JavaScript Throw Exception

Summary: in this tutorial, you'll learn how to use the JavaScript `throw` statement to throw an exception.

Introduction to the JavaScript throw statement

The `throw` statement allows you to throw an exception. Here's the syntax of the `throw` statement:

```
throw expression;
```

In this syntax, the `expression` specifies the value of the exception. Typically, you'll use a new instance of the `Error` class or its subclasses.

When encountering the `throw` statement, the JavaScript engine stops executing and passes the control to the first `catch` block in the call stack. If no `catch` block exists, the JavaScript engine terminates the script.

JavaScript throw exception examples

Let's take some examples of using the `throw` statement.

1) Using the JavaScript throw statement to throw an exception

The following example uses the `throw` statement to throw an exception in a function:

```
function add(x, y) {
  if (typeof x !== 'number') {
    throw 'The first argument must be a number';
  }
  if (typeof y !== 'number') {
    throw 'The second argument must be a number';
```

```
  }

  return x + y;
}

const result = add('a', 10);
console.log(result);
```

How it works.

First, define the `add()` function that accepts two arguments and returns the sum of them. The `add()` function uses the `typeof` operator to check the type of each argument and throws an exception if the type is not a number.

Second, call the `add()` function and pass a string and a number into it.

Third, show the result to the console.

The script causes an error because the first argument (`"a"`) is not a number:

```
Uncaught The first argument must be a number
```

To handle the exception, you can use the `try...catch` statement. For example:

```
function add(x, y) {
  if (typeof x !== 'number') {
    throw 'The first argument must be a number';
  }
  if (typeof y !== 'number') {
    throw 'The second argument must be a number';
  }

  return x + y;
}

try {
  const result = add('a', 10);
  console.log(result);
```

```
} catch (e) {
  console.log(e);
}
```

Output:

```
The first argument must be a number
```

In this example, we place the call to the `add()` function in a `try` block. Because the `expression` in the `throw` statement is a string, the exception in the `catch` block is a string as shown in the output.

2) Using JavaScript throw statement to throw an instance of the Error class

In the following example, we throw an instance of the `Error` class rather than a string in the `add()` function;

```
function add(x, y) {
  if (typeof x !== 'number') {
    throw new Error('The first argument must be a number');
  }
  if (typeof y !== 'number') {
    throw new Error('The second argument must be a number');
  }

  return x + y;
}

try {
  const result = add('a', 10);
  console.log(result);
} catch (e) {
  console.log(e.name, ':', e.message);
}
```

Output:

```
Error : The first argument must be a number
```

As shown in the output, the exception object in the `catch` block has the `name` as `Error` and the `message` as the one that we pass to the `Error()` constructor.

3) Using JavaScript throw statement to throw a user-defined exception

Sometimes, you want to throw a custom error rather than the built-in `Error`. To do that, you can define a custom error class that extends the `Error` class and throw a new instance of that class. For example:

First, define the `NumberError` that extends the `Error` class:

```
class NumberError extends Error {
  constructor(value) {
    super(`"${value}" is not a valid number`);
    this.name = 'InvalidNumber';
  }
}
```

The `constructor()` of the `NumberError` class accepts a value that you'll pass into it when creating a new instance of the class.

In the `constructor()` of the `NumberError` class, we call the constructor of the `Error` class via the `super` and pass a string to it. Also, we override the name of the error to the literal string `NumberError`. If we don't do this, the `name` of the `NumberError` will be `Error`.

Second, use the `NumberError` class in the `add()` function:

```
function add(x, y) {
  if (typeof x !== 'number') {
    throw new NumberError(x);
  }
  if (typeof y !== 'number') {
    throw new NumberError(y);
  }
```

```
  return x + y;
}
```

In the `add()` function, we throw an instance of the `NumberError` class if the argument is not a valid number.

Third, catch the exception thrown by the `add()` function:

```
try {
  const result = add('a', 10);
  console.log(result);
} catch (e) {
  console.log(e.name, ':', e.message);
}
```

Output:

```
InvalidNumber : "a" is not a valid number
```

In this example, the exception name is `NumberError` and the message is the one that we pass into the `super()` in the `constructor()` of the `NumberError` class.

Summary

- Use the JavaScript `throw` statement to throw a user-defined exception.