数组

目录

- 定义
- 数组的本质
- length 属性
- in 运算符
- for...in 循环和数组的遍历
- 数组的空位
- 类似数组的对象
- 参考链接

定义

数组（ array）是按次序排列的一组值。每个值的位置都有编号（从 0开始），整个数组用方括号表示。

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

上面代码中的 a、b、c就构成一个数组，两端的方括号是数组的标志。 a是 0号位置， b是 1号位置， c是 2号位置。

除了在定义时赋值，数组也可以先定义后赋值。

```
var arr = [];

arr[0] = 'a';
arr[1] = 'b';
arr[2] = 'c';
```

任何类型的数据，都可以放入数组。

```
var arr = [
  {a: 1},
```

```
  [1, 2, 3],
  function() {return true;}
];

arr[0] // Object {a: 1}
arr[1] // [1, 2, 3]
arr[2] // function (){return true;}
```

上面数组 `arr` 的 3个成员依次是对象、数组、函数。

如果数组的元素还是数组，就形成了多维数组。

```
var a = [[1, 2], [3, 4]];
a[0][1] // 2
a[1][1] // 4
```

数组的本质

本质上，数组属于一种特殊的对象。 `typeof` 运算符会返回数组的类型是 `object` 。

```
typeof [1, 2, 3] // "object"
```

上面代码表明， `typeof` 运算符认为数组的类型就是对象。

数组的特殊性体现在，它的键名是按次序排列的一组整数（ 0，1，2...）。

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

Object.keys(arr)
// ["0", "1", "2"]
```

上面代码中， `Object.keys` 方法返回数组的所有键名。可以看到数组的键名就是整数 0、1、2。

由于数组成员的键名是固定的（默认总是 0、1、2...
），因此数组不用为每个元素指定键名，而对象的每个成员都必须指定键名。 JavaScript
语言规定，对象的键名一律为字符串，所以，数组的键名其实也是字符串。之所以可以用数值读取，是因为非
字符串的键名会被转为字符串。

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

```
arr['0'] // 'a'
arr[0] // 'a'
```

上面代码分别用数值和字符串作为键名，结果都能读取数组。原因是数值键名被自动转为了字符串。

注意，这点在赋值时也成立。一个值总是先转成字符串，再作为键名进行赋值。

```
var a = [];

a[1.00] = 6;
a[1] // 6
```

上面代码中，由于 `1.00` 转成字符串是 `1`，所以通过数字键 `1` 可以读取值。

上一章说过，对象有两种读取成员的方法：点结构（ `object.key` ）和方括号结构（ `object[key]` ）。但是，对于数值的键名，不能使用点结构。

```
var arr = [1, 2, 3];
arr.0 // SyntaxError
```

上面代码中， `arr.0` 的写法不合法，因为单独的数值不能作为标识符（ identifier ）。所以，数组成员只能用方括号 `arr[0]` 表示（方括号是运算符，可以接受数值）。

length 属性

数组的 `length` 属性，返回数组的成员数量。

```
['a', 'b', 'c'].length // 3
```

JavaScript 使用一个 32 位整数，保存数组的元素个数。这意味着，数组成员最多只有 4294967295 个（ 2^{32} - 1）个，也就是说 `length` 属性的最大值就是 4294967295。

只要是数组，就一定有 `length` 属性。该属性是一个动态的值，等于键名中的最大整数加上 `1` 。

```
var arr = ['a', 'b'];
arr.length // 2

arr[2] = 'c';
arr.length // 3

arr[9] = 'd';
```

```
arr.length // 10

arr[1000] = 'e';
arr.length // 1001
```

上面代码表示，数组的数字键不需要连续，`length`属性的值总是比最大的那个整数键大`1`。另外，这也表明数组是一种动态的数据结构，可以随时增减数组的成员。

`length`属性是可写的。如果人为设置一个小于当前成员个数的值，该数组的成员数量会自动减少到`length`设置的值。

```
var arr = [ 'a', 'b', 'c' ];
arr.length // 3

arr.length = 2;
arr // ["a", "b"]
```

上面代码表示，当数组的`length`属性设为`2`（即最大的整数键只能是`1`）那么整数键`2`（值为`c`）就已经不在数组中了，被自动删除了。

清空数组的一个有效方法，就是将`length`属性设为`0`。

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

arr.length = 0;
arr // []
```

如果人为设置`length`大于当前元素个数，则数组的成员数量会增加到这个值，新增的位置都是空位。

```
var a = ['a'];

a.length = 3;
a[1] // undefined
```

上面代码表示，当`length`属性设为大于数组个数时，读取新增的位置都会返回`undefined`。

如果人为设置`length`为不合法的值，JavaScript 会报错。

```
// 设置负值
[].length = -1
// RangeError: Invalid array length
```

```
// 数组元素个数大于等于2的32次方
[].length = Math.pow(2, 32)
// RangeError: Invalid array length

// 设置字符串
[].length = 'abc'
// RangeError: Invalid array length
```

值得注意的是，由于数组本质上是一种对象，所以可以为数组添加属性，但是这不影响 `length` 属性的值。

```
var a = [];

a['p'] = 'abc';
a.length // 0

a[2.1] = 'abc';
a.length // 0
```

上面代码将数组的键分别设为字符串和小数，结果都不影响 `length` 属性。因为，`length` 属性的值就是等于最大的数字键加 1，而这个数组没有整数键，所以 `length` 属性保持为 `0`。

如果数组的键名是添加超出范围的数值，该键名会自动转为字符串。

```
var arr = [];
arr[-1] = 'a';
arr[Math.pow(2, 32)] = 'b';

arr.length // 0
arr[-1] // "a"
arr[4294967296] // "b"
```

上面代码中，我们为数组 `arr` 添加了两个不合法的数字键，结果 `length` 属性没有发生变化。这些数字键都变成了字符串键名。最后两行之所以会取到值，是因为取键值时，数字键名会默认转为字符串。

in 运算符

检查某个键名是否存在的运算符 `in`，适用于对象，也适用于数组。

```
var arr = [ 'a', 'b', 'c' ];
2 in arr  // true
'2' in arr // true
4 in arr // false
```

上面代码表明，数组存在键名为 `2` 的键。由于键名都是字符串，所以数值 `2` 会自动转成字符串。

注意，如果数组的某个位置是空位，`in` 运算符返回 `false`。

```
var arr = [];
arr[100] = 'a';

100 in arr // true
1 in arr // false
```

上面代码中，数组 `arr` 只有一个成员 `arr[100]`，其他位置的键名都会返回 `false`。

for...in 循环和数组的遍历

`for...in` 循环不仅可以遍历对象，也可以遍历数组，毕竟数组只是一种特殊对象。

```
var a = [1, 2, 3];

for (var i in a) {
  console.log(a[i]);
}
// 1
// 2
// 3
```

但是，`for...in` 不仅会遍历数组所有的数字键，还会遍历非数字键。

```
var a = [1, 2, 3];
a.foo = true;

for (var key in a) {
  console.log(key);
}
// 0
```

```
// 1
// 2
// foo
```

上面代码在遍历数组时，也遍历到了非整数键 `foo`。所以，不推荐使用 `for...in` 遍历数组。

数组的遍历可以考虑使用 `for` 循环或 `while` 循环。

```
var a = [1, 2, 3];

// for循环
for(var i = 0; i < a.length; i++) {
  console.log(a[i]);
}

// while循环
var i = 0;
while (i < a.length) {
  console.log(a[i]);
  i++;
}

var l = a.length;
while (l--) {
  console.log(a[l]);
}
```

上面代码是三种遍历数组的写法。最后一种写法是逆向遍历，即从最后一个元素向第一个元素遍历。

数组的 `forEach` 方法，也可以用来遍历数组，详见《标准库》的 Array 对象一章。

```
var colors = ['red', 'green', 'blue'];
colors.forEach(function (color) {
  console.log(color);
});
// red
// green
// blue
```

数组的空位
当数组的某个位置是空元素，即两个逗号之间没有任何值，我们称该数组存在空位（ hole）。

var a = [1, , 1];
a.length // 3

上面代码表明，数组的空位不影响 length属性。虽然这个位置没有值，引擎依然认为这个位置是有效的。

需要注意的是，如果最后一个元素后面有逗号，并不会产生空位。也就是说，有没有这个逗号，结果都是一样
的。

var a = [1, 2, 3,];

a.length // 3
a // [1, 2, 3]

上面代码中，数组最后一个成员后面有一个逗号，这不影响 length属性的值，与没有这个逗号时效果一样。

数组的空位是可以读取的，返回 undefined。

var a = [, , ,];
a[1] // undefined

使用 delete命令删除一个数组成员，会形成空位，并且不会影响 length属性。

var a = [1, 2, 3];
delete a[1];

a[1] // undefined
a.length // 3

上面代码用 delete命令删除了数组的第二个元素，这个位置就形成了空位，但是对 length
属性没有影响。也就是说， length属性不过滤空位。所以，使用 length
属性进行数组遍历，一定要非常小心。

数组的某个位置是空位，与某个位置是 undefined，是不一样的。如果是空位，使用数组的 forEach方法、 for...in 结构、以及 Object.keys方法进行遍历，空位都会被跳过。

var a = [, , ,];

```
a.forEach(function (x, i) {
  console.log(i + '. ' + x);
})
// 不产生任何输出

for (var i in a) {
  console.log(i);
}
// 不产生任何输出

Object.keys(a)
// []
```

如果某个位置是 `undefined`，遍历的时候就不会被跳过。

```
var a = [undefined, undefined, undefined];

a.forEach(function (x, i) {
  console.log(i + '. ' + x);
});
// 0. undefined
// 1. undefined
// 2. undefined

for (var i in a) {
  console.log(i);
}
// 0
// 1
// 2

Object.keys(a)
// ['0', '1', '2']
```

这就是说，空位就是数组没有这个元素，所以不会被遍历到，而 `undefined` 则表示数组有这个元素，值是 `undefined`，所以遍历不会跳过。

类似数组的对象

如果一个对象的所有键名都是正整数或零，并且有 `length` 属性，那么这个对象就很像数组，语法上称为“类似数组的对象”（array-like object）。

```
var obj = {
  0: 'a',
  1: 'b',
  2: 'c',
  length: 3
};

obj[0] // 'a'
obj[1] // 'b'
obj.length // 3
obj.push('d') // TypeError: obj.push is not a function
```

上面代码中，对象 `obj` 就是一个类似数组的对象。但是，“类似数组的对象”并不是数组，因为它们不具备数组特有的方法。对象 `obj` 没有数组的 `push` 方法，使用该方法就会报错。

“类似数组的对象”的根本特征，就是具有 `length` 属性。只要有 `length` 属性，就可以认为这个对象类似于数组。但是有一个问题，这种 `length` 属性不是动态值，不会随着成员的变化而变化。

```
var obj = {
  length: 0
};
obj[3] = 'd';
obj.length // 0
```

上面代码为对象 `obj` 添加了一个数字键，但是 `length` 属性没变。这就说明了 `obj` 不是数组。

典型的“类似数组的对象”是函数的 `arguments` 对象，以及大多数 DOM 元素集，还有字符串。

```
// arguments对象
function args() { return arguments }
var arrayLike = args('a', 'b');

arrayLike[0] // 'a'
arrayLike.length // 2
arrayLike instanceof Array // false

// DOM元素集
var elts = document.getElementsByTagName('h3');
```

```
elts.length // 3
elts instanceof Array // false

// 字符串
'abc'[1] // 'b'
'abc'.length // 3
'abc' instanceof Array // false
```

上面代码包含三个例子，它们都不是数组（`instanceof`运算符返回`false`），但是看上去都非常像数组。

数组的`slice`方法可以将“类似数组的对象”变成真正的数组。

```
var arr = Array.prototype.slice.call(arrayLike);
```

除了转为真正的数组，“类似数组的对象”还有一个办法可以使用数组的方法，就是通过`call()`把数组的方法放到对象上面。

```
function print(value, index) {
  console.log(index + ' : ' + value);
}

Array.prototype.forEach.call(arrayLike, print);
```

上面代码中，`arrayLike`代表一个类似数组的对象，本来是不可以使用数组的`forEach()`方法的，但是通过`call()`，可以把`forEach()`嫁接到`arrayLike`上面调用。

下面的例子就是通过这种方法，在`arguments`对象上面调用`forEach`方法。

```
// forEach 方法
function logArgs() {
  Array.prototype.forEach.call(arguments, function (elem, i) {
    console.log(i + '. ' + elem);
  });
}

// 等同于 for 循环
function logArgs() {
  for (var i = 0; i < arguments.length; i++) {
    console.log(i + '. ' + arguments[i]);
  }
}
```

□□□□□□□□□□□□□□□□□□□□ `Array.prototype.forEach.call` □□□

```
Array.prototype.forEach.call('abc', function (chr) {
  console.log(chr);
});
// a
// b
// c
```

□□□□□□□□□□□□□□□□□□□□ `forEach` □□□□□□□□□□□□ “□□□□□□□□ ”
□□□□□□□□□□□□□□□□□□□□ `forEach` □□□

```
var arr = Array.prototype.slice.call('abc');
arr.forEach(function (chr) {
  console.log(chr);
});
// a
// b
// c
```

□□□□

- Axel Rauschmayer, Arrays in JavaScript
- Axel Rauschmayer, JavaScript: sparse arrays vs. dense arrays
- Felix Bohm, What They Didn’t Tell You About ES5’s Array Extras
- Juriy Zaytsev, How ECMAScript 5 still does not allow to subclass an array

---

□□□ #2
□ □□□ □□□ 2018-01-04 12:03:19 UTC
□ □□□ □□□ 2021-09-11 19:43:39 UTC