Programming with NodeJS

Lesson 2: Callbacks, buffers, streams, APIs and Express

Febuary 14th, 2017

Summary of last week

Last week we:

Summary of last week

Last week we:

- We have installed NodeJS and NPM

Summary of last week

Last week we:

- We have installed NodeJS and NPM
- We have learnt about JS syntax

Summary of last week

Last week we:

- We have installed NodeJS and NPM
- We have learnt about JS syntax
- We have learnt about NodeJS and what it's useful for

Summary of last week

Last week we:

- We have installed NodeJS and NPM
- We have learnt about JS syntax
- We have learnt about NodeJS and what it's useful for
- We have made a very simple http server

Callbacks? Huh?

NodeJS is a language developed around the idea of everything being asynchronous.

Callbacks? Huh?

NodeJS is a language developed around the idea of everything being asynchronous.
Asynchronous programming is programming where an operation is executed after the previous one has been terminated.

Callbacks? Huh?

NodeJS is a language developed around the idea of everything being asynchronous.
Asynchronous programming is programming where an operation is executed after the previous one has been terminated.
How does asynchronous programming differ from normal programming?

Callbacks? Huh?

NodeJS is a language developed around the idea of everything being asynchronous.
Asynchronous programming is programming where an operation is executed after the previous one has been terminated.
How does asynchronous programming differ from normal programming?
Consider this example:

Asynchronous programming example

Say you really want to have a chat with your mate Paul about the weather. You have two ways to get in touch with him. You can either:

Asynchronous programming example

Say you really want to have a chat with your mate Paul about the weather. You have two ways to get in touch with him. You can either:

- Phone him

Asynchronous programming example

Say you really want to have a chat with your mate Paul about the weather. You have two ways to get in touch with him. You can either:

- Phone him
- Send him a letter

Both have distinct advantages and disadvantages.

Both have distinct advantages and disadvantages.
If you phone him and ask him what the weather is like, you have to sit there and wait for him to look outside and reply.

Both have distinct advantages and disadvantages.
If you phone him and ask him what the weather is like, you have to sit there and wait for him to look outside and reply.
This is considered synchronous. You have a guarantee that he will reply as soon as possible, but you have to wait the entire time he's looking outside.

Both have distinct advantages and disadvantages.
If you phone him and ask him what the weather is like, you have to sit there and wait for him to look outside and reply.
This is considered synchronous. You have a guarantee that he will reply as soon as possible, but you have to wait the entire time he's looking outside.
Alternatively, if you send him a letter asking what the weather is like, you can do other things while you wait for the postman to come with his response. However, you have no idea how long until you will recieve that response is.

Both have distinct advantages and disadvantages.
If you phone him and ask him what the weather is like, you have to sit there and wait for him to look outside and reply.
This is considered synchronous. You have a guarantee that he will reply as soon as possible, but you have to wait the entire time he's looking outside.
Alternatively, if you send him a letter asking what the weather is like, you can do other things while you wait for the postman to come with his response. However, you have no idea how long until you will recieve that response is.
This is considered asynchronous.

NodeJS is asynchronous.

NodeJS is asynchronous.
The reason for this is just the same as with Paul and the weather.

NodeJS is asynchronous.
The reason for this is just the same as with Paul and the weather.
If instead of talking to just Paul, you want to talk to 100s of people about the weather they are all experiencing, an asynchronous approach will allow you to keep conversations open with all of these people at the same time.

Seperating IO bound and CPU bound tasks

Note, this asynchronous feature is really strong for some purposes and really weak for others.
There are two types of functionality we will want in our program. They are:

Seperating IO bound and CPU bound tasks

Note, this asynchronous feature is really strong for some purposes and really weak for others.
There are two types of functionality we will want in our program. They are:

- CPU bound

Seperating IO bound and CPU bound tasks

Note, this asynchronous feature is really strong for some purposes and really weak for others.
There are two types of functionality we will want in our program. They are:

- CPU bound - stuff which the CPU has to do e.g. sorting an array, adding numbers together

Seperating IO bound and CPU bound tasks

Note, this asynchronous feature is really strong for some purposes and really weak for others.
There are two types of functionality we will want in our program. They are:

- CPU bound - stuff which the CPU has to do e.g. sorting an array, adding numbers together
- I/O bound

Seperating IO bound and CPU bound tasks

Note, this asynchronous feature is really strong for some purposes and really weak for others.
There are two types of functionality we will want in our program. They are:

- CPU bound - stuff which the CPU has to do e.g. sorting an array, adding numbers together
- I/O bound - stuff which the CPU tells something else to do, where it then gives it back to the CPU at another time e.g. opening a file (your hard drive takes time to retrieve the file), downloading something from the web (your network card takes time to download)

Seperating IO bound and CPU bound tasks

Note, this asynchronous feature is really strong for some purposes and really weak for others.
There are two types of functionality we will want in our program. They are:

- CPU bound - stuff which the CPU has to do e.g. sorting an array, adding numbers together
- I/O bound - stuff which the CPU tells something else to do, where it then gives it back to the CPU at another time e.g. opening a file (your hard drive takes time to retrieve the file), downloading something from the web (your network card takes time to download)

This distinction is important for NodeJS as it defines which bits NodeJS can do seemingly concurrently and which it won't be able to do well.

NodeJS and concurrency

Specifically, NodeJS is a single threaded program which uses an event loop to manage pseudo-concurrency.

NodeJS and concurrency

Specifically, NodeJS is a single threaded program which uses an event loop to manage pseudo-concurrency.
What this means is that, if NodeJS is doing a CPU bound task, the entire program has to stop and wait for the CPU task to be completed before **anything** can happen.

NodeJS and concurrency

Specifically, NodeJS is a single threaded program which uses an event loop to manage pseudo-concurrency.
What this means is that, if NodeJS is doing a CPU bound task, the entire program has to stop and wait for the CPU task to be completed before **anything** can happen.
If however NodeJS is executing an IO bound task, it can say "ok you do that IO, and once you've done the IO task, tell me and i'll execute this bit of code".

NodeJS and concurrency

Specifically, NodeJS is a single threaded program which uses an event loop to manage pseudo-concurrency.
What this means is that, if NodeJS is doing a CPU bound task, the entire program has to stop and wait for the CPU task to be completed before **anything** can happen.
If however NodeJS is executing an IO bound task, it can say "ok you do that IO, and once you've done the IO task, tell me and i'll execute this bit of code".
The "bit of code" is called a callback function and is a function which you give to an IO bound function, which is executed once the IO bound function is done.

Callbacks

```
var fs = require('fs');

fs.readFile('example.txt', function(err, data) {
  if(err) {
    console.log(err);
  } else {
    //Now we can do stuff with the data in the file
    console.log(data);
  }
});
```

Callbacks

```
var fs = require('fs');

fs.readFile('example.txt', function(err, data) {
  if(err) {
    console.log(err);
  } else {
    //Now we can do stuff with the data in the file
    console.log(data);
  }
});
```

Buffers

Buffers are a class in NodeJS which allows us to deal with big chunks of bytes.

Buffers

Buffers are a class in NodeJS which allows us to deal with big chunks of bytes.
Useful for times where you're getting a chunk of bytes in from somewhere e.g. downloading a file or opening a file from your file system

Buffers

You can make a new buffer with *var myBuffer = new Buffer(BUFFER SIZE);*.

Buffers

You can make a new buffer with *var myBuffer = new Buffer(BUFFER SIZE);*.
You can write to a buffer via *myBuffer.write(CONTENT);*.

Buffers

You can make a new buffer with *var myBuffer = new Buffer(BUFFER SIZE);*.
You can write to a buffer via *myBuffer.write(CONTENT);*.
Most usefully, you can convert a buffer to a string with *myBuffer.toString()*.

The Express framework

The Express framework is an HTTP server framework for NodeJS with a bunch of really powerful features, making it much stronger than the framework we used last week, 'http'.

Hello world in Express

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

app.get('/', function (req, res) {
  res.send('Hello World!')
});

app.listen(3000, function () {
  console.log('Listening on port 3000')
});
```

Hello world in Express

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

app.get('/', function (req, res) {
  res.send('Hello World!')
});

app.listen(3000, function () {
  console.log('Listening on port 3000')
});
```

Hello world in Express

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

app.get('/', function (req, res) {
  res.send('Hello World!')
});

app.listen(3000, function () {
  console.log('Listening on port 3000')
});
```

Hello world in Express

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

app.get('/', function (req, res) {
  res.send('Hello World!')
});

app.listen(3000, function () {
  console.log('Listening on port 3000')
});
```

Middleware

One of the strong points of Express is it's middleware capabilities.

Middleware

One of the strong points of Express is it's middleware capabilities.Middleware is code which is executed inbetween when you get the request and when you deal with it.

Middleware

One of the strong points of Express is it's middleware capabilities.Middleware is code which is executed inbetween when you get the request and when you deal with it.Some examples of middleware include adding authentication to every request, so only people who are logged in can go to certain places in the site.

Endpoints

An endpoint is a *destination*. When you go to 127.0.0.1:3000 you are going to the root endpoint /.

Endpoints

An endpoint is a *destination*. When you go to 127.0.0.1:3000 you are going to the root endpoint /.
If instead you go to 127.0.0.1:3000/hello/world, you are going to the endpoint */hello/world*.

Templates

Templates are pretty much HTML++.

Templates

Templates are pretty much HTML++.
They are pre-rendered on the server side and can be passed in values.

EJS

EJS is a templating language developed to be very strong when paired with javascript (like NodeJS). It mixes normal HTML with embedded javascript which is compiled before it is sent to the client. It looks a little like this:

```
<ul>
<% for(var i=0; i<supplies.length; i++) {%>
   <li><%= supplies[i] %></li>
<% } %>
</ul>
```

Putting it all together

That's all for tonight!

To summarise:

- We have learnt about callbacks and the NodeJS event loop
- We have learnt about buffers
- We have learnt about the Express framework
- We have made a complex server using Express

For next week

Source code plus lecture slides will be available online soon after the lesson.
If you are new to HackSocNotts, please join us on *http://hacksocnotts.slack.com*.
If you have any questions, feel free to ask now or over slack.