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

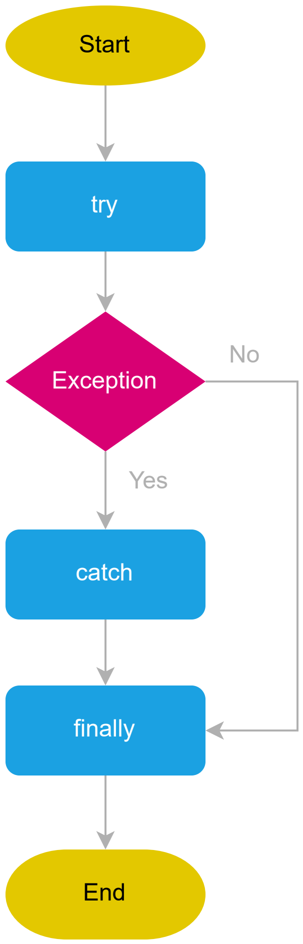
Introduction to the JavaScript try...catch...finally statement

The `try...catch` 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
}
```

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 });
}
```

Output:

```
add is not defined
{ result: 0 }
```

How it works.

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

```
let result = 0;
```

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 });
}
```

Output:

```
{ result: 30 }
```

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());
```

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.