[JavaScript字符串操作方法大全，包含ES6方法](https://segmentfault.com/a/1190000016603159)https://sponsor.segmentfault.com/lg.php?bannerid=0&campaignid=0&zoneid=25&loc=https%3A%2F%2Fsegmentfault.com%2Fa%2F1190000016603159&referer=https%3A%2F%2Fwww.google.com%2F&cb=c9763282c3

一、charAt()

返回在指定位置的字符。

var str="abc"

console.log(str.charAt(0))*//a*

二、charCodeAt()

返回在指定的位置的字符的 Unicode 编码。

var str="abc"

console.log(str.charCodeAt(1))*//98*

三、concat()

连接字符串。

**var** a = "abc";

**var** b = "def";

**var** c = a.concat(b);

console.log(c);*//abcdef*

四、indexOf()

检索字符串。indexOf() 方法对大小写敏感！

var str="Hello world!"

console.log(str.indexOf("Hello"))*//0*

console.log(str.indexOf("World"))*//-1*

console.log(str.indexOf("world"))*///6*

五、match()

match() 方法可在字符串内检索指定的值，或找到一个或多个正则表达式的匹配。该方法类似 indexOf() 和 lastIndexOf()，但是它返回指定的值，而不是字符串的位置。

**var** str="1 abc 2 def 3"

console.log(str.match(/\d+/g))*//123*

六、replace()

replace() 方法用于在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子串。

**var** str="abc Def!"

console.log(str.replace(/abc/, "CBA"))*//CBA Def!*

七、search()

search() 方法用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的子字符串。要执行忽略大小写的检索，请追加标志 i。如果没有找到任何匹配的子串，则返回 -1。

**var** str="abc DEF!"

console.log(str.search(/DEF/))*//4*

八、slice()

提取字符串的片断，并在新的字符串中返回被提取的部分。  
stringObject.slice(**start,end**);  
**start** :要抽取的片断的起始下标。如果是负数，则该参数规定的是从字符串的尾部开始算起的位置。也就是说，-1 指字符串的最后一个字符，-2 指倒数第二个字符，以此类推。  
**end**：紧接着要抽取的片段的结尾的下标。若未指定此参数，则要提取的子串包括 start 到原字符串结尾的字符串。如果该参数是负数，那么它规定的是从字符串的尾部开始算起的位置。

var str="abc def ghk"

console.log(str.slice(6))*//f ghk*

九、split()

把字符串分割为字符串数组。

var str="abc def ghi jkl"

console.log(str.split(" "))//["abc", "def", "ghi", "jkl"]

console.log(str.split("") )//["a", "b", "c", " ", "d", "e", "f", " ", "g", "h", "i", " ", "j", "k", "l"]

console.log(str.split(" ",3))//["abc", "def", "ghi"]

十、toLocaleLowerCase()

把字符串转换为小写。

var str="ABC def!"

console.log(str.toLocaleLowerCase())*//abc def!*

十一、toLocaleUpperCase()

把字符串转换为大写。

var str="ABC def!"

console.log(str.toLocaleUpperCase())*//ABC DEF!*

十二、toLowerCase()

把字符串转换为小写。

var str="ABC def!"

console.log(str.toLowerCase())*//abc def!*

十三、toUpperCase()

把字符串转换为大写。

var str="ABC def!"

console.log(str.toUpperCase())*//ABC DEF!*

十四、substr()

从起始索引号提取字符串中指定数目的字符。  
stringObject.substr(**start,length**)。  
**start：**必需。要抽取的子串的起始下标。必须是数值。如果是负数，那么该参数声明从字符串的尾部开始算起的位置。也就是说，-1 指字符串中最后一个字符，-2 指倒数第二个字符，以此类推。  
**length：**可选。子串中的字符数。必须是数值。如果省略了该参数，那么返回从 stringObject 的开始位置到结尾的字串。

var str="abc def"

console.log(str.substr(2))*//c def*

console.log(str.substr(2,4))*// c de*

十五、substring()

提取字符串中两个指定的索引号之间的字符。  
stringObject.substring(**start,stop**)。  
**start ：**必需。一个非负的整数，规定要提取的子串的第一个字符在 stringObject 中的位置。  
**stop ：**可选。一个非负的整数，比要提取的子串的最后一个字符在 stringObject 中的位置多 1。如果省略该参数，那么返回的子串会一直到字符串的结尾。

var str="abc def"

console.log(str.substring(2))//c def

console.log(str.substring(2,4))// c

**相同点：**如果只是写一个参数，两者的作用都一样：都是是截取字符串从当前下标以后直到字符串最后的字符串片段。  
substr(startIndex);  
substring(startIndex);

var str = '123456789';

console.log(str.substr(2)); // "3456789"

console.log(str.substring(2)) ;// "3456789"

**不同点：**第二个参数  
substr（startIndex,lenth）： 第二个参数是截取字符串的长度（从起始点截取某个长度的字符串）；  
substring（startIndex, endIndex）： 第二个参数是截取字符串最终的下标 （截取2个位置之间的字符串,‘含头不含尾’）。

console.log("123456789".substr(2,5)); // "34567"

console.log("123456789".substring(2,5)) ;// "345"

ES6新增的操作字符串的方法

一、codePointAt()

let s = '𠮷a';

s.codePointAt(0) // 134071

s.codePointAt(1) // 57271

s.codePointAt(2) // 97

codePointAt方法的参数，是字符在字符串中的位置（从 0 开始）。上面代码中，JavaScript 将“𠮷a”视为三个字符，codePointAt 方法在第一个字符上，正确地识别了“𠮷”，返回了它的十进制码点 134071（即十六进制的20BB7）。在第二个字符（即“𠮷”的后两个字节）和第三个字符“a”上，codePointAt方法的结果与charCodeAt方法相同。

二、String.fromCodePoint()

ES5 提供String.fromCharCode方法，用于从码点返回对应字符，但是这个方法不能识别 32 位的 UTF-16 字符（Unicode 编号大于0xFFFF）。

String.fromCharCode(0x20BB7)

*// "ஷ"*

上面代码中，String.fromCharCode不能识别大于0xFFFF的码点，所以0x20BB7就发生了溢出，最高位2被舍弃了，最后返回码点U+0BB7对应的字符，而不是码点U+20BB7对应的字符。  
ES6 提供了String.fromCodePoint方法，可以识别大于0xFFFF的字符，弥补了String.fromCharCode方法的不足。在作用上，正好与codePointAt方法相反。

String.fromCodePoint(0x20BB7)

*// "𠮷"*

String.fromCodePoint(0x78, 0x1f680, 0x79) === 'x\uD83D\uDE80y'

*// true*

三、字符串的遍历器接口 for of

**for** (let codePoint of 'abc') {

console.log(codePoint)

}

// "a"

// "b"

// "c"

除了遍历字符串，这个遍历器最大的优点是可以识别大于0xFFFF的码点，传统的for循环无法识别这样的码点。

四、at（）

at方法可以识别Unicode 编号大于0xFFFF的字符，返回正确的字符。

‘abc’.at(0)//"a"

'吉'.at(0)//"吉"

五、normalize()

许多欧洲语言有语调符号和重音符号。为了表示它们，Unicode 提供了两种方法。一种是直接提供带重音符号的字符，比如Ǒ（u01D1）。另一种是提供合成符号（combining character），即原字符与重音符号的合成，两个字符合成一个字符，比如O（u004F）和ˇ（u030C）合成Ǒ（u004Fu030C）。

这两种表示方法，在视觉和语义上都等价，但是 JavaScript 不能识别。

'\u01D1'==='\u004F\u030C' //false

'\u01D1'.length // 1

'\u004F\u030C'.length // 2

上面代码表示，JavaScript 将合成字符视为两个字符，导致两种表示方法不相等。  
ES6 提供字符串实例的normalize()方法，用来将字符的不同表示方法统一为同样的形式，这称为 Unicode 正规化。

'\u01D1'.**normalize**() === '\u004F\u030C'.**normalize**()

*// true*

六、includes(), startsWith(), endsWith()

传统上，JavaScript 只有indexOf方法，可以用来确定一个字符串是否包含在另一个字符串中。ES6 又提供了三种新方法。

\*\*includes()\*\*：返回布尔值，表示是否找到了参数字符串。

\*\***startsWith**()\*\*：返回布尔值，表示参数字符串是否在原字符串的头部。

\*\***endsWith**()\*\*：返回布尔值，表示参数字符串是否在原字符串的尾部。

let s = 'Hello world!';

s.**startsWith**('Hello') *// true*

s.**endsWith**('!') *// true*

s.includes('o') *// true*

这三个方法都支持第二个参数，表示开始搜索的位置。

let s = 'Hello world!';

s.startsWith('world', 6) // true

s.endsWith('Hello', 5) // true

s.includes('Hello', 6) // false

上面代码表示，使用第二个参数n时，endsWith的行为与其他两个方法有所不同。它针对前n个字符，而其他两个方法针对从第n个位置直到字符串结束。

七、repeat()

repeat方法返回一个新字符串，表示将原字符串重复n次。

'x'.repeat(3) // "xxx"

'hello'.repeat(2) // "hellohello"

'na'.repeat(0) // ""

参数如果是小数，会被取整。

'na'.repeat(2.9) // "nana"

如果repeat的参数是负数或者Infinity，会报错。

'na'.**repeat***(Infinity)*

*// RangeError*

'na'.**repeat***(-1)*

*// RangeError*

八、padStart()，padEnd()

ES2017 引入了字符串补全长度的功能。如果某个字符串不够指定长度，会在头部或尾部补全。padStart()用于头部补全，padEnd()用于尾部补全。

'x'.padStart(5, 'ab') // 'ababx'

'x'.padStart(4, 'ab') // 'abax'

'x'.padEnd(5, 'ab') // 'xabab'

'x'.padEnd(4, 'ab') // 'xaba'

上面代码中，padStart和padEnd一共接受两个参数，第一个参数用来指定字符串的最小长度，第二个参数是用来补全的字符串。  
如果原字符串的长度，等于或大于指定的最小长度，则返回原字符串。

'xxx'.padStart(2, 'ab') // 'xxx'

'xxx'.padEnd(2, 'ab') // 'xxx'

如果用来补全的字符串与原字符串，两者的长度之和超过了指定的最小长度，则会截去超出位数的补全字符串。

'abc'.padStart(10, '0123456789')

// '0123456abc'

如果省略第二个参数，默认使用空格补全长度。

'x'.padStart(4) // ' x'

'x'.padEnd(4) // 'x '

padStart的常见用途是为数值补全指定位数。下面代码生成 10 位的数值字符串。

'1'.padStart(10, '0') // "0000000001"

'12'.padStart(10, '0') // "0000000012"

'123456'.padStart(10, '0') // "0000123456"

另一个用途是提示字符串格式。

'12'.padStart(10, 'YYYY-MM-DD') // "YYYY-MM-12"

'09-12'.padStart(10, 'YYYY-MM-DD') // "YYYY-09-12"

九、matchAll()

matchAll方法返回一个正则表达式在当前字符串的所有匹配。

十、字符串模板

模板字符串（template string）是增强版的字符串，用反引号（`）标识。它可以当作普通字符串使用，也可以用来定义多行字符串，或者在字符串中嵌入变量。-----字符串模板，工作中用到比较多。

*// 普通字符串*

`In JavaScript '\n' is a line-feed.`

*// 多行字符串*

`In JavaScript **this** is

**not** legal.`

console.log(`string text line 1

string text line 2`);

*// 字符串中嵌入变量*

let name = "Bob", time = "today";

`Hello ${name}, how are you ${time}?`

上面代码中的模板字符串，都是用反引号表示。如果在模板字符串中需要使用反引号，则前面要用反斜杠转义。

let greeting = `\`Yo\` World!`;

如果使用模板字符串表示多行字符串，所有的空格和缩进都会被保留在输出之中。

$('*#list').html(`*

<ul>

<li>first</li>

<li>second</li>

</ul>

`);

上面代码中，所有模板字符串的空格和换行，都是被保留的，比如<ul>标签前面会有一个换行。如果你不想要这个换行，可以使用trim方法消除它。

$('*#list').html(`*

<ul>

<li>first</li>

<li>second</li>

</ul>

`.trim());

模板字符串中嵌入变量，需要将变量名写在${}之中。

**function** authorize(user, action) {

**if** (!user.hasPrivilege(action)) {

throw new Error(

// 传统写法为

// 'User '

// + user.name

// + ' is not authorized to do '

// + action

// + '.'

`User ${user.name} is not authorized to **do** ${action}.`);

}

}

大括号内部可以放入任意的 JavaScript 表达式，可以进行运算，以及引用对象属性。

let x = 1;

let y = 2;

`${x} + ${y} = ${x + y}`

// "1 + 2 = 3"

`${x} + ${y \* 2} = ${x + y \* 2}`

// "1 + 4 = 5"

let obj = {x: 1, y: 2};

`${obj.x + obj.y}`

// "3"

模板字符串之中还能调用函数。

**function** **fn**() {

**return** "Hello World";

}

`foo ${fn()} bar`

*// foo Hello World bar*

如果大括号中的值不是字符串，将按照一般的规则转为字符串。比如，大括号中是一个对象，将默认调用对象的toString方法。  
如果模板字符串中的变量没有声明，将报错。

// 变量place没有声明

let msg = `Hello, ${place}`;

// 报错

由于模板字符串的大括号内部，就是执行 JavaScript 代码，因此如果大括号内部是一个字符串，将会原样输出。

`Hello ${'World'}`

*// "Hello World"*

模板字符串甚至还能嵌套。

const tmpl = addrs => `

<table>

${addrs.**map**(addr => `

<tr><td>${addr.first}</td></tr>

<tr><td>${addr.last}</td></tr>

`).join('')}

</table>

`;