Array

JavaScript 数组对象是用于构造数组的全局对象; 它是高阶，类似列表的对象。

创建一个数组

```
let fruits = ["Apple", "Banana"];

console.log(fruits.length);
// 2
```

通过索引访问数组元素

```
let first = fruits[0];
// Apple

let last = fruits[fruits.length - 1];
// Banana
```

遍历一个数组

```
fruits.forEach(function (item, index, array) {
    console.log(item, index);
});
// Apple 0
// Banana 1
```

添加元素到数组的末尾

```
var newLength = fruits.push("Orange");
// ["Apple", "Banana", "Orange"]
```

删除数组末尾的元素

```
let last = fruits.pop();
// remove Orange (from the end)

// ["Apple", "Banana"];
```

删除数组最前面（头部）的元素

```
let first = fruits.shift();
// remove Apple from the front

// ["Banana"];
```

添加到数组的前面（头部）

```
let newLength = fruits.unshift("Strawberry");
// add to the front

// ["Strawberry", "Banana"];
```

找到某个元素在数组中的索引

```
fruits.push('Mango');
// ["Strawberry", "Banana", "Mango"]

let index = fruits.indexOf("Banana");
// 1
```

通过索引删除某个元素

```
let removedItem = fruits.splice(pos, 1);
// this is how to remove an item

// ["Strawberry", "Mango"]
```

从一个索引位置删除多个元素

```
let vegetables = ['Cabbage', 'Turnip', 'Radish', 'Carrot'];
console.log(vegetables);

// ["Cabbage", "Turnip", "Radish", "Carrot"]

let pos = 1, n = 2;

let removedItems = vegetables.splice(pos, n);

// this is how to remove items,
// n defines the number of items to be removed,
// from that position(pos) onward to the end of array.

console.log(vegetables);
// ["Cabbage", "Carrot"] (the original array is changed)

console.log(removedItems);
// ["Turnip", "Radish"]
```

复制一个数组

```
var shallowCopy = fruits.slice();
// this is how to make a copy

// ["Strawberry", "Mango"]
```

语法

```
[element0, element1, ..., elementN]
new Array(element0, element1[, ...[, elementN]])
new Array(arrayLength)
```

元素列 —— **elementN**

`Array` 构造器将会根据给出的元素创建一个 JavaScript 数组，但是当参数仅有一个参数且其为数字时除外，参考下面的 `arrayLength` 。值得注意的是，这种情况仅在使用 `Array` 构造器时出现，使用带方括号的数组字面量则不会。

数组长度 —— **arrayLength**

向 `Array` 构造函数传入一个在 0 到 2^{32}-1 之间的整数，将返回一个以此为长度的数组对象。通过 length 属性可以访问这个值。如果传入的参数不是有效的数值，则抛出 `RangeError` 异常。

描述

数组是类似列表的对象，在原型中提供了遍历以及改变其中元素的很多方法。 数组的长度及其中元素的类型都是不固定的。因为数组的长度可读可写，有时数组中的元素也不是在连续的位置，所以 JavaScript 数组不一定是密集的。通常情况下，这是一些方便的特性；如果这些特性不适用于你的特定使用场景，你可以考虑使用固定类型数组。

有些人认为 you shouldn't use an array as an associative array。在任何情况下, 你可以使用一般的对象来代替，不过这样做会出现需要注意的地方。请看例子： Lightweight JavaScript dictionaries with arbitrary keys .

访问数组里的元素

JavaScript 数组的索引值（index）从0开始，即数组第一个元素的索引值为0。最后一个元素的索引值等于该数组的长度减1（Array.length -1）。

```
var arr = ['this is the first element', 'this is the second element'];
console.log(arr[0]);              // logs 'this is the first element'
console.log(arr[1]);              // logs 'this is the second element'
console.log(arr[arr.length - 1]); // logs 'this is the second element'
```

数组中的元素像一个对象以索引为属性名,元素为属性值，或者arr=['a','b']有点像arrObj={0:'a',1:'b'},一个对象的属性是可以通过"."来访问,但是使用下面这样使用会抛出语法错误，因为属性名称是非法的：

```
console.log(arr.0); // a syntax error
```

这是由非法属性造成的，不是 Array 特有的。JavaScript 中数字开头的属性不能跟在点号后面；必须在方括号中使用。比如说，如果你有一个名为 '3d' 的属性，它只能通过方括号的形式进行访问。 换言之访问合法的属性名可以用"."或者"[ ]",但是访问非法的属性只能用"[ ]",如下所示：

```
var years = [1950, 1960, 1970, 1980, 1990, 2000, 2010];
console.log(years.0);   // a syntax error
console.log(years[0]);  // works properly
```

```
renderer.3d.setTexture(model, 'character.png');     // a syntax error
renderer['3d'].setTexture(model, 'character.png');  // works properly
```

在3d的例子中，'3d'的引号是必须的。该方法也可以用在 JavaScript 数组中（如：years['2'] 可以代替 years[2]），不过这不是必需的。在 years[2] 中，2会被Javascript 引擎自动调用 toString 转换成一个 string 类型的变量。正因为如此，'2' 和 '02' 在years 对象中将会指向不同的位置，下面这个例子将会打印 true：

```
console.log(years['2'] != years['02']);
```

类似的，想要使用保留字作为对象的属性名称的，只能通过以单引号包裹其字符串的形式访问：

```
var promise = {
  'var'  : 'text',
  'array': [1, 2, 3, 4]
};

console.log(promise['array']);
```

长度和数值下标属性性质之间的关系

JavaScript array 的 `length` 属性和其数字下标是有关系的。几个内置数组的方法 (例如, `join` , `slice` , `indexOf` , 等) 被调用的时候会使用 到 `length` 属性。 有些别的方法 (例如, `push` , `splice` , 等) 会改变array的 `length` 属性。

```
var fruits = [];
fruits.push('banana', 'apple', 'peach');

console.log(fruits.length); // 3
```

当你在 array 上使用一个合法的数组下标，而且该下标超出了当前数组的大小的时候，引擎会根据其值自动更新 array 的 `length` 属性：

```
fruits[5] = 'mango';
console.log(fruits[5]); // 'mango'
console.log(Object.keys(fruits));  // ['0', '1', '2', '5']
console.log(fruits.length); // 6
```

增大 `length` 。

```
fruits.length = 10;
console.log(Object.keys(fruits)); // ['0', '1', '2', '5']
console.log(fruits.length); // 10
```

减小 array 的 `length` 属性会删掉超出的元素。

```
fruits.length = 2;
console.log(Object.keys(fruits)); // ['0', '1']
console.log(fruits.length); // 2
```

使用正则匹配的结果来创建数组

正则表达式与字符串之间的匹配结果可以创建一个数组。这个数组包含了正则匹配的属性与匹配结果。 `RegExp.exec` ， `String.match` ， 与 `String.replace` 的返回值就是这样的数组。下面的例子可以帮助理解这些属性和元素。

```
// Match one d followed by one or more b's followed by one d
// Remember matched b's and the following d
// 忽略大小写

myRe = /d(b+)(d)/i;
myArray = myRe.exec("cdbBdbsbz");
```

该正则匹配返回的属性/元素列表：

Property/Element	说明	例子
`input`	原始的输入字符串，只读属性。	cdbBdbsbz
`index`	匹配的子字符串的第一个字符在原始字符串中的位置（从0开始的索引，只读）。	1
[0]	最后一次匹配的元素,只读。	dbBd
`[1], ...[n]`	出现在正则匹配中的子匹配（如果有）。	[1]: bB [2]: d

属性

For properties available on `Array` instances, see Properties of Array instances.

`Array.length`

`Array` 构造函数的 length 属性，其值为1。

`Array.prototype`

允许为所有数组对象附加属性。

方法

For methods available on `Array` instances, see Methods of Array instances.

`Array.from()` 🧪

从类数组或者迭代对象（iterable object）中创建一个新的数组实例。

`Array.isArray()`

假如一个变量是数组则返回 `true` ，否则返回 `false` 。

`Array.observe()` 🧪

异步监视数组的修改情况，与对象的 `Object.observe()` 方法类似。该方法会根据修改事件发生顺序提供一个修改流。

`Array.of()` 🧪

创建一个有可变数量的参数的新的数组实例，无论参数有多少数量，而且可以是任意类型。

数组实例

所有数组实例继承自 `Array.prototype` 。Array 构造函数的原型对象是可修改的，其会影响所有的数组实例。

属性

`Array.prototype.constructor`

所有的数组实例都继承了这个属性，它的值就是 `Array` ，表明了所有的数组都是由 `Array` 构造出来的。

`Array.prototype.length`

上面说了，因为 `Array.prototype` 也是个数组，所以它也有 `length` 属性，这个值为 `0` ，因为它是个空数组。

方法

Mutator 方法

下面的这些方法会改变调用它们的对象自身的值：

`Array.prototype.copyWithin()` 🧪
在数组内部，将一段元素序列拷贝到另一段元素序列上，覆盖原有的值。

`Array.prototype.fill()` 🧪
将数组中指定区间的所有元素的值，都替换成某个固定的值。

`Array.prototype.pop()`
删除数组的最后一个元素，并返回这个元素。

`Array.prototype.push()`
在数组的末尾增加一个或多个元素，并返回数组的新长度。

`Array.prototype.reverse()`
颠倒数组中元素的排列顺序，即原先的第一个变为最后一个，原先的最后一个变为第一个。

`Array.prototype.shift()`
删除数组的第一个元素，并返回这个元素。

`Array.prototype.sort()`
对数组元素进行排序，并返回当前数组。

`Array.prototype.splice()`
在任意的位置给数组添加或删除任意个元素。

`Array.prototype.unshift()`
在数组的开头增加一个或多个元素，并返回数组的新长度。

Accessor 方法

下面的这些方法绝对不会改变调用它们的对象的值，只会返回一个新的数组或者返回一个其它的期望值。

`Array.prototype.concat()`
返回一个由当前数组和其它若干个数组或者若干个非数组值组合而成的新数组。

`Array.prototype.includes()` 🧪
判断当前数组是否包含某指定的值，如果是返回 `true`，否则返回 `false`。

`Array.prototype.join()`
连接所有数组元素组成一个字符串。

`Array.prototype.slice()`

抽取当前数组中的一段元素组合成一个新数组。

Array.prototype.toSource() ⚠

返回一个表示当前数组字面量的字符串。遮蔽了原型链上的 `Object.prototype.toSource()` 方法。

Array.prototype.toString()

返回一个由所有数组元素组合而成的字符串。遮蔽了原型链上的 `Object.prototype.toString()` 方法。

Array.prototype.toLocaleString()

返回一个由所有数组元素组合而成的本地化后的字符串。遮蔽了原型链上的 `Object.prototype.toLocaleString()` 方法。

Array.prototype.indexOf()

返回数组中第一个与指定值相等的元素的索引，如果找不到这样的元素，则返回 -1。

Array.prototype.lastIndexOf()

返回数组中最后一个（从右边数第一个）与指定值相等的元素的索引，如果找不到这样的元素，则返回 -1。

Iteration 方法

在下面的众多遍历方法中，有很多方法都需要指定一个回调函数作为参数。在回调函数执行之前，数组的长度会被缓存在某个地方，所以，如果你在回调函数中为当前数组添加了新的元素，那么那些新添加的元素是不会被遍历到的。此外，如果在回调函数中对当前数组进行了其它修改，比如改变某个元素的值或者删掉某个元素，那么随后的遍历操作可能会受到未预期的影响。总之，不要尝试在遍历过程中对原数组进行任何修改，虽然规范对这样的操作进行了详细的定义，但为了可读性和可维护性，请不要这样做。

Array.prototype.forEach()

为数组中的每个元素执行一次回调函数。

Array.prototype.entries() 🧪

返回一个数组迭代器对象，该迭代器会包含所有数组元素的键值对。

Array.prototype.every()

如果数组中的每个元素都满足测试函数，则返回 `true` ，否则返回 `false`。

Array.prototype.some()

如果数组中至少有一个元素满足测试函数，则返回 true，否则返回 false。

`Array.prototype.filter()`
将所有在过滤函数中返回 `true` 的数组元素放进一个新数组中并返回。

`Array.prototype.find()`
找到第一个满足测试函数的元素并返回那个元素的值，如果找不到，则返回 `undefined` 。

`Array.prototype.findIndex()`
找到第一个满足测试函数的元素并返回那个元素的索引，如果找不到，则返回 `-1` 。

`Array.prototype.keys()`
返回一个数组迭代器对象，该迭代器会包含所有数组元素的键。

`Array.prototype.map()`
返回一个由回调函数的返回值组成的新数组。

`Array.prototype.reduce()`
从左到右为每个数组元素执行一次回调函数，并把上次回调函数的返回值放在一个暂存器中传给下次回调函数，并返回最后一次回调函数的返回值。

`Array.prototype.reduceRight()`
从右到左为每个数组元素执行一次回调函数，并把上次回调函数的返回值放在一个暂存器中传给下次回调函数，并返回最后一次回调函数的返回值。

`Array.prototype.values()`
返回一个数组迭代器对象，该迭代器会包含所有数组元素的值。

`Array.prototype[@@iterator]()`
和上面的 `values()` 方法是同一个函数。

数组通用方法

> **generics方法是非标准的，已弃用的，未来将会被移除的数组方法。** 需注意的是此方法同时有跨浏览器问题. 但是 Github上有可用的shim。

有时你想对字符串或其他类似数组的对象使用数组的方法（如函数arguments）。通过这样做，你可以把一个字符串作为（或以其他方式把非数组作为数组）数组里的字符来使用。例如，为了检查变量 `str` 每一个字符是否是字母，你会这样写：

```
1 function isLetter(character) {
2   return character >= 'a' && character <= 'z';
3 }
```

```
4
5  if (Array.prototype.every.call(str, isLetter)) {
6    console.log("The string '" + str + "' contains only letters!");
7  }
```

这种方法是相当费时的，在JavaScript1.6中其引入了一个通用的简写：

```
1  if (Array.every(str, isLetter)) {
2    console.log("The string '" + str + "' contains only letters!");
3  }
```

Generics在 `String` 也可用。

这并不是 ECMAScript 标准的一部分（虽然 ES2015 标准中的 `Array.from()` 可以用来实现这个）。 下面是一个shim，其可以在所有的浏览器中运行：

```
1  // Assumes Array extras already present (one may use polyfills for these
2  (function() {
3    'use strict';
4
5    var i,
6      // We could also build the array of methods with the following, but t
7      //   getOwnPropertyNames() method is non-shimable:
8      // Object.getOwnPropertyNames(Array).filter(function(methodName) {
9      //   return typeof Array[methodName] === 'function'
10     // });
11     methods = [
12       'join', 'reverse', 'sort', 'push', 'pop', 'shift', 'unshift',
13       'splice', 'concat', 'slice', 'indexOf', 'lastIndexOf',
14       'forEach', 'map', 'reduce', 'reduceRight', 'filter',
15       'some', 'every', 'find', 'findIndex', 'entries', 'keys',
16       'values', 'copyWithin', 'includes'
17     ],
18     methodCount = methods.length,
19     assignArrayGeneric = function(methodName) {
20       if (!Array[methodName]) {
21         var method = Array.prototype[methodName];
22         if (typeof method === 'function') {
23           Array[methodName] = function() {
24             return method.call.apply(method, arguments);
25           };
```

```
        }
      }
    };

  for (i = 0; i < methodCount; i++) {
    assignArrayGeneric(methods[i]);
  }
}());
```

示例

创建一个数组

在下面这个例子里，首先创建了一个长度为0的空数组 `msgArray` ，接着给 `msgArray[0]` 赋值，然后给 `msgArray[99]` 赋值，接着数组长度就变成了100。

```
var msgArray = [];
msgArray[0] = 'Hello';
msgArray[99] = 'world';

if (msgArray.length === 100) {
  console.log('数组长度为100。');
}
```

创建一个二维数组

这个例子创建了一个二维数组 `myVar` ，然后赋值。

```
var board = [
  ['R','N','B','Q','K','B','N','R'],
  ['P','P','P','P','P','P','P','P'],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  ['p','p','p','p','p','p','p','p'],
  ['r','n','b','q','k','b','n','r'] ];

console.log(board.join('\n') + '\n\n');

```

```
// Move King's Pawn forward 2
board[4][4] = board[6][4];
board[6][4] = ' ';
console.log(board.join('\n'));
```

下面是输出：

```
R,N,B,Q,K,B,N,R
P,P,P,P,P,P,P,P
 , , , , , , ,
 , , , , , , ,
 , , , , , , ,
 , , , , , , ,
p,p,p,p,p,p,p,p
r,n,b,q,k,b,n,r

R,N,B,Q,K,B,N,R
P,P,P,P,P,P,P,P
 , , , , , , ,
 , , , , , , ,
 , , , ,p, , ,
 , , , , , , ,
p,p,p,p, ,p,p,p
r,n,b,q,k,b,n,r
```

规范

规范	状态	说明
ECMAScript 1st Edition (ECMA-262)	ST Standard	初始定义。
ECMAScript 5.1 (ECMA-262) Array	ST Standard	新增方法: `Array.isArray`, `indexOf`, `lastIndexOf`, `every`, `some`, `forEach`, `map`, `filter`, `reduce`, `reduceRight`
ECMAScript 2015 (6th Edition, ECMA-262) Array	ST Standard	新增方法： `Array.from`, `Array.of`, `find`, `findIndex`, `fill`, `copyWithin`
ECMAScript Latest Draft (ECMA-	LS	新增方法： `Array.prototype.includes()`

262) Array	Living Standard	

浏览器支持

Desktop Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	(Yes)	(Yes)	(Yes)	(Yes)	(Yes)

相关链接

- JavaScript Guide: “Indexing object properties”
- JavaScript Guide: “Predefined Core Objects: `Array` Object”
- Array comprehensions
- Polyfill for JavaScript 1.8.5 Array Generics and ECMAScript 5 Array Extras
- Typed Arrays