NodeJs Http

What Is HTTP?

Hypertext Transfer Protocol (HTTP) is long-standing protocol for communicating on the Internet. Essentially, it defines how a server and client should send and receive data while communicating.
You use HTTP every day when your web browser loads up web pages where your browser is the client connecting to the server of the website you are browsing.

Node.js allows you to create both servers and clients using a low-level application programming interface (API) with the HTTP module.

HTTP Servers with Node.js

Node.js excels at HTTP. It allows developers to create servers and clients with few lines of code.
Understanding the fundamentals of Node's HTTP module takes you a long way to understanding what Node.js can offer.

```
var http = require('http');
http.createServer(function (req, res) {
res.end('Hello world¥n');
}).listen(3000, "127.0.0.1");
console.log('Server running at http://127.0.0.1:3000/');
```

This example contains a lot of information about the HTTP module. Here is what's happening:

1. The HTTP module is required from the Node.js core and assigned to a variable so it can be used later in the script. This allows the script to access methods that enable using the HTTP protocol via Node.js.

2. A new web server object is created using `http.createServer` .

3. The script passes the web server an anonymous function telling the web server object what should happen every time it receives a request. In this case, when a request comes in, it should respond with the string 'Hello World' and then close the connection.

4. On line 4 of the script, the port and host for the web server are defined. This means the server can be accessed at http://127.0.0.1:3000.

5. The script logs a message to console of where the server can be accessed.

Adding Headers to the Server

```
var http = require('http');
http.createServer(function (req, res) {

res.writeHead(200, {
'Content-Type': 'text/plain'
});

res.end('Hello world\n');
}).listen(3000, "127.0.0.1");
console.log('Server running at http://127.0.0.1:3000/');
```

A Redirect in Node.js

Using the techniques learned in this hour, it is easy to create a simple server that redirects visitors to another web page.

The criteria for a redirect are as follows:

- Send the client a 301 response code, telling the client that the resource has moved to another location.

- Send a Location Header to tell the client where to redirect to. In this case, we direct visitors to Strong Bad's home page

```
var http = require('http');
http.createServer(function (req, res) {
res.writeHead(301, {
'Location': 'http://https://www.mycomp.com/'
});
res.end();
}).listen(3000, "127.0.0.1");
console.log('Server running at http://127.0.0.1:3000/');
```

Responding to Different Requests

Until this point, you have been creating simple HTTP servers with Node.js that return a single response. But, what if you want your server to respond to more than one type of request? In this scenario, you need to add routes to your application. Node.js makes this straightforward with the URL module. The URL module makes it possible to read a URL, parse it, and then do something with the output.

Routes Define Responses

Routing refers to the requests that your application will respond to. So, for example, if you want to show an About Us page at /about-us, you need to set up a route to respond to that request in your application.

Adding Routes to Your Server

```
var http = require('http'),
url = require('url');
http.createServer(function (req, res) {
var pathname = url.parse(req.url).pathname;
if (pathname === '/') {
res.writeHead(200, {
'Content-Type': 'text/plain'
});
res.end('Home Page\n')
} else if (pathname === '/about') {
res.writeHead(200, {
'Content-Type': 'text/plain'
});
res.end('About Us\n')
} else if (pathname === '/redirect') {
res.writeHead(301, {
'Location': '/'
});
res.end();

} else {
res.writeHead(404, {
'Content-Type': 'text/plain'
});
res.end('Page not found\n')
}
}).listen(3000, "127.0.0.1");
console.log('Server running at http://127.0.0.1:3000/');
```

HTTP Clients with Node.js

Although you can create HTTP servers with Node.js, it is also possible to create HTTP clients.

An HTML Client Is Not Always a Browser

HTML clients are anything that requests a response from a server. Examples of an HTML client include a web browser, a search engine robot, an email client, and a web scraper.

Some scenarios where you might want to use an HTML client include

- Monitoring the uptime of a server
- Scraping web content that isn't available by an API
- Creating a mashup that combines two or more sources of information from the web
- Making an API call to a popular web service, like Twitter or Flickr

An HTML Client in Node.js

```
var http = require('http');
var options = {
host: 'shapeshed.com',
port: 80,
path: '/'
};
http.get(options, function(res) {
if (res.statusCode == 200) {
console.log("The site is up!");
}
else {
console.log("The site is down!");
}
}).on('error', function(e) {
console.log("There was an error: " + e.message);
});
```