Express Middleware

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 cycle. These functions are used to modify req and res objects for tasks like parsing request bodies, adding response headers, etc.

These functions are capable of performing the below-listed tasks:

- Execution of any code
- Modify the request and the response objects.
- End applications request-response cycle.
- Call the next middleware present in the cycle.

In case, the current function doesn't terminate the request-response cycle

then it must invoke the next() function in order to pass on the control to the next available middleware function.

If not done, then the request will be left incomplete.

Below is the list of majorly used middleware in any Express.js application:

- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware
- Third-party middleware

Here is a simple example of a middleware function in action −

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

//Simple request time logger
app.use(function(req, res, next){
   console.log("A new request received at " + Date.now());

   //This function call is very important. It tells that more processing is
   //required for the current request and is in the next middleware
   function/route handler.
   next();
});

app.listen(3000);
```

The above middleware is called for every request on the server. So after every request, we will get the following message in the console −

A new request received at 1467267512545

To restrict it to a specific route (and all its subroutes), provide that route as the first argument of *app.use()*. For Example,

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

//Middleware function to log request protocol
app.use('/things', function(req, res, next){
   console.log("A request for things received at " + Date.now());
   next();
});

// Route handler that sends the response
app.get('/things', function(req, res){
   res.send('Things');
});

app.listen(3000);
```

Now whenever you request any subroute of '/things', only then it will log the time.

Order of Middleware Calls

One of the most important things about middleware in Express is the order in which they are written/included in your file; the order in which they are executed, given that the route matches also needs to be considered.

For example, in the following code snippet, the first function executes first, then the route handler and then the end function. This example summarizes how to use middleware before and after route handler; also how a route handler can be used as a middleware itself.

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

//First middleware before response is sent
app.use(function(req, res, next){
   console.log("Start");
   next();
});

//Route handler
app.get('/', function(req, res, next){
   res.send("Middle");
   next();
});
app.use('/', function(req, res){
   console.log('End');
});

app.listen(3000);
```

When we visit '/' after running this code, we receive the response as Middle and on the console.

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

some of the most commonly used middleware

body-parser:

This is used to parse the body of requests which have payloads attached to them. To mount body parser, we need to install it using

npm install body-parser

and to mount it, include the following lines (Parse incoming request bodies in a middleware before your handlers, available under the req.body property)

```
var bodyParser = require('body-parser');

//To parse URL encoded data

app.use(bodyParser.urlencoded({ extended: false }))

//To parse json data

app.use(bodyParser.json())
```

cookie-parser:

It parses Cookie header and populate req.cookies with an object keyed by cookie names. To mount cookie parser, we need to install it using

npm install cookie-parser

and to mount it, include the following lines.

```
var cookieParser = require('cookie-parser');

app.use(cookieParser())
```