Introduction to Node.js

Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project!

Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser. This allows Node.js to be very performant.

A Node.js app runs in a single process, without creating a new thread for every request. Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in Node.js are written using non-blocking paradigms, making blocking behavior the exception rather than the norm.

When Node.js performs an I/O operation, like reading from the network, accessing a database or the filesystem, instead of blocking the thread and wasting CPU cycles waiting, Node.js will resume the operations when the response comes back.

This allows Node.js to handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency, which could be a significant source of bugs.

Node.js has a unique advantage because millions of frontend developers that write JavaScript for the browser are now able to write the server-side code in addition to the client-side code without the need to learn a completely different language.

In Node.js the new ECMAScript standards can be used without problems, as you don't have to wait for all your users to update their browsers - you are in charge of deciding which ECMAScript version to use by changing the Node.js version, and you can also enable specific experimental features by running Node.js with flags.

An Example Node.js Application

The most common example Hello World of Node.js is a web server:

CJS MJS

```
const { createServer } = require('node:htt

const hostname = '127.0.0.1';
const port = 3000;

const server = createServer((req, res) =>
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plai
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${
});
```

JavaScript Copy to clipboard

To run this snippet, save it as a **`server.js`** file and run **`node server.js`** in your terminal. If you use mjs version of the code, you should save it as a **`server.mjs`** file and run **`node server.mjs`** in your terminal.

This code first includes the Node.js `http` module.

Node.js has a fantastic standard library, including first-class support for networking.

The **`createServer()`** method of **`http`** creates a new HTTP server and returns it.

The server is set to listen on the specified port and host name. When the server is ready, the callback function is called, in this case informing us that the server is running.

Whenever a new request is received, the `request` event is called, providing two objects: a request (an `http.IncomingMessage` object) and a response (an `http.ServerResponse` object).

Those 2 objects are essential to handle the HTTP call.

The first provides the request details. In this simple example, this is not used, but you could access the request headers and request data.

The second is used to return data to the caller.

In this case with:

```javascript
res.statusCode = 200;
```

JavaScript Copy to clipboard

we set the `statusCode` property to `200`, to indicate a successful response.

We set the `Content-Type` header:

```javascript
res.setHeader('Content-Type', 'text/plain'
```

JavaScript Copy to clipboard

and we close the response, adding the content as an argument to `end()`:

```javascript
res.end('Hello World\n');
```

JavaScript Copy to clipboard

If you haven't already done so, download Node.js.

Next >
How much JavaScript do you need to know to use Node.js?

Getting Started > Introduction to Node.js

Reading Time

3 min

Authors

+10

Contribute

Edit this page

Table of Contents

Trademark Policy Privacy Policy Version Support Code of Conduct

Security Policy