Node.js Tutorial

Node.js is an open source **server environment**.

**Node.js allows you to run JavaScript on the server.**

# Node.js Introduction

## What is Node.js?

* Node.js is an open source server environment
* 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!**

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:

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

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. 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.

## What Can Node.js Do?

* Node.js can generate dynamic page content
* Node.js can create, open, read, write, delete, and close files on the server
* Node.js can collect form data
* Node.js can add, delete, modify data in your database

## What is a Node.js File?

* Node.js files contain tasks that will be executed on certain events
* A typical event is someone trying to access a port on the server
* Node.js files must be initiated on the server before having any effect
* Node.js files have extension ".js"

# Node.js Get Started

## Download Node.js

The official Node.js website has installation instructions for Node.js: [https://nodejs.org](https://nodejs.org/)

Once you have downloaded and installed Node.js on your computer, let's try to display "Hello World" in a web browser.

Create a Node.js file named "myfirst.js", and add the following code:

var http = require('http');

http.createServer(function (req, res) {

    res.writeHead(200, { 'Content-Type': 'text/html' });

    res.end('Hello World!');

}).listen(8080);

## Command Line Interface

Node.js files must be initiated in the "Command Line Interface" program of your computer.

## Initiate the Node.js File

The file you have just created must be initiated by Node.js before any action can take place.

Start your command line interface, write node myfirst.js and hit enter:



# Node.js Modules

Consider modules to be the same as JavaScript libraries.

A set of functions you want to include in your application.

## Built-in Modules

Node.js has a set of built-in modules which you can use without any further installation.

## Include Modules

To include a module, use the require() function with the name of the module:

const http = require('http');

## Create Your Own Modules

You can create your own modules, and easily include them in your applications.

The following example creates a module that returns a date and time object:

exports.myDateTime = function () {

    return Date();

};

Use the exports keyword to make properties and methods available outside the module file.

## Include Your Own Module

var http = require('http');

var dt = require('./myfirstmodule');

http.createServer(function (req, res) {

  res.writeHead(200, {'Content-Type': 'text/html'});

  res.write("The date and time are currently: " + dt.myDateTime());

  res.end();

}).listen(8080);

Notice that we use ./ to locate the module, that means that the module is located in the same folder as the Node.js file.

# Node.js HTTP Module

## The Built-in HTTP Module

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

To include the HTTP module, use the require() method:

const http = require('http');

## Node.js as a Web Server

The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

Use the createServer() method to create an HTTP server:

var http = require('http');

*//create a server object:*

http.createServer(function (req, res) {

    res.write('Hello World!'); *//write a response to the client*

    res.end(); *//end the response*

}).listen(8080); *//the server object listens on port 8080*

The function passed into the http.createServer() method, will be executed when someone tries to access the computer on port 8080.

## Add an HTTP Header

If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:

var http = require('http');

http.createServer(function (req, res) {

**res.writeHead(200, {'Content-Type': 'text/html'});**

  res.write('<h1>Hello World!</h1>');

  res.end();

}).listen(8080);

The first argument of the res.writeHead() method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

## Read the Query String

The function passed into the http.createServer() has a req argument that represents the request from the client, as an object (http.IncomingMessage object).

var http = require('http');

http.createServer(function (req, res) {

  res.writeHead(200, {'Content-Type': 'text/html'});

  res.write(req.url);

  res.end();

}).listen(8080);

## Split the Query String

There are built-in modules to easily split the query string into readable parts, such as the URL module.

var http = require('http');

var url = require('url');

http.createServer(function (req, res) {

  res.writeHead(200, {'Content-Type': 'text/html'});

  var q = url.parse(req.url, true).query;

  var txt = q.year + " " + q.month;

  res.end(txt);

}).listen(8080);

<http://localhost:8080/?year=2017&month=July> //output: 2017 July

# Node.js File System Module

## Node.js as a File Server

The Node.js file system module allows you to work with the file system on your computer.

To include the File System module, use the require() method:

var fs = require('fs');

Common use for the File System module:

* Read files
* Create files
* Update files
* Delete files
* Rename files

## Read Files

The fs.readFile() method is used to read files on your computer.

Create a Node.js file that reads the HTML file, and return the content:

var http = require('http');

var fs = require('fs');

http.createServer(function (req, res) {

  fs.readFile('demofile1.html', function(err, data) {

    res.writeHead(200, {'Content-Type': 'text/html'});

    res.write(data);

    return res.end();

  });

}).listen(8080);

## Create Files

The File System module has methods for creating new files:

* fs.appendFile()
* fs.open()
* fs.writeFile()

The fs.appendFile() method appends specified content to a file. If the file does not exist, the file will be created:

var fs = require('fs');

fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {

    if (err) throw err;

    console.log('Saved!');

});

The fs.open() method takes a "flag" as the second argument, if the flag is "w" for "writing", the specified file is opened for writing. If the file does not exist, an empty file is created:

var fs = require('fs');

fs.open('mynewfile3.txt', 'w', function (err, file) {

    if (err) throw err;

    console.log('Saved!');

});

The fs.writeFile() method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created:

var fs = require('fs');

fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {

    if (err) throw err;

    console.log('Saved!');

});

## Update Files

The File System module has methods for updating files:

* fs.appendFile()
* fs.writeFile()

The fs.appendFile() method appends the specified content at the end of the specified file:

var fs = require('fs');

fs.appendFile('mynewfile1.txt', ' This is my text.', function (err) {

  if (err) throw err;

  console.log('Updated!');

});

The fs.writeFile() method replaces the specified file and content:

var fs = require('fs');

fs.writeFile('mynewfile3.txt', 'This is my text', function (err) {

    if (err) throw err;

    console.log('Replaced!');

});

## Delete Files

To delete a file with the File System module,  use the fs.unlink() method.

The fs.unlink() method deletes the specified file:

var fs = require('fs');

fs.unlink('mynewfile3.txt', function (err) {

    if (err) throw err;

    console.log('File deleted!');

});

## Rename Files

To rename a file with the File System module,  use the fs.rename() method.

The fs.rename() method renames the specified file:

var fs = require('fs');

fs.rename('mynewfile3.txt', 'myrenamedfile.txt', function (err) {

  if (err) throw err;

  console.log('File Renamed!');

});

# Node.js URL Module

## The Built-in 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'*

## Node.js File Server

Create a Node.js file that opens the requested file and returns the content to the client. If anything goes wrong, throw a 404 error:

var http = require('http');

var url = require('url');

var fs = require('fs');

http.createServer(function (req, res) {

    var q = url.parse(req.url, true);

    var filename = "." + q.pathname;

    fs.readFile(filename, function (err, data) {

        if (err) {

            res.writeHead(404, { 'Content-Type': 'text/html' });

            return res.end("404 Not Found");

        }

        res.writeHead(200, { 'Content-Type': 'text/html' });

        res.write(data);

        return res.end();

    });

}).listen(8080);

# Node.js NPM

NPM is a package manager for Node.js packages, or modules if you like.

## What is a Package?

A package in Node.js contains all the files you need for a module.

Modules are JavaScript libraries you can include in your project.

Downloading a package is very easy.

Open the command line interface and tell NPM to download the package you want.

Once the package is installed, it is ready to use.

Include the "upper-case" package the same way you include any other module:

var http = require('http');

var uc = require('upper-case');

http.createServer(function (req, res) {

    res.writeHead(200, { 'Content-Type': 'text/html' });

    res.write(uc.upperCase("Hello World!"));

    res.end();

}).listen(8080);

# Node.js Events

Node.js is perfect for event-driven applications.

## Events in Node.js

Every action on a computer is an event. Like when a connection is made or a file is opened.

Objects in Node.js can fire events, like the readStream object fires events when opening and closing a file:

var fs = require('fs');

var rs = fs.createReadStream('./demofile.txt');

rs.on('open', function () {

    console.log('The file is open');

});

## Events Module

Node.js has a built-in module, called "Events", where you can create-, fire-, and listen for- your own events.

To include the built-in Events module use the require() method. In addition, all event properties and methods are an instance of an EventEmitter object. To be able to access these properties and methods, create an EventEmitter object:

var events = require('events');

var eventEmitter = new events.EventEmitter();

## The EventEmitter Object

You can assign event handlers to your own events with the **EventEmitter** object.

In the example below we have created a function that will be executed when a "scream" event is fired.

To fire an event, use the emit() method.

var events = require('events');

var eventEmitter = new events.EventEmitter();

*//Create an event handler:*

var myEventHandler = function () {

    console.log('I hear a scream!');

}

*//Assign the event handler to an event:*

eventEmitter.on('scream', myEventHandler);

*//Fire the 'scream' event:*

eventEmitter.emit('scream');

# Node.js Upload Files

## The Formidable Module

There is a very good module for working with file uploads, called "Formidable".

The Formidable module can be downloaded and installed using NPM:

var formidable = require('formidable');

## Upload Files

## Step 1: Create an Upload Form

Create a Node.js file that writes an HTML form, with an upload field:

var http = require('http');

http.createServer(function (req, res) {

  res.writeHead(200, {'Content-Type': 'text/html'});

  res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');

  res.write('<input type="file" name="filetoupload"><br><br>');

  res.write('<input type="submit">');

  res.write('</form>');

  return res.end();

}).listen(8080);

## Step 2: Parse the Uploaded File

Include the Formidable module to be able to parse the uploaded file once it reaches the server.

When the file is uploaded and parsed, it gets placed on a temporary folder on your computer.

var http = require('http');

var formidable = require('formidable');

http.createServer(function (req, res) {

    if (req.url == '/fileupload') {

        var form = new formidable.IncomingForm();

        form.parse(req, function (err, fields, files) {

            res.write('File uploaded');

            res.end();

        });

    } else {

        res.writeHead(200, { 'Content-Type': 'text/html' });

        res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');

        res.write('<input type="file" name="filetoupload"><br>');

        res.write('<input type="submit">');

        res.write('</form>');

        return res.end();

    }

}).listen(8080);

## Step 3: Save the File

When a file is successfully uploaded to the server, it is placed on a temporary folder.

The path to this directory can be found in the "files" object, passed as the third argument in the parse() method's callback function.

var http = require('http');

var formidable = require('formidable');

var fs = require('fs');

http.createServer(function (req, res) {

    if (req.url == '/fileupload') {

        var form = new formidable.IncomingForm();

        form.parse(req, function (err, fields, files) {

            var oldpath = files.filetoupload.path;

            var newpath = 'C:/Users/abc/Desktop/Lockdown - notes/Notes(DOC)' + files.filetoupload.name;

            fs.rename(oldpath, newpath, function (err) {

                if (err) throw err;

                res.write('File uploaded and moved!');

                res.end();

            });

        });

    } else {

        res.writeHead(200, { 'Content-Type': 'text/html' });

        res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');

        res.write('<input type="file" name="filetoupload"><br>');

        res.write('<input type="submit">');

        res.write('</form>');

        return res.end();

    }

}).listen(8080);

# Node.js Send an Email

## The Nodemailer Module

The Nodemailer module makes it easy to send emails from your computer.

The Nodemailer module can be downloaded and installed using npm:

After you have downloaded the Nodemailer module, you can include the module in any application:

## Send an Email

Now you are ready to send emails from your server.

Use the username and password from your selected email provider to send an email. This tutorial will show you how to use your Gmail account to send an email:

var nodemailer = require('nodemailer');

var transporter = nodemailer.createTransport({

    service: 'gmail',

    auth: {

        user: 'youremail@gmail.com',

        pass: 'yourpassword'

    }

});

var mailOptions = {

    from: 'youremail@gmail.com',

    to: 'myfriend@yahoo.com',

    subject: 'Sending Email using Node.js',

    text: 'That was easy!'

};

transporter.sendMail(mailOptions, function (error, info) {

    if (error) {

        console.log(error);

    } else {

        console.log('Email sent: ' + info.response);

    }

});

## Multiple Receivers

To send an email to more than one receiver, add them to the "to" property of the mailOptions object, separated by commas:

var mailOptions = {

    from: 'youremail@gmail.com',

**to: 'myfriend@yahoo.com, myotherfriend@yahoo.com',**

    subject: 'Sending Email using Node.js',

    text: 'That was easy!'

}

## Send HTML

To send HTML formatted text in your email, use the "html" property instead of the "text" property:

var mailOptions = {

    from: 'youremail@gmail.com',

    to: 'myfriend@yahoo.com',

    subject: 'Sending Email using Node.js',

**html: '<h1>Welcome</h1><p>That was easy!</p>'**

}

# Node.js MySQL

Node.js can be used in database applications.

## MySQL Database

Once you have MySQL up and running on your computer, you can access it by using Node.js.

To access a MySQL database with Node.js, you need a MySQL driver. This tutorial will use the "mysql" module, downloaded from NPM.

C:\Users\abc*\***npm install mysql**

Now you have downloaded and installed a mysql database driver.

Node.js can use this module to manipulate the MySQL database:

var mysql = require('mysql');

## Create Connection

Use the username and password from your MySQL database.

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!");

});

## Query a Database

Use SQL statements to read from (or write to) a MySQL database. This is also called "to query" the database.

The connection object created in the example above, has a method for querying the database:

con.connect(function(err) {

  if (err) throw err;

  console.log("Connected!");

  con.query(sql, function (err, result) {

    if (err) throw err;

    console.log("Result: " + result);

  });

});

The query method takes an sql statements as a parameter and returns the result.

Node.js MySQL Create Database

## Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

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!");

  con.query("CREATE DATABASE mydb", function (err, result) {

    if (err) throw err;

    console.log("Database created");

  });

});