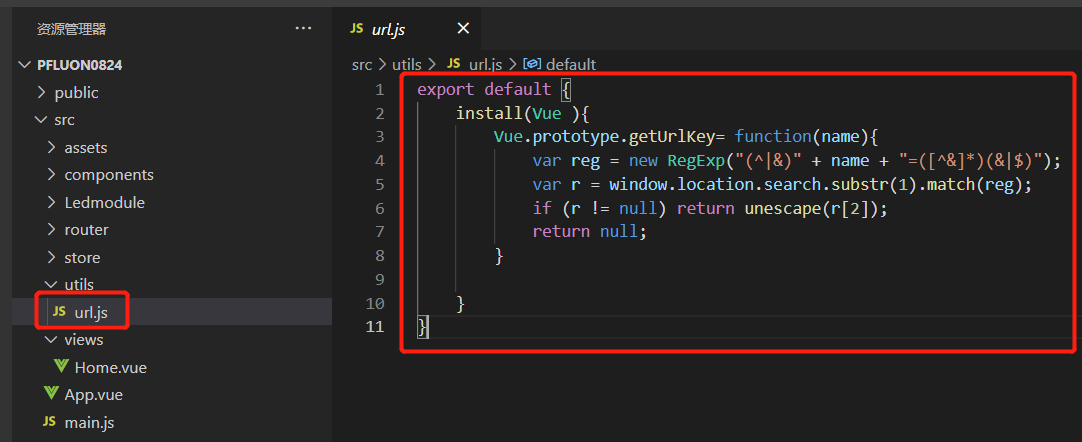
# 初始化url并传参

### 通过参数名获取请求参数的方法



export default {

    install(Vue ){

        Vue.prototype.getUrlKey= function(name){

            var reg = new RegExp("(^|&)" + name + "=([^&]\*)(&|$)");

            var r = window.location.search.substr(1).match(reg);

            if (r != null) return unescape(r[2]);

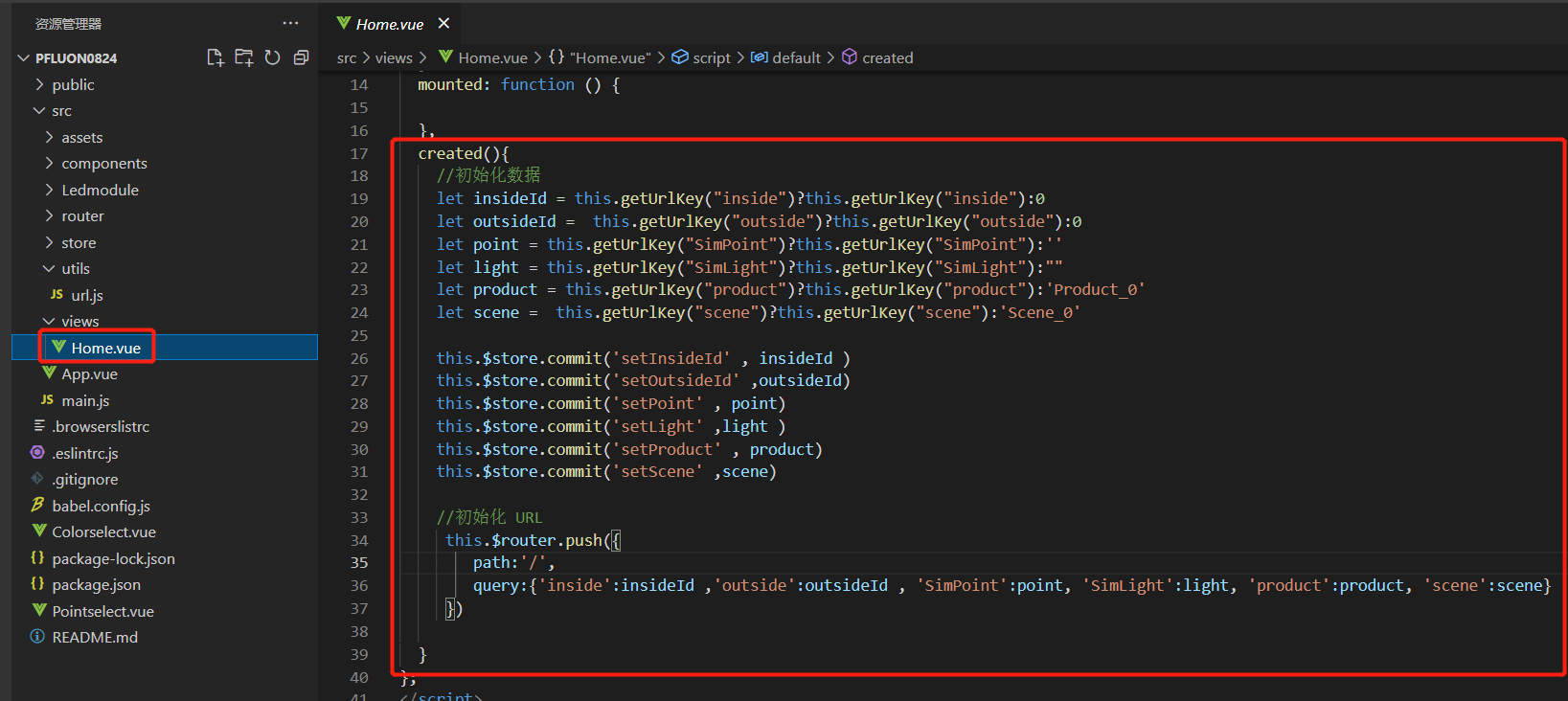
            return null;

        }

    }

}

### 初始化数据和URL



    //初始化数据

    let insideId = this.getUrlKey("inside")?this.getUrlKey("inside"):0

    let outsideId =  this.getUrlKey("outside")?this.getUrlKey("outside"):0

    let point = this.getUrlKey("SimPoint")?this.getUrlKey("SimPoint"):''

    let light = this.getUrlKey("SimLight")?this.getUrlKey("SimLight"):""

    let product = this.getUrlKey("product")?this.getUrlKey("product"):'Product\_0'

    let scene =  this.getUrlKey("scene")?this.getUrlKey("scene"):'Scene\_0'

    // 将获取到的参数发送给vuex，方便程序使用

    this.$store.commit('setInsideId' , insideId )

    this.$store.commit('setOutsideId' ,outsideId)

    this.$store.commit('setPoint' , point)

    this.$store.commit('setLight' ,light )

    this.$store.commit('setProduct' , product)

    this.$store.commit('setScene' ,scene)

    //初始化 URL

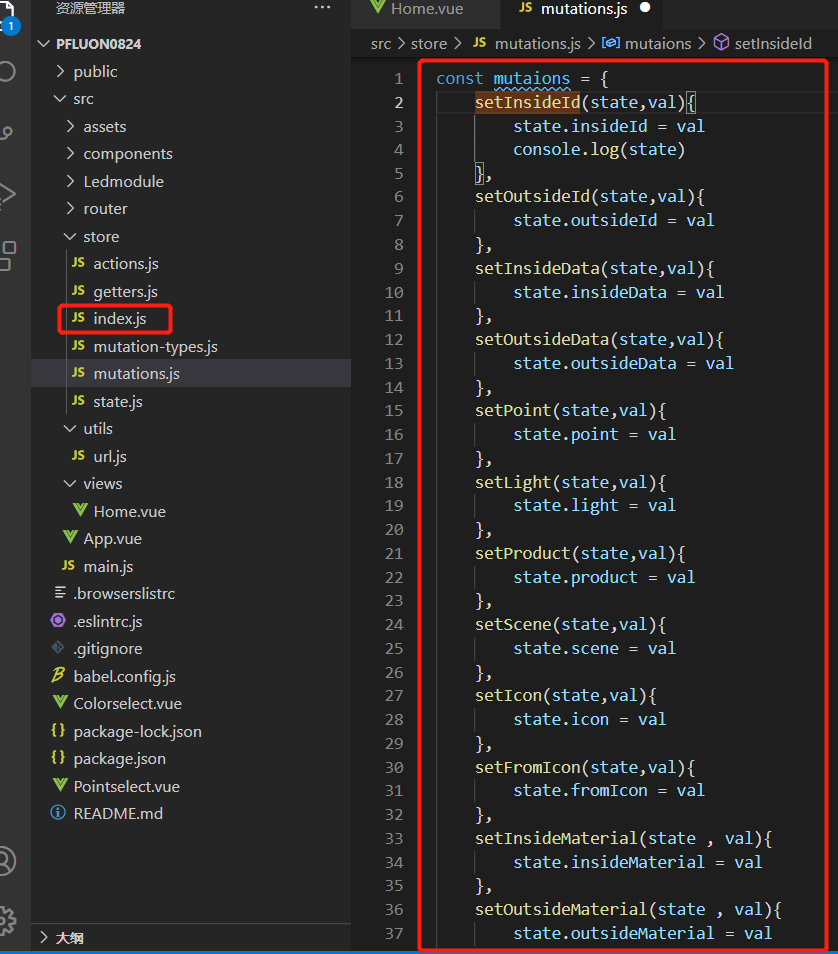
     this.$router.push({

        path:'/',

        query:{'inside':insideId ,'outside':outsideId , 'SimPoint':point, 'SimLight':light, 'product':product, 'scene':scene}

     })

在vuex中保存参数



    setInsideId(state,val){

        state.insideId = val

        console.log(state)

    },

    setOutsideId(state,val){

        state.outsideId = val

    },

    setInsideData(state,val){

        state.insideData = val

    },

    setOutsideData(state,val){

        state.outsideData = val

    },

    setPoint(state,val){

        state.point = val

    },

    setLight(state,val){

        state.light = val

    },

    setProduct(state,val){

        state.product = val

    },

    setScene(state,val){

        state.scene = val

    },

    setIcon(state,val){

        state.icon = val

    },

    setFromIcon(state,val){

        state.fromIcon = val

    },

    setInsideMaterial(state , val){

        state.insideMaterial = val

    },

    setOutsideMaterial(state , val){

        state.outsideMaterial = val

    },

# 可滑动的li列表

## 效果



## ul的父元素样式

overflow: hidden;

## ul的样式

    background-color: transparent;

    list-style-type: none;

    padding: 0px;

    margin: 0px;

    // 设置list滑块的方法

    // 设置弹性盒子

    display: flex;

    // 主轴自左向右排列

    flex-direction: row;

    // 不自动换行

    flex-wrap: nowrap;

    // 辅轴居中

    align-items: center;

    // 横向滚动

    overflow-y: scroll;

    // 隐藏滚动条

    &::-webkit-scrollbar {

      display: none;

    }

## li的样式

    margin-right: 10px;

    width: 35px;

    height: 35px;

    background-color: #0077b6;

    margin-top: 15px;

    border-radius: 50%;

    background-size: cover;

    // 不收缩子元素

    flex-shrink: 0;

# 封装axios请求



// 引入axios

import axios from 'axios'

// 设置基地址

axios.defaults.baseURL = "http://120.25.126.174:8160/";

// 设置请求拦截器

axios.interceptors.request.use(config =>{

    // 设置请求头，为Authorization赋值token令牌

    config.headers.Authorization = window.sessionStorage.getItem("token");

    // 在最后必须return config

    return config;

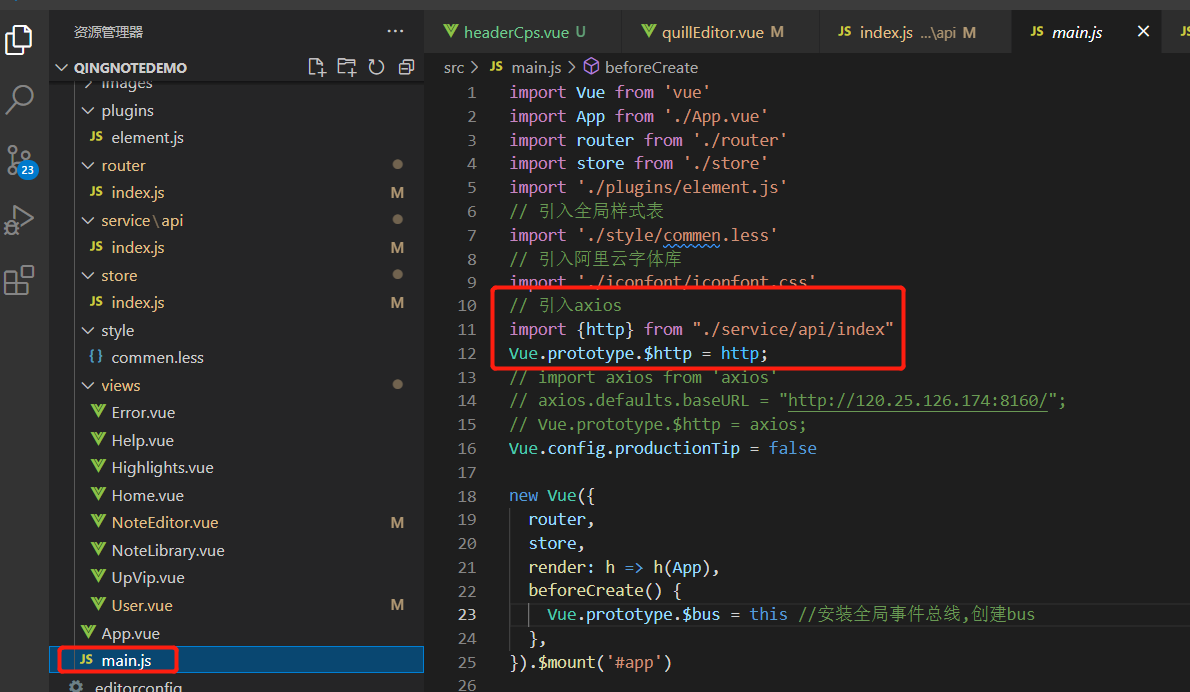
})

//获取邮箱验证码

const getEmailVerify = (email)=>axios.get(`educenter/email/send/${email}`);

const emailLogin = (emailInfo)=>axios.post("/educenter/email/login",emailInfo);

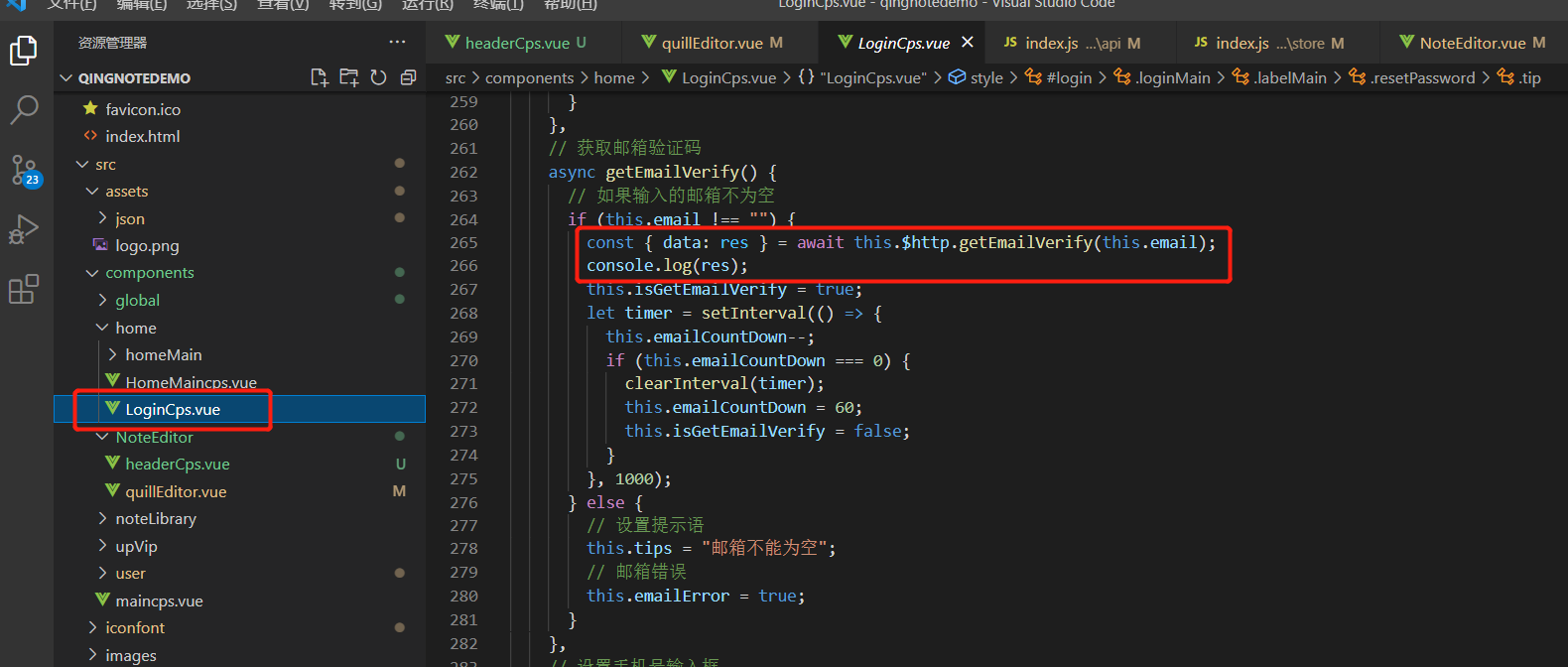
export const http = {getEmailVerify:getEmailVerify,emailLogin:emailLogin};



// 引入axios

import {http} from "./service/api/index"

Vue.prototype.$http = http;

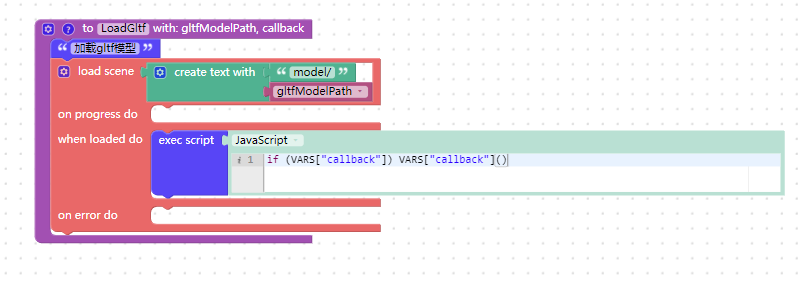


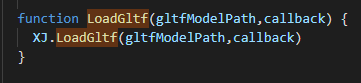
        const { data: res } = await this.$http.getEmailVerify(this.email);

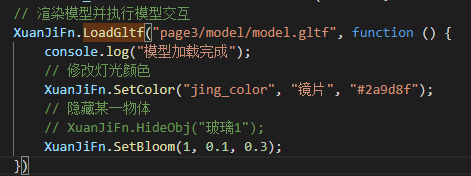
        console.log(res);

# 回调函数

·如果需要在某段函数加载完在执行相应的代码，可以为函数设置callback回调函数的参数，然后if(callback) callback()，在回调函数中执行相应代码即可







# Element-UI 给el-dropdown添加滚动条

下拉菜单el-dropdwn的HTML代码：



其中设置了project-dropdown的CSS样式如下：



.project-dropdown{

//设置高度才能显示出滚动条 !important

height:300px;

overflow: auto;

}

.project-dropdown::-webkit-scrollbar

{

width: 5px;

height: 5px;

background-color: #F5F5F5;

}

.project-dropdown::-webkit-scrollbar-track

{

//-webkit-box-shadow: inset 0 0 6px rgba(0,0,0,0.3);

border-radius: 10px;

background-color: #F5F5F5;

}

# Js实现点击某元素之外触发事件

  created() {

    document.addEventListener("click", (e) => {

      // 某元素

      const ProfileMore = document.querySelector(".ProfileMore");

      // 如果点击的不是ProfileMore和Profile元素

        if (!ProfileMore.contains(e.target)) {

          // 业务逻辑

        }

    });

  },

# jQuery对象与DOM对象的转换

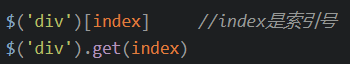
## DOM对象转换为jQuery对象

·使用$()，得到的就是jQuery对象





## jQuery转换为DOM对象



# 在 Vue 中使用 Jquery UI

## 安装依赖

·npm install jquery --save

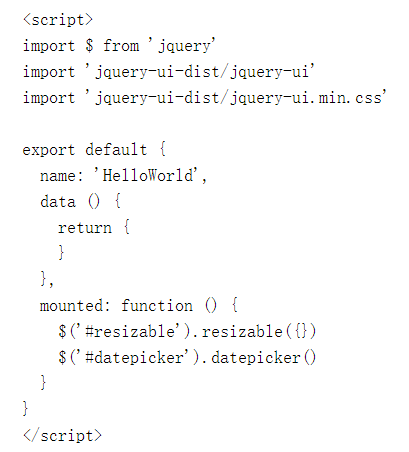
·npm install jquery-ui-dist --save

## 在组件里引用

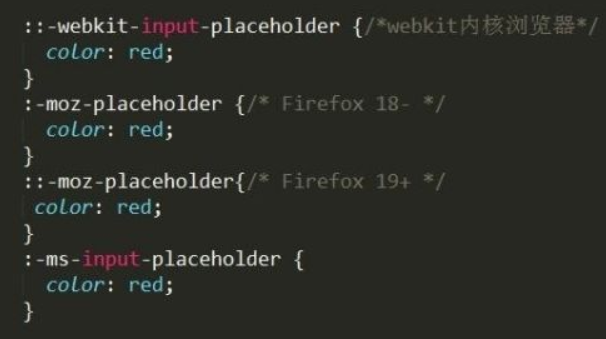
·import $ from 'jquery'

·import 'jquery-ui-dist/jquery-ui'

·import 'jquery-ui-dist/jquery-ui.min.css'



# 修改placeholder的字体样式



# 使input失去焦点

input.blur();

# vue中监听input标签事件

input实时监听事件

在vue中，原生input标签（这里不说UI框架封装过的input标签）监听value值实时变化的事件为v-on:input方法

IMG_256

input获取焦点事件：

<input type=“text” placeholder=“请输入” @blur=“XXX()”>

IMG_256

input失去焦点事件：

<input type=“text” placeholder=“请输入” @focus=“xxx()”>

IMG_256

# vue-awesome-swiper

## 安装

cnpm i vue-awesome-swiper@3.1.3 -S

## 使用

<template>

  <!-- v-if解决循环轮播失效问题 -->

  <swiper

    class="swiper"

    ref="mySwiper"

    :options="swiperOption"

    v-if="sowing\_list.length > 1"

  >

    <!-- 遍历所有轮播图 -->

    <swiper-slide

      v-for="(sowing, index) in sowing\_list"

      :key="sowing.public\_id"

    >

      <img :src="sowing.icon\_url" :alt="sowing.public\_name" />

    </swiper-slide>

    <div class="swiper-pagination" slot="pagination"></div>

  </swiper>

</template>

<script>

//页面引入swiper

import "swiper/dist/css/swiper.css";

import { swiper, swiperSlide, directive } from "vue-awesome-swiper";

export default {

  name: "Sowing",

  //   props: {

  //     // 使用props接收轮播图数据，数据类型为Array

  //     sowing\_list: Array,

  //   },

  // 使用props接收轮播图数据

  props: ["sowing\_list"],

  //注册轮播图组件

  components: {

    swiper,

    swiperSlide,

  },

  directives: {

    swiper: directive,

  },

  data() {

    return {

      swiperOption: {

        notNextTick: true,

        pagination: {

          el: ".swiper-pagination",

          //当单击指示器时会执行过渡动画到目标slide

          clickable: true,

          //定义pagination 分页器内当前活动块的指示小点的类名。

          bulletActiveClass: "my-swiper-select",

        },

        loop: true, // 循环

        autoplay: {

          delay: 1000, // 自动播放,切换时长

          // 用户操作swiper之后自动切换不会停止，每次都会重新启动autoplay。

          disableOnInteraction: false,

          // 鼠标置于swiper时暂停自动切换，鼠标离开时恢复自动切换。

          pauseOnMouseEnter: true,

        },

        speed: 600, // 速度

        direction: "horizontal", // 滚动方向

        on: {

          // 滑动后监听,回调函数，当切换图片时

          slideChangeTransitionEnd() {

            // 获取当前图片索引

            // console.log(this.activeIndex);

          },

        },

      },

    };

  },

  // 计算属性

  computed: {},

  mounted() {},

};

</script>

<style lang="less" scoped>

.swiper {

  height: 14rem;

  width: 100%;

  background-color: transparent;

  position: relative;

  img {

    width: 90%;

    height: 90%;

    position: absolute;

    // 图片居中

    left: 50%;

    top: 50%;

    transform: translate(-50%,-50%);

    // 圆角

    border-radius: 10px;

  }

  // 设置轮播图指示点的样式

  & /deep/ .my-swiper-select {

    background-color: #75a342;

    opacity: 1;

  }

}

</style>

# Canvas使用

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title></title>

<script src="https://cdn.bootcdn.net/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

<style>

\* {

margin: 0;

padding: 0;

font-size: 2.1333328vw;

}

html, body {

height: 100%;

width: 100%;

}

canvas {

display: block;

background: #D6F8FF;

}

@font-face {

/\* 指定字体的名字 \*/

font-family: 'nameFont' ;

/\* 服务器中字体的路径 \*/

src: url('font/nameFont.ttf') format("truetype");

}

#tu{

width: 100%;

/\*height: 500px;\*/

/\*background-color: cornflowerblue;\*/

}

</style>

<script type="text/javascript">

var ***canvas*** = ***document***.createElement('canvas');

***canvas***.id = "myCanvas";

$(function () {

***console***.log(***canvas***);

//添加窗口尺寸改变响应监听

$(***window***).resize(resizeCanvas);

//页面加载后先设置一下canvas大小

resizeCanvas();

});

//窗口尺寸改变响应（修改canvas大小）

function resizeCanvas() {

***canvas***.width = ***window***.innerWidth\*3;

***canvas***.height = ***window***.innerHeight\*3;

canvasa("欧阳娜娜");

};

function canvasa(text) {

var c = ***canvas***;

// console.log(c);

// 创建 context 对象,getContext("2d") 对象是内建的 HTML5 对象，拥有多种绘制路径、矩形、圆形、字符以及添加图像的方法。

var ctx = c.getContext("2d");

// 设置背景图

// 创建img标签

var img = new ***Image***();

// 引入图片

img.src = "./imgs/img3.png";

//当前元素加载完成时才执行

$(img).on('load', function () {

// console.log('imgload', this.width)//190

// 图片高度

let imgWidth = ***window***.innerWidth;

// 图片宽度

let imgHeight = (img.height / img.width) \* imgWidth;

***canvas***.height = imgHeight;

// 将背景图画到画板上

ctx.drawImage(img, 0, 0, imgWidth, imgHeight);

// 修改填充颜色

ctx.fillStyle = '#CAA775';

// 设置字体样式，粗体、30px、字体名

ctx.font = "1.8rem nameFont";

let str1 = [];

str1 = text.split('');

***console***.log(str1);

for (var i = 0; i < str1.length; i++) {

// 填充文字，和文字的坐标

ctx.fillText(str1[i], imgWidth \* 0.43, imgHeight \* 0.46+(i\*(28.8/750)\*imgWidth));

}

***console***.log(***canvas***.toDataURL());

const img1 = ***document***.getElementById("img1");

img1.src = ***canvas***.toDataURL();

img1.width = imgWidth;

})

}

</script>

</head>

<body>

<div id="tu">

<img id="img1">

</div>

</canvas>

</body>

</html>

# 下载base64文件

*// 传入base64字符串和文件名下载文件*

      this.downloadFileByBase64(*file*.content, *file*.file.name);

*// 传入base64字符串和文件名下载文件*

    downloadFileByBase64(*base64*, *name*) {

*// 将base64字符串转成Blob类型*

      var myBlob = this.dataURLtoBlob(*base64*);

*// var myUrl = URL.createObjectURL(myBlob);*

      this.downloadFile(myBlob, *name*);

    },

*// 将base64字符串转成Blob类型*

    dataURLtoBlob(*dataurl*) {

*// 以,号分割字符串为数组*

      var arr = *dataurl*.split(","),

*// 获取文件类型，匹配两个字符串:与;中间的字符串且不包含:和;，也就是(:,;)*

        mime = arr[0].match(/(?<=:).\*(?=;)/)[0],

*//获取base64解码后的字符串*

        bstr = atob(arr[1]),

*//获取字符串长度*

        n = bstr.length,

*//创建8 位无符号整数值的类型化数组，长度为n,可以以对象的方式或使用数组下标索引的方式引用数组中的元素*

        u8arr = new Uint8Array(n);

*// 将字符串转换为8 位无符号整数值的类型化数组*

      while (n--) {

*// 获取第n个字符的字符编码*

        u8arr[n] = bstr.charCodeAt(n);

      }

*// 将u8arr合并为一个数组，转换为Blob类型，文件类型为mime*

      return new Blob([u8arr], { type: mime });

    },

*// 传入blob和文件名下载文件*

    downloadFile(

*myBlob*,

*name* = this.$nowdate.format(new Date(), "YYYY-MM-DD hh:mm:ss")

    ) {

*// FileReader主要用于将文件内容读入内存*

*// 创建FileReader对象*

      var reader = new FileReader();

*// 将文件读取为 DataURL*

      reader.readAsDataURL(*myBlob*);

*// onload当读取操作成功完成时调用*

      reader.onload = function (*e*) {

*// 创建a标签*

        var a = document.createElement("a");

*// 设置下载的文件名*

        a.download = *name*;

*// 设置下载的url*

        a.href = *e*.target.result;

*// 添加a标签*

        document.body.appendChild(a);

*// 点击a标签*

        a.click();

*// 释放a标签*

        document.body.removeChild(a);

      };

    },

改成这样也可以

*// 传入blob和文件名下载文件*

    downloadFile(

*myBlob*,

*name* = this.$nowdate.format(new Date(), "YYYY-MM-DD hh:mm:ss")

    ) {

*// 将文件读取为 DataURL*

      var reader = URL.createObjectURL(*myBlob*)

*// 创建a标签*

        var a = document.createElement("a");

*// 设置下载的文件名*

        a.download = *name*;

*// 设置下载的url*

        a.href = reader;

*// 添加a标签*

        document.body.appendChild(a);

*// 点击a标签*

        a.click();

*// 释放a标签*

        document.body.removeChild(a);

*// };*

    },

# VUE项目中DIV生成并导出图片

在项目中安装插件html2canvas

npm install --save html2canvas

模块引入

import html2Canvas from "html2canvas";

导出事件处理

*// 导出图片*

    exportPic(*DivID*) {

      html2Canvas(document.querySelector("#" + *DivID*)).then((*canvas*) => {

        let dataURL = *canvas*.toDataURL("image/png");

        if (dataURL !== "") {

*// 创建a标签*

          var a = document.createElement("a"); *// 设置下载的文件名*

          a.download = "试试"; *// 设置下载的url*

          a.href = dataURL; *// 添加a标签*

          document.body.appendChild(a); *// 点击a标签*

          a.click(); *// 释放a标签*

          document.body.removeChild(a);

        }

      });

    },

# 修改输入框提示文字的颜色

*// 提示文字的颜色*

::-webkit-input-placeholder {

  color: white;

}

:-moz-placeholder {

  color: white;

}

::-moz-placeholder {

  color: white;

}

:-ms-input-placeholder {

  color: white;

}

# 文件上传

<template>

  <div id="upload">

*<!-- 上传文件按钮，点击时调用upload方法 -->*

    <div class="uploadBtn" @click="upload">

*<!-- 文字 -->*

      <span>上传文件</span>

*<!-- 上传文件输入框 -->*

*<!-- file为文件 -->*

*<!-- multiple多文件上传 -->*

*<!-- 可设置accept类型指定文件类型，如accept="image/\*"或accept=".jpg"或accept="image/.png" -->*

*<!-- @change="getFile"当输入框的值改变时调用getFile方法 -->*

      <input type="file" @change="getFile" ref="file" multiple />

    </div>

*<!-- 图片显示 -->*

    <img :src="imgSrc" alt="图片" width="50%" />

*<!-- 正在添加提示 -->*

    <div class="overlay" v-if="isLoading"></div>

    <van-loading size="24px" vertical v-if="isLoading">正在添加...</van-loading>

  </div>

</template>

<script>

export default {

  name: "Upload",

  data() {

    return {

*// 创建formData对象*

      formData: new FormData(),

*// 文件列表*

      files: [],

*// 可支持上传的文件类型*

      SupportType: "image,video",

*// 是否上传，如果没有可上传文件则不上传*

      isUpLoad: false,

*//是否正在上传*

      isLoading:false,

      imgSrc: "",

    };

  },

  methods: {

    upload() {

*// 点击input标签*

      this.$refs.file.click();

    },

    getFile() {

*// 将获取到的文件赋值给this.files*

      this.files = this.$refs.file.files;

*// console.log(this.files);*

*// 创建FileReader用于读取文件*

      const reader = new FileReader();

*//获得图片地址  放在reader.result内*

      reader.readAsDataURL(this.$refs.file.files[0]);

*// 当reader获取完成时调用*

      reader.onload = () => {

*// 将获取到的图片数据赋值给imgSrc*

        this.imgSrc = reader.result;

      };

*// 文件类型*

      let fileType;

*// 文件大小*

      let fileSize;

*// 遍历this.files数组*

      for (let item of this.files) {

*// 获取文件类型*

        fileType = item.type.split("/")[0];

*// 如果支持上传该文件类型*

        if (this.SupportType.indexOf(fileType) != -1) {

*// 获取文件大小，单位mb*

          fileSize = item.size / (1024 \* 1024);

          if (fileSize < 100) {

*// 将文件添加到formData*

            this.formData.append("file", item);

*// 有可上传文件*

            this.isUpLoad = true;

          } else {

            this.$toast("上传的图片或视频应小于100M");

          }

        } else {

          this.$toast("仅支持上传图片或视频");

        }

      }

*// 如果有可上传文件，发送ajax请求*

      if (this.isUpLoad) {

        this.isLoading = true;

*// 设置其他参数*

        this.formData.append("advContent", "");

*// 设置axios的参数*

        var options = {

*// 请求api*

          url: "addAdvContent",

*// 请求体*

          data: this.formData,

*// 请求方式*

          method: "post",

*// 请求头*

          headers: {

            "Content-Type": "multipart/form-data",

          },

        };

*// 发送ajax请求*

        this.$http(options)

          .then((*res*) => {

*// 成功时回调*

            console.log(*res*);

            this.isLoading = false;

            this.$toast("已成功添加");

          })

          .catch((*req*) => {

*// 失败时回调*

            this.isLoading = false;

            this.$toast("添加失败");

            console.log(*req*);

          });

*// 重置isUpLoad*

        this.isUpLoad = false;

      }

    },

  },

};

</script>

<style lang="less" scoped>

#upload {

  position: relative;

  width: 100%;

  height: 100%;

  display: flex;

  flex-direction: column;

  justify-content: center;

  align-items: center;

*// 上传文件按钮*

  .uploadBtn {

    width: 60%;

    height: 6.25rem;

    background-image: linear-gradient(to right, #0080db, #21c392);

    border-radius: 999px;

    display: flex;

    justify-content: center;

    align-items: center;

*// input输入框*

    input {

      display: none;

    }

*// 文本*

    span {

      color: white;

      font-size: 2.5rem;

    }

  }

  .overlay{

    position: absolute;

    top: 0;

    width: 100%;

    height: 100%;

    background-color: rgba(0, 0, 0, .5);

  }

  .van-loading{

    position: absolute;

    top: 50%;

    left: 50%;

    transform: translate(-50%,-50%);

  }

}

</style>

# For in和for of的区别

for…in循环读取键名，for…of循环读取键值

# 项目初始化

## 前端项目初始化步骤

·安装vue脚手架

·通过vue脚手架创建项目

·配置vue路由

·配置element-ui组件库

·配置axios库

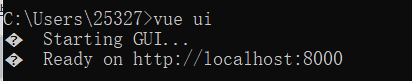
·初始化git远程仓库

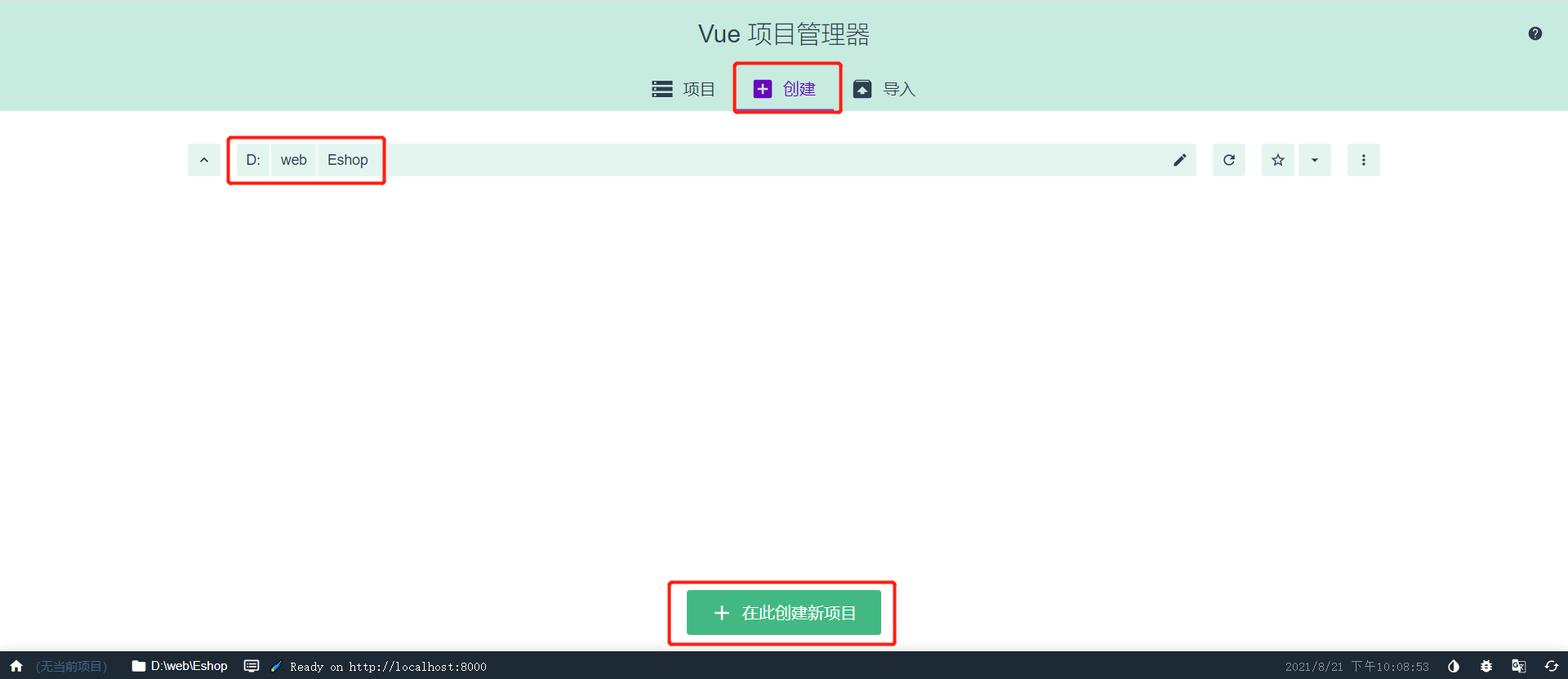
·将本地项目托管到码云中

## 前端项目初始化

·打开vue可视化面板

- vue ui



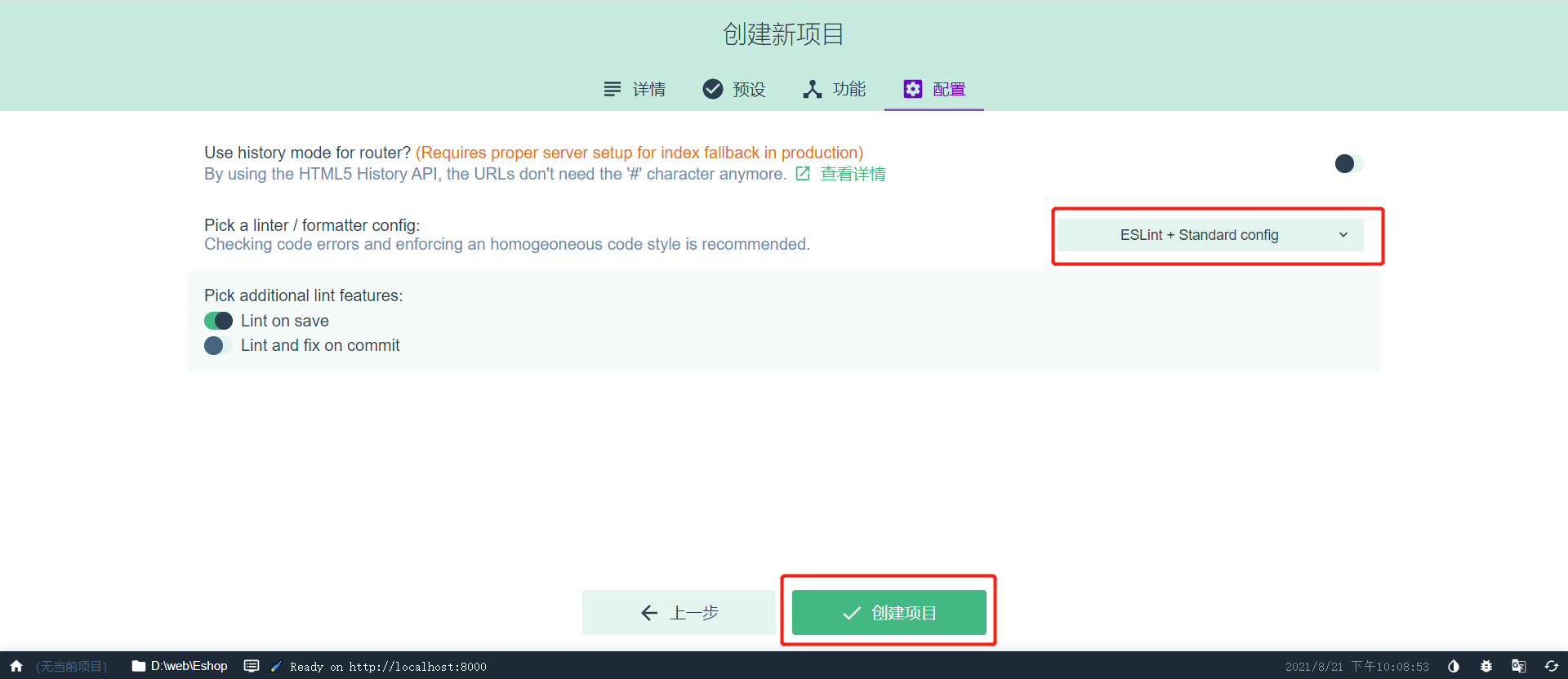






·安装bable、router、linter/formatter、使用配置文件



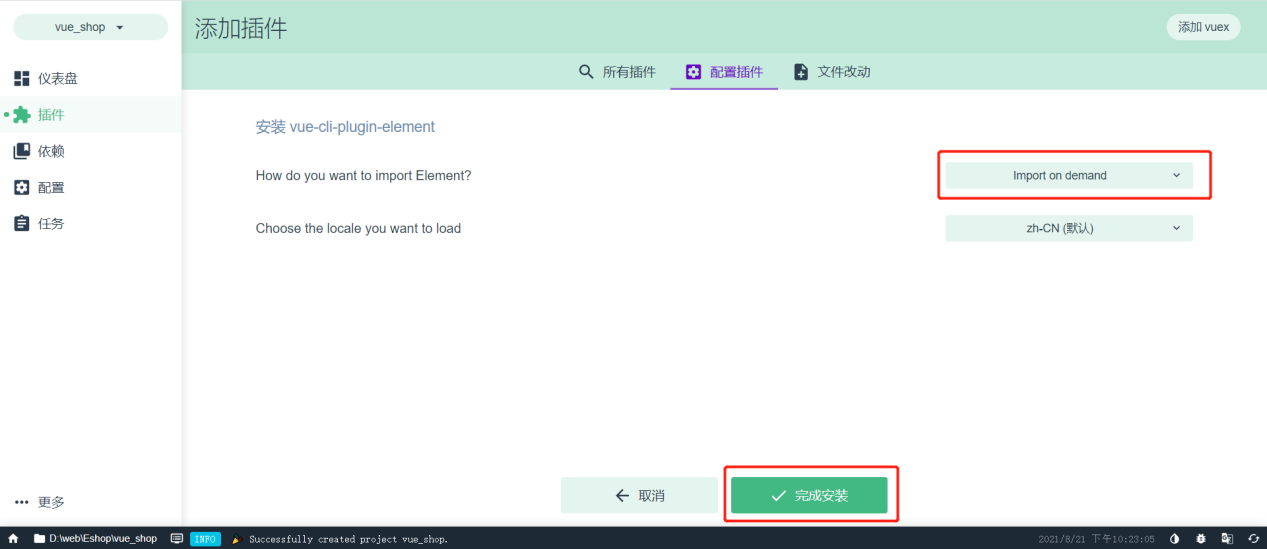




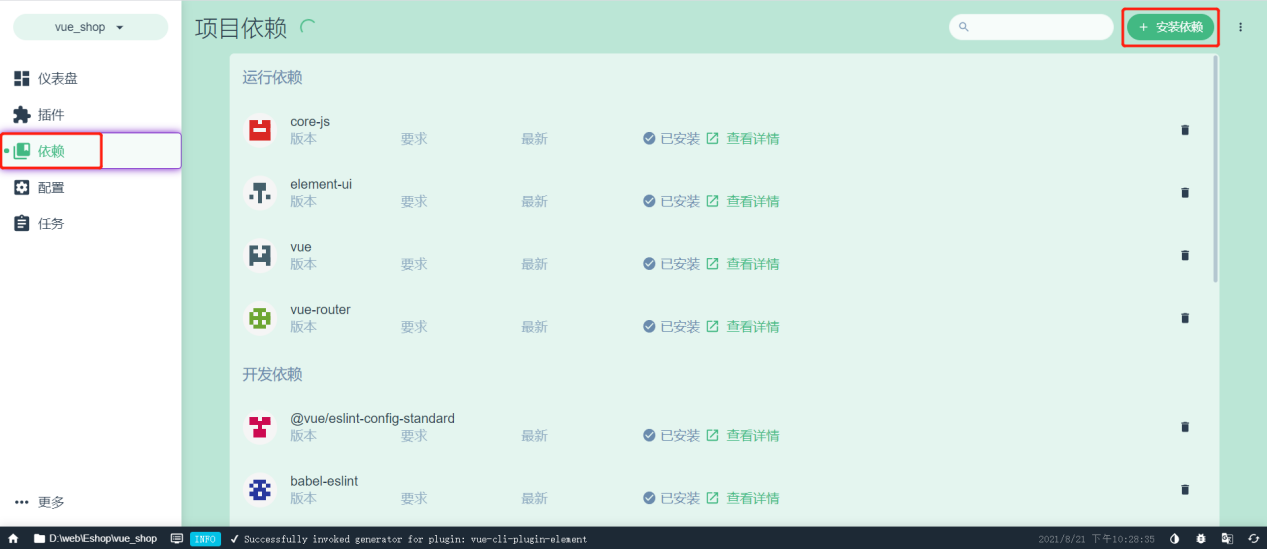
·安装element/vant







·安装axios





·安装less-loader和less







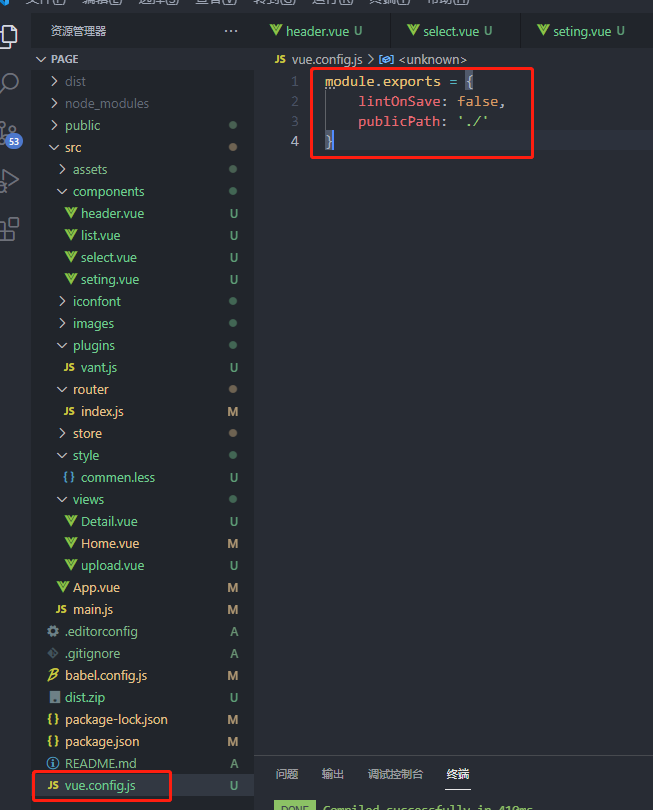
·重新启动项目



·less-loader版本过高可能会报错，建议安装6.0版本

npm install [less-loader@6.0.0](mailto:less-loader@6.0.0)

·vue.config.js



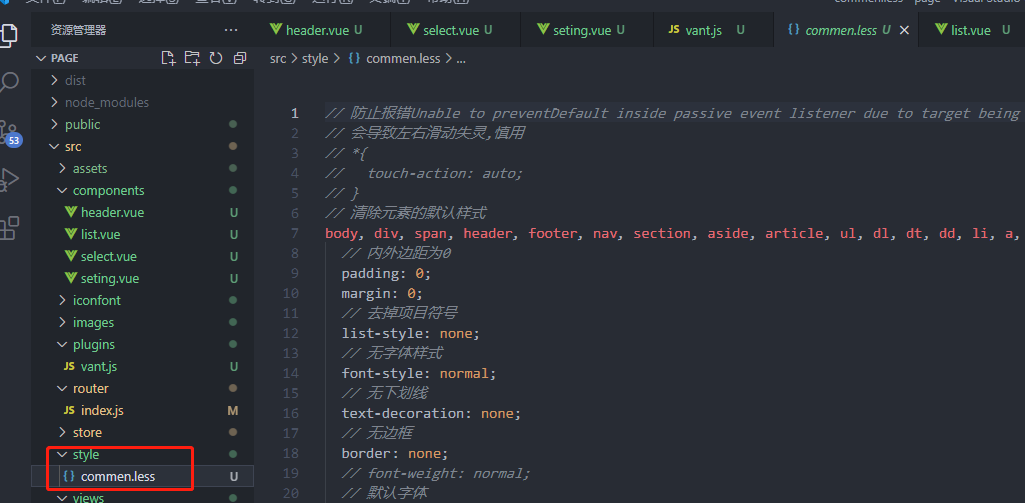
module.exports = {

    lintOnSave: false,

    publicPath: './'

}

·全局样式表



*// 防止报错Unable to preventDefault inside passive event listener due to target being treated as passive*

*// 会导致左右滑动失灵,慎用*

*// \*{*

*//   touch-action: auto;*

*// }*

*// 清除元素的默认样式*

body, div, span, header, footer, nav, section, aside, article, ul, dl, dt, dd, li, a, p, h1, h2, h3, h4,h5, h6, i, b, textarea, button, input, select, figure, figcaption{

*// 内外边距为0*

  padding: 0;

  margin: 0;

*// 去掉项目符号*

  list-style: none;

*// 无字体样式*

  font-style: normal;

*// 无下划线*

  text-decoration: none;

*// 无边框*

  border: none;

*// font-weight: normal;*

*// 默认字体*

  font-family: "Microsoft Yahei", serif;

*// 宽度高度为整个可见框大小*

  box-sizing: border-box;

*// 当点击一个链接或者通过Javascript定义的可点击元素的时候,背景设置为透明,只用于ios*

  -webkit-tap-highlight-color:transparent;

*// 字体抗锯齿渲染*

  -webkit-font-smoothing: antialiased;

*// 鼠标移入时*

  &:hover{

*// 清除轮廓*

    outline: none;

  }

}

html,body,#app{

*// 宽高占满整个屏幕*

  height: 100%;

  width: 100%;

*// 字体大小为2.1333328vw,相当于750px设计图的16px，即在750设计稿中，1rem为16px*

  font-size: 2.1333328vw;

*// font-size: 16px;*

}

*/\* 根据dpr显示2x图/3x图 \*/*

*// .bg-image(@url){*

*//   background-image:~"url('@{url}@2x.png')";*

*//   @media (-webkit-min-device-pixel-ratio: 3),(min-device-pixel-ratio: 3){*

*//     background-image:~"url('@{url}@3x.png')";*

*//   }*

*// }*

*// 清除输入框的边框并移除原生样式*

input,textarea {

  border: 0;

  -webkit-appearance: none;

}

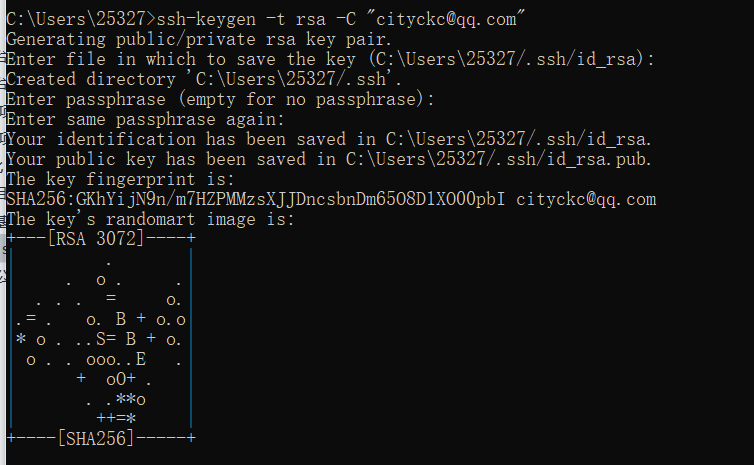
## 码云的基础设置

### 生成ssh公钥的方法

<https://gitee.com/help/articles/4181>

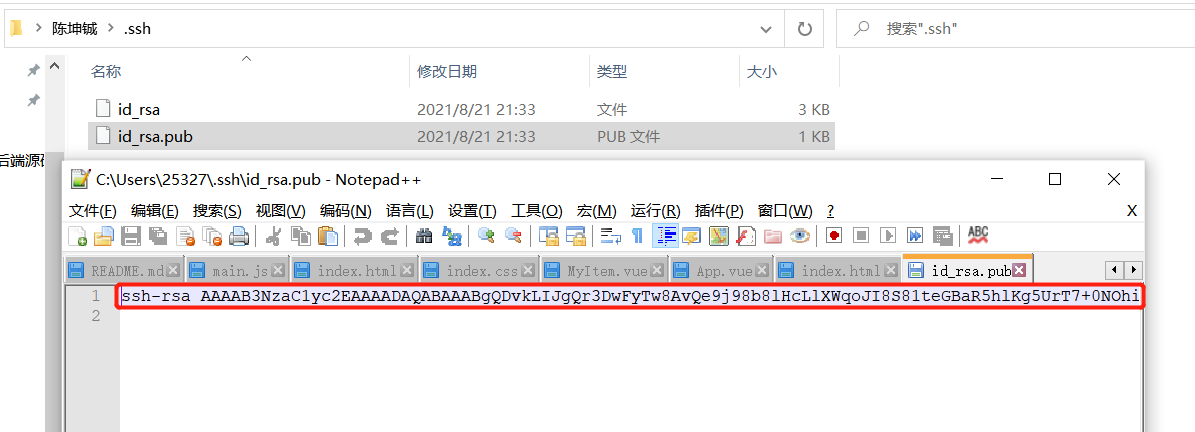
·命令提示符输入ssh-keygen -t rsa -C ["cityckc@qq.com"](mailto:\"cityckc@qq.com\")

·并按三次回车



### ssh公钥设置

·将里面的内容复制

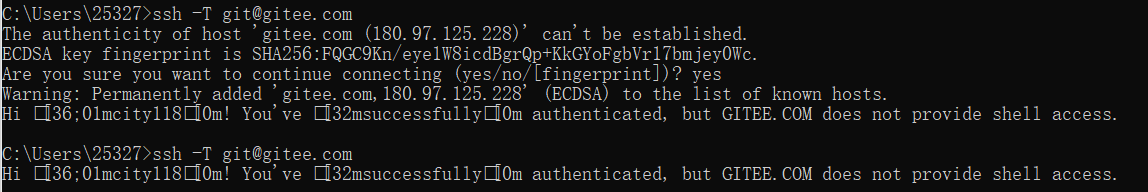


·将内容粘贴到此处，并修改标题即可



### 测试公钥

·ssh -T git@gitee.com



## 将本地项目托管到码云仓库中

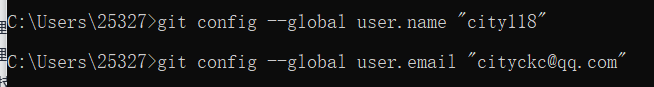
### 新建仓库



### Git 全局设置

git config --global user.name "city118"

git config --global user.email "cityckc@qq.com"



·配置“在此处打开powershell”的方法

<https://blog.csdn.net/weixin_41287692/article/details/105783468>

·在项目文件中打开powershell

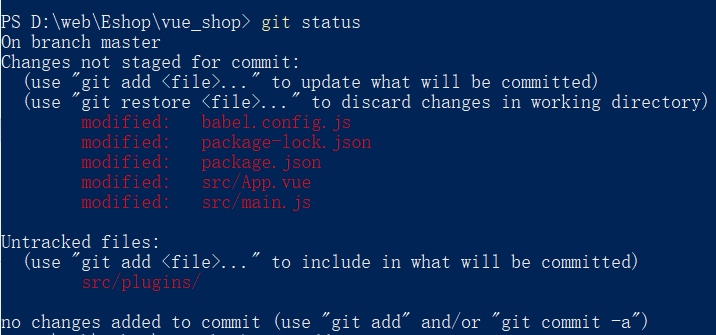


·初始化git

- git init

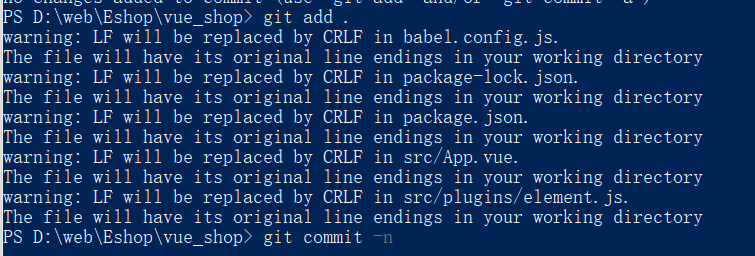
·检测项目的状态

git status



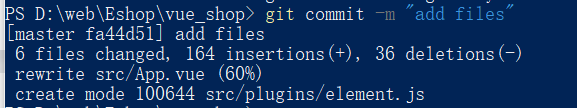
·把所有文件都添加到暂存区

git add .



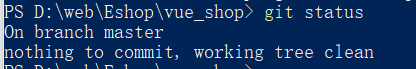
·提交到本地

git commit -m "add files"



·检测项目的状态

git status



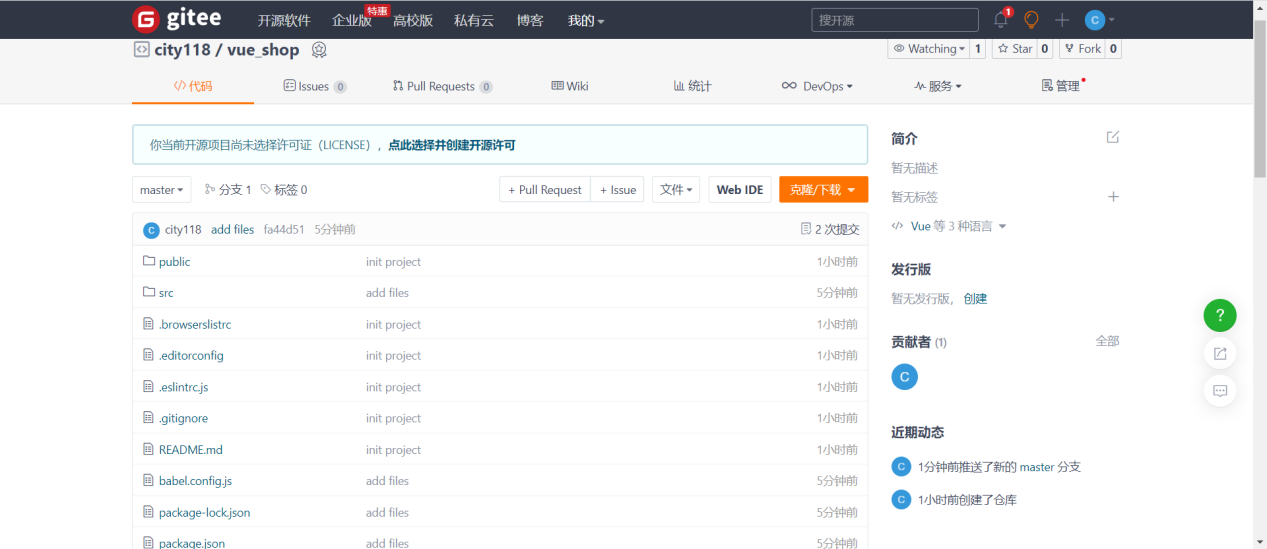
·将项目上传到码云中

git remote add origin <https://gitee.com/city118/vue_shop.git>

git push -u origin master



·上传成功



### 创建git分针

·通过element-ui组件实现布局

- el-form

- el-form-item

- el-input

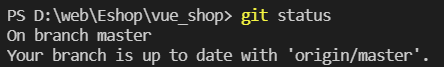
- el-button

- 字体图标

·使用vscode打开项目

·查看工作区是否干净

git status



·创建一个分支

git checkout -b login



·查看所有分支

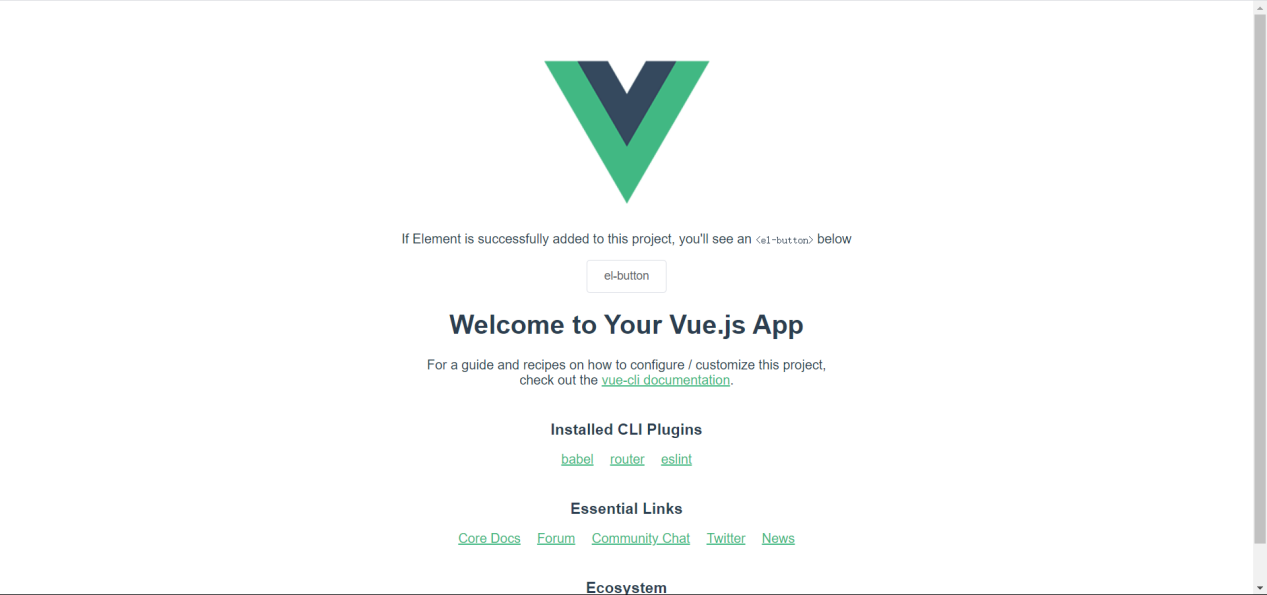
git branch

\*代表当前处于的分支



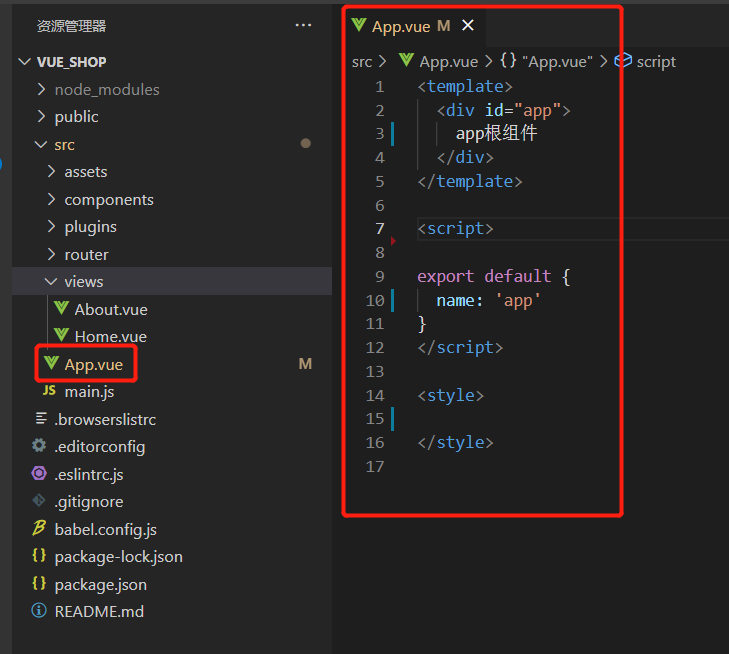
·查看项目的运行效果





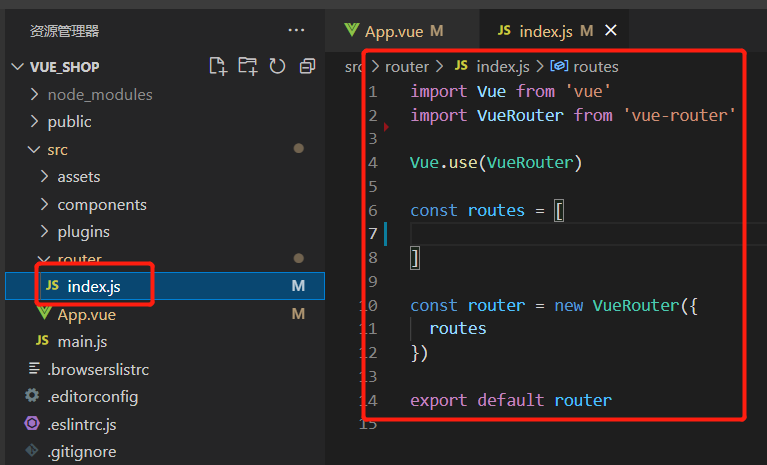
### 删除不必要的文件和布局

·修改vue.js



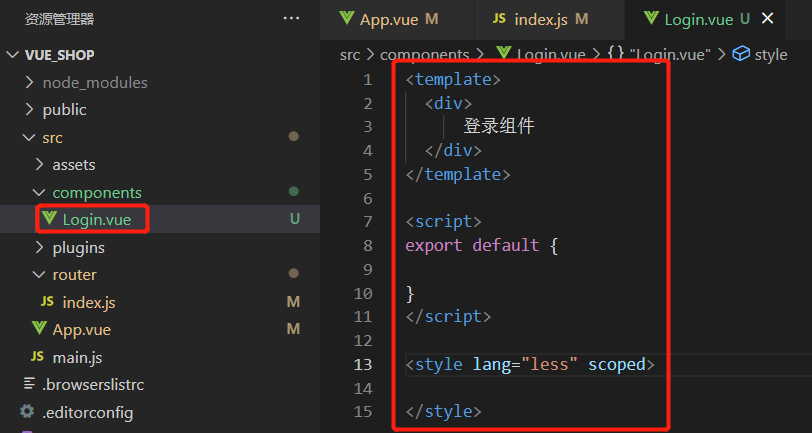
·删除views目录和components的helloWorld.vue

·修改router

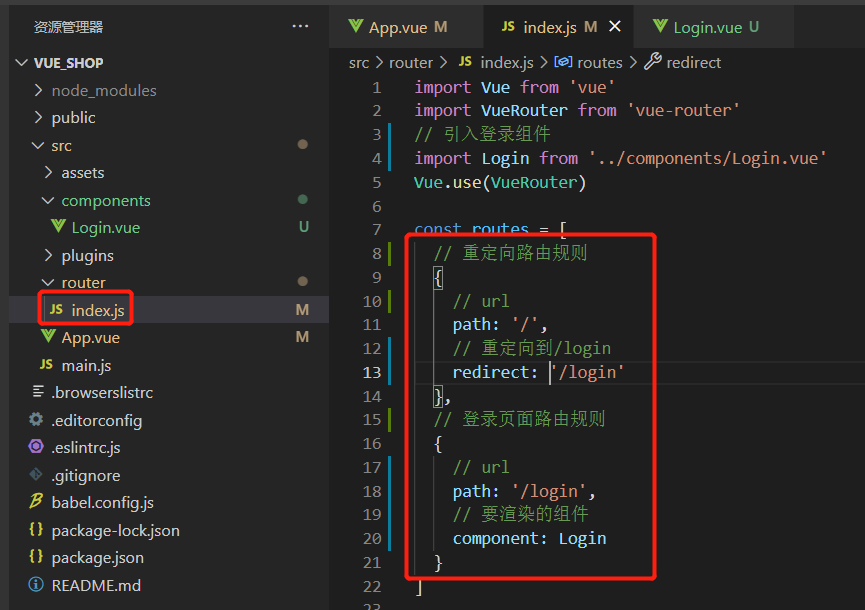


### 登录组件的路由配置

·创建Login.vue



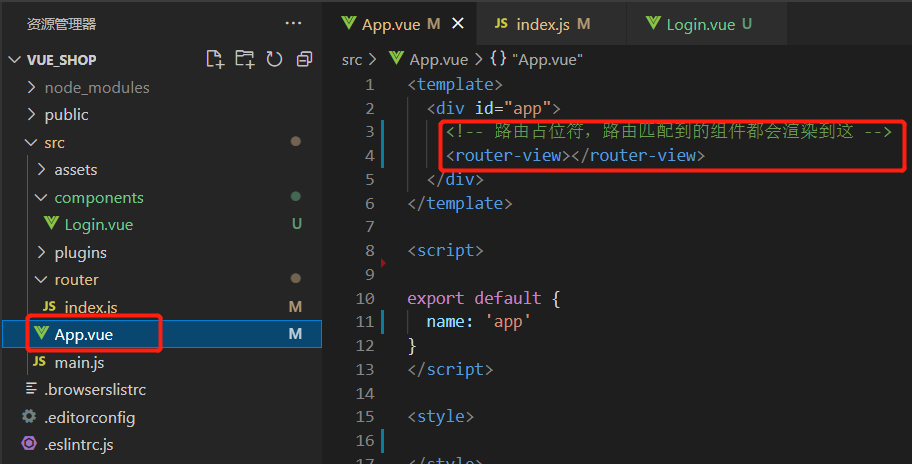
·修改router将Login.vue渲染到页面



·自动修复语法

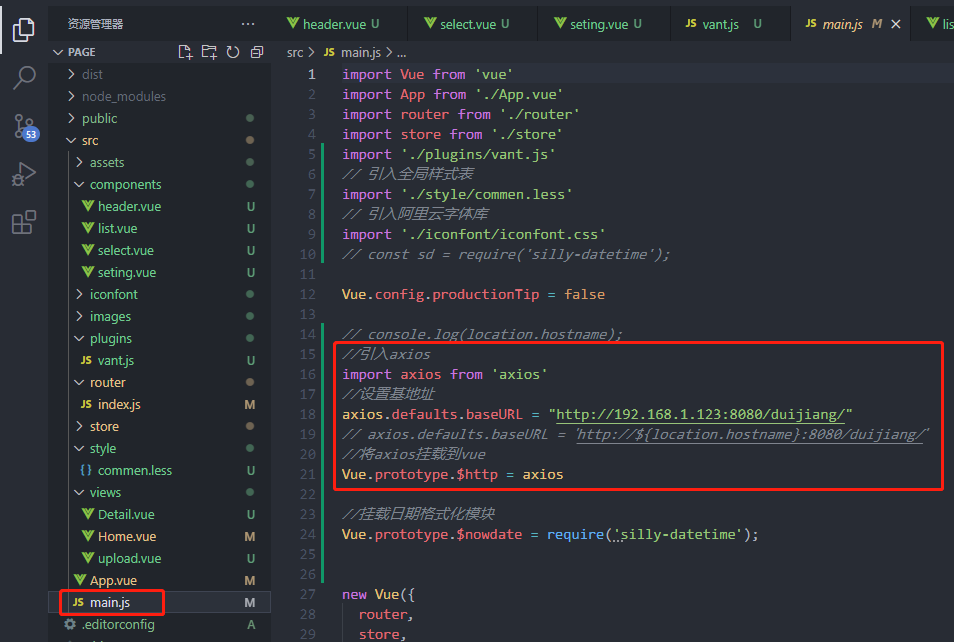
- npm run lint

·修改App.vue



# axios请求api的方法

·引入axios



·请求方式一

    async testAxios() {

      const { data: res } = await this.$http.post("findHuodongInfo", {});

    },

·请求方式二

          this.$http

            .post("delAdvContent", {

              ownId: *ownId*,

            })

            .then((*res*) => {

              console.log(*res*.data);

            })

            .catch((*req*) => {

              console.log(*req*);

            });

·请求方式三

*// 设置axios的参数*

        var options = {

*// 请求api*

          url: "addAdvContent",

*// 请求体*

          data: this.formData,

*// 请求方式*

          method: "post",

*// 请求头*

          headers: {

            "Content-Type": "multipart/form-data",

          },

        };

*// 发送ajax请求*

        this.$http(options)

          .then((*res*) => {

*// 成功时回调*

            console.log(*res*);

          })

          .catch((*req*) => {

*// 失败时回调*

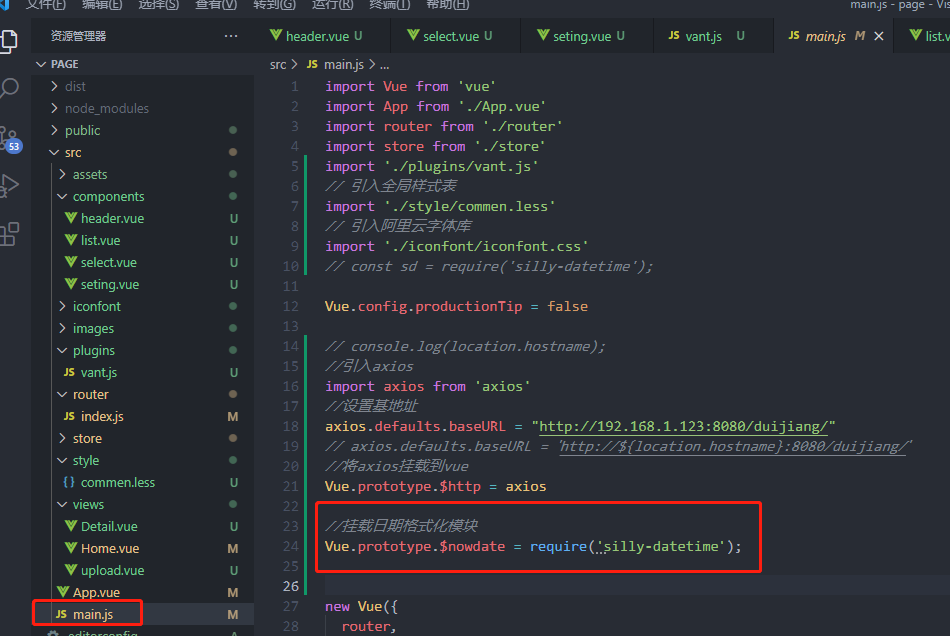
            console.log(*req*);

          });

# 日期格式化模块

·安装silly-datetime模块

- npm install silly-datetime --save



*//挂载日期格式化模块*

Vue.prototype.$nowdate = require('silly-datetime');

*// 获取当前时间为兑奖时间，格式为2021-12-19 10:53:10*

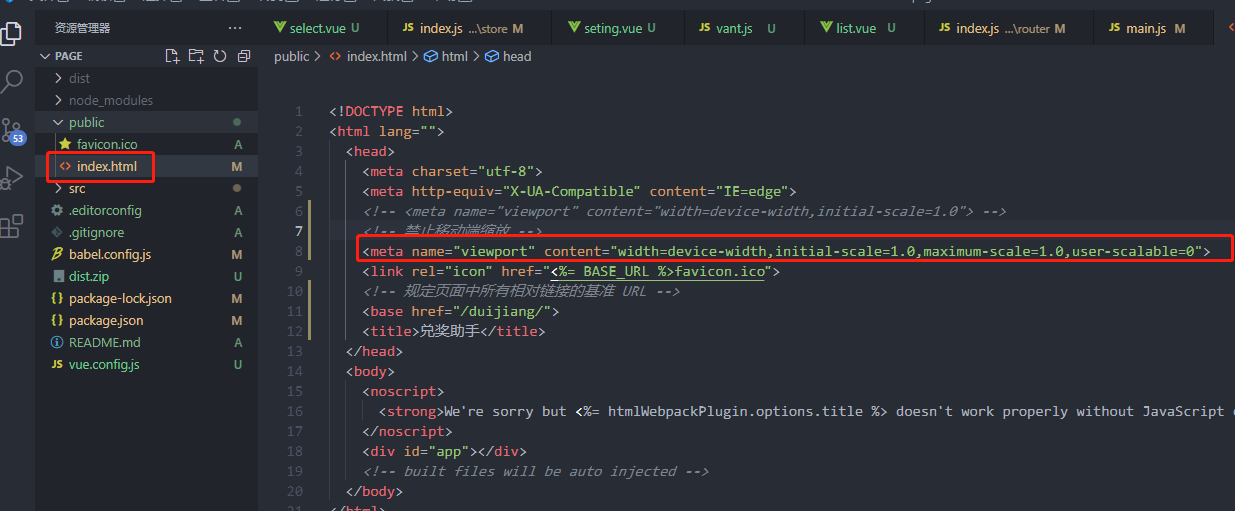
              this.nowDate = this.$nowdate.format(

                new Date(),

                "YYYY-MM-DD hh:mm:ss"

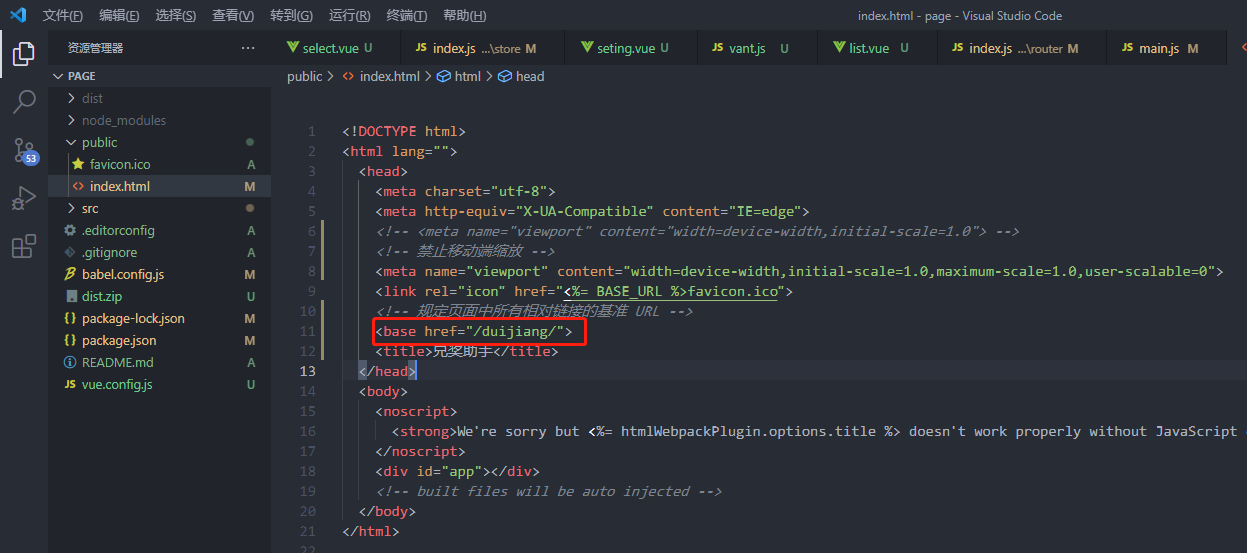
              );

# 禁止移动端缩放



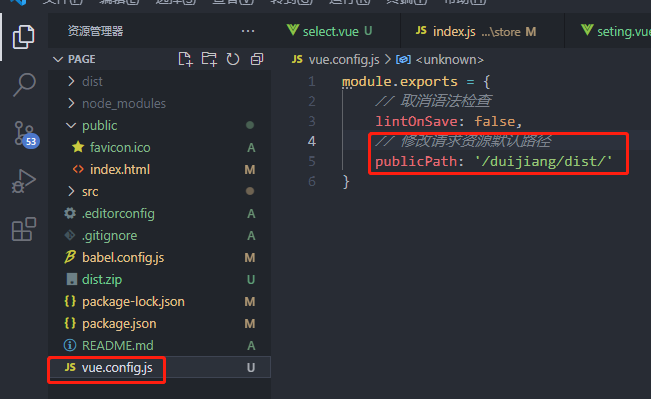
    <meta name="viewport" content="width=device-width,initial-scale=1.0,maximum-scale=1.0,user-scalable=0">

# 规定页面中所有相对链接的基准 URL



    <base href="/duijiang/">

# 修改请求资源默认路径



    publicPath: '/duijiang/dist/'

# 移动端登录页面组件

<template>

*<!-- 登录页面 -->*

  <div id="login">

*<!-- 主内容区 -->*

    <div class="loginMain">

*<!-- 用户名 -->*

      <div class="input">

*<!-- 用户名输入框 -->*

        <input

          type="text"

          v-model="userName"

          @input="setInput($event, 'userName')"

        />

      </div>

*<!-- 密码 -->*

      <div class="input">

*<!-- 密码输入框 -->*

        <input

          :type="isShowPwd ? 'text' : 'password'"

          v-model="passWord"

          @input="setInput($event, 'passWord')"

        />

*<!-- 显示/隐藏按钮 -->*

*<!-- 点击时调用showPwd -->*

        <i

          :class="

            isShowPwd ? 'iconfont icon-zhengyan' : 'iconfont icon-biyanjing'

          "

          @click="showPwd"

        ></i>

      </div>

*<!-- 登录按钮 -->*

      <button class="loginBtn" @click="login">

*<!-- 登录箭头图标 -->*

        <i class="iconfont icon-youjiantou1"></i>

      </button>

    </div>

  </div>

</template>

<script>

export default {

  name: "Login",

  data() {

    return {

*// 是否显示密码*

      isShowPwd: false,

*//   用户名*

      userName: "",

*//   密码*

      passWord: "",

    };

  },

  methods: {

*// 显示/隐藏密码*

    showPwd() {

*// 将this.isShowPwd取反*

      this.isShowPwd = !this.isShowPwd;

*// console.log(this.$refs.password);*

    },

*// 输入框获取焦点时调用，实时监听输入框内容*

    setInput(*e*, *name*) {

*// 最多输入16个字*

      this[name] = e.target.value.slice(0, 16);

    },

*// 登录*

    login() {

*// 判空*

      if (this.userName != "" && this.passWord != "") {

*// 判断是否非汉汉字*

        if (

          this.userName.search(/^[^\u4e00-\u9fa5]{6,}$/) != -1 &&

          this.passWord.search(/^[^\u4e00-\u9fa5]{6,}$/) != -1

        ) {

          console.log("登录");

        } else {

          console.log("不能输入汉字");

        }

      } else {

        console.log("账号密码不能为空！");

      }

    },

  },

};

</script>

<style lang="less" scoped>

*// 此组件的样式*

#login {

*// 将此组件占满整个窗口*

  width: 100%;

  height: 100%;

*// 相对定位*

  position: relative;

*// 背景图片*

*//   background-image: url("../images/background.png");*

*//   // 不重复*

*//   background-repeat: no-repeat;*

*//   // 铺满整个元素*

*//   background-size: cover;*

*//   // 固定到页面，不随滚轮移动*

*//   background-attachment: fixed;*

*//   // 居中显示*

*//   background-position: center;*

*// 主内容区*

  .loginMain {

*// 宽*

    width: 80%;

*// height: 50vh;*

*// background-color: aqua;*

*// 绝对定位*

    position: absolute;

*// 水平垂直居中*

    left: 50%;

    top: 50%;

    transform: translate(-50%, -50%);

*// 弹性盒布局*

    display: flex;

*// 子元素纵向排列*

    flex-direction: column;

*// 子元素水平居中*

    align-items: center;

*// 输入框容器*

    .input {

*// 宽高*

      width: 100%;

      height: 6.25rem;

*// 渐变色背景*

      background-image: linear-gradient(to right, #0080db, #21c392);

*// 底部外边距*

      margin-bottom: 1.25rem;

*// 圆角*

      border-radius: 999px;

*// 相对定位*

      position: relative;

*// 输入框*

      input {

*// 宽高*

        width: 100%;

        height: 100%;

*// 渐变色背景*

        background-image: linear-gradient(to right, #0080db, #21c392);

*// 圆角*

        border-radius: 999px;

*// 字体颜色*

        color: white;

*// 字体大小*

        font-size: 2.5rem;

*// 字体居中*

        text-align: center;

*// 左右内边距*

        padding: 0 3.125rem;

      }

*// 字体图标样式*

      .iconfont {

*// 图标颜色*

        color: white;

        font-size: 2rem;

*// 绝对定位*

        position: absolute;

*// 距父元素右边*

        right: 3.125rem;

*// 垂直居中*

        top: 50%;

        transform: translateY(-50%);

      }

    }

*// 登陆按钮*

    .loginBtn {

*// 宽高*

      width: 7.5rem;

      height: 7.5rem;

*// 渐变色背景*

      background-image: linear-gradient(to right, #0080db, #21c392);

*// 顶部外边距*

      margin-top: 3.125rem;

*// 圆角*

      border-radius: 50%;

*// 右箭头图标*

      .icon-youjiantou1 {

*// 图标颜色*

        color: white;

*// 图标大小*

        font-size: 3.75rem;

      }

    }

  }

}

</style>

# 获取文件资源

require("../images/tuan1.png"),

# 元素水平垂直居中

## 弹性盒flex

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>测试</title>

</head>

<style>

html,body{

margin: 0;

padding: 0;

width: 100%;

height: 100%;

display: flex;

justify-content: center;

align-items: center;

}

.ceshi{

width: 100px;

height: 100px;

background-color: aquamarine;

}

</style>

<body>

<div class="ceshi"></div>

</body>

</html>

## -webkit-box

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>测试</title>

</head>

<style>

html,body{

margin: 0;

padding: 0;

width: 100%;

height: 100%;

display:-webkit-box;

-webkit-box-align: center;

-webkit-box-pack: center;

/\*display: flex;\*/

/\*justify-content: center;\*/

/\*align-items: center;\*/

}

.ceshi{

width: 100px;

height: 100px;

background-color: aquamarine;

}

</style>

<body>

<div class="ceshi"></div>

</body>

</html>

## margin:auto法

·脱离文档流

·这种方式只适用于大小确定的元素

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>测试</title>

</head>

<style>

html,body{

margin: 0;

padding: 0;

width: 100%;

height: 100%;

position: relative;

}

.ceshi{

width: 100px;

height: 100px;

background-color: aquamarine;

position: absolute;

margin: auto;

left: 0;

right: 0;

top: 0;

bottom: 0;

}

</style>

<body>

<div class="ceshi"></div>

</body>

</html>

## translate负50%法

·脱离文档流

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>测试</title>

</head>

<style>

html,body{

margin: 0;

padding: 0;

width: 100%;

height: 100%;

position: relative;

}

.ceshi{

width: 100px;

height: 100px;

background-color: aquamarine;

position: absolute;

top: 50%;

left: 50%;

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

}

</style>

<body>

<div class="ceshi"></div>

</body>

</html>

## table-cell

·未脱离文档流

·父元素宽高需确定

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8">

<title>测试</title>

</head>

<style>

html,body{

margin: 0;

padding: 0;

width: 100%;

height: 100%;

display: table-cell;

vertical-align: middle;

text-align: center;

}

.ceshi{

width: 500px;

height: 500px;

background-color: aquamarine;

display: table-cell;

vertical-align: middle;

text-align: center;

}

.d1{

width: 100px;

height: 80px;

background-color: #0086b3;

margin: 0 auto;

}

</style>

<body>

<div class="ceshi">

<div class="d1"></div>

</div>

</body>

</html>

# 元素拖拽drag

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8">

<title>测试</title>

</head>

<style>

html, body {

margin: 0;

padding: 0;

width: 100%;

height: 100%;

}

.tuo {

width: 100px;

height: 100px;

display: inline-block;

}

#one{

background-color: aqua;

}

#two{

background-color: blueviolet;

}

.drop {

width: 500px;

height: 500px;

border: 1px solid #000;

}

</style>

<body>

<div class="box">

<div id="one" draggable="true" class="tuo"></div>

<div id="two" draggable="true" class="tuo"></div>

<div class="drop"></div>

</div>

<script>

// 获取父元素

var ***box*** = ***document***.getElementsByClassName('box')[0];

// 获取可拖入元素

var ***drop*** = ***document***.getElementsByClassName('drop')[0];

// 当子元素的可拖动元素开始拖动时触发

***box***.ondragstart = function (e) {

***console***.log('开始拖动');

// 设置被拖动数据的数据类型和值

e.dataTransfer.setData('id',e.target.id);

}

// 当子元素的可拖动元素正在拖动时触发

***box***.ondrag = function (e) {

***console***.log('正在拖动',e.target.id);

}

// 当子元素的可拖动元素拖动结束时触发

***box***.ondragend = function (e) {

***console***.log('拖动结束');

}

// 可拖动元素进入可拖入元素时触发

***drop***.ondragenter = function (e) {

***console***.log('进入可拖入元素');

// 阻止默认事件，使元素可拖入其他元素

e.preventDefault();

}

// 可拖动元素在可拖入元素拖动时触发

***drop***.ondragover = function (e) {

***console***.log('在可拖入元素拖动');

// 阻止默认事件，使元素可拖入其他元素

e.preventDefault();

}

// 可拖动元素离开可拖入元素时触发

***drop***.ondragleave = function (e) {

***console***.log('离开可拖入元素');

// 阻止默认事件，使元素可拖入其他元素

e.preventDefault();

}

// 可拖动元素放入可拖入元素时触发

***drop***.ondrop = function (e) {

***console***.log('放入可拖入元素');

// 阻止默认事件，使元素可拖入其他元素

e.preventDefault();

// 获得拖动元素的数据

let data = e.dataTransfer.getData("id");

// 将可拖动元素变为可拖入元素的子元素

e.target.appendChild(***document***.getElementById(data));

}

</script>

</body>

</html>

# js实现主动使input失去焦点

      var input = document.getElementById("your-input-id");

      input.blur();

# input标签事件总结（vue、H5）

## 一、在Html中input标签的相关事件写法：

1、onfocus（常用）

input标签获取焦点事件

2、onblur（常用）

input失去焦点事件（触发条件：先获取焦点，再失去焦点触发）

3、onchange

input失去焦点并且它的value值发生变化时触发

4、oninput

input框输入过程中value值改变时实时触发，及没输入一个字符都会触发

5、onclick

input标签type="button"时的点击事件

6、onkeydown

input框输入时键盘按钮按下事件

7、onkeyup

input框输入时键盘按钮抬起事件，触发onkeyup事件之前一定触发onkeydown事件

8、onselect

input标签内容选中时触发事件

写法：

1、标签内写法：

1640222612(1)

2、js绑定事件写法：

1640222663(1)

## 二、vue中监听input标签事件

vue中监听iinput标签事件的写法与H5+js的写法有些不同。

1、input实时监听事件

在vue中，原生input标签（这里不说UI框架封装过的input标签）监听value值实时变化的事件为v-on:input方法

1640222722(1)

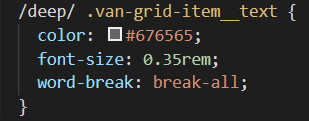
2、input获取焦点事件：

1640222781(1)

1. input失去焦点事件：

1640222815(1)

# vue 修改ui框架自带的样式



# 设置圆角按钮

  border-radius: 999px;

# 正则表达式大全

## 一、校验数字的表达式

1. 数字：^[0-9]\*$

2. n位的数字：^\d{n}$

3. 至少n位的数字：^\d{n,}$

4. m-n位的数字：^\d{m,n}$

5. 零和非零开头的数字：^(0|[1-9][0-9]\*)$

6. 非零开头的最多带两位小数的数字：^([1-9][0-9]\*)+(.[0-9]{1,2})?$

7. 带1-2位小数的正数或负数：^(\-)?\d+(\.\d{1,2})?$

8. 正数、负数、和小数：^(\-|\+)?\d+(\.\d+)?$

9. 有两位小数的正实数：^[0-9]+(.[0-9]{2})?$

10. 有1~3位小数的正实数：^[0-9]+(.[0-9]{1,3})?$

11. 非零的正整数：^[1-9]\d\*$ 或 ^([1-9][0-9]\*){1,3}$ 或 ^\+?[1-9][0-9]\*$

12. 非零的负整数：^\-[1-9][]0-9"\*$ 或 ^-[1-9]\d\*$

13. 非负整数：^\d+$ 或 ^[1-9]\d\*|0$

14. 非正整数：^-[1-9]\d\*|0$ 或 ^((-\d+)|(0+))$

15. 非负浮点数：^\d+(\.\d+)?$ 或 ^[1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*|0?\.0+|0$

1. 非正浮点数：^((-\d+(\.\d+)?)|(0+(\.0+)?))$

或 ^(-([1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*))|0?\.0+|0$

1. 正浮点数：^[1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*$

或 ^(([0-9]+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*))$

1. 负浮点数：^-([1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*)$

或 ^(-(([0-9]+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*)))$

1. 浮点数：^(-?\d+)(\.\d+)?$

或 ^-?([1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*|0?\.0+|0)$

## 二、校验字符的表达式

1. 汉字：^[\u4e00-\u9fa5]{0,}$

2. 英文和数字：^[A-Za-z0-9]+$ 或 ^[A-Za-z0-9]{4,40}$

3. 长度为3-20的所有字符：^.{3,20}$

4. 由26个英文字母组成的字符串：^[A-Za-z]+$

5. 由26个大写英文字母组成的字符串：^[A-Z]+$

6. 由26个小写英文字母组成的字符串：^[a-z]+$

7. 由数字和26个英文字母组成的字符串：^[A-Za-z0-9]+$

8. 由数字、26个英文字母或者下划线组成的字符串：^\w+$ 或 ^\w{3,20}$

9. 中文、英文、数字包括下划线：^[\u4E00-\u9FA5A-Za-z0-9\_]+$

10. 中文、英文、数字但不包括下划线等符号：^[\u4E00-\u9FA5A-Za-z0-9]+$ 或 ^[\u4E00-\u9FA5A-Za-z0-9]{2,20}$

11. 可以输入含有^%&',;=?$\"等字符：[^%&',;=?$\x22]+ 12 禁止输入含有~的字符：[^~\x22]+

## 三、特殊需求表达式

1. Email地址：^\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*$

2. 域名：[a-zA-Z0-9][-a-zA-Z0-9]{0,62}(/.[a-zA-Z0-9][-a-zA-Z0-9]{0,62})+/.?

3. InternetURL：[a-zA-z]+://[^\s]\* 或 ^http://([\w-]+\.)+[\w-]+(/[\w-./?%&=]\*)?$

1. 手机号码：

^(13[0-9]|14[01456879]|15[0-35-9]|16[2567]|17[0-8]|18[0-9]|19[0-35-9])\d{8}$

5. 电话号码("XXX-XXXXXXX"、"XXXX-XXXXXXXX"、"XXX-XXXXXXX"、"XXX-XXXXXXXX"、"XXXXXXX"和"XXXXXXXX)：^(\(\d{3,4}-)|\d{3.4}-)?\d{7,8}$

6. 国内电话号码(0511-4405222、021-87888822)：\d{3}-\d{8}|\d{4}-\d{7}

7. 身份证号(15位、18位数字)：^\d{15}|\d{18}$

8. 短身份证号码(数字、字母x结尾)：^([0-9]){7,18}(x|X)?$ 或 ^\d{8,18}|[0-9x]{8,18}|[0-9X]{8,18}?$

9. 帐号是否合法(字母开头，允许5-16字节，允许字母数字下划线)：^[a-zA-Z][a-zA-Z0-9\_]{4,15}$

10. 密码(以字母开头，长度在6~18之间，只能包含字母、数字和下划线)：^[a-zA-Z]\w{5,17}$

11. 强密码(必须包含大小写字母和数字的组合，不能使用特殊字符，长度在8-10之间)：^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{8,10}$

12. 日期格式：^\d{4}-\d{1,2}-\d{1,2}

13. 一年的12个月(01～09和1～12)：^(0?[1-9]|1[0-2])$

14. 一个月的31天(01～09和1～31)：^((0?[1-9])|((1|2)[0-9])|30|31)$

15. 钱的输入格式：

1. 有四种钱的表示形式我们可以接受:"10000.00" 和 "10,000.00", 和没有 "分" 的 "10000" 和 "10,000"：^[1-9][0-9]\*$
2. 这表示任意一个不以0开头的数字,但是,这也意味着一个字符"0"不通过,所以我们采用下面的形式：^(0|[1-9][0-9]\*)$
3. 一个0或者一个不以0开头的数字.我们还可以允许开头有一个负号：^(0|-?[1-9][0-9]\*)$

4.这表示一个0或者一个可能为负的开头不为0的数字.让用户以0开头好了.把负号的也去掉,因为钱总不能是负的吧.下面我们要加的是说明可能的小数部分：^[0-9]+(.[0-9]+)?$

5.必须说明的是,小数点后面至少应该有1位数,所以"10."是不通过的,但是 "10" 和 "10.2" 是通过的：^[0-9]+(.[0-9]{2})?$

6.这样我们规定小数点后面必须有两位,如果你认为太苛刻了,可以这样：^[0-9]+(.[0-9]{1,2})?$

7.这样就允许用户只写一位小数.下面我们该考虑数字中的逗号了,我们可以这样：^[0-9]{1,3}(,[0-9]{3})\*(.[0-9]{1,2})?$

8.1到3个数字,后面跟着任意个 逗号+3个数字,逗号成为可选,而不是必须：^([0-9]+|[0-9]{1,3}(,[0-9]{3})\*)(.[0-9]{1,2})?$

9.备注：这就是最终结果了,别忘了"+"可以用"\*"替代如果你觉得空字符串也可以接受的话(奇怪,为什么?)最后,别忘了在用函数时去掉那个反斜杠,一般的错误都在这里

16. xml文件：^([a-zA-Z]+-?)+[a-zA-Z0-9]+\\.[x|X][m|M][l|L]$

17. 中文字符的正则表达式：[\u4e00-\u9fa5]

18. 双字节字符：[^\x00-\xff] (包括汉字在内，可以用来计算字符串的长度(一个双字节字符长度计2，ASCII字符计1))

19. 空白行的正则表达式：\n\s\*\r (可以用来删除空白行)

20. HTML标记的正则表达式：<(\S\*?)[^>]\*>.\*?</\1>|<.\*? /> (网上流传的版本太糟糕，上面这个也仅仅能部分，对于复杂的嵌套标记依旧无能为力)

21. 首尾空白字符的正则表达式：^\s\*|\s\*$或(^\s\*)|(\s\*$) (可以用来删除行首行尾的空白字符(包括空格、制表符、换页符等等)，非常有用的表达式)

22. 腾讯QQ号：[1-9][0-9]{4,} (腾讯QQ号从10000开始)

23. 中国邮政编码：[1-9]\d{5}(?!\d) (中国邮政编码为6位数字)

24. IP地址：\d+\.\d+\.\d+\.\d+ (提取IP地址时有用)

1. IP地址：

((?:(?:25[0-5]|2[0-4]\\d|[01]?\\d?\\d)\\.){3}(?:25[0-5]|2[0-4]\\d|[01]?\\d?\\d))

1. 匹配两个字符串A与B中间的字符串包含A与B：

表达式: A.\*?B（“.“表示任意字符，“？”表示匹配0个或多个）

示例: Abaidu.comB

结果: Abaidu.comB

1. 匹配两个字符串A与B中间的字符串包含A但是不包含B：

表达式: A.\*?(?=B)

示例: Awww.apizl.comB

结果: Awww.apizl.com

1. 匹配两个字符串A与B中间的字符串且不包含A与B：

表达式: (?<=A).\*?(?=B)

示例: Awww.apizl.comB

结果: [www.apizl.com](http://www.apizl.com)

# Vue页面生成二维码

给定一个网址，在页面生成二维码，手机扫码后直接进入此网站。

## 安装全局的插件

npm install qrcodejs2 -S

## qr.vue组件



<template>

    <div>

        <div id="qrcode"></div> *<!-- 创建一个div，并设置id为qrcode -->*

    </div>

</template>

<script>

*// 二维码*

    import QRCode from 'qrcodejs2'  *// 引入qrcode*

    export default {

        name: 'qr',

        mounted() {

*// 调用qrcode方法*

            this.qrcode();

        },

*// props传参*

        props:{

*// 宽*

            width:{

*// 数据类型*

                type:Number,

*// 默认值*

                default(){

                    return 132

                }

            },

*// 高*

            height:{

*// 数据类型*

                type:Number,

*// 默认值*

                default(){

                    return 132

                }

            },

*// 二维码地址*

            url:{

*// 数据类型*

                type:String,

*// 默认值*

                default(){

                    return 'https://www.baidu.com'

                }

            }

        },

        methods: {

*// 生成页面二维码*

*// 第一个参数是要添加到父元素的id*

            qrcode() {

                let qrcode = new QRCode('qrcode', {

                    width: this.width,

                    height: this.height,

                    text: this.url,

                    colorDark: "#000",

                    colorLight: "#fff",

                })

            },

        }

    }

</script>

<style scoped>

</style>

## Qrcode.vue组件

<template>

  <div id="qrcode">

*<!-- props传参 -->*

    <qrcode :url="url" :width="width" :height="height"></qrcode>

  </div>

</template>

<script>

import qrcode from "@/components/qr";

export default {

  data() {

    return {

      width: 100,

      height: 100,

*// 获取当前页面二维码并赋值给url*

      url: location.href,

    };

  },

  components: {

    qrcode,

  },

};

</script>

<style scoped>

</style>

# 几个很实用的BOM属性对象方法?

·location对象

- location.href

-- 返回或设置当前文档的URL

- location.search

-- 返回URL中的查询字符串部分。

-- 例如 http://www.dreamdu.com/dreamdu.php?id=5&name=dreamdu

-- 返回?id=5&name=dreamdu

- location.hash

-- 返回URL#后面的内容

-- 如果没有#，返回空

- location.host

-- 返回URL中的域名部分

-- 例如www.dreamdu.com

- location.hostname

-- 返回URL中的主域名部分

-- 例如dreamdu.com

- location.pathname

-- 返回URL的域名后的部分

-- 例如 http://www.dreamdu.com/xhtml/

-- 返回/xhtml/

- location.port

-- 返回URL中的端口部分

-- 例如 <http://www.dreamdu.com:8080/xhtml/>

-- 返回8080

- location.protocol

-- 返回URL中的协议部分

-- 例如 http://www.dreamdu.com:8080/xhtml/

-- 返回http:

- location.assign

-- 设置当前文档的URL

-- 可实现网页跳转

- location.replace()

-- 设置当前文档的URL

-- 并且在history对象的地址列表中移除这个URL

-- location.replace(url);

- location.reload()

-- 重载当前页面

·history对象

- history.go()

-- 前进或后退指定的页面数

-- history.go(2);

- history.back()

-- 后退一页

- history.forward()

-- 前进一页

·Navigator对象

- navigator.userAgent

-- 返回用户代理头的字符串表示(包含浏览器版本信息)

- navigator.cookieEnabled

-- 返回浏览器是否支持(启用)cookie

# background-image样式和img标签如何选择

## background-image的优势：

·节省img标签所占用的空间

·便于布局（例如将文字布局在图片上，直接使用padding调整即可，使用margin会导致边距冲突）

## 什么情况下应该使用img标签：

·由于搜索引擎不搜索css，所以当图片可以帮助网页获取流量时，应该使用img标签，且同时设置alt属性对图像进行说明

·装饰性的图片就使用background-img，如果和文体内容很相关就使用img

# 解决移动端click事件延迟300ms和点击穿透问题

## 方式一：禁用缩放

*<!-- 禁止移动端缩放 -->*

    <meta name="viewport" content="width=device-width,initial-scale=1.0,maximum-scale=1.0,user-scalable=0">

## 方式二： fastclick

安装fastclick

npm install fastclick --save

在main.js里面添加

import  FastClick  from  'fastclick'

# 切换组件时缓存数据

·添加keep-alive标签

    <keep-alive>

      <List></List>

    </keep-alive>

# 全局事件总线和vuex

## 全局事件总线

所有组件都可以利用 $on和$emit在$bus属性上绑定方法

通过方法参数可以在不同组件传递数据

适用于任意组件中的通信

缺点：

数据本身存放在某一组件内部，然后其他的组件通过触发事件回调，来进行数据的修改。

如果我们要实现组件通信，就必须在子组件中写一个方法来触发父组件中的事先绑定好的回调函数。

如果很多组件要操作此数据时，会变得繁琐。

## Vuex

定义：将数据作为全局变量，统一放在某个地方进行统一管理。

优点：

集中管理共享的数据，便于开发和后期进行维护

高效的实现组件之间的数据共享，提高开发效率

数据是响应式的，当数据发生改变时，页面中的数据也会同步更新

缺点：

数据需要经过,Actions或者Mutations来修改，如果项目比较简单使用vuex就显得有些繁琐。

## 如何选择

大项目或多组件操作数据时用Vuex,小项目或较少组件操作数据时用事件总线

# Axios添加拦截器

*// 添加请求拦截器*

axios.interceptors.request.use(function (*config*) {

*// 在发送请求之前做些什么*

  return *config*

}, function (*error*) {

*// 对请求错误做些什么*

  return Promise.reject(*error*)

});

*// 添加响应拦截器*

axios.interceptors.response.use(function (*response*) {

*// 对响应数据做点什么*

  return *response*

}, function (*error*) {

*// 对响应错误做点什么*

  return Promise.reject(*error*)

});

# scoped下修改子组件样式

vue组件中，在style设置为scoped的时候，里面在写样式对子组件是不生效的，如果想让某些样式对所以子组件都生效，可以使用 /deep/ 深度选择器。

·使用/deep/深度选择器

        /deep/ .van-uploader\_\_input-wrapper {

          width: 100%;

        }

# H5主内容区屏幕自适应高度

1.设置:style="`height: ${listHeight}rem;`"

  <div class="ListWrp" :style="`height: ${listHeight}rem;`">

1. 初始化宽高

  mounted () {

    // 获取hasduijiang为this.$store.state.select的后台数据

    this.initData(this.$store.state.select)

    // 获取屏幕宽高

    const screenWidth = document.body.clientWidth

    const screenHeight = document.body.clientHeight

    // 得到兑奖列表的高度，750为设计稿宽度，16为根目录字体大小(单位px)，15.25为头部区+底部区高度(单位rem)

    this.listHeight = ((screenHeight / screenWidth) \* 750) / 16 - 15.25

    // 当窗口大小改变时调用

    window.onresize = this.getListHeight

  },

1. 响应式宽高

  methods: {

    getListHeight () {

      // 获取屏幕宽高

      const screenWidth = document.body.clientWidth

      const screenHeight = document.body.clientHeight

      // 得到兑奖列表的高度，750为设计稿宽度，16为根目录字体大小(单位px)，15.25为头部区+底部区高度(单位rem)

      this.listHeight = ((screenHeight / screenWidth) \* 750) / 16 - 15.25

},

}

# vue路由传参

## 编程式路由传参

### 通过 params 传递

路径参数用冒号 : 表示。

const routes = [

  // 动态段以冒号开始

  { path: 'details/:id', name: "details", component: Details },

]

router.push() 方法的参数可以是一个字符串路径，或者一个描述地址的对象。

// 字符串路径

this.$router.push('/details/001')

// 带有路径的对象

this.$router.push({ path: '/details/001' })

// 命名路由，路由配置时，需要 name 字段

this.$router.push({ name: 'details', params: { id: '001' } })

注意，如果提供了 path，params 会被忽略：

// `params` 不能与 `path` 一起使用

router.push({ path: '/details', params: { id: '001' } }) // -> /details

组件获取参数，当一个路由被匹配时，它的 params 的值将在每个组件中以 this.$route.params 的形式暴露出来。

this.$route.params.id

const Details = {

  template: '<div>Details {{ $route.params.id }} </div>',

  created() {

    // 监听路由变化

    this.$watch(

      () => this.$route.params,

      (toParams, previousParams) => {

        // 对路由变化做出响应...

      }

    )

  },

}

### 通过 query 传递

这种情况下 query （查询参数）传递的参数会显示在 url 后面，如：/details/001?kind=car

使用 query 时，以下三种方式都是可行的：

this.$router.push('/details/001?kind=car')

this.$router.push({ path: '/details/001', query: { kind: "car" }})

this.$router.push({ name: 'details', params: { id: '001' }, query: { kind: 'car' }})

组件通过 $route.query 获取：

this.$route.query.kind

const Details = {

  template: '<div>Details {{ $route.query.kind }} </div>',

  created() {

    // 监听路由变化

    this.$watch(

      () => this.$route.query,

      (toParams, previousParams) => {

        // 对路由变化做出响应...

      }

    )

  },

}

### 通过 hash 传递

通过此方式，url 路径中带有 hash，例如：/details/001#car

使用 hash 时，以下三种方式都是可行的（同 query）：

this.$router.push('/details/001#car')

this.$router.push({ path: '/details/001', hash: '#car'})

this.$router.push({ name: 'details', params: { id: '001' }, hash: 'car'})

组件通过 $route.hash.slice(1) 获取：

this.$route.hash.slice(1)

## 通过 props 进行传递

在组件中使用 $route 会与路由紧密耦合，这限制了组件的灵活性，因为它只能用于特定的 URL。虽然这不一定是件坏事，但我们可以通过 props 配置来解除这种行为

以解耦的方式使用 props 进行参数传递，主要是在路由配置中进行操作

### 布尔模式

当 props 设置为 true 时，route.params 将被设置为组件的 props。

const User = {

  props: ['id'], // 组件中通过 props 获取 id

  template: '<div>User {{ id }}</div>'

}

// 路由配置中，增加 props 字段，并将值 设置为 true

const routes = [{ path: '/user/:id', component: User, props: true }]

注意：对于有命名视图的路由，你必须为每个命名视图定义 props 配置：

const routes = [

  {

    path: '/user/:id',

    components: { default: User, sidebar: Sidebar },

    // 为 User 提供 props

    props: { default: true, sidebar: false }

  }

]

### 对象模式

当 props 是一个对象时，它将原样设置为组件 props。当 props 是静态的时候很有用

const routes = [

  {

    path: '/hello',

    component: Hello,

    props: { name: 'World' }

  }

]

组件中获取数据

const Hello = {

  props: {

    name: {

      type: String,

      default: 'Vue'

    }

  },

  template: '<div> Hello {{ name }}</div>'

}

<Hello /> 组件默认显示 Hello Vue，但路由配置了 props 对象，当路由跳转到 /hello 时，会显示传递过来的 name， 页面会显示为 Hello World

### 函数模式

可以创建一个返回 props 的函数。这允许你将参数转换为其他类型，将静态值与基于路由的值相结合等等

使用函数模式时，返回 props 的函数接受的参数为路由记录 route

// 创建一个返回 props 的函数

const dynamicPropsFn = (route) => {

  return { name: route.query.say + "!" }

}

const routes = [

  {

    path: '/hello',

    component: Hello,

    props: dynamicPropsFn

  }

]

组件获取参数，当 URL 为 /hello?say=World 时， 将传递 {name: 'World!'} 作为 props 传给 Hello 组件

const Hello = {

  props: {

    name: {

      type: String,

      default: 'Vue'

    }

  },

  template: '<div> Hello {{ name }}</div>'

}

注意：请尽可能保持 props 函数为无状态的，因为它只会在路由发生变化时起作用。如果你需要状态来定义 props，请使用包装组件，这样 vue 才可以对状态变化做出反应

## 其他方式

### 通过 Vuex 进行传递

### 通过前端本地存储等方式

    1. Local Storage;

    2. Session Storage;

    3. IndexedDB;

    4. Web SQL;

    5. Cookies。

# 实时监听窗口大小

## 方式一：

  mounted() {

    // 设置监听窗口大小事件，当窗口大小改变时回调handleResize方法

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

  },

  methods: {

    handleResize() {

      this.screenWidth = document.body.clientWidth; //获取窗口宽度

      this.screenHeight = document.body.clientHeight;//获取窗口高度

},

}

## 方式二：

  mounted () {

    // 当窗口大小改变时调用

    window.onresize = this.getListHeight

  },

  methods: {

    getListHeight () {

      // 获取屏幕宽高

      const screenWidth = document.body.clientWidth

      const screenHeight = document.body.clientHeight

},

}

# 实时监听滚轮事件

  mounted() {

    // 设置监听滚动事件，当滚轮时回调handleScroll方法

    window.addEventListener("scroll", this.handleScroll);

  },

  methods: {

    handleScroll() {

      this.scrollTop =

        document.body.scrollTop || document.documentElement.scrollTop; //获取滚轮到顶部的距离兼容浏览器

},

}

# Ajax 请求到底应该放在 created 里还是 mounted 里

放在哪差别并不大，但是在 mounted 里整体逻辑会更清晰一点。

如果将 API 请求放到 created 里的话，实际上是这样一个过程：

created => API请求 => 获取数据 => 组件重新渲染

            => mounted => 组件首次渲染

也就是说，再发送 API 请求以后，就会产生 2 个分支，代码逻辑比较混乱。

放在 mounted 里的话，逻辑就会清晰很多

created => mounted =>  组件首次渲染 => API请求 => 获取到数据 => 组件 重新渲染

# 资源获取优化

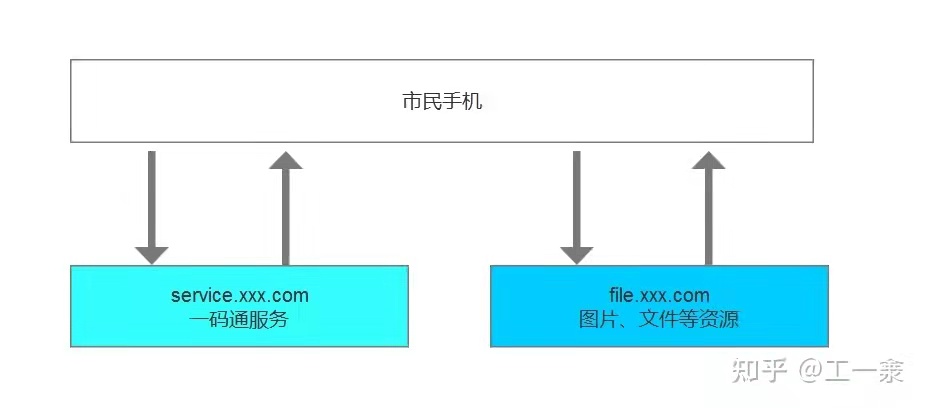
若是高频访问的页面出现1MB的图片，本身就是产品设计缺陷，正常来说，为提高访问速度，图片是要做多种格式的压缩的，至少提供缩略图、原图两种形式。

在总览页面一般展示缩略图，以减少带宽，并提升加载速度；点击详情再展示原图。这是图片展示的通用做法。

从系统的整体设计来说，通常图片、文件、视频等资源与系统应用应该进行流量隔离，现在的产品通常都采用微服务架构，前后端分离。后端通常返回的都是json数据，数据包相对资源文件小得多。

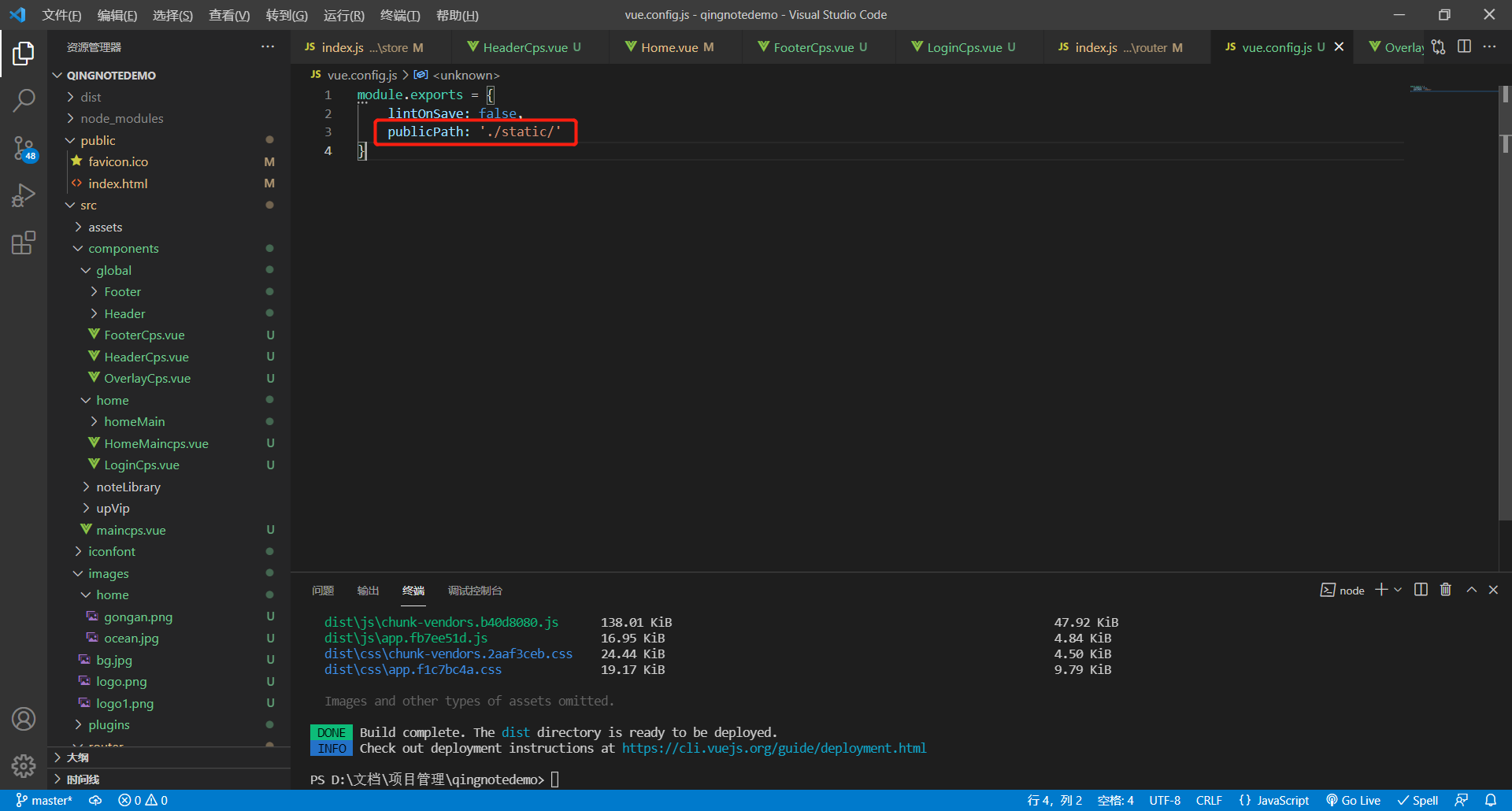
上线系统一般会将图片等资源映射到单独的子域名，若是担心带宽或容量问题，更好的选择可使用成熟的第三方文件系统，例如阿里云、七牛云等。

若是问题中提到的码是指健康码的话，一般不建议在服务器端生成，建议在本地直接生成图片即可，可大幅降低服务器资源的占用。

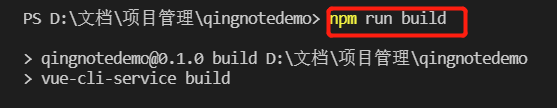


# linux服务器宝塔面板中部署node服务

·vue配置中修改资源基路径



·vue打包项目



·新建文件夹并用vscode打开

·npm init命令

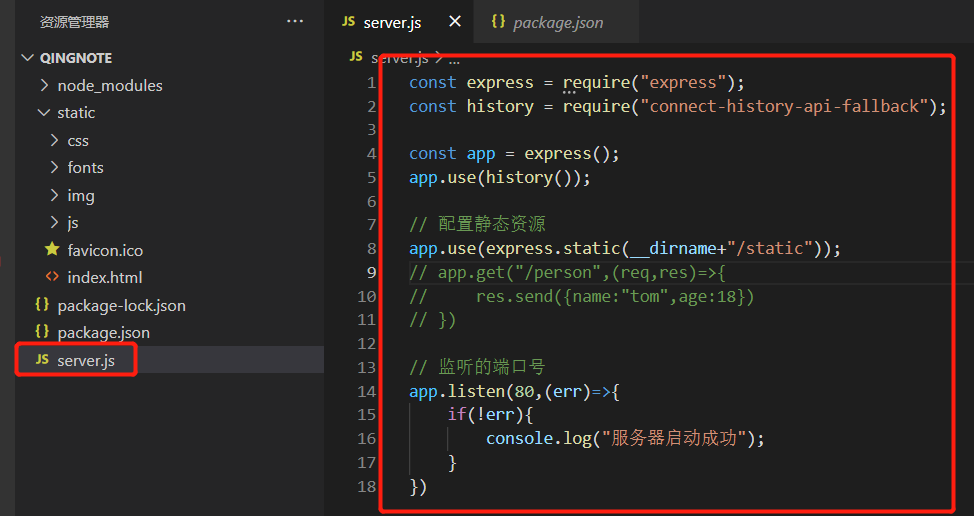
·package name为xxx\_sever

·其他的直接敲回车

·npm i express命令安装express

·npm install --save connect-history-api-fallback 命令安装history插件

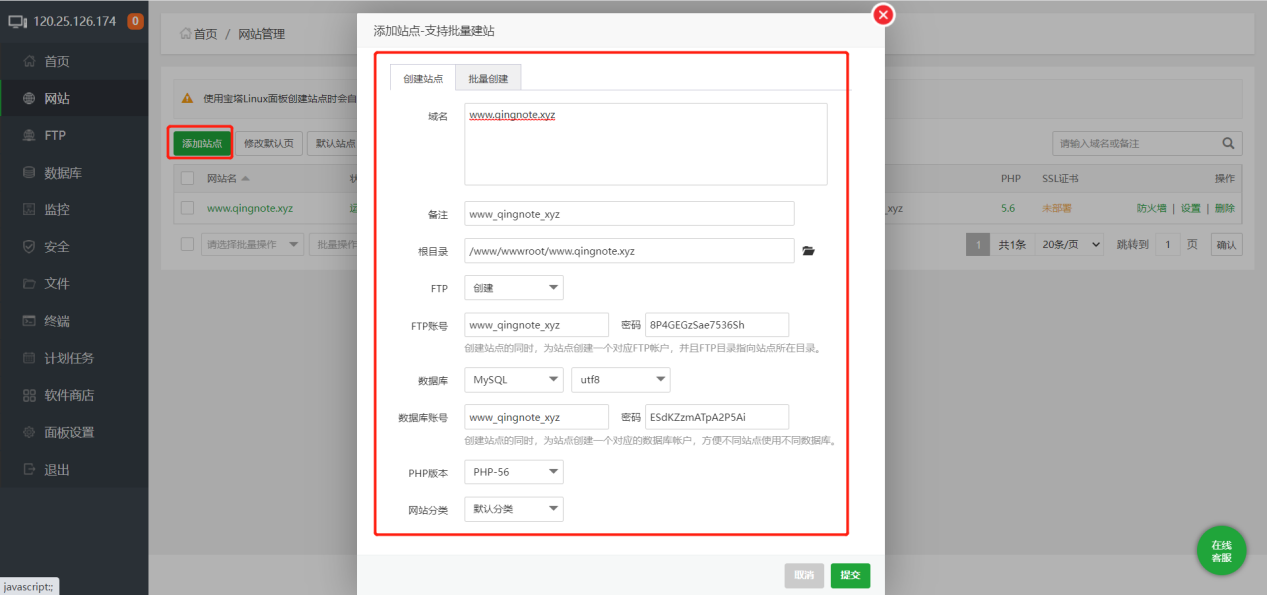
·如图配置即可



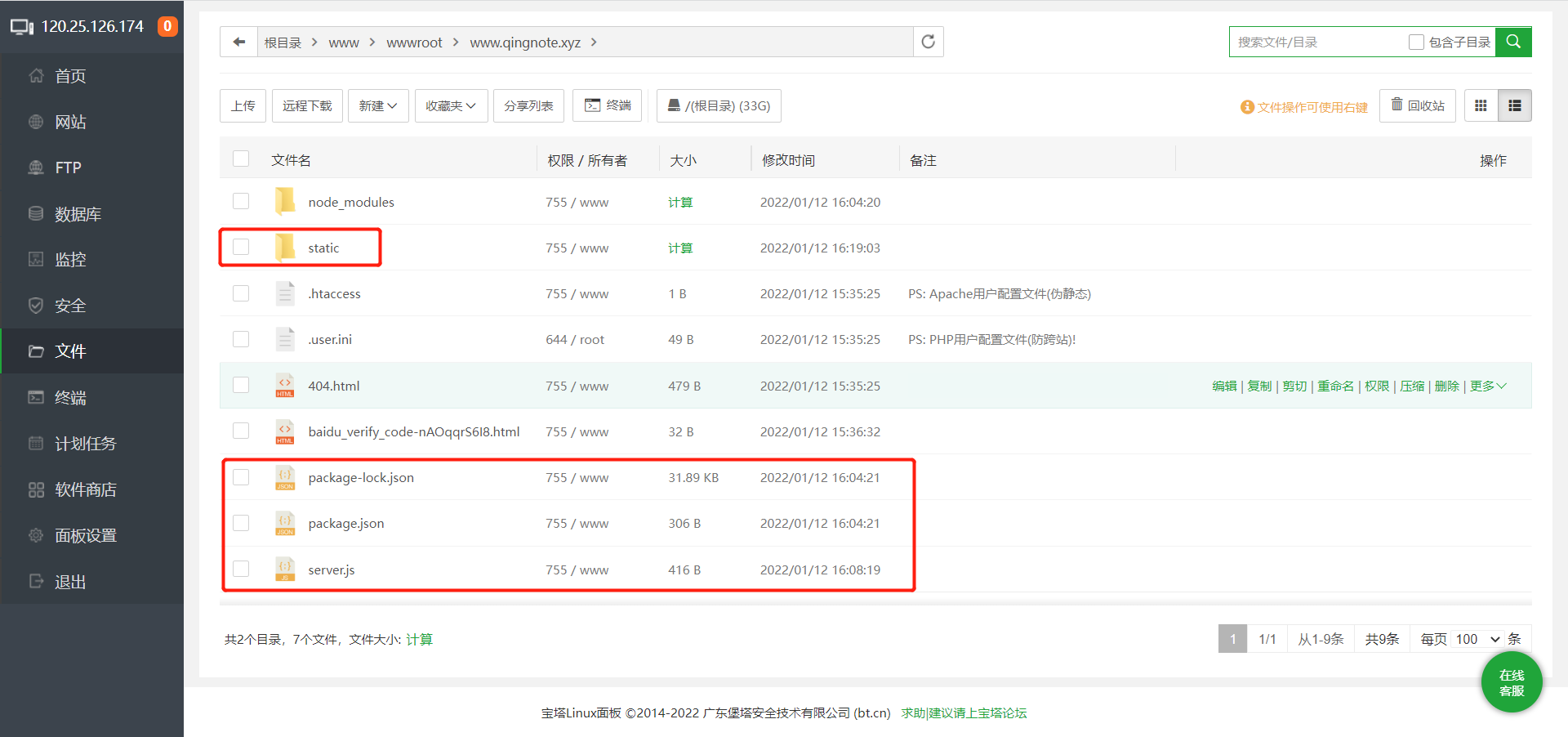
·宝塔面板安装pm2管理器



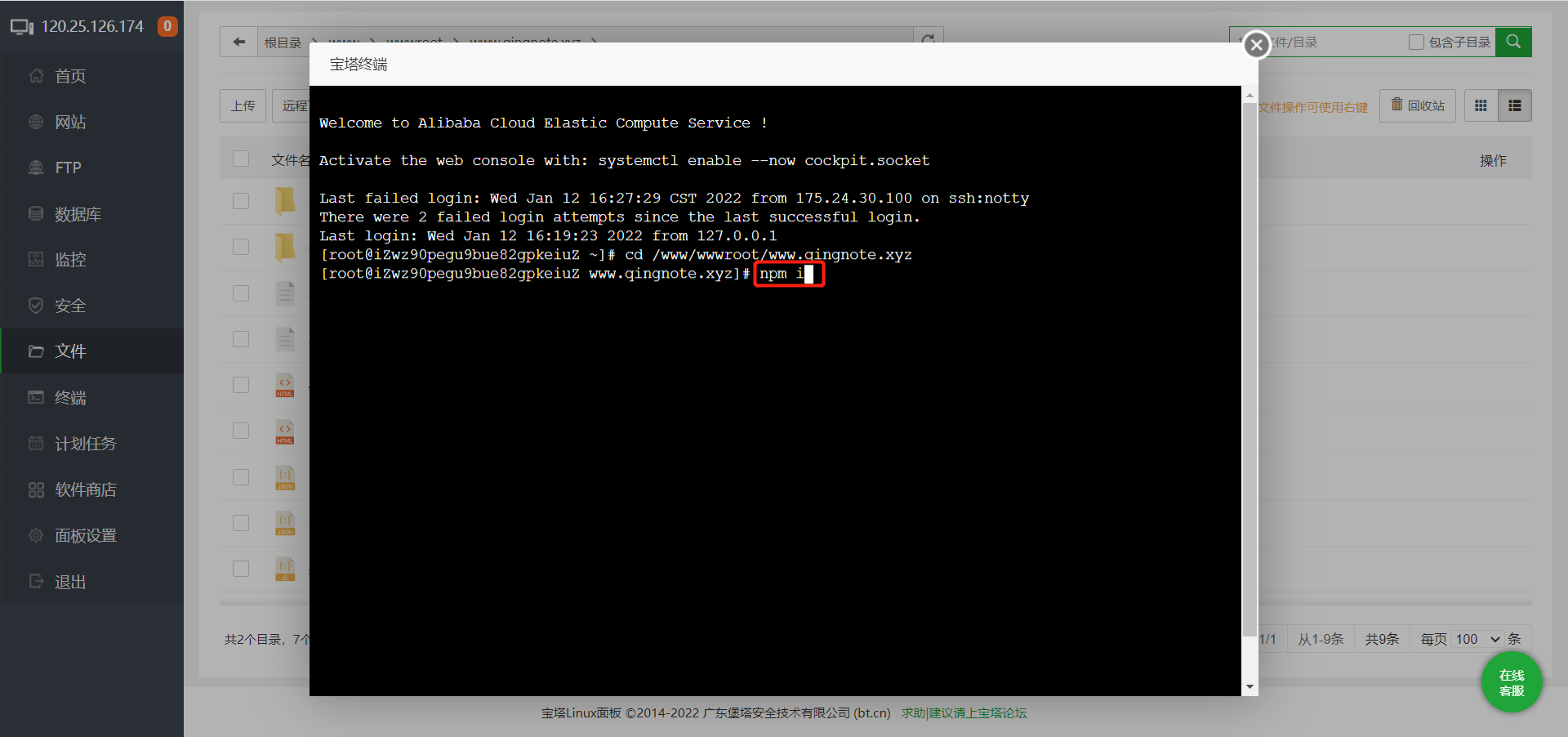
·添加站点



·将node.js项目文件放到站点根目录，除依赖文件



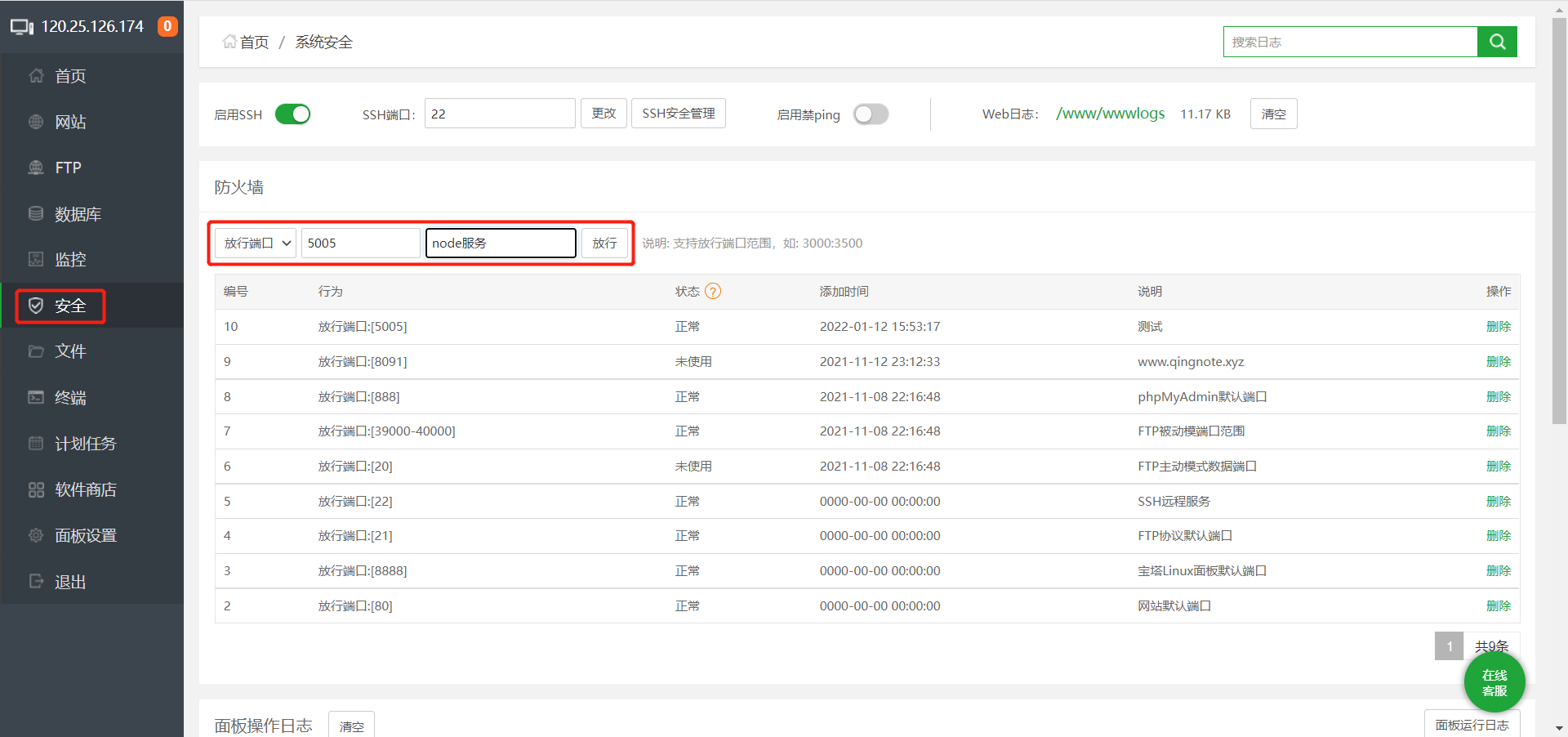
·打开终端，npm i安装依赖



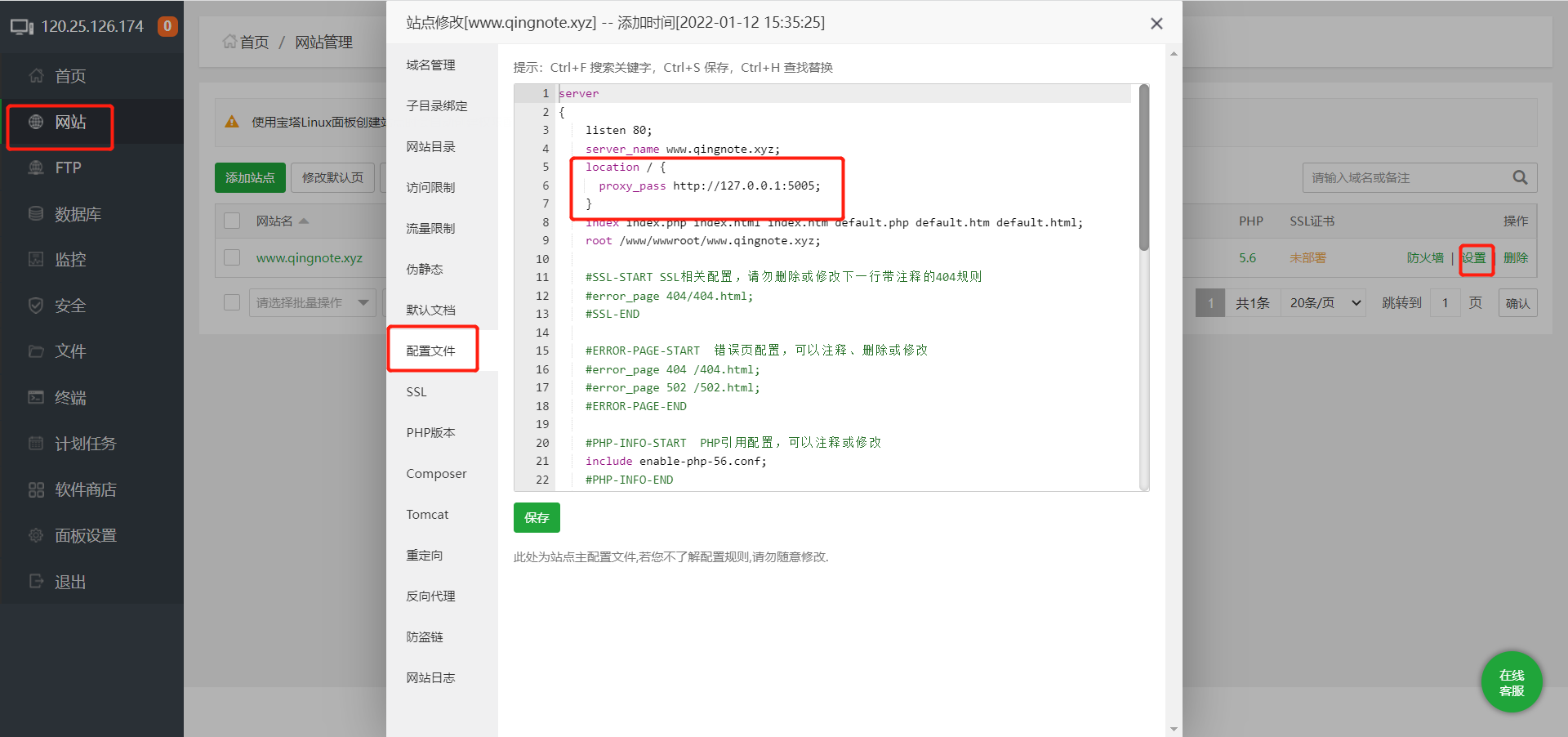
·关闭终端重新刷新根目录

·打开终端输入node server.js启动node服务

·放行端口



·修改站点配置文件



·打开浏览器输入域名进入node服务

·打开pm2管理器添加项目



# Vue富文本编辑器使用

·安装依赖

npm install vue-quill-editor -S

npm install quill-image-resize-module -S

npm install quill-image-drop-module -S

npm install quill -S

·vue.config.js中配置webpack

var webpack = require('webpack');

module.exports = {

    // 取消语法检查

    lintOnSave: false,

    // publicPath: './static/'

    // 要引入webpack

    configureWebpack: {

        plugins: [

            new webpack.ProvidePlugin({

                'window.Quill': 'quill/dist/quill.js',//注意路径，可能与你们路径不一致

                'Quill': 'quill/dist/quill.js' //注意路径，可能与你们路径不一致

            }),

        ]

    }

}

·创建组件



<template>

  <div id="home">

    <!-- 富文本编辑器组件 -->

    <!-- options配置富文本编辑器 -->

    <!-- focus编辑器获取焦点事件回调 -->

    <!-- blur编辑器失去焦点事件回调 -->

    <!-- change编辑器内容改变事件回调 -->

    <quill-editor

      v-model="content"

      ref="myQuillEditor"

      :options="editorOption"

      @focus="onEditorFocus($event)"

      @blur="onEditorBlur($event)"

      @change="onEditorChange($event)"

      class="editor"

    ></quill-editor>

    <!-- 文件上传的表单 -->

    <form

      action

      method="post"

      enctype="multipart/form-data"

      id="uploadFormMulti"

    >

      <!-- 上传图片 -->

      <input

        style="display: none"

        type="file"

        name="file"

        multiple

        accept="image/jpg, image/jpeg, image/png, image/gif"

        @change="uploadImg()"

        ref="upImg"

      />

      <!-- 上传视频 -->

      <input

        style="display: none"

        ref="upVideo"

        type="file"

        name="file"

        multiple

        accept="video/\*"

        @change="uploadVideo()"

      />

    </form>

    <!-- 插入图片对话框 -->

    <div class="insertImg" v-if="showInsertImg">

      <!-- 头部 -->

      <div class="insertImgHeader">

        <!-- 关闭按钮 -->

        <i class="iconfont icon-guanbi" @click="hideInsertImg"></i>

      </div>

      <!-- 插入网络图片 -->

      <div class="insertImgMain">

        <span>URL:</span>

        <input

          type="text"

          placeholder="网络图片地址"

          v-model="imgUrl"

          @keydown.enter="NetImg(imgUrl)"

        />

        <button @click="NetImg(imgUrl)">确定</button>

      </div>

      <!-- 上传本地图片 -->

      <button class="upLoadImgBtn" @click="upLoadImgClick">上传图片</button>

    </div>

    <!-- 插入视频对话框 -->

    <div class="insertImg" v-if="showInsertVideo">

      <!-- 头部 -->

      <div class="insertImgHeader">

        <!-- 关闭按钮 -->

        <i class="iconfont icon-guanbi" @click="hideInsertVideo"></i>

      </div>

      <!-- 插入网络视频 -->

      <div class="insertImgMain">

        <span>URL:</span>

        <input

          type="text"

          placeholder="网络视频嵌入代码"

          v-model="videoUrl"

          @keydown.enter="NetVideo(videoUrl)"

        />

        <button @click="NetVideo(videoUrl)">确定</button>

      </div>

      <!-- 上传本地视频 -->

      <button class="upLoadImgBtn" @click="upLoadVideoClick">上传视频</button>

    </div>

  </div>

</template>

<script>

// 引入编辑器模块

import "quill/dist/quill.core.css";

import "quill/dist/quill.snow.css";

import "quill/dist/quill.bubble.css";

import { quillEditor } from "vue-quill-editor";

// 引入图片缩放和拖拽模块

import Quill from "quill";

// const Clipboard = Quill.import("modules/clipboard");

// import Delta from "quill-delta";

import ImageResize from "quill-image-resize-module"; // 引用

import { ImageDrop } from "quill-image-drop-module";

Quill.register("modules/imageDrop", ImageDrop);

Quill.register("modules/imageResize", ImageResize); // 注册

// 工具栏配置

const toolbarOptions = [

  [{ header: [1, 2, 3, 4, 5, 6, false] }, { size: [] }], // 标题

  // [{ size: [] }], // 字体大小

  [{ color: [] }, { background: [] }], // 字体颜色、字体背景颜色

  [{ align: [] }, "bold", "italic", "underline", "strike"], // 加粗 斜体 下划线 删除线

  [{ font: [] }], // 字体种类

  ["blockquote", "code-block"], // 引用  代码块

  // [{ header: 1 }, { header: 2 }], // 1、2 级标题

  [{ list: "ordered" }, { list: "bullet" }], // 有序、无序列表

  [{ script: "sub" }, { script: "super" }], // 上标/下标

  [{ indent: "-1" }, { indent: "+1" }], // 缩进

  // [{ direction: "rtl" }], // 文本方向

  // [{ align: [] }], // 对齐方式

  ["clean"], // 清除文本格式

  ["link", "image", "video"], //图片和视频

];

export default {

  name: "Home",

  // 注册组件

  components: {

    quillEditor,

  },

  data() {

    return {

      // 是否显示插入图片

      showInsertImg: false,

      // 图片链接

      imgUrl: "",

      // 是否显示插入视频

      showInsertVideo: false,

      // 视频链接

      videoUrl: "",

      // 富文本编辑器默认内容

      content: "",

      // 配置

      editorOption: {

        // 编辑器主题

        theme: "snow",

        // 当编辑器为空时显示的占位符文字

        placeholder: "请输入正文",

        // 需要引入模块的对象，引入图片拖拽和缩放模块

        modules: {

          // 图片拖拽，false是拖拽不生成新图

          imageDrop: false,

          // 图片缩放

          imageResize: {

            // 样式

            displayStyles: {

              // 左下角尺寸的背景颜色

              backgroundColor: "#B49FCC",

              // 边框

              border: "none",

              // 字体颜色

              color: "white",

            },

            // 引入的模块Resize为尺寸，DisplaySize为显示尺寸，Toolbar为工具栏

            modules: ["Resize", "DisplaySize", "Toolbar"],

            // modules: ["Resize", "DisplaySize"],

          },

          //  富文本编辑器配置

          toolbar: {

            container: toolbarOptions,

          },

        },

      },

      // 窗口高度

      screenHeight: "",

      // 编辑器光标位置

      selection: null,

      // 创建formData对象

      formData: new FormData(),

      // 文件列表

      files: [],

      // 可支持上传的文件类型

      SupportType: "image,video",

      // 是否上传，如果没有可上传文件则不上传

      isUpLoad: false,

    };

  },

  // 计算属性

  computed: {

    //当前富文本实例

    editor() {

      return this.$refs.myQuillEditor.quill;

    },

  },

  mounted() {

    // 点击插入图片时弹出上传图片框事件

    var imgHandler = async (image) => {

      if (image) {

        // 保存光标位置

        this.selection = this.editor.getSelection();

        // 显示插入图片

        this.showInsertImg = true;

        // 隐藏插入图片对话框

        this.showInsertVideo = false;

      }

    };

    // 将imgHandler事件挂载到工具栏的图片按钮

    this.editor.getModule("toolbar").addHandler("image", imgHandler);

    // 点击插入视频时弹出上传图片框

    var videoHandler = async (video) => {

      if (video) {

        // 保存光标位置

        this.selection = this.editor.getSelection();

        // 显示插入视频

        this.showInsertVideo = true;

        // 隐藏插入图片对话框

        this.showInsertImg = false;

      }

    };

    // 将videoHandler事件挂载到工具栏的图片按钮

    this.editor.getModule("toolbar").addHandler("video", videoHandler);

    // console.log(this.editor);

    // this.editor.clipboard.addMatcher(Node.TEXT\_NODE, function (node, delta) {

    //   console.log("node");

    //   console.log(node);

    //   console.log("delta");

    //   // delta.ops[0].insert=

    //   console.log(delta.ops[0]);

    //   return new Delta().insert(node.data);

    // });

    // 工具栏提示文字

    // document.querySelector(".ql-header").title = "标题";

    document.querySelector(".ql-size").title = "字号";

    document.querySelector(".ql-color").title = "字体颜色";

    document.querySelector(".ql-background").title = "文字底色";

    document.querySelector(".ql-align").title = "对齐";

    document.querySelector(".ql-bold").title = "粗体";

    document.querySelector(".ql-italic").title = "斜体";

    document.querySelector(".ql-underline").title = "下划线";

    document.querySelector(".ql-strike").title = "删除线";

    document.querySelector(".ql-blockquote").title = "引用";

    document.querySelector(".ql-code-block").title = "代码块";

    document.querySelectorAll(".ql-list")[0].title = "有序列表";

    document.querySelectorAll(".ql-list")[1].title = "无序列表";

    document.querySelectorAll(".ql-script")[0].title = "下标";

    document.querySelectorAll(".ql-script")[1].title = "上标";

    document.querySelectorAll(".ql-indent")[0].title = "左缩进";

    document.querySelectorAll(".ql-indent")[1].title = "右缩进";

    document.querySelector(".ql-clean").title = "清除格式";

    document.querySelector(".ql-link").title = "链接";

    document.querySelector(".ql-image").title = "图片";

    document.querySelector(".ql-video").title = "视频";

    // 设置监听窗口大小事件，当窗口大小改变时回调handleResize方法

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

    // 初始化高度

    this.handleResize();

  },

  methods: {

    // 准备富文本编辑器

    onEditorReady() {},

    // 富文本编辑器 失去焦点事件

    onEditorBlur() {

      console.log(this.content);

    },

    // 富文本编辑器 获得焦点事件

    onEditorFocus() {

      // this.editor.clipboard.dangerouslyPasteHTML(

      //   5,

      //   "<p>Hello&nbsp;<strong>World</strong>!</p>",

      //   "https://www.jb51.net/article/87056.htm?tdsourcetag=s\_pctim\_aiomsg"

      // );

    },

    // 富文本编辑器 内容改变事件

    onEditorChange() {},

    // 上传视频点击确认时调用

    async uploadVideo() {

      console.log("上传视频");

    },

    // 上传图片点击确认时调用

    uploadImg() {

      // // var \_this = this;

      // //构造formData对象

      // var formData = new FormData();

      // // 将要上传的图片添加到formData

      // formData.append("file", document.getElementById("upImg").files[0]);

      // try {

      //   //axios调用上传文件接口

      //   //返回上传文件的地址

      //   let url =

      //     "https://gimg2.baidu.com/image\_search/src=http%3A%2F%2Fc-ssl.duitang.com%2Fuploads%2Fitem%2F202004%2F14%2F20200414164613\_nmoji.thumb.1000\_0.jpg&refer=http%3A%2F%2Fc-ssl.duitang.com&app=2002&size=f9999,10000&q=a80&n=0&g=0n&fmt=jpeg?sec=1644733886&t=a65efcc48fd25840d17a8e6d047a100a";

      //   // 如果url不为空

      //   if (url != null && url.length > 0) {

      //     // 获取当前光标位置

      //     let Range = this.editor.getSelection();

      //     //上传文件成功之后在富文本中回显(显示)

      //     this.editor.insertEmbed(

      //       Range != null ? Range.index : 0,

      //       "image",

      //       url

      //     );

      //   } else {

      //     console.log("图片上传失败");

      //   }

      // } catch ({ message: msg }) {}

      // //将文件的文本框的value置空

      // document.getElementById("upImg").value = "";

      // this.showInsertImg = false;

      // 将获取到的文件赋值给this.files

      this.files = this.$refs.upImg.files; // console.log(this.files);

      // 创建FileReader用于读取文件

      // const reader = new FileReader();

      //获得图片地址  放在reader.result内

      // reader.readAsDataURL(this.$refs.file.files[0]); // 当reader获取完成时调用

      // reader.onload = () => {

      //   // 将获取到的图片数据赋值给imgSrc

      //   this.imgSrc = reader.result;

      // };

      // 文件类型

      // let fileType;

      // 文件大小

      // let fileSize;

      // 遍历this.files数组

      for (let item of this.files) {

        // 获取文件类型

        // fileType = item.type.split("/")[0];

        // 如果支持上传该文件类型

        // if (this.SupportType.indexOf(fileType) != -1) {

        // 获取文件大小，单位mb

        // fileSize = item.size / (1024 \* 1024);

        // if (fileSize < 100) {

        // 将文件添加到formData

        this.formData.append("file", item);

        // 有可上传文件

        // this.isUpLoad = true;

        // } else {

        // this.$toast("上传的图片或视频应小于100M");

        // }

        // } else {

        // this.$toast("仅支持上传图片或视频");

        // }

      }

      // 如果有可上传文件，发送ajax请求

      // if (this.isUpLoad) {

      // this.isLoading = true;

      // 设置其他参数

      // this.formData.append("advContent", "");

      // 设置axios的参数

      var options = {

        // 请求api

        url: "", // 请求体

        data: this.formData, // 请求方式

        method: "post", // 请求头

        headers: {

          "Content-Type": "multipart/form-data",

        },

      }; // 发送ajax请求

      try {

        this.$http(options)

          .then((res) => {

            // 成功时回调

            console.log(res);

            // this.isLoading = false;

            // this.$toast("已成功添加");

            //   //返回上传文件的地址

            let url =

              "https://gimg2.baidu.com/image\_search/src=http%3A%2F%2Fc-ssl.duitang.com%2Fuploads%2Fitem%2F202004%2F14%2F20200414164613\_nmoji.thumb.1000\_0.jpg&refer=http%3A%2F%2Fc-ssl.duitang.com&app=2002&size=f9999,10000&q=a80&n=0&g=0n&fmt=jpeg?sec=1644733886&t=a65efcc48fd25840d17a8e6d047a100a";

            // 如果url不为空

            if (url != null && url.length > 0) {

              // 获取当前光标位置

              let Range = this.editor.getSelection();

              //上传文件成功之后在富文本中回显(显示)

              this.editor.insertEmbed(

                Range != null ? Range.index : 0,

                "image",

                url

              );

            } else {

              console.log("图片上传失败");

            }

          })

          .catch((req) => {

            // 失败时回调

            // this.isLoading = false;

            // this.$toast("添加失败");

            console.log(req);

          });

      } catch (e) {

        console.log(e);

      }

      // 重置isUpLoad

      // this.isUpLoad = false;

      // }

    },

    // 窗口大小改变时调用

    handleResize() {

      this.screenHeight = document.body.clientHeight; //获取窗口高度

      // 工具栏高度

      let toolbarHeight =

        document.querySelector(".ql-toolbar").clientHeight + 3;

      // console.log(toolbarHeight);

      // 获取编辑器主内容元素

      let ql\_container = document.querySelector(".ql-container");

      // 设置编辑器主内容的高度

      ql\_container.style.height = `${this.screenHeight - toolbarHeight}px`;

    },

    // 隐藏插入图片

    hideInsertImg() {

      this.showInsertImg = false;

    },

    // 隐藏插入视频

    hideInsertVideo() {

      this.showInsertVideo = false;

    },

    // 上传图片

    upLoadImgClick() {

      // document.getElementById("upImg").click(); //点击上传图片的输入框

      this.$refs.upImg.click();

    },

    // 上传视频

    upLoadVideoClick() {

      // document.getElementById("upVideo").click(); //点击上传视频的输入框

      this.$refs.upVideo.click();

    },

    // 插入网络图片

    NetImg(imgUrl) {

      // 获取网络图片url

      let url = imgUrl;

      // 如果url不为空

      if (url != null && url.length > 0) {

        // 获取光标位置

        let Range = this.selection;

        //上传文件成功之后在富文本中回显(显示)

        this.editor.insertEmbed(Range != null ? Range.index : 0, "image", url);

        // 隐藏插入图片

        this.showInsertImg = false;

        // 清空图片链接

        this.imgUrl = "";

      }

    },

    // 插入网络视频

    NetVideo(videoUrl) {

      let url;

      // 获取视频url

      try {

        // 获取从//到" 的字符串再按"分组,获得第一组

        url = videoUrl.match(/(?<=\/\/).\*(?="\s)/)[0].split('"')[0];

      } catch (e) {

        try {

          // 获取从//到' 的字符串再按'分组,获得第一组

          url = videoUrl.match(/(?<=\/\/).\*(?='\s)/)[0].split("'")[0];

        } catch (e) {

          // 获取//后的字符串

          url = videoUrl.split("//")[1];

        }

      }

      url = "https://" + url;

      // 如果url不为空

      if (url != null && url.length > 0) {

        // 获取光标位置

        let Range = this.selection;

        //上传文件成功之后在富文本中回显(显示)

        this.editor.insertEmbed(Range != null ? Range.index : 0, "video", url);

        // 隐藏插入视频

        this.showInsertVideo = false;

        // 清空视频链接

        this.videoUrl = "";

      }

    },

    ctrlv() {

      console.log("粘贴");

    },

  },

};

</script>

<style lang="less" scoped>

#home {

  width: 100%;

  height: 100%;

  // background-image: url("../images/bg.jpg");

  // background-size: cover;

  // background-repeat: no-repeat;

  // background-position: center;

  .insertImg {

    width: 380px;

    // height: 300px;

    background-color: rgb(234, 246, 255);

    // background-color: white;

    position: absolute;

    left: 50%;

    top: 50%;

    transform: translate(-50%, -50%);

    border-radius: 10px;

    // border: rgb(47, 47, 47) 1px solid;

    // z-index: 101;

    .insertImgHeader {

      border-radius: 10px 10px 0 0;

      width: 100%;

      height: 30px;

      background-color: rgb(106, 168, 212);

      display: flex;

      align-items: center;

      justify-content: flex-end;

      i {

        margin-right: 10px;

        color: white;

        font-size: 14px;

        cursor: pointer;

      }

    }

    .insertImgMain {

      width: 100%;

      display: flex;

      justify-content: center;

      align-items: center;

      margin: 20px 0;

      span {

        // color: white;

        font-size: 16px;

        margin-right: 10px;

      }

      input {

        // border: none;

        // background-color: rgb(106, 168, 212);

        border: 1px solid rgb(106, 168, 212) s;

        border-radius: 5px;

        width: 60%;

        height: 30px;

        padding: 0 20px;

        border: 1px solid transparent;

        // 鼠标移入输入框包裹器时

        &:hover {

          // 边框

          border: 1px solid #0182c7;

        }

      }

      button {

        width: 60px;

        height: 30px;

        background-color: rgb(106, 168, 212);

        margin-left: 10px;

        border-radius: 5px;

        font-size: 14px;

        color: white;

        cursor: pointer;

      }

    }

    .upLoadImgBtn {

      display: block;

      width: 90px;

      height: 30px;

      background-color: rgb(106, 168, 212);

      margin: 0 auto;

      border-radius: 999px;

      text-align: center;

      line-height: 30px;

      color: white;

      margin-bottom: 20px;

      cursor: pointer;

    }

  }

  // .overLay{

  //   width: 100%;

  //   height: 100%;

  //   background-color: rgba(0, 0, 0, .5);

  //   z-index: 100;

  //   position: absolute;

  //   left: 0;

  //   top: 0;

  // }

}

// 插入的视频样式

/deep/ .ql-editor .ql-video {

  // 宽高

  width: 487px;

  height: 361px;

  // 最大宽度

  max-width: 100%;

}

// 输入框内容

/deep/ .ql-editor {

  // 隐藏滚动条

  &::-webkit-scrollbar {

    display: none;

  }

}

// 输入框

/deep/ .ql-container.ql-snow {

  // 清除输入框边框

  border: none;

}

// 工具栏

/deep/ .ql-toolbar.ql-snow {

  border: none;

  border-bottom: 1px solid #ccc;

}

// /deep/ .ql-snow .ql-editor img{

//   width: 50%;

// }

// 编辑器

.editor {

  // 行高

  line-height: normal !important;

  // 高度

  height: 100%;

  // 宽度

  // width: 1024px;

  width: 100%;

  // 水平居中

  // margin: 0 auto;

  // 背景颜色

  background-color: white;

}

/deep/ .ql-snow .ql-tooltip[data-mode="link"]::before {

  content: "请输入链接地址:";

}

/deep/ .ql-snow .ql-tooltip.ql-editing a.ql-action::after {

  border-right: 0px;

  content: "保存";

  padding-right: 0px;

}

/deep/ .ql-snow .ql-tooltip[data-mode="video"]::before {

  content: "请输入视频地址:";

}

/deep/ .ql-snow .ql-picker.ql-size .ql-picker-label::before,

/deep/ .ql-snow .ql-picker.ql-size .ql-picker-item::before {

  content: "14px";

}

/deep/ .ql-snow .ql-picker.ql-size .ql-picker-label[data-value="small"]::before,

/deep/ .ql-snow .ql-picker.ql-size .ql-picker-item[data-value="small"]::before {

  content: "10px";

}

/deep/ .ql-snow .ql-picker.ql-size .ql-picker-label[data-value="large"]::before,

/deep/ .ql-snow .ql-picker.ql-size .ql-picker-item[data-value="large"]::before {

  content: "18px";

}

/deep/ .ql-snow .ql-picker.ql-size .ql-picker-label[data-value="huge"]::before,

/deep/ .ql-snow .ql-picker.ql-size .ql-picker-item[data-value="huge"]::before {

  content: "32px";

}

// /deep/ .ql-snow .ql-picker.ql-header .ql-picker-label::before,

// .ql-snow .ql-picker.ql-header .ql-picker-item::before {

//   content: "文本";

// }

/deep/ [data-v-fae5bece] .ql-snow .ql-picker.ql-header .ql-picker-label::before,

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-item[data-v-fae5bece]::before {

  content: "正文";

}

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-label::before,

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-item::before {

  content: "正文";

}

// .ql-snow .ql-picker.ql-header .ql-picker-label[data-value="1"]::before,

// .ql-snow .ql-picker.ql-header .ql-picker-item[data-value="1"]::before {

//   content: "标题1";

// }

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-label[data-value="1"]::before,

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-item[data-value="1"]::before {

  content: "标题1";

}

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-label[data-value="2"]::before,

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-item[data-value="2"]::before {

  content: "标题2";

}

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-label[data-value="3"]::before,

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-item[data-value="3"]::before {

  content: "标题3";

}

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-label[data-value="4"]::before,

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-item[data-value="4"]::before {

  content: "标题4";

}

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-label[data-value="5"]::before,

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-item[data-value="5"]::before {

  content: "标题5";

}

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-label[data-value="6"]::before,

/deep/ .ql-snow .ql-picker.ql-header .ql-picker-item[data-value="6"]::before {

  content: "标题6";

}

/deep/ .ql-snow .ql-picker.ql-font .ql-picker-label::before,

/deep/ .ql-snow .ql-picker.ql-font .ql-picker-item::before {

  content: "标准字体";

}

/deep/ .ql-snow .ql-picker.ql-font .ql-picker-label[data-value="serif"]::before,

/deep/ .ql-snow .ql-picker.ql-font .ql-picker-item[data-value="serif"]::before {

  content: "衬线字体";

}

/deep/

  .ql-snow

  .ql-picker.ql-font

  .ql-picker-label[data-value="monospace"]::before,

/deep/

  .ql-snow

  .ql-picker.ql-font

  .ql-picker-item[data-value="monospace"]::before {

  content: "等宽字体";

}

/deep/ .ql-snow .ql-tooltip::before {

  content: "打开链接";

}

/deep/ .ql-snow .ql-tooltip a.ql-action::after {

  content: "编辑";

}

/deep/ .ql-snow .ql-tooltip a.ql-remove::before {

  content: "移除";

}

@media (max-width: 1024px) {

  .editor {

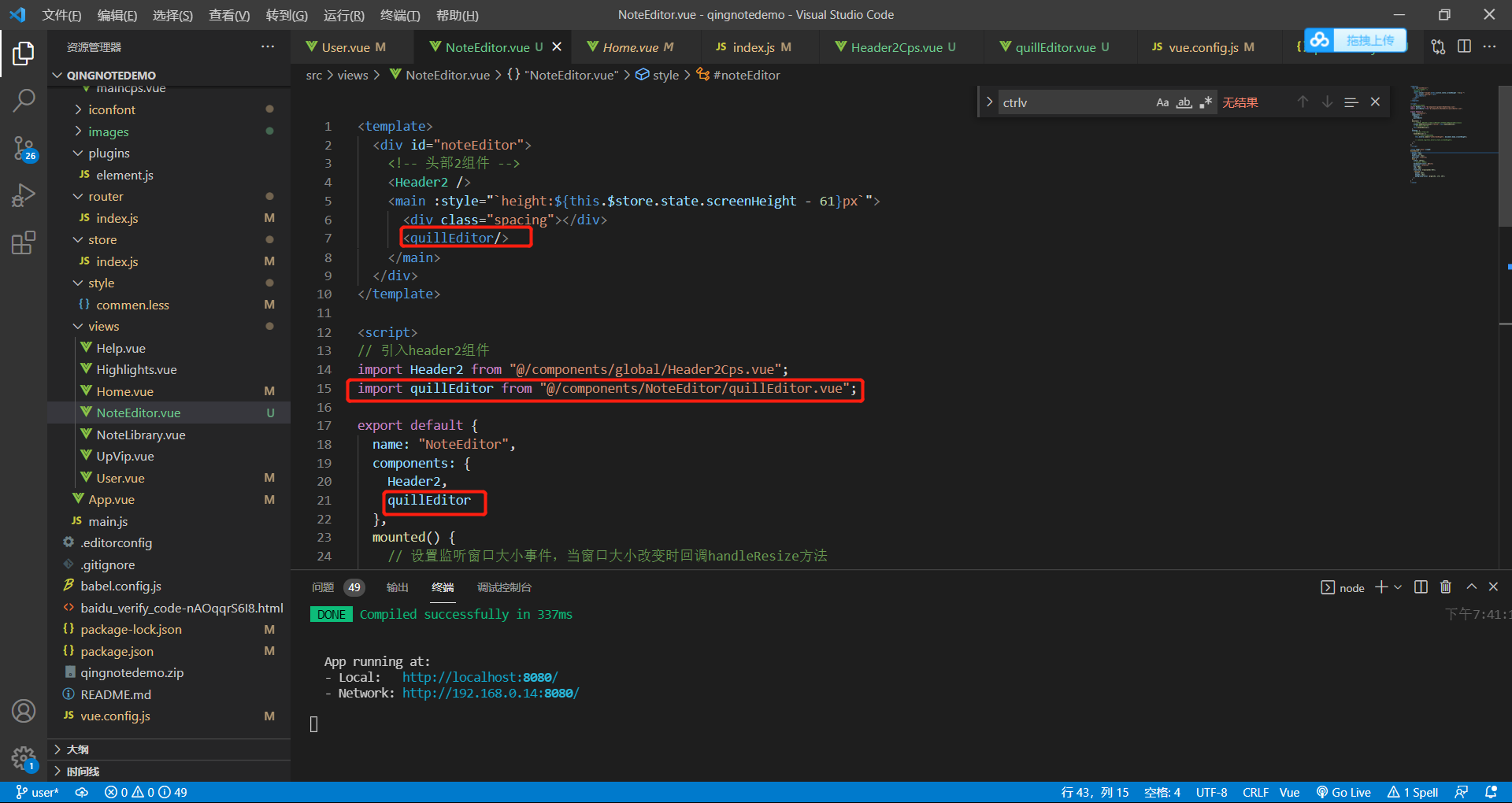
    width: 100%;

  }

}

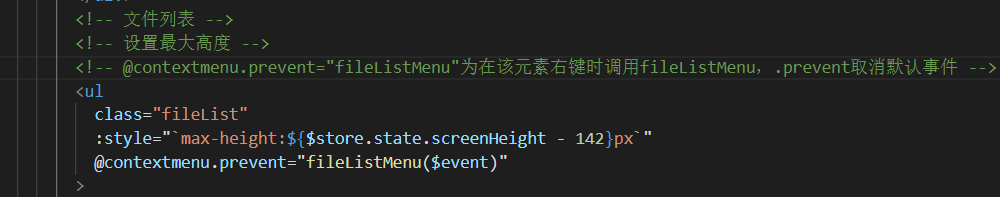
</style>

·在想要的位置引入组件



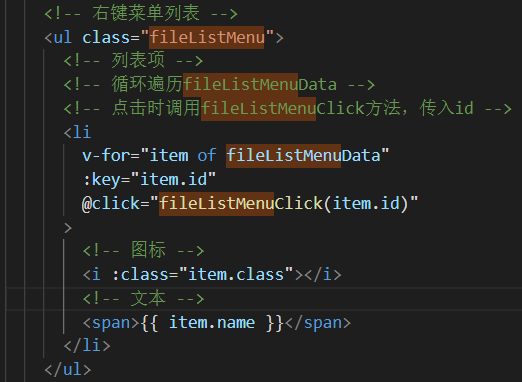
# vue重写右键点击事件

右键点击事件的元素



@contextmenu.prevent="fileListMenu($event)"

母菜单列表元素



    <!-- 母右键菜单列表 -->

    <ul class="fileListMenu">

      <!-- 列表项 -->

      <!-- 循环遍历fileListMenuData -->

      <!-- 点击时调用fileListMenuClick方法，传入id -->

      <li

        v-for="item of fileListMenuData"

        :key="item.id"

        @click="fileListMenuClick(item.id)"

      >

        <!-- 图标 -->

        <i :class="item.class"></i>

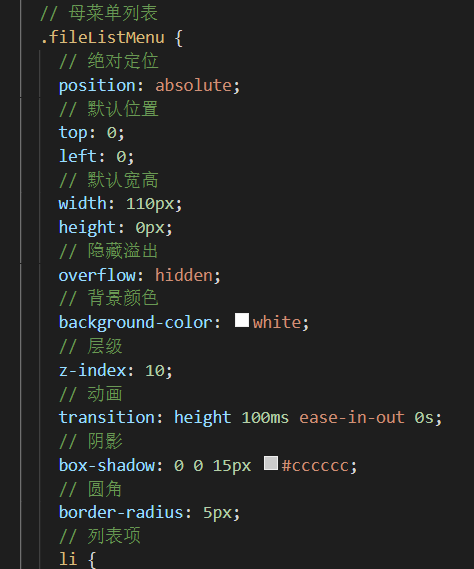
        <!-- 文本 -->

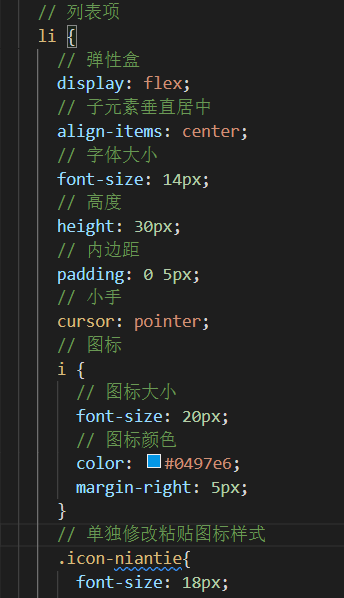
        <span>{{ item.name }}</span>

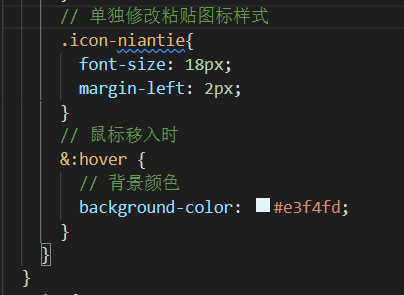
      </li>

    </ul>

母菜单列表css样式







  // 母菜单列表

  .fileListMenu {

    // 绝对定位

    position: absolute;

    // 默认位置

    top: 0;

    left: 0;

    // 默认宽高

    width: 110px;

    height: 0px;

    // 隐藏溢出

    overflow: hidden;

    // 背景颜色

    background-color: white;

    // 层级

    z-index: 10;

    // 动画

    transition: height 100ms ease-in-out 0s;

    // 阴影

    box-shadow: 0 0 15px #cccccc;

    // 圆角

    border-radius: 5px;

    // 列表项

    li {

      // 弹性盒

      display: flex;

      // 子元素垂直居中

      align-items: center;

      // 字体大小

      font-size: 14px;

      // 高度

      height: 30px;

      // 内边距

      padding: 0 5px;

      // 小手

      cursor: pointer;

      // 图标

      i {

        // 图标大小

        font-size: 20px;

        // 图标颜色

        color: #0497e6;

        margin-right: 5px;

      }

      // 单独修改粘贴图标样式

      .icon-niantie{

        font-size: 18px;

        margin-left: 2px;

      }

      // 鼠标移入时

      &:hover {

        // 背景颜色

        background-color: #e3f4fd;

      }

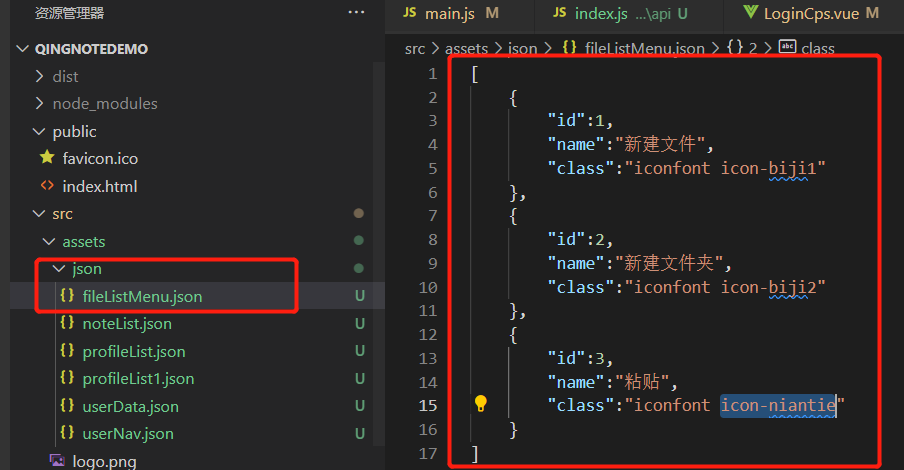
    }

  }

引入列表数据



列表数据



[

    {

        "id":1,

        "name":"新建文件",

        "class":"iconfont icon-biji1"

    },

    {

        "id":2,

        "name":"新建文件夹",

        "class":"iconfont icon-biji2"

    },

    {

        "id":3,

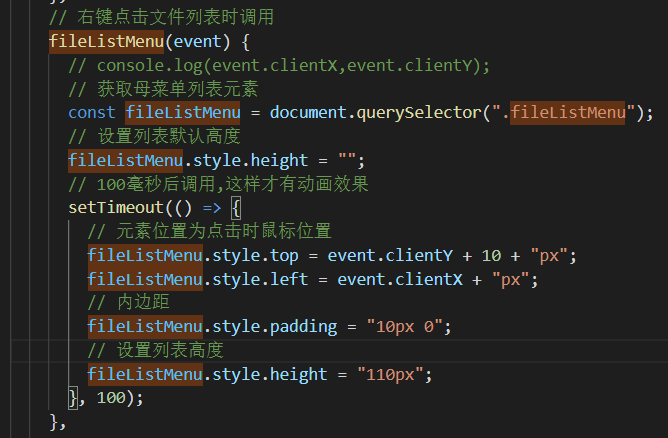
        "name":"粘贴",

        "class":"iconfont icon-niantie"

    }

]

右键点击时调用



// 右键点击文件列表时调用

    fileListMenu(event) {

      // console.log(event.clientX,event.clientY);

      // 获取母菜单列表元素

      const fileListMenu = document.querySelector(".fileListMenu");

      // 设置列表默认高度

      fileListMenu.style.height = "";

      // 100毫秒后调用,这样才有动画效果

      setTimeout(() => {

        // 元素位置为点击时鼠标位置

        fileListMenu.style.top = event.clientY + 10 + "px";

        fileListMenu.style.left = event.clientX + "px";

        // 内边距

        fileListMenu.style.padding = "10px 0";

        // 设置列表高度

        fileListMenu.style.height = "110px";

      }, 100);

    },

点击非列表位置时调用



  created() {

    // document点击事件

    document.addEventListener("click", (e) => {

      // 排序列表

      const fileListMenu = document.querySelector(".fileListMenu");

      // 头像

      const addFile = document.querySelector(".addFile");

      // 如果sortList不为null

      if (fileListMenu != null) {

        // 如果点击的不是ProfileMore和addFile元素

        if (!fileListMenu.contains(e.target) ) {

          // 隐藏新建列表

          fileListMenu.style.height = "";

          setTimeout(() => {

            fileListMenu.style.padding = "";

          }, 100);

        }

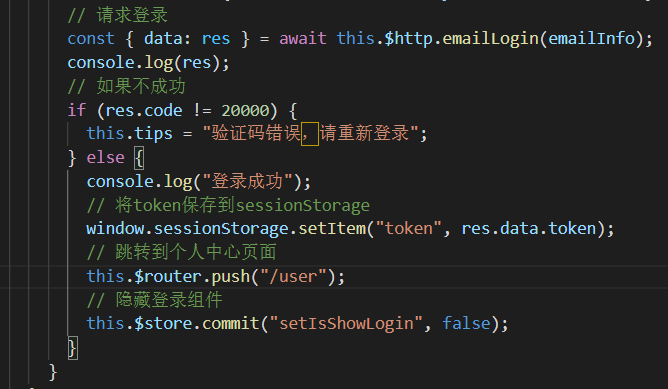
      }

    });

  },

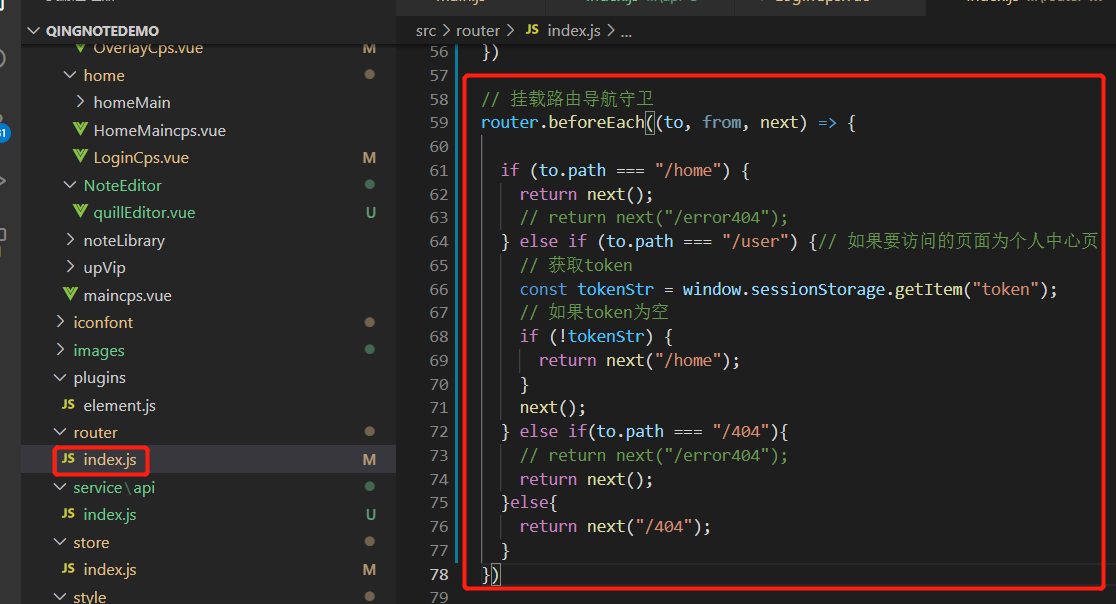
# vue登录后操作

将token保存到sessionStorage



         window.sessionStorage.setItem("token", res.data.token);

挂载路由导航守卫



// 挂载路由导航守卫

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

  if (to.path === "/home") {

    return next();

    // return next("/error404");

  } else if (to.path === "/user") {// 如果要访问的页面为个人中心页

    // 获取token

    const tokenStr = window.sessionStorage.getItem("token");

    // 如果token为空

    if (!tokenStr) {

      return next("/home");

    }

    next();

  } else if(to.path === "/404"){

    // return next("/error404");

    return next();

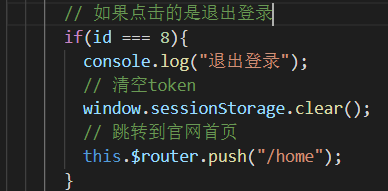
  }else{

    return next("/404");

  }

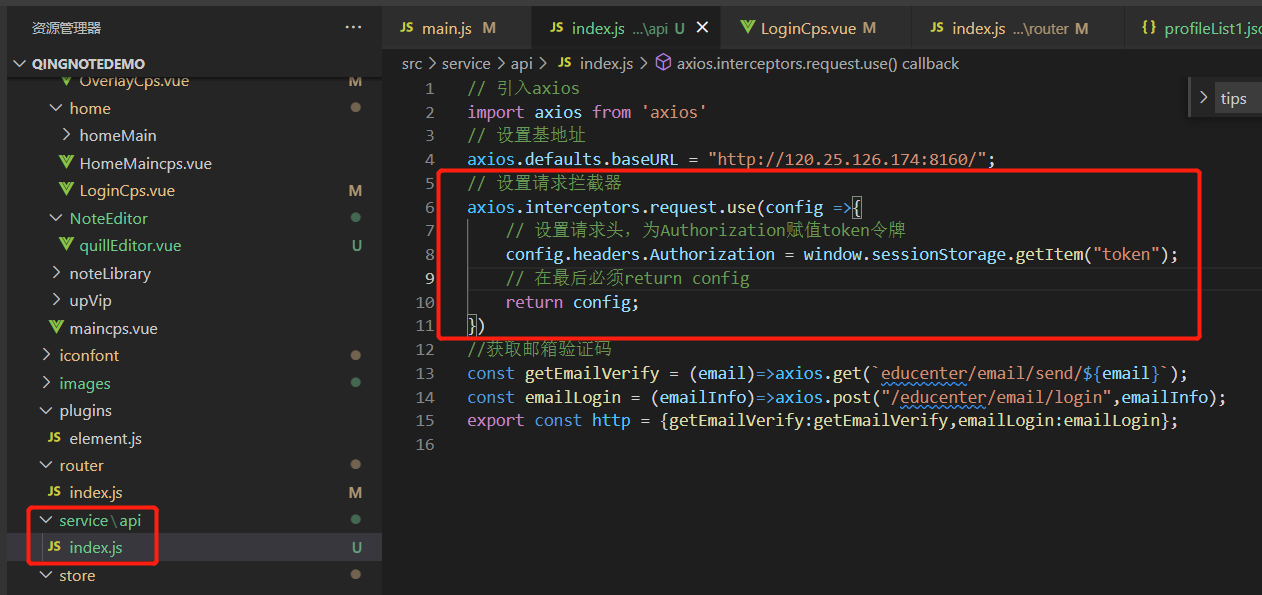
})

点击退出登录时



        window.sessionStorage.clear();

通过axios拦截器添加token验证



// 设置请求拦截器

axios.interceptors.request.use(config =>{

    // 设置请求头，为Authorization赋值token令牌

    config.headers.Authorization = window.sessionStorage.getItem("token");

    // 在最后必须return config

    return config;

})

注：这一步要根据情况把Authorization更改为后端根据请求的获取token的对应字段

# axios传入字符串报错问题

修改请求头"Content-Type": "application/json"



# 获取字符串的字节数

   //获取字符串的字节数

    changeByte(str) {

      let total = 0;

      let charCode;

      for (let i = 0, len = str.length; i < len; i++) {

        charCode = str.charCodeAt(i);

        if (charCode <= 0x007f) {

          total += 1;

        } else if (charCode <= 0x07ff) {

          total += 2;

        } else if (charCode <= 0xffff) {

          total += 3;

        } else {

          total += 4;

        }

      }

      // 将字节转kb

      total /= 1024;

      // 获取小数点最后两位

      console.log(total.toFixed());

      return total.toFixed("2");

    },

# 将指定内容复制到剪切板

      // 直接构建input

      let input = document.createElement("input");

      //设置只读，否则移动端会有键盘弹出

      input.setAttribute("readonly", "readonly");

      // 设置内容

      input.value = "data";

      // 添加临时实例

      document.body.appendChild(input);

      // 选择实例内容

      input.select();

      // 执行复制

      document.execCommand("Copy");

      // 删除临时实例

      document.body.removeChild(input);