Modelo asíncrono (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 trabajado con operaciones síncronas y asíncronas.

Objetivos

El objetivo de la práctica es mostrar cómo trabaja el modelo asíncrono de Node. Para ello, primero leeremos un archivo síncronamente y, después, lo haremos asíncronamente.

Ejecución síncrona

Para comenzar, vamos a ver cómo trabajar síncronamente con operaciones asíncronas. Parte de la funcionalidad asíncrona de los paquetes o módulos de sistema se encuentra disponible en forma síncrona. Recordemos que, cuando éste es el caso, el rendimiento se reduce, ya que la operación síncrona bloqueará el flujo de ejecución hasta que la operación de E/S finalice.

1. Abrir una consola.
2. Crear el directorio de la práctica e ir a él.
3. Crear el archivo archivo.txt con el contenido esto es el contenido:

```
$ echo "esto es el contenido" > archivo.txt
$ cat archivo.txt
esto es el contenido
$
```

4. Ejecutar node en modo interactivo.
5. Crear una función síncrona que muestre un mensaje, a continuación, el contenido de un archivo y, finalmente, otro mensaje:

```
> function muestra(ini, arch, fin) {
... console.log(ini);
... console.log(fs.readFileSync(arch).toString());
... console.log(fin);
... }
undefined
>
```

6. Ejecutar la función como sigue:

```
> muestra("INICIO", "./archivo.txt", "FIN")
INICIO
esto es el contenido

FIN
undefined
>
```

Ejecución asíncrona

Ahora, vamos a crear la misma función, pero usando la función asíncrona readFile():

1. Ir a la consola.
2. Definir la función muestraAsync():

```
> function muestraAsync(ini, arch, fin) {
... console.log(ini);
```



```
... console.log(fs.readFile(arch));
... console.log(fin);
... }
undefined
>
```

3. Invocar la función como sigue:

```
> muestraAsync("INICIO", "./archivo.txt", "FIN")
INICIO
undefined
FIN
undefined
>
```

¡No se muestra el contenido del archivo! ¿Por qué? Sencillo. Primero, revise la invocación de la función asíncrona readFile(). Hemos usado la versión asíncrona para leer el contenido. En estos casos, las funciones suelen devolver undefined, no el contenido. Recordemos que la invocación desemboca en una solicitud al sistema operativo para que realice la operación de E/S. En nuestro caso, la lectura del contenido del archivo. Una vez el sistema operativo ha terminado, avisa a node y le pasa los datos. Entonces, node añade la función *callback* adjuntada en la función asíncrona y la añade a la cola de espera. Así pues, la función asíncrona no puede devolver nada. En este punto, podemos preguntar: ¿dónde está la función *callback*? Simplemente, no la hemos definido. De ahí que no accedamos al contenido del archivo.

Así pues, cuando accedemos a datos mediante una función asíncrona, lo que hay que hacer es conocer bien su signatura. En este caso, es la siguiente:

```
function readFile(file, callback)
function readFile(file, opts, callback)
```

La función espera como primer parámetro el archivo a leer. Y como último, la función *callback*, recordemos, aquella que debe ejecutar la función cuando haya finalizado su E/S. Tan importante como la signatura de la función asíncrona, es la de la función *callback*. Depende de cada función asíncrona. En este caso:

```
function callback(error, data)
```

4. Reimplementar la función muestraAsync() para que muestre el contenido del archivo:

```
> function muestraAsync(ini, arch, fin) {
... console.log(ini);
... fs.readFile(arch, function(error, data) {
..... if (error) return console.error(error);
..... console.log(data.toString());
..... });
... console.log(fin);
... }
undefined
>
```

5. Invocar la función:

```
> muestraAsync("INICIO", "./archivo.txt", "FIN")
INICIO
FIN
undefined
> esto es el contenido
```

¿¡Y ahora qué!? ¿Por qué aparece el contenido fuera de sitio? Porque la ejecución es asíncrona. El intérprete ejecuta la proposición que le hemos indicado. No espera a que nada termine. Sin esperar a que la operación asíncrona finalice, vuelve a mostrar el *prompt*. Y mientras espera que le pasemos la siguiente proposición JavaScript a ejecutar, termina la operación asíncrona y la función *callback* se ejecuta mostrando el contenido del archivo.

Observe que las funciones de *callback* nunca interfieren el flujo normal de ejecución. Se ejecutan cuando se registran en la cola de espera y les llega el turno. Y esto no sucede hasta que el sistema operativo ha finalizado la E/S. De ahí que primero se muestre FIN y luego el contenido del archivo.

6. Reimplementar la función para que el resultado sea lo que deseamos, primero el mensaje de inicio, después el contenido y finalmente el mensaje de fin:

```
> function muestraAsync(ini, arch, fin) {
```



```
... console.log(ini);
... fs.readFile(arch, function(error, data) {
..... if (error) return console.error(error);
..... console.log(data.toString());
..... console.log(fin);
..... });
... }
undefined
>
```

7. Invocar la función:

```
> muestraAsync("INICIO", "./archivo.txt", "FIN")
INICIO
undefined
> esto es el contenido

FIN
```

Todo ha ido como esperábamos. Primero, el mensaje de inicio. Después, el contenido. Y finalmente, el mensaje de fin. Observe que los bloques que siguen a la operación de E/S se deben ubicar en la función *callback*. No hay que hacerlo tras la invocación asíncrona. Sino dentro de la función *callback* que pasamos a la función asíncrona.

No siempre esto tiene que ser así. Existe algunas ocasiones, contables con los dedos de una mano, en las que, tras invocar la operación asíncrona, se desea ejecutar algunas proposiciones JavaScript, sin esperar a que la operación de E/S haya finalizado. Pero siempre que deba trabajar con los datos devueltos por la E/S, no tendrá más remedio que añadir ese código a la función *callback*.

8. Cerrar el intérprete.

