JavaScript - Errors & Exceptions Handling

Error handling in JavaScript is a process to detect and handle errors that occurs during the execution of a program. Errors can be a syntax, runtime or logical errors. An error occurred during the execution of the program is called a runtime error or an exception.

In JavaScript, errors can occur due to programming mistakes, incorrect user input, etc. Errors can disrupt code execution and lead to bad user experience. Effective error & exception handling is required for building robust, reliable and user friendly applications in JavaScript.

What is an Error?

An error is an event that occurs during the execution of a program that prevents it from continuing normally. Errors can be caused by a variety of factors, such as **syntax** errors, **runtime** errors, and **logical** errors.

Syntax Errors

Syntax errors, also called **parsing errors**, occur at compile time in traditional programming languages and at interpret time in JavaScript.

For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
<script>
   window.print();
</script>
```

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected, and the rest of the code in other threads gets executed, assuming nothing in them depends on the code containing the error.

Runtime Errors (Exceptions)

Runtime errors, also called **exceptions**, occur during execution (after compilation/interpretation).

For example, the following line causes a runtime error because the syntax is correct here, but at runtime, it is trying to call a method that does not exist.

```
<script>
   window.printme();
</script>
```

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

There are many JavaScript runtime errors (exceptions), some are as follows −

- **ReferenceError** − Trying to access an undefined variable/ method.
- **TypeError** − Attempting an operation on incompatible data types.
- **RangeError** − A value exceeds the allowed range.

Logical Errors

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and do not get the expected result.

For example, when you divide any numeric value with 10, it returns undefined.

```
<script>
   let num = 10/0;
</script>
```

What is Error Handling?

Whenever any error occurs in the JavaScript code, the JavaScript engine stops the execution of the whole code. If you handle such errors in the proper way, you can skip the code with errors and continue to execute the other JavaScript code.

You can use the following mechanisms to handle the error.

- try...catch...finally statements
- throw statements
- the onerror() event handler property
- Custom Errors

The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try...catch...finally construct as well as the throw operator to handle exceptions.

You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors.

Here is the try...catch...finally block syntax –

```
<script>
   try {
      // Code to run
      [break;]
   }
   catch ( e ) {
      // Code to run if an exception occurs
      [break;]
   }
   [ finally {
      // Code that is always executed regardless of
      // an exception occurring
   }]
</script>
```

The try block must be followed by either exactly one catch block or one finally block (or one of both). When an exception occurs in the try block, the exception is placed in e and the catch block is executed. The optional finally block executes unconditionally after try/catch.

Example

Here is an example where we are trying to call a non-existing function which in turn is raising an exception.

Let us try to catch this exception using try...catch and display a user-friendly message. You can also suppress this message, if you want to hide this error from a user.

You can use finally block which will always execute unconditionally after the try/catch.

```
<html>
<head>
<script>
   try {
      var a = 100;
      alert(myFunc(a));
   }
   catch (e) {
      alert(e);
   }
   finally {
      alert("Finally block will always execute!" );
   }
</script>
</head>
<body>
   <p>Exception handling using try...catch...finally statements</p>
</body>
</html>
```

Output

```
Exception handling using try...catch...finaly statements
```

The throw Statement

You can use throw statement to raise your built-in exceptions or your customized exceptions. Later these exceptions can be captured and you can take an appropriate action.

Example

The following example demonstrates how to use a throw statement.

```
<html>
<head>
<script>
   function myFunc() {
      var a = 100;
      var b = 0;

      try {
         if ( b == 0 ) {
            throw( "Divide by zero error." );
         } else {
            var c = a / b;
         }
      }
      catch ( e ) {
         alert("Error: " + e );
      }
   }
</script>
</head>
<body>
   <p>Click the following to see the result:</p>
   <form>
      <input type = "button" value = "Click Me" onclick = "myFunc();" />
   </form>
</body>
</html>
```

Output

```
Click the following to see the result:
Click Me
```

You can raise an exception in one function using a string, integer, Boolean, or an object and then you can capture that exception either in the same function as we did above, or in another function using a try...catch block.

The onerror Event Handler Property

The onerror event handler was the first feature to facilitate error handling in JavaScript. The onerror is an event handler property of the 'window' object, which automatically triggers when any error occurs on any element of the web page. You can call the callback function when any error occurs to handle the error.

You can follow the syntax below to use the onerror event handler property.

```
window.onerror = errorhandler_func;
OR
<ele onerror="errorhandler_func()"> </ele>
```

In the above syntax, errorhandler_func() will be executed when any error will occur.

The **onerror** event handler provides three pieces of information to identify the exact nature of the error –

- **Error message** – The same message that the browser would display for the given error
- **URL** – The file in which the error occurred
- **Line number** – The line number in the given URL that caused the error

Example

In the code below, we added the onclick event on the <input> element, and we called the myFunc() function when users click the input element. The myFunc() function is not defined. So, it will throw an error.

We used the 'onerror' event handler to catch the error. In the callback function, we print the error message, file URL, and line number in the file where the error occurs.

```
<html>
<body>
   <p> Click the following button to see the result:</p>
   <form>
      <input type = "button" value = "Click Me" onclick = "myFunc();" />
   </form>
   <div id = "demo"> </div>
   <script>
      const output = document.getElementById("demo");
      window.onerror = function (msg, url, line) {
         output.innerHTML = "Error: " + msg + "<br>";
         output.innerHTML += "URL: " + url + "<br>";
         output.innerHTML += "Line: " + line + "<br>";
      }
   </script>
</body>
</html>
```

Output

```
Click the following button to see the result:
Click Me
Error: Uncaught ReferenceError: myFunc is not defined
URL: file:///C:/Users/Lenovo/Desktop/intex.html
Line: 5
```

You can use an onerror method, as shown below, to display an error message in case there is any problem in loading an image.

```
<img src="myimage.gif" onerror="alert('An error occurred loading the image.')" />
```

You can use onerror with many HTML tags to display appropriate messages in case of errors.

The JavaScript Error Class and Error Object

Whenever any error occurs in the code, JavaScript throws an instance (object) of the error class. The error object contains the information about the error.

However, Error() is a generic constructor for all types of errors, but different objects exist for different types of errors.

JavaScript Custom Errors

You can also throw an error with the custom message using the Error() constructor.

```
const customError = new Error(message);
customError.name = "CustomError";
```

Here, we have created the 'Error' class instance and passed the 'message' as an argument. Also, we have changed the value of the 'name' property. Similarly, you can change the value of the 'message' property if you don't want to pass it as an Error() constructor argument.

JavaScript Error Object Reference

JavaScript Error Types or Constructor

JavaScript contains the below types of errors. You can also use it as a constructor to create a new error of the specific type.

Error Type/Object	Description
Error	It is a generic constructor for the error. You can also create custom errors by extending the Error object.
SyntaxError	The instance of the SyntaxError is thrown when any error is in the syntax. For example, missing parenthesis, invalid JSON, etc.
ReferenceError	The reference error occurs when you try to access variables not defined in the current scope.
TypeError	When the types of the variables is not valid, JavaScript throws the type error.
RangeError	When numeric input is out of range, it throws the range error.
URIError	JavaScript throws the URIError when you pass invalid arguments to the decodeURI or encodeURI methods.
EvalError	Deprecated.
AggregateError	It is used to aggregate multiple error objects into a single error object, and it allows you to handle multiple error objects.

Error Object Properties

The Error object contains the two properties.

Property	Description
name	It is used to set or get an error name.
message	It is used to set or get an error message.

Non-Standard Properties and Methods

Here is the list of the non-standard properties and methods of the Error object. However, they are not supported by all browsers. So, you should avoid using them.

Property	Description
columnNumber	It is supported in the Firefox browser only.
description	It is supported in the Microsoft browser only.
displayName	It is supported in the Firefox browser only.
fileName	It is supported in the Firefox browser only.
lineNumber	It is supported in the Firefox browser only.
number	It is supported in the Microsoft browser only.
stack	It is supported in the Firefox browser only.
internalError()	It is supported in the Firefox browser only.
toSource()	It is a Non Standard method of the Error object.