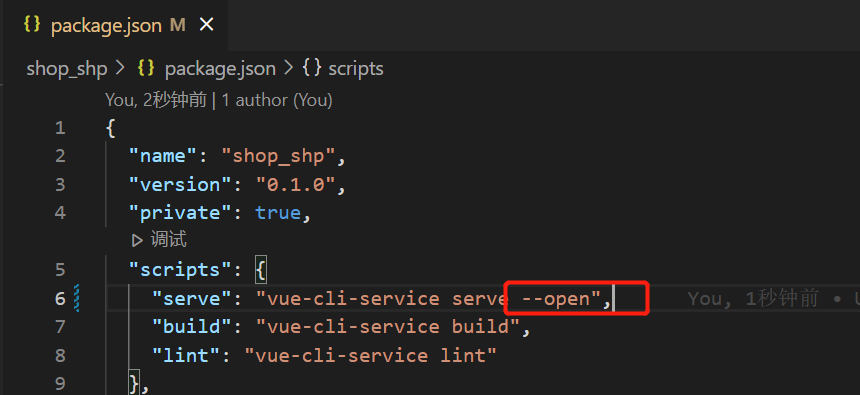
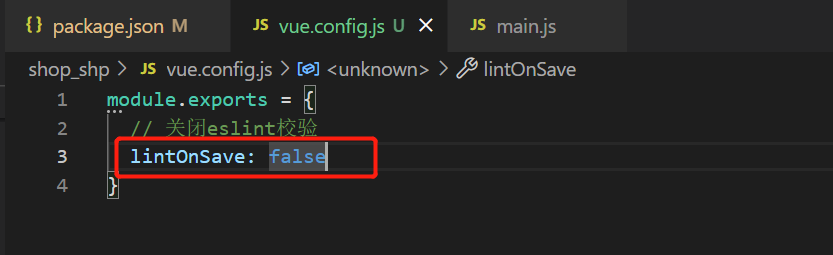
下次开发练习重点：

1. 事件委派+data-xxx [18](#_18.编程式导航+事件委派：利用自定义属性确认a标签和传参——even)
2. ES6语法：Object.assign（obj1，obj2，，，） [27](#_27.Object.assign（）：------ES6语法，用于合并)
3. ES6语法：arr.every（item=>{}） [50](#_50.ES6，数组的every方法，判断每个元素是否符合某条件——表格)
4. Watch中直接监听$route的变化 [28](#_28.Watch中可以直接监听_$route的变化)
5. scrollBehavior [35](#_35.scrollBehavior：解决_上一页面滚动到底，跳转到下一)
6. utils文件夹的使用 [45](#_45._uuid：使用uuid生成唯一token) [57](#_57.Vuex存储数据不是持久化的——刷新数据丢失——刷新token丢)

### 1.项目配置：项目启动后浏览器自动打开



### 2.项目配置：关闭eslint校验



### 3.项目配置：src路径配置为@

创建jsconfig.json文件

{

"compilerOptions": {

"baseUrl": "./",

"paths": {

"@/\*": [

"src/\*"

]

}

},

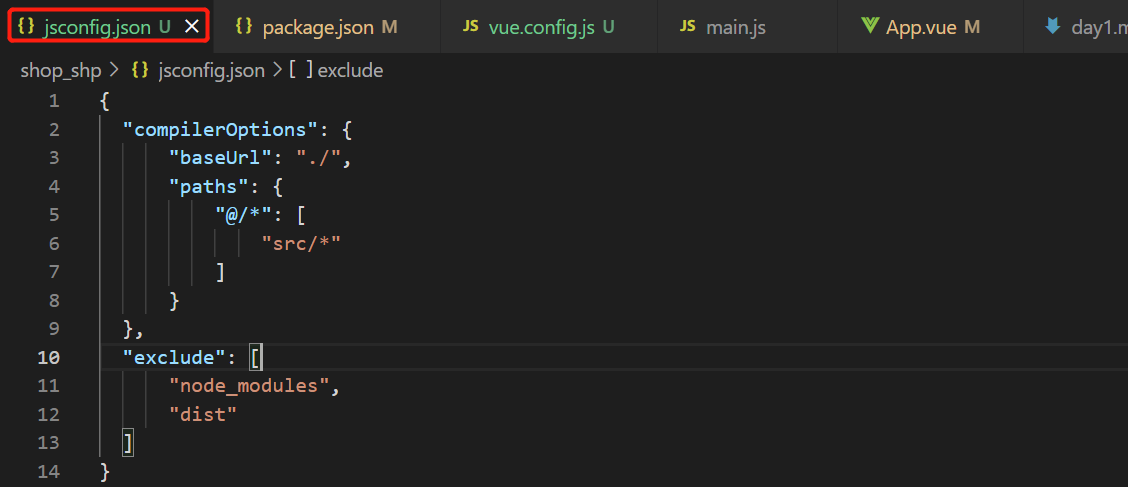
"exclude": [

"node\_modules",

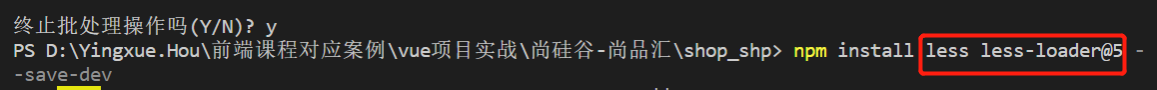
"dist"

]

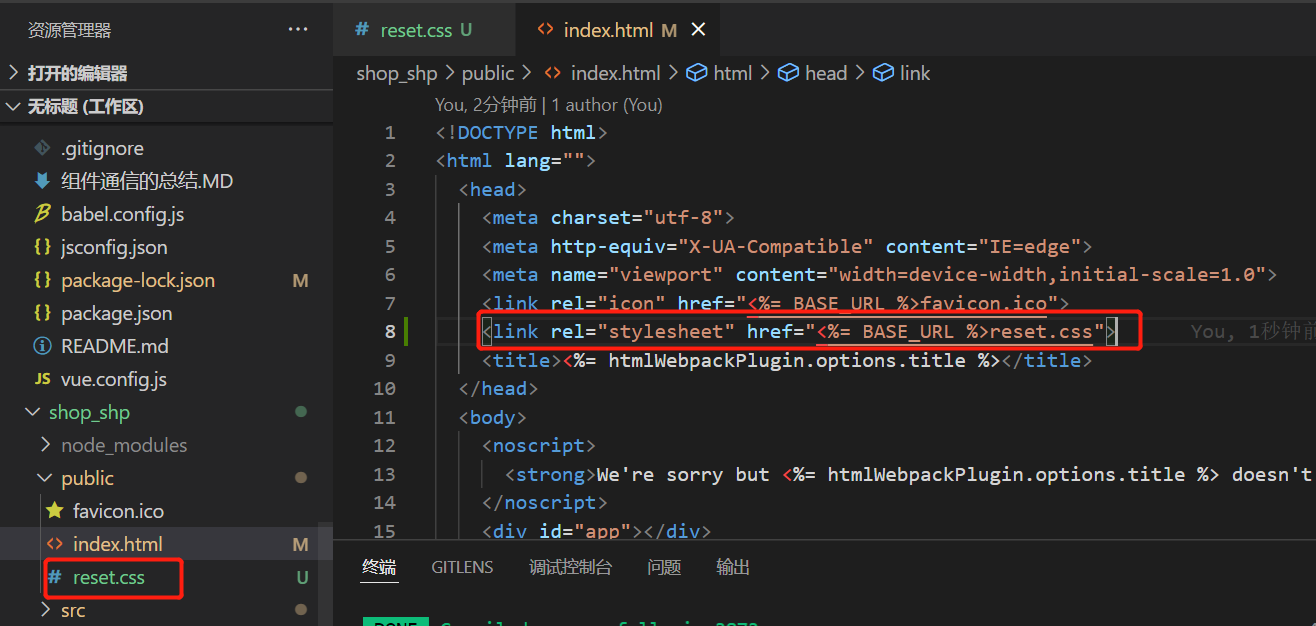
}



### 4.项目配置：less-loader安装5版本，版本太高会报错



### 5.全局文件：reset.css>public中，index.html引入



### 6.路由配置：根目录/重定向



### 7.路由配置：重写push与replace方法>解决重复点击某路由，控制台报错

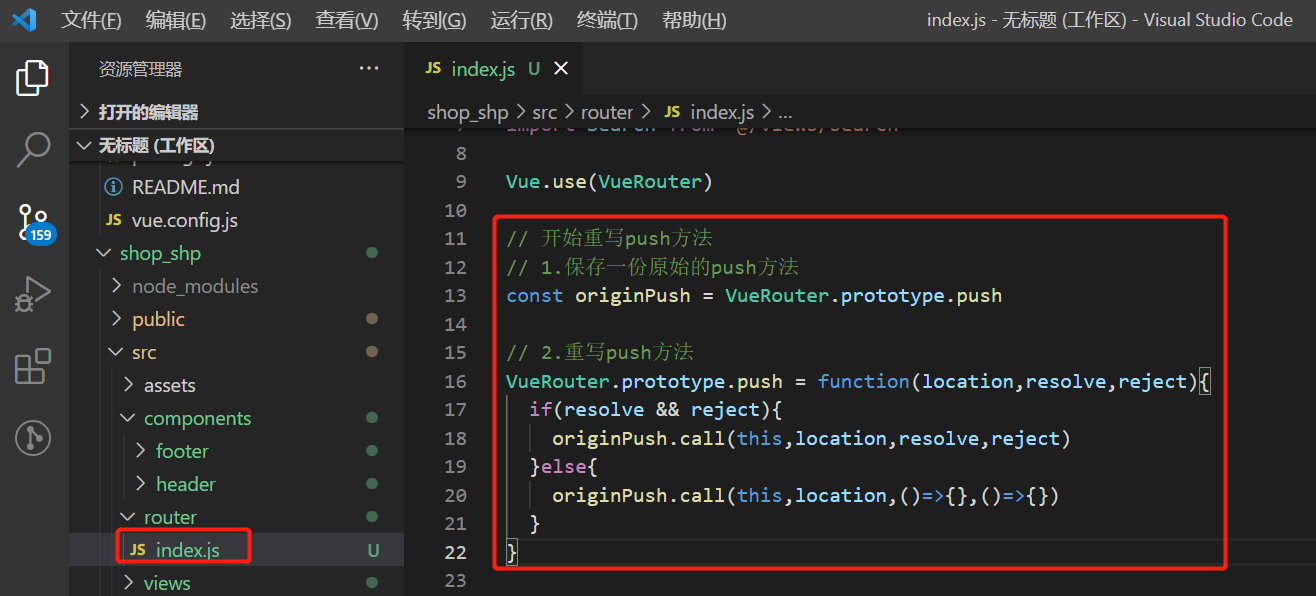
注意:编程式导航（push|replace）才会有这种情况的异常，声明式导航是没有这种问题，因为声明式导航内部已经解决这种问题。

这种异常，对于程序没有任何影响的。

为什么会出现这种现象:

由于vue-router最新版本3.5.2，引入了promise，当传递参数多次且重复，会抛出异常，因此出现上面现象





// 开始重写push方法

// 1.保存一份原始的push方法

const originPush = VueRouter.prototype.push

// 2.重写push方法

VueRouter.prototype.push = function(location,resolve,reject){

if(resolve && reject){

originPush.call(this,location,resolve,reject)

}else{

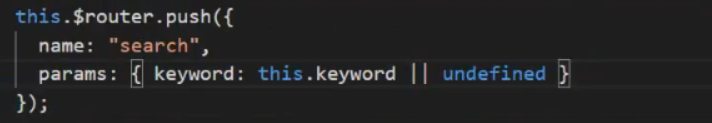
originPush.call(this.location,()=>{},()=>{})

}

}

### 8.路由传值：params参数设置可传可不传

方式1：传参时

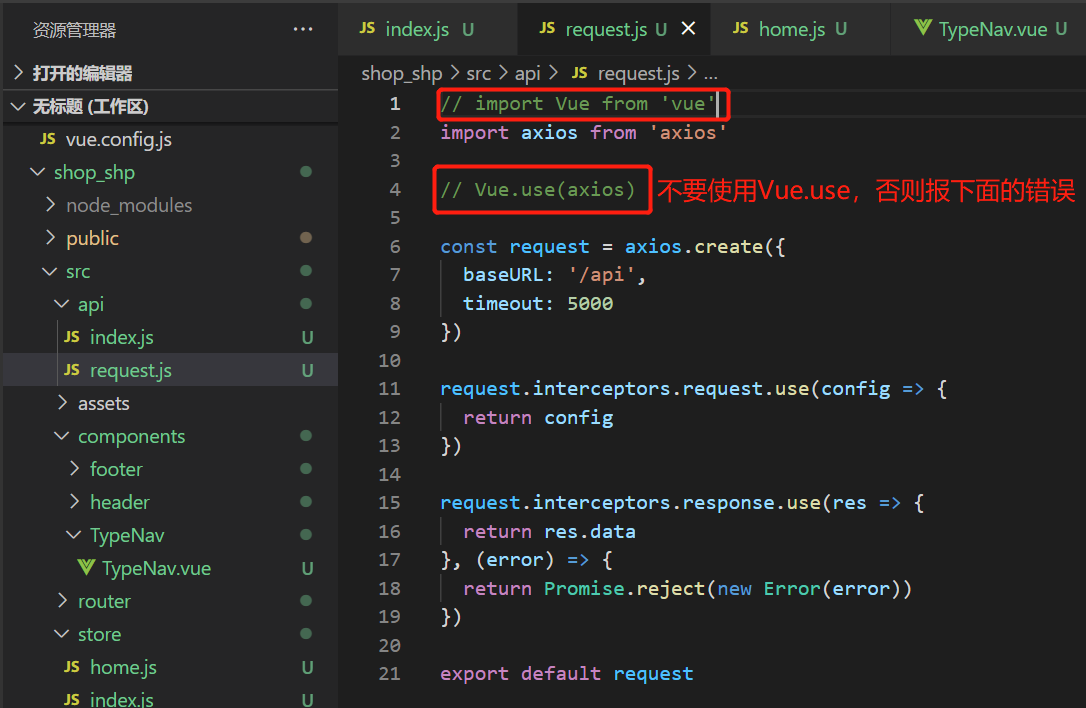


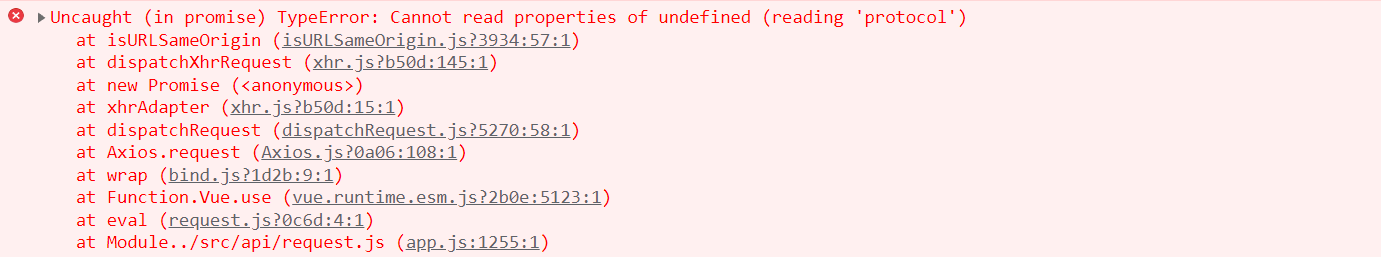
方式二：配置路由时



### 9.注册全局组件：TypeNav—三级联动导航

### 10.axios全局封装：安装axios





项目最终axios的配置：

文本

描述已自动生成

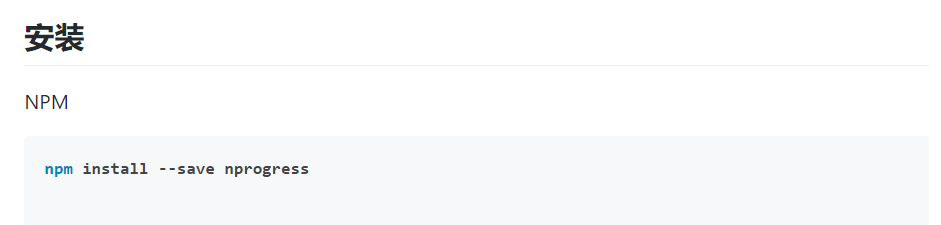
### 11.接口统一管理：proxy代理

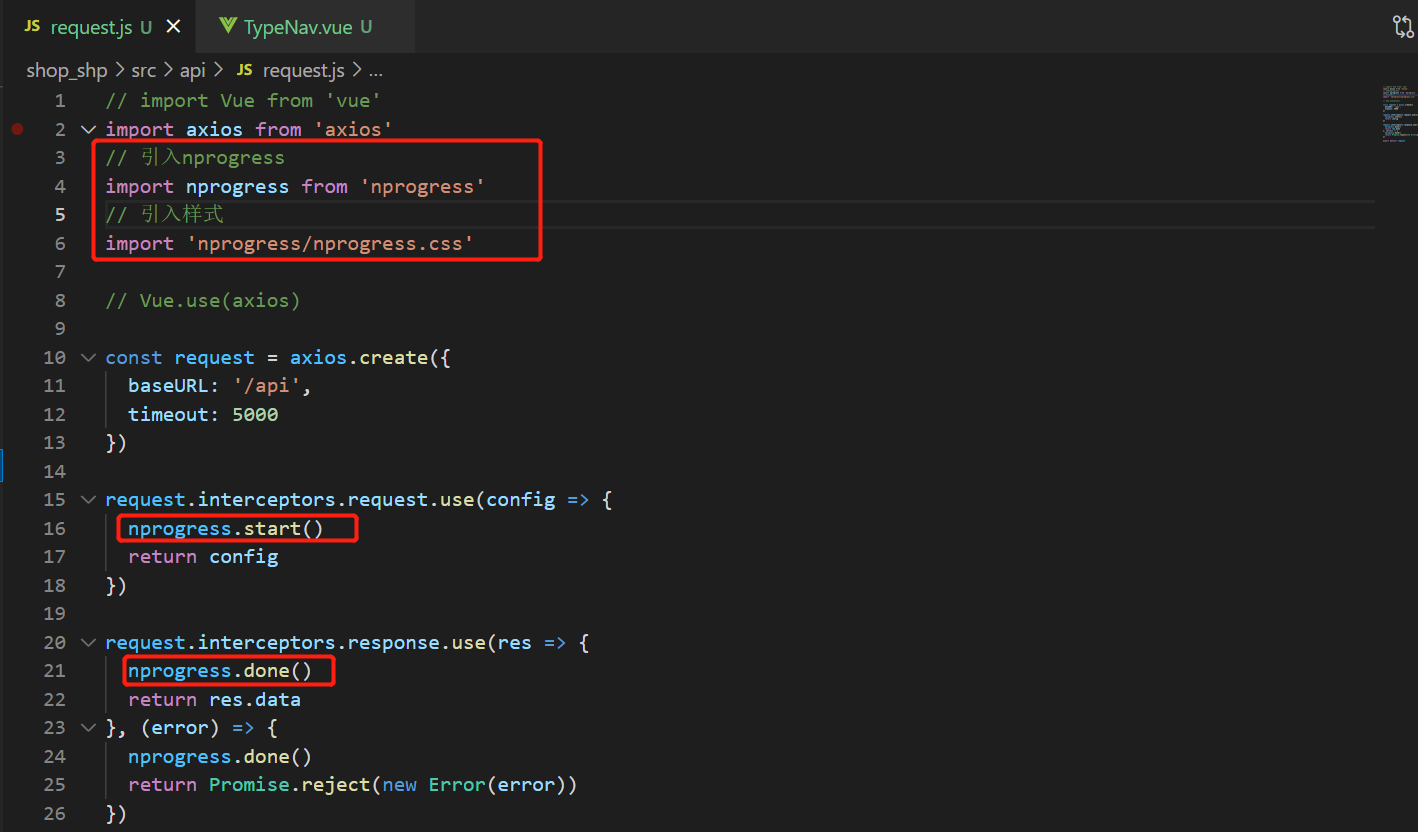


图形用户界面, 文本

描述已自动生成

### 12.Nprogress插件：实现进度条





只需要 start和done两个方法。如果想修改进度条样式，直接去修改nprogress.css文件。

### 13.Vuex配置：模块化-------vue2需要指定vuex版本为3版本



项目最终的vuex：

文本

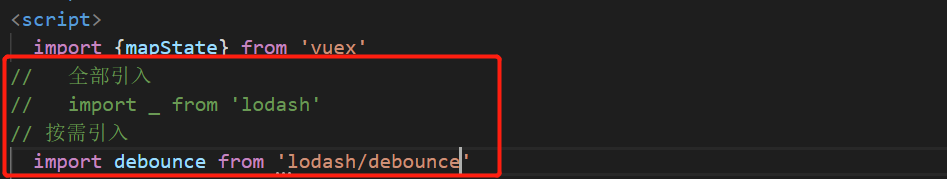
描述已自动生成

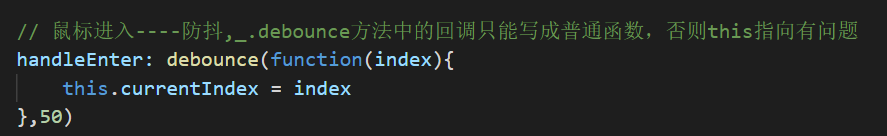
### 14.函数防抖：一定时间内同一个请求发送多次时，只发送最后一次请求，前面的请求取消

### 15.函数节流：一定时间内，某个函数最多执行一次（第一次）

### 16.lodash：里面封装函数的防抖与节流业务 \_.debounce()、\_.throttle()——按需引入

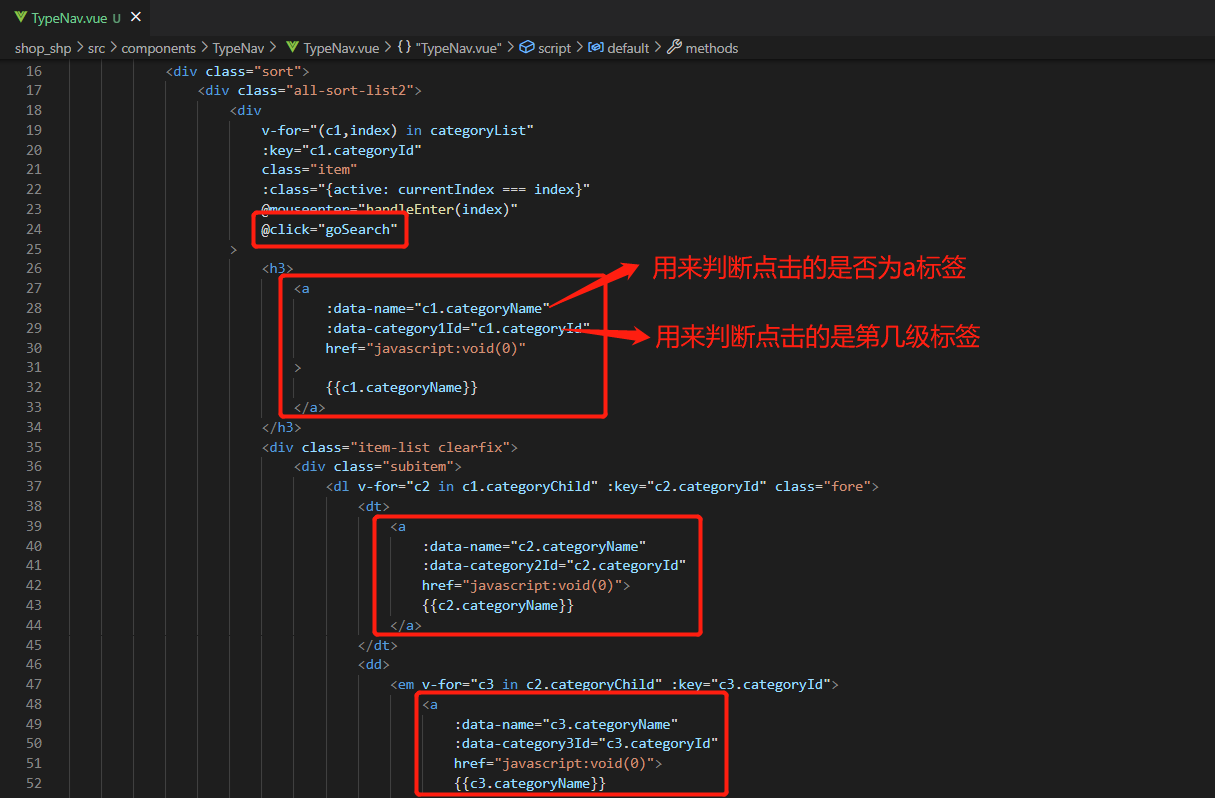
脚手架默认已经引入了lodash，记得按需引入





### 17. router-link ：<router-link>标签是一个组件（router的内置组件），三级联动组建中，如果使用声明式的方式跳转路由（<routerlink>标签），页面一加载时，可能会创建上千个组件，很耗性能——组件的创建和销毁都很消耗性能——改为编程式导航

### 18.编程式导航+事件委派：利用自定义属性确认a标签和c参数——event.dataset





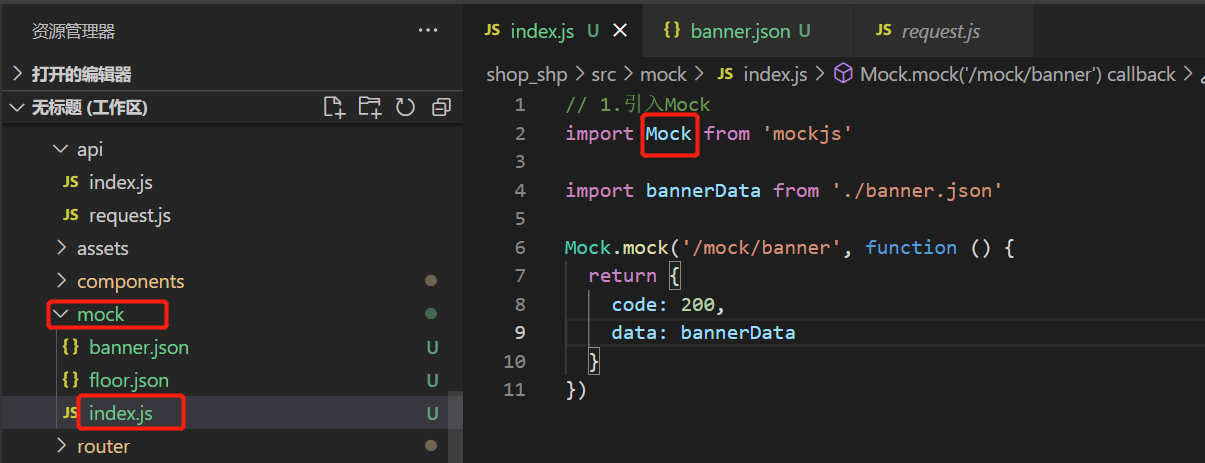
### 19.过渡动画：前提页面元素使用了 v-if 或 v-show—— <transition name=”xxx”></transition>

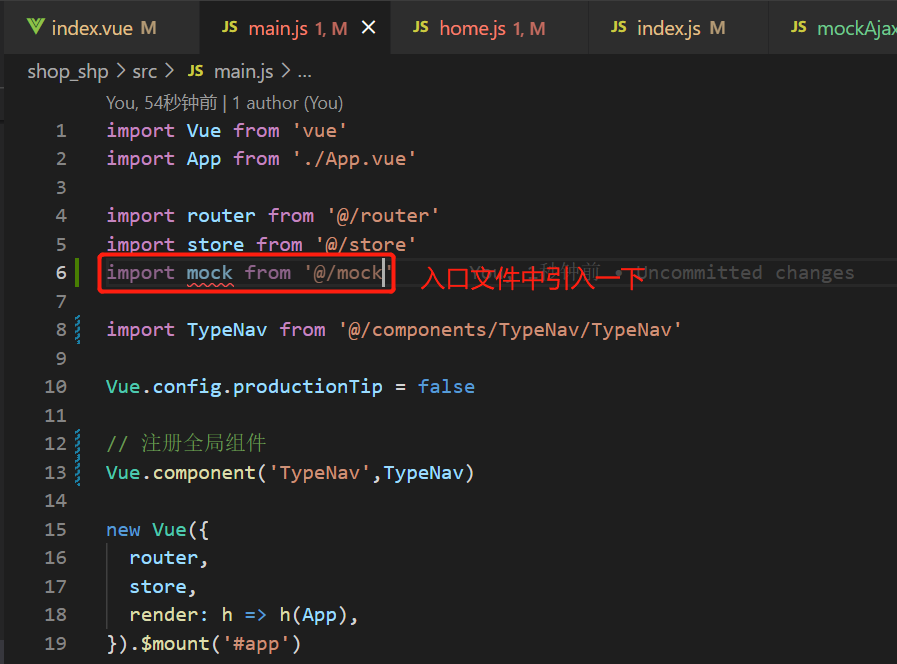
### 20.Query参数和params参数配合使用：首页跳转到search页面时传参

使用场景：点击两个位置都可以跳转到搜索页面，但是两个位置跳转时，带的参数不同。这时，可以 一个使用query传参，一个使用params传参。在搜索页面中，就监听this.$route的变化，把query参数和params参数都拿到，调用接口获取数据。

### 21. 印记中文：推荐一个网站，里面能搜索很多前端库

### 22.Mock.js的使用

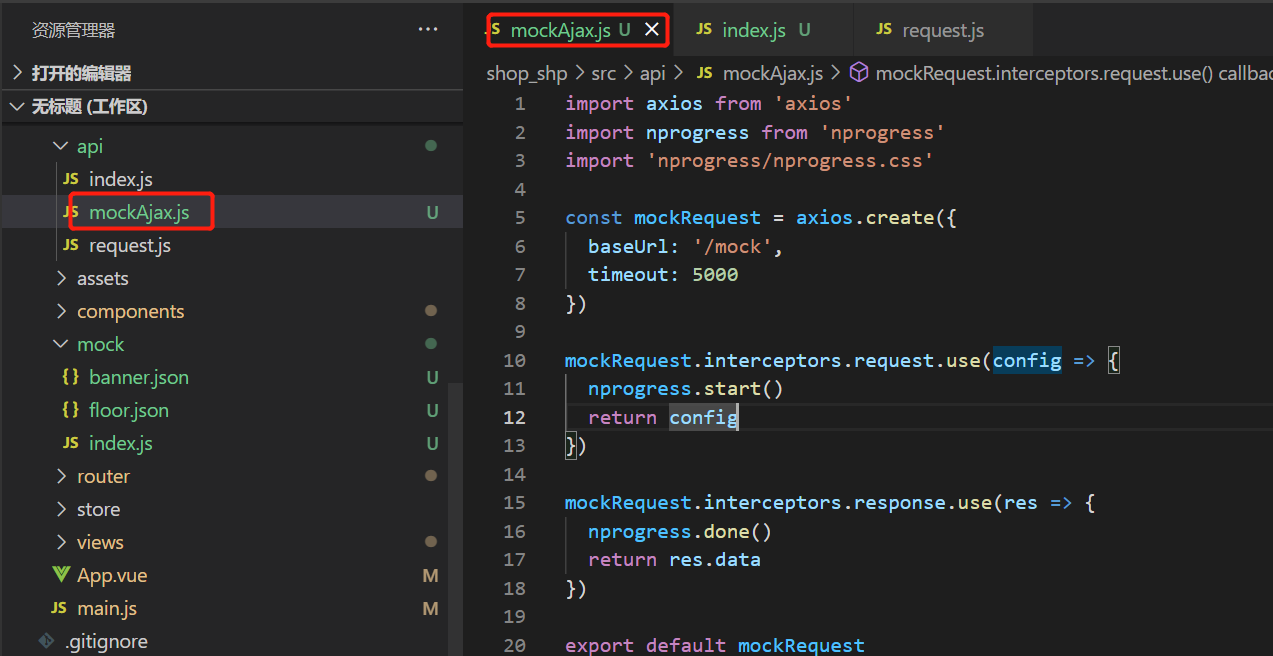


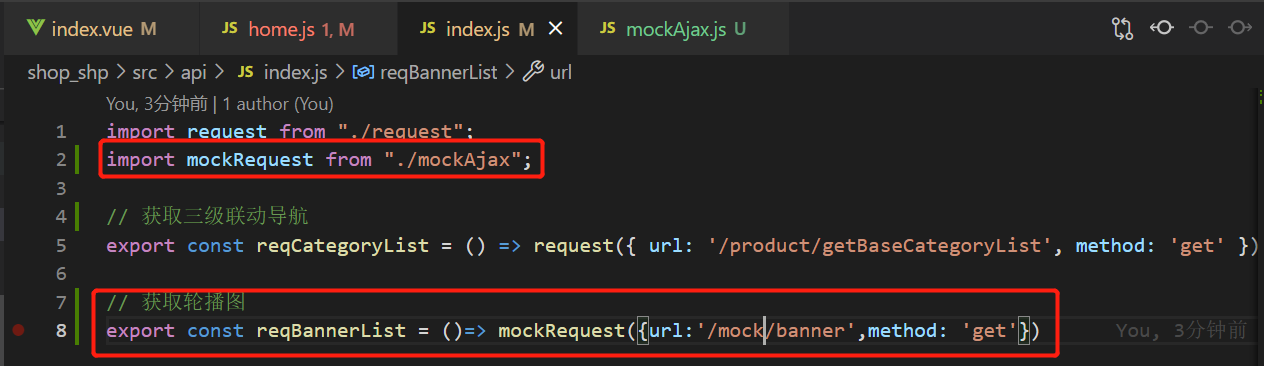


#### 1.图片、json数据格式，都是默认暴露的，不需要export暴露就能使用import引入

#### 2.针对向mock发请求的接口，创建api > mockAjax.js

因为调用mock的接口的前缀是不同的（即baseUrl），所以需要重新封装一个





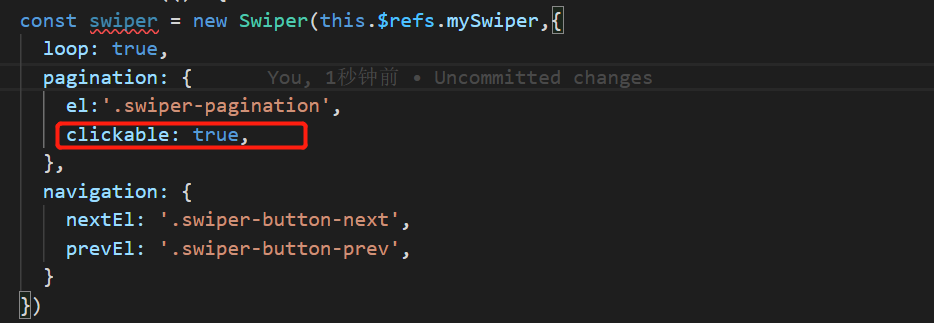
### 23.Swiper插件：PC端和移动端都适用-----最新是6版本，不够稳定，安装5版本

#### 1.new Swiper之前，必须要有结构----------v-for必须已经完整的执行完毕才行---------- wacth + $nextTick()----------将回调延迟到下次 DOM 更新循环之后执行。在修改数据之后立即使用它，然后等待 DOM 更新。----new Swiper的时机



#### 2.入口文件中引入swiper的css：两个组件中都可以用--------提取出公共组件后，可在公共组件中引入

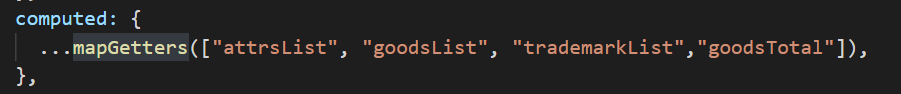
#### 3.Swiper分页器默认不可点击：需要在初始化时手动开启

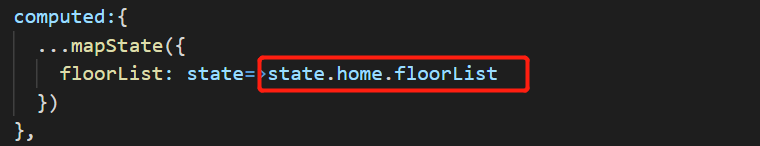


### 24.Axios发请求传参--------封装的请求需要传参时，参数至少是一个空对象

### 25.Getters作用：简化仓库中的数据

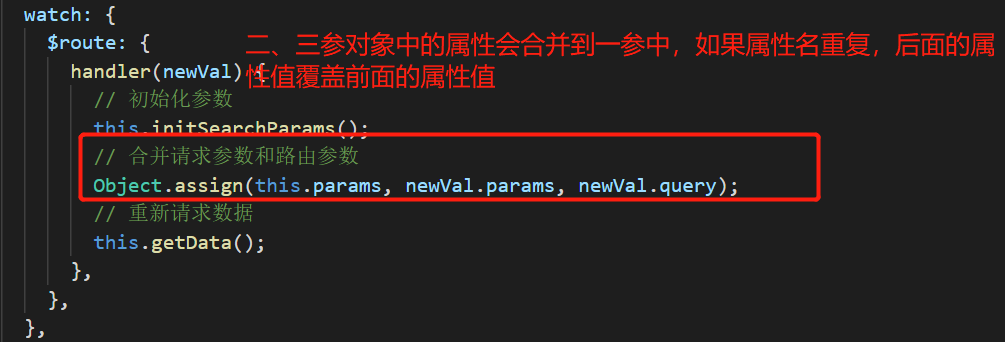
#### 1.vuex仓库中的state是分模块的，getters是不分模块的



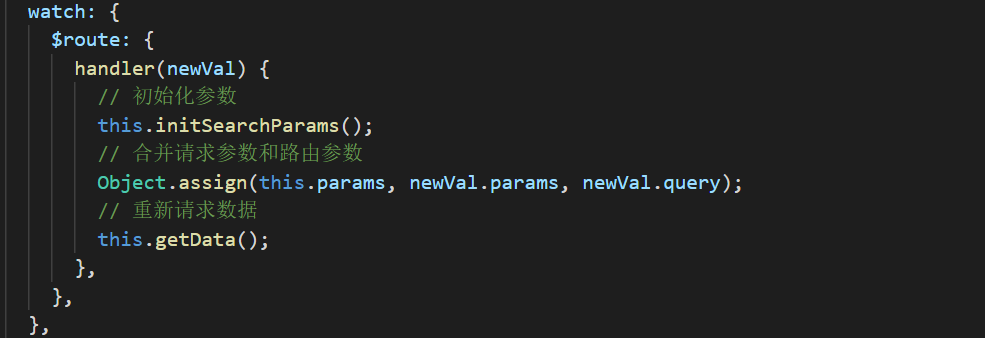


### 26.纠正之前的一个思维：在created中调接口发请求。后面尽量在mounted中调接口发请求，在created中准备好发请求时需要的参数。

### 27.Object.assign（）：------ES6语法，用于合并对象



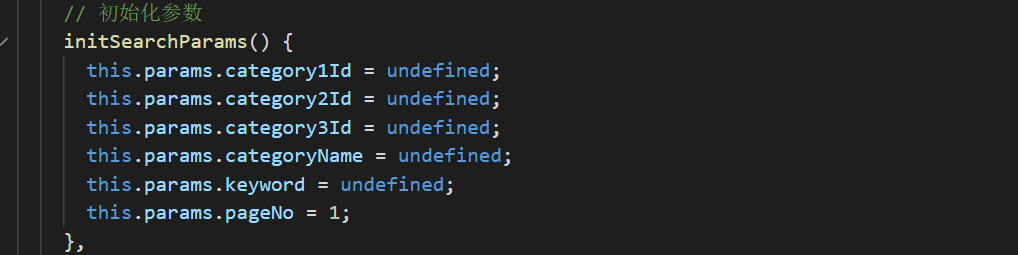
### 28.Watch中可以直接监听 $route的变化



### 29.解耦合：改参数就只改参数（单独写一个方法），发请求就只发请求（单独写一个方法），分开写，更解耦，更好控制

使用场景：根据表单中数据，获取列表数据

### 30.请求参数整理：发送axios请求传参时，把某个属性值设置为undefined，该参数就不会发送给服务器------便于整理参数



### 31.全局事件总线：实现search组件和header组件之间通信：keywords--------Vue.property.$bus=this

### 32.阿里巴巴矢量图：--------在线代码，需要自己补充https协议-----个人实验后发现不应该补充协议，补充协议后无法使用

### 33.js原生三巨头：分页、轮播、日历！！！！！！！！！

### 34.分页器功能分析：

#### 1.传入的参数：当前第几页，每页多少条，总共多少条，连续页码数（指下图，下图是连续5页）



#### 2.在开发时，先自己传假数据调试分页器，调试好之后，再使用服务器的数据

#### 3.计算页数，直接向上取整 Math.ceil(条数/每页几条)

#### 4.分页器很重要的一个东西：算出连续分页的起始位置和结束位置

--------老师提供了很妙的思路：先按正常页码 算出 连续页面的起始和结束 数字，再考虑不正常页码情况，修正刚算出的起始和结束数字。

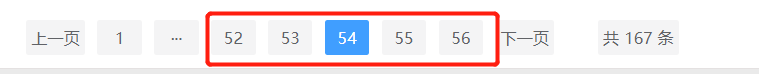
具体思路：

##### 1.开始数字的计算：

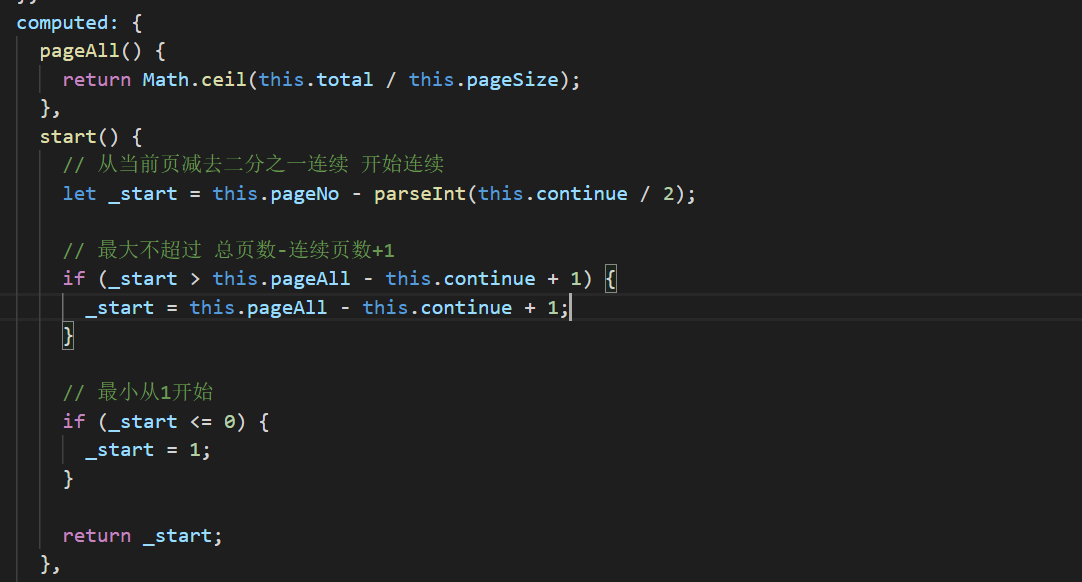
算出正常情况下的开始数字：当前第几页-二分之一连续页数（向下取整）

算出非正常情况下的开始数字：

当前页数 大于 总页数-二分之一连续页数+1 （即最后几页，如共56页，连续数为5，当前页为54,55,56时，开始数字都应该是52）



如果计算出的开始数字小于1，则应该设为1



1. 开始计算结束数字：

思路1：（主要计算开始数字）

如果总页数小于连续数，开始等于1，返回——否则，如果开始数小于0，开始数等于1，返回——否则，如果开始数大于总页数-二分之一连续数，开始数等于总页数-连续数—+1，返回

思路2：（前提是已经精确计算出了开始数字【在思路1的基础上】）

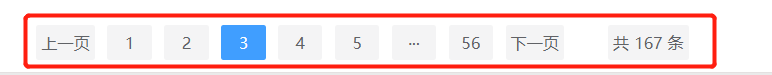
1. 结束数字等于 开始数组+连续页数-1
2. 如果结束数字大于总页数，结束数字置为总页数

思路3：（计算正常情况下的结束数字，再考虑极端情况下的结束数字）

算出正常情况下的结束数字：当前页 + 二分之一连续页（向下取整）

算出极端情况下的结束数字：

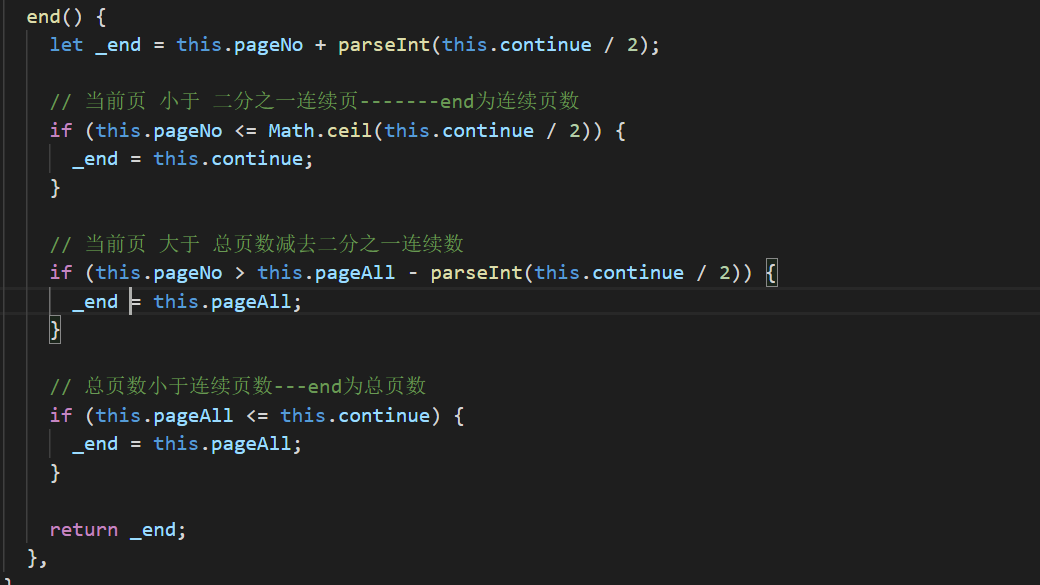
当前页小于 二分之一连续页（向下取整）时，结束数字应为连续页数（如下，1,2,3页时，结束数字总为5）



当前页 大于 总页数-二分之一连续页数，结束数字总为总页数：（如下，54,55,56页，结束数字总为56）



总页数小于连续页数时，结束数字应为总页数



#### 5.分页的静态结构分成上中下三部分，动态实现分页时，先考虑分页中间部分

#### 6.Buttom标签的disable属性利用起来

### 35.scrollBehavior：解决 上一页面滚动到底，跳转到下一页后，滚动条位置不能回到顶部的问题：路由滚动行为



### 36.总结：一个页面 开发流程：静态页-》组件拆分-》请求数据-》vuex三步走-》页面中派发action获取数据-》页面数据动态展示

### 37.使用arr[0].xxx时，记得给数组赋初始值[]，否则，在获取到数组之前会有假报错

### 38.做“排他”操作时，可以把整个数组作为实参传入，方便操作——摒弃之前只传id或者从根源开始遍历的思维

图形用户界面, 应用程序

描述已自动生成

### 39.Swiper配置同时显示三张图片：sliderPerView---------配置每次切换几张图片：sliderPerGroup

文本

描述已自动生成

### 40.放大镜效果 结构及思路：

截图里有图片

描述已自动生成

文本

描述已自动生成

### 41.输入框只能输入数字：非数字的字符串\*1=NaN：可用于排除输入框中输入的非法字符串（非数字的字符串）

### 42.async返回值：如果一个函数前面使用了async关键字，那么，这个函数默认返回一个Promise。

43.try-catch：可以使用try。。。catch来捕捉 Promise.reject（new Error（‘falil’））抛出的错误。

——这个写法针对的是 调用接口后，需要知道成功、失败两种结果。比如：删除某数据，不会有返回值，只需要提示“删除成功”或“删除失败”

文本

描述已自动生成

电脑萤幕的截图

描述已自动生成

### 43.sessionStorage：路由传参 需要传 对象参数时，使用query方式传参，把对象转换为字符串（JSON.stringify（）），但是仍然不推荐——推荐使用会话存储！！！——简单数据通过query形式路由传参，复杂参数通过会话存储传递——路由跳转需要传递多个参数时（有些是简单参数，如数字、字符串，有些是复杂参数，如对象、数组），要有这种意识：简单参数使用路由传参传递，复杂参数通过 会话存储传递——合理选择更高效。

### 44.localStorage：持久化，有上限，大概5M sessionStorage：非持久化——会话结束，数据消失。——两种存储方式，存的都是字符串JSON.stringify（）

### 45. uuid：使用uuid生成唯一token

图形用户界面, 应用程序, 网站

描述已自动生成

#### 1.Vue-cli脚手架默认已经下载了uuid

### 46.添加购物车时的逻辑分析：

之前 添加购物车的操作是调用接口，并且接口只接收 商品id和数量 两个参数。我们在购物车页面 调用接口获取购物车列表时，返回的数据会有问题：因为添加购物车的时候，我们没有标明身份（我们的用户名），所以在获取购物车列表时，数据库不知道你是谁，也就不知道要返给你哪些数据。所以，我们在加入购物车时，应当向接口表明我们的身份。但是，接口只接收两个参数，我们怎么才能把我们的身份信息（uuid生成）传递过去呢？——请求头！！！！！

### 47.localStorage+uuid：一旦生成了你的身份（uuid），那么每次登陆时你的身份不能变（单例）——localStorage——从本地存储中获取token（uuid生成），有则拿来用，没有就重新获取（通过调用uuid的方法）——token（uuid）应该放在vuex中存储。——这是游客登录的token

电脑萤幕的截图

描述已自动生成

电脑萤幕的截图

描述已自动生成

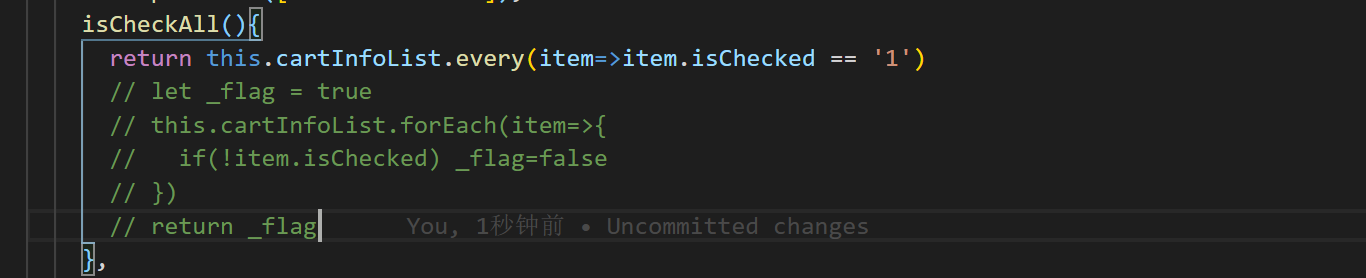
### 48.utils文件夹：新建utils文件夹，用来放置一些功能函数——使用uuid生成token的

### 49.Token的使用：在请求拦截中，给请求头中添加token（对应的字段名需要和后台商量好）

文本

描述已自动生成

### 50. 数组的every方法：ES6，数组的every方法，判断每个元素是否符合某条件——表格复选框 全选、全不选时可用



### 51.修改数量，需要重新发请求——节流





### 52. Promise.all()：删除全部商品 的思路：Promise.all()

前提是 全选复选框被勾选上 我们只有一次删一件商品的接口（在action中调用），接口接收一个参数：商品id。删除全部商品时，只需要在action中定义一个方法，遍历Vuex中购物车列表数组，循环调用删除一个商品的方法即可。

同时，需要配合 Promise.all方法，全删成功才是成功

文本

描述已自动生成

文本

描述已自动生成

### 53.Assets文件夹中，放置的是所有组件公用的静态资源

### 54.之前配置的src文件夹路径的别名@，在css中也可以使用，需要在前面加~号

### 55.Axios传参使用data和params的问题

对get请求来说，data和params参数都会拼接到请求接口地址中

对post请求来说，params参数会拼接到请求接口地址中，data参数会在请求体中

最好都都使用data参数

### 56.登陆：一般登陆成功后，服务器会下发token（令牌，用户唯一标识）——前台需要持久化存储token——本地存储前端拿着token去换取用户信息（通过调接口）。——后面所有的接口调用之前（请求拦截中），先去看vuex中有没有token，有则添加到请求头中，带到后台去。

### 57.Vuex存储数据不是持久化的——刷新数据丢失——刷新token丢失——action中调用登陆接口拿到token后，存放到 本地存储中一份，并调用commit设置state中的token值 —— state中的token直接从本地存储中取

图形用户界面, 应用程序

描述已自动生成

文本

描述已自动生成

### 58.获取用户信息的时机：前置路由守卫中——继续往下看

### 59.登陆成功后，不应该通过回退按钮回退到登录页——转变之前的思维：之前我认为这个应该是通过this.$router.replace控制的，但其实这里应该是使用前置路由守卫控制，通过判断to.path是否为login页面 来实现——继续向下看，下面会统一截图

### 60.路由守卫：前置路由钩子——next（）：直接放行——next(路径)：放行到指定路由页面——next（false）：去查阅官网api，不常用，意思是从哪儿来回哪儿去

### 61.路由守卫在项目中的应用场景：1.）有些页面（购物车）只有登陆才能查看，否则直接跳转到登陆页面去，让用户登陆，登陆成功后，直接跳转到购物车页面（而不是home页） 2）如果用户已经登陆了，就不能再去到登陆页面——to.path判断 3）重新获取用户信息——继续往下看

### 62.捋一下前置路由守卫中的逻辑：

1. 先去vuex中拿token，有则已登录，无则未登录
2. 如果有token（已登录）：
3. 不能再跳转到登陆页面（如果被发现，直接放行跳转到home页去）
4. 如果要去的不是登陆页面，去vuex中拿用户信息（用户名）
5. 如果有用户信息，放行
6. 如果没有用户信息，拿着token调接口去获取用户信息（派发actions）
7. 如果获取用户信息成功，放行
8. 如果获取用户信息失败，说明登陆过期（token过期），退出登录（派发actions，这个actions中做了这几个操作：调接口通知服务器清除token、清除vuex和本地存储中的token、清除vuex中的用户信息、跳转到登陆页面，跳转到登录页面），放行
9. 其他情况，放行
10. 如果没有token（未登录）：
11. 如果跳转到登陆、注册页面，放行
12. 如果跳转到购物车等页面，放行到登录页
13. 如果想跳转到其他正常页面，放行

文本

描述已自动生成

### 63.为什么不直接把用户信息也放到本地存储：

也就是登陆成功后，拿到了token，直接拿token去获取用户信息，然后把用户信息直接存到本地存储中，就不用每次刷新页面后再拿token去调接口获取用户信息了——

第二版回答：第一版回答（下面部分）是想当然了，不把用户信息放到本地存储中的主要原因，是因为会造成用户信息泄露，不安全，所以最好用户信息还是存储到vuex中去。

*第一版答：在这个案例中，这样写就没办法判断token是否过期了：如果拿token获取用户信息失败，则代表token过期，即登陆过期，需要重新登陆。如果只登陆一次之后，直接就把用户信息存在本地，每次刷新页面后从本地取用户信息，那么要根据哪个时机判断登陆过期呢？——这个解释仅限本项目。如果其他项目中，token过期后调任何接口都会失败（比如和后端规定好，如果token过期，就返回203），那么完全是可以在登陆成功后，拿着token获取用户信息并将用户信息存到本地的。Token过期后，只需要将本地和仓库中的token、用户信息都清除掉，再跳转到登录页面重新登陆即可，就不用再像本项目中，在用户登陆且没有用户信息的前提下（即刷新页面的），在路由前置守卫中去获取用户信息了，直接将本地存储中的用户信息赋值给vuex即可。*

### 64.退出登陆逻辑：向服务器发请求，通知服务器清除token；清除本地的token、userInfo信息



### 65.ES6中，数组的find方法：查找符合条件的（第一个）元素并返回这个元素

### 66.练习项目中没有vuex

### 67.路由传参，使用props

### 68.项目中不允许给生命周期函数加async

### 69.Elementui的使用：支付

### 70.安装vue-helper插件——友好提示

### 71.Qrcode：可以生成二维码的插件

### 72.支付的问题：二维码生成之后，前端需要一直向后端发请求，直到确认已支付或支付失败