**VSCode**

**1、安装的插件：**

Bracket Pair Colorizer 2（选择大括号的时候，会用线指引出大括号的内容）

Beautify（ctrl+shift+f快捷键：优化格式）

Live Server（虚拟服务器）

Vue 2 Snippets

Vetur

Auto Close Tag、Auto Rename Tag（自动补齐代码）

(Vue) CSS Peek

Color Hightlight

HTML Snippets html—标签提示

HTML CSS Support –-css语法提示

Path Autocomplete –文件自动路径提示

vscode-vue-peek –查看定义，可以看到定义的源文件

Prettier - Code formatter

koroFileHeader—添加头部注释（备注抬头：ctrl+alt+i，备注插入点：ctrl+alt+t）

使用Settings Sync同步：

①Settings Sync是vscode中同步设置和安装插件的小工具  
 ②登陆Github>Your profile> settings>Developer settings>personal access tokens>generate new token，输入名称，勾选Gist，提交  
 ③保存Github Access Token  
 ④打开vscode，Ctrl+Shift+P打开命令框，输入sync，找到update/upload settings，输入Token，上传成功后会返回Gist ID，保存此Gist ID.  
 ⑤若需在其他机器上DownLoad插件的话，同样，Ctrl+Shift+P打开命令框，输入sync，找到Download settings，会跳转到Github的Token编辑界面，点Edit，regenerate token，保存新生成的token，在vscode命令框中输入此Token，回车，再输入之前的Gist ID，即可同步插件和设置。

**2、ESLint配置**（在首选项→设置→ESlint→settings.json中加入以下代码）

{

// #每次保存的时候自动格式化

"editor.formatOnSave": true,

// #每次保存的时候将代码按eslint格式进行修复

"eslint.autoFixOnSave": true,

// 添加vue...的支持

"eslint.validate": [

"javascript",

"javascriptreact",

"html",

"vue",

{

"language": "html",

"autoFix": true

},

{

"language": "vue",

"autoFix": true

},

{

"language": "typescript",

"autoFix": true

},

{

"language": "typescriptreact",

"autoFix": true

}

],

// #去掉代码结尾的分号

"prettier.semi": false,

// #使用单引号替代双引号

"prettier.singleQuote": true,

// #让函数(名)和后面的括号之间加个空格

"javascript.format.insertSpaceBeforeFunctionParenthesis": true,

//配置 fileheader头部注释

"fileheader.customMade": {

"Description": "",

"Author": "ly",

"Date": "Do not edit",

"LastEditors": "ly",

"LastEditTime": "Do not edit",

},

//此为函数注释

"fileheader.cursorMode": {

"description": "",

"param": "",

"return": "",

"author": "ly"

},

// 保存时，自动调整import语句相关顺序，能够让你的import语句按照字母顺序进行排列

"editor.codeActionsOnSave": {

"source.fixAll.eslint": true

},

//每次默认选中推荐值first(第一个)/recentlyUsedByPrefix(上次选择)

"editor.suggestSelection": "first",

"vsintellicode.modify.editor.suggestSelection": "automaticallyOverrodeDefaultValue",

// 语言的文件关联，优先级高于已安装的语言默认关联

"files.associations": {

"\*.js": "javascriptreact"

}

}

**3、VSCode统计代码行数**

打开需要统计的项目

①选择编辑->在文件中查找

②需要查找的文件使用正则查找：^b\*[^:b#/]+.\*$

③要包含的文件：./src

**4、Chrome浏览器安装vue-devtools插件**

1.打开https://github.com/vuejs/vue-devtools直接下载该项目，或者cmd方式直接输入：git Clone https://github.com/vuejs/vue-devtools.git 下载该项目

2.进入该项目安装依赖：npm install

3.npm run build

4.进入Chrome，右上菜单栏 选择更多工具>扩展程序>勾选开发者模式

5.进入Chrome，右上菜单栏 选择更多工具>扩展程序>加载已解压程序>选择文件 vue-devtools-master>shells>chrome 确定(重启没有执行直接在此文件npm run dev)

6.重启浏览器就可以了

**------------------------------------------------------------------------------------------**

**一、获取数据网址：**

随机图片地址：

"http://lorempixel.com/400/200（400/200分别为宽高）

电影列表API：

http://api.douban.com/v2/movie/in\_theaters?apikey=0df993c66c0c636e29ecbb5344252a4a&start=0&count=10

电影详情API：

<http://api.douban.com/v2/movie/subject/1088314?apikey=0df993c66c0c636e29ecbb5344252a4a>

新闻列表：

http://www.liulongbin.top:3005/api/getnewslist

商品列表：

http://www.liulongbin.top:3005/api/getprodlist

**————————————————————————————————————**

**二、Vue的安装**

①npm install -g vue-cli（已经全局安装不需要此步）

或者使用国内的淘宝镜像 npm install -g cnpm --registry=https://registry.npm.taobao.org

②vue init webpack vue-demo 搭建vue项目, “vue-demo” 是项目名称

③cd vue-demo

④npm install（安装必备包）

⑤npm run dev

**版本问题：**

npm view vue-cli versions --json（查询vue的版本号）

npm install -g [vue-cli@2.5.1（选择安装的vue版本）](mailto:vue-cli@2.5.1（选择安装的vue版本）)

vue -V（查看当前vue版本）

**报错：**

vue-style-loader!css-loader——>>>说明是css解析的时候出了问题。

所以，解决方案就要根据情况而定，看你使用的CSS语言是什么,是 常规的 或者 less 或者 sass。

如果是 常规 的，执行 npm install stylus-loader css-loader style-loader --save-dev 安装依赖就行。

如果是 less 的，执行 npm install less less-loader --save-dev 安装依赖就行。

如果是 sass 的，执行 npm install sass sass-loader --save-dev 安装依赖就行。或者（$npm intall sass-loader --save ; $npm install node-sass –save）

**目录结构如下：**

**安装组件:**

|  |
| --- |
| ①elementui：npm i element-ui -S  main.js中引入：  局部：import {Steps} from 'element-ui'  import 'element-ui/lib/theme-chalk/index.css'  Vue.component('el-steps',Steps)  全局：import ElementUI from 'element-ui';  import 'element-ui/lib/theme-chalk/index.css';  Vue.use(ElementUI)  ②mint-ui：npm install --save mint-ui  import 'mint-ui/lib/style.css'; //样式 |

**引入jQuery**

①首先安装依赖：npm install jQuery --save；

②在build文件夹下找到webpack.base.conf.js文件，添加

const webpack=require('webpack')

然后在module.exports里输入：

plugins: [

new webpack.ProvidePlugin({

$:"jquery",

jQuery:"jquery",

"windows.jQuery":"jquery"

})

],

③在入口文件main.js中输入：import $ from 'jquery'引入jQuery就可以全局使用了；

也可在单独的页面局部引用

**------------------------------------------------------------------------------------------**

**打包上线**

项目开发完成后，需要把请求数据的代理地址改成和提供数据的服务器在同一个域的地址，因为最终会把前端代码放在和数据在同一个域的服务器下面运行。

// 将下面的请求地址

'/apis/index\_data'

// 改成

'/index\_data'

改完请求地址后，就可以将代码打包，生成最终可以上线的单文件结构：

// 打开终端，ctrl+c停掉开发服务器，执行下面的命令

npm run build

**一、路径问题**

1、脚手架+webpack打包通过npm run build，但是后台tomcat部署上线的时候，会衍生出一些问题，比如，路径问题（因为在项目中，我们使用了绝对路径，这里必须要使用相对路径，但是打包后还是会报错，说是找不到assets文件夹下的各种资源，包括images,css,js,是因为，在当初在全局引用的时候写在了index.html中，安全起见，可以写在app.vue中）；

2、在打包前还要在config文件夹中的index.js中设置一个路径问题，不然也会报错(打开空白页)，在build的index.js中找到build:{assetsPublicPath:'/'}，默认路径是‘/’,需要加上'.'，写成：'./’（一般正常编辑代码是只进行本步修改即可）。

如果图片的路径是写在data中，打包后不显示，可以采用require引入

img:require("../../../../static/images/home\_1.png")

**二、favicon.ico问题**

1、favicon图标也在index.html中引入了link的那句话，这个可以在build文件夹中的webpack.dev.conf.js

中的new HtmlWebpackPlugin({ favicon:'./logo.ico' })加上这个路径，便可以找到这个图标了。

2.element图标不显示

在build中的utils.js文件中，在40行左右添加publicPath:'../../'

if (loader) {

loaders.push({

loader: loader + '-loader',

options: Object.assign({}, loaderOptions, {

sourceMap: options.sourceMap,

publicPath:'../../'

})

})

}

3.样式不显示

修改 目录 dist ==> index.html 中引用路径

所有 /static/..... 相关的修改为 ./static....…

4.背景图片不显示

修改 目录 build ==> utils.js

在下图中添加 publicPath:' ../../ '

if (options.extract) {

return ExtractTextPlugin.extract({

use: loaders,

publicPath:'../../',

fallback: 'vue-style-loader'

})

} else {

return ['vue-style-loader'].concat(loaders)

}

}

**三、打包好后可以自己测试运行是否正常问题**

方法1、这个时候需要用到node中的express，方法如下：

安装express:

npm install -g express

最新版的express4.0版本中将命令工具分家出来，还需要安装一个命令工具：

npm install -g express-generator

创建一个express工程：

express helloworld(express项目名字)

进入项目主目录：

cd helloworld

安装必备包：

npm install

启动express程序：

npm start

最后把打包好的dist内的文件(static+index.html)放在public文件夹里，访问http://localhost:3000

方法2、使用live-server第三方包。

首先全局安装live-server：

npm install -g live-server

然后在打包好的目录下运行，项目就会自动打开：

live-server –port=9527

**四、打包过后的dist文件过大**

①将config中的index.js文件build中productionSourceMap改为false

productionSourceMap: false,

②懒加载路由（需要用页面再加载）

{

path: '/clientLogin',

component: () =>import ('../components/wx/clientLogin.vue')

}

或者使用resolve

{

path: '/index',

name: 'index',

redirect: 'index/main\_page',

component: resolve => require(['../views/index/index'], resolve),

children: [

// index下的子路由

{

// 系统管理模块-菜单管理（目录只到文件夹层，文件下面都是使用index.vue）

path: 'main\_page',

name: 'main\_page',

component: resolve => require(['../views/main\_page'], resolve)

},

]

**五、目录报错**

npm run build的时候, 一开始就会提示：

Built files are meant to be served over an HTTP server. Opening index.html over file:// won't work.

因为vue-cli的默认配置中, publishPath是用绝对目录, 所以dist文件夹里的文件必须放在服务器的根目录, 如果你想本地打开的话, 可以在npm run build完成之后执行以下命令:  
npm install -g http-server // 该命令只需执行一次, 安装过之后, 以后就不需要重复安装了.

**六、生产环境下清除所有的console.log**

在build下webpack.prod.conf.js中添加：

new UglifyJsPlugin({

uglifyOptions: {

mangle: {

safari10: true

},

compress: {

warnings: false,

drop\_debugger: true, //console

drop\_console: true,

pure\_funcs: ['console.log'] //移除console

},

},

sourceMap: config.build.productionSourceMap,

cache: true,

parallel: true,

}),

**————————————————————————————————————**

**三、Vue的app项目连接手机测试**

方法一、①进入cmd命令，输入ipconfig便可看到自己的ip地址：

②在项目的config/index.js中修改host为自己电脑的ip地址：

③然后就重新运行一下vue项目，便可以通过http://ip地址:8080/来访问。例如你的ip地址为192.176.56.87，你就可以在浏览器中输入http://192.176.56.87:8080/来访问vue项目

④修改wifi的代理步骤如下：

手机连上某个wifi，然后点击“修改网络”，再选择“显示高级选项”，点击“代理”，选择“手动”，然后输入ip地址和端口，点击保存即可。

成功后可在浏览器中输入http://192.176.56.87:8080/#/login...来访问vue项目

方法二、①电脑连接手机热点；

②cmd命令，输入ipconfig找到IP地址；

③将config/index.js中修改host为自己电脑的IP地址，手机上输入相同IP后就可以访问了。

**————————————————————————————————————**

**四、项目中的问题：**

**1、禁用浏览器快捷键**

mounted() {

window.addEventListener('keydown', this.onClick); //绑定事件，可移除

},

methods: {

onClick (val) {

console.log('按下' + val.key)

console.log('keycode:' + val.keyCode)

console.log('thiscode:' + this.code)

if (val.keyCode === 123) {

val.returnValue = false

}

},

}

**1.1.vue禁用浏览器快捷键**

shortcutKeys() {

let that = this;

document.onkeydown = function (e) {

let key = e.keyCode;

if (key >= 49 && key <= 54 && e.ctrlKey) {

e.returnValue = false; //禁用

that.businessId = Math.floor(key) - 48;

}

if (key == 113) {

e.returnValue = false;

that.modal11 = true; //增加自己的快捷键

that.queryCall();

}

}

},

**2、iview中Tabs标签页刷新后，高亮回到第一项**

（value当前激活 tab 面板的 name，可以使用 v-model 双向绑定数据String默认为第一项的 name）

解决：

<Tabs :animated="false" @on-click="turnTo" :value="$route.name">

2.1element中navmenu高亮问题

①后面无子级菜单

:default-active="$route.path"

②后面有子级菜单

当activeNav与'/index/'+item.name的值相等的时候触发高亮

<el-menu :default-active="activeNav" class="el-menu-demo" mode="horizontal" active-text-color="white" router>

<el-menu-item v-for="(item,i) of menuList" :index="'/index/'+item.name" :key="i">{{item.lname}}</el-menu-item>

</el-menu>

mounted:{

heightLight(){

let href=window.location.href; //拿到这个url（[http://localhost:8080/#/index/speciaResPage/saPage](http://localhost:8080/" \l "/index/speciaResPage/saPage)）

href=href.split('/index/')[1]; //获得speciaResPage/saPage

href=href.split('/')[0]; //获得speciaResPage

this.activeNav ="/index/"+href //始终在得到(/index/speciaResPage)在index下面的高亮

},

}

**3.切换页面改变标题**

document.addEventListener("visibilitychange", function(){

document.title = document.hidden ? "用户离开了" : "用户回来了"; //页面隐藏显示"用户离开了"

});

**4.vue页面强制渲染**

this.$forceUpdate()

**5.清除浏览器右侧滚动条**

<body style="overflow:-Scroll;overflow-y:hidden">（index.html）

<div id="app"></div>

<!-- built files will be auto injected -->

</body>

<style>

::-webkit-scrollbar{

display:none;

}

</style>

**6.监听对象属性**

data() {

　　return {

　　　　bet: {

　　　　　　pokerState: 53,

　　　　　　pokerHistory: 'local'

　　　　}

}

},

watch: {

　　bet: {

　　　　handler : (newValue, oldValue)=> {

　　　　　　console.log(newValue)

　　　　},

　　　　deep: true, //确认是否深入监听，监听对象使用

//immediate:true或false //确认是否以在初始值时就执行handler的函数。

　　}

}

如果是监听父组件传入的props变化（对象也可直接写方法）

props: {

businessData: {

//商家信息

type: Object,

default: {}

}

},

watch: {

businessData() {

//确保父组件传入的数据已经加载到页面（获取页面元素时，不然会报错找不到元素）

this.$nextTick(() => {

this.$refs.selectAllTags.checked = true;

let leng = this.classifyList.conditions.length;

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

this.$refs.selectAll[i].checked = true;

}

});

}

}

**7.如果 URL 匹配不到任何静态资源，则应该返回同一个 index.html页面（可以专门做一个404页面）**

export const routes = [

{path: "/", name: "homeLink", component:Home}

{path: "/homeLink", name: "homeLink", component:Home}

children: [

//二级路由

{

path: "/homeLink",

redirect: "mainPage" //重定向，进入"/homeLink"就到mainPage页面

},

{

path: "mainPage",

name: "mainPage",

component: MainPage

}]

{path: "/register", name: "registerLink", component: Register},

{

path: '/clientLogin',

component: () =>import ('../components/wx/clientLogin.vue')

},

{path: "/login", name: "loginLink", component: Login},

{path: "\*", redirect: "/"} //404页面

]

**8.全局定义Loading和判断token**

****8.1发送请求**前判断token**

**①//全局定义Loading，在loading.js中：**

**import { Loading } from 'element-ui' //项目已经全局引入element的话可以不单独引入**

**let loading //定义loading变量**

**export function startLoading() { //使用Element loading-start 方法**

**loading = Loading.service({**

**lock: false,**

**text: '加载中……',**

**background: 'rgba(0, 0, 0, 0.7)'**

**})**

**}**

**export function endLoading() { //使用Element loading-close 方法**

**loading.close()**

**}**

**②//在htttp.js中：**

**import http from 'axios'**

**import {**

**Message**

**} from 'element-ui'**

**import router from '../../router'**

**import {**

**startLoading,**

**endLoading**

**} from './loading'**

**import VueCookies from 'vue-cookies'**

**http.defaults.timeout = 1200000 //请求发送时间--loading加载时间**

**http.defaults.withCredentials = true //发送请求携带验证信息cookies**

**http.interceptors.request.use(config => {**

**// request.credentials = true;**

**// 需要判断token使用**

**// let token = request.getCookies();**

**// if (token) {**

**// config.headers={**

**// token : token,**

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

**// }**

**// }**

**startLoading() //出现loading...**

**return config**

**},**

**error => {**

**endLoading() //关闭loading...**

**return Promise.reject(error)**

**})**

**http.interceptors.response.use(**

**resp => {**

**endLoading() //关闭loading...**

**switch (resp.data.returnCode) {**

**// 成功获取数据，那么返回数据**

**case '000000':**

**return resp**

**// 数据获取失败，且状态码为登录相关的特殊状态码，返回后端错误信息，跳转到登录页**

**case '200009':**

**Message.error({**

**message: resp.data.description**

**})**

**// 解决返回问题，清除localstorage**

**localStorage.clear()**

**//router.currentRoute.fullPath在刷新页面会清空内容，使用location**

**let href = window.location.href;**

**let newArray = href.split("#");**

**let** **path = newArray[1];**

**router.push({**

**name: 'login',**

**query: {**

**backUrl:path //登录页根据当前路径，再登陆后返回此路径**

**}**

**})**

**return Promise.reject(resp) //阻断**

**// 数据获取失败，且状态码为登录相关的特殊状态码，返回后端错误信息，跳转到登录页**

**case '200010':**

**return Promise.reject(resp)**

**case '200024': //尚未登录（只在登录页做判断）**

**return Promise.reject(resp)**

**case '999999':**

**return Promise.reject(resp)**

**// 数据获取失败，状态码为非特殊状态码，返回后端错误信息**

**default:**

**Message.error({**

**message: resp.data.description**

**})**

**return Promise.reject(resp)**

**}**

**},**

**error => {**

**console.log(error)**

**endLoading() //关闭loading...**

**if (error && error.response) {**

**switch (error.response.status) {**

**case 400:**

**error.message = '错误请求'**

**// Message.error('错误请求')**

**break**

**case 401:**

**error.message = '未授权，请重新登录'**

**Message.error('未授权，请重新登录')**

**break**

**case 403:**

**error.message = '拒绝访问'**

**Message.error('拒绝访问')**

**break**

**case 404:**

**error.message = '请求错误,未找到该资源'**

**Message.error('请求错误,未找到该资源')**

**break**

**case 405:**

**error.message = '请求方法未允许'**

**Message.error('请求方法未允许')**

**break**

**case 408:**

**error.message = '请求超时'**

**Message.error('请求超时')**

**break**

**case 500:**

**error.message = '服务器端出错'**

**Message.error('服务器端出错')**

**break**

**case 501:**

**error.message = '网络未实现'**

**Message.error('网络未实现')**

**break**

**case 502:**

**error.message = '网络错误'**

**Message.error('网络错误')**

**break**

**case 503:**

**error.message = '服务不可用'**

**Message.error('服务不可用')**

**break**

**case 504:**

**error.message = '网络超时'**

**Message.error('网络超时')**

**break**

**case 505:**

**error.message = 'http版本不支持该请求'**

**Message.error('http版本不支持该请求')**

**break**

**default:**

**error.message = '连接错误' + error.response.status**

**Message.error('连接错误' + error.response.status)**

**}**

**} else {**

**error.message = '连接到服务器失败'**

**Message.error('连接到服务器失败')**

**}**

**return Promise.reject(error.message)**

**})**

**export default http**

**③//在main.js中引入：**

**import http from '../src/assets/js/htttp'**

**Vue.prototype.$http = http**

****8.2路由中判断token****

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

**let token = localStorage.getItem('token') //判断本地是否存在token**

**if (to.fullPath.indexOf("login") == -1) {**

**if (token) {**

**next();**

**} else {**

**next({**

**path: '/login'**

**})**

**// next({name: "login",query:{backUrl:to.fullPath}}) 保存原来的路径，登陆后返回**

**}**

**} else if (to.fullPath == "/" || to.fullPath == "/login") {**

**next();**

**}**

**});**

**9.input限制输入内容**

去除红色波浪线，在input中加入属性spellcheck="false"  
<input type='text' @input="handleInput" :value="val"/>

handleInput(e){

this.val=e.target.value.replace(/[^\d]/g,''); //只能输入数字

①this.val=e.target.value.replace(/\s+/g,''); //不能输入空格

}

②<input type='text' oninput="this.value=this.value.replace(/\s+/g,'')" :value="val"/>

③<input type='text' @input="val=val.replace(/\s+/g,'')" :value="val"/>

输入大小写字母、数字、下划线：

<input type="text" onkeyup="this.value=this.value.replace(/[^\w\_]/g,'');">

输入小写字母、数字、下划线：

<input type="text" onkeyup="this.value=this.value.replace(/[^a-z0-9\_]/g,'');">

输入数字和点

<input type="text" onkeyup="value=value.replace(/[^\d.]/g,'')">

输入中文:

<input type="text" onkeyup="this.value=this.value.replace(/[^\u4e00-\u9fa5]/g,'')">

输入数字:

<input type="text" onkeyup="this.value=this.value.replace(/\D/g,'')">

输入英文:

<input type="text" onkeyup="this.value=this.value.replace(/[^a-zA-Z]/g,'')">

输入中文、数字、英文:

<input onkeyup="value=value.replace(/[^\w\u4E00-\u9FA5]/g, '')">

输入数字和字母：

<input onKeyUp="value=value.replace(/[\W]/g,'')">

除了英文的标点符号以外，其他的都可以中文，英文字母，数字，中文标点

<input type="text" onkeyup="this.value=this.value.replace(/^[^!@#$%^&\*()-=+]/g,'')">

只能输入数字代码(小数点也不能输入)

<input onkeyup="this.value=this.value.replace(/\D/g,'')" onafterpaste="this.value=this.value.replace(/\D/g,'')">

只能输入数字,能输小数点.

<input onkeyup="if(isNaN(value))execCommand('undo')" onafterpaste="if(isNaN(value))execCommand('undo')">

<input name=txt1 onchange="if(/\D/.test(this.value)){alert('只能输入数字');this.value='';}">

数字和小数点方法二

<input type=text t\_value="" o\_value="" onkeypress="if(!this.value.match(/^[\+\-]?\d\*?\.?\d\*?$/))this.value=this.t\_value;else this.t\_value=this.value;if(this.value.match(/^(?:[\+\-]?\d+(?:\.\d+)?)?$/))this.o\_value=this.value" onkeyup="if(!this.value.match(/^[\+\-]?\d\*?\.?\d\*?$/))this.value=this.t\_value;else this.t\_value=this.value;if(this.value.match(/^(?:[\+\-]?\d+(?:\.\d+)?)?$/))this.o\_value=this.value" onblur="if(!this.value.match(/^(?:[\+\-]?\d+(?:\.\d+)?|\.\d\*?)?$/))this.value=this.o\_value;else{if(this.value.match(/^\.\d+$/))this.value=0+this.value;if(this.value.match(/^\.$/))this.value=0;this.o\_value=this.value}">

只能输入字母和汉字

<input onkeyup="value=value.replace(/[\d]/g,'') "onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[\d]/g,''))" maxlength=10 name="Numbers">

只能输入英文字母和数字,不能输入中文

<input onkeyup="value=value.replace(/[^\w\.\/]/ig,'')">

只能输入数字和英文

<input onKeyUp="value=value.replace(/[^\d|chun]/g,'')">

小数点后只能有最多两位(数字,中文都可输入),不能输入字母和运算符号:

<input onKeyPress="if((event.keyCode<48 || event.keyCode>57) && event.keyCode!=46 || /\.\d\d$/.test(value))event.returnValue=false">

小数点后只能有最多两位(数字,字母,中文都可输入),可以输入运算符号:

<input onkeyup="this.value=this.value.replace(/^(\-)\*(\d+)\.(\d\d).\*$/,'$1$2.$3')">

**9.1.input回车事件**

<input type="text" v-model="username" @keyup.enter="searchIfo">

**iview组件中input绑定禁止输入空格事件**

<Input v-model="newUser.userName" placeholder="3~16位字符" :maxlength=16 @keyup.native="newUser.userName=newUser.userName.replace(/\s+/g,'')"></Input>

**9.2.input在一行中，自动填充满整行剩余部分**

|  |
| --- |
| <div class="mainpart-item">  <span>\*</span>  <span>证件姓名</span>  <p>  <input type="text">  </p>  </div> |
| span{  float: left;  }  p{  overflow:hidden; //利用这个属性  input{  width: 100%;  display: inline-block;  box-sizing: border-box;  text-align: right;  }  } |

**9.3.input、button边框和文字渐变颜色**

|  |
| --- |
| button:first-child{  margin-right: 96px;  //边框渐变颜色  border:2px solid transparent;  background-clip:padding-box,border-box;  background-origin:padding-box,border-box;  background-image:linear-gradient(90deg,#fff,#fff),linear-gradient(90deg,#fe9477,#d12ffe);  span{ //文字渐变颜色  background-image:linear-gradient(90deg,#fe9477,#d12ffe);  -webkit-background-clip:text; //非正式属性，目前仅仅FF、Chrome、Safari支持，并且必须带前缀。  -webkit-text-fill-color:transparent;  }  } |

**9.4.input正则验证**

①邮箱验证：

reg=/^[a-zA-Z0-9\_.-]+@[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)\*\.[a-zA-Z0-9]{2,6}$/;

@之前必须有内容且只能是字母（大小写）、数字、下划线(\_)、减号（-）、点（.）；

@和最后一个.之间必须有内容且只能是字母（大小写）、数字、点（.）、减号（-），且两个点不能挨着；

最后一个点(.)之后必须有内容且内容只能是字母（大小写）、数字且长度为大于等于2个字节，小于等于6个字节。

②正则表达式，只含有汉字、数字、字母、下划线不能以下划线开头和结尾：

reg=/^(?!\_)(?!.\*?\_$)[\u4e00-\u9fa5a-zA-Z0-9\_]+$/

  其中：^  与字符串开始的地方匹配

(?!\_)　　不能以\_开头

(?!.\*?\_$)　　不能以\_结尾

[a-zA-Z0-9\_/u4e00-/u9fa5]+　　至少一个汉字、数字、字母、下划线

$　　与字符串结束的地方匹配

③长度8位~20位，必须包含数字、字母和字符

reg=/^(?![^a-zA-Z]+$)(?!\D+$)(?![a-zA-Z0-9]+$).{8,20}$/;

**④屏蔽敏感词汇： ([我卧]|wo)\s\*([草艹槽操]|cao)**

**手机号： ((\+86|0086)\s+)?1[3-9]\d{9}**

**身份证号： \d{15}(\d\d[0-9x])?**

**开头和结尾的空格：^\s+|\s+$**

**10.每次请求加随机数，不然请求地址一样，IE会使用缓存**

**this.$http.get(\_global.serverUrl + "/evaluation/queryEvaluationScoreById/" + evaluationId+"?t="+new Date().getTime())**

**11.浏览器调用缓存问题**

**keep-alive内置组件已经封装了两个属性，include和exclude表示那些组件需要缓存那些组件和不需要缓存的组件，用法大致如下:**

**①<keep-alive include="test-keep-alive">**

**<!-- 将缓存name为test-keep-alive的组件 -->**

**<component></component>**

**</keep-alive>**

**②<keep-alive include="a,b">**

**<!-- 将缓存name为a或者b的组件，结合动态组件使用 -->**

**<component :is="view"></component>**

**</keep-alive>**

**③<!-- 使用正则表达式，需使用v-bind -->**

**<keep-alive :include="/a|b/">**

**<component :is="view"></component>**

**</keep-alive>**

**④<!-- 动态判断 -->**

**<keep-alive :include="includedComponents">**

**<router-view></router-view>**

**</keep-alive>**

**⑤<keep-alive exclude="test-keep-alive">**

**<!-- 将不缓存name为test-keep-alive的组件 -->**

**<component></component>**

**</keep-alive>**

**activated,deactivated这两个生命周期函数一定是要在使用了keep-alive组件后才会有的，否则不存在：**

**activated(){**

**console.log('activated每次页面打开时触发');**

**},**

**deactivated(){**

**console.log('deactivated页面关闭时触发');**

**}**

**12.引入样式**

**<style lang="less" scoped>**

**@import url('../../assets/css/index.less');**

**</style>**

**// 注册全局组件（main.js中）**

**Vue.component(name = 'myPaging2', myPaging2)**

**13.全屏显示**

|  |
| --- |
| 1、项目中使用的是sreenfull插件，执行命令安装  npm install --save screenfull  2、第一步完成之后就是怎么监听到全屏的变化，因为如果是通过Esc键退出全屏，此时是没办法监听到的  解决办法如下：  <script>  import screenfull from 'screenfull'  export default {  data () {  return {  isFullscreen: false  }  }，  methods: {  \* 全屏事件  screenfull() {  if (!screenfull.enabled) {  this.$message({  message: 'Your browser does not work',  type: 'warning'  })  return false  }  screenfull.toggle()  this.isFullscreen = true  },  //是否全屏并按键ESC键的方法  checkFull() {  var isFull = document.fullscreenEnabled || window.fullScreen || document.webkitIsFullScreen || document.msFullscreenEnabled  // to fix : false || undefined == undefined  if (isFull === undefined) {  isFull = false  }  return isFull  }  },  mounted() {  window.onresize = () => {  // 全屏下监控是否按键了ESC  if (!this.checkFull()) {  // 全屏下按键esc后要执行的动作  this.isFullscreen = false  }  }  }  }  </script>  其实主要是增加了对Esc键一个监听，然后进行相应处理，这样一个全屏切换就完成啦 |
| //打开全屏新窗口  window.open (window.origin+'/#/index','','fullscreen=3')  //关闭窗口  window.close()  <https://www.runoob.com/jsref/met-win-open.html> |

**14.视口占满整个屏幕**

**.container{**

**min-width: 1200px;**

**max-width: 1920px;**

**.inner-container{**

**position: absolute;**

**top: 80px;**

**bottom: 0px;**

**width: 100%;**

**min-width: 1200px**

**}**

**}**

****14.1弹窗(可做loading效果)****

.loadding::before {

content: "";

position: fixed;

z-index: 2000;

top: 0;

left: 0;

width: 100%;

height: 100%;

background-color: #fff;

opacity: 0.9;

}

.loadding::after {

content: "加载中..."; //更改提示内容

position: fixed;

z-index: 2001;

display: inline-block;

font-size: 1.5rem;

text-align: center;

top: 40%;

left: 20%;

right: 20%;

color: #fe9477;

text-rendering: auto;

-webkit-font-smoothing: antialiased;

-webkit-animation: fa-spin 2s infinite linear;

animation: fa-spin 2s infinite linear;

}

**14.2.防止模态框穿透（可点击下方元素）**

scrollStop(){

this.scrollTop=document.scrollingElement.scrollTop; //记录滚动条位置

let m = function(e){

e.preventDefault()

}

document.documentElement.style.position = "fixed"; //模态框是fixed状态

document.body.style.overflow = "hidden"; //隐藏滚动条

},

scrollStart(){

let m = function(e){

e.preventDefault()

}

document.documentElement.style.position = "static";

document.body.style.overflow = ""; //出现滚动条

document.scrollingElement.scrollTop = this.scrollTop; //还原滚动条位置

**}**

**15.动态绑定属性(class)**

**（1）绑定class类**

**①多个属性选择**

**:class="mycount == 1 ? 'containerMon111' : 'containerMon1'"**

**:class="{'hidden':isHidden,'bigger':isBigger}"**

**②单个类样式**

**:class="{'containerMon111' : mycount == 1}"**

③class 的表达式过长或逻辑复杂时，我们可以通过计算属性来绑定。

<div :class="customClasses"></div>

data: {

isHidden: false,

isBigger: true

},

computed: {

customClasses: function () {

return {

display: !this.isHidden,

'bigger-text': !this.isHidden && this.isBigger

}

}

}注意：如果样式名称带有 **-**，那么必须加上单引号（示例中的 bigger-text）才能被正确识别。

（2）绑定style样式

<div :style="border">

data: {

border:{

border:'1px solid #00F',

textShadow:'0 0 .3em gray'

}

}

如果需要计算的使用calc：this.dialogWidth="calc(50% - " + width/2 + "px)";

注意：因为 JS 属性不支持短横分隔命名，所以我们这里使用 CSS 也支持的驼峰命名法。

**16.状态共享（应用不大）**

**随着组件的细化，就会遇到多组件状态共享的情况， Vuex当然可以解决这类问题，不过就像 Vuex官方文档所说的，如果应用不够大，为避免代码繁琐冗余，最好不要使用它，今天我们介绍的是 vue.js 2.6 新增加的 Observable API ，通过使用这个 api 我们可以应对一些简单的跨组件数据状态共享的情况。**

**如下这个例子，我们将在组件外创建一个 store，然后在 App.vue组件里面使用 store.js 提供的 store和 mutation方法，同理其它组件也可以这样使用，从而实现多个组件共享数据状态。**

**首先创建一个 store.js，包含一个 store和一个 mutations，分别用来指向数据和处理方法。**

**import Vue from "vue";**

**export const store = Vue.observable({ count: 0 });**

**export const mutations = {**

**setCount(count) {**

**store.count = count;**

**}**

**};**

**然后在 App.vue里面引入这个 store.js，在组件里面使用引入的数据和方法**

**<template>**

**<div id="app">**

**<img width="25%" src="./assets/logo.png">**

**<p>count:{{count}}</p>**

**<button @click="setCount(count+1)">+1</button>**

**<button @click="setCount(count-1)">-1</button>**

**</div>**

**</template>**

**<script>**

**import { store, mutations } from "./store";**

**export default {**

**name: "App",**

**computed: {**

**count() {**

**return store.count;**

**}**

**}**

**methods: {**

**setCount: mutations.setCount**

**}**

**};**

**17.作用域插槽（子->父数据）**

**首先在子组件child里面，将$index作为一个 <slot>元素的特性绑定上去：**

**<div v-for="(item, index) in items">**

**<slot name="cont" :$index="index"></slot> //父组件template中slot=first的内容放在此插槽内**

**<slot name="second"></slot> //具名插槽，父组件template中slot=second的内容放在此插槽内**

**</div>**

**props:{**

**items: Array //接收父组件的参数**

**}**

**//props:["items"] //用数组写法接收父组件参数**

**①父组件引用的时候，给 v-slot带一个值来定义我们提供的插槽 prop 的名字：**

**<child>**

**<template v-slot:default="slotProps">**

**{{ users[slotProps.$index].id }}**

**</template>**

**</childr>**

**独占默认插槽的缩写语法：**

**<child v-slot="slotProps">**

**{{ users[slotProps.$index].name }}**

**</child>**

**②另一种写法，用scope：**

**父组件：**

**<child :items="users">**

**<template slot="first" scope="props">**

**{{users[props.$index].id}} //拿到子组件$index**

**{{users[props.$index].name}}**

**{{users[props.$index].age}}**

**</template>**

**<template slot="second">**

**<div>这是第二个插槽</div>**

**</template>**

**</child>**

**data () {**

**return {**

**users: [**

**{id: 1, name: '张三', age: 20},**

**{id: 2, name: '李四', age: 22}**

**]**

**}**

**}**

**18.去除多余的样式**

**随着项目越来越大，书写的不注意，不自然的就会产生一些多余的 css，小项目还好，一旦项目大了以后，多余的 css 会越来越多，导致包越来越大，从而影响项目运行性能，所以有必要在正式环境去除掉这些多余的css，这里推荐一个库purgecss，支持 CLI、JavascriptApi、Webpack 等多种方式使用，通过这个库，我们可以很容易的去除掉多余的 css。**

**<h1>Hello Vanilla!</h1>**

**<div>**

**We use Parcel to bundle this sandbox, you can find more info about Parcel**

**<a href="https://parceljs.org" target="\_blank" rel="noopener noreferrer">here</a>.**

**</div>**

**body {**

**font-family: sans-serif;**

**}**

**a {**

**color: red;**

**}**

**ul {**

**li {**

**list-style: none;**

**}**

**}**

**import Purgecss from "purgecss";**

**const purgecss = new Purgecss({**

**content: ["\*\*/\*.html"],**

**css: ["\*\*/\*.css"]**

**});**

**const purgecssResult = purgecss.purge();**

**最终产生的 purgecssResult结果如下，可以看到多余的 a和 ul标签的样式都没了**

**19.axios跨域问题**

**方式1：服务器端设置跨域**

**header(“Access-Control-Allow-Origin:\*”);**

**header(“Access-Control-Allow-Headers:content-type”);**

**header(“Access-Control-Request-Method:GET,POST”);**

**方式2：在config下index.js中设置一个代理服务器，使用proxyTable 我比较推荐这种方法。**

**首先在config/index.js 里面找到proxyTable ：{} ，然后在里面加入**

**'/api': {**

**target: 'http://71.158.4.120:8080/', //设置你调用的接口域名和端口号，别忘了加http**

**changeOrigin: true,**

**secure: false,**

**headers: { //访问地址为https加入此项**

**Referer: 'https://192.169.100.151/api/'**

**},**

**pathRewrite: {**

**'^/api': '' //这里理解成用‘/api’代替target里面的地址，后面组件中我们掉接口时直接用api代替，比如我要调用'http://40.00.100.100:3002/user/add'，直接写‘/api/user/add’即可**

**}**

**}**

**重启项目(每次改这个地方都要重启)**

**为了方便打包后去除'/api',建议把'/api'设成全局，在main.js中添加**

**Vue.prototype.api = process.env.NODE\_ENV === 'production' ? '' : '/api'**

**调用接口的时候的url就可以写成api + '接口地址'**

**方式3:使用jsonp方式。**

**20.屏幕适配**

**//- 设置html标签font-size**

**在main.js中引入即可import fontSize from "./assets/js/fontSize"**

**(function (doc, win) {**

**var \_root = doc.documentElement,**

**resizeEvent = 'orientationchange' in window ? 'orientationchange' : 'resize',**

**resizeCallback = function () {**

**var clientWidth = \_root.clientWidth,fontSize = 20;**

**if (!clientWidth) return;**

**if(clientWidth>=1200){**

**fontSize = 100 \* (clientWidth / 1920);**

**}else{**

**fontSize = 100 \* (1200 / 1920);**

**}**

**\_root.style.fontSize = fontSize + 'px';**

**};**

**if (!doc.addEventListener) return;**

**win.addEventListener(resizeEvent, resizeCallback, false);**

**doc.addEventListener('DOMContentLoaded', resizeCallback, false);**

**})(document, window);**

****20.1采用淘宝的lib-flexible的库,来做屏幕自适应处理（vue-PC端）****

**①在build中until.js中的配置同移动端，更改**

remUnit: 192, //根据视觉稿，rem为px的十分之一，1920px 192 rem

②在main.js中引入flexble.js 文件：

import './assets/js/flexble.js'

③修改 flexble.js文件中适配手机的设置

function refreshRem() {

var width = docEl.getBoundingClientRect().width;

if (width / dpr > 540) {

width = width \* dpr;

}

var rem = width / 10;

docEl.style.fontSize = rem + 'px';

flexible.rem = win.rem = rem;

}

（\*\*\*重启项目单位px会转为rem，字体建议使用px）

****20.2采用淘宝的lib-flexible的库,来做屏幕自适应处理（vue移动端）****

**①安装lib-flexible：npm i lib-flexible -S（npm安装不上去的,可以切换到cnpm上来安装）**

**②在主文件main.js里面引入：import 'lib-flexible'**

**③在index.html文件中设置meta标签：<meta name='viewport' content='width=device-width , initial-scale=1.0'>**

**④项目中写css时会自动将rem转化为px,需要安装px2rem这个工具：**

**npm install px2rem-loader --save-dev**

**⑤在build中until.js配置文件对loader做相关配置:**

**const cssLoader = {**

**loader: 'css-loader',**

**options: {**

**sourceMap: options.sourceMap,**

**importLoaders:6 //importLoaders的数量默认是0，意思是一个加载器都不用，1的意思是用postcss-loader加载器，2的意思是用postcss-loaders和sass-loader加载器，不行就加大数字**

**}**

**}**

**新增以下：**

**const px2remLoader = {**

**loader : 'px2rem-loader',**

**options : {**

**//这个参数是通过psd设计稿的 宽度 / 10 来计算,这里以750为标准**

**remUnit : 75,**

**// remPrecision: 8//换算的rem保留几位小数点**

**}**

**}**

**然后将utils.js文件中的generateLoaders函数的loaders常量改为[cssLoader, px2remLoader],如下:**

**function generateLoaders (loader, loaderOptions) {**

**const loaders = options.usePostCSS ? [cssLoader, px2remLoader, postcssLoader] : [cssLoader, px2remLoader]**

**...}**

****21.vue elementui走马灯高度自适应（轮播图）****

**<div class="slide">**

**<el-carousel indicator-position="none" :height="imgHeight+'px'">**

**<el-carousel-item v-for="(item,i) of imgArray" :key="i">**

**<img ref="imgHeight" :src="item.img" @load="imgLoad">**

**</el-carousel-item>**

**</el-carousel>**

**</div>**

methods: {

// 加载获取轮播图的图片高度并赋值给轮播最外层元素

imgLoad() {

this.$nextTick(() => {

this.imgHeight = this.$refs.imgHeight[0].height;

});

}

},

mounted() {

this.imgLoad();

// 屏幕变化时改变轮播图高度

window.addEventListener("resize", this.imgLoad);

},

beforeDestroy() {

window.removeEventListener("resize", this.imgLoad);

}

**22.父组件主动获取子组件的数据和方法**  
①调用子组件的时候 定义一个ref  
 <headerchild ref="headerChild"></headerchild>  
②在父组件里面通过  
 this.$refs.headerChild.属性 //另一种写法this.$refs.headerChild (“事件名m”,”参数e”)  
 this.$refs.headerChild.方法  
**22.1子组件主动获取父组件的数据和方法**  
在子组件里面通过：  
 this.$parent.属性  
 this.$parent.方法

**22.2以允许一个祖先组件向其所有子孙后代注入一个依赖**（不论组件层次有多深，并在起上下游关系成立的时间里始终生效）

provide：是一个对象，或者是一个返回对象的函数。包含要给子孙后代的东西，也就是属性和属性值。

inject：一个字符串数组或者对象，属性值可以是一个对象，包含from和default默认值

const Child = {

inject: { //对象

foo: {

from: 'bar',

default: 'foo'

}

}

}

from表示在可用的注入内容中搜索用的 key，default当然就是默认值。

|  |
| --- |
| 祖先组件：（注入刷新页面的方法）  <template>  <div id="app">  <router-view v-if="isRouterAlive"></router-view>  </div>  </template>  <script>  export default {  provide(){  return{  alive: this.isRouterAlive,  reload: this.reload  }  },  data(){  return{  isRouterAlive: true  }  },  methods:{  reload(){  this.isRouterAlive = false;  this.$nextTick(()=>{  this.isRouterAlive = true;  })  }  }  } |
| 后代组件：  export default {  inject:['alive', 'reload'], //数组  methods:{  ...=this.reload(); //想要刷新页面的时候就调用此方法  }  } |

这样也就可以访问了，当做当前vue实例的属性。这样做的好处，相当于给了一个捷径，不用使用$parent一级一级的访问。

我们可以把依赖注入看做一部分“大范围有效的prop”。

**\*\*\*父组件传递到子组件的参数props，如果是对象，只能用一层属性，比如：props.info，当用到第二次属性会报错，比如：props.info.name（报错：name不错在）——解决：前面加一个v-if判断 props.info**

**如果是本身要在v-if判断第二层 props.info.name，可以在watch中监控传入的参数，当他为true(变化)的时候重新赋值**

**23.子组件传递参数到父组件中（子->父）**

父组件中：

<router-view @handeEvent="getParams">

getParams(e){

console.log(“接收到子组件的参数”+e)

}

子组件中：

onChild(){

//子组件此事件触发会激发父组件中的handeEvent事件，从而触发事件getParams获得参数

this.$emit("handeEvent","参数e")

}

**23.1组件之间路由跳转传参**

①通过params传参

如果push后面是用{ path:"/details/:lid",component:Details,props:true }跳转，先在路由path里那个组件需要传递参数，定义一个参数，用于组件传递，params刷新页面数据会丢失。

如果定义了props:true，lid参数会自动传递给props中同名的属性: props:["lid"]

this.lid=5

然后需要用的组件接受，通过beforeRouteEnter进入路由之前执行的函数

如果push后面用name:"chatView"来跳转，直接(不用在路由后面加参数)：

created(){

this.user=this.$route.params.lid

}

②通过query传参

刷新页面query不会消失，两者的区别就在于query会把传递的参数显示在url地址中

JSON.stringify()方法是将一个JavaScript值(对象或者数组)转换为一个 JSON字符串(如果不转换，传递的参数是对象/数组，刷新以后只得到[Object Object]，不能得到其中值)然后需要用的组件接受，通过beforeRouteEnter进入路由之前执行的函数或者在created(){

this.user=JSON.parse( this.$route.query.user)

}

**23.2非父子组件之间事件触发/传参**

假设 bb 组件里面有个按钮，点击按钮，把 123 传递给 aa 组件（如果方法没有实现，切换一下页面，再点击触发事件）

①// 根组件（this.$root)

new Vue({

el: '#app',

router,

render: h => h(App),

data: {

// 空的实例放到根组件下，所有的子组件都能调用

Bus: new Vue()

}

})

bb 组件内调用事件触发↓

<button @click="submit">提交</button>

methods: {

submit() {

// 事件名字自定义，用不同的名字区别事件

this.$root.Bus.$emit('eventName', 123)

}

}

aa 组件内调用事件接收↓

// 当前实例创建完成就监听这个事件

created(){

this.$root.Bus.$on('eventName', value => {

this.print(value)

})

},

methods: {

print(value) {

console.log(value)

}

},

// 在组件销毁时别忘了解除事件绑定

beforeDestroy() {

this.$root.Bus.$off('eventName')

},

问题一：如果有多个组件组件需要通信，是不是要在根组件上多建几个 Bus？

答：不需要的，只要保证事件名 （eventName）不一样就行了。

②公共实例文件bus.js，作为公共数控中央总线

import Vue from "vue";

export default new Vue();

bb 组件内调用事件触发↓

import bus from "../../bus"

<button @click="submit">提交</button>

methods : {

submit() {

// 事件名字自定义，用不同的名字区别事件

bus.$emit('eventName', 123)

}

}

aa 组件内调用事件接收↓

import bus from "../../bus"

// 当前实例创建完成就监听这个事件

created(){

bus.$on('eventName', value => {

this.print(value)

})

},

methods: {

print(value) {

console.log(value)

}

},

// 在组件销毁时别忘了解除事件绑定

beforeDestroy() {

bus.$off('eventName')

},

③在main.js中将bus添加到vue属性中

Vue.prototype.bus = new Vue();

调用bus前面加this（this.bus.$emit()**/**this.bus.$on()）

**24.回到顶部(设置定时器)**

data(){

return {

timer:null

}

}

methods:{

jump() {

this.timer=setTimeout(()=>{

document.documentElement.scrollTop = document.body.scrollTop = 0;

},200)

}

},

beforeDestroy() {

clearTimeout(this.timer)

this.timer=null

}

**25.下拉再加载更多信息**

<https://www.jianshu.com/p/257a39625c10>

页面常见高度、宽度相关计算

网页可见区域宽： document.body.clientWidth

网页可见区域高： document.body.clientHeight

网页可见区域宽： document.body.offsetWidth (包括边线的宽)

网页可见区域高： document.body.offsetHeight (包括边线的高)

网页正文全文宽： document.body.scrollWidth

网页正文全文高： document.body.scrollHeight

网页被卷去的高： document.body.scrollTop （当前滚动条已移动的高）

网页被卷去的左： document.body.scrollLeft

网页正文部分上： window.screenTop

网页正文部分左： window.screenLeft

屏幕分辨率的高： window.screen.height

屏幕分辨率的宽： window.screen.width

屏幕可用工作区高度： window.screen.availHeight

屏幕可用工作区宽度： window.screen.availWidth

**25.1.图片懒加载**

1.安装插件：npm install vue-lazyload --save-dev

2.在入口文件main.js中引入并使用：import VueLazyload from 'vue-lazyload'

3.直接使用：Vue.use(VueLazyload)

或者添加自定义选项

Vue.use(VueLazyload, {

preLoad: 1.3,

error: 'dist/error.png',

loading: 'dist/loading.gif',

attempt: 1

})

4.修改图片显示方式为懒加载（将 :src 属性直接改为v-lazy，图片的路径定义在data中require引入imgUrl:require("../../../assets/logo.png")）

<a href="javascript:;"><img v-lazy=" item.productImage"></a>

注意：

src\*\*同级目录\*\*中有static文件夹

src\*\*子文件夹\*\*中有assets文件夹

在使用vue-lazyload，设置error或loading属性的图片路径时，

图片在assets文件夹，就需要使用require（）进行引入。

Vue.use(VueLazyload, {

preLoad: 1.3,

loading: require('./assets/123.gif'),

attempt: 1

})

图片在static文件夹，就可以直接写路径了

Vue.use(VueLazyload, {

preLoad: 1.3,

loading: '../static/123.gif',

attempt: 1

})

vue-lazyload是在main.js文件中引入，不会被webpack进行编译，src中的文件会被webpack编译，包括assets，assets文件夹中的图片地址，会在编译过程中改变。因此vue-lazyload无法正确获得图片地址，就不能显示图片了

**26.监听组件的生命周期**

比如有父组件 Parent和子组件 Child，如果父组件监听到子组件挂载 mounted就做一些逻辑处理，常规的写法可能如下：

// Parent.vue

<Child @mounted="doSomething"/>

// Child.vue

mounted() {

this .$emit("mounted");

}

这里提供一种特别简单的方式，子组件不需要任何处理，只需要在父组件引用的时候通过 @hook来监听即可，代码重写如下：

<Child @hook:mounted="doSomething"/>

当然这里不仅仅是可以监听 mounted，其它的生命周期事件，例如： created， updated等都可以

**27.IE浏览器url的编码方式不同(汉字模糊查询不出)**

更改编码方式：

let url=\_global.searchUrl + "/search/ESKnowledge?pageNumber=" + this.currentPage + "&pageSize=" + this.pageSize + "&searchContent=" + this.Keyword + "&wordType=" + this.wordType;

this.$http.get(encodeURI(url)).then(data => {...}

**28.table组件**

**28.1iview的table**

<Table stripe :columns="columns" :data="data1" highlight-row @on-current-change="currentChange" :row-class-name="tableRowClassName"></Table>

method:{

//点击修改样式

tableRowClassName (row,index) {

if (index === this.index) {

return 'demo-table-info-row'; // 样式名

} else{

return '';

}

}

<style>

.ivu-table {

font-size: 1.1rem !important

}

.ivu-table-row-highlight td {

background-color: rgba(0,0,0,0) !important; //默认选中的样式

}

.ivu-table .demo-table-info-row td{

background-color: lightpink !important;

}

</style>

**28.2elementui的table用header-row-style设置多个属性**

方式1：

直接在标签上添加上属性值：

<el-table :header-cell-style="{background:'#F3F4F7',color:'#555'}"></el-table>y

也可写成一个对象变量

styleObj:{

background:"#f00"

},

方式2：

然后在el-table标签中使用方法：

<el-table :header-cell-style="rowClass"></el-table>

在method里面写上方法：

rowClass({ row, rowIndex}) {

console.log(rowIndex) //表头行下标

return 'background:#F3F4F7;color:#555'

}

**29.监控路由变化时，实现刷新等**

watch:{

//① "$route":["querySeatingTeam"] //第一种写法：直接加事件名

②'$route' (to, from) {

//to：进入的页面

//from：来自的页面

this.querySeatingTeam(); //在created函数执行的方法，放到该处

}

}

**30.Vuex的使用**

①安装vuex：npm install vuex –save-dev

②在vue项目目录下建立store文件夹，并在store中，建立以下目录文件：

③需要在项目main.js文件中引入store

import store from './store/index'

省略代码.....

new Vue({

el: '#app',

router,

store,

components: { App },

template: '<App/>'

})

|  |
| --- |
| index.js中：  import Vue from 'vue'  import Vuex from 'vuex'  import logIn from './modules/logIn'  import state from './state'  import actions from './actions'  import mutations from './mutations'  Vue.use(Vuex);  const store = new Vuex.Store({  modules: {  logIn  },  state,  actions, //异步  mutations, //同步  })  export default store; |
| state.js中：  const count = 0  const answer="正确"  export default {  count  } |
| mutations.js中：  const CALLING = function (state, sum) {  state.count+=sum //state中的变量count加传入的sum值  }  export default {  CALLING  } |
| actions.js中：  //自定义触发mutations里函数的方法，context与store 实例具有相同方法和属性  // const outCalling = function (context) {  // context.commit('CALLING')  // }  //另一种写法：解构  const outCalling = function ({ commit }, newSum) {  return new Promise((resolve, reject) => {  setTimeout(() => { //自己写的一个异步函数  commit('CALLING',newSum)  resolve()  }, 1000)  })  }  export default {  outCalling  } |
| 组件中使用：  ①import { mapState, mapGetters, mapMutations, mapActions } from 'vuex'  //映射vuex中的变量  computed: {  mapState([ 'count', 'answer']),  //如果你想将一个 getters 属性另取一个名字，使用对象形式：(其他的一样)  mapGetters({  //把 `this.doneCount` 映射为 `this.$store.getters.doneTodosCount`  doneCount: 'doneTodosCount'  })  },  methods: {  //映射vuex里面的方法  ...mapActions([ 'outCalling', 'releaseCall' ]),  ...mapMutations([ 'CALLING' ]),  myClick(){  this.outCalling(10) //直接当作方法使用，点击一次加10  }  },  created(){  this. CALLING(5)  }  ②this.$store.state.count; //获得变量  this.$store.commit('CALLING') //同步调用(mutation中的方法)  this.$store.dispatch('outCalling'，10) //异步调用(actions中的方法)，10要变化的实参 |

**31.async/await组合**

async function timeout(flag) {

if (flag) {

return 'hello world'

} else {

throw 'my god, failure'

}

}

console.log(timeout(true)) // 调用Promise.resolve() 返回promise 对象。

console.log(timeout(false)); // 调用Promise.reject() 返回promise 对象。

timeout(true/false).then(result=>{

console.log(result)

}).catch(err => {

console.log(err)

})

**异步函数同步执行**

// 2s 之后返回双倍的值

function doubleAfter2seconds(num) {

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

setTimeout(() => {

resolve(2 \* num)

}, 2000);

} )

}

async function testResult() {

let first = await doubleAfter2seconds(30);

let second = await doubleAfter2seconds(50);

console.log(first + second); //160

await doubleAfter2seconds(30).then(){

console.log(third); //60

};

}

testResult(); //调用方法，它里面遇到了await, 表示等一下，代码就暂停到这里，不再向下执行了，它等什么呢？等后面的promise对象执行完毕，然后拿到promise的 resolve 的值并进行返回，返回值拿到之后，它继续向下执行。

**32.自定义指令**

①import Vue from 'vue'

import http from "../assets/js/htttp"

export default (Vue) => {

Vue.directive('play', { //名字，使用直接v-play

inserted(el, binding) {

el.addEventListener("play", () => { //绑定播放事件

let params = {

caseId: binding.value

}

let api = process.env.NODE\_ENV === 'production' ? '/api' : '/api'

http.post(api + '/case/addPlayCount', params).then(res => {

console.log(res)

})

})

}

})

}

【\* el：指令所绑定的元素，可以用来直接操作 DOM 。

\* binding：一个对象，包含以下属性：

\* name：指令名，不包括 v- 前缀。

\* value：指令的绑定值，例如：v-my-directive="1 + 1" 中，绑定值为 2。

\* oldValue：指令绑定的前一个值，仅在 update 和 componentUpdated 钩子中可用。无论值是否改变都可用。

\* expression：字符串形式的指令表达式。例如 v-my-directive="1 + 1"中，表达式为 "1 + 1"。

\* arg：传给指令的参数，可选。例如 v-my-directive:foo 中，参数为 "foo"。

\* modifiers：一个包含修饰符的对象。例如：v-my-directive.foo.bar 中，修饰符对象为 { foo: true, bar: true }。

\* vnode：Vue 编译生成的虚拟节点。移步 VNode API 来了解更多详情。

\* oldVnode：上一个虚拟节点，仅在 update 和 componentUpdated 钩子中可用。】

②在main.js中配置

import playCount from './directive/playCount'

Vue.use(playCount)

③使用的时候

<video v-play="videoInfo.id" ref="videoPlayer" controls></video>

点击播放视频时，把当前的视频id传入进去

**33.文件上传和下载**

**33.1上传**

<input id="upload-ipt" type="file" style="display:none;">

<div class="upload\_content">

<p class="attachment\_name">

<span class="name" id="file-name"></span>

<img class="del\_img" src="./lib/images/评审/del.png">

</p>

</div>

<button id="upload-btn">点击上传</button>

<span class="introduce">支持PNG、JGP、Word、Excel、PDF、RAR、PPTX等格式，文件大小不超过20M</span>

// 点击上传按钮

$("#upload-btn").on("click", function () {

$("#upload-ipt").click();

});

// 触发input

let file=""

$("#upload-ipt").on("change",function(e){

// var formData = new FormData(),

fs = e.target.files;

let max\_size = 1024 \* 1024 \* 20

var d = fs[0]

if(d.size > max\_size){ //文件必须小于100M

webToast("上传文件不能大于20M！","top",2000);

return

}else{

// 显示名字

$("#file-name").text(fs[0].name);

file=fs[0]

}

})

// 提交

$("#submit-btn").on("click", function () {

var formData = new FormData();

formData.append("applyName",$("#applyName").val());

formData.append("applyPhone",$("#applyPhone").val());

formData.append("file",file);

$.ajax({

headers: {

token:sessionStorage.getItem("userToken"),

},

type:"post",

url:\_path + "/review/addApply",

data:formData,

// dataType:"json",

processData: false,

contentType: false,

success:function(data) {

if (data.status == 0) {

window.location.href = "reviewSubmit.html";

}else if(data.status == 1){

webToast(data.data,"top",2000);

}

},

error:function(data){

if(!!data.responseJSON.message){

if(data.responseJSON.returnCode==="20009"){

webToast(data.responseJSON.message,"top",2000);

sessionStorage.clear();

login();

}

}else{

webToast("由于系统原因，提交失败！","top",2000);

}

}

});

});

**33.2下载**

①下载本地放置的图片

<img id="testimg1" src="./static/images/logo2.png">

<a href="javascript:;" class="down\_btn\_a">点击下载</a>

js：//判断是否为Trident内核浏览器(IE等)函数

function browserIsIe() {

if (!!window.ActiveXObject || "ActiveXObject" in window){

return true;

}

else{

return false;

}

}

//创建iframe并赋值的函数,传入参数为图片的src属性值.

function createIframe(imgSrc) {

//如果隐藏的iframe不存在则创建

if ($("#IframeReportImg").length === 0){

$('<iframe style="display:none;" id="IframeReportImg" name="IframeReportImg" onload="downloadImg();" width="0" height="0" src="about:blank"></iframe>').appendTo("body");

}

//iframe的src属性如不指向图片地址,则手动修改,加载图片

if ($('#IframeReportImg').attr("src") != imgSrc) {

$('#IframeReportImg').attr("src",imgSrc);

} else {

//如指向图片地址,直接调用下载方法

downloadImg();

}

}

//下载图片的函数

function downloadImg() {

//iframe的src属性不为空,调用execCommand(),保存图片

if ($('#IframeReportImg').src != "about:blank") {

window.frames["IframeReportImg"].document.execCommand("SaveAs");

}

}

//接下来进行事件绑定

var aBtn = $(".down\_btn\_a");

if (browserIsIe()) {

//是ie等,绑定事件

aBtn.on("click", function() {

var imgSrc = $(this).siblings("img").attr("src");

//调用创建iframe的函数

createIframe(imgSrc);

});

} else {

aBtn.each(function(i,v){

//支持download,添加属性.

var imgSrc = $(v).siblings("img").attr("src");

$(v).attr("download",imgSrc);

$(v).attr("href",imgSrc);

})

}

②在VUE中下载：

downTrainFile(row){

let \_this = this;

this.$http({

url: \_global.serverUrl + "/train/trainFileDownLoad/" + row.fileName,

method: 'get',

headers: {

'Content-Type': 'application/json; application/octet-stream'

},

responseType: 'blob',

}).then(function (res) {

const content = res.data;

const blob = new Blob([content]);

const fileName = row.fileName;

if ('download' in document.createElement('a')) { // 非IE下载

const elink = document.createElement('a');

elink.download = fileName;

elink.style.display = 'none';

elink.href = URL.createObjectURL(blob);

document.body.appendChild(elink);

elink.click();

URL.revokeObjectURL(elink.href); // 释放URL 对象

document.body.removeChild(elink);

} else { // IE10+下载

navigator.msSaveBlob(blob, fileName);

}

})

//或者使用 FileReader

axios.post('/path/to/download/url', this.searchParams, {

responseType: 'blob'

}).then(res => {  
 let blob = res.data  
 let reader = new FileReader()  
 reader.readAsDataURL(blob)  
 reader.onload = (e) => {  
 let a = document.createElement('a')  
 a.download = `表格名称.xlsx`  
 a.href = e.target.result  
 document.body.appendChild(a)  
 a.click()  
 document.body.removeChild(a)  
 }  
 })

}

③常用的下载方式

1、iframe加载请求此地址时会自动下载 //据说兼容先较好，推荐，使用的时候吧iframe设置为display;none即可

　　<iframe name="hehe" src='下载url'></iframe>

2、a点击请求此地址时会自动下载

　　<a href="下载url">下载</a>

　　H5的download属性 <a download href="下载url">下载</a> 这时候，如果返回的是img这种浏览器可以打开的文件时，也会下载（否则会直接打开）

3、form发起submit请求此地址时会自动下载

<form action="下载ur" method="get" >

<button type="submit">下载</button>

</form>

4、window.location.href 赋值为此地址时会自动下载 ，如果是其他地址，会跳转，但，当是下载的url，浏览器会发起请求，下载下来，并且不会改变本页的href

　　window.location.href = “下载url”

5、window.open赋值为此地址时会自动下载 ，但会先打开新标签页再下载再关闭新标签页，体验不佳

　　window.open = “下载url”

**33.3上传视频/图片预览/上传进度**  
 ①var reader = new FileReader();  
 this.file = e.target.files[0];  
 reader.onload = function () {

// \_this.$refs.video.src = this.result;(这种方式智能获取短的MP4格式属性，可用来浏览图片使用)

let url = URL.createObjectURL(\_this.file);

\_this.$refs.video.src = url;

//获取视频的时长，视频宽、高

\_this.$refs.video.addEventListener("loadedmetadata", function(e) {

\_this.$nextTick(() => {

console.log("获取参数");

});

console.log(this.duration, this.videoWidth, this.videoHeight);

});  
 };  
 reader.readAsDataURL(this.file);

问题：上面这种方法会造成每次new一个render，绑定很了多事件，即使上传到不是视频文件，只是后缀名相同（此时不会提示文件上传成功），当上传真正的视频文件时候，才会把上面的多有文件一起上传。

解决：直接绑定元素

<video ref="video" style="display:none;" @canplaythrough="myFunction($event)"></video>

<input id="uploadFile" type="file" style="display: none" @change="getUpLoadFiles($event)" />

// video上绑定的事件，视频赋值src触发

myFunction(ele) {

console.log(ele.target.duration,ele.target.videoWidth,ele.target.videoHeight);

}

// input上传事件

getUpLoadFiles($event) {

this.uploadTheFile = $event.target.files[0];

// 渲染视频

let url = URL.createObjectURL(this.uploadTheFile);

this.$refs.video.src = url; //会触犯video上的canplaythrough事件

}

②通过音频audio获取视频的时长

//获取视频或者音频时长

let fileurl = URL.createObjectURL(theFile);

//经测试，发现audio也可获取视频的时长

let audioElement = new Audio(fileurl);

let duration;

let time;

audioElement.addEventListener("loadedmetadata", \_event => {

time = Math.ceil(audioElement.duration); //单位：秒

uploadedFile.time = "00:00:" + time;

console.log(time);

if (time > 60) {

this.$message.warning("文件时长不能超过60秒！");

document.getElementById("uploadFile").value = "";

return;

}

}）

③文件上传，视频上传的进度

let formData = new window.FormData();

formData.append("file", theFile);

let config = {

headers: { "Content-Type": "multipart/form-data" },

//绑定进度

onUploadProgress: progressEvent => {

var complete =

((progressEvent.loaded / progressEvent.total) \* 100) | 0;

this.videoProgress = parseInt(complete);

console.log(this.videoProgress);

}

};

this.$http.post(this.api + "/file/uploadFile.do", formData, config).then(res=>{}）

**34.Vue获取元素高度**

// 当前区域  
<div ref="elememt"></div>   
①获取高度值 （内容高+padding+边框）  
let height= this.$refs.elememt.offsetHeight; //100  
②获取元素样式值 （存在单位）  
let height = window.getComputedStyle(this.$refs.element).height; // 100px  
③获取元素内联样式值（非内联样式无法获取）  
let height = this.$refs.element.style.height; // 100

**35.vue中cookie的使用**

方法一：①在js中定义好方法：

//获取cookie  
export function getCookie(name) {

var cookieArr = document.cookie.split("; ");

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

var arr = cookieArr[i].split("=");

if (arr[0] === key) {

return arr[1];

}

}

return false;  
 }  
 //设置cookie,增加到vue实例方便全局调用  
 export function setCookie (c\_name, value, expiredays) {  
 var oDate = new Date();

oDate.setDate(oDate.getDate() + iDay);

document.cookie = key + "=" + value + ";expires=" + oDate;  
 };  
 //删除cookie  
 export function delCookie (name) {  
 setCookie(name, ' ', -1); //设置有效时间为过去就删除  
 }

②main.js中全局配置，防止刷新丢失，返回的值undefined，需要在beforeMount中配置：  
import {setCookie, getCookie, delCookie} from './util/util.js';  
//参数因为是三个方法，所以用大括号来进行调用了，路径一定要写对  
export default {  
 name: 'App',  
 beforeMount () {  
 Vue.prototype.$cookieStore = {  
 setCookie,  
 getCookie,  
 delCookie  
 }  
 }  
}

③组件中调用：

设置：this.$cookieStore.setCookie( 'name' , this.ruleForm2.name) //‘name’是你自己取得名字，后面的this.ruleForm2.name是我写的一个input内v-model的值

获取：this.$cookieStore.getCookie( 'name')

删除：this.$cookieStore.delCookie( 'name');

方法二：使用vue-cookies插件

1.先安装

cnpm install vue-cookies --save-dev

2.在main.js引入

import VueCookies from 'vue-cookies'

Vue.use(VueCookies)

3.Api

设置 cookie：

this.$cookies.set(keyName, time) //return this

获取cookie：

this.$cookies.get(keyName) // return value

删除 cookie：

this.$cookies.remove(keyName) // return false or true , warning： next version return this； use isKey(keyname) return true/false,please

查看一个cookie是否存在（通过keyName）：

this.$cookies.isKey(keyName) // return false or true

获取所有cookie名称：

this.$cookies.keys() // return a array

时间格式

y 年

m 月

d 天

h 小时

min 分钟

s 秒

例如：

this.$cookies.set('key',value, '7d') //保存七天cookie

**36.视频和音频**

2.2:html5新特性-(2)-视频

<video src="x.mp4" width="400" height="300"></video>

知识补充:

(1)常用视频格式:.mp4 .flv .webm

(2)如果浏览器播放指定视频，需要安装"解码器"

遇到问题不能播放

解决:使用工具软件"格式工厂" x.mp4->x.flv->x.webm依次选择

<video>

<source src="x.mp4" />

<source src="x.flv" />

<source src="x.webm" />

您的浏览器版本太低，请升级!

</video>

-controls --是否显示控件 <video controls></video>

-autoplay --自动播放 <video autoplay></video>

#兼容性很差

-loop --是否循环播放 <video loop></video>

-muted --是否是静音播放

-poster --在播放之前显示广告图片 post="x.jpg"

-preload --预加载策略

auto:预先加载一定时长视频与元数据

metadata:预加载元数据(视频时长;尺寸;第一帧画面)——针对手机流量少的用户

none:不预加载数据

属性

var x = document.getElementById("myVideo");

-volume:1 当前音量

-playbackRate = 5; 修改视频回放速率 大于1快放 小于1慢播

-paused 获取当前视频播放状态(true为暂停/false为播放)

-x.play() 播放视频(可在script中控制播放和暂停)

-x.pause() 暂停播放视频

视频高级特性[事件]

-canplaythrough 当前视频加载结束可以播放会触发事件

属性：duration 获取当前视频总长度

-timeupdate 视频正在播放中(触发多次)

属性：currentTime 当前播时间点

-ended 播放结束

②html5新特性-音频

音频格式: x.mp3 x.wav x.ogg

<audio src="x.mp3"></audio>

#音频元素默认不可见，加controls

#属性和方法与视频大部分相同

练习:创建复选框控制背景音频播放

当复选框选中 背景音乐播放 play()

当复选框清空 背景音乐暂停 pause()

cb.change cb.checked==true

可以使用如下代码检测视频是否加载完成

<video id="video"></video>

let video = document.getElementById('video')

if (video.readyState === 4) {

console.log('finish!')

}

**37.获取定位**

①在工程目录index.html中引入：

<script type="text/javascript" src="http://api.map.baidu.com/api?v=2.0&ak=您的密钥"></script>

ps：秘钥可以去百度开发者平台申请

②在build/webpack.base.conf.js（与entry同级）中配置：

entry：{

app:'./src/main.js'

},

externals:{

"Bmap":"BMap"

},

③重新运行一下npm install或者yarn；

④配置好以后到对应需要获取城市名的页面引入百度地图对象 BMap：

import BMap from 'BMap'

⑤获取城市：

html部分

<div>{{LocationCity}}</div> <!-- 渲染层 -->

js部分

export default{

data(){

return{

LocationCity:"正在定位" //给渲染层定义一个初始值

}

},

mounted(){

this.city() //触发获取城市方法

},

methods:{

city(){ //定义获取城市方法

const geolocation = new BMap.Geolocation();

var \_this = this

geolocation.getCurrentPosition(function getinfo(position){

let city = position.address.city; //获取城市信息

let province = position.address.province; //获取省份信息

\_this.LocationCity = city

}, function(e) {

\_this.LocationCity = "定位失败"

}, {provider: 'baidu'});

}

}

}

有的时候会报错，BMap需要安装或者未定义，挂载类库出错的问题

解决：首先跟入口js一样，新建一个map.js  
export function MP(ak) {  
 return new Promise(function (resolve, reject) {  
 window.onload = function () {  
 resolve(BMap)  
 }  
 var script = document.createElement("script");  
 script.type = "text/javascript";  
 script.src = "http://api.map.baidu.com/api?v=2.0&ak="+ak+"&callback=init";  
 script.onerror = reject;  
 document.head.appendChild(script);  
 })

}

import {MP} from './map.js'  
mounted(){  
 this.$nextTick(function(){  
 var \_this = this;  
 MP(\_this.ak).then(BMap => {  
 //在此调用api

this.city();  
 })   
 })

}

**38.vue引入字体**

①自己定义一个文件夹common，放入下载好的.ttf文件

②创建一个font.css文件，内容：

@font-face {

font-family: "ContractedFont";

src: url('./ContractedFont.ttf');

font-weight: normal;

font-style: normal;

}

③在build的webpack.base.conf.js 文件中配置(有的自动已生成)

{

test: /\.(woff2?|eot|ttf|otf)(\?.\*)?$/,

loader: 'url-loader',

options: {

limit: 10000,

name: utils.assetsPath('fonts/[name].[hash:7].[ext]')

}

}

③在App.vue引入并使用

<style>

@import "./common/font/font.css";

#app {

font-family: "ContractedFont";

text-align: center;

color: #2c3e50;

margin-top: 10px;

}

</style>

**39.vue修改页面网站上显示的小图标**

①将所需图标放在static目录下，不放在src下就是因为浏览器找不到，找不到就加载不了，就不会有效果

②在index.html的head标签中添加

<link rel="icon" type="image/x-icon" href="static/webicon.ico" />

**40.IE兼容问题**

**40.1.vue兼容问题**

一、ie11打开vue2.0项目空白,控制台报错 vuex requires a Promise polyfill in this browser  
Promise为ES6语法，有的浏览器不支持ES6,所以需要安装babel-polyfill来将ES6转换为浏览器可识别的ES5语法  
①npm install --save-dev babel-polyfill  
②在main.js中引入 import "babel-polyfill"  
③在build文件夹中webpack.base.conf.js将  
//entry: {  
// app: ‘./src/main.js’  
// },  
替换为  
entry: {  
app:['babel-polyfill', './src/main.js']  
},  
二、ie11打不开vue项目，报错 “对象不支持“addEventListener”属性或方法”  
在index.html中head标签中加上  
< meta http-equiv=“X-UA-Compatible” content=“IE=EDGE”/>  
这个属性主要是设置浏览器优先使用什么模式来渲染页面的。#代码IE=edge告诉IE使用最新的引擎渲染网页

**40.2.带中文的URL搜索不了**

键输入的内容进行编译：encodeURI($("#mySearch").val(), "UTF-8");

①发送请求的时候将url进行编译

this.$http.get(encodeURI(url)).then...

②传输文件的时候将文件名编译

let fd = new FormData();

fd.append('file', file,encodeURI(file.name));

后台用java.net.URLDecoder进行解码：

fileName = URLDecoder.decode(fileDetail.getFileName(), "UTF-8");

**40.3IE跳转页面出现此错误**

代码中含有这个报的错

<span class="upload-file-size">{{parseFloat(item.size/1000000).toFixed(2)+'MB'}}</span>

**41.单个html页面使用elementUI**

方式一：

引入网络的css样式、网络的js

<link rel="stylesheet" href="https://unpkg.com/element-ui/lib/theme-chalk/index.css">

<script src="https://unpkg.com/element-ui/lib/index.js"></script>

方式二：

①将网络地址上的css和js下载下来再引入

<link rel="stylesheet" href="./static/css/index.css">

<script src="./src/index.js"></script>

②在css文件下建一个fonts文件夹，将下载好的字体放入

引用网络的css时候可以看到如下的字体引用链接：

问题：图标无法显示或者乱码（版本问题）

去查看elementUI的最新版本号，重新去下载字体fonts

https://unpkg.com/browse/element-ui@2.4.11/lib/theme-chalk/fonts/

**42.获取用户的IP地址**

<head>

<script src="<http://pv.sohu.com/cityjson?ie=utf-8>"></script>

</head>

<script>

// 获取用户ip和地址

console.log(returnCitySN["cip"] + "," + returnCitySN["cname"]);

</script>

**43.vue每次在新窗口打开页面**

import router from "../../router/index"

export function openNewPage(name, query = {}) {

const { href } = router.resolve({

name,

query

});

window.open(href, '\_blank');

}

**44.Element组件库中的el-table表格导出：**

需要的主要是两个依赖:(xlsx 和 file-saver)，安装：npm install --save xlsx file-saver

<!--给表格添加一个id，导出文件事件需要使用-->

<el-table :data="tableData" style="width: 100%" id="out-table"></el-table>

<!--给按钮绑定事件-→

<button @click="exportExcel">点击导出</button>

// 引入导出Excel表格依赖

import FileSaver from "file-saver";

import XLSX from "xlsx";

export default {

data() {

return {

tableData: []

};

},

methods: {

//定义导出Excel表格事件

exportExcel() {

/\* 从表生成工作簿对象 \*/

var wb = XLSX.utils.table\_to\_book(document.querySelector("#out-table"));

/\* 获取二进制字符串作为输出 \*/

var wbout = XLSX.write(wb, {

bookType: "xlsx",

bookSST: true,

type: "array"

});

try {

FileSaver.saveAs(

//Blob 对象表示一个不可变、原始数据的类文件对象。

//Blob 表示的不一定是JavaScript原生格式的数据。

//File 接口基于Blob，继承了 blob 的功能并将其扩展使其支持用户系统上的文件。

//返回一个新创建的 Blob 对象，其内容由参数中给定的数组串联组成。

new Blob([wbout], { type: "application/octet-stream" }),

//设置导出文件名称

"sheetjs.xlsx"

);

} catch (e) {

if (typeof console !== "undefined") console.log(e, wbout);

}

return wbout;

}

};

**45.过滤器**

①//在sliceString.js中：

import Vue from 'vue'

// 字符串长度过滤器

export default Vue.filter('sliceString', function (str, start, length) { //str是竖杠前面传入的值，start和length 是传入的参数

if (str) {

if (str.length <= (length - start)) {

return str

} else {

return str.slice(start, length) + '...'

}

}

})

②//在main.js中引入：

import "./filter/sliceString"

③在vue中使用：

<span>{{item.content | sliceString(0,30)}}</span>

**46.vue-router**

属性：

$router.app ：配置了router的Vue根实例

$router.mode：路由模式，这里是hash

$router.currentRoute：当前路由的路由信息对象，包含了当前匹配路由的信息，如下：

|  |
| --- |
| fullPath: "/home/case\_center"  hash: ""  matched: [  {path: "/home", regex: /^\/home(?:\/(?=$))?$/i, components: {…}, instances: {…}, name: "home", …}  {path: "/home/case\_center", regex: /^\/home\/case\_center(?:\/(?=$))?$/i, components: {…}, instances: {…}, name: "case\_center", …}  ]  meta:{  requireAuth: true  type: ["2"]  }  name: "case\_center"  params: {}  path: "/home/case\_center"  query: {} |

方法：

router.addRoutes(routes)：动态添加路由规则，参数为符合routes选项要求的数组。

**47.页面加载进度条(html的顶端)**

①安装 nprogress：npm install --save nprogress

②在main.js中配置：

import NProgress from 'nprogress'

import 'nprogress/nprogress.css'

NProgress.configure({

easing: 'ease', // 动画方式

speed: 500, // 递增进度条的速度

showSpinner: false, // 是否显示加载ico

trickleSpeed: 200, // 自动递增间隔

minimum: 0.3 // 初始化时的最小百分比

})

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

// 每次切换页面时，调用进度条

NProgress.start();

next();

});

router.afterEach(() => {

// 在即将进入新的页面组件前，关闭掉进度条

NProgress.done()

})

**48.Axios 各种请求方式传递参数**

get delete 方法较为不同  
 get请求方式将需要入参的数据作为 params 属性的值，最后整体作为参数传递  
 delete请求方式将需要入参的数据作为 data 属性的值，最后整体作为参数传递  
①get：  
Axios.get('demo/url', {  
 params: {  
 id: 123,  
 name: 'Henry'  
 }  
})  
②delete：  
Axios.delete('demo/url', {  
 data: {  
 id: 123,  
 name: 'Henry',  
 }  
})  
③post/put/patch：  
Axios.post('demo/url', {  
 id: 123,  
 name: 'Henry'  
})

**49.Vue中使echarts自适应大小**

mounted(){

const myChart = this.$echarts.init(document.getElementById('map'))

myChart.setOption(option);

// 安装 element-resize-detector

var elementResizeDetectorMaker = require("element-resize-detector");

var erd = elementResizeDetectorMaker();

// var erdUltraFast = elementResizeDetectorMaker({

// strategy: 'scroll' //<- For ultra performance.

// })

//监听元素size变化，触发响应事件

erd.listenTo(document.getElementById("map"), function(element) {

// 获取到echarts的画本对象，调用resize方法重新绘图

myChart.resize();

});

}

**50.vue-router钩子beforeRouteEnter**

beforeRouteEnter (to, from, next) {  
 // 在渲染该组件的对应路由被 confirm 前调用  
 // 不！能！获取组件实例 `this`，因为当钩子执行前，组件实例还没被创建  
 },  
 beforeRouteUpdate (to, from, next) {  
 // 在当前路由改变，但是该组件被复用时调用  
 // 举例，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转时  
 // 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。  
 // 可以访问组件实例 `this`  
 },  
 beforeRouteLeave (to, from, next) {  
 // 导航离开该组件的对应路由时调用  
 // 可以访问组件实例 `this`  
 }

|  |
| --- |
| 通过vm访问内部变量/方法  export default {  data(){  return {  num: 18  }  },  beforeRouteEnter(to, from, next){  next(vm=>{  vm.num=19; //不能使用this  })  } } |

**51.修改数据和对象-$set（防止不会触发更新）**

什么情况下使用$set？  
 受现代 JavaScript 的限制，Vue 不能检测到对象属性的添加或删除。由于 Vue 会在初始化实例时对属性遍历执行 getter/setter 转化过程，所以属性必须在 data 对象上存在才能让 Vue 转换它，这样才能让它是响应的。  
①数组：this.$set(Array, index, newValue)  
当你利用索引直接设置一个项时，例如：vm.items[indexOfItem] = newValue  
解决：用$set方法  
当你修改数组的长度时，例如：vm.items.length = newLength  
解决：vm.items.splice(newLength)  
②对象：this.$set(Object, key, value)  
有时你想向已有对象上添加一些属性，例如使用 Object.assign() 或 \_.extend() 方法来添加属性。  
但是，添加到对象上的新属性不会触发更新。  
在这种情况下可以创建一个新的对象，让它包含原对象的属性和新的属性：  
this.someObject = Object.assign({}, this.someObject, { a: 1, b: 2 })

**五、CSS样式**

**1.文字显示不下用省略号(...)**

①显示一行：  
 overflow:hidden;  
 text-overflow:ellipsis; //省略号  
 white-space:nowrap; //不换行  
②显示多行：  
 overflow: hidden;  
 text-overflow: ellipsis;  
 display: -webkit-box;  
 -webkit-line-clamp:2; //只适用webkit内核，在第二行的时候用...  
 -webkit-box-orient: vertical;

**2.修改checkbox框样式**

<input id="select1" type="checkbox" :value="11" v-model="text"> //v-model存了所有选择的value值

<label for="select1" @click="con"></label> //用label覆盖input

input{

margin-right: 0.1rem;

display: none;

}

label{

display: block;

width: 15px;

height: 15px;

line-height: 15px;

cursor: pointer;

position: absolute;

top: calc(50% - 7.5px); //-两边必须空格

left:-15px;

background: rgba(255, 255, 255, 1);

border:1px solid #aaa;

}

input:checked + label::before{

display: block;

content: "\2714";

text-align: center;

font-size: 16px;

color: #000;

}

**3.淡入淡出动画效果**

/\*定义淡入淡出动画的初始状态,常驻元素\*/

.fade{

top:50%;

left:50%;

width:0px;

height:50px;

margin-left:0px;

margin-top:-35px;

opacity:0;

transition:all 1s linear;

}

/\*定义淡入淡出动画的结束状态\*/

.in{

top:50%;

left:50%;

width:350px;

height:50px;

margin-left:-185px;

margin-top:-35px;

opacity:1;

}

**4.文字两端对齐**

<ul class="arter">

<li><span>作品名称<i></i></span>： 宝贝儿</li>

<li><span>作品类型<i></i></span>： 油画</li>

<li><span>艺术家<i></i></span>： 张玉瀛</li>

<li><span>风格<i></i></span>： 超现实</li>

</ul>

css样式：

li {

line-height: 0.4rem;

span {

height:0.4rem;

line-height: 0.4rem;

width: 20%;

text-align: justify;

display: inline-block;

overflow: hidden;

vertical-align: middle;

i {

display: inline-block;

width: 100%;

height: 0;

}

}

}

**5.当需要栅格布局时，各占50%元素会自动换行**

原因：inline-block默认会有4px的边距，是由于代码本身的换行、空格产生的，无法通过margin:0消除

解决：父元素加属性font-size=0，此时所有字符包过空字符在内，字体大小均为0，不会占位置，子元素再设置字体大小

**6.border画三角形**

div {

width: 0;

height: 0;

border-width: 0 40px 40px; //上边边框宽度0，不占面积

border-style: solid;

border-color: transparent transparent red; //下边红色三角形

}

**7.js改变伪类:after**

<div class="showInfo" ref="showInfo" id="showInfo" @click="handle">点我</div>

handle(){

document.styleSheets[0].addRule('#showInfo::after', 'top:86px;border-top:12.5px solid white;border-bottom:0');

}

**8.css动画做进度条**

div{

height: 10px;

border: 1px solid;

background: linear-gradient(#0ff,#0ff);

background-repeat: no-repeat;

background-size: 0;

animation: move 2s linear forwards;

}

div:hover{

animation-play-state:paused //鼠标放上去暂停动画，默认值：running播放

}

@keyframes move{

100%{

background-size: 100%;

}

}

**9.禁止文本被复制/阻止事件冒泡**

user-select: none;

pointer-events: none;

**10.长文本保留格式——空格、换行等（用户协议等）**

<pre class="text">

所有文字内容

</pre>

.text {

font-family: "微软雅黑";

white-space: pre-wrap; //自动换行

}

**11.滚动条的设置**

滚动条的设置  
1. ::-webkit-scrollbar 滚动条整体部分(高宽分别对应横竖滚动条的尺寸)  
2. ::-webkit-scrollbar-button 滚动条两端的按钮  
3. ::-webkit-scrollbar-track 外层轨道  
4. ::-webkit-scrollbar-track-piece 内层滚动槽  
5. ::-webkit-scrollbar-thumb 滚动的滑块  
6. ::-webkit-scrollbar-corner 边角  
7. ::-webkit-resizer 定义右下角拖动块的样式

1.object-fit在ie11和edge中不兼容，可以一些css hacks

①使用background-size和background-position替换object-fit设置图片的样式

<img class="loadingImage" src="url"/>

.loadingImage {

width: 100%;

height: 100%;

transition: all 1s ease;

object-fit: cover;

}

上述代码中可以修改为如下：

<div class="loadingImage"></div>

.loadingImage {

width: 100%;

height: 100%;

background-size: cover;

background-position: center;

background-image: url(url);

}

②对于视频播放，object-fit:cover可以解决视频不会随着屏幕缩放的问题

<video class="video"></video>

.video {

width: 100%;

height: auto;

position: relative;

left: 50%;

top: 50%;

transform: translate(-50%, -50%);

object-fit: cover;

}

可以使用如下css设置video标签，解决object-fit不兼容ie和edge的问题

<video class="video"></video>

.video {

width: 100%;

height: auto;

position: relative;

left: 50%;

top: 50%;

transform: translate(-50%, -50%);

object-fit: fill;

}

**六.JS知识**

1.数组API

①ES6提供了Array.includes()函数判断是否包含某一元素，除了不能定位外，解决了indexOf的上述的两个问题。它直接返回true或者false表示是否包含元素，对NaN一样能有有效。

const arr1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', NaN]

console.log('%s', arr1.includes('c')) ->true

console.log('%s', arr1.includes('z')) ->false

console.log('%s', arr1.includes(NaN)) ->true

②find()函数用来查找目标元素，找到就返回该元素，找不到返回undefined。

findIndex()函数也是查找目标元素，找到就返回元素的位置，找不到就返回-1。

find((element) => (element.name == '李四'));返回的是{name: "李四", gender: "男", age: 20}这个元素  
stu.findIndex((element)=>(element.name =='李四'));返回的是索引下标：2

③查找: indexOf(元素值,starti) 【等同String中的indexOf("关键词",starti)】

从下标i位置开始，找下一个"元素"的位置。

在数组中找到指定元素，返回元素下标；没找到，则返回-1。

④判断:

(1) 判断所有元素是否都符合要求

var bool=arr.every(function(elem, i, arr){

return 根据elem,I,arr判断结果; ——>true/false

})

只有回调函数在每个元素上执行都返回true时，才整体返回true；只要碰到一个false，就返回false。

(2) 判断数组中是否包含符合要求的元素

var bool=arr.some(function(elem, i, arr){

return 判断条件

})

只要碰到一个true，就返回true。

⑤ 遍历API: 对数组中每个元素执行相同的操作（无返回值）

(1) 遍历原数组中每个元素，执行相同的操作:

arr.forEach(function(elem, i, arr){

对原数组中的元素值，执行操作

})

(2) 依次取出原数组中每个元素，执行相同的操作后，放入新数组中返回（有返回值）:

var arr1=arr.map(

//自动先在内部创建新数组newArr[ ]

function(elem,i,arr){

return 新值; //返回新数组中，最后给arr1

})

⑥数组去重

方法一：var arr = [1, 2, 2, 3, 4, 5, 5, 6, 7, 7];

var arr2 = arr.filter(function(x, index,self) {

return self.indexOf(x)===index; //indexOf每次找到的是第1个的下标

});

console.log(arr2); //[1, 2, 3, 4, 5, 6 ,7]

方法二：const list = [ 1, 1, 2, 3, 6, 45, 8, 5, 4, 6, 5]  
const uniqueList=[...new Set(list)]

2.js 判断页面是否加载完成

document.onreadystatechange = subSomething; 　　　 //当页面加载状态改变的时候执行这个方法

function subSomething()

{

if(document.readyState == “complete”) //判断页面加载状态

myform.submit();

}

说明如下：

Document.readyState属性描述了文档的加载状况，一个文档的readyState可能是以下的其中一个：

loading：文档仍然在加载

interactive：文档已经加载完毕而且已经被解析，但是一些子资源，例如图像，样式表和框架还在加载。这个状态表明DOMContentLoaded事件已经被触发。

complete：文档和全部的子资源已经加载完毕。这个状态表明load事件已经被触发。

当readyState属性值改变的时候，document对象的readystatechange事件被触发。

⑦获取数组中最大/最小值

方法一：使用 Math 中的 max/min 方法

可以使用apply来实现。apply传入的是一个数组。

var arr = [22,13,6,55,30];

var max = Math.max.apply(null, arr);

var min = Math.min.apply(null, arr);

console.log(max, min) // 55,6

方法二：使用ES6的扩展运算符

var arr = [22,13,6,55,30];

console.log(arr.Math(...arr)); // 55

方法三：是用sort自取值或者假想法

⑧获取数组中对象某一个值的最大/最小值

list: [{ id: 1, name: 'jack' },  
 { id: 2, name: 'may' },  
 { id: 3, name: 'shawn' },  
 { id: 4, name: 'tony' }]

方法一：Math方法  
 let max = Math.max.apply(Math,this.list.map(item => { return item.id }))  
 let min = Math.min.apply(Math,this.list.map(item => { return item.id }))  
方法二：sort排序  
注意：sort()对数组排序，不开辟新的内存，对原有数组元素进行调换, 所以这种操作会使得原来的数组元素的位置发生变化  
 this.list.sort((a, b) => { return b-a })[0].id   
 this.list.sort((a, b) => { return a-b })[0].id

**2.时间转换**

exptime: (val) => {

// 日期格式yyyy-MM-dd HH-mm-ss

// 转换时间

let endtime = new Date(val.replace(/-/g, '/')).getTime()

let now = new Date().getTime()

// 截止日期减去现在的日期

var my\_time = endtime - now

// console.log(endtime)

// console.log(now)

// console.log(my\_time)

var days = my\_time / 1000 / 60 / 60 / 24

var daysRound = Math.floor(days)

var hours = my\_time / 1000 / 60 / 60 - (24 \* daysRound)

var hoursRound = Math.floor(hours)

var minutes = my\_time / 1000 / 60 - (24 \* 60 \* daysRound) - (60 \* hoursRound)

var minutesRound = Math.floor(minutes)

var seconds = my\_time / 1000 - (24 \* 60 \* 60 \* daysRound) - (60 \* 60 \* hoursRound) - (60 \* minutesRound)

// console.log('转换时间:', daysRound + '天', hoursRound + '时' + minutesRound + '分' + seconds + '秒')

var time = daysRound + '天' + hoursRound + '小时' + minutesRound + '分' + parseInt(seconds) + '秒'

return time

}

**3.try的使用**(页面需判断是否有返回值，再决定是否调用另一个接口时使用)

// 如果横屏的案例没有，调用竖屏的案例

methods:{

async beforeGetBusinessData() {

try {

let result = await this.$http.post(this.api +"/case/providerCase", this.params1)

.then(res => {

return res.data.data;

});

//判断返回值并判断是否抛出异常

if (result.cases.length > 0) {

this.businessDetailsData = result;

this.totalNumber = result.totalNum;

} else {

throw "-1";

}

//alert("I am try"); //只要上面的语句出现错误或者throw语句这句将不会被执行

} catch (err) { //err是throw抛出的错误

//如果try中发现错误，则执行catch中的语句，如果没有错误，则跳过catch

console.log(err);

if (err === "-1") {

this.$http.post(this.api + "/case/providerCase", this.params2)

.then(res => {

this.businessDetailsData = res.data.data;

this.totalNumber = res.data.data.totalNum;

});

}

} finally {

alert("无论如何都会执行");

}

}

created(){

this. beforeGetBusinessData()

}

方法二：使用promise解决调用顺序

methods:{

// 如果横屏的案例没有，调用竖屏的案例

beforeGetBusinessData() {

new Promise((resolve, reject) => {

this.$http.post(this.api + "/case/providerCase", this.params1)

.then(res => {

if (!!res.data.data) {

this.businessDetailsData = res.data.data;

this.totalNumber = res.data.data.totalNum;

}

resolve(this.totalNumber);

});

}).then(num => { //num是resolve返回的值

// 判断是否有横屏的案例，没有就调用竖屏的案例

if (!num) {

this.againGetBusinessData();

}

});

},

// 再次获取竖屏的案例

againGetBusinessData() {

this.$http.post(this.api + "/case/providerCase", this.params2)

.then(res => {

if (!!res.data.data) {

this.businessDetailsData = res.data.data;

this.totalNumber = res.data.data.totalNum;

}

});

},

},

created(){

this. BeforeGetBusinessData();

}

**4.解构**

// 对象解构

let node = { type: "Identifier", name: "foo"};

①let { type, name } = node;

console.log(type); //"Identifier"

console.log(name); //"foo"

②重命名原对象中的属性名：

let { type: localType, name: localName, value = true } = node;

console.log(localType); //"Identifier"

console.log(localName); //"foo"

console.log(value); //true

//数组解构

let arr = ["0", "1", "哈哈"];

①将数组分解成单个的字符串

console.log(...arr) //0 1 哈哈

②通过值在数组中的位置进行选取，且可以存储在任意变量中，未显式声明的元素都会直接被忽略。在这个过程中，数组本身不会发生任何变化：

let [first, second] = arr;

console.log(first); //"0"

console.log(second); //"1"

③通过…语法将数组中的其余元素赋值给一个特定的变量

let [a, ...narr] = arr;

console.log(a); //0

console.log(narr); //["1", "哈哈"]

④ 数组解构为对象(默认对象名：0,1,2,3,...)：

let obj = { ...arr };

console.log(obj); //{ 0:"0", 1:"0", 2:"哈哈" }

再解构对象并重命名对象名：

let { 0: a, 1: b, 2: c } = obj;

console.log(a); //0

console.log(b); //1

console.log(c); //哈哈

**5.if的简写**

①三目：条件 ？条件成立 : 条件不成立  
 2+3 ? alert("true") : alert("false")

2+3 ? alert("true") : null  
②只有一个语句：if() 语句  
 if(true) alert("true")  
③&& 条件为真走&&后面的语句  
 9%3！=0 && alert('成立')  
④|| 条件不管成不成立，都走||后面的语句  
 9%3！=0 || alert('成立')

**6.sessionStorage**

分为四步：

①在浏览器中打开这个 index.html，我们称之为标签页 A。注意：需要用 http 协议打开！例如 http://localhost/index.html

②点击页面上的链接，此时会弹出来标签页 B。

③在标签页 B 中打开控制台并执行 sessionStorage.getItem('j')，得到 's'

④新建一个新标签页 D，然后在地址栏内输入 http://localhost/index.html 打开同样的页面， 然后执行 sessionStorage.getItem('j') 。

-->>预期：标签页 D 得到的应该还是 's'，毕竟我认为 sessionStorage 的数据是在同一网站的多个标签页之间共享的。

-->>结果：得到的结果是 null。

-->>原因：通过点击链接（或者用了 window.open）打开的新标签页之间是属于同一个 session 的，但新开一个标签页总是会初始化一个新的 session，即使网站是一样的，它们也不属于同一个 session。