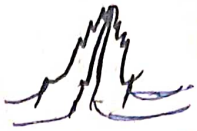
Namaste Node Js



" Any application that can be written in JS will eventually be written in JS."

Node Js is a cross platform, Open Source JavaScript runtime environment that can run JS anywhere.

or

Run Javascript Everywhere

It is a javascript runtime built on Chrome's V8 JS engine

→ Node.js has an event-driven architecture, capable of asynchronous I/O. / Non Blocking I/O

History

→ 2009 → Ryan Dahl → Joyent

→ Spider Monkey (Firefox) 2 day ∝ { (JS engine)

→ V8 (google chrome)

Web.js → created for web servers

↓

Node.js

as it is non blocking server while → Apache HTTP server → Blocking

→ 2010 → NPM → package manager for Node.js

→ 2011 → Windows support (Joyent + MS)

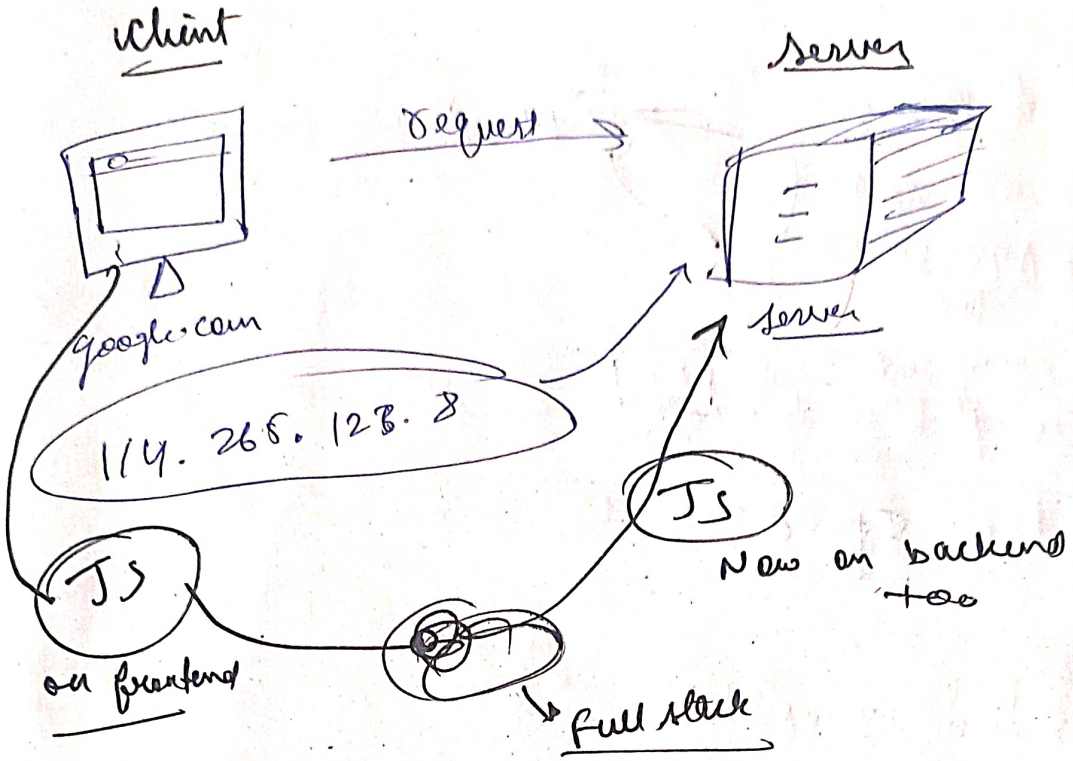
→ 2012 → Ryan left the project. Issac lead then

→ 2014 → Fedor fork Node.js & create IO.js

→ 2015 → IO.js & Node.js merged → Node Js foundation

→ 2019 → Node Js foundation + JS foundation → OpenJS foundation

Node JS → JS on server



Note: Node JS is C++ code

JS engine - V8 (Google) is written in C++

V8 is a Google open source high performance Javascript & Web Assembly engine, written in C++. It is used in chrome, & in Nodejs, among others

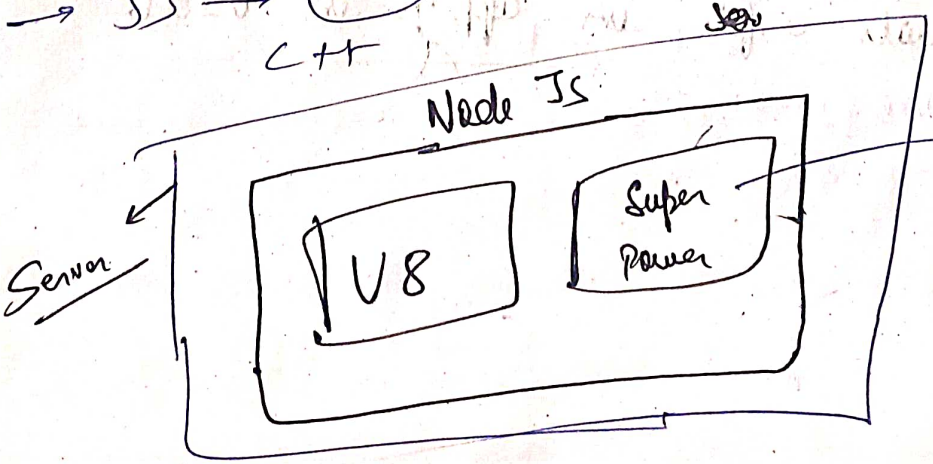
→ It implements Ecmascript & Web Assembly & runs on Windows, macOS, Linux, etc.

→ V8 can be embedded into any c++ application

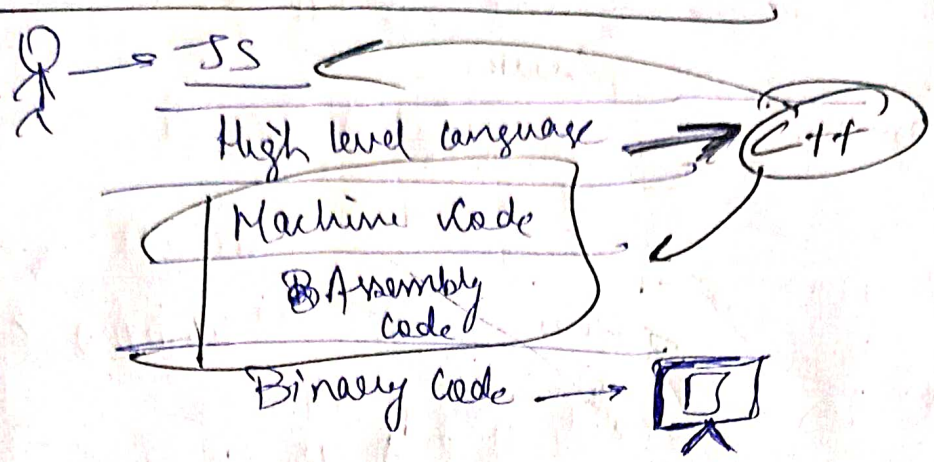
→ V8 → execute JS code

→ JS → V8 → Machine understandable

C++



V8 is a C++ code ??



Ep 03

node -v
→ gives the version of nodejs

npm -v
→ gives the version of npm.

Node REPL

REPL → Read, Evaluate, Print, Loop

for Node Repl, write command.

→ node

Welcome to Node.js v22.6.0
Now, we can do any program that we ~~able~~ able to do in js.
But in ~~root~~ console, Browser use V8 engine.

Now create a file ~~in~~ app.js in vscode

Write code, then write

→ node app.js

output

Note: We know that 'window' is the global object in our console window. which is given by Browser, not by v8 engine.

If we type 'this',
then we got window as global object

But in node.js, we have "GLOBAL" as the global object

i.e

console.log(global)

→ we will got <ref *1> Object [global] {

Note: .global is not a part of v8 engine so, it is not used in console.

But global is given by Node.js | Extra super powers that Node.js have

- Set Timeout
- set Interval

So, v8 engine only follows Ecmascript. It don't understand global. It only understand when global is given by node.js

console.log(global) ≠ console.log(this)
↓ ↓
global object empty object

Even in console

> window,
> this,
> self,
> frames

All gives window, i.e. global object

So, different runtime have different name for global,
↓
So, Open JS foundation decided to use "globalThis" = universal

So, ∀ globalThis is supported by every browser/runtime
i.e Node.js, Deno, Samsung Internet, Chrome, safari, edge.

So, on console,

→ globalThis
→ window

So, in Node js terminal

globalThis → global;

→ To check,

console.log(globalThis === global) → true

Q. If we have different .js files, we want to link them or on running one file, we are also able to run second. ? How.

Let app.js & src.js are two files which are seperate having no relation. so, we can say app.js and src.js are two different modules.

Q. So, How do you make two module work together?

→ the require function → which tells we require another thing.

i.e. require("./src.js");

then on run, both will work

→ Suppose, we have some f(n) & some variable in src.js. also there is some console.log("Hola src")

Now, we req. src.js in app.js. & Now, we want to use the function & variable of src.js. do we able to get it?

NO, as when we ~~run~~ require src.js . src.js runs first, all the consol.s msg printed. But f(x) & variable we are not able to fetch

Why?

Because by default, f(x) & variables are protected in modules.

(Or)

Modules protect their variables and functions from ~~leaking~~ leaking.

So, how can we access? ~~to use private stuff we have to export~~

We would export this f(x) from this file and import this f(x) into another file.

So we have to explicitly export this. How?

→ module.exports = function/var name

Now, we have to ~~import~~ import this where we want to ~~use~~ require

So, in app.js file

const ~~fun~~ func name/var name = require('./src.js');

Now if we want to export multiple stuffs. Then we export it as object. ie

module.exports = {
x : x,
calculator : calculate
}

in app.js const obj = require('./src.js');

obj.x (x)

obj.calculator →

Note:- Normally, during import, we use to destructure it. then we don't need to use obj everywhere.

i.e. const { x, calculator } = require("./sec.js");

✓ then we dont need, obj.x or obj.calculator();

we can write x, or calculator(a,b);

Note:- We can even skip .js extension during import.

~~Common JS Modules~~ (cjs)

Common Js modules (cjs)

- module.exports
- → require();
- → by default used in Node.js
- → Older way
- → Synchronous
- → non strict mode

ES Modules (esm) (mjs)

first we have to change the default configuration of type from commonjs to "module";

export function name() {

}

import { } from "./sec.js";

- → by default used in React, Angular,
- → newer way.
- → Async option available
- → (strict mode.)

(8) What is ~~module.js~~ module.exports ?

It is empty object {}.

So, during export, we can write

```
module.exports.x = x
module.exports.calculate = calculate
```

Note: We can create a folder as module.

→ What we have to do is, create a index.js file, where we have to import all ~~the~~ ~~all~~ the ~~five~~ things that we wanna export from this folder.

At last export all this things from index.js.

Now, we only import this folder, ~~of~~ not necessarily folder/index.js

to use this as module.

like, const {—, —, — ...} = require("./folder");

Check Day 1 code.