# 使用 JavaScript 和 DOM 接口遍历 HTML 表格

本文概述了一些强大的，基本的 DOM 1 级别中的方法以及如何在 JavaScript 中使用它们。你将会如何动态地创建、访问、控制以及移除 HTML 元素。这里提到的 DOM 方法，并非是 HTML 专有的；它们在 XML 中同样适用。这里提供的演示在任何现代浏览器中都能正常工作。

> **备注：** 这里介绍的 DOM 方法是文档对象模型 1 级规范核心的一部分。DOM 1 级既包括通用的文档访问和操作的方法（DOM 1 核心），也包括专门针对 HTML 文档的方法（DOM 1 HTML）。

## 动态创建 HTML 表格

### 示例

在本示例中，当按钮被点击时，会向页面中添加一个表格。

#### HTML

HTML Play

```
<input type="button" value="生成表格" onclick="generateTable()" />
```

#### JavaScript

JS Play

```
function generateTable() {
  // 创建一个 <table> 元素和一个 <tbody> 元素
  const tbl = document.createElement("table");
  const tblBody = document.createElement("tbody");
```

```
  // 创建单元格
  for (let i = 0; i < 2; i++) {
    // 创建一行
    const row = document.createElement("tr");

    for (let j = 0; j < 2; j++) {
      // 创建 <td> 元素和文本节点，文本节点是 <td> 的内容，并将 <td>
      // 放在表格最后一行
      const cell = document.createElement("td");
      const cellText = document.createTextNode(`cell in row ${i}, column ${j}`);
      cell.appendChild(cellText);
      row.appendChild(cell);
    }

    // 将该行添加到表格的末尾
    tblBody.appendChild(row);
  }
  // 将 <tbody> 放置在 <table> 内
  tbl.appendChild(tblBody);
  // 将 <table> 放置在 <body> 内
  document.body.appendChild(tbl);
  // 将 tbl 的 border 属性设置为 '2'
  tbl.setAttribute("border", "2");
}
```

## 结果

Play

生成表格

## 解释

注意我们创建元素和文本节点的顺序：

1. 首先我们创建了 `<table>` 元素。

2. 然后，我们创建了 `<table>` 的子元素 `<tbody>`。
3. 然后，我们使用循环语句创建了 `<tbody>` 的子元素， `<tr>`。
4. 对于每一个 `<tr>` 元素，我们使用一个循环语句创建它的子元素 `<td>`。
5. 对于每一个 `<td>` 元素，我们创建单元格内的文本节点。

现在，我们创建了 `<table>`、 `<tbody>`、 `<tr>` 和 `<td>` 等元素，然后创建了文本节点；接下来，我们将每一个对象逆序地接在各自的父节点上：

1. 首先，我们使用这段代码将每一个文本节点接在 `<td>` 元素上。

```
JS
cell.appendChild(cellText);
```

2. 然后，我们将每一个 `<td>` 元素接在它的父元素 `<tr>` 上。

```
JS
row.appendChild(cell);
```

3. 然后，我们将每一个 `<tr>` 元素接在它的父元素 `<tbody>` 上。

```
JS
tblBody.appendChild(row);
```

4. 下一步，我们将 `<tbody>` 元素接在它的父元素 `<table>` 上。

```
JS
tbl.appendChild(tblBody);
```

5. 最后，我们将 `<table>` 元素接在它的父元素 `<body>` 上。

```
JS
document.body.appendChild(tbl);
```

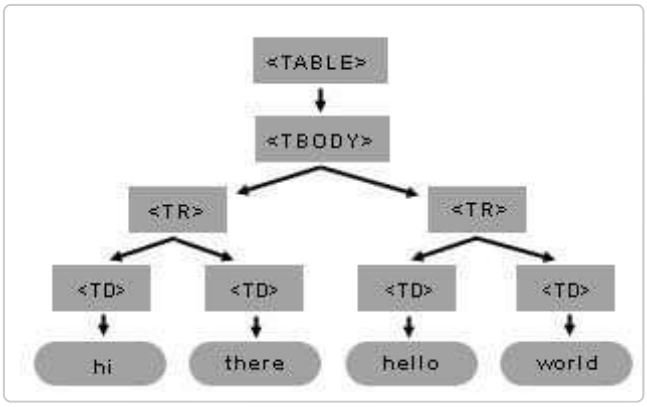
请记住这个机制。你将会在 W3C DOM 编程中经常使用它。首先，你从上到下的创建元素；然后你从下向上的将子元素接在他们的父元素上。

下面是由 JavaScript 代码生成的 HTML 代码：

```
HTML
```

```
<table border="2">
  <tbody>
    <tr>
      <td>cell is row 0 column 0</td>
      <td>cell is row 0 column 1</td>
    </tr>
    <tr>
      <td>cell is row 1 column 0</td>
      <td>cell is row 1 column 1</td>
    </tr>
  </tbody>
</table>
```

下面是由代码生成的 `<table>` 及其子元素的 DOM 对象树：




你只需使用一些 DOM 方法就可以建立这个表及其内部子元素。记住要牢记你打算创建的结构的树形模型；这将使你更容易写出必要的代码。在图 1 的 `<table>` 树中，元素 `<table>` 有一个子节点：元素 `<tbody>` 。 `<tbody>` 有两个子节点。每个 `<tbody>` 的子节点（ `<tr>` ）有两个子节点（ `<td>` ）。最后，每个 `<td>` 有一个子节点：一个文本节点。

# 设置段落的背景颜色

## 示例

在本示例中，当按钮被点击时，段落的背景颜色将会改变。

### HTML

HTML Play

```
<body>
  <input type="button" value="设置段落背景颜色" onclick="setBackground()" />
```

```
  <p>hi</p>
  <p>hello</p>
</body>
```

## JavaScript

JS Play

```
function setBackground() {
  // 获取文档中所有的 p 元素
  const paragraphs = document.getElementsByTagName("p");

  // 从列表中获取第二个元素
  const secondParagraph = paragraphs[1];

  // 设置内联样式
  secondParagraph.style.background = "red";
}
```

## 结果

Play

设置段落背景颜色

hi

hello

## 解释

`getElementsByTagName(tagNameValue)` 是任何 DOM `Element` 或根 `Document` 元素中的一个方法。当被调用时，它返回一个数组，其中包含所有与标签名称相匹配的元素的后代。列表中的第一个元素位于数组中的 `[0]` 位置。

我们进行了以下步骤：

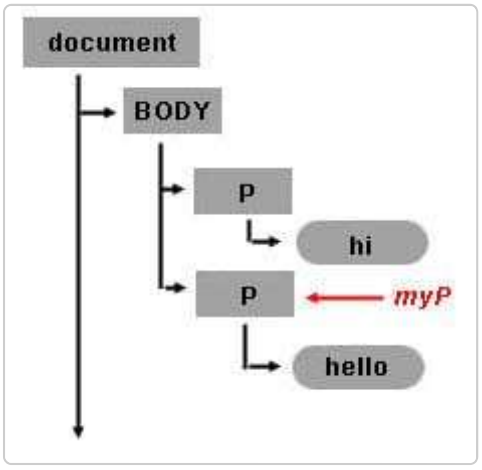
1. 首先，我们获取了文档中所有的 `p` 元素：

   JS

   ```
   const paragraphs = document.getElementsByTagName("p");
   ```

2. 然后，我们从 `p` 元素的列表中获取了第二个段落元素：

```js
const secondParagraph = paragraphs[1];
```




3. 最后，我们使用 paragraph 对象的 style 属性，将背景色设置为红色：

```js
secondParagraph.style.background = "red";
```

## 使用 document.createTextNode("..") 创建文本节点

使用文档对象来调用 `createTextNode` 方法并创建你自己的文本节点。你只需要传递文字内容给这个函数。返回的值就是一个代表那个文本节点信息的对象。

```js
myTextNode = document.createTextNode("world");
```

这表示你已经创建了一个 `TEXT_NODE` （一个文字片断）类型的节点，并且它的内容是 `"world"`，任何你对 `myTextNode` 的引用都指向这个节点对象。如果想将这个文本插入到 HTML 页面中，你还需要将它作为其他节点元素的子元素。

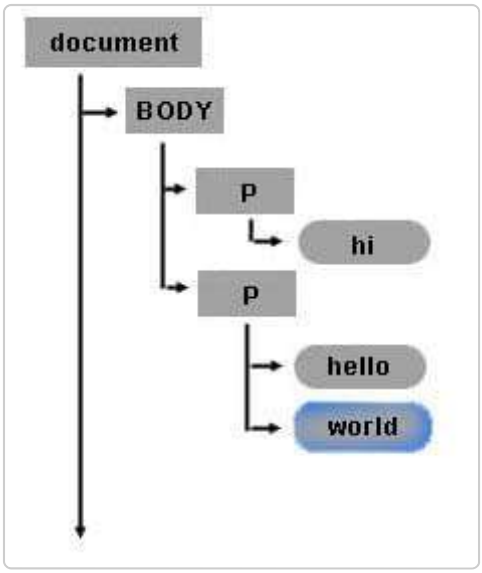
## 使用 appendChild(..) 插入元素

那么，通过调用 `secondParagraph.appendChild(node_element)` 你可以将这个元素设置成为第二个 `<p>` 元素的一个新的子元素。

```js
secondParagraph.appendChild(myTextNode);
```

在测试了这个例子之后，我们注意到，hello 和 world 单词被组合在了一起：helloworld。事实





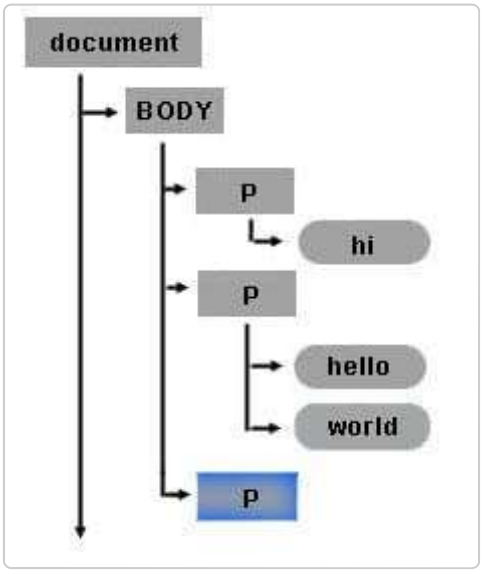

> **备注：** `createTextNode()` 和 `appendChild()` 是在 *hello* 和 *world* 这两个词之间加入空白的简单方法。另一个重要的注意点是，`appendChild` 方法将在最后一个子节点之后追加子节点，就像 *world* 这个词已经被添加到 *hello* 这个词之后一样。因此，如果你想在 *hello* 和 *world* 之间追加一个文本节点，你需要使用 `insertBefore` 而不是 `appendChild` 。

## 使用文档对象和 createElement(..) 方法创建新的元素

你可以使用 `createElement` 来创建新的 HTML 元素或者任何其他你想要的元素。比如，如果你想要创建一个新的 `<p>` 元素作为 `<body>` 的子元素，你可以使用前面例子的 `myBody` 并给它附加一个新的元素节点。使用 `document.createElement("tagname")` 可以方便的创建一个节点。如下：

JS

```
myNewPTagNode = document.createElement("p");
myBody.appendChild(myNewPTagNode);
```




## 使用 removeChild(..) 方法移除节点

每一个节点都可以被移除。下面的代码从第二个 `<p>` 元素 `secondParagraph` 中移除文本节点 `myTextNode` （其中包含单词“world”）。

JS

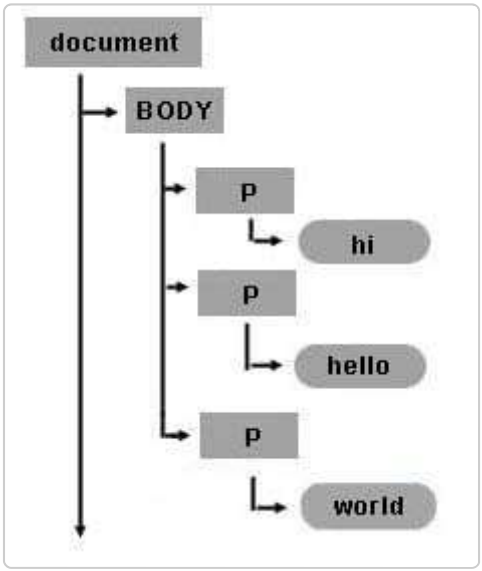
```
secondParagraph.removeChild(myTextNode);
```

文本节点 `myTextNode` （包含单词“world”）仍然存在。下面的代码将 `myTextNode` 附加到最近创建的 `<p>` 元素， `myNewPTagNode` 。

JS

```
myNewPTagNode.appendChild(myTextNode);
```

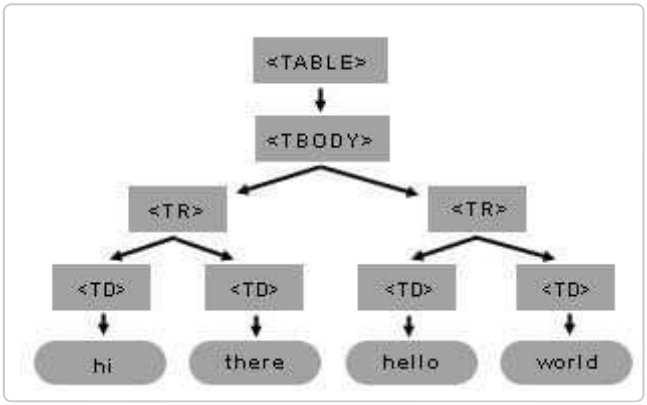
被修改的对象树的最后的状态如下：




## 动态创建一个表格（回到 Sample1.html）

这一段落的剩余部分我们将继续修改我们的 sample1.html。下面的图展示了我们在示例中创建的表格的对象树的结构。

### 回顾 HTML 表格结构




### 创建元素节点并将它们插入到文档树中

sample1.html 中创建表格的基本步骤是：

- 获得 body 对象（文档对象的第一个元素）
- 创建所有元素。
- 最后，根据表格结构（上面图中所示）将每一个孩子节点拼接起来。下面的一段源码是经过修改的 sample1.html

> **备注：** 在 `start` 函数的最后，有一行新的代码。该表的 `border` 属性是用另一个 DOM 方法 `setAttribute()` 设置的。 `setAttribute()` 有两个参数：属性名和属性值。你可以使用 `setAttribute` 方法设置任何元素的任何属性。

HTML

```
<html lang="zh-CN">
  <head>
    <title>示例代码——使用 JavaScript 和 DOM 接口遍历 HTML 表格</title>
    <script>
      function start() {
        // 获取 body 的引用
        const myBody = document.getElementsByTagName("body")[0];

        // 创建 <table> 和 <tbody> 元素
        const myTable = document.createElement("table");
        const myTableBody = document.createElement("tbody");

        // 创建单元格
        for (let j = 0; j < 3; j++) {
          // 创建一个 <tr> 元素
          const myCurrentRow = document.createElement("tr");

          for (let i = 0; i < 4; i++) {
            // 创建一个 <td> 元素
            const myCurrentCell = document.createElement("td");
            // 创建文本节点
            const currentText = document.createTextNode(
              `cell is row ${j}, column ${i}`,
            );
            // 将文本节点附加至 <td> 中
            myCurrentCell.appendChild(currentText);
            // 将单元格 <td> 附加至行 <tr> 中
            myCurrentRow.appendChild(myCurrentCell);
          }
          // 将行 <tr> 附加至 <tbody> 中
          myTableBody.appendChild(myCurrentRow);
        }

        // 将 <tbody> 附加至 <table> 中
        myTable.appendChild(myTableBody);
        // 将 <table> 附加至 <body> 中
        myBody.appendChild(myTable);
        // 将 myTable 的 border 属性设为 2
```

```
        myTable.setAttribute("border", "2");
      }
    </script>
  </head>
  <body onload="start()"></body>
</html>
```

# 使用 CSS 和 DOM 来操作表格

## 从表格中获得一个文字节点

示例介绍了两个新的 DOM 属性。首先，使用 `childNodes` 属性来获得 myCell 的孩子节点列表。`childNodes` 列表包括所有的孩子节点，无论它们的名称或类型是什么。像 `getElementsByTagName()` 一样，它返回了一个节点列表。

不同的是， `getElementsByTagName()` 只返回指定标签名称的元素，且它会返回任何级别的后代，而不仅仅是直接子元素。

一旦你获得了返回的列表，你可以使用 `[x]` 方法来使用指定的元素。这个例子在表格的第二行第二个单元格中的 `myCellText` 中保存了一个文字节点。

然后，为了显示本例中的结果，它创建了一个新的文本节点，其内容是 `myCellText` 的数据，并将其作为 `<body>` 元素的一个子节点进行附加。

> **备注：** 如果你的对象是一个文字节点，你可以使用 data 属性来获取节点的文字内容。

JS

```
myBody = document.getElementsByTagName("body")[0];
myTable = myBody.getElementsByTagName("table")[0];
myTableBody = myTable.getElementsByTagName("tbody")[0];
myRow = myTableBody.getElementsByTagName("tr")[1];
myCell = myRow.getElementsByTagName("td")[1];

// myCell 子节点列表中的第一个元素
myCellText = myCell.childNodes[0];

// currentText 的内容是 myCellText 的数据内容
```

```
currentText = document.createTextNode(myCellText.data);
myBody.appendChild(currentText);
```

## 获得属性的值

在 sample1 的最后我们在 `myTable` 对象上调用了 `setAttribute`。这个调用是用来设置表格的边框属性的。为了获取属性的值，需要使用 `getAttribute` 方法：

JS

```
myTable.getAttribute("border");
```

## 通过改变样式属性来隐藏一列

一旦你在你的 JavaScript 变量中保存了一个对象，你就可以直接为它设置 `style` 属性。下面的代码是修改后的 sample1.html，在这里，第二列的每一个单元格都被隐藏了。而且第一列中的每一个单元格改为使用红色背景。注意， `style` 属性是被直接设置的。

HTML

```
<html lang="zh-CN">
  <body onload="start()"></body>
  <script>
    function start() {
      const myBody = document.getElementsByTagName("body")[0];
      const myTable = document.createElement("table");
      const myTableBody = document.createElement("tbody");

      for (let row = 0; row < 2; row++) {
        const myCurrentRow = document.createElement("tr");
        for (let col = 0; col < 2; col++) {
          const myCurrentCell = document.createElement("td");
          const currentText = document.createTextNode(`cell is: ${row}${col}`);
          myCurrentCell.appendChild(currentText);
          myCurrentRow.appendChild(myCurrentCell);
          // 如果列值为 0，设置单元格背景颜色
          // 如果列值为 1，隐藏单元格
          if (col === 0) {
            myCurrentCell.style.background = "rgb(255, 0, 0)";
          } else {
            myCurrentCell.style.display = "none";
          }
        }
        myTableBody.appendChild(myCurrentRow);
```

```
      }
      myTable.appendChild(myTableBody);
      myBody.appendChild(myTable);
    }
  </script>
</html>
```

## Help improve MDN

Was this page helpful to you?

Yes No

Learn how to contribute.

This page was last modified on 2023年8月3日 by MDN contributors.