What's the difference between "Array()" and "[]" while declaring a JavaScript array?

Asked 15 years, 6 months ago Modified 1 year, 1 month ago Viewed 938k times

966

What's the real difference between declaring an array like this:

```
var myArray = new Array();
```

and

```
var myArray = [];
```

javascript arrays declaration

Share

Improve this question

Follow

edited Dec 25, 2015 at 9:48

Mi-Creativity

9,624 ● 10 ● 39 ● 48

asked May 31, 2009 at 11:23

Amr Elgarhy

68.8k ● 70 ● 192 ● 310

2 `[]` token: `ARRAY_INIT`; `new Array` tokens: `NEW, IDENTIFIER`; `new Array()` tokens: `NEW, IDENTIFIER, CALL` – noobninja Feb 18, 2019 at 6:16

20 Answers

Sorted by: Highest score (default)

1074

There is a difference, but there is no difference in that example.

Using the more verbose method: `new Array()` does have one extra option in the parameters: if you pass a number to the constructor, you will get an array of that length:

```
x = new Array(5);
alert(x.length); // 5
```

To illustrate the different ways to create an array:

```
var a = [],            // these are the same
    b = new Array(),   // a and b are arrays with length 0

    c = ['foo', 'bar'],           // these are the same
```

```
    d = new Array('foo', 'bar'),  // c and d are arrays with 2 strings

    // these are different:
    e = [3]             // e.length == 1, e[0] == 3
    f = new Array(3),   // f.length == 3, f[0] == undefined

;
```

Another difference is that when using `new Array()` you're able to set the size of the array, which affects the stack size. This can be useful if you're getting stack overflows (Performance of Array.push vs Array.unshift) which is what happens when the size of the array exceeds the size of the stack, and it has to be re-created. So there can actually, depending on the use case, be a performance increase when using `new Array()` because you can prevent the overflow from happening.

As pointed out in this answer, `new Array(5)` will not actually add five `undefined` items to the array. It simply adds space for five items. Be aware that using `Array` this way makes it difficult to rely on `array.length` for calculations.

Share

Improve this answer

Follow

edited Jun 7, 2019 at 10:57

Adam Gerthel

693 ● 3 ● 10 ● 24

answered May 31, 2009 at 11:25

nickf

546k ● 198 ● 658 ● 725

70 This is slightly wrong. There is one very important difference between new Array() and [] I'll elaborate in my answer. – coderjoe Aug 13, 2009 at 19:01

34 But as noted in your answer, it's only different if you are completely insane and overwrite the Array function..? – nickf Aug 13, 2009 at 22:59

21 Well the importance is that using the new operator causes the interpreter to take all sorts of extra steps to go to the global scope, look for the constructor, call the constructor and assign the result... which in the majority case is going to be a a runtime array. You can avoid the overhead of looking for the global constructor by just using []. It may seem small, but when you're shooting for near real-time performance in your app, it can make a difference. – coderjoe Aug 14, 2009 at 2:02

5 There is a huge performance difference: jsperf.com/create-an-array-of-initial-size/2 – Marco Luglio Jan 23, 2013 at 20:28

4 True, but you also don't need semicolons or line-breaks in most places where I add them. It's about consistency and legibility. You know, IMHO. – nickf Nov 7, 2013 at 1:15

815

The difference between creating an array with the implicit array and the array constructor is subtle but important.

When you create an array using

```
var a = [];
```

You're telling the interpreter to create a new runtime array. No extra processing necessary at all. Done.

If you use:

```
var a = new Array();
```

You're telling the interpreter, I want to call the constructor "`Array`" and generate an object. It then looks up through your execution context to find the constructor to call, and calls it, creating your array.

You may think "Well, this doesn't matter at all. They're the same!". Unfortunately you can't guarantee that.

Take the following example:

```
function Array() {
    this.is = 'SPARTA';
}

var a = new Array();
var b = [];

alert(a.is);  // => 'SPARTA'
alert(b.is);  // => undefined
a.push('Woa'); // => TypeError: a.push is not a function
b.push('Woa'); // => 1 (OK)
```

In the above example, the first call will alert 'SPARTA' as you'd expect. The second will not. You will end up seeing undefined. You'll also note that b contains all of the native Array object functions such as `push`, where the other does not.

While you may expect this to happen, it just illustrates the fact that `[]` is not the same as `new Array()`.

It's probably best to just use `[]` if you know you just want an array. I also do not suggest going around and redefining Array...

Share

Improve this answer

Follow

edited Sep 28, 2017 at 17:52

Community Bot

1 • 1

answered Aug 13, 2009 at 19:00

coderjoe

11.2k • 2 • 27 • 25

188 Well, good to know I suppose. What sort of person would overwrite the array class, I do not know... – nickf Aug 13, 2009 at 22:58

171 You're absolutely right. Only a madman would overwrite the array class. Now take a moment and consider all the extra work using new Array() makes the interpreter do to support these

madmen. I just avoid it all together with []. – coderjoe Aug 14, 2009 at 2:01

57 Good example of the kind of global pollution that is possible with JavaScript. – David Snabel-Caunt Feb 4, 2010 at 23:11

8 Worth noting that the new Array(size) is faster than other possible methods using the [] notation. Source: jsperf.com/create-an-array-of-initial-size/2 – Marco Luglio Jan 23, 2013 at 20:25

10 Unfortunately that test is improperly prepared. It's testing an initialization of an Array with initializations of arrays followed by Array access. There's no control to prove that the browsers are actually pre-allocating the memory (which the specification does not say they must do). If we can assume that array access is constant and the majority of the time will be spent allocating memory in both examples then [] could be preferable if you're instantiating millions of arrays. jsperf.com/array-instanciation – coderjoe Jan 30, 2013 at 15:34

131

There is an important difference that no answer has mentioned yet.

From this:

```
new Array(2).length           // 2
new Array(2)[0] === undefined // true
new Array(2)[1] === undefined // true
```

You might think the `new Array(2)` is equivalent to `[undefined, undefined]`, **but it's NOT!**

Let's try with `map()`:

```
[undefined, undefined].map(e => 1)  // [1, 1]
new Array(2).map(e => 1)            // "(2) [undefined × 2]" in Chrome
```

See? The semantics are totally different! So why is that?

According to ES6 Spec 22.1.1.2, the job of `Array(len)` is just creating a new array whose property `length` is set to the argument `len` and that's it, meaning there isn't any **real element** inside this newly created array.

Function `map()`, according to spec 22.1.3.15 would firstly check `HasProperty` then call the callback, but it turns out that:

```
new Array(2).hasOwnProperty(0) // false
[undefined, undefined].hasOwnProperty(0) // true
```

And that's why you can not expect any iterating functions working as usual on arrays created from `new Array(len)`.

BTW, Safari and Firefox have a much better "printing" to this situation:

```
// Safari
new Array(2)             // [](2)
new Array(2).map(e => 1) // [](2)
[undefined, undefined]   // [undefined, undefined] (2)

// Firefox
new Array(2)             // Array [ <2 empty slots> ]
new Array(2).map(e => 1) // Array [ <2 empty slots> ]
[undefined, undefined]   // Array [ undefined, undefined ]
```

I have already submitted an issue to Chromium and ask them to fix this confusing printing: https://bugs.chromium.org/p/chromium/issues/detail?id=732021

UPDATE: It's already fixed. Chrome now printed as:

```
new Array(2)             // (2) [empty × 2]
```

Share

Improve this answer

Follow

edited Dec 19, 2019 at 0:45

answered Jun 10, 2017 at 9:26

Hux

1,457 ● 1 ● 9 ● 6

1 This makes sense, I just bumped into a different construct I can't find documented references for: `[...Array(2)]` which is equivalent to `[undefined, undefined]` from a results standpoint. – Roberto Andrade Feb 28, 2019 at 22:05

Looks like the answer to my previous comments is "Spread operator": javascript.info/rest-parameters-spread-operator – Roberto Andrade Feb 28, 2019 at 22:13

@RobertoAndrade that make sense because spread operator will need to read off the entries so it can apply such copying...and reading such empty slot return `undefined` as usual. – Hux Dec 19, 2019 at 0:52

The equivalent to `new Array(2)` should be `[,,]`, not `[undefined, undefined]`, no? – angleKH Aug 22, 2021 at 9:50

54

Oddly enough, `new Array(size)` is almost 2x faster than `[]` in Chrome, and about the same in FF and IE (measured by creating and filling an array). It only matters if you know the approximate size of the array. If you add more items than the length you've given, the performance boost is lost.

More accurately: `Array(` is a fast constant time operation that allocates no memory, wheras `[]` is a linear time operation that sets type and value.

Share

Improve this answer

edited Feb 5, 2019 at 0:38

answered Jun 28, 2010 at 12:35

Follow

Eric Leschinski
153k ● 96 ● 421 ● 335

gblazex
50.1k ● 12 ● 99 ● 92

3 I've tested it much in Node.js: when you need to put some amount of items in array, `new Array(length)` on 0 <= size <= ~1000, on size > ~1000 wins `[]` – glukki Jun 21, 2013 at 15:33

1 check this stackoverflow.com/questions/7375120/… – Xsmael Sep 16, 2017 at 17:01

44

For more information, the following page describes why you never need to use `new Array()`

> You never need to use `new Object()` in JavaScript. Use the object literal `{}` instead. Similarly, don't use `new Array()`, use the array literal `[]` instead. Arrays in JavaScript work nothing like the arrays in Java, and use of the Java-like syntax will confuse you.
>
> Do not use `new Number`, `new String`, or `new Boolean`. These forms produce unnecessary object wrappers. Just use simple literals instead.

Also check out the comments - the `new Array(length)` form does not serve any useful purpose (at least in today's implementations of JavaScript).

Share

Improve this answer

Follow

edited Apr 12, 2018 at 20:40

Aliaksandr Sushkevich
12.4k ● 8 ● 40 ● 46

answered May 31, 2009 at 16:29

BarelyFitz
1,977 ● 13 ● 17

3 Crockford, also, says to use [] rather than new Array(). Unfortunately, he doesn't say why in the linked article. I assume it's just a matter of space and speed. javascript.crockford.com/code.html – Nosredna May 31, 2009 at 16:48

4 Crockford's not a fan of the using the "new" keyword to create a new instance of an object in Javascript. In lectures he's stated it's his belief that it creates ambiguity and doesn't fit in well with Javascript's prototype style inheritance. He's specifically referring to user created object constructors, but given that belief, it's easy to see why he'd recommend you not use it with the built-ins as well when there's an alternative syntax. – Alana Storm May 31, 2009 at 17:15

@Alan Storm: at least for Number, String, and Boolean he says "these forms produce unnecessary object wrappers", but I guess that wouldn't apply to Array. – BarelyFitz May 31, 2009 at 19:10

8

The first one is the default object constructor call. You can use it's parameters if you want.

```
var array = new Array(5); //initialize with default length 5
```

The second one gives you the ability to create not empty array:

```
var array = [1, 2, 3]; // this array will contain numbers 1, 2, 3.
```

Share Improve this answer Follow

answered May 31, 2009 at 11:53

Bogdan Gusiev
8,307 ● 16 ● 62 ● 83

1 You can do the same thing with verbose constructor: var array = new Array(1, 2, 3); – nickf May 31, 2009 at 12:15

So then I guess you can do `var array = [5]` using the square brackets but not using the constructor as `var array = Array(5)` makes an empty array of 5 elements. – cdmckay May 31, 2009 at 16:31

cdmckay - that's incorrect. var a=[5] would be an array with a single item - the number five. – BarelyFitz May 31, 2009 at 16:44

3 @BarelyFitz: That's what I said. In Bogdans' answer, he says that the constructor call can't be used to initalize an array, but he was wrong. My comment was merely to clarify that you can't use the constructor call to initialize an array of a single element. – cdmckay May 31, 2009 at 16:52

1 @cdmckay: sorry, I misunderstood your comment. To clarify: new Array(arg) - if arg is numeric this creates an empty array with length=arg; new Array(arg1, arg2) - creates a new array and initializes the array elements. So if you want to create an array with one numeric element like [5], you cannot do it using new Array(5). But really you should never use new Array() so this is a moot point. – BarelyFitz May 31, 2009 at 19:01

8

In order to better understand `[]` and `new Array()`:

```
> []
  []
> new Array()
  []
> [] == []
  false
> [] === []
  false
> new Array() == new Array()
  false
> new Array() === new Array()
  false
> typeof ([])
  "object"
> typeof (new Array())
  "object"
> [] === new Array()
```

```
  false
> [] == new Array()
  false
```

The above result is from Google Chrome console on Windows 7.

Share Improve this answer Follow

answered Jul 27, 2012 at 22:56

Peter Lee

13.8k ● 11 ● 74 ● 100

4 but why is [] == [] or [] === [] false ? – anu Mar 5, 2015 at 2:40

5 "An expression comparing Objects is only true if the operands reference the same Object." (source: developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/…) – srph Nov 20, 2015 at 1:26

1 This answer cherry picks examples to imply that the two constructs are identical. Other posts on this page point out the differences, such as `Array(3)` or `new Array(3)` is not the same as `[3]`. – ggorlen Jun 4, 2019 at 16:51

5

I can explain in a more specific way starting with this example that's based on Fredrik's good one.

```
var test1 = [];
test1.push("value");
test1.push("value2");

var test2 = new Array();
test2.push("value");
test2.push("value2");

alert(test1);
alert(test2);
alert(test1 == test2);
alert(test1.value == test2.value);
```

I just added another value to the arrays, and made four alerts: The first and second are to give us the value stored in each array, to be sure about the values. They will return the same! Now try the third one, it returns false, that's because

> JS treats **test1** as a **VARIABLE with a data type of array**, and it treats **test2** as an **OBJECT with the functionality of an array**, and there are few slight differences here.

The first difference is when we call test1 it calls a variable without thinking, it just returns the values that are stored in this variable disregarding its data type! But, when we call test2 it calls the *Array()* function and then it stores our *"Pushed"* values in its

"Value" property, and the same happens when we alert test2, it returns the *"Value"* property of the array object.

So when we check if test1 equals test2 of course they will never return true, one is a function and the other is a variable (with a type of array), even if they have the same value!

To be sure about that, try the 4th alert, with the .value added to it; it will return true. In this case we tell JS "Disregarding the type of the container, whether was it function or variable, please compare the values that are stored in each container and tell us what you've seen!" that's exactly what happens.

I hope I said the idea behind that clearly, and sorry for my bad English.

Share Improve this answer Follow

answered May 6, 2013 at 16:52

Kholio

331 ● 3 ● 6

19 It's amazing that such complete and utter nonsense has been voted up. The comparison between the arrays will be false no matter how you make them because it compares the object identity and they are different objects. Arrays have no value property. `[]` and `new Array()` is identical; `.value` will be `undefined` in both cases, and comparing them will always be false. – slikts Sep 30, 2015 at 22:26

Agreed, this answer makes no sense and shows nothing. There is no such thing as `array.value`. and both `typeof []` and `typeof new Array()` return `object`. It's one of the reasons why there is a function called `Array.isArray` – user128511 Dec 19, 2019 at 0:57

Well guys, I agree with you :) I wrote a complete nonsense at that time :). Now I've added the correct answer to the same question. Let me know what do you see. I really appreciate your opinions. – Kholio Jun 14, 2020 at 15:24

5

Well, `var x = new Array()` is different than `var x = []` is different in some features I'll just explain the most useful two (in my opinion) of them.

Before I get into explaining the differences, I will set a base first; when we use `x = []` defines a new variable with data type of Array, and it inherits all the methods that belong to the array prototype, something pretty similar (but not exactly) to extending a class. However, when we use `x = new Array()` it initilizes a clone of the array prototype assigned to the variable `x`.

Now let's see what are the difference

The First Difference is that using `new Array(x)` where `x` is an integer, initilizes an array of `x` undefined values, for example `new Array(16)` will initialize an array with

16 items all of them are undefined. This is very useful when you asynchronously fill an array of a predefined length. For example (again :)) let's say you are getting the results of 100 competitiors, and you're receiving them asynchronously from a remote system or db, then you'll need to allocate them in the array according to the rank once you receive each result. In this very rare case you will do something like `myArray[result.rank - 1] = result.name`, so the rank 1 will be set to the index 0 and so on.

The second difference is that using `new Array()` as you already know, instanciates a whole new clone of the array prototype and assigns it to your variable, that allows you to do some magic (not recommended by the way). This magic is that you can overwrite a specific method of the legacy array methods. So, for example you can set the `Array.push` method to push the new value to the beginning of the array instead of the end, and you can also add new methods (this is better) to this specific clone of the Array Prototype. That will allow you to define more complex types of arrays throughout your project with your own added methods and use it as a class.

Last thing, if you're from the very few people (that I truly love) that care about processing overhead and memory consumption of your app, you'd never touch `new Array()` without being desperate to use it :).

I hope that has explained enough about the beast `new Array()` :)

Share

Improve this answer

Follow

edited Nov 14, 2023 at 21:37

M H

565 ● 7 ● 19

answered Jun 14, 2020 at 15:14

Kholio

331 ● 3 ● 6

4

There is no difference when you initialise array without any length. So `var a = []` & `var b = new Array()` is same.

But if you initialise array with length like `var b = new Array(1);`, it will set array object's length to 1. So its equivalent to `var b = []; b.length=1;`.

This will be problematic whenever you do array_object.push, it add item after last element & increase length.

```
var b = new Array(1);
b.push("hello world");
console.log(b.length); // print 2
```

vs

```
var v = [];
a.push("hello world");
```

```
console.log(b.length); // print 1
```

Share Improve this answer Follow

answered Mar 13, 2019 at 18:59

Aniruddha

3,618 ● 6 ● 30 ● 38

4

There's more to this than meets the eye. Most other answers are correct **BUT ALSO**..

`new Array(n)`

- Allows engine to reallocates space for `n` elements
- Optimized for array **creation**
- Created array is marked **sparse** which has the least performant array operations, that's because each index access has to check bounds, see if value exists and **walk the prototype chain**
- If array is marked as sparse, there's no way back (at least in V8), it'll always be slower during its lifetime, even if you fill it up with content (packed array) 1ms or 2 hours later, doesn't matter

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

- Created array is marked **packed** (unless you use `delete` or `[1,,3]` syntax)
- Optimized for array **operations** (`for ..`, `forEach`, `map`, etc)
- Engine needs to reallocate space as the array grows

> This *probably* isn't the case for older browser versions/browsers.

Share Improve this answer Follow

answered Dec 10, 2019 at 15:49

Solo

6,957 ● 7 ● 37 ● 74

3

The first one is the default object constructor call.mostly used for dynamic values.

```
var array = new Array(length); //initialize with default length
```

the second array is used when creating static values

```
var array = [red, green, blue, yellow, white]; // this array will contain values.
```

Share

Improve this answer

Follow

edited Aug 31, 2016 at 17:13

miken32

42.7k ● 16 ● 121 ● 171

answered Aug 31, 2016 at 17:06

Parth Raval

4,395 ● 3 ● 24 ● 39

2

The difference of using

```
var arr = new Array(size);
```

Or

```
arr = [];
arr.length = size;
```

As been discussed enough in this question.

I would like to add the speed issue - the **current** fastest way, on `google chrome` is the second one.

But pay attention, these things tend to change a lot with updates. Also the run time will differ between different browsers.

For example - the 2nd option that i mentioned, runs at 2 million [ops/second] on `chrome`, but if you'd try it on `mozilla dev.` you'd get a surprisingly higher rate of 23 million.

Anyway, I'd suggest you check it out, every once in a while, on different browsers (and machines), using site as such

Share

Improve this answer

Follow

edited Dec 18, 2016 at 15:46

Xavier Egea

4,762 ● 3 ● 26 ● 39

answered Mar 6, 2016 at 9:23

Meir Elad

75 ● 2 ● 11

2

As I know the diference u can find the slice(or the other funcitons of Array) like **code1**.and **code2** show u *Array* and his *instances*:

code1:

```
[].slice; // find slice here
var arr = new Array();
```

```
arr.slice // find slice here
Array.prototype.slice // find slice here
```

code2:

```
[].__proto__ == Array.prototype; // true
var arr = new Array();
arr.__proto__ == Array.prototype; // true
```

conclusion:

as u can see `[]` and `new Array()` create a new instance of Array.And they all get the prototype functions from `Array.prototype`

They are just different instance of Array.so this explain why `[] != []`

:)

Share

Improve this answer

Follow

edited Dec 27, 2016 at 6:32

answered Dec 27, 2016 at 6:14

sammy

437 ● 5 ● 11

2

There is no big difference, they basically do the same thing but doing them in different ways, but read on, look at this statement at W3C:

```
var cars = ["Saab", "Volvo","BMW"];
```

and

```
var cars = new Array("Saab", "Volvo", "BMW");
```

> The two examples above do exactly the same. There is no need to use new Array().
> For simplicity, readability and execution speed, use the first one (the array literal method).

But at the same time, creating new array using `new Array` syntax considered as a bad practice:

> **Avoid new Array()**

> There is no need to use the JavaScript's built-in array constructor new Array().
> Use [] instead.
> These two different statements both create a new empty array named points:

```
var points = new Array();         // Bad
var points = [];                  // Good
```

> These two different statements both create a new array containing 6 numbers:

```
var points = new Array(40, 100, 1, 5, 25, 10); // Bad
var points = [40, 100, 1, 5, 25, 10];          // Good
```

> The **new** keyword only complicates the code. It can also produce some unexpected results:

```
var points = new Array(40, 100);  // Creates an array with two elements (40 and
100)
```

> What if I remove one of the elements?

```
var points = new Array(40);       // Creates an array with 40 undefined
elements !!!!!
```

So basically not considered as the best practice, also there is one minor difference there, you can pass length to `new Array(length)` like this, which also not a recommended way.

Share

Improve this answer

Follow

edited May 27, 2017 at 0:49

answered May 26, 2017 at 16:23

Alireza

105k ● 27 ● 277 ● 173

Hi Alireza, are these copied and pasted from somewhere? Please add a link to the page where the text is copied from. See this help center page for details. Thank you. – Pang May 29, 2017 at 1:50

Useful for `new Array(40).fill(123)` – user128511 Dec 19, 2019 at 1:00

2

I've incurred in a weird behaviour using [].

We have Model "classes" with fields initialised to some value. E.g.:

```
require([
  "dojo/_base/declare",
  "dijit/_WidgetBase",
], function(declare, parser, ready, _WidgetBase){

   declare("MyWidget", [_WidgetBase], {
     field1: [],
     field2: "",
     function1: function(),
     function2: function()
   });
});
```

I found that when the fields are initialised with `[]` then it would be shared by all Model objects. Making changes to one affects all others.

This doesn't happen initialising them with `new Array()`. Same for the initialisation of Objects (`{}` vs new `Object()`)

TBH I am not sure if its a problem with the framework we were using (Dojo)

Share Improve this answer Follow

answered Sep 19, 2017 at 15:26

algiogia

955 ● 2 ● 14 ● 44

0

I found a difference while using promises. While using array of promises (say arr, initialised as arr=[]), got an error in Promise.all(arr). Whereas when declared as arr = Array(), did not get compilation issues. Hope this helps.

Share Improve this answer Follow

answered Jun 2, 2022 at 4:08

p

pranav karthik

21 ● 3

0

Keep in mind:

Work with new or skip it, there will be no effect on the functioning of the code.

Consider this example:

```
const x = Array(1, 2, 3); //an array defined simply
const y = new Array(1, 2, 3); // an array defined with new keyword


console.log(x); // output: [1, 2, 3]
console.log(y); // output: [1, 2, 3]


console.log(x.length);  // output: 3
console.log(y.length);  // output: 3
```

Run code snippet Expand snippet

Reasons to Use: New

- the new Array() does One more thing then Array() : **You can pass a number to the constructor, and you will get an array of that length**.

```
var myArray = new Array(); // some output
var myArray = []; // will return te same output as above
```

Run code snippet Expand snippet

- Another advantage of using this method is that **it creates the required size of the array beforehand in the stack**.

Reason not to use: New

-Avoid the extra coding, as sometimes code becomes virtually synonymous.

Hope it helps.

Share Improve this answer Follow

answered Mar 15, 2023 at 6:06

Muhammad Ali

1,300 ● 3 ● 20 ● 23

-5

I've found one difference between the two constructions that bit me pretty hard.

Let's say I have:

```
function MyClass(){
  this.property1=[];
  this.property2=new Array();
};
var MyObject1=new MyClass();
var MyObject2=new MyClass();
```

In real life, if I do this:

```
MyObject1.property1.push('a');
MyObject1.property2.push('b');
MyObject2.property1.push('c');
MyObject2.property2.push('d');
```

What I end up with is this:

```
MyObject1.property1=['a','c']
MyObject1.property2=['b']
MyObject2.property1=['a','c']
MyObject2.property2=['d']
```

I don't know what the language specification says is supposed to happen, but if I want my two objects to have unique property arrays in my objects, I have to use `new Array()`.

Share Improve this answer Follow

answered Sep 24, 2015 at 0:08

Bucky
89 ● 1 ● 1

This JSFiddle shows that the output is what you'd expect with the array literal `[]` and `new Array()` constructor resulting in one item per array per property. You must have something else going on in your code to end up with the result you show above. – gfullam Sep 24, 2015 at 13:33

Bucky, that doesn't happen for me, in any browser. To get that behavior you'd have to do something like this: `var property1static=[]; function MyClass(){ this.property1=property1static; this.property2=new Array(); };` – Dave Burton Jan 6, 2016 at 18:50

Are you sure you weren't doing `var MyObject2 = MyObject1;`? – Sebastian Simon Sep 3, 2021 at 9:55

-5

Using the Array constructor makes a new array of the desired length and populates each of the indices with undefined, the assigned an array to a variable one creates the indices that you give it info for.

Share Improve this answer Follow

answered Nov 19, 2015 at 16:20

Timothy Trousdale

65 • 1

2 No, array is NOT populated, there are no index/keys inside. For example .forEach would not work. – CFrei Jan 2, 2016 at 13:31

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.