1、目前共存在四种事件处理模型分别是：**原始事件模型、标准事件模型、IE事件模型**，还有一种Netscape4事件模型，但基本可忽略

2、事件处理模型又可以分为**基本事件处理**和**高级事件处理**两种，原始事件模型属于基本事件处理，标准事件模型和IE事件模型属于高级事件处理

**一、基本事件处理：**

基本事件处理主要是指原始事件模型实现的事件处理。其主要分为以下两种：

     （**1）、作为HTML标签性质的事件处理**，比如 <div onmouseover=”var a=1; alert();”>……</div>           //在这里，onmouseover只是一个代表，还包含其他很多事件

               在这种方式中，赋给onmouseover等事件处理函数的是JS代码串，系统会把这些代码串自动包装在一个匿名函数中。其中可以有**this**关键字，它指向的是这个标签元素，也可以有**event**关键字，它表示的是事件发生时的事件对象（用在标准浏览器中）。如<div onmouseover=”f(this,event);”>……</div>

               其实我们可以把onmouseover等看成是一个函数，在没给它赋值前，它就是一个空函数。给它赋了js代码后，就相当于往空函数里添加了代码。因为onmouseover等其实是一个函数，所以我们可以显示的调用它，例如 element.onmouseover(),但这样并不会引起mouseover事件的真正发生。

               可以给事件函数（即onmoouseover等）返回false来举止默认动作的发生。

               函数是运行在定义它的作用域中，因此如果给事件处理函数赋值js代码，就相当于是在这个HTML标签环境中定义了一个函数，这种环境比较特殊，它的高一级的作用域并不是window全局对象环境，在这两者之间还相隔这至少一种作用域环境。而在<script>中定义的函数，它的高一级作用域环境就是window(当然嵌套函数又要另当别论)。所以，在HTML标签中，最好把代码放在一个在<script>中定义的函数里，然后再把这个函数调用赋给事件函数,即<div onmouseover=“function();”>……</div>

**(2)、作为javascript属性的事件处理**    比如element.onmouseover=function(){……}

               注意在这种方式，不能再给事件处理函数赋值js代码串了，而是直接把函数（不是函数调用）赋给它，或是赋一个匿名函数，如 element.onmouseover=function(){……}  或    element.onmouseover=f； f为一个函数，在这里不能加上括号

**基本事件处理也会以冒泡的形式向上传播**

**二、高级事件处理：**

高级事件处理只要是指标准事件模型和IE事件模型实现的事件处理。

       【概念理解】 事件的传播可分两种：一种是捕获传播，一种是冒泡传播。

        捕获传播：即事件从外传到里，每一级都发生了该事件

        冒泡传播；即事件从里传到外，每一级都发生了该事件，并不是所有的事件都会冒泡

**（1）标准事件模型**

  标准事件模型既能发生冒泡传播也能发生捕获传播。

             【 事件注册函数】

**addEventListener()** 该方法为特定元素注册事件处理程序，有三个参数，第一个参数是事件名，注意没有前缀on，第二个是处理函数（当然也可以是匿名函数），事件发生时，系统会自动给该函数的第一个参数传入一个Event对象。最后一个参数是布尔值，如果为true,则只用于事件捕获传播阶段，如果为false则只用于事件冒泡传播阶段，一般我们把它设为false

              例如：element.addEventListener(“click”,f,false)    //其中f为一个函数

              f函数可以这样定义：function f(e){……}  //其中的参数在事件发生时就代表Event对象

            高级事件处理的一大优势是可以给同一个元素注册多个事件处理函数，这些事件函数执行的顺序并不能确定，但一般来说是按注册的先后顺序来执行，如果注册了重复的事件函数，则只有第一个会生效。

**removeEventListener()** 该方法用来解除事件注册，它的三个参数与**addEventListener()** 相同

**（2）IE事件模型**

IE事件模型只支持事件冒泡传播

           【 事件注册函数】

**attacthEvent()**  该方法只有2个参数，一个为事件名，注意有前缀on，第二个为事件处理函数。例如 element.attachEvent(“onclick”,f)

            IE事件模型的Event对象并不是作为事件发生时做为参数传入事件处理函数的，IE的Event对象是一个window的全局对象，只有在事件发生时才可以访问。

            所以f函数可以这样定义：  function f(){var e=window.event;……}   //其中e就取得了Event对象

**detachEvent()**  该方法用来取消事件注册，参数与**attacthEvent()**  相同

**addEventListener()** 与**attacthEvent()**  的一个重要差别是**attacthEvent()**注册的事件处理函数中的**this**关键字永远是指向window对象的，而**addEventListener()** 注册的事件处理函数中的**this**指向的是发生了事件的元素

**（3）、IE与标准事件模型的Event对象比较**

|  |  |  |  |
| --- | --- | --- | --- |
| **IE 事件对象** | **IE事件对象** | **标准事件对象** | **标准事件对象** |
| altKey | true表示按下了ALT键，false表示没有 | altKey | true表示按下了ALT键。false表示没有 |
| ctrlKey | true表示按下了CTRL键，false表示没有 | ctrlKey | true表示按下了CTRL键，false表示没有 |
| shiftKey | true表示按下了SHIFT键，false表示没有 | shiftKey | true表示按下了SHIFT键，false表示没有 |
| button | 鼠标事件。0表示没有按下鼠标键，1为按下左键，2为按下右键，4为中间键，3为同时按下左右键，5为按下左键和中键，6为按下右键和中键，7为按下左键、中键、右键 | button | 0为左键，1为中键，2为右键 |
| clientX | 事件发生时，鼠标在浏览器窗口（不包含工具栏、滚动条等）的X坐标 | clientX | 事件发生时，鼠标在浏览器窗口（不包含工具栏、滚动条等）的X坐标 |
| clientY | 同上 | clientY | 同上 |
| screenX | 事件发生时，鼠标在整个屏幕上的X坐标 | screenX | 事件发生时，鼠标在整个屏幕上的X坐标 |
| screenY | 同上 | screenY | 同上 |
| type | 事件的名称（如click) | type | 事件的名称(如click) |
| srcElement | 引起事件的元素 | target | 引起事件的元素 |
| keyCode | 对于keypress事件，表示按钮的unicode字符，对于keydown和keyup事件则表示按钮的数字代码 | charCode | 表示按键的Unicode字符 |
|  |  | keyCode | 表示按键的数字代码 |
| cancelBubble | 值为true时将阻止事件继续向上冒泡 | stopPropagation | 可以调用这个方法来阻止事件继续向上冒泡 |
|  |  | cancelBubble | true表示事件冒泡已被取消，false表示没有 |
| returnValue | 值为false时将会阻止事件的默认行为 | preventDefault() | 调用该方法可以阻止事件的默认行为 |
| offsetX | 获取事件发生时鼠标相对于引起事件的元素的X坐标，即以引起事件的元素的本身（不用计算padding和margin)的左上角为原点 | layerX | 当引发事件的元素没有动态定位时，返回鼠标相对于引发事件的元素的最近的一个设置了动态定位的父元素里的X坐标，以其父元素的边框的左上角为原点。          当引发事件的元素设置了动态定位后，则返回鼠标相对于引发事件的元素的X坐标，以该元素边界的左上角为原点。 |
| x | 获取鼠标相对于引发事件的元素的最近一个设置了动态定位的父元素的X坐标，以该父元素的边框i的坐上角为原点 |  |  |

**资料链接**: <http://www.cnblogs.com/2050/archive/2010/07/25/1784712.html>