本文主要知识点如下：

* 新蜂商城搜索功能分析
* 新蜂商城搜索接口传参解析及返回字段定义
* 新蜂商城搜索接口实现分析
* 新蜂商城搜索接口实现

## 新蜂商城搜索功能分析

首页接口和分类接口已介绍完毕，接下来就是商品列表页面的制作，我们需要完成商品搜索的后端接口，在功能实现之前，我们先来想一下，进入到商品列表页面会有哪些入口？

* 搜索框中输入关键字并点击回车
* 点击三级分类跳转至搜索列表页面

当前 Vue 版本的代码中，搜索框在首页、商品分类页、商品列表页面都有，这些页面都可以触发商品搜索接口，效果如下图所示：



当然，以上为关键字搜索，而商品分类页面中，点击对应的商品分类还可以根据商品分类 id 来搜索商品，效果如下图所示：



商品列表数据的展示则只有一个页面，即商品列表页，不管是上述的哪种方式，都会跳转到商品列表页并调用后端的商品搜索接口，之后进行数据的渲染和分页逻辑的实现，本文我们主要的内容就是商品搜索接口的分析和实现。

获取用户输入的搜索关键字或者分类 id，跳转到搜索结果页面，搜索结果页面渲染，这些都是前端开发人员来实现，前端页面实现这些效果时，需要获取当前页面要渲染的数据，也就是在跳转到搜索结果页后，会组装商品搜索的请求参数并向后端接口发送搜索请求，拿到后端返回的数据后才会进行页面渲染。

后端开发人员需要定义接口参数、接口地址、接口返回字段，后端接口需要接收前端传过来的参数，根据参数查询出数据，组装数据并返回给前端，之后由前端进行渲染。

这是当前新峰商城 Vue 版本的商品搜索实现逻辑，当然，大家可以灵活改动。

## 新蜂商城搜索接口传参解析及返回字段定义

#### 搜索接口传参

页面 UI 还原和样式编写是由前端开发人员来实现，前端开发人员包括 Web 开发和移动端原生 APP 开发人员，他们负责页面及交互，我们后端开发人员则需要实现一个列表数据的返回接口，刚刚我们也已经提到了，根据用户输入的关键字和或者用户点击的分类，来返回的数据，即通常我们所说的搜索接口。

因此搜索接口传参的两个重要参数为搜索关键字+商品的三级分类 id，我们将其分别定义为 keyword 和 goodsCategoryId，类型为 String 和 Long。另外还有两个参数是排序方式 orderBy 和 分页参数 pageNumber。

#### 分页逻辑

虽说是返回一个列表，不过有一个隐藏的知识点，那就是该接口返回的数据可能数据条数很多，因此我们也需要加入分页的逻辑，前文中的两个搜索动图效果中，在展示搜索内容后，我在页面上不断往上拉，数据也会不断的加载进来，这里就用到了分页逻辑，在移动端的实现效果就是大家常说的“上拉加载”。虽然没有分页页码和翻页按钮，这里依然是分页展示的逻辑，移动端页面通常的做法就是如此，毕竟移动端的操作习惯与 PC 端不同，移动端页面的面积也比 PC 端小了很多，不可能完全做成 PC 端的分页效果。

分页数据我也做了一个封装，因为所有的分页结果基本都包括以下几个属性，分页结果集的数据格式定义如下（注：完整代码位于ltd.newbee.mall.utils.PageResult）：

*//分页的通用结果类***public** **class** **PageResult** **implements** **Serializable** {

*//总记录数*

**private** **int** totalCount;

*//每页记录数*

**private** **int** pageSize;

*//总页数*

**private** **int** totalPage;

*//当前页数*

**private** **int** currPage;

*//列表数据*

**private** List<?> list;

}

实现分页功能中的返回对象 PageResult 中定义了以下四个参数：当前页的所有数据列表、当前页的页码、总页码、总记录数量，并将其放入 Result 返回结果的 data 属性中，在商品搜索接口中我们最终得到的返回对象为： Result<PageResult<List<NewBeeMallSearchGoodsVO>>>，最外层是 Result 对象，这个不用多说，所有的接口返回类型都是 Result 类型，里面一层是 PageResult 对象，因为搜索接口需要返回分页信息，PageResult 对象中则是具体的当前页所需要的商品列表信息 List<NewBeeMallSearchGoodsVO>。

之后由前端开发人员对直接读取对应的参数并这些数据进行处理，这就是前后端进行数据交互时分页数据的格式定义，希望大家能够结合代码以及实际的分页效果进行理解和学习。

#### 返回数据格式

如下图所示即为商品分页列表中需要渲染的内容，首先商品分页列表肯定是一个 List 对象。同时，因为有分页功能，还需要返回分页字段，因此最终接收到的结果返回格式为 PageResult 对象，而列表中的单项对象中的字段则需要通过下图中的内容进行确认。



通过图片我们可以看到商品预览图字段、商品标题字段、商品简介字段、商品价格字段，当然这里通常会设计成可跳转的形式，即点击标题或者预览图后会跳转到对应的商品详情页面中，因此还需要一个商品实体的 id 字段，因此返回数据的格式就得出来了，编码如下：

**package** ltd.newbee.mall.controller.vo;**import** java.io.Serializable;

*/\*\**

*\* 搜索列表页商品VO*

*\*/***public** **class** **NewBeeMallSearchGoodsVO** **implements** **Serializable** {

**private** Long goodsId;

**private** String goodsName;

**private** String goodsIntro;

**private** String goodsCoverImg;

**private** Integer sellingPrice;

**public** Long **getGoodsId**() {

**return** goodsId;

}

**public** **void** **setGoodsId**(Long goodsId) {

this.goodsId = goodsId;

}

**public** String **getGoodsName**() {

**return** goodsName;

}

**public** **void** **setGoodsName**(String goodsName) {

this.goodsName = goodsName;

}

**public** String **getGoodsIntro**() {

**return** goodsIntro;

}

**public** **void** **setGoodsIntro**(String goodsIntro) {

this.goodsIntro = goodsIntro;

}

**public** String **getGoodsCoverImg**() {

**return** goodsCoverImg;

**public** **void** **setGoodsCoverImg**(String goodsCoverImg) {

this.goodsCoverImg = goodsCoverImg;

}

**public** Integer **getSellingPrice**() {

**return** sellingPrice;

}

**public** **void** **setSellingPrice**(Integer sellingPrice) {

this.sellingPrice = sellingPrice;

}

}

## 搜索接口实现

#### 商品表结构介绍

接下来是数据查询的功能实现，上述商品搜索分页列表中的字段，可以通过直接查询 tb\_newbee\_mall\_goods\_info 商品表来获取，表结构如下：

**CREATE** **TABLE** `tb\_newbee\_mall\_goods\_info` (

`goods\_id` **bigint**(20) unsigned **NOT** **NULL** AUTO\_INCREMENT COMMENT '商品表主键id',

`goods\_name` **varchar**(200) **NOT** **NULL** **DEFAULT** '' COMMENT '商品名',

`goods\_intro` **varchar**(200) **NOT** **NULL** **DEFAULT** '' COMMENT '商品简介',

`goods\_category\_id` **bigint**(20) **NOT** **NULL** **DEFAULT** '0' COMMENT '关联分类id',

`goods\_cover\_img` **varchar**(200) **NOT** **NULL** **DEFAULT** '/admin/dist/img/no-img.png' COMMENT '商品主图',

`goods\_carousel` **varchar**(500) **NOT** **NULL** **DEFAULT** '/admin/dist/img/no-img.png' COMMENT '商品轮播图',

`goods\_detail\_content` text **NOT** **NULL** COMMENT '商品详情',

`original\_price` **int**(11) **NOT** **NULL** **DEFAULT** '1' COMMENT '商品价格',

`selling\_price` **int**(11) **NOT** **NULL** **DEFAULT** '1' COMMENT '商品实际售价',

`stock\_num` **int**(11) unsigned **NOT** **NULL** **DEFAULT** '0' COMMENT '商品库存数量',

`tag` **varchar**(20) **NOT** **NULL** **DEFAULT** '' COMMENT '商品标签',

`goods\_sell\_status` tinyint(4) **NOT** **NULL** **DEFAULT** '0' COMMENT '商品上架状态 1-下架 0-上架',

`create\_user` **int**(11) **NOT** **NULL** **DEFAULT** '0' COMMENT '添加者主键id',

`create\_time` datetime **NOT** **NULL** **DEFAULT** CURRENT\_TIMESTAMP COMMENT '商品添加时间',

`update\_user` **int**(11) **NOT** **NULL** **DEFAULT** '0' COMMENT '修改者主键id',

`update\_time` datetime **NOT** **NULL** **DEFAULT** CURRENT\_TIMESTAMP COMMENT '商品修改时间',

**PRIMARY** KEY (`goods\_id`) **USING** BTREE

) ENGINE=InnoDB **DEFAULT** CHARSET=utf8 ROW\_FORMAT=**DYNAMIC**;

新蜂商城第一版的表结构也是如此，新版本中没有改动，表中有商品名称、商品简介以及商品的分类 id，因此在查询时，只对该表进行字段过滤的查询即可，并不会涉及到多表操作。

#### 搜索商品的业务代码实现

主要的实现逻辑在 NewBeeMallGoodsService 业务类的 searchNewBeeMallGoods() 方法中，代码如下：

**public** PageResult **searchNewBeeMallGoods**(PageQueryUtil pageUtil) {

List<NewBeeMallGoods> goodsList = goodsMapper.findNewBeeMallGoodsListBySearch(pageUtil);

**int** total = goodsMapper.getTotalNewBeeMallGoodsBySearch(pageUtil);

List<NewBeeMallSearchGoodsVO> newBeeMallSearchGoodsVOS = **new** **ArrayList**<>();

**if** (!CollectionUtils.isEmpty(goodsList)) {

newBeeMallSearchGoodsVOS = BeanUtil.copyList(goodsList, NewBeeMallSearchGoodsVO.class);

**for** (NewBeeMallSearchGoodsVO newBeeMallSearchGoodsVO : newBeeMallSearchGoodsVOS) {

**String** goodsName = newBeeMallSearchGoodsVO.getGoodsName();

**String** goodsIntro = newBeeMallSearchGoodsVO.getGoodsIntro();

*// 字符串过长导致文字超出的问题*

**if** (goodsName.length() > 28) {

goodsName = goodsName.substring(0, 28) + "...";

newBeeMallSearchGoodsVO.setGoodsName(goodsName);

}

**if** (goodsIntro.length() > 30) {

goodsIntro = goodsIntro.substring(0, 30) + "...";

newBeeMallSearchGoodsVO.setGoodsIntro(goodsIntro);

}

}

}

**PageResult** pageResult = **new** **PageResult**(newBeeMallSearchGoodsVOS, total, pageUtil.getLimit(), pageUtil.getPage());

**return** pageResult;

}

我们定义了 searchNewBeeMallGoods() 方法并传入 PageQueryUtil 对象作为参数，商品类目 id 、关键字 keyword 字段、分页所需的 page 字段、排序字段等都作为属性放在了这个对象中，关键字或者商品类目 id 用来过滤想要的商品列表，page 用于确定查询第几页的数据，之后通过 SQL 查询出对应的分页数据，再之后是填充数据，某些字段太长会导致页面上的展示效果不好，所以对这些字段进行了简单的字符串处理并设置到 NewBeeMallSearchGoodsVO 对象中，最终返回的数据是 PageResult 对象，包含了当前页返回的商品列表数据还有分页信息。

#### 查询 SQL

具体的 SQL 语句如下（注：完整代码位于 ****resources/mapper/ NewBeeMallGoodsMapper.xml****）：

<select id="findNewBeeMallGoodsListBySearch" parameterType="Map" resultMap="BaseResultMap">

select

<include refid="Base\_Column\_List"/>

from tb\_newbee\_mall\_goods\_info

<where>

<if test="keyword!=null and keyword!=''">

and (goods\_name like CONCAT('%',#{keyword},'%') or goods\_intro like CONCAT('%',#{keyword},'%'))

</if>

<if test="goodsCategoryId!=null and goodsCategoryId!=''">

and goods\_category\_id = #{goodsCategoryId}

</if>

</where>

<if test="orderBy!=null and orderBy!=''">

<choose>

<when test="orderBy == 'new'">

*<!-- 按照发布时间倒序排列 -->*

order by goods\_id desc

</when>

<when test="orderBy == 'price'">

*<!-- 按照售价从小到大排列 -->*

order by selling\_price asc

</when>

<otherwise>

*<!-- 默认按照库存数量从大到小排列 -->*

order by stock\_num desc

</otherwise>

</choose>

</if>

<if test="start!=null and limit!=null">

limit #{start},#{limit}

</if>

</select>

<select id="getTotalNewBeeMallGoodsBySearch" parameterType="Map" resultType="int">

select count(\*) from tb\_newbee\_mall\_goods\_info

<where>

<if test="keyword!=null and keyword!=''">

and (goods\_name like CONCAT('%',#{keyword},'%') or goods\_intro like CONCAT('%',#{keyword},'%'))

</if>

<if test="goodsCategoryId!=null and goodsCategoryId!=''">

and goods\_category\_id = #{goodsCategoryId}

</if>

</where>

</select>

根据前端传输过来的关键字和商品类目 id 对商品记录进行检索，使用 MySQL 数据库的 LIKE 语法对关键字进行过滤，根据 goods\_category\_id 字段对商品类目进行过滤，之后是根据 orderBy 字段来进行商品搜索分页结果的排序，最后是 start 和 limit 两个分页所必需的的关键字（注意：start 和 limit 是项目中定义的，也可以命名为其它的字段，比如 startNum 和 size，****为了避免给大家造成误导，这里做一下解释，字段的命名你可以自行决定，别报错就行****）。

#### 接口实现

最终，我们的商品搜索接口实现代码如下：

**@GetMapping("/search")**

**@ApiOperation(value = "商品搜索接口", notes = "根据关键字和分类id进行搜索")**

**public** Result<PageResult<List<NewBeeMallSearchGoodsVO>>> **search**(**@RequestParam(required = false)** **@ApiParam(value = "搜索关键字")** String keyword,

**@RequestParam(required = false)** **@ApiParam(value = "分类id")** Long goodsCategoryId,

**@RequestParam(required = false)** **@ApiParam(value = "orderBy")** String orderBy,

**@RequestParam(required = false)** **@ApiParam(value = "页码")** Integer pageNumber,

**@TokenToMallUser** MallUser loginMallUser) {

logger.info("goods search api,keyword={},goodsCategoryId={},orderBy={},pageNumber={},userId={}", keyword, goodsCategoryId, orderBy, pageNumber, loginMallUser.getUserId());

**Map** params = **new** **HashMap**(4);

*//两个搜索参数都为空，直接返回异常*

**if** (goodsCategoryId == null && StringUtils.isEmpty(keyword)) {

NewBeeMallException.fail("非法的搜索参数");

}

**if** (pageNumber == null || pageNumber < 1) {

pageNumber = 1;

}

params.put("goodsCategoryId", goodsCategoryId);

params.put("page", pageNumber);

params.put("limit", Constants.GOODS\_SEARCH\_PAGE\_LIMIT);

*//对keyword做过滤 去掉空格*

**if** (!StringUtils.isEmpty(keyword)) {

params.put("keyword", keyword);

}

**if** (!StringUtils.isEmpty(orderBy)) {

params.put("orderBy", orderBy);

}

*//搜索上架状态下的商品*

params.put("goodsSellStatus", Constants.SELL\_STATUS\_UP);

*//封装商品数据*

**PageQueryUtil** pageUtil = **new** **PageQueryUtil**(params);

**return** ResultGenerator.genSuccessResult(newBeeMallGoodsService.searchNewBeeMallGoods(pageUtil));

}

方法名称为 search()，请求类型为 GET，路径映射为 /api/v1/search，所有的传参我们都是用 @RequestParam注解进行接收，前端传过来的参数主要有：

* keyword
* goodsCategoryId
* orderBy
* pageNumber

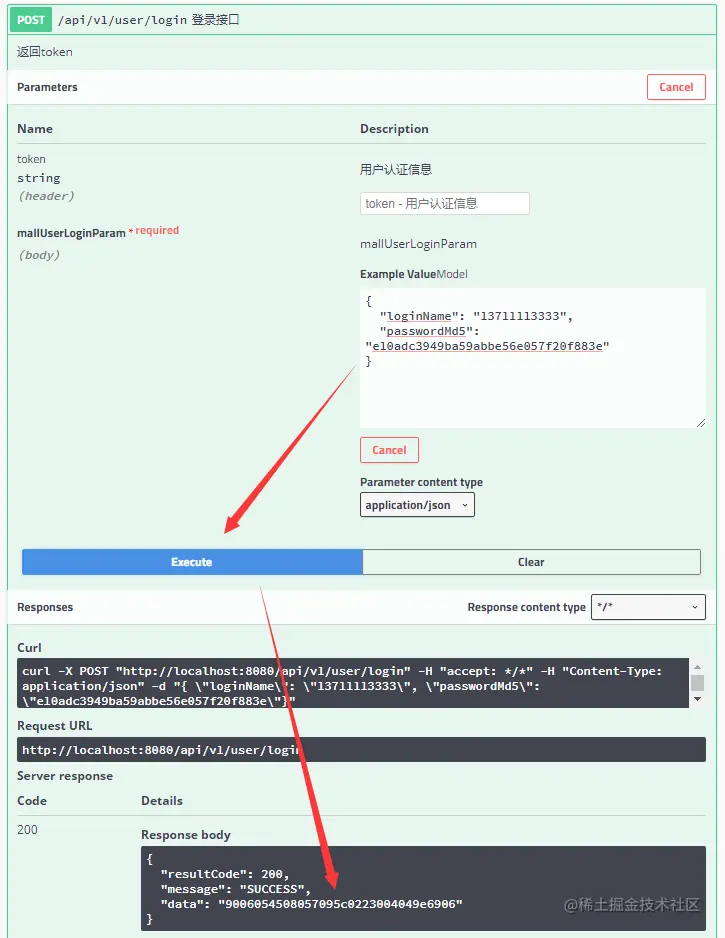
pageNumber 参数是分页所必需的字段，如果不传的话默认为第 1 页，keyword 参数是关键字，用来过滤商品名和商品简介，goodsCategoryId 参数是用来过滤商品分类 id 的字段，orderBy 则是排序字段，传过来不同的排序方式，返回的数据也会不同。另外还有一个参数是当前登陆用户的信息，已经用 @TokenToMallUser 注解来接收，相关逻辑在之前的文章中已经介绍过，这里不做赘述。根据以上字段进行查询参数的封装，之后通过 SQL 查询出对应的分页数据 pageResult 。响应结果类型为 Result，实际的 data 属性类型为 PageResult<List<NewBeeMallSearchGoodsVO>> 对象，即商品搜索结果的分页列表数据。

实现逻辑就是调用商品业务实现类 NewBeeMallGoodsService 中的查询方法，将所需的数据查询并响应给前端，所有的实现逻辑都是在业务实现类中处理的，包括查询和返回字段的内容设置，在控制层代码中，主要是进行参数判断和参数的封装，之后将获得的数据结果设置到 Result 对象并返回。

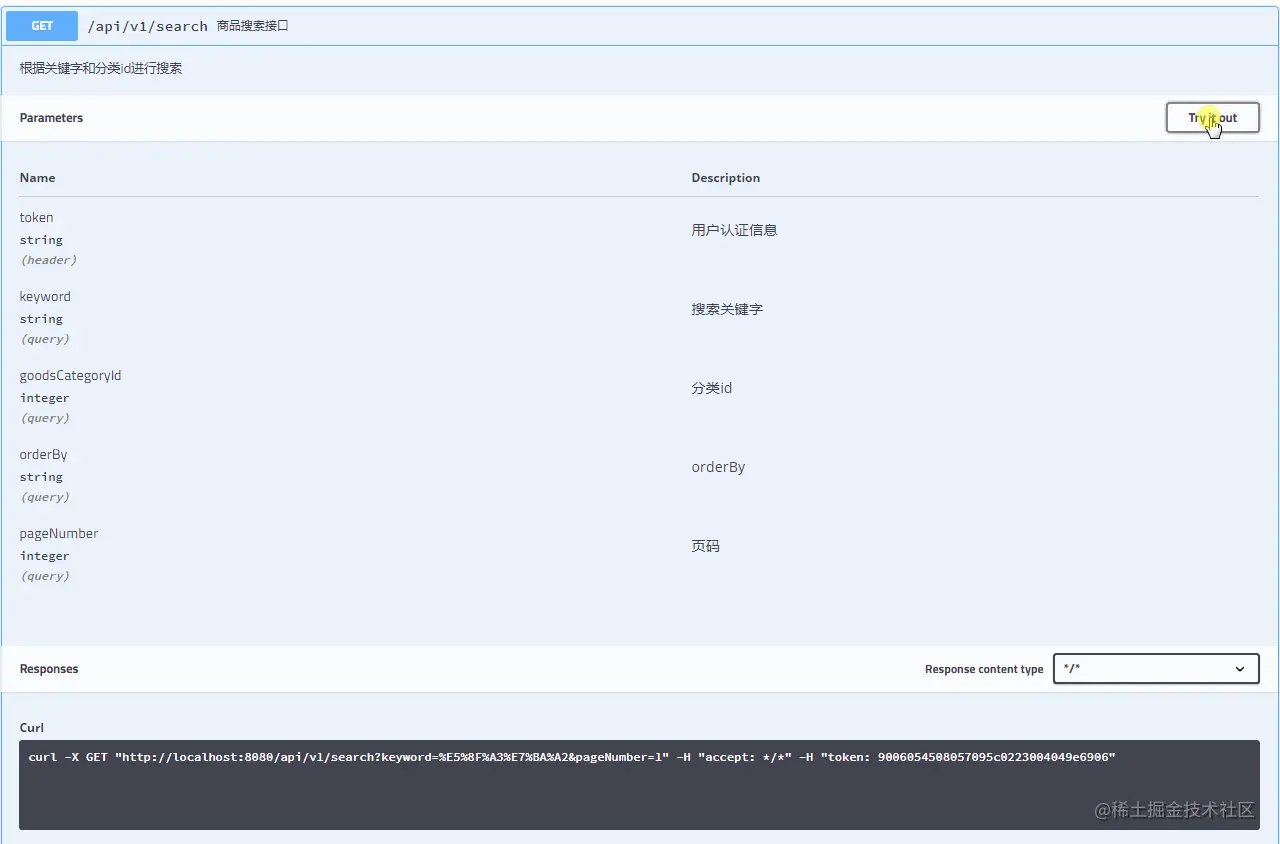
#### 商品搜索接口测试

最后，我们通过 Swagger 页面来测试一下商品搜索接口。

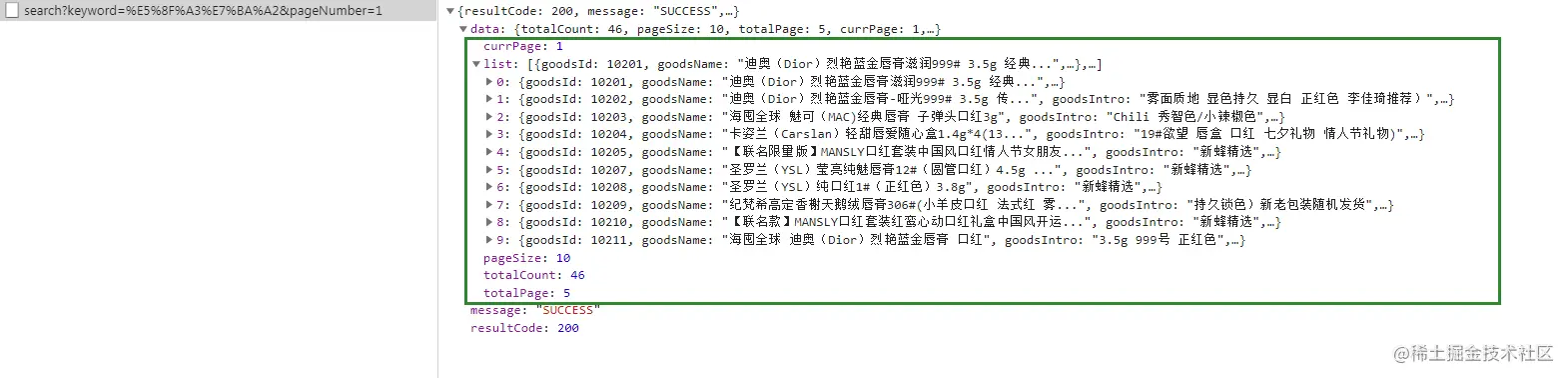
重启项目，打开 swagger-ui 页面，点开“商品搜索接口”，该接口需要登陆状态下才能访问，因此需要先调用登陆接口获取一个 token 值，如下图所示：



之后在 token 输入框中填入刚刚获取的 token 值，然后输入关键字参数和页码参数，之后点击”Execute“按钮，这里我查询的是关键字是“口红”，页码分别传了 1 和 2，即查询了两次，分别是第一页的数据和第二页的数据，整个测试过程如下图所示：



截图无法截全，这里我们 F12 打开控制台，通过控制台中 Network 面板中的内容来看一下返回的数据结构，如下图所示，这里我们点开刚刚请求的 /api/v1/search 链接，左侧点击 Preview 选项卡就可以看到返回的数据：



前端接收到的是一个 Result 类型的 json 格式数据，包括 resultCode、message、data 三个键值对，搜索结果就在 data 键中，不仅仅包含了商品列表对象，还有分页信息，包括总页数、当前页数等数据。

前端获取到这些数据后，就可以进行商品列表的页面渲染，也可以根据用户上拉和加载进行翻页操作，商品搜索接口实现成功！