尚品汇总结：

https://blog.csdn.net/weixin\_43424325/article/details/121684101

## **前端Vue核心**

开发一个前端模块可以概括为以下几个步骤：  
（1）写静态页面、拆分为静态组件；  
（2）发请求（API）；  
（3）vuex（actions、mutations、state三连操作）；  
（4）组件获取仓库数据，动态展示；

## **1、vue文件目录分析**

****public文件夹****：静态资源，webpack进行打包的时候会原封不动打包到dist文件夹中。

****src文件夹****（程序员代码文件夹）

assets： 存放公用的静态资源

components： 非路由组件（全局组件），其他组件放在views或者pages文件夹中

App.vue： 唯一的跟组件

main.js： 程序入口文件，最先执行的文件

****babel.config.js****: 配置文件（babel相关）  
****package.json****: 项目的详细信息记录  
****package-lock.json****: 缓存性文件（各种包的来源）

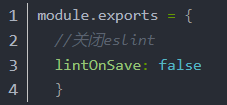
## **项目配置**

项目运行，浏览器自动打开

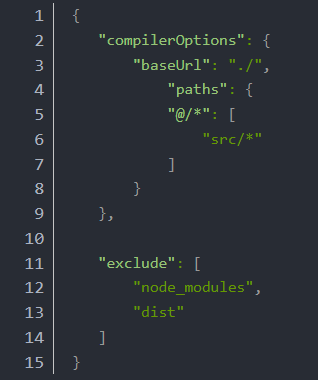
"serve": "vue-cli-service serve --open",

关闭eslint校验工具（不关闭会有各种规范，不按照规范就会报错）

目录下创建vue.config.js,进行配置



src文件夹配置别名,创建jsconfig.json，用@/代替src/，exclude表示不可以使用该别名的文件



## **组件页面样式**

使用的是less样式，浏览器不识别该样式，需要下载相关依赖

npm install --save less less-loader@5  
如果想让组件识别less样式，则在组件中设置  
<script scoped lang="less">

## **清除vue页面默认的样式**

需要修改public下的index.html文件

<link rel="stylesheet" href="reset.css">

## **pages文件夹**

创建router文件夹，并创建index.js进行路由配置，最终在main.js中引入注册

路由组件和非路由组件区别：

非路由组件放在components中，路由组件放在pages或views中

非路由组件通过标签使用，路由组件通过路由使用

在main.js注册玩路由，所有的路由和非路由组件身上都会拥有$router $route属性

$router：一般进行编程式导航进行路由跳转

$route： 一般获取路由信息（name path params等）

路由跳转方式

声明式导航router-link标签 ,可以把router-link理解为一个a标签，它 也可以加class修饰

编程式导航 ：声明式导航能做的编程式都能做，而且还可以处理一些业务

## **6、footer组件显示与隐藏**

footer在登录注册页面是不存在的，所以要隐藏，v-if 或者 v-show

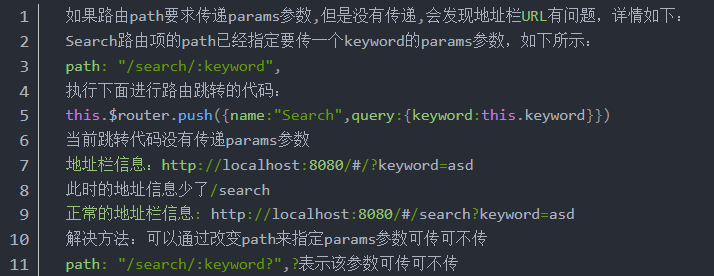
使用v-show，因为v-if会频繁的操作dom元素消耗性能，v-show只是通过样式将元素显示或隐藏

## **7、路由传参**

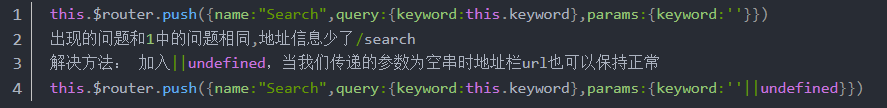
query、params

query参数：不属于路径当中的一部分，类似于get请求，地址栏表现为 /search?k1=v1&k2=v2

params参数：属于路径当中的一部分，需要注意，在配置路由的时候，需要****占位**** ,地址栏表现为 /search/v1/v2



如果传递的是空串，如何解决 。



件能不能传递props数据？

可以，三种方法

· **布尔模式**：适用于直接将路由参数传递为组件 props。

· **对象模式**：适用于传递静态的 props 数据给组件。

· **函数模式**：适用于根据路由信息动态生成 props 数据，并传递给组件。

传参方法

字符串形式

this.$router.push(“/search/”+this.params传参+“?k=”+this.query传参)

模板字符串

this.r o u t e r . p u s h ( " / s e a r c h / + router.push("/search/+router.push("/search/+{this.params传参}?k=${this.query传参}")

注意： 上面字符串的传参方法可以看出params参数和’/'结合，query参数和？结合

http://localhost:8080/#/search/asd?keyword=asd

上面url中asd为params的值，keyword=asd为query传递的值。

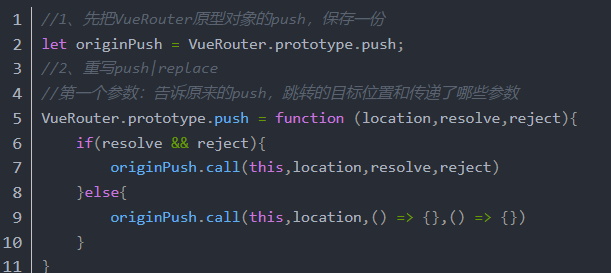
对象（常用）

this.$router.push({name:“路由名字”,params:{传参},query:{传参})。

以对象方式传参时，如果我们传参中使用了params，只能使用name，不能使用path，如果只是使用query传参，可以使用path 。

## **多次执行相同的push问题**

多次执行相同的push问题，控制台会出现警告，push是一个promise，promise需要传递成功和失败两个参数，我们的push中没有传递。在router中的index重写该方法即可

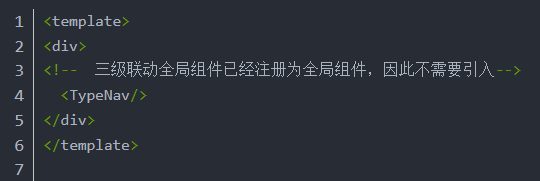


## **定义全局组件**

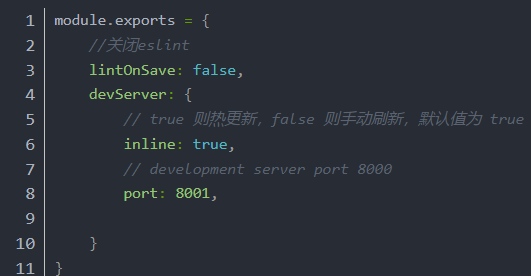
三级联动组件是全局组件，全局的配置都需要在main.js中配置



Home组件中使用该全局组件



## **10、代码改变时实现页面自动刷新**



## **封装axios**

在根目录下创建api文件夹，创建request.js文件。

可以配置请求拦截器和响应拦截器



## **前端通过代理解决跨域问题**

在封装axios的时候已经设置了baseURL为api,所以所有的请求都会携带/api

webpack-dev-server 的 proxy 选项配置的就是这样一个代理，它接收前端发送的请求，然后将请求转发给目标服务器



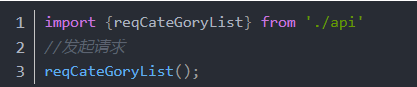
## **请求接口统一封装**

文件夹api中创建index.js文件，用于封装所有请求

****每个请求封装为一个函数，并暴露出去，组件只需要调用相应函数即可，这样当我们的接口比较多时，如果需要修改只需要修改该文件即可。****



当组件想要使用相关请求时，只需要导入相关函数即可，



## **nprogress进度条插件**

**我们发起请求的时候开启进度条，在请求成功后关闭进度条，所以只需要在request.js中进行配置**

**在请求拦截那里添加**



## **手动引入vuex**

main.js:  
(1) 引入文件  
(2) 注册store

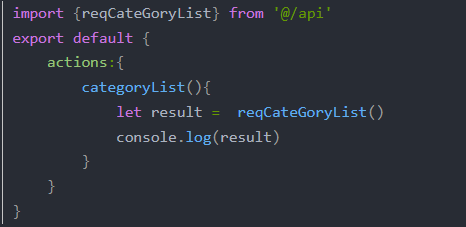
****在main.js中的Vue实例中注册的实体，在所有的组件中都会有（this.$.实体名）属性****



## **17、async await使用**

****async/await 提供了一种更简洁、直观的方式来处理异步操作，尤其是在处理多个异步任务或需要更复杂的错误处理时，async/await 能够让代码看起来更像是同步执行的，从而提高代码的可读性和可维护性。****

****比如例子中如果我们直接调用：****

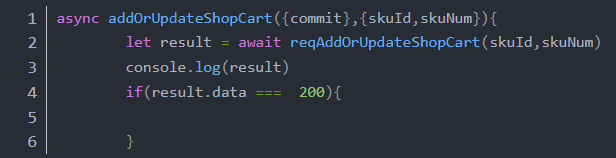


****获得的是promise，但是我们想要的是图片中的data数据。****

使用then的话还会让后面代码先执行，所以使用async和await，await含义是****async标识的函数体内的并且在await标识代码后面的代码****先等待await标识的异步请求执行完，再执行。这也使得只有reqCateGoryList执行完，result 得到返回值后，才会执行后面的输出操作。

## **18、vuex**

注意actions时不支持多参数传递，如果需要传递多个参数，需要把多个参数组合为一个对象传入



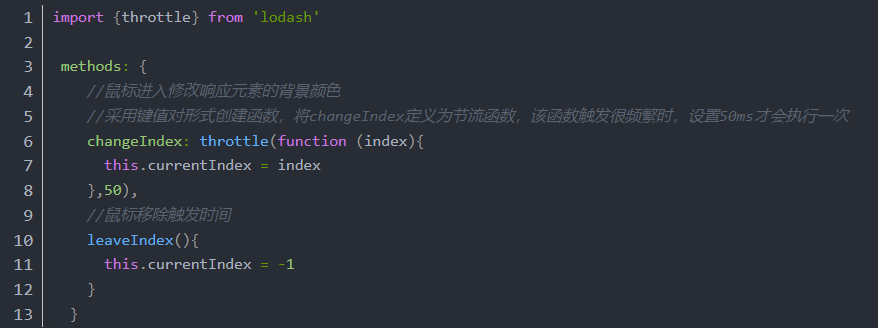
****注意****：****使用action时，函数的第一个参数，必须是{commit}****，即使不涉及到mutations操作，也必须加上该参数，否则会报错

## **19、loadsh插件防抖和节流**

**在进行窗口的resize、scroll，输入框内容校验等操作时，如果事件处理函数调用的频率无限制，会加重浏览器的负担，导致用户体验非常糟糕。此时我们可以采用debounce（防抖）和throttle（节流）的方式来减少调用频率，同时又不影响实际效果。**

**安装lodash插件，该插件提供了防抖和节流的函数，我们可以引入js文件，直接调用。当然也可以自己写防抖和节流的函数**

防抖：用户操作很频繁，但是只执行一次，减少业务负担。  
节流：用户操作很频繁，但是把频繁的操作变为少量的操作，使浏览器有充分时间解析代码



## **编程式导航+事件委托实现路由跳转**

· 导航式路由的问题：

· 如果页面中有很多 <a> 标签，每个都用 <router-link> 生成，可能会在渲染大量链接时出现性能问题，特别是在频繁的用户交互时，可能导致卡顿。

· 编程式路由的问题：

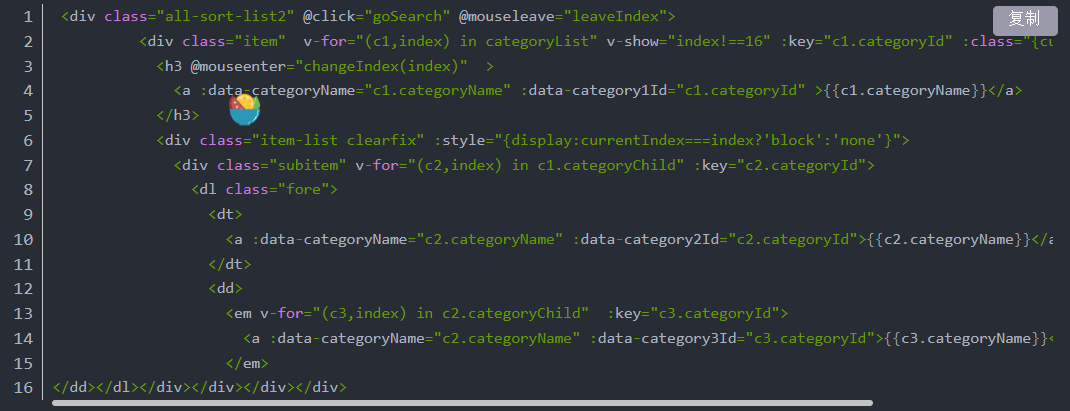
· 虽然编程式路由可以避免页面的卡顿，但每个 <a> 标签都需要单独绑定点击事件处理函数，这在大型页面中会导致大量重复代码，也会影响性能和可维护性。

解决方案：事件委托 + 编程式导航

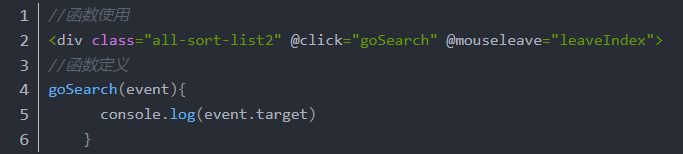
事件委托是一种常见的优化手段，即将子元素的事件处理委托给父元素，而不是为每个子元素单独添加事件处理函数。这样只需要一个事件处理函数，能够极大地减少代码量，并提高性能。

****事件委派问题：****  
（1）如何确定我们点击的一定是a标签呢？如何保证我们只能通过点击a标签才跳转呢？

为三个等级的a标签添加自定义属性date-categoryName绑定商品标签名称来标识a标签  
（2）如何获取子节点标签的商品名称和商品id(****我们是通过商品名称和商品id进行页面跳转的****)

为三个等级的a标签再添加自定义属性data-category1Id、data-category2Id、data-category3Id来获取三个等级a标签的商品id，用于路由跳转。  


****注意****：event是系统属性，所以我们只需要在函数定义的时候作为参数传入，在函数使用的时候不需要传入该参数。



## **Vue路由销毁问题**

Vue在路由切换的时候会销毁旧路由。

我们在三级列表全局组件TypeNav中的mounted进行了请求一次商品分类列表数据。

由于Vue在路由切换的时候会销毁旧路由，当我们再次使用三级列表全局组件时还会发一次请求。

如下图所示：当我们在包含三级列表全局组件的不同组件之间进行切换时，都会进行一次信息请求。

由于信息都是一样的，出于性能的考虑我们希望该数据只请求一次，所以我们把这次请求放在App.vue的mounted中

## **22、mock插件使用**

自定义的数据用于测试前端接口。

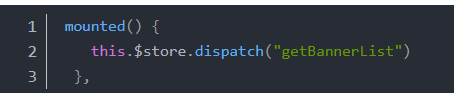
banner、floor分别为轮播图和页面底部的假数据。



## **23、vuex数据存储与使用**

把公共的数据放在store中，然后使用时再去store中取

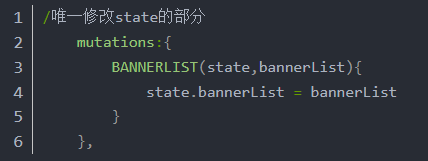
轮播图组件ListContainer.vue组件加载完毕后发起轮播图数据请求。



1. 请求实际是在store中的actions中完成的



获取到数据后存入store仓库，在mutations完成



轮播图组件ListContainer.vue组件在store中获取轮播图数据。由于在这个数据是通过异步请求获得的，所以我们要通过计算属性computed获取轮播图数据。



## **swiper插件实现轮播图**

（1）安装swiper  
（2）在需要使用轮播图的组件内导入swpier和它的css样式  
（3）在组件中创建swiper需要的dom标签（html代码，参考官网代码）  
（4）创建swiper实例

注意：在创建swiper对象时，我们会传递一个参数用于获取展示轮播图的DOM元素，官网直接通过class（而且这个class不能修改，是swiper的css文件自带的）获取。但是这样有缺点：当页面中有多个轮播图时，因为它们使用了相同的class修饰的DOM，就会出现所有的swiper使用同样的数据，这肯定不是我们希望看到的。

解决方法：在轮播图最外层DOM中添加ref属性  
<div class="swiper-container" id="mySwiper" ref="cur">  
通过ref属性值获取DOM  
let mySwiper = new Swiper(this.$refs.cur,{...})

什么候去加载这个swiper，我们第一时间想到的是在mounted中创建这个实例。

但是会出现无法加载轮播图片的问题,我们在mounted中先去异步请求了轮播图数据，然后又创建的swiper实例。由于请求数据是异步的，所以浏览器不会等待该请求执行完再去创建swiper，而是先创建了swiper实例，但是此时我们的轮播图数据还没有获得，就导致了轮播图展示失败。

等我们的数据请求完毕后再创建swiper实例。只需要加一个1000ms时间延迟再创建swiper实例.但不是最好的

解决方法二：我们可以使用watch监听bannerList轮播图列表属性，因为bannerList初始值为空，当它有数据时，我们就可以创建swiper对象，但是我们是通过v-for遍历bannerList中的图片数据，并不能保证此时v-for已经执行完了，假如watch先监听到bannerList数据变化，执行回调函数创建了swiper对象，之后v-for才执行，这样也是无法渲染轮播图图片（因为swiper对象生效的前提是html即dom结构已经渲染好了）。

完美解决方案：使用watch+[this.$nextTick()](https://cn.vuejs.org/v2/api/" \l "vm-nextTick)  
官方介绍：this. $nextTick它会将回调延迟到下次 DOM 更新循环之后执行（循环就是这里的v-for）。

就是等我们页面中的结构都有了再去执行回调函数

注意：之前我们在学习watch时，一般都是监听的定义在data中的属性，但是我们这里是监听的computed中的属性，这样也是完全可以的

## **25、props父子组件通信**

父组件设置一个属性绑定要传递的数据  
子组件props接受该属性值

Floor是子组件，我们在home组件中调用了Floor，我们把home组件认为父组件，我们在home组件中实现了由home组件向Floor组件传递信息的操作，即父组件向子组件传递信息。

## **26、将轮播图模块提取为公共组件**

我们要把定义swiper对象放在mounted中执行，并且还要设置immediate：true属性，这样可以实现，无论数据有没有变化，上来立即监听一次。

将轮播图列表传递给子组件，原理相同。



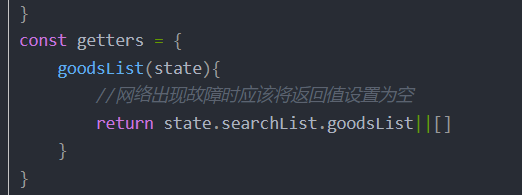
将轮播图组件已经提取为公共组件Carouse，所以我们只需要在Carouse中引入swiper和相应css样式。

## **27、getters使用**

如果不使用getters属性，我们在组件获取state中的数据表达式为：this.$store.state.子模块.属性

“getter”（可以认为是 store 的计算属性）。就像计算属性一样，getter 的返回值会根据它的依赖被缓存起来，且只有当它的依赖值发生了改变才会被重新计算。getters 是全局的，无论在哪个模块中定义，都会在全局作用域内暴露出来，可以通过 this.$store.getters 访问。store也是全局的，mutation和action也是因为默认全局命名空间是全局的

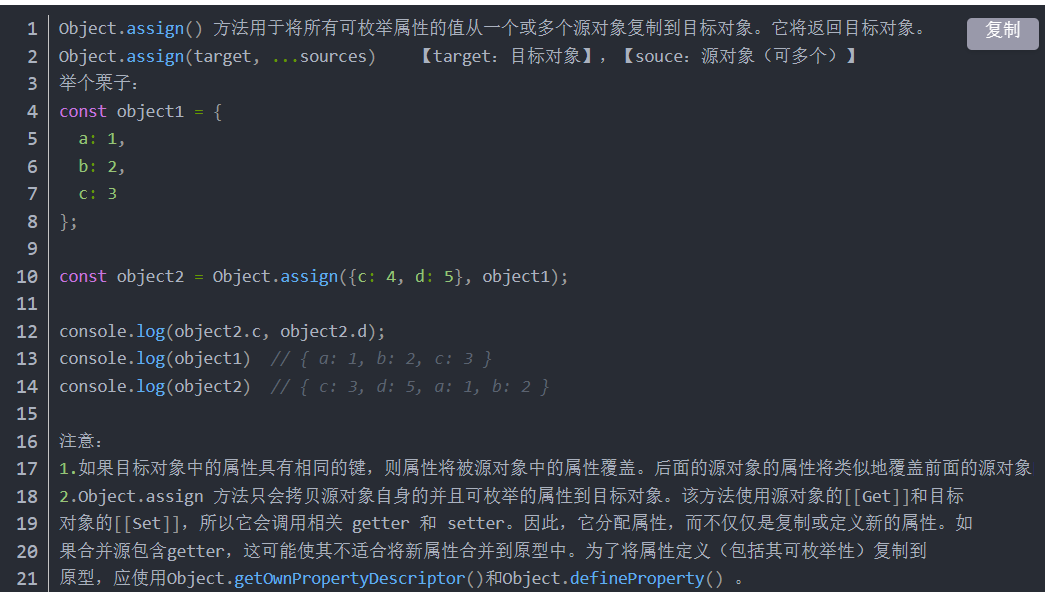
当网络出现故障时应该将返回值设置为空，如果不设置返回值就变成了undefined。

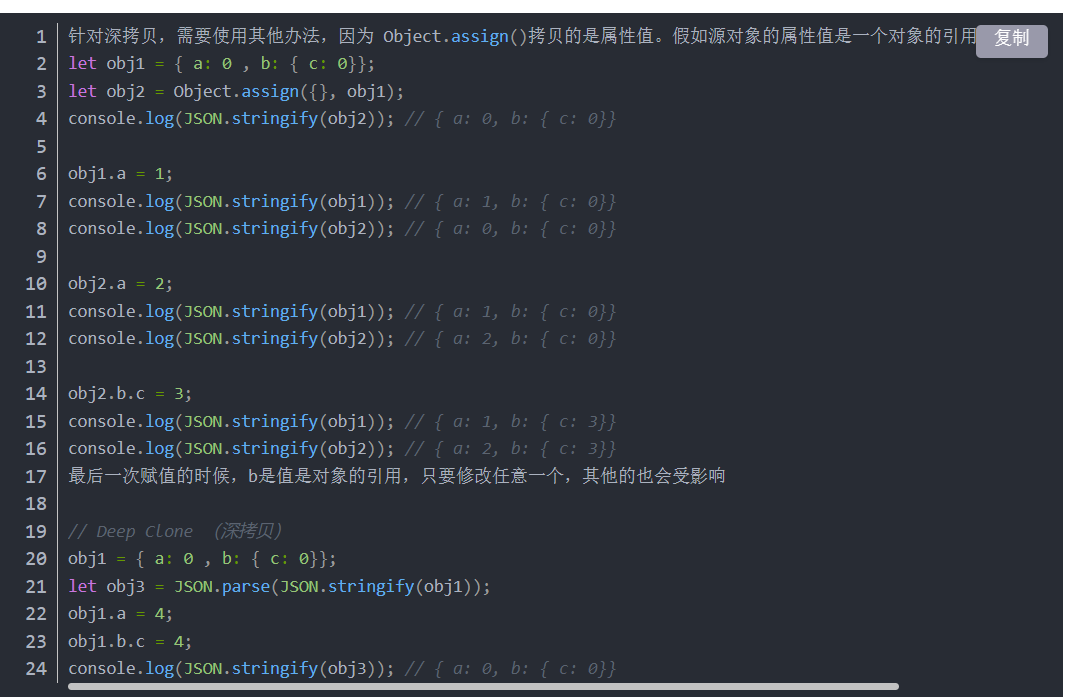


在Search组件中使用getters获取仓库数据



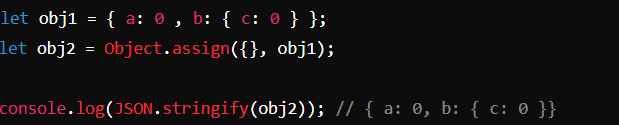
## **28、Object.asign实现对象拷贝**

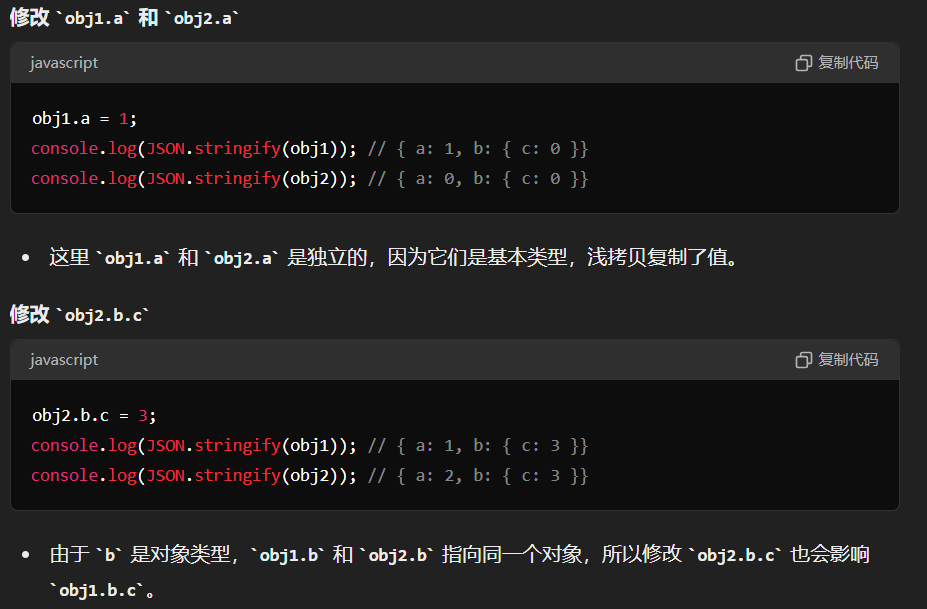




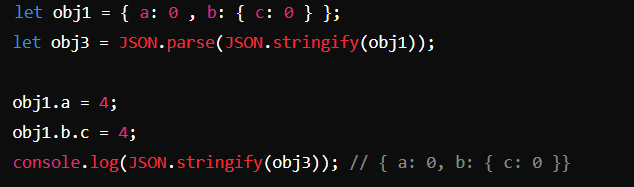
浅拷贝：当你复制一个对象时，只复制了对象的最外层属性。如果属性值是一个引用类型（比如另一个对象或数组），那么拷贝的只是这个引用，而不是引用的值本身。因此，拷贝后的对象和原对象共享对同一个内层对象的引用，修改其中一个会影响另一个。

深拷贝：深拷贝会递归地复制对象的每一层属性。这样，拷贝后的对象和原对象完全独立，互不影响。





深拷贝：**一种常见的深拷贝方法**是使用 JSON.parse(JSON.stringify())：



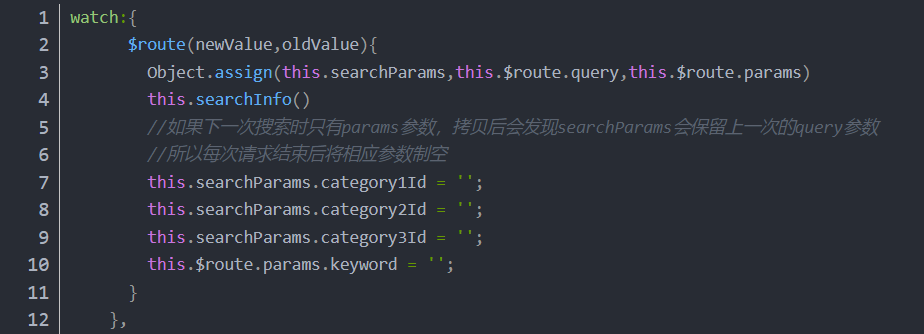
## **30、利用路由信息变化实现动态搜索**

最初想法：在每个三级列表和收缩按钮加一个点击触发事件，只要点击了就执行搜索函数。  
这是一个很蠢的想法，如果这样就会生成很多回调函数，很耗性能。

最佳方法：我们每次进行新的搜索时，我们的query和params参数中的部分内容肯定会改变，而且这两个参数是路由的属性。我们可以通过监听路由信息的变化来动态发起搜索请求

$route是组件的属性，所以watch是可以监听的（watch可以监听组件data中所有的属性）

data的属性包括：自己定义的、系统自带的（如 $route）、父组件向子组件传递的等等。

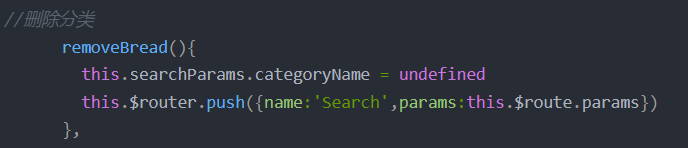


## **面包屑相关操作**

主要就是两个删除逻辑，分类属性（query）删除时删除面包屑同时修改路由信息。

搜索关键字（params）删除时删除面包屑、修改路由信息、同时删除输入框内的关键字。

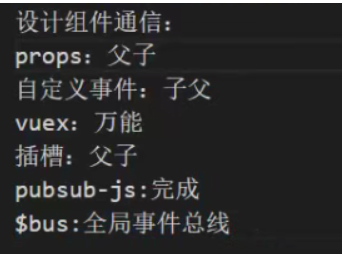
因为此部分在面包屑中是通过categoryName展示的，所所以删除时应将该属性值制空或undefined。  
可以通过路由再次跳转修改路由信息和url链接



1. params删除时

不同点是此部分会多一步操作：删除输入框内的关键字（因为params参数是从输入框内获取的）

输入框在Header组件中的

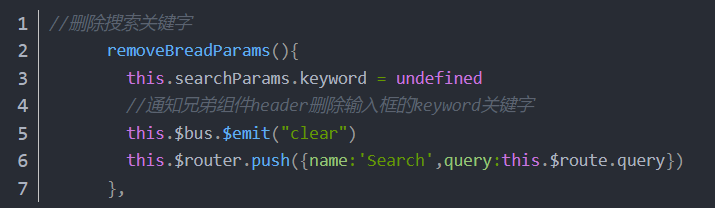


这里通过$bus实现header和search组件的通信

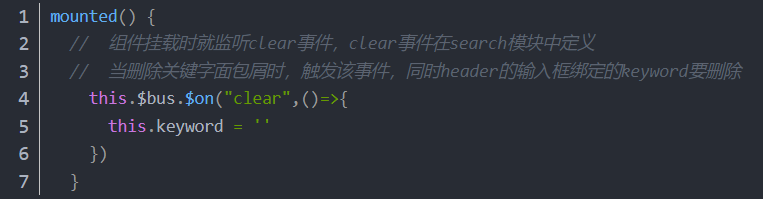
1. 在main.js中注册



1. search组件使用$bus通信，第一个参数可以理解为为通信的暗号，还可以有第二个参数（用于传递数据）



1. header组件接受$bus通信



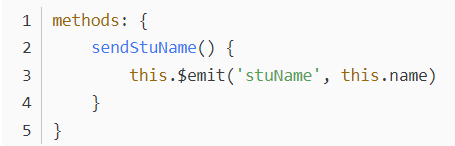
## **组件通信方式**

$ on、$emit自定义事件实现子组件给父组件传递信息。 props实现父组件给子组件传递数据。

自定义事件与全局事件总线：<https://blog.csdn.net/DogEgg_001/article/details/132004374>

父组件使用$on 子组件使用$emit

父：

子：

****第二种全局事件总线 $bus****（适用于所有的场景）

****第三种Vuex****  
****第四中插槽****（适用于父子组件通信）

## **SearchSelector子组件传参及面包屑操作**

之前描述了通过query、params参数生成面包屑，以及面包屑的删除操作对应地址栏url的修改。

SearchSelector组件有两个属性也会生成面包屑，分别为品牌名、手机属性。



涉及到子组件向父组件传递信息操作

设计知识点：

字符串拼接 ·${}·

、

总结：面包屑由四个属性影响：parads、query、品牌、手机属性

****生成逻辑：****判断searchParams相关属性是否存在，存在即显示。

## **商品排序**

只是改变一下请求参数中的order字段，后端会根据order值返回不同的数据来实现升降序。



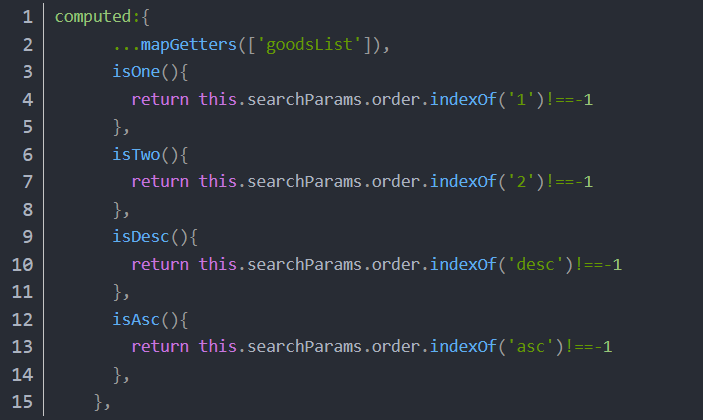
图标是iconfont网站的图标，通过引入在线css的方式引入图标

在public文件index引入该css  
<link rel="stylesheet" href="https://at.alicdn.com/t/font\_2994457\_qqwrvmss9l9.css">

在search模块使用该图标



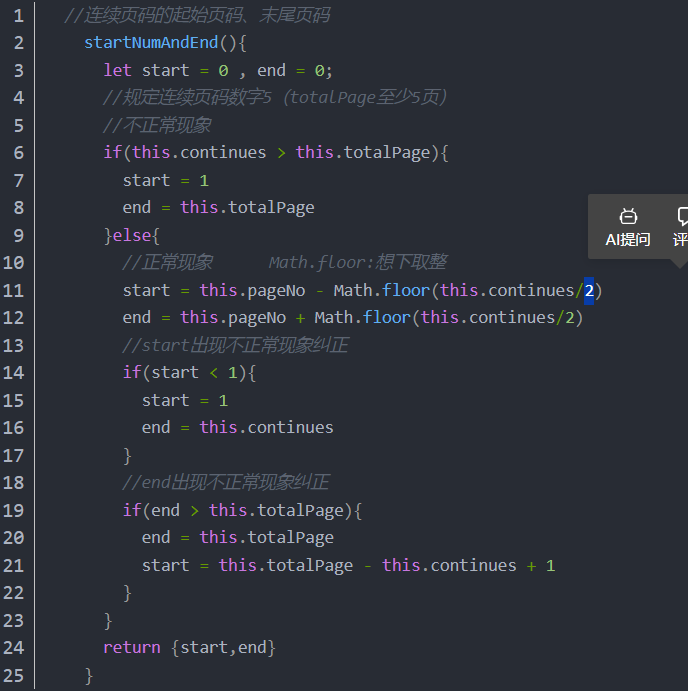
isOne、isTwo、isAsc、isDesc计算属性代码



## **手写分页器**

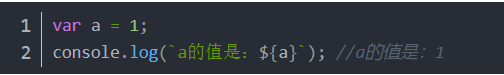
实际开发中是不会手写的，一般都会用一些开源库封装好的分页，比如element ui。

核心逻辑是获取连续页码的起始页码和末尾页码，通过计算属性获得。



## **36、字符串拼接**

如果想在你的字符串内加入某个变量的值，就需要字符串拼接使用 ``（飘符号），由于 飘在markdown是单行代码标记所以下面我们用··代替。  
字符串拼接 ·${}·，使用方法如下：



****在html中使用****

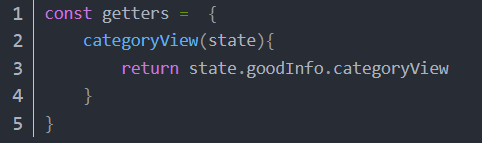


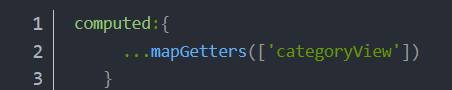
## **37、滚动条**

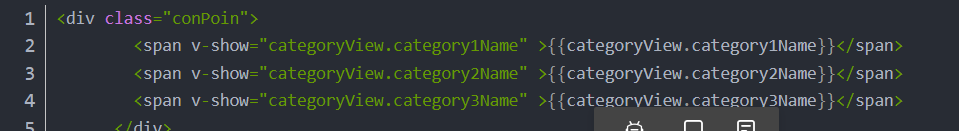
用前端路由，当切换到新路由时，想要页面滚到顶部，或者是保持原先的滚动位置，就像重新加载页面那样。 vue-router 能做到，而且更好，它让你可以自定义路由切换时页面如何滚动。

## **38、undefined细节(\*\*\*\*\*)**

访问undefined的属性值会引起红色警告，可以不处理，但是要明白警告的原因。







如果getters按上面代码写为return state.goodInfo.categoryView，页面可以正常运行，但是会出现红色警告。

原因：假设我们网络故障，导致goodInfo的数据没有请求到，即goodInfo是一个空的对象，当我们去调用getters中的return state.goodInfo.categoryView时，因为goodInfo为空，所以也不存在categoryView，即我们getters得到的categoryView为undefined。所以我们在html使用该变量时就会出现没有该属性的报错。

****总结****：所以我们在写getters的时候要养成一个习惯在返回值后面加一个||条件。即当属性值undefined时，会返回||后面的数据，这样就不会报错。  
如果返回值为对象加||{}，数组：||[ ]。

## **商品详情**

老师的方法很巧妙：在轮播图组件中设置一个currendIndex，用来记录所点击图片的下标，并用currendIndex实现点击图片高亮设置。当符合图片的下标满足currentIndex===index时，该图片就会被标记为选中。

<https://blog.csdn.net/weixin_40758850/article/details/89445629>

## **失焦事件**

· blur 事件 是基于焦点变化的，即使没有任何输入或输入没有导致值变化，它仍会触发。

· change 事件 仅在输入内容发生实际变化并且用户离开输入框时才会触发。

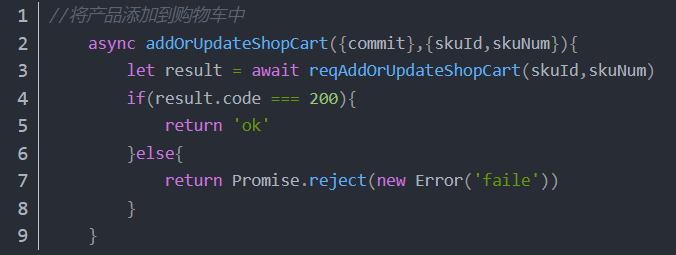
## **41、加入购物车成功路由**

点击加入购物车时，会向后端发送API请求，但是该请求的返回值中data为null，所以我们只需要根据状态码code判断是否跳转到‘加入购物车成功页面’。

Detail加入购物车请求：



detail store对应代码



当我们想要实现两个毫无关系的组件传递数据时，首相想到的就是路由的query传递参数，但是query适合传递单个数值的简单参数，所以如果想要传递对象之类的复杂信息，就可以通过Web Storage实现。

sessionStorage、localStorage概念：

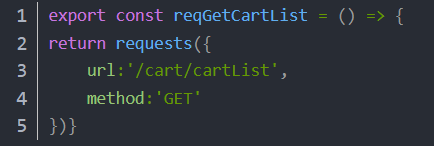
sessionStorage：为每一个给定的源维持一个独立的存储区域，该区域在页面会话期间可用（即只要浏览器处于打开状态，包括页面重新加载和恢复）。

localStorage：同样的功能，但是在浏览器关闭，然后重新打开后数据仍然存在。

注意：无论是session还是local存储的值都是字符串形式。如果我们想要存储对象，需要在存储前JSON.stringify()将对象转为字符串，在取数据后通过JSON.parse()将字符串转为对象。

## **42、购物车组件开发**

Api接口：

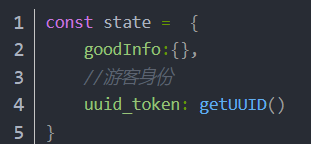


如果想要获取详细信息，还需要一个用户的uuidToken，用来验证用户身份。但是该请求函数没有参数，所以我们只能把uuidToken加在请求头中。

创建utils工具包文件夹，创建生成uuid的js文件，对外暴露为函数（记得导入uuid => npm install uuid）。  
****生成临时游客的uuid（随机字符串）,每个用户的uuid不能发生变化，还要持久存储****



用户的uuid\_token定义在store中的detail模块



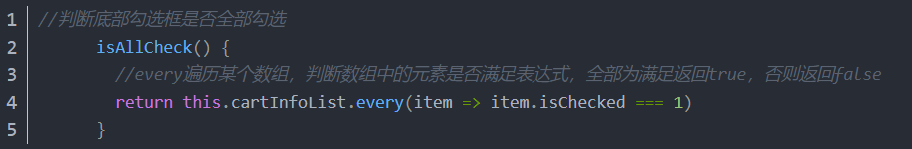
在request.js中设置请求头：



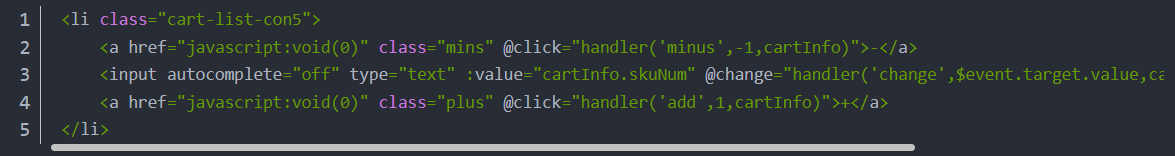
## **43、购物车商品数量修改及个人疑问**

every函数使用：every遍历某个数组，判断数组中的元素是否满足表达式，全部为满足返回true，否则返回false

判断底部勾选框是否全部勾选代码部分：



当使用函数的参数个数不同时，例如：handler(‘minus’,-1,cartInfo)和 handler(‘minus’,-1)，执行的还是同一个method的handler函数，只是对应的参数变为了undefined。即js不会像java一样，有代码的重载。



handler函数，修改商品数量时，加入****节流****操作。

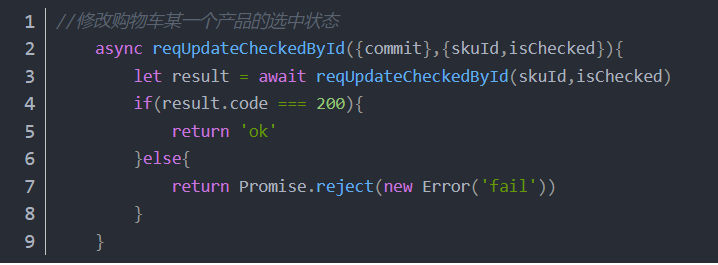
添加到购物车和对已有物品进行数量改动使用的同一个api，可以查看api文档。  
handler函数有三个参数，type区分操作，disNum用于表示数量变化（正负）,cart商品的信息



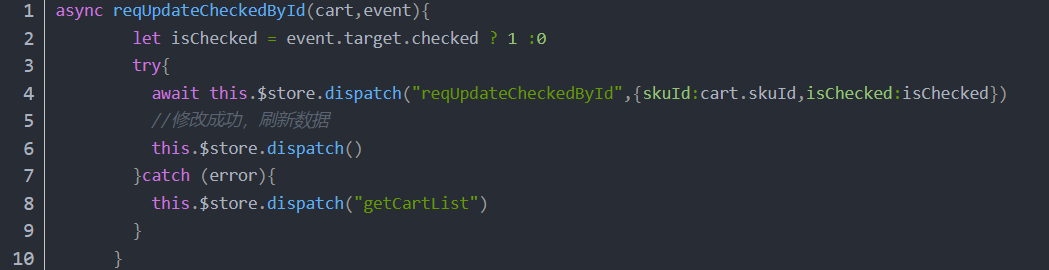
在最后一步更新商品信息时，我最初的想法时使用this.$router.push('/shopcart')实现。因为，我们shopcart页面的mounted会去获取商品信息。当路由跳转时会执行mounted的内容，实现商品数据信息更新。

## **44、购物车状态修改和商品删除**

action部分：以删除购物车某个商品数据为例



method部分：（****重点是try、catch****）

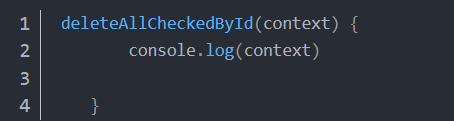


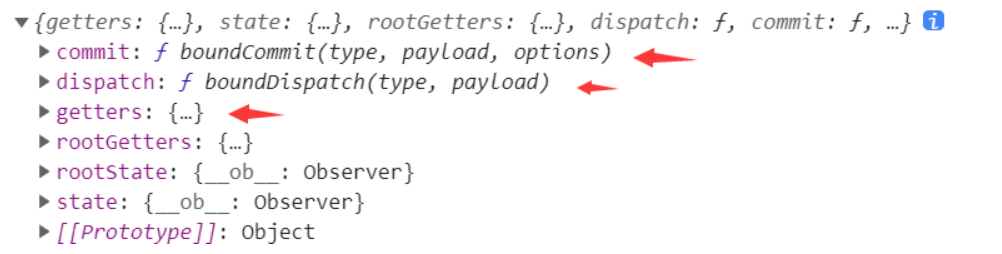
## **删除多个商品（action扩展**

由于后台只提供了删除单个商品的接口，所以要删除多个商品时，只能多次调用actions中的函数。

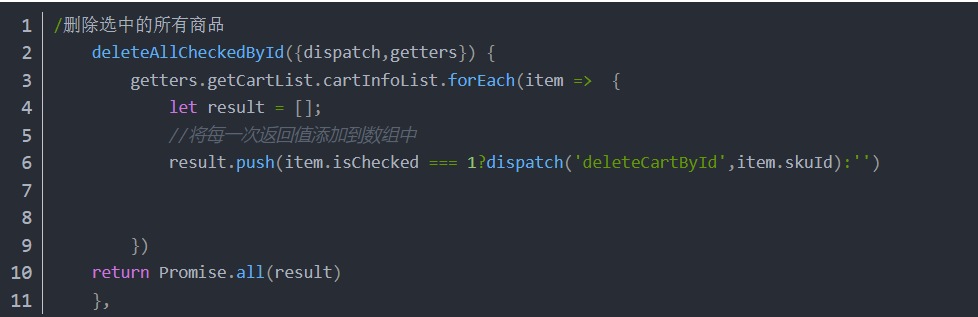
我们可能最简单的方法是在method的方法中多次执行dispatch删除函数，当然这种做法也可行，但是为了深入了解actions，我们还是要将批量删除封装为actions函数。

actions扩展





context中是包含dispatch、getters、state的，即我们可以在actions函数中通过dispatch调用其他的actions函数，可以通过getters获取仓库的数据。



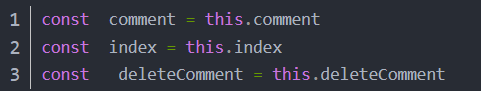
Promise.all可以将多个Promise实例包装成一个新的Promise实例。同时，成功和失败的返回值是不同的，成功的时候返回的是一个结果数组，而失败的时候则返回最先被reject失败状态的值。

## **46、注册登录业务(ES6 const新用法)**

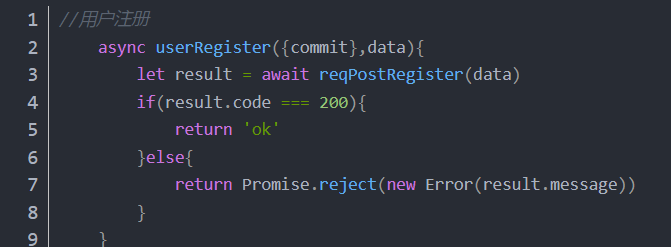
ES6 const新用法



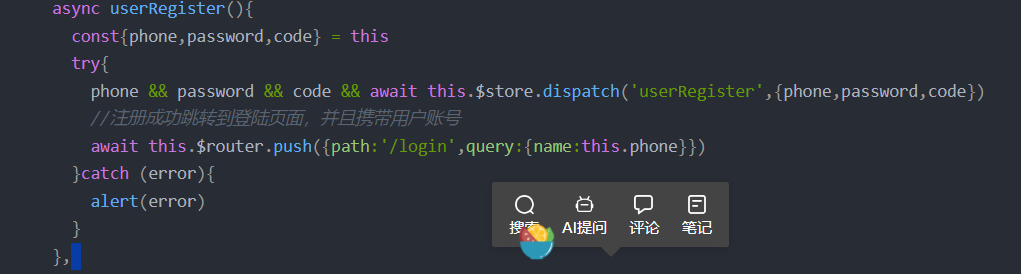
含义相当于:



用户注册模块：

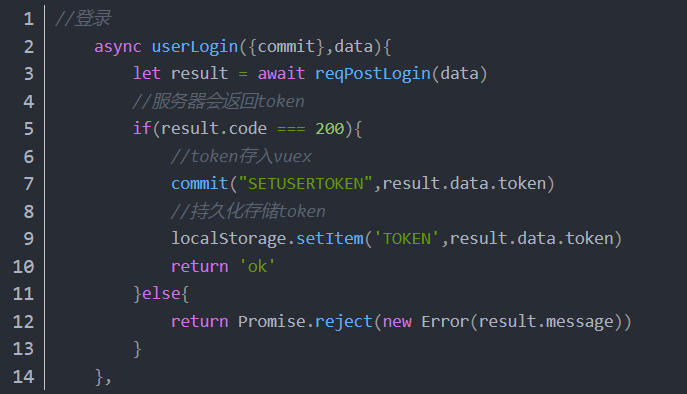


注册成功跳转到登陆页面，并且携带用户账号（个人完善）

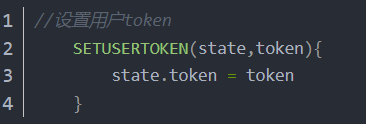




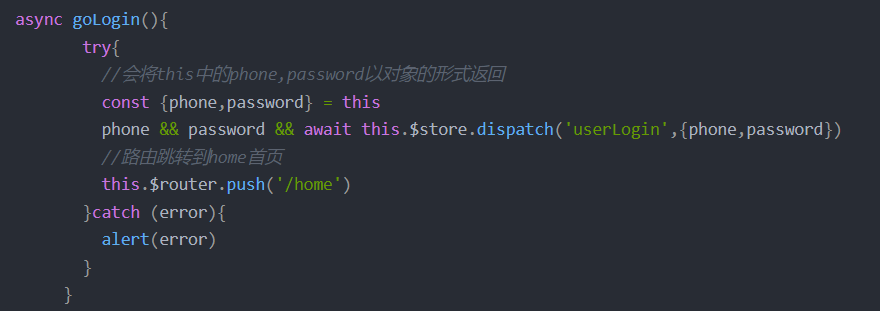
登录按钮的父节点是一个form表单，如果使用@click触发登录事件，form表单会执行默认事件action实现页面跳转。这里我们使用@click.prevent，它可以阻止自身默认事件的执行。



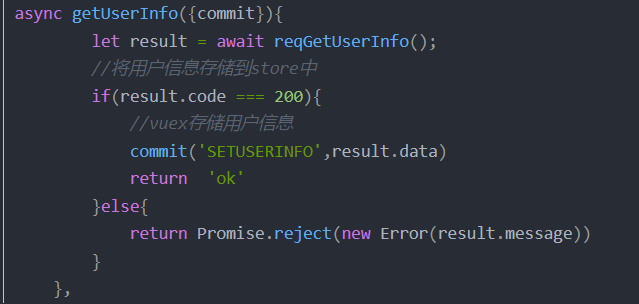
mutations设置用户token



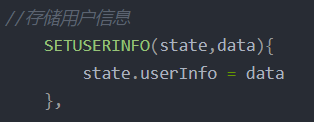
登陆组件methods登陆函数



登陆成功后获取用户信息



mutations存储用户信息



## **47.导航守卫**

**全局守卫：在应用的任何路由切换时都会触发的守卫。 适用于需要在所有路由切换时执行的逻辑，比如用户身份验证、日志记录等。**

**独享路由守卫 是针对特定路由定义的守卫。**

**组件内守卫 是定义在组件内部的守卫。**

## **交易模块**

1. 封装API  
   （2）vuex三件套  
   （3）dispatch发请求  
   （4）数据渲染

虽然二级路由在[后台管理系统](https://so.csdn.net/so/search?q=%E5%90%8E%E5%8F%B0%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_43424325/article/details/_blank)常用，但是我们的个人中心也可以用到二级路由。



****总结警告缘由****：当某个路由有子级路由时，父级路由须要一个默认的路由，因此父级路由不能定义name属性，解决办法是去掉name:'Center’就好了。

## **51、图片懒加载**

[懒加载vue-lazyload插件官网](https://www.npmjs.com/package/vue-lazyload)  
插件的使用直接参考官方教程，很简单。

vue使用插件的原理

每个插件都会有一个install方法，install后就可以在我们的代码中可以使用该插件。这个install有两类参数，第一个为Vue实例，后面的参数可以自定义。

vue使用插件的步骤

1、引入插件 import VueLazyload from "vue-lazyload";

2、注册插件Vue.use(VueLazyload)

这里的Vue.use()实际上就是调用了插件的install方法。如此之后，我们就可以使用该插件了。

## **52、表单验证**

表单验证个人推荐使用element ui的from表单验证，看一下官网的示例就会用。  
[element ui from表单验证链接](https://element.eleme.cn/" \l "/zh-CN/component/form)

## **53、路由懒加载**

当打包构建应用时，JavaScript 包会变得非常大，影响页面加载。如果我们能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了。  
[路由懒加载链接](https://router.vuejs.org/zh/guide/advanced/lazy-loading.html)



## **54、打包项目**

项目文件夹下执行npm run build，生成dist打包文件

****map文件作用****：因为代码是经过加密的，如果运行时报错，输出的错误信息无法准确得知时那里的代码报错。有了map就可以向未加密的代码一样，准确的输出是哪一行那一列有错。

## **ref使用**

我们可以通过ref获取组件的信息，并且可以写该组件的信息：

例如：父组件想要获取子组件的信息或者修改子组件的数据，就可以通过ref获取。

使用步骤：

（1）在被操作的标签定义ref=“name”

（2）在父组件可以通过this.$refs.name获取标签的全部信息，也可以进行数据的修改。

不使用parents：子组件通过之前常用的自定事件来通知父组件修改money，然后父组件money执行减操作。  
使用parents：子组件直接通过this.$parent属性获取父组件，然后在子组件内部对money执行减操作

****插槽的原理就是在子组件（HintButton）内定义一个slot（插槽），父组件可以向该插槽内插入数据。****