Node Js

What is Node.js?

* **Node is not a Languages**
* **Node.js is an open source server environment**
* **Node can connect with database but Javascript Not direct**
* **Code & syntex is very similar to javascript but Not exactly same**
* **Node.js is free**
* **Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)**
* **Node.js uses JavaScript on the server**
* **Node.js runs the chrome V8 JavaScript engine, the core of Google Chrome, outside of the browser.**

* **Written in c,c++,javascript**

**==============================================================**

* It is a popular tool for almost any kind of project!
* Most useful for create api and also create website
* 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.

Why do we use Node.js?

* **Node Mostly used for api create (Fast API WORK)**
* **So we can connect the same database with web app, Mobile App**
* **Node easy because who know javascript**
* **Node super fast for APIs**
* **With Node & Javascript , you can become full stack Developer / Mern Stack**

**====================================================================**

Javascript and Node are same?

* **Javascript and Node.js code syntax is same**
* **If you know javascript then you can learn node very easily**
* **But Both are note same**
* **You can not connect javascript with database**
* **Node js run on server side**
* **Javascript run on browser side**

**===================================================================**

**Topic**

* **Node js Introduction**
* **How works**
* **Install and run**
* **Make basic api**
* **Use with express.js**
* **UI with Node**
  + **UI EVENT**
  + **FORMS**
  + **WEB PAGES**

**====================================================================**

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"

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

Install and Setup Node JS

* **Download Node**

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

* **Install NPM AND NODE  => node js**
* **Code Editors**

**VS CODE**

**Sublime**

**========================================================================**

First Script with Node

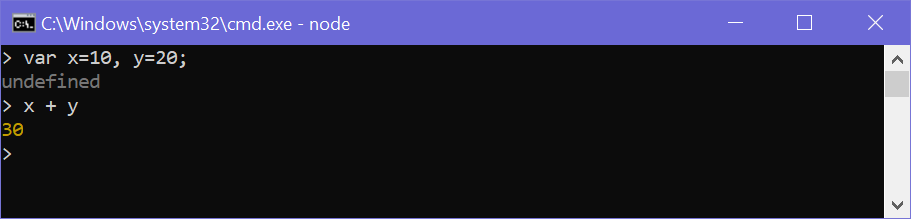
Node.js Console/REPL Terminal

Node.js comes with virtual environment called REPL (aka Node shell). REPL stands for Read-Eval-Print-Loop. It is a quick and easy way to test simple Node.js/JavaScript code.

To launch the REPL (Node shell), open command prompt (in Windows) or terminal (in Mac or UNIX/Linux) and type *node* as shown below. It will change the prompt to > in Windows and MAC.

Run : start => node.js

You can also define variables and perform some operation on them.



You can execute an external JavaScript file by executing the node fileName command. For example, the following runs mynodejs-app.js on the command prompt/terminal and displays the result.

mynodejs-app.js Copy

console.log("Hello World");

Now, you can execute mynodejs-app from command prompt as shown below.



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

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’

The following table lists important REPL commands.

|  |  |
| --- | --- |
| REPL Command | Description |
| .help | Display help on all the commands |
| tab Keys | Display the list of all commands. |
| Up/Down Keys | See previous commands applied in REPL. |
| .save filename | Save current Node REPL session to a file. |
| .load filename | Load the specified file in the current Node REPL session. |
| ctrl + c | Terminate the current command. |
| ctrl + c (twice) | Exit from the REPL. |
| ctrl + d | Exit from the REPL. |
| .break | Exit from multiline expression. |
| .clear | Exit from multiline expression. |

**Interview Que**

**Que :why its error come in cmd like undefined :**

Ans : Not return anything

**Que :how to different node console & javascript console**

Ans : We write console in cmd its Node build in module & we write in js file console its javascript browser functionality

in node module provide console for debugging

console.log()

console.error()

console.warn()

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

With File script

Step: 1 Create file **index.js**

**Code file** : console.log(‘Hello world’);

**Run cmd : node index.js**

Node.js Basics

* Node.js supports JavaScript. So, JavaScript syntax on Node.js is similar to the browser's JavaScript syntax.
* Javascript Fundamental for Node.js
* Condition , Loop , Array
* The Import Function and variables from another file

**NodeJS module system**

Modulein Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.jsapplication Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope. Also, each module can be placed in a separate .js file under a separate folder.

1. Core Modules

2. Own Modules or Local Modules

3. NPM Modules or Third Party Module

* 1. **CoreModules**

Core Modules covers minimum functionality of Node.js

These core modules are compiled into its binary distribution and load automatically when Node.js process starts You only need to import the core module first in order to use it in your application

Loading Core Modules

**var module = require('module\_name')**

|  |  |
| --- | --- |
| **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 |

**Exa 1 : http**

**App.js**

var fs=require('fs');

fs.appendFileSync('demo.txt','Hi in am first created file by fs Core module');

**Exa 2 : http**

**var http = require('http');**

**http.createServer((req,resp)=>{**

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

**//resp.write('<h1>Hi i am http module sserver </h1>')**

**resp.write(JSON.stringify({name:"raj",age:31,mobile:"1234567891"}));**

**resp.end();**

**}).listen(5000);**

* 1. **Own Modules or Local Modules**

**Local modules are modules created locally in your Node.js application.**

**These modules include different functionalities of your application in separate**

**files and folders.**

**You can also package it and distribute it via NPM,**

**so that Node.js communitycan use it.**

**We need module.exports to define what couldbe access from other file**

**todatDate.js**

**// this my custome own /local module**

**var todayDate=()=>{**

**var today = new Date();**

**var date = (today.getDate())+'-'+ (today.getMonth()+1) +'-'+(today.getFullYear());**

**return date;**

**}**

**// local module export**

**module.exports = todayDate ;**

**app.js**

// import

var date=require('./todayDate.js');

var result=date();

console.log(result);

/\*

* 1. **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

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

NPM - Node Package Manager

Node Package Manager (NPM) is a command line tool that installs, updates or uninstalls Node.js packages in your application. It is also an online repository for open-source Node.js packages. The node community around the world creates useful modules and publishes them as packages in this repository.

NPM is included with Node.js installation. After you install Node.js, verify NPM installation by writing the following command in terminal or command prompt

C:\> npm -v

If you have an older version of NPM then you can update it to the latest version using the following command.

C:\> npm install npm -g

To access NPM help, write npm help in the command prompt or terminal window.

C:\> npm help

NPM performs the operation in two **modes: global and local.** In the global mode, NPM performs operations which affect all the Node.js applications on the computer whereas in the local mode, NPM performs operations for the particular local directory which affects an application in that directory only.

Install Package Locally

Use the following command to install any third party module in your local Node.js project folder.

C:\>npm install <package name>

For example, the following command will install ExpressJS into MyNodeProj folder.

C:\MyNodeProj> npm install express

All the modules installed using NPM are installed under node\_modules folder. The above command will create ExpressJS folder under node\_modules folder in the root folder of your project and install Express.js there.

Add Dependency into package.json

Use --save at the end of the install command to add dependency entry into package.json of your application.

For example, the following command will install ExpressJS in your application and also adds dependency entry into the package.json.

C:\MyNodeProj> npm install express --save

C:\MyNodeProj> npm install express --save

The package.json of NodejsConsoleApp project will look something like below.

Package.json

{

"name": "NodejsConsoleApp",

"version": "0.0.0",

"description": "NodejsConsoleApp",

"main": "app.js",

"author":

{

"name": "Dev",

"email": ""

},

"dependencies":

{

"express": "^4.13.3"

}

}

Install Package Globally

NPM can also install packages globally so that all the node.js application on that computer can import and use the installed packages. NPM installs global packages into */<User>/local/lib/node\_modules* folder.

Apply -g in the install command to install package globally. For example, the following command will install ExpressJS globally.

C:\MyNodeProj> npm install -g express

Update Package

C:\MyNodeProj> npm update <package name>

The following command will update the existing ExpressJS module to the latest version.

C:\MyNodeProj> npm update express

Uninstall Packages

C:\>npm uninstall <package name>

C:\MyNodeProj> npm uninstall express

Que : What happened if node\_module delete & how load again

ans : npm install

Que :Correct way to push data on git

Ans : create file **.gitignore**  in root dire

.gitignore    add belove code in file means not push beloved folder

/node\_module

**Nodemon Package / Time Saving Module**

**All time we have to run node index.js for server run if any changes any file**

Cmd : npm i nodemon -g    globally  in window

          Sudo npm i nodeon -g      in linux

Now Run : **nodemon index.js**  or **nodemon**   and save all time working

**Note : index.js auto take nodemon so not required to write**

**If some securities added in pc then not work globally that time**

Go : search : cmd with administrative then we can done globally

Input get from Command Line

Core Module (process)

process.argv

The process.argv in node is used to access and interact with the command-line arguments.

It is an array that contains the command line arguments and helps user to

interact node app using CLI.

This property returns an array containing the arguments passed to the process

when run in the command line.

process.argv[0]: is the process execution path node

process.argv[1]: is the path for the js file. myfile.txt

process.argv[2] and more: other items in the array are the additional arguments passed by the user.

We can take input from user using command line arguments (process.argv)

node app.js add

- process.argv is Global object like console

console.log(process.argv)

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

node app.js add

0 1 2 3 4 5

cmd go : node app.js raj mahesh nikita nipun // argument start with 0 index

**Example : app.js**

// Importing the process module

const process = require('process');

// Printing process.argv property value

console.log(process.argv);

const data=process.argv;

if(data[2]=="raj")

{

console.log('my name is raj');

}

else if(data[3]=="nikita")

{

console.log('my name is nikita');

}

else

{

console.log('not user');

}

**Yargs : Third party core module : Command Line get & create custom command**

Yargs module is used for creating your own command-line commands in node.js and helps

in generating an elegant user interface. This module makes command-line arguments

flexible and easy to use.

Arguments Parsing with Yargs

Now in above example

node app.js add –title=“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

**Example : app.js**

const yargs = require('yargs');

// Create read command

yargs.command({

command: 'read',

describe: 'Read a note',

handler: function () {

console.log('Reading a note')

}

})

Example 2: app.js

// nodemon app.js add --title="Diwali Sale" --body="50% discount on Diwali sale"

const yargs = require('yargs')

// Customize yargs version

yargs.version('1.1.0')

// Create add command

yargs.command({

command: 'add',

describe: 'Add a new note',

builder: {

title: {

describe: 'Note title',

demandOption: true,

type: 'string'

},

body: {

describe: 'Note body',

demandOption: true,

type: 'string'

}

},

handler: function (argv) {

console.log('Title: ' + argv.title)

console.log('Body: ' + argv.body)

}

})

yargs.parse()

**File Management System By : fs module**

Node.js File System (core Module fs )

Node.js includes fs module to access physical file system. The fs module is responsible for all the asynchronous or synchronous file I/O operations.

Let's see some of the common I/O operation examples using fs module.

Common use for the File System module:

Read files

Create files

Update files

Delete files

Rename files

Reading File

fs.readFile(fileName [,options], callback)

Use fs.readFile() method to read the physical file asynchronously.

**Example Read file : app.js**

var fs = require('fs');

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

{

console.log(data);

});

**Example Read file in server: app.js**

/\*

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);

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

**File Management CRUD**

Node.js File System / Crud

1) Read

fs.readFile(fileName [,options], callback)

var data = fs.readFileSync('dummyfile.txt', 'utf8');

2) create & Write

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

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

3) update & write

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

4) delete

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

5) Rename

fs.rename(oldPath, newPath, callback)

\*/

const fs = require('fs');

/\*

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

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

})

\*/

// read

/\*

fs.readFile('raj.txt','utf8',(err,data)=>{

console.log(data);

})

\*/

// update

/\*

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

if(!err) {

console.log("file is updated");

}

})

\*/

// rename

/\*

fs.rename('raj.txt','rajnew.txt',(err)=>{

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

})

\*/

// Delete fs.unlink('test.txt', function ())

/\*

fs.unlinkSync('rajnew.txt',(err)=>{

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

});

\*/

**File Management CRUD in Dir (path core module)**

/\*

Node.js File System / Crud in dir

core module : path

const path = require('node:path');

const notes = '/users/joe/notes.txt';

path.dirname(notes); // /users/joe

path.basename(notes); // notes.txt

path.extname(notes); // .txt

\*/

const fs = require('fs');

//path Core Module

const path = require('path'); // folder module

const dirPath= path.join(\_\_dirname,'myDir'); // current crud folder path

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

//create file &Write data in myDir folder

/\*

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

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

})

\*/

// read

/\*

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

console.log(data);

})

/\*

**==============================================**

**File Management CRUD by process.argv**

Example : app.js

var fs = require('fs');

//var data = fs.writeFileSync('dummyfile.txt', 'writecode');

const input=process.argv;

if(input[2]=='add')

{

fs.writeFileSync(input[3],input[4]);

}

else if(input[2]=='read')

{

fs.readFile(input[3],'utf8',(err,data)=>{

console.log(data);

})

}

else if(input[2]=='remove')

{

fs.unlinkSync(input[3]);

}

else

{

console.log('Invalid input');

}

//cmd : node index.js add myfile.txt 'hi hello' // add file

//cmd : node index.js read myfile.txt // read file

//cmd : node index.js remove myfile.txt // remove file

**Nodemon Package / Time Saving Module**

**All time we have to run node index.js for server run if any changes any file**

Cmd : npm i nodemon -g    globally  in window

          Sudo npm i nodeon -g      in linux

Now Run : **nodemon index.js**  or **nodemon**   and save all time working

**Note : index.js auto take nodemon so not required to write**

**If some securities added in pc then not work globally that time**

Go : search : cmd with administrative then we can done globally

**Object Destructuring**

const user = {

    'name': 'Alex',

    'address': '15th Park Avenue',

    'age': 43

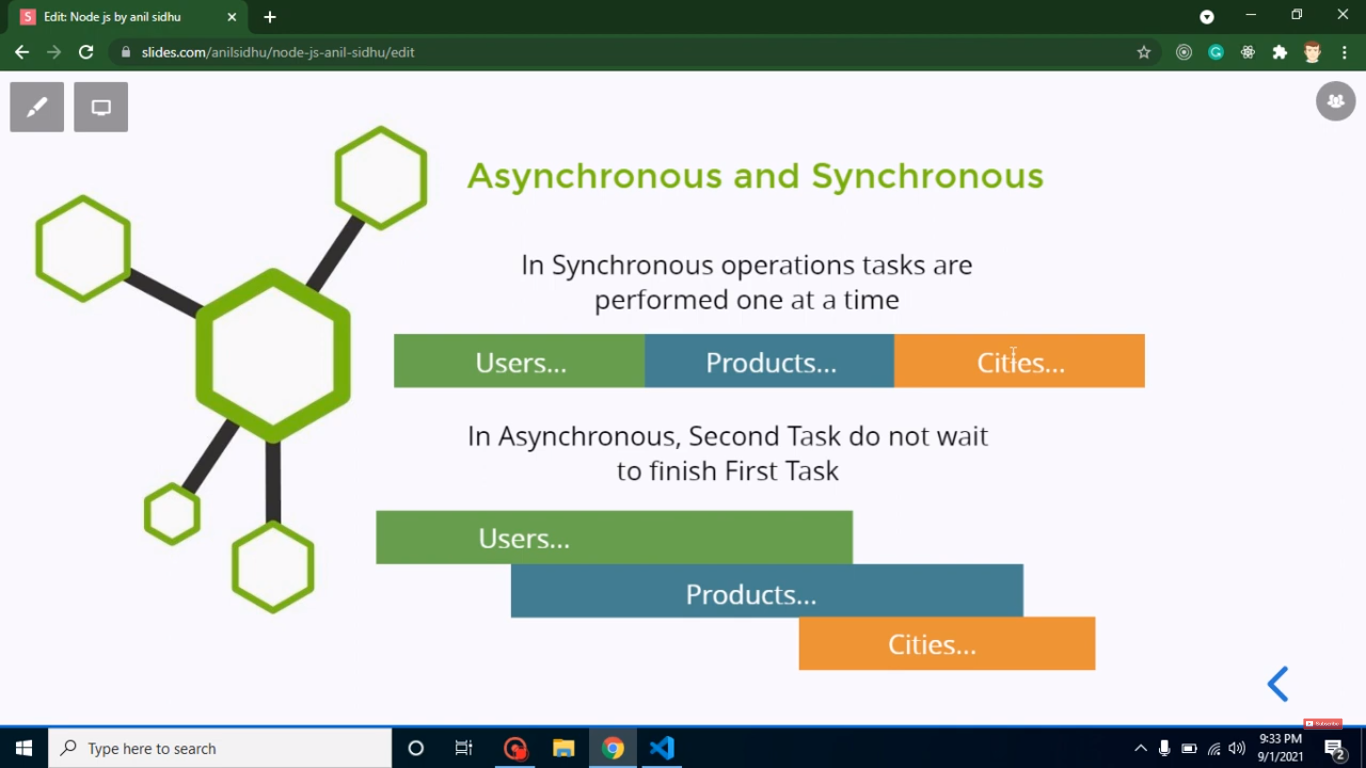
}

**const { name, age } = user;**

console.log(name, age); // Output, Alex 43

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

**Asynchronous and Synchronous**



Int Que: node is sync or async  language

Ans : async

4 all stript run together

**If 1 run & then 2 run and if 3 has problem then stop for 3 then 4 run that's called sync**

But In node :

**If 1 run & then 2 run and if 3 has problem then not wait for 3 , next 4 run and after last 3 run**

So not wait in running so its super fast Node js

Example

let a=20;

let b=0;

setTimeout(()=>{

   b=30;

},2000)

console.log(a+b)

**Handel\_Async by Promise**

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)

})

What is Callback?

Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.

Blocking Code Example

Create a text file named input.txt with the following content −

Tutorials Point is giving self learning content

to teach the world in simple and easy way!!!!!

Create a js file named main.js with the following code −

var fs = require("fs");

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

console.log(data.toString());

console.log("Program Ended");

Now run the main.js to see the result −

$ node main.js

Verify the Output.

Tutorials Point is giving self learning content

to teach the world in simple and easy way!!!!!

Program Ended

Non-Blocking Code Example

Create a text file named input.txt with the following content.

Tutorials Point is giving self learning content

to teach the world in simple and easy way!!!!!

Update main.js to have the following code −

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");

Now run the main.js to see the result −

$ node main.js

Verify the Output.

Program Ended

Tutorials Point is giving self learning content

to teach the world in simple and easy way!!!!!

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

Chaining in Node.js can be achieved using the async npm module. In order to install the async module, we need to run the following script in our directory:

npm init

npm i async

There are two most commonly used methods for chaining functions provided by the async module:

* **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.

**Example 1: Parallel Chaining**

**Filename: index.js**

const async = require('async');

async.parallel([

(callback) => {

setTimeout(() => {

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

callback(null, 1);

}, 500);

},

(callback) => {

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

callback(null, 2);

}

], (err, results) => {

if (err) console.error(err);

console.log(results);

});

**Cmd: node index.js**

**Output:**

**This is the second function**

**This is the first function**

**[ 1, 2 ]**

# Node.js Web Server

In this section, we will learn how to create a simple Node.js web server and handle HTTP requests.

To access web pages of any web application, you need a [web server](https://en.wikipedia.org/wiki/Web_server). The web server will handle all the http requests for the web application e.g IIS is a web server for ASP.NET web applications and Apache is a web server for PHP or Java web applications.

Node.js provides capabilities to create your own web server which will handle HTTP requests asynchronously. You can use IIS or Apache to run Node.js web application but it is recommended to use Node.js web server.

## Create Node.js Web Server

Node.js makes it easy to create a simple web server that processes incoming requests asynchronously.

var http = require('http'); // 1 - Import Node.js core module var server = http.createServer(function (req, res) {

// 2 - creating server //handle incoming requests here..

});

server.listen(5000); //3 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')

## Handle HTTP Request

The **http.createServer()** method includes [request](https://nodejs.org/api/http.html#http_http_incomingmessage) and [response](https://nodejs.org/api/http.html#http_class_http_serverresponse) parameters which is supplied by Node.js. The request object can be used to get information about the current HTTP request e.g., url, request header, and data. The response object can be used to send a response for a current HTTP request.

var http = require('http'); // Import Node.js core module

var server = http.createServer(function (req, res) { //create web server

if (req.url == '/')

{

//check the URL of the current request

// set response header

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

// set response content

res.write('<html><body><p>Hello World</p></body></html>');

res.end();

}

else if (req.url == "/student")

{

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

res.write('<html><body><p>Hello student</p></body></html>');

res.end();

}

else res.end('Invalid Request!');

});

server.listen(5000); //6 - listen for any incoming requests console.log('Node.js web server at port 5000 is running..')

C:\> node server.js

Node.js web server at port 5000 is running..

http://localhost:5000

## Sending JSON Response / Make API

The following example demonstrates how to serve JSON response from the Node.js web server.

Var http=require('http');

var server = http.createServer(function (req, res) {

if (req.url == '/data')

{

//check the URL of the current request

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

res.write(JSON.stringify({ message: "Hello World"}));

res.end();

}

});

server.listen(5000);

console.log('Node.js web server at port 5000 is running..')

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

# Node.js File System / Crud

Node.js includes fs module to access physical file system. The fs module is responsible for all the asynchronous or synchronous file I/O operations.

Let's see some of the common I/O operation examples using fs module.

## Reading File

Use fs.readFile() method to read the physical file asynchronously.

fs.readFile(fileName [,options], callback)

Parameter Description:

* filename: Full path and name of the file as a string.
* options: The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r".
* callback: A function with two parameters err and fd. This will get called when readFile operation completes.

var fs = require('fs');

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

{

    if (err) throw err;

    console.log(data);

});

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

//Use fs.readFileSync() method to read file synchronously as shown below.

var fs = require('fs');

var data = fs.readFileSync('dummyfile.txt', 'utf8');

console.log(data);

## Writing File

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.

var fs = require('fs');

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

{

if (err) console.log(err);

else

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

});

**Update File**

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

Example: Append File Content  / Update File

var fs = require('fs');

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

{

if (err) console.log(err);

else

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

});

## Open File

Alternatively, you can open a file for reading or writing using fs.open() method.

### Flags

The following table lists all the flags which can be used in read/write operation.

|  |  |
| --- | --- |
| Flag | Description |
| r | Open file for reading. An exception occurs if the file does not exist. |
| r+ | Open file for reading and writing. An exception occurs if the file does not exist. |
| rs | Open file for reading in synchronous mode. |
| rs+ | Open file for reading and writing, telling the OS to open it synchronously. See notes for 'rs' about using this with caution. |
| w | Open file for writing. The file is created (if it does not exist) or truncated (if it exists). |
| wx | Like 'w' but fails if path exists. |
| w+ | Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists). |
| wx+ | Like 'w+' but fails if path exists. |
| a | Open file for appending. The file is created if it does not exist. |
| ax | Like 'a' but fails if path exists. |
| a+ | Open file for reading and appending. The file is created if it does not exist. |
| ax+ | Like 'a+' but fails if path exists. |

## Delete File

Use fs.unlink() method to delete an existing file.

fs.unlink(path, callback);

var fs = require('fs');

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

{

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

});

|  |  |
| --- | --- |
| Method | Description |
| fs.readFile(fileName [,options], callback) | Reads existing file. |
| fs.writeFile(filename, data[, options], callback) | Writes to the file. If file exists then overwrite the content otherwise creates new file. |
| fs.open(path, flags[, mode], callback) | Opens file for reading or writing. |
| fs.rename(oldPath, newPath, callback) | Renames an existing file. |
| fs.chown(path, uid, gid, callback) | Asynchronous chown. |
| fs.stat(path, callback) | Returns fs.stat object which includes important file statistics. |
| fs.link(srcpath, dstpath, callback) | Links file asynchronously. |
| fs.symlink(destination, path[, type], callback) | Symlink asynchronously. |
| fs.rmdir(path, callback) | Renames an existing directory. |
| fs.mkdir(path[, mode], callback) | Creates a new directory. |
| fs.readdir(path, callback) | Reads the content of the specified directory. |
| fs.utimes(path, atime, mtime, callback) | Changes the timestamp of the file. |
| fs.exists(path, callback) | Determines whether the specified file exists or not. |
| fs.access(path[, mode], callback) | Tests a user's permissions for the specified file. |
| fs.appendFile(file, data[, options], callback) | Appends new content to the existing file. |

—------------------------------------------------------------

**Crud File System**

const fs = require('fs');

const path = require('path');

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

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

//create

// fs.writeFileSync(filePath,'this is a simple file');

// 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`);

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

# Input get from Command Line

Get input from command line then create file and add in file

Index.js

console.log(process.argv)

**Cmd : nodemon index.js hi hello**   // get all argument on console

console.log(process.argv[2]);

**Cmd : node index.js hi hello**   // get 3rd position argument on console as per array

**Now create file by cmd input**

Index.js

var fs = require('fs');

//var data = fs.writeFileSync('dummyfile.txt', 'writecode');

const input=process.argv;

fs.writeFileSync(input[2],input[3]);

Cmd : node index.js apple.txt  ‘hi hello ’

—-------------------------------------------------------------------------

**Create and remove file by cmd input**

var fs = require('fs');

//var data = fs.writeFileSync('dummyfile.txt', 'writecode');

const input=process.argv;

if(input[2]=='add')

{

fs.writeFileSync(input[3],input[4]);

}

else if(input[2]=='remove')

{

fs.unlinkSync(input[3]);

}

else

{

console.log('Invalid input');

}

//cmd : node index.js add myfile.txt 'hi hello'  // add file

//cmd : node index.js remove myfile.txt  // remove file

# Display file list from folder / read from file in path

const fs= require('fs');

const path=require('path');

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

console.log(dirPath)

// for(i=0;i<5;i++)

// {

//     fs.writeFileSync(`${dirPath}/hello${i}.txt`,"some simple text in file")

// }

**Show file list**

fs.readdir(dirPath,(err,files)=>{

    files.forEach((item)=>{

        console.warn("file name is : ",item)

    });

}

)

Cmd : node index.js

# Debug Node.js Application

In this section, you will learn ways to debug Node.js application.

You can debug Node.js application using various tools including following:

1. Core Node.js debugger
2. Node Inspector
3. Built-in debugger in IDEs

## Core Node.js Debugger

Node.js provides built-in non-graphic debugging tool that can be used on all platforms. It provides different commands for debugging Node.js application.

Consider the following simple Node.js application contained in app.js file.

App.js

var fs = require('fs');

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

debugger;

if (err) throw err;

console.log(data);

});

Write debugger in your JavaScript code where you want debugger to stop. For example, we want to check the "data" parameter in the above example. So, write debugger; inside callback function as above.

Now, to debug the above application, run the following command.

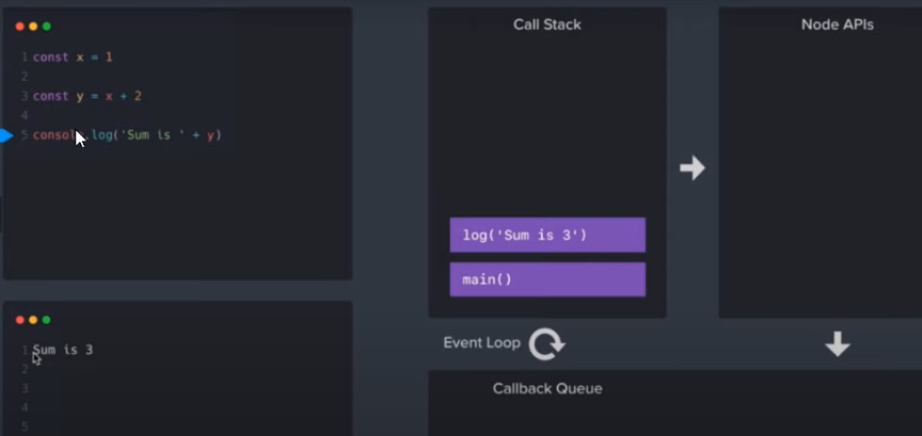
node debug app.js

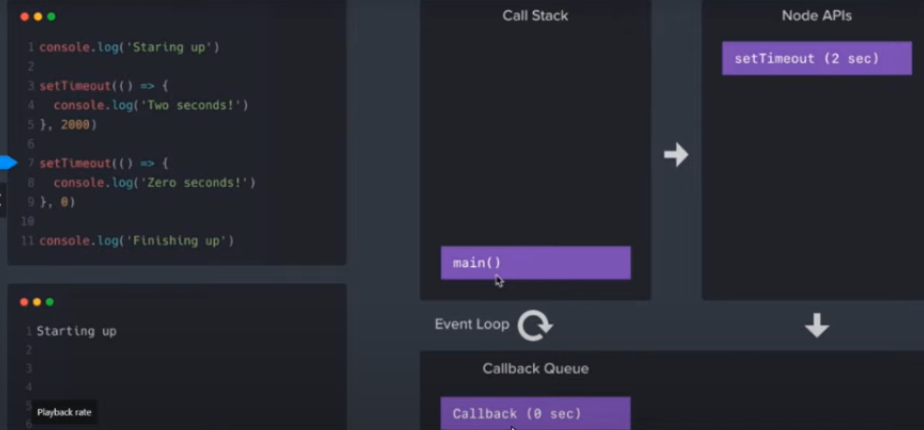
The above command starts the debugger and stops at the first line as shown below.

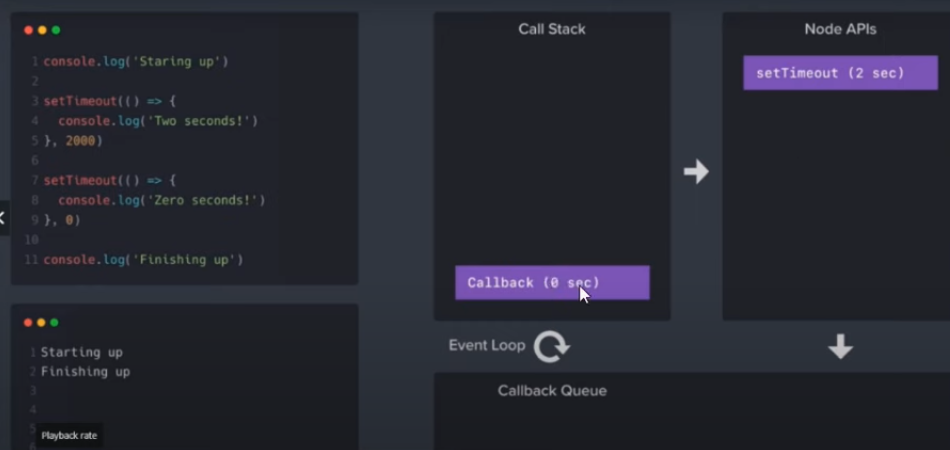
The following table lists important debugging commands:

|  |  |
| --- | --- |
| Command | Description |
| next | Stop at the next statement. |
| cont | Continue execute and stop at the debugger statement if any. |
| step | Step in function. |
| out | Step out of function. |
| watch | Add the expression or variable into watch. |
| watcher | See the value of all expressions and variables added into watch. |
| Pause | Pause running code. |

**How Node Work**

****





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

# Frameworks for Node.js

**There are various third party open-source frameworks available in Node Package Manager which makes Node.js application development faster and easy. You can choose an appropriate framework as per your application requirements.**

**The following table lists frameworks for Node.js.**

|  |  |
| --- | --- |
| [**Express.js**](https://expressjs.com/) | **Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. This is the most popular framework as of now for Node.js.** |
| [**Geddy**](http://geddyjs.org/) | **Geddy is a simple, structured web application framework for Node.js based on MVC architecture.** |
| [**Locomotive**](http://locomotivejs.org/) | **Locomotive is MVC web application framework for Node.js. It supports MVC patterns, RESTful routes, and convention over configuration, while integrating seamlessly with any database and template engine. Locomotive builds on Express, preserving the power and simplicity you've come to expect from Node.** |
| [**Koa**](http://koajs.com/) | **Koa is a new web framework designed by the team behind Express, which aims to be a smaller, more expressive, and more robust foundation for web applications and APIs.** |

Que : define all api status code

200  OK

201 [201 Created](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/201)

403 [403 Forbidden](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/403)

404 [404 Not Found](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404)

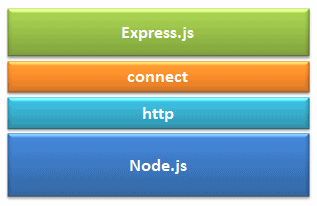
500 [500 Internal Server Error](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/500)

# Express.js

**Express is a fast, unopin ionated minimalist web framework for Node.js" - official web site:** [**Expressjs.com**](https://expressjs.com/)

**Express.js is a web application framework for Node.js. It provides various features that make web application development fast and easy which otherwise takes more time using only Node.js.**

**Express.js is based on the Node.js middleware module called** *connect* **which in turn uses http module. So, any middleware which is based on connect will also work with Express.js.**

****

## Advantages of Express.js

1. **Makes Node.js web application development fast and easy.**
2. **Easy to configure and customize.**
3. **Allows you to define routes of your application based on HTTP methods and URLs.**
4. **Includes various middleware modules which you can use to perform additional tasks on request and response.**
5. **Easy to integrate with different template engines like Jade, Vash, EJS etc.**
6. **Allows you to define an error handling middleware.**
7. **Easy to serve static files and resources of your application.**
8. **Allows you to create REST API server.**
9. **Easy to connect with databases such as MongoDB, Redis, MySQL**

## Install Express.js - Environment

<https://expressjs.com/>

**You can install express.js using npm. The following command will install latest version of express.js globally on your machine so that every Node.js application on your machine can use it.**

**npm install -g express**

**The following command will install latest version of express.js local to your project folder.**

**C:\MyNodeJSApp> npm install express --save**

**As you know, --save will update the package.json file by specifying express.js dependency.**

**Example of express js**

const express = require('express');

const app = express();

app.get("", (req, resp) => {

    resp.send("Welcome Index Page");

});

app.get("/about", (req, resp) => {

    resp.send("Welcome, This is a About Page");

});

app.get("/help", (req, resp) => {

    resp.send("Welcome, This is a Help Page");

});

app.listen(5000);

**Routing**

Web frameworks provide resources such as HTML pages, scripts, images, etc. at different routes.

## app.method(path, handler)

This METHOD can be applied to any one of the HTTP verbs – get, set, put, delete. An alternate method also exists, which executes independent of the request type.

**app.get('/',(req,res)=>{**

**res.send('<h1>Hello word</h1>');**

**})**

**app.get('/about',(req,res)=>{**

**res.send('<h1>Hello about</h1>');**

**})**

**app.get('/contact',(req,res)=>{**

**res.send('<h1>Hello contact</h1>');**

**})**

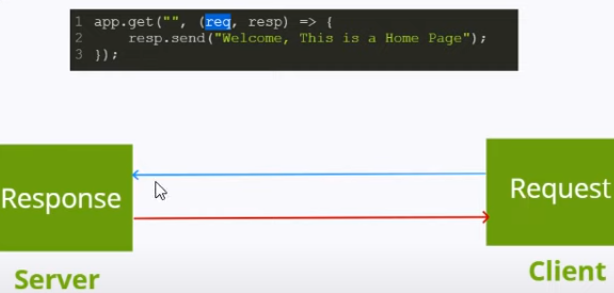
**app.get('/blog',(req,res)=>{**

**res.send('<h1>Hello blog</h1>');**

**})**

**app.listen(3000);**

# Routing Params - Request and Response

****

* In above code we just get **resp** not send any request so just only take blank req
* Now Want to send Request from url then use **req**

Index.js

app.get("/", (req, resp) => {

 //   console.log(“Data sent by Browser”, req.query) // get object

//   console.log(“Data sent by Browser”, req.query.name) // get name

    resp.send("Welcome Index Page : " + req.query.name);

});

Google : **localhost:5000/name=’Raj Nagar’**

Output :

**Data sent by Browser {name:Raj nagar}**

**Data sent by Browser Raj Nagar**

OnScreen : Welcome Index Page : Raj Nagar

**Send JSON Data & make API**

app.get("/api", (req, resp) => {

resp.send([{

name:'Peter',

email:'peter@test.com'

},

{

name:'Bruce',

email:'bruce@test.com'

}

]);

});

var obj=[

{firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"},

{firstName:"John1", lastName:"Doe1", age:50, eyeColor:"blue1"},

{firstName:"John2", lastName:"Doe2", age:50, eyeColor:"blue2"}

];

app.get("/api1", (req, resp) => {

resp.send(obj);

});

**Routing Static PAGE :**

**Make Folder for HTML file and access it**

Index.js

const express = require('express');

const path = require('path');  // path for make path

const app = express();

const publicPath=**path.join(\_\_dirname,'public')**

// app.use access any Static page from public

**app.use(express.static(publicPath));**

app.listen(5000);

Make Public folder in root and also add html pages

Index.html

About.html

Run : localhost:5000      //autocall index.html

Run : localhost:5000/about.html    about.html page run

**Roting : Remove extension from URL :**

**Make HTML Pages & Routing**

const path = require('path');

var publicpath=path.join(\_\_dirname,'public')

app.get('/',(req,resp)=>{

resp.sendFile(`${publicpath}/index.html`);

})

app.get('/about',(req,resp)=>{

resp.sendFile(`${publicpath}/about.html`);

})

app.get('/blog',(req,resp)=>{

resp.sendFile(`${publicpath}/blog.html`);

})

app.get('/\*',(req,resp)=>{

resp.sendFile(`${publicpath}/pnf.html`);

})

app.listen(3000);

Que :How to load page by get method in node.js

Ans : resp.sendFile()  function

**=========================================**