**Modules in node.js**

Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files .

Each module in Node.js has its own context, so it cannot interfere with other modules.

Each module can be placed in a separate (.js )file under a separate folder.

So that every module is reusable in Node.js.

There are three types of modules:

1. Core Modules
2. Local Modules
3. Third Party Modules

Core Modules:

The following table lists some of the important core modules in Node.js.

http, url, querystring, path, fs, util.

In order to use Node.js core or NPM modules, you first need to import it using require() function.

var module=require(‘mod\_name);

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.

Create a module with name (name) and save it as name.js and export it.

Var mymodule=require(‘./name.js’);

By writing this line we can use it.

-----------------------------------------------------------------------------------------

NPM:

Node Package Manager (NPM) provides two main functionalities −

1. Online repositories for node.js packages/modules which are searchable on [search.nodejs.org](https://search.nodejs.org/" \t "_blank)
2. Command line utility to install Node.js packages, do version management and dependency management of Node.js packages.

Installing npm modules by typing command:

>>npm install 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 |
| [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 |

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.

Ex: var fs= require(‘fs’);

fs.readFile(‘file1.txt’,function(err,data){

if(err)

return console.error(err);

console.log(data.toString());

});

console.log(“program for callbacks”);

o/p: program for callbacks

input file content

What is Callback Hell in node.js ?

Callback hell is that when we try to write multiple asynchronous operations one after the other.

No readability, complex to handle this type of code.Debugging is tougher.Below example demonstrates the complexity.

Ex: doSwap(a,b,function(err,data){

doAdd(a,b,function(err,result\_in\_a){

doSub(a,b,function(err,result\_in\_b){

doSub1(a,b,function(err,res\_in\_a){

console(“completed swap”);});});});});

What are promises ?

The core idea behind promises is that a promise represents the result of an asynchronous operation. A promise is in one of three different states:

pending - The initial state of a promise.

fulfilled - The state of a promise representing a successful operation.

rejected - The state of a promise representing a failed operation.

**function** readFile(filename, enc){

**return** **new** Promise(**function** (fulfill, reject){

fs.readFile(filename, enc, **function** (err, res){

**if** (err) reject(err);

**else** fulfill(res);

});});}

We use new Promise to construct the promise.

We give the constructor a factory function which does the actual work.

This function has two arguments.

The first argument fulfills the promise and the second argument rejects the promise.

Once the operation has completed, we call the appropriate function.

Advantages

**promises** are cleaner way for running asynchronous tasks to synchronous

Also provide catching mechanism which are not in **callbacks**.

**Promises** are built **over callbacks, elegant way to handle asynchronous programs.**

**Promises**  allow cleaner and better, functional code with less errors.

Different promise libraries?

Bluebird and Q are famous one.

Bluebird:  
Bluebird is a fully-featured Promise library for JavaScript. The strongest feature of Bluebird is that it allows you to "promisify" other Node modules in order to use them asynchronously. Promisify is a concept applied to callback functions. This concept is used to ensure that every callback function which is called returns some sort of value.

****npm install bluebird //install it****

**Include by writing**

var Promise = require('bluebird');

Var mongo=promise.promisifyAll(require(‘mongodb’)).Mongoclient;

you can use BlueBird to make the[MongoDB](https://www.guru99.com/mongodb-tutorials.html)module run asynchronously,The next step is to include the bluebird module in your code and promisify the entire MongoDB module. By promisify, we mean that bluebird will ensure that each and every method defined in the MongoDB library returns a promise.

****Q library:****

var Q = require('q');

function async() {

return Q.delay(4000)

}

async()

.then(function() {

console.log('async called back');

});

**O/p: async called back//after 4 secs output displayed.**

**Differences:**

 Bluebird attempts to follow and augment the ES6 promise standard, while Q uses its own API. Either of these libraries can be used to help you manage and coordinate asynchronous operations.

Https and Http module:

The HTTPS module provides a way of making Node.js transfer data over HTTP TLS/SSL protocol, which is the secure HTTP protocol.

The syntax for including the HTTPS module :

var https = require('https');

Methods:

createServer():Creates an HTTPS Server.

Ex:

var https = require('https');//for including https module  
  
https.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/plain'});

// 200 means that all is OK its a status code, the second argument is an object containing the response headers.

//added a http header  
  res.write('Hello World!');//output  
  res.end();  
}).listen(8080);port number

O/p:Hello World!

Http module:

The HTTP module provides a way of making Node.js transfer data over HTTP (Hyper Text Transfer Protocol).

Same as above example but http module.

------------------------------------------------------------------------------------------------

Files:

Reading a file:

var fs= require(“fs”);//importing fs module

fs.readFile(‘file1.txt’,function(err,data)

{

if(err)

return console.error(err);

console.log(data.toString());

});

o/p: program for callbacks

input file content

Writing into file :

var fs = require('fs');

//create a file named newfile.txt:

fs.writeFile('newfile.txt', 'Hello berkadia!', function (err)

{

if (err) throw err;

console.log('Saved!');

});//save vth .js and execute

//open newfile.txt and check the content

Net Module:

Net module provides a way of creating TCP servers and TCP clients.

Syntax for including net module:

var net = require('net');

Methods & Description:

connect(): creates a new connection to server and return a new socket.

createConnection(): same as connect

createServer(): creates a new server

isIP(), isIP4(), isIPV6(): checks whether ip address is valid one or not.

Ex: server.js

var net = require('net');//including module

var server = net.createServer(function(connection) {

console.log('client connected');

connection.on('end', function() {

console.log('client disconnected');

});

connection.write('Hello World!\r\n');//writing

connection.pipe(connection);

});

server.listen(8080, function() {

console.log('server is listening');});

Client.js:

var net = require('net');//including module

var client = net.connect({port: 8080}, function() {

console.log('connected to server!');

});

client.on('data', function(data) {

console.log(data.toString());//you will get HelloWorld! as data

client.end();

});

client.on('end', function() {

console.log('disconnected from server');

});//execute server.js and client.js in parallel.

Output:

Server is listening

Connected to server

Hello World!

Disconnected from server.

Async/Await :---

A couple of months ago **async/await landed in V8**, the JavaScript engine.

await**may only be used in functions marked with the**async**keyword.**

If the awaited expression isn’t a promise, its casted into a promise.

 you should wrap await in try / catch so that you can capture and handle errors in awaited promises from within the async function.

Furthermore, an Async Function always returns a Promise. That promise is rejected in the case of uncaught exceptions, and it’s otherwise resolved to the return value of the async function. This enables us to invoke an async function and mix that with regular promise-based continuation as well.

This little example shows you how you can fetch data using the Fetch API and Promises.

Example using async and await:

function add()

{ a=4;//if a is definrd then error is printed there on terminal

return a;}

async function getTrace ()

{

let Content

try {

Content = await add(); }

catch (e) { console.error(e);//prints error if any in add() method

} return Content //if no error executes

} getTrace() .then()