

此页面由社区从英文翻译而来。了解更多并加入 MDN Web Docs 社区。



在编写网页和应用程序时，你最想做的事情之一是以某种方式操纵文档结构。这通常是通过使用文档对象模型（DOM）来实现的，这是一套用于控制 HTML 和样式信息的 API，大量使用了 [Document](#) 对象。在这篇文章中，我们将详细了解如何使用 DOM，以及其他一些有趣的 API，它们可以以有趣的方式改变你的环境。

预备条件：	基础的计算机常识，基本了解 HTML、CSS 和 JavaScript，包括 JavaScript 对象。
目标：	熟悉核心 DOM API，以及其他和 DOM 与文档操作相关的常见 API。

web 浏览器的重要部分

web 浏览器是非常复杂的软件，有许多活动部件，其中许多部件不能由 web 开发者用 JavaScript 控制或操纵。你可能认为这种限制是件坏事，但浏览器作出这些限制是有原因的，主要是围绕安全问题。想象一下，如果网站可以访问你存储的密码或其他敏感信息，且像你本人一样登录那些网站，会发生什么？

尽管有这些限制，Web API 仍然给我们提供了大量的功能，使我们能够用网页做很多事情。在你的代码中，有几个非常明显的部分你会经常参考。考虑下图，它代表了浏览器中直接参与浏览网页的主要部分：



- 窗口（window）是载入网页的浏览器标签；在 JavaScript 中，它由 [Window](#) 对象表示。使用这个对象上的方法，你可以做一些事情，比如返回窗口的大小（见 [Window.innerWidth](#) 和 [Window.innerHeight](#)），操作加载到窗口的文档，在客户端存储该文档的特定数据（例如使用本地数据库或其他存储机制），为当前窗口附加一个[事件处理器](#)等。
- 导航器（navigator）在网络上出现时，代表浏览器的状态和身份（即用户代理）。在 JavaScript 中，它由 [Navigator](#) 对象表示。你可以用这个对象来检索用户的首选语言、用户网络摄像头的媒体流等信息。
- 文档（document，在浏览器中用 DOM 表示）是加载到窗口的实际页面，在 JavaScript 中，它由 [Document](#) 对象表示。你可以使用这个对象来返回和操作构成文档的 HTML 和 CSS 的信息，例如，在 DOM 中获得一个元素的引用，改变其文本内容，对其应用新的样式，创建新的元素并将其作为子元素添加到当前元素中，甚至完全删除它。

在本文中，我们主要关注操作文档的方法，但是也会稍微关注一下其他有用的部分。

文档对象模型

目前在你的每一个浏览器标签中加载的文档是由一个文档对象模型表示的。这是一个由浏览器创建的“树状结构”表示法，使 HTML 结构能够被编程语言轻松访问。例如，浏览器本身在渲染页面

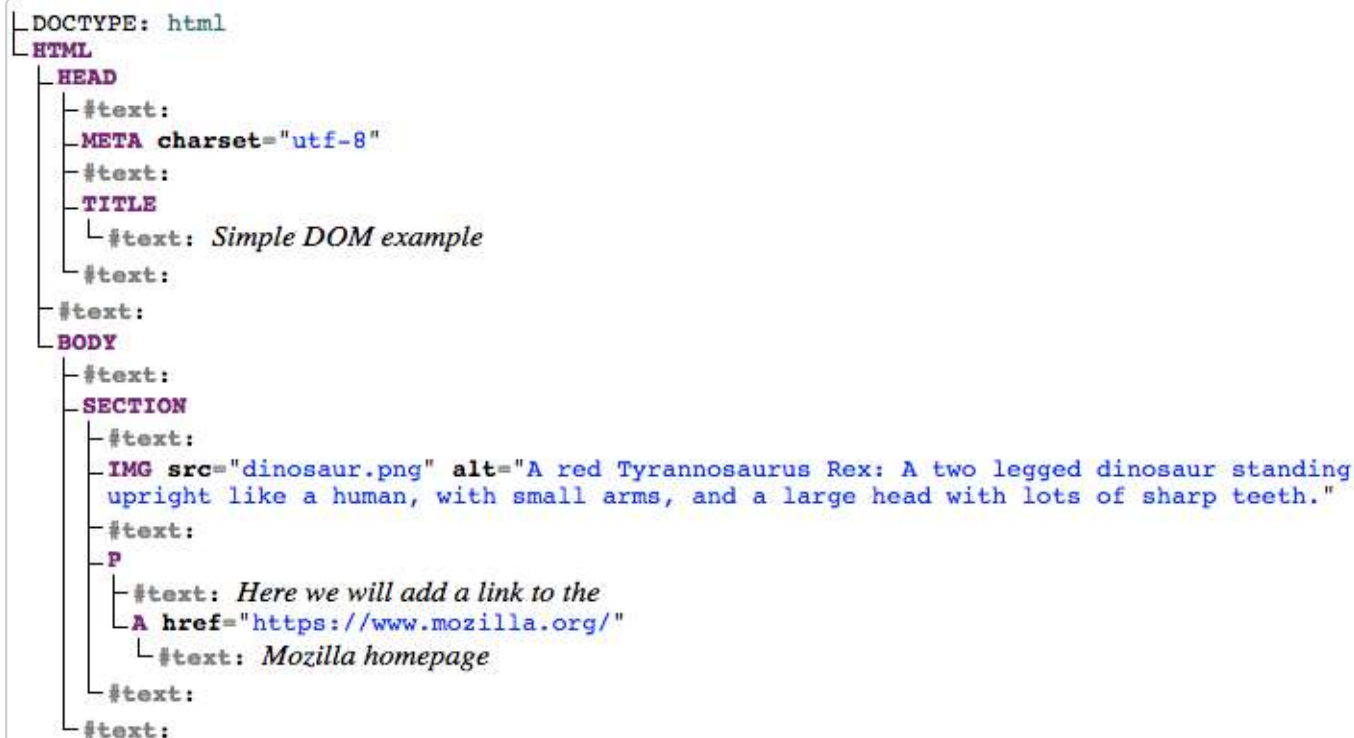
时使用它将样式和其他信息应用于正确的元素，而开发者可以在页面渲染后用 JavaScript 来操作 DOM。

我们在 [dom-example.html](#) （[也可以看看在线的例子](#)）上创建了一个简单的示例页面。试着在浏览器中打开它。这是一个非常简单的页面，包含一个 `<section>` 元素，里面有一张图片，还有一个带链接的段落。该 HTML 源代码看起来像这样：

HTML

```
<!doctype html>
<html lang="en-US">
  <head>
    <meta charset="utf-8" />
    <title>Simple DOM example</title>
  </head>
  <body>
    <section>
      
      <p>
        Here we will add a link to the
        <a href="https://www.mozilla.org/">Mozilla homepage</a>
      </p>
    </section>
  </body>
</html>
```

其 DOM 树如下所示：



备注： 这个 DOM 树状图是用 Ian Hickson 的 [Live DOM viewer](#) 生成的。

树上的每个条目都被称为**节点**。你可以在上图中看到，一些节点代表元素（标识为 HTML、HEAD、META 等），另一些代表文本（标识为 #text）。还有[其他类型的节点](#)，但这些都是你会遇到的主要类型。

节点也通过它们在树中相对于其他节点的位置来指代：

- **根节点:** 树中顶层节点，在 HTML 的情况下，总是一个 HTML 节点（其他标记词汇，如 SVG 和定制 XML 将有不同的根元素）。
- **子节点:** 直接位于另一个节点内的节点。例如上面例子中，IMG 是 SECTION 的子节点。
- **后代节点:** 位于另一个节点内任意位置的节点。例如上面例子中，IMG 是 SECTION 的子节点，也是一个后代节点。IMG 不是 BODY 的子节点，因为它在树中比 BODY 低了两级，但它是 BODY 的后代之一。
- **父节点:** 里面有另一个节点的节点。例如上面的例子中 BODY 是 SECTION 的父节点。
- **兄弟节点:** DOM 树中位于同一等级的节点。例如上面例子中，IMG 和 P 是兄弟。

在使用 DOM 之前，熟悉这些术语是很有用的，因为你会遇到的一些代码术语都会用到它们。如果你学习过 CSS，你可能也会遇到它们（例如，子孙选择器、子选择器）。

动手练习：基本的 DOM 操作

要开始学习 DOM 操作，我们先做一个实际的例子。

1. 将 [dom-example.html 页面](#) 和与之配套的图片 拷贝到本地。
2. 在闭合的 `</body>` 标签上面加入 `<script></script>` 元素。
3. 要操作 DOM 内的元素，首先需要选择它，并将它的引用存储在一个变量中。在 script 元素中，添加下列代码行：

JS

```
const link = document.querySelector("a");
```

4. 现在我们已经将元素引用存储在一个变量中，我们可以开始使用可用的属性和方法来操作它（它们定义在 `<a>` 元素的 [HTMLAnchorElement \(en-US\)](#) 接口上，它继承于更一般的父接口 [HTMLElement](#)，以及 [Node](#)——它代表 DOM 中所有节点）。首先，让我们通过更新 [Node.textContent](#) 属性的值来改变链接中的文本。在前一行下面添加以下内容：

JS

```
link.textContent = "Mozilla Developer Network";
```

5. 我们也能修改链接指向的 URL，使得它被点击时不会走向错误的位置。在底部再次加入下列代码：

JS

```
link.href = "https://developer.mozilla.org";
```

请注意，就像 JavaScript 中所做的那样，有许多方法可以选择一个元素并将其引用存储在一个变量中。[Document.querySelector\(\)](#) 是推荐的现代方法。它很方便，因为它允许你使用 CSS 选择器来选择元素。上面的 `querySelector()` 调用将匹配文档中出现的第一个 `<a>` 元素。如果你想对多个元素进行匹配和操作，你可以使用 [Document.querySelectorAll\(\)](#)，它可以匹配文档中与选择器相匹配的每个元素，并将它们的引用存储在一个叫做 [NodeList](#) 的数组对象中。

对于获取元素引用，还有一些更旧的方法，如：

- [Document.getElementById\(\)](#)，选择一个 id 属性值已知的元素，例如 `<p id="myId">My paragraph</p>`。ID 作为参数传递给函数，即 `const elementRef = document.getElementById('myId')`。
- [Document.getElementsByTagName\(\)](#)，返回页面中包含的所有已知类型元素的数组。如 `<p>`、`<a>` 等。元素类型作为参数传递给函数，即 `const elementRefArray =`

```
document.getElementsByTagName('p')。
```

这两种方法在旧的浏览器中比现代方法如 `querySelector()` 更好用，但没有那么方便。看一看，看看你还能找到什么其他的方法！

创建并放置新的节点

以上只是让你稍微尝试一下你可以做的事情，让我们进一步看看我们可以怎样来创建新的元素。

1. 回到当前的例子，我们先获取到 [<section>](#) 元素的引用。在已有 script 中添加下列代码（其他代码也同样处理）：

JS

```
const sect = document.querySelector("section");
```

2. 现在用 [Document.createElement\(\)](#) 创建一个新的段落，用与之前相同的方法赋予相同的文本：

JS

```
const para = document.createElement("p");  
para.textContent = "We hope you enjoyed the ride.";
```

3. 现在可以用 [Node.appendChild\(\)](#) 方法在后面追加新的段落：

JS

```
sect.appendChild(para);
```

4. 最后，在内部链接的段落中添加文本节点，完美的结束句子。首先我们要使用 [Document.createTextNode\(\)](#) 创建一个文本节点：

JS

```
const text = document.createTextNode(  
  " – the premier source for web development knowledge.",  
);
```

5. 现在获取内部连接的段落的引用，并把文本节点附加到这个节点上：

JS

```
const linkPara = document.querySelector("p");  
linkPara.appendChild(text);
```

这是给 DOM 添加节点要做的大部分工作——在构建动态界面时，你将做大量使用这些方法（我们在后面可以看到一些例子）。

移动和删除元素

也许有时候你想移动或从 DOM 中删除节点，这是完全可能的。

如果你想把具有内部链接的段落移到 section 的底部，简单的做法是：

JS

```
sect.appendChild(linkPara);
```

这样可以把段落下移到 section 的底部。你可能认为它会产生第二个副本，但事实并非如此——`linkPara` 是对该段落唯一副本的引用。如果你想复制并添加它，你需要使用 [Node.cloneNode\(\)](#) 来代替。

删除节点也非常的简单，至少，你拥有要删除的节点和其父节点的引用。在当前情况下，我们只要使用 [Node.removeChild\(\)](#) 即可，如下：

JS

```
sect.removeChild(linkPara);
```

要删除一个仅基于自身引用的节点可能稍微有点复杂，这也是很常见的。你可以使用 [Element.remove\(\)](#)：

JS

```
linkPara.remove();
```

此方法在较旧的浏览器中不受支持，它们没有方法告诉一个节点删除自己，所以你必须这样做：

JS

```
linkPara.parentNode.removeChild(linkPara);
```

把上述代码行加到你的代码中去。

操作样式

通过 JavaScript 以不同的方式来操作 CSS 样式是可能的。

首先，你可以使用 [Document.styleSheets](#) 来获得一个附加在文档上的所有样式表的列表，它返回一个包含 [CSSStyleSheet](#) 对象的类数组。然后你就可以根据需要添加/删除样式了。然而，我们打算对这些功能进行扩展，因为它们是一种有点过时的、难以操作样式的方式。还有更多更简单的方法。

第一种方法是直接将内联样式添加到你想动态样式的元素上。这是通过 [HTMLElement.style](#) 属性实现的，它包含了文档中每个元素的内联样式信息。你可以设置这个对象的属性来直接更新元素样式。

1. 作为示例，把下面的代码行加到我们的例子中：

JS

```
para.style.color = "white";
para.style.backgroundColor = "black";
para.style.padding = "10px";
para.style.width = "250px";
para.style.textAlign = "center";
```

2. 重新载入页面，你将看到样式已经应用到段落中。如果在浏览器的 [Page Inspector/DOM inspector](#) 中查看段落，你会看到这些代码的确为文档添加了内联样式：

HTML

```
<p
  style="color: white; background-color: black; padding: 10px; width: 250px; text-align:
center;">
  We hope you enjoyed the ride.
</p>
```

备注： 请注意，CSS 样式的 JavaScript 属性版本是用小驼峰命名法书写的，而 CSS 版本是连字符的（例如，`backgroundColor` 对 `background-color`）。确保你不要把这些混为一谈，否则将无法工作。

还有一种在你的文档上动态操作样式的常见方法，我们现在就来看看。

1. 删除之前添加到 JavaScript 中的五行代码。
2. 在 HTML 的 [<head>](#) 中添加下列代码：


```
<style>
  .highlight {
    color: white;
    background-color: black;
    padding: 10px;
    width: 250px;
    text-align: center;
  }
</style>
```

3. 现在我们改为使用 HTML 操作的常用方法——[Element.setAttribute\(\)](#)——它接受两个参数：想在元素上设置的属性、要为其设置的值。在这种情况下，我们在段落中设置类名为 highlight：

JS

```
para.setAttribute("class", "highlight");
```

4. 刷新你的页面，不会看到任何改变——CSS 仍然应用于该段落，但这次是通过给它一个类，由我们的 CSS 规则选择，而不是作为内联 CSS 样式。

两种方式各有优缺点，选择哪种取决于你自己。第一种方法需要较少的设置，适合于简单的使用，而第二种方法更纯粹（混合 CSS、JavaScript 和内联样式通常不是一种好的实践，而该方法不会产生这些）。当你开始构建更大和更多的应用程序时，你可能会更多地开始使用第二种方法，但这真的取决于你。

在这一点上，我们还没有做任何有用的事！使用 JavaScript 来创建静态内容是没有意义的，你还不如直接把它写进你的 HTML，而不使用 JavaScript。它比 HTML 更复杂，而且用 JavaScript 创建你的内容也有其他附带的问题（比如不能被搜索引擎阅读）。

在接下来的几节中我们将看看 DOM API 一些更实际的用途。

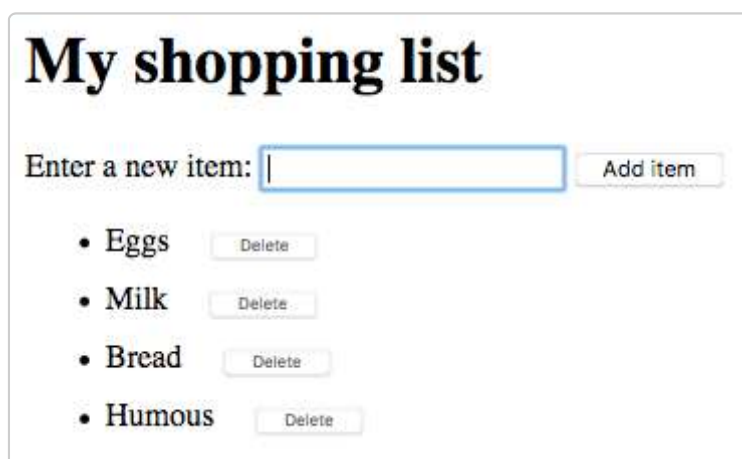
备注： 你可以在 GitHub 上找到我们的 [dom-example.html](#) 的完成版 演示（也可以看看它的在线运行版本 ）。

动手练习：一个动态的购物单

在这个挑战中，我们想做一个简单的购物清单例子，允许你使用表单输入和按钮动态地将物品添加到清单中。当你在输入中添加一个项目并按下按钮时：

- 购物项应该出现在清单中。
- 每个购物项都应该给出一个按钮，可以按下按钮从清单中删除该项。
- 输入框应该是清空的，并已经聚焦，为你准备好输入另一个项。

完成后的演示程序看起来有点像这样的：



My shopping list

Enter a new item: Add item

- Eggs Delete
- Milk Delete
- Bread Delete
- Humous Delete

要完成实验，要按照下面的步骤，确保购物单的行为如上所述。

1. 首先，下载 [shopping-list.html](#) 文件，并存入本地。你会看到它有一些极小的 CSS，一个带有 label、input 和 button 的 div 和一个空的列表以及 `<script>` 元素。要添加的所有程序都在 script 里面。
2. 创建三个变量来保存列表（``）、`<input>` 和 `<button>` 元素的引用。
3. 创建一个函数响应点击按钮。
4. 在函数体内，开始要在一个变量中存储输入框的当前值。
5. 然后，为输入框元素设置空字符串 '' 以清空它。
6. 创建三个新元素：一个列表元素（``）、`` 和 `<button>`，并把它们存入变量之中。
7. 将 span 和按钮附加到列表元素的子节点。
8. 把之前保存的输入框元素的值设置为 span 的文本内容，按钮的文本内容设置为“Delete”。
9. 将列表元素附加到列表的子节点中。
10. 为删除按钮绑定事件处理程序。当点击按钮时，删除它所在的整个列表元素（`...`）。

11. 最后，使用 [focus\(\)._\(en-US\)](#) 方法聚焦输入框准备输入下一个购物项。

备注： 如果你卡住了，请查看[完成的购物清单](#)（[查看其在线版本](#)）。

总结

我们已经结束了对文档和 DOM 操作的研究。在这一点上，你应该明白 Web 浏览器在控制文档和用户网络体验的其他方面有哪些重要部分。最重要的是，你应该明白什么是文档对象模型，以及如何操作它来创建有用的功能。

参见

你还可以使用更多的特性来操作文档，查看这些参考，看看你能发现些什么：

- [Document](#)
- [Window](#)
- [Node](#)
- [HTMLElement](#)、[HTMLInputElement](#)、[HTMLImageElement](#) 等

(请参阅我们的 [Web API 索引](#)，了解 MDN 上记录的 Web API 的完整列表！)

Help improve MDN

Was this page helpful to you?

Yes

No

[Learn how to contribute.](#)

This page was last modified on 2023年7月24日 by [MDN contributors](#).

