项目介绍

1 项目代码

1.1 代码结构

- mall-parent 是所有其他maven工程的父工程，不包含任何代码， 集中管理我们项目中所使用到的，第三方开源框架的版本
- .mall-commons(mall-core， mall-lock， mall-mq， mall-tool)

  1）commons-core 就放了一些公共的请求，响应的公共类等

  2) commons-lock 分布式锁，相关的一些公共工具类

  3) commons-mq 放和消息中间件相关的公共的工具类

  4) commons-tool 放的是一些，很多模块都需要使用到的一些工具类
- gateway: 我们自己实现的一ontroller实现)个简易网关(Controller)
- order-service，shopping-service， user-service

  1）每一个service，就代表了一个服务

  2) xxx-api 该服务对外暴露的公共接口，以及公共的请求和响应的实体类

  3) xxx-provider 该工程中的代码，才是真正的服务提供者的实现代码(服务的实现)

  4）xxx-provider 一定依赖于 xxx-api

1.2 代码细节

注意：在我们的整个项目，不会有任何一个maven工程依赖于xxx-provider

- gateway

  1）站在请求接入的角度，只有gateway需要对外接收http请求，gateway需要运行在web服务器上， 所以依赖spring-

  boot-starter-web

  2）它可以独立启动(独立运行在tomcat服务器上的)

  3）站在服务的角度，gateway是服务的消费者，调用服务实现对应的业务功能，所以它 会依赖对应服务暴露的公

  共，所以它会依赖各个服务对应的 xxx-api

  4）多环境配置，相应的dubbo配置
- order-service，shopping-service， user-service

  1）xxx-api

  a. 公共的响应码及对应的描述字符串(枚举类，或者常量类)

  b. dto，服务提供者，和服务的消费者，都需要使用到的，请求或者响应的dto类

  2）xxx-provider

  a. 每一个xxx-provider作为真正的服务实现者，可以独立启动的

  b. 有一个包service，这个包中，放的就是对外暴露的服务接口的实现

  c. converter对应的DO类与DTO类的转化器

d. dal，里面放了访问的数据库Mapper，以及Mapper对应的映射文件

e. provider要能正常启动，必须先启动zookeeper，有可能还需要启动redis

3) TokenInterceptor

2 数据库

- tb_address 地址信息
- tb_express 物流商信息
- tb_item 商品表，存储商品的详细信息

```
1. tb_item表中的image字段是字符串类型的，其中包含多个url字符串，用逗号分隔，表示一件商品
 的多张图片
2. 在tb_item对应的实体类Item中，有一个字段叫imageBig，其值是image字段中的第一张图片
 url，可以看下Item类的getImageBig()方法
```

- tb_item_cat 商品类目表

```
该表主要存储了，首页导航栏商品中，所显示的各个商品类目，以及商品类目之间的父子关系，类目之 间的
关系主要通过parentId字段来体现
```

- tb_item_desc 商品详情表

```
所谓商品的详情数据其实主要就是一张图片
```

- tb_member 用户表存储用户注册信息

```
1. 需要注意用户的激活状态isverified字段，它表示用户注册的账号是否被激活
2. username，phone，email都有唯一索引，即用户注册时，所有用户的username，phone，email不
能重复
```

- tb_order 订单表，需要关注订单的状态，其取值以常量类OrderConstants中定义的为准
- tb_order_item 订单商品条目表
- tb_order_shipping 订单的邮寄信息
- tb_panel 定义了首页显示的各个版块

```
在首页中，每一个版块，包含多个版块商品，所以tb_panel和tb_panel_content是一对多的关系
```

- tb_panel_content

```
1. HEADER_PANEL_ID=0;//导航栏板块表id
   对于查询导航栏数据，直接在tb_panel_content表中查询id为HEADER_PANEL_ID的数据即可
2. RECOMMEND_PANEL_ID=6;//推荐商品板块表id
   对于推荐商品接口数据的查询，在tb_panel_content表中查询，id为RECOMMEND_PANEL_ID的数据
即可
```

- tb_payment 支付表，tb_promo_session 秒杀场次表， tb_promo_item 秒杀商品表（暂时不用管，这是第二个阶段所要完成的功能）

```
1. stock_count 代表的是商品的可售卖库存
2. lock_count 代表的是用户下单，但是还未支付成功的商品数量
```

- tb_user_verify 用户激活验证表，记录了用户注册账号的激活信息

3 细节

3.1 注册邮件发送

- 激活邮件的发送，最好实现异步发送，最简单的方式是，在子线程中运行邮件发送的代码
- 不要偷懒，最好用自己的账号去发送邮件

3.2 登录逻辑

1. 用户登录成功后，JWT放入Cookie中返回给用户
2. 当用户登录时，登录请求首先被TokenInterceptor拦截，如果是静态资源请求，或者请求对应的Controller方法有Anonymous注解，则放行
3. 否则，取请求中的Cookie，看下名为acess_token的cookie，如果没有，则说明用户未登录，返回响应让用户登录
4. 如果有，调用LoginService中的方法，解密验证cookie中的token，如果解密失败则让用户重新登录。
5. 否则，如果解密验证成功，则将封装到token中的数据取出(一个包含用户名和用户id的Map)，放入所拦截的请求的Request对象的，名为“userInfo”的属性中
6. 所以，在首次登录之后，如何判断用户是否登录过呢？在判断用户是否登录过的Controller方法中，只要看下request中的userInfo属性有没有值，就可判断用户是否登录过了。

3.3 购物车

在我们的项目中，我们采用redis来存储用户的购物车数据。

那么用redis中的5种基本数据结构中的哪一种来存储用户的购物车数据合适呢？ hash数据结构

```
key(用户的uid)      value(hash数据结构)
                      field      value
                      productId  购物车商品对象
```

3.4 订单

- 所谓发号器，简单来说就是在下单的时候，生成订单的和订单商品条目的全局唯一id的。其实现方式可以有多种，在项目中对应的工具为order- provider中的com.mall.order.utils.GlobalIdGeneratorUtil，调用该类的getNextSeq方法，即可获取全局唯一id。

```
/*
keyname: 表示在redis中存储的，上一个全局唯一id的key，不同的全局唯一id， 定义不同的key，比如订单的和订单条目的全局唯一id，就需要定义不同的key
increby: 全局唯一id的增量，比如通常给1 生成全局唯一id的逻辑
具体生成过程：
1. 每次生成全局唯一id之前，先生成一个备选的唯一id值，该值的组成如下： 15位精确到毫秒的时间字符(yyMMddHHmmssSSS) + 随机生成的位数的随机字符串
2. 去redis中找下，有没有key为keyname对应的value值，如果没有那么就直接将备选的id值当做当前的全局唯一id值，以 keyname为key，存储到redis中，并返回给调用者
3. 如果keyname对应的key有值，则比较一下，keyname对应的value值和当前的备用全局唯一id值，如果keyname应的值存储到redis，并返回该值value值小，则将备选的id值当做当前的全局唯一id值作为keyname对
4. 否则，取出keyname对应的值，给该值 + incrby，作为keyname对应的新的全局唯一id的值存储，并返回
*/
public String getNextSeq(String keyName, int incrby) {}
```

- 在下订单的时候，其中有一步是扣减库存，需要大家自己去tb_stock这张表中，增加一下对应，下单商品的库存!

4 服务的超时与重试

4.1 服务调用超时

- 通常，一次远程调用过程中，服务消费者不可能无限制的等待服务提供者返回的结果
- 正常情况下，服务提供者的一次调用执行过程也会执行很长时间(除非服务提供者出了问题)
- 所以为防止非正常情况下服务消费者在调用过程中的长时间等待，对于一次服务调用过程，我们会设置其超时时间。
- 超时未返回即认为调用失败

4.2 服务的重试

- 服务之间的调用需要经过网络，而网络是一个不稳定因素，一旦出现网络抖动，就可能会导致服务调用失败，或者服务调用超时。
- 为了防止因为网络抖动或其他偶然因素而出现的服务调用失败的情况，Dubbo在某服务调用超时或失败的时候，会自动重试对该服务的调用
- 而当服务调用失败时，重试的此时也是可以设置的

```
/*
timeout: 超时时间，以ms为单位，因为我们是个人pc，而且同时运行很多其他软件，所以建议设置为3000，默认1000 retries：服务调用超时或失败时，自动重试的次数默认2次，如果要debug，为了不受重试影响，建议 设置为0
check: 设置消费者启动的时候，是否检查服务提供者的状态，默认为true，建议设置为false
*/
@Reference(timeout = 3000,retries = 2, check = false)
IAddressService addressService;
```

5 测试

在测试的时候，不要动不动就打开前端测试，那样的话测试效率很低，建议：

- 用单元测试测试对数据库的访问，以及服务的调用

```
    // 测试数据库访问
```

```
    @Autowired
    ItemMapper itemMapper;
    @Test
    public void testMapper() {
        Item item = itemMapper.selectByPrimaryKey(100023501);
        System.out.println(item);
    }

    // 测试服务
    @Reference
    ITestProductDetailService productDetailService;
    @Test
    public void testService() {
        TestProductDetailRequest request = new TestProductDetailRequest();
        request.setProductId(100023501L);
        TestProductDetailResponse productDetail =
productDetailService.getProductDetail(request);
    }
```

- 用postman测试网关Controller
- 最后在打开前端来测试某个功能的整个流程

注意如果使用单元测试的话，一定要想停止正在运行的服务，否则会出现端口占用的错误

6 前端

在启动前端项目的时候，会用到cnpm命令(注意一定要使用cnpm命令，使用npm配置淘宝镜像是不行的)，但是可能部分同学的cnpm命令使用的时候可能会报错

此时，可能并非是cnpm有问题，而是环境变量的问题。所以首先找到你的cnpm命令所在的位置

然后，在该目录下，打开一个命令行，输入cnpm，如果如下图所示的提示信息(图片没截全)，那么就说明cnpm命令没问题，只需要配置cnpm命令的path环境变量即可