Tema 8: Excepciones en JavaScript

Javascript, como otros muchos lenguajes de programación, incluye soporte nativo para el manejo de excepciones.

Las excepciones, son errores o imprevistos que ocurren durante la ejecución de un programa que alteran el flujo normal de la ejecución.

Su función, es **separar el código para el manejo de errores de la lógica de aplicación del programa**. En aquellos lenguajes que incluyen soporte para el manejo de excepciones, al producirse un error, se desciende en la pila de ejecución hasta encontrar un manejador para la excepción. Ese manejador toma el control de la ejecución desde el momento en el que produce el error.

Por lo general, los errores en Javascript son crípticos y poco informativos, así que las excepciones nos valen para **manejar errores con nuestros propios mensajes y realizando acciones cuando algo vaya mal**.

Excepciones con 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**. Sin embargo, la opción más usual es enviar el objeto nativo *Error*:

```
throw new Error( "Algo malo ha ocurrido." );
```

Un ejemplo típico de uso es insertarlo como estamento en un condicional:

```
function sumar(){
  var result = 0,
  if(arguments.length > 10 ) throw console.error('Demasiados datos' );
    for( var i = 0; i < l; i++ ){
    result += arguments[x];
  }
  return result;
}

console.log( sumar( 3, 4, 34, 5, 7, 8, 1, 32 ) );  // 94
console.log( sumar( 3, 4, 34, 5, 7, 8, 1, 32, 3, 5, 8 ) );  // Error:
Demasiados datos
// El resto del código, será ignorado tras lanzar la excepción...
```

La función anterior suma el valor numérico de los argumentos que le pasemos pero, si la pasamos más de 10 lanza una excepción abortando la ejecución.

Es importante tener en cuenta que *throw* **detiene completamente la ejecución del hilo actual (no sólo el ámbito o contexto del error), por lo que no el resto del código**

será inmediatamente ignorado. Es por ello que se considera una buena práctica el colocar el estamento *throw* al final del resto de evaluaciones en un condicional:

```
function comprobarNumbero( valor ){
  var result = "";
  if( isNaN(valor)) throw(console.error( valor + ' no es un
número!'));
  if(parseInt(valor) < 0) result = valor + ' menor que 0!';
  if(parseInt(valor) > 1000) result = valor + ' es mayor que 1000!';

  return result;
}

console.log(comprobarNumbero(-17));
console.log(comprobarNumbero('10'));
console.log(comprobarNumbero(1009));
console.log(comprobarNumbero('no')); //Excepcion
```

Excepciones con Try / Catch

Try - Catch corresponde a un tipo de estructura de control Javascript con la que comprobar el flujo de un programa frente a comportamientos inesperados. La diferencia entre esta estructura y la anterior es que mientras *throw* detiene completamente la ejecución, **catch realiza una acción determinada frente a los errores para proseguir después con el flujo definido**.

Este tipo de excepciones se estructuran mediante un bloque de código que evalúa una condición previa y **propone en consecuencia una ejecución predefinida y otra alternativa frente a anormalidades**. La sintaxis es sencilla y actúa como un condicional más:

```
function comprobarPass( texto ){
  var msg = {};
  try {
    if( texto.length < 6 ) throw 'Corto';
    if( texto.length > 10 ) throw 'Largo';
    msg.status = 'El password es correcto';
  } catch( err ) {
    if( err == 'Corto' ) msg.status = 'El password es muy corto';
    if( err == 'Largo' ) msg.status = 'El password es muy Largo';
  } finally {
    console.log( 'Error password: ' + msg.status );
  }
}

comprobarPass( '1234' );
comprobarPass( '12345678901' );
comprobarPass( '12345678' );

//La ejecución continua...
```

Conclusión

En el desarrollo de software, el manejo de excepciones es una **técnica necesaria para asegurarnos un correcto funcionamiento de nuestras aplicaciones**. En el caso de Javascript, este factor tiene una mayor importancia si cabe dado que hablamos de un lenguaje donde los errores suelen aparecer de una forma silenciosa: por lo general, el usuario no se percatará (ni será advertido) de que algo ha fallado; únicamente la ejecución se detendrá sin más.

Es por esto que las excepciones, debemos entenderlas como algo más que una herramienta para el desarrollador. Tenemos que entender las excepciones como una **metodología más que permite mejorar la experiencia de usuario y aplicar una capa más de control sobre los flujos de datos que manejamos**.

Usando las estructuras de control que hemos mostrado correctamente, podemos evitar la aparición de errores incomprensibles (y bloqueantes) permitiéndonos reconducir el flujo según lo necesitemos sin detener el programa.

Bibliografía

https://www.nczonline.net/blog/2009/03/03/the-art-of-throwing-javascript-errors/

https://www.sitepoint.com/debugging-javascript-throw-away-your-alerts/