项目实战（二）

chileme 在线点餐系统

内容导读

目前，随着餐饮业的高速发展和餐饮店规模的不断扩大，手工点菜方式的工作效率，点菜正确率已经难以适应企业发展的需求，制约了餐饮业的发展，餐饮企业的特色和个性化经营更加明显，管理更趋近于信息化，使得传统餐饮正逐步向现代餐饮方向转化，而现代餐饮最显著的特征就是使用计算机信息化管理系统，这也是现代餐饮企业经营者们的共识。计算机信息化管理系统以信息量大、数据准确、速度快、管理效能高的特点已获得了广大餐饮、娱乐企业管理者的认可。

吃了么在线点餐系统，是一种全新的集登录、注册、点餐、商品分类、在线下单以及订单查询等功能于一体的信息化点餐系统。适用于餐饮，酒店，咖啡厅等场所的餐台管理、点菜录单、结算、信息反馈与传递。结合传统的点菜管理系统，为餐饮、酒店、咖啡厅等行业的经营管理提供了一整套稳定可靠、先进的解决方案，改变了餐饮等行业的手工经营方式，提高了服务效率和顾客满意程度，提升了店面形象，最终提升了企业竞争力与经营效益。

课程目标

- 巩固已经学到专业知识
- 扩展学生思维疆域
- 启迪思维，灵活运用技术

实战目标

- 实现注册及登录功能开发
- 实现商品分类及商品查询功能开发
- 实现下单及订单查询功能开发

1.1 系统分析

1.1.1 需求分析

随着科技的不断进步，互联网的不断发展，传统餐饮的点餐模式，现在已经不能满足人们的需求。传统的点餐方式存在难计算、难查找、难更改、易出错、效率低等缺点，而顾客的时间成本和餐饮企业的管理成本都很高，通过点餐系统的建立，可以有效节约成本，而且通过对点餐数据的分析，可以更好地进行菜品管理，既节约采购成本，又可以更好地满足客户需求。本项目正是希望用计算机来解决以上问题。

项目需求：

1. 用户管理功能，通过用户注册和用户登录功能对用户的信息进行统一的管理。
2. 商品分类功能，向用户展示门店的所有商品，并且进行分类。
3. 常用商品功能，向用户展示门店的特色或热销商品。
4. 在线下单功能，用户可以通过电脑或者手机进行在线下单。
5. 订单管理功能，用户可以查看已经选定的订单。

1.1.2 开发工具选择

本项目编辑器使用 Hbuilder，兼容 Angular/jQuery/Bootstrap 等各种前端框架，能封装 H5APP，非常适合新手使用。

后台使用 Node，Node 是一个基于 Chrome V8 引擎的 JavaScript 运行平台，使用高效、轻量级的事件驱动、非阻塞 I/O 模型。它的包生态系统——NPM，是目前世界上最大的开源库生态系统。

数据库使用 MongoDB， MongoDB 是一个面向文档存储的数据库，操作起来比较简单和容易，也是一个基于分布式文件存储的开源数据库系统。

前端采用 Vue.js 组件化开发单页应用，Vue.js 是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。页面布局采用 Element UI，Element UI 是一套采用 Vue 2.0 为基础框架实现的组件库，它面向企业级的后台应用，能够帮助我们快速地搭建网站，极大地减少研发的人力与时间成本。数据交互方面采用 axios 插件，axios 是通过 promise 实现对 AJAX 技术的一种封装。

1.2 总体设计

1.2.1 项目规划

本项目是一个在线的订餐系统全栈，使用 Vue.js+Node+MongoDB，可以实现用户注册到下单的完整业务流程。业务流程图见图 1-1。

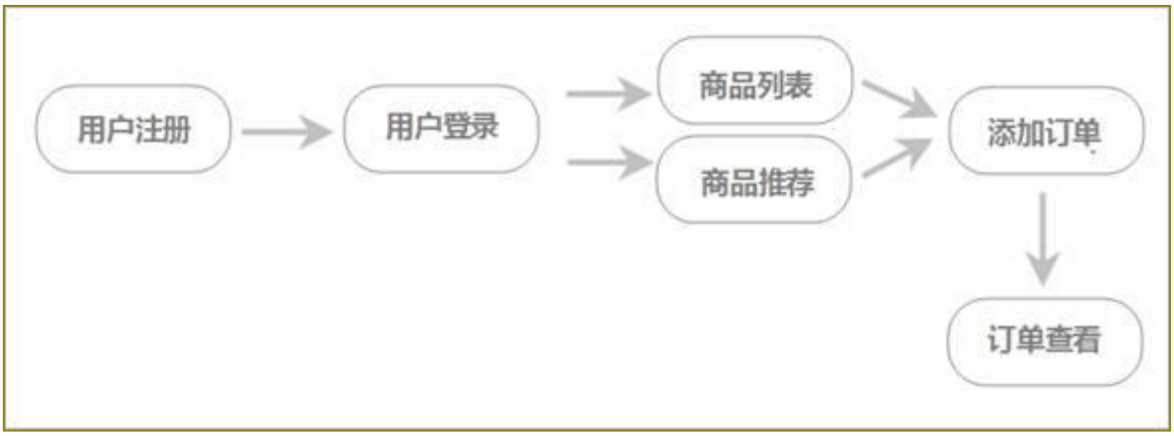

图 1-1 业务流程图

1.2.2 项目结构

本软件项目包含一个解决方案 chileme，其中包含两个子项目：

1. chilemeSev

项目的服务端程序，负责接受客户端的请求，并且依据请求的数据操作数据库及返回响应。

2. chilemeCli

项目的客户端程序，负责展现页面和交互，并且将客户的请求发送到服务端。

1.2.3 项目功能分析

1. 用户注册

客户端完成初步的表单验证采集用户输入的用户名、密码、手机号，并将数据发送到服务端，服务端进行数据的进一步验证，注册成功将用户信息保存在数据库中。

2. 用户登录

客户端完成初步的表单验证采集用户输入的用户名、密码，并将数据发送到服务端，服务端进行数据的进一步验证，登录成功返回授权信息作为响应。

3. 商品分类列表

客户端将商品类别及查询数量数据发送到服务端，服务端根据请求数据进行数据库查询操作，并且将查询结果作为响应返回。

4. 常用商品列表

客户端将常用商品参数及查询数量数据发送到服务端，服务端根据请求数据进行数据库查询操作，并且将查询结果作为响应返回。

5. 订单添加

客户端将商品名称，商品价格，商品图片，商品数量等数据发送到服务端，服务端根据请求数据进行数据库插入操作，并且将插入结果作为响应返回。

6. 订单查看

客户端将用户名和授权信息发送到服务端，服务端根据请求数据进行数据库查询操作，并且将查询结果作为响应返回。

1.3 系统设计

1.3.1 设计目标

本系统属于在线点餐系统雏形，目前主要实现了最核心的功能，即在线点餐。完整体业务还包括库存数据的导入导出，销售数据的统计分析以及会员系统的建立，在后续课程中，可以对其中使用技术进行扩展，完善这个项目的功能，具体可以实现以下 4 点：

1. 开发会员管理页面，实现会员商品的折扣。
2. 开发库存商品页面，实现库存商品的添加和删除。
3. 开发移动端适配功能，实现电脑端和移动设备的兼容。
4. 开发支付功能，实现系统真实在线收款。

1.3.2 开发及运行环境

网站开发平台：Node
网站开发语言：JavaScript
网站后台数据库：Mongodb
网站前端开发框架：Vue.js 2.5.9
网站页面布局框架：Element UI
网站数据交互：axios
运行平台：Microsoft Windows 7/8/10/2000/2003/2008

1.3.3 数据库设计

本项目数据库使用的是 MongoDB，所有的集合在 MongoDB 中 chileme 数据库中，，其中用户集合用来记录用户信息，包括用户名、用户密码、注册手机号。用户集合结构见表 1-1 表所示。

表 1-1 用户集合（users）结构

集合名	键名	数据类型	描述
users	username	string	用户名
	password	string	用户密码
	usertel	number	注册手机号

用户授权集合用来记录用户授权信息，包括用户名、授权码、授权时间、过期时间，当涉及到用户的敏感操作，包括下单、订单查询、订单删除等，需要核对用户是否授权。用户授权集合结构见表 1-2 所示。

表 1-2 授权集合（tokens）结构

集合名	键名	数据类型	描述
tokens	username	string	用户名
	tokenid	string	授权码
	gettime	date	授权时间
	overtime	date	过期时间

商品集合用来记录用商品，包括商品名称、商品类型、商品图片、商品价格等，客户端的商品列表页面需要

读取该集合中的内容用来生成商品列表。商品集合结构见表 1-3 所示。

表 1-3 商品集合（goods）结构

集合名	键名	数据类型	描述
goods	goodsName	string	商品名称
	goodType	number	商品类型
	goodsImg	string	商品图片
	goodOfften	number	商品推荐
	price	number	商品价格
	count	number	商品数量

订单集合用来记录订单详情、下单用户、下单时间、订单状态等，商品集合结构见表 1-4 所示。

表 1-4 订单集合（orderlists）结构

集合名	键名	数据类型	描述
orderlists	username	string	下单用户
	orderlist	string	订单详情
	creattime	date	下单时间
	orderstate	string	订单状态

1.3.4 客户端页面设计

1. 注册页在浏览器中的预览结果如图 1-2。

图 1-2 注册界面

2. 登录页在浏览器中的预览结果见图 1-3。

图 1-3 登录界面

3. 点餐页在浏览器中的预览结果如图 1-4。

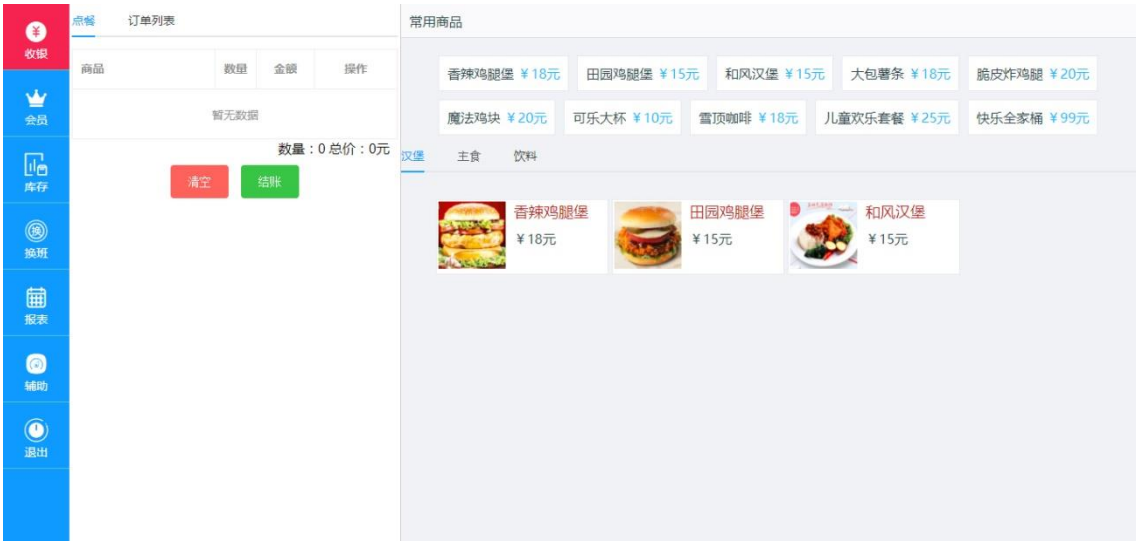

图 1-4 点餐界面

1.4 项目准备工作

1.4.1 创建项目客户端

1. 通过 Vue-cli 搭建开发环境。
2. 通过 NPM 安装 axios 以及 Element UI 插件并配置到项目中。
3. 在阿里巴巴矢量图标库（http://www.iconfont.cn/）中，下载项目所需图标。

1.4.2 创建项目服务端

1. 创建项目的服务端，命名为 chilemeSev。
2. 在 chilemeSev 中创建 img 文件夹，该文件夹为项目中用到的图片文件夹。
3. 在 chilemeSev 中创建 model 文件夹，该文件夹为项目中用到的 MongoDB 模型文件夹。
4. 在 chilemeSev 中创建 mong 文件夹，该文件夹为项目中用到的数据库连接文件夹。
5. 在 chilemeSev 中创建 route 文件夹，该文件夹为项目中用到的路由文件夹。
6. 在 chilemeSev 中创建 app.js 文件，该文件为项目的入口文件。

项目服务端结构见图 1-5。

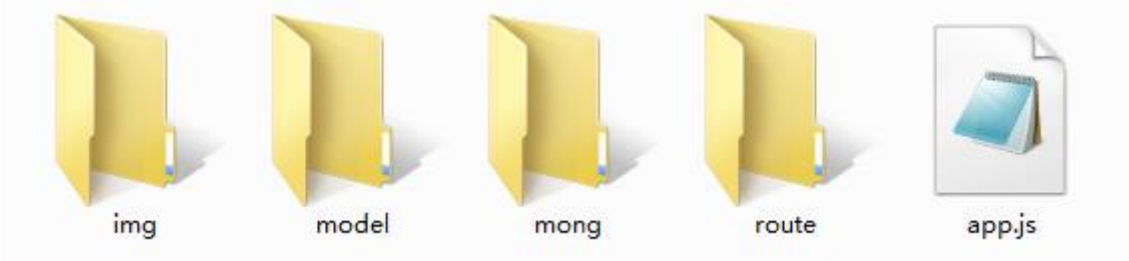

图 1-5 项目服务端结构

1.5 注册功能开发

1.5.1 需求分析

本节将完成项目的注册功能，包含注册功能服务端开发和注册功能客户端开发。

1.5.2 注册功能服务端开发思路

注册功能服务端开发需要用到的模块有 mongoose、koa、koa2-cors、koa-body、koa-route 等，程序运行逻辑如下：

1. 客户端将用户名、密码、手机号以 POST 请求方式发送到服务端；
2. 服务端获取请求，将用户名和手机号分别放到数据库用户集合中进行查询操作；
3. 如果查询到对应文档，则用户名或手机号已经被注册，返回响应；
4. 如果查询不到对应文档，则允许注册，将注册信息插入到数据库，并返回响应；
5. 客户端根据响应在页面输出提示或进行页面跳转。

1.5.3 注册功能服务端核心代码

1. 使用 NPM 安装所需的模块。
2. 在服务端程序文件夹 mong 中创建 mongo.js 文件，该文件是数据库的连接文件，代码如下。

```
var mongoose = require('mongoose'),      //加载 mongoose 模块
DB_URL = 'mongodb://localhost:27017/chileme';
mongoose.Promise = global.Promise;
//连接数据库
mongoose.connect(DB_URL,{useMongoClient: true});
//连接成功
mongoose.connection.on('connected', function () {
    console.log('Mongoose connection open to ' + DB_URL);
});
// 连接异常
mongoose.connection.on('error',function (err) {
    console.log('Mongoose connection error: ' + err);
});
// 连接断开
mongoose.connection.on('disconnected', function () {
    console.log('Mongoose connection disconnected');
});
module.exports = mongoose;    //导出 mongoose 对象
```

3. 在服务端程序文件夹 model 中创建 user.js 文件，该文件是用户集合的模型文件，代码如下。

```
let mongoose = require('../mong/mongo.js'),      //加载 mongoose 对象
    Schema = mongoose.Schema;
let UserSchema = new Schema({
    username : { type: String },
```

```
    password: {type: String},
    usertel: {type: String}
});
module.exports = mongoose.model('User',UserSchema)
```

4. 在服务端程序文件夹 route 中创建 reg.js 文件，该文件是注册功能的路由文件，代码如下。

```
const User = require("../model/user.js");
const reg = async function(ctx) {
  let username = ctx.request.body.username;
    let password = ctx.request.body.password;
    let usertel = ctx.request.body.usertel;
    let result = {
        errCode: 0,
    };
  let user = await User.findOne({
            username
        }).exec()
    let tel = await User.findOne({
            usertel
        }).exec()
  if(!tel){
       if (!user) {
            user = new User({
                username,
                password,
                usertel
            })
        } else {
            //用户已经存在
            result.errCode = 1
            result.errMsg = '用户名重复了！请换一个！'
            ctx.body = result
            return
        }
        try {
            //写入数据库
            user = await user.save();
            result.errCode = 0
```

```
                result.errMsg = '注册成功'
                ctx.body = result
            } catch (e) {
                console.log('md,err==', e)
            }
        }else{
            result.errCode = 3
            result.errMsg = '手机号已经被注册！'
            ctx.body = result
            return
        }
};
module.exports = reg;
```

5. 在服务端程序入口文件 app.js 中加载 koa、koa2-cors、koa-body、koa-route 模块，进行跨域配置，并且载入注册功能的路由，代码如下。

```
const Koa = require('koa');
const koaBody = require('koa-body');
const route = require('koa-route');
const cors = require('koa2-cors');
const app = new Koa();
//解决跨域
app.use(cors({
    origin: "*",
    exposeHeaders: ['WWW-Authenticate', 'Server-Authorization', 'Date'],
    maxAge: 100,
    credentials: true,
    allowMethods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
    allowHeaders: ['Content-Type', 'Authorization', 'Accept', 'X-Custom-Header', 'anonymous'],
}));
app.use(koaBody());
//载入路由
app.use(route.post('/reg', require("./route/reg.js")));
//设置端口
app.listen(3000);
```

注册功能服务端接口开发完成，接口信息见表 1-5 所示。

表 1-5 注册功能接口

接口名	接口地址	请求方式	请求参数	参数描述

注册	服务器地址+/reg	post	username	用户名
			password	用户密码
			usertel	注册手机号

1.5.4 注册功能客户端开发思路

注册功能客户端开发需要用到的插件有 axios、Element UI 以及带路由功能的 webpack 模板。页面实现思路如下：

1. 根据整个项目逻辑，完成页面路由功能设置及相关插件部署。
2. 新建注册组件，在该组件中通过 Element UI 插件，布局注册页相关表单选项并完成插件提供的表单验证功能。
3. 在注册组件中，为“注册”按钮绑定点击事件。在该事件函数中，使用 axios 向服务端发起 POST 请求并根据服务端需求发送用户名、密码以及电话号码。
4. 如果发送请求成功，则需要判断服务端状态码。根据状态码给出相应提示，如果状态码为 0 则表示注册成功，页面跳转至登录页面。如果请求失败，则表示网络错误，需要给出相应提示。

1.5.5 注册功能客户端核心代码

1. 使用 NPM 安装所需插件。
2. src 目录下的 main.js 文件配置如下。

```
import Vue from 'vue'
import ElementUI from 'element-ui';                    // 导入 element-ui 插件
import 'element-ui/lib/theme-chalk/index.css';         // 导入 element-ui 样式表文件
import App from './App.vue';
import router from './router';
import axios from 'axios'                              // 导入 axios 插件
Vue.prototype.$http = axios                            // 挂载 axios 到原型链上，方便全局调用
// 默认服务端请求地址，可以根据具体搭建环境进行修改
axios.defaults.baseURL = 'http://10.10.4.205:3000'
// 全局定义站点图片地址，可以根据具体搭建环境进行修改
Vue.prototype.imgUrl = "http://10.10.4.205:3000/"

Vue.config.productionTip = false
Vue.use(ElementUI);

new Vue({
  el: '#app',
  router,
  components: { App },
```

```
  template: '<App/>'
})
```

3. router 目录下路由 index.js 配置如下。

```
import Vue from 'vue'
import Router from 'vue-router'
import Reg from '@/components/page/Reg'       // 引入 Reg 组件
Vue.use(Router)
export default new Router({
  routes: [
    {                                          // 设置注册组件路由地址
      path: '/reg',
      name: 'Reg',
      component: Reg
    }
  ]
})
```

4. 在 components 目录下新建 page 目录，该目录下新建注册组件 Reg.vue，Reg.vue 组件中 template 标签中的代码如下。

```
<template>
  <div class="login">
    <div class="hd">
      <i class="icon iconfont icon-chushimao"></i>
    </div>
    <div class="main">
      <el-form :model="ruleForm2" status-icon :rules="rules2" ref="ruleForm2" label-width="100px" class="demo-ruleForm">
        <el-form-item label="用户名" prop="username">
          <el-input type="text" v-model="ruleForm2.username" auto-complete="off"></el-input>
        </el-form-item>
        <el-form-item label="密码" prop="pass">
          <el-input type="password" v-model="ruleForm2.pass" auto-complete="off"></el-input>
        </el-form-item>
        <el-form-item label="确认密码" prop="checkPass">
          <el-input type="password2" v-model="ruleForm2.checkPass" auto-complete="off"></el-input>
        </el-form-item>
```

```
                <el-form-item label="手机号码" prop="tel">
                    <el-input type="password2" v-model="ruleForm2.tel" auto-complete="off"></el-input>
                </el-form-item>
                <el-form-item align="center" id="btn">
                    <el-button type="primary" @click="submitForm()">注 册</el-button>
                    <el-button @click="resetForm('ruleForm2')">重置</el-button>
                </el-form-item>
            </el-form>
            <div class="bottom-bar">
                <p>吃了么点餐系统隶属于格莱科技旗下产品，最终解释权属于本公司。<br>我有账号<a href="javascript:void(0)" @click="jumpLogin">点击登录</a></p>
            </div>
        </div>
    </div>
</template>
```

5. Reg.vue 组件中表单验证的 data 选项中的数据代码如下。

```
data() {
    var validateUser = (rule, value, callback) => {    // 验证用户名为空
      if (value === '') {
        callback(new Error('请输入用户名'));
      }
    };
    var validatePass = (rule, value, callback) => {    // 验证密码为空
      if (value === '') {
        callback(new Error('请输入密码'));
      }else {
        if (this.ruleForm2.checkPass !== '') {
          this.$refs.ruleForm2.validateField('checkPass');
        }
        callback();
      }
    };
    var validatePass2 = (rule, value, callback) => {   // 验证密码不一致
      if (value === '') {
        callback(new Error('请再次输入密码'));
```

```
    } else if (value !== this.ruleForm2.pass) {
      callback(new Error('两次输入密码不一致!'));
    } else {
      callback();
    }
  };
   var checkTel = (rule, value, callback) => {  // 验证手机号，必须符合规定
     let pattern = /^1[34578]\d{9}$/;
    if (!value) {
      return callback(new Error('手机号码不能为空'));
    }
    if (!pattern.test(value)) {
      return callback(new Error('手机号码格式不正确'));
    }
  };
  return {
    ruleForm2: {
      username: '',
      pass: '',
      checkPass:'',
      tel:''
    },
    rules2: {                              // Element UI 默认表单提示效果
      username: [
        { validator: validateUser, trigger: 'blur' }
      ],
      pass: [
        { validator: validatePass, trigger: 'blur' }
      ],
      checkPass: [
        { validator: validatePass2, trigger: 'blur' }
      ],
      tel: [
        { validator: checkTel, trigger: 'blur' }
      ]
    }
  };
```

```
},
```

6. Reg.vue 组件中 methods 选项中函数代码如下。

```
methods: {
    submitForm() {                          //  “注册”按钮触发事件
      if (this.ruleForm2.username=="" || this.ruleForm2.pass=="") {
          this.$message('信息不完整');
      } else {
          axios.post('/reg',{               // 验证成功后向服务端发送请求
            username:this.ruleForm2.username,
            password:this.ruleForm2.pass,
            usertel:this.ruleForm2.tel
          })
          .then(response=>{                 // 处理发送请求成功的状态码
            this.$message(response.data.errMsg)
            if (response.data.errCode!=0) {
                this.$message.error(response.data.errMsg)
            } else {
              this.$message(response.data.errMsg)
              this.$router.push('/')    // 验证成功后，路由跳转至登录界面
            }
          })
          .catch(error=>{                   // 处理发送请求失败，给出提示信息
              this.$message.error("网络问题，请稍候连接。")
              console.log(error.data)
          })
      }
    },
    resetForm(formName) {               // 重置表单
      this.$refs[formName].resetFields();
    },
    jumpLogin(){                            // 跳转至登录页链接设置
        this.$router.push('/')
    }
  }
}
```

1.6 登录功能开发

1.6.1 需求分析

本节将完成项目的登录功能，包含登录功能服务端开发和登录功能客户端开发。

1.6.2 登录功能服务端开发思路

登录功能服务端开发需要用到的模块有 koa、koa2-cors、koa-body、koa-route、string-random 等，程序运行逻辑如下：

1. 客户端将用户名、密码以 POST 请求方式发送到服务端；
2. 服务端获取请求，将用户名和密码放到数据库用户集合中进行查询操作；
3. 如果查询到对应文档，则用户密码输入正确，生成授权码，将授权信息存在数据库授权集合中，并且返回响应。
4. 如果查询不到对应文档，则用户密码输出错误，返回响应。
5. 客户端根据响应在页面输出提示或进行页面跳转。

1.6.3 登录功能服务端核心代码

1. 使用 NPM 安装所需的模块。
2. 在服务端程序文件夹 route 中创建 login.js 文件，该文件是登录功能的路由文件，代码如下。

```
const User = require("../model/user.js");
const Token = require("../model/token.js");
const stringRandom = require('string-random');
    async function login(ctx){
        let username = ctx.request.body.username;
        let password = ctx.request.body.password;
        let result = {
            errCode: 0
        };
        let user = await User.findOne({
                username: username,
                password: password
            }).exec()
        if (!user) {
            result.errCode = 1
            result.errMsg = '账户或者密码错误！'
            ctx.body = result
            return
        }else {
            //用户已经存在
            let token=stringRandom(32);
            let time=Date.now();
            let overtime=Date.now()+1000*60*60*24;
```

```
        let seachtoken = await Token.findOne({
            username: username
        }).exec()
        if(!seachtoken){
          addtoken = new Token({
                  username: username,
                    tokenid:    token,
                    gettime: time,
                    overtime: overtime
              })
              try {
                    addtoken = await addtoken.save();//写入数据库
              } catch (e) {
                   result.errMsg = 'token 写入失败'
                    return next
              }
        }else{
          await Token.update(
                {username: username},
                {tokenid: token,gettime: time,overtime: overtime}
          ).exec()
        }
        result.errCode = 0
        result.errMsg = '登录成功！'
        result.username=username
        result.tokenID =token
         ctx.body = result
      }
    }
module.exports = login;
```

3. 在服务端程序入口文件 app.js 中载入登录功能的路由，添加的代码如下。

```
app.use(route.post('/login', require("./route/login.js")));
```

登录功能服务端接口开发完成，接口信息见表 1-6 所示。

表 1-6 登录功能接口

接口名	接口地址	请求方式	请求参数	参数描述
登录	服务器地址+/login	post	username	用户名

			password	用户密码

1.6.4 登录功能客户端开发思路

登录功能客户端开发需要用到的插件有 axios、Element UI 插件。页面实现思路如下：

1. 新建登录组件，在该组件中通过 Element UI 插件，布局注册页相关表单选项并完成插件提供的表单验证功能。
2. 在路由配置文件中添加登录组件跳转链接。
3. 在登录组件中，为“提交”按钮绑定点击事件。在该事件函数中，使用 axios 向服务端发起 POST 请求并根据服务端需求发送用户名和密码。
4. 如果发送请求成功，则需要判断服务端状态码。根据状态码给出相应提示，如果状态码为 0 则表示登录成功，需保存服务端返回的 tokenID 以及输入的用户名，然后跳转至点餐界面。如果请求失败，则表示网络错误，需要给出相应提示。

1.6.5 登录功能客户端核心代码

1. router 目录下为路由 index.js 文件添加登录组件跳转地址，代码如下。

```
routes: [
    {
      path: '/',
      name: 'Login',
      component: Login
    },
  ]
```

2. 在 page 目录下新建登录组件 Login.vue，Login.vue 组件 template 标签内的代码如下。

```
<template>
  <div class="login">
      <div class="hd">
        <i class="icon iconfont icon-chushimao"></i>
        </div>
      <div class="main">
        <el-form :model="ruleForm2" status-icon :rules="rules2" ref="ruleForm2" label-width="100px" class="demo-ruleForm">
            <el-form-item label="用户名" prop="username">
              <el-input type="text" v-model="ruleForm2.username" auto-complete="off"></el-input>
            </el-form-item>
            <el-form-item label="密码" prop="pass">
              <el-input type="password" v-model="ruleForm2.pass" auto-complete="off"></el-input>
            </el-form-item>
```

```
                <el-form-item align="center" id="btn">
                    <el-button type="primary" @click="submitForm()">提交</el-button>
                    <el-button @click="resetForm('ruleForm2')">重置</el-button>
                </el-form-item>
            </el-form>
            <div class="bottom-bar">
                <p>吃了么点餐系统隶属于格莱科技旗下产品，最终解释权属于本公司。<br>还没账号请<a href="javascript:void(0)" @click="jumpReg">点击注册</a></p>
            </div>
        </div>
    </div>
</template>
```

3. Login.vue 组件 script 标签内 data 选项中的代码如下。

```
data() {
    var validateUser = (rule, value, callback) => {          // 验证用户名
      if (value === '') {
        callback(new Error('请输入用户名'));
      }
    };
    var validatePass = (rule, value, callback) => {          // 验证密码
      if (value === '') {
        callback(new Error('请输入密码'));
      }
    };
    return {
      ruleForm2: {
        username: '',
        pass: ''
      },
      rules2: {
        username: [
          { validator: validateUser, trigger: 'blur' }
        ],
        pass: [
          { validator: validatePass, trigger: 'blur' }
        ]
      }
```

```
    };
  },
```

4. Login.vue 组件 script 标签内 methods 选项中的代码如下。

```
methods: {
    submitForm() {                                  // 提交功能
      if (this.ruleForm2.username=="" || this.ruleForm2.pass=="") {
          this.$message.error('信息不完整!');
      } else {
          axios.post('/login',{                     // 发送服务端请求
            username:this.ruleForm2.username,
            password:this.ruleForm2.pass
          }).then((response)=>{
            if (response.data.errCode!=0) {
                this.$message.error(response.data.errMsg)
            } else {
              this.$message(response.data.errMsg)
              localStorage.setItem('tokenID',response.data.tokenID) // 保存 tokenID
              localStorage.setItem('userName',this.ruleForm2.username)
              this.$router.push('/pos')        // 提交成功后，跳转至点餐界面
            }
          })
          .catch((error)=>{                         // 提交失败，提示网络问题
            this.$message.error("网络问题，请稍候连接。")
            console.log(error)
          })
      }
    },
    resetForm(formName) {
      this.$refs[formName].resetFields();
    },
    jumpReg(){
        this.$router.push('reg')
    }
  }
```

1.7 商品分类功能开发

1.7.1 需求分析

本节讲完成项目的商品分类功能，包含商品分类功能服务端开发和商品分类功能客户端开发。

1.7.2 商品分类功能服务端开发思路

商品分类功能服务端开发需要用到的模块有 koa、koa2-cors、koa-body、koa-route、koa-static、path 等，程序运行逻辑如下：

1. 客户端向服务端发起请求；
2. 服务端获取请求，在数据库商品集合中进行查询，将查询结果作为响应按商品分类返回给客户端；
3. 客户端获取响应，将响应数据渲染在页面中。

1.7.3 商品分类功能服务端核心代码

1. 在服务端程序文件夹 model 中创建 good.js 文件，该文件是商品集合的模型文件，代码如下。

```
let mongoose = require('../mong/mongo.js'),
    Schema = mongoose.Schema;
let GoodSchema = new Schema({
    goodname : { type: String },
    goodtype: {type: String},
    goodprice: {type: Number},
    goodimg: {type: String},
    goodofften: {type: Number}
});
module.exports = mongoose.model('Good',GoodSchema)
```

2. 在服务端程序文件夹 route 中创建 goodslist.js 文件，该文件是商品分类功能的路由文件，代码如下。

```
const Good = require("../model/good.js");
//商品分类查询功能
const goodslist = async function(ctx) {
    let result = {
        errCode: 0
    };
    let goods1 = await Good.find({"goodType": 1});
    let goods2 = await Good.find({"goodType": 2});
    let goods3 = await Good.find({"goodType": 3});
    if(goods1.length==0&&goods2.length==0&&goods3.length==0){
        result.errCode = 1
        result.errMsg = '没有找到查询结果'
        ctx.body = result
        return
    }else{
```

```
            result.errCode = 0
             result.errMsg = '查询到结果'
             result.list={type1:goods1,type2:goods2,type3:goods3}
             ctx.body = result
             return
        }
    };
    module.exports = goodslist;
```

3. 将项目用到的商品图片文件拷贝到项目的 img 文件夹中。
4. 在数据库的 goods 集合中添加多个商品信息文档，核心代码如下。

```
{
    "goodsImg" : "img/pos001.jpg",
    "goodsName" : "香辣鸡腿堡",
    "price" : 18,
    "goodType" : 1,
    "goodOfften" : 1,
    "count" : 1
}
```

5. 在服务端程序入口文件 app.js 中载入 koa-static、path 模块及商品分类功能的路由，添加的代码如下。

```
const koastatic = require('koa-static');
const path = require('path');
const koastatics = koastatic(path.join(__dirname));
app.use(koastatics);
app.use(route.get('/goodslist', require("./route/goodslist.js")));
```

商品分类功能服务端接口开发完成，接口信息见表 1-7 所示。

表 1-7 商品分类功能接口

接口名	接口地址	请求方式	请求参数	参数描述
商品分类	服务器地址+/ goodslist	get	无	无

1.7.4 商品分类功能客户端开发思路

商品分类功能客户端功能，实现思路如下：

1. 新建点餐组件，在路由配置文件中添加点餐组件路由跳转地址。
2. 新建点餐组件左侧公共导航组件，使用阿里巴巴矢量图标库中的字体完成左侧栏目布局。
3. 完成左侧栏目中“退出”功能的制作。其原理是，点击“退出”按钮清除本地存储的 tokenID。
4. 在点餐组件中使用 Element UI 完成中间区域“点餐”列表布局。
5. 在点餐组件中使用 Element UI 完成右侧商品分类列表布局。通过 axios 向服务端发送 get 请求，获

取商品分类数据并渲染到页面结构中。

1.7.5 商品分类功能客户端核心代码

1. 在 page 目录下新建点餐组件 Pos.vue，切换到 router 目录下为路由 index.js 文件添加 Pos.vue 组件跳转地址，代码如下。

```
import Pos from '@/components/page/Pos'        // 引入 Pos.vue 组件
routes: [
    {
      path: '/pos',
      name: 'Pos',
      component: Pos
    },
  ]
```

2. 在 components 目录下新建 common 目录，该目录下新建公共导航组件 LeftNav.vue，LeftNav.vue 组件中代码如下。

```
    <template>
        <div class="left-bar">
            <ul>
              <li><a href="#"><i class="icon iconfont icon-rechargefill"></i> 收 银 </a></li>
              <li><a href="#"><i class="icon iconfont icon-huiyuan"></i>会员</a></li>
              <li><a href="#"><i class="icon iconfont icon-navicon-kczb"></i> 库 存 </a></li>
              <li><a href="#"><i class="icon iconfont icon-jiaohuan"></i>换班</a></li>
              <li><a href="#"><i class="icon iconfont icon-biao"></i>报表</a></li>
              <li><a href="#"><i class="icon iconfont icon-ceshifuzhu"></i> 辅 助 </a></li>
              <li><a href="javascript:void(0)" @click="loginOut"><i class="icon iconfont icon-tuichu-"></i>退出</a></li>
            </ul>
        </div>
    </template>
    <script>
    export default {
      name: 'LeftNav',
      methods: {
          loginOut(){                                    // 点击退出功能
            localStorage.removeItem('tokenID')
            this.$message('退出成功!');
```

```
                this.$router.push('/')
            }
        }
    }
</script>
```

3. 打开 Pos.vue 组件，template 标签中代码如下。

```
<template>
    <div class="pos">
        <left-nav></left-nav>
        <div class="pos-main">
            <el-row>
                <!-- 待支付订单结构代码，后面再添加 -->
                <el-col :span="17" id="goods-list">
                    <!-- 常用商品结构代码，后面再添加 -->
                    <div class="goods-type">
                        <el-tabs>
                            <el-tab-pane label="汉堡">
                                <ul class='cookList'>
                                <li v-for="goods in type1Goods"  @click="addOrderList(goods)">
                                    <span
class="foodImg"><img :src="imgUrl+goods.goodsImg" width="100%"></span>
                                    <span
class="foodName">{{goods.goodsName}}</span>
                                    <span  class="foodPrice">  ￥ {{goods.price}} 元
</span>
                                </li>
                                </ul>
                            </el-tab-pane>
                            <el-tab-pane label="主食">
                                <ul class='cookList'>
                                <li v-for="goods in type2Goods"  @click="addOrderList(goods)">
                                    <span
class="foodImg"><img :src="imgUrl+goods.goodsImg" width="100%"></span>
                                    <span
class="foodName">{{goods.goodsName}}</span>
                                <span class="foodPrice">￥{{goods.price}}元</span>
                                </li>
                                </ul>
```

```
                    </el-tab-pane>
                    <el-tab-pane label="饮料">
                        <ul class='cookList'>
                    <li v-for="goods in type3Goods"   @click="addOrderList(goods)">
                            <span
class="foodImg"><img :src="imgUrl+goods.goodsImg" width="100%"></span>
                            <span
class="foodName">{{goods.goodsName}}</span>
                            <span class="foodPrice">￥{{goods.price}}元</span>
                        </li>
                    </ul>
                    </el-tab-pane>
                </el-tabs>
            </div>
        </el-col>
    </el-row>
  </div>
 </div>
</template>
```

4. Pos.vue 组件，script 标签中代码如下。

```
<script>
import LeftNav from '@/components/common/LeftNav'    // 导入左侧组件
export default {
  name: 'Pos',
  components:{
    LeftNav
  },
    data () {
      return {
            type1Goods:[],
            type2Goods:[],
            type3Goods:[]
      }
  },
 created(){
        //向服务端发送请求，读取分类商品列表
       this.$http.get('/goodslist')
```

```
                .then(response=>{
                    this.type1Goods=response.data.list.type1;
                    this.type2Goods=response.data.list.type2;
                    this.type3Goods=response.data.list.type3;
                })
                .catch(error=>{
                     alert('网络错误，不能访问');
                })
            }
        }
    </script>
```

1.8 常用商品列表功能开发

1.8.1 需求分析

本节将完成项目的常用商品列表功能，包含常用商品列表功能服务端开发和常用商品列表功能客户端开发。

1.8.2 常用商品列表功能服务端开发思路

商品列表功能服务端开发需要用到的模块有 koa、koa2-cors、koa-body、koa-route、koa-static、path 等，程序运行逻辑如下：

1. 客户端向服务端发起请求，将需要查询的常用商品参数及数量发送到服务端；
2. 服务端获取请求，在数据库商品集合中通过 offten 字段进行查询，将查询结果作为响应返回；
3. 客户端获取响应，将响应数据渲染在页面中。

1.8.3 常用商品列表功能服务端核心代码

1. 在服务端程序文件夹 route 中创建 goodsofftenlist.js 文件，该文件是常用商品列表功能的路由文件，代码如下。

```
const Good = require("../model/good.js");
//商品列表查询功能
const goodslist = async function(ctx) {
    let ctx_query = ctx.query;
    let num = Number(ctx_query.num);
    let offten =1
      let result = {
           errCode: 0
      };
    let goods = await Good.find({goodOfften: offten}).limit(num);
    if(goods.length==0){
        result.errCode = 1
```

```
            result.errMsg = '没有找到查询结果'
            ctx.body = result
            return
        }else{
            result.errCode = 0
            result.errMsg = '查询到结果'
            result.list=goods
            ctx.body = result
            return
        }
    };
    module.exports = goodslist;
```

2. 在服务端程序入口文件 app.js 中载入常用商品列表功能的路由，添加的代码如下。

```
app.use(route.get('/goodsofftenlist', require("./route/goodsofftenlist.js")));
```

常用商品功能服务端接口开发完成，接口信息见表 1-8 所示。

表 1-8 常用商品功能接口

接口名	接口地址	请求方式	请求参数	参数描述
常用商品	服务器地址+/goodsofftenlist	post	num	查询数量

1.8.4 常用商品列表功能客户端开发思路

常用商品分类功能客户端功能，实现思路如下：

1. 在点餐组件右侧添加常用商品结构布局。
2. 在点餐组件中通过 axios 向服务端发送 POST 请求，获取常用商品数据并渲染到页面结构中。

1.8.5 常用商品列表功能客户端核心代码

1. 在 Pos.vue 组件找到 1.7.5 预留的常用商品列表注释代码，在注释下放添加布局结构代码，代码如下。

```
<div class="often-goods">
    <div class="title">常用商品</div>
        <div class="often-goods-list">
            <ul>
                <li v-for="goods in oftenGoods" @click="addOrderList(goods)">
                    <span>{{goods.goodsName}}</span>
                    <span class="o-price">￥{{goods.price}}元</span>
                </li>
            </ul>
        </div>
    </div>
```

2. 在 Pos.vue 组件 data 选项中声明 oftenGoods 数组并在 created 选项中使用 axios 向服务端发送请

求，代码如下。

```
    data () {
    return {
        oftenGoods:[],
    }
  },
created(){
    this.$http.get('/goodsofftenlist',{                    // 获取常用商品列表
        params:{
          num:10,
        }
      })
      .then(response=>{
        this.oftenGoods=response.data.list;
      })
      .catch(error=>{
        this.$message('网络连接失败,请稍候再试！')
        this.$router.push('/')
      })
}
```

1.9 下单功能开发

1.9.1 需求分析

本节讲完成项目的下单功能，包含下单功能服务端开发和下单功能客户端开发。

1.9.2 下单功能服务端开发思路

下单功能服务端开发需要用到的模块有 mongoose、koa、koa2-cors、koa-body、koa-route 等，程序运行逻辑如下：

1. 客户端将用户名、授权码、已选商品列表以 POST 请求方式发送到服务端；
2. 服务端获取请求，核对授权信息是否合法；
3. 如果授权信息合法，将订单信息保存在数据库中，返回响应；
4. 如果授权信息不合法，直接返回响应；
5. 客户端根据响应在页面输出提示或进行页面跳转。

1.9.3 下单功能服务端核心代码

1. 在服务端程序文件夹 model 中创建 orderlist.js 文件，该文件是订单集合的模型文件，代码如下。

```
let mongoose = require('../mong/mongo.js'),      //加载 mongoose 对象
    Schema = mongoose.Schema;
```

```
let OrderlistSchema = new Schema({
    username: { type: String },
    goodslist: {type: Object},
    creattime: {type: Date},
    orderstate: {type: String},
    ordertotal: {type: Number}
});
module.exports = mongoose.model('Orderlist',OrderlistSchema)
```

2. 在服务端程序文件夹 route 中创建 orderlistAdd.js 文件，该文件是下单功能的路由文件，代码如下。

```
const Token = require("../model/token.js");
const Orderlist = require("../model/orderlist.js");
//订单添加功能
const orderlistAdd = async function(ctx) {
        let tokenid = ctx.request.body.tokenid;
        let username = ctx.request.body.username;
        let goodslist = ctx.request.body.orderList;
        let ordertotal = ctx.request.body.orderTotal;
    let creattime = Date.now();
    let result = {
        errCode: 0
    };
  let seachtoken = await Token.findOne({
            tokenid: tokenid
        }).exec()
  if (!seachtoken||seachtoken.overtime<creattime) {
        result.errCode = 1
        result.errMsg = '登录超时或者未登录，请重新登录'
        ctx.body = result
        return
 }else {
        orderlistadd = new Orderlist({
            'username' : username,
            'creattime' : creattime,
            'orderstate' : "nopay",
            'goodslist' : goodslist,
            'ordertotal':ordertotal
        })
```

```
            orderlistadd = await orderlistadd.save();
            result.errCode = 0
            result.errMsg = '订单增加成功'
            ctx.body = result
            return
        }
    };
    module.exports = orderlistAdd;
```

3. 在服务端程序入口文件 app.js 中载入下单功能的路由，添加的代码如下。

```
app.use(route.post('/orderlistAdd', require("./route/orderlistAdd.js")));
```

下单功能服务端接口开发完成，接口信息见表 1-9 所示。

表 1-9 下单功能接口

接口名	接口地址	请求方式	请求参数	参数描述
下单	服务器地址+/orderlistAdd	post	tokenid	授权码
			username	用户名
			orderList	订单详情
			orderTotal	订单总价

1.9.4 下单功能客户端开发思路

下单功能实现思路如下：

1. 在点餐组件中使用 Element UI 完成中间区域“点餐”列表布局。
2. 点击右侧任意商品时，根据点击商品 id 进行比对，如果该商品存在于待支付订单中，就增加该商品个数，否则将该商品添加至待支付订单数组中，并触发价格统计函数。
3. 在待支付订单中，可根据选中的商品 id，删除该商品；删除完成后触发价格统计函数。
4. 点击待支付订单中的“清空”按钮，可以清空待支付订单中所有商品，实现方法，只需将订单数组、价格总价以及商品总数清空即可。
5. 点击“结账”按钮，判断商品总数是否为零，如果不为零，将遍历待支付订单中所有商品的名称、数量及商品价格，最后将遍历得到的对象存放在订单详情空数组中。
6. 通过 axios 向服务端发送请求，并提交授权码、用户名、订单详情及订单总价。

1.9.5 下单功能客户端核心代码

1. 在 Pos.vue 组件 template 标签内，找到 1.7.5 预留的待支付订单结构注释代码，在注释下放添加布局结构代码，代码如下。

```
<el-col :span='7' class="post-list" id="order-list">
    <el-tabs v-model="activeName" @tab-click="handleClick">
        <el-tab-pane label="点餐" name="first">
            <el-table :data="tableData" border style="width: 100%" >
```

```
                <el-table-column prop="goodsName" label="商品"  ></el-table-column>
                    <el-table-column prop="count" label="数量" align="center" width="50"></el-table-column>
                    <el-table-column prop="price" label="金额"  align="center" width="70"></el-table-column>
                    <el-table-column   label="操作" width="100" fixed="right" align="center">
                        <template slot-scope="scope">
                            <el-button type="text" size="small" @click="delSingleGoods(scope.row)">删除</el-button>
                        </template>
                    </el-table-column>
                </el-table>
                    <p style="text-align:right;padding-right:10px">数量：{{totalCount}} 总价：{{totalMoney}}元</p>
                <!-- 三个按钮 -->
                <div class="btn-group">
                    <el-button type="danger" @click="delAllGoods">清空</el-button>
                    <el-button type="success" @click="checkout">结账</el-button>
                </div>
            </el-tab-pane>
            <el-tab-pane label="订单列表" name="second">
                <div class="orderlist">
                    <div class="order" v-for="(order,index) in orderList">
                        <div class="order-hd">
                            订单编号：{{order._id}}
                            <span>订单总价：{{order.ordertotal}}元</span>
                        </div>
                        <div class="goods-info">
                            <table class="order-table" cellpadding="0" cellspacing="0">
                                <tr>
                                    <td>商品名称</td>
                                    <td>商品数量</td>
                                    <td>商品单价</td>
                                </tr>
                                <tr v-for="goodsInfo in order.goodslist">
                                    <td>{{goodsInfo.orderName}}</td>
                                    <td>{{goodsInfo.orderNum}}</td>
```

```
                                    <td>{{goodsInfo.orderPrice}}</td>
                                </tr>
                            </table>
                        </div>
                    </div>
                </div>
            </el-tab-pane>
        </el-tabs>
    </el-col>
```

2. 在 Pos.vue 组件 data 选项代码如下。

```
    data () {
    return {
        activeName: 'first',
        totalMoney:0,
        totalCount:0,
        tableData:[],      // 待支付数组
        orderList:[]       // 订单详情数组（提交给服务端）
    }
  }
```

3. 在 Pos.vue 组件 methods 选项中，编写添加待支付订单功能，代码如下。

```
// 添加待支付订单功能
addOrderList(goods){
    this.goodsList=[];
    this.totalCount=0; //汇总数量清 0
    this.totalMoney=0;
    let isHave=false;
    //判断是否这个商品已经存在于订单列表
    for (let i=0; i<this.tableData.length;i++){
        if(this.tableData[i].goodsId==goods._id){
            isHave=true; //存在
        }
    }
    //根据 isHave 的值判断订单列表中是否已经有此商品
    if(isHave){
        //存在就进行数量添加
            let arr = this.tableData.filter(o =>o.goodsId == goods._id);
            arr[0].count++;
```

```
        }else{
            //不存在就推入数组
            let
newGoods={goodsId:goods._id,goodsName:goods.goodsName,price:goods.price,count:go
ods.count};
                this.tableData.push(newGoods);
        }
        this. getAllMoney()              // 执行价格统计
    },
```

4. 在 Pos.vue 组件 methods 选项中，编写删除单个商品功能，代码如下。

```
//删除单个商品
  delSingleGoods(goods){
     this.tableData=this.tableData.filter(o => o.goodsId != goods.goodsId);
     this.getAllMoney()
   },
```

5. 在 Pos.vue 组件 methods 选项中，编写汇总数量和金额功能，代码如下。

```
// 数量汇总和金额
getAllMoney(){
     this.totalCount=0;
     this.totalMoney=0;
     if(this.tableData){
          this.tableData.forEach(element => {
          this.totalCount+=element.count;
          this.totalMoney=this.totalMoney+(element.price*element.count);
     });
     }
},
```

6. 在 Pos.vue 组件 methods 选项中，编写删除所有商品功能，代码如下。

```
// 删除所有商品
delAllGoods() {
          this.tableData = [];          // 清空待支付订单
          this.totalCount = 0;          // 设置商品总数为 0
          this.totalMoney = 0;          // 设置总金额为 0
   },
```

7. 在 Pos.vue 组件 methods 选项中，编写订单提交功能，代码如下。

```
// 订单提交功能
checkout() {                                          //   提交订单
```

```
                if (this.totalCount!=0) {
                for (let i = 0; i < this.tableData.length; i++) {
                 var goodsArr =
{orderName:this.tableData[i].goodsName,orderNum:this.tableData[i].count,orderPrice:this.tableData[i
].price}
                    this.orderList.push(goodsArr)
                }
                this.$http.post('/orderlistAdd',{                    //  向服务端提交订单
                    tokenid:localStorage.getItem('tokenID'),
                    username:localStorage.getItem('userName'),
                    orderList:this.orderList,
                    orderTotal:this.totalMoney
                })
                .then(response=>{                                    //  提交订单完成后重置待支付订单
                    this.tableData = [];
                    this.totalCount = 0;
                    this.totalMoney = 0;
                    this.orderList = [];
                    this.$message({
                            message: '结账成功，欢迎下次光临!',
                            type: 'success'
                        });
                    this.orderListGet()                    // 提交完成后需要重新获取已支付订单（暂未实现）
                    })
                }
                else{
                    this.$message.error('未选择商品，下单失败！');
                }
            }
```

1.10 订单查询开发

1.10.1 需求分析

本节讲完成项目的订单查询功能，包含订单查询功能服务端开发和订单查询功能客户端开发。

1.10.2 订单查询服务端开发思路

订单查询功能服务端开发需要用到的模块有 mongoose、koa、koa2-cors、koa-body、koa-route 等，程序运行逻辑如下：

1. 客户端将用户名、授权码以 POST 请求方式发送到服务端；
2. 服务端获取请求，核对授权信息是否合法；
3. 如果授权信息合法，将用户名放到订单集合中查询，并将查询结果返回响应；
4. 如果授权信息不合法，直接返回响应；
5. 客户端根据响应在页面输出订单信息。

1.10.3 订单查询服务端核心代码

1. 在服务端程序文件夹 route 中创建 orderlistGet.js 文件，该文件是订单查询功能的路由文件，代码如下。

```
const Token = require("../model/token.js");
const Orderlist = require("../model/orderlist.js");
//商品详情查询功能
const orderlistGet = async function(ctx) {
        let tokenid = ctx.request.body.tokenid;
        let username = ctx.request.body.username;
    let nowtime = Date.now();
    let result = {
        errCode: 0
    };
    let seachtoken = await Token.findOne({
            tokenid : tokenid
        }).exec()

    if (!seachtoken||seachtoken.overtime<nowtime) {
        result.errCode = 1
        result.errMsg = '登录超时或者未登录，请重新登录'
        ctx.body = result
        return
    }else {
        let orderconget = await Orderlist.find({
            "username" : username,
            "orderstate" : "nopay"
        }).exec()
        if(!orderconget){
            result.errCode = 2
          result.errMsg = '没有订单'
          ctx.body = result
            return
```

```
            }else{
                result.errCode = 0
                result.errMsg = '查询订单成功'
                result.list=orderconget
                ctx.body = result
                return
            }
        }
    };
    module.exports = orderlistGet;
```

2. 在服务端程序入口文件 app.js 中载入下单功能的路由，添加的代码如下。

```
app.use(route.post('/orderlistGet', require("./route/orderlistGet.js")));
```

订单查询功能服务端接口开发完成，接口信息见表 1-10 所示。

表 1-10 订单查询功能接口

接口名	接口地址	请求方式	请求参数	参数描述
订单查询	服务器地址+/orderlistGet	post	tokenid	授权码
			username	用户名

1.10.4 订单查询客户端开发思路

订单查询功能实现思路如下：

1. 当进入点餐组件时，需要向服务端发送请求获取已支付订单数据，点击“订单列表”按钮显示订单详情。
2. 当点击“结账”按钮时，需要重新向服务端发送请求获取已支付订单数据。
3. 当进入点餐组件时，需要授权验证，禁止用户直接访问点餐组件。

1.10.5 订单查询客户端核心代码

1. 通过路由进入 Pos.vue 组件时，需要向服务端发送请求获取已支付订单数据，在 Pos.vue 组件中添加 mounted 选项，代码如下。

```
mounted:function(){
        this.orderListGet()          // 执行订单查询函数
    },
```

2. 在 Pos.vue 组件 methods 选项中，编写订单查询功能，代码如下。

```
orderListGet(){                                          // 获取订单列表
    this.$http.post('/orderlistGet',{
        tokenid:localStorage.getItem('tokenID'),
        username:localStorage.getItem('userName')
```

```
        })
        .then(response=>{
            this.orderList = response.data.list
            console.log(response)
        })
        .catch(error=>{
            this.$message.error('网络问题，订单列表读取失败！');
        })
    }
```

3. 在 Pos.vue 组件 script 标签中，添加路由钩子函数实现授权验证功能，代码如下。

```
    // 访问授权验证
    beforeRouteEnter: (to, from, next) => {
        var checkTokenID=localStorage.getItem('tokenID')
        if (!checkTokenID) {
            next('/')
        }
        else{
            next()
        }
    }
```