字符串

概述

定义

字符串就是零个或多个排在一起的字符，放在单引号或双引号之中。

```javascript
'abc'
"abc"
```

单引号字符串的内部，可以使用双引号。双引号字符串的内部，可以使用单引号。

```javascript
'key = "value"'
"It's a long journey"
```

上面两个都是合法的字符串。

如果要在单引号字符串的内部，使用单引号，就必须在内部的单引号前面加上反斜杠，用来转义。双引号字符串内部使用双引号，也是如此。

```javascript
'Did she say \'Hello\'?'
// "Did she say 'Hello'?"

"Did she say \"Hello\"?"
// "Did she say "Hello"?"
```

由于 HTML 语言的属性值使用双引号，所以很多项目约定 JavaScript 语言的字符串只使用单引号，本教程遵守这个约定。当然，只使用双引号也完全可以。重要的是坚持使用一种风格，不要一会使用单引号表示字符串，一会又使用双引号表示。

字符串默认只能写在一行内，分成多行将会报错。

```javascript
'a
b
c'
// SyntaxError: Unexpected token ILLEGAL
```

上面代码将一个字符串分成三行，JavaScript 就会报错。

如果长字符串必须分成多行，可以在每一行的尾部使用反斜杠。

```javascript
var longString = 'Long \
long \
long \
string';

longString
// "Long long long string"
```

上面代码表示，加了反斜杠以后，原来写在一行的字符串，可以分成多行书写。但是，输出的时候还是单行，效果与写在同一行完全一样。注意，反斜杠的后面必须是换行符，而不能有其他字符（比如空格），否则会报错。

连接运算符（`+`）可以连接多个单行字符串，将长字符串拆成多行书写，输出的时候也是单行。

```javascript
var longString = 'Long '
  + 'long '
  + 'long '
  + 'string';
```

如果想输出多行字符串，有一种利用多行注释的变通方法。

```javascript
(function () { /*
line 1
line 2
line 3
*/}).toString().split('\n').slice(1, -1).join('\n')
// "line 1
// line 2
// line 3"
```

上面的例子中，输出的字符串就是多行。

转义

反斜杠（\）在字符串内有特殊含义，用来表示一些特殊字符，所以又称为转义符。

需要用反斜杠转义的特殊字符，主要有下面这些。

- `\0`：null（`\u0000`）
- `\b`：后退键（`\u0008`）
- `\f`：换页符（`\u000C`）
- `\n`：换行符（`\u000A`）

- `\r`： 回车键（`\u000D`）
- `\t`： 制表符（`\u0009`）
- `\v`： 垂直制表符（`\u000B`）
- `\'`： 单引号（`\u0027`）
- `\"`： 双引号（`\u0022`）
- `\\`： 反斜杠（`\u005C`）

上面这些字符前面加上反斜杠，都表示特殊含义。

```javascript
console.log('1\n2')
// 1
// 2
```

上面代码中，`\n`表示换行，输出的时候就分成了两行。

反斜杠还有三种特殊用法。

（1）`\HHH`

反斜杠后面紧跟三个八进制数（`000`到`377`），代表一个字符。`HHH`对应该字符的 Unicode 码点，比如`\251`表示版权符号。显然，这种方法只能输出256种字符。

（2）`\xHH`

`\x`后面紧跟两个十六进制数（`00`到`FF`），代表一个字符。`HH`对应该字符的 Unicode 码点，比如`\xA9`表示版权符号。这种方法也只能输出256种字符。

（3）`\uXXXX`

`\u`后面紧跟四个十六进制数（`0000`到`FFFF`），代表一个字符。`XXXX`对应该字符的 Unicode 码点，比如`\u00A9`表示版权符号。

下面是这三种字符特殊写法的例子。

```javascript
'\251' // "©"
'\xA9' // "©"
'\u00A9' // "©"

'\172' === 'z' // true
'\x7A' === 'z' // true
'\u007A' === 'z' // true
```

如果在非特殊字符前面使用反斜杠，则反斜杠会被省略。

```javascript
'\a'
// "a"
```

上面代码中，`a`是一个正常字符，前面加反斜杠没有特殊含义，反斜杠会被自动省略。

如果字符串的正常内容之中，需要包含反斜杠，则反斜杠前面需要再加一个反斜杠，用来对自身转义。

```javascript
"Prev \\ Next"
// "Prev \ Next"
```

字符串与数组

字符串可以被视为字符数组，因此可以使用数组的方括号运算符，用来返回某个位置的字符（位置编号从0开始）。

```javascript
var s = 'hello';
s[0] // "h"
s[1] // "e"
s[4] // "o"

// 直接对字符串使用方括号运算符
'hello'[1] // "e"
```

如果方括号中的数字超过字符串的长度，或者方括号中根本不是数字，则返回`undefined`。

```javascript
'abc'[3] // undefined
'abc'[-1] // undefined
'abc'['x'] // undefined
```

但是，字符串与数组的相似性仅此而已。实际上，无法改变字符串之中的单个字符。

```javascript
var s = 'hello';

delete s[0];
s // "hello"

s[1] = 'a';
s // "hello"
```

```
s[5] = '!';
s // "hello"
```

上面代码表示，字符串内部的单个字符无法改变和增删，这些操作会默默地失败。

length 属性

`length`属性返回字符串的长度，该属性也是无法改变的。

```javascript
var s = 'hello';
s.length // 5

s.length = 3;
s.length // 5

s.length = 7;
s.length // 5
```

上面代码表示字符串的`length`属性无法改变，但是不会报错。

字符集

JavaScript 使用 Unicode 字符集。JavaScript 引擎内部，所有字符都用 Unicode 表示。

JavaScript 不仅以 Unicode 储存字符，还允许直接在程序中使用 Unicode 码点表示字符，即将字符写成`\uxxxx`的形式，其中`xxxx`代表该字符的 Unicode 码点。比如，`\u00A9`代表版权符号。

```javascript
var s = '\u00A9';
s // "©"
```

解析代码的时候，JavaScript 会自动识别一个字符是字面形式表示，还是 Unicode 形式表示。输出给用户的时候，所有字符都会转成字面形式。

```javascript
var f\u006F\u006F = 'abc';
foo // "abc"
```

上面代码中，第一行的变量名`foo`是 Unicode 形式表示，第二行是字面形式表示。JavaScript 会自动识别。

我们还需要知道，每个字符在 JavaScript 内部都是以16位（即2个字节）的 UTF-16 格式储存。也就是说，JavaScript 的单位字符长度固定为16位长度，即2个字节。

但是，UTF-16 有两种长度：对于码点在`U+0000`到`U+FFFF`之间的字符，长度为16位（即2个字节）；对于码点在`U+10000`到`U+10FFFF`之间的字符，长度为32位（即4个字节），而且前两个字节在`0xD800`到`0xDBFF`之间，后两个字节在`0xDC00`到`0xDFFF`之间。举例来说，码点`U+1D306`对应的字符为`≡，`它写成 UTF-16 就是`0xD834 0xDF06`。

JavaScript 对 UTF-16 的支持是不完整的，由于历史原因，只支持两字节的字符，不支持四字节的字符。这是因为 JavaScript 第一版发布的时候，Unicode 的码点只编到`U+FFFF`，因此两字节足够表示了。后来，Unicode 纳入的字符越来越多，出现了四字节的编码。但是，JavaScript 的标准此时已经定型了，统一将字符长度限制在两字节，导致无法识别四字节的字符。上一节的那个四字节字符`≡`，浏览器会正确识别这是一个字符，但是 JavaScript 无法识别，会认为这是两个字符。

```javascript
'≡'.length // 2
```

上面代码中，JavaScript 认为`≡`的长度为2，而不是1。

总结一下，对于码点在`U+10000`到`U+10FFFF`之间的字符，JavaScript 总是认为它们是两个字符（`length`属性为2）。所以处理的时候，必须把这一点考虑在内，也就是说，JavaScript 返回的字符串长度可能是不正确的。

Base64 转码

有时，文本里面包含一些不可打印的符号，比如 ASCII 码0到31的符号都无法打印出来，这时可以使用 Base64 编码，将它们转成可以打印的字符。另一个场景是，有时需要以文本格式传递二进制数据，那么也可以使用 Base64 编码。

所谓 Base64 就是一种编码方法，可以将任意值转成 0～9、A～Z、a-z、`+`和`/`这64个字符组成的可打印字符。使用它的主要目的，不是为了加密，而是为了不出现特殊字符，简化程序的处理。

JavaScript 原生提供两个 Base64 相关的方法。

- `btoa()`：任意值转为 Base64 编码
- `atob()`：Base64 编码转为原来的值

```javascript
var string = 'Hello World!';
btoa(string) // "SGVsbG8gV29ybGQh"
atob('SGVsbG8gV29ybGQh') // "Hello World!"
```

```

注意，这两个方法不适合非 ASCII 码的字符，会报错。

```javascript
btoa('你好') // 报错
```

要将非 ASCII 码字符转为 Base64 编码，必须中间插入一个转码环节，再使用这两个方法。

```javascript
function b64Encode(str) {
  return btoa(encodeURIComponent(str));
}

function b64Decode(str) {
  return decodeURIComponent(atob(str));
}

b64Encode('你好') // "JUU0JUJEJUEwJUU1JUE1JUJE"
b64Decode('JUU0JUJEJUEwJUU1JUE1JUJE') // "你好"
```

参考链接

- Mathias Bynens, [JavaScript's internal character encoding: UCS-2 or UTF-16?](http://mathiasbynens.be/notes/javascript-encoding)
- Mathias Bynens, [JavaScript has a Unicode problem](http://mathiasbynens.be/notes/javascript-unicode)
- Mozilla Developer Network, [Window.btoa](https://developer.mozilla.org/en-US/docs/Web/API/Window.btoa)
```