undici

Node CI passing | code style standard | npm package 5.10.0 | codecov 95%

A HTTP/1.1 client, written from scratch for Node.js.

> *Undici means eleven in Italian. 1.1 -> 11 -> Eleven -> Undici. It is also a Stranger Things reference.*

Have a question about using Undici? Open a Q&A Discussion or join our official OpenJS Slack channel.

Install

```
npm i undici
```

Benchmarks

The benchmark is a simple `hello world` example using a number of unix sockets (connections) with a pipelining depth of 10 running on Node 16. The benchmarks below have the simd feature enabled.

Connections 1

Tests	Samples	Result	Tolerance	Difference with slowest
http - no keepalive	15	4.63 req/sec	± 2.77 %	-
http - keepalive	10	4.81 req/sec	± 2.16 %	+ 3.94 %
undici - stream	25	62.22 req/sec	± 2.67 %	+ 1244.58 %
undici - dispatch	15	64.33 req/sec	± 2.47 %	+ 1290.24 %
undici - request	15	66.08 req/sec	± 2.48 %	+ 1327.88 %
undici - pipeline	10	66.13 req/sec	± 1.39 %	+ 1329.08 %

Connections 50

Tests	Samples	Result	Tolerance	Difference with slowest
http - no keepalive	50	3546.49 req/sec	± 2.90 %	-
http - keepalive	15	5692.67 req/sec	± 2.48 %	+ 60.52 %
undici - pipeline	25	8478.71 req/sec	± 2.62 %	+ 139.07 %
undici - request	20	9766.66 req/sec	± 2.79 %	+ 175.39 %
undici - stream	15	10109.74 req/sec	± 2.94 %	+ 185.06 %
undici - dispatch	25	10949.73 req/sec	± 2.54 %	+ 208.75 %

Quick Start

```
import { request } from 'undici'

const {
  statusCode,
  headers,
  trailers,
  body
} = await request('http://localhost:3000/foo')

console.log('response received', statusCode)
console.log('headers', headers)

for await (const data of body) {
  console.log('data', data)
}

console.log('trailers', trailers)
```

Using the body mixin from the Fetch Standard.

```
import { request } from 'undici'

const {
  statusCode,
  headers,
  trailers,
  body
} = await request('http://localhost:3000/foo')

console.log('response received', statusCode)
console.log('headers', headers)
console.log('data', await body.json())
console.log('trailers', trailers)
```

Common API Methods

This section documents our most commonly used API methods. Additional APIs are documented in their own files within the docs folder and are accessible via the navigation list on the left side of the docs site.

`undici.request([url, options]): Promise`

Arguments:

- **url** `string | URL | UrlObject`
- **options** `RequestOptions`
  - **dispatcher** `Dispatcher` - Default: getGlobalDispatcher
  - **method** `String` - Default: `PUT` if `options.body`, otherwise `GET`
  - **maxRedirections** `Integer` - Default: `0`

Returns a promise with the result of the `Dispatcher.request` method.

Calls `options.dispatcher.request(options)`.

See Dispatcher.request for more details.

`undici.stream([url, options, ]factory): Promise`

Arguments:

- **url** `string | URL | UrlObject`
- **options** `StreamOptions`
  - **dispatcher** `Dispatcher` - Default: getGlobalDispatcher
  - **method** `String` - Default: `PUT` if `options.body`, otherwise `GET`
  - **maxRedirections** `Integer` - Default: `0`
- **factory** `Dispatcher.stream.factory`

Returns a promise with the result of the `Dispatcher.stream` method.

Calls `options.dispatcher.stream(options, factory)`.

See Dispatcher.stream for more details.

`undici.pipeline([url, options, ]handler): Duplex`

Arguments:

- **url** `string | URL | UrlObject`
- **options** `PipelineOptions`
  - **dispatcher** `Dispatcher` - Default: getGlobalDispatcher
  - **method** `String` - Default: `PUT` if `options.body`, otherwise `GET`
  - **maxRedirections** `Integer` - Default: `0`
- **handler** `Dispatcher.pipeline.handler`

Returns: `stream.Duplex`

Calls `options.dispatch.pipeline(options, handler)`.

See Dispatcher.pipeline for more details.

`undici.connect([url, options]): Promise`

Starts two-way communications with the requested resource using HTTP CONNECT.

Arguments:

- **url** `string | URL | UrlObject`
- **options** `ConnectOptions`
  - **dispatcher** `Dispatcher` - Default: getGlobalDispatcher
  - **maxRedirections** `Integer` - Default: `0`
- **callback** `(err: Error | null, data: ConnectData | null) => void` (optional)

Returns a promise with the result of the `Dispatcher.connect` method.

Calls `options.dispatch.connect(options)`.

See Dispatcher.connect for more details.

`undici.fetch(input[, init]): Promise`

Implements fetch.

- https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/fetch
- https://fetch.spec.whatwg.org/#fetch-method

Only supported on Node 16.5+.

This is experimental and is not yet fully compliant with the Fetch Standard. We plan to ship breaking changes to this feature until it is out of experimental. Help us improve the test coverage by following instructions at nodejs/undici/#951.

Basic usage example:

```
    import {fetch} from 'undici';

    async function fetchJson() {
        const res = await fetch('https://example.com')
        const json = await res.json()
        console.log(json);
    }
```

`request.body`

A body can be of the following types:

- ArrayBuffer
- ArrayBufferView
- AsyncIterables
- Blob
- Iterables
- String
- URLSearchParams
- FormData

In this implementation of fetch, `request.body` now accepts `Async Iterables`. It is not present in the Fetch Standard.

```
import { fetch } from "undici";

const data = {
  async *[Symbol.asyncIterator]() {
    yield "hello";
    yield "world";
  },
};

(async () => {
  await fetch("https://example.com", { body: data, method: 'POST' });
})();
```

`response.body`

Nodejs has two kinds of streams: web streams which follow the API of the WHATWG web standard found in browsers, and an older Node-specific streams API. `response.body` returns a readable web stream. If you would prefer to work with a Node stream you can convert a web stream using `.fromWeb()`.

```
    import {fetch} from 'undici';
    import {Readable} from 'node:stream';

    async function fetchStream() {
        const response = await fetch('https://example.com')
        const readableWebStream = response.body;
        const readableNodeStream = Readable.fromWeb(readableWebStream);
    }
```

Specification Compliance

This section documents parts of the Fetch Standard which Undici does not support or does not fully implement.

Garbage Collection

- https://fetch.spec.whatwg.org/#garbage-collection

The Fetch Standard allows users to skip consuming the response body by relying on garbage collection to release connection resources. Undici does not do the same. Therefore, it is important to always either consume or cancel the response body.

Garbage collection in Node is less aggressive and deterministic (due to the lack of clear idle periods that browser have through the rendering refresh rate) which means that leaving the release of connection resources to the garbage collector can lead to excessive connection usage, reduced performance (due to less connection re-use), and even stalls or deadlocks when running out of connections.

```
// Do
const headers = await fetch(url)
  .then(async res => {
    for await (const chunk of res.body) {
      // force consumption of body
    }
    return res.headers
  })

// Do not
const headers = await fetch(url)
  .then(res => res.headers)
```

`undici.upgrade([url, options]): Promise`

Upgrade to a different protocol. See MDN - HTTP - Protocol upgrade mechanism for more details.

Arguments:

- **url** `string | URL | UrlObject`
- **options** `UpgradeOptions`

    - **dispatcher** `Dispatcher` - Default: getGlobalDispatcher
    - **maxRedirections** `Integer` - Default: `0`
- **callback** `(error: Error | null, data: UpgradeData) => void` (optional)

Returns a promise with the result of the `Dispatcher.upgrade` method.

Calls `options.dispatcher.upgrade(options)`.

See Dispatcher.upgrade for more details.

`undici.setGlobalDispatcher(dispatcher)`

- dispatcher `Dispatcher`

Sets the global dispatcher used by Common API Methods.

`undici.getGlobalDispatcher()`

Gets the global dispatcher used by Common API Methods.

Returns: `Dispatcher`

`UrlObject`

- **port** `string | number` (optional)
- **path** `string` (optional)
- **pathname** `string` (optional)
- **hostname** `string` (optional)
- **origin** `string` (optional)
- **protocol** `string` (optional)
- **search** `string` (optional)

Specification Compliance

This section documents parts of the HTTP/1.1 specification which Undici does not support or does not fully implement.

Expect

Undici does not support the `Expect` request header field. The request body is always immediately sent and the `100 Continue` response will be ignored.

Refs: https://tools.ietf.org/html/rfc7231#section-5.1.1

Pipelining

Undici will only use pipelining if configured with a `pipelining` factor greater than `1`.

Undici always assumes that connections are persistent and will immediately pipeline requests, without checking whether the connection is persistent. Hence, automatic fallback to HTTP/1.0 or HTTP/1.1 without pipelining is not supported.

Undici will immediately pipeline when retrying requests after a failed connection. However, Undici will not retry the first remaining requests in the prior pipeline and instead error the corresponding callback/promise/stream.

Undici will abort all running requests in the pipeline when any of them are aborted.

- Refs: https://tools.ietf.org/html/rfc2616#section-8.1.2.2
- Refs: https://tools.ietf.org/html/rfc7230#section-6.3.2

Manual Redirect

Since it is not possible to manually follow an HTTP redirect on server-side, Undici returns the actual response instead of an `opaqueredirect` filtered one when invoked with a `manual` redirect. This aligns `fetch()` with the other implementations in Deno and Cloudflare Workers.

Refs: https://fetch.spec.whatwg.org/#atomic-http-redirect-handling

Collaborators

- **Daniele Belardi**, https://www.npmjs.com/~dnlup
- **Ethan Arrowood**, https://www.npmjs.com/~ethan_arrowood
- **Matteo Collina**, https://www.npmjs.com/~matteo.collina
- **Robert Nagy**, https://www.npmjs.com/~ronag
- **Szymon Marczak**, https://www.npmjs.com/~szmarczak
- **Tomas Della Vedova**, https://www.npmjs.com/~delvedor

Releasers

- **Ethan Arrowood**, https://www.npmjs.com/~ethan_arrowood
- **Matteo Collina**, https://www.npmjs.com/~matteo.collina
- **Robert Nagy**, https://www.npmjs.com/~ronag

License

MIT