有用的字符串方法

现在我们已经了解了字符串的基础知识，让我们更进一步，开始思考我们可以使用内置方法对字符串执行哪些有用的操作，例如查找文本字符串的长度、连接和拆分字符串，将字符串中的一个字符替换为另一个字符，等等。

先决条件：	基本的计算机知识，对 HTML 和 CSS 的基本理解，对 JavaScript 是什么的理解。
客观的：	了解字符串是对象，并学习如何使用这些对象上可用的一些基本方法来操作字符串。

作为对象的字符串

大多数东西都是 JavaScript 中的对象。创建字符串时，例如使用

```
const string = 'This is my string';
```

您的变量成为一个字符串对象实例，因此具有大量可用的属性和方法。如果您转到 `String` 对象页面并查看页面一侧的列表，您可以看到这一点！

现在，在你的大脑开始融化之前，别担心！您真的不需要在学习旅程的早期就了解其中的大部分内容。但是有一些您可能会经常使用的，我们将在此处查看。

让我们在浏览器开发人员控制台中输入一些示例。

查找字符串的长度

这很简单——你使用这个 `length` 属性。尝试输入以下行：

```
const browserType = 'mozilla';
browserType.length;
```

这应该返回数字 7，因为“mozilla”有 7 个字符长。这很有用，原因有很多；例如，您可能想要找到一系列名称的长度，以便按长度顺序显示它们，或者让用户知道他们在表单字段中输入的用户名如果超过一定长度就太长了.

检索特定的字符串字符

在相关说明中，您可以使用方括号表示法返回字符串中的任何字符——这意味着您 `[]` 在变量名称的末尾包含方括号 ( )。在方括号内，包括要返回的字符数，例如要检索第一个字母，您可以这样做：

```
browserType[0];
```

请记住：计算机从 0 开始计数，而不是 1!

*要检索任何*字符串的最后一个字符，我们可以使用以下行，将此技术与 `length` 我们在上面查看的属性相结合：

```
browserType[browserType.length-1];
```

字符串“mozilla”的长度是7，但是因为从0开始计数，所以最后一个字符的位置是6；使用 `length-1` 让我们得到最后一个字符。

测试字符串是否包含子字符串

有时你会想知道一个较小的字符串是否存在于一个较大的字符串中（我们通常说*一个子字符串是否存在于一个字符串中*）。这可以使用 方法来完成 `includes()`，该方法采用单个参数——您要搜索的子字符串。

`true` 如果字符串包含子字符串，则返回，否则返回 `false` 。

```
const browserType = 'mozilla';
```

```
if (browserType.includes('zilla')) {
  console.log('Found zilla!');
} else {
  console.log('No zilla here!');
}
```

通常您会想知道字符串是以特定子字符串开头还是结尾。这是一个足够普遍的需求，为此有两种特殊的方法： `startsWith()` 和 `endsWith()` ：

```
const browserType = 'mozilla';

if (browserType.startsWith('zilla')) {
  console.log('Found zilla!');
} else {
  console.log('No zilla here!');
}
```

```
const browserType = 'mozilla';

if (browserType.endsWith('zilla')) {
  console.log('Found zilla!');
} else {
  console.log('No zilla here!');
}
```

查找子字符串在字符串中的位置

您可以使用该方法找到较大字符串中子字符串的位置 `indexOf()` 。此方法有两个参数——您要搜索的子字符串，以及一个指定搜索起点的可选参数。

如果字符串包含子字符串， `indexOf()` 则返回子字符串第一次出现的索引。如果字符串不包含子字符串， `indexOf()` 则返回 `-1` 。

```
const tagline = 'MDN - Resources for developers, by developers';
console.log(tagline.indexOf('developers')); // 20
```

从 开始 `0` ，如果您从字符串的开头计算字符数（包括空格），则子字符串的第一次出现 `"developers"` 在索引处 `20` 。

```
console.log(tagline.indexOf('x')); // -1
```

另一方面，这会返回， `-1` 因为该字符 `x` 不存在于字符串中。

那么既然您知道如何找到子串的第一次出现，那么您如何着手寻找后续出现的地方呢？您可以通过将大于上一次出现的索引的值作为该方法的第二个参数传递来实现。

```
const firstOccurrence = tagline.indexOf('developers');
const secondOccurrence = tagline.indexOf('developers',
firstOccurrence + 1);

console.log(firstOccurrence); // 20
console.log(secondOccurrence); // 35
```

这里我们告诉方法搜索 `"developers"` 从 index `21` (`firstOccurrence + 1`) 开始的子字符串，并返回 index `35` 。

从字符串中提取子字符串

您可以使用该方法从字符串中提取子字符串 `slice()` 。你通过它：

- 开始提取的索引
- 停止提取的索引。这是排他性的，意味着该索引处的字符不包含在提取的子字符串中。

例如：

```
const browserType = 'mozilla';
console.log(browserType.slice(1, 4)); // "ozi"
```

索引处的字符 1 是 `"o"` ， 索引 4 处的字符是 `"l"` 。所以我们提取所有开始 `"o"` 和结束之前的字符 `"l"` ， 给我们 `"ozi"` 。

如果您知道要提取字符串中某个字符之后的所有剩余字符，则不必包含第二个参数。相反，您只需要包括要从中提取字符串中剩余字符的字符位置。尝

试以下操作：

```
browserType.slice(2); // "zilla"
```

这将返回 "zilla" — 这是因为 2 的字符位置是字母 "z"，并且因为您没有包含第二个参数，所以返回的子字符串是字符串中的所有剩余字符。

> **注意：** slice() 还有其他选项；研究该 slice() 页面，看看您还能找到什么。

改变大小写

字符串方法 toLowerCase() 和 toUpperCase() 获取一个字符串并将所有字符分别转换为小写或大写。例如，如果您想在将所有用户输入的数据存储到数据库之前对其进行规范化，这将很有用。

让我们尝试输入以下几行，看看会发生什么：

```
const radData = 'My NaMe Is MuD';
console.log(radData.toLowerCase());
console.log(radData.toUpperCase());
```

更新字符串的一部分

您可以使用该方法将字符串中的一个子字符串替换为另一个子字符串 replace() 。

在这个例子中，我们提供了两个参数——我们想要替换的字符串，以及我们想要替换它的字符串：

```
const browserType = 'mozilla';
const updated = browserType.replace('moz','van');

console.log(updated);      // "vanilla"
console.log(browserType);  // "mozilla"
```

请注意 `replace()`，与许多字符串方法一样，它不会更改调用它的字符串，而是返回一个新字符串。如果要更新原始 `browserType` 变量，则必须执行以下操作：

```
let browserType = 'mozilla';
browserType = browserType.replace('moz','van');

console.log(browserType);  // "vanilla"
```

另请注意，我们现在必须声明 `browserType` using `let`，而不是 `const`，因为我们正在重新分配它。

请注意，`replace()` 在这种形式中，只会更改第一次出现的子字符串。如果你想改变所有的事件，你可以使用 `replaceAll()`：

```
let quote = 'To be or not to be';
quote = quote.replaceAll('be','code');

console.log(quote);  // "To code or not to code"
```

主动学习实例

在本节中，我们将让您尝试编写一些字符串操作代码。在下面的每个练习中，我们都有一个字符串数组，以及一个处理数组中每个值并将其显示在项目符号列表中的循环。您现在不需要了解数组或循环——这些将在以后的文章中进行解释。在每种情况下，您需要做的就是编写代码，以我们希望的格式输出字符串。

每个示例都带有一个“重置”按钮，如果您犯了错误并且无法使其再次运行，您可以使用它来重置代码，还有一个“显示解决方案”按钮，您可以在遇到问题时按下以查看可能的答案真的卡住了。

过滤问候消息

在第一个练习中，我们将从简单开始 — 我们有一组贺卡消息，但我们想对它们进行排序以仅列出圣诞节消息。我们希望您在结构中填写条件测试 `if ()` 以测试每个字符串，并且仅在它是圣诞节消息时才将其打印在列表中。

想一想如何测试每种情况下的消息是否是圣诞节消息。所有这些消息中都存在什么字符串，您可以使用什么方法来测试它是否存在?

Live output

- Happy Birthday!
- Merry Christmas my love
- A happy Christmas to all the family
- You're all I want for Christmas
- Get well soon

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
const list =
document.querySelector('.output ul');
list.innerHTML = '';
const greetings = ['Happy Birthday!',
                   'Merry Christmas my
love',
                   'A happy Christmas to
all the family',
                   'You\'re all I want
for Christmas',
                   'Get well soon'];

for (const greeting of greetings) {
  // Your conditional test needs to go
inside the parentheses
  // in the line below, replacing
what's currently there
  if (greeting) {
    const listItem =
```

Reset Show solution

固定大小写

在这个练习中，我们有英国的城市名称，但大小写都搞砸了。我们希望您将它们更改为小写，除了首字母大写。执行此操作的一个好方法是：

1. 将变量中包含的整个字符串转换 `city` 为小写并将其存储在新变量中。
2. 在此新变量中获取字符串的第一个字母并将其存储在另一个变量中。
3. 以这个最新的变量为子串，将小写字符串的首字母替换为大写的小写字符串首字母。将此替换过程的结果存储在另一个新变量中。

4. 将变量的值更改 `result` 为等于最终结果，而不是 `city`。

> **注意：**提示——字符串方法的参数不必是字符串文字；它们也可以是变量，甚至是调用方法的变量。

Live output

- lonDon
- ManCHESTer
- BiRmiNGHAM
- liVERpoOL

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
const list =
document.querySelector('.output ul');
list.innerHTML = '';
const cities = ['lonDon', 'ManCHESTer',
'BiRmiNGHAM', 'liVERpoOL'];

for (const city of cities) {
  // write your code just below here

  const result = city;
  const listItem =
document.createElement('li');
  listItem.textContent = result;
  list.appendChild(listItem);
}
```

Reset Show solution

用旧零件制作新琴弦

在最后一个练习中，该数组包含一堆字符串，其中包含有关英格兰北部火车站的信息。这些字符串是包含三个字母站代码的数据项，后面是一些机器可读的数据，然后是分号，然后是人类可读的站名。例如：

```
MAN675847583748sjt567654;Manchester Piccadilly
```

我们要提取车站代码和车站名称，并将它们放在一个具有以下结构的字符串中：

```
MAN: Manchester Piccadilly
```

我们建议这样做：

1. 提取三个字母的站点代码并将其存储在一个新变量中。
2. 查找分号的字符索引号。
3. 使用分号字符索引号作为参考点提取人类可读的站点名称，并将其存储在新变量中。
4. 连接两个新变量和一个字符串文字以构成最终字符串。
5. 将变量的值更改 `result` 为最终字符串，而不是 `station` .

Live output

- MAN675847583748sjt567654;Manchester Piccadilly
- GNF576746573fhdg4737dh4;Greenfield
- LIV5hg65hd737456236dch46dg4;Liverpool Lime Street
- SYB4f65hf75f736463;Stalybridge
- HUD5767ghtyfyr4536dh45dg45dg3;Huddersfield

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
const list = document.querySelector('.output
ul');
list.innerHTML = '';
const stations =
['MAN675847583748sjt567654;Manchester
Piccadilly',

'GNF576746573fhdg4737dh4;Greenfield',

'LIV5hg65hd737456236dch46dg4;Liverpool Lime
Street',

'SYB4f65hf75f736463;Stalybridge',

'HUD5767ghtyfyr4536dh45dg45dg3;Huddersfield'];

for (const station of stations) {
  // write your code just below here
```

Reset Show solution

测试你的技能！

您已读完本文，但您还记得最重要的信息吗？在继续之前，您可以找到一些进一步的测试来验证您是否保留了这些信息——请参阅测试您的技能：字符串。

结论

您无法回避这样一个事实，即能够在编程中处理单词和句子非常重要——尤其是在 JavaScript 中，因为网站都是关于与人交流的。本文为您提供了目前需要了解的有关操作字符串的基础知识。当您将来进入更复杂的主题时，这应该对您很有帮助。接下来，我们来看一下短期内需要重点关注的最后一种主要数据类型——数组。

此页面最后修改于 2023 年 2 月 26 日由MDN 贡献者提供。