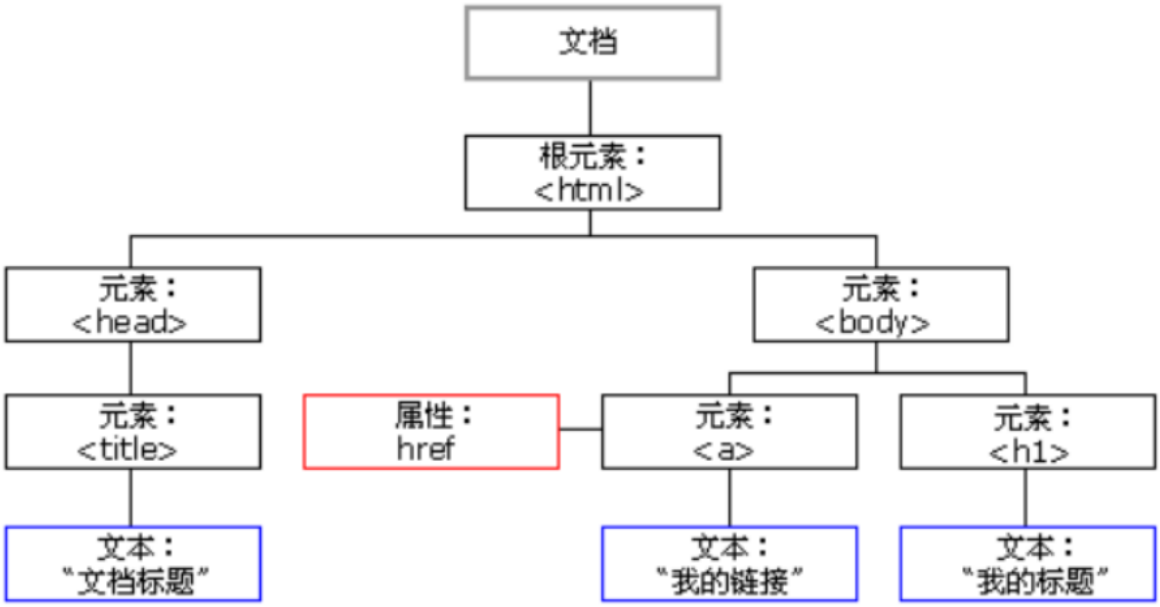
第四章 JavaScript DOM

学习JavaScript这一篇就够了_javascript学习这一篇就够了-CSDN博客

4.1、DOM概述

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 模型被结构化为 对象树：



通过这个对象模型，JavaScript 获得创建动态 HTML 的所有力量：

- JavaScript 能改变页面中的所有 HTML 元素
- JavaScript 能改变页面中的所有 HTML 属性
- JavaScript 能改变页面中的所有 CSS 样式
- JavaScript 能删除已有的 HTML 元素和属性
- JavaScript 能添加新的 HTML 元素和属性
- JavaScript 能对页面中所有已有的 HTML 事件作出反应
- JavaScript 能在页面中创建新的 HTML 事件 换言之：HTML DOM 是关于如何获取、更改、添加或删除 HTML 元素的标准。

4.2、DOM文档节点

4.2.1、节点概述

节点Node，是构成我们网页的最基本的组成部分，网页中的每一个部分都可以称为是一个节点。

比如：html标签、属性、文本、注释、整个文档等都是一个节点。

虽然都是节点，但是实际上它们的具体类型是不同的。

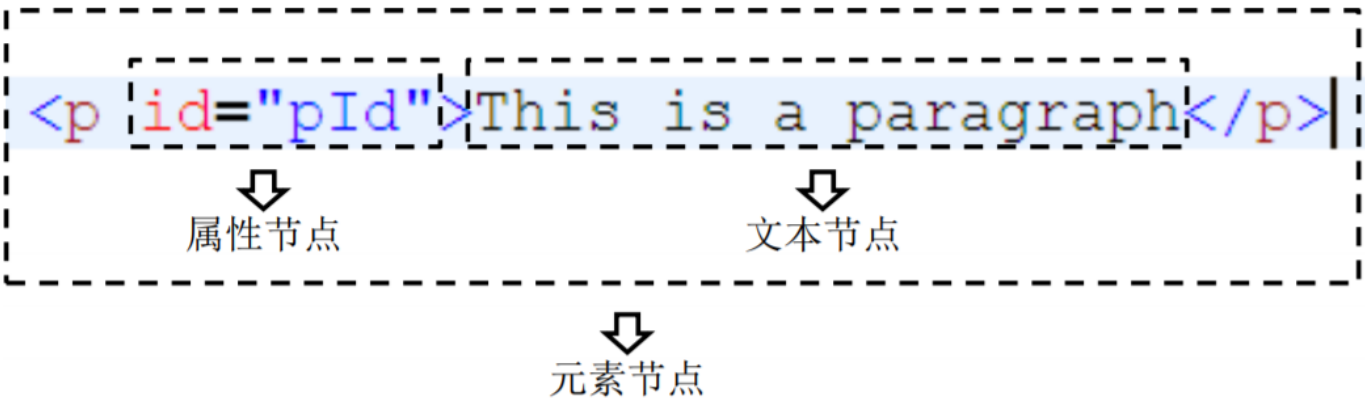
比如：标签我们称为元素节点、属性称为属性节点、文本称为 文本节点、文档称为文档节点。

节点的类型不同，属性和方法也都不尽相同。

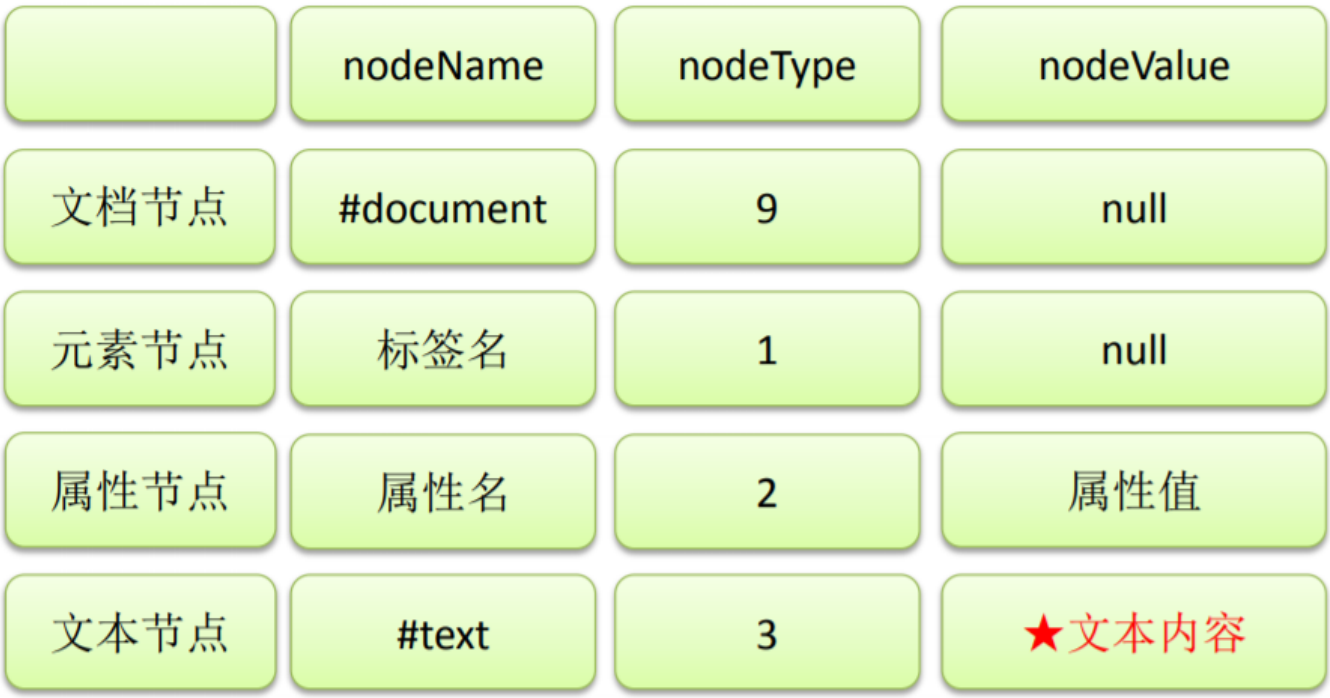
节点：Node——构成HTML文档最基本的单元。

常用节点分为四类：

- 文档节点：整个HTML文档
- 元素节点：HTML文档中的HTML标签
- 属性节点：元素的属性
- 文本节点：HTML标签中的文本内容



4.2.2、节点属性



4.2.3、文档节点

文档节点（Document）代表的是整个HTML文 档，网页中的所有节点都是它的子节点。

document对象作为window对象的属性存在的，我们不用获取可以直接使用。

通过该对象我们可以在整个文档访问内查找节点对象，并可以通过该对象创建各种节点对象。

4.2.4、元素节点

HTML中的各种标签都是元素节点（Element），这也是我们最常用的一个节点。

浏览器会将页面中所有的标签都转换为一个元素节点， 我们可以通过document的方法来获取元素节点。

例如：document.getElementById()，根据id属性值获取一个元素节点对象。

4.2.5、属性节点

属性节点（Attribute）表示的是标签中的一个一个的属 性，这里要注意的是属性节点并非是元素节点的子节点，而是元素节点的一部分。可以通过元素节点来获取指定的属性节点。

例如：元素节点.getAttributeNode("属性名")，根据元素节点的属性名获取一个属性节点对象。

> 注意：我们一般不使用属性节点。

4.2.6、文本节点

文本节点（Text）表示的是HTML标签以外的文本内容，任意非HTML的文本都是文本节点，它包括可以字面解释的纯文本内容。文本节点一般是作为元素节点的子节点存在的。获取文本节点时，一般先要获取元素节点，在通过元素节点获取文本节点。

例如：元素节点.firstChild;，获取元素节点的第一个子节点，一般为文本节点。

4.3、DOM文档操作

文档对象代表您的网页，，如果您希望访问 HTML 页面中的任何元素，那么您总是从访问 document 对象开始。

下面是一些如何使用 document 对象来访问和操作 HTML 的实例。

4.3.1、查找 HTML 元素

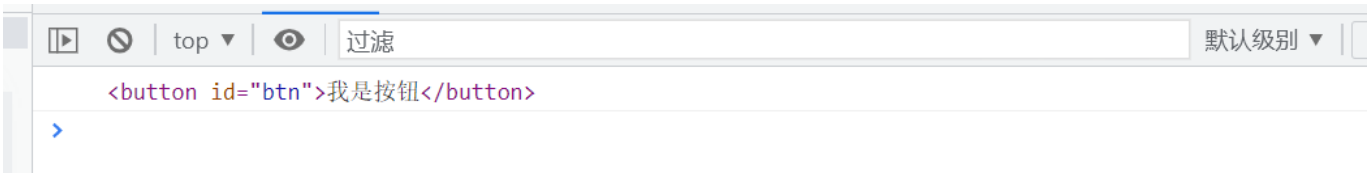
4.3.1.1、方法介绍

方法	描述
document.getElementById(*id*)	通过元素 id 来查找元素。
document.getElementsByTagName(*name*)	通过标签名来查找元素。
document.getElementsByClassName(*name*)	通过类名来查找元素。
document.querySelector(*CSS选择器*)	通过CSS选择器选择一个元素。
document.querySelectorAll(*CSS选择器*)	通过CSS选择器选择多个元素。

4.3.1.2、方法演示

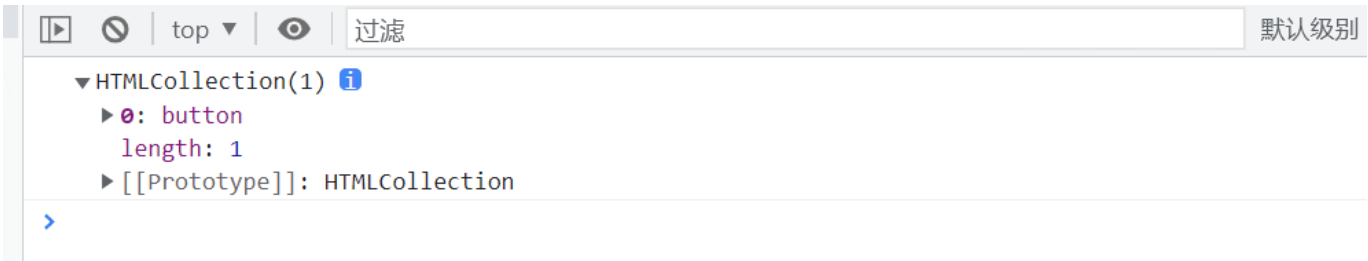
需求描述：创建一个按钮，通过id获取按钮节点对象

```
<body>
<button id="btn">我是按钮</button>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var btn = document.getElementById("btn");
    console.log(btn);
</script>
</body>
```



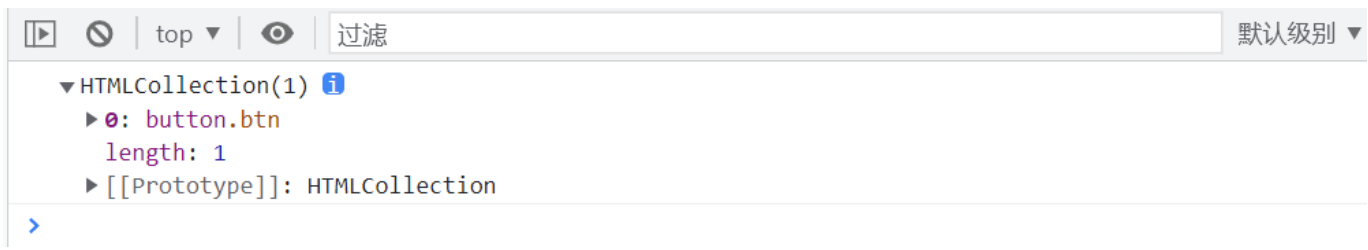
需求描述：创建一个按钮，通过标签名获取按钮节点对象数组

```
<body>
<button>我是按钮</button>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var btn = document.getElementsByTagName("button");
    console.log(btn);
</script>
</body>
```



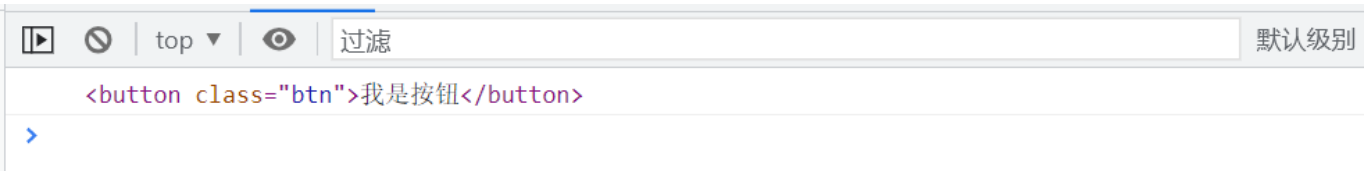
需求描述：创建一个按钮，通过类名获取按钮节点对象数组

```
<body>
<button class="btn">我是按钮</button>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var btn = document.getElementsByClassName("btn");
    console.log(btn);
</script>
</body>
```



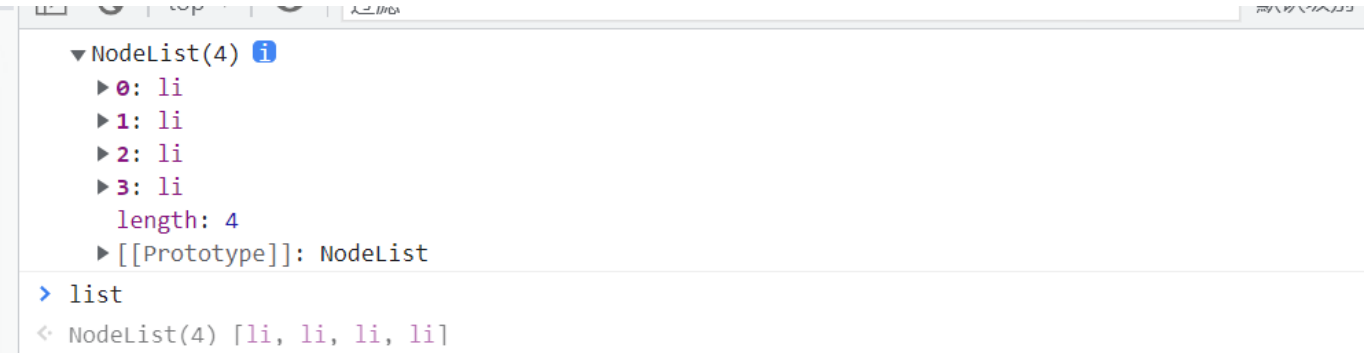
需求描述：创建一个按钮，通过CSS选择器选择该按钮

```
<body>
<button class="btn">我是按钮</button>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var btn = document.querySelector(".btn");
    console.log(btn);
</script>
</body>
```



需求描述：创建一个无序列表，通过CSS选择器选择该列表的所有li

```
<body>
<ul class="list">
    <li>列表项1</li>
    <li>列表项2</li>
    <li>列表项3</li>
    <li>列表项4</li>
</ul>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var list = document.querySelectorAll(".list li");
    console.log(list);
</script>
</body>
```



4.3.2、获取 HTML 的值

4.3.2.1、方法介绍

方法	描述
元素节点.innerText	获取 HTML 元素的 inner Text。
元素节点.innerHTML	获取 HTML 元素的 inner HTML。
元素节点.属性	获取 HTML 元素的属性值。
元素节点.getAttribute(*attribute*)	获取 HTML 元素的属性值。
元素节点.style.样式	获取 HTML 元素的行内样式值。

4.3.2.2、方法演示

需求描述：创建一个按钮，然后获取按钮的文本内容

```
<body>
<button id="btn">我是按钮</button>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var btn = document.getElementById("btn");
    console.log(btn.innerText);
</script>
</body>
```

需求描述：创建一个div，然后在div中插入一个h1标题，获取div中的html代码

```
<body>
<div id="box">
    <h1>我是Box中的大标题</h1>
</div>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var box = document.getElementById("box");
    console.log(box.innerHTML);
</script>
</body>
```

需求描述：创建一个超链接，默认为空，设置href属性为https://www.baidu.com ，使用JavaScript代码读取href属性

```
<body>
<a id="a" href="https://www.baidu.com">打开百度，你就知道！</a>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var a = document.getElementById("a");
    console.log(a.href);
</script>
</body>
```

```
<body>
<a id="a" href="https://www.baidu.com">打开百度，你就知道！</a>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var a = document.getElementById("a");
    console.log(a.getAttribute("href"));
</script>
</body>
```

需求描述：创建一个正方形div，默认颜色为红色，使用JavaScript代码获取div的宽度

> 注意：如果CSS的样式名中含有-，这种名称在JS中是不合法的比如background-color，需要将这种样式名修改为驼峰命名法，去掉-，然后将-后的字母大写，我们通过style属性设置的样式都是行内样式，同样的获取也是行内样式，而行内样式有较高的优先级，所以通过JS修改的样式往往会立即显示，但是如果在样式中写了!important，则此时样式会有最高的优先级，即使通过JS也不能覆盖该样式，此时将会导致JS修改样式失效，所以尽量不要为样式添加!important

```
<body>
<div style="width: 100px;height: 100px;background: red;" id="box"></div>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var box = document.getElementById("box");
    console.log(box.style.width);
</script>
</body>
```

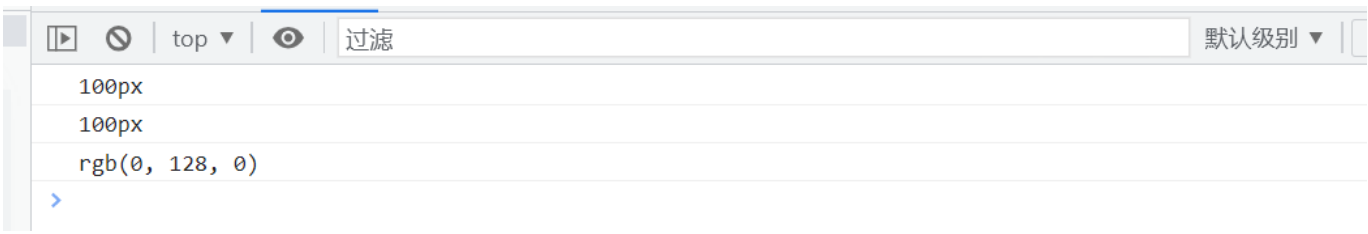
拓展知识1：

通过style属性设置和读取的都是内联样式，无法读取样式表中的样式或者说正在应用的样式，如果想要读取当前正在应用的样式属性我们可以使用元素.currentStyle.样式名来获取元素的当前显示的样式，它可以用来读取当前元素正在显示的样式，如果当前元素没有设置该样式，则获取它的默认值，但是currentStyle只有IE浏览器支持，其它的浏览器都不支持，在其它浏览器中可以使用getComputedStyle()这个方法来获取元素当前的样式，这个方法是window的方法，可以直接使用，但是需要两个参数：

- 第一个参数：要获取样式的元素
- 第二个参数：可以传递一个伪元素，一般都传null

该方法会返回一个对象，对象中封装了当前元素对应的样式，可以通过 对象.样式名 来读取样式，如果获取的样式没有设置，则会获取到真实的值，而不是默认值，比如：没有设置width，它不会获取到auto，而是一个长度，但是该方法不支持IE8及以下的浏览器。通过currentStyle和getComputedStyle()读取到的样式都是只读的，不能修改，如果要修改必须通过style属性，因此，我们可以写一个**适配各个浏览器的读取元素样式的方法。**

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
    <style>
        /*样式表的样式*/
        #box {
            width: 200px;
            height: 200px;
            background-color: green;
        }
    </style>
</head>
<body>
<div style="width: 100px;height: 100px;" id="box"></div>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    /*通用的获取元素样式的方法*/
    function getStyle(obj, name) {
        if (window.getComputedStyle) {
            //正常浏览器的方式，具有getComputedStyle()方法
            return getComputedStyle(obj, null)[name];
        } else {
            //IE8的方式，没有getComputedStyle()方法
            return obj.currentStyle[name];
        }
    }

    var box = document.getElementById("box");

    console.log(getStyle(box, "width"));
    console.log(getStyle(box, "height"));
    console.log(getStyle(box, "background-color"));
</script>
</body>
</html>
```



> 建议设置颜色数值一般采用rgb或者rgba

拓展知识2：编写一段兼容性代码，用来获取任意标签的文本内容

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<a href="https://www.baidu.com" id="a">打开百度，你就知道！</a>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var a = document.getElementById("a");

    console.log(getInnerText(a));

    /*获取任意标签的内容*/
    function getInnerText(element) {
        // 判断浏览器是否支持textContent,如果支持，
        //则使用textContent获取内容，否则使用innerText获取内容。
        if(typeof element.textContent == "undefined") {
            return element.innerText;
        } else {
            return element.textContent;
        }
    }
</script>
</body>
</html>
```

4.3.3、改变 HTML 的值

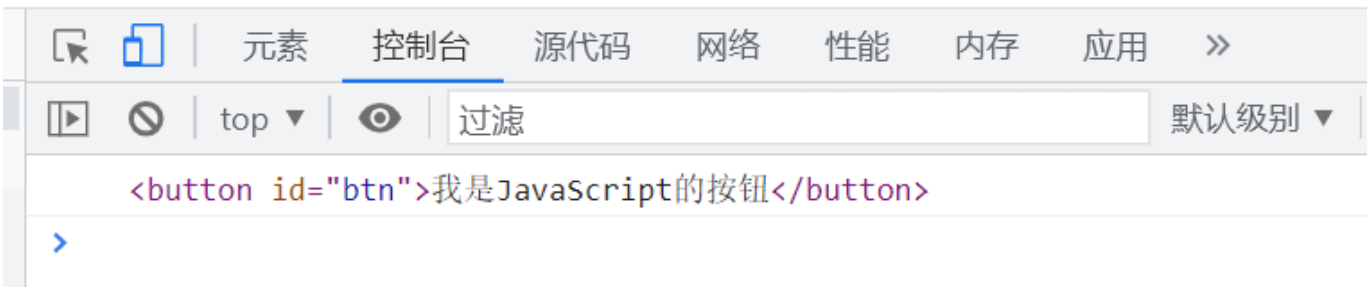
4.3.3.1、方法介绍

方法	描述
元素节点.innerText = *new text content*	改变元素的 inner Text。
元素节点.innerHTML = *new html content*	改变元素的 inner HTML。
元素节点.属性 = *new value*	改变 HTML 元素的属性值。
元素节点.setAttribute(*attribute*, *value*)	改变 HTML 元素的属性值。
元素节点.style.样式 = *new style*	改变 HTML 元素的行内样式值。

4.3.3.2、方法演示

需求描述：创建一个按钮，然后改变按钮的文本内容

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<button id="btn">我是按钮</button>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var btn = document.getElementById("btn");
    btn.innerText = "我是JavaScript的按钮";
    console.log(btn);
</script>
</body>
</html>
```



需求描述：创建一个div，然后在div中插入一个h1标题

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<div id="box"></div>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var box = document.getElementById("box");
    box.innerHTML = "<h1>我是Box中的大标题</h1>";
    console.log(box);
</script>
</body>
</html>
```

我是Box中的大标题

需求描述：创建一个超链接，默认为空，使用JavaScript代码设置href属性为 https://www.baidu.com

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<a id="a" href="">打开百度，你就知道！</a>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var a = document.getElementById("a");
    a.href="https://www.baidu.com";
    console.log(a);
</script>
</body>
</html>
```

```
<body>
<a id="a" href="">打开百度，你就知道！</a>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var a = document.getElementById("a");
    a.setAttribute("href", "https://www.baidu.com");
    console.log(a);
</script>
</body>
```

需求描述：创建一个正方形div，默认颜色为红色，使用JavaScript代码改变为绿色

> 注意：如果CSS的样式名中含有-，这种名称在JS中是不合法的比如background-color，需要将这种样式名修改为驼峰命名法，去掉-，然后将-后的字母大写，我们通过style属性设置的样式都是行内样式，同样的获取也是行内样式，而行内样式有较高的优先级，所以通过JS修改的样式往往会立即显示，但是如果在样式中写了!important，则此时样式会有最高的优先级，即使通过JS也不能覆盖该样式，此时将会导致JS修改样式失效，所以尽量不要为样式添加!important

```
<body>
<div style="width: 100px;height: 100px;background: red;" id="box"></div>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var box = document.getElementById("box");
    box.style.background = "green";
    console.log(box);
</script>
</body>
```

拓展知识1：

修改节点的内容除了常用的innerHTML和innerText之外，还有insertAdjacentHTML和insertAdjacentText方法，可以在指定的地方插入内容。insertAdjacentText方法与insertAdjacentHTML方法类似，只不过是插入纯文本，参数相同。

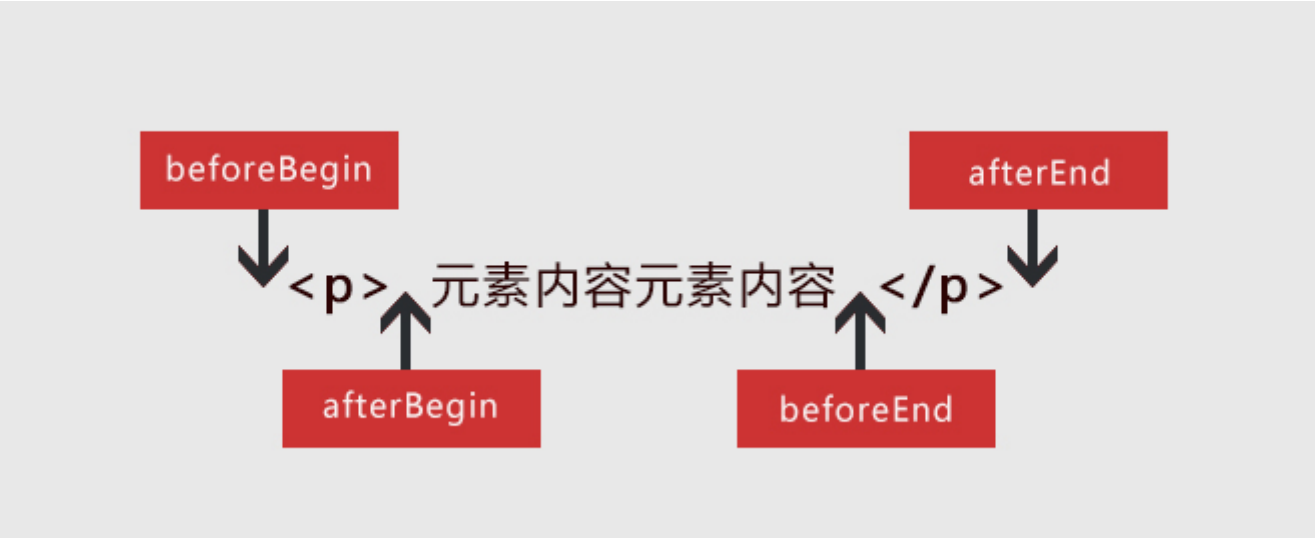
语法说明：

```
<script>
object.insertAdjacentHTML(where,html);
object.insertAdjacentText(where,text)
</script>
```

参数说明：

where：

- beforeBegin：插入到开始标签的前面
- beforeEnd：插入到结束标签的前面
- afterBegin：插入到开始标签的后面
- afterEnd：插入到结束标签的后面



- html：一段html代码
- text：一段文本值

注意事项：

1.这两个方法必须等文档加载好后才能执行，否则会出错。 2.insertAdjacentText只能插入普通文本，insertAdjacentHTML插入html代码。 3.使用insertAdjacentHTML方法插入script脚本文件时，必须在script元素上定义defer属性。 4.使用insertAdjacentHTML方法插入html代码后，页面上的元素集合将发生变化。 5.insertAdjacentHTML方法不适用于单个的空的元素标签(如img，input等)。

例子演示：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<div id="insert">
    <p>你是我的小丫小苹果</p>
</div>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var div = document.getElementById("insert");
    div.insertAdjacentHTML('beforeBegin', '你是我的小丫小苹果');
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html> == $0
  <head>…</head>
  <body>
    "你是我的小丫小苹果"
    <div id="insert">
      <p>你是我的小丫小苹果</p>
    </div>
    <!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
    <script>…</script>
  </body>
</html>
```

拓展知识2：编写一段兼容性代码，用来设置任意标签的文本内容

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<a href="https://www.baidu.com" id="a">打开百度，你就知道！</a>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var a = document.getElementById("a");

    setInnerText(a, "你要打开百度吗？");

    console.log(getInnerText(a));

    /*获取任意标签的内容*/
    function getInnerText(element) {
        // 判断浏览器是否支持textContent,如果支持，
//则使用textContent获取内容，否则使用innerText获取内容。
        if (typeof element.textContent == "undefined") {
            return element.innerText;
        } else {
            return element.textContent;
        }
    }

    /*设置任意标签的内容*/
    function setInnerText(element, text) {
// 判断浏览器是否支持textContent,
//如果支持，则使用textContent设置内容，否则使用innerText设置内容。
        if (typeof element.textContent == "undefined") {
            return element.innerText = text;
        } else {
            return element.textContent = text;
        }
    }
</script>
</body>
</html>
```

4.3.4、修改 HTML 元素

4.3.4.1、方法介绍

方法	描述
document.createElement(*element*)	创建 HTML 元素节点。
document.createAttribute(*attribute*)	创建 HTML 属性节点。
document.createTextNode(*text*)	创建 HTML 文本节点。
元素节点.removeChild(*element*)	删除 HTML 元素。
元素节点.appendChild(*element*)	添加 HTML 元素。
元素节点.replaceChild(*element*)	替换 HTML 元素。
元素节点.insertBefore(*element*)	在指定的子节点前面插入新的子节点。

4.3.4.2、方法演示

案例演示1：创建一个ul列表，然后在该列表中追加4个li标签

第一种方法：

```
<body>
<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var ul = document.createElement("ul");

    var li1 = document.createElement("li");
    var text1 = document.createTextNode("列表项1");
    li1.appendChild(text1);
    ul.appendChild(li1);

    var li2 = document.createElement("li");
    var text2 = document.createTextNode("列表项2");
    li2.appendChild(text2);
    ul.appendChild(li2);

    var li3 = document.createElement("li");
    var text3 = document.createTextNode("列表项3");
    li3.appendChild(text3);
    ul.appendChild(li3);

    var li4 = document.createElement("li");
    var text4 = document.createTextNode("列表项4");
    li4.appendChild(text4);
    ul.appendChild(li4);

    document.getElementsByTagName("body")[0].appendChild(ul);
    console.log(document.getElementsByTagName("body")[0]);
</script>
</body>
```

第二种方法：

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var ul = document.createElement("ul");

    var li1 = document.createElement("li");
    li1.innerHTML = "列表项1";
    ul.appendChild(li1);

    var li2 = document.createElement("li");
    li2.innerHTML = "列表项2";
    ul.appendChild(li2);

    var li3 = document.createElement("li");
    li3.innerHTML = "列表项3";
    ul.appendChild(li3);

    var li4 = document.createElement("li");
    li4.innerHTML = "列表项4";
    ul.appendChild(li4);

    document.getElementsByTagName("body")[0].appendChild(ul);
</script>
</body>
</html>
```

第三种方法：

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var ul = document.createElement("ul");

    var li1 = "<li>列表项1</li>";
    var li2 = "<li>列表项2</li>";
    var li3 = "<li>列表项3</li>";
    var li4 = "<li>列表项4</li>";
    ul.innerHTML = li1 + li2 + li3 + li4;

    document.getElementsByTagName("body")[0].appendChild(ul);
</script>
</body>
</html>
```

- 列表项1
- 列表项2
- 列表项3
- 列表项4

案例演示2：创建一个ul列表，里边有四个li子元素，删除第一个li，替换最后一个li

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<ul id="ul">
    <li id="first">列表项1</li>
    <li>列表项2</li>
    <li>列表项3</li>
    <li id="last">列表项4</li>
</ul>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var ul = document.getElementById("ul");
    var first = document.getElementById("first");
    var last = document.getElementById("last");

    /*删除第一个*/
    ul.removeChild(first);

    /*替换最后一个*/
    var replaceLi = document.createElement("li");
    replaceLi.innerHTML = "列表4的替换";
    ul.replaceChild(replaceLi, last);
</script>
</body>
</html>
```

案例演示3：创建一个ul列表，里边有四个li子元素，在第一个li前边插入一个id为zero的li

```
<body>
<ul id="ul">
    <li id="first">列表项1</li>
    <li>列表项2</li>
    <li>列表项3</li>
    <li>列表项4</li>
</ul>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var ul = document.getElementById("ul");
    var first = document.getElementById("first");

    var zero = document.createElement("li");
    zero.innerHTML = "列表0的新增";

    ul.insertBefore(zero, first);
</script>
</body>
```

4.3.5、查找 HTML 父子

4.3.5.1、方法介绍

方法	描述
元素节点.parentNode	返回元素的父节点。
元素节点.parentElement	返回元素的父元素。
元素节点.childNodes	返回元素的一个子节点的数组（包含空白文本Text节点）。
元素节点.children	返回元素的一个子元素的集合（不包含空白文本Text节点）。
元素节点.firstChild	返回元素的第一个子节点（包含空白文本Text节点）。
元素节点.firstElementChild	返回元素的第一个子元素（不包含空白文本Text节点）。
元素节点.lastChild	返回元素的最后一个子节点（包含空白文本Text节点）。
元素节点.lastElementChild	返回元素的最后一个子元素（不包含空白文本Text节点）。
元素节点.previousSibling	返回某个元素紧接之前节点（包含空白文本Text节点）。
元素节点.previousElementSibling	返回指定元素的前一个兄弟元素（相同节点树层中的前一个元素节点）。
元素节点.nextSibling	返回某个元素紧接之后节点（包含空白文本Text节点）。
元素节点.nextElementSibling	返回指定元素的后一个兄弟元素（相同节点树层中的下一个元素节点）。

4.3.5.2、方法演示

案例演示：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<div id="box">
    <ul id="ul">
        <li><a href="https://www.baidu.com">我是超链接1</a></li>
        <li id="two"><a href="https://www.baidu.com">我是超链接2</a></li>
        <li><a href="https://www.baidu.com">我是超链接3</a></li>
        <li><a href="https://www.baidu.com">我是超链接4</a></li>
    </ul>
</div>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var box = document.getElementById("box");
    var ul = document.getElementById("ul");
    var two = document.getElementById("two");

</script>
</body>
</html>
```

```
> ul.parentElement
  ▶ <div id="box">…</div>
> ul.parentNode
  ▶ <div id="box">…</div>
```

```
> box.childNodes
  ▶ NodeList(3) [text, ul#ul, text]
> box.children
  ▶ HTMLCollection [ul#ul, ul: ul#ul]
```

```
> ul.firstChild
  ▶ #text
> ul.firstElementChild
  ▶ <li>…</li>
> ul.lastChild
  ▶ #text
> ul.lastElementChild
  ▶ <li>…</li>
>
```

```
> two.previousSibling
  ▶ #text
> two.previousElementSibling
  ▶ <li>…</li>
> two.nextSibling
  ▶ #text
> two.nextElementSibling
  ▶ <li>…</li>
```

兼容性方法：

```
<script>
    /*获取任意一个父级元素的第一个子元素*/
function getfirstElementChild(element) {
    if(element.firstElementChild) {
        return element.firstElementChild;
    } else {
        var node = element.firstChild;
        while(node && node.nodeType != 1) {
            node = node.nextSibling;
        }
        return node;
    }
}

/*获取任意一个父级元素的最后一个子元素*/
function getLastElementChild(element) {
    if(element.lastElementChild) {
        return element.lastElementChild;
    } else {
        var node = element.lastChild;
        while(node && node.nodeType != 1) {
            node = node.previousSibling;
        }
        return node;
    }
}

/*获取任意一个子元素的前一个兄弟元素*/
function getPreviousElementSibling(element) {
    if(element.previousElementSibling) {
        return element.previousElementSibling;
    } else {
        var node = element.previousSibling;
        while(node && node.nodeType != 1) {
            node = node.previousSibling;
        }
        return node;
    }
}

/*获取任意一个子元素的后一个兄弟元素*/
function getNextElementSibling(element) {
    if(element.nextElementSibling) {
        return element.nextElementSibling;
    } else {
        var node = element.nextSibling;
        while(node && node.nodeType != 1) {
            node = node.nextSibling;
        }
        return node;
```

```
    }
}

</script>
```

4.4、DOM文档事件

4.4.1、事件概述

HTML事件可以触发浏览器中的行为，比方说当用户点击某个 HTML 元素时启动一段 JavaScript。

4.4.2、窗口事件

由窗口触发该事件 (同样适用于 < body > 标签)：

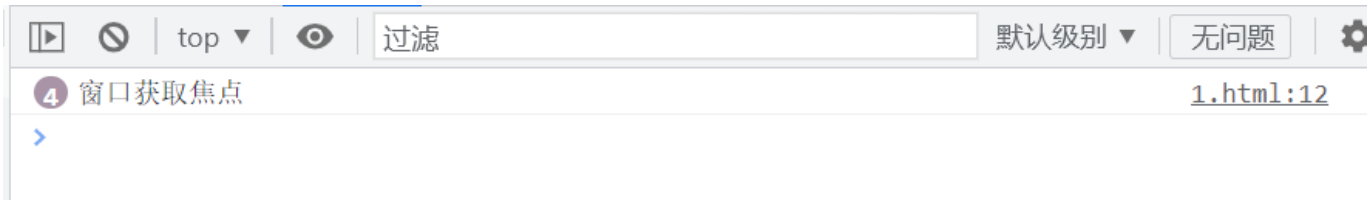
属性	描述
onblur	当窗口失去焦点时运行脚本。
onfocus	当窗口获得焦点时运行脚本。
onload	当文档加载之后运行脚本。
onresize	当调整窗口大小时运行脚本。
onstorage	当 Web Storage 区域更新时（存储空间中的数据发生变化时）运行脚本。

案例演示1：当窗口失去焦点时，输出“窗口失去焦点”

```
<script>
    window.onblur = function () {
        console.log("窗口失去焦点");
    };
</script>
```

案例演示2：当窗口获取焦点时，输出“窗口获取焦点”

```
<script>
    window.onfocus = function () {
        console.log("窗口获取焦点");
    };
</script>
```



案例演示3：当页面文档加载完成后，输出"Hello, World"

```
<script>
    window.onload = function () {
        console.log("Hello,World");
    };
</script>
```

案例演示4：当调整窗口大小时，输出"窗口大小正在改变"

```
<script>
    window.onresize = function () {
        console.log("窗口大小正在改变");
    };
</script>
```

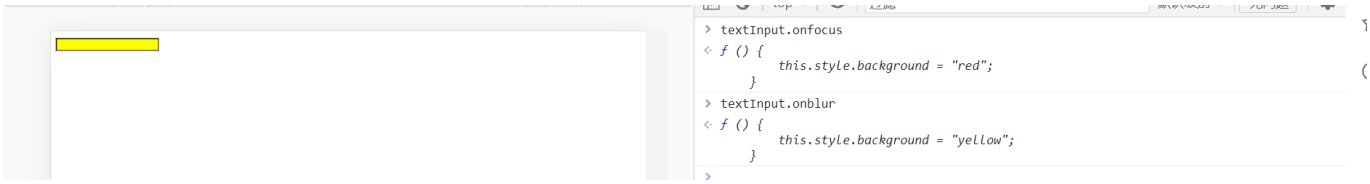
4.4.3、表单事件

表单事件在HTML表单中触发 (适用于所有 HTML 元素，但该HTML元素需在form表单内)：

属性	描述
onblur	当元素失去焦点时运行脚本。
onfocus	当元素获得焦点时运行脚本。
onchange	当元素改变时运行脚本。
oninput	当元素获得用户输入时运行脚本。
oninvalid	当元素无效时运行脚本。
onselect	当选取元素时运行脚本。
onsubmit	当提交表单时运行脚本。

案例演示1：当文本框获取焦点，文本框背景为红色，当文本框失去焦点，文本框背景为黄色

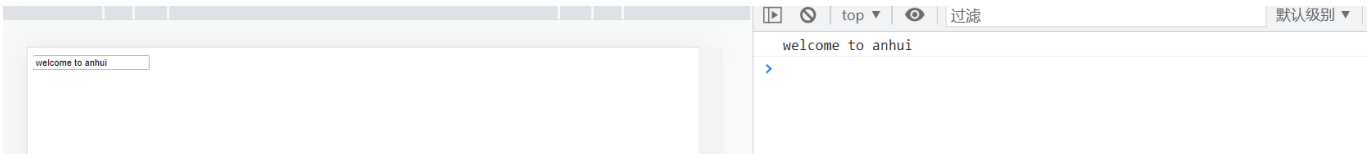
```
<body>
<form>
    <input type="text" id="text">
</form>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var textInput = document.getElementById("text");

    /* 当文本框获取焦点，文本框背景为红色 */
    textInput.onfocus = function () {
        this.style.background = "red";
    };

    /* 当文本框失去焦点，文本框背景为绿色 */
    textInput.onblur = function () {
        this.style.background = "yellow";
    };
</script>
</body>
```



> 注意：这里为什么要用this，你不用this也可以，就直接textInput.style.background = "red";也不是不可以的，但是方法的调用规则就是谁调用this，this就指向谁，这样我们就可以简化代码了

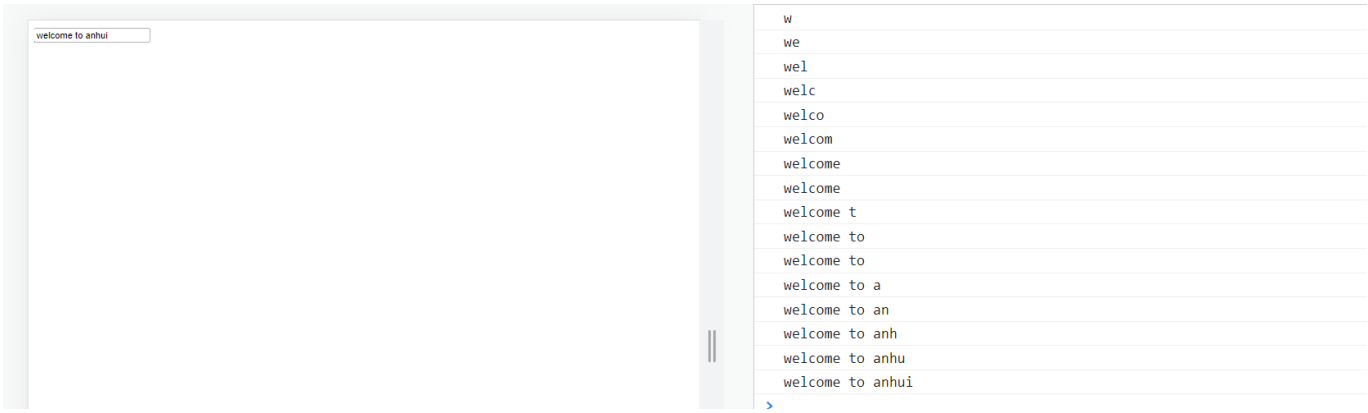
案例演示2：当文本框内容改变时，鼠标离开文本框，自动将文本框的内容输出到控制台

```
<body>
<form>
    <input type="text" id="text">
</form>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var textInput = document.getElementById("text");

    /* 当文本框内容改变时，鼠标离开文本框，自动将文本框的内容输出到控制台 */
    textInput.onchange = function () {
        console.log(this.value);
    };
</script>
</body>
```



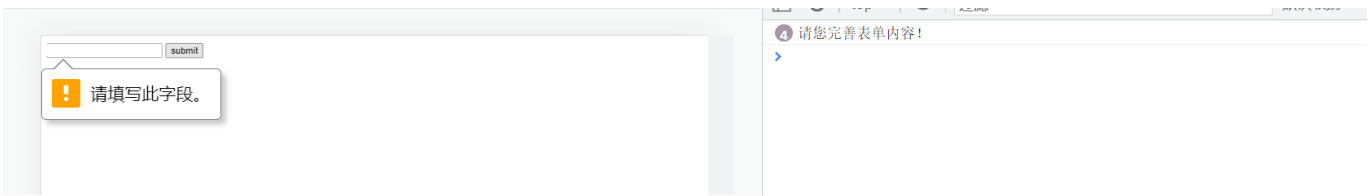
案例演示3：当文本框内容改变时，立即将改变的内容输出到控制台

```
<body>
<form>
    <input type="text" id="text">
</form>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var textInput = document.getElementById("text");

    /* 当文本框内容改变时，立即将改变的内容输出到控制台 */
    textInput.oninput = function () {
        console.log(this.value);
    };
</script>
</body>
```



案例演示4：如果单击“submit”，不填写文本字段，将发生警报消息

```
<body>
<form>
    <input type="text" id="text" required>
    <input type="submit" value="submit">
</form>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var textInput = document.getElementById("text");

    /* 如果单击“submit”，则不填写文本字段，将发生警报消息 */
    textInput.oninvalid = function () {
        console.log("请您完善表单内容！");
    };
</script>
</body>
```

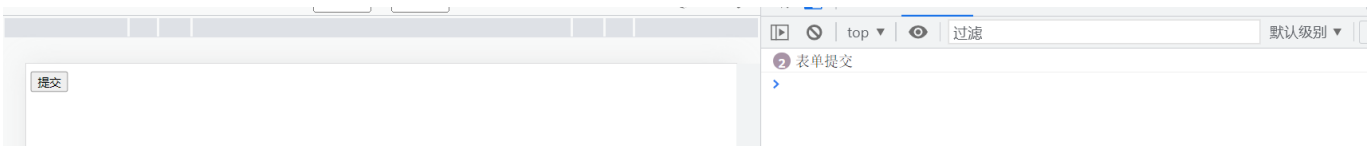


案例演示5：当选中文本框的内容时，输出“您已经选择了文本框内容！”

```
<body>
<form>
    <input type="text" id="text">
</form>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var textInput = document.getElementById("text");

    /* 当选中文本框的内容时，输出“您已经选择了文本框内容！” */
    textInput.onselect = function () {
        console.log("您已经选择了文本框内容！");
    };
</script>
</body>
```

案例演示6：当提交表单的时候，在控制台输出“表单提交”

```
<body>
<form id="myform">
    <input type="submit" id="submit">
</form>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var myform = document.getElementById("myform");

    /* 当提交表单的时候，在控制台输出“表单提交” */
    myform.onsubmit = function () {
        console.log("表单提交");
        return false;/* 用来阻止表单提交的，你不写它会跳转请求 */
    };
</script>
</body>
```



4.4.4、键盘事件

通过键盘触发事件，类似用户的行为：

属性	描述
onkeydown	当按下按键时运行脚本。
onkeyup	当松开按键时运行脚本。
onkeypress	当按下并松开按键时运行脚本。

案例演示1：当键盘按下判断当前的按键是不是 a ， 如果是就输出true，否则输出false

```
<body>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    /* 当键盘按下判断当前的按键是不是 a ， 如果是就输出true，否则输出false */
    window.onkeydown = function (event) {
        /* 解决兼容性问题 */
        event = event || window.event;

        if (event.keyCode == 65) {
            console.log("true");
        } else {
            console.log("false");
        }
    };
</script>
</body>
```

案例演示2：使div可以根据不同的方向键向不同的方向移动

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<div id="box" style="width: 100px;height: 100px;backgroun
d: red;position: absolute;"></div>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var box = document.getElementById("box");

    //为document绑定一个按键按下的事件
    document.onkeydown = function (event) {
        event = event || window.event;

        // 定义移动速度
        var speed = 10;

        // 选择移动方向
        switch (event.keyCode) {
            case 37:
                box.style.left = box.offsetLeft - speed + "px";
                break;
            case 39:
                box.style.left = box.offsetLeft + speed + "px";
                break;
            case 38:
                box.style.top = box.offsetTop - speed + "px";
                break;
            case 40:
                box.style.top = box.offsetTop + speed + "px";
                break;
        }
    };
</script>
</body>
</html>
```

拓展知识：

当事件的响应函数被触发时，浏览器每次都会将一个事件对象作为实参传递进响应函数。

Event 对象代表事件的状态，比如事件在其中发生的元素、键盘按键的状态、鼠标的位置、鼠标的状态。

在IE8中，响应函数被触发时，浏览器不会传递事件对象，在IE8及以下的浏览器中，是将事件对象作为window对象的属性保存的。

解决事件对象的兼容性问题：event = event || window.event;

4.4.5、鼠标事件

键鼠属性：

clientX	以浏览器窗口左上顶角为原点，定位 x 轴坐标	所有浏览器，不兼容 Safari
clientY	以浏览器窗口左上顶角为原点，定位 y 轴坐标	所有浏览器，不兼容 Safari
offsetX	以当前事件的目标对象左上顶角为原点，定位 x 轴坐标	所有浏览器，不兼容 Mozilla
offsetY	以当前事件的目标对象左上顶角为原点，定位 y 轴坐标	所有浏览器，不兼容 Mozilla
pageX	以 document 对象（即文档窗口）左上顶角为原点，定位 x 轴坐标	所有浏览器，不兼容 IE
pageY	以 document 对象（即文档窗口）左上顶角为原点，定位 y 轴坐标	所有浏览器，不兼容 IE
screenX	计算机屏幕左上顶角为原点，定位 x 轴坐标	所有浏览器
screenY	计算机屏幕左上顶角为原点，定位 y 轴坐标	所有浏览器
layerX	最近的绝对定位的父元素（如果没有，则为 document 对象）左上顶角为元素，定位 x 轴坐标	Mozilla 和 Safari
layerY	最近的绝对定位的父元素（如果没有，则为 document 对象）左上顶角为元素，定位 y 轴坐标	Mozilla 和 Safari

通过鼠标触发事件，类似用户的行为：

属性	描述
onclick	当单击鼠标时运行脚本。
ondblclick	当双击鼠标时运行脚本。
onmousedown	当按下鼠标按钮时运行脚本。
onmouseup	当松开鼠标按钮时运行脚本。
onmousemove	当鼠标指针移动时运行脚本。
onmouseover	当鼠标指针移至元素之上时运行脚本，不可以阻止冒泡。
onmouseout	当鼠标指针移出元素时运行脚本，不可以阻止冒泡。
onmouseenter	当鼠标指针移至元素之上时运行脚本，可以阻止冒泡。
onmouseleave	当鼠标指针移出元素时运行脚本，可以阻止冒泡。
onmousewheel	当转动鼠标滚轮时运行脚本。
onscroll	当滚动元素的滚动条时运行脚本。

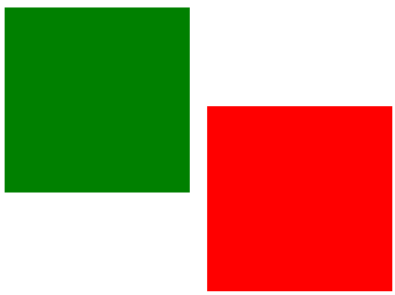
案例演示1：创建一个正方形div，默认颜色为黑色，当鼠标移入div，背景颜色变为红色，当鼠标移出div，背景颜色变为绿色

```
<body>
<div id="box" style="width: 100px;height: 100px
;background: black;"></div>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var box = document.getElementById("box");

    /* 当鼠标移入div，背景颜色变为红色 */
    box.onmouseenter = function () {
        this.style.background = "red";
    };

    /* 当鼠标移出div，背景颜色变为绿色 */
    box.onmouseleave = function () {
        this.style.background = "green";
    };
</script>
</body>
```

案例演示2：编写一个通用的拖拽元素函数，创建两个div，进行拖拽演示，要求兼容IE8、火狐、谷歌等主流浏览器

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<div id="box1" style="width: 100px;height: 100px;background: red;
position: absolute;"></div>
<div id="box2" style="width: 100px;height: 100px;background: green;
position: absolute;"></div>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var box1 = document.getElementById("box1");
    var box2 = document.getElementById("box2");

    drag(box1);
    drag(box2);

    /*
     * 提取一个专门用来设置拖拽的函数
     * 参数：开启拖拽的元素
     */
    function drag(obj) {
        //当鼠标在被拖拽元素上按下时，开始拖拽
        obj.onmousedown = function (event) {
            // 解决事件的兼容性问题
            event = event || window.event;

            // 设置obj捕获所有鼠标按下的事件
            /**
             * setCapture():
             * 只有IE支持，但是在火狐中调用时不会报错，
             * 而如果使用chrome调用，它也会报错
             */
            obj.setCapture && obj.setCapture();

            // obj的偏移量 鼠标.clentX - 元素.offsetLeft
            // obj的偏移量 鼠标.clentY - 元素.offsetTop
            var ol = event.clientX - obj.offsetLeft;
            var ot = event.clientY - obj.offsetTop;

            // 为document绑定一个鼠标移动事件
            document.onmousemove = function (event) {
                // 解决事件的兼容性问题
                event = event || window.event;
                // 当鼠标移动时被拖拽元素跟随鼠标移动
                // 获取鼠标的坐标
                var left = event.clientX - ol;
```

```
                var top = event.clientY - ot;
                // 修改obj的位置
                obj.style.left = left + "px";
                obj.style.top = top + "px";
            };

            // 为document绑定一个鼠标松开事件
            document.onmouseup = function () {
                // 取消document的onmousemove事件
                document.onmousemove = null;
                // 取消document的onmouseup事件
                document.onmouseup = null;
                // 当鼠标松开时，取消对事件的捕获
                obj.releaseCapture && obj.releaseCapture();
            };

            /*
             * 当我们拖拽一个网页中的内容时，浏览器会默认去搜索引擎中搜索内容，
             * 此时会导致拖拽功能的异常，这个是浏览器提供的默认行为，
             * 如果不希望发生这个行为，则可以通过return false来取消默认行为，
             * 但是这招对IE8不起作用
             */
            return false;
        };
    }
</script>
</body>
</html>
```



4.4.6、媒体事件

通过视频（videos），图像（images）或音频（audio） 触发该事件。

属性	描述
onabort	当发生中止事件时运行脚本。
oncanplay	当媒介能够开始播放但可能因缓冲而需要停止时运行脚本。
oncanplaythrough	当媒介能够无需因缓冲而停止即可播放至结尾时运行脚本。
ondurationchange	当媒介长度改变时运行脚本。
onemptied	当媒介资源元素突然为空时（网络错误、加载错误等）运行脚本。
onended	当媒介已抵达结尾时运行脚本。
onerror	当在元素加载期间发生错误时运行脚本。
onloadeddata	当加载媒介数据时运行脚本。
onloadedmetadata	当媒介元素的持续时间以及其它媒介数据已加载时运行脚本。
onloadstart	当浏览器开始加载媒介数据时运行脚本。
onpause	当媒介数据暂停时运行脚本。
onplay	当媒介数据将要开始播放时运行脚本。
onplaying	当媒介数据已开始播放时运行脚本。
onprogress	当浏览器正在取媒介数据时运行脚本。
onratechange	当媒介数据的播放速率改变时运行脚本。
onreadystatechange	当就绪状态（ready-state）改变时运行脚本。
onseeked	当媒介元素的定位属性不再为真且定位已结束时运行脚本。
onseeking	当媒介元素的定位属性为真且定位已开始时运行脚本。
onstalled	当取回媒介数据过程中（延迟）存在错误时运行脚本。
onsuspend	当浏览器已在取媒介数据但在取回整个媒介文件之前停止时运行脚本。
ontimeupdate	当媒介改变其播放位置时运行脚本。
onvolumechange	当媒介改变音量亦或当音量被设置为静音时运行脚本。
onwaiting	当媒介已停止播放但打算继续播放时运行脚本。

4.4.7、其它事件

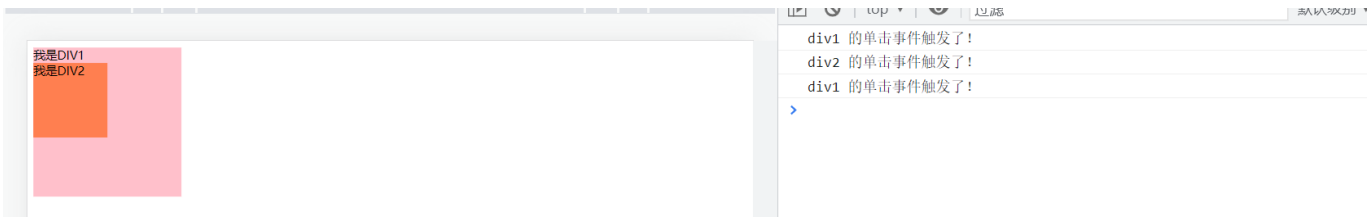
属性	描述
onshow	当 <menu> 元素在上下文显示时触发。
ontoggle	当用户打开或关闭 <details> 元素时触发。

4.4.8、事件冒泡

事件的冒泡（Bubble）：所谓的冒泡指的就是事件的向上传导，当后代元素上的事件被触发时，其祖先元素的相同事件也会被触发，在开发中大部分情况冒泡都是有用的，如果不希望发生事件冒泡可以通过事件对象来取消冒泡。

案例演示1：创建两个div，叠放在一起，分别绑定单击事件，点击最里边的div，会触发两个div的单击事件

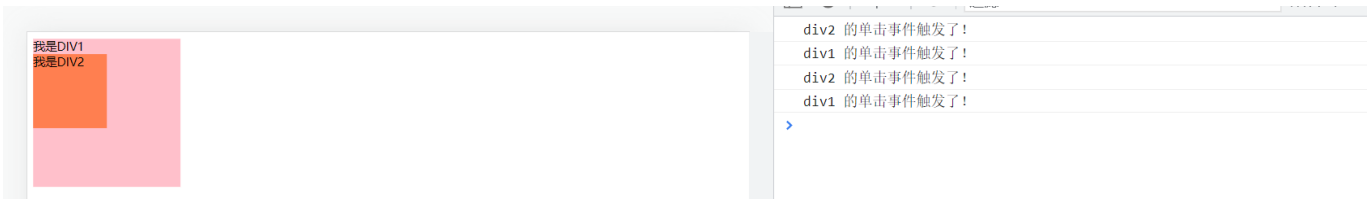
```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
    <style>
        #div1 {
            width: 200px;
            height: 200px;
            background: pink;
        }

        #div2 {
            width: 100px;
            height: 100px;
            background: coral;
        }
    </style>
</head>
<body>
<div id="div1">
    我是DIV1
    <div id="div2">
        我是DIV2
    </div>
</div>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var div1 = document.getElementById("div1");
    var div2 = document.getElementById("div2");

    // 为div1绑定单击事件
    div1.onclick = function () {
        console.log("div1 的单击事件触发了！");
    };

    // 为div2绑定单击事件
    div2.onclick = function () {
        console.log("div2 的单击事件触发了！");
    };
</script>
</body>
</html>
```



案例演示2：创建两个div，叠放在一起，分别绑定单击事件，点击最里边的div，不会触发两个div的单击事件，只会触发自己的单击事件，这时候我们可以取消事件冒泡

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
    <style>
        #div1 {
            width: 200px;
            height: 200px;
            background: pink;
        }

        #div2 {
            width: 100px;
            height: 100px;
            background: coral;
        }
    </style>
</head>
<body>
<div id="div1">
    我是DIV1
    <div id="div2">
        我是DIV2
    </div>
</div>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var div1 = document.getElementById("div1");
    var div2 = document.getElementById("div2");

    // 为div1绑定单击事件
    div1.onclick = function () {
        console.log("div1 的单击事件触发了！");
        stopBubble();
    };

    // 为div2绑定单击事件
    div2.onclick = function () {
        console.log("div2 的单击事件触发了！");
        stopBubble();
    };

    // 取消事件冒泡
    function stopBubble(event) {
        // 如果提供了事件对象，则这是一个非IE浏览器
        if (event && event.stopPropagation) {
            // 因此它支持W3C的stopPropagation()方法
            event.stopPropagation();
```

```
        } else {
            // 否则，我们需要使用IE的方式来取消事件冒泡
            window.event.cancelBubble = true;
        }
    }
</script>
</body>
</html>
```



案例演示3：当点击a标签的时候，阻止a标签的默认跳转事件，采用事件阻止

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
    <style>
        #div1 {
            width: 200px;
            height: 200px;
            background: pink;
        }

        #div2 {
            width: 100px;
            height: 100px;
            background: coral;
        }
    </style>
</head>
<body>
<a href="https://www.baidu.com" id="a">打开百度，你就知道！</a>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var a = document.getElementById("a");

    // 为a绑定单击事件
    a.onclick = function () {
        stopDefault();
    };

    // 阻止浏览器的默认行为
    function stopDefault(event) {
        if (event && event.preventDefault) {
            // 阻止默认浏览器动作(W3C)
            event.preventDefault();
        } else {
            // IE中阻止函数器默认动作的方式
            window.event.returnValue = false;
        }
        return false;
    }
</script>
</body>
</html>
```

4.4.9、事件委派

我们希望只绑定一次事件，即可应用到多个的元素上，即使元素是后添加的，我们可以尝试将其绑定给元素的共同的祖先元素，也就是事件的委派。事件的委派，是指将事件统一绑定给元素的共同的祖先元素，这样当后代元素上的事件触发时，会一直冒泡到祖先元素，从而通过祖先元素的响应函数来处理事件。事件委派是利用了事件冒泡，通过委派可以减少事件绑定的次数，提高程序的性能。

案例演示：为ul列表中的所有a标签都绑定单击事件

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<ul id="u1">
    <li><a href="javascript:;" class="link">超链接一</a></li>
    <li><a href="javascript:;" class="link">超链接二</a></li>
    <li><a href="javascript:;" class="link">超链接三</a></li>
</ul>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var u1 = document.getElementById("u1");

    // 为ul绑定一个单击响应函数
    u1.onclick = function (event) {
        event = event || window.event;
        // 如果触发事件的对象是我们期望的元素，则执行，否则不执行
        if (event.target.className == "link") {
            console.log("我是ul的单击响应函数");
        }
    };
</script>
</body>
</html>
```

4.4.10、事件绑定

一个事件对应多个函数

案例演示：为按钮1的单击事件绑定两个函数，然后点击按钮2取消按钮1的单机事件绑定函数f1

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<button id="btn1">按钮1</button>
<button id="btn2">按钮2</button>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    function f1() {
        console.log("output1 ...");
    };

    function f2() {
        console.log("output2 ...");
    };

    // 为按钮1的单击事件绑定两个函数
    addEventListener(document.getElementById("btn1"), "click", f1);
    addEventListener(document.getElementById("btn1"), "click", f2);

    // 点击按钮2取消按钮1的单机事件绑定函数f1
    document.getElementById("btn2").onclick = function () {
        removeEventListener(document.getElementById("btn1"), "click", f1);
    };

    /*为元素绑定事件兼容性代码*/
    function addEventListener(element, type, fn) {
        if (element.addEventListener) {
            element.addEventListener(type, fn, false);
        } else if (element.attachEvent) {
            element.attachEvent("on" + type, fn);
        } else {
            element["on" + type] = fn;
        }
    }

    /*为元素解绑事件兼容性代码*/
    function removeEventListener(element, type, fnName) {
        if (element.removeEventListener) {
            element.removeEventListener(type, fnName, false);
        } else if (element.detachEvent) {
            element.detachEvent("on" + type, fnName);
        } else {
            element["on" + type] = null;
        }
    }
```

```
</script>
</body>
</html>
```

4.4.11、事件传播

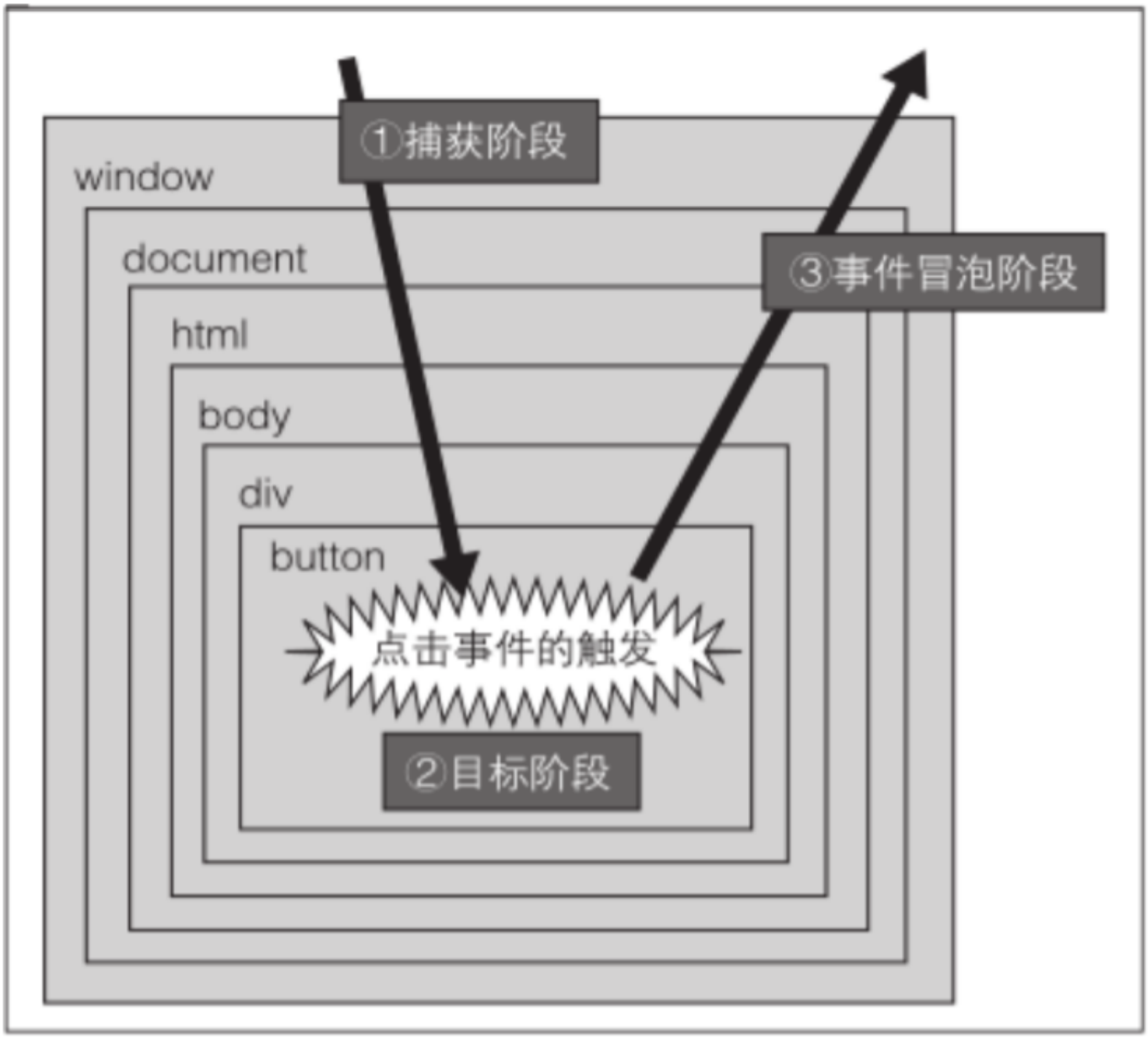
事件的传播：关于事件的传播网景公司和微软公司有不同的理解

微软公司认为事件应该是由内向外传播，也就是当事件触发时，应该先触发当前元素上的事件，然后再向当前元素的祖先元素上传播，也就说事件应该在冒泡阶段执行。

网景公司认为事件应该是由外向内传播的，也就是当前事件触发时，应该先触发当前元素的最外层的祖先元素的事件，然后在向内传播给后代元素。

W3C综合了两个公司的方案，将事件传播分成了三个阶段：

捕获阶段：在捕获阶段时从最外层的祖先元素，向目标元素进行事件的捕获，但是默认此时不会触发事件 目标阶段：事件捕获到目标元素，捕获结束开始在目标元素上触发事件 冒泡阶段：事件从目标元素向它的祖先元素传递，依次触发祖先元素上的事件



> 注意：如果希望在捕获阶段就触发事件，可以将addEventListener()的第三个参数设置为true，一般情况下我们不会希望在捕获阶段触发事件，所以这个参数一般都是false，并且注意，IE8及以下的浏览器中没有捕获阶段，我们可以使用event.stopPropagation();取消事件传播。