**Some NodeJS core module**

**Path Module**

* In JavaScript we work with DOM, Browser, window and they are enclosed with browser only. But NodeJS run in machine and server. Where has Operating system, file, etc. so we can access the Operating system in NodeJS.
* Some core module name: crepto, os, file system, events, http, path: w3school for details
* Const path = require(‘path’);

Path.basename(mypath). It expect a path. It give base name means recent file name

const path = require("path");

const myPath =

  "C:/Users/User/OneDrive/Desktop/Nodejs/2. Node core module/practice code/index.js";

//path.basename(myPath); expect a path and return recent file base name

console.log(path.basename(myPath));

index.js

console.log(path.dirname(myPath));

C:/Users/User/OneDrive/Desktop/Nodejs/2. Node core module/practice code

console.log(path.extname(myPath));

.js

console.log(path.parse (myPath));

{

  root: 'C:/',

  dir: 'C:/Users/User/OneDrive/Desktop/Nodejs/2. Node core module/practice code',

  base: 'index.js',

  ext: '.js',

  name: 'index'

}

**OS Module**

* By using ‘OS’ module we can extract operating system information or server information. We can work with our operating system. It related to the server PC. Some of uses cases here:
* const os = require("os");
* console.log(os.platform());
* //win32
* console.log(os.homedir());
* //C:\Users\User
* console.log(os.freemem());
* // 2646953984
* // it gives the cpu information with core
* console.log(os.cpus());

[

  {

    model: 'Intel(R) Core(TM) i5-10400 CPU @ 2.90GHz',

    speed: 2904,

    times: {

      user: 946546,

      nice: 0,

      sys: 2828906,

      idle: 55126250,

      irq: 1766921

    }

  },

**File Module**

* we can work with file system in our system. File read, write, delete, update
* const fs = require("fs");
* //if we use writeFileSync, it write always as new. it doesn't append to existing files.
* // if we want to add new text to the file then we need to use appendFileSync method
* fs.writeFileSync("myFile.txt", "Hello file system how are you");
* // by default readFileSync give binary file binary buffer
* // need to convert to get orginal text using toString method
* const data = fs.readFileSync("myFile.txt");
* console.log(data.toString());

why we use ‘sync’ like readFileSync. Sync means synchronous. When we use sync it means we block the main thread. After finishing readFile synchronous then other work will be done.

* If we want to work with asynchronously then we need to use for file read like ‘readFile(‘filename’, callback);
* Callback has two parameter one is ‘err’ and other is ‘data’. At a time one of them will be null. Either err is null or data is null.

**Events Module**

NodeJS is event driven non-blocking runtime

* Event means Something has happened

First we need to create event emit. Then we need to implement that event

Event raise example: suppose when our period is over like 1hr period in high school life. When 1hr is finished then doftori raised the event.

So first we need to raise the event.

After raise event we say that like our class hour is finished. This is called event listener. Or event happened.

So secondly we need event run. Like what actually will be happened when event will be emit.

In generally in browser has many events like click, mouseover, etc. but in server we have not any button.

So we need to give a event name manually. Register a event. Must it has which type of event. And give a callback.

The type of event we manually add with emit raise.

const EventEmitter = require("events");

//here EventEmitter is a class

// we need to create EventEmitter instance variable

const emitter = new EventEmitter();

// here ‘bellring’ similar to click, submit. But

In this case this is custom. Own event.

// register a listener for 'bellring' event

emitter.on("bellring", (event) => {

  console.log("we need to run!");

});

// raise an event

emitter.emit("bellring");

Note: must do register event listener first then emit the event

* Register event
* Raise event

We can pass parameter to the raise event and can receive in the register event

// register a listener for 'bellring' event

emitter.on("bellring", (event) => {

  console.log("we need to run!, because ourr", event, "is oever");

});

// raise an event

emitter.emit("bellring", "First period");

**Note:** event instance must be same for the emit raise and event register listener

So better create a class then export this class and then create an instance of this class and use.

**Below is one example with two separate file:**

**Main.js**

const School = require("./other");

const school = new School();

school.on("bellRing", ({ period, text }) => {

  console.log(`we need to run becasue ${period} ${text}`);

});

school.startPeriod();

**otherFile.js**

const EventEmitter = require("events");

class School extends EventEmitter {

  startPeriod() {

    console.log("class started");

    // raise event when bell rings

    //raise an event

    setTimeout(() => {

      this.emit("bellRing", { period: "first", text: "period ended" });

    }, 2000);

  }

}

module.exports = School;

**http Module**

http module works with network where we need to transfer data

const http = require("http");

// here server is event emiter

const server = http.createServer((req, res) => {

  res.write("Hello programmers this is your first api called");

  res.write("this is second line");

  res.end();

});

//when a new connection is established means server is hitting

// then this "connection" event will be   fired

server.on("connection", () => {

  console.log("a new connection established");

});

// Note: rather doing connection events. we can pass a callback function to the createServer for each connection request for    each connection request. it will be fired when a new connection is established

// here is ther server register listener

server.listen(4000, () => {

  console.log("server listening on port ", 4000);

});

We can define path by using req.url===’/’, or req.url===’about’ etc then perform action inside if blocks