Array 对象

`Array` 是 JavaScript 的原生对象，同时也是一个构造函数，可以用它生成新的数组。

1. 实例方法

1.1. `join()`

`join()` 方法以指定参数作为分隔符，将所有数组成员连接为一个字符串返回。**如果不提供参数，默认用逗号分隔。**

```
let a = [1, 2, 3, 4];

a.join(" ");              // '1 2 3 4'
a.join(" | ");            // "1 | 2 | 3 | 4"
a.join();                 // "1,2,3,4"
a.join() === a.join(","); // true
```

如果数组成员是 `undefined` 或 `null` 或空位，会被转成空字符串。

```
[undefined, null].join('#'); // '#'
['a',, 'b'].join('-');       // 'a--b'
```

通过 `call` 方法，这个方法也可以用于字符串或类似数组的对象。

```
Array.prototype.join.call("hello", "-"); // "h-e-l-l-o"

let obj = { 0: "a", 1: "b", length: 2 };
Array.prototype.join.call(obj, "-");     // 'a-b'
```

1.2 `concat()`

`concat()` **方法用于合并两个或多个数组。** 此方法不会更改现有数组，而是返回一个新数组。`concat()` 方法内的参数可以是任意类型。

```
["hello"].concat(["world"]);      // (2) ["hello", "world"]
["hello"].concat(1);              // (2) ["hello", 1]
["hello"].concat("!");            // (2) ["hello", "!"]
["hello"].concat(true);           // (2) ["hello", true]
["hello"].concat(function () {}); // (2) ["hello", ƒ]
["hello"].concat({ a: 1 });       // (2) ["hello", { a: 1 }]
```

1.3. reverse()

reverse() 方法用于颠倒排序数组元素，返回改变后的数组。该方法将改变原数组。

```
let a4 = [1, 2, 3, 4, 5];
let a5 = a4.reverse();
a4; // [5, 4, 3, 2, 1], 原数组被改变
a5; // [5, 4, 3, 2, 1]
```

1.4. slice()

slice() 方法用于提取目标数组的一部分，返回一个新数组，原数组不变。返回的新数组的长度等于第二个数减去第一个数的值。

```
let a6 = [1, 2, 3, 4, 5];
let a7 = a6.slice(-3, -1); // 负数表示倒数计算的位置
a6; // (5) [1, 2, 3, 4, 5]
a7; // (5) [3, 4]
```

如果第一个参数大于等于数组长度，或者第二个参数小于第一个参数，则返回空数组。

```
let a = ["a", "b", "c"];
a.slice(4);    // []
a.slice(2, 1); // []
```

slice() 方法的一个重要应用，是将类似数组的对象转为真正的数组。

类似数组的对象有字符串、DOM 集合、arguments 对象。以及其他定义了 length 属性和以自然数作为属性的对象。如 {0: 'a', length: 1}。

```
Array.prototype.slice.call('123'); // ['1', '2', '3']
Array.prototype.slice.call({ 0: "a", 1: "b", length: 2 }); // ['a', 'b']
Array.prototype.slice.call(document.querySelectorAll("div")); // (184) [...]
Array.prototype.slice.call((function fn(){return arguments})()); // []
```

1.5. splice()

splice() 方法删除原数组的一部分成员，并可以在删除的位置添加新的数组成员，返回值是被删除的元素组成的数组。该方法可以接收一个到 n 个参数，表示不同的含义。

- 第一个参数表示开始删除的位置，如果只提供一个参数，表示从此位置开始删除后面的全部元素。第一个参数是负数表示从倒数计算位置。
- 第二个参数表示删除的元素个数，返回的数组的长度和该值相同。第二个参数为 0 或负数表示不删除。

> - 后面的参数表示要添加在被删除元素的位置的元素。

```
// 一个参数情况，从索引为 2 位置开始删除后面的的全部成员
let a1 = [1, 2, 3, 4];
a1.splice(2); // [3, 4]
a1;           // [1, 2]

// 两个参数情况，从倒数第四位置开始删除后面的两位成员
let a2 = ["a", "b", "c", "d", "e", "f"];
a2.splice(-4, 2); // ["c", "d"]
a2;               // ["a", "b", "e", "f"]

// 三个参数情况，从索引为 1 位置开始不删除成员，添加成员 2
let a3 = [1, 1, 1];
a3.splice(1, 0, 2); // []
a3;                 // [1, 2, 1, 1]
```

1.6. sort()

`sort` 方法对数组成员进行排序，默认是按照字典顺序排序。排序后，原数组将被改变。

```
[1, 4, 2, 6, 0, 6, 2, 6].sort((a, b) => a - b);
// 数组将按照升序排列 [0, 1, 2, 2, 4, 6, 6, 6]
```

1.7. map()

`map` 方法还可以接受第二个参数，用来绑定回调函数内部的 `this` 变量。

```
let arr = ["a", "b", "c"];
[1, 2].map(function (e) {
  return this[e];
}, arr);
// (2) ["b", "c"]
// 第一轮循环 e 是 1，this 指向 arr，返回 ["a", "b", "c"][1]，即 "b"。
// 第二轮循环 e 是 2，this 指向 arr，返回 ["a", "b", "c"][2]，即 "c"。
```

不能使用箭头函数，箭头函数没有 `this` 概念，`this` 指向的是 `Window` 对象。

```
[2, 3].map((item) => {
  return this[item];
}, a);
// (2) [undefined, undefined]
```

1.8. forEach()

`forEach()` 方法与 `map()` 方法很相似，区别是前者不返回值。**`forEach()` 方法无法中断执行，总是会将所有成员遍历完。如果希望符合某种条件时，就中断遍历，要使用 for 循环。**

1.9. `filter()`

`filter()` 方法用于过滤数组成员，满足条件的成员组成一个新数组返回。该方法不改变原数组。

```
let arr = [0, 1, "a", false];
arr.filter(Boolean); // (2) [1, 'a']
```

上例中，`filter` 方法返回数组 `arr` 里面所有布尔值为 `true` 的成员。

1.10. `every()` 和 `some()`

`every()` 和 `some()` 返回一个布尔值，表示判断数组成员是否符合某种条件。

`every()` 接受第一个参数是函数，当遍历数组有不满足该函数条件时，返回 `false`。不再遍历后面的成员，若全部成员满足函数条件，则返回 `true`。类似于逻辑与（`&&`）运算符，只有全部表达式为真时才返回为 `true`，否则返回为 `false` 且当前面的表达式为 `false` 时，不计算后面的表达式。

`some()` 接受第一个参数是函数，当遍历数组有满足该函数条件时，返回 `true`。不再遍历后面的成员，若全部成员都不满足函数条件，则返回 `false`。类似于逻辑与（`||`）运算符，只有全部表达式为假时才返回为 `false`，否则返回为 `true` 且当前面的表达式为 `true` 时，不计算后面的表达式。

```
let a1 = [1, 2, 3, 4, 5];
a1.some(function (elem, index, arr) {
  console.log(elem);
  return elem >= 3;
});
// 1
// 2
// 3
// true

let a2 = [1, 2, 3, 4, 5];
a2.every(function (elem, index, arr) {
  console.log(elem);
  return elem >= 3;
});
// 1
// false
```

1.11. `reduce()` 和 `reduceRight()`

`reduce()` 方法和 `reduceRight()` 方法依次处理数组的每个成员，最终累计为一个值。它们的差别是，`reduce()` 是从左到右处理（从第一个成员到最后一个成员），`reduceRight()` 则是从右到左（从最后一个成员到第一个成员），其他完全一样。

`reduce()` 方法和 `reduceRight()` 方法的第一个参数都是一个函数。该函数接受四个参数：

1. 累积变量，默认为数组的第一个成员
2. 当前变量，默认为数组的第二个成员
3. 当前位置（从 0 开始）
4. 原数组

这四个参数之中，只有前两个是必须的，后两个则是可选的。

```
let arr = [1, 2, 3, 4, 5];
arr.reduce(function (a, b) {
  console.log("a = ", a, ", b = ", b);
  return 20;
});
// a =  1 , b =  2
// a =  20 , b =  3
// a =  20 , b =  4
// a =  20 , b =  5
```

上面返回是：

- 第一轮 `a` 的值是数组第一个成员 `1`，`b` 的值是数组第二个成员 `2`。
- 第二轮 `a` 的值是上一轮的返回结果 `20`，`b` 的值是数组第三个成员 `3`。
- 第三轮 `a` 的值是上一轮的返回结果 `20`，`b` 的值是数组第四个成员 `4`。
- 第四轮 `a` 的值是上一轮的返回结果 `20`，`b` 的值是数组第五个成员 `5`。

```
let arr = [1, 2, 3, 4];
arr.reduce(function (a, b) {
  console.log("a = ", a, ", b = ", b);
});
// a =  1 , b =  2
// a =  undefined , b =  3
// a =  undefined , b =  4
```

上面返回是：

- 第一轮 `a` 的值是数组第一个成员 `1`，`b` 的值是数组第二个成员 `2`。
- 第二轮 `a` 的值是上一轮的返回结果 `undefined`，因为没有返回，`b` 的值是数组第三个成员 `3`。
- 第三轮 `a` 的值是上一轮的返回结果 `undefined`，因为没有返回，`b` 的值是数组第四个成员 `4`。

如果要对累积变量指定初值，可以把它放在 `reduce()` 方法和 `reduceRight()` 方法的第二个参数。

与没有第二个参数相比较，这个参数将作为累计的初始值，也就是第一轮遍历时赋值给函数第一个参数。

```
let arr = [1, 2, 3, 4];
arr.reduce(function (a, b) {
  console.log("a = ", a, ", b = ", b);
}, 20);
```

```
// a =  20 , b =  1
// a =  undefined , b =  2
// a =  undefined , b =  3
// a =  undefined , b =  4
```

上面返回是：

- 第一轮 a 的值是累计初始值 20，b 的值是数组第一个成员 1。
- 第二轮 a 的值是上一轮的返回结果 undefined，因为没有返回，b 的值是数组第二个成员 2。
- 第三轮 a 的值是上一轮的返回结果 undefined，因为没有返回，b 的值是数组第三个成员 3。
- 第四轮 a 的值是上一轮的返回结果 undefined，因为没有返回，b 的值是数组第四个成员 4。

```
function subtract(prev, cur) {
  return prev - cur;
}
[3, 2, 1].reduce(subtract);         // 0
[(3, 2, 1)].reduceRight(subtract); // -4
```

reduce() 方法相当于 3 减去 2 再减去 1，reduceRight() 方法相当于 1 减去 2 再减去 3。

1.12. indexOf() 和 lastIndexOf()

indexOf 方法返回给定元素在数组中第一次出现的位置，如果没有出现则返回 -1。

这两个方法不能用来搜索 NaN 的位置，即它们无法确定数组成员是否包含 NaN。这是因为这两个方法内部，使用严格相等运算符（===）进行比较，而 NaN 是唯一一个不等于自身的值。

```
[NaN].indexOf(NaN);     // -1
[NaN].lastIndexOf(NaN); // -1
```

1.13. 链式调用

上面这些数组方法之中，有不少返回的还是数组，所以可以链式使用。

```
let arr = [1, 2, 3, 4, 5];
arr
  .map((item) => item * 3)                // 将原数组成员乘以 3
  .filter((item) => item % 2 !== 0)       // 然后取出其中奇数成员
  .reverse()                              // 再将数组成员的位置颠倒
  .forEach((item) => console.log(item)); // 最后打印数组成员
// 15
// 9
// 3
```