

Node JS : Introduction to Server-Side Javascript

Andrew Gall

November 9, 2012

Section 1

Introduction

My Experience

- Drexel University
- Worked for a start-up called CoupedOut for the past two years
- Work totally remotely

Your Experience

- Java?
- Python?
- Javascript?
- JQuery?

Follow Along

- Presentation: <http://housepage.org/nodejs.html>
- Code: <http://github.com/housepage/nodejs-tutorial>

Section 2

Javascript

What have you done?

- What is it used for?

What have you done?

- What is it used for?
- What have you built with it?

What is Javascript?

- Scripting Language

What is Javascript?

- Scripting Language
- Prototype-based

What is Javascript?

- Scripting Language
- Prototype-based
- Dynamic and weakly typed

What is Javascript?

- Scripting Language
- Prototype-based
- Dynamic and weakly typed
- First-class functions

Hello World

- Just writes to standard out
- File: helloworld.js

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

- Run It:

```
node helloworld.js
```

Section 3

Node.JS

What is Node JS?

- Server side software system designed for writing scalable Internet applications, notably web servers.
- Written in Javascript using event-driven, asynchronous I/O to minimize overhead and maximize scalability.

A Sample Application in Node JS

Let's keep it simple, but realistic:

- The user should be able to use our application with a web browser
- The user should see a welcome page when requesting `http://domain/start` which displays a file upload form
- By choosing an image file to upload and submitting the form, this image should then be uploaded to `http://domain/upload`, where it is displayed once the upload is finished

A Couple of Key Parts

- An HTTP Server - We want to serve web pages, therefore we need an HTTP server.
- A Router - Our server will need to answer differently to requests, depending on which URL the request was asking for, thus we need some kind of router in order to map requests to request handlers
- Request Handlers - To fulfill the requests that arrived at the server and have been routed using the router, we need actual request handlers
- Request Data Handling - The router probably should also treat any incoming POST data and give it to the request handlers in a convenient form, thus we need request data handling
- We not only want to handle requests for URLs, we also want

Basic HTTP Server

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

http.createServer(function(request, response) {
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello World");
    response.end();
}).listen(8888);
```

Break It Down

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

The first line requires the http module that ships with Node.js and makes it accessible through the variable http.

Break It Down

```
var server = http.createServer();  
server.listen(8888);
```

This part creates the server and makes it listen on port 8888. The server currently does nothing.

Break It Down

```
http.createServer(function(request, response) {  
    response.writeHead(200, {"Content-Type": "text/plain"});  
    response.write("Hello World");  
    response.end();  
}).listen(8888);
```

We pass a function into the server which it uses to handle all incoming requests. We make changes to the response object and then these are transmitted when we call:

```
response.end();
```

Section 4

Asynchronous Callbacks

Why do we need them?

```
var result = database.query("SELECT * FROM hugetable");  
console.log("Hello World");
```

- Database query might take a long time
- Node.JS runs in an event loop
- Important characteristics of how things work in Node.JS
- If you want to know more, check out Felix Geisendörfer's excellent post on [node.js](#)

It becomes this

```
database.query("SELECT * FROM hugetable", function(rows) {  
    var result = rows;  
});  
console.log("Hello World");
```

- Allows us to do other things while we wait for our results

Section 5

Organization

Organizing our code into modules

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

function start() {
  function onRequest(request, response) {
    /* ... */
  }

  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}

exports.start = start;
```

Using our new module

Now this file can be imported elsewhere and will expose that function or property. So we can do:

```
var server = require('./server');  
server.start()
```

Section 6

Routing

Making the Router

We start simply and just do a print in our router.

```
function route(pathname) {  
  console.log("About to route a request for " + pathname);  
}  
  
exports.route = route;
```

Wiring the Router In

Now, we add this routing to our HTTP Server:

```
function start(route) {  
  function onRequest(request, response) {  
    /* ... */  
  
    route(pathname);  
  
    response.writeHead(200, {"Content-Type": "text/plain"});  
    response.write("Hello World");  
    response.end();  
  }  
}
```

Final Piece

Importing our router and passing it into our HTTP Server:

```
var server = require("./server");  
var router = require("./router");  
  
server.start(router.route);
```

Now we have a basic router. It doesn't do anything but now we can write our request handlers.

Section 7

Request Handlers

Basic Request Handlers

```
function start() {  
  console.log("Request handler 'start' was called.");  
}  
  
function upload() {  
  console.log("Request handler 'upload' was called.");  
}  
  
exports.start = start;  
exports.upload = upload;  
  
requestHandlers.js
```

Wire This Up in App.js

```
var server = require("./server"); var router = require("./router");  
var requestHandlers = require("./requestHandlers");  
  
var handle = {} handle["/"] = requestHandlers.start;  
handle["/start"] = requestHandlers.start; handle["/upload"] =  
requestHandlers.upload;  
  
server.start(router.route, handle);
```

We create a dictionary where we defined what paths use what handlers.

Add support for this to our server

```
var http = require("http");
var url = require("url");

function start(route, handle) {
  function onRequest(request, response) {
    /* ... */

    route(handle, pathname);

    /* ... */
  }
}
```

Using That Information To Do Some Routing

```
function route(handle, pathname) {  
  console.log("About to route a request for " + pathname);  
  if (typeof handle[pathname] === 'function') {  
    handle[pathname]();  
  } else {  
    console.log("No request handler found for " + pathname);  
  }  
}  
  
exports.route = route;
```

We look in our dictionary of handlers to see which handler we should activate. If we find a valid path, we do something. If not, it

Heading over to the code to run this.

Let's do this.

Now, we have request handlers that are activated but let's actually do something with them.

Section 8

Returning Content

Passing in the Response object

```
function start(route, handle) {  
  function onRequest(request, response) {  
    /* ... */  
  
    route(handle, pathname, response);  
  
    /* We get rid of the code here that immediately responds  
  }  
  
  /* ... */  
}
```

We start by passing that response object into our routing function

Modifying our Routing function

```
function route(handle, pathname, response) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
    handle[pathname](response);
  } else {
    console.log("No request handler found for " + pathname);
    response.writeHead(404, {"Content-Type": "text/plain"});
    response.write("404 Not found");
    response.end();
  }
}
```

We pass that response on to the request handler and if there is

Finally Adding to the Request Handlers

```
var exec = require("child_process").exec;

function start(response) {
  console.log("Request handler 'start' was called.");

  exec("ls -lah", function (error, stdout, stderr) {
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write(stdout);
    response.end();
  });
}
```

Introduction
Javascript
Node.JS
Asynchronous Callbacks
Organization
Routing
Request Handlers
Returning Content
Conclusion

Let's see that work

Go!

Let's Make It Return A Simple HTML Page

Section 9

Conclusion

Introduction
Javascript
Node.JS
Asynchronous Callbacks
Organization
Routing
Request Handlers
Returning Content
Conclusion

Questions?

Links

- Presentation based on The Node Beginner Book
- Understanding node.js
- Node.JS Website
- Complete Source Code for Application