

day12-优惠券使用

现在，用户已经可以页面领取优惠券，并且也能在个人中心查看到自己所有的优惠券了：



不过，新的问题来了，用户购物的时候自然要选择优惠券来使用。而现在主流的购物网站都会有优惠券的智能推荐功能，那么：

- 优惠券的类型不同，折扣计算规则该如何用代码表示？
- 如何组合优惠券使用才能让用户得到最大优惠？
- 优惠券叠加的计算算法是怎样的？
- 如果下单时使用了优惠券，用户退款时又该如何处理？

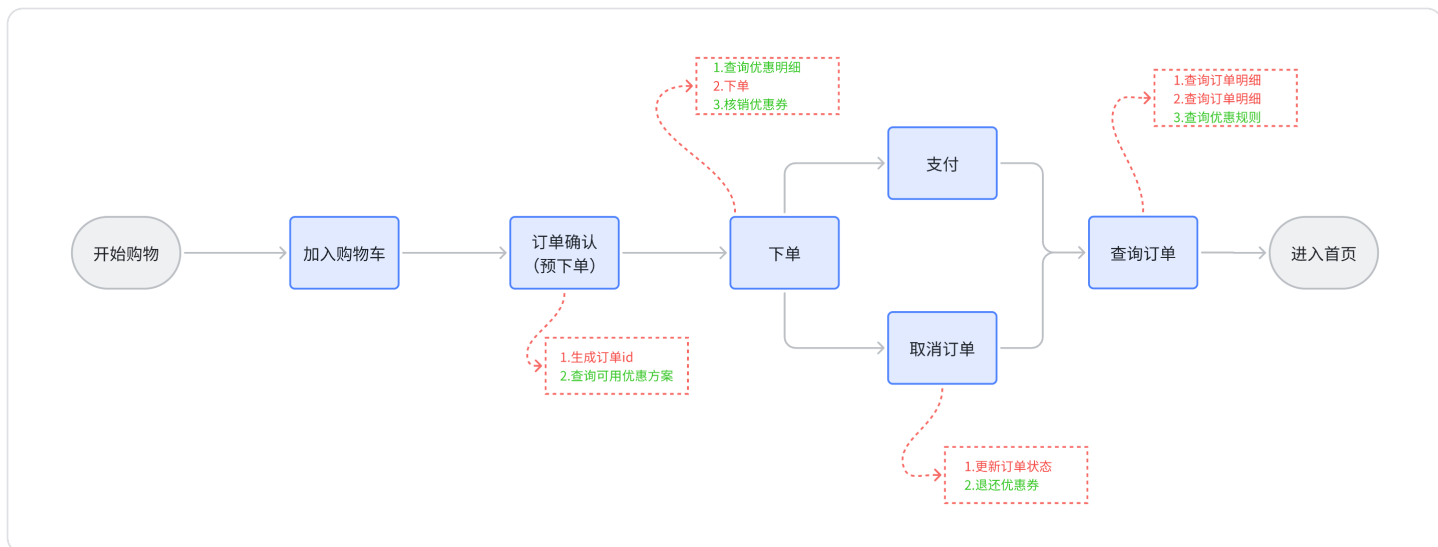
这些问题都能在今天的课程中找到答案。

1.优惠券规则定义

优惠券的折扣规则有各不相同，如何从其中找到共同点，抽象出程序模型就成了首要任务。我们首先从业务流程入手，找到对优惠券折扣的需求，基于需求来抽取模型。

1.1.业务流程分析

优惠券的使用流程就是下单购物的流程，核心流程如下：



与优惠券有关的核心接口有5个：

- 根据订单查询可用优惠方案
- 根据订单和优惠方案查询优惠明细：
- 核销优惠券
- 退还优惠券
- 查询优惠规则

1.2.优惠券规则定义

在订单确认页对需要推荐针对订单的优惠方案，可以看出其中对优惠券的一些功能要求：

① 优惠券是否可用于当前订单？

Java基础课-MySQL数据库安装	¥ 749
永久有效	
Java基础课-MySQL数据库安装	¥ 546
永久有效	

订单总价：¥ 1295

优惠券：满199-50 -¥ 50.00

实付金额：¥1245.00

② 计算订单使用优惠券后的优惠金额

③ 生成优惠券的规则的描述

所谓的优惠券方案推荐，就是从用户的所有优惠券中筛选出可用的优惠券，并且计算哪种优惠方案用券最少，优惠金额最高。

因此这里包含了对优惠券的下列需求：

- **判断一个优惠券是否可用**，也就是检查订单金额是否达到优惠券使用门槛
- **按照优惠规则计算优惠金额**，能够计算才能比较并找出最优方案
- **生成优惠券规则描述**，目的是在页面直观的展示各种方案，供用户选择

因此，任何一张优惠券都应该具备上述3个功能，这样就能满足后续对优惠券的计算需求了。

我们抽象一个接口来标示优惠券规则：

```
1 package com.tianji.promotion.strategy.discount;
2
3 import com.tianji.promotion.domain.po.Coupon;
4
5 /**
6  * <p>优惠券折扣功能接口</p>
7  */
8 public interface Discount {
9     /**
10      * 判断当前价格是否满足优惠券使用限制
11      * @param totalAmount 订单总价
12      * @param coupon 优惠券信息
13      * @return 是否可以使用优惠券
14      */
15     boolean canUse(int totalAmount, Coupon coupon);
16
17     /**
18      * 计算折扣金额
19      * @param totalAmount 总金额
20      * @param coupon 优惠券信息
21      * @return 折扣金额
22      */
23     int calculateDiscount(int totalAmount, Coupon coupon);
24
25     /**
26      * 根据优惠券规则返回规则描述信息
27      * @return 规则描述信息
28      */
29     String getRule(Coupon coupon);
30 }
```

有了规则接口，就需要根据优惠券去实现接口了。不过优惠券成百上千，五花八门，我们不可能为每一张优惠券编写单独的实现类。

其实在天机学堂中，优惠券规则从类型来说就4种：

- 每满减：例如每满100减10
- 折扣：例如满100打9.5折，最大不超过50
- 无门槛：例如直接抵扣10元
- 满减：例如满100减15

优惠类型（discountType）对应到数据库的字段就是discount_type.

另外，优惠券描述中基本都会包含下列字段：

- 优惠门槛（thresholdAmount）：也就是优惠的基本条件，例如满100
- 优惠值（discountValue）：也就是具体折扣，例如减10、打9.5折
- 最大优惠（maxDiscountValue）：最大折扣，限制折扣金额

对应的数据库表结构如下：

discount_type	TINYINT	折扣类型， 1：每满减， 2：折扣， 3：无门槛， 4：满减
discount_value	INT	折扣值，如果是满减则存满减金额，如果是折扣，则存折扣
threshold_amount	INT	使用门槛， 0：表示无门槛，其他值：最低消费金额
max_discount_amount	INT	最高优惠金额，满减最大， 0：表示没有限制，不为0，则

因此各种各样的优惠券其规则都可以用上述4个字段标示，而规则根据优惠类型（discountType）来看就分为4种，不同优惠仅仅是其它3个字段值不同而已。

所以优惠券的规则定义四种不同实现类即可，将来我们可以根据优惠类型不同选择具体的实现类来完成功能。像这种定义使用场景可以利用策略模式来定义规则。

在课前资料中我们提供了写好的优惠规则策略：

资料 > 天机学堂 > 课件 > day12-优惠券使用规则 > 资料 > strategy > discount > ⌵ ↺ 🔍

名称	类型	大小
 Discount.java	Java 源文件	1 KB
 DiscountStrategy.java	Java 源文件	1 KB
 NoThresholdDiscount.java	Java 源文件	1 KB
 PerPriceDiscount.java	Java 源文件	2 KB
 PriceDiscount.java	Java 源文件	1 KB
 RateDiscount.java	Java 源文件	2 KB

各个类功能如下：

2.优惠券智能推荐

第一个就是优惠券券方案推荐功能。在订单确认页面，前端会向交易微服务发起**预下单**请求，以获取id和优惠方案列表，页面请求如图：

可以看出这个请求的地址是 `/ts/orders`，也就是交易服务。交易服务首先需要查询课程信息，生成订单id，然后还需要调用优惠促销服务。而促销服务则需要根据订单中的课程信息查询当前用户的优惠券，并给出推荐的优惠组合方案，供用户在页面选择：

优惠券：

请选择优惠券

叠加4券：【优惠108元】

单券：【每满100减12，上限200】

单券：【满300减30】

单券：【满100减20】

订单总价：¥ 448.00

优惠金额：¥ 0

实付金额：¥ 448.00

去结算

我们要实现的就是优惠券的查询和组合方案推荐的部分。

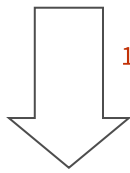
2.1.思路分析

简单来说，这就是一个查询优惠券、计算折扣、筛选最优解的过程。整体流程如下：

▶ Java入门
200元

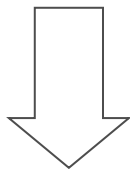
▶ UI设计
99元

▶ 吉他入门
99元



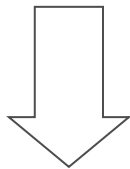
1.查询当前用户
所有可用优惠券

1	25元 满200减25	1
2	9折 满300打9折	2
3	10元 每满100减10	3
4	60元 满500减60	4



2.初步筛选，主要是判断
优惠券使用门槛是否满足

1	25元 满200减25	1
2	9折 满300打9折	2
3	10元 每满100减10	3

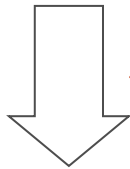


3.对可用优惠券做
全排列组合

1	2	3
1	3	2

2	1	3
2	3	1

3	1	2
3	2	1



4.并发计算每种
组合方案的优惠

worker1

1
↓
2
↓
3

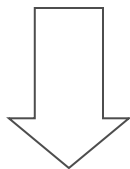
worker2

1
↓
3
↓
2

...

workerN

3
↓
2
↓
1



5.筛选最优解

最终找出优惠券组合方案的最优解，需要满足：

- 用券相同时，优惠金额最高的方案

- 优惠金额相同时，用券最少的方案

OK，接下来我们就逐步来完成这个功能，大概步骤包括：

1. 定义接口
2. 查询用户的优惠券
3. 初步筛选
4. 细筛并完成全排列
5. 计算优惠明细
6. 基于CompletableFuture做并行计算
7. 筛选最优解

2.2.定义接口

首先我们按照Restful风格来定义接口

2.2.1.接口基础信息

接口基础信息如下：

参数	说明			
请求方式	POST			
请求路径	/user-coupons/available			
请求参数	Array	参数名	类型	说明
		id	long	课程id
		cateId	long	三级分类id
		price	int	课程价格
返回值	Array	参数名	类型	说明
		ids	Array	优惠券id集合（优惠券组合方案）
		rules	Array	优惠券规则集合
		discountAmount	integer	该组合的优惠金额




说明：

- **请求方式**：由于请求参数比较复杂，所以请求方式采用了POST，以JSON格式传输
- **请求路径**：资源名称，代表可用优惠券
- **请求参数**：订单中的所有课程信息，因此是一个集合。课程信息主要有：
 - id：主键，区分课程

- price: 价格, 用于计算优惠金额
- catelId: 三级分类id, 用于筛选优惠券
- **返回值:** 多个优惠券的组合方案, 因此是一个集合。每一个方案包含下列属性:
 - ids: 方案中包含的优惠券id集合
 - rules: 方案中的优惠券规则描述集合
 - discountAmount: 该方案最终优惠金额

2.2.2.接口实体

接口中的请求参数和返回值都需要定义对应的实体, 由于接口是微服务之间调用, 所以都采用DTO后缀。在课前资料中已经提供了两个实体:

(D:) > 课程资料 > 天机学堂 > 课件 > day12-优惠券使用规则 > 资料 > dto >			^	
名称	类型	大小		
 CouponDiscountDTO.java	Java 源文件	1 KB		
 OrderCouponDTO.java	Java 源文件	1 KB		
 OrderCourseDTO.java	Java 源文件	1 KB		

说明:

- OrderCourseDTO: 就是请求参数中订单的课程信息
- CouponDiscountDTO: 返回值中的优惠方案信息

2.2.3.接口代码

OK, 接下来就是具体编码了。

由于是用户券的查询和使用, 我们将接口定义到 `UserCouponController` 中:

```
1 package com.tianji.promotion.controller;
2
3 import com.tianji.api.dto.promotion.CouponDiscountDTO;
4 import com.tianji.api.dto.promotion.OrderCourseDTO;
5 // ... 略
6
7 /**
8  * <p>
```

```

9  * 用户领取优惠券的记录，是真正使用的优惠券信息 控制器
10 * </p>
11 *
12 * @author 虎哥
13 */
14 @RestController
15 @RequiredArgsConstructor
16 @RequestMapping("/user-coupons")
17 @Api(tags = "优惠券相关接口")
18 public class UserCouponController {
19
20     private final IUserCouponService userCouponService;
21
22     private final IDiscountService discountService;
23
24     // ... 略
25
26     @ApiOperation("查询我的优惠券可用方案")
27     @PostMapping("/available")
28     public List<CouponDiscountDTO> findDiscountSolution(@RequestBody
    List<OrderCourseDTO> orderCourses){
29         return discountService.findDiscountSolution(orderCourses);
30     }
31 }

```

注意，由于这部分业务主要是对优惠方案计算，并且IUserCouponService已经比较复杂，我们将与优惠券方案计算有关的接口定义到一个新的service，IDiscountService中：

```

1 package com.tianji.promotion.service;
2
3 import com.tianji.api.dto.promotion.CouponDiscountDTO;
4 import com.tianji.api.dto.promotion.OrderCourseDTO;
5
6 import java.util.List;
7
8 public interface IDiscountService {
9     List<CouponDiscountDTO> findDiscountSolution(List<OrderCourseDTO>
    orderCourses);
10 }

```

最后是实现类：

```

1 package com.tianji.promotion.service.impl;
2
3 import com.tianji.api.dto.promotion.CouponDiscountDTO;
4 import com.tianji.api.dto.promotion.OrderCourseDTO;
5 import java.util.*;
6
7 @Slf4j
8 @Service
9 @RequiredArgsConstructor
10 public class DiscountServiceImpl implements IDiscountService {
11
12     @Override
13     public List<CouponDiscountDTO> findDiscountSolution(List<OrderCourseDTO>
        orderCourses) {
14         return null;
15     }
16 }

```

2.3.查询用户券并初步筛选

接下来，我们查询当前用户的优惠券，并基于课程总价做初步筛选。

2.3.1.编写查询SQL语句

我们首先实现对用户券（`UserCoupon`）的查询，查询条件有两个：

- 必须属于当前用户
- 券状态必须是未使用

不过，需要注意的是，查询的结果必须包含 `Coupon` 中的折扣相关信息，因此这条语句是 `coupon` 表和 `user_coupon` 表的联合查询，必须手写 SQL 语句。

查询的结果大部分都是 `Coupon` 中的字段，因此可以用 `Coupon` 对象接收。不过 `UserCoupon` 的 `id` 也需要返回，我们计划用 `Coupon` 的 `creator` 字段来接收，因此查询时需要将 `UserCoupon` 的 `id`起别名为 `creator`

首先在 `UserCouponMapper` 接口中定义方法：

```

1 public interface UserCouponMapper extends BaseMapper<UserCoupon> {
2
3     List<Coupon> queryMyCoupons(@Param("userId") Long userId);
4 }

```

然后在 `resource/mapper/UserCouponMapper.xml` 中定义SQL语句：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
3 <mapper namespace="com.tianji.promotion.mapper.UserCouponMapper">
4
5     <select id="queryMyCoupons"
      resultType="com.tianji.promotion.domain.po.Coupon">
6         SELECT c.id, c.discount_type, c.`specific`, c.discount_value,
          c.threshold_amount,
7             c.max_discount_amount, uc.id AS creator
8         FROM user_coupon uc
9             INNER JOIN coupon c ON uc.coupon_id = c.id
10        WHERE uc.user_id = #{userId} AND uc.status = 1
11    </select>
12 </mapper>
```

2.3.2.实现查询和初筛

我们首先实现对用户券（UserCoupon）的查询，查询条件有两个：

- 必须属于当前用户
- 券状态必须是未使用

```
1 package com.tianji.promotion.service.impl;
2
3 import com.tianji.api.dto.promotion.CouponDiscountDTO;
4 import com.tianji.api.dto.promotion.OrderCourseDTO;
5 import com.tianji.common.utils.CollUtils;
6 import com.tianji.common.utils.UserContext;
7 import com.tianji.promotion.domain.po.Coupon;
8 import com.tianji.promotion.mapper.UserCouponMapper;
9 import com.tianji.promotion.service.IDiscountService;
10 import com.tianji.promotion.strategy.discount.Discount;
11 import com.tianji.promotion.strategy.discount.DiscountStrategy;
12 import lombok.RequiredArgsConstructor;
```

```
13 import lombok.extern.slf4j.Slf4j;
14 import org.springframework.stereotype.Service;
15
16 import java.util.*;
17 import java.util.stream.Collectors;
18
19 @Slf4j
20 @Service
21 @RequiredArgsConstructor
22 public class DiscountServiceImpl implements IDiscountService {
23
24     private final UserCouponMapper userCouponMapper;
25
26     @Override
27     public List<CouponDiscountDTO> findDiscountSolution(List<OrderCourseDTO>
orderCourses) {
28         // 1. 查询我的所有可用优惠券
29         List<Coupon> coupons =
userCouponMapper.queryMyCoupons(UserContext.getUser());
30         if (CollUtils.isEmpty(coupons)) {
31             return CollUtils.emptyList();
32         }
33         // 2. 初筛
34         // 2.1. 计算订单总价
35         int totalAmount =
orderCourses.stream().mapToInt(OrderCourseDTO::getPrice).sum();
36         // 2.2. 筛选可用券
37         List<Coupon> availableCoupons = coupons.stream()
38             .filter(c ->
DiscountStrategy.getDiscount(c.getDiscountType()).canUse(totalAmount, c))
39             .collect(Collectors.toList());
40         if (CollUtils.isEmpty(availableCoupons)) {
41             return CollUtils.emptyList();
42         }
43         // 3. 排列组合出所有方案
44         // 3.1. 细筛 (找出每一个优惠券的可用的课程, 判断课程总价是否达到优惠券的使用需求)
45
46         // 3.2. 排列组合
47
48         // 4. 计算方案的优惠明细
49
50         // 5. 筛选最优解
51         return null;
52     }
53 }
```

2.4.细筛

每个优惠券的都有自己的使用范围（指定的课程分类），而订单中的课程也会有不同的分类，因此每张优惠券可以使用的课程可能不同。

我们之前在初筛时，是基于所有课程计算总价，判断优惠券是否可用，这显然是不合适的。应该找出优惠券限定范围内的课程，然后计算总价，判断是否可用。

因此，细筛步骤有两步：

- 首先要基于优惠券的限定范围对课程筛选，找出可用课程。如果没有可用课程，则优惠券不可用。
- 然后对可用课程计算总价，判断是否达到优惠门槛，没有达到门槛则优惠券不可用

可以发现，细筛需要查询每一张优惠券的限定范围，找出可用课程。这就需要查询 `coupon_scope` 表，还是比较麻烦的。而且，后期计算优惠明细的时候我们还需要知道每张优惠券的可用课程，因此在细筛完成后，建议把每个优惠券及对应的可用课程缓存到一个 `Map` 中，形成映射关系，避免后期重复查找。

所以我们在 `DiscountServiceImpl` 中封装一个方法用以细筛和查询优惠券的使用范围内的课程：

```
1 private final ICouponScopeService scopeService;
2
3 private Map<Coupon, List<OrderCourseDTO>> findAvailableCoupon(
4     List<Coupon> coupons, List<OrderCourseDTO> courses) {
5     Map<Coupon, List<OrderCourseDTO>> map = new HashMap<>(coupons.size());
6     for (Coupon coupon : coupons) {
7         // 1.找出优惠券的可用的课程
8         List<OrderCourseDTO> availableCourses = courses;
9         if (coupon.getSpecific()) {
10             // 1.1.限定了范围，查询券的可用范围
11             List<CouponScope> scopes =
12                 scopeService.lambdaQuery().eq(CouponScope::getCouponId, coupon.getId()).list();
13             // 1.2.获取范围对应的分类id
14             Set<Long> scopeIds =
15                 scopes.stream().map(CouponScope::getBizId).collect(Collectors.toSet());
16             // 1.3.筛选课程
17             availableCourses = courses.stream()
18                 .filter(c ->
19                     scopeIds.contains(c.getCateId()))
20                 .collect(Collectors.toList());
21         }
22         if (CollUtils.isEmpty(availableCourses)) {
23             // 没有任何可用课程，抛弃
24         }
25     }
26     return map;
27 }
```

```

20         continue;
21     }
22     // 2.计算课程总价
23     int totalAmount =
        availableCourses.stream().mapToInt(OrderCourseDTO::getPrice).sum();
24     // 3.判断是否可用
25     Discount discount =
        DiscountStrategy.getDiscount(coupon.getDiscountType());
26     if (discount.canUse(totalAmount, coupon)) {
27         map.put(coupon, availableCourses);
28     }
29 }
30 return map;
31 }

```

2.5.优惠方案全排列组合

在完成优惠券细筛以后，我们就可以拿到所有的优惠券。由于优惠券的使用顺序不同，可能导致最终的优惠金额不同。

我们要找出优惠金额最高的优惠券组合，就必须先找出所有的排列组合，然后分别计算出优惠金额，然后对比并找出最优解。

这里我们采用的思路是这样的：


- 优惠券放在一个List集合中，他们的角标就是0~N的数字
- 找优惠券的全排列组合，就是找N个不重复数字的全排列组合
 - 例如2个数字：[0,1]，排列就包含：[0,1]、[1,0]两种
- 然后按照角标排列优惠券即可

找N个不重复数字的全排列组合可以使用回溯算法，对应的LeetCode练习题：

<https://leetcode.cn/problems/permutations/>

当然，我们课前资料也提供了一个回溯算法的工具类：

D:) > 课程资料 > 天机学堂 > 课件 > day12-优惠券使用规则 > 资料 > util >

名称	类型	大小
 PermuteUtil.java	Java 源文件	2 KB

现在，在 `DiscountServiceImpl` 中添加**细筛**和**全排列**的逻辑：

```
1 package com.tianji.promotion.service.impl;
2
3 import com.tianji.api.dto.promotion.CouponDiscountDTO;
4 import com.tianji.api.dto.promotion.OrderCourseDTO;
5 import com.tianji.common.utils.CollUtils;
6 import com.tianji.common.utils.UserContext;
7 import com.tianji.promotion.domain.po.Coupon;
8 import com.tianji.promotion.domain.po.CouponScope;
9 import com.tianji.promotion.mapper.UserCouponMapper;
10 import com.tianji.promotion.service.ICouponScopeService;
11 import com.tianji.promotion.service.IDiscountService;
12 import com.tianji.promotion.strategy.discount.Discount;
13 import com.tianji.promotion.strategy.discount.DiscountStrategy;
14 import com.tianji.promotion.utils.PermuteUtil;
15 import lombok.RequiredArgsConstructor;
16 import lombok.extern.slf4j.Slf4j;
17 import org.springframework.stereotype.Service;
18
19 import java.util.*;
20 import java.util.concurrent.CompletableFuture;
21 import java.util.concurrent.CountDownLatch;
22 import java.util.concurrent.Executor;
23 import java.util.concurrent.TimeUnit;
24 import java.util.stream.Collectors;
25
26 @Slf4j
27 @Service
28 @RequiredArgsConstructor
29 public class DiscountServiceImpl implements IDiscountService {
30
31     private final UserCouponMapper userCouponMapper;
32     private final ICouponScopeService scopeService;
33
34     @Override
35     public List<CouponDiscountDTO> findDiscountSolution(List<OrderCourseDTO>
        orderCourses) {
```



```

36      // 1.查询我的所有可用优惠券
37      List<Coupon> coupons =
        userCouponMapper.queryMyCoupons(UserContext.getUser());
38      if (CollUtils.isEmpty(coupons)) {
39          return CollUtils.emptyList();
40      }
41      // 2.初筛
42      // 2.1.计算订单总价
43      int totalAmount =
        orderCourses.stream().mapToInt(OrderCourseDTO::getPrice).sum();
44      // 2.2.筛选可用券
45      List<Coupon> availableCoupons = coupons.stream()
46          .filter(c ->
            DiscountStrategy.getDiscount(c.getDiscountType()).canUse(totalAmount, c))
47          .collect(Collectors.toList());
48      if (CollUtils.isEmpty(availableCoupons)) {
49          return CollUtils.emptyList();
50      }
51      // 3.排列组合出所有方案
52      // 3.1.细筛 (找出每一个优惠券的可用的课程, 判断课程总价是否达到优惠券的使用需求)
53      Map<Coupon, List<OrderCourseDTO>> availableCouponMap =
        findAvailableCoupon(availableCoupons, orderCourses);
54      if (CollUtils.isEmpty(availableCouponMap)) {
55          return CollUtils.emptyList();
56      }
57      // 3.2.排列组合
58      availableCoupons = new ArrayList<>(availableCouponMap.keySet());
59      List<List<Coupon>> solutions = PermuteUtil.permute(availableCoupons);
60      // 3.3.添加单券的方案
61      for (Coupon c : availableCoupons) {
62          solutions.add(List.of(c));
63      }
64
65      // 4.计算方案的优惠明细
66
67      // 5.筛选最优解
68      return null;
69  }
70 }

```

需要注意的是，全排列中只包含券组合方案，但是页面渲染的时候需要展示单张券供用户选择。因此我们将单张券也作为组合添加进去。

```
// 3. 排列组合出所有方案
// 3.1. 细筛（找出每一个优惠券的可行的课程，判断课程总价是否达到优惠券的使用需求）
Map<Coupon, List<OrderCourseDTO>> availableCouponMap = findAvailableCoupon(availableCoupons, orderCourses);
if (CollUtils.isEmpty(availableCouponMap)) {
    return CollUtils.emptyList();
}
// 3.2. 排列组合
availableCoupons = new ArrayList<>(availableCouponMap.keySet());
List<List<Coupon>> solutions = PermuteUtil.permute(availableCoupons);
// 3.3. 添加单券的方案
for (Coupon c : availableCoupons) {
    solutions.add(List.of(c));
}
```

2.5.计算优惠明细

2.5.1.单张优惠券算法

单张优惠券的优惠金额计算流程如下：

- 1) 判断优惠券限定范围，找出范围内的课程
- 2) 计算课程总价
- 3) 判断券是否可用
- 4) 计算优惠金额

例如现在有一些商品：

商品		
序号	price	分类
1	100	a
2	100	b
3	100	b

然后有一张优惠券：

优惠券			
序号	满	减	分类
2	200	100	b

我们按照上述算法来判断：

- 1) 判断限定范围：这张券限定分类 b，对应的商品序号是2、3
- 2) 计算课程总价：商品序号2、3的总价为200
- 3) 判断是否可用：总价刚好达到优惠券满减门槛200，可以使用
- 4) 计算优惠：满200减100，因此最终优惠金额就是100元

2.5.2.券叠加算法

券叠加就是按照券组合的顺序，依次计算每张券的优惠金额，最终优惠金额就是所有权的优惠累加。

需要注意的是：由于一张券计算完优惠后，商品的金额会发生变化，因此下一张券的计算金额会随之改变，因此券叠加的顺序非常重要。

而且为了方便计算后续券的优惠金额，我们必须知道商品金额具体的变化，也就是弄清楚每一张优惠券使用后，每个商品的具体优惠金额，我们称之为**优惠明细**，我们可以用一个表格来记录：

商品	
序号	优惠明细
1	
2	
3	

因此，券叠加算法比单券算法需要多一步：

- 1) 判断优惠券限定范围，找出范围内的课程
- 2) 计算课程总价
- 3) 判断券是否可用

- 4) 计算优惠金额
- 5) 计算优惠明细

例如现在有一些商品：

商品		
序号	price	分类
1	100	a
2	100	b
3	100	b

然后有一组优惠券：

优惠券			
序号	满	减	分类
1	100（每）	20	a,b
2	200	100	b
3	80	20	a

最终的计算步骤如下：

[illegible]

3	100	b	3	80	20	a	3	20
---	-----	---	---	----	----	---	---	----

券1的计算步骤如下:

- 1) 判断范围：券1可用于所有分类，因此商品序号1、2、3都可以用
- 2) 计算总价：所有商品累加共300元
- 3) 判断是否可用：券1门槛是100，符合要求
- 4) 计算优惠金额：每满100减20，因此总共折扣就是60元
- 5) 计算优惠明细：

优惠明细的计算算法如下：

- 正常情况下，按照商品价格在商品总价中的比例，乘以优惠总金额
- 最后一个商品，为了避免出现精度损失导致的金额不一致，最后一个商品的优惠明细等于优惠总金额减去其它商品的优惠明细之和

例如，商品1、2的折扣： $(100 / 300) * 60 = 20$ ，商品3的折扣等于： $60 - 20 - 20 = 20$

券2:	商品			优惠券					商品优惠明细	
	序号	价格	分类	序号	满	减	分类	折扣	序号	优惠明细
	1	100	a	1	100(每)	20	a,b	0.8	1	20
	2	100	b	2	200	100	b	0.5	2	20
	3	100	b	3	80	20	a	0.8	3	20

券2的计算步骤如下:

- 1) 判断范围：券2可用于分类b，因此商品序号2、3都可以用
- 2) 计算总价：商品2已经优惠了20，现在价格是80，商品3已经优惠了20，现在价格是80。因此商品总价是160
- 3) 判断是否可用：券2门槛是200，不符合要求，跳过

商品		
序号	价格	分类

优惠券				
序号	满	减	分类	有效期

商品优惠明细	
序号	优惠明细

券3:	1	100	a	1	100(每)	20	a,b	1	40
	2	100	b	2	200	100	b	2	20
	3	100	b	3	80	20	a	3	20

券3的计算步骤如下：

- 1) 判断范围：券3可用于所有分类，因此商品序号1可以用
- 2) 计算总价：商品1原价100元，已经优惠20，现价80元
- 3) 判断是否可用：券3门槛是80，符合要求
- 4) 计算优惠金额：满80减20，因此总共折扣就是20元
- 5) 计算优惠明细：由于只有商品1可用，商品1优惠明细就是20元

2.5.3.编码实现算法

首先是查询优惠方案的主体方法，如下：

```

1 package com.tianji.promotion.service.impl;
2
3 import com.tianji.api.dto.promotion.CouponDiscountDTO;
4 import com.tianji.api.dto.promotion.OrderCourseDTO;
5 import com.tianji.common.utils.CollUtils;
6 import com.tianji.common.utils.UserContext;
7 import com.tianji.promotion.domain.po.Coupon;
8 import com.tianji.promotion.domain.po.CouponScope;
9 import com.tianji.promotion.mapper.UserCouponMapper;
10 import com.tianji.promotion.service.ICouponScopeService;
11 import com.tianji.promotion.service.IDiscountService;
12 import com.tianji.promotion.strategy.discount.Discount;
13 import com.tianji.promotion.strategy.discount.DiscountStrategy;
14 import com.tianji.promotion.utils.PermuteUtil;
15 import lombok.RequiredArgsConstructor;
16 import lombok.extern.slf4j.Slf4j;
17 import org.springframework.stereotype.Service;
18
19 import java.util.*;
20 import java.util.concurrent.CompletableFuture;
21 import java.util.concurrent.CountDownLatch;
22 import java.util.concurrent.Executor;
23 import java.util.concurrent.TimeUnit;
24 import java.util.stream.Collectors;
25

```

```

26 @Slf4j
27 @Service
28 @RequiredArgsConstructor
29 public class DiscountServiceImpl implements IDiscountService {
30
31     private final UserCouponMapper userCouponMapper;
32
33     @Override
34     public List<CouponDiscountDTO> findDiscountSolution(List<OrderCourseDTO>
orderCourses) {
35         // 1.查询我的所有可用优惠券
36         List<Coupon> coupons =
userCouponMapper.queryMyCoupons(UserContext.getUser());
37         if (CollUtils.isEmpty(coupons)) {
38             return CollUtils.emptyList();
39         }
40         // 2.初筛
41         // 2.1.计算订单总价
42         int totalAmount =
orderCourses.stream().mapToInt(OrderCourseDTO::getPrice).sum();
43         // 2.2.筛选可用券
44         List<Coupon> availableCoupons = coupons.stream()
45             .filter(c ->
DiscountStrategy.getDiscount(c.getDiscountType()).canUse(totalAmount, c))
46             .collect(Collectors.toList());
47         if (CollUtils.isEmpty(availableCoupons)) {
48             return CollUtils.emptyList();
49         }
50         // 3.排列组合出所有方案
51         // 3.1.细筛 (找出每一个优惠券的可用的课程, 判断课程总价是否达到优惠券的使用需求)
52         Map<Coupon, List<OrderCourseDTO>> availableCouponMap =
findAvailableCoupon(availableCoupons, orderCourses);
53         if (CollUtils.isEmpty(availableCouponMap)) {
54             return CollUtils.emptyList();
55         }
56         // 3.2.排列组合
57         availableCoupons = new ArrayList<>(availableCouponMap.keySet());
58         List<List<Coupon>> solutions = PermuteUtil.permute(availableCoupons);
59         // 3.3.添加单券的方案
60         for (Coupon c : availableCoupons) {
61             solutions.add(List.of(c));
62         }
63
64         // 4.计算方案的优惠明细
65         List<CouponDiscountDTO> list =
66             Collections.synchronizedList(new ArrayList<>
(solutions.size()));

```

```

67         for (List<Coupon> solution : solutions) {
68             list.add(calculateSolutionDiscount(availableCouponMap,
orderCourses, solution));
69         }
70         // 5. 筛选最优解
71         return null;
72     }
73
74     // ... 略
75 }

```

具体的计算逻辑同样在 `DiscountServiceImpl` 中，具体如下：

```

1 private CouponDiscountDTO calculateSolutionDiscount(
2     Map<Coupon, List<OrderCourseDTO>> couponMap, List<OrderCourseDTO>
courses, List<Coupon> solution) {
3     // 1. 初始化DTO
4     CouponDiscountDTO dto = new CouponDiscountDTO();
5     // 2. 初始化折扣明细的映射
6     Map<Long, Integer> detailMap =
courses.stream().collect(Collectors.toMap(OrderCourseDTO::getId, oc -> 0));
7     // 3. 计算折扣
8     for (Coupon coupon : solution) {
9         // 3.1. 获取优惠券限定范围对应的课程
10        List<OrderCourseDTO> availableCourses = couponMap.get(coupon);
11        // 3.2. 计算课程总价 (课程原价 - 折扣明细)
12        int totalAmount = availableCourses.stream()
13            .mapToInt(oc -> oc.getPrice() -
detailMap.get(oc.getId()))
14            .sum();
15        // 3.3. 判断是否可用
16        Discount discount =
DiscountStrategy.getDiscount(coupon.getDiscountType());
17        if (!discount.canUse(totalAmount, coupon)) {
18            // 券不可用，跳过
19            continue;
20        }
21        // 3.4. 计算优惠金额
22        int discountAmount = discount.calculateDiscount(totalAmount, coupon);
23        // 3.5. 计算优惠明细
24        calculateDiscountDetails(detailMap, availableCourses, totalAmount,
discountAmount);
25        // 3.6. 更新DTO数据
26        dto.getIds().add(coupon.getCreator());
27        dto.getRules().add(discount.getRule(coupon));

```



```

27         dto.setDiscountAmount(discountAmount + dto.getDiscountAmount());
28     }
29     return dto;
30 }
31
32 private void calculateDiscountDetails(Map<Long, Integer> detailMap,
    List<OrderCourseDTO> courses,
33                                     int totalAmount, int discountAmount) {
34     int times = 0;
35     int remainDiscount = discountAmount;
36     for (OrderCourseDTO course : courses) {
37         // 更新课程已计算数量
38         times++;
39         int discount = 0;
40         // 判断是否是最后一个课程
41         if (times == courses.size()) {
42             // 是最后一个课程，总折扣金额 - 之前所有商品的折扣金额之和
43             discount = remainDiscount;
44         } else {
45             // 计算折扣明细（课程价格在总价中占的比例，乘以总的折扣）
46             discount = discountAmount * course.getPrice() / totalAmount;
47             remainDiscount -= discount;
48         }
49         // 更新折扣明细
50         detailMap.put(course.getId(), discount +
            detailMap.get(course.getId()));
51     }
52 }

```

2.6.CompleteableFuture并发计算

可以发现，上节的优惠券算法还是比较复杂的。而且由于优惠方案很多，目前我们此案有的是for循环逐个方案串行计算，整体性能可想而知。

所以，为了提高计算效率，我们可以利用多线程并行计算。具体步骤如下：

- 定义一个线程池
- for循环将每个方案交给一个线程去任务执行
- 等待所有任务计算完毕，返回结果

这里的难点有两个：

- 1) 线程任务是带返回值的任务
- 2) 虽然是多线程运行，但是我们要等待所有线程都执行完毕后才返回结果

针对第二个点，我们可以利用 JUC 包提供的工具 `CountDownLatch` 来实现。

针对第一个点，我们则需要利用一个JDK1.8的新工具： `CompletableFuture` 来实现。

`CompletableFuture`的用法详解：



知乎 <https://zhuanlan.zhihu.com/p/344431341>

CompletableFuture用法详解

@ TOC前言JAVA支持的多线程开启方式根据Oracle官方出具的Java文档说明，创建线程的方式只有两种：继承Thread或者实现Runnable接口。但是这两种方法都存在...

我们首先在 `com.tianji.promotion.config.PromotionConfig`中 自定义一个线程池：



具体代码：

```
1 package com.tianji.promotion.config;
2
3 import lombok.extern.slf4j.Slf4j;
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;
7
```

```

8 import java.util.concurrent.Executor;
9 import java.util.concurrent.ThreadPoolExecutor;
10
11 @Slf4j
12 @Configuration
13 public class PromotionConfig {
14
15     // ... 略
16
17     @Bean
18     public Executor discountSolutionExecutor(){
19         ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
20         // 1.核心线程池大小
21         executor.setCorePoolSize(12);
22         // 2.最大线程池大小
23         executor.setMaxPoolSize(12);
24         // 3.队列大小
25         executor.setQueueCapacity(99999);
26         // 4.线程名称
27         executor.setThreadNamePrefix("discount-solution-calculator-");
28         // 5.拒绝策略
29         executor.setRejectedExecutionHandler(new
30             ThreadPoolExecutor.AbortPolicy());
31         executor.initialize();
32         return executor;
33     }
34 }

```

然后就是修改DiscountServiceImpl中查询优惠方案的函数主体：

```

1 package com.tianji.promotion.service.impl;
2
3 import com.tianji.api.dto.promotion.CouponDiscountDTO;
4 import com.tianji.api.dto.promotion.OrderCourseDTO;
5 import com.tianji.common.utils.CollUtils;
6 import com.tianji.common.utils.UserContext;
7 import com.tianji.promotion.domain.po.Coupon;
8 import com.tianji.promotion.domain.po.CouponScope;
9 import com.tianji.promotion.mapper.UserCouponMapper;
10 import com.tianji.promotion.service.ICouponScopeService;
11 import com.tianji.promotion.service.IDiscountService;
12 import com.tianji.promotion.strategy.discount.Discount;
13 import com.tianji.promotion.strategy.discount.DiscountStrategy;
14 import com.tianji.promotion.utils.PermuteUtil;

```

```

15 import lombok.RequiredArgsConstructor;
16 import lombok.extern.slf4j.Slf4j;
17 import org.springframework.stereotype.Service;
18
19 import java.util.*;
20 import java.util.concurrent.CompletableFuture;
21 import java.util.concurrent.CountDownLatch;
22 import java.util.concurrent.Executor;
23 import java.util.concurrent.TimeUnit;
24 import java.util.stream.Collectors;
25
26 @Slf4j
27 @Service
28 @RequiredArgsConstructor
29 public class DiscountServiceImpl implements IDiscountService {
30
31     private final UserCouponMapper userCouponMapper;
32     private final ICouponScopeService scopeService;
33     private final Executor discountSolutionExecutor;
34
35     @Override
36     public List<CouponDiscountDTO> findDiscountSolution(List<OrderCourseDTO>
orderCourses) {
37         // 1. 查询我的所有可用优惠券
38         List<Coupon> coupons =
userCouponMapper.queryMyCoupons(UserContext.getUser());
39         if (CollUtils.isEmpty(coupons)) {
40             return CollUtils.emptyList();
41         }
42         // 2. 初筛
43         // 2.1. 计算订单总价
44         int totalAmount =
orderCourses.stream().mapToInt(OrderCourseDTO::getPrice).sum();
45         // 2.2. 筛选可用券
46         List<Coupon> availableCoupons = coupons.stream()
47             .filter(c ->
DiscountStrategy.getDiscount(c.getDiscountType()).canUse(totalAmount, c))
48             .collect(Collectors.toList());
49         if (CollUtils.isEmpty(availableCoupons)) {
50             return CollUtils.emptyList();
51         }
52         // 3. 排列组合出所有方案
53         // 3.1. 细筛 (找出每一个优惠券的可用的课程, 判断课程总价是否达到优惠券的使用需求)
54         Map<Coupon, List<OrderCourseDTO>> availableCouponMap =
findAvailableCoupon(availableCoupons, orderCourses);
55         if (CollUtils.isEmpty(availableCouponMap)) {
56             return CollUtils.emptyList();

```

```

57     }
58     // 3.2.排列组合
59     availableCoupons = new ArrayList<>(availableCouponMap.keySet());
60     List<List<Coupon>> solutions = PermuteUtil.permute(availableCoupons);
61     // 3.3.添加单券的方案
62     for (Coupon c : availableCoupons) {
63         solutions.add(List.of(c));
64     }
65
66     // 4.计算方案的优惠明细
67     List<CouponDiscountDTO> list = Collections.synchronizedList(new
ArrayList<>(solutions.size()));
68     // 4.1.定义闭锁
69     CountDownLatch latch = new CountDownLatch(solutions.size());
70     for (List<Coupon> solution : solutions) {
71         // 4.2.异步计算
72         CompletableFuture
73             .supplyAsync(
74                 () ->
calculateSolutionDiscount(availableCouponMap, orderCourses, solution),
75                 discountSolutionExecutor
76             ).thenAccept(dto -> {
77                 // 4.3.提交任务结果
78                 list.add(dto);
79                 latch.countDown();
80             });
81     }
82     // 4.4.等待运算结束
83     try {
84         latch.await(1, TimeUnit.SECONDS);
85     } catch (InterruptedException e) {
86         log.error("优惠方案计算被中断, {}", e.getMessage());
87     }
88
89     // TODO 5.筛选最优解
90     return null;
91 }
92
93 // ... 略
94 }

```

2.7.筛选最优解

现在，我们计算出了成吨的优惠方案及其优惠金额，但是该如何从其中筛选出最优方案呢？最优方案的标准又是什么呢？

首先来看最优标准：

- 用券相同时，优惠金额最高的方案
- 优惠金额相同时，用券最少的方案

这里的 **用券相同** 只关心用了哪些券，不关心顺序。例如【1,2,3】、【2,1,3】、【3,1,2】都用了1、2、3这三张券，属于用券相同，我们要找出其中优惠金额最高的方案。再比如：【1,2,3】和【2,3】、【3,1】就是用券不同，就需要分别去找优惠金额更高的。

那么该如何寻找最优解呢？

其实寻找最优解的流程跟找数组中最小值类似：

- 定义一个变量记录最小值
- 逐个遍历数组，判断当前元素是否比最小值更小
- 如果是，则覆盖最小值；如果否，则放弃
- 循环结束，变量中记录的就是最小值

我们寻找最优解可以参考上述过程，不过略有差异，核心原因是：**券组合有多种，因此最优解不止一个**。因此我们不能用一个变量类记录最优解，而是用Map来记录，结构如下：

券id组合	优惠方案

优惠金额	优惠方案

其中：

- 第一个Map用来记录用券相同时，优惠金额最高的方案；
- 第二个Map用来记录优惠金额相同时，用券最少的方案。

最终，两个Map的values的交集就是我们要找的最优解。

具体实现逻辑如下：

```
1
2 package com.tianji.promotion.service.impl;
3
4 import com.tianji.api.dto.promotion.CouponDiscountDTO;
5 import com.tianji.api.dto.promotion.OrderCourseDTO;
6 import com.tianji.common.utils.CollUtils;
7 import com.tianji.common.utils.UserContext;
8 import com.tianji.promotion.domain.po.Coupon;
9 import com.tianji.promotion.domain.po.CouponScope;
10 import com.tianji.promotion.mapper.UserCouponMapper;
11 import com.tianji.promotion.service.ICouponScopeService;
12 import com.tianji.promotion.service.IDiscountService;
13 import com.tianji.promotion.strategy.discount.Discount;
14 import com.tianji.promotion.strategy.discount.DiscountStrategy;
15 import com.tianji.promotion.utils.PermuteUtil;
16 import lombok.RequiredArgsConstructor;
17 import lombok.extern.slf4j.Slf4j;
18 import org.springframework.stereotype.Service;
19
20 import java.util.*;
21 import java.util.concurrent.CompletableFuture;
22 import java.util.concurrent.CountDownLatch;
23 import java.util.concurrent.Executor;
24 import java.util.concurrent.TimeUnit;
25 import java.util.stream.Collectors;
26
27 @Slf4j
28 @Service
29 @RequiredArgsConstructor
30 public class DiscountServiceImpl implements IDiscountService {
31
32     private final UserCouponMapper userCouponMapper;
33     private final ICouponScopeService scopeService;
34     private final Executor discountSolutionExecutor;
35
36     @Override
37     public List<CouponDiscountDTO> findDiscountSolution(List<OrderCourseDTO>
orderCourses) {
38         // 1. 查询我的所有可用优惠券
39         List<Coupon> coupons =
userCouponMapper.queryMyCoupons(UserContext.getUser());
```

```

40         if (CollUtils.isEmpty(coupons)) {
41             return CollUtils.emptyList();
42         }
43         // 2.初筛
44         // 2.1.计算订单总价
45         int totalAmount =
orderCourses.stream().mapToInt(OrderCourseDTO::getPrice).sum();
46         // 2.2.筛选可用券
47         List<Coupon> availableCoupons = coupons.stream()
48             .filter(c ->
DiscountStrategy.getDiscount(c.getDiscountType()).canUse(totalAmount, c))
49             .collect(Collectors.toList());
50         if (CollUtils.isEmpty(availableCoupons)) {
51             return CollUtils.emptyList();
52         }
53         // 3.排列组合出所有方案
54         // 3.1.细筛 (找出每一个优惠券的可用的课程, 判断课程总价是否达到优惠券的使用需求)
55         Map<Coupon, List<OrderCourseDTO>> availableCouponMap =
findAvailableCoupon(availableCoupons, orderCourses);
56         if (CollUtils.isEmpty(availableCouponMap)) {
57             return CollUtils.emptyList();
58         }
59         // 3.2.排列组合
60         availableCoupons = new ArrayList<>(availableCouponMap.keySet());
61         List<List<Coupon>> solutions = PermuteUtil.permute(availableCoupons);
62         // 3.3.添加单券的方案
63         for (Coupon c : availableCoupons) {
64             solutions.add(List.of(c));
65         }
66
67         // 4.计算方案的优惠明细
68         List<CouponDiscountDTO> list = Collections.synchronizedList(new
ArrayList<>(solutions.size()));
69         // 4.1.定义闭锁
70         CountDownLatch latch = new CountDownLatch(solutions.size());
71         for (List<Coupon> solution : solutions) {
72             // 4.2.异步计算
73             CompletableFuture
74                 .supplyAsync(
75                     () ->
calculateSolutionDiscount(availableCouponMap, orderCourses, solution),
discountSolutionExecutor
76                 ).thenAccept(dto -> {
77                     // 4.3.提交任务结果
78                     list.add(dto);
79                     latch.countDown();
80                 });
81         }

```



```

82     }
83     // 4.4. 等待运算结束
84     try {
85         latch.await(1, TimeUnit.SECONDS);
86     } catch (InterruptedException e) {
87         log.error("优惠方案计算被中断, {}", e.getMessage());
88     }
89
90     // 5. 筛选最优解
91     return findBestSolution(list);
92 }
93
94 private List<CouponDiscountDTO> findBestSolution(List<CouponDiscountDTO>
list) {
95     // 1. 准备Map记录最优解
96     Map<String, CouponDiscountDTO> moreDiscountMap = new HashMap<>();
97     Map<Integer, CouponDiscountDTO> lessCouponMap = new HashMap<>();
98     // 2. 遍历, 筛选最优解
99     for (CouponDiscountDTO solution : list) {
100         // 2.1. 计算当前方案的id组合
101         String ids = solution.getIds().stream()
102             .sorted(Long::compare).map(String::valueOf).collect(Collectors.joining(","));
103         // 2.2. 比较用券相同时, 优惠金额是否最大
104         CouponDiscountDTO best = moreDiscountMap.get(ids);
105         if (best != null && best.getDiscountAmount() >=
solution.getDiscountAmount()) {
106             // 当前方案优惠金额少, 跳过
107             continue;
108         }
109         // 2.3. 比较金额相同时, 用券数量是否最少
110         best = lessCouponMap.get(solution.getDiscountAmount());
111         int size = solution.getIds().size();
112         if (size > 1 && best != null && best.getIds().size() <= size) {
113             // 当前方案用券更多, 放弃
114             continue;
115         }
116         // 2.4. 更新最优解
117         moreDiscountMap.put(ids, solution);
118         lessCouponMap.put(solution.getDiscountAmount(), solution);
119     }
120     // 3. 求交集
121     Collection<CouponDiscountDTO> bestSolutions = CollUtils
122         .intersection(moreDiscountMap.values(),
lessCouponMap.values());
123     // 4. 排序, 按优惠金额降序
124     return bestSolutions.stream()

```

```

125     .sorted(Comparator.comparingInt(CouponDiscountDTO::getDiscountAmount).reversed(
    ))
126     .collect(Collectors.toList());
127 }
128
129 // ... 略
130
131 }

```

3.练习

用户确认订单信息后，会提交订单，页面如下：

购买课程

课程名称	单价(元)
 木老师的吉他入门	¥ 100.00
 Java程序员的前端课程	¥ 149.00
 可能是史上最全的微服务技术栈课程	¥ 199.00

优惠券：
 叠加4券：【优惠108元】

订单总价：¥ 448.00
 优惠金额：¥ 108.00
 实付金额：¥ 340.00

去结算

请求会提交到交易服务，交易服务需要做以下事情：

- 查询商品信息
- 查询优惠明细
- 保存订单信息
- 核销优惠券

因此，优惠促销服务需要提供两个接口：

- 一个是根据订单和优惠方案计算优惠明细
- 一个是核销优惠券

3.1.根据券方案计算订单优惠明细

用户去下单了，一定是选择了一种优惠方案，也就是一张或几张券的组合。而在下单时需要将优惠明细记录到订单详情中。

因此必须向优惠促销服务查询优惠明细信息，而要计算优惠明细，必须知道订单中的课程以及选择的优惠方案。所以，查询时一定要传递订单中的课程信息、用户选择的优惠方案信息。

3.1.1.接口信息

接口信息如下：

参数	说明				
请求方式	POST				
请求路径	/user-coupons/discount				
请求参数	参数名	类型	说明		
	userCouponIds	array	用户优惠券组合，也就是优惠方案		
	courseList	array	参数名	类型	说明
			id	long	课程id
			cateId	long	三级分类id
			price	int	课程价格
返回值	参数名	类型	说明		
	ids	Array	优惠券id集合（优惠券组合）		
	discountAmount	integer	该组合的优惠金额		
	discountDetail	Map	{ courseId: discountAmount, ... }		

解读：




- **请求方式**：POST，虽说是查询优惠明细，但是由于参数较复杂，为了传参方便，这里使用POST
- **请求路径**：/user-coupons/discount，就是折扣信息的意思
- **请求参数**：订单及订单中的优惠方案
 - userCouponIds：用户使用的优惠券id集合，也就是优惠方案
 - courseList：订单中包含的课程，用于计算优惠金额和优惠明细
- **返回值**：优惠金额以及明细

- discountAmount: 计算出的优惠金额
- discountDetail: 优惠明细, 也就是精确到每个课程优惠了多少钱

3.1.2. 实体

请求参数实体新定义, 在课前资料中已经提供了:

(D:) > 课程资料 > 天机学堂 > 课件 > day12-优惠券使用规则 > 资料 > dto >

名称	类型	大小
 CouponDiscountDTO.java	Java 源文件	1 KB
 OrderCouponDTO.java	Java 源文件	1 KB
 OrderCourseDTO.java	Java 源文件	1 KB

返回值实体可以沿用查询优惠方案时用到的 `CouponDiscountDTO`, 只不过需要添加一个新的字段:

```
@Data
@ApiModel(description = "订单的可用优惠券及折扣信息")
@NoArgsConstructor
@AllArgsConstructor
@Accessors(chain = true)
public class CouponDiscountDTO {
    @ApiModelProperty("用户优惠券id集合")
    private List<Long> ids = new ArrayList<>();
    @ApiModelProperty("优惠券规则")
    private List<String> rules = new ArrayList<>();
    @ApiModelProperty("本订单最大优惠金额")
    private Integer discountAmount = 0;
    @ApiModelProperty("优惠明细,key是课程id, value是课程优惠金额")
    private Map<Long, Integer> discountDetail;
}
```

3.1.3. 实现接口

在 `UserCouponController` 中定义接口：

```
1 @ApiOperation("根据券方案计算订单优惠明细")
2 @PostMapping("/discount")
3 public CouponDiscountDTO queryDiscountDetailByOrder(
4     @RequestBody OrderCouponDTO orderCouponDTO){
5     return discountService.queryDiscountDetailByOrder(orderCouponDTO);
6 }
```

然后是在 `IDiscountService` 中定义 service 接口：

```
1 public interface IDiscountService {
2
3     List<CouponDiscountDTO> findDiscountSolution(List<OrderCourseDTO>
4         orderCourses);
5     CouponDiscountDTO queryDiscountDetailByOrder(OrderCouponDTO
6         orderCouponDTO);
7 }
```

最后是 `DiscountServiceImpl` 中定义实现：

```
1 @Override
2 public CouponDiscountDTO queryDiscountDetailByOrder(OrderCouponDTO
3     orderCouponDTO) {
4     // 1. 查询用户优惠券
5     List<Long> userCouponIds = orderCouponDTO.getUserCouponIds();
6     List<Coupon> coupons =
7         userCouponMapper.queryCouponByUserCouponIds(userCouponIds,
8             UserCouponStatus.UNUSED);
9     if (CollUtils.isEmpty(coupons)) {
10         return null;
11     }
12     // 2. 查询优惠券对应课程
13     Map<Coupon, List<OrderCourseDTO>> availableCouponMap =
14         findAvailableCoupon(coupons, orderCouponDTO.getCourseList());
15     if (CollUtils.isEmpty(availableCouponMap)) {
16         return null;
17     }
18 }
```

```

13     }
14     // 3. 查询优惠券规则
15     return calculateSolutionDiscount(availableCouponMap,
        orderCouponDTO.getCourseList(), coupons);
16 }

```

需要注意的有两点：

- 返回的DTO中要添加优惠明细
- 查询用户券的时候是根据id批量查询，需要定义新的SQL

首先是优惠明细处理，我们要修改 `calculateSolutionDiscount` 方法的逻辑：

```

private CouponDiscountDTO calculateSolutionDiscount(
    Map<Coupon, List<OrderCourseDTO>> couponMap, List<OrderCourseDTO> courses, List<Coupon> solution) {
    // 1. 初始化DTO
    CouponDiscountDTO dto = new CouponDiscountDTO();
    // 2. 初始化折扣明细的映射
    Map<Long, Integer> detailMap = courses.stream().collect(Collectors.toMap(OrderCourseDTO::getId, oc -> 0));
    dto.setDiscountDetail(detailMap);
    // 3. 计算折扣
    for (Coupon coupon : solution) {
        // 3.1. 获取优惠券限定范围对应的课程
        List<OrderCourseDTO> availableCourses = couponMap.get(coupon);
        // 3.2. 计算课程总价(课程原价 - 折扣明细)
        int totalAmount = availableCourses.stream()
            .mapToInt(oc -> oc.getPrice() - detailMap.get(oc.getId()))
            .sum();
    }
}

```

然后是，根据用户券id集合查询优惠券信息，需要多表联合查询。因此我们要定义一个mapper方法和对应的SQL：

```

1 public interface UserCouponMapper extends BaseMapper<UserCoupon> {
2
3     List<Coupon> queryMyCoupons(@Param("userId") Long userId);
4
5     List<Coupon> queryCouponByUserCouponIds(
6         @Param("userCouponIds") List<Long> userCouponIds,
7         @Param("status") UserCouponStatus status);
8 }

```

最后是resources/mapper/UserCouponMapper.xml中定义对应的SQL语句：

```

1 <select id="queryCouponByUserCouponIds"
  resultType="com.tianji.promotion.domain.po.Coupon">
2     SELECT c.id, c.discount_type, c.`specific`, c.discount_value,
      c.threshold_amount,
3         c.max_discount_amount, uc.id AS creator
4     FROM user_coupon uc
5     INNER JOIN coupon c on uc.coupon_id = c.id
6     WHERE uc.id IN
7     <foreach collection="userCouponIds" separator="," item="id" open="("
      close=")">
8         #{id}
9     </foreach>
10    AND uc.status = #{status}
11 </select>

```

3.1.4.定义FeignClient方法

在 `tj-api` 模块的 `com.tianji.api.client.promotion.PromotionClient` 中添加新的接口：

```

1 @ApiOperation("根据券方案计算订单优惠明细")
2 @PostMapping("/user-coupons/discount")
3 CouponDiscountDTO queryDiscountDetailByOrder(@RequestBody OrderCouponDTO
  orderCouponDTO);

```

在 `tj-api` 模块的 `com.tianji.api.client.promotion.fallback.PromotionClientFallback` 中添加接口的降级逻辑：

```

1 package com.tianji.api.client.promotion.fallback;
2
3 import com.tianji.api.client.promotion.PromotionClient;
4 import com.tianji.api.dto.promotion.CouponDiscountDTO;
5 import com.tianji.api.dto.promotion.OrderCouponDTO;
6 import com.tianji.api.dto.promotion.OrderCourseDTO;
7 import com.tianji.common.exceptions.BizIllegalException;
8 import lombok.extern.slf4j.Slf4j;
9 import org.springframework.cloud.openfeign.FallbackFactory;
10
11 import java.util.Collections;
12 import java.util.List;

```

```

13
14 @Slf4j
15 public class PromotionClientFallback implements
    FallbackFactory<PromotionClient> {
16     @Override
17     public PromotionClient create(Throwable cause) {
18         log.error("查询促销服务出现异常, ", cause);
19         return new PromotionClient() {
20             @Override
21             public List<CouponDiscountDTO>
findDiscountSolution(List<OrderCourseDTO> orderCourses) {
22                 return Collections.emptyList();
23             }
24
25             @Override
26             public CouponDiscountDTO queryDiscountDetailByOrder(OrderCouponDTO
orderCouponDTO) {
27                 return null;
28             }
29         };
30     }
31 }

```

3.1.5.改造交易服务接口

找到 `tj-trade` 服务的 `com.tianji.trade.service.impl.OrderServiceImpl` 类中的 `placeOrder` 方法，修改其中的代码：

```

1 @Override
2 @Transactional
3 public PlaceOrderResultVO placeOrder(PlaceOrderDTO placeOrderDTO) {
4     Long userId = UserContext.getUser();
5     // 1.查询课程费用信息，如果不可购买，这里直接报错
6     List<CourseSimpleInfoDTO> courseInfos =
getOnShelfCourse(placeOrderDTO.getCourseIds());
7     // 2.封装订单信息
8     Order order = new Order();
9     // 2.1.计算订单金额
10    Integer totalAmount = courseInfos.stream()
11        .map(CourseSimpleInfoDTO::getPrice).reduce(Integer::sum).orElse(0);
12    // 2.2.计算优惠金额
13    order.setDiscountAmount(0);
14    List<Long> couponIds = placeOrderDTO.getCouponIds();

```



```

15     CouponDiscountDTO discount = null;
16     if (CollUtils.isEmpty(couponIds)) {
17         List<OrderCourseDTO> orderCourses = courseInfos.stream()
18             .map(c -> new
OrderCourseDTO().setId(c.getId()).setCateId(c.getThirdCateId()).setPrice(c.getP
rice()))
19             .collect(Collectors.toList());
20         discount = promotionClient.queryDiscountDetailByOrder(new
OrderCouponDTO(couponIds, orderCourses));
21         if(discount != null) {
22             order.setDiscountAmount(discount.getDiscountAmount());
23             order.setCouponIds(discount.getIds());
24         }
25     }
26     Integer realAmount = totalAmount - order.getDiscountAmount();
27     // 2.3.封装其它信息
28     order.setUserId(userId);
29     order.setTotalAmount(totalAmount);
30     order.setRealAmount(realAmount);
31     order.setStatus(OrderStatus.NO_PAY.getValue());
32     order.setMessage(OrderStatus.NO_PAY.getProgressName());
33     // 2.4.订单id
34     Long orderId = placeOrderDTO.getOrderid();
35     order.setId(orderId);
36
37     // 3.封装订单详情
38     List<OrderDetail> orderDetails = new ArrayList<>(courseInfos.size());
39     for (CourseSimpleInfoDTO courseInfo : courseInfos) {
40         Integer discountValue = discount == null ?
41             0 :
discount.getDiscountDetail().getOrDefault(courseInfo.getId(), 0);
42         orderDetails.add(packageOrderDetail(courseInfo, order, discountValue));
43     }
44
45     // 4.写入数据库
46     saveOrderAndDetails(order, orderDetails);
47
48     // 5.删除购物车数据
49     cartService.deleteCartByUserAndCourseIds(userId,
placeOrderDTO.getCourseIds());
50
51     // 6.构建下单结果
52     return PlaceOrderResultVO.builder()
53         .orderId(orderId)
54         .payAmount(realAmount)
55         .status(order.getStatus())

```

```
56     .payOutTime(LocalDateTime.now().plusMinutes(tradeProperties.getPayOrderTTLMinutes()))
57         .build();
58 }
59
60 @Override
61 @Transactional
62 public PlaceOrderResultVO enrolledFreeCourse(Long courseId) {
63     Long userId = UserContext.getUser();
64     // 1.查询课程信息
65     List<Long> cIds = CollUtils.singletonList(courseId);
66     List<CourseSimpleInfoDTO> courseInfos = getOnShelfCourse(cIds);
67     if (CollUtils.isEmpty(courseInfos)) {
68         // 课程不存在
69         throw new BizIllegalException(TradeErrorInfo.COURSE_NOT_EXISTS);
70     }
71     CourseSimpleInfoDTO courseInfo = courseInfos.get(0);
72     if(!courseInfo.getFree()){
73         // 非免费课程，直接报错
74         throw new BizIllegalException(TradeErrorInfo.COURSE_NOT_FREE);
75     }
76     // 2.创建订单
77     Order order = new Order();
78     // 2.1.基本信息
79     order.setUserId(userId);
80     order.setTotalAmount(0);
81     order.setDiscountAmount(0);
82     order.setRealAmount(0);
83     order.setStatus(OrderStatus.ENROLLED.getValue());
84     order.setFinishTime(LocalDateTime.now());
85     order.setMessage(OrderStatus.ENROLLED.getProgressName());
86     // 2.2.订单id
87     Long orderId = IdWorker.getId(order);
88     order.setId(orderId);
89
90     // 3.订单详情
91     OrderDetail detail = packageOrderDetail(courseInfo, order, 0);
92
93     // 4.写入数据库
94     saveOrderAndDetails(order, CollUtils.singletonList(detail));
95
96     // 5.发送MQ消息，通知报名成功
97     rabbitMqHelper.send(
98         MqConstants.Exchange.ORDER_EXCHANGE,
99         MqConstants.Key.ORDER_PAY_KEY,
100         OrderBasicDTO.builder()
```

```

101         .orderId(orderId)
102         .userId(userId)
103         .courseIds(cIds)
104         .finishTime(order.getFinishTime())
105         .build()
106     );
107     // 6.返回vo
108     return PlaceOrderResultVO.builder()
109         .orderId(orderId)
110         .payAmount(0)
111         .status(order.getStatus())
112         .build();
113 }
114
115
116 private OrderDetail packageOrderDetail(CourseSimpleInfoDTO courseInfo, Order
order, Integer discountValue) {
117     OrderDetail detail = new OrderDetail();
118     detail.setUserId(order.getUserId());
119     detail.setOrderId(order.getId());
120     detail.setStatus(order.getStatus());
121     detail.setCourseId(courseInfo.getId());
122     detail.setPrice(courseInfo.getPrice());
123     detail.setCoverUrl(courseInfo.getCoverUrl());
124     detail.setName(courseInfo.getName());
125     detail.setValidDuration(courseInfo.getValidDuration());
126     detail.setDiscountAmount(discountValue);
127     detail.setRealPayAmount(courseInfo.getPrice() -
detail.getDiscountAmount());
128     return detail;
129 }

```

3.2.核销优惠券

在下单完成后需要去核销优惠券。

3.2.1.接口信息

参数	说明		
请求方式	PUT		
请求路径	/user-coupons/use		
请求参数	参数名	类型	说明
	couponIds	Array	用户券id集
返回值	无		
描述	注意： <ul style="list-style-type: none">• 校验用户券状态是否已经使用或过期• 更新优惠券已经使用数量		

解读：

- **请求方式：**PUT，核销优惠就是更新优惠券状态
- **请求路径：**/user-coupons/use，代表优惠券使用
- **请求参数：**待核销的优惠券的id结合
- **返回值：**无

3.2.2.实现接口

在 `tj-promotion` 的 `com.tianji.promotion.controller.UserCouponController` 中定义接口：

```
1 @ApiOperation("核销指定优惠券")
2 @PutMapping("/use")
3 public void writeOffCoupon(@ApiParam("用户优惠券id集合")
4   @RequestParam("couponIds") List<Long> userCouponIds){
5   userCouponService.writeOffCoupon(userCouponIds);
6 }
```

然后是 `tj-promotion` 的 `com.tianji.promotion.service.IUserCouponService` 中的方法声明：

```
1 void writeOffCoupon(List<Long> userCouponIds);
```

最后是 `tj-promotion` 的

`com.tianji.promotion.service.impl.UserCouponServiceImpl` 中的方法实现：

```
1 @Override
2 @Transactional
3 public void writeOffCoupon(List<Long> userCouponIds) {
4     // 1.查询优惠券
5     List<UserCoupon> userCoupons = listByIds(userCouponIds);
6     if (CollUtils.isEmpty(userCoupons)) {
7         return;
8     }
9     // 2.处理数据
10    List<UserCoupon> list = userCoupons.stream()
11        // 过滤无效券
12        .filter(coupon -> {
13            if (coupon == null) {
14                return false;
15            }
16            if (UserCouponStatus.UNUSED != coupon.getStatus()) {
17                return false;
18            }
19            LocalDateTime now = LocalDateTime.now();
20            return !now.isBefore(coupon.getTermBeginTime()) &&
21                !now.isAfter(coupon.getTermEndTime());
22        })
23        // 组织新增数据
24        .map(coupon -> {
25            UserCoupon c = new UserCoupon();
26            c.setId(coupon.getId());
27            c.setStatus(UserCouponStatus.USED);
28            return c;
29        })
30        .collect(Collectors.toList());
31    // 4.核销, 修改优惠券状态
32    boolean success = updateBatchById(list);
33    if (!success) {
34        return;
35    }
36    // 5.更新已使用数量
37    List<Long> couponIds =
38        userCoupons.stream().map(UserCoupon::getCouponId).collect(Collectors.toList());
39    int c = couponMapper.incrUsedNum(couponIds, 1);
40    if (c < 1) {
41        throw new DbException("更新优惠券使用数量失败!");
42    }
```

3.2.3.定义FeignClient方法

在 `tj-api` 模块的 `com.tianji.api.client.promotion.PromotionClient` 中添加新的接口：

```
1 @ApiOperation("核销指定优惠券")
2 @PutMapping("/user-coupons/use")
3 void writeOffCoupon(@ApiParam("用户优惠券id集合") @RequestParam("couponIds")
   List<Long> userCouponIds);
```

在 `tj-api` 模块的

`com.tianji.api.client.promotion.fallback.PromotionClientFallback` 中添加接口的降级逻辑：

```
1 package com.tianji.api.client.promotion.fallback;
2
3 import com.tianji.api.client.promotion.PromotionClient;
4 import com.tianji.api.dto.promotion.CouponDiscountDTO;
5 import com.tianji.api.dto.promotion.OrderCouponDTO;
6 import com.tianji.api.dto.promotion.OrderCourseDTO;
7 import com.tianji.common.exceptions.BizIllegalException;
8 import lombok.extern.slf4j.Slf4j;
9 import org.springframework.cloud.openfeign.FallbackFactory;
10
11 import java.util.Collections;
12 import java.util.List;
13
14 @Slf4j
15 public class PromotionClientFallback implements
   FallbackFactory<PromotionClient> {
16     @Override
17     public PromotionClient create(Throwable cause) {
18         log.error("查询促销服务出现异常, ", cause);
19         return new PromotionClient() {
20             @Override
21             public List<CouponDiscountDTO>
   findDiscountSolution(List<OrderCourseDTO> orderCourses) {
22                 return Collections.emptyList();
23             }
24         }
25     }
26 }
```

```

24
25         @Override
26         public CouponDiscountDTO queryDiscountDetailByOrder(OrderCouponDTO
orderCouponDTO) {
27             return null;
28         }
29
30         @Override
31         public void writeOffCoupon(List<Long> userCouponIds) {
32             throw new BizIllegalException(500, "核销优惠券异常", cause);
33         }
34     };
35 }
36 }

```

3.2.4.改造交易服务接口

最后，需要改造 `tj-trade` 中的 `com.tianji.trade.service.impl.OrderServiceImpl` 下单接口，在最后添加核销优惠券的逻辑：

```

1  @Override
2  @Transactional
3  public PlaceOrderResultVO placeOrder(PlaceOrderDTO placeOrderDTO) {
4      Long userId = UserContext.getUser();
5      // 1. 查询课程费用信息，如果不可购买，这里直接报错
6      List<CourseSimpleInfoDTO> courseInfos =
getOnShelfCourse(placeOrderDTO.getCourseIds());
7      // 2. 封装订单信息
8      Order order = new Order();
9      // 2.1. 计算订单金额
10     Integer totalAmount = courseInfos.stream()
11         .map(CourseSimpleInfoDTO::getPrice).reduce(Integer::sum).orElse(0);
12     // 2.2. 计算优惠金额
13     order.setDiscountAmount(0);
14     List<Long> couponIds = placeOrderDTO.getCouponIds();
15     CouponDiscountDTO discount = null;
16     if (CollUtils.isNotEmpty(couponIds)) {
17         List<OrderCourseDTO> orderCourses = courseInfos.stream()
18             .map(c -> new
OrderCourseDTO().setId(c.getId()).setCateId(c.getThirdCateId()).setPrice(c.getP
rice()))
19             .collect(Collectors.toList());

```

```
20         discount = promotionClient.queryDiscountDetailByOrder(new
OrderCouponDTO(couponIds, orderCourses));
21         if(discount != null) {
22             order.setDiscountAmount(discount.getDiscountAmount());
23             order.setCouponIds(discount.getIds());
24         }
25     }
26     Integer realAmount = totalAmount - order.getDiscountAmount();
27     // 2.3.封装其它信息
28     order.setUserId(userId);
29     order.setTotalAmount(totalAmount);
30     order.setRealAmount(realAmount);
31     order.setStatus(OrderStatus.NO_PAY.getValue());
32     order.setMessage(OrderStatus.NO_PAY.getProgressName());
33     // 2.4.订单id
34     Long orderId = placeOrderDTO.getOrderId();
35     order.setId(orderId);
36
37     // 3.封装订单详情
38     List<OrderDetail> orderDetails = new ArrayList<>(courseInfos.size());
39     for (CourseSimpleInfoDTO courseInfo : courseInfos) {
40         Integer discountValue = discount == null ?
41             0 :
discount.getDiscountDetail().getOrDefault(courseInfo.getId(), 0);
42         orderDetails.add(packageOrderDetail(courseInfo, order, discountValue));
43     }
44
45     // 4.写入数据库
46     saveOrderAndDetails(order, orderDetails);
47
48     // 5.删除购物车数据
49     cartService.deleteCartByUserAndCourseIds(userId,
placeOrderDTO.getCourseIds());
50
51     // 6.核销优惠券
52     promotionClient.writeOffCoupon(couponIds);
53
54     // 7.构建下单结果
55     return PlaceOrderResultVO.builder()
56         .orderId(orderId)
57         .payAmount(realAmount)
58         .status(order.getStatus())
59
        .payOutTime(LocalDateTime.now().plusMinutes(tradeProperties.getPayOrderTTLMinutes()))
60         .build();
61 }
```


3.2.5.分布式事务

注意退还优惠券、核销优惠券的分布式事务问题，可以基于Seata来解决。

3.3.退还优惠券

当用户取消订单，或者订单被超时取消时，如果用户使用了优惠券，则需要去退还优惠券。

3.3.1.接口信息

参数	说明		
请求方式	PUT		
请求路径	/user-coupons/refund		
请求参数	参数名	类型	说明
	couponIds	Array	用户券id集
返回值	无		
描述	注意： <ul style="list-style-type: none">• 校验用户券状态是否是已经使用，如果不是无需退还• 更新优惠券已经使用数量		

这个接口与核销优惠券类似，不再赘述

3.3.2.实现接口

在 `tj-promotion` 的 `com.tianji.promotion.controller.UserCouponController` 中定义接口：

```
1 @ApiOperation("退还指定优惠券")
2 @PutMapping("/refund")
3 public void refundCoupon(@ApiParam("用户优惠券id集合") @RequestParam("couponIds")
4     List<Long> userCouponIds){
5     userCouponService.refundCoupon(userCouponIds);
6 }
```

```
5 }
```

然后是 `tj-promotion` 的 `com.tianji.promotion.service.IUserCouponService` 中的方法声明：

```
1 void refundCoupon(List<Long> userCouponIds);
```

最后是 `tj-promotion` 的 `com.tianji.promotion.service.impl.UserCouponServiceImpl` 中的方法实现：

```
1 @Override
2 @Transactional
3 public void refundCoupon(List<Long> userCouponIds) {
4     // 1.查询优惠券
5     List<UserCoupon> userCoupons = listByIds(userCouponIds);
6     if (CollUtils.isEmpty(userCoupons)) {
7         return;
8     }
9     // 2.处理优惠券数据
10    List<UserCoupon> list = userCoupons.stream()
11        // 过滤无效券
12        .filter(coupon -> coupon != null && UserCouponStatus.USED ==
coupon.getStatus())
13        // 更新状态字段
14        .map(coupon -> {
15            UserCoupon c = new UserCoupon();
16            c.setId(coupon.getId());
17            // 3.判断有效期，是否已经过期，如果过期，则状态为 已过期，否则状态为 未
使用
18            LocalDateTime now = LocalDateTime.now();
19            UserCouponStatus status = now.isAfter(coupon.getTermEndTime())
?
20                UserCouponStatus.EXPIRED : UserCouponStatus.UNUSED;
21            c.setStatus(status);
22            return c;
23        }).collect(Collectors.toList());
24
25    // 4.修改优惠券状态
26    boolean success = updateBatchById(list);
27    if (!success) {
28        return;
29    }
```

```

30     // 5.更新已使用数量
31     List<Long> couponIds =
        userCoupons.stream().map(UserCoupon::getCouponId).collect(Collectors.toList());
32     int c = couponMapper.incrUsedNum(couponIds, -1);
33     if (c < 1) {
34         throw new DbException("更新优惠券使用数量失败! ");
35     }
36 }

```

3.3.3.添加FeignClient方法

在 `tj-api` 模块的 `com.tianji.api.client.promotion.PromotionClient` 中添加新的接口:

```

1 @ApiOperation("退还指定优惠券")
2 @PutMapping("/user-coupons/refund")
3 void refundCoupon(@ApiParam("用户优惠券id集合") @RequestParam("couponIds")
    List<Long> userCouponIds);

```

在 `tj-api` 模块的

`com.tianji.api.client.promotion.fallback.PromotionClientFallback` 中添加接口的降级逻辑:

```

1 package com.tianji.api.client.promotion.fallback;
2
3 import com.tianji.api.client.promotion.PromotionClient;
4 import com.tianji.api.dto.promotion.CouponDiscountDTO;
5 import com.tianji.api.dto.promotion.OrderCouponDTO;
6 import com.tianji.api.dto.promotion.OrderCourseDTO;
7 import com.tianji.common.exceptions.BizIllegalException;
8 import lombok.extern.slf4j.Slf4j;
9 import org.springframework.cloud.openfeign.FallbackFactory;
10
11 import java.util.Collections;
12 import java.util.List;
13
14 @Slf4j
15 public class PromotionClientFallback implements
    FallbackFactory<PromotionClient> {
16     @Override
17     public PromotionClient create(Throwable cause) {

```

```

18         log.error("查询促销服务出现异常, ", cause);
19         return new PromotionClient() {
20             @Override
21             public List<CouponDiscountDTO>
findDiscountSolution(List<OrderCourseDTO> orderCourses) {
22                 return Collections.emptyList();
23             }
24
25             @Override
26             public CouponDiscountDTO queryDiscountDetailByOrder(OrderCouponDTO
orderCouponDTO) {
27                 return null;
28             }
29
30             @Override
31             public void writeOffCoupon(List<Long> userCouponIds) {
32                 throw new BizIllegalException(500, "核销优惠券异常", cause);
33             }
34
35             @Override
36             public void refundCoupon(List<Long> userCouponIds) {
37                 throw new BizIllegalException(500, "退还优惠券异常", cause);
38             }
39         };
40     }
41 }

```

3.3.4.改造交易服务接口

最后，需要改造 `tj-trade` 中的 `com.tianji.trade.service.impl.OrderServiceImpl` 取消订单接口，在最后添加取消优惠券的逻辑：

```

1 @Override
2 @Transactional
3 public void cancelOrder(Long orderId) {
4     Long userId = UserContext.getUser();
5     // 1.查询订单
6     Order order = getById(orderId);
7     if (order == null || !userId.equals(order.getUserId())) {
8         throw new BadRequestException(ORDER_NOT_EXISTS);
9     }
10    // 2.判断订单状态是否已经取消，幂等判断
11    if (OrderStatus.CLOSED.equalsValue(order.getStatus())){

```

```

12      // 订单已经取消, 无需重复操作
13      return;
14  }
15  // 3.判断订单是否未支付, 只有未支付订单才可以取消
16  if(!OrderStatus.NO_PAY.equalsValue(order.getStatus())){
17      throw new BizIllegalException(ORDER_ALREADY_FINISH);
18  }
19  // 4.可以更新订单状态为取消了
20  boolean success = lambdaUpdate()
21      .set(Order::getStatus, OrderStatus.CLOSED.getValue())
22      .set(Order::getMessage, "用户取消订单")
23      .set(Order::getCloseTime, LocalDateTime.now())
24      .eq(Order::getStatus, OrderStatus.NO_PAY.getValue())
25      .eq(Order::getId, orderId)
26      .update();
27  if (!success) {
28      return;
29  }
30  // 5.更新订单条目的状态
31  detailService.updateStatusByOrderId(orderId,
    OrderStatus.CLOSED.getValue());
32
33  // 6.退还优惠券
34  promotionClient.refundCoupon(order.getCouponIds());
35  }

```

3.3.5.分布式事务

注意退还优惠券、核销优惠券的分布式事务问题, 可以基于Seata来解决。

3.4.查询优惠券

在用户中心查询订单详情的时候, 需要展示订单使用的优惠券信息:

课程信息

课程名称	课程价格	实付金额	操作
木老师的吉他入门	100.00	81.36	--
Java程序员的前端课程	149.00	110.53	--
可能是史上最全的微服务技术栈课程	199.00	148.11	--

订单总价: ¥ 448.00

优惠券: 满70减10
每满100减12, 上限200
满300减30
满100减20

优惠金额: ¥ 108.00

实付金额: ¥ 340.00

然而订单中只保存了使用过的优惠券的id:

```
CREATE TABLE `order` (  
  `id` BIGINT(19) NOT NULL COMMENT '订单id',  
  `user_id` BIGINT(19) NOT NULL COMMENT '用户id',  
  `pay_order_no` BIGINT(19) NULL DEFAULT NULL COMMENT '交易流水支付单号',  
  `status` TINYINT(3) NOT NULL DEFAULT '1' COMMENT '订单状态, 1: 待支付, 2: 已支付, 3:',  
  `message` VARCHAR(255) NOT NULL DEFAULT '' COMMENT '状态备注' COLLATE 'utf8mb4_0900_',  
  `total_amount` INT(10) NOT NULL COMMENT '订单总金额, 单位分',  
  `real_amount` INT(10) NOT NULL COMMENT '实付金额, 单位分',  
  `discount_amount` INT(10) NOT NULL DEFAULT '0' COMMENT '优惠金额, 单位分',  
  `pay_channel` VARCHAR(20) NULL DEFAULT '' COMMENT '支付渠道' COLLATE 'utf8mb4_0900_a:',  
  `coupon_ids` JSON NULL DEFAULT NULL COMMENT '优惠券id',  
  `create_time` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建订单时间',  
  `pay_time` DATETIME NULL DEFAULT NULL COMMENT '支付时间',
```

所以, 必须在查询订单详情的过程中, 根据优惠券id查询优惠规则描述信息。

3.4.1.定义接口

参数	说明		
请求方式	GET		
请求路径	/user-coupons/rules		
请求参数	参数名	类型	说明
	couponIds	Array	用户券id集
返回值	参数名	类型	说明
	rules	Array	券规则集合

3.4.2.实现接口

在 `tj-promotion` 的 `com.tianji.promotion.controller.UserCouponController` 中定义接口：

```
1 @ApiOperation("分页查询我的优惠券接口")
2 @GetMapping("/rules")
3 public List<String> queryDiscountRules(
4     @ApiParam("用户优惠券id集合") @RequestParam("couponIds") List<Long>
5     userCouponIds){
6     return userCouponService.queryDiscountRules(userCouponIds);
7 }
```

然后是 `tj-promotion` 的 `com.tianji.promotion.service.IUserCouponService` 中的方法声明：

```
1 List<String> queryDiscountRules(List<Long> userCouponIds);
```

最后是 `tj-promotion` 的 `com.tianji.promotion.service.impl.UserCouponServiceImpl` 中的方法实现：

```
1 @Override
2 public List<String> queryDiscountRules(List<Long> userCouponIds) {
3     // 1.查询优惠券信息
```

```

4     List<Coupon> coupons =
      baseMapper.queryCouponByUserCouponIds(userCouponIds, UserCouponStatus.USED);
5     if (CollUtils.isEmpty(coupons)) {
6         return CollUtils.emptyList();
7     }
8     // 2. 转换规则
9     return coupons.stream()
10        .map(c ->
      DiscountStrategy.getDiscount(c.getDiscountType()).getRule(c))
11        .collect(Collectors.toList());
12 }

```

3.4.3.添加FeignClient方法

在 `tj-api` 模块的 `com.tianji.api.client.promotion.PromotionClient` 中添加新的接口：

```

1 @ApiOperation("分页查询我的优惠券接口")
2 @GetMapping("/user-coupons/rules")
3 List<String> queryDiscountRules(@ApiParam("用户优惠券id集合")
   @RequestParam("couponIds") List<Long> userCouponIds);

```

在 `tj-api` 模块的 `com.tianji.api.client.promotion.fallback.PromotionClientFallback` 中添加接口的降级逻辑：

```

1 package com.tianji.api.client.promotion.fallback;
2
3 import com.tianji.api.client.promotion.PromotionClient;
4 import com.tianji.api.dto.promotion.CouponDiscountDTO;
5 import com.tianji.api.dto.promotion.OrderCouponDTO;
6 import com.tianji.api.dto.promotion.OrderCourseDTO;
7 import com.tianji.common.exceptions.BizIllegalException;
8 import lombok.extern.slf4j.Slf4j;
9 import org.springframework.cloud.openfeign.FallbackFactory;
10
11 import java.util.Collections;
12 import java.util.List;
13
14 @Slf4j

```



```

15 public class PromotionClientFallback implements
    FallbackFactory<PromotionClient> {
16     @Override
17     public PromotionClient create(Throwable cause) {
18         log.error("查询促销服务出现异常, ", cause);
19         return new PromotionClient() {
20             @Override
21             public List<CouponDiscountDTO>
                findDiscountSolution(List<OrderCourseDTO> orderCourses) {
22                 return Collections.emptyList();
23             }
24
25             @Override
26             public CouponDiscountDTO queryDiscountDetailByOrder(OrderCouponDTO
                orderCouponDTO) {
27                 return null;
28             }
29
30             @Override
31             public void writeOffCoupon(List<Long> userCouponIds) {
32                 throw new BizIllegalException(500, "核销优惠券异常", cause);
33             }
34
35             @Override
36             public void refundCoupon(List<Long> userCouponIds) {
37                 throw new BizIllegalException(500, "退还优惠券异常", cause);
38             }
39
40             @Override
41             public List<String> queryDiscountRules(List<Long> userCouponIds) {
42                 return Collections.emptyList();
43             }
44         };
45     }
46 }

```

3.4.4.改造交易服务接口

最后，需要改造 `tj-trade` 中的 `com.tianji.trade.service.impl.OrderServiceImpl` 查询订单详情的接口，添加查询优惠券规则描述的逻辑：

```

1 @Override
2 public OrderVO queryOrderById(Long id) {

```

```

3    // 1.查询订单
4    Order order = getById(id);
5    if (order == null) {
6        throw new BadRequestException(ORDER_NOT_EXISTS);
7    }
8    // 2.查询订单详情
9    List<OrderDetail> details = detailService.queryById(id);
10   // 3.转换VO
11   // 3.1.订单
12   OrderVO vo = BeanUtils.toBean(order, OrderVO.class);
13   // 3.2.订单详情
14   List<OrderDetailVO> dvs = BeanUtils.copyList(details, OrderDetailVO.class,
15       (d, v) -> v.setCanRefund(
16           // 订单已经支付, 且 退款没有在进行中, 标记为可退款状态
17           OrderStatus.canRefund(d.getStatus()) &&
18           !RefundStatus.inProgress(v.getRefundStatus())
19       ));
20   vo.setDetails(dvs);
21   // 3.3.订单进度
22   vo.setProgressNodes(detailService.packageProgressNodes(order, null));
23   // 3.4.优惠明细
24   List<String> rules =
25       promotionClient.queryDiscountRules(order.getCouponIds());
26   vo.setCouponDesc(String.join("/", rules));
27   return vo;
28 }

```

4.面试问题

4.1.你们的优惠券规则是如何编码实现的？

答：我们的优惠规则是基于策略模式来定义的。在初期做调研的时候也考虑过规则引擎，不过考虑到我们的优惠规则并不复杂，而且规则引擎太重，增加了学习和维护成本，最终选择了基于策略模式来自定义规则。

4.2.你在项目中有使用到设计模式？

答：当然用到过，比如在优惠券功能中就使用了策略模式来定义优惠规则。还有我实现的基于注解的通用分布式锁组件，也使用到了策略模式、工厂模式

4.3.你在项目中有没有使用到线程池或者并发编程？

答：当然，项目中很多地方都有用到。比如在实现优惠券的推荐算法时，我们采用的是排列组合多种优惠方案，然后分别计算，最终筛选出最优解的思路。

由于需要计算的优惠方案可能较多，为了提高计算效率，我们利用了CompletableFuture来实现多方案的并行计算。并且由于要筛选最优解，那就需要等待所有方案都计算完毕，再来筛选。因此就使用了CountDownLatch来做多线程的并行控制。

4.4.那你能不能聊一聊CountDownLatch的基本原理？

略，参考面试宝典

4.5.使用优惠券的订单可能包含多个商品，如果出现部分商品退款的情况，你们如何处理退款金额？优惠券是如何处理的？

答：这里处理的方案有很多种，可以选择退券或不退券。不过基于产品的需求，我们采用的是不退券的方案。

具体来说，就是在一开始下单时，就会根据优惠券本身的使用范围，筛选出订单中可以参与优惠的商品，然后计算出每一个被优惠的商品具体的优惠金额分成，以及对应的实付金额。

而在退款的时候，如果用户选择只退部分商品，我们就可以根据每个商品的实付金额来退款，实现订单拆分退款。同时也满足退款不退券的原则。

当然，如果订单未支付，直接取消或者超时关闭，是可以退还优惠券的。