## 一、安装

1、 利用npm安装npm install axios --save  
2、 利用bower安装bower install axios --save  
3、 直接利用cdn引入<script src="https://unpkg.com/axios/dist/axios.min.js"></script>

## 二、例子

1、 发送一个GET请求

//通过给定的ID来发送请求

axios.get('/user?ID=12345')

.then(function(response){

console.log(response);

})

.catch(function(err){

console.log(err);

});//以上请求也可以通过这种方式来发送

axios.get('/user',{

params:{

ID:12345

}

})

.then(function(response){

console.log(response);

})

.catch(function(err){

console.log(err);

});

2、 发送一个POST请求

axios.post('/user',{

firstName:'Fred',

lastName:'Flintstone'

})

.then(function(res){

console.log(res);

})

.catch(function(err){

console.log(err);

});

3、 一次性并发多个请求

function getUserAccount(){

return axios.get('/user/12345');

}function getUserPermissions(){

return axios.get('/user/12345/permissions');

}

axios.all([getUserAccount(),getUserPermissions()])

.then(axios.spread(function(acct,perms){

//当这两个请求都完成的时候会触发这个函数，两个参数分别代表返回的结果

}))

## 三、axios的API

#### （一） axios可以通过配置（config）来发送请求

1、 axios(config)

//发送一个`POST`请求

axios({

method:"POST",

url:'/user/12345',

data:{

firstName:"Fred",

lastName:"Flintstone"

}

});

2、 axios(url[,config])

//发送一个`GET`请求（默认的请求方式）

axios('/user/12345');

#### （二）、 请求方式的别名，这里对所有已经支持的请求方式都提供了方便的别名

axios.request(config);

axios.get(url[,config]);

axios.delete(url[,config]);

axios.head(url[,config]);

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

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

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

* 注意：当我们在使用别名方法的时候，url,method,data这几个参数不需要在配置中声明

#### （三）、 并发请求（concurrency）,即是帮助处理并发请求的辅助函数

//iterable是一个可以迭代的参数如数组等

axios.all(iterable)//callback要等到所有请求都完成才会执行

axios.spread(callback)

#### （四）、创建一个axios实例，并且可以自定义其配置

1、 axios.create([config])

var instance = axios.create({

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

timeout:1000,

headers: {'X-Custom-Header':'foobar'}

});

2、 实例的方法

* 一下是实例方法，注意已经定义的配置将和利用create创建的实例的配置合并

axios#request(config)

axios#get(url[,config])

axios#delete(url[,config])

axios#head(url[,config])

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

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

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

## 四、请求的配置（request config）

* 以下就是请求的配置选项，只有url选项是必须的，如果method选项未定义，那么它默认是以GET的方式发出请求

{

//`url`是请求的服务器地址

url:'/user',

//`method`是请求资源的方式

method:'get'//default

//如果`url`不是绝对地址，那么`baseURL`将会加到`url`的前面

//当`url`是相对地址的时候，设置`baseURL`会非常的方便

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

//`transformRequest`选项允许我们在请求发送到服务器之前对请求的数据做出一些改动

//该选项只适用于以下请求方式：`put/post/patch`

//数组里面的最后一个函数必须返回一个字符串、-一个`ArrayBuffer`或者`Stream`

transformRequest:[function(data){

//在这里根据自己的需求改变数据

return data;

}],

//`transformResponse`选项允许我们在数据传送到`then/catch`方法之前对数据进行改动

transformResponse:[function(data){

//在这里根据自己的需求改变数据

return data;

}],

//`headers`选项是需要被发送的自定义请求头信息

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

//`params`选项是要随请求一起发送的请求参数----一般链接在URL后面

//他的类型必须是一个纯对象或者是URLSearchParams对象

params: {

ID:12345

},

//`paramsSerializer`是一个可选的函数，起作用是让参数（params）序列化

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

paramsSerializer: function(params){

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

},

//`data`选项是作为一个请求体而需要被发送的数据

//该选项只适用于方法：`put/post/patch`

//当没有设置`transformRequest`选项时dada必须是以下几种类型之一

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

//仅仅浏览器：FormData/File/Bold

//仅node:Stream

data {

firstName:"Fred"

},

//`timeout`选项定义了请求发出的延迟毫秒数

//如果请求花费的时间超过延迟的时间，那么请求会被终止

timeout:1000,

//`withCredentails`选项表明了是否是跨域请求

withCredentials:false,//default

//`adapter`适配器选项允许自定义处理请求，这会使得测试变得方便

//返回一个promise,并提供验证返回

adapter: function(config){

/\*..........\*/

},

//`auth`表明HTTP基础的认证应该被使用，并提供证书

//这会设置一个authorization头（header）,并覆盖你在header设置的Authorization头信息

auth: {

username:"zhangsan",

password: "s00sdkf"

},

//返回数据的格式

//其可选项是arraybuffer,blob,document,json,text,stream

responseType:'json',//default

//

xsrfCookieName: 'XSRF-TOKEN',//default

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

//`onUploadProgress`上传进度事件

onUploadProgress:function(progressEvent){

//下载进度的事件

onDownloadProgress:function(progressEvent){

}

},

//相应内容的最大值

maxContentLength:2000,

//`validateStatus`定义了是否根据http相应状态码，来resolve或者reject promise

//如果`validateStatus`返回true(或者设置为`null`或者`undefined`),那么promise的状态将会是resolved,否则其状态就是rejected

validateStatus:function(status){

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

},

//`maxRedirects`定义了在nodejs中重定向的最大数量

maxRedirects: 5,//default

//`httpAgent/httpsAgent`定义了当发送http/https请求要用到的自定义代理

//keeyAlive在选项中没有被默认激活

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

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

//proxy定义了主机名字和端口号，

//`auth`表明http基本认证应该与proxy代理链接，并提供证书

//这将会设置一个`Proxy-Authorization` header,并且会覆盖掉已经存在的`Proxy-Authorization` header

proxy: {

host:'127.0.0.1',

port: 9000,

auth: {

username:'skda',

password:'radsd'

}

},

//`cancelToken`定义了一个用于取消请求的cancel token

//详见cancelation部分

cancelToken: new cancelToken(function(cancel){

})

}

## 五、请求返回的内容

{

data:{},

status:200,

//从服务器返回的http状态文本

statusText:'OK',

//响应头信息

headers: {},

//`config`是在请求的时候的一些配置信息

config: {}

}

* 你可以这样来获取响应信息

axios.get('/user/12345')

.then(function(res){

console.log(res.data);

console.log(res.status);

console.log(res.statusText);

console.log(res.headers);

console.log(res.config);

})

## 六、默认配置

* 你可以设置默认配置，对所有请求都有效

1、 全局默认配置

axios.defaults.baseURL = 'http://api.exmple.com';

axios.defaults.headers.common['Authorization'] = AUTH\_TOKEN;

axios.defaults.headers.post['content-Type'] = 'appliction/x-www-form-urlencoded';

2、 自定义的实例默认设置

//当创建实例的时候配置默认配置var instance = axios.create({

baseURL: 'https://api.example.com'

});

//当实例创建时候修改配置

instance.defaults.headers.common["Authorization"] = AUTH\_TOKEN;

3、 配置中的有优先级

* config配置将会以优先级别来合并，顺序是lib/defauts.js中的默认配置，然后是实例中的默认配置，最后是请求中的config参数的配置，越往后等级越高，后面的会覆盖前面的例子。

//创建一个实例的时候会使用libray目录中的默认配置//在这里timeout配置的值为0，来自于libray的默认值var instance = axios.create();//回覆盖掉library的默认值//现在所有的请求都要等2.5S之后才会发出

instance.defaults.timeout = 2500;//这里的timeout回覆盖之前的2.5S变成5s

instance.get('/longRequest',{

timeout: 5000

});

## 七、拦截器

* 你可以在请求、响应在到达then/catch之前拦截他们

//添加一个请求拦截器

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

//在请求发出之前进行一些操作

return config;

},function(err){

//Do something with request error

return Promise.reject(error);

});//添加一个响应拦截器

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

//在这里对返回的数据进行处理

return res;

},function(err){

//Do something with response error

return Promise.reject(error);

})

2、取消拦截器

var myInterceptor = axios.interceptor.request.use(function(){/\*....\*/});

axios.interceptors.request.eject(myInterceptor);

3、 给自定义的axios实例添加拦截器

var instance = axios.create();

instance.interceptors.request.use(function(){})

## 八、错误处理

axios.get('/user/12345')

.catch(function(error){

if(error.response){

//请求已经发出，但是服务器响应返回的状态吗不在2xx的范围内

console.log(error.response.data);

console.log(error.response.status);

console.log(error.response.header);

}else {

//一些错误是在设置请求的时候触发

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

}

console.log(error.config);

});

## 九、取消

* 你可以通过一个cancel token来取消一个请求

1. 你可以通过CancelToken.source工厂函数来创建一个cancel token

var CancelToken = axios.CancelToken;var source = CancelToken.source();

axios.get('/user/12345',{

cancelToken: source.token

}).catch(function(thrown){

if(axios.isCancel(thrown)){

console.log('Request canceled',thrown.message);

}else {

//handle error

}

});

//取消请求（信息的参数可以设置的）

source.cance("操作被用户取消");

1. 你可以给cancelToken构造函数传递一个executor function来创建一个cancel token:

var cancelToken = axios.CancelToken;var cance;

axios.get('/user/12345',{

cancelToken: new CancelToken(function(c){

//这个executor函数接受一个cancel function作为参数

cancel = c;

})

})//取消请求

cancel();