

JavaScript API

蓝桥杯Web组 省赛备赛

JavaScript 学习资源

[MDN](#)

[JavaScript 权威指南](#)

JavaScript速览

1. JavaScript是一门动态、解释性编程语言，非常适合面向对象和函数式编程风格。
2. 核心JavaScript定义了最小限度的API，用于操作数值、文本、数组、Map、Set等。
3. 浏览器是JavaScript最早的运行环境；Node.js是JavaScript的另一个运行环境，给予了JavaScript访问整个操作系统的权限。
4. JavaScript类型可以分为原始类型（Number、String、Boolean、null、undefined、Symbol） 和 对象类型（Array、Set、Map、RegExp、Date等）。
5. JavaScript有一个灵活而被诟病的自动转换规则。

Number

数学计算

```
Math.round(1.5) // 2
Math.fround(1.5) // 1.5, 舍入到最接近的32位浮点数
Math.ceil(1.1) // 2
Math.floor(1.9) // 1
Math.sqrt(4) // 2
Math.pow(2, 10) // 1024
Math.random() // 0.0 ~ 1.0
Math.max(1, 2, 3) // 3
Math.min(1, 2, 3) // 1
```

上溢出? 下溢出? 被零除?

```
Number.MAX_VALUE // 1.7976931348623157e+308
Number.MIN_VALUE // 5e-324
Number.isNaN(x)
Number.isFinite(x)
Number.isInteger(x)
Number.isSafeInteger(x)
```

String

```
let str = 'hello world';
str.length // 11
str[0] // 'h'
str[str.length - 1] // 'd'
str.substring(0, 5) // 'hello',
str.slice(0, 5) // 'hello'
str.slice(6) // 'world', 默认截取到结尾
str.slice(-1) // 'd', 支持负数
```

```
let str = 'hello world';
str.indexOf('world') // 6
str.lastIndexOf('o') // 7
str.includes('hello') // true
str.startsWith('hello') // true
str.endsWith('world') // true
```

```
let str = 'hello world';
str.toUpperCase() // 'HELLO WORLD'
str.toLowerCase() // 'hello world'
```

String

```
let str = 'hello world';
str.split(' ') // ['hello', 'world']
str.replace('world', 'javascript') // 'hello javascript'
```

```
let str = ' hello world ';
str.trim() // 'hello world'
str.trimStart() // 'hello world '
str.trimEnd() // ' hello world '
```

```
"x".padStart(3, "*") // '**x'
"x".padEnd(3, "*") // 'x**
"x".padStart(3) // '  x'

"◇".repeat(3) // '◇◇◇'
```

Boolean

```
Boolean(0) // false
Boolean(NaN) // false
Boolean('') // false
Boolean(null) // false
Boolean(undefined) // false
Boolean([]) // true!!!
Boolean({}) // true!!!
Boolean(' ') // true
Boolean('false') // true
```

原始值类型转换

隐式的类型转换：

```
x + ""    // String(x)  
`${x}`    // String(x), recommended  
+x        // Number(x)  
!! x      // Boolean(x)
```

显式的类型转换：

```
String(x)  
Number(x)  
Boolean(x)
```

在每次进行类型转换时，都需要想一下 undefined 和 null！

控制数字转字符串的格式：

```
let num = 123456.789;  
num.toFixed(2) // "123456.79"  
num.toPrecision(5) // "1.2346e+5"  
num.toExponential(2) // "1.23e+5"
```

Object

```
1  let obj = {  
2    name: 'Alice',  
3    age: 18,  
4    sayHello: function() {  
5      console.log('Hello, my name is ' + this.name);  
6    },  
7    sayHello2() { // 简写语法  
8      console.log('Hello, my name is ' + this.name);  
9    },  
10   ['key' + '1']: 'value1', // 方括号里支持 JavaScript 表达式  
11   get info() {  
12     return this.name + ' ' + this.age;  
13   }, // getter, 计算属性  
14   set setName(name) {  
15     this.name = name;  
16   } // setter  
17 }  
18  
19 obj.name // 'Alice'  
20 obj.age.length // undefined  
21 "xxx" in obj // false  
22 obj.xxx.length // Error  
23 obj?.xxx?.length // 可空属性访问, undefined  
24 obj?.xxx?.length ?? 0 // 默认值
```

Array

⚠ 数组是一种对象，其下标是一种特殊的对象属性，这可以解释很多JavaScript数组与其他语言不同的语法特性。

```
1 let arr = [1, , , 2]
2 arr.length // 4
3
4 arr[0] // 1
5 arr[-1] // undefined
6 arr[10] // undefined
7
8 arr = new Array(10)
9 arr.length // 10
10 arr[0] // undefined
11
12 Array.of(1, 2, 3) // [1, 2, 3]
13
14 Array.from('hello') // ['h', 'e', 'l', 'l', 'o']
15 [... 'hello'] // the same
```

Array 的就地操作

- `push(item)` : 末尾添加元素。
- `pop()` : 删除并返回末尾元素。

使用 `push` 和 `pop` 可以模拟栈。

- `unshift(item)` : 开头添加元素。
- `shift()` : 删除并返回开头元素。

使用 `push` 和 `shift` 可以模拟队列。

- `splice(loc)` : 删从索引 `loc` 开始的所有元素，返回被删除的元素数组。
- `splice(loc, count)` : 切出从索引 `loc` 开始的 `count` 个元素，返回被删除的元素数组。
- `splice(loc, count, ... items)` : 在切出元素的基础上，在 `loc` 位置插入元素。

⚠ 这几个方法都是对数组的就地操作，会改变原数组但不会返回原数组。

Array 的其他操作

- `slice(start, end)` : 返回从 `start` 到 `end` (不包括 `end`) 的片段。
- `fill(item, start, end)` : 用 `item` 填充从 `start` 到 `end` (不包括 `end`) 的片段。
- `sort(func)` : 按照指定规则排序。默认按照字符串字典序排序。

```
[1, 2, 3].sort((a, b) => a - b) // [1, 2, 3]
[1, 2, 3].sort((a, b) => b - a) // [3, 2, 1]
[
  {name: "Alice", grade: 1},
  {name: "Bob", grade: 2}
].sort((a, b) => a.grade - b.grade)
// [{name: "Alice", grade: 1}, {name: "Bob", grade: 2}]
```

- `reverse()` : 反转数组。
- `concat(... items)` : 返回新数组，新数组是原数组的副本，并在末尾添加 `items` 。
- `join(sep)` : 返回字符串，字符串由原数组的元素组成，元素之间用 `sep` 分隔。

Array 的迭代 ✕

- `forEach(func)` : 对每个元素调用 `func`。
- `map(func)` : 对每个元素调用 `func`， 返回新数组。
- `filter(func)` : 对每个元素调用 `func`， 返回新数组， 其中只包含 `func` 返回 `true` 的元素。
- `reduce(func, init)` : 对每个元素调用 `func`， `func` 的返回值作为下一次调用 `func` 时的第一个参数，
`init` 是第一次调用 `func` 时的第一个参数。

```
// 生成一个1~20的列表，筛选出其中的奇数，然后求和
new Array(20).fill(0)
  .map((_, idx) => idx + 1)
  .filter(x => x % 2 != 0)
  .reduce((a, b) => a + b, 0)
```

Array 的迭代 ✕

- `every(func)` : 对每个元素调用 `func` , 如果所有 `func` 都返回 `true` , 则返回 `true` 。
- `some(func)` : 对每个元素调用 `func` , 如果任意一个 `func` 返回 `true` , 则返回 `true` 。
- `find(func)` : 对每个元素调用 `func` , 返回第一个 `func` 返回 `true` 的元素。
- `findIndex(func)` : 对每个元素调用 `func` , 返回第一个 `func` 返回 `true` 的元素的索引。
- 其它数组方法: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Set

与数组不同的是，集合没有索引或顺序，也不允许重复：一个值要么是集合的成员，要么不是；不可能存在一个值在一个集合中出现多次。

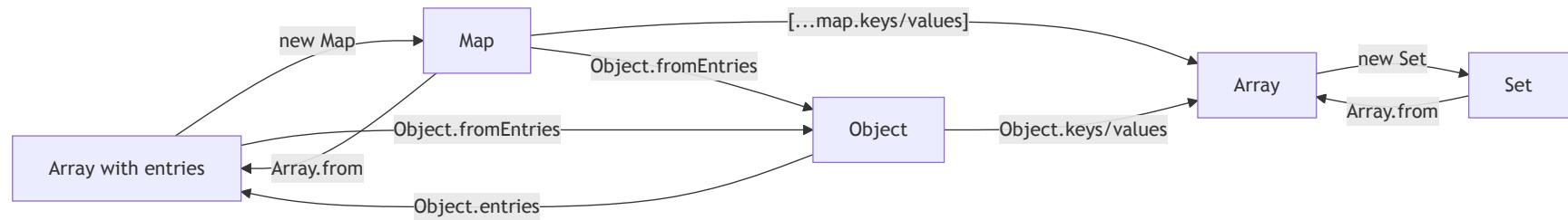
```
let set = new Set([1, 2, 3])
set.add(4)
set.delete(2)
set.has(3) // true
set.size // 2
```

Map

Map 对象保存键值对。任何值(对象或者原始值)都可以作为一个键或一个值。

```
let map = new Map()
map.set('name', 'Alice')
map.set('age', 20)
map.get('name') // 'Alice'
map.has('age') // true
map.delete('age')
map.size // 1
```

Array , Set , Map , Object 之间的转换



RegExp

学习正则表达式: Geek Hour

```
// `String.search`: 返回第一个匹配项起点的位置, 或-1.  
"JavaScript".search(/script/ui) // => 4  
"Python".search(/script/ui) // => -1  
  
// `String.replace`: 按正则替换字符串, 支持捕获组和命名捕获组。  
let quote = /"(^[^"]*)" /g // 一个引号 + 任意多个非引号字符 + 引号  
'He said "stop"'.replace(quote, '<q>$1</q>') // => 'He said <q>stop</q>'  
  
let quote = /"(?<quote>[^"]*)" /g  
'He said "stop"'.replace(quote, '<q>$<quote></q>') // => 'He said <q>stop</q>'
```

replace 传入函数的高级用法: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace#specifying_a_function_as_the_replacement

```
function replacer(match, p1, p2, p3, offset, string, groups) {  
  // p1 is non-digits, p2 digits, and p3 non-alphanumerics  
  return [p1, p2, p3].join(" - ");  
}  
const newString = "abc12345#$*%".replace(/([^\d]*)(\d*)([^ \w]*)/, replacer); // abc - 12345 - #$*%
```

```
// `String.match`：返回一个数组，数组中包含所有匹配项（如果有g标识），或第一个匹配项的详细信息（如果没有g标识）。
let nums = "12345678"
nums.match(/\d/g) // => ["1", "2", "3", "4", "5", "6", "7", "8"]
nums.match(/\d/) // => ["1", index: 0, input: "12345678", groups: undefined]

// `String.matchAll`：适用于循环遍历所有匹配项，必须带g标识。
[ ...nums.matchAll(/\d/g)] // => 0: ['1', index: 0, input: '12345678', groups: undefined]
//      1: ['2', index: 1, input: '12345678', groups: undefined]
//      ...
//      7: ['8', index: 7, input: '12345678', groups: undefined]
```

⚠ 会被g标志影响的方法: `String.match()`, `String.replace()`。

⚠ 不会被g标志影响的方法: `String.search()`

⚠ 必须带g标志的方法: `String.matchAll()`。

RegExp 的方法:

`test()`: 返回一个布尔值，表示当前模式是否能匹配参数字符串。

`exec()`: 始终返回一个匹配项。每次匹配后会更新搜索起点。

Date

```
let date = new Date()
date.getFullYear() // => 2025
date.getMonth() // => 0 (0表示1月)
date.getDate() // => 1
date.getDay() // => 0 (0表示星期日)
date.getTime() // => 1704128000000 (时间戳, 毫秒)
date.setFullYear(2020) // => 1577836800000
date.setMonth(date.getMonth() + 1) // 可进行计算
date > new Date(2020, 0, 1) // => true, 可进行比较
```

更多用法: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date