自定义事件

1. 什么是自定义事件？

自定义事件就是有别于浏览器特定的行为事件（类似click ，mouseover，mouseout，submit等事件）事件名称可以随意定义，可以通过特定的方法今进行添加，触发，删除。

1. JS自定义事件

**函数式实现**

var \_listener = {};

var addEvent = function(type, fn) {

// 添加

};

var fireEvent = function(type) {

// 触发

};

var removeEvent = function(type, fn) {

// 删除

};

addEvent("alert", function() {

alert("弹出！");

});

// 触发自定义alert事件

fireEvent("alert");

但是，函数式写法缺点显而易见，过多暴露在外的全局变量（全局变量是魔鬼），方法无级联等。这也是上面懒得显示完整代码的原因，略知即可。

**字面量实现**

Var Event ={

\_listeners:{},

addEvent：function(type,fn){

if(this.\_listeners[type]===”undefined”){

This.\_listeners[type]=[];

}

If(typeof fn ===”function” ){

This.\_listeners[type].push(fn);

}

return this;

},

firstEvent:function(type){

Var arrayEvent = this.\_listeners[type];

If(arrayEvent instanceof Array){

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

If(typeof arrayEvent[i]===”function”){

arrayEvent[i]({type:type});

}

}

}

return this;

},

removeEvent:function(type ,fn){

var arrayEvent=this.\_listeners[type];

If(typeof type===”string”&&arrayEvent instsnceof Array){

If(typeof fn===”function”){

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

If(arrayEvent[i]===fn){

this.\_listeners[type].splice(i,1);

break;

}

}

}

else{

delete this.\_listeners[type];

}

}

return this;

}

}；

使用这个自定义事件：

增加

Event.addEvent(“alert”,function(){

alert(“弹出”);

});

使用

Event.firstEvent(“alert”);

**原型模式实现**

var EventTarget = function() {

this.\_listener = {};

};

EventTarget.prototype = {

constructor: this,

addEvent: function(type, fn) {

if (typeof type === "string" && typeof fn === "function") {

if (typeof this.\_listener[type] === "undefined") {

this.\_listener[type] = [fn];

} else {

this.\_listener[type].push(fn);

}

}

return this;

},

addEvents: function(obj) {

obj = typeof obj === "object"? obj : {};

var type;

for (type in obj) {

if ( type && typeof obj[type] === "function") {

this.addEvent(type, obj[type]);

}

}

return this;

},

fireEvent: function(type) {

if (type && this.\_listener[type]) {

var events = {

type: type,

target: this

};

for (var length = this.\_listener[type].length, start=0; start<length; start+=1) {

this.\_listener[type][start].call(this, events);

}

}

return this;

},

fireEvents: function(array) {

if (array instanceof Array) {

for (var i=0, length = array.length; i<length; i+=1) {

this.fireEvent(array[i]);

}

}

return this;

},

removeEvent: function(type, key) {

var listeners = this.\_listener[type];

if (listeners instanceof Array) {

if (typeof key === "function") {

for (var i=0, length=listeners.length; i<length; i+=1){

if (listeners[i] === key){

listeners.splice(i, 1);

break;

}

}

} else if (key instanceof Array) {

for (var lis=0, lenkey = key.length; lis<lenkey; lis+=1) {

this.removeEvent(type, key[lenkey]);

}

} else {

delete this.\_listener[type];

}

}

return this;

},

removeEvents: function(params) {

if (params instanceof Array) {

for (var i=0, length = params.length; i<length; i+=1) {

this.removeEvent(params[i]);

}

} else if (typeof params === "object") {

for (var type in params) {

this.removeEvent(type, params[type]);

}

}

return this;

}

};

//------------- 以下为测试代码 -----------

var myEvents = new EventTarget();

myEvents.addEvents({

"once": function() {

alert("该弹框只会出现一次！");

this.removeEvent("once");

},

"infinity": function() {

alert("每次点击页面，该弹框都会出现！");

}

});

document.onclick = function(e) {

e = e || window.event;

var target = e.target || e.srcElement;

if (!target || !/input|pre/i.test(target.tagName)) {

myEvents.fireEvents(["once", "infinity"]);

}

};

### 三、DOM自定义事件

HTMLElement.prototype.addEvent = function(type, fn, capture) {

var el = this;

if (window.addEventListener) {

el.addEventListener(type, function(e) {

fn.call(el, e);

}, capture);

} else if (window.attachEvent) {

el.attachEvent("on" + type, function(e) {

fn.call(el, e);

});

}

};

elImage.addEvent("click", function() {

alert("我是点击图片之后的弹出！");

});

**这个事件定义是不包括IE6/7**

/zxx: 上面代码中的HTMLElement表示HTML元素。以一个<p>标签元素举例，其向上寻找原型对象用过会是这样：HTMLParagraphElement.prototype → HTMLElement.prototype → Element.prototype → Node.prototype → Object.prototype → null。这下您应该知道HTMLElement所处的位置了吧，上述代码HTMLElement直接换成Element也是可以的，但是会让其他元素（例如文本元素）也扩展addEvent方法，有些浪费了。

直接在DOM上进行事件方法扩展其实是个糟糕的做法