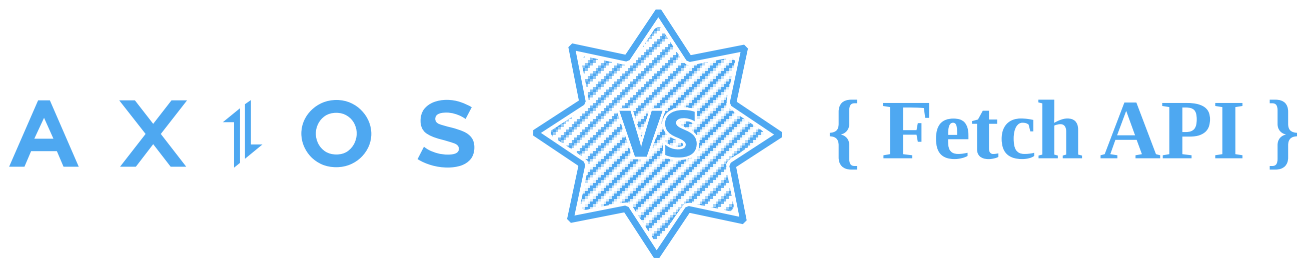
Axios vs Fetch: Which HTTP Client to Choose in JS?

by Mostafa Gouda Oct 29, 2024 #HTTP #NodeJS #API



Making HTTP requests is a fundamental feature in any programming language, and JavaScript is no exception. As the development community has moved away from using `XMLHttpRequest`, making HTTP requests in JavaScript typically involves either the built-in Fetch API or the axios package.

In this article, we'll explore Fetch vs Axios — compare their key differences, pros, cons, and help you determine which option is best for your project.

What is Fetch?

Fetch is a built-in browser API that allows you to make HTTP requests from the client side, providing a more modern and flexible alternative to the older `XMLHttpRequest`. It is known for its **promise-based** syntax, making it easy to handle asynchronous operations. While Fetch was initially designed for the browser environment, it has also been officially supported in **Node.js v18**. This means developers can now use Fetch in both the browser and server-side environments, making it a versatile tool for modern JavaScript development.

> ℹ️ Fetch is also supported by other JS runtimes like Bun and Deno as well as serverless runtimes like Cloudflare Workers and Edge Runtime

Example of a basic Fetch request:

.then() async/await

```
fetch('https://httpbin.dev/json')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

The Fetch API's integration in Node.js allows developers to work with the same API on both client and server sides, simplifying the codebase and making Fetch a powerful tool across the entire stack that is now considered a default option for handling HTTP requests in many JavaScript applications.

What is Axios?

Axios is a popular third-party HTTP client library that simplifies making HTTP requests in JavaScript. Built on top of JavaScript's `XMLHttpRequest` interface in the browser and Node.js's `http` module on the server, Axios provides a cleaner and more intuitive syntax for making requests and handling responses.

Example of a basic Axios request:

.then() async/await

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

axios.get('https://api.example.com/data')
  .then(response => console.log(response.data))
  .catch(error => console.error('Error:', error));
```

Fetch vs Axios - How are They Different?

Both Fetch and Axios are capable of handling HTTP requests, but they differ in several key areas.

Availability

Fetch is a built-in API available in the browser, Node.js, and most JS runtimes and serverless environments making it one less dependency for your project.In contrast, Axios is a third-party library, meaning it requires installation via a package manager or inclusion through a CDN.

While both axios and fetch are isomorphic, meaning that they can be used in both server and client-side environments, each supports a different yet overlapping set of browsers, node.js versions, and JS runtimes.

Feature	Axios	Fetch
Client-side Availability	Available via npm/CDN	Built into modern browsers
Browser Compatibility	Fully Compatible (including IE11)	All Modern browsers (No IE11 support)
Polyfill Requirements	Requires ES6 promises polyfill for non ES6 environments	Requires polyfill for IE11 and older browsers
Server Runtime Support		
Node.js	✅	- Native support since Node.js 18 (May 2022) - Stable since Node.js 21 (Nov 2023) - Earlier versions require `node-fetch`
Deno	✅	✅
Bun	✅	✅
Serverless Support		
AWS Lambda	✅	✅
Azure Functions	✅	✅
Google Cloud Functions	✅	✅
Cloudflare Workers	🚫	✅
Netlify Functions	✅	✅
Netlify Edge Functions	✅	✅
Vercel Serverless Funcitons (Node.js)	✅	✅
Vercel Edge Functions (Edge Runtime)	🚫	✅
Deno Deploy	✅	✅

Features

While fetch Provides basic functionality for making HTTP requests, it lacks many built-in advanced features that axios supports by default. Let's explore some of those features.

1. Interceptors

Interceptors in Axios allow you to modify requests and responses before they are processed. They are particularly useful for tasks like adding authentication tokens, logging requests, or handling global error responses.

Using interceptors you can centralize logic that applies to all requests or responses, enhancing maintainability and flexibility in your application.

```
// 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) {
    // Any status code that lie within the range of 2xx cause this function to trigger
    // Do something with response data
    return response;
  }, function (error) {
    // Any status codes that falls outside the range of 2xx cause this function to
trigger
    // Do something with response error
    return Promise.reject(error);
  });
```

2. Timeouts

Timeouts in Axios provide a way to control how long the client should wait for a server response before aborting the request. This is essential when dealing with unreliable networks or APIs that may take too long to respond.

With Axios, you can easily configure request timeouts to ensure your app doesn't hang indefinitely waiting for a response.

```
// Setting a timeout of 5 seconds (5000 milliseconds)
axios.get('https://httpbin.dev/json', {
  timeout: 5000
})
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    if (error.code === 'ECONNABORTED') {
      console.error('Request timed out');
    } else {
      console.error('Error:', error.message);
    }
  });
```

3. Query parameters serialization

Query parameters allow you to pass data in the URL of a GET request. They are typically used to pass additional configuration options for an endpoint, such as filters and pagination.

Axios makes it simple to serialize query parameters into the proper format, ensuring they are correctly appended to the URL. It also supports serlializing nested objects without reaching out to a third party library like qs.

```
const params = {
  page: 2,
  order: 'desc',
  category: 'apparel'
};

axios.get('https://web-scraping.dev/api/products', { params })
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error:', error.message);
  });
```

4. Automatic request body serialization

Request body serialization depends on the type of content sent in the body. Whether it's JSON, URL-encoded data, or multipart form data, each type has its very own serialization technique.

Axios automatically converts the request body to the appropriate format based on the content type, eliminating the hassle of manually handling it using built-in language features like `JSON.stringify` or third-party libraries.

```
const data = {
  name: 'John Doe',
  age: 30,
  occupation: 'Developer'
};

// Default (application/json)
axios.post('https://httpbin.dev/post', data)
  .then(response => console.log(response.data))
  .catch(error => console.error('Error:', error.message));

// 'application/x-www-form-urlencoded'
axios.post('https://httpbin.dev/post', data, {
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded'
  }
}).then(response => console.log(response.data));

//'multipart/form-data'
const formData = {
  ...data,
  profile_picture: document.querySelector('#fileInput').files[0]
}
axios.post('https://httpbin.dev/post', formData, {
  headers: {
    'Content-Type': 'multipart/form-data'
  }
}).then(response => console.log(response.data));
```

5. Automatic Response Parsing

Automatic response parsing refers to Axios's ability to automatically parse the server's response into the appropriate data format.

If the response is in a valid JSON object, Axios will parse it into a JavaScript object without requiring you to manually call .json(), making it easier to handle responses with minimal effort.

```
axios.get('https://httpbin.dev/json')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error:', error.message);
  });

// compared to fetch
fetch('https://httpbin.dev/json')
  .then(response => response.json()) // a bit more verbose compared to axios
  .then(data => {
    console.log(data)
  })
  .catch(error => {
    console.error('Error:', error.message);
  });
```

6. Progress capturing

Tracking the progress of data uploads and downloads in real-time is a very common task encountered by developers. Unfortunately, fetch doesn't support progress tracking and implementing a solution around will lead you into a bunch of browser compatibility issues.

Using `onUploadProgress` and `onDownloadProgress` callbacks in Axios, you can easily monitor how much data has been transferred to and from the server, which is especially useful when dealing with large files or slow network conditions.

```
axios.post('https://httpbin.dev/post', formData, {
  headers: {
    'Content-Type': 'multipart/form-data'
  },
  onUploadProgress: (progressEvent) => {
    const percentCompleted = Math.round((progressEvent.loaded * 100) /
progressEvent.total);
    console.log(`Upload Progress: ${percentCompleted}%`);
  }
}).then(response => console.log(response.data));
```

```
axios.get('https://api.example.com/content.txt'
  onDownloadProgress: (progressEvent) => {
    const percentCompleted = Math.round((progressEvent.loaded * 100) /
progressEvent.total);
    console.log(`Download Progress: ${percentCompleted}%`);
  }
}).then(response => console.log(response.data));
```

Although axios has built-in support for all those features, they can all be easily implemented in fetch with some boilerplate code.

Developer Experience

When comparing Fetch and Axios from a user experience perspective, **Axios offers a more streamlined and feature-rich approach** providing:

- automatic JSON parsing
- built-in error handling
- and more features like interceptors and request cancellation.

This reduces boilerplate code and makes it easier to handle complex HTTP requests.

In contrast, **Fetch is more minimalistic but more readily available** and requires manual handling for tasks like transforming data and checking for non-2xx status codes, leading to more verbose code.

While Fetch offers flexibility and is natively supported in browsers, Axios enhances productivity by offering a cleaner, more user-friendly API, particularly for complex requests.

Axios's ability to handle global configuration, error management, and response transformations makes it a more developer-friendly choice for larger applications, while Fetch might be more suited for simpler, straightforward tasks.

Performance

In terms of performance, there is generally **no significant difference** between Fetch and Axios for most typical HTTP requests. Both tools handle HTTP requests efficiently, and any performance differences are usually negligible in real-world applications.

Key Points to Consider:

- **Fetch** is a native browser API, which means it's lightweight and directly integrated into the browser's environment. This can make Fetch slightly faster in some cases due to the lack of external dependencies, especially for simple requests.
- **Axios**, being a third-party library, introduces a tiny amount of overhead because it has additional features built on top of the basic request-handling mechanisms. However, this overhead is minimal and usually doesn't impact performance significantly, especially when the added convenience and functionality of Axios are considered.

For basic use cases, Fetch might have a slight performance advantage due to its lightweight nature, but for more complex tasks, Axios provides a smoother experience with minimal performance impact.

In most cases, the performance *difference is so small* that the choice between Fetch and Axios should be based on functionality and developer experience rather than speed alone.

Axios Alternatives

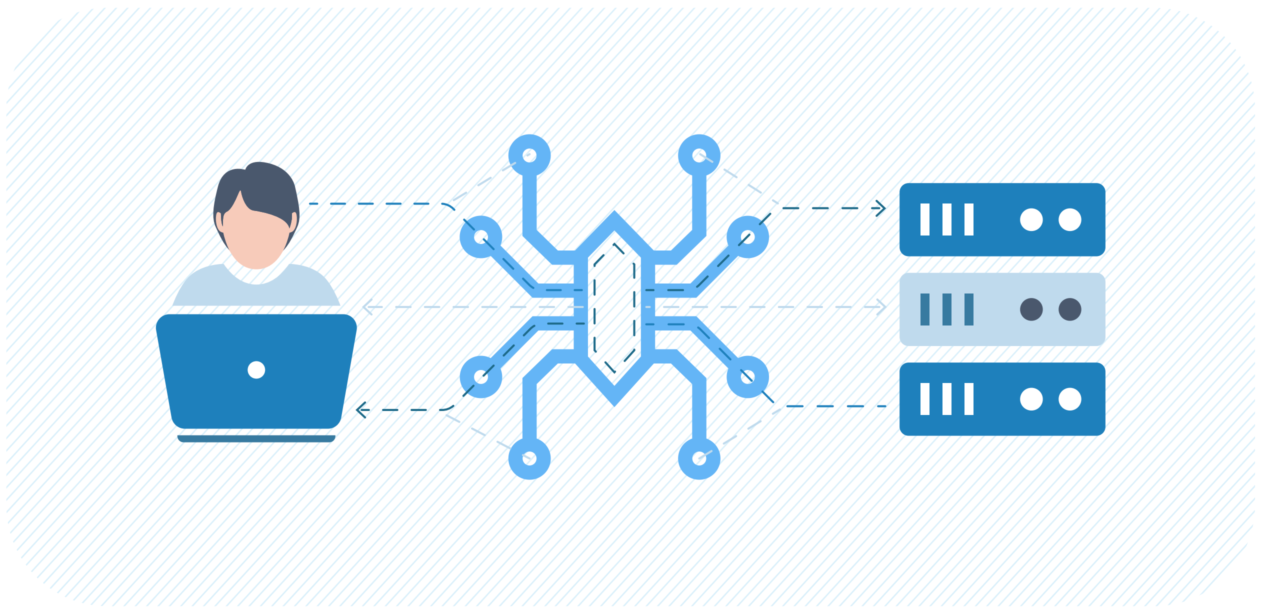
While **Axios** remains one of the most popular HTTP libraries, several modern alternatives provide similar functionality, often with added features or optimizations. Some modern alternatives include:

- **Superagent**: A flexible and powerful HTTP request library that provides a simple API for making requests in both Node.js and browsers. It supports promises, streams, and file uploads.
- **Got**: A lightweight and feature-rich HTTP request library for Node.js. It's known for its powerful error handling, high-performance, and promise-based API. It also offers advanced features like retry mechanisms, hooks, and custom agents.
- **Ky**: By the maker of **Got**. A smaller, modern wrapper around Fetch designed for browsers. It's promise-based, tiny in size, and offers features like automatic JSON parsing, retry logic, and simpler syntax than vanilla Fetch.

Each of these alternatives provides different features, and the best choice depends on your project's specific requirements.

Power Up with Scrapfly

While both Fetch and Axios are excellent HTTP clients they are not ideal for tasks like web automation and web scraping.



ScrapFly provides web scraping, screenshot, and extraction APIs for data collection at scale.

- Anti-bot protection bypass - scrape web pages without blocking!
- Rotating residential proxies - prevent IP address and geographic blocks.
- JavaScript rendering - scrape dynamic web pages through cloud browsers.
- Full browser automation - control browsers to scroll, input and click on objects.
- Format conversion - scrape as HTML, JSON, Text, or Markdown.
- Python and Typescript SDKs, as well as Scrapy and no-code tool integrations.

FAQ

Before we wrap this article up let's take a look at some frequently asked questions regarding fetch and axios comparisson that we haven't covered in this article:

Is axios better than fetch?

Whether Axios is better than Fetch depends on your needs. Axios offers a more feature-rich, user-friendly experience making it a powerful choice for handling complex HTTP requests. However, Fetch is a native browser API, which makes it lightweight and ideal for simple requests. It’s flexible, built into modern browsers, and requires no additional dependencies, making it more suitable for smaller projects or when simplicity is preferred.

What is the difference between node-fetch and fetch?

Before Fetch was natively supported in Node.js as of version 18, `node-fetch` was required for Fetch-like functionality on the server. `node-fetch` is a lightweight module that brings the Fetch API to Node.js environments, allowing server-side JavaScript to make HTTP requests in a similar way.

Can Fetch and Axios be used in web scraping?

Just like any http client, Fetch and Axios can both be used to request webpages or REST APIs for the purpose of scraping data. Check out our dedicated article on web scarping with Node.js and Javascript.

Summary

In summary, both **Fetch** and **Axios** are powerful tools for making HTTP requests in JavaScript, each with unique advantages:

Fetch

- Native to browsers, Node.js (from v18 onwards), and mainstream JS runtimes and serverless environments
- Lightweight and ideal for simpler requests with minimal configuration.
- Requires more manual handling for JSON parsing, error handling, and advanced features.

Axios

- Third-party library with more features for handling complex requests.
- Provides automatic JSON parsing, built-in error handling, request cancellation, and interceptors.
- More user-friendly and efficient for large-scale applications needing advanced configurations.

Ultimately, the choice between Fetch and Axios should be based on the specific needs of your project, balancing simplicity with functionality.

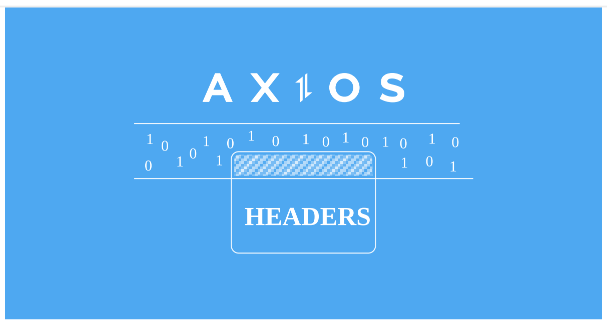
Related Questions

How to Copy as cURL With Safari?
How to Copy as cURL With Firefox?
How To Copy as cURL With Google Chrome?
What case should HTTP headers be in? Lowercase or Pascal-Case?
Python httpx vs requests vs aiohttp - key differences
How to use proxies with PHP Guzzle?
How to use proxies with NodeJS axios?
What is Asynchronous Web Scraping?
How to Copy as cURL With Brave?
How to Copy as cURL With Edge?
What Python libraries support HTTP2?
What are some PhantomJS alternatives for automating browsers?
How to install mitmproxy certificate on Chrome and Chromium?
How to use proxies with Python httpx?
How to add headers to every or some scrapy requests?

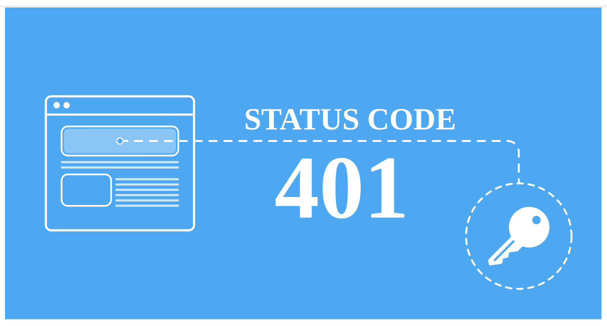
More >

HTTP NODEJS API

Related Posts



Dec 26, 2024



Dec 20, 2024

Guide to Axios Headers

Learn about Javascript's Axios headers. How to configure, update, inspect headers in request and responses, how to set defaults and useful tips

HTTP NODEJS

What is HTTP 401 Error and How to Fix it

Discover the HTTP 401 error meaning, its causes, and solutions in this comprehensive guide. Learn how 401 unauthorized errors occur.

HTTP

Dec 05, 2024

Comprehensive Guide to OkHttp for Java and Kotlin

Learn how to simplify network communication in Java and Android applications using OkHttp.

HTTP TOOLS

Company

Careers
Terms of service
Privacy Policy
Data Processing Agreement
KYC Compliance
Status

Integrations

Zapier
Make
N8n
LlamaIndex
LangChain

Social

Tools

Convert cURL commands to Python code
JA3/TLS Fingerprint
HTTP2 Fingerprint
Xpath/CSS Selector Tester

Resources

API Documentation
Web Scraping Academy
Is Web Scraping Legal?

Web Scraping Tools
FAQ

Learn Web Scraping

Web Scraping with Python
Web Scraping with PHP
Web Scraping with Ruby
Web Scraping with R
Web Scraping with NodeJS
Web Scraping with Python Scrapy
How to Scrape without getting blocked tutorial
Web Scraping with Python and BeautifulSoup
Web Scraping with Nodejs and Puppeteer
How To Scrape Graphql
Best Proxies for Web Scraping
Top 5 Best Residential Proxies

Usage

What is Web Scraping used for?
Web Scraping for AI Training
Web Scraping for Compliance
Web Scraping for eCommerce
Web Scraping for Finance
Web Scraping for Fraud Detection
Web Scraping for Jobs
Web Scraping for Lead Generation
Web Scraping for News & Media
Web Scraping for Real Estate
Web Scraping for SERP & SEO
Web Scraping for Social Media
Web Scraping for Travel