JavaScript
TUTORIAL

JavaScript try...catch

Summary: in this tutorial, you will learn how to use the JavaScript `try...catch` statement to handle exceptions.

Introduction to JavaScript try...catch statement

The following example attempts to call the `add()` function that doesn't exist:

```
let result = add(10, 20);
console.log(result);

console.log('Bye');
```

And the JavaScript engine issues the following error:

```
Uncaught TypeError: add is not a function
```

The error message states that the `add` is not a function and the error type is `TypeError`.

When the JavaScript engine encounters an error, it issues that error and immediately terminates the execution of the entire script. In the above example, the code execution stops at the first line.

Sometimes, you want to handle the error and continue the execution. To do that, you use the `try...catch` statement with the following syntax:

```
try {
  // code may cause error
} catch(error){
  // code to handle error
}
```

In this syntax:

- First, place the code that may cause an error in the `try` block.
- Second, implement the logic to handle the error in the `catch` block.

If an error occurs in the `try` block, the JavaScript engine immediately executes the code in the `catch` block. Also, the JavaScript engine provides you with an error object that contains detailed information about the error.

Basically, the error object has at least two properties:

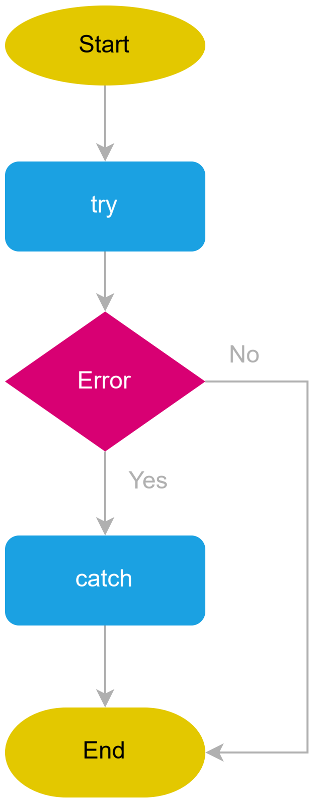
- `name` specifies the error name.
- `message` explains the error in detail.

If no error occurs in the `try` block, the JavaScript engine ignores the `catch` block.

> Note that web browsers may add more properties to the `error` object. For example, Firefox adds `filename`, `lineNumber`, and `stack` properties to the `error` object.

It's a good practice to place only the code that may cause an exception in the `try` block.

The following flowchart illustrates how the `try...catch` statement works:



JavaScript try...catch statement examples

The following example uses the `try...catch` statement to handle the error:

```
try {
  let result = add(10, 20);
  console.log(result);
} catch (e) {
  console.log({ name: e.name, message: e.message });
}
console.log('Bye');
```

Output

```
{name: 'TypeError', message: 'add is not a function'}
Bye
```

In this example, we call the `add()` function and assign the return value to the `result` variable. Because the `add()` function doesn't exist, the JavaScript engine skips the statement that outputs the result to the console:

```
console.log(result);
```

And it immediately executes the statement in the `catch` block that outputs the error name and message:

```
console.log({ name: e.name, message: e.message });
```

Since we already handled the error, the JavaScript engine continues to execute the last statement:

```
console.log('Bye');
```

Ignoring the catch block

The following example defines the `add()` function that returns the sum of two arguments:

```
const add = (x, y) => x + y;

try {
  let result = add(10, 20);
  console.log(result);
} catch (e) {
  console.log({ name: e.name, message: e.message });
}
console.log('Bye');
```

Output:

```
30
Bye
```

In this example, no error occurs because the `add()` function exists. Therefore, the JavaScript engine skips the `catch` block.

The exception identifier

When an exception occurs in the try block, the exception variable (e) in the catch block stores the exception object.

If you don’t want to use the exception variable, you can omit it like this:

```
try {
  //...
} catch {
  //...
}
```

For example, the following uses the try...catch statement without the exception variable:

```
const isValidJSON = (str) => {
  try {
    JSON.parse(str);
    return true;
  } catch {
    return false;
  }
};

let valid = isValidJSON(`{"language":"JavaScript"}`);
console.log(valid);
```

How it works.

First, define the `isValidJSON()` function that accepts a string and returns `true` if that string is a valid JSON or `false` otherwise.

To validate JSON, the `isValidJSON()` function uses the `JSON.parse()` method and `try...catch` statement.

The JSON.parse() method parses a JSON string and returns an object. If the input string is not valid JSON, the `JSON.parse()` throws an exception.

If no exception occurs, the function returns `true` in the `try` block. Otherwise, it returns `false` in the `catch` block.

Second, call the `isValidJSON()` function and pass a JSON string into it:

```
let valid = isValidJSON(`{"language":"JavaScript"}`);
```

Since the input string is valid JSON format, the function returns `true`.

Third, output the `result` to the console:

```
console.log(valid);
```

Summary

- Use the `try...catch` statement to handle exceptions in JavaScript.
- Place only the code that may cause an exception in the `try` block.