# 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

（1）DO：Data Object，数据对象，一个DO一般是对应于一张表，或者是Mapper查询接口中的返回类型，mapper和dao返回的都是DO

（2）DTO：Data Transfer Object，数据传输对象，service组件需要将dao组件返回的DO对象给封装成DTO对象，再返回给controller组件，或者是别的service组件。service必须将数据封装为DTO，返回给其他的controller或者是service来使用

（3）VO：view object，其实就是将页面中需要展示的数据封装在VO里面，然后在以前是将VO给传输给页面模板，比如JSP（JSTL标签语法，可以将VO中的数据给展示出来），freemarker，velocity，thymeleaf。restful接口，在接口中可以将一个接口要返回的数据，封装在VO里面，从接口返回即可。

（4）Query：查询条件，controller里面接收的页面发送过来的查条件，哪怕是service层对外暴露的接口，也可以用query来接收复杂的一些查询条件

（5）PO：就是将DO不是作为data object，而是作为domain object，然后将PO作为persistence object，一个PO对应一张表，DO就是我们平时放在domain包中的领域模型对象

（6）BO：service组件对外可以返回这个BO，这个BO的话呢是包含业务逻辑的，不光是包含一堆数据和getter和setter方法，还包括了这个很多的业务逻辑方法

（7）AO：application object，就是跟展示层相关的一些数据

大家有没有发现迄今为止，我们耗费了大量的精力在设计这个接口文档。但是其实接口怎么设计，设计的怎么样，不是什么技术问题。接口只不过是两套系统之间互相交互的一个协议。

所以，我是这么想的，如果咱们继续耗费大量的精力在接口设计这块，可能会导致我们的课程进度较为缓慢。学习的价值不是特别大。

就是说，如果是那种面向前端的接口，controller里面的接口，我们就不在这里设计了。写很多json格式的输入和输出，很麻烦，耗费的时间太大了。咱们这块直接在详细设计这个环节给省略掉。当然了，如果你们做项目，一定要按照我之前几讲的示范，所有的接口都在详细设计文档里来写。

但是我们这里，就是说对前端的接口，我们在这里不写了，后面写代码的时候，咱们直接就把接口定义出来，然后就直接写代码。

但是，有一点很重要的是，这个模块与模块之间的交互的接口，比如说采购中心和调度中心之间的交互，都是需要依靠接口的，而且我们需要在整个工程初始化的时候，就将这个接口给定义出来。所以模块与模块之间的交互的接口，必须是要设计的。s

## 2.1 用户当天第一次登录

MembershipFacadeService

Boolean informFirstLoginDailyEvent(Long userAccountId)

## 2.2 用户支付订单

MembershipFacadeService

Boolean informPayOrderEvent(Long userAccountId, Long totalOrderAmount)

~~Boolean updateMemberLevel(Long userAccountId, Long updatedGrowthValue) -> 废了，订单中心居然要来care，我完成了一个订单，我要记住是把订单金额的10%？20%？来累加在用户成长值中？还有完成了成长值的一个累加之后，我还得去care多少分~多少分是什么级别？~~

~~我是不是对会员等级的service调用了一次payOrder接口之后，还要对这个会员积分的service再调用一次payOrder接口，先通知你更新会员等级，再通知你更新会员积分~~

~~订单中心的同学，居然要去care说，会员中心由会员等级，会员积分，会员白条，会员账户余额，会员抵用券~~

~~相当于是MembershipFacadeService，就作为会员中心唯一一个对外的门面接口类，在这个门面接口类里面，会调用会员中心自己的大量的service组件、设计模式的comand组件、dao组件，反正都不用你来care了~~

~~而且你通知我有个用户支付了这个订单之后，我就知道了，剩余要干什么事儿，我自己来care，不用你管~~

~~其实，到了后面的课程，高并发，大家会发现，这种业务场景可以大量的使用MQ来解耦。我们阶段一干出来这一整套的电商业务系统，到了后面还会不断慢慢完善更多更加细致和更加复杂的电商业务。~~

~~这个真的是很棒，大量的MQ、缓存、高并发架构、高可用架构，全部有复杂的业务系统在这里支撑着，非常棒，非常好玩儿~~

## 2.3 用户完成退货

MembershipFacadeService

Boolean informFinishReturnGoodsEvent(Long userAccountId, Long totalOrderAmount)

## 2.4 用户发表评论

MembershipFacadeService

Boolean informPublishCommentEvent(Long userAccountId, Boolean showPictures)

## 2.5 删除用户评论

MembershipFacadeService

Boolean informRemoveCommentEvent(Long userAccountId, Boolean showPictures)

# 3、开发架构

## 3.1 实现类图

设计模式的最高境界，其实就是无招胜有招，刚开始学会的人，总是在考虑，模式怎么套进去使用。当你彻底领悟了设计模式的思想真谛之后，你在设计系统的时候，详细设计的时候，一边脑子里思考业务需求，一边考虑系统设计，一边就可能就是条件反射式的就会想到可以综合起来运用某些某些设计模式，来设计某个业务，让这个业务后续的一些扩展非常的好。

### 3.1.1 门面模式

facade门面模式，其实在这里，就是我们每个子系统对外暴露接口的，统一都是用一个叫做facade的设计模式就可以了，大家应该还记得我之前设计那些对外暴露的接口的service，都是叫什么service。

但是其实那些service包括咱们本讲的service，你都可以在service前加一个facade。因为这些service其实不是我们传统意义上的service，传统意义上的service可能就是给我们的自己的controller调用的，完成内部的一些业务逻辑

但是设计的这些对其他子系统的service，相当于是本系统对其他系统的一个门面service，封装了大量的内部的细节，对外就暴露一些含有业务意义的简单方法。

我们这里提一句，我们在这个系统中，所有的对外的service，全部是facade模式

### 3.1.2 命令模式+模板方法模式

在这里更新会员等级和会员积分，天然可以用命令模式+模板方法模式

上面的那些接口

首先，模板方法模式，这个是肯定要用的。因为按照抽象的思路来想象一下，如果不用模板方法模式，会导致每个接口里都写了相同的更新步骤（5个步骤），还有一些步骤的代码是一样的。

如果不用模板方法模式，相同的更新步骤，不同的地方都定义了，但是可能不同的人写，定义的一些每个步骤的方法名称都不一样，都是有可能的。甚至每个步骤干嘛可能都不一样，甚至有的人还把两个步骤合并为了一个步骤。

有人把更新成长值和更新会员等级放一个方法，有人把更新成长值和更新会员等级放两个方法。那些接口里面的执行步骤本质是一样，结果看起来是不一样的，维护成本很高。

所以说用模板方法模式限定了通用的统一的执行步骤，那么每个接口对应的command的执行步骤看起来都是一样的，维护成本就很低。

用模板方法模式，还可以将一些通用的逻辑，比如说会员等级的计算，放到抽象父command里面去，就可以实现相同逻辑放一个地方

命令模式

如果不用命令模式，会怎么样呢？其实就是说按照一个比较简单的思路，就是单纯用模板方法模式抽象出一套类来。比如我们后续要对每个操作做一些特殊的处理，到了后面，比如对会员积分的更新，要做成异步的操作。

比如说通过纯内存队列来执行，我们可能是需要将不同类型的接口对对应的操作发送到不同的内存队列中去排队，甚至我们可能还需要对每个命令设计一套撤销这个命令的逻辑。比如说我们现在执行这个命令，这个命令有多个步骤，其中某个步骤失败了，此时可以察觉到之后对这个命令执行反向操作。

比如某个步骤增加growth\_value，那么反向操作就是减少；某个步骤是插入明细数据到数据库，反向操作就是删除这条明细数据。

也就是说，我们如果将这个更新的操作封装成命令的概念，后续就可以对这个命令做各种副复杂的高级的操作，排队，撤销。其实就是说，命令模式真正的真谛，就在于说将某个操作封装成一个命令的概念，在后面可以对这些命令做一些复杂的操作，排队、撤销。

命令模式 策略模式 ->

从代码实现上，可能看起来是差不多的，都是一个接口，一堆实现类

命令模式，重要在一个行为，你将同一类不同的操作，封装成一堆命令，这些命令的意义在于说，后续可以基于这些命令执行各种复杂和高级的操作

比如说命令在队列里面排队，这个就是很典型的这么一个例子，不同的命令给分类，放到不同的内存队列里面去排队，异步化

命令，支持撤销操作 -> 执行一个命令，撤销一个命令 -> 这个多个命令可以给一个通用的基础类，封装一些通用的基础逻辑

如果说用户支付了一个订单，那么我们会对这个操作，执行一个命令，这个命令里面，可能会去累加成长值，升级会员等级，累加会员积分；但是，如果之后这个商品被退货了，那么此时就需要对这个命令执行一个撤销操作，这个命令就会基于之前的执行操作的一些结果或者是中间状态的一些记录，将当时执行的那个命令给他撤销掉

这个就成了咱们这个项目里，命令模式一个完美的实践场景，命令模式，一个接口加一堆实现类，但是他的整套用法是不一样的

策略模式 -> if else if else的一坨代码 if 后面跟100行代码，if else后面跟200行代码，else 后面跟300行代码，屎。if后面跟了一堆方法，if else后面跟了一堆方法。会导致你这个复杂的业务逻辑，完全是面向过程在进行编程，没有任何面向对象的设计思路。到了后面，会导致说你的代码很难度懂，很难维护。很难修改。

将if后面的代码，抽到一个策略类里去；else if后面的代码抽到一个策略类里去；else里的代码给抽到一个策略类里去。给每个类以及类中的方法设计更加合理，清晰，面向对象的名称，让整个复杂的业务逻辑，代码结构更加的清晰，便于你未来可以读懂，利于维护这套代码。

只有三种命令

1. 用户每天第一次登录，这个是一个命令，但是是没有撤销操作的
2. 支付订单，是一个命令，是有撤销操作的，如果下一次对这个订单进行了退货，就要将这个命令执行他的撤销操作
3. 发表评论，是一个命令，是有撤销操作的，如果下一次这个评论刚开始通过了审核，上了，但是后面又被人投诉了，被删除了，此时就要执行这个命令的撤销操作

给这些命令搞一个抽象的基类，里面封装一些命令相关的通用基础逻辑

### 3.1.3 单例模式

实际上，单例模式，就是在大量的工厂模式中来使用，工厂一般都做成单例的

## 3.2 包设计

# 4、运行流程

## 4.1 用户账号注册与登录

### 4.1.1 用户注册

输入用户名和密码，在后台校验一下是否已经有这个用户名了，如果有就不允许注册这个账号；如果没有这个用户名，就可以注册这个账号。我们之前就给user\_account这个表搞了一条记录，但是我们其实同时应该给user\_detail这个表搞一条记录，但是大部分字段都是null就可以了。

### 4.1.2 用户登录

（1）根据用户提交的用户名和密码，去账号表里查是否有，是否正确，如果正确，那么就允许登录

（2）登录之后将用户账号信息放在session中，代表已经登录了

### ~~4.1.3 用户登录过滤~~

~~所有的请求，都要走一个用户是否登录的过滤器，看一下当前session中是否有对应的账号信息，如果没有，代表还没登录，返回一个异常值，让前端同学给跳转到登录界面；如果session中有账号，代表就已经登录了~~

## 4.2 个人信息管理

### 4.2.1 个人信息查询

上来可以将用户的个人信息都给查询出来

### 4.2.2 个人信息设置

可以设置自己的详细的个人信息

### 4.2.3 收货地址管理

可以查询收货地址列表，新增收货地址，编辑收货地址，删除收货地址

### 4.2.4 修改密码

要求的是输入以前的旧密码，再输入两遍这个新密码就可以了

### 4.2.5 查询会员等级变更明细

点击自己的会员等级，可以进入一个明细列表页面，显示了明细的所有数据，可以分页查询，不支持条件筛选。

### 4.2.6 查询会员积分变更明细

点击自己的会员积分，可以进入一个明细列表页面，显示了明细的所有数据，可以分页查询，不支持条件筛选

## 4.3 会员等级变更

每天的第一次登录：获得成长值奖励，+5分

购物：订单金额 \* 成长系数，算出来一个成长值奖励，10%，100元的商品 \* 10% = +10分

退货：扣减成长值，商品金额 \* 10%，扣减你的成长值

评价：发表一个评论，可以积累成长值，+10分

晒单：进行晒单，可以积累成长值，+10分

评论被投诉后删除了：扣减成长值，扣减你的评论的成长值

100分以下：铜牌会员，1

100分~500分：银牌会员，2

500分~1000分：金牌会员，3

1000分以上：钻石会员，4

在对应的操作发生的时候，需要过来调用会员中心的接口，需要根据上面的算法完成会员成长值的变更，同时完成会员等级的变更，同时同时会员等级变更明细的记录。

## 4.4 会员积分变更

每天的第一次登录：获得成长值奖励，+5分

购物：订单金额 \* 成长系数，算出来一个成长值奖励，10%，100元的商品 \* 10% = +10分

退货：扣减成长值，商品金额 \* 10%，扣减你的成长值

评价：发表一个评论，可以积累成长值，+10分

晒单：进行晒单，可以积累成长值，+10分

评论被投诉后删除了：扣减成长值，扣减你的评论的成长值

在对应的操作发生的时候，需要过来调用会员中心的接口，需要根据上面的算法完成会员积分的变更，同时完成会员积分变更明细的记录。

# 5、测试用例设计

# 6、日志设计