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');

Code language: JavaScript (javascript)

And the JavaScript engine issues the following error:

Uncaught TypeError: add is not a function

Code language: JavaScript (javascript)

The error message states that the add is not a [function](https://www.javascripttutorial.net/javascript-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*

}

Code language: JavaScript (javascript)

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');

Code language: JavaScript (javascript)

Output

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

Bye

Code language: CSS (css)

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);

Code language: JavaScript (javascript)

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 });

Code language: CSS (css)

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

console.log('Bye');

Code language: JavaScript (javascript)

Ingoring 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');

Code language: JavaScript (javascript)

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 store the exception object.

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

try {

*//...*

} catch {

*//...*

}

Code language: JavaScript (javascript)

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);

Code language: JavaScript (javascript)

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"}`);

Code language: JavaScript (javascript)

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

Third, output the result to the console:

console.log(valid);

Code language: JavaScript (javascript)

Summary

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

JavaScript try…catch…finally

**Summary**: in this tutorial, you’ll learn how to use the JavaScript try...catch...finally statement to catch exceptions and execute a block whether the exceptions occur or not

Introduction to the JavaScript try…catch…finally statement

The [try...catch](https://www.javascripttutorial.net/javascript-try-catch/) statement allows you to catch exceptions and handle them gracefully. Sometimes, you want to execute a block whether exceptions occur or not. In this case, you can use the try...catch...finally statement with the following syntax:

try {

*// code may cause exceptions*

} catch (error) {

*// code to handle exceptions*

} finally {

*// code to execute whether exceptions occur or not*

}

Code language: JavaScript (javascript)

In this syntax, the finally block always executes after the try and catch blocks complete and whether exceptions occur or not.

The following flowchart illustrates how the try...catch...finally works:

JavaScript try…catch…finally statement example

The following example illustrates how to use the try...catch...finally statement:

let result = 0;

try {

result = add(10, 20);

} catch (e) {

console.log(e.message);

} finally {

console.log({ result });

}

Code language: JavaScript (javascript)

Output:

add is not defined

{ result: 0 }

Code language: CSS (css)

How it works.

First, declare the result variable and initialize its value with 0.

let result = 0;

Code language: JavaScript (javascript)

Second, call the add() function and assign the return value to the result variable in the try block. Because the add() function does not exist, the JavaScript engine raises an exception.

Because of the exception, the statement in the catch block executes to show the error message.

Third, output the result to the console in the try block.

In the following example, we define the add() function and call it in the try block:

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

let result = 0;

try {

result = add(10, 20);

} catch (e) {

console.log(e.message);

} finally {

console.log({ result });

}

Code language: JavaScript (javascript)

Output:

{ result: 30 }

Code language: CSS (css)

Because the add() function exists, no exception occurs in the try block. Therefore, the finally block outputs the value of the result variable, which is the sum of 10 and 20.

In both examples, the finally block always runs.

try…catch…finally and return

The finally block always executes whether exceptions occur or not. Also, you can do nothing to prevent it from executing including using a return statement. For example:

function fn() {

try {

return 1;

} catch {

return 2;

} finally {

return 3;

}

}

console.log(fn());

Code language: JavaScript (javascript)

Output:

3

In this example, the return statement in the try block returns 1. Hence, the fn() function should have returned 1. However, it is not the case.

Because the finally block always executes, the return statement in the finally block returns 3. Therefore, the fn() function returns 3.

In other words, the return statements in the try and catch blocks are ignored in the try...catch...finally statement.

Summary

* Use the finally clause in the try...catch...finally statement to execute a block whether exceptions occur or not.
* The try...catch...finally statement ignores the return statement in the try and catch blocks because the finally block always executes.

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;

Code language: JavaScript (javascript)

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](https://www.javascripttutorial.net/javascript-try-catch/) block in the [call stack](https://www.javascripttutorial.net/javascript-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](https://www.javascripttutorial.net/javascript-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);

Code language: JavaScript (javascript)

How it works.

First, define the add() function that accepts two arguments and returns the sum of them. The add() function uses the [typeof](https://www.javascripttutorial.net/javascript-typeof/) operator to check the type of each argument and throws an exception if the type is not 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](https://www.javascripttutorial.net/javascript-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);

}

Code language: JavaScript (javascript)

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);

}

Code language: JavaScript (javascript)

Output:

Error : The first argument must be a number

Code language: JavaScript (javascript)

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';

}

}

Code language: JavaScript (javascript)

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 NunberError 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;

}

Code language: JavaScript (javascript)

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);

}

Code language: JavaScript (javascript)

Output:

NumberError : "a" is not a valid number

Code language: JavaScript (javascript)

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-define exception.

Optional catch Binding

**Summary**: in this tutorial, you will learn how to use the optional catch binding in the try...catch statement.

Introduction to the optional catch binding

The [try...catch](https://www.javascripttutorial.net/javascript-try-catch/) statement is used to handle any errors that may occur. Generally, you place the code that may cause an error in the try block and the code that handles the error in the catch block, like this:

try {

*// code that may cause an error*

} catch (error) {

*// code that handles the error*

}

Code language: JavaScript (javascript)

In the catch block, you can access the Error object that contains detailed information on the error.

In practice, you may want to use the try...catch statement to check if a feature is implemented in the web browser. If it isn’t, you want to fall back to a less desirable feature with broader support, for example:

try {

*// check if a feature is implemented*

} catch (error) {

*// fall back to a less desirable feature*

}

Code language: JavaScript (javascript)

In this case, the error object is declared but never used.

ES2019 introduced the optional catch binding that allows you to omit the catch binding and its surrounding parentheses, like this:

try {

} catch {

}

Code language: JavaScript (javascript)

Summary

* Since ES2019, you can omit the catch binding in the try...catch statement.