Get all unique values in a JavaScript array (remove duplicates)

Asked 9 years, 7 months ago Active 8 days ago Viewed 1.1m times

1149

343

I have an array of numbers that I need to make sure are unique. I found the code snippet below on the internet and it works great until the array has a zero in it. I found this other script here on SO that looks almost exactly like it, but it doesn't fail.

So for the sake of helping me learn, can someone help me determine where the prototype script is going wrong?

```
Array.prototype.getUnique = function() {
 var o = {}, a = [], i, e;
 for (i = 0; e = this[i]; i++) {o[e] = 1};
 for (e in o) {a.push (e)};
 return a;
}
```

More answers from duplicate question:

- Remove Duplicates from JavaScript Array

Similar question:

- Get all values with more than one occurrence (i.e.: not unique) in an array

javascript arrays unique

edited Jul 19 '18 at 0:25
APerson
5,230 4 29 46

asked Dec 25 '09 at 4:28
Mottie
57.5k 18 109 214

3 @hippietrail That older question is about finding and returning only the duplicates (I was confused too!). My question is more about why this function fails when an array has a zero in it. – Mottie Feb 12 '14 at 17:34

You probably want to make your question title less vague too. – hippietrail Feb 12 '14 at 17:38

For future readers, when start finding that you have to algorithmically modify the contents of your data structure all the time, (order them, remove repeating elements, etc.) or search for elements inside it at every iteration, it's safe to assume that you're using the wrong data structure in the first place and start using one that is more appropriate for the task at hand (in this case a hash set instead of array). – nurettin Dec 30 '14 at 11:16

1 e = element! :-) – RhinoDevel Aug 12 '15 at 12:45

1 @SamuelLiew That question is about finding and returning only the duplicate values - this question is about finding the unique values (with duplicates removed). – MT0 Nov 15 '17 at 10:16

73 Answers

1 2 3 next

2122

With *JavaScript 1.6* / *ECMAScript 5* you can use the native `filter` method of an Array in the following way to get an array with unique values:

```
function onlyUnique(value, index, self) {
    return self.indexOf(value) === index;
}

// usage example:
var a = ['a', 1, 'a', 2, '1'];
var unique = a.filter( onlyUnique ); // returns ['a', 1, 2, '1']
```

The native method `filter` will loop through the array and leave only those entries that pass the given callback function `onlyUnique`.

`onlyUnique` checks, if the given value is the first occurring. If not, it must be a duplicate and will not be copied.

This solution works without any extra library like jQuery or prototype.js.

It works for arrays with mixed value types too.

For old Browsers (<ie9), that do not support the native methods `filter` and `indexOf` you can find work arounds in the MDN documentation for filter and indexOf.

If you want to keep the last occurrence of a value, simple replace `indexOf` by `lastIndexOf`.

With ES6 it could be shorten to this:

```
// usage example:
var myArray = ['a', 1, 'a', 2, '1'];
var unique = myArray.filter((v, i, a) => a.indexOf(v) === i);
```

```
// unique is ['a', 1, 2, '1']
```

Thanks to Camilo Martin for hint in comment.

ES6 has a native object `Set` to store unique values. To get an array with unique values you could do now this:

```
var myArray = ['a', 1, 'a', 2, '1'];

let unique = [...new Set(myArray)];

// unique is ['a', 1, 2, '1']
```

The constructor of `Set` takes an iterable object, like Array, and the spread operator `...` transform the set back into an Array. Thanks to Lukas Liese for hint in comment.

edited Aug 1 '18 at 15:17
Thomas Ruiz
3,190 2 15 30

answered Jan 21 '13 at 12:46
TLindig
23.5k 2 22 29

38 This solution will run much slower, unfortunately. You're looping twice, once with filter and once with index of – Jack Franzen Nov 23 '13 at 10:11

16 @JackFranzen Slower than what? The solution from *Rafael*? *Rafaels* solution do not work for mixed type arrays. For my example `['a', 1, 'a', 2, '1']` you would get `['a', 1, 2]`. But this is not what I expected. BTW, much slower is very relative. – TLindig Nov 23 '13 at 17:40

21 In modern JS: `.filter((v,i,a)=>a.indexOf(v)==i)` (fat arrow notation). – Camilo Martin Jul 24 '16 at 8:43

96 `let unique_values = [...new Set(random_array)];` developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/… – Lukas Nov 19 '16 at 15:07

4 For a much more detailed answer, including many possibilities - such as sorting first, and dealing with varying data types - see stackoverflow.com/a/9229821/368896 – Dan Nissenbaum Jan 22 '17 at 21:08

709

Updated answer for ES6/ES2015: Using the Set, the single line solution is:

```
var items = [4,5,4,6,3,4,5,2,23,1,4,4,4]
var uniqueItems = Array.from(new Set(items))
```

Which returns

```
[4, 5, 6, 3, 2, 23, 1]
```

As le_m suggested, this can also be shortened using spread operator , like

```
var uniqueItems = [...new Set(items)]
```

edited Nov 14 '17 at 10:57

answered Oct 14 '15 at 9:42
A.T.
12.4k 5 31 54

8 Notice, that inner array wouldn't work `Array.from(new Set([[1,2],[1,2],[1,2,3]]))` – Alexander Goncharov Oct 24 '16 at 13:49

2 Performance of this solution compared to `myArray.filter((v, i, a) => a.indexOf(v) === i);` ? – Simoyw Mar 9 '17 at 10:08

26 Please note that if you use the `Set` and add objects instead of primitive values it will contain unique *references* to the objects. Thus the set `s` in `let s = new Set([{Foo:"Bar"}, {Foo:"Bar"}]);` will return this: `Set { { Foo: 'Bar' }, { Foo: 'Bar' } }` which is a `Set` with unique object references to objects that contain the same values. If you write `let o = {Foo:"Bar"};` and then create a set with two *references* like so: `let s2 = new Set([o,o]);` , then s2 will be `Set { { Foo: 'Bar' } }` – mortb Apr 5 '17 at 9:14

1 On Chrome 63, spread seems to outperform `Array.from` by about 35%. The difference on Firefox 57 is about 13%, and it's only about 10% difference on Safari 11.0.2. jsperf.com/set-conversion-using-array-from-vs-spread/1 – Matthew Herbst Jan 10 '18 at 18:56

1 new test case jsperf.com/array-filter-unique-vs-new-set/1 seems like `new Set` 's trophy – shukar Jan 13 at 18:55

130

You can also use underscore.js.

```
console.log(_.uniq([1, 2, 1, 3, 1, 4]));
```

```
<script src="http://underscorejs.org/underscore-min.js"></script>
```

Run code snippet Expand snippet

which will return:

```
[1, 2, 3, 4]
```

edited Mar 23 '17 at 4:00
Ruslan López
3,214 1 16 28

answered Jul 11 '12 at 16:25
kornfridge
3,100 5 24 38

16 Please do this folks. Don't jack something onto to the Array prototype. Please. – Jacob Dalton Apr 26 '16 at 20:06

4 @JacobDalton - This isn't extending the Array prototype. It's namespaced in the _ object. – superluminary Jun 23 '16 at 15:37

18 @superluminary I know that's why I said please **do** this. The accepted solution suggests modifying the Array prototype. DON'T do that. – Jacob Dalton Jun 24 '16 at 0:40

2 @JacobDalton There's no problem with modifying prototypes in your own code. If any code uses `for (var i in array)` , that's what you should be throwing away. That said, libraries probably shouldn't mess with prototypes, because they might be used in legacy/broken environments where a drunk intern at 4AM wrote code that might possibly break when used alongside clean JS that modifies the array prototype. – Camilo Martin Jul 24 '16 at 8:53

10 @JacobDalton Please don't do this. There's no need to add an extra library just for a small job that can be done with `array = [...new Set(array)]` – K48 Jul 6 '18 at 7:02

128

I realise this question has more than 30 answers already. But I've read through all the existing answers first and made my own research.

I split all answers to 4 possible solutions:

1. Use new ES6 feature: `[...new Set( [1, 1, 2] )];`
2. Use object `{ }` to prevent duplicates
3. Use helper array `[ ]`
4. Use `filter + indexOf`

Here's sample codes found in answers:

Use new ES6 feature: `[...new Set( [1, 1, 2] )];`

```
function uniqueArray0(array) {
  var result = Array.from(new Set(array));
  return result
}
```

Use object `{ }` to prevent duplicates

```
function uniqueArray1( ar ) {
  var j = {};

  ar.forEach( function(v) {
    j[v+ '::' + typeof v] = v;
  });

  return Object.keys(j).map(function(v){
    return j[v];
  });
}
```

Use helper array `[ ]`

```
function uniqueArray2(arr) {
    var a = [];
    for (var i=0, l=arr.length; i<l; i++)
        if (a.indexOf(arr[i]) === -1 && arr[i] !== '')
            a.push(arr[i]);
    return a;
}
```

Use `filter + indexOf`

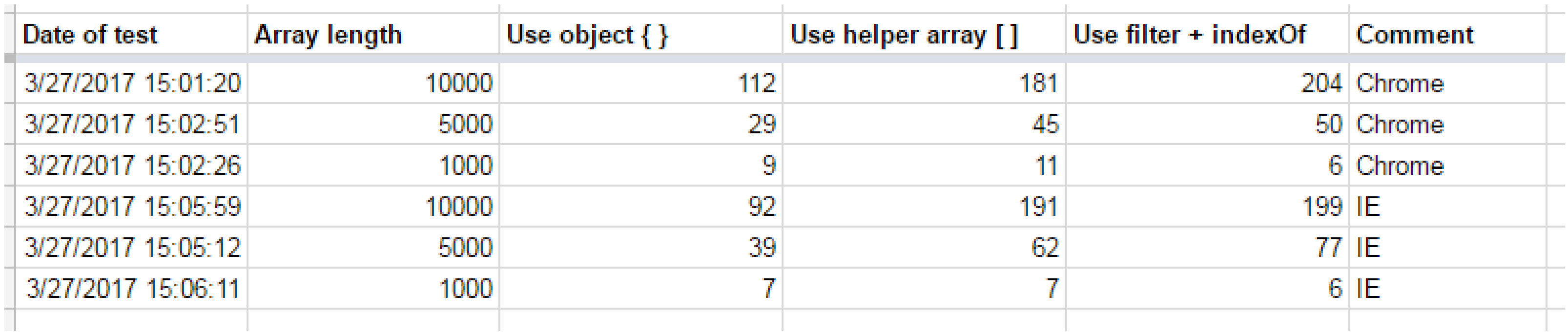
```
function uniqueArray3(a) {
  function onlyUnique(value, index, self) {
      return self.indexOf(value) === index;
  }

  // usage
  var unique = a.filter( onlyUnique ); // returns ['a', 1, 2, '1']

  return unique;
}
```

And I wondered which one is faster. I've made sample Google Sheet to test functions. Note: ECMA 6 is not avaliable in Google Sheets, so I can't test it.

Here's the result of tests:



Date of test	Array length	Use object { }	Use helper array [ ]	Use filter + indexOf	Comment
3/27/2017 15:01:20	10000	112	181	204	Chrome
3/27/2017 15:02:51	5000	29	45	50	Chrome
3/27/2017 15:02:26	1000	9	11	6	Chrome
3/27/2017 15:05:59	10000	92	191	199	IE
3/27/2017 15:05:12	5000	39	62	77	IE
3/27/2017 15:06:11	1000	7	7	6	IE

I expected to see that code using object `{ }` will win because it uses hash. So I'm glad that tests showed best results for this algorithm in Chrome and IE. Thanks to @rab for the code.

edited May 23 '17 at 11:55
Community ♦
1 1

answered Mar 27 '17 at 12:24
Max Makhrov
11k 4 20 37

1 In `uniqueArray2` , what is `&& arr[i] !== ''` for? – xcatliu Jun 30 '17 at 9:21

The option "filter + indexOf" is extremely slow on arrays over 100.000 items. I had to use "object map" approach however it breaks original sorting. – liberborn Sep 14 '17 at 13:18

The ES6 version is the fastest: jsperf.com/zorn-unique-array/1 – Timothy Zorn Aug 20 '18 at 11:17

Is higher number slower, not faster? – fletchsod Mar 20 at 15:32

1 @ fletchsod , numbers are the time in ms to run the code. – Max Makhrov Mar 20 at 15:38

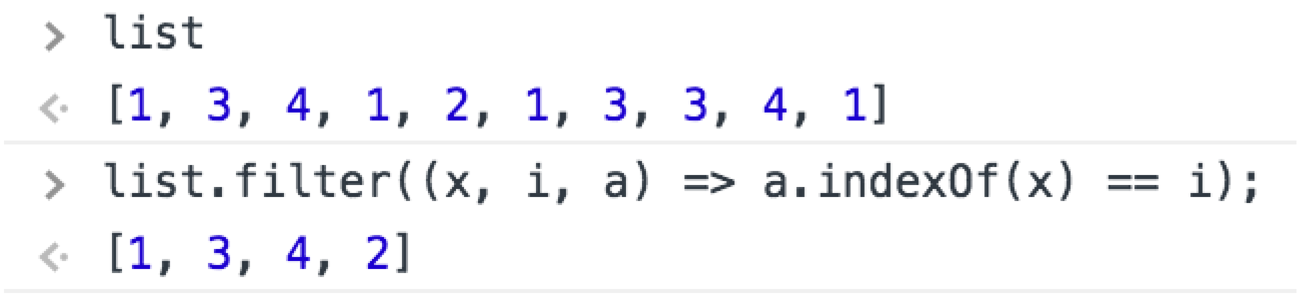
56

One Liner, Pure JavaScript

With ES6 syntax

```
list = list.filter((x, i, a) => a.indexOf(x) == i)
```

```
x --> item in array
i --> index of item
a --> array reference, (in this case "list")
```



```
> list
< [1, 3, 4, 1, 2, 1, 3, 3, 4, 1]
> list.filter((x, i, a) => a.indexOf(x) == i);
< [1, 3, 4, 2]
```

With ES5 syntax

```
list = list.filter(function (x, i, a) {
    return a.indexOf(x) == i;
});
```

Browser Compatibility: IE9+

edited Sep 13 '16 at 6:14

answered Sep 1 '16 at 13:32
Vamsi
7,211 3 32 41

9 @Spets It has quadratic cost, so it's not the best answer – Oriol Oct 15 '16 at 23:29

@Spets like all uniques / distincts... yes, be smart, use radix to sort first, be slower than 0(n2) quicksearch in most cases. – doker Dec 7 '16 at 15:53

thanks guys! Didn't realize it when I first looked at the code but now its a bit more obvious. – Spets Dec 15 '16 at 0:52

50

I have since found a nice method that uses jQuery

```
arr = $.grep(arr, function(v, k){
    return $.inArray(v ,arr) === k;
});
```

Note: This code was pulled from [Paul Irish's duck punching post](#) - I forgot to give credit :P

edited Dec 18 '12 at 14:27

answered Jul 12 '12 at 15:41

Mottie
57.5k 18 109 214

7 A concise solution, but calling inArray is way less efficient than calling hasOwnProperty. – Mister Smith Jun 5 '13 at 14:16

This is also O(N^2), right? Whereas the dictionary or hasOwnProperty approach would likely be O(N*logN). – speedplane Aug 24 '17 at 4:46

This worked for me, all other solutions were not supported be Internet Explorer. – kerl Jan 26 '18 at 19:47

46

Shortest solution with ES6: `[...new Set( [1, 1, 2] )];`

Or if you want to modify the Array prototype (like in the original question):

```
Array.prototype.getUnique = function() {
    return [...new Set( [this] )];
};
```

EcmaScript 6 is only partially implemented in modern browsers at the moment (Aug. 2015), but Babel has become very popular for transpiling ES6 (and even ES7) back to ES5. That way you can write ES6 code today!

If you're wondering what the `...` means, it's called the spread operator. From MDN: «The spread operator allows an expression to be expanded in places where multiple arguments (for function calls) or multiple elements (for array literals) are expected». Because a Set is an iterable (and can only have unique values), the spread operator will expand the Set to fill the array.

Resources for learning ES6:

- Exploring ES6 by Dr. Axel Rauschmayer
- Search "ES6" from JS weekly newsletters
- ES6 in depth articles from the Mozilla Hacks blog

edited Aug 18 '15 at 13:21
gregers
8,768 7 38 39

answered Apr 23 '14 at 12:42
Torsten Becker
3,885 2 16 19

3 you can do it even shorter, with `a = [...Set(a)]` , but, anyway, this is Firefox only, for now. – c69 Apr 27 '14 at 21:44

@c69, right, won't get shorter than that. SpiderMonkey users will appreciate, too. – Torsten Becker Apr 28 '14 at 13:32

Works with Babel. You just need to include Array.from shim also: `require ( "core-js/fn/array/from" );` – Vladislav Rastrusny Jun 26 '15 at 10:15

should it be Array.prototype.getUnique = function() { return [...new Set([this])]; }; or, Array.prototype.getUnique = function() { return [...new Set(this)]; }; I applied it on the following- $('body').html().toLowerCase().match(/([a-zA-Z0-9._+-]+@[a-zA-Z0-9._-]+\.[a-zA-Z0-9._-]+)/gi).getUnique(); First one returned me one unique value only where the second one retuned all removing duplicates. – ashique Jan 26 '18 at 4:51

34

Simplest solution:

```
var arr = [1, 3, 4, 1, 2, 1, 3, 3, 4, 1];
console.log([...new Set(arr)]);
```

Run code snippet Expand snippet

Or:

```
var arr = [1, 3, 4, 1, 2, 1, 3, 3, 4, 1];
console.log(Array.from(new Set(arr)));
```

Run code snippet Expand snippet

edited Mar 23 '17 at 1:09
Ruslan López
3,214 1 16 28

answered Mar 14 '16 at 22:01
Pedro L.
6,322 3 20 25

4 Note that this is not supported on IE10 and below. Note that IE11+ and Safari offer limited support (not even sure that above example works) developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/… – oriadam Apr 19 '16 at 11:21

31

The simplest, and fastest (in Chrome) way of doing this:

```
Array.prototype.unique = function() {
    var a = [];
    for (var i=0, l=this.length; i<l; i++)
        if (a.indexOf(this[i]) === -1)
            a.push(this[i]);
    return a;
}
```

Simply goes through every item in the array, tests if that item is already in the list, and if it's not, push to the array that gets returned.

According to jsPerf, this function is the fastest of the ones I could find anywhere - feel free to add your own though.

The non-prototype version:

```
function uniques(arr) {
    var a = [];
    for (var i=0, l=arr.length; i<l; i++)
        if (a.indexOf(arr[i]) === -1 && arr[i] !== '')
            a.push(arr[i]);
    return a;
}
```

Sorting

When also needing to sort the array, the following is the fastest:

```
Array.prototype.sortUnique = function() {
    this.sort();
    var last_i;
    for (var i=0;i<this.length;i++)
        if ((last_i = this.lastIndexOf(this[i])) !== i)
            this.splice(i+1, last_i-i);
    return this;
}
```

or non-prototype:

```
function sortUnique(arr) {
    arr.sort();
    var last_i;
    for (var i=0;i<arr.length;i++)
        if ((last_i = arr.lastIndexOf(arr[i])) !== i)
            arr.splice(i+1, last_i-i);
    return arr;
}
```

This is also faster than the above method in most non-chrome browsers.

edited Jan 30 '14 at 1:00

answered Jan 30 '14 at 0:10
Joeytje50
13.7k 6 50 70

On Linux, Chrome 55.0.2883 prefers your arr.unique() and swilliams' arrclone2.sortFilter() is slowest (78% slower). However, Firefox 51.0.0 (with lots of addons) has swilliams as fastest (yet still slower by Ops/sec than any other Chrome result) with mottie's jQuery $.grep(arr, jqFilter) being slowest (46% slower). Your arr.uniq() was 30% slower. I ran each test twice and got consistent results. Rafael's arr.getUnique() got second place in both browsers. – Adam Katz Feb 7 '17 at 0:11

jsPerf is buggy at the moment, so my edit to this test didn't commit everything, but it did result in adding two tests: Cocco's toUnique() beats Vamsi's ES6 list.filter() on both browsers, beating swilliams' sortFilter() for #1 on FF (sortFilter was 16% slower) and beating your sorted testing (which was slower by 2%) for #3 on Chrome. – Adam Katz Feb 7 '17 at 0:21

Ah, I hadn't caught that those tests were trivially small and don't really matter. A comment to the accepted answer describes that problem and offers a correction in a revision to the test, in which Rafael's code is easily the fastest and Joetje50's arr.unique code is 98% slower. I've also made another revision as noted in this comment. – Adam Katz Feb 7 '17 at 1:21

1 Well, actually the algorithm you implemented in `unique` function has O(n^2) complexity while the one in `getUnique` is O(n). The first one may be faster on small data sets, but how can you argue with the maths :) You can make sure the latter one is faster if you run it on an array of, say, 1e5 unique items – Mikhail Dudin Nov 14 '18 at 10:55

29

PERFORMANCE ONLY! this code is probably 10X faster than all the codes in here *works on all browsers and also has the lowest memory impact.... and more

if you don't need to reuse the old array;btw do the necessary other operations before you convert it to unique here is probably the fastest way to do this, also very short.

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

then you can try this

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

function toUnique(a, b, c) { //array,placeholder,placeholder
```

```
  b = a.length;
  while (c = --b)
    while (c--) a[b] !== a[c] || a.splice(c, 1);
  return a // not needed ;)
}
console.log(toUnique(array));
//[3, 4, 5, 6, 7, 8, 9, 0, 2, 1]
```

Run code snippet Expand snippet

I came up with this function reading this article...

http://www.shamasis.net/2009/09/fast-algorithm-to-find-unique-items-in-javascript-array/

I don't like the for loop. it has to many parameters.i like the while-- loop. while is the fastest loop in all browsers except the one we all like so much... chrome.

anyway i wrote the first function that uses while.And yep it's a little faster than the function found in the article.but not enough. `unique2()`

next step use modern js. `Object.keys` i replaced the other for loop with js1.7's Object.keys... a little faster and shorter (in chrome 2x faster) ;). Not enough!. `unique3()` .

at this point i was thinking about what i really need in MY unique function. i don't need the old array, i want a fast function. so i used 2 while loops + splice. `unique4()`

Useless to say that i was impressed.

chrome: the usual 150,000 operations per second jumped to 1,800,000 operations per second.

ie: 80,000 op/s vs 3,500,000 op/s

ios: 18,000 op/s vs 170,000 op/s

***safari:* 80,000 op/s vs 6,000,000 op/s**

Proof http://jsperf.com/wgu or better use console.time... microtime... whatever

`unique5()` is just to show you what happens if you want to keep the old array.

Don't use `Array.prototype` if yu don't know what your doing. i just did alot of copy and past. Use `Object.defineProperty(Array.prototype,...,writable:false,enumerable:false})` if you want to create a native prototype.example:

https://stackoverflow.com/a/20463021/2450730

Demo http://jsfiddle.net/46S7g/

NOTE: your old array is destroyed/becomestheunique after this operation.

if you can't read the code above ask, read a javascript book or here are some explainations about shorter code.
https://stackoverflow.com/a/21353032/2450730

some are using `indexOf` ... don't ... http://jsperf.com/dgfgghfghfghghgfhgfhfghfhgfh

for empty arrays

```
!array.length||toUnique(array);
```

edited May 23 '17 at 11:47
Community ♦
1 1

answered Aug 1 '14 at 14:49
cocco
13k 6 43 69

seems good to me – P6345uk Aug 5 '14 at 14:57

4 tested on node.js, with a 100k array of Urls (strings). The result was 2x slower than underscore.js _.uniq... although a separate jsperf agrees with you (jsperf.com/uniq-performance/5), I'm disappointed :(– xShirase Aug 13 '14 at 0:06

theoretically my toUnique should be faster especially on big arrays as i don't use indexOf and the array is smaller every iteration because i remove the duplicates. jsperf.com/dgfgghfghfghghgfhgfhfghfhgfh tests only the indexOf, chrome altough likes also indexOf & lastIndexOF. in any case looping through objects with in could not be faster than while or for.... i hope everyone knows that...anyway thx for the tests, i see they removed my test case from jsperf....btw ... you should round,ceil floor the numbers in your test else you don't have many duplicates.filter,map or other new stuff- – cocco Aug 13 '14 at 11:30

3 you are not testing correctly in jsperf... in your example you define the function everytime... but the underscore.js functions are already defined.. this penalizes my function. also test 3 & 4. another thing i should say is that if you use mixed variables (strings & numbers) you should replace a[b]!==a[c] with a[b]!=a[c] – cocco Aug 13 '14 at 15:54

1 I'm not sure if did the jsPerf correct, but seems that the Reduce alternative (that, for me is easier to understand) is somewhat 94% faster than your solution. jsperf.com/reduce-for-distinct Edit: Yours is slower on chrome, the same on Edge and faster on Firefox. – Victor Ivens Feb 3 '17 at 19:44

Many of the answers here may not be useful to beginners. If de-duping an array is difficult, will they really know about the prototype chain, or even jQuery?

27

In modern browsers, a clean and simple solution is to store data in a Set, which is designed to be a list of unique values.

```
const cars = ['Volvo', 'Jeep', 'Volvo', 'Lincoln', 'Lincoln', 'Ford'];
const uniqueCars = Array.from(new Set(cars));
```

The `Array.from` is useful to convert the Set back to an Array so that you have easy access to all of the awesome methods (features) that arrays have. There are also other ways of doing the same thing. But you may not need `Array.from` at all, as Sets have plenty of useful features like forEach.

If you need to support old Internet Explorer, and thus cannot use Set, then a simple technique is to copy items over to a new array while checking beforehand if they are already in the new array.

```
// Create a list of cars, with duplicates.
var cars = ['Volvo', 'Jeep', 'Volvo', 'Lincoln', 'Lincoln', 'Ford'];
// Create a list of unique cars, to put a car in if we haven't already.
var uniqueCars = [];

// Go through each car, one at a time.
cars.forEach(function (car) {
    // The code within the following block runs only if the
    // current car does NOT exist in the uniqueCars list
    // - a.k.a. prevent duplicates
    if (uniqueCars.indexOf(car) === -1) {
        // Since we now know we haven't seen this car before,
        // copy it to the end of the uniqueCars list.
        uniqueCars.push(car);
    }
});
```

To make this instantly reusable, let's put it in a function.

```
function deduplicate(data) {
    if (data.length > 0) {
        var result = [];

        data.forEach(function (elem) {
            if (result.indexOf(elem) === -1) {
                result.push(elem);
            }
        });

        return result;
    }
}
```

So to get rid of the duplicates, we would now do this.

```
var uniqueCars = deduplicate(cars);
```

The `deduplicate(cars)` part **becomes** the thing we named *result* when the function completes.

Just pass it the name of any array you like.

edited May 23 '17 at 11:55
Community ♦
1 1

answered Jan 15 '14 at 6:56
Seth Holladay
3,861 17 30

By the way, I used an array full of strings to show that my technique is flexible. It will work properly for numbers. – Seth Holladay Jan 17 '14 at 5:11

20

```
["Defects", "Total", "Days", "City", "Defects"].reduce(function(prev, cur) {
  return (prev.indexOf(cur) < 0) ? prev.concat([cur]) : prev;
 }, []);

[0,1,2,0,3,2,1,5].reduce(function(prev, cur) {
  return (prev.indexOf(cur) < 0) ? prev.concat([cur]) : prev;
 }, []);
```

answered Sep 19 '14 at 19:09
sergeyz
1,108 8 12

Can you explain why you didn't simply `push` the element onto the array instead of using `concat` ? I tried using push and it failed. I'm looking for an explanation. – makenova Jul 14 '15 at 15:20

1 The reason is in the return value. concat returns the modified array (exactly what needs to be returned inside the reduce function), while push returns an index at which you can access pushed value. Does that answer your question? – sergeyz Jul 14 '15 at 16:37

Apparently, this solution is quite fast in chrome (and node). jsperf.com/reduce-for-distinct That's the ES6 version: `[0,1,2,0,3,2,1,5].reduce((prev, cur) => ~prev.indexOf(cur) ? prev : prev.concat([cur]), []);` – Victor Ivens Feb 3 '17 at 19:50

side note: this implementation is not `NaN` -friendly – Alireza Jul 23 at 11:04

17

This prototype `getUnique` is not totally correct, because if i have a Array like: `["1",1,2,3,4,1,"foo"]` it will return `["1","2","3","4"]` and `"1"` is string and `1` is a integer; they are different.

Here is a correct solution:

```
Array.prototype.unique = function(a){
    return function(){ return this.filter(a) }
}(function(a,b,c){ return c.indexOf(a,b+1) < 0 });
```

using:

```
var foo;
foo = ["1",1,2,3,4,1,"foo"];
foo.unique();
```

The above will produce `["1",2,3,4,1,"foo"]` .

edited Apr 10 '14 at 10:26
Gajus
32.3k 38 180 328

answered Jul 27 '12 at 16:57
Gabriel Silveira
308 2 4

1 Note that `$foo = 'bar'` is the PHP way of declaring variables. It will work in javascript, but will create an implicit global, and generally shouldn't be done. – Camilo Martin Jun 12 '13 at 5:58

@CamiloMartin sorry but you're wrong, $foo is global because the example is not in a closure and he's missing the var keyword. Nothing to do with the dollar jsfiddle.net/robaldred/L2MRb – Rob Jul 17 '13 at 13:09

8 @Rob that's exactly what I'm saying, PHP people will think `$foo` is the way of declaring variables in javascript while actually `var foo` is. – Camilo Martin Jul 18 '13 at 17:57

16

We can do this using ES6 sets:

```
var duplicatedArray = [1, 2, 3, 4, 5, 1, 1, 1, 2, 3, 4];
var uniqueArray = Array.from(new Set(duplicatedArray));

console.log(uniqueArray);
```

Run code snippet Expand snippet

//The output will be

```
uniqueArray = [1,2,3,4,5];
```

edited Mar 22 at 16:04
demo
2,541 4 42 92

answered Apr 9 '18 at 3:49
chinmayan
930 9 11

12

Without extending Array.prototype (it is said to be a bad practice) or using jquery/underscore, you can simply `filter` the array.

By keeping last occurrence:

```
    function arrayLastUnique(array) {
        return array.filter(function (a, b, c) {
            // keeps last occurrence
            return c.indexOf(a, b + 1) < 0;
        });
    },
```

or first occurrence:

```
    function arrayFirstUnique(array) {
        return array.filter(function (a, b, c) {
            // keeps first occurrence
            return c.indexOf(a) === b;
        });
    },
```

Well, it's only javascript ECMAScript 5+, which means only IE9+, but it's nice for a development in native HTML/JS (Windows Store App, Firefox OS, Sencha, Phonegap, Titanium, ...).

edited May 23 '13 at 14:21 answered Apr 17 '13 at 16:48
Cœur **21.5k** 10 125 171

2 The fact that it's js 1.6 does not mean you can't use `filter`. At the MDN page they have an implementation for Internet Explorer, I mean, older browsers. Also: JS 1.6 refers only to Firefox's js engine, but the right thing to say it's that it is ECMAScript 5. – Camilo Martin May 23 '13 at 14:06

@CamiloMartin I changed 1.6 to ECMAScript5. Thanks. – Cœur May 23 '13 at 14:22

11

Magic

```
a.filter(e=>!(t[e]=e in t))
```

O(n) performance; we assume your array is in `a` and `t={}`. Explanation here (+Jeppe impr.)

Show code snippet

edited May 31 at 13:47 answered Nov 28 '18 at 19:45
Kamil Kiełczewski **18.7k** 8 85 112

11 this look so super cool, that without a solid explanation i fell you're gonna mine bitcoins when i run this – Ondřej Želazko Jan 8 at 14:16

3 what i meant is that you should expand your answer with some explanation and commented deconstruction of it. don't expect people will find useful answers like this. (though it really looks cool a probably works) – Ondřej Želazko Jan 9 at 9:49

Not magic, but is much like the "Set"-answers, using O(1) key-lookups in the dictionary. Do you need to increment the counters though? How about "e=>!(t[e]=e in t)". Nice answer though. – Jeppe Jan 13 at 20:21

1 @Jeppe when I run your improvement then I experience aha effect (before I don't know that I can use `in` operator outside the other construction than `for` loop :P) - Thank you - I appreciate it and will give +2 to your other good answers. – Kamil Kiełczewski Jan 14 at 3:32

1 doesn't that create a global variable `t` which keeps alive after the filtering…?? – philipp Mar 1 at 12:31

10

That's because `0` is a falsy value in JavaScript.

`this[i]` will be falsy if the value of the array is 0 or any other falsy value.

answered Dec 25 '09 at 4:32
Luca Matteis
24.2k 18 100 157

Ahhhh, ok I see now... but would there be an easy fix to make it work? – Mottie Dec 25 '09 at 4:46

10

```
Array.prototype.getUnique = function() {
    var o = {}, a = []
    for (var i = 0; i < this.length; i++) o[this[i]] = 1
    for (var e in o) a.push(e)
    return a
}
```

answered Dec 25 '09 at 5:11
ephemient
159k 31 236 362

I think this won't work if the array contains objects/arrays, and I'm not sure if it will preserve the type of scalars. – Camilo Martin May 23 '13 at 14:02

Yes, everything gets stringified. That could be fixed by storing the original value in `o` instead of just a `1`, although equality comparison would still be stringwise (although, out of all the possible Javascript equalities, it doesn't seem too unreasonable). – ephemient May 23 '13 at 17:43

The Array.prototype could be extended only with non enumerable methods .... Object.defineProperty(Array.prototype,"getUnique",{}) ... but the idea of using a helper object is very nice – bortunac Nov 17 '16 at 11:05

10

If you're using Prototype framework there is no need to do 'for' loops, you can use http://www.prototypejs.org/api/array/uniq like this:

```
var a = Array.uniq();
```

Which will produce a duplicate array with no duplicates. I came across your question searching a method to count distinct array records so after

> uniq()

I used

> size()

and there was my simple result. p.s. Sorry if i misstyped something

edit: if you want to escape undefined records you may want to add

> compact()

before, like this:

```
var a = Array.compact().uniq();
```

edited Nov 1 '11 at 13:26

answered Nov 1 '11 at 13:18
Decebal
923 1 16 26

14 because i found a better answer, i think about topics are for all people not just for the one who asked – Decebal Nov 1 '11 at 15:10

7

I had a slightly different problem where I needed to remove objects with duplicate id properties from an array. this worked.

```
let objArr = [{
  id: '123'
}, {
  id: '123'
}, {
  id: '456'
}];
```

```
objArr = objArr.reduce((acc, cur) => [
  ...acc.filter((obj) => obj.id !== cur.id), cur
], []);

console.log(objArr);
```

Run code snippet   Expand snippet

edited Mar 22 at 16:03
demo
2,541 4 42 92

answered Oct 11 '18 at 13:30
shunryu111
3,108 3 17 14

6

I'm not sure why Gabriel Silveira wrote the function that way but a simpler form that works for me just as well and without the minification is:

```
Array.prototype.unique = function() {
  return this.filter(function(value, index, array) {
    return array.indexOf(value, index + 1) < 0;
  });
};
```

or in CoffeeScript:

```
Array.prototype.unique = ->
  this.filter( (value, index, array) ->
    array.indexOf(value, index + 1) < 0
  )
```

answered Jun 7 '13 at 3:30
Dan Fox
61 1 1

Finding unique Array values in simple method

6

```
function arrUnique(a){
  var t = [];
  for(var x = 0; x < a.length; x++){
    if(t.indexOf(a[x]) == -1)t.push(a[x]);
  }
  return t;
}
arrUnique([1,4,2,7,1,5,9,2,4,7,2]) // [1, 4, 2, 7, 5, 9]
```

edited Jan 7 '16 at 13:39

answered Jan 7 '16 at 13:33
Saravanan Rajaraman
698 1 10 19

6

strange this hasn't been suggested before.. to remove duplicates by object key (`id` below) in an array you can do something like this:

```
const uniqArray = array.filter((obj, idx, arr) => (
  arr.findIndex((o) => o.id === obj.id) === idx
))
```

answered Feb 1 '18 at 16:36
daviestar
2,639 2 19 40

5

From **Shamasis Bhattacharya**'s blog (O(2n) time complexity) :

```
Array.prototype.unique = function() {
    var o = {}, i, l = this.length, r = [];
    for(i=0; i<l;i+=1) o[this[i]] = this[i];
    for(i in o) r.push(o[i]);
    return r;
};
```

From **Paul Irish**'s blog: improvement on JQuery `.unique()` :

```
(function($){
        var _old = $.unique;

        $.unique = function(arr){

            // do the default behavior only if we got an array of elements
            if (!!arr[0].nodeType){
                return _old.apply(this,arguments);
            } else {
                // reduce the array to contain no dupes via grep/inArray
                return $.grep(arr,function(v,k){
                    return $.inArray(v,arr) === k;
                });
            }
        };
    })(jQuery);

    // in use..
    var arr = ['first',7,true,2,7,true,'last','last'];
    $.unique(arr); // ["first", 7, true, 2, "last"]

    var arr = [1,2,3,4,5,4,3,2,1];
    $.unique(arr); // [1, 2, 3, 4, 5]
```

edited Dec 18 '12 at 14:14

answered Dec 18 '12 at 14:09
Frosty Z
16.2k 9 62 92

Looks like it isn't the fastest actually: jsperf.com/unique-in-array/20 – Vladislav Rastrusny Apr 29 '15 at 13:10

5

To address the problem the other way around, it may be useful to have no duplicate while you load your array, the way Set object would do it but it's not available in all browsers yet. It saves memory and is more efficient if you need to look at its content many times.

```
Array.prototype.add = function (elem) {
   if (this.indexOf(elem) == -1) {
      this.push(elem);
   }
}
```

Sample:

```
set = [];
[1,3,4,1,2,1,3,3,4,1].forEach(function(x) { set.add(x); });
```

Gives you `set = [1,3,4,2]`

edited Jan 27 '16 at 20:52

answered Jan 27 '16 at 20:43

Le Droid
2,711 1 25 28

5

Es6 based solution...

```
var arr = [2, 3, 4, 2, 3, 4, 2];
const result = [...new Set(arr)];
console.log(result);
```

Run code snippet   Expand snippet

edited Mar 22 at 16:01
demo
2,541 4 42 92

answered Feb 9 at 15:17
Vahid Akhtar
359 3 6

5

Now using sets you can remove duplicates and convert them back to the array.

```
var names = ["Mike","Matt","Nancy", "Matt","Adam","Jenny","Nancy","Carl"];

console.log([...new Set(names)])
```

Run code snippet   Expand snippet

answered May 17 at 8:18

Krishnadas PC
2,290 20 22

4

If you're okay with extra dependencies, or you already have one of the libraries in your codebase, you can remove duplicates from an array in place using LoDash (or Underscore).

Usage

If you don't have it in your codebase already, install it using npm:

```
npm install lodash
```

Then use it as follows:

```
import _ from 'lodash';
let idArray = _.uniq ([
    1,
    2,
    3,
    3,
    3
]);
console.dir(idArray);
```

Out:

```
[ 1, 2, 3 ]
```

edited Jun 21 '18 at 13:29
Constant Meiring
2,237 2 30 50

answered Jun 11 '18 at 11:34
NikeshPathania
129 5

3

If anyone using knockoutjs

```
ko.utils.arrayGetDistinctValues()
```

BTW have look at all `ko.utils.array*` utilities.

answered May 23 '13 at 13:33
Torbjörn Nomell
2,803 2 19 19

3

I found that serializing they hash key helped me get this working for objects.

```
Array.prototype.getUnique = function() {
        var hash = {}, result = [], key;
        for ( var i = 0, l = this.length; i < l; ++i ) {
            key = JSON.stringify(this[i]);
            if ( !hash.hasOwnProperty(key) ) {
                hash[key] = true;
                result.push(this[i]);
            }
        }
        return result;
    }
```

answered Jul 18 '13 at 1:52
Jason
965 1 7 7

---

1 2 3 next

protected by Samuel Liew ♦ Nov 9 '17 at 3:15

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?