jQuery中的事件

# 事件绑定

事件绑定：bind(type,[data],fn)或者on(events,[selector],[data],handler)

删除绑定：unbind(type,[data],fn)或者off(events,[selector],[data],handler)

|  |
| --- |
| $("#panel h5.head").bind("click", function() {  var $content = $(this).next();  if ($content.is(":visible")) {  $content.hide();  } else {  $content.show();  }  })  #简写方式  $("#panel h5.head").mouseover(function() {  $(this).next().show();  }).mouseout(function() {  $(this).next().hide();  }) |

注意：绑定多个事件可以使用

$("selector").bind("事件1 事件2 ",function(){ ... });

## on绑定事件

|  |
| --- |
| //多事件绑定一  $("#test2").on('mousedown mouseup', function(e) {  $(this).text('触发事件：' + e.type)  })  $("#test3").on({  mousedown: function(e) {  $(this).text('触发事件：' + e.type)  },  mouseup: function(e) {  $(this).text('触发事件：' + e.type)  }  })  **委托机制**  $("div").on("click","p",fn)  事件绑定在最上层div元素上，当用户触发在a元素上，事件将往上冒泡，一直会冒泡在div元素上。如果提供了第二参数，那么事件在往上冒泡的过程中遇到了选择器匹配的元素，将会触发事件回调函数  //给body绑定一个click事件  //没有直接a元素绑定点击事件  //通过委托机制，点击a元素的时候，事件触发  $('body').on('click', 'a', function(e) {  alert(e.target.textContent)  }) |
| on和bind绑定事件的区别  on方法比.bind方法多一个参数'selector'，on的selector参数是筛选出调用.on方法的dom元素的指定子元素，如：$('ul').on('click', 'li', function(){console.log('click');})就是筛选出ul下的li给其绑定click事件；  好处在于.on方法为动态添加的元素也能绑上指定事件  总结：bind绑定的事件不能给其追加的子元素也绑定上事件,但是on通过选择器选项可以实现。 |

## off删除事件

|  |
| --- |
| //删除事件  $("button:last").click(function() {  $(".aaron:last").off()  }) |

## 常见的事件交互

|  |
| --- |
| #表单事件  **toggle(fn1,fn2,...)**：每次点击依次调用函数。  **blur([fn])**：失去焦点触发事件，并绑定相关函数  **focus([fn])** ：获取焦点触发事件，并绑定相关函数（focus()在元素本身产生，focusin()在元素包含的元素中产生）  **change([fn])**：触发change事件（文本被改变）, 绑定相关函数  input元素：监听value值的变化，当有改变时，失去焦点后触发change事件。对于单选按钮和复选框，当用户用鼠标做出选择时，该事件立即触发。  select元素：对于下拉选择框，当用户用鼠标作出选择时，该事件立即触发  textarea元素：多行文本输入框，当有改变时，失去焦点后触发change事件  **select([fn])**：触发每一个选择事件(input文本被选中), 并绑定相关函数  **click([fn])**：单击事件, 并绑定相关函数  **dbclick([fn])**：触发双击事件, 并绑定相关函数  **submit([fn])**：触发每一个提交事件(return false;//防止跳转)  #键盘事件  **keydown([fn]))**：用户在一个元素上第一次按下键盘上字母键时候，就会触发它。  **keyup([fn])**：当用户在一个元素上第一次松手键盘上的键的时候，就会触发它。  **keypress([fn])**：KeyPress主要用来接收字母、数字等ANSI字符，而 KeyDown 和 KeyUP 事件过程可以处理任何不被 KeyPress 识别的击键。  #鼠标事件  mousedown([fn])按下鼠标,  mousedown([fn])鼠标弹起  mouseover(fn)鼠标经过  mouseout(fn)鼠标移出 |

# 合成事件

## hover()方法

语法结构：hover(enter,leave)，当光标移动到元素上的时候，会触发指定的第一个函数enter，当光标移出这个元素的时候，会触发第二个函数leave

|  |
| --- |
| $("#panel h5.head").hover(function() {  $(this).next().show();  }, function() {  $(this).next().hide();  })  注意：css有伪类选择器，但是ie6支持超链接元素。hover方法是替代bind("mouseenter")，bind("mouseleave")， |

## toggle()方法

语法结构：toggle(fn1,fn2,...)：用于模拟鼠标连续单击事件。

|  |
| --- |
| $("#panel h5.head").toggle(function() {  $(this).next().show();  }, function() {  $(this).next().hide();  }) |
| #toggle方法另一个作用，切换元素的可见性  $("#panel h5.head").toggle(function() {  $(this).next().toggle();  }, function() {  $(this).next().toggle();  }) |

# 事件冒泡

冒泡事件：多个嵌套的元素中绑定了同一个事件。当事件触发的时候，事件会从最低层往上一个一个触发。

事件对象：由于IE-DOM,和标准DOM实现事件的方式各不相同。jQuery封装了事件对象来解决兼容性问题。另外，事件对象只有在事件处理函数才能访问到。事件处理函数执行完毕后，事件对象就消失了。

|  |
| --- |
| 注意：事件对象是用来记录一些事件发生时的相关信息的对象。事件对象只有事件发生时才会产生，并且只能是事件处理函数内部访问，在所有事件处理函数运行结束后，事件对象就被销毁 |

$("element").bind("click",function(event){...}

停止事件冒泡：event.stopPropagation();事件是可以冒泡的，为防止事件冒泡到DOM树上，也就是不触发的任何前辈元素上的事件处理函数

阻止默认行为：event.preventDefault()；//阻止单击超链接，提交默认行为

#同时停止冒泡和默认行为可以使用return false;实现

# 事件对象属性

event.type：获取事件类型，例如单击click等

event.preventDefault()：阻止默认行为

event.stopPropagation()：停止冒泡

event.target：代表当前触发事件的元素，可以通过当前元素对象的一系列属性来判断是不是我们想要的元素

event.pageX和event.pageY：用于获取光标相对于x坐标和y坐标

event.which：用于获取鼠标点击事件中鼠标左中右键（分别对应1,2,3）。

event.currentTarget : 在事件冒泡过程中的当前DOM元素，冒泡前的当前触发事件的DOM对象, 等同于this.

**this和event.target的区别**：js中事件是会冒泡的，所以this是可以变化的，但event.target不会变化，它永远是直接接受事件的目标DOM元素；.this和event.target都是dom对象

如果要使用jquey中的方法可以将他们转换为jquery对象。比如this和$(this)的使用、event.target和$(event.target)的使用；

# 移除事件

|  |
| --- |
| #同一个元素可以绑定多个事件，多个元素绑定同一个事件  $('#btn').bind("click", function(){  $('#test').append("<p>我的绑定函数1</p>");  }).bind("click", function(){  $('#test').append("<p>我的绑定函数2</p>");  }).bind("click", function(){  $('#test').append("<p>我的绑定函数3</p>");  });  #删除所有绑定事件  $('#delAll').click(function(){  $('#btn').unbind("click");  });  #删除指定事件  $('#btn').bind("click", myFun1 = function() {  $('#test').append("<p>我的绑定函数1</p>");  }).bind("click", myFun2 = function() {  $('#test').append("<p>我的绑定函数2</p>");  }).bind("click", myFun3 = function() {  $('#test').append("<p>我的绑定函数3</p>");  });  $('#delTwo').click(function() {  $('#btn').unbind("click", myFun2);  });  #绑定一次事件  $('#btn').one("click", function() {  $('#test').append("<p>我的绑定函数1</p>");  }).one("click", function() {  $('#test').append("<p>我的绑定函数2</p>");  }).one("click", function() {  $('#test').append("<p>我的绑定函数3</p>");  }); |

# 模拟操作

|  |
| --- |
| #模拟用户操作：$('#btn').trigger("click");或者$('#btn').click();  #模拟自定义事件  ('#btn').bind("myClick", function(event, message1, message2) {  $('#test').append("<p>" + message1 + message2 + "</p>");  });  $('#btn').click(function() {  $(this).trigger("myClick", ["我的自定义", "事件"]);  }).trigger("myClick", ["我的自定义", "事件"]);  #执行默认行为  $('#old').bind("click", function() {  $("input").trigger("focus");  });  $('#new').bind("click", function() {  $("input").triggerHandler("focus");  });  $("input").focus(function() {  $("body").append("<p>focus.</p>");  })  注意：trigge会触发focus事件并获取焦点。而triggerHandler不会获取焦点。若要触发通过 jQuery 绑定的事件处理函数，不触发原生事件，使用triggerHandler() 来代替triggerHandler与trigger的用法是一样的，重点看不同之处：   * triggerHandler不会触发浏览器的默认行为，.triggerHandler( "submit" )将不会调用表单上的.submit() * .trigger() 会影响所有与 jQuery 对象相匹配的元素，而 .triggerHandler() 仅影响第一个匹配到的元素 * 使用 .triggerHandler() 触发的事件，并不会在 DOM 树中向上冒泡。 如果它们不是由目标元素直接触发的，那么它就不会进行任何处理 * 与普通的方法返回 jQuery 对象(这样就能够使用链式用法)相反，.triggerHandler() 返回最后一个处理的事件的返回值。如果没有触发任何事件，会返回 undefined |