Excepciones

Antonio Espín Herranz

Contenidos

- Usando try / catch
- Lanzar nuestras excepciones
- Usando finally

Captura de excepciones

- Utilizamos las palabras try y catch para monitorizar un bloque de instrucciones y poder capturar los posibles errores en tiempo de ejecución.

- El bloque catch se utiliza para realizar el tratamiento de la excepción.

```
try {
        //intento algo que puede producir un error
} catch(mierror){
          //hago algo cuando el error se ha detectado
}
```

Captura de excepciones

- El Bloque try se ejecuta tal cual, hasta que un posible error se ha detectado.
- Si no se detecta un error durante la ejecución del bloque try, el catch se deja del lado y no se realiza.
- En caso que sí se detecte un error en el bloque try, se ejecuta las sentencias que teníamos en el catch.
- Tener en cuenta que dentro del bloque try si se produce un error en la línea i, las instrucciones i+1, i+2 etc. Ya no se ejecutan.

Ejemplo

```
try {
  //intento algo que puede producir un error
  funcion_que_no_existe()
}catch(mierror){
  alert("Error detectado: " + mierror.description)
}
```

También es válido:

```
try {
        throw ‘mensaje’;
} catch (err){
        console.log(err);
}
```

Lanzar excepciones: **throw**

- **La forma más sencilla para lanzar errores es utilizando *throw***. Este comando permite enviar al navegador un evento similar al que se produce cuando ocurre algún imprevisto o nos encontramos ante un tipo inesperado de datos.

- El lenguaje **permite enviar todo tipo de elementos, incluyendo texto, números, valores booleanos o incluso objetos**.

- La otra opción es enviar el objeto nativo ***Error***:
  - **throw new Error**("Something bad happened.");

Objeto Error

- El objeto **Error** tiene dos propiedades: **name** y **message**
- Que se suelen utilizar dentro de bloque catch para su tratamiento.

Ejemplo con una cadena

```
function funcion(cadena) {

    try {
        if (cadena == null) throw "Cadena vacía";
        console.log('la cadena es: ', cadena);

    } catch (err) {
        console.log("ERROR: ", err);

    } finally {
        console.log('finally');

    }
}
```

Ejemplo con un objeto Error

```
function funcion2(cadena) {

    try {
        if (cadena == null) throw new Error("Cadena vacía");
        console.log('la cadena es: ', cadena);

    } catch (e) {
        console.log("ERROR: ", e.name, ' ', e.message);

    } finally {
        console.log('finally');

    }
}
```

finally

- Este bloque va después del catch y se ejecuta siempre haya o no error.

try {

    - Bloque de código a evaluar

} catch (){

    - Tratamiento de la excepción

} finally {

    - Este bloque se ejecutará siempre, incluso si dentro de una función, hacemos un return dentro del bloque catch.

}

Excepciones personalizadas

- Heredamos de la clase Error, se pueden añadir otras propiedades adicionales: el nombre, la fecha/hora, etc.
- Se le pasarán los parámetros a la clase padre Error.
- Utilizaremos la declaración **…params**
- Tener en cuenta que nuestro constructor del error personalizado se le mandarán al menos 2 parámetros: por ejemplo: el nombre y el mensaje que pasará a la clase padre.

Ejemplo

```
class MiError extends Error {
    constructor(nombre = "", ...params) {
        // Pasar los parametros a la clase Error (la padre)
        super(...params);

        this.name = nombre;
        this.fecha = new Date();
    }
}

try {
    throw new MiError('MiError', 'Este es un error de prueba');

} catch (e) {
    console.log('name: ', e.name);
    console.log('message: ', e.message);
    console.log('fecha: ', e.fecha);
}
```

Tipos de error

- Además del constructor genérico Error, hay otros siete constructores de errores en el núcleo de JavaScript. Para conocer las excepciones de lado del cliente.

  - **EvalError**
    - Crea una instancia que representa un error que ocurre con respecto a la función global eval() (en-US).
  - **InternalError**
    - Crea una instancia que representa un error que ocurre cuando se produce un error interno en el motor de JavaScript. Por ejemplo: "demasiada recursividad".
  - **RangeError**
    - Crea una instancia que representa un error que ocurre cuando una variable numérica o parámetro está fuera de su rango válido.
  - **ReferenceError**
    - Crea una instancia que representa un error que ocurre cuando se quita la referencia a una referencia no válida.
  - **SyntaxError**
    - Crea una instancia que representa un error de sintaxis.
  - **TypeError**
    - Crea una instancia que representa un error que ocurre cuando una variable o parámetro no es de un tipo válido.
  - **URIError**
    - Crea una instancia que representa un error que ocurre cuando encodeURI() o decodeURI() pasan parámetros no válidos.

Como capturar el tipo de error

```
try {
   // bloque de instrucciones que pueden generar una excepción
} catch (err){
    if (err instanceof ZeroDivisionError)
        console.log('Salta zero')

    else if (err instanceof RangeError)
        console.log('Salta Range')

    else if (err instanceof SyntaxError)
        console.log('Salta syntax')

    else if (err instanceof ColeccionError)
        console.log('Salta Col:'+err.message)

    else
        console.log('otro tipo: '+err.name+" "+err.message)
} finally {
    console.log('Se ha ejecutado finally')
}
```

Las clases generadas

```
class ZeroDivisionError extends Error {

    constructor(...params){
        super(params)
        this.name = 'ZeroDivisionError'
    }
}

class ColeccionError extends Error {

    constructor(...params){
        super(params)
        this.name = 'ColeccionError'
    }
}
```

Enlaces

- http://www.etnassoft.com/2011/01/30/excepciones-en-javascript/
- https://es.javascript.info/custom-errors
- https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Error