CS4220 Node.js & Vue.js

Cydney Auman
Albert Cervantes
CSULA

Node.js Server

Node Core Modules

HTTP/HTTPS

- Interfaces designed to support features of the http or https protocol. It provides functionality for running HTTP servers and making HTTP requests.

```
const http = require('http')
const https = require('https')
```

Node HTTP Server

A function passed as an argument to createServer and is called every time a client tries to connect to the server.

The `request` and `response` variables are objects representing the incoming and outgoing data.

```
const http = require("http")

const server = http.createServer((request, response) => {
  response.writeHead(200, { "Content-Type": "text/plain" })
  response.end("Hello World")
})


server.listen(8000, "localhost")
console.log("Server running at http://localhost:8000/")
```

Request

`request` is a request that comes from the client - this sometimes shortened to req.

The request argument contains information about the request, such as its url, http method, headers and etc.

```
http.createServer((request, response) => {
  console.log("Request URL: " + request.url)
  console.log("Request Method: " + request.method)
  console.log("Request Headers: " + JSON.stringify(request.headers))
})
```

Response

The `response` is the next argument in the function. Just like the prior argument is often shortened to res.

```
http.createServer((request, response) => {

  response.writeHead(200, {"Content-Type": "text/plain"})
  response.end("Server is running.")

})
```

With each response, you get the data ready to send, and then you call `response.end()`. Eventually, you ***must*** call this method. This method does the actual sending of data. If this method is not called, the server just hangs forever.

Node with express.js

Express.js describes itself as a "a minimal and flexible Node.js web application framework, providing a robust set of features for building single and multi-page, and hybrid web applications."

In short, it's a framework for building web applications with Node.js.

Node with express.js

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

app.get("/",(request, response) => {
  response.writeHead(200, { "Content-Type": "text/plain" })
  response.end("Hello World!")
})

app.listen(8080)
```

Middleware in express.js

Middleware consists of functions that are invoked by the express.js routing layer before the final request handler. It sits in the middle between a raw request and the final intended route. These functions have access to the request object (req) and the response object (res). Since they are always invoked in the order they are added, they are treated like a stack.

Middleware functions can perform the following tasks:

- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware function in the stack.

Middleware Examples in express.js

```
const
   bodyParser = require('body-parser'),
   express = require('express'),
   path = require('path')

const app = express()

// parse incoming request bodies before the handlers, available under req.body
app.use(bodyParser.json())

// built-in middleware to serve static files such as images, CSS, & JavaScript
app.use(express.static(path.join(__dirname, '..', '/client')))

app.listen(8080, () => {
   console.log('Server is running.')
})
```

References and Reading

HTTP Node Servers
-- http://eloquentjavascript.net/20_node.html

Express.js Middleware
-- https://expressjs.com/en/guide/using-middleware.html
-- https://expressjs.com/en/resources/middleware.html

Middleware Demystified
-- https://www.safaribooksonline.com/blog/2014/03/10/express-js-middleware-demystified