Node.js Basics

By : Rishabh & Sakshi

AGENDA

1. Introduction to Node.js

2. Setting up environment
    i. Installing Node
    ii. Writing your first “Hello World” in node
    iii. Node REPL
    iv. Building a web server in Node

3. Modules in Node.js
    i. Introduction to NPM
    ii. Understanding package.json
    iii. The “require” function

AGENDA Conti...

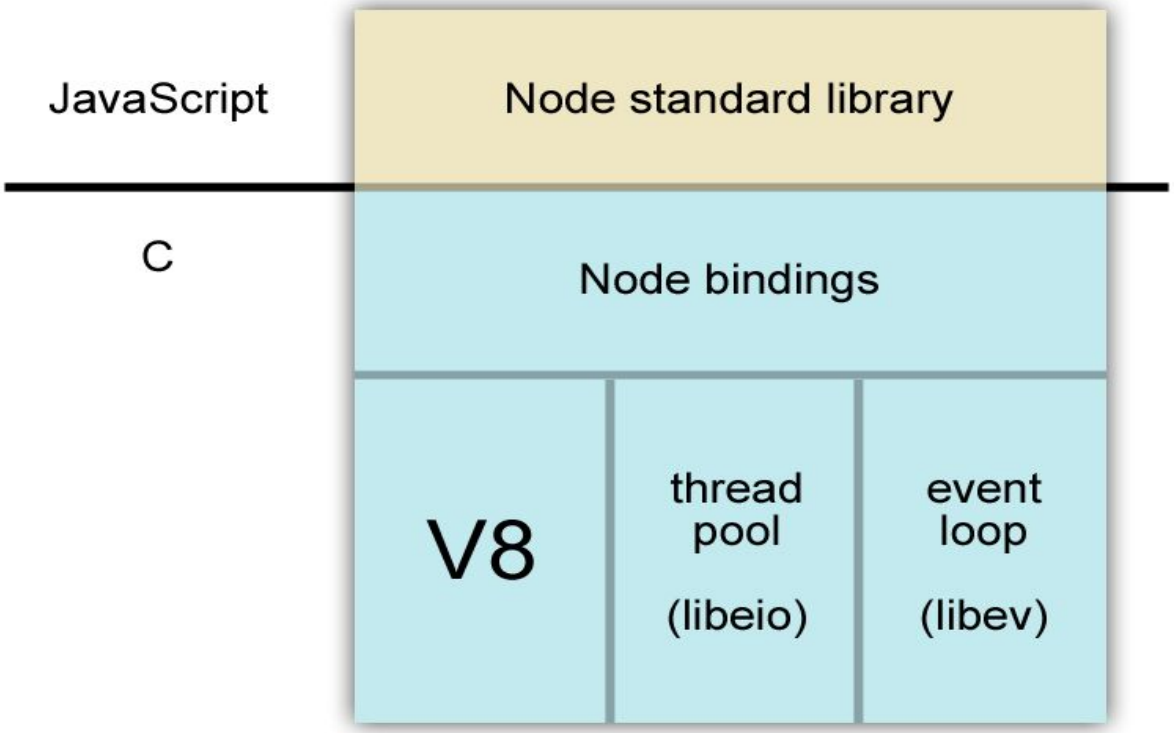
4. Node Fundamentals
    i. Callbacks
    ii. File System
    iii. Error Handling

5. Exercise

Introduction - What is Node.js?

- Node.js is a JavaScript runtime built on Chrome's V8 Javascript engine.
- Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.
- Node.js is single threaded.
- Node.js is an open source, cross-platform runtime environment for developing server-side applications.

Architecture Diagram



Setting Up Environment - Installing Node.js

https://www.tutorialspoint.com/nodejs/nodejs_environment_setup.htm

Using NVM
http://www.liquidweb.com/kb/how-to-install-nvm-node-version-manager-for-node-js-on-ubuntu-14-04-lts/
https://www.liquidweb.com/kb/how-to-install-node-js-via-nvm-node-version-manager-on-ubuntu-14-04-lts/

http://exponential.io/blog/install-nodejs-on-linux/

Setting Up Environment - Hello World Example

- Create a file having only console.log as below :

```
console.log('Hello World');
```

- Save this file as a javascript file i.e with .js extension like : demo.js

- Execute this demo.js file using node :

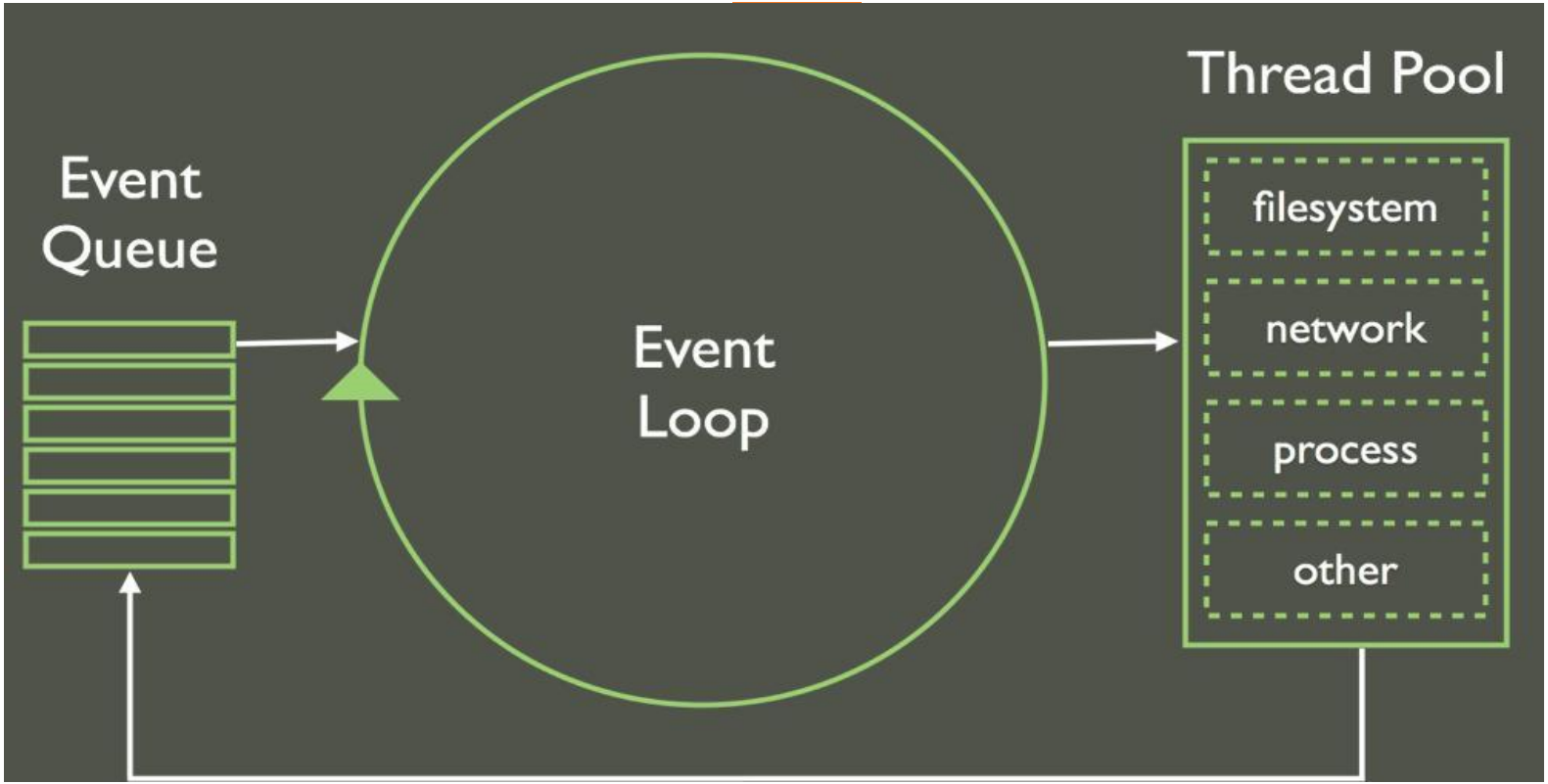
```
$ node demo.js
```

  This above command will print on console : Hello World

Event Loop

- Event loop is single-threaded .
- Event loops is core of javascript , all events , requests are handled by event loop.
- Works on event driver framework.
- Handle highly concurrent requests .
- Blocking the event loop can have catastrophic effects on the Node application.

Event Loop


Thread Pool
Event Queue
Event Loop
filesystem
network
process
other

Setting Up Environment - Node REPL

REPL stands for **Read Eval Print Loop** and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode.

Node.js comes bundled with a REPL environment. It performs the following tasks −

- **Read** − Reads user's input, parses the input into JavaScript data-structure, and stores in memory.
- **Eval** − Takes and evaluates the data structure.
- **Print** − Prints the result.
- **Loop** − Loops the above command until the user presses **ctrl-c** twice.

Setting Up Environment - Node REPL ...

- **REPL** can be started by simply running node on shell/console without any arguments as follows.

  Eg : $ node
  Output : >

  Now here you can write javascript/node code and see output on console.
  Eg: > 1+2
  Output: > 3

Setting Up Environment - Building a web server in Node

```
var http = require('http');
const PORT = 3000;

var server = http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World');
});

//Lets start our server
server.listen(PORT, function(){
    //Callback triggered when server is successfully listening.
    console.log("Server listening on: http://localhost:%s", PORT);
});
```

Modules in Node

1. **Modules** are the building blocks of a node application.

2. Separate our components based on business logic.

3. **Module** can be organized in a single file or in a directory containing one or more file which can then be reused within our Node.js application.

4. **Module Types:** Node.js includes three types of modules:
   a. **Core Modules**

   b. **Local Modules**

   c. **Third Party Modules**

Modules in Node : Core Modules

- The **core modules** load automatically when Node.js process starts. However, you need to import the core module first in order to use it in your application.
- In order to use Node.js core or NPM modules, you first need to import it using require() function as shown below :

```
var module = require('module_name');

eg: Load and Use Core http Module

var http = require('http');

var server = http.createServer(function(req, res){

  //write code here

});

server.listen(5000);
```

- In the above example, require() function returns an object because http module returns its functionality as an object, you can then use its properties and methods using dot notation e.g. http.createServer().

Modules in Node : 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.
- In Node.js, module should be placed in a separate JavaScript file. So, create a util.js file and write the following code in it.

util.js

```
function getMinutes(milliseconds) {

var min;

min = Math.round(milliseconds / 60000);

return min;

}

module.exports.getMinutes = getMinutes;
```

Modules in Node : Local Modules Cont …

- **Loading local module :** To use local modules in your application, you need to load it using require() function in the same way as core module. However, you need to specify the path of JavaScript file of the module.

  app.js

```
var util = require('./util.js');

var milliseconds = 60000;

console.log(milliseconds + 'milliseconds in min are : ', util.getMinutes(milliseconds));
```

- In the above example, app.js is using util module. First, it loads the util module using require() function from specified path where util module is stored. So, we have specified the path './util.js' in the require() function.

Modules in Node : Third Party Modules

- **Third Party Modules :** To use third party modules in your application, you need to load it using require() function in the same way as core module. However, first you need to install them using NPM. Like: `npm install <module-name>`

  Eg: **Express** is a third party module available on NPM which can be used as a web framework for creating API's in your node application. So you can use it in your node application as follows:

  ```
  var express = require('express');
  ```

  But before running your node application you should first install this third party module using npm install.

  ```
  $ npm install express
  ```

Modules in Node : Introduction to NPM

- **NPM** stands for **Node Package Manager** and is an online repositories for node.js packages/modules
- It also provides command line utility to install Node.js packages
- **Global vs Local Installation**
  - By **default**, NPM installs any dependency in the **local** mode. Here local mode refers to the package installation in **node_modules** directory lying in the folder where Node application is present.
  - **Global** install (with **-g**): puts stuff in /usr/local or wherever node is installed. Eg: `npm install -g <module-name>`
  - Install modules **locally** if you're going to **require()** it.
  - If you’re installing something that you want to use in your *shell*, on the command line or something, install it globally.

Modules in Node : Introduction to NPM Cont ...

- Use **npm ls** command to list down all the locally installed modules.
- To check globally installed modules use the following command :

```
$ npm ls -g
```

- **Uninstalling a module :** To uninstall a module use the following command

```
$ npm uninstall <module-name>
```

- **Updating a module :** To update a module us the following command

```
$ npm update <module-name>
```

Modules in Node : Understanding package.json

- The best way to manage locally installed npm packages is to create a **package.json** file.
- A **package.json** file offers you following things:
  - It serves as documentation for what packages your project depends on.
  - It allows you to specify the versions of a package that your project can use using.
  - Makes your build reproducable which means that its *way* easier to share with other developers.
- **Requirements** for creating a package.json : A package.json file must have
  - “name”
  - “version” (in the form of **x.x.x**)

Modules in Node : Understanding package.json cont...

Eg:

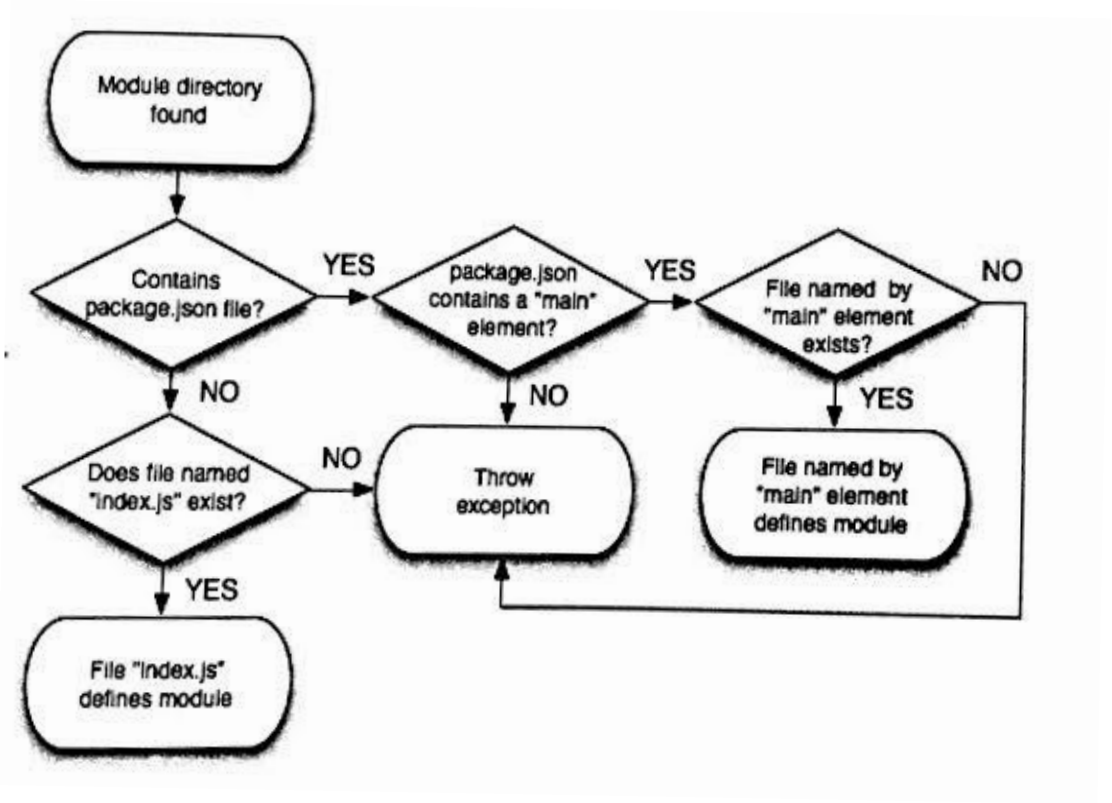
```
{
    “name”: “my-package”,
    “version”: “1.0.0”
}
```

- **Creating a package.json :**

  `> npm init` This command will prompt you questionnaire

  `> npm init --yes(or -y)` This command will create a package.json file with default configuration.

Modules in Node : the require() function



Node Fundamentals-Callbacks

- **Callbacks :** a function which is passed as an argument to a async function.
- It is main concept behind asynchronous programming.
- For example :

```
function myFun(name, cb){
    console.log("myfun");
    cb();
}

myFun("rishabh",function(){
    console.log('Callback called');
})
```

Node Fundamentals-Callbacks

- In Node file handling :

```
var fs=require('fs');
fs.readFile('filename',myFun);
function myFun(err, content){
    if(err)
    console.log(err);
    console.log(content);
}
```

Node Fundamentals-File handling

- Node File System (fs) module can be imported using following syntax:
- var fs = require("fs")
- Few methods of fs
  - fs.open(path, flags[, mode], callback) .
  - fs.stat(path, callback)
    - stats.isFile() Returns true if file type of a simple file.
    - stats.isDirectory() Returns true if file type of a directory.
    - stats.isBlockDevice() Returns true if file type of a block device.
    - stats.isCharacterDevice() Returns true if file type of a character device.
    - stats.isSymbolicLink() Returns true if file type of a symbolic link.
    - stats.isFIFO() Returns true if file type of a FIFO.
    - stats.isSocket() Returns true if file type of asocket.
- fs.writeFile(filename, data[, options], callback).
- fs.read(fd, buffer, offset, length, position, callback)
- fs.close(fd, callback)

Node Fundamentals-Error handling

- Error handling is a pain .
- Easy to get by for a long time in Node.js without dealing with many errors correctly .
- Building robust Node.js apps requires dealing properly with errors
  - Use **try catch** block: to handle errors.
  - **Throw** error : to throw customise error
- process.on('uncaughtException')-To handle unhandled exception which stops your node server.

Thank You