axios

Promise based HTTP client for the browser and node.js

# Features

* Make [XMLHttpRequests](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) from the browser
* Make [http](http://nodejs.org/api/http.html) requests from node.js
* Supports the [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) API
* Intercept request and response
* Transform request and response data
* Cancel requests
* Automatic transforms for JSON data
* Client side support for protecting against [XSRF](http://en.wikipedia.org/wiki/Cross-site_request_forgery)

跨站点请求伪造

# Installing

Using npm:

$ npm install axios

Using bower:

$ bower install axios

Using cdn:

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

# Example

## Performing a GET request

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

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

  .then(function (response) {

    console.log(response);

  })

  .catch(function (error) {

    console.log(error);

  });

*// Optionally the request above could also be done as*

axios.get('/user', {

    params**:** {

      ID**:** 12345

    }

  })

  .then(function (response) {

    console.log(response);

  })

  .catch(function (error) {

    console.log(error);

  });

## Performing a POST request

axios.post('/user', {

    firstName**:** 'Fred',

    lastName**:** 'Flintstone'

  })

  .then(function (response) {

    console.log(response);

  })

  .catch(function (error) {

    console.log(error);

  });

## Performing multiple concurrent requests

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

Requests can be made by passing the relevant config to axios.

### axios(config)

*// Send a POST request*

axios({

  method**:** 'post',

  url**:** '/user/12345',

  data**:** {

    firstName**:** 'Fred',

    lastName**:** 'Flintstone'

  }

});

### // GET request for remote image

axios({

  method**:**'get',

  url**:**'http://bit.ly/2mTM3nY',

  responseType**:**'stream'

})

  .then(function(response) {

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

});

### axios(url[, config])

*// Send a GET request (default method)*

axios('/user/12345');

## Request method aliases

For convenience aliases have been provided for all supported request methods.

### 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]])

###### NOTE

When using the alias methods url, method, and data properties don't need to be specified in config.

## Concurrency

Helper functions for dealing with concurrent requests.

### axios.all(iterable)

### axios.spread(callback)

## Creating an instance

You can create a new instance of axios with a custom config.

### axios.create([config])

var instance **=** axios.create({

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

  timeout**:** 1000,

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

});

## Instance methods

The available instance methods are listed below. The specified config will be merged with the instance config.

### 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]])

# Request Config

These are the available config options for making requests. Only the url is required. Requests will default to GET if method is not specified.

{

*// `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 instance of Buffer, ArrayBuffer,*

*// FormData or Stream*

*// You may modify the headers object.*

  transformRequest**:** [function (data, headers) {

*// 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*

*// Must be a plain object or a URLSearchParams object*

  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 of one of the following types:*

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

*// - Browser only: FormData, File, Blob*

*// - Node only: Stream, Buffer*

  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.*

*// Return a promise and supply a valid response (see lib/adapters/README.md).*

  adapter**:** function (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', 'stream'*

  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*

*// `onUploadProgress` allows handling of progress events for uploads*

  onUploadProgress**:** function (progressEvent) {

*// Do whatever you want with the native progress event*

  },

*// `onDownloadProgress` allows handling of progress events for downloads*

  onDownloadProgress**:** function (progressEvent) {

*// Do whatever you want with the native progress event*

  },

*// `maxContentLength` defines the max size of the http response content allowed*

  maxContentLength**:** 2000,

*// `validateStatus` defines whether to resolve or reject the promise for a given*

*// HTTP response status code. If `validateStatus` returns `true` (or is set to `null`*

*// or `undefined`), the promise will be resolved; otherwise, the promise will be*

*// rejected.*

  validateStatus**:** function (status) {

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

  },

*// `maxRedirects` defines the maximum number of redirects to follow in node.js.*

*// If set to 0, no redirects will be followed.*

  maxRedirects**:** 5, *// default*

*// `socketPath` defines a UNIX Socket to be used in node.js.*

*// e.g. '/var/run/docker.sock' to send requests to the docker daemon.*

*// Only either `socketPath` or `proxy` can be specified.*

*// If both are specified, `socketPath` is used.*

  socketPath**:** null, *// default*

*// `httpAgent` and `httpsAgent` define a custom agent to be used when performing http*

*// and https requests, respectively, in node.js. This allows options to be added like*

*// `keepAlive` that are not enabled by default.*

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

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

*// 'proxy' defines the hostname and port of the proxy server*

*// Use `false` to disable proxies, ignoring environment variables.*

*// `auth` indicates that HTTP Basic auth should be used to connect to the proxy, and*

*// supplies credentials.*

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

*// `Proxy-Authorization` custom headers you have set using `headers`.*

  proxy**:** {

    host**:** '127.0.0.1',

    port**:** 9000,

    auth**:** {

      username**:** 'mikeymike',

      password**:** 'rapunz3l'

    }

  },

*// `cancelToken` specifies a cancel token that can be used to cancel the request*

*// (see Cancellation section below for details)*

  cancelToken**:** **new** CancelToken(function (cancel) {

  })

}

# Response Schema

The response for a request contains the following information.

{

*// `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*

*// All header names are lower cased*

  headers**:** {},

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

  config**:** {},

*// `request` is the request that generated this response*

*// It is the last ClientRequest instance in node.js (in redirects)*

*// and an XMLHttpRequest instance the browser*

  request**:** {}

}

When using then, you will receive the response as follows:

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);

  });

When using catch, or passing a [rejection callback](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then) as second parameter of then, the response will be available through the error object as explained in the [Handling Errors](https://www.npmjs.com/package/axios#handling-errors) section.

# Config Defaults 默认配置

You can specify config defaults that will be applied to every request.

## 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';

## Custom instance defaults

*// Set config defaults when creating the instance*

var instance **=** axios.create({

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

});

*// Alter defaults after instance has been created*

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

## Config order of precedence

Config will be merged with an order of precedence. The order is library defaults found in [lib/defaults.js](https://github.com/axios/axios/blob/master/lib/defaults.js#L28), 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 library*

var 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

});

# Interceptors 拦截器（通用）

You can intercept requests or responses before they are handled by then or catch.

*// Add a request interceptor*

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);

  });

*// Add a response interceptor*

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

*// Do something with response data*

**return** response;

  }, function (error) {

*// Do something with response error*

**return** Promise.reject(error);

  });

If you may need to remove an interceptor later you can.

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

axios.interceptors.request.eject(myInterceptor);

You can add interceptors to a custom instance of axios.

var instance **=** axios.create();

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

# Handling Errors 错误处理

axios.get('/user/12345')

  .catch(function (error) {

**if** (error.response) {

*// The request was made and the server responded with a status code*

*// that falls out of the range of 2xx*

      console.log(error.response.data);

      console.log(error.response.status);

      console.log(error.response.headers);

    } **else** **if** (error.request) {

*// The request was made but no response was received*

*// `error.request` is an instance of XMLHttpRequest in the browser and an instance of*

*// http.ClientRequest in node.js*

      console.log(error.request);

    } **else** {

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

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

    }

    console.log(error.config);

  });

You can define a custom HTTP status code error range using the validateStatus config option.

axios.get('/user/12345', {

  validateStatus**:** function (status) {

**return** status **<** 500; *// Reject only if the status code is greater than or equal to 500*

  }

})

# Cancellation

You can cancel a request using a cancel token.

The axios cancel token API is based on the withdrawn [cancelable promises proposal](https://github.com/tc39/proposal-cancelable-promises).

You can create a cancel token using the CancelToken.source factory as shown below:

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*

  }

});

axios.post('/user/12345', {

  name**:** 'new name'

}, {

  cancelToken**:** source.token

})

*// cancel the request (the message parameter is optional)*

source.cancel('Operation canceled by the user.');

You can also create a cancel token by passing an executor function to the CancelTokenconstructor:

var CancelToken **=** axios.CancelToken;

var cancel;

axios.get('/user/12345', {

  cancelToken**:** **new** CancelToken(function executor(c) {

*// An executor function receives a cancel function as a parameter*

    cancel **=** c;

  })

});

*// cancel the request*

cancel();

Note: you can cancel several requests with the same cancel token.

# Using application/x-www-form-urlencoded format

By default, axios serializes JavaScript objects to JSON. To send data in the application/x-www-form-urlencoded format instead, you can use one of the following options.

## Browser

In a browser, you can use the [URLSearchParams](https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams) API as follows:

var params **=** **new** URLSearchParams();

params.append('param1', 'value1');

params.append('param2', 'value2');

axios.post('/foo', params);

Note that URLSearchParams is not supported by all browsers (see [caniuse.com](http://www.caniuse.com/#feat=urlsearchparams)), but there is a [polyfill](https://github.com/WebReflection/url-search-params) available (make sure to polyfill the global environment).

Alternatively, you can encode data using the [qs](https://github.com/ljharb/qs) library:

var qs **=** require('qs');

axios.post('/foo', qs.stringify({ 'bar'**:** 123 }));

## Node.js

In node.js, you can use the [querystring](https://nodejs.org/api/querystring.html) module as follows:

var querystring **=** require('querystring');

axios.post('http://something.com/', querystring.stringify({ foo**:** 'bar' }));

You can also use the [qs](https://github.com/ljharb/qs) library.

# Semver

Until axios reaches a 1.0 release, breaking changes will be released with a new minor version. For example 0.5.1, and 0.5.4 will have the same API, but 0.6.0 will have breaking changes.

# Promises

axios depends on a native ES6 Promise implementation to be [supported](http://caniuse.com/promises). If your environment doesn't support ES6 Promises, you can [polyfill](https://github.com/jakearchibald/es6-promise).

# TypeScript

axios includes [TypeScript](http://typescriptlang.org/) definitions.

**import** axios **from** 'axios';

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

# Resources

* [Changelog](https://github.com/axios/axios/blob/master/CHANGELOG.md)
* [Upgrade Guide](https://github.com/axios/axios/blob/master/UPGRADE_GUIDE.md)
* [Ecosystem](https://github.com/axios/axios/blob/master/ECOSYSTEM.md)
* [Contributing Guide](https://github.com/axios/axios/blob/master/CONTRIBUTING.md)
* [Code of Conduct](https://github.com/axios/axios/blob/master/CODE_OF_CONDUCT.md)

# Credits

axios is heavily inspired by the [$http service](https://docs.angularjs.org/api/ng/service/$http) provided in [Angular](https://angularjs.org/). Ultimately axios is an effort to provide a standalone $http-like service for use outside of Angular.

# License

MIT

# Keywords

* [**xhr**](https://www.npmjs.com/search?q=keywords:xhr)
* [**http**](https://www.npmjs.com/search?q=keywords:http)
* [**ajax**](https://www.npmjs.com/search?q=keywords:ajax)
* [**promise**](https://www.npmjs.com/search?q=keywords:promise)
* [**node**](https://www.npmjs.com/search?q=keywords:node)