# 1、数据架构

## 1.1 数据库ER模型

## 1.2 数据库逻辑模型

## 1.3 数据库物理模型

# 2、接口

到了后面工程初始化的时候，我会给大家演示的，如果在开发过程中需要调节或者是新增这个接口，应该怎么弄。

刚开始拉分支的时候，单独拉一个common-api的分支出来，就是包含所有的公共接口和api的。然后如果互相之间调用的接口要修改或者是新增接口库，那么可以在common-api分支里面调整接口，然后将common-api分支合并到每个人的feature分支上即可。

## 2.1 运费计算

LogisticsFacadeServcie

Double calculateFreight(OrderGoodsSkuDTO OrdergoodsSkuDTO)

# 3、开发架构

## 3.1 实现类图

设计模式这块，必须使用策略模式来开发运费计算这块；用代理模式，来代理远程的第三方物流接口

## 3.2 包设计

# 4、运行流程

## 4.1 运费模板管理

### 4.1.1 查询运费模板列表

支持筛选条件，可以分页查询

### 4.1.2 新增运费模板

在列表页的上面，有一个新增按钮，点击之后进入一个界面。

这个界面里，可以输入运费模板名称，然后选择运费模板类型，这里会根据选择不同的类型，下面给你显示出来不一样的操作UI组件。

比如说，如果选择了固定运费：让你设置一个固定运费是多少元，固定运费20元。

{“fixed\_freigt”: 20}

比如说，选择了满X元包邮：让你设置一个满多少元可以包邮，没达到这么多钱的话，固定运费就是多少钱。

{“threshold”: 100, “less\_than\_threshold\_freight”: 10}

比如说，选择了自定义规则之后：就是下面会出来多个UI组件，你可以加多个规则，每个规则：加一个规则，选择发货到哪些哪些省份，多少重量以内是多少钱；每增加多少g，增加多少钱。

[

{

“provinces”: “山西,陕西,甘肃,辽宁,黑龙江,吉林”,

“threshold”: “100g”,

“threshold\_freight”: 10,

“incr\_step”: “2g”,

“incr\_freight”: 2

},

{

“provinces”: “云南,广西,湖南,江西”,

“threshold”: “100g”,

“threshold\_freight”: 5,

“incr\_step”: “1g”,

“incr\_freight”: 1

}

]

输入一个备注

### 4.1.3 根据ID查询运费模板

根据ID将运费模板的数据查询出来在界面上回显即可

### 4.1.4 编辑运费模板

可以调整和修改运费模板的名称、类型、规则、备注，就ok了

## 4.2 运费计算

准备提交订单的确认页面上，就会根据每个商品的运费模板，计算出来每个商品对应的一个运费

订单中心会将订单的数据传送过来，然后这里会根据每个商品绑定的运费模板，计算出来每个商品对应的一个运费。包括一个订单的总运费。

就是说接收到订单条目之后，拿到商品id，去商品中心获取商品，就可以拿到这个商品绑定的运费模板id，就可以取到自己本地对应的那个运费模板的数据

看到这种场景，直接就上策略模式，如果你不用策略模式的话，可能代码会写成什么样子呢？

if (type == 1) {

// 200行代码

} else if(type == 2) {

// 300行代码

} else if(type == 3) {

// 1000行代码

}

if (type == 1) {

type1()

} else if(type == 2) {

type2()

} else if(type == 3) {

type3()

}

type1() {

}

type2() {

}

type3() {

}

Strategy strategy = StrategyFactory.get(type);

Double freight = strategy.calculate(orderItem);

如果是自定义规则这种比较复杂的形式，那么在用户选择了商品数量和目的地之后，就可以参照目的地的计算规则，然后算出来重量，根据重量就可以算出来对应的运费

运费的计算公式=首重费用+（实际邮寄重量-首重）/续重\*续重费用。以江浙沪皖同省跨市件为例，首重1kg，首重费用为10元，在50kg内，每增加1kg，则需要增加2元的运费。那么快递3.5kg的物品的运费=10+（3.5-1）/1\*2=16（运费需要向上取整，3.5Kg按4kg计算)

接收到一个请求之后，一个商品+运费模板ID，我们就可以获取到这个商品购买了几件，商品的单价是多少，这个商品的总价是多少，这个商品的单个毛重是多少，商品总毛重是多少。根据运费模板的规则，就可以将运费给计算出来。

## 4.3 获取物流单

在调度中心调度销售出库的时候，会针对这个订单中的商品，去跟第三方物流厂商的接口进行通信，告诉第三方物流厂商，我们要从哪个仓库里，发送哪些商品，每个商品的重量，长宽高的一些数据，下一个订单，让物流厂商自己去我们的仓库里，取货；或者是，我们是告诉物流厂商，下这么一个物流的订单，然后在仓库里，拣货打包之后，将物流单贴在包裹上，接着将贴了物流单的包裹交给仓库里跟物流厂商的对接人员，物流厂商的人就根据这个物流单号的信息，去进行发货了。

后面实际开发的时候再说，选用一个快递api的免费接口，带着大家来下电子面单就可以了

我们可以用一些第三方的物流api接口，然后的话，当然我们不能真实的跟人家进行通信，但是我们可以参照和按照人家的接口的标准，来写这个代码，给大家进行示范和演示。

跟第三方的api接口进行通信的时候，其实完全是可以用代理模式的

假设我们有5个业务场景，都需要跟第三方物流api接口进行调用，我们有一种比较low的选择：

1. 在5个业务场景的代码中，都比如说直接使用HttpClient去向第三方物流接口发起请求。特别恶心，特别不好，因为实际上跟远程接口进行请求，你要考虑一些事情，比如说请求参数的封装，一些特殊的http参数的设置，还有响应结果的处理，是否成功。如果你每个地方都做这么一些事情，就会导致很麻烦。
2. 我们如果有那种针对远程接口的一个调用的场景，可以来考虑用代理模式。远程接口的访问，rpc调用为经典的使用场景。

远程有个服务暴露了一个接口，比如说UserService.sayHello()

然后如果我们本地要访问远程的那个接口，可能需要处理一些网络通信的一些细节，才能调用到人家的UserService.sayHello()

rpc，就可以做到在这个本地给你一个代理类，你就不要care什么远程接口了，你就可以假装认为那个远程的接口就在自己的本地，你就调用本地的代理类的接口就ok了

UserService.sayHello() -> 你就调用自己本地的这个接口 -> 本地的UserService.sayHello() -> 网络通信调用远程的UserService.sayHello()

你现在在自己本地，你要调用远程的第三方服务厂商的接口，那个接口，现在可能是http，参数可能是json，也许以后会变成用xml，里面有很多通信的细节要考虑。比如说你现在是使用某个厂商的物流接口，比如说快递鸟api接口，走的是他的一套接口的规范和调用方式。但是如果后面又变成，使用快递100的接口，接口的规范和格式，都不一样了

如果你每个地方都自己去调用远程接口，处理一些通信细节，以后做一些修改的时候，就特别的麻烦。

你可以针对物流接口这块，专门在自己本地做一个代理类，你所有的业务代码，如果需要调用u物流接口的，都是直接走自己本地的代理类，然后你的代理类负责跟各种厂商api的通信，cover掉所有的通信细节

## 4.4 物流追踪

在物流发货之后，会自动将订单锁定一个物流单号，然后之后就是查看物流进度的时候，就会调用物流公司的接口，进行物流状态的查询

用快递api的免费接口，可以查询物流单号对应的快递的进度

提供一个前台接口，就接收一个orderNo，相当于是那个orderCode，也就是订单编号，然后物流公司编号可以写死，然后还需要当时申请到的那个物流单号，logisticCode。

物流中心需要接收两个参数，orderNo和orderId。然后将orderId调用wms中心的一个接口，wms中心根据orderId查询到这个订单对应的一个销售出库单，然后可以根据销售出库单查询到对应的物流单，从物流单里面可以取出来物流单号。

然后物流中心拿到物流单号之后，可以基于订单编号，物流单号，快递公司编号，去找第三方物流厂商，去获取这个物流单当前的一个发货的进度，进行追踪

# **5、测试用例**

# 6、日志设计