近来在使用vue重构公司m站时，使用了axios来进行数据的请求，由于项目的需要，对axios进行了二次封装，[点击进入axios](https://github.com/axios/axios" \t "_blank)

|  |
| --- |
| //引入axios  import axios from 'axios'    let cancel ,promiseArr = {}  const CancelToken = axios.CancelToken;  //请求拦截器  axios.interceptors.request.use(config => {      //发起请求时，取消掉当前正在进行的相同请求      if (promiseArr[config.url]) {          promiseArr[config.url]('操作取消')          promiseArr[config.url] = cancel      } else {          promiseArr[config.url] = cancel      }        return config  }, error => {      return Promise.reject(error)  })    //响应拦截器即异常处理  axios.interceptors.response.use(response => {      return response  }, error => {      if (error && err.response) {        switch (err.response.status) {          case 400:            err.message = '错误请求'            break;          case 401:            err.message = '未授权，请重新登录'            break;          case 403:            err.message = '拒绝访问'            break;          case 404:            err.message = '请求错误,未找到该资源'            break;          case 405:            err.message = '请求方法未允许'            break;          case 408:            err.message = '请求超时'            break;          case 500:            err.message = '服务器端出错'            break;          case 501:            err.message = '网络未实现'            break;          case 502:            err.message = '网络错误'            break;          case 503:            err.message = '服务不可用'            break;          case 504:            err.message = '网络超时'            break;          case 505:            err.message = 'http版本不支持该请求'            break;          default:            err.message = `连接错误${err.response.status}`        }      } else {        err.message = "连接到服务器失败"      }      message.error(err.message)        return Promise.resolve(error.response)  })    axios.defaults.baseURL = '/api'  //设置默认请求头  axios.defaults.headers = {      'X-Requested-With': 'XMLHttpRequest'  }  axios.defaults.timeout = 10000    export default {    //get请求      get (url,param) {        return new Promise((resolve,reject) => {          axios({            method: 'get',            url,            params: param,            cancelToken: new CancelToken(c => {              cancel = c            })          }).then(res => {            resolve(res)          })        })      },    //post请求      post (url,param) {        return new Promise((resolve,reject) => {          axios({            method: 'post',            url,            data: param,            cancelToken: new CancelToken(c => {              cancel = c            })          }).then(res => {            resolve(res)          })        })       }    } |

**说明**

1.为防止发起请求时，当前正在进行的相同请求，在请求拦截器中加入了hash判断，将相同请求url拦截  
2.将axios中get，post公共配置抽离出来

|  |
| --- |
| axios.defaults.baseURL = '/api'  //设置默认请求头  axios.defaults.headers = {      'X-Requested-With': 'XMLHttpRequest'  }  axios.defaults.timeout = 10000 |

3.get，post请求的封装

可能你会问，这里的axios返回的就是promise对象，为什么还要再次对get,post封装一次promise.因为我这边的话，在开发中使用async await会出现数据请求失败的情况，报的错就是返回的不是promise对象。(ps：可async await返回的就是promise呀,这个问题后续再搞一下)就直接return了一个promise对象，以避免上面的错误。下面是请求接口的一个例子

|  |
| --- |
| import req from '../api/requestType'  /\*\*   4. 拼团详情   \*/  export const groupDetail = param => {      return req.get('/RestHome/GroupDetail',param)  } |

下面是数据的获取

|  |
| --- |
| async getData() {      const params = {          TopCataID: 0,          pageNumber: this.pageNumber,          pageSize: this.pageSize      }      const res = await groupList(params)  }, |

1.在相应拦截器中对请求常见的错误进行了全局异常处理

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

到这里我们就简单的封装了一下适合自己项目的axios

原文链接：<https://segmentfault.com/a/1190000012332982>