Taller NodeJS – Express: Módulo Router

Instructor: Abner Saavedra
Fecha: sept. de 2024
Email: ingenieroabnersaavedra@gmail.com
GracoSoft Centro Empresarial Plaza Madrid, piso 9, oficinas 9-7 a la 9-10

Agenda

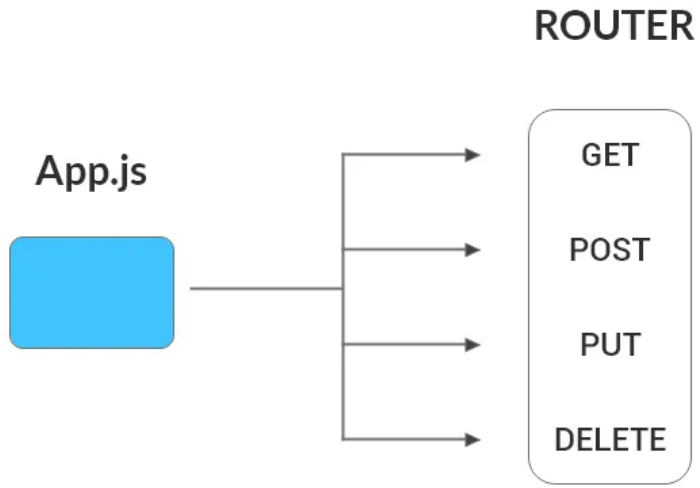
- Introducción a las rutas Express
- Definición y uso de módulos de ruta independientes
- Funciones de ruta
- Verbos HTTP
- Rutas de acceso
- Parámetros de ruta
- Manejo de errores en las funciones de ruta
- Control de excepciones en funciones de ruta

Introducción a las rutas Express

Una ruta es una sección de código Express que asocia un verbo HTTP (GET, POST, PUT, DELETE, etc.), una ruta/patrón de URL y una función que se llama para manejar ese patrón.

Existen varias formas de crear rutas. En esta oportunidad, vamos a utilizar el express.Router middleware, ya que nos permite agrupar los controladores de ruta para una parte particular de un sitio y acceder a ellos mediante un prefijo de ruta común. Mantendremos todas nuestras rutas relacionadas con la biblioteca en un módulo de "catálogo" y, si agregamos rutas para gestionar cuentas de usuario u otras funciones, podemos mantenerlas agrupadas por separado.

El uso de Router es muy similar a definir rutas directamente en el objeto de la aplicación Express .



Definición y uso de módulos de ruta independientes

El código siguiente proporciona un ejemplo concreto de cómo podemos crear un módulo de ruta y luego usarlo en una aplicación *Express* .

Primero creamos rutas para una wiki en un módulo llamado **wiki.js.** El código primero importa el objeto de aplicación Express, lo usa para obtener un Router objeto y luego le agrega un par de rutas usando el get() método. Por último, el módulo exporta el Router objeto.

```
JS
// wiki.js - Wiki route module.

const express = require("express");
const router = express.Router();

// Home page route.
router.get("/", function (req, res) {
  res.send("Wiki home page");
});

// About page route.
router.get("/about", function (req, res) {
  res.send("About this wiki");
});

module.exports = router;
```

Nota: Arriba, estamos definiendo nuestras devoluciones de llamada del controlador de ruta directamente en las funciones del enrutador.

Para usar el módulo de enrutador en nuestro archivo de aplicación principal, primero usamos el módulo de ruta (wiki.js). A continuación, llamamos a la aplicación Express para que agregue el enrutador a la ruta de manejo de middleware, especificando una ruta de URL de 'wiki'. require() use()

```
JS
const wiki = require("./wiki.js");
// ...
app.use("/wiki", wiki);
```

Las dos rutas definidas en nuestro módulo de ruta wiki son entonces accesibles desde /wiki/ y /wiki/about/

Funciones de ruta

Nuestro módulo anterior define un par de funciones de ruta típicas. La ruta "about" (reproducida a continuación) se define utilizando el método, que responde solo a las solicitudes HTTP GET. El primer argumento de este método es la ruta de acceso de la URL, mientras que el segundo es una función de devolución de llamada que se invocará si se recibe una solicitud HTTP GET con la ruta.Router.get()

```
JS
router.get("/about", function (req, res) {
  res.send("About this wiki");
});
```

La devolución de llamada toma tres argumentos (generalmente denominados como se muestra: , ,), que contendrán el objeto de solicitud HTTP, la respuesta HTTP y la siguiente función en la cadena de middleware.req res next

Nota: Las funciones del enrutador son middleware Express, lo que significa que deben completar (responder a) la solicitud o llamar a la función en la cadena. En el caso anterior, completamos la solicitud usando res, por lo que el argumento no se usa (y elegimos no especificarlo). next send() next.
La función de enrutador anterior toma una sola devolución de llamada, pero puede especificar tantos argumentos de devolución de llamada como desee, o una matriz de funciones de devolución de llamada. Cada función es parte de la cadena de middleware y se llamará en el orden en que se agregue a la cadena (a menos que una función anterior complete la solicitud).

La función de devolución de llamada aquí llama a send() en la respuesta para devolver la cadena "Acerca de este wiki" cuando recibimos una solicitud GET con la ruta (). Hay una serie de otros métodos de respuesta para finalizar el ciclo de solicitud/respuesta. Por ejemplo, puedes llamar a res.json() para enviar una respuesta JSON o a res.sendFile() para enviar un archivo. El método de respuesta que usaremos con más frecuencia a medida que construimos la biblioteca es render(), que crea y devuelve archivos HTML usando plantillas y datos, ¡hablaremos mucho más sobre eso en un artículo posterior! /about

Verbos HTTP

Las rutas de ejemplo anteriores utilizan el método para responder a las solicitudes HTTP GET con una ruta determinada. Router.get()

También proporciona métodos de ruta para todos los demás verbos HTTP, que en su mayoría se usan exactamente de la misma manera: , , y

Router post() put() delete() options() trace() copy() lock() mkcol() move() purge() propfind() proppatch() unlock() report() mkactivity() checkout() merge() m-search() notify() subscribe() unsubscribe() patch() search() connect()

Por ejemplo, el código siguiente se comporta igual que la ruta anterior, pero solo responde a las solicitudes HTTP POST. /about

```
JS
router.post("/about", (req, res) => {
  res.send("About this wiki");
});
```

Rutas de acceso

Las rutas de ruta definen los puntos de conexión en los que se pueden realizar las solicitudes. Los ejemplos que hemos visto hasta ahora solo han sido cadenas, y se usan exactamente como están escritos: '/', '/about', '/book', '/any-random.path'.

Las rutas de ruta también pueden ser patrones de cadena. Los patrones de cadena usan una forma de sintaxis de expresión regular para definir patrones de puntos de conexión que coincidirán. La sintaxis se enumera a continuación (tenga en cuenta que el guión () y el punto () se interpretan literalmente mediante rutas basadas en cadenas): - .

- ? : El punto final debe tener 0 o 1 del carácter (o grupo) anterior, por ejemplo, una ruta de ruta coincidirá con los puntos finales o .'/ab?cd' acd abcd
- + : El punto final debe tener 1 o más del carácter (o grupo) anterior, por ejemplo, una ruta de ruta coincidirá con los puntos finales , , , etc.'/ab+cd' abcd abbcd abbbcd
- : El punto final puede tener una cadena arbitraria donde se coloca el carácter. Por ejemplo, una ruta de ruta coincidirá con los puntos finales , , , y así sucesivamente.*'/ab*cd' abcd abXcd abSOMErandomTEXTcd
- () : Coincidencia de agrupación en un conjunto de caracteres para realizar otra operación, p. ej. realizará un -match en el grupo — coincidirá y .'/ab(cd)?e' ? (cd) abeabcde

  Las rutas de ruta también pueden ser expresiones regulares de JavaScript. Por ejemplo, la ruta que se muestra a continuación coincidirá con y , pero no con , , y así sucesivamente. Tenga en cuenta que la ruta de acceso de una expresión regular utiliza la sintaxis de expresión regular (no es una cadena entrecomillada como en los casos anteriores). catfish dogfish catflap catfishhead

```JS
app.get(/.*fish$/, function (req, res) {
  // ...
});
```

Parámetros de ruta

Los parámetros de ruta son segmentos de URL con nombre que se utilizan para capturar valores en posiciones específicas de la URL. Los segmentos con nombre tienen el prefijo dos puntos y luego el nombre (por ejemplo, /:your_parameter_name). Los valores capturados se almacenan en el objeto utilizando los nombres de los parámetros como claves (por ejemplo, /req.params req.params.your_parameter_name).

Así, por ejemplo, considere una URL codificada para contener información sobre usuarios y libros: . Podemos extraer esta información como se muestra a continuación, con los parámetros y path: http://localhost:3000/users/34/books/8989 userId bookId

```
Js
app.get("/users/:userId/books/:bookId", (req, res) => {
  // Access userId via: req.params.userId
  // Access bookId via: req.params.bookId
  res.send(req.params);
});
```

Los nombres de los parámetros de ruta deben estar formados por "caracteres de palabra" (A-Z, a-z, 0-9 y _).

Manejo de errores en las funciones de ruta

Todas las funciones de ruta mostradas anteriormente tienen argumentos y, que representan la solicitud y la respuesta, respectivamente. Las funciones de ruta también se llaman con un tercer argumento, que se puede usar para pasar errores a la cadena de middleware Express. req res next

El código siguiente muestra cómo funciona esto, utilizando el ejemplo de una consulta de base de datos que toma una función de devolución de llamada y devuelve un error o algunos resultados. Si se devuelve error, se llama como valor en su primer parámetro (finalmente, el error se propaga a nuestro código de manejo de errores global). Si se realiza correctamente, se devuelven los datos deseados y, a continuación, se utilizan en la respuesta. err err next err

```
Js
router.get("/about", (req, res, next) => {
  About.find({}).exec((err, queryResults) => {
    if (err) {
      return next(err);
    }
    //Successful, so render
    res.render("about_view", { title: "About", list: queryResults });
  });
});
```

Control de excepciones en funciones de ruta

En la sección anterior se muestra cómo Express espera que las funciones de ruta devuelvan errores. El marco está diseñado para su uso con funciones asincrónicas que toman una función de devolución de llamada (con un argumento de error y resultado), a la que se llama cuando se completa la operación. Eso es un problema si por ejemplo realizamos consultas a la base de datos de Mongoose que usan API basadas en promesas, y que pueden generar excepciones en nuestras funciones de ruta (en lugar de devolver errores en una devolución de llamada).

Para que el marco controle correctamente las excepciones, se deben detectar y, a continuación, reenviar como errores, como se muestra en la sección anterior.

Nota: Se espera que Express 5, que actualmente se encuentra en versión beta, maneje las excepciones de JavaScript de forma nativa.

Reimaginando el ejemplo simple de la sección anterior con una consulta de base de datos que devuelve una promesa, podríamos escribir la función de ruta dentro de un try... catch como este: About.find().exec()

```Js
exports.get("/about", async function (req, res, next) {
  try {
    const successfulResult = await About.find({}).exec();
    res.render("about_view", { title: "About", list: successfulResult });
  } catch (error) {
    return next(error);
  }
});
```

Eso es una gran cantidad de código repetitivo para agregar a cada función. En su lugar, en esta oportunidad usaremos el módulo express-async-handler. Esto define una función entonadora que oculta el bloque y el código para reenviar el error. El mismo ejemplo ahora es muy simple, porque solo necesitamos escribir código para el caso en el que asumamos el éxito: try...catch

```
JS
// Import the module
const asyncHandler = require("express-async-handler");

exports.get(
  "/about",
  asyncHandler(async (req, res, next) => {
    const successfulResult = await About.find({}).exec();
    res.render("about_view", { title: "About", list: successfulResult });
  }),
);
```

Ejercicio

Construir una API utilizando Node.js y Express para gestionar productos. La API debe permitir realizar las operaciones básicas de creación, lectura, actualización y eliminación (CRUD) de productos

Requisitos:

1. **Creación de Producto:**
   - Implementa una ruta para agregar un nuevo producto a la base de datos.
   - El cuerpo de la solicitud (`body`) debe contener la información necesaria del producto, como nombre, precio y cantidad.
2. **Consulta de Producto:**
   - Implementa una ruta para consultar la información de un producto específico por su identificador único.
   - La información del producto debe incluir al menos el nombre, precio y cantidad.
3. **Modificación de Producto:**
   - Crea una ruta para actualizar la información de un producto existente.
   - El cuerpo de la solicitud debe contener los campos que se desean actualizar.
4. **Eliminación de Producto:**
   - Implementa una ruta para eliminar un producto según su identificador único.
5. **Filtrado de Información:**
   - Proporciona la capacidad de filtrar la información de los productos según ciertos criterios (por ejemplo, por precio o cantidad).
   - Los criterios de filtrado deben ser parámetros de la URL.

Notas Adicionales:

- Utiliza Express para crear el servidor web.
- Utiliza el módulo Express Router para crear las rutas de acceso.
- Puedes almacenar la información de los productos en una base de datos simple, como un archivo JSON o en memoria.

GRACOSOFT ES EXCELENCIA EDUCATIVA