mdn web docs_

Deep copy

A **deep copy** of an object is a copy whose properties do not share the same references (point to the same underlying values) as those of the source object from which the copy was made. As a result, when you change either the source or the copy, you can be assured you're not causing the other object to change too. That behavior contrasts with the behavior of a shallow copy, in which changes to nested properties in the source or the copy may cause the other object to change too.

Two objects `o1` and `o2` are *structurally equivalent* if their observed behaviors are the same. These behaviors include:

1. The properties of `o1` and `o2` have the same names in the same order.
2. The values of their properties are structurally equivalent.
3. Their prototype chains are structurally equivalent (although when we deal with structural equivalence, these objects are usually plain objects, meaning they both inherit from `Object.prototype`).

Structurally equivalent objects can either be the same object (`o1 === o2`) or *copies* (`o1 !== o2`). Because equivalent primitive values always compare equal, you cannot make copies of them.

We can now define deep copies more formally as:

1. They are not the same object (`o1 !== o2`).
2. The properties of `o1` and `o2` have the same names in the same order.
3. The values of their properties are deep copies of each other.
4. Their prototype chains are structurally equivalent.

Deep copies may or may not have their prototype chains copied (and often they do not). But two objects with structurally non-equivalent prototype chains (for example, one is an array and the other is a plain object) are never copies of each other.

The copy of an object whose properties all have primitive values fits the definition of both a deep copy and a shallow copy. It is somewhat useless to talk about the depth of such a copy, though, because it has no nested properties and we usually talk about deep copying in the context of mutating nested properties.

In JavaScript, standard built-in object-copy operations (spread syntax, `Array.prototype.concat()`, `Array.prototype.slice()`, `Array.from()`, and `Object.assign()`) do not create deep copies (instead, they create shallow copies).

One way to make a deep copy of a JavaScript object, if it can be serialized, is to use `JSON.stringify()` to convert the object to a JSON string, and then `JSON.parse()` to convert the string back into a (completely new) JavaScript object:

JS

```
const ingredientsList = ["noodles", { list: ["eggs", "flour", "water"] }];
const ingredientsListDeepCopy = JSON.parse(JSON.stringify(ingredientsList));
```

Because a deep copy shares no references with its source object, any changes made to the deep copy do not affect the source object.

JS

```
// Change the value of the 'list' property in ingredientsListDeepCopy.
ingredientsListDeepCopy[1].list = ["rice flour", "water"];
// The 'list' property does not change in ingredients_list.
console.log(ingredientsList[1].list);
// Array(3) [ "eggs", "flour", "water" ]
```

However, while the object in the code above is simple enough to be serializable, many JavaScript objects are not serializable at all — for example, functions (with closures), Symbols, objects that represent HTML elements in the HTML DOM API, recursive data, and many other cases. Calling `JSON.stringify()` to serialize the objects in those cases will fail. So there's no way to make deep copies of such objects.

The web API `structuredClone()` also creates deep copies and has the advantage of allowing transferable objects in the source to be *transferred* to the new copy, rather than just cloned. It also handles more data types, such as `Error`. But note that `structuredClone()` isn't a feature of the JavaScript language itself — instead it's a feature of browsers and other JavaScript hosts that implement web APIs. And calling `structuredClone()` to clone a non-serializable object will fail in the same way that calling `JSON.stringify()` to serialize it will fail.

See also

- Related glossary terms:
  - Shallow copy
- `Window.structuredClone()`
- `WorkerGlobalScope.structuredClone()`