第 5 章

CHAPTER 5

JavaScript 交互编程

学习目标

- 了解 DOM 概念。
- 掌握 document 对象的使用。
- 掌握使用 DOM 查找节点的各种方法。
- 掌握使用 DOM 进行 HTML 元素属性控制、HTML 内容控制。
- 掌握使用 DOM 进行创建和操作 HTML 元素节点。
- 掌握使用 DOM 进行样式编程。
- 了解事件的概念,掌握常用的一些事件以及事件的监听方法。

DOM 操作与事件是 JavaScript 最核心的组成部分之一,它们赋予了页面无限的想象空间,在 HTML5 App 中就是依靠它们实现交互的。本章主要讲解在 HTML5 App 开发中必须掌握的一些 DOM 编程基础以及事件的使用,以便于实现高效和便捷的页面交互。

5.1 DOM 介绍

DOM(Document Object Model,文档对象模型)是 HTML 和 XML 的应用程序接口(API)。DOM 将整个页面规划成由节点层级构成的文档。例如下面这个 HTML 页面:

```
<!DOCTYPE html>
<html>
    <head>
    <meta charset="UTF-8">
    <title>测试页面</title>
</head>
<body>
    <p>Hello HTML5</p>
</body>
</html>
```

这段 HTML 代码可以用 DOM 绘制成一个节点层次图,如图 5-1 所示。

DOM 通过创建树来表示 HTML 文档,从而使开发者对文档的内容和结构具有很强的控制力,可以使用 DOM API 对这棵树的节点作各种变化:增加节点、删除节点、查找节点、修改节点等,DOM 技术还使得用户页面可以动态地变化,如可以动态地显示或隐藏一个元

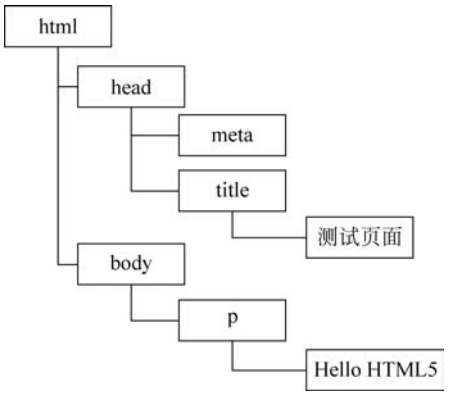


图 5-1　HTML 节点层次

素、改变它们的属性、增加一个元素等,大大地增强了页面的交互性。

5.2　使用 DOM

在 HTML5 App 开发的过程中,JavaScript 极为重要的一个功能就是 DOM 对象的操作,本节将讨论各种 DOM 操作,以便于实现高效率和便捷的页面交互。

5.2.1　document 对象

在浏览器引擎中,与用户进行数据交换都是通过客户端的 JavaScript 代码来实现的,而完成这些交互工作大多数是由 document 对象及其部件进行的,因此 document 对象是一个比较重要的对象。document 对象是文档的根节点,window. document 属性就指向这个对象。也就是说,只要浏览器开始载入 HTML 文档,这个对象就开始存在了,就可以直接调用。

表 5-1 列出了 document 对象的常用属性。

表 5-1　document 对象的常用属性

属　性	具体描述
alinkColor	表示激活链接(焦点在此链接上)的颜色
bgColor	表示页面背景色
body	表示< body >节点
charset	表示页面的字符集
doctype	表示文档类型节点,也就是<!DOCTYPE html >节点
documentElement	表示< html >节点
fgColor	表示前景色(文本颜色)
forms	表示页面上的所有 form 元素
head	表示< head >节点
images	表示页面上的所有 img 元素
lastModified	表示最终修改的日期
linkColor	表示未单击过的链接颜色

续表

属 性	具体描述
links	表示页面上的所有 a 元素
scripts	表示页面上的所有 script 元素
styleSheets	表示页面上的所有 link 或 style 元素
title	表示 title 的内容
URL	表示当前文档的 URL
vlinkColor	表示已单击过的链接颜色

【例 5-1】 document 对象的各属性使用示例,代码如下:

```
<script>
    document.alinkColor = "yellow";
    document.vlinkColor = "brown";
    document.bgColor = "bisque";
    document.fgColor = "crimson";
    //输出[object DocmentType]
    console.log(document.doctype);
    //输出[object HTMLHtmlElement]
    console.log(document.documentElement);
    //输出[object HTMLBodyElement]
    console.log(document.body);
    //输出[object HTMLHeadElement]
    console.log(document.head);
    //输出"document 对象"
    console.log(document.title);
    //输出"页面上有 x 个 script 标签"
    console.log("页面上有" + document.scripts.length + "个 script 标签");
    //输出"页面上 x 个超链接"
    console.log("页面上有" + document.links.length + "个超链接");
    //输出"页面上有 x 张图片"
    console.log("页面上有" + document.images.length + "张图片");
    //输出"页面上有 x 处样式"
    console.log("页面上有" + document.styleSheets.length + "处样式");
    //输出页面最后修改的日期
    console.log("页面修改日期:" + document.lastModified);
    //输出页面的 URL 地址
    console.log("页面地址:" + document.URL);
</script>
```

5.2.2 查找节点

在 HTML5 程序开发中,经常要修改某个 HTML 元素的样式、内容等,如何获取相应的元素,是首先要解决的问题,DOM 中提供了一些方法来方便快捷地访问指定的 HTML 元素节点,以下分别讲解。

1. getElementsByTagName 方法

这个方法用来返回一个页面上所有包含 tagName(标签名)等于某个指定值的元素节

点对象集合。当得到相应的节点集合以后,就可以使用方括号来访问其中某个子节点。

【例 5-2】 getElementsByTagName 使用示例,代码如下:

```
<body>
    <img src="../img/baidu.png" /><br />
    <input type="text" value="hello world"/><br />
    <input type="password" value="123456"/>
    <script>
        var oImg = document.getElementsByTagName("img");
        console.log(oImg[0].tagName);          //输出"IMG"
      oImg[0].src = "../img/163.png";
      var oInput = document.getElementsByTagName("input");
      console.log(oInput[0].value);            //输出"hello world"
      oInput[0].value = "Hello HTML5";
    </script>
</body>
```

在这个例子中,从节点对象中取得某个标签节点对象,意味着我们可以对标签相应的属性进行读取或设置。按照第 4 章讲解的调试断点方法,可以看到节点对象的属性和方法,如图 5-2 所示为 oImg[0]节点对象属性的部分截图。

图 5-2 节点对象属性的部分截图

在 HTML DOM 中,每一部分都是节点:

- 文档本身是文档节点;
- 所有 HTML 元素是元素节点;
- 所有 HTML 属性是属性节点;
- HTML 元素内的文本是文本节点;
- 注释是注释节点。

节点对象在 DOM 中定义为 Node 对象,Node 对象定义了一些属性和方法,表 5-2 中列出了这些属性和方法。

表 5-2 Node 对象的常用属性和方法

属性/方法	返回类型	具体描述
innerHTML	String	表示当前节点的内部标签
innerText	String	表示当前节点的文字内容
length	Number	返回 NodeList 中的节点数
nodeName	String	节点名称,根据节点的类型而定义
nodeValue	String	节点的值,根据节点的类型而定义
nodeType	Number	节点的类型常量值之一
firstChild	Node	指向在 childNodes 节点集合中的第一个节点
lastChild	Node	指向在 childNodes 节点集合中的最后一个节点
parentNode	Node	指向所在节点的父节点
childNodes	NodeList	所有子节点的集合
previousSibling	Node	指向前一个兄弟节点,如果当前节点本身就是第一个兄弟节点,则返回 null
nextSibling	Node	指向后一个兄弟节点,如果当前节点本身就是最后一个兄弟节点,则返回 null
hasChildNodes()	Boolean	是否包含一个或多个子节点
AppendChild(node)	Node	将 node 添加到 childNodes 的末尾
removeChild(node)	Node	从 childNodes 中删除 node
replaceChild(newnode,oldnode)	Node	将 childNodes 中 oldnode 替换成 newnode
insertBefore(newnode,refnode)	Node	在 childNodes 中在 refnode 之前插入 newnode
cloneNode(deep)	Node	deep 为 true 是深复制,复制当前节点以及子节点,为 false 是浅复制,只复制当前节点

2. getElementById 方法

当需要在 HTML 页面中查找一个特定的节点对象时,最有效的方法就是为该节点的标签元素添加一个 id 属性,并使用 getElementById 方法进行查找,这个方法会返回唯一的节点对象,例如通过下面的代码,可以迅速找到需要的 input 标签元素:

```
<input type = "text" value = "hello html5" id = "myTest"/>
<script>
    var oInput = document.getElementById("myTest");
    alert(oInput.value);
oInput:value = "hello world";
</script>
```

3. getElementsByClassName 方法

当需要在 HTML 页面中查找多个对象时,可以为这些对象指定相同的 class 属性,并使用 getElementsByClassName 方法进行查找,这个方法会返回所有 class 属性为指定值的节点对象集合。

【例 5-3】 getElementsByClassName 使用示例,代码如下:

```
<body>
    <div class = "example">第 1 个 div</div>
    <div>第 2 个 div</div>
```

```
    <p class = "example">第 1 个 p</p>
    <script>
        var elems = document.getElementsByClassName("example");
        alert(elems[1].tagName);   //输出"p"
    </script>
</body>
```

4. querySelectorAll 方法

作为查找 DOM 的又一途径,这个方法相当灵活,极大地方便了开发者。它可以接受一个 CSS 选择器参数,调用后可以返回 HTML5 页面中所有匹配 CSS 选择器的元素节点对象集合,目前所有的主流浏览器都支持这一方法。

5. querySelector 方法

和方法 querySelectorAll 完全类似,也是使用 CSS 选择器查找节点,不同的是,这个方法只返回匹配选择器的第 1 个元素节点对象,而 querySelectorAll 返回的是所有匹配的元素节点对象集合。

【例 5-4】 querySelectorAll 和 querySelector 使用示例,代码如下:

```
<body>
    <div id = "test">
        我是 id 为 test 的 div
    </div>
    <div class = "mytest">
        <p>我是 div 里的 p 标签</p>
    </div>
    <script>
        var oDiv1 = document.querySelector("#test");
        alert(oDiv1.innerText);          //输出"我是 id 为 test 的 div"
        var oDiv2 = document.querySelectorAll("#test");
        alert(oDiv2[0].innerText);       //输出"我是 id 为 test 的 div"
        var oP1 = document.querySelector("div.mytest > p");
        alert(oP1.innerText);            //输出"我是 div 里的 p 标签"
        var oP2 = document.querySelectorAll("div.mytest > p");
        alert(oP2[0].innerText);         //输出"我是 div 里的 p 标签"
    </script>
</body>
```

查找 HTML 元素节点时,应确保它已在 DOM 树上构建,否则会出现找不到的情况。

例如下面这种情况,就未能成功查找到按钮对象,必须把 JavaScript 代码放在 HTML 代码之后:

```
<script>
    var oInput = document.getElementById("myButton");
    alert(oInput);          //输出 null
```

```
</script>
<input type="button" value="测试" id="myButton"/>
```

为了提高程序性能，一定要避免重复的DOM查找，例如下面这段代码就会造成性能问题：

```
<script>
    for(var i=0;i<10;i++){
        var oDiv=document.getElementById("mydiv");
        …
    }
</script>
```

正确的方式是在循环体之外声明一个变量用来存储查找出来的HTML元素节点对象，修改后的代码如下：

```
<script>
    var oDiv=document.getElementById("mydiv");
    for(var i=0;i<10;i++)
    {
        …
    }
</script>
```

5.2.3 处理属性

对于HTML元素节点，DOM提供了3种方法来处理其属性，这些方法相当有用：

- getAttribute(name)：获取某个属性的值；
- setAttribute(name,newvalue)：设置某个属性的值；
- reomveAttribute(name)：移除某个属性。

【例5-5】 DOM控制HTML元素属性示例，代码如下：

```
<body>
    <input type="button" value="测试" id="mybutton"
data-test="1234"/>
    <script>
        var oInput=document.getElementById("mybutton");
        alert(oInput.getAttribute("type"));     //输出"button"
        oInput.setAttribute("type","text");     //将按钮切换成输入框
        //设置自定义属性
        oInput.setAttribute("data-myattr","just a test");
        //读取自定义属性,输出"just a test"
        alert(oInput.getAttribute("data-myattr"));
        oInput.removeAttribute("id");           //输出"id"属性
```

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

这里介绍自定义属性,HTML5 标准中规定自定义属性需要添加前缀 data-,目的是提供与页面渲染无关的信息,运行这个例子后会发现,input 标签添加了 data-test 属性后,对于界面的显示并无任何影响。使用自定义属性可以很方便地存储页面或应用程序的私有自定义数据,目前所有主流浏览器都支持 data-* 属性。

HTML 标签元素中只有标准属性才会以属性的形式添加到 DOM 对象中,DOM 对象只能访问这些标准属性,而 getAttribute 方法可以访问所有属性。

5.2.4 读取和设置内容

在 HTML5 App 开发中,经常涉及读取或设置 HTML 标签元素内部所包含的文字或子 HTML 标签,DOM 提供了以下属性以供使用。

1. innerText 属性

DOM 中通过 innerText 属性可以操作元素中包含的所有文本内容,无论文本位于子文档树中的什么位置。在通过 innerText 读取值时,它会按照由浅入深的顺序,将子文档树中所有文本拼接起来。以下面的 HTML 代码为例。

【例 5-6】 innerText 属性使用示例,代码如下:

```
<body>
    <div id="content1">
        <p>This is a <strong>paragraph</strong>
        with a list following it.</p>
        <ul>
            <li>item1</li>
            <li>item2</li>
            <li>item3</li>
            <li>item4</li>
        </ul>
    </div>
    <script>
        var oDiv = document.getElementById("content");
        alert(oDiv.innerText);
        oDiv.innerText = "hello html5";
    </script>
</body>
```

对于这个例子中的 div 而言,其 innerText 会返回下列字符串:

```
This is a paragraph with a list following it.

item1
item2
item3
item4
```

设置 div 的 innerText，它的内容变成了：

```
<div id = "content"> hello html5 </div>
```

使用 innerText 或 innerHTML 为 HTML 元素对象设置内容时，会先将对象开始标签和结束标签之间的内容全清空。

2. innerHTML 属性

几乎所有的 DOM 对象都有 innerHTML 属性，它是一个字符串，用来设置或获取位于 HTML 标签对象起始和结束标签内的 HTML 代码。

【例 5-7】 innerHTML 属性使用示例，代码如下：

```
    <body>
        <div id = content2 >
            <p> This is a <strong> paragraph </strong> with a list following it. </p>
            <ul>
                <li> item1 </li>
                <li> item2 </li>
                <li> item3 </li>
                <li> item4 </li>
            </ul>
        </div>
        <script>
            var oDiv = document.getElementById("content");
            alert(oDiv.innerHTML);
            oDiv.innerHTML = '<img src = "../img/baidu.png" />';
        </script>
    </body>
</html>
```

对于这个例子中的 div 而言，其 innerHTML 会返回下列 HTML 字符串：

```
<p> This is a <strong> paragraph </strong> with a list following it. </p>
<ul>
    <li> item1 </li>
    <li> item2 </li>
    <li> item3 </li>
    <li> item4 </li>
</ul>
```

设置 div 的 innerHTML 后，它的内容会变成一张图片显示：

```
<div id = "content">
    <img src = "../img/baidu.png" />
</div>
```

DOM操作结束后，如果单击鼠标右键，单击“查看网页源代码”命令后，会发现源代码没有任何变化。要查看变化后的标签，必须使用Chrome浏览器的“开发者工具”进行查看，这是用HTML5 App开发调试必须掌握的技能。

5.2.5 操作节点

前面已经介绍了如何利用DOM查找HTML元素节点，不过这只是DOM所能实现功能的一小部分，DOM还可以添加、删除、替换(或其他操作)节点。正是这些功能才使得DOM具有真正意义上的动态性和交互性。

1. 创建新节点

对于一个好的页面来说，为了让用户体验做到极致，动态创建页面节点是必不可少的。DOM中有一些方法可以用于创建不同类型的节点，最常用到的几个方法见表5-3。

表5-3 创建节点的常用方法

方 法	具体描述
createTextNode(text)	创建包含文本text的文本节点
createElement(tagName)	创建标签为tagName的HTML元素节点
createDocumentFragment()	创建文档碎片节点

2. 追加节点appendChild

appendChild用于向一父节点的尾部追加子节点，下面我们用一个具体的例子来学习它的使用。

【例5-8】 动态创建和追加节点使用示例，例如有个页面显示如下：

```
<body>
    <div id="container"></div>
</body>
```

现在想使用DOM操作来添加以下HTML代码到页面的容器中：

```
<div id="mydiv">
    <p>HELLO HTML5</p>
</div>
```

这里可以首先使用createElement方法和createTextNode方法来实现节点对象的创建，再使用appendChild方法追加到容器中，实现步骤如下。

① 首先，创建div元素，并设置其id属性值为“mydiv”：

```
var oDiv = document.createElement("div");
oDiv.setAttribute("id", "mydiv");
```

② 第二步，创建 p 元素：

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

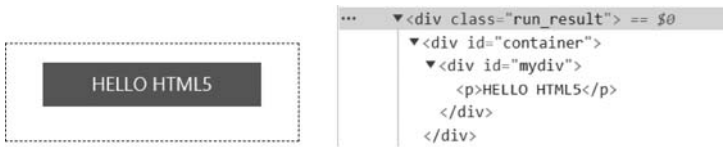
③ 第三步，创建文本节点：

```
var oText = document.createTextNode("HELLO HTML5");
```

④ 最后，使用 appendChild 方法依次把各子节点添加到相应节点的尾部：

```
oP.appendChild(oText);
oDiv.appendChild(oP);
document.getElementById("container").appendChild(oDiv);
```

运行后的界面和页面的 HTML 动态变化如图 5-3 所示。



(a) 界面　　(b) 动态变化

图 5-3　动态创建节点效果

所有的 DOM 操作必须在页面完全载入后才能进行。当页面正在载入时，要向 DOM 插入相关节点是不可能的，因为 DOM 树还没有构建完成。

3. 移除节点 removeChild

既然可以添加节点，当然也可以删除节点，这就是 removeChild 方法所能完成的。这个方法接受一个参数，代表要删除的节点对象，返回值也是这个节点对象，删除时要尽量使用节点的 parentNode 特性来确保能访问到它真正的父节点。如图 5-4 所示的例子中，需要移除其中红色的 div 节点。

图 5-4　节点移除前后效果

【例 5-9】 动态移除节点使用示例，代码如下：

```
<body>
    <div id="parent">
```

```
        <div id = "child">
            HELLO HTML5
        </div>
    </div>
    <script>
        var oDiv = document.getElementById("child");
        oDiv.parentNode.removeChild(oDiv);
    </script>
</body>
```

这个页面加载后,在看到它之前,红色的 div 节点已被自动移除。

4. 替换节点 replaceChild

如果想将节点替换成新的节点,则需要使用 replaceChild 方法。replaceChild 方法有两个参数——被添加的节点对象和被替换的节点对象。例如图 5-5 中,需要将第一个节点替换。

【例 5-10】 动态替换节点使用示例,代码如下:

```
<body>
    <ul class = "dataList1">
        <li>XHTML</li>
        <li>CSS</li>
        <li>JavaScript</li>
    </ul>
    <script>
        var oli = document.querySelector(".dataList1 li:first-child");
        var nli = document.createElement("li");
        nli.innerText = "HTML5";
        oli.parentNode.replaceChild(nli,oli);
    </script>
</body>
```

5. 节点前插入 insertBefore

当向页面中添加节点时,如果想让新节点出现在某个节点之前,可使用 insertBefore 方法。这个方法接受两个参数——要添加的节点对象和目标节点对象。例如图 5-6 中,需要在列表项 CSS 之前插入列表项 Photoshop。

图 5-5 节点替换前后效果

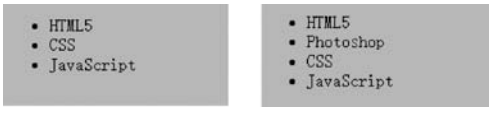


图 5-6 节点插入前后效果

【例 5-11】 动态插入节点使用示例,代码如下:

```
<body>
        <ul class = "dataList2">
            <li>Android</li>
```

```
            <li>iOS</li>
            <li>HTML5</li>
        </ul>
        <script>
          var oli = document.querySelector(".dataList2 li:nth-of-type(2)");
          var nli = document.createElement("li");
          nli.innerText = "photoshop";
          oli.parentNode.insertBefore(nli,oli);
        </script>
    </body>
```

6. 创建文档碎片 createDocumentFragment

在 HTML5 App 开发中，经常会遇到根据服务器返回的数据，在某个节点处生成多个列表项的场景，通常这部分可以根据数据的条数，使用 createElement 循环生成对应的节点对象后，附加到相应的父节点，但 DOM 修改会导致页面重绘、重新排版。重新排版会阻塞用户的操作，同时，如果频繁重排，CPU 使用率也会猛涨，App 的性能会受到严重影响。所以，为了得到更高的性能，一般使用 createDocumentFragment 创建文档碎片，把所有的新节点附加其上，然后把文档碎片一次性添加到指定的节点上。

【例 5-12】 createDocumentFragment 使用示例，向一个 ul 添加 100 条列表项，代码如下：

```
<body>
    <ul class="dataList3">
    </ul>
    <script>
        var dList = document.querySelector(".dataList3");
        var fragment = document.createDocumentFragment();
        for(var i = 0;i<100;i++)
        {
            var oLi = document.createElement("li");
            oLi.innerText = "Item" + (i + 1);
            fragment.appendChild(oLi);
        }
        dList.appendChild(fragment);
    </script>
</body>
```

7. 拷贝节点 cloneNode

cloneNode 这个方法主要用来实现对节点对象的复制，并返回复制的节点对象。它有一个输入参数，类型是 Boolean，默认为 false，表示浅复制，如果是 true，表示深复制。

【例 5-13】 cloneNode 使用示例，代码如下：

```
    <body>
        <div id="content">
            <ul class="mylist">
```

```
            <li>HTML5</li>
            <li>CSS3</li>
            <li>JavaScript</li>
        </ul>
    </div>
    <script>
        var oDiv = document.getElementById("content");
        var oUl = document.getElementsByClassName("mylist")[0];
        //实现深复制
        oDiv.appendChild(oUl.cloneNode(true));
        //实现浅复制
        oDiv.appendChild(oUl.cloneNode(false));
    </script>
</body>
```

页面运行后,显示的效果如图 5-7 所示,可以明显看出对页面上的列表进行深复制和浅复制的区别:浅复制,只复制当前节点对象;深复制,复制包括当前节点对象和它下面的所有子节点对象。

图 5-7 cloneNode 的深复制和浅复制

5.3 DOM 的样式编程

DOM 样式编程,就是通过 JavaScript 来操作页面 HTML 元素的样式,实现页面的特定显示效果,例如页面的隐藏显示效果、旋转效果等。

5.3.1 className 属性

className 属性可以用来读取或设置 HTML 元素对象的 class 属性,标准的 DOM 属性不包含 class 属性,因为 class 是 JavaScript 的保留关键字。将 className 属性设置为空字符串时,代表移除所有的样式。

【例 5-14】 className 使用示例,代码如下:

```
<body>
    <div id = "mydiv1" class = "mystytle">
    </div>
    <span id = "desc"></span>
    <script>
        var oDiv = document.getElementById("mydiv1");
        var oSpan = document.getElementById("desc");
```

```
            oSpan.innerText = "div 的样式为:" + oDiv.className;
            setTimeout(function(){
                oDiv.className = "mynewstyle";
                oSpan.innerText = "div 的样式为:" + oDiv.className;
                setTimeout(function(){
                    oDiv.className = "";
                    oSpan.innerText = "移除 div 的所有样式";
                },2000);
            },2000);
        </script>
    </body>
```

程序运行后,先在页面上显示一个黄色的 div,2s 后自动变成一个带边框的淡蓝色 div,再过 2s 颜色和边框自动消失,效果如图 5-8 所示。

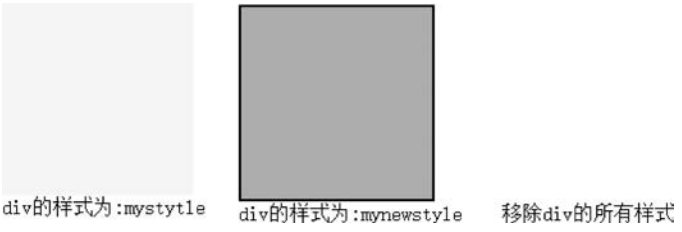


图 5-8 className 修改样式和移除样式

5.3.2 classList 对象

className 属性使用比较简单,但如果 HTML 元素对象的 class 属性值中有多个样式类应用时,对其样式分别进行控制就不太方便,例如:

```
<div id = "mydiv" class = "style1 style2 style3"></div>
```

为了解决这个问题,在 HTML5 API 里,页面 DOM 里的每个节点上都有一个 classList 对象,提供了如表 5-4 所示的方法新增、删除、修改节点上的样式类。

表 5-4 classList 对象方法和属性

方 法	具 体 描 述
length	返回 HTML 元素对象 class 属性中样式类的个数
add(className)	给 HTML 元素对象的 class 属性添加一个样式类
remove(className)	从 HTML 元素对象的 class 属性中删除一个指定的样式类
toggle(className)	若 HTML 元素对象的 class 属性有指定的样式类,则执行 remove 操作,若没有则执行 add 操作
contains(className)	检测 HTML 元素对象的 class 属性中是否包含指定的样式类

【例 5-15】 classList 对象使用示例,代码如下:

```
<body>
    <div id="mydiv2" class="mystyle1">
    </div>
    <span id="desc1"></span><br />
    <span id="desc2"></span>
    <script>
        var oDiv = document.getElementById("mydiv2");
        var oSpan1 = document.getElementById("desc1");
        var oSpan2 = document.getElementById("desc2");
        //获取样式类个数
        oSpan1.innerText = "div 的样式类数目:" + oDiv.classList.length;
        //检测是否包含 mystyle1 样式
        oSpan2.innerText = "包含 style1 样式类:"
                 + oDiv.classList.contains("mystyle1");
        setTimeout(function(){
                //移除样式 mystyle1
                oDiv.classList.remove("mystyle1");
                //添加样式 mystyle2
                oDiv.classList.add("mystyle2");
                setInterval(function(){
                    //切换样式 mystyle2,有则去掉,无则加上
                    oDiv.classList.toggle("mystyle2");
                },500);
        },1000);
    </script>
</body>
```

代码运行后,先在页面上显示一个黄色的带边框的矩形 div,1s 后自动去除黄色背景,然后每隔 0.5s 加上或去除边框,效果如图 5-9 所示。

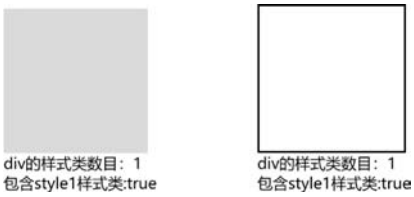


图 5-9　classList 对象动态添加、移除样式

className 属性和 classList 对象在修改样式时,都是对 HTML 元素对象的 class 属性进行修改,可以使用 Chrome 浏览器中的“开发者工具”进行查看,并将其作为界面调试的重要手段。

5.3.3　style 对象

使用 className 和 classList 对象已经可以很方便地修改 HTML 元素对象的样式,但这种方式只适合于样式的值是固定不变的,如果样式的值需要引入变量形式,则必须借助于每个 HTML 元素对象的 style 对象,用它来访问 HTML 元素对象的样式信息。

style 对象包含了与每个 CSS 样式对应的属性,只是格式略有不同:

(1) 对于单个单词的CSS样式,以相同的名字属性来表示(例如,color样式通过style.color表示)。

(2) 对于两个单词的CSS样式,则是通过去除横杠,将第一个单词加上首字母大写的第二个单词(例如:background-color样式对应style.backgroundColor)。表5-5列出了一些常用的CSS样式以及它们对应的JavaScript中style对象的属性表示。

表5-5 CSS样式与JavaScript中style对象属性的对应示例

CSS样式	style中的属性
background-color	style.backgorundColor
color	style.color
font	style.font
font-family	style.fontFamily
font-weight	style.fontWeight

使用style对象可以方便地获取HTML元素对象的style属性所定义的CSS样式值,但它无法获取在外部定义的CSS样式。

【例5-16】 style对象修改样式使用示例,代码如下:

```
    <body>
        <div id="mydiv3" style="background-color: red;">
            Hello HTML5
        </div>
        <span id="desc3"></span><br />
        <span id="desc4"></span>
        <script>
            var oDiv = document.getElementById("mydiv3");
            var oSpan1 = document.getElementById("desc3");
            var oSpan2 = document.getElementById("desc4");
            oSpan1.innerText = "div 的背景色为"
                    + oDiv.style.backgroundColor;
            oSpan2.innerText = "div 中的字体颜色为:"
                    + oDiv.style.color;   //color 是在外部定义的,无法读取
            var w = 300;
            setTimeout(function(){
                oDiv.style.width = w + "px";
                oDiv.style.height = w + "px";
                oDiv.style.backgroundColor = "yellow";
                oDiv.style.color = "#000";
                oDiv.style.fontSize = "40px";
                oSpan1.innerText = "div 的背景色为"
                        + oDiv.style.backgroundColor;
            },5000);
        </script>
    </body>
</html>
```

程序运行后,页面上出现一个红色的div,里面含有白色的文字,5s后使用style对象修改div的大小和背景色,内部文字的大小和颜色也作了修改,如图5-10所示。

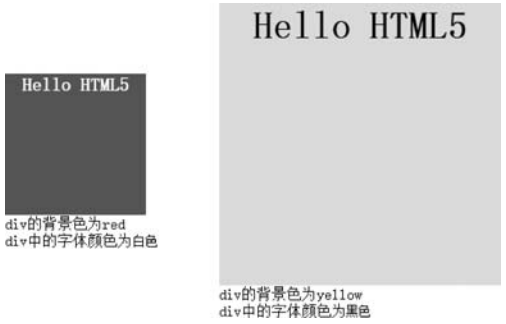


图 5-10　style 对象读取和设置样式

使用 style 对象设置样式时，实际就是修改 HTML 元素对象的 style 属性，同样地，对它读取时也只能读取到 style 属性中的定义，如果要取到最终的 CSS 样式值，可以使用类似 window.getComputedStyle(odiv).backgroundColor 这样的代码。

5.4　事件

JavaScript 是基于对象(object-based)的语言，基于对象的基本特征，就是采用事件驱动(event drive)。事件是在浏览器中可以被 JavaScript 检测到的行为，例如用户单击了按钮、移动了手指，都会触发相应的事件。用来响应某个事件的函数则称为事件处理程序，或者称为事件监听函数。

5.4.1　常用的一些事件

在页面浏览过程中，由鼠标或键盘、触摸屏会驱动一系列事件的激发，表 5-6 列出了一些常用的事件。

表 5-6　常用事件举例

归类	事 件 名 称	描　　述
键盘	onkeydown	某个键盘的键被按下触发
	onkeypress	某个键盘的键被按住触发
	onkeyup	某个键盘的键被松开触发
鼠标	onmousedown	鼠标键按下触发
	onmousemove	鼠标被移动触发
	onmouseout	鼠标从某个 HTML 元素移开触发
	onmouseover	鼠标移动到某个 HTML 元素上触发
	onmouseup	松开鼠标键触发
	onclick	鼠标单击某个 HTML 元素触发
	ondbclick	鼠标双击某个 HTML 元素触发
触摸	ontouchstart	当手指触摸屏幕时候触发，即使已经有一个手指放在屏幕上也会触发
	ontouchmove	当手指在屏幕上滑动的时候连续地触发
	ontouchend	当手指从屏幕上离开的时候触发
	ontouchcancel	当系统停止跟踪触摸的时候触发(例如有电话或短信时)

续表

归类	事件名称	描　述
其他	onabort	图像加载中断后触发
	onblur	HTML 元素失去焦点时触发
	onchange	内容发生改变时触发
	onerror	当加载文档或图像时发生错误时触发
	onfocus	HTML 元素获得焦点时触发
	onload	加载页面或图像时触发
	onresize	窗口调整尺寸时触发
	onunload	退出页面时触发

5.4.2 内联属性监听事件

这种方法是指在 HTML 元素里面直接填写事件有关属性，属性值为相关 JavaScript 代码，即可在触发该事件的时候，执行属性值里面的代码。

【例 5-17】 使用 HTML 元素内联属性监听事件使用示例，代码如下：

```
<body>
    <input type="button" value="运行 JS 语句"
onclick="alert('hello');"/>
    <input type="button" value="运行 JS 函数"
onclick="eventTest();"/>
    <script>
        function eventTest(){
            alert("函数运行");
        }
    </script>
</body>
```

在这个例子中，当单击按钮时，就会触发 click 事件，执行 onclick 属性中的 JavaScript 语句。显而易见，使用这种方法时，JavaScript 代码与 HTML 代码耦合在了一起，不便于维护和开发，所以要尽量避免使用这种方法。

5.4.3 DOM 属性监听事件

使用 DOM 属性绑定事件可以解决代码的耦合问题，使用也比较简单，只需要在相应的节点对象上直接使用表 5-6 中的事件名称作为属性，赋予匿名函数或函数名即可。它比较简单易懂，而且有较好的兼容性。但是也有缺陷：因为直接赋值给对应事件属性，如果在后面代码中再次为节点对象绑定一个回调函数，会覆盖之前回调函数的内容。

【例 5-18】 使用 DOM 属性监听事件使用示例，代码如下：

```
<body>
    <input type="button" value="使用匿名函数" id="mybutton1"/>
    <input type="button" value="使用函数名" id="mybutton2"/>
```

```
    <script>
        var oInput1 = document.getElementById("mybutton1");
        var oInput2 = document.getElementById("mybutton2");
        oInput1.onclick = function(){
            alert("hello world");
        }
        //再次使用 DOM 属性监听,覆盖以前的监听事件
        oInput1.onclick = function(){
            alert("hello html5");
        }
        oInput2.onclick = sayHello;
        function sayHello(){
            alert("hello");
        }
    </script>
</body>
```

使用 DOM 属性监听事件时，最容易犯错的地方是赋予的值不是函数名，而是执行函数，造成无法触发相应的事件，例如：

```
oInput2.onclick = sayHello(); //错误,sayHello 返回值是 undefined
```

5.4.4 标准的事件监听函数

在 HTML5 App 开发中,一般都是使用标准的事件监听函数来实现对某个事件的触发,它的语法格式为：

```
addEventListener(eventName,callback,usecaptuer);
```

事件监听函数同样是对于 HTML 元素对象使用,当监听到有相应事件发生的时候,调用 callback 回调函数。至于 usecapture 这个参数,表示该事件监听是在“捕获”阶段中监听(设置为 true)还是在“冒泡”阶段中监听(设置为 false)。usecapture 一般设置为 false。

标准的事件监听函数和前面的内联方式以及 DOM 方式都不同,如果对同一个 HTML 元素多次添加回调函数,这些回调函数会按先后次序依次执行。

如果想解除监听函数绑定,则需要使用 removeEventListener 方法：

```
removeEventListener (eventName,callback,usecapture);
```

需要注意的是,如果要移除监听函数绑定,绑定事件时的回调函数不能是匿名函数,必须是一个声明的函数,因为解除事件绑定时需要传递这个回调函数的引用(即函数名),才可以解开绑定。

【例 5-19】 使用标准的监听函数使用示例，代码如下：

```
<body>
    <div id="mydiv">
    </div>
    <span id="desc"></span><br /><br />
    <input type="button" value="实现绑定" id="btnAdd"/>
    <input type="button" value="解除绑定" id="btnRemove"/>
    <script>
        var oInput1 = document.getElementById("btnAdd");
        var oInput2 = document.getElementById("btnRemove");
        var oDiv = document.getElementById("mydiv");
        var oSpan = document.getElementById("desc");
        //对 btnAdd 添加 click 事件监听 1
        oInput1.addEventListener("click",function(){
            //对 div 添加 mouseover 和 mouseout 事件监听
            oDiv.addEventListener("mouseover",changeBkColor,false);
            oDiv.addEventListener("mouseout",changeBkColor,false);
        },false);
        //对 btnAdd 添加 click 事件监听 2
        oInput1.addEventListener("click",function(){
            desc.innerText = "绑定成功,请将鼠标放在 div 上或移出 div";
        },false);
        //对 btnRemove 添加 click 事件监听 1
        oInput2.addEventListener("click",function(){
            //对 div 移除 mouseover 和 mouseout 事件监听
    oDiv.removeEventListener("mouseover",changeBkColor,false);
    oDiv.removeEventListener("mouseout",changeBkColor,false);
        },false);
        //对 btnRemove 添加 click 事件监听 2
        oInput2.addEventListener("click",function(){
            desc.innerText = "解除绑定";
        },false);
        //对 div 的样式类 newBk 实现 toggle
        function changeBkColor()
        {
            oDiv.classList.toggle("newBk");
        }
    </script>
</body>
```

在这个例子中，单击“实现绑定”按钮后，鼠标放在黄色的 div 上，div 会自动变成红色，鼠标移出 div，会恢复为黄色；当单击“解除绑定”按钮后，鼠标移动或移出 div 不再有效果出现，如图 5-11 所示。

图 5-11　标准的事件监听函数

5.4.5　事件触发过程

前文大体介绍了事件是什么、如何监听并执行某些操作，但还未涉及事件触发的整个过程，下面来讨论事件的触发过程，它分为捕获阶段、目标阶段、冒泡阶段，如图 5-12 所示。

图 5-12　事件触发过程

1. 捕获阶段

当 DOM 树的某个节点发生了一些操作(例如单击、鼠标移动上去)，就会有一个事件被触发。这个事件从 Window 发出，不断经过下级节点直到目标节点。在到达目标节点之前的过程，就是捕获阶段(Capture Phase)。

所有经过的节点，都会触发这个事件。捕获阶段的任务就是建立这个事件传递路线，以便后面冒泡阶段顺着这条路线返回 Window。

监听某个在捕获阶段触发的事件，需要在将标准的事件监听函数的参数 usecapture 设置为 true。

2. 目标阶段

当事件流到达事件触发目标节点那里，最终在目标节点上触发这个事件，这就是目标阶段(Target Phase)。需要注意的是，事件触发的目标总是最底层的节点。例如单击表格中的一段文字，以为事件目标节点在< td >上，但实际上触发在它的文字子节点上。

3. 冒泡阶段

当事件达到目标节点之后，就会沿着原路返回，由于这个过程类似水泡从底部浮到顶部，所以称作冒泡阶段(Bubbling Phase)。在实际使用中，并不需要把事件监听函数准确绑定到最底层的节点也可以正常工作。例如为< td >绑定单击时的回调函数时，无须为它下面的所有子节点全部绑定单击事件，只需要为< td >这一个节点绑定即可。因为发生它子节点的单击事件，都会冒泡上去，发生在< td >上。

所以在使用标准的事件监听函数时，usecapture 参数一般设为 false，这样监听事件时只会监听冒泡阶段发生的事件。

因为事件有冒泡机制，所有子节点的事件都会顺着父级节点传递回去，所以可以通过监听父级节点来实现监听子节点的功能，这就是事件代理。使用事件代理主要有两个优势：

(1) 减少事件绑定，提升性能。以前需要绑定一堆子节点，而现在只需要绑定一个父节点即可，减少了绑定事件监听函数的数量。

(2) 动态变化的 DOM 结构，仍然可以监听。当一个 DOM 动态创建之后，不会带有任何事件监听，除非重新执行事件监听函数，而使用事件监听无须担忧这个问题。

5.4.6 事件的 Event 对象

当一个事件被触发的时候，会创建一个事件对象(Event Object)，这个对象里面包含了一些有用的属性或者方法。事件对象会作为第一个参数，传递给回调函数。事件对象包含了很多有用的信息，例如事件触发时，鼠标在屏幕上的坐标、被触发的 DOM 详细信息等，表 5-7 列出了一些在 HTML5 App 中常用的事件对象属性和方法。

表 5-7 Event 对象常用的一些属性和方法

属性/方法	类型	可读/可写	描 述
cancelable	Boolean	只读	表示事件能否取消
cancelBubble	Boolean	只读	表示事件冒泡是否取消
clientX	Number	只读	事件发生时，鼠标或触摸点在客户端区域(不包含工具栏、滚动条等)的 x 坐标
clientY	Number	只读	事件发生时，鼠标或触摸点在客户端区域(不包含工具栏、滚动条等)的 y 坐标
currentTarget	Object	只读	触发事件的 HTML 元素对象
eventPhase	Number	只读	事件所处阶段，0—捕获阶段，1—目标阶段，2—冒泡阶段
pageX	Number	只读	鼠标或触摸点相对于页面的 x 坐标
pageY	Number	只读	鼠标或触摸点相对于页面的 y 坐标
preventDefault()	Function	只读	阻止事件的默认行为
screenX	Number	只读	相对于屏幕的鼠标 x 坐标
screenY	Number	只读	相对于屏幕的鼠标 y 坐标
stopPropagation()	Function	只读	阻止事件冒泡

续表

属性/方法	类型	可读/可写	描　　述
target	Object	只读	引起事件的 HTML 元素对象
type	String	只读	触发的事件类型
isTrusted	Boolean	只读	事件是浏览器触发(用户真实操作触发),还是 JavaScript 代码触发的

在这个表格中,我们看到有很多相关的坐标属性,这些坐标很容易混淆,图 5-13 示意了各属性的具体含义,可以进行相应的参考。

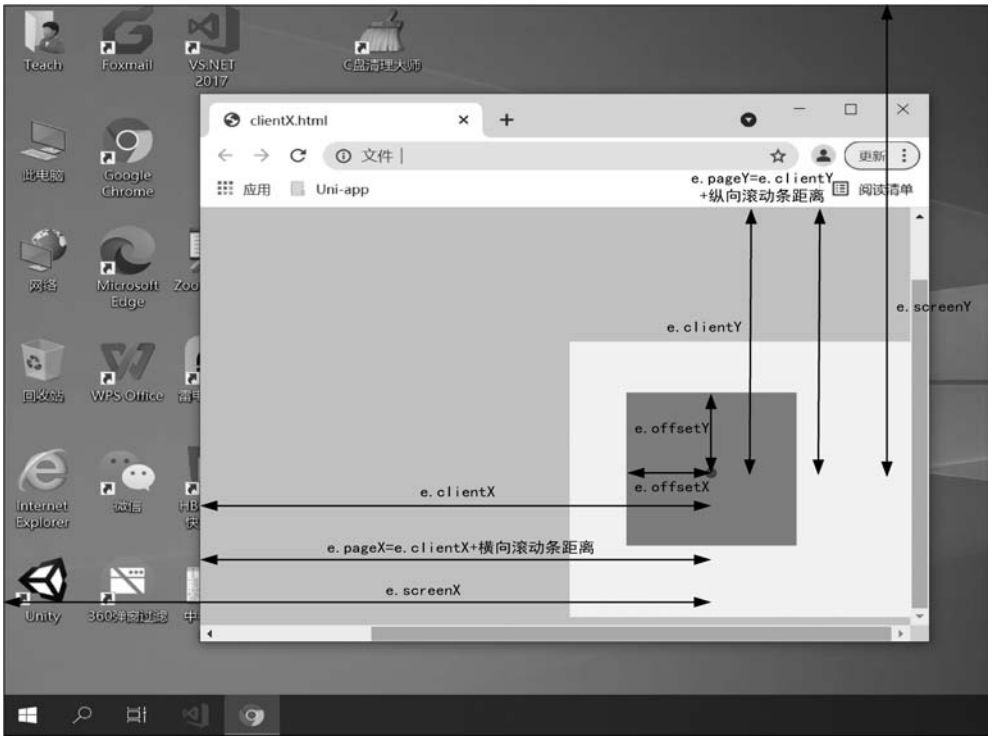


图 5-13　Event 对象中各坐标示意

【例 5-20】 事件代理和 Event 对象使用示例,代码如下:

```
<body>
    <ul id="data_list">
    </ul>
    <button id="btnAdd">添加项目</button>
    <script>
        var obtn = document.getElementById("btnAdd");
        var list = document.getElementById("data_list");
        var count = 0;
        //每单击一次按钮,增加 1 个项目
        obtn.addEventListener("click",function(){
            var oli = document.createElement("li");
```

```
            oli.innerText = "Item" + (++count);
            list.appendChild(oli);
        },false);
        //事件代理
        list.addEventListener("click",function(e){
            if(e.target.nodeName.toLowerCase() == "li"){
                alert(e.target.innerText);
            }
        },false);
    </script>
</body>
```

运行代码后，效果如图 5-14 所示，每单击 1 次按钮，就会自动增加一条项目，使用事件代理，我们将 onclick 事件附加到了其父容器 ul 上，这样就避免了添加新的 li 后必须在 li 上重新添加事件监听的麻烦，这里借助 click 事件的 Event 对象，很轻松地“拿”到了相应的 li 对象。

图 5-14 事件代理和 Event 对象效果图

【例 5-21】 触摸事件和 Event 对象使用示例，代码如下：

```
<body>
    <div id = "showDiv"></div>
    <div id = "objDiv"></div>
    <script>
        //开始触摸时的 x 和 y 坐标
        var sx;
        var sy;
        var oDiv = document.getElementById("showDiv");
        var obj2 = document.getElementById("objDiv");
        function touches(ev) {
          switch(ev.type) {
            case 'touchstart':
                //阻止出现滚动条
                ev.preventDefault();
                oDiv.innerHTML = 'Touch start';
                sx = ev.touches[0].clientX - obj2.offsetLeft;
                sy = ev.touches[0].clientY - obj2.offsetTop;
                break;
            case 'touchmove':
                oDiv.innerHTML = 'Touch move(' +
                          ev.changedTouches[0].clientX + ', ' +
                          ev.changedTouches[0].clientY + ')';
                //阻止出现滚动条
                ev.preventDefault();
                var mx = ev.changedTouches[0].clientX - sx;
```

```
                var my = ev.changedTouches[0].clientY - sy;
                obj2.style.marginLeft = mx - 200 + "px";
                obj2.style.marginTop = my - 200 + "px";
                break;
            case 'touchend':
                oDiv.innerHTML = 'Touch end';
                break;
                }
        }
        window.addEventListener("load", function() {
            document.addEventListener('touchstart', touches, false);
            obj2.addEventListener('touchmove', touches, false);
            document.addEventListener('touchend', touches, false);
        }, false);
    </script>
</body>
```

代码运行后,在 Chrome 中打开“开发者工具”,把它切换成“移动设备模式”,鼠标自动模拟手指触点,按下鼠标左键,单击页面,页面显示触摸的 touchstart 事件被触发,按住鼠标左键对红色的 div 实现拖放操作,触发 touchmove 事件,当松开鼠标左键后,将自动触发 touchend 事件,效果如图 5-15 所示。

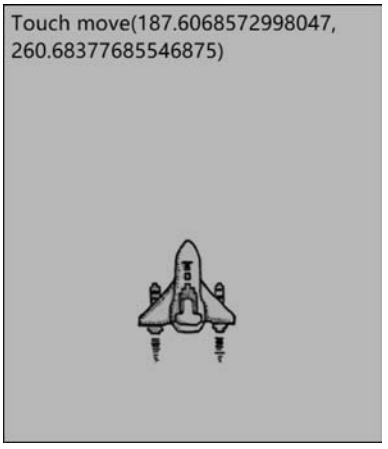


图 5-15　触摸事件和 Event 对象

5.5　实战演练:表格 DOM 操作

【例 5-22】 如图 5-16 所示,在这个例子中,表中的“平均分”这列是使用 DOM 动态生成的列,另外平均成绩≥87 分的同学中,若是党员的女同学全用黄色实现标注,这是 DOM 样式的动态附加,数据可以单行删除。也可以实现批量删除。最后还示范了目前项目开发中常见的 checkbox 的全选和取消全选功能。请用手机扫描对应二维码,结合本书的配套源代码,参看本例的讲解。

图 5-16 表格 DOM 操作效果

小结

本章主要讲解 JavaScript DOM 和事件的编程基础，介绍 DOM 概念、document 对象的使用，详细讲解 DOM 查找节点的各种方法，使用 DOM 进行属性操作、修改节点内部 HTML 和文字内容，创建节点、添加、插入、删除、替换、复制节点的各种方法，介绍了事件和一些常用事件以及如何实现事件监听。本章所讲解的内容主要应用在 HTML5 App 交互编程中，需要重点掌握。

习题

一、选择题

1. document 对象中查找节点最有效的方法是(　　)。

A. getElementsByTagName　　B. getElementById

C. getElementsByClassName　　D. querySelectorAll

2. 要动态改变 div 节点对象中的内容，可以使用的方法有(　　)。

A. innerText　　B. innerHTML　　C. split　　D. join

3. 对于手指在触摸屏上移动，应该监听 HTML 节点对象的(　　)事件。

A. ontouchstart　　B. ontouchmove　　C. ontouchend　　D. ontouchcancel

4. 当使用 DOM 进行样式编程时，如果样式的值需要使用变量，应该使用(　　)对象。

A. className　　B. classList　　C. style　　D. css

5. 当需要频繁添加子节点时，创建子节点应该采用(　　)方法，再一次性附加到父节点。

A. createElement　　B. createTextNode

C. createNode　　D. createDocumentFragment

二、判断题

1. document.querySelector 会返回所有匹配 CSS 选择器的节点对象集合。 ( )
2. 使用 style 对象不能读取 HTML 元素对象 style 属性中定义的样式。 ( )
3. 为提高效率,查找固定的 HTML 元素节点对象最好不要放在循环中。 ( )
4. 为 HTML 元素增加自定义 data-属性,浏览器会报错。 ( )
5. cloneNode 的深复制和浅复制没有区别。 ( )

三、填空题

1. DOM 的英文全称是_____,中文意思是_____。
2. 使用 DOM 编程,可以对节点对象_____、_____、_____、_____、_____。
3. 使用 classList 对象可以为 HTML 元素对象_____、_____、_____样式类。
4. 事件的触发过程分为_____、_____、_____三个阶段。
5. JavaScript 中,每个事件的处理函数都有一个_____对象作为参数,它代表事件的状态,例如发生事件中的元素、触点的位置等。

四、简答题

1. 常用的事件有哪些? 如何实现事件监听?
2. 请解释什么是 DOM,它有何用处?

五、编程题

1. 模拟实现上滑刷新和下拉加载效果(见图 5-17),每次生成新的 3 条列表项,其中上滑附加到列表头部,而下拉加载追加到列表尾部,单击每个列表项,还能取出其中的文字(注:请使用触摸事件,并使用 Chrome 的移动设备模式测试)。

2. 实现 tab 选项卡和频道内容切换,完成如图 5-18 所示的效果。

图 5-17 上滑刷新和下拉加载效果

图 5-18 tab 选项卡切换效果