chenlulouis

导航
博客园
首页
新随笔
联系
订阅 XML
管理

< 2016年8月 >

日	一	二	三	四	五	六
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

addEventListener

addEventListener一开始

前面零散地写了些关于 addEventListener 的内容，觉得比较散，有些地方可能也说得不够清楚明白，所以决定以连载的形式从头到尾再写一篇。

addEventListener 用于注册事件处理程序，IE 中为 attachEvent，我们为什么讲 addEventListener 而不讲 attachEvent 呢？一来 attachEvent 比较简单，二来 addEventListener 才是 DOM 中的标准内容。

简介

addEventListener 为文档节点、document、window 或 XMLHttpRequest 注册事件处理程序，在以前我们一般是 <input type="button" onclick="...", 或 document.getElementById("testButton").onclick = FuncName， 而在 DOM 中，我们用 addEventListener（IE 中用 attachEvent）。

语法

target.addEventListener(type, listener, useCapture);

- target 文档节点、document、window 或 XMLHttpRequest。
- type 字符串，事件名称，不含“on”，比如“click”、“mouseover”、“keydown”等。
- listener 实现了 EventListener 接口或者是 JavaScript 中的函数。
- useCapture 是否使用捕捉，看了后面的事件流一节后就明白了，一般用 false。

示例

常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签
asp.net(2)
debug(2)
rdlc(2)
高并发(1)
高负载(1)
架构(1)
http(1)
linux(1)

随笔档案
2012年10月 (1)
2012年7月 (3)
2012年3月 (1)
2012年1月 (1)
2011年7月 (1)
2011年4月 (1)
2011年2月 (1)
2011年1月 (1)
2010年12月 (4)
2010年11月 (1)
2010年8月 (1)
2010年7月 (4)
2010年6月 (4)
2010年5月 (15)
2010年3月 (5)
2010年1月 (5)
2009年12月 (12)
2009年11月 (3)
2009年10月 (12)
2009年9月 (1)
2009年8月 (8)
2009年7月 (5)
2009年6月 (1)

blog
.NET源码
Michael's blog
老赵
雨痕

community
codeproject
msdn

dictionary
iciba

forum

```
function Go()
{
   //...
}

document.getElementById("testButton").addEventListener("click", Go, false);
```

或者 listener 直接就是函数

```
document.getElementById("testButton").addEventListener("click", function () { ... }, false);
```

addEventListener—事件流

说到 addEventListener 不得不说到事件流，先说事件流对后面的解释比较方便。

当一个事件发生时，分为三个阶段：

捕获阶段 从根节点开始顺序而下，检测每个节点是否注册了事件处理程序。如果注册了事件处理程序，并且 useCapture 为 true，则调用该事件处理程序。（IE 中无此阶段。）

目标阶段 触发在目标对象本身注册的事件处理程序，也称正常事件派发阶段。

冒泡阶段 从目标节点到根节点，检测每个节点是否注册了事件处理程序，如果注册了事件处理程序，并且 useCapture 为 false，则调用该事件处理程序。

举例

```
<div id="div1">
  <div id="div2">
    <div id="div3">
      <div id="div4">
      </div>
    </div>
  </div>
```

```
</div>
```

如果在 d3 上点击鼠标，事件流是这样的：

捕获阶段 在 div1 处检测是否有 useCapture 为 true 的事件处理程序，若有，则执行该程序，然后再同样地处理 div2。

目标阶段 在 div3 处，发现 div3 就是鼠标点击的节点，所以这里为目标阶段，若有事件处理程序，则执行该程序，这里不论 useCapture 为 true 还是 false。

冒泡阶段 在 div2 处检测是否有 useCapture 为 false 的事件处理程序，若有，则执行该程序，然后再同样地处理 div1。

注意，上述捕获阶段和冒泡阶段中，实际上 div1 之上还应该有结点，比如有 body，但这里不讨论。

addEventListener—第三个参数 useCapture

addEventListener 有三个参数：第一个参数表示事件名称（不含 on，如 "click"）；第二个参数表示要接收事件处理的函数；第三个参数为 useCapture，本文就讲解它。

```
<div id="outDiv">
  <div id="middleDiv">
    <div id="inDiv">请在此点击鼠标。</div>
  </div>
</div>

<div id="info"></div>
```

(21)
4. 用悲观并发方式处理数据库并发冲突以保证数据一直性的代码处理方法(19)
5. 在数据库中，并发控制有乐观锁和悲观锁之间，什么时候用乐观锁比较好什么时候用悲观锁比较好？(16)

推荐排行榜

1. httpModules 与 httpHandlers(14)
2. GAC(11)
3. C# 静态类(9)
4. addEventListener(9)
5. 采用软件负载均衡器实现web服务器集群(6)

```
var outDiv =
document.getElementById("outDiv");
var middleDiv =
document.getElementById("middleDiv");
var inDiv = document.getElementById("inDiv");
var info = document.getElementById("info");

outDiv.addEventListener("click", function () {
info.innerHTML += "outDiv" + "<br>"; }, false);
middleDiv.addEventListener("click", function () {
info.innerHTML += "middleDiv" + "<br>"; },
false);
inDiv.addEventListener("click", function () {
info.innerHTML += "inDiv" + "<br>"; }, false);
```

上述是我们测试的代码，根据 info 的显示来确定触发的顺序，有三个 addEventListener，而 useCapture 可选值为 true 和 false，所以 2*2*2，可以得出 8 段不同的程序。

- 全为 false 时，触发顺序为：inDiv、middleDiv、outDiv；
- 全为 true 时，触发顺序为：outDiv、middleDiv、inDiv；
- outDiv 为 true，其他为 false 时，触发顺序为：outDiv、inDiv、middleDiv；
- middleDiv 为 true，其他为 false 时，触发顺序为：middleDiv、inDiv、outDiv；
- ……

最终得出如下结论：

- true 的触发顺序总是在 false 之前；
- 如果多个均为 true，则外层的触发先于内层；
- 如果多个均为 false，则内层的触发先于外层。

下面提供全部代码，您可以更改其中的 true、false 值，来进行测试。注意，不适用于 IE。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Language" content="zh-cn" />
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>useCapture</title>
<style type="text/css">
#outDiv
{
   padding:10px 10px 10px 10px;
   border:1px solid red;
}

#middleDiv
{
   padding:10px 10px 10px 10px;
   border:1px solid green;
}

#inDiv
{
   padding:10px 10px 10px 10px;
   border:1px solid blue;
}
</style>
</head>

<body>

<div id="outDiv">
  <div id="middleDiv">
    <div id="inDiv">请在此点击鼠标。</div>
  </div>
```

```
</div>

<div id="info"></div>

<script language="javascript"
type="text/javascript">
<!--

var outDiv =
document.getElementById("outDiv");
var middleDiv =
document.getElementById("middleDiv");
var inDiv = document.getElementById("inDiv");
var info = document.getElementById("info");

outDiv.addEventListener("click", function () {
info.innerHTML += "outDiv" + "<br>"; }, false);
middleDiv.addEventListener("click", function () {
info.innerHTML += "middleDiv" + "<br>"; },
false);
inDiv.addEventListener("click", function () {
info.innerHTML += "inDiv" + "<br>"; }, false);
//-->
</script>

</body>

</html>
```

addEventListener—event 对象的属性和方法

事件触发时，会将一个 Event 对象传递给事件处理程序，比如：

```
document.getElementById("testText").addEventL
istener("keydown", function (event) {
alert(event.keyCode); }, false);
```

事件类型

DOM 事件类型是分为 UIEvent、UIEvent:KeyEvent、UIEvent:MouseEvent，不同的事件有不同的属性和方法，但常用的来说我们都不会用错，比如我们不会在鼠标事件中去取键盘值（Ctrl、Alt、Shift 除外），所以我们没有必要深究。

该对象的属性和方法有：

view 只读，对象，发生事件的 Window 对象。

type 只读，字符串。比如鼠标点击事件的类型：click。

eventPhase 只读，数字，事件流正经历的阶段。1－捕获，2－目标，3－冒泡。

target 只读，对象，派发事件的目标对象。比如鼠标是点击在哪个按钮上的。

currentTarget 只读，对象，当前正在调用监听器的对象，也就是当前 addEventListener 是绑定在哪个对象上的。

timeStamp 只读，数字，用毫秒表示事件发生时距计算机开机的时间。

---

cancelable 只读，布尔，处理事件的默认行为是否可以停止。主要针对一些系统事件，如果值为 true，则 event 的 preventDefault 方法可以使用，否则不可用。

preventDefault() 阻止浏览器的默认行为，比如在文本框中打字触发 keydown，如果 keydown 事件处理程序中调用了 preventDefault()，所打的字就不会跑到文本框中去，注意，此时不要弹出 alert 对话框，否则可能不起作用。IE 中在事件处理程序中用 return false 实现类似功能。

---

bubbles 只读，布尔，事件是否开启冒泡功能。

stopImmediatePropagation 这个东西在 JavaScript 中是个属性，而不是方法，布尔，但具体测试并未发现其用途，不知是不是 bug。

stopPropagation() 停止当前的事件流传播，但不会停止当前正在处理的对象。IE 中用 event.cancelBubble = true 实现类似功能。

cancelBubble 布尔，是否取消冒泡，不建议使用，用 stopPropagation() 代替。

preventBubble() 阻止冒泡，不建议使用，用 stopPropagation() 代替。

preventCapture() 阻止捕获，不建议使用，用 stopPropagation() 代替。

---

detail 只读，数字，提供时间的额外信息，对于 click 事件、mousedown 事件和 mouseup 事件，这个字段代表点击的次数。

isChar 只读，布尔，按下的按键值是否是字符，比如按下 Ctrl 键时，就返回 false。不过您在 Firefox 中测试时，该值总是 false，Firefox 官方已经说明这是一个 bug。

altKey 只读，布尔，是否按下了 Alt 键。

ctrlKey 只读，布尔，是否按下了 Ctrl 键。

shiftKey 只读，布尔，是否按下了 Shift 键。

metaKey 只读，布尔，是否按下了 Meta 键。

下面一些属性很有意思，请仔细区别。

charCode 只读，数字，字符（英文、数字、符号）的 Unicode 值。

- 只用于 keypress。

keyCode 只读，数字，键盘按键值。

- 用于 keypress 时：返回非字符按键值（除 Ctrl、Shift、Alt、Caps Lock、单行文本框中按向上键等）；
- 用于 keydown、keyup 时：返回任意键值。

button 只读，数字，鼠标按键值。

- 用于 click 时：0－左键。
- 用于 mousedown、mouseup 时：0－左键，1－中间键（滚轮），2－右键。

which 只读，数字，键盘按键值或鼠标按键值。

- 用于 keypress 时：等同于 charCode + 回退键 + 回车键；
- 用于 keydown、keyup 时：返回任意键值；
- 用于 click 时：1－左键，与 button 的值略有区别。
- 用于 mousedown、mouseup 时：1－左键，2－中间键（滚轮），3－右键，与 button 的值略有区别。

可以看出，which 只有一点没有包括：那就是 keypress 时，不如 keyCode 那么全，但实际上，keypress 事件中用于非字符键的情况较少，所以一般还是用 which 代替全部。

addEventListener－有用的笔记

为什么用 addEventListener

- 可以对同一物件的同一事件绑定多个事件处理程序。
- 可以通过事件流三个阶段更好地控制何时触发事件处理程序。
- 工作于 DOM 元素，而不仅是 HTML 元素。

事件分发时添加 eventListener

不会立即触发 eventListener，可能会在下一个事件流（比如冒泡阶段）中触发。

多个相同的 eventListener

如下，三个参数完全相同，并且第二个参数不是匿名函数。

```
document.getElementById("myBox").addEventListener("click", Go, false);
document.getElementById("myBox").addEventListener("click", Go, false);
document.getElementById("myBox").addEventListener("click", Go, false);
```

会抛弃多余的，只保留一个，对应的 removeEventListener 也只用一次就可以了（removeEventListener 用法和 addEventListener 完全相同）。

但如果是第二个参数是匿名函数，比如：

```
document.getElementById("outDiv").addEventListener("click", function () {
document.getElementById("info").innerHTML += "1";}, false);
document.getElementById("outDiv").addEventListener("click", function () {
document.getElementById("info").innerHTML += "1";}, false);
document.getElementById("outDiv").addEventListener("click", function () {
document.getElementById("info").innerHTML += "1";}, false);
```

则三个均有效，并且无法用 removeEventListener 除去。

this

事件处理程序中，this 变成了触发事件的控件，但我们仍推荐用 event.target 或 event.currentTarget。

早期的事件监听

在 DOM0 中，我们用 obj.onclick = FuncName，由于兼容性好，应用非常广泛，只是功能不如 addEventListener 强大。

内存问题

前面提到了许多使用域名函数的地方，有时这是没办法的，请参见在各浏览器中动态添加事件－参数篇，但这会导致内存问题。

一旦事件绑定之后，该绑定代码作用域的变量就都保留下来，不会被 JavaScript 引擎回收，这可能会引起占用大量内存的问题，由于 removeEventListener 无法删除匿名函数的事件处理程序，只有在物件（比如按钮）去除之后，该内存才可能得到回收。

« 上一篇：DOS
» 下一篇：arguments.callee 与 函数.caller

posted on 2009-10-19 09:35 chenlulouis 阅读(13143) 评论(2) 编辑 收藏

评论

#1楼 2009-12-20 10:15 阿K&LiveCai

好文章支持下，

顺便转载下：呵呵
http://www.cnblogs.com/qq419524837/articles/1628067.html

支持(0) 反对(0)

#2楼 2016-04-03 22:27 阿林十一

好棒！支持！

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新**IT**新闻:

- 亚马逊或最早下月推流媒体音乐服务 每月9.99美元
- 支付宝能种树了！蚂蚁金服给4.5亿人开通了碳账户
- 榨干诺基亚！微软直板功能机入网
- 校园贷出现新变身 分期消费平台盯上了大学生
- 韩国电脑游戏行业面临危机 因为干不过中国手游

» 更多新闻...

最新知识库文章:

- 程序猿媳妇儿注意事项
- 可是姑娘，你为什么要编程呢？
- 知其所以然（以算法学习为例）
- 如何给变量取个简短且无歧义的名字
- 编程的智慧

» 更多知识库文章...