**事件委托**

**事件委托定义:**

本来是要注册给自己的事件, 注册给了父元素. 事件触发后的事情, 委托给父元素执行.

**事件委托原理: 事件冒泡**

代码节选:

var ul = document.getElementById( 'ul' );

ul.onclick = function( e ){

if( e.target === ul ){

return;

}

console.log( e.target );

}

**具体实现过程:**

当我们点击ul中的li时, 浏览器捕捉到的事件源是那个被点击的li, 所以e.target就代表li. (这里应该注意, 即使我们没有给li注册点击事件, 但是当我们点击这个li时, 是有点击事件发生的, 只是说我们没有给这个li注册点击事件来显式地捕捉这个事件, 但是浏览器是会自动捕捉到的)

而e这个对象是会被事件处理函数接收的 ( 至于为什么事件处理函数会接收这个事件对象e? 这就是js中规定好的 ), 而li又没有事件处理函数, 所以它只能等一个对应的事件处理函数(这里因为是li被点击, 所以一定要是点击事件处理函数)出现, 并且有显式的传一个参数像这样: function(e){...}, 这样它就一定能到这个函数里被接收了.

接着ul虽然没被点击, 但是因为事件冒泡的原因, 也会被触发. 虽然它被触发点击事件, 但是它没有被点击, 它不是事件源, 所以浏览器中不会创建一个事件对象e来传给ul的事件处理函数, 而浏览器给li创建的事件对象e就会被ul的事件处理函数捕获(因为这个e被浏览器创建出来就是为了给对应的事件处理函数捕获的(这里是点击事件)), 从而我们可以在ul的点击事件处理函数中做一些关于它的子元素li的一些操作.

这就是事件委托的基于事件冒泡的原理.(我感觉是…).

拓展:

代码性能优化 🡪其它优化问题,比如回流重绘, 封装等

动态创建的元素绑定事件只能用事件委托 🡪模板🡪数据交互🡪ajax…