Node.js Non-Tutorial

> *Node.js® is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices [source].*

This a lot of mumbo-jumbo for: node.js is a platform for supporting network applications where one process speaks to another (potentially on a different machine). It is conduit through which processes can talk to each other.

We will use it in this class as it provides us with a sturctured mechanism for initiating and closing connections, as well as passing messages. The abstraction layer allows us to focus on the functionality of the applications rather than the plumbing. Beyond this, it provides us with an engine that is truly cross-platform (as it uses a standard serialization mechanism called JSON). I also just think it's kind of cool.

Much of this learning is stuff you will need to do on your own. The purpose of this document is merely to guide your explorations.

There are three learning objectives as they relate to node.js.

1. Demonstrating competency in designing and developing real, working network applications.
2. Providing working experience with a new, cutting-edge platform for network applications that is rapidly becoming a defacto standard in web-based applications and platforms.
3. Messing around with two new programming languages (Javascript, C#), and understanding basic interop issues.

Note: Not everyone loves node.js. There are a number of really good critiques out there (e.g. that node code ends up being very long and undecipherable). For our purposes, it provides a nice platform for thinking about issues that are relevant to us, and is good (I think) for teaching these basic concepts. My suggestion: withold judgement until you make it through building some basic apps with node.js, and then try to rebuild them using other platforms (e.g. straight C#).

Installing node.js

1. Navigate to http://nodejs.org/
2. Press the "Install" link

This will install a few different binaries on your machine: `node` and `npm` are the most important of the two.

- `node` is the engine that powers everthing. You can run scripts by running `node myscript.js`, or enter the REPL straight up by typing in `node`.
- `npm` is the node package manager. It can be used to install packages or modules from off the web.

Some Simple Demos

The node.js main page has a couple of quick examples that you can try. For completeness, I'll reproduce them here.

Create a small webserver

1. Dump the following into a file: `example.js`

Notice: only five lines of code! And, one is basically a printf statement!

```
var http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

2. Open up a web browser, point it at `http://127.0.0.1:1337`

You'll see your web server say: "Hello World!"

What's happening here?

`http.createServer` function creates a server. The `.listen(1337, '127.0.0.1')` bit basically says, "Make the server listen to the interface 127.0.0.1 at port 1337." Remember that the interface is local network device (your laptop probably has at least three network devices--the local loopback device, the ethernet device, and the wifi device); the port is essentially the "mailbox number" that we care about.

The function inside the `createServer` bit is the function that is going to be called when the server gets hit. `req` refers to the request that is sent by the incoming connection, `res` refers to the response that is going to be returned.

Since we are creating a small webserver, node.js provides us with a few convenience functions: `writeHead` and `end`.

3. Add some debugging messages

Now open up your `example.js` file again, and add the following lines below the "Hello World" line:

```
console.log(req.url);
console.log(req.headers);
```

If you restart your server now and hit it with the web browser again, you'll see a bunch of text spew out onto the console. A conventional web browser will actually hit a server with two requests almost immediately -- one for the root document `/`, and one for `/favicon.ico`. This is why you see two requests on the server, even though you only hit the web server once yourself. The other thing that's being dumped back out is all the headers.

4. For kicks, connect via telnet

Open up a telnet session to pretend you're a client: `telnet 127.0.0.1 1337`

Once the session is open, type in: `GET / HTTP/1.0`, and press enter twice.

Take note of how the console dump looks different.

Exercises

As exercises, try doing the following:

- Implement a simple counter on the server. E.g. each time you hit the server, it reports, "Hi! *x* people have hit this server."
- Implement a simple mechanism that reports how much time the server has been up when you hit the server.
- Respond differently to requests to `/`, `/increment` and `/decrement`. `/` responds with the counter, and the other two URLs increment and decrement the counter respectively.

Additional Reading

- HTTP Made Really Easy

TCP Echo Server

You can also create an echo server that replies with anything that you send it. Dump the following into `echoserver.js`, and then connect to it using telnet.

```
var net = require('net');
var server = net.createServer(function (socket) {
    socket.write('Echo server\r\n');
    socket.pipe(socket);
});
server.listen(1337, '127.0.0.1');
```

Back out to the Real World

nodeschool.io has a lot of tutorials that we will use to familiarize ourselves with both Javascript and node.js. I've included time estimates (based on my going through the material) at the end of each line.

In order:

- `javascripting` (~30 mins)
- Art of Node: Read through to the end of (at least) Events (~10 mins)
- `learnyounode` (~5 hours)

If you are running this from OS X, you may need to install the packages using `sudo`, e.g. `sudo npm install -g javascripting`.

These tutorials vary in terms of depth and quality, but they will give you a basic primer on the information usually through a series of short text blurbs (colourfied!) and an exercise. Once you complete the exercise (as a small text file), you ask the tutorial to verify your exercise. I've found this to be a little tempermental, but basically it works.

You may get stuck on a few of the `learnyounode` tutorials (starting around exercise 7). Feel free to post on the D2L board when you get stuck.

Some additional references:

- Art of Node
- Using cURL to simulate a POST
- Javascript - string.toUpperCase()

Test your Knowledge

You should now be capable of writing some basic Javascript and node.js code. This is already pretty cool, and you're really starting to get dangerous.

For kicks, you should be able to do the following exercises:

- Create a simple HTTP server that actually serves up a set of html and image files (perhaps pointed to sample directory)

- Create a simple web crawler. Given a URL, it grabs the initial page, scrapes all the `href` mentions, and grabs all the files and image files that are referenced. (Once you have done this, you should be able to write a recursive version.)