Express Middleware, Cookies, & Sessions

Middleware – the Traditional View

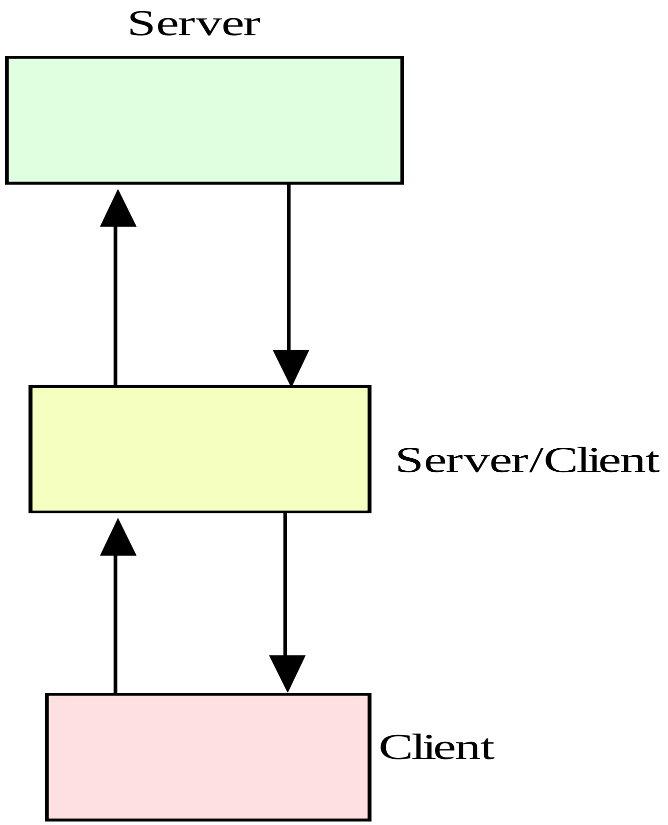


Figure from *Distributed Computing*, M. Liu

A traditional Enterprise-view of middleware is related to our ideas of distribute components, tiered enterprise systems, and Separation of Concerns.

In this view, based on a *layered architecture style*, one or more layers sits between clients and servers.

If the *client* in this diagram is Tomcat, and the Server is our database, then our middleware might be a messaging platform, or a distributed object cache

In web development, we refer to the sequences of calls made between accepting the HTTP request and generating the response the *request processing pipeline*. In the traditional middleware view, the pipeline is a set of distributed process invocations (think, distributed call stack, kinda)

Middleware – The Node/Express View

Recall our conversation about *functional programming (FP)* versus *object-oriented programming (OOP):*

- OOP encapsulates behaviors, state and identity
  - The application is view as a collection of communicating objects
    - The *communication* is represented in your sequence diagrams
- FP encapsulates discrete computational tasks, or functions
  - The application is viewed as a discretized *workflow of tasks*
- The MVC pattern provides us with some idea of *control flow* to the application
  - Remember the 6 steps!
  - In the Java (OOP) world, the Controller *delegates* to the Model
  - In the Node (FP) world, a *route* defines an entry point into our set of discrete tasks to generate a response
  - Problem is, we haven't talked about how we structure those tasks
  - The *request processing pipeline* provides us some guidance

Déjà vu all over again

Remember the Template Pattern?

- In there we identified 6 generic parts to the request processing pipeline
- No different in Node (or Django, or RoR, or …)
- The key is in how you organize the work to
  - Make sense of it! (Maintainability)
  - Optimize use of shared resources (Scalability)
  - Work fast (Performance)
  - While remaining loosely-coupled

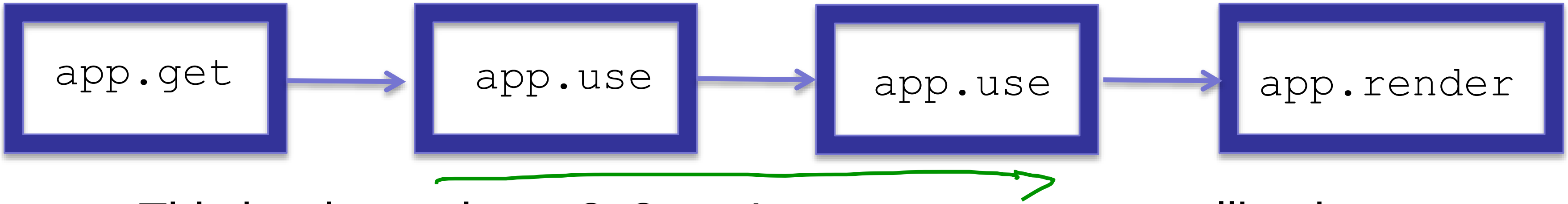
1. Process HTTP request headers
2. Process HTTP request parameters
3. Route the request to the appropriate handler to do the work
4. Assemble the response payload
5. Set the response headers
6. Write the response

Back to Node/Express Middleware

Express refers to the set of functions comprising the request processing pipeline, after the route handling of the HTTP verb and the rendering step of a template engine (or other subflow) as *middleware*

CoR

- Middleware steps identified by *app.use*
- Each step must invoke *next()* to continue the pipeline
- The ordering of the pipeline depends upon the order in which you set routes and middleware up, so pay attention to where things go!



app.get

app.use

app.use

app.render

- This is why we have 2, 3, or 4 parameters to our callbacks:
  - Request and response
  - Next – if you do not have a next callback or do not call next(), the pipeline just stops (this could be very bad if unintended)
  - Error-handling – optional but as always a good idea

Writing your own Middleware

Writing your own middleware is easy, it is just functions!

```
// express_listen_mw1.js
var app = require('express')();
```

Function

```
app.get('/', function(req, res, next){// mandatory signature of cb
  res.send('Hello from Express');
  next();                               // always call next at end
});
```

path

```
app.use('/foo',function(req, res, next) { // default path is ‘/’
   console.log('First app.use call');
   next();
});
app.use function(req, res, next) {   // if / this called 1st
   console.log('2nd app.use call');
   next();
});
app.use function(req, res, next) { // chain set up by order of use calls
   console.log('3rd app.use call');
});
app.listen(8081);
```

Some Common Node/Express Middleware

1. json – parse JSON-encoded request payloads. This is becoming more popular, particularly with REST APIs

2. urlencoded – parse `application/x-www-form-urlencoded` request payloads (more popular than json for now)

3. body-parser – includes both json and urlencoded

4. compress – compress response data with gzip

5. query – converts query string into Javascript object

6. static – serves static files

7. cookie-parser – *stay tuned…*

8. express-session – *stay tuned...*

You can npm install a whole bunch of others...

Cookies

— personal, User Profile Stored in Browser

Part of HTTP as we discussed before

HTTP:

Set-Cookie:

- First, include the cookie parsing middleware
  - `app.use(require('cookie-parser')());`
- Second, set a cookie in the response
  - `response.cookie(`*`<cookie-name>`*`, `*`<cookie-value>`*`);`
  - Example: `response.cookie('foo', 'bar');`
- Third, get rid of a cookie:
  - `response.clearCookie('foo');`
- To read a cookie value, just use the property syntax:
  - `request.cookies.foo // evaluates to 'bar'`
- *That's it!* *(see example express_cookie.js)*
- Cookies have the usual properties
  - Domain, path, maxAge, secure (https only)
  - *httpOnly*: Only server can modify the cookie, not browser Javascript
  - *signed*: sign the cookie with a shared secret

Signed Cookies

We did not discuss these before

- Note when we required the cookie-parser
  - `app.use(require(‘cookie-parser’)());`
- There was no param to the constructor. Let’s add one
  - `app.use(require(‘cookie-parser’)(<secret>));`
    - Where `<secret>` is a shared secret string
- Now when you access a cookie, you can require that it is signed by your own secret string
  - Set the signed cookie:
    - `response.cookie(‘foo’, ‘bar’, {signed : true});`
  - Read the value of the signed cookie:
    - `request.signedCookies.foo // evaluates to ‘bar’`
- If the cookie was tampered with on the client then the server will reject reading the value!

Sessions – Conversational State

More middleware

bounded interaction

Shopping cart

- Actually several flavors for Express (no “standard” ala JCP)
- We will use the basic module express-session, which uses cookies to store an identifier the server uses to access a local memory store for the session information (sound familiar)
- `app.use(require(‘cookie-parser’)());`
- `app.use(require(‘express-session’)(<options>));`
  - Where <options> is a configuring object – you shouldn’t need to modify the default option settings

- Sessions are accessed on the request object
  - Use Javascript’s property syntax
    - `request.session.<property> [= value]; // to get [set]`
  - Use the delete operator to remove a property
    - `delete request.session.foo`
- Example: express_session.js

Client
cookie-id
Server

Summary

- In a distributed component-based or service-oriented world, this word suggests an additional layer, or tier, in our computational model
- In Node/Express, it is functional expression of the request processing pipeline
- In web development, the *request processing pipeline* refers to the stages of process applied to a web request to transform it to a web response
- Cookies are from HTTP, Node/Express provides its own modules to make it easy
- Sessions are not from HTTP, and the Node community provides a lot of modules for doing them
    - Not part of any standard as in Java