Array.from()

`Array.from()` 方法从一个类似数组或可迭代的对象中创建一个新的数组实例。

```
const bar = ["a", "b", "c"];
Array.from(bar);
// ["a", "b", "c"]

Array.from('foo');
// ["f", "o", "o"]
```

语法

```
Array.from(arrayLike[, mapFn[, thisArg]])
```

参数

`arrayLike`
想要转换成真实数组的类数组对象或可遍历对象。

`mapFn` 可选
可选参数，如果指定了该参数，则最后生成的数组会经过该函数的加工处理后再返回。

`thisArg` 可选
可选参数，执行 `mapFn` 函数时 `this` 的值。

返回值

一个新的 `Array` 实例

描述

`Array.from()` 允许你从下面两者来创建数组：

- 类数组对象（拥有一个 `length` 属性和若干索引属性的任意对象）
- 可遍历对象（你可以从它身上迭代出若干个元素的对象，比如有 `Map` 和 `Set` 等）

`Array.from()` 方法有一个可选参数 `mapFn`，让你可以在最后生成的数组上再执行一次 `map` 方法后再返回。也就是说 `Array.from(obj, mapFn, thisArg)` 就相当于 `Array.from(obj).map(mapFn, thisArg)`，除非创建的不是可用的中间数组。这对一些数组的子类，如 typed arrays 来说很重要，因为中间数组的值在调用 map() 时需要是适当的类型。

`from()` 的 `length` 属性为 1。

在 ES2015 中，`Class` 语法允许我们为内置类型（比如 `Array`）和自定义类新建子类（比如叫 `SubArray`）。这些子类也会继承父类的静态方法，比如 `SubArray.from()`，调用该方法后会返回子类 `SubArray` 的一个实例，而不是 `Array` 的实例。

示例

Array from a `String`

```
Array.from('foo');
// ["f", "o", "o"]
```

Array from a `Set`

```
let s = new Set(['foo', window]);
Array.from(s);
// ["foo", window]
```

Array from a `Map`

```
let m = new Map([[1, 2], [2, 4], [4, 8]]);
Array.from(m);
// [[1, 2], [2, 4], [4, 8]]
```

Array from an Array-like object (arguments)

```
function f() {
  return Array.from(arguments);
}

f(1, 2, 3);

// [1, 2, 3]
```

Using arrow functions and `Array.from`

```
// Using an arrow function as the map function to
// manipulate the elements
Array.from([1, 2, 3], x => x + x);
// [2, 4, 6]


// Generate a sequence of numbers
// Since the array is initialized with `undefined` on each position,
// the value of `v` below will be `undefined`
Array.from({length: 5}, (v, i) => i);
// [0, 1, 2, 3, 4]
```

Polyfill

ECMA-262 第六版标准添加了 `Array.from` 。有些实现中可能尚未包括在其中。你可以通过在脚本前添加如下内容作为替代方法，以使用未原生支持的 `Array.from` 方法。该算法按照 ECMA-262 第六版中的规范实现，并假定 `Object` 和 `TypeError` 有其本身的值， `callback.call` 对应 `Function.prototype.call` 。此外，鉴于无法使用 Polyfill 实现真正的的迭代器，该实现不支持规范中定义的泛型可迭代元素。

```
// Production steps of ECMA-262, Edition 6, 22.1.2.1
// Reference: https://people.mozilla.org/~jorendorff/es6-draft.html#sec-a
if (!Array.from) {
  Array.from = (function () {
    var toStr = Object.prototype.toString;
    var isCallable = function (fn) {
      return typeof fn === 'function' || toStr.call(fn) === '[object Func
    };
    var toInteger = function (value) {
      var number = Number(value);
      if (isNaN(number)) { return 0; }
      if (number === 0 || !isFinite(number)) { return number; }
      return (number > 0 ? 1 : -1) * Math.floor(Math.abs(number));
    };
    var maxSafeInteger = Math.pow(2, 53) - 1;
    var toLength = function (value) {
      var len = toInteger(value);
      return Math.min(Math.max(len, 0), maxSafeInteger);
```

```
    };

    // The length property of the from method is 1.
    return function from(arrayLike/*, mapFn, thisArg */) {
      // 1. Let C be the this value.
      var C = this;

      // 2. Let items be ToObject(arrayLike).
      var items = Object(arrayLike);

      // 3. ReturnIfAbrupt(items).
      if (arrayLike == null) {
        throw new TypeError("Array.from requires an array-like object - r
      }

      // 4. If mapfn is undefined, then let mapping be false.
      var mapFn = arguments.length > 1 ? arguments[1] : void undefined;
      var T;
      if (typeof mapFn !== 'undefined') {
        // 5. else
        // 5. a If IsCallable(mapfn) is false, throw a TypeError exceptio
        if (!isCallable(mapFn)) {
          throw new TypeError('Array.from: when provided, the second argu
        }

        // 5. b. If thisArg was supplied, let T be thisArg; else let T be
        if (arguments.length > 2) {
          T = arguments[2];
        }
      }

      // 10. Let lenValue be Get(items, "length").
      // 11. Let len be ToLength(lenValue).
      var len = toLength(items.length);

      // 13. If IsConstructor(C) is true, then
      // 13. a. Let A be the result of calling the [[Construct]] internal
      // of C with an argument list containing the single item len.
      // 14. a. Else, Let A be ArrayCreate(len).
      var A = isCallable(C) ? Object(new C(len)) : new Array(len);

      // 16. Let k be 0.
      var k = 0;
```

```
      // 17. Repeat, while k < len… (also steps a - h)
      var kValue;
      while (k < len) {
        kValue = items[k];
        if (mapFn) {
          A[k] = typeof T === 'undefined' ? mapFn(kValue, k) : mapFn.cal
        } else {
          A[k] = kValue;
        }
        k += 1;
      }
      // 18. Let putStatus be Put(A, "length", len, true).
      A.length = len;
      // 20. Return A.
      return A;
    };
  }());
}
```

规范

Specification	Status	Comment
ECMAScript 2015 (6th Edition, ECMA-262) Array.from	ST Standard	Initial definition.
ECMAScript Latest Draft (ECMA-262) Array.from	LS Living Standard	

浏览器兼容性

Desktop Mobile

Feature	Chrome	Firefox (Gecko)	Edge	Internet Explorer	Opera	Safari
Basic support	45	32 (32)	(Yes)	未实现	(Yes)	9.0

相关链接

- `Array`
- `Array.prototype.map()`

- `TypedArray.from()`