**NodeJS**

* Node.Js is Javascript Runtime at server which allows the client applications to execute
* Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications.
* Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.
* Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.
* Node.Js is command line tool which will executes JavaScript at server
* Node.Js used in the situation when the end user requires to get any updation with out page refresh

Npm – node package manager

Bower – is another package manager

Framework = a sets of classes or packages + execution engine + language + package manager +

If some classes are not available by default in the framework , we can download it from global stores , which are called as repository . In .net it is Nuget package manager , in Node it is npm

**nodeJS = httpServer**

NodeJs is not actually Framework , but we can say it as Framework , bcz it has all the components of framework

nodeJS cane be defined as framework of Javascript to act as server

It can host any Javascript Related Applications

Bower is a third party package manager , mostly used to download front end libraries like bootstrap, but can download some server side also.

Bower is installed with NPM

But Most of the libraries installed with NPM

Node JS

Built In Classes

Node

(Execution Engine)

JavaScript

(Language)

NPM

(PackageManager)

**NODEJS FRAMEWORK**

Installation Guide : --

LTS versions = long term support , which is open source Software

Better always work with LTS version, Not with the current version

Sample node Code

var http = require('http')

http.createServer(function(req,res){

res.send("Hello World....")

}).listen(8070)

Open command prompt

Node –v

Npm –v

Node –h

Node press enter will give us command Node prompt

The script fiels can be executed by using - node filename.js

Structure of Project

Root Folder

Index.html

App(or Js or scripts )

Css(or Styles)

Package.Json

Package.Json Structure - Explain the structure

Name

Version

Author

Entry

Description

Dependencies

Dev-dependencies

Package installations

Global Installation - Angular, react

Local Installation

1. Development Dependencies - Like installations of Vscode , VsStudio
2. Production Dependencies – Like Jquery , BootStrap

* Local to the current project

BootStrap, Jquery comes under a develeoper / production Dependency

* Generally production dependencies as locally

To move the project to another machine or folder , move it with out bower/node\_modules components

We can re-install the dependencies and packages required by using bower install command same as in npm

* Talk and Explain the Bower and NPM Commands both

To install the packages into a specified folder

{

"directory":"myBowerPacks"

"proxy":"http://192.168.100.1:8081"

"https-proxy":"http://192.168.100.1:8081"

}

* Better to keep default directories

**NodeJS**

It is a javascript Framework .

**I**

**I**

**S**

Asp File

DbFile

Xml File

Html File

Client 1

Client 2

**Web Server**

Request

Response

Node = http server which can handle request and response

Node is faster when compared to others

**General Server Working**

**Explanation :**

When a request send from the client , it will be received by the IIS Web server . Then IIS Server assign the request to a thread at threadPool. Let us consider T1 . So T1 will be processing for the response , where as if any blocking operation like DB Queries , Streaming and others , the thread T1 will be marked as busy even though the background work is going on. So unless the blocking work completed , T1 will be in busy state only . once the result got from the back ground work , T1 send it to IIS and IIS Sends it to the client as Response. T1 will be in Free state now

**NodeJs server working / Execution**

works with single threaded architecture with call back functions

V8 Execution

Engine C++

Event Loop

(Libev)

Thread pool

(libeio)

DNS

Crypto

JS Code (Client Code )

Node Libraries

Node binding(Http, Https Secure , ..)

Event loop recieves the request

Single threaded – can handle only one request at a time

* Asynchronus Non-Blocking IO Concept used
* Single threaded – Event Loop Architecture

Event Queue

Client 1

Client 2

Client 3

Files

Data Base

Audio/vedio Stream

**Event**

**Loop**

**NodeJS Server**

**Execution**

In Node Server the lets consider client1 requests for some audio file .

Request received by event loop , the event loop immediately assign it to a thread . let it be T1.

T1 will assign the task to the callback function , and the thread T1 will become free immediately , The callback function is responsible for getting the executed result ,

When the Response loaded into callback , then it will be assigned with whatever the thread is free, let us consider T3, then from T3 event loop will send the response to the client1.

Let us consider the scenario , at the time of getting the callback response , if no threads are free, then the callback will be in Event Queue and will be assigned to a thread immediately when its free .

So this is makes the threads not to be in waiting mode for some response . so utilizing the thread resources to the max Extent.

**Disadvantages**

* Asynchronous programming is not reliable
* As it working on single threaded , if that is down , all the server will be down ( so it is necessary to have Scaling )

**Call back Functions**

Call back functions will be called automatically after a long function executed

Generally used to catch the results

For function we can pass value, successcallback, failure callback

Function FunName(Param, SuccCallBack, FailureCallback)

{

PerformSome Logic and Call the callbackFunction();

}

Ex:

function printEvenOdd(num, callback)

{

if(i%2==0)

callback("Even", num);

else

callback("odd", num,"Odd should Allowed");

}

function myfirstcallback(msg, num, err)

{

if(err)

{

console.log("Error..");

return;

}

else

{

console.log(msg+ num);

}

}

var sodd=0, seven = 0;

function mySecondCallback(msg, num, err)

{

if(msg=="Even")

seven+= num;

else

sodd+=num;

}

for(var i =1; i <20; i++)

{

printEvenOdd(i,myfirstcallback);

printEvenOdd(i,mySecondCallback);

}

console.log("EVen Sum " + seven);

console.log("Odd Sum" + sodd);

**Multi CallBackFunctions Example**

function printnumbers(num, callbackone, callbacktwo)

{

if(num%2==0)

{

callbackone(num);

}

else

{

callbacktwo(num);

}

}

function callbacktwo(num, err)

{

if(err)

{

console.log("Error...");

return;

}

console.log("Passed Odd " + num);

}

function callbackone(num, err)

{

if(err)

{

console.log("Error...");

return;

}

console.log("Passed Even "+ num)

}

for(var i =1; i<20; i++)

{

printnumbers(i,callbackone, callbacktwo);

}

**Exception Handling :**

// Error Handling

var divide = function (num1, num2, callback) {

try

{

if (num2 === 0) {

throw new Error("Division Error..");

}

else {

var res = num1 / num2;

callback(res);

}

}

catch(err)

{

console.log("Error Occured : " + err)

}

}

var myCallBackFunc = function (data) {

console.log(data)

}

divide(2, 0, myCallBackFunc)

We can use 2 different callbacks fro success and error functions

// Error Handling

var divide = function (num1, num2, SCBFunc,ECBFunc) {

try

{

if (num2 === 0) {

throw new Error("Division Error..");

}

else {

var res = num1 / num2;

SCBFunc(res);

}

}

catch(err)

{

ECBFunc(err);

}

}

var successCallBackFunc = function (data) {

console.log(data)

}

var errorCallBackFunc = function (err) {

console.log(err)

}

divide(2,0, successCallBackFunc, errorCallBackFunc);

**Another Way of using only one callback function**

// Error Handling

var divide = function (num1, num2, callback) {

try

{

if (num2 === 0) {

throw new Error("Division Error..");

}

else {

var res = num1 / num2;

callback(res);

}

}

catch(err)

{

callback(undefined, err);

}

}

var callbackFunction = function(data, err)

{

if(err)

{

console.log("Erorr occured......");

}

else

{

console.log(data);

}

}

divide(2,0, callbackFunction);

**Working modules**

Module is a collection of functions

We can us module in another .js file by importing it

Where as first we need to export from module

Export syn at module file

Module.export = funcName;

Import Syn

Var xyz = require(“./filename.js”)

* What ever exported from module that modules can be collected into xyz

**Example**

**Export Module File**

var divide = function (num1, num2, callback) {

try

{

if (num2 === 0) {

throw new Error("Division Error..");

}

else {

var res = num1 / num2;

callback(res);

}

}

catch(err)

{

callback(undefined, err);

}

}

module.exports.divide= divide;

**Importing and using the Module**

var divide = require("./ModuleEx.js");

var callbackFunction = function(data, err)

{

if(err)

{

console.log("Erorr occured......");

}

else

{

console.log(data);

}

}

divide.divide(5,0,callbackFunction);

**Exporting a complete module at a time**

We can export the functions in an JSON Object Format

module.exports = {

divide: function (num1, num2, callback) {

try {

if (num2 === 0) {

throw new Error("Division Error..");

}

else {

var res = num1 / num2;

callback(res);

}

}

catch (err) {

callback(undefined, err);

}

}

,

add : function (num1, num2, callback) {

try {

if (num2 === 0) {

throw new Error("Add Error..");

}

else {

var res = num1 + num2;

callback(res);

}

}

catch (err) {

callback(undefined, err);

}

}

,

subt : function (num1, num2, callback) {

try {

if (num2 === 0) {

throw new Error("Subt Error..");

}

else {

var res = num1 - num2;

callback(res);

}

}

catch (err) {

callback(undefined, err);

}

}

}

**Consuming Ex:**

var calc = require("./ModuleEx.js");

var callbackFunction = function(data, err)

{

if(err)

{

console.log("Erorr occured......");

}

else

{

console.log(data);

}

}

calc.add(10,20,callbackFunction);

calc.divide(10,20,callbackFunction);

calc.subt(10,20,callbackFunction);

**NodeJS Globals**

**---------------------------**

Some variables or class available as globals for nodeJS

\_ \_ dirname

\_ \_ filename

Ex :

console.log(\_\_dirname);

console.log(\_\_filename);

global.authName = "Ramas";

global.companyName = "TSE";

console.log(global["authName"]);

console.log(global["companyName"]);

here we had added user defined globalvaribales and used them

**path : is a built in module come with nodeJS . used to work with path concept**

var path = require(“path”);

Explain some functions regarding path

Path.extname()

Path.basename()

Path.resolve()

Path.dirname()

Path.normalize()- removes the additional / and .. in the path

**Class Room activity : Explain how to Upload package into npmStore**

Npm adduser – to create new use r

Npm login – to login

Npm publish – to upload the packages

Create a folder named ramas

Include package.Json

Write the following code for Module .

The main file specified in package.json will get uploaded

Create a user

Login

Check the availability of Module

Open new Application

Include package.json

Npm install urPckage

specify it in .js file by using require

**Reading the file data**

we had provided with fs built in module . we need to use that

var fs = require("fs");

var path =require("path");

var fn = path.resolve(\_\_dirname,"./package.json");

console.log(fn);

fs.readFile(fn,"UTF-8",function(err, data){

if(err)

{

console.log("Error occured ...");

return;

}

else

{

//console.log(data);

var ob = JSON.parse(data);

console.log(ob.version);

console.log(ob.name);

}

})

* **Explain two more Examples – How use Open(), read()**
* One example showing Synchronous reading the data

**Writing the data to the File**

var fs = require("fs");

var path = require("path");

var msg = "Learning Node JS is evry easy .....";

var fn = path.resolve(\_\_dirname,"../mydata.txt");

fs.writeFile(fn,msg,"utf8", function(err){

if(err){

console.log("Cannt write the data....");

}

})

**Writing the file Synchronously**

var fs = require("fs");

var path = require("path");

var msg = "Learning Node JS is evry easy ....with Rama.";

var fn = path.resolve(\_\_dirname,"../mydata2.txt");

fh = fs.openSync(fn,"w");

var buf = new Buffer(msg);

if(!fh)

{

console.log("Cannot open the file ");

}

else

{

var wr = fs.writeSync(fh,buf,0,buf.length)

if(!wr)

{

console.log("Unable to write ..");

}

else

{

console.log("Saved succes...");

}

}

fs.closeSync(fh);

**Working with File Watcher**

Used to get notify when any modification made to file

Give an example of fs.exists() Function

Can use fs.watch() to get notify when the file got modified

Ex:

var fs = require("fs");

var path = require("path");

var msg = "Learning Node JS is evry easy ....with Rama.";

var fn = path.resolve(\_\_dirname,"../mydata2.txt");

fs.exists(fn,function(flag){

if(flag){

console.log("File is under Watching ...");

fs.watchFile(fn,{interval:5000},function(cur,pre){

console.log(cur.mtime);

console.log(cur.size);

})

}

})

* The exists method is going to deprecated in the upcoming feature ,Inspite of Exists we can use stat();

Stat() Ex:

var fs = require("fs");

var path = require("path");

var msg = "Learning Node JS is evry easy ....with Rama.";

var fn = path.resolve(\_\_dirname,"../mydata2.txt");

fs.stat(fn,function(err, stat){

if(stat){

console.log("File is under Watching ...");

console.log(stat);

}

})

**Working with Events**

**----------------------------------**

Events

--------

What is Event - we know

events - used to create the event in nodejs

Eventemitter – allows us to generate custom events . fires the event by assigning a method.

on() – used for registering the events

emit() – for firing the events

* **All the libraries at NodeJS uses eventEmitter , its is the base class for all the classes**

**steps**

\* create a events variable

\* create an object of eventEmitter using events variable

\* define the events and assign the handler methods to the event by using emitterObj.on

\* call emitter.emit() with event Name

Ex:

var ev = require('events');

var evem = new ev.EventEmitter();

// binding the event with method

evem.on("click", function(){

console.log("Button clicked...");

});

evem.on("click", function(){

console.log("Button2 clicked.........");

});

// firing the event

evem.emit("click");

**Ex2 :**

var ev = require('events');

class myclass extends ev{

}

var obj = new myclass();

obj.on("btnclick",()=>{console.log("Button Click Event called ...")});

obj.emit("btnclick");

**passing Parameters to events**

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

var ev = require('events');

class myclass extends ev{

}

var obj = new myclass();

obj.on("btnclick",(caller)=>{console.log("Button Click Event called by : " + caller)});

obj.emit("btnclick","Rama");

**Multiple parameters**

obj.on("btnclick",(caller,time)=>{console.log(`Button Click Event called by :${caller} at time : ${time} `)});

obj.emit("btnclick","Rama",new Date());

obj.emit("btnclick","Rama",new Date(2012,06,06));

**\* Discus An example for writing and reading with events , ie when write completed , write to console, same way when read completed , write to console**

**Solution**

// Working with events file reading and writing

var fs = require("fs");

var path = require("path");

var ev = require("events");

var eve = new ev.EventEmitter();

class filemngmt {

constructor(fileName) {

this.fileName = fileName;

}

myWrite(data, callBack) {

eve.emit("start", "write");

fs.writeFile(this.fileName, data, "utf8", function (err) {

if (err) {

eve.emit("end", "write", 0);

callBack(err);

return;

}

else {

console.log("successfully written into file.... ");

eve.emit("end", "write", 1);

callBack();

}

})

}

myRead(fn, callBack) {

fs.readFile(fn, "utf8", function (err, data) {

eve.emit("start", "Read");

if (err) {

eve.emit("end", "Read", 0);

callBack(err)

}

else {

callBack(undefined, data);

eve.emit("end", "Read", 1);

}

})

}

}

var fname = path.resolve(\_\_dirname, "data.txt");

var obj = new filemngmt(fname);

eve.on("start", function (taskname) {

console.log("Started " + taskname + " at : " + new Date());

});

eve.on("end", function (taskname, status) {

if (status === 1) {

console.log("Completed " + taskname + " at : " + new Date());

}

else {

console.log("Completed " + taskname + " at : " + new Date() + "with Errors...");

}

});

/\*

obj.myWrite("Hello world.....", function (err) {

if (err) {

console.log("Some error occred....");

}

});

\*/

obj.myRead(fname, function (err, data) {

if (err) {

console.log("Error...");

}

else {

console.log(data);

}

})

**Process is built in global Object available at node JS**

* **Gives the info abt current process**
* Title , Platform , version , env, env.LOGNAME

Process is the current executing program

Node is a Single process application

Process.on – beforeExit

Exit

uncaughtException are the predefined events of process class

code is 0 or 1 for success and error

process.on("beforeExit",function(code){

console.log("Code is going to exit ..");

})

process.on("exit", function(code){

console.log("Code is at exit ", code );

})

We can execute multiple process from a main process

To work with multiple process, we need to use child\_process Library

Child\_process .exec() - used to execute a command as a separate process

.execfile()

.spawn()

.fork()

Ex:

var prc = require("child\_process");

prc.exec("dir",function(err, stdout,stderr){

console.log(stdout);

} )

* Executes dir command of windows

Ex2 :

var prc = require("child\_process");

prc.exec("code e:/soft",function(err, stdout,stderr){

console.log(stdout);

} )

* Open visual studio code at e:/soft Folder

prc.exec("node index3.js",function(err, stdout,stderr){

console.log(stdout);

} )

* Will executes the file index3.js

We want execute some code when a process end or completed we can u se process method to exit

Spawn()

* Returns the data in stream format
* Data can be collected in stream format , where as exec returns the data in buffer format , ie after completely reading the data

Fork()

Creates an IPC between parent and child

Read and work an example of fork

Example : send message from parent to a child and get acknowledgement from child

**Working with HTTP :**

Import the http library

Use createServer(), listen()

Generally http servers run on 8080 port

We can use custom portnumber while developing

Port number > 1024 and < 65536

ftp – 21

http – 80

https – 443

all the port numbers below are called as swell known port numbers which were already allocated to some applications and protocols

status codes start with 2 – success

3 – redirection

4 – Client Error

5 – Server error ,

Request handler process must be attached to createServer(). A Request Handler is nothing but a function which contains mainly two arguments request and response

Ex:

var http = require('http')

var server = http.createServer();

server.on("request",(req,res)=>{res.end("Welcome to NodeJS....")})

server.listen(3004, function(err){

if(err)

{

console.log("Error in starting the server ..")

}

else{

console.log("Server started at : " + new Date());

}

})

Example 2

Index.js File

var http = require('http')

var reqhandler = require("./request-handler.js");

var server = http.createServer();

server.on("request",reqhandler)

server.listen(3004, function(err){

if(err)

{

console.log("Error in starting the server ..")

}

else{

console.log("Server started at : " + new Date());

}

})

Re-handler.js file

module.exports= (req, res)=> {

console.log(req.url);

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

{

res.end("<h1>NodeJS Learning....</h1>")

}

else if (req.url ==="/home.html"){

res.writeHead(200,{"content-type":"text/html"});

var html = `<html>

<title>NodeJS</title>

<head>

</head>

<body>

<h1>Welcome to Node JS Learning...</h1>

</body>

</html>`

res.end(html);

}

else if (req.url ==="/login.html"){

res.end("<h1>Login Page </h1>....");

}

else if (req.url ==="/register.html"){

res.end("<h1>Register Page </h1>....");

}

else

{

res.end("Invalid Requ....");

}

}

* If the content-type :application/html, browser will download the page
* Id for client side javascript , name for server accessing

url module is used for acquiring only the uri of the url

var url = require(‘url’);

can use url.parse(req.url);

will gives the complete info as JSON Object

We can access any properties of the JSON Object like pathname, query ,

Ex :

Var parsedurl = url.parse(req.url);

Parsedurl.pathname

Parsedurl.query

Parsedurl.

To get the query value

Var mydata = parsedurl.query.split(‘&’);

Mydata will be a array of keyvalue pair

Mydata [‘uname=rama’,’pwd=abc’,’submit=sub’]

Split the array using map to store the data in pairs

Var map = new map();

For (var item of mydata)

{

Var narray = item.split(‘=’);

Map.set(narray[0],narray[1]);

}

Res.end (map.get(“uname”);