Node JS Training

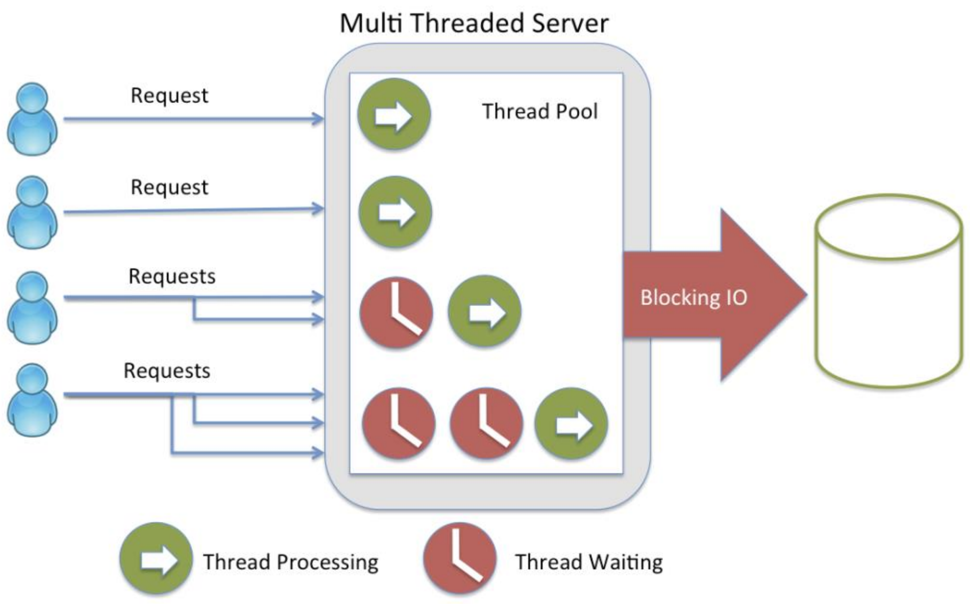
Node JS

- Server side runtime environment built on Chrome's V8 JavaScript engine.
- Event driven, non-blocking (asynchronous) I/O
- Environment for building highly scalable server-side application
- Node.js was written and introduced by Ryan Dahl in 2009

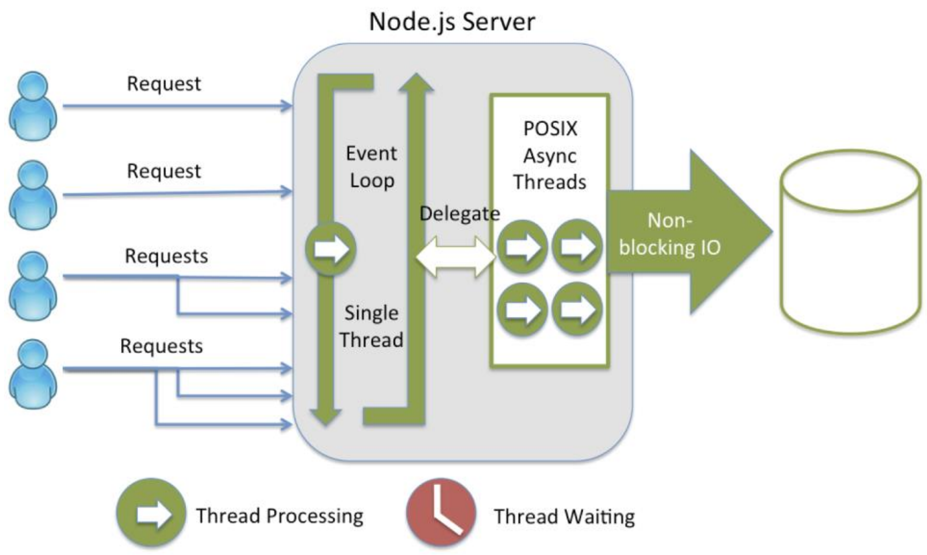
Advantages

- Uses JavaScript to build entire server side application.
- Lightweight framework that includes bare minimum modules.
- Modules can be included as per the need of an application.
- Asynchronous by default. So it performs faster than other frameworks.
- Cross-platform framework that runs on Windows, MAC or Linux

3

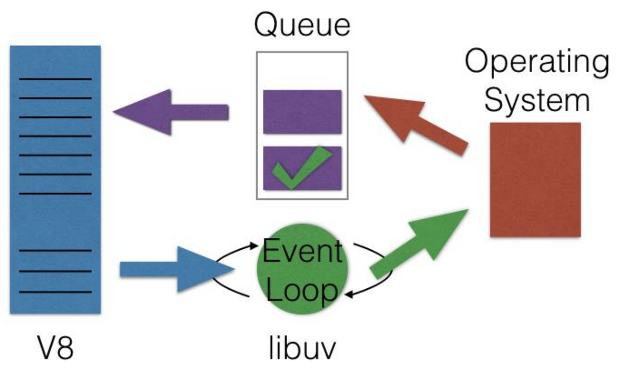


4



5

Node.js
Non-Blocking Event Driven I/O



6

- V8 runs JavaScript that requires an asynchronous task to be performed
- The libuv library submits a request to the OS to perform the task
- The task is placed onto a queue of tasks that will complete sometime in the future
- The event loop constantly checks to see if any tasks in the queue are complete
- Once the event loop finds a completed task it returns it to continue executing the corresponding JavaScript callback

Blocking

```
const fs = require('fs');
const data = fs.readFileSync('/file.md'); // blocks here until file is read
console.log(data);
// moreWork(); will run after console.log
```

Non Blocking

```
const fs = require('fs');
fs.readFile('/file.md', (err, data) => {
  if (err) throw err;
  console.log(data);
});
// moreWork(); will run before console.log
```

***Blocking** methods execute **synchronously** and **non-blocking** methods execute **asynchronously**.

Installation

- https://nodejs.org
- Download the MSI and double-click on it to start the installation
- >node –version
- https://github.com/nodejs/node-v0.x-archive/wiki/Installation

```
C:\>node
>
```

```
> "Hello" + "World"
Hello World
```

```
> 10 + 20
30
```

```
> "Hello" + "World"
Hello World
```

```
> var x = 10, y = 20;
> x + y
30
```

node-example.js

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

```
>node  node-example.js
Hello World
```

Primitive Types

- String
- Number
- Boolean
- Undefined
- Null
- RegExp

Process Object

- Each Node.js script runs in a process.
- It includes process object to get all the information about the current process of Node.js application.

```
> process.execPath
'C:\\Program Files\\nodejs\\node.exe'
> process.pid
1652
> process.cwd()
'C:\\'
```

Global Scope

- Global object represents the global scope.
- To add something in global scope, you need to export it using export or module.export.
- The same way, import modules/object using require() function to access it from the global scope.

Node.js Global Objects

- __dirname
- __filename
- console
- process
- buffer
- setImmediate(callback[, arg][, ...])
- setInterval(callback, delay[, arg][, ...])
- setTimeout(callback, delay[, arg][, ...])
- clearImmediate(immediateObject)
- clearInterval(intervalObject)
- clearTimeout(timeoutObject)

Node.js Module

- Encapsulating code in a separate logical unit.
- Functionality organized in single or multiple JavaScript files.
- Each module in Node.js has its own context
- Each module can be placed in a separate .js file under a separate folder

Types of modules

- Core Modules
- Local Modules
- Third Party Modules

Core

- http
- url
- querystring
- path
- fs
- Util

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

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

var server = http.createServer(function(req, res)
{
  //write code here
});

server.listen(5000);
```

Local Module

- Local modules are modules created locally in your Node.js application.
- Include different functionalities of your application in separate files and folders.
- You can also package it and distribute it via NPM, so that Node.js community can use it.

Simple Module – log.js

```
var log = {
         info: function (info) {
            console.log('Info: ' + info);
         },
         warning:function (warning) {
            console.log('Warning: ' + warning);
         },
         error:function (error) {
            console.log('Error: ' + error);
         }
   };

module.exports = log
```

Loading Local Module -app.js

```
var myLogModule = require('./log.js');

myLogModule.info('Node.js started');
```

Message.js

```
module.exports = 'Hello world';
//or
exports = 'Hello world';
```

app.js

```
var msg = require('./Messages.js');
console.log(msg);
```

```
C:\> node app.js
Hello World
```

Message.js

```
exports.SimpleMessage = 'Hello world';
//or
module.exports.SimpleMessage = 'Hello world';
```

app.js

```
var msg = require('./Messages.js');
console.log(msg.SimpleMessage);
```

```
C:\> node app.js
Hello World
```

Log.js

```
module.exports.log =
function (msg) {
    console.log(msg);
};
```

app.js

```
var msg = require('./Log.js');

msg.log('Hello World');

C:\> node app.js
Hello World
```

data.js

```
module.exports = {
    firstName: 'James',
    lastName: 'Bond'
}
```

app.js

```
var person = require('./data.js');
console.log(person.firstName + ' ' +
person.lastName);
```

23

Log.js

```
module.exports = function (msg) {
    console.log(msg);
};
```

app.js

```
var msg = require('./Log.js');

msg('Hello World');

C:\> node app.js
Hello World
```

24

Person.js

```
module.exports = function (firstName, lastName) {
   this.firstName = firstName;
   this.lastName = lastName;
   this.fullName = function () {
      return this.firstName + ' ' + this.lastName;
   }
}
```

app.js

```
var person = require('./Person.js');
var person1 = new person('James', 'Bond');
console.log(person1.fullName());
```

Node Package Manager

- Node Package Manager (NPM) is a command line tool that installs, updates or uninstalls Node.js packages in your application.
- It is also an online repository for open-source Node.js packages.
- The node community around the world creates useful modules and publishes them as packages in this repository.

Global & Local

- Global mode
  - NPM performs operations which affect all the Node.js applications on the computer

    > npm install -g express –save

- Local mode,
  - NPM performs operations for the particular local directory which affects an application in that directory only.

    > npm install express

Uninstall

- C:\>npm uninstall <package name>
- C:\MyApp> npm uninstall express

Command Line Arguments

- process.argv - array contains command line arguments that were passed when starting Node.js process.
- By default argument 0 is the path to node program and argument 1 is the path to the Node Java Script file.

```
process.argv.forEach((val, index) => {
  console.log(`${index}: ${val}`);
});
```

YARGS

- Yargs helps you build interactive command line tools, by parsing arguments and generating an elegant user interface.
- > npm i yargs --save

```
const yargs = require('yargs');
console.log(yargs.argv);
conole.log(yargs.argv._[0]);
```

```
var yargs = require('yargs')
console.log(yargs.argv)
console.log('__________________')
console.log(yargs.argv._)
console.log('__________________')
console.log(yargs.argv.func)
console.log(yargs.argv.n1 + yargs.argv.n1)
```

```
>node .\yargs.js --func add --n1 100 --n2 200
```

Request

- Request is designed to be the simplest way possible to make http calls.
- It supports HTTPS and follows redirects by default.

- npm install request –save

```
var request = require('request');
request('http://www.google.com', function (error, response, body) {
  console.log('error:', error); // Print the error if one occurred
  console.log('statusCode:', response && response.statusCode);
console.log('body:', body); // Print the HTML for the Google homepage.
});
```

Simple Web Server

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

// 1 - Import Node.js core module

var server = http.createServer(function (req, res) {
  // 2 - creating server

    //handle incomming requests here..

});

server.listen(5000); //3 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')
```

33

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

var server = http.createServer(function (req, res) {
if (req.url == '/') {
res.writeHead(200, { 'Content-Type': 'text/html' });
res.write('<html><body><p>This is homePage.</p></body></html>');
 res.end();
}
else if (req.url == "/student") {
 res.writeHead(200, { 'Content-Type': 'text/html' });
 res.write('<html><body><p>This is student Page.</p></body></html>');
 res.end();
  }
```

34

```
else if (req.url == "/admin") {
 res.writeHead(200, { 'Content-Type': 'text/html' });
 res.write('<html><body><p>This is admin Page.</p></body></html>');
 res.end();
    }
 else
res.end('Invalid Request!');
});
server.listen(5000); //6 - listen for any incoming requests
console.log('Node.js web server at port 5000 is running..')
```

JSON Response

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

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

   if (req.url == '/data') { //check the URL of the current request
         res.writeHead(200, { 'Content-Type': 'application/json' });
         res.write(JSON.stringify({ message: "Hello World"}));
         res.end();
   }
});

server.listen(5000);

console.log('Node.js web server at port 5000 is running..')
```

Node.js File System

- The fs module is responsible for all the asynchronous or synchronous file I/O operations.

- fs.readFile(fileName [,options], callback)
  - filename: Full path and name of the file as a string.
  - options: The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r".
  - callback: A function with two parameters err and fd. This will get called when readFile operation completes.

37

Read

- fs.readFile(fileName [,options], callback)

```
var fs = require('fs');
s.readFile('TestFile.txt','utf-8' ,function (err, data) {
    if (err) throw err;
    console.log(data);
});
```

TextFile.txt

This is test file to test fs module of Node.js

38

Read - Sync

```
var fs = require('fs');
var data = fs.readFileSync('dummyfile.txt', 'utf8');
console.log(data);
```

Write

fs.writeFile()

method to write data to a file.

fs.writeFile(filename, data[, options], callback)

```
var fs = require('fs');
fs.writeFile('test.txt', 'Hello World!', function (err) {
  if (err)
   console.log(err);
else
   console.log('Write operation complete.');
});
```

Stream

- Stream in Node.js simply means a sequence of data being moved from one point to the other over time.
- You have a huge amount of data to process, but you don’t need to wait for all the data to be available before you start processing it.

Buffer

- Buffer = “waiting area”
- Buffers are instances of the Buffer class in node, which is designed to handle raw binary data.
- It is a small physical location in your computer, usually in the RAM, where data are temporally gathered, wait, and are eventually sent out for processing during streaming.

Creating Buffer

- **var** buffer = **new** Buffer(8);
- **var** buffer = **new** Buffer([8, 6, 7, 5, 3, 0, 9]);
- **var** buffer = **new** Buffer("I'm a string!", "utf-8")
- buffer.toString('utf-8')

Node.js EventEmitter

- Create and handle custom events easily by using events module.
- Event module includes EventEmitter class which can be used to raise and handle custom events.

```
var events = require('events');
var em = new events.EventEmitter();

em.on('FirstEvent', function (data) {
   console.log('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', ‘Event emitter example.');
```

```
var emitter = require('events').EventEmitter;
var em = new emitter();
//Subscribe FirstEvent
em.addListener('FirstEvent', function (data) {
   console.log('First subscriber: ' + data);
});
//Subscribe SecondEvent
em.on('SecondEvent', function (data) {
   console.log('First subscriber: ' + data);
});
// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
// Raising SecondEvent
em.emit('SecondEvent', 'This is my second Node.js event emitter
    example.');
```

url Module

- The URL module splits up a web address into readable parts.
- Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties:

```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';
var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2017&month=february'

var qdata = q.query; //returns an object: { year: 2017, month: 'february' }
console.log(qdata.month); //returns 'february'
```

Database Access

- Node.js can be used in database applications.

- First need to install drivers for the database you want to use.

- npm install mssql
- npm install oracledb
- npm install mysql

Connection

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
```

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

   var con = mysql.createConnection({
    host: "localhost",
    user: "yourusername",
    password: "yourpassword",
    database: "mydb"
   });

   con.connect(function(err) {
    if (err) throw err;
    console.log("Connected!");
    var sql = "CREATE TABLE customers (name VARCHAR(255), address
   VARCHAR(255))";
    con.query(sql, function (err, result) {
     if (err) throw err;
     console.log("Table created");
    });
   });
```

```
con.connect(function(err) {
    if (err) throw err;
    console.log("Connected!");
    var sql = "INSERT INTO customers (name, address) VALUES
   ('Company Inc', 'Highway 37')";
    con.query(sql, function (err, result) {
     if (err) throw err;
     console.log("1 record inserted");
    });
   });
```

Database Types

- RDBMS (Relational Database Management System)
- OLAP (Online Analytical Processing)
- NoSQL (recently developed database)

MongoDB

- MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability.
- MongoDB works on concept of collection and document.

Terminology

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)

Database

- Database is a physical container for collections. Each database gets its own set of files on the file system.
- A single MongoDB server typically has multiple databases.

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
   likes: 100,
  comments: [
    {
      user:'user1',
      message: 'My first comment',
like: 0
    },
    {
      user:'user2',
      message: 'My second comments',
like: 5
    }  ] }
```

Start MongoDB

- C:\...>\mongod.exe -- dbpath d:\test\mongodb\data
- C:\..> mongo (run shell)

Database Commands

- >db   (current database)
- > show dbs (list databases)
- >use moviedb
- >db.movie.insert({name:'Independence Day',stars:'5'})
- >db.movie.find()
- >db.movie.find().limit(4);
- >db.movie.find({stars:'3'});
- >db.dropDatabase()

Create Collection

- db.createCollection(**name**, options)

- >use test
- >db.createCollection("students")
- >show collections

Note : MongoDB creates collections automatically when you insert some documents.

Drop Collection

- db.COLLECTION_NAME.**drop**()

- >use mydb
- > show collections
- >db.movies.drop()

Insert Documents

```
>db.collection.insert()

> db.products.insert( { item: "card", qty: 15 } )

>db.products.find()
```

Insert Multiple Documents

```
db.products.insert( [
{ _id: 11, item: "pencil", qty: 50, type: "no.2" },
{ item: "pen", qty: 20 }, { item: "eraser", qty: 25 }
] )
```

Find()

- **find()** method will display all the documents in a non-structured way.

- >db.mycol.find()
- >db.mycol.find().pretty()
- >db.mycol.find({},{"title":1,_id:0})

- db.mycol.find({"by":"john"}).pretty()
- db.mycol.find({"likes":{$lt:50}}).pretty()
- db.mycol.find({"likes":{$lte:50}}).pretty()
- db.mycol.find({"likes":{$gt:50}}).pretty()
- db.mycol.find({"likes":{$gte:50}}).pretty()
- db.mycol.find({"likes":{$ne:50}}).pretty()

- >db.mycol.find({$and:[{"by":"me"},{"title": "MongoDB Overview"}]}).pretty() {

- >db.mycol.find({$or:[{"by":"tp"},{"title": "MongoDB Overview"}]}).pretty()

Update Documents

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New
    MongoDB Tutorial'}})

>db.mycol.update({'title':'MongoDB Overview'}, {$set:{'title':'New
    MongoDB Tutorial'}},{multi:true})
```

Remove

- `>db.mycol.remove({'title':'MongoDB Overview'})`

Delete Document

- db.products.delete({ item: "card" })

Sort

- db.collection_name.find().sort({**KEY**:1})
- db.movies.find().sort({“name":1})
- is used for ascending order sorting.
- -1 is used for descending order sorting.

Express

- Express is a flexible Node.js web application framework
- Provides a robust set of features for web applications.
- Most popular *Node* web framework
- Easy to integrate with different template engines like Jade,hbs etc.

Installing

- $ mkdir myapp
- $ cd myapp
- $ npm init
- $ npm install express –save

Hello World!!

```
const express = require('express')
const app = express()

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(3000, () => console.log('Example app listening on port 3000!'))

//$ node app.js
//load http://localhost:3000/ in a browser to see the output.
```

Routing

- How an application responds to a client request to a particular endpoint.
- Each route can have handler functions, which are executed when the route is matched.
- Structure
  - app.METHOD(PATH, HANDLER)

```
app.get('/', function (req, res) {
res.send('Hello World!')
});

app.post('/', function (req, res) {
res.send('Got a POST request')
});
```

Middleware Functions

- An Express application is essentially a series of middleware function calls.
- Functions that have access to the request object (req), the response object (res), and the next function in the application's request-response cycle.
- Middleware are called in the order that they are declared.

Application-level middleware

```
var app = express()
app.use(function (req, res, next) {
  console.log('Time:', Date.now())
  next()
})

var myLogger = function (req, res, next) {
  console.log('LOGGED')
  next()
}
```

Serving Static Content

```
app.use(express.static('public'))


app.use(express.static('files'))
```

URL Parameters

- Set dynamically in a page's URL
  - /make/car/BMW
  - /course/java/121

- placeholder variable name (:)

- req.params

```
app.get("/car/make/:makeId", (req, res)
=> {
    console.log(req.params);


})
```

Route parameters

- Route parameters are named URL segments that are used to capture the values specified at their position in the URL.

- The captured values are populated in the req.params object

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }


app.get('/users/:userId/books/:bookId', function (req, res) {
  res.send(req.params)
})
```

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

app.get('/users/:userId/books/:bookId', function (req, res) {
res.send(req.params)
})

app.listen(3000,()=>{
console.log('server is ready.......')
});
//http://localhost:3000/users/ss/books/100
```

Multiple callback functions

```
app.get('/example/b', function (req, res, next) {
  console.log('the response will be sent by the next function ...')
  next()
}, function (req, res) {
  res.send('Hello from B!')
})
```

Response Methods

res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
res.sendFile()	Send a file as an octet stream.
res.sendStatus()	Set the response status code and send its string representation as the response body.

- res.redirect('/foo/bar');
- res.download('/report-12345.pdf');
- res.status(404).end();
- res.get('Content-Type');
- res.json({ user: 'tobi' });
- res.set('Content-Type', 'text/plain');

Chainable route handlers

```
app.route('/book')
  .get(function (req, res) {
    res.send('Get a random book')
  })
  .post(function (req, res) {
    res.send('Add a book')
  })
  .put(function (req, res) {
    res.send('Update the book')
  })
```

```
var cb0 = function (req, res, next) {
  console.log('CB0')
  next()
}

var cb1 = function (req, res, next) {
  console.log('CB1')
  next()
}

var cb2 = function (req, res) {
  res.send('Hello from C!')
}

app.get('/example/c', [cb0, cb1, cb2])
```

Template Engine

- A template engine enables you to use static template files in your application. (pug,mustache,ejs)
- At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client.
- This approach makes it easier to design an HTML page.
- $ npm install hbs --save

Application Properties

- app.set('view engine', ejs')
- app.set('views', './views') – default

Express Generator

- Express Generator is a Node JS Module.
- It is used to quick start and develop Express JS applications very easily.
- It does not come as part of Node JS Platform basic installation.

Express Generator

- Install
  - npm install express-generator –g
- Generate Application
  - express --view=ejs myapp
- install dependencies:
  - npm install