Tipos de Errores en JavaScript

Hay una serie de tipos de error predefinidos en JavaScript. Son elegidos y definidos automáticamente por el entorno de ejecución de JavaScript cuando el programador no maneja explícitamente los errores en la aplicación.

Esta sección te guiará a través de algunos de los tipos de error más comunes en JavaScript y entenderás cuándo y por qué se producen.

RangeError

Un RangeError se lanza cuando se establece una variable con un valor fuera de su rango de valores legales. Suele ocurrir cuando se pasa un valor como argumento a una función, y el valor dado no se encuentra en el rango de los parámetros de la función. A veces puede ser difícil de solucionar cuando se utilizan bibliotecas de terceros mal documentadas, ya que necesitas conocer el rango de valores posibles de los argumentos para pasar el valor correcto.

Algunas de las situaciones más comunes en las que se produce el RangeError son:

- Intentar crear un array de longitudes ilegales mediante el constructor Array.
- Pasar valores erróneos a métodos numéricos como toExponential(), toPrecision(), toFixed(), etc.
- Pasando valores ilegales a funciones de cadena como normalize().

ReferenceError

Un ReferenceError se produce cuando algo está mal con la referencia de una variable en tu código. Puede que hayas olvidado definir un valor para la variable antes de usarla, o puede que estés intentando utilizar una variable inaccesible en tu código. En cualquier caso, revisar el seguimiento de la pila proporciona amplia información para encontrar y arreglar la referencia de la variable que está en mal estado.

Algunas de las razones más comunes por las que se producen errores de referencia son:

- Escribir un error en el nombre de una variable.
- Intentar acceder a variables de ámbito de bloque fuera de su ámbito.
- Hacer referencia a una variable global de una biblioteca externa (como $ de jQuery) antes de que se cargue.

Error de Sintaxis

Estos errores son uno de los más sencillos de solucionar, ya que indican un error en la sintaxis del código. Dado que JavaScript es un lenguaje de

scripting que se interpreta y no se compila, se lanzan cuando la aplicación ejecuta el script que contiene el error. En el caso de los lenguajes compilados, estos errores se identifican durante la compilación. Por lo tanto, los binarios de la aplicación no se crean hasta que se corrigen.

Algunas de las razones más comunes por las que pueden producirse SyntaxErrors son:

- Faltan comillas
- Falta de paréntesis de cierre
- Alineación incorrecta de las llaves u otros caracteres

Es una buena práctica utilizar una herramienta de linting en tu IDE para identificar estos errores antes de que lleguen al navegador.

TypeError

TypeError es uno de los errores más comunes en las aplicaciones de JavaScript. Este error se produce cuando algún valor no resulta ser de un determinado tipo esperado. Algunos de los casos más comunes en los que se produce son:

- Invocar objetos que no son métodos.
- Intentar acceder a propiedades de objetos nulos o indefinidos
- Tratar una cadena como un número o viceversa

Hay muchas más posibilidades en las que puede producirse un TypeError. Más adelante veremos algunos casos famosos y aprenderemos a solucionarlos.

InternalError

El tipo InternalError se utiliza cuando se produce una excepción en el motor de ejecución de JavaScript. Puede indicar o no un problema con tu código.

La mayoría de las veces, el InternalError solo se produce en dos casos:

- Cuando un parche o una actualización del entorno de ejecución de JavaScript conlleva un error que lanza excepciones (esto ocurre muy raramente).
- Cuando tu código contenga entidades demasiado grandes para el motor de JavaScript (por ejemplo, demasiados casos de conmutación, inicializadores de matrices demasiado grandes, demasiada recursión).

El enfoque más adecuado para resolver este error es identificar la causa a través del mensaje de error y reestructurar la lógica de tu aplicación, si es posible, para eliminar el repentino aumento de la carga de trabajo en el motor de JavaScript.

URIError

El URIError se produce cuando se utiliza de forma ilegal una función global de manejo de URI como decodeURIComponent. Suele indicar que el parámetro pasado a la llamada del método no se ajustaba a los estándares de URI y, por tanto, no fue analizado por el método correctamente.

Diagnosticar estos errores suele ser fácil, ya que solo tienes que examinar los argumentos para ver si están mal formados.

EvalError

Un EvalError se produce cuando se produce un error en una llamada a la función eval(). La función eval() se usa para ejecutar código JavaScript almacenado en cadenas. Sin embargo, como el uso de la función eval() está muy desaconsejado por cuestiones de seguridad y las especificaciones actuales de ECMAScript ya no lanzan la clase EvalError, este tipo de error existe simplemente para mantener la compatibilidad con el código JavaScript antiguo.

Si estás trabajando en una versión antigua de JavaScript, es posible que te encuentres con este error. En cualquier caso, lo mejor es investigar el código ejecutado en la llamada a la función eval() para ver si hay alguna excepción.

Crear Tipos de Error Personalizados

Aunque JavaScript ofrece una lista adecuada de clases de tipos de error para cubrir la mayoría de los escenarios, siempre puedes crear un nuevo tipo de error si la lista no satisface tus necesidades. La base de esta flexibilidad reside en el hecho de que JavaScript te permite lanzar literalmente cualquier cosa con el comando throw.

Así que, técnicamente, estas declaraciones son totalmente legales:

throw 8

throw "An error occurred"

Sin embargo, lanzar un tipo de dato primitivo no proporciona detalles sobre el error, como su tipo, nombre o el seguimiento de la pila que lo acompaña. Para solucionar esto y estandarizar el proceso de gestión de errores, se ha proporcionado la clase Error. También se desaconseja utilizar tipos de datos primitivos al lanzar excepciones.

Puedes extender la clase Error para crear tu clase de error personalizada. Aquí tienes un ejemplo básico de cómo puedes hacerlo:

```
class ValidationError extends Error {
    constructor(message) {
        super(message);
        this.name = "ValidationError";
    }
}
```

Y puedes utilizarlo de la siguiente manera:

```
throw ValidationError("Property not found: name")
Y puedes identificarlo utilizando la palabra clave instanceof:
try {
    validateForm() // code that throws a ValidationError
} catch (e) {
    if (e instanceof ValidationError)
    // do something
    else
    // do something else
}
```