
express

Content

- Introduction
- Installation
- Routing
- Serving Static Files
- File Upload
- Cookies Management

Introduction

- Express is a minimal and flexible Node.js web application framework
- Opensource, primarily developed and maintained by TJ Holowaychuk.
- provides a robust set of features to develop web and mobile applications
- facilitates the rapid development of Node based Web applications

Features of Express

- Allows to set up middlewares to respond to HTTP Requests
- Defines a routing table which is used to perform different actions based on HTTP Method and URL
- Allows to dynamically render HTML Pages based on passing arguments to templates

Project Creation

- mkdir hello_proj
- cd hello_proj
- npm init
- npm install express
- npm install body-parser
- npm i multer
- npm i nodemon -g

Modules Description

- body-parser − This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data
- multer − This is a node.js middleware for handling multipart/form-data

Hello World

```
var express = require('express');

var app = express();

app.get('/', function (req, res) {

        res.send('Hello World');

})

app.listen(8000);
```

Request Object Properties

Sr. No.	Properties & Description
1	**req.app** This property holds a reference to the instance of the express application that is using the middleware.
2	**req.baseUrl** The URL path on which a router instance was mounted.
3	**req.body** Contains key-value pairs of data submitted in the request body. By default, it is undefined, and is populated when you use body-parsing middleware such as body-parser
4	**req.cookies** When using cookie-parser middleware, this property is an object that contains cookies sent by the request.
5	**req.fresh** Indicates whether the request is "fresh." It is the opposite of req.stale.

Request Object Properties

Sr. No.	Properties & Description
6	**req.hostname** Contains the hostname from the "Host" HTTP header.
7	**req.ip** The remote IP address of the request.
8	**req.ips** When the trust proxy setting is true, this property contains an array of IP addresses specified in the "X-Forwarded-For" request header.
9	**req.originalUrl** This property is much like req.url; however, it retains the original request URL, allowing you to rewrite req.url freely for internal routing purposes.
10	**req.params** An object containing properties mapped to the named route "parameters". For example, if you have the route /user/:name, then the "name" property is available as req.params.name. This object defaults to {}.

Request Object Properties

Sr. No.	Properties & Description
11	**req.path** Contains the path part of the request URL.
12	**req.protocol** The request protocol string, "http" or "https" when requested with TLS.
13	**req.query** An object containing a property for each query string parameter in the route.
14	**req.route** The currently-matched route, a string.
15	**req.secure** A Boolean that is true if a TLS connection is established.

Request Object Properties

Sr. No.	Properties & Description
16	**req.signedCookies** When using cookie-parser middleware, this property contains signed cookies sent by the request, unsigned and ready for use.
17	**req.stale** Indicates whether the request is "stale"
18	**req.subdomains** An array of subdomains in the domain name of the request.
19	**req.xhr** A Boolean value that is true if the request's "X-Requested-With" header field is "XMLHttpRequest", indicating that the request was issued by a client library such as jQuery.

Request Object Methods

- req.accepts(types)

checks if the specified content types are acceptable, based on the request's Accept HTTP header field

```
// Accept: text/html

req.accepts('html');


// Accept: text/*, application/json

req.accepts('html');
```

Request Object Methods

- req.get(field)

returns the specified HTTP request header field

```
req.get('Content-Type');
// => "application/json"

req.get('Accept');
// => "text/plain"
```

Request Object Methods

- req.is(type)

returns true if the incoming request's "Content-Type" HTTP header field matches the MIME type specified by the type parameter

```
// Content-Type: text/html; charset=utf-8

req.is('html');

req.is('text/html');

req.is('text/*');

// => true
```

Request Object Methods

- req.param(name [, defaultValue])

returns the value of param name when present

```
// ?name=john
req.param('name')

// POST name=john
req.param('name')

// /user/john for /user/:name
req.param('name')
```

Basic Routing (routing.js)

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
        res.send('Hello GET');
})
app.post('/', function (req, res) {
        res.send('Hello POST');
})
```

Basic Routing (routing.js)

```
app.delete('/del_user', function (req, res) {
        res.send('Hello DELETE');
})
app.get('/list_user', function (req, res) {
        res.send('Page Listing');
})
app.get('/ab*cd', function(req, res) {
        res.send('Page Pattern Match');
})
app.listen(8000);
```

Router (my_routes.js)

```
var express = require('express');
var router = express.Router();
router.get('/', function(req, res){
        res.send('GET route on /abc');
});
router.post('/pqr', function(req, res){
        res.send('POST route on /abc/pqr');
});
module.exports = router;
```

Router (using_my_routes.js)

```
var express = require('Express');
var app = express();

var routes = require('./my_routes.js');

// Mount the routes as middleware at path '/abc'
app.use('/abc', routes);

app.listen(8000);
```

URL Building (url_building_01.js)

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

app.get('/:id', function(req, res){
   res.send('The id you specified is ' + req.params.id);
});

app.listen(8000);
```

URL Building (url_building_02.js)

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

app.get('/abc/:name/:id', function(req, res){
   res.send('id: ' + req.params.id + ' and name: ' + req.params.name);
});

app.listen(8000);
```

URL Building (url_building_03.js)

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

//id must be 3 digit number
app.get('/abc/:name/:id([0-9]{3})', function(req, res){
 res.send('id: ' + req.params.id + ' and name: ' + req.params.name);
});

app.listen(8000);
```

Serving Static Files

- Express provides a built-in middleware express.static to serve static files, such as images, CSS, JavaScript, etc.
- You simply need to pass the name of the directory where you keep your static assets
- If you keep your images, CSS, and JavaScript files in a directory named public, you can do this –

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

Serving Static Files

```
var express = require('express');

var app = express();

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

app.get('/', function (req, res) {

        res.send('Hello World');

})

app.listen(8000);
```

http://localhost:8000/pink_lotus.jpg

http://localhost:8000/test.html

Processing GET Request (index.html)

```
<html>
  <body>
    <form action = "http://127.0.0.1:8000/process_get" method = "GET">
      First Name: <input type = "text" name = "first_name">  <br>
      Last Name: <input type = "text" name = "last_name">
      <input type = "submit" value = "Submit">
    </form>
  </body>
</html>
```

Processing GET Request (process_get.js)

```
app.get('/index.htm', function (req, res) {
        res.sendFile( __dirname + "/" + "index.html" );
})
app.get('/process_get', function (req, res) {
        response = {
                first_name : req.query.first_name,
                last_name : req.query.last_name
        };
        res.end(JSON.stringify(response));
})
app.listen(8000);
```

Middlewares

- Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response life cycle
- Similar to filters in servlet/jsp, Message Handlers in Web API

Middleware in Action

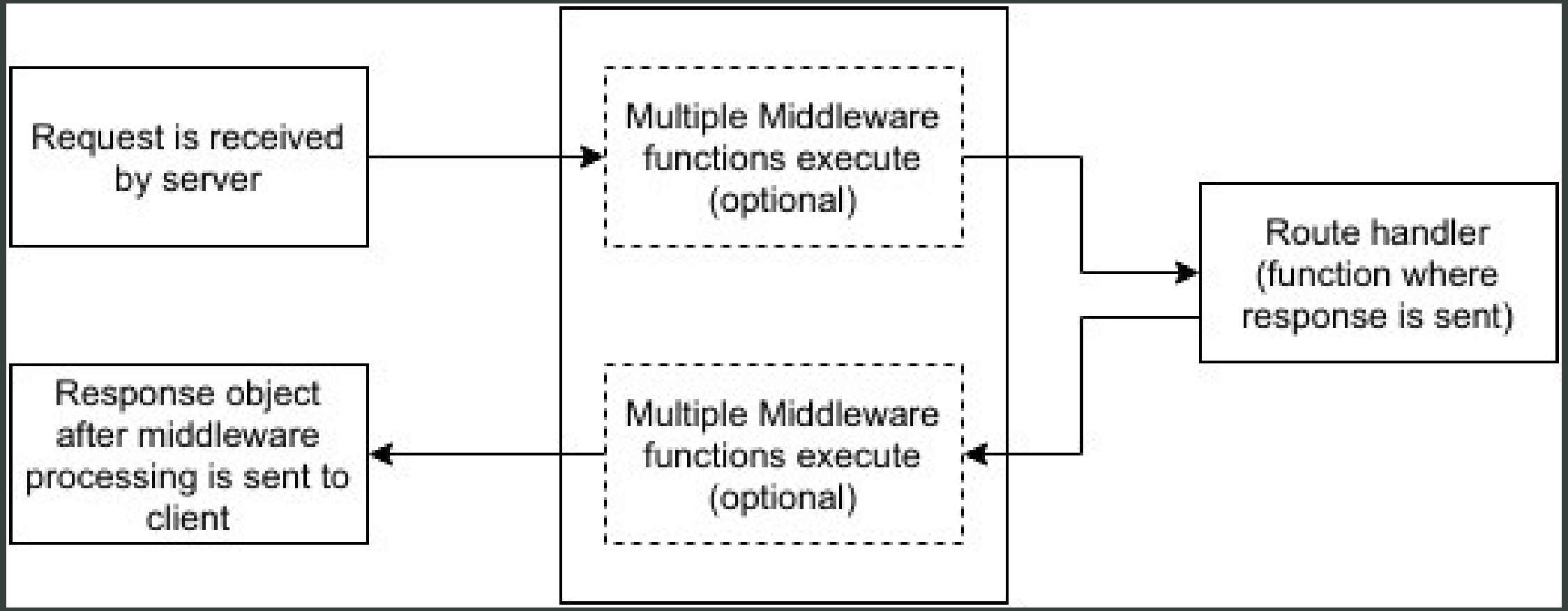

Image Source: https://www.tutorialspoint.com/expressjs/expressjs_middleware.htm

Middleware Functions

- Logging requests
- Authenticating/authorizing requests
- Parsing the body of requests
- End a request – response lifecycle
- Call the next middleware function in the stack

Built-in Middleware

- express.static serves static assets such as HTML files, images, and so on.
- express.json parses incoming requests with JSON payloads. **NOTE: Available with Express 4.16.0+**
- express.urlencoded parses incoming requests with URL-encoded payloads. **NOTE: Available with Express 4.16.0+**

Middleware Example

```
var express = require('express');
var app = express();
app.use(function(req, res, next) {
        req.reqTime = new Date().toString();
        next();
});
app.get('/', function (req, res) {
        res.send('Hello World!<br>' + req.reqTime);
})
app.listen(8000);
```

Middleware (specific to some path)

```
app.use('/abc', function(req, res, next){
        console.log( new Date().toString() );
        next();
});
app.use('/abc', function(req, res){
        res.send("I am the second middleware.");
});
app.get('/', function(req, res) {
        res.send("HTTP GET Request received.");
});
app.listen(8000);
```

Middleware Chaining

```
app.use(middlewareA)

app.use(middlewareB)

app.get('/', [middlewareC, middlewareD], handler)
```

Middleware Chaining Example 1

```
app.use(function (req, res, next) {
        console.log("first middle ware");
        next();
});
app.use(function (req, res, next) {
        console.log("second middle ware");
        next();
});
app.get('/', function (req, res) {
        res.send("page render finished");
});
```

Middleware Chaining Example 2

```
function middleHandler(req, res, next) {
    console.log("execute middle ware");          next();
}
app.use(function (req, res, next) {
        console.log("first middle ware");        next();
});
app.use(function (req, res, next) {
        console.log("second middle ware");       next();
});
app.get('/', middleHandler, function (req, res) {
        res.send("page render finished");            });
```

Middleware - Error Handling

```
app.get('/b', function(req, res, next){
        throw new Error('b failed');
});
app.use('/b', function(err, req, res, next){
        console.log('/b : error detected and passed on');
        next(err);
});
app.use(function(err, req, res, next){
        console.log('unhandled error detected: ' + err.message);
        res.send('500 - server error');
});
```

References

1. http://expressjs.com/
2. https://www.tutorialspoint.com/expressjs/index.htm