## 1 $nextTick 异步更新队列

<div id="app">  
 <div id="div" v-if="showDiv">文本</div>  
 <button @click="getText">点击获取文本内容</button>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 new Vue({  
 el: '#app',  
 data: {  
 showDiv: false  
 },  
 methods:{  
 getText: function () {  
 this.showDiv = true;  
 let text = document.getElementById('div').innerHTML;  
 console.log(text);  
 }  
 }  
 });  
</script>

示例代码无法获取到文本内容，且报错：cannot read property ‘innerHTML’。

vue观察到数据时，并不是直接更新DOM，而是开启一个队列，缓冲在同一个事件循环中发生的所有数据变化，缓冲时会去除重复数据。在下一个事件循环tick中，刷新队列并执行更新。所以如果用一个for循环来动态改变数据100次，其实只会应用最后一次。

vue会依据当前浏览器环境优先使用源生的Promise.then和MutationObserver，如果都不支持，使用setTimeout代替。

上述案例中，执行this.showDiv=true时候，div还没被创建，直到下一个vue事件循环，才开始创建。$nextTick可以知道什么时候DOM更新完成。

getText: function () {  
 this.showDiv = true;  
 this.$nextTick(function () {  
 let text = document.getElementById('div').innerHTML;  
 console.log(text);  
 });  
}

## 2 X-Templates定义模板

如果不使用webpack、gulp，那么template模板的书写是一场噩梦。vue支持在script表使用text/x-template类型，并指定id，将id赋值给template。此时可以愉快的书写html，不用考虑换行问题。

<script type=”text/x-template” id=”my-comp”>

//书写html模板

</script>

//在vue中的使用

template: ‘#my-comp’

## 3 手动挂载实例

一般我们通过new Vue()创建vue实例，如果需要动态创建，需要使用Vue.extend和$mount两个方法手动挂载实例。

vue在实例化时，没有el，那么会处于未挂载状态，没有关联的dom元素，使用$mount手动挂载，这个方法返回实例本身，因而可以链式调用。

<div id="app">  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let MyComponent = Vue.extend({  
 template: '<div>hi</div>'  
 });  
 new MyComponent().$mount('#app');  
</script>

也可以写为：

new MyComponent({

el:’#app’

});

## 4 自定义指令

可以在选项对象的directives属性中添加：

|  |
| --- |
| **directives**: {  **'focus'**: {  update(el,binding){  **if**(binding.**value**){  el.focus();  }  }  } },  在标签中直接使用v-focus即可 |

## 5 watch

在上述案例中，我们从loacalStorage中存储数据，在界面中，依据业务需要，对该数据进行增删改，分别对应了vue的method中的增删改方法，但是如果一旦数据需要变动，增删改三个方法都要进行修改，这是很麻烦的。这时候我们可以使用watch来统一监控属性改变。



var list = store.fetch(‘**new-calss**’);

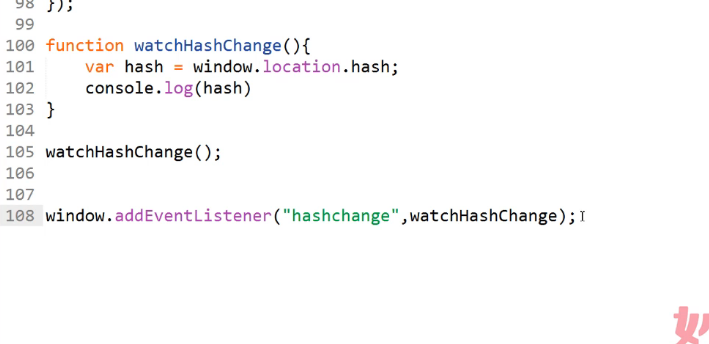
|  |
| --- |
| **watch**: {  list: **function** () {  store.save(**'new-calss'**,**this**.list);  }  }, |

此时只要list发生变化，就会执行上述function。无需再在 编辑，插入，删除等方法中进行数据存储。

但是这样无法对list进行深度监控，即当list对象内部属性改变的时候，仍然无法监控到，下列方法可以实现深度监控：

|  |
| --- |
| **watch**: {  **list**: {  handler: **function** () {  *//存储数据* },  **deep**: **true** } }, |

## 6 获取hash



## 8 模板render函数

### 8.1 render函数初体验

以下代码：

<div id="app">  
 <mycomp :level="2" title="特性">特性</mycomp>  
 <script type="text/x-template" id="my-comp">  
 <div>  
 <h1 v-if="level === 2"><a :href="'#' + title"><slot></slot></a></h1>  
 <h1 v-if="level === 3"><a :href="'#' + title"><slot></slot></a></h1>  
 <h1 v-if="level === 4"><a :href="'#' + title"><slot></slot></a></h1>  
 </div>  
 </script>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 Vue.component('mycomp',{  
 template: '#mycomp',  
 props: {  
 level: {  
 type: Number,  
 required: true  
 },  
 title: {  
 type: String,  
 default: ''  
 }  
 }  
 });  
 new Vue({  
 el: '#app'  
 });  
</script>

使用render的写法：

<div id="app">  
 <mycomp :level="2" title="特性">特性</mycomp>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 Vue.component('mycomp',{  
 template: '#mycomp',  
 props: {  
 level: {  
 type: Number,  
 required: true  
 },  
 title: {  
 type: String,  
 default: ''  
 }  
 },  
 render: function (createElement) {  
 return createElement(  
 'h' + this.level,  
 [  
 createElement(  
 'a',  
 {  
 domProps: {  
 href: '#' + this.title  
 }  
 },  
 this.$slots.default  
 )  
 ]  
 );  
 }  
 });  
 new Vue({  
 el: '#app'  
 });  
</script>

render函数通过createElement参数创建虚拟dom。（访问slot的使用场景在render函数里）

常见属性有：

class:{} 绑定class，和v-bind:class 一样

style:{} 绑定样式，和v-bind:style一样

attrs:{} 添加行间属性

domProps:{} DOM元素属性

on:{} 绑定事件

nativeOn:{} 监听原生事件

directives:{} 自定义指令

注意：domPropps属性中可以直接书写dom对象本身的方法，如：

domPropps:{

innerHTML: ‘<h1>test</h1>’

}

这里要谨慎使用，因为他会替换掉createElement中的元素。

## 9 vue与axios

一、axios介绍和使用mock数据

vue本身不提供发送请求，拿数据的功能的，官方推荐我们使用axios这个模块。axios不仅仅可以使用在vue中，它可以使用在任何地方，只要你想发送ajax请求，都可以使用这个模块；

所以我们有必要学习一下这个模块，以及怎么去使用它提供的一些方法

※ 简介：

基于promise用于浏览器和nodejs的与服务器端通信库；

这句话可以看出，它是完全基于promise，也就是完全支持promise API；可以用于浏览器端去请求，也就用script标签引入，然后发送请求；也可以使用在node端，发送请求，拿到数据。

※ 特征：

支持promise API

可以拦截请求和响应

可以转换请求和响应数据

可以取消请求

可以自动转换json数据

※ 使用：

CDN地址：http://unpkg.com/axios@0.16.2/dist/axios.min.js

CDN的形式就是请求一个远程的js文件（也可以将js文件下载到本地）

node installl axios --save

node使用的时候，安装一下就行了

1、使用axios

用axios就是发送请求拿到数据的，所以可以使用任何的环境。这里我们用vue-cli搭建的环境，然后在浏览器里面去跑，它帮我们拿数据，我们在控制台里面去看拿到了哪些数据。

a、在需要的模块中引入使用：

import axios from 'axios'

b、语法：

a、axios(config) 第一种写法：直接给个配置

b、axios[method]() 第二种写法：

两种方式都返回值为promise，也就意味着它可以使用then来去捕获到成功，用catch捕获到错误。

引入axios：import axios from 'axios'

在生命周期钩子函数 created(){} 里面去拿需要的数据：

cteated(){

axios({

method: 'get',

url:'https://easy-mock.com/mock/5a3a14090bd9de68557f35d6/example/listTest'

})

.then((response)=>{

console.log(response)

response.data // 拿到真正的数据

})

.catch((error)=>{

console.log(error)

})

}

这里是axios的第一种写法，直接给个配置，传入一个对象。

method 就是请求方式（get\post\delete....），如果不写method，那么默认就是get；

url 就是请求发地址

整个axios()会给我们返回一个promise，所以在后面我们直接.then()

请求成功返回了个response结果，但是在axios里面不是返回的结果，而是把结果放在data里面，data里面才是真正要的数据；后端返回的是json格式，它自动回解析成对象的格式，所以就不用 JSON.parse() 直接去解析了

response 返回的是axios加工过的数据（包含的有config\data\headers\request\status\...）

除了get请求，还可以用post请求：

cteated(){

axios({

method: 'post',

url:'https://easy-mock.com/mock/5a3a14090bd9de68557f35d6/example/postRequest'

})

}

c、还有一种方式，更加清晰明了展现请求方式的：

axios.get(url[,config])

axios.post(url[,data [,config]])

axios.delete(url[,config])

axios.head(url[,config])

axios.options(url[,data [,config]])

axios.put(url[,data [,config]])

axios.patch(url[,data [,config]])

除了前面两种，后面五种都需要后端的配置支持

2、当get或post需要发送一些数据的时候，我们该放在什么位置？

a、get发送数据：

get()的第一个参数是地址url，第二个参数就是我们要发送的数据；第二个参数是一个对象，就是对 的配置；在这个配置里面，提供了一个params对象；如果要发送一个 abc 为miaov 的参数，则书写为:

axios.get('',{

params: {

abc: 'miaov'

}

})

这个时候就会把数据以问号的形式，呈现在地址后面，查询字符串

https://easy-mock.com/mock/5a3a14090bd9de68557f35d6/example/listTest?abc=miaov

当发送的数据有多个的时候，可以直接在params里面添加；

axios.get('',{

params: {

abc: 'miaov',

miaov: 'ketang'

}

})

然后以key:value 形式添加在地址后面，然后以 & 的方式连接起来

https://easy-mock.com/mock/5a3a14090bd9de68557f35d6/example/listTest?abc=miaov&miaov=ketang

b、post发送数据：

post()发送数据的时候，第一个参数也是url，第二个参数也是一个对象，也是一个配置，但是直接可以在配置里面写参数：

axios.post('',{

abc: 'miaov',

miaov: 'ketang'

})

二、转换请求数据配置

1、自定义请求实例：

创造一个自定义的请求实例：一个页面中，我们可能请求A地址，也可能请求B地址；A地址里面我们可能请求了N多个内容接口，那么这时候，我们就不需要每次请求一个接口都去配置一遍，我们可以去配置一个统一的，然后只需要请求就可以了。

这就是可以允许我们自定义一些请求实例，这一类配置一遍，哪个地方用的时候，只需要用它这个实例去请求就行。

意思就是，请求的地址，前面都一样，后面的地址不一样，以及请求形式也可能不一样；其他的（比如：超时，转换数据 都是一样的）；所以我们可以相同的地址提取出来，配置一遍就可以了，以后用到这个地址里面的接口，只要用这个实例就可以了

a、创建实例：

首先我们需要用axios.create(config)去创建实例，之后请求的时候，就通过实例下面的方法就可以了，

import axios from 'axios'

创建实例，接收一个对象，这个对象就是我们传入的配置项；

★get请求：

var HTTP = axios.create({

baseURL: 'https://easy-mock.com/mock/5a3a14090bd9de68557f35d6/example/',

timeout: 1000,

headers:{

'custome-heade': 'miaov'

},

responseType: 'json',

params: {

// 查询一个 book，ID为 123

book: '123'

}

})

★post请求：

var HTTP = axios.create({

baseURL: 'https://easy-mock.com/mock/5a3a14090bd9de68557f35d6/example/',

timeout: 1000,

headers:{

'custome-heade': 'miaov'

},

responseType: 'json',

transformRequest:[function(data){

console.log(data)

return queryString.stringify(data)

}]，

transformResponse: [function(data){

data.abc = 'miaov'

return data

}],

validateStatus: function (){

}

})

配置项详解：

▶ 基本路径baseURL就是 路径中相同的部分,baseURL中的url是大写

▶ timeout就是连接超时。就是在发送请求的时候，有个时间段返回回来，当超过多少秒的时候，我们就中断发送；所以这里可以设置一个超时时间；这里的值，1000代表1秒，1代表1毫秒

▶ headers 可以通过它去设置请求头（自定义请求头）；设置的这个请求头，后端是可以拿到的，我们可以和后端进行约定，最终可以拿到数据。

也可以设置系统里面提供的请求头

▶ responseType 配置返回的数据类型，这里的类型是json；

请求完了之后，获取的json数据自动写成对象形式

▶ params 就是地址栏后面要发送的查询字符串相关的数据参数；需要查询什么就发送什么；它会自动放在地址栏的后面出现。

只要设置成params，无论是post还是get都会在地址栏后面加上这个查询字符串，所以post的时候，需要发送数据就写在地址url后面。

▶ transformRequest 转换请求数据；只适合于 put、post 和 patch。

它是个数组，数组里面写上一个函数，函数会接收一个参数，这个参数就是传给后端的数据。

函数没有返回值，相当于一个中间件一样，发送数据，经过塔去转换，转换的时候没有return出来任何数据，所以默认为 undefined ,所以函数里面需要把数据return出去。

当return出去以后，在调试工具的NetWork里面，我们发现发送的数据显示为：[object,object] 的形式，这种形式是不对的，因为会把这个对象转成字符串‘object’，那么这时候的发送数据就错了，所以就需要把它的数据进行转换。

如果发送的是对象的形象，就默认为json格式的，现在我们不需要json格式，我们需要发送请求的时候是：miaov=ketang&userName=leo 这种格式。所以需要用transformRequest 转换。

怎么转换呢？函数里的data不能直接return出去；我们需要把发送的对象参数的 miaov 和 userName 给提出来，不要大括号的形式，我们需要格式化一下；怎么格式化？我们需要引入它内部的一个模块： queryString ；

这个模块是系统提供的，不需要安装，但是需要引入，

import querystring from 'querystring'

这个模块是用来处理查询字符串的格式的。queryString 下面有个方法为 stringify(),用它来格式化成一个字符串，这时候调试工具的NetWork里面显示的结果为：

miaov: ketang

userName: leo

此时就不是一个对象，而是一个字符串。

但是也不是我们需要的格式，这时我们需要在请求头里面设置一个：

'content-type': 'application/x-www-form-urlencoded'

这样会自动格式化数据。这就是让post支持这种格式的

其实post还支持许多格式，比如上传图片的时候 content-type 的值 就对应的设置成 图片格式的值

这一块其实是请求头、transformRequest 共同的一个作用，这里是设置成以 & 连接起来的数据，你也可以设置成其他格式的数据

这里还有一个重要的功能，就是：你发送之前，可以对数据进行一些处理；比如在发送之前加上一些数据，加上年龄查询的属性：

transformRequest:[function(data){

data.age = 30

return queryString.stringify(data)

}]

▶ transformResponse 对请求回来的数据做进一步的处理。 它也是一个数组，里面是个函数，函数接收到一个参数，这个参数就是从后端发送过来的数据。这个函数也需要把数据 return 出去，不然就是 undefinde。

拿到数据后就可以做进一步的处理了，比如：拿到数据后都有一个属性abc的值为miaov；data.abc = 'miaov'

只要通过通过这个配置，然后THHP去请求的，无论是 get 还是 post 获取的数据， transformResponse都可以将数据转换，转换的时候添加或其他处理数据的操作都可以在此进行。这样很方便，不用每次单独去加了

▶ validateStatus 可以影响到你走哪个状态（then 或 catch）。它是个函数，它可以接收到一个参数，这个参数是状态码 status

这个的作用是：根据函数的返回值来判定你走哪个状态。

如果 return 一个true ，那么一律都是成功的，包括错误的时候也是走的正确的信息，而不是走的错误的信息catch（），不过这个时候的如果是错误信息，拿到的数据是没有的

如果 return 的是一个范围： return status < 500 , 那么小于500的状态码都会走then，而大于等于500的状态码都走catch。

这个根据自己的需求，哪个范围内是失败的，哪个范围内不让它失败

b、上面这块可以提取到一个文件里面，哪个模块用就可以import把整个文件引入进来就行，然后在模块里的created()钩子函数里去发送请求：

★get请求：

created(){

// get的时候，我们不要写完整的路径，只需要把相对路径 listTest 写进去就行

HTTP.get('listTest')

.then((response) => {

console.log(response.data)

})

.catch((error) => {

console.log(error)

})

}

★post请求：

created(){

// post的时候，我们也不要写完整的路径，只需要把相对路径 postRequest 写进去就行，

如果需要发送查询字符串，就写在地址后面：

此时虽然请求回来了，但是发送的数据我们看不到

HTTP.post('postRequest',{

miaov: 'ketang',

userName: 'leo'

})

.then((response) => {

console.log(response.data)

})

.catch((error) => {

console.log(error)

})

}

三、取消请求：

有时候我们可能会在发送之前取消请求，有时候会请求时间太长了而取消请求

1、需要取消请求就先要创建一个取消请求的令牌：

var CancelToken = axios.CancelToken;

var source = CancelToken.source()

2、创建完之后，我们要在配置项里面去配置一下：CancelToken: source.token

var HTTP = axios.create({

baseURL: 'https://easy-mock.com/mock/5a3a14090bd9de68557f35d6/example/',

CancelToken: source.token

})

3、取消请求：source.cancel()

created(){

HTTP.post('postRequest',{

miaov: 'ketang',

userName: 'leo'

})

.then((response) => {

console.log(response.data)

})

.catch((error) => {

console.log(error)

})

source.cancel('请求被取消了')

}

其实这里走的是error，因为请求没有成功，所以走 error

有时候为了区分，到底是服务端的错误还是用户取消，我们进行了判断：

.catch((error) => {

if(axios.isCancel(error)){

console.log(error.message)

}else{

console.log(error)

}

}

如果是用户取消就把message打出来，否则打出来请求失败的原因

四、并发请求：

如果每次有两个请求或更多请求同时发送，发送之后同时回来才能算成功，这个时候需要用到 axios 里面的 all

1、 现在发送两个请求，把两个请求封装在各自的函数中去：http1,http2；需要把两个都请求成功之后，再返回数据；

这个时候我们调用就要调用axios.all()，all()里面是一个数组，数组里面是每个函数的调用：[http1(),http2()]

cteated (){

function http1(){

return HTTP.get('listTest')

}

function http2(){

return HTTP.post('postRequest')

}

axios.all([http1(),http2()])

.then((response)=>{

console.log(response

})

.catch((error) => {

console.log(error)

})

}

2、如果发送多个请求，按照all（）方法，我们需要获取下标去请求，这里的then就不需要用到回调函数了，而是有另一个函数axios.spread(callback) 可以帮我们进行分割：

spread(callback)里面也是回调函数，回调函数的第一个参数就是请求之后的返回结果res1，依次第二个参数是第二个的返回结果res2：

axios.all([http1(),http2()])

.then((axios.spread(res1,res2 => {

console.log(res1)

console.log(res2)

}))

.catch((error) => {

console.log(error)

})

五、拦截器：

在发送之前，我们可以对请求进行拦截，还可以对响应进行拦截；

1、全局拦截器：

a、拦截请求：

axios下有个属性interceptors，这个属性下面的request 镖师发送请求的时候要拦截；use() 类似于中间件，在即将要发送和发送的时候做些事情

use接收的参数 config就是我们的配置文件。如果要拦截，不让它发送成功，就不要return config

HTTP.interceptors.request.use(function (config){

consolelog(config)

//在此根据需求对config进一步操作

return config

},function (error){

return Promise.reject(error)

})

b、拦截响应：

响应拦截的语法和结构和请求拦截是一样的，不过接收的不是config配置项了，而是data；

HTTP.interceptors.response.use(function (data){

consolelog(data)

//在此根据需求对data进一步操作

return data //如果不 return的话，我们拿到就是空的

},function (error){

return Promise.reject(error)

})

2、取消拦截：

axios.interceptors.request.eject(myInterceptor)

六、在Vue中使用：

我们没有必要每个模块用的时候，每次都要引进来，我们可以让它做为vue的插件,在全局里，放在原型插件去使用：

1、安装 : 我们需要安装两个模块：axios和vue-axios

npm install axios vue-axios --save

2、作为插件: VueAxios 在前面， Axios在后面

Vue.use(VueAxios.Axios)

3、在组建中使用:

在组件里面件使用的时候不要去引，直接this每个组件里面的 $http 这个属性，再找到对应的方法就行；

this.$http[method]()

操作步骤：

a 、在main.js里面引入安装好的两个模块：

main.js：

import Axios from 'axios'

import VueAxios from 'vue-axios'

作为插件使用：

Vue.use(VueAxios，Axios)

然后就可以在每个模块（组件）里面去使用了

b、在组件里面使用：

在created（）函数里面，实例this下有个属性 $http ；如果是get方式请求的，就接上get（）方法：

created(){

this.$http.get('地址')

.then((response) => {

console.(response)

})

.catch((error) =>{

console.log(error)

})

}