# Epet项目说明

## 路由

* 首页
  + 首页
  + 菜单项
* 分类
  + 产品分类
  + 品牌
* 购物车
* 登录
* 注册
* 商品列表
* 产品详情
* 地址列表

有广告的页面：首页、产品页、列表页、详情页

## 项目结构（src）

* Main.js
* App.vue
* Router
  + Index.js
* Components
  + mainContainer(首页)
  + Categories(分类)
  + Shopcart(购物车)
  + Login(登录注册)
  + Goodlist(商品列表)
  + Gooddetail(商品详情)

## 技术选型

* Vue、vue-cli、vue-router
* 模块化：ES6 、webpack
* CSS预处理器：stylus **npm install stylus stylus-loader --save-dev**
* 滚动
  + 横向滚动：better-scrolll
  + 竖向滑屏：vue-scroller
* UI组件：mint-ui（按需引入：npm install babel-plugin-component -D）
* Mock数据：mockjs

## Vue-router懒加载

使用懒加载形式引入路由组件

*const* shopcart = () *=>* import('../components/shopcart/shopcart.vue')

## 难点

### 更改mint-ui样式

* 更改mint-ui 的swipe 的 小圆点的样式，在使用的组件中直接修改，不生效，因为该组件并没有直接写小圆点的代码。可以将更改的样式写在公共样式组件中，在main.js中引入

### 竖向滑屏设置父元素高度

* 设置整个容器竖向flex布局，滚动区域外部容器flex: 1

### 最下方被遮挡

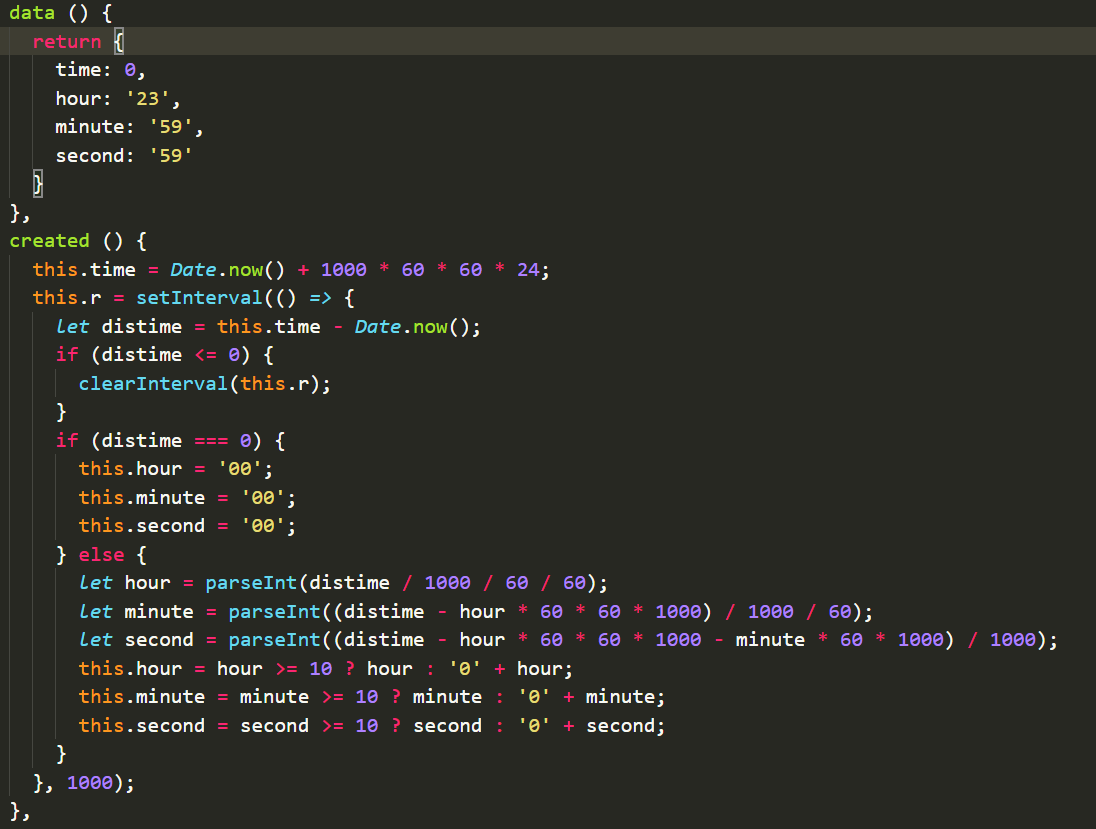
* 底部footer遮挡滚动区最下面，可以在滚动区域最下面添加一个div，设置高度为footer高度。

### 路由组件显示loading图

* 路由变化时，显示loading图：在app组件中，监视$route属性变化，变化后显示loading图。路由组件mouted后，隐藏。

### 倒计时

* 倒计时



### 横向滑屏

* 横向可以滑动：

Div

Width: 100%

Over-flow: hidden（清除浮动）

Ul

White-space: no-wrap

Float: left（使ul宽度由内容撑开）

Li

Display: inline-block

Width: 固定值

### 图片加载慢导致滚动区域高度较小

* 首页使用better-scroll竖向滑屏，在mounted时创建BScroller对象，有可能mounted时图片尚未加载完，滑屏区域高度会在图片加载完后变化，导致一部分滑不动。
  + 解决办法（不好）：在一秒之后刷新scroller对象
  + 为图片设置高度，占位（可使用图片懒加载插件）

### 小狗睁眼闭眼动画

* 右下角狗狗睁眼闭眼动画：睁眼闭眼画面为一张图，设置为div的背景，div的宽度为图片的一半，初始时background-position: 0.
  + 动画

.switchtype

*position* fixed

*width* 41px

*height* 46px

*bottom* 15%

*right* -1px

*z-index* 2

*background* url('./godog.png') no-repeat

*background-size* 80px auto

*background-position* 0 0

*animation* dog 2.5s steps(2) infinite

@keyframes dog

0%

*background-position* 0px

100%

*background-position* -81px

### 取消输入框点击时边框

* input框没有边框，得到焦点时会有一个边框。设置：outline:none

# 使用express+mongodb实现登录注册

直接在/build/dev-server.js中编写服务器代码

## 引入session

npm install express-session --save

*var* session = require('express-session')

// 配置 session

app.use(session({

resave: false,

saveUninitialized: false,

secret: 'aa'

}));

## 引入body-parser

安装

npm install body-parser --save

配置

// 引入body-parser

*var* bodyParser = require('body-parser');

// 使用 body-parser 处理 post 请求体数据

app.use(bodyParser.urlencoded({ extended: false }));

## 配置路由

在根路径下创建 router 文件夹，添加 router.js 文件，在 dev-server.js 中引入

*var* router = require('../router/router.js')

// 配置路由

app.use('/epet', router);

路由编写

*var* express = require('express')

*var* router = express.Router()

*module.exports* = router;

## 实现图片验证码

引入插件

npm install [captchapng](https://www.npmjs.com/package/captchapng) --save

在 router.js 中编写获取验证码的路由，生成验证码图片返回，并将验证码存入session

*var* Captchpng = require('captchapng')

router.get('/getCaptcha', *function* (req, res) {

*var* verifyCode = parseInt(*Math*.random() \* 9000 + 1000);

req.session.verifyCode = verifyCode;

*var* captch = new Captchpng(85, 30, verifyCode);

captch.color(255, parseInt(*Math*.random() \* 256), 0, 255);

captch.color(0, parseInt(*Math*.random() \* 256), 255, 255);

*var* img = captch.getBase64();

// var imgbase64 = new Buffer(img, 'base64');

// res.set('Content-Type', 'image/png');

res.send(img);

});

登录页面通过 axios 获取验证码图片

图片设置ref，和点击事件，点击可以更换图片

<img ref="verifyImg" src="" alt="" @click="getCaptcha">

在methods中

getCaptcha () {

axios({

method: 'get',

url: '/epet/getCaptcha',

responseType: 'stream'

})

.then(*response* *=>* {

this.$refs.verifyImg.src = 'data:image/png;base64,' + response.data;

})

}

Mounted时调用该方法

mounted () {

this.getCaptcha();

this.hideLoading();

},

验证

登录时将用户输入的验证码传到服务器，与session中存储的验证码对比

## 实现短信验证码

注册容联云通讯账号

添加测试号码

安装依赖

npm install blueimp-md5 js-base64 moment request --save

在根目录添加 util 文件夹，创建 sms\_util.js 文件

Sms\_util.js（不要直接复制老师的，用下面的，我已经改过）

*var* md5 = require('blueimp-md5')

*var* moment = require('moment')

*var* Base64 = require('js-base64').Base64;

*var* request = require('request');

/\*

生成指定长度的随机数

\*/

*function* randomCode (length) {

*var* chars = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'];

*var* result = '' // 统一改名: alt + shift + R

for (*var* i = 0; i < length; i++) {

*var* index = *Math*.ceil(*Math*.random() \* 9);

result += chars[index];

}

// console.log('验证码: ', result)

return result;

}

// console.log(randomCode(6));

*exports*.randomCode = randomCode;

/\*

向指定号码发送指定验证码

\*/

*function* sendCode (phone, code, callback) {

*var* ACCOUNT\_SID = '8aaf07085ea24877015ebc256e2a08a';

*var* AUTH\_TOKEN = '63765a1365d948a5b58292394d0e56e';

*var* RestURL = 'https://app.cloopen.com:8883';

*var* AppID = '8aaf07085ea24877015ebc256f9308a';

// 1. 准备请求url

/\*

1.使用MD5加密（账户Id + 账户授权令牌 + 时间戳）。其中账户Id和账户授权令牌根据url的验证级别对应主账户。

时间戳是当前系统时间，格式"yyyyMMddHHmmss"。时间戳有效时间为24小时，如：20140416142030

2.SigParameter参数需要大写，如不能写成sig=abcdefg而应该写成sig=ABCDEFG

\*/

*var* sigParameter = '';

*var* time = moment().format('YYYYMMDDHHmmss');

sigParameter = md5(ACCOUNT\_SID + AUTH\_TOKEN + time);

*var* url = RestURL + '/2013-12-26/Accounts/' + ACCOUNT\_SID + '/SMS/TemplateSMS?sig=' + sigParameter;

// 2. 准备请求体

*var* body = {

to: phone,

appId: AppID,

templateId: '1',

'datas': [code, '1']

}

// body = JSON.stringify(body);

// 3. 准备请求头

/\*

1.使用Base64编码（账户Id + 冒号 + 时间戳）其中账户Id根据url的验证级别对应主账户

2.冒号为英文冒号

3.时间戳是当前系统时间，格式"yyyyMMddHHmmss"，需与SigParameter中时间戳相同。

\*/

*var* authorization = ACCOUNT\_SID + ':' + time;

authorization = Base64.encode(authorization);

*var* headers = {

'Accept': 'application/json',

'Content-Type': 'application/json;charset=utf-8',

'Content-Length': *JSON*.stringify(body).length + '',

'Authorization': authorization

}

// 4. 发送请求, 并得到返回的结果, 调用callback

request({

method: 'POST',

url: url,

headers: headers,

body: body,

json: true

}, *function* (error, response, body) {

if (error) {

*console*.log(error, response, body);

}

callback(body.statusCode === '000000');

// callback(true);

});

}

*exports*.sendCode = sendCode;

在router/router.js 中添加发送短信验证码的路由

*var* SmsUtil = require('../util/sms\_util.js')

router.get('/smsValid', *function* (req, res) {

*var* code = SmsUtil.randomCode(6);

req.session.smsValidCode = code;

SmsUtil.sendCode(手机号码, code, *function* (success) {

if (success) {

res.send({

code: 1

});

} else {

res.send({

code: 0

});

}

})

});

登录页面添加方法

sendSmsValid () {

axios.get('/epet/smsValid')

.then(*response* *=>* {

*let* res = response.data;

if (res.code === 1) {

alert('发送成功');

} else {

alert('发送失败');

}

})

}

绑定事件

<a href="javascript:;" @click="sendSmsValid">获取动态密码</a>

## 使用mongoose搭建用户数据库

安装mongoose

npm install mongoose --save

在util文件夹下创建conn\_mongodb.js文件，用于连接mongodb数据库

*var* mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1:27017/epet', {

useMongoClient: true

});

mongoose.connection.once('open', *function* (err) {

if (!err) {

*console*.log('mongodb数据库已连接');

} else {

*console*.log(err);

}

});

在根路径下创建 model 文件夹，创建userModel.js 文件，用于创建 user 表

*var* mongoose = require('mongoose');

*var* UserSchema = mongoose.Schema({

username: {

type: *String*,

unique: true

},

password: {

type: *String*

},

mobile: *Number*,

email: *String*

});

*var* UserModel = mongoose.model('user', UserSchema);

*module.exports* = UserModel;

在 dev\_server.js 中引入 conn\_mongodb.js，连接数据库

require('../util/conn\_mongodb.js')

在 router.js 中引入 userModel，用于操作users数据库

*var* UserModel = require('../model/userModel.js')

# 插件

## 图片验证

gVerify：前端插件（张丽）

1.首先将 gVerify.js 加载到src 文件中；

2.在哪个组件中使用就在哪里引入；--import GVerify from ‘gVerify.js文件的路径’

3.在mounted（这时又体现了与created的区别）异步创建动态验证码对象（在this.$nextTick（（）=>{ this.verifyCode = new GVerify ( "配置对象")}））

4.在比较input框中输入的验证码是否与动态验证码一致

[captchapng](https://www.npmjs.com/package/captchapng) ：node端插件

## 难点

### 使用axios发送post请求

使用axios发送post请求时，发现通过 req.body 获取不到发送的数据，解决方式如下：

1）发送的data对象不能是传统object 类型，而是 URLSearchParams 类型

2）设置请求头

*var* params = new URLSearchParams();

params.append('aa', '灿灿');

axios.post('/epet/testPost', params, {

headers: {

'Content-Type': 'application/x-www-form-urlencoded'

}

})

.then(*response* *=>* {

*console*.log(response.data)

})

### 动态切换路由

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

# 总结--张丽

## 在后台添加路由获取首页数据

mock数据

1.自己mock一个 json 数据

json数据： ①结构--名称，数据类型（结构不能变）；②value（可以变）

2.利用 Node+express提供模拟数据

在build/dev-server.js

加载 json 数据

var data = require('../src/mock/data.json')--

var fristPageGoods = data.fristPageGoods;

--创建一个路由器

var epetRouter = express.Router()

注册路由

epetRouter.get('/fristPageGoods',function(req,res){

res.send({

code:0, //0代表正确数据

data:fristPageGoods })

})

启用路由器

app.use('/api',epetRouter)

3.ajax 请求mock数据接口

n axios是浏览器/服务器端通用的ajax请求库

npm install axios --save

4.

import axios from 'axios' ---首先引入axios

axios.get('/api/foods') /\*发送通过后台的接口发送请求\*/

.then((response)=> { /\*然后得到响应的数据（形参）\*/

const result = response.data /\*获取我们mock的整个数据\*/

if(result.code===0){ /\*状态码为0表示请求成功\*/

this.foods = result.data /\*获取到foods数组\*/

console.log(this.foods)}})

## 正则

4.正则表达式

1.test ( )方法

该方法的返回值是布尔值，通过该值可以匹配字符串中是否存在正则表达式相匹配的结果，如果有匹配的内容，返回true，否则返回false

基本语法：objReg.test(objStr)

objReg 必选项 RegExp对象名称

objStr 要进行匹配检测的字符串

2.手机号码正则表达式验证

function checkPhone(){

var phone = document.getElementById('phone').value;

if(!(/^1[34578]\d{9}$/.test(phone))){

alert("手机号码有误，请重填");

return false; } }

3.正则小知识点

{ n,m}--表示范围在n和m之间，

\w--表示任意的字母数字下划线

## 总结oninput、onchange与onpropertychange事件的用法和区别

1、onchange事件与onpropertychange事件的区别：

onchange事件在内容改变（两次内容有可能还是相等的）且失去焦点时触发；onpropertychange事件却是实时触发，即每增加或删除一个字符就会触发，通过js改变也会触发该事件，但是该事件IE专有。

2.oninput事件与onpropertychange事件的区别：

oninput事件是IE之外的大多数浏览器支持的事件，在value改变时触发，实时的，即每增加或删除一个字符就会触发，然而通过js改变value时，却不会触发；onpropertychange事件是任何属性改变都会触发的，而oninput却只在value改变时触发，oninput要通过addEventListener()来注册，onpropertychange注册方式跟一般事件一样。（此处都是指在js中动态绑定事件，以实现内容与行为分离）

3、oninput与onpropertychange失效的情况：

（1）oninput事件：a). 当脚本中改变value时，不会触发；b).从浏览器的自动下拉提示中选取时，不会触发。

（2）onpropertychange事件：当input设置为disable=true后，onpropertychange不会触发。