

Event事件对象

0.1. 本节主要内容：



0.1. 学习目标：

节数	知识点	要求
第一节 event对象	关于event对象	了解
	event对象的使用	了解
第二节 event中常用的属性和方法	event中常用属性	了解
	event中常用的方法	了解
第三节 ie中常用的属性和方法	ie浏览器常用的属性和方法	掌握
第四节 鼠标滚轮事件	鼠标滚轮事件	掌握
第五节 文档事件	文档事件	掌握
第六节 事件委托	事件委托	掌握

1. event对象

1.1. 关于event对象

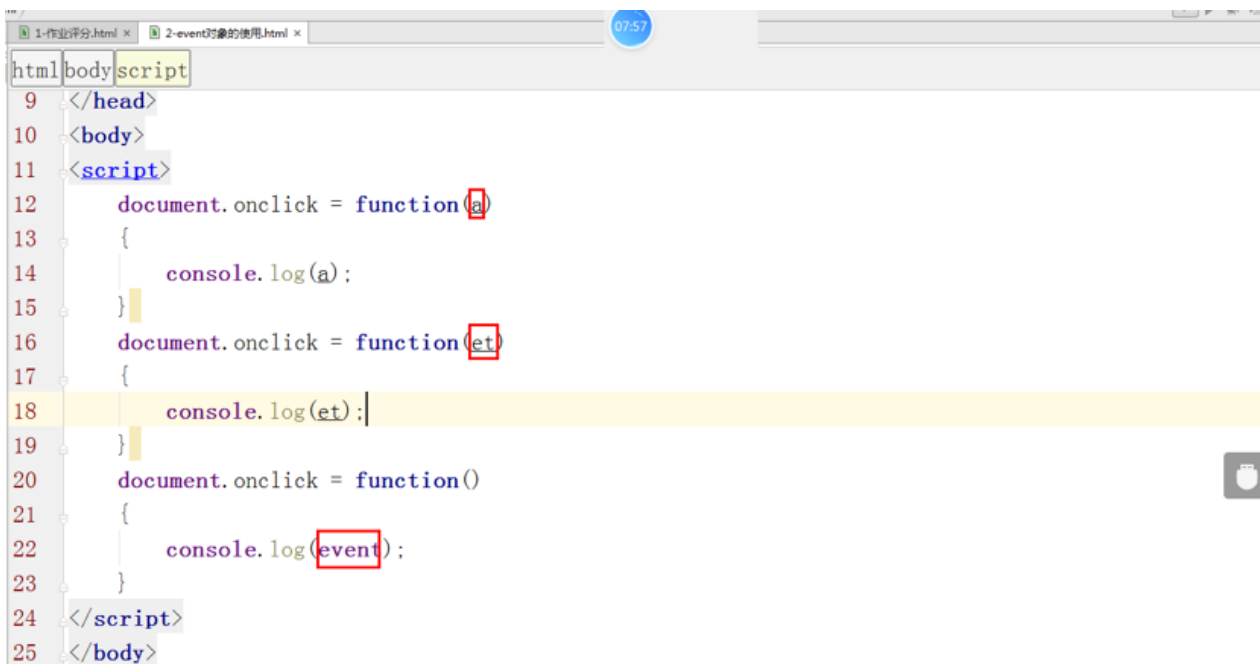
概念： Event 对象代表事件的状态，当dom tree中某个事件被触发的时候，会同时自动产生一个用来描述事件所有的相关信息（比如事件在其中发生的元素、键盘按键的状态、鼠标的位置、鼠标按钮的状态。）的对象，这个对象就是event（事件对象）。

1.2. Event事件对象的使用

直接通过event来获取

通过函数传参数的形式 还可以通过函数传参数的形式来使用，一般而言我们使用【形参e或event】来代替。

```
1.document.querySelector("#d1").onclick = function(e){
console.log(e);
}; 2.document.querySelector("#d1").onmousemove =
function(eve){
console.log(eve);
}
3.document.querySelector("#d1").onkeyup = function(){
console.log(event);
};
```



```
9 </head>
10 <body>
11 <script>
12     document.onclick = function(a)
13     {
14         console.log(a);
15     }
16     document.onclick = function(et)
17     {
18         console.log(et);
19     }
20     document.onclick = function()
21     {
22         console.log(event);
23     }
24 </script>
25 </body>
```

因为event对象是用来描述【发生的事件的信息】的，而event对象当中所提供的一系列属性和方法正是用来获取这些信息的途径。

2. event中常用的属性和方法

2.1. event中常用的属性

因为event对象是用来描述【发生的事件的信息】的，而event对象当中所提供的一系列属性和方法正是用来获取这些信息的途径。

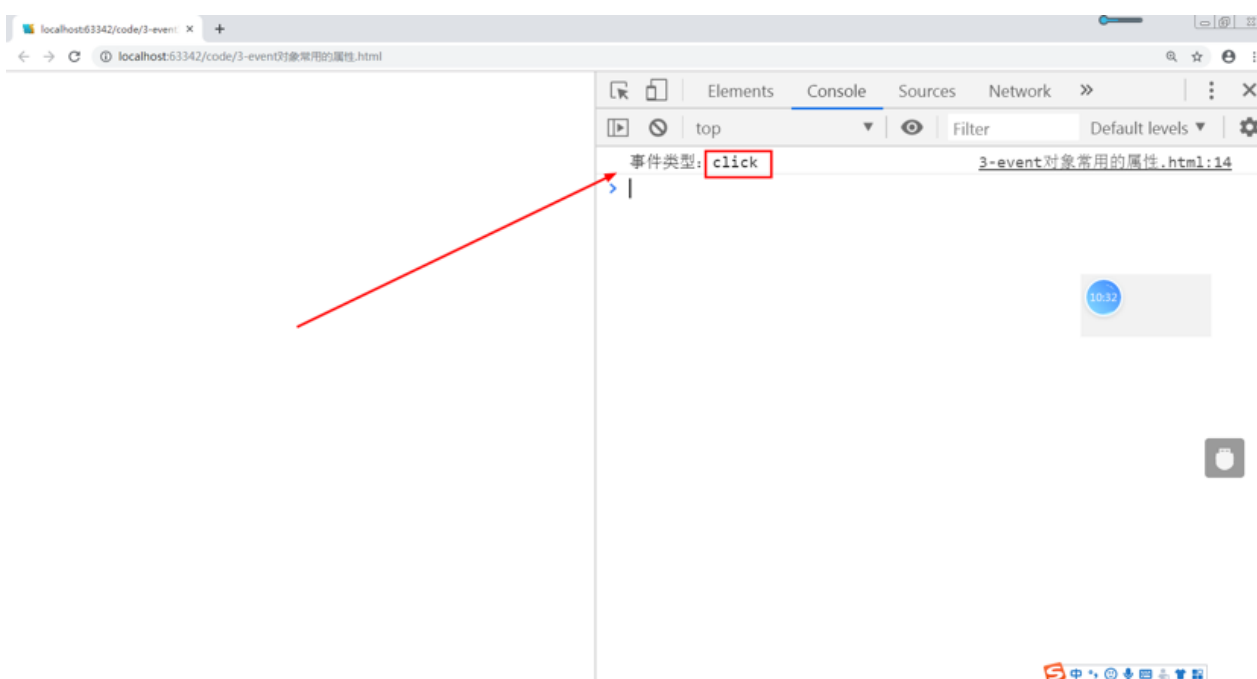
2.2. type 属性

type属性用来获得【当前触发事件】的类型，此属性只读。

```
document.getElementById("d1").onclick = function () {
    console.log(event); console.log(event.type); //依赖于事件的触发
    而存在，只读属性 };

```

```
1-作业评分.html × 2-event对象的使用.html × 3-event对象常用的属性.html ×
html body
6 <style>
7
8 </style>
9 </head>
10 <body>
11 <script>
12     document.onclick = function(a)
13     {
14         console.log("事件类型: "+a.type); //返回的是事件类型
15     }
16
17 </script>
18 </body>
19 </html>
```



2.3. bubbles属性

bubbles属性用来获得【当前触发事件的类型】是否冒泡，如果当前事件类型支持冒泡则返回true，否则返回false。

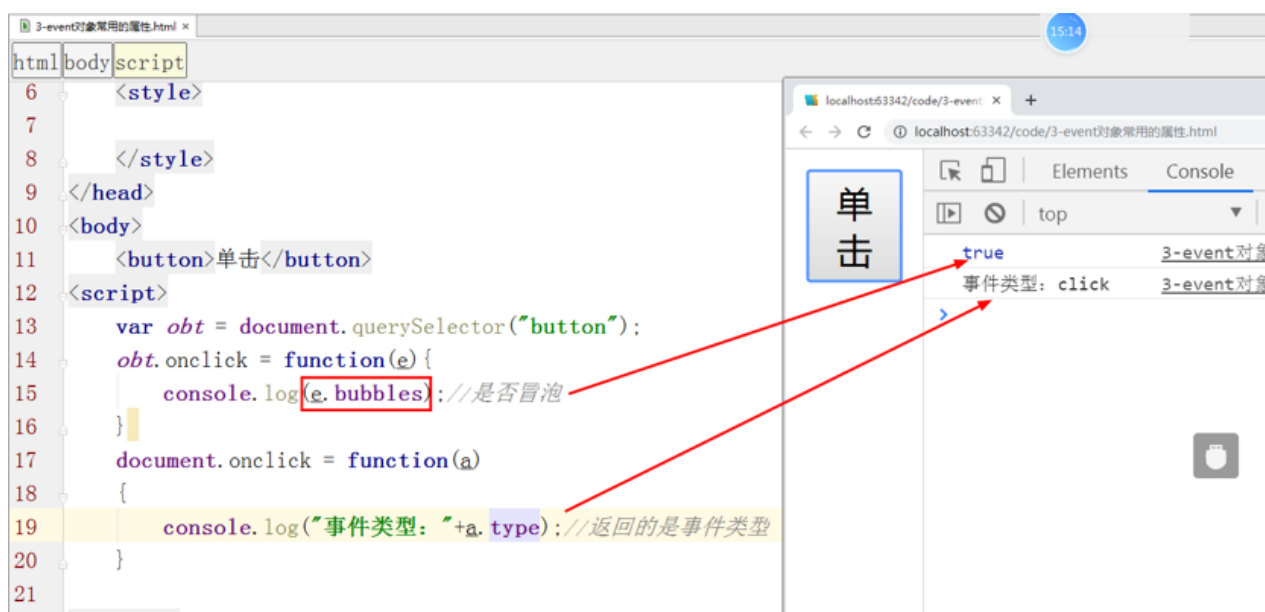
必须注意的是：bubbles属性指的是该事件是否冒泡。和事件处理机制无关！！！！

```
document.getElementById("d1").onclick = function () {
    console.log(event.bubbles);
};
```

```
document.getElementById("d1").addEventListener('mouseenter',function (e) { console.log(e.bubbles); });
```

因为鼠标【点击事件】这个事件本身支持冒泡。

因此当存在点击事件被触发后，**event**对象的**bubbles**属性返回的就是**true**，表示当前事件支持冒泡。



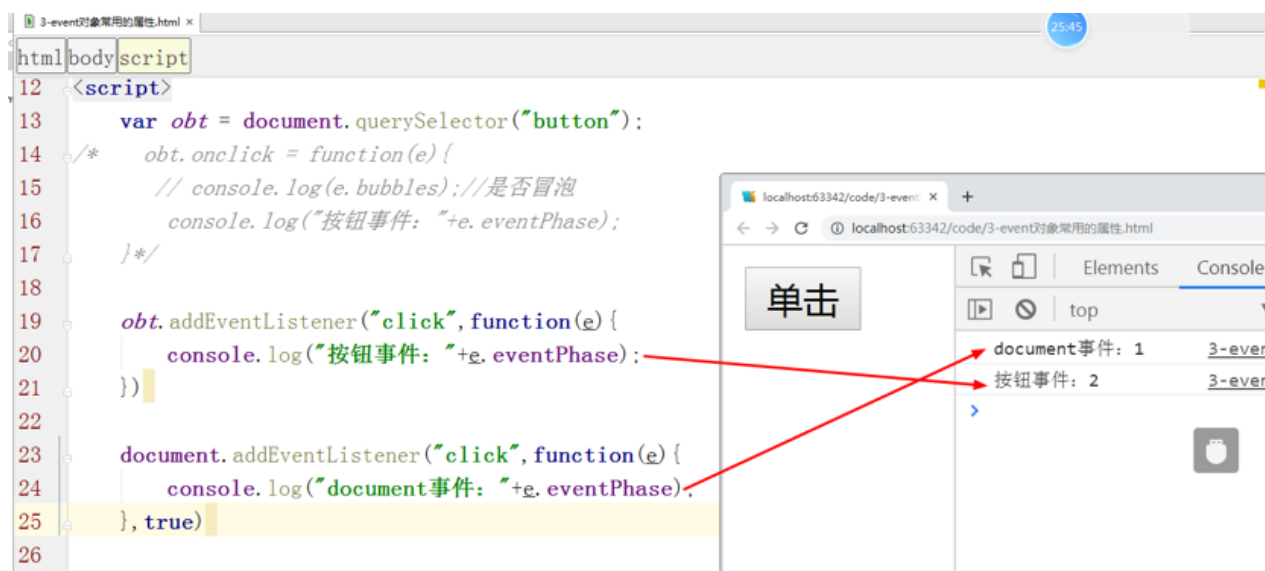
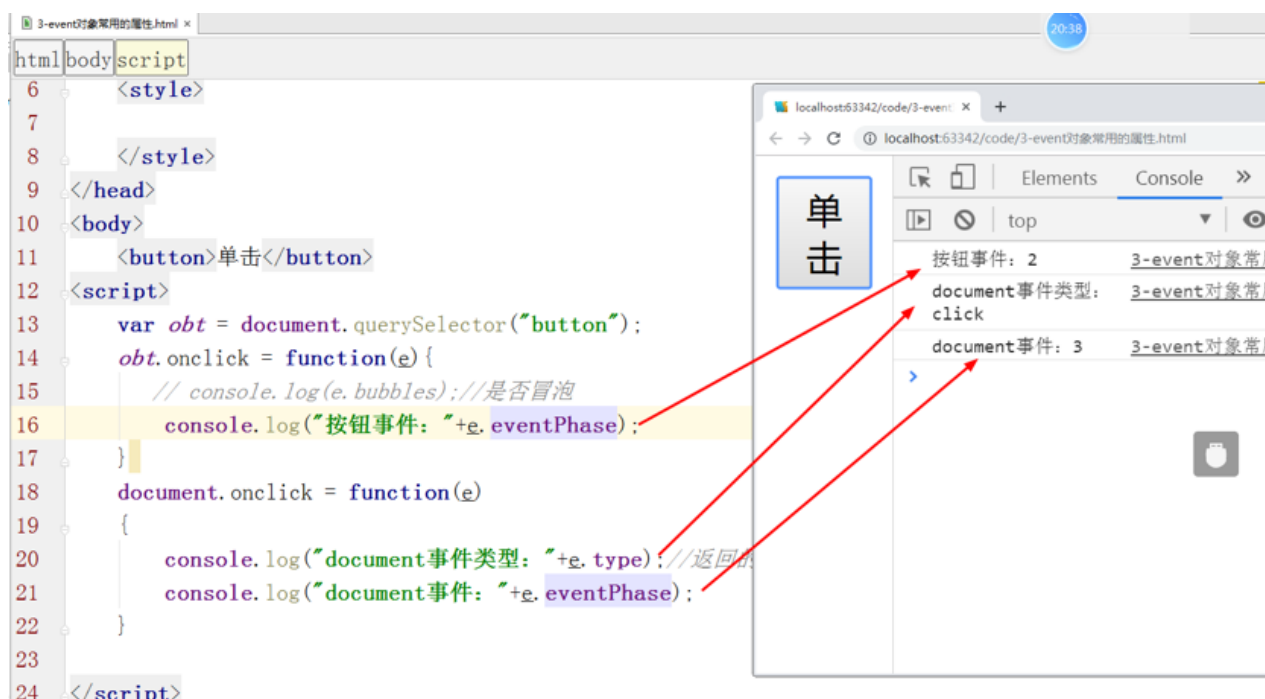
2.4. eventPhase 属性

eventPhase：事件传导至【当前节点】时处于什么的状态。

- 1：事件处于捕获状态
- 2：事件处于真正的触发者
- 3：事件处于冒泡状态

```
d1.onclick = function (e) {console.log(this, e.eventPhase);};  
d2.onclick = function (e) {console.log(this, e.eventPhase);};  
document.onclick = function (e) {console.log(this,  
e.eventPhase);};
```

```
d1.addEventListener('click', function (e) {console.log(this,
e.eventPhase)}, true); d2.addEventListener('click', function (e)
{console.log(this, e.eventPhase)}, true);
document.addEventListener('click', function (e)
{console.log(this, e.eventPhase)}, true);
```



2.5. target 属性和 currentTarget 属性

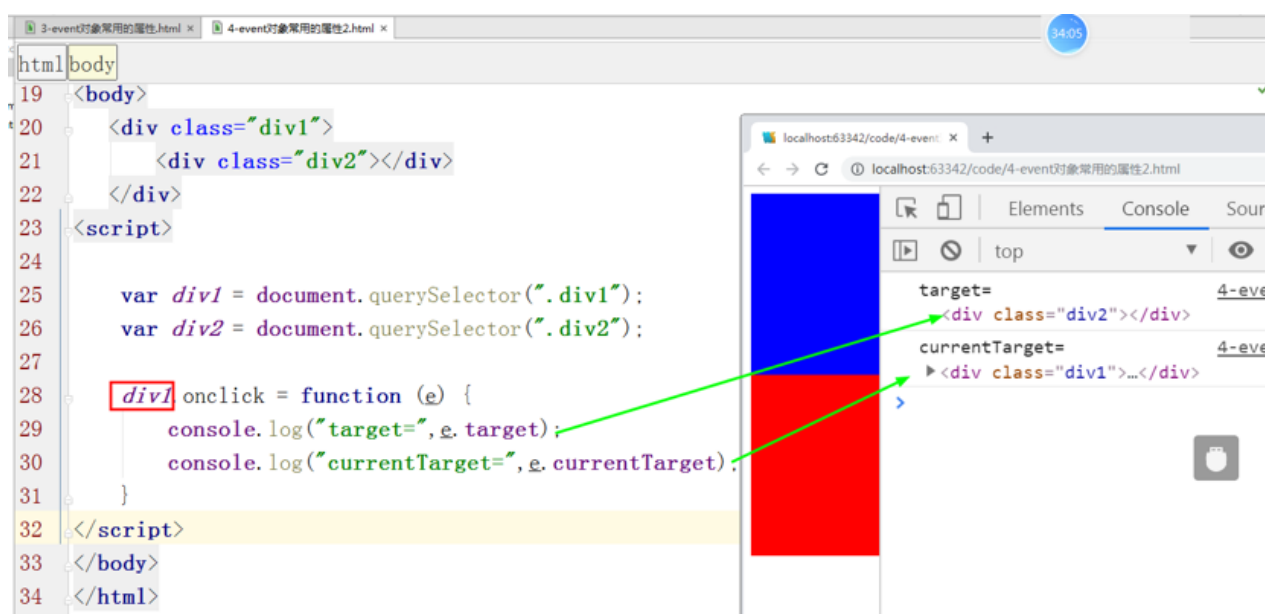
target: 返回事件真正的触发者

currentTarget: 返回事件的监听者(触发的事件绑定到了哪个节点, 就返回谁)

d1

d2

```
document.getElementById("d1").onclick = function (e) {  
  console.log(e.target); console.log(e.currentTarget); };
```



2.6. button属性

button 返回当事件被触发时, 哪个鼠标按钮被点击。

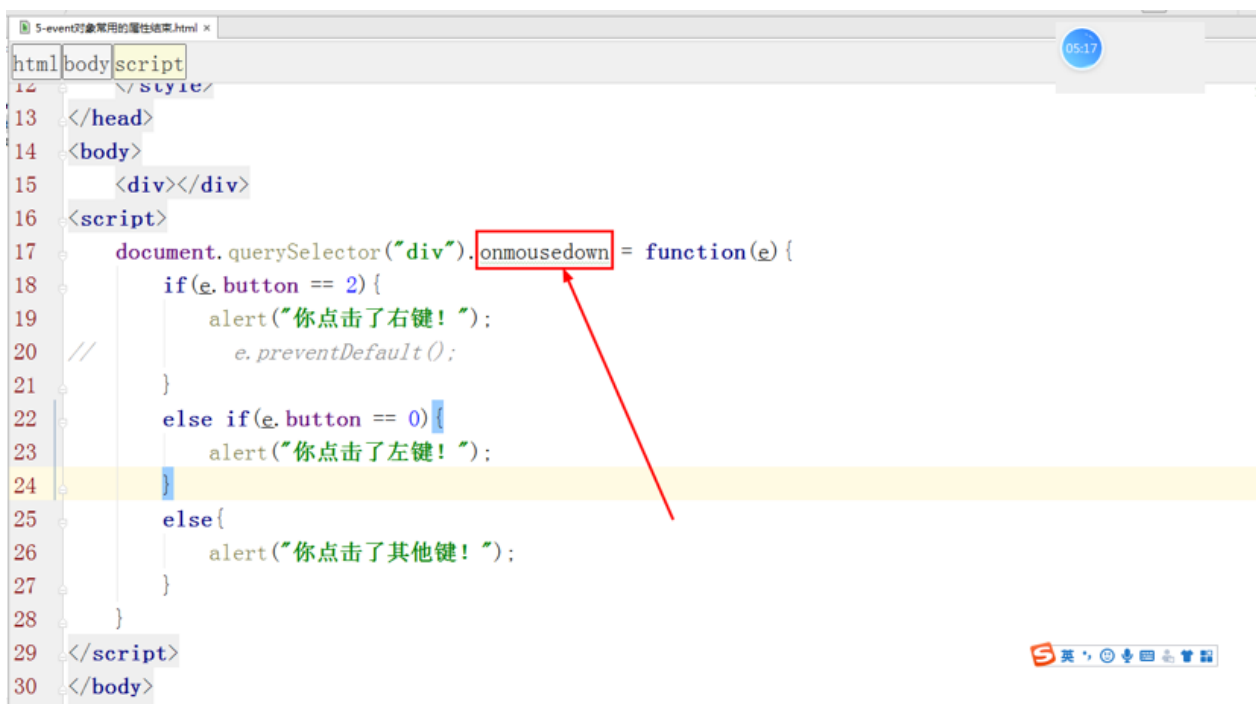
event.button=0 | 1 | 2

参数 描述

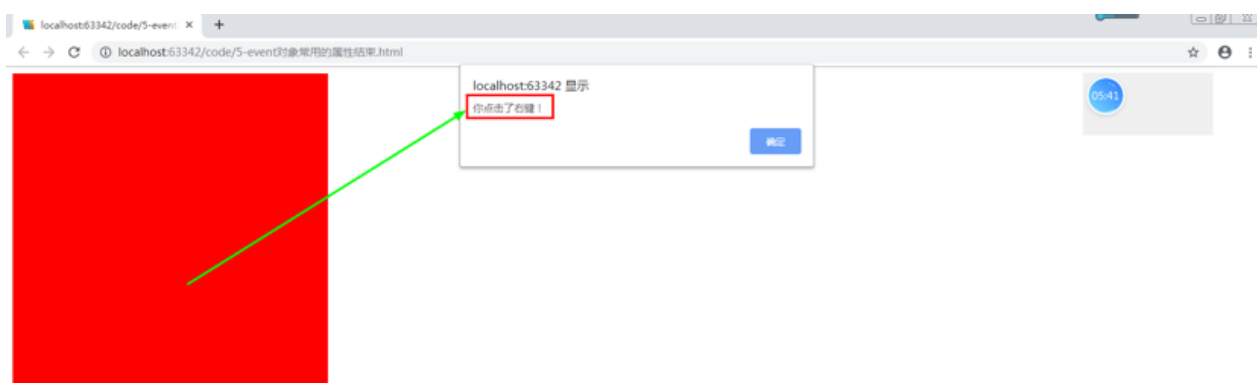
0 指定鼠标左键。

1 指定鼠标中键。

2 指定鼠标右键。



```
12 </style>
13 </head>
14 <body>
15   <div></div>
16 <script>
17   document.querySelector("div").onmousedown = function(e) {
18     if(e.button == 2) {
19       alert("你点击了右键!");
20       e.preventDefault();
21     }
22     else if(e.button == 0) {
23       alert("你点击了左键!");
24     }
25     else {
26       alert("你点击了其他键!");
27     }
28   }
29 </script>
30 </body>
```



注意： Internet Explorer 8 及更早IE版本有不同的参数：

参数 描述

1 指定鼠标左键。(IE8及更早IE版本)

4 指定鼠标中键。(IE8及更早IE版本)

2 指定鼠标右键。

2.7. key和keyCode属性

key是哪个键

keyCode返回keydown何keyup事件发生的时候按键的代码，以及keypress 事件的**Unicode**字符(**ASCII码值**)；(firefox2不支持event.keyCode，可以用 event.which替代)

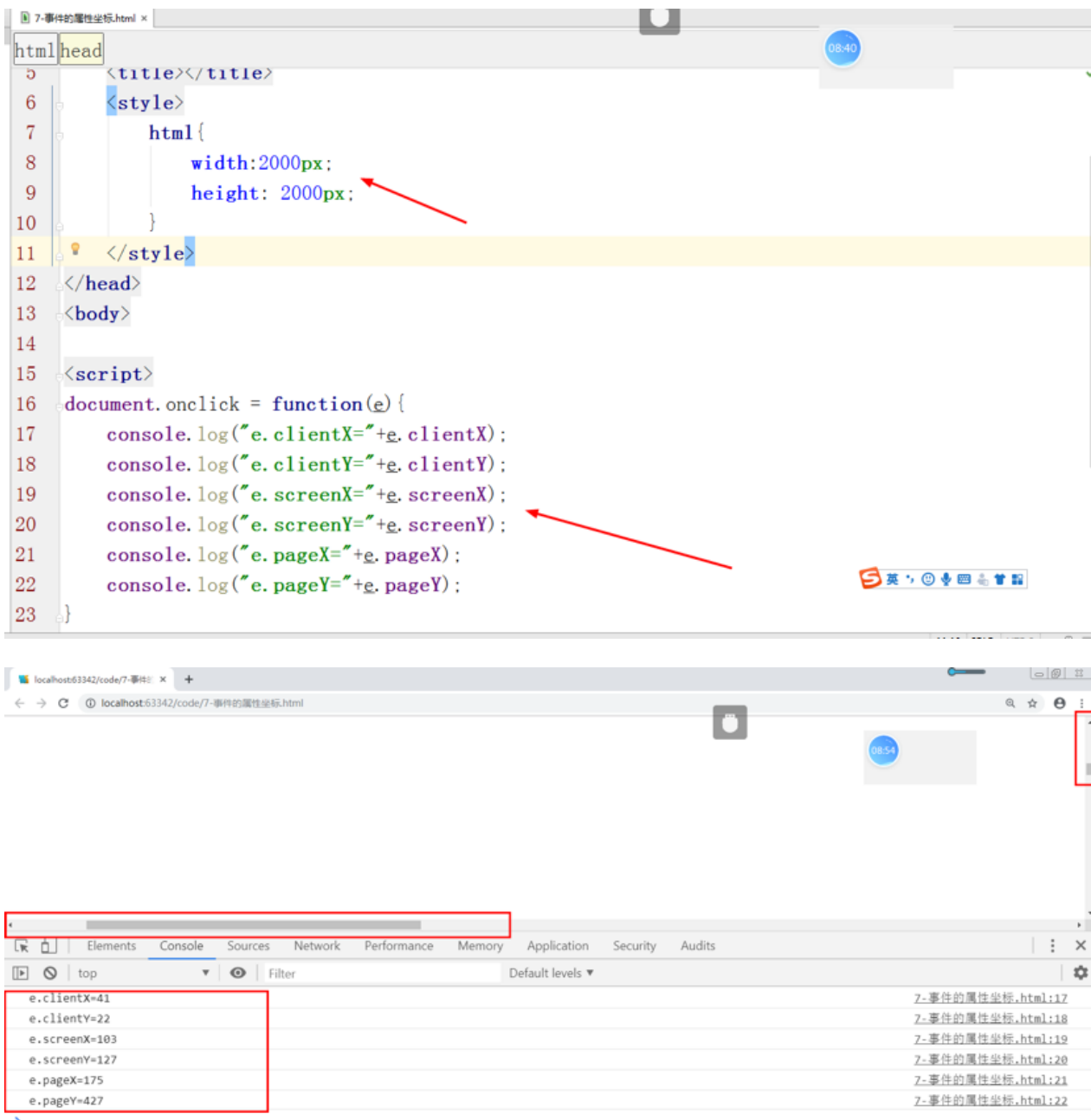
一些常见的键盘按键编码：

数字值	实际键值
48到57	0到9
65到90	a到z (A到Z)
112到135	F1到F24
8	BackSpace (退格)
9	Tab
13	Enter (回车)
20	Caps_Lock (大写锁定)
32	Space (空格键)
37	Left (左箭头)
38	Up (上箭头)
39	Right (右箭头)
40	Down (下箭头)

```
29
30 document.onkeypress = function(e) {
31     console.log(e.key, e.keyCode);
32 }
33
```

2.8. 获取当前坐标的属性

clientX	得到当前屏幕可视区域x坐标的值(不包含滚动条)
clientY	得到当前屏幕可视区域y坐标的值(不包含滚动条)
screenX	得到当前屏幕x坐标的值
screenY	得到当前屏幕y坐标的值
pageX	得到当前屏幕可视区域x坐标的值(包含滚动条)
pageY	得到当前屏幕可视区域y坐标的值(包含滚动条)



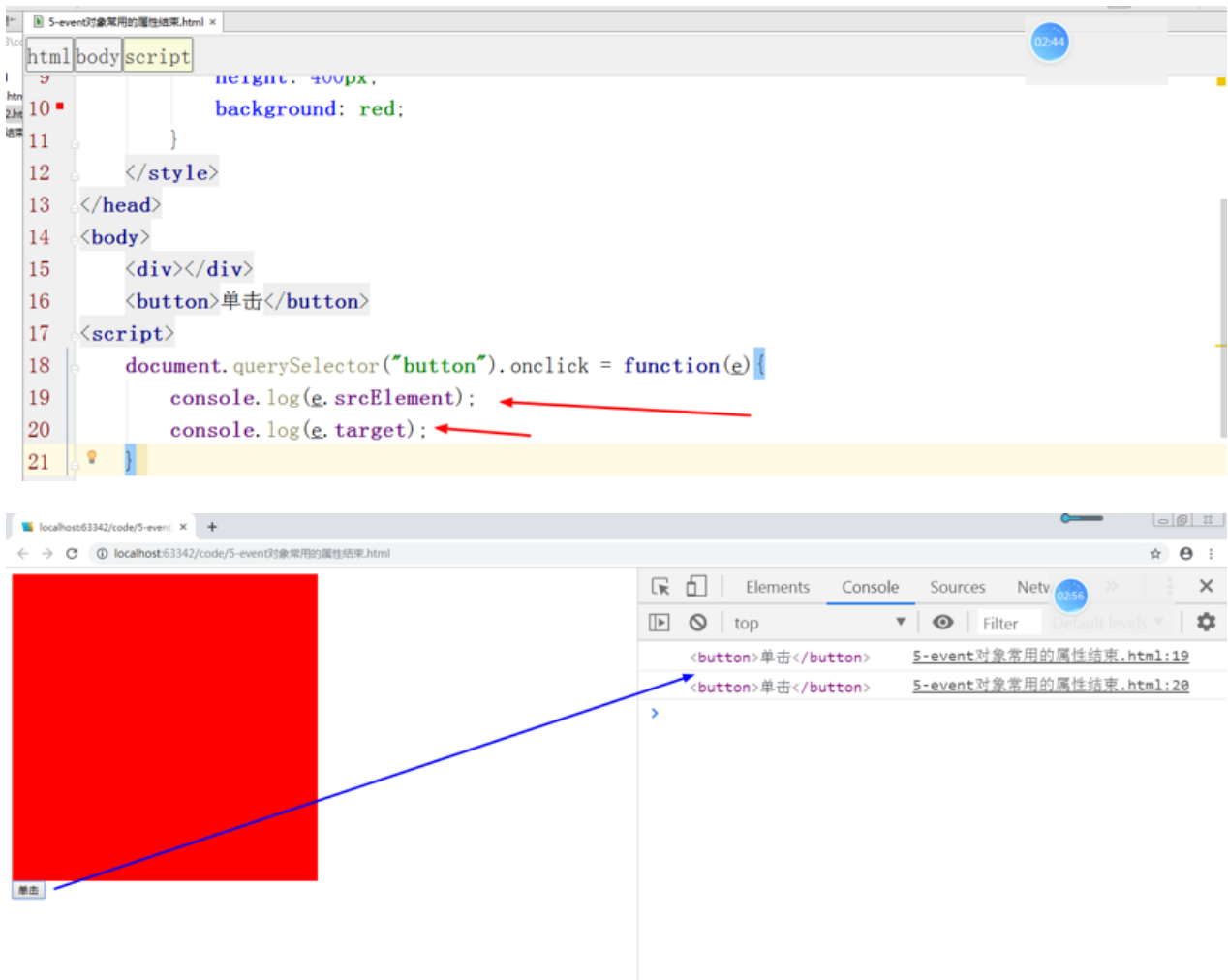
2.9. event中常用的方法

`event.stopPropagation()`: 阻止冒泡。`event.cancelBubble=true`阻止冒泡

`event.preventDefault()`: 默认阻止事件。`return false`默认阻止事件
`event.returnValue=false`默认阻止事件

3. IE中的event对象的常用属性和方法

1、srcElement (target) 属性：返回的是目标对象



2、cancelBubble 属性:取消冒泡

3、returnValue属性：默认阻止事件

4、兼容性问题

通过对IE下event的方法和非IE的方法的介绍，能够感觉到两种情况的很多属性和方法大致都相同，只不过会在某些特殊方法上面存在不同的兼容性写法。因此我们可以提出一些同时满足不同浏览器兼容性的写法。

属性的兼容性写法我们已经说过：

```
var target = eve.target || eve.srcElement;
```

```
var eve = eve || window.event
```

方法的兼容性写法我们也可以仿照这个来进行编写。思路如下：

(1)因为两种情况下的event对象获取方式并不同，所以希望能够自定义一个对象来替代event对象的使用。

(2)因为想要自定义对象在功能上和系统event对象的方法相同，所以需要给自定义对象添加方法

参考代码：

```
var Event = {
```

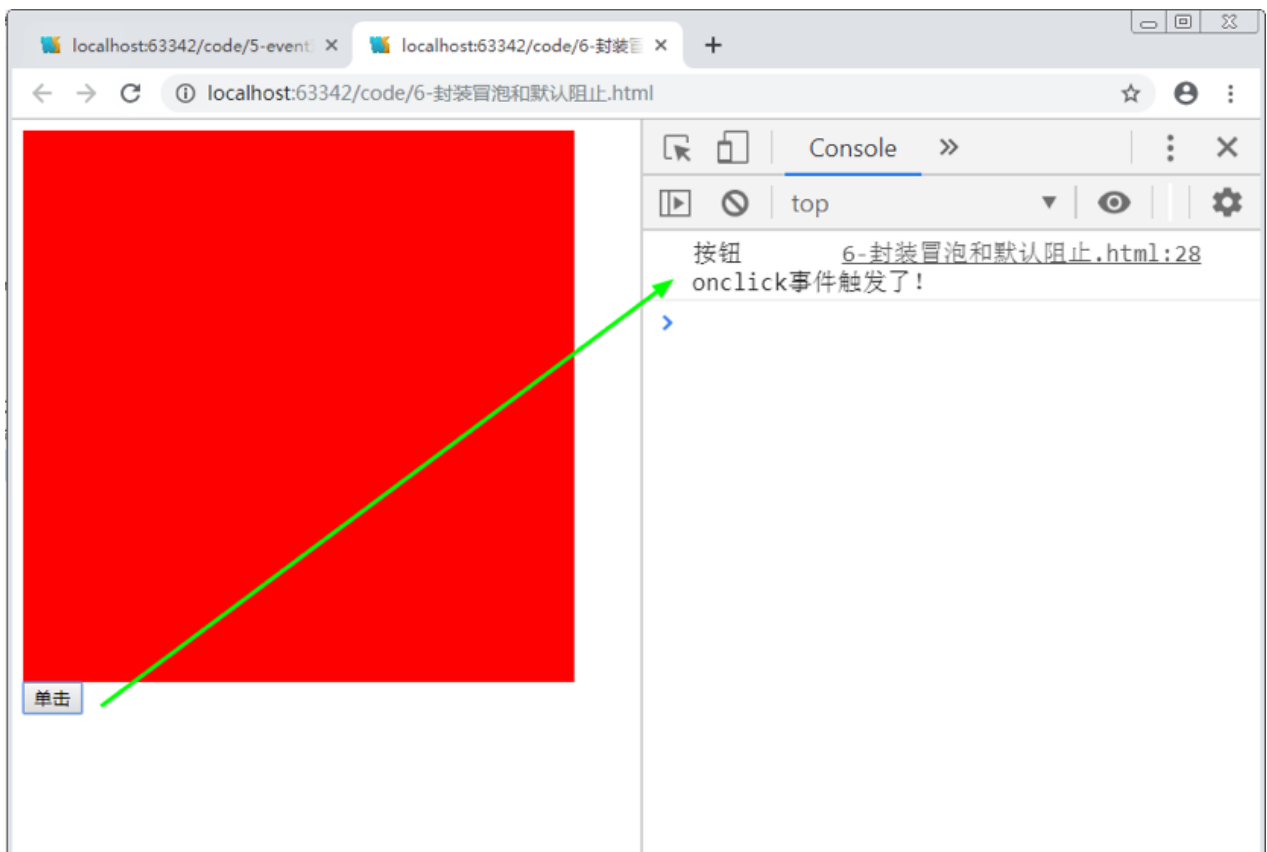
```
stop:function () { if(event.stopPropagation){  
event.stopPropagation(); }else { event.cancelBubble = true; } },
```

```
event.stopPropagation?event.stopPropagation():  
(event.cancelBubble = true)
```

```
cancelDefault:function () { if(event.preventDefault){  
event.preventDefault(); }else { event.returnValue = false; } } };
```

```
5-event对象常用的属性结束.html x 6-封装冒泡和默认阻止.html x
html body script
18 var obt = document.querySelector("button");
19 var myevent = {
20     stop: function() {
21         if (event.stopPropagation()) {
22             event.stopPropagation(); // 阻止冒泡
23         } else {
24             event.cancelBubble = true;
25         }
26     },
27     quxiao: function() {
28         if (event.preventDefault()) {
29             event.preventDefault(); // 取消默认行为
30         } else {
31             event.returnValue = false;
32         }
33     }
34 }
35
36
```

```
38 obt.onclick = function() {
39     console.log("按钮onclick事件触发了!");
40     myevent.stop();
41 }
42
43 document.onclick = function() {
44     console.log("document的onclick事件触发了!");
45 }
46
```



4. 鼠标滚轮事件

滚轮就是鼠标上的滚轮，它滚动的时候触发事件


4.1. 滚轮事件

onmousewheel就是鼠标滚动事件，mouse鼠标，wheel就是轮子。

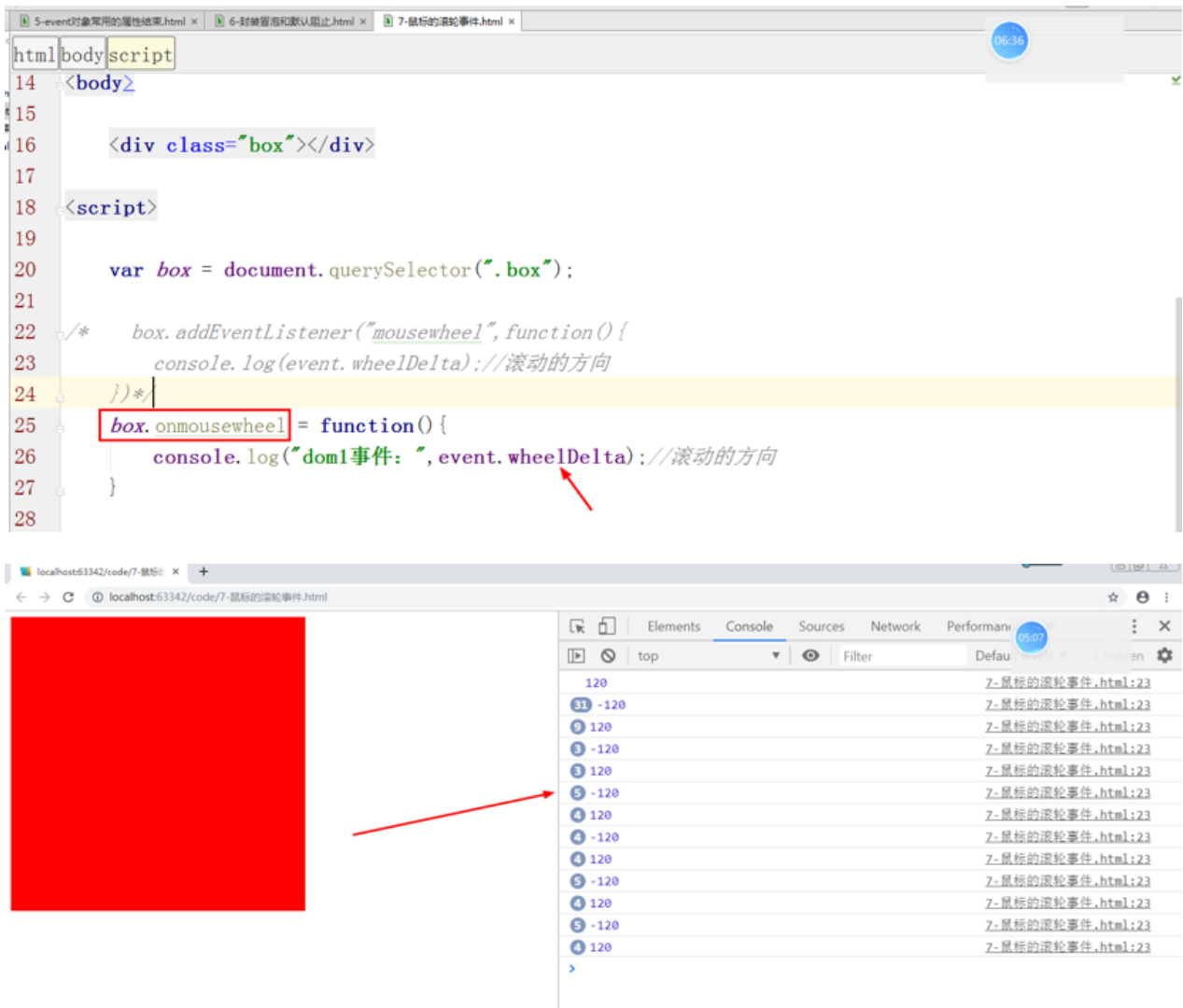
event参数最最重要的事就**event.wheelDelta**属性，表示滚动的方向。这是浏览器的规定：

鼠标往上滚， 120

鼠标往下滚， -120



```
html body script
14 <body>
15
16 <div class="box"></div>
17
18 <script>
19
20 var box = document.querySelector(".box");
21
22 box.addEventListener("mousewheel", function() {
23     console.log(event.wheelDelta); // 滚动的方向
24 })
25
26 </script>
27 </body>
28 </html>
```

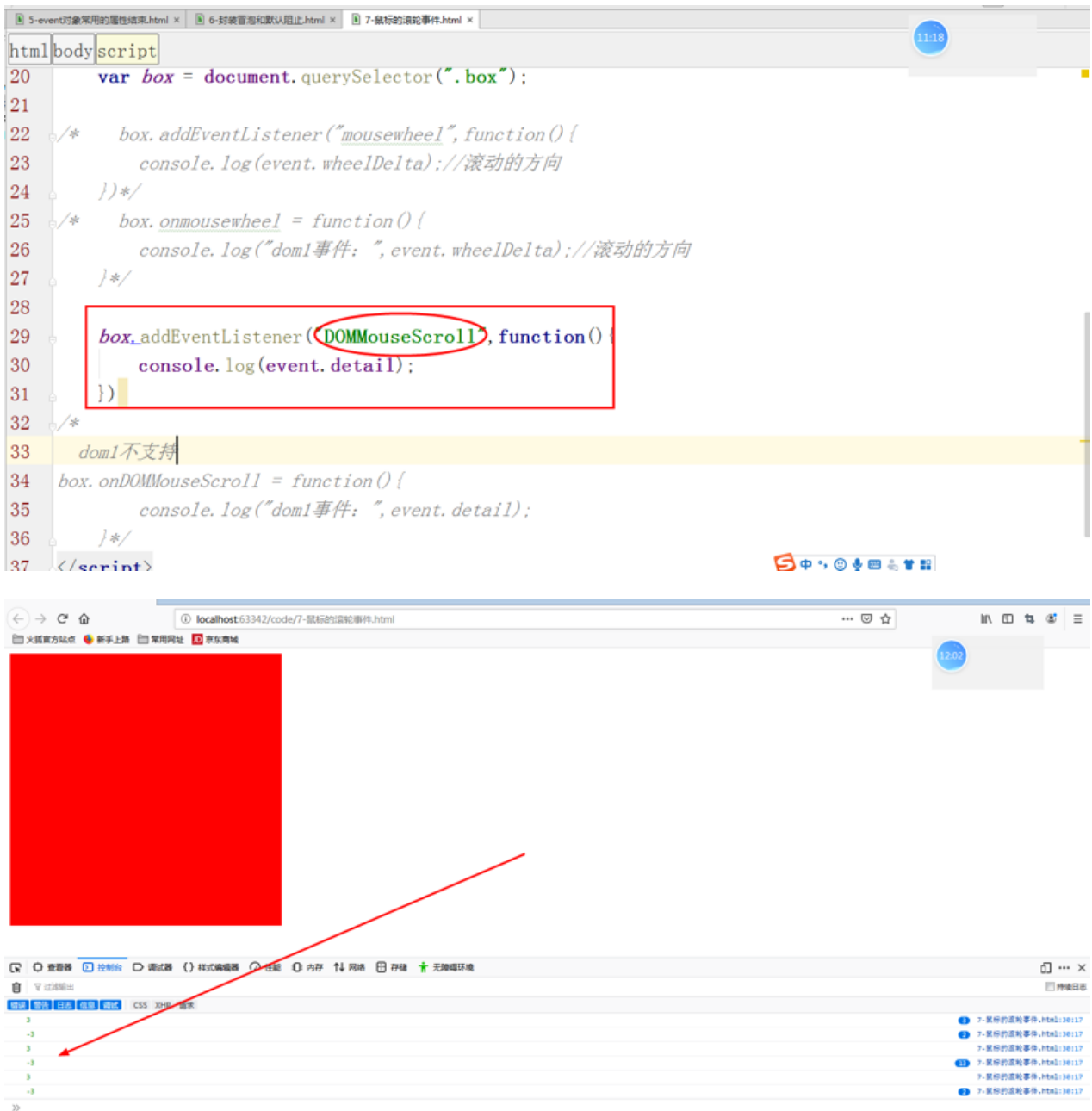


4.2. 火狐浏览器滚轮事件

火狐浏览器不兼容，火狐使用自己的专用事件 **DOMMouseScroll**，并且这个事件只能通过标准的 **DOM2级** 的事件绑定方式添加。

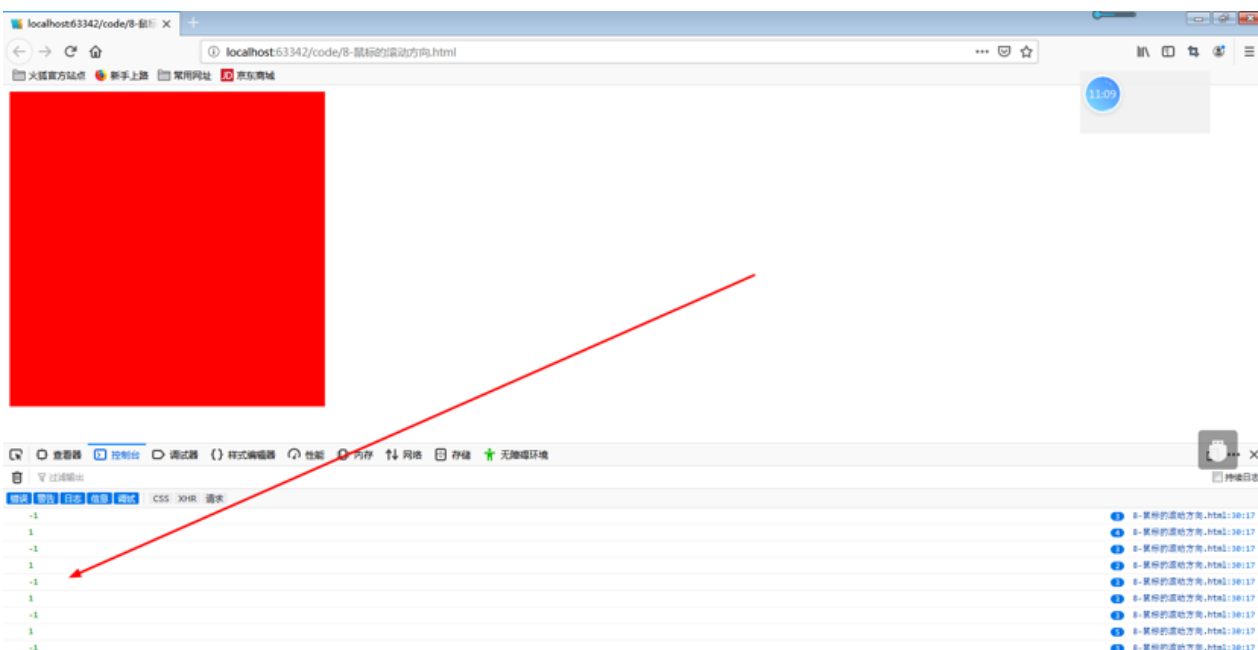
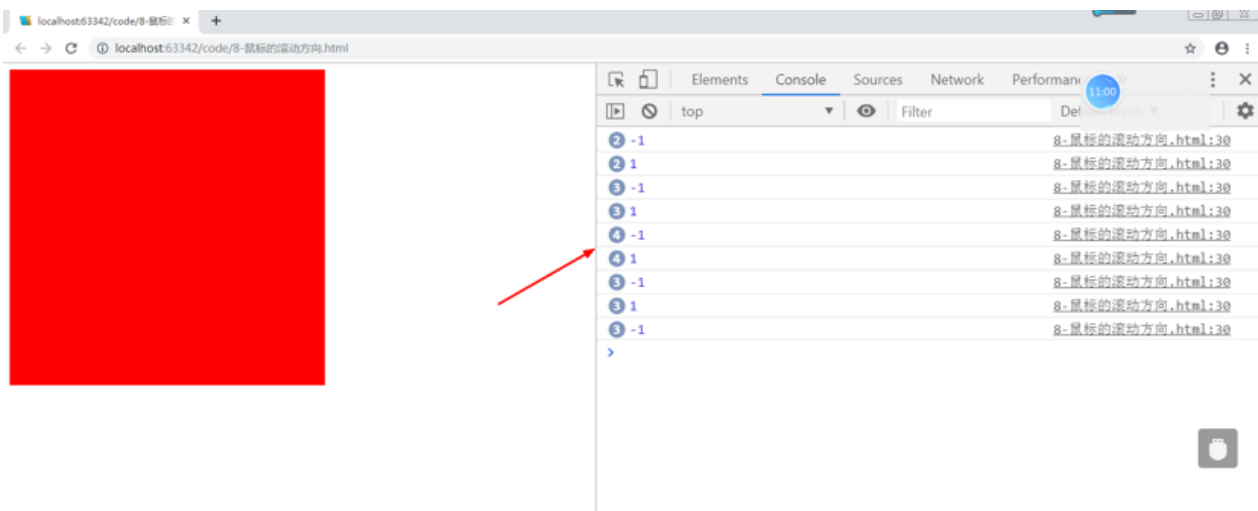
火狐添加的事件，是它自己的专门事件 **DOMMouseScroll**，表示滚动方向的事件的属性叫做 **event.detail**。

detail就是细节的意思。反着的，滚动方向往上，-3；滚动方向往下，3。



4.3. 封装浏览器兼容性

```
htmlbodyscript
17
18 <script>
19     var box = document.querySelector(".box");
20     //age = age>18?"成年人":"未成年人";
21
22     function gd(e) {
23         var direction = 0;
24         if(e.wheelDelta) {
25             direction = e.wheelDelta>0?1:-1;
26         }
27         else{
28             direction = e.detail<0?1:-1;
29         }
30         console.log(direction);
31     }
32     box.onmousewheel = gd; //针对谷歌
33     box.addEventListener("DOMMouseScroll", gd); //针对火狐
34
35 </script>
```

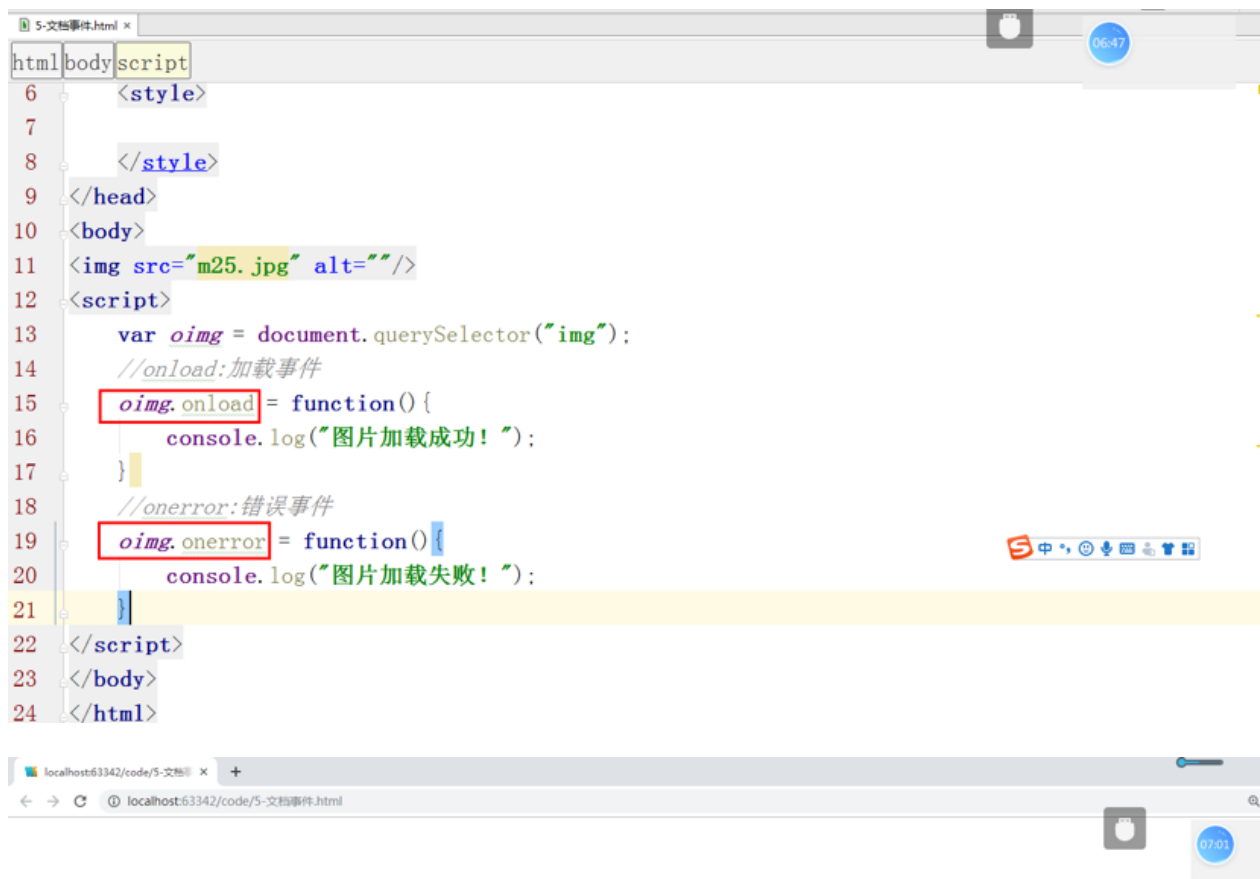


5. 文档事件

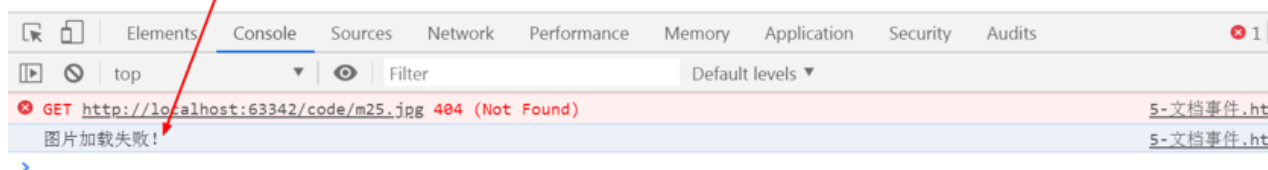
5.1. 加载成功\失败事件：load\error

load事件指的是：节点加载成功时自动发生回调事件

error事件指的是：节点加载失败时自动发生的回调事件



```
5-文档事件.html x
htmlbodyscript
6    <style>
7
8    </style>
9  </head>
10 <body>
11 
12 <script>
13   var oimg = document.querySelector("img");
14   //onload:加载事件
15   oimg.onload = function() {
16     console.log("图片加载成功!");
17   }
18   //onerror:错误事件
19   oimg.onerror = function() {
20     console.log("图片加载失败!");
21   }
22 </script>
23 </body>
24 </html>
```



5.2. 当DOM加载完成时触发事件： **DOMContentLoaded**

DOMContentLoaded事件和load事件的区别是触发的时机不一样，先触发**DOMContentLoaded**事件，后触发**load**事件。

DOM文档加载的步骤为：解析HTML结构。加载外部脚本和样式表文件。解析并执行脚本代码。DOM树构建完成。

//DOMContentLoaded执行 加载图片等外部文件。页面加载完毕。 //load执行

5.3. 文档加载状态判断事件： **readystatechange**

众所周知，document节点中拥有一个属性叫做readyState。其拥有三个可能值：

loading：加载DOM中

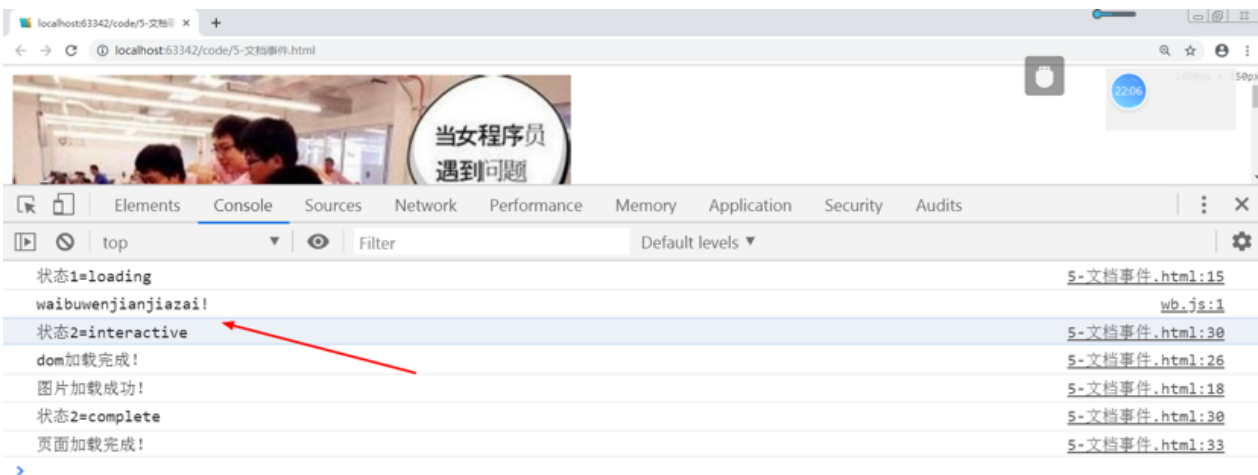
interactive：加载外部资源

complete：加载完成

而readystatechange事件正是在这个状态发生改变时调用的事件。

```
html body script
10 </head>
11 <body>
12 
13 <script>
14     var oimg = document.querySelector("img");
15     console.log("状态1="+document.readyState);//dom加载状态
16     //onload:加载事件
17     oimg.onload = function() {
18         console.log("图片加载成功!");
19     }
20     //onerror:错误事件
21     oimg.onerror = function() {
22         console.log("图片加载失败!");
23     }
24
25     document.addEventListener("DOMContentLoaded", function() {
26         console.log("dom加载完成!");
27     })
28
```

```
29     document.addEventListener("readystatechange", function() {
30         console.log("状态2="+document.readyState);
31     })
32     window.onload = function() {
33         console.log("页面加载完成!");
34     }
35 </script>
36 <script src="wb.js"></script>
37 </body>
38 </html>
```

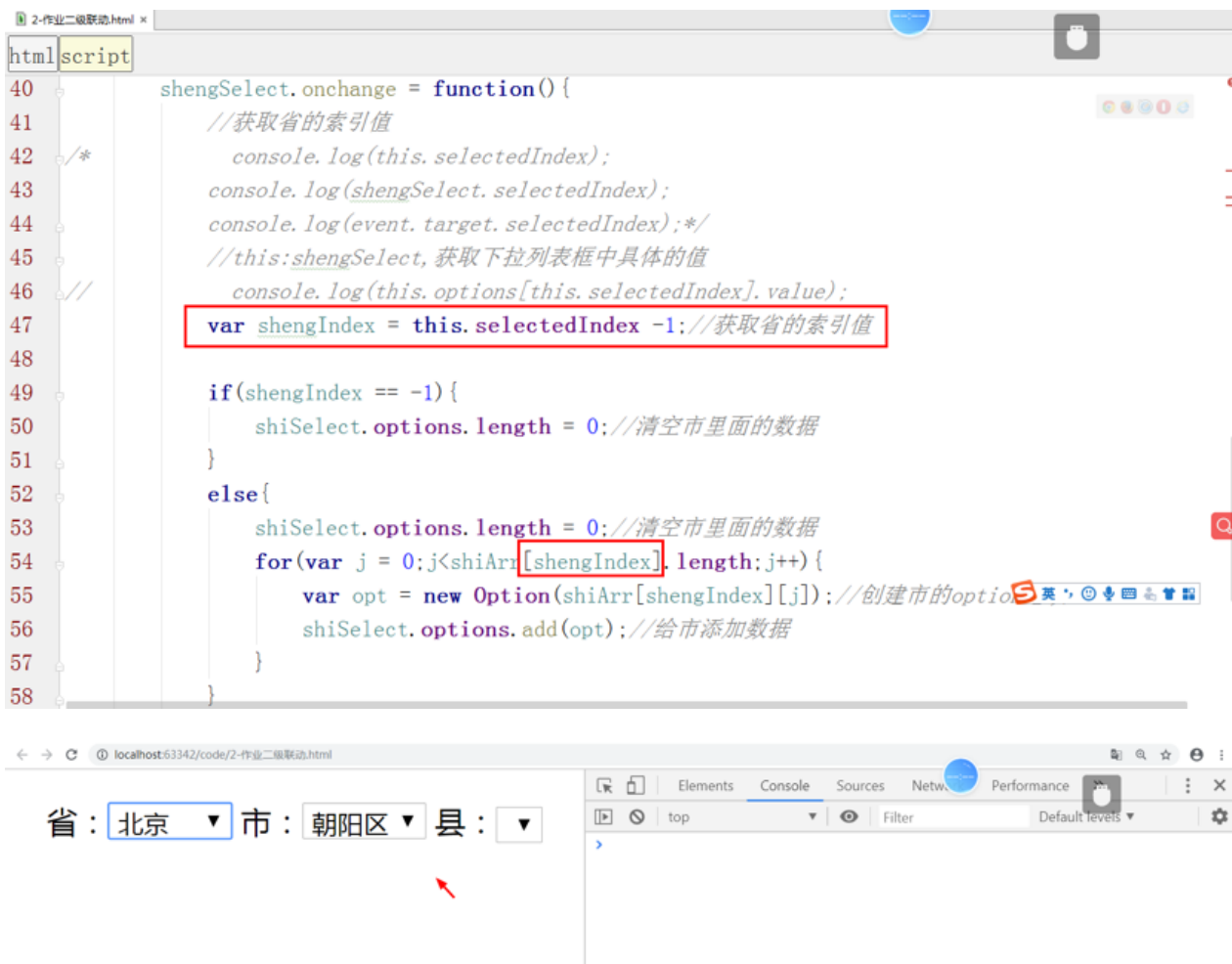


6. Event案例

6.1. 二级联动

```
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Document</title>
6 </head>
7 <body>
8
9 <p align="center">
10   <span>省: </span><select name="" id="" class="shengSelect">
11     <option value="">请选择</option>
12   </select>
13   <span>市: </span><select name="" id="" class="shiSelect"></select>
14   <span>县: </span><select name="" id="" class="xianSelect"></select>
15 </p>
16
17 </body>
18 <script>
19
```

```
20 window.onload = function() {
21   //获取相应的元素
22   var shengSelect = document.querySelector(".shengSelect");
23   var shiSelect = document.querySelector(".shiSelect");
24
25   var shengArr = ["北京", "东京", "纽约"];
26
27   var shiArr = [
28     ["朝阳区", "海淀区", "大兴区"],
29     ["新宿区", "文京区", "中央区", "台东区", "港区"],
30     ["曼哈顿", "皇后区", "布朗克斯区"]
31   ];
32   //给省添加数据
33   for(var i = 0; i < shengArr.length; i++) {
34     var opt = new Option(shengArr[i]); //创建option选项
35     // console.log(opt);
36     shengSelect.options.add(opt); //给select添加option选项
37     // console.log(shengSelect.options);
38   }
39 }
```



6.2. 三级联动

6.3. 放大镜案例

7. 事件委托

7.1. 什么是事件委托

事件委托/事件代理：事件委托就是利用【事件冒泡】，自己本身做不了这个事，让让上一级来做这个事，只指定一个事件处理程序，就可以管理某一类型的所有事件。


7.2. 具体实例：

Li委托ul来做这个事

```

14 <script>
15     var oli = document.querySelectorAll("li");
16     var oul = document.querySelector("ul");
17     //创建的Li元素
18     var ali = document.createElement("li");
19     ali.innerHTML = "很好! ";
20     oul.appendChild(ali);
21
22     oul.onclick = function() {
23         console.log(event.target); //返回的对象的真正触发者
24         console.log(event.target.innerHTML);
25     }
26     /* for(var i=0;i<oli.length;i++){
27         oli[i].onclick = function() {
28             console.log(oli[i].innerHTML);
29             console.log(this);
30             console.log(this.innerHTML);
31         }
32     } */

```



The screenshot shows a web browser window with a list of items: 11, 22, 33, and 很好!. A red arrow points from the text '很好!' to the Chrome DevTools Console. The Console shows the following log entries:

- 很好! (3-事件委托.html:23)
- 很好! (3-事件委托.html:24)
- 33 (3-事件委托.html:23)
- 33 (3-事件委托.html:24)
- 22 (3-事件委托.html:23)
- 22 (3-事件委托.html:24)
- 11 (3-事件委托.html:23)
- 11 (3-事件委托.html:24)