> Esta página ha sido traducida del inglés por la comunidad. Aprende más y únete a la comunidad de MDN Web Docs.

mdn web docs

Array.prototype.map()

Baseline Widely available

El método `map()` crea un nuevo array con los resultados de la llamada a la función indicada aplicados a cada uno de sus elementos.

```
JS
var numbers = [1, 5, 10, 15];
var doubles = numbers.map(function (x) {
  return x * 2;
});
// doubles is now [2, 10, 20, 30]
// numbers is still [1, 5, 10, 15]

var numbers = [1, 4, 9];
var roots = numbers.map(function (num) {
  return Math.sqrt(num);
});
// roots is now [1, 2, 3]
// numbers is still [1, 4, 9]
```

Sintaxis

```
var nuevo_array = arr.map(function callback(currentValue, index, array) {
    // Elemento devuelto de nuevo_array
}[, thisArg])
```

Parámetros

`callback`

Función que producirá un elemento del nuevo array, recibe tres argumentos:

> `currentValue`
>
> El elemento actual del array que se está procesando.
>
> `index`
>
> El índice del elemento actual dentro del array.
>
> `array`
>
> El array sobre el que se llama `map`.

`thisArg`

Opcional. Valor a usar como `this` al ejecutar `callback`.

Valor devuelto

Un nuevo array en la que cada elemento es el resultado de ejecutar `callback`.

Descripción

`map` llama a la función `callback` provista **una vez por elemento** de un array, en orden, y construye un nuevo array con los resultados. `callback` se invoca sólo para los índices del array que tienen valores asignados; no se invoca en los índices que han sido borrados o a los que no se ha asignado valor.

`callback` es llamada con tres argumentos: el valor del elemento, el índice del elemento, y el objeto array que se está recorriendo.

Si se indica un parámetro `thisArg` a un `map`, se usará como valor de `this` en la función `callback`. En otro caso, se pasará `undefined` como su valor `this`. El valor de `this` observable por el `callback` se determina de acuerdo a las reglas habituales para determinar el valor this visto por una función.

`map` no modifica el array original en el que es llamado (aunque `callback`, si es llamada, puede modificarlo).

El rango de elementos procesado por `map` es establecido antes de la primera invocación del `callback`. Los elementos que sean agregados al array después de que la llamada a `map` comience no serán visitados por el `callback`. Si los elementos existentes del array son modificados o eliminados, su valor pasado al `callback` será el valor en el momento que el `map` lo visita; los elementos que son eliminados no son visitados.

Ejemplos

Procesar un array de números aplicándoles la raíz cuadrada

El siguiente código itera sobre un array de números, aplicándoles la raíz cuadrada a cada uno de sus elementos, produciendo un nuevo array a partir del inicial.

JS

```
var numeros = [1, 4, 9];
var raices = numeros.map(Math.sqrt);
// raices tiene [1, 2, 3]
// numeros aún mantiene [1, 4, 9]
```

Usando map para dar un nuevo formato a los objetos de un array

El siguiente código toma un array de objetos y crea un nuevo array que contiene los nuevos objetos formateados.

JS

```
var kvArray = [
  { clave: 1, valor: 10 },
  { clave: 2, valor: 20 },
  { clave: 3, valor: 30 },
];

var reformattedArray = kvArray.map(function (obj) {
  var rObj = {};
  rObj[obj.clave] = obj.valor;
  return rObj;
});

// reformattedArray es ahora [{1:10}, {2:20}, {3:30}],

// kvArray sigue siendo:
// [{clave:1, valor:10},
//  {clave:2, valor:20},
//  {clave:3, valor: 30}]
```

Mapear un array de números usando una función con un argumento

El siguiente código muestra cómo trabaja `map` cuando se utiliza una función que requiere de un argumento. El argumento será asignado automáticamente a cada elemento del arreglo conforme `map` itera el arreglo original.

JS

```
var numeros = [1, 4, 9];
var dobles = numeros.map(function (num) {
  return num * 2;
});

// dobles es ahora [2, 8, 18]
// numeros sigue siendo [1, 4, 9]
```

Usando `map` de forma genérica

Este ejemplo muestra como usar `map` en `String` para obtener un arreglo de bytes en codifcación ASCII representando el valor de los caracteres:

JS

```
var map = Array.prototype.map;
var valores = map.call("Hello World", function (char) {
  return char.charCodeAt(0);
});
// valores ahora tiene [72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100]
```

Usando `map` genérico con `querySelectorAll`

Este ejemplo muestra como iterar sobre una colección de objetos obtenidos por `querySelectorAll`. En este caso obtenemos todas las opciones seleccionadas en pantalla y se imprimen en la consola:

JS

```
var elems = document.querySelectorAll("select option:checked");
var values = [].map.call(elems, function (obj) {
  return obj.value;
});
```

Usando `map` para invertir una cadena

JS

```
var str = "12345";
[].map
  .call(str, function (x) {
    return x;
  })
  .reverse()
  .join("");

// Salida: '54321'
// Bonus: usa'===' para probar si la cadena original era un palindromo
```

Caso de uso engañoso

(inspirado por este artículo)

Es común utilizar el callback con un argumento (el elemento siendo pasado). Ciertas funciones son también usadas comunmente con un argumento, aún cuando toman argumentos adicionales opcionales. Estos hábitos pueden llevar a comportamientos confusos.

JS

```
// Considera:
["1", "2", "3"].map(parseInt);
// Mientras uno esperaría [1, 2, 3]
// en realidad se obtiene [1, NaN, NaN]
```

```
// parseInt se usa comúnmente con un argumento, pero toma dos.
// El primero es una expresión y el segundo el radix.
// a la función callback, Array.prototype.map pasa 3 argumentos:
// el elemento, el índice y el array.
// El tercer argumento es ignorado por parseInt, pero no el segundo,
// de ahí la posible confusión. Véase el artículo del blog para más detalles

function returnInt(element) {
  return parseInt(element, 10);
}

["1", "2", "3"].map(returnInt); // [1, 2, 3]
// El resultado es un arreglo de números (como se esperaba)

// Un modo más simple de lograr lo de arriba, mientras de evita el "gotcha":
["1", "2", "3"].map(Number); // [1, 2, 3]
```

Polyfill

`map` fue agregado al estandar ECMA-262 en la 5th edición; por lo tanto podría no estar presente en todas la implementaciones del estándar. Puedes sobrepasar esto insertando el siguiente código al comienzo de tus scripts, permitiendo el uso de `map` en implementaciones que no lo soportan de forma nativa. Este algoritmo es exactamente el mismo especificado en ECMA-262, 5th edición, asumiendo `Object`, `TypeError`, y `Array` tienen sus valores originales y que el `callback.call` evalua el valor original de `Function.prototype.call`.

JS

```
// Production steps of ECMA-262, Edition 5, 15.4.4.19
// Reference: http://es5.github.io/#x15.4.4.19
if (!Array.prototype.map) {
  Array.prototype.map = function (callback, thisArg) {
    var T, A, k;

    if (this == null) {
      throw new TypeError(" this is null or not defined");
    }

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

    // 2. Let lenValue be the result of calling the Get internal
    //    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 TypeError exception.
    // See: http://es5.github.com/#x9.11
    if (typeof callback !== "function") {
      throw new TypeError(callback + " is not a function");
    }

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

    // 6. Let A be a new array created as if by the expression new Array(len)
    //    where Array is the standard built-in constructor with that name and
    //    len is the value of len.
    A = new Array(len);

    // 7. Let k be 0
    k = 0;
```

```
    // 8. Repeat, while k < len
    while (k < len) {
      var kValue, mappedValue;

      // a. Let Pk be ToString(k).
      //   This is implicit for LHS operands of the in operator
      // b. Let kPresent be the result of calling the HasProperty 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 Get internal
        //    method of O with argument Pk.
        kValue = O[k];

        // ii. Let mappedValue be the result of calling the Call internal
        //     method of callback with T as the this value and argument
        //     list containing kValue, k, and O.
        mappedValue = callback.call(T, kValue, k, O);

        // iii. Call the DefineOwnProperty internal method of A with arguments
        // Pk, Property Descriptor
        // { Value: mappedValue,
        //   Writable: true,
        //   Enumerable: true,
        //   Configurable: true },
        // and false.

        // In browsers that support Object.defineProperty, use the following:
        // Object.defineProperty(A, k, {
        //   value: mappedValue,
        //   writable: true,
        //   enumerable: true,
        //   configurable: true
        // });

        // For best browser support, use the following:
        A[k] = mappedValue;
      }
      // d. Increase k by 1.
      k++;
    }

    // 9. return A
    return A;
  };
}
```

Especificaciones

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

Compatibilidad con navegadores

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS
`map`	Chrome 1	Edge 12	Firefox 1.5	Opera 9.5	Safari 3	Chrome Android 18	Firefox for Android 4	Opera Android 10.1	Safari on iOS 1

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

Full support

Véase también

- `Array.prototype.forEach()`
- `Map` (inglés) object
- `Array.from()`

Help improve MDN

Was this page helpful to you?

Yes No

Learn how to contribute.

This page was last modified on 5 ago 2023 by MDN contributors.