learnyounode Lesson 13 – HTTP JSON API Server

2016-06-16

This is the last lesson in the course. If you have made it this far, your tenacity and dedication to learning node.js is all but confirmed. By now, you have hopefully mastered callbacks and streaming, two fundamental concepts in node.js, on top of all of the other foundational JavaScript knowledge you need to have. For the final lesson, we need to make a JSON API server that accepts a url query string that includes an ISO time string. The server should respond with either an object that contains the hour, minute, and second, or an object that contains the Unix epoch time depending on which endpoint is included in the url query string.

At this point, we know how to set up a server. However, not much has been said about parsing urls. The url module has some useful methods, including the parse() method. The parse() method returns an object with the various components of a url.

Try entering the command below into the command line as the hint suggests.

```
node -pe "require('url').parse('/test?q=1', true)"
```

This command parses a url in a test file included in learnyounode's directory. The command -pe is short for 'print' and 'evaluate'. You are evaluating your program and printing the result to the console. For more on node command line options, check out the documentation.

After you hit enter, you should see a result similar to the example below. The components of the url '/test?q=1' are displayed as an object called Url.

```
Url {
  protocol: null,
  slashes: null,
  auth: null,
  host: null,
  port: null,
  hostname: null,
  hash: null,
  search: '?q=1',
  query: { q: '1' },
  pathname: '/test',
  path: '/test?q=1',
  href: '/test?q=1' }
```

As you can see, the parse() method is handy because the we can access the various properties of the object. For example, we can assign a url that makes a request to our server to a variable called url via dot notation with request.url. See below for an example.

```
var http = require('http')
var url = require('url')

http.createServer(function (request, response) {
  // assign request.url to variable url
  url = url.parse(request.url, true)
}).listen(+process.argv[2])
```

You can build on the example above to get an idea of what the request that is sent to your server looks like. Try creating the program in the example below and running it with the learnyounode run command.

```
var http = require('http')
var url = require('url')

http.createServer(function (request, response) {
  // assign request.url to variable url
  url = url.parse(request.url, true)
  // log contents of url to console
  console.log(url)
}).listen(+process.argv[2])
```

```
learnyounode run myTest.js
```

You should see an object similar to what you saw when you tested the parse() method earlier.

```
Url {
  protocol: null,
  slashes: null,
  auth: null,
  host: null,
  port: null,
  hostname: null,
  hash: null,
  search: '?iso=2016-06-14T15:29:28.234Z',
  query: { iso: '2016-06-14T15:29:28.234Z' },
  pathname: '/api/unixtime',
  path: '/api/unixtime?iso=2016-06-14T15:29:28.234Z',
  href: '/api/unixtime?iso=2016-06-14T15:29:28.234Z' }
```

Notice how the query property of the Url object is also and object?

```
query: { iso: '2016-06-14T15:29:28.234Z' }
```

The object has a property called iso, for iso format time, and the value is an iso format time string. You can access the value of the iso property with dot notation. For example, if you log url.query.iso to the console, your result will be and iso time string similar to this:

```
'2016-06-14T15:29:28.234Z'
```

Between the hints here and the lesson examples, you should have enough to go on to build your program. I'm going to leave you with one more hint. You will have the most luck getting through this lesson if create a function or two that accetp an argument such as time, and return an object that uses a 'hour',

'minute', and 'second', or 'unixtime' as object property names. The object property values can methods such as getHours() or getTime() to return the results you are after. To create a function like this, start with something similar to the example below.

```
var myFunc = function (argument) {
  return {
    myProperty: argument.method()
  }
}
```

If you pass this function to the JSON.stringify() method, it convert the return value of the function to a JSON object.

```
JSON.Stringify(myFunc())
```

My Solution

To solve this challenge, I wrote a program with three functions and and http server. I created a function to parse an iso time string and return the hour, minute, and second as a JSON object, and I created another function which accepts an iso time string and returns a JSON object with the unix epoch conversion of the iso string. Additionally, I created a function which uses a switch statement to call my other functions depending on what endpoint is accessed.

To view the solution that I came up with, or the official solution, click the link to reveal the solution.

```
var http = require('http')
var url = require('url')

var port = process.argv[2]

var parseTime = function (time) {
  return {
    hour: time.getHours(),
    minute: time.getMinutes(),
```

```
10      second: time.getSeconds()
11    }
12  }
13
14  function unixTime (time) {
15    return {unixtime: time.getTime()}
16  }
17
18  var parseQuery = function (url) {
19    switch (url.pathname) {
20      case '/api/parsetime':
21        return parseTime(new Date(url.query.iso))
22      case '/api/unixtime':
23        return unixTime(new Date(url.query.iso))
24      default: return 'please enter a valid endpoint url'
25    }
26  }
27
28  http.createServer(function (request, response) {
29    if (request.method === 'GET') {
30      response.writeHead(200, {'Content-Type': 'application/json'})
31      url = url.parse(request.url, true)
32      response.end(JSON.stringify(parseQuery(url)))
33    } else {
34      response.writeHead(405)
35      response.end()
36    }
37  }).listen(+port, function () {
38    console.log('Server listening on http://localhost:%s', port)
39  })
```

In my solution, I am assigning the http and url modules to the variables http and url. I am also assigning the first argument to my program to the variable port. My first function is called parseTime. The function accepts an argument called time, and returns an object with the hour, minute, and second parsed from the time argument. I am using the getHours(), getMinutes(), and getSeconds() methods to parse the iso time string that is passed as an argument to the parseTime function. By returning an object in JavaScript object notation format, it is easy to convert the results of my function to JSON format with the JSON.stringify() method.

The second function is called unixTime, and is very similar to parseTime. The unixTime function also accepts an argument called time, which we are expecting to be an iso time string. Instead of parsing the string to return the hour, minute, and second, I am using the getTime() method, which returns the

unix epoch equivalent. I am also returning a JavaScript object notation format object in this function withe a property called unixtime.

The last function is parseQuery. The parseQuery function accepts an argument called url and uses a switch statement to parse the url. Why use a switch statement? If I ever wanted to added additional endpoints, a switch statement can help direct urls to those endpoints without making a long chain if if/else if/else statements. Switch statements can also have a default value such as 'please enter a valid endpoint url'. Lastly, switch statements are known to have better performance, though this can vary from browser to browser.

The switch statement condition in parseQuery is the url.pathname property. If the path is '/api/parsetime', returning the results of my parseTime function, which I am passing a new Date() object containing the url.query.iso property. If the path name is '/api/unixtime', I am returning the result of the unixTime function, which I am also passing a new Date() object containing the url.query.iso property. If the value of url.pathname does not equal either endpoint, I am returning the default value of the switch statement, which is 'please enter a valid endpoint url'.

After I declare all of my functions, I am creating a server. If the request method (request.method) of the url that is sent to my server is equal to 'GET', I am writing status 200 and and the content type, which is application/json. Then I am parsing the value of request.url with the url.parse() method, and assigning it to the variable url. Finally, I am calling my parseQuery function and passing the variable url to it. The parseQuery function is passed to the JSON.stringify() method, which converts the result to a JSON object, and the JSON object is passed to the response.end() method. If the request method is not equal to 'GET', I am writing status 405 to the request head, and ending the response.

Finally, I am listening on the port that is passed as the first argument to my program and logging the server url to the console.

Official Solution

The official solution varies slightly from my solution. A third function to parse the query is not used. Additionally, if/else if statements are used in place of a switch statement. To see the official solution, click the link to reveal it.

Click to show/hide solution

Related

learnyounode Unofficial Companion
2016-06-16
In "learnyounode"

SQL Query Strings with Node.js
2017-01-11
In "node.js"

learnyounode Lesson 2 - Baby Steps
2016-06-16
In "learnyounode"

CATEGORIES: LEARNYOUNODE NODE.JS

Written by Joe

Joe Creager is a writer, software developer, and film camera aficionado from the pacific northwest.

ARCHIVES

- May 2017
- April 2017
- January 2017
- December 2016
- September 2016
- August 2016
- July 2016
- June 2016
- January 2016

CATEGORIES

- Book Reviews
- Business
- Data Science
- Programming
  - node.js
    - learnyounode
  - R

META

- Log in
- Entries RSS
- Comments RSS
- WordPress.org

- RSS - Posts
- RSS - Comments

Gravit Theme powered by WordPress