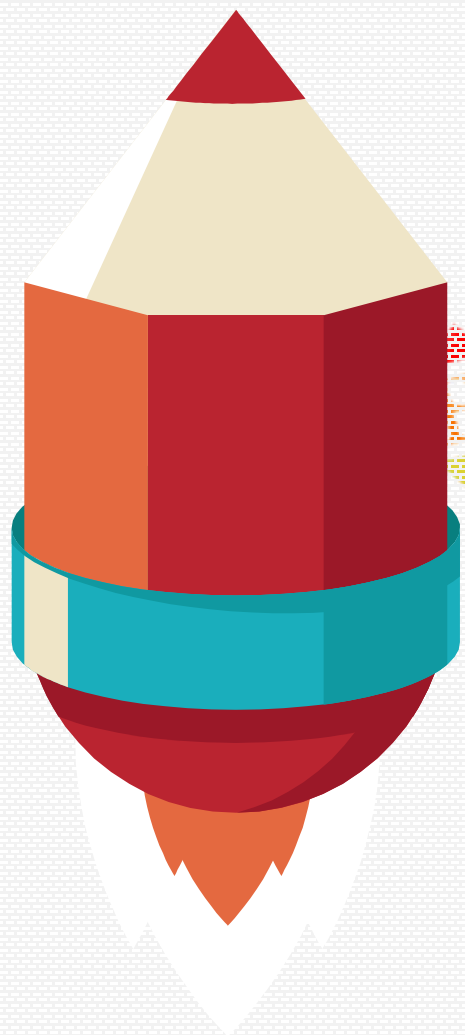




第19课：BOM高级事件

■ 主讲老师：万章



四知

三种事件类型

UI事件

焦点/鼠标/滚轴事件

键盘事件



三种事件类型

事件简介

在浏览器发生的事情统称为事件，比如点击事件，鼠标移动事件，键盘事件，点击事件，请求事件，加载完成事件。面对突发事件，我们都需要用函数处理进行处理

函数可以处理事件但是函数摸不着事件的头脑。于是，浏览器出面帮助函数处理事件，浏览器整理了一下事件触发的一状态，整理成了一个事件对象（也就是事件触发函数传入的ev参数或是window.event）。事件对象存储了事件触发的各种状态 包括事件触发的主体对象，事件类型，事件触发的位置等等。

事件绑定

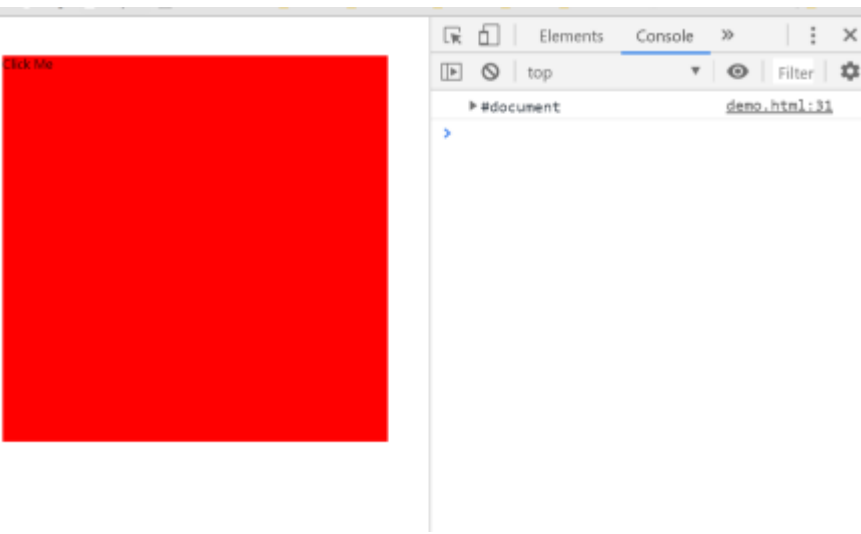
Obj.on+事件名称=function(){}

```
document.onclick = function () {  
    console.log(1)  
}
```

//点击文档 打印1

```
document.onclick = function () {  
    console.log(this)  
}
```

//点击文档 打印document元素



1. 事件绑定同一个对象只能给同一个事件绑定唯一——一个事件处理函数.如果绑定第二个,第一个会被清除掉.因为本质上只是给对象的on事件属性上添加了一个函数
2. 事件绑定函数得this指向当前调用(触发)事件的主体对象(元素)
3. 在绑定事件之前, 事件属性的处理函数默认是null
4. 清除事件的方式只需要将此事件的触发函数改成null

我们称on这种方式绑定的事件是DOM0级事件

优点: 兼容所有的浏览器, 因为它是最原始的事件, js出来的那一天事件都是这么绑定的。

缺点: 这种绑定事件的方法, 一个对象只能绑定一个事件, 不能绑定多个事件。

事件监听

`Obj.addEventListener(“事件名称”, function() {}, 布尔值)`

当事件触发的时候，我们不直接写明处理的函数来响应事件。

咱们叫一个播报员去某个元素那守着，如果触发了某个事件，那就回来通知我，然后我指派函数执行。这种行为类似监听。我们管他叫事件监听

```
document.addEventListener("click", function () {  
    console.log(1)  
})  
document.addEventListener("click", function () {  
    console.log(2)  
})  
//点击 打印1 2
```

我们称这种事件的监听方式是DOM2级事件

1. 事件监听接受两个参数(三个) 第一个是需要监听的事件类型，第二个是事件的触发的回调函数，第三个是是否事件冒泡
2. 事件类型不需要加on+事件类型，直接事件类型即可。
3. 一次可以绑定多个事件，相互之间不影响。并且触发顺序就是绑定顺序
4. 事件监听处理函数得this指向当前调用(触发)事件的主体对象
5. 取消事件监听用对应的方式，removeEventListener，传入的参数要和添加的参数完全一致

事件监听

```
function handle(e) {  
  if (e.target.tagName.toLowerCase() === "li") {  
    console.log(1)  
  }  
}  
let li = document.querySelector("li")  
li.addEventListener("click", handle)  
// 点击li 触发  
li.removeEventListener("click", handle)  
// 点击就没有反应了
```

有名函数是可以直接取消绑定的

```
li.addEventListener("click", function () {  
  console.log(1)  
})  
// 点击触发  
li.removeEventListener("click", function () {  
  console.log(1)  
})  
// 关闭不了 因为此function不是彼function
```

特殊的：匿名事件处理函数是没法取消绑定事件的

其中addEventListener不兼容低版本的IE浏览器

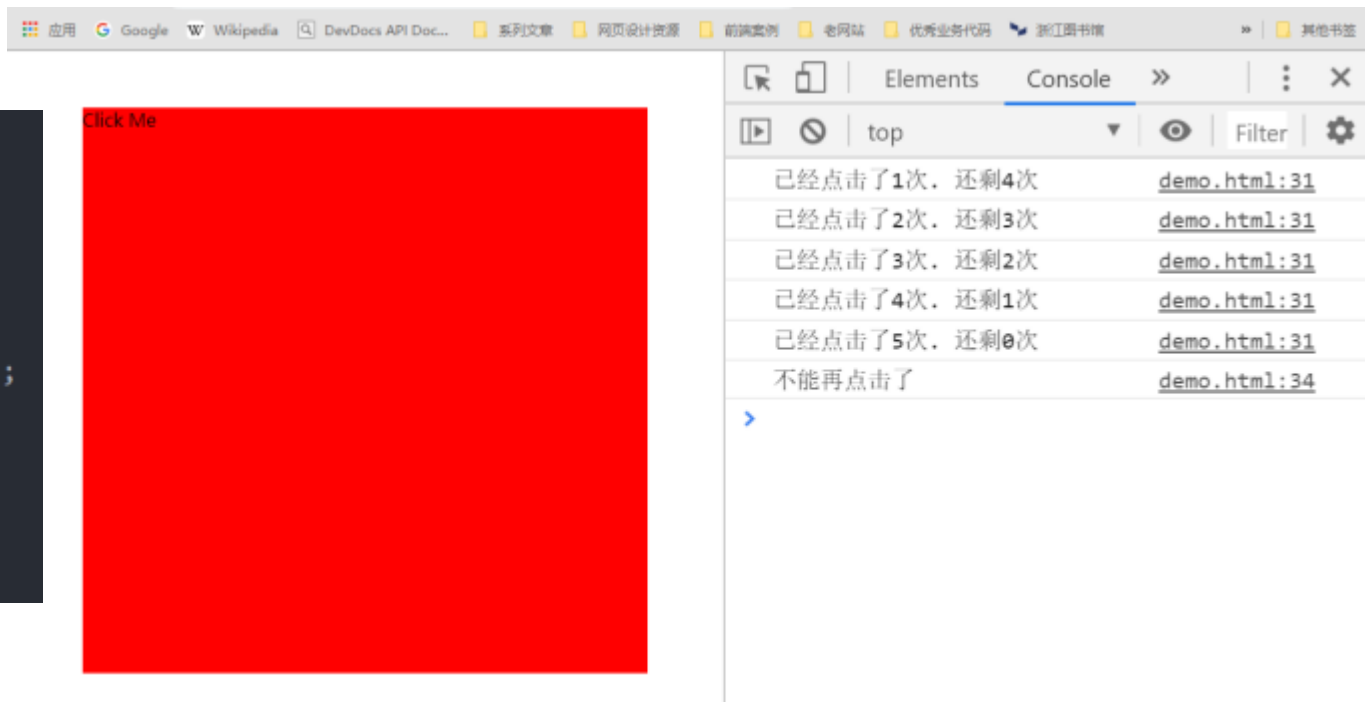
在低版本的IE浏览器中使用：attachEvent和detachEvent

事件监听之奇淫巧技

```
let dom = document.getElementById("content"),
    count = 0;
const MAXCOUNT=5;

dom.addEventListener("click", function (e) {
    count++;
    console.log('已经点击了' + count + "次, 还剩"+ (MAXCOUNT-count) + "次");
    if (count > 4) {
        dom.removeEventListener(e.type, arguments.callee, false);
        console.log("不能再点击了");
    }
});
```

Click Me



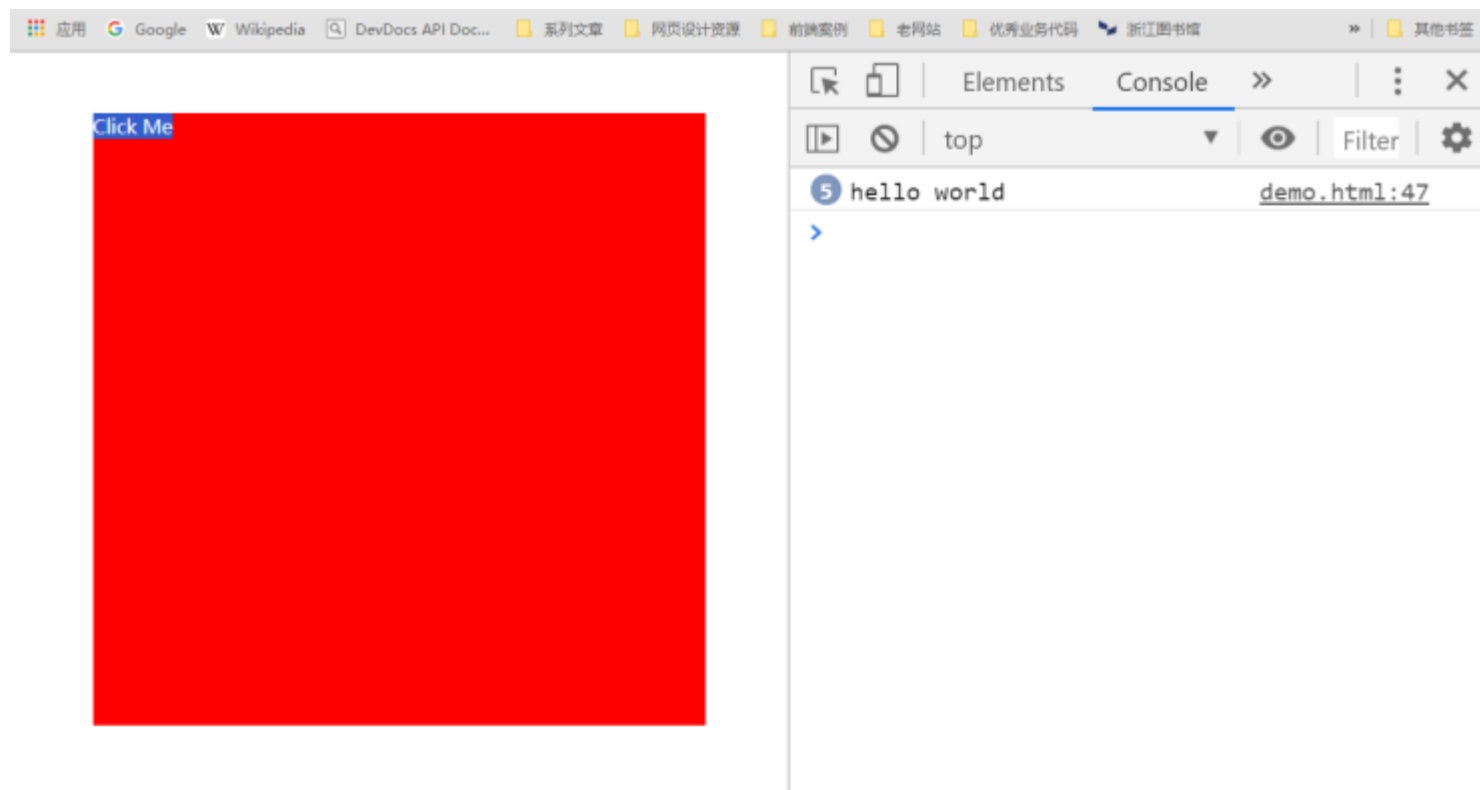
解绑匿名函数，函数的自我清除，哈哈哈!!!

事件添加的通用方法

```
let EventUtil = {
  addHandle: function (el, type, handler) {
    if (el.addEventListener) {
      el.addEventListener(type, handler)
    } else if (el.attachEvent) {
      el.attachEvent("on" + type, handler)
    } else {
      el["on" + type] = handler
    }
  },
  removeHandle: function (el, type, handler) {
    if (el.removeEventListener) {
      el.removeEventListener(type, handler)
    } else if (el.deleteEvent) {
      el.deleteEvent("on" + type, handler)
    } else {
      el["on" + type] = null
    }
  }
}
```

// 针对event的兼容

```
let con=document.querySelector("div");
EventUtil.addHandle(con,"click",function(){
  console.log("hello world");
})
```



兼容性处理

事件委托

当我们需要绑定事件的时候，需要给事件添加绑定或者监听。当事件触发的时候，我们会对事件的主体进行操作。但是如果有大量dom对象需要绑定事件，并且事件的处理函数是同一个的时候。这种绑定方式或者监听方式就是很蠢。并且修改起来也会很麻烦。

```
let ali = document.querySelectorAll("li")

for (let i = 0, l = ali.length; i < l; i++) {
  ali[i].addEventListener("click", function fn(e) {
    console.log(this.innerText)
  }) // 点击每个li输出这个li元素的文本内容

  // ali[i].onclick = function fn(e) {
  //   console.log(this.innerText)
  // }
}

// 修改的话需要再循环一次删除和添加
```

这是最糟糕的形式，每个元素的点击事件都有一个一模一样的函数，而且每一个函数都会在内存里面单独开辟一个空间进行存储，既慢还浪费内存空间

```
let ali = document.querySelectorAll("li")

function fn(e) {
  // handle it
  console.log(this.innerText)
}

for (let i = 0, l = ali.length; i < l; i++) {
  ali[i].addEventListener("click", fn) // 点击每个li输出这个li元素的文本内容
  // ali[i].onclick = fn
}

// 修改的话需要再循环一次删除和添加
```

这种稍好，至少省去了内存里面大量重复的函数，但是依然需要每个元素都来添加一次，如果要更改函数的话，还得重新循环一次，浪费时间

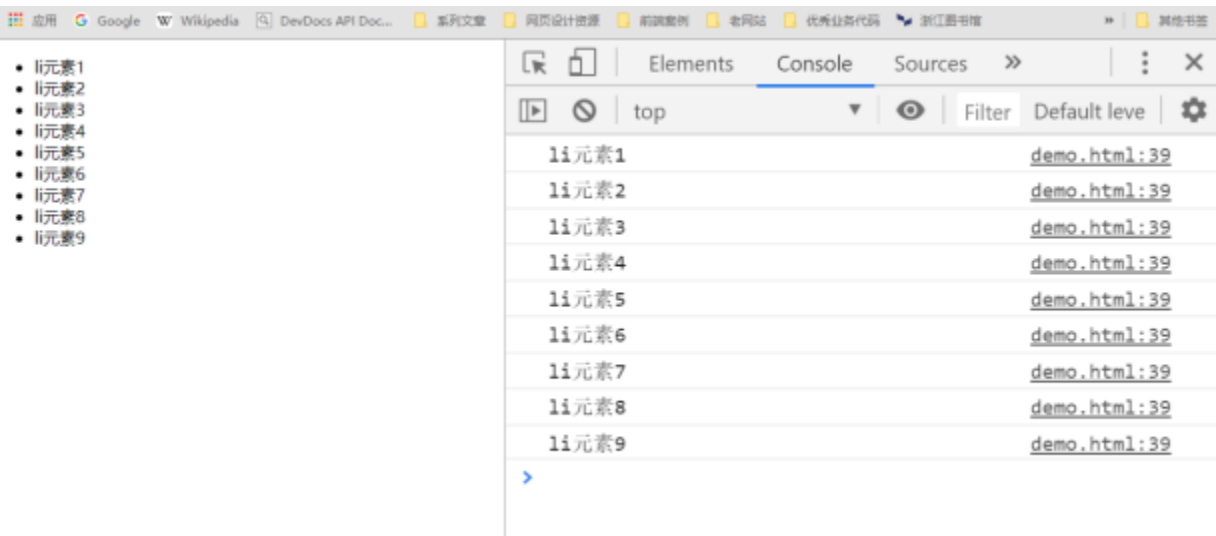
事件委托

于是我们就设计了另外一个监听方式，通过监听公共父元素的形式监听事件。

```
let list = document.querySelector("ul")
let ali = document.querySelectorAll("li")

function fn(ev) {
  if(ev.target.tagName.toLowerCase()=== "li"){
    // 当我们点击到了ul元素里面的li元素时就输出这个li元素的文本内容
    console.log(ev.target.innerText);
  }
}

list.addEventListener("click", fn)
```



在事件处理函数fn中，咱们可以通过事件对象来进行识别和判断触发事件的对象,从而达到事件处理的目的。识别触发事件的对象是用e.target（当前触发函数的DOM对象）来控制的。

比如：再一个DOM结构中有`ul>li*100`这样的结构，我们对ul进行监听，比如点击事件。当我们点击ul下面的li的时候： 会触发ul的点击事件(因为li是在ul中，所以也会触发。事件捕捉)。然后对此事件的target与自己设定的目标元素进行判断，是否符合要求

事件委托概念模型



需求:有人戳到身上的某个位置就说某个位置疼

事件绑定或是事件监听

1. 当有人戳到我的手, 我就说手疼
2. 当有人戳到我的头, 我就说头疼
3. 当有人戳到我的眼, 我就说眼疼
4. 当有人戳到我的背, 我就说背疼
5. 当有人戳到我的屁股, 我就说屁股疼

事件委托

1. 当有人戳到我, 戳我哪里我就说哪里疼



UI事件

UI事件

load: 当页面完全加载后在window 上面触发, 当所有框架都加载完毕时在框架集上面触发, 当图像加载完毕时在元素上面触发, 或者当嵌入的内容加载完毕时在<object>元素上面触发。

error: 当发生JavaScript 错误时在window 上面触发, 当无法加载图像时在元素上面触发, 当无法加载嵌入内容时在<object>元素上面触发, 或者当有一或多个框架无法加载时在框架集上面触发。

select: 当用户选择文本框 (<input>或<texterea>) 中的一或多个字符时触发。

resize: 当窗口或框架的大小变化时在window 或框架上面触发。

scroll: 当用户滚动带滚动条的元素中的内容时, 在该元素上面触发。<body>元素中包含所加载页面的滚动条。多数这些事件都与window 对象或表单控件相关。



焦点/鼠标/滚轴事件

焦点事件

1. blur: 在元素失去焦点时触发。这个事件不会冒泡;
2. focus: 在元素获得焦点时触发。这个事件不会冒泡;
3. focusin: 在元素获得焦点时触发。这个事件与HTML 事件focus 等价, 但它冒泡
4. focusout: 在元素失去焦点时触发。这个事件是HTML 事件blur 的通用版本。

当焦点从页面中的一个元素移动到另一个元素, 会依次触发下列事件:

- (1) focusout 在失去焦点的元素上触发;
- (2) focusin 在获得焦点的元素上触发;
- (3) blur 在失去焦点的元素上触发;
- (4) focus 在获得焦点的元素上触发;

鼠标事件

详细的鼠标事件在之前就讲过了，现在在这里我们再来梳理一下事件的触发顺序

只有在同一个元素上相继触发mousedown 和mouseup 事件，才会触发click 事件；如果mousedown 或mouseup 中的一个被取消，就不会触发click 事件。

类似地，只有触发两次click 事件，才会触发一次dblclick 事件。如果有代码阻止了连续两次触发click 事件（可能是直接取消click事件，也可能通过取消mousedown 或mouseup 间接实现），那么就不会触发dblclick 事件了。这4个事件触发的顺序始终如下：

- (1) mousedown
- (2) mouseup
- (3) click
- (4) mousedown
- (5) mouseup
- (6) click
- (7) dblclick

显然，click 和dblclick 事件都会依赖于其他先行事件的触发；而mousedown 和mouseup 则不受其他事件的影响。

鼠标事件

在按下鼠标时键盘上的某些键的状态也可以影响到所要采取的操作。这些修改键就是Shift、Ctrl、Alt 和Meta（在Windows 键盘中是Windows 键，在苹果机中是Cmd 键），它们经常被用来修改鼠标事件的行为。

DOM 为此规定了4 个属性，表示这些修改键的状态：shiftKey、ctrlKey、altKey 和metaKey。这些属性中包含的都是布尔值，如果相应的键被按下了，则值为true，否则值为false。当某个鼠标事件发生时，通过检测这几个属性就可以确定用户是否同时按下了其中的键

例：如果用户在按下了ctrl键的同时鼠标点击了，那么这个事件对象里面的ctrlKey的值就是true



```
demo.html:40
MouseEvent {isTrusted: true, screenX: -1572, screenY: 198, clientX: 68, clientY: 34, ...}
  altKey: false
  bubbles: true
  button: 0
  buttons: 0
  cancelBubble: false
  cancelable: true
  clientX: 68
  clientY: 34
  composed: true
  ctrlKey: false
  currentTarget: null
  defaultPrevented: false
  detail: 1
  eventPhase: 0
  fromElement: null
  isTrusted: true
  layerX: 68
  layerY: 34
  metaKey: false
  movementX: 0
  movementY: 0
  offsetX: 28
  offsetY: 18
  pageX: 68
  pageY: 34
  path: (6) [li, ul, body, html, document, Window]
  relatedTarget: null
  returnValue: true
  screenX: -1572
  screenY: 198
  shiftKey: false
  sourceCapabilities: InputDeviceCapabilities {firesTouchEvents: false}
  srcElement: li
  target: li
  timeStamp: 2992.2349999978906
  toElement: li
  type: "click"
  view: Window {postMessage: f, blur: f, focus: f, close: f, parent: Wind...
    which: 1
    x: 68
    y: 34
```

滚轴事件

当用户通过鼠标滚轮与页面交互、在垂直方向上滚动页面时（无论向上还是向下，有些特种鼠标还可以横向滑动），就会触发mousewheel事件

这个事件可以在任何元素上面触发，最终会冒泡到document（IE8）或window（IE9、Opera、Chrome 及Safari）对象。

多数时间触发的情况下，只要知道鼠标滚轮滚动的方向就够了，而这通过检测wheelDelta 的正负号就可以确定。
-向下，+向上， 数字时滚动了多少像素(和鼠标的灵敏度设置有关)

```
demo.html:40
▼ WheelEvent {isTrusted: true, deltaX: -0, deltaY: 25, deltaZ: 0, deltaMode: 0, ...}
  altKey: false
  bubbles: true
  button: 0
  buttons: 0
  cancelBubble: false
  cancelable: true
  clientX: 87
  clientY: 60
  composed: true
  ctrlKey: false
  currentTarget: null
  defaultPrevented: false
  deltaMode: 0
  deltaX: -0
  deltaY: 25
  deltaZ: 0
  detail: 0
  eventPhase: 0
  fromElement: null
  isTrusted: true
  layerX: 87
  layerY: 60
  metaKey: false
  movementX: 0
  movementY: 0
  offsetX: 47
  offsetY: 2
  pageX: 87
  pageY: 60
  path: (6) [li, ul, body, html, document, Window]
  relatedTarget: null
  returnValue: true
  screenX: -1833
  screenY: 193
  shiftKey: false
  sourceCapabilities: null
  srcElement: li
  target: li
  timeStamp: 2007.0200000045588
  toElement: li
  type: "wheel"
  view: Window {postMessage: f, blur: f, focus: f, close: f, parent: Window, ...}
    wheelDelta: -30
    wheelDeltaX: 0
    wheelDeltaY: -30
    which: 0
    x: 87
    y: 60
```



键盘事件

键盘事件

用户在使用键盘时会触发键盘事件,有3 个键盘事件, 简述如下。

keydown: 当用户按下键盘上的任意键时触发, 而且如果按住不放的话, 会重复触发此事件。

keypress: 当用户按下键盘上的字符键时触发, 而且如果按住不放的话, 会重复触发此事件。

按下Esc 键也会触发这个事件。

keyup: 当用户释放键盘上的键时触发。

虽然所有元素都支持以上3 个事件, 但只有在用户通过文本框输入文本时才最常用到。然后就是一些个网页游戏上

在用户按了一下键盘上的字符键时,

1. 首先会触发keydown 事件
2. 然后紧跟着是keypress 事件,
3. 最后会触发keyup 事件。

其中, keydown 和keypress 都是在文本框发生变化之前被触发的;

而keyup事件则是在文本框已经发生变化之后被触发的。

如果用户按下了一个字符键不放, 就会重复触发keydown 和keypress 事件, 直到用户松开该键为止

键盘事件

如果用户按下的是一个非字符键：

1. 那么首先会触发keydown 事件,
2. 然后就是keyup 事件

如果按住这个非字符键不放，那么就会一直重复触发keydown 事件，直到用户松开这个键，此时会触发keyup事件。

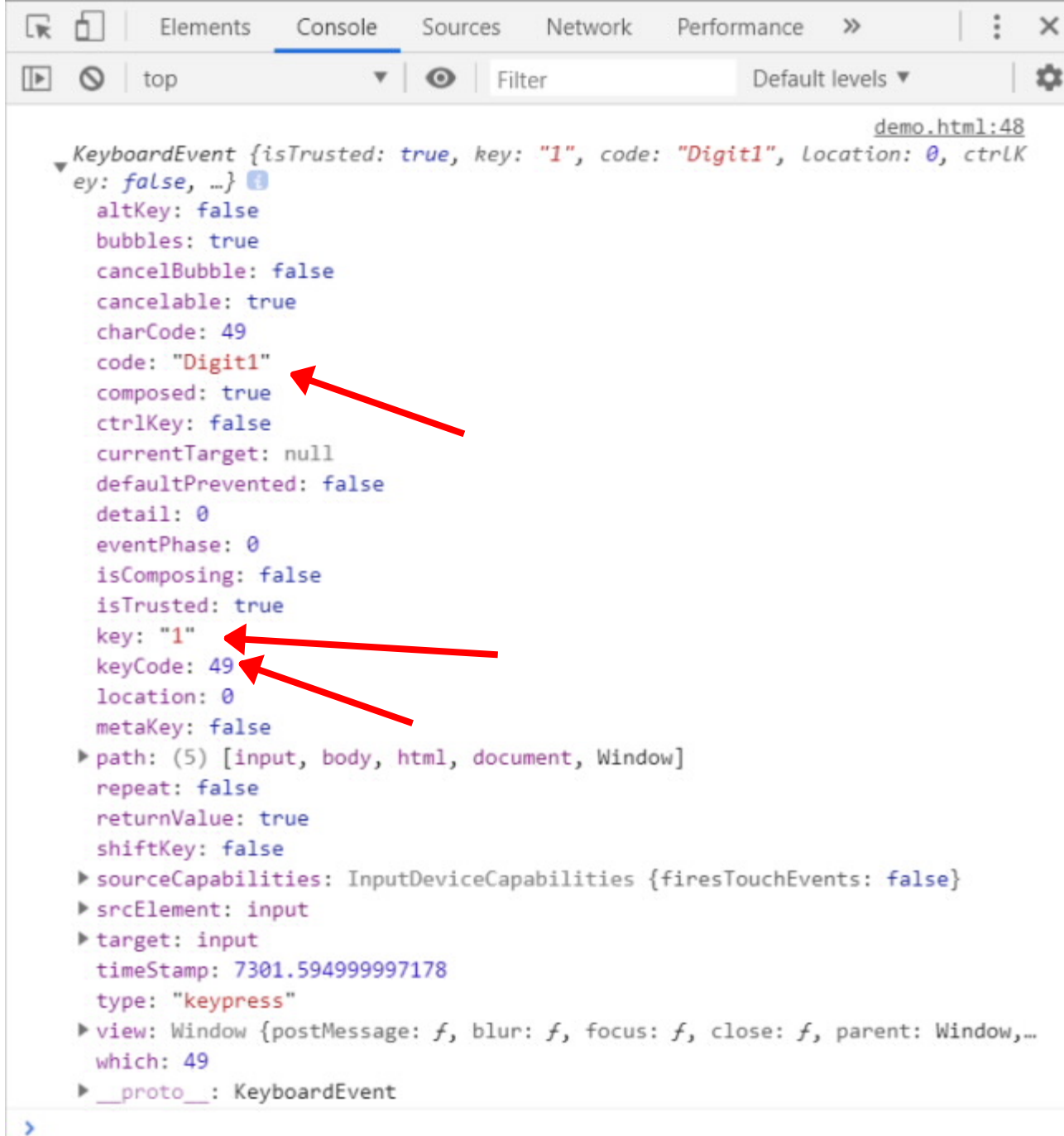
键码

keyCode属性: 在发生keydown 和keyup 事件时, event 对象的keyCode 属性中会包含一个代码, 与键盘上一个特定的键对应。对数字字母字符键, keyCode 属性的值与ASCII 码中对应小写字母或数字的编码相同。

因此, 数字键7 的keyCode 值为55, 而字母A 键的keyCode 值为65——与Shift 键的状态无关。

code属性: 按键的英文名称

key属性: 按下的是什么键



```
demo.html:48
KeyboardEvent {isTrusted: true, key: "1", code: "Digit1", location: 0, ctrlKey: false, ...}
  altKey: false
  bubbles: true
  cancelBubble: false
  cancelable: true
  charCode: 49
  code: "Digit1"
  composed: true
  ctrlKey: false
  currentTarget: null
  defaultPrevented: false
  detail: 0
  eventPhase: 0
  isComposing: false
  isTrusted: true
  key: "1"
  keyCode: 49
  location: 0
  metaKey: false
  path: (5) [input, body, html, document, Window]
  repeat: false
  returnValue: true
  shiftKey: false
  sourceCapabilities: InputDeviceCapabilities {firesTouchEvents: false}
  srcElement: input
  target: input
  timeStamp: 7301.5949999997178
  type: "keypress"
  view: Window {postMessage: f, blur: f, focus: f, close: f, parent: Window, ...}
  which: 49
  __proto__: KeyboardEvent
```


键码表

键	键 码	键	键 码
退格 (Backspace)	8	数字小键盘1	97
制表 (Tab)	9	数字小键盘2	98
回车 (Enter)	13	数字小键盘3	99
上档 (Shift)	16	数字小键盘4	100
控制 (Ctrl)	17	数字小键盘5	101
Alt	18	数字小键盘6	102
暂停/中断 (Pause/Break)	19	数字小键盘7	103
大写锁定 (Caps Lock)	20	数字小键盘8	104
退出 (Esc)	27	数字小键盘9	105
上翻页 (Page Up)	33	数字小键盘+	107
下翻页 (Page Down)	34	数字小键盘及大键盘上的-	109
结尾 (End)	35	数字小键盘.	110
开头 (Home)	36	数字小键盘 /	111
左箭头 (Left Arrow)	37	F1	112
上箭头 (Up Arrow)	38	F2	113
右箭头 (Right Arrow)	39	F3	114
下箭头 (Down Arrow)	40	F4	115
插入 (Ins)	45	F5	116
删除 (Del)	46	F6	117
左Windows键	91	F7	118
右Windows键	92	F8	119
上下文菜单键	93	F9	120
数字小键盘0	96	F10	121

键	键 码	键	键 码
F11	122	正斜杠	191
F12	123	沉音符 (`)	192
数字锁 (Num Lock)	144	等于	61
滚动锁 (Scroll Lock)	145	左方括号	219
分号(IE/Safari/Chrome中)	186	反斜杠 (\)	220
分号 (Opera/FF中)	59	右方括号	221
小于	188	单引号	222
大于	190		

键码表

键码	字符	名称
65	A	大写字母A
66	B	大写字母B
67	C	大写字母C
68	D	大写字母D
69	E	大写字母E
70	F	大写字母F
71	G	大写字母G
72	H	大写字母H
73	I	大写字母I
74	J	大写字母J
75	K	大写字母K
76	L	大写字母L
77	M	大写字母M
78	N	大写字母N
79	O	大写字母O
80	P	大写字母P
81	Q	大写字母Q
82	R	大写字母R
83	S	大写字母S
84	T	大写字母T
85	U	大写字母U
86	V	大写字母V
87	W	大写字母W
88	X	大写字母X
89	Y	大写字母Y
90	Z	大写字母Z

键码	字符	名称
97	a	小写字母a
98	b	小写字母b
99	c	小写字母c
100	d	小写字母d
101	e	小写字母e
102	f	小写字母f
103	g	小写字母g
104	h	小写字母h
105	i	小写字母i
106	j	小写字母j
107	k	小写字母k
108	l	小写字母l
109	m	小写字母m
110	n	小写字母n
111	o	小写字母o
112	p	小写字母p
113	q	小写字母q
114	r	小写字母r
115	s	小写字母s
116	t	小写字母t
117	u	小写字母u
118	v	小写字母v
119	w	小写字母w
120	x	小写字母x
121	y	小写字母y
122	z	小写字母z