


## JavaScript 语言精粹五（数组与方法）

知人者智，自知者明。胜人者有力，自胜者强。 - 老子

### 起步：

数组是一段线性分配的内存，通过整数计算偏移，并访问其中的元素，数组是一种性能出色的数据结构。

数组性能优越在读取时间为  $O(1)$ ，[数组和链表](#)  [点击](#)

- JavaScript 中提供了一些类数组特性的对象（array-like）。
- 调用：它把数组下标改为字符串，用其作为属性。
- 速度：它明显比数组慢，但它的优点是使用起来更方便。
- 检索和更新：它的方式和对象一模一样，不过多个可以用整数作为属性名的特性。

### 数组字面量

- 字面量方式提供了非常简单的创建新数组的表示法
  - 概念：数组字面量是方括号包括空或多个以逗号分隔的值得表达式
  - 使用：可以在任何允许表达式的地方出现
  - 潜规则：数组首值属性名为'0'，后边以此类推
- 例如：

```
var container = [];  
var nums = ["zero", "one", "two", "three"];  
if (true) {  
    container[1]; // undefined  
    nums[1]; // one  
    container.length; // 0  
    nums.length; //4  
}  
//-对象字面量  
var nums_object = {  
    0: "zero",  
    1: "one",  
    2: "two",  
    3: "three",  
};
```

- 两者产生的结果几乎相同
  - 相似点
    - `nums` 和 `nums_object` 都包含相同对象
    - 它们的属性刚好名字和值也都相同
  - 区别，掌握原型链的知道
    - `nums` 继承自 `Array.prototype`，`nums_object` 继承自 `Object.prototype`
    - `nums` 特有一个 `length` 属性，而 `nums_object` 没有

- 特点 - 多数语言中同一个数组要求所有元素为相同类型 - **JavaScript** 中允许混合任意你需要的类型值

比如：

```
var YaHu=[  
  'poo',233,false,null,undefined,['compose1','compose2']  
  {object:true}  
]
```

## 长度

- 数组虽然都有一个 **length** 属性，但在 **JavaScript** 中，它的长度是没有限制的。
- 以大于当前数组长度的数字下标存储元素，那么它会自动扩充
- 补充：**语言的区别导致对数组的限制不同** [🔗 点击](#)
- 常识：
  - 在 **JavaScript** 中，**length** 属性与当前数组属性个数无关
  - 它的值等于当前数组最大整数属性名+1
  - 设置比数组属性个数大的 **length** 值不会给数组分配更多内存空间
  - 设置比数组属性个数小的则会删除所有下标大于等于 **length** 的属性

```
> var arr=['土豆','番茄'];  
    arr[94]='香瓜'  
    arr.length  
↵ 95
```

- 当然也可以利用数组从 **Array.prototype** 中继承来的方法来进行数组一系列操作
  - 使用频率较高的方法 [🔗 点击](#)

## 删除

- **JavaScript** 中数组也属于对象，所以可以使用 **delete** 运算符移除数组中某元素

```
var arrs=['番茄','黄瓜','土豆','茄子']  
delete arrs[2]  
arrs// ["番茄", "黄瓜", empty, "茄子"]
```

- 很明显可以看出，虽然这种方法删除了指定元素，但是却留下一个空白空间
  - 这是因为删除当前元素后方的元素会保留其原有属性不变，因此产生空挡
- 那么可以使用更优的解法——数组方法 **splice**
  - firstPar 指数组的序号，secondPar 指删除个数，后边再有参数指在序号后插入数组的元素
  - 它会删除特定元素，并且会改变后方元素的属性

```
arrs.splice(2, 1); // ["土豆"]  
arrs; // ["番茄", "黄瓜", "茄子"]
```

删除元素后边的元素之所以会变，是因为删除元素后边所有元素都要以新键值重新插入，也因此若操作量级较大的数组时，当前这种方法效率很低。

## 枚举

- 因为JavaScript 中数组其实就是对象
  - 所以 `for in` 可用来遍历数组所有属性
    - 但这种方法无法保证其顺序，故存在很大的隐患
  - 而使用 `for` 语句可避免这些问题
    - 经典三从句控制，1，初始循环；2，执行条件检测；3，增量运算

```
for(let i=0; i<arrs.length;i++){  
  do something--  
}
```

## 容易混淆的地方

- JavaScript 中，很容易出现必须使用数组时而使用了对象或者相反的情况
  - 属性名小而连续且为整数时，应该使用数组，否则使用对象
- 因为语言的关系，在 javascript 中使用 `typeof` 运算符检测数组没有任何意义
- 那么用于区分数组和对象，通常可利用 `constructor` 自定义函数的方式来解决
  - `constructor` 用来执行当前对象的构造函数
  - 也可以不编写函数，直接调用 `Array.isArray` 方法判定

```
function isArray(obj) {  
  return obj && typeof(obj) === 'object' && obj.constructor === Array  
}
```

## 数组扩充

- JavaScript 中提供了一套数组可用的方法，这些数组存储在 `Array.prototype` 中。

可在控制台进行查看 `dir(Array.prototype)`

- 无论是 `Object.prototype` 还是 `Array.prototype` 都是可扩充的
- 那么我们可以给数组扩充一个方法，用于数组的计算

```
Array.methods('reduce', function (f, value) { })  
//--书中给出是上方写法，我就以自己喜欢的方式改写了  
Array.prototype.fakereduce = function (fn, value) {
```

```

    for (let i = 0; i < this.length; i++) {
      value = fn(this[i], value)
    }
    return value;
  }
  let data = [1, 2, 4, 6];
  let sum = (a, b) => {
    return a + b;
  }
  data.fakereduce(sum, 0)//13

```

## 方法集

javascript 包含一套小型可用于标准类型上的方法集

### Array

- `array.concat(item)`
  - `concat` 方法产生一个新数组，包含一份 `array` 的浅拷贝（**shallow copy**）
  - 并把后方的 `item` 附加在数组上，若 `item` 为数组，则它的元素会分别添加。

```

let arr1 = ["first", "second"];
let arr2 = ["zero", "one"];
let newArr = arr1.concat(arr2, "flag");
// [ 'first', 'second', 'zero', 'one', 'flag' ]

```

- `array.join(separator)`
  - `join` 方法把一个 `array` 构造成字符串
    - 它先把 `array` 中每个元素构造成字符串
    - 然后使用 `separator` 分隔符进行拼接，默认 `separator` 是逗号
    - 不想使用逗号可使用空格进行替换

```

let arr1 = ["first", "second"];
let arr2 = ["zero", "one"];
let newArr = arr1.join(); // first,second
let newArr2 = arr2.join(""); // zeroone

```

- `array.push(item...)`
  - `push` 方法把一个或多个 `item` 附加到一个数组的尾部
  - （与 `concat` 方法不同的是，会改变原数组）
    - 若参数是数组，则把数组整个添加到原数组中（非逐个），并返回 `array` 的新长度值

```

let arr1 = ["first", "second"];
let arr2 = ["||"];

```

```
let arr3 = ["cherry", "watermelon"];
let newArr1 = arr1.push(arr2, "flag");
arr1; // [ 'first', 'second', [ '||' ], 'flag' ]
newArr1; // 4
let newArr2 = arr1.push(arr3);
arr1; //[ 'first', 'second', [ 'cherry', 'watermelon' ] ]
newArr2; // --3
```

- `array.reverse()`
  - `reverse` 方法会反转 `array` 中的元素顺序，并返回它本身；

```
let arr = ["first", "second", "third"];
let revArr = arr.reverse();
console.log(revArr); // [ 'third', 'second', 'first' ]
```

- `array.shift()`
  - `shift` 方法会移除数组中第一个元素，并返回该元素；
  - 若数组为空，则此方法返回 `undefined`
  - （通常情况下，`shift` 比 `pop` 要慢得多）

```
let arr = ["first", "second", "third"];
let revArr = arr.shift();
console.log(revArr); // first
/* 这种方法可以这样手动实现*/
Array.prototype.fakeShift = function () {
  return this.splice(0, 1)[0];
};
var revArr2 = arr.fakeShift();
console.log(revArr2); // second
```

- `array.slice(star,end)`
  - `slice` 方法会对数组中指定的一段进行浅复制；
    - 从 `array[star]` 复制到 `array[end]`
    - `end` 为非必须参数，默认值为当前数组的长度
    - 若两参数有任一负值，则会与数组长度相加，试图回正
    - 若 `star` 值大于数组长度，则回返回一个新的空数组

```
let arr = ["first", "second", "third", "fourth"];
// let revArr = arr.slice(1,2); // [ 'second' ]
// let revArr = arr.slice(1,-2); // [ 'second' ] -- -2+4=2
// let revArr = arr.slice(-3,2); // [ 'second' ] -- -3+4=1
```

- `array.sort(comparefn)`
  - `sort` 方法会对数组中的内容进行排序，但不可用于一组数字排序；

- 因为它回默认元素是字符串进行比较，所以进行排序时往往结果都是错的。
- 按字母顺序对数组中的元素进行排序，说得更精确点，是按照字符编码的顺序进行排序

### 引自 ——W3school

```
let arr1 = ["14", "3", "7", "11"];
let arr2 = ["d", "c", "a", "x"];
arr1.sort();
arr2.sort();
console.log(arr1); // [ '11', '14', '3', '7' ]
console.log(arr2); // [ 'a', 'c', 'd', 'x' ]
```

- 补救的方法就是补增一个比较函数
  - 若 a 小于 b，在排序后的数组中 a 应该出现在 b 之前，则返回一个小于 0 的值。
  - 若 a 等于 b，则返回 0。
  - 若 a 大于 b，则返回一个大于 0 的值。

### 引自 ——W3school

```
let arr3 = ["14", "3", "7", "11"];
arr3.sort(function (a, b) {
  return a - b;
});
console.log(arr3); // [ '3', '7', '11', '14' ]
```

- 修改后对纯数字的排序是解决了，但不适用于参数为字符串类型
  - 那么在函数中需考虑非纯数字的数组，做以下修正

```
var arr4 = ["b", "r", 14, "a", 3, 7, 11];
arr4.sort(function (a, b) {
  if (a === b) return 0;
  if (typeof a === typeof b) return a - b ? -1 : 1;
  return typeof a < typeof b ? -1 : 1;
});
console.log(arr4); // [ 3, 7, 11, 14, 'a', 'b', 'r' ]
```

- `array.splice(star,deleteCount,item..)`
  - `splice` 方法会从数组中易出一个或多个元素，并将新的 `item` 进行替换。
    - `star` 是指移除元素开始的位置
    - `deleteCount` 是指移除的个数
    - 若有额外的参数，则默认会插入到移除元素的位置

```
var arr4 = ["王花花", "赵光光", "李大脚"];
arr4.splice(1, 1, "孙漂亮");
console.log(arr4); // [ '王花花', '孙漂亮', '李大脚' ]
```

- `array.unshift(item..)`
  - `unshift` 方法会把 `item` 插入到数组的开始部分，并且返回数组的新长度

```
var arr5 = ["王花花", "赵光光", "李大脚"];
arr5.unshift("孙漂亮");
console.log(arr5); //[ '孙漂亮', '王花花', '赵光光', '李大脚' ]
```

- ◦ `unshift` 方法可以用如下方式实现

```
var arr6 = ["王花花", "赵光光", "李大脚"];
Array.prototype.fakeunshift = function () {
  this.splice.apply(
    this,
    [0, 0].concat(Array.prototype.slice.apply(arguments))
  );
  return this.length;
};
arr6.fakeunshift("孙漂亮");
console.log(arr6); //[ '孙漂亮', '王花花', '赵光光', '李大脚' ]
```

---

## 总结：

本章节描述的是在JavaScript语言中的数组的一些核心且常用的知识点，我手写了一些例子加个人总结，同时移除一些正则相关知识点（因为它的时间投入/收获比实在太低了），数组因为内容多单独分一个章节，下个章节总结，`Function`、`Number`、`String`、`Object` 等特性。