操纵文件

在编写网页和应用程序时，您最常要做的事情之一就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) 来完成的，DOM 是一组用于控制 HTML 和大量使用对象的样式信息的 API `Document` 。在本文中，我们将详细了解如何使用 DOM，以及可以以有趣的方式改变您的环境的其他一些有趣的 API。

先决条件：	基本的计算机知识，对 HTML、CSS 和 JavaScript 的基本理解——包括 JavaScript 对象。
客观的：	熟悉核心 DOM API 以及通常与 DOM 和文档操作相关的其他 API。

Web 浏览器的重要组成部分

Web 浏览器是非常复杂的软件，有很多活动部件，其中许多无法由使用 JavaScript 的 Web 开发人员控制或操纵。你可能认为这样的限制是一件坏事，但浏览器被锁定是有充分理由的，主要是围绕安全性。想象一下，如果一个网站可以访问您存储的密码或其他敏感信息，并像您一样登录网站?

尽管有这些限制，Web API 仍然使我们能够访问许多功能，使我们能够使用网页做很多事情。您会在代码中经常引用一些非常明显的位 — 考虑下图，它表示直接参与查看网页的浏览器的主要部分：

- 窗口是加载网页的浏览器选项卡；this 在 JavaScript 中由对象表示 `Window` 。使用此对象上可用的方法，您可以执行诸如返回窗口大小（参见 `Window.innerWidth` 和 `Window.innerHeight` ）、操作加载到该窗口中的文档、在客户端存储特定于该文档的数据（例如使用本地数据库或其他存储机制）等操作)、将事件处理程序附加到当前窗口等。
- 导航器代表浏览器（即用户代理）在 Web 上的状态和身份。在 JavaScript 中，这由对象表示 `Navigator` 。您可以使用此对象检索用户的首选语言、来自用户网络摄像头的媒体流等内容。
- 文档（在浏览器中由 DOM 表示）是加载到窗口中的实际页面，在 JavaScript 中由对象表示 `Document` 。您可以使用此对象返回和操作有关构成文档的 HTML 和 CSS 的信息，例如获取对 DOM 中元素的引用、更改其文本内容、对其应用新样式、创建新元素并将它们添加到当前元素作为子元素，甚至完全删除它。

在本文中，我们将主要关注如何操作文档，但除此之外，我们还将展示一些其他有用的内容。

文档对象模型

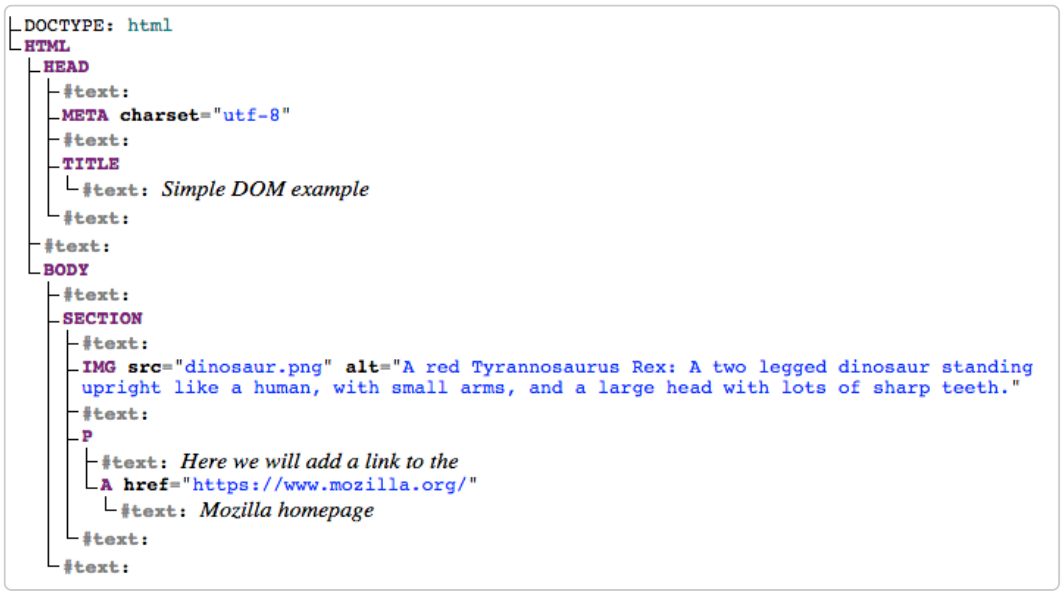
当前在每个浏览器选项卡中加载的文档由一个文档对象模型表示。这是一种由浏览器创建的“树结构”表示，它使 HTML 结构可以很容易地被编程语言访问——例如，浏览器本身使用它来在呈现页面时将样式和其他信息应用于

正确的元素，而开发人员就像您可以在页面呈现后使用 JavaScript 操作 DOM 一样。

我们在dom-example.html 上创建了一个简单的示例页面（也可以实时查看 ）。尝试在浏览器中打开它——这是一个非常简单的页面，其中包含一个 `<section>` 元素，您可以在其中找到一张图片，以及一个带有链接的段落。HTML 源代码如下所示：

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8" />
    <title>Simple DOM example</title>
  </head>
  <body>
    <section>
      <img
        src="dinosaur.png"
        alt="A red Tyrannosaurus Rex: A two legged dinosaur
standing upright like a human, with small arms, and a large head
with lots of sharp teeth." />
      <p>
        Here we will add a link to the
        <a href="https://www.mozilla.org/">Mozilla homepage</a>
      </p>
    </section>
  </body>
</html>
```

另一方面，DOM 看起来像这样：



> **注意：**此 DOM 树图是使用 Ian Hickson 的Live DOM 查看器创建的。

树中的每个条目称为一个**节点**。您可以在上图中看到一些节点代表元素（标识为 HTML 、等），其他节点代表文本（标识为）。还有其他类型的节点，但这些是您将遇到的主要节点。 HEAD META #text

节点也通过它们在树中相对于其他节点的位置来引用：

- **根节点**：树中的顶部节点，在 HTML 的情况下始终是节点 HTML （其他标记词汇表，如 SVG 和自定义 XML 将具有不同的根元素）。
- **子节点**：*直接*位于另一个节点内的节点。例如，在上面的例子中 IMG 是的孩子。 SECTION
- **后代节点**：另一个节点内*任何位置的*节点。比如上例中 IMG 是的孩子 SECTION ，也是后代。 IMG 不是 的孩子 BODY ，因为它在树中比它低两层，但它是 的后代 BODY 。
- **父节点**：内部有另一个节点的节点。例如，是上例中 BODY 的父节点。 SECTION
- **兄弟节点**：位于 DOM 树中同一级别的节点。例如， IMG 和 P 是上例中的兄弟姐妹。

在使用 DOM 之前熟悉这些术语很有用，因为您将遇到的许多代码术语都会用到它们。如果您学习过 CSS（例如，后代选择器、子选择器），您可能也遇到过它们。

主动学习：基本的 DOM 操作

要开始学习 DOM 操作，让我们从一个实际示例开始。

1. 获取dom-example.html 页面 的本地副本以及随附的图像。
2. `<script></script>` 在结束标记的正上方添加一个元素 `</body>` 。
3. 要在 DOM 中操作一个元素，您首先需要选择它并将对它的引用存储在一个变量中。在您的脚本元素中，添加以下行：

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

4. 现在我们将元素引用存储在一个变量中，我们可以开始使用它可用的属性和方法来操作它（这些是在接口上定义的，就像在 `HTMLAnchorElement` 元素的情况下 `<a>` ，它的更通用的父接口 `HTMLElement` ，以及 `Node` - 它代表中的所有节点一个 DOM）。首先，让我们通过更新 `Node.textContent` 属性的值来更改链接内的文本。在上一行下面添加以下行：

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

5. 我们还应该更改链接指向的 URL，以便在单击时不会转到错误的位置。再次在底部添加以下行：

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

请注意，与 JavaScript 中的许多事情一样，有许多方法可以选择元素并将对它的引用存储在变量中。 `Document.querySelector()` 是推荐的现代方法。它很方便，因为它允许您使用 CSS 选择器选择元素。上面的调用将匹配出现在文档中的 `querySelector()` 第一个元素。 `<a>` 如果你想匹配多个元素并对其进行操作，你可以使用 `Document.querySelectorAll()` , 它匹配文档中与选择器匹配的每个元素，并将对它们的引用存储在一个名为 a 的类似数组的对象中 `NodeList` 。

有一些较旧的方法可用于获取元素引用，例如：

- `Document.getElementById()`，它选择具有给定属性值的元素 `id`，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()`，它返回一个类数组对象，其中包含页面上给定类型的所有元素，例如 `<p>`s、`<a>`s 等。元素类型作为参数传递给函数，即 `const elementRefArray = document.getElementsByTagName('p');`。

这两种方法在旧版浏览器中比 等现代方法效果更好 `querySelector()`，但不那么方便。看看你还能找到什么！

创建和放置新节点

上面的内容让您对可以做的事情有了一些了解，但让我们更进一步看看我们如何创建新元素。

1. 回到当前示例，让我们从获取对 `<section>` 元素的引用开始——在现有脚本的底部添加以下代码（对其他行也执行相同的操作）：

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

2. 现在让我们使用 `Document.createElement()` 与以前相同的方式创建一个新段落并为其提供一些文本内容：

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

3. 您现在可以使用以下方法在该部分的末尾附加新段落 `Node.appendChild()`：

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

4. 最后，对于这一部分，让我们在链接所在的段落中添加一个文本节点，以很好地完成句子。首先，我们将使用以下方法创建文本节点 `Document.createTextNode()`：

```
const text = document.createTextNode(' — the premier source
for web development knowledge.');
```

5. 现在我们将获取对链接所在段落的引用，并将文本节点附加到它：

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

这就是向 DOM 添加节点所需的大部分内容 — 在构建动态界面时，您将大量使用这些方法（稍后我们将查看一些示例）。

移动和删除元素

有时您可能想要移动节点，或者将它们从 DOM 中完全删除。这是完全可能的。

如果我们想将带有链接的段落移动到该部分的底部，我们可以这样做：

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

这会将段落向下移动到该部分的底部。你可能认为它会复制它的第二个副本，但事实并非如此 - `linkPara` 是对该段落的唯一副本的引用。如果您想复制并添加它，则需要使用 `Node.cloneNode()` 。

删除节点也非常简单，至少当您有对要删除的节点及其父节点的引用时。在我们当前的例子中，我们只是使用 `Node.removeChild()` , 就像这样：

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

当您只想根据对自身的引用删除节点时（这很常见），您可以使用 `Element.remove()` ：

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

旧版浏览器不支持此方法。他们没有办法告诉节点删除自身，因此您必须执行以下操作。

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

尝试将以上几行添加到您的代码中。

操纵样式

可以通过 JavaScript 以多种方式操作 CSS 样式。

首先，您可以使用 获取附加到文档的所有样式表的列表 `Document.stylesheets` ，它返回一个带有对象的类数组对象 `CSSStyleSheet` 。然后您可以根据需要添加/删除样式。但是，我们不打算扩展这些功能，因为它们是一种有点陈旧且难以操纵样式的方法。有更简单的方法。

第一种方法是将内联样式直接添加到要动态设置样式的元素上。这是通过 `HTMLElement.style` 属性完成的，该属性包含文档中每个元素的内联样式信息。您可以设置此对象的属性以直接更新元素样式。

1. 例如，尝试将这些行添加到我们正在进行的示例中：

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

2. 重新加载页面，您会看到样式已应用于该段落。如果您在浏览器的 Page Inspector/DOM inspector 中查看该段落，您会发现这些行确实在向文档添加内联样式：

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

> **注意：** 请注意 CSS 样式的 JavaScript 属性版本是如何以小驼峰形式编写的，而 CSS 版本是用连字符连接的（例如 `backgroundColor` vs `background-color`）。确保不要将这些混淆，否则将无法正常工作。

还有另一种动态操作文档样式的常用方法，我们现在就来看一下。

1. 删除您添加到 JavaScript 的前五行。
2. 在您的 HTML 中添加以下内容 `<head>`：

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

3. 现在我们将转向一个对一般 HTML 操作非常有用的方法——`Element.setAttribute()` 它有两个参数，一个是您要在元素上设置的属性，另一个是您要将其设置为的值。在这种情况下，我们将在我们的段落上设置一个 highlight 的类名：

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

4. 刷新您的页面，您将看不到任何变化 — CSS 仍然应用于该段落，但这次是给它一个由我们的 CSS 规则选择的类，而不是作为内联 CSS 样式。

您选择哪种方法取决于您；两者都有其优点和缺点。第一种方法需要较少的设置并且适用于简单的使用，而第二种方法更纯粹（不混合 CSS 和 JavaScript，没有内联样式，这被视为不好的做法）。当您开始构建更大、更复杂的应用程序时，您可能会更多地开始使用第二种方法，但这完全取决于您。

在这一点上，我们还没有真正做任何有用的事情！使用 JavaScript 来创建静态内容是没有意义的——您还不如将其写入您的 HTML，而不使用 JavaScript。它比 HTML 更复杂，使用 JavaScript 创建内容还存在其他问题（例如搜索引擎无法读取）。

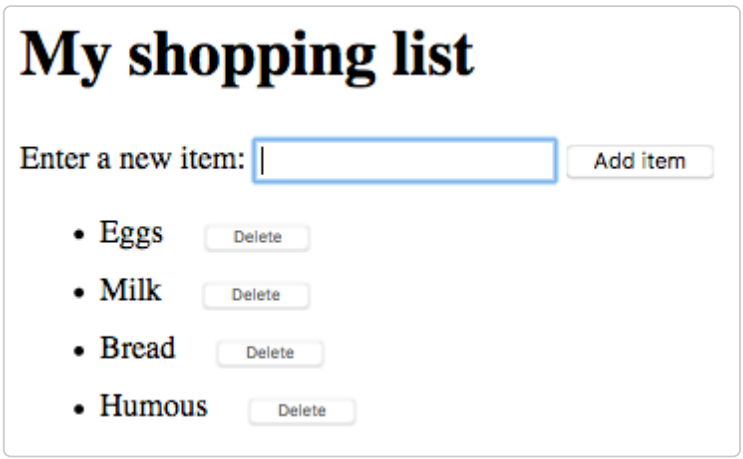
在下一节中，我们将了解 DOM API 的更实际用途。

> **注意：**您可以在 GitHub 上找到我们完成的 dom-example.html 演示版本 （也可以实时查看 ）。

主动学习：动态购物清单

在这个挑战中，我们想制作一个简单的购物清单示例，允许您使用表单输入和按钮将项目动态添加到列表中。当您将项目添加到输入并按下按钮时：

- 该项目应出现在列表中。
- 每个项目都应该有一个按钮，可以按下该按钮从列表中删除该项目。
- 输入应该清空并聚焦，以供您输入另一个项目。



要完成练习，请按照以下步骤操作，并确保列表的行为如上所述。

1. 首先，下载我们的shopping-list.html 起始文件的副本并在某处复制它。您会看到它有一些最小的 CSS，一个带有标签、输入和按钮的

div，以及一个空列表和 `<script>` 元素。您将在脚本中添加所有内容。

2. 创建三个变量来保存对列表（ `<ul>` ）、 `<input>` 和 `<button>` 元素的引用。
3. 创建一个将运行以响应单击按钮的函数。
4. 在函数体内，首先将输入元素的当前值存储在一个变量中。
5. 接下来，通过将其值设置为空字符串来清空输入元素 — `''` 。
6. 创建三个新元素 — 列表项（ `<li>` ）、 `<span>` 和 `<button>` ，并将它们存储在变量中。
7. 将跨度和按钮附加为列表项的子项。
8. 将 span 的文本内容设置为您之前保存的输入元素值，并将按钮的文本内容设置为“删除”。
9. 将列表项附加为列表的子项。
10. 将事件处理程序附加到删除按钮，以便在单击时删除整个列表项（ `<li>...</li>` ）。
11. 最后，使用该 `focus()` 方法使输入元素聚焦，准备进入下一个购物清单项目。

> **注意：** 如果您真的遇到困难，请查看我们完成的购物清单 （也可以实时查看 ）。

概括

我们对文档和 DOM 操作的研究已经结束。至此，您应该了解 Web 浏览器在控制文档和用户 Web 体验的其他方面方面的重要部分。最重要的是，您应该了解文档对象模型是什么，以及如何操作它来创建有用的功能。

也可以看看

您可以使用更多功能来操作文档。查看我们的一些参考资料，看看您能发现什么：

- `Document`
- `Window`

- `Node`
- `HTMLElement`，`HTMLInputElement`，`HTMLImageElement` 等

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

此页面最后修改于 2023 年 2 月 24 日由MDN 贡献者提供。