Introduction to Node.js

Getting started guide to Node.js, the server-side JavaScript runtime environment. Node.js is built on top of the Google Chrome V8 JavaScript engine, and it's mainly used to create web servers - but it's not limited to just that.

Allows developers to run JavaScript code on the server side. Node.js enables the execution of JavaScript code outside of a web browser, allowing developers to use JavaScript for server-side scripting.

ARTICLE AUTHORS

- easy to learn
- single lang for frontend & backend
- fullstack JS
- offers high performance
- support of large & active community

▶ TABLE OF CONTENTS

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:

index.js

```
1  const http = require('http')
2
3  const hostname = '127.0.0.1'
4  const port = 3000
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200
8    res.setHeader('Content-Type', 'text/plain')
9    res.end('Hello World\n')
10 })
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostnam
     :${port}/`)
14 })
```

Incompatible Web Browser

WebContainers currently work in Chromium-based browsers, Firefox, and Safari 16.4. We're hoping to add support for more browsers as they implement the necessary Web Platform features.

Read more about browser support

Fork on StackBlitz Editor Preview Both

js copy

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

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

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

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

this includes Node.js http module

create new HTTP server & returns

indicate successful response

server is set to listen on the specified port & hostname & when server is ready, callback func is called.

To run this snippet, save it as a `server.js` file and run `node server.js` 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:

```
js
res.statusCode = 200;
```

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

We set the Content-Type header:

```
js
res.setHeader('Content-Type', 'text/plain');
```

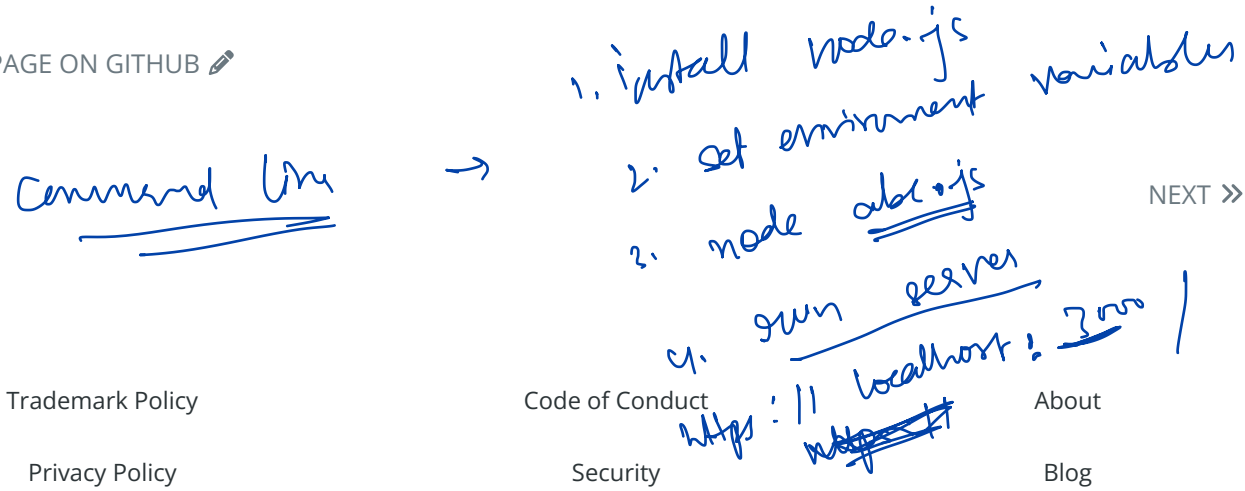
and we close the response, adding the content as an argument to `end()`:

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

More Examples

See https://github.com/nodejs/examples for a list of Node.js examples that go beyond hello world.

EDIT THIS PAGE ON GITHUB



NEXT »