Node.js (24.1.0) Docs

About This Documentation :-

- Node.js is a javascript runtime built on the V8 Javascript engine

CHATGPT :-

Node.js :-

- open source, cross-platform runtime environment that lets you run javascript code outside the browser.
- used to build server-side applications like web servers, APIs and command line tools.
- Before Node.js, javascript was mainly used in browsers (like chrome / web page) to control webpage behavior.

Ex :- If u want to run a JS file in the browser :-

- create script.js and write alert("Hello from the JS file");
- create HTML file and link the JS file

```
<!DOCTYPE html>
<html>
  <head>
    <title> Run JS File in Browser </title>
  </head>
  <body>
    <h1> Hello HTML Page </h1>
    <script src="script.js"></script>
  </body>
</html>
```

- Save index.html and script.js in the same folder
- double click index.html or right click → "Open with" → choose browser like chrome.
- You will see HTML page, a pop up will appear
  ✓ Hello from the JS file

Summary for Javascript:

- Your script.js file is on your local system. yes.
- But needs to be loaded through a HTML file and opened in a browser (like chrome)
- So technically, the browser is the one executing the JS.

Think of the browser as the "host" that runs Javascript using its built-in engine (like Chrome V8)

With Node.js :-

- Your script.js file is still on your local system.
- But now, you can run it directly using the terminal with `node script.js`
- No browser / HTML file needed.
- Node.js uses the same V8 engine as chrome, but it's outside the browser.

Summary:-

Feature	In Browser	With Node.js (Terminal)
Needs HTML file?	✓	✗
Runs in browser	✓	✗
Runs in terminal	✗	✓
Can access files?	✗ (for security)	✓ (fs module)
Engine used	V8 (inside browser)	V8 (inside Node.js)

So why is this useful?

Because now, Javascript is not just for webpages — you can:

- create web servers
- Read and write files
- Build tools.
- Run apps on backend.

All using Javascript with Node.js

Q:- Can access files? ✗ No (for security) ✓ Yes (fs module)

Ans:- Javascript in the browser — ✗ Can't access files

Why not?

Because JS in the browser is running on a website and that website should not be able to access your personal files (like documents, passwords...) This is called browser sandbox. ⇒ A sandbox means — limited access to protect your system.

```
const fs = require("fs"); // ✗ Will throw 'fs is not defined'
```

The browser will throw an error like:

Uncaught ReferenceError: require is not defined.

Reason: The browser blocks this for your safety.

Javascript in Node.js — Can access files using the fs module.

Node.js is a runtime environment meant for server-side or local use, not webpages. So it access full access to your file system using the fs (File System) module.

Why Javascript in Node.js can access files — Node.js is meant to run on your computer, not on random websites.

So it gives you special built-in modules like:

- fs → File System (for reading/writing files)
- path → To handle file paths
- os → To get OS-level information

What is V8 Javascript Engine?

V8 is the engine that reads and runs Javascript code. It was built by Google and used in the chrome and Node.js

Analogy:-

If JS is a script, V8 is the actor that performs it.

It takes Javascript and converts it into machine code so that computer can understand and run it.

What is runtime?

A runtime is like the environment or platform that helps your code actually run — it gives:

- The engine to understand the language (like JS)
- Useful tools / libraries to do things (like file handling, network)
- A way to execute and manage code.

Analogy:- A kitchen

Think of:

- Your code = the receipe
- Runtime = the kitchen
- Javascript engine (V8) = the chef
- Libraries (like fs, http) = kitchen tools.

Without a kitchen, the chef has no place to cook. Without a runtime, Javascript has no environment to run in.

Let's compare 2 Javascript Runtimes

Runtime	Where it runs	What it gives you
Browser	Chrome, Firefox	DOM access, alert, document
Node.js	Your system/terminal	fs, http, path... (server-side)

A runtime environment - interpreter + extra tools + system access
runtime environment is the actual thing that executes code, often with extra tools like (APIs, modules, memory management)

runtime environment — can execute code
— understand language (like Js)

Non-runtime environment — can't execute code
system/tool that might help u write, edit or organize code.
But can't run the code by itself.

Ex:- You can write Js code in it.
But if u click Run in Vs code. It's not Vs code itself running the code. It's using Node.js / browser behind the scenes to execute it.
So Vs code is just an editor, not a runtime.

Hello-world.js

```
const http = require ('node: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}/`);
});
```

Terminal: node Hello-world.js

Reason:- node:http and http both work the same way
↳ more explicit way of writing http
This module allows you to create HTTP servers and handle HTTP requests / response.

http.createServer ⟷ creates HTTP server.
200 → statusCode → OK → Success
res.setHeader ('Content-Type', 'text/plain'); → Tell the browser that the content being returned is plain text.

Assertion Testing

The node:assert module provides a set of assertion functions for verifying invariants.

- node:assert
  - built-in module for Node.js
  - use it to perform assertions — tests that verify whether some condition in your code is true.
  - commonly used in test and debugging.

```
const assert = require('node:assert');
```

- provides a set of assertion functions :-
  - the module includes multiple functions like
    - assert.strictEqual(a,b) — check if a===b
    - assert.ok(value) — check if value is truthy
    - assert.deepStrictEqual(obj1, obj2) — checks deep equality
  - These functions throw an error if assertion fails.

- "for verifying invariants"
  - An invariant is a condition that is expected to always be true during the execution of a program.
  - for example, if your code expects an array to always have at least 1 element, that's an invariant.

Assertion help verify these invariants while the code is running, especially during development or testing.

Example:

```
const assert = require('node:assert');

function divide(a, b) {
    assert.notStrictEqual(b, 0, 'Denominator should not be 0');
    return a/b;
}

divide(4, 2); // works
divide(4, 0); // Throws: AssertionError: Denominator should not be 0
```

Here, the assertion verifies an 'invariant': b should never be 0 while dividing.

Node.js has 2 modes of assertions:

- legacy mode [default if you use `require('assert')`;]
- strict mode [enabled if u use `require('node:assert/strict')`]

In strict mode, non-strict assertion methods act like strict versions

Legacy mode (non-strict)

```
const assert = require('assert'); // legacy mode
// Loose comparision
assert.deepEqual({a: 1}, {a: '1'}); // ✓ Passes (1 == '1')
```

Strict mode

~~const assert = require('assert');~~

const assert = require('assert/strict'); // strict mode

// Strict comparision

assert.deepEqual({a:1}, {a:'1'}); // X ~~Fails (1 !== '1')~~

Fails (1 !== '1')

In strict mode, if assertion fails, you'll get a clear diff showing what's diff bw the actual and expected values.

AssertionError [ERR_ASSERTION]: Expected values to be strictly deep-equal:

+ actual — expected

```
{
- "a": 1
+ "a": "1"
}
```

Legacy mode Error (more vague/truncated):

AssertionError [ERR_ASSERTION]: {a: 1} deepEqual {a: '1'}

To deactivate the colors, use the NO_COLOR or NODE_DISABLE_COLORS environment variables. This will also deactivate colors in the REPL.

Note: When node.js throws errors (like from assert) or runs in a terminal, it adds colors to the output — for example, making errors red, diff green/red...

But sometimes, you might want to turn off those colors — ex:- ① when you're saving logs to a file ② when your terminal doesn't support colors ③ for accessibility/automation reasons

on windows :-

set NO_COLOR=1 && node app.js

set NODE_DISABLE_COLORS=1 && node app.js

deactivate the colors in the REPL