js 添加事件 attachEvent 和 addEventListener 的用法

一般我们在JS中添加事件,是这样子的

obj.onclick=method

这种绑定事件的方式，兼容主流浏览器,但如果一个元素上添加多次同一事件呢?

obj.onclick=method1;

obj.onclick=method2;

obj.onclick=method3;

如果这样写,那么只有最后绑定的事件,这里是method3会被执行,这个时候我们就不能用onclick这样的写法了,主角改登场了,在IE中我们可以使用attachEvent方法

//object.attachEvent(event,function);

btn1Obj.attachEvent("onclick",method1);

btn1Obj.attachEvent("onclick",method2);

btn1Obj.attachEvent("onclick",method3);

使用格式是前面是事件类型,注意的是需要加on,比如onclick,onsubmit,onchange,执行顺序是

method3->method2->method1

可惜这个微软的私人方法,火狐和其他浏览器都不支持,幸运的是他们都支持W3C标准的addEventListener方法

//element.addEventListener(type,listener,useCapture);

btn1Obj.addEventListener("click",method1,false);

btn1Obj.addEventListener("click",method2,false);

btn1Obj.addEventListener("click",method3,false);

执行顺序为method1->method2->method3

做前端开发工程师,最悲剧的某过于浏览器兼容问题了,上面有两种添加事件的方法,为了同一添加事件的方法,我们不得不再重新写一个通用的添加事件函数,幸亏再有前人帮我们做了这件事

function addEvent(elm, evType, fn, useCapture) {

if (elm.addEventListener) {

elm.addEventListener(evType, fn, useCapture);//DOM2.0

return true;

}

else if (elm.attachEvent) {

var r = elm.attachEvent(‘on‘ + evType, fn);//IE5+

return r;

}

else {

elm['on' + evType] = fn;//DOM 0

}

}

下面是Dean Edwards 的版本

function addEvent(element, type, handler) {

//为每一个事件处理函数分派一个唯一的ID

if (!handler.$$guid) handler.$$guid = addEvent.guid++;

//为元素的事件类型创建一个哈希表

if (!element.events) element.events = {};

//为每一个"元素/事件"对创建一个事件处理程序的哈希表

var handlers = element.events[type];

if (!handlers) {

handlers = element.events[type] = {};

//存储存在的事件处理函数(如果有)

if (element["on" + type]) {

handlers[0] = element["on" + type];

}

}

//将事件处理函数存入哈希表

handlers[handler.$$guid] = handler;

//指派一个全局的事件处理函数来做所有的工作

element["on" + type] = handleEvent;

};

//用来创建唯一的ID的计数器

addEvent.guid = 1;

function removeEvent(element, type, handler) {

//从哈希表中删除事件处理函数

if (element.events && element.events[type]) {

delete element.events[type][handler.$$guid];

}

};

function handleEvent(event) {

var returnValue = true;

//抓获事件对象(IE使用全局事件对象)

event = event || fixEvent(window.event);

//取得事件处理函数的哈希表的引用

var handlers = this.events[event.type];

//执行每一个处理函数

for (var i in handlers) {

this.$$handleEvent = handlers[i];

if (this.$$handleEvent(event) === false) {

returnValue = false;

}

}

return returnValue;

};

//为IE的事件对象添加一些“缺失的”函数

function fixEvent(event) {

//添加标准的W3C方法

event.preventDefault = fixEvent.preventDefault;

event.stopPropagation = fixEvent.stopPropagation;

return event;

};

fixEvent.preventDefault = function() {

this.returnValue = false;

};

fixEvent.stopPropagation = function() {

this.cancelBubble = true;

};

功能非常强悍，解决IE的this指向问题，event总是作为第一个参数传入，跨浏览器就更不在话下。

最后贡献一个HTML5工作组的版本：

var addEvent=(function(){

if(document.addEventListener){

return function(el,type,fn){

if(el.length){

for(var i=0;i&lt;el.length;i++){

addEvent(el[i],type,fn);

}

}else{

el.addEventListener(type,fn,false);

}

};

}else{

return function(el,type,fn){

if(el.length){

for(var i=0;i&lt;el.length;i++){

addEvent(el[i],type,fn);

}

}else{

el.attachEvent(‘on‘+type,function(){

return fn.call(el,window.event);

});

}

};

}

})();

可能细心的读者发现了IE的attachEvent和W3C标准的addEventListener绑定多个事件的执行顺序是不一样的