http://www.axios-js.com/zh-cn/docs/index.html

（1）Axios是一个基于promise的HTTP库，可以用在浏览器和node.js中。

（2）axios API

（2.1）axios(config)

（2.2）axios(url[, config]) 默认get请求

（3）请求方法的别名

axios.request(config)

axios.get(url[, config])

axios.delete(url[, config])

axios.head(url[, config])

axios.options(url[, config])

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

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

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

（4）并发

axios.all([functionName1(), functionName2()]).then(axios.spread(function (functionReturn, functionReturn2) {

}));

（5）创建实例

const axiosInstance = axios.create(config);

（6）请求配置

{

url: '/url',

method: 'get',

baseURL: 'https://baseURL/', //加在url前面。若url是绝对url，则baseURL失效

transformRequest: [function (data, headers) { //向服务器发送前，修改请求数据。仅用于post，put，patch。返回字符串或ArrayBuffer或Stream

return data;

}],

transformResponse: [function (data) { //then/catch 前，修改响应数据

return data;

}],

headers: {'headerName': 'headerValue'}, //自定义请求头

params: { //url参数，是plain object或URLSearchParams

},

paramsSerializer: function(params) { //params序列化

return qs.stringify(params, {arrayFormat: 'brackets'})

},

//请求体数据。仅用于post，put，patch。

// 若没有设置 `transformRequest`，必须是以下类型之一：

// string, plain object, ArrayBuffer, ArrayBufferView, URLSearchParams

// 浏览器专属：FormData, File, Blob

// Node专属：Stream

data: {

},

timeout: 1000, //请求超时的毫秒数，0为无超时时间。

withCredentials: false, //跨域请求是否需要凭证

adapter: function (config) { //自定义处理请求，返回promise

},

auth: { //使用HTTP基础验证，设置并覆盖headers的Authorization

},

responseType: 'json', //服务器响应的数据类型，可以是 'arraybuffer', 'blob', 'document', 'json', 'text', 'stream'

responseEncoding: 'utf8', //解码响应。responseType是'stream'或者浏览器的请求会被忽略

xsrfCookieName: 'XSRF-TOKEN', //xsrf token的cookie的名称

xsrfHeaderName: 'X-XSRF-TOKEN', //xsrf token的header的名称

onUploadProgress: function (progressEvent) { //上传处理进度

},

onDownloadProgress: function (progressEvent) { //下载处理进度

},

maxContentLength: 2000, //响应内容的最大尺寸

validateStatus: function (status) { //根据状态码，promise是resolve还是reject。返回true，null，undefined是resolve。否则是rejecte

return status >= 200 && status < 300;

},

maxRedirects: 5, //在node.js中follow的最大重定向数目。设置为0，将不会follow重定向

socketPath: null, //在node.js中使用的UNIX Socket。socketPath和proxy都可以使用，socketPath优先级更高

httpAgent: new http.Agent({ keepAlive: false }), //在node.js中用于定义执行http时使用的自定义代理

httpsAgent: new https.Agent({ keepAlive: false }), //在node.js中用于定义执行https时使用的自定义代理

proxy:{

host: '127.0.0.1', //代理服务器的主机名

port: 9000, //代理服务器的端口

auth: { //使用HTTP基础验证连接代理，设置并覆盖headers的Proxy-Authorization

}

},

cancelToken: new CancelToken(function (cancel) { //取消请求

})

}

（7）响应结构

使用then时的response：

{

data: {},

status: 200, //服务器响应状态码

statusText: 'OK', //服务器响应状态信息

headers: {}, //服务器响应头

config: {}, //请求的配置信息

request: {} node.js端重定向最后一次ClientRequest实例或浏览器端XMLHttpRequest实例

}

（8）配置默认值

（8.1）全局的 axios 默认值

axios.defaults.config中项

（8.2）自定义实例默认值

axiosInstance.defaults.config中项

（8.3）配置的优先顺序

请求config>axiosInstance.defaults>axios.defaults

（9）拦截器

（9.1）请求拦截器定义

const 请求拦截器名=axios.interceptors.request.use(function (config) {

//请求发送之前处理

return config;

},function (error) {

//对请求错误之后处理

return Promise.reject(error);

});

（9.2）请求拦截器移除

axios.interceptors.request.eject(请求拦截器名);

（9.3）响应拦截器定义

const 响应拦截器名=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);

});

（9.4）响应拦截器移除

axios.interceptors.response.eject(响应拦截器名);

（9.5）可为axiosInstance添加拦截器

（10）错误处理

.catch(function (error) {

if (error.response) {

//有请求有返回，返回状态码不对

console.log(error.response.data);

console.log(error.response.status);

console.log(error.response.headers);

} else if (error.request) {

//有请求无返回

console.log(error.request);

} else {

//设置请求出错

console.log('Error', error.message);

}

console.log(error.config);

});

（11）取消

（11.1.1）

const CancelToken = axios.CancelToken;

const source = CancelToken.source();

（11.1.2）

{

cancelToken: source.token

}

（11.1.3）

source.cancel('message');

（11.1.4）

.catch(function(thrown) {

if (axios.isCancel(thrown)) {

console.log(thrown.message);

} else {

//处理错误

}

});

（11.2.1）

const CancelToken = axios.CancelToken;

let cancel;

（11.2.2）

{

cancelToken: new CancelToken(function executor(cancel) {

cancel = cancel;

})

}

（11.2.3）

cancel();

（11.3）可以使用同一个cancelToken取消多个请求

（12）使用 application/x-www-form-urlencoded format

默认，axios将对象序列化为JSON。要以application / x-www-form-urlencoded格式发送数据，可以:

qs.stringify({})