Understanding Errors Lesson

Mohamed Emary

August 26, 2023

1 The anatomy of an error

An error is a type of **object** built into the JS language, consisting of a name/type and a message

When you type a JavaScript code that has a problem and it throws an error, the error message will contain the following information:

- **Name/Type**: The name of the error. For example `ReferenceError`.
- **Message**: A description of the error.
- **Stack**: A **stack trace** that helps you understand when the error was thrown in your application, and what functions were called that led up to the error.

When you click the link in the stack trace, it will take you to the line of code that caused the error in sources tab in the dev tools.

1.1 `ReferenceError`

A `ReferenceError` is thrown when one refers to a variable that is not declared and/or initialized within the current scope - or it has been spelled incorrectly!.

```
const a = "Hello";
const b = "World";

console.log(c);  // error
```

```
Uncaught ReferenceError: c is not defined                    script.js:4
    at script.js:4:13
```

Figure 1: `ReferenceError` Message

Here, the error originates from the fourth line of script.js, which is displayed as a link under the error message with the text at `script.js:4`. If you click this link, most browsers will navigate to the exact line of code and the rest of your script in the Sources tab of the Developer Tools.

Sometimes your browser's console will also display the column (or character) in the line at which the error is occurring. In our example, this would be at `script.js:4:13`.

Stack trace helps you understand when the error was thrown in your application, and what functions were called that led up to the error. So, for example, if we have the following code:

```
const a = 5;
const b = 10;

function add() {
  return c;
}

function print() {
  add();
}

print();
```

Our function `print()` should call on `add()`, which returns a variable named `c`, which currently has not been declared. The corresponding error is as follows:

```
Uncaught ReferenceError: c is not defined                    script.js:5
    at add (script.js:5)
    at print (script.js:9)
    at script.js:12
```

Figure 2: Stack Trace in `ReferenceError` Message

The stack trace tells us that:

1. `c is not defined` in scope of `add()`, which is declared on *line 5*
2. `add()` was called by `print()`, which was declared on *line 9*
3. `print()` itself was called on *line 12*.

Thus the stack trace lets you trace the evolution of an error back to its origin, which here is the declaration of `add()`.

1.2 SyntaxError

A syntax error occurs when the code you are trying to run is not written correctly, i.e., in accordance with the grammatical rules of JavaScript. For example this:

will throw the following error, because we forgot the parentheses for `console.log()`!

```
1 function helloWorld() {
2   console.log "Hello World!"
3 }
```

```
Uncaught SyntaxError: Invalid or unexpected token
```

Figure 3: `SyntaxError` Message

1.3 TypeError

These errors are thrown for a few different reasons:

- an operand or argument passed to a function is incompatible with the type expected by that operator or function;
- or when attempting to modify a value that cannot be changed;
- or when attempting to use a value in an inappropriate way.

A good note to keep in mind when faced with a `TypeError` is to consider the data type you are trying to run a method or operation against. You'll likely find that it is not what you think, or the operation or method is not compatible with that type.

For Example:

```
1 const str1 = "Hello";
2 const str2 = "World!";
3 const message = str1.push(str2);
```

```
Uncaught TypeError: str1.push is not a function          script.js:3
    at script.js:3:22
```

Figure 4: `TypeError` Message

1.4 Tips for resolving errors

At this point, you might be wondering how we can resolve these errors.

1. Read the error carefully and try to understand it on your own.
2. use `console.log()`. There are other useful methods such as `console.table()`, `console.trace()`, and more! You can find additional methods here.
3. Use the debugger! This tutorial dives into the Chrome Debugger.
4. Google the error!