Express

Esta tradução fornecida pelo StrongLoop / IBM.

Este documento pode estar desatualizado em relação à documentação em Inglês. Para obter as atualizações mais recentes, consulte a documentação em Inglês.

Roteamento

O ***Roteamento*** refere-se à definição de terminais do aplicativo (URIs) e como eles respondem às solicitações do cliente. Para obter uma introdução a roteamento, consulte Roteamento básico.

O código a seguir é um exemplo de uma rota muito básica.

```
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');
});
```

Métodos de roteamento

Um método de roteamento é derivado a partir de um dos métodos HTTP, e é anexado a uma instância da classe `express`.

o código a seguir é um exemplo de rotas para a raiz do aplicativo que estão definidas para os métodos GET e POST.

```
// 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');
});
```

O Express suporta os seguintes métodos de roteamento que correspondem aos métodos 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`, e `connect`.

> Para métodos de rota que são traduzidos para nomes de variáveis inválidas no Javascript, use a notação de colchetes. Por exemplo, `app['m-search']('/', function ...`

Existe um método de roteamento especial, `app.all()`, que não é derivado de nenhum método HTTP. Este método é usado para carregar funções de middleware em um caminho para todos os métodos de solicitação.

No exemplo a seguir, o manipulador irá ser executado para solicitações para "/secret" se você estiver usando GET, POST, PUT, DELETE, ou qualquer outro método de solicitação HTTP que é suportado no módulo http.

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

Caminhos de rota

Caminhos de rota, em combinação com os métodos de solicitação, definem os terminais em que as solicitações podem ser feitas. Caminhos de rota podem ser sequências de caracteres, padrões de sequência, ou expressões regulares.

> O Express usa o path-to-regexp para verificar a correspondência de caminhos de rota; consulte a documentação do path-to-regexp para obter todas as possibilidades nas definições de caminhos de rota. O Express Route Tester é uma ferramenta útil para testar rotas básicas do Express, apesar de não suportar a correspondência de padrões.

> Sequências de consulta não fazem parte dos caminhos de rota.

Aqui estão alguns exemplos de caminhos de rota baseados em sequências de caracteres

Este caminho de rota corresponde a solicitações à rota raiz, `/`.

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

Este caminho de rota irá corresponder a solicitações ao `/about`.

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

Este caminho de rota irá corresponder a solicitações ao `/random.text`.

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

```
});
```

Aqui estão alguns exemplos de caminhos de rota baseados em padrões de sequência

Este caminho de rota irá corresponder ao `acd` e `abcd`.

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

Este caminho de rota irá corresponder ao `abcd`, `abbcd`, `abbbcd`, e assim por diante.

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

Este caminho de rota irá corresponder ao `abcd`, `abxcd`, `abRABDOMcd`, `ab123cd`, e assim por diante.

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

Este caminho de rota irá corresponder ao `/abe` e `/abcde`.

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

> Os caracteres ?, +, *, e () são subconjuntos de suas contrapartes em expressões regulares. O hífen (-) e o ponto (.) são interpretados literalmente por caminhos baseados em sequências de caracteres.

Exemplos de caminhos de rota baseados em expressões regulares:

Este caminho de rota irá corresponder a qualquer coisa com um “a” no nome.

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

Este caminho de rota irá corresponder a `butterfly` e `dragonfly`, mas não a `butterflyman`, `dragonfly man`, e assim por diante.

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

```
});
```

Manipuladores de rota

É possível fornecer várias funções de retorno de chamada que se comportam como middleware para manipular uma solicitação. A única exceção é que estes retornos de chamada podem chamar `next('route')` para efetuar um bypass nos retornos de chamada da rota restantes. É possível usar este mecanismo para impor pré-condições em uma rota, e em seguida passar o controle para rotas subsequentes se não houveram razões para continuar com a rota atual.

Manipuladores de rota podem estar na forma de uma função, uma matriz de funções, ou combinações de ambas, como mostrado nos seguintes exemplos.

Uma única função de retorno de chamada pode manipular uma rota. Por exemplo:

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

Mais de uma função de retorno de chamada pode manipular uma rota (certifique-se de especificar o objeto `next` object). Por exemplo:

```
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!');
});
```

Uma matriz de funções de retorno de chamada podem manipular uma rota. Por exemplo:

```
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]);
```

Uma combinação de funções independentes e matrizes de funções podem manipular uma rota. Por exemplo:

```
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!');
});
```

Métodos de resposta

Os métodos do objeto de resposta (`res`) na seguinte tabela podem enviar uma resposta ao cliente, e finalizar o ciclo solicitação-resposta. Se nenhum destes métodos forem chamados a partir de um manipulador de rota, a solicitação do cliente será deixada em suspenso.

Método	Descrição
res.download()	Solicita que seja efetuado o download de um arquivo
res.end()	Termina o processo de resposta.
res.json()	Envia uma resposta JSON.
res.jsonp()	Envia uma resposta JSON com suporta ao JSONP.
res.redirect()	Redireciona uma solicitação.
res.render()	Renderiza um modelo de visualização.
res.send()	Envia uma resposta de vários tipos.
res.sendFile	Envia um arquivo como um fluxo de octeto.
res.sendStatus()	Configura o código do status de resposta e envia a sua representação em sequência de caracteres como o corpo de resposta.

app.route()

É possível criar manipuladores de rota encadeáveis para um caminho de rota usando o `app.route()`. Como o caminho é especificado em uma localização única, criar rotas modulares é útil, já que reduz redundâncias e erros tipográficos. Para obter mais informações sobre rotas, consulte: documentação do Router().

Aqui está um exemplo de manipuladores de rotas encadeáveis que são definidos usando `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

Use a classe `express.Router` para criar manipuladores de rota modulares e montáveis. Uma instância de `Router` é um middleware e sistema de roteamento completo; por essa razão, ela é frequentemente referida como um "mini-aplicativo"

O seguinte exemplo cria um roteador como um módulo, carrega uma função de middleware nele, define algumas rotas, e monta o módulo router em um caminho no aplicativo principal.

Crie um arquivo de roteador com um arquivo chamado `birds.js` no diretório do aplicativo, com o seguinte conteúdo:

```
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;
```

Em seguida, carregue o módulo roteador no aplicativo:

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

O aplicativo será agora capaz de manipular solicitações aos caminhos `/birds` e `/birds/about`, assim como chamar a função de middleware `timeLog` que é específica para a rota.