node-fetch TS

3.2.3 • Public • Published a month ago

Node Fetch

A light-weight module that brings **Fetch API** to Node.js.

Consider supporting us on our Open Collective:

You might be looking for the v2 docs

Motivation

Instead of implementing `XMLHttpRequest` in Node.js to run browser-specific **Fetch polyfill**, why not go from native `http` to `fetch` API directly? Hence, `node-fetch`, minimal code for a `window.fetch` compatible API on Node.js runtime.

See Jason Miller's **isomorphic-unfetch** or Leonardo Quixada's **cross-fetch** for isomorphic usage (exports `node-fetch` for server-side, `whatwg-fetch` for client-side).

Features

- Stay consistent with `window.fetch` API.

- Make conscious trade-off when following WHATWG fetch spec and stream spec implementation details, document known differences.
- Use native promise and async functions.
- Use native Node streams for body, on both request and response.
- Decode content encoding (gzip/deflate/brotli) properly, and convert string output (such as `res.text()` and `res.json()`) to UTF-8 automatically.
- Useful extensions such as redirect limit, response size limit, explicit errors for troubleshooting.

Difference from client-side fetch

- See known differences:
  - As of v3.x
  - As of v2.x
- If you happen to use a missing feature that `window.fetch` offers, feel free to open an issue.
- Pull requests are welcomed too!

Installation

Current stable release (`3.x`) requires at least Node.js 12.20.0.

```
npm install node-fetch
```

Loading and configuring the module

ES Modules (ESM)

```
import fetch from 'node-fetch';
```

CommonJS

`node-fetch` from v3 is an ESM-only module - you are not able to import it with `require()`.

If you cannot switch to ESM, please use v2 which remains compatible with CommonJS. Critical bug fixes will continue to be published for v2.

```
npm install node-fetch@2
```

Alternatively, you can use the async `import()` function from CommonJS to load `node-fetch` asynchronously:

```
// mod.cjs
const fetch = (...args) => import('node-fetch').then(({default: fet
```

Providing global access

To use `fetch()` without importing it, you can patch the `global` object in node:

```
// fetch-polyfill.js
import fetch, {
  Blob,
  blobFrom,
  blobFromSync,
  File,
  fileFrom,
  fileFromSync,
  FormData,
  Headers,
  Request,
  Response,
} from 'node-fetch'

if (!globalThis.fetch) {
  globalThis.fetch = fetch
  globalThis.Headers = Headers
  globalThis.Request = Request
  globalThis.Response = Response
}

// index.js
import './fetch-polyfill'
```

```
// ...
```

Upgrading

Using an old version of node-fetch? Check out the following files:

- 2.x to 3.x upgrade guide
- 1.x to 2.x upgrade guide
- Changelog

Common Usage

NOTE: The documentation below is up-to-date with `3.x` releases, if you are using an older version, please check how to **upgrade**.

Plain text or HTML

```
import fetch from 'node-fetch';

const response = await fetch('https://github.com/');
const body = await response.text();

console.log(body);
```

JSON

```
import fetch from 'node-fetch';

const response = await fetch('https://api.github.com/users/github')
const data = await response.json();

console.log(data);
```

Simple Post

```
import fetch from 'node-fetch';

const response = await fetch('https://httpbin.org/post', {method: '
const data = await response.json();

console.log(data);
```

Post with JSON

```
import fetch from 'node-fetch';

const body = {a: 1};

const response = await fetch('https://httpbin.org/post', {
        method: 'post',
        body: JSON.stringify(body),
        headers: {'Content-Type': 'application/json'}
});
const data = await response.json();

console.log(data);
```

Post with form parameters

`URLSearchParams` is available on the global object in Node.js as of v10.0.0. See **official documentation** for more usage methods.

NOTE: The `Content-Type` header is only set automatically to `x-www-form-urlencoded` when an instance of `URLSearchParams` is given as such:

```
import fetch from 'node-fetch';

const params = new URLSearchParams();
params.append('a', 1);

const response = await fetch('https://httpbin.org/post', {method: '
```

```
const data = await response.json();

console.log(data);
```

Handling exceptions

NOTE: 3xx-5xx responses are *NOT* exceptions, and should be handled in `then()`, see the next section.

Wrapping the fetch function into a `try/catch` block will catch *all* exceptions, such as errors originating from node core libraries, like network errors, and operational errors which are instances of FetchError. See the **error handling document** for more details.

```
import fetch from 'node-fetch';

try {
        await fetch('https://domain.invalid/');
} catch (error) {
        console.log(error);
}
```

Handling client and server errors

It is common to create a helper function to check that the response contains no client (4xx) or server (5xx) error responses:

```
import fetch from 'node-fetch';

class HTTPResponseError extends Error {
        constructor(response, ...args) {
                super(`HTTP Error Response: ${response.status} ${re
                this.response = response;
        }
}

const checkStatus = response => {
        if (response.ok) {
```

```
                // response.status >= 200 && response.status < 300
                return response;
        } else {
                throw new HTTPResponseError(response);
        }
}

const response = await fetch('https://httpbin.org/status/400');

try {
        checkStatus(response);
} catch (error) {
        console.error(error);

        const errorBody = await error.response.text();
        console.error(`Error body: ${errorBody}`);
}
```

Handling cookies

Cookies are not stored by default. However, cookies can be extracted and passed by manipulating request and response headers. See **Extract Set-Cookie Header** for details.

Advanced Usage

Streams

The "Node.js way" is to use streams when possible. You can pipe `res.body` to another stream. This example uses **stream.pipeline** to attach stream error handlers and wait for the download to complete.

```
import {createWriteStream} from 'node:fs';
import {pipeline} from 'node:stream';
import {promisify} from 'node:util'
import fetch from 'node-fetch';

const streamPipeline = promisify(pipeline);
```

```
const response = await fetch('https://github.githubassets.com/image

if (!response.ok) throw new Error(`unexpected response ${response.s

await streamPipeline(response.body, createWriteStream('./octocat.pn
```

In Node.js 14 you can also use async iterators to read `body`; however, be careful to catch errors -- the longer a response runs, the more likely it is to encounter an error.

```
import fetch from 'node-fetch';

const response = await fetch('https://httpbin.org/stream/3');

try {
	for await (const chunk of response.body) {
		console.dir(JSON.parse(chunk.toString()));
	}
} catch (err) {
	console.error(err.stack);
}
```

In Node.js 12 you can also use async iterators to read `body`; however, async iterators with streams did not mature until Node.js 14, so you need to do some extra work to ensure you handle errors directly from the stream and wait on it response to fully close.

```
import fetch from 'node-fetch';

const read = async body => {
	let error;
	body.on('error', err => {
		error = err;
	});
```

```
        for await (const chunk of body) {
                console.dir(JSON.parse(chunk.toString()));
        }

        return new Promise((resolve, reject) => {
                body.on('close', () => {
                        error ? reject(error) : resolve();
                });
        });
};

try {
        const response = await fetch('https://httpbin.org/stream/3'
        await read(response.body);
} catch (err) {
        console.error(err.stack);
}
```

Accessing Headers and other Metadata

```
import fetch from 'node-fetch';

const response = await fetch('https://github.com/');

console.log(response.ok);
console.log(response.status);
console.log(response.statusText);
console.log(response.headers.raw());
console.log(response.headers.get('content-type'));
```

Extract Set-Cookie Header

Unlike browsers, you can access raw `Set-Cookie` headers manually using `Headers.raw()`. This is a `node-fetch` only API.

```
import fetch from 'node-fetch';

const response = await fetch('https://example.com');

// Returns an array of values, instead of a string of comma-separat
console.log(response.headers.raw()['set-cookie']);
```

Post data using a file

```
import fetch {
  Blob,
  blobFrom,
  blobFromSync,
  File,
  fileFrom,
  fileFromSync,
} from 'node-fetch'

const mimetype = 'text/plain'
const blob = fileFromSync('./input.txt', mimetype)
const url = 'https://httpbin.org/post'

const response = await fetch(url, { method: 'POST', body: blob })
const data = await response.json()

console.log(data)
```

node-fetch comes with a spec-compliant **FormData** implementations for posting multipart/form-data payloads

```
import fetch, { FormData, File, fileFrom } from 'node-fetch'

const httpbin = 'https://httpbin.org/post'
const formData = new FormData()
const binary = new Uint8Array([ 97, 98, 99 ])
```

```
const abc = new File([binary], 'abc.txt'), { type: 'text/plain' })

formData.set('greeting', 'Hello, world!')
formData.set('file-upload', abc, 'new name.txt')

const response = await fetch(httpbin, { method: 'POST', body: formD
const data = await response.json()

console.log(data)
```

If you for some reason need to post a stream coming from any arbitrary place, then you can append a **Blob** or a **File** look-a-like item.

The minium requirement is that it has:

1. A `Symbol.toStringTag` getter or property that is either `Blob` or `File`
2. A known size.
3. And either a `stream()` method or a `arrayBuffer()` method that returns a ArrayBuffer.

The `stream()` must return any async iterable object as long as it yields Uint8Array (or Buffer) so Node.Readable streams and whatwg streams works just fine.

```
formData.append('upload', {
        [Symbol.toStringTag]: 'Blob',
        size: 3,
  *stream() {
    yield new Uint8Array([97, 98, 99])
        },
        arrayBuffer() {
                return new Uint8Array([97, 98, 99]).buffer
        }
}, 'abc.txt')
```

Request cancellation with AbortSignal

You may cancel requests with `AbortController`. A suggested implementation is `abort-controller`.

An example of timing out a request after 150ms could be achieved as the following:

```
import fetch, { AbortError } from 'node-fetch';

// AbortController was added in node v14.17.0 globally
const AbortController = globalThis.AbortController || await import(

const controller = new AbortController();
const timeout = setTimeout(() => {
        controller.abort();
}, 150);

try {
        const response = await fetch('https://example.com', {signal
        const data = await response.json();
} catch (error) {
        if (error instanceof AbortError) {
                console.log('request was aborted');
        }
} finally {
        clearTimeout(timeout);
}
```

See **test cases** for more examples.

API

fetch(url[, options])

- `url` A string representing the URL for fetching
- `options` Options for the HTTP(S) request
- Returns: `Promise<Response>`

Perform an HTTP(S) fetch.

`url` should be an absolute URL, such as `https://example.com/`. A path-relative URL (`/file/under/root`) or protocol-relative URL (`//can-be-http-or-https.com/`) will result in a rejected `Promise`.

Options

The default values are shown after each option key.

```
{
        // These properties are part of the Fetch Standard
        method: 'GET',
        headers: {},            // Request headers. format is the i
        body: null,             // Request body. can be null, or a 
        redirect: 'follow',     // Set to `manual` to extract redir
        signal: null,           // Pass an instance of AbortSignal 

        // The following properties are node-fetch extensions
        follow: 20,             // maximum redirect count. 0 to not 
        compress: true,         // support gzip/deflate content enc
        size: 0,                // maximum response body size in by
        agent: null,            // http(s).Agent instance or functi
        highWaterMark: 16384,   // the maximum number of bytes to s
        insecureHTTPParser: false       // Use an insecure HTTP par
}
```

Default Headers

If no values are set, the following request headers will be sent automatically:

Header	Value
`Accept-Encoding`	`gzip,deflate,br` *(when `options.compress === true`)*
`Accept`	`*/*`

Header	Value
`Connection`	`close` *(when no `options.agent` is present)*
`Content-Length`	*(automatically calculated, if possible)*
`Host`	*(host and port information from the target URI)*
`Transfer-Encoding`	`chunked` *(when `req.body` is a stream)*
`User-Agent`	`node-fetch`

Note: when `body` is a `Stream`, `Content-Length` is not set automatically.

Custom Agent

The `agent` option allows you to specify networking related options which are out of the scope of Fetch, including and not limited to the following:

- Support self-signed certificate
- Use only IPv4 or IPv6
- Custom DNS Lookup

See `http.Agent` for more information.

In addition, the `agent` option accepts a function that returns `http`(s)`.Agent` instance given current **URL**, this is useful during a redirection chain across HTTP and HTTPS protocol.

```
import http from 'node:http';
import https from 'node:https';

const httpAgent = new http.Agent({
        keepAlive: true
});
const httpsAgent = new https.Agent({
        keepAlive: true
});
```

```
const options = {
        agent: function(_parsedURL) {
                if (_parsedURL.protocol == 'http:') {
                        return httpAgent;
                } else {
                        return httpsAgent;
                }
        }
};
```

Custom highWaterMark

Stream on Node.js have a smaller internal buffer size (16kB, aka `highWaterMark`) from client-side browsers (>1MB, not consistent across browsers). Because of that, when you are writing an isomorphic app and using `res.clone()`, it will hang with large response in Node.

The recommended way to fix this problem is to resolve cloned response in parallel:

```
import fetch from 'node-fetch';

const response = await fetch('https://example.com');
const r1 = await response.clone();

const results = await Promise.all([response.json(), r1.text()]);

console.log(results[0]);
console.log(results[1]);
```

If for some reason you don't like the solution above, since `3.x` you are able to modify the `highWaterMark` option:

```
import fetch from 'node-fetch';

const response = await fetch('https://example.com', {
```

```
        // About 1MB
        highWaterMark: 1024 * 1024
});

const result = await res.clone().arrayBuffer();
console.dir(result);
```

Insecure HTTP Parser

Passed through to the `insecureHTTPParser` option on http(s).request. See `http.request` for more information.

Manual Redirect

The `redirect: 'manual'` option for node-fetch is different from the browser & specification, which results in an **opaque-redirect filtered response**. node-fetch gives you the typical **basic filtered response** instead.

```
const fetch = require('node-fetch');

const response = await fetch('https://httpbin.org/status/301', { re

if (response.status === 301 || response.status === 302) {
        const locationURL = new URL(response.headers.get('location'
        const response2 = await fetch(locationURL, { redirect: 'man
        console.dir(response2);
}
```

Class: Request

An HTTP(S) request containing information about URL, method, headers, and the body. This class implements the **Body** interface.

Due to the nature of Node.js, the following properties are not implemented at this moment:

- `type`
- `destination`

- `mode`
- `credentials`
- `cache`
- `integrity`
- `keepalive`

The following node-fetch extension properties are provided:

- `follow`
- `compress`
- `counter`
- `agent`
- `highWaterMark`

See **options** for exact meaning of these extensions.

new Request(input[, options])

(spec-compliant)

- `input` A string representing a URL, or another `Request` (which will be cloned)
- `options` [Options][#fetch-options] for the HTTP(S) request

Constructs a new `Request` object. The constructor is identical to that in the **browser**.

In most cases, directly `fetch(url, options)` is simpler than creating a `Request` object.

Class: Response

An HTTP(S) response. This class implements the **Body** interface.

The following properties are not implemented in node-fetch at this moment:

- `trailer`

new Response([body[, options]])

(spec-compliant)

- `body` A `String` or `Readable` stream
- `options` A `ResponseInit` options dictionary

Constructs a new `Response` object. The constructor is identical to that in the **browser**.

Because Node.js does not implement service workers (for which this class was designed), one rarely has to construct a `Response` directly.

response.ok

(spec-compliant)

Convenience property representing if the request ended normally. Will evaluate to true if the response status was greater than or equal to 200 but smaller than 300.

response.redirected

(spec-compliant)

Convenience property representing if the request has been redirected at least once. Will evaluate to true if the internal redirect counter is greater than 0.

response.type

(deviation from spec)

Convenience property representing the response's type. node-fetch only supports `'default'` and `'error'` and does not make use of **filtered responses**.

Class: Headers

This class allows manipulating and iterating over a set of HTTP headers. All methods specified in the **Fetch Standard** are implemented.

new Headers([init])

(spec-compliant)

- `init` Optional argument to pre-fill the `Headers` object

Construct a new `Headers` object. `init` can be either `null`, a `Headers` object, an key-value map object or any iterable object.

```
// Example adapted from https://fetch.spec.whatwg.org/#example-head
import {Headers} from 'node-fetch';

const meta = {
        'Content-Type': 'text/xml'
};
```

```
const headers = new Headers(meta);

// The above is equivalent to
const meta = [['Content-Type', 'text/xml']];
const headers = new Headers(meta);

// You can in fact use any iterable objects, like a Map or even ano
const meta = new Map();
meta.set('Content-Type', 'text/xml');
const headers = new Headers(meta);
const copyOfHeaders = new Headers(headers);
```

Interface: Body

`Body` is an abstract interface with methods that are applicable to both `Request` and `Response` classes.

body.body

(deviation from spec)

- Node.js `Readable` stream

Data are encapsulated in the `Body` object. Note that while the **Fetch Standard** requires the property to always be a WHATWG `ReadableStream`, in node-fetch it is a Node.js `Readable` **stream**.

body.bodyUsed

(spec-compliant)

- `Boolean`

A boolean property for if this body has been consumed. Per the specs, a consumed body cannot be used again.

body.arrayBuffer()

body.formData()

body.blob()

body.json()

body.text()

`fetch` comes with methods to parse `multipart/form-data` payloads as well as `x-www-form-urlencoded` bodies using `.formData()` this comes from the idea that Service Worker can intercept such messages before it's sent to the server to alter them. This is useful for anybody building a server so you can use it to parse & consume payloads.

▶ Code example

Class: FetchError

(node-fetch extension)

An operational error in the fetching process. See **ERROR-HANDLING.md** for more info.

Class: AbortError

(node-fetch extension)

An Error thrown when the request is aborted in response to an `AbortSignal`'s `abort` event. It has a `name` property of `AbortError`. See **ERROR-HANDLING.MD** for more info.

TypeScript

Since `3.x` types are bundled with `node-fetch`, so you don't need to install any additional packages.

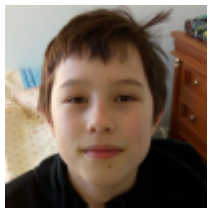
For older versions please use the type definitions from **DefinitelyTyped**:

```
npm install --save-dev @types/node-fetch@2.x
```

Acknowledgement

Thanks to **github/fetch** for providing a solid implementation reference.

Team

				
David Frank	Jimmy Wärting	Antoni Kepinski	Richie Bendall	Gregor Martynus

Former

- Timothy Gu
- Jared Kantrowitz

License

MIT

Keywords

fetch http promise request curl wget xhr whatwg

Install

```
> npm i node-fetch
```

Repository

github.com/node-fetch/node-fetch

Homepage

github.com/node-fetch/node-fetch

❤**Fund** this package

Weekly Downloads

37,647,352

37,647,352

Version	License
3.2.3	MIT
Unpacked Size	**Total Files**
106 kB	17
Issues	**Pull Requests**
115	15

Last publish

a month ago

Collaborators

>_Try on RunKit

Report malware

Support

Help

Advisories

Status

Contact npm

Company

About

Blog

Press

Terms & Policies

Policies

Terms of Use

Code of Conduct

Privacy