Manejo de Archivos (2da parte): forma asincrónica

Teoría

Introducción

Ya vimos que NodeJS ofrece un módulo que nos permite trabajar sobre el sistema de archivos de nuestro sistema operativo, ya aprendimos sobre las operaciones más comunes que solemos realizar sobre archivos, y ya vimos cómo realizarlas en forma sincrónica. Nos resta entonces revisar sus versiones asincrónicas.

Notarán que todas las funciones de este apunte tienen el mismo nombre que sus versiones sincrónicas, pero sin la palabra “Sync” al final, y en cambio, todas reciben un nuevo último parámetro: un callback.

En este caso, la mayoría de los callbacks siguen la convención de que el callback deberá recibir un primer parámetro destinado al error (si lo hubiere) para saber cómo manejarlo.

Algunos callbacks requieren además un segundo parámetro, en caso de que la función en cuestión devuelva algún resultado, para indicar qué hacer con el mismo.

Dado que estas funciones poseen un parámetro que nos permite elegir cómo manejar los errores que pueden surgir de su ejecución, no será necesario ejecutarlas utilizando `try / catch`.

Leer un archivo

Para leer un archivo usaremos la función `readFile(ruta, encoding, callback)`. Recibe los mismos parámetros que su versión sincrónica, más el callback.

Al igual que su versión sincrónica, la función se encarga internamente de abrir y cerrar el archivo una vez finalizado su uso.

Ejemplo de uso:

```
fs.readFile('/ruta/al/archivo', 'utf-8', (error, contenido) => {
    if (error) {
        // hubo un error, no pude leerlo, hacer algo!
    } else {
        // en este punto del código, puedo acceder a todo el contenido
        // del archivo a través de la variable "contenido".
        console.log(contenido)
```

```
    }
})
```

Sobreescribir un archivo

Para escribir un archivo usaremos la función `writeFile(ruta, datos, callback)`. Recibe los mismos parámetros que su versión sincrónica, más el callback, que en este caso, solo precisa recibir un parámetro para manejar algún eventual error.

Al igual que su versión sincrónica, la función se encarga internamente de abrir y cerrar el archivo una vez finalizado su uso.

Ejemplo de uso:

```
fs.writeFile('/ruta/al/archivo', 'TEXTO DE PRUEBA\n', error => {
    if (error) {
        // hubo un error, no pude sobreescribirlo, hacer algo!
    } else {
        // no hubo errores, hacer algo (opcional)
        console.log('guardado!')
    }
})
```

Agregar contenidos a un archivo

Para agregar contenidos a un archivo usaremos la función appendFile`(ruta, datos, callback)`. Recibe los mismos parámetros que su versión sincrónica, más el callback, que en este caso, solo precisa recibir un parámetro para manejar algún eventual error.

Al igual que su versión sincrónica, la función se encarga internamente de abrir y cerrar el archivo una vez finalizado su uso.

Ejemplo de uso:

```
fs.appendFile('/ruta/al/archivo', 'TEXTO A AGREGAR\n', error => {
    if (error) {
        // hubo un error, no pude agregarlo, hacer algo!
    } else {
        // no hubo errores, hacer algo (opcional)
        console.log('guardado!')
    }
})
```

Renombrar un archivo

Para escribir un archivo usaremos la función `rename(rutaVieja, rutaNueva, callback)`. Recibe los mismos parámetros que su versión sincrónica, más el callback, que en este caso, solo precisa recibir un parámetro para manejar algún eventual error.

Ejemplo de uso:

```
fs.rename(rutaVieja, rutaNueva, error => {
    if (error) {
        // hubo un error, no pude renombrarlo, hacer algo!
    } else {
        // no hubo errores, hacer algo (opcional)
        console.log('renombrado!')
    }
})
```

Borrar un archivo

Para borrar un archivo usaremos la función `unlink(ruta, callback)`. El mismo parámetro que su versión sincrónica, más el callback, que en este caso, solo precisa recibir un parámetro para manejar algún eventual error.

Ejemplo de uso:

```
fs.unlink(ruta, error => {
    if (error) {
        // hubo un error, no pude borrarlo, hacer algo!
    } else {
        // no hubo errores, hacer algo (opcional)
        console.log('borrado!')
    }
})
```

Crear una carpeta

Para crear una carpeta usaremos la función `mkdir(ruta, callback)`. Recibe el mismo parámetro que su versión sincrónica, más el callback, que en este caso, solo precisa recibir un parámetro para manejar algún eventual error.

Ejemplo de uso:

```
fs.mkdir(ruta, error => {
    if (error) {
        // hubo un error, no pude crear la carpeta! hacer algo!
    } else {
        // no hubo errores, hacer algo (opcional)
        console.log('carpeta creada!')
    }
})
```

Leer el contenido de una carpeta

Para obtener los nombres de los archivos y carpetas que se encuentran dentro de una carpeta usaremos la función `readdir(ruta, callback)`. Recibe el mismo parámetro que su versión sincrónica, más el callback.

Ejemplo de uso:

```
fs.readdir(ruta, (error, nombres) => {
    if (error) {
        // hubo un error, no pude leer la carpeta! hacer algo!
    } else {
        // hacer algo con los nombres!
        console.log(nombres)
    }
})
```