try...catch

La declaración try...catch señala un bloque de instrucciones a intentar (try), y especifica una respuesta si se produce una excepción (catch).

Sintaxis

```
try {
   try_statements
}
[catch (exception_var_1 if condition_1) { // non-standard
   catch_statements_1
}]
...
[catch (exception_var_2) {
   catch_statements_2
}]
[finally {
   finally_statements
}]
```

`try_statements`
: Las sentencias que serán ejecutadas.

`catch_statements_1`, `catch_statements_2`
: Sentencias que se ejecutan si una excepción es lanzada en el bloque `try`.

`exception_var_1`, `exception_var_2`
: Identificador que contiene un objeto de excepcion asociado a la cláusula `catch`.

`condition_1`
: Una expresión condicional.

`finally_statements`
: Sentencias que se ejecutan después de que se completa la declaración `try` . Estas sentencias se ejecutan independientemente de si una excepcion fue lanzada o capturada.

Descripción

La sentencia try consiste en un bloque try que contiene una o más sentencias. Las llaves {} se deben utilizar siempre , incluso para una bloques de una sola sentencia. Al menos un bloque catch o un bloque finally debe estar presente. Esto nos da tres formas posibles para la sentencia try :

1. try...catch
2. try...finally
3. try...catch...finally

Un bloque catch contiene sentencias que especifican que hacer si una excepción es lanzada en el bloque try . Si cualquier sentencia dentro del bloque try (o en una funcion llamada desde dentro del bloque try) lanza una excepción, el control cambia inmediatamente al bloque catch . Si no se lanza ninguna excepcion en el bloque try , el bloque catch se omite.

La bloque finally se ejecuta despues del bloque try y el/los bloque(s) catch hayan finalizado su ejecución. Éste bloque siempre se ejecuta, independientemente de si una excepción fue lanzada o capturada.

Puede anidar una o más sentencias try . Si una sentencia try interna no tiene una bloque

Cuando solo se utiliza un bloque catch , el bloque catch es ejecutado cuando cualquier excepción es lanzada. Por ejemplo, cuando la excepción ocurre en el siguiente código, el control se transfiere a la cláusula catch .

```
try {
    throw "myException"; // genera una excepción
}
catch (e) {
    // sentencias para manejar cualquier excepción
    logMyErrors(e); // pasa el objeto de la excepción al manejador de error
}
```

El bloque catch especifíca un identificador (e en el ejemplo anterior) que contiene el valor de la excepción. Este valor está solo disponible en el scope de el bloque catch

Tambien se pueden crear "bloques catch condicionales", combinando bloques try...catch con estructuras if...else if...else como estas:

```
try {
      myroutine();     // puede lanzar tres tipos de excepciones
} catch (e) {
      if (e instanceof TypeError) {
            //    sentencias para manejar excepciones TypeError }
      else if (e instanceof RangeError) {
            //    sentencias para manejar excepciones RangeError }
      else if (e instanceof EvalError) {
            //    sentencias para manejar excepciones EvalError }
      else {
          //    sentencias para manejar cualquier excepción no especificada
```

```
  } catch {
    return false;
  }
}
```

La cláusula **finally**

La cláusula finally contiene sentencias a ejecutarse después de que las cláusulas try y catch se ejecuten, pero antes de las sentencias que le siguen al bloque try..catch..finally . Note que la cláusula finally se ejecuta sin importar si una excepción es o no lanzada. Si una excepción es lanzada, las instrucciones en la cláusula finally se ejecutan incluso si ninguna cláusula catch maneja la excepción.

Usted puede usar la cláusula finally para hacer que su script falle plácidamente cuando una excepción ocurra; por ejemplo, para hacer una limpieza general, usted puede necesitar liberar un recurso que su script haya retenido.

Puede parecer extraño tener una cláusula relacionada a una excepción que se ejecuta sin importar si hay una excepción o no, pero esta concepción en realidad sirve a un propósito. El punto importante no es que la cláusula finally siempre se ejecuta, si no más bien que el codigo ordinario que le sigue a try..catch no.

Por ejemplo, si otra excepción ocurre dentro de un bloque catch de una declaración try , cualquier codigo restante en el mismo bloque exterior try que encierra ese try..catch (o en el flujo principal, si no es un bloque try exterior) , no será ejecutado, dado que el

mientras el archivo está abierto, la cláusula finally cierra el archivo antes de que el script

falle. El código en finally también se ejecuta después de un retorno explícito de los

bloques try o catch .

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

Ejemplos

Bloques try anidados

Primero, veamos que pasa con esto:

```
try {
  try {
    throw new Error('oops');
  }
  finally {
    console.log('finally');
  }
}
catch (ex) {
  console.error('outer', ex.message);
}

// Output:
// "finally"
// "outer" "oops"
```

Ahora, si nosotros ya capturamos la excepción en una declaración try interna agregando un bloque catch.

```
try {
  try {
    throw new Error('oops');
  }
  catch (ex) {
    console.error('inner', ex.message);
  }
  finally {
    console.log('finally');
  }
}
catch (ex) {
  console.error('outer', ex.message);
}

// Output:
// "inner" "oops"
// "finally"
```

Y ahora vamos a relanzar el error.

```
try {
  try {
    throw new Error('oops');
  }
  catch (ex) {
    console.error('inner', ex.message);
    throw ex;
  }
  finally {
    console.log('finally');
  }
}
catch (ex) {
  console.error('outer', ex.message);
}

// Output:
// "inner" "oops"
// "finally"
// "outer" "oops"
```

Cualquier excepción dad será capturada solo una vez por el bloque catch más cercano a menos que sea relanzado. Por supuesto cualquier nueva excepción que se origine en el bloque 'interno' (porque el código en el bloque catch puede hacer algo que lanze un error), será capturado por el bloque 'externo'.

Retornando de un bloque finally

Si el bloque finally retorna un valor, este valor se convierte en el valor de retorno de toda la producción try-catch-finally, a pesar de cualquier sentencia return en los bloques try y catch. Esto incluye excepciones lanzadas dentro del bloque catch.

```
(function() {
  try {
    try {
      throw new Error('oops');
    }
    catch (ex) {
      console.error('inner', ex.message);
      throw ex;
    }
    finally {
      console.log('finally');
      return;
    }
  }
  catch (ex) {
    console.error('outer', ex.message);
  }
})();

// Output:
// "inner" "oops"
// "finally"
```

El "oops" externo no es lanzado debido al retorno en el bloque finally. Lo mismo aplicaría para cualquier valor retornado del bloque catch.