，这个数据变得很大很大一个数，然后超过了它价格的最大值，超过了存储最大值的范围他可能就突然变成一个负的数，或一个很小的一个数，我们就花了很少的钱买了很多东西，当然这样的很多商品它也不可能发货，但是这确实是存在一个安全问题，如果我们合理去利用的话，就可能会造成一些现实生活中遇到的这种真实的发生的这种场景，或者是造成一个shell。

## 支付安全：1.溢出


思考以下C++代码是否会打印 "exit" ？

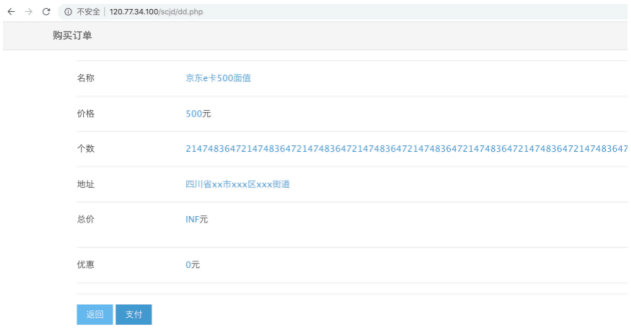
```
#include<iostream>
using namespace std;
int main()
{
    for(int i=1;i>0;i++)
    {
        cout<<i<<endl;
    }
    cout<<"exit";
    exit(0);
}
```

然后我们来思考一下下面的C++代码是否会走到退出exit这一行。这段代码解释一下，很简单，前面两行不用管，然后这是一个主函数，然后这里面就是一个循环，这个循环是先定义了一个I这样一个变量，这个是他这个循环的一个判断的依据，当A大于零的时候就一直去循环的跑，然后每一次I都会递增一。

当i一直加，加到最大值，2的31次方减一，再加一的时候，就会变成-1。它是小于零的，然后就会退出。这就是一个溢出的情况。这种情况就是好多开发人员可能觉得我就一直增，它始终在这样一个循环，就不会到最后，但其实它是会存在溢出的。

# 支付安全：1.溢出

**溢出**：购买很大数量的商品时，金额出现溢出问题。




看一下购买平台的例子，我们来买一个东西，假如说这里个数，我们填一个很大的数，然后提交。可以看到这里价格出现了一些问题，就是他还是能算出价格，我们可以看到这时候总价已经显示不出来了。这种漏洞危害都是比较严重的，因为它直接跟金钱相关的，能够通过去获得一个比较大的利益的时候，都是是一个很严重的问题。

# 支付安全：2. 精度问题

**精度问题**：对于浮点数精度的取舍，如果业务逻辑代码前后不一致就很有可能出现问题。

**向上取整**：当小数部分不为0时，取整时舍去小数部分，保留整数部分并将该整数加1

**向下取整**：取整时舍去小数部分，保留整数部分

**保留精度**：取整时保留系统允许的精度，一般为两位小数（支付金额通常精确到0.01）

**四舍五入**：取整时按保留精度进行四舍五入

例如：
1.在计算金额时，商品数量*单价，保留两位小数，而计算数量时使用四舍五入。
2.在计算金额时，商品数量向下取整*单价，而计算数量时保持精度或四舍五入。

然后第二类问题我们再来讨论一下精度问题。精度问题是这样的，对于有一些浮点数小数的场景，业务处理可能会存在一些这种取舍。比如说我们的小数过长，他肯定要保留一个精度，保留一个两位小数，或者保留一个三位小数，但是它总是要取舍。因为它小数如果过长的话，就没办法把它全部存储下来，所以这里就要保存精度。

系统允许的精度一般是两位小数，对于支付的这种场景，金额我们都是精确到一分。
这个精度的问题主要是由于业务两端前后取法不一致造成的，有些取得是向上取整的，有的是向下取整，还有一种是四舍五入。这些取证方式怎么取都没关系，但是一定要统一，不统一就会出现问题。
比如说在计算金额的时候，我们的商品计算价格说是商品数量乘以单价，保留两位小数，买个1.5个商品，这时候商品数量乘以单价就是1.5个商品，计算的数量的时候又用了四舍五入，我们就买到两个商品。
还有就是在计算的时候商品数量是向下取整的，我买一个时候他把变成数量变成0.5个，我们买1.5个商品的时候，这时候这个商品是向下取整，就变成用一来乘以单价，然后但是在取数量的时候，又四舍五入了，就买到两个，就花一个商品的钱买到两份商品。


我们来看一下这个例子，同样是买京东卡，这时候我们就买一个1.5个，提交之后，价格就计算出来是750。京东卡买1.5个，不管向上取整还是四舍五入就变成了二，我们就少花了250块钱，用这样的一个方法这样重复买我们就能刷很多钱。这是一个相当于一个刷钱的一个漏洞。这种问题即便是在真实场景中间，我们在含在某个地方测试出来的话，因为没有授权也是违法的。类似的举个例子，有一个ATM机出了问题，然后大家都去那边直接去拿钱一样去拿钱的，这些人其实都是违法，虽然是ATM机本身的一个bug造成，但是你通过这个漏洞去获取的利益。这种测试需要有一个授权。

# 支付安全：3. 负数问题

卡亭科技 CHAITIN

**负数问题**：用户输入的数量、金额等可能会存在负数，此时应当对业务逻辑进行严格判断，避免负数影响业务正常逻辑，进而造成漏洞。

例如：
1.购物车中的商品正负相抵，支付时支付较少金额即可付款成功，而负数商品在商家界面不做显示。
2.负数商品购买造成余额增多或积分增加。


第三类是一个复数问题，复述问题就是我们在购买的时候，我们试一下用负金额付款，看看是否会存在一些问题。之前曝光过也是乌云上的一个案例，某一个商城里面买手机，iPhone卖4999，把一个数量为-1的iPhone和20块钱手机壳加入购物车，去提交订单，展示出来也是-1个iPhone和手机壳，总账单20元。然后到商家那边，负的东西就直接被拿掉了，商家就看到只看到了一个iPhone和一个手机课，就去发货了，就等于说他只花了一个手机壳钱就买了一个iPhone和一个手机壳。包括我们有些平台使用一个付金额去购买的话，我们的可能因为负数的这种订单是无法支付的，零元的订单也是无法支付的，必须是最少一分钱订单才能传到银联的接口去支付。但是有些平台是这样的，如果是零元的订单也是可以支付成功的，它不会调用第三方接口，他就直接会提交成功，还有一些平台负数金额的这种订单也是会提交成功的，提交成功以后不会去让支付，然后反而会给你余额增加一些，这都是一些bug。

# 支付安全：4. 修改参数



**修改参数**：支付环节中涉及许多参数，当系统逻辑校验不严格时，可通过修改某些参数值来引起不当得利。例如： 金额、优惠券、数量、积分、运费、币种、余额、付款账号、分期付款期次、支付奖励金等

例如：
1.直接修改价格参数，0元购车，1分钱充值100元话费。
2.修改币种，将货币类型改为日元，在金额不变的情况下使支付实际金额减少。
3.将运费修改为负数。
4. 修改分期付款期次，改为无限大。

然后我们再来看下一种，刚才我们提到的都是输入的问题，比如说输入很大的数，输入很小的时候，输入一个精度的有问题的数，输入一个负数。我们看在哪些地方可能会存在问题，支付里面所涉及到的参数有很多，只要是允许我们去修改参数的地方都是可以去修改的，比如金额和数量，优惠券有时候也是可以修改的，比如说满100-20，我们能不能修改成满100-99，能不能买100-200？还有积分，有些时候我们用钱加积分是可以打折来买东西，那积分这里能不能修改？有些限制了你一个商品就只能用20的积分，你能不能用50积分用一百积分，还有就是运费也可以修改，有些运费是你修改成一个负的运费，这时候你的总的价格可能就会减小，还有币种，这个遇到比较少，但是也有去修改货币币种，货币类型改成日元，然后包括分期付款的期次，然后一些分期付款你可以分很多期，一期一个月你把它分成很多无限大，1001000万的这样一个期次，每期不到一块钱，每个月只要一块钱就可以买一个手机什么的，分期付款其次这也是可以修改。还有一些支付奖励金，比如说你付完款之后他奖励你多少钱，这个钱有些也是可以修改的。修改付款账号，一般都是用余额直接付款，因为你修改成别人的账号，如果要他去交易再去用支付宝或者微信支付的这种，他肯定能看到，如果是用余额直接从他账号里面扣钱，这种就可以去实现，还要修改我们自己的余额。这都是可以修改的地方，我们可以去注意在支付时候有哪些参数，然后哪些地方是可以去修改的。

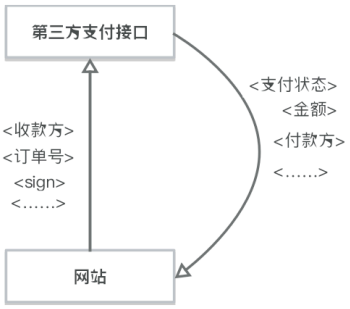
# 支付安全：4. 修改参数

修改参数：直接将金额改为1元，购买商品。


然后这里是把这个金额通过抓包发送改成了1元，直接去买一个一块钱去买一张500的京东卡。

# 支付安全：5. 支付接口问题

支付接口问题：支付时，网站一般回调用第三方支付接口进行支付，当付款成功以后，第三方支付接口一般会回传支付成功的信息。在这个过程中如果缺乏合理校验，就有可能出现问题。




1.自定义支付接口
2.替换返回状态
3.支付订单仿冒
4.修改支付金额
5.修改收款人信息

然后下一个问题是支付接口问题。刚才我们看到这些支付环节，都是我们在和商家做交互的时候存在的一些问题，而支付接口是商家去跟第三方的支付接口的一个问题。
买东西的时候，画一个图，普通用户要去访问这个网站，要去买东西的时候，我们是请求到这里，然后当我们的订单已经下单已经成功了，然后这时候其实是网站他去和第三方接口做一个通信，它把我们的订单和帮我们订单和一些信息，比如说商品的信息价格和我们的用户名，然后网站的收款方信息做一个统一的绑定，还有包括订单号签名，最后还要做一个签

名，签名作用就是防止中间修改。
就是这个东西，它有好多参数，这时候如果这中间我把这个地方篡改了，怎么样识别出来，那就把这些东西一起做一个签名，做一个签名，如果中间这些东西都修改了，这签名也是要随之而变的，但是这个签名的算法我们很难去生成的话，他每次都是这样，我们中间就不能修改这些参数，所以要做一个签名，把这个东西网站把这些东西打包好，把这些要提交给你，打包好发给我们的用户，发给我们用户，然后让我们用户去到第三方接口来去提交订单，然后用户去提交这一串东西，这个东西在比如说微信支付，就在这微信接口上去支付完之后，我们直接付钱，支付完之后，是从第三方支付接口返回给网站，它把支付成功的专栏信息返回给网站，然后包括支付是否付款成功，实际付了多少钱，然后付款方式收款方式等等这些信息都会在返回给网站，网站一就是服务器依据这些信息来判断你是否付款成功，然后再返回给你，如果付款失败，就反而显示给你一个页面就是付款失败，如果付款成功可能就返还给你的页面已经付款成功了，这样一套一个流程。

- 在这个流程中可能会存在哪些问题，我们来看一下。这时候网站它调用的是第三方的支付接口，有些网站它可能会支持很多个支付接口，比如说它支持微信的支付，支持支付宝的，支持银联的，各种网银的。我们自己能不能做一个这种支付接口，假如说他不做限制的话，我们自定一个支付接口，让他修改完以后，让他到我们自己的支付接口去支付我们自己的支付接口，我们都是可以操纵的，然后这时候就也显示支付成功了，返回给网站这样一个状态，支付成功，这是第一种，就是支付接口这块可能会存在这种自定义的支付接口问题。
- 还有就是替换返回的状态。这个过程虽然是支付接口直接到直接返回给网站的，按正常逻辑是支付接口应该直接返回给网站，但是有一些设计的不合理，他会先给用户，然后再由用户到这个网站，让用户再去提交给网站，这时候就出现问题，这是比较早期的一个设计上的缺陷。但是除了这个以外，就算他直接返回给网站也是可能会存在问题的，现在就是有一些不过这种事比较新的这种技术，就有一些东西他现在在研究的有一些APP的这种存在着，比如说APP支付的到第三方接口通信的过程中有可能会遭遇到劫持，在这个过程中一旦清楚了第三方支付的服务器以后，他就可以预先的往这里面去插这个数据包，就是让它始终去插入一个返回的一个状态，就告诉这个网站已经支付成功了。相当于是伪造支付接口去给这个网站发一个请求，然后告诉这个网站已经支付成功了，替换了返回状态，其实你这边不用真正付钱，然后这个网站也能识别出你这个订单已经付款成功了，会有一些这种案例来去替换放回的状态。
- 还有支付订单，仿冒你在这个网站下单，网站会把这些东西打包以后返还给你，你下两个订单，一个订单是正常的，你买一千块钱的京东卡，还有一个订单，你买一个一块钱的一个东西，生成两个订单，生成两个订单以后你去付款的时候你付一块钱的订单，到这个接口付付完之后返回过来应该是一个付款成功的一个状态，网站有些情况它就可能会识别成你已经把一千块钱的订单给付款成功了，这是他一个混淆，你去用这一块钱的支付成功的状态去仿冒一千块钱的付款，这个问题根源在于网站它校验返回的时候可能没有校验金额，现在好多网站去校验支付装都只交易支付状态。
- 还有一个是修改支付的金额，就是在这个环节中返回的金额是否允许被修改？如果是从这过来到用户这边，在用户提交这个金额就可以，你可以修改你的充值的金额，还有修改收款人的信息。这里收款方假如说签名被破解了以后，你可以改。我刚才说网

站打包信息，然后交给用户加了一个签名，交给用户用户去提交，在这个过程中，假如说这签名被破解了，用户就有可能去修改这些修改支付的金额，修改收款人的信息，然后修改收款人的信息，会造成什么一样的，你支付给自己，然后订单收款方式你自己，而不是网站的拥有者，这时候把钱付给了自己，然后第三方接口一样返回给网站，一个支付成功的这样一个状态，就会存在问题。

第三方支付接口相关的漏洞其实是有很多的，而且在最近几年的研究里面发现，APP的场景下面支付接口可能遇到问题的更多。

## 条件竞争漏洞


**条件竞争漏洞**：当同一资源拥有多个申请占有者时会出现竞争性问题。

思考以下两个问题是否存在逻辑漏洞：

1. 电商平台买卖商品时，设计代码逻辑：在购买前判断商品剩余数量是否大于0，若是则继续售出商品。

2. 用户使用积分兑换商品时，在兑换前判断用户积分余额是否大于本次所要消耗的积分，若是则允许兑换。

我们刚才把关于支付方面的一些这种支付场景下面的一些问题总结了一下，然后下面我们来看一个比较新型的一个漏洞，条件竞争漏洞，这个漏洞出现其实并不是很早，最近几年这个漏洞会变得很火。在业务安全领域里面的条件竞争，就是同一个资源拥有多个申请者是同时占用的时候，就会出现一个竞争性问题，比如说我这时候就剩一件商品了，然后有两个人同时都要购买，我应该把商品分配给谁？火车票抢票的时候也是这样，假如说这趟列车车次的火车票只剩一张了，我们在这边买和那边在上海有一个人同时也在买这张火车票，应该卖给谁？这也是一个问题。
然后我们来考虑一下以下两个问题，是否存在逻辑漏洞?

- 第一个是某个电商平台买卖商品的时候，代码逻辑是这样写的，在购买前判断一下商品数量是否大于零，如果是大于零就继续卖这个商品。
- 第二个是用户使用积分兑换商品时的判断，用户积分是否大于余额，如果大于就是是否大于本次所需要消耗的积分，如果是就允许兑换。

这两种设计的方式是否存在逻辑漏洞？第一个可能同时有两个人正好都判断了这个，但是又同时下单，然后就导致减一下商品数量变成了-1。第二个，用户使用积分兑换商品是在兑换

前判断用户余额是否大于本次要消耗积分，当两个客户端卡在同一秒钟下单，只要这个时间就要小于那个服务器的代码的判断时间，就是服务器代码里面它的逻辑就是先判断，然后再去允许下单。如果发送请求的速度比它还快，同时发送两个，就产生了两个订单，用一件的积分购买到两件商品。

## 条件竞争漏洞


条件竞争漏洞的防御需要在数据库中添加事务锁。在考虑线程、进程并发时如果有需要，应当添加线程锁和进程锁。

通过条件竞争漏洞实现“一券（优惠券）多用”。

那么造成条件竞争就漏洞为什么会出现呢？这个问题归根结底是网站在设计的时候，没有考虑到并发的情况，不同的请求过来，他都会分配一个线程资源去处理。现在有一些不止一个数据库，叫分布式数据库，就是很多个数据库，他们数据会定期同步一下，但是他们处理的时候也是分开的，这时候也会造成这种竞争性问题。
然后这种问题应该怎么去处理？大家都已经意识到这里面可能会存在一个逻辑的bug，但是怎么去处理这个漏洞？这种漏洞需要在数据库中去添加事务锁，它可以防止这种竞争性问题。
然后我们来介绍一下，什么是事务锁？当两个请求同时过来，他们都通过了逻辑层验证都到达了数据库，他们就同时要请求这一个资源了，事务所处理原则就是这样的，在数据库操作的时候，第一条处理语句来的时候，在执行之前就加一个锁。这两个总会有一个稍微快那么一点先加上锁，然后等他处理，就把它整个的所有的数据操作都放在这个锁之内，等他用完之后再把锁给解开再解开，然后再让第二个去操作这个数据。这样的话，当这个锁第一个使用完之后，解开之后，第二个再去使用的时候，它资源就已经没有了，就不会再出现这种问题了。

# 重放攻击漏洞



**重放攻击漏洞**：攻击者可以通过记录请求数据并多次重新提交实现重放攻击，当网站未对重放攻击进行合理防护时可能会出现逻辑漏洞。

例如：
1. 重放退款请求造成多次退款
2. 重放投票数据造成重复刷票
3. 重放订单下单操作，锁定大量商品资源

有时，重放攻击还与条件竞争漏洞结合，通过短时间内多次重放来造成具体危害。

---

下一个是重放攻击漏洞，重方攻击它本来是一种攻击，一般来说就是我们经常提到的这种暴力破解，但是为什么又加了一个漏洞，就把他站在一个漏洞的角度，网站可能缺乏对重放攻击的这种防护，造成的一一种漏洞。
重放的很容易理解，就是我把相同的请求多次发再发一遍，或者再发多几次，然后这种重放的这种场景会出现在哪些情况下：

- 比如说在退款的时候，我们能不能多发几次退款的请求，有些网站这样他发一次退款请求之后，订单可能并没有马上就销毁它，可能是等后续的操作，比如说把等商品退还入库，或者把钱打过来之后，这个订单才会消费，在这时候我们一直去利用一直去重放的话，就可能造成了多次退款的这种请求。然后对于服务端来说，他可能就是没有又没有校验到是否是在重复请求同一个订单，可能它统一都审批了都允许退款，这时候就一件商品的钱就退了多次。
- 然后还有对于这种投票来说，也有可能会去重放，就把投票的数据抓下来之后一直去提交，看是否可以允许重复的去刷票。抢票也是这样，有些黄牛可能他也不去买，他就一直去重放把这订单下了很多次，锁定资源，当有人就是愿意买黄牛票多花钱的时候，他就把这订单在付钱再去买，可能也是利用这个方法来去售票。
- 然后有时候重放还与条件竞争漏洞结合，就是在短时间内去多次重访。因为有些情况你重放，可能比如说我们刚才说退款这个场景，你可能从退款发起之后你再发起，这时候就已经来不及，那个系统就已经把钱这个订单就已经销毁了。如果我们利用短时间内一下子来多次重访，这时候服务器判断的时候订单的状态，可能两个请求该就刚才那个图2个请求同时到达，然后他服务端都会判断这个订单是否存在，然后制定单确实都还存在，然后就允许这两个请求同时往后发，然后就生成了两个退款动作。

今天就介绍到这里了。