**NodeJs Documentation**

**NodeJs (**[**https://www.tutorialspoint.com/nodejs/nodejs\_introduction.htm**](https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm%20) **)**

**NodeJS is introduced because** before nodejs javascript is only used inside browsers.

Node.js is a platform built on [Chrome's JavaScript runtime](https://code.google.com/p/v8/) for easily building fast and scalable network applications. 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.

**Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth. For example, a system is considered scalable if it is capable of increasing its total output under an increased load when resources are added**.

**Why NodeJs:**

* Non-blocking I/O model:

Here non-blocking means asynchronous

**NPM**

NPM, short for Node Package Manager, is two things: first and foremost, it is an online repository for the publishing of open-source Node.js projects; second, it is a command-line utility for interacting with said repository that aids in package installation, version management, and dependency management.

npm makes it easy for JavaScript developers to share and reuse code, and it makes it easy to update the code that you're sharing.

**ExpressJs**

Express.js, or simply Express, is a web application framework for Node.js, It is designed for building web applications and APIs.

const express = require('express')

const app = express()

app.get('/', function (req, res) {

res.send('Hello World!')

})

app.listen(3000, function () {

console.log('Example app listening on port 3000!')

})

Example:

var express = require('express');

var app = express();

var path = require("path");

var mongojs = require("mongojs");

var db = mongojs('catportal', ['user', 'trainerDetails', 'courseDetails', 'assessmentDetails']);

var bodyParser = require("body-parser");

var session = require('express-session');

app.set('trust proxy', 1) // trust first proxy

app.use(express.static(\_\_dirname + "/public")); //making public folder static

app.use(bodyParser.json());

**MEAN Stack**

The term MEAN stack refers to a collection of JavaScript based technologies used to develop web applications. MEAN is an acronym for MongoDB, ExpressJS, AngularJS and Node.js. From client to server to database, MEAN is full stack JavaScript.

**MongoDB**

MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas.

**What is Node.js? (**<https://www.w3schools.com/nodejs/nodejs_intro.asp>**)**

* Node.js is an open source server environment
* Node.js uses asynchronous programming!
* Node.js is free
* Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
* Node.js uses JavaScript on the server

**Why Node.js?**

Node.js uses asynchronous programming!

Example:

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:

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

Here is how Node.js handles a file request:

* Sends the task to the computer's file system.
* Ready to handle the next request.
* 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, asynchronously programming, which is very memory efficient.

* Several built-in modules. (fs, http, crypto, zip etc)
* Asynchronous APIs
* If built-in modules is not enough then we can add c++ addon packages to node. Because node has first-class support for C++ packages.
* Node has debugger utilities.
* NPM
* Module dependency manager.

**What is a Module in Node.js?**

Consider modules to be the same as JavaScript libraries. A module is file or a folder which contains set of functions.

**Node Wrapper:**

Every file in node is wrapped inside a function with five arguments:

**exports, module, require, \_\_filename, \_\_dirname**

inside node file:

console.log(arguments);

let g = 1; // the variable g defined here is not global.

is same as

function(exports, module, require, \_\_filename, \_\_dirname){

console.log(arguments);

let g =1;

}

And every built-in wrapping function returns :

module.exports;

**Note:** Always try to use module.exports instead of only exports because only exports is a alias to the module.exports.

**Node REPL : (Read Eval Print Loop)**

To enable it type Node in cli and press enter.

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

**Node Event Loop: (Node uses event lopp to handle slow I/O operations)**

This is what node uses to process asynchronous action and interface them for you so that you don’t have to deal with threads.

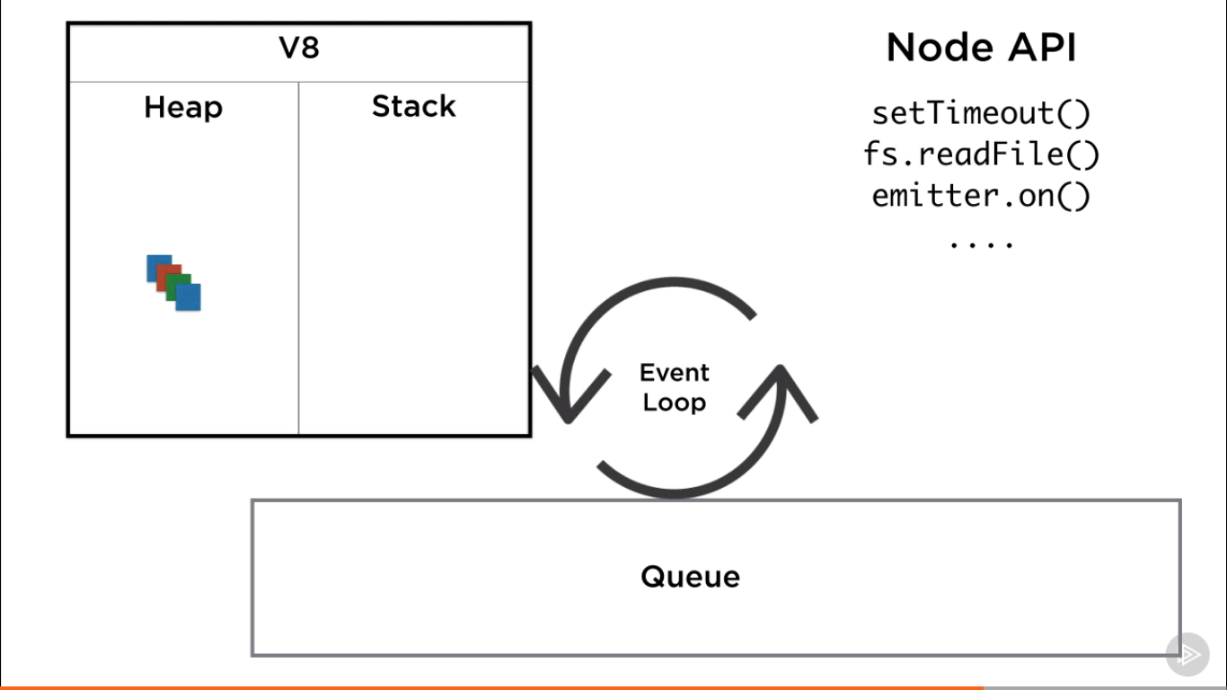
Or

This entity that handles external events and converts them into callback invocations.

Or

A loop that picks the events from the event queue and push it to the call stack.

Event loop automatically starts when node starts executing the script and exit the loop when there is no more callback to perform.



**The CALL STACK:**

* It follows first-in last out.
* Since Javascript is single threaded it has only one stack.

If we have two functions:

Var add = (a) => a+a;

Var printDouble = (a) => {

add(a);

console.log(a);

}

printDouble(9);

then in stack:

first: printDouble(); then add();

seconde: first add() will return then console.log() is pushed into stack called inside printDouble()

third: console.log() will return and then printDouble();

**How call backs works:**

**Queue:** It follows first in first out.

When the call stack is empty and queue is not empty then the event loop will dequeue one callback and then another push it back to call stack. And queue will not execute the callbacks unless the call stack is empty.

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

In Node any Async method execution will happen in two iterations:

For ex. Fs.readFile(\_\_dirname, function cb(err, data) {

Console.log(data);

})

* In first iteration node will call the readfile function and define the callback function and it will ask the operating system for the data(inside cb function) and when the OS is ready with data then
* 2nd iteration will start and execute the callback function

**Error vs Exception:**

**Error:** It is a problem.

**Exception:** It is a condition.

**Process.nextTick()**

**Settimeout is provided by node not v8.**

**url.parse(‘url’) => it transforms the url into object with url details(like query, pathname,hostname). And the query param is parssed if the 2nd argument is true.**

**url.format(urlObj) => it will return the complete url. It transforms the url parsed object into complete url.**

Node Architechture:

* Node has two main building blocks- V8 and libuv
* Node runs on V8 VM(virtual machine)
* See video inside video folder

**Fs Module:**

While initialization use readFileSync method but when reading file every time a user requests then use readFile method.

In readFileSync(), use try catch to handle error

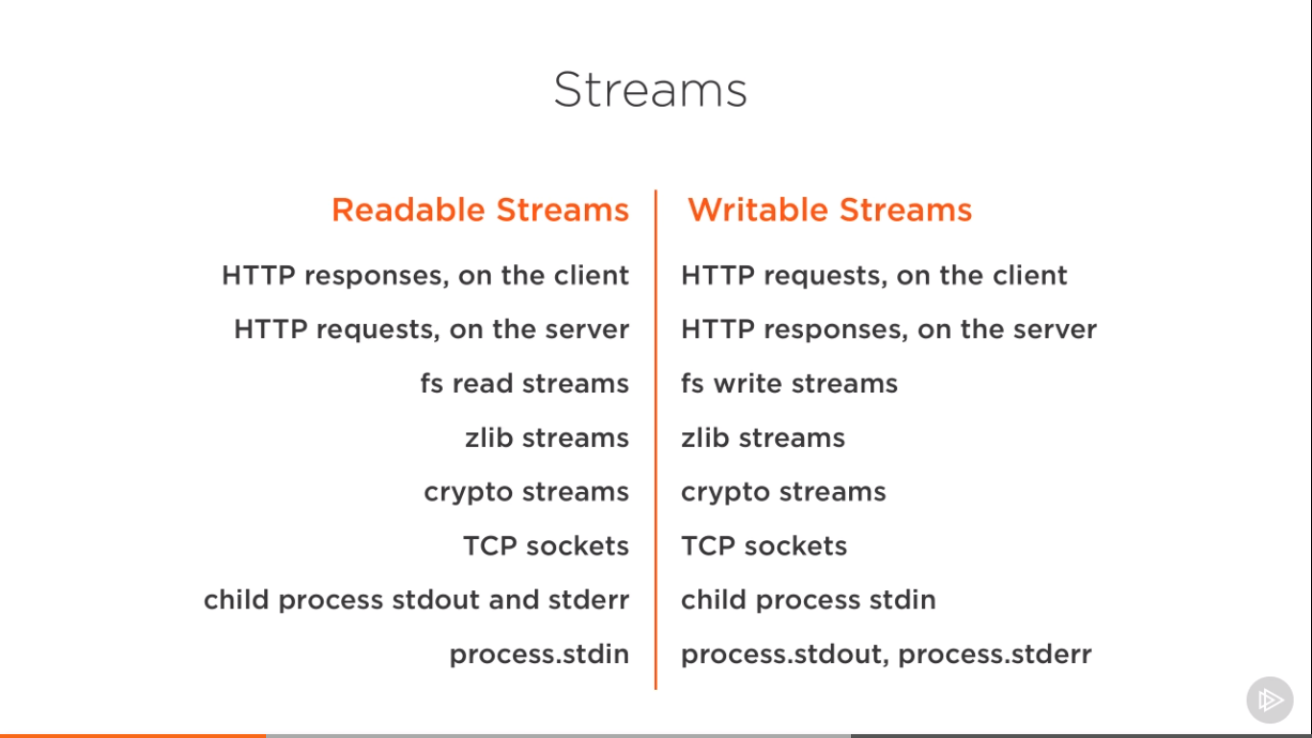
**To see the code please go through all files inside fs folder (mainly watchDir.js, deleteOldFiles.js, createFilesChangeTIme.js, truncate.js)**

**HTTP Module:**

**For ex. See switch.js**

**Streams: (**<https://medium.freecodecamp.org/node-js-streams-everything-you-need-to-know-c9141306be93>**)**

Working with big amount of data means working with stream.



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

**CLUSTERS AND CHILD PROCESSES: (**<https://medium.freecodecamp.org/node-js-child-processes-everything-you-need-to-know-e69498fe970a>**)**

Using multiple processes is the only way to scale node.js. and nodejs is highly scalable.

**Why scalability is needed?**

* To reduce workload
* Increase availability
* Increase Fault tolerance

**Scalability strtergies:**

* Cloning : easy and highly effective
* **Decomposing : refers to multiple different application with their own dedicated databases (mainly used with micro-services)**
* **Spliiting:** split the application into multiple instances and each instance is responsible for small portion of the application data. (also known as data sharding in DB)

**Create Child processes:**

Four Ways to create child processes:

* spawn()
* fork()
* exec()
* execFile()

**IMPORTANT QUESTIONS:**

* <https://edgecoders.com/how-well-do-you-know-node-js-36b1473c01c8>
* <https://asafdav2.github.io/2017/how-well-do-you-know-node-js-answers-part-1/>