Shallow Copy

An introduction to cloning objects and shallow copy

Introduction

JavaScript made its way from being a toy language for simple animations to becoming a language for client side and server side web application development. Some generic concepts also made their way to the JavaScript world and developers become more and more aware of them. Cloning objects is one of these concepts.

Cloning can be quite complex. Prototypal inheritance, reference types and methods associated with an object may require a specialized approach. Restrictions on cloned data may simplify cloning. It is the responsibility of the developers to understand and apply the correct cloning method on a case by case basis.

Cloning methods of most libraries are implemented using shallow copying. One example is `_.clone`, the clone method of UnderscoreJs.

Shallow Copy :
All field keys and values of the original object are copied to the new object.

This definition has some implications. Recall the `shopTransaction` object from the last chapter.

```
var shopTransaction = {
    items: [ { name: 'Astro Mint Chewing Gum' } ],
    price: 1,
    amountPaid: 1000
}

var clonedTransaction = _.clone( shopTransaction );

clonedTransaction.price = 3;
clonedTransaction.items.push( { name: 'Tom&Berry Frozen Yoghurt' } );

console.log( 'clonedTransaction.price = ', clonedTransaction.price );
console.log( 'shopTransaction.price = ', shopTransaction.price );
```

```
console.log( 'clonedTransaction.items.length = ', clonedTransaction.items.length );

console.log( 'shopTransaction.items.length = ', shopTransaction.items.length );
```

Both `clonedTransaction` and `shopTransaction` have the same properties. As `price` and `amountPaid` are value types (numbers), their values are copied while cloning. The `items` property is a reference type. Therefore, both shopTransaction and clonedTransaction point to the exact same array in memory.

Changing the value of primitive types in `clonedTransaction` has no effects on the original object. However, `shopTransaction.items` and `clonedTransaction.items` are references pointing to the exact same array. It does not matter which reference we use to add an element to the array, the results are going to be visible under both references. Therefore, the result of the execution of the above code is:

```
clonedTransaction.price =  3
shopTransaction.price =  1
clonedTransaction.items.length =  2
shopTransaction.items.length =  2
```

> Modifying a value through a reference inside a cloned object also modifies the original object.

Check out a visual representation of the objects below.