<WA1/>

2020

express

A look at the server side

Enrico Masala
Fulvio Corno
Luigi De Russis

Goal

- Implement a (simple, minimal) web server
  - In JavaScript
  - For hosting static contents
  - For hosting dynamic APIs
  - Supporting persistence in a Database

https://expressjs.com/
https://github.com/expressjs/express

The Express Handbook, Flavio Copes
https://flaviocopes.com/page/express-handbook/

A simple and easy to use HTTP and Application server

EXPRESS

Web Frameworks in Node

- Node already contains a ‘http’ module to activate a web server
  - Low-level, non very friendly
- Several other frameworks were developed
- Express is among one of the most popular, and quite easy to use

```
npm init
npm install express
node index.js
```

http://nodeframework.com/

First steps with Express

- Calling `express()` creates an application object `app`
- `app.listen()` starts the server on the specified port (3000)
- Incoming HTTP request are routed to a callback according to
  - **path**, e.g., `'/'`
  - **method**, e.g., `get`
- Callback receives Request and Response objects (**`req, res`**)

```
// Import package
const express = require('express') ;
// Create application
const app = express() ;

// Define routes and web pages
app.get('/', (req, r =>
       res.send('Hello Worldes)!')) ;

// Activate server
app.listen(3000, () =>
       console.log('Server  ready')) ;
```

Routing

- `app.method(path, handler);`
  - `app`: the express instance
  - `method`: an HTTP Request method (`get`, `post`, `put`, `delete`, …)
    - `app.all()` catches all request types
  - `path`: a path on the server
    - Matched with the path in the HTTP Request Message
  - `handler`: callback executed when the route is matched

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

Handler callbacks

```
function (req, res) { ... }
```

req (Request object)

Property	Description
.app	holds a reference to the Express app object
.baseUrl	the base path on which the app responds
.body	contains the data submitted in the request body (must be parsed and populated manually before you can access it)
.cookies	contains the cookies sent by the request (needs the `cookie-parser` middleware)
.hostname	the server hostname
.ip	the server IP
.method	the HTTP method used
.params	the route named parameters
.path	the URL path
.protocol	the request protocol
.query	an object containing all the query strings used in the request
.secure	true if the request is secure (uses HTTPS)
.signedCookies	contains the signed cookies sent by the request (needs the `cookie-parser` middleware)
.xhr	true if the request is an XMLHttpRequest

res (Response object)

Method	Description
res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
res.sendFile()	Send a file as an octet stream.
res.sendStatus()	Set the response status code and send its string representation as the response body.

https://expressjs.com/en/guide/routing.html

Generate a response

- `res.send('something')` sets the response body and returns it to the browser
- `res.end()` sends an empty response
- `res.status()` sets the response status code
  - `res.status(200).send()`
  - `res.status(404).end()`
- `res.json()` sends an object by serializing it into JSON
  - `res.json({a:3, b:7})`
- `res.download()` prompts the user do download (not display) the resource

Redirects

- `res.redirect('/go-there')`

Extending express with middleware

- **Middleware**: a function that is called for every request
- `function(req, res, next)`
  - Receives `(req,res)`, may process and modify them
  - Calls `next()` to activate the next middleware function
- **Register** a middleware with
  - `app.use(callback)`
  - `app.use(path, callback)` // only requests in the specified path

Serving static requests

- Middleware: `express.static(root, [options])`
- All files under the root are served automatically
  - No need to register app.get handlers

```
app.use(express.static('public'));

Serves files from ./public as:
http://localhost:3000/images/kitten.jpg
http://localhost:3000/css/style.css
http://localhost:3000/js/app.js
http://localhost:3000/images/bg.png
http://localhost:3000/hello.html
```

```
app.use('/static', express.static('public'));

Serves files from ./public as:
http://localhost:3000/static/images/kitten.jpg
http://localhost:3000/static/css/style.css
http://localhost:3000/static/js/app.js
http://localhost:3000/static/images/bg.png
http://localhost:3000/static/hello.html
```

https://expressjs.com/en/starter/static-files.html

Interpreting request parameters

Request method	Parameters	Values available in	Middleware requested

GET	URL-encoded
	`/login?user=fc&pass=123`
		`req.query`
		`req.query.user`
		`req.query.pass`
			none

POST/PUT	FORM-encoded in the body	`req.body`
		`req.body.user`
		`req.body.pass`
			`express.urlencoded`

POST/PUT	JSON stored in the body `{ "user": "fc", "pass": "123" }`		`express.json`

Paths

Path type	Example
Simple paths (String prefix)	`app.get('/abcd', (req, res, next)=> {`

Path Pattern (Regular expressions)	`app.get('/abc?d', (req, res, next)=> {`
	`app.get('/ab+cd', (req, res, next)=> {`
	`app.get('/ab\*cd', (req, res, next)=> {`
	`app.get('/a(bc)?d', (req, res, next)=> {`

JS Regexp object	`app.get(/\/abc|\/xyz/, (req, res, next)=> {`
Array (more than one path)	`app.get(['/abcd', '/xyza', /\/lmn|\/pqr/],` `(req, res, next)=> {`

https://expressjs.com/en/4x/api.html#path-examples

Parametric paths

- A Path may contain one or more *parametric segments*:
  - Using the '`:id`' syntax
  - Free matching segments
  - Bound to an identifier
  - Available in `req.params`
- May specify a matching regexp
  - `/user/:userId(\d+)`

```
app.get('/users/:userId/books/:bookId', (req,
res) => {
  res.send(req.params)
});

Request URL:
http://localhost:3000/users/34/books/8989

Results in:
req.params.userId == "34"
req.params.bookId == "8989"
```

https://expressjs.com/en/guide/routing.html#route-parameters

Logging

- By default, express does not log the received requests
- For debugging purposes, it's useful to activate a logging middleware
- Example: `morgan`
  - https://github.com/expressjs/morgan (`npm install morgan`)
  - `const morgan = require('morgan');`
  - `app.use(morgan('tiny'));`

Validating input

- https://express-validator.github.io/docs/ (`npm install express-validator`)
- Declarative validator for query parameters

```
app.post('/user', [   // additional (2nd) parameter in app.post to pre-process request
check('username').isEmail(), // username must be an email
check('password').isLength({ min: 5 }) // password must be at least 5 chars long
], (req, res) => {
const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(422).json({ errors: errors.array() });
  }
 . . . Process request
});
```

https://github.com/validatorjs/validator.js#validators

Other middleware

Middleware module	Description	Replaces built-in function (Express 3)
body-parser	Parse HTTP request body. See also: body, co-body, and raw-body.	express.bodyParser
compression	Compress HTTP responses.	express.compress
connect-rid	Generate unique request ID.	NA
cookie-parser	Parse cookie header and populate `req.cookies`. See also cookies and keygrip.	express.cookieParser
cookie-session	Establish cookie-based sessions.	express.cookieSession
cors	Enable cross-origin resource sharing (CORS) with various options.	NA
csurf	Protect from CSRF exploits.	express.csrf
errorhandler	Development error-handling/debugging.	express.errorHandler
method-override	Override HTTP methods using header.	express.methodOverride
morgan	HTTP request logger.	express.logger
multer	Handle multi-part form data.	express.bodyParser
response-time	Record HTTP response time.	express.responseTime
serve-favicon	Serve a favicon.	express.favicon
serve-index	Serve directory listing for a given path.	express.directory
serve-static	Serve static files.	express.static
session	Establish server-based sessions (development only).	express.session
timeout	Set a timeout period for HTTP request processing.	express.timeout
vhost	Create virtual domains.	express.vhost

https://expressjs.com/en/resources/middleware.html

Guidelines for implementing back-end APIs

REST API IN EXPRESS

REST API implementation

- REST API endpoints are just regular HTTP requests
- Request URL contain the Resource Identifiers (`/dogs/1234`)
  - Extensive usage of parametric paths (`/dogs/:dogId`)
- Request/response Body contain the Resource Representation (in JSON)
  - `req.body` with `express.json` middleware
  - `res.json()` to send the response
- Always validate input parameters
- Always validate input parameters
- Really, always validate input parameters

	Collections	Elements

GET	`app.get('/courses', (req, res) => {`
	`  dao.listCourses().then((courses) => {`
	`  **res.json(courses);**`
	`  });`
	`});`
		`app.get('/courses/:code', (req, res) => {`
		`  // validation of req.params.code!!`
		`  dao.readCourse(req.params.code)`
		`  .then((course)=>res.json(course));`
		`});`

POST
PUT
		`app.use(express.json);`
		
		`app.post('/exams', (req, res) => {`
		`  const exam = req.body;`
		`  // validation of exam!!`
		`  dao.createExam(exam);`
		`});`

How to store data in a database

DATA PERSISTENCE

Server-side persistence

- The web server should normally store into a persistent database
- Node supports most databases
  - Cassandra, Couchbase, CouchDB, LevelDB, MySQL, MongoDB, Neo4j, Oracle, PostgreSQL, Redis, SQL Server, SQLite, Elasticsearch
- An easy solution for simple and small-volume applications is **SQLite** (in-process on-file relational database)

https://expressjs.com/en/guide/database-integration.html

SQLite

- Uses the ‘sqlite’ npm module
- Documentation: https://github.com/mapbox/node-sqlite3/wiki

```
npm install sqlite3
```

```
const sqlite = require('sqlite3');
const db = new sqlite.Database('exams.sqlite', // DB filename
  (err) => { if (err) throw err; });

...
db.close();
```

SQlite: queries

```
rows.forEach((row) => {
   console.log(row.name);
 });
```

- const sql = "SELECT…";
- `db.all(sql, [params], (err, rows) => { } )`
  - Executes sql and returns all the rows in the callback
  - If err is true, some error occurred. Otherwise, rows contains the result
  - Rows is an array. Each item contains the fields of the result
- `db.get(sql, [params], (err, row) => { } )`
  - Get only the first row of the result (e.g., when the result has 0 or 1 elements: primary key queries, aggregate functions, …)
- `db.each(sql, [params], (err, row) => { } )`
  - Executes the callback once per each result row (no need to store all of them)

https://www.sqlitetutorial.net/sqlite-nodejs/

Parametric queries

- The SQL string may contain parameter placeholders: ?
- The placeholders are replaced by the values in the [params] array
  - In order: one param per each ?

```
const sql = 'SELECT * FROM course WHERE code=?';
db.get(sql, [code], (err, row) => {
```

- Always use parametric queries – never string+concatenation nor `template strings`

SQlite: queries

- `db.run(sql, [params], (err) => { } )`
  - For statement that do not return a value
  - CREATE TABLE
  - INSERT
  - UPDATE
  - In the callback function
    - `this.changes` == number of affected rows
    - `this.lastID` == number of inserted row ID (for INSERT queries)

https://www.sqlitetutorial.net/sqlite-nodejs/

License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for commercial purposes.
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- https://creativecommons.org/licenses/by-nc-sa/4.0/