

## Chapter 19.5: node.js

CS 80: Internet Programming

Instructor: Mark Edmonds

### Overview

- Node (node.js) enables server-side JavaScript
- Not a programming language, but rather a server-side application that allows you to program server-side applications in JavaScript
- Much of what we did with PHP can be done with node.js
- Node has much more functionality than PHP; you can basically write whatever JavaScript program you want without needing to execute in a browser

### Overview

- Node is designed to make it easy to write I/O-based programs that run on a server
- I/O-based programs include web servers, databases, etc.

### Overview

- Node uses event-based asynchronous processing
- We will use same event-listener and callbacks we learned in JavaScript

### Installing Node

- Node can be installed from [nodejs.org](https://nodejs.org)
- Node is a command-line program, and you start node with by typing `node` at your terminal/command prompt

### Hello, world!

- Save the following in `hello_world.js`

```
1 console.log("Hello, world!");
```

- Launch the program with

```
1 node hello_world.js
```

**Example: http\_server.js**

```
1 // basic HTTP server with an export
2 var http = require("http"); // require the node HTTP module
3
4 function onRequest(request, response)
5 {
6   console.log("Request received.");
7   response.writeHead(200,
8     {
9       "Content-Type": "text/plain"
10    }); // set HTTP response header
11   response.write("Hello World"); // write content into HTTP request
12   response.end(); // finishes the response
13 }
14
15 http.createServer(onRequest).listen(8888);
16
17 console.log("Server has started.");
```

**Modules**

- We wrote `var http = require("http");` in the HTTP server example
- `http` is a module that our node application requires
- But we also want to write our own models
- This is accomplished using `exports`

**Example: http\_server\_export.js**

```
1 // basic HTTP server with an export
2 var http = require("http"); // require the node HTTP module
3
4 function start()
5 {
6   function onRequest(request, response)
7   {
8     console.log("Request received.");
9     response.writeHead(200,
10       {
11         "Content-Type": "text/plain"
12       }); // set HTTP response header
```

```
13     response.write("Hello World"); // write content into HTTP request
14     response.end(); // finishes the response
15 }
16
17 http.createServer(onRequest).listen(8888);
18 console.log("Server has started.");
19 }
20
21 exports.start = start;
```

### Example: index.js

```
1 var server = require("./http_server_export");
2
3 server.start();
```

## Modules

- Modules are a core component of node.js
- They allow you to modularize code
- This breaks our I/O-based application easier to manage and scalable
- Each module can be responsible for a specific kind of I/O

## Routing

- So far, every HTTP request was handled the same way
- Routing allows us to specify which modules process certain HTTP requests
- We'll look at the URL and the data in the GET/POST parameters and make a decision about where this HTTP request should be routed.

### Example: router.js

```
1 // create a router; this could also route to different applications (
    controllers) based on the file extension, or other information
2 // here, we just print the request we are routing
3 function route(pathname)
4 {
5     console.log("About to route a request for " + pathname);
```

```
6 }  
7  
8 exports.route = route;
```

### Example: http\_server\_router.js

```
1 var http = require("http"); // require the node HTTP module  
2 var url = require("url");  
3  
4 function start(route)  
5 {  
6   function onRequest(request, response)  
7   {  
8     var pathname = url.parse(request.url).pathname;  
9     console.log("Request for " + pathname + " received.");  
10  
11     route(pathname);  
12  
13     response.writeHead(200,  
14     {  
15       "Content-Type": "text/plain"  
16     }); // set HTTP response header  
17     response.write("Request for " + pathname + " received."); // write  
18       content into HTTP request  
19     response.end(); // finishes the response  
20   }  
21   http.createServer(onRequest).listen(8888);  
22   console.log("Server has started.");  
23 }  
24  
25 exports.start = start
```

### Example: index\_router.js

```
1 var server = require("./http_server_router");  
2 var router = require("./router");  
3  
4 server.start(router.route);
```

## Fileserver

- But we actually want our webserver to *do* something.
- Let's write a basic webserver, capable of serving files to the client (similar to our Python Simple-HTTPServer we used earlier)

### Example: `http_fileserver.js`

```
1 var http = require('http');
2 var url = require('url');
3 var fs = require('fs');
4
5 function start(route)
6 {
7   function onRequest(request, response)
8   {
9     // look for file in current directory
10    var filename = "." + url.parse(request.url).pathname;
11    console.log("Request for " + filename + " received.");
12    fs.readFile(filename, function(err, data)
13    {
14      // error getting the file, return a 404
15      if (err)
16      {
17        response.writeHead(404,
18        {
19          'Content-Type': 'text/html'
20        });
21        return response.end("404 Not Found");
22      }
23      // successfully retrieved file
24      response.writeHead(200,
25      {
26        'Content-Type': 'text/html'
27      });
28      // write the file's data into the response
29      response.write(data);
30      return response.end();
31    });
32  }
33  http.createServer(onRequest).listen(8888);
34  console.log("Server has started.");
```

```
35 }  
36  
37 exports.start = start
```

### Example: index\_fileserver.js

```
1 var server = require("./http_fileserver");  
2 var router = require("./router");  
3  
4 server.start(router.route);
```