

Express

Writing middleware for use in Express apps

Overview

Middleware functions are functions that have access to the [request object](#) (`req`), the [response object](#) (`res`), and the `next` function in the application's request-response cycle. The `next` function is a function in the Express router which, when invoked, executes the middleware succeeding the current middleware.

Middleware functions can perform the following tasks:

- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware in the stack.

If the current middleware function does not end the request-response cycle, it must call `next()` to pass control to the next middleware function. Otherwise, the request will be left hanging.

The following figure shows the elements of a middleware function call:

HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function, called "next" by convention.

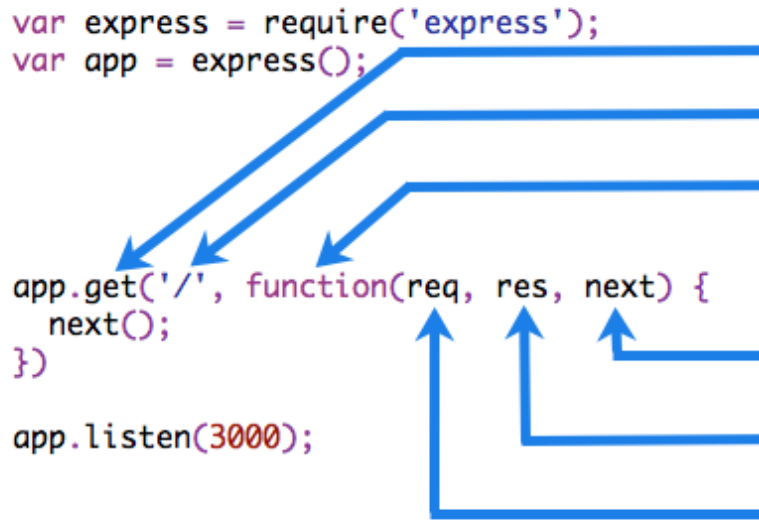
HTTP [response](#) argument to the middleware function, called "res" by convention.

HTTP [request](#) argument to the middleware function, called "req" by convention.

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

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

app.listen(3000);
```



Example

Here is an example of a simple “Hello World” Express application. The remainder of this article will define and add two middleware functions to the application: one called `myLogger` that prints a simple log message and another called `requestTime` that displays the timestamp of the HTTP request.

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

app.get('/', function (req, res) {
  res.send('Hello World!')
})

app.listen(3000)
```

Middleware function `myLogger`

Here is a simple example of a middleware function called “`myLogger`”. This function just prints “LOGGED” when a request to the app passes through it. The middleware function is assigned to a variable named `myLogger`.

```
var myLogger = function (req, res, next) {  
  console.log('LOGGED')  
  next()  
}
```

Notice the call above to `next()`. Calling this function invokes the next middleware function in the app. The `next()` function is not a part of the Node.js or Express API, but is the third argument that is passed to the middleware function. The `next()` function could be named anything, but by convention it is always named “next”. To avoid confusion, always use this convention.

To load the middleware function, call `app.use()`, specifying the middleware function. For example, the following code loads the `myLogger` middleware function before the route to the root path (`/`).

```
var express = require('express')  
var app = express()  
  
var myLogger = function (req, res, next) {  
  console.log('LOGGED')  
  next()  
}  
  
app.use(myLogger)  
  
app.get('/', function (req, res) {  
  res.send('Hello World!')  
})  
  
app.listen(3000)
```

Every time the app receives a request, it prints the message “LOGGED” to the terminal.

The order of middleware loading is important: middleware functions that are loaded first are also executed first.

If `myLogger` is loaded after the route to the root path, the request never reaches it and the app doesn't print “LOGGED”, because the route handler of the root path terminates the request-response cycle.

The middleware function `myLogger` simply prints a message, then passes on the request to the next middleware function in the stack by calling the `next()` function.

Middleware function `requestTime`

Next, we'll create a middleware function called "`requestTime`" and add a property called `requestTime` to the request object.

```
var requestTime = function (req, res, next) {  
  req.requestTime = Date.now()  
  next()  
}
```

The app now uses the `requestTime` middleware function. Also, the callback function of the root path route uses the property that the middleware function adds to `req` (the request object).

```
var express = require('express')  
var app = express()  
  
var requestTime = function (req, res, next) {  
  req.requestTime = Date.now()  
  next()  
}  
  
app.use(requestTime)  
  
app.get('/', function (req, res) {  
  var responseText = 'Hello World!<br>'  
  responseText += '<small>Requested at: ' + req.requestTime + '</small>'  
  res.send(responseText)  
})  
  
app.listen(3000)
```

When you make a request to the root of the app, the app now displays the timestamp of your request in the browser.

Because you have access to the request object, the response object, the next middleware function in the stack, and the whole Node.js API, the possibilities with middleware functions are endless.

For more information about Express middleware, see: [Using Express middleware](#).

Configurable middleware

If you need your middleware to be configurable, export a function which accepts an options object or other parameters, which, then returns the middleware implementation based on the input parameters.

File: my-middleware.js

```
module.exports = function (options) {  
  return function (req, res, next) {  
    // Implement the middleware function based on the options object  
    next()  
  }  
}
```

The middleware can now be used as shown below.

```
var mw = require('./my-middleware.js')  
  
app.use(mw({ option1: '1', option2: '2' })))
```

Refer to [cookie-session](#) and [compression](#) for examples of configurable middleware.

Documentation translations provided by [StrongLoop/IBM](#): [French](#), [German](#), [Spanish](#), [Italian](#), [Japanese](#), [Russian](#), [Chinese](#), [Traditional Chinese](#), [Korean](#), [Portuguese](#).

Community translation available for: [Slovak](#), [Ukrainian](#), [Uzbek](#), [Turkish](#) and [Thai](#).



Express is a project of the Node.js Foundation.

[Edit this page on GitHub](#).

Copyright © 2017 StrongLoop, IBM, and other expressjs.com contributors.



This work is licensed under a [Creative Commons Attribution-ShareAlike 3.0 United States License](#).