mdn web docs

Array.prototype.concat()

Baseline Widely available

The `concat()` method of `Array` instances is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

Try it

JavaScript Demo: Array.prototype.concat()

```
const array1 = ["a", "b", "c"];
const array2 = ["d", "e", "f"];
const array3 = array1.concat(array2);

console.log(array3);
// Expected output: Array ["a", "b", "c", "d", "e", "f"]
```

Run

Reset

Syntax

JS

```
concat()
concat(value1)
concat(value1, value2)
concat(value1, value2, /* …, */ valueN)
```

Parameters

`value1`, ..., `valueN` Optional

Arrays and/or values to concatenate into a new array. If all `valueN` parameters are omitted, `concat` returns a shallow copy of the existing array on which it is called. See the description below for more details.

Return value

A new `Array` instance.

Description

The `concat` method creates a new array. The array will first be populated by the elements in the object on which it is called. Then, for each argument, its value will be concatenated into the array — for normal objects or primitives, the argument itself will become an element of the final array; for arrays or array-like objects with the property `Symbol.isConcatSpreadable` set to a truthy value, each element of the argument will be independently added to the final array. The `concat` method does not recurse into nested array arguments.

The `concat()` method is a copying method. It does not alter `this` or any of the arrays provided as arguments but instead returns a shallow copy that contains the same elements as the ones from the original arrays.

The `concat()` method preserves empty slots if any of the source arrays is sparse.

The `concat()` method is generic. The `this` value is treated in the same way as the other arguments (except it will be converted to an object first), which means plain objects will be directly prepended to the resulting array, while array-like objects with truthy `[Symbol.isConcatSpreadable]` will be spread into the resulting array.

Examples

Concatenating two arrays

The following code concatenates two arrays:

```js
const letters = ["a", "b", "c"];
const numbers = [1, 2, 3];

const alphaNumeric = letters.concat(numbers);
console.log(alphaNumeric);
// results in ['a', 'b', 'c', 1, 2, 3]
```

Concatenating three arrays

The following code concatenates three arrays:

```js
const num1 = [1, 2, 3];
const num2 = [4, 5, 6];
const num3 = [7, 8, 9];

const numbers = num1.concat(num2, num3);

console.log(numbers);
// results in [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Concatenating values to an array

The following code concatenates three values to an array:

```js
const letters = ["a", "b", "c"];

const alphaNumeric = letters.concat(1, [2, 3]);

console.log(alphaNumeric);
// results in ['a', 'b', 'c', 1, 2, 3]
```

Concatenating nested arrays

The following code concatenates nested arrays and demonstrates retention of references:

```js
const num1 = [[1]];
const num2 = [2, [3]];

const numbers = num1.concat(num2);

console.log(numbers);
// results in [[1], 2, [3]]

// modify the first element of num1
num1[0].push(4);

console.log(numbers);
// results in [[1, 4], 2, [3]]
```

Concatenating array-like objects with Symbol.isConcatSpreadable

`concat` does not treat all array-like objects as arrays by default — only if `Symbol.isConcatSpreadable` is set to a truthy value (e.g., `true`).

```js
const obj1 = { 0: 1, 1: 2, 2: 3, length: 3 };
const obj2 = { 0: 1, 1: 2, 2: 3, length: 3, [Symbol.isConcatSpreadable]:
true };
console.log([0].concat(obj1, obj2));
// [ 0, { '0': 1, '1': 2, '2': 3, length: 3 }, 1, 2, 3 ]
```

Using concat() on sparse arrays

If any of the source arrays is sparse, the resulting array will also be sparse:

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

Calling concat() on non-array objects

If the `this` value is not an array, it is converted to an object and then treated in the same way as the arguments for `concat()`. In this case the return value is

always a plain new array.

```JS
console.log(Array.prototype.concat.call({}, 1, 2, 3)); // [{}, 1, 2, 3]
console.log(Array.prototype.concat.call(1, 2, 3)); // [ [Number: 1], 2, 3 ]
const arrayLike = {
  [Symbol.isConcatSpreadable]: true,
  length: 2,
  0: 1,
  1: 2,
  2: 99, // ignored by concat() since length is 2
};
console.log(Array.prototype.concat.call(arrayLike, 3, 4)); // [1, 2, 3, 4]
```

Specifications

Specification
ECMAScript® 2026 Language Specification # sec-array.prototype.concat

Browser compatibility

Report problems with this compatibility data • View data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android	WebView on iOS	Deno	Node.js
`concat`	✓ 1	✓ 12	✓ 1	✓ 4	✓ 1	✓ 18	✓ 4	✓ 10.1	✓ 1	✓ 1	✓ 1	✓ 1	✓ 1	✓ 0.10

Tip: you can click/tap on a cell for more information.

✓ Full support

See also

- Polyfill of `Array.prototype.concat` in `core-js` with fixes and implementation of modern behavior like `Symbol.isConcatSpreadable` support
- es-shims polyfill of `Array.prototype.concat`
- Indexed collections guide
- `Array`
- `Array.prototype.push()`
- `Array.prototype.unshift()`
- `Array.prototype.splice()`
- `String.prototype.concat()`
- `Symbol.isConcatSpreadable`