CS5220 Advanced Topics in Web Programming

Web Development Using Express Framework

Chengyu Sun
California State University, Los Angeles

HelloWorld in Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

@WebServlet( “/HelloWorld” )
public class HelloWorld extends HttpServlet {
    public void doGet( HttpServletRequest request,
                       HttpServletResponse response )
        throws ServletExceptoin, IOException
    {
        response.setContentType("text/plain");
        response.getWriter().println( “Hello World” );
    }
}
```

HelloWorld in Node.js

```
const http = require('http');

const server = http.createServer();

server.on( 'request', (request, response) => {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.write('Hello world!');
  response.end();
});

server.listen(8080);
```

- There is no separate application server

EventEmitter

- http.Server, http.IncomingMessage, http.ServerResponse, and many other things in Node.js, are EventEmitter
- An EventEmitter emits events
- Each event has a name and can have one or more event listeners (a.k.a. event handlers)

EventEmitter API

- `addListener()` and `on()`
- `once()`: the event listener will be called at most once
- `removeEventListener()` and `removeAllEventListeners()`: remove one or all listeners on the given event

Running Node.js Server Applications

- Run server applications using nodemon during development
  - Automatically restart the application when changes in the project are detected
- Deploy server applications using pm2
  - Run server applications as managed background processes

Using `npx`

- Execute commands from local `node_modules/.bin` (useful for running commands installed as dev packages)
  - E.g. `npx nodemon index.js`
- Execute some one-off commands without installing them as packages
  - E.g. `npx gitignore node`

Graceful Shutdown

- Properly save data and release resources (e.g. database connections) when a server application is stopped

The Process Object …

- process is a Node.js global object that provides information and control over the current Node.js process
- The beforeExit and exit event are not what you think – they are related to "normal" exit like when no more events are left in the event loop
- Instead, we need to handle Signal Events

… The Process Object …

- Process termination is a bit complicated
  - There are many ways to terminate a process
  - Some "termination" events do not accept event handlers
  - Different platforms work differently
  - Process managers like nodemon can further complicate things

… The Process Object

- Signal Events that should be handled for process termination
  - `SIGINT`: Ctrl-C, PM2 stop/reload/restart
  - `SIGTERM`: used by some cloud service like Heroku
  - `SIGUSR2`: nodemon restart
- Example: webapp.js

What's Still Missing

- URL mapping, a.k.a. *routing*
- Request parsing: request parameters, JSON body, uploaded files …
- Session management
- View support
- Basic error handling, e.g. rejecting malformed requests
- … …

Express

- https://expressjs.com/
- The E in MEAN/MERN stack
- A *minimalistic, unopinionated* web application framework for Node.js
- The platform for many useful packages and the basis for many popular frameworks

Using Express

- Application
- Routing
- Handling requests
- Generating responses
- Middleware
- Handling errors
- Scaffolding an MVC application
- View engines
- Configuration using environment variables

HelloWorld in Express

- Install the `express` package

```
const express = require('express');
const app = express();

app.get('/', (req, res) =>
  res.send('Hello World!'));

app.listen(3000, () =>
  console.log('Listening on port 3000'));
```

Application

```
const app = express();
```

- The Application object
  - Routing requests
  - Rendering views
  - Configuring middleware

Routing Methods in App

- `app.METHOD( path, callback, [,callback …])`
  - `METHOD` is one of the routing methods, e.g. `get`, `post`, and so on
- `app.all( path, callback [,callback …])`

Modularize Endpoints Using Express Router …

- Example
  - List users: `/users/`, `GET`
  - Add user: `/users/`, `POST`
  - Get user: `/users/:id`, `GET`
  - Delete user: `/users/:id`, `DELETE`

… Modularize Endpoints Using Express Router …

users.js

```
const router = express.Router();

router.get('/', …);
router.post('/', …);
… …

module.exports = router;
```

- A router is like a "mini app"

… Modularize Endpoints Using Express Router

app.js

```
const users = require('./users');
app.use('/users', users);
```

- Attach the router to the main app at the URL

Handling Requests

- Request
  - Properties for basic request information such as URL, method, cookies
  - Get header: get()
  - User input
    - Request parameters: req.query
    - Route parameters: req.params
    - Form data: req.body
    - JSON data: req.body

Generating Response

- Response
  - Set status: status()
    - end()
  - Send JSON: json()
  - Send other data: send()
  - Redirect: redirect()
  - Other methods for set headers, cookies, download files etc.

Example: Add

- GET: `/add?a=10&b=20`
- GET: `/add/a/10/b/20`
- POST (Form): `/add`
  - Body: `a=10&b=20`
- POST (JSON): `/add`
  - Content-Type: `application/json`
  - Body: `{"a": "10", "b": "20"}`

Using Express Middleware

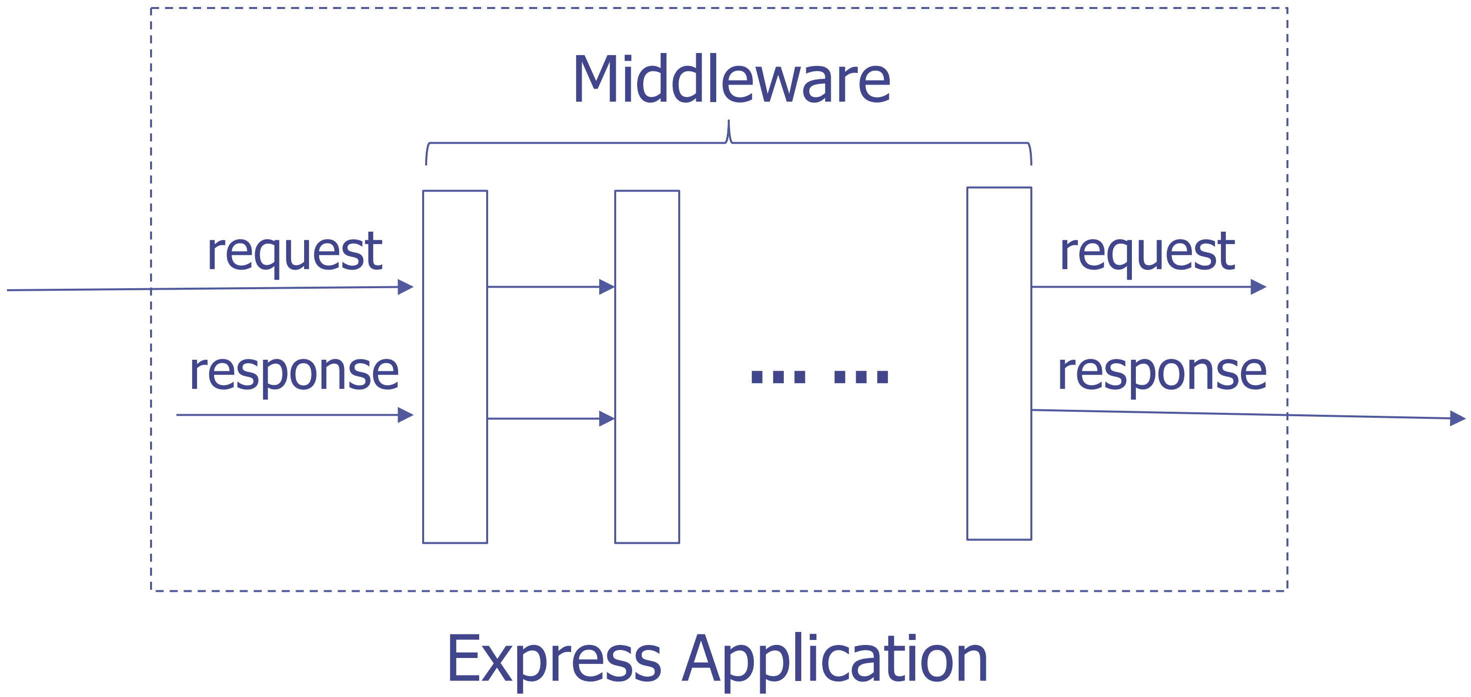
Parses urlencoded request body (i.e. form submission) and add request parameters to `req.body`

```
app.use( express.urlencoded() );
```

```
app.use( express.json() );
```

Parses JSON request body and add JSON object properties to `req.body`

Middleware



- A middleware is a function that has access to three arguments: the `request` object, the `response` object, and a `next` function that passes control to the next middleware function

Middleware Example

- Create a simple request logger middleware that prints out request URL, method, and time
  - The `next` argument
  - Add the middleware to the application using `app.use()`
  - Middleware can also be added at the router level with `router.use()`

About Middleware

- Route handler functions are also middleware
  - Where is `next`??
  - Remember to use `next` if you have more than one handler for a route
- *Middleware order is important!*

Error Handling Middleware

- Error handling middleware has an extra argument `err`, e.g. `(err, req, res, next)`
- Calling `next(err)` will bypass the rest of the regular middleware and pass control to the next error handling middleware
  - `err` is an Error object
- Express adds a default error handling middleware at the end of the middleware chain

Error Handling Example

- Create a middleware that handles 404 errors and returns JSON instead of an error page

Using Express Application Generator

- Install `express-generator` as a global package or run it with `npx`
- Command line options
  - `-h, --help`
  - `-v, --view`
  - `-c, --css`
  - `--git`

Template Engine

- A template engine combines templates with data to produce documents
- A.K.A. view engine, though it’s not just for *views* in MVC, e.g. email templates

JSP As A Template Engine

```
<p>Hello, ${user.firstName}!</p>

<c:if test="${not empty tasks}">
    <p>Your tasks for today:</p>
    <ul>
        <c:forEach items="${tasks}" var="task">
        <li>${task}</li>
        </c:forEach>
    </ul>
</c:if>
```

Basic Characteristics of a Template Engine

- Template language
  - Access objects and properties
  - Basic expression and control flow statements
- Server and/or client-side rendering
- Support for formatting and i18n
- Performance

Hogan / HJS

```
<p>Hello, {{user.firstName}}!</p>

{{#hasTasks}}
    <p>Your tasks for today:</p>
    <ul>
        {{#tasks}}
        <li>{{.}}</li>
        {{/tasks}}
    </ul>
{{/hasTasks}}
```

- Developed by Twitter
- Use the “logic-less” Mustache template language

Handlebars

```
<p>Hello, {{user.firstName}}!</p>

{{#if tasks}}
    <p>Your tasks for today:</p>
    <ul>
        {{#each tasks}}
        <li>{{.}}</li>
        {{/each}}
    </ul>
{{/if}}
```

- Extension to Mustache (arguably making it easier to use)

Dust

```
<p>Hello, {user.firstName}!</p>

{?tasks}
    <p>Your tasks for today:</p>
    <ul>
        {#tasks}
        <li>{.}</li>
        {/tasks}
    </ul>
{/tasks}
```

- Developed by LinkedIn

EJS

```
<p>Hello, <%= user.firstName %>!</p>

<% if (tasks) { %>
    <p>Your tasks for today:</p>
    <ul>
        <% for( var i=0 ; i < tasks.length ; ++i ) { %>
        <li><%= tasks[i] %></li>
        <% } %>
    </ul>
<% } %>
```

- Syntax is similar to JSP scripting elements

Twig

```
<p>Hello, {{user.firstName}}!</p>

{% if tasks %}
    <p>Your tasks for today:</p>
    <ul>
        {% for task in tasks %}
        <li>{{task}}</li>
        {% endfor %}
    </ul>
{% endif %}
```

- Originally a PHP template engine

Vash

```
<p>Hello, @user.firstName!</p>

if( tasks ){
    <p>Your tasks for today:</p>
    <ul>
        @tasks.forEach( function(task) {
        <li>@task</li>
        }
    </ul>
}
```

- Use the Razor (i.e. the view engine in ASP.NET MVC) syntax

Jade / Pug

```
p Hello, #{user.firstName}!

if tasks
    p Your tasks for today:
    ul
        each task in tasks
            li= task
```

- Jade is renamed to Pug due to trademark problem

Application Structure

- `/public` for static resources
- `/routes` for controllers
- `/views` for view templates
- `/bin` for executables
- `package.json`
  - Packages
  - npm start

Customize Configuration Using Environment Variables

```
process.env.PORT || '3000'
```

- We often need to change runtime configuration such as port number, database url/username/password, log file location and so on
- Java applications usually use property files
- Node.js application prefer environment variables

Set Environment Variables Using `dotenv` Package

- Put the variables in a `.env` file, e.g.

```
PORT=3000
USERNAME=cysun
PASSWORD=abcd
```

- Run `require('dotenv').config()` at the beginning of the application
- Include and version control a `.env.sample` file

Controller

```
router.get('/users', function(req, res, next) {
  User.find( (err, users) => {
    if(err) return next(err);
    res.render('users’, {title: 'Users', users: users});
  });
});
```

Handle Error

Render View

View Name

“Locals” (model attributes in Spring)

Handlebars Views

- `layout.hbs` is the default master page
  - Set a `layout` local to use a different master page
- A child view is combined with the master page as `{{{body}}}`
  - Triple braces mean do not escape content

Readings

- [Express Documentation]