

Node Js

## What is Node.js?

* Node.js is an open source server environment
* Node.js is free
* Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
* Node.js uses JavaScript on the server
* 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.
* A Node.js app is run 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 needs to perform 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.

Node.js uses asynchronous programming!

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

=>

<=

=>=>=>

<=

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

## What Can Node.js Do?

* Node.js can generate dynamic page content
* Node.js can create, open, read, write, delete, and close files on the server
* Node.js can collect form data
* Node.js can add, delete, modify data in your database

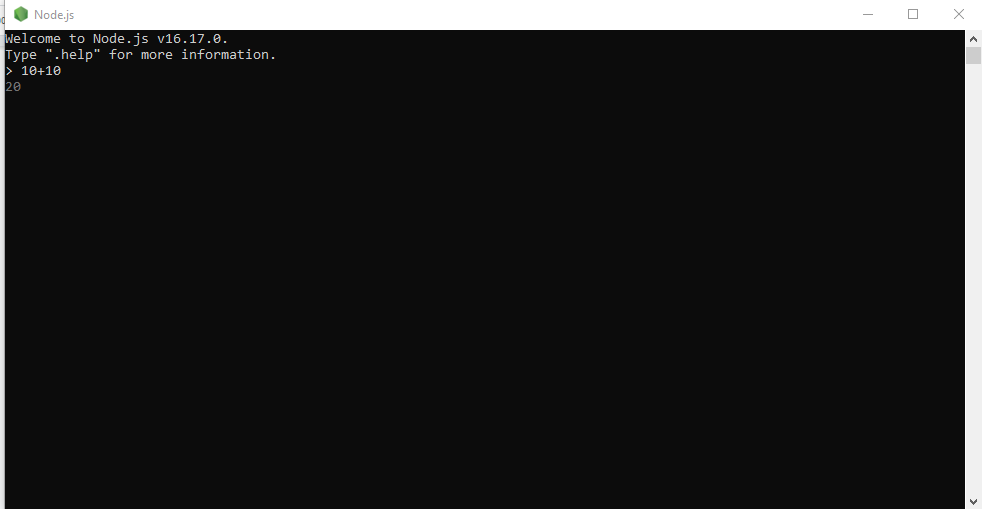
## What is a Node.js File?

* Node.js files contain tasks that will be executed on certain events
* A typical event is someone trying to access a port on the server
* Node.js files must be initiated on the server before having any effect
* Node.js files have extension ".js"

## Download Node.js

The official Node.js website has installation instructions for Node.js: [https://nodejs.org](https://nodejs.org/)

Run : start => node.js



Opermn one cmd node

Create a Node.js file named "myfirst.js", and add the following code:

Myfirst.js

Code : console.log(‘Hello World’);

Run : node Myfirst

Output : Hello World’

var http = require('http');

http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/html'});

res.end('Hello World!');

}).listen(8080);

Run in cmd : C:\Users\*Your Name*>node myfirst.js

Run : localhost:8080

# Node.js Modules

## What is a Module in Node.js?

Consider modules to be the same as JavaScript libraries.

A set of functions you want to include in your application.

## Built-in Modules

Node.js has a set of built-in modules which you can use without any further installation.

Node.js has a set of built-in modules which you can use without any further installation.

Here is a list of the built-in modules of Node.js version 6.10.3:

| **Module** | **Description** |
| --- | --- |
| [assert](https://www.w3schools.com/nodejs/ref_assert.asp) | Provides a set of assertion tests |
| [buffer](https://www.w3schools.com/nodejs/ref_buffer.asp) | To handle binary data |
| child\_process | To run a child process |
| [cluster](https://www.w3schools.com/nodejs/ref_cluster.asp) | To split a single Node process into multiple processes |
| [crypto](https://www.w3schools.com/nodejs/ref_crypto.asp) | To handle OpenSSL cryptographic functions |
| [dgram](https://www.w3schools.com/nodejs/ref_dgram.asp) | Provides implementation of UDP datagram sockets |
| [dns](https://www.w3schools.com/nodejs/ref_dns.asp) | To do DNS lookups and name resolution functions |
| domain | Deprecated. To handle unhandled errors |
| [events](https://www.w3schools.com/nodejs/ref_events.asp) | To handle events |
| [fs](https://www.w3schools.com/nodejs/ref_fs.asp) | To handle the file system |
| [http](https://www.w3schools.com/nodejs/ref_http.asp) | To make Node.js act as an HTTP server |
| [https](https://www.w3schools.com/nodejs/ref_https.asp) | To make Node.js act as an HTTPS server. |
| [net](https://www.w3schools.com/nodejs/ref_net.asp) | To create servers and clients |
| [os](https://www.w3schools.com/nodejs/ref_os.asp) | Provides information about the operation system |
| [path](https://www.w3schools.com/nodejs/ref_path.asp) | To handle file paths |
| punycode | Deprecated. A character encoding scheme |
| [querystring](https://www.w3schools.com/nodejs/ref_querystring.asp) | To handle URL query strings |
| [readline](https://www.w3schools.com/nodejs/ref_readline.asp) | To handle readable streams one line at the time |
| [stream](https://www.w3schools.com/nodejs/ref_stream.asp) | To handle streaming data |
| [string\_decoder](https://www.w3schools.com/nodejs/ref_string_decoder.asp) | To decode buffer objects into strings |
| [timers](https://www.w3schools.com/nodejs/ref_timers.asp) | To execute a function after a given number of milliseconds |
| [tls](https://www.w3schools.com/nodejs/ref_tls.asp) | To implement TLS and SSL protocols |
| tty | Provides classes used by a text terminal |
| [url](https://www.w3schools.com/nodejs/ref_url.asp) | To parse URL strings |
| [util](https://www.w3schools.com/nodejs/ref_util.asp) | To access utility functions |
| v8 | To access information about V8 (the JavaScript engine) |
| [vm](https://www.w3schools.com/nodejs/ref_vm.asp) | To compile JavaScript code in a virtual machine |
| [zlib](https://www.w3schools.com/nodejs/ref_zlib.asp) | To compress or decompress files |

## Include Modules

To include a module, use the require() function with the name of the module:

var http = require('http');

Now your application has access to the HTTP module, and is able to create a server:

http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/html'});

res.end('Hello World!');

}).listen(8080);

## Create Your Own/custome/local Modules

You can create your own modules, and easily include them in your applications.

The following example creates a module that returns a date and time object:

### Example

### Create a module that returns the current date and time:

### exports.myDateTime = function () {

return Date();

};

THEN YOU CAN ACCESS ANYWHERE IN MODULE

Notice that we use ./ to locate the module, that means that the module is located in the same folder as the Node.js file.

var http = require('http');

var dt = require('./myfirstmodule');

http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/html'});

res.write("The date and time are currently: " + dt.myDateTime());

res.end();

}).listen(8080);

## 

## NPM Modules or Third Party Module

There are many modules available online which could be used in Node.js.

Node Package Manager (NPM) helps to install those modules, extend them if

necessary and publish them to repositories like Github for access to

distributed machines

To start working with npm module

1. we have to initialize npm in our application

npm init

2. we have to install all module we want to use

npm install validator@versi

const validator = require('validator')

const getNotes = require('./notes.js')

const msg = getNotes()

console.log(msg)

console.log(validator.isURL('https://www.tops-int.com/'))

console.log(validator.isEmail('abcgmail.com'))

=======================================================================

## Node.js as a Web Server

The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

Use the createServer() method to create an HTTP server:

The function passed into the http.createServer() method, will be executed when someone tries to access the computer on port 8080.

var http = require('http');

//create a server object:

http.createServer(function (req, res) {

res.write('Hello World!'); //write a response to the client

res.end(); //end the response

}).listen(8080); //the server object listens on port 8080

## Add an HTTP Header

If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:

var http = require('http');

http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/html'});

res.write('Hello World!');

res.end();

}).listen(8080);

The first argument of the res.writeHead() method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

## Read the Query String

The function passed into the http.createServer() has a req argument that represents the request from the client, as an object (http.IncomingMessage object).

This object has a property called "url" which holds the part of the url that comes after the domain name:

var http = require('http');

http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/html'});

res.write(req.url);

res.end();

}).listen(8080);

If you have followed the same steps on your computer, you should see two different results when opening these two addresses:

<http://localhost:8080/summer>

Will produce this result:

/winter

**Global npm module & nodemon**

- Search for nodemon in npm it helps to run code automatically as we save code

npm i nodemon@version –g

- Here –g for Global install in OS

nodemon app.js

# Node.js - REPL Terminal

The Node.js interactive shell works on the principle of REPL, which is an acronym for READ, EVALUATE, PRINT and LOOP.

The Node.js interactive REPL terminal is like the Powershell or Command prompt terminal, or a bash terminal in Linux. It performs the following tasks −

**Read** − Reads user's input, parses the input into JavaScript data-structure, and stores in memory.

**Eval** − Takes and evaluates the data structure.

**Print** − Prints the result.

**Loop** − The terminal is ready to receive next input from the user.

To start the Node.js REPL on your computer, simply enter node in the command terminal (without the javascript file name as done before). The Node.js prompt > will appear.

> 10+20

30

> "Hello"+"World"

'HelloWorld'

> a=10

10

> b=20

20

> a+b

30

> Math.random()

0.5423940959293392

>

## **Multiline Expression**

> x=0

0

> do {

... x++;

... console.log("x: "+x);

... }

... while (x<5);

x: 1

x: 2

x: 3

x: 4

x: 5

undefined

>

## **Dot commands**

The REPL has some special commands, all starting with a dot .. They are

|  |  |
| --- | --- |
| **Sr.No** | **Dot Commands & Description** |
| 1 | **.help**  shows the dot commands help |
| 2 | **.editor**  enables editor mode, to write multiline JavaScript code with ease. Once you are in this mode, enter ctrl-D to run the code you wrote. |
| 3 | **.break**  when inputting a multi-line expression, entering the .break command will abort further input. Same as pressing ctrl-C. |
| 4 | **.clear**  resets the REPL context to an empty object and clears any multi-line expression currently being input. |
| 5 | **.load**  loads a JavaScript file, relative to the current working directory |
| 6 | **.save**  saves all you entered in the REPL session to a file (specify the filename) |
| 7 | **.exit**  exits the repl (same as pressing ctrl-C two times) |
| 8 | **Up/Down Keys**  see command history and modify previous commands. |
| 9 | **tab Keys**  list of current commands. |

# Node.js - Command Line Options

By Node

PS D:\nodejs> node hello.js

By Nodemon

Npm i -g nodemon

PS D:\nodejs> nodemon hello.js

======================================================================================

PS D:\nodejs> node --help

Version

PS D:\nodejs> node -v

v20.9.0

PS D:\nodejs> node --version

v20.9.0

Evaluate Script

PS D:\nodejs> node --eval "console.log(123)"

123

PS D:\nodejs> node -e "console.log(123)"

123

## **Load module**

PS D:\nodejs> node -r "http"

PS D:\nodejs> node –require "http"

NPM

Node.js - NPM

NPM − an acronym for Node Package Manager, refers to the Command line utility to install

Node.js packages, perform version management and dependency management of Node.js packages.

npm -v

npm install npm -g

======================================

Install Package Locally

npm install <Module Name>

======================================

Create a new folder for a new Node.js project, and run pnm init command inside it −

PS D:\nodejs\newnodeapp> npm init

package name: (newnodeapp) newnodeapp

version: (1.0.0)

description: Test Node.js App

entry point: (index.js)

test command:

git repository:

keywords: test, nodejs

author: mvl

license: (ISC)

Then yout package.json file ready to userAgent

=============================================================

npm i chalk --save

-S: --save

-D: --save-dev installs and adds the entry to the package.json file devDependencies

-O: --save-optional installs and adds the entry to the package.json file optionalDependencies

================================================

Install Package Globally

npm install express -g

================================================

Update Package

npm update <package name>

================================================

Uninstall Packages

npm uninstall <package name>

Command line arguments

We can take input from user using command line arguments

process.argv() is Global object like console

- so we can pass value from command line like what operation we want to perform

node app.js add

/ Node.js program to demonstrate the use of process.argv

// Importing the process module

const process = require('process');

// Printing property value for process.argv

console.log(process.argv);

Nodemon app.js add

Arguments Parsing with Yargs

Npm I yargs

Now in above example

node app.js add --title=”” –body=””

This is Title

This will print same text that we pass through commands

We need to parse the value for that we can use

yargs npm install yargs@version

As shown in code snippet for adding note we can also write code for removing note, listing notes, and reading note similarly

## **Node.js as a File Server**

The Node.js file system module allows you to work with the file system on your computer.

To include the File System module, use the require() method:

var fs = require('fs');

Common use for the File System module:

* Read files **readFile**
* Create files
* Update files
* Delete files
* Rename files

Read files

**Read In Console**

var fs = require('fs');

fs.readFile('TextFile.txt','utf8', function (err, data)

{

if (err) throw err;

console.log(data);

});

**Read HTML file on Server**

Create a Node.js file that reads the HTML file, and return the content:

var http = require('http');

var fs = require('fs');

http.createServer(function (req, res) {

fs.readFile('demofile1.html','utf8', function (err, data)

{

res.writeHead(200, {'Content-Type': 'text/html'});

res.write(data);

return res.end();

});

}).listen(8080);

node demo\_readfile.js

## **Create Files**

The File System module has methods for creating new files:

* **fs.writeFile()**
* **fs.appendFile()**
* **fs.open()**

Use fs.writeFile() method to write data to a file.

If file already exists then it overwrites the existing content otherwise it creates a new file and writes data into it.

fs.writeFile(filename, data[, options], callback)

The following example creates a new file called test.txt and writes "Hello World" into it asynchronously.

1) **fs.writeFile(filename, data[, options], callback)**

The following example creates a new file called test.txt and writes "Hello World" into it asynchronously.

var fs = require('fs');

fs.writeFile('test.txt', 'Hello World!', function (err) {

if (err)

console.log(err);

else

console.log(' create and Write operation complete.');

});

2) In the same way, use fs.appendFile() method to append the content to an existing file.

Example**: Append File Content**

var fs = require('fs');

fs.appendFile('testappend.txt', 'Hello World!', function (err) {

if (err)

console.log(err);

else

console.log(' create and Write/append operation complete.');

});

// Create a new, **empty file using the open()** method:

var fs = require('fs');

fs.open('TestFile.txt', 'w', function (err, file) {

if (err) throw err;

console.log('Saved!');

});

**Rename Files**

To rename a file with the File System module, use the fs.rename() method.

The fs.rename() method renames the specified file:

var fs = require('fs');

fs.rename('myoldname.txt', 'myrenamedfile.txt', function (err) {

if (err) throw err;

console.log('File Renamed!');

});

**Delete Files**

To delete a file with the File System module, use the fs.unlink() method.

The fs.unlink() method deletes the specified file:

var fs = require('fs');

fs.unlink('test.txt', function () {

console.log('Delete operation complete.');

});

**Crud of File System**

const fs = require('fs');

const path = require('path');

const dirPath= path.join(\_\_dirname,'crud');

const filePath = `${dirPath}/apple.txt`;

//create file in crud folder

fs.writeFileSync(filePath,'this is a simple file',(err)=>{

if(!err) console.log("file is created")

})

// read

/\*

fs.readFile(filePath,'utf8',(err,item)=>{

console.log(item);

})

\*/

// update

/\*

fs.appendFile(filePath,' for fruits',(err)=>{

if(!err) console.log("file is updated")

})

\*/

// rename

/\*

fs.rename(filePath, `${dirPath}/fruit.txt`,(err)=>{

if(!err) console.log("file name is updated")

})

\*/

// Delete

/\*

fs.unlinkSync(`${dirPath}/fruit.txt`,(err)=>{

if(!err) console.log("file is deleted")

})

\*/

## **What is Callback?**

A Callback in Node.js is an asynchronous equivalent for a function. It is a special type of function passed as an argument to another function. Node.js makes heavy use of callbacks. Callbacks help us make asynchronous calls. All the APIs of Node are written in such a way that they support callbacks.

1. Call back

setTimeout(function () {

console.log('This prints after 1000 ms');

}, 1000);

console.log("Hello World");

1. Blocking Code Example

var fs = require("fs");

var data = fs.readFileSync('input.txt');

console.log(data.toString());

console.log("Program Ended");

1. Non-Blocking Code Example

var fs = require("fs");

fs.readFile('input.txt', function (err, data) {

if (err) return console.error(err);

console.log(data.toString());

});

console.log("Program Ended");

**What is Chaining in Node.js ?**

npm init

npm i async

There are two most commonly used methods for chaining functions

**parallel(tasks, callback):** The tasks is a collection of functions that

runs parallel in practice through I/O switching.

If any function in the collection tasks returns an error, the callback function is fired.

Once all the functions are completed, the data is passed to the callback function

as an array. The callback function is optional.

**series(tasks, callback):** Each function in tasks run only after the

previous function is completed. If any of the functions throw an error,

the subsequent functions are not executed and the callback is fired with an

error value. On completion of tasks, the data is passed into the callback

function as an array.

**parallel**

Note : call as per statement

const async = require('async');

async.parallel([

()=>{

setTimeout(() => {

console.log('This is the first function');

}, 5000);

callback(null, 1);

},

()=>{

console.log('This is the second function');

callback(null, 2);

}

], (err, results) => {

if (err) console.error(err);

console.log(results);

});

**Series : run step by step and waiting for completion of earlier function**

async.series([

(callback) => {

setTimeout(() => {

console.log('This is the first function');

callback(null, 1);

}, 5000);

},

(callback) => {

console.log('This is the second function');

callback(null, 2);

}

], (err, results) => {

if (err) console.error(err);

console.log(results);

});

**Promise**

A Promise contains both the producing code and calls to the consuming code:

let myPromise = new Promise(function(myResolve, myReject) {

// "Producing Code" (May take some time)

myResolve(); // when successful

myReject(); // when error

});

myPromise.then(

function(value) { code if successful },

function(error) { code if some error }

);

Example

let a=20;

let b=0;

let waitingData= new Promise((resolve,reject)=>{

setTimeout(()=>{

resolve(30);

},2000)

})

waitingData.then((result)=>{

b=result;

console.log(a+b)

})