回到首页

芋道源码 —— 知识星球

我是一段不羁的公告！
记得给芳芳这 3 个项目加油，添加一个 STAR 噢。
https://github.com/YunaiV/SpringBoot-Labs
https://github.com/YunaiV/onemall
https://github.com/YunaiV/ruoyi-vue-pro
2018-03-01
数据库实体设计

数据库实体设计 —— 交易（2.1）之订单信息

> 芳芳目前正在做一个开源的电商项目，胖友可以 star 下。
> https://gitee.com/zhijiantianya/onemall

1. 概述

本文主要分享交易模块的订单信息的数据库实体设计。

基于如下信息，逆向猜测数据库实体：

> 有赞云提供的商家管理订单API
> 有赞云提供的买家下单API
> 有赞微商城的订单管理
> 有赞云开发指南的交易场景【订单正向调用】
> 店铺后台如何管理订单
>
> 【护脸旁白】
> 笔者非电商行业出身 && 非有赞工程师，所以有错误或不合理的地方，烦请斧正和探讨。
> 有赞是个各方面都很 NICE 的公司，推荐 。

2. 背景了解

参见 《订单中心设计》 ，写的非常非常非常棒。

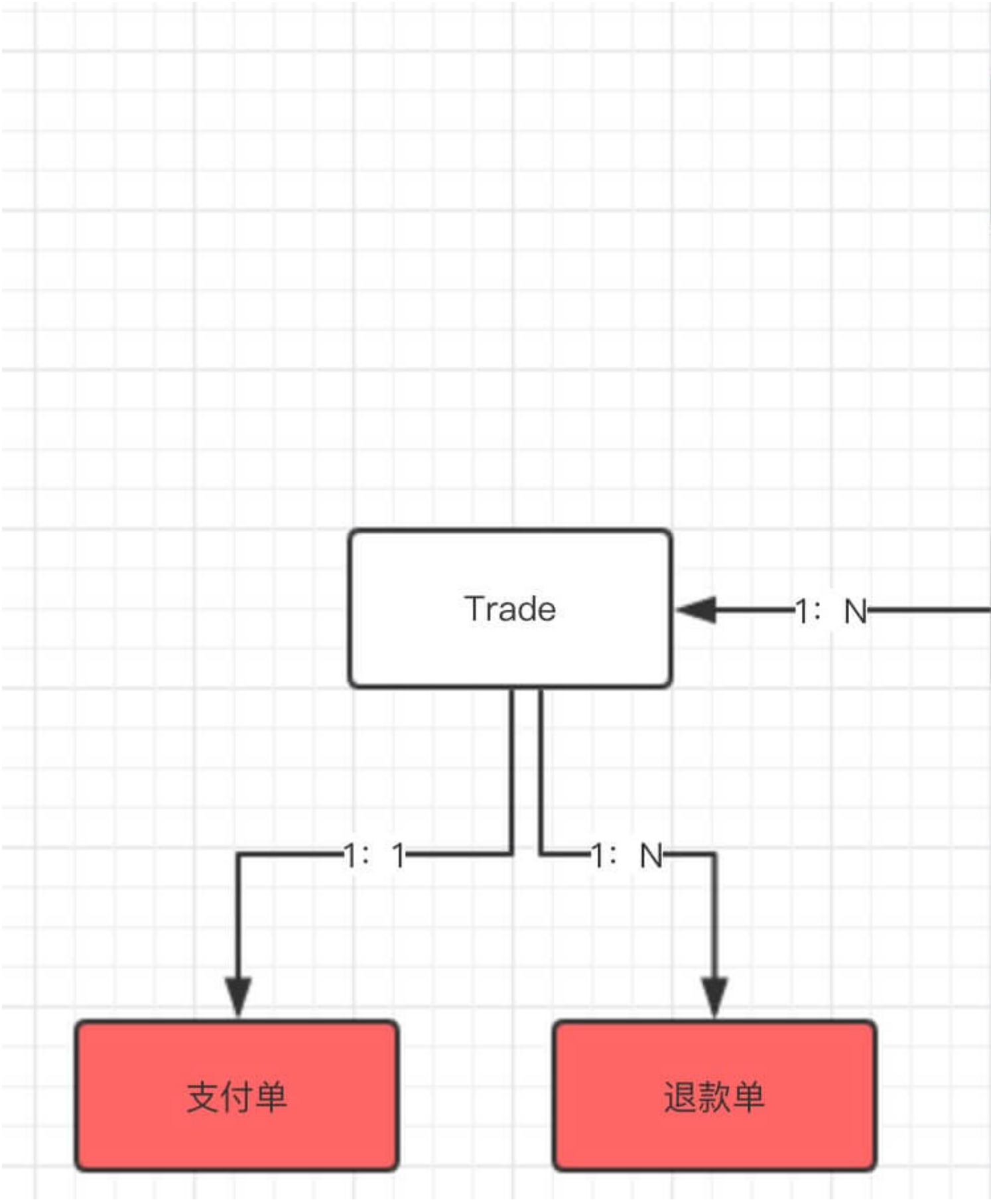
2.1 界面

1. 运营后台-订单管理-所有订单

2. 运营后台-订单管理-订单详情

3. 数据库实体

整体实体类关系如下图：



本文内容
  - Trade ： TradeOrder = 1：N 。一次交易，可以有多个交易明细（交易订单）。

未来章节的内容
  - Trade ： 支付单 = 1：1 。一次交易，一次支付。
  - Trade ： 退款单 = 1：N 。一个交易明细，可以多次退款，所以可以有多个退款单。
  - TradeOrder ： TradeRefund = 1：1 。一个交易明细，可以发起一次退款维权。
  - TradeOrder ： 物流单 = N：1 。多个交易明细，可以合并发货。

注意，不同于我们常用的京东电商平台，不存在拆单的情况：

基于商家的拆单

- 因为有赞是一个 Sass 化的去中心化的电商系统，而不是一个中心化的电商平台，没有跨商家下单的需求

基于仓库的自动拆单

- 因为有赞买家购买的每种（ 相同商品购买多个，也是一种 ）商品都会生成一条交易明细，卖家可以组合排列发货。例如，买家购买了 A、B、C 商品，卖家可以按照 [A,B,C]，也可以 [A,B]+[C]，也可以 [A]+[B]+[C] 以及等等手动发货。后面，我们会另外单独分享物流这块。

关于订单拆单的逻辑，胖友可以阅读 《订单拆单：把产品逻辑拆给你看》 文章：

> 订单拆单在电商订单中很常见，也比较复杂。拆单也有两次：
>
> 一次是在用户提交订单之后、支付之前拆单，这次是拆分的订单；
> 另一次是在用户下单之后，商家发货之前，去拆分发货单（SKU层面）。

两次拆单的原则不同：

第一次拆单是为了区分平台商家、方便财务结算；
第二次拆单是为了按照最后的发货包裹进行拆单，如不同仓库、不同运输要求的SKU、包裹重量体积限制等因素（第二次拆单的有些步骤也可以放在第一步）。

3.1 Trade

Trade ，交易。一次下单，生成一条交易记录，即使多种商品。

字段较多，我们进行简单的切块。

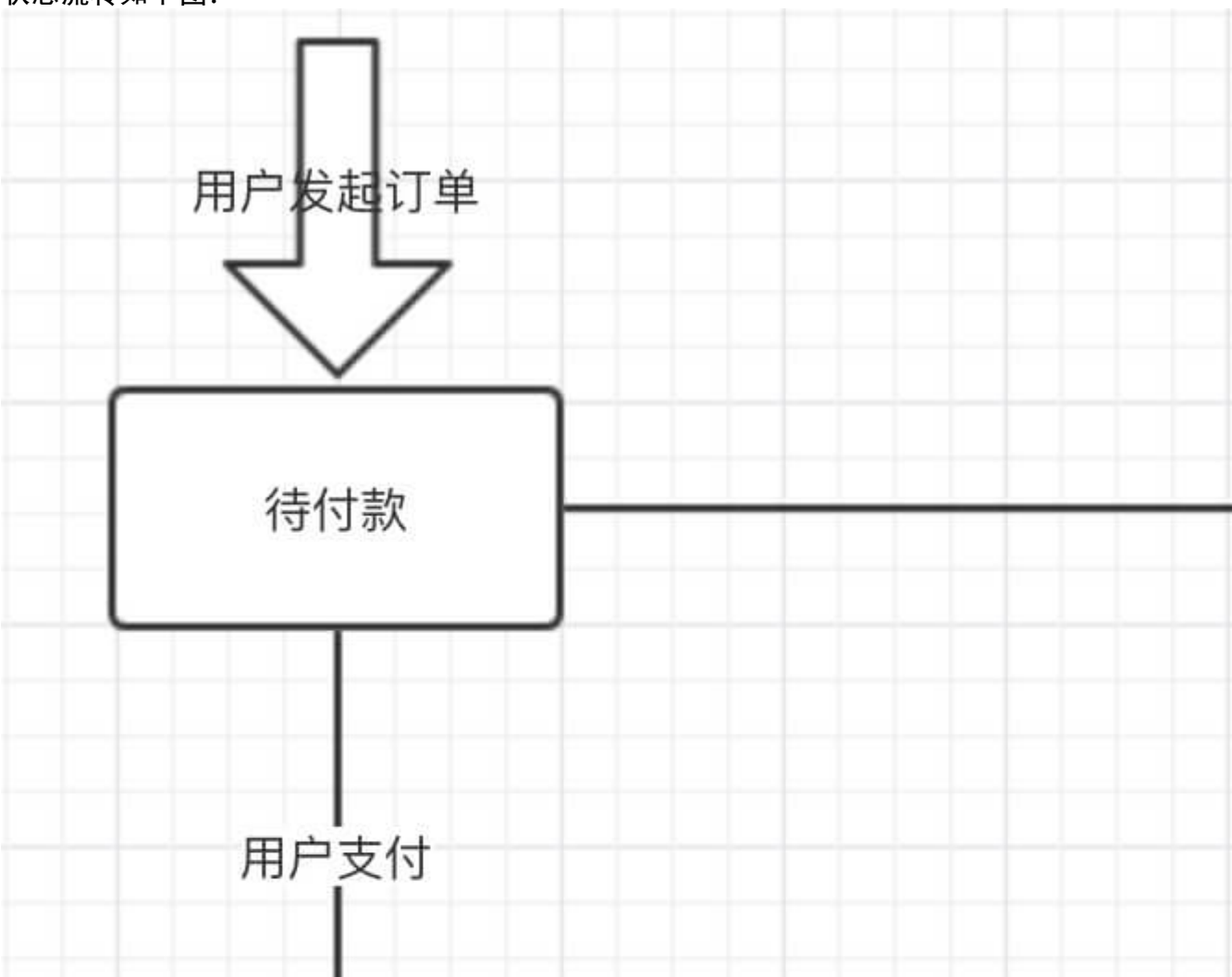
3.1.1 基础字段

```
/**
 * 交易编号
 *
 * 唯一，例如：E20180125232933007700006
 */
private String id;
/**
 * 店铺编号
 */
private Integer shopId;
/**
 * 交易类型
 *
 * 0：FIXED （一口价）
 * 1：GIFT （送礼）
 * 2：BULK_PURCHASE（来自分销商的采购）
 * 3：PRESENT （赠品领取）
 * 4：GROUP （拼团订单）
 * 5：PIFA （批发订单）
 * 6：COD （货到付款）
 * 7：PEER （代付）
 * 8：QRCODE（扫码商家二维码直接支付的交易）
 * 9：QRCODE_3RD（线下收银台二维码交易）
 */
private Integer type;
/**
 * 交易主状态。
 *
 * TODO 芋艿：TRADE_NO_CREATE_PAY（没有创建支付交易）
 * 3：WAIT_BUYER_PAY（等待买家付款）
 * TODO 芋艿：WAIT_PAY_RETURN（等待支付确认）
 * TODO 芋艿：WAIT_GROUP（等待成团，即：买家已付款，等待成团）
 * 5：WAIT_SELLER_SEND_GOODS（等待卖家发货，即：买家已付款）
 * 6：WAIT_BUYER_CONFIRM_GOODS（等待买家确认收货，即：卖家已发货）
 * 100：TRADE_BUYER_SIGNED（买家已签收，即：交易已完成）
 * 99：TRADE_CLOSED（付款以后用户退款成功或者付款超时或商家取消，即：交易已关闭）
 */
private Integer status;
/**
 * 交易创建时间
 */
private Date createTime;
/**
```

```
     * 交易更新时间。
     *
     * 当交易的：状态改变、备注更改、星标更改 等情况下都会刷新更新时间
     */
    private Date updateTime;
    /**
     * 关闭类型
     *
     * 1-超时未支付
     * 2-退款关闭
     * 4-买家取消
     * 15-已通过货到付款交易
     * ... 可能还有其他关闭原因
     */
    private Integer closeType;
    /**
     * 关闭时间
     */
    private Date closeTime;
```

id ，交易号。例如：E20180125232933007700006 。笔者猜测格式为：首字母 + 年月日时分秒 + 0077（可能是有什么业务含义） + 自增序列。
shopId ，店铺编号。
type ，订单类型。有赞支持多种订单类型，本文以 type = 0 ，普通订单类型。其他类型，我们会另外写文章进行分享。
status ，交易主状态。交易会有较多分支状态，例如退款状态、物流状态等等状态。不同于分支状态，主状态控制整个订单的生命周期。

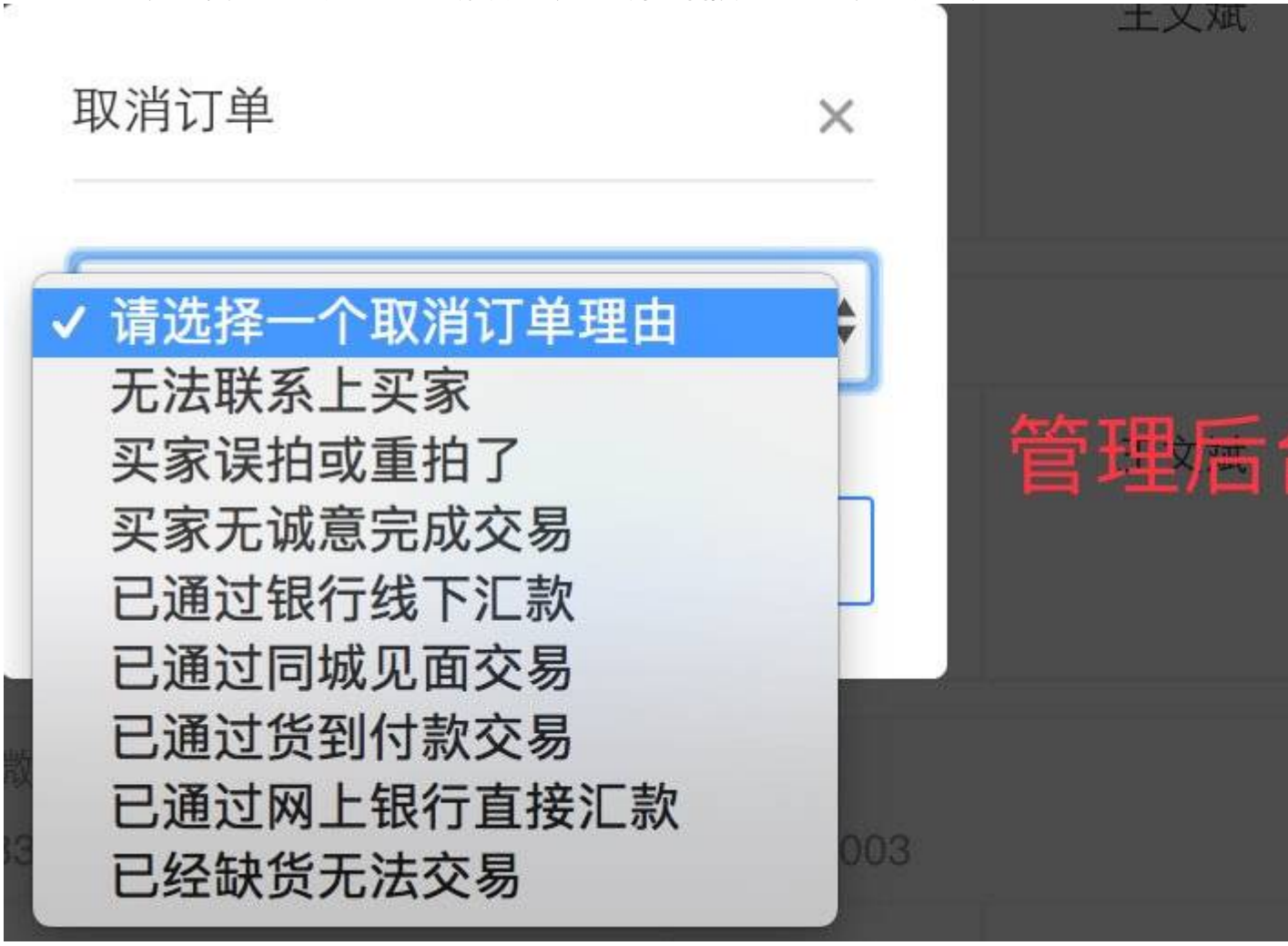
- 如下状态笔者暂未测试到，等未来进行补充：1）没有创建支付交易；2）等待支付确认；3）买家已付款，等待成团。
- 状态流转如下图：



- 状态的设计是非常值得探讨，期盼胖友的讨论。

createTime ，交易创建时间。注意，记录流程的数据库表，尽量记录每个操作的时间，方便业务上回溯与数据分析。

closeType ，关闭类型。交易状态最终变成已关闭有多种情况，通过该字段记录。



3.1.2 商品信息

```
/**
 * 商品购买数量。
 *
 * 当一个 trade 对应多个 order 的时候，值为所有商品购买数量之和
 */
private Integer num;
/**
 * 商品购买种类
 *
 * 当一个 trade 对应多个 order 的时候，值为 order 的数量
 */
private Integer kind;
/**
 * 商品数字编号。
 *
 * 当一个trade对应多个order的时候，值为第一个交易明细中的商品的编号
 */
private Integer itemId;
/**
```

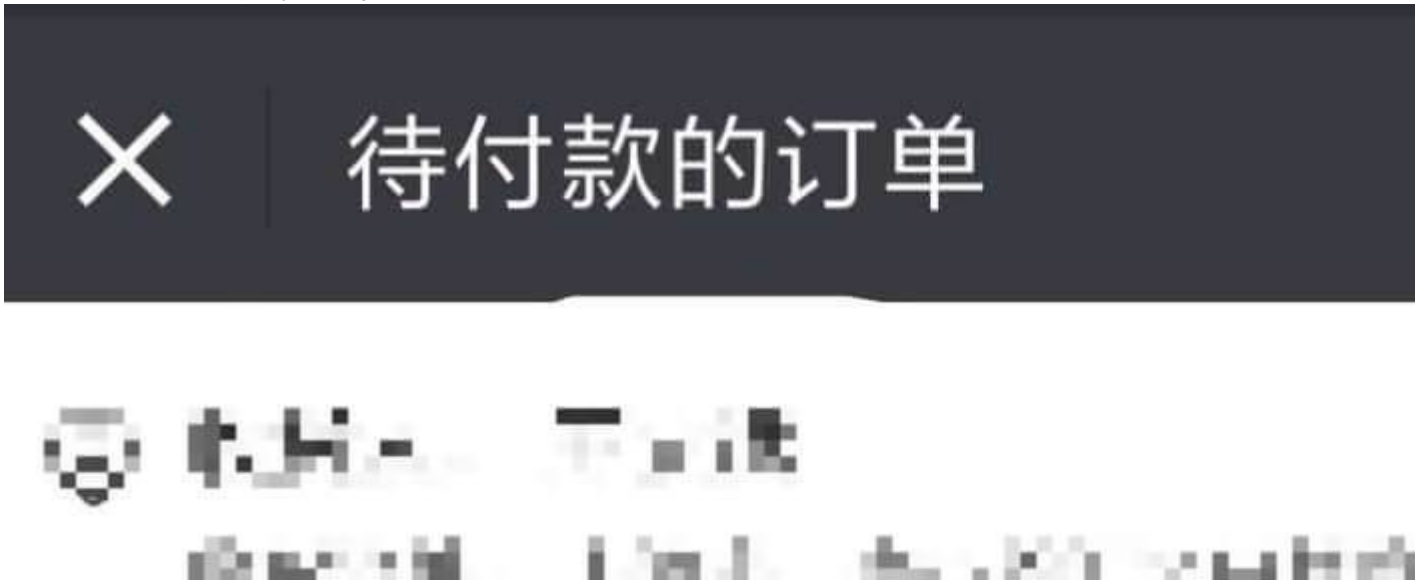
```
 * 商品价格，单位：分。
 *
 * 当一个trade对应多个order的时候，值为第一个交易明细中的商品的价格
 */
private Integer price;
/**
 * 商品主图片地址。
 *
 * 当一个trade对应多个order的时候，值为第一个交易明细中的商品的图片地址
 */
private String picPath;
/**
 * 交易标题。
 *
 * 以首个商品标题作为此标题的值
 */
private String title;
```

我们可以看到，该快字段主要冗余了第一个商品的基础信息到交易记录中。

3.1.3 买家信息

```
/**
 * 买家用户编号
 */
private Integer buyerId;
//    @JsonProperty(value = "fans_info")
//    /**
//     * 用户信息
//     */
//    private YouzanTradeGetResult.FansInfo fansInfo;
//    /**
//     * 三方APP用户id
//     */
//    private String outerUserId;
/**
 * 买家购买附言
 */
private String buyerMessage;
```

buyerId ， 买家用户编号。
TODO 6005 ： CRM 客户信息相关
buyerMessage ， 买家购买附言。



3.1.4 收货人信息

```
/**
 * 收货人的地址编号
 *
 * 当无需物流时，该字段为零
 */
private Integer receiverAddressId;
/**
 * 收货人的姓名
 */
private String receiverName;
/**
 * 收货人的地区编号
 */
private Integer receiverPlace;
/**
 * 收货人的详细地址
 */
private String receiverAddress;
/**
 * 收货人的邮编
 */
private String receiverZip;
/**
 * 创建交易时的物流方式。
 *
 * 取值范围：
 * 1：express（快递）
 * 2：fetch（到店自提）
 * 3：local（同城配送）
 */
private Integer shippingType;
/**
 * 卖家发货时间
 */
private Date consignTime;
/**
 * 买家签收时间
 */
private Date signTime;
```

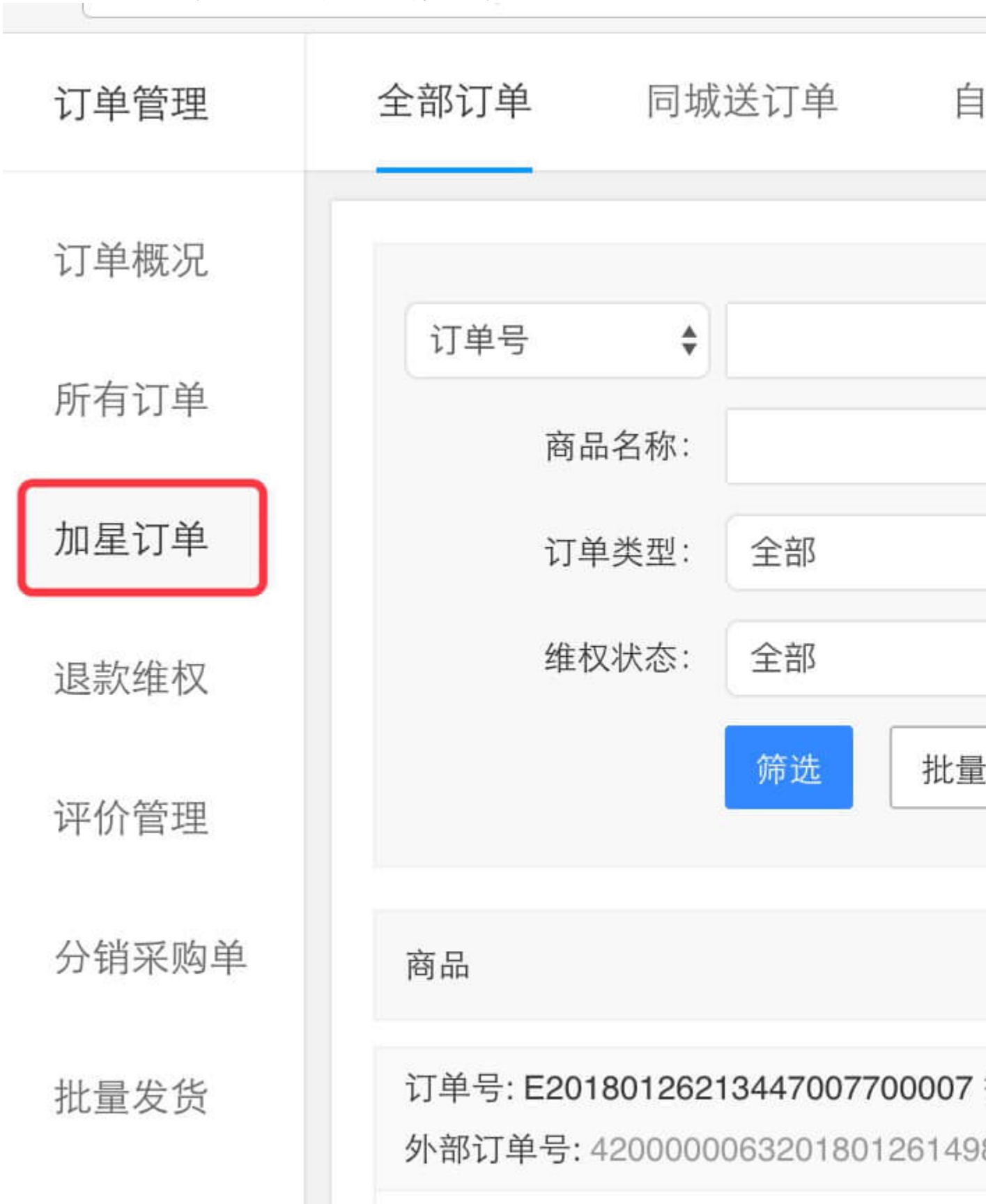
receiverAddressId ，收货人的地址编号。
receiverName 等等，冗余交易的收货地址信息，避免买家修改收货人地址编号对应的地址信息。
shippingType ，不同的交易类型，有不同的物流方式。
consignTime ，卖家发货时间。若有多种商品需要多次发货，全部发货后，才设置卖家发货时间字段。
signTime ，买家签收时间。多次发货，买家只确认收货一次。

3.1.5 卖家信息

```
/**
 * 卖家对该交易的备注
 */
private String tradeMemo;
/**
```

```
 * 卖家备注星标。
 *
 * 取值范围 1、2、3、4、5;
 * 如果为0，表示没有备注星标
 */
private Integer sellerFlag;
```

tradeMemo ，卖家对交易增加备注。注意，一个订单可能会添加多次备注，需要根据业务需要记录操作日志或者订单备注历史日志。
sellerFlag ，标记订单星标标识，用于卖家筛选订单。



3.1.6 支付信息

```
/**
 * 支付流水号（支付单号）
 */
private String transactionTid;
/**
 * 买家付款时间
 */
private Date payTime;
/**
 * 支付类型。
 *
 * 取值范围：
 * 1-WEIXIN（微信自有支付）
 * 2-WEIXIN_DAIXIAO（微信代销支付）
 * 3-ALIPAY（支付宝支付）
 * 4-BANKCARDPAY（银行卡支付）
 * 5-PEERPAY（代付）
 * 6-CODPAY（货到付款）
 * 7-BAIDUPAY（百度钱包支付）
 * 8-PRESENTTAKE（直接领取赠品）
 * 9-COUPONPAY（优惠券/码全额抵扣）
 * 10-BULKPURCHASE（来自分销商的采购）
 * 11-MERGEDPAY（合并付货款）
 * 12-ECARD（有赞E卡支付）
 * 13-PREPAIDCARD（储值卡支付）
 * 14-MARKPAY （标记支付）
 * 15-OFCASH（现金支付）
 */
private Integer payType;
/**
 * 外部交易编号。
 *
 * 比如，如果支付方式是微信支付，就是财付通的交易单号
 */
private String outerTid;
```

`transactionTid` ，支付单号。例如：180124230146000002 。

- 在买家下单时，创建交易的同时，调用支付中心，创建支付单。通过这样的方式，进行支付界面时，在支付单中可以读取到支付的内容信息，从而解耦。
- 从 API 上看返回的 String 类型，从数据例子上看，可能是 SnowFlake 的分布式 ID 算法生成的 Long 类型。胖友实际设计，可以根据自己需要做调整。

`payment` ，实付金额。笔者统一整理在了 「 3.1.9 价格信息 」 。

如下字段，通过支付成功后回调设置：

- `payTime` ，买家付款时间。
- `payType` ，支付类型。实际我们在做设计时，可以对一些操作方式做进一步细化，例如支付宝有支付宝 App 支付、支付宝手机网站支付、支付宝电脑网站支付等等。详细参见：《pint++ —— 支付渠道属性值》 。
- `outerTid` ，外部交易编号。比如，如果支付方式是微信支付，就是财付通的交易单号。

3.1.7 退款维权信息

在 《数据库实体设计 —— 交易（2.5）之退款维权》 详细分享。

3.1.8 优惠信息

// TODO 6008 优惠信息

3.1.9 价格信息

```
/**
 * 商品总价（商品价格乘以数量的总金额）。单位：分
 */
private Integer totalFee;
/**
 * 运费。单位：分
 */
private Integer postFee;
/**
 * 交易优惠金额（不包含交易明细中的优惠金额）。单位：分
 *
 * 【不包括】例如，购买的商品参加限制折扣活动 https://help.youzan.com/qa#/menu/2189/detail/919?_k=00rukd
 * 【包括】】另外，购买的商品使用优惠券 https://help.youzan.com/qa#/menu/2185/detail/915?_k=lih1k9
 */
private Integer discountFee;
/**
 * 实付金额。单位：分
 */
private Integer payment;
/**
 * 交易完成后退款的金额。单位：分
 */
private Integer refundedFee;
```

`totalFee` ，商品总价。该字段通过 TradeOrder 的 `payment` 求和计算。
`postFee` ，运费总价。该字段通过 TradeOrder 的商品的运费价格求和计算。
`discountFee` ，交易优惠金额。注意，TradeOrder 的 `discountFee` 。是否有些难以理解？我们来看两种场景（假设运维价格为零）：

- 购买的商品参加折扣活动，原价 100 元，折扣价 10 元。那么数据如下（我们会看到折扣活动跟着 TradeOrder 走）：
  - TradeOrder ： `totalFee = 100` ， `discountFee = 90` ， `payment = 10` 。
  - Trade ： `totalFee = TradeOrder.payment = 10` ， `discountFee = 0` ， `payment = 10` 。
- 购买的商品使用优惠券，在上面例子的基础上，优惠券打 2 折。那么数据如下（我们会看到优惠券跟着 Trade 走）：
  - TradeOrder ： `totalFee = 100` ， `discountFee = 90` ， `payment = 10` 。
  - Trade ： `totalFee = TradeOrder.payment = 10` ， `discountFee = 2` ， `payment = 8` 。

`payment` ，实付价格。计算公式为 `payment = totalFee + postFee - discountFee` + 涨价或减价。涨价或减价在 「3.2 TradeAdjustFee」 分享。
`refundedFee` ，退款金额。

3.2 TradeAdjustFee

商家在后台可以调整未付款的交易的价格：

涨减价格。注意，涨减的不是商品的价格，而是最终实付的价格。
调整运费价格

TradeAdjustFee ，交易改价信息。

```
/**
 * 交易编号 {@link Trade#id}
 */
private Integer id;
/**
 * 涨减价格
 */
private Integer payChange;
/**
 * 运费调价差值
 */
private Integer postChange;
/**
 * 更新时间
 */
private Date updateTime;
```

id ，交易编号。通过该字段和 Trade 1:1 关联。该表只记录最后一次的修改结果，如果胖友需要记录每次的调整，可以增加日志表。
payChange ，涨减价格。
postChange ，运费调整差价。注意，是差价。修改后，Trade.postFee 会改成运费价格，而 postChange 记录和原有价格的差价。
updateTime ，更新时间。

3.3 TradeOrder

TradeOrder ，交易明细。
交易下的每种商品生成一条交易明细记录。

字段较多，我们进行简单的切块。

3.3.1 基础字段

```
/**
 * 交易明细编号。
 */
private Long id;
/**
 * 交易编号。{@link Trade#id}
 */
private String tid;
/**
 * 店铺编号
 */
private Integer shopId;
/**
 * 交易明细主状态
 *
 * {@link Trade#status} 一致
```

```
 */
private Integer status;
```

id ， 交易明细编号。
tid ， 交易编号，指向 Trade.id 。
shopId ， 店铺编号。
status ， 交易明细主状态，和 Trade.status 一致。
    - 状态的设计是非常值得探讨，期盼胖友的讨论。

3.3.2 商品信息

```
/**
 * 商品编号
 */
private Integer itemId;
/**
 * 商品标题
 */
private String title;
/**
 * 商品主图片地址
 */
private String picPath;
/**
 * 商品类型。
 *
 * 0：自营；
 * 10：分销商品；
 */
private Long goodsType;
/**
 * 商品类型 {@link cn.iocoder.doraemon.itemgroup.item.entity.Item#itemType}
 */
private Integer itemType;
/**
 * Sku 编号
 */
private Integer skuId;
/**
 * SKU的值，即：商品的规格。如：机身颜色:黑色;手机套餐:官方标配
 */
private String skuPropertiesName;
/**
 * 商品价格。单位：分
 */
private Integer price;
/**
 * 商品购买数量
 */
private Integer num;
/**
 * 交易明细中买家留言的数据结构 {@link cn.iocoder.doraemon.itemgroup.item.entity.Item#messages}
 */
private String buyerMessages;
/**
 * 商品快照编号 {@link cn.iocoder.doraemon.itemgroup.snapshot.entity.ItemSnapshot#id}
 */
```

```
private String snapshotId;
```

itemId ， 商品编号，指向 Item.id 。

冗余商品字段：

- title ， 商品标题。
- picPath ， 商品主图片地址。
- goodsType ， 商品类型。
- itemType ， 商品类型。

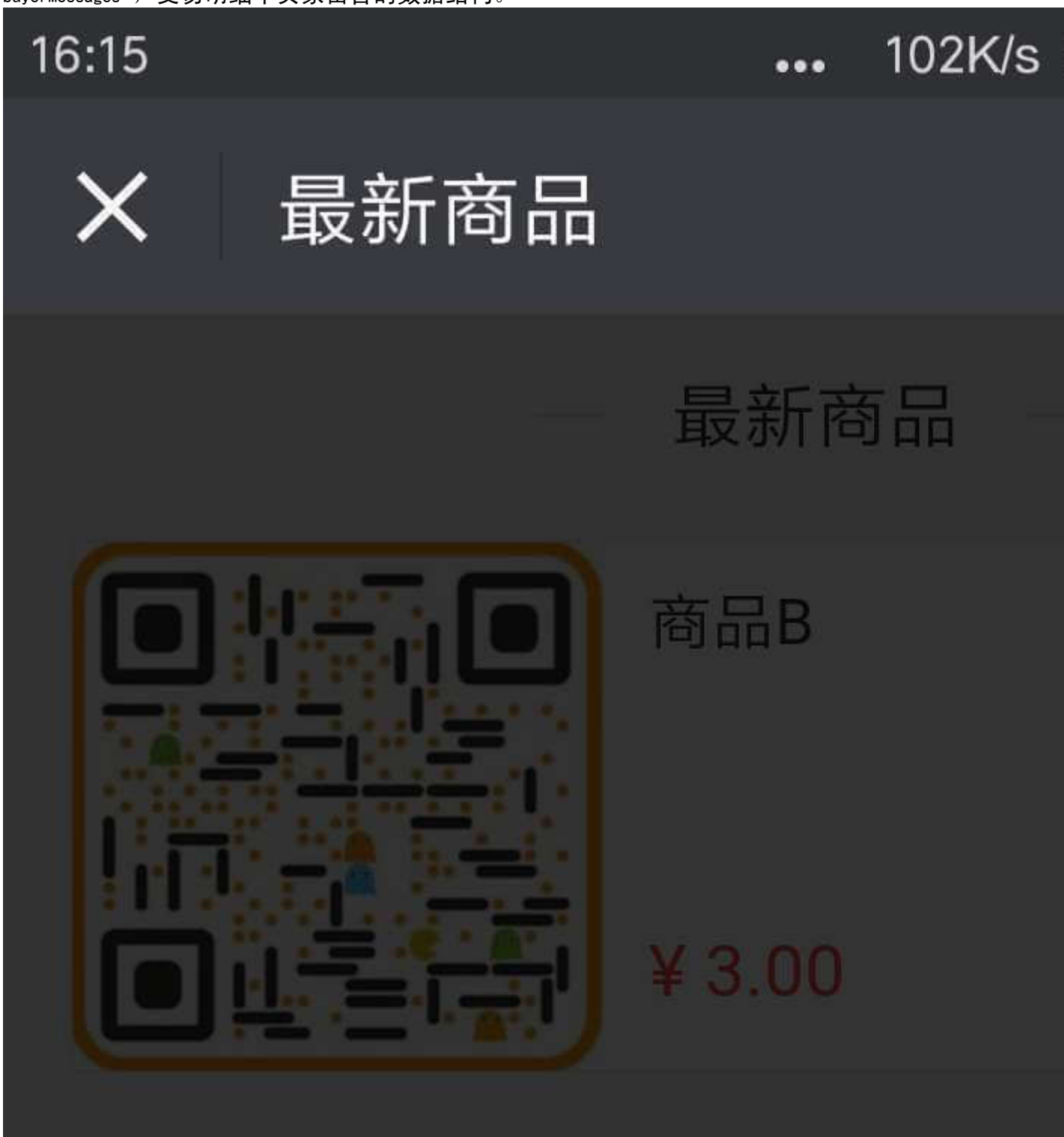
skuId ， 商品 SKU 编号，指向 ItemSku.id 。

冗余商品 SKU 字段：

- skuPropertiesName ， SKU的值，即：商品的规格。如：机身颜色:黑色;手机套餐:官方标配。
- price ， 商品价格。

num ， 购买数量。

buyerMessages ， 交易明细中买家留言的数据结构。



snapshotId ， 商品快照编号，指向 ItemSnapshot.id 。

3.3.3 退款维权

// TODO 6006：退款维权

3.3.4 优惠信息

// TODO 6008 优惠信息

3.3.5 价格信息

```
/**
 * 商品总价（商品价格乘以数量的总金额）。单位：分
 */
private Integer totalFee;
/**
 * 交易明细内的优惠金额。单位：分
 */
private Integer discountFee;
/**
 * 实付金额。单位：分
 */
private Integer payment;
/**
 * 退款金额。单位：分
 */
private Float refundedFee;
```

总体和 「3.1.9 价格信息」 类似。
totalFee ， 商品总价。该字段通过 price * num 计算。
discountFee ， 订单优惠金额。
payment ， 实付价格。计算公式为 payment = discountFee - discountFee 。
refundedFee ， 退款金额。

3.3.6 物流

```
/**
 * 是否允许发货
 */
private Boolean allowSend;
/**
 * 是否已经发货
 */
private Boolean isSend;
```

4. API

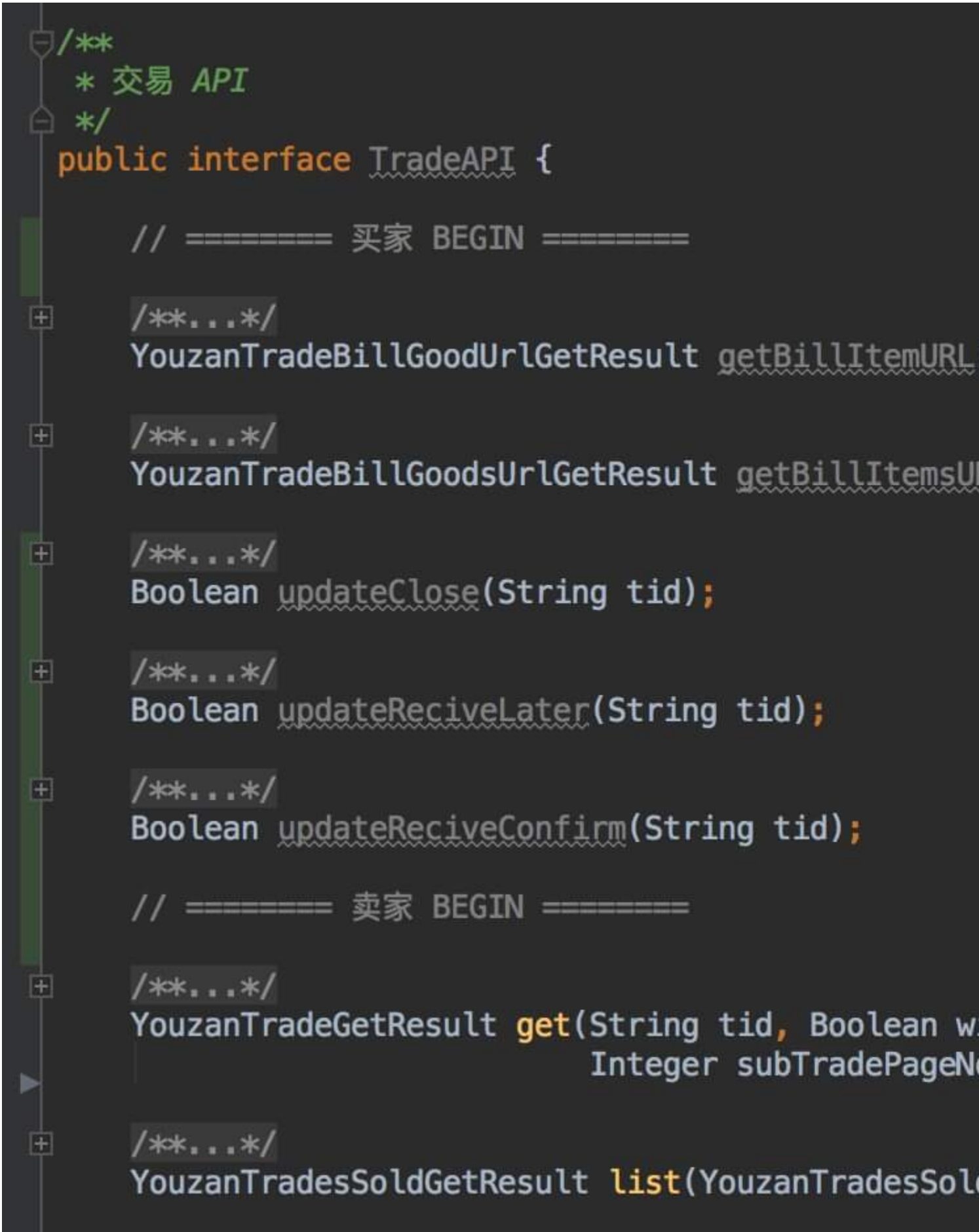
基于如下整理 API 类。

有赞云提供的商家管理订单API

有赞云提供的买家下单API

4.1 TradeAPI

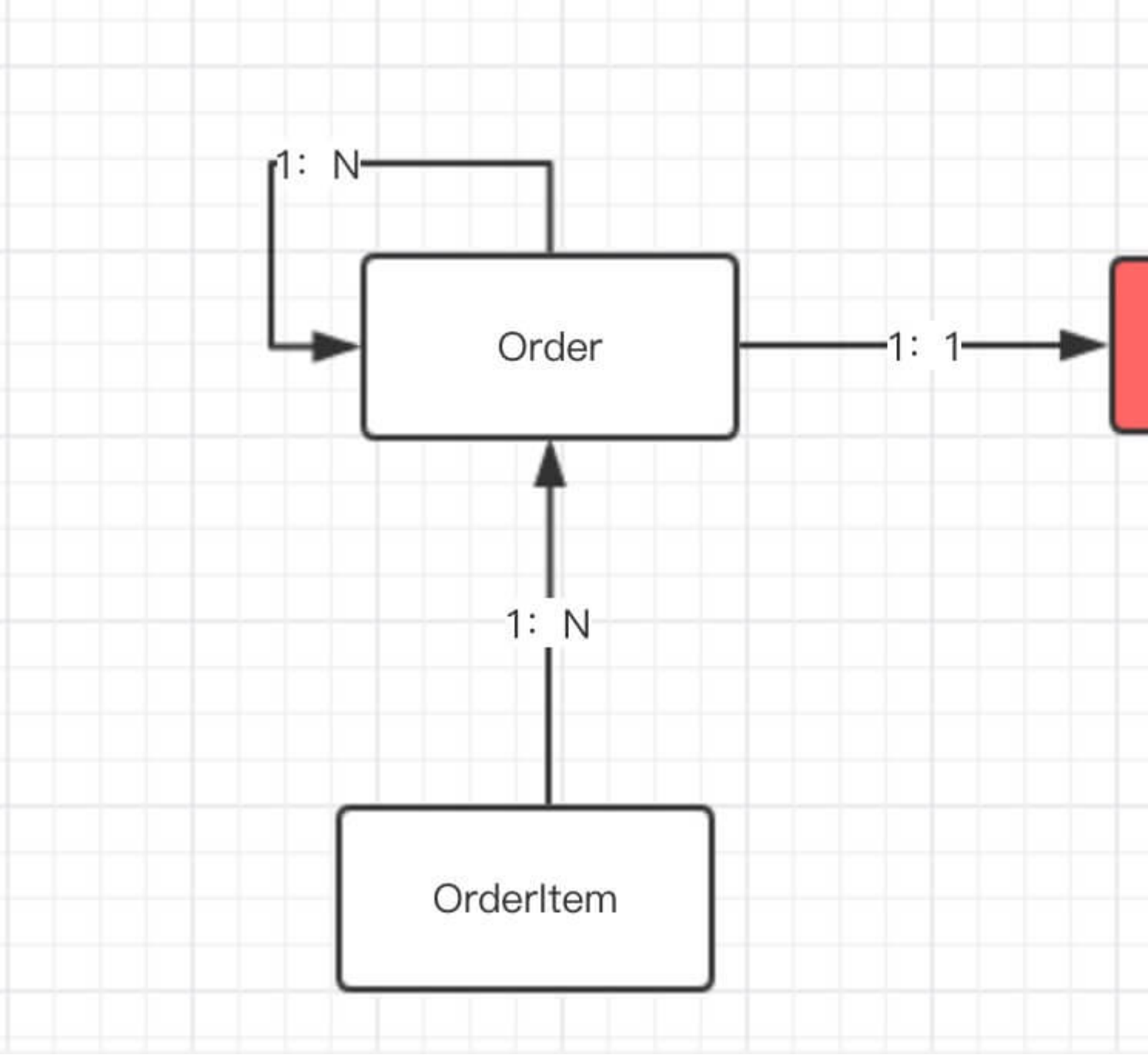
TradeAPI ，交易 API 。



友情提示，实际场景下，API 会分拆成买家和卖家两个 API 类。

5. 京东订单结构

基于 订单API —— 检索单个订单信息 和 订单API —— 订单SOP出库 猜想。



Order ： 物流单 ： OrderItem = 1 ： 1 ： N 。
- 每个 Order 可以有多个 OrderItem ，即多个商品。
- 每个 Order 发货绑定一个物流单。

用户下单时候，Order 基于商家、物流等拆单，子 Order 通过 parentOrderId 指向父 Order 。
听老徐说，由于下单时就会基于物流拆单，实际会存在库存不够等等原因进行调仓，即实际发货时，一个 Order 也可能再拆成多个 Order 。

666. 彩蛋

断断续续折腾了一周的时间，主要在如下点卡了比较久：

> Trade 和 TradeOrder 的关系，因为一度以为有拆单的逻辑。
> 主状态，期盼胖友能够探讨一下。

文章目录

回到首页