String 对象

概述

`String`对象是 JavaScript 原生提供的三个包装对象之一，用来生成字符串对象。

```javascript
var s1 = 'abc';
var s2 = new String('abc');

typeof s1 // "string"
typeof s2 // "object"

s2.valueOf() // "abc"
```

上面代码中，变量`s1`是字符串，`s2`是对象。由于`s2`是字符串对象，`s2.valueOf`方法返回的就是它所对应的原始字符串。

字符串对象是一个类似数组的对象（很像数组，但不是数组）。

```javascript
new String('abc')
// String {0: "a", 1: "b", 2: "c", length: 3}

(new String('abc'))[1] // "b"
```

上面代码中，字符串`abc`对应的字符串对象，有数值键（`0`、`1`、`2`）和`length`属性，所以可以像数组那样取值。

除了用作构造函数，`String`对象还可以当作工具方法使用，将任意类型的值转为字符串。

```javascript
String(true) // "true"
String(5) // "5"
```

上面代码将布尔值`true`和数值`5`，分别转换为字符串。

静态方法

String.fromCharCode()

`String`对象提供的静态方法（即定义在对象本身，而不是定义在对象实例的方法），主要是`String.fromCharCode()`。该方法的参数是一个或多个数值，代表 Unicode 码点，返回值是这些码点组成的字符串。

```javascript
```

```
String.fromCharCode() // ""
String.fromCharCode(97) // "a"
String.fromCharCode(104, 101, 108, 108, 111)
// "hello"
```

上面代码中，`String.fromCharCode`方法的参数为空，就返回空字符串；否则，返回参数对应的 Unicode 字符串。

注意，该方法不支持 Unicode 码点大于`0xFFFF`的字符，即传入的参数不能大于`0xFFFF`（即十进制的 65535）。

```javascript
String.fromCharCode(0x20BB7)
// "ஷ"
String.fromCharCode(0x20BB7) === String.fromCharCode(0x0BB7)
// true
```

上面代码中，`String.fromCharCode`参数`0x20BB7`大于`0xFFFF`，导致返回结果出错。`0x20BB7`对应的字符是汉字`吉`，但是返回结果却是另一个字符（码点`0x0BB7`）。这是因为`String.fromCharCode`发现参数值大于`0xFFFF`，就会忽略多出的位（即忽略`0x20BB7`里面的`2`）。

这种现象的根本原因在于，码点大于`0xFFFF`的字符占用四个字节，而 JavaScript 默认支持两个字节的字符。这种情况下，必须把`0x20BB7`拆成两个字符表示。

```javascript
String.fromCharCode(0xD842, 0xDFB7)
// "吉"
```

上面代码中，`0x20BB7`拆成两个字符`0xD842`和`0xDFB7`（即两个两字节字符，合成一个四字节字符），就能得到正确的结果。码点大于`0xFFFF`的字符的四字节表示法，由 UTF-16 编码方法决定。

实例属性

String.prototype.length

字符串实例的`length`属性返回字符串的长度。

```javascript
'abc'.length // 3
```

实例方法

String.prototype.charAt()

`charAt`方法返回指定位置的字符，参数是从`0`开始编号的位置。

```javascript
var s = new String('abc');

s.charAt(1) // "b"
s.charAt(s.length - 1) // "c"
```

这个方法完全可以用数组下标替代。

```javascript
'abc'.charAt(1) // "b"
'abc'[1] // "b"
```

如果参数为负数，或大于等于字符串的长度，`charAt`返回空字符串。

```javascript
'abc'.charAt(-1) // ""
'abc'.charAt(3) // ""
```

String.prototype.charCodeAt()

`charCodeAt`方法返回字符串指定位置的 Unicode 码点（十进制表示），相当于`String.fromCharCode()`的逆操作。

```javascript
'abc'.charCodeAt(1) // 98
```

上面代码中，`abc`的`1`号位置的字符是`b`，它的 Unicode 码点是`98`。

如果没有任何参数，`charCodeAt`返回首字符的 Unicode 码点。

```javascript
'abc'.charCodeAt() // 97
```

如果参数为负数，或大于等于字符串的长度，`charCodeAt`返回`NaN`。

```javascript
'abc'.charCodeAt(-1) // NaN
'abc'.charCodeAt(4) // NaN
```

注意，`charCodeAt`方法返回的 Unicode 码点不会大于65536（0xFFFF），也就是说，只返回两个字节的字符的码点。如果遇到码点大于 65536 的字符（四个字节的字符），必需连续使用两次`charCodeAt`，不仅读入`charCodeAt(i)`，还要读入`charCodeAt(i+1)`，将两个值放在一起，才能得到准确的字符。

String.prototype.concat()

`concat`方法用于连接两个字符串，返回一个新字符串，不改变原字符串。

```javascript
var s1 = 'abc';
var s2 = 'def';

s1.concat(s2) // "abcdef"
s1 // "abc"
```

该方法可以接受多个参数。

```javascript
'a'.concat('b', 'c') // "abc"
```

如果参数不是字符串，`concat`方法会将其先转为字符串，然后再连接。

```javascript
var one = 1;
var two = 2;
var three = '3';

''.concat(one, two, three) // "123"
one + two + three // "33"
```

上面代码中，`concat`方法将参数先转成字符串再连接，所以返回的是一个三个字符的字符串。作为对比，加号运算符在两个运算数都是数值时，不会转换类型，所以返回的是一个两个字符的字符串。

String.prototype.slice()

`slice`方法用于从原字符串取出子字符串并返回，不改变原字符串。它的第一个参数是子字符串的开始位置，第二个参数是子字符串的结束位置（不含该位置）。

```javascript
'JavaScript'.slice(0, 4) // "Java"
```

如果省略第二个参数，则表示子字符串一直到原字符串结束。

```javascript
'JavaScript'.slice(4) // "Script"
```

如果参数是负值，表示从结尾开始倒数计算的位置，即该负值加上字符串长度。

```javascript
'JavaScript'.slice(-6) // "Script"
'JavaScript'.slice(0, -6) // "Java"
'JavaScript'.slice(-2, -1) // "p"
```

如果第一个参数大于第二个参数，`slice`方法返回一个空字符串。

```javascript
'JavaScript'.slice(2, 1) // ""
```

String.prototype.substring()

`substring`方法用于从原字符串取出子字符串并返回，不改变原字符串，跟`slice`方法很相像。它的第一个参数表示子字符串的开始位置，第二个位置表示结束位置（返回结果不含该位置）。

```javascript
'JavaScript'.substring(0, 4) // "Java"
```

如果省略第二个参数，则表示子字符串一直到原字符串的结束。

```javascript
'JavaScript'.substring(4) // "Script"
```

如果第一个参数大于第二个参数，`substring`方法会自动更换两个参数的位置。

```javascript
'JavaScript'.substring(10, 4) // "Script"
// 等同于
'JavaScript'.substring(4, 10) // "Script"
```

上面代码中，调换`substring`方法的两个参数，都得到同样的结果。

如果参数是负数，`substring`方法会自动将负数转为0。

```javascript
'JavaScript'.substring(-3) // "JavaScript"
'JavaScript'.substring(4, -3) // "Java"
```

上面代码中，第二个例子的参数`-3`会自动变成`0`，等同于`'JavaScript'.substring(4, 0)`。由于第二个参数小于第一个参数，会自动互换位置，所以返回`Java`。

由于这些规则违反直觉，因此不建议使用`substring`方法，应该优先使用`slice`。

String.prototype.substr()

`substr`方法用于从原字符串取出子字符串并返回，不改变原字符串，跟`slice`和`substring`方法的作用相同。

`substr`方法的第一个参数是子字符串的开始位置（从0开始计算），第二个参数是子字符串的长度。

```javascript
'JavaScript'.substr(4, 6) // "Script"
```

如果省略第二个参数，则表示子字符串一直到原字符串的结束。

```javascript
'JavaScript'.substr(4) // "Script"
```

如果第一个参数是负数，表示倒数计算的字符位置。如果第二个参数是负数，将被自动转为0，因此会返回空字符串。

```javascript
'JavaScript'.substr(-6) // "Script"
'JavaScript'.substr(4, -1) // ""
```

上面代码中，第二个例子的参数`-1`自动转为`0`，表示子字符串长度为`0`，所以返回空字符串。

String.prototype.indexOf()，String.prototype.lastIndexOf()

`indexOf`方法用于确定一个字符串在另一个字符串中第一次出现的位置，返回结果是匹配开始的位置。如果返回`-1`，就表示不匹配。

```javascript
'hello world'.indexOf('o') // 4
'JavaScript'.indexOf('script') // -1
```

`indexOf`方法还可以接受第二个参数，表示从该位置开始向后匹配。

```javascript
'hello world'.indexOf('o', 6) // 7
```

`lastIndexOf`方法的用法跟`indexOf`方法一致，主要的区别是`lastIndexOf`从尾部开始匹配，`indexOf`则是从头部开始匹配。

```javascript
'hello world'.lastIndexOf('o') // 7
```

另外，`lastIndexOf`的第二个参数表示从该位置起向前匹配。

```javascript
'hello world'.lastIndexOf('o', 6) // 4
```

String.prototype.trim()

`trim`方法用于去除字符串两端的空格，返回一个新字符串，不改变原字符串。

```javascript
'  hello world  '.trim()
// "hello world"
```

该方法去除的不仅是空格，还包括制表符（`\t`、`\v`）、换行符（`\n`）和回车符（`\r`）。

```javascript
'\r\nabc \t'.trim() // 'abc'
```

String.prototype.toLowerCase()，String.prototype.toUpperCase()

`toLowerCase`方法用于将一个字符串全部转为小写，`toUpperCase`则是全部转为大写。它们都返回一个新字符串，不改变原字符串。

```javascript
'Hello World'.toLowerCase()
// "hello world"

'Hello World'.toUpperCase()
// "HELLO WORLD"
```

String.prototype.match()

`match`方法用于确定原字符串是否匹配某个子字符串，返回一个数组，成员为匹配的第一个字符串。如果没有找到匹配，则返回`null`。

```javascript
'cat, bat, sat, fat'.match('at') // ["at"]
'cat, bat, sat, fat'.match('xt') // null
```

```

返回的数组还有`index`属性和`input`属性，分别表示匹配字符串开始的位置和原始字符串。

```javascript
var matches = 'cat, bat, sat, fat'.match('at');
matches.index // 1
matches.input // "cat, bat, sat, fat"
```

`match`方法还可以使用正则表达式作为参数，详见《正则表达式》一章。

String.prototype.search()，String.prototype.replace()

`search`方法的用法基本等同于`match`，但是返回值为匹配的第一个位置。如果没有找到匹配，则返回`-1`。

```javascript
'cat, bat, sat, fat'.search('at') // 1
```

`search`方法还可以使用正则表达式作为参数，详见《正则表达式》一节。

`replace`方法用于替换匹配的子字符串，一般情况下只替换第一个匹配（除非使用带有`g`修饰符的正则表达式）。

```javascript
'aaa'.replace('a', 'b') // "baa"
```

`replace`方法还可以使用正则表达式作为参数，详见《正则表达式》一节。

String.prototype.split()

`split`方法按照给定规则分割字符串，返回一个由分割出来的子字符串组成的数组。

```javascript
'a|b|c'.split('|') // ["a", "b", "c"]
```

如果分割规则为空字符串，则返回数组的成员是原字符串的每一个字符。

```javascript
'a|b|c'.split('') // ["a", "|", "b", "|", "c"]
```

如果省略参数，则返回数组的唯一成员就是原字符串。

```javascript

```
'a|b|c'.split() // ["a|b|c"]
```

如果满足分割规则的两个部分紧邻着（即两个分割符中间没有其他字符），则返回数组之中会有一个空字符串。

```javascript
'a||c'.split('|') // ['a', '', 'c']
```

如果满足分割规则的部分处于字符串的开头或结尾（即它的前面或后面没有其他字符），则返回数组的第一个或最后一个成员是一个空字符串。

```javascript
'|b|c'.split('|') // ["", "b", "c"]
'a|b|'.split('|') // ["a", "b", ""]
```

`split`方法还可以接受第二个参数，限定返回数组的最大成员数。

```javascript
'a|b|c'.split('|', 0) // []
'a|b|c'.split('|', 1) // ["a"]
'a|b|c'.split('|', 2) // ["a", "b"]
'a|b|c'.split('|', 3) // ["a", "b", "c"]
'a|b|c'.split('|', 4) // ["a", "b", "c"]
```

上面代码中，`split`方法的第二个参数，决定了返回数组的成员数。

`split`方法还可以使用正则表达式作为参数，详见《正则表达式》一节。

String.prototype.localeCompare()

`localeCompare`方法用于比较两个字符串。它返回一个整数，如果小于0，表示第一个字符串小于第二个字符串；如果等于0，表示两者相等；如果大于0，表示第一个字符串大于第二个字符串。

```javascript
'apple'.localeCompare('banana') // -1
'apple'.localeCompare('apple') // 0
```

该方法的最大特点，就是会考虑自然语言的顺序。举例来说，正常情况下，大写的英文字母小于小写字母。

```javascript
'B' > 'a' // false
```

上面代码中，字母`B`小于字母`a`。因为 JavaScript 采用的是 Unicode 码点比较，`B`的码点是66，而`a`的码点是97。

但是，`localeCompare`方法会考虑自然语言的排序情况，将`B`排在`a`的前面。

```javascript
'B'.localeCompare('a') // 1
```

上面代码中，`localeCompare`方法返回整数1，表示`B`较大。

`localeCompare`还可以有第二个参数，指定所使用的语言（默认是英语），然后根据该语言的规则进行比较。

```javascript
'ä'.localeCompare('z', 'de') // -1
'ä'.localeCompare('z', 'sv') // 1
```

上面代码中，`de`表示德语，`sv`表示瑞典语。德语中，`ä`小于`z`，所以返回`-1`；瑞典语中，`ä`大于`z`，所以返回`1`。

参考链接

- Ariya Hidayat, [JavaScript String: substring, substr, slice](http://ariya.ofilabs.com/2014/02/javascript-string-substring-substr-slice.html)