# Array.prototype.sort()

**sort()** 方法在适当的位置对数组的元素进行排序，并返回数组。 sort 排序不一定是稳定的。默认排序顺序是根据字符串Unicode码点。

```
var fruit = ['cherries', 'apples', 'bananas'];
fruit.sort();
// ['apples', 'bananas', 'cherries']

var scores = [1, 10, 21, 2];
scores.sort();
// [1, 10, 2, 21]
// 注意10在2之前,
// 因为在 Unicode 指针顺序中"10"在"2"之前

var things = ['word', 'Word', '1 Word', '2 Words'];
things.sort();
// ['1 Word', '2 Words', 'Word', 'word']
// 在Unicode中, 数字在大写字母之前,
// 大写字母在小写字母之前.
```

## 语法

```
arr.sort()

arr.sort(compareFunction)
```

## 参数

**compareFunction**

可选。用来指定按某种顺序进行排列的函数。如果省略，元素按照转换为的字符串的诸个字符的Unicode位点进行排序。

## 返回值

返回排序后的数组。原数组已经被排序后的数组代替。

## 描述

如果没有指明 `compareFunction` ，那么元素会按照转换为的字符串的诸个字符的Unicode位点进行排序。例如 "Banana" 会被排列到 "cherry" 之前。数字比大小时，9 出现在 80 之前，但这里比较时数字会先被转换为字符串，所以 "80" 比 "9" 要靠前。

如果指明了 `compareFunction` ，那么数组会按照调用该函数的返回值排序。即 a 和 b 是两个将要被比较的元素：

- 如果 `compareFunction(a, b)` 小于 0 ，那么 a 会被排列到 b 之前；
- 如果 `compareFunction(a, b)` 等于 0 ， a 和 b 的相对位置不变。备注： ECMAScript 标准并不保证这一行为，而且也不是所有浏览器都会遵守（例如 Mozilla 在 2003 年之前的版本）；
- 如果 `compareFunction(a, b)` 大于 0 ， b 会被排列到 a 之前。
- `compareFunction(a, b)` 必须总是对相同的输入返回相同的比较结果，否则排序的结果将是不确定的。

所以，比较函数格式如下：

```
function compare(a, b) {
  if (a is less than b by some ordering criterion) {
    return -1;
  }
  if (a is greater than b by the ordering criterion) {
    return 1;
  }
  // a must be equal to b
  return 0;
}
```

希望比较数字而非字符串，比较函数可以简单的以 a 减 b，如下的函数将会将数组升序排列

```
function compareNumbers(a, b) {
  return a - b;
}
```

`sort` 方法可以使用 函数表达式 方便地书写：

```
var numbers = [4, 2, 5, 1, 3];
numbers.sort(function(a, b) {
  return a - b;
});
console.log(numbers);

// [1, 2, 3, 4, 5]
```

对象可以按照某个属性排序：

```
var items = [
  { name: 'Edward', value: 21 },
  { name: 'Sharpe', value: 37 },
  { name: 'And', value: 45 },
  { name: 'The', value: -12 },
  { name: 'Magnetic' },
  { name: 'Zeros', value: 37 }
];

items.sort(function (a, b) {
  if (a.value > b.value) {
    return 1;
  }
  if (a.value < b.value) {
    return -1;
  }
  // a 必须等于 b
  return 0;
});
```

## 示例

### 创建、显示及排序数组

下述示例创建了四个数组，并展示原数组。之后对数组进行排序。对比了数字数组分别指定与不指定比较函数的结果。

```
var stringArray = ["Blue", "Humpback", "Beluga"];
var numericStringArray = ["80", "9", "700"];
var numberArray = [40, 1, 5, 200];
```

```
var mixedNumericArray = ["80", "9", "700", 40, 1, 5, 200];

function compareNumbers(a, b)
{
  return a - b;
}

console.log('stringArray:' + stringArray.join());
console.log('Sorted:' + stringArray.sort());

console.log('numberArray:' + numberArray.join());
console.log('Sorted without a compare function:'+ numberArray.sort());
console.log('Sorted with compareNumbers:'+ numberArray.sort(compareNumber

console.log('numericStringArray:'+ numericStringArray.join());
console.log('Sorted without a compare function:'+ numericStringArray.sort
console.log('Sorted with compareNumbers:'+ numericStringArray.sort(compar

console.log('mixedNumericArray:'+ mixedNumericArray.join());
console.log('Sorted without a compare function:'+ mixedNumericArray.sort(
console.log('Sorted with compareNumbers:'+ mixedNumericArray.sort(compare
```

该示例的返回结果如下。输出显示，当使用比较函数后，数字数组会按照数字大小排序。

```
stringArray: Blue,Humpback,Beluga
Sorted: Beluga,Blue,Humpback

numberArray: 40,1,5,200
Sorted without a compare function: 1,200,40,5
Sorted with compareNumbers: 1,5,40,200

numericStringArray: 80,9,700
Sorted without a compare function: 700,80,9
Sorted with compareNumbers: 9,80,700

mixedNumericArray: 80,9,700,40,1,5,200
Sorted without a compare function: 1,200,40,5,700,80,9
Sorted with compareNumbers: 1,5,9,40,80,200,700
```

## 对非 ASCII 字符排序

当排序非 ASCII 字符的字符串（如包含类似 e, é, è, a, ä 等字符的字符串）。一些非英语语言的字符串需要使用 `String.localeCompare` 。这个函数可以将函数排序到正确的顺序。

```
var items = ['réservé', 'premier', 'cliché', 'communiqué', 'café', 'adieu
items.sort(function (a, b) {
  return a.localeCompare(b);
});

// items is ['adieu', 'café', 'cliché', 'communiqué', 'premier', 'réserve
```

## 使用映射改善排序

`compareFunction` 可能需要对元素做多次映射以实现排序，尤其当 `compareFunction` 较为复杂，且元素较多的时候，某些 `compareFunction` 可能会导致很高的负载。使用 map 辅助排序将会是一个好主意。基本思想是首先将数组中的每个元素比较的实际值取出来，排序后再将数组恢复。

```
// 需要被排序的数组
var list = ['Delta', 'alpha', 'CHARLIE', 'bravo'];

// 对需要排序的数字和位置的临时存储
var mapped = list.map(function(el, i) {
  return { index: i, value: el.toLowerCase() };
})

// 按照多个值排序数组
mapped.sort(function(a, b) {
  return +(a.value > b.value) || +(a.value === b.value) - 1;
});

// 根据索引得到排序的结果
var result = mapped.map(function(el){
  return list[el.index];
});
```

## 规范

| Specification | Status | Comment |
| --- | --- | --- |
| ECMAScript 1st Edition (ECMA-262) | ST Standard | Initial definition. |
| ECMAScript 5.1 (ECMA-262) | | |

| | | |
|---|---|---|
| Array.prototype.sort | ST Standard | |
| ECMAScript 2015 (6th Edition, ECMA-262)<br>Array.prototype.sort | ST Standard | |
| ECMAScript Latest Draft (ECMA-262)<br>Array.prototype.sort | LS Living Standard | |

## 浏览器兼容性

**Desktop** Mobile

| Feature | Chrome | Firefox (Gecko) | Internet Explorer | Opera | Safari |
|---|---|---|---|---|---|
| Basic support | 1.0 | 1.0 (1.7 or earlier) | 5.5 | (Yes) | (Yes) |

## 参考

- `Array.prototype.reverse()`
- `String.prototype.localeCompare()`