# 1.js中的有用函数

## 1.typeof(x) ,返回x的数据类型

|  |
| --- |
|  |

# 2.Strings操作

## 1.+

### “1” + “2” = “12”

## 2.获取一个字符串的字符个数,str.length

## 3.字符串切片函数slice(start,end)

## 4.把字符串转换为大写,str.toUpperCase()

## 5.把字符串转换为小写,str.toLowerCase()

# 扩展: javaScript之字符串常用方法

String 对象，用于处理文本(字符串)的对象，允许操作和格式化文本字符串以及确定和定位字符串中的子字符串,String，基础数据类型。前面这句话呢，是我对string的简介，下面呢是我给大家一起分享string的一些常用方法，以及使用的注意事项，如有不足和不对的地方，请大家帮忙加以指证，谢谢。

* charAt()
* concat()
* indexOf(), lastIndexOf()
* localeCompare()
* slice()
* split()
* substr(), substring()
* match()
* replace()
* search()
* toLocaleLowerCase(), toLocaleUpperCase(),toLowerCase(),toUpperCase()
* includes(), startsWith(), endsWith()
* repeat()
* 模板字符串``反引号

**charAt()**

charAt,返回指定索引位置处的字符，它的功能实现类似于数组用中括号获取相应下标位置的数据，当然，字符串也可以使用中括号，下面看代码： `

代码解读

复制代码

var str1 = 'abcdefg';

console.log(str1.charAt(2)) // 输出 'c'

console.log(str1[2]) // 输出 'c'

`charAt在日常开发的使用中，更多会用于处理兼容IE的低版本浏览器，IE6，7；在低版本的IE浏览器，中括号的方式是会输出undefined;

**注意事项**：charAt 方法返回一个字符值，该字符值等于指定 index 位置的字符。 一个字符串中的第一个字符位于索引位置 0，第二个字符位于索引位置 1，依此类推。 超出范围的 index 返回空字符串。

**concat()**

concat,在数组的常用方法里也有这个方法，功能是用来返回一个合并拼接两个或两个以上数组的，字符串这个concat也是一样的功能，返回一个两个或者两个以上的字符串拼接的字符串，并不影响原字符串。下面看代码： `

var str1 = 'abcdefg', str2 = '1234567';

var str3 = str1.concat(str2);

console.log(str3) // 输出 'abcdefg1234567'

` **注意事项**：concat 方法的结果等同于：result = string1 + string2 + string3 + stringN。源字符串中或结果字符串中的值的变化都不会影响另一个字符串中的值。如果有不是字符串的参数，则它们在连接到 string1 之前将首先被转换为字符串。

**indexOf,lastIndexOf**

indexOf,返回一个字符在字符串中首次出现的位置，lastIndexOf返回一个字符在字符串中最后一次出现的位置，这两个方法功能类似，在实际使用中都是根据需要使用，下面看代码： `

var str1 = 'abcdcefcg';

console.log(str1.indexOf('c')) // 输出 '2'

console.log(str1.lastIndexOf('c')) // 输出 '7'

`indexOf和lastIndexOf都接收两个参数，第一个是要索引的字符，必填参数，第二个是可选参数，就是索引位置，从什么地方开始索引查找必填字符。 **注意事项**：indexOf,lastIndexOf方法返回 String 对象中的子字符串的位置。如果未找到子字符串，则返回 -1。如果 startindex 为负，则 startindex将被视为零。如果它大于最大索引，则将其视为最大索引。

**localeCompare()**

localeCompare() 方法用本地特定的顺序来比较两个字符串。localeCompare 方法对 stringVar 和 stringExp 执行区分区域设置的字符串比较，并返回以下结果之一，这取决于系统默认区域设置的排序顺序：-1，如果 stringVar 排在 stringExp 之前。+1，如果 stringVar 排在 stringExp 的后面。 如果两个字符串相等，则为 0（零）： `

str1 = 'GBDCFEA'

var strArr = str1.split('').sort(function(strVar, strExp){

return strVar.localeCompare(strExp)

})

console.log(strArr.join('')) // ABCDEFG

` **注意事项**:字符串排序的方法本身也是借助数组的sort来实现的，sort方法的用法请见我的[javaScript数组之数组常用方法](https://juejin.cn/post/6844903681159069704)

**slice()**

slice提取字符串的片断，并把提取的字符串作为新的字符串返回出来。下面请看代码： `

var str = 'abcdefg';

str.slice() // 输出 'abcdefg', 不传递参数默认复制整个字符串

str.slice(1) // 输出 'bcdefg',传递一个，则为提取的起点，然后到字符串结尾

str.slice(2, str.length-1)// 输出'cdef',传递两个，为提取的起始点和结束点

`**注意事项**：如果 起始点 为负，则将其视为 length + 起始点，此处 length 为字符串的长度。如果 结束点 为负，则会将其视为 length + 结束点。如果省略 结束点，则将一直复制到 stringObj 的结尾。如果 结束点 出现在 起始点 之前，则不会将任何字符复制到新字符串中。

**split()**

使用指定的分隔符将一个字符串拆分为多个子字符串，并将其以数组形式返回。下面看例子： `

var str = 'A\*B\*C\*D\*E\*F\*G';

str.split('\*') // 输出 ["A", "B", "C", "D", "E", "F", "G"]

`split是字符当中使用的非常频繁的一个方法，多数情况下是把字符串转数组来处理，比如前面的localeCompare()就有使用到。**注意事项**:split接收两个可选参数：参数一:一个字符串或正则表达式对象，标识用于分隔字符串的一个或多个字符。如果忽略该参数，则将返回包含整个字符串的单元素数组;参数二：一个用于限制数组中返回的元素数量的值。

**substr(), substring()**

截取一个字符串的片段，并返回截取的字符串。这两个方法的功能非常的类似，下面先看代码演示： `

var str = 'ABCDEFGHIJKLMN';

str.substr(2) // 输出 'CDEFGHIJKLMN'

str.substring(2) // 输出 'CDEFGHIJKLMN'

str.substr(2, 9) // 输出 'CDEFGHIJK'

str.substring(2, 9) // 输出 'CDEFGHI'

` **substr和substring这两个方法不同的地方就在于参数二，substr的参数二是截取返回出来的这个字符串指定的长度，substring的参数二是截取返回这个字符串的结束点，并且不包含这个结束点**。而它们的参数一，都是一样的功能，截取的起始位置。 **注意事项**：substr的参数二如果为0或者负数，则返回一个空字符串，如果未填入，则会截取到字符串的结尾去。substring的参数一和参数二为NAN或者负数，那么它将被替换为 0。

**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，并返回一个包含该搜索结果的数组。下面请看例子：`

var str = '2018年结束了，2019年开始了，2020年就也不远了';

var rex = /\d+/g // 这里是定义匹配规则，匹配字符串里的1到多个数字

str,match(rex) //输出符合匹配规则的内容，以数组形式返回 ['2018', '2019', '2020']

str.match('20')// 不使用正则 ["20", index: 0, input: "2018年结束了，2019年开始了"]

` **注意事项**:如果 match 方法没有找到匹配，将返回 null。如果找到匹配，则 match 方法会把匹配到以数组形式返回，如果正则规则未设置全局修饰符g，则 match 方法返回的数组有两个特性：input 和 index。 input 属性包含整个被搜索的字符串。 index 属性包含了在整个被搜索字符串中匹配的子字符串的位置。

**replace()** replace，翻译成中文就是替换，把一个目标字符串当中的文本，替换成自己需要的字符。replace接收两个参数，参数一是需要替换掉的字符或者一个正则的匹配规则，参数二，需要替换进去的字符，仔实际的原理当中，参数二，你可以换成一个回调函数，具体实现请看代码： `

var str = '2018年结束了，2019年开始了，2020年就也不远了',str1='',str2='';

var rex = /\d+/g // 这里是定义匹配规则，匹配字符串里的1到多个数字

str1 = str.replace(rex, '\*\*\*\*') //输出："\*\*\*\*年结束了，\*\*\*\*年开始了,\*\*\*\*年也不远了";

str2 = str.replace(rex, function(item){

console.log(arguments) // 看下面的图片

var arr = ['零', '壹', '贰', '叁', '肆', '伍', '陆', '柒', '捌', '玖']

var newStr = '';

item.split('').map(function(i){

newStr += arr[i]

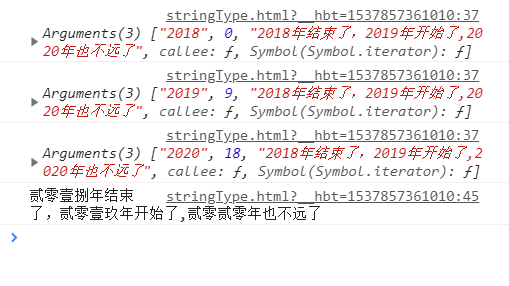
})

return newStr

})

console.log(str2)// 输出：贰零壹捌年结束了，贰零壹玖年开始了,贰零贰零年也不远了

`

在这个代码体里，我们用了两种替换，第一种是直接把参数二定义成一个用于替换进去的字符，第二种是把参数二换成了一个回调函数，两种方式，其实都可以，这个取决于我们的工作需要，就以上面这个例子来说，如果我们是转换成大写的话，那就用回调函数的方式。其实回调函数的内部机制里，也是根据正则的匹配规则去匹配自己需要被替换掉的字符，把匹配到的字符，就可以在回调函数里写自己的逻辑，然后就把它返回出去，回调函数本身，接收三个参数，参数一是匹配到的需要被替换掉的字符，参数二是匹配到的需要被替换掉的字符在原字符串中开始的位置，参数三就是原字符串。

**search()**

search，翻译成中文就是搜索，在目标字符串中搜索与正则规则相匹配的字符，搜索到，则返回第一个匹配项在目标字符串当中的位置，没有搜索到则返回一个-1。 `

var str = '2018年结束了，2019年开始了，2020年就也不远了',str1='',str2='';

var rex = /\d+/i // 这里是定义匹配规则,匹配字符串里的1到多个数字

str.search(rex) // 输出 0 这里搜索到的第一项是从位置0开始的

`

**toLocaleLowerCase(), toLocaleUpperCase(),toLowerCase(),toUpperCase()**

toLocaleLowerCase和toLowerCase都是把字母转换成小写，toLocaleUpperCase和toUpperCase则是把字母转换成大写。toLocaleLowerCase和toLocaleUpperCase 方法转换字符串中的字符，同时考虑到宿主环境的当前区域设置。大多数情况下，此结果与使用 toLowerCase和toUpperCase 方法所获得的结果相同。如果语言规则与常规的 Unicode 大小写映射冲突，则结果将会不同。 `

var str = 'abcdefg';

str.toLocaleUpperCase(); // 输出：'ABCDEFG'

str.toUpperCase(); // 输出： 'ABCDEFG'

`

**includes(), startsWith(), endsWith()**

includes，startsWith，endsWithes6的新增方法，includes用来检测目标字符串对象是否包含某个字符，返回一个布尔值，startsWith用来检测当前字符是否是目标字符串的起始部分，相对的endwith是用来检测是否是目标字符串的结尾部分。请看演示代码： `

var str = 'Excuse me, how do I get to park road?';

str.includes('how') // 输出：true;

str.startsWith('Excuse') // 输出： true;

str.endsWith('?') // 输出： true;

这三个方法都支持第二个参数，表示开始搜索的位置。

var str = 'Excuse me, how do I get to park road?';

str.includes('how', 0) // 输出：true;

str.startsWith('Excuse', 0) // 输出： true;

str.endsWith('?', 0) // 输出： true;

`

**repeat()** 返回一个新的字符串对象，新字符串等于重复了指定次数的原始字符串。接收一个参数，就是指定重复的次数。 `

代码解读

var str = 'http';

str.repeat(3) // 输出： 'httphttphttp';

`注意：当该参数为负或正无穷时，此方法会引发 RangeError。

**模板字符串``反引号**

反引号是es6新引入的定义字符串的方式，用``包裹需要定义成字符串的字符，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。下面看代码演示： `

代码解读

var str = 'https://juejin.im/timeline'; // 以前的定义方式

var str1 = `https://juejin.im/timeline`; // 反引号的方式

定义多行字符串

代码解读

var strHtml = '<ul>' +

'<li></li>' +

'<li></li>' +

'</ul>'; // 以前的定义方式

var strHtml1 = `<ul>

<li></li>

<li></li>

</ul>`; // 反引号的方式,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。

` 传递变量,模板字符串中嵌入变量，需要将变量名写在${}之中,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,模板字符串之中还能调用函数。

`

反引号里面可以使用插值表达式

let name = "Kenny"

console.log(`hello:${name}`);

输出:

