Handling errors in JavaScript

There are 3 types of errors in the program: (a) syntax error (Syntax Error), (b) error while running the program (Runtime Error), and (c) error of logic of program structure (Logical Error) .

There are 3 types of errors in the program: (a) syntax error (Syntax Error), (b) error while running the program (Runtime Error), and (c) error of logic of the program structure (Logical Error) .

Syntax Error

Syntax Error, also called parsing error, occurs at compile time in traditional program languages ??and at interpreting time in JavaScript.

For example, the following line causes a syntax error because it lacks closed parentheses.

When a syntax error occurs in JavaScript, only the code that contains the same thread is affected and the rest of the code in the other thread is still executing when assuming it does not depend on the code containing the error.

Runtime Error

Runtime Error, also called Exceptions, occurs during execution (after compiling / interpreting).

For example, the following line creates a Runtime Error because here the syntax is correct, but while running, it tries to call a method that does not exist.

Runtime Error also affects the thread in which they occur, allowing another thread in JavaScript to continue executing normally.

Logical Error

Logical errors are a type of error that is hard to find traces. These errors are not the result of syntax or error while running. Instead, they happen when you create a logical error that controls your script and you don't get the expected results.

You cannot catch these errors, because it depends on the requirements and logic types you put into the program.

The try . catch . finally command

The latest version of JavaScript adds the ability to handle program run errors. JavaScript implements **try . catch . finally** as well as **throw** operators to handle Runtime Error.

You can **catch** the Runtime Error but you cannot **catch** the syntax errors (Syntax Error).

The **try . catch . finally** block syntax is as follows:

The **try** block must be followed by either a **catch** block or a **finally** block (or either). When a Runtime Error occurs in the **try** block, this error is placed in e and the **catch** block is executed. **Finally** block optionally executes unconditionally after **try / catch conditions** .

For example

Here is an example where we try to call a non-existent function that creates Runtime Error. We see how it executes without using **try . catch** :

```
Click the following to see the result:
 type = "button" value = "Click Me" onclick = " myFunc (); " />
```

Result

Run the above command to see the result

Now we try to catch this error by using **try . catch** and displaying a user-friendly message (user-fiendly). You can also dismiss this message, if you want to hide this error, do not show it to the user.

```
Click the following to see the result:
 type = "button" value = "Click Me" onclick = " myFunc (); " />
```

Result

Run the above command to see the result

You can use **finally** blocks which will always execute unconditionally after **try / catch** . Here is an example:

```
Click the following to see the result:
 type = "button" value = "Click Me" onclick = " myFunc (); " />
```

Result

Run the above command to see the result

Throw command

You can use the **throw** command to raise available Runtime Error errors or your custom errors. Then these errors can be captured and you can make a reasonable action.

For example

The following example illustrates how to use a **throw** command.

```
Click the following to see the result:
 type = "button" value = "Click Me" onclick = " myFunc (); " />
```

Result

Run the above command to see the result

You can raise an error in a function by using a string, integer, logic, or an object and then you can catch this error or in the same function as we did above, or follow another function. by using **try . catch** block.

Onerror method ()

The **onerror** method is the first feature to handle errors in JavaScript. The error event is triggered on the window object whenever an exception occurs on the page.

```
Click the following to see the result:
 type = "button" value = "Click Me" onclick = " myFunc (); " />
```

Result

Run the above command to see the result

The **onerror** error handling **method** provides 3-part information to determine the exact error.

Error message - The notification that the browser will display with a given error.

URL - File where the error occurred.

Line number - Which line in the given URL causes an error.

Here is an example of how to extract this information:

For example

```
Click the following to see the result:
 type = "button" value = "Click Me" onclick = " myFunc (); " />
```

Result

Run the above command to see the result

You can display the extracted information in any way you think it is good.

You can use an **onerror** method, as shown below, to display an error message in case of any problems while loading an image.

```
src = "myimage.gif" onerror = " alert ( 'An error occurred loading the image
```

You can use onerror with multiple HTML tags to display appropriate messages in case of an error.

According to Tutorialspoint

Previous post: Document Object Model (DOM) in JavaScript

Next article: Form Validation in JavaScript