Módulo fs (práctica)

Tiempo estimado: 15min

El objeto de esta práctica es afianzar, reforzar y consolidar los conocimientos teóricos presentados en la lección.

Al finalizarla, el estudiante:

- Habrá desarrollado una aplicación que utiliza el módulo fs.

Objetivos

El objetivo de la práctica es crear una aplicación que liste las entradas del directorio actual. Algo como lo siguiente:

```
$ myll
-rwxrwxr-x .editorconfig
-rwxrw-r-- .eslintignore
-rwxrw-r-- .eslintrc
-rwxrwxr-x .gitignore
-rwxrwxr-x .travis.yml
drwxrwxr-x bin
drwxrwxr-x build
drwxrwxr-x dist
-rwxrwxr-x index.js
-rwxrw-r-- Justo.js
drwxrwxr-x lib
drwxrwxr-x node_modules
-rwxrw-r-- package.json
-rwxrw-r-- README.md
drwxrwxr-x test
$
```

El estudiante podría pensar que como hacemos tanto hincapié en el modelo asíncrono, todo debe hacerse así. Pues no. Esta práctica tiene como objeto mostrar que a veces Node debe utilizarse en modo síncrono. No tiene por qué hacerlo siempre en modo asíncrono.

¿Y por qué esta práctica se recomienda hacerla síncronamente? Sencillo. Porque el objeto del programa es muy simple: solicitar las entradas del directorio actual o de uno pasado como argumento y listarlas una a una.

Las cosas serían distintas si estuviésemos desarrollando un servidor web que tiene que servir cientos o incluso miles de peticiones HTTP al mismo tiempo. Si lo hiciéramos síncronamente, se resolvería una petición cada vez. Esto reduciría considerablemente el rendimiento de la aplicación. Pues hasta que no se ha terminado de atender una, no se puede comenzar con otra. Y si el disco tiene mucho trabajo, las cosas serían peor todavía, pues aumentaría la latencia de las operaciones de E/S. Para resolver este problema, lo desarrollaríamos asíncronamente. Se solicitaría al sistema de ficheros el archivo a remitir al usuario y se pasaría a la siguiente petición. A medida que el sistema de ficheros fuese proporcionando el contenido de los archivos, iríamos cerrando peticiones. Pero tras cada cierre, podríamos tener ya otras operaciones de E/S adicionales pedidas. Se aumenta el rendimiento, la aplicación vuela y se reduce la latencia.

Creación del proyecto

Comencemos creando el proyecto:

1. Abrir una consola.
2. Crear el directorio de la práctica e ir a él. Llámelo myll.
3. Crear un proyecto de aplicación mediante el generador de Justo:

   ```
   $ justo -g node
   ```


4. Instalar dependencias:

```
$ npm install
```

5. Mostrar el catálogo de tareas automatizadas del proyecto:

```
$ justo -c
```

Creación del módulo de ayuda

Ahora, vamos a crear un módulo que exporte una función que formatea una entrada:

1. Crear el archivo lib/format.js:

```
//imports
import fs from "fs";
import path from "path";

/**
 * Format a fs entry.
 *
 * @param dir:string    The directory.
 * @param entry:string  The entry name into the directory.
 *
 * @return string
 */
export default function format(dir, entry) {
  var stats, type, usr, grp, oth;

  //(1) get stats
  stats = fs.lstatSync(path.join(dir, entry));

  //(2) determine values
  if (stats.isDirectory()) type = "d";
  else if (stats.isFile()) type = "-";
  else if (stats.isSymbolicLink()) type = "l";
  else type = "X";

  oth = (stats.mode & 4 ? "r" : "-") +
        (stats.mode & 2 ? "w" : "-") +
        (stats.mode & 1 ? "x" : "-");
  grp = (stats.mode & 40 ? "r" : "-") +
        (stats.mode & 20 ? "w" : "-") +
        (stats.mode & 10 ? "x" : "-");
  usr = (stats.mode & 400 ? "r" : "-") +
        (stats.mode & 200 ? "w" : "-") +
        (stats.mode & 100 ? "x" : "-");

  //(3) return
  return `${type}${usr}${grp}${oth} ${entry}`;
}
```

Definición del módulo comando

A continuación, vamos a definir el módulo que contiene el comando que instalará el paquete:

1. Editar el archivo bin/myll.js.
2. Modificarlo como sigue:

```
#!/usr/bin/env node

"use strict";

//imports
const fs = require("fs");
const format = require("../lib/format").default;

//data
const dir = (process.argv.length >= 3 ? process.argv[2] : process.cwd());
const entries = fs.readdirSync(dir).sort(function(a, b) {
  a = a.toLowerCase();
```



```
    b = b.toLowerCase();

    if (a < b) return -1;
    else if (a > b) return 1;
    else return 0;
  });

  //main
  for (let entry of entries) {
    console.log(format(dir, entry));
  }
```

Compilación e instalación

Para compilar e instalar el paquete, usaremos las tareas automatizadas del catálogo del proyecto build e install:

1. Ir a la consola.
2. Compilar:

   ```
   $ justo build
   ```
3. Instalar globalmente el paquete:

   ```
   $ justo install
   ```
   Recordemos que esto lo podemos hacer mediante el comando npm install como sigue:

   ```
   $ npm install -g dist/es5/nodejs/myll/
   ```
   En el fondo, es este comando lo que ejecuta la tarea install.

Prueba

Y para finalizar, la prueba:

1. Ir a la consola.
2. Invocar el comando myll como sigue:

   ```
   $ myll
   ```
3. Invocar el comando myll como sigue:

   ```
   $ myll ..
   ```

