Copy array items into another array

Asked 8 years, 10 months ago Active 18 days ago Viewed 902k times

821

107

I have a JavaScript array `dataArray` which I want to push into a new array `newArray`. Except I don't want `newArray[0]` to be `dataArray`. I want to push in all the items into the new array:

```
var newArray = [];

newArray.pushValues(dataArray1);
newArray.pushValues(dataArray2);
// ...
```

or even better:

```
var newArray = new Array (
   dataArray1.values(),
   dataArray2.values(),
   // ... where values() (or something equivalent) would push the individual values into
the array, rather than the array itself
);
```

So now the new array contains all the values of the individual data arrays. Is there some shorthand like `pushValues` available so I don't have to iterate over each individual `dataArray`, adding the items one by one?

javascript arrays

edited Apr 25 at 11:40
IARI
471 7 22

asked Nov 11 '10 at 15:33
bba
5,266 9 25 25

See this url stackoverflow.com/questions/351409/appending-to-array – Jakir Hossain Jan 10 '16 at 10:50

This should be the answer davidwalsh.name/combining-js-arrays – starikovs Sep 22 '17 at 8:24

1113

Use the concat function, like so:

```
var arrayA = [1, 2];
var arrayB = [3, 4];
var newArray = arrayA.concat(arrayB);
```

The value of `newArray` will be `[1, 2, 3, 4]` (`arrayA` and `arrayB` remain unchanged; `concat` creates and returns a new array for the result).

edited Apr 22 '14 at 18:15
Tom Wadley
107k 1 20 28

answered Nov 11 '10 at 15:37
WiseGuyEh
13.4k 1 16 20

5 A better reference would be: developer.mozilla.org/en/JavaScript/Reference/Global_Objects/… [1]: developer.mozilla.org/en/JavaScript/Reference/Global_Objects/… – Jamund Ferguson Feb 23 '11 at 5:41

82 Slow though: jsperf.com/concat-vs-push-apply/10 – w00t Aug 20 '12 at 19:42

14 I agree that performant execution is very nice. **BUT** isn't concat exactly for **that** purpose to *concat* to arrays? So it should be **standard**. Or is there other better things to do with concat? And it might be slow only because of bad implementation of the JS engine of the browser or wherever you're using it in? It might be fixed one day. I would choose code maintainability over hacky speed optimizations. Hmm .... – Bitterblue Jun 10 '16 at 9:04

2 Also I just benchmarked the situation: concat vs. push.apply. `Google Chrome` : fast (concat = winner), `Opera` : fast (concat = winner), `IE` : slower (concat = winner), `Firefox` : slow (push.apply = winner, yet 10 times slower than Chrome's concat) ... speak of bad JS engine implementation. – Bitterblue Jun 10 '16 at 9:25

9 How is *concatenating* two arrays the accepted answer for how to *push* one into another?! Those are two different operations. – kaqqao Mar 6 '17 at 19:18

622

Provided your arrays are not huge (see caveat below), you can use the `push()` method of the array to which you wish to append values. `push()` can take multiple parameters so you can use its `apply()` method to pass the array of values to be pushed as a list of function parameters. This has the advantage over using `concat()` of adding elements to the array in place rather than creating a new array.

However, it seems that for large arrays (of the order of 100,000 members or more), **this trick can fail**. For such arrays, using a loop is a better approach. See https://stackoverflow.com/a/17368101/96100 for details.

```
var newArray = [];
newArray.push.apply(newArray, dataArray1);
newArray.push.apply(newArray, dataArray2);
```

You might want to generalize this into a function:

```
function pushArray(arr, arr2) {
    arr.push.apply(arr, arr2);
}
```

... or add it to `Array` 's prototype:

```
Array.prototype.pushArray = function(arr) {
    this.push.apply(this, arr);
};

var newArray = [];
newArray.pushArray(dataArray1);
newArray.pushArray(dataArray2);
```

... or emulate the original `push()` method by allowing multiple parameters using the fact that `concat()` , like `push()` , allows multiple parameters:

```
Array.prototype.pushArray = function() {
    this.push.apply(this, this.concat.apply([], arguments));
};

var newArray = [];
newArray.pushArray(dataArray1, dataArray2);
```

Here's a loop-based version of the last example, suitable for large arrays and all major browsers, including IE <= 8:

```
Array.prototype.pushArray = function() {
    var toPush = this.concat.apply([], arguments);
    for (var i = 0, len = toPush.length; i < len; ++i) {
        this.push(toPush[i]);
    }
};
```

Community ♦
1 1

Tim Down
261k 59 393 478

8 note: `newArray.push.apply(newArray, dataArray1);` gives the same as `Array.prototype.push.applay(newArray,dataArra1);` – user669677 Jul 5 '13 at 11:02

Is this compatible with older browsers? – Damien Ó Ceallaigh Apr 6 '18 at 0:53

developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/… – Damien Ó Ceallaigh Apr 6 '18 at 0:53

1 @DamienÓCeallaigh: Yes. This is all compatible with browsers going back to IE 6 and even earlier. – Tim Down Apr 9 '18 at 11:16

This is the epitome of poor programming practices. `Array.prototype.concat` is not bad, rather it is simply misunderstood and sometimes misused. Under these circumstances, it is only misunderstood, but not misused. – Jack Giffin Aug 15 '18 at 10:23

356

I will add one more "future-proof" reply

In ECMAScript 6, you can use the Spread syntax:

```
let arr1 = [0, 1, 2];
let arr2 = [3, 4, 5];
arr1.push(...arr2);

console.log(arr1)
```

Run code snippet Expand snippet

Spread syntax is not yet included in all major browsers. For the current compatibility, see this (continuously updated) compatibility table.

You can, however, use spread syntax with Babel.js.

edit:

See Jack Giffin's reply below for more comments on performance. It seems concat is still better and faster than spread operator.

edited Jun 11 at 9:20 community wiki

4 You can also use the spread operator if you are using TypeScript. If you target ES5, it will compile to `newArray.apply(newArray, dataArray1)` . – AJ Richardson Feb 26 '16 at 21:28

5 Note: if you need the result in a third array (thus not modifying arr1, as the initial question seemed to require), you can do newArray = [...arr1, ...arr2] – dim Sep 29 '17 at 19:42

Oh I didn't know that. Thanks. I have added an edit comment. – Karel Bílek Aug 17 '18 at 5:47

Similar to how concat would function then, with the bonus of being persistent (mutating the original array). – Rob Sep 1 '18 at 8:37

@robertmylne The spread operator obviously does not modify the original array, instead creating a new copy of all the arrays contents desparsed. – Jack Giffin Nov 20 '18 at 2:44

138

Found an elegant way from MDN

```
var vegetables = ['parsnip', 'potato'];
var moreVegs = ['celery', 'beetroot'];

// Merge the second array into the first one
// Equivalent to vegetables.push('celery', 'beetroot');
Array.prototype.push.apply(vegetables, moreVegs);

console.log(vegetables); // ['parsnip', 'potato', 'celery', 'beetroot']
```

Or you can use the `spread operator` feature of ES6:

```
let fruits = [ 'apple', 'banana'];
const moreFruits = [ 'orange', 'plum' ];

fruits.push(...moreFruits); // ["apple", "banana", "orange", "plum"]
```

edited Jul 9 '16 at 16:18

answered Sep 10 '15 at 21:12
Believe2014
2,915 2 19 33

1 This is not what the question was asking for. The question is asking for a way to create a new array each time, not modify an old array. – Jack Giffin Aug 15 '18 at 9:16

13

```
var a=new Array('a','b','c');
var b=new Array('d','e','f');
var d=new Array('x','y','z');
var c=a.concat(b,d)
```

Does that solve your problem ?

answered Nov 11 '10 at 15:37
Sébastien VINCENT
1,001 6 11

12

The following seems simplest to me:

```
var newArray = dataArray1.slice();
newArray.push.apply(newArray, dataArray2);
```

As "push" takes a variable number of arguments, you can use the `apply` method of the `push` function to push all of the elements of another array. It constructs a call to push using its first argument ("newArray" here) as "this" and the elements of the array as the remaining arguments.

The `slice` in the first statement gets a copy of the first array, so you don't modify it.

Update If you are using a version of javascript with slice available, you can simplify the `push` expression to:

```
newArray.push(...dataArray2)
```

edited Mar 19 '18 at 13:58

answered Apr 11 '16 at 4:32
shaunc
2,921 21 42

1 Is also mentioned here at MDN as an example for "Merging two arrays" – Wilt Apr 27 '18 at 15:48

9

```
var dataArray1 = [1, 2];
var dataArray2 = [3, 4, 5];
var newArray = [ ];
Array.prototype.push.apply(newArray, dataArray1); // newArray = [1, 2]
Array.prototype.push.apply(newArray, dataArray2); // newArray = [1, 2, 3, 4, 5]
console.log(JSON.stringify(newArray)); // Outputs: [1, 2, 3, 4, 5]
```

Run code snippet Expand snippet

If you have ES6 syntax:

```
var dataArray1 = [1, 2];
var dataArray2 = [3, 4, 5];
var newArray = [ ];
newArray.push(...dataArray1); // newArray = [1, 2]
newArray.push(...dataArray2); // newArray = [1, 2, 3, 4, 5]
console.log(JSON.stringify(newArray)); // Outputs: [1, 2, 3, 4, 5]
```

Run code snippet Expand snippet

edited Sep 23 '17 at 1:14

answered Jul 3 '17 at 1:36
Stephen Quan
12.4k 3 53 56

8

The function below doesn't have an issue with the length of arrays and performs better than all suggested solutions:

```
function pushArray(list, other) {
    var len = other.length;
    var start = list.length;
    list.length = start + len;
    for (var i = 0; i < len; i++ , start++) {
        list[start] = other[i];
    }
```

```
       Name           |   ops/sec    |  ± %  | runs sampled
for loop and push     |      177506  |  0.92 | 63
Push Apply            |      234280  |  0.77 | 66
spread operator       |      259725  |  0.40 | 67
set length and for loop |    284223  |  0.41 | 66
```

where

for loop and push is:

```
    for (var i = 0, l = source.length; i < l; i++) {
        target.push(source[i]);
    }
```

Push Apply:

```
target.push.apply(target, source);
```

spread operator:

```
    target.push(...source);
```

and finally the 'set length and for loop' is the above function

answered May 20 '17 at 15:01
Panos Theof
1,142 1 19 23

This question is looking for a way to create a new array each time, not modify an existing array. – Jack Giffin Aug 15 '18 at 9:17

7

With JavaScript ES6, you can use the ... operator as a spread operator which will essentially convert the array into values. Then, you can do something like this:

```
const newArray = [
  ...myArray,
  ...moreData,
];
```

While the syntax is concise, I do not know how this works internally and what the performance implications are on large arrays.

edited Oct 25 '18 at 15:23

answered Jun 29 '16 at 18:24
Ryan H.
2,991 24 32

2 If you take a look at how babel converts it, you'll see that it should not be any slower than using `Array.push.apply` technique. – emil.c Dec 11 '17 at 9:19

1 @JackGiffin I was just referring to what Ryan mentioned that he doesn't know how it works internally and what are performance implications, I wasn't actually suggesting this approach. In any case, you've done a very good job on your answer, nice research, it's always good to know such details. – emil.c Nov 22 '18 at 19:50

6

Here's the ES6 way

```
var newArray = [];
let dataArray1 = [1,2,3,4]
let dataArray2 = [5,6,7,8]
newArray = [...dataArray1, ...dataArray2]
console.log(newArray)
```

Run code snippet Expand snippet

> The above method is good to go for most of the cases and the cases it is not please consider `concat` , like you have hundred thousands of items in arrays.

```
    let dataArray1 = [1,2,3,4]
```

Run code snippet Expand snippet

edited Nov 20 '18 at 6:46

answered Jun 15 '18 at 5:10
Black Mamba
3,928 2 31 52

5

Research And Results

For the facts, a performance test at jsperf and checking some things in the console are performed. For the research, the website irt.org is used. Below is a collection of all these sources put together plus an example function at the bottom.

Method	Concat	slice&push.apply	push.apply x2	ForLoop	Spread
mOps/Sec	179	104	76	81	28
Sparse arrays kept sparse	YES!	Only the sliced array (1st arg)	no	Maybe[2]	no
Support (source)	MSIE 4 NNav 4	MSIE 5.5 NNav 4.06	MSIE 5.5 NNav 4.06	MSIE 4 NNav 3	Edge 12 ~~MSIE NNav~~
Array-like acts like an array	no	Only the pushed array (2nd arg)	YES!	YES!	If have iterator[1]

[1] If the array-like object does not have a *Symbol.iterator* property, then trying to spread it will throw an exception.
[2] Depends on the code. The following example code "YES" preserves sparseness.

```
function mergeCopyTogether(inputOne, inputTwo){
   var oneLen = inputOne.length, twoLen = inputTwo.length;
   var newArr = [], newLen = newArr.length = oneLen + twoLen;
   for (var i=0, tmp=inputOne[0]; i !== oneLen; ++i) {
        tmp = inputOne[i];
        if (tmp !== undefined || inputOne.hasOwnProperty(i)) newArr[i] = tmp;
    }
    }
    return newArr;
}
```

As seen above, I would argue that Concat is almost always the way to go for both performance and the ability to retain the sparseness of spare arrays. Then, for array-likes (such as DOMNodeLists like `document.body.children`), I would recommend using the for loop because it is both the 2nd most performant and the only other method that retains sparse arrays. Below, we will quickly go over what is meant by sparse arrays and array-likes to clear up confusion.

The Future

At first, some people may think that this is a fluke and that browser vendors will eventually get around to optimizing Array.prototype.push to be fast enough to beat Array.prototype.concat. WRONG! Array.prototype.concat will always be faster (in principle at least) because it is a simple copy-n-paste over the data. Below is a simplified persuado-visual diagram of what a 32-bit array implementation might look like (please note real implementations are a LOT more complicated)

```
Byte ║ Data here
═════╬═══════════
0x00 ║ int nonNumericPropertiesLength = 0x00000000
0x01 ║ ibid
0x02 ║ ibid
0x03 ║ ibid
0x00 ║ int length = 0x00000001
0x01 ║ ibid
0x02 ║ ibid
0x03 ║ ibid
0x00 ║ int valueIndex = 0x00000000
0x01 ║ ibid
0x02 ║ ibid
0x03 ║ ibid
0x00 ║ int valueType = JS_PRIMITIVE_NUMBER
0x01 ║ ibid
0x02 ║ ibid
0x03 ║ ibid
0x00 ║ uintptr_t valuePointer = 0x38d9eb60 (or whereever it is in memory)
0x01 ║ ibid
0x02 ║ ibid
0x03 ║ ibid
```

As seen above, all you need to do to copy something like that is almost as simple as copying it byte for byte. With

An alternative way to think of it is this. The source array one is a large stack of papers stapled together. The source array two is also another large stack of papers. Would it be faster for you to

1. Go to the store, buy enough paper needed for a copy of each source array. Then put each source array stacks of paper through a copy-machine and staple the resulting two copies together.
2. Go to the store, buy enough paper for a single copy of the first source array. Then, copy the source array to the new paper by hand, ensuring to fill in any blank sparse spots. Then, go back to the store, buy enough paper for the second source array. Then, go through the second source array and copy it while ensuring no blank gaps in the copy. Then, staple all the copied papers together.

In the above analogy, option #1 represents Array.prototype.concat while #2 represents Array.prototype.push.apply. Let us test this out with a similar JSperf differing only in that this one tests the methods over sparse arrays, not solid arrays. One can find it right here.

Therefore, I rest my case that the future of performance for this particular use case lies not in Array.prototype.push, but rather in Array.prototype.concat.

Clarifications

Spare Arrays

When certain members of the array are simply missing. For example:

```
// This is just as an example. In actual code,
// do not mix different types like this.
var mySparseArray = [];
mySparseArray[0] = "foo";
mySparseArray[10] = undefined;
mySparseArray[11] = {};
mySparseArray[12] =  10;
mySparseArray[17] = "bar";
console.log("Length:   ", mySparseArray.length);
console.log("0 in it:  ", 0 in mySparseArray);
console.log("arr[0]:   ", mySparseArray[0]);
console.log("10 in it: ", 10 in mySparseArray);
console.log("arr[10]   ", mySparseArray[10]);
console.log("20 in it: ", 20 in mySparseArray);
console.log("arr[20]:  ", mySparseArray[20]);
```

Run code snippet Expand snippet

Alternatively, javascript allows you to initialize spare arrays easily.

```
var mySparseArray = ["foo",,,,,,,,,,undefined,{},10,,,,,"bar"];
```

Array-Likes

An array-like is an object that has at least a `length` property, but was not initialized with `new Array` or `[]` ; For example, the below objects are classified as array-like.

```
{0: "foo", 1: "bar", length:2}
```

```
document.body.children
```

```
new Uint8Array(3)
```

- This is array-like because although it's a(n) (typed) array, coercing it to an array changes the constructor.

```
(function(){return arguments})()
```

Observe what happens using a method that does coerce array-likes into arrays like slice.

Show code snippet

- **NOTE:** It is bad practice to call slice on function arguments because of performance.

Observe what happens using a method that does **not** coerce array-likes into arrays like concat.

Show code snippet

edited Aug 18 at 22:00

answered Aug 15 '18 at 14:43
Jack Giffin
1,533 1 15 30

1

```
let a = [2, 4, 6, 8, 9, 15]

function transform(a) {
    let b = ['4', '16', '64']
    a.forEach(function(e) {
        b.push(e.toString());
    });
    return b;
}

transform(a)

[ '4', '16', '64', '2', '4', '6', '8', '9', '15' ]
```

edited Jan 9 '18 at 7:02

answered Jun 7 '17 at 14:33

KARTHIKEYAN.A

6,450 3 52 60

1 Please don't just post code as an answer. Explain what the code does and how it solves the problem. – Patrick Hund Jun 7 '17 at 18:28

0

If you want to modify the original array, you can spread and push:

```
var source = [1, 2, 3];
var range = [5, 6, 7];
var length = source.push(...range);

console.log(source); // [ 1, 2, 3, 5, 6, 7 ]
console.log(length); // 6
```

If you want to make sure only items of the same type go in the `source` array (not mixing numbers and strings for example), then use TypeScript.

```
/**
 * Adds the items of the specified range array to the end of the source array.
 * Use this function to make sure only items of the same type go in the source array.
 */
function addRange<T>(source: T[], range: T[]) {
```

answered Aug 18 at 21:15
orad
6,350 14 54 88

-2

instead of push() function use concat function for IE. example,

```
var a=a.concat(a,new Array('amin'));
```

answered Dec 27 '13 at 1:06
user1911703
329 4 6 20

both are very IE compatible – Jack Giffin Aug 16 '18 at 8:38

-3

This is a working code and it works fine:

```
var els = document.getElementsByTagName('input'), i;
var invnum = new Array();
var k = els.length;
for(i = 0; i < k; i++){invnum.push(new Array(els[i].id,els[i].value))}
```

answered Dec 12 '14 at 12:31
user4354031
21

protected by T J Jan 4 '16 at 17:12

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).