# axios

基于 Promise 的 HTTP 请求客户端，可同时在浏览器和 node.js 中使用

## 功能特性

* 在浏览器中发送 [XMLHttpRequests](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest" \t "http://blog.csdn.net/binginsist/article/details/_blank) 请求
* 在 node.js 中发送 [http](http://nodejs.org/api/http.html" \t "http://blog.csdn.net/binginsist/article/details/_blank)请求
* 支持 [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise" \t "http://blog.csdn.net/binginsist/article/details/_blank) API
* 拦截请求和响应
* 转换请求和响应数据
* 自动转换 JSON 数据
* 客户端支持保护安全免受 [XSRF](http://en.wikipedia.org/wiki/Cross-site_request_forgery" \t "http://blog.csdn.net/binginsist/article/details/_blank) 攻击

## 浏览器支持

## 安装

使用 bower:

$ bower install axios

使用 npm:

$ npm install axios

## 例子

发送一个 GET 请求

// Make a request for a user with a given ID

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

.then(function (response) {

console.log(response);

})

.catch(function (response) {

console.log(response);

});

// Optionally the request above could also be done as

axios.get('/user', {

params: {

ID: 12345

}

})

.then(function (response) {

console.log(response);

})

.catch(function (response) {

console.log(response);

});

发送一个 POST 请求

axios.post('/user', {

firstName: 'Fred',

lastName: 'Flintstone'

})

.then(function (response) {

console.log(response);

})

.catch(function (response) {

console.log(response);

});

发送多个并发请求

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) {

// Both requests are now complete

}));

## axios API

可以通过给 axios传递对应的参数来定制请求：

##### axios(config)

// Send a POST requestaxios({

method: 'post',

url: '/user/12345',

data: {

firstName: 'Fred',

lastName: 'Flintstone'

}});

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

// Sned a GET request (default method)axios('/user/12345');

### 请求方法别名

为方便起见，我们为所有支持的请求方法都提供了别名

##### 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

当使用别名方法时， url、 method 和 data 属性不需要在 config 参数里面指定。

### 并发

处理并发请求的帮助方法

##### axios.all(iterable)

##### axios.spread(callback)

### 创建一个实例

你可以用自定义配置创建一个新的 axios 实例。

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

var instance = axios.create({

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

timeout: 1000,

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

### 实例方法

所有可用的实例方法都列在下面了，指定的配置将会和该实例的配置合并。

##### 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]])

## 请求配置

下面是可用的请求配置项，只有 url 是必需的。如果没有指定 method ，默认的请求方法是 GET。

{

// `url` is the server URL that will be used for the request

url: '/user',

// `method` is the request method to be used when making the request

method: 'get', // default

// `baseURL` will be prepended to `url` unless `url` is absolute.

// It can be convenient to set `baseURL` for an instance of axios to pass relative URLs

// to methods of that instance.

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

// `transformRequest` allows changes to the request data before it is sent to the server

// This is only applicable for request methods 'PUT', 'POST', and 'PATCH'

// The last function in the array must return a string or an ArrayBuffer

transformRequest: [function (data) {

// Do whatever you want to transform the data

return data;

}],

// `transformResponse` allows changes to the response data to be made before

// it is passed to then/catch

transformResponse: [function (data) {

// Do whatever you want to transform the data

return data;

}],

// `headers` are custom headers to be sent

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

// `params` are the URL parameters to be sent with the request

params: {

ID: 12345

},

// `paramsSerializer` is an optional function in charge of serializing `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` is the data to be sent as the request body

// Only applicable for request methods 'PUT', 'POST', and 'PATCH'

// When no `transformRequest` is set, must be a string, an ArrayBuffer or a hash

data: {

firstName: 'Fred'

},

// `timeout` specifies the number of milliseconds before the request times out.

// If the request takes longer than `timeout`, the request will be aborted.

timeout: 1000,

// `withCredentials` indicates whether or not cross-site Access-Control requests

// should be made using credentials

withCredentials: false, // default

// `adapter` allows custom handling of requests which makes testing easier.

// Call `resolve` or `reject` and supply a valid response (see [response docs](#response-api)).

adapter: function (resolve, reject, config) {

/\* ... \*/

},

// `auth` indicates that HTTP Basic auth should be used, and supplies credentials.

// This will set an `Authorization` header, overwriting any existing

// `Authorization` custom headers you have set using `headers`.

auth: {

username: 'janedoe',

password: 's00pers3cret'

}

// `responseType` indicates the type of data that the server will respond with

// options are 'arraybuffer', 'blob', 'document', 'json', 'text'

responseType: 'json', // default

// `xsrfCookieName` is the name of the cookie to use as a value for xsrf token

xsrfCookieName: 'XSRF-TOKEN', // default

// `xsrfHeaderName` is the name of the http header that carries the xsrf token value

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

// `progress` allows handling of progress events for 'POST' and 'PUT uploads'

// as well as 'GET' downloads

progress: function(progressEvent) {

// Do whatever you want with the native progress event

}}

## 响应的数据结构

响应的数据包括下面的信息：

{

// `data` is the response that was provided by the server

data: {},

// `status` is the HTTP status code from the server response

status: 200,

// `statusText` is the HTTP status message from the server response

statusText: 'OK',

// `headers` the headers that the server responded with

headers: {},

// `config` is the config that was provided to `axios` for the request

config: {}}

当使用 then 或者 catch 时, 你会收到下面的响应：

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);});

## 默认配置

你可以为每一个请求指定默认配置。

### 全局 axios 默认配置

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';

### 自定义实例默认配置

// Set config defaults when creating the instancevar instance = axios.create({

baseURL: 'https://api.example.com'});

// Alter defaults after instance has been created

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

### 配置的优先顺序

Config will be merged with an order of precedence. The order is library defaults found in lib/defaults.js, then defaults property of the instance, and finally config argument for the request. The latter will take precedence over the former. Here's an example.

// Create an instance using the config defaults provided by the library// At this point the timeout config value is `0` as is the default for the libraryvar instance = axios.create();

// Override timeout default for the library// Now all requests will wait 2.5 seconds before timing out

instance.defaults.timeout = 2500;

// Override timeout for this request as it's known to take a long time

instance.get('/longRequest', {

timeout: 5000});

## 拦截器

你可以在处理 then 或 catch 之前拦截请求和响应

// 添加一个请求拦截器

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

// Do something before request is sent

return config;

}, function (error) {

// Do something with request error

return Promise.reject(error);

});

// 添加一个响应拦截器

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

// Do something with response data

return response;

}, function (error) {

// Do something with response error

return Promise.reject(error);

});

移除一个拦截器：

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

axios.interceptors.request.eject(myInterceptor);

你可以给一个自定义的 axios 实例添加拦截器：

var instance = axios.create();

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

## 错误处理

axios.get('/user/12345')

.catch(function (response) {

if (response instanceof Error) {

// Something happened in setting up the request that triggered an Error

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

} else {

// The request was made, but the server responded with a status code

// that falls out of the range of 2xx

console.log(response.data);

console.log(response.status);

console.log(response.headers);

console.log(response.config);

}

});

## Promises

axios 依赖一个原生的 ES6 Promise 实现，如果你的浏览器环境不支持 ES6 Promises，你需要引入 [polyfill](https://github.com/jakearchibald/es6-promise" \t "http://blog.csdn.net/binginsist/article/details/_blank)

## TypeScript

axios 包含一个 [TypeScript](http://typescriptlang.org/" \t "http://blog.csdn.net/binginsist/article/details/_blank) 定义

/// <reference path="axios.d.ts" />

import \* as axios from 'axios';

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

## Credits

axios is heavily inspired by the [$http service](https://docs.angularjs.org/api/ng/service/$http" \t "http://blog.csdn.net/binginsist/article/details/_blank) provided in [Angular](https://angularjs.org/" \t "http://blog.csdn.net/binginsist/article/details/_blank). Ultimately axios is an effort to provide a standalone $http-like service for use outside of Angular.

## License

MIT