Black Lives Matter.
Support the Equal Justice Initiative.

Cette traduction fournie par StrongLoop / IBM.

Il se peut que ce document soit obsolète par rapport à la documentation en anglais. Pour connaître les mises à jour les plus récentes, reportez-vous à la documentation en anglais.

Ecriture de middleware utilisable dans les applications Express

Présentation

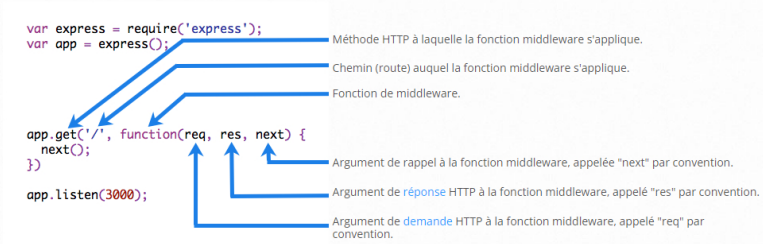
Les fonctions de ***middleware*** sont des fonctions qui peuvent accéder à l'objet Request (`req`), l'objet response (`res`) et à la fonction middleware suivant dans le cycle demande-réponse de l'application. La fonction middleware suivant est couramment désignée par une variable nommée `next`.

Les fonctions middleware effectuent les tâches suivantes :

- Exécuter tout type de code.
- Apporter des modifications aux objets de demande et de réponse.
- Terminer le cycle de demande-réponse.
- Appeler le middleware suivant dans la pile.

Si la fonction middleware en cours ne termine pas le cycle de demande-réponse, elle doit appeler la fonction `next()` pour transmettre le contrôle à la fonction middleware suivant. Sinon, la demande restera bloquée.

L'exemple suivant montre les éléments d'un appel de fonction middleware:



Voici un exemple d'une application Express "Hello World" simple, pour laquelle vous allez définir deux fonctions middleware :

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

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

app.listen(3000);
```

Développement

Voici un exemple simple de fonction middleware appelée "myLogger". Cette fonction imprime simplement "LOGGED" lorsqu'une demande traverse l'application. La fonction middleware est affectée à une variable nommée `myLogger`.

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

Notez l'appel ci-dessus de la fonction `next()`, qui appelle la fonction middleware suivant dans l'application. La fonction `next()` ne fait pas partie du Node.js ou de l'API Express, mais c'est le troisième argument qui est transmis à la fonction middleware. La fonction `next()` peut porter n'importe quel nom, mais par convention elle est toujours appelée "next". Pour éviter toute confusion, il est préférable de respecter cette convention.

Pour charger la fonction middleware, appelez `app.use()` en spécifiant la fonction middleware. Par exemple, le code suivant charge la fonction middleware `myLogger` avant la route au chemin racine (/).

```
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);
```

Chaque fois que l'application reçoit une demande, elle imprime le message "LOGGED" sur le terminal.

L'ordre de chargement des middleware est important : les fonctions middleware chargées en premier sont également exécutées en premier.

Si `myLogger` est chargé après la route au chemin racine, la demande ne l'atteindra jamais et l'application n'imprimera pas "LOGGED", car le gestionnaire de route du chemin racine interrompra le cycle de demande-réponse.

La fonction middleware `myLogger` imprime simplement un message, puis traite la demande à la fonction middleware suivant dans la pile en appelant la fonction `next()`.

L'exemple suivant ajoute une propriété appelée `requestTime` à l'objet Request. Nous nommerons cette fonction middleware "requestTime".

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

L'application utilise désormais la fonction middleware `requestTime`. De plus, la fonction callback de la route du chemin racine utilise la propriété que la fonction middleware ajoute à `req` (l'objet Request).

```
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!';
  responseText += 'Requested at: ' + req.requestTime + '';
  res.send(responseText);
});

app.listen(3000);
```

Si vous effectuez une demande à la racine de l'application, cette dernière affiche maintenant l'horodatage de la demande dans le navigateur.

Puisque vous avez accès à l'objet Request, à l'objet Response, à la fonction middleware suivant dans la pile et à l'API Node.js complète, le champ des possibles avec les fonctions middleware est infini.

Pour plus d'informations sur les middleware Express, voir Utilisation de middleware Express.

Star 58,580 Express est un projet de la Fondation OpenJS. Consultez le site Web GitHub. Copyright © StrongLoop, Inc., et autres contributeurs expressjs.com.
Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution - Partage dans les Mêmes Conditions 3.0 États-Unis.