1.axios配置项:

{

// `url`是将用于请求的服务器URL

url: '/user',

// `method`是发出请求时使用的请求方法

method: 'get', // 默认

// `baseURL`将被添加到`url`前面，除非`url`是绝对的。

// 可以方便地为 axios 的实例设置`baseURL`，以便将相对 URL 传递给该实例的方法。

baseURL: 'https://some-domain.com/api/',

// `transformRequest`允许在请求数据发送到服务器之前对其进行更改

// 这只适用于请求方法'PUT'，'POST'和'PATCH'

// 数组中的最后一个函数必须返回一个字符串，一个 ArrayBuffer或一个 Stream

transformRequest: [function (data) {

// 做任何你想要的数据转换

return data;

}],

// `transformResponse`允许在 then / catch之前对响应数据进行更改

transformResponse: [function (data) {

// Do whatever you want to transform the data

return data;

}],

// `headers`是要发送的自定义 headers

headers: {'X-Requested-With': 'XMLHttpRequest'},

// `params`是要与请求一起发送的URL参数

// 必须是纯对象或URLSearchParams对象

params: {

ID: 12345

},

// `paramsSerializer`是一个可选的函数，负责序列化`params`

// (e.g. https://www.npmjs.com/package/qs, http://api.jquery.com/jquery.param/)

paramsSerializer: function(params) {

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

},

// `data`是要作为请求主体发送的数据

// 仅适用于请求方法“PUT”，“POST”和“PATCH”

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

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

// - Browser only: FormData, File, Blob

// - Node only: Stream

data: {

firstName: 'Fred'

},

// `timeout`指定请求超时之前的毫秒数。

// 如果请求的时间超过'timeout'，请求将被中止。

timeout: 1000,

// `withCredentials`指示是否跨站点访问控制请求

// should be made using credentials

withCredentials: false, // default

// `adapter'允许自定义处理请求，这使得测试更容易。

// 返回一个promise并提供一个有效的响应（参见[response docs]（＃response-api））

adapter: function (config) {

/\* ... \*/

},

// `auth'表示应该使用 HTTP 基本认证，并提供凭据。

// 这将设置一个`Authorization'头，覆盖任何现有的`Authorization'自定义头，使用`headers`设置。

auth: {

username: 'janedoe',

password: 's00pers3cret'

},

// “responseType”表示服务器将响应的数据类型

// 包括 'arraybuffer', 'blob', 'document', 'json', 'text', 'stream'

responseType: 'json', // default

//`xsrfCookieName`是要用作 xsrf 令牌的值的cookie的名称

xsrfCookieName: 'XSRF-TOKEN', // default

// `xsrfHeaderName`是携带xsrf令牌值的http头的名称

xsrfHeaderName: 'X-XSRF-TOKEN', // default

// `onUploadProgress`允许处理上传的进度事件

onUploadProgress: function (progressEvent) {

// 使用本地 progress 事件做任何你想要做的

},

// `onDownloadProgress`允许处理下载的进度事件

onDownloadProgress: function (progressEvent) {

// Do whatever you want with the native progress event

},

// `maxContentLength`定义允许的http响应内容的最大大小

maxContentLength: 2000,

// `validateStatus`定义是否解析或拒绝给定的promise

// HTTP响应状态码。如果`validateStatus`返回`true`（或被设置为`null` promise将被解析;否则，promise将被

  // 拒绝。

validateStatus: function (status) {

return status >= 200 && status < 300; // default

},

// `maxRedirects`定义在node.js中要遵循的重定向的最大数量。

// 如果设置为0，则不会遵循重定向。

maxRedirects: 5, // 默认

// `httpAgent`和`httpsAgent`用于定义在node.js中分别执行http和https请求时使用的自定义代理。

// 允许配置类似`keepAlive`的选项，

// 默认情况下不启用。

httpAgent: new http.Agent({ keepAlive: true }),

httpsAgent: new https.Agent({ keepAlive: true }),

// 'proxy'定义代理服务器的主机名和端口

// `auth`表示HTTP Basic auth应该用于连接到代理，并提供credentials。

// 这将设置一个`Proxy-Authorization` header，覆盖任何使用`headers`设置的现有的`Proxy-Authorization` 自定义 headers。

proxy: {

host: '127.0.0.1',

port: 9000,

auth: : {

username: 'mikeymike',

password: 'rapunz3l'

}

},

// “cancelToken”指定可用于取消请求的取消令牌

// (see Cancellation section below for details)

cancelToken: new CancelToken(function (cancel) {

})

}

2.使用 then 时，您将收到如下响应：

|  |  |
| --- | --- |
| axios.get('/user/12345')  .then(function(response) {  console.log(response.data);  console.log(response.status);  console.log(response.statusText);  console.log(response.headers);  console.log(response.config);  }); |  |