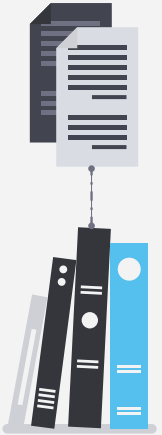
# Middlewares

Visão geral

# Definição



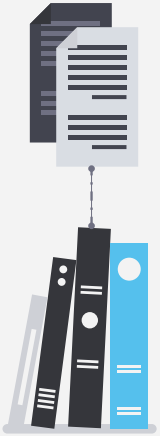
“Funções de Middleware são funções que têm acesso ao objeto de solicitação (req), o objeto de resposta (res), e a próxima função de middleware no ciclo solicitação-resposta do aplicativo. A próxima função middleware é comumente denotada por uma variável chamada next.”

Funções de middleware podem executar as seguintes tarefas:

- Executar qualquer código.
- Fazer mudanças nos objetos de solicitação e resposta.
- Encerrar o ciclo de solicitação-resposta.
- Chamar o próximo middleware na pilha.

https://expressjs.com/pt-br/guide/writing-middleware.html

# Como usar



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

Chamada desta função chama a próxima função de middleware no aplicativo. A função next() não faz parte do Node.js ou da API Express, mas é o terceiro argumento que é passado para a função de middleware. A função next() poderia ter qualquer nome, mas por convenção ela é sempre chamada de “next”. Para evitar confusão, sempre use esta convenção.

# Como usar

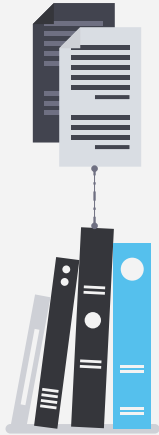
```
var express = require('express');
var app = express();

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);
```

Para carregar a função de middleware, chame app.use(), especificando a função de middleware. Por exemplo, o código a seguir carrega a função de middleware do myLogger antes da rota para o caminho raiz (/).

A ordem de carregamento do middleware é importante: funções de middleware que são carregadas primeiro também são executadas primeiro.

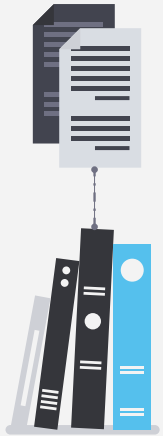
Se myLogger é carregada após a rota para o caminho raiz, a chamada nunca chegará a ela e o aplicativo não imprimirá “LOGGED”.

A função de middleware myLogger simplesmente imprime uma mensagem, e em seguida passa a solicitação para a próxima função de middleware na pilha chamando a função next().



# Tipos de middleware

# Middleware de nível do aplicativo



```
var app = express();

app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});
```

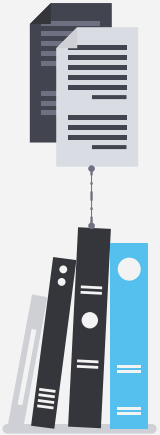
Vincule middlewares de nível do aplicativo a uma instância do objeto app usando as funções app.use() e app.METHOD(), onde METHOD é o método HTTP da solicitação que a função de middleware manipula (como GET, PUT, ou POST) em letras minúsculas.

# Middleware de nível de roteador

Middlewares de nível de roteador funcionam da mesma forma que os middlewares de nível do aplicativo, mas estão vinculados a uma instância do express.Router().

```
var app = express();
var router = express.Router();

// a middleware function with no mount path. This code is executed for every request to the router
router.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});

// a middleware sub-stack shows request info for any type of HTTP request to the /user/:id path
router.use('/user/:id', function(req, res, next) {
  console.log('Request URL:', req.originalUrl);
  next();
}, function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});

// a middleware sub-stack that handles GET requests to the /user/:id path
router.get('/user/:id', function (req, res, next) {
  // if the user ID is 0, skip to the next router
  if (req.params.id == 0) next('route');
  // otherwise pass control to the next middleware function in this stack
  else next(); //
}, function (req, res, next) {
  // render a regular page
  res.render('regular');
});

// handler for the /user/:id path, which renders a special page
router.get('/user/:id', function (req, res, next) {
  console.log(req.params.id);
  res.render('special');
});

// mount the router on the app
app.use('/', router);
```

# Middleware de manipulação de erros



```
app.use(function(err, req, res, next) {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
```

Middlewares de manipulação de erros sempre levam quatro argumentos. Você deve fornecer quatro argumentos para identificá-lo como uma função de middleware de manipulação de erros. Mesmo se você não precisar usar o objeto next, você deve especificá-lo para manter a assinatura. Caso contrário, o objeto next será interpretado como um middleware comum e a manipulação de erros falhará.