

# 字符串的新增方法

## 1. 实例方法: `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
```

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。

## 2. 实例方法: `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
```

但是，如果参数是 0 到 -1 之间的小数，则等同于 0，这是因为会先进行取整运算。0 到 -1 之间的小数，取整以后等于 -0，repeat 视同为 0。

```
'na'.repeat(-0.9); // ""
```

参数 NaN 等同于 0。

```
'na'.repeat(NaN); // ""
```

如果 repeat 的参数是字符串，则会先转换成数字。

```
'na'.repeat('na'); // ""  
'na'.repeat('3'); // "nanana"
```

### 3. 实例方法：padStart(), padEnd()

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'
```

padStart() 和 padEnd() 一共接受两个参数，第一个参数是字符串补全生效的最大长度，第二个参数是用来补全的字符串。

**如果原字符串的长度，等于或大于最大长度，则字符串补全不生效，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最大长度，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"
```

## 4. 实例方法: `trimStart()`, `trimEnd()`

`trimStart()` 和 `trimEnd()` 这两个方法。它们的行为与 `trim()` 一致，`trimStart()` 消除字符串头部的空格，`trimEnd()` 消除尾部的空格。它们返回的都是新字符串，不会修改原始字符串。

```
const s = '  abc  ';

s.trim(); // "abc"
s.trimStart(); // "abc  "
s.trimEnd(); // "  abc"
s; // '  abc  '
```

除了空格键，这两个方法对字符串头部（或尾部）的 `tab` 键、换行符等不可见的空白符号也有效。

浏览器还部署了额外的两个方法，`trimLeft()` 是 `trimStart()` 的别名，`trimRight()` 是 `trimEnd()` 的别名。

## 5. 实例方法: `replaceAll()`

历史上，字符串的实例方法 `replace()` 只能替换第一个匹配。

```
'aabbcc'.replace('b', '_')
// 'aa_bcc'
```

如果要替换所有的匹配，不得不使用正则表达式的 `g` 修饰符。

```
'aabbcc'.replace(/b/g, '_')  
// 'aa__cc'
```

`replaceAll()` 方法，可以一次性替换所有匹配。

```
'aabbcc'.replaceAll('b', '_')  
// 'aa__cc'
```

## 6. 实例方法： `at()`

`at()` 方法接受一个整数作为参数，返回参数指定位置的字符，支持负索引（即倒数的位置）。

```
const str = 'hello';  
str.at(1) // "e"  
str.at(-1) // "o"  
str.at(0) // "h"  
str.at() // "h" , 不传参数取第一位  
str.at(9) // undefined  
str.at(-9) // undefined
```

如果参数位置超出了字符串范围，`at()` 返回 `undefined`。