**js事件监听机制(事件捕获)总结**

添加事件的js方法也很多，有直接加到页面结构上的，有使用一些js事件监听的方法，由于各个浏览器对事件冒泡事件监听的机制不同

在前端开发过程中我们经常会遇到给页面元素添加事件的问题，添加事件的js方法也很多，有直接加到页面结构上的，有使用一些js事件监听的方法，由于各个浏览器对事件冒泡事件监听的机制不同，le浏览器只有事件冒泡，没有事件监听的机制，对于事件监听的兼容性问题是最大的难题：

1.直接把事件的方法写在页面结构上

[?](http://www.jb51.net/article/53438.htm)

|  |  |
| --- | --- |
| 1  2  3  4 | function eventfun(){  //console.log(this);  }  <input type="button" onclick="eventfun()" value="button" />//这里涉及到一个this作用域的问题，eventfun再这里是一个全局函数, 对象是[object Window],this指向的是window. |

要解决this作用域的问题，可以使用为全局函数添加event变量的方法，在页面结构上将this对象作为参数传递到函数内部使用

[?](http://www.jb51.net/article/53438.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <input type="button" onclick="eventfun2(this)" value="button2" />  function eventfun2(eve){//在这里把事件对象作为参数传递到全局方法里  eve.name="alex";  window.name="robin";  console.log(this);//[object Window]  console.log(eve);// [object HTMLInputElement]  console.log(this.name);// robin  console.log(eve.name);// alexvar  self=eve;  console.log(this.name);//robin  console.log(self.name);//alex  alert(window.name);  alert(self.name);  } |

2. 使用给事件属性赋值的方法,是一种为事件绑定的方法，但是这种方法的局限性就是只能为事件绑定一个方法，如果绑定多个就会以后一个方法为准

HTMLElementobject.onclick = fucntion(){//使用这种为事件属性赋值的方法，this的指针会指向window对象，而不是被事件对象，所以这种方法是引用

[?](http://www.jb51.net/article/53438.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | //js code  fun1();  fun2();  fun3();  console.log(this);//window.object  }  function dosomething(){  //js code  }  HTMLElementobject.onclick = dosomething;//使用这种为事件对象属性赋值的形式，this指针指向事件执行对象  console.log(this);//htmlElementObject |

3.事件传播——冒泡与捕获  
DOM事件标准定义了两种事件流，这两种事件流有着显著的不同并且可能对你的应用有着相当大的影响。这两种事件流分别是捕获和冒泡。和许多Web技 术一样，在它们成为标准之前，Netscape和微软各自不同地实现了它们。Netscape选择实现了捕获事件流，微软则实现了冒泡事件流。幸运的 是，W3C决定组合使用这两种方法,并且大多数新浏览器都遵循这两种事件流方式。  
默认情况下，事件使用冒泡事件流，不使用捕获事件流。然而，在Firefox和Safari里，你可以显式的指定使用捕获事件流，方法是在注册事件时传入useCapture参数，将这个参数设为true。  
冒泡事件流  
当事件在某一DOM元素被触发时，例如用户在客户名字节点上点击鼠标，事件将跟随着该节点继承自的各个父节点冒泡穿过整个的DOM节点层次，直到它 遇到依附有该事件类型处理器的节点，此时，该事件是onclick事件。在冒泡过程中的任何时候都可以终止事件的冒泡，在遵从W3C标准的浏览器里可以通 过调用事件对象上的stopPropagation()方法，在Internet Explorer里可以通过设置事件对象的cancelBubble属性为true。如果不停止事件的传播，事件将一直通过DOM冒泡直至到达文档根。  
捕获事件流  
事件的处理将从DOM层次的根开始，而不是从触发事件的目标元素开始，事件被从目标元素的所有祖先元素依次往下传递。在这个过程中，事件会被从文档 根到事件目标元素之间各个继承派生的元素所捕获，如果事件监听器在被注册时设置了useCapture属性为true,那么它们可以被分派给这期间的任何 元素以对事件做出处理；否则，事件会被接着传递给派生元素路径上的下一元素，直至目标元素。事件到达目标元素后，它会接着通过DOM节点再进行冒泡。  
现代事件绑定方法  
针对如上节课所讨论的，使用传统事件绑定有许多缺陷，比如不能在一个对象的相同事件上注册多个事件处理函数。而浏览器和W3C也并非没有考虑到这一点，因此在现代浏览器中，它们有自己的方法绑定事件。  
W3C DOM  
obj.addEventListener(evtype,fn,useCapture)——W3C提供的添加事件处理函数的方法。obj是要添 加事件的对象，evtype是事件类型，不带on前缀，fn是事件处理函数，如果useCapture是true，则事件处理函数在捕获阶段被执行，否则 在冒泡阶段执行  
obj.removeEventListener(evtype,fn,useCapture)——W3C提供的删除事件处理函数的方法  
微软IE方法  
obj.attachEvent(evtype,fn)——IE提供的添加事件处理函数的方法。obj是要添加事件的对象，evtype是事件类型，带on前缀，fn是事件处理函数，IE不支持事件捕获  
obj.detachEvent(evtype,fn,)——IE提供的删除事件处理函数的方法，evtype包含on前缀

整合两者的方法

[?](http://www.jb51.net/article/53438.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | function addEvent(obj,evtype,fn,useCapture) {  if (obj.addEventListener) {  obj.addEventListener(evtype,fn,useCapture);  } else {  obj.attachEvent("on"+evtype,fn);//IE不支持事件捕获  } else {  obj["on"+evtype]=fn;//事实上这种情况不会存在  }  }  function delEvent(obj,evtype,fn,useCapture) {  if (obj.removeEventListener) {  obj.removeEventListener(evtype,fn,useCapture);  } else {  obj.detachEvent("on"+evtype,fn);  } else {  obj["on"+evtype]=null;  }  } |

IE的attach方法有个问题，就是使用attachEvent时在事件处理函数内部，this指向了window，而不是obj！当然，这个是有解决方案的！

但IE的attachEvent方法有另外一个问题，同一个函数可以被注册到同一个对象同一个事件上多次，解决方法：抛弃IE的 attachEvent方法吧！IE下的attachEvent方法不支持捕获，和传统事件注册没多大区别(除了能绑定多个事件处理函数)，并且IE的 attachEvent方法存在内存泄漏问题！  
addEvent,delEvent现代版

[?](http://www.jb51.net/article/53438.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291 | <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "<http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd>">  <html xmlns="<http://www.w3.org/1999/xhtml>">  <head>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  <title>js事件监听</title>  <style>  table td{font:12px; border-bottom:1px solid #efefef;}  </style>  </head>    <body>  <div id="outEle" style="padding:10px; border:1px solid #b2b2b2; background:#efefef;">  <input type="button" onclick="eventfun()" id="button" value="button" /><br />  <input type="button" onclick="eventfun2(this);" id="button2" value="button2" /><br />  <input type="button" id="button3" value="button3" /><br />  <input type="button" id="button4" value="button4" /><br />  <table id="htmlEleTable" width="100%" border="0" style="border:1px solid #b2b2b2; background:#fff;">  <tr id="1111"><td>111111111111111111111111111111</td></tr>  <tr id="22222"><td>222222222222222222222222222222</td></tr>  <tr id="33333"><td>333333333333333333333333333333</td></tr>  <tr id="4444"><td>444444444444444444444444444444</td></tr>  <tr id="55555"><td>555555555555555555555555555555</td></tr>  </table>  </div>  <script language="javascript" type="text/javascript">  function eventfun(){//1.直接把js方法写在页面结构上  console.log(this);//这里涉及到一个this作用域的问题，eventfun再这里是一个全局函数, 对象是window,this指向的是window  alert(this);  }  function eventfun2(eve){//在这里把事件对象作为参数传递到全局方法里  eve.name="alex";//  window.name="robin";  console.log(this);//[object Window]  console.log(eve);// [object HTMLInputElement]  console.log(this.name);// robin  console.log(eve.name);// alex  var self=eve;  console.log(this.name);//robin  console.log(self.name);//alex  alert(window.name);  alert(self.name);  }  function eventfun3(){//1.直接把js方法写在页面结构上  console.log(this);//这里涉及到一个this作用域的问题，eventfun再这里是一个全局函数, 对象是window,this指向的是window  console.log(this.id);  alert(this);  alert(this.id);  //var outEleObj = EventUtil.$("outEle");  //removeEvent(outEleObj,"click",eventfun3);  }  /\*  var EventUtil = {};  EventUtil.$ = function(id){  return document.getElementById(id);  }  EventUtil.openmes = eventfun3;  EventUtil.addEventHandle = function(eventTarget,eventtype,eventHandle){//定义事件监听的对象元素,事件类型，事件函数  if(eventTarget.attachEvent){  eventTarget.attachEvent("on"+eventtype,eventHandle);  }else if(eventTarget.addEventListener){  eventTarget.addEventListener(eventtype,eventHandle,false)  }else{  eventTarget["on" + eventtype] = null;  }    };  EventUtil.deleEventHandle = function(eventTarget,eventtype,eventHandle){//定义事件监听的对象元素,事件类型，事件函数  if(eventTarget.detachEvent){  alert("on"+eventtype);  alert("on"+eventHandle);  eventTarget.detachEvent("on"+eventtype,eventHandle);  }else if(eventTarget.removeEventListener){  eventTarget.removeEventListener(eventtype,eventHandle,false)  }else{  eventTarget["on" + eventtype] = null;  }    };\*/  var EventUtil={  $:function(id){  return document.getElementById(id);  },  but4fun:function(){  console.log(this);  this.addEventHandle();  },  eventfun3:function (){  console.log(this);  alert(this);  delEvent(obj,evtype,fn,useCapture);  }  }  /\*\*\*使用addEventListener,attachEvent进行dom事件的监听  function addEvent(obj,evtype,fn,useCapture){  if (obj.addEventListener) {  obj.addEventListener(evtype,fn,useCapture);  }else if(obj.attachEvent){  obj.attachEvent("on"+evtype,function () {  fn.call(obj);  });  }else {  obj["on"+evtype]=fn;//事实上这种情况不会存在  }  }  function delEvent(obj,evtype,fn,useCapture) {  if (obj.removeEventListener) {  obj.removeEventListener(evtype,fn,useCapture);  } else if(obj.detachEvent){  obj.detachEvent("on"+evtype,fn);  } else {  obj["on"+evtype]=null;  }  }        function addEvent(obj,evtype,fn,useCapture) {  if (obj.addEventListener) {//优先考虑W3C事件注册方案  obj.addEventListener(evtype,fn,!!useCapture);  } else {//当不支持addEventListener时(IE),由于IE同时也不支持捕获,所以不如使用传统事件绑定  if (!fn.\_\_EventID) {fn.\_\_EventID = addEvent.\_\_EventHandlesCounter++;}  //为每个事件处理函数分配一个唯一的ID    if (!obj.\_\_EventHandles) {obj.\_\_EventHandles={};}  //\_\_EventHandles属性用来保存所有事件处理函数的引用    //按事件类型分类  if (!obj.\_\_EventHandles[evtype]) {//第一次注册某事件时  obj.\_\_EventHandles[evtype]={};  if (obj["on"+evtype]) {//以前曾用传统方式注册过事件处理函数  (obj.\_\_EventHandles[evtype][0]=obj["on"+evtype]).\_\_EventID=0;//添加到预留的0位  //并且给原来的事件处理函数增加一个ID  }  obj["on"+evtype]=addEvent.execEventHandles;  //当事件发生时，execEventHandles遍历表obj.\_\_EventHandles[evtype]并执行其中的函数  }  }  }    addEvent.\_\_EventHandlesCounter=1;//计数器,0位预留它用  addEvent.execEventHandles = function (evt) {//遍历所有的事件处理函数并执行  if (!this.\_\_EventHandles) {return true;}  evt = evt || window.event;  var fns = this.\_\_EventHandles[evt.type];  for (var i in fns) {  fns[i].call(this);  }  };  /\*  function delEvent(obj,evtype,fn,useCapture) {  if (obj.removeEventListener) {//先使用W3C的方法移除事件处理函数  obj.removeEventListener(evtype,fn,!!useCapture);  } else {  if (obj.\_\_EventHandles) {  var fns = obj.\_\_EventHandles[evtype];  if (fns) {delete fns[fn.\_\_EventID];}  }  }  }    function fixEvent(evt) {//fixEvent函数不是单独执行的，它必须有一个事件对象参数，而且只有事件发生时它才被执行！最好的方法是把它整合到addEvent函数的execEventHandles里面  if (!evt.target) {  evt.target = evt.srcElement;  evt.preventDefault = fixEvent.preventDefault;  evt.stopPropagation = fixEvent.stopPropagation;  if (evt.type == "mouseover") {  evt.relatedTarget = evt.fromElement;  } else if (evt.type =="mouseout") {  evt.relatedTarget = evt.toElement;  }  evt.charCode = (evt.type=="keypress")?evt.keyCode:0;  evt.eventPhase = 2;//IE仅工作在冒泡阶段  evt.timeStamp = (new Date()).getTime();//仅将其设为当前时间  }  return evt;  }  fixEvent.preventDefault =function () {  this.returnValue = false;//这里的this指向了某个事件对象，而不是fixEvent  };  fixEvent.stopPropagation =function () {  this.cancelBubble = true;  };\*/  //console.log(EventUtil.$("button3"));//返回EventUtil函数的对象属性  //EventUtil.$("button3").onclick= eventfun;//2.使用为对象事件属性赋值的方法来实现事件的监听  //EventUtil.$("button3").onclick= eventfun2;//为事件属性添加多个方法时，为后者  //EventUtil.$("button3").onclick= eventfun;//事件捕获是从事件对象逐层外父级检察一直到window对象  var EventUtil =function(){  function getByid(id){  return document.getElementById(id);  };  // written by Dean Edwards, 2005  // with input from Tino Zijdel, Matthias Miller, Diego Perini    // <http://dean.edwards.name/weblog/2005/10/add-event/>    function addEvent(element, type, handler) {  if (element.addEventListener) {  element.addEventListener(type, handler, false);  } else {  // assign each event handler a unique ID  if (!handler.$$guid) handler.$$guid = addEvent.guid++;  // create a hash table of event types for the element  if (!element.events) element.events = {};  // create a hash table of event handlers for each element/event pair  var handlers = element.events[type];  if (!handlers) {  handlers = element.events[type] = {};  // store the existing event handler (if there is one)  if (element["on" + type]) {  handlers[0] = element["on" + type];  }  }  // store the event handler in the hash table  handlers[handler.$$guid] = handler;  // assign a global event handler to do all the work  element["on" + type] = handleEvent;  }  };  // a counter used to create unique IDs  addEvent.guid = 1;    function removeEvent(element, type, handler) {  if (element.removeEventListener) {  element.removeEventListener(type, handler, false);  } else {  // delete the event handler from the hash table  if (element.events && element.events[type]) {  delete element.events[type][handler.$$guid];  }  }  };    function handleEvent(event) {  var returnValue = true;  // grab the event object (IE uses a global event object)  event = event || fixEvent(((this.ownerDocument || this.document || this).parentWindow || window).event);  // get a reference to the hash table of event handlers  var handlers = this.events[event.type];  // execute each event handler  for (var i in handlers) {  this.$$handleEvent = handlers[i];  if (this.$$handleEvent(event) === false) {  returnValue = false;  }  }  return returnValue;  };    function fixEvent(event) {  // add W3C standard event methods  event.preventDefault = fixEvent.preventDefault;  event.stopPropagation = fixEvent.stopPropagation;  return event;  };  fixEvent.preventDefault = function() {  this.returnValue = false;  };  fixEvent.stopPropagation = function() {  this.cancelBubble = true;  };  function tableAddEvent(){    };    return{  add:addEvent,  remove:removeEvent,  $:getByid  }  }();    var outEleObj = EventUtil.$("outEle");  //addEvent.apply(EventUtil,[outEleObj,"click",eventfun3]);  //EventUtil.add(outEleObj,"click",eventfun3);  var inputObj = EventUtil.$("button4");  var tableEle = EventUtil.$("htmlEleTable");  var tabTrEle = tableEle.getElementsByTagName("tr");  EventUtil.add(tableEle,"click",eventfun3);  for (i=0; i<tabTrEle.length; i++){  EventUtil.add(tabTrEle[i],"click",eventfun3);  }  EventUtil.remove(tableEle,"click",eventfun3);//事件冒删除方法  EventUtil.add(tableEle,"click",eventfun3);//事件冒泡添加方法  //EventUtil.add(inputObj,"click",eventfun3);  //EventUtil.remove(outEleObj,"click",eventfun3);  //console.log(addEvent);  //addEvent(inputObj,"click",eventfun3,true);  //delEvent(outEleObj,"click",eventfun3,false);  </script>  </body>  </html> |

**PS：这里再为大家提供一个关于JS事件的在线工具，归纳总结了JS常用的事件类型与函数功能：**

**javascript事件与功能说明大全：**

<http://tools.jb51.net/table/javascript_event>