String 对象

`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。

```
let s1 = "abc";
let s2 = new String("abc");

typeof s1; // 'string'
typeof s2; // 'object'

s2.valueOf(); // 'abc'
```

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。

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

除了用作构造函数创建类似数组的对象外，`String` 还可以当做工具函数使用，将任意类型的值转换为字符串。

1. 静态方法

1.1. 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。

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

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）。

```
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）。

3. 实例属性

3.1. String.prototype.length

```
"abc".length; // 3
```

4. 实例方法

4.1. String.prototype.charAt()

`charAt()` 方法返回某个位置的字符，位置从 0 开始。

```
let s = new String("abc");
s.charAt(); // "a"
s.charAt(0); // "a"
s.charAt(1); // "b"
s.charAt(s.length - 1); // "c"
s.charAt(-1); // ""
s.charAt(4); // ""
```

从上面代码可以看出，`charAt()` 方法：

- 不传参数时，返回第一个字符，相当于传参 `0`。
- 传参为负数时，返回空字符串。
- 传参为一个大于字符串长度的数时，返回空字符串。

4.2. 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。

```
"d".charCodeAt(0); // 100
String.fromCharCode(100); // "d"
```

4.3. String.prototype.concat()

`concat` 方法用于连接多个字符，返回一个新字符串，不改变原字符串。

```
let s11 = "qwe";
let s12 = "asd";
let s13 = s11.concat(s12);
```

```
s13; // "qweasd";
s11; // "qwe"

let s14 = "zxc";
let s15 = s14.concat(s12, s13);
s15; // "zxcasdqweasd"
```

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。

```
var one = 1;
var two = 2;
var three = "3";

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

4.4. String.prototype.search()

`search()` 匹配字符串，返回匹配到的第一个位置。如果没有匹配到，返回 `-1`。类似于 `indexOf()`。

```
"dog,pig,monkey".search("mo"); // 8
"dog,pig,monkey".search("money"); // -1

"dog,pig,monkey".indexOf("mo"); // 8
"dog,pig,monkey".indexOf("money"); // -1
```

4.5. String.prototype.replace()

`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项。返回一个新的字符串，不改变原字符串。

```
let s = "aaa";
let s1 = s.replace("a", "b");
s; // "aaa"
s1; // "baa"
```

4.6. String.prototype.match()

```
str.match(regexp);
```

`match` 方法用于确定原字符串是否匹配某一个子字符串，返回一个数组。如果传入一个非正则表达式对象，则会隐式地使用 `new RegExp(obj)` 将其转换为一个 `RegExp` 。

返回值包括：

- 匹配的成员。
- `groups`: 一个捕获组数组 或 undefined（如果没有定义命名捕获组）。
- `index`: 匹配的结果的开始位置。
- `input`: 搜索的字符串。

```
let a = "cat, bat, sat, fat".match("at");
a instanceof Array; // true
a[0]; // "at"
a["index"]; // 1
["at", (index: 1), (input: "cat, bat, sat, fat"), (groups: undefined)];
```

4.7. 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。

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

从上面代码可以看出：

- 如果分割的字符串和原字符串相同，则返回两个空字符串成员组成的数组。
- 如果分割的字符串是空字符串，则返回数组的成员是原字符串的每一个字符。
- 如果省略参数，则返回数组的唯一成员就是原字符串。
- 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。
- 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。

`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。

```
"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"]
```

4.8. 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。

```
`   hello world  `.trim(); // "hello world"
```

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

```
"\r\n\vabc \t".trim(); // "abc"

("\n");
// "
// "

("\t"); // "    "
("\v"); // "□"
```

4.9. String.prototype.toLowerCase() 和 String.prototype.toUpperCase()

toLowerCase 方法用于将一个字符串中字符全部转换为小写，toUpperCase 方法将一个字符串中字符全部转换为大写。返回新的字符串，不改变原字符。

```
let s = "Hello World";
let sLower = s.toLowerCase();
sLower; // "hello world"
s; // "Hello World"

let s1 = "Hello World";
let s1Upper = s1.toUpperCase();
s1Upper; // "HELLO WORLD"
s1; // "Hello World"
```

4.10. 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。

```
"dog".localeCompare("dog"); // 0
"apple".localeCompare("banana"); // -1
"e".localeCompare("d"); // 1
```

JavaScript 采用的是 Unicode 码点比较，直接通过比较运算符来比较两个字符时 B 小于 a。

```
"B" > "a"; // false
```

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

```
"B".localeCompare("a"); // 1
```

4.11. String.prototype.substr()、String.prototype.substring() 和 String.prototype.slice()

`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。区别是：

4.11.1. 第二个参数的含义不同

- `substr` 方法的**第二个参数表示子字符串的长度。**
- `substring` 方法的第二个参数表示子字符串的结束位置（不含该位置）。
- `slice` 方法的第二个参数表示子字符串的结束位置（不含该位置）。

```
"JavaScript".substr(1, 4); // "avaS"
"Javascript".substring(1, 4); // "ava"
"Javascript".slice(1, 4); // "ava"
```

4.11.2. 对于参数为负的处理不同

- `substr` 方法，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，会被自动转换成 0，会返回空字符串。
- `substring` 方法，任何一个参数是负数将自动转换为 0.
- `slice` 方法，参数是负数表示从结尾开始倒数计算的位置，即该负数加上字符串长度。

```
"JavaScript".substr(2, -2); // "", 第二个参数为负会被转换为 0，表示截取 0 个字符，于是返回为空。。
"JavaScript".substr(-7, 4); // "aScr", 从倒数第 7 位开始取 4 位字符。

"JavaScript".substring(-7, 4); // "Java", 第一个参数会被自动转换为 0，从 0 开始取到第 4 位（不包括第 4 位）。
"JavaScript".substring(-7, -4); // "", 两个参数会被自动转换为 0。

"Javascript".slice(-5, -4); // "c" , 表示从倒数第 5 位取到倒数第 4 位（不包括倒数第 4 位）。
"Javascript".slice(-7, 4); // "a" , 表示从倒数第 7 位（10 - 7 = 3，第 3 位）取到倒数第 4 位（不包括倒数第 4 位）。
```

4.11.3. 对于第一个参数大于第二个参数的处理不同

- `substr` 方法，分别按照上面的规则处理第一个参数、第二个参数。
- `substring` 方法，如果第一个参数大于第二个参数，自动调换位置。
- `slice` 方法，如果第一个参数大于第二个参数，且同时为正或同时为负时，返回空字符串，其他情况参考前面的标准。

```

```

```
"JavaScript".substr(3, 2); // "aS", 从第 3 位截取两个字符。
"JavaScript".substr(-3, -5); // "", 如果第二个参数是负数，会被自动转换成 0，会返回空字符串。

"JavaScript".substring(1, -4); // "J", 第一个参数大于第二个参数，相互调换位置，第一个参数为负转换为 0，表示从 0  位取到第 1 位（不包括第一位）
"JavaScript".substring(-5, -6); // ""，  任何一个参数为负都被转换为 0。

"JavaScript".slice(3, 2); // "", 返回空字符串。
"JavaScript".slice(-3, -4); // "", 返回空字符串。
"JavaScript".slice(3, -4); // "aSc", 从第 3 位截取到第  (10 - 4 = 6)  位（不包括第6位）。
```

4.12. String.prototype.indexOf() 和 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。

```
"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。

```
"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。

```
"hello world".lastIndexOf("o"); // 7, 从最后一个字符开始向前匹配，  第一次匹配到 'o', 是 "world" 中的 'o', 所在的位置是 7
```

另外，`lastIndexOf` 的第二个参数表示从该位置起向前匹配。

```
"hello world".lastIndexOf("o", 6); // 4, 从第 6 个字符 ('w') 开始向前匹配，第 1 次匹配到 'o' 是 "hello" 中的 'o', 所在的位置是 4
"hello world".lastIndexOf("o", 7); // 7, 从第 7 个字符 ('o') 开始向前匹配，第 1 次匹配到 'o' 是 "world" 中的 'o', 所在的位置是 7
```