设计模式综合应用

> 3020001267 王旭
>
> 同组成员：刘书任 洪图 张翰林

题目要求：分析“东软教育实战化教学项目资源包—饿了么外卖平台”（以下简称“饿了么项目”）中存在的问题，并给出改进建议。可以从以下几方面分析：

（1） 软件架构设计中存在的问题
（2） 接口设计中存在的问题
（3） 违反面向对象设计原则的问题
（4） 违反数据库设计原则的问题
（5） 违反用户使用的习惯和需求的问题
（6） 其它问题，如拼写错误、错别字等

分析内容包括“饿了么项目”从项目一到项目四的内容，重点分析项目三前端（VUE 技术）与项目四后端（Spring Boot 技术）结合中存在的问题，并给出相应的改进建议。

1. 软件架构设计中存在的问题

（1）vue代码冗余

在原项目中有部分功能存在冗余。比如前端对后端的axios请求，几乎在每个页面都会重新写一遍this.$axios.post().then()请求模板，这个请求模板完全可以作为一个通用的请求函数。这样在每个axios请求时就可以调用这个函数，达到减少代码行数，可读性更好的效果。

（2）CS架构对于兼容性的问题

老式的浏览器比如IE浏览器并不支持vue，如果客户使用这些浏览器的话，那么根本无法打开页面。

（3）对SprintBoot架构的介绍

建议增加对SpringBoot框架的介绍，使得学生在进行本饿了么项目时可以不仅仅会用SpringBoot，也对其架构建立一定的了解。明确三层的意义，更有助于理解“为什么要这么写”的问题。

2. 接口设计中存在的问题

（1）vue中输入参数验证校验不全面

接口文档以及具体实现中都出现了这个问题，如对入参数据类型、长度边界，范围边界等没有限制，在本项目大部分情况就是用户输入什么，就传送什么参数，没有对传入参数进行校验，这样会给后端带来很多压力。

（2）vue在发送请求时滥用post方法

axios支持get、post、put等多种请求方法。一般来说get一般用于获取数据、post用于提交数据（表单提交+文件上传）。而在本项目中，大部分接口目的在于从后端获取数据，而不是向后端上传数据，所以使用get方法会更加合理，但是本项目的所有接口的请求方法都是post。

（3）安全性不足

本项目中存在涉及到登录的接口getUserByIdByPass，参数时用户ID、密码。但是在传入密码时并没有进行任何加密措施，如果有恶意监听者监听这个接口产生的数据则会造成极大的安全隐患。

（4）接口参数与返回值太过笼统

在给定的接口设计中，大多数返回值都为一个对象，而实际前端用到的可能其中一个成员。这严重违反了最小知识原则。

对此修改，我们应该明确接口的参数类型，只提供需要的内容。

对于SpringBoot内部的三层间的接口，同样涉及这种问题，比如Controller层只需要一个UserId，但是底层却传来了一个User对象。

3. 违反面向对象设计原则的问题

（1）滥用getter、setter方法

getter、setter方法原本是为了一些被private的成员可以修改和访问，但是本项目对所有成员都加了private方法，变相的使得private和public基本无差别。

对此改进，应该在PO层对所有对象的成员进行详细判断，区分其是否真的需要getter和setter。

（2）基于贫血模型开发

本项目的后端实质上是贫血模型的开发模式，是彻彻底底的面向过程编程风格的。本项目的所有类都只是对应了数据库的一条记录，除了getter、setter以外就没有对应的操作，所有对对象的操作都是在Service层中实现的。这就使得本项目在应对增删改查这类简单的业务逻辑时尚且游刃有余，但是涉及到更复杂的业务逻辑时，特别是需要频繁迭代的商业系统时，就会凸显出大量的缺点。

（3）基本看不到面向对象的影子（所以根本无从谈起违反面向对象设计原则）

本项目在实现时（SpringBoot部分）的唯一感受就是太简单了。SpringBoot把你需要做的事情都继承好了，Mapper层要么直接注解写SQL超级简单，要么XML写SQL难一点但是也好理解；Service层分为接口和实现两层，关系也非常明显（其实这里可能可以考虑添加一下对于同种接口的两种实现，但是业务逻辑简单也不涉及这些）；Controller层更为简单，写好与上层接口调用的关系即可。

总的来说，这里唯一的缺点就是看不到面向对象，或者说这里唯一的面向对象在于有这些分层包括PO对象等，但是其并没有体现出面向对象的优点。获取正如上一点所说，当我们基于充血模型开发后才能看到面向对象的优越。

无论是现在的业务逻辑简单，还是SpringBoot架构轻松，我觉得都不能称为减少面向对象的原因。希望在程序设计中级实践中可以利用好学到的面向对象设计原则。

4. 违反数据库设计原则的问题

（1）对订单的删除操作

目前对订单的删除操作是直接删除，这样可能会导致数据不可恢复的消失。当我们有必要对过去数据进行查询时，这种由用户直接对数据库的操作会导致数据不完整，以至于我们无法获取到正确的数据。这违反了数据库设计的完整性原则。

对其修改，我们可以对订单行增加deleted列，使之成为一个用0/1标记的元组。这样当用户删除时，只需修改deleted即可，查询时给用户返回deleted为0的订单行。而真正有高级用户（顶级管理员等）想要对订单进行查询时，可以不关注deleted列等方式获取到完整的数据信息。

（2）对密码的存储

目前数据库中的密码存储是明文存储的，对于一个项目来说，用户的密码是不应被管理员所知的。而现在，管理员可以轻而易举的获取用户的密码（当初答辩的时候就存在这样的问题，老师注册账号后我可以直接看到老师的注册密码），如果用户多平台的密码都一样的话，这样会造成很严重的安全问题。这违反了数据库设计的安全性原则。

对其修改，我们可以使用一些密码加密算法，这涉及到密码学。我们需要的是一个双向加密的算法，即管理员无法通过密文和密钥（如果有）来获取明文密码，或者管理员无权获取密钥。如果自己实现较为困难，可以考虑调用第三方加密算法。

（3）对表的统一

目前项目中有两套表，即阶段一和阶段二三四。我们在进行阶段一可视化（称之为阶段五）时，延续了阶段一的表。但是，这样对商家的修改是无法同步到阶段二三四的表中的。这违反了数据库设计的一致性原则。

对其修改，我们可以再次对整个项目进行概览，明确项目的用户关系（管理员、商家、客户等），建立好一套统一的表，确保表的一致性，即对某内容修改时，所有项目中设计这个内容的地方对该信息的了解应该是一致的。

（4）对表的索引

目前对表的查询较为简单，逻辑清晰。但当用户量庞大时（以亿计），这种查询方式完全无法因对高强度的用户查询。

因此，要对数据库进行优化，增加索引等方式，尽可能使数据库查询的效率达到最高。

5. 违反用户使用的习惯和需求的问题

（1）vue返回按钮的缺失

任何一个界面都没有返回按钮，应该设置一个按钮可以让用户返回到上一个页面。

（2）vue缺乏一些动态特效

页面比较死板，没有动效，显得比较粗糙。

（3）在Orders.vue也就是确认订单页面，显示的是用户名。而此处显示收货人的姓名更好

vue.config.js　UserAddress.vue　Orders.vue

```
5       <p>确认订单</p>
6     </header>
7     <!-- 订单信息部分 -->
8     <div class="order-info">
9       <h5>订单配送至：</h5>
10      <div class="order-info-address" @click="toUserAddress">
11        <p>
12          {{
13            deliveryaddress != null ? deliveryaddress.address : "请选择送货地址"
14          }}
15        </p>
16        <i class="fa fa-angle-right"></i>
17      </div>
18      <p>{{ user.userName }}{{ user.userSex | sexFilter }} {{ user.userId }}</p>
19    </div>
20    <h3>{{ business.businessName }}</h3>
```

改正后如下：

```
<p>
  {{ deliveryaddress != null ? deliveryaddress.contactName : ""
  }}
  {{
    (deliveryaddress != null ? deliveryaddress.contactSex : "3")
      | sexFilter
  }}
  {{ deliveryaddress != null ? deliveryaddress.contactTel : "" }}
</p>
```

6. 其它问题，如拼写错误、错别字等

（1）1.3.2.1 business（商家表）中，起送费虽好是startPrice

1.5.2.1.business（商家表）

No	字段名	数据类型	size	默认值	约束	说明
1	businessId	int			PK、AI、NN	商家编号
2	password	varchar	20		NN	密码
3	businessName	varchar	40		NN	商家名称
4	businessAddress	varchar	50			商家地址
5	businessExplain	varchar	40			商家介绍
6	starPrice	decimal	(5,2)	0.00		起送费
7	deliveryPrice	decimal	(5,2)	0.00		配送费

（2）4.2.1.5 OrderDetailetMapper类中，参数拼写错误

4.2.1.5.OrderDetailet

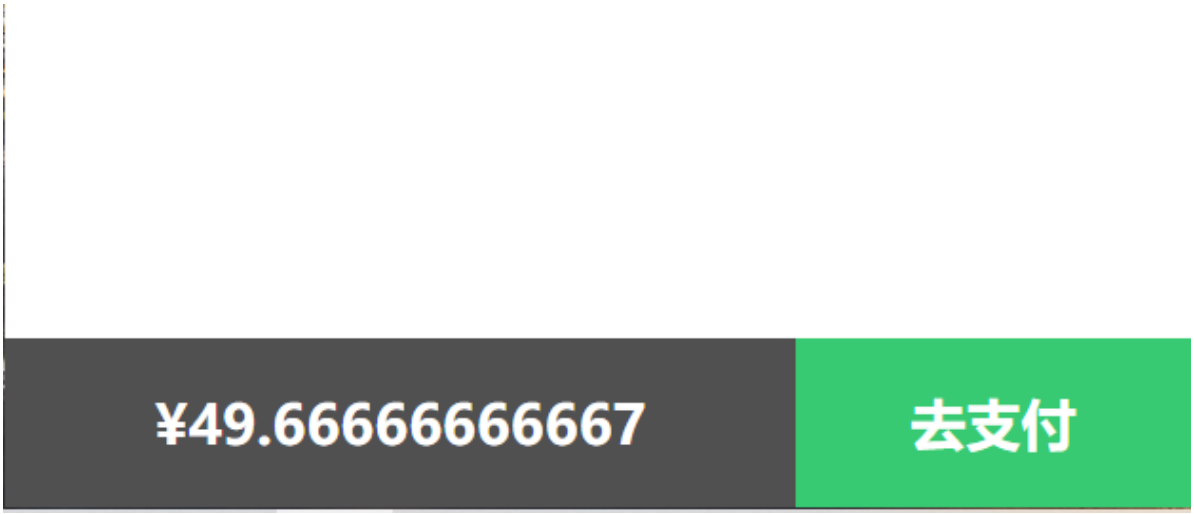
```
package com.neusoft.elmboot.mapper;

import java.util.List;
import org.apache.ibatis.annotations.Mapper;

import com.neusoft.elmboot.po.OrderDetailet;

@Mapper
public interface OrderDetailetMapper {

    public int saveOrderDetailetBatch(List<OrderDetailet> list);

    public List<OrderDetailet> listOrderDetailetByOrderId(Integer orderOd);
}
```

（3）4.2.1.6 OrdersMapper类的课件代码给的是UserMapper的

4.2.1.6.Orders

```
package com.neusoft.elmboot.mapper;

import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import com.neusoft.elmboot.po.User;

@Mapper
public interface UserMapper {

    @Select("select * from user where userId=#{userId} and password=#{password}")
    public User getUserByIdByPass(User user);

    @Select("select count(*) from user where userId=#{userId}")
    public int getUserById(String userId);

    @Insert("insert into user values(#{userId},#{password},#{userName},#{userSex},null,1)")
    public int saveUser(User user);
}
```

（4）vue对小数位数没有限制



有些数字浮点数并不能精确表示，在科学计算时还无伤大雅，但是在前端页面的展示时就需要十分注意了，并且不只是展示的问题，由于后端数据库表中的小数都是由保留两位小数的decimal精确表示的。若前端中不对小数位数进行限制，还会给后端带来极大的麻烦。在纠正这一问题时，只需要对小数进行保留两位的取整即可。

```
return Math.round((total *100))/100.0;
```