EBook Gratis

APRENDIZAJE express

Free unaffiliated eBook created from
Stack Overflow contributors.

#express

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: express

It is an unofficial and free express ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official express.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con Express

Observaciones

Express.js es, en palabras de los desarrolladores, un "marco web minimalista, rápido y no insinuado para Node.js."

Diseñado para ser mínimo y flexible, Express ofrece un conjunto de características para crear aplicaciones web y móviles. Desde métodos HTTP hasta middleware incorporado, Express está diseñado para proporcionarle las funciones que necesita para crear una aplicación web o móvil en Node.js.

Si desea crear una aplicación en Node.js Express es una excelente opción, ya sea que use vanilla Express o uno de los muchos marcos basados en Express o construidos sobre Express. Algunos de estos marcos se pueden encontrar aquí .

Versiones

Versiones desde **aquí .**

Versión	Notas	Fecha de lanzamiento
4.15.3		2017-05-16
4.15.2		2017-03-06
4.15.1		2017-03-05
4.15.0		2017-03-01
4.14.1		2017-01-28
4.14.0		2016-06-16
4.13.4		2016-01-21
4.13.3		2015-08-02
4.13.2		2015-07-31
4.13.1		2015-07-05
4.13.0		2015-06-20
4.12.4		2015-05-17
4.12.3		2015-03-17

Versión	Notas	Fecha de lanzamiento
4.12.2		2015-03-02
4.12.1		2015-03-01
4.12.0		2015-02-23
4.11.2		2015-01-20
4.11.1		2015-01-20
4.11.0		2015-01-13
4.10.8		2015-01-13
4.10.7		2015-01-04
4.10.6		2014-12-12
4.10.5		2014-12-10
4.10.4		2014-11-24
4.10.3		2014-11-23
4.10.2		2014-11-09
4.10.1		2014-10-28
4.10.0		2014-10-23
4.9.8		2014-10-17
4.9.7		2014-10-10
4.9.6		2014-10-08
4.9.5		2014-09-24
4.9.4		2014-09-19
4.9.3		2014-09-18
4.9.2		2014-09-17
4.9.1		2014-09-16
4.9.0		2014-09-08
4.8.8		2014-09-04

Versión	Notas	Fecha de lanzamiento
4.8.7		2014-08-29
4.8.6		2014-08-27
4.8.5		2014-08-18
4.8.4		2014-08-14
4.8.3		2014-08-10
4.8.2		2014-08-07
4.8.1		2014-08-06
4.8.0		2014-08-05
4.7.4		2014-08-04
4.7.3		2014-08-04
4.7.2		2014-07-27
4.7.1		2014-07-26
4.7.0		2014-07-25
4.6.1		2014-07-12
4.6.0		2014-07-11
4.5.1		2014-07-06
4.5.0		2014-07-04
4.4.5		2014-06-26
4.4.4		2014-06-20
4.4.3		2014-06-11
4.4.2		2014-06-09
4.4.1		2014-06-02
4.4.0		2014-05-30
4.3.2		2014-05-28
4.3.1		2014-05-23

Versión	Notas	Fecha de lanzamiento
4.3.0		2014-05-21
4.2.0		2014-05-11
4.1.2		2014-05-08
4.1.1		2014-04-27
4.1.0		2014-04-24
4.0.0		2014-04-09
3.21.2	De aquí a	2015-07-31
3.21.1	3.18.6 fechas	2015-07-05
3.21.0	parecer mal	2015-06-18
3.20.3		2015-05-17
3.20.2		2015-03-16
3.20.1		2015-02-28
3.20.0		2015-02-18
3.19.2		2015-02-01
3.19.1		2015-01-20
3.19.0		2015-01-09
3.18.6		2014-12-12
3.18.5		2014-12-11
3.18.4		2014-11-23
3.18.3		2014-11-09
3.18.2		2014-10-28
3.18.1		2014-10-22
3.18.0		2014-10-17
3.17.8		2014-10-15
3.17.7		2014-10-08

Versión	Notas	Fecha de lanzamiento
3.17.6		2014-10-02
3.17.5		2014-09-24
3.17.4		2014-09-19
3.17.3		2014-09-18
3.17.2		2014-09-15
3.17.1		2014-09-08
3.17.0		2014-09-08
3.16.10		2014-09-04
3.16.9		2014-08-29
3.16.8		2014-08-27
3.16.7		2014-08-18
3.16.6		2014-08-14
3.16.5		2014-08-11
3.16.4		2014-08-10
3.16.3		2014-08-07
3.16.2		2014-08-07
3.16.1		2014-08-06
3.16.0		2014-08-05
3.15.3		2014-08-04
3.15.2		2014-07-27
3.15.1		2014-07-26
3.15.0		2014-07-22
3.14.0		2014-07-11
3.13.0		2014-07-03
3.12.1		2014-06-26

Versión	Notas	Fecha de lanzamiento
3.12.0		2014-06-21
3.11.0		2014-06-19
3.10.5		2014-06-11
3.10.4		2014-06-09
3.10.3		2014-06-05
3.10.2		2014-06-03
3.10.1		2014-06-03
3.10.0		2014-06-02
3.9.0		2014-05-30
3.8.1		2014-05-27
3.8.0		2014-05-21
3.7.0		2014-05-18
3.6.0		2014-05-09
3.5.3		2014-05-08
3.5.2		2014-04-24
3.5.1		2014-03-25
3.5.0		2014-03-06
3.4.8		2014-01-13
3.4.7		2013-12-10
3.4.6		2013-12-01
3.4.5		2013-11-27
3.4.4		2013-10-29
3.4.3		2013-10-23
3.4.2		2013-10-18
3.4.1		2013-10-15

Versión	Notas	Fecha de lanzamiento
3.4.0		2013-09-07
3.3.8		2013-09-02
3.3.7		2013-08-28
3.3.6		2013-08-27
3.3.4		2013-07-08
3.3.3		2013-07-04
3.3.2		2013-07-03
3.3.1		2013-06-27
3.3.0		2013-06-27
3.2.6		2013-06-02
3.2.5		2013-05-21
3.2.4		2013-05-09
3.2.3		2013-05-07
3.2.2		2013-05-03
3.2.1		2013-04-29
3.2.0		2013-04-15
3.1.2		2013-04-12
3.1.1		2013-04-01
3.1.0		2013-01-25
3.0.6		2013-01-04
3.0.5		2012-12-19
3.0.4		2012-12-05
3.0.3		2012-11-13
3.0.2		2012-11-08
3.0.1		2012-11-01

Versión	Notas	Fecha de lanzamiento
3.0.0		2012-10-23
3.0.0rc5		2012-09-18
3.0.0rc4		2012-08-30
3.0.0rc3		2012-08-13
3.0.0rc2		2012-08-03
3.0.0rc1		2012-07-24
3.0.0beta7		2012-07-16
3.0.0beta6		2012-07-13
3.0.0beta5		2012-07-03
3.0.0beta4		2012-06-25
3.0.0beta3		2012-06-15
3.0.0beta2		2012-06-06
3.0.0beta1		2012-06-01
3.0.0alpha5		2012-05-30
3.0.0alpha4		2012-05-09
3.0.0alpha3		2012-05-04
3.0.0alpha2		2012-04-26
3.0.0alpha1		15/04/2012
2.5.9		2012-04-02
2.5.8		2012-02-08
2.5.7		2012-02-06
2.5.6		2012-01-13
2.5.5		2012-01-08
2.5.4		2012-01-02
2.5.3		2011-12-30

Versión	Notas	Fecha de lanzamiento
2.5.2		2011-12-10
2.5.1		2011-11-17
2.5.0		2011-10-24
2.4.7		2011-10-05
2.4.6		2011-08-22
2.4.5		2011-08-19
2.4.4		2011-08-05
2.4.3		2011-07-14
2.4.2		2011-07-06
2.4.1		2011-07-06
2.4.0		2011-06-28
2.3.12		2011-06-22
2.3.11		2011-06-04
2.3.10		2011-05-27
2.3.9		2011-05-25
2.3.8		2011-05-24
2.3.7		2011-05-23
2.3.6		2011-05-20
2.3.5		2011-05-20
2.3.4		2011-05-08
2.3.3		2011-05-03
2.3.2		2011-04-27
2.3.1		2011-04-26
2.3.0		2011-04-25
2.2.2		2011-04-12

Versión	Notas	Fecha de lanzamiento
2.2.1		2011-04-04
2.2.0		2011-03-30
2.1.1		2011-03-29
2.1.0		2011-03-24
2.0.0		2011-03-17
2.0.0rc3		2011-03-17
2.0.0rc2		2011-03-17
2.0.0rc		2011-03-14
2.0.0beta3		2011-03-09
2.0.0beta2		2011-03-07
2.0.0beta		2011-03-03
1.0.8		2011-03-01
1.0.7		2011-02-07
1.0.6		2011-02-07
1.0.5		2011-02-05
1.0.4		2011-02-05
1.0.3		2011-01-13
1.0.2		2011-01-10
1.0.1		2010-12-29
1.0.0		2010-11-16
1.0.0rc4		2010-10-14
1.0.0rc3		2010-09-20
1.0.0rc2		2010-08-17
1.0.0rc		2010-07-28
1.0.0beta2		2010-07-23

Versión	Notas	Fecha de lanzamiento
1.0.0beta		2010-07-15
0.14.0		2010-06-15
0.13.0		2010-06-01
0.12.0		2010-05-22
0.11.0		2010-05-06
0.10.1		2010-05-03
0.10.0		2010-04-30
0.9.0		2010-04-14
0.8.0		2010-03-19
0.7.6		2010-03-19
0.7.5		2010-03-16
0.7.4		2010-03-16
0.7.3		2010-03-16
0.7.2		2010-03-16
0.7.1		2010-03-16
0.7.0		2010-03-15
0.6.0		2010-03-11
0.5.0		2010-03-10
0.4.0		2010-02-11
0.3.0		2010-02-11
0.2.1		2010-02-05
0.2.0		2010-02-03
0.1.0		2010-02-03
0.0.2		2010-01-10
0.0.1	Lanzamiento intial	2010-01-03

Examples

Instalación

Express JS es el marco de goto para desarrollar `Web Applications` , `APIs` y casi cualquier tipo de `Backend` utilizando Node.

Para instalar `express` , todo lo que tienes que hacer es ejecutar el comando **npm**

```
npm install express --save
```

Y tu estas listo.

Para crear y ejecutar un nuevo servidor express.

crea un archivo `app.js` y agrega este código

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

// when "/" is opened in url, this function will be called.
app.get('/', function (req, res) {
  res.json({ code: 200, message: 'success' });
})

app.listen( 3000, function () {
  console.log('Express server running at http://localhost:3000');
});
```

- En su terminal, ejecute `node app.js` y
- Abra la url `http://localhost:3000` en el navegador web para ver su servidor Express recién creado.

También es una buena idea instalar `body-parser` y `express-session` junto con `express` ya que la mayoría del tiempo deseará leer los datos enviados en la solicitud `POST` y administrar las sesiones de usuario.

- body-parser en github
- Sesión Express en Github

Hello World App, usando ExpressJS 4 y Node> = 4

Prefacio

Necesitará `node >= 4` y `express 4` para este proyecto. Puede obtener la última distribución de `node` desde su página de descarga .

Antes de este tutorial, debe inicializar su proyecto de nodo ejecutando

```
$ npm init
```

desde la línea de comandos y rellenando la información que desee. Tenga en cuenta que puede cambiar la información en cualquier momento editando el archivo `package.json` .

Instalación

Instalar `express` con `npm` :

```
$ npm install --save express
```

Después de instalar Express como módulo de nodo, podemos crear nuestro punto de entrada. Esto debería estar en el mismo directorio que nuestro `package.json`

```
$ touch app.js
```

Contenidos del directorio

La carpeta debe tener la siguiente estructura de directorio:

```
<project_root>
 |-> app.js
 |-> node_modules/
 '-> package.json
```

Código

Abra `app.js` en su editor preferido y siga estos cuatro pasos para crear su primera aplicación Express:

```
// 1. Import the express library.
import express from 'express';

// 2. Create an Express instance.
const app = express();

// 3. Map a route. Let's map it to "/", so we can visit "[server]/".
app.get('/', function(req, res) {
   res.send('Hello World');
});

// 4. Listen on port 8080
app.listen(8080, function() {
   console.log('Server is running on port 8080...');
});
```

Ejecución

Desde el directorio del proyecto, podemos ejecutar nuestro servidor usando el comando

```
$ node app.js
```

Deberías ver el texto

```
$ Our Express App Server is listening on 8080...
```

Ahora, visite `http://localhost:8080/` y verá el texto "¡Hola mundo!"

¡Enhorabuena, has creado tu primera aplicación Express!

Iniciar una aplicación con el generador Express.

Para comenzar rápidamente con Express, puede usar el generador Express, que creará un esqueleto de aplicación para usted.

Primero, instálalo globalmente con npm:

```
npm install express-generator -g
```

Es posible que deba poner `sudo` antes de este comando si recibe un error de "permiso denegado".

Una vez que el generador está instalado, puede iniciar un nuevo proyecto como este:

```
express my_app
```

El comando anterior creará una carpeta llamada `my_app` con un archivo `package.json` , un archivo `app.js` y algunas subcarpetas como `bin` , `public` , `routes` , `views` .

Ahora navega a la carpeta e instala las dependencias:

```
cd first_app
npm install
```

Si estás en Linux o macOS, puedes iniciar la aplicación así:

```
DEBUG=myapp:* npm start
```

O, si estás en Windows:

```
set DEBUG=myapp:* & npm start
```

Ahora, cargue http: // localhost: 3000 / en su navegador web y debería ver las palabras "Welcome

to Express".

Creando una aplicación EJS

```
a@coolbox:~/workspace$ express --ejs my-app
a@coolbox:~/workspace$ cd my-app
a@coolbox:~/workspace/my-app$ npm install
a@coolbox:~/workspace/my-app$ npm start
```

Lea Empezando con Express en línea: https://riptutorial.com/es/express/topic/1616/empezando-con-express

Capítulo 2: ¿Cómo funciona ExpressJs?

Examples

Solicitud de manejo / respuesta

El azúcar sintáctico

La mayoría de los ejemplos iniciales de ExpressJs incluyen este fragmento de código.

```
var express = require('express');
var app = express();
...
app.listen(1337);
```

Bueno, `app.listen` es solo un atajo para:

```
var express = require('express');
var app = express();
var http = require('http');
http.createServer(app).listen(1337);
```

La aplicación Express

El famoso `http.createServer` acepta una función que se conoce como el controlador. El manejador toma 2 parámetros de **solicitud** y **respuesta** como entradas, luego los manipula dentro de su alcance para hacer varias cosas.

Básicamente, `app = express()` es una función, que tiene lugar como manejador y se ocupa de la solicitud, respuesta a través de un conjunto de componentes especiales denominados middlewares.

Pila de Middlewares

Un middleware básico es una función que toma 3 argumentos de **solicitud** , **respuesta** y **siguiente** .

Luego, por `app.use` , se monta un middleware en la aplicación Express Middlewares Stack. La solicitud y la respuesta se manipulan en cada middleware y luego se canalizan al siguiente a través de la llamada de `next()` .

Por ejemplo, el siguiente código:

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

```
var app = express();

app.use((request, response, next) => {
    request.propA = "blah blah";
    next();
});

app.use('/special-path', (request, response, next) => {
    request.propB = request.propA + " blah";
    if (request.propB === "blah blah blah")
        next();
    else
        response.end('invalid');
});

app.use((request, response, next) => {
    response.end(request.propB);
});

app.listen(1337);
```

Se puede traducir aproximadamente a:

```
var http = require('http');
http.createServer((request, response) => {

    //Middleware 1
    if (isMatch(request.url, '*')) {
        request.propA = "blah blah";
    }

    //Middleware 2
    if (isMatch(request.url, "/special-path")) {
        request.propB = request.propA + " blah";
        if (request.propB !== "blah blah blah")
            return response.end('invalid');
    }

    //Middleware 3
    if (isMatch(request.url, "*")) {
        return response.end(request.propB);
    }
});

server.listen(1337);
```

Lea ¿Cómo funciona ExpressJs? en línea: https://riptutorial.com/es/express/topic/7815/-como-funciona-expressjs-

Capítulo 3: Conectar

Examples

Conectar y Expresar

Express se basa en Connect, que es lo que proporciona la funcionalidad de middleware de Express. Para entender qué es la conexión, puede ver que proporciona la estructura básica de la aplicación que usa cuando usa Express

```
const connect = require('connect')

const app = connect()
app.listen(3000)
```

Esto abrirá un servidor http "vacío" que responderá 404 a todas las solicitudes.

Middleware

El middleware se adjunta al objeto de la aplicación, generalmente antes de que se llame a la escucha. Ejemplo de un middleware de registro simple:

```
app.use(function (req, res, next) {
    console.log(`${req.method}: ${req.url}`)
    next()
})
```

Todo lo que hará es registrar `GET: /example` si `localhost:3000/example` GET `localhost:3000/example` . Todas las solicitudes seguirán devolviendo 404 ya que no está respondiendo con ningún dato.

El próximo middleware de la cadena se ejecutará tan pronto como el anterior llame a `next()` , para que podamos seguir adelante y responder a las solicitudes agregando otro middleware como este:

```
app.use(function (req, res, next) {
    res.end(`You requested ${req.url}`)
})
```

Ahora, cuando solicite ´localhost: 3000 / example `you will be greeted with "You requested /example". There is no need to call` next` esta vez ya que este middleware es el último de la cadena (pero no pasará nada malo si lo hicieras),

Programa completo hasta aquí:

```
const connect = require('connect')

const app = connect()
```

```
app.use(function (req, res, next) {
    console.log(`${req.method}: ${req.url}`)
    next()
})

app.use(function (req, res, next) {
    res.end(`You requested ${req.url}`)
    next()
})

app.listen(3000)
```

Errores y middleware de errores.

Si quisiéramos limitar el acceso a nuestra aplicación, ¡podríamos escribir un middleware para eso también! Este ejemplo solo le otorga acceso los días de la semana, pero un ejemplo del mundo real podría ser, por ejemplo, *la autenticación del usuario* . Un buen lugar para colocar esto sería después del middleware de registro, pero antes de enviar cualquier contenido.

```
app.use(function (req, res, next) {
    if (new Date().getDay() !== 4) {
        next('Access is only granted on thursdays')
    } else {
        next()
    }
})
```

Como puede ver en este ejemplo, enviar un error es tan fácil como proporcionar un prameter a la función `next()` .

Ahora, si visitamos el sitio web un día diferente al jueves, recibiremos un error 500 y la cadena `'Access is only granted on thursdays'` .

Ahora, esto no es lo suficientemente bueno para nuestro sitio. Preferimos enviar al usuario un mensaje HTML en otro middleware:

```
app.use(function (err, req, res, next) {
    res.end(`<h1>Error</h1><p>${err}</p>`)
})
```

Esto funciona como un bloque catch: cualquier error en el middleware anterior al middleware de error se enviará al primero. Un middleware de error se identifica por sus 4 parámetros.

También puede usar el middleware de error para recuperarse del error llamando nuevamente al siguiente método:

```
app.use(function (err, req, res, next) {
    // Just joking, everybody is allowed access to the website!
    next()
})
```

Lea Conectar en línea: https://riptutorial.com/es/express/topic/4031/conectar

Capítulo 4: Enrutamiento

Examples

Enrutamiento hola mundo

El archivo de aplicación principal carga el archivo de rutas donde se definen las rutas.

app.js

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

app.use('/', require('./routes'));

app.listen('3000');
```

rutas.js

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

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

module.exports = router;
```

Middleware de enrutamiento

El software intermedio se ejecuta antes de la ejecución de la ruta y puede decidir si ejecutar el enrutador de acuerdo con la URL.

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

router.use(function (req, res, next) {
    var weekDay = new Date().getDay();
    if (weekDay === 0) {
        res.send('Web is closed on Sundays!');
    } else {
        next();
    }
})

router.get('/', function(req, res) {
    res.send('Sunday is closed!');
});

module.exports = router;
```

También se puede enviar un middleware específico a cada controlador de enrutador.

```
var closedOnSundays = function (req, res, next) {
    var weekDay = new Date().getDay();
    if (weekDay === 0) {
        res.send('Web is closed on Sundays!');
    } else {
        next();
    }
}

router.get('/', closedOnSundays, function(req, res) {
    res.send('Web is open');
});

router.get('/open', function(req, res) {
    res.send('Open all days of the week!');
});
```

Múltiples rutas

El archivo de aplicación principal carga cualquier archivo de rutas en el que le gustaría definir rutas. Para ello necesitamos la siguiente estructura de directorios: app.js route / index.js route / users.js

app.js

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

app.use('/', require('./routes/index'));
app.use('/users', require('./routes/users'))

app.listen('3000');
```

rutas / index.js

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

router.get('/', function(req, res) {
    res.send('Index Page');
});

router.get('/about', function(req, res) {
    res.send('About Page');
});

module.exports = router;
```

rutas / usuarios.js

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

router.get('/', function(req, res) {
    res.send('Users Index Page');
});
```

```
router.get('/list', function(req, res) {
    res.send('Users List Page');
});

module.exports = router;
```

Ejecutando `$ node app.js` ahora debería haber páginas en las siguientes direcciones URL:

- localhost: 3000 / - Muestra "Página de índice"
- localhost: 3000 / about - Muestra "Acerca de la página"
- localhost: 3000 / usuarios - Muestra "Página de índice de usuarios"
- localhost: 3000 / users / list - Muestra "Página de lista de usuarios"

Lea Enrutamiento en línea: https://riptutorial.com/es/express/topic/2589/enrutamiento

Capítulo 5: Escribiendo Middleware Express

Sintaxis

1. Especifique la instancia de expreso que desea utilizar. Esto es comúnmente la *aplicación* .
2. Defina el método HTTP para el que se aplica la función. En el ejemplo, esto es *obtener* .
3. Defina la ruta a la que se aplica la función. En el ejemplo, esto es *'/'* .
4. Definir como una función con la palabra clave de *función* .
5. Agregue los parámetros requeridos: req, res, next. (Ver nota en la sección de comentarios)
6. Pon un código en la función para hacer lo que quieras.

Parámetros

Parámetro	Detalles
req	El objeto de solicitud.
res	El objeto de respuesta.
siguiente	La siguiente () llamada de middleware.

Observaciones

Una función de middleware es una función con acceso al objeto de solicitud (*req*), al objeto de respuesta (*res*) y a la función de middleware *siguiente ()* en el ciclo de solicitud-respuesta de la aplicación. La función middleware *next ()* es comúnmente denotada por una variable llamada next.

Las funciones de middleware están diseñadas para realizar las siguientes tareas:

- Ejecutar cualquier código.
- Realizar cambios en los objetos de solicitud y respuesta. (Ver el ejemplo de requestTime)
- Finaliza el ciclo de solicitud-respuesta.
- Llame al siguiente middleware en la pila. (Llamando al *siguiente ()* middleware)

Nota: no tiene que ser nombrado a continuación. Pero si usas algo más, nadie sabrá a qué te refieres y serás despedido. Y tu código no funcionará. Por lo tanto, sólo el nombre a continuación. Esta regla se aplica al objeto de solicitud y respuesta. Algunas personas usarán solicitud y respuesta en lugar de req y res, respectivamente. Esta bien. Se desperdicia pulsaciones, pero está bien.

Examples

Logger Middleware

Si no está familiarizado con el middleware en Express, consulte la Información general en la sección Comentarios.

Primero, vamos a configurar una aplicación simple Hello World a la que se hará referencia y se agregará durante los ejemplos.

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

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

app.listen(3000);
```

Aquí hay una función de middleware simple que registrará "LOGGED" cuando se llame.

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

*Llamar a **next ()** invoca la siguiente función de middleware en la aplicación.*

Para cargar la función, llame a app.use () y especifique la función a la que desea llamar. Esto se hace en el siguiente bloque de código que es una extensión del bloque Hello World.

```
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);
```

Ahora, cada vez que la aplicación recibe una solicitud, imprime el mensaje "LOGGED" en el terminal. Entonces, ¿cómo agregamos condiciones más específicas cuando se llama middleware? Mira el siguiente ejemplo y mira.

requestTime Middleware

Vamos a crear middleware que agregue una propiedad llamada requestTime al objeto de solicitud.

```
var requestTime = function (req, res, next) {
```

```
    req.requestTime = Date.now();
    next();
};
```

Ahora modifiquemos la función de registro del ejemplo anterior para utilizar el middleware requestTime.

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

Agreguemos el middleware a nuestra aplicación:

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

myLogger = function (req, res, next) {
    console.log('LOGGED at ' + req.requestTime);
    next();
};

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

app.use(requestTime);

app.use(myLogger);

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

app.listen(3000);
```

Ahora la aplicación registrará el momento en que se realizó la solicitud. Esto cubre los conceptos básicos de escritura y uso de middleware Express. Para obtener más información, consulte Uso de Middleware Express .

TODO: Crear usando la sección Express Middleware !!!

CORS Middleware

Este ejemplo demuestra cómo se puede manejar una solicitud http de origen cruzado utilizando un middleware.

Fondo CORS

CORS es un método de control de acceso adoptado por todos los navegadores principales para evitar las vulnerabilidades de secuencias de comandos cruzadas inherentes a ellos. En general, la seguridad del navegador, los scripts deben mantener que todas las solicitudes de XHR deben

realizarse solo en la fuente de la que se sirven los mismos scripts. Si se realiza una solicitud XHR fuera del dominio al que pertenecen los scripts, se rechazará la respuesta.

Sin embargo, si el navegador admite CORS, haría una excepción a esta regla si los encabezados apropiados en la respuesta indican que se permite el dominio desde el cual se origina la solicitud. El siguiente encabezado indica que cualquier dominio está permitido:

```
Access-Control-Allow-Origin: *
```

Ejemplo

El siguiente ejemplo muestra cómo el middleware Express puede incluir estos encabezados en su respuesta.

```
app.use(function(request, response, next){

    response.header('Access-Control-Allow-Origin', '*');
    response.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE,OPTIONS');
    response.header('Access-Control-Allow-Headers', 'Content-Type, Authorization, Content-
Length, X-Requested-With');

   //Handle Preflight
   if (reqest.method === 'OPTIONS') {
      response.status(200).send();
   }
   else {
      next();
   }

});
```

Manejo de Preflight

La última parte del ejemplo anterior maneja Preflight. Preflight es una OPCIÓN especial que el navegador envía para probar CORS si la solicitud contiene encabezados personalizados.

Referencias útiles

MDN - Tutorial CORS Http

Lea Escribiendo Middleware Express en línea:
https://riptutorial.com/es/express/topic/6993/escribiendo-middleware-express

Capítulo 6: Explicar enrutamiento en expreso

Examples

Express Router

Express Router le permite crear múltiples "mini aplicaciones" para que pueda asignar un espacio de nombre a su api, público, autenticación y otras rutas en sistemas de enrutamiento separados.

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

router.get('/', function(req, res){
   res.send('Get request received');
});

router.post('/', function(req, res){
   res.send('Post requestreceived');
});

app.use('/', router);

app.listen(8080);
```

Controladores de ruta de Chainable para una ruta de acceso mediante el uso de app.ruta

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

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

Lea Explicar enrutamiento en expreso en línea:
https://riptutorial.com/es/express/topic/6536/explicar-enrutamiento-en-expreso

Capítulo 7: Explotación florestal

Observaciones

`morgan` es un middleware de registro de solicitudes HTTP para node.js

Examples

Instalación

Primero, instale el middleware `morgan` en su proyecto.

```
npm install --save morgan
```

Registro simple exprés de toda solicitud a STDOUT

Agregue el siguiente código a su archivo `app.js` :

```
var express = require('express')
var morgan = require('morgan')

var app = express()

app.use(morgan('combined'))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Ahora, cuando acceda a su sitio web, verá en la consola que utilizó para iniciar el servidor que se registran las solicitudes.

Escribir registros expresos en un solo archivo

Primero, instale `fs` y `path` en su proyecto

```
npm install --save fs path
```

Agregue el siguiente código a su archivo `app.js` :

```
var express = require('express')
var fs = require('fs')
var morgan = require('morgan')
var path = require('path')

var app = express()

// create a write stream (in append mode)
```

```
var accessLogStream = fs.createWriteStream(path.join(__dirname, 'access.log'), {flags: 'a'})

// setup the logger
app.use(morgan('combined', {stream: accessLogStream}))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Ahora, cuando acceda a su sitio web, verá que se creó un archivo `access.log` en el directorio de su proyecto

Escribir registros Express en un archivo de registro rotativo

Primero, instale `fs` , `file-stream-rotator` y `path` en su proyecto

```
npm install --save fs file-stream-rotator path
```

Agregue el siguiente código a su archivo `app.js` :

```
var FileStreamRotator = require('file-stream-rotator')
var express = require('express')
var fs = require('fs')
var morgan = require('morgan')
var path = require('path')

var app = express()
var logDirectory = path.join(__dirname, 'log')

// ensure log directory exists
fs.existsSync(logDirectory) || fs.mkdirSync(logDirectory)

// create a rotating write stream
var accessLogStream = FileStreamRotator.getStream({
  date_format: 'YYYYMMDD',
  filename: path.join(logDirectory, 'access-%DATE%.log'),
  frequency: 'daily',
  verbose: false
})

// setup the logger
app.use(morgan('combined', {stream: accessLogStream}))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Ahora, cuando acceda a su sitio web, verá que se creó un directorio de `log` se creó un archivo de registro con un formato de nombre de `access-%DATE%.log` en su directorio de registro

Lea Explotación florestal en línea: https://riptutorial.com/es/express/topic/7191/explotacion-florestal

Capítulo 8: expreso-generador

Parámetros

Parámetro	Definición
-h, --ayudar	información de uso de salida
-V, --version	mostrar el número de versión
-e, --ejs	agregar pjs (JavaScript incorporado) compatibilidad con el motor de plantillas (por defecto a jade, que ha sido renombrado a Pug)
--hbs	añadir manillares modelando el soporte del motor
-H, --hogan	añadir soporte del motor hogan.js
--git	añadir .gitignore
-f, --fuerza	fuerza en el directorio no vacío
-c <motor>, --css <motor>	agregar soporte de hojas de estilo <engine> (less, stylus, compass, sass) (el valor predeterminado es css)

Observaciones

Express generator es una gran herramienta para poner en marcha un proyecto rápidamente. Una vez que entiendes la organización que implementa, es un ahorro de tiempo real.

Examples

Instalación de Express Generator

```
npm --install express-generator -g
```

Crear una aplicación

```
express my-app
```

Iniciar aplicación

Usando la opción de inicio

```
npm start
```

Usando Nodemon

```
nodemon
```

Usando para siempre

```
forever start 'js file name'
```

Parar para siempre

```
forever stop ''js file name'
```

Para reiniciar en siempre

```
forever restart 'js filename'
```

Listar el servidor ruuning utilizando para siempre

```
forever list
```

Lea expreso-generador en línea: https://riptutorial.com/es/express/topic/4512/expreso-generador

Capítulo 9: Integración expresa de la base de datos

Examples

Conéctate a MongoDB con Node y Express

En primer lugar, asegúrese de que ha instalado *mongodb* y *expresar* a través de la NGP. Luego, en un archivo titulado convencionalmente *db.js* , use el siguiente código:

```
var MongoClient = require('mongodb').MongoClient

var state = {
    db: null,
}

exports.connect = function(url, done) {
    if (state.db) return done()

    MongoClient.connect(url, function(err, db) {
        if(err) return done(err)
        state.db = db
        done()
    })
}

exports.get = function() {
    return state.db
}

exports.close = function(done) {
    if (state.db) {
        state.db.close(function(err, result) {
            state.db = null;
            state.mode = null;
            done(err);
        })
    }
}
```

Este archivo se conectará a la base de datos y luego puede usar el objeto **db** devuelto por el método **get** .

Ahora necesita incluir el archivo db exigiéndolo en su archivo app.js. Suponiendo que su archivo *db.js* esté en el mismo directorio que *app.js* , puede insertar la línea:

```
var db = require('./db');
```

Sin embargo, esto no lo conecta realmente a su instancia de MongoDB. Para hacerlo, inserte el siguiente código antes de llamar al método app.listen. En nuestro ejemplo, integramos el manejo de errores y el método app.listen en la conexión de la base de datos. Tenga en cuenta que este

código solo funciona si está ejecutando su instancia de mongo en la misma máquina en la que se encuentra la aplicación Express.

```
db.connect('mongodb://localhost:27017/databasename', function(err) {
    if (err) {
        console.log('Unable to connect to Mongo.');
        process.exit(1);
    } else {
        app.listen(3000, function() {
            console.log('Listening on port 3000...');
        });
    }
});
```

Ahí lo tienes, tu aplicación Express ahora debería estar conectada a tu base de datos Mongo. Felicidades

Lea Integración expresa de la base de datos en línea:
https://riptutorial.com/es/express/topic/7002/integracion-expresa-de-la-base-de-datos

Capítulo 10: Manejo de archivos estáticos

Sintaxis

1. Para servir archivos estáticos (Imágenes, CSS, archivos JS, etc.) use la función **express.static** middleware.
2. Pase el nombre del directorio que contiene los recursos a **express.static** para servir los archivos directamente. (Mira el *ejemplo básico*)
3. Puede usar varios directorios, simplemente llame a **express.static** varias veces. Recuerde, Express busca archivos en el orden en que establece los directorios con **express.static** . (Mira el *ejemplo de directorios múltiples*)
4. Puede crear un prefijo de ruta virtual (es decir, uno donde la ruta no existe realmente en el sistema de archivos) con **express.static** , solo especifique una ruta de montaje. (Mira el *ejemplo del prefijo de ruta virtual*)
5. Todas las rutas anteriores han sido relativas al directorio desde donde ejecutó el proceso del *nodo* . Por lo tanto, generalmente es más seguro utilizar la ruta absoluta del directorio que desea servir. (Mire la *ruta absoluta al ejemplo del directorio de archivos estáticos*)
6. Puede combinar y combinar las opciones de este método, como se ve en el *Ejemplo de Prefijo de Ruta Absoluta a Directorio y Ruta Virtual*

Observaciones

Todos los ejemplos se pueden ejecutar en el nodo. Simplemente copie y pegue en un proyecto de nodo con Express instalado y ejecútelos con el **nombre de archivo de nodo** . Para ver un ejemplo de cómo instalar Express, haga clic aquí y asegúrese de que tiene npm instalado, luego siga las instrucciones sobre la instalación de paquetes para instalar "express".

Examples

Ejemplo básico

```
// Basic code for Express Instance
var express = require('express');
var app = express();

// Serve static files from directory 'public'
app.use(express.static('public'));

// Start Express server
app.listen(3030);
```

Ejemplo de directorios múltiples

```
// Set up Express
var express = require('express');
```

```
var app = express();

// Serve static assets from both 'public' and 'files' directory
app.use(express.static('public');
app.use(express.static('files');

// Start Express server
app.listen(3030);
```

Ejemplo de prefijo de ruta virtual

```
// Set up Express
var express = require('express');
var app = express();

// Specify mount path, '/static', for the static directory
app.use('/static', express.static('public'));

// Start Express server
app.listen(3030);
```

Ejemplo de directorio de ruta absoluta a archivos estáticos

```
// Set up Express
var express = require('express');
var app = express();

// Serve files from the absolute path of the directory
app.use(express.static(__dirname + '/public'));

// Start Express server
app.listen(3030);
```

Ruta absoluta al directorio y ejemplo de prefijo de ruta virtual

```
// Set up Express
var express = require('express');
var app = express();

/*  Serve from the absolute path of the directory that you want to serve with a
 */ virtual path prefix
app.use('/static', express.static(__dirname + '/public'));

// Start Express server
app.listen(3030);
```

Ejemplo de archivos estáticos básicos y favicon.

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');

var app = express();
```

```
app.use(favicon(__dirname + '/public/img/favicon.ico'));
app.use(express.static(path.join(__dirname, 'public')));

app.listen(3000, function() {
  console.log("Express App listening on port 3000");
})
```

Lea Manejo de archivos estáticos en línea: https://riptutorial.com/es/express/topic/6954/manejo-de-archivos-estaticos

Capítulo 11: Manejo de errores

Sintaxis

- `app.use(function(err, req, res, next) {})` // middleware básico

Parámetros

Nombre	Descripción
`err`	Objeto con información de error
`req`	Objeto de solicitud HTTP
`res`	Objeto de respuesta HTTP
`next`	Función utilizada para iniciar la próxima ejecución de middleware.

Examples

Muestra basica

A diferencia de otras funciones de middleware, las funciones de manejo de errores de middleware tienen cuatro argumentos en lugar de tres: `(err, req, res, next)` .

Muestra:

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

Lea Manejo de errores en línea: https://riptutorial.com/es/express/topic/2739/manejo-de-errores

Capítulo 12: utilizando https con express

Examples

Usando https con express

Primero tienes que generar claves públicas y privadas usando OpenSSL (tutorial).

```
var express = require("express");
var http =require ("http");
var https=require ("https");
var fs=require("fs");
var app=express();
var httpsKeys={
key:fs.readFileSync("<key.pem>");
crtifcte:fs.readFileSync("<certificate.pem>");
};
http.createserver(app).listen(3000);
https.createserver(httpsKeys,app).listen(3030);
```

Lea utilizando https con express en línea: https://riptutorial.com/es/express/topic/7844/utilizando-https-con-express

Capítulo 13: Ver la configuración del motor

Introducción

A menudo, el servidor necesita servir páginas dinámicamente. Por ejemplo, un usuario Mr.X visita la página y ve algo como "Bienvenido Mr. X a mi página de inicio". En este caso, las vistas pueden ser útiles. Incluso para rellenar una vista de tabla puede ser práctico Las variables se pueden inyectar en HTML dinámicamente usando el motor de visualización. El motor de visualización es algo que representa las vistas. Uno puede mantener las vistas para que se sirvan en una carpeta llamada vista y se sirva a petición. La ruta de la carpeta se puede mostrar a Express usando la ruta. método de resolución

Observaciones

instale ejs usando lo siguiente (sé que es obvio)

```
sudo npm install ejs --save
```

Examples

1: configurando las vistas

```
var express=require("express");    //express is included
var path=require("path");    //path is included

var app=express();    //app is an Express type of application

app.set("views",path.resolve(__dirname,"views"));    //tells express about the location of the
views in views folder
app.set("view engine","ejs");    //tells express that ejs template engine is used
```

Ejemplo de archivo 2.EJS (consulte 1. configuración ... antes de esto)

el siguiente es un archivo ejs.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Hello, world!</title>
    </head>
    <body>
        <%= message %>
    </body>
</html>
```

Vista de 3.rendering con expreso (consulte el archivo 2.EJS ... antes de esto)

```
app.get("/",function(req,res){
response.render("index",{                  //render the index when root(/) is requested
    message:"rendered view with ejs"
        });
});
```

4. Después de crear el HTML final se crea (consulte 3.rendering ... antes de esto)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello, world!</title>
</head>
<body>
    message:"rendered view with ejs"
</body>
</html>
```

Lea Ver la configuración del motor en línea: https://riptutorial.com/es/express/topic/8104/ver-la-configuracion-del-motor

Creditos

S. No	Capítulos	Contributors
1	Empezando con Express	Akshay Khale, Community, David Vogel, Dima Grossman, dkimot, Everettss, Gregory Worrall, Guillaume Lrv, Jared Hooper, jawadhoot, Kilmazing, Random User, Sumner Evans, user6939352
2	¿Cómo funciona ExpressJs?	rocketspacer
3	Conectar	Henrik Karlsson, Overflowh
4	Enrutamiento	jawadhoot, Kelvin, phobos, S.L. Barth, zurfyx
5	Escribiendo Middleware Express	Charlie H, dkimot
6	Explicar enrutamiento en expreso	Dima Grossman, Sujithrao
7	Explotación florestal	Mor Paz
8	expreso-generador	rickrizzo, Rupali Pemare
9	Integración expresa de la base de datos	dkimot
10	Manejo de archivos estáticos	dkimot, Sujithrao
11	Manejo de errores	gevorg, jawadhoot, Kilmazing
12	utilizando https con express	nilakantha singh deo
13	Ver la configuración del motor	Daniele Giussani, nilakantha singh deo