

NODE.JS

FS MODULE

wawiwa

Recordatorio - try y catch mdn

La declaración try...catch marca un bloque de declaraciones para “try” (tratar) y especificar una respuesta si es que una excepción debe ser retornada.

```
try {  
    nonExistentFunction();  
} catch (error) {  
    console.error(error);  
}  
// resultado esperado: ReferenceError: nonExistentFunction is not defined  
// Nota - los errores de mensaje varían dependiendo el navegador
```

Recordatorio - try y catch mdn

```
try {  
    //...  
}  
  
try_statements  
  
catch_statements  
  
catch (exception_var) {  
    //...  
}  
  
catch_statements  
  
finally {  
    //...  
}  
  
finally_statements
```

try_statements
Las declaraciones a ser ejecutadas.

try_statements

catch (exception_var) {
 //...
}

catch_statements

Declaración que es ejecutada si una excepción es retornada en el bloque de try.

finally {
 //...
}

exception_var
Un identificador opcional que contiene un objeto de excepción para el bloque catch asociado.

finally_statements

Declaraciones que se ejecutan después de que la declaración try se complete.
Estas declaraciones se ejecutan sin importar si una excepción fue retornada o “cachada”.

Bloque “catch” incondicional

Cuando un bloque “catch” es usado, este se ejecuta cuando se lanza una excepción en el bloque “try”. Por ejemplo, cuando la excepción ocurre en el siguiente código, el control se transfiere al bloque “catch”.

```
try {  
    throw 'myException'; // genera una excepción  
}  
catch (e) {  
    // declaraciones para manejar cualquier excepción  
  
    logMyErrors(e); // pasa el objeto de excepción a manejo de errores  
}
```

Bloque “catch” condicional

Puedes crear "Bloques catch condicionales" combinando bloques try...catch con estructuras if...else if...else, así:

```
try {
    myroutine(); // puede tirar tres tipos de excepciones
} catch (e) {
    if (e instanceof TypeError) {
        // declaraciones para manejar excepciones de TypeError
    } else if (e instanceof RangeError) {
        // declaraciones para manejar excepciones de RangeError
    } else if (e instanceof EvalError) {
        // declaraciones para manejar excepciones de EvalError
    } else {
        // declaraciones para manejar cualquier excepción no especificada
        logMyErrors(e); // pasa el objeto de excepción al manejador de errores
    }
}
```

El identificador de excepciones

Cuando se lanza una excepción en el bloque “try”, exception_variable (por ejemplo, la “e” en catch (e)) tiene un valor de excepción. Puedes usar este identificador para obtener información de la excepción lanzada. Este identificador está disponible únicamente en la extensión del bloque “catch”. Si no necesitas el valor de la excepción, puede ser omitida.

```
function isValidJSON(text) {  
  try {  
    JSON.parse(text);  
    return true;  
  } catch {  
    return false;  
}
```

El bloque finally

El bloque “finally” contiene declaraciones para ejecutar después de que los bloques “try” y “catch” se ejecuten, pero antes que los siguientes bloques try...catch...finally. Nota que el bloque “finally” se ejecuta si una excepción es lanzada o no. También, si una excepción es lanzada, las declaraciones en el bloque “finally” se ejecutan aunque el bloque “catch” no maneje la excepción.

El siguiente ejemplo muestra un caso de uso del bloque “finally”. El código abre un archivo, después ejecuta las declaraciones que ese archivo usa, el bloque “finally” se asegura que el archivo siempre se cierre después de que se use incluso si se lanza una excepción.

```
openMyFile();  
try {  
    // atar un recurso  
    writeMyFile(theData);  
} finally {  
    closeMyFile(); // siempre cerrar el recurso  
}
```

Leer el contenido de un archivo

Usando node, podemos usar el sistema de archivos "module" para manejar archivos en nuestro servidor.

El primer programa que construiremos leerá el contenido del archivo.

Crea un archivo llamado a.txt y escribe en el archivo la frase 'hello world':

```
a.txt  x  
chap4 - fs > a.txt  
1 hello world
```

Ler el contenido de un archivo

f1.js

```
const fs = require("fs");

try {
  const data = fs.readFileSync("a.txt", 'utf-8');
  console.log(data);
} catch (e) {
  console.log("can't read from a.txt");
}
```

Lee los archivos de a.txt.
"Sync" significa que no
continuaremos hasta que
terminemos de leerlos

En caso de haber un problema leyendo
los archivos de la biblioteca (la biblioteca
no existe, etc...)

```
asafamir@Asafs-MBP chap4-fs % node f1
hello world
```

Más-Leer el contenido de un archivo async

En lib/first.html escribe "hello!", para que después se pueda leer. f2_read_file_content_async.js

```
const fs = require("fs");
const path = require("path");

const filePath = path.join(__dirname, "lib", "first.html");
```

“path” es un módulo incorporado en node para construir la ruta de un archivo

dirname es el directorio actual de la ruta.

Entonces la ruta final es:
C:\Users\X5-i7\OneDrive\Desktop\Programming\Asaf\my-node-project\chap3\lib\first.html

```
const stats = fs.statSync(filePath);

if (stats.isFile()) {
  fs.readFile(filePath, "UTF-8", (err, contents) => {
    console.log(contents);
  });
}
```

Debemos esperar a que el módulo “path” termine de construirse antes de que podamos continuar continue

fs considera directorios como archivos también, entonces debemos verificar que en realidad sea un archivo

Escribir en un archivo síncrono

f3_write_file_sync.js

```
const fs = require("fs");
const path = require("path");
```

```
const content = `
So
Much
Content!!!
`;
```

```
fs.writeFileSync("a.txt", content.trim());
```

El nombre del archivo será a.txt

El método trim() elimina los espacios blancos de los dos lados del string.

Escribir en un archivo asíncrono

f4_write_file_async.js

```
const fs = require("fs");
const path = require("path");

const content = `
so
Much
Content!!!`;
```

```
const filePath = path.join(__dirname, "lib", "f1.txt");

fs.writeFile(filePath, content.trim(), err => {
  if (err) {
    console.log(err);
  } else {
    console.log("file created");
  }
});
```

The file name will be
'f1.txt' and it will be under lib

The trim() method removes
whitespace from both sides
of a string.

Escribir un archivo- adjuntar un archivo

f5-append_content.js

También podemos adjuntar contenido al final de un archivo existente:

```
const fs = require("fs");
const path = require("path");

const filePath = path.join(__dirname, "lib", "f1.txt");

fs.appendFile(filePath, "hi - how do u feel?", err => {
  if (err) {
    console.error(err);
  } else {
    console.log("appended content");
  }
});
```

Escribir un archivo

Entonces el resultado final de f1.txt es:

So

Much

Content!!hi - how do u feel?

Hazlo tú mismo 1

Crea un programa en node que **escriba sincrónicamente** a un archivo llamado test.txt con tu nombre completo.

Hazlo tú mismo 2

Crea un programa en node que **escriba asíncronicamente** a un archivo llamado test.txt con tu nombre completo.

Hazlo tú mismo 3

Crea un programa en node que **lea sincrónicamente** un archivo llamado test.txt y que imprima el texto a en la consola

Hazlo tú mismo 4

Crea un programa en node que **lea asíncronicamente** un archivo llamado test.txt y que imprima el texto a en la consola

Renombrar un archivo

f6_rename_sync.js

Asíncrono:

```
fs.renameSync("./lib/f1.txt", "f1 after.txt");
console.log("f1 rename to 'f1 after'");
```

¡Atención!

También cambiamos la ruta,
de ./lib/ a nuestro dir actual

f6b_rename_async.js

Asynchronous:

```
fs.rename("./lib/f1.txt", "f1 after.txt", err => {
  if (err) {
    console.log(err)
  } else {
    console.log("file move");
  }
});
```

Borrar un archivo

f7_delete_file_sync.js

Síncrono:

```
try {
  fs.unlinkSync("./f1.txt");
} catch (e) {
  console.log(e);
}
```

Borrar un archivo

f8_delete_file_async.js

Asíncrono:

```
fs.unlink("f1.txt", err => {
  if (err) {
    console.error(err);
  } else {
    console.log("file deleted");
  }
});

console.log("hi");
```

Crear un Directorio

fs9_make_dir.js

```
if (!fs.existsSync("mydir")) {  
  fs.mkdir("mydir", err => {  
    if (err)  
      console.error(err);  
  
    else  
      console.log("directory created");  
  });  
}  
  
else {  
  console.log("directory exist");  
}
```

Crea el directorio solamente si todavía no existe la ruta relativia entonces lo busca y lo crea, la ruta es relativa entonces lo busca y lo crea bajo 'chap 4', o nuestro dir actual

Leer un archivo de un directorio - sincrónicamente

f10_read_file_names_sync.js

En fs1.js:

```
try {
  const files = fs.readdirSync("./lib");
  console.log(files);
} catch (e) {
  console.error("can't read from lib");
}

console.log("hello world");
```

Ler archivos del lib.

Sync significa que no va a continuar hasta que termine de leer

En caso de que haya un problema al leer los archivo del lib (el lib no exista, etc...)

Ler un archivo de un directorio - asincrónicamente

f11_read_file_names_sync.js

Y otro ejemplo. En fs2.js:

```
try {
  fs.readdir("./lib", (err, files) => {
    if (err) throw err;
    console.log(files);
  });
} catch (e) {
  console.log("can't read files");
}

console.log("hello world");
```

Ler los archivos de lib.
Esto va a ser asíncrono así que
el programa va a continuar sin
esperar a los archivos

En caso de error al leer,
podemos manejarlo o tirarlo
y "cacharlo" afuera

Leer una lista de archivos

El resultado de con asíncrono es:

```
hello world  
[ 'first.html' , 'second.html' ]
```

Porque leer los archivos toma más tiempo que registrarlos.

Ler el contenido de los archivos del directorio

f12_read_files content_async.js

Un ejemplo para leer los contenidos de todos los archivos en un directorio:

```
const fs = require("fs");
const path = require('path');

fs.readdir("./lib", (err, files) => {
  files.forEach(filename => {
    const filePath = path.join(__dirname, "lib", filename);
    const stats = fs.statsSync(filePath);

    if (stats.isFile()) {
      fs.readFile(filePath, "UTF-8", (err, contents) => console.log(contents));
    }
  });
});

Output : hello from second too!
hello!
```

Borrar un directorio

f13_delete_dir.js

```
fs.readdirSync("./lib/data").forEach(filename =>
  fs.unlinkSync("./lib/data/" + filename));
}

fs.rmdir("./lib/data", err => {
  if (err) {
    console.error(err);
    return;
  }
  console.log("remove data directory");
});
```

Primero selecciona todos los archivos en el directorio "data"

Borra el directorio 'data' ahora que está vacío

Hazlo tú mismo 1

Crea un programa en nodo que escriba **sincrónicamente** un archivo llamado memory.txt con tu ciudad y tu país.

Hazlo tú mismo 2

Crea un programa en nodo que escriba **asincrónicamente** un archivo llamado memory.txt con tu ciudad y tu país.

Hazlo tú mismo 3

Escribe un programa que lea **sincrónicamente** de un archivo llamado `test.txt` e imprime el texto en la consola.

Crea un archivo servidor

f14_files_server.js

Retorna al cliente el contenido de un archivo:

```
const http = require('http');
const fs = require('fs');

http.createServer((req, res) => {
  fs.readFile("./lib/first.html", (err, data) =>
    res.setHeader('Content-Type', 'text/html').end(data));
}).listen(3000);
```

Lee el archivo
first.html' najo 'lib'

Crea una respuesta





wawiwa

¿Preguntas?