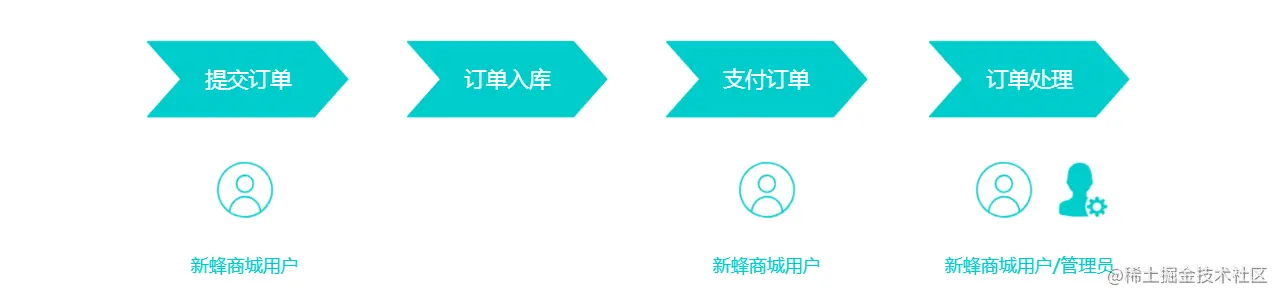
## 新蜂商城订单模块

把心仪的商品加到购物车后，紧接着壕气的用户就要下手了，在确定了需要购买的商品和对应的数量之后，就可以执行提交订单的操作，此时也就由购物车模块切换到了另外一个电商流程中--****订单流程****，之前是商品模块和购物车模块，接下来的两篇文章我们要介绍的就是订单模块，关于订单的生成和后续处理流程，不同公司或者不同商城项目中具体的具体需求与业务场景多多少少会有一些差异，但是从订单的生成阶段到订单完成阶段大体的流程还是类似的，只是会存在一些具体业务中的不同。

新蜂商城中订单的生成到处理结束，主要有以下几个阶段：

1. 提交订单（由新蜂商城用户发起）
2. 订单入库（后台逻辑，用户无感知）
3. 支付订单（由新蜂商城用户发起）
4. 订单流程处理（包括确认订单、取消订单、修改订单信息等操作，新蜂商城用户和管理员都可以对支付成功后的订单进行处理）



这篇文章会讲解订单流程中的前两个步骤：提交订单和订单入库，实际的功能介绍是订单的生成过程，后续两个步骤，支付和订单处理我会在接下来的文章中介绍。

主要知识点如下：

* 新蜂商城订单确认页信息获取接口
* 新蜂商城订单生成逻辑实现和介绍

## 订单确认

### 商城中的订单确认步骤

订单确认步骤是订单生成功能中的一个很重要的功能，基本上我们日常接触到的商城项目基本都有这个环节，在线上商城剁手购物时，肯定会经历“订单确认”这样一个环节，以淘宝商城的订单确认页为例：

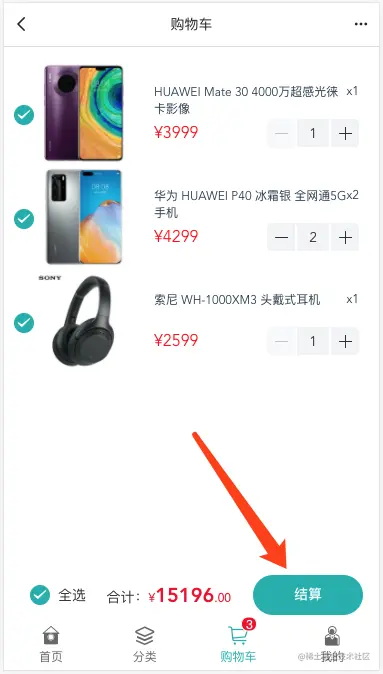


这个页面有刚刚在购物车中选择的商品信息，还有收获地址信息、运费信息、优惠信息等等，在购物车中只有商品信息，在这个页面则是多种信息的集合，只有这些信息齐全了我们才能够生成订单数据，这是一个确认的过程，在该页面将所有信息都确认无误后，就可以进行提交订单的操作，之后就会生成一条订单记录了。

### 新蜂商城的订单确认页面

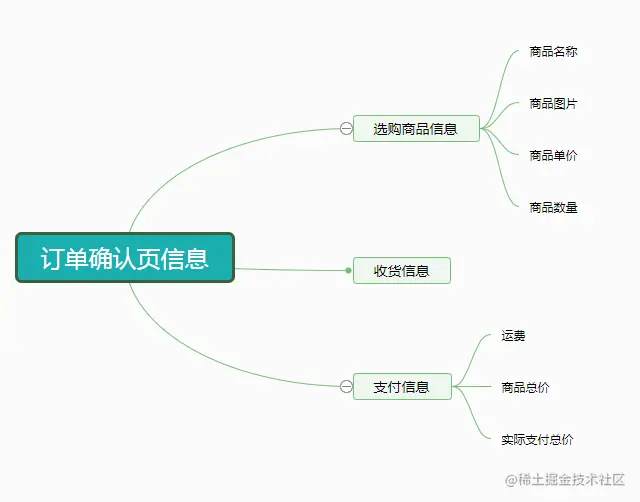
#### 发起请求

我们是通过购物车页面中的结算按钮进入到订单确认页面的，订单提交的发起是在购物车页面，点击提交按钮就能够开始这个步骤。



#### 数据格式

订单确认页需要展示的数据与购物车列表中展示的商品数据类似，不过该页面不仅仅只有商品数据，还有用户数据和支付数据，所以该页面中主要的数据是****商品数据+用户数据+支付数据****，仅仅选择商品是不行的，我们还需要把购买者以及购买者的收货信息给记录下来，其他的内容还有运费金额、优惠金额、实际支付金额等等，新蜂商城的订单确认页信息如下：



## 订单确认页数据获取接口

订单确认页的信息是用户在购物车中选择结算的购物项，用户可能会全选并进行结算，也可能选择部分商品进行结算，因此这个页面中的数据不能直接使用购物车列表接口，而需要新增一个信息获取接口。

购物车列表接口不需要额外传参，只需要将用户数据放在请求中就可以根据 userId 将其购物车中的数据查询出来，而订单确认页面的数据获取接口则不同，我们需要查询出用户选择的结算商品数据，因此需要另外去定义参数。新蜂商城 API 项目的这个接口参数设计为 cartItemIds，即用户选择的购物项 id 数组，返回对象的数据格式与购物车列表中的字段一致，因此继续使用 NewBeeMallShoppingCartItemVO，当然，也可以另外定义。

在 NewBeeMallShoppingCartAPI 类中新增 toSettle() 方法，该方法用于处理 /shop-cart/settle 请求，并将订单确认页所需的数据返回，代码如下：

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

**@ApiOperation(value = "根据购物项id数组查询购物项明细", notes = "确认订单页面使用")**

**public** Result<List<NewBeeMallShoppingCartItemVO>> **toSettle**(Long[] cartItemIds, **@TokenToMallUser** MallUser loginMallUser) {

**if** (cartItemIds.length < 1) {

NewBeeMallException.fail("参数异常");

}

**int** priceTotal = 0;

List<NewBeeMallShoppingCartItemVO> itemsForSettle = newBeeMallShoppingCartService.getCartItemsForSettle(Arrays.asList(cartItemIds), loginMallUser.getUserId());

**if** (CollectionUtils.isEmpty(itemsForSettle)) {

*//无数据则抛出异常*

NewBeeMallException.fail("参数异常");

} **else** {

*//总价*

**for** (NewBeeMallShoppingCartItemVO newBeeMallShoppingCartItemVO : itemsForSettle) {

priceTotal += newBeeMallShoppingCartItemVO.getGoodsCount() \* newBeeMallShoppingCartItemVO.getSellingPrice();

}

**if** (priceTotal < 1) {

NewBeeMallException.fail("价格异常");

}

}

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

}

业务实现代码为：

**public** List<NewBeeMallShoppingCartItemVO> **getCartItemsForSettle**(List<Long> cartItemIds, Long newBeeMallUserId) {

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

**if** (CollectionUtils.isEmpty(cartItemIds)) {

NewBeeMallException.fail("购物项不能为空");

}

List<NewBeeMallShoppingCartItem> newBeeMallShoppingCartItems = newBeeMallShoppingCartItemMapper.selectByUserIdAndCartItemIds(newBeeMallUserId, cartItemIds);

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

NewBeeMallException.fail("购物项不能为空");

}

**if** (newBeeMallShoppingCartItems.size() != cartItemIds.size()) {

NewBeeMallException.fail("参数异常");

}

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

}

实际执行的 SQL 查询语句为：

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

select

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

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

where

cart\_item\_id in

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

#{id}

</foreach>

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

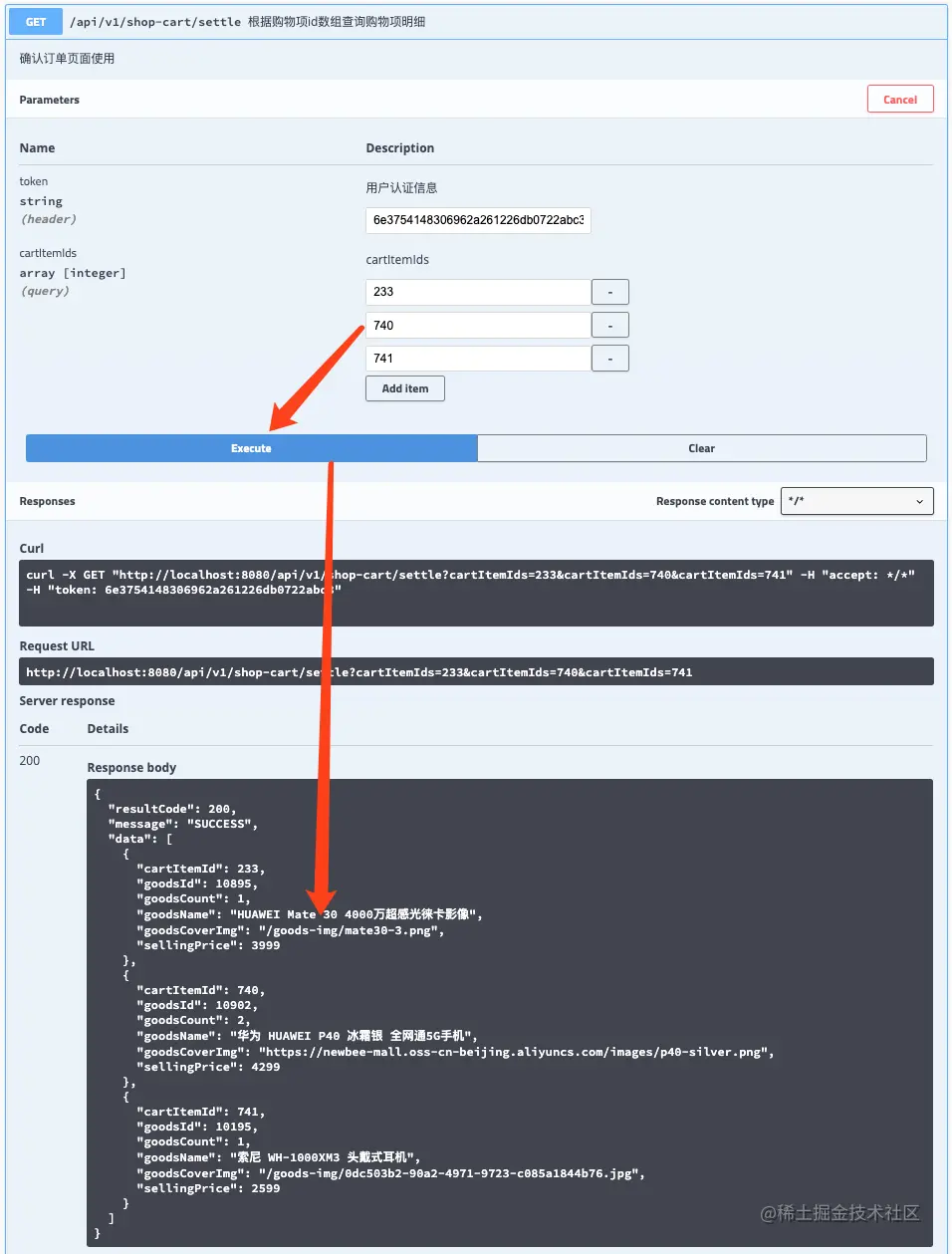
</select>

订单确认页数据获取接口实现逻辑如下：

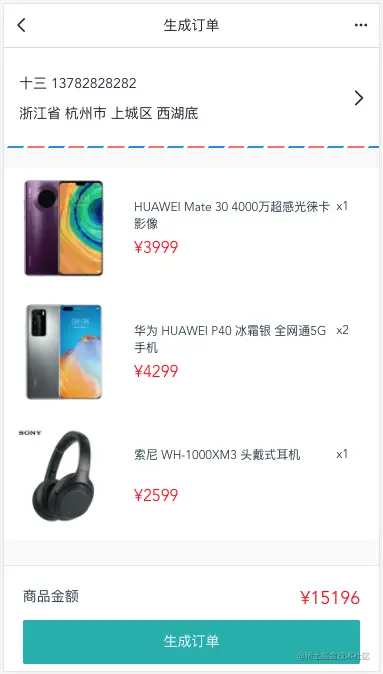
* 首先判断接收的参数，cartItemIds 为空或者未登录则返回错误信息。
* 根据 cartItemIds 和 userId 去查询 tb\_newbee\_mall\_shopping\_cart\_item 表中的数据。
* 如果查询到的购物项数据为空，则表示接收到的 cartItemIds 为非法参数，需要返回错误信息。
* 最后，封装数据并通过接口返回给前端。

### 订单确认页数据获取接口测试

重启项目，打开 swagger-ui 页面，点开“根据购物项id数组查询购物项明细”，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，然后输入购物项 id，cartItemIds 是一个数组，如果有多个你可以点击”Add item“继续添加，如果只有一个则只填写一个购物项 id 也可以。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，再依次渲染到页面中，页面效果如下：



本文只讲解后端接口实现，前端页面这部分内容会在后续的 Vue 开发实践中介绍。

## 表结构设计

### 订单主表和订单项关联表设计

新蜂商城系统的订单功能模块，主要牵涉到数据库中的三张表，因为一个订单中可能会存在多个商品，还有用户的收货地址数据。所以除了订单主表 tb\_newbee\_mall\_order 外，还有一个订单项关联表 tb\_newbee\_mall\_order\_item 和一个订单地址关联表，订单主表中存储关于订单的相关信息，而订单项表中主要存储关联的商品字段，。

订单主表 tb\_newbee\_mall\_order 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下的订单
* order\_no：订单号，订单号是用来唯一的标识订单和后续查询订单时用的，这是每个电商系统都会有的设计
* pay\_status/pay\_type/pay\_time：支付信息字段，包括支付状态、支付方式、支付时间
* order\_status：订单状态
* create\_time：添加到购物车中的时间

**CREATE** **TABLE** `tb\_newbee\_mall\_order` (

`order\_id` **bigint**(20) **NOT** **NULL** AUTO\_INCREMENT COMMENT '订单表主键id',

`order\_no` **varchar**(20) **NOT** **NULL** **DEFAULT** '' COMMENT '订单号',

`user\_id` **bigint**(20) **NOT** **NULL** **DEFAULT** '0' COMMENT '用户主键id',

`total\_price` **int**(11) **NOT** **NULL** **DEFAULT** '1' COMMENT '订单总价',

`pay\_status` tinyint(4) **NOT** **NULL** **DEFAULT** '0' COMMENT '支付状态:0.未支付,1.支付成功,-1:支付失败',

`pay\_type` tinyint(4) **NOT** **NULL** **DEFAULT** '0' COMMENT '0.无 1.支付宝支付 2.微信支付',

`pay\_time` datetime **DEFAULT** **NULL** COMMENT '支付时间',

`order\_status` tinyint(4) **NOT** **NULL** **DEFAULT** '0' COMMENT '订单状态:0.待支付 1.已支付 2.配货完成 3:出库成功 4.交易成功 -1.手动关闭 -2.超时关闭 -3.商家关闭',

`extra\_info` **varchar**(100) **NOT** **NULL** **DEFAULT** '' COMMENT '订单body',

`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 (`order\_id`)

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

订单项表 tb\_newbee\_mall\_order\_item 表结构设计如下，主要字段为：order\_id：关联的订单主键 id，标识该订单项是哪个订单中的数据goods\_id/goods\_name/goods\_cover\_img/selling\_price/goods\_count：订单中的商品信息，且主要字段都做了保存，而且是下单当时的商品数据，因为商品数据也是可以更改的，如果数据发生了改变，只关联商品主键 goods\_id 是不够的，更改后再去查看订单详情就不是下单时的商品数据了，因此对这几个字段都做了保存处理，作为订单快照，****记录下单当时的商品信息****

* create\_time：添加到购物车中的时间

**DROP** **TABLE** IF **EXISTS** `tb\_newbee\_mall\_order\_item`;**CREATE** **TABLE** `tb\_newbee\_mall\_order\_item` (

`order\_item\_id` **bigint**(20) **NOT** **NULL** AUTO\_INCREMENT COMMENT '订单关联购物项主键id',

`order\_id` **bigint**(20) **NOT** **NULL** **DEFAULT** 0 COMMENT '订单主键id',

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

`goods\_name` **varchar**(200) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci **NOT** **NULL** **DEFAULT** '' COMMENT '下单时商品的名称(订单快照)',

`goods\_cover\_img` **varchar**(200) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci **NOT** **NULL** **DEFAULT** '' COMMENT '下单时商品的主图(订单快照)',

`selling\_price` **int**(11) **NOT** **NULL** **DEFAULT** 1 COMMENT '下单时商品的价格(订单快照)',

`goods\_count` **int**(11) **NOT** **NULL** **DEFAULT** 1 COMMENT '数量(订单快照)',

`create\_time` datetime(0) **NOT** **NULL** **DEFAULT** CURRENT\_TIMESTAMP COMMENT '创建时间',

**PRIMARY** KEY (`order\_item\_id`) **USING** BTREE

) ENGINE = InnoDB AUTO\_INCREMENT = 1 **CHARACTER** **SET** = utf8 **COLLATE** = utf8\_general\_ci ROW\_FORMAT = **Dynamic**;

订单地址关联表 tb\_newbee\_mall\_order\_address 表结构设计如下，主要字段为：

order\_id：关联的订单主键 id，标识该地址是哪个订单中的收货地址数据

user\_name/user\_phone/province\_name/city\_name/region\_name/detail\_address：收件地址信息，该表与订单表是一对一的关系，主键也是 order\_id，这些字段完全可以放在订单表中，由于字段多，为了防止订单表字段太多，所以将地址信息单独做了一张关联表。

有些商城的订单设计，关于收货地址只会关联一个地址 id 字段，这样做是不合理的，因为该地址 id 关联的地址表中的记录是可以修改和删除的，也就是说如果做了这些操作后，订单的收件信息就不是下单时的数据了，可能会出现乌龙事件，因此需要冗余这些字段到订单表中，记录下单当时的收件信息。

**CREATE** **TABLE** `tb\_newbee\_mall\_order\_address` (

`order\_id` **bigint**(20) **NOT** **NULL**,

`user\_name` **varchar**(30) **NOT** **NULL** **DEFAULT** '' COMMENT '收货人姓名',

`user\_phone` **varchar**(11) **NOT** **NULL** **DEFAULT** '' COMMENT '收货人手机号',

`province\_name` **varchar**(32) **NOT** **NULL** **DEFAULT** '' COMMENT '省',

`city\_name` **varchar**(32) **NOT** **NULL** **DEFAULT** '' COMMENT '城',

`region\_name` **varchar**(32) **NOT** **NULL** **DEFAULT** '' COMMENT '区',

`detail\_address` **varchar**(64) **NOT** **NULL** **DEFAULT** '' COMMENT '收件详细地址(街道/楼宇/单元)',

**PRIMARY** KEY (`order\_id`)

) ENGINE=InnoDB **DEFAULT** CHARSET=utf8 COMMENT='订单收货地址关联表';

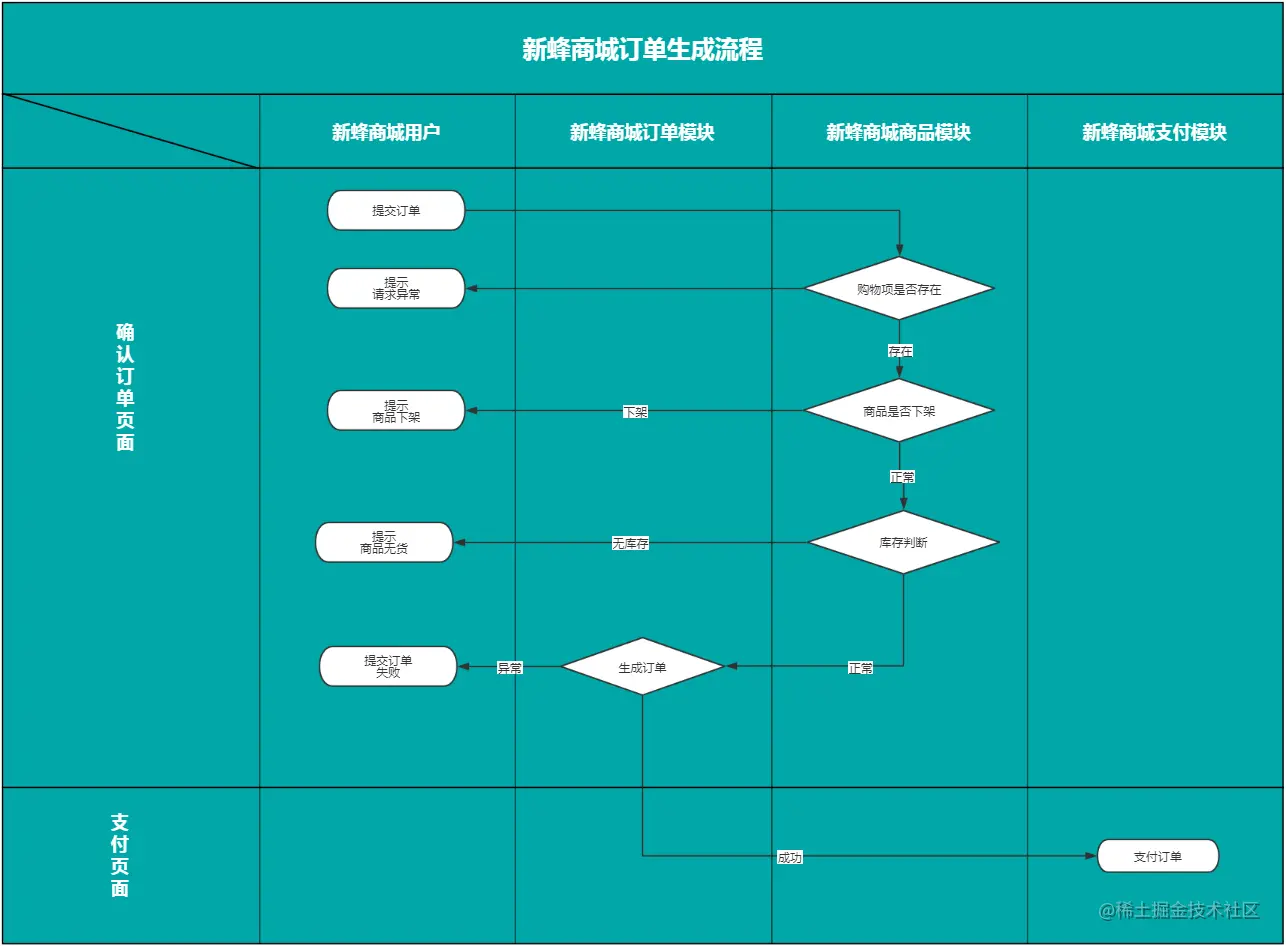
每个字段对应的含义都在上面的 SQL 中有介绍，大家可以对照 SQL 进行理解，正确的把建表 SQL 导入到数据库中即可，关于两张表中的快照字段，包括收件信息字段和商品信息字段，大家可以做一下思考，也可以参考淘宝商城的订单快照来理解，这些信息都是可以更改的，因此不能只关联一个主键 id，一旦更改，订单信息也随之更改，不再是下单时的数据了。

## 订单生成功能实现

### 新蜂商城订单生成流程

****在订单确认页面将以上这些信息都处理完毕后，紧接着就是生成订单的环节，此时的用户点击"提交订单"按钮，我们的系统中也就对应的生成了一笔订单数据，接下来我们来讲解订单生成的功能实现。****

点击提交订单按钮后，此时后台会进行一系列的操作，数据查询、数据判断、数据整合等等，订单生成流程图如下：



大家可以仔细看一下这张流程图，并结合流程图去更好的理解实现代码，接下来我们再来讲解一下具体的订单生成代码实现。

### 新蜂商城订单生成编码实现

#### 订单生成接口

首先来分析一下参数，与订单确认页的数据相同，在订单生成接口中，我们依然需要把用户选择的购物项 id 数组传给后端接口，同时需要将用户选择的地址数据一并传给后端接口。这里我定义了一个 SaveOrderParam 对象用于接收参数，字段如下：

*/\*\**

*\* 保存订单param*

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

**@ApiModelProperty("订单项id数组")**

**private** Long[] cartItemIds;

**@ApiModelProperty("地址id")**

**private** Long addressId;

}

之后在 api 包中新建 NewBeeMallOrderAPI 类用于处理订单模块的相关请求，之后新增 saveOrder() 方法用于处理订单生成的请求，代码如下：

**package** ltd.newbee.mall.api;

**import** io.swagger.annotations.Api;**import** io.swagger.annotations.ApiOperation;**import** io.swagger.annotations.ApiParam;**import** ltd.newbee.mall.api.param.SaveOrderParam;**import** ltd.newbee.mall.api.vo.NewBeeMallShoppingCartItemVO;**import** ltd.newbee.mall.common.NewBeeMallException;**import** ltd.newbee.mall.common.ServiceResultEnum;**import** ltd.newbee.mall.config.annotation.TokenToMallUser;**import** ltd.newbee.mall.entity.MallUser;**import** ltd.newbee.mall.entity.MallUserAddress;**import** ltd.newbee.mall.service.NewBeeMallOrderService;**import** ltd.newbee.mall.service.NewBeeMallShoppingCartService;**import** ltd.newbee.mall.service.NewBeeMallUserAddressService;**import** ltd.newbee.mall.util.Result;**import** ltd.newbee.mall.util.ResultGenerator;**import** org.springframework.util.CollectionUtils;**import** org.springframework.web.bind.annotation.PostMapping;**import** org.springframework.web.bind.annotation.RequestBody;**import** org.springframework.web.bind.annotation.RequestMapping;**import** org.springframework.web.bind.annotation.RestController;

**import** javax.annotation.Resource;**import** java.util.Arrays;**import** java.util.List;

**@RestController@Api(value = "v1", tags = "新蜂商城订单操作相关接口")@RequestMapping("/api/v1")public** **class** **NewBeeMallOrderAPI** {

**@Resource**

**private** NewBeeMallShoppingCartService newBeeMallShoppingCartService;

**@Resource**

**private** NewBeeMallOrderService newBeeMallOrderService;

**@Resource**

**private** NewBeeMallUserAddressService newBeeMallUserAddressService;

**@PostMapping("/saveOrder")**

**@ApiOperation(value = "生成订单接口", notes = "传参为地址id和待结算的购物项id数组")**

**public** Result<String> **saveOrder**(**@ApiParam(value = "订单参数")** **@RequestBody** SaveOrderParam saveOrderParam, **@TokenToMallUser** MallUser loginMallUser) {

**int** priceTotal = 0;

**if** (saveOrderParam == null || saveOrderParam.getCartItemIds() == null || saveOrderParam.getAddressId() == null) {

NewBeeMallException.fail(ServiceResultEnum.PARAM\_ERROR.getResult());

}

**if** (saveOrderParam.getCartItemIds().length < 1) {

NewBeeMallException.fail(ServiceResultEnum.PARAM\_ERROR.getResult());

}

List<NewBeeMallShoppingCartItemVO> itemsForSave = newBeeMallShoppingCartService.getCartItemsForSettle(Arrays.asList(saveOrderParam.getCartItemIds()), loginMallUser.getUserId());

**if** (CollectionUtils.isEmpty(itemsForSave)) {

*//无数据*

NewBeeMallException.fail("参数异常");

} **else** {

*//总价*

**for** (NewBeeMallShoppingCartItemVO newBeeMallShoppingCartItemVO : itemsForSave) {

priceTotal += newBeeMallShoppingCartItemVO.getGoodsCount() \* newBeeMallShoppingCartItemVO.getSellingPrice();

}

**if** (priceTotal < 1) {

NewBeeMallException.fail("价格异常");

}

**MallUserAddress** address = newBeeMallUserAddressService.getMallUserAddressById(saveOrderParam.getAddressId());

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

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

}

*//保存订单并返回订单号*

**String** saveOrderResult = newBeeMallOrderService.saveOrder(loginMallUser, address, itemsForSave);

**Result** result = ResultGenerator.genSuccessResult();

result.setData(saveOrderResult);

**return** result;

}

**return** ResultGenerator.genFailResult("生成订单失败");

}

}

该方法处理的映射地址为 /saveOrder，请求方法为 GET，过程如下：

参数非空判断，如果有空数据则直接返回错误信息。

验证购物车中是否有数据，有则继续后续流程，无则返回异常信息。

验证收货地址信息，有则继续后续流程，无则返回异常信息。

将购物项数据和用户信息作为参数传给业务层的 saveOrder() 方法进行订单生成的业务逻辑操作。

如果订单生成成功，业务层的 saveOrder() 方法会返回订单号，拿到订单号之后通过接口返回给前端即可。

#### 订单生成逻辑实现

接下来是在 service 包中新建订单模块的业务实现类，并实现订单生成的业务逻辑，代码如下：

**public** String **saveOrder**(MallUser loginMallUser, MallUserAddress address, List<NewBeeMallShoppingCartItemVO> myShoppingCartItems) {

List<Long> itemIdList = myShoppingCartItems.stream().map(NewBeeMallShoppingCartItemVO::getCartItemId).collect(Collectors.toList());

List<Long> goodsIds = myShoppingCartItems.stream().map(NewBeeMallShoppingCartItemVO::getGoodsId).collect(Collectors.toList());

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

*//检查是否包含已下架商品*

List<NewBeeMallGoods> goodsListNotSelling = newBeeMallGoods.stream()

.filter(newBeeMallGoodsTemp -> newBeeMallGoodsTemp.getGoodsSellStatus() != Constants.SELL\_STATUS\_UP)

.collect(Collectors.toList());

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

*//goodsListNotSelling 对象非空则表示有下架商品*

NewBeeMallException.fail(goodsListNotSelling.get(0).getGoodsName() + "已下架，无法生成订单");

}

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

*//判断商品库存*

**for** (NewBeeMallShoppingCartItemVO shoppingCartItemVO : myShoppingCartItems) {

*//查出的商品中不存在购物车中的这条关联商品数据，直接返回错误提醒*

**if** (!newBeeMallGoodsMap.containsKey(shoppingCartItemVO.getGoodsId())) {

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

}

*//存在数量大于库存的情况，直接返回错误提醒*

**if** (shoppingCartItemVO.getGoodsCount() > newBeeMallGoodsMap.get(shoppingCartItemVO.getGoodsId()).getStockNum()) {

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

}

}

*//删除购物项*

**if** (!CollectionUtils.isEmpty(itemIdList) && !CollectionUtils.isEmpty(goodsIds) && !CollectionUtils.isEmpty(newBeeMallGoods)) {

**if** (newBeeMallShoppingCartItemMapper.deleteBatch(itemIdList) > 0) {

List<StockNumDTO> stockNumDTOS = BeanUtil.copyList(myShoppingCartItems, StockNumDTO.class);

**int** updateStockNumResult = newBeeMallGoodsMapper.updateStockNum(stockNumDTOS);

**if** (updateStockNumResult < 1) {

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

}

*//生成订单号*

**String** orderNo = NumberUtil.genOrderNo();

**int** priceTotal = 0;

*//保存订单*

**NewBeeMallOrder** newBeeMallOrder = **new** **NewBeeMallOrder**();

newBeeMallOrder.setOrderNo(orderNo);

newBeeMallOrder.setUserId(loginMallUser.getUserId());

*//总价*

**for** (NewBeeMallShoppingCartItemVO newBeeMallShoppingCartItemVO : myShoppingCartItems) {

priceTotal += newBeeMallShoppingCartItemVO.getGoodsCount() \* newBeeMallShoppingCartItemVO.getSellingPrice();

}

**if** (priceTotal < 1) {

NewBeeMallException.fail(ServiceResultEnum.ORDER\_PRICE\_ERROR.getResult());

}

newBeeMallOrder.setTotalPrice(priceTotal);

**String** extraInfo = "";

newBeeMallOrder.setExtraInfo(extraInfo);

*//生成订单项并保存订单项纪录*

**if** (newBeeMallOrderMapper.insertSelective(newBeeMallOrder) > 0) {

*//生成订单收货地址快照，并保存至数据库*

**NewBeeMallOrderAddress** newBeeMallOrderAddress = **new** **NewBeeMallOrderAddress**();

BeanUtil.copyProperties(address, newBeeMallOrderAddress);

newBeeMallOrderAddress.setOrderId(newBeeMallOrder.getOrderId());

*//生成所有的订单项快照，并保存至数据库*

List<NewBeeMallOrderItem> newBeeMallOrderItems = **new** **ArrayList**<>();

**for** (NewBeeMallShoppingCartItemVO newBeeMallShoppingCartItemVO : myShoppingCartItems) {

**NewBeeMallOrderItem** newBeeMallOrderItem = **new** **NewBeeMallOrderItem**();

*//使用BeanUtil工具类将newBeeMallShoppingCartItemVO中的属性复制到newBeeMallOrderItem对象中*

BeanUtil.copyProperties(newBeeMallShoppingCartItemVO, newBeeMallOrderItem);

*//NewBeeMallOrderMapper文件insert()方法中使用了useGeneratedKeys因此orderId可以获取到*

newBeeMallOrderItem.setOrderId(newBeeMallOrder.getOrderId());

newBeeMallOrderItems.add(newBeeMallOrderItem);

}

*//保存至数据库*

**if** (newBeeMallOrderItemMapper.insertBatch(newBeeMallOrderItems) > 0 && newBeeMallOrderAddressMapper.insertSelective(newBeeMallOrderAddress) > 0) {

*//所有操作成功后，将订单号返回，以供Controller方法跳转到订单详情*

**return** orderNo;

}

NewBeeMallException.fail(ServiceResultEnum.ORDER\_PRICE\_ERROR.getResult());

}

NewBeeMallException.fail(ServiceResultEnum.DB\_ERROR.getResult());

}

NewBeeMallException.fail(ServiceResultEnum.DB\_ERROR.getResult());

}

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

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

}

订单生成的方法中共 80 行代码，总结一下就是先验证，之后进行订单数据封装，最后将订单数据和订单项数据保存到数据库中。

我们结合订单生成流程图来理解，订单生成的详细过程如下：

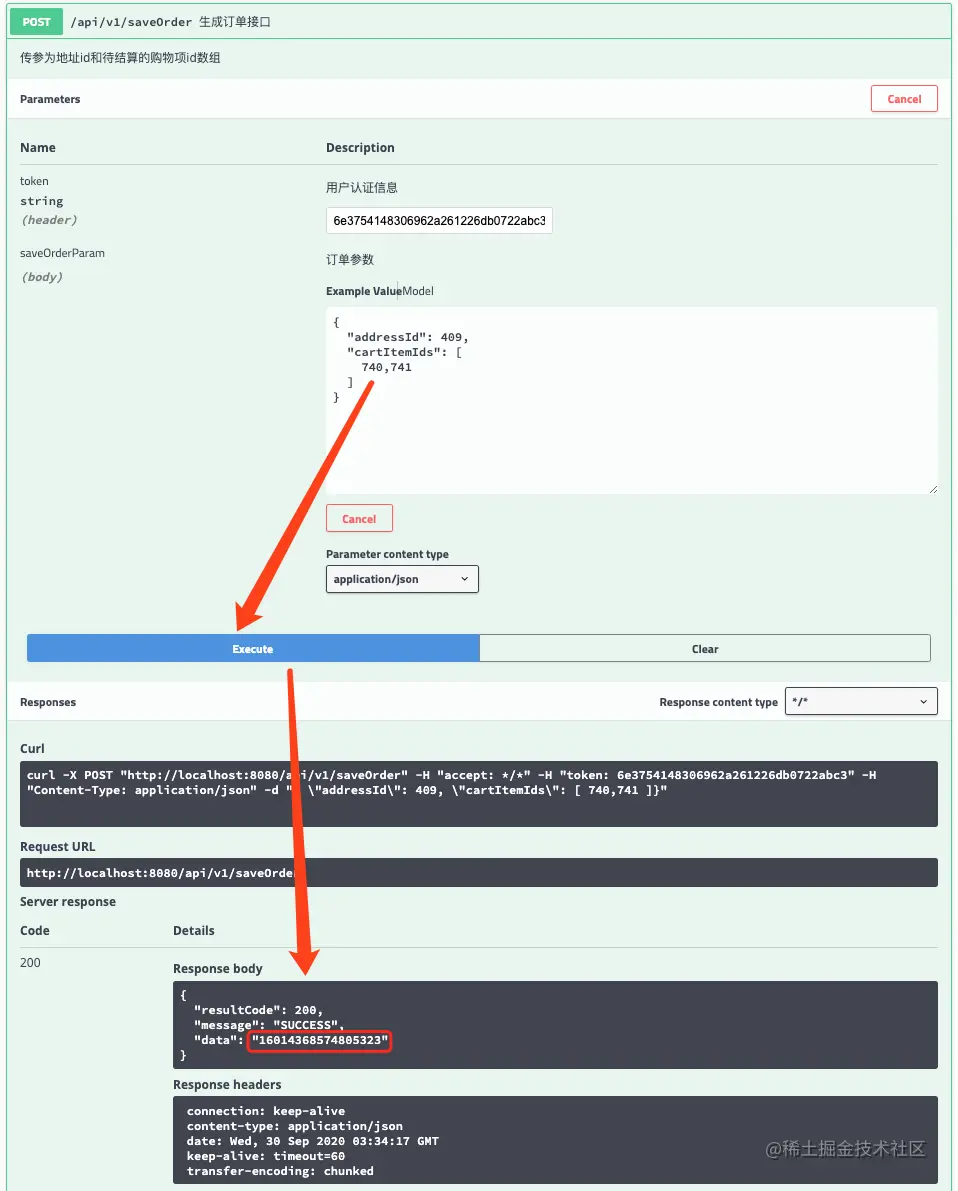
* 首先是检查是否包含已下架商品，如果有则抛出一个异常，无下架商品则继续后续流程
* 继续判断商品数据和商品库存，如果商品数据有误或者商品库存不足也会抛出异常，一切正常则继续后续流程
* 对象的非空判断
* 生成订单后，购物项数据需要删除，这里调用 NewBeeMallShoppingCartItemMapper.deleteBatch() 方法将这些数据批量删除
* 更新商品库存记录
* 判断订单价格，如果所有购物项加起来的数据为 0 或者小于 0 则不继续生成订单
* 生成订单号并封装 NewBeeMallOrder 对象，保存订单记录到数据库中
* 封装订单项数据并保存订单项数据到数据库中
* 生成订单收货地址快照，并保存至数据库
* 返回订单号

至此，订单生成的逻辑就完成了，在自己测试时可以关注一下数据库中的相关记录，功能完成时购物项是否成功删除、商品库存是否成功修改、订单和订单项、订单收货地址是否成功生成。订单即商品和用户信息的结合，需要商品信息、需要用户信息，这样才能组成订单。当然，订单生成只是订单模块中的第一步，后续还有很多步骤需要我们来完成。

由于该方法中同时修改了多张表中的记录，为了保证事务的一致性，在该方法上添加了 @Transactional 注解，一旦该方法执行过程中发生异常，立刻回滚事务，不然可能会出现库存扣了但是订单没生成等等的尴尬场景。

#### 订单生成接口测试

重启项目，打开 swagger-ui 页面，点开“生成订单接口”，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，然后输入购物项 id 和用户的收货地址 id，之后点击”Execute“按钮，结果如下：



最终，生成订单成功，并且得到了一个订单号，得到订单号之后就可以继续进行后续操作了，比如支付、查看订单详情、取消订单等等操作。

## 总结

订单流程图和订单生成步骤都在文中详细的介绍了，大家可以把本文的源码下载到本地并启动项目，结合源码和实际的操作去理解，订单生成的功能牵涉到多张表的数据更改，在自己测试的时候也一定要注意订单数据是否生成，以及数据库中商品、购物项、订单等数据是否正确的修改。