## 购物车功能模块

在商品功能模块和订单结算模块之间，有这样一个功能模块，它是整个购物环节中由商品到订单转化过程中，处于一个中间状态的模块，将商品和订单这两个模块进行打通，它就是购物车模块，这篇文章我们主要来介绍一下购物车模块相关接口的实现，同时也会介绍一下商品详情接口的开发。

我会从功能介绍讲起，并介绍页面设计和排版，最后是功能实现和页面展示，主要功能点如下：

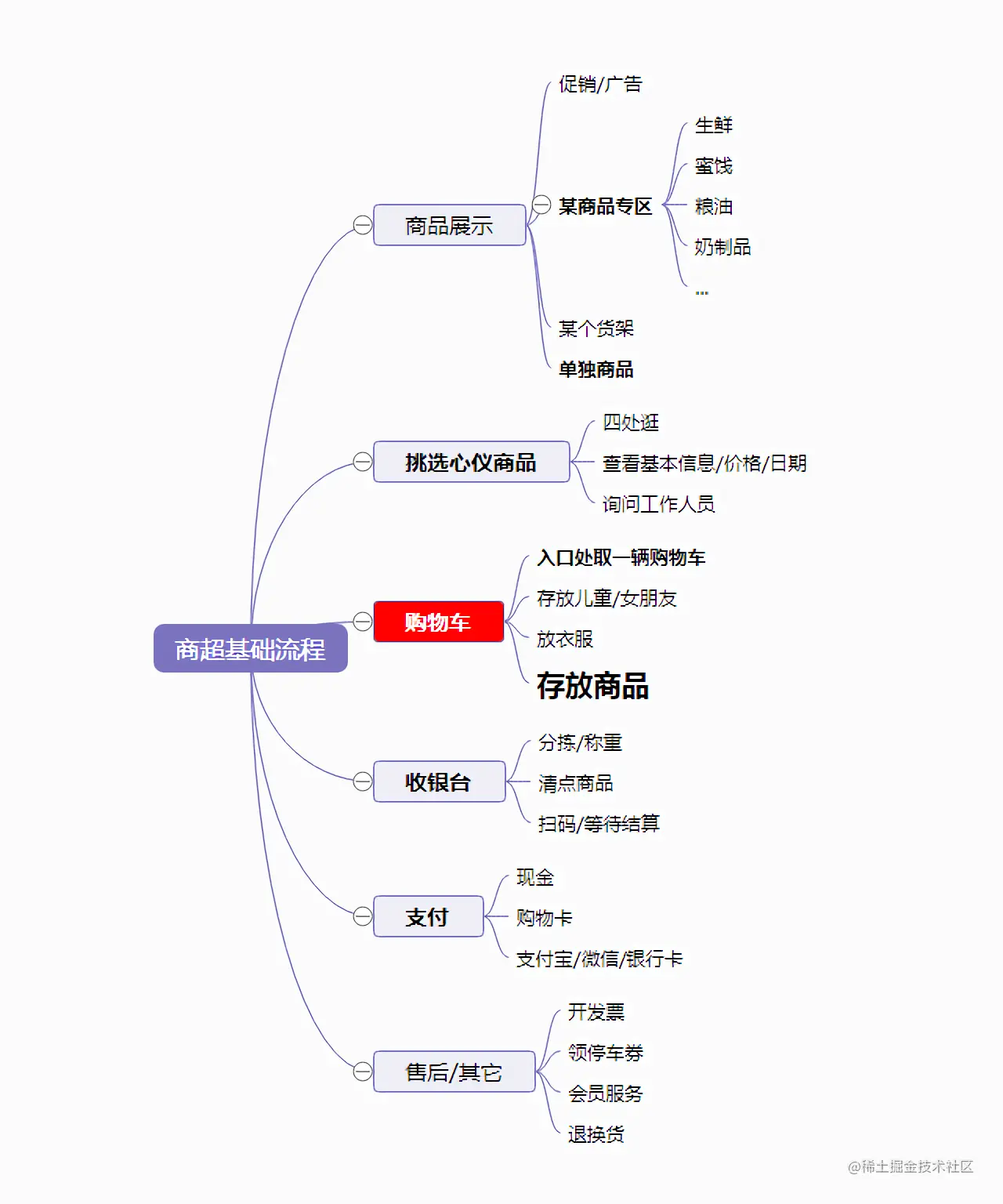
* 商品详情接口
* 加入购物车接口
* 购物车列表接口
* 购物项编辑接口
* 购物项删除接口

## 购物车模块简介

为了让大家更好的理解购物车这个功能点，我们先从线下超市或者商场中的购物流程讲起，过程如下：

1. 进入商场
2. 在入口处取一辆购物车或者购物篮
3. 四处逛
4. 在不同的区域选择不同的商品（水果、茶叶、蔬菜、熟食、衣服、家用品等等）
5. 经过一番筛选后，将想要购买的货品放入购物车/购物篮
6. 某些商品需要称重或者其他处理
7. 到收银台清点商品并计算价格
8. 结账（微信/支付宝/刷卡）
9. 离开商场、回家

简单梳理一下购物流程：筛选商品、放入购物车、清点结算、付钱，这是日常生活中大家都有的体验。当然，可能还有其他的一些流程或逻辑，根据生活经验我们可以整理出它的大概流程，如下图所示：



大部分线上商城中的购物车功能模块都是将线下的这个购物车进行了抽象而开发的一个功能，新蜂商城也是如此，与线下实体的购物车不同，它不能用来放衣服，孩子/女朋友也无法坐在购物车里，它的作用就是用来存放用户挑选的商品数据。

代码中的购物车模块，用来存储用户选择的商品数据，也为了订单结算做准备，是相距结算环节最近的一个步骤和功能点，接下来我们来讲一下它的相关功能实现。

## 商品接口实现

商品详情页面是让用户能够看到更多的商品信息，以便他们更好的进行选择和比对。不过获取商品详情信息这个接口并不是一个复杂的逻辑，实现逻辑就是根据 id 查询到商品表中的记录并返回给前端即可，因此我也并没有单独用一篇文章来讲解，将其划分到购物车功能模块实现这篇文章中，主要是因为商品详情页与购物车功能模块有一定的关联，新蜂商城目前只在商品详情页中设计了“加入购物车”的按钮，如果想在购物车中添加数据则必须通过商品详情页面。

所以我选择在这一小节来讲解商品详情接口，一是这个功能不复杂，二是这个功能与购物车功能模块有关联。

### 商品详情接口

详情页通常是通过点击首页或者搜索列表页中的单个商品卡片中的链接跳转而来的，商品详情接口的路径我们定义为 /goods/detail/{goodsId}，我们在 GoodsController 中新增 goodsDetail() 方法来处理这个请求，新增如下代码：

**@GetMapping("/goods/detail/{goodsId}")**

**@ApiOperation(value = "商品详情接口", notes = "传参为商品id")**

**public** Result<NewBeeMallGoodsDetailVO> **goodsDetail**(**@ApiParam(value = "商品id")** **@PathVariable("goodsId")** Long goodsId, **@TokenToMallUser** MallUser loginMallUser) {

logger.info("goods detail api,goodsId={},userId={}", goodsId, loginMallUser.getUserId());

**if** (goodsId < 1) {

**return** ResultGenerator.genFailResult("参数异常");

}

**NewBeeMallGoods** goods = newBeeMallGoodsService.getNewBeeMallGoodsById(goodsId);

**if** (goods == null) {

**return** ResultGenerator.genFailResult("参数异常");

}

**if** (Constants.SELL\_STATUS\_UP != goods.getGoodsSellStatus()) {

NewBeeMallException.fail(ServiceResultEnum.GOODS\_PUT\_DOWN.getResult());

}

**NewBeeMallGoodsDetailVO** goodsDetailVO = **new** **NewBeeMallGoodsDetailVO**();

BeanUtil.copyProperties(goods, goodsDetailVO);

goodsDetailVO.setGoodsCarouselList(goods.getGoodsCarousel().split(","));

**return** ResultGenerator.genSuccessResult(goodsDetailVO);

}

goodsId 这个参数就是商品主键 id，通过 @PathVariable 注解读取路径中的这个字段值，并根据这个值去调用 NewBeeMallGoodsService 中的 getNewBeeMallGoodsById() 方法获取到 NewBeeMallGoods 对象，getNewBeeMallGoodsById() 方法的实现方式就是根据主键 id 去查询数据库中的商品表并返回商品实体数据，最后将查询到的商品详情数据转换为 NewBeeMallGoodsDetailVO 对象并返回给前端。商品表中并不是所有的字段都需要返回，所以这里做了一次对象的转换，NewBeeMallGoodsDetailVO 字段如下：

*/\*\**

*\* 商品详情页VO*

*\*/***@Datapublic** **class** **NewBeeMallGoodsDetailVO** **implements** **Serializable** {

**@ApiModelProperty("商品id")**

**private** Long goodsId;

**@ApiModelProperty("商品名称")**

**private** String goodsName;

**@ApiModelProperty("商品简介")**

**private** String goodsIntro;

**@ApiModelProperty("商品图片地址")**

**private** String goodsCoverImg;

**@ApiModelProperty("商品价格")**

**private** Integer sellingPrice;

**@ApiModelProperty("商品标签")**

**private** String tag;

**@ApiModelProperty("商品图片")**

**private** String[] goodsCarouselList;

**@ApiModelProperty("商品原价")**

**private** Integer originalPrice;

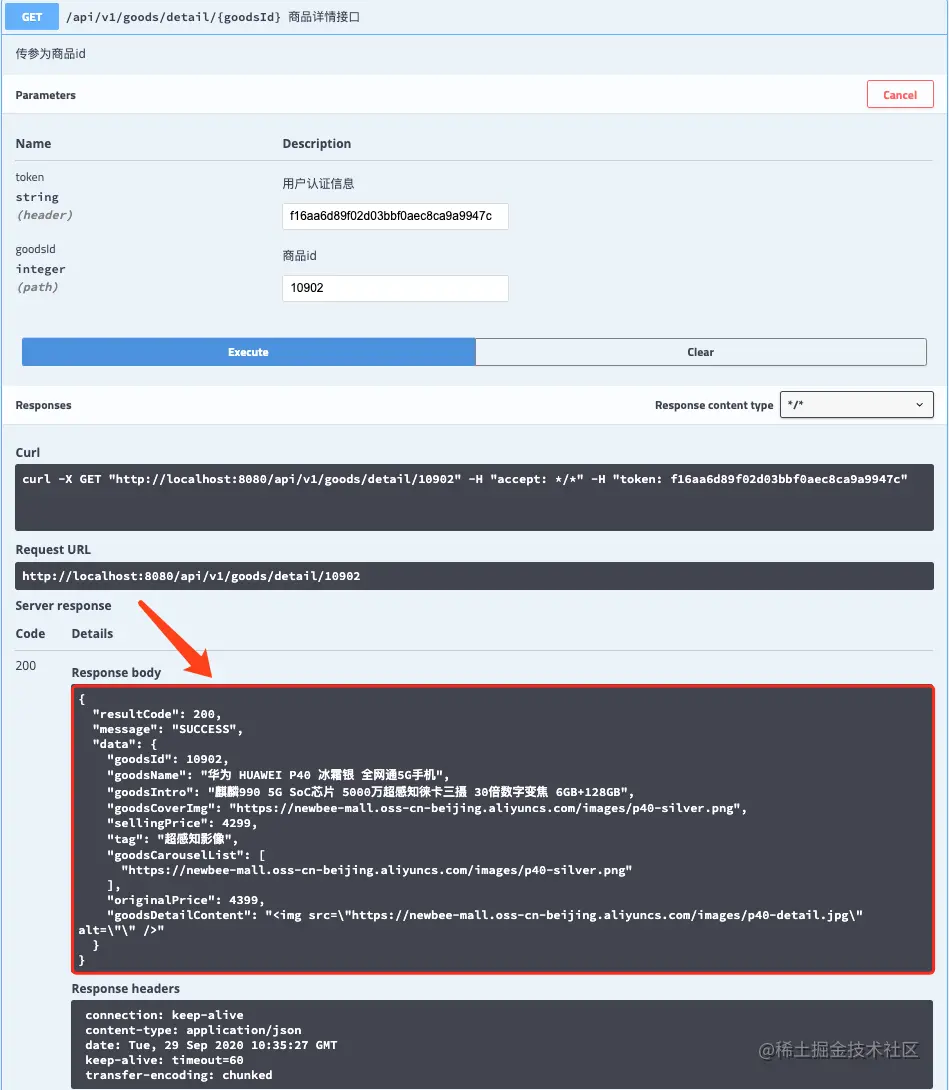
**@ApiModelProperty("商品详情字段")**

**private** String goodsDetailContent;

}

#### 商品详情接口测试

重启项目，打开 swagger-ui 页面，点开“商品详情接口”，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 值并在 token 输入框中填入 token 值，然后输入商品 id，之后点击”Execute“按钮，结果如下：



当前端通过商品详情接口获取到这些数据后，就可以渲染到页面上了，这部分内容会在后续的 Vue 开发实践中介绍。

接下来，我来讲解一下购物车相关的几个接口实现。

## 购物车表结构设计

商城系统的购物车功能模块，用到的购物项表结构设计如下，主要字段为：

* user\_id：用户的 id，我们根据这个字段来确定是哪个用户购物车中的数据，
* goods\_id：关联的商品id，我们根据这个字段来查询对应的商品信息并显示到页面上
* goods\_count：购物车中某商品的数量
* create\_time：添加到购物车中的时间

**DROP** **TABLE** IF **EXISTS** `tb\_newbee\_mall\_shopping\_cart\_item`;

**CREATE** **TABLE** `tb\_newbee\_mall\_shopping\_cart\_item` (

`cart\_item\_id` **bigint**(20) **NOT** **NULL** AUTO\_INCREMENT COMMENT '购物项主键id',

`user\_id` **bigint**(20) **NOT** **NULL** COMMENT '用户主键id',

`goods\_id` **bigint**(20) **NOT** **NULL** **DEFAULT** '0' COMMENT '关联商品id',

`goods\_count` **int**(11) **NOT** **NULL** **DEFAULT** 1 COMMENT '数量(最大为5)',

`is\_deleted` tinyint(4) **NOT** **NULL** **DEFAULT** '0' COMMENT '删除标识字段(0-未删除 1-已删除)',

`create\_time` datetime **NOT** **NULL** **DEFAULT** CURRENT\_TIMESTAMP COMMENT '创建时间',

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

**PRIMARY** KEY (`cart\_item\_id`)

) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;

每个字段对应的含义都在上面的 SQL 中有介绍，大家可以对照 SQL 进行理解，正确的把建表 SQL 导入到数据库中即可，接下来我们进行编码工作。

## 将商品放入购物车

### 后端接口逻辑实现

该接口负责接收前端的 POST 请求并处理其中的参数，接收的参数为 goodsId 字段和 goodsCount 字段，在这个方法里我使用 @RequestBody 注解将其转换为 SaveCartItemParam 对象参数。

* 控制层代码（注：完整代码位于 ****ltd.newbee.mall.api.NewBeeMallShoppingCartAPI.java****）：

接口的映射地址为 /shop-cart，请求方法为 POST，代码如下：

**@PostMapping("/shop-cart")**

**@ApiOperation(value = "添加商品到购物车接口", notes = "传参为商品id、数量")**

**public** Result **saveNewBeeMallShoppingCartItem**(**@RequestBody** SaveCartItemParam saveCartItemParam,

**@TokenToMallUser** MallUser loginMallUser) {

**String** saveResult = newBeeMallShoppingCartService.saveNewBeeMallCartItem(saveCartItemParam, loginMallUser.getUserId());

*//添加成功*

**if** (ServiceResultEnum.SUCCESS.getResult().equals(saveResult)) {

**return** ResultGenerator.genSuccessResult();

}

*//添加失败*

**return** ResultGenerator.genFailResult(saveResult);

}

* 业务层代码（注：完整代码位于 ****ltd.newbee.mall.service.impl.NewBeeMallShoppingCartServiceImpl.java****）：

*/\*\**

*\* 保存商品至购物车中*

*\**

*\*/*

**public** String **saveNewBeeMallCartItem**(SaveCartItemParam saveCartItemParam, Long userId) {

**NewBeeMallShoppingCartItem** temp = newBeeMallShoppingCartItemMapper.selectByUserIdAndGoodsId(userId, saveCartItemParam.getGoodsId());

**if** (temp != null) {

*//已存在则修改该记录*

NewBeeMallException.fail(ServiceResultEnum.SHOPPING\_CART\_ITEM\_EXIST\_ERROR.getResult());

}

**NewBeeMallGoods** newBeeMallGoods = newBeeMallGoodsMapper.selectByPrimaryKey(saveCartItemParam.getGoodsId());

*//商品为空*

**if** (newBeeMallGoods == null) {

**return** ServiceResultEnum.GOODS\_NOT\_EXIST.getResult();

}

**int** totalItem = newBeeMallShoppingCartItemMapper.selectCountByUserId(userId);

*//超出单个商品的最大数量*

**if** (saveCartItemParam.getGoodsCount() < 1) {

**return** ServiceResultEnum.SHOPPING\_CART\_ITEM\_NUMBER\_ERROR.getResult();

}

*//超出单个商品的最大数量*

**if** (saveCartItemParam.getGoodsCount() > Constants.SHOPPING\_CART\_ITEM\_LIMIT\_NUMBER) {

**return** ServiceResultEnum.SHOPPING\_CART\_ITEM\_LIMIT\_NUMBER\_ERROR.getResult();

}

*//超出最大数量*

**if** (totalItem > Constants.SHOPPING\_CART\_ITEM\_TOTAL\_NUMBER) {

**return** ServiceResultEnum.SHOPPING\_CART\_ITEM\_TOTAL\_NUMBER\_ERROR.getResult();

}

**NewBeeMallShoppingCartItem** newBeeMallShoppingCartItem = **new** **NewBeeMallShoppingCartItem**();

BeanUtil.copyProperties(saveCartItemParam, newBeeMallShoppingCartItem);

newBeeMallShoppingCartItem.setUserId(userId);

*//保存记录*

**if** (newBeeMallShoppingCartItemMapper.insertSelective(newBeeMallShoppingCartItem) > 0) {

**return** ServiceResultEnum.SUCCESS.getResult();

}

**return** ServiceResultEnum.DB\_ERROR.getResult();

}

首先对参数进行校验，校验步骤如下：

* 根据用户信息和商品信息去查询购物项表中是否已存在相同的记录，如果存在则进行修改操作，不存在则进行后续操作
* 判断商品数据是否正确
* 判断用户的购物车中的商品数量是否已超出最大限制

校验通过后再进行新增操作，将该记录保存到数据库中，以上操作中都需要调用 SQL 语句来完成。

* NewBeeMallShoppingCartItemMapper.xml （注：完整代码位于 ****resources/mapper/NewBeeMallShoppingCartItemMapper.xml****）：

SQL 语句在 NewBeeMallShoppingCartItemMapper.xml 文件中，包括查询语句和插入语句，插入是标准的 insert 语句，将传参解析后拼成 SQL 语句执行，由于占用篇幅这里就不贴了。

<select id="selectByUserIdAndGoodsId" resultMap="BaseResultMap">

select

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

from tb\_newbee\_mall\_shopping\_cart\_item

where user\_id = #{newBeeMallUserId,jdbcType=BIGINT} and goods\_id=#{goodsId,jdbcType=BIGINT} and is\_deleted = 0

limit 1

</select>

<select id="selectCountByUserId" resultType="int">

select

count(\*)

from tb\_newbee\_mall\_shopping\_cart\_item

where user\_id = #{newBeeMallUserId,jdbcType=BIGINT} and is\_deleted = 0

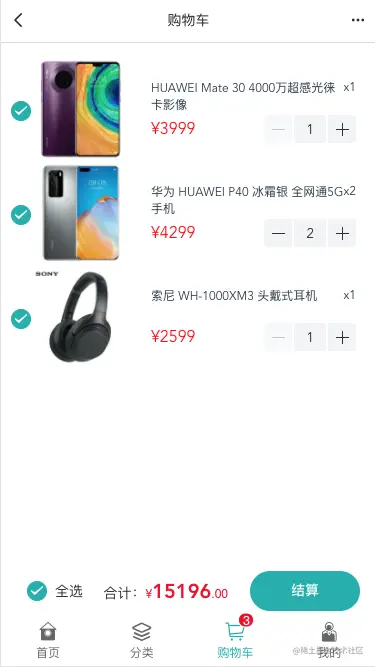
</select>

以上为两个查询语句，在插入购物项数据之前先校验数据是否正常。

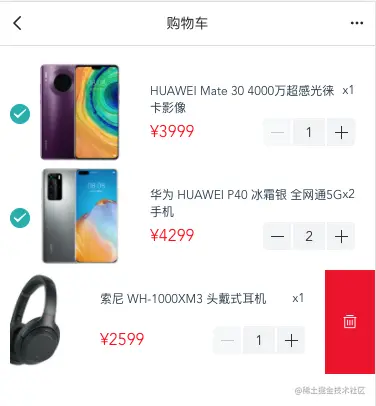
## 购物车列表接口

### 数据格式定义

如下图所示即为新蜂商城购物车页面中需要渲染的内容，购物车中的商品项列表肯定是一个 List 对象，因此后台返回数据时需要一个购物项列表对象，同时还有一些总结性的字段，这些字段以及列表中单项对象中的字段我们可以通过下图中的内容进行确认：



通过上方这张购物车页面的截图，我们看到了购物项列表数据和底部的小计数据，列表数据中包括商品标题字段、商品预览图字段、商品价格字段，这些字段我们可以通过购物项表中的 goods\_id 来查询，还有商品数量字段。



在单个商品项向左滑动后还会出现一个删除按钮，这就表示我们还需要把购物项 id 字段也返回给前端，分析后，需要返回数据的格式就得出来了，购物项 VO 对象编码如下：

*/\*\**

*\* 购物车页面购物项VO*

*\*/***@Datapublic** **class** **NewBeeMallShoppingCartItemVO** **implements** **Serializable** {

**@ApiModelProperty("购物项id")**

**private** Long cartItemId;

**@ApiModelProperty("商品id")**

**private** Long goodsId;

**@ApiModelProperty("商品数量")**

**private** Integer goodsCount;

**@ApiModelProperty("商品名称")**

**private** String goodsName;

**@ApiModelProperty("商品图片")**

**private** String goodsCoverImg;

**@ApiModelProperty("商品价格")**

**private** Integer sellingPrice;

}

### 数据获取

接下来是数据查询的功能实现，上述购物车列表中的字段，可以通过分别查询 tb\_newbee\_mall\_shopping\_cart\_item 购物项表和 tb\_newbee\_mall\_goods\_info 商品表来获取，实现逻辑如下：

#### 查询逻辑

定义 service 方法，业务层代码如下（注：完整代码位于 ltd.newbee.mall.service.impl.NewBeeMallShoppingCartServiceImpl.java）：

*/\*\**

*\* 根据userId获取购物车列表数据*

*\*/*

**public** List<NewBeeMallShoppingCartItemVO> **getMyShoppingCartItems**(Long newBeeMallUserId) {

List<NewBeeMallShoppingCartItemVO> newBeeMallShoppingCartItemVOS = **new** **ArrayList**<>();

List<NewBeeMallShoppingCartItem> newBeeMallShoppingCartItems = newBeeMallShoppingCartItemMapper.selectByUserId(newBeeMallUserId, Constants.SHOPPING\_CART\_ITEM\_TOTAL\_NUMBER);

**return** getNewBeeMallShoppingCartItemVOS(newBeeMallShoppingCartItemVOS, newBeeMallShoppingCartItems);

}

*/\*\**

*\* 数据转换*

*\**

*\* @param newBeeMallShoppingCartItemVOS*

*\* @param newBeeMallShoppingCartItems*

*\* @return*

*\*/*

**private** List<NewBeeMallShoppingCartItemVO> **getNewBeeMallShoppingCartItemVOS**(List<NewBeeMallShoppingCartItemVO> newBeeMallShoppingCartItemVOS, List<NewBeeMallShoppingCartItem> newBeeMallShoppingCartItems) {

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

*//查询商品信息并做数据转换*

List<Long> newBeeMallGoodsIds = newBeeMallShoppingCartItems.stream().map(NewBeeMallShoppingCartItem::getGoodsId).collect(Collectors.toList());

List<NewBeeMallGoods> newBeeMallGoods = newBeeMallGoodsMapper.selectByPrimaryKeys(newBeeMallGoodsIds);

Map<Long, NewBeeMallGoods> newBeeMallGoodsMap = **new** **HashMap**<>();

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

newBeeMallGoodsMap = newBeeMallGoods.stream().collect(Collectors.toMap(NewBeeMallGoods::getGoodsId, Function.identity(), (entity1, entity2) -> entity1));

}

**for** (NewBeeMallShoppingCartItem newBeeMallShoppingCartItem : newBeeMallShoppingCartItems) {

**NewBeeMallShoppingCartItemVO** newBeeMallShoppingCartItemVO = **new** **NewBeeMallShoppingCartItemVO**();

BeanUtil.copyProperties(newBeeMallShoppingCartItem, newBeeMallShoppingCartItemVO);

**if** (newBeeMallGoodsMap.containsKey(newBeeMallShoppingCartItem.getGoodsId())) {

**NewBeeMallGoods** newBeeMallGoodsTemp = newBeeMallGoodsMap.get(newBeeMallShoppingCartItem.getGoodsId());

newBeeMallShoppingCartItemVO.setGoodsCoverImg(newBeeMallGoodsTemp.getGoodsCoverImg());

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

*// 字符串过长导致文字超出的问题*

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

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

}

newBeeMallShoppingCartItemVO.setGoodsName(goodsName);

newBeeMallShoppingCartItemVO.setSellingPrice(newBeeMallGoodsTemp.getSellingPrice());

newBeeMallShoppingCartItemVOS.add(newBeeMallShoppingCartItemVO);

}

}

}

**return** newBeeMallShoppingCartItemVOS;

}

我们定义了 getMyShoppingCartItems() 方法并传入 userId 即用户 id 作为参数，之后通过 SQL 查询出当前 userId 下的购物项列表数据，因为购物车页面需要展示商品信息，所以通过购物项表中的 goods\_id 来获取到每个购物项对应的商品信息，再之后是填充数据，将相关字段封装到 NewBeeMallShoppingCartItemVO 对象中，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，最终将封装好的 List 对象返回。

#### 查询 SQL

****根据 userId 查询当前用户在 tb\_newbee\_mall\_shopping\_cart\_item 表中的记录：****

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

<select id="selectByUserId" resultMap="BaseResultMap">

select

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

from tb\_newbee\_mall\_shopping\_cart\_item

where user\_id = #{newBeeMallUserId,jdbcType=BIGINT} and is\_deleted = 0

limit #{number}

</select>

****根据商品 id 列表查询商品记录信息：****

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

<select id="selectByPrimaryKeys" resultMap="BaseResultMap">

select

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

from tb\_newbee\_mall\_goods\_info

where goods\_id in

<foreach item="id" collection="list" open="(" separator="," close=")">

#{id}

</foreach>

</select>

某些朋友喜欢在查询多条数据时，使用 for 循环中调用 SQL 的方式，这种查询方式或者代码习惯不是特别推荐，建议使用这种一次性将数据查询出来的方式，只需要一条 SQL 就可以获取到想要的结果。

### 列表接口

商品详情接口的路径我们定义为 /shop-cart，我们在 NewBeeMallShoppingCartAPI 中新增 cartItemList() 方法来处理这个请求，新增如下代码：

**@GetMapping("/shop-cart")**

**@ApiOperation(value = "购物车列表(网页移动端不分页)", notes = "")**

**public** Result<List<NewBeeMallShoppingCartItemVO>> **cartItemList**(**@TokenToMallUser** MallUser loginMallUser) {

**return** ResultGenerator.genSuccessResult(newBeeMallShoppingCartService.getMyShoppingCartItems(loginMallUser.getUserId()));

}

该接口并没有使用分页逻辑，主要是参考了一些主流购物网站移动端的购物车数据，通常在移动网页端不会做分页形式，而是一次性加载所有的购物车数据，所以这个接口直接返回一个 List 对象，而不是一个 PageResult 对象，****新蜂商城 Vue 版本中使用的是该接口****。

当然，在 APP 端会做下拉或者上拉分页的动作，因此还需要分页接口的，所以我也准备了这部分接口实现：

**@GetMapping("/shop-cart/page")**

**@ApiOperation(value = "购物车列表(每页默认5条)", notes = "传参为页码")**

**public** Result<PageResult<List<NewBeeMallShoppingCartItemVO>>> **cartItemPageList**(Integer pageNumber, **@TokenToMallUser** MallUser loginMallUser) {

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

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

pageNumber = 1;

}

params.put("userId", loginMallUser.getUserId());

params.put("page", pageNumber);

params.put("limit", Constants.SHOPPING\_CART\_PAGE\_LIMIT);

*//封装分页请求参数*

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

**return** ResultGenerator.genSuccessResult(newBeeMallShoppingCartService.getMyShoppingCartItems(pageUtil));

}

与购物车列表接口有一点不同，该接口中包含分页逻辑，且返回对象也是 PageResult 对象，大家可以自行查看源码学习。

## 编辑购物车中的商品

在购物车列表中我们可以编辑需要购买的商品数量，也可以将商品从购物车中删除，接下来我们来实现这两个功能。

### 后端接口逻辑实现

#### 编辑接口

编辑功能主要是修改当前购物项的数量，后端的编辑接口负责接收前端的 PUT 请求并进行处理，接收的参数为 cartItemId 字段和 goodsCount 字段，通过这两个字段就可以确定需要修改哪一条记录以及商品数量值修改为多少，在这个方法里使用 @RequestBody 注解将其转换为 UpdateCartItemParam 对象参数。

* 控制层代码（注：完整代码位于 ****ltd.newbee.mall.api.NewBeeMallShoppingCartAPI.java****）：

接口的映射地址为 /shop-cart，请求方法为 PUT，该 Controller 中的几个方法路径都是 /shop-cart，根据请求方法的不同作为接口的区分方式，POST 方法是新增接口，GET 方法是购物车列表页面显示，PUT 方法是修改接口。

代码如下：

**@PutMapping("/shop-cart")**

**@ApiOperation(value = "修改购物项数据", notes = "传参为购物项id、数量")**

**public** Result **updateNewBeeMallShoppingCartItem**(**@RequestBody** UpdateCartItemParam updateCartItemParam,

**@TokenToMallUser** MallUser loginMallUser) {

**String** updateResult = newBeeMallShoppingCartService.updateNewBeeMallCartItem(updateCartItemParam, loginMallUser.getUserId());

*//修改成功*

**if** (ServiceResultEnum.SUCCESS.getResult().equals(updateResult)) {

**return** ResultGenerator.genSuccessResult();

}

*//修改失败*

**return** ResultGenerator.genFailResult(updateResult);

}

* 业务层代码（注：完整代码位于 ****ltd.newbee.mall.service.impl.NewBeeMallShoppingCartServiceImpl.java****）：

*/\*\**

*\* 修改数量*

*\**

*\*/*

**public** String **updateNewBeeMallCartItem**(UpdateCartItemParam updateCartItemParam, Long userId) {

**NewBeeMallShoppingCartItem** newBeeMallShoppingCartItemUpdate = newBeeMallShoppingCartItemMapper.selectByPrimaryKey(updateCartItemParam.getCartItemId());

**if** (newBeeMallShoppingCartItemUpdate == null) {

**return** ServiceResultEnum.DATA\_NOT\_EXIST.getResult();

}

**if** (!newBeeMallShoppingCartItemUpdate.getUserId().equals(userId)) {

NewBeeMallException.fail(ServiceResultEnum.REQUEST\_FORBIDEN\_ERROR.getResult());

}

*//超出单个商品的最大数量*

**if** (updateCartItemParam.getGoodsCount() > Constants.SHOPPING\_CART\_ITEM\_LIMIT\_NUMBER) {

**return** ServiceResultEnum.SHOPPING\_CART\_ITEM\_LIMIT\_NUMBER\_ERROR.getResult();

}

newBeeMallShoppingCartItemUpdate.setGoodsCount(updateCartItemParam.getGoodsCount());

newBeeMallShoppingCartItemUpdate.setUpdateTime(**new** **Date**());

*//修改记录*

**if** (newBeeMallShoppingCartItemMapper.updateByPrimaryKeySelective(newBeeMallShoppingCartItemUpdate) > 0) {

**return** ServiceResultEnum.SUCCESS.getResult();

}

**return** ServiceResultEnum.DB\_ERROR.getResult();

}

对参数进行校验，校验步骤如下：

* 首先根据前端传参的购物项主键 id 去查询购物项表中是否存在该记录，如果不存在则返回错误信息，存在则进行后续操作
* 判断用户的购物车中的商品数量是否已超出最大限制

校验通过后再进行修改操作，将该购物项记录的数量和修改时间修改掉，以上操作中都需要调用 SQL 语句来完成。

SQL 语句在 NewBeeMallGoodsMapper.xml 文件中，包括查询语句和修改语句，将传参解析后拼成 SQL 语句执行，由于占用篇幅这里就不贴了，大家可以在我提供的源码中看到。

#### 删除接口

如果购物车中一些商品，你不想进行后续的结算购买操作，可以选择将其删除。

后端的编辑接口负责接收前端的 DELETE 请求并进行处理，接收的参数为 cartItemId 字段，之后调用删除方法即可完成删除操作。

接口的映射地址为 /shop-cart/{newBeeMallShoppingCartItemId}，请求方法为 DELETE，代码如下：

**@DeleteMapping("/shop-cart/{newBeeMallShoppingCartItemId}")**

**@ApiOperation(value = "删除购物项", notes = "传参为购物项id")**

**public** Result **updateNewBeeMallShoppingCartItem**(**@PathVariable("newBeeMallShoppingCartItemId")** Long newBeeMallShoppingCartItemId,

**@TokenToMallUser** MallUser loginMallUser) {

**NewBeeMallShoppingCartItem** newBeeMallCartItemById = newBeeMallShoppingCartService.getNewBeeMallCartItemById(newBeeMallShoppingCartItemId);

**if** (!loginMallUser.getUserId().equals(newBeeMallCartItemById.getUserId())) {

**return** ResultGenerator.genFailResult(ServiceResultEnum.REQUEST\_FORBIDEN\_ERROR.getResult());

}

**Boolean** deleteResult = newBeeMallShoppingCartService.deleteById(newBeeMallShoppingCartItemId);

*//删除成功*

**if** (deleteResult) {

**return** ResultGenerator.genSuccessResult();

}

*//删除失败*

**return** ResultGenerator.genFailResult(ServiceResultEnum.OPERATE\_ERROR.getResult());

}

## 总结

内容比较多，希望大家结合源码来实际的动手，并跟着文章提到的功能点分步骤去学习，以免学习时看到文章看着看着就出现烦躁的心理。

文中提到的几个需要注意的知识点，大家一定要自己动手操作一下。另外，这些功能都牵涉到数据的更改，在自己测试的时候也一定要注意数据库中的数据是否正确的添加和修改。