的节点从技术上说仍然被同一个文档所拥有，但文档中已经没有它的位置。

要移除节点而不是替换节点，可以使用 removeChild()方法。这个方法接收一个参数，即要移除的节点。被移除的节点会被返回，如下面的例子所示：

```
// 删除第一个子节点
let formerFirstChild = someNode.removeChild(someNode.firstChild);

// 删除最后一个子节点
let formerLastChild = someNode.removeChild(someNode.lastChild);
```

与 replaceChild()方法一样，通过 removeChild()被移除的节点从技术上说仍然被同一个文档所拥有，但文档中已经没有它的位置。

上面介绍的 4 个方法都用于操纵某个节点的子元素，也就是说使用它们之前必须先取得父节点（使用前面介绍的 parentNode 属性）。并非所有节点类型都有子节点，如果在不支持子节点的节点上调用这些方法，则会导致抛出错误。

4. 其他方法

所有节点类型还共享了两个方法。第一个是 cloneNode()，会返回与调用它的节点一模一样的节点。cloneNode()方法接收一个布尔值参数，表示是否深复制。在传入 true 参数时，会进行深复制，即复制节点及其整个子 DOM 树。如果传入 false，则只会复制调用该方法的节点。复制返回的节点属于文档所有，但尚未指定父节点，所以可称为孤儿节点（orphan）。可以通过 appendChild()、insertBefore()或 replaceChild()方法把孤儿节点添加到文档中。以下面的 HTML 片段为例：

```
<ul>
  <li>item 1</li>
  <li>item 2</li>
  <li>item 3</li>
</ul>
```

如果 myList 保存着对这个<ul>元素的引用，则下列代码展示了使用 cloneNode()方法的两种方式：

```
let deepList = myList.cloneNode(true);
alert(deepList.childNodes.length);    // 3（IE9 之前的版本）或 7（其他浏览器）

let shallowList = myList.cloneNode(false);
alert(shallowList.childNodes.length); // 0
```

在这个例子中，deepList 保存着 myList 的副本。这意味着 deepList 有 3 个列表项，每个列表项又各自包含文本。变量 shallowList 则保存着 myList 的浅副本，因此没有子节点。deepList.childNodes.length 的值会因 IE8 及更低版本和其他浏览器对空格的处理方式而不同。IE9 之前的版本不会为空格创建节点。

> **注意**　cloneNode()方法不会复制添加到 DOM 节点的 JavaScript 属性，比如事件处理程序。这个方法只复制 HTML 属性，以及可选地复制子节点。除此之外则一概不会复制。IE 在很长时间内会复制事件处理程序，这是一个 bug，所以推荐在复制前先删除事件处理程序。

本节要介绍的最后一个方法是 normalize()。这个方法唯一的任务就是处理文档子树中的文本节点。由于解析器实现的差异或 DOM 操作等原因，可能会出现并不包含文本的文本节点，或者文本节点

之间互为同胞关系。在节点上调用 normalize()方法会检测这个节点的所有后代，从中搜索上述两种情形。如果发现空文本节点，则将其删除；如果两个同胞节点是相邻的，则将其合并为一个文本节点。这个方法将在本章后面进一步讨论。

14.1.2 Document 类型

Document 类型是 JavaScript 中表示文档节点的类型。在浏览器中，文档对象 document 是 HTMLDocument 的实例（HTMLDocument 继承 Document），表示整个 HTML 页面。document 是 window 对象的属性，因此是一个全局对象。Document 类型的节点有以下特征：

- nodeType 等于 9；
- nodeName 值为"#document"；
- nodeValue 值为 null；
- parentNode 值为 null；
- ownerDocument 值为 null；
- 子节点可以是 DocumentType（最多一个）、Element（最多一个）、ProcessingInstruction 或 Comment 类型。

Document 类型可以表示 HTML 页面或其他 XML 文档，但最常用的还是通过 HTMLDocument 的实例取得 document 对象。document 对象可用于获取关于页面的信息以及操纵其外观和底层结构。

1. 文档子节点

虽然 DOM 规范规定 Document 节点的子节点可以是 DocumentType、Element、ProcessingInstruction 或 Comment，但也提供了两个访问子节点的快捷方式。第一个是 documentElement 属性，始终指向 HTML 页面中的<html>元素。虽然 document.childNodes 中始终有<html>元素，但使用 documentElement 属性可以更快更直接地访问该元素。假如有以下简单的页面：

```
<html>
  <body>

  </body>
</html>
```

浏览器解析完这个页面之后，文档只有一个子节点，即<html>元素。这个元素既可以通过 documentElement 属性获取，也可以通过 childNodes 列表访问，如下所示：

```
let html = document.documentElement;      // 取得对<html>的引用
alert(html === document.childNodes[0]);  // true
alert(html === document.firstChild);      // true
```

这个例子表明 documentElement、firstChild 和 childNodes[0]都指向同一个值，即<html>元素。

作为 HTMLDocument 的实例，document 对象还有一个 body 属性，直接指向<body>元素。因为这个元素是开发者使用最多的元素，所以 JavaScript 代码中经常可以看到 document.body，比如：

```
let body = document.body; // 取得对<body>的引用
```

所有主流浏览器都支持 document.documentElement 和 document.body。

Document 类型另一种可能的子节点是 DocumentType。<!doctype>标签是文档中独立的部分，其信息可以通过 doctype 属性（在浏览器中是 document.doctype）来访问，比如：

```
let doctype = document.doctype; // 取得对<!doctype>的引用
```

另外，严格来讲出现在<html>元素外面的注释也是文档的子节点，它们的类型是 Comment。不过，由于浏览器实现不同，这些注释不一定能被识别，或者表现可能不一致。比如以下 HTML 页面：

```
<!-- 第一条注释 -->
<html>
  <body>

  </body>
</html>
<!-- 第二条注释 -->
```

这个页面看起来有 3 个子节点：注释、<html>元素、注释。逻辑上讲，document.childNodes 应该包含 3 项，对应代码中的每个节点。但实际上，浏览器有可能以不同方式对待<html>元素外部的注释，比如忽略一个或两个注释。

一般来说，appendChild()、removeChild()和 replaceChild()方法不会用在 document 对象上。这是因为文档类型（如果存在）是只读的，而且只能有一个 Element 类型的子节点（即<html>，已经存在了）。[①]

2. 文档信息

document 作为 HTMLDocument 的实例，还有一些标准 Document 对象上所没有的属性。这些属性提供浏览器所加载网页的信息。其中第一个属性是 title，包含<title>元素中的文本，通常显示在浏览器窗口或标签页的标题栏。通过这个属性可以读写页面的标题，修改后的标题也会反映在浏览器标题栏上。不过，修改 title 属性并不会改变<title>元素。下面是一个例子：

```
// 读取文档标题
let originalTitle = document.title;

// 修改文档标题
document.title = "New page title";
```

接下来要介绍的 3 个属性是 URL、domain 和 referrer。其中，URL 包含当前页面的完整 URL（地址栏中的 URL），domain 包含页面的域名，而 referrer 包含链接到当前页面的那个页面的 URL。如果当前页面没有来源，则 referrer 属性包含空字符串。所有这些信息都可以在请求的 HTTP 头部信息中获取，只是在 JavaScript 中通过这几个属性暴露出来而已，如下面的例子所示：

```
// 取得完整的 URL
let url = document.URL;

// 取得域名
let domain = document.domain;

// 取得来源
let referrer = document.referrer;
```

URL 跟域名是相关的。比如，如果 document.URL 是 http://www.wrox.com/WileyCDA/，则 document.domain 就是 www.wrox.com。

在这些属性中，只有 domain 属性是可以设置的。出于安全考虑，给 domain 属性设置的值是有限

① 元素是 HTMLHtmlElement 的实例，HTMLHtmlElement 继承 HTMLElement，HTMLElement 继承 Element，因此 HTML 文档可以包含子节点，但不能多于一个。——译者注

制的。如果URL包含子域名如p2p.wrox.com，则可以将domain设置为"wrox.com"（URL包含“www”时也一样，比如www.wrox.com）。不能给这个属性设置URL中不包含的值，比如：

```
// 页面来自 p2p.wrox.com

document.domain = "wrox.com";       // 成功

document.domain = "nczonline.net"; // 出错!
```

当页面中包含来自某个不同子域的窗格（<frame>）或内嵌窗格（<iframe>）时，设置document.domain是有用的。因为跨源通信存在安全隐患，所以不同子域的页面间无法通过JavaScript通信。此时，在每个页面上把document.domain设置为相同的值，这些页面就可以访问对方的JavaScript对象了。比如，一个加载自www.wrox.com的页面中包含一个内嵌窗格，其中的页面加载自p2p.wrox.com。这两个页面的document.domain包含不同的字符串，内部和外部页面相互之间不能访问对方的JavaScript对象。如果每个页面都把document.domain设置为wrox.com，那这两个页面之间就可以通信了。

浏览器对domain属性还有一个限制，即这个属性一旦放松就不能再收紧。比如，把document.domain设置为"wrox.com"之后，就不能再将其设置回"p2p.wrox.com"，后者会导致错误，比如：

```
// 页面来自 p2p.wrox.com

document.domain = "wrox.com";      // 放松，成功

document.domain = "p2p.wrox.com"; // 收紧，错误!
```

3. 定位元素

使用DOM最常见的情形可能就是获取某个或某组元素的引用，然后对它们执行某些操作。document对象上暴露了一些方法，可以实现这些操作。getElementById()和getElementsByTagName()就是Document类型提供的两个方法。

getElementById()方法接收一个参数，即要获取元素的ID，如果找到了则返回这个元素，如果没找到则返回null。参数ID必须跟元素在页面中的id属性值完全匹配，包括大小写。比如页面中有以下元素：

```
<div id="myDiv">Some text</div>
```

可以使用如下代码取得这个元素：

```
let div = document.getElementById("myDiv"); // 取得对这个<div>元素的引用
```

但参数大小写不匹配会返回null：

```
let div = document.getElementById("mydiv"); // null
```

如果页面中存在多个具有相同ID的元素，则getElementById()返回在文档中出现的第一个元素。

getElementsByTagName()是另一个常用来获取元素引用的方法。这个方法接收一个参数，即要获取元素的标签名，返回包含零个或多个元素的NodeList。在HTML文档中，这个方法返回一个HTMLCollection对象。考虑到二者都是“实时”列表，HTMLCollection与NodeList是很相似的。例如，下面的代码会取得页面中所有的<img>元素并返回包含它们的HTMLCollection：

```
let images = document.getElementsByTagName("img");
```

这里把返回的 HTMLCollection 对象保存在了变量 images 中。与 NodeList 对象一样，也可以使用中括号或 item()方法从 HTMLCollection 取得特定的元素。而取得元素的数量同样可以通过 length 属性得知，如下所示：

```
alert(images.length);       // 图片数量
alert(images[0].src);       // 第一张图片的 src 属性
alert(images.item(0).src);  // 同上
```

HTMLCollection 对象还有一个额外的方法 namedItem()，可通过标签的 name 属性取得某一项的引用。例如，假设页面中包含如下的<img>元素：

```
<img src="myimage.gif" name="myImage">
```

那么也可以像这样从 images 中取得对这个<img>元素的引用：

```
let myImage = images.namedItem("myImage");
```

这样，HTMLCollection 就提供了除索引之外的另一种获取列表项的方式，从而为取得元素提供了便利。对于 name 属性的元素，还可以直接使用中括号来获取，如下面的例子所示：

```
let myImage = images["myImage"];
```

对 HTMLCollection 对象而言，中括号既可以接收数值索引，也可以接收字符串索引。而在后台，数值索引会调用 item()，字符串索引会调用 namedItem()。

要取得文档中的所有元素，可以给 getElementsByTagName()传入*。在 JavaScript 和 CSS 中，* 一般被认为是匹配一切的字符。来看下面的例子：

```
let allElements = document.getElementsByTagName("*");
```

这行代码可以返回包含页面中所有元素的 HTMLCollection 对象，顺序就是它们在页面中出现的顺序。因此第一项是<html>元素，第二项是<head>元素，以此类推。

> **注意**　对于 document.getElementsByTagName()方法，虽然规范要求区分标签的大小写，但为了最大限度兼容原有 HTML 页面，实际上是不区分大小写的。如果是在 XML 页面（如 XHTML）中使用，那么 document.getElementsByTagName()就是区分大小写的。

HTMLDocument 类型上定义的获取元素的第三个方法是 getElementsByName()。顾名思义，这个方法会返回具有给定 name 属性的所有元素。getElementsByName()方法最常用于单选按钮，因为同一字段的单选按钮必须具有相同的 name 属性才能确保把正确的值发送给服务器，比如下面的例子：

```
<fieldset>
  <legend>Which color do you prefer?</legend>
  <ul>
    <li>
      <input type="radio" value="red" name="color" id="colorRed">
      <label for="colorRed">Red</label>
    </li>
    <li>
      <input type="radio" value="green" name="color" id="colorGreen">
      <label for="colorGreen">Green</label>
    </li>
    <li>
      <input type="radio" value="blue" name="color" id="colorBlue">
```

```
        <label for="colorBlue">Blue</label>
      </li>
    </ul>
  </fieldset>
```

这里所有的单选按钮都有名为"color"的 name 属性，但它们的 ID 都不一样。这是因为 ID 是为了匹配对应的<label>元素，而 name 相同是为了保证只将三个中的一个值发送给服务器。然后就可以像下面这样取得所有单选按钮：

```
let radios = document.getElementsByName("color");
```

与 getElementsByTagName()一样，getElementsByName()方法也返回 HTMLCollection。不过在这种情况下，namedItem()方法只会取得第一项（因为所有项的 name 属性都一样）。

4. 特殊集合

document 对象上还暴露了几个特殊集合，这些集合也都是 HTMLCollection 的实例。这些集合是访问文档中公共部分的快捷方式，列举如下。

- document.anchors 包含文档中所有带 name 属性的<a>元素。
- document.applets 包含文档中所有<applet>元素（因为<applet>元素已经不建议使用，所以这个集合已经废弃）。
- document.forms 包含文档中所有<form>元素（与 document.getElementsByTagName ("form") 返回的结果相同）。
- document.images 包含文档中所有<img>元素（与 document.getElementsByTagName ("img") 返回的结果相同）。
- document.links 包含文档中所有带 href 属性的<a>元素。

这些特殊集合始终存在于 HTMLDocument 对象上，而且与所有 HTMLCollection 对象一样，其内容也会实时更新以符合当前文档的内容。

5. DOM 兼容性检测

由于 DOM 有多个 Level 和多个部分，因此确定浏览器实现了 DOM 的哪些部分是很必要的。document.implementation 属性是一个对象，其中提供了与浏览器 DOM 实现相关的信息和能力。DOM Level 1 在 document.implementation 上只定义了一个方法，即 hasFeature()。这个方法接收两个参数：特性名称和 DOM 版本。如果浏览器支持指定的特性和版本，则 hasFeature()方法返回 true，如下面的例子所示：

```
let hasXmlDom = document.implementation.hasFeature("XML", "1.0");
```

可以使用 hasFeature()方法测试的特性及版本如下表所列。

特　性	支持的版本	说　明
Core	1.0、2.0、3.0	定义树形文档结构的基本 DOM
XML	1.0、2.0、3.0	Core 的 XML 扩展，增加了对 CDATA 区块、处理指令和实体的支持
HTML	1.0、2.0	XML 的 HTML 扩展，增加了 HTML 特定的元素和实体
Views	2.0	文档基于某些样式的实现格式
StyleSheets	2.0	文档的相关样式表
CSS	2.0	Cascading Style Sheets Level 1
CSS2	2.0	Cascading Style Sheets Level 2

（续）

特　性	支持的版本	说　明
Events	2.0、3.0	通用 DOM 事件
UIEvents	2.0、3.0	用户界面事件
TextEvents	3.0	文本输入设备触发的事件
MouseEvents	2.0、3.0	鼠标导致的事件（单击、悬停等）
MutationEvents	2.0、3.0	DOM 树变化时触发的事件
MutationNameEvents	3.0	DOM 元素或元素属性被重命名时触发的事件
HTMLEvents	2.0	HTML 4.01 事件
Range	2.0	在 DOM 树中操作一定范围的对象和方法
Traversal	2.0	遍历 DOM 树的方法
LS	3.0	文件与 DOM 树之间的同步加载与保存
LS-Async	3.0	文件与 DOM 树之间的异步加载与保存
Validation	3.0	修改 DOM 树并保证其继续有效的方法
XPath	3.0	访问 XML 文档不同部分的语言

由于实现不一致，因此 hasFeature()的返回值并不可靠。目前这个方法已经被废弃，不再建议使用。为了向后兼容，目前主流浏览器仍然支持这个方法，但无论检测什么都一律返回 true。

6. 文档写入

document 对象有一个古老的能力，即向网页输出流中写入内容。这个能力对应 4 个方法：write()、writeln()、open()和 close()。其中，write()和 writeln()方法都接收一个字符串参数，可以将这个字符串写入网页中。write()简单地写入文本，而 writeln()还会在字符串末尾追加一个换行符（\n）。这两个方法可以用来在页面加载期间向页面中动态添加内容，如下所示：

```
<html>
<head>
  <title>document.write() Example</title>
</head>
<body>
  <p>The current date and time is:
  <script type="text/javascript">
    document.write("<strong>" + (new Date()).toString() + "</strong>");
  </script>
</p>
</body>
</html>
```

这个例子会在页面加载过程中输出当前日期和时间。日期放在了元素中，如同它们之前就包含在 HTML 页面中一样。这意味着会创建一个 DOM 元素，以后也可以访问。通过 write()和 writeln()输出的任何 HTML 都会以这种方式来处理。

write()和 writeln()方法经常用于动态包含外部资源，如 JavaScript 文件。在包含 JavaScript 文件时，记住不能像下面的例子中这样直接包含字符串"</script>"，因为这个字符串会被解释为脚本块的结尾，导致后面的代码不能执行：

```
<html>
<head>
```

```
  <title>document.write() Example</title>
</head>
<body>
  <script type="text/javascript">
    document.write("<script type=\"text/javascript\" src=\"file.js\">" +
      "</script>");
  </script>
</body>
</html>
```

虽然这样写看起来没错，但输出之后的"</script>"会匹配最外层的<script>标签，导致页面中显示出");。为避免出现这个问题，需要对前面的例子稍加修改：

```
<html>
<head>
  <title>document.write() Example</title>
</head>
<body>
  <script type="text/javascript">
    document.write("<script type=\"text/javascript\" src=\"file.js\">" +
      "<\/script>");
  </script>
</body>
</html>
```

这里的字符串"<\/script>"不会再匹配最外层的<script>标签，因此不会在页面中输出额外内容。

前面的例子展示了在页面渲染期间通过 document.write()向文档中输出内容。如果是在页面加载完之后再调用 document.write()，则输出的内容会重写整个页面，如下面的例子所示：

```
<html>
<head>
  <title>document.write() Example</title>
</head>
<body>
  <p>This is some content that you won't get to see because it will be
  overwritten.</p>
  <script type="text/javascript">
    window.onload = function(){
      document.write("Hello world!");
    };
  </script>
</body>
</html>
```

这个例子使用了 window.onload 事件处理程序，将调用 document.write()的函数推迟到页面加载完毕后执行。执行之后，字符串"Hello world!"会重写整个页面内容。

open()和 close()方法分别用于打开和关闭网页输出流。在调用 write()和 writeln()时，这两个方法都不是必需的。

> **注意** 严格的 XHTML 文档不支持文档写入。对于内容类型为 application/xml+xhtml 的页面，这些方法不起作用。

14.1.3 Element 类型

除了 Document 类型，Element 类型就是 Web 开发中最常用的类型了。Element 表示 XML 或 HTML 元素，对外暴露出访问元素标签名、子节点和属性的能力。Element 类型的节点具有以下特征：

- nodeType 等于 1；
- nodeName 值为元素的标签名；
- nodeValue 值为 null；
- parentNode 值为 Document 或 Element 对象；
- 子节点可以是 Element、Text、Comment、ProcessingInstruction、CDATASection、EntityReference 类型。

可以通过 nodeName 或 tagName 属性来获取元素的标签名。这两个属性返回同样的值（添加后一个属性明显是为了不让人误会）。比如有下面的元素：

```
<div id="myDiv"></div>
```

可以像这样取得这个元素的标签名：

```
let div = document.getElementById("myDiv");
alert(div.tagName); // "DIV"
alert(div.tagName == div.nodeName); // true
```

例子中的元素标签名为 div，ID 为"myDiv"。注意，div.tagName 实际上返回的是"DIV"而不是"div"。在 HTML 中，元素标签名始终以全大写表示；在 XML（包括 XHTML）中，标签名始终与源代码中的大小写一致。如果不确定脚本是在 HTML 文档还是 XML 文档中运行，最好将标签名转换为小写形式，以便于比较：

```
if (element.tagName == "div"){ // 不要这样做，可能出错！
  // do something here
}

if (element.tagName.toLowerCase() == "div"){ // 推荐，适用于所有文档
  // 做点什么
}
```

这个例子演示了比较 tagName 属性的情形。第一个是容易出错的写法，因为 HTML 文档中 tagName 返回大写形式的标签名。第二个先把标签名转换为全部小写后再比较，这是推荐的做法，因为这对 HTML 和 XML 都适用。

1. HTML 元素

所有 HTML 元素都通过 HTMLElement 类型表示，包括其直接实例和间接实例。另外，HTMLElement 直接继承 Element 并增加了一些属性。每个属性都对应下列属性之一，它们是所有 HTML 元素上都有的标准属性：

- id，元素在文档中的唯一标识符；
- title，包含元素的额外信息，通常以提示条形式展示；
- lang，元素内容的语言代码（很少用）；
- dir，语言的书写方向（"ltr"表示从左到右，"rtl"表示从右到左，同样很少用）；
- className，相当于 class 属性，用于指定元素的 CSS 类（因为 class 是 ECMAScript 关键字，所以不能直接用这个名字）。

所有这些都可以用来获取对应的属性值，也可以用来修改相应的值。比如有下面的 HTML 元素：

```
<div id="myDiv" class="bd" title="Body text" lang="en" dir="ltr"></div>
```

这个元素中的所有属性都可以使用下列 JavaScript 代码读取：

```
let div = document.getElementById("myDiv");
alert(div.id);          // "myDiv"
alert(div.className);   // "bd"
alert(div.title);       // "Body text"
alert(div.lang);        // "en"
alert(div.dir);         // "ltr"
```

而且，可以使用下列代码修改元素的属性：

```
div.id = "someOtherId";
div.className = "ft";
div.title = "Some other text";
div.lang = "fr";
div.dir ="rtl";
```

并非所有这些属性的修改都会对页面产生影响。比如，把 `id` 或 `lang` 改成其他值对用户是不可见的（假设没有基于这两个属性应用 CSS 样式），而修改 `title` 属性则只会在鼠标移到这个元素上时才会反映出来。修改 `dir` 会导致页面文本立即向左或向右对齐。修改 `className` 会立即反映应用到新类名的 CSS 样式（如果定义了不同的样式）。

如前所述，所有 HTML 元素都是 `HTMLElement` 或其子类型的实例。下表列出了所有 HTML 元素及其对应的类型（斜体表示已经废弃的元素）。

元　素	类　型	元　素	类　型
A	HTMLAnchorElement	COL	HTMLTableColElement
ABBR	HTMLElement	COLGROUP	HTMLTableColElement
ACRONYM	HTMLElement	DD	HTMLElement
ADDRESS	HTMLElement	DEL	HTMLModElement
APPLET	*HTMLAppletElement*	DFN	HTMLElement
AREA	HTMLAreaElement	*DIR*	*HTMLDirectoryElement*
B	HTMLElement	DIV	HTMLDivElement
BASE	HTMLBaseElement	DL	HTMLDListElement
BASEFONT	*HTMLBaseFontElement*	DT	HTMLElement
BDO	HTMLElement	EM	HTMLElement
BIG	HTMLElement	FIELDSET	HTMLFieldSetElement
BLOCKQUOTE	HTMLQuoteElement	*FONT*	*HTMLFontElement*
BODY	HTMLBodyElement	FORM	HTMLFormElement
BR	HTMLBRElement	FRAME	HTMLFrameElement
BUTTON	HTMLButtonElement	FRAMESET	HTMLFrameSetElement
CAPTION	HTMLTableCaptionElement	H1	HTMLHeadingElement
CENTER	*HTMLElement*	H2	HTMLHeadingElement
CITE	HTMLElement	H3	HTMLHeadingElement
CODE	HTMLElement	H4	HTMLHeadingElement