Maharishi International University - Fairfield, Iowa

How Express Works – Part 1

Express starts by reading all your middleware functions, routes, and error handlers and **registers/schedule** them **by order**, so it knows exactly the execution order.

When a request arrives, Express invokes the first request handler function that matches the Route (HTTP verb and URL) and passes the **`request`**, **`response`**, and a reference to the **`next`** request handler function in order.

How Express Works – Part 2

A request handler may perform one of these actions:

- call **`next()`** to invoke the next scheduled request handler function in order, that matches the Route, which will also receive the same **`request`**, **`response`** objects, along with a reference to the **`next`** scheduled request handler function in order.
- call **`next(something)`** to skip all upcoming request handlers and invoke the error handler function, passing the same **`request`**, **`response`** objects.
- **send out the response**, when you call **`res.status().json()`**, you send out the response and you cannot call **`next()`** afterward. You may need to use a **`return`** statement to stop the execution flow.

App Configurations

There are two ways to configure express application instance:

app.set()/app.get()

```
app.set('port', process.env.PORT || 3000);
const port = app.get('port);
```

app.enable()/app.disable()

```
app.enable('etag') === app.set('etag', true)
app.disable('etag') === app.set('etag', false)
```

Request Object

`request.headers` `Example: req.headers['authorization'];`

`request.params`

`request.query`

`request.route` Returns currently-matched route

`request.body` You need to use a middleware to parse the request body

Other Request Properties/Methods https://expressjs.com/en/5x/api.html#req

Request Object Examples

request.query

Optional

```
http://localhost:3000/search?q=nodejs&lang=eng
{"q": "nodejs", "lang": "eng"}
```

request.params

Mandatory

```
app.get('/api/:id/:name/:city',
        function(req, res) {
                console.log(req.params);
        });
http://localhost:3000/api/1/Asaad/Fairfield
{ id: 1, name: 'Asaad', city: 'Fairfield' }
```

request.body

```
app.use(express.json());
app.post('/api', function(req, res){
        console.log(req.body);
});
```

Response Object

- **`response.redirect(url)`** Redirect to new path with status 302
- **`response.send(data)`** Send response
- **`response.json(data)`** Send JSON with proper headers
- **`response.download(pathToFile, newName)`**
- **`response.status(status)`** Send status code

Other Response Properties/Methods https://expressjs.com/en/5x/api.html#res

The `response.send()` method conveniently outputs any data application thrown at it (such as strings, JavaScript objects, and even Buffers) with automatically generated proper HTTP headers (Content-Length, ETag, or Cache-Control).

Manipulating the Response Header

`res.set()` is used to set the headers of the response.

```
// single header
res.set('content-type', 'application/json');

// multiple headers can be set
res.set({
   'content-type': 'application/json',
   'content-length': '100’,
   'warning': "this course is the best course ever"
});
```

Middleware

A Middleware is a Request Handler, a useful pattern that allows developers to reuse code within their applications and even share it with others in the form of NPM modules.

The request (req) and response (res) objects are the same for the subsequent middleware.

Use a Middleware

To use a middleware, we call the `app.use()` method which accepts:

- One **optional URL path**.
- One **request handler callback function**.

```
const middleware: RequestHandler<T, S, U, V> = function(req, res, next) {
        // ...
        return next();
}
// apply to all routes
app.use(middleware);

// apply to a specific route
app.verb('/', middleware, requestHandler);
```

`next()`

`next()`

Go to the next scheduled request handler function (middleware, route)

`next(something)`

Invoke the Error handler

Built-in Middlewares

Express comes with many built-in middlewares, some that we will use:

- `express.static()`
- `express.json()`
- `express.urlencoded()`
- `express.Router()`

express.static()

static is a built-in middleware, it enables pass-through requests for static assets.

```
// will setup a middleware on the provided path and return a middleware function
// files will be read from that path and will be streamed immediately to response


import { join } from 'path';

app.use('/images', express.static(join(__dirname, 'upload')))
```

`express.json()` and `express.urlencoded()`

Built-in middleware functions in Express to parse incoming requests body with JSON or URL encoded payloads and assign them to **`req.body`**.

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

express.Router()

The Router class is a mini Express application that has only middleware and routes. This is useful for **abstracting modules** based on the business logic that they perform.

myRoute.js

```
import express from 'express';
const router = express.Router();

router.get('/', get_all_handler);
router.post('/', express.json(), post_handler);
router.get('/:id', get_one_handler);
router.put('/:id', express.json(), put_handler);
router.delete('/:id', delete_handler);

export default router; // export the Router middleware
```

Pass `{ mergeParams: true }` to the sub-entity Router instance, to extend the main route params.

Middleware Order

When using middleware, the order in which middleware functions are applied matters, because **this is the order in which they'll be executed**.

Useful 3rd-party middleware:

```
'morgan', '@types/morgan' // logger
'cors', '@types/cors' // accept CORS requests
'multer', '@types/multer' // upload files
'helmet' // secure Express apps by setting HTTP response headers
```

Throwing an Error

If you pass anything to the **`next()`** function, Express considers the current request as being in error and will skip any remaining non-error handling routing and middleware functions and passes the request/response to error handlers.

```
router.get('/user', request_handler);

const request_handler: RequestHandler =  async (req, res, next) => {
    try {
       // your logic here
       if(somethingWrong) throw new Error(`User Not found`)
    } catch (error) {
        next(error)
    }
  }
```

Error Handlers in Express

Define error-handling middleware functions in the same way as other middleware functions, except error-handling functions have four arguments instead of three: `(error, req, res, next)`

```
const errorHandler: ErrorRequestHandler = (error, req, res, next)=>{
        console.error(error.stack);
        res.status(500).json({'msg': error.message });
}

app.use(errorHandler);
```

IMPORTANT: You define error-handling middleware last, after other middleware and routes calls.