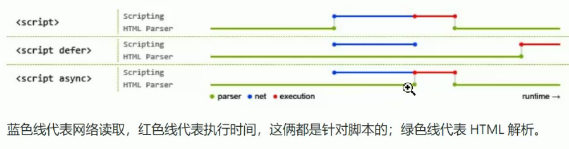
**事件**

如果需要加载完页面后再加载js，可以在script中添加async或者defer

如果script标签没有defer或者async属性

浏览器遇到script标签会立即停止渲染页面 开始加载js文件



**defer**

加载所有的js文件后，顺序执行js文件

**async**

加载完整个DOM树后，哪个体积小 哪个先执行

（优先用defer）

**事件的分类**

**1.鼠标事件**

onclick 单击事件

ondbclick 双击事件

onmousedown 按下事件

onmouseup 鼠标松开

onmousewheel 鼠标的滚轮事件

onmouseenter 鼠标进入(遇到子元素'不会'触发)

onmouseleave 鼠标离开

onmosueover 鼠标进入(遇到子元素'会'触发)

onmosueout 鼠标离开

onmousemove 鼠标移动

**2.键盘事件**

**onkeypress 某个键被按下并松开（对非功能键有效）**

**onkeydown 按键被按下**

对于onkeydown来说如果已知按着某个按键不放的话 事件会一直触发

当onkeydown连续触发时，第一次和第二次之间会间隔稍微长一点，其他的会非常的快

这种设计是为了防止误操作的发生。

**onkeyup 按键被松开**

对于onkeyup来说按键松开事件只会触发一次

键盘事件一般都会绑定给一些可以获取到焦点的对象或者是document

**keyCode**

通过keyCode可以获取被按下的按键的编码，由此来判断哪个按键被按下

配合onkeydown使用 keyCode返回的字符键码只有大写字母 65-90 也能识别到功能键的键码

配合onkeypress使用 返回的键码则区分大小写 不识别功能键

**charCode**

如果在onkeydown中使用则只会返回0

配合onkeypress使用 返回的键码区分大小写 只识别字符的键码

event事件对象中还提供了几个属性

**altKey** alt键是否被按下

**ctrlKey** ctrl键是否被按下

**shiftKey** shift键是否被按下

**metaKey** windows键是否被按下

如果被按下则返回true 没被按下返回false

**3.html事件**

onload 加载事件 一张页面或一幅图像完成加载

window.onload 页面加载完成且外部资源也加载完成才会触发

凡是带有src属性的标签都具备onload事件

img、link、script

onunload 卸载事件 用户退出页面 ( <body> 和 <frameset>) 会发生回流

DOMContentLoad事件：DOM结构加载完成就触发

onscroll 当用户滚动带滚动条的元素时触发

onresize 窗口或框架被重新调整大小

用媒体查询实现响应式布局的缺点是代码量大，css文件体积大，加载速度慢

可以用js来实现响应式布局，速度比较快，但是改变页面的时候会发生重绘和回流，对性能有影响

oncontextmenu 右击菜单事件

onwheel 滚轮事件

**4.表单事件**

onselect 当用户选择文本框(input 或 textarea)中的内容触发。

onchange 当文本框(input 或 textarea)内容改变并且失去焦点后触发。

onfocus 当页面或者元素获得焦点时触发。

onblur 当页面或元素失去焦点时触发。

onsubmit 当用户点击提交按钮在<form>元素节点上触发。

onreset 当用户点击重置按钮在<form>元素节点上触发。

**节流(throttle) 和 防抖(debounce)**

去第三方找一个库，lodash,引入js文件

引入了loadsh.js，就可以使用\_这个标识符

**语法：**

\_.throttle(fn,delay)

**节流和防抖的共同点**

都是减少执行的次数

**不同点:**

节流: 同一时间内,稀释执行的次数

防抖: 最后一次,才生效,如果中间就被触发,那就一直延迟执行的时间

oBox.onmousemove = \_.throttle( function( ) {

console.log( "onmousemove ");

},600);

**事件对象**

触发事件的时候，产生的一条记录对象

记录：点击位置，事件类型，目标元素等等属性

当事件的响应函数被触发时，浏览器每次都会将一个事件对象作为实参传递进响应函数

我们只要定义一个形参就可以在响应函数中使用这个事件对象

在事件对象中封装了当前事件的一切信息，

比如：鼠标的坐标、键盘的哪个按键被按下、鼠标滚轮滚动的方向

注意：在ie8及以下的浏览器中，事件对象是作为window对象的属性保存的

**兼容写法：写在单击响应函数里**

if(!event){

event = window.event;

}

或者可以这么写 更常用

event = event || window.event;

**事件对象的相关属性**

**clientX、clientY**

**获取鼠标指针在当前可见窗口的水平坐标和垂直坐标**

**语法：**

event.clientX//兼容火狐

window.event.clientX//兼容IE8

**pageX、pageY**

**获取鼠标指针在当前页面的水平坐标和垂直坐标**

**语法：**

event.clientX//兼容火狐

window.event.clientX//兼容IE8

注意：这两个属性不支持IE8及以下浏览器，要兼容的话还是得用clientX和clientY

**offsetX、offsetY**

获取鼠标指针到当前元素（左上角）的水平坐标和垂直坐标

**语法：**

event.offsetX//兼容火狐

window.event.offsetX//兼容IE8

**screenX、screenY**

获取鼠标指针到屏幕（左上角）的水平坐标和垂直坐标

**语法：**

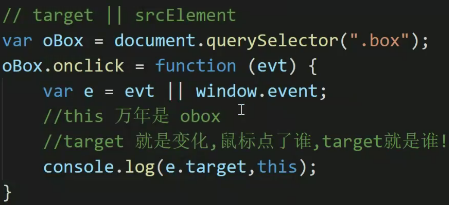
event.screenX//兼容火狐

window.event.screenX//兼容IE8

**target | | srcElement**

event中的target表示的触发事件的对象

由谁触发的 target就是谁 不等价于this



**button**

按下哪个键 0是左键 1是滚轮 2是右键 只能在onmousedown里看到 onclick看不到

**事件的冒泡（Bubble）**

所谓的事件冒泡就是指事件的向上传导，当后代元素的事件被触发时，其所有祖先元素的相同事件也会被触发

在开发中大部分情况都是有用的，如果不希望发生事件冒泡可以通过事件对象来阻止冒泡

**语法：**

event.cancelBubble = true;

cancelBubble不支持火狐 其他基本支持

cancelBubble属性用于控制事件流的冒泡阶段。如果该属性设置为 true，则事件将在当前级别停止冒泡。如果该属性设置为 false，则该事件将向上冒泡到其父级。此属性的默认值由事件类型决定。

event里还有一个stopPropagation()方法 可以阻止捕获和冒泡阶段中当前事件的进一步传播。

但是，它不能防止任何默认行为的发生； 例如，对链接的点击仍会被处理。

兼容写法：

if (e.stopPropagation){

e.stopPropagation();

}else {

e.cancelBubble = true;

}

**事件的委托**

指将事件统一绑定给元素的共同的祖先元素，这样当后代元素上的事件触发时，会一直冒泡到祖先元素，从而通过祖先元素的响应函数来处理事件。结合target去实现，没有target无法实现。

事件委派是利用了冒泡，通过委托可以减少事件绑定的次数，提高程序的性能

如果子元素都是动态创建的 推荐使用事件委托

好处：可以减少事件绑定的次数，减少事件绑定带来的开销问题，提高程序的性能

缺点：找目标元素的代码比较繁琐

**DOM级别与DOM事件**

DOM级别一共可以分为四个级别：DOM0级、DOM1级、DOM2级和DOM3级。

而DOM事件分为3个级别：DOM0级事件处理，DOM2级事件处理和DOM3级事件处理。

因为1级DOM标准并没有定义事件相关的内容，所以没有所谓的1级DOM事件模型。

**1.DOM0级处理事件**

将一个函数赋值给一个事件处理属性

<button id="btn" type="button"></button>

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

btn.onclick = function() {

console.log('Hello World');

}

将一个函数赋值给了一个事件处理属性onclick 这样的方法就是DOM0级处理事件的体现。我们可以通过给事件处理属性赋值null来解绑事件。DOM0级事件处理程序的缺点在于一个处理程序无法同时绑定多个处理函数，比如我还想再点击按钮事件上加上另外一个函数。

**2.DOM2级事件**

DOM2级事件在DOM0级事件的基础上弥补了一个处理程序
无法同时绑定多个处理函数的缺点。允许给一个程序添加多个处理函数。

<button id="btn" type="button"></button>

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

function showFn() {

alert('Hello World');

}

btn.addEventListener('click', showFn, false);

// btn.removeEventListener('click', showFn, false); 解绑事件

DOM2级事件定义了addEventListener 和 removeEventListener两个方法，分别用来绑定和解绑事件，方法中包含三个参数，分别是绑定的事件处理的属性名称（没有on）处理函数和是否在捕获时候执行事件处理函数如果我们还需要添加一个鼠标的移入的方法，只需要：

btn.addEventlistener('mouseover',showfn,false)

IE8以下版本不支持 addEventlistener 和 removeEventListerner

需要使用attachEvent和detachEvent实现：

这里我们不需要传入第三个参数，因为IE8以下版本只支持冒泡型事件。

btn.attachEvent('onclick', showFn); // 绑定事件

btn.detachEvent('onclick', showFn); // 解绑事件

**3.DOM3级事件**

DOM3级事件是在DOM2级事件的基础上添加很多事件类型。

|  |  |
| --- | --- |
| UI事件 | 当用户与页面上的元素交互时触发，如：load、scroll |
| 焦点事件 | 当元素获得或失去焦点时触发，如：blur、focus |
| 鼠标事件 | 当用户通过鼠标在页面执行操作时触发如：dblclick、mouseup |
| 滚轮事件 | 当使用鼠标滚轮或类似设备时触发，如：mousewheel |
| 文本事件 | 当在文档中输入文本时触发，如：textInput |
| 键盘事件 | 当用户通过键盘在页面上执行操作时触发，如：keydown、keypress |
| 合成事件 | 当为IME（输入法编辑器）输入字符时触发，如：compositionstart |
| 变动事件 | 当底层DOM结构发生变化时触发，如：DOMsubtreeModified |

同时DOM3级事件也允许使用者自定义一些事件。

**事件的绑定**

**1.对象.事件=函数**

使用对象.事件=函数的形式绑定响应函数,

它只能同时为一个元素的一个事件绑定一个响应函数,

不能绑定多个,如果绑定了多个,则后边会覆盖掉前边的

**2.addEventListener()**

通过这个方法也可以为元素绑定响应函数

参数：

第一个参数是事件的字符串 去掉on

第二个参数是事件执行函数名称(没有双引号, 也没有( )，事件触发时函数会被调用

第三个参数是一个对象

如果只传布尔值，则表示是否在捕获阶段触发事件，一般都传false（除了特殊情况）

如果是{once:true} 则表示只响应一次表示捕获阶段触发(逗号隔开)

使用这种方法可以同时为一个元素的相同事件绑定多个响应函数

当事件被触发时，响应函数会按顺序被执行 先绑定先执行

如果要拆的话说明执行顺序不重要 重要的写在一起

注意：这种方法不支持IE8及以下的浏览器

addEventListener()中的this是绑定事件的对象 r

使用addEventListener()方法绑定响应函数，取消默认行为时不能使用return false，需要使用event来取消默认行为event.preventDefault();但是IE8不支持event.preventDefault();这个玩意，如果直接调用会报错

**语法：**

btn.addEventListener("click",function(){

alert(3);

},false);

**3.attachEvent()**

在IE8中可以使用attachEvent()来绑定事件

参数：

第一个参数是事件的字符串，要on

第二个参数是回调函数

这个方法也可以同时为一个事件绑定多个处理函数，

不同的是它是后绑定先执行，执行顺序和addEventListener()相反

attachEvent()中的this是window

注意：这种方法不支持IE8以上及其他的浏览器

**语法：**

btn.attachEvent("onclick",function(){

alert(this);

});

**4.定义一个函数用来为指定元素绑定多个响应函数**

**参数：**

obj 要给那个对象绑定时间

eventStr 事件的字符串 、具体是什么事件

callback 回调函数，就是事件响应函数

**eg.**

**function bind**(obj, eventStr, callback){

**//看看obj中是**否含有addEventListener这个方法 有的话会返回一个对象

**if(obj.ad**dEventListener){

**//大部分浏览器**的兼容方式

**obj.addE**ventListener(eventStr,callback,false)

**}else{**

**//IE**8及以下的兼容方式

**obj.atta**chEvent("on"+eventStr , function(){

**//在匿名函数中调用**回调函数

**callback.c**all(obj);

**});**

**}**

**}**

**注意的点：**

1.当obj调用attachEvent时 obj.attachEvent("on"+eventStr , callback);

回调函数callback中的this指向的是window、

而obj.addEventListener(eventStr,callback,false)中 回调函数callback中的this指向的是调用函数的对象obj

2.所以 我们可以在obj.attachEvent("on"+eventStr , )的第二个参数中传入一个匿名函数，当浏览器调用匿名函数，匿名函数中调用我们传入的callback回调函数

3.并且通过.call(obj)的方法调用这个回调函数，将回调函数中的this强行改为obj

知识点：

/\*

\* this是谁由调用方式决定

\* callback.call(obj)

\*/



**事件的传播**

关于事件的传播网景公司和微软公司有不同的理解：

微软公司认为事件应该是由内向外传播，也就是当事件触发时，应该先触发当前元素上的事件，
然后再向当前元素的祖先元素上传播，也就说事件应该在冒泡阶段执行。

网景公司认为事件应该是由外向内传播的，也就是当前事件触发时，应该先触发当前元素的最外层的祖先元素的事件，
然后在向内传播给后代元素、

W3C综合了两个公司的方案，将**事件传播分成了三个阶段**

**1.捕获阶段**

- 在捕获阶段时从最外层的祖先元素，向目标元素进行事件的捕获，但是默认此时不会触发事件

**2.目标阶段**

- 事件捕获到目标元素，捕获结束开始在目标元素上触发事件

**3.冒泡阶段**

- 事件从目标元素向他的祖先元素传递，依次触发祖先元素上的事件

当我们拖拽一个网页中的内容时，浏览器会默认去搜索引擎中搜索内容，此时会导致拖拽功能的异常，这个是浏览器提供的默认行为，

如果不希望发生这个行为，则可以通过return false来取消默认行为

超链接的跳转行为同理，也可以这么取消，但是这招对IE8不起作用

**setCapture() 和releaseCapture()**

**1.功能：**

在当前线程的指定窗口里设置鼠标捕获。一旦窗口捕获了鼠标，所有鼠标输入都针对该窗口，无论光标是否在窗口的边界内还是边界外。同一时刻只能有一个窗口捕获鼠标。

**2. 失效条件:**

A. 当鼠标在其他窗口按下

B.调用ReleaseCapture释放

**3. SetCapture和ReleaseCapture必须成对出现**

只有IE支持，但是在火狐中调用时不会报错，

而如果使用chrome调用，会报错

//设置btn01对鼠标按下相关的事件进行捕获

//当调用一个元素的setCapture()方法以后，这个元素将会把下一次所有的

//鼠标按下相关的事件捕获到自身上

//捕获的两种写法

if(box1.setCapture){

box1.setCapture();

}

box1.setCapture && box1.setCapture();

//释放的写法

box1.releaseCapture && box1.releaseCapture();

**滚轮事件**

onmousewheel鼠标滚轮滚动的事件，会在滚轮滚动时触发，
但是火狐不支持该属性

**1.**

除了火狐以外的浏览器可以用obj.onmousewheel = fun;来绑定滚轮事件对应的函数

但是在火狐中则需要使用DOMMouseScroll绑定滚动，这个事件需要addEventListener来绑定，我们可以写一个bind函数来绑定

box1.onmousewheel = funciton(){}

bind(box1,"DOMMouseScroll",box1.onmousewheel);

**2.**

在判定滚轮是往上滚还是往下滚的时候，火狐需要用event.detail来获取滚轮的上滚和下滚

而其他浏览器包括ie只需要event.wheelDeltal来访问

区别就是火狐中往下滚动event.detail是3（大于0）的，其他浏览器往下滚event.wheelDelta是（小于0）的

火狐中往上滚动event.detail是-3的（小于0），其他浏览器往上滚event.wheelDelta是120（大于0）的

写法：

if(event.wheelDelta < 0 ||event.detail > 0){

box1.style.height = box1.clientHeight + 10 +"px";

}else{

box1.style.height = box1.clientHeight - 10 +"px";

}

**浏览器的默认行为**

当浏览器有滚动条的时候 不管在哪个元素上滚动滚轮，浏览器的滚动条都会发生滚动，如果不希望鼠标在box1上滚动的时候浏览器的滚动条滚动，则在box1的滚动响应函数末尾添加return false即可取消默认行为

当点击超链接的时候浏览器会刷新页面或者跳转，这是a标签的默认行为，如果不希望响应事函数执行的同时默认行为也发生的话则在a标签的响应函数最后添加return false即可

在input框中输入内容属于onkeydown的默认行为，如果在onkeydown中取消了默认行为，则输入的内容不会出现在input框中

input框的focus事件触发时 光标闪烁是默认行为

在form表单中输入内容，提交时表单会向地址栏添加参数，这也是表单的默认行为，如果用return false取消了默认行为，则输入的内容不会被添加到地址栏

鼠标点击img并移动的时候img图片默认会被拖拽，我们可以通过给img标签添加属性*ondragstart*="return false"，来取消默认被拖拽的效果

但是使用addEventListener函数给事件绑定响应函数的时候，取消默认行为不能使用return false

需要使用event来取消默认行为，event.preventDefault()

但是又有新的点要注意，IE678不支持event.preventDefault(),直接调用会报错

event.preventDefault && event.preventDefault();

event.returnValue = false;

return false;