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.

Routage

Routage fait référence à la définition de points finaux d'application (URI) et à la façon dont ils répondent aux demandes client. Pour une introduction au routage, voir Basic routing.

Le code suivant est un exemple de routage très basique.

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

// respond with "hello world" when a GET request is made to the homepage
app.get('/', function(req, res) {
  res.send('hello world');
});
```

Méthodes de routage

Une méthode de routage est dérivée de l'une des méthodes HTTP, et est liée à une instance de la classe `express`.

Le code suivant est un exemple de routes qui sont définies pour les méthodes GET et POST jusqu'à la route de l'application.

```
// GET method route
app.get('/', function (req, res) {
  res.send('GET request to the homepage');
});

// POST method route
app.post('/', function (req, res) {
  res.send('POST request to the homepage');
});
```

Express prend en charge les méthodes de routage suivantes qui correspondent aux méthodes HTTP : `get`, `post`, `put`, `head`, `delete`, `options`, `trace`, `copy`, `lock`, `mkcol`, `move`, `purge`, `propfind`, `proppatch`, `unlock`, `report`, `mkactivity`, `checkout`, `merge`, `m-search`, `notify`, `subscribe`, `unsubscribe`, `patch`, `search`, and `connect`.

Pour router des méthodes qui se traduisent par des noms de variables JavaScript non valides, utilisez la notation entre crochets. For example, `app['m-search']('/', function ...`

Il existe une méthode de routage spéciale, `app.all()`, qui n'est pas dérivée d'une méthode HTTP. Cette méthode est utilisée pour charger des fonctions middleware à un chemin d'accès pour toutes les méthodes de demande.

Dans l'exemple suivant, le gestionnaire sera exécuté pour les demandes de "/secret", que vous utilisiez GET, POST, PUT, DELETE ou toute autre méthode de demande HTTP prise en charge dans le module http.

```
app.all('/secret', function (req, res, next) {
  console.log('Accessing the secret section ...');
  next(); // pass control to the next handler
});
```

Chemins de routage

Les chemins de routage, combinés à une méthode de demande, définissent les noeuds finaux sur lesquels peuvent être effectuées les demandes. Les chemins de routage peuvent être des chaînes, des masques de chaîne ou des expressions régulières.

Express utilise path-to-regexp pour faire correspondre les chemins de routage ; pour connaître toutes les façons de définir des chemins de routage, voir la documentation path-to-regexp. Express Route Tester est un outil pratique permettant de tester des routes Express de base, bien qu'il ne prenne pas en charge le filtrage par motif.

Les chaînes de requête ne font pas partie du chemin de routage.

Il s'agit d'exemples de chemins de routage basés sur des chaînes.

Ce chemin de routage fera correspondre des demandes à la route racine, `/`.

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

Ce chemin de routage fera correspondre des demandes à `/about`.

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

Ce chemin de routage fera correspondre des demandes à `/random.text`.

```
app.get('/random.text', function (req, res) {
  res.send('random.text');
});
```

Il s'agit d'exemples de chemins de routage basés sur des masques de chaîne.

Ce chemin de routage fait correspondre `acd` et `abcd`.

```
app.get('/ab?cd', function(req, res) {
  res.send('ab?cd');
});
```

Ce chemin de routage fait correspondre `abcd`, `abbcd`, `abbbcd`, etc.

```
app.get('/ab+cd', function(req, res) {
  res.send('ab+cd');
});
```

Ce chemin de routage fait correspondre `abcd`, `abxcd`, `abRABDOMcd`, `ab123cd`, etc.

```
app.get('/ab*cd', function(req, res) {
  res.send('ab*cd');
});
```

Ce chemin de routage fait correspondre `/abe` et `/abcde`

Ce chemin de routage fait correspondre `/abe` et `/abcde`.

```
app.get('/ab(cd)?e', function(req, res) {
 res.send('ab(cd)?e');
});
```

> Les caractères ?, +, * et () sont des sous-ensembles de leur expression régulière équivalente. Le trait d'union (-) et le point (.) sont interprétés littéralement par des chemins d'accès basés sur des chaînes.

Exemples de chemins de routage basés sur des expressions régulières :

Ce chemin de routage fera correspondre tout élément dont le nom de chemin comprend la lettre "a".

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

Ce chemin de routage fera correspondre `butterfly` et `dragonfly`, mais pas `butterflyman`, `dragonfly man`, etc.

```
app.get(/.*fly$/, function(req, res) {
  res.send('/.*fly$/');
});
```

Gestionnaires de routage

Vous pouvez fournir plusieurs fonctions de rappel qui se comportent comme des middleware pour gérer une demande. La seule exception est que ces fonctions de rappel peuvent faire appel à `next('route')` pour ignorer les rappels de route restants. Vous pouvez utiliser ce mécanisme pour imposer des conditions préalables sur une route, puis passer aux routes suivantes si aucune raison n'est fournie pour traiter la route actuelle.

Les gestionnaires de route se trouvent sous la forme d'une fonction, d'un tableau de fonctions ou d'une combinaison des deux, tel qu'indiqué dans les exemples suivants.

Une fonction de rappel unique peut traiter une route. Par exemple :

```
app.get('/example/a', function (req, res) {
  res.send('Hello from A!');
});
```

Plusieurs fonctions de rappel peuvent traiter une route (n'oubliez pas de spécifier l'objet `next`). Par exemple :

```
app.get('/example/b', function (req, res, next) {
  console.log('the response will be sent by the next function ...');
  next();
}, function (req, res) {
  res.send('Hello from B!');
});
```

Un tableau de fonctions de rappel peut traiter une route. Par exemple :

```
var cb0 = function (req, res, next) {
  console.log('CB0');
  next();
}

var cb1 = function (req, res, next) {
  console.log('CB1');
  next();
}

var cb2 = function (req, res) {
  res.send('Hello from C!');
}

app.get('/example/c', [cb0, cb1, cb2]);
```

Une combinaison de fonctions indépendantes et des tableaux de fonctions peuvent gérer une route. Par exemple :

```
var cb0 = function (req, res, next) {
  console.log('CB0');
  next();
}

var cb1 = function (req, res, next) {
  console.log('CB1');
  next();
}

app.get('/example/d', [cb0, cb1], function (req, res, next) {
  console.log('the response will be sent by the next function ...');
  next();
}, function (req, res) {
  res.send('Hello from D!');
});
```

Méthodes de réponse

Les méthodes de l'objet de réponse (`res`) décrites dans le tableau suivant peuvent envoyer une réponse au client, et mettre fin au cycle de demande-réponse. Si aucune de ces méthodes n'est appelée par un gestionnaire de routage, la demande du client restera bloquée.

Méthode	Description
res.download()	Vous invite à télécharger un fichier.
res.end()	Met fin au processus de réponse.
res.json()	Envoie une réponse JSON.
res.jsonp()	Envoie une réponse JSON avec une prise en charge JSONP.
res.redirect()	Redirige une demande.
res.render()	Génère un modèle de vue.
res.send()	Envoie une réponse de divers types.
res.sendFile()	Envoie une réponse sous forme de flux d'octets.
res.sendStatus()	Définit le code de statut de réponse et envoie sa représentation sous forme de chaîne comme corps de réponse.

app.route()

Vous pouvez créer des gestionnaires de routage sous forme de chaîne pour un chemin de routage en utilisant `app.route()`. Etant donné que le chemin est spécifié à une seul emplacement, la création de routes modulaires est utile car elle réduit la redondance et les erreurs. Pour plus d'informations sur les routes, voir la documentation Router().

Voici quelques exemples de gestionnaires de chemin de chaînage définis à l'aide de `app.route()`.

```
app.route('/book')
  .get(function(req, res) {
    res.send('Get a random book');
  })
  .post(function(req, res) {
    res.send('Add a book');
  })
  .put(function(req, res) {
    res.send('Update the book');
  });
```

express.Router

Utilisez la classe `express.Router` pour créer des gestionnaires de route modulaires et pouvant être montés. Une instance `Router` est un middleware et un système de routage complet ; pour cette raison, elle est souvent appelée "mini-app".

L'exemple suivant créé une routeur en tant que module, charge une fonction middleware, définit des routes et monte le module de routeur sur un chemin dans l'application principale.

Créez un fichier de routage nommé `birds.js` dans le répertoire app, avec le contenu suivant :

```
var express = require('express');
var router = express.Router();

// middleware that is specific to this router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});
// define the home page route
router.get('/', function(req, res) {
  res.send('Birds home page');
});
// define the about route
router.get('/about', function(req, res) {
  res.send('About birds');
});

module.exports = router;
```

Puis, chargez le module de routage dans l'application :

```
var birds = require('./birds');
...
app.use('/birds', birds);
```

L'application pourra dorénavant gérer des demandes dans `/birds` et `/birds/about`, et appeler la fonction middleware `timeLog` spécifique à la route.