Merge/flatten an array of arrays

Asked 7 years, 3 months ago Active 13 days ago Viewed 570k times

953

I have a JavaScript array like:

```
[["$6"], ["$12"], ["$25"], ["$25"], ["$18"], ["$22"], ["$10"]]
```

How would I go about merging the separate inner arrays into one like:

```
["$6", "$12", "$25", ...]
```

260

javascript arrays flatten

edited Apr 11 at 21:46

Alexander Abakumov
5,699 5 50 79

asked Jun 2 '12 at 18:53

Andy
6,139 12 34 51

gist.github.com/Nishchit14/4c6a7349b3c778f7f97b912629a9f228 This link describe ES5 & ES6 flatten – Nishchit Dhanani Nov 17 '16 at 6:42

15 All of the solutions that use `reduce + concat` are O((N^2)/2) where as a accepted answer (just one call to `concat`) would be at most O(N*2) on a bad browser and O(N) on a good one. Also Denys solution is optimized for the actual question and upto 2x faster than the single `concat`. For the `reduce` folks it's fun to feel cool writing tiny code but for example if the array had 1000 one element subarrays all the reduce+concat solutions would be doing 500500 operations where as the single concat or simple loop would do 1000 operations. – gman Jul 30 '17 at 15:45

2 Simply user spread operator `[].concat(...array)` – Oleg Mar 30 at 19:17

@gman how is reduce + concat solution O((N^2)/2)? stackoverflow.com/questions/52752666/… mentions a different complexity. – Usman Jun 19 at 12:13

71 Answers

1632

You can use `concat` to merge arrays:

```
var arrays = [
  ["$6"],
  ["$12"],
  ["$25"],
  ["$25"],
  ["$18"],
  ["$22"],
  ["$10"]
];
var merged = [].concat.apply([], arrays);

console.log(merged);
```

Run code snippet Expand snippet

Using the `apply` method of `concat` will just take the second parameter as an array, so the last line is identical to this:

```
var merged2 = [].concat(["$6"], ["$12"], ["$25"], ["$25"], ["$18"], ["$22"], ["$10"]);
```

There is also an experimental `Array.prototype.flat()` method (not yet part of the ECMAScript standard) which you could use to flatten the arrays, although it is only available in Node.js starting with version 11, and not at all in Edge.

```
const arrays = [
      ["$6"],
      ["$12"],
      ["$25"],
      ["$25"],
      ["$18"],
      ["$22"],
      ["$10"]
    ];
const merge3 = arrays.flat(1); //The depth level specifying how deep a nested array
structure should be flattened. Defaults to 1.
console.log(merge3);
```

Run code snippet Expand snippet

edited Jun 11 at 17:32

Stephen Horvath
2,086 1 18 25

answered Jun 2 '12 at 18:56

Gumbo
533k 94 690 779

9 Note that `concat` does not modify the source array, so the `merged` array will remain empty after the call to `concat`. Better to say something like: `merged = merged.concat.apply(merged, arrays);` – Nate Jan 24 '13 at 21:12

62 `var merged = [].concat.apply([], arrays);` seems to work fine to get it on one line. edit: as Nikita's answer already shows. – Sean Mar 15 '13 at 16:37

42 Or `Array.prototype.concat.apply([], arrays)`. – danhbear Jan 16 '14 at 1:03

28 Note: this answer only flattens one level deep. For a recursive flatten, see the answer by @Trindaz. – Phrogz Feb 21 '14 at 14:01

175 Further to @Sean's comment: ES6 syntax makes this super concise: `var merged = [].concat(...arrays)` – Sethi Jul 8 '15 at 13:50

449

+100

Here's a short function that uses some of the newer JavaScript array methods to flatten an n-dimensional array.

```
function flatten(arr) {
  return arr.reduce(function (flat, toFlatten) {
    return flat.concat(Array.isArray(toFlatten) ? flatten(toFlatten) : toFlatten);
  }, []);
}
```

Usage:

```
flatten([[1, 2, 3], [4, 5]]); // [1, 2, 3, 4, 5]
flatten([[[1, [1.1]], 2, 3], [4, 5]]); // [1, 1.1, 2, 3, 4, 5]
```

edited Jul 2 '15 at 17:40

answered Feb 22 '13 at 17:41

Noah Freitas
13.8k 8 41 64

9 What's the memory usage profile for this solution? Looks like it creates a lot of intermediate arrays during the tail recursion.... – JBRWilkinson Jul 28 '15 at 18:28

6 @ayjay, it's the starting accumulator value for the reduce function, what mdn calls the initial value. In this case it's the value of `flat` in the first call to the anonymous function passed to `reduce`. If it is not specified, then the first call to `reduce` binds the first value out of the array to `flat`, which would eventually result in `1` being bound to `flat` in both the examples. `1.concat` is not a function. – Noah Freitas Nov 18 '15 at 2:32

17 Or in a shorter, sexier form: `const flatten = (arr) => arr.reduce((flat, next) => flat.concat(next), []);` – Tsvetomir Tsonev Aug 8 '16 at 14:18

10 Riffing on @TsvetomirTsonev and Noah's solutions for arbitrary nesting: `const flatten = (arr) => arr.reduce((flat, next) => flat.concat(Array.isArray(next) ? flatten(next) : next), []);` – Will Aug 16 '17 at 14:24

301

There is a confusingly hidden method, which constructs a new array without mutating the original one:

```
var oldArray = [[1],[2,3],[4]];
var newArray = Array.prototype.concat.apply([], oldArray);
console.log(newArray); // [ 1, 2, 3, 4 ]
```

Run code snippet Expand snippet

edited May 17 at 3:32

answered Mar 13 '13 at 22:12

Nikita Volkov

34.3k 10 77 156

11 In CoffeeScript this is `[].concat([[1],[2,3],[4]]...)` – amoebe Jun 21 '13 at 18:49

8 @amoebe your answer gives `[[1],[2,3],[4]]` as a result. The solution that @Nikita gives is correct for CoffeeScript as well as JS. – Ryan Kennedy Jul 31 '13 at 17:22

5 Ahh, I found your error. You have an extra pair of square brackets in your notation, should be `[].concat([1],[2,3],[4],...)`. – Ryan Kennedy Aug 6 '13 at 15:27

21 I think I found your error, too. The `...` are actual code, not some ellipsis dots. – amoebe Feb 23 '14 at 20:01

23 @paldepind 1. You should get yourself acquainted at least with the definition of functional programming before making nonsense accusations. My code does not mutate values, that's why it's functional. Period. 2. JS's native functions get executed by the VM, which in most cases provides an optimized

compiles to machine code, which is all about mutating memory cells, so by your definition every language is imperative, which is simply absurd. – Nikita Volkov Mar 25 '14 at 15:20

178

It can be best done by javascript reduce function.

```
var arrays = [["$6"], ["$12"], ["$25"], ["$25"], ["$18"], ["$22"], ["$10"], ["$0"],
["$15"],["$3"], ["$75"], ["$5"], ["$100"], ["$7"], ["$3"], ["$75"], ["$5"]];

arrays = arrays.reduce(function(a, b){
     return a.concat(b);
}, []);
```

Or, with ES2015:

```
arrays = arrays.reduce((a, b) => a.concat(b), []);
```

[js-fiddle]

[Mozilla docs]

edited Jun 22 '17 at 7:39

Daniel Wolf
6,654 5 35 62

answered Aug 19 '13 at 5:58

user2668376
2,342 1 10 4

1 @JohnS Actually reduce feeds onto concat a single (array) value per turn. Proof? take out reduce() and see if it works. reduce() here is an alternative to Function.prototype.apply() – André Werlang Feb 23 '15 at 14:55

3 I would have used reduce as well but one interesting thing I learned from this snippet is that **most of the time you don't need to pass a initialValue** :) – José F. Romaniello Aug 21 '15 at 19:44

2 The problem with reduce is that the array cannot be empty, so you will need to add an extra validation. – calbertts Jul 5 '16 at 22:57

4 @calbertts Just pass the initial value `[]` and no further validations are necessary. – user6445533 Aug 17 '16 at 12:38

5 Since you use ES6, you could also use the spread-operator as array literal. `arrays.reduce((flatten, arr) => [...flatten, ...arr])` – Putzi San Feb 19 '18 at 14:51

68

Most of the answers here don't work on huge (e.g. 200 000 elements) arrays, and even if they do, they're slow. polkovnikov.ph's answer has the best performance, but it doesn't work for deep flattening.

Here is **the fastest solution, which works also on arrays with multiple levels of nesting**:

```
const flatten = function(arr, result = []) {
  for (let i = 0, length = arr.length; i < length; i++) {
    const value = arr[i];
    if (Array.isArray(value)) {
      flatten(value, result);
    } else {
      result.push(value);
    }
  }
  return result;
};
```

Examples

Huge arrays

```
flatten(Array(200000).fill([1]));
```

It handles huge arrays just fine. On my machine this code takes about 14 ms to execute.

Nested arrays

```
flatten(Array(2).fill(Array(2).fill(Array(2).fill([1]))));
```

It works with nested arrays. This code produces `[1, 1, 1, 1, 1, 1, 1, 1]`.

Arrays with different levels of nesting

```
flatten([1, [1], [[1]]]);
```

It doesn't have any problems with flattening arrays like this one.

Michał Perłakowski
50k 16 114 131

Except your huge array is pretty flat. This solution won't work for deeply nested arrays. No recursive solution will. In fact no browser but Safari has TCO right now, so no recursive algorithm will perform well. – nitely Mar 4 '17 at 16:01

@nitely But in what real-world situation would you have arrays with more than a few levels of nesting? – Michał Perłakowski Jun 6 '17 at 13:45

2 Usually, when the array is generated out of user generated content. – nitely Jun 6 '17 at 17:05

How slow was the other way using apply / spread? – 0xcaff Jun 18 '17 at 15:18

2 what's the O notation complexity of this? – George Katsanos Sep 22 '17 at 7:05

54

Update: it turned out that this solution doesn't work with large arrays. It you're looking for a better, faster solution, check out this answer.

```
function flatten(arr) {
  return [].concat(...arr)
}
```

Is simply expands `arr` and passes it as arguments to `concat()`, which merges all the arrays into one. It's equivalent to `[].concat.apply([], arr)`.

You can also try this for deep flattening:

```
function deepFlatten(arr) {
  return flatten(           // return shalowly flattened array
    arr.map(x=>             // with each x in array
      Array.isArray(x)      // is x an array?
        ? deepFlatten(x)    // if yes, return deeply flattened x
        : x                 // if no, return just x
    )
  )
}
```

- Spread operator
- Arrow functions

Side note: methods like `find()` and arrow functions are not supported by all browsers, but it doesn't mean that you can't use these features right now. Just use Babel — it transforms ES6 code into ES5.

edited May 23 '17 at 12:18
Community ♦
1 1

answered Jan 24 '16 at 19:31
Michał Perłakowski
50k 16 114 131

Because almost all the replies here misuse `apply` in this way, I removed my comments from yours. I still think using `apply` /spread this way is bad advise, but since no one cares... – user6445533 Aug 17 '16 at 13:17

@LUH3417 It's not like that, I really appreciate your comments. It turned out you're right -- this solution indeed doesn't work with large arrays. I posted another answer which works fine even with arrays of 200 000 elements. – Michał Perłakowski Aug 17 '16 at 14:55

If you are using ES6, you can reducer further to: `const flatten = arr => [].concat(...arr)` – Eruant Sep 14 '17 at 10:49

What do you mean "does not work with large arrays"? How large? What happens? – GEMI Feb 6 '18 at 8:12

3 @GEMI For example trying to flatten a 500000-element array using this method gives "RangeError: Maximum call stack size exceeded". – Michał Perłakowski Feb 6 '18 at 15:21

50

You can use Underscore:

```
var x = [[1], [2], [3, 4]];

_.flatten(x); // => [1, 2, 3, 4]
```

answered Jun 2 '12 at 18:58
Todd Yandell
13.9k 2 45 37

19 Whoah, did someone add Perl to JS? :-) – JBRWilkinson May 31 '13 at 14:39

7 For the sake of completeness regarding @styfle's comment: learnyouahaskell.com – Simon A. Eugster Feb 4 '15 at 11:38

48

There's a new native ECMA 2019 method called flat to do this exactly.

```
const arr1 = [1, 2, [3, 4]];
arr1.flat();
// [1, 2, 3, 4]

const arr2 = [1, 2, [3, 4, [5, 6]]];
arr2.flat();
// [1, 2, 3, 4, [5, 6]]

// Flatten 2 levels deep
const arr3 = [2, 2, 5, [5, [5, [6]], 7]];
arr3.flat(2);
// [2, 2, 5, 5, 5, [6], 7];

// Flatten all levels
const arr4 = [2, 2, 5, [5, [5, [6]], 7]];
arr4.flat(Infinity);
// [2, 2, 5, 5, 5, 6, 7];
```

edited Aug 13 at 13:03

str
21.1k 9 62 91

answered Jun 23 '18 at 22:00

Alister
14k 4 30 26

3 It's a shame this isn't even on the first page of answers. This feature is available in Chrome 69 and Firefox 62 (and Node 11 for those working in the backend) – Matt M. Oct 28 '18 at 4:30

And there's developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/… – gsc Dec 7 '18 at 10:22

2 -1; nope, this isn't part of ECMAScript 2018. It's still just a proposal that hasn't made it to any ECMAScript spec. – Mark Amery Jan 20 at 15:22

1 I think now we can consider this .. because now it's part of standard (2019) .. can we revisit the performance part of this once ? – Yuvaraj Feb 16 at 12:01

Seems it's not yet supported by any Microsoft browser though (at least at the time I write this comment) – Laurent S. Aug 22 at 14:42

39

`concatMap` (or `flatMap`) is exactly what we need in this situation.

```
// concat :: ([a],[a]) -> [a]
const concat = (xs,ys) =>
  xs.concat (ys)

// concatMap :: (a -> [b]) -> [a] -> [b]
const concatMap = f => xs =>
  xs.map(f).reduce(concat, [])

// id :: a -> a
const id = x =>
  x

// flatten :: [[a]] -> [a]
const flatten =
  concatMap (id)

// your sample data
const data =
  [["$6"], ["$12"], ["$25"], ["$25"], ["$18"], ["$22"], ["$10"]]

console.log (flatten (data))
```

Run code snippet Expand snippet

foresight

And yes, you guessed it correctly, it only flattens **one** level, which is exactly how it *should* work

Imagine some data set like this

```
// Player :: (String, Number) -> Player
const Player = (name,number) =>
  [ name, number ]

// team :: ( . Player) -> Team
const Team = (...players) =>
  players
```

```
  [ teamA, teamB ]

// sample data
const teamA =
  Team (Player ('bob', 5), Player ('alice', 6))

const teamB =
  Team (Player ('ricky', 4), Player ('julian', 2))

const game =
  Game (teamA, teamB)

console.log (game)
// [ [ [ 'bob', 5 ], [ 'alice', 6 ] ],
//   [ [ 'ricky', 4 ], [ 'julian', 2 ] ] ]
```

Run code snippet Expand snippet

Ok, now say we want to print a roster that shows all the players that will be participating in `game` …

```
const gamePlayers = game =>
  flatten (game)

gamePlayers (game)
// => [ [ 'bob', 5 ], [ 'alice', 6 ], [ 'ricky', 4 ], [ 'julian', 2 ] ]
```

If our `flatten` procedure flattened nested arrays too, we'd end up with this garbage result …

```
const gamePlayers = game =>
  badGenericFlatten(game)

gamePlayers (game)
// => [ 'bob', 5, 'alice', 6, 'ricky', 4, 'julian', 2 ]
```

rollin' deep, baby

That's not to say sometimes you don't want to flatten nested arrays, too – only that shouldn't be the default behaviour.

```
// concat :: ([a],[a]) -> [a]
const concat = (xs,ys) =>
  xs.concat (ys)

// concatMap :: (a -> [b]) -> [a] -> [b]
const concatMap = f => xs =>
  xs.map(f).reduce(concat, [])

// id :: a -> a
const id = x =>
  x

// flatten :: [[a]] -> [a]
const flatten =
  concatMap (id)

// deepFlatten :: [[a]] -> [a]
const deepFlatten =
  concatMap (x =>
    Array.isArray (x) ? deepFlatten (x) : x)

// your sample data
const data =
  [0, [1, [2, [3, [4, 5], 6]]], [7, [8]], 9]

console.log (flatten (data))
// [ 0, 1, [ 2, [ 3, [ 4, 5 ], 6 ] ], 7, [ 8 ], 9 ]

console.log (deepFlatten (data))
// [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

Run code snippet Expand snippet

There. Now you have a tool for each job – one for squashing one level of nesting, `flatten`, and one for obliterating all nesting `deepFlatten`.

Maybe you can call it `obliterate` or `nuke` if you don't like the name `deepFlatten`.

Don't iterate twice !

Using a trusty combinator I'm calling `mapReduce` helps keep the iterations to a minium; it takes a mapping function `m :: a -> b`, a reducing function `r :: (b,a) ->b` and returns a new reducing function - this combinator is at the heart of *transducers*; if you're interested, [I've written other answers about them]

```
// mapReduce = (a -> b, (b,a) -> b, (b,a) -> b)
const mapReduce = (m,r) =>
  (acc,x) => r (acc, m (x))

// concatMap :: (a -> [b]) -> [a] -> [b]
const concatMap = f => xs =>
  xs.reduce (mapReduce (f, concat), [])

// concat :: ([a],[a]) -> [a]
const concat = (xs,ys) =>
  xs.concat (ys)

// id :: a -> a
const id = x =>
  x

// flatten :: [[a]] -> [a]
const flatten =
  concatMap (id)

// deepFlatten :: [[a]] -> [a]
const deepFlatten =
  concatMap (x =>
    Array.isArray (x) ? deepFlatten (x) : x)

// your sample data
const data =
  [ [ [ 1, 2 ],
      [ 3, 4 ] ],
    [ [ 5, 6 ],
      [ 7, 8 ] ] ]

console.log (flatten (data))
// [ [ 1. 2 ], [ 3, 4 ], [ 5, 6 ], [ 7, 8 ] ]

console.log (deepFlatten (data))
// [ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

edited Sep 28 '17 at 7:21

answered Aug 18 '16 at 19:34

user633183

77.8k 21 151 192

Frequently, when I see your replies I want to withdraw mine, because they have become worthless. Great answer! `concat` itself doesn't blow up the stack, only `...` and `apply` does (along with very large arrays). I didn't see it. I just feel terrible right now. – user6445533 Aug 18 '16 at 22:39

@LUH3417 you shouldn't feel terrible. You also provide good answers with well-written explanations. Everything here is a learning experience – I often make mistakes and write bad answers too. I revisit them in a couple months and think "wtf was I thinking?" and end up completely revising things. No answer is perfect. We're all at some point on a learning curve ^_^ – user633183 Aug 19 '16 at 2:13

1 Please note that `concat` in Javascript has a different meaning than in Haskell. Haskell's `concat ( [[a]] -> [a] )` would be called `flatten` in Javascript and is implemented as `foldr (++) []` (Javascript: `foldr(concat) ([])` assuming curried functions). Javascript's `concat` is a weird append (`(++)` in Haskell), which can handle both `[a] -> [a] -> [a]` and `a -> [a] -> [a]` . – user6445533 Feb 3 '17 at 9:36

I guess a better name were `flatMap` , because that is exactly what `concatMap` is: The `bind` instance of the `list` monad. `concatpMap` is implemented as `foldr ((++) . f) []` . Translated into Javascript: `const flatMap = f => foldr(comp(concat) (f)) ([])` . This is of course similar to your implementation without `comp` . – user6445533 Feb 3 '17 at 9:36

what's the complexity of that algorithm? – George Katsanos Sep 22 '17 at 7:08

31

A solution for the more general case, when you may have some non-array elements in your array.

```
function flattenArrayOfArrays(a, r){
    if(!r){ r = []}
    for(var i=0; i<a.length; i++){
        if(a[i].constructor == Array){
            r.concat(flattenArrayOfArrays(a[i], r));
        }else{
            r.push(a[i]);
        }
    }
    return r;
}
```

edited Mar 28 '17 at 1:28

aug

6,949 5 48 72

answered Jun 6 '13 at 4:41

Trindaz

6,561 16 69 99

1 Added as a method of arrays: `Object.defineProperty(Array.prototype,'flatten',{value:function(r){for(var a=this,i=0,r=r||[];i<a.length;++i)if(a[i]!=null)a[i] instanceof Array?a[i].flatten(r):r.push(a[i]);return r}});` – Phrogz Feb 21 '14 at 13:59

This will break if we manually pass in the second argument. For example, try this: `flattenArrayOfArrays (arr, 10)` or this `flattenArrayOfArrays(arr, [1,[3]]);` - those second arguments are added to the output. – Om Shankar Aug 7 '15 at 19:10

2 this answer is not complete! the result of the recursion is never assigned anywhere, so result of recursions are lost – r3wt May 31 '16 at 18:29

1 @r3wt went ahead and added a fix. What you need to do is make sure that the current array you are maintaining, `r`, will actually concatenate the results from the recursion. – aug Mar 28 '17 at 1:29

27

Another ECMAScript 6 solution in functional style:

Declare a function:

```
const flatten = arr => arr.reduce(
  (a, b) => a.concat(Array.isArray(b) ? flatten(b) : b), []
);
```

and use it:

```
flatten( [1, [2,3], [4,[5,[6]]]] ) // -> [1,2,3,4,5,6]
```

```
const flatten = arr => arr.reduce(
        (a, b) => a.concat(Array.isArray(b) ? flatten(b) : b), []
      );


console.log( flatten([1, [2,3], [4,[5],[6,[7,8,9],10],11],[12],13]) )
```

Run code snippet Expand snippet

Consider also a native function Array.prototype.flat() (proposal for ES6) available in last releases of modern browsers. *Thanks to @(Константин Ван) and @(Mark Amery) mentioned it in the comments.*

```
[1, 2, [3, 4]].flat();                // -> [1, 2, 3, 4]

[1, 2, [3, 4, [5, 6]]].flat();        // -> [1, 2, 3, 4, [5, 6]]

[1, 2, [3, 4, [5, 6]]].flat(2);       // -> [1, 2, 3, 4, 5, 6]

[1, 2, [3, 4, [5, 6]]].flat(Infinity); // -> [1, 2, 3, 4, 5, 6]
```

```
let arr = [1, 2, [3, 4]];

console.log( arr.flat() );

arr =  [1, 2, [3, 4, [5, 6]]];

console.log( arr.flat() );
console.log( arr.flat(1) );
console.log( arr.flat(2) );
console.log( arr.flat(Infinity) );
```

Run code snippet Expand snippet

edited Jul 4 at 5:40

answered Feb 1 '16 at 10:49

diziaq

3,003 9 29 41

3 This is nice and neat but I think you have done an ES6 overdose. There is no need for the outer function to be an arrow-function. I would stick with the arrow-function for the reduce callback but flatten itself ought to be a normal function. – Stephen Simpson Feb 23 '16 at 15:17

3 @StephenSimpson but is there a need for the outer function to be a *non*-arrow-function ? *"flatten itself ought to be a normal function"* – by "normal" you mean "non-arrow", but why? *Why* use an arrow function in the call to reduce then? Can you supply your line of reasoning? – user633183 Oct 3 '17 at 19:44

@naomik My reasoning is that it is unnecessary. It's mainly a matter of style; I should have much clearer in my comment. There is no major coding reason to use one or the other. However, the function is easier to see and read as non-arrow. The inner function is useful as an arrow function as it is more compact (and no context created of course). Arrow functions are great for creating compact easy to read function and avoiding this confusion. However, they can actually make it more difficult to read when a non-arrow would suffice. Others may disagree though! – Stephen Simpson Oct 4 '17 at 9:29

26

What about using `reduce(callback[, initialValue])` method of `JavaScript 1.8`

```
list.reduce((p,n) => p.concat(n),[]);
```

Would do the job.

edited Oct 29 '18 at 13:11

answered Apr 30 '13 at 11:40

rab
3,326 1 24 38

8 `[[1], [2,3]].reduce( (a,b) => a.concat(b), [] )` is more sexy. – golopot Jul 16 '16 at 3:12

5 Don't need second argument in our case simple `[[1], [2,3]].reduce( (a,b) => a.concat(b))` – Umair Ahmed Dec 20 '16 at 13:40

25

To flatten an array of single element arrays, you don't need to import a library, a simple loop is both the simplest and most efficient solution :

```
for (var i = 0; i < a.length; i++) {
  a[i] = a[i][0];
}
```

To downvoters: please read the question, don't downvote because it doesn't suit your very different problem. This solution is both the fastest and simplest for the asked question.

edited Aug 8 '16 at 7:20

answered Jun 2 '12 at 18:56

Denys Séguret
292k 61 614 621

4 I would say, "Don't look for things more cryptic." ^^ – user166390 Jun 2 '12 at 19:03

4 I mean... when I see people advising to use a library for that... that's crazy... – Denys Séguret Jun 2 '12 at 19:05

4 It doesn't really matter how cryptic it is. This code "flattens" this `['foo', ['bar']]` to `['f', 'bar']` . – jonschlinkert Feb 9 '14 at 0:06

20

```
const common = arr.reduce((a, b) => [...a, ...b], [])
```

edited Aug 27 '18 at 8:53

answered Jun 25 '18 at 7:04

YairTawil

1,433 8 15

14

Please note: When `Function.prototype.apply` (`[].concat.apply([], arrays)`) or the spread operator (`[].concat(...arrays)`) is used in order to flatten an array, both can cause stack overflows for large arrays, because every argument of a function is stored on the stack.

Here is a stack-safe implementation in functional style that weighs up the most important requirements against one another:

- reusability
- readability
- conciseness
- performance

```
// small, reusable auxiliary functions:

const foldl = f => acc => xs => xs.reduce(uncurry(f), acc); // aka reduce

const uncurry = f => (a, b) => f(a) (b);

const concat = xs => y => xs.concat(y);


// the actual function to flatten an array - a self-explanatory one-line:

const flatten = xs => foldl(concat) ([]) (xs);

// arbitrary array sizes (until the heap blows up :D)
```

```
console.log(flatten(xs));


// Deriving a recursive solution for deeply nested arrays is trivially now


// yet more small, reusable auxiliary functions:

const map = f => xs => xs.map(apply(f));

const apply = f => a => f(a);

const isArray = Array.isArray;


// the derived recursive function:

const flattenr = xs => flatten(map(x => isArray(x) ? flattenr(x) : x) (xs));

const ys = [1,[2,[3,[4,[5],6,],7],8],9];

console.log(flattenr(ys));
```

Run code snippet Expand snippet

As soon as you get used to small arrow functions in curried form, function composition and higher order functions, this code reads like prose. Programming then merely consists of putting together small building blocks that always work as expected, because they don't contain any side effects.

edited Aug 23 '16 at 16:34 answered Aug 17 '16 at 21:00

user6445533

1 Haha. Totally respect your answer, although reading functional programming like this is still like reading Japanese character by character to me (English speaker). – Tarwin Stroh-Spijer Nov 3 '16 at 20:55

2 If you find yourself implementing features of language A in language B not as a part of project with the sole goal of doing exactly this then someone somewhere had taken a wrong turn. Could it be you? Just going with `const flatten = (arr) => arr.reduce((a, b) => a.concat(b), []);` saves you visual garbage *and* explanation to your teammates why you need 3 extra functions *and* some function calls too. – Daerdemandt Feb 20 '17 at 16:29

3 @Daerdemandt But if you write it as separate functions, you will probably be able to reuse them in other code. – Michał Perłakowski Jun 18 '17 at 16:08

10

ES6 One Line Flatten

See lodash flatten, underscore flatten (shallow `true`)

```
function flatten(arr) {
  return arr.reduce((acc, e) => acc.concat(e), []);
}
```

or

```
function flatten(arr) {
  return [].concat.apply([], arr);
}
```

Tested with

```
test('already flatted', () => {
  expect(flatten([1, 2, 3, 4, 5])).toEqual([1, 2, 3, 4, 5]);
});

test('flats first level', () => {
  expect(flatten([1, [2, [3, [4]], 5]])).toEqual([1, 2, [3, [4]], 5]);
});
```

ES6 One Line Deep Flatten

See lodash flattenDeep, underscore flatten

```
function flattenDeep(arr) {
  return arr.reduce((acc, e) => Array.isArray(e) ? acc.concat(flattenDeep(e)) :
acc.concat(e), []);
}
```

```
test('already flatted', () => {
  expect(flattenDeep([1, 2, 3, 4, 5])).toEqual([1, 2, 3, 4, 5]);
});

test('flats', () => {
  expect(flattenDeep([1, [2, [3, [4]], 5]])).toEqual([1, 2, 3, 4, 5]);
});
```

answered Mar 22 '18 at 22:00

zurfyx
14.8k 11 79 114

Your 2nd example is better written as `Array.prototype.concat.apply([], arr)` because you create an extra array just to get to the `concat` function. Runtimes may or may not optimize it away when they run it, but accessing the function on the prototype doesn't look any uglier than this already is in any case. – Mörre Jul 14 '18 at 20:58

8

A Haskellesque approach

```
function flatArray([x,...xs]){
  return x ? [...Array.isArray(x) ? flatArray(x) : [x], ...flatArray(xs)] : [];
}

var na = [[1,2],[3,[4,5]],[6,7,[[[8],9]]],10];
    fa = flatArray(na);
console.log(fa);
```

Run code snippet Expand snippet

answered Jun 1 '17 at 15:49

Redu
13.9k 2 26 40

2 Why doesn't this answer have more upvotes?!! – monners Aug 9 '17 at 6:22

8

ES6 way:

```
const flatten = arr => arr.reduce((acc, next) => acc.concat(Array.isArray(next) ?
flatten(next) : next), [])

const a = [1, [2, [3, [4, [5]]]]]
console.log(flatten(a))
```

Run code snippet Expand snippet

ES5 way for `flatten` function with ES3 fallback for N-times nested arrays:

```
var flatten = (function() {
  if (!!Array.prototype.reduce && !!Array.isArray) {
    return function(array) {
      return array.reduce(function(prev, next) {
        return prev.concat(Array.isArray(next) ? flatten(next) : next);
      }, []);
    };
  } else {
    return function(array) {
      var arr = [];
      var i = 0;
      var len = array.length;
      var target;

      for (; i < len; i++) {
        target = array[i];
        arr = arr.concat(
          (Object.prototype.toString.call(target) === '[object Array]') ?
flatten(target) : target
        );
      }

      return arr;
    };
  }
}());
```

Run code snippet Expand snippet

edited Feb 20 '18 at 10:50

answered May 10 '17 at 12:11

Artem Gavrysh
126 1 3

ES6 way works for me. – Axel Feb 23 '18 at 12:48

7

If you only have arrays with 1 string element:

```
[["$6"], ["$12"], ["$25"], ["$25"]].join(',').split(',');
```

will do the job. Bt that specifically matches your code example.

answered Jun 2 '12 at 19:18

Florian Salihovic
3,387 2 13 21

3 Whoever down voted, please explain why. I was searching for a decent solution and of all the solutions I liked this one the most. – Anonymous Apr 18 '14 at 22:34

2 @Anonymous I didn't downvote it since it technically meets the requirements of the question, but it's likely because this is a pretty poor solution that isn't useful in the general case. Considering how many better solutions there are here, I'd never recommend someone go with this one as it breaks the moment you have more than one element, or when they're not strings. – Thor84no May 1 '15 at 15:53

2 I love this solution =) – Huei Tan Aug 10 '15 at 10:01

2 It doesn't handle just arrays with 1 string elements, it also handles this array `['$4', ["$6"], ["$12"], ["$25"], ["$25", "$33", ['$45']]].join(',').split(',')` – alucic Jan 25 '16 at 1:17

I discovered this method on my own, however knew it must have already been documented somewhere, my search ended here. The drawback with this solution is, it coerces numbers, booleans etc to strings, try `[1,4, [45, 't', ['e3', 6]]].toString().split(',')` ---- or ----- `[1,4, [45, 't', ['e3', 6], false]].toString().split(',')` – Sudhansu Choudhary Jul 16 at 14:42

7

```
var arrays = [["a"], ["b", "c"]];
Array.prototype.concat.apply([], arrays);

// gives ["a", "b", "c"]
```

(I'm just writing this as a separate answer, based on comment of @danhbear.)

answered Feb 16 '15 at 7:32

VasiliNovikov
5,739 2 27 40

7

I recommend a space-efficient generator function:

```
function* flatten(arr) {
  if (!Array.isArray(arr)) yield arr;
  else for (let el of arr) yield* flatten(el);
}

// Example:
console.log(...flatten([1,[2,[3,[4]]]])); // 1 2 3 4
```

Run code snippet Expand snippet

If desired, create an array of flattened values as follows:

```
let flattened = [...flatten([1,[2,[3,[4]]]])]; // [1, 2, 3, 4]
```

answered May 22 '17 at 2:32

le_m
12.6k 3 42 50

I like this approach. Similar to stackoverflow.com/a/35073573/1175496 , but uses the spread operator `...` to iterate through the generator. – The Red Pea Oct 27 '18 at 14:49

6

I would rather transform the whole array, as-is, to a string, but unlike other answers, would do that using `JSON.stringify` and not use the `toString()` method, which produce an unwanted result.

With that `JSON.stringify` output, all that's left is to remove all brackets, wrap the result with start & ending brackets yet again, and serve the result with `JSON.parse` which brings the string back to "life".

- Can handle infinite nested arrays without any speed costs.
- Can rightly handle Array items which are strings containing commas.

```
var arr = ["abc",[[[6]]],["3,4"],"2"];

var s = "[" + JSON.stringify(arr).replace(/\[|]/g,'') +"]";
var flattened = JSON.parse(s);

console.log(flattened)
```

Run code snippet **Expand snippet**

- *Only for multidimensional Array of Strings/Numbers (not Objects)*

edited Nov 2 '16 at 9:02

answered Nov 1 '16 at 21:38

vsync
54.7k 38 176 245

Your solution is incorrect. It will contain the comma when flattening inner arrays `["345", "2", "3,4", "2"]` instead of separating each of those values to separate indices – pizza-r0b Nov 1 '16 at 22:05

@realseanp - you misunderstood the value in that Array item. I intentionally put that comma as a value and not as an Array delimiter comma to emphasize the power of my solution above all others, which would output `"3,4"` . – vsync Nov 1 '16 at 23:20

1 I did misunderstand – pizza-r0b Nov 2 '16 at 0:20

that seems definitely the fastest solution I've seen for this; are you aware of any pitfalls @vsync (except the fact it looks a bit hacky of course - treating nested arrays as strings:D) – George Katsanos Sep 26 '17 at 19:58

@GeorgeKatsanos - This method will not work for array items (and nested items) which are not of Primitive value, for example an item which points to a DOM element – vsync Sep 27 '17 at 8:24

6

You can use `Array.flat()` with `Infinity` for any depth of nested array.

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


let flatten = arr.flat(Infinity)

console.log(flatten)
```

Run code snippet Expand snippet

check here for browser compatibility

answered Jan 20 at 14:41

Code Maniac
23.7k 4 15 41

5

It looks like this looks like a job for RECURSION!

- Handles multiple levels of nesting
- Handles empty arrays and non array parameters
- Has no mutation
- Doesn't rely on modern browser features

Code:

```
var flatten = function(toFlatten) {
  var isArray = Object.prototype.toString.call(toFlatten) === '[object Array]';

  if (isArray && toFlatten.length > 0) {
    var head = toFlatten[0];
    var tail = toFlatten.slice(1);
```

```
  }
};
```

Usage:

```
flatten([1,[2,3],4,[[5,6],7]]);
// Result: [1, 2, 3, 4, 5, 6, 7]
```

edited Apr 1 '14 at 3:35

answered Apr 1 '14 at 2:36

Jai
776 8 15

careful, `flatten(new Array(15000).fill([1]))` throws `Uncaught RangeError: Maximum call stack size exceeded` and freezed my devTools for 10 seconds – pietrovismara May 4 '17 at 22:42

@pietrovismara, my test is around 1s. – Ivan Yan Jun 6 '17 at 3:46

5

I have done it using recursion and closures

```
function flatten(arr) {

  var temp = [];

  function recursiveFlatten(arr) {
    for(var i = 0; i < arr.length; i++) {
      if(Array.isArray(arr[i])) {
        recursiveFlatten(arr[i]);
      } else {
        temp.push(arr[i]);
      }
    }
  }
  recursiveFlatten(arr);
  return temp;
}
```

answered Apr 29 '15 at 12:10

1 Simple and sweet, this answer works better than the accepted answer. It flattens deeply nested levels to, not just the first level – Om Shankar Aug 7 '15 at 18:58

1 AFAIK that is lexical scoping and not a closure – dashambles Feb 3 '16 at 20:49

@dashambles is correct - the difference is that if it was a closure you would return the inner function to the outside and when the outer function is finished you can still use the inner function to access its scope. Here the lifetime of the outer function is longer than that of the inner function so a "closure" is never created. – Mörre Jul 14 '18 at 20:26

5

I was goofing with ES6 Generators the other day and wrote this gist. Which contains...

```
function flatten(arrayOfArrays=[]){
  function* flatgen() {
    for( let item of arrayOfArrays ) {
      if ( Array.isArray( item )) {
        yield* flatten(item)
      } else {
        yield item
      }
    }
  }

  return [...flatgen()];
}

var flatArray = flatten([[1, [4]],[2],[3]]);
console.log(flatArray);
```

Basically I'm creating a generator that loops over the original input array, if it finds an array it uses the yield* operator in combination with recursion to continually flatten the internal arrays. If the item is not an array it just yields the single item. Then using the ES6 Spread operator (aka splat operator) I flatten out the generator into a new array instance.

I haven't tested the performance of this, but I figure it is a nice simple example of using generators and the yield* operator.

But again, I was just goofing so I'm sure there are more performant ways to do this.

answered Jan 28 '16 at 22:42

ashwell

1 Similar answer (using generator and yield delegation), but in PHP format – The Red Pea Oct 27 '18 at 3:40

5

just the best solution without lodash

```
let flatten = arr => [].concat.apply([], arr.map(item => Array.isArray(item) ?
flatten(item) : item))
```

answered Sep 7 '16 at 10:20

Vlad Ankudinov

1,084 1 8 16

4

That's not hard, just iterate over the arrays and merge them:

```
var result = [], input = [["$6"], ["$12"], ["$25"], ["$25"], ["$18"]];

for (var i = 0; i < input.length; ++i) {
    result = result.concat(input[i]);
}
```

answered Jun 2 '12 at 18:56

Niko

24k 7 76 105

4

The logic here is to convert input array to string and remove all brackets([]) and parse output to array. I'm using ES6 template feature for this.

```
var x=[1, 2, [3, 4, [5, 6,[7], 9],12, [12, 14]]];

var y=JSON.parse(`[${JSON.stringify(x).replace(/\[|]/g,'')}]`);

console.log(y)
```

R Santosh Reddy
59 6

1 Please edit your answer to include some explanation. Code-only answers do very little to educate future SO readers. Your answer is in the moderation queue for being low-quality. – mickmackusa Apr 21 '17 at 16:39

You are a god hahaha best answer ever – Waldemar Neto Apr 25 '17 at 2:04

this is the fastest and cleverest way to do it. I was using this to avoid recursion and easily rebuild the array, but yours is 3% faster. Way to go :) `const flatten = function (A) { return A .toString() .split(',') .reduce( (a,c) => { let i = parseFloat(c); c = (!Number.isNaN(i)) ? i : c; a.push(c); return a; }, []);` – Ady Ngom Sep 26 '17 at 12:14

3

Nowadays the best and easy way to do this is joining and spliting the array like this.

```
var multipleArrays = [["$6","$Demo"], ["$12",["Multi","Deep"]], ["$25"], ["$25"],
["$18"], ["$22"], ["$10"], ["$0"], ["$15"],["$3"], ["$75"], ["$5"], ["$100"], ["$7"],
["$3"], ["$75"], ["$5"]]

var flattened = multipleArrays.join().split(",")
```

This solution works with multiple levels and is also oneliner.

DEMO

EDIT for ECMAScript 6

Since ECMAScript 6 has been standardized, you can change the operation `[].concat.apply([], arrays);` for `[].concat(...arrays);`

```
var flattened = [].concat(...input);
```

DEMO

EDIT Most Efficient solution

The most efficient way to solve the problem is using a loop. You can compare the "ops/sec" velocity here

```
        for (var j=0; j<current.length; ++j)
            flattened.push(current[j]);
    }
```

DEMO

Hope It Helps

edited Jan 26 '16 at 12:41

answered Jan 26 '16 at 11:32

Sapikelio

1,972 1 10 36

1 It doesn't work when array contains strings which contain commas, for example: `[[","]].join().split(",")` doesn't give desired result. – Michał Perłakowski Jan 26 '16 at 11:36

1 2 3 next

protected by Machavity Jan 2 '18 at 20:04

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?