**Node.js - Express - Desmistificando o que é e como criar um Middleware** 27 Nov 2014

**Esse artigo foi traduzido!**

**Introdução by Suissa: Middleware será como um filtro para nossas requisições onde pode-se modificar o que passa por ele entregando o conteúdo modificado para o próximo filtro.**

Middleware é a ideia central por trás do processo de requisição e roteamento do Express.js. Ao compreender como funciona o middleware, você pode criar aplicativos mais fáceis de manter e com menos código. No momento em que um pedido é recebido por um app Express.js, ele aciona várias funções referidas como middleware.

**O que é middleware?**

Middleware é qualquer número de funções que são invocadas pela camada de roteamento Express.js antes de seu manipulador final e assim fica no meio entre um pedido cru e a rota pretendida. Nós muitas vezes nos referimos a estas funções como a *middleware stack* (stack = pilha), uma vez que são sempre invocados na ordem em que são adicionados.

Pegue o app mais básico Express.js:

var app = express();

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

res.send('Hello World!');

});

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

res.send('Nope.. nothing to see here');

});

O app Express.js acima responderá com Olá mundo! para uma solicitação na / e Nope.. nothing to see here para solicitações em /help.

Agora, suponha que você deseja gravar toda vez que receber uma solicitação. Você poderia ir para cada rota e adicionar a sua lógica de *logging*, ou você pode usar middleware!

var app = express();

app.use(function(req, res, next) {

console.log('%s %s', req.method, req.url);

next();

});

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

res.send('Hello World!');

});

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

res.send('Nope.. nothing to see here');

});

Aqui nós adicionamos uma nova função para invocar a cada solicitação via app.use(). Existem algumas coisas importantes a serem observados sobre esta nova função:

* Middleware é uma função, como manipuladores de rotas e é chamado da mesma maneira.
* A assinatura da função parece idêntica ao que usamos em nossas rotas.
* Nós adicionamos essa função antes de nossos dois manipuladores de rota, porque queremos que ele execute antes de qualquer um.
* Nós temos pleno acesso aos mesmos objetos de *request* e *response*que vai encontrar seu caminho para as rotas.
* Foi utilizado um terceiro parâmetro, denominado next como uma função para indicar que o nosso middleware foi terminado.
* Podemos adicionar mais middlewares acima ou abaixo utilizando a mesma API.

Vamos revisitar vários desses itens em mais detalhes. Este é apenas um exemplo básico mostrando como *logar* cada requisição que sua aplicação receber. Middleware pode ser tão simples como uma linha, ou tão complexo como a manipulação da sessão. Middlware é comumente usado para executar tarefas como *parsear* o *body* para URL ou JSON, *parsear* cookies para tratamento básico, ou mesmo a construção de módulos JavaScript *on the fly*.

Ao adicionar um novo pedaço de middleware, leia a documentação sobre como o middleware pode afetar o objeto de *request* ou *response*. Lembre-se que middleware tem acesso a ambos os objetos e pode modificar ou adicionar propriedades.

**O argumerto next()**

Anteriormente, você viu o middleware chamar o argumento next que foi fornecido como o terceiro argumento para a função middleware. Para que o pedido possa continuar o processamento (outro middleware ou até mesmo os nossos manipuladores de rota) você deve chamar esse argumento; simplesmente deixar a execução da função finalizar não é suficiente.

A razão de next ser exposta e precisa ser explicitamente chamada é para o middleware, que realiza *I/O* ou operações assíncronas. Express.js tem nenhuma maneira de saber quando a operação estiver concluída antes de poder continuar para o próximo middleware ou rota.

Imagine um cenário onde você queria carregar uma sessão a partir do banco de dados antes de processar qualquer solicitação.

app.use(function(req, res, next) {

db.load(function(err, session) {

if (err) {

return next(err);

}

else if (!session) {

return next(new Error('no session found'));

}

req.session = session;

next();

});

});

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

// we can use req.session because middleware HAD to run first

});

Sem a obrigação de chamar next, Express.js não teria nenhuma maneira de saber quando o sua pesquisa no banco de dados de foi completada. Em nosso exemplo, você chama next uma vez que você tem uma sessão carregada a partir do banco de dados informando a camada de roteamento do Express.js que o próximo middleware ou rota pode ser invocado.

**Limitando caminhos para o Middleware**

Até agora, você viu como adicionar middleware para cada *request*, mas que se você quiser limitar middleware para determinados caminhos. Vamos dizer que as rotas sob /users precisam de algum middleware especial. Como você pode conseguir isso?

app.use('/users', function(req, res, next) {

// invoked for any request starting with /users

next();

});

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

app.use() aceita um parâmetro de caminho opcional, assim como o app.get() e outras rotas. A principal diferença, no entanto, é como este parâmetro do caminho é tratado, como um prefixo. Se tivéssemos de passar /users a uma app.get(), a nossa função só será invocada quando alguém visitou exatamente /users, no entanto, quando passamos /userspara app.use(), qualquer pedido que começa com /users irá invocar a nossa função!

**Adendo: É assim que o Express 4 trabalha com os módulos de rotas, como um middleware.**

Nós podemos facilmente obter o sub-caminho em /users via a propriedade req.path que está definida pela camada de roteamento do Express.js.

app.use('/users', function(req, res, next) {

// req.path will be the req.url with the /users prefix stripped

console.log('%s', req.path);

next();

});

Quando um pedido de /users/daily é processado, req.path será /dailyno middleware. Nota-se que req.path será /users/daily em nosso manipulador da rota porque req.path é uma propriedade do método .use().

**Armadilhas do Middleware**

Aqui estão algumas armadilhas comuns para ficar atento ao trabalhar com middleware.

**Ordem importa**

Lembre-se que a ordem importa. O exemplo a seguir mostra um middleware que não irá executar para solicitações **GET**, mas irá para solicitações **POST** simplesmente ordenando-o de forma diferente.

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

app.use(function(req, res, next) {

next();

});

app.post('/', function(req, res) { res.send('bye'); });

Embora nem sempre uma armadilha, é importante compreender que, embora app.use() é chamado para cada verbo **HTTP**, middleware é processado em ordem com a manipulação de rota. Desde que o nosso manipulador de rota de **GET** responda ao pedido, o nosso middleware nunca irá rodar para solicitações **GET** , mas o nosso manipulador do **POST** vem após o middleware, permitindo assim que ele seja invocado pela primeira vez.

**Explicação do Suissa:** Como o GET é executado antes ele nunca cairá no middleware abaixo e para a requisição do POST o middleware é executado antes de chegar nele.

**Esquecendo next()**

Se você acabou de adicionar alguns middlewares e ao testar sua página ou API parece travar, ou nunca está respondendo, então você provavelmente esqueceu de chamar next(). Como o roteamento Express.js não tem idéia do que o seu middleware é feito, ele nunca irá conseguir chamar qualquer middleware ou rotas que o seguem.

**Sobrescrevendo propriedades**

Lembre-se que o argumentos *request* e *response* para todos os middleware e rotas são a mesma instância. Isto significa que se você tem dois middleware que modificam as propriedades do objeto de diferentes maneiras, eles podem causar erros em seu app. Esteja atento das modificações que seu middleware faz nas propriedades.

**Conclusão**

Aqui está uma breve recapitulação do que foi abordado neste post:

* Middleware é sempre invocado na ordem adicionado.
* Você pode ter múltiplos middlewares para o mesmo caminho.
* Middleware tem pleno acesso aos objetos de *request* e *response*.
* Esteja atento sobre middleware substituindo campos de outro middleware.
* Middlewares podem ser pulados usando a função next.

E aqui estão algumas dicas úteis para considerar ao usar Express.js:

* Sempre use a assinatura req, res, e next, mesmo em rotas em que você não chama next.
* Passe erros para next de modo que eles podem ser tratados separadamente.
* Olhe em express.Router() para *middleware stacks* isolados.
* Middleware pode ser usado para limitar ou registrar o tempo total de resposta, se você quiser resgatar.

*ps do Suissa: é com um middleware que você irá trabalhar com sessões de usuário.*

Artigo original: <https://blog.safaribooksonline.com/2014/03/10/express-js-middleware-demystified/>

Share this post: 

**Comentários**

**Jean Carlo Nascimento aka Suissa**

Evangelista de tecnologias como NoSQL e Node.js. Formado pela UTF-PR, fez especialização em Desenvolvimento Web. Organiza os eventos Sampa.js, FrontinSampa e FrontinPG. @osuissa facebook.com/suissa

[**Subscribe!**](http://nomadev.com.br/rss/)

All content copyright [Nomadev - O blog de um dev nômade](http://nomadev.com.br/) © 2013 • All rights