1. 系统总体设计

传统的跑腿式校车预约流程十分烦琐，所以校车预约系统的目的是实现传递流程的无纸化和便捷性。出于对系统安全性的考虑，需要给不同的使用者分配相对应的权限，使用者只能在此权限内进行操作。通过对使用教室的申请流程的调查了解，再根据本校实际情况，对教室申请者 (以下称为用户) 和教室管理者 (以下称为管理员) 赋予以下功能需求:

1) 用户: 用户登录系统后可以查询校车使用情况，根据需要填写申请信息并提交申请;提交申请后可以查看目前审核情况。

2) 管理员: 管理员登录系统后可以查看到目前空校车的基本信息、申请理由、申请状态，对已提交申请的教室进行审核。此外，系统界面必须简洁美观，交互操作必须简单方便，核心功能要放在显眼的位置。

* 1. 功能模块设计

1. 登录认证模块。用户需要通过账户密码进行登录，方可使用系统进行操作，以确保系统使用的安全性。具体实现效果如图所示。
2. 图形用户界面, 文本, 应用程序

   描述已自动生成

2)校车预约模块。用户登录后进入系统首页，点击“预约”可进行预约操作。点击“预约”后进入预约成功，再次点击后可取消预约。具体实现效果如图所示。

图形用户界面, 应用程序

描述已自动生成

3)用户管理模块。用户登录后进入系统首页，点击“我的”可进入用户管理页面。页面显示用户或管理员基本信息，可删除或添加角色。具体实现效果如图所示。

图形用户界面, 应用程序

描述已自动生成

通过对上述需求以及功能模块分析，开发人员能够根据需求进行相对应的模块开发，从而可以画出对应的系统功能模块结构图如图3-1所示。

图示

描述已自动生成

图3-1 系统功能模块图

* 1. 系统流程分析

首先进入系统主页，用户可以以自身角色身份浏览，用户可以通过想要的日期选择感兴趣的校车进行校车预约操作。

用户在相关角色管理页可查看相关的角色，如用户，管理员，可通过添加或删除角色进行角色管理操作。

用户登录后进入个人中心可查看个人信息、收货地址、订单信息。系统流程图如图3-2所示，登录子流程图如图3-3所示。

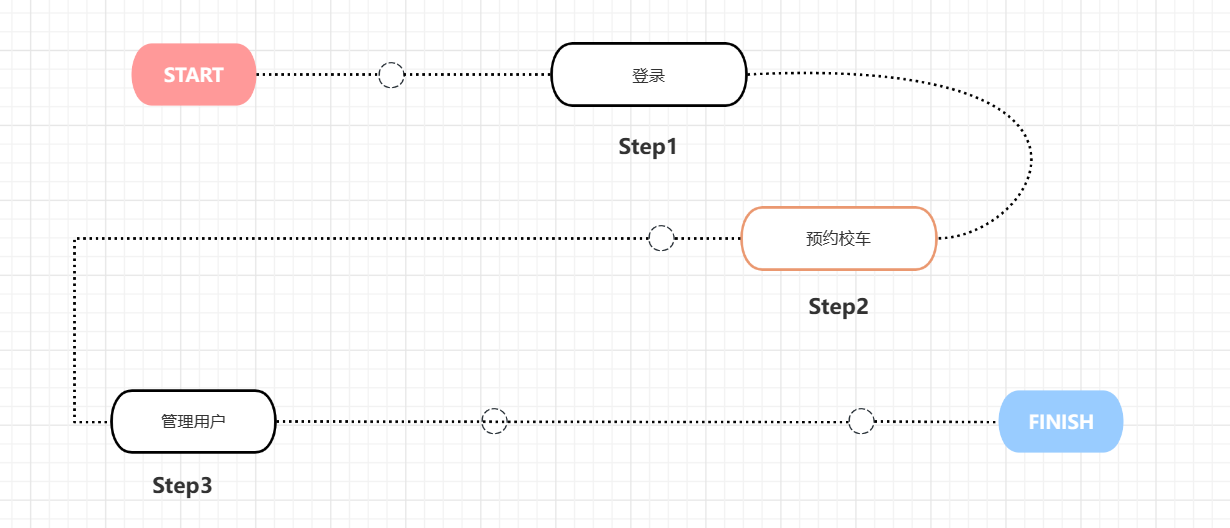


图3-2 系统流程图

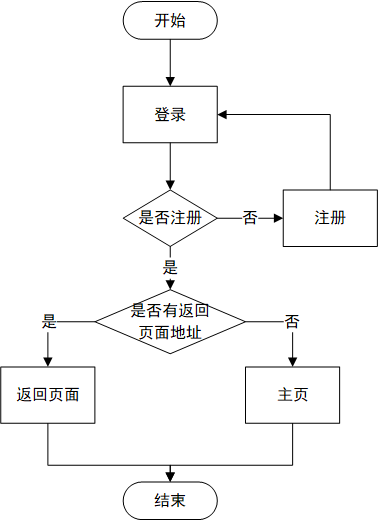


图3-3 登录子流程图

homePageAdmin页面

将startTime转换为对应的格式，再设置startFormStatus弹出层状态为true，以及将Form赋值

// 预约成员事件 设置状态为true，数组为空数组，然后通过startTime属性的值查找云数据库的数据，查找数据后将该值赋值给reservationList。

StartFormClick(e) {

let self = this

self.setData({

reservationStatus: true,

reservationList: []

})

const db = wx.cloud.database()

db.collection('reservation').where({

startTime: e.currentTarget.dataset.item.startTime

}).get({

success(res){

self.data.reservationList = res.data

self.setData({

reservationList: self.data.reservationList

})

}

})

},

// 删除

delClick(e) {

let self = this

wx.cloud.database().collection('car\_record').doc(e.currentTarget.dataset.item.\_id).remove().then(res=>{

Message.success({

context: self,

offset: [20, 32],

duration: 5000,

content: '删除成功',

});

self.getDataStartPlan()

})

},

// 根据日期获取当天发车计划

getDataStartPlan() {

let self = this

self.setData({

startPlanList: []

})

let startPlanList = []

const db = wx.cloud.database()

let dataTime = self.data.selectDay.year + '-' + (self.data.selectDay.month > 10?self.data.selectDay.month:'0' + self.data.selectDay.month) + '-' + (self.data.selectDay.day > 10?self.data.selectDay.day:'0' + self.data.selectDay.day)

db.collection('car\_record').where({

startTime: db.RegExp({

//inputValue为输入框的值，也是就要查询内容，可以自己定义

regexp: dataTime,

//大小写不区分

options: 'i',

})

}).get({

success(res){

if(res.data && res.data.length > 0) {

res.data.map(item => {

item.day = dayjs(item.startTime).format('YYYY年MM月DD日')

item.hour = dayjs(item.startTime).format('HH:mm')

})

startPlanList = res.data

}

self.setData({

startPlanList

})

}

})

},

// 选择日期和事件确定事件

onConfirm(e) {

let self = this

self.data.startInfoForm.startTime = e.detail.value

self.setData({

startInfoForm: self.data.startInfoForm

})

},

// 点击发车时间事件

startFormBindTap() {

let self = this

self.setData({

datetimeVisible: true

})

},

// 新增发车时间

onIconTap() {

let self = this

self.data.startInfoForm.startTime = dayjs().format('YYYY-MM-DD HH:mm')

self.setData({

startFormStatus: true,

startInfoForm: self.data.startInfoForm

})

},

// 监听输入框事件

startFormInputChange(e) {

let self = this

self.data.startInfoForm[e.currentTarget.dataset.name] = e.detail.value

self.setData({

startInfoForm: self.data.startInfoForm

})

},

// 新增表单保存

saveAdd() {

let self = this

wx.cloud.database().collection('car\_record').add({

data:{

startTime: self.data.startInfoForm.startTime, // 发车时间

carNumber: self.data.startInfoForm.carNumber,// 车牌号

startlocation: self.data.startInfoForm.startlocation, // 发车地点

getonLocalhost: self.data.startInfoForm.getonLocalhost,// 上车地点

},

success(res){

Message.success({

context: self,

offset: [20, 32],

duration: 5000,

content: '增加成功',

});

self.setData({

startFormStatus: false

})

}

})

},

swiperChange(e) { // 日历滑动时触发的方法

if (this.data.backChange) {

this.setData({

backChange: false

})

return

}

//计算第三个索引

let rest = 3 - e.detail.current - this.data.oldCurrent

let dif = e.detail.current - this.data.oldCurrent

let date

if (dif === -2 || (dif > 0 && dif !== 2)) { //向右划的情况，日期增加

if (this.data.open) {

date = new Date(this.data.selectDay.year, this.data.selectDay.month)

this.setMonth(date.getFullYear(), date.getMonth() + 1, undefined)

this.getIndexList({

setYear: this.data.selectDay.year,

setMonth: this.data.selectDay.month,

dateIndex: rest

})

} else {

date = new Date(this.data.selectDay.year, this.data.selectDay.month - 1, this.data.selectDay.day + 7)

this.setMonth(date.getFullYear(), date.getMonth() + 1, date.getDate())

this.getIndexList({

setYear: this.data.selectDay.year,

setMonth: this.data.selectDay.month - 1,

setDay: this.data.selectDay.day + 7,

dateIndex: rest

})

}

} else { //向左划的情况，日期减少

if (this.data.open) {

date = new Date(this.data.selectDay.year, this.data.selectDay.month - 2)

this.setMonth(date.getFullYear(), date.getMonth() + 1, undefined)

this.getIndexList({

setYear: this.data.selectDay.year,

setMonth: this.data.selectDay.month - 2,

dateIndex: rest

})

} else {

date = new Date(this.data.selectDay.year, this.data.selectDay.month - 1, this.data.selectDay.day - 7)

this.setMonth(date.getFullYear(), date.getMonth() + 1, date.getDate())

this.getIndexList({

setYear: this.data.selectDay.year,

setMonth: this.data.selectDay.month - 1,

setDay: this.data.selectDay.day - 7,

dateIndex: rest

})

}

}

this.setData({

oldCurrent: e.detail.current

})

this.setSwiperHeight(e.detail.current)

},

setSwiperHeight(index) { // 根据指定位置数组的大小计算长度

this.setData({

swiperHeight: this.data[`dateList${index}`].length / 7 \* 82 + 18

})

},

//更新指定的索引和月份的列表

getIndexList({

setYear,

setMonth,

setDay = void 0,

dateIndex

}) {

let appointMonth

if (setDay)

appointMonth = new Date(setYear, setMonth, setDay)

else

appointMonth = new Date(setYear, setMonth)

let listName = `dateList${dateIndex}`

this.setData({

[listName]: this.dateInit({

setYear: appointMonth.getFullYear(),

setMonth: appointMonth.getMonth() + 1,

setDay: appointMonth.getDate(),

hasBack: true

}),

})

},

//设置月份

setMonth(setYear, setMonth, setDay) {

const day = Math.min(new Date(setYear, setMonth, 0).getDate(), this.data.selectDay.day)

if (this.data.selectDay.year !== setYear || this.data.selectDay.month !== setMonth) {

const data = {

selectDay: {

year: setYear,

month: setMonth,

day: setDay ? setDay : day

},

}

if (!setDay) {

data.open = true

}

this.setData(data, () => {

this.triggerEvent("selectDay", this.data.selectDay)

})

} else {

const data = {

selectDay: {

year: setYear,

month: setMonth,

day: setDay ? setDay : day

},

}

this.setData(data, () => {

this.triggerEvent("selectDay", this.data.selectDay)

})

}

},

//展开收起

openChange() {

this.setData({

open: !this.data.open

})

this.triggerEvent("aaa", {

a: 0

})

// 更新数据

const selectDate = new Date(this.data.selectDay.year, this.data.selectDay.month - 1, this.data.selectDay.day)

if (this.data.oldCurrent === 0) {

this.updateList(selectDate, -1, 2)

this.updateList(selectDate, 0, 0)

this.updateList(selectDate, 1, 1)

} else if (this.data.oldCurrent === 1) {

this.updateList(selectDate, -1, 0)

this.updateList(selectDate, 0, 1)

this.updateList(selectDate, 1, 2)

} else if (this.data.oldCurrent === 2) {

this.updateList(selectDate, -1, 1)

this.updateList(selectDate, 0, 2)

this.updateList(selectDate, 1, 0)

}

this.setSwiperHeight(this.data.oldCurrent)

},

// 选中并切换今日日期

switchNowDate() {

const now = new Date()

const selectDate = new Date(this.data.selectDay.year, this.data.selectDay.month - 1, this.data.selectDay.day)

let dateDiff = (selectDate.getFullYear() - now.getFullYear()) \* 12 + (selectDate.getMonth() - now.getMonth())

let diff = dateDiff === 0 ? 0 : dateDiff > 0 ? -1 : 1

const diffSum = (x) => (3 + (x % 3)) % 3

if (this.data.oldCurrent === 0) {

this.updateList(now, -1, diffSum(2 + diff))

this.updateList(now, 0, diffSum(0 + diff))

this.updateList(now, 1, diffSum(1 + diff))

} else if (this.data.oldCurrent === 1) {

this.updateList(now, -1, diffSum(0 + diff))

this.updateList(now, 0, diffSum(1 + diff))

this.updateList(now, 1, diffSum(2 + diff))

} else if (this.data.oldCurrent === 2) {

this.updateList(now, -1, diffSum(1 + diff))

this.updateList(now, 0, diffSum(2 + diff))

this.updateList(now, 1, diffSum(0 + diff))

}

this.setData({

swiperCurrent: diffSum(this.data.oldCurrent + diff),

oldCurrent: diffSum(this.data.oldCurrent + diff),

backChange: dateDiff !== 0,

})

this.setData({

selectDay: {

year: now.getFullYear(),

month: now.getMonth() + 1,

day: now.getDate()

}

}, () => {

this.triggerEvent("selectDay", this.data.selectDay)

})

this.setSwiperHeight(this.data.oldCurrent)

},

//日历主体的渲染方法

dateInit({

setYear,

setMonth,

setDay = this.data.selectDay.day,

hasBack = false

} = {

setYear: this.data.selectDay.year,

setMonth: this.data.selectDay.month,

setDay: this.data.selectDay.day,

hasBack: false

}) {

let dateList = []; //需要遍历的日历数组数据

let now = new Date(setYear, setMonth - 1) //当前月份的1号

let startWeek = now.getDay(); //目标月1号对应的星期

let resetStartWeek = startWeek == 0 ? 6 : startWeek - 1; //重新定义星期将星期天替换为6其余-1

let dayNum = new Date(setYear, setMonth, 0).getDate() //当前月有多少天

let forNum = Math.ceil((resetStartWeek + dayNum) / 7) \* 7 //当前月跨越的周数

let selectDay = setDay ? setDay : this.data.selectDay.day

this.triggerEvent("getDateList", {

setYear: now.getFullYear(),

setMonth: now.getMonth() + 1

})

if (this.data.open) {

//展开状态，需要渲染完整的月份

for (let i = 0; i < forNum; i++) {

const now2 = new Date(now)

now2.setDate(i - resetStartWeek + 1)

let obj = {};

obj = {

day: now2.getDate(),

month: now2.getMonth() + 1,

year: now2.getFullYear()

};

dateList[i] = obj;

}

} else {

//非展开状态，只需要渲染当前周

for (let i = 0; i < 7; i++) {

const now2 = new Date(now)

//当前周的7天

now2.setDate(Math.ceil((selectDay + (startWeek - 1)) / 7) \* 7 - 6 - (startWeek - 1) + i)

let obj = {};

obj = {

day: now2.getDate(),

month: now2.getMonth() + 1,

year: now2.getFullYear()

};

dateList[i] = obj;

}

}

if (hasBack) {

return dateList

}

this.setData({

dateList1: dateList

})

},

//一天被点击时

selectChange(e) {

const year = e.currentTarget.dataset.year

const month = e.currentTarget.dataset.month

const day = e.currentTarget.dataset.day

const selectDay = {

year: year,

month: month,

day: day,

}

if (this.data.open && (this.data.selectDay.year !== year || this.data.selectDay.month !== month)) {

if ((year \* 12 + month) > (this.data.selectDay.year \* 12 + this.data.selectDay.month)) { // 下个月

if (this.data.oldCurrent == 2)

this.setData({

swiperCurrent: 0

})

else

this.setData({

swiperCurrent: this.data.oldCurrent + 1

})

} else { // 点击上个月

if (this.data.oldCurrent == 0)

this.setData({

swiperCurrent: 2

})

else

this.setData({

swiperCurrent: this.data.oldCurrent - 1

})

}

this.setData({

['selectDay.day']: day

}, () => {

this.triggerEvent("selectDay", this.data.selectDay)

})

} else if (this.data.selectDay.day !== day) {

this.setData({

selectDay: selectDay

}, () => {

this.triggerEvent("selectDay", this.data.selectDay)

})

}

this.getDataStartPlan()

},

updateList(date, offset, index) {

if (this.data.open) { //打开状态

const setDate = new Date(date.getFullYear(), date.getMonth() + offset \* 1) //取得当前日期的上个月日期

this.getIndexList({

setYear: setDate.getFullYear(),

setMonth: setDate.getMonth(),

dateIndex: index

})

} else {

const setDate = new Date(date.getFullYear(), date.getMonth(), date.getDate() + offset \* 7) //取得当前日期的七天后的日期

this.getIndexList({

setYear: setDate.getFullYear(),

setMonth: setDate.getMonth(),

setDay: setDate.getDate(),

dateIndex: index

})

}

},

},

lifetimes: {

attached() {

let now = this.data.defaultTime ? new Date(this.data.defaultTime) : new Date()

let selectDay = {

year: now.getFullYear(),

month: now.getMonth() + 1,

day: now.getDate()

}

this.setData({

nowDay: {

year: now.getFullYear(),

month: now.getMonth() + 1,

day: now.getDate()

},

selectDay

})

this.setMonth(selectDay.year, selectDay.month, selectDay.day)

this.updateList(now, -1, 0)

this.updateList(now, 0, 1)

this.updateList(now, 1, 2)

this.setSwiperHeight(1)

this.getDataStartPlan()

}

1. 系统详细设计
   1. 用户注册模块

用户通过注册从游客身份转为商城用户，并在系统中保存用户的用户名及验证下一次登录时需要验证身份的密码，因此在注册页面设计了用户名、设置初始密码、确认密码三个文本输入框及注册按钮。

注册页面如图5-1所示：



图5-1 注册页面

具体实现流程：

在用户输入用户名、密码及确认密码时，在前端页面对其输入的值进行动态校验，用户名规定为长度符合8到20个的字符，密码规定为长度符合6到16个的字符，并且再确认密码处检验确认密码是否与初始密码相同。当这些规定都满足时，用户注册信息符合规范，标红提示信息不再出现。否则需重新输入相关信息。在前端vue中，将用户名、密码、确认密码动态的添加到vm模型中，以便提交注册时将数据传输给后端。点击提交注册按钮后，通过ajax通过post异步请求到地址为user/regist的url，并将数据请求给后端，请求结束后获得一个json格式的结果，当结果中的状态码为10000时则注册成功，否则失败。

后端通过url映射到Post请求接口，调用控制层的方法，再调用业务逻辑层，对前端传过来的用户名进行查询，寻找该用户名是否被注册，无则对用户的基本信息进行设置并通过MD5Utils工具类的md5方法对密码进行加密返回一个加密后的字符串，封装到user对象中，调用数据访问层对用户数据进行保存，保存成功则返回状态码、状态信息、数据。

后端查询用户名时使用mybatis的逆向工程中生成实例的对应的example对象，example用于添加条件，相当SQL语句where条件，可以便于简单数据的增删改查，以下是查询用户名是否被注册的后端代码：

|  |
| --- |
| Example example = new Example(Users.class);  Example.Criteria criteria = example.createCriteria();  criteria.andEqualTo("username", name);//添加username字段等于name条件  List<Users> users = usersMapper.selectByExample(example);//查条件下的user |

* 1. 用户登录模块

用户登录模块是用户对后续添加购买查看操作的前提模块，没有登录就没有访问权限，只有浏览商品的权限，而不可进行下一步操作。本系统对登录界面设计了用户名、密码两个文本输入框及登录、注册两个按钮。

登录页面如图5-2所示：



图5-2 登录页面

具体实现流程：

用户在前端页面键入用户名和密码，前端会动态的校验用户名和密码，并保存到vm模型中，用户点击登录后，前端通过ajax的get请求，将用户名、密码携带请求到url地址访问后端接口。得到相应结果后，当状态码为10000时，将token、用户ID、用户名、用户头像保存到cookie中，用于下一个页面信息显示，随后跳转到主页面。

后端得到请求后，先对用户名进行查询，查询用户名是否存在，若存在，即对密码进行验证，在前面注册时密码进行了加密，则在登录时密码也要经过同样的加密，两者加密之后对比加密字符串相同则认为密码输入正确，利用JWT中的Jwts类的builder方法生成JwtBuilder对象，通过该对象的一系列set方法生成token加密字符串，用于访问权限设置，最后返回状态码、token以及user对象。

// 弹窗确定事件 绑定classify和role的值

onPickerChange(e) {

        this.data.loginForm.classify = e.detail.label[0]

        this.data.loginForm.role = e.detail.value[0]

        this.setData({

            loginForm: this.data.loginForm

        })

    },

// 监听文本框输入 双向绑定表单的值

bindchange(e) {

let self = this

let name = e.currentTarget.dataset.name

self.data.loginForm[name] = e. detail.value

self.setData({

loginForm: self.data.loginForm

})

},

// 登录 通过云数据库查询用户名和密码再判断用户输入的值，如果输入正确则跳转至homePageAdmin页面，期间通过Message框提示用户进行操作。

loginClick() {

let self = this

let loginForm = self.data.loginForm

if (!loginForm.username) {

Message.error({

context: this,

offset: [20, 32],

duration: 5000,

content: '请输入用户名',

});

return

}

if (!loginForm.password) {

Message.error({

context: this,

offset: [20, 32],

duration: 5000,

content: '请输入密码',

});

return

}

// 查询用户信息

wx.cloud.database().collection('user').where({

username: loginForm.username,

role: loginForm.role

}).get({

success(res){

if (res.data.length > 0) {

let result = res.data[0]

// 判断密码是否正确

if (result.password == loginForm.password) {

appInstance.globalData.userInfo = loginForm

// 密码正确

Message.success({

context: self,

offset: [20, 32],

duration: 5000,

content: '登录成功',

});

if (loginForm.role == 0) {

wx.navigateTo({

url: '/pages/reservation/reservation',

})

} else if (loginForm.role == 1) {

wx.switchTab({

url: '/pages/homePageAdmin/homePageAdmin',

})

}

} else {

// 密码错误

Message.error({

context: self,

offset: [20, 32],

duration: 5000,

content: '用户名或密码不正确',

});

}

} else {

// 用户名不存在

Message.error({

context: self,

offset: [20, 32],

duration: 5000,

content: '用户名或密码不正确',

});

}

}

})

},

// 获取用户信息 通过云数据库向user表插入用户名为admin 密码为123456，role为1的角色，如果插入成功则通过小程序api获取微信用户的详细信息。

getUserinfo() {

// wx.cloud.database().collection('user').add({

// data:{

// username: 'admin',

// password: 123456,

// role: 1,

// },

// success(res){

// console.log('增加成功', res)

// }

// })

wx.cloud.database().collection('user').where({

}).get({

success(res){

console.log(res)

}

})

}

* 1. 主页展示模块

当用户第一次登录后，跳至主页面，商城的主页面包含用户名称的显示、搜索框、分类列表、轮播图、商品推荐、分类商品下的商品展示。在这里本系统将搜索、商品推荐抽离成单独模块，主页展示模块专注于分类列表、轮播图及零食的显示。主页展示页面如图5-3所示。

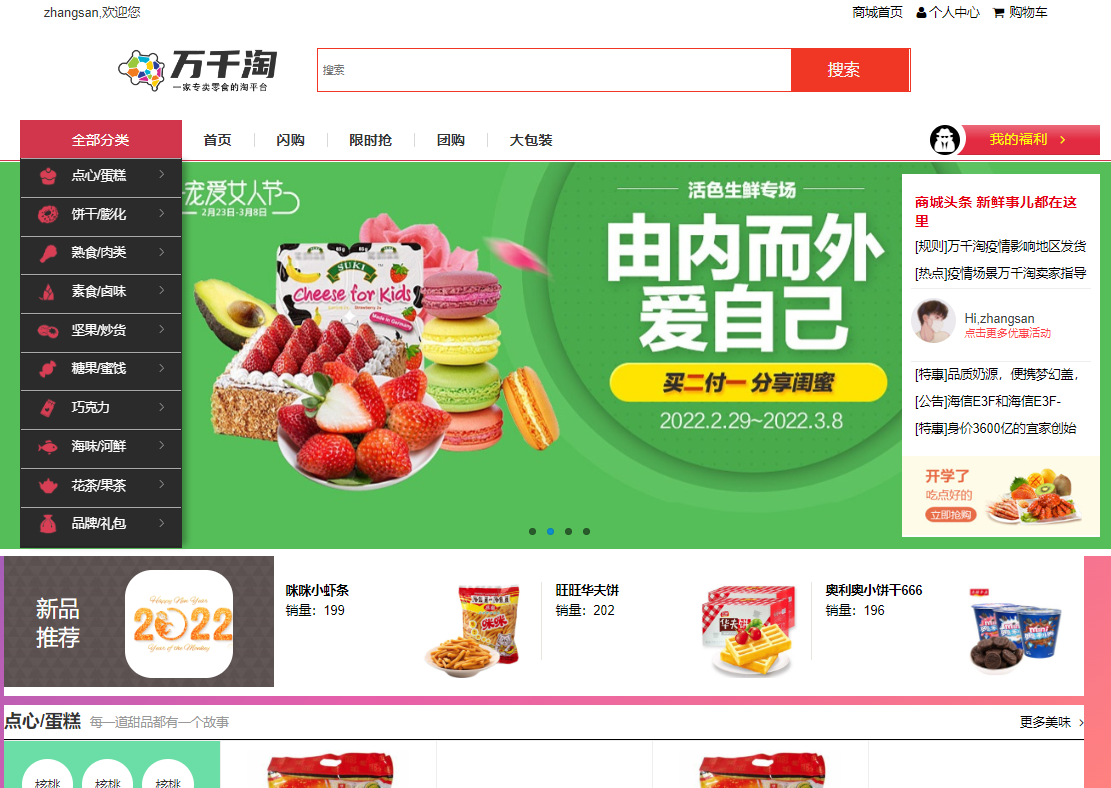


图5-3 主页展示

对于页面的所有信息，都应该在页面出现之前得到数据，用户点击登录之后跳转到主页，主页立刻向后端发送异步请求，请求数据并渲染页面。在本主页面中需要加载的数据有分类列表、轮播图、分类下的零食商品信息及其他功能模块数据。

对于分类列表，本系统采取一次性查询该一级分类下的二级三级分类，在这样的策略下用户鼠标移动至一级菜单列表时，系统不需要再次去查询该一级分类下的二级三级分类，使得用户选择一级菜单时数据响应时间快，分类列表图如图5-4显示。



图5-4 分类列表图

对于轮播图，前端向后端发送异步请求，请求轮播图信息并加载到vm模型中，进而显示到页面。

分类列表与轮播图的查询与加载类似，当查询得到结果之后需要重新添加动画效果，并且在对vm模型中定义的值赋值时，不能同时进行特别的渲染操作即设置动画效果，赋值和渲染属于同步操作，只有赋值先于渲染，渲染才可生效，本系统在渲染时加上延时器，使得赋值先于动画渲染，完成轮播图功能。

对于分类商品，前端向后端发送异步请求，后端查询一级分类下的的零食商品信息并返回结果集，前端进行渲染页面。

* 1. 商品推荐模块
  2. 个人中心模块

个人中心是用户对个人信息的管理中心，本系统在个人中心具有三个基本功能：

1. 用户个人信息的查询展示及修改，如图5-13所示。

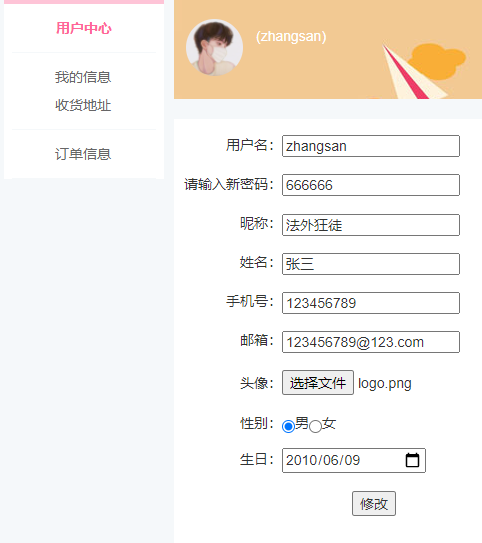


图5-13 个人信息展示