厦門大學

软件工程导论

测试报告

组　　别　3-4 组

组　　员　黄子安 崔方博 范周喆 侯妤欣 徐森彬

2024　年　**1**　月　**11**　日

目录

1 引言 3
1.1 编写目的 3
1.2 项目背景 3
1.3 定义 4
1.4 参考资料 4
2 测试计划实行情况 5
2.1 机构和人员 5
2.2 测试方案 5
2.2 测试结果 9
3 软件需求测试结论 32
4 评价 34
4.1 软件能力评价 34
4.2 缺陷管理 34
4.3 分析与建议 36
4.4 测试结论 37

1 引言

1.1 编写目的

本项目为厦门大学信息学院软件工程专业软件工程系大三上学期《面向对象分析与设计》、《软件工程导论》、《JavaEE 平台技术》课程大作业。本文档编写目的在于介绍课程大作业中的售后模块、服务模块的详细设计。

项目的总目标为开发一个能支持高并发、大负载的电子商城系统（OOMALL）的订单和顾客模块。用户可以使用此系统完成和目前主流电商平台相似的购物流程，如顾客购买商品、领取优惠券、使用优惠券；商家确认订单等。项目选用 Java 代码，由五人进行敏捷开发，并进行软件功能，性能和安全性等方面的测试。

在此软件测试报告中，针对本小组负责的顾客、订单模块进行了进一步的测试。软件测试报告的编写旨在为项目提供全面而系统的测试信息，以评估软件在各个方面的功能、性能和稳定性。通过报告，我们能够向项目管理者、开发团队和其他利益相关方传达测试的进展和结果，准确反映系统是否符合预期的质量标准。此外，测试报告还为发现的问题和缺陷提供详细的描述，以便开发团队能够及时修复。通过对测试覆盖范围、测试环境、测试用例、执行结果和问题汇总等方面的全面记录，测试报告有助于为软件交付提供决策支持，确保最终交付的系统达到用户期望的质量水平。

1.2 项目背景

本项目是厦门大学信息学院软件工程专业《软件工程导论》 《面向对象分析与设计》和《JavaEE 平台技术》三门课程的联合课程设计，委托开发单位为厦门大学 21 级信息学院软件工程系软件工程专业 3-4 小组。

项目测试现阶段主要针对订单和顾客模块进行**功能测试**。

- 项目开发设计参与者：崔方博、范周喆、侯妤欣、黄子安、徐森彬
- 项目客户端用户：有电商系统使用需求的所有用户，包括消费者和商家。
- 项目后台管理人员：电商系统管理员，商家具有部分管理权限。

1.3 定义

- **缺陷报告：**允许用户提交缺陷报告，包括缺陷的描述、复现步骤、期望的行为以及实际的观察结果。
- **缺陷状态管理：**跟踪缺陷的不同状态，如新建、已分配、已解决、已验证等，以便了解缺陷的当前状态。
- **缺陷优先级和严重性：**允许为每个缺陷设置优先级和严重性，以帮助团队确定修复的优先级。
- **分配和责任追踪：**将缺陷分配给相应的团队成员，并跟踪责任人，以确保缺陷得到及时处理。
- **版本和模块信息：**记录缺陷所影响的软件版本和模块信息，帮助开发人员更容易定位和修复问题。
- **历史记录和日志：**记录缺陷的完整历史记录，包括状态变更、评论、修复和验证等信息，以进行审计和追溯。
- **报表：**生成缺陷报表和分析，帮助团队了解缺陷的趋势、修复速度等，以支持决策和改进过程。

1.4 参考资料

- 需求规格说明书 3-4 小组
- 概要设计说明书 3-4 小组
- 详细设计说明书 3-4 小组
- OOMALL 项目 WiKi

2 测试计划实行情况

2.1 机构和人员

测试机构：3-4 组

负责人：黄子安

开发人员：崔方博、范周喆、侯妤欣、黄子安、徐森彬

指导老师：邱明

测试职责分配：

职位	职责
负责人	负责对测试结果进行评估审核
指导老师	对测试的编写和运行进行指导和审核
开发人员	负责编写测试用例，完成单元测试的黑盒测试和白盒测试

2.2 测试方案

对选择的接口采用白盒测试和黑盒测试的方法完成单元测试，对于单元测试需要保证测试前后所有数据不能被实际改变，因此我们使用了 Spring 提供的测试框架，使得在进行数据访问后会自动进行回滚操作，避免产生对持久化数据源的影响，从而保证单元测试的可恢复性。

因为今年 OOAD 选做的模块没有进行集成测试，所以在碰到调用其他的模块时需要使用 Mock 对象来进行模拟即实现桩模块，具体技术上我们使用的 Mock 框架为 Mockito，是一个流行的 Java Mock 框架，用于创建和操作 Mock 对象，它提供了简单且强大的 API，用于设置 Mock 对象的行为，验证方法调用等。

- 白盒测试中，我们关注于**测试接口的内部实现逻辑**，我们根据选定的接口，编写测试用例对接口的指定输入，验证接口的代码逻辑是否被正确的覆盖，其中的函数的调用关系、分支的跳转关系、对应的函数的输出是否符合预期。通过创建 Mock 对象，我们可以模拟控制输入实现桩模块，验证函数的调用关系和分支跳转关系是否正确。白盒测试的结果主要通过 Jacoco 测试的覆盖率体现，方便查看是否有分支或者条件未被覆盖。

- 在黑盒测试中，我们关注于测试**接口的功能**是否符合预期。使用等价类划分和边界值分析来设计测试用例，确保接口能够正确处理各种输入情况。Mockito 在这个阶段同样可以用于模拟外部依赖，使测试更加独立和可控。
  - 等价类划分：将输入划分为不同的等价类，设计测试用例以覆盖每个等价类。使用 Mockito 设置 Mock 对象的行为，确保外部依赖的影响被控制在测试范围内。
  - 边界值分析：针对输入的边界情况设计测试用例，以验证接口在边界条件下的行为。利用 Mockito 保证外部依赖的模拟，使得测试在各种极端情况下也能进行。

  黑盒测试中，采用等价类划分，边界值分析，着重测试每个接口能否完成预期功能；

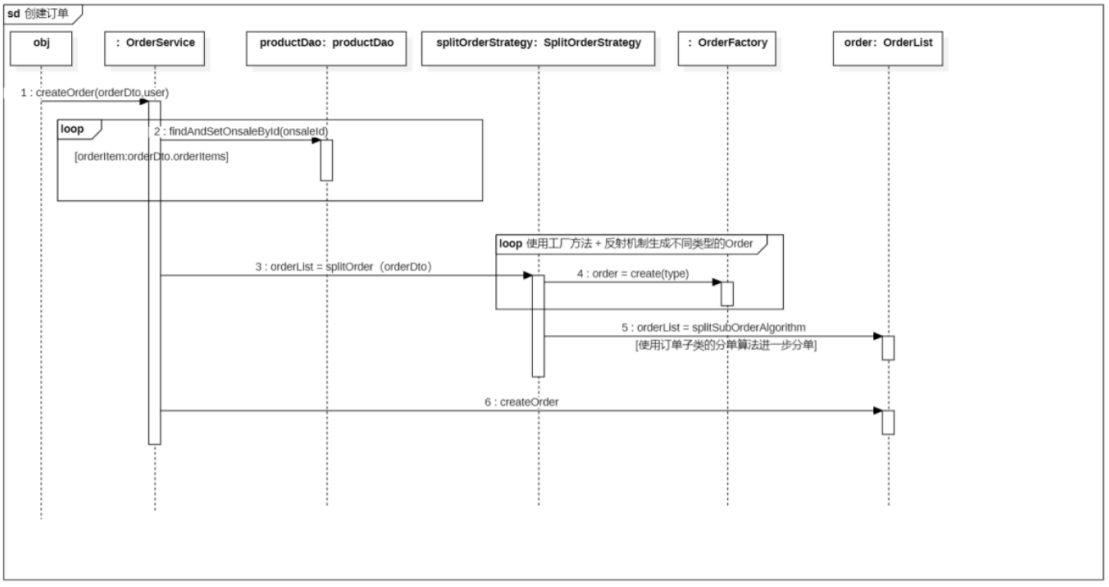
本项目计划采用面向对象的测试方法，测试所编写的类之间的调用关系，继承封装关系是否符合预期。

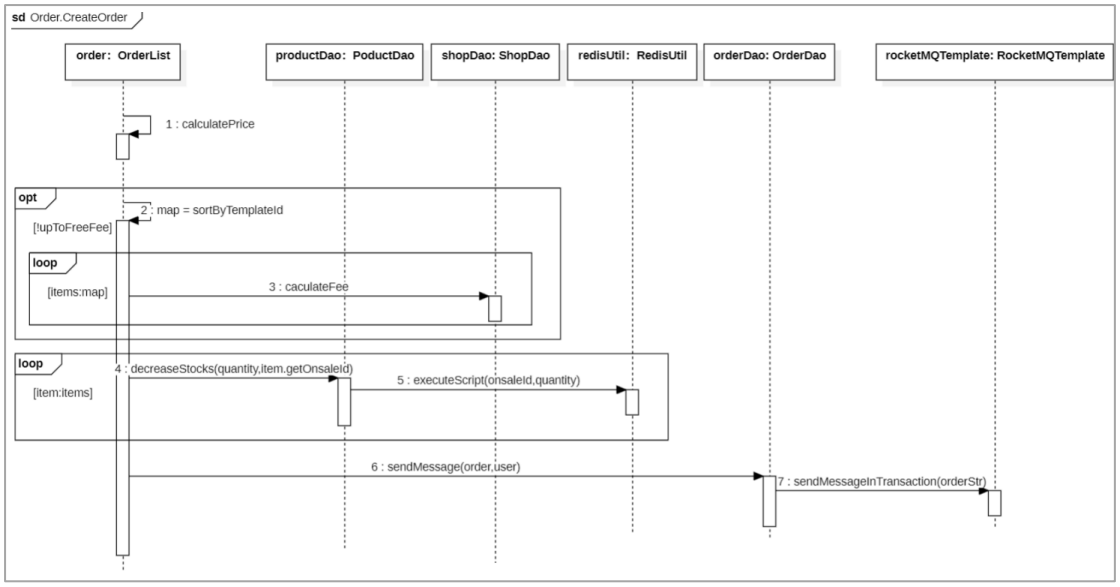
2.2.1 测试功能选取

我们选择的功能为订单模块**创建订单**的功能，订单模块是 OOMALL 电子商城系统的核心的模块，创建订单又是订单模块最复杂的业务，需要调用到顾客模块、产品模块、支付模块、物流模块的内容，所以此接口具有一定的复杂性和代表性，我们选择此接口进行测试。

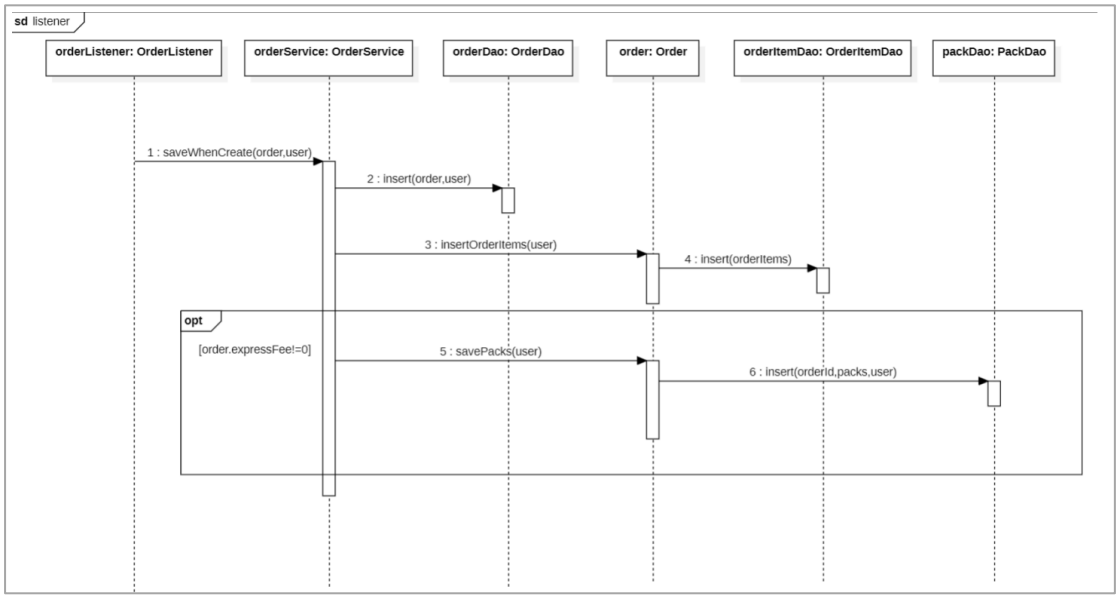
API	请求方式	功能
/orders	POST	创建订单

为了方便文档展示，这里添加下测试接口的执行的时序图（详细细节见详细设计文档），该模块需要用到 OpenFeign 调用下层模块和使用 RocketMQ 发消息给上层的模块，RocketMQ 发送消息为了保证事务需要先发送预订单，之后在 Listener 中完成存储





Listener 中要根据是否包邮了判断是否需要存储包裹



2.2.2 测试用例选取

测试用例的选取应该符合以下规则：

- 执行路径覆盖：

保证所有独立执行路径至少被执行一次，以验证代码在不同条件下的行为。

使用测试用例覆盖每个可能的分支和决策点，以实现代码路径的全面覆盖。

- 逻辑判定覆盖：

确保对所有逻辑判定的真/假情况至少进行一次测试，以验证代码在各种逻辑条件下的正确性。

编写测试用例覆盖逻辑表达式的各个可能取值，包括边缘情况。

- 循环边界和范围覆盖：

在上下边界和可操作的范围内执行所有循环，以测试循环在各种情况下的正确性。
着重考虑循环次数为最小值、最大值和一般情况的测试用例，确保循环正常工作。

- 数据结构验证：

检查代码内部的数据结构是否正确，包括数据的初始化、更新和清理。
编写测试用例验证数据结构的各个方面，确保其符合设计和预期。

- 错误处理测试：

测试程序在出现错误时的错误处理是否符合预期，包括异常处理、错误消息的输出等。
编写测试用例模拟可能的错误场景，确保程序能够正确地处理异常情况。

- 接口调用验证：

测试接口之间的调用是否正确，确保模块间的协作能够正常进行。
使用 Mockito 或其他 Mock 框架模拟接口的行为，验证调用关系是否符合预期。

- 性能测试要求：

对性能进行要求，关注测试用例的执行时间，确保程序在可接受的时间内完成。
编写性能测试用例，关注系统在不同负载下的响应时间、资源消耗等指标。

具体的测试用例数据在下一节中和结果一起给出

2.2 测试结果

白盒测试结果：

在设计测试用例后运行 Spring 和 Maven 的自动测试功能并结合 Jacoco 插件即可自动生成代码行的覆盖情况，经过测试后具体情况如下所示：

下图是整个模块的代码覆盖率情况，其中 po 对象是自动生成的，vo 对象用于接受前端输入，这两个的覆盖率不在统计范围内，要求覆盖的主要重点围绕 service 层、bo 对象和 dao 层进行，尤其对于分单算法等部分应保证较高的覆盖率

prodorder

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
cn.edu.xmu.oomall.order.mapper.openfeign.po		30%		5%	289	392	156	252	197	300	7	26
cn.edu.xmu.oomall.order.mapper.po		42%		n/a	73	192	21	192	73	192	3	7
cn.edu.xmu.oomall.order.controller.dto		35%		0%	83	153	23	129	53	123	3	9
cn.edu.xmu.oomall.order.dao.bo		85%		74%	65	448	46	821	25	370	0	32
cn.edu.xmu.oomall.order.controller.vo		35%		0%	25	53	3	45	6	34	0	3
cn.edu.xmu.oomall.order.dao.openfeign		88%		70%	13	58	19	189	3	41	0	6
cn.edu.xmu.javaee.core.util		90%		n/a	10	37	28	266	10	37	0	1
cn.edu.xmu.oomall.order.dao		86%		61%	11	42	19	151	5	33	0	5
cn.edu.xmu.oomall.order.dao.dto		32%		n/a	9	18	2	15	9	18	1	2
cn.edu.xmu.oomall.order.mapper.openfeign		57%		n/a	9	17	12	16	9	17	0	3
cn.edu.xmu.oomall.order.mapper.mongo.po		40%		n/a	7	12	2	9	7	12	1	2
cn.edu.xmu.oomall.order.service.dto		73%		n/a	1	15	1	20	1	15	0	1
cn.edu.xmu.oomall.order.config		94%		50%	1	11	3	44	0	10	0	3
cn.edu.xmu.oomall.order.dao.openfeign.dto		87%		n/a	4	23	4	35	4	23	0	4
cn.edu.xmu.oomall.order		0%		n/a	2	2	3	3	2	2	1	1
cn.edu.xmu.oomall.order.listener		95%		50%	1	5	1	30	0	4	0	1
cn.edu.xmu.oomall.order.service		99%		93%	1	20	1	72	0	12	0	1
cn.edu.xmu.oomall.order.controller		100%		n/a	0	8	0	20	0	8	0	2
cn.edu.xmu.oomall.order.model		100%		n/a	0	5	0	18	0	5	0	1
Total	4,160 of 12,498	66%	331 of 510	35%	604	1,511	344	2,327	404	1,256	16	110

controller 层和 service 层实现了全部覆盖

prodorder > cn.edu.xmu.oomall.order.controller

cn.edu.xmu.oomall.order.controller

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
CustomerOrderController		100%		n/a	0	5	0	13	0	5	0	1
AdminOrderController		100%		n/a	0	3	0	7	0	3	0	1
Total	0 of 88	100%	0 of 0	n/a	0	8	0	20	0	8	0	2

prodorder > cn.edu.xmu.oomall.order.service

cn.edu.xmu.oomall.order.service

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
OrderService		99%		93%	1	20	1	72	0	12	0	1
Total	3 of 366	99%	1 of 16	93%	1	20	1	72	0	12	0	1

bo 对象覆盖率也达到了 80%

cn.edu.xmu.oomall.order.dao.bo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
AdvanceOrder.new PayOrderExecutor() {...}		71%		76%	6	23	4	50	0	10	0	1
Order		94%		74%	22	131	5	237	4	96	0	1
CouponOrder.new PayOrderExecutor() {...}		73%		50%	3	12	2	28	0	9	0	1
GrouponOrder.new PayOrderExecutor() {...}		73%		50%	3	12	2	28	0	9	0	1
AdvanceOrder		81%		87%	2	15	2	37	0	7	0	1
Onsale		67%		n/a	5	33	7	48	5	33	0	1
RefundTrans		74%		n/a	4	27	4	41	4	27	0	1
CouponOrder		78%		83%	1	11	2	28	0	8	0	1
GrouponOrder		82%		90%	1	10	1	27	0	5	0	1
Pack		91%		n/a	1	27	1	46	1	27	0	1
CouponActivity		67%		n/a	1	10	1	14	1	10	0	1
Payment.PaymentBuilder		66%		n/a	2	7	0	1	2	7	0	1
Payment		89%		n/a	1	25	1	37	1	25	0	1
Shop		60%		n/a	1	6	4	11	1	6	0	1
PackItem		57%		n/a	3	6	5	8	3	6	0	1
Activity		70%		n/a	2	7	3	10	2	7	0	1
OrderItem		98%		62%	7	50	2	68	1	42	0	1
Order.new HashMap() {...}		100%		n/a	0	1	0	10	0	1	0	1
Express		100%		n/a	0	8	0	21	0	8	0	1
Order.new HashMap() {...}		100%		n/a	0	1	0	12	0	1	0	1
Order.new HashMap() {...}		100%		n/a	0	1	0	11	0	1	0	1
Coupon		100%		n/a	0	9	0	13	0	9	0	1

Dao 层覆盖率如下所示

prodorder > cn.edu.xmu.oomall.order.dao

cn.edu.xmu.oomall.order.dao

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
PackDao		68%		75%	3	9	9	31	2	7	0	1
OrderDao		90%		50%	4	16	5	72	2	13	0	1
OrderFactory		73%		50%	1	3	3	12	0	2	0	1
OrderItemDao		94%		66%	3	13	2	35	1	10	0	1
PayOrderExecutor		100%		n/a	0	1	0	1	0	1	0	1
Total	83 of 623	86%	7 of 18	61%	11	42	19	151	5	33	0	5

下面对创建订单这个 API 的过程覆盖率进行追踪下

```
31.
32.     /**
33.      * @author huangzian
34.      * 创建订单
35.      */
36.     @PostMapping("/orders")
37.     @Audit
38.     public ReturnObject createOrder(@RequestBody @Validated OrderVo orderVo, @LoginUser UserDto user) {
39.         OrderDto orderDto = CloneFactory.copy(new OrderDto(), orderVo);
40.         orderDto.setItems(orderVo.getItems());
41.         this.orderService.createOrder(orderDto, user);
42.         return new ReturnObject(ReturnNo.CREATED);
43.     }
44.
```

```
52.     /**
53.      * 创建订单
54.      *
55.      * @param orderDto
56.      * @param user
57.      * @author huangzian
58.      * @author Zhouzhe Fan
59.      */
60.     public void createOrder(OrderDto orderDto, UserDto user) {
61.         // 1. 参数校验
62.         List<OrderItem> orderItemList = orderDto.getItems();
63.
64.         //异步查询orderItems所有的onsale
65.         logger.debug("findOnsaleByIdAsync: {}",orderItemList);
66.         this.productDao.findAndSetOnsaleById(orderItemList);
67.
68.         orderItemList.forEach(orderItem -> {
69.             Onsale onsale = orderItem.getOnsale();
70.             // 检查购买数量限制
71.             if (orderItem.getQuantity() > onsale.getMaxQuantity()) {
72.                 throw new BusinessException(ReturnNo.ITEM_OVERMAXQUANTITY, String.format(ReturnNo.ITEM_OVERMAXQUANTITY.getMessage(), onsale.getId(), orderItem.getQuantity(), onsale.getMaxQuantity()));
73.             }
74.             //检查活动Id是否在活动列表里
75.             if (onsale.getActList().stream().noneMatch(act -> Objects.equals(act.getId(), orderItem.getActId()))) {
76.                 throw new BusinessException(ReturnNo.RESOURCE_ID_NOTEXIST, String.format(ReturnNo.RESOURCE_ID_NOTEXIST.getMessage(), "活动", orderItem.getActId()));
77.             }
78.             //设置orderItem的属性值
79.             orderItem.setId(snowFlakeIdWorker.nextId());
80.             orderItem.setProductName(onsale.getName());
81.             orderItem.setProductPrice(onsale.getPrice());
82.             orderItem.setDiscountPrice(onsale.getPrice());
83.             orderItem.setPoint(0L);
84.             orderItem.setDiscountPrice(orderItem.getProductPrice());
85.             orderItem.setCommissionRate(onsale.getCommissionRatio());
86.             orderItem.setDeposit(onsale.getDeposit());
87.         });
88.
89.         // 2. 分单
90.         // 调用分单算法进行分单
91.         SplitOrderStrategy strategy = new SplitOrderStrategy(orderFactory);
92.         List<Order> orderList =strategy.splitOrderAlgorithm(orderDto);
93.
94.         //3.执行创建订单
95.         orderList.forEach(order -> {
96.             this.orderDao.build(order);
97.             order.setCustomerId(user.getId());
98.             order.setId(snowFlakeIdWorker.nextId());
99.             order.setStatus(Order.TO_PAY);
100.            order.setDeposit(0L);
101.            order.setPoint(0L);
102.            order.setRegionId(orderDto.getRegionId());
103.            order.createOrder(user);
104.        });
105.    }
106.    public void saveOrderWhenCreate(Order order, UserDto user) throws RuntimeException {
107.        this.orderDao.insert(order, user);
108.        order.insertOrderItems(user);
109.        //如果没有那么计算了免邮，需要保存包裹
110.        if (order.getExpressFee() != 0L) {
111.            order.savePacks(user);
112.        }
113.    }
114.
115.
```

```
592.
593.     /**
594.      * 扣Redis里的库存
595.      */
596.     private void deductInventory() {
597.         List<List<Long>> backup = new ArrayList<>();
598.         this.getOrderItems().forEach(orderItem -> {
599.             if (this.productDao.decreaseOnsale(orderItem.getOnsaleId(), orderItem.getQuantity()) < 0L) {
600.                 backup.forEach(o -> this.productDao.rollbackOnsale(o.get(0), o.get(1).intValue()));
601.                 throw new BusinessException(ReturnNo.GOODS_STOCK_SHORTAGE, String.format(ReturnNo.GOODS_STOCK_SHORTAGE.getMessage(), orderItem.getOnsale().getId()));
602.             }
603.             backup.add(new ArrayList<>(List.of(orderItem.getOnsaleId(), orderItem.getQuantity().longValue())));
604.         });
605.     }
606.
```

```
547.
548.     /**
549.      * 创建订单
550.      *
551.      * @param user
552.      */
553.     public void createOrder(UserDto user) {
554.         this.setStatus(TO_PAY);
555.
556.         // 1.计算原价现价
557.         calculatePrice();
558.
559.         // 2.计算运费
560.         if (upToFreeFee()) {
561.             expressFee = 0L;
562.         } else {
563.             calculateFee();
564.         }
565.
566.         // 3.子类创建订单时的额外操作
567.         extraOperationWhenCreate(user);
568.
569.         // 4.扣除库存
570.         deductInventory();
571.
572.         // 5.保存订单，在mq listener中实现，向mq发送一条消息，表示我们可以把订单存到数据库
573.         this.orderDao.sendMessage(this, user);
574.     }
575.
```

黑盒测试结果：

给出每一测试项目的：实测结果数据：与预期结果数据的偏差，该项测试表明的事实，该项测试发现的问题。

测试 API：

API	请求方式	功能
/orders	POST	创建订单

测试 API：post　/orders　创建订单

1.　正常情况 1 创建优惠订单

输入：


```
{
    {
      "id": 1,
      "customerId": 1,
      "regionId": 1,
      "consignee": "zhangsan",
      "address": "翻斗花园",
      "mobile": "12345678901",
      "message": "买买买",
      "expressFee": 0,
      "originPrice": 36,
      "point": 0,
      "discountPrice": 32,
      "deposit": 0,
      "type": 0,
```

```
    "payments": [],
    "orderItems": [
      {
        "id": 1,
        "quantity": 4,
        "productPrice": 9,
        "discountPrice": 8,
        "point": 0,
        "productName": "巧乐兹 1",
        "actId": 5,
        "commissionRate": 40,
        "onsaleId": 1,
        "deposit": 0,
        "productId": 1,
        "commissionPrice": 12,
        "depositCommissionPrice": 0
      }
    ],
    "refundTransList": [],
    "status": 101
  }
```


预期输出：


```
  {
    "errno": 1,
    "data": {
      "id":  雪花算法生成的订单 ID
    },
    "message": "创建成功"
  }
```


输出结果：


```
  {
    "errno": 1,
    "data": {
```

```
        "id": 400637398939209728
    },
    "message": "创建成功"
}
```

2. 正常情况 2 创建预售订单

输入：

```
{
  "orderItems": [
    {
      "onsaleId": 1,
      "quantity": 4,
      "actId": 5
    }
  ],
  "consignee": "zhangsan",
  "mobile": "12345678901",
  "regionId": 1,
  "address": "翻斗花园",
  "message": "买买买"
}
```

预期输出：

```
{
  "errno": 1,
  "data": {
    "id":  雪花算法生成的订单 ID
  },
  "message": "创建成功"
}
```

输出结果：

```
{
  "errno": 1,
  "data": {
    "id": 400637398939209729
  },
  "message": "创建成功"
}
```

3. 正常情况 3 创建团购订单

输入：

```
{
  "orderItems": [
    {
      "onsaleId": 1,
      "quantity": 4,
      "actId": 5
    }
  ],
  "consignee": "zhangsan",
  "mobile": "12345678901",
  "regionId": 1,
  "address": "翻斗花园",
  "message": "买买买"
}
```

预期输出：

```
{
  "errno": 1,
  "data": {
    "id":  雪花算法生成的订单 ID
  },
  "message": "创建成功"
}
```

输出结果：

```
{
  "errno": 1,
  "data": {
    "id": 400637398939209730
  },
  "message": "创建成功"
}
```

4. 正常情况 4 创建多种混合订单

输入：

```
{
  "orderItems": [
    {
      "onsaleId": 1,
      "quantity": 4,
      "actId": 5
    },
    {
      "onsaleId": 2,
      "quantity": 4,
      "actId": 5
    }
  ],
  "consignee": "zhangsan",
  "mobile": "12345678901",
  "regionId": 1,
  "address": "翻斗花园",
  "message": "买买买"
}
```

预期输出：

```
{
  "errno": 1,
  "data": {
    "id":  雪花算法生成的订单 ID
  },
  "message": "创建成功"
}
```

输出结果：

```
{
  "errno": 1,
```

```
    "data": {
      "id": 400637398939209728
    },
    "message": "创建成功"
}
```


5. 正常情况 5 创建团购订单，需分单

输入：


```
{
    "orderItems": [
      {
        "onsaleId": 1,
        "quantity": 4,
        "actId": 5
      },
      {
        "onsaleId": 2,
        "quantity": 4,
        "actId": 6
      },
      {
        "onsaleId": 2,
        "quantity": 2,
        "actId": 6
      }
    ],
    "consignee": "zhangsan",
    "mobile": "123",
    "regionId": 1,
    "address": "翻斗花园",
    "message": "买买买"
}
```


预期输出：


```
{
    "errno": 1,
```

```
    "data": {
      "id": 雪花算法生成的订单 ID
    },
    "message": "创建成功"
  }
```

输出结果：

```
{
  {
    "errno": 1,
    "data": {
      "id": 400637398939209728
    },
    "message": "创建成功"
  }
```

6. 正常情况 6 创建预售订单，需分单

输入：


```
{
  "id": 1,
  "customerId": 1,
  "regionId": 1,
  "consignee": "zhangsan",
  "address": "翻斗花园",
  "mobile": "12345678901",
  "message": "买买买",
  "expressFee": 0,
  "originPrice": 36,
  "point": 0,
  "discountPrice": 32,
  "deposit": 0,
  "type": 3,
  "payments": [],
  "orderItems": [
    {
      "id": 1,
      "quantity": 4,
      "productPrice": 9,
      "discountPrice": 8,
      "point": 0,
      "productName": "巧乐兹 1",
      "actId": 5,
      "commissionRate": 40,
      "onsaleId": 1,
      "deposit": 0,
      "productId": 1,
      "commissionPrice": 12,
      "depositCommissionPrice": 0
    }
  ],
  "refundTransList": [],
```

```
    "status": 101
  }
```

预期输出：

```
  {
    "errno": 1,
    "data": {
      "id":  雪花算法生成的订单 ID
    },
    "message": "创建成功"
  }
```

输出结果：

```
{
  {
    "errno": 1,
    "data": {
      "id": 400637398939209728
    },
    "message": "创建成功"
  }
```

7. 异常情况 1 创建优惠订单,但 onsale 是无效的

输入：

```
{
  {
    "orderItems": [
      {
        "onsaleId": 1,
        "quantity": 4,
        "actId": 5
      }
    ],
    "consignee": "zhangsan",
    "mobile": "12345678901",
```

```
        "regionId": 1,
        "address": "翻斗花园",
        "message": "买买买"
    }
```

预期输出：

```
{
    "errno": 106,
    "errmsg": "调用 orders 接口失败",
    }
```

输出结果：

```
{
    "errno": 106,
    "errmsg": "调用 orders 接口失败",
    }
```

8. 异常情况 2 创建优惠订单，但数量超过限制

输入：

```
{
    {
      "orderItems": [
        {
          "onsaleId": 1,
          "quantity": 7,
          "actId": 5
        }
      ],
      "consignee": "zhangsan",
      "mobile": "12345678901",
      "regionId": 1,
      "address": "翻斗花园",
      "message": "买买买"
    }
```

预期输出：

```
{
    "errno": 802,
    "errmsg": "销售对象(id=1)的数量(7)超过单次可购买数量(1)",
    }
```

输出结果：

```
{
    "errno": 802,
    "errmsg": "销售对象(id=1)的数量(5)超过单次可购买数量(1)",
    }
```

9. 异常情况 3 创建订单 缺少 orderItem

输入：

```
    {
      "consignee": "zhangsan",
      "mobile": "12345678901",
      "regionId": 1,
      "address": "翻斗花园",
      "message": "买买买"
    }
```

预期输出：

```
    {
      "errno": 20,
      "errmsg": "服务器配置参数 orderItems 错误",
    }
```

输出结果：

```
    {
      "errno": 20,
      "errmsg": "服务器配置参数 orderItems 错误",
```

```
    }
```

10. 异常情况 4 创建订单 地区 id 不存在

输入：

```
    {
      "orderItems": [
        {
          "onsaleId": 1,
          "quantity": 4,
          "actId": 5
        }
      ]
      "consignee": "zhangsan",
      "mobile": "12345678901",
      "regionId": 9999,
      "address": "翻斗花园",
      "message": "买买买"
}
```

预期输出：

```
    {
      "errno": 4,
      "errmsg": "地区对象(id=9999)不存在",
}
```

预期输出：

```
    {
      "errno": 4,
      "errmsg": "地区对象(id=9999)不存在",
}
```

11. 异常情况 5 创建订单 onsaleid 不存在

输入：

```
    {
      "orderItems": [
        {
          "onsaleId": 9999,
          "quantity": 4,
          "actId": 5
        }
      ]
      "consignee": "zhangsan",
      "mobile": "12345678901",
      "regionId": 1,
      "address": "翻斗花园",
      "message": "买买买"
}
```

预期输出：

```
    {
      "errno": 4,
      "errmsg": "销售对象(id=9999)不存在",
}
```

预期输出：

```
    {
      "errno": 4,
      "errmsg": "销售对象(id=9999)不存在",
}
```

12. 异常情况 6 创建订单 未输入必要参数地区

输入：

```
    {
      "orderItems": [
```

```
            {
                "onsaleId": 1,
                "quantity": 4,
                "actId": 5
            }
        ]
        "consignee": "zhangsan",
        "mobile": "12345678901",
        "regionId": ,
        "address": "翻斗花园",
        "message": "买买买"
}
```

预期输出：

```
    {
        "errno": 20,
        "errmsg": "服务器配置参数 regionId 错误",
}
```

预期输出：

```
    {
        "errno": 20,
        "errmsg": "服务器配置参数 regionId 错误",
}
```

13. 异常情况 7 创建订单，但该商品没有参与该活动

输入：

```
    {
        "orderItems": [
            {
                "onsaleId": 1,
                "quantity": 3,
                "actId": 1
```

```
            }
        ],
        "consignee": "zhangsan",
        "mobile": "123",
        "regionId": 1,
        "address": "翻斗花园",
        "message": "买买买"
}
```

预期输出：

```
    {
        "errno": 4,
        "errmsg": "活动对象(id=1)不存在",
}
```

预期输出：

```
    {
        "errno": 4,
        "errmsg": "销售对象(id=9999)不存在",
}
```

14. 异常情况 8 创建订单 actId 不存在

输入：

```
    {
        "orderItems": [
            {
                "onsaleId": 1,
                "quantity": 4,
                "actId": 9999
            }
        ]
        "consignee": "zhangsan",
        "mobile": "12345678901",
```

```
        "regionId": 1,
        "address": "翻斗花园",
        "message": "买买买"
}
```

预期输出：

```
    {
        "errno": 4,
        "errmsg": "活动对象(id=9999)不存在",
}
```

预期输出：

```
    {
        "errno": 4,
        "errmsg": "活动对象(id=9999)不存在",
}
```

15. 异常情况 9 商品库存不足

输入：

```
    {
        "orderItems": [
            {
                "onsaleId": 1,
                "quantity": 4,
                "actId": 5
            }
        ]
        "consignee": "zhangsan",
        "mobile": "12345678901",
        "regionId": 1,
        "address": "翻斗花园",
        "message": "买买买"
}
```

预期输出：

```
    {
      "errno": 296,
      "errmsg": "货品（id = 7）库存不足"
}
```

预期输出：

```
    {
      "errno": 296,
      "errmsg": "货品（id = 7）库存不足"
}
```

16. 异常情况 10 手机号格式不正确（非 11 位）

输入：

```
    {
      "orderItems": [
        {
          "onsaleId": 1,
          "quantity": 4,
          "actId": 5
        }
      ]
      "consignee": "zhangsan",
      "mobile": "12345678901",
      "regionId": 1,
      "address": "翻斗花园",
      "message": "买买买"
}
```

预期输出：

```
    {
      "errno": 3,
```

```
    "errmsg": “mobile 字段不合法”
}
```

预期输出：

```
{
    "errno": 3,
    "errmsg": “mobile 字段不合法”
}
```

17. 异常情况 11 创建优惠订单，但优惠券已使用

输入：

```
{
    "orderItems": [
        {
            "onsaleId": 1,
            "quantity": 4,
            "actId": 5,
            "couponId": 1
        }
    ],
    "consignee": "zhangsan",
    "mobile": "12345678901",
    "regionId": 1,
    "address": "翻斗花园",
    "message": "买买买"
}
```

预期输出：

```
{
    "errno": 7,
    "errmsg": " coupon 对象（id=1）已使用状态禁止此操作"
}
```

预期输出：

```
    {
      "errno": 7,
      "errmsg": " coupon 对象（id=1）已使用状态禁止此操作"
}
```

18. 权限测试 未登录

输入：未携带 token

预期输出：

```
{
    "errno": 15,
    "errmsg": "需要先登录",
    }
```

输出结果：

```
{
    "errno": 15,
    "errmsg": "需要先登录",
    }
```

19. 边界值测试：购买数量等于限制数量

输入：

```
{
    {
      "orderItems": [
        {
          "onsaleId": 1,
          "quantity": 5,
          "actId": 5
        }
      ],
      "consignee": "zhangsan",
      "mobile": "12345678901",
      "regionId": 1,
      "address": "翻斗花园",
      "message": "买买买"
```

```
}
```

预期输出：

```
{
    {
      "errno": 1,
      "data": {
        "id": 400637398939209732
      },
      "message": "创建成功"
}
```

输出结果

```
{
    {
      "errno": 1,
      "data": {
        "id": 400637398939209732
      },
      "message": "创建成功"
}
```

3 软件需求测试结论

按顺序给出每一项需求测试的结论。包括：通过测试得出的软件能力的结论，局限性（即某项需求未得到充分测试的情况及原因)。

API	功能	测试用例	用例描述	是否通过
/orders	创建订单	1-正常情况 1	创建优惠订单	通过
		2-正常情况 2	创建预售订单	通过
		3-正常情况 3	创建团购订单	通过
		4-正常情况 4	创建多种混合订单	通过
		5-正常情况 5	创建团购订单需分单	通过
		6-正常情况 6	创建预售订单需分单	通过
		7-异常情况 1	创建优惠订单，但 onsale 是无效的	通过
		8-异常情况 2	创建优惠订单，但数量超过限制	通过
		9-异常情况 3	创建订单 缺少 orderItem	通过
		10-异常情况 4	创建订单 地区 id 不存在	通过
		11-异常情况 5	创建订单 onsaleid 不存在	通过
		12-异常情况 6	创建订单 未输入必要参数地区	通过
		13-异常情况 7	创建订单，但该商品没有参与该活动	通过
		14-异常情况 8	创建订单 actId 不存在	通过
		15-异常情况 9	商品库存不足	通过

		16-异常情况 10	手机号格式不正确（非 11 位）	通过
		17-异常情况 11	创建优惠订单，但优惠券已使用	通过
		18-权限测试	未登录，不携带 token	通过
		19-边界值测试	购买数量等于限制数量	通过

4 评价

4.1 软件能力评价

通过缺陷统计的方式全面分析当前接口的能力，我们考察了各项等价类与边界值的测试用例，并且对通过与未通过的情况进行了记录。通过整个测试的过程，掌握了接口在各个测试场景下的表现情况，为项目质量的提升提供了依据。

在创建订单的功能测试中，测试过程中通过了所有正常和异常情况的测试用例，包括有效等价类、无效等价类边界值和权限测试，显示了软件的鲁棒性和稳定性。这些通过的测试用例表明当前接口在各个方面都表现出色，为进一步的优化和发展奠定了坚实的基础。通过缺陷统计，我们能够更全面地了解软件性能，有助于及时发现和解决潜在问题，确保软件在实际应用中能够持续稳定运行。

4.2 缺陷管理

我们需要通过缺陷管理工具对缺陷进行实时的管理和分配，具体的功能和步骤如下：

- 缺陷报告：允许用户提交缺陷报告，包括缺陷的描述、复现步骤、期望的行为以及实际的观察结果。
- 缺陷状态管理：缺陷状态包括新建、已分配、已解决、已验证等，以便了解缺陷的当前状态。
- 缺陷优先级和严重性：允许为每个缺陷设置优先级和严重性，以帮助团队确定修复的优先级。
- 分配和责任追踪：将缺陷分配给相应的团队成员，并跟踪责任人，以确保缺陷得到及时处理。
- 版本和模块信息：记录缺陷所影响的软件版本和模块信息，帮助开发人员更容易定位和修复问题。
- 历史记录和日志：记录缺陷的完整历史记录，包括状态变更、评论、修复和验证等信息，以进行审计和追溯。
- 报表：生成缺陷报表和分析，帮助团队了解缺陷的趋势、修复速度等，以支持决策和改进过程。

具体的项目开发中的缺陷列表如下：

编号	优先级	严重性	描述	复现步骤	责任人
1	高	高	创建订单接口在高并发情况下可能导致系统崩溃	同时发起 1000 个创建订单请求	开发人员
2	高	高	订单金额计算错误	创建订单时选择特定商品，金额计算不正确	开发人员
3	低	低	订单状态更新不及时	支付成功后，订单状态没有及时更新	开发人员
4	高	高	支付回调接口可能存在安全漏洞	发起支付请求时传入恶意参数	开发人员
5	中	中	部分订单无法正常关闭	尝试关闭已支付成功的订单	开发人员
6	低	低	物流信息获取接口响应时间过长	查询包含大量商品的订单的物流信息	开发人员
7	中	中	订单创建时未进行库存检查	创建订单时选择超过库存的商品	开发人员
8	高	高	部分订单支付成功后未生成支付记录	发起支付请求时选择特定订单	开发人员

4.3 分析与建议

1. 创建订单接口可能存在数据库死锁：
- 实施数据库事务管理，确保事务的原子性，减少死锁的可能性。
- 优化创建订单流程，使用消息队列和内存数据库减少压力。
- 考虑采用数据库连接池技术，优化数据库连接的使用。

2. 订单金额可能出现计算错误：
- 增加订单金额计算的相关测试，确保计算逻辑正确。

3. 订单状态更新不及时：
- 修复订单状态更新逻辑。
- 引入异步处理机制，将订单状态更新操作异步化，减少对主线程的影响。
- 使用合适的定时任务或消息队列来处理订单状态的更新，确保及时性。

4. 支付回调接口安全漏洞：
- 增强支付回调接口的安全性
- 使用 HTTPS 协议保障支付回调接口的数据传输安全。
- 对支付回调数据进行严格的验证和过滤，防范恶意攻击。
- 采用令牌验证或签名验证确保只有合法的支付平台可以调用回调接口。

5. 部分订单无法正常关闭：
- 修复订单关闭逻辑。
- 实施详细的订单关闭机制，检查关闭操作的前提条件，并确保关闭流程的完整性。
- 提供管理员手动关闭订单的选项，以应对特殊情况。

6. 物流信息获取接口响应时间过长：
- 优化物流信息获取接口。
- 优化物流信息获取接口的代码，考虑引入缓存机制减少对物流系统的频繁调用。
- 引入并发处理机制，确保多个请求可以同时进行，提高接口的响应速度。

7. 订单创建时未进行库存检查：
- 添加库存检查逻辑。
- 在订单创建前引入库存检查环节，确保订单创建时库存足够。
- 使用乐观锁或悲观锁机制，避免在检查库存和创建订单之间发生不一致的情况。

8. 部分订单支付成功后未生成支付记录：
- 修复支付记录生成逻辑。
- 在支付成功后立即生成支付记录，确保支付记录与支付状态同步。
- 实施支付记录生成的事务，以保障原子性。

4.4 测试结论

在经过细致而全面的测试后，我们充分确认了该接口的功能符合项目需求和设计要求。通过对软件的多方面检验，我们得出了积极而可靠的结论，认为该软件的表现达到了预期标准，并且完全满足用户的需求。其稳定性、可靠性等关键指标均经过充分验证，表现出色，为进入下一阶段的顺利推进提供了坚实的基础。此测试结果为项目的顺利进行奠定了坚实的基础，我们对软件的具体表现充满信心，期待在后续的开发和应用中取得更为显著的成果。