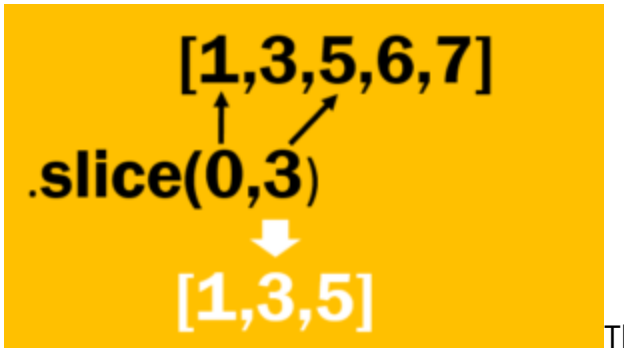
3 Pragmatic Uses of JavaScript Array slice() Method



The `Array.prototype` object provides the `slice()` method that allows you to extract subset elements of an array and add them to the new array. In this tutorial, we will show you the practical uses of the JavaScript array `slice()` method.

Introduction to JavaScript Array slice() method

The `slice()` method accepts two optional parameters as follows:

```
slice(start, stop);
```

Both `start` and `stop` parameters are optional.

The `start` parameter determines the zero-based index at which to start extraction. If the `start` is `undefined`, `slice()` begins at `0`.

The `stop` parameter, as its name implies, is a zero-based index at which to end extraction. The `slice()` method extracts up to `stop-1`. It means that the `slice()` method doesn't include the element at the `stop` position in the new array. If you omit the `stop` parameter, the `slice()` method will use the length of the array for the `stop` parameter.

The `slice()` returns a new array that contains the elements of the original array. It's important to keep in mind that the `slice()` method performs the shallow copy of elements to the new array

only. In addition, it doesn't change the source array.

Clone an array

The `slice()` is used to clone an array as shown in the following example:

```
var numbers = [1,2,3,4,5];
var newNumbers = numbers.slice();
```

In this example, the `newNumbers` array contains all the elements of the `numbers` array.

Copy a portion of an array

The typical use of the `slice()` method is to copy a portion of an array without modifying the source array. Here is an example:

```
var colors = ['red','green','blue','yellow','purple'];
var rgb = colors.slice(0,3);
console.log(rgb); // ["red", "green", "blue"]
```

The `rgb` array contains the first three elements of the `colors` array. The source array `colors` remains intact.

Convert array-like objects into arrays

The `slice()` method is used to convert an array-like object into an array. For example:

```
function toArray() {
  return Array.prototype.slice.call(arguments);
}

var classification = toArray('A','B','C');

console.log(classification); // ["A", "B", "C"]
```

In this example, the `arguments` of the `toArray()` function is an array-like object. Inside the `toArray()` function, we called the `slice()` method to convert the arguments object into an array.

Every argument we pass to the `toArray()` function will be the elements of the new array.

Another typical example that you often see is converting a `NodeList` into an array as follows:

```
var p = document.querySelectorAll('p');
var list = Array.prototype.slice.call(p);
```

In this example, first, we used the `document.querySelectorAll()` to get all `p` nodes of the HTML document. The result of this method is a `NodeList` object, which is an array-like object. Then, we called the `slice()` method to convert the `NodeList` object into an array.

Sometimes, you see the following syntax:

```
var list = [].slice.call(document.querySelectorAll('p'));
```

In this example, we instantiated an empty array `[]` and indirectly accessed the `slice()` method of the `Array.prototype` method through the empty array. The effect is the same as the one that uses the `Array.prototype` directly.

In this tutorial, you have learned how to use the JavaScript array `slice()` method to copy an entire or a subset of the array and convert an array-like object into an array.