Array.prototype.forEach()

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

Try it

JavaScript Demo: Array.forEach()

```
const array1 = ['a', 'b', 'c'];

array1.forEach(element => console.log(element));

// expected output: "a"
// expected output: "b"
// expected output: "c"

```

Run ›

Reset

Syntax

```
// Arrow function
forEach((element) => { /* … */ })
forEach((element, index) => { /* … */ })
forEach((element, index, array) => { /* … */ })

// Callback function
forEach(callbackFn)
forEach(callbackFn, thisArg)

// Inline callback function
forEach(function (element) { /* … */ })
forEach(function (element, index) { /* … */ })
forEach(function (element, index, array) { /* … */ })
forEach(function (element, index, array) { /* … */ }, 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

`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.

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

`forEach()` does not mutate the array on which it is called, but the function provided as `callbackFn` can. Note, however, that the length of the array is saved *before* the first invocation of `callbackFn`. Therefore:

- `callbackFn` will not visit any elements added beyond the array's initial length when the call to `forEach()` began.
- Changes to already-visited indexes do not cause `callbackFn` to be invoked on them again.
- If an existing, yet-unvisited element of the array is changed by `callbackFn`, its value passed to the `callbackFn` will be the value at the time that element gets visited. Deleted elements are not visited.

> ⚠ **Warning:** Concurrent modifications of the kind described above frequently lead to hard-to-understand code and are generally to be avoided (except in special cases).

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.

```
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

Using forEach() on sparse arrays

```
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.

Converting a for loop to forEach

```
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:

```
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:

```
class Counter {
  constructor() {
    this.sum = 0;
    this.count = 0;
  }
  add(array) {
    // Only function expressions will have its own this binding
    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.

```
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
```

Modifying the array during iteration

The following example logs `one`, `two`, `four`.

When the entry containing the value `two` is reached, the first entry of the whole array is shifted off—resulting in all remaining entries moving up one position. Because element `four` is now at an earlier position in the array, `three` will be skipped.

`forEach()` does not make a copy of the array before iterating.

```
const words = ["one", "two", "three", "four"];
words.forEach((word) => {
  console.log(word);
  if (word === "two") {
    words.shift(); //'one' will delete from array
  }
}); // one // two // four

console.log(words); // ['two', 'three', 'four']
```

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()`.

```
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]
```

Calling forEach() on non-array objects

The `forEach()` method reads the `length` property of `this` and then accesses each integer index.

```
const arrayLike = {
  length: 3,
  0: 2,
  1: 3,
  2: 4,
};
Array.prototype.forEach.call(arrayLike, (x) => console.log(x));
// 2
```

```
// 3
// 4
```

Specifications

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

Browser compatibility

Report problems with this compatibility data on GitHub

	Desktop					Mobile			
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS
`forEach`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1.5	✓ Opera 9.5	✓ Safari 3	✓ Chrome 18 Android	✓ Firefox 4 for Android	✓ Opera 10.1 Android	✓ Safari 1 on iOS

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

✓ Full support

See also

- Polyfill of `Array.prototype.forEach` in `core-js`
- `Array.prototype.find()`
- `Array.prototype.findIndex()`
- `Array.prototype.map()`
- `Array.prototype.filter()`
- `Array.prototype.every()`
- `Array.prototype.some()`
- `Map.prototype.forEach()`
- `Set.prototype.forEach()`

Last modified: Dec 13, 2022, by MDN contributors