第 6 章　事　　件

事件可以说是 JavaScript 最引人注目的特性，因为它提供了一个平台，让用户不仅可以浏览页面中的内容，而且能够跟页面进行交互。本章围绕 JavaScript 处理事件的特性进行讲解，主要包括事件流、事件的监听、事件的类型以及浏览器的兼容性问题等。

6.1　事件流

浏览器在最初开始支持事件时，同一个事件仅仅只有一个元素能够响应，而到了 IE 4 和 Netscape Navigator 4 时代，Microsoft、Netscape 这两家公司都认为仅支持单一事件是不够的，因此纷纷提出了事件流（event flow）的概念。本节主要介绍事件流的相关背景，为后续章节打下基础。

6.1.1　冒泡型事件

浏览器中的事件模型分为两种：捕获型事件和冒泡型事件。由于 IE 浏览器不支持捕获型事件，因此这里主要讲解冒泡型事件。冒泡型（dubbed bubbling）事件指的是事件按照从最特定的事件目标到最不特定的事件目标的顺序逐一触发，如例 6.1 所示。

【例 6.1】冒泡型事件（光盘文件：第 6 章\6-1.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>冒泡型事件</title>
<script language="javascript">
function add(sText){
    var oDiv = document.getElementById("display");
    oDiv.innerHTML += sText; //输出单击顺序
}
</script>
</head>

<body onclick="add('body<br>');">
      <div onclick="add('div<br>');">
         <p onclick="add('p<br>');">Click Me</p>
</div>
<div id="display"></div>
</body>
</html>
```

以上代码为<p>标记、 标记、<body>标记都添加了 onclick 函数，用来处理单击鼠标的事件，运行页面，并单击<p>标记中的文字，效果如图 6.1 所示，会发现 3 个 onclick 函数都被触发了，且触发的顺序为<p>标记，它的父标记 ，以及最后的<body>标记。

从例6.1中可以看到，单击鼠标的事件像冒泡一样从DOM层次结构的最底端往上一级级升，如图 6.2 所示。

图 6.1　冒泡型事件

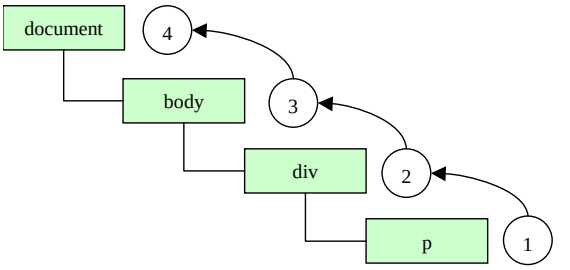

图 6.2　冒泡过程

在 IE 7 中甚至<html>标记都可以添加 onclick 函数，修改代码如下：

```
<html onclick="add('html<br>');">
```

这时 IE 7 上的运行结果如图 6.3 所示，冒泡过程中多了<html>一步。

但是这个事件在 Firefox 上的出现顺序与 IE 浏览器有出入，Firefox 上<html>事件出现在<body>事件之前，如图 6.4 所示。

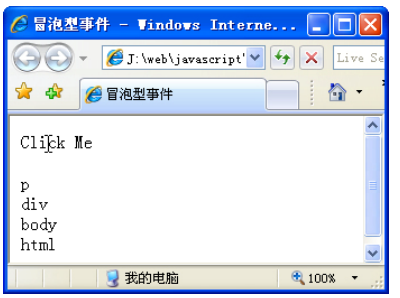

图 6.3　<html>标记的 onclick 函数

图 6.4　Firefox 上的冒泡顺序

经验：

不光 Firefox，IE 以前的版本在处理<html>标记级别的事件时顺序也有出入，因此无论任何情况，都应该尽量避免在<html>标记级别上处理事件。

6.1.2　捕获型事件

在 Netscape Navigator 4 时代，还有一种捕获型事件（event capturing），它与冒泡型事件正好是相反的，即从不精确的对象到最精确的对象。如果设置了捕获型事件，前面的例子将会反向进行，如图 6.5 所示。

这种事件也被称作自顶向下事件模型，因为它是从 DOM 层次的顶端开始向下延伸的。由于 IE 浏览器不支持这种类型的事件，因此读者只需要了解有这样一种事件即可。

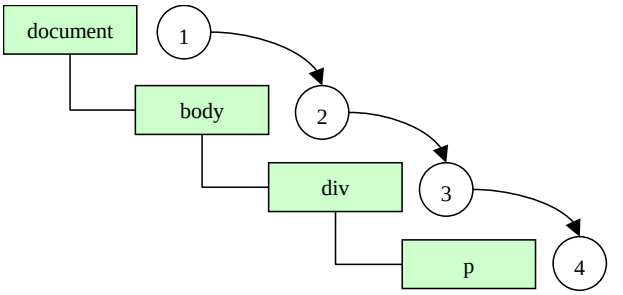

图 6.5 捕获型事件

6.2 事件监听

从上一节的例子可以看到，页面中的事件都需要一个函数来响应，这类函数通常称之为事件处理函数（event handler），或者从另外一个角度来看，这些函数都在实时监听着是否有事件发生，称之为事件监听函数（event listener）。然而对于不同的浏览器而言，事件监听函数的调用区别比较大，本节分别讨论各个浏览器的事件监听方法。

6.2.1 通用监听方法

通常对于简单的事件，没有必要编写大量复杂的代码，直接在 HTML 的标签中就可以分配事件处理函数，而且通常兼容性很好，例如：

```
<p onclick="add('p<br>');">Click Me</p>
```

这是例 6.1 中的<p>标签，直接添加 onclick 函数进行事件的监听。在 HTML 中几乎所有的标签都有 onclick 方法。另外，还可以在标签中直接采用 JavaScript 语句，例如：

```
<p onclick="alert('我被点击了');">Click Me</p>
```

这种方法在主流的浏览器中的兼容性也十分地强，在 Firefox 中的运行结果如图 6.6 所示，单击<p>标签直接弹出对话框。

图 6.6 直接添加监听函数

另外，考虑到结构、行为的分离，通常采用如例 6.2 所示的方法来实现事件监听，这种方法也是实际中运用比较多的。

【例 6.2】实现事件监听（光盘文件：第 6 章\6-2.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
```

```
<head>
<title>监听函数</title>
<script language="javascript">
window.onload = function(){
    var oP = document.getElementById("myP");     //找到对象
    oP.onclick = function(){                                    //设置事件监听函数
        alert('我被点击了');
    }
}
</script>
</head>

<body>
    <div>
        <p id="myP">Click Me</p>
    </div>
</body>
</html>
```

以上代码没有在 HTML 文档结构中采用任何 JavaScript，仅仅只是设置了<p>标签的 id 属性。然后在代码段中为该标记添加了和以下代码所示的匿名函数。

```
oP.onclick = function(){                    //设置事件监听函数
    alert('我被点击了');
}
```

同时将这段函数放到了 window 对象的 onload 函数中，这也保证了 DOM 结构在完全建立后再搜索<p>节点。该例的运行结果如图 6.7 所示。

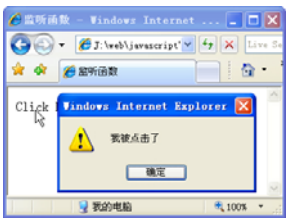

图 6.7 onclick 函数

以上介绍的这两种方法都十分便捷，在处理一些小功能时通常被广大开发者所喜爱。但是对于同一个事件，它们都只能添加一个函数。例如对于<p>标记的 onclick 函数，利用这两种方法都只能有一个函数。因此，IE 浏览器有了自己的解决办法，同时标准 DOM 则规定了另外一种方法。

6.2.2 IE 中的监听方法

在 Microsoft 公司的 IE 浏览器中，每个元素都有两个方法来处理事件的监听，分别是 attachEvent()和 detachEvent()。从它们的函数名称就能看出来，attachEvent()是用来给某个元素添加事件处理的函数，而 detachEvent()则是用来删除元素上的事件监听的函数，它们的语法如下：

```
[object].attachEvent("event_handler", fnHandler);
[object].detachEvent("event_handler", fnHandler);
```

其中 event_handler 表示事件的名称，如“onclick”、“onload”、“onmouseover”等，fnHandler 即为监听函数的名称。

注意：

这里使用的是函数的名称，而不是加上括号的运行结果，类似“ fnHandler() ”是错误的写法。

上一节的示例中可以用 attachEvent()方法替代添加监听函数，而当单击了一下以后，可以用 detachEvent()将监听函数删除，使其不再响应单击事件，如例 6.3 所示。

【例 6.3】IE 的事件监听函数（光盘文件：第 6 章\6-3.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>IE 的监听函数</title>
<script language="javascript">
function fnClick(){
    alert("我被点击了");
    oP.detachEvent("onclick",fnClick);        //单击了一次后删除监听函数
}
var oP;
window.onload = function(){
    oP = document.getElementById("myP"); //找到对象
    oP.attachEvent("onclick",fnClick);        //添加监听函数
}
</script>
</head>

<body>
    <div>
        <p id="myP">Click Me</p>
    </div>
</body>
</html>
```

通过以上的代码可以清晰地看到 attachEvent()和 detachEvent()的使用方法，在 IE 7 中运行的结果如图 6.8 所示。在用户单击了一次<p>标记后，监听函数被删除，再单击则没有对话框弹出，这也是前面的方法所无法实现的。

图 6.8 IE 的监听函数

正如前面提到的，这种方法可以为同一个元素添加多个监听函数，如例 6.4 所示。

【例 6.4】同时添加多个事件监听函数（光盘文件：第 6 章\6-4.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>多个监听函数</title>
<script language="javascript">
function fnClick1(){
    alert("我被 fnClick1 点击了");
}
function fnClick2(){
    alert("我被 fnClick2 点击了");
}
var oP;
window.onload = function(){
    oP = document.getElementById("myP");     //找到对象
    oP.attachEvent("onclick",fnClick1);     //添加监听函数 1
    oP.attachEvent("onclick",fnClick2);     //添加监听函数 2
}
</script>
```

```
</head>

<body>
    <div>
        <p id="myP">Click Me</p>
    </div>
</body>
</html>
```

这样在单击标签<p>时两个函数便同时调用了，如图 6.9 所示。这里还需要注意两个函数调用的顺序，在 IE 7 中实测，后加入的函数 fnClick2()先被调用了，而先加入的 fnClick1 是后调用的。

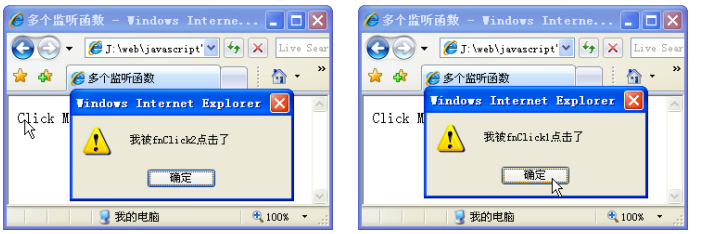

图 6.9　多个监听函数

尽管 fnClick2()先调用，但并不意味着这两个函数是严格意义上的先后，在 fnClick2()开始运行后 fnClick1()马上也开始运行，因为它们都实时监听着<p>标记的 onclick 事件。为了证明这一点，可以在 fnClick2()中加入 detachEvent()语句来删除 fnClick1()，代码如下：

```
function fnClick2(){
    alert("我被 fnClick2 点击了");
    oP.detachEvent("onclick",fnClick1);          //删除监听函数 1
}
```

可以看到第一次单击<p>标签时显示弹出了“我被 fnClick2 点击了”，确定后又弹出了“我被 fnClick1 点击了”，之后的再次单击才是只有 fnClick2()生效，读者可以自行试验。

6.2.3　标准 DOM 的监听方法

与 IE 浏览器的两个方法对应，标准 DOM 也定义了两个方法分别来添加和删除监听函数，即 addEventListener()和 removeEventListener()。与 IE 不同之处在于这两个函数接受 3 个参数，即事件的名称、要分配的函数名和是用于冒泡阶段还是捕获阶段。第 3 个参数如果是捕获阶段则为 true，否则为 false，语法如下：

```
[object].addEventListener("event_name", fnHandler, bCapture);
[object].removeEventListener("event_name", fnHandler, bCapture);
```

这两个函数的使用方法与 IE 的基本类似，只不过需要注意 event_name 中的名称是“click”、“mousemove”等，而不是 IE 中的“onclick”或者“onmousemove”。另外，第 3 个参数 bCapture 通常设置为 false，即冒泡阶段，如例 6.5 所示。

【例 6.5】标准 DOM 的事件监听方法（光盘文件：第 6 章\6-5.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>标准 DOM 的事件监听</title>
```

```
<script language="javascript">
function fnClick1(){
    alert("我被 fnClick1 点击了");
    //oP.removeEventListener("click",fnClick2,false);       //删除监听函数 2
}
function fnClick2(){
    alert("我被 fnClick2 点击了");
}
var oP;
window.onload = function(){
    oP = document.getElementById("myP");          //找到对象
    oP.addEventListener("click",fnClick1,false);              //添加监听函数 1
    oP.addEventListener("click",fnClick2,false);              //添加监听函数 2
}
</script>
</head>

<body>
    <div>
            <p id="myP">Click Me</p>
    </div>
</body>
</html>
```

例 6.5 直接由例 6.4 修改而来，可以看到同样可以为同一个对象的相同事件添加多个监听函数，但运行结果却有较大的区别。在 Firefox 中与 IE 正好相反，先添加的监听函数 fnClick1()先运行，后添加的 fnClick2()后运行，如图 6.10 所示。

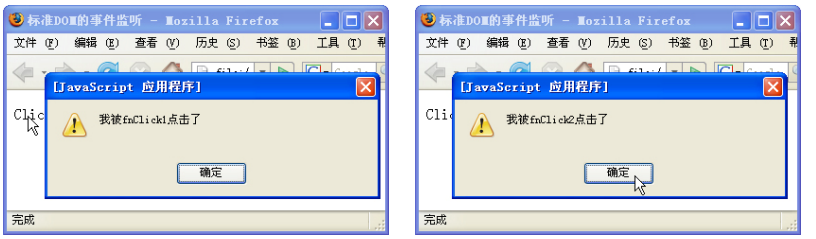

图 6.10 标准 DOM 浏览器的监听函数

如果在先添加的函数 fnClick1()中使用 removeEventListener()将 fnClick2()删除，就会发现标准 DOM 中的监听函数是严格按顺序执行的，即执行完 fnClick1()后才执行 fnClick2()，因此 fnClick1()中的 removeEventListener()会使得 fnClick2()不再运行，这与 IE 浏览器也是比较大的区别，读者可以自行试验。

6.3 事件对象

浏览器中的事件都是以对象的形式存在的，同样 IE 浏览器与标准 DOM 浏览器之间在获取事件对象上也存在差别。在 IE 浏览器中事件对象是 window 对象的一个属性 event，访问时通常采用如下方法。

```
oP.onclick = function(){
    var oEvent = window.event;
}
```

尽管它是 window 对象的属性，但 event 对象还是只能在事件发生时被访问，所有的事件处理函数执行完之后，该对象就消失了。

而标准的 DOM 中规定 event 对象必须作为惟一的参数传给事件处理函数，因此在类似 Firefox 浏览器中访问事件对象通常将其作为参数，代码如下：

```
oP.onclick = function(oEvent){

}
```

因此为了兼容两种浏览器，通常采用下面的方法。

```
oP.onclick = function(oEvent){
    if(window.event) oEvent = window.event;
}
```

浏览器在获取了事件对象后就可以通过它的一系列属性和方法来处理各种具体事件了，例如鼠标事件、键盘事件和浏览器事件等。表 6.1 罗列了事件常用的属性和方法，供读者具体使用时查询。

表 6.1　　事件常用的属性和方法

IE	标准 DOM	类　型	可读/可写	说　明
altKey	altKey	Boolean	R/W	按下 Alt 键则为 true，否则为 false
button	button	Integer	R/W	鼠标事件，值对应按下的鼠标键，详见 6.4.1 节
cancelBubble	cancelBubble	Boolean	IE 中 R/W，标准 DOM 中 R	IE 中设置为 true 可取消事件向上冒泡，标准 DOM 中只读
--	stopPropagation()	Function	N/A	可以调用该方法来阻止事件向上冒泡
clientX	clientX	Integer	IE 中 R/W，标准 DOM 中 R	鼠标指针在客户端区域的坐标，不包括工具栏、滚动条等
chilentY	clientY			
ctrlKey	ctrlKey	Boolean	同上	按下 Ctrl 键则为 true，否则为 false
fromElement	relatedTarget	Element	同上	鼠标指针所离开的元素
toElement	relatedTarget	Element	同上	鼠标指针正在进入的元素
--	charCode	Integer	R	按下按键的 Unicode 值
keyCode	keyCode	Integer	R/W	IE 中 keypress 事件表示按下按键的 Unicode 值，keydown/keyup 事件为按键的数字代号。标准 DOM 中 keypress 时为 0，其余为按下按键的数字代号
--	detail	Integer	R	鼠标按键被单击的次数
returnValue	--	Boolean	R/W	设置为 false 时可取消事件的默认行为
--	preventDefault()	Function	N/A	可以调用该方法来阻止事件的默认行为
screenX	screenX	Integer	IE 中 R/W，标准 DOM 中 R	鼠标指针相对于整个计算机屏幕的坐标值
screenY	screenY			
shiftKey	shiftKey	Boolean	同上	按下 Shift 键则为 true，否则为 false
srcElement	target	Element	同上	引起事件的元素/对象
type	type	String	同上	事件的名称

从表 6.1 中可以看出，两类浏览器处理事件还是有一些相似之处的，例如 type 属性便是各种浏览器所兼容的，它表示获取的事件类型，返回类似“click”、“mousemove”之类的值。

```
var sType = oEvent.type;
```

这对于同一个函数处理多种事件时十分有用，如例 6.6 所示。

【例 6.6】用同一个函数处理多种事件（光盘文件：第 6 章\6-6.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html>
<head>
<title>事件的类型</title>
<script language="javascript">
function handle(oEvent){
    var oDiv = document.getElementById("display");
    if(window.event) oEvent = window.event;                    //处理兼容性，获得事件对象
    if(oEvent.type == "click")                                 //检测事件名称
        oDiv.innerHTML += "你点击了我  ";
    else if( oEvent.type == "mouseover")
        oDiv.innerHTML += "你移动到我上方了  ";

}
window.onload = function(){
    var oImg = document.getElementsByTagName("img")[0];
    oImg.onclick = handle;
    oImg.onmouseover = handle;
}
</script>
</head>

<body>
    <img src="01.jpg" border="0">
    <div id="display"></div>
</body>
</html>
```

以上代码为图片<img>添加了两个事件响应函数，而这两个事件采用的却是同一个函数，在这个函数中首先考虑兼容性获得事件对象，然后利用 type 属性判断事件的名称。运行结果如图 6.11 所示。

图 6.11 事件的类型

在检测 Shift、Alt、Ctrl 这 3 个按键时，两类浏览器使用的方法也完全一样，都具有 shiftKey、altKey 和 ctrlKey 这 3 个属性，代码如下：

```
var bShift = oEvent.shiftKey;
var bAlt = oEvent.altKey;
var bCtrl = oEvent.ctrlKey;
```

另外，在获取鼠标指针位置上两类浏览器都有两套值可用，分别为 clientX、clientY 和 screenX、screenY。其中 clientX 和 clientY 表示鼠标指针在客户端区域的位置，不包括浏览器的状态栏、菜单栏等，代码如下，效果如图 6.12 所示。

```
var iClientX = oEvent.clientX;
var iClientY = oEvent.clientY;
```

图 6.12 clientX 和 clientY

而 screenX 和 screenY 则指的是鼠标指针在整个计算机屏幕的位置，代码如下，效果如图 6.13 所示。

```
var iScreenX = oEvent.screenX;
var iScreenY = oEvent.screenY;
```

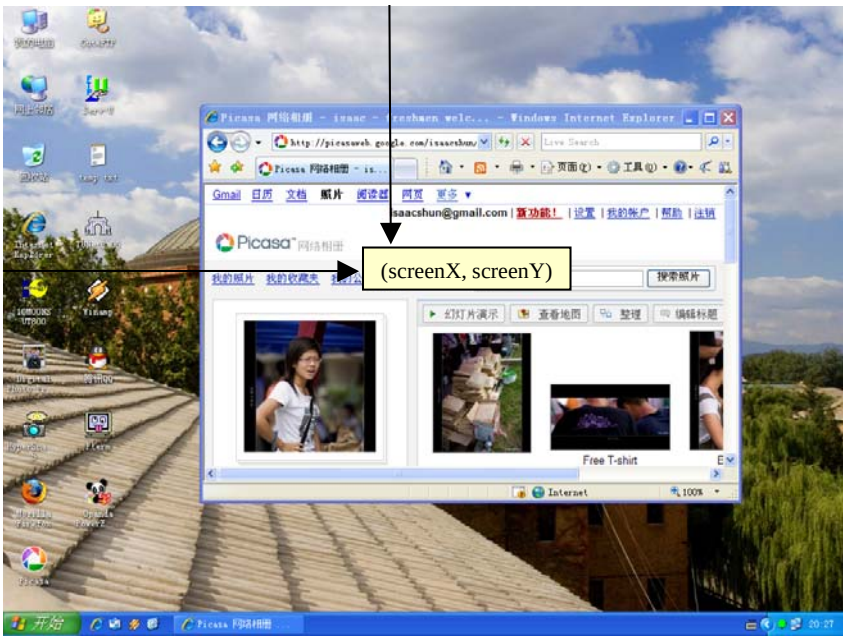

图 6.13 screenX 和 screenY

在这两种获取鼠标指针位置的方法中坐标的原点都是各自的左上角，*x* 轴的正方向为从左到右，*y* 轴的正方向为从上到下，如图 6.14 所示。

很多时候，开发者希望知道事件是由哪个对象触发的，即事件的目标（target）。假设为<p>元素分配 onclick 事件处理函数，触发 click 事件时<p>就被认为是目标。

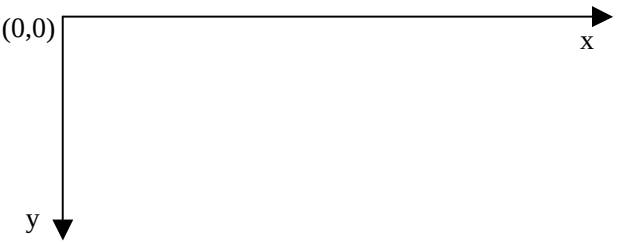

图 6.14 坐标系

在 IE 浏览器中目标包含在 event 对象的 srcElement 属性中，代码如下：

```
var oTarget = oEvent.srcElement;
```

而在标准的 DOM 浏览器中，目标则包含在 target 属性中，代码如下：

```
var oTarget = oEvent.target;
```

完整代码如例 6.7 所示。

【例 6.7】获取事件的目标（光盘文件：第 6 章\6-7.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>事件的目标</title>
<script language="javascript">
function handle(oEvent){
    if(window.event) oEvent = window.event;        //处理兼容性，获得事件对象
    var oTarget;
    if(oEvent.srcElement)                          //处理兼容性，获取事件目标
        oTarget = oEvent.srcElement;
    else
        oTarget = oEvent.target;
    alert(oTarget.tagName);                        //弹出目标的标记名称
}
window.onload = function(){
    var oImg = document.getElementsByTagName("img")[0];
    oImg.onclick = handle;
}
</script>
</head>

<body>
    <img src="02.jpg" border="0">
</body>
</html>
```

由于事件目标的属性在两类浏览器上不同，因此代码首先必须保证兼容性，通常的做法就是直接将对象作为 if 语句的条件，代码如下：

```
if(oEvent.srcElement)                          //处理兼容性，获取事件目标
    oTarget = oEvent.srcElement;
else
    oTarget = oEvent.target;
```

这种方法在其他属性中也是常用的，代码的运行结果如图 6.15 所示。当单击图片时对话框显示了事件的目标 IMG。

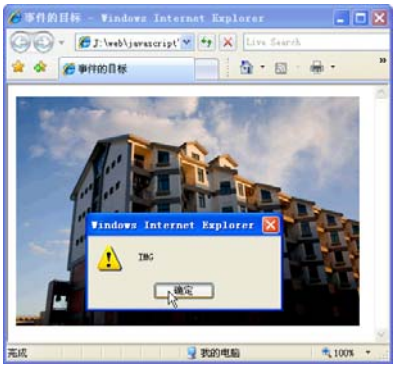

图 6.15　对象的目标

6.4　事件的类型

对于用户而言，最常用的事件无非是鼠标、键盘和浏览器，本节对这 3 种事件分别进行介绍，使读者对处理具体的事件问题有一个总体的概念。

6.4.1　鼠标事件

鼠标事件是用户最常用的事件，通常包括表 6.2 中所列的几种。

表 6.2　　鼠标事件的种类

事 件 名 称	说　　明
click	单击鼠标左键时触发
dbclick	双击鼠标左键时触发
mousedown	单击任意一个鼠标按键时触发
mouseout	鼠标指针在某个元素上，移出该元素边界时触发
mouseover	鼠标指针移到另一个元素上时触发
mouseup	松开鼠标任意一个按键时触发
mousemove	鼠标指针在某个元素上移动时持续触发

表 6.2 中的这些事件几乎每个都被频繁使用，下面的例 6.8 简单地描述了鼠标事件的类型，也可以用来测试各种鼠标事件。

【例 6.8】控制鼠标事件（光盘文件：第 6 章\6-8.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD
/xhtml1-transitional.dtd">
<html>
<head>
<title>鼠标事件</title>
<script language="javascript">
function handle(oEvent){
    if(window.event) oEvent = window.event;        //处理兼容性，获得事件对象
    var oDiv = document.getElementById("display");
    oDiv.innerHTML += oEvent.type + "<br>";              //输出事件名称
}
window.onload = function(){
    var oImg = document.getElementsByTagName("img")[0];
    oImg.onmousedown = handle;     //将鼠标事件除了 mousemove 外都监听
```

```
        oImg.onmouseup = handle;
        oImg.onmouseover = handle;
        oImg.onmouseout = handle;
        oImg.onclick = handle;
        oImg.ondblclick = handle;
    }
    </script>
    </head>

    <body>
        <img src="03.jpg" border="0" style="float:left; padding:0px 8px 0px 0px;">
        <div id="display"></div>
    </body>
    </html>
```

以上代码将除了 mousemove 外的所有鼠标事件都予以监听，这样便可以很清楚地看到鼠标在图片上的动作所触发的一系列事件。例如单击鼠标左键会先触发 mousedown，然后是 mouseup，最后才是常用的 click 事件，如图 6.16 所示。

而如果是双击鼠标左键，两类浏览器又再次发生了区别，标准的 DOM 浏览器会按照 mousedown→mouseup→click→mousedown→mouseup→click→dblclick 的顺序触发，即两次单击合成一次双击。在 Firefox 中的效果如图 6.17 所示。

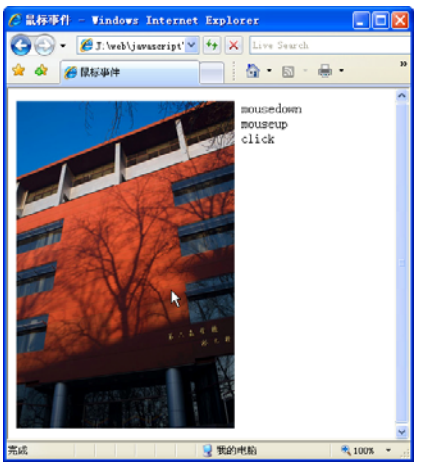

图 6.16 鼠标的 click 事件

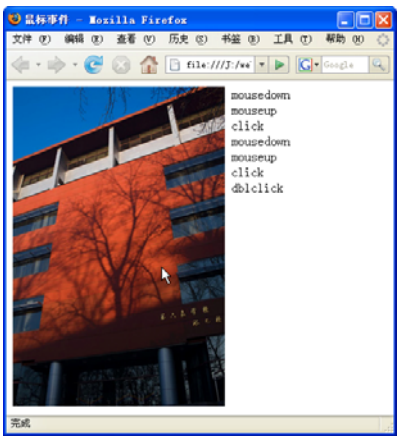

图 6.17 鼠标的双击 dblclick 事件

而双击事件在 IE 浏览器中的触发顺序为 mousedown→mouseup→click→mouseup→dblclick，即一次单击紧接着 mouseup，然后判断为双击。

经验：

对于鼠标事件的触发，如果编程时需要涉及双击事件，应当尽量避免使用它的触发过程，原因就在于两类浏览器的不同。

对于其他鼠标事件的触发过程，情况也是十分类似的，读者可以利用该程序自己测试，这里就不一一讲解。

鼠标事件中另外一个重要的属性就是 button，它表示鼠标按键的键值。非常遗憾的是 IE、

Firefox 这两个浏览器对该属性的支持大相径庭，具体如表 6.3 所示。

表 6.3　　button 属性及其不同浏览器中的键值

button 的值	IE 中的按键	Firefox 中的按键
0	未按下按键	左键
1	左键	中键（滑轮）
2	右键	右键
3	同时按下左、右键	不支持组合键，未按下任何键时 button 值为 undefined
4	中键（滑轮）	
5	同时按下左、中键	
6	同时按下右、中键	
7	同时按下左、中、右键	

简单的测试程序如例 6.9 所示。

【例 6.9】输出鼠标事件 button 属性的值（光盘文件：第 6 章\6-9.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>button 属性</title>
<script language="javascript">
function TestClick(oEvent){
    var oDiv = document.getElementById("display");
    if(window.event)
        oEvent = window.event;
    oDiv.innerHTML += oEvent.button;       //输出 button 的值
}
document.onmousedown = TestClick;
window.onload = TestClick;      //测试未按下任何键
</script>
</head>

<body>
<div id="display"></div>
</body>
</html>
```

以上代码简单地将鼠标事件 mousedown 的 button 值进行输出。由于 mousedown 在按任何按键时都能响应，因此能很好地描述 button 的各项值，其中 window.onload 测试未按下任何按键时 button 的值。在 IE 中的运行结果如图 6.18 所示。

实际测试中会发现，当同时按下两个按键时，都会显示两个值，前一个是其中某个按键的值，后一个才是两个按键同时按下时的值。这也是因为任何双击都不可能做到 1 毫秒不差，浏览器非常灵敏的缘故。同时按下 3 键则会显示 3 个值，道理是一样的。

而以上代码在 Firefox 浏览器中的运行效果如图 6.19 所示，初始化时为 undefined，之后只有 0、1、2 这 3 个值。

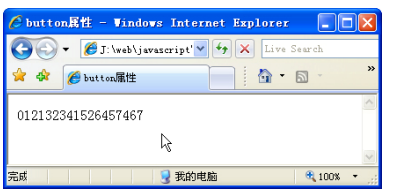

图 6.18　IE 中 button 属性的值

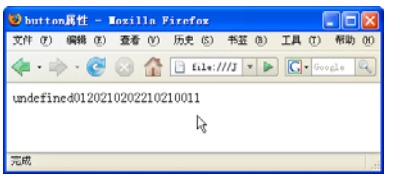

图 6.19　Firefox 中的 button 属性值

6.4.2 键盘事件

除了鼠标之外，与用户打交道最多的恐怕就是键盘了。键盘的事件种类不多，仅 3 种事件，具体如表 6.4 所示。

表 6.4 键盘事件的种类

事 件	说 明
keydown	按下键盘上某个按键时触发，一直按住某键则会持续触发
keypress	按下某个按键并产生字符时触发，即忽略 Shift、Alt、Ctrl 等功能键
keyup	释放某个按键时触发

尽管所有的元素都支持键盘事件，但通常键盘事件只有在文本框中才显得有实际的意义，测试程序如例 6.10 所示。

【例 6.10】控制键盘事件（光盘文件：第 6 章\6-10.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>键盘事件</title>
<script language="javascript">
function handle(oEvent){
    if(window.event) oEvent = window.event;        //处理兼容性，获得事件对象
    var oDiv = document.getElementById("display");
    oDiv.innerHTML += oEvent.type + "  ";          //输出事件名称
}
window.onload = function(){
    var oTextArea = document.getElementsByTagName("textarea")[0];
    oTextArea.onkeydown = handle;   //监听所有键盘事件
    oTextArea.onkeyup = handle;
    oTextArea.onkeypress = handle;
}
</script>
</head>

<body>
    <textarea rows="4" cols="50"></textarea>
    <div id="display"></div>
</body>
</html>
```

与例 6.8 一样，该例对所有的键盘事件都进行监听，可以看到按下某个会产生字符的按键时，触发的顺序为 keydown→keypress→keyup。IE 7 和 Firefox 中运行效果分别如图 6.20 和图 6.21 所示，两类浏览器中的显示结果是相同的。

图 6.20 IE 7 键盘事件

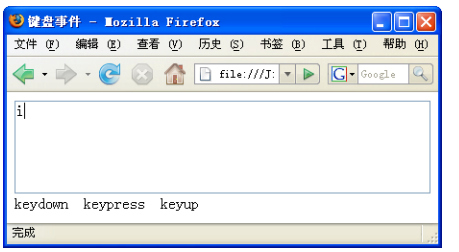

图 6.21 Firefox 键盘事件

对于键盘事件而言，最重要的并不是事件的名称，而是所按的是什么键。由于 IE 浏览器没有 charCode 属性，而 keyCode 只有在 keydown、keyup 事件发生时才与标准 DOM 的 keyCode 相同，在 keypress 事件中等同于 charCode，因此常采用如下方法。

```
oEvent.charCode = (oEvent.type == "keypress") ? oEvent.keyCode : 0;
```

之所以通常不采用 keyCode 是因为它表示键盘按键，而不是输出的字符，因此输出“a”和“A”时，keyCode 的值是相等的，charCode 则以字符为区分。另外，在 keypress 事件中，标准 DOM 的 keyCode 值始终为 0，如例 6.11 所示。

【例 6.11】键盘事件的相关属性（光盘文件：第 6 章\6-11.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>键盘事件</title>
<script language="javascript">
function handle(oEvent){
    if(window.event){
        oEvent = window.event;          //处理兼容性，获得事件对象
        //设置 IE 的 charCode 值
        oEvent.charCode = (oEvent.type == "keypress") ? oEvent.keyCode : 0;
    }
    var oDiv = document.getElementById("display");
    //输出测试
    oDiv.innerHTML += oEvent.type + ": charCode:" + oEvent.charCode + " keyCode:" + oEvent.keyCode + "<br>";
}
window.onload = function(){
    var oTextArea = document.getElementsByTagName("textarea")[0];
    oTextArea.onkeypress = handle;
    oTextArea.onkeydown = handle;
}
</script>
</head>

<body>
    <textarea rows="4" cols="50"></textarea>
    <div id="display"></div>
</body>
</html>
```

在 IE 浏览器中先输入小写字母“a”，再按 Shift 键输入一个大写字母“A”，可以看到 charCode 和 keyCode 在不同事件类型中的值，如图 6.22 所示。

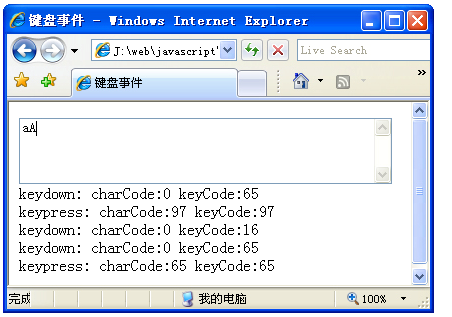

图 6.22 IE 中的 charCode 和 keyCode

可以看到只有 keypress 事件的 charCode 能够很好地区分键值（“a”为 97，“A”为 65）。第 3 行为按 Shift 键，charCode 不予显示。keyCode 并不能区分大小写，“a”和“A”都输出了 65。

在 Firefox 中的情况类似，charCode 的值较明显地反映了输出字符，而 keyCode 却不能运行在 keypress 事件中，如图 6.23 所示。

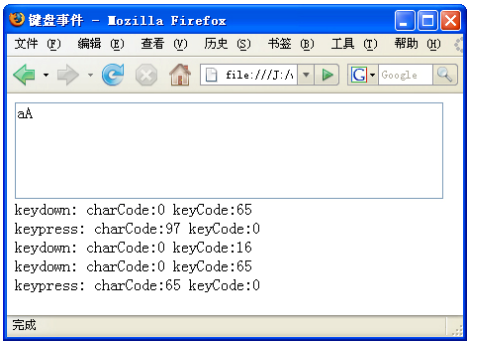

图 6.23 Firefox 中的 charCode 和 keyCode

6.4.3 HTML 事件

对于浏览器而言，各种 HTML 对象同样有着自己的事件，有一些也是用户常常会接触到的，例如 load、error 和 select 等。一些常用的 HTML 事件如表 6.5 所示。

表 6.5 常用的 HTML 事件

事 件	说 明
load	页面完全加载后在 window 对象上触发，图片加载完成后在其上触发
unload	页面完全卸载后在 window 对象上触发，图片卸载完成后在其上触发
error	脚本出错时在 window 对象上触发，图像无法载入时在其上触发
select	选择了文本框的一个或多个字符时触发
change	文本框失去焦点时，并且在它获取焦点后内容发生过改变时触发
submit	单击“提交”按钮时在表单 form 上触发
focus	任何元素或窗口获取焦点时触发
blur	任何元素或窗口失去焦点时触发

载入事件 load 是最常用的事件之一，因为在页面载入完成之前，DOM 的框架还没有搭建完毕，因此任何相关操作都不能发生。给 window 对象分配 load、unload 事件等同于<body>标记的 onload、onunload 方法，即：

```
<script language="javascript">
window.onload = function(){
    alert("Page Loaded.");
}
</script>
```

等同于：

```
<body onload="alert('Page Loaded.');">
```

unload 事件与 load 正好相反，发生在页面卸载的时候，使用频率不高，但一些电子商务的网站通常在用户关闭窗口后弹出对话框表示感谢、欢迎再次光临等效果，就是采用 unload 事件实现的。

6.5 实例 1：屏蔽鼠标右键

有时为了某些原因需要屏蔽鼠标的右键，让用户不能使用它的快捷功能，而只能使用网页本身提供的功能。本例介绍两种方法以分别实现不同的屏蔽效果。

6.5.1 方法 1

如 6.4.1 节中所描述，鼠标事件中 button 的值在各个浏览器上大相径庭，但非常幸运的是按下鼠标右键时值都为 2，因此屏蔽鼠标右键最直接的方法莫过于在 button 值为 2 的时候进行相应的处理，如例 6.12 所示。

【例 6.12】屏蔽鼠标右键方法 1（光盘文件：第 6 章\6-12.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>屏蔽鼠标右键</title>
<script language="javascript">
function block(oEvent){
    if(window.event)
        oEvent = window.event;
    if(oEvent.button == 2)
        alert("鼠标右键不可用");
}
document.onmousedown = block;
</script>
</head>

<body>
    <p>屏蔽鼠标右键</p>
</body>
</html>
```

以上代码在 IE 浏览器中的运行结果如图 6.24 所示。单击鼠标右键时弹出了对话框，因此右键的菜单被变相屏蔽了。

不过非常可惜的是，这种想当然的方法在 Firefox 中再次遇到了困难。Firefox 并没有因为变相地弹出对话框而屏蔽掉右键的菜单，如图 6.25 所示。

图 6.24 屏蔽鼠标右键

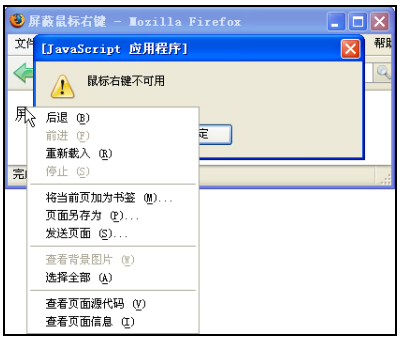

图 6.25 Firefox 中的情况

从图 6.25 中可以看到，确实弹出了警告对话框，但右键菜单也在其上弹了出来，这是浏览器兼容性方面的问题。接下来将要介绍的第 2 种方法能够解决这个兼容性问题。

6.5.2 方法 2

其实鼠标右键会触发另外一个事件，即右键菜单 contextmenu 事件。如果希望彻底屏蔽鼠标右键，最有效的办法就是屏蔽 document 对象的 contextmenu 事件。这里需要应用 IE 浏览器的

returnValue 属性和标准 DOM 的 preventDefault()方法，如例 6.13 所示。

【例 6.13】屏蔽鼠标右键方法 2（光盘文件：第 6 章\6-13.html）

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>屏蔽鼠标右键</title>
<script language="javascript">
function block(oEvent){
    if(window.event){
        oEvent = window.event;
        oEvent.returnValue = false;     //取消默认事件
    }else
        oEvent.preventDefault();        //取消默认事件
}
document.oncontextmenu = block;
</script>
</head>

<body>
    <p>屏蔽鼠标右键</p>
</body>
</html>
```

以上代码将右键菜单完全屏蔽了，而且在两个浏览器中的效果都很好，运行结果如图 6.26 所示。

图 6.26 屏蔽鼠标右键

contextmenu 事件在自定义右键菜单时也常常使用，即屏蔽系统菜单后自定义一个 块来显示新的菜单，读者可以自己试验。

6.6 实例 2：伸缩的两级菜单

在 3.8.2 节的例 3.26 中介绍了无需表格的菜单的制作方法，但该例的菜单只有一级，实际网页中常常需要多级菜单。本节在原例的基础之上加入了 JavaScript 代码，实现两级菜单的单击显隐，最终效果如图 6.27 所示。

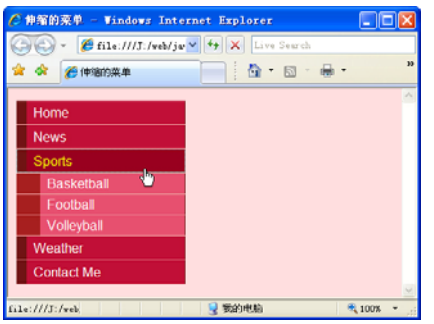

图 6.27 伸缩的两级菜单

6.6.1 建立 HTML 框架

首先并不一定所有的一级菜单都有子菜单；其次考虑到代码的通用性，一级菜单和二级菜单的项数随时都可能变化，因此 HTML 框架如例 6.14 所示。

【例 6.14】制作伸缩的两级菜单（光盘文件：第 6 章\6-14.html）

```
<div id="navigation">
    <ul id="listUL">
        <li><a href="#">Home</a></li>
        <li><a href="#">News</a>
        <ul>
            <li><a href="#">Lastest News</a></li>
                <li><a href="#">All News</a></li>
            </ul>
        </li>
        <li><a href="#">Sports</a>
        <ul>
            <li><a href="#">Basketball</a></li>
                <li><a href="#">Football</a></li>
                <li><a href="#">Volleyball</a></li>
            </ul>
        </li>
        <li><a href="#">Weather</a>
        <ul>
            <li><a href="#">Today's Weather</a></li>
                <li><a href="#">Forecast</a></li>
            </ul>
        </li>
        <li><a href="#">Contact Me</a></li>
    </ul>
</div>
```

以上 HTML 框架直接由原来的例 3.26 修改而来，可以看到首尾两个一级菜单没有子菜单，而中间的 3 个都有子菜单。

6.6.2 设置各级菜单的 CSS 样式风格

考虑到子菜单的样式风格应该区别于一级菜单，因此将原来的 CSS 样式都加入子选择器。代码如下：

```
#navigation > ul {
    list-style-type:none;                    /* 不显示项目符号 */
    margin:0px;
    padding:0px;
}
#navigation > ul > li {
    border-bottom:1px solid #ED9F9F;         /* 添加下划线 */
}
#navigation > ul > li > a{
    display:block;                           /* 区块显示 */
    padding:5px 5px 5px 0.5em;
    text-decoration:none;
    border-left:12px solid #711515;          /* 左边的粗红边 */
    border-right:1px solid #711515;          /* 右侧阴影 */
}
#navigation > ul > li > a:link, #navigation > ul > li > a:visited{
    background-color:#c11136;
```

```
    color:#FFFFFF;
}
#navigation > ul > li > a:hover{                    /* 鼠标经过时 */
    background-color:#990020;                       /* 改变背景色 */
    color:#ffff00;                                  /* 改变文字颜色 */
}
```

这样的话原先设置的 CSS 样式只能作用到一级菜单，此时显示效果如图 6.28 所示。如果读者对子选择器有不清楚的地方，可参考 3.3.6 节。

为了配合一级菜单，可为二级菜单也添加相应的 CSS 样式风格，代码如下：

```
/* 子菜单的 CSS 样式 */
#navigation ul li ul{
    list-style-type:none;
    margin:0px;
    padding:0px 0px 0px 0px;
}
#navigation ul li ul li{
    border-top:1px solid #ED9F9F;
}
#navigation ul li ul li a{
    display:block;
    padding:3px 3px 3px 0.5em;
    text-decoration:none;
    border-left:28px solid #a71f1f;
    border-right:1px solid #711515;
}
#navigation ul li ul li a:link, #navigation ul li ul li a:visited{
    background-color:#e85070;
    color:#FFFFFF;
}
#navigation ul li ul li a:hover{
    background-color:#c2425d;
    color:#ffff00;
}
```

这些样式风格都是基于一级菜单的，具体的设置细节这里不再重复，方法与一级菜单的完全相同，读者可参照例 3.26 中的讲解。此时显示效果如图 6.29 所示。

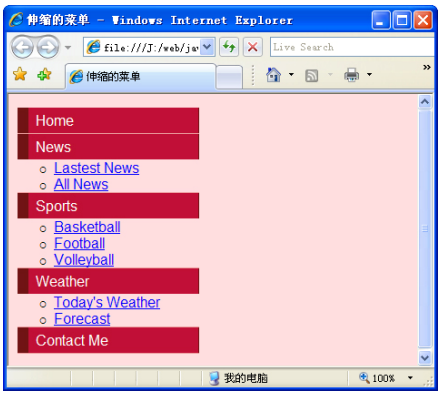

图 6.28 修改 CSS

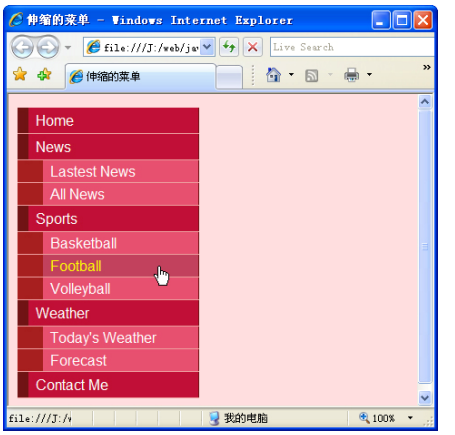

图 6.29 二级菜单

考虑到二级菜单伸缩的两种不同状态，为其分别添加两个 CSS 样式应用于其中的<ul>标记上，代码如下：

```
#navigation ul li ul.myHide{     /* 隐藏子菜单 */
    display:none;
}
#navigation ul li ul.myShow{     /* 显示子菜单 */
    display:block;
}
```

而 HTML 部分也做相应的修改，为二级菜单的<ul>标记添加 class 属性，并设置为 myHide，使得初始化加载页面时二级菜单隐藏，例如：

```
<li><a href="#">News</a>
      <ul class="myHide">
          <li><a href="#">Lastest News</a></li>
          <li><a href="#">All News</a></li>
   </ul>
</li>
```

此时显示效果如图 6.30 所示。

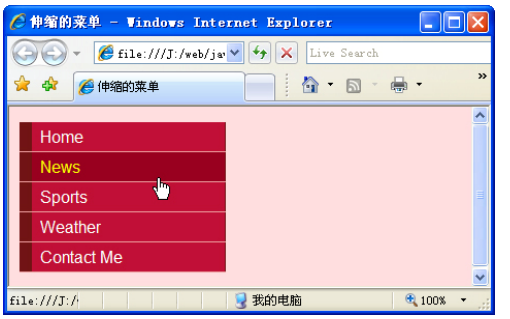

图 6.30　初始化隐藏二级菜单

6.6.3　为菜单添加伸缩效果

由于一级菜单和二级菜单在实际运用中项数变化较大，因此所有的事件控制均采用动态加载的方法，即放在 window.onload 函数中：

```
window.onload = function(){
    //...
}
```

首先找到一级菜单中所有的<li>元素，即<ul id="listUL">中的所有子<li>（不包括孙<li>），然后对于每一个<li>判断其是否拥有二级菜单，即<li>是否包含<ul>，如果包含则说明有二级菜单，可为其添加 onclick 事件，代码如下：

```
window.onload = function(){
    var oUl = document.getElementById("listUL");
    var aLi = oUl.childNodes;            //子元素
    var oA;
    for(var i=0;i<aLi.length;i++){
        //如果子元素为 li，且这个 li 有子菜单 ul
        if(aLi[i].tagName == "LI" && aLi[i].getElementsByTagName("ul").length){
            oA = aLi[i].firstChild;      //找到超链接
            oA.onclick = change;         //动态添加单击函数
        }
    }
}
```

之所以要在 for 循环中再次判断 aLi[i].tagName == "LI"，是为了兼容在 Firefox 中空格也为子元素的特点，这点在前面的章节中也反复提到了。

通过 DOM 对<li>的遍历，无论以后菜单的项数怎么变化，只需要修改 HTML 框架即可，JavaScript 部分则不需要再改动。

而 change 函数的思路很简单，即通过 className 属性，单击鼠标时切换二级菜单中<ul>的风格样式即可，代码如下：

```
function change(){
    //通过父元素 li，找到兄弟元素 ul
    var oSecondDiv = this.parentNode.getElementsByTagName("ul")[0];
    //CSS 交替更换来实现显、隐
    if(oSecondDiv.className == "myHide")
        oSecondDiv.className = "myShow";
    else
        oSecondDiv.className = "myHide";
}
```

这样便完成了整个菜单单击鼠标的伸缩效果，完整代码在光盘第 6 章\6-13.html 文件中，读者可以自行查阅。在 Firefox 中的运行效果如图 6.31 所示，与 IE 中完全相同。

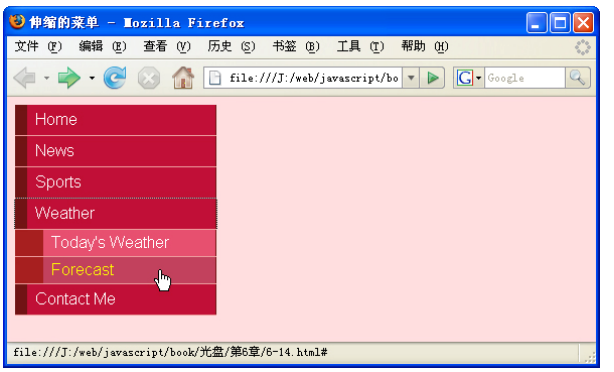

图 6.31 伸缩的菜单

对于三级或更多级的菜单，其制作原理与二级菜单是完全相同的，读者可以自行练习其制作。