hviana /
faster

Code Issues Pull requests Actions Projects Security

识别到网页内正在下载...

来自当前网页的文件
https://github.com/hviana...

该文件可转存至百度网盘
转存地址：我的网盘/来自：浏览器扩展

不占用电脑空间 一键转存，秒速下载

暂不转存 转存至网盘

30天内不再提醒

A fast and optimized middleware server with an absurdly small amount of code (300 li
no dependencies. It also has a collection of useful middlewares: log file, serve static, C
parsers, redirect, proxy and handle upload. For Deno Deploy and other enviroments!

MIT license
53 stars **9** forks **3** watching Branches Tags Activity

Public repository

master **1** Branch **1** Tag Go to file t Go to file + Add file Code ···

hviana update readme d300d91 · 2 months ago

middlewares	fixes	2 months ago
tests	send	3 months ago
vendor	fix vendor exports kv fs	3 months ago
.gitignore	send	3 months ago
LICENSE	send	3 months ago
README.md	update readme	2 months ago
deno.json	fix server	2 months ago
deps.ts	temp	3 months ago
mod.ts	temp	3 months ago
server.ts	fix server	2 months ago

README License

🚀 Faster

> **Important**
>
> Please give a star! ⭐

🌟 Introduction

Faster is a **fast and optimized middleware server** with an incredibly small codebase (~300 lines), built on top of native HTTP APIs **with no dependencies**. It includes a collection of useful middlewares (Some are specific to Deno):

- 📄 Log file
- 🗂️ Serve static
- 🌐 CORS
- 🔐 Session
- ⏱️ Rate limit
- 🛡️ Token

- 📥 Body parsers
- 🔀 Redirect
- 🔌 Proxy
- 📤 Handle upload

Fully compatible with **Deno Deploy** and other environments. Examples of all resources ar
ideology is simple: all you need is an optimized middleware manager; all other functiona

📚 Contents

⚡ Benchmarks

The middleware is built on top of Deno's native HTTP APIs. See the benchmarks (for a 'Hello World' server):

Machine: 8 GiB RAM, Intel® Core™ i5-10210U CPU @ 2.11GHz × 4
Method: `autocannon -c 100 -d 40 -p 10 localhost:80`
Environment: Deno v1.46.3, Ubuntu 24.04 LTS

Framework	Version	Router?	Results
Express	4.19.2	✓	167k requests in 40.11s, **29 MB** read
Fastify	4.28.1	✓	1105k requests in 40.07s, **193 MB** read

Framework	Version	Router?	Results
Oak	17.0.0	✓	260k requests in 40.09s, **45 MB** read
Faster	**12.1**	**✓**	**1432k requests in 40.17s, 250 MB read**

> **Note:** In addition to its performance, Faster is a very complete framework considering

🚀 Example

Defining Routes

Static Routes: `/foo` , `/foo/bar`

Parameterized Routes:

Simple: `/:title` , `/books/:title` , `/books/:genre/:title`

With Suffix: `/movies/:title.mp4` , `/movies/:title.(mp4|mov)`

Optional Parameters: `/:title?` , `/books/:title?` , `/books/:genre/:title?`

Wildcards: `*` , `/books/*` , `/books/:genre/*`

POST: Read and Return JSON

```
import { req, res, Server } from "https://deno.land/x/faster/mod.ts";

const server = new Server();

server.post(
  "/example_json",
  res("json"),
  req("json"),
  async (ctx: any, next: any) => {
    console.log(ctx.body);
    ctx.res.body = { msg: "json response example" };
    await next();
  },
);

await server.listen({ port: 80 });

//or with the portable command "serve":
export default { fetch: server.fetch };
```

🌐 GET: Return HTML

```
server.get(
  "/example_html",
  res("html"),
  async (ctx: any, next: any) => {
    ctx.res.body = `
      <!DOCTYPE html>
      <html>
        <head>
          <meta charset="utf-8">
          <title>Title Example</title>
        </head>
        <body>
          HTML body example
        </body>
      </html>
    `;
    await next();
  },
);
```

🔍 Get URL Params

```
server.get(
  "/example_params/:ex1?foo=bar",
  async (ctx: any, next: any) => {
    console.log(ctx.params.ex1);
    console.log(ctx.url.searchParams.get("foo")); // Explore the URL (ctx.url) ob
    await next();
  },
);
```

🍪 Cookies

```
import {
  Cookie,
  deleteCookie,
  getCookies,
  getSetCookies,
  Server,
  setCookie,
} from "https://deno.land/x/faster/mod.ts"; // Alias to Deno std

server.get(
  "/cookies",
  async (ctx: any, next: any) => {
    setCookie(ctx.res.headers, { name: "user_name", value: "San" }); // Explore interface 'Cookie' for more options
    deleteCookie(ctx.res.headers, "last_order");
    console.log(getCookies(ctx.req.headers));
    await next();
  },
);
```

↩️ Redirect

Use: `ctx.redirect([status,] "/my_custom_url_or_path")`. The default status is `302`.

```
server.get(
  "/redirect_example",
  async (ctx: any, next: any) => {
    ctx.redirect(303, "/my_custom_url_or_path");
    await next();
  },
);

server.get(
  "/redirect_example2",
  async (ctx: any, next: any) => {
    ctx.redirect("/my_custom_url_or_path");
    await next();
  },
);
```

💬 WebSockets

By default, the server will reject WebSocket connections to prevent vulnerabilities. To accept connections, use the `acceptOrRejectSocketConn` function, which should return an ID to retrieve the WebSocket later. If the function returns `undefined`, `""`, `null`, `0`, etc., the connection will be rejected.

Example:

```
server.acceptOrRejectSocketConn = async (ctx: Context) => {
  // Returning undefined, "", null, or 0 will reject the connection.
  return ctx.req.headers.get("Host")!; // Return ID
};
```

Retrieving the Socket by ID:

```
server.openedSockets.get(yourId); // As in the example, ctx.req.headers.get("Host
```

Receiving WebSocket Events:

```
server.onSocketMessage = async (id: string, socket: WebSocket, event: any) => {
  console.log(id);
  console.log(socket);
  console.log(event);
};

server.onSocketClosed = async (id: string, socket: WebSocket) => {
  console.log(id);
  console.log(socket);
};
//... server.onSocketError, server.onSocketOpen
```

🛠️ Middlewares

This project has a standard set of middlewares useful for most cases.

📦 Set Deno KV and Deno KV FS

You need to launch Deno KV and Deno KV FS as several middlewares depend on it.

```
const kv = await Deno.openKv(); // Use your parameters here to launch a custom Deno.Kv
Server.setKv(kv);
```

Now, you can globally access instances in `Server.kv` and `Server.kvFs`.

> **Deno KV File System (`Server.kvFs`):** Compatible with Deno Deploy. Saves files in 64KB chunks. You can organize files into directories, control the KB/s rate for saving and reading files, impose rate limits, set user space limits, and limit concurrent operations—useful for controlling uploads/downloads. Utilizes the Web Streams API.

See more at: deno_kv_fs

📝 Logger

```
logger(save: boolean = true, print: boolean = true)
```

Initialize Deno KV (if not already done):

```
const kv = await Deno.openKv();
Server.setKv(kv);
```

Usage:

```
// You can also use useAtBeginning
server.use(logger()); // With default options: save and print are true
```

Access Log Data:

> **Retrieve Logs:** `await FasterLog.get(startMillis, endMillis)`
> **Delete Logs:** `await FasterLog.delete(startMillis, endMillis)`

Body Parsers (`res` and `req`)

Example:

```
server.post(
  "/example_parsers",
  res("json"), // Response parser
  req("json"), // Request parser
  async (ctx: any, next: any) => {
    console.log(ctx.body); // The original (unparsed) body is in ctx.req.body
    ctx.res.body = { msg: "json response example" };
    await next();
  },
);
```

Supported Options:

- `req` Parsers: `"arrayBuffer"`, `"blob"`, `"formData"`, `"json"`, `"text"`
- `res` Parsers: `"json"`, `"html"`, `"javascript"`

Custom Parsing Example:

```
server.post(
  "/custom_parse",
  async (ctx: any, next: any) => {
    ctx.res.headers.set("Content-Type", "application/json");
    const data = await customParseBody(ctx.req.body); // Handle ctx.req.body manually
    ctx.res.body = JSON.stringify({ msg: "ok" });
    await next();
  },
);
```

Rate Limit

Usage:

```
// You can also use useAtBeginning
server.use(rateLimit());
```

Options (with default values):

```
rateLimit({
  attempts: 30,
  interval: 10,
  maxTableSize: 100000,
  id: (ctx: Context) => ctx.req.headers.get("Host")!,
});
```

Serve Static

Example (route must end with `/*`):

```
server.get(
  "/pub/*",
  serveStatic("./pub"),
);
```

Set CORS

Example:

```
server.options("/example_cors", setCORS()); // Enable pre-flight request

server.get(
  "/example_cors",
  setCORS(),
  async (ctx, next) => {
    await next();
  },
);
```

Specify Allowed Hosts:

```
setCORS("http://my.custom.url:8080");
```

🔑 Token

This middleware is encapsulated in an entire static class. It uses Bearer Token and default options with the "HS256" algorithm, generating a random secret when starting the application (you can also set a secret manually).

Usage:

```
server.get(
  "/example_verify_token", // Send token to server in Header => Authorization: Bearer TOKEN
  Token.middleware,
  async (ctx, next) => {
    console.log(ctx.extra.tokenPayload);
    console.log(ctx.extra.token);
    await next();
  },
);
```

Generate Token:

```
await Token.generate({ user_id: "172746" }, null); // Null for never expire; defaults to "1h"
```

Set Secret:

```
Token.setSecret("a3d2r366wgb3dh6yrwzw99kzx2"); // Do this at the beginning of your application
```

Get Token Payload Outside Middleware:

```
await Token.getPayload("YOUR_TOKEN_STRING"); // For example, to get token data from token string in URL parameter
```

Set Configurations:

```
Token.setConfigs(/* your configurations */);
```

↩️ Redirect Middleware

Usage: `redirect([status,] "/my_custom_url_or_path")`. The default status is `302`.

Example:

```
server.get(
  "/my_url_1",
  redirect(303, "/my_url_2"), // Or the full URL
);

server.get(
  "/my_url_2",
  redirect("/my_url_3"), // Or the full URL
);
```

🔐 Session

Initialize Deno KV (if not already done):

```
const kv = await Deno.openKv();
Server.setKv(kv);
```

Example

```
// You can also use useAtBeginning
server.use(session());

// In routes:
server.get(
  "/session_example",
  async (ctx, next) => {
    console.log(ctx.extra.session); // Get session data
    ctx.extra.session.value.foo = "bar"; // Set session data (foo => "bar")
    await next();
  },
);
```

The default engine uses Deno KV and is optimized.

Expiration Policies

Absolute Expiration: The object in the cache will expire after a certain time from when it was inserted, regardless of its usage. A value of `0` disables this expiration.

Sliding Expiration: The object expires after a configured time from the last request (`get` or `set`). A value of `0` disables this expiration.

Note: If both `slidingExpiration` and `absoluteExpiration` are `0`, expiration is disabled. If both are greater than `0`, `absoluteExpiration` cannot be less than `slidingExpiration`.

Session Storage Engine Interface:

```
constructor(
  slidingExpiration: number = 0,
  absoluteExpiration: number = 0
)
```

Default Values:

```
session(engine: SessionStorageEngine = new KVStorageEngine()) // Default is 60 min slidingExpiration
```

🔌 Proxy

Usage:

```
// You can also use useAtBeginning
server.use(proxy({ url: "https://my-url-example.com" }));
server.use(proxy({ url: async (ctx) => "https://my-url-example.com" }));
```

In Routes:

```
server.get(
  "/proxy_example",
  async (ctx, next) => {
    console.log(ctx.req); // Request points to the proxy
    console.log(ctx.res); // Response contains the proxy answer
    await next();
  },
);
```

Specific Proxy Route:

```
server.get(
  "/proxy_example",
  proxy({
    url: "https://my-url-example.com/proxy_ex2",
    replaceProxyPath: false, // Specific proxy route for "/proxy_example"
  }),
  async (ctx, next) => {
    console.log(ctx.req);
    console.log(ctx.res);
    await next();
  },
);
```

Conditional Proxy:

```
server.get(
  "/proxy_example",
  proxy({
    url: "https://my-url-example.com/proxy_ex3",
    condition: (ctx) => {
      return ctx.url.searchParams.get("foo") ? true : false;
    },
  }),
  async (ctx, next) => {
    console.log(ctx.extra.proxied); // True if proxy condition is true
    console.log(ctx.req);
    console.log(ctx.res);
    await next();
  },
);
```

Options (with default values):

```
proxy({
  url: string,
  replaceReqAndRes: true,
  replaceProxyPath: true,
  condition: (ctx: Context) => true,
});
```

> **Warning:** Do not use "res body parsers" with `replaceReqAndRes: true` (default)!
> **Note:** If you don't use Request body information before the proxy or in your condition, avoid using "req body parsers" to reduce processing cost.

📤 Upload

Initialize Deno KV (if not already done):

```
const kv = await Deno.openKv();
Server.setKv(kv);
```

This middleware uses Deno KV File System (deno_kv_fs).

🚀 Upload Usage

Example:

```
// The route must end with *
server.post("/files/*", upload(), async (ctx: any, next: any) => {/* ... */});
server.get("/files/*", download(), async (ctx: any, next: any) => {/* ... */});
```

With Custom Options:

Download:

```
server.post(
  "/files/*",
  upload({
    allowedExtensions: async (ctx: Context) => ["jpg"],
    maxSizeBytes: async (ctx: Context) =>
      (ctx.extra.user.isPremium() ? 1 : 0.1) * 1024 * 1024 * 1024, // 1GB or 100M
    maxFileSizeBytes: async (ctx: Context) =>
      (ctx.extra.user.isPremium() ? 1 : 0.1) * 1024 * 1024 * 1024, // 1GB or 100MB
    chunksPerSecond: async (ctx: Context) =>
      (ctx.extra.user.isPremium() ? 10 : 1) /
      kvFs.getClientReqs(ctx.extra.user.id),
    maxClientIdConcurrentReqs: async (
      ctx: Context,
    ) => (ctx.extra.user.isPremium() ? 10 : 1),
    clientId: async (ctx: Context) => ctx.extra.user.id,
    validateAccess: async (ctx: Context, path: string[]) =>
      ctx.extra.user.hasDirAccess(path),
  }),
  async (ctx: any, next: any) => {/* ... */},
);
```

Upload:

```
server.get(
  "/files/*",
  download({
    chunksPerSecond: async (ctx: Context) =>
      (ctx.extra.user.isPremium() ? 10 : 1) /
      kvFs.getClientReqs(ctx.extra.user.id),
    maxClientIdConcurrentReqs: async (
      ctx: Context,
    ) => (ctx.extra.user.isPremium() ? 10 : 1),
    clientId: async (ctx: Context) => ctx.extra.user.id,
    validateAccess: async (ctx: Context, path: string[]) =>
      ctx.extra.user.hasDirAccess(path),
    maxDirEntriesPerSecond: async (
      ctx: Context,
    ) => (ctx.extra.user.isPremium() ? 1000 : 100),
    pagination: async (ctx: Context) => true,
    cursor: async (ctx: Context) => ctx.url.searchParams.get("cursor"),
  }),
);
```

💻 Upload Examples in Frontend and Backend

Frontend (AJAX with multiple files):

```
const files = document.querySelector("#yourFormId input[type=file]").files;
const name = document.querySelector("#yourFormId input[type=file]")
  .getAttribute("name");

const form = new FormData();
for (let i = 0; i < files.length; i++) {
  form.append(`${name}_${i}`, files[i]);
}
const userId = 1; // Example
const res = await fetch(`/files/${userId}`, {
  method: "POST",
  body: form,
}).then((response) => response.json());
```

```
console.log(res);
```

Backend (Deno):

```
import {
  download,
  res,
  Server,
  upload,
} from "https://deno.land/x/faster/mod.ts";

const server = new Server();

server.post(
  "/files/*", // For example: /files/general/myFile.xlsx
  res("json"),
  upload(), // Using default options. No controls.
  async (ctx: any, next: any) => {
    ctx.res.body = ctx.extra.uploadedFiles;
    await next();
  },
);

server.get(
  "/files/*",
  download(), // Using default options. No controls.
);

server.get("/", res("html"), async (ctx: any, next: any) => {
  ctx.res.body = `
    <form id="yourFormId" enctype="multipart/form-data" action="/upload" method="post">
      <input type="file" name="file1" multiple><br>
      <input type="submit" value="Submit">
    </form>
  `;
  await next();
});

await server.listen({ port: 80 });

//or with the portable command "serve":
export default { fetch: server.fetch };
```

📁 Organizing Routes in Files

It's possible to organize routes into files using native JavaScript resources.

Main File:

```
import { Server } from "https://deno.land/x/faster/mod.ts";
import exampleRoutes from "./example_routes.ts";

const server = new Server();
exampleRoutes("example", server);

await server.listen({ port: 80 });

//or with the portable command "serve":
export default { fetch: server.fetch };
```

Secondary Route File (`example_routes.ts`):

```
import { req, res, Server } from "https://deno.land/x/faster/mod.ts";

export default function exampleRoutes(namespace: string, server: Server) {
  server.post(
    `${namespace}/json`,
    res("json"),
    req("json"),
    async (ctx: any, next: any) => {
      console.log(ctx.body);
      ctx.res.body = { msg: "json response example" };
      await next();
    },
  );

  server.get(
    `${namespace}/html`,
    res("html"),
    async (ctx: any, next: any) => {
      ctx.res.body = `
        <!DOCTYPE html>
        <html>
          <head>
            <meta charset="utf-8">
            <title>Title Example</title>
          </head>
          <body>
            HTML body example
          </body>
        </html>
      `;
      await next();
    },
  );
}
```

📦 All Imports

```
import {
  Context,
  ContextResponse, // Type
  Cookie, // Type, alias to Deno std
  deleteCookie, // Alias to Deno std
  download,
  FasterLog,
  getCookies, // Alias to Deno std
  getSetCookies, // Alias to Deno std
  KVStorageEngine,
  logger,
  NextFunc, // Type
  Params, // Type
  parse,
  ProcessorFunc, // Type
  proxy,
  rateLimit,
  redirect,
  req,
  res,
  Route, // Type
  RouteFn, // Type
  Server,
  serveStatic,
  Session, // Type
  session,
  SessionStorageEngine,
  setCookie, // Alias to Deno std
  setCORS,
  Token,
  upload,
} from "jsr:@hviana/faster";
import * as jose from "jsr:@hviana/faster/jose"; // jsr port of deno panva/jose (v5.9.6)
import * as deno_kv_fs from "jsr:@hviana/faster/deno-kv-fs"; // Alias to jsr @hviana/deno-kv-fs (v1.0.1)
```

🌐 Example Deploy in Ubuntu

Example of deploying an application named "my-deno-app" in a Ubuntu environment. C
yours.

🛠️ Create Service

Create Run Script ("run-server.sh") in Your Application Folder:

```
#!/bin/bash
/home/ubuntu/.deno/bin/deno run --allow-all --unstable-kv /home/ubuntu/my-deno-ap
```

Give Execution Permission to the Script:

```
chmod +x run-server.sh
```

Create Service Files:

```
sudo touch /etc/systemd/system/my-deno-app.service
sudo nano /etc/systemd/system/my-deno-app.service
```

In "my-deno-app.service" (change "Description", "WorkingDirectory", and "ExecStart" to yours):

```
[Unit]
Description=My Deno App

[Service]
WorkingDirectory=/home/ubuntu/my-deno-app
ExecStart=/home/ubuntu/my-deno-app/run-server.sh
TimeoutSec=30
Restart=always
RestartSec=1

[Install]
WantedBy=multi-user.target
```

If Your Application Depends on Another Service (e.g., MongoDB):

```
[Unit]
Description=My Deno App
After=mongod.service
```

Enable the "my-deno-app" Service:

```
sudo systemctl enable my-deno-app.service
```

Start and Stop the "my-deno-app" Service:

```
sudo service my-deno-app stop
sudo service my-deno-app start
```

View Logs:

```
journalctl -u my-deno-app.service --since=today -e
```

🔒 Configure HTTPS

Install Certbot:

```
sudo apt install certbot
```

Generate Certificates (Port 80 Must Be Free):

```
sudo certbot certonly --standalone
```

During Setup:

When prompted:

```
Please enter the domain name(s) you would like on your certificate (comma and/or
cancel):
```

Enter your domains and subdomains, e.g.: `yourdomain.link www.yourdomain.link`

Run Your Application on HTTPS (Change "yourdomain.link" to Your Domain):

```
await server.listen({
  port: 443,
  cert: await Deno.readTextFile(
    "/etc/letsencrypt/live/yourdomain.link/fullchain.pem",
  ),
  key: await Deno.readTextFile(
    "/etc/letsencrypt/live/yourdomain.link/privkey.pem",
  ),
});

//or with the portable command "serve":
//in this case you need to pass arguments such as port and certificate in the command.
export default { fetch: server.fetch };
```

Set Up Automatic Certificate Renewal:

The certificate is valid for a short period. Set up a cron job to renew automatically.

Edit Root's Crontab:

```
sudo crontab -e
```

Add to the End of the File (to Check and Renew Every 12 Hours):

```
0 */12 * * * certbot -q renew --standalone --preferred-challenges=http
```

Alternatively, Check Every 7 Days:

```
0 0 * * 0 certbot -q renew --standalone --preferred-challenges=http
```

💡 See Also: Faster with React

Check out the complete framework with Faster and React:

👉 https://github.com/hviana/faster_react

🧑‍💻 About

Releases 1

v12.1 Latest
on Nov 8, 2024

识别到网页内正在下载...

来自当前网页的文件
https://github.com/hviana...

该文件可转存至百度网盘

转存地址：我的网盘/来自：浏览器扩展

不占用电脑空间 一键转存，秒速下载

30天内不再提醒

Packages

No packages published

Contributors 2

hviana Henrique Emanoel Viana

exside luk

Languages

● **TypeScript** 100.0%