Vue3.0

1. 引入
   1. main.js里

//要是用vue3.0的写法要引入此插件

import VueCompositionApi from '@vue/composition-api'

Vue.use(VueCompositionApi)

1. 数据定义
   1. 3.0之前的版本：

  data(){

    return{

      title:'hello name'

    }

  },

* 1. 3.0版本的：

  setup() {

    const state = reactive({

      title: "hello name",

    });

    return state

  },

* 1. 目的：reactive专门用来处理响应式数据
  2. 只返回少量需要的数据

  setup() {

    const state = reactive({

      title: "hello name",

      name:"马克",

      age:"年龄"

    });

    return {

      title:state.title,

      name:state.name

    }

  },

* 1. 返回多个数据，可以用扩展运算符

  setup() {

    const state = reactive({

      title: "hello name",

      name:"马克",

      age:"年龄"

    });

    return {

      ...state

    }

  },

1. 方法定义
   1. 3.0之前的版本：

  methods:{

    play(){

      alert('11111');

    }

  },

* 1. 3.0版本的：

toRefs:在组件里引入，说明当前的数据和方法都是响应式的，可以实现双向绑定

import { reactive , toRefs} from "@vue/composition-api";

  setup() {

    const state = reactive({

      title: "hello name",

      name: "马克",

      age: 10,

    });

    let play = () => {

      state.age += 2;

    };

    return {

      ...toRefs(state),

      play,

    };

  },

1. 计算属性
   1. 3.0之前的版本：

  computed:{

    num(){

      return 100;

    }

  },

* 1. 3.0版本的：

computed：组件中引入computed

import { reactive, toRefs , computed } from "@vue/composition-api";

  setup() {

    const state = reactive({

      num:computed(()=>100)

    });

    return {

      ...toRefs(state)

    };

  },

1. 其他
   1. mounted(){}
      1. 3.0版本的：

import { reactive, toRefs, computed, onMounted } from "@vue/composition-api";

  setup() {

    const state = reactive({

      title: "hello name",

    });

    const play = () => {

      alert("welcome")

    };

    onMounted(()=>{

      play()

    });

    return {

      ...toRefs(state),

    };

  },

1. 父子组件的传值
   1. 父传子：provide传值，provide的两个参数：(对应数据的名字；要传的值)

import {

  reactive,

  toRefs,

  computed,

  onMounted,

  provide,

} from "@vue/composition-api";

  setup() {

    const state = reactive({

      title: "hello name",

      hellomsg:"打招呼"

    });

    provide('hellomsg',state.hellomsg)

    return {

      ...toRefs(state),

    };

  },

* 1. 子组件接受父组件的值用inject

setup() {

    const state = reactive({

      hellomsg: inject("hellomsg"),

    });

    return {

      ...state,

    };

  },

1. setup
   1. 默认两个参数(props,object)

 setup(props, { root }) {

    const state = reactive({

    });

    return {

      ...state,

    };

  },

* 1. main.js文件里

//相当于在Vue的原型对象里添加了一个属性$http(此名可换)

axios.default.baseURL = "ip地址+端口号"

Vue.prototype.$http = axios;

* 1. 由于setup里没有this这个属性，所以用setup参数里的root属性

 setup(props, { root }) {

    const state = reactive({

    });

    let getData = () => {

      //这个root是setup的object对象中属性的值

      root.$http.get("url").then((res) => {}),

        (error) => {

          console.log(error);

        };

    };

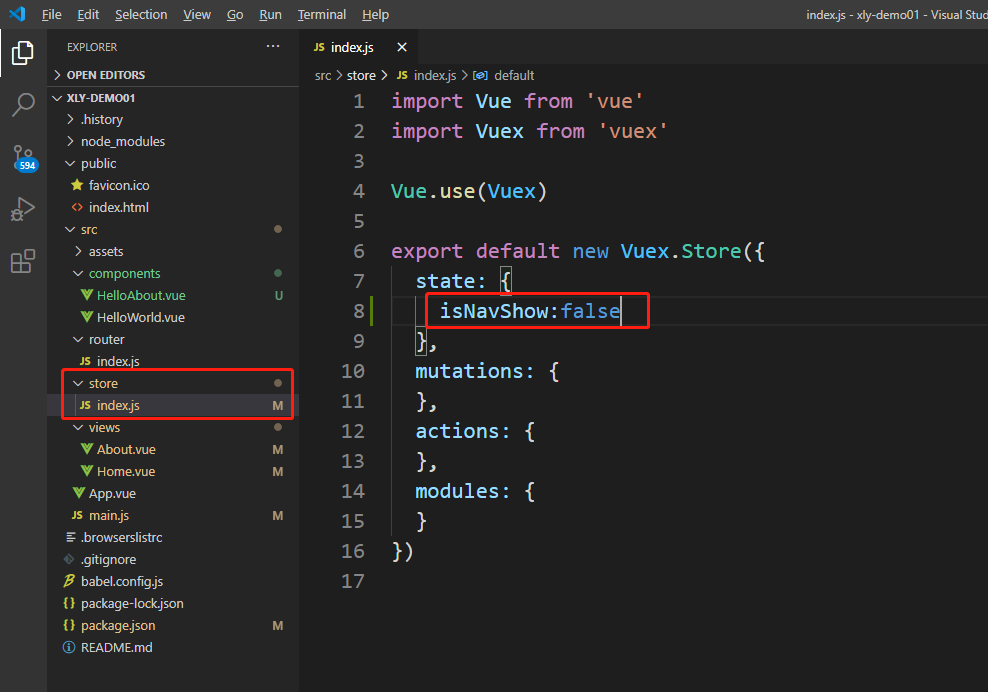
    return {

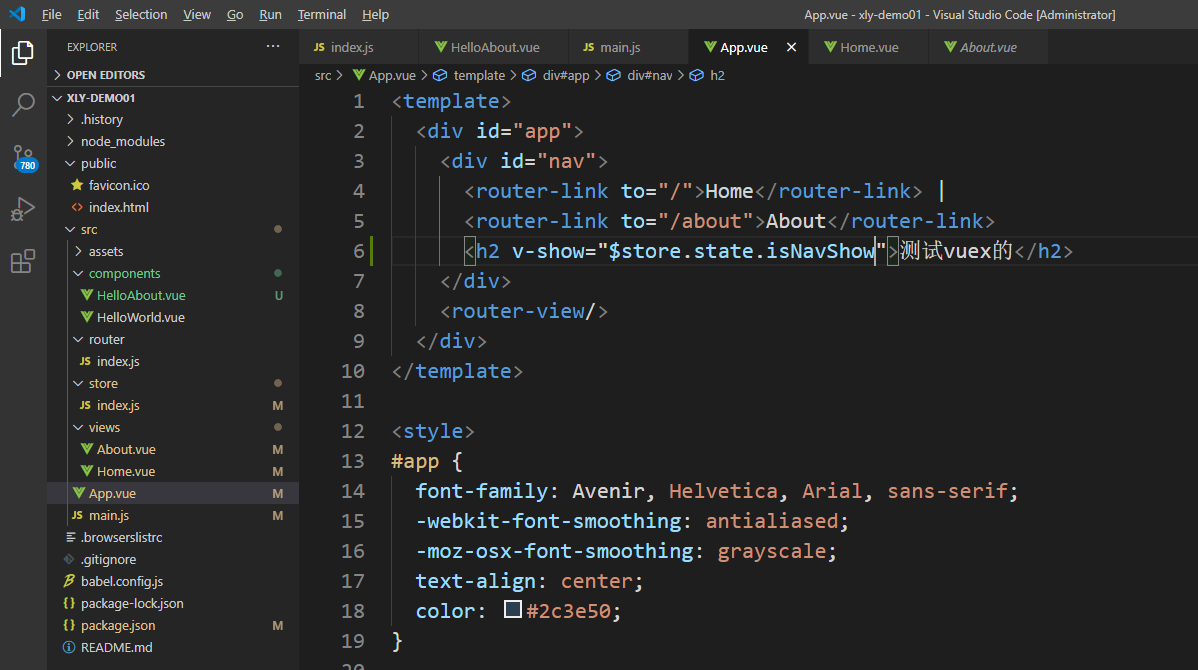
      ...state,

      getData,

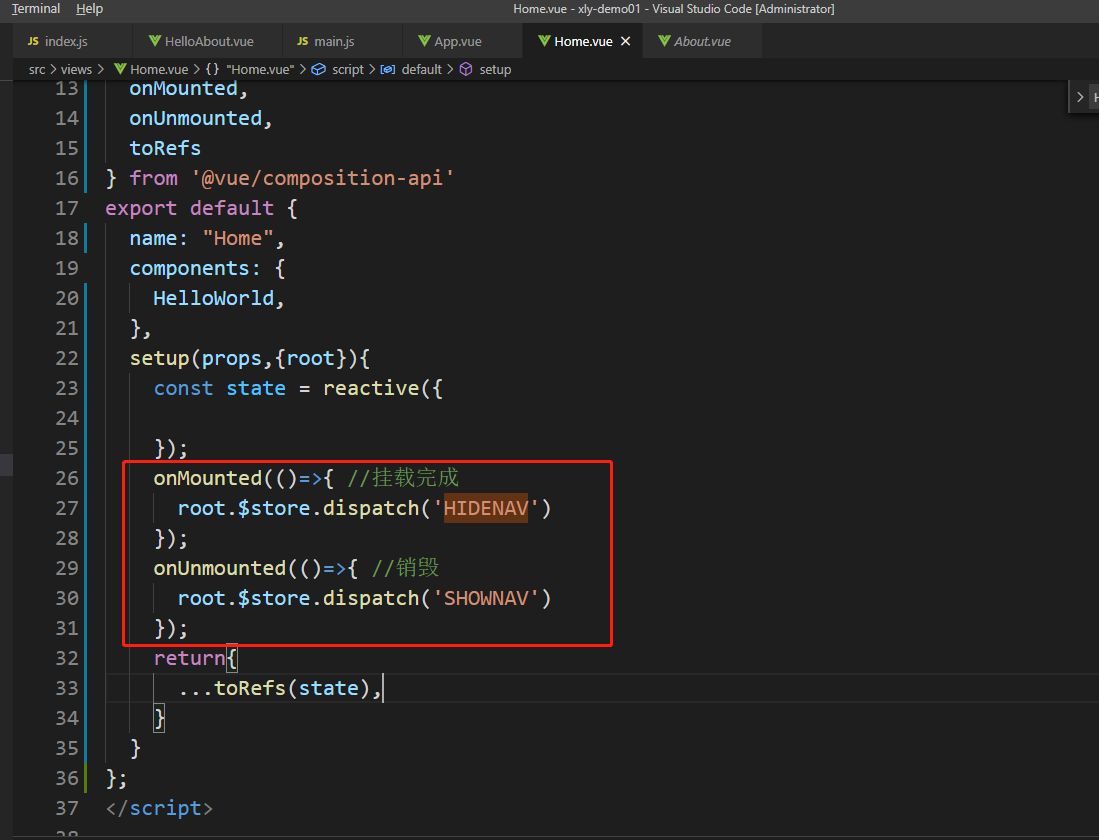
    };

  },

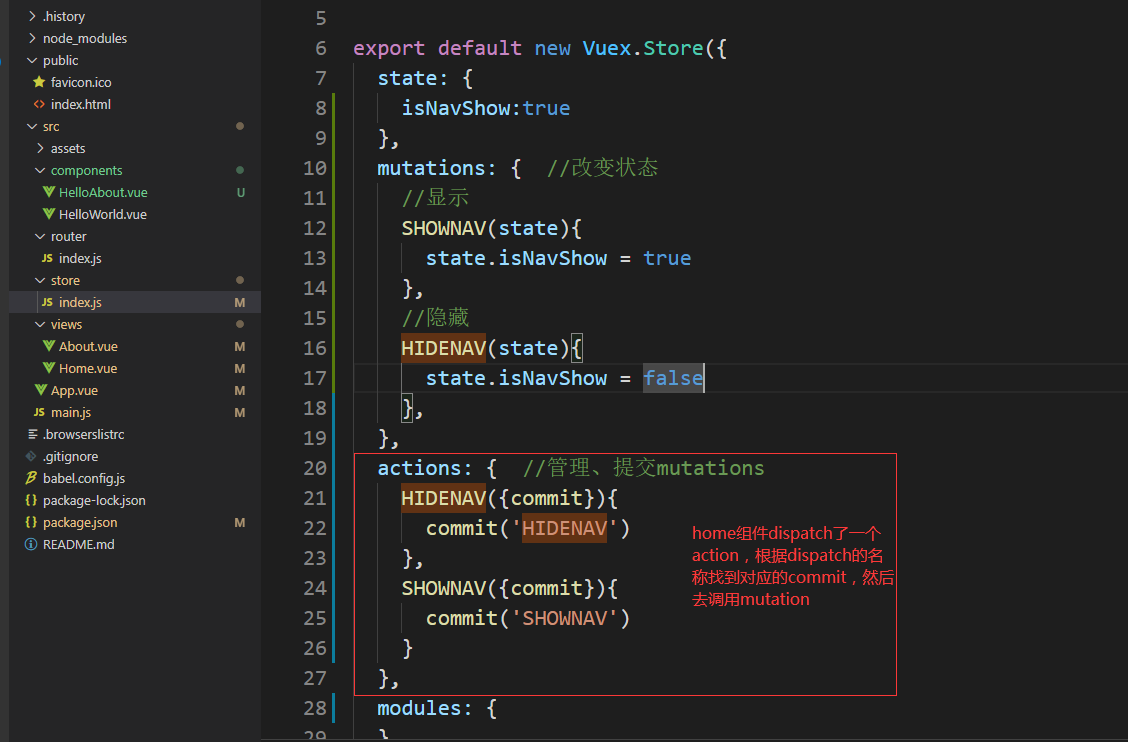
1. vuex
   1. 原先搭建vue框架的时候有添加vuex，在store的index.js里定义一个全局变量的值
   2. 全局App.vue里面设置v-show绑定



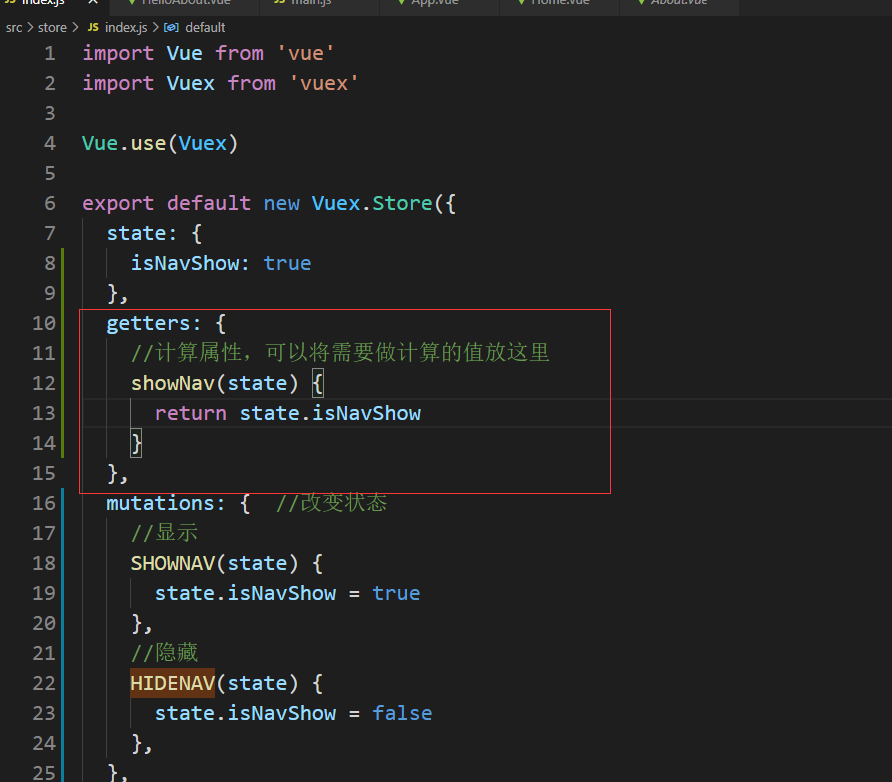
* 1. 如果想要部分页面可见，部分页面不可见，就可以在组件里对这个在vuex的值进行修改
  2. 例：我需要在home组件里隐藏，点击about跳转到about组件的时候是可见的状态



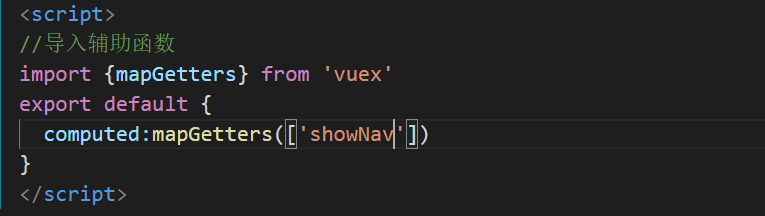
* 1. 上图可以看到，组件加载完成以及组件销毁的时候，都dispatch了一个action，这时，action就会commit一个mutation，根据提交的名称找到对应的mutation里的方法，对state的值进行修改



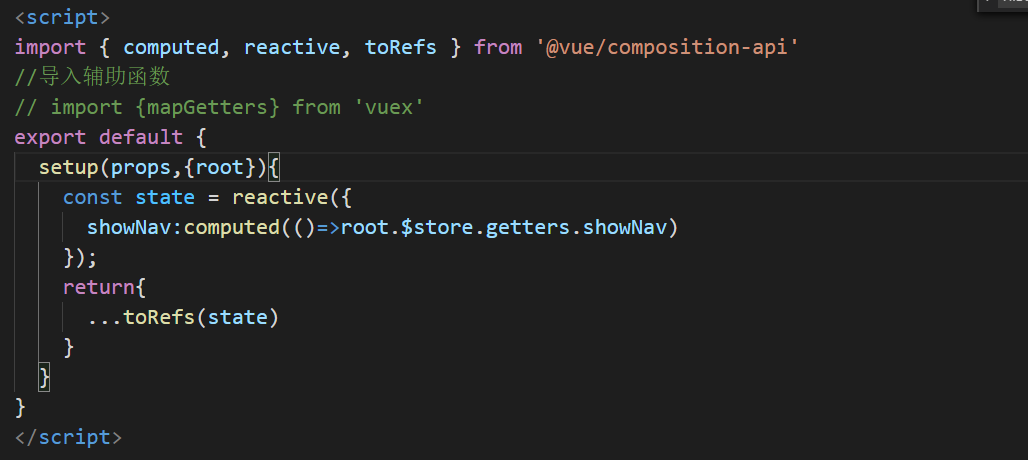
* 1. vuex的辅助函数



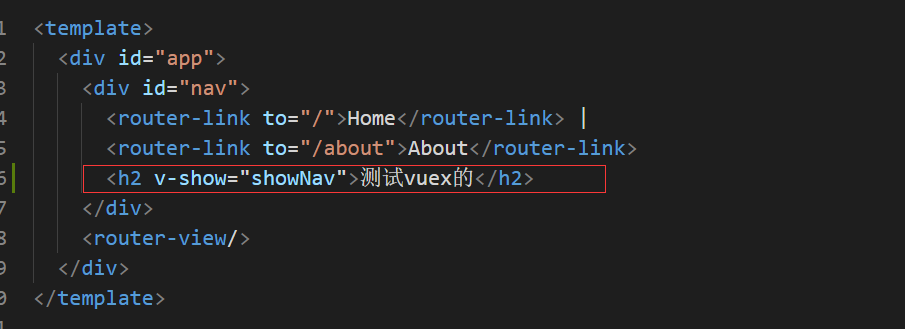
* 1. 页面引用的时候，这个getters就写在计算属性computed里
     1. 页面引入该函数，将值写入计算属性里
        1. 2.0版本的写法



* + - 1. 3.0版本的写法



* + 1. 页面调用



* 1. 网络请求的封装
     1. 网络请求
        1. 原生的XMLHTTPRequest，现在一般使用第三方的封装插件jQuery，axios，后来js又出现的新的api——fetch
     2. axios一般使用
        + 1. axios.get (url,config)

// axios.get 发起get请求

// 参数一：表示请求地址

// 参数二：表示配置信息

//  params 表示传递到服务器端的数据，以url参数的形式拼接在请求地址后面

//    { page:1,per:3 }

//    如:http://api/v1/product

//    最终生成的URL为http://api/v1/product？page=1&per=3

//  header 表示请求头

// axios.get('http://api/v1/product',{

//   params:{

//     page:1,

//     per:3

//   }

// })

* + - * 1. axios.post(url,data,config)

// post请求传递三个参数

// 参数一：表示请求地址

// 参数二：传递的数据 在请求体中传递

//  axios默认发送的数据格式是json格式的

//    配置信息

//    header

//    默认情况下的请求头：content-type:'application/json'

// axios.get('http://api/vi/login', {

//   userName: admin,

//   password: 123456

// }, {

//   //这里面选写配置信息

//   params:{

//     // 这里也有params参数,如果配置了，在url里面也会携带

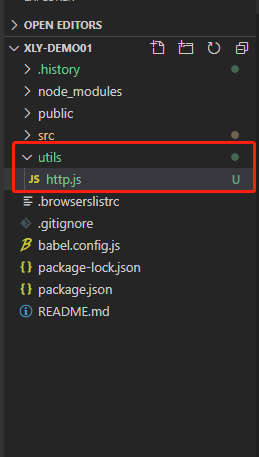
//   }

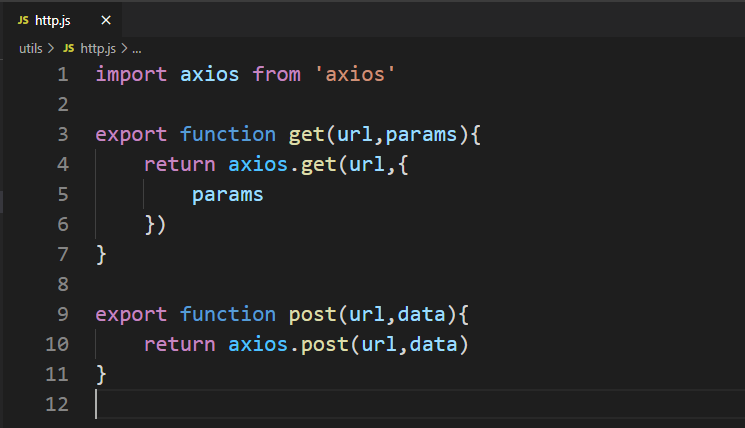
// })

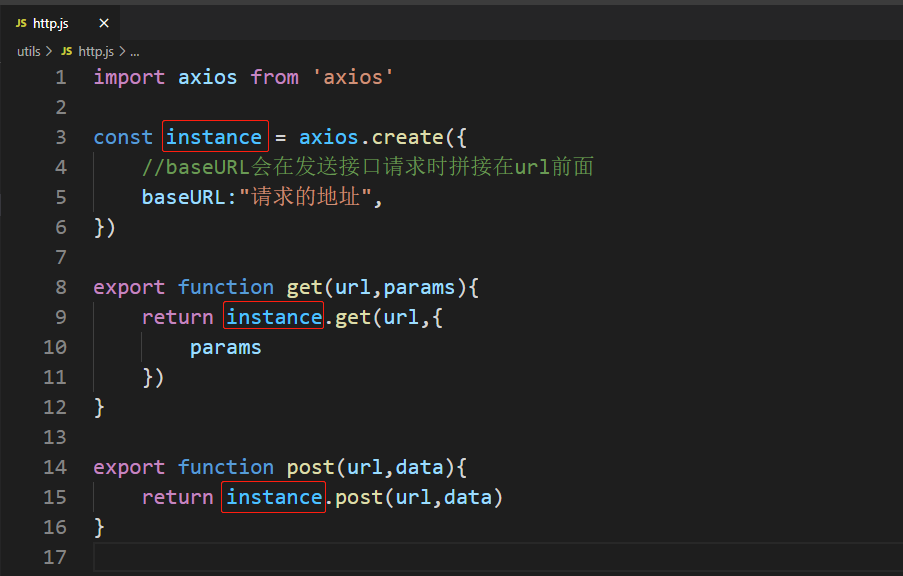
* + 1. axios封装
       1. 为什么要进行axios封装

如果像上面的代码一样，重复的写axios.get或者axios.post这样调用接口，如果说需要更换环境，那url的地址势必要更换，工作量大且重复

* + - 1. 操作步骤
         1. 根目录下新建utils文件夹，新建http.js



* + - * 1. 安装axios（如果已安装，请忽视），导入axios，将axios的各种请求方式提取出来
        2. 以上仅仅是提取了请求方式，并没有将url提取出来。要统一配置url请求的地址——创建axios对象，设置baseURL（这里注意，如果是分布式项目有多个ip地址，可以将这个地址配置在api那边的接口前）



* + - * 1. 这个baseURL就是请求的域名，url是接口名
      1. 全局拦截配置
         1. 有的接口请求都需要在请求头里加token请求，对于这样情况，可以做一个请求拦截，在发送网络请求之前先走请求拦截
         2. 定义请求拦截和响应拦截，可以在接口请求之前或者接口响应之后做出一些操作

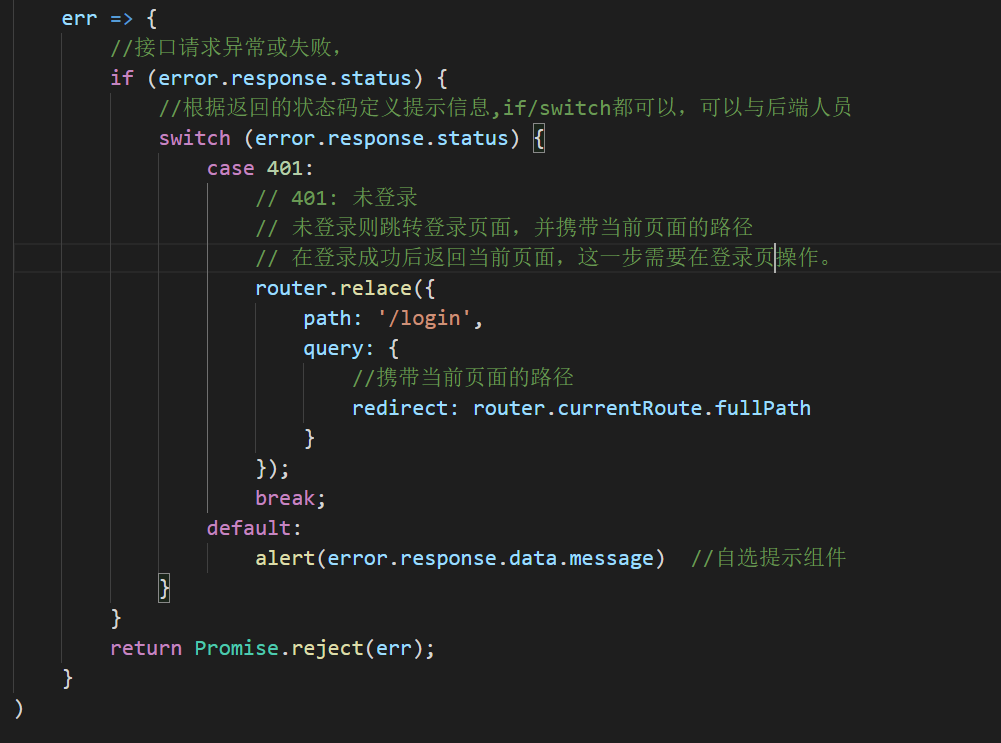


* + - * 1. 请求拦截：自定义请求内容



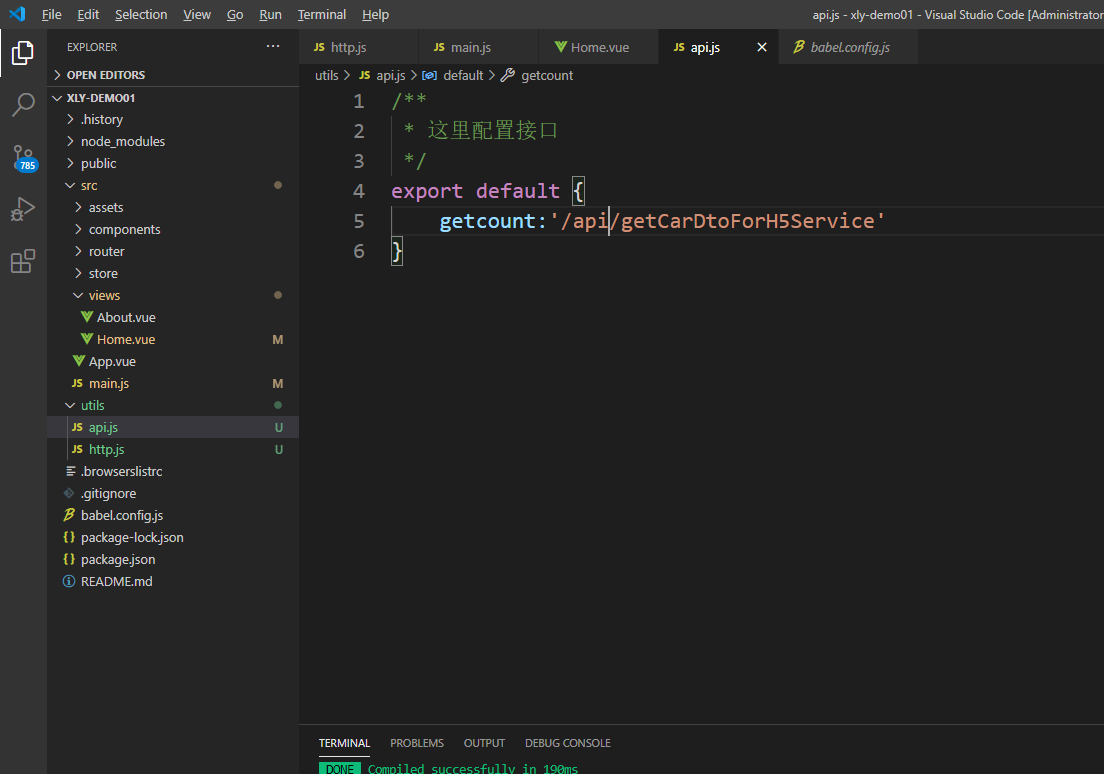
* + - * 1. 响应拦截：根据服务器返回状态码做对应处理



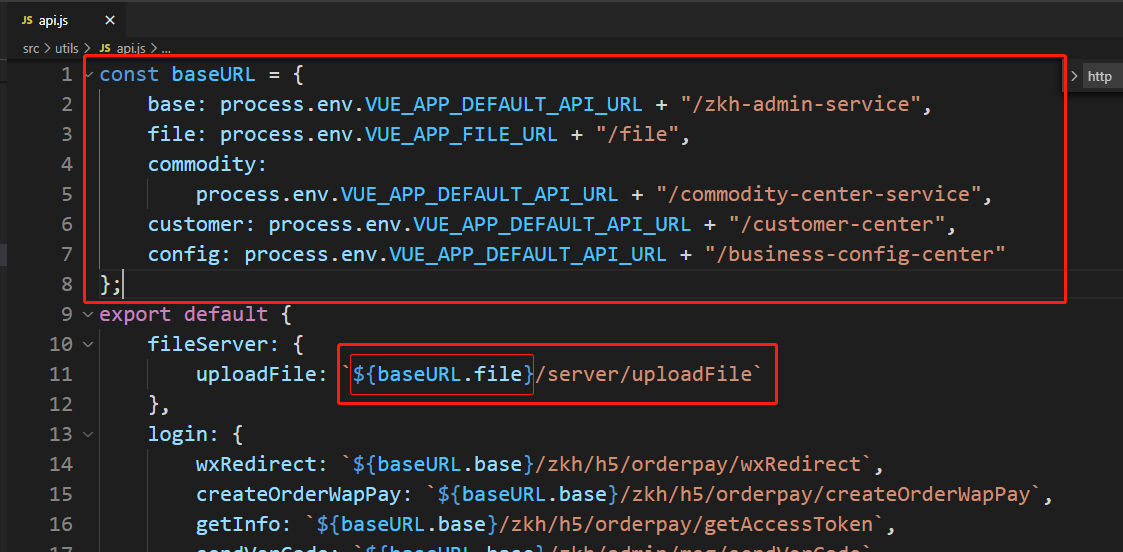


* + 1. api封装

将接口地址配置在api文件里，不具体介绍



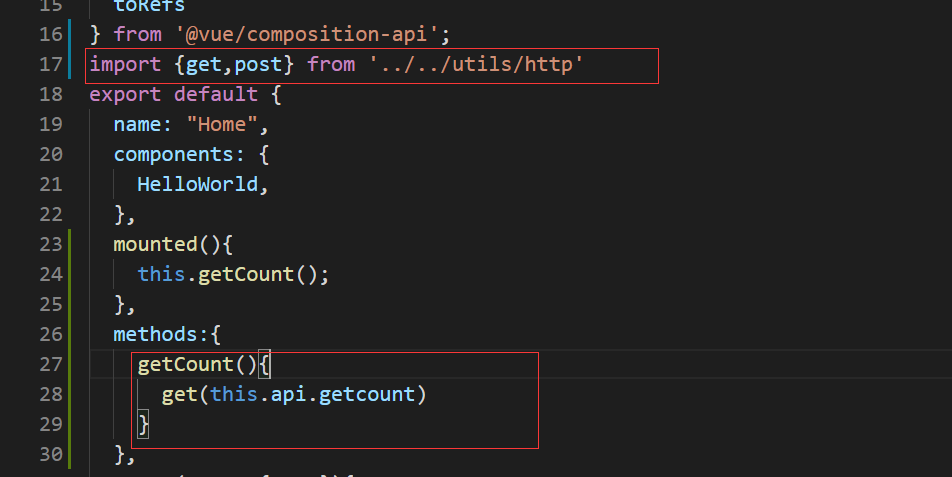
如果是分布式项目的话，可以将IP地址和接口统一配置在api.js里面。



* + 1. 引用这两个文件
       1. 在main.js里面引用



* + - 1. 组件里引用

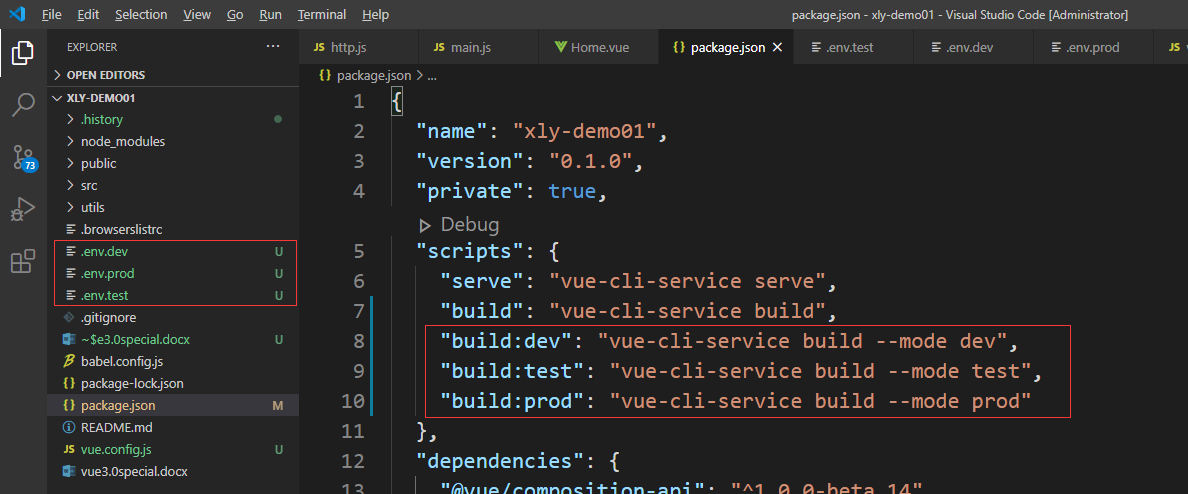


学习的网址：<https://www.cnblogs.com/chaoyuehedy/p/9931146.html>

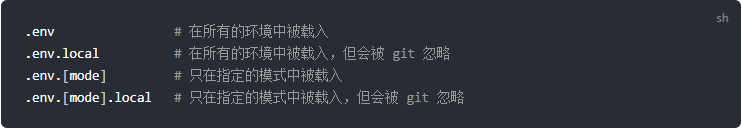
1. 配置文件
   1. vue-cli3.0配置不同的网址，可参考官网详细讲解

<https://cli.vuejs.org/zh/guide/mode-and-env.html#%E6%A8%A1%E5%BC%8F>

* + 1. 根据不同的命令，切换不同的api信息
       1. package.json中配置



* + - 1. 然后根目录下新建.env文件，通过为 .env 文件增加后缀来设置某个模式下特有的环境变量



* + - 1. --mode staging:指定模式



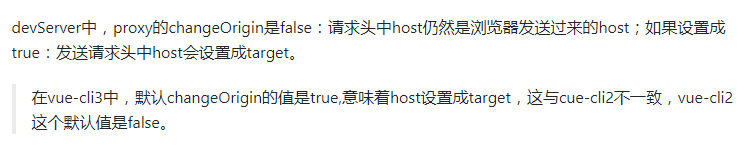
* + - 1. .env文件下的两个变量
         1. NODE\_DEV:是development,production,test的其中一个
         2. BASE\_URL:会和 vue.config.js 中的 publicPath 选项相符，即你的应用会部署到的基础路径。
         3. 在代码中访问 process.env.NODE\_DEV
  1. 配置vue.config.js

<https://www.cnblogs.com/jkko123/p/10234362.html>

可看官网配置：

<https://cli.vuejs.org/zh/config/#%E5%85%A8%E5%B1%80-cli-%E9%85%8D%E7%BD%AE>

* + 1. 和package.json会被 @vue/cli-service 自动加载,按照package.json的格式来写
    2. Source Map:为了解决文件打包后查看报错信息所在原始位置的问题，Source map就是一个信息文件，里面储存着位置信息。也就是说，转换后的代码的每一个位置，所对应的转换前的位置。
    3. devServer.proxy



* + 1. pathRewrite



* + 1. webpack alias:文件别名
       1. 在webpack项目中，我们经常会设置alias来引入文件，避免文件路径写的过长过深，但是使用alias的时候会发现路径和函数的智能提示不见了，如果路径名称很复杂的话很容易写错而且也不方便

1. 项目问题
   * 1. 如何保持用户登录状态，各种储存token的方式

<https://blog.csdn.net/kevinfan2011/article/details/95166073>