JavaScript hoisting behavior

Learn what and why JavaScript has a hoisting mechanism.

Posted on February 02, 2021

Hoisting is a JavaScript internal behavior where variable, function, and class declarations are saved to the memory during the *compile phase*, right before your code is *executed*.

Let's see a technical example of hoisting in action. Suppose you have the following JavaScript code:

```
console.log(`The value of num is: ${num}`);
var num = 8;
```

At first, you may think that JavaScript should throw an error because the code is trying to log the value of `num` before the variable is declared and initialized. But here's what you will get when you run the code above:

```
The value of num is: undefined
```

Because hoisting puts the variable declaration `num` into memory before execution, your code will be executed like this by JavaScript:

```
var num;
console.log(`The value of num is: ${num}`); // undefined
num = 8;
```

Hoisting will automatically assign `undefined` as the initial value for all variables declared using the `var` keyword. If you declare the variable using `let` or `const` keyword, JavaScript will still throw an error because the variable is not initialized.

The following code will throw a `ReferenceError`:

```
console.log(`The value of num is: ${num}`);
let num = 8; // ReferenceError: num is not defined
```

This is hoisting in a nutshell. It puts all declarations: variables, functions, and classes into memory before code execution.

Is hoisting good or bad?

JavaScript developers tend to write code in a way that avoids hoisting. This is because hoisting is bad for variables,

but it's actually good for functions and classes.

As you've seen in the example above, variables declared with `var` keyword can easily cause bugs by returning `undefined` value to the caller instead of returning the value that you initialized. It's not much of a problem when you declare variables with `let` or `const` keyword.

But for functions and classes, you may want to call them before declarations because it will give you a more readable code.

Here's an example of a function call before declaration:

```
greetings("Jack", "Charlie");

function greetings(sender, addressee) {
  console.log(`Hello, my name is ${sender}`);
  console.log(`Nice to meet you, ${addressee}!`);
}
```

Even though it's subjective, most people would consider the above code is more readable than the following:

```
function greetings(sender, addressee) {
  console.log(`Hello, my name is ${sender}`);
  console.log(`Nice to meet you, ${addressee}!`);
}

greetings("Jack", "Charlie");
```

There's no right or wrong answer when it comes to hoisting because it's not something you can do about. This is what the JavaScript compiler does. That being said, you should keep in mind two rules about hoisting when you write your code. Here they are:

- JavaScript only hoists declarations
- Function and class expressions are **not** hoisted

By now you should be familiar with the first rule, so let me explain the second.

JavaScript allows you to create a function using *function declaration*, which looks like the following:

```
function greetings() {
  console.log("Hello World!");
}
```

And an alternative syntax called function expression as you can see below:

```
let greetings = function () {
  console.log("Hello World!");
};
```

Both are valid functions, but since function expressions are not hoisted, calling on function expression before it has been defined will cause an error:

```
greetings();

let greetings = function () {
  console.log("Hello World!");
};
```

Output:

```
Error: Cannot access 'greetings' before initialization
```

The same goes for classes:

```
let tesla = new Car("Tesla Model 3");

let Car = class {
  constructor(name) {
    this.name = name;
  }
};
```

Output:

```
Error: Cannot access 'Car' before initialization
```

Still, you don't need to worry about the second rule because the most common way to write functions and classes is to use declarations instead of expressions. I hope this article has helped you to understand what hoisting does and how it affects the way you can code JavaScript programs.

Related articles:

- Understanding var, let and const keywords and using them in the right way
- JavaScript global variable
- Find Fibonacci sequence number using recursion in JavaScript
- Understanding JavaScript hasOwnProperty method
- JavaScript methods to remove an element from an array