**Node JS Module**

Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.

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.

# Node.js Local Module

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 community can use it. For example, if you need to connect to MongoDB and fetch data then you can create a module for it, which can be reused in your application.

Writing Simple Module

Let's write simple logging module which logs the information, warning or error to the console.

In Node.js, module should be placed in a separate JavaScript file. So, create a Log.js file and write the following code in it.

Log.js Copy

var log = {

info: function (info) {

console.log('Info: ' + info);

},

warning:function (warning) {

console.log('Warning: ' + warning);

},

error:function (error) {

console.log('Error: ' + error);

}

};

module.exports = log

In the above example of logging module, we have created an object with three functions - info(), warning() and error(). At the end, we have assigned this object to module.exports. The module.exports in the above example exposes a log object as a module.

The module.exports is a special object which is included in every JS file in the Node.js application by default. Use module.exports or exports to expose a function, object or variable as a module in Node.js.

Now, let's see how to use the above logging module in our application.

Loading Local Module

To use local modules in your application, you need to load it using require() function in the same way as core module. However, you need to specify the path of JavaScript file of the module.

The following example demonstrates how to use the above logging module contained in Log.js.

app.js Copy

var myLogModule = require('./Log.js');

myLogModule.info('Node.js started');

In the above example, app.js is using log module. First, it loads the logging module using require() function and specified path where logging module is stored. Logging module is contained in Log.js file in the root folder. So, we have specified the path './Log.js' in the require() function. The '.' denotes a root folder.

The require() function returns a log object because logging module exposes an object in Log.js using module.exports. So now you can use logging module as an object and call any of its function using dot notation e.g myLogModule.info() or myLogModule.warning() or myLogModule.error()

Run the above example using command prompt (in Windows) as shown below.

C:\> node app.js

Info: Node.js started

Thus, you can create a local module using module.exports and use it in your application.

# Export Module in Node.js

Here, you will learn how to expose different types as a module using module.exports.

The module.exports is a special object which is included in every JavaScript file in the Node.js application by default. The module is a variable that represents the current module, and exports is an object that will be exposed as a module. So, whatever you assign to module.exports will be exposed as a module.

Let's see how to expose different types as a module using module.exports.

## Export Literals

As mentioned above, exports is an object. So it exposes whatever you assigned to it as a module. For example, if you assign a string literal then it will expose that string literal as a module.

The following example exposes simple string message as a module in Message.js.

Message.js Copy

module.exports = 'Hello world';

Now, import this message module and use it as shown below.

app.js Copy

var msg = require('./Message.js'); console.log(msg);

Run the above example and see the result, as shown below.

C:\> node app.js

Hello World

ExExport Object

The exports is an object. So, you can attach properties or methods to it. The following example exposes an object with a string property in Message.js file.

Message.js Copy

exports.SimpleMessage = 'Hello world'; //or module.exports.SimpleMessage = 'Hello world';

In the above example, we have attached a property SimpleMessage to the exports object. Now, import and use this module, as shown below.

app.js Copy

var msg = require('./Messages.js'); console.log(msg.SimpleMessage);

In the above example, the require() function will return an object { SimpleMessage : 'Hello World'} and assign it to the msg variable. So, now you can use msg.SimpleMessage.

Run the above example by writing node app.js in the command prompt and see the output as shown below.

C:\> node app.js

Hello World

In the same way as above, you can expose an object with function. The following example exposes an object with the log function as a module.

Log.js Copy

module.exports.log = function (msg) { console.log(msg); };

The above module will expose an object- { log : function(msg){ console.log(msg); } } . Use the above module as shown below.

app.js Copy

var msg = require('./Log.js'); msg.log('Hello World');

Run and see the output in command prompt as shown below.

C:\> node app.js

Hello World

You can also attach an object to module.exports, as shown below.

data.js Copy

module.exports = { firstName: 'James', lastName: 'Bond' }

app.js Copy

var person = require('./data.js'); console.log(person.firstName + ' ' + person.lastName);

Run the above example and see the result, as shown below.

C:\> node app.js

James Bond

## Export Function

You can attach an anonymous function to exports object as shown below.

Log.js Copy

module.exports = function (msg) { console.log(msg); };

Now, you can use the above module, as shown below.

app.js Copy

var msg = require('./Log.js'); msg('Hello World');

The msg variable becomes a function expression in the above example. So, you can invoke the function using parenthesis (). Run the above example and see the output as shown below.

C:\> node app.js

Hello World

## Export Function as a Class

In JavaScript, a function can be treated like a class. The following example exposes a function that can be used like a class.

Person.js Copy

module.exports = function (firstName, lastName) { this.firstName = firstName; this.lastName = lastName; this.fullName = function () { return this.firstName + ' ' + this.lastName; } }

The above module can be used, as shown below.

app.js Copy

var person = require('./Person.js'); var person1 = new person('James', 'Bond'); console.log(person1.fullName());

As you can see, we have created a person object using the new keyword. Run the above example, as shown below.

C:\> node app.js

James Bond

In this way, you can export and import a local module created in a separate file under root folder.

Node.js also allows you to create modules in sub folders. Let's see how to load module from sub folders.

## Load Module from the Separate Folder

Use the full path of a module file where you have exported it using module.exports. For example, if the log module in the log.js is stored under the utility folder under the root folder of your application, then import it, as shown below.

app.js Copy

var log = require('./utility/log.js');

In the above example, . is for the root folder, and then specify the exact path of your module file. Node.js also allows us to specify the path to the folder without specifying the file name. For example, you can specify only the utility folder without specifying log.js, as shown below.

app.js Copy

var log = require('./utility');

In the above example, Node.js will search for a package definition file called package.json inside the utility folder. This is because Node assumes that this folder is a package and will try to look for a package definition. The package.json file should be in a module directory. The package.json under utility folder specifies the file name using the main key, as shown below.

./utility/package.json Copy

{ "name" : "log", "main" : "./log.js" }

Now, Node.js will find the log.js file using the main entry in package.json and import it.

port Object

The exports is an object. So, you can attach properties or methods to it. The following example exposes an object with a string property in Message.js file.

Message.js Copy

exports.SimpleMessage = 'Hello world';

//or

module.exports.SimpleMessage = 'Hello world';

In the above example, we have attached a property SimpleMessage to the exports object. Now, import and use this module, as shown below.

app.js Copy

var msg = require('./Messages.js');

console.log(msg.SimpleMessage);

In the above example, the require() function will return an object { SimpleMessage : 'Hello World'} and assign it to the msg variable. So, now you can use msg.SimpleMessage.

Run the above example by writing node app.js in the command prompt and see the output as shown below.

C:\> node app.js

Hello World

In the same way as above, you can expose an object with function. The following example exposes an object with the log function as a module.

Log.js Copy

module.exports.log = function (msg) {

console.log(msg);

};

The above module will expose an object- { log : function(msg){ console.log(msg); } } . Use the above module as shown below.

app.js Copy

var msg = require('./Log.js');

msg.log('Hello World');

Run and see the output in command prompt as shown below.

C:\> node app.js

Hello World

You can also attach an object to module.exports, as shown below.

data.js Copy

module.exports = {

firstName: 'James',

lastName: 'Bond'

}

app.js Copy

var person = require('./data.js');

console.log(person.firstName + ' ' + person.lastName);

Run the above example and see the result, as shown below.

C:\> node app.js

James Bond

Export Function

You can attach an anonymous function to exports object as shown below.

Log.js Copy

module.exports = function (msg) {

console.log(msg);

};

Now, you can use the above module, as shown below.

app.js Copy

var msg = require('./Log.js');

msg('Hello World');

The msg variable becomes a function expression in the above example. So, you can invoke the function using parenthesis (). Run the above example and see the output as shown below.

C:\> node app.js

Hello World

Export Function as a Class

In JavaScript, a function can be treated like a class. The following example exposes a function that can be used like a class.

Person.js Copy

module.exports = function (firstName, lastName) {

this.firstName = firstName;

this.lastName = lastName;

this.fullName = function () {

return this.firstName + ' ' + this.lastName;

}

}

The above module can be used, as shown below.

app.js Copy

var person = require('./Person.js');

var person1 = new person('James', 'Bond');

console.log(person1.fullName());

As you can see, we have created a person object using the new keyword. Run the above example, as shown below.

C:\> node app.js

James Bond

In this way, you can export and import a local module created in a separate file under root folder.

Node.js also allows you to create modules in sub folders. Let's see how to load module from sub folders.

Load Module from the Separate Folder

Use the full path of a module file where you have exported it using module.exports. For example, if the log module in the log.js is stored under the utility folder under the root folder of your application, then import it, as shown below.

app.js Copy

var log = require('./utility/log.js');

In the above example, . is for the root folder, and then specify the exact path of your module file. Node.js also allows us to specify the path to the folder without specifying the file name. For example, you can specify only the utility folder without specifying log.js, as shown below.

app.js Copy

var log = require('./utility');

In the above example, Node.js will search for a package definition file called package.json inside the utility folder. This is because Node assumes that this folder is a package and will try to look for a package definition. The package.json file should be in a module directory. The package.json under utility folder specifies the file name using the main key, as shown below.

./utility/package.json Copy

{

"name" : "log",

"main" : "./log.js"

}

Now, Node.js will find the log.js file using the main entry in package.json and import it.

Note:

If the package.json file does not exist, then it will look for index.js file as a module file by default.

# 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

2.11.3

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

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

package.json Copy

{ "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

To update the package installed locally in your Node.js project, navigate the command prompt or terminal window path to the project folder and write the following update command.

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

Use the following command to remove a local package from your project.

C:\>npm uninstall <package name>

The following command will uninstall ExpressJS from the application.

C:\MyNodeProj> npm uninstall express

## 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

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

package.json Copy

{ "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

To update the package installed locally in your Node.js project, navigate the command prompt or terminal window path to the project folder and write the following update command.

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

Use the following command to remove a local package from your project.

C:\>npm uninstall <package name>

The following command will uninstall ExpressJS from the application.

C:\MyNodeProj> npm uninstall express

# 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.

The following example is a simple Node.js web server contained in server.js file.

server.js Copy

var http = require('http'); // 1 - Import Node.js core module var server = http.createServer(function (req, res) { // 2 - creating server //handle incomming requests here.. }); server.listen(5000); //3 - listen for any incoming requests console.log('Node.js web server at port 5000 is running..')

In the above example, we import the http module using require() function. The http module is a core module of Node.js, so no need to install it using NPM. The next step is to call createServer() method of http and specify callback function with request and response parameter. Finally, call listen() method of server object which was returned from createServer() method with port number, to start listening to incoming requests on port 5000. You can specify any unused port here.

Run the above web server by writing node server.js command in command prompt or terminal window and it will display message as shown below.

C:\> node server.js

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

This is how you create a Node.js web server using simple steps. Now, let's see how to handle HTTP request and send response in Node.js web server.

## 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.

The following example demonstrates handling HTTP request and response in Node.js.

server.js Copy

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>This is home Page.</p></body></html>'); res.end(); } else if (req.url == "/student") { res.writeHead(200, { 'Content-Type': 'text/html' }); res.write('<html><body><p>This is student Page.</p></body></html>'); res.end(); } else if (req.url == "/admin") { res.writeHead(200, { 'Content-Type': 'text/html' }); res.write('<html><body><p>This is admin Page.</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..')

In the above example, req.url is used to check the url of the current request and based on that it sends the response. To send a response, first it sets the response header using writeHead() method and then writes a string as a response body using write() method. Finally, Node.js web server sends the response using end() method.

Now, run the above web server as shown below.

C:\> node server.js

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

To test it, you can use the command-line program curl, which most Mac and Linux machines have pre-installed.

curl -i http://localhost:5000

You should see the following response.

HTTP/1.1 200 OK

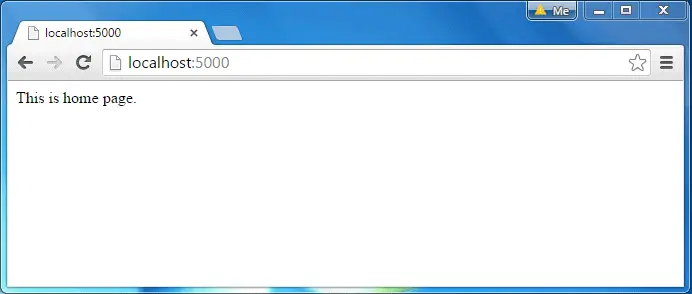
Content-Type: text/plain

Date: Tue, 8 Sep 2015 03:05:08 GMT

Connection: keep-alive

This is home page.

For Windows users, point your browser to *http://localhost:5000* and see the following result.



# 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.

The following example is a simple Node.js web server contained in server.js file.

server.js Copy

var http = require('http'); // 1 - Import Node.js core module var server = http.createServer(function (req, res) { // 2 - creating server //handle incomming requests here.. }); server.listen(5000); //3 - listen for any incoming requests console.log('Node.js web server at port 5000 is running..')

In the above example, we import the http module using require() function. The http module is a core module of Node.js, so no need to install it using NPM. The next step is to call createServer() method of http and specify callback function with request and response parameter. Finally, call listen() method of server object which was returned from createServer() method with port number, to start listening to incoming requests on port 5000. You can specify any unused port here.

Run the above web server by writing node server.js command in command prompt or terminal window and it will display message as shown below.

C:\> node server.js

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

This is how you create a Node.js web server using simple steps. Now, let's see how to handle HTTP request and send response in Node.js web server.

## 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.

The following example demonstrates handling HTTP request and response in Node.js.

server.js Copy

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>This is home Page.</p></body></html>'); res.end(); } else if (req.url == "/student") { res.writeHead(200, { 'Content-Type': 'text/html' }); res.write('<html><body><p>This is student Page.</p></body></html>'); res.end(); } else if (req.url == "/admin") { res.writeHead(200, { 'Content-Type': 'text/html' }); res.write('<html><body><p>This is admin Page.</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..')

In the above example, req.url is used to check the url of the current request and based on that it sends the response. To send a response, first it sets the response header using writeHead() method and then writes a string as a response body using write() method. Finally, Node.js web server sends the response using end() method.

Now, run the above web server as shown below.

C:\> node server.js

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

To test it, you can use the command-line program curl, which most Mac and Linux machines have pre-installed.

curl -i http://localhost:5000

You should see the following response.

HTTP/1.1 200 OK

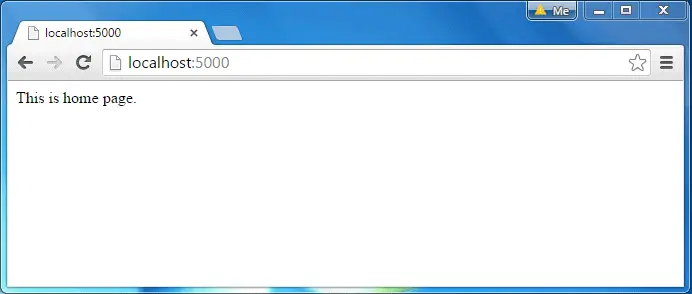
Content-Type: text/plain

Date: Tue, 8 Sep 2015 03:05:08 GMT

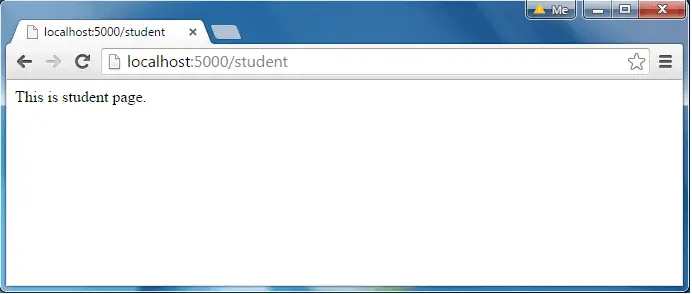
Connection: keep-alive

This is home page.

For Windows users, point your browser to *http://localhost:5000* and see the following result.



The same way, point your browser to *http://localhost:5000/student* and see the following result.



It will display "Invalid Request" for all requests other than the above URLs.

## Sending JSON Response

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

server.js Copy

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..')

So, this way you can create a simple web server that serves different responses.