More on Modules

http, crypto, npm, request, and cheerio

Albert F. Cervantes
Cydney Auman
CS 4220 - CSULA

Node Modules

- Core Modules
  - Modules compiled into the node binary.
- Other Modules
  - Essentially, modules are JavaScript libraries.
  - 3rd-Party Libraries to achieve or simplify certain tasks.

AGENDA

- Create our own module
- Introduce Node Package Manager
- Review http - Core node.js module
  - request - Simplified http requests
  - cheerio - Web Scraping
- crypto - Core node.js module
  - Bitcoin and Hashing
  - Signing/Encrypting
  - Collisions and Identifying duplicate files

Custom Modules

Custom Modules

- Sometimes, we need additional functionality beyond the Core Modules.
- Custom Modules are created to provide new functionality and/or simplify existing functions.

The exports Keyword

- The exports keyword makes properties and methods available outside of the module
- In your custom module, simply assign properties and methods to exports.

```
1    exports.getWelcomeMessage = (name) => {
2        name = name ? name : 'Stranger'
3        return `Hello, ${name}!`
4    }
```

Include Your Module

- To include your module, you can require it like any other Core module
- However, be sure to specify the relative path to your module.

```
1   // Include our custom module
2   const mymodule = require('./my-module')
3
4   // Call an exported function from my custom module
5   console.log( mymodule.getWelcomeMessage('Albert') )
```

Node Package Manager

Node Package Manager

- The Node Package Manager, or NPM for short, “is the package manager for JavaScript and the world’s largest software registry”
- NPM provides a command-line interface (CLI) that allows us to add any package in the registry to our node projects.

http Review

http - Review

- The core *http* module can be used to:
  - Create a local server to service incoming requests
  - Submit http requests to other servers
- Documentation: https://nodejs.org/api/http.html

Creating a server with http

- In addition to sending requests, http can be used for handling incoming HTTP requests

```
1   // Include http to create a new web server
2   var http = require('http');
3
4   // Include our custom module
5   const mymodule = require('./my-module')
6
7   http.createServer(function (request, response) {
8
9       // Extract Query String parameters from the request
10      const params = mymodule.getParameters(request.url)
11
12      // Set the content type in the header
13      response.writeHead(200, {'Content-Type': 'text/html'})
14
15      // Write the welcome message
16      response.write( mymodule.getWelcomeMessage(params.name) )
17
18      // Conclude our response
19      response.end();
20  }).listen(8080);
```

Submitting an HTTP Request

http.request()

- Require *http* module
- Create *options* object
- Submit http request
- Read status
- Read headers

```
1   const http = require('http')
2
3   const options = {
4       hostname: 'albertcervantes.com',
5       port: 80,
6       path: '/hello.html',
7       method: 'GET',
8       headers: {
9           'Content-Type': 'text/html'
10      }
11  };
12
13  const req = http.request(options, (res) => {
14      console.log(`STATUS: ${res.statusCode}`);
15      console.log(`HEADERS: ${JSON.stringify(res.headers)}`);
16      res.setEncoding('utf8');
```

http.request()

- Read response body *in chunks*
- Handle errors
- Call .end()

```
17        res.on('data', (chunk) => {
18            console.log(`BODY: ${chunk}`);
19        });
20        res.on('end', () => {
21            console.log('No more data in response.');
22        });
23    });
24
25    req.on('error', (e) => {
26        console.log(`problem with request: ${e.message}`);
27    });
28
29    req.end();
```

One must always call *req.end()* to signify that you're done with the request - even if there is no data being written to the request body.

.request vs .get

http - .get()

- Connect to external server
- Request *hello.html*
- Handle errors

```
1   const http = require('http')
2
3   http.get('http://albertcervantes.com/hello.html', (res) => {
4
5       // Read some information about the response
6       const statusCode = res.statusCode;
7       const contentType = res.headers['content-type'];
8
9       let error;
10      if (statusCode !== 200) {
11          error = new Error(`Request Failed.\n` +
12              `Status Code: ${statusCode}`);
13      }
14      if (error) {
15          console.log(error.message);
16
17          // consume response data to free up memory
18          res.resume();
19
20          return;
21      }
```

http - .get()

- Read response
- Display on console

```
22
23        // Read the contents of the response in chunks
24        res.setEncoding('utf8');
25        let html = '';
26        res.on('data', (chunk) => html += chunk);
27
28        res.on('end', () => {
29            console.log(html)
30        });
31
32    }).on('error', (e) => {
33        console.log(`Got error: ${e.message}`);
34    });
```

Since most requests are GET requests without bodies, Node.js provides this convenience method.

The only difference between this method and **http.request()** is that it sets the method to GET and calls req.end() automatically.

There has to be an easier way!

Every Developer

Request

- The *request* module is a “Simplified HTTP request client.”
- Github: https://github.com/request/request
- Install: `npm install request`
- Our goal is to simplify making http requests, submitting form-data, and processing results.

Module: Request

- Require *request* module
- Make request
- *Error*(s), *Response*, and *Body* handled automatically

```
1   var request = require('request');
2   request('http://albertcervantes.com/hello.html', (error, response, body) => {
3       // Print the error if one occurred
4       console.log('error:', error);
5
6       // Print the response status code if a response was received
7       console.log('statusCode:', response && response.statusCode);
8
9       // Print the HTML for the hello.html doc
10      console.log('body:', body);
11  });
```

Web Scraping

Web Scraping

- Our goal is to work with HTML on the server in the same way we would work with it in the browser.
  - jQuery is a typical client-side library used for this purpose
    - http://jquery.com
- How do we interrogate the HTML that is returned from an HTTP request?
  - In the browser we rely on the Document Object Model (DOM) to ask questions about the structure of the document.
  - Node.js does not provide a DOM to interrogate, and many off-the-shelf client-side libraries fail when you try to use them in node.

Teach your server HTML.

Cheerio.js

Cheerio

- The cheerio module is a “[f]ast, flexible, and lean implementation of core jQuery designed specifically for the server”.
- Github: https://cheerio.js.org
- Install: `npm install cheerio`

Cheerio

- Require *cheerio* module
- Load the *HTML* into cheerio
  - Assign result to variable, typically *$*
- Use *$* like jQuery

```
1   const cheerio = require('cheerio'),
2       $ = cheerio.load('<h2 class = "title">Hello world</h2>');
3
4   console.log('Before manipulation: ', $.html());
5
6   $('h2.title').text('Hello there!');
7   $('h2').addClass('welcome');
8
9   console.log('After manipulation: ', $.html());
```

Cheerio

- Answer the following questions using cheerio and the following html document: http://albertcervantes.com/hello.html
  - How many paragraphs are present on the page?
  - How many images?
  - What are the URLs of all images on the page?
  - What is the average length of all paragraphs?
  - What is the average word count of all paragraphs?

Hashing & Signing

A hash function is any function that can be used to map data of arbitrary size to data of fixed size.

Wikipedia

Hashing

- SHA256
  - The SHA (Secure Hash Algorithm) is one of a number of cryptographic hash functions.
  - A cryptographic hash is like a signature for a text or a data file.
  - SHA-256 algorithm generates an almost-unique, fixed size 256-bit (32-byte) hash.
  - Hash is a one way function – it cannot be decrypted back.
  - This makes it suitable for password validation, challenge hash authentication, anti-tamper, digital signatures.

Source: http://www.xorbin.com/tools/sha256-hash-calculator

Crypto

- The crypto module provides cryptographic functionality that includes a set of wrappers for OpenSSL's hash, HMAC, cipher, decipher, sign and verify functions.
- We can use a SHA256 hash to generate a unique identifier based on the *contents* of a file.
- Documentation: https://nodejs.org/api/crypto.html#crypto_class_hash

Lab

- Write a node.js application that mines ‘Hello, World!’ Strings such that the Sha256 hash of the string + an incremental number (nonce), yields a hash that begins with three zeroes.
- Write a node.js application that takes an array of objects, and determines if the messages in each object are valid based on the provided signature.