**一、二次封装axios并使用**

1.新增axios.js文件，内容如下（二次封装axios,方便我们后续项目的使用）

参数的数据格式分两种

1.json对象类型  {name:1,password:1356565}

2.字符串类型    name:1,password:1356598   (需要下载qs插件进行转换)

import axios from 'axios'

import Qs from 'qs'（如果只是get请求就不需要引qs）

import { Message } from 'element-ui'  //引入element中的弹窗，这个不是必须的可根据项目来

axios.defaults.withCredentials = false;  //false  不跨域  true 跨域

axios.defaults.headers.common['token'] = localstorage.getItem('token'); //设置请求头，这个是不是必须的

export  default *function*(*path*, *params*, *method* = "GET", *headerType* = "json")=>{

*let* baseURL = "", data = {}

    baseURL = process.env.BASE\_URL //这里是在config里面配置了请求路径

      //设置请求头

    if (method === 'post') {   //post请求

        if (headerType === "json") {

            axios.defaults.headers["Content-Type"] = 'application/json;charset=UTF-8'

            data = params

        } else {

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

            data = Qs.stringify(params);

        }

    }

    //get请求

    if (method === 'get') {

        if (headerType == !'json') {

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

        }

        data = Qs.stringify(params);

        path = path + '?' + data

        data = {};

    }

     // 请求拦截

    axios.interceptors.request.use(*function* (*config*) {

*let* token = storage.getStorage("token")

        if (token) {

*config*.headers['token'] = token;

        }

        return *config*;

       }, *function* (*error*) {

        return *Promise*.reject(*error*);

       });

      //响应拦截

      axios.interceptors.response.use(*response=>*{

          if (*response*.headers.token) {

            storage.setStorage('token', *response*.headers.token)

          }

          if (*response*.data.statusCode == 20009) {  //这里的状态码是根据后台设置的来

               Message.error({ message: '登录过期，请重新登录' })

           }

           return *response*;

           },*error=>*{

              return *Promise*.resolve(*error*.response)

        })

       //发送请求

      return new *Promise*(()*=>*{

         axios({

            baseURL

            method

            url: path,

            headers，

            data:params,

            timeout:6000,

            }).then(*result=>*{

                resolve(*result*.data)

            }).catch(*err=>*{

                reject(*err*)

            })

         })

}

2.新增api.js文件，内容如下（项目中所有的请求地址以及方式等都放在这个文件中，并导出）

import http from '@/utils/axios'  //引入我们二次封装的axios.js文件

a. post请求，参数是json类型

          export *const* passwordUpdate = *function* (*params*) {

*const* url= '/account/accountUpdate';  //请求的地址

                return https(PASSWORDUPDARE, *params*, "post");

          };

b.post请求，参数是字符串类型

           export *const* passwordUpdate = *function* (*params*) {

*const* url= "/account/accountUpdate";  //请求的地址

                return https(PASSWORDUPDARE , *params*, "post", "from");

            };

c.get请求

           export *const* passwordUpdate = *function* (*params*) {

*const* url= "/account/accountUpdate";  //请求的地址

                return https(PASSWORDUPDARE , *params*);

            };

3.页面中使用我们封装好的请求

1.先引入：import { requestLogin } from '@/api/api'  //需要调用哪个接口，就引入哪个，按需引入

2.使用

*let* params = {

  name:'zhangsan',

  password:'123456'

}

requestLogin(params ).then(*res=>*{   //我们只需要传递一个参数，其他的都已经在api.js中写好了

    //获取到后台响应的参数res，并进行操作

})

**（法二）、介绍axios的基本使用(vue中使用axios)**

**特点**

1. 支持Promise API
2. 拦截请求和响应。拦截请求，可以过滤请求参数；拦截响应，可以处理响应异常
3. 取消请求。请求可以手动取消

**vue中使用axios**

1. 安装axios模块  
   npm install --save axios
2. 在index.js文件中引入axios

import axios from 'axios'

new Vue({

// el: '#app',

// router,

axios // 加入axios

})

**基本API**

**1. 执行get请求，有两种方式**

// 第一种方式 将参数直接写在url中

axios.get('/getMainInfo?id=123')

.then((res) => {

console.log(res)

})

.catch((err) => {

console.log(err)

})

// 第二种方式 将参数直接写在params中

axios.get('/getMainInfo', {

params: {

id: 123

}

})

.then((res) => {

console.log(res)

})

.catch((err) => {

console.log(err)

})

**2. 执行post请求，注意执行post请求的入参，不需要写在params字段中，这个地方要注意与get请求的第二种方式进行区别。**

axios.post('/getMainInfo', {

id: 123

})

.then((res) => {

console.log(res)

})

.catch((err) => {

console.log(err)

})

**3. 通过向axios传递的相关配置来创建请求axios(config)**

**(1)下面是常用的配置字段：**

* url：请求路径，**string类型**
* method：请求方法，**string类型**
* baseURL：baseURL会自动加在url前面，除非url是一个绝对URL，**string类型**
* transformRequest：允许在向服务器发送请求之前，修改请求数据，只适用于**post，put，patch**请求，**数组类型，数组里面的最后一个函数必须返回一个字符串**

[fucntion(data) {

// 对data进行更改

return data

}]

* transformResponse：这里提前处理返回的数据
* headers：自定义请求头
* params：即将于请求一起发送的url参数（一般只用于get请求）**一般是对象类型**
* data：作为请求主体被发送是数据（一般只用于post请求）**一般是对象类型**
* timeout：超时时间，超过时间，请求将被中断
* withCredentials：表示跨域请求时是否需要使用凭证
* responseType：响应数据的类型，默认是'json'，可以是'text', 'json','document','arraybuffer','blob','stream'
* maxContentLength：响应数据的最大尺寸
* proxy：代理服务器主机名称和端口

proxy: {

host: '127.0.0.1',

port: 9000,

auth: : {

username: 'mikeymike',

password: 'rapunz3l'

}

},

**(2)下面是响应的数据接口**

{

data: // 服务器响应数据

status: // http状态码

statusText: // 服务器响应的http状态信息

headers: // 响应头

config: // 请求配置数据

}

**(3)举例**

// 发送 POST 请求

axios({

method: 'post',

url: '/getMainInfo',

data: {

id: 123

}

}).then(res => {

console.log(res.data)

console.log(res.status)

console.log(res.statusText)

console.log(res.headers)

console.log(res.config)

}).catch(err => {

console.log(err)

})

**4. 配置默认值default**

axios.default.baseURL = ''  
axios.default.headers.common['Authorization'] = AUTH\_TOKEN  
...等等默认配置

**5. 执行多个并发**

axios.all()和Promise.all()执行机制是一样的，要么全部成功走then，要么就走catch

function getUserName() {

return axios.get('/getUseName?id=123')

}

function getUserAge() {

return axios.get('getUserAge?id=123')

}

axios.all([getUserName(), getUserAge()]) .then(

axios.spread(function(acct, perms) { // 处理并发请求的助手函数

console.log(acct, perms) // 全部请求成功

})).catch(err => {

console.log(err) // 只要任意一个请求出错

})

**6. 拦截器**

**(1)在请求之前拦截请求**

// 添加请求拦截器

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

// 在发送请求之前做些什么

return config;

}, function (error) {

// 对请求错误做些什么

return Promise.reject(error);

})

**(2)在被then，catch处理前拦截响应**

// 添加响应拦截器

axios.interceptors.response.use(function (response) {

// 对响应数据做点什么

return response;

}, function (error) {

// 对响应错误做点什么

return Promise.reject(error);

});

**(3)可以为自定义 axios 实例添加拦截器**

var instance = axios.create([config]);

instance.interceptors.request.use(function () {/\*...\*/});

**7. 封装axios**

在开发中，总会有很多的http请求，那我们就要封装好一个axios，使用更方便一些

**(1)封装**

// request.js文件

import axios from 'axios'

import qs from 'qs'

// 引入基础参数文件 和 baseURL配置文件

import baseParams from './baseParams' // 基本参数，比如一些登录信息之类的参数

import { BASE\_URL } from './config' // baseURL写在config.js文件中

// 默认的全局配置

axios.defaults.baseURL = BASE\_URL

axios.defaults.timeout = 10000

// 请求发送之前拦截，进行处理（根据业务需求进行拦截处理）

axios.interceptors.request.use(success => {

return success

}, error => {

return reject(error)

})

// then,catch处理之前，进行拦截处理（根据业务需求进行拦截处理）

axios.interceptors.response.use(response => {

return response

}, error => {

return Promise.reject(error)

})

// 导出post请求

export function post (url, data, withBaseParams = false) {

return new Promise((resolve, reject) => {

axios({

method: 'post',

url,

data: withBaseParams ? qs.stringify({...baseParams, data}) : qs.stringify(data),

headers: {

'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8'

}

})

.then(res => successHandle(res, resolve)) // 将数据处理之后传给页面

.catch(err => errorHandle(err, reject))

})

}

// 导出get请求

export function get (url, params) {

return new Promise((resolve, reject) => {

axios({

method: 'get',

url,

params,

headers: {}

})

.then(res => successHandle(res, resolve))

.catch(err => errorHandle(err, reject))

})

}

// then处理执行了successHandle

function successHandle(data, resolve) {

if (res.success && res.errorCode == '60000') {

resolve(res)

} else {

// 弹出toast报错

Toast({

message: res.msg,

duration: 2000

})

}

}

// catch处理执行了errorHandle

function errorHandle(err, reject) {

reject(err)

}

**(2)引入使用**

// 引入使用

import {get, post} from 'request.js'

post('/getMainInfo', {id: 123}, true)

.then(res => {

console.log(res)

})

.catch(err => {

console.log(err)

})

**VUE 中封装axios**

Npm安装

npm install --save axios

Main.js中引入

import axios from 'axios'

import qs from 'axios'

import api from './axios/api.js'

Vue.prototype.$axios = api

Vue.prototype.$qs = qs

import VueAxios from 'vue-axios'

Vue.use(VueAxios, axios)

实例商挂载

var vm = new Vue({

    el: '#app',

    render(createElements) {

        return createElements(App)

    },

    router,

    store,

    axios,

})

创建文件夹并在文件夹axios下创建文件api.js和axios.js

api.js用于管理接口

axios.js用于管理axiosde 封装

在api.js 中写入

import { get, post } from './http.js'

/\*

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：

\*/

const apiAddress = p => post('api/v1/users/my\_address/address\_edit\_before', p);

const api = [

    apiAddress()

]

export default api

在axios.js 中写入

// http.js文件用来封装我们的axios

// 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

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';

}

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

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

// 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/store.js';

axios.interceptors.request.use( //请求拦截器

    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);

    })

// 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) {

            return Promise.resolve(response);

        } else {

            return Promise.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。

                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({ //vant的toast提示框组件

                        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)

            })

    });

}