第 3 章 JavaScript 事件处理

用户可以通过多种方式与浏览器中的页面进行交互，而事件是交互的桥梁。Web 应用程序开发人员通过 JavaScript 脚本内置的和自定义的事件处理器来响应用户的动作，就可以开发出更具交互性、动态性的页面。

本章主要介绍 JavaScript 脚本中的事件处理的概念、方法，列出了 JavaScript 预定义的事件处理器，并且介绍了如何编写用户自定义的事件处理函数以及如何将它们与页面中用户的动作相关联，以得到预期的交互性能。同时讲述了 IE4 和 NN4 对基本事件模型的扩展，以及 DOM2 标准事件模型的架构等。

3.1 什么是事件

广义上讲，JavaScript 脚本中的事件是指用户载入目标页面直到该页面被关闭期间浏览器的动作及该页面对用户操作的响应。事件的复杂程度大不相同，简单的如鼠标的移动、当前页面的关闭、键盘的输入等，复杂的如拖曳页面图片或单击浮动菜单等。

事件处理器是与特定的文本和特定的事件相联系的 JavaScript 脚本代码，当该文本发生改变或者事件被触发时，浏览器执行该代码并进行相应的处理操作，而响应某个事件而进行的处理过程称为事件处理。

下面就是简单的事件触发和处理过程，如图 3.1 所示。

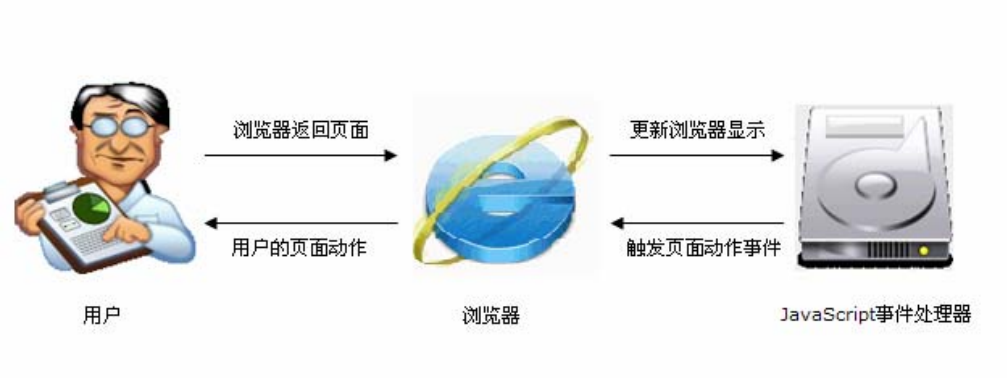

图 3.1 基本的用户动作触发事件示意图

如前所述，JavaScript 脚本中的事件并不限于用户的页面动作如 MouseMove、Click 等，还包括浏览器的状态改变，如在绝大多数浏览器获得支持的 Load、Resize 事件等。Load 事件在浏览器载入文档时触发，如果某事件（如启动定时器、提前加载图片等）要在文档载入时触发，一般都在<body>标记里面加入类似于“onload="MyFunction()"”的语句；Resize 事件则在用户改变了浏览器窗口的大小时触发，当用户改变窗口大小时，有时需改变文档页面的内容布局，使其以恰当、友好的方式显示给用户。

浏览器响应用户的动作，如鼠标单击按钮、链接等，并通过默认的系统事件与该动作相关联，但用户可以编写自己的脚本，来改变该动作的默认事件处理器。举个简单的例子，模拟用户单击页面链接的例子，该事件产生的默认操作是浏览器载入链接的 href 属性对应的

URL 地址所代表的页面，但利用 JavaScript 脚本可很容易编写另外的事件处理器来响应该单击鼠标的动作。考察如下代码：

```
<a name=MyA href="http://www.baidu.com/"
onclick="javascript:this.href='http://www.sina.com/'">MyLinker</a>
```

鼠标单击页面中名为“MyLinker”的文本链接，其默认操作是浏览器载入该链接的 href 属性对应的 URL 地址（本例中为“http://www.baidu.com/”）所代表的页面，但程序员编写了自定义的事件处理器即：

```
onclick="javascript:this.href='http://www.sina.com/'
```

通过该 JavaScript 脚本代码，上述事件处理器取代了浏览器默认的事件处理器，并将页面引导至 URL 地址为“http://www.sina.com/”指向的页面。

现代事件模型中引入 Event 对象，它包含其它对象使用的常量和方法的集合。当事件发生后，产生临时的 Event 对象实例，并附加当前事件的信息如鼠标定位、事件类型等，然后传递给相关的事件处理器进行处理。事件处理完毕后，该临时 Event 对象实例所占据的内存空间被清理出来，浏览器等待其他事件的出现并进行处理。如果短时间内发生的事件较多，浏览器按事件按发生的顺序将这些事件排序，然后按照该顺序依次执行。

事件发生的场合很多，包括浏览器本身的状态改变和页面中的按钮、链接、图片、层等。同时根据 DOM 模型，文本也可以作为对象，并响应相关动作，如鼠标双击、，文本被选择等。当然，事件的处理方法甚至于结果同浏览器环境有很大的关系，但总的来说，浏览器的版本越新，所支持的事件处理器就越多，支持也就越完善。基于此，在编写 JavaScript 脚本时，要充分考虑浏览器的兼容性，以编写适合大多数浏览器的安全脚本。

3.2 HTML 文档事件

HTML 文档事件包括用户载入目标页面直到该页面被关闭期间浏览器的动作及该页面对用户操作的响应，主要分为浏览器事件和 HTML 元素事件两大类。在了解这两类事件之前，先来了解事件捆绑的概念。

3.2.1 事件捆绑

HTML 文档将元素的常用事件（如 onclick、onmouseover 等）当作属性捆绑在 HTML 元素上，当该元素的特定事件发生时，对应于此特定事件的事件处理器就被执行，并将处理结果返回给浏览器。事件捆绑导致特定的代码放置在其所处对象的事件处理器中。考察如下代码：

```
<a href="http://www.baidu.com/" onclick="javascript:alert('You have Clicked the link!')">MyLinker
</a>
```

上述代码为“MyLinker”文本链接定义了一个 Click 事件的处理器，返回警告框“You have Clicked the link!”。

同样，也可将该事件处理器独立出来，编成单独的函数来实现同样的功能。将下列代码加入文档的<body>和</body>标记对之间：

```
<a href="http://www.baidu.com/" onclick="MyClick()">MyLinker</a>
```

自定义的函数 MyClick()实现如下：

```
function MyClick()
{
   alert("You have Clicked the link!");
```

```
}
```

鼠标单击“MyLinker”链接后，浏览器调用自定义的 Click 事件处理器，并将结果（警告框“You have Clicked the link!”）返回给浏览器。由事件处理器的实现形式来看，<a>标记的 onclick 事件与其 href 属性地位均等，实现了 HTML 中的事件捆绑策略。

3.2.2 浏览器事件

浏览器事件指载入文档直到该文档被关闭期间的浏览器事件，如浏览器载入文档事件 onload、关闭该文档事件 onunload、浏览器失去焦点事件 onblur、获得焦点事件 onfocus 等。先考察如下的代码：

```
//源程序 3.1
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script type="text/javascript">
<!--
window.onload = function ()
{
   var msg="\nwindow.load 事件 : \n\n";
   msg+="            浏览器载入了文档!";
   alert(msg);
}
window.onfocus = function ()
{
   var msg="\nwindow.onfocus 事件 :  \n\n";
   msg+="            浏览器取得了焦点!";
   alert(msg);
}
window.onblur = function ()
{
   var msg="\nwindow.onblur 事件 :  \n\n";
   msg+="            浏览器失去了焦点!";
   alert(msg);
}
window.onscroll = function ()
{
   var msg="\nwindow.onscroll 事件 :  \n\n";
   msg+="            用户拖动了滚动条!";
   alert(msg);
}
window.onresize = function ()
{
   var msg="\nwindow.onresize 事件 :  \n\n";
   msg+="            用户改变了窗口尺寸!";
   alert(msg);
}
//-->
</script>
```

```
</head>
<body>
<br>
   <p>载入文档:</p>
   <p>取得焦点:</p>
   <p>失去焦点:</p>
   <p>拖动滚动条:</p>
   <p>变换尺寸:</p>
</body>
</html>
```

将上述源程序保存为*.html（或*.htm）文档，双击该文档后系统调用默认的浏览器进行浏览。

当载入该文档时，触发 window.load 事件，弹出警告框如图 3.2 所示。

图 3.2 载入文档时触发 window.load 事件

当把焦点给该文档页面时，触发 window.onfocus 事件，弹出警告框如图 3.3 所示。

图 3.3 文档取得焦点时触发 window.onfocus 事件

当该页面失去焦点时，触发 window.blur 事件，弹出警告框如图 3.4 所示。

图 3.4 文档失去焦点时触发 window.onblur 事件

当用户拖动滚动条时，触发 window.onscroll 事件，弹出警告框如图 3.5 所示。

图 3.5 拖动滚动条，触发 window.onscroll 事件

当用户改变文档页面大小时，触发 window.onresize 事件，弹出警告框如图 3.6 所示。

图 3.6 改变文档页面大小，触发 window.onresize 事件

浏览器事件一般用于处理窗口定位、设置定时器或者根据用户喜好设定页面层次和内容等场合，在页面的交互性、动态性方面使用较为广泛。

注意：Netscape Navigator 4 支持 window.onmove 事件，该事件在当前浏览器窗口被用户移动时触发，主要用于窗口的定位方面。Internet Explorer 不支持 window.onmove 事件。

3.2.3 HTML 元素事件

页面载入后，用户与页面的交互主要指发生在如按钮、链接、表单、图片等 HTML 元素上的用户动作以及该页面对此动作所作出的响应。如简单的鼠标单击按钮事件，元素为 button，事件为 click，事件处理器为 onclick()。只要了解了该事件的相关信息，程序员就可以编写此接口的事件处理程序，也称事件处理器，以完成诸如表单验证、文本框内容选择等功能。

HTML 文档中元素对应的事件因元素类型而异，表 3.1 按 HTML4 标记类型和字母顺序列出了当前通用版本浏览器上支持的事件。其中的标记代表标记对，如<A>代表<A>和</A>标记对，其余类似。

表 3.1 通用浏览器上定义的事件

标记类型	标记列表	事件触发模型	简要说明

链接	<A>	onclick	鼠标单击链接
		ondbclick	鼠标双击链接
		onmouseDown	鼠标在链接的位置按下
		onmMouseOut	鼠标移出链接所在的位置
		onmouseOver	鼠标经过链接所在的位置
		onmouseUp	鼠标在链接的位置放开
		onKeyDown	键被按下
		onkeyPress	按下并放开该键
		onkeyUp	键被松开
图片	<IMG>	onerror	加载图片出现错误时触发
		onload	图片加载时触发
		onkeyDown	键被按下
		onkeyPress	按下并放开该键
		onkeyUp	键被松开
区域	<AREA>	ondbClick	双击该图形映射区域
		onmouseOut	鼠标从该图形映射区域内移动到该区域之外
		onmouseOver	鼠标从该图形映射区域外移动到区域之内
文档主体	<BODY>	onblur	文档正文失去焦点
		onclick	在文档正文中单击鼠标
		ondbClick	在文档正文中双击单击鼠标
		onkeyDown	在文档正文中键被按下
		onkeyPress	在文档正文中按下并放开该键
		onkeyUp	在文档正文中键被松开
		onmouseDown	在文档正文中鼠标按下
		onmouseUp	在文档正文中鼠标松开
帧、帧组	<FRAME> <FRAMESET>	onblur	当前窗口失去焦点
		onerror	装入窗口时发生错误
		onfocus	当前窗口获得焦点
		onload	载入窗口时触发
		onresize	窗口尺寸改变
		onunload	用户关闭当前窗口
窗体	<FORM>	onreset	窗体复位
		onsubmit	提交窗体里的表单
按钮	<INPUT TYPE="button">	onblur	按钮失去焦点
		onclick	鼠标在按钮响应范围单击
		onfocus	按钮获得焦点
		onmouseDown	鼠标在按钮响应范围按下
		onmouseUp	鼠标在按钮响应范围按下后弹起
复选框 单选框	<INPUT TYPE="checkbox"> or "radio"	onblur	复选框（或单选框）失去焦点
		onclick	鼠标单击复选框（或单选框）
		onfocus	复选框（或单选框）得到焦点
复位按钮 提交按钮	<INPUT TYPE="reset"> or "submit"	onblur	复位（或确认）按钮失去焦点
		onclick	鼠标单击复位（或确认）按钮
		onfocus	复位（或确认）按钮得到焦点
口令字段	<INPUT TYPE="password">	onblur	口令字段失去当前输入焦点
		onfocus	口令字段得到当前输入焦点
文本字段	<INPUT TYPE="text">	onblur	文本框失去当前输入焦点
		onchange	文本框内容发生改变并且失去当前输入焦点
		onfocus	文本框得到当前输入焦点
		onselect	选择文本框中的文本
文本区	<TEXTAREA>	onblur	文本区失去当前输入焦点
		onchange	文本区内容发生改变并且失去当前输入焦点
		onfocus	文本区得到当前输入焦点
		onkeyDown	在文本区中键被按下
		onkeyPress	在文本区中按下并放开该键
		onkeyUp	在文本区中键被松开
		onselect	选择文本区中的文本

选项	<SELECT>	onblur	选择元素失去当前输入焦点
		onchange	选择元素内容发生改变且失去当前输入焦点
		onfocus	选择元素得到当前输入焦点

上表总结了 JavaScript 定义的通用浏览器事件，HTML 文档中事件捆绑特性决定了脚本程序员可以将这些事件当作目标的属性，在使用过程中只需修改其属性值即可。考察如下文本框各事件的测试代码：

```
//源程序 3.2
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyBlur()
{
   var msg="\n 文本框 onblur()事件 : \n\n";
   msg+="          文本框失去了当前输入焦点!";
   alert(msg);
}
function MyFocus()
{
   var msg="\n 文本框 onfocus()事件 : \n\n";
   msg+="          文本框获得了当前输入焦点!";
   alert(msg);
}
function MyChange()
{
   var msg="\n 文本框 onchange()事件 : \n\n";
   msg+="          文本框的内容发生了改变!";
   alert(msg);
}
function MySelect()
{
   var msg="\n 文本框 onselect()事件 : \n\n";
   msg+="          选择了文本框中的某段文本!";
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form name=MyForm>
   <input type=text name=MyText size=40
   value="Welcome to JavaScript world!"
   onblur="MyBlur()"
   onfocus="MyFocus()"
   onchange="MyChange()"
   onselect="MySelect()">
</form>
</center>
```

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

程序运行后，根据用户的页面动作触发不同的事件处理器（即对应的函数）。

鼠标点击文本框外的其他文档区域后，文本框失去当前输入焦点，触发 MyBlur()函数，返回警告框如图 3.7 所示。

图 3.7 文本框失去当前输入焦点

鼠标点击文本框后，文本框获得当前输入焦点，触发 MyFocus()函数，返回警告框如图 3.8 所示。

图 3.8 文本框获得当前输入焦点

修改文本框的文本后，鼠标在文本框外文档中任意位置点击，触发 MyBlur()函数的同时，触发 MyChange()函数，返回警告框如图 3.9 所示。

图 3.9 改变文本框内容并将焦点移出文本框

在文本框获得焦点后，用鼠标选择某段文本，触发函数 MySelect()函数，返回警告框如图 3.10 所示。

图 3.10 选择文本框中某段文本

HTML 元素事件在表单提交、在线办公、防止网站文章被复制、禁止下载网页中图片等方面应用十分广泛，主要是能有效识别用户的动作并做出相应的反应，如返回警告框、执行 window.close()方法关闭页面等操作。

通用浏览器上实现的诸多事件基本涵盖了页面中用户的动作，但随着 Web 技术的深入发展，出于友好、保密、版权等方面的考虑，通用浏览器上实现的事件已经不能满足 JavaScript 脚本开发人员的需求，各大浏览器厂商都更新了自己的事件模型，扩展了自身支持的事件类型，其中又以 IE 扩展的事件最为完备。

3.2.4 IE 扩展的事件

IE 浏览器从 IE4 版开始在更新其版本号的同时，逐步引入了一些事件用于捕捉更复杂的动作（如监视剪贴板等），主要用于文本、数据源操作等方面。表 3.2 列出了当前 IE 浏览器中广泛使用的一些事件：

表 3.2 IE浏览器的扩展事件

标记类型	标记列表	事件触发模型	简要说明
链接、图片、区域、地址、表单及其他文本属性标记	<A>、<ADDRESS>、<AREA>、<FORM>、<IMG>及其他属性标记*	onbeforecopy	在选择的内容复制和放在系统剪贴板之前触发
		onbeforecut	在选择的内容剪贴和放在系统剪贴板之前触发
		onbeforepaste	在选择的内容粘贴到文本之前触发
		oncopy	当选择的内容从文本复制到剪贴板时触发
		oncut	当选择的内容从文本中剪贴并添加到剪贴板时触发
		ondragstart	当用户拖动高亮选择时触发
		onerrorupdate	数据传输由onbeforeupdate事件处理器取消时触发*
		onpaste	当选择的内容粘贴到文本时触发
图片	<IMG>	onabort	由用户中断加载图片或者类似效果时触发
		onafterupdate	在数据从标记转移到数据提供者完成后触发
		onbeforeupdate	当数据从标记转移到数据提供者之前触发
		onreadystatechange	对象就绪状态发生改变时触发
		onresize	图片大小发生改变时触发
		onrowenter	指明绑定的数据行已经发生改变，新数据可用
		onrowexit	在绑定数据源控制改变当前行之前触发
		onscroll	当滚动标记重新定位时触发
Applet 文档主体	<APPLET>、<BODY>	onafterupdate	在数据从标记转移到数据提供者完成后触发
		onbeforecut	在选择的内容剪贴和放在系统剪贴板之前触发
		onbeforepaste	在选择的内容粘贴到文本之前触发
		onbeforeupdate	在数据从标记转移到数据提供者之前触发
		oncut	当选择的内容从文本中剪贴并添加到剪贴板时触发
		ondragstart	当用户拖动高亮的选择时触发
		onpaste	当选择的内容粘贴到文本时触发

		onrowenter	指明绑定的数据行已经发生改变，新数据可用
		onrowexit	在绑定数据源控制改变当前行之前触发
		onscroll	当滚动标记重新定位时触发
Applet	<APPLET>、<OBJECT>	onafterupdate	在数据从标记转移到数据提供者完成后触发
		onbeforeupdate	当数据从标记转移到数据提供者之前触发
		ondataavailable	当数据从不同步传输数据的数据源到达时触发
		ondatasetchanged	当来自数据源初始数据可用或发生改变时触发
		ondatasetcomplete	指明数据源中所有数据为可用
		ondragstart	当用户拖动高亮的选择时触发
		onerrorupdate	数据传输由onbeforeupdate事件处理器取消时触发
		onreadystatechange	对象就绪状态发生改变时触发
		onresize	对象大小发生改变时触发
		onrowenter	指明绑定的数据行已经发生改变，新数据可用
		onrowexit	在绑定数据源控制改变当前行之前触发
文档主体帧组	<BODY>、<FRAMESET>	onafterprint	在用户打印当前或先前的文本时触发
		onbeforeprint	在用户打印当前或先前的文本之前触发
		onbeforeunload	在文本从窗口卸载之前触发
		ondragdrop	用户将一个对象拖动到浏览器窗口试图加载时触发
		ondragstart	当用户拖动高亮的选择时触发
		onerror	当文本的加载或者脚本执行出现错误时触发
		onmove	当用户移动窗口时触发
		onreadystatechange	对象就绪状态发生改变时触发
		onresize	对象大小发生改变时触发
按钮	Button	onafterupdate	在数据从标记转移到数据提供者完成后触发
		onbeforecut	在选择的内容剪贴和放在系统剪贴板之前触发
		onbeforepaste	在选择的内容粘贴到文本之前触发
		onbeforeupdate	当数据从标记转移到数据提供者之前触发
		oncut	当选择的内容从文本中剪贴并添加到剪贴板时触发
		ondragstart	当用户拖动高亮的选择时触发
		onpaste	当选择的内容粘贴到文本时触发
		onresize	对象大小发生改变时触发
		onrowenter	指明绑定的数据行已经发生改变，新数据可用
		onrowexit	在绑定数据源控制改变当前行之前触发
文本区	<TEXTAREA>	onafterupdate	在数据从标记转移到数据提供者完成后触发
		onbeforeupdate	当数据从标记转移到数据提供者之前触发
		onerrorupdate	数据传输由onbeforeupdate事件处理器取消时触发*
		onrowenter	指明绑定的数据行已经发生改变，新数据可用
		onrowexit	在绑定数据源控制改变当前行之前触发
		onscroll	当滚动标记重新定位时触发
选项	<SELECT>	onafterupdate	在数据从标记转移到数据提供者完成后触发
		onbeforeupdate	当数据从标记转移到数据提供者之前触发
		ondragstart	当用户拖动高亮选择时触发
		onerrorupdate	数据传输由onbeforeupdate事件处理器取消时触发*
		onresize	对象大小发生改变时触发
		onrowenter	指明绑定的数据行已经发生改变，新数据可用
		onrowexit	在绑定数据源控制改变当前行之前触发

注意：（1）本表的*号依次表示：其他属性标记为<b>、<big>、<blockquote>、<caption>、 、<cite>、<code>、<dd>、<dfn>、<dir>、 、<dl>、<dt>、、<h1>--<h6>、<i>、<li>、<listing>、<menu>、<ol>、<p>、<plaintext>、<pre>、<s>、<samp>、<small>、、<strike>、、<sub>、<sup>、<td>、<textarea>、<th>、<tr>、<tt>、<u>、<ul>等；<AREA>、<ADDRESS>、<IMG>等文本属性标记无 onerrorupdate 事件。（2）如出现一个标记两个项目的情况，为两个项目共有的事件。（3）上述事件大部分在 IE4 中获得支持，在 IE5、IE5.5 中逐步完善，IE6+得到完美地支持。此 IE 包括以 IE 为内核的浏览器，不单纯指 Microsoft Internet Explorer。

表 3.2 中列举的事件为 HTML4 规则定义之外、Microsoft 为扩展其 IE 浏览器中动作捕

获能力而增加的 HTML 元素事件。由于 IE（包括 IE 核心）浏览器有着无与伦比的技术优势和覆盖范围，在编制 JavaScript 脚本程序的过程中上述事件原型得到了广泛的应用。

3.3 JavaScript 如何处理事件

尽管 HTML 事件属性可以将事件处理器绑定为文本的一部分，但其代码一般较为短小，功能较弱，只适用于只需做简单的数据验证、返回相关提示信息等场合。相比较对而言，使用 JavaScript 脚本可以更为方便处理各种事件，特别是 Internet Explorer、Netscape Navigator 等浏览器厂商在其第 4 代浏览器中推出更为先进的事件模型后，使用 JavaScript 脚本处理事件显得顺理成章。

JavaScript 脚本处理事件主要可通过匿名函数、显式声明、手工触发等方式进行，这几种方法在隔离 HTML 文本结构与逻辑关系的程度方面略为不同。

3.3.1 匿名函数

匿名函数的方式即使用 Function 对象（第 6 章详细叙述）构造匿名的函数，并将其方法复制给事件，此时该匿名的函数成为该事件的事件处理器。考察如下的代码：

```
//源程序 3.3
<! DOCTYPE HTML PUBLIC    "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
</head>
<body>
<center>
<br>
<p>单击“事件测试”按钮，通过匿名函数处理事件</p>
<form name=MyForm id=MyForm>
   <input type=button name=MyButton id=MyButton value="事件测试">
</form>
<script language="JavaScript" type="text/javascript">
<!--
document.MyForm.MyButton.onclick=new Function()
{
    alert("Your Have clicked me!");
}
-->
</script>
</center>
</body>
</html>
```

程序运行结果如图 3.11 所示。

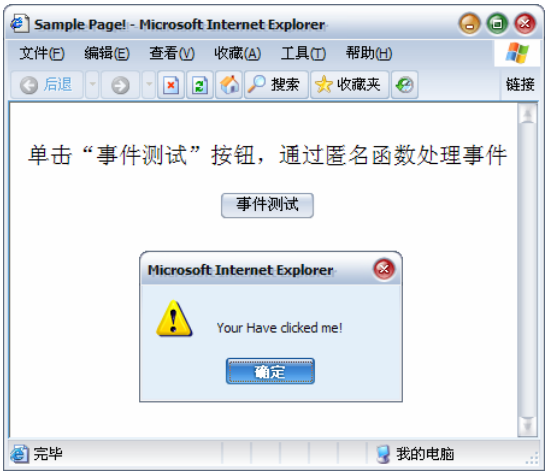

图 3.11 通过匿名函数处理事件

关键代码：

```
document.MyForm.MyButton.onclick=new Function()
{
    alert("Your Have clicked me!");
}
```

此句将名为 MyButton 的 button 元素的 click 动作的事件处理器设置为新生成的 Function 对象的匿名实例，即该匿名函数。鼠标单击该按钮后，响应“单击”事件，返回警告框。

3.3.2 显式声明

当然，设置事件处理器时，也可不使用以上匿名函数，而是将该事件的处理器设置为已经存在的函数。

考察如下图片翻转的实例：

```
//源程序 3.4
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyImageA()
{
    document.all.MyPic.src="2.jpg";
}
function MyImageB()
{
    document.all.MyPic.src="1.jpg";
}
-->
</script>
</head>
<body>
```

```
<center>
<p>在图片内外移动鼠标，图片轮换</p>
<img name="MyPic" id="MyPic" src="1.jpg" width=300 height=200></img>
<script language="JavaScript" type="text/javascript">
<!--
document.all.MyPic.onmouseover=MyImageA;
document.all.MyPic.onmouseout=MyImageB;
-->
</script>
</center>
</body>
</html>
```

程序运行结果如图 3.12 所示。

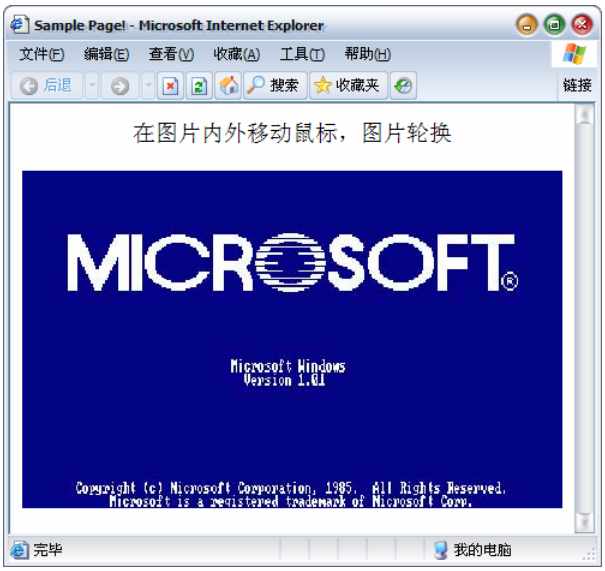

图 3.12 文档载入或鼠标移离图片时，显示默认图片“1.jpg”

当鼠标移动到图片区域时，图片发生变化，即显示图片“2.jpg”，如图 3.13 所示。

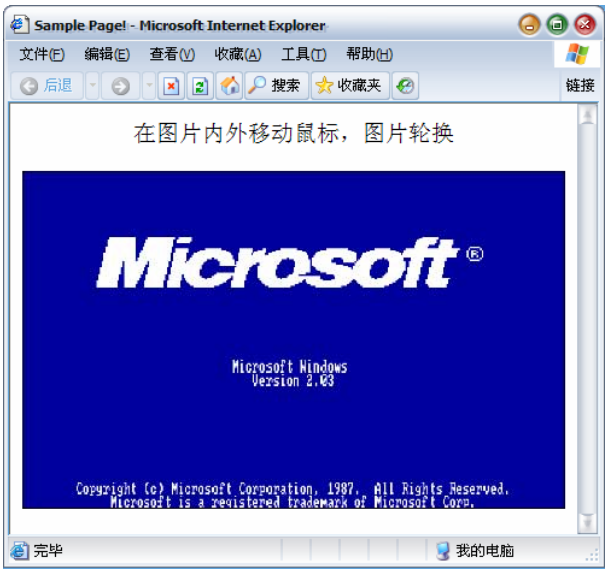

图 3.13 当鼠标移动到图片区域时显示图片“2.jpg”

当鼠标移离图片区域时，显示默认图片“1.jpg”，实现了图片的翻转。可将此法扩展为多幅新闻图片定时轮流播放的广告模式。

首先在<head>和</head>标记对之间嵌套 JavaScript 脚本定义两个函数：

```
function MyImageA()
{
    …
}
function MyImageB()
{
    …
}
```

然后通过 JavaScript 脚本代码将<IMG>标记元素的 mouseover 事件的处理器设置为已定义的函数 MyImageA()，将其 mouseout 事件的处理器设置为已定义的函数 MyImageB()：

```
document.all.MyPic.onmouseover=MyImageA;
document.all.MyPic.onmouseout=MyImageB;
```

由以上调用过程可以看出，通过显式声明的方式定义事件的处理器代码紧凑、可读性强，且对显式声明的函数没有任何限制，还可将该函数作为其他事件的处理器。较之匿名函数的方式实用。

3.3.3 手工触发

手工触发事件的原理相当简单，就是通过其他元素的方法来触发一个事件而不需要通过用户的动作来触发该事件。在源程序 3.4 的</script>和 标记之间插入如下代码：

```
<form name="MyForm" id="MyForm">
  <input type=button name=MyButton id=MyButton value="测试"
  onclick="document.all.MyPic.onmouseover();"
  onblur="document.all.MyPic.onmouseout();">
<form>
```

保存文件，程序运行后显示默认图片“1.pg”；单击“测试”按钮，将显示图片“2.jpg”，如图 3.13 所示；按钮失去焦点后，图片发生变化，显示图片“1.pg”，如图 3.12 所示。

如果某个对象的事件有其默认的处理器，此时再设置该事件的处理器时，将有可能出现意外的情况出现。考察如下的代码：

```
//源程序 3.5
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
  var msg="默认提交与其他提交方式返回不同的结果：\n\n";
  msg+="          点击"测试"按钮,直接提交表单.\n";
  msg+="          点击"确认"按钮,触发 onsubmit()方法,然后提交表单.\n";
  alert(msg);
```

```
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm1 id=MyForm1 onsubmit="MyTest()" method=post action="target.asp">
   <input type=button value="测试" onclick="document.all.MyForm1.submit();">
   <input type=submit value="确认">
</center>
</body>
</html>
```

程序运行后，单击“测试”按钮，触发表单的提交事件，并直接将表单提交给目标页面“target.asp”；单击表单默认触发提交事件的“确认”按钮，将弹出如图 3.14 所示的警告框，单击“确认”按钮后，将表单提交给目标页面“target.asp”。

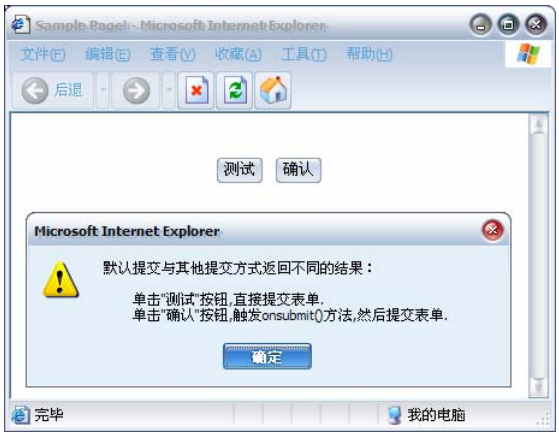

图 3.14 单击“确认”按钮后弹出警告框

由上面提交表单的例子可以看出，当事件在事实上已包含导致事件发生的方法时，该方法不会调用有问题的事件处理器，而会导致与该方法对应的行为立即发生。

> 注意：使用 JavaScript 脚本设置事件处理器时要分外小心，因为 JavaScript 事件处理器是大小写敏感的。设置目标对象中并不存在的事件的处理器将会给对象添加一个新的属性，而调用目标对象中并不存在的属性一般将导致页面运行错误。

3.4 事件处理器的返回值

事件通过给发送消息的方式来触发事件处理器对用户的动作做出相关响应来达到交互的目的，但此交互一般只是单方面的交互，即事件发送消息给事件处理器的过程，而不包括事件处理器将处理结果返回给事件的过程。事实上，事件处理器能将结果返回给事件，并由此影响事件的默认行为。考察如下代码：

```
//源程序 3.6
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
function ch_input()
{
   var msg="\n 系统提示信息 : \n\n";
   if(document.all.MyForm.name.value=='')
   {
      msg+="            您输入的用户名为空,请重新填写并确认! \n";
      alert(msg);
      document.all.MyForm.name.focus();
      return false;
   }
   if(document.all.MyForm.password.value=='')
   {
      msg+="            您输入的密码为空,请重新填写并确认! \n";
      alert(msg);
      document.all.MyForm.password.focus()
      return false;
   }
}
</script>
</head>
<body>
<center>
<form name=MyForm method="post" action="target.asp"    onsubmit=return(ch_input())>
   用户:<input name="name" size="18" ><br>
   密码:<input type="password" name="password" size="19"><br>
   <input type="submit" value="提交" name="B1" >
   <input type="reset" value="重置" name="B2" ></p></form>
</center>
</body>
</html>
```

程序运行后，若表单的“用户”字段为空，鼠标单击“提交”按钮后，将弹出“错误”警告框如图 3.15 所示。

图 3.15 “用户”字段为空时弹出“错误”警告框

在该警告框中单击“确定”按钮，浏览器返回原始页面，并将“用户”字段设置为当前的输入焦点。

若表单的“用户”字段非空，则继续判断“密码”字段。同样，若表单的“密码”字段为空，鼠标单击“提交”按钮后，将弹出“错误”警告框如图 3.16 所示。

图 3.16 “密码”字段为空时弹出“错误”警告框

在该警告框中单击“确定”按钮，浏览器返回原始页面，并将“密码用户”字段设置为当前的输入焦点。

该段代码中，当客户端单击页面默认的表单 submit 按钮即“提交”按钮后，使用下列语句触发表单 submit 事件的处理器：

```
onsubmit=return(ch_input())
```

事件处理器所对应的函数 ch_input()根据页面表单各字段的情况判断：

```
if(document.all.MyForm.name.value=='')
{
   msg+='            您输入的用户名为空,请重新填写并确认! \n';
   alert(msg);
   document.all.MyForm.name.focus();
   return false;
}
```

判断各字段时，如果该字段为空，则弹出警告框，用户单击“确认”按钮后，浏览器返回设置该字段为当前的输入焦点，并将布尔值 false 作为结果返回给 submit 事件；如果各字段都不为空，则将布尔值 true 作为结果返回给 submit 事件。

submit 事件接受返回的结果，如果返回的结果为 true，浏览器将页面跳转到目标页面（本例中为 target.asp）；如果返回的结果为 false，则浏览器取消加载目标页面的动作，而继续将浏览器焦点定位在原始页面中。

在浏览器事件和 HTML 元素事件中，事件处理器的返回值将直接影响其下一步的动作。现将常用的事件返回值与其导致的行为之间关系列表如下：

表 3.3 常见事件与其返回值之间的关系

事件	返回值	对象	简要说明
Click	False	submit	取消表单提交
		reset	不重置表单
		radio、checkbox	单选框、复选框未被选择
		link	浏览器不执行页面跳转操作
DragDrop*	false	body、frameset	取消对象拖放

KeyDown	False	applet、font等	取消随后的KeyPress事件
KeyPress	False	applet、font等	取消KeyPress事件
MouseDown	False	applet、font等	取消鼠标的默认操作，如link的开始链接等
MouseOver	true	applet、font等	将窗口的status属性的变化在浏览器中表现出来
MouseOver	False	applet、font等	忽略窗口的status属性变化，直到MouseOut事件出现
Submit	False	submit	取消表单提交

注意：表 3.3 列举的是常见事件接受事件处理器返回非默认值时发生的动作，各个事件对应的对象请参见表 3.1 和表 3.2。

在实际应用中，经常在表单提交给服务器之前对其进行确认以检查通常的拼写错误或者非法的数据，极大减轻了服务器负担并充分保证其安全性，同时通过检查，提高用户提交数据的准确性。

3.5 事件处理器设置的灵活性

由于 HTML 将事件看成对象的属性，可以通过给该属性赋值的方式来改变事件的处理器，这给使用 JavaScript 脚本来设置事件处理器带来了很大的灵活性。考察如下的实例：

```
//源程序 3.7
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//设置事件处理器 MyHandlerA
function MyHandlerA()
{
   var msg="提示信息 : \n\n";
   msg+="          1、触发该按钮的 Click 事件的处理器 MyHandlerA()!\n";
   msg+="          2、改变该按钮的 Click 事件的处理器为 MyHandlerB()!\n";
   alert(msg);
   //修改按钮 value 属性
   document.all.MyForm.MyButton.value="测试按钮 ： 触发事件处理器 B";
   //修改按钮的 Click 事件的处理器为 MyHandlerB
   document.all.MyForm.MyButton.onclick=MyHandlerB;
}
//设置事件处理器 MyHandlerB
function MyHandlerB()
{
   var msg="提示信息 : \n\n";
   msg+="          1、触发该按钮的 Click 事件的处理器 MyHandlerB()!\n";
   msg+="          2、改变该按钮的 Click 事件的处理器为 MyHandlerA()!\n";
   alert(msg);
   document.all.MyForm.MyButton.value="测试按钮 ： 触发事件处理器 A";
   document.all.MyForm.MyButton.onclick=MyHandlerA;
}
-->
</script>
```

```
</head>
<body>
<center>
<form name="MyForm">
   <input type="button" name="MyButton" id="MyButton" value="测试按钮 ： 触发事件处理器 A">
   <br>
</form>
</center>
<script language="JavaScript" type="text/javascript">
<!--
//设置按钮的 Click 事件的初始处理器为 MyHandlerA
document.all.MyForm.MyButton.onclick=MyHandlerA;
-->
</script>
</body>
</html>
```

程序运行后，单击“测试按钮 ： 触发事件处理器 A”按钮，弹出警告框如图 3.17 所示。

图 3.17 第一次单击按钮，弹出警告框

在上述警告框中单击“确定”按钮后，返回原始页面，更改按钮的 value 属性为“测试按钮 ： 触发事件处理器 B”。继续单击该按钮后，弹出警告框如图 3.18 所示。

图 3.18 第二次单击按钮，弹出警告框

在上述警告框中单击“确定”按钮后，返回原始页面，更改按钮的 value 属性为“测试按钮 ： 触发事件处理器 B”，继续操作可以发现过程循环。

由程序结果可见，主要过程分为 4 步：

（1）文档载入后，通过属性赋值的方式将按钮的 Click 事件默认的事件处理器设置为 MyHandlerA：

```
document.all.MyForm.MyButton.onclick=MyHandlerA;
```

（2）单击按钮后，触发 Click 事件当前的事件处理器 MyhandlerA，后者返回提示信息并将按钮的 value 属性更改，同时将其 Click 事件当前的事件处理器设置为 MyhandlerB：

```
document.all.MyForm.MyButton.value="测试按钮 ： 触发事件处理器 A";
document.all.MyForm.MyButton.onclick=MyHandlerA;
```

（3）在提示页面单击“确定”按钮返回原始页面后，再次单击按钮，触发 Click 事件当前的事件处理器 MyhandlerB，后者返回提示信息并将按钮的 value 属性更改，同时将其 Click 事件当前的事件处理器设置为 MyhandlerA：

```
document.all.MyForm.MyButton.value="测试按钮 ： 触发事件处理器 B";
document.all.MyForm.MyButton.onclick=MyHandlerB;
```

（4）在提示页面单击“确定”按钮返回原始页面后，返回步骤（2）。

在 JavaScript 脚本中根据复杂的客户端环境及时更改事件的处理器，可大大提高了页面的交互能力。

值得注意的是，给对象的事件属性赋值为事件处理函数时，后者要省略函数后面的括号，且对象和函数要在显式赋值语句之前定义。

3.6 现代事件模型与 Event 对象

在 Internet Explorer 4（IE4，下同）和 Netscape Navigator 4（NN4，下同）浏览器版本发布之前，事件一直遵循基本事件模型标准，但后者存在着如下的诸多缺陷：

- 基本事件模型只支持最基本的事件，事件的类型和数目很有限，同时处理事件的方式也非常有局限性；
- 在基本事件模型中，某事件发生后触发事件处理器进行相关操作的过程中缺少必要的一些信息，如事件发生的位置、发生事件的对象等，事件处理器只能很被动地响应事件；
- 在基本事件模型中，没有任何途径解决对象继承关系中不同层次事件处理器之间的交互，对象的继承关系并没有反映到事件处理器的相互关系上。

基于此，在第 4 代浏览器版本中，各大浏览器厂商都扩展了其基本事件模型，添加了新的事件及处理方式。它们与基本事件模型之间最大的区别在于新的事件模型增加了 Event 对象，该对象将事件发生的快照内容如事件发生的位置、触发的按键等传递给事件处理器，事件处理器根据事件提供的这些信息进行相应的操作，从而极大扩展了 JavaScript 脚本的功能。

同时，新的事件模型中的事件可通过文本的继承关系进行事件传播，但各种浏览器中定义的事件流又不大不同，如 IE4 和 NN4 事件流的方向就完全相反：

- 在 IE4 中多数事件是由它们发生的地方开始向上回溯继承关系，上溯的事件在继承关系的每层都用合适的事件处理器来处理、改向或者将事件沿着关系树传递，当然，某些定义清晰的事件如表单提交、按钮接收焦点等不会沿着继承关系上溯；
- 在 NN4 中事件从顶端对象开始至末端对象结束，并使得这些对象能够更改、处理或取消该事件，在较高层次上的事件处理器比低层次的事件处理器有更多机会处理事件。

图 3.19 显示了 IE4 和 NN4 新事件模型中事件流的不同：

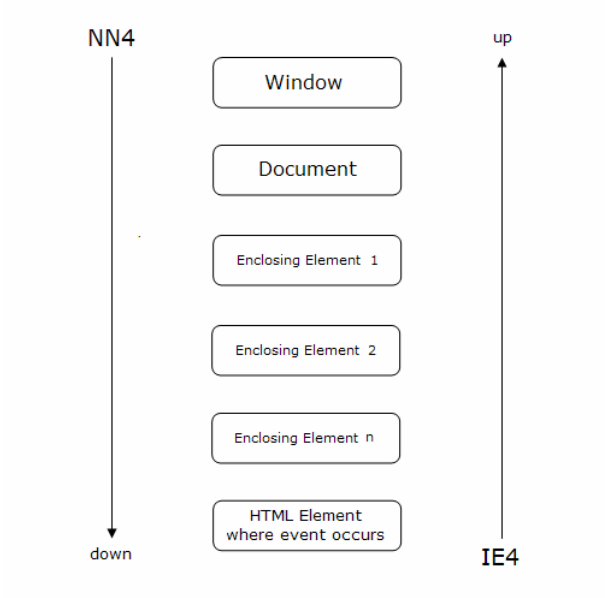

图 3.19 IE4 和 NN4 中新事件模型的事件传播方向

除了事件传播方向上的不同外，IE4 和 NN4 中定义的 Event 对象也有很大的差异，主要表现在对象实例的传递、作用范围等方面。

JavaScript1.2 版本中引入 Event 对象作为事件与事件处理器之间传递信息的桥梁，这些信息如事件发生的位置、触发事件的原始对象等通过 Event 对象的属性提供。IE4 和 NN4 通过不同途径提供 Event 对象的诸多属性供脚本开发人员调用。

3.7 IE4 中的 Event 对象

由于 IE4 中文档的每个元素都作为一个对象而存在，增加了事件发生的概率和可用的事件数目。当事件发生的时候，浏览器创建全局的 Event 对象，并使它对于合适的事件处理器可用。

3.7.1 对象属性

IE4 中的 Event 对象提供非常丰富的属性供脚本程序员调用，如事件发生的原始对象、相对位置等，其常见属性及其功能介绍如表 3.4 所示。

表 3.4 IE4 中Event对象的常见属性

属性	简要说明
altkey、crlkey、shiftkey	设置为true或者false，表示事件发生时是否按下Alt、Ctrl和Shift键
buttoon	发生事件时鼠标所按下的键（0表示无，1表示左键，2表示右键，4表示中键）
cancelBubble	设置为true或者false，表示是取消还是启用事件上溯
clientX、clientY	光标相对于事件所在Web页面的水平和垂直位置（像素）
fromElement、toElement	指向在MouseOver或MouseOut中鼠标要移开和移向的标记
keyCode	表示所按键的Unicode键盘内码
offsetX、offsetY	光标相对于事件所在容器的水平和垂直位置（像素）
reason	表示数据源对象的数据传输状态
returnValue	设置为true或false，表示事件处理器的返回值

screenX、screenY	光标相对于屏幕的水平和垂直位置（像素）
srcElement	表示发生事件的原始对象
srcFilter	指定产生onfilterchange事件的filter对象
type	指明发生事件的类型
x、y	光标相对于事件所在文档的水平和垂直位置（像素）

理解了事件与其处理器之间交互时 Event 对象携带的信息，以及其全局特性，很容易通过调用对象属性的方式根据需要获取事件发生的诸多信息。考察如下获取事件发生相对位置等信息的代码：

```
//源程序 3.8
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="";
msg+="Click 事件发生后创建的 Event 对象信息:\n\n";
function MyAlert()
{
  msg+="发生事件的类型 :\n";
  msg+="          type = " +event.type+ "\n\n";
  msg+="发生事件的原始对象 :\n";
  msg+="          target = " +event.srcElement+ "\n\n";
  msg+="光标相对于事件所在文档的水平和垂直位置(像素) :\n";
  msg+="          x = " +event.x+ "          y = " +event.y+ "\n\n";
  msg+="光标相对于事件所在容器的水平和垂直位置(像素) :\n";
  msg+="          x = " +event.offsetX+ "          y = " +event.offsetY+ "\n\n";
  msg+="光标相对于事件所在屏幕的水平和垂直位置(像素) :\n";
  msg+="          x = " +event.screenX+ "          y = " +event.screenY+ "\n\n";
  msg+="光标相对于事件所在 Web 页面的水平和垂直位置(像素) :          \n";
  msg+="          x = " +event.clientX+ "          y = " +event.clientY+ "\n";
  alert(msg);
}
-->
</script>
</head>
<body onclick="MyAlert();">
  <table>
    <tr>
      <td>
        <p>
          Click the
          <em>
            EM text
          </em>
          to Test!
        </p>
      <td>
    <tr>
  </table>
</body>
```

```
</html>
```

程序运行后，鼠标单击文档中文本段的任何一个位置，弹出包含当前发生事件信息的警告框如图 3.20 所示。

图 3.20 实例中 Click 事件的相关信息警告框

在 IE4 中，任何事件发生后生成的 Event 对象对该文档而言是透明的，可将其看成是全局变量使用，即在文档任何地方都不需通过传入参数的方式来调用。该变量生成于其对应的事件发生之时，且在文档被浏览器关闭时对象所占据的内存空间被释放出来。

3.7.2 事件上溯

IE4 中的大部分事件会沿着关系树上溯（Bubble，也称冒泡），在继承关系的每一层如果存在合适的事件处理器则调用，不存在则继续上溯，直至上升到关系树的顶端或者被某个层所取消，但不支持上溯的事件仅能调用当前事件发生的原始对象那个层次上可用的事件处理器。表 3.5 列出了 IE4 中常用的事件，以及该事件能否上溯、能否取消等特性。

表 3.5 IE4 中常用的事件及回溯、取消情况

事件	能否上溯	能否取消	事件	能否上溯	能否取消
abort	否	是	keypress	是	是
afterupdate	是	否	keyup	是	否
beforeunload	否	是	load	否	否
beforeupdate	是	是	mousedown	是	是
blur	否	否	mousemove	是	否
bounce	否	是	mouseout	是	否
change	否	是	mouseover	是	是
click	是	是	mouseup	是	是
dataavailable	是	否	readystatechange	否	否
datasetchanged	是	否	reset	否	是
datasetcomplete	是	否	resize	否	否
dbclick	是	是	rowenter	是	否
dragstart	是	是	rowexit	否	是
error	否	是	scroll	否	否
errorupdate	是	否	select	否	是
filterchange	否	否	selectstart	是	是
finish	否	是	start	否	否
focus	否	否	submit	否	是
help	是	是	unload	否	否
keydown	是	是			

注意：在上表中，事件对应的处理器一般为事件首字母改写为小写，然后前面加上“on”即可，如 Dbclick 事件其对应的事件处理器为 ondbclick。文档中各元素支持何种事件，请参阅表 3.1 和表 3.2。

某事件不能上溯，表明该事件只对当前发生事件的对象有效；某事件能取消表明可以通过设置其对应 Event 对象的 cancelBuuble 属性为 true 来阻止事件上溯到其上层对象。

考察如下演示 IE4（IE4 以上版本都适用）中事件上溯的代码：

```
//源程序 3.9
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="";
msg+="事件上溯结果:\n\n";
msg+="Click 事件开始:\n\n";
function MyAlert()
{
   msg+="Click 事件结束:\n";
   alert(msg);
}
-->
</script>
</head>
<body onclick="javascript:msg+='-->事件定位于 Body,转向下面事件\n\n';MyAlert();">
<br>
<center>
   <table onclick="javascript:msg+='-->事件定位于 Table,转向下面事件\n\n'">
      <tr onclick="javascript:msg+='-->事件定位于 Tr,转向下面事件\n\n'">
         <td onclick="javascript:msg+='-->事件定位于 Td,转向下面事件\n\n'">
            <p onclick="javascript:msg+='-->事件定位于 p,转向下面事件\n\n'">
               Click the
               //事件发生的原始对象
               <em onclick="javascript:msg+='-->事件定位于 em,转向下面事件\n\n'">
                  EM text
               </em>
               to Test!
            </p>
         <td>
      <tr>
   </table>
</center>
</body>
</html>
```

程序运行后，鼠标点击页面中使用和标记对的“EM text”字符串，弹出对话框如图 3.21 所示。

图 3.21 IE4 中事件的回溯方向实例

可以清楚看出 Click 事件由标记处开始触发，然后按标记的层次顺序上溯至<p>、<td>、<tr>、<table>和<body>，直至 Document 对象，但并不上溯到 Window 对象。

3.7.3 阻止事件上溯

IE4 中的事件严格按照文档中元素对象的层次关系上溯而不管实际应用中是否需要将该事件上溯到对象关系中特定的层次。为达到在对象关系中指定层次中断事件上溯的目的，程序员可设置 Event 对象的 cancleBubble 属性为 true 来实现。

考察如下代码：

```
//源程序 3.10
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="";
msg+="阻止事件上溯结果:\n\n";
msg+="Click 事件开始:\n\n";
function MyAlert()
{
   msg+="Click 事件结束:\n";
   alert(msg);
}
-->
</script>
</head>
<body onclick="javascript:msg+='-->事件定位于 Body,转向下面事件\n\n'">
```

```
<center>
<table onclick="javascript:msg+='-->事件定位于 Table,转向下面事件\n\n'">
  <tr onclick="javascript:msg+='-->事件定位于 Tr,事件停止上溯\n\n';event.cancelBubble=true;">
    <td onclick="javascript:msg+='-->事件定位于 Td,转向下面事件\n\n'">
      <p onclick="javascript:msg+='-->事件定位于 p,转向下面事件\n\n'">
        Click the
        <em onclick="javascript:msg+='-->事件定位于 em,转向下面事件\n\n'">
          EM text
        </em>
        to Test!
      </p>
    <td>
  <tr>
</table>
<form>
  测试方法:<br>
      1、点击文本中斜体字符串;<br>
      2、鼠标单击"测试"按钮.<br>
  <input type=button value="测试" onclick="MyAlert()">
</form>
<center>
</body>
</html>
```

程序运行后，单击页面中的“EM text”文本段，然后单击“测试”按钮，弹出对话框如图 3.22 所示。

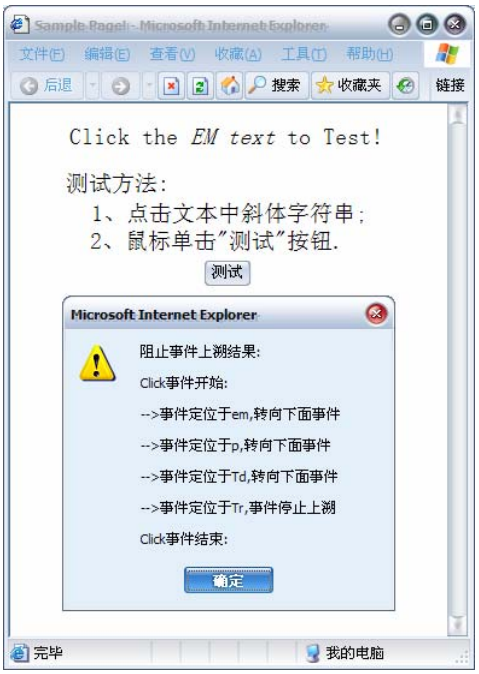

图 3.22 使用 Event 对象的 cancelBubble 属性阻止事件上溯

可以看出，Click 事件在由、<p>、<td>标记对象上溯到<tr>后，上溯过程中止，而不像源程序 3.9 的运行结果那样，事件从标记对象开始上溯到<body>为止。

关键语句：

```
<tr onclick="javascript:msg+='-->事件定位于 Tr,事件停止上溯\n\n';event.cancelBubble=true;">
```

当 Click 事件上溯到<tr>标记后，执行当前的事件处理器 onclick()，后者首先更新输出信息，然后设置 Event 对象的 cancelBubble 属性为 true：

```
event.cancelBubble=true;
```

程序结果显示，在对象模型某层次设置 Event 对象的 cancelBubble 属性后，事件的上溯过程被中断，达到阻止事件上溯的目的。

3.7.4 事件改向

IE4 中事件严格按照文档中元素对象的层次关系上溯的特性，事件流的方向人为很难掌握，给程序员精确控制事件的流向带来不小的难度。IE5.5 引进了了对象的 fireEvent()方法，该方法可以将当前事件传递到某个特定的对象上。语法如下：

```
object.fireEvent(arg1,arg2);
```

该方法需要给定两个参数 arg1 和 arg2，其中参数 arg1 表示目标对象的事件处理器，参数 arg2 表示当前事件。

考察如下演示事件转向的代码：

```
//源程序 3.11
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="";
msg+="事件转向实例:\n\n";
msg+="Click 事件开始:\n\n";
function ChangeDir()
{
   msg+="-->事件定位于 Td,准备事件转向\n\n";
   event.cancelBubble=true;
   document.body.fireEvent("onclick",event);
}
function MyAlert()
{
   msg+="Click 事件结束:\n";
   alert(msg);
}
-->
</script>
</head>
<body onclick="javascript:msg+='-->事件定位于 Body,转向下面事件\n\n'">
<center>
<table onclick="javascript:msg+='-->事件定位于 Table,转向下面事件\n\n'">
   <tr onclick="javascript:msg+='-->事件定位于 Tr,转向下面事件\n\n'">
      <td onclick="ChangeDir()">
         <p onclick="javascript:msg+='-->事件定位于 p,转向下面事件\n\n'">
            Click the
            <em onclick="javascript:msg+='-->事件定位于 em,转向下面事件\n\n'">
               EM text
```

```
            </em>
            to Test!
          </p>
        <td>
    <tr>
</table>
<form>
    测试方法:<br>
        1、点击文本中斜体字符串;<br>
        2、鼠标单击"测试"按钮.<br>
    <input type=button value="测试" onclick="MyAlert()">
</form>
<center>
</body>
</html>
```

程序运行后，点击页面中的“EM text”文本段，然后单击“测试”按钮，弹出对话框如图 3.23 所示。

图 3.23 IE4 中事件改向实例

由程序运行结果可看出，Click 事件由、<p>标记上溯到<td>标记之后，直接改向到<body>标记。关键代码：

```
<td onclick="ChangeDir()">
```

此句将<td>标记的 Click 事件的处理器设置为事件处理器函数 ChangeDir()：

```
function ChangeDir()
{
    msg+="-->事件定位于 Td,准备事件转向\n\n";
    event.cancelBubble=true;
    document.body.fireEvent("onclick",event);
}
```

该函数首先通过设置 Event 对象的 cancelBubble 属性为 true 将 Click 事件的上溯过程中止，同时通过<body>标记的 fireEvent()方法将 Click 事件改向自身，并触发<body>标记的

onclick 事件处理器更新全局变量 msg。单击“测试”按钮，输出相关提示信息。

IE4 中的事件模型为很经典的模型架构，通过其提供的诸多属性，脚本程序员很容易把握其事件的触发机制，编制出高质量的事件控制脚本。

> 注意：事件上溯、阻止事件上溯、事件改向等特性在目前的 IE 浏览器（包括以其为核心的浏览器）版本中仍然适用。

3.8 NN4 中的 Event 对象

与 IE4 相同，NN4 中发生事件时，浏览器创建 Event 对象，并将其传递给事件处理器使用，但两者创建的 Event 对象在很多方面存在很大的差异，如对象实例的传递、作用范围、事件的流向等。

3.8.1 对象属性

较之 IE4 而言，NN4 中的 Event 对象提供较为简单的属性供程序员使用，如事件发生的原始对象、相对位置等，其常见属性及其功能介绍如表 3.6 所示。

表 3.6 NN4 中Event对象的常见属性

属性	简要说明
data	包含DragDrop事件放置的对象URL字符串的数组
height、width	窗口（或帧）的高度和宽度
layerX、layerY	光标相对于事件所在层的水平和垂直位置（像素）
modifiers	与鼠标或按键相关联的组合键（ALT_MASK,CONTROL_MASK,META_MASK, SHIFT_MASK等），其中META_MASK为Macintosh机上的Command键，下同
pageX、pageY	光标相对于页面的水平和垂直位置（像素）
target	发生事件的原始对象
type	指明发生事件的类型
which	发生事件时鼠标所按下的键（1表示左，2表示中，3表示右）或所按键的ASCII值

NN4 中创建的 Event 对象并不像 IE4 中一样在文档整个范围内透明，可以当成全局变量直接调用，在触发事件处理器的过程中不需要将 Event 对象显式传递给事件处理器。在 NN4 中创建的 Event 对象只在其对应的事件处理器内可见，必须将其显式传递给事件处理器。考察如下代码：

```
//源程序 3.12
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="";
msg+="Click 事件发生后创建的 Event 对象信息:\n\n";
function MyAlert(event)
{
   msg+="发生事件的类型 :\n";
   msg+="            type = " +event.type+ "\n\n";
```

```
    msg+="发生事件的原始对象  :\n";
    msg+="              target = " +event.target+ "\n\n";
    msg+="发生事件时鼠标所按下的键(1:左,2:中,3:右) :\n";
    msg+="              which = " +event.which+ "\n\n";
    msg+="光标相对于事件所在文档的水平和垂直位置(像素) :\n";
    msg+="              x = " +event.layerX+ "              y = " +event.layerY+ "\n\n";
    msg+="光标相对于页面的水平和垂直位置(像素) :\n";
    msg+="              x = " +event.pageX+ "              y = " +event.pageY+ "\n\n";
    alert(msg);
}
-->
</script>
</head>
<body onclick="MyAlert(event);">
    <table>
        <tr>
            <td>
                <p>
                    Click the
                    <em>
                        EM text
                    </em>
                    to Test!
                </p>
            <td>
        <tr>
    </table>
</body>
</html>
```

程序运行后，鼠标单击文档中文本段的任何一个位置，弹出包含当前发生事件信息的警告框如图 3.24 所示。

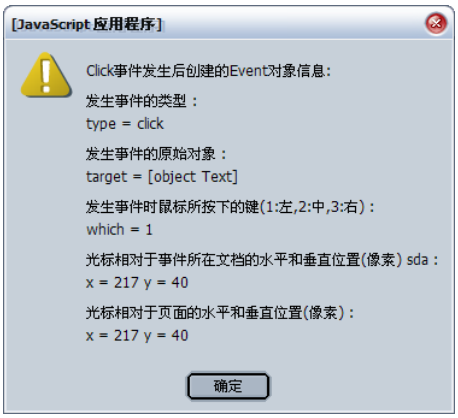

图 3.24 Netscape 中通过 Event 对象返回的事件信息

如果在触发事件处理器的过程中，仍然像 IE4 中一样，不显式传递 Event 对象，则事件处理器不能正常工作。

3.8.2 事件捕获

NN4 中没有事件上溯的概念，与此功能相类似的是事件捕获。事件捕获使 window、document、layer 对象能捕获窗口、文档和层中低层次对象的事件。如 document 对象能捕获发生在文档页面中的事件如鼠标单击某文本段等，并将此事件交给该层的事件处理器处理。事件捕获对于处理事件或者在更高的层次上定义事件处理器来取代低层次的多个事件处理器是非常有用的，典型的如表单的提交等。

NN4 中的对象提供 captureEvents()方法实现事件捕获，如下列语句实现捕获 layer 中鼠标双击事件功能：

```
layer.captureEvents(Event.DBCLICK);
```

其中 Event.DBCLICK 参数为 NN4（及 NN4 以上版本）特有的关键字常量，其指明事件的类型，同时，NN4 给出了几个常用的辅助键关键字如代表 Shift 按键的 SHIFT_MASK 等。表 3.7 列出了 NN4 中常见的关键字常量：

表 3.7 NN4 中常见的关键字常量

类型	关键字			
事件型	ABORT	BLUR	CHANGE	CLICK
	DBCLICK	DRAGDROP	ERROR	FOCUS
	KEYDOWN	KEYPRESS	KEYUP	LOAD
	MOUSEDOWN	MOUSEMOVE	MOUSEUP	MOUSEOUT
	MOUSEOVER	MOVE	RESET	RESIZE
	SELECT	SUBMIT	UNLOAD	
按键型	ALT_MASK	CONTROL_MASK	META_MASK	SHIFT_MASK

要识别特定的事件类型，可通过 Event.Name 的形式调用，Name 为上表中的关键字。考察如下捕获辅助键是否按下事件的代码（仅适用于 NN4）：

```
//源程序 3.13
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//设置事件处理器
function MyHandler(MyEvent)
{
   var msg="";
   msg+="\n 键盘辅助键识别结果 : \n\n";
   if(MyEvent.modifiers&Event.ALT_MASK)
      msg+="          事件触发时按住了 ALT 键!\n";
   if(MyEvent.modifiers&Event.CONTROL_MASK)
      msg+="          事件触发时按住了 CTRL 键!\n";
   if(MyEvent.modifiers&Event.META_MASK)
      msg+="          事件触发时按住了 Command 键!\n";
   if(MyEvent.modifiers&Event.SHIFT_MASK)
      msg+="          事件触发时按住了 SHIFT 键!\n";
   alert(msg);
}
-->
```

```
</script>
</head>
<body>
<center>
   <p>按住 ALT\CTRL\SHIFT 然后点击页面，返回识别结果！</a>
</center>
<script language="JavaScript" type="text/javascript">
<!--
//文档对象捕获事件
document.captureEvents(Event.MOUSEDOWN);
document.onmousedown=MyHandler;
-->
</script>
</body>
</html>
```

程序运行后，在文档任意地方单击鼠标，如果单击的同时按住了 Shift 和 Ctrl 键，则弹出警告框如图 3.25 所示。

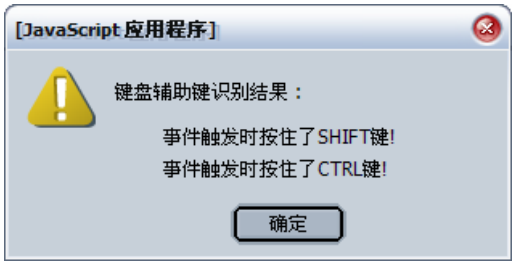

图 3.25 辅助键识别结果

关键语句：

```
document.captureEvents(Event.MOUSEDOWN);
document.onmousedown=MyHandler;
```

第一句中文档对象捕获 MouseDown 事件，第二句将 MouseDown 事件连接到其对应的事件处理器即 MyHandler()函数上面。

如果需要捕捉多个事件，各事件之间要用管道符“|”隔开。如捕获文档中所有的 Click 和 DbClick 事件可使用如下语句：

```
document.captureEvents(Event.CLICK|Event.DBCLICK);
```

3.8.3 关闭事件捕获

IE4 中通过设置 Event 对象的 cancleBubble 属性为 true 来阻止事件的上溯，而 NN4 中则通过内置的 releaseEvent()方法来将某层次上的特定事件设置为不捕获。该方法语法如下：

```
object.releaseEvent(arg);
```

其中 arg 为要设置为不捕获的目标事件列表，在表 3.7 中取值。当需要设置为不捕获的事件有很多时，各事件之间要用管道符“|”隔开。假如要将 document 层次上的所有 Click 事件和 DbClick 事件设置为不捕获，可用下列语句：

```
document.releaseEvents(Event.CLICK|Event.DBCLICK);
```

关闭事件捕获一般用于设置某种类型的事件在对象层次关系中的特定层中不需进行特别关注的场合。

3.8.4 事件传递

前面已经了解，NN4 中事件流的流向与 IE4 中正好相反，是从对象继承关系中较高层次的对象如 window、document 到较低层次的对象如标记<p>、<b>等。同时较高层次上的事件处理器往往比较低层次上的处理器触发的机会更大。

如果在实际应用中，某事件不需要在顶级对象如 document 对象层次上调用事件处理器进行处理，而只需在<table>标记的层次进行处理，此时就需要将该事件往下传递。与 IE4 中使用 fireEvent()方法改向不同，NN4 中采用 Event 对象的 routeEvent()方法将事件往下传递。该方法的语法如下：

```
routeEvent(event);
```

该方法接受一个参数 event，表示当前发生事件。该方法提供了一种在对象关系树中控制事件流向的途径，同时新的事件处理器处理完该事件后，能将处理结果返回给原始对象，后者可根据该结果更改本层的事件处理器的行为。将下列脚本加入文档的<head>和</head>标记对之间：

```
<script language="JavaScript" type="text/javascript">
<!--
function MyHandler(event)
{
   if(event.modifiers&Event.SHIFT_MASK)
      alert("鼠标单击且按住了 Shift 键");
   else
      routeEvent(event);
}
-->
</script>
```

然后将事件捕获的代码加入文档的</body>标记之前：

```
<script language="JavaScript" type="text/javascript">
<!--
window.captureEvent(Event_DBCLICK);
window.ondbclick=MyHandler;
-->
</script>
```

该实例运行后，在窗口中双击，如果此时也按下了组合键 Shift，则弹出警告框“鼠标单击且按住了 Shift 键”，否则，事件转向 window 对象所在层的下一层，即 document 对象，并将该事件通过参数的形式传到 document 对象的 DbClick 事件处理器中进行处理。

3.9 DOM 的解决之道

以 IE4 和 NN4 为代表的各大浏览器对基本事件模型的扩展都不同，给 Web 应用程序开发人员在脚本兼容性方面带来了很大的挑战，为统一标准，W3C 面向业界及时推出了一个标准的事件模型 DOM2（Document Object Model Level 2，简称 DOM2，下同），该模型在有机结合 IE4 和 NN4 事件模型的基础上，扩展了 DOM Level 1，同时加入了新的事件。DOM2 给脚本开发人员提供了一个功能十分强大的事件处理环境，并逐步获得了主流浏览器的良好支持。

注意：DOM2 最先获得支持的浏览器是 NN6，在 Microsoft 的 IE 中 DOM2 没有获得完善的支持，而是

在其最新浏览器版本中开发了类似的功能。

DOM2 提供标准的方法来访问文档中已结构化的各种对象，在 IE4 和 NN4 的基础上尽量缩小各浏览器事件模型之间的差异。并提出“事件处理器”、“UI 事件”等新概念。

3.9.1 事件流方向

在事件流的方向问题上，DOM2 综合了 IE4 和 NN4 的事件回溯和下传的做法：

- 事件下传阶段：该过程模仿 NN4 中事件流的流动过程，事件从继承关系的顶端开始搜索，直到到达目标对象为止。事件在下传过程中，可被当前的事件处理器先处理或者改向。一旦到达目标对象，该事件的处理器立即被触发。
- 事件上溯过程：事件到达目标对象之后，将模仿 IE4 中事件流的流动过程，沿着继承关系上溯至顶端，在上溯的过程中调用每一层相应的事件处理器。

事件上溯过程并不是必须的，只有当程序员在该事件下传过程中以 NN4 的方式显示捕获过此事件时，事件上溯过程才发生。

在事件下传和上溯过程中，DOM2 引入了“事件监听器”概念对事件处理器和对象进行捆绑。事件监听器本质上也是一种事件处理器，只不过它被绑定在对象继承关系的特定节点上，并可在事件生命周期的特定阶段被触发。DOM2 使用节点的 addEventListener()方法和 deleteEventListener()方法分别对事件监听器进行绑定和删除。

在事件的流动过程中，DOM2 提供节点的 dispatchEvent()方法来将当前事件改向，该方法以当前 Event 对象为参数。目标节点使用该方法后，变为新的事件节点，事件将按照正常的继承关系传递到其他的目标。

3.9.2 Event 对象

在综合 IE4 和 NN4 事件流的流动方法之外，DOM2 定义了 Event 对象的一些属性，其中常见的属性如表 3.8 所示。

表 3.8 DOM2 中Event对象的常见属性

属性	简要说明
altKey、crlKey、shiftKey、metaKey	布尔值，表示事件发生时是否按下Alt、Ctrl、Shift和Meta键
Bubbles	布尔值，指明事件是否上溯
Button	指明发生事件时鼠标所按下的键（1：左键，2：中键，3：右键）
Cancellable	指明事件是否取消
charCode	在KeyPress事件中按下的键的ASCII码
clientX、clientY	事件发生位置相对于当前Web页面的水平和垂直位置（像素）
currentTarget	事件下传和上溯过程中当前的节点
eventPhase	指明当前事件位于事件流的阶段（1：下传，2：目标，3：上溯）
keyCode	在KeyDown和KeyUp事件中所按键的ASCII码
relatedTarget	引用与事件相关的节点
screenX、screenY	光标相对于屏幕的水平和垂直位置（像素）
Target	表示发生事件的原始节点
Type	指明发生事件的类型

目标事件的属性与事件的类型紧密相关，并不是每个事件都能拥有 Event 对象的全部属性。如键盘 KeyPress 事件中就不可能拥有 button 属性，鼠标的 Click 事件就不拥有 keyCode 属性等。

3.9.3 事件类型

DOM2 基本融合了 IE4 和 NN4 中的鼠标事件、键盘事件、浏览器事件和文档事件，表 3.9 列出了 DOM2 中定义的常用事件以及能否上溯、取消等特性：

表 3.9 DOM2 中定义的基本事件及其特性

事件	能否上溯	能否取消	事件	能否上溯	能否取消
abort	是	否	mouseout	是	是
blur	否	否	mouseover	是	是
change	是	否	mouseup	是	是
click	是	是	reset	是	否
error	是	否	resize	是	否
focus	否	否	scroll	是	否
load	否	否	select	是	否
mousedown	是	是	submit	是	是
mousemove	是	否	unload	否	否

由表 3.5 和表 3.9 可以看出，DOM2 综合了 IE4 和 NN4 中种类繁多的事件，将它们融合为最基本的事件类型，如 DOM2 中的鼠标单击事件综合为 click 事件，简化了事件模型。同时事件的上溯和取消等特性也作了一定的调整。

除此之外，DOM2 还引进了“UI 事件”的概念。UI（用户界面）事件使用“DOM”作前缀，以区别于普通的事件。表 3.10 列举了几种 UI 事件及其能否上溯和取消的特性：

表 3.10 DOM2 中UI事件及其特性

事件	简要说明	能否上溯	能否取消
DOMActive	当对象激活（如鼠标单击按钮）时触发	是	是
DOMFocusIn	元素（广义的，不单指表单）获得焦点	是	否
DOMFocusOut	元素（广义的，不单指表单）失去焦点	是	否

在实际使用中，一般使用 DOM2 中定义的基本事件，而很少使用 UI 事件，其更多的是提供一种用户界面事件的模型。

上面几节简要介绍了基本事件模型、IE4 和 NN4 对基本模型的扩展以及 DOM2 标准事件模型。脚本程序员在决定使用哪个模型之前，要充分了解其目标客户使用的浏览器信息。浏览器发展的总趋势是向互相兼容的方向，也不排除短期内各浏览器厂商各自扩展标准事件模型而朝向不同方向发展的可能。总的原则是在 DOM2 标准事件模型基础上，尽量使用事件模型中兼容性较强的事件进行相关操作。

3.10 本章小结

本章主要介绍了 JavaScript 事件处理方面的相关知识，主要包括事件的类型、JavaScript 中预定义的事件处理器、如何自定义事件处理器等。同时着重讲述了 IE4 和 NN4 对基本事件模型的扩展，Event 对象在 IE4 和 NN4 中的异同点，以及在 DOM2 标准事件模型中引入的几个重要概念。

事件的基石是对象，对 JavaScript 事件处理有了基本认识之后，下章讲述 JavaScript 基于对象编程的相关知识。