事件绑定、事件监听、事件委托

github.com/tsrot 事件绑定 事件监听 事件委托

地址：http://blog.xieliqun.com/2016/08/12/event-delegate/

> 在JavaScript的学习中，我们经常会遇到JavaScript的事件机制，例如，事件绑定、事件监听、事件委托（事件代理）等。这些名词是什么意思呢，有什么作用呢？

事件绑定

要想让 JavaScript 对用户的操作作出响应，首先要对 DOM 元素绑定事件处理函数。所谓事件处理函数，就是处理用户操作的函数，不同的操作对应不同的名称。

在JavaScript中，有三种常用的绑定事件的方法：

- 在DOM元素中直接绑定；
- 在JavaScript代码中绑定；
- 绑定事件监听函数。

在DOM中直接绑定事件

我们可以在DOM元素上绑定onclick、onmouseover、onmouseout、onmousedown、onmouseup、ondblclick、onkeydown、onkeypress、onkeyup等。好多不一一列出了。如果想知道更多事件类型请查看，DOM事件。

```
<input type="button" value="click me" onclick="hello()">

<script>
function hello(){
    alert("hello world!");
}
</script>
```

在JavaScript代码中绑定事件

在JavaScript代码中（即`script`标签内）绑定事件可以使JavaScript代码与HTML标签分离，文档结构清晰，便于管理和开发。

```
<input type="button" value="click me" id="btn">

<script>
document.getElementById("btn").onclick = function(){
    alert("hello world!");
}
</script>
```

使用事件监听绑定事件

绑定事件的另一种方法是用 addEventListener() 或 attachEvent() 来绑定事件监听函数。下面详细介绍，事件监听。

事件监听

关于事件监听，W3C规范中定义了3个事件阶段，依次是捕获阶段、目标阶段、冒泡阶段。

起初Netscape制定了JavaScript的一套事件驱动机制（即事件捕获）。随即IE也推出了自己的一套事件驱动机制（即事件冒泡）。最后W3C规范了两种事件机制，分为捕获阶段、目标阶段、冒泡阶段。IE8以前IE一直坚持自己的事件机制（前端人员一直头痛的兼容性问题），IE9以后IE也支持了W3C规范。

W3C规范

语法：

```
element.addEventListener(event, function, useCapture)
```

event：（必需）事件名，支持所有DOM事件。
function：（必需）指定要事件触发时执行的函数。
useCapture：（可选）指定事件是否在捕获或冒泡阶段执行。true，捕获。false，冒泡。默认false。

注：IE8以下不支持。

```
<input type="button" value="click me" id="btn1">

<script>
document.getElementById("btn1").addEventListener("click",hello);
function hello(){
    alert("hello world!");
}
</script>
```

IE标准

语法：

```
element.attachEvent(event, function)
```

event：（必需）事件类型。需加“on“，例如：onclick。
function：（必需）指定要事件触发时执行的函数。

```
<input type="button" value="click me" id="btn2">

<script>
document.getElementById("btn2").attachEvent("onclick",hello);
function hello(){
    alert("hello world!");
}
</script>
```

事件监听的优点

1、可以绑定多个事件。

```
<input type="button" value="click me" id="btn3">

<script>
var btn3 = document.getElementById("btn3");
btn3.onclick = function(){
    alert("hello 1"); //不执行
}
btn3.onclick = function(){
    alert("hello 2"); //执行
}
</script>
```

常规的事件绑定只执行最后绑定的事件。

```
<input type="button" value="click me" id="btn4">

<script>
var btn4 = document.getElementById("btn4");
btn4.addEventListener("click",hello1);
btn4.addEventListener("click",hello2);

function hello1(){
    alert("hello 1");
}
function hello2(){
    alert("hello 2");
}
</script>
```

两个事件都执行了。

2、可以解除相应的绑定

```
<input type="button" value="click me" id="btn5">

<script>
var btn5 = document.getElementById("btn5");
btn5.addEventListener("click",hello1);//执行了
btn5.addEventListener("click",hello2);//不执行
btn5.removeEventListener("click",hello2);

function hello1(){
    alert("hello 1");
}
function hello2(){
    alert("hello 2");
}
</script>
```

封装事件监听

```
<input type="button" value="click me" id="btn5">

//绑定监听事件
function addEventHandler(target,type,fn){
    if(target.addEventListener){
        target.addEventListener(type,fn);
    }else{
        target.attachEvent("on"+type,fn);
    }
}

//移除监听事件
function removeEventHandler(target,type,fn){
    if(target.removeEventListener){
        target.removeEventListener(type,fn);
    }else{
        target.detachEvent("on"+type,fn);
    }
}

//测试
var btn5 = document.getElementById("btn5");
addEventHandler(btn5,"click",hello1);//添加事件hello1
addEventHandler(btn5,"click",hello2);//添加事件hello2
removeEventHandler(btn5,"click",hello1);//移除事件hello1
```

事件委托

事件委托就是利用冒泡的原理，把事件加到父元素或祖先元素上，触发执行效果。

```
<input type="button" value="click me" id="btn6">

var btn6 = document.getElementById("btn6");
document.onclick = function(event){
    event = event || window.event;
    var target = event.target || event.srcElement;
    if(target == btn6){
        alert(btn5.value);
    }
}
```

上面只是个例子，代码尽可能的简化了。在实际的代码中 我们可能用到jQuery的live()、delegate()、bind()、on()等。

事件委托优点

1、提高JavaScript性能。事件委托可以显著的提高事件的处理速度，减少内存的占用。实例分析JavaScript中的事件委托和事件绑定，这篇文章写得还不错。

传统写法

```
<ul id="list">
  <li id="item1" >item1</li>
  <li id="item2" >item2</li>
  <li id="item3" >item3</li>
</ul>

<script>
var item1 = document.getElementById("item1");
var item2 = document.getElementById("item2");
var item3 = document.getElementById("item3");

item1.onclick = function(){
    alert("hello item1");
}
item2.onclick = function(){
    alert("hello item2");
}
item3.onclick = function(){
    alert("hello item3");
}
</script>
```

事件委托

```
<ul id="list">
  <li id="item1" >item1</li>
  <li id="item2" >item2</li>
  <li id="item3" >item3</li>
</ul>

<script>
var item1 = document.getElementById("item1");
var item2 = document.getElementById("item2");
var item3 = document.getElementById("item3");

document.addEventListener("click",function(event){
    var target = event.target;
    if(target == item1){
        alert("hello item1");
    }else if(target == item2){
        alert("hello item2");
    }else if(target == item3){
        alert("hello item3");
    }
})
</script>
```

2、动态的添加DOM元素，不需要因为元素的改动而修改事件绑定。

传统写法

```
<ul id="list">
  <li id="item1" >item1</li>
  <li id="item2" >item2</li>
  <li id="item3" >item3</li>
</ul>

<script>
var list = document.getElementById("list");

var item = list.getElementsByTagName("li");
for(var i=0;i<item.length;i++){
    (function(i){
        item[i].onclick = function(){
            alert(item[i].innerHTML);
        }
    })(i)
}

var node=document.createElement("li");
var textnode=document.createTextNode("item4");
node.appendChild(textnode);
list.appendChild(node);

</script>
```

点击item1到item3都有事件响应，但是点击item4时，没有事件响应。说明传统的事件绑定无法对动态添加的元素而动态的添加事件。

事件委托

```
<ul id="list">
  <li id="item1" >item1</li>
  <li id="item2" >item2</li>
  <li id="item3" >item3</li>
</ul>

<script>
var list = document.getElementById("list");

document.addEventListener("click",function(event){
    var target = event.target;
    if(target.nodeName == "LI"){
        alert(target.innerHTML);
    }
})

var node=document.createElement("li");
var textnode=document.createTextNode("item4");
node.appendChild(textnode);
list.appendChild(node);

</script>
```

当点击item4时，item4有事件响应。说明事件委托可以为新添加的DOM元素动态的添加事件。