Introduction to Computer Science I
COMP 2406A – Winter 2020

Intro to Middleware and Connect

Dave McKenney
david.mckenney@carleton.ca

Learning Outcomes

by the End of this Lecture, Students that have Completed the Reading Assignment and Review Questions should be Able to:

Understand the function of middleware in Node.js

Create/Use your own middleware functions

Design an HTTP server that relies on middleware

Use existing middleware within your apps

Current State of Affairs

We have used the HTTP module to make a few simple web sites

We can accept requests, extract meaning/data from them, handle them, and send the appropriate response

Current State of Affairs

Our current design uses a single handler function

There is a lot of lower-level programming (parsing URLs, extracting data, etc.)

It is not very extensible, scalable, modular

Connect

Today we will talk about Connect, which is a Node.js middleware framework

After this, we will look at the more advanced/popular Express framework

What is Middleware?

The term ‘middleware’ has many meanings, depending on context

Traditionally, referred to ‘software glue’ that provided a service between OS and applications

What is Middleware?

In Node.js, we can view a middleware as a function that has references for:

1. **The request object**
2. **The response object**
3. **The next middleware function in the 'chain'/'stack' of middleware**

Middleware within this context makes application development easier too

What is Middleware?

A middleware function will typically do some of:

1. **Execute some code**
2. **Make changes to the request/response object(s)**
3. **End the response process**
4. **Call the next middleware function**

Note: it must do either 3 or 4

Middleware ‘Stack’ or ‘Chain’?

We define several middleware functions as an ordered ‘stack’ or ‘chain’ of middleware

Each can be responsible for a part of the server-side processing of requests

Allows for more modular code and gives us ability to handle requests in steps

The ‘Connect’ Framework

"Connect is an extensible HTTP server framework for node using ‘plugins’ known as middleware“[1]

Allows definition of middleware ‘stack’

Created by same team that works on Express (which we will cover next)

https://www.npmjs.com/package/connect

The ‘Connect’ Framework

Connect allows us to write/include our own middleware functions within our apps

We will also see that there are many existing middlewares that work with Connect (and Express)

Again, this leads to simplified, robust, modular code

https://www.npmjs.com/package/connect

Getting Started with Connect

To get started, you must install the connect package:

npm install connect

This will allow you to require ‘connect’ within your Node.js server application

Consider the barebones connect application in 10-ex1-connect-barebones.js

This application doesn't do anything

But we do get 404 messages if we make a request (why?)

‘Use’ing Middleware

Connect and Express apps ‘use’ middleware

We define the functions the app uses to process requests/responses in a particular order

When a request is received, middleware are called in order until next( ) is not called (i.e., the response is completed)

‘Use’ing Middleware

A basic way to add middleware to your app is:

```
app.use(function middlewareName(req, res, next){
    //Middleware code
});
```

The middleware code can use/modify the request/response, and call next( ) to pass to the next middleware function

Adding Our First Middleware

Add a middleware to perform logging for a server

The log should include information about the incoming request whenever a request is made

See 10-ex2-first-middleware.js

‘Mounting’ Middleware

The previous example was a general piece of middleware to run for every request

You can also ‘mount’ specific middleware functions to specific ‘routes’

‘Mounting’ Middleware

```
app.use(“/somepath”, someFunc);
```

Adds *someFunc* as a middleware for requests that start with /somepath

‘Mounting’ Middleware

Consider the code in 10-ex3-rebuilding-the-bank.js

This code begins implementing the server for the bank example using Connect middleware

Adding Another General Middleware

Our bank example requires processing of POST requests with data in the body

So far, we have done this with request stream events:

req.on('data', buildingFunc)

req.on('end' handlingFunc)

Add a middleware to process incoming bodies

‘Mounting’ Multiple Middleware

You can mount multiple middleware for the same route by making multiple calls to app.use

The middleware are still called in order until next( ) is not called

So if we don’t want to process body of ALL requests, we can add our body parser to specific requests

Adding Error Handling

What happens in our current code if we make a GET request for http://localhost:3000/account.html?

Ideally, our server will gracefully handle errors

Adding Error Handling

You can add error handler middleware into the chain

Error handling middleware takes four arguments:

(error, request, response, next)

To ‘throw’ an error from a middleware, you can call: next(“with some error string”);

Adding Error Handling

When next is called with a string, Connect looks for the next middleware in the chain with four arguments

Skips any previous error middleware and any non-error middleware on the way

See 10-ex5-error-middleware.js

Adding More Middleware

Previously, we only handled a subset of all HTTP methods for each resource (e.g., GET, POST)

For example, we only want people to POST to /account.html, to GET /login.html, etc.

How could we add middleware to achieve this? (i.e., send a 4xx/5xx error for an invalid request)

Pre-Made Middleware

When we move to Express, we will see a better way of achieving this then writing our own middleware

In fact, there is a lot of existing middleware packages to make our lives easier

Pre-Made Middleware – body-parser

The ‘body-parser’ package provides an improved version of the body parser we created

Allows us to easily handle different content types and encodings

Automatically adds a ‘body’ property to the request object, so we can use it in latter handlers

Pre-Made Middleware – body-parser

npm install body-parser

let bodyParser = require(“body-parser”);

app.use(bodyParser.json(*[options]*));

Will parse JSON data automatically – uses options object, if provided, for configuration

Common JSON Parsing Options:

inflate: true/false - accept/inflate compressed data or reject

limit: #bytes – sets maximum body size in bytes (e.g., 100kb, 5mb, 1000)

type: “type/subtype” – sets the type(s) the parser will handle (e.g., “application/json”, “*/*”)

Pre-Made Middleware – body-parser

npm install body-parser

let bodyParser = require(“body-parser”);

app.use(bodyParser.text(*[options]*));

Will parse text data automatically – uses options object, if provided, for configuration

Common JSON Parsing Options:

defaultCharset: “utf-8” – sets default encoding if one is not specified in Content-Type header

inflate: true/false - accept/inflate compressed data or reject

limit: #bytes – sets maximum body size in bytes (e.g., 100kb, 5mb, 1000)

type: “type/subtype” – sets the type(s) the parser will handle (e.g., “text/plain”, “text/html”)

Pre-Made Middleware – body-parser

For an example of body-parser in action, see

10-ex7-built-in-middleware.js

The external module provides improved functionality over our previous body parser

Pre-Made Middleware – serve-static

A very useful existing middleware is ‘serve-static’

Allows you to easily serve static content from specific locations

Automatically provides: proper MIME types, good response codes, security (/../../ issue), default index pages, etc.

Pre-Made Middleware – serve-static

```
npm install serve-static
```

```
let serveStatic = require('serve-static’);
```

```
app.use(serveStatic(path, [options]));
```

Serves static files in ‘./path/’ using given options

Pre-Made Middleware – serve-static

Options can be used to specify things such as:

Cache control – last modified, max age, etc.

Show hidden files or not

Filter extensions to server

Fallthrough – go to next middleware if resource doesn't exist, or send error response

Default index file name

Pre-Made Middleware – serve-static

So with this package, we can create a static server with very little code

This can also allow us to organize our resources easily:

```
/public
 |--- /js
        |--- script1.js
        |--- script2.js
 |--- /css
        |--- mystyles.css
 |--- /images
        |--- logo.jpeg
        |--- meme.gif
 |--- index.html
 |--- otherpage.html
```

Pre-Made Middleware – serve-index

Another useful middleware is serve-index

Like serve-static, this will automatically handle many responses for us

Specifically, it provides a way of serving the directory/file structure, but not the files themselves

Pre-Made Middleware – serve-index

```
npm install serve-index
```

```
let serveIndex = require('serve-index’)
```

```
app.use(serveIndex(path, options));
```

Serves an index for all directories in/under path

Pre-Made Middleware – serve-index

Commonly used options:

Filter: specifies a function to indicate whether to serve a directory or not

Hidden: include hidden files or not

Icons: display the file-type icons

Stylesheet: provides a way of supplying a custom style for the auto-generated pages

Middleware-Based Library

So our library example from a previous lecture can easily be implemented using just the serve-index and serve-static modules

See 10-ex9-library.js

Summary

So, Connect allows us to specify middleware

This middleware can process/handle requests in the order specified

We can design more modular systems, in which we can easily add/remove/change functionality

Summary

Additionally, we can use available middleware software within our own apps

Allows us to build more robust, secure, powerful systems without writing a lot of low-level code

Summary

Next, we will look at Express, which builds upon Connect to create a middleware-based web framework

Questions?