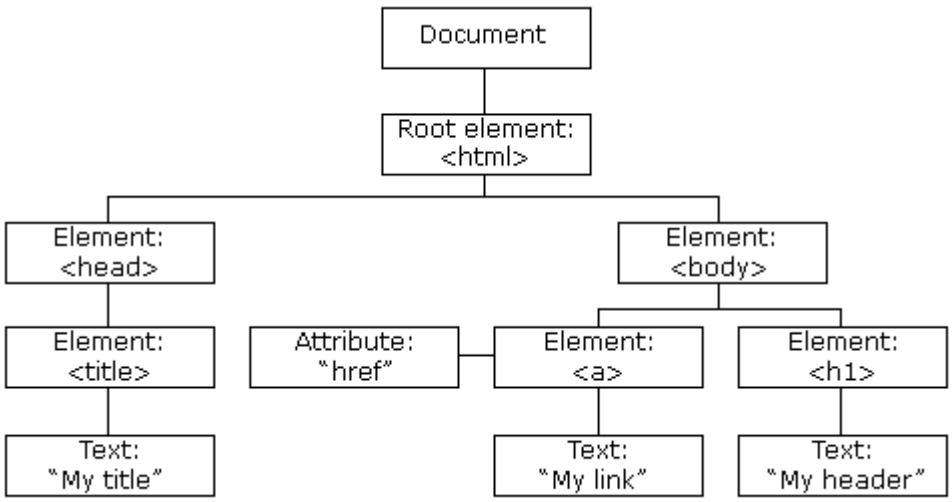
DOM

通过 HTML DOM，可访问 JavaScript HTML 文档的所有元素。

HTML DOM (文档对象模型)

当网页被加载时，浏览器会创建页面的文档对象模型（Document Object Model）。

HTML DOM 模型被构造为对象的树：



通过可编程的对象模型，JavaScript 获得了足够的能力来创建动态的 HTML。

- JavaScript 能够改变页面中的所有 HTML 元素
- JavaScript 能够改变页面中的所有 HTML 属性
- JavaScript 能够改变页面中的所有 CSS 样式
- JavaScript 能够对页面中的所有事件做出反应

查找 HTML 元素

通常，通过 JavaScript，您需要操作 HTML 元素。

为了做到这件事情，您必须首先找到该元素。有三种方法来做这件事：

- 通过 id 找到 HTML 元素
- 通过标签名找到 HTML 元素
- 通过类名找到 HTML 元素

通过 id 查找 HTML 元素

在 DOM 中查找 HTML 元素的最简单的方法，是通过使用元素的 id。

本例查找 id="intro" 元素：

```
<p id="intro">你好世界!</p>
var x=document.getElementById("intro");
```

如果找到该元素，则该方法将以对象（在 x 中）的形式返回该元素。

如果未找到该元素，则 x 将包含 null。

通过标签名查找 HTML 元素

本例查找 id="main" 的元素，然后查找 id="main" 元素中的所有 < p > 元素：

```
<p>你好世界!</p>
<div id="main">
<p> DOM 是非常有用的。</p>
<p>该实例展示了  <b>getElementsByTagName</b> 方法</p>
</div>
<script>
var x=document.getElementById("main");
var y=x.getElementsByTagName("p");
document.write('id="main"元素中的第一个段落为: ' + y[0].innerHTML)

//运行结果

你好世界!

DOM 是非常有用的。

该实例展示了 getElementsByTagName 方法
```

```
id="main"元素中的第一个段落为： DOM 是非常有用的。
```

通过类名找到 HTML 元素

本例通过 **getElementsByClassName** 函数来查找 class="intro" 的元素：

```
<p class="intro">你好世界!</p>
<p>该实例展示了 <b>getElementsByClassName</b> 方法!</p>
<script>
x=document.getElementsByClassName("intro");
document.write("<p>文本来自 class 为 intro 段落: " + x[0].innerHTML + "</p>");
</script>
<p><b>注意: </b>Internet Explorer 8 及更早 IE 版本不支持 getElementsByClassName() 方法。</p>

//运行结果

你好世界!

该实例展示了 getElementsByClassName 方法!

文本来自 class 为 intro 段落: 你好世界!

注意: Internet Explorer 8 及更早 IE 版本不支持 getElementsByClassName() 方法。
```

HTML DOM - 改变 HTML

HTML DOM 允许 JavaScript 改变 HTML 元素的内容。

改变 HTML 输出流

JavaScript 能够创建动态的 HTML 内容：

今天的日期是： **Mon Apr 10 2023 20:11:30 GMT+0800 (中国标准时间)**

在 JavaScript 中，document.write() 可用于直接向 HTML 输出流写内容。

```
<!DOCTYPE html>
<html>
<body>
```

```
<script>
document.write(Date());
</script>

</body>
</html>
```

注意：绝对不要在文档(DOM)加载完成之后使用 document.write()。这会覆盖该文档。

改变 HTML 内容

innerHTML

修改 HTML 内容的最简单的方法是使用 innerHTML 属性。

如需改变 HTML 元素的内容，请使用这个语法：

```
document.getElementById(id).innerHTML=新的 HTML
```

```
<!DOCTYPE html>
<html>
<body>

<h1 id="header">Old Header</h1>

<script>
var element=document.getElementById("header");
element.innerHTML="新标题";
</script>

</body>
</html>
```

实例讲解：

- 上面的 HTML 文档含有 id="header" 的 < h1 > 元素
- 我们使用 HTML DOM 来获得 id="header" 的元素
- JavaScript 更改此元素的内容 (innerHTML)

innerText属性

innerText，即使后面是一段HTML代码，也只是将其当做普通的字符串来看待

```
// 设置div的内容
// 第一步:获取div对象
var divElt = document.getElementById("div1");
// 第二步:使用innerHTML属性来设置元素内部的内容
// divElt.innerHTML = "fjdkslajfkdlsajkfldsjaklfds";
// divElt.innerHTML = "<font color='red'>用户名不能为空! </font>";
divElt.innerText = "<font color='red'>用户名不能为空! </font>";
```

改变 HTML 属性

如需改变 HTML 元素的属性，请使用这个语法：

```
document.getElementById(id).attribute=新属性值
```

```
<!DOCTYPE html>
<html>
<body>

<img id="image" src="smiley.gif">

<script>
document.getElementById("image").src="landscape.jpg";
</script>

</body>
</html>
```

实例讲解：

- 上面的 HTML 文档含有 id="image" 的 < img > 元素
- 我们使用 HTML DOM 来获得 id="image" 的元素
- JavaScript 更改此元素的属性（把 "smiley.gif" 改为 "landscape.jpg"）

HTML DOM - 改变CSS

HTML DOM 允许 JavaScript 改变 HTML 元素的样式。

改变 HTML 样式

如需改变 HTML 元素的样式，请使用这个语法：

```
document.getElementById( id ).style.property =新样式
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>

<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>
<script>
document.getElementById("p2").style.color="blue";
document.getElementById("p2").style.fontFamily="Arial";
document.getElementById("p2").style.fontSize="larger";
</script>
<p>以上段落通过脚本修改。</p>

</body>
</html>
```

使用事件

HTML DOM 允许我们通过触发事件来执行代码。

比如以下事件：

- 元素被点击。
- 页面加载完成。
- 输入框被修改。
- ……

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id1">我的标题 1</h1>
<button type="button"
onclick="document.getElementById('id1').style.color='red'">点我!</button>

<p id="p1">这是一个文本。 这是一个文本。 这是一个文本。 这是一个文本。 这是一个文本。 这是一个文本。 这是一个文本。</p>
```

```
<input type="button" value="隐藏文本"
onclick="document.getElementById('p1').style.visibility='hidden'" />
<input type="button" value="显示文本"
onclick="document.getElementById('p1').style.visibility='visible'" />


</body>
</html>
```

HTML DOM 事件

HTML DOM 使 JavaScript 有能力对 HTML 事件做出反应。

对事件做出反应

我们可以在事件发生时执行 JavaScript，比如当用户在 HTML 元素上点击时。

如需在用户点击某个元素时执行代码，请向一个 HTML 事件属性添加 JavaScript 代码：

```
onclick=JavaScript
```

HTML 事件的例子：

- 当用户点击鼠标时
- 当网页已加载时
- 当图像已加载时
- 当鼠标移动到元素上时
- 当输入字段被改变时
- 当提交 HTML 表单时
- 当用户触发按键时

在本例中，当用户在 < h1 > 元素上点击时，会改变其内容：

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="this.innerHTML='Ooops!'">点击文本!</h1>
</body>
</html>
```

本例从事件处理器调用一个函数：

```
<!DOCTYPE html>
<html>
<head>
<script>
function changetext(id)
{
    id.innerHTML="Ooops!";
}
</script>
</head>
<body>
<h1 onclick="changetext(this)">点击文本!</h1>
</body>
</html>
```

HTML 事件属性

如需向 HTML 元素分配 事件，您可以使用事件属性。

```
<button onclick="displayDate()">点这里</button>
```

使用 HTML DOM 来分配事件

HTML DOM 允许您使用 JavaScript 来向 HTML 元素分配事件：

```
<script>
document.getElementById("myBtn").onclick=function(){displayDate()};
</script>
```

onload 和 onunload 事件

onload 和 onunload 事件会在用户进入或离开页面时被触发。

onload 事件可用于检测访问者的浏览器类型和浏览器版本，并基于这些信息来加载网页的正确版本。

onload 和 onunload 事件可用于处理 cookie。

```
<body onload="checkCookies()">

<script>
function checkCookies(){
        if (navigator.cookieEnabled==true){
                alert("Cookies 可用")
        }
        else{
                alert("Cookies 不可用")
        }
}
</script>
<p>弹窗-提示浏览器 cookie 是否可用。</p>
```

onchange 事件

onchange 事件常结合对输入字段的验证来使用。

下面是一个如何使用 onchange 的例子。当用户改变输入字段的内容时，会调用 upperCase() 函数。

```
<input type="text" id="fname" onchange="upperCase()">
```

onmouseover 和 onmouseout 事件

onmouseover 和 onmouseout 事件可用于在用户的鼠标移至 HTML 元素上方或移出元素时触发函数。

```
<div onmouseover="mOver(this)" onmouseout="mOut(this)" style="background-
color:#D94A38;width:120px;height:20px;padding:40px;">Mouse Over Me</div>
<script>
function mOver(obj){
        obj.innerHTML="Thank You"
}
function mOut(obj){
        obj.innerHTML="Mouse Over Me"
}
</script>
```

HTML DOM EventListener

addEventListener() 方法

在用户点击按钮时触发监听事件：

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

1. addEventListener() 方法用于向指定元素添加事件句柄。
2. addEventListener() 方法添加的事件句柄不会覆盖已存在的事件句柄。
3. 你可以向一个元素添加多个事件句柄。
4. 你可以向同个元素添加多个同类型的事件句柄，如：两个 "click" 事件。
5. 你可以向任何 DOM 对象添加事件监听，不仅仅是 HTML 元素。如： window 对象。
6. addEventListener() 方法可以更简单的控制事件（冒泡与捕获）。
7. 当你使用 addEventListener() 方法时, JavaScript 从 HTML 标记中分离开来，可读性更强， 在没有控制HTML标记时也可以添加事件监听。
8. 你可以使用 removeEventListener() 方法来移除事件的监听。

语法

```
element.addEventListener(event, function, useCapture);
```

1. 第一个参数是事件的类型 (如 "click" 或 "mousedown").
2. 第二个参数是事件触发后调用的函数。
3. 第三个参数是个布尔值用于描述事件是冒泡还是捕获。该参数是可选的。
4. 注意:不要使用 "on" 前缀。 例如，使用 "click" ,而不是使用 "onclick"。

向原元素添加事件句柄

当用户点击元素时弹出 "Hello World!" :

```
element.addEventListener("click", function(){ alert("Hello World!"); });
```

```
<p>该实例使用 addEventListener() 方法在按钮中添加点击事件。 </p>
<button id="myBtn">点我</button>
<script>
document.getElementById("myBtn").addEventListener("click", function(){
    alert("Hello World!");
});
</script>
```

向同一个元素中添加多个事件句柄

addEventListener() 方法允许向同一个元素添加多个事件，且不会覆盖已存在的事件：

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

向 Window 对象添加事件句柄

addEventListener() 方法允许你在 HTML DOM 对象添加事件监听， HTML DOM 对象如： HTML 元素, HTML 文档, window 对象。或者其他支持的事件对象如: xmlHttpRequest 对象。

```
<p>实例在 window 对象中使用 addEventListener() 方法。</p>
<p>尝试重置浏览器的窗口触发 "resize" 事件句柄。</p>
<p id="demo"></p>
<script>
window.addEventListener("resize", function(){
    document.getElementById("demo").innerHTML = Math.random();
});
</script>
```

传递参数

当传递参数值时，使用"匿名函数"调用带参数的函数：

```
<p>实例演示了在使用 addEventListener() 方法时如何传递参数。</p>
<p>点击按钮执行计算。</p>
```

```
<button id="myBtn">点我</button>
<p id="demo"></p>
<script>
var p1 = 5;
var p2 = 7;
document.getElementById("myBtn").addEventListener("click", function() {
    myFunction(p1, p2);
});
function myFunction(a, b) {
    var result = a * b;
    document.getElementById("demo").innerHTML = result;
}
</script>
```

事件冒泡或事件捕获？

事件传递有两种方式：冒泡与捕获。

事件传递定义了元素事件触发的顺序。 如果你将 < p > 元素插入到 < div > 元素中，用户点击 < p > 元素, 哪个元素的 "click" 事件先被触发呢？

在 *冒泡 *中，内部元素的事件会先被触发，然后再触发外部元素，即： < p > 元素的点击事件先触发，然后会触发 < div > 元素的点击事件。

在 *捕获 *中，外部元素的事件会先被触发，然后才会触发内部元素的事件，即： < div > 元素的点击事件先触发 ，然后再触发 < p > 元素的点击事件。

addEventListener() 方法可以指定 "useCapture" 参数来设置传递类型：

```
addEventListener(event, function, useCapture);
```

默认值为 false, 即冒泡传递，当值为 true 时, 事件使用捕获传递。

```
<p>实例演示了在添加不同事件监听时，冒泡与捕获的不同。</p>
<div id="myDiv">
        <p id="myP">点击段落，我是冒泡。</p>
</div><br>
<div id="myDiv2">
        <p id="myP2">点击段落，我是捕获。 </p>
</div>
<script>
document.getElementById("myP").addEventListener("click", function() {
    alert("你点击了 P 元素!");
}, false);
document.getElementById("myDiv").addEventListener("click", function() {
```

```
    alert(" 你点击了 DIV 元素 !");
}, false);
document.getElementById("myP2").addEventListener("click", function() {
    alert("你点击了 P2 元素!");
}, true);
document.getElementById("myDiv2").addEventListener("click", function() {
    alert("你点击了 DIV2 元素 !");
}, true);
</script>
```

removeEventListener() 方法

removeEventListener() 方法移除由 addEventListener() 方法添加的事件句柄:

```
element.removeEventListener("mousemove", myFunction);
```

HTML DOM 元素 (节点)

本章节介绍如何向文档中添加和移除元素(节点)。

创建新的 HTML 元素 (节点) - appendChild()

要创建新的 HTML 元素 (节点)需要先创建一个元素，然后在已存在的元素中添加它。

```
<div id="div1">
<p id="p1">这是一个段落。</p>
<p id="p2">这是另外一个段落。</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("这是一个新的段落。");
para.appendChild(node);

var element = document.getElementById("div1");
element.appendChild(para);
</script>
```

这是一个段落。

这是另外一个段落。

```
这是一个新的段落。
```

实例解析

以下代码是用于创建 < p > 元素:

```
var para = document.createElement("p");
```

为 < p > 元素创建一个新的文本节点：

```
var node = document.createTextNode("这是一个新的段落。");
```

将文本节点添加到 < p > 元素中：

```
para.appendChild(node);
```

最后，在一个已存在的元素中添加 p 元素。

查找已存在的元素：

```
var element = document.getElementById("div1");
```

添加到已存在的元素中:

```
element.appendChild(para);
```

创建新的 HTML 元素 (节点) - insertBefore()

以上的实例我们使用了 **appendChild()** 方法，它用于添加新元素到尾部。

如果我们需要将新元素添加到开始位置，可以使用 **insertBefore()** 方法:

```
<div id="div1">
<p id="p1">这是一个段落。</p>
```

```
<p id="p2">这是另外一个段落。</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("这是一个新的段落。");
para.appendChild(node);

var element = document.getElementById("div1");
var child = document.getElementById("p1");
element.insertBefore(para, child);
</script>
```

这是一个新的段落。

这是一个段落。

这是另外一个段落。

移除已存在的元素

要移除一个元素，你需要知道该元素的父元素。

```
<div id="div1">
<p id="p1">这是一个段落。</p>
<p id="p2">这是另外一个段落。</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.removeChild(child);
</script>
```

如果能够在不引用父元素的情况下删除某个元素，就太好了。不过很遗憾。DOM 需要清楚您需要删除的元素，以及它的父元素。

以下代码是已知要查找的子元素，然后查找其父元素，再删除这个子元素（删除节点必须知道父节点）：

```
var child = document.getElementById("p1");
child.parentNode.removeChild(child);
```

替换 HTML 元素 - replaceChild()

我们可以使用 replaceChild() 方法来替换 HTML DOM 中的元素。

```
<div id="div1">
<p id="p1">这是一个段落。</p>
<p id="p2">这是另外一个段落。</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("这是一个新的段落。");
para.appendChild(node);

var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.replaceChild(para, child);
</script>
```

HTML DOM 集合(Collection)

本章节介绍 DOM 集合的使用。

HTMLCollection 对象

getElementsByTagName() 方法返回 HTMLCollection 对象。

HTMLCollection 对象类似包含 HTML 元素的一个数组。

以下代码获取文档所有的 < p > 元素：

```
var x = document.getElementsByTagName("p");

集合中的元素可以通过索引(以 0 为起始位置)来访问。

访问第二个 <p> 元素可以是以下代码：
y = x[1];
```

HTMLCollection 对象 length 属性

HTMLCollection 对象的 length 属性定义了集合中元素的数量。

```
var myCollection = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML = myCollection.length;
```

集合 length 属性常用于遍历集合中的元素。

```
var myCollection = document.getElementsByTagName("p");
var i;
for (i = 0; i < myCollection.length; i++) {
    myCollection[i].style.backgroundColor = "red";
}
```

注意

HTMLCollection 不是一个数组！HTMLCollection 看起来可能是一个数组，但其实不是。你可以像数组一样，使用索引来获取元素。HTMLCollection 无法使用数组的方法：valueOf(), pop(), push(), 或 join() 。

HTML DOM 节点列表（NodeList）

NodeList 对象是一个从文档中获取的节点列表 (集合) 。

NodeList 对象类似 HTMLCollection 对象。

一些旧版本浏览器中的方法（如： **getElementsByClassName()** ）返回的是 NodeList 对象，而不是 HTMLCollection 对象。

所有浏览器的 **childNodes** 属性返回的是 NodeList 对象。

大部分浏览器的 **querySelectorAll()** 返回 NodeList 对象。

以下代码选取了文档中所有的 < p > 节点：

```
var myNodeList = document.querySelectorAll("p");

NodeList 中的元素可以通过索引(以 0 为起始位置)来访问。

访问第二个 <p> 元素可以是以下代码：
y = myNodeList[1];
```

NodeList 对象 length 属性

NodeList 对象 length 属性定义了节点列表中元素的数量。

```
var myNodelist = document.querySelectorAll("p");
document.getElementById("demo").innerHTML = myNodelist.length;
```

注意

节点列表看起来可能是一个数组，但其实不是。

你可以像数组一样，使用索引来获取元素。

节点列表无法使用数组的方法： valueOf(), pop(), push(), 或 join() 。

HTMLCollection 与 NodeList 的区别

1. HTMLCollection 是 HTML 元素的集合。
2. NodeList 是一个文档节点的集合。
3. NodeList 与 HTMLCollection 有很多类似的地方。
4. NodeList 与 HTMLCollection 都与数组对象有点类似，可以使用索引 (0, 1, 2, 3, 4, ...) 来获取元素。
5. NodeList 与 HTMLCollection 都有 length 属性。
6. HTMLCollection 元素可以通过 name，id 或索引来获取。
7. NodeList 只能通过索引来获取。
8. 只有 NodeList 对象有包含属性节点和文本节点。