Shallow and Deep Copy in JavaScript: A Guide with Lodash, structuredClone, and JSON Methods

Navneet Singh · Follow
3 min read · Oct 12, 2023

When working with JavaScript objects and arrays, understanding the concepts of shallow copy and deep copy is crucial. In this article, we'll explore the key differences between these two copy methods and discuss how to achieve them using various techniques, including Lodash's `cloneDeep`, the structuredClone method, JSON methods (`JSON.stringify` and `JSON.parse`), and `Object.assign`.

Introduction

In JavaScript, objects and arrays are reference types, which means that copying them is not as straightforward as it may seem. Let's begin by understanding the difference between shallow copy and deep copy.

Shallow Copy vs. Deep Copy

Shallow Copy

A shallow copy creates a new object or array and copies references to the elements from the original object or array. This means that if the elements themselves are objects or arrays, the copy still references those nested objects.

```
const originalArray = [1, 2, [3, 4]];
const shallowCopy = Object.assign([], originalArray);
shallowCopy[2][0] = 99;
console.log(originalArray); // [1, 2, [99, 4]]
```

Deep Copy

A deep copy creates a completely new object or array, including copies of all nested elements. Changes made to the deep copy won't affect the original object.

```
const originalArray = [1, 2, [3, 4]];
const deepCopy = JSON.parse(JSON.stringify(originalArray));
deepCopy[2][0] = 99;
console.log(originalArray); // [1, 2, [3, 4]]
```

Copying with Lodash's cloneDeep

Lodash provides a `cloneDeep` method that easily creates a deep copy of an object or array. It's a reliable choice for deep copying complex data structures.

```
const _ = require('lodash');

const originalArray = [1, 2, [3, 4]];
const deepCopy = _.cloneDeep(originalArray);
deepCopy[2][0] = 99;
console.log(originalArray); // [1, 2, [3, 4]]
```

Using structuredClone

The `structuredClone` method, available in modern browsers, is designed for deep cloning structured data, including objects and arrays. It's particularly useful for cloning objects with non-JSON-safe values like functions.

```
const originalObject = { name: 'John', sayHello: function() { console.log('Hello
const deepCopy = structuredClone(originalObject);
deepCopy.name = 'Alice';
deepCopy.sayHello(); // Outputs "Hello!" even for the deep copy
```

JSON.stringify and JSON.parse

For cases where the data is JSON-safe (does not include functions or non-JSON data types), you can use `JSON.stringify` to serialize the object and then `JSON.parse` to deserialize it, effectively creating a deep copy:

```
const originalObject = { name: 'John', age: 30 };
const deepCopy = JSON.parse(JSON.stringify(originalObject));
deepCopy.name = 'Alice';
console.log(originalObject.name); // John
```

Using Object.assign

`Object.assign` can create a shallow copy of an object. It's simple and effective for creating a copy with one level of depth.

```
const originalObject = { name: 'John', age: 30 };
const shallowCopy = Object.assign({}, originalObject);
shallowCopy.name = 'Alice';
console.log(originalObject.name); // John
```

Conclusion

Understanding the distinction between shallow copy and deep copy in JavaScript is crucial when dealing with complex data structures. Shallow copies merely clone references, while deep copies create entirely independent copies. Techniques like Lodash's `cloneDeep`, `structuredClone`, JSON methods (`JSON.stringify` and `JSON.parse`), and `Object.assign` offer various ways to achieve deep and shallow copies, each with its own strengths and limitations. Depending on your specific use case, choose the method that best suits your needs to ensure proper data copying and manipulation in your JavaScript projects.

Written by Navneet Singh
223 Followers
https://taplink.cc/navneetskahlon
Follow