电商项目介绍：

我们的项目主要分为前台购物和后台管理两部分：

前台购物包括：商品浏览、查询商品、订购商品、购物车、个人用户信息维护等功能

后台管理包括： 商品管理（上架、定价等等）、订单管理 和 会员管理、卡卷中心（折扣）、配送物流管理、评论管理、运营管理（运营概况-新增注册，访问流量等、领取优惠卷、报表类）

整个的业务流程：客户浏览商品 ->选定商品 —> 加入购物车 –>下单 -> 编辑订单属性 -> 支付 -> 订单生成（付款） （这个时候我们会进行订单拆分，价格拆分） –> 物流 –> 收货 完成。当然我们也是可以退款的，退款分为退货退款和仅退款。

****测试的重点****

主要关注订单正确性、支付正确性（人工和数据库数据进行对比，测试支付的算法），单个产品单个订单、单个订单多个产品、以及不同场景的支付（金额足够、金额不足、重复支付、无网支付(断网)、弱网支付、同账号多平台一起支付、余额宝微信信用卡等多种支付方式、不同支付方式的组合）的处理情况，还有各种报表的正确性，主要构造各种场景的数据（修改数据库数据的值，日期），我们根据这些内容分别设计对应的测试用例，保证电商功能的覆盖率。

也会做性能测试，性能测试一般放在功能测试完毕后，系统比较稳定后进行。

****订单模块介绍****

订单一般包括订单编号，订单时间，收货人、收货地址，商品信息（商品名称或其他属性）、商品总价、支付信息（哪种支付方式）、订单状态等，我们主要测试上面的这些字段的内容进行测试：

1) 比如检查订单号是否唯一，是否重复？

2) 分别构造不同的订单状态，对不同的订单状态进行测试，如未付款,已付款,已取消、已发货,已签收,退货申请,退货中,已退货,取消交易等等场景分别进行设计用例

3) 对单个商品、多商品、以及它们的数量

4) 对单个订单（同商户）、多笔 订单（不同商户会拆分多个订单）

5) 不同的支付渠道等等设计用例

6) 以及性能方面的并发测试，还有一些异常方面的测试，比如网络中断啊，服务器停止，页面是否能给出合理的提示等等

支付模块介绍

答：我们测试支付是使用真实支付场景测试的（也就是实际付款了，公司会报销）

主要是针对各种支付渠道、支付场景、合并支付、统计、安全性方面设计对应的支付测试用例

1）支付渠道：包括银联、微信、支付宝等，

2）每种支付渠道设计支付场景：包括正常场景、异常场景、弱网无网场景：

金额足够、金额不足、重复支付、错误金额支付、无网支付(断网)、弱网支付等

3）合并支付：除了测试单笔订单支付外，还要考虑单个渠道多个订单一起支付、不同支付方式的组合等

4）统计功能：还要测试后台的支付报表，要测试月收入报表、季度收入报表、年收入报表、

5）各种支付渠道，平台的分成等 （支付渠道分成大概在总交易金额的 0.6%~1%之间，平台分成稍高，一般在 10%以下）

6）支付安全性：支付是否需要签名认证（支付提交中附带支付秘钥就是签名）、

是否加密传输（这个接口定义能看出来），支付密码是否为掩码，是否可复制，输入错误次数等

支付测试案例:

1. 支付流程测试:
验证用户点击支付后，选择支付方式的过程是否流畅无误。
确认订单金额与应支付金额是否一致。
检查输入密码后是否能成功完成支付。
2. 支付方式测试:
对每种支付方式进行测试，包括但不限于微信、支付宝、信用卡等。
对于同一种支付方式，测试不同的支付入口，如支付宝的扫码支付和网页支付。
3. 金额边界测试:
验证正常金额支付的正确性。
测试最小金额值（如 0.01 元）的支付情况。
检查无意义值（如 0 元）和最大金额值的支付处理。
4. 购买结果测试:
确认支付成功后，产品购买是否成功，例如会员服务到期时间是否正确延长。
验证购买商品后，订单状态和商品种类、数量是否更新正确。
5. 账户余额测试:
确认支付成功后，用户的账户余额是否被正确扣除。
6. 第三方支付接口测试:
测试下单接口、支付接口和退款接口的功能是否正常。
验证前台页面跳转显示支付结果及后台收到支付结果通知的逻辑是否正确。
7. 异常情况测试:
模拟网络不稳定、支付中断等情况，检查系统的异常处理能力。
考虑支付过程中可能出现的各种异常情况，如支付超时、支付失败后的处理逻辑。
8. 安全性测试:
验证支付过程中信息是否加密传输，防止数据泄露。
检查防重复支付和其他潜在的安全风险
9. 支付限额测试：验证支付系统是否正确处理了单笔交易和总支付限额
10.支付失败处理测试：模拟支付失败的情况，检查系统是否能够正确处理并通知用户
11.支付状态更新测试：检查支付成功后，订单状态是否更新为“已支付”。
12.支付回调测试：检查支付成功后，支付网关是否正确回调商家系统。
10. 易用性和兼容性测试:
检查支付界面是否友好，操作是否简便易懂。
在不同设备和浏览器上测试支付流程，确保功能正常可用。
11. 性能压力测试：在高并发情况下测试支付系统的稳定性和响应时间。

订单状态不同步，那怎么把错订单状态改回来?

开发写一个定时任务，每分钟拉取前 30 分钟的失败订单，到我们的 mq 里面，这时候 mq 会去处理这些消息，进行和三方查询接口进行对账，如果三方那边是成功的，那么会及时更正状态做的实时解决，后期添加性能测试对之前的异步回调处理的 mq 进行压测，测试其消息最大处理能力以及最大并发能力，并去优化达到预期值

它的原理其实就是。开发去请求去请求三方接口。然后通过三方接口去确认状态，然后去更正状态。

支付幂等性：

幂等性问题的触发场景（为什么会出现重复请求？/支付幂等性问题一般是怎么产生的？）

用户操作层面：用户多次点击“支付”按钮。

前端层面：前端为防止超时自动重试。

网络层面：网络延迟或超时，客户端重复发送请求。

第三方回调层面：支付渠道（如微信、支付宝）在未收到确认时自动多次回调。

消息队列层面：消费者异常，消息被重新投递，导致同一请求被处理多次。

未做幂等控制的后果？

重复扣款、生成多条相同支付记录、重复写入订单日志、重复回调通知、重复发货或发券

面试常问（what、why、how）：

什么是幂等性？

幂等性指的是同一个请求被重复执行多次，**系统的结果保持一致，不会因为重复执行而改变最终状态。**
例如支付接口，多次重复请求，最终只扣一次钱、生成一条支付记录。

为什么支付接口要保证幂等性？

在支付场景中，由于用户重复操作、网络重试、第三方回调或消息重试等情况，同一笔支付可能被多次请求。如果接口没有幂等控制，可能会导致重复扣款、重复发货、重复写日志等问题，因此必须保证支付的幂等性，保证相同请求只被处理一次。

你们公司/项目的幂等性是怎么实现的？

因为我们支付请求量较大，考虑性能与一致性，我们使用了订单号幂等 Key + 状态校验的方案实现。每笔支付请求都带有一个唯一的订单号（oreder_id）作为幂等 Key，后端收到请求时，会先查询该订单状态：
如果已经处理过，（状态为“已支付”），直接返回支付成功；
如果未处理，则执行支付逻辑并记录状态。
同时，在数据库中建立唯一索引，防止重复写入。
对于回调接口，也用相同的逻辑校验，保证同一笔订单不会重复处理。

超卖：
超卖问题的触发场景：

在高并发场景下，多个请求同时扣减库存，导致库存数量被多次扣减，出现卖出数量超过实际库存的情况。

为什么会出现超卖？（Why）

主要因为库存校验与扣减不具备原子性，如多个请求并发读取相同库存，数据库未加锁、缓存不同步、重复消费等。

怎么解决？（多种解决方法）

缓存层（Redis）控制

1. Redis 原子扣减，Redis 自带原子性，天然避免并发超卖。
2. 分布式锁（SETNX + EXPIRE）给库存 Key 加锁，处理完再释放；（适合库存较小、操作较重场景）
3. 预扣减 + 异步校验：下单时先扣 redis 库存，再异步写数据库，失败则回补库存

唯一请求 + 幂等控制

防止同一用户重复请求扣库存；搭配订单号、请求号、Redis 记录已处理请求

库存预分配

大促/秒杀前将库存按用户分片或预分配，防止竞争；比如：库存=100，分成 10 份，每份 10 个，分组处理。

你们公司/项目的超卖是怎么实现的？

我们项目采用多层防护的方案来防止超卖：首先在缓存层通过 **Redis 原子扣减** 实现高并发下的库存控制，保证每次扣减操作原子性；同时对关键商品使用 **分布式锁**（如 Redis SETNX）控制并发访问，防止极端情况下的竞争写入；最后在数据库层以 UPDATE stock = stock - 1 WHERE stock > 0 作为兜底校验，确保即使缓存异常也不会出现库存为负的情况。三层结合，实现高性能、强一致的库存控制，有效避免了超卖。

对于极端情况，还有定时对账机制进行数据修复，确保库存与订单状态一致。

为什么既要用 Redis 扣减库存，还要数据库兜底？只用 Redis + 分布式锁不就够了吗？

redis + 分布式锁可以解决高并发下的并发安全问题，但 Redis 属于缓存层，无法提供强一致性保证。我们通过 **Redis 原子扣减** 提升性能，**数据库兜底校验** 确保数据一致性和安全性，双层保障才能彻底防止超卖。

为了保持性能，我们会以 Redis 为准实时扣减，异步回写数据库，但仍保留数据库层的约束，定期对账修正差异，实现 **最终一致性**。这样既能高性能抗压，又能保证业务安全。