How to clone an array in ES6?

In ES5, we used the concat method to make a copy of the array. Also, some developers used the slice() method by passing 0 as a parameter to slice all elements of the referenced array and create a new array.

Example

Users can follow the example below to clone the array using the slice() method. We have created array1, which contains the values of different data types. After that, we used the slice() method to make a copy of array1 and store it in the 'clone' variable.

Edit & Run

```
<html>
<body>
   <h2>Using the <i>slice()</i> method to clone the array in JavaScript</h2>
   <div id = "output"> </div>
   <script>
      let output = document.getElementById('output');
      let array1 = [10, "hello", 30, true];
      output.innerHTML += "The original array: " + array1 + " <br>";
      let clone = array1.slice(0);
      output.innerHTML += "The cloned array: " + clone + " <br>";
   </script>
</body>
</html>
```

Users have learned how we were cloning the array in ES5.

Also, users can think about copying the array as they copy normal variables like string, number, and boolean using the assignment operator.

Users can face a problem while copying the array using the assignment operator. Let's understand it via the example below.

Copying Array using Assignment Operator

In the example below, the strings array contains various strings. We have assigned the strings array to the strings2 array. After that, we pushed the new string value to the strings2 array.

Example

Edit & Run

```
<html>
<body>
```

```
    <h2>Using the <i>assignment</i> operator to clone the array in JavaScript</h2>
    <div id = "output"> </div>
    <script>
       let output = document.getElementById('output');
       let array1 = ["Hi", " users", "Welcome"];
       let array2 = array1;
       array2.push("New value");
       output.innerHTML += "The value of array2: " + array2 + " <br>";
       output.innerHTML += "The value of array1: " + array1 + " <br>";
    </script>
 </body>
 </html>
```

In the above output, users can observe that when we have pushed the string value to the strings2 array, it also pushed to the strings array. Why did this happen?

Here, the concept of mutable and immutable objects comes into play.

Mutable VS immutable objects

In JavaScript, Array and objects are mutable means we can change their value once we initialize them after creating them. So, there is no existence of the strings2 array in the above example. When we assign the strings array to the strings2 array, it generates the reference to the strings array. So, whenever we change the strings2 array, it will also change the strings array.

So, we need to create an actual copy of the array without referencing the other array

Now, let's learn to clone the array in ES6.

Using the spread operator (…) to clone an array in ES6

The syntax of the spread operator is three dots (...). We can use it to spread the iterable object like an array. The spread operator creates a new copy of the array or object.

Syntax

Users can follow the syntax below to copy the array using the spread operator.

```
let booleansCopy = [...booleans];
```

Example

In the example below, we have created the boolean array, which contains the different boolean values. After that, we created the copy of the boolean array using the spread operator and assigned that copy to the booleanCopy variable.

In the output, users can observe that the booleanCopy array contains the same values as the boolean array contains.

```
<html>
<body>
   <h2>Using the <i> spread operator </i> to clone the array in JavaScript</h2>
   <div id = "output"> </div>
   <script>
      let output = document.getElementById('output');
      let booleans = [true, false, false, true, true];
      let booleansCopy = [...booleans];
      output.innerHTML += "The values of the booleansCopy array: " + booleansCopy + "
<br>";
   </script>
</body>
</html>
```

Edit & Run

Example

In the example below, the sizes array contains different number values. We have created the copy of the sizes array using the spread operator and stored that array in the sizesClone variable using the assignment operator.

After that, we pushed 60 to the sizesClone array.

```
<html>
<body>
   <h2>Using the <i>spread</i> operator to clone the array in JavaScript</h2>
   <div id = "output"> </div>
   <script>
      let output = document.getElementById('output');
      let sizes = [34, 43, 45, 47, 49, 50];
      output.innerHTML += "-------Before Push-------" + "<br>";
      output.innerHTML += "The values of the sizes array: " + sizes + "<br>";
      let sizesClone = [...sizes];
      output.innerHTML += "The values of the sizesClone array: " + sizesClone + "<br>";
      sizesClone.push(60);
      output.innerHTML += "------After Push------" + "<br>";
      output.innerHTML += "The values of the sizes array: " + sizes + "<br>";
      output.innerHTML += "The values of the sizesClone array:" + sizesClone + "<br>";
   </script>
</body>
</html>
```

Edit & Run

In the above output, users can observe that 60 reflects in the *sizesClone array* but not in the sizes array because we have created the actual copy of the array rather than referencing the array like in

example 2.

Now, users clearly understand why not to use the assignment operator in ES6 and why to use the *spread operator* to clone the array.