Array.prototype.forEach()

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

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

arr.forEach(function(element) {
    console.log(element);
});

// a
// b
// c
```

Syntax

```
arr.forEach(function callback(currentValue, index,
array) {
    //your iterator
}[, thisArg]);
```

Parameters

`callback`

Function to execute for each element, taking three arguments:

`currentValue`
The value of the current element being processed in the array.

`index`
The index of the current element being processed in the array.

`array`
The array that `forEach()` is being applied to.

`thisArg` Optional
Value to use as **`this`** (i.e the reference `Object`) when executing `callback`.

Return value

`undefined`.

Description

`forEach()` executes the provided `callback` once for each element present in the array in ascending order. It is not invoked for index properties that have been deleted or are uninitialized (i.e. on sparse arrays).

`callback` is invoked with **three arguments**:

- the **element value**
- the **element index**
- the **array being traversed**

If a `thisArg` parameter is provided to `forEach()`, it will be used as callback's `this` value. Otherwise, the value `undefined` will be used as its `this` value. The `this` value ultimately observable by `callback` is determined according to the usual rules for determining the `this` seen by a function.

The range of elements processed by `forEach()` is set before the first invocation of `callback`. Elements that are appended to the array after the call to `forEach()` begins will not be visited by `callback`. If the values of existing elements of the array are changed, the value passed to `callback` will be the value at the time `forEach()` visits them; elements that are deleted before being visited are not visited. If elements that are already visited are removed (e.g. using `shift()`) during the iteration, later elements will be skipped - see example below.

`forEach()` executes the `callback` function once for each array element; unlike `map()` or `reduce()` it always returns the value `undefined` and is not chainable. The typical use case is to execute side effects at the end of a chain.

`forEach()` does not mutate the array on which it is called (although `callback`, if invoked, may do so).

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. Use a plain loop instead. If you are testing the array elements for a predicate and need a Boolean return value, you can use `every()` or `some()` instead. If available, the new methods `find()` or `findIndex()` can be used for early termination upon true predicates as well.

Examples

Converting from for to forEach

before

```
const items = ['item1', 'item2', 'item3'];
const copy = [];

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

after

```
const items = ['item1', 'item2', 'item3'];
const copy = [];

items.forEach(function(item){
  copy.push(item)
});
```

Printing the contents of an array

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

```
function logArrayElements(element, index, array) {
  console.log('a[' + index + '] = ' + element);
}

// Notice that index 2 is skipped since there is no it
// 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:

```
function Counter() {
  this.sum = 0;
  this.count = 0;
}
Counter.prototype.add = function(array) {
  array.forEach(function(entry) {
    this.sum += entry;
    ++this.count;
  }, this);
  // ^---- Note
};

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

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

> If passing the function argument using an arrow function expressionthe `thisArg` parameter can be omitted as 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 ECMAScript 5 `Object.*` meta property functions.

```
function copy(obj) {
  const copy = Object.create(Object.getPrototypeOf(obj
  const propNames = Object.getOwnPropertyNames(obj);

  propNames.forEach(function(name) {
    const desc = Object.getOwnPropertyDescriptor(obj,
    Object.defineProperty(copy, name, desc);
  });

  return copy;
}

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

If the array is modified during iteration, other elements might be skipped.

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, which results 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.

```
var words = ['one', 'two', 'three', 'four'];
words.forEach(function(word) {
  console.log(word);
  if (word === 'two') {
    words.shift();
  }
});
// one
// two
// four
```

Polyfill

`forEach()` was added to the ECMA-262 standard in the 5th edition; as such it may not be present in other implementations of the standard. You can work around this by inserting the following code at the beginning of your scripts, allowing use of `forEach()` in implementations that don't natively support it. This algorithm is exactly the one specified in ECMA-262, 5th edition, assuming `Object` and `TypeError` have their original values and that `callback.call()` evaluates to the original value of `Function.prototype.call()`.

```
// Production steps of ECMA-262, Edition 5, 15.4.4.18
// Reference: http://es5.github.io/#x15.4.4.18
if (!Array.prototype.forEach) {

  Array.prototype.forEach = function(callback/*, thisA

    var T, k;

    if (this == null) {
      throw new TypeError('this is null or not defined

    }

    // 1. Let O be the result of calling toObject() pa
    // |this| value as the argument.
    var O = Object(this);

    // 2. Let lenValue be the result of calling the Ge
    // method of O with the argument "length".
    // 3. Let len be toUint32(lenValue).
    var len = O.length >>> 0;

    // 4. If isCallable(callback) is false, throw a Ty
    // See: http://es5.github.com/#x9.11
    if (typeof callback !== 'function') {
      throw new TypeError(callback + ' is not a functi
    }

    // 5. If thisArg was supplied, let T be thisArg; e
    // T be undefined.
    if (arguments.length > 1) {
      T = arguments[1];
    }

    // 6. Let k be 0.
    k = 0;

    // 7. Repeat while k < len.
    while (k < len) {

      var kValue;

      // a. Let Pk be ToString(k).
      //    This is implicit for LHS operands of the i
```

```
          // b. Let kPresent be the result of calling the
          //    internal method of O with argument Pk.
          //    This step can be combined with c.
          // c. If kPresent is true, then
          if (k in O) {

            // i. Let kValue be the result of calling the
            // method of O with argument Pk.
            kValue = O[k];

            // ii. Call the Call internal method of callba
            // the this value and argument list containing
            callback.call(T, kValue, k, O);
          }
          // d. Increase k by 1.
          k++;
        }
        // 8. return undefined.
      };
    }
```

Specifications

Specification	Status	Comment
ECMAScript 5.1 (ECMA-262) The definition of 'Array.prototype.forEach' in that specification.	Standard	Initial definition. Implemented in JavaScript 1.6.
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Array.prototype.forEach' in that specification.	Standard	
ECMAScript Latest Draft (ECMA-262) The definition of 'Array.prototype.forEach' in that specification.	Living Standard	

Browser compatibility

Desktop Mobile

Feature	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari
Basic Support	(Yes)	(Yes)	1.5	9	(Yes)	(Yes)

See also

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

Was this article helpful?

Tags: Array ECMAScript 5 JavaScript Method Prototype Reference

Contributors to this page: SphinxKnight, xgqfrms-GitHub, vitaly-zdanevich, malakhim,Sheppy, fscholz, jesselpalmer, chalin, naugtur, 360cid, erikadoyle, KevinNorth, Konrud,getify, oksana-khristenko, jimwmg, Perelandric, nmve, thynctank, kdex, Lumexralph,rwaldron, BenB, jpmedley, preston-geiken, tarnos12, yozlet, eduardoboucas, ibratoev,marktcotton, dggluznube, cutofmyjib, silverwind, tjcrowder, aleksa000777, manu, PointedEars, Maian, Erik1, Anonymous

Last updated by: SphinxKnight, Oct 23, 2017, 3:28:47 AM

See also

Standard built-in objects

`Array`

Properties

`Array.length`

`Array.prototype`

`Array.prototype[@@unscopables]`

Methods

`Array.from()`

`Array.isArray()`

`Array.observe()`

`Array.of()`

`Array.prototype.concat()`

`Array.prototype.copyWithin()`

`Array.prototype.entries()`

`Array.prototype.every()`

`Array.prototype.fill()`

`Array.prototype.filter()`

`Array.prototype.find()`

`Array.prototype.findIndex()`

`Array.prototype.forEach()`

`Array.prototype.includes()`

`Array.prototype.indexOf()`

`Array.prototype.join()`

`Array.prototype.keys()`

`Array.prototype.lastIndexOf()`

`Array.prototype.map()`

`Array.prototype.pop()`

`Array.prototype.push()`

`Array.prototype.reduce()`

`Array.prototype.reduceRight()`

`Array.prototype.reverse()`

`Array.prototype.shift()`

`Array.prototype.slice()`

`Array.prototype.some()`

`Array.prototype.sort()`

`Array.prototype.splice()`

`Array.prototype.toLocaleString()`

`Array.prototype.toSource()`

`Array.prototype.toString()`

`Array.prototype.unshift()`

`Array.prototype.values()`

`Array.prototype[@@iterator]()`

`Array.unobserve()`

`get Array[@@species]`

Inheritance:

`Function`

Properties

Methods

`Object`

Properties

Methods

you@example.com

Sign up now

Web Technologies

Learn Web Development

About MDN

Feedback

About

Contact Us

Donate

Firefox

Other languages: English (US) (en-US)

Terms Privacy Cookies