GitHub Gist Search... All gists Back to GitHub

SoftwareDevPro / axios_cheatsheet.md
Created 4 years ago • Report abuse

Subscribe Star 0 Fork 0

Code Revisions 1 Embed <script src="https://j Download ZIP

Axios is a promise based HTTP client for the browser and Node

axios_cheatsheet.md Raw

Axios

Axios is a promise based HTTP client for the browser and Node. Axios makes it easy to send asynchronous HTTP requests to REST API endpoints and perform CRUD (create,read,update,delete) operations. It can be used in vanilla plain JavaScript or with a library such as Vue or React.

Installing Axios

- npm: `npm install axios`
- pnpm: `pnpm install axios`
- bower: `bower install axios`
- yarn: `yarn add axios`
- jsDelivr CDN: `<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>`
- unpkg CDN: `<script src="https://unpkg.com/axios/dist/axios.min.js"></script>`

Request Configuration

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

Option	Description
url	the server URL that will be used for the request (e.g. '/user')
method	the request method to be used when making the request (e.g. 'get')

baseURL	`baseURL` will be prepended to `URL` unless `URL` is absolute.
transformRequest	allows changes to the request data before it is sent to the server
transformResponse	allows changes to the response data to be made before it is passed to then/catch
headers	custom headers to be sent
params	he URL parameters to be sent with the request.
paramsSerializer	optional function in charge of serializing `params`
data	data to be sent as the request body. Only applicable for PUT, POST, and PATCH
timeout	number of milliseconds before the request times out.
withCredentials	whether or not cross-site Access-Control requests should be made using credentials
adapter	allows custom handling of requests which makes testing easier.
auth	indicates that HTTP Basic auth should be used, and supplies credentials.
responseType	type of data that the server will respond with
responseEncoding	indicates encoding to use for decoding responses
xsrfCookieName	the name of the cookie to use as a value for xsrf token
xsrfHeaderName	the name of the http header that carries the xsrf token value
onUploadProgress	allows handling of progress events for uploads
onDownloadProgress	allows handling of progress events for downloads
maxContentLength	the max size of the http response content in bytes allowed
maxBodyLength	(Node only option) the max size of the http request content in bytes allowed
validateStatus	whether to resolve or reject the promise for a given
maxRedirects	the maximum number of redirects to follow in. If 0, no redirects will be followed.
socketPath	defines a UNIX Socket to be used in Node
httpAgent/httpsAgent	custom agent to be used when performing http/s requests, in Node.

proxy	defines the hostname and port of the proxy server.
cancelToken	specifies a cancel token that can be used to cancel the request
decompress	indicates whether or not the response body should be decompressed automatically.

Configuration Defaults

Global Defaults

```
axios.defaults.baseURL = 'https://example.com/api';
axios.defaults.headers.common['Authorization'] = AUTH_TOKEN;
axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded';
```

Custom Defaults

```
// Set config defaults when creating the instance
let instance = axios.create({
  baseURL: 'https://example.com/api'
});

// Alter defaults after instance has been created
instance.defaults.headers.common['Authorization'] = AUTH_TOKEN;
```

Order Of Precedence

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

// Override timeout default for the library
// Now all requests will wait 5 seconds before timing out
instance.defaults.timeout = 5000;

// Override timeout for this request as it's known to take a long time
instance.get('/longRequestUrl', {
  timeout: 60000
});
```

Response Object

The response object for a request contains the following data:

Field	Description

`data: {}`	The response that was provided by the server
`status`	HTTP status code from the server response (e.g. 200)
`statusText`	HTTP status message from the server response (e.g. 'OK')
`headers: {}`	headers that the server responded with all header names are lowercased and can be accessed using the bracket notation
`config: {}`	config that was provided to axios for the request
`request: {}`	request that generated this response it is the last ClientRequest instance in Node (in redirects) or XMLHttpRequest instance (browser)

Shorthand Methods

Method	Description
axios.request(config)	Send a REQUEST request
axios.get(url,config)	Send a GET request
axios.delete(url,config)	Send a DELETE request
axios.head(url,config)	Send a HEAD request
axios.options(url,config)	Send a OPTIONS request
axios.post(url,data,config)	Send a POST request
axios.put(url,data,config)	Send a PUT request
axios.patch(url,data,config)	Send a PATCH request

Basic Examples

GET

```
// Make a request for a user with a given ud
axios.get('/user?ID=199912112')
  .then(function (resp) {
    console.log(resp);
  })
  .catch(function (err) {
    console.log(err);
  });

// Optionally the request above could also be done as:
```

```
axios.get('/user', {
    params: { ID: 199912112 }
}).then(function (resp) {
    console.log(resp);
}).catch(function (err) {
    console.log(err);
});

// Using async/await
async function getUser() {
  try {
    const resp = await axios.get('/user?ID=199912112');
    console.log(resp);
  } catch (err) {
    console.error(err);
  }
};
```

POST

```
axios.post('/user', {
    first: 'Joey',
    last: 'Gladstone'
}).then(function (resp) {
    console.log(resp);
}).catch(function (err) {
    console.log(err);
});
```

Multiple Concurrent Requests

```
function getUserAcct() {
  return axios.get('/user/543');
}

function getUserPerms() {
  return axios.get('/user/543/permissions');
}

axios.all([getUserAcct(), getUserPerms()])
  .then(axios.spread(function (acct, perms) {
    // Both requests are now complete
}));
```

POST config

```
axios({
```

```
axios({
  method: 'POST',
  url: '/user/543',
  data: {
    first: 'Joey',
    last: 'Gladstone'
  }
});
```

GET config

```
axios({
  method: 'GET',
  url: 'url_to_stream',
  responseType: 'stream'
}).then(function(resp) {
  resp.data.pipe(fs.createWriteStream('pic.jpg'))
});
```

Create instance

```
let instance = axios.create({
  baseURL: 'https://example.com/api/',
  timeout: 1000,
  headers: {'X-Custom-Header': 'header'}
});
```

Error Handling

```
axios.get('/user/1902109').catch(function (err) {

    if (err.response) {
      // The request was made and the server responded with a status code
      // that falls out of the range of 2xx

      // ...
      // ...

    } else if (err.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

      // ...
      // ...
```

```
    } else {

      // Something happened in setting up the request that triggered an Error
      console.log('Error', err.message);
    }

    console.log(err.config);
  });
```

Using validateStatus, you can define HTTP code(s) that should throw an error.

```
axios.get('/user/1902109', {
  validateStatus: function (status) {
    // Resolve only if the status code is less than 500
    return status < 500;
  }
});
```

Using toJSON you get an object with more information about the HTTP error.

```
axios.get('/user/1902109').catch(function (err) {
  console.log(err.toJSON());
});
```

Interceptors

Using interceptors, one can intercept requests/responses before they are handled by then or catch.

```
// Add a request interceptor
axios.interceptors.request.use(function (cfg) {

  // Do something before request is sent
  return cfg;

}, function (err) {

    // Do something with request error
    return Promise.reject(err);
});

// Add a response interceptor
axios.interceptors.response.use(function (resp) {

    // Any status code that is within the range of 2xx will cause this function
    // to trigger.... do something with response data
    return resp;
```

```
    return resp;

}, function (err) {

    // Any status codes that fall outside the range of 2xx will cause this
    // function to trigger, ... do something with response error

    return Promise.reject(err);
});
```

To remove an interceptor:

```
const myIntercept = axios.interceptors.request.use(function () {

    // ...
    // ...

});

axios.interceptors.request.eject(myIntercept);
```

Adding interceptors to a custom instance of axios.

```
const customInstance = axios.create();

customInstance.interceptors.request.use(function () {

 // ...
 // ...

});
```

Cancellation

Create cancel token

```
let CancelToken = axios.CancelToken;
let cancel;

axios.get('/user/543', {
  cancelToken: new CancelToken(function executor(c) {
    // An executor function receives a cancel function as a parameter
    cancel = c;
  })
});

// cancel the request
```

```
cancel();
```

Cancel request with cancel token

```
let CancelToken = axios.CancelToken;
let source = CancelToken.source();

axios.get('/user/543', {
  cancelToken: source.token
}).catch(function(throwable) {
  if (axios.isCancel(throwable)) {
    console.log('Request canceled', throwable.message);
  } else {
    // handle error
  }
});

axios.post('/user/543', {
  name: 'name'
}, {
  cancelToken: source.token
})

// cancel the request (the message parameter is optional)
source.cancel('Operation canceled by the user.');
```

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 API as follows:

```
const p = new URLSearchParams();
p.append('param1', 'value1');
p.append('param2', 'value2');
axios.post('/url', params);
```

Note that URLSearchParams is not supported by all browsers, but there is a polyfill available (make sure to polyfill the global environment).

Alternatively, you can encode data using the qs library:

```
const qs = require('qs');
```

```
axios.post('/url', qs.stringify({ 'abc': 789 }));
```

Or in another way (ES6),

```
import qs from 'qs';

const data = { 'abc': 789 };
const options = {
  method: 'POST',
  headers: { 'content-type': 'application/x-www-form-urlencoded' },
  data: qs.stringify(data),
  url,
};

axios(options);
```

Node

Query string

In Node, you can use the querystring module as follows:

```
const querystring = require('querystring');
axios.post('http://example/', querystring.stringify({ abc: 'def' }));
```

or 'URLSearchParams' from 'url module' as follows:

```
const url = require('url');
const params = new url.URLSearchParams({ abc: 'def' });
axios.post('http://example.com/', params.toString());
```

Form data

In Node, you can use the form-data library as follows:

```
const FormData = require('form-data');

const form = new FormData();

form.append('field', 'value');
form.append('buffer', new Buffer(10));
form.append('file', fs.createReadStream('pic.jpg'));

axios.post('https://example.com', form, { headers: form.getHeaders() })
```

Alternatively, use an interceptor:

```
axios.interceptors.request.use(cfg => {
  if (cfg.data instanceof FormData) {
    Object.assign(cfg.headers, cfg.data.getHeaders());
  }
  return config;
});
```

Write Preview

Leave a comment

Markdown is supported

Paste, drop, or click to add files

Comment

Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

© 2024 GitHub, Inc.