JavaScript Array.of()

Summary: in this tutorial, you will learn how to improve array construction using the JavaScript `Array.of()` method in ES6.

Introduction to the JavaScript Array.of() method

The `Array.of()` static method allows you to create a new Array from a variable number of arguments.

Here's the syntax of the `Array.of()` method:

```
Array.of(element1, element2, ...)
```

In this syntax:

- `element1` , `element2` , ... are the elements of the new array.

The `Array.of()` method returns a new instance of the `Array` that includes the `element1` , `element2` , ...

JavaScript Array.of() examples

Let's take some examples of using the `Array.of()` method.

Basic Array.of() method example

The following example uses the `Array.of()` method to create a new array that contains one number:

```
let numbers = Array.of(10);
```

```
console.log({ numbers });
console.log({ length: numbers.length });
```

Output:

```
{ numbers: [ 10 ] }
{ length: 1 }
```

In this example, the `Array.of()` method returns a new array that contains a single number 10.

The following example uses the `Array.of()` method to create an array from the three letters `'A'`, `'B'`, and `'C'`:

```
let chars = Array.of('A', 'B', 'C');

console.log({ chars });
console.log({ length: chars.length });
```

Output:

```
{ chars: [ 'A', 'B', 'C' ] }
{ length: 3 }
```

Practical Array.of() method example

When selecting DOM elements, you often receive a `NodeList` or `HTMLCollection` objects. These objects are array-like objects, not `Array` objects.

To manipulate DOM elements more effectively, you can use the `Array.of()` with the spread operator `(...)` to create a new array of DOM elements.

For example:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>JavaScript Array.of() method</title>
  </head>
  <body>
    <ul>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
    </ul>
    <script>
      const listItems = document.getElementsByTagName('li');
      Array.of(...listItems).forEach((item) => (item.style.color = 'red'));
    </script>
  </body>
</html>
```

How it works.

The HTML document has a `ul` element that contains three `li` elements.

First, select the `li` elements using the `getElementsByTagName()` method:

```
const listItems = document.getElementsByTagName('li');
```

The return value (`listItems`) is a `HTMLCollection` .

Second, create a new `Array` from the elements of the `HTMLCollection` object, iterate over each element, and change its color to red:

```
Array.of(...listItems).forEach((item) => (item.style.color = 'red'));
```

Array.of() method vs. Array constructor

When creating a new array using the `Array` constructor with a number, it'll create an array with the corresponding length. For example:

```
const arr = new Array(3);

console.log({ arr });
console.log({ length: arr.length });
```

Output:

```
{ arr: [ <3 empty items> ] }
{ length: 3 }
```

In this example, the `Array` constructor creates a new array with the length 3 and includes 3 empty items.

But when you pass a value that is not a number to the `Array` constructor, it'll create a new array with the length 1 and include that value as an element:

```
const arr = new Array('3');

console.log({ arr });
console.log({ length: arr.length });
```

Output:

```
{ arr: [ '3' ] }
{ length: 1 }
```

This behavior can be confusing and error-prone because you might not know the type of value that you're passing to the `Array` constructor.

ES6 introduced the `Array.of()` static method to solve this problem.

The `Array.of()` method is similar to the `Array` constructor, but it does not treat a single numeric value special. In other words, the `Array.of()` method always creates an array containing the values you pass to it regardless of the types or the number of arguments.

Summary

- Use the JavaScript `Array.of()` method to create a new instance of an Array from a number of values.