APRENDERAPROGRAMAR.COM

ERROR, THROW Y TRY CATCH JAVASCRIPT . MESSAGE, FILENAME, LINENUMBER. EVAL Y RANGEERROR, REFERENCE ERROR. EJEMPLOS (CU01187E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº87 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

TRATAMIENTO DE ERRORES

Los programadores JavaScript se enfrentan con frecuencia a la aparición de errores durante la ejecución del código en los navegadores. Al igual que otros lenguajes de programación, JavaScript permite la captura de esos errores con instrucciones que permiten identificar y tratar dichos errores.

ERROR, THROW Y TRY CATCH

La gestión de errores en tiempo de ejecución tiene como elementos principales las sentencias try – catch, throw, y el tipo de datos definido por JavaScript Error.

El tipo de datos Error es un tipo de datos predefinido de JavaScript que permite crear objetos de tipo Error. Los objetos de tipo Error son creados automáticamente por JavaScript cuando se produce un error en tiempo de ejecución, pero también podemos definir la creación de objetos Error a través del código.

Un objeto Error representa un error y tiene propiedades asociadas (por ejemplo propiedades que informan del tipo de error de que se trata). Normalmente un error no se crea y "se guarda", sino que un error "se lanza" (is thrown). Al lanzarse un error, el flujo o ejecución prevista del script se verá alterado. Esa alteración puede dar lugar a la detención del script si el error impide que prosiga la ejecución, o a que el script continúe ejecutándose con anomalías, etc.. Como forma de prevenir que el script se detenga sin más, o que aparezcan disfunciones no previstas, existe la posibilidad de capturar y tratar el error usando las sentencias try – catch.

Un esquema típico para tratamiento de errores será el siguiente:

```
try {
//Código que vamos a ejecutar
// Si se produce un error se lanza una excepción y se salta al catch
}
catch (e) {
  // e representa el error lanzado
  // mensajes de alerta, acciones a ejecutar, etc.
}
```

Vamos a partir del siguiente ejemplo que al ser ejecutado dará lugar a que salte un error. Escribe este código, activa la consola de tu navegador y comprueba cómo al ejecutar el código te aparece un mensaje de error cuando haces click sobre el texto "Probar".

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploError() {
cambiarImagen(imagen1);
alert('Se cambió la imagen'+window.imagen1);
alert('A continuación se le solicitarán los datos fiscales');
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploError()"> Probar </div>
</body>
</html>
```

El resultado esperado es que no ocurra nada y que en la consola se muestre un mensaje de error similar a este: <<ReferenceError: cambiarImagen is not defined>>. Este error nos informa de que se está invocando una función que no está definida (además tenemos un segundo error, ya que imagen1 tampoco está definido).

Supongamos que este error aparezca "sin querer", debido a un error del programador o debido a que el usuario o el código HTML no facilitan los datos que se espera. Un bloque try catch puede envolver a un fragmento de código (tan amplio como se desee) y establecer un tratamiento para el error, de modo que el script continuará ejecutándose siempre que sea posible.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploError() {
try{
        cambiarImagen(imagen1);
        alert('Se cambió la imagen'+imagen1);
}
catch(e){ alert('Se produjo un error. Referencia: '+e);}
alert('A continuación se le solicitarán los datos fiscales');
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploError()"> Probar </div>
</body></html>
```

El resultado esperado es que se muestre una ventana de mensaje de alerta con un texto similar al siguiente: <<Se produjo un error. Referencia: ReferenceError: cambiarImagen is not defined>> y que seguidamente se muestre el mensaje <<A continuación se le solicitarán los datos fiscales>>.

La diferencia entre este código y el anterior donde no existía un bloque try catch es que como comprobamos aquí el error se captura, es tratado y la ejecución del script continúa. En el caso anterior no llegábamos a ver el mensaje "A continuación se le solicitarán los datos fiscales" porque el script quedaba detenido.

Un bloque try catch se introducirá, típicamente, en fragmentos de código donde sea previsible que pueda producirse un error, con el fin de capturarlo y tratarlo.

Otro esquema típico para tratamiento de errores será el siguiente:

```
try {
//Código que vamos a ejecutar
if (evaluación indica que existe un error) {throw new Error("Descripción del error"); }
}
catch (e) {
  // e representa el error lanzado
  // mensajes de alerta, acciones a ejecutar, etc.
}
```

Escribe este código y comprueba sus resultados al introducir diferentes números:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploError() {
try{
        var numeroUsuario = prompt('Introduzca un número del 1 al 9 por favor: ');
        if (isNaN(numeroUsuario)|| numeroUsuario<1 ||numeroUsuario>9) {
                throw new Error('Número introducido no válido');
        }
}
catch(e){ alert('Se produjo un error. Referencia: '+e);}
alert('A continuación se le solicitarán los datos fiscales');
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploError()"> Probar </div>
</body></html>
```

El resultado esperado es que si introducimos un número que se considera válido (por ejemplo el 4), la ejecución se realice con normalidad. Por el contrario, si introducimos sun número que se considera no válido como el 88, se lanzará la excepción y se mostrará el mensaje <<Número introducido no válido>>. Aquí realmente no necesitábamos lanzar una excepción (ya que podríamos haber controlado el proceso de otra manera), por lo que este código debemos tomarlo únicamente como “un ejemplo” y no como una referencia de cómo hacer las cosas. Lo más habitual entre programadores es no lanzar excepciones excepto ante circunstancias que difícilmente pueden ser tratadas de otra manera (por ejemplo que el acceso a un recurso que normalmente esté disponible, como un fichero, pueda no estar disponible en un momento concreto).

CLÁUSULA FINALLY

Una instrucción try además de bajo el esquema try {...} catch { ... } que hemos visto anteriormente, se puede combinar con una cláusula finally de alguna de estas maneras: try {...} catch {...} finally {...} y también con try {...} finally {...}.

Las sentencias incluidas dentro de una cláusula finally se ejecutarán independientemente de que se haya producido un error o no durante la ejecución del bloque try. El objetivo habitual de una cláusula finally es liberar un recurso (por ejemplo cerrar un archivo) que haya podido ser comprometido anteriormente.

El esquema habitual será el siguiente:

```
abrirElRecurso(); //Por ejemplo accedemos a un fichero
try {
  // Ejecutamos acciones previstas con el recurso
  escribirEnElFichero(fechaActual);
}
finally {
  cerrarElRecurso(); // Por ejemplo cerramos el fichero
}
```

Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploError() {
try{
        var numeroUsuario = prompt('Introduzca un número del 1 al 9 por favor: ');
        if (isNaN(numeroUsuario)|| numeroUsuario<1 ||numeroUsuario>9) {
                throw new Error('Número introducido no válido');
        }
}
catch(e){ alert('Se produjo un error. Referencia: '+e);}
finally {alert('Se ejecuta siempre haya o no captura de error');}
alert('A continuación se le solicitarán los datos fiscales');
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploError()"> Probar </div>
</body></html>
```

El resultado esperado es que el mensaje <<Se ejecuta siempre haya o no captura de error>> aparezca tanto cuando se lanza un error como cuando no se lanza.

TIPOS ESPECIALIZADOS DE ERROR

Además de los objetos Error que indican un error genérico, podemos lanzar errores especializados. Entre los errores especializados tenemos los siguientes:

Tipo de error	Significado	Ejemplo
EvalError	Representa un error relacionado con la ejecución de la función eval()	throw new EvalError();
RangeError	Representa que un valor está fuera del rango permitido para una variable o parámetro.	throw new RangeError();
ReferenceError	Representa que se ha intentado invocar una función u objeto y no existe en ese ámbito	throw new ReferenceError();
SyntaxError	Representa un error en la sintaxis del código que se pretende ejecutar con eval()	throw new SyntaxError();
TypeError	Representa un error debido a que una variable o parámetro no tienen un tipo válido.	throw new TypeError();
URIError	Representa un error cuando se pasan parámetros no válidos a las funciones encodeURI() ó decodeURI()	Throw new URIError();

ANIDAMIENTO DE TRY

Se pueden incluir bloques try dentro de otros bloques try. Normalmente cada bloque try llevará su catch correspondiente, pero en caso de que un bloque try interno a otro carezca de catch, en caso de lanzarse una excepción se accederá al catch del bloque externo.

PROPIEDADES DE LOS OBJETOS ERROR

La creación de objetos Error puede realizarse indicando simplemente new Error() o bien introduciendo parámetros. Los parámetros admitidos son:

```
new Error (opcional_message, opcional_fileName, opcional_lineNumber);
```

Esta información puede ser usada durante el tratamiento del error de la forma en que se considere oportuno. Por ejemplo podríamos escribir:

catch(e) { alert ('Se produjo un error. Referencia: '+e.message + ' Línea: '+e.lineNumber +' Fichero: '+e.fileName);}

Los errores que se lanzan automáticamente llevarán asociado un contenido para estas propiedades de forma automática. Al mensaje descriptivo se puede acceder con la propiedad message, y al nombre de archivo y número de línea con las propiedades lineNumber y fileName.

En los errores que lanzamos nosotros con new Error las propiedades del objeto pueden ser establecidas por nosotros, pero si no establecemos las propiedades, éstas tomarán los valores asociados automáticamente.

Otra propiedad de los objetos Error es name. Esta propiedad inicialmente toma el valor <<Error>>, pero podemos establecer por ejemplo e.name ='Error-Usuario'.

Al invocar e.toString() se devuelve la concatenación de name con message.

EJERCICIO

Un programador ha desarrollado un código y nos han pedido que lo revisemos. Escribe este código en un editor, ejecútalo y responde a las siguientes preguntas:

```
<html>
 <head>
<meta charset="utf-8">
<style type="text/css"> input {margin:10px;} </style>
<script>
function validarPassword(password){
  try  {
    if(password.length < 5 ) {      throw "SHORT";    }else if(password.length > 10 ) {  throw "LONG";    }
    alert("Password Validated!");
  } catch(e) {
    if(e == "SHORT"){  alert("Not enough characters in password!"); }
    else if(e == "LONG"){ alert("Password contains too many characters!");   }
  }finally{    document.miFormulario.password.value="";   }
  alert("La revisión ha terminado.");
}
 </script>
   </head>
   <body>
      <form name="miFormulario" onsubmit="validarPassword(document.getElementById('pass').value)" action="#" >
         Nombre de usuario: <input type="text" name="campo1"><br>
         Password: <input id="pass" type="password" name="password"><br>
         <input type="submit" value="Comprobar" name="comprobar">
      </form>
   </body>
</html>
```

a) Busca información en internet y respode: ¿Qué significado tiene una instrucción como throw "SHORT";? ¿A qué da lugar? ¿Qué diferencia hay entre throw "SHORT" y throw new Error('SHORT')?

b) ¿Cuál es el objetivo que parecía pretender cumplir el autor del código?

c) ¿En qué casos se ejecuta la cláusula finally incluida en el código?

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01188E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206