# 事件

管理科学与工程学科
耿方方

# 主要内容

- 事件处理程序
- 事件流
- 事件对象
- 事件类型

- JavaScript与HTML之间的交互是通过事件实现的。事件,就是文档或浏览器窗口发生的一些特定的交互瞬间。也就是鼠标或热键的动作。可以使用侦听器（或处理程序）来预订事件，以便事件发生时执行相应的代码。

# 事件处理程序



- 当用户在Web页面上同HTML进行交互时，事件触发JavaScript代码的过程分为三个步骤，这些步骤被称为“事件处理”。

  1、选中需要使用脚本进行事件响应的元素节点。

  2、声明需要在选中节点上响应触发的事件。

  3、指定当事件发生时需要运行的代码。

# 事件处理程序



- 事件就是用户或浏览器自身执行的某种动作。诸如click、load和mouseover，都是事件的名字。而响应某个事件的函数叫做事件处理程序（或事件侦听器）事件处理程序的名字以“on”开头，因此click事件的事件处理程序就是onclick,load事件的事件处理程序就是onload。为事件指定处理程序的方式有好几种：
- HTML事件处理程序：
- 早期版本的HTML中会包含一组属性用来响应它所属元素的事件。这些属性的名字和事件的名字相匹配，它们的值则是当事件发生时需要运行的函数的名称。
- 例如：<a onclick=”hide()”>表示当用户点击这个<a>元素后，hide（）方法会被调用。
- 这种事件处理的方法用的很少，由于要实现JavaScript和HTML的分离。

# 事件处理程序



- 案例1:
- 
- <h1>List King</h1>
- <h2>New Account</h2>
- <form method="post" action="http://www.example.org/register">
- <label for="username">Create a username: </label>
- <input type="text" id="username" onblur="checkUsername()" />
- 
- <label for="password">Create a password: </label>
- <input type="password" id="password" />
- <input type="submit" value="sign up!" />
- </form>

```
function checkUsername() {
  var elMsg =
document.getElementById('feedback');
  var elUsername =
document.getElementById('username');
  if (elUsername.value.length < 5) {
    elMsg.textContent = 'Username must be 5
characters or more'; // Set msg
  } else {
    elMsg.textContent = '';
  }
```

# 事件处理程序



- 传统的DOM事件处理程序

所有现代浏览器都支持这种创建事件处理程序的方法，不过每个事件处理程序只能附加一个函数。语法：

element.onevent=functionName;

element:元素，目标DOM节点。

onevent：事件，绑定到该节点的事件，使用前缀on。

functionName:需要调用的函数的名称（后面不带小括号）。

# 事件处理程序



- 案例2：

```
<form method="post" action="http://www.example.org/register">
        <label for="username">Create a username: </label>
        <input type="text" id="username" />
        <div id="feedback"></div>
        <label for="password">Create a password: </label>
        <input type="password" id="password" />
        <input type="submit" value="sign up" />
    </form>
```

```
<script>
    function checkUsername() {
  var elMsg = document.getElementById('feedback');
  if (this.value.length < 5) {
    elMsg.textContent = 'Username must be 5 characters or more';   }
else {
    elMsg.textContent = '';
  }
}

var elUsername = document.getElementById ('username');
elUsername.onblur = checkUsername;
 // When it loses focus call checkuserName()
   </script>
```

# 事件处理程序



- DOM事件处理程序

近几年又新增了一种事件处理程序，就是事件监听器,它可以同时触发多个函数,但是在旧的浏览器中不被支持。

语法:

element.addEventListerner('event',functionName[,Boolean]);

element: 目标DOM元素节点。

event: 在引号中指定需要绑定到节点的事件。

functionName:需要调用的函数的名称。

事件流: 指定是否为捕获方式的事件响应，通常被设置为false。

# 事件处理程序



- 案例3：
- <form method="post" action="http://www.example.org/register">
- <label for="username">Create a username: </label>
- <input type="text" id="username" />
- 
- <label for="password">Create a password: </label>
- <input type="password" id="password" />
- <input type="submit" value="sign up" />
- </form>

```
<script>
        function checkUsername() {
  var elMsg = document.getElementById('feedback');
  if (this.value.length < 5) {
    elMsg.textContent = 'Username must be 5
 characters or more';
  } else {
    elMsg.textContent = '';
  }
}

var elUsername = document.getElementById
('username');

elUsername.addEventListener
('blur', checkUsername, false);
        </script>
```

# 事件处理程序



- DOM事件处理程序和事件监听器中使用参数

由于在注册事件处理程序和事件监听器时，在函数名称的后面是没有小括号的，因此需要采用其他的手段来传递参数。

语法：

```
element.addEventListerner('event',function(){functionName(  参数); }[,Boolean]);
```

# 事件处理程序



- 案例4：

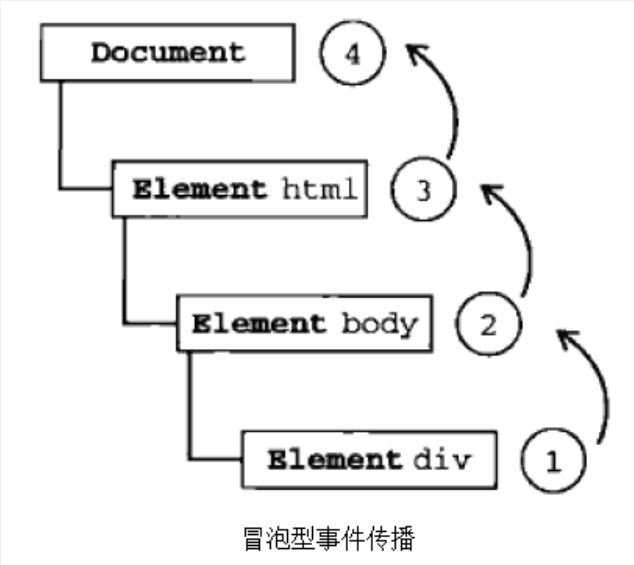
```
<script>
 var elUsername = document.getElementById('username');   // Username input
var elMsg      = document.getElementById('feedback');   // Error msg element

function checkUsername(minLength) {                  // Declare function
  if (elUsername.value.length < minLength) {         // If username too short
    // Set the error message
    elMsg.innerHTML = 'Username must be ' + minLength + ' characters or more';
  } else {                                           // Otherwise
    elMsg.innerHTML = '';                            // Clear msg
  }
}

elUsername.addEventListener('blur', function() {       // When it loses focus
  checkUsername(5);                                    // Pass argument here
}, false);
          </script>
```

# 事件流

- 事件流：由于DOM结构是一个树型结构，HTML元素都位于另一些元素中。如果移动鼠标到一个链接上，或者点击一个链接，同样会把鼠标移动到它的父元素上，或者点击它的父元素。因此事件流描述的是从页面中接收事件的顺序，事件流包括事件冒泡和事件捕获。
- 冒泡事件：事件的传播是从最特定的事件目标到最不特定的事件目标。即从DOM树的叶子到根。
- 捕获事件：事件的传播是从最不特定的事件目标到最特定的事件目标。即从DOM树的根到叶子。

# 事件流

- 冒泡事件：以下面的网页为例：
- <html>
- <head>
- <title>冒泡事件</title>
- </head>
- <body>
- Click me
- </body>
- <html>

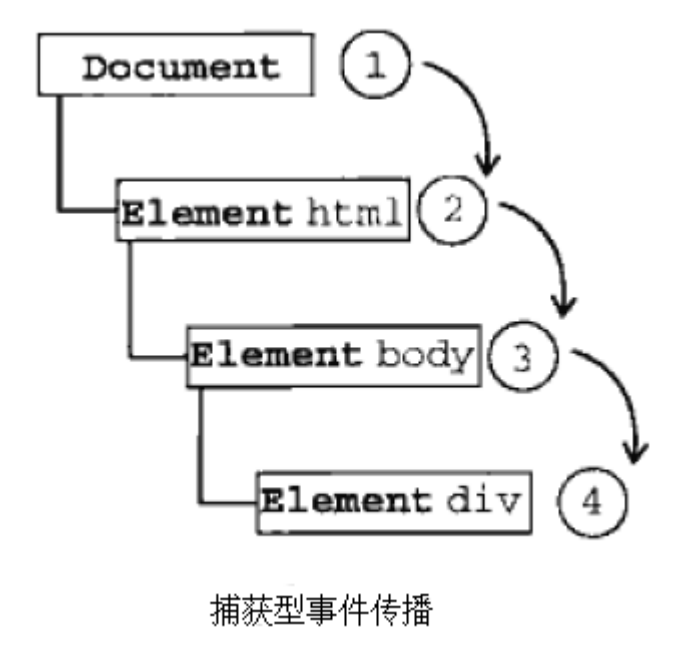


冒泡型事件传播

1、所有现代浏览器都支持事件冒泡，但在具体实现中略有差别：
IE5.5及更早版本中事件冒泡会跳过<html>元素(从body直接跳到document)。
IE9、Firefox、Chrome、和Safari则将事件一直冒泡到window对象。

# 事件流

- 捕获事件：



捕获型事件传播

2、IE9、Firefox、Chrome、Opera、和Safari都支持事件捕获。

尽管DOM标准要求事件应该从document对象开始传播，但这些浏览器都是从window对象开始捕获事件的。

3、由于老版本浏览器不支持，很少有人使用事件捕获。建议使用事件冒泡。

# 事件流

- 当代码在一个元素和其祖先元素或后代元素上都有事件处理程序时，事件流才会变得非常重要。
- 对于DOM事件处理程序，所有现代浏览器默认都会使用事件冒泡模型而不是事件捕获模型。addEventListener()方法的最后一个参数允许选择事件触发的方向：
- true表示捕获方式
- false表示冒泡方式。

# 事件流

- 案例5：<body>
- 
- <h1>List King</h1>
- <h2>Bubble</h2>
- <ul id="list">
- <li id="item"><a id="link">fresh figs</a></li>
- </ul>
- 
- 
- <h1>List King</h1>
- <h2>Capture</h2>
- <ul id="list2">
- <li id="item2"><a id="link2">fresh figs</a></li>
- </ul>
- 

```
<script>
        function showElement() {
  alert(this.innerHTML);
}

el = document.getElementById("list");
el.addEventListener('click', showElement, false);

el = document.getElementById("item");
el.addEventListener('click', showElement, false);

el = document.getElementById("link");
el.addEventListener('click', showElement, false);

el = document.getElementById("list2");
el.addEventListener('click', showElement, true);

el = document.getElementById("item2");
el.addEventListener('click', showElement, true);

el = document.getElementById("link2");
el.addEventListener('click', showElement, true);
        </script>
```

# 事件对象

- 在触发DOM上的某个事件时，会产生一个事件对象event，这个对象中包含所有与特定事件相关的信息。包括导致事件的元素，事件的类型以及其他与特定事件相关的信息。
- 例如，鼠标操作导致的事件对象中，会包含鼠标位置的信息，而键盘操作导致的事件对象中，会包含与按下的键有关的信息。所有浏览器都支持event对象，但支持方式不同。
- 如果需要传递一个参数给命名函数，事件对象作为匿名封装函数的第一个参数传递进去，然后需要为命名函数指定相应的参数。
- 当事件对象被传递给函数时，它的参数名称通常都是e(event的缩写)。不过需要注意的是，有些程序员使用参数e来表示错误对象，所以在某些脚本中e来表示错误对象，也可能表示错误。

# 事件对象

- 无参数的事件监听器

```
function checkUsername(e){
var target=e.target;
}

var el=document.getElementById('username');
el.addEventListener('blur',checkUsername,false);
```

# 事件对象

- 带参数的事件监听器

```
function checkUsername(e, minLength){
var target=e.target;
}
var el=document.getElementById('username');
el.addEventListener('blur',function(e){
checkUsername(e,5);
 },false);
```

# 事件对象

- 案例6:

```
<script>
        function checkLength(e, minLength) {         // Declare function
  var el, elMsg;                                     // Declare variables
  if (!e) {                                          // If event object doesn't exist
    e = window.event;                                // Use IE fallback
  }
  el = e.target || e.srcElement;                     // Get target of event
  elMsg = el.nextSibling;                            // Get its next sibling

  if (el.value.length < minLength) {                 // If length is too short set msg
    elMsg.innerHTML = 'Username must be ' + minLength + ' characters or more';
  } else {                                           // Otherwise
    elMsg.innerHTML = '';                            // Clear message
  }
}
```

# 事件对象

- 案例6:

```
var elUsername = document.getElementById('username');// Get username input
if (elUsername.addEventListener) {           // If event listener supported
  elUsername.addEventListener('blur', function(e) {  // On blur event
      checkLength(e, 5);                              // Call checkLength()
  }, false);                                        // Capture in bubble phase
} else {                                            // Otherwise
  elUsername.attachEvent('onblur', function(e) {    // IE fallback onblur
      checkLength(e, 5);                              // Call checkLength()
  });
}
            </script>
```

# 事件对象



- 事件委托
- 如果用户可以和页面中的大量元素进行交互，比如：UI中的大量按钮，一个很长的列表，表格中的每一个单元格。向这些元素中分别添加事件监听器就会使用大量内存，从而降低性能。
- 事件可以影响到容器元素（或祖先元素），因此可以将事件处理程序放置在一个容器元素上，然后使用事件对象的target属性找到它的后代中是哪一个发生了事件。
- 因此可以将事件监听器的工作委托给这些元素的父元素

# 事件对象

- 改变默认行为：事件对象有一些方法可以改变一个元素的默认行为，以及它的祖先元素如何对这个事件做出响应。
- preventDefault():有一些事件，比如点击链接或者提交表单，会把用户导向另一个页面。为了阻止这类元素的这种默认行为，可以使用事件对象的该方法。IE5-IE8有个具有相同功能的属性returnValue。

```
if(event.preventDefault){
    event.preventDefault();
}else{
    event.returnValue=false;
}
```

# 事件对象

- 改变默认行为：事件对象有一些方法可以改变一个元素的默认行为，以及它的祖先元素如何对这个事件做出响应。
- stopPropagation():处理完某个元素上的事件之后，可能需要阻止这个事件向其祖先元素继续冒泡传播，使用事件对象的stopPropagation()方法。
- 在IE8和更早的版本的IE中，使用拥有同样功能的属性cancelBubble，将其设置为true可以达到同样的效果。

```
if(event.stopPropagation){
    event.stopPropagation();
}else{
    event.cancelBubble=true;
}
```

# 事件对象

- 改变默认行为：事件对象有一些方法可以改变一个元素的默认行为，以及它的祖先元素如何对这个事件做出响应。
- 同时使用这两者。在同样的场景下，可能在某些函数中看到使用了下列的语句，return false；这种方式既阻止了元素的默认行为，也阻止了事件继续向上冒泡或向下传播，它可以运行在所有浏览器中。

# 事件对象

- 事件代理举例7-06-1：

```
<script>
        function getTarget(e){
            if(!e){
                e=window.event;
            }
            return e.target||e.srcElement;
        }
        function itemDone(e){
            var target,elParent;
            target=getTarget(e);
            elParent=target.parentNode;

            elParent.removeChild(target);
            if (e.preventDefault){
                e.preventDefault();
            }
        else{
                e.returnValue=false;
            }
        }
        var el=document.getElementById('liebiao');
        if(el.addEventListener){
          el.addEventListener('click',function(e){
            itemDone(e);
          },false);}
        else
        {
          el.attachEvent('onclick',function(e){
            itemDone(e);
          });
        }
    </script>
```

# 事件类型



- 用户界面事件：用户界面（UI）事件发生在用户与浏览器本身（而不是其中的HTML界面）进行交互的过程中，例如当页面加载完成时，或是浏览器窗口的大小发生变化时。UI事件处理程序/监听器附加在浏览器窗口上。

| 事件 | 触发 |
| --- | --- |
| load | 当Web页面加载完成时触发。它同样可以用于其他元素节点的加载事件，比如图片、脚本或者对象。 |
| unload | 当Web页面卸载时，通常指因为请求了一个新页面的发生，该事件会在用户离开页面之前触发。 |
| error | 当浏览器遇到JavaScript错误或不存在的资源时触发。 |
| resize | 当浏览器窗口改变大小时触发。 |
| scroll | 当用户向上或向下滚动页面时触发。它可以关联整个页面，也可以关联页面中某些特定元素。 |

# 事件类型

- 案例7：

```
<body>
        <img src="Images/7-07.jpg" name="img1" onload="blowup()" onmouseover="reduce()" alt="img1">
        <script language="javascript">
        var h=img1.height;
        var w=img1.width;
       function blowup(){//缩小图片
    if (img1.height>=h){
            img1.height=h*0.5;
            img1.width=h*0.5;
    }
}
function reduce(){//恢复图片的原始大小
        if (img1.height<h){
                img1.height=h;
                img1.width=w;
        }
}
</script>

  </body>
```

# 事件类型

- 案例8：

```
<body onresize="alert('You have changed the size of the window')">
<p>Try to resize the browser window.</p>
</body>
```

# 事件类型



- 表单事件：实际上就是对元素获得或失去焦点的动作进行控制。可以利用表单事件来改变获得或失去焦点元素样式。
- 获得焦点和失去焦点事件
- 获得焦点事件（onfocus）是当某个元素获得焦点时触发事件处理程序。失去焦点事件（onblur）是当前元素失去焦点时触发事件处理程序。在一般情况下，这两个事件是同时使用的。

# 事件类型

- 案例9：
- <tr>
- <td width="108">用户名:</td>
- <td width="213"><form name="form1" method="post" action="">
- <input type="text" name="textfield" onfocus="txtfocus()" onBlur="txtblur()">
- </form></td>
- </tr>

```
<script language="javascript">
function txtfocus(e){
        var e=window.event;
        var obj=e.target||e.srcElement;
        obj.style.background="#FFFF66";
}
function txtblur(e){
        var e=window.event;
        var obj=e.target||e.srcElement;
        obj.style.background="#FFFFFF";
}
</script>
```

# 事件类型

- 失去焦点内容改变事件（onchange）：是当前元素失去焦点并且元素的内容发生改变时触发事件处理程序。例如该事件在下拉文本框，选中一个单选按钮时，选中或取消一个复选按钮时。

# 事件类型

- 案例10:
- <form name="form1" method="post" action="">
- <input name="textfield" type="text" size="23">
- <select name="menu1" onChange="Fcolor()">
- <option value="black">黑</option>
- <option value="yellow">黄</option>
- <option value="blue">蓝</option>
- <option value="green">绿</option>
- <option value="red">红</option>
- <option value="purple">紫</option>
- </select>
- </form>

```
<script language="javascript">
function Fcolor(){
        var e=window.event;
        var obj=e.srcElement;
        form1.textfield.style.color=
obj.options[obj.selectedIndex].value;
}
</script>
```

# 事件类型



- 提交与重置事件
- 表单提交事件（onsubmit）是在用户提交表单时（通常使用“提交”按钮，也就是将按钮的type属性设为submit），在表单提交之前被触发，因此，该事件的处理程序通过返回false值来阻止表单的提交。该事件可以用来验证表单输入项的正确性。
- 表单重置事件（onreset）与表单提交事件的处理过程相同，该事件只是将表单中的各元素的值设置为原始值。一般用于清空表单中的文本框。

# 事件类型

- 案例11：
- <form name="form1" onReset="return AllReset()" onsubmit="return AllSubmit()">

```
<script language="javascript">
function AllReset(){
        if (window.confirm("是否进行重置？"))
                return true;
        else
                return false;
}
function AllSubmit(){
        var T=true;
        var e=window.event;
        var obj=e.srcElement;
        for (var i=1;i<=7;i++){
                if (eval("obj."+"txt"+i).value==""){
                        T=false;
                        break;
                }
        }
        if (!T){
                alert("提交信息不允许为空");
        }
        return T;
}
</script>
```

# 事件类型



- 鼠标点击事件
- 鼠标点击事件包括单击（click）和双击（dblclick）。
  - 单击事件（onclick）是在鼠标单击时被触发的事件。单击是指鼠标停留在对象上，按下鼠标键，在没有移动鼠标的同时放开鼠标键的这一完整过程。
  - 单击事件一般应用于Button对象、Checkbox对象、Image对象、Link对象、Radio对象、Reset对象和Submit对象，Button对象一般只会用到onclick事件处理程序，因为该对象不能从用户那里得到任何信息，如果没有onclick事件处理程序，按钮对象将不会有任何作用。

# 事件类型



- 案例12：

```
    <body>
<script language="javascript">
var Arraycolor=new
Array("olive","teal","red","blue","maroon","navy","lime","fuschia","green","purple","gray","yellow","aqua","white","silver");
var n=0;//为变量赋初值
function turncolors(){//自定义函数
     if (n==(Arraycolor.length-1)) n=0;//判断数组指针是否指向最后一个元素
     n++;//变量自加1
     document.bgColor = Arraycolor[n];//设置背景颜色为对应数组元素的值
}
</script>
<form name="form1" method="post" action="">
<p>
    <input type="button" name="Submit" value="变换背景" onclick="turncolors()">
</p>
<p>用按钮随意变换背景颜色.</p>
</form>
</body>
```

# 事件类型



- 鼠标按下和松开
- 鼠标的按下和松开事件分别是onmousedown和onmouseup事件。其中，onmousedown事件用于在鼠标按下时触发事件处理程序，onmouseup事件是在鼠标松开时触发事件处理程序。在用鼠标单击对象时，可以用这两个事件实现其动态效果。

# 事件类型



- 案例13：

```
<body>
   <form name="form1" method="post" action="">
  <p id="p1" style="color:#AA9900 " onmousedown="mousedown()" onmouseup="mouseup()"><u>河南中医药大学</u></p>
</form>

<script language="javascript">
function mousedown(){
      var obj=document.getElementById('p1');
      obj.style.color='#0022AA';
}
function mouseup(){
      var obj=document.getElementById('p1');
      obj.style.color='#AA9900';
      window.open("http://www.hactcm.edu.cn","河南中医药大学","");
}
</script>
</body>
```

# 事件类型



- 鼠标移入和移出事件
- 鼠标的移入和移出事件分别是onmouseover和onmouseout事件。其中，onmouseover事件在鼠标移动到对象上方时触发事件处理程序，onmouseout事件在鼠标移出对象上方时触发事件处理程序。可以用这两个事件在指定的对象上移动鼠标时，实现其对象的动态效果。

# 事件类型



- 鼠标移动事件
- 鼠标移动事件（onmousemove）是鼠标在页面上进行移动时触发事件处理程序，可以在该事件中用document对象实时读取鼠标在页面中的位置。

# 事件类型



- 键盘事件包含onkeypress、onkeydown和onkeyup事件，其中onkeypress事件是在键盘上的某个键被按下并且释放时触发此事件的处理程序，一般用于键盘上的单键操作。onkeydown事件是在键盘上的某个键被按下时触发此事件的处理程序，一般用于组合键的操作。onkeyup事件是在键盘上的某个键被按下后松开时触发此事件的处理程序，一般用于组合键的操作。

# 事件类型

- 案例14：

```
<div id="page">
    <h1>List King</h1>
    <form id="messageForm">
      <h2>My profile</h2>
  <textarea id="message"></textarea>
      <div id="charactersLeft">180 characters</div>
      <div id="lastKey"></div>
    </form>
  </div>
```

键盘事件

```
var el;                                    // Declare variables
function charCount(e) {                    // Declare function
  var textEntered, charDisplay, counter, lastkey;          // Declare variables
  textEntered = document.getElementById('message').value;  // User's text
  charDisplay = document.getElementById('charactersLeft'); // Counter element
  counter = 180 - (textEntered.length);    // Num of chars left
  charDisplay.textContent = counter;       // Show chars left
  lastkey = document.getElementById('lastKey');            // Get last key elem
  lastkey.textContent = 'Last key in ASCII code: ' + e.keyCode;  // Create msg
}
el = document.getElementById('message');
el.addEventListener('keyup', charCount, false);
```

# 事件类型



- 文本编辑事件是指对浏览器中的内容进行选择、复制、剪切和粘贴时所触发的事件。
- 复制事件是在浏览器中复制被选中的部份或全部内容时触发事件处理程序，复制事件有onbeforecopy和oncopy两个事件，onbeforecopy事件是将网页内容复制到剪贴版时触发事件处里程序，oncopy事件是在网页中复制内容时触发事件处里程序。
- 注意：必须在函数名前面加return语句。否则，当前事件返回的值一律是true值，也是允许复制的。

即：<body oncopy=”return false”>
</body>

- 例如：<body oncopy=” return p()” >
- </body>
- <script>
- function p(){alert('该页面内容不允许复制');
- return false;}
- </script>

# 事件类型



- **剪切事件**：是在浏览器中剪切被选中的内容时触发事件处理程序，剪切事件有onbeforecut和oncut两个事件，onbeforecut事件是当页面中的一部分或全部内容被剪切到浏览者系统剪贴板时触发事件处理程序，oncut事件是当页面中被选择的内容被剪切时触发事件处理程序。
- 粘贴事件：粘贴事件（onbeforepaste）是将内容要从浏览者的系统剪贴板中粘贴到页面上时所触发的事件处理程序。可以利用该事件避免浏览者在添写信息时，对验证信息进行粘贴，如密码文本框和确定密码文本框中的信息。

# 事件类型



- 选择事件：选择事件是用户在body、input或textarea表单区域中选择文本时触发事件处理程序。选择事件有onselect和onselectstart两个事件。

onselect事件是当文本内容被选择时触发事件处理程序。只能在相应的文本中选择一个字符或是一个汉字后触发本事件，并不是用鼠标选择文本后，松开鼠标时触发。

onselectstart事件是开始对本文的内容进行选择时触发事件处理程序。在该事件中可以用return语句屏蔽文本的选择操作。

# 事件类型

文本编辑事件

案例15：

```
<body>
<form name="form1" method="post" action="">
    <input name="textfield" type="text" onselect="return Tselect()" value="hello!JavaScript">
</form>
<script language="javascript">
function Tselect(){
    var txt=document.selection.createRange().text;//获取当前所选中的文本
    if (txt=="hello!"){
            alert("你所选择的内容为："+txt);
    }
}
</script>
</body>
```

# 事件类型



- 任何时候当元素被添加到DOM中或从DOM中移除时，DOM的结构就发生了变化，而这种变化会触发变动事件。

| 事件 | 触发 |
| --- | --- |
| DOMNodeInserted | 当一个节点被插入到DOM树中触发，例如使用appendChild()、replaceChild（）时。 |
| DOMNodeRemoved | 当一个节点从DOM树中移除时触发，例如使用removeChild()、replaceChild(). |
| DOMSubtreeModified | 当DOM结构发生变化时触发，它会在上述两个事件发生之后再触发。 |
| DOMNodeInsertedIntoDocument | 当一个节点作为后代节点被插入到另一个已经存在于文档中的节点时触发。 |
| DOMNodeRemovedFromDocument | 当一个节点作为后代节点从另一个已经存在于文档中的节点移除时触发。 |

# 事件类型



- 案例16
- <body>
- 
- <h1>List King</h1>
- <h2>Buy Groceries 1</h2>
- <ul id="list">
- <li>fresh figs</li>
- </ul>
- <a href="/additem" class="add">Add list item</a>
- 
- <script>
- var elList, addLink, newEl, newText, counter, listItems; // Declare variables
- elList = document.getElementById('list'); // Get list
- addLink = document.querySelector('a'); // Get add item button
- counter = document.getElementById('counter');

```
function addItem(e) {                          // Declare function
  e.preventDefault();                          // Prevent link action
  newEl = document.createElement('li');        // New <li> element
  newText = document.createTextNode('New list item');    // New text node
  newEl.appendChild(newText);                  // Add text to <li>
  elList.appendChild(newEl);                   // Add <li> to list
}

function updateCount() {                       // Declare function
  listItems = elList.getElementsByTagName('li').length;  // Get total of <li>s
  counter.innerHTML = listItems;               // Update counter
}

addLink.addEventListener('click', addItem, false);       // Click on button
elList.addEventListener('DOMNodeInserted', updateCount, false); // DOM updated
      </script>
```

# 事件类型

- 有三个页面级的事件在HTML5版本的规范中被引入，并且迅速变得流行起来。

| 事件 | 触发 |
| --- | --- |
| DOMContentLoaded | 当DOM树形成后触发（与此同时，图片、CSS和JavaScript脚本可能还在加载）。在这个事件中，脚本运行要早于load事件，因为load事件会等待所有资源。这种方式让页面看起来加载速度更快。 |
| hashchange | 当URL的hash值变化时触发 |
| beforeunload | 当页面被卸载之前在window对象上触发，应该只用来帮助用户。例如，可以帮助用户知道在表单中修改了数据但是尚未保存。 |

# 事件类型

- 案例17：

```
<div id="page">
    <h1>List King</h1>
    <h2>Profile</h2>
    <form id="messageForm" action="http://example.org/">

      <textarea id="message"></textarea>
      <input type="submit" value="next" />

    </form>
  </div>
```

```
<script>
        function setup() {
  var textInput;
  textInput = document.getElementById('message');
  textInput.focus();
}

window.addEventListener('DOMContentLoaded', setup, false);

window.addEventListener('beforeunload', function(event) {
   var message = 'You have changes that have not been saved';
  (event || window.event).returnValue = message;
  return message;
});
      </script>
```

# 想一想

- 制作一个简单的用户注册页面，应用表单事件中的失去焦点事件判断用户输入的用户名和密码，以及确认用户名和密码是否符合要求？

- 用户名：不能少于3个字符
- 密码：　不能少于6个字符
- 确认密码：

- 提交　　重置