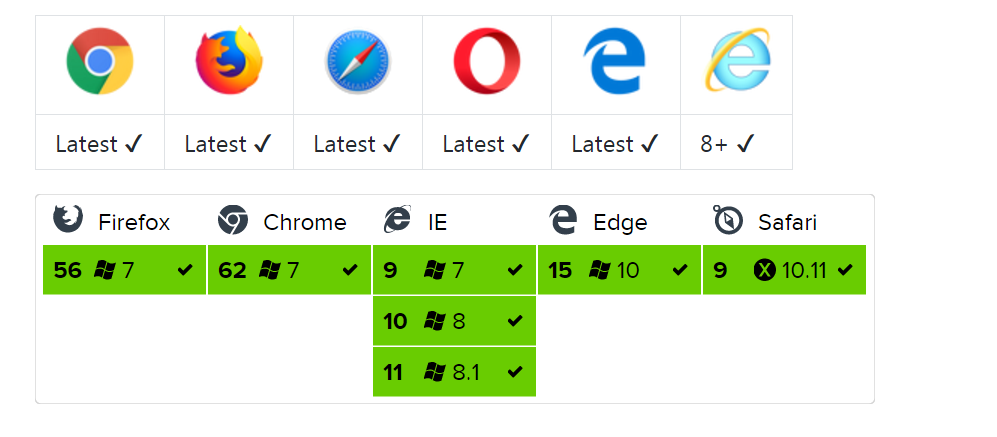
## axios使用指南

## 1、axios 简介

axios 是一个基于Promise 用于浏览器和 nodejs 的 HTTP 客户端，它本身具有以下特征：

* 从浏览器中创建 XMLHttpRequest
* 从 node.js 发出 http 请求
* 支持 Promise API
* 拦截请求和响应
* 转换请求和响应数据
* 取消请求
* 自动转换JSON数据
* 客户端支持防止 CSRF/XSRF

## 2、浏览器兼容性



## 3、引入方式

1. 使用npm

$ npm install axios

1. 使用 bower

$ bower install axios

1. 使用cdn：

<script src="https://unpkg.com/axios/dist/axios.min.js"></script>

## 4、示例

1）发起一个GET请求

//发起一个user请求，参数为给定的ID

axios.get('/user?ID=1234')

.then(function(respone){

console.log(response);

})

.catch(function(error){

console.log(error);

});

//上面的请求也可选择下面的方式来写

axios.get('/user',{

params:{

ID:12345

}

})

.then(function(response){

console.log(response);

})

.catch(function(error){

console.log(error)

});

2）发起一个POST请求

axios.post('/user',{

firstName:'friend',

lastName:'Flintstone'

})

.then(function(response){

console.log(response);

})

.catch(function(error){

console.log(error);

});

3）发起一个多重并发请求

function getUserAccount(){

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

}

function getUserPermissions(){

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

}

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

.then(axios.spread(function(acc,pers){

//两个请求现在都完成

}));

## 5、axios API

可以对axios进行一些设置来生成请求。

axios(config)

//发起 POST请求

axios({

method:'post',//方法

url:'/user/12345',//地址

data:{//参数

firstName:'Fred',

lastName:'Flintstone'

}

});

//通过请求获取远程图片axios({

method:'get',

url:'http://bit.ly/2mTM3Ny',

responseType:'stream'

})

.then(function(response){

response.data.pipe(fs.createWriteStream('ada\_lovelace.jpg'))

})

axios(url[,config])

//发起一个GET请求

axios('/user/12345/);

## 6、请求方法的重命名

为了方便，axios提供了所有请求方法的重命名支持

****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]])****

注意：当时用重命名方法时url,method,以及data属性不需要在config中指定。

## 7、并发 Concurrency

有用的方法

****axios.all(iterable)  
axios.spread(callback)****

## 8、创建一个实例

你可以使用自定义设置创建一个新的实例

****axios.create([config])****

var instance = axios.create({

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

timeout:1000,

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

});

## 9、实例方法

下面列出了一些实例方法。具体的设置将在实例设置中被合并。

****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]])****

## 10、Requese Config请求设置

以下列出了一些请求时的设置选项。只有url是必须的，如果没有指明method的话，默认的请求方法是GET.

{

//`url`是服务器链接，用来请求

url:'/user',

//`method`是发起请求时的请求方法

method:`get`,

//`baseURL`如果`url`不是绝对地址，那么将会加在其前面。

//当axios使用相对地址时这个设置非常方便

//在其实例中的方法

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

//`transformRequest`允许请求的数据在传到服务器之前进行转化。

//这个只适用于`PUT`,`GET`,`PATCH`方法。

//数组中的最后一个函数必须返回一个字符串或者一个`ArrayBuffer`,或者`Stream`,`Buffer`实例,`ArrayBuffer`,`FormData`

transformRequest:[function(data){

//依自己的需求对请求数据进行处理

return data;

}],

//`transformResponse`允许返回的数据传入then/catch之前进行处理

transformResponse:[function(data){

//依需要对数据进行处理

return data;

}],

//`headers`是自定义的要被发送的头信息

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

//`params`是请求连接中的请求参数，必须是一个纯对象，或者URLSearchParams对象

params:{

ID:12345

},

//`paramsSerializer`是一个可选的函数，是用来序列化参数

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

paramsSerializer: function(params){

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

},

//`data`是请求提需要设置的数据

//只适用于应用的'PUT','POST','PATCH'，请求方法

//当没有设置`transformRequest`时，必须是以下其中之一的类型（不可重复？）：

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

//-仅浏览器：FormData,File,Blob

//-仅Node：Stream

data:{

firstName:'fred'

},

//`timeout`定义请求的时间，单位是毫秒。

//如果请求的时间超过这个设定时间，请求将会停止。

timeout:1000,

//`withCredentials`表明是否跨网站访问协议，

//应该使用证书

withCredentials:false //默认值

//`adapter`适配器，允许自定义处理请求，这会使测试更简单。

//返回一个promise，并且提供验证返回（查看[response docs](#response-api)）

adapter:function(config){

/\*...\*/

},

//`auth`表明HTTP基础的认证应该被使用，并且提供证书。

//这个会设置一个`authorization` 头（header），并且覆盖你在header设置的Authorization头信息。

auth:{

username:'janedoe',

password:'s00pers3cret'

},

//`responsetype`表明服务器返回的数据类型，这些类型的设置应该是

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

responsetype:'json',

//`xsrfHeaderName` 是http头（header）的名字，并且该头携带xsrf的值

xrsfHeadername:'X-XSRF-TOKEN'，//默认值

//`onUploadProgress`允许处理上传过程的事件

onUploadProgress: function(progressEvent){

//本地过程事件发生时想做的事

},

//`onDownloadProgress`允许处理下载过程的事件

onDownloadProgress: function(progressEvent){

//下载过程中想做的事

},

//`maxContentLength` 定义http返回内容的最大容量

maxContentLength: 2000,

//`validateStatus` 定义promise的resolve和reject。

//http返回状态码，如果`validateStatus`返回true（或者设置成null/undefined），promise将会接受；其他的promise将会拒绝。

validateStatus: function(status){

return status >= 200 && stauts < 300;//默认

},

//`httpAgent` 和 `httpsAgent`当产生一个http或者https请求时分别定义一个自定义的代理，在nodejs中。

//这个允许设置一些选选个，像是`keepAlive`--这个在默认中是没有开启的。

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

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

//`proxy`定义服务器的主机名字和端口号。

//`auth`表明HTTP基本认证应该跟`proxy`相连接，并且提供证书。

//这个将设置一个'Proxy-Authorization'头(header)，覆盖原先自定义的。

proxy:{

host:127.0.0.1,

port:9000,

auth:{

username:'cdd',

password:'123456'

}

},

//`cancelTaken` 定义一个取消，能够用来取消请求

//（查看 下面的Cancellation 的详细部分）

cancelToken: new CancelToken(function(cancel){

})

}

## 11、返回响应概要 Response Schema

一个请求的返回包含以下信息

{

//`data`是服务器的提供的回复（相对于请求）

data{},

//`status`是服务器返回的http状态码

status:200,

//`statusText`是服务器返回的http状态信息

statusText: 'ok',

//`headers`是服务器返回中携带的headers

headers:{},

//`config`是对axios进行的设置，目的是为了请求（request）

config:{}

}

使用then，你将接收下面这样的响应：

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);

});

使用catch时，或者传入一个reject callback作为then的第二个参数，那么返回的错误信息将能够被使用。

## 12、默认设置（Config Default)

你可以设置一个默认的设置，这设置将在所有的请求中有效。

### 1）全局默认设置 Global axios defaults

axios.defaults.baseURL = 'https://api.example.com';

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

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

### 2）实例中自定义默认值 Custom instance default

//当创建一个实例时进行默认设置

var instance = axios.create({

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

});

//在实例创建之后改变默认值

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

### 3）设置优先级 Config order of precedence

设置(config)将按照优先顺序整合起来。首先的是在lib/defaults.js中定义的默认设置，其次是defaults实例属性的设置，最后是请求中config参数的设置。越往后面的等级越高，会覆盖前面的设置。  
看下面这个例子：

//使用默认库的设置创建一个实例，

//这个实例中，使用的是默认库的timeout设置，默认值是0。

var instance = axios.create();

//覆盖默认库中timeout的默认值

//此时，所有的请求的timeout时间是2.5秒

instance.defaults.timeout = 2500;

//覆盖该次请求中timeout的值，这个值设置的时间更长一些

instance.get('/longRequest',{

timeout:5000

});

## 13、拦截器 interceptors

你可以在请求或者返回被then或者catch处理之前对他们进行拦截。

//添加一个请求拦截器

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

//在请求发送之前做一些事

return config;

},function(error){

//当出现请求错误是做一些事

return Promise.reject(error);

});

//添加一个返回拦截器

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

//对返回的数据进行一些处理

return response;

},function(error){

//对返回的错误进行一些处理

return Promise.reject(error);

});

如果你需要在稍后移除拦截器,你可以

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

axios.interceptors.rquest.eject(myInterceptor);

你可以在一个axios实例中使用拦截器

var instance = axios.create();

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

## 14、错误处理 Handling Errors

axios.get('user/12345')

.catch(function(error){

if(error.response){

//存在请求，但是服务器的返回一个状态码

//他们都在2xx之外

console.log(error.response.data);

console.log(error.response.status);

console.log(error.response.headers);

}else{

//一些错误是在设置请求时触发的

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

}

console.log(error.config);

});

你可以使用validateStatus设置选项自定义HTTP状态码的错误范围。

axios.get('user/12345',{

validateStatus:function(status){

return status < 500;//当返回码小于等于500时视为错误

}

});

## 15、取消 Cancellation

你可以使用cancel token取消一个请求

axios的cancel token API是基于\*\*cnacelable promises proposal\*\*，其目前处于第一阶段。

你可以使用CancelToken.source工厂函数创建一个cancel token，如下：

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

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

cancelToken:source.toke

}).catch(function(thrown){

if(axiso.isCancel(thrown)){

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

}else{

//handle error

}

});

//取消请求(信息参数设可设置的)

source.cancel("操作被用户取消");

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 cancel;

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

cancelToken: new CancelToken(function executor(c){

//这个executor 函数接受一个cancel function作为参数

cancel = c;

})

});

//取消请求

cancel();

注意：你可以使用同一个cancel token取消多个请求。