vue 中 Axios 的封装和 API 接口的管理

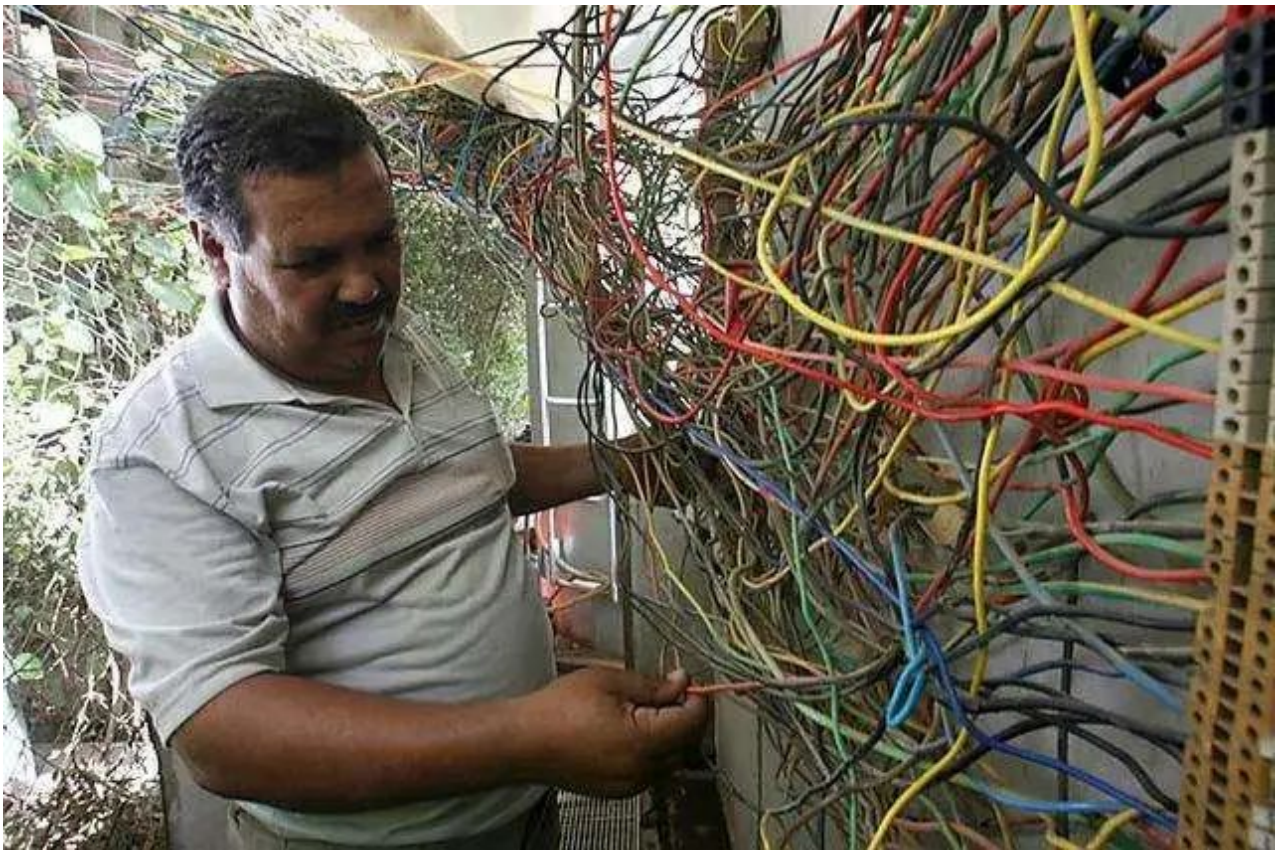
mp.weixin.qq.com/s/SH3LBCNFnPuU9x_Q3g0I0Q

↓推荐关注↓

大前端技术之路
分享Web前端，Node.js、React Native等大前端技术栈精选

18篇原创内容

公众号



如图，面对一团糟代码的你~~~真的想说，What F~U~C~K！！！

回归正题，我们所要的说的axios的封装和api接口的统一管理，其实主要目的就是在帮助我们简化代码和利于后期的更新维护。

一、axios的封装

在vue项目中，和后台交互获取数据这块，我们通常使用的是axios库，它是基于promise的http库，可运行在浏览器端和node.js中。他有很多优秀的特性，例如拦截请求和响应、取消请求、转换json、客户端防御XSRF等。所以我们的尤大大也是果断放弃了对其官方库vue-resource的维护，直接推荐我们使用axios库。如果还对axios不了解的，可以移步axios文档。

安装

```
npm install axios; // 安装axios
```

引入

一般我会在项目的src目录中，新建一个request文件夹，然后在里面新建一个http.js和一个api.js文件。http.js文件用来封装我们的axios，api.js用来统一管理我们的接口。

```
// 在http.js中引入axios
import axios from 'axios'; // 引入axios
import QS from 'qs'; // 引入qs模块，用来序列化post类型的数据，后面会提到
// vant的toast提示框组件，大家可根据自己的ui组件更改。
import { Toast } from 'vant';
```

环境的切换

我们的项目环境可能有开发环境、测试环境和生产环境。我们通过node的环境变量来匹配我们的默认的接口url前缀。axios.defaults.baseURL可以设置axios的默认请求地址就不多说了。

```
// 环境的切换
if (process.env.NODE_ENV == 'development') {
    axios.defaults.baseURL = 'https://www.baidu.com';}
else if (process.env.NODE_ENV == 'debug') {
    axios.defaults.baseURL = 'https://www.ceshi.com';
}
else if (process.env.NODE_ENV == 'production') {
    axios.defaults.baseURL = 'https://www.production.com';
}
```

设置请求超时

通过axios.defaults.timeout设置默认的请求超时时间。例如超过了10s，就会告知用户当前请求超时，请刷新等。

```
axios.defaults.timeout = 10000;
```

post请求头的设置

post请求的时候，我们需要加上一个请求头，所以可以在这里进行一个默认的设置，即设置post的请求头为 `application/x-www-form-urlencoded;charset=UTF-8`

```
axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-
urlencoded;charset=UTF-8';
```

请求拦截

我们在发送请求前可以进行一个请求的拦截，为什么要拦截呢，我们拦截请求是用来做什么的呢？比如，有些请求是需要用户登录之后才能访问的，或者post请求的时候，我们需要序列化我们提交的数据。这时候，我们可以在请求被发送之前进行一个拦截，从而进行我们想要的操作。

请求拦截

```
// 先导入vuex,因为我们要使用到里面的状态对象
// vuex的路径根据自己的路径去写
import store from'@/store/index';

 // 请求拦截器axios.interceptors.request.use(
    config => {
        // 每次发送请求之前判断vuex中是否存在token
        // 如果存在，则统一在http请求的header都加上token，这样后台根据token判断你的登录情况
        // 即使本地存在token，也有可能token是过期的，所以在响应拦截器中要对返回状态进行判断
        const token = store.state.token;
        token && (config.headers.Authorization = token);
        return config;
    },
    error => {
        return Promise.error(error);
})
```

这里说一下token，一般是在登录完成之后，将用户的token通过localStorage或者cookie存在本地，然后用户每次在进入页面的时候（即在main.js中），会首先从本地存储中读取token，如果token存在说明用户已经登陆过，则更新vuex中的token状态。然后，在每次请求接口的时候，都会在请求的header中携带token，后台人员就可以根据你携带的token来判断你的登录是否过期，如果没有携带，则说明没有登录过。这时候或许有些小伙伴会有疑问了，就是每个请求都携带token，那么要是一个页面不需要用户登录就可以访问的怎么办呢？其实，你前端的请求可以携带token，但是后台可以选择不接收啊！

响应的拦截

```
// 响应拦截器
axios.interceptors.response.use(
response => {
// 如果返回的状态码为200，说明接口请求成功，可以正常拿到数据
// 否则的话抛出错误
if (response.status === 200) {
returnPromise.resolve(response);
        } else {
returnPromise.reject(response);
        }
    },
// 服务器状态码不是2开头的的情况
// 这里可以跟你们的后台开发人员协商好统一的错误状态码
// 然后根据返回的状态码进行一些操作，例如登录过期提示，错误提示等等
// 下面列举几个常见的操作，其他需求可自行扩展
    error => {
if (error.response.status) {
switch (error.response.status) {
// 401: 未登录
// 未登录则跳转登录页面，并携带当前页面的路径
// 在登录成功后返回当前页面，这一步需要在登录页操作。
case401:
                    router.replace({
path: '/login',
query: {
redirect: router.currentRoute.fullPath
                        }
                    });
break;



// 403 token过期
// 登录过期对用户进行提示
// 清除本地token和清空vuex中token对象
// 跳转登录页面
case403:
                     Toast({
message: '登录过期，请重新登录',
duration: 1000,
forbidClick: true
                    });
// 清除token
                    localStorage.removeItem('token');
                    store.commit('loginSuccess', null);
// 跳转登录页面，并将要浏览的页面fullPath传过去，登录成功后跳转需要访问的页面
                    setTimeout(() => {
                        router.replace({
path: '/login',
query: {
redirect: router.currentRoute.fullPath
                            }
                        });
                    }, 1000);
break;


请求不存在
                case 404:
```

```
                    Toast({
                        message: '网络请求不存在',
                        duration: 1500,
                        forbidClick: true
                    });
                    break;
                // 其他错误，直接抛出错误提示
                default:
                    Toast({
                        message: error.response.data.message,
                        duration: 1500,
                        forbidClick: true
                    });
            }
            return Promise.reject(error.response);
        }
    }
});
```

响应拦截器很好理解，就是服务器返回给我们的数据，我们在拿到之前可以对他进行一些处理。例如上面的思想：如果后台返回的状态码是200，则正常返回数据，否则的根据错误的状态码类型进行一些我们需要的错误，其实这里主要就是进行了错误的统一处理和没登录或登录过期后调整登录页的一个操作。

要注意的是，上面的Toast()方法，是我引入的vant库中的toast轻提示组件，你根据你的ui库，对应使用你的一个提示组件。

封装get方法和post方法

我们常用的ajax请求方法有get、post、put等方法，相信小伙伴都不会陌生。axios对应的也有很多类似的方法，不清楚的可以看下文档。但是为了简化我们的代码，我们还是要对其进行一个简单的封装。下面我们主要封装两个方法：get和post。

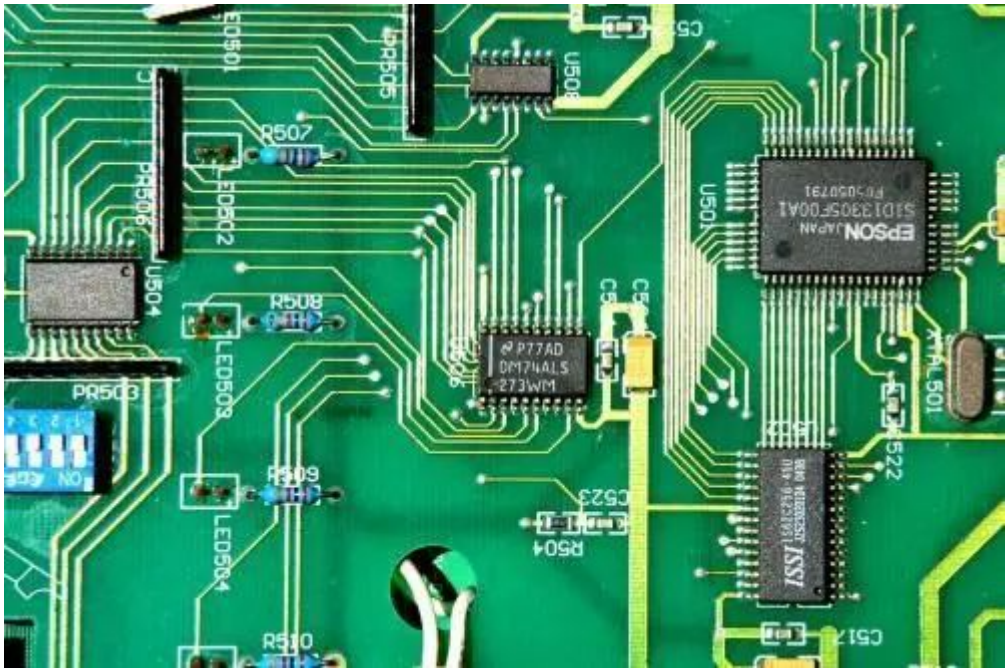
get方法：我们通过定义一个get函数，get函数有两个参数，第一个参数表示我们要请求的url地址，第二个参数是我们要携带的请求参数。get函数返回一个promise对象，当axios其请求成功时resolve服务器返回 值，请求失败时reject错误值。最后通过export抛出get函数。

```
/**
 * get方法，对应get请求
 * @param {String} url [请求的url地址]
 * @param {Object} params [请求时携带的参数]
 */
export function get(url, params){
    return new Promise((resolve, reject) =>{
        axios.get(url, {
            params: params
        }).then(res => {
            resolve(res.data);
        }).catch(err =>{
            reject(err.data)
    })
});}
```

post方法：原理同get基本一样，但是要注意的是，post方法必须要使用对提交从参数对象进行序列化的操作，所以这里我们通过node的qs模块来序列化我们的参数。这个很重要，如果没有序列化操作，后台是拿不到你提交的数据的。这就是文章开头我们import QS from 'qs';的原因。如果不明白序列化是什么意思的，就百度一下吧，答案一大堆。

```
/**
 * post方法，对应post请求
 * @param {String} url [请求的url地址]
 * @param {Object} params [请求时携带的参数]
 */
export function post(url, params) {
    return new Promise((resolve, reject) => {
         axios.post(url, QS.stringify(params))
        .then(res => {
            resolve(res.data);
        })
        .catch(err =>{
            reject(err.data)
        })
    });
}
```

这里有个小细节说下，`axios.get()`方法和`axios.post()`在提交数据时参数的书写方式还是有区别的。区别就是，get的第二个参数是一个{}，然后这个对象的params属性值是一个参数对象的。而post的第二个参数就是一个参数对象。两者略微的区别要留意哦！



axios的封装基本就完成了，下面再简单说下api的统一管理。

整齐的api就像电路板一样，即使再复杂也能很清晰整个线路。上面说了，我们会新建一个api.js,然后在这个文件中存放我们所有的api接口。

首先我们在api.js中引入我们封装的get和post方法

```
/**
 * api接口统一管理
 */
import { get, post } from './http'
```

现在，例如我们有这样一个接口，是一个post请求：

```
http://www.baiodu.com/api/v1/users/my_address/address_edit_before
```

我们可以在api.js中这样封装：

```
export const apiAddress = p => post('api/v1/users/my_address/address_edit_before',
```

我们定义了一个 `apiAddress` 方法，这个方法有一个参数p，p是我们请求接口时携带的参数对象。而后调用了我们封装的 `post` 方法， `post` 方法的第一个参数是我们的接口地址，第二个参数是 `apiAddress` 的p参数，即请求接口时携带的参数对象。最后通过export导出 `apiAddress` 。

然后在我们的页面中可以这样调用我们的api接口：

```
import { apiAddress } from '@/request/api';// 导入我们的api接口
export default {
    name: 'Address',
    created () {
        this.onLoad();
    },
    methods: {
        // 获取数据
        onLoad() {
            // 调用api接口，并且提供了两个参数
            apiAddress({
                type: 0,
                sort: 1
            }).then(res => {
                // 获取数据成功后的其他操作
                ……
            })
        }
    }
}
```

其他的api接口，就在pai.js中继续往下面扩展就可以了。友情提示，为每个接口写好注释哦！！！

api接口管理的一个好处就是，我们把api统一集中起来，如果后期需要修改接口，我们就直接在api.js中找到对应的修改就好了，而不用去每一个页面查找我们的接口然后再修改会很麻烦。关键是，万一修改的量比较大，就规格gg了。还有就是如果直接在我们的业务代码修改接口，一不小心还容易动到我们的业务代码造成不必要的麻烦。

好了，最后把完成的axios封装代码奉上。

```
/**axios封装
 * 请求拦截、相应拦截、错误统一处理
 */
import axios from'axios';import QS from'qs';
import { Toast } from'vant';
import store from'../store/index'



// 环境的切换
if (process.env.NODE_ENV == 'development') {
    axios.defaults.baseURL = '/api';
} elseif (process.env.NODE_ENV == 'debug') {
    axios.defaults.baseURL = '';
} elseif (process.env.NODE_ENV == 'production') {
    axios.defaults.baseURL = 'http://api.123dailu.com/';
}



// 请求超时时间
axios.defaults.timeout = 10000;

// post请求头
axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-
urlencoded;charset=UTF-8';

// 请求拦截器
axios.interceptors.request.use(
config => {
// 每次发送请求之前判断是否存在token，如果存在，则统一在http请求的header都加上token，不用每次
请求都手动添加了
// 即使本地存在token，也有可能token是过期的，所以在响应拦截器中要对返回状态进行判断
const token = store.state.token;
        token && (config.headers.Authorization = token);
return config;
    },
    error => {
returnPromise.error(error);
    })

                                                // 响应拦截器
axios.interceptors.response.use(
    response => {
        if (response.status === 200) {
            return Promise.resolve(response);
        } else {
            return Promise.reject(response);
        }
    },
    // 服务器状态码不是200的情况
    error => {
        if (error.response.status) {
            switch (error.response.status) {
                // 401: 未登录
                // 未登录则跳转登录页面，并携带当前页面的路径
                // 在登录成功后返回当前页面，这一步需要在登录页操作。
                case 401:
                    router.replace({
                        path: '/login',
```

```
                        query: { redirect: router.currentRoute.fullPath }
                    });
                    break;
                // 403 token过期
                // 登录过期对用户进行提示
                // 清除本地token和清空vuex中token对象
                // 跳转登录页面
                case 403:
                    Toast({
                        message: '登录过期，请重新登录',
                        duration: 1000,
                        forbidClick: true
                    });
                    // 清除token
                    localStorage.removeItem('token');
                    store.commit('loginSuccess', null);
                    // 跳转登录页面，并将要浏览的页面fullPath传过去，登录成功后跳转需要访问
的页面
                    setTimeout(() => {
                        router.replace({
                            path: '/login',
                            query: {
                                redirect: router.currentRoute.fullPath
                            }
                        });
                    }, 1000);
                    break;
                // 404请求不存在
                case 404:
                    Toast({
                        message: '网络请求不存在',
                        duration: 1500,
                        forbidClick: true
                    });
                break;
                // 其他错误，直接抛出错误提示
                default:
                    Toast({
                        message: error.response.data.message,
                        duration: 1500,
                        forbidClick: true
                    });
            }
            return Promise.reject(error.response);
        }
    }
);
/**
 * get方法，对应get请求
 * @param {String} url [请求的url地址]
 * @param {Object} params [请求时携带的参数]
 */
export function get(url, params){
    return new Promise((resolve, reject) =>{
        axios.get(url, {
            params: params
        })
        .then(res => {
            resolve(res.data);
```

```
        })
        .catch(err => {
            reject(err.data)
        })
    });
}
/**
 * post方法，对应post请求
 * @param {String} url [请求的url地址]
 * @param {Object} params [请求时携带的参数]
 */
export function post(url, params) {
    return new Promise((resolve, reject) => {
        axios.post(url, QS.stringify(params))
        .then(res => {
            resolve(res.data);
        })
        .catch(err => {
            reject(err.data)
        })
    });
}
```

如果喜欢，就给个❤❤吧(*^▽^*)

**********华丽丽的分割线**********华丽丽的分割线**********华丽丽的分割线**********华丽丽的分割线**********华丽丽的分割线**********华丽丽的分割线**********

2018.8.14更新

axios的封装根据需求的不同而不同。这里非常感谢评论里一些很中肯的建议，我也对此进行了思考和针对不同需求的改善。主要有以下改变：

1.优化axios封装，去掉之前的get和post

2.断网情况处理

3.更加模块化的api管理

4.接口域名有多个的情况

5.api挂载到vue.prototype上省去引入的步骤

http.js中axios封装的优化，先直接贴代码：

```
/**
 * axios封装
 * 请求拦截、响应拦截、错误统一处理
 */
import axios from'axios';
import router from'../router';
import store from'../store/index';
import { Toast } from'vant';



/**
 * 提示函数
 * 禁止点击蒙层、显示一秒后关闭
 */
const tip = msg => {
    Toast({
message: msg,
duration: 1000,
forbidClick: true
    });
}



/**
 * 跳转登录页
 * 携带当前页面路由，以期在登录页面完成登录后返回当前页面
 */
const toLogin = () => {
    router.replace({
path: '/login',
query: {
redirect: router.currentRoute.fullPath
        }
    });
}



/**
 * 请求失败后的错误统一处理
 * @param {Number} status 请求失败的状态码
 */
const errorHandle = (status, other) => {
// 状态码判断
switch (status) {
// 401: 未登录状态，跳转登录页
case401:
            toLogin();
break;
// 403 token过期
// 清除token并跳转登录页
case403:
            tip('登录过期，请重新登录');
            localStorage.removeItem('token');
            store.commit('loginSuccess', null);
            setTimeout(() => {
                toLogin();
            }, 1000);
```

```
break;
// 404请求不存在
case404:
            tip('请求的资源不存在');
break;
default:
console.log(other);
        }}



// 创建axios实例
var instance = axios.create({    timeout: 1000 * 12});
// 设置post请求头
instance.defaults.headers.post['Content-Type'] = 'application/x-www-form-
urlencoded';
/**
 * 请求拦截器
 * 每次请求前，如果存在token则在请求头中携带token
 */
instance.interceptors.request.use(
config => {
// 登录流程控制中，根据本地是否存在token判断用户的登录情况
// 但是即使token存在，也有可能token是过期的，所以在每次的请求头中携带token
// 后台根据携带的token判断用户的登录情况，并返回给我们对应的状态码
// 而后我们可以在响应拦截器中，根据状态码进行一些统一的操作。
const token = store.state.token;
        token && (config.headers.Authorization = token);
return config;
    },
    error => Promise.error(error))



// 响应拦截器
instance.interceptors.response.use(
// 请求成功
res => res.status === 200 ? Promise.resolve(res) : Promise.reject(res),
// 请求失败
    error => {
const { response } = error;
if (response) {
// 请求已发出，但是不在2xx的范围
            errorHandle(response.status, response.data.message);
returnPromise.reject(response);
        } else {
// 处理断网的情况
// eg:请求超时或断网时，更新state的network状态
// network状态在app.vue中控制着一个全局的断网提示组件的显示隐藏
// 关于断网组件中的刷新重新获取数据，会在断网组件中说明
if (!window.navigator.onLine) {
               store.commit('changeNetwork', false);
            } else {
returnPromise.reject(error);
            }
        }
    });
```

这个axios和之前的大同小异，做了如下几点改变：

1.去掉了之前get和post方法的封装，通过创建一个axios实例然后export default方法导出，这样使用起来更灵活一些。

2.去掉了通过环境变量控制baseUrl的值。考虑到接口会有多个不同域名的情况，所以准备通过js变量来控制接口域名。这点具体在api里会介绍。

3.增加了请求超时，即断网状态的处理。说下思路，当断网时，通过更新vuex中network的状态来控制断网提示组件的显示隐藏。断网提示一般会有重新加载数据的操作，这步会在后面对应的地方介绍。

4.公用函数进行抽出，简化代码，尽量保证单一职责原则。

下面说下api这块，考虑到一下需求：

1.更加模块化

2.更方便多人开发，有效减少解决命名冲突

3.处理接口域名有多个情况

这里这里呢新建了一个api文件夹，里面有一个index.js和一个base.js，以及多个根据模块划分的接口js文件。index.js是一个api的出口，base.js管理接口域名，其他js则用来管理各个模块的接口。

先放index.js代码：

```
/**
 * api接口的统一出口
 */
// 文章模块接口
import article from'@/api/article';
// 其他模块的接口……

 // 导出接口
export default {
    article,
    // ……
}
```

index.js是一个api接口的出口，这样就可以把api接口根据功能划分为多个模块，利于多人协作开发，比如一个人只负责一个模块的开发等，还能方便每个模块中接口的命名哦。

base.js:

```
/**
 * 接口域名的管理
 */
const base = {
sq: 'https://xxxx111111.com/api/v1',
bd: 'http://xxxxx22222.com/api'
}

        export default base;
```

通过base.js来管理我们的接口域名，不管有多少个都可以通过这里进行接口的定义。即使修改起来，也是很方便的。

最后就是接口模块的说明，例如上面的article.js:

```
/**
 * article模块接口列表
 */

import base from'./base'; // 导入接口域名列表
import axios from'@/utils/http'; // 导入http中创建的axios实例
import qs from'qs'; // 根据需求是否导入qs模块



const article = {
// 新闻列表
    articleList () {
return axios.get(`${base.sq}/topics`);
    },
// 新闻详情,演示
    articleDetail (id, params) {
return axios.get(`${base.sq}/topic/${id}`, {
params: params
        });
    },
// post提交
    login (params) {
return axios.post(`${base.sq}/accesstoken`, qs.stringify(params));
    }
// 其他接口……
}

                                                    export default article;
```

1.通过直接引入我们封装好的axios实例，然后定义接口、调用axios实例并返回，可以更灵活的使用axios，比如你可以对post请求时提交的数据进行一个qs序列化的处理等。

2.请求的配置更灵活，你可以针对某个需求进行一个不同的配置。关于配置的优先级，axios文档说的很清楚，这个顺序是：在 `lib/defaults.js` 找到的库的默认值，然后是实例的 `defaults` 属性，最后是请求的 `config` 参数。后者将优先于前者。

3.restful风格的接口，也可以通过这种方式灵活的设置api接口地址。

最后，为了方便api的调用，我们需要将其挂载到vue的原型上。在main.js中：

```
import Vue from'vue'
import App from'./App'
import router from'./router'// 导入路由文件
import store from'./store'// 导入vuex文件
import api from'./api'// 导入api接口

        Vue.prototype.$api = api; // 将api挂载到vue的原型上
```

然后我们可以在页面中这样调用接口，eg：

```
methods: {
    onLoad(id) {
        this.$api.article.articleDetail(id, {
            api: 123
        }).then(res=> {
            // 执行某些操作
        })
    }
}
```

再提一下断网的处理，这里只做一个简单的示例：

```
<template>
<div id="app">
<div v-if="!network">
<h3>我没网了</h3>
<div @click="onRefresh">刷新</div>
</div>
<router-view/>
</div>
</template>


    import { mapState } from 'vuex';
    export default {
        name: 'App',
        computed: {
            ...mapState(['network'])
        },
        methods: {
            // 通过跳转一个空页面再返回的方式来实现刷新当前页面数据的目的
            onRefresh () {
                this.$router.replace('/refresh')
            }
        }
    }
</script>
```

这是app.vue，这里简单演示一下断网。在http.js中介绍了，我们会在断网的时候，来更新vue中network的状态，那么这里我们根据network的状态来判断是否需要加载这个断网组件。断网情况下，加载断网组件，不加载对应页面的组件。当点击刷新的时候，我们通过

跳转refesh页面然后立即返回的方式来实现重新获取数据的操作。因此我们需要新建一个refresh.vue页面，并在其 `beforeRouteEnter` 钩子中再返回当前页面。

```
// refresh.vue
beforeRouteEnter (to, from, next) {
    next(vm => {
        vm.$router.replace(from.fullPath)
    })
}
```

这是一种全局通用的断网提示，当然了，也可以根据自己的项目需求操作。具体操作就仁者见仁智者见智了。

如果更多的需求，或者说是不一样的需求，可以根据自己的需求进行一个改进。

> 转自：愣锤
>
> https://juejin.im/post/6844903652881072l4

- EOF -

推荐阅读 点击标题可跳转
1、Axios 如何取消重复请求？

2、axios 是如何封装 HTTP 请求的

3、GitHub 四万星标的 axios！如何写一个像它那样牛叉的请求库

觉得本文对你有帮助？请分享给更多人

推荐关注「前端大全」，提升前端技能

前端大全
点击获取精选前端开发资源。「前端大全」日常分享 Web 前端相关的技术文章、实用案例、工具资源、精选课程、热点资讯。

201篇原创内容

公众号

点赞和在看就是最大的支持❤️