

**ooMall系统**

**详细设计文档**

**2-3：在线互啄**

**（模块选择：Share、Freight、Collect、Cart、Address、Picture）**

24320172203200 孙鲁喆

24320172203123 董怡帆

24320172203144 黄烨钒

24320172203140 黄祺尧

24320172203199 苏 畅

2019.12.22

目录

[一、组件图 3](#_Toc27944750)

[二、部署图 4](#_Toc27944751)

[三、模块详细设计 5](#_Toc27944752)

[3.1运费模块 5](#_Toc27944753)

[3.1.1模块的需求 5](#_Toc27944754)

[3.1.2模块的API说明 7](#_Toc27944755)

[3.1.3模块的静态模型 11](#_Toc27944756)

[3.1.4模块的动态模型 13](#_Toc27944757)

[3.1.5模块的设计说明 17](#_Toc27944758)

[3.2分享模块 18](#_Toc27944759)

[3.2.1模块的需求 18](#_Toc27944760)

[3.2.2模块的API说明 19](#_Toc27944761)

[3.2.3模块中对象的状态机图 21](#_Toc27944762)

[3.2.4模块的静态模型 21](#_Toc27944763)

[3.2.5模块的动态模型 21](#_Toc27944764)

[3.2.6模块的设计说明 24](#_Toc27944765)

[3.3收藏模块 26](#_Toc27944766)

[3.3.1模块需求分析 26](#_Toc27944767)

[3.3.2模块API设计 26](#_Toc27944768)

[3.3.3静态模型 27](#_Toc27944769)

[3.3.4模块设计说明 28](#_Toc27944770)

[3.4购物车模块 29](#_Toc27944771)

[3.4.1模块需求分析 29](#_Toc27944772)

[3.4.2模块API说明 29](#_Toc27944773)

[3.4.3静态模型 31](#_Toc27944774)

[3.4.4动态模型 33](#_Toc27944775)

[3.4.5模块设计说明 34](#_Toc27944776)

[3.5地址模块 35](#_Toc27944777)

[3.5.1模块需求分析 35](#_Toc27944778)

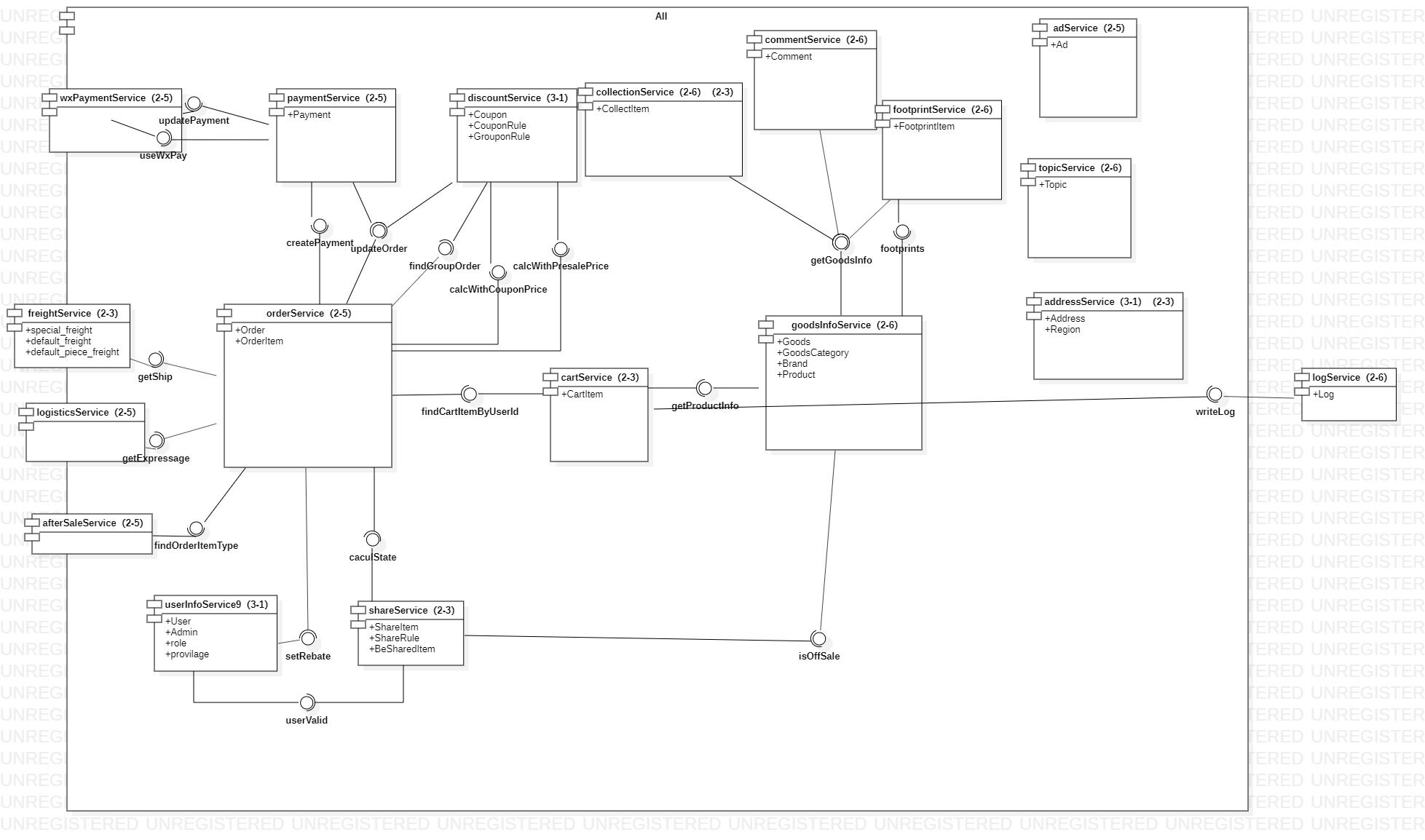
[3.5.2模块API说明 35](#_Toc27944779)

[3.5.3静态模型 37](#_Toc27944780)

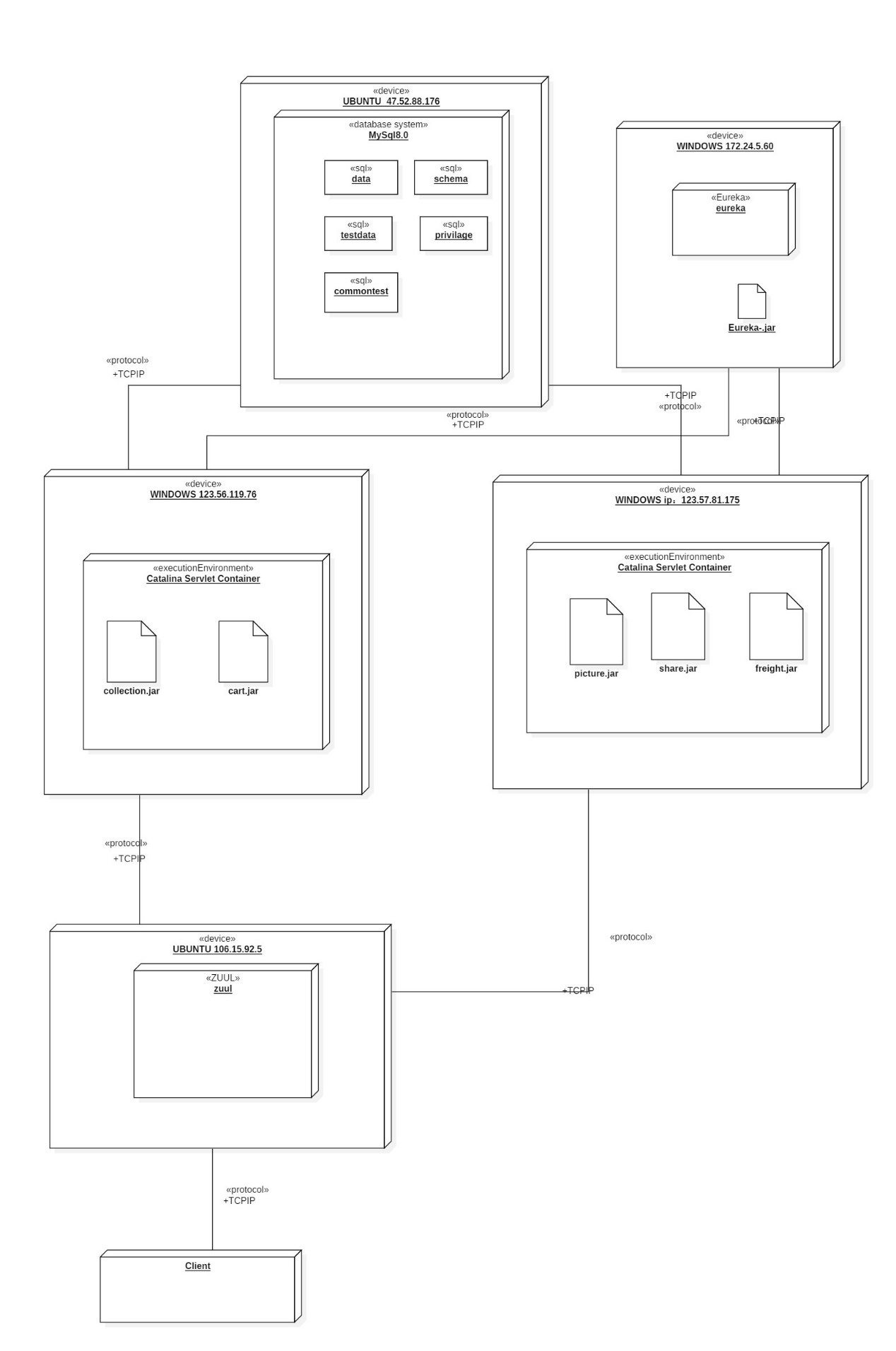
[3.5.4动态模型 39](#_Toc27944781)

[3.5.5模块设计说明 39](#_Toc27944782)

# 一、组件图



# 二、部署图



# 三、模块详细设计

## 3.1运费模块

### 3.1.1模块的需求

实现运费模板的功能，运费模板分为默认运费模板和单品运费模板。

* + - * 1. 运费模板的管理：

1. 默认运费模板

默认运费模板是按照订单货品的总重量以及收件地区来设定订单的运费。

主要实现以下四种功能：

获取全部的默认运费模板（根据页码及页容量）；

新增默认运费模板

修改默认运费模板

删除默认运费模板

1. 默认单品运费模板

单品运费模板是针对体积特殊的商品而单独定制的运费模板，单品运费模板按照件数来计算运费。

主要实现以下四种功能：

获取全部的默认单品运费模板（根据页码及页容量）；

新增默认单品运费模板

修改默认单品运费模板

删除默认单品运费模板

1. 特殊商品运费计价规则

主要实现以下五种功能：

获取全部的特殊运费规则（根据页码及页容量）

新增特殊运费规则

获得单条特殊运费规则

修改特殊运费规则

删除特殊运费规则

（其中，以上的所有的操作都需满足表中相应属性的约束。）

### 3.1.2模块的API说明

get

/defaultPieceFreights

默认运费比率表

Requestparam(page) Requestparam(limit)

Return List<DefaultPieceFreightPo>

post

/defaultPieceFreights

新增默认比率表

Requestbody（DefaultPieceFreightPo）

Return DefaultPriceFreightPo(新建的对象)

put

/defaultPieceFreights /{id}

修改默认比率

Pathvariable （id） Requestbody（DefaultPieceFreightPo）

Return DefaultPieceFreightPo(修改后的对象)

delete

/defaultPieceFreights /{id}

删除默认比率

Pathvariable （id）

GET

/defaultFreights

获取默认运费规则

RequestparampageRequestparamlimit

return List<defaultFreightPo>

POST

/defaultFreights

新增默认运费规则

requestbodydefaultFreightPo

return defaultFreightPo

PUT

/defaultFreights/{id}

修改默认运费规则

pathvariable (id) requestbody (defaultFreightPo)

return defaultFreightPo

DELETE

/defaultFreights/{id}

删除默认运费规则

Pathvariable (id)

GET

/ specialFreights

获取特殊运费规则

RequestparampageRequestparamlimit

return List<specialFreights>

POST

/specialFreights

新增特殊运费规则

requestbody specialFreight

return specialFreight

GET

/specialFreight/{id}

获得单条特殊运费规则

Pathvariable id

Return specialFreght

PUT

/specialFreights/{id}

修改特殊运费规则

pathvariable (id) requestbody (specialFreight)

return specialFreight

DELETE

/specialFreights/{id}

删除特殊运费规则

Pathvariable (id)

内部API：

POST

/freightPrice

根据订单计算运费

RequestBody order

Return price

GET

/inner/specialFreights/{id}

通过ID查找特殊运费模板

PathVariable Integer id

POST

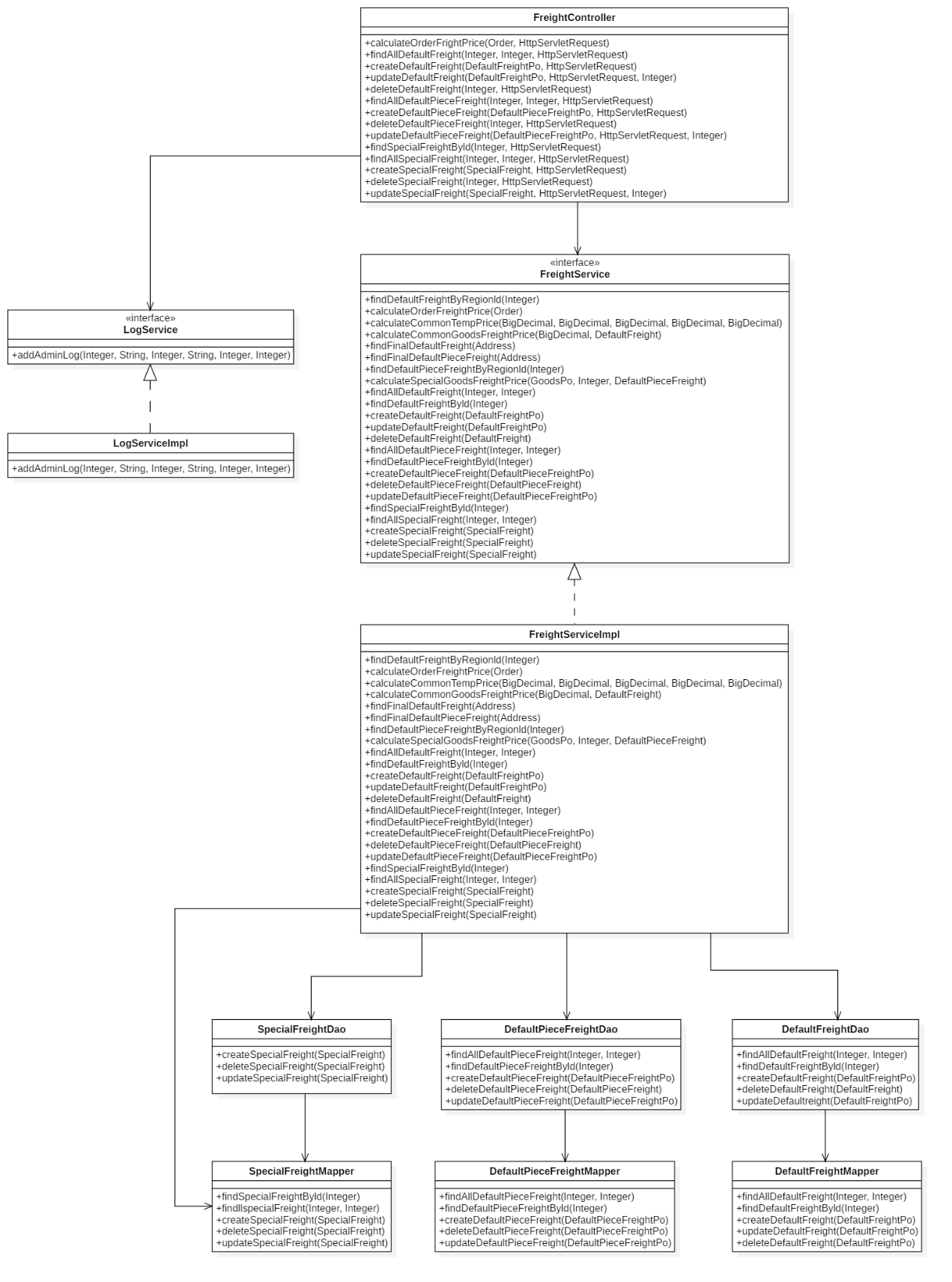
/inner/address

通过省市县名称查找省市县ID

RequestBody address

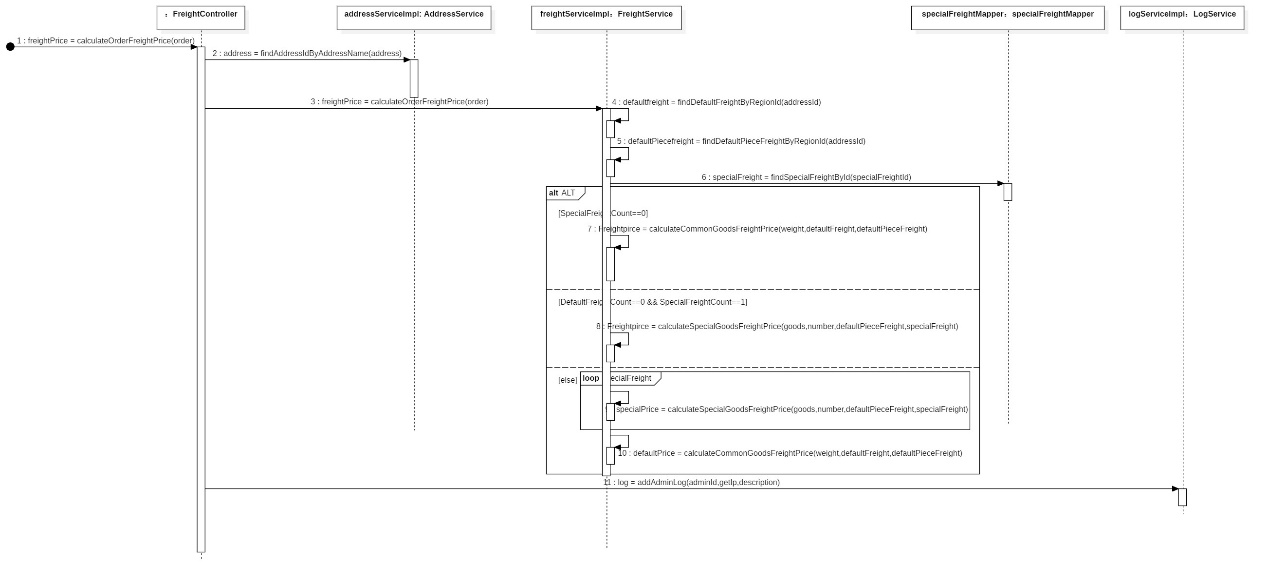
Return address

### 3.1.3模块的静态模型

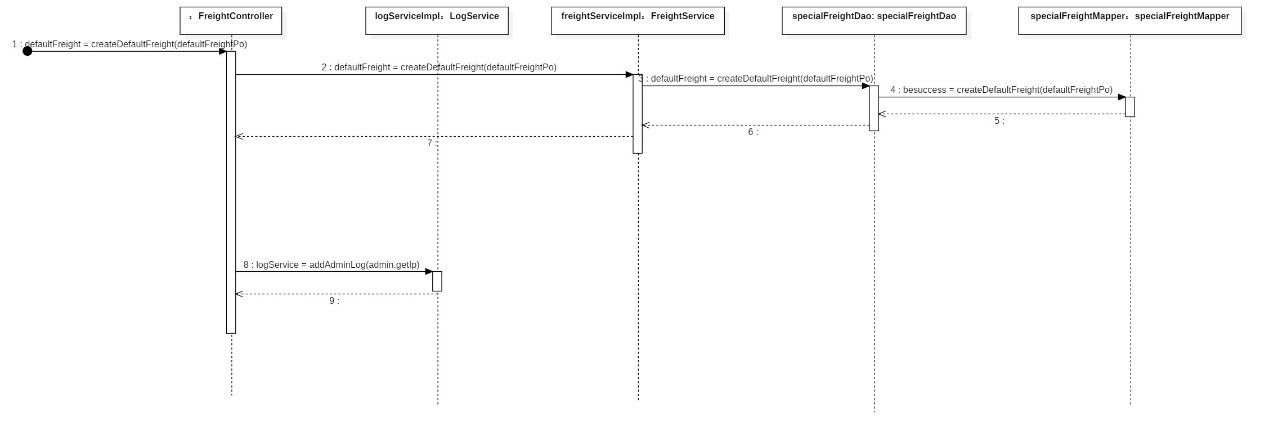
类图

### 3.1.4模块的动态模型

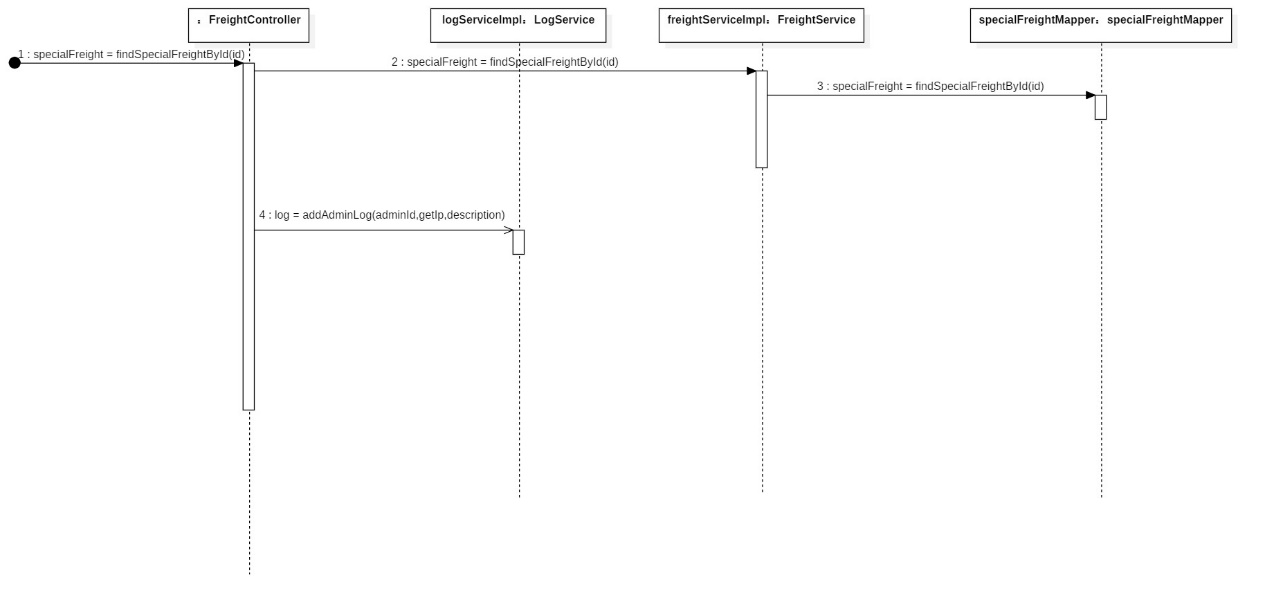
1. 计算运费



2. 创建默认运费模板



3. 通过ID查找特殊运费模板



### 3.1.5模块的设计说明

1、分层设计说明

Domain：Domain对象共有SpecialFreight、DefaultFreight、DefaultPieceFreight、DefaultFreightPO,DefaultPieceFreightPO,其中DefaultFreight、DefaultPieceFreight进行了重写，采用装饰器设计模式，内部实例PO对象，重写get/set方法，将String类型属性destination解析为List<Integer>使用，其余对象沿用标准组设计。

Mapper：使用myBatis映射到数据库，主要以XML映射为主。

Dao：对数据进行处理，为每个经过的对象装载修改时间，返回值类型转换、PO对象转换实体对象等。

FreightService：主要逻辑的调用和处理，遵循代码下沉原则逻辑较少，包含运费计算逻辑实现。

FreightController：请求的获取，数据有效性判断，异常处理返回对应错误码。

2、功能设计说明

计算运费操作：

当订单的价格未达到包邮门槛时，按照运费模板计算运费。

调用地址服务，从而获取包含了provinceId,cityId,countyId的地址对象，通过Address的 provinceId,cityId,countyId与数据库中default\_freight所有模板的目的地字段相匹配，首先进行provinceId的匹配，若未成功匹配则进行cityId的匹配，若未成功匹配则进行countyId的匹配，找到地址对应的默认运费或默认单件运费模板。

根据订单中的商品种类（是否为特殊商品）、商品属性（重量）、商品购置件数以及收货地址进行相应的订单运费的计算，主要分为以下三种情况：

a. 订单中仅包含普通商品，根据默认运费模板，按照其重量之和计算运费；

b. 订单中仅包含特殊商品，根据默认单品运费模板及商品对应的特殊运费模板，按照其件数计算运费；

c. 订单中同时包含单品运费模板商品和默认运费模板商品，分别使用订单中包含的所有运费模板，对所有商品计算一次运费，取最大的值作为最终运费。

## 3.2分享模块

### 3.2.1模块的需求

* + - * 1. 分享模块需要完成增加某个用户的被分享明细的需求。

当用户点击了其它用户分享给他的连接，或通过他人分享的二维码浏览某商品时，需要增加一条被分享记录明细，方便之后进行返点的计算。

* + - * 1. 分享模块需要完成用户或管理员查看某一特定商品的分享规则的需求。
        2. 分享模块需要完成管理员删除某一特定商品的分享规则的需求。（当对应商品被删除，此时分享规则应被级联删除）
        3. 分享模块需要完成计算返点的需求。
        4. 分享模块需要完成管理员根据新建一条分享规则的需求。
        5. 分享模块需要完成管理员根据删除特定id的分享规则的需求。
        6. 分享模块需要完成修改特定分享规则的需求。

### 3.2.2模块的API说明

外部API：

GET

/goods/{id}/shareRules

查看一个商品对应的分享规则（一个商品只有一个分享规则）

Pathvariable {id}

return shareRules

(这边等界面,应该在商品那边设计一个按钮查看对应分享规则)

POST

/shareRules

新建一条分享规则

Requestbody (shareRulesPo)

Return shareRulesPo(新建的对象)

DELETE

/shareRules/{id}

删除一条分享规则

Pathvariable id

PUT

/shareRules/{id}(修改了，加了/id)

修改一条分享规则

Pathvariable （id） Requestbody（shareRulesPo）

ReturnsharerulesPo（修改的对象）

POST

/beSharedItems

增加某个用户的被分享表（用户被分享时系统为其创建被分享表，用户id在状态里，分享者id在二维码里）

Requestbody (besharedItems)

ReturnbeshareItems

内部API

POST

/rebate

计算返点

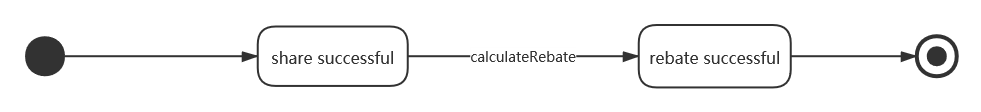
Requestbody（orders）

Return Map<Integer,Integer>rebateMap

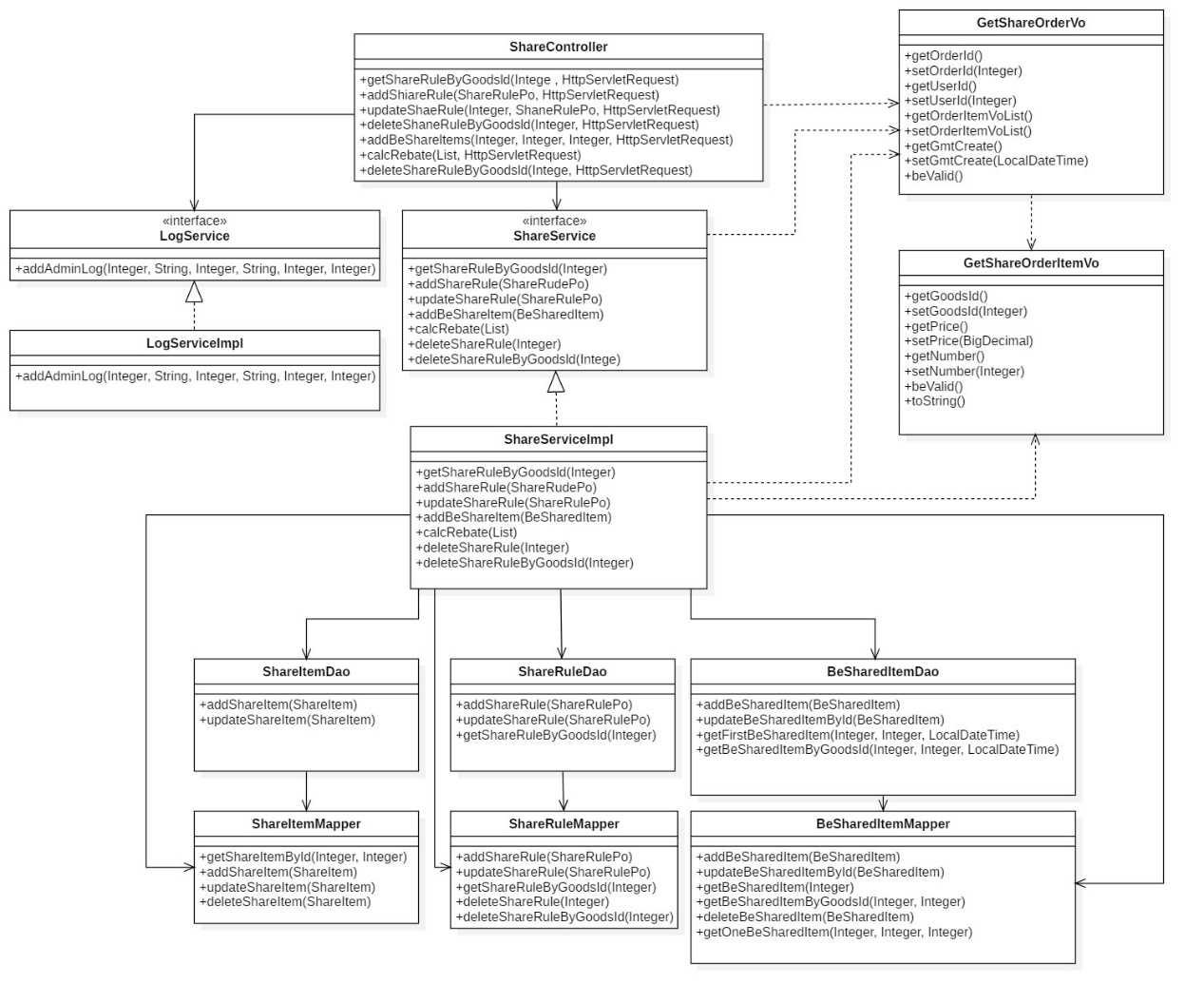
DELETE

/goods/{id}/shareRules

### 3.2.3模块中对象的状态机图

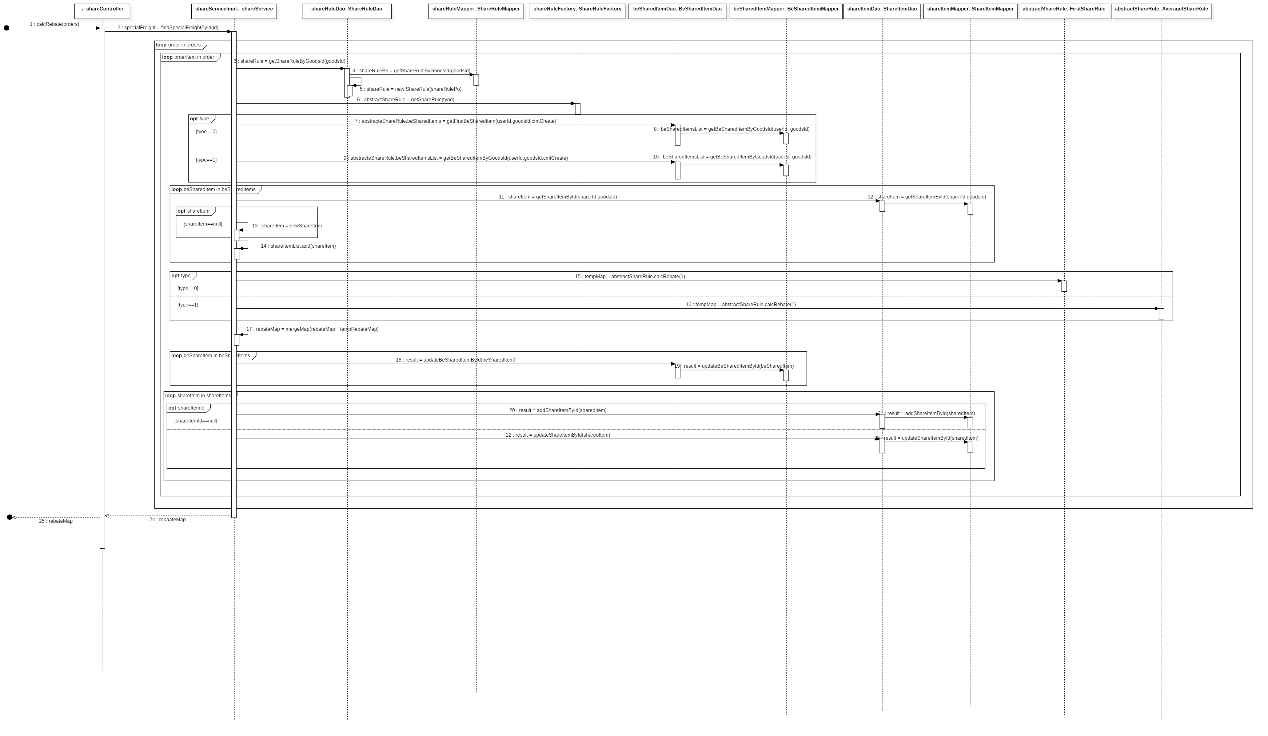


### 3.2.4模块的静态模型



### 3.2.5模块的动态模型

计算返点



### 3.2.6模块的设计说明

1、分层设计说明

Domain：Domain对象共有ShareRule、ShareRulePo、BeSharedItem、ShareItem，AbstractShareRule、FirstShareRule、AverageShareRule。后三个Domain对象是为了多态实现根据不同的分享规则对返点进行计算。同时，将其封装，也方便了之后的重构、复用。其中对ShareRule重写构造方法，以及增加了一个方法jsonToStrategyList，将String类型属性shareLevelStrategy解析为List<Strategy>使用，其余对象沿用标准组设计。

Mapper：使用myBatis映射到数据库，主要以XML映射为主。

Dao：对数据进行处理，为每个经过的对象装载修改时间，返回值类型转换、PO对象转换实体对象等。

ShareService：主要逻辑的调用和处理，遵循代码下沉原则逻辑较少，包含运费计算逻辑实现。

ShareController：请求的获取，数据有效性判断，异常处理返回对应错误码。

1. 功能设计说明

计算返点操作：

从OrderService模块获得一个List<GetShareOrderVo>，GetShareOrderVo包含userId、gmtCreate、List<GetShareOrderItemVo>等属性，包含有用于计算的所有需要的数据。

遍历每个GetShareOrderVo中的每个GetShareOrderItemVo，然后取出这个OrderItem对应的goodsId，根据goodsId取出对应的shareRule，根据shareRule其中的shareType，利用工厂类，动态产生子类。

根据order中的userId与goodsId，找到用户的被分享表，与对应分享者的shareItem，然后根据不同的分享策略，对用户计算返点。目前主要有两种策略：最先分享，和平摊返点。这两种策略本质都是一样的，可以视为一个用户获得了n件商品（平摊返点，即n/平摊人数件商品），均价x元的返点。

最终合并成一个HashMap返回给Order模块。

## 3.3收藏模块

### 3.3.1模块需求分析

1、加入收藏夹

2、删除对应id的收藏夹明细

3、获取用户收藏夹

### 3.3.2模块API设计

GET

/collections

参数：

page: Integer 第几页

limit: Integer 一页多少

返回值：List<collectItem>

用户查看自己的收藏（userid在token，不是参数）

POST

/collections

参数：Requestbody(collectItemPo)

前端可传参数：

goodsId

userId

返回值：collectItemPo

DELETE

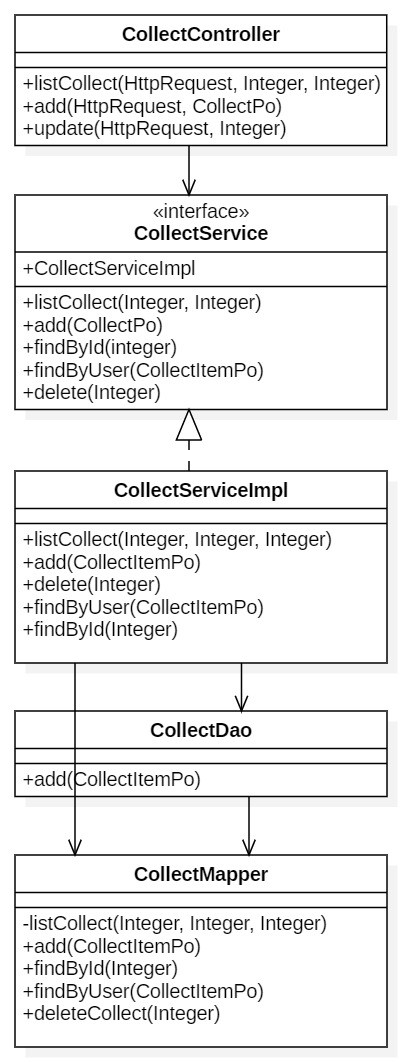
/collections/{id}

参数：Pathvariable(id)

返回值：无

### 3.3.3静态模型

#### 类图

### 3.3.4模块设计说明

1、加入收藏夹

需要调用商品服务，得到goodsId对应的GoodsPo，并进行封装返回

2、查看用户收藏夹

根据当前登录的用户id获得相应的CollectItem的list对象

3、删除收藏夹记录

需要检查对应id 的CollectItem，如果在CollectItem表中已经不存在则显示删除失败。还需要检查删除这个CollectItem的对象与id对应的userId是否是一致的。因为删除购物车是用户特有的，每个用户只能删除自己的购物车。这样做算是提供一层保障。

## 3.4购物车模块

### 3.4.1模块需求分析

购物车模块需求：

1、需要根据当前登录的用户id获取用户的购物车信息

2、将商品规格添加到购物车中，形成一条购物车明细，需要判断加入购物车的数量不能超过商品规格的安全库存量。如果购物车明细表中，该用户原本就存在该商品规格的一项纪录，则修改该条纪录为两次添加的商品规格数量之和，这个数量和不能超过安全库存。

3、点击“快速购买”，能在购物车明细表中自动生成一条购物车明细，以便提供给订单。快速订单中的商品规格如果原本就在购物车明细表中有存在，则用第二次添加的数量覆盖之前的数量

4、能修改购物车中一条纪录的商品规格，数量，是否勾选等

### 3.4.2模块API说明

Get:

/cartItems:

获取购物车信息

return list<cartItem>

Post:

/cartItems:

添加商品到购物车

Requestbody cartItemPo

Return cartItemPo

Post:

/fastAddCartItems

快速购买的url

RequestbodycartItemPo

Return cartItemPo

Put:

/cartItems/{id}:

修改某个购物车项信息

Pathvariable id Requestbody cartItemPo

Return cartItemPo

Delete

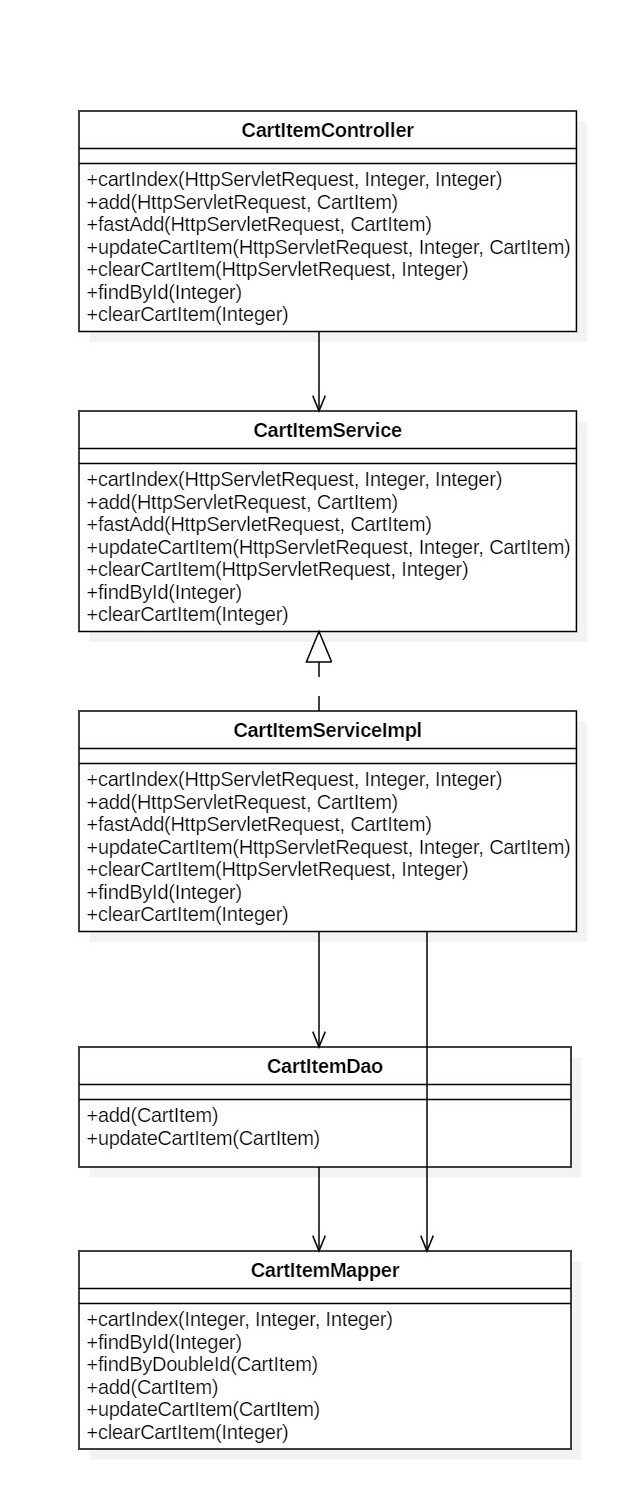
/cartItems/{id}:

删除一个购物车项

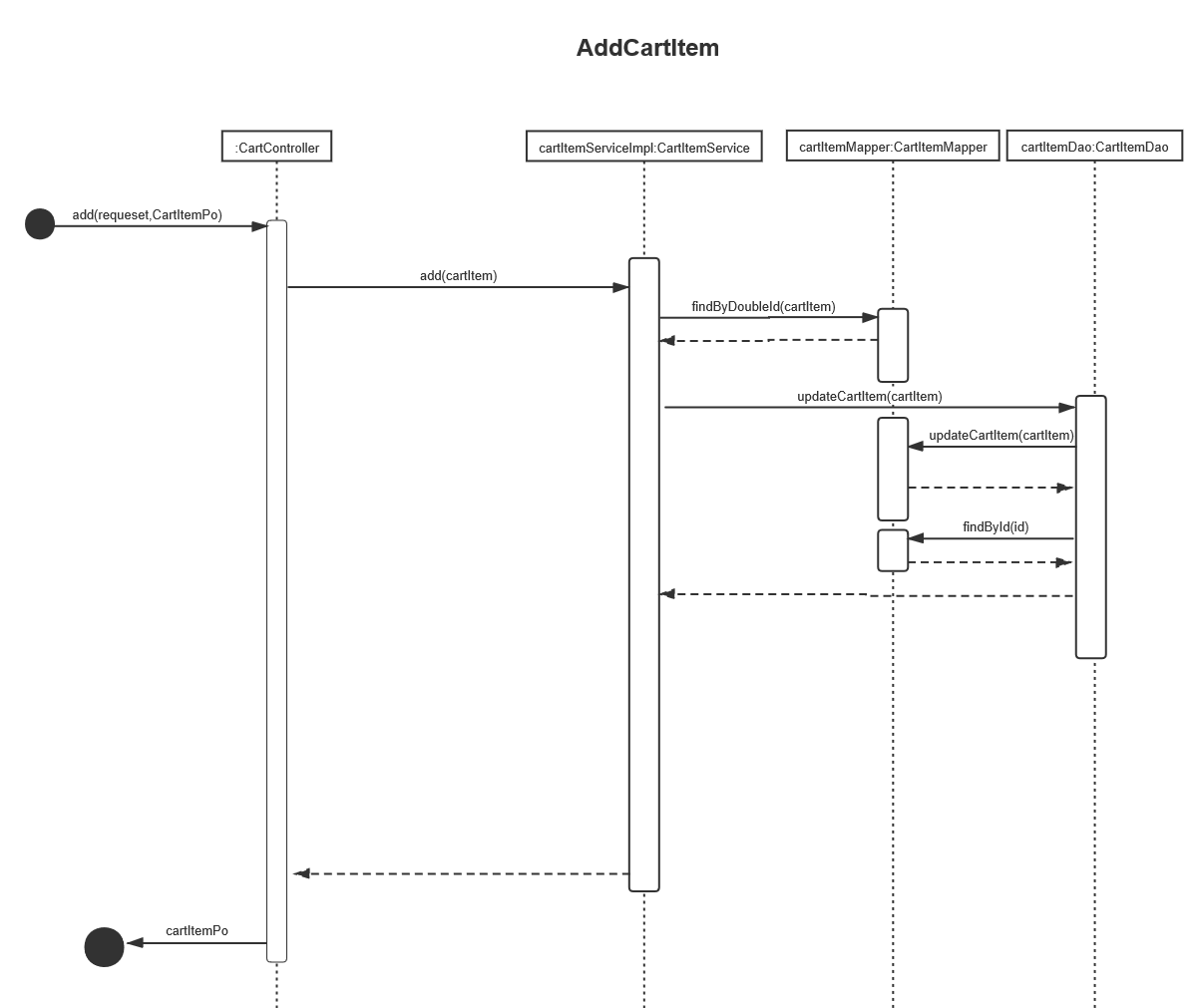
Pathvariable id

### 3.4.3静态模型

#### 类图



### 3.4.4动态模型



### 3.4.5模块设计说明

1、获取用户购物车

当CartItem的product属性，通过微服务调用查询，退出该商品规格以被逻辑上删除，这时依然会将该CartItem返回，其中product置为null。这样做是考虑到不能突然在界面中消失，会让客户感到疑问，并且如果以后该商品规格重新上架，则购物车中应该能重新下单该商品。

2、加入购物车

需要判断该product是否被逻辑删除，并且要判断加入购物车的数量是否超过商品规格的安全库存量。除去第一次把一个商品规格加入购物车是insert，之后每次都是在原来数量上增加，但是每次需要检查增加之后的数量是否满足库存限制。

3、立即购买

它与加入购物车类似，但是除去第一次把一个商品规格加入购物车是insert，之后每次都是覆盖之前的数量。其他要求与立即加入购物车功能一样。

4、删除对应id的CartItem

需要检查对应id 的CartItem，如果在CartItem表中已经不存在则显示删除失败。还需要检查删除这个CartItem的对象与id对应的userId是否是一致的。因为删除购物车是用户特有的，每个用户只能删除自己的购物车。这样做算是提供一层保障。

**3.5 图片上传模块**

**3.5.1模块需求分析**

实现上传图片功能

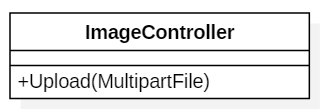
**3.5.2模块API设计**

POST  
/pics  
参数：@RequestParam MultipartFile

返回值：String url

**3.5.3静态模型**

类图



**3.5.4模块设计说明**

上传图片功能，由于该功能比较简单，只需要controller即可实现功能