

Intro to Node and Node web servers

Michael Chang
Spring 2020

Overall course outline

HTML and CSS

Client-side JavaScript

Server-side JavaScript

Full stack topics

Plan for today

What is a web server?

Ports, servers, HTTP

What is NodeJS?

New libraries, functions

Packages and npm

Web servers in NodeJS

Responding manually

Using Express

Routing

Definitions

Port: a number from 0 to 65535

When connecting to a machine, specifies what application we want (web, mail, file transfer)

Server: a program that listens on a port

"Speaks" a certain protocol, i.e. expects to see certain kinds of messages

HTTP: a protocol for getting web resources and sending data

Methods, paths, headers, all that

HTTP (or "web") server

A server that listens on a port and responds to HTTP requests

Default port is 80 (insecure) or 443 (secure)

Web servers

Can be written in any language

As long as computer can run the program

Example: Python [http.server](#)

Demo: the "Michael types the response" server

Production servers often written in C or C++

Lots of systems and network code

Examples: Apache, nginx, lighttpd

NodeJS

NodeJS (or Node.js, or Node)

"Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine."

...Huh?

NodeJS

NodeJS (or Node.js, or Node)

"Node.js® is a **JavaScript runtime** built on **Chrome's V8 JavaScript engine**."

Chrome's V8 JavaScript engine

The code Chrome uses to interpret JS code

Parsing JS syntax, defining variables, calling functions

Does not include any "browser" stuff (DOM, user input)

JavaScript runtime

Libraries and APIs (in a general sense)

Rather than interact with web pages, these interact with the machine Node is running on

Start servers, read/write files, etc.

Node basics

Run node with no arguments (REPL)

Can type JS expressions, all the language stuff we've learned

Sort of like the browser console

node <script>

Runs a script file

const package = require("package");

Access an external library, such as a Node builtin library

Node's version of `import`

Node calls them modules, but these aren't the same as modules we've seen

Aside: "use strict"

"Sloppy mode"

Variables without let are global

Can overwrite some keywords

```
let undefined = 42; // ???
```

Old syntactic constructs (e.g. with)

Strict mode cleans up some of this

To activate, add "use strict"; (including quotes) at top of file

ES6 modules always use strict mode

Best practice: always add "use strict"; to every Node.js file

Node http module

```
"use strict";  
const http = require("http");  
let server = http.createServer();  
server.listen(1930, () => {  
    console.log("Listening on port 1930");  
});
```

http.createServer()

Create a new HTTP server

server.listen(port[, callback])

Have the server listen on port

If callback provided, call it when we start listening

Handle a request

```
server.on("request", (req, res) => {  
  console.log(`${req.method} ${req.url}`);  
  res.setHeader("Content-Type", "text/plain");  
  res.end("Here's your response!");  
});
```

obj.on(type, callback)

Register a callback for when event "emitted"

Like addEventListener from DOM

```
server.on("request", (req, res) => { ... });
```

Emitted when an HTTP request arrives

req has info about request, res lets us define response

External modules

Let's not reinvent the wheel

Surely someone has written a web server module

npm: Node package manager

Manages dependencies for our Node projects

Reads/writes a `package.json` file with project metadata

Command: `npm install <package>`

Install a package and make it a dependency of our project

Find packages in the [npm package registry](https://www.npmjs.com/)

Express

Express (npm)

A "web framework"--basically a wrapper around `http.Server`

Express API reference

```
const express = require("express");  
let app = express();  
app.listen(port, () => { ... });
```

`express([server])`

Create a new express "app"

If server not given, creates one

Can pass a server if some parts aren't managed by Express

Routing

We can define "routes" (paths) to respond to

Terminology: for APIs these are called "endpoints"

```
app.get("/file.txt", (req, res) => {  
  res.send("Here is your file.");  
});
```

app.get(path, callback)

Call our function when client requests path via GET

Also: `app.post`, `app.patch`, etc.

Also: `app.all`: response to all requests for path

res.send(body)

Like `res.end` from before, but does some extra stuff (like setting some headers)

Serving static files

`express.static(path)`

Returns a function(!) that handles a request by sending a file under path

`app.use([prefix,]callback)`

Call the function for any request that starts with prefix (or all requests if no prefix)

Example

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

Router parameters

Example

```
app.get("/api/:id", (req, res) => {  
  let id = req.params.id;  
});
```

:id is a route parameter

Handles all requests of the form "GET /api/____"

(Only one component, i.e. no slashes)

req.params

Object mapping parameter names to values

req.query

Object with key/values passed on query string

REST APIs in Express

res.json(value)

Return value as JSON

res.status(num)

Set HTTP status to num

Aside: returns res, so you can "chain" calls

Example

```
app.get("/api/:id", (req, res) => {  
  if (req.params.id === "secret")  
    res.json({ data: "Your secret data " });  
  else  
    res.status(404).json({ error: "Nope " });  
});
```

Gotchas

If you don't call `res.send/res.json`

- No response is sent

- The client will wait forever

If there's an error

- By default, Express sends the error backtrace to client

- ...But it won't be JSON

Summary

Today

Web servers, NodeJS, Express

Serving files, JSON

Route parameters and query string

Next time

What about request body?

Middleware, decomposition

Closing the loop: fetching from our API

Storing data