[**js中的事件委托或是事件代理详解**](http://www.cnblogs.com/liugang-vip/p/5616484.html)

起因：

1、这是前端面试的经典题型，要去找工作的小伙伴看看还是有帮助的；

2、其实我一直都没弄明白，写这个一是为了备忘，二是给其他的知其然不知其所以然的小伙伴们以参考；

概述：

那什么叫事件委托呢？它还有一个名字叫事件代理，JavaScript高级程序设计上讲：事件委托就是利用事件冒泡，只指定一个事件处理程序，就可以管理某一类型的所有事件。那这是什么意思呢？网上的各位大牛们讲事件委托基本上都用了同一个例子，就是取快递来解释这个现象，我仔细揣摩了一下，这个例子还真是恰当，我就不去想别的例子来解释了，借花献佛，我摘过来，大家认真领会一下事件委托到底是一个什么原理：

有三个同事预计会在周一收到快递。为签收快递，有两种办法：一是三个人在公司门口等快递；二是委托给前台MM代为签收。现实当中，我们大都采用委托的方案（公司也不会容忍那么多员工站在门口就为了等快递）。前台MM收到快递后，她会判断收件人是谁，然后按照收件人的要求签收，甚至代为付款。这种方案还有一个优势，那就是即使公司里来了新员工（不管多少），前台MM也会在收到寄给新员工的快递后核实并代为签收。

这里其实还有2层意思的：

第一，现在委托前台的同事是可以代为签收的，即程序中的现有的dom节点是有事件的；

第二，新员工也是可以被前台MM代为签收的，即程序中新添加的dom节点也是有事件的。

为什么要用事件委托：

一般来说，dom需要有事件处理程序，我们都会直接给它设事件处理程序就好了，那如果是很多的dom需要添加事件处理呢？比如我们有100个li，每个li都有相同的click点击事件，可能我们会用for循环的方法，来遍历所有的li，然后给它们添加事件，那这么做会存在什么影响呢？

在JavaScript中，添加到页面上的事件处理程序数量将直接关系到页面的整体运行性能，因为需要不断的与dom节点进行交互，访问dom的次数越多，引起浏览器重绘与重排的次数也就越多，就会延长整个页面的交互就绪时间，这就是为什么性能优化的主要思想之一就是减少DOM操作的原因；如果要用事件委托，就会将所有的操作放到js程序里面，与dom的操作就只需要交互一次，这样就能大大的减少与dom的交互次数，提高性能；

每个函数都是一个对象，是对象就会占用内存，对象越多，内存占用率就越大，自然性能就越差了（内存不够用，是硬伤，哈哈），比如上面的100个li，就要占用100个内存空间，如果是1000个，10000个呢，那只能说呵呵了，如果用事件委托，那么我们就可以只对它的父级（如果只有一个父级）这一个对象进行操作，这样我们就需要一个内存空间就够了，是不是省了很多，自然性能就会更好。

事件委托的原理：

事件委托是利用事件的冒泡原理来实现的，何为事件冒泡呢？就是事件从最深的节点开始，然后逐步向上传播事件，举个例子：页面上有这么一个节点树，div>ul>li>a;比如给最里面的a加一个click点击事件，那么这个事件就会一层一层的往外执行，执行顺序a>li>ul>div，有这样一个机制，那么我们给最外面的div加点击事件，那么里面的ul，li，a做点击事件的时候，都会冒泡到最外层的div上，所以都会触发，这就是事件委托，委托它们父级代为执行事件。

事件委托怎么实现：

终于到了本文的核心部分了，哈哈，在介绍事件委托的方法之前，我们先来看一段一般方法的例子：

子节点实现相同的功能：

<ul id="ul1">

<li>111</li>

<li>222</li>

<li>333</li>

<li>444</li>

</ul>

实现功能是点击li，弹出123：

[复制代码](javascript:void(0);)

window.onload = function(){

var oUl = document.getElementById("ul1");

var aLi = oUl.getElementsByTagName('li');

for(var i=0;i<aLi.length;i++){

aLi[i].onclick = function(){

alert(123);

}

}

}

 上面的代码的意思很简单，相信很多人都是这么实现的，我们看看有多少次的dom操作，首先要找到ul，然后遍历li，然后点击li的时候，又要找一次目标的li的位置，才能执行最后的操作，每次点击都要找一次li；

那么我们用事件委托的方式做又会怎么样呢？

window.onload = function(){

var oUl = document.getElementById("ul1");

oUl.onclick = function(){

alert(123);

}

}

 这里用父级ul做事件处理，当li被点击时，由于冒泡原理，事件就会冒泡到ul上，因为ul上有点击事件，所以事件就会触发，当然，这里当点击ul的时候，也是会触发的，那么问题就来了，如果我想让事件代理的效果跟直接给节点的事件效果一样怎么办，比如说只有点击li才会触发，不怕，我们有绝招：

Event对象提供了一个属性叫target，可以返回事件的目标节点，我们成为事件源，也就是说，target就可以表示为当前的事件操作的dom，但是不是真正操作dom，当然，这个是有兼容性的，标准浏览器用ev.target，IE浏览器用event.srcElement，此时只是获取了当前节点的位置，并不知道是什么节点名称，这里我们用nodeName来获取具体是什么标签名，这个返回的是一个大写的，我们需要转成小写再做比较（习惯问题）：

window.onload = function(){  
　　var oUl = document.getElementById("ul1");  
　　oUl.onclick = function(ev){  
　　　　var ev = ev || window.event;  
　　　　var target = ev.target || ev.srcElement;  
　　　　if(target.nodeName.toLowerCase() == 'li'){  
　 　　　　　　 alert(123);  
　　　　　　　  alert(target.innerHTML);  
　　　　}  
　　}  
}

 这样改下就只有点击li会触发事件了，且每次只执行一次dom操作，如果li数量很多的话，将大大减少dom的操作，优化的性能可想而知！

上面的例子是说li操作的是同样的效果，要是每个li被点击的效果都不一样，那么用事件委托还有用吗？

<div id="box">

<input type="button" id="add" value="添加" />

<input type="button" id="remove" value="删除" />

<input type="button" id="move" value="移动" />

<input type="button" id="select" value="选择" />

</div>

window.onload = function(){

var Add = document.getElementById("add");

var Remove = document.getElementById("remove");

var Move = document.getElementById("move");

var Select = document.getElementById("select");

Add.onclick = function(){

alert('添加');

};

Remove.onclick = function(){

alert('删除');

};

Move.onclick = function(){

alert('移动');

};

Select.onclick = function(){

alert('选择');

}

}

上面实现的效果我就不多说了，很简单，4个按钮，点击每一个做不同的操作，那么至少需要4次dom操作，如果用事件委托，能进行优化吗？

window.onload = function(){

var oBox = document.getElementById("box");

oBox.onclick = function (ev) {

var ev = ev || window.event;

var target = ev.target || ev.srcElement;

if(target.nodeName.toLocaleLowerCase() == 'input'){

switch(target.id){

case 'add' :

alert('添加');

break;

case 'remove' :

alert('删除');

break;

case 'move' :

alert('移动');

break;

case 'select' :

alert('选择');

break;

}

}

}

}

用事件委托就可以只用一次dom操作就能完成所有的效果，比上面的性能肯定是要好一些的

 现在讲的都是document加载完成的现有dom节点下的操作，那么如果是新增的节点，新增的节点会有事件吗？也就是说，一个新员工来了，他能收到快递吗？

看一下正常的添加节点的方法：

<input type="button" name="" id="btn" value="添加" />

<ul id="ul1">

<li>111</li>

<li>222</li>

<li>333</li>

<li>444</li>

</ul>

现在是移入li，li变红，移出li，li变白，这么一个效果，然后点击按钮，可以向ul中添加一个li子节点

window.onload = function(){

var oBtn = document.getElementById("btn");

var oUl = document.getElementById("ul1");

var aLi = oUl.getElementsByTagName('li');

var num = 4;

//鼠标移入变红，移出变白

for(var i=0; i<aLi.length;i++){

aLi[i].onmouseover = function(){

this.style.background = 'red';

};

aLi[i].onmouseout = function(){

this.style.background = '#fff';

}

}

//添加新节点

oBtn.onclick = function(){

num++;

var oLi = document.createElement('li');

oLi.innerHTML = 111\*num;

oUl.appendChild(oLi);

};

}

这是一般的做法，但是你会发现，新增的li是没有事件的，说明添加子节点的时候，事件没有一起添加进去，这不是我们想要的结果，那怎么做呢？一般的解决方案会是这样，将for循环用一个函数包起来，命名为mHover，如下：

window.onload = function(){

var oBtn = document.getElementById("btn");

var oUl = document.getElementById("ul1");

var aLi = oUl.getElementsByTagName('li');

var num = 4;

function mHover () {

//鼠标移入变红，移出变白

for(var i=0; i<aLi.length;i++){

aLi[i].onmouseover = function(){

this.style.background = 'red';

};

aLi[i].onmouseout = function(){

this.style.background = '#fff';

}

}

}

mHover ();

//添加新节点

oBtn.onclick = function(){

num++;

var oLi = document.createElement('li');

oLi.innerHTML = 111\*num;

oUl.appendChild(oLi);

mHover ();

};

}

虽然功能实现了，看着还挺好，但实际上无疑是又增加了一个dom操作，在优化性能方面是不可取的，那么有事件委托的方式，能做到优化吗？

window.onload = function(){

var oBtn = document.getElementById("btn");

var oUl = document.getElementById("ul1");

var aLi = oUl.getElementsByTagName('li');

var num = 4;

//事件委托，添加的子元素也有事件

oUl.onmouseover = function(ev){

var ev = ev || window.event;

var target = ev.target || ev.srcElement;

if(target.nodeName.toLowerCase() == 'li'){

target.style.background = "red";

}

};

oUl.onmouseout = function(ev){

var ev = ev || window.event;

var target = ev.target || ev.srcElement;

if(target.nodeName.toLowerCase() == 'li'){

target.style.background = "#fff";

}

};

//添加新节点

oBtn.onclick = function(){

num++;

var oLi = document.createElement('li');

oLi.innerHTML = 111\*num;

oUl.appendChild(oLi);

};

}

看，上面是用事件委托的方式，新添加的子元素是带有事件效果的，我们可以发现，当用事件委托的时候，根本就不需要去遍历元素的子节点，只需要给父级元素添加事件就好了，其他的都是在js里面执行，这样可以大大的减少dom操作，这才是事件委托的精髓所在。

--------------------------------------------------华丽的分割线-------------- -----------------------------------------------------------------------------------------------------

在这里先感谢一下@苍茫大地NV 的提问，提的问题非常好！👏👏👏

他的问题是：

现在给一个场景 ul > li > div > p，div占满li，p占满div，还是给ul绑定事件，需要判断点击的是不是li（假设li里面的结构是不固定的），那么e.target就可能是p，也有可能是div，这种情况你会怎么处理呢？

那我们现在就再现一下他给的场景

　　<ul id="test">

<li>

<p>11111111111</p>

</li>

<li>

<div>

22222222

</div>

</li>

<li>

<span>3333333333</span>

</li>

<li>4444444</li>

</ul>

如上列表，有4个li，里面的内容各不相同，点击li，event对象肯定是当前点击的对象，怎么指定到li上，下面我直接给解决方案：

　　var oUl = document.getElementById('test');

oUl.addEventListener('click',function(ev){

var target = ev.target;

while(target !== oUl ){

if(target.tagName.toLowerCase() == 'li'){

console.log('li click~');

break;

}

target = target.parentNode;

}

})

 核心代码是while循环部分，实际上就是一个递归调用，你也可以写成一个函数，用递归的方法来调用，同时用到冒泡的原理，从里往外冒泡，直到currentTarget为止，当当前的target是li的时候，就可以执行对应的事件了，然后终止循环，恩，没毛病！

这里看不到效果，大家可以复制过去运行一下！

--------------------------------------------------------------------华丽的分割线----------------------------------------------------------------

 总结：

那什么样的事件可以用事件委托，什么样的事件不可以用呢？

适合用事件委托的事件：click，mousedown，mouseup，keydown，keyup，keypress。

值得注意的是，mouseover和mouseout虽然也有事件冒泡，但是处理它们的时候需要特别的注意，因为需要经常计算它们的位置，处理起来不太容易。

不适合的就有很多了，举个例子，mousemove，每次都要计算它的位置，非常不好把控，再比如说focus，blur之类的，本身就没有冒泡的特性，自然就不能用事件委托了。

好了，今天就到这里，下次我想介绍一下事件绑定，欢迎大家关注和阅读，以上纯属个人见解，如有不对的地方，万望指正，不胜感谢！