JavaScript Nullish Coalescing Operator

Summary: in this tutorial, you'll learn about the JavaScript nullish coalescing operator (`??`) that accepts two values and returns the second value if the first one is `null` or `undefined` .

Introduction to the JavaScript nullish coalescing operator

ES2020 introduced the nullish coalescing operator denoted by the double question marks (`??`). The nullish coalescing operator is a logical operator that accepts two values:

```
value1 ?? value2
```

The nullish coalescing operator returns the second value (`value2`) if the first value (`value2`) is `null` or `undefined` . Technically, the nullish coalescing operator is equivalent to the following block:

```
const result = value1;
if(result === null || result === undefined) {
   result = value2;
}
```

> A nullish value is a value that is either `null` or `undefined` .

The following example uses the nullish coalescing operator (`??`) to return the string `'John'` because the first value is `null` :

```
const name = null ?? 'John';
console.log(name); // 'John'
```

And this example returns `28` because the first value is `undefined`:

```
const age = undefined ?? 28;
console.log(age);
```

Why nullish coalescing operator

When assigning a default value to a variable, you often use the logical OR operator (`||`). For example:

```
let count;
let result = count || 1 ;
console.log(result); // 1
```

In this example, the `count` variable is `undefined`, it is coerced to `false`. Therefore, the `result` is `1`.

However, the logical `OR` operator (`||`) sometimes is confusing if you consider `0` or empty strings `''` as a valid value like this:

```
let count = 0;
let result = count || 1;
```

The result is 1, not 0, which you may not expect.

The nullish coalescing operator helps you to avoid this pitfall. It only returns the second value when the first one is either `null` or `undefined`.

The nullish coalescing operator is short-circuited

Similar to the logical OR and AND operators, the nullish coalescing operator does not evaluate the second value if the first operand is neither `undefined` nor `null`.

For example:

```
let result = 1 ?? console.log('Hi');
```

In this example, the operator `??` does not evaluate the second expression that displays the “Hi” to the console because the first value is `1`, which is not `null` or `undefined`.

The following example evaluates the second expression because the first one is `undefined`:

```
let result = undefined ?? console.log('Hi');
```

Output:

```
'Hi'
```

Chaining with the AND or OR operator

A `SyntaxError` will occur if you combine the logical AND or OR operator directly with the nullish coalescing operator like this:

```
const result = null || undefined ?? 'OK'; // SyntaxError
```

However, you can avoid this error by wrapping the expression on the left of the `??` operator in parentheses to explicitly specify the operator precedences:

```
const result = (null || undefined) ?? 'OK';
console.log(result); // 'OK'
```

Summary

- The nullish coalescing operator (`??`) is a logical operator that accepts two values and returns the second value if the first one is `null` or `undefined`.
- The nullish coalescing operator is short-circuited and cannot directly combine with the logical AND or OR operator.