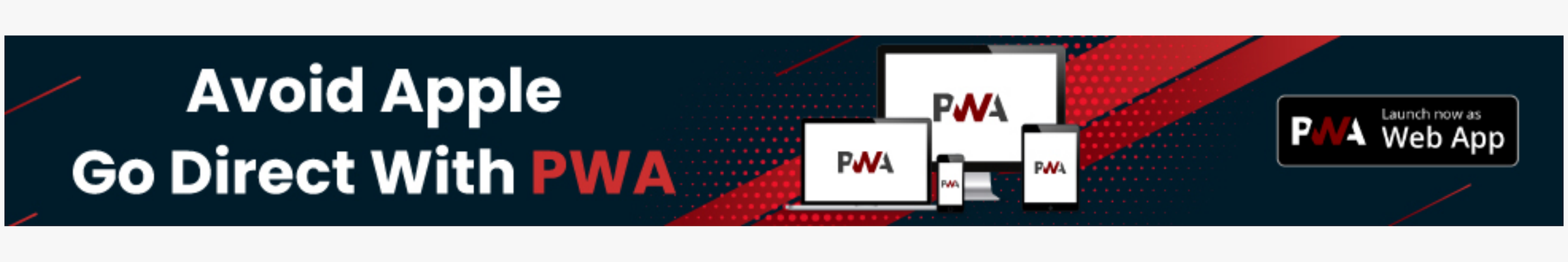


9 Ways to Remove Elements From A JavaScript Array - Plus How to Safely Clear JavaScript Arrays

Chris Love
Last Updated - Sat Jan 09 2021

JavaScript arrays allow you to group values and iterate over them. You can add and remove array elements in different ways. Unfortunately there is not a simple Array.remove method.

Remove Items From JavaScript Arrays

So, how do you delete an element from a JavaScript array?

Instead of a delete method, the JavaScript array has a variety of ways you can clean array values.

You can remove elements from the end of an array using pop, from the beginning using shift, or from the middle using splice. The JavaScript Array filter method to create a new array with desired items, a more advanced way to remove unwanted elements.

- Removing Elements from End of a JavaScript Array
- Removing Elements from Beginning of a JavaScript Array
- Using Splice to Remove Array Elements
- Removing Array Items By Value Using Splice
- The Lodash Array Remove Method
- Making a Remove Method
- Explicitly Remove Array Elements Using the Delete Operator
- Clear or Reset a JavaScript Array
- Summary

There are different methods and techniques you can use to remove elements from JavaScript arrays:

- pop - Removes from the End of an Array
- shift - Removes from the beginning of an Array
- splice - removes from a specific Array index
- filter - allows you to programatically remove elements from an Array

You will also learn some other ways you can remove elements from an array that may not be so obvious, like with LoDash.

Removing Elements from End of a JavaScript Array

JavaScript Array elements can be removed from the end of an array by setting the length property to a value less than the current value. Any element whose index is greater than or equal to the new length will be removed.

```
var ar = [1, 2, 3, 4, 5, 6];

ar.length = 4; // set length to remove elements
console.log( ar ); // [1, 2, 3, 4]
```

The pop method removes the last element of the array, returns that element, and updates the length property. The pop method modifies the array on which it is invoked, This means unlike using delete the last element is removed completely and the array length reduced.

```
var ar = [1, 2, 3, 4, 5, 6];

ar.pop(); // returns 6

console.log( ar ); // [1, 2, 3, 4, 5]
```

Removing Elements from Beginning of a JavaScript Array

How do you remove the first element of a JavaScript array?

The shift method works much like the pop method except it removes the first element of a JavaScript array instead of the last.

There are no parameters since the shift method only removed the first array element. When the element is removed the remaining elements are shifted down.

```
var ar = ['zero', 'one', 'two', 'three'];

ar.shift(); // returns "zero"

console.log( ar ); // ["one", "two", "three"]
```

The shift method returns the element that has been removed, updates the indexes of remaining elements, and updates the length property. It modifies the array on which it is invoked.

If there are no elements, or the array length is 0, the method returns undefined.

Using Splice to Remove Array Elements in JavaScript

The splice method can be used to add or remove elements from an array. The first argument specifies the location at which to begin adding or removing elements. The second argument specifies the number of elements to remove. The third and subsequent arguments are optional; they specify elements to be added to the array.

Here we use the splice method to remove two elements starting from position three (zero based index):

```
var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0];
var removed = arr.splice(2,2);
```

An array containing the removed elements is returned by the splice method. You can see the removed array contains [3, 4] and the original array contains the remaining values.

The splice method can also be used to remove a range of elements from an array.

```
var list = ["bar", "baz", "foo", "qux"];

list.splice(0, 2);
// Starting at index position 0, remove two elements ["bar", "baz"] and retains ["foo", "qux"].
```

Removing Array Items By Value Using Splice

If you know the value you want to remove from an array you can use the splice method. First you must identify the index of the target item. You then use the index as the start element and remove just one element.

```
var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0];

for( var i = 0; i < arr.length; i++){

    if ( arr[i] === 5) {

        arr.splice(i, 1);
    }

}

//=> [1, 2, 3, 4, 6, 7, 8, 9, 0]
```

This is a simple example where the elements are integers. If you have an array of objects you would need a more sophisticated routine.

This works if you only want to remove a single item. If you want to remove multiple items that match your criteria there is a glitch.

As the items are removed from the array the index still increments and the next item after your matched value is skipped.

The simple solution is to modify the above example to decrement the index variable so it does not skip the next item in the array.

```
var arr = [1, 2, 3, 4, 5, 5, 6, 7, 8, 5, 9, 0];

for( var i = 0; i < arr.length; i++){

    if ( arr[i] === 5) {
        arr.splice(i, 1);
        i--;
    }
}

//=> [1, 2, 3, 4, 6, 7, 8, 9, 0]
```

In the modified example I added 2 additional 5 values to the array. I also added 'i--;' after the splice call.

Now when you execute the loop it will remove every matching item.

Thanks to Kristian Sletten for pointing out the issue with the loop skipping the following item.

Using the Array filter Method to Remove Items By Value

Unlike the splice method, filter creates a new array. filter() does not mutate the array on which it is called, but returns a new array.

filter() has a single parameter, a callback method. The callback is triggered as the filter method iterates through the array elements. It will pass three values to the callback: the current value or element, the current array index and the full array.

The callback method should return either true or false. It is your responsibility to test the value (element) to see if it meets your criteria. If it does you can return true. Elements that return true are added to the new, filtered array.

```
var array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0];
var filtered = array.filter(function(value, index, arr){
    return value > 5;
});
//filtered => [6, 7, 8, 9]
//array => [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

You should note a new array containing matching values is returned. The original array is left untouched. I find this useful because I often want to retain an original data source, but retrieve subsets based on different logic sets.

The Lodash Array Remove Method

Sometimes utility libraries are the best way to solve more complex problems. Lodash provides a rich set of array manipulation methods, one being remove.

The Lodash remove method works much like the array filter method, but sort of in reverse. It does not save the original array values, but removes matching elements. It returns the matching elements as a new array.

```
var array = [1, 2, 3, 4];var evens = _.remove(array, function(n) { return n % 2 === 0;});console.log(array);// => [1, 3]console.log(evens);// => [2,
```

Making a Remove Method

As I mentionmed before, there is no native Array.remove method. The Lodash method does solve this problem, but you may not always want to use Lodash. This does not mean you cannot create a utility method. John Resig gave us a model to follow, however he extended the Array prototype, which is a bad idea.

Instead I created an Array remove utility method that can be added to a helper or utility library. Like the Lodash remove method the first parameter is the target array. It uses Array.filter to return elements not matching a value.

```
function arrayRemove(arr, value) {

    return arr.filter(function(ele){
        return ele != value;
    });
}

var result = arrayRemove(array, 6);
// result = [1, 2, 3, 4, 5, 7, 8, 9, 0]
```

This method is simple, it assumes simple values like numbers or strings. You could modify this method to use a customcomparison method, but I think it would be easier to just use the filter method directly.

Explicitly Remove Array Elements Using the Delete Operator

You can remove specific array elements using the delete operator:

```
var ar = [1, 2, 3, 4, 5, 6];

delete ar[4]; // delete element with index 4

console.log( ar );

// [1, 2, 3, 4, undefined, 6]

alert( ar );

// 1,2,3,4,,6
```

Using the delete operator does not affect the length property. Nor does it affect the indexes of subsequent elements. The array becomes sparse, which is a fancy way of saying the deleted item is not removed but becomes undefined. Compare using delete with the splice method described below.

The delete operator is designed to remove properties from JavaScript objects, which arrays are objects.

The reason the element is not actually removed from the array is the delete operator is more about freeing memory than deleting an element. The memory is freed when there are no more references to the value.

Clear or Reset a JavaScript Array

What if you want to empty an entire array and just dump all of it's elements?

There are a couple of techniques you can use to create an empty or new array.

The simplest and fastest technique is to set an array variable to an empty array:

```
var ar = [1, 2, 3, 4, 5, 6];
//do stuffar = [];
//a new, empty array!
```

The problem this can create is when you have references to the variable. The references to this variable will not change, they will still hold the original array's values. This of course can create a bug🐛.

This is an over simplified example of this scenario:

```
var arr1 = [1, 2, 3, 4, 5, 6];
var arr2 = arr1;
    // Reference arr1 by another variable arr1 = [];

console.log(arr2);
// Output [1, 2, 3, 4, 5, 6]
```

A simple trick to clear an array is to set its length property to 0.

```
var ar = [1, 2, 3, 4, 5, 6];

console.log(ar);
// Output [1, 2, 3, 4, 5, 6]

ar.length = 0;

console.log(ar);

// Output []
```

Another, sort of unnatural technique, is to use the splice method, passing the array length as the 2nd parameter. This will return a copy of the original elements, which may be handy for your scenario.

```
var ar = [1, 2, 3, 4, 5, 6];

console.log(ar);
// Output [1, 2, 3, 4, 5, 6]
ar.splice(0, ar.length);
console.log(ar);
// Output []
```

The last two techniques don't create a new array, but change the array's elements. This means references should also update.

There is another way, using a while loop. It feels a little odd to me, but at the same time looks fancy, so it may impress some friends!

```
var ar = [1, 2, 3, 4, 5, 6];

console.log(ar);
// Output [1, 2, 3, 4, 5, 6]

while (ar.length) {
    ar.pop();
}

console.log(ar);

// Output []
```

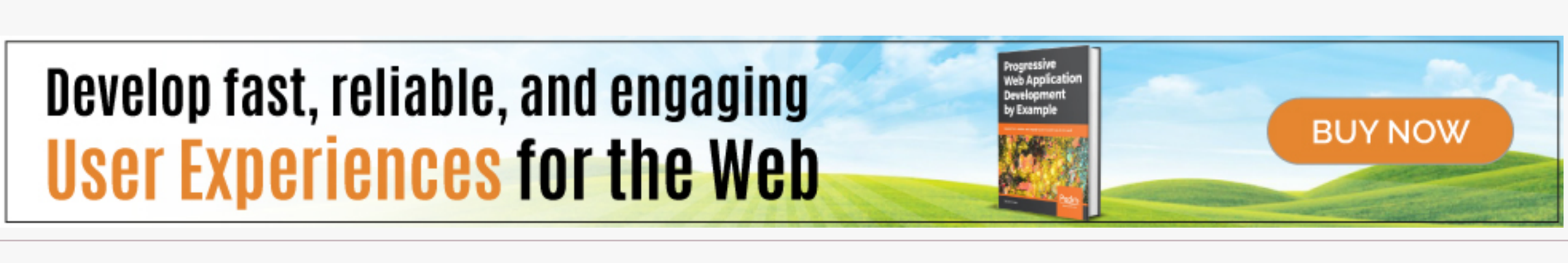
Not a way I would go about clearing a JavaScript array, but it works and it is readable. Some performance test have also shown this to be the fastest technique, so maybe it is better than I originally thought!

Summary

Removing JavaScript Array items is important to managing your data. There is not a single 'remove' method available, but there are different methods and techniques you can use to purge unwanted array items.

This article has reviewed these methods and how they can be used. You also saw how to create a helper method that makes removing items from an array a bit easier and consistent.

Thanks to Rob Sherwood for pointing out some syntactical typos!



Share This Article With Your Friends!

Why jQuery is Obsolete and Time to Stop Using It

Why jQuery is Obsolete and Time to Stop Using It

What is the !! (not not) operator in JavaScript?

What is the !! (not not) operator in JavaScript?

rel='noopener' Keeps You Safe and Fast

rel='noopener' Keeps You Safe and Fast

Service Worker Development Best Practices

Service Worker Development Best Practices