mdn web docs

Array.prototype.forEach()

Baseline Widely available

The `forEach()` method of `Array` instances executes a provided function once for each array element.

Try it

JavaScript Demo: Array.prototype.forEach()

```
1 const array1 = ["a", "b", "c"];
2
3 array1.forEach((element) => console.log(element));
4
5 // Expected output: "a"
6 // Expected output: "b"
7 // Expected output: "c"
8
```

Run

Reset

Syntax

JS

```
forEach(callbackFn)
forEach(callbackFn, thisArg)
```

Parameters

`callbackFn`

A function to execute for each element in the array. Its return value is discarded. The function is called with the following arguments:

> `element`
>
> The current element being processed in the array.
>
> `index`
>
> The index of the current element being processed in the array.
>
> `array`
>
> The array `forEach()` was called upon.

`thisArg` Optional

A value to use as `this` when executing `callbackFn`. See iterative methods.

Return value

None (`undefined`).

Description

The `forEach()` method is an iterative method. It calls a provided `callbackFn` function once for each element in an array in ascending-index order. Unlike `map()`, `forEach()` always returns `undefined` and is not chainable. The typical use case is to execute side effects at the end of a chain. Read the iterative methods section for more information about how these methods work in general.

`callbackFn` is invoked only for array indexes which have assigned values. It is not invoked for empty slots in sparse arrays.

The `forEach()` method is generic. It only expects the `this` value to have a `length` property and integer-keyed properties.

There is no way to stop or break a `forEach()` loop other than by throwing an exception. If you need such behavior, the `forEach()` method is the wrong tool.

Early termination may be accomplished with looping statements like `for`, `for...of`, and `for...in`. Array methods like `every()`, `some()`, `find()`, and `findIndex()` also stops iteration immediately when further iteration is not necessary.

`forEach()` expects a synchronous function — it does not wait for promises. Make sure you are aware of the implications while using promises (or async functions) as `forEach` callbacks.

JS

```
const ratings = [5, 4, 5];
let sum = 0;

const sumFunction = async (a, b) => a + b;

ratings.forEach(async (rating) => {
  sum = await sumFunction(sum, rating);
});

console.log(sum);
// Naively expected output: 14
// Actual output: 0
```

To run a series of asynchronous operations sequentially or concurrently, see promise composition.

Examples

Converting a for loop to forEach

JS

```
const items = ["item1", "item2", "item3"];
const copyItems = [];
```

```
// before
for (let i = 0; i < items.length; i++) {
  copyItems.push(items[i]);
}

// after
items.forEach((item) => {
  copyItems.push(item);
});
```

Printing the contents of an array

> **Note:** In order to display the content of an array in the console, you can use `console.table()`, which prints a formatted version of the array.
>
> The following example illustrates an alternative approach, using `forEach()`.

The following code logs a line for each element in an array:

JS

```
const logArrayElements = (element, index /*, array */) => {
  console.log(`a[${index}] = ${element}`);
};

// Notice that index 2 is skipped, since there is no item at
// that position in the array.
[2, 5, , 9].forEach(logArrayElements);
// Logs:
// a[0] = 2
// a[1] = 5
// a[3] = 9
```

Using thisArg

The following (contrived) example updates an object's properties from each entry in the array:

JS

```
class Counter {
  constructor() {
    this.sum = 0;
    this.count = 0;
  }
  add(array) {
    // Only function expressions have their own this bindings.
    array.forEach(function countEntry(entry) {
      this.sum += entry;
      ++this.count;
    }, this);
  }
}

const obj = new Counter();
obj.add([2, 5, 9]);
console.log(obj.count); // 3
console.log(obj.sum); // 16
```

Since the `thisArg` parameter (`this`) is provided to `forEach()`, it is passed to `callback` each time it's invoked. The callback uses it as its `this` value.

> **Note:** If passing the callback function used an arrow function expression, the `thisArg` parameter could be omitted, since all arrow functions lexically bind the `this` value.

An object copy function

The following code creates a copy of a given object.

There are different ways to create a copy of an object. The following is just one way and is presented to explain how `Array.prototype.forEach()` works by using `Object.*` utility functions.

JS

```
const copy = (obj) => {
  const copy = Object.create(Object.getPrototypeOf(obj));
  const propNames = Object.getOwnPropertyNames(obj);
  propNames.forEach((name) => {
```

```
    const desc = Object.getOwnPropertyDescriptor(obj, name);
    Object.defineProperty(copy, name, desc);
  });
  return copy;
};

const obj1 = { a: 1, b: 2 };
const obj2 = copy(obj1); // obj2 looks like obj1 now
```

Flatten an array

The following example is only here for learning purpose. If you want to flatten an array using built-in methods, you can use `Array.prototype.flat()`.

JS

```
const flatten = (arr) => {
  const result = [];
  arr.forEach((item) => {
    if (Array.isArray(item)) {
      result.push(...flatten(item));
    } else {
      result.push(item);
    }
  });
  return result;
};

// Usage
const nested = [1, 2, 3, [4, 5, [6, 7], 8, 9]];
console.log(flatten(nested)); // [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Using the third argument of callbackFn

The `array` argument is useful if you want to access another element in the array, especially when you don't have an existing variable that refers to the array. The following example first uses `filter()` to extract the positive values and then uses `forEach()` to log its neighbors.

JS

```
const numbers = [3, -1, 1, 4, 1, 5];
numbers
```

```
  .filter((num) => num > 0)
  .forEach((num, idx, arr) => {
    // Without the arr argument, there's no way to easily access the
    // intermediate array without saving it to a variable.
    console.log(arr[idx - 1], num, arr[idx + 1]);
  });
// undefined 3 1
// 3 1 4
// 1 4 1
// 4 1 5
// 1 5 undefined
```

Using forEach() on sparse arrays

JS

```
const arraySparse = [1, 3, /* empty */, 7];
let numCallbackRuns = 0;

arraySparse.forEach((element) => {
  console.log({ element });
  numCallbackRuns++;
});

console.log({ numCallbackRuns });

// { element: 1 }
// { element: 3 }
// { element: 7 }
// { numCallbackRuns: 3 }
```

The callback function is not invoked for the missing value at index 2.

Calling forEach() on non-array objects

The `forEach()` method reads the `length` property of `this` and then accesses each property whose key is a nonnegative integer less than `length`.

JS

```
const arrayLike = {
  length: 3,
  0: 2,
  1: 3,
```

```
  2: 4,
  3: 5, // ignored by forEach() since length is 3
};
Array.prototype.forEach.call(arrayLike, (x) => console.log(x));
// 2
// 3
// 4
```

Specifications

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

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
`forEach`	✓ 1	✓ 12	✓ 1.5	✓ 9.5	✓ 3	✓ 18	✓ 4	✓ 10.1	✓ 1	✓ 1	✓ 4.4	✓ 1	✓ 1	✓ 0.10

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

✓ Full support

See also

- Polyfill of `Array.prototype.forEach` in `core-js`
- es-shims polyfill of `Array.prototype.forEach`
- Indexed collections guide
- `Array`

- `Array.prototype.find()`
- `Array.prototype.map()`
- `Array.prototype.filter()`
- `Array.prototype.every()`
- `Array.prototype.some()`
- `TypedArray.prototype.forEach()`
- `Map.prototype.forEach()`
- `Set.prototype.forEach()`

Help improve MDN

Was this page helpful to you?

Yes No

Learn how to contribute.

This page was last modified on Mar 14, 2025 by MDN contributors.