NodeJS Introduction

https://www.facebook.com/groups/nodejsmnnit

Ravi Shankar, 20124025
Mob: 7571872733
ravishankarkumar@live.com

What is node.js?

- Node.js is an open-source, cross-platform runtime environment for developing server-side applications.
- Node.js Applications are written in JavaScript
- Node.js apps run on hell lot of platforms including OS-X, Ubuntu, Windows, FreeBSD etc.
- Node.js is primarly used for web-apps, but beautiful desktop apps such as Atom and VSCode has been written indirectly using node.js technology
  - Refer to nw.js and electron if interested

Lets see few node libraries

crypto

```
var crypto = require("crypto");

var token =
crypto.randomBytes(48).toString("hex");

var link = "http://ravishankarkumar.com" +
"/reset/" + token;

console.log(link);


//passwordReset.js
```

passwordReset.js Explanation

```
var crypto = require("crypto");
```

just like #include in C, we have require in JavaScript to import libraries

```
var token = crypto.randomBytes(48).toString("hex");
```

In crypto library we have randomBytes method, so to call that method, we can do crypto.randomBytes()

In JacaScript, if we want to call function A() on the output of B() which is itself called on the output of C(), we can do this by C().B().A();//mChain.js

This is called method chaining

fs

- This is a node library for handling file system
- Create a file using the code:

```
var fs = require("fs");
fs.writeFile("rs.txt", "Rohit Shetty", function(err){
  if(err){
    console.log("oops... error occurred");
  } else {
    console.log("yay.... file created");
  }
});// createFile.js
```

Explaining fs.writeFile()

```
fs.writeFile("rs.txt", "Rohit Shetty", function(err){..});
```

- Here fs.writeFile() is taking 3 arguments:
- First Two are strings and third one is a functon.
- As you would already have noticed, this function has no name, these type of functions are called anonymous functions.
- This function is passed as a argument while calling another function, this is a very good example of callback mechanism.
- In callback mechanism, when you call a function, you give it a custom function. And you tell them that after you have performed your task, call this custom function.

Explaining fs.writeFile()

```
fs.writeFile("rs.txt", "Rohit Shetty", function(err){..});
```

- Here when the writeFile has successfully created a file named “rs.txt”, with content “Rohit Shetty”, it will call our function with the argument null.
- If the writeFile fails to create a file, then it will call our function with the error as an argument.
- As you have seen, in the body of our function, we are checking whether the error value is null or not.
- This was an example of callback mechanism.
- More on callback and asynchronous programming later.

delete a file

```
//this will give error if rs.txt doesn't exist
var fs = require("fs");
fs.unlink("rs.txt", function(err){
  if(err){
    console.log("error occurred");
  } else {
    console.log("Destruction is what i love!");
  }
});// deleteFile.js
```

rename a file

```
//this will give error if rs.txt doesn't exist
var fs = require("fs");
fs.rename("rs.txt", "boss.txt", function(err){
  if(err){
    console.log("error occurred");
  } else {
    console.log("Status upgraded!");
  }
});// renameFile.js
```

First Node Server

```
const http = require('http');

http.createServer( function(request, response){
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(8124);

console.log('Server running at http://127.0.0.1:8124/');
//myFirstServer.js
```

Explaining First node Server

createServer takes as an argument a request listener function.

This function automatically gets added to the request event, in case of a request.

request is an object that contains all the details of incoming request

response is an object that contains all the details of the response to be sent

we can set status code, content-type and data to be sent in response object

Run firstServer.js

- The app will listen on port 8124
- We can access the app on “localhost:8124” or “http://127.0.0.1:8124”

Activity

- Create a hotspot on your android phone
- Connect several laptops to this hotspot
- Change the hello world text in your server code to something unique, such as “Rohit is cool”, if your name is Rohit
- Find your ip address using ifconfig or ipconfig command, lets say it is “yourIpAddress”
- From the browser of your friend laptopn in same network, open “yourIpAddress:8124”, if you haven't changed the port
- Congratulate yourself for succesfully hosting a web-server!

Exercise

I want the “Googling is..” statement to be printed after 5 seconds, will the below program do my task?

```
setTimeout(function(){
    console.log(“Googling is a very important skill”);
}, 5000);
```

//setty.js

Solution

I want the “Googling is..” statement to be printed after 5 seconds, will the below program do my task?

```
setTimeout(function(){
    console.log(“googling is a very important skill”);
}, 5000);
```

yes, it does!

Exercise

I want the “Judge a ..” statement to be printed after 5 seconds, and again after 5 seconds the statement “And it wil...” to get printed. Will the below program do my task?

```
setTimeout(function(){
    console.log(“Judge a fish by its capacity to fly”);
}, 5000);

setTimeOut(function(){
    console.log(“And it will think forever, that it is useless”);
}, 5000);
//doubleSet.js
```

Solution

I want the “Judge a fish..” statement to be printed after 5 seconds, and again after 5 seconds the statement “And it wil...” to get printed. Will the below program do my task?

```
setTimeout(function(){
    console.log(“Judge a fish by its capacity to fly”);
}, 5000);
setTimeOut(function(){
    console.log(“And it will think forever, that it is useless”);
}, 5000);
```

//doubleSet.js

No, it doesn't. Why?

Why it failed?

- It failed because, Javascript doesn't wait for time taking process to complete.
- When a process performs i/o or lengthy calculation or unnecessarily waste time, JS start executing next process
- In the last example, as soon as the first setTimeout started waiting, JS started executing second setTimeout. This is the reason, both the statement printed almost simultaneously.

What is correct solution?

```
function iWaitYou(moreWait){
  setTimeout(function(){
    console.log("If you chase.");
    moreWait();
  }, 5000);
}

iWaitYou(function(){
  setTimeout(function(){
    console.log("They run away.");
  }, 5000);
}); // iWait.js
```

Thank you!