基于Vue+Node实现网上购物商城

鞠卫南

摘 要

随着科技的发展，时代的进步，相较于百年前已经发生了翻天覆地的变化。尤其是已经普及到千家万户的计算机互联网，已经彻底的改变了人们的传统生活方式。比如当下大火的网上购票、网上求职、网上预约、网上购物等。近些年来，电商行业如雨后春笋般涌出。正是由于其方便快捷、商品样式种类琳琅满目、价格便宜、送货上门等众多优点深深地拢获了一大批购物者的兴趣；很多厂家也开始做起了微商，通过网络这个渠道，更好地推销自己的产品。因此再也不用担心商场是否人满为患，时间是否充足等问题，使得买家更省心，卖家更安心。

本课题是通过对当今电商行业的发展现状分析设计了只可以浏览商品的游客和可以支付下单的会员两种用户类型的网站购物商城，包括热门推荐、商品分类、商品搜索、商品详情信息展示、加入购物车、支付下单、评论以及登录注册等功能。

本商城使用了Vue全家桶技术对页面进行结构搭建和数据渲染；使用scss预编译语言对样式进行美化；通过Node.js作为服务器语言为商城提供数据接口；然后通过基于Promise封装的axios异步请求将前后端连接起来。所有的数据都放在MySQL数据库中管理。

摘要：网上购物；Vue全家桶；Node；Express；MySQL；前后端分离

**Abstract**

With the development of science and technology, the progress of the times, compared with a hundred years ago, has changed dramatically. Especially the computer Internet, which has been widely used in thousands of households, has completely changed people's traditional way of life. For example, the current fire of online ticket buying, online job hunting, online booking, online shopping and so on. In recent years, e-commerce industry has sprung up. It is because of its convenience, variety of commodity styles, low price, door-to-door delivery and many other advantages that a large number of shoppers are deeply interested in; many manufacturers have also started micro businesses, through the network channel, to better promote their products. Therefore, there is no need to worry about whether the shopping mall is overcrowded, whether the time is sufficient and other issues, so that buyers are more worry free and sellers are more secure.

Based on the analysis of the current development of e-commerce industry, this paper designs two types of website shopping malls: tourists who can only browse commodities and members who can pay for orders, including hot recommendation, commodity classification, commodity search, commodity detail information display, adding shopping cart, paying for orders, comments and login registration.

In this mall, Vue family bucket technology is used to build the structure of the page and render the data; SCSS precompiled language is used to beautify the style; through Node.js As a server language, it provides the data interface for the mall; then, it connects the front and back end through the Axios asynchronous request based on promise encapsulation. All data is managed in MySQL database.

**Key Words:** Online shopping; Vue family bucket; node; Express; MySQL;

front and back end separation

目 录

[1 背景介绍 1](#_Toc12683)

[1.1 课题背景和意义 1](#_Toc10696)

[1.2 项目开发目的 1](#_Toc10921)

[1.3 系统可行性分析 2](#_Toc1199)

[2 系统基础知识理论 2](#_Toc21584)

[2.1 系统开发环境介绍 2](#_Toc12608)

[2.1.1 VSCode 3](#_Toc18472)

[2.1.2 Webstorm 3](#_Toc1718)

[2.1.3 Navicat Permium 4](#_Toc23528)

[2.2 系统涉及到的技术点介绍 4](#_Toc6576)

[2.2.1 前端技术 4](#_Toc6255)

[2.2.2 后端技术 6](#_Toc25443)

[2.2.3 版本控制技术 7](#_Toc7844)

[3 系统总体设计 8](#_Toc885)

[3.1 系统功能分析 8](#_Toc4208)

[3.2 系统模块结构 8](#_Toc31141)

[3.3 系统界面设计 8](#_Toc1603)

[4 数据库设计 8](#_Toc23972)

[4.1 数据库分析设计 8](#_Toc7683)

[4.2 数据库概念结构设计 9](#_Toc3790)

[4.3 数据库逻辑结构设计 11](#_Toc22117)

[4.3 数据库连接原理 15](#_Toc25257)

[5 系统详细设计 15](#_Toc16881)

[5.1 系统操作流程图 15](#_Toc20559)

[5.2 系统实现的具体逻辑 15](#_Toc14147)

[6 程序运行与测试 46](#_Toc18478)

[6.1 测试的目的和意义 46](#_Toc4171)

[6.2 程序测试分析 46](#_Toc8899)

[6.3 其他错误 48](#_Toc22684)

[结束语 48](#_Toc9791)

[参考文献 49](#_Toc2817)

[致 谢 49](#_Toc25136)

1 背景介绍

* 1. 课题背景和意义

当今时代，随着时代的发展和科技的进步，互联网已经走进了千家万户，甚至已经是每个人生活中不可缺少的一部分了。越来越多的事情都可以通过网络来处理，比如网上求职、网上购票打车、定外卖等新兴产业。随着阿里巴巴这个世界级的电商企业问世，网上购物已然成为当今时代购物方式的主流，越来越多的企业在这个电商互联的潮流下如雨后春笋般涌出。

在电商行业飞速发展的时代，由于其方便快捷、商品样式种类琳琅满目、价格便宜、送货上门等众多优点深深地拢获了一大批购物者的兴趣；越来越多的商家也一边线下经营店铺，另一边则通过网络这个渠道，更好地推销自己的产品，能够让全国各地的人购买；也有更多的人加入到快递小哥的队伍中，承担起商家和买家之间的责任的桥梁。但是由于加入的人越来越多，导致市场竞争更是越来越激烈了。那么，怎么做才能够使自己能够在这个大潮流下不被淘汰然后再脱颖而出是一个必须要重视的问题。

本着一切以人为本，为人民服务的态度，所以作为一个电商平台，我们要尽其所能的给用户带来更好的体验、更多福利，能让人们付出同样的努力收获更满意的结果。对用户的财产、个人信息给予最安全的保障，这才能真正的做到网络改变生活。

* 1. 项目开发目的

通过上面对于背景的介绍和意义的总结，我们可以看出网上购物确实是给人们带来了更放心、更安心、更省心的体验。既节约了时间，有提高了生活。商家接触到了网上购物商城后，可以一方面继续线下维持传统经营模式，另一方面将自己的商品发布到网络上进行售卖。这样既不会对传统经营方式造成不良影响，还能扩大经营范围。同样很多顾客由于工作忙时间有限，所以相比于花时间去商场里走来走去的购物他们更愿意选择网上购物这种方式。另一方面，网上购物也催生了快递行业的发展。所以本程序带来的是三赢的局面。解决了购物难，卖货难的等为题，更催生了快递行业的发展。

1.3 系统可行性分析

从技术上来说，本程序采用了前后端分离的开发方式。前端整体上是SPA（单页面应用）开发方式，只负责页面结构搭建，样式美化，通过axios异步请求方式调取后端提供的数据接口动态渲染页面；后端使用的是Node.js中的Express框架，主要负责页面用户的交互事件触发的接口，根据触发的接口，对数据库进行操作，并将状态以及结果返回给浏览器；这种开发的好处是，前端和后端各司其职，如果出现错误时，能直接通过控制台定位到是问题是出现在前端还是出现在后端，这样大大提高了代码的开发效率，为了日后的维护或者功能修改提供了很大的方便。

从社会经济上来说，此程序一方面为顾客提供了更加方便快捷的服务，能够随时随地，随意根据自己的喜好购买心仪的商品。节约了时间，去做更有意义的事情。另一方面可以有更多的传统经营买卖的商家可以开通线上营销的渠道，更好的推销自己的商品，也推动了社会经济的发展。同时，快递行业也日益发展壮大，开通了一个新的行业，也让更多没有工作的人能够有就职的机会。做到了一个系统能够推动三方面的经济进步。

从操作上来说，程序界面简洁大方，可视性好，页面每个模块都是一个独立的部分，模块之间不会产生冲突，并且都有文字标题作为提示。对于部分不可逆的操作都会有弹框提示，防止用户不小心触摸造成不可逆的错误。从浏览商品到下单支付，收获评论等操作流程与淘宝等电商App基本一致。大大提高了用户使用该系统的购物体验。

从法律上来讲，本程序功能点符合大多数购物商城应有的基本功能，但是用户体验以及界面上并不一致，因此不存在抄袭现象。技术点使用的也是当今主流的开源技术，功能实现的方法以及使用的插件都是开源代码，所以不存在侵权行为，完全可以放心使用。

综上所述，开发的网上购物商城是完全可行的。

2 系统基础知识理论

2.1 系统开发环境介绍

2.1.1 **VSCode**

在Build 2015大会上，微软发布的除了Windows以外，最大的惊喜莫过于其推出的可以免费使用的Visual Studio Code 编辑器了。这对于前端开发者来说，无疑是一个好消息。作为轻量级编译器，使开发者不需要高配置的电脑也可以更快的打开和使用；工欲善其事，必先利其器，vscode为开发者提供了多种多样的插件库，每个人可以根据自己的喜好下载插件并配置好这款工具；他支持大多数主流开发语言的语法高亮显示；同时我们还可以根据个人喜好自定义快捷键和关键字，一键生成代码模版；VSCode对于Git的支持也更佳友好，更清晰的根据文件目录显示的颜色和字母来判断代码相对于Git版本控制的状态，并且有版本冲突时也会有更详细的对比；而且他还支持汉化功能，对于英语不是很好的程序员来说中文的编译器简直帮了很大的忙。这款编译器给我最完美的使用体验莫过于他和ESLint的结合了，我们只需要下载ESlint插件，并配置好开发中的约束规范json文件，不管代码写的多乱，只需要手动保存一下，你的代码就会按照约定好的规范自动修改，从而使你写代码会有飞一般的感觉。

2.1.2 **Webstorm**

作为JetBrains旗下唯一一款且深受前端程序员喜爱的编译器，拥有着强大的后台支持以及更完善的开发社区。它自身集成了多种版本控制工具，相比较传统的输入指令操作版本，提供了可视化的版本控制操作。还拥有独特的的本地历史功能，可以通过本地历史查看过去时间的代码，能够将代码版本随时恢复到某个时间点。Webstorm还有一个即时模版功能，我们可以根据自己的需求编写一个代码文件的模版，在文件中输入关键字后，点击键盘上的tab即可一键生成整个文件。WebStorm支持多光标，可以同时修改多行相同的代码，大大提高了代码的开发效率。webstorm提供了一个自动修改引用文件名字以及文件位置的功能，给开发者带来了很大的便利。它可以集成Node.js。我们只需要给webstorm添加服务配置，点击编译器上的三角形就能直接启动服务，同时还支持debug等功能。

但是，这款编译器也有一个致命的缺点，就是它是一个付费的软件，虽然现在有很多种方法可以破解，但是还是会给刚刚涉及前端领域的同学代码很大的麻烦和不便。其次就是这款编译器体积很大，如果电脑配置不是很高的同学不建议使用，因为在启动软件的时候会浪费很长的时间，而且在编写代码的过程中也会出现卡顿或者未响应的体验。

2.1.3 **Navicat Permium**

作为数据库的一款可视化管理工具，他支持mysql、mongodb、oracle等多种数据库，让管理各种数据库中的数据变得更加方便。它支持传统的命令式数据库操作方式的同时也支持通过可视化界操作数据库和表等操作。系统界面简洁友好，支持汉化。我们可以更直观的查看数据库中的所有数据。它还支持sql数据文件倒入以及数据库文件的导出。

2.2 系统涉及到的技术点介绍

2.2.1 前端技术

1) Vue全家桶技术

Vue作为前端主流三大框架之一，相比于其他两个框架它的学习成本相对更低，采用了前端非常流行的的MVVM（Model-View-ViewModel）开发模式，是MVC模式的改进版。在vue中，Model代表的是数据层，也是最核心的部分，View作为视图层负责展示，他们两者之间是不能之间通信的，这样开发做到了最大程度的数据结构相分离。作为他们之间沟通的桥梁的ViewModel，会感知用户操作View产生的变化后及时通知Model同步数据，反之数据发生变化时也会及时通知View做出更新渲染。它一方面与视图层View进行数据的双向绑定，另一边与Model层进行数据交互和修改，起到了承上启下的作用。如下图:



我们还要知道，前端在渲染DOM的时候开销是很大的，比如我们操作修改了某个数据，这将会引起整个DOM的重绘和重排，而你只是改变了很小的一部分。Vue恰恰为我们做出了很好的优化工作，它使用了diff算法，先根据真实的DOM生成一棵virtual（虚拟）DOM树，当页面发生变化时又会生成另一棵新的虚拟DOM树，这时只需比较两棵树并将不一样的地方渲染在真实的DOM树上，从而大大降低了整个DOM都被重新渲染带来的问题。

Vue-router是vue开发成为单页面应用的重要角色，他可以在页面内容发生改变时不加载整个网页。由于单页面应用没有页面之间的切换，所以就不会出现白屏或者假死闪烁这种不好的现象。对于服务器来说压力也会更小，服务器只需要将数据提供给前端就好了，大大提高了吞吐能力。而且vue-router支持路由懒加载，所以也不会出现第一次加载项目时速度太慢的问题。

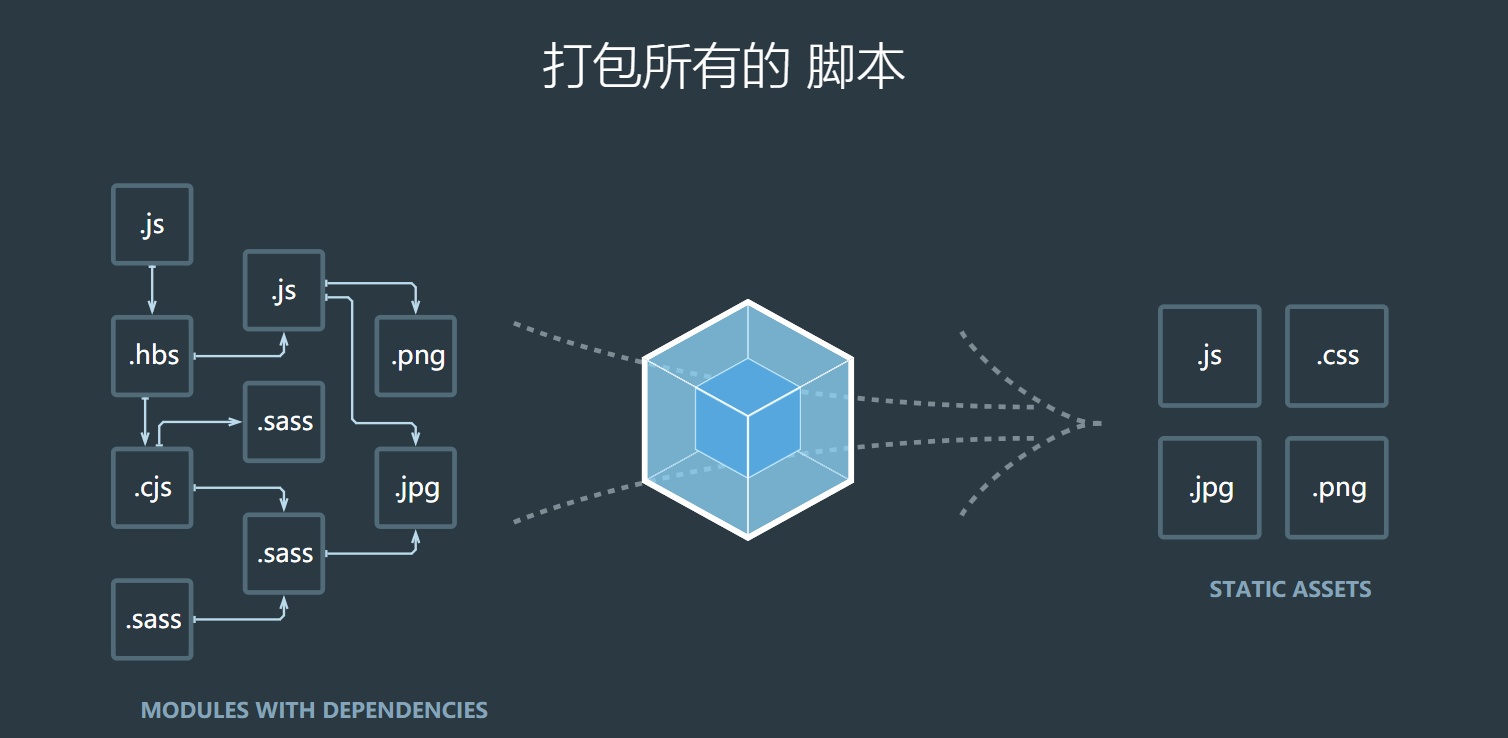
我们上面有讲到vue中的核心就是数据，所以就会出现一个数据多处使用的问题，虽然vue支持父子组件传值，但是层级过于复杂时也会让代码难以维护。所以这个全局状态管理器就起到了很大的作用，我们只需要按照规定的语法，就能很方便的对存储在vuex中的数据修改和使用。

1. SCSS

这是一个给CSS加入了编程元素的CSS预处理器。他的好处就是和CSS编写样式一样，不需要花时间学习一门新的样式语言的同时我们还可以就像写js一样定义变量、函数和代码片段，用到的时候只需要引用变量名就可以了，它加强和提升了CSS，极大地确保了整个程序设计的一致性和可用性。他还支持嵌套形式的开发格式，这样就省去了还要引用父节点类名的时间。通过这种新的编程语言，可以使用最高效的方式，最少的代码开发复杂的程序设计。

1. Webpack

在我刚刚接触前端的时候，所有的静态资源文件都是需要手动引入页面中，所见即所得。这种开发方式最后的结果就是开发一时爽，维护火葬场。后来接触了webpack。这款自动打包工具的好处就是不但可以将项目中的所有CSS和JS文件分别压缩合成单个文件，在发布项目到服务器时能减小服务器压力。他还能够将浏览器不能识别的SCSS预编译语言、ES6、7、8转换成浏览器能识别的代码文件。同时webpack支持模块化和组件化开发，尤其是对于前端主流的Vue、React框架也有更好的支持。最重要的是，webpack拥有非常庞大的社区，资源相当丰富，大大降低了学习成本。



1. Axios

最早出现的发送后端请求技术是ajax，他使用的核心是XMLHTTPRequest对象，但是由于项目需求庞大，很多请求的发起是建立在前一个请求结束上的，这样就造成了回调地狱的问题。后来jQuery对原生的ajax进行了封装，并且经过一次次的版本迭代已经可以非常方便的使用了。但是很多新项目中不可能为了一个ajax引入整个jQuery，这样不但会造成资源的浪费，还会使项目体积过于庞大，影响整体运行速度。

在Vue2.0之后，尤雨溪推荐大家使用axios代替jQuery中的ajax。它是基于ES6中的Promise对象封装的一个异步请求插件，他可以在请求和响应拦截器中处理公共业务，每个接口只需要专心的处理自己的业务。还支持restful风格的接口请求，支持链式和函数式的接口调用方式，还能够自动将后端返回的数据转换为json格式等等。

2.2.2 后端技术

1. Node.js

Node区别于传统的Java、PHP独特之处在于很多模块都是异步执行的，用户使用js调用node的核心部分，将回调函数和回调参数传入核心模块中进行封装，将得到的结果储存在请求对象的result属性中，提醒回调成功。如果还有未完成的，就在进入对象请求I/O观察者队列之后当做事件处理，也就形成了异步I/O。所有的阻塞事件全部放在一个线程池中，然后主线程以队列的方式和线程池交互工作，我们不需要关心线程的相关问题，然后通过多个回调函数完善功能。Node底层选择使用C++和谷歌的V8引擎实现，即使面对着大规模的http请求，也能凭借着事件驱动很好地完成。

1. Express框架

Express是一个功能简单，完全通过路由、中间件构成的一个轻量级的node服务器。当前端的每一个请求进来的时候，都会通过express处理之后在响应给用户，无论多复杂的业务都变成线性的。而且express中的所有请求都被当做事件来触发，通过接受浏览器传入的接口地址，找到对应的业务处理函数，并通过最普通的callback方式将异步操作的结果返回给浏览器，但是对于这种古老的callback方式变得业务不可组合，异常也没有办法捕获，致使很多人不愿意为它买单。不过对于业余后端人员来说也足够编写优美的代码了。

2.2.3 版本控制技术

Git是很好的版本控制技术。他执行的所有语句都是Linux语句。比较适合分布式开发，每个人只需要开发自己的功能文件，如果两个开发者之间代码产生冲突也可以非常轻易的解决。代码提交速度快，管理起来更加方便灵活。但是git操作的命令过多，没有系统的中文资料能够学习。而且代码保密性较差，只要克隆整个代码库就可以看到代码的所有版本功能等信息。

3 系统总体设计

3.1 系统功能分析

该程序是一个面向用户的H5网上购物商城，设计理念和操作方式如同淘宝、京东等APP购物平台。选择H5的设计是因为H5兼容性更好，且可移植到APP或者微信小程序中。界面简单大方，用户操作提示信息友好。用户可以在程序中完成注册登录功能，忘记密码时还设计了通过绑定的邮箱发送验证码的方式重置密码。登录之后用户可以查看商品分类、品牌，搜索商品，查看商品详情信息。对于喜欢的商品可以收藏和添加购物车，输入支付密码即可下单、催单、收货、评论商品信息，可以自由设置个人信息和收货地址等功能。

3.2 系统模块结构

系统功能设计图

3.3 系统界面设计

程序界面设计主要是参照了原型官网上的一个电商APP、淘宝等外加自己的一些想法设计的。因为本程序是面向购物者开发的，所以在易操作、规范性、合理性和美观上与大多数电商APP基本保持一致。在页面设计上尽量做到见名知意，清楚地传达给用户这里是做什么的，都有什么功能，以及如何操作，整体上符合大多数人的审美，相似的模块保持样式基本一致，颜色搭配美观大气。页面上所有的内容符合规范，不传达任何消极违法信息。尽量做到在视觉、交互效果上给用户最完美的体验。

4 数据库设计

4.1 数据库分析设计

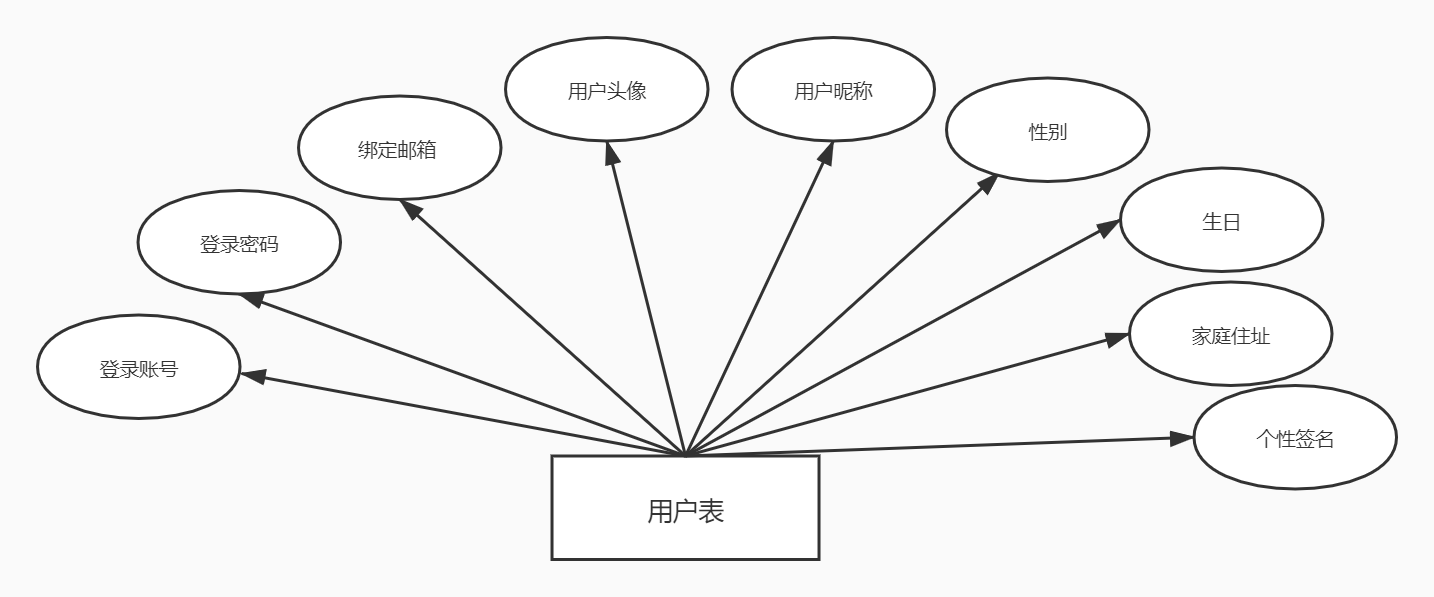
数据库为系统中的所有数据进行了持久化的管理，有了数据库，我们可以更好的对数据进行增加、删除、修改和查看。我们需要根据项目中的业务逻辑设计好E-R关系图，然后根据E-R图对数据库中的表作出相应的设计，这样的数据库设计方式会让后端接口在操作数据的时候也会更加方便。每一个表中都必须有一个递增的主键作为表中一条消息的唯一约束，为了使表与表之间的数据更好的关联，字表中应该还要有主表的唯一标识，这样的表结构会更加清晰，利于维护和优化。

4.2 数据库概念结构设计

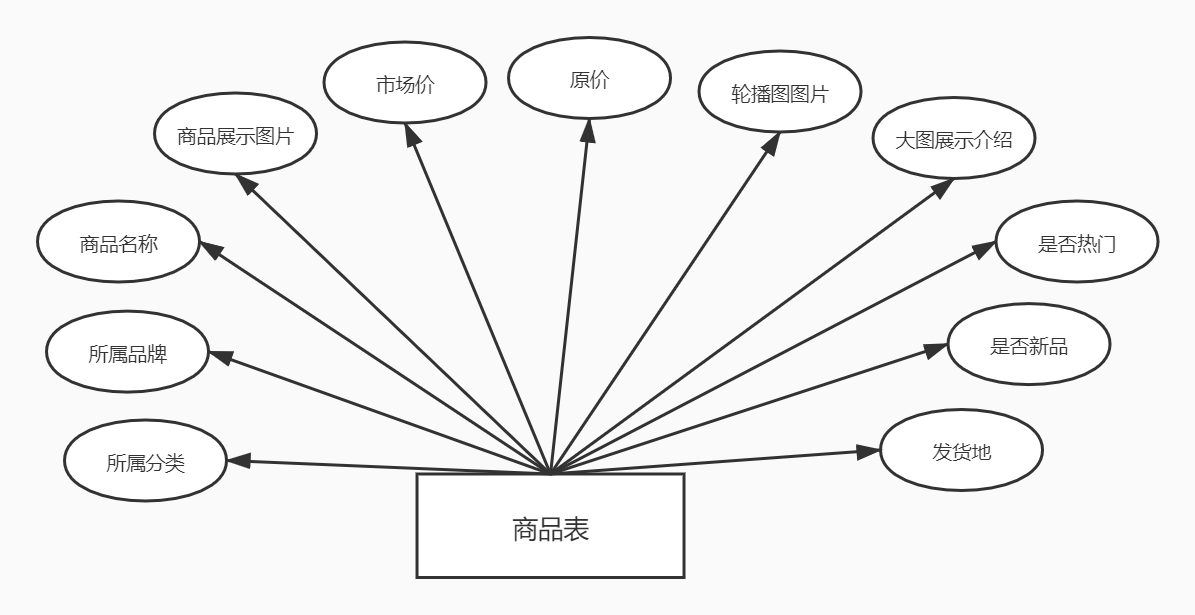
本设计中最主要的表结构就是登录用户表和展示的商品表。在用户已经登录和商品存在的情况下可以加入购物车、生成订单、选择收货地址、发表评论等。根据他们之间的关系，可以实现一下关系图：



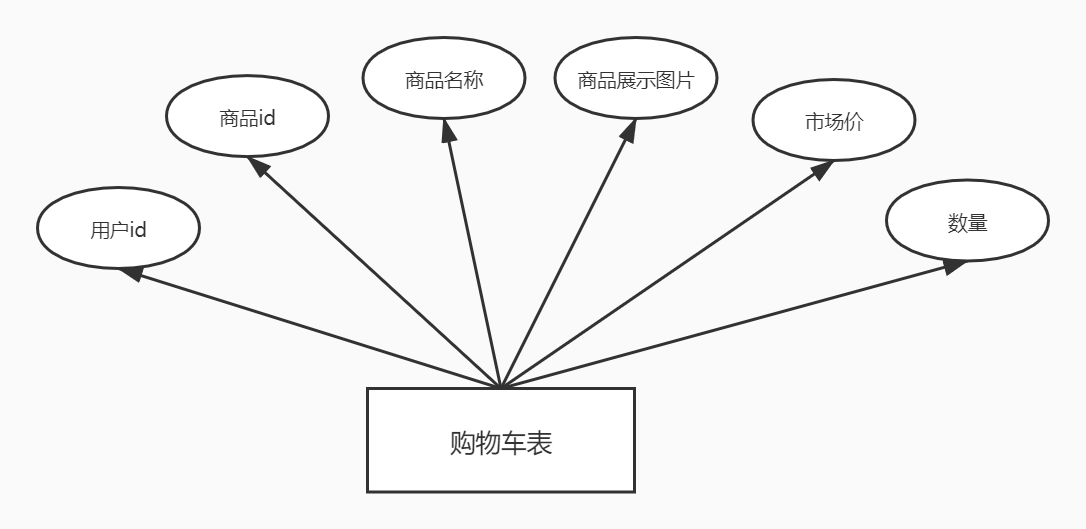
1. 登录用户实体E-R图



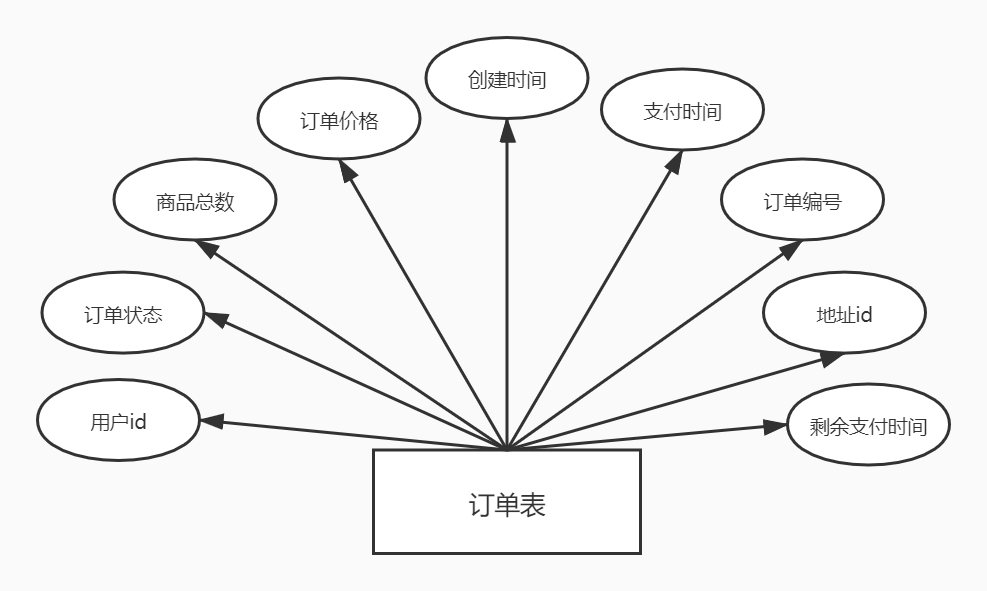
1. 商品实体E-R图



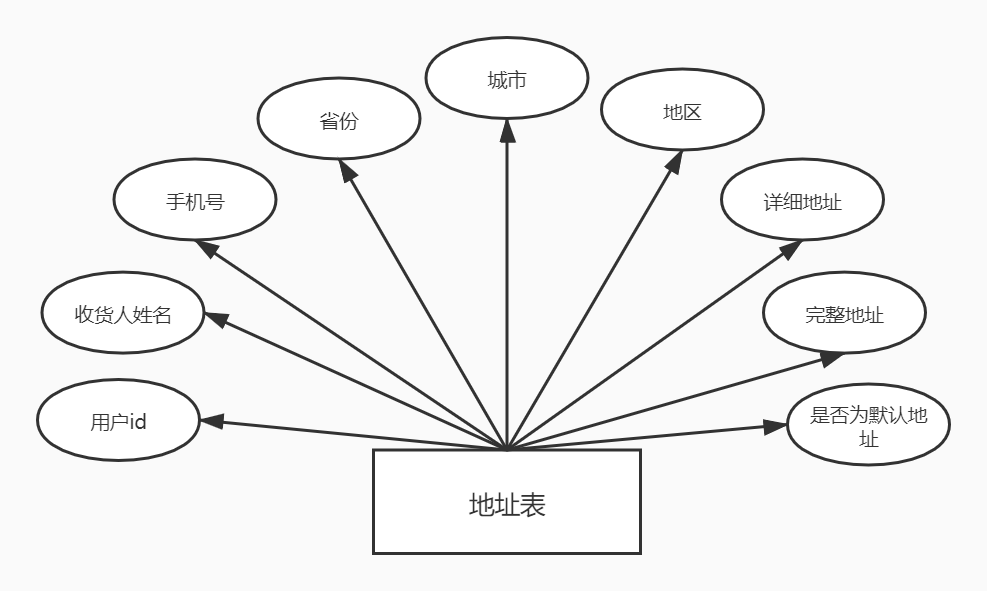
1. 购物车实体E-R图



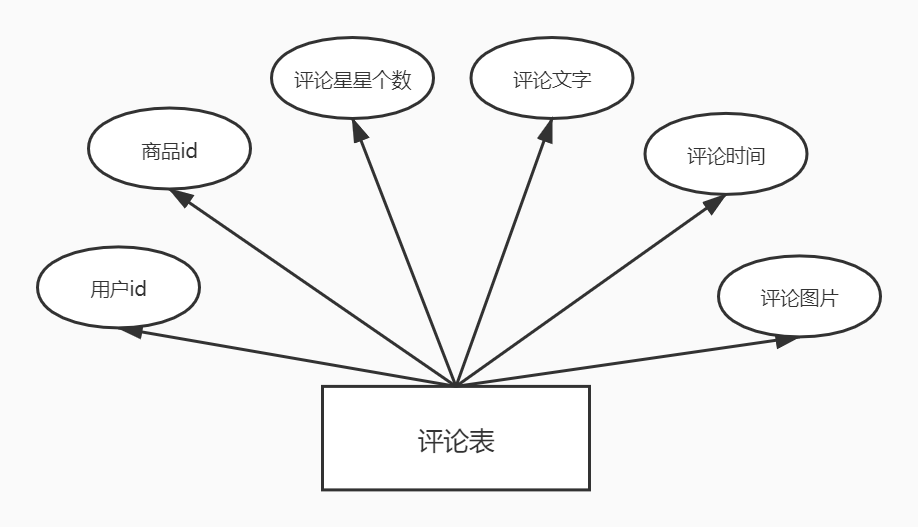
1. 订单实体E-R图



1. 收货地址实体E-R图



1. 评论实体E-R图



4.3 数据库逻辑结构设计

数据库的概念模型相对于程序来说是独立的，我们需要根据程序涉及到的业务需求来对数据库的表结构进行设计和修改。数据库的表结构也应该遵循E-R模型的原则：即每个实体都要转换成一个关系，所有的主键不为空，且不能重复，对于有一对一或一对多的关联表应该设置外键进行约束和连接。

用户表用于存储登录用户信息，结构如表4-1所示

| **字段名** | **类型** | **长度** | **允许为null** | **是否主键** | **说明** |
| --- | --- | --- | --- | --- | --- |
| id | int | 50 | 否 | 是 | 用户id，唯一主键 |
| username | varchar | 255 | 否 | 否 | 账号 |
| password | varchar | 255 | 否 | 否 | 密码 |
| email | varchar | 255 | 否 | 否 | 邮箱，用于找回密码 |
| avatar | varchar | 255 | 是 | 否 | 头像 |
| nickname | varchar | 255 | 是 | 否 | 昵称 |
| sex | varchar | 255 | 是 | 否 | 性别 |
| birthday | varchar | 255 | 是 | 否 | 生日 |
| address | varchar | 255 | 是 | 否 | 家庭住址 |
| desc | varchar | 255 | 是 | 否 | 个性签名 |

表4-1

商品表用于存储所有展示商品的信息，结构如表4-2所示

| **字段名** | **类型** | **长度** | **允许为null** | **是否主键** | **说明** |
| --- | --- | --- | --- | --- | --- |
| id | Int | 50 | 否 | 是 | id，唯一主键 |
| brand\_id | int | 50 | 否 | 否 | 所属分类 |
| category\_id | int | 50 | 否 | 否 | 所属分类 |
| name | varchar | 255 | 否 | 否 | 名称 |
| list\_pic\_url | varchar | 255 | 否 | 否 | 展示图片 |
| market\_price | decimal | 10 | 否 | 否 | 单价 |
| origin\_price | decimal | 10 | 是 | 否 | 原价 |
| swiper\_list\_pic\_url | varchar | 4096 | 否 | 否 | 轮播图展示图片 |
| show\_list\_pic\_url | varchar | 4096 | 否 | 否 | 大图介绍 |
| is\_hot | varchar | 255 | 是 | 否 | 是否热门 |
| is\_new | varchar | 255 | 是 | 否 | 是否新品 |
| delivery\_place | varchar | 255 | 否 | 否 | 发货地 |

表4-2

地址表用于存储所有用户的收货地址信息，结构如表4-3所示

| **字段名** | **类型** | **长度** | **允许为null** | **是否主键** | **说明** |
| --- | --- | --- | --- | --- | --- |
| id | int | 50 | 否 | 是 | 地址id，唯一主键 |
| user\_id | int | 50 | 否 | 否 | 用户id |
| name | varchar | 255 | 否 | 否 | 收货人姓名 |
| tel | varchar | 255 | 否 | 否 | 收货人手机号 |
| province | varchar | 255 | 否 | 否 | 省份 |
| city | varchar | 255 | 否 | 否 | 城市 |
| county | varchar | 255 | 否 | 否 | 地区 |
| addressDetail | varchar | 255 | 否 | 否 | 详细地址 |
| address | varchar | 255 | 否 | 否 | 完整地址 |
| areaCode | varchar | 255 | 否 | 否 | 地址编号 |
| isDefault | tinyint | 1 | 否 | 否 | 是否为默认地址 |

表4-3

品牌表用于存储品牌的信息，结构如表4-4所示

| **字段名** | **类型** | **长度** | **允许为null** | **是否主键** | **说明** |
| --- | --- | --- | --- | --- | --- |
| id | int | 50 | 否 | 是 | 品牌id，唯一主键 |
| list\_pic\_url | varchar | 255 | 是 | 否 | 展示列表图片 |
| name | varchar | 255 | 是 | 否 | 品牌名称 |
| banner\_pic\_url | varchar | 255 | 是 | 否 | 背景图 |
| banner\_logo | varchar | 255 | 是 | 否 | logo |
| desc | varchar | 255 | 是 | 否 | 品牌简介 |
| is\_hot | varchar | 255 | 是 | 否 | 是否热门 |

表4-4

分类表用于存储商品分类信息，结构如表4-5所示

| **字段名** | **类型** | **长度** | **允许为null** | **是否主键** | **说明** |
| --- | --- | --- | --- | --- | --- |
| id | int | 50 | 否 | 是 | 分类id，唯一主键 |
| name | varchar | 255 | 否 | 否 | 分类名称 |
| list\_pic\_url | varchar | 255 | 否 | 否 | 分类图片 |
| is\_parent | int | 50 | 否 | 否 | 是否是一级分类 |
| parent\_id | int | 50 | 是 | 否 | 父级分类id |
| is\_hot | varchar | 255 | 是 | 否 | 是否热门 |

表4-5

购物车表用于存储用户添加购物车商品信息，结构如表4-6所示

| **字段名** | **类型** | **长度** | **允许为null** | **是否主键** | **说明** |
| --- | --- | --- | --- | --- | --- |
| id | int | 50 | 否 | 是 | 购物车id，唯一主键 |
| user\_id | int | 50 | 否 | 否 | 用户id |
| good\_id | int | 50 | 否 | 否 | 商品id |
| list\_pic\_url | varchar | 255 | 是 | 否 | 登录邮箱，用于找回密码 |
| name | varchar | 255 | 是 | 否 | 商品名称 |
| market\_price | varchar | 255 | 是 | 否 | 商品价格 |
| number | varchar | 255 | 是 | 否 | 加购数量 |

表4-6

评论表主要是用于存储用户评论信息，结构如表4-7所示

| **字段名** | **类型** | **长度** | **允许为null** | **是否主键** | **说明** |
| --- | --- | --- | --- | --- | --- |
| id | int | 50 | 否 | 是 | 评论id，唯一主键 |
| user\_id | varchar | 255 | 否 | 否 | 评论用户id |
| good\_id | varchar | 255 | 否 | 否 | 评论商品id |
| star | varchar | 255 | 是 | 否 | 评论星星数量 |
| content | varchar | 255 | 是 | 否 | 评论内容 |
| add\_time | varchar | 255 | 是 | 否 | 评论时间 |

表4-7

评论图片表主要存储用户评论图片信息，结构如表4-8所示

| **字段名** | **类型** | **长度** | **允许为null** | **是否主键** | **说明** |
| --- | --- | --- | --- | --- | --- |
| id | int | 50 | 否 | 是 | 评论图片id，唯一主键 |
| comment\_id | int | 50 | 否 | 否 | 对应的评论id |
| list\_pic\_url | varchar | 255 | 是 | 否 | 评论图片 |

表4-8

订单表主要存储所有用户订单信息，结构如表4-9所示

| **字段名** | **类型** | **长度** | **允许为null** | **是否主键** | **说明** |
| --- | --- | --- | --- | --- | --- |
| id | int | 50 | 否 | 是 | 订单id，唯一主键 |
| user\_id | int | 50 | 是 | 否 | 用户id |
| order\_status | varchar | 255 | 是 | 否 | 订单状态 |
| good\_num | varchar | 255 | 是 | 否 | 订单商品数量 |
| order\_price | varchar | 255 | 是 | 否 | 订单价格 |
| create\_time | varchar | 255 | 是 | 否 | 创建时间 |
| pay\_time | varchar | 255 | 是 | 否 | 支付时间 |
| order\_no | varchar | 255 | 是 | 否 | 订单编号 |
| address\_id | varchar | 255 | 是 | 否 | 收货地址id |
| remaining\_time | varchar | 255 | 是 | 否 | 剩余支付时间 |

表4-9

4.3 数据库连接原理

在Node服务器中，使用express框架连接数据库就显得非常简洁易懂了，我们不需要配置大量的参数，只需要下载mysql中间件，然后通过调用中间件的createConnection方法，并将数据库信息传入方法中即可建立连接，具体代码实现如下：

const mysql = require('mysql');

const conn = mysql.createConnection({

host: "localhost",

user: "root",

password: "root",

database: "graduation\_design"

});

conn.connect();

module.exports = conn;

由于很多方法都是mysql中间件为我们封装好了的，所以我们无需考虑如何配置很多参数，而是可以把全部的精力都放在业务逻辑的处理上。

5 系统详细设计

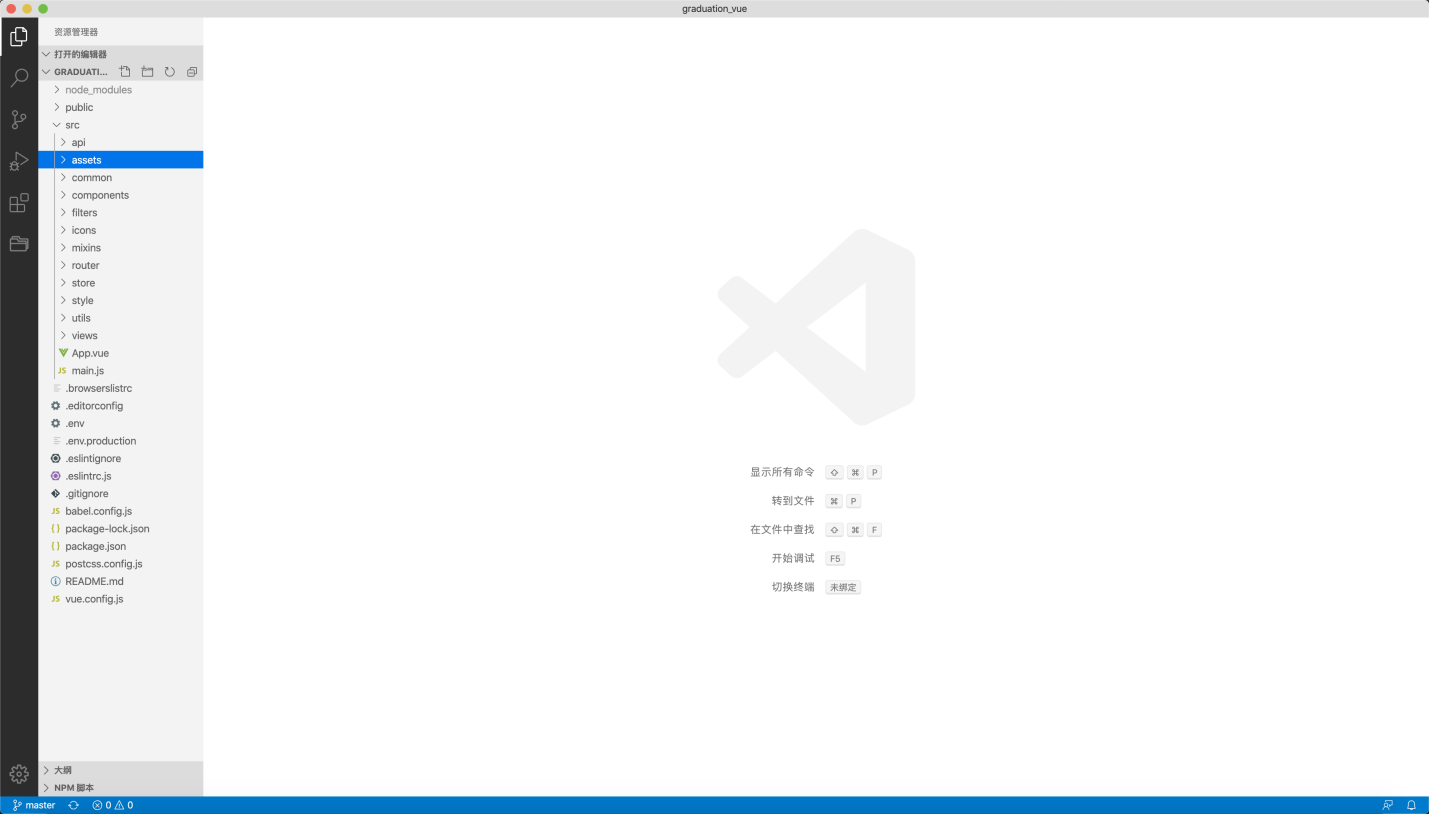
5.1 系统操作流程图

系统操作流程图

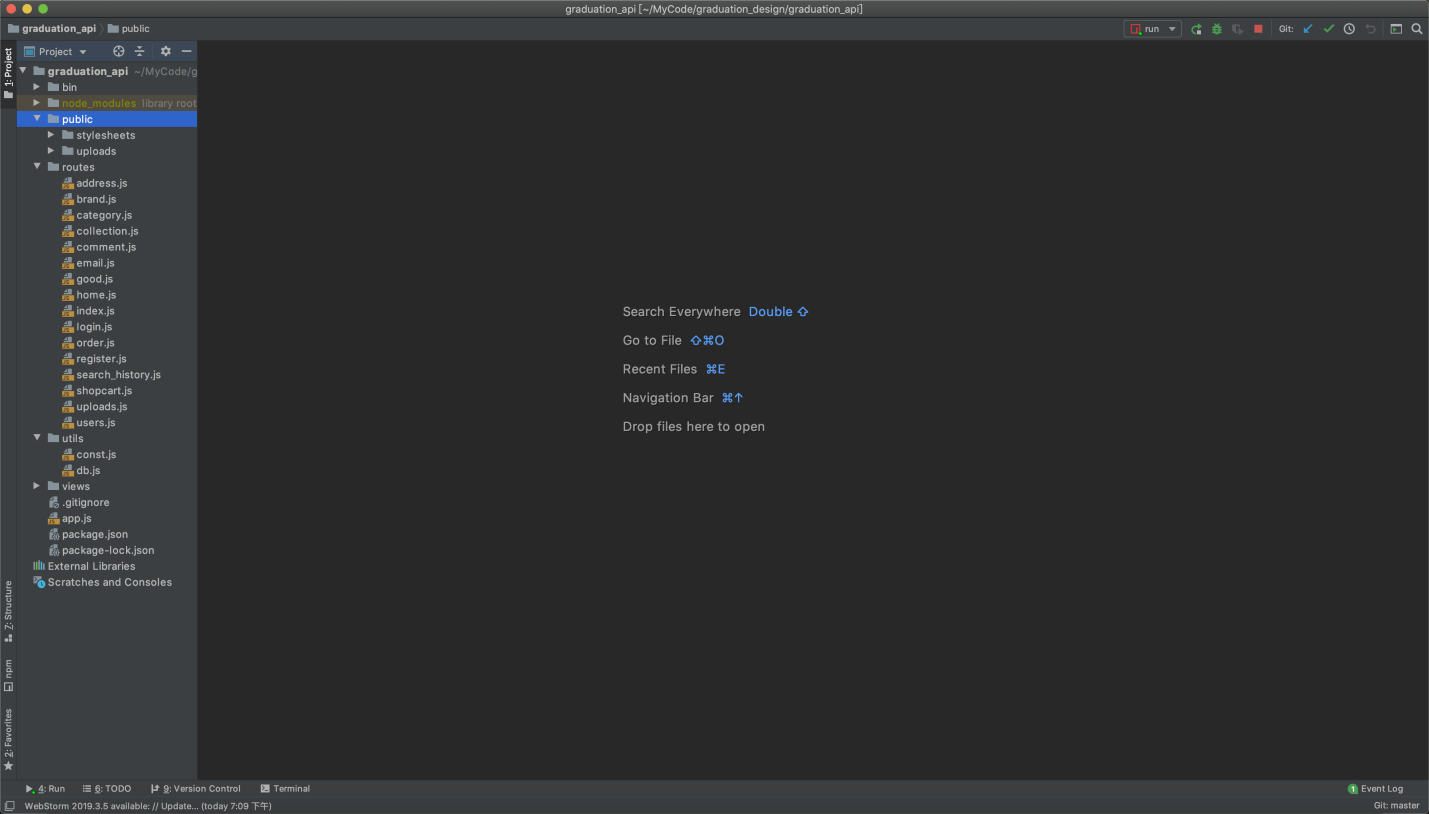
5.2 系统实现的具体逻辑

本程序使用的是前后端分离的方式和理念进行开发的。这种开发方式通常都会遇到跨域的问题。跨域是大多数浏览器中都有一个同源策略，只允许相同协议、域名和端口下的服务之间可以通信。我使用的跨域方式是在webpack中配置Vue在开发模式下启动服务时，通过代理和路径重写的方式将前端请求地址代理到服务器地址上，服务器对前端所有的请求做cors处理，这样就能够跨域请求数据了。当然还可以用nginx配置反向代理，这种方式也是很常见的。

在开发的过程中分别使用了vue-cli3和express作为前后端的脚手架。前端中项目中所有的业务全放在src文件夹中完成。src下对于处理不同功能逻辑的代码也是要做详细的区分的。作为整个项目的入口文件App.vue和main.js，负责将所有的功能文件整合到一起。具体结构如下图所示：



服务器项目也是要根据不同的功能对整个项目文件进行细分，根目录下的app.js作为整个项目的入口文件，utils文件夹用来放置封装好的工具类，public中存放静态资源，比如样式文件和服务器图品文件，最后，把每个接口业务处理方法放在router中集体管理。具体目录格式如下：



1. 注册功能

本系统的注册功能逻辑较为简单，用户需要输入一个有效邮箱，可以在找回密码时接收验证码，然后设置登录账号和密码，输入动态生成的验证码。验证码主要是为了防止垃圾注册等恶意行为。点击注册之后，后端接收到数据首先对验证码进行校验。其次判断注册的账号是否已经存在，如果数据库中已经存在该账号则提示用户账号已经注册，并引导用户直接去登录或者换个账号，反之则将数据插入数据库中，并提示用户注册成功，跳转到登录页面。具体代码如下：

handleSubmit() {

if (!this.form.email) {

this.$toast('请输入邮箱')

} else if (!this.form.username) {

this.$toast('请设置登录账号')

} else if (!this.form.password) {

this.$toast('请设置登录密码')

} else if (!this.form.captcha) {

this.$toast('请输入验证码')

} else if (!email.test(this.form.email)) {

this.$toast('请输入正确的邮箱地址')

} else {

this.$api.register.createAccount(this.form).then(res => {

if (res.code === '000000' || res.code === '111111') {

this.$router.push({ path: '/login' })

} else if (res.code === '111000') {

this.form.captcha = ''

this.captchaUrl = `${this.captchaUrl}?time=${new Date()}`

}

this.$toast(res.message)

}).catch(err => {

console.log(err)

})

}

}



1. 登录功能

因为程序是针对两种用户使用的。对于浏览商品、搜索商品、查看商品是任何人都可以操作的，但是涉及到个人的比如添加购物车、支付下单、评论等功能就要求用户必须登录了。这部分逻辑是由前端路由来控制的，这部分放到后面讲。首先来说一下登录流程，在用户输入账号和密码后，服务器拿到前端的数据后查询数据库，如果账号不存在或者密码不正确则提示用户去注册或重新输入密码。反之，用户登录成功后，在服务端使用jsonwebtoken中间件生成一个token字符串，token中除了存放了用户id和用户名以外，还要传入一个token秘钥，需要特别注意的是这个token秘钥是放在服务器里不能给任何人看的，因为只有通过这个秘钥才能正确解析token，然后在放入一个有效期限。最后将经过中间价加密生成的token返回给前端。具体代码如下：

const jwt = require('jsonwebtoken');

const {secret} = require('../utils/const');

router.post('/', (req, res, next) => {

const {username, password} = req.body;

const sql = `SELECT \* FROM user where username = '${username}'`;

conn.query(sql, (err, results, fields) => {

if (err) throw err;

if (results[0]) {

// 生成token

const token = jwt.sign(

{\_id: results[0].\_id, username: results[0].username},

secret,

{expiresIn: 7 \* 24 \* 60 \* 60 \* 1000}

);

res.status(200).json({code: "000000", token: token, message: "登录成功"});

}

});

});



前端在接收到token秘钥之后会将token秘钥放在vuex中做全局状态管理。但是vuex中的数据会随着页面的刷新重新初始化，这就造成了用户不能持久化登录的问题，所以我又将token存到了浏览器的localStorage中，每次vuex初始化的时候都会先去localStorage中取，只要不手动删除本地缓存就会一直存在。具体代码如下：

const state = {

token: localStorage.getItem('token')

}

const mutations = {

GET\_TOKEN: (state, token) => {

state.token = token

}

}

const actions = {

login({ commit }, data) {

return new Promise((resolve, reject) => {

login(data).then(res => {

if (res.code === '000000') {

localStorage.setItem('token', res.token)

commit('GET\_TOKEN', res.token)

}

resolve(res)

}).catch(err => {

reject(err)

})

})

}

}

前面我们有讲到部分操作是需要用户登录才可以的，这里我做了双重验证。我们都知道前端单页面应用是通过路由实现组件之间的切换的，所以我给每一个路由都在mate对应的对象中设置auth: true这个键值对，用来表示这个路由组件是需要用户登录才可以访问的。然后调用路由的前置守卫beforeEach，根据定义好的auth判断，如果需要登录，再去验证vuex中的token是否存在，如果为空，则跳转到登录页面，否则全部通过。具体代码如下：

import Vue from 'vue'

import Router from 'vue-router'

import store from 'store/index'

import routes from './routes'

Vue.use(Router)

const router = new Router({ routes })

router.beforeEach((to, from, next) => {

const auth = to.meta.auth

const token = store.getters['token'] || localStorage.getItem('token')

if (auth) { // 需要登录

if (token) {

next()

} else {

next({

path: '/login',

query: {

redirect: to.fullPath

}

})

}

} else {

next()

}

})

前端验证再完美都算不上特别安全，因为我只验证了token是否存在，并没有且没有办法去验证token的有效性。所以，对于页面中的任何业务上的请求都需要进一步拦截，拦截的点就在于token这把钥匙。我们在登录成功后将返回的token放在axios的请求拦截器的请求头中，后端接收到请求后会先验证请求头中token的有效性。我这里使用的是express-jwt中间件，它使用设置的秘钥对请求头中的token进行解析，还可以设置白名单以过滤掉不需要验证的请求。具体代码如下：

前端请求拦截器代码：

import axios from 'axios'

import store from 'store/index'

const service = axios.create({})

service.interceptors.request.use(

config => {

const token = store.getters.token || localStorage.getItem('token')

if (token) {

config.headers['authorization'] = `Bearer ${token}`

}

return config

}

)

服务端请求拦截器：

const expressJWT = require("express-jwt");

const secret = "jwn\_secret";

app.use(expressJWT({

secret: secret

}).unless({

// 不需要token验证白名单地址

path: [ /^\/login.\*$/, .... ]

}));

app.use((err, req, res, next) => {

if (req.user) {

next()

} else {

// token失效处理函数

}

});

这种双重验证的方式在开发工作中是很常见的，就算了有人篡改了token也会因为解析失败无法进行操作，从而保障了账户安全。

1. 忘记密码功能

用户点击忘记密码后会提示用户数据忘记密码的账号和绑定的邮箱，如果这两个都正确，则开始发送邮箱验证码。首先开启一个SMTP连接池，随机生成的6位验证码，并全局保存，生成邮箱内容模板并将验证码拼接进去，最后配置一下发送邮箱参数，调用连接池的sendMail方法发送验证码。这里我还对验证码有效期简单的做了一层处理，就是在邮箱发送成功后，设置一个延时器，5分钟将验证码状态设为过期状态，此时用户再输入这个验证码则会提示验证码无效，请重新发送验证码。代码详细内容如下：

// 开启一个SMTP连接池

const transport = nodeMailer.createTransport(smtpTransport({

host: 'smtp.qq.com', // QQ邮箱主机，固定

secure: true, // 使用SSL

secureConnection: true, // 使用SSL

port: 465, // SMTP端口

auth: {

user: '1@qq.com', // 自己邮箱

pass: 'rgpfacplc' // 开启SMPT获取的密码

}

}));

// 发送验证码

router.post('/send', (req, res, next) => {

const {username, email} = req.body;

const sql = `select email from user where username = '${username}'`;

conn.query(sql, (err, results, fields) => {

if(err) throw err;

if(results[0]) {

// 验证邮箱是否是绑定的邮箱

if(results[0].email === email) {

let code = getValidCode();

golbol.code = code;

let email\_content = initFormat(code);

const mailOptions = {

from: '1@qq.com',

to: req.body.email,

subject: "购物网",

text: '',

html: email\_content

};

transport.sendMail(mailOptions, (err, response) => {

if(err) {

res.json({ code: '111111', message: "验证码发送失败" });

}else{

res.json({ code: "000000", message: "验证码发送成功" });

// 清除上一次延时器

clearTimeout(golbol.timer);

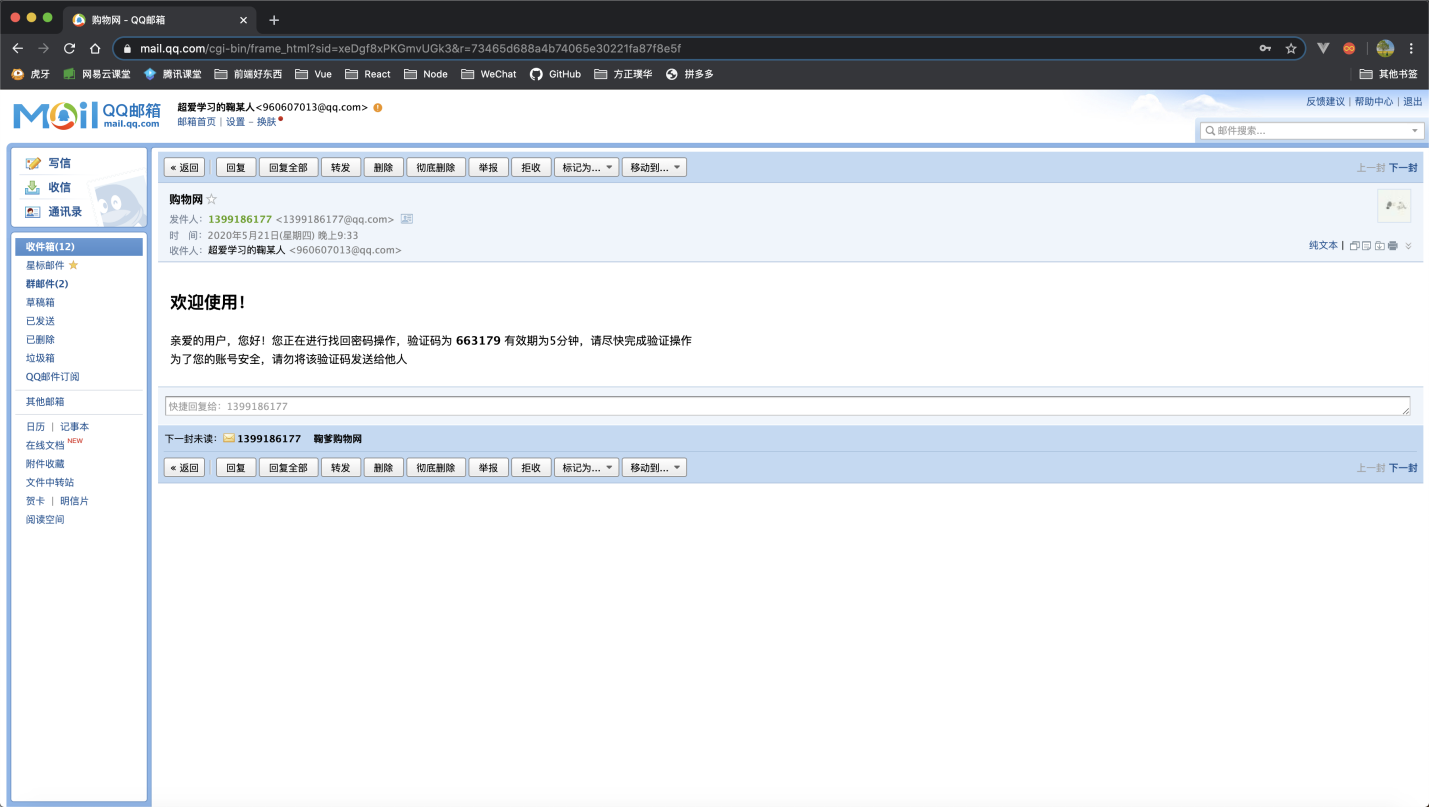
golbol.expired = false;

golbol.timer = setTimeout(() => {

// 五分钟后状态为过期

golbol.expired = true

}, 1000 \* 60 \* 5)}});}});});



用户输入邮箱验证码之后，点击下一步，如果验证码正确成功进入到重置密码环节，用户输原密码和两遍新密码后，前端会先判断两遍新密码的一致性，然后服务器接收到参数后将新密码更新到数据库中即完成整个操作。

1. 首页、分类、商品列表功能

首页中主要有轮播图，热门分类、热门品牌和商品列表这几部分。其中花费时间较长的还是用在处理页面展示效果和样式美化上面。用到的最多且效果最直观的就是弹性盒布局，它的特点在于会自动根据视口的大小对页面中的元素组件进行排版。对于样式和处理的业务都相同的组件做了全局的封装，通过父组件对子组件的引入、遍历、父子组件传值的方式动态渲染页面，这样就可以做到一处开发多处使用的效果，而且如果UI上有修改只需要对公共组件做处理，更利于维护。

商品列表展示会有一下几种情况。根据搜索框中的关键字模糊查询；根据分类查询；根据品牌查询；找相似查询（这里也是根据分类做筛选）；还有分页查询。这几种情况我分别用不同的字段作区分，服务端在接受请求后会对参数进行判断，最后拼接为带条件的SQL语句查询数据库，并将正确的结果返回。

分类组件在设计上分为两部分，左边是一级分类菜单，右边是二级分类内容。为了让菜单和内容更好的联动起来，我使用了better-scroll这款插件辅助实现了这种效果。首先初始化better-scroll对象，然后把每个分类整体内容距离页面的高度全记录下来并按顺序添加到数组中，记录点击的一级菜单的索引，然后根据索引取到记录高度的数组中对应的数据，然后调用scroll中的scrollTo方法，滚动到取出的值的位置；当用户鼠标滚动右边分类内容时，scroll对象会实时监听滚动的位置，只需要判断滚动的距离相对于数组中的索引是什么，然后用这个索引去给菜单栏对应的位置添加样式；这样即可很好地实现左右联动的功能，操作体验也会提升很多。具体代码如下：

对better-scroll进行二次封装

\_initBScroll() {

this.scroll = new BScroll(this.$refs.wrapper, {

probeType: this.probeType,

click: true,

pullDownRefresh: this.pullDownRefresh,

pullUpLoad: this.pullUpLoad,

})

this.scroll.on('scroll', (pos) => {

this.$emit('scroll', pos)

})

},

scrollTo(x, y, time = 300) {

this.scroll.scrollTo(x, y, time)

},

scrollToElement(el, time, offsetX, offsetY) {

this.scroll.scrollToElement(el, time, offsetX, offsetY)

}

分类组件：

// 点击菜单选项右边滚动

handleMenuClick(index) {

this.scrollY = this.categoryItemTop[index]

this.$refs.categoryDetail.scrollTo(0, -this.scrollY)

}

// 右边滚动时，联动左边跟着滚动

menuScroll(index) {

const menuItem = this.$refs.categoryMenuItem[index]

this.$refs.categoryMenu.scrollToElement(menuItem, 300, 0, -150)

}

// 监听右边分类内容列表位置

categoryItemScroll(pos) {

this.scrollY = Math.abs(Math.round(pos.y))

// 右边滚动左边联动滚动

this.menuScroll(this.activeMenuIndex)

}

// 获取分类内容距离顶部高度

getCategoryItemTop() {

const temp = [0]

const items = document.getElementsByClassName('categoryItem')

// 将伪数组转化为真数组

Array.prototype.slice.call(items).forEach((item, index) => {

if (index === items.length - 1) {

// ref获取到的vue对象，通过$el取到原生组件

item.style.paddingBottom = `${this.$refs.categoryDetail.$el.clientHeight - item.clientHeight + 20}px`

}

// 累加高度添加到数组中

temp.push((temp[temp.length - 1] + item.clientHeight))

})

this.categoryItemTop = temp

}





1. 购物车功能

这个模块是整个网上购物商城较为重要的部分。在每次执行添加购物车操作时，首先需要根据请求头中的token判断出是哪个用户添加的，然后判断该用户有没有添加过该商品，如果有则在原有的商品数量上累加，如果没有则将新添加购物车的数据执行插入操作。还需要注意的是必须要给每条购物车表消息添加user\_id，这样才能做到每个用户不会相互影响。具体代码如下：

onAddCartClicked(value) {

const data = {

good\_id: this.goodDetail.id,

list\_pic\_url: this.goodDetail.list\_pic\_url,

name: this.goodDetail.name,

market\_price: this.goodDetail.market\_price,

number: value.selectedNum

}

this.$api.shopcart.addShopCartGood(data).then(res => {

if (res.code === '000000') {

this.$toast.success(res.code)

}

})

}

router.post("/add", (req, res, next) => {

const user\_id = req.user.\_id;

const {good\_id, list\_pic\_url, name, market\_price, number} = req.body;

const sql = `select \* from shop\_cart where user\_id = ${user\_id} and good\_id = ${good\_id}`;

conn.query(sql, (err, results, fields) => {

if (err) throw err;

if (results[0]) {

const sql = `update shop\_cart set number = ${number + results[0].number} where id = ${results[0].id}`;

conn.query(sql, (err, results, fields) => {

if (err) throw err;

res.status(200).json({

code: "000000",

message: "添加购物车成功"

});

})

} else {

const sql = `insert into shop\_cart values (null, '${user\_id}', '${good\_id}', '${list\_pic\_url}', '${name}', '${market\_price}', '${number}')`;

conn.query(sql, (err, results, fields) => {

if (err) throw err;

res.status(200).json({

code: "000000",

message: "添加购物车成功"

});

在页面渲染购物车列表时，我们要给每个购物车商品信息对象初始化一个checked：false的属性，目的是绑定购物车商品是否被选中的状态。正是为了绑定这个属性，让我采了很大的坑。因为在vue中数据都是双向绑定的，通过普通的obj.key=value这种赋值方式是没有用的，这种方式只是把属性赋给了对象，却没有挂载到vue实例上，所以无论怎么点击页面中的选择框，都不会得到想要的效果。后来看了一下vue官方文档，如果要给一个对象赋值是要通过Vue.$set()这种方式就可以给一个响应式的对象添加一个响应的属性了。具体代码如下：

getShopCartList() {

this.$api.shopcart.getShopCartList().then(res => {

this.shopCartList = res.data

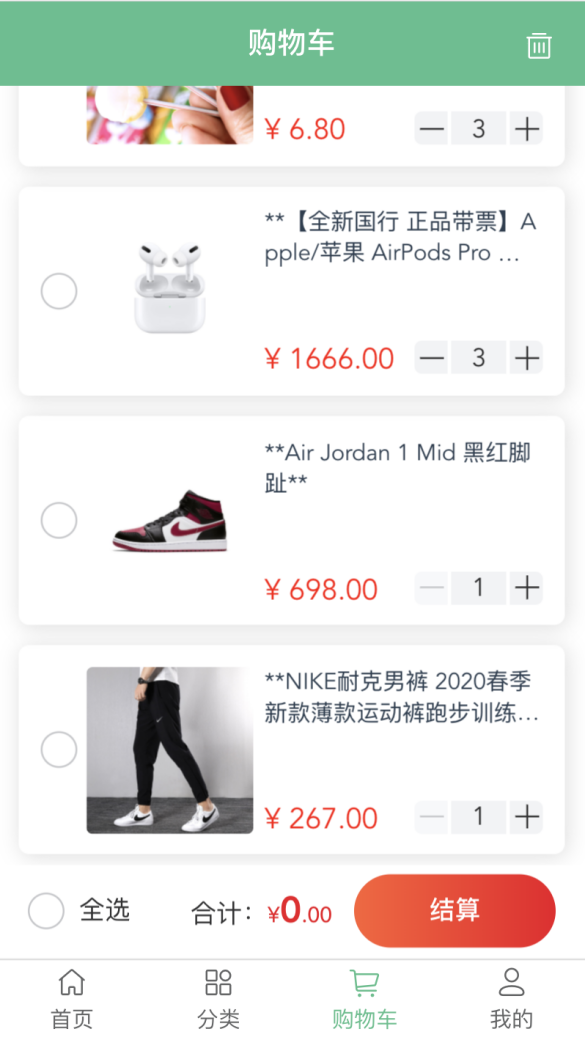
this.shopCartList.forEach(item => {

this.$set(item, 'checked', false)

})

})

}



1. 订单功能

用户在购物车点击结算之后，会把选中的商品带入确认订单页面。这里会自动给用户选择一个默认配送地址，当然也允许用户手动修改，点击去支付按钮跳到输入支付密码页面，这时订单已经生成且为未支付状态，购物车中的与相应的商品也会被删除。

关于进入支付页面我做了细化处理，首先会判断该用户是否已经设置过支付密码，如果没有，则引导用户去设置密码，否则订单永远无法支付。反之，则会根据路由携带的参数判断该操作时支付还是修改密码，组件在根据这个参数去引导用户一步步向下进行。具体代码如下：

<template>

<!-- 设置密码 -->

<div v-if="!hasPassword" class="setPassword">

<my-password v-if="passwordOne" ref="myPassword1" info="请设置支付密码" @commit="setPasswordOne" />

<my-password v-else info="请确认设置的支付密码" @commit="setPasswordAgain" />

</div>

<!-- 输入支付密码 -->

<div v-else-if="orderId !== ''" class="payPassword">

<my-password info="请输入支付密码" @commit="inputPayPassword" />

</div>

<!-- 修改密码 -->

<div v-else class="updatePassword">

<my-password v-if="passwordOld" ref="myPassword2" info="请输入旧密码" @commit="inputOldPwd" />

<my-password v-else-if="passwordOne" ref="myPassword3" info="请设置新的支付密码" @commit="inputNewPwd" />

<my-password v-else info="请再次确认新的支付密码" @commit="inputNewPwdAgain" />

</div>

</div>

</template>

<script>

import myPassword from './password'

export default {

components: {

myPassword

},

data() {

return {

// 是否已经设置过密码

hasPassword: false,

passwordOne: true,

setPassword: '',

// 修改密码

oldPassword: '',

passwordOld: true,

// 标题

title: '支付密码',

// 支付订单id

orderId: ''

}

},

created() {

this.checkHasPassword()

this.orderId = this.$route.params.id || ''

if (this.orderId) {

this.title = '请输入支付密码'

} else {

this.title = '修改密码'

}

},

methods: {

// 设置密码--第一次输入

setPasswordOne(pwd) {

this.setPassword = pwd

this.passwordOne = false

this.$refs.myPassword1.value = ''

},

// 设置密码确认密码

setPasswordAgain(pwd) {

if (this.setPassword === pwd) {

this.$api.user.setPayPassword({ password: pwd }).then(res => {

if (res.code === '000000') {

this.$toast.success('支付密码设置成功')

}

})

} else {

this.$toast.fail('两次输入的密码不一致')

}

},

// 修改密码--输入旧密码

inputOldPwd(pwd) {

this.passwordOld = false

this.$refs.myPassword2.value = ''

this.oldPassword = pwd

},

// 修改密码--输入新密码

inputNewPwd(pwd) {

this.passwordOne = false

this.$refs.myPassword3.value = ''

this.setPassword = pwd

},

// 修改密码--确认新密码

inputNewPwdAgain(pwd) {

if (this.setPassword === pwd) {

this.$api.user.updatePayPassword({ oldPassword: this.oldPassword, password: this.setPassword }).then(res => {

if (res.code === '000000') {

this.$toast.success('密码修改成功')

} else {

this.$toast.fail('原密码输入不正确')

}

this.$router.replace({ name: 'personal' })

})

} else {

this.$toast.fail('两次输入的密码不一致')

}

},

// 支付密码

inputPayPassword(password) {

this.$api.user.inputPayPassword({ password }).then(res => {

if (res.code === '000000') {

this.payOrder()

} else {

this.$toast.fail(res.message)

}

})

},

checkHasPassword() {

this.$api.user.checkHasPassword().then(res => {

if (res.code === '000000') {

this.hasPassword = res.data

if (!res.data) {

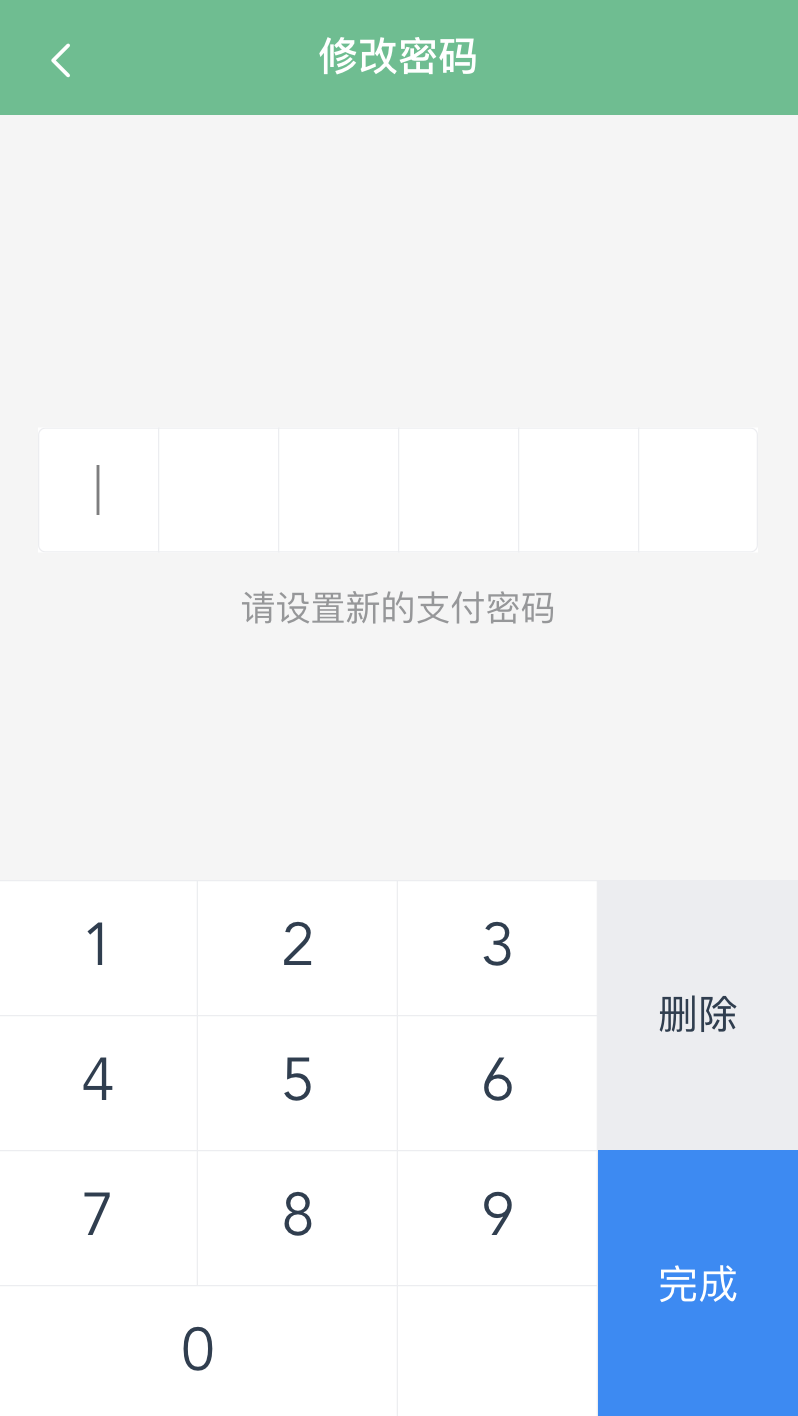
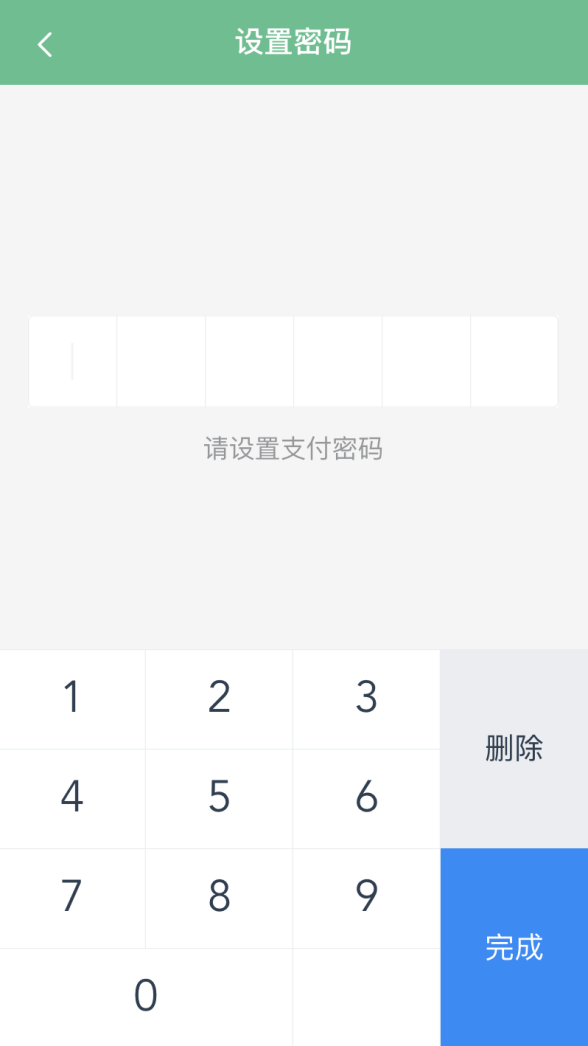
this.title = '设置密码'

}

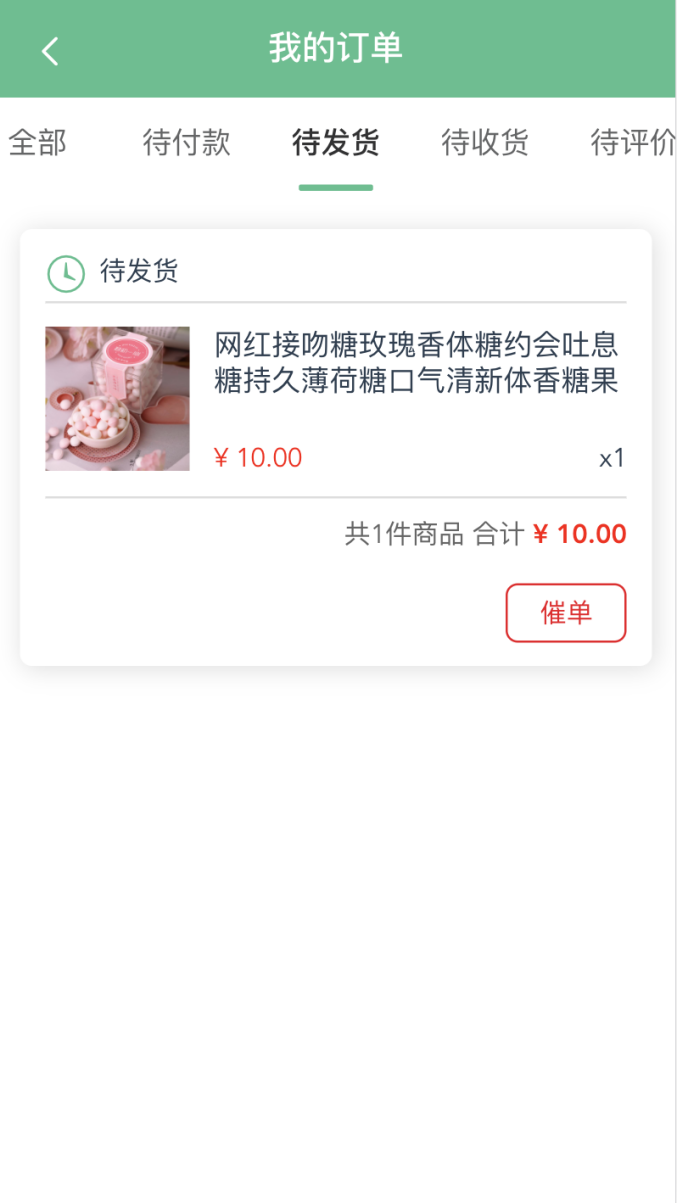
}

})

</script>



如果订单在生成后没有立即支付，则会为提醒用户请在30分钟之内支付，超过30分钟则被自动取消。这里是通过setTimeOut延时器来实现的，也就是说在订单生成后开始倒计时，30分钟后，会执行将订单状态设置为已取消的装填。本程序中订单的不同状态是通过order\_status字段控制的。为了体验到不同的订单状态效果，采用了通过点击不同的业务按钮来直接改变订单状态这种简单的方式实现的。



1. 评论功能

这部分功能分别在添加评论和获取评论列表时踩过坑。先来说一下添加评论功能，再添加评论时需要将评论用户信息、评论图片信息、评论文字和评论图片提交给服务器，也因为图片文件和文字的格式不一致导致无论前端还是后端处理起来都相当的麻烦。

后来我想到的解决办法是选择完图片后，先将图片提交到服务器，存在服务器uploads文件夹下，并返回给前端一个服务器图片地址，这样在提交订单的时候就像正常的像服务器发送数据即可，也不需要对参数做判断以及特殊处理。具体代码如下：

const formidable = require('formidable');

const path = require('path');

const picUrl = 'http://localhost:3000/uploads/';

router.post('/', (req, res, next) => {

const form = new formidable.IncomingForm();

// 图片存放地址

form.uploadDir = path.join(\_\_dirname, '../public/uploads');

// 保留文件名

form.keepExtensions = true;

form.parse(req, (err, fields, files) => {

if (err) throw err;

res.status(200).json({

code: '000000',

message: "上传成功",

data: picUrl + path.basename(files.file.path)

});

});



因为在数据库表结构设计中，评论图片是用一个单独的表存放的，所以在获取评论列表时，首先在评论主表中查询到所有的评论信息，然后遍历评论信息数组，根据每个评论的id在评论图片表中查询，并将查询结果放在每个评论对象中。看似很平常的操作，却让我浪费了两天时间，在写的过程中，发现返回给前端的数据中并没有评论图片，但是评论图片确实查到了，而且查到之后也赋给了评论对象。最后了解到由于node中的所有的操作都是异步的，所以之前的写法在数据还没彻底遍历赋值时候就已经返回给前端了。这也是完全异步带来的一些麻烦，不过好在node中有async这个中间件。通过调用async.map(arr, (item, callback) => {})这种方式遍历，然后在最后一个评论的图片已经赋值完后，在回调函数中就能返回完整的数据了。具体代码如下：

router.get('/list/withGood', (req, res, next) => {

const good\_id = req.query.id;

const sql = `select \* from comment where goods\_id = ${good\_id} order by id DESC`;

conn.query(sql, (err, results, fields) => {

if (err) throw err;

let len = results.length;

let index = 0;

async.map(results, (comment, callback) => {

const id = comment.id;

const user\_id = comment.user\_id;

const sql = `select list\_pic\_url from comment\_pic where comment\_id = ${id}`;

conn.query(sql, (err, results, fields) => {

if (err) throw err;

comment.list\_pic\_url = [];

results.forEach(value => {

comment.list\_pic\_url.push(value.list\_pic\_url);

index += 1;

callback(null, comment)

}, (err, results) => {

if (index === len) {

res.status(200).json({

code: "000000",

message: "成功",

data: {list: results}})}});});});

6 程序运行与测试

6.1 测试的目的和意义

程序测试工作是一般来说是一个程序开发周期中十分漫长且至关重要的一部分。测试环节最直接的目的就是对程序的质量进行检测，通过从小到大范围对程序进行压力测试，尽可能的彻底检查出程序中的问题，提高程序的完整性和可靠性，做到程序能够和需求规格中的说明保持一致。在测试中发现的问题越多，从一定角度来说能够保证上线之后的问题就越少。

6.2 程序测试分析

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 模块名称 | 测试用例 | 预期结果 | 实际结果 | 是否通过 |
| 注册模块 | 邮箱：null  用户名：null 密码：null  验证码：null | 弹出错误提示，请输入邮箱、账号、密码、验证码 | 注册失败，请输入邮箱、账号、密码、验证码 | 通过 |
| 注册模块 | 邮箱：960607013@qq.com  用户名：test 密码：test  验证码：错误验证码 | 弹出错误提示，验证码错误，请重新输入，并更新图形验证码 | 注册失败，提示输入正确的验证码，自动更新图形验证码 | 通过 |
| 注册模块 | 邮箱：960607013@qq.com  用户名：test 密码：test  验证码：错正确验证码 | 提示注册成功，跳转到登录页面引导用户登录 | 注册成功，页面跳转到登录页面 | 通过 |
| 登录模块 | 账号：null  密码：null | 提示登录失败，请输入账号密码 | 登录失败，输入账号密码 | 通过 |
| 登录模块 | 账号：qwer  密码：test | 提示账号不存在 | 登录失败，提示账号不存再 | 通过 |
| 登录模块 | 账号：test  密码：qewr | 提示密码错误 | 登录失败，提示密码错误，请重新输入密码 | 通过 |
| 登录模块 | 账号：test  密码：test | 提示登录成功 | 登录成功，跳转到首页或redirect页面 | 通过 |
| 忘记密码第一步 | 邮箱：123  账号：test | 点击发送验证码按钮，提示邮箱格式不正确 | 点击发送验证码按钮，页面提示邮箱格式不正确 | 通过 |
| 忘记密码第一步 | 邮箱：[123@qq.com](mailto:123@qq.com)  账号：text | 点击发送验证码按钮提示邮箱账号不正确 | 点击发送验证码按钮提示邮箱账号不正确 | 通过 |
| 忘记密码第一步 | 邮箱：[96967913@qq.com](mailto:96967913@qq.com)  账号：test | 点击发送验证码按钮提示验证码发送成功 | 点击发送验证码按钮提示验证码发送成功 | 通过 |
| 忘记密码第一步 | 邮箱：[96967913@qq.com](mailto:96967913@qq.com)  账号：test  验证码：正确的邮箱验证码 | 点击下一步，显示重置密码表单 | 点击下一步，显示重置密码表单 | 通过 |
| 忘记密码第二步 | 新密码：123  确认密码：321 | 点击完成，提示两次输入的密码不一致，请重新输入 | 点击完成，提示两次输入的密码不一致，请重新输入 | 通过 |
| 忘记密码第二步 | 新密码：123  确认密码：123 | 点击完成，提示修改成功，引导用户去登录 | 修改成功，跳转登录页面 | 通过 |

6.3 其他错误

大多数的错误都是出现在前后端接口联调时接口传参出现问题，导致请求失败，也有很多错误是因为对SQL语句运用不当导致的，不过都比较容易的解决掉了。

结束语

本设计让我把大学四年所学的全部知识得到了很好的运用，也算是给了自己一个比较满意的交代。这个程序设计让我初步的体验了全栈开发的快感，无论是使用到的技术点还是开发方式都是当下最热门的，这也为毕业之后的工作打下了基础。

在校期间有学习过vue这门技术，但是那时候还是按部就班的跟着网上老师的思路去实现每一个功能，这种学习方式的弊端就是遇到问题时让我不由自主的按着老师教的方式去解决，缺少了自己的思考与理解。这个毕业设计虽然花费了不少心思，中间也出现过很多难以解决的问题，但是最后都是在自己的思考，网上一些文档博客中找到了最佳的解决办法，正是出现的这些让人头疼的问题才让我对于vue这门技术有了更深的理解，使用起来也更得心应手了。相比于原生的开发语言，vue真的是为前端开发带来了很大的便利，是值得学习一门语言。

作为一个前端程序员，一直认为后端很难，却又总想着触及一下后端领域的工作。最后发现node这门技术，他最适合我的地方就是可以用JavaScript语言也能搭建出一个健全的服务器。而且很多功能都是可以通过中间件来完成的，又给我开发服务器业务节省了很多时间。让我知道了怎么合理的操作数据库，怎样封装数据接口才能让前端人员使用起来更加方便。不过此程序涉及到的后端相对来说还是比较容易的，项目也有很多地方需要改进完善。

通过这个程序让我更加坚定的踏上这条开发的道路上。在把自己所学的东西合理的运用的同时，也让自己认识到了自己的不足之处，对于项目中的很多地方还需要进一步完善，然后对于每个问题都要找到最好的解决方式。最重要的是，在这个技术不断更新的时代，我们应该时时刻刻保持最饱满的状态努力学习，这样才能让自己变得更好！

参考文献

[1] Elizabeth Castro,Bruce Hyslop.HTML5与CSS3基础教程(第8版),人民邮电出版社,2014年5月.

[2] Nicholas C.Zakas.JavaScript高级程序设计(第3版),人民邮电出版社,2006年1月.

[3] 张海藩,吕云翔.软件工程(第4版),人民邮电出版社,2013年9月.

[4] 刘博文.深入浅出Vue.js,人民邮电出版社,2019年3月.

[5] 张耀春,黄秩,王静,苏伟,王瑾,殷献勇.Vue.js权威指南,电子工业出版社,2016年9月.

[6] 陆凌牛.狼书:更了不起的Node.js,电子工业出版社,2019年7月.

[7] 李银城.高效前端:web高效编程与优化实践,机械工业出版社,2018年3月.

[8]《ES6标准入门》 阮一峰 电子工业出版社 2017

[9]《Webpack实战：入门、进阶和调优》 居玉皓 机械工业出版社 2019

[10]《高性能MySQL》 [美]Baron Schwartz 电子工业出版社 2014

[11]《前端工程化体系设计与实践》 周俊鹏 电子工业出版社 2013

致 谢

四年时光，转瞬即逝。不知不觉就到了自己的毕业季，回想过去的点点滴滴，仿佛一切又在眼前。大四9月份实习到现在已经有大半年的时间了，没想到的是因为疫情再也不能回去那个写满青春的地方了，去年的9月份也是我最后一次以学生的身份待在学校了。在这里我要感谢教过我的每一个老师，无论课堂上的孜孜不倦，实验室的耐心指导，实训时的无私奉献，还是生活中的言传身教，都让我难以忘记，让我从一个一无所知的新生变成了能够在社会中生存下去的勇敢少年。这四年里让我学到了很多知识，所以非常感觉老师们的努力和付出，我也会继续努力下去不辜负你们的希望。还要感谢一下我的大学同学，尤其是我们寝室的小伙伴，感谢这四年的陪伴与鼓励，多亏有了你们，才会让这个大学更加多姿多彩，遗憾的是没有机会再回学校拍毕业照，好好地疯一回。我更是要感谢一下曾经的自己，感谢自己再过去的时间里没有荒废，能够充分利用课余时间去学习技术和知识，能够为未来的自己打好基础。

最后，很荣幸能够与大家相识，也感谢大家的照顾，祝我们每个人前程似锦，未来可期！