订单成功生成之后还会有其他的后续操作，比如订单支付、订单列表、订单详情查看、商家确认订单、取消订单等等操作，接下来的两篇文章我会接受这部分功能的后端接口实现。

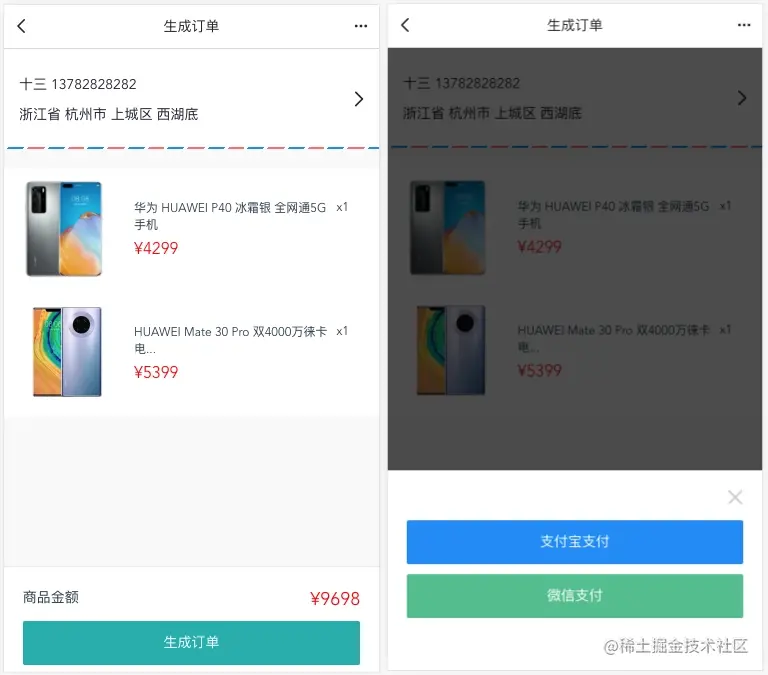
本篇文章的主要内容如下：

* 新蜂商城订单支付模拟接口实现
* 新蜂商城订单详情获取接口实现
* 新蜂商城订单列表接口实现

## 订单支付模拟接口

在新蜂商城第一版本中，订单生成成功后会跳转到详情页，后续步骤也是在详情页进行操作，比如支付的发起、取消订单等等操作。

Vue 版本中，对以上流程做了一些调整，如下图所示：

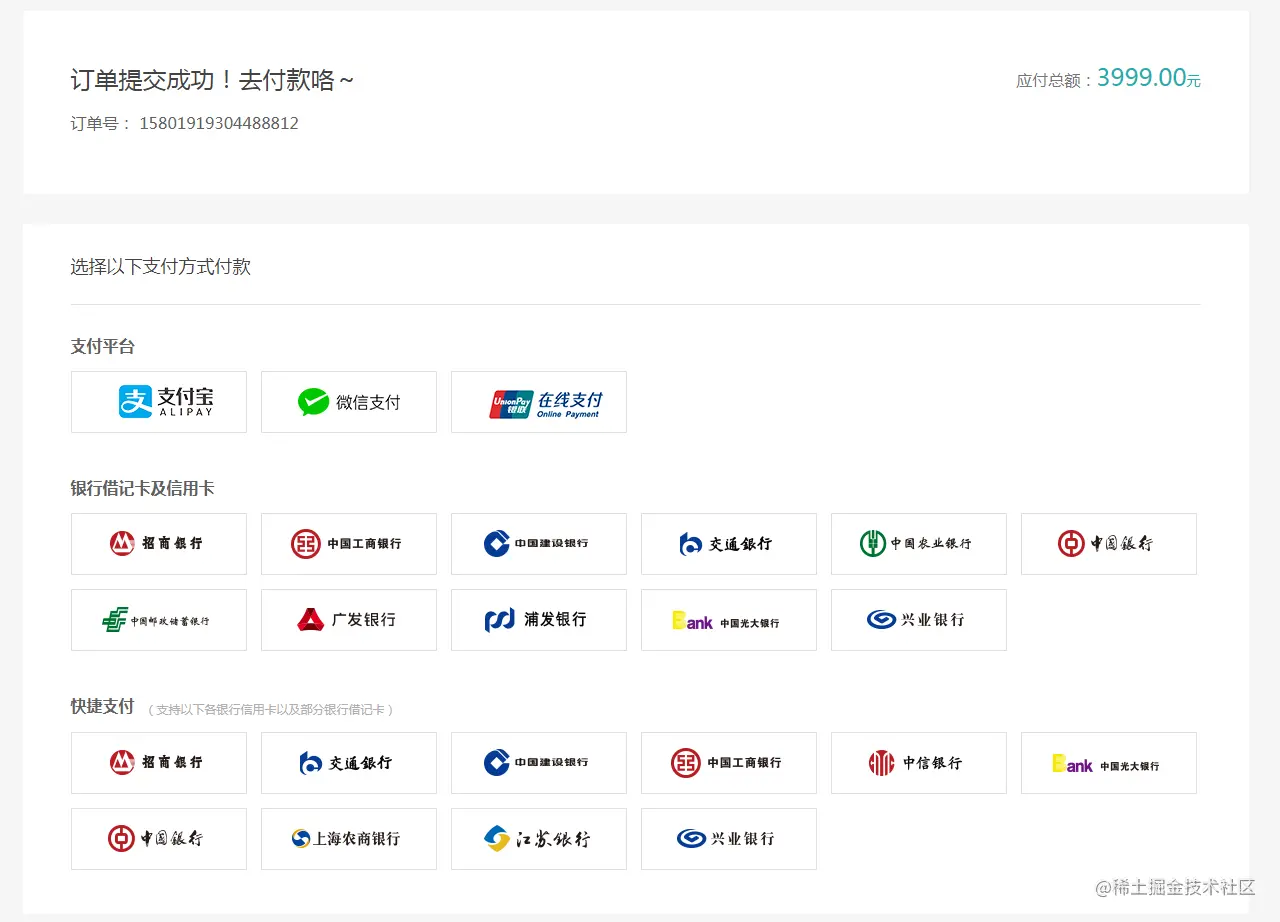


生成订单成功后，会在订单确认页面中出现一个支付的底部弹窗，即模拟支付功能也在该订单确认页面完成。由于没有公司资质无法申请微信和支付宝相关的接口接入权限，所以只能模拟这个功能实现了。

支付步骤如下：

* 选择支付方式

当今社会移动支付非常方便，我们的衣食住行可能都离不开它，甚至每天都会用支付宝或者微信进行交易。不仅仅是第三方支付，各个银行目前也都有网上银行的功能，支持网页上进行支付操作，下图为新蜂商城第一版本中的选择支付页面：



PC 端页面放置多种支付方式并不会显得违和，但是在移动端页面中并没有选择这个设计，Vue 版本的新蜂商城支付弹窗中只保留了“微信支付”和“支付宝支付”两种方式。

* 支付成功

点击支付弹框中的任意一个按钮，就会向支付回调地址发送请求，该按钮是模拟正式的第三方支付接口回调，表示已经支付成功，可以修改订单状态了。

#### 模拟支付接口传参确认

这里我们先来简单的分析一下发送支付请求时向后端请求时的参数：

订单已经生成，而且订单号也已经知道，所以我们需要一个请求参数，即“订单号”。另外，我们还需要记录订单的支付方式，所以另外一个请求参数为“支付方式”。

#### 模拟支付接口编码实现

支付这个步骤最终是为了把订单状态改为支付成功，同时也记录下支付的相关信息，这里记录了支付时间和支付方式，一般还会记录支付的参数和第三方回调信息。

处理代码如下：

**@GetMapping("/paySuccess")**

**@ApiOperation(value = "模拟支付成功回调的接口", notes = "传参为订单号和支付方式")**

**public** Result **paySuccess**(**@ApiParam(value = "订单号")** **@RequestParam("orderNo")** String orderNo, **@ApiParam(value = "支付方式")** **@RequestParam("payType")** **int** payType) {

**String** payResult = newBeeMallOrderService.paySuccess(orderNo, payType);

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

**return** ResultGenerator.genSuccessResult();

} **else** {

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

}

}

这是 Controller 层的代码，负责接收支付回调数据，参数为订单号和支付方式，我们根据这两个参数对订单的状态进行修改，调用的 service 层方法为 paySuccess()，代码如下：

*/\*\**

*\* 订单支付成功*

*\* @param orderNo*

*\* @param payType*

*\* @return*

*\*/*

**@Override**

**public** String **paySuccess**(String orderNo, **int** payType) {

**NewBeeMallOrder** newBeeMallOrder = newBeeMallOrderMapper.selectByOrderNo(orderNo);

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

**if** (newBeeMallOrder.getOrderStatus().intValue() != NewBeeMallOrderStatusEnum.ORDER\_PRE\_PAY.getOrderStatus()) {

NewBeeMallException.fail("非待支付状态下的订单无法支付");

}

newBeeMallOrder.setOrderStatus((**byte**) NewBeeMallOrderStatusEnum.OREDER\_PAID.getOrderStatus());

newBeeMallOrder.setPayType((**byte**) payType);

newBeeMallOrder.setPayStatus((**byte**) PayStatusEnum.PAY\_SUCCESS.getPayStatus());

newBeeMallOrder.setPayTime(**new** **Date**());

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

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

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

} **else** {

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

}

}

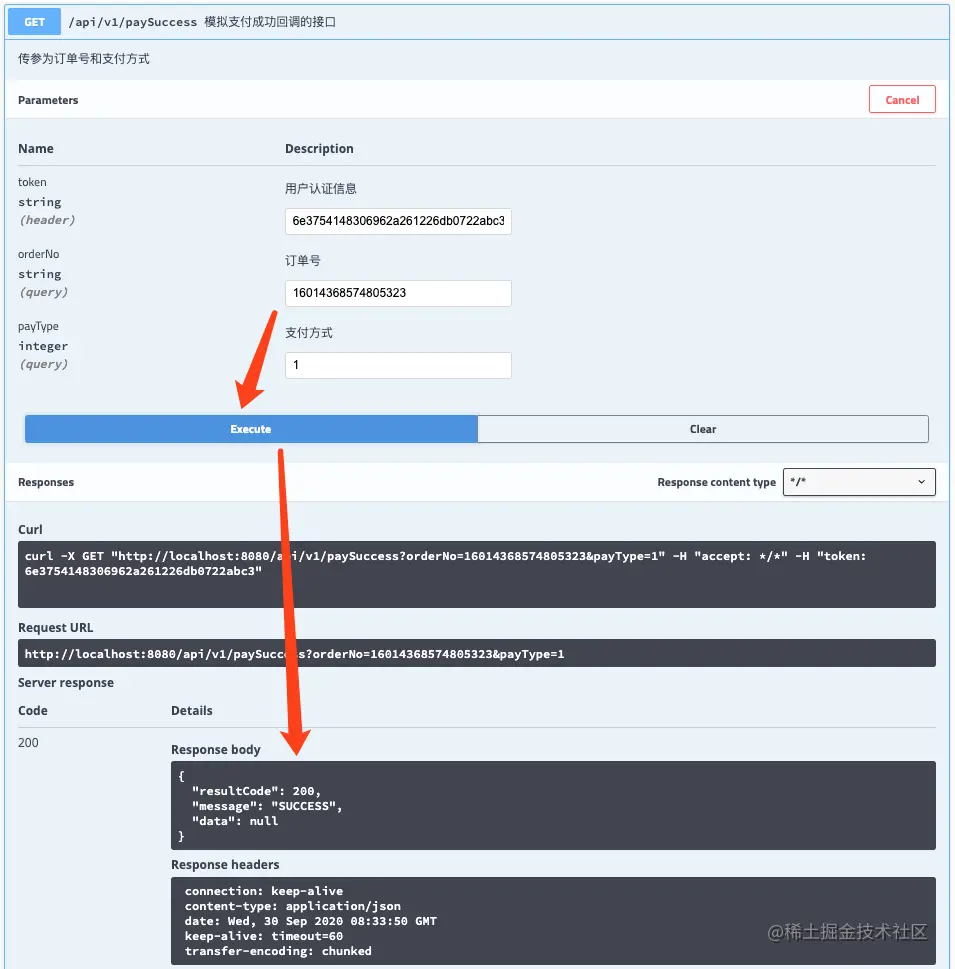
**return** ServiceResultEnum.ORDER\_NOT\_EXIST\_ERROR.getResult();

}

根据订单号查询订单，并进行非空判断和订单状态的判断，如果订单已经不是“待支付”状态下的订单则不进行后续操作，如果验证通过，将该订单的相关状态和支付时间进行修改，之后调用 Mapper 方法进行实际的入库操作。

#### 接口测试

重启项目，打开 swagger-ui 页面，点开“模拟支付成功回调的接口”，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，然后输入订单号和支付方式，之后点击”Execute“按钮，结果如下：



返回的 resultCode 为 200，代表接口请求成功。

测试时，你可以再去查一下数据库，看一下在接口调用后，订单信息是否已经被正确的修改。

## 订单详情接口实现

订单详情页面也是订单流程中非常重要的一个环节。

#### 订单详情页设计

该页面是商家与客户之间最直接的一个纽带，对于商家来说，体现了商家对消费者提供的销售服务，商品信息、订单信息、物流信息都在该页面实时展示给用户，同时也为商家带来了便捷，商家的任何变动都会实时显示在该页面；对于消费者来说，该页面会展现订单的重要信息，所有用户关心的信息都展示在这里，同时，用户也可以在该页面实时关注订单的变化和新的动态。

订单详情页主要有两个功能：

##### 展示基本的订单信息

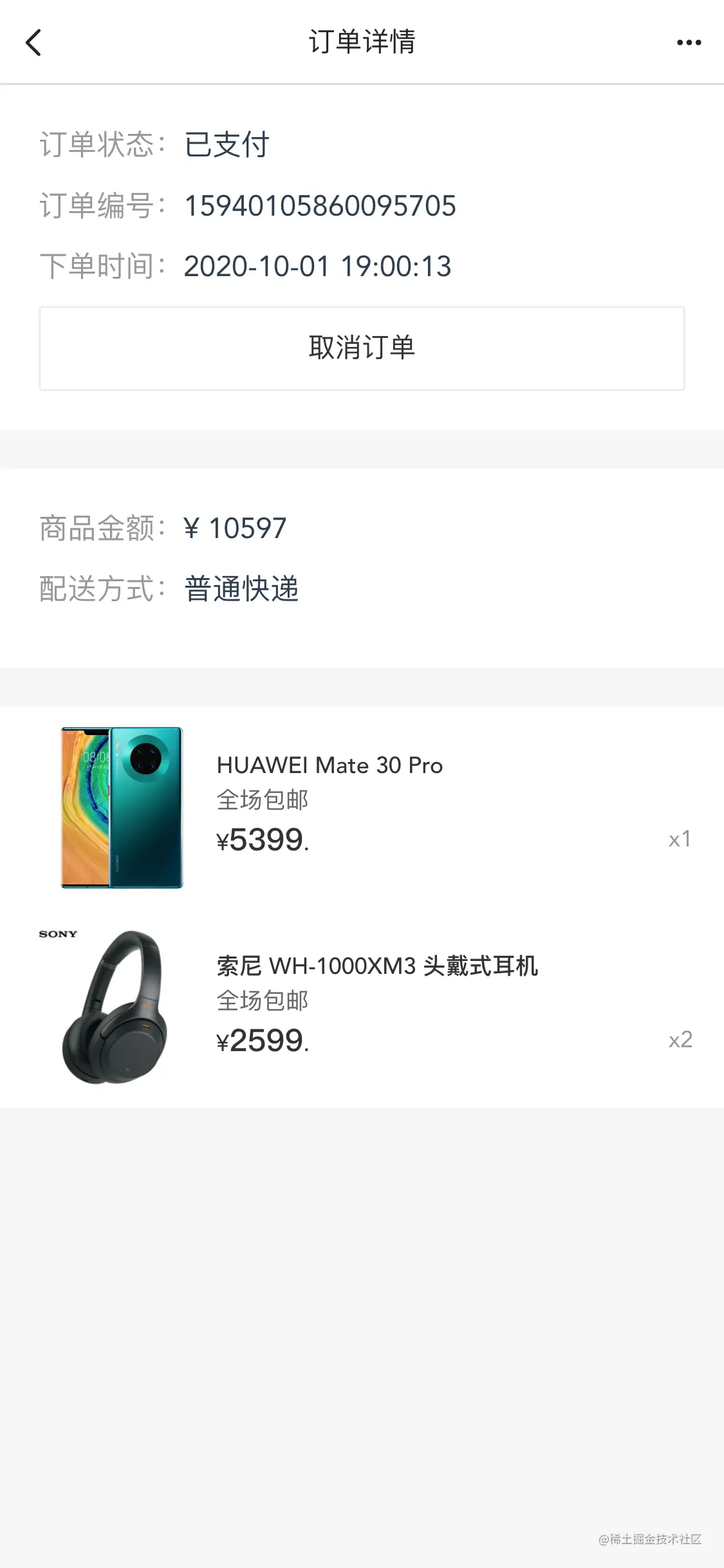
* 订单基本信息：订单号、订单状态、价格等等信息
* 配送信息：物流信息、收货地址信息
* 商品信息：商品名称、商品图、购买数量等等
* 发票信息：发票信息和开票状态
* 客服：联系商家或者拨打电话

##### 为用户提供订单操作

用户可以在该页面进行支付订单、取消订单、确认订单等等操作。

#### 数据格式定义

关于订单详情页的展示内容我们已经介绍完毕，接下来我们结合实际图片来讲解订单详情页接口返回的数据格式。如下图所示即为页面中需要渲染的内容，新蜂商城订单详情页与淘宝、京东等线上商城的订单详情页相比是简单了很多，并没有配送信息、发票信息等等，只有简单的订单信息展示和商品信息展示。



针对详情页进行分析后，需要返回数据的格式就得出来了，订单详情 VO 对象编码如下：

*/\*\**

*\* 订单详情页页面VO*

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

**@ApiModelProperty("订单号")**

**private** String orderNo;

**@ApiModelProperty("订单价格")**

**private** Integer totalPrice;

**@ApiModelProperty("订单支付状态码")**

**private** Byte payStatus;

**@ApiModelProperty("订单支付方式")**

**private** Byte payType;

**@ApiModelProperty("订单支付方式")**

**private** String payTypeString;

**@ApiModelProperty("订单支付实践")**

**private** Date payTime;

**@ApiModelProperty("订单状态码")**

**private** Byte orderStatus;

**@ApiModelProperty("订单状态")**

**private** String orderStatusString;

**@ApiModelProperty("创建时间")**

**@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8")**

**private** Date createTime;

**@ApiModelProperty("订单项列表")**

**private** List<NewBeeMallOrderItemVO> newBeeMallOrderItemVOS;

}

订单详情 VO 对象中还有一个订单项列表 newBeeMallOrderItemVOS，订单详情页中展示的商品信息就是使用这部分数据，NewBeeMallOrderItemVO 对象字段如下：

*/\*\**

*\* 订单详情页页面订单项VO*

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

**@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;

}

#### 接口实现

详情接口的路径我们定义为 /orders/{orderNo}，接收的参数为订单号，一般也没有商城系统会直接把订单 id 暴露出去，在 NewBeeMallOrderAPI 中新增 orderDetailPage() 来对这个接口请求进行处理，新增如下代码：

**@GetMapping("/order/{orderNo}")**

**@ApiOperation(value = "订单详情接口", notes = "传参为订单号")**

**public** Result<NewBeeMallOrderDetailVO> **orderDetailPage**(**@ApiParam(value = "订单号")** **@PathVariable("orderNo")** String orderNo, **@TokenToMallUser** MallUser loginMallUser) {

**return** ResultGenerator.genSuccessResult(newBeeMallOrderService.getOrderDetailByOrderNo(orderNo, loginMallUser.getUserId()));

}

orderNo 这个参数是订单记录的唯一订单号，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 NewBeeMallOrderService 中的 getOrderDetailByOrderNo() 方法获取到 NewBeeMallOrderDetailVO 对象，getOrderDetailByOrderNo() 方法的实现方式就是根据主键订单号去查询数据库中的订单表并返回订单详情所需的数据，业务层实现代码如下：

*/\*\**

*\* 获取订单详情*

*\**

*\* @param orderNo*

*\* @param userId*

*\* @return*

*\*/*

**@Override**

**public** NewBeeMallOrderDetailVO **getOrderDetailByOrderNo**(String orderNo, Long userId) {

**NewBeeMallOrder** newBeeMallOrder = newBeeMallOrderMapper.selectByOrderNo(orderNo);

*//非空判断*

**if** (newBeeMallOrder == null) {

NewBeeMallException.fail(ServiceResultEnum.DATA\_NOT\_EXIST.getResult());

}

*//是否为自己的订单*

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

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

}

List<NewBeeMallOrderItem> orderItems = newBeeMallOrderItemMapper.selectByOrderId(newBeeMallOrder.getOrderId());

*//获取订单项数据*

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

List<NewBeeMallOrderItemVO> newBeeMallOrderItemVOS = BeanUtil.copyList(orderItems, NewBeeMallOrderItemVO.class);

**NewBeeMallOrderDetailVO** newBeeMallOrderDetailVO = **new** **NewBeeMallOrderDetailVO**();

BeanUtil.copyProperties(newBeeMallOrder, newBeeMallOrderDetailVO);

newBeeMallOrderDetailVO.setOrderStatusString(NewBeeMallOrderStatusEnum.getNewBeeMallOrderStatusEnumByStatus(newBeeMallOrderDetailVO.getOrderStatus()).getName());

newBeeMallOrderDetailVO.setPayTypeString(PayTypeEnum.getPayTypeEnumByType(newBeeMallOrderDetailVO.getPayType()).getName());

newBeeMallOrderDetailVO.setNewBeeMallOrderItemVOS(newBeeMallOrderItemVOS);

**return** newBeeMallOrderDetailVO;

} **else** {

NewBeeMallException.fail(ServiceResultEnum.ORDER\_ITEM\_NULL\_ERROR.getResult());

**return** null;

}

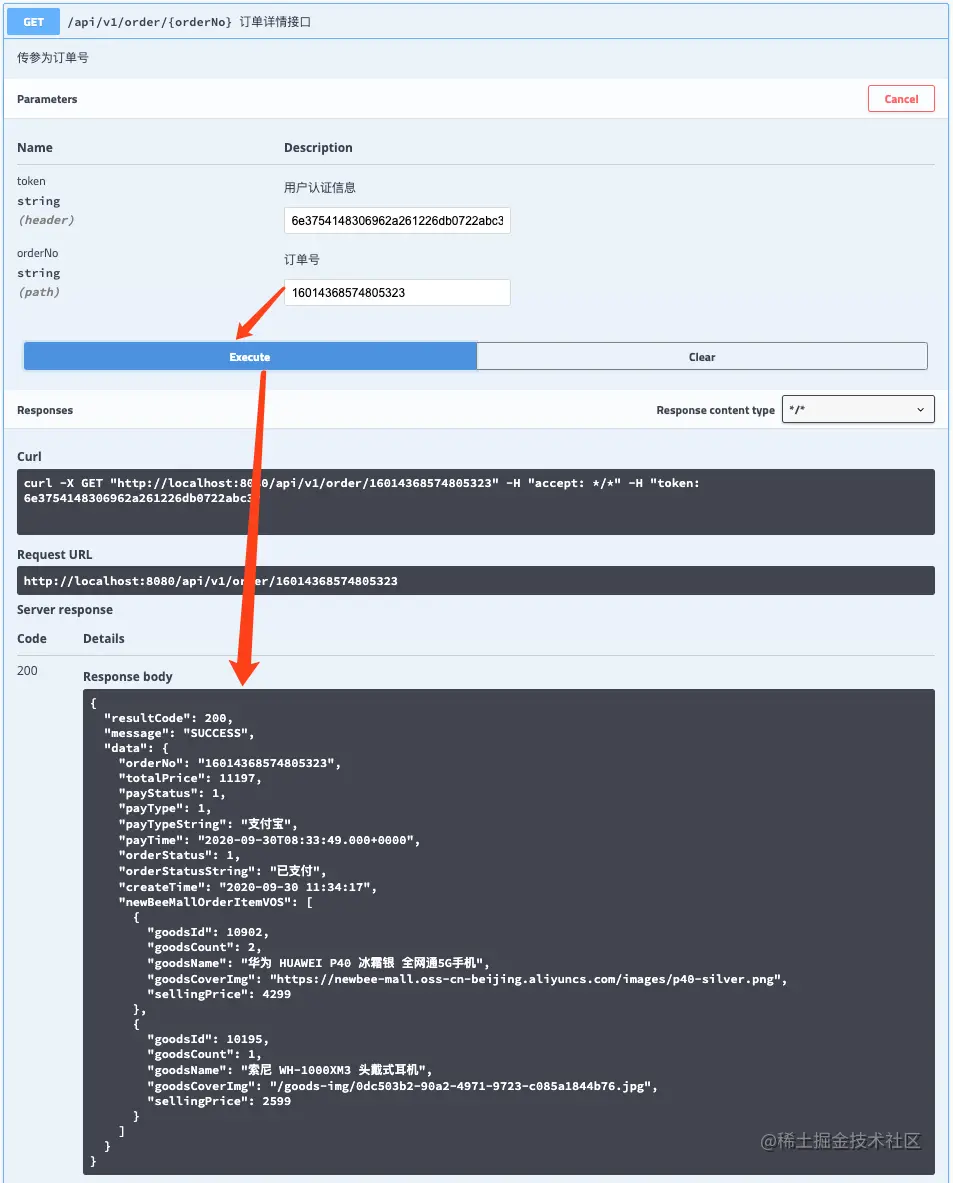
}

逻辑如下：

* 根据 orderNo 订单号来查询订单数据，如果不存在则提示错误信息，存在则继续后续逻辑。
* 判断订单的 userId 是否为当前登录的用户 id，如果不是则为非法请求，不能查看别人的订单信息。
* 之后根据订单 id 去订单项表查询数据。
* 最后封装 NewBeeMallOrderDetailVO 订单详情页数据并返回。

#### 接口测试

重启项目，打开 swagger-ui 页面，点开“订单详情接口”，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，然后输入订单号，之后点击”Execute“按钮，结果如下：



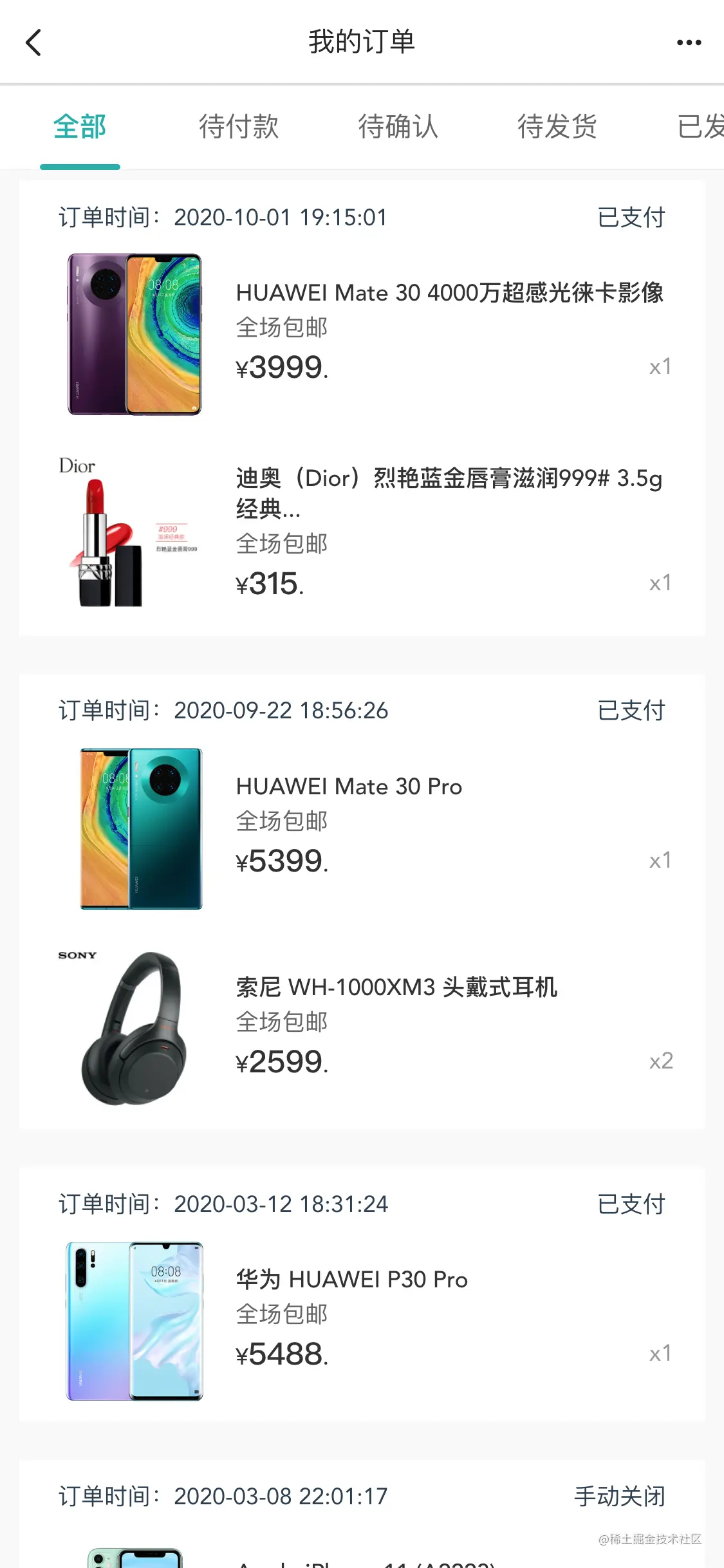
订单详情页所需的数据都通过该接口返回了，接口测试成功。

## 订单列表接口

订单生成后就能够在个人中心的订单列表中看到相关数据了，各种状态的订单都在这个列表中显示，这里我们就来讲解一下订单列表接口的功能实现。

#### 数据格式定义

如下图所示即为新蜂商城订单列表页面中需要渲染的内容，订单列表肯定是一个 List 对象，因此后台返回数据时需要一个订单列表对象，对象中的字段有订单表中的字段，也有订单项表中的字段，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：



列表数据中包括订单状态、订单交易时间、商品标题字段、商品预览图字段、商品价格字段、商品购买数量字段，通过分析大家应该看出来它与购物车列表中的字段类似，不过购物项表和订单项表在新蜂商城的表结构设计中是完全不同的，订单列表对象中的字段是读取的订单表和订单项表，与购物车列表中读取商品表的做法有很大的区别。

一个订单中可能会有多个订单项，所以订单 VO 对象中也有一个订单项 VO 的列表对象，订单列表中返回的 VO 对象编码如下：

*/\*\**

*\* 订单列表页面VO*

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

**private** Long orderId;

**@ApiModelProperty("订单号")**

**private** String orderNo;

**@ApiModelProperty("订单价格")**

**private** Integer totalPrice;

**@ApiModelProperty("订单支付方式")**

**private** Byte payType;

**@ApiModelProperty("订单状态码")**

**private** Byte orderStatus;

**@ApiModelProperty("订单状态")**

**private** String orderStatusString;

**@ApiModelProperty("创建时间")**

**@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8")**

**private** Date createTime;

**@ApiModelProperty("订单项列表")**

**private** List<NewBeeMallOrderItemVO> newBeeMallOrderItemVOS;

}

订单项 VO 对象编码如下：

*/\*\**

*\* 订单详情页页面订单项VO*

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

**@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，上述订单列表中的字段，可以通过分别查询 tb\_newbee\_mall\_order 订单表和 tb\_newbee\_mall\_order\_item 订单项表来获取，实现逻辑如下：

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

*/\*\**

*\* 我的订单列表*

*\*/*

**@Override**

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

**int** total = newBeeMallOrderMapper.getTotalNewBeeMallOrders(pageUtil);

List<NewBeeMallOrder> newBeeMallOrders = newBeeMallOrderMapper.findNewBeeMallOrderList(pageUtil);

List<NewBeeMallOrderListVO> orderListVOS = **new** **ArrayList**<>();

**if** (total > 0) {

*//数据转换 将实体类转成vo*

orderListVOS = BeanUtil.copyList(newBeeMallOrders, NewBeeMallOrderListVO.class);

*//设置订单状态中文显示值*

**for** (NewBeeMallOrderListVO newBeeMallOrderListVO : orderListVOS) {

newBeeMallOrderListVO.setOrderStatusString(NewBeeMallOrderStatusEnum.getNewBeeMallOrderStatusEnumByStatus(newBeeMallOrderListVO.getOrderStatus()).getName());

}

List<Long> orderIds = newBeeMallOrders.stream().map(NewBeeMallOrder::getOrderId).collect(Collectors.toList());

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

List<NewBeeMallOrderItem> orderItems = newBeeMallOrderItemMapper.selectByOrderIds(orderIds);

Map<Long, List<NewBeeMallOrderItem>> itemByOrderIdMap = orderItems.stream().collect(groupingBy(NewBeeMallOrderItem::getOrderId));

**for** (NewBeeMallOrderListVO newBeeMallOrderListVO : orderListVOS) {

*//封装每个订单列表对象的订单项数据*

**if** (itemByOrderIdMap.containsKey(newBeeMallOrderListVO.getOrderId())) {

List<NewBeeMallOrderItem> orderItemListTemp = itemByOrderIdMap.get(newBeeMallOrderListVO.getOrderId());

*//将NewBeeMallOrderItem对象列表转换成NewBeeMallOrderItemVO对象列表*

List<NewBeeMallOrderItemVO> newBeeMallOrderItemVOS = BeanUtil.copyList(orderItemListTemp, NewBeeMallOrderItemVO.class);

newBeeMallOrderListVO.setNewBeeMallOrderItemVOS(newBeeMallOrderItemVOS);

}

}

}

}

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

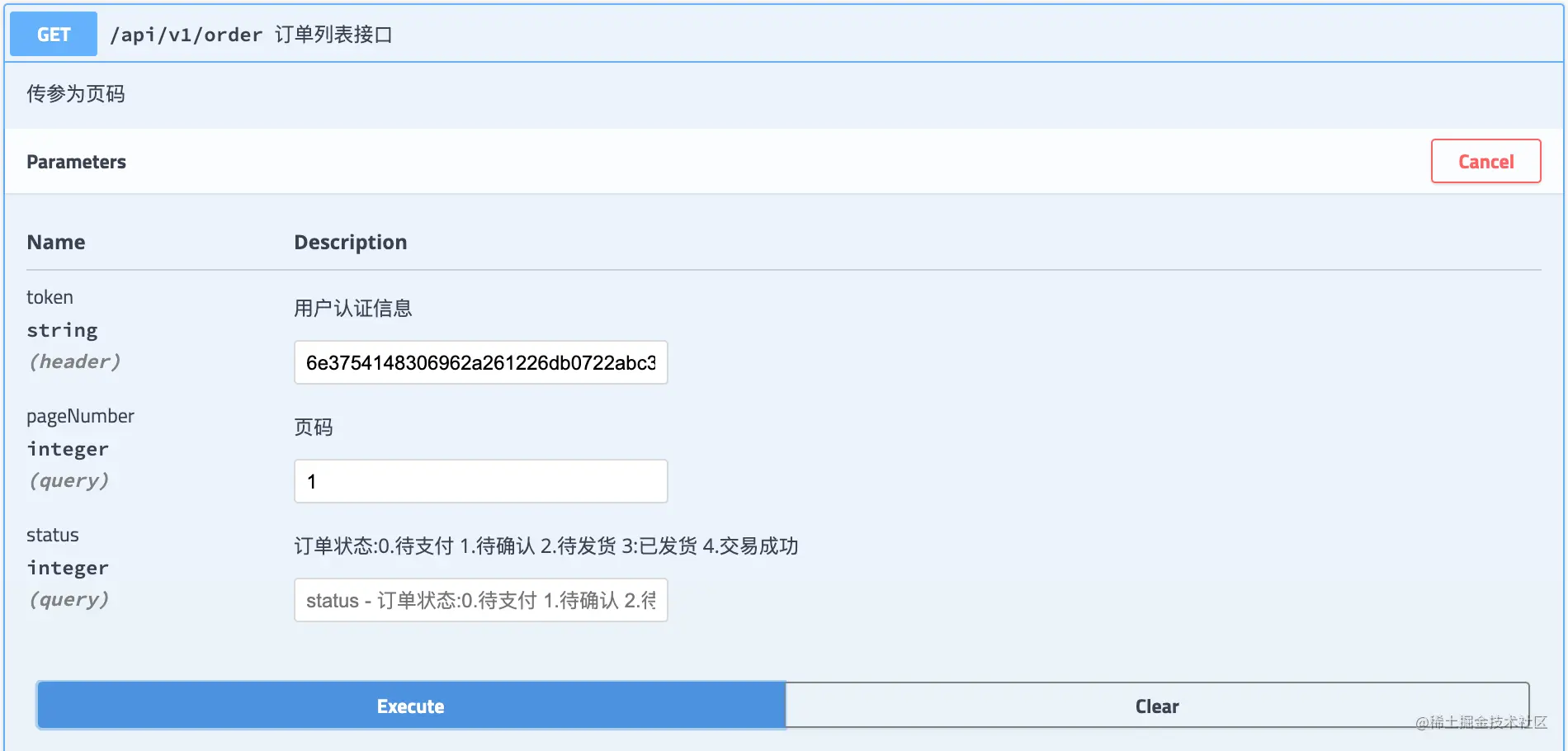
**return** pageResult;

}

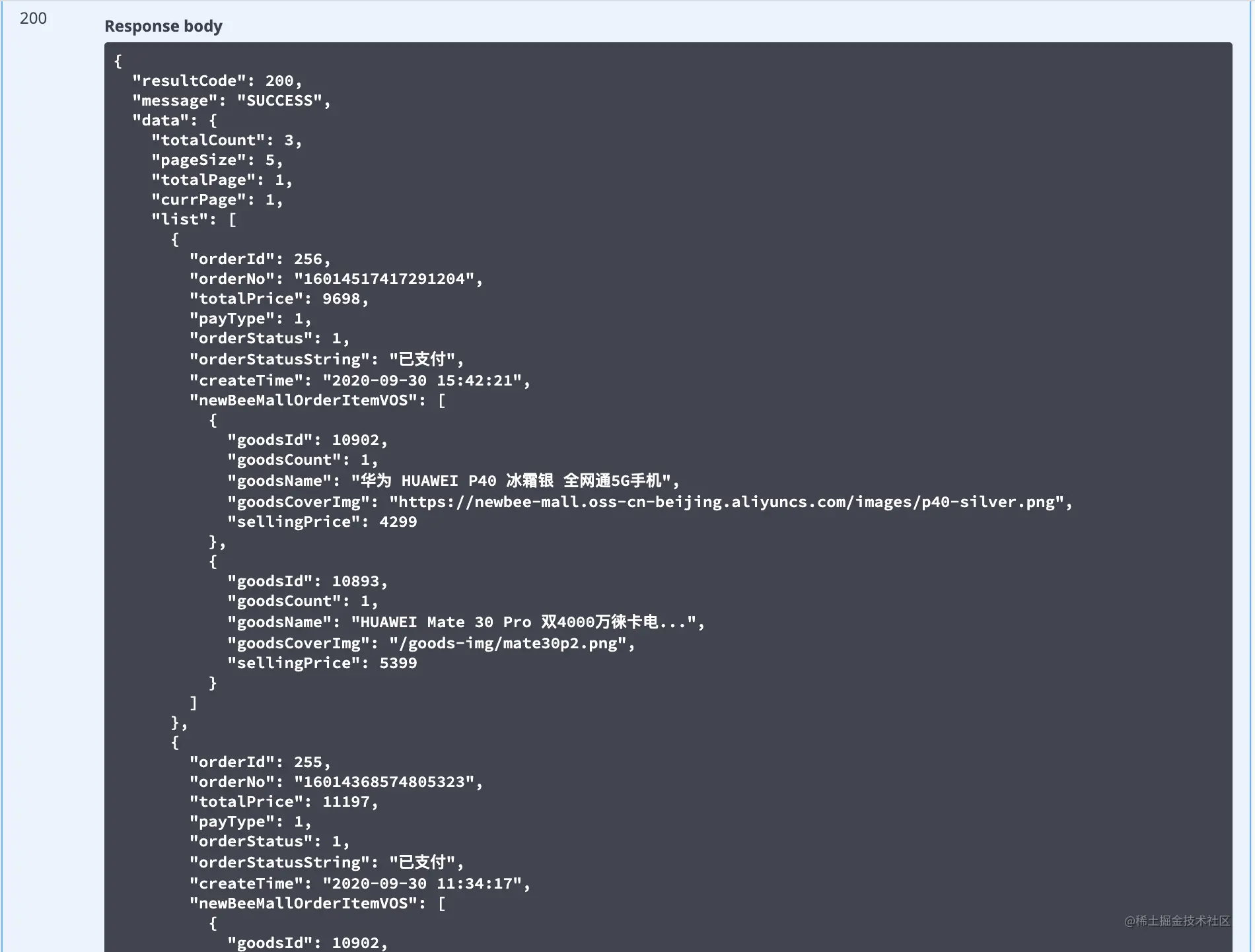
我们定义了 getMyOrders() 方法并传入 PageUtil 对象作为参数，该对象中会有分页参数和用户的 userId，之后通过 SQL 查询出当前 userId 下的订单列表数据以及每个订单所关联的订单项列表数据，再之后是填充数据，将相关字段封装到 NewBeeMallOrderListVO 对象中，并将封装好的 List 对象返回。

#### 接口测试

重启项目，打开 swagger-ui 页面，点开“订单列表接口”，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，另外两个参数是页码和订单状态，表示请求第几页的数据和查询哪种状态下的订单列表，如果不传订单状态则表示全部查出来。



之后点击”Execute“按钮，结果如下：



列表接口请求成功。