# 事件委托

事件委托也称之为事件代理（Event Delegation）。是JavaScript中绑定事件的常用技巧。顾名思义，“事件代理”即是把原本需要绑定在子元素的响应事件委托给父元素，让父元素担当事件监听的职务。事件代理的原理是DOM元素的事件冒泡。

有些异步生成的dom元素，给其添加事件，一定要使用事件委托方式

比如：弹窗

直接写绑定事件，此时代码执行时，dom元素还未生成，元素上无法绑定对应的事件

jquery添加事件委托

$(“body”).on(“submit”, “.add-form”, function(){})

# BFC

BFC 即块级格式上下文，根据盒模型可知，每个元素都被定义为一个矩形盒子，然而盒子的布局会受到**尺寸，定位，盒子的子元素或兄弟元素，视口的尺寸**等因素决定，所以这里有一个浏览器计算的过程，计算的规则就是由一个叫做**视觉格式化模型**的东西所定义的，BFC 就是来自这个概念，它是 CSS 视觉渲染的一部分，**用于决定块级盒的布局及浮动相互影响范围的一个区域**。

BFC 具有一些特性：

1. 块级元素会在垂直方向一个接一个的排列，和文档流的排列方式一致。
2. 在 BFC 中上下相邻的两个容器的 margin  会重叠，创建新的 BFC 可以避免外边距重叠。
3. 计算 BFC 的高度时，需要计算浮动元素的高度。
4. BFC 区域不会与浮动的容器发生重叠。
5. BFC 是独立的容器，容器内部元素不会影响外部元素。
6. 每个元素的左 margin  值和容器的左 border  相接触。

利用这些特性，我们可以解决以下问题：

* 利用 4  和 6 ，我们可以实现三栏（或两栏）自适应布局。
* 利用 2 ，我们可以避免 margin  重叠问题。
* 利用 3 ，我们可以避免高度塌陷。

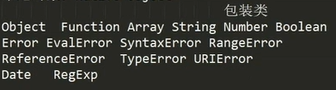
创建 BFC 的方式：

* 绝对定位元素（position 为 absolute 或 fixed ）。
* 行内块元素，即 display 为 inline-block 。
* overflow 的值不为 visible 。

# JS中的三种对象

## 本地对象 – native object

就是需要new出来的对象。如Object、Function、Array、String、Boolean、Number、Date、RegExp、Error、EvalError、RangeError、ReferenceError、SyntaxError、TypeError、URIError等



## 内置对象 – build-in object

内置对象，就是不用创建，可以直接使用的对象，也就是我们常说的不用new的，可以直接使用的。如arguments、this、event等。

Global 这是一个统称,js中并没有该关键字

在global下的方法：parseInt() isNaN() Number() decode URI encodeURI …

在global下的属性：undefined NaN infinity

本地对象和内置对象都是javascript提供的

## 宿主对象 – host object

执行js脚本环境所提供的对象，具有兼容性问题

浏览器对象DOM和BOM，DOM其实是在BOM下面的，只不过DOM有W3C规范，所以单独被抽取出来

## 小知识：

HTML是由XML过渡来的。XML的标签可以自定义

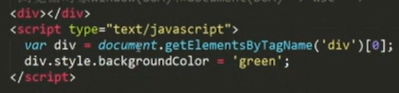
<person></person> 表示标签

<person>。。。</person> 表示元素(标签加标签里面的内容)

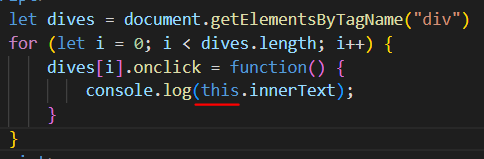


dom中不是直接操作css样式(jquery也是一样)，下方操作css样式被修改，是因为在dom元素中添加了style属性,style优先级更高，覆盖了css中的样式。通过行内样式间接的修改样式





循环类数组的时候，可以使用this指代dives[i]

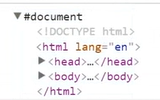


<a href=”javascript:void(0)”></a> void(0) 就是一个return 0的意思 javascript:; 这个是一个伪协议

window.open(“https://xxxxx”) 跳转页面

# DOM

document 包括了页面中所有的dom元素，它是一个对象



## querySelector和getElementsBy系列的区别

* querySelector 属于 W3C 中的 Selectors API 规范接收的参数是一个css选择符。
* getElementsBy 系列则属于 W3C 的 DOM 规范 。
* querySelector返回的都是NodeList对象（即文档节点的集合）
* getElementsBy系列返回的是HTMLCollection（HTML元素的集合）
* 注意这两者返回的都是一个集合，如果要使用其中的元素，都要通过索引值取出；
* getElement获取的是动态集合，
* querySelector获取的是静态集合
* 说到底这两个特性有什么区别呢: 其实就是动态选出的元素会随着文档的改变而改变，而静态的不会



## document.querySelector和querySelectorAll

document.querySelector() 没有获取到元素返回null

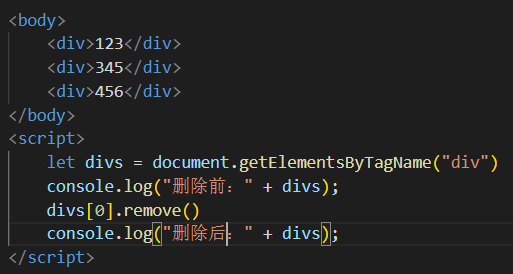
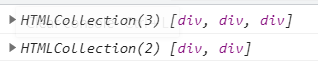
document.querySelectorAll() 没有获取到元素返回 Nodelist []

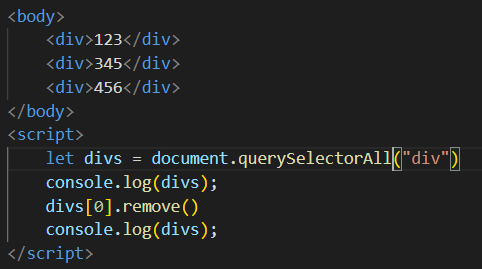
HTML5 引入的webApi，但是在很早之前就有了，只是在HTML5中正式引入了标准

兼容性：IE7都兼容它，6就不兼容了

缺点：

1. 性能不好，查询比get系列的方法慢
2. 不实时

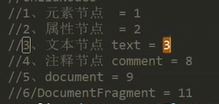
 

## 节点与元素的区别

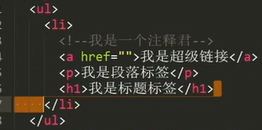
节点不是元素，节点包含了元素

元素在节点这块呢，叫做元素节点。

### 节点类型：

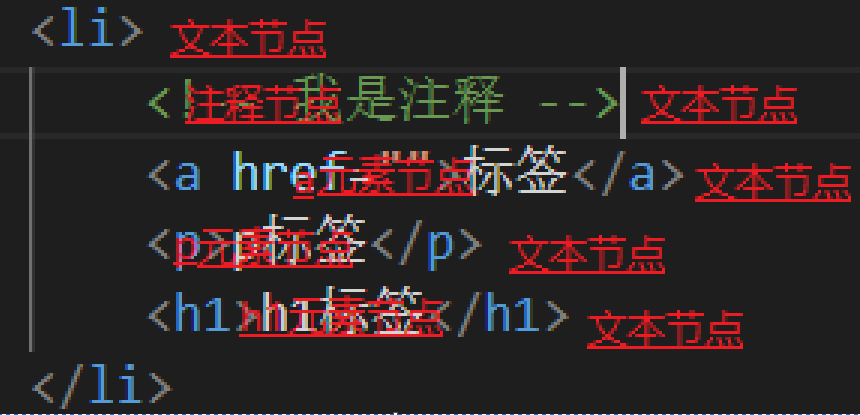
 后面的数字是对应的节点号

### 题目：childNodes打印出什么？





答案：

为什么？：

## 节点和元素的区别再次记忆

let p = document.getElementsBtTagName(“p”)

p.parentElement….一直向上找元素，最终找到html元素，html的parentElement为null

document是一个节点，不是元素，所以为null

parentNode最顶层是document

parentElement最顶层是HTML

## nodeName 节点属性-输出节点的名称 只读

let div = document.getElementsByTagName(“div”)[0]

console.log(div.nodeName) // DIV 元素节点的nodeName是标签名大写 转换成小写可以使用toLowerCase

## nodeValue 节点属性-输出节点的值 可读写

元素节点-1 没有该属性 使用该属性，返回的是一个null 可以使用innerHtml 或 innerText

属性节点-2、文本节点-3、注释节点-8 有该属性

let div = document.getElementsByTagName(“div”)[0]

console.log(div.childNodes[0].nodeValue)

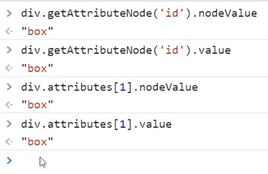
div.childNodes[0].nodeValue = “value被修改了”

## nodeType 节点类型

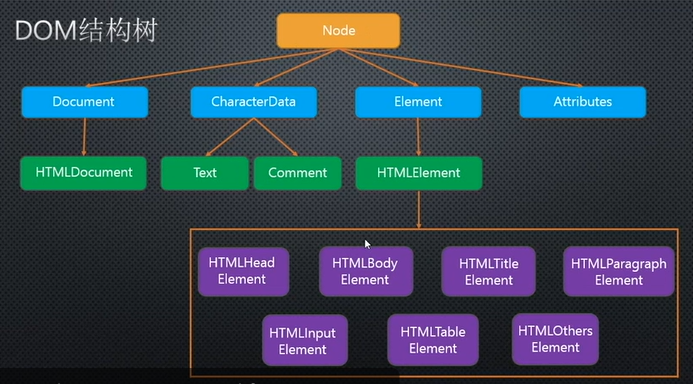
let div = document.getElementsByTagName(“div”)[0]

console.log(div.nodeType)

## 获取属性节点的方法



## DOM结构树

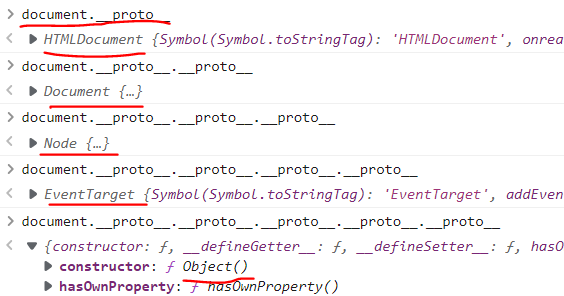


### 了解Document下的HTMLDocument和XMLDocument

[3\_节点属性\_方法\_封装方法\_DOM结构操作深入-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/14175208848295278)

1小时10-17分

### document继承的原型链



原型的顶点都是Object.prototype

### 通过Object.prototype.toString.call()查看元素节点或其他的节点，数据类型



### Element下的继承关系

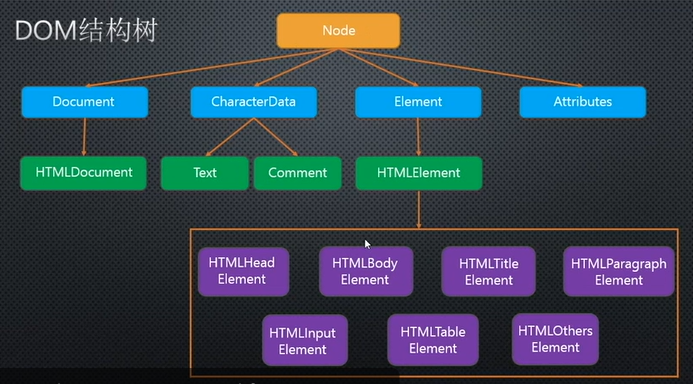
### getElementById() 和getElementsByName()

通过id获取节点，只能用document对象调用。IE6～8也可以使用，但在IE8以下存在一个bug：若name属性和id属性的内容相同，则浏览器会自动取先出现的属性，所以有可能会取到name属性，而不是id属性，该bug只在IE8版本以下的浏览器出现。

这两个方法只在Document.prototype上有。所以只有继承了它的对象才能调用该方法，

看下面的图，只有HTMLDocument下的节点才有，

HTMLElement及其他的都没有



通过document.getElement…的方式仅仅是获取到对应的元素,要想转换成Dom对象，需要通过对应的构造函数去构造

所以document.getElementsByTagName(“div”)[0]得到的div数据类型是HTMLDivElement

它是由HTMLDivElement构造而来的 所以沿着原型链找，并没有继承Document中才有的getElementById()方法

### getElementsByTagName() getElementsByClassName() querySelector() querySelectorAll()

上面四个方法在Document中有，在Element中也有

### document获取body、head、title

在HTMLDocument.prototype上有两个属性body和head

可以直接通过该属性获取对应的元素节点

let body = document.body

let head = document.head

但是

let title = document.title 获取到的不是元素节点而是文本

title = “标题文本”

### document获取html documentElement

属性来自Document.prototype



### element.contains(element)

这个方法是用来计算是否包含某个元素节点的

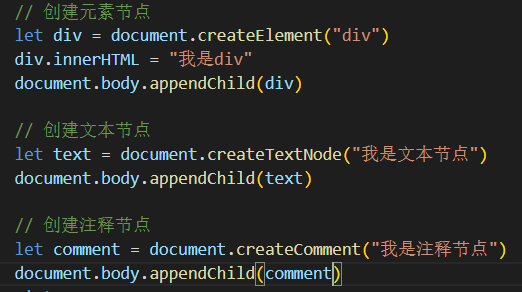


## 操作节点

### document.createElement()

属性来自Document.prototype

### 创建节点 createElement()





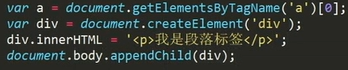
### 添加节点 appendChild()

方法来自Node.prototype

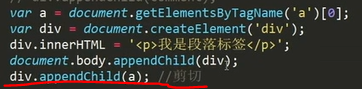
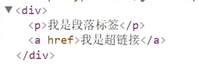
appendChild(节点 也叫DOM元素) 里面放的是节点(也叫DOM元素)。不是一个字符串”<div>…</div>”会报错的

返回值是**添加的节点**

appendChild最大的两个功能：动态的增加节点，动态的剪切节点



### 插入节点 c.insertBefore(a,b)

在父级c节点下的 b节点之前插入到a节点

方法处于Node.prototype

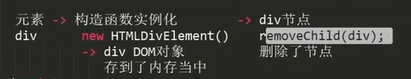
返回值 **插入的节点**

### 删除节点 父节点.removeChild(子节点)

方法处于Node.prototype

**返回被’删除’的节点**，节点实际没有被删除，只是被剪切了，存在于内存中

[4\_节点创建删除\_元素属性设置获取\_节点属性-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/14175268977837422) 34分钟有解说



通过get…的方法获取到对应元素-🡪内部通过相应的构造函数进行实例化成元素节点，将元素节点保存在内存中，removeChild()删除的只是DOM树上的节点，并没有删除内存中的DOM对象

节点.remove() 这才是在内存中将节点给销毁了

### 替换节点 父节点.replaceChild(新节点,旧节点)

返回值 **替换的节点**

### inner HTML 和 innerText

#### innerHTML 可读可写

div.innerHTML 返回的是一个字符串

div.innerHTML = “123” 覆盖

div.innerHTML += “456” 追加

div.innerHTML 获取内容

可以识别html标签

保留空格和换行

#### innerText 可读可写

无法识别html标签

div.innerText = “<a href=”#”>链接</a>” 会直接输出字符串到页面中

获取内容会去除换行和空格

## 获取select下的option



## 操作节点属性

### getAttribute()

返回值： 指定属性值

div.getAttribute(“属性名”)

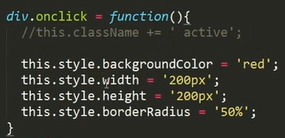
### setAttribute()

没有返回值

div.setAttribute(“属性名”, “值”)

## 修改或添加元素属性值注意点

不能直接用下面的方式书写，因为.语法是要消耗资源的



给一个类，里面封装好相应的css样式，在dom操作中附加对应的类名





## classList

### 方法

add( String[,String])：添加指定的类值。如果这些类已经存在于元素的属性中，那么他们将被忽略

remove(String[,String])：删除指定的类值

item(Number)：按集合中的索引返回类值

toggle(String[,force])：

当只有一个参数的时，切换class value ，即如果类存在，删除它，并返回false，如果类不存在，则添加它并返回true。

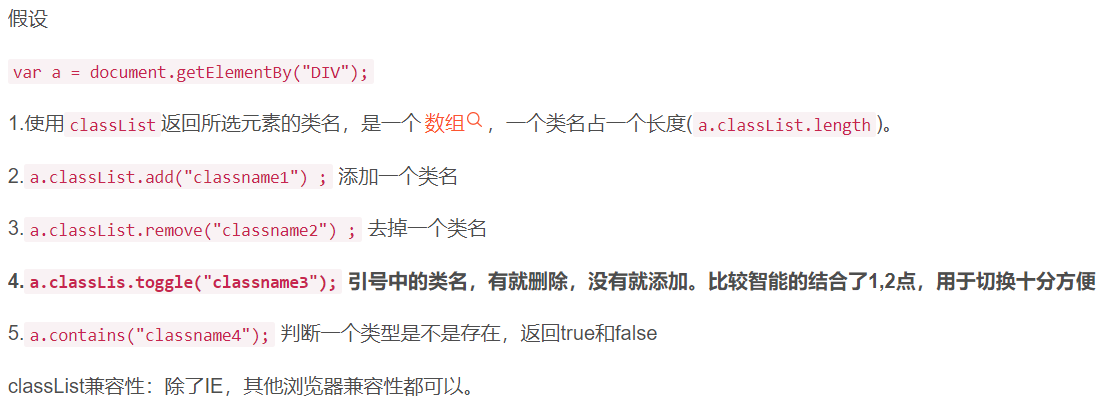
当存在第二个参数时，如果第二个参数的计算结果为true，则添加指定的类值，如果计算结果为false，则删除它

contains(string)：检查元素的类属性中是否存在指定的类值

### 属性

length：返回类列表中类的数量，该属性是只读的

### 示例



## DOM无法直接操作CSS样式表

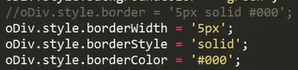
直接在元素上添加属性style来进行样式的覆盖(行内样式权重更高)

也无法读取直接读取css中的样式

 输出空

但是在设置了son.style.width = “100px”之后可以得到,访问的是行内样式上的高

### DOM间接修改样式时，复合值要拆开写



### 使用保留字，名称前要加css



## 使用 元素节点.style 可以得到该节点所有可以设置的样式的集合



## 使用window.getComputedStyle获取值，获取值比较准 比如获取元素宽高

window.getComputedStyle(元素节点, null).属性名

**返回值带px单位，并且是字符串**

IE8及以下不支持,只支持currentStyle

使用该方法会强制触发回流

该方法第二个参数null作用：

用来填伪元素，获取伪元素属性的



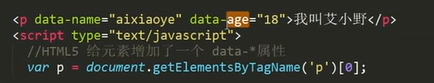
另外一种操作伪元素的方法



### getComputedStyle得到的属性时只读的，不可修改；而element.style得到的属性可以读也可以写

## 自定义属性 html5新增

在dom元素上写data-\* \*为任意名称





使用 元素节点.dataset.属性名称 调用 属性名只写data- 后面的名称.***该方法兼容性没有getAttribute()好***

也可以使用

元素节点.getAttribute(“data-name”)获取 属性名写全

元素节点.setAttribute(“data-name”, ” ”)修改

## 文档片段 createDocumentFragment()

let oFrag = document.createDocumentFragment()

可以将需要重复添加的元素，添加到文档碎片中，然后再将文档碎片添加到最终需要添加的元素节点上

减少回流的次数



## Date()构造函数

### 常用方法：

new Date().getDay() 获取 0-6的数字 0是星期天

new Date().getMonth() 获取0-11的数字 0是一月

new Date().getFullYear() 获取当前年份

getHours()

getMinutes()

getSeconds()

三种手动设置时间方式 还有其他方法



年 月 日 时 分 秒

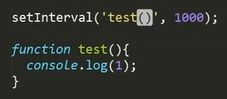


### dateTime()

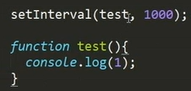
返回 1970 年 1 月 1 日至今的毫秒数

## 定时器

参数可以传入字符串，需要添加括号

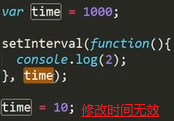


传入函数，不添加执行符号()，定时器内部会帮你完成



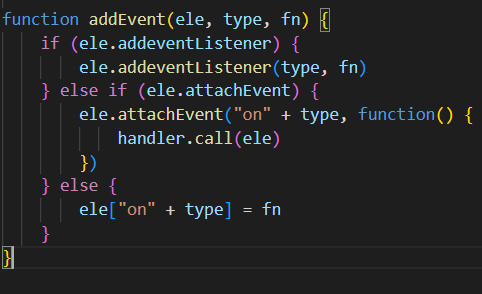
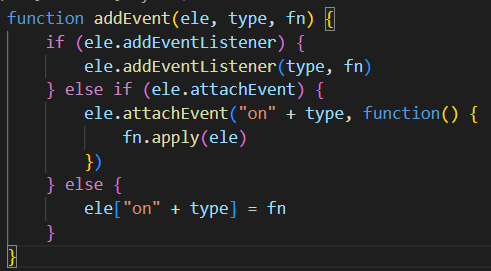
### 注意点

在定时器运行之后，再修改time是不起作用的



## DOM事件的兼容性写法封装

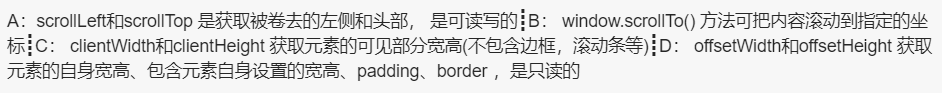


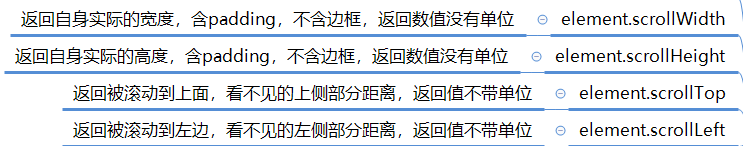
 

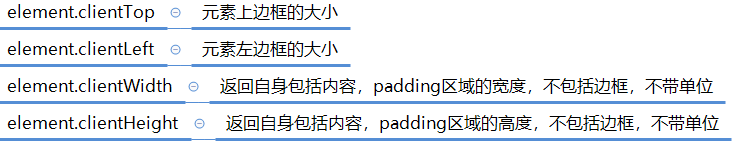
## 下面的如果看不懂可以结合xmind观看

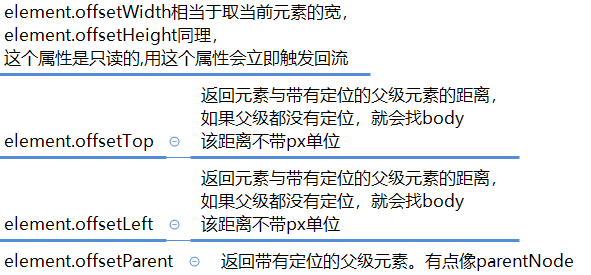
## 三大获取元素属性的方法对比

## scrollLeft和scrollTop是可读可写的，其他的都是只读的



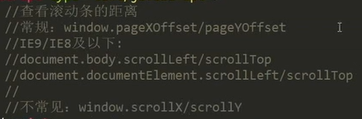






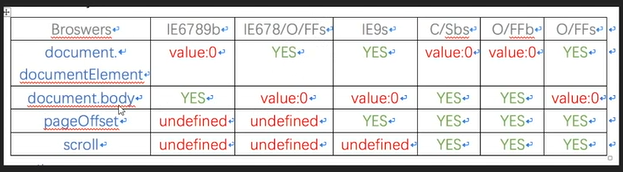
offsetWidth/Height这个属性包含边框

## 查看滚动条距离

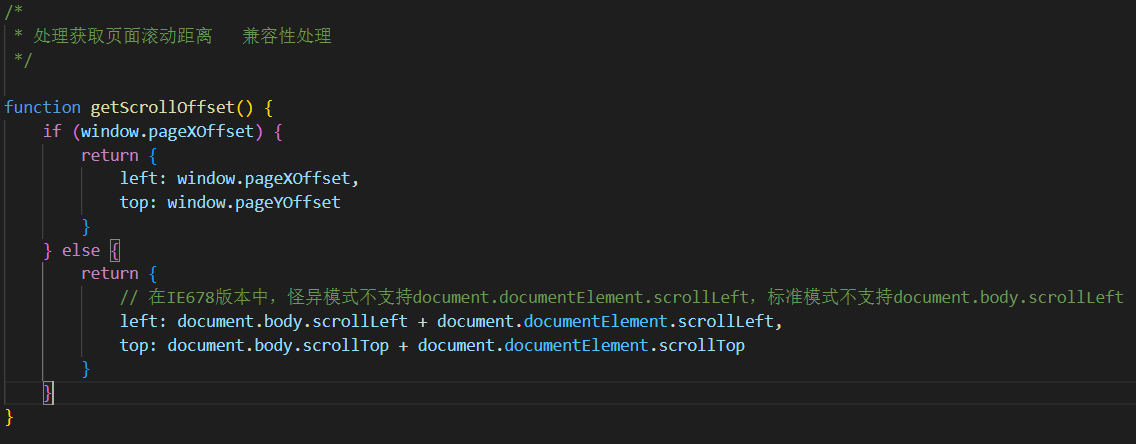


chrome不支持document.documentElement.scrollLeft/scrollTop

介绍 上图的东西 [7\_滚动距离与高度、兼容模式、可视尺寸-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/14175281862739310) 4



### 兼容性写法封装



### window.scroll(x轴距离,y轴距离) window.scrollTo(x轴距离,y轴距离) 两个是一样的用法,没有兼容性问题

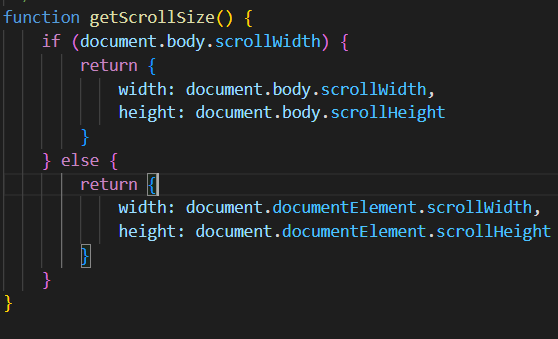
### 把内容滚动指定的像素

window.scrollBy(xnum,ynum)

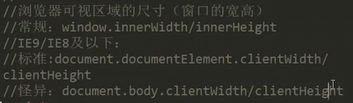
### 自身的高度 <= 页面视口高度 + 滚出屏幕的高度



## 返回自身实际的宽/高，含padding，不含边框，返回数值没有单位



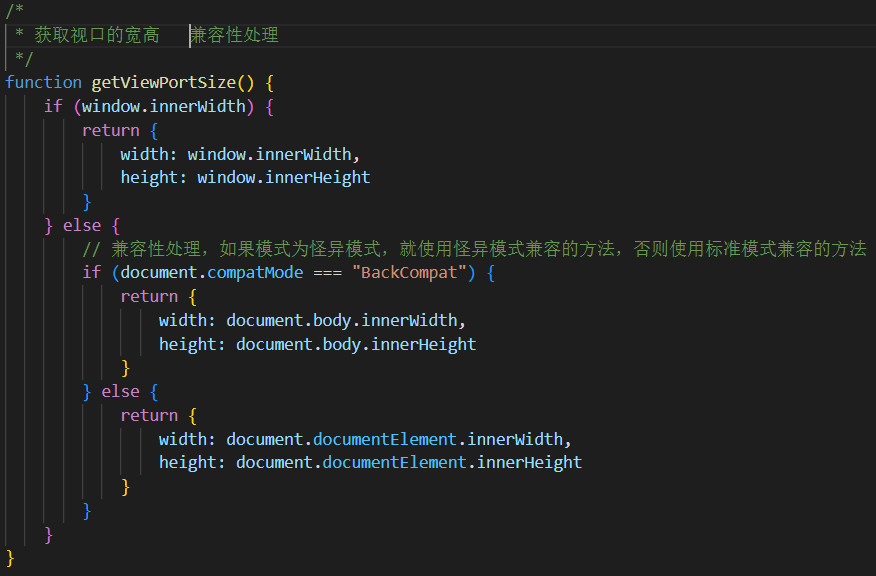
## 浏览器可视区域（窗口的宽高）



使用怪异模式时，如果页面有滚动条，页面视口大小不会包括滚动条

使用标准模式或常规方法，则会包括

兼容性写法封装：



## element.offsetLeft/offsetTop 和 element.offsetParent

返回元素与带有定位的父级元素的距离，如果父级都没有定位，就会找body 【该距离不带px单位】

ele.offsetWidth/offsetHeight 返回元素自身的宽高(content+padding+border)

原理：到渲染引擎里面去访问，元素到底是多少宽度，是从底层访问到的

该属性的弊端：[8\_读写样式属性、操作伪元素、元素运动初探-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/14175286157706606?_wv=2147487745)

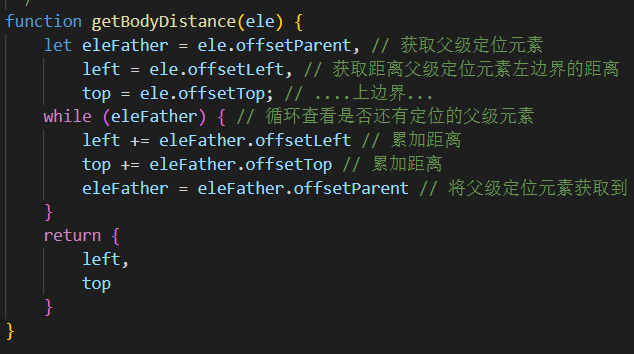
27分钟

**当你动态的改变元素的宽高时，会将padding值也一起加进去计算**：

比如元素宽高各100px padding 10px 每点击一次元素宽度加10 但是因为会将padding一起计算，所以每次点击增加30px的宽度

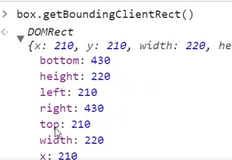
element.offsetParent：返回带有定位的父级元素。有点像parentNode

### 小案例：寻找当前元素距离最外层的边界的距离



## getBoundingClientRect()

返回元素的大小及其相对于视口的位置



在IE浏览器中，不显示宽高信息 可以使用left和right值相减，bottom和top值相减

添加margin ,方向的数值会变化

添加padding ,元素大小会变化

不是动态的



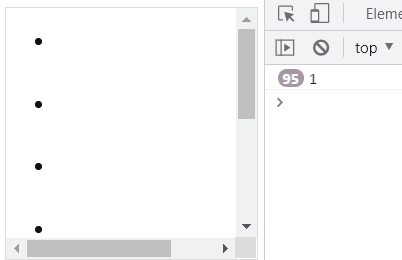


修改元素宽度之后，重新调用info，获取到的还是旧的数据

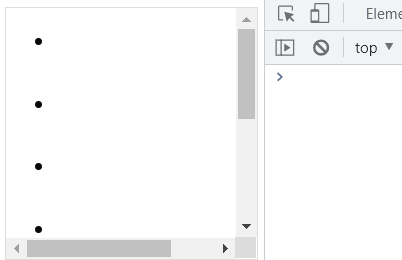
# DOM事件

## scroll事件和wheel事件的区别

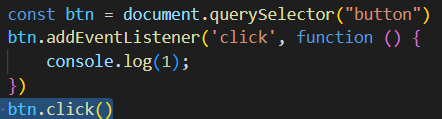
scroll在滚动条滚动时会触发(通过拖动滚动条或者鼠标滚轮滚动)，滚动条到底部时，不会再触发



wheel只会在鼠标滚轮滚动时触发，并且滚动条到底部时，依然会触发

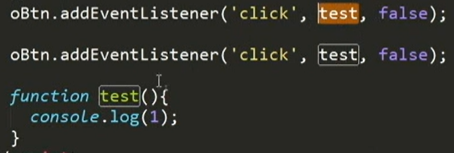


## elem.click() 直接触发元素的点击事件



绑定事件并不是绑定事件本身，而是绑定事件的处理函数。

## addEventListener() 事件绑定函数的注意点 attachEvent()没有这个问题



输出1

虽然可以给同一个元素的同一个事件注册多个监听器，但是图里面的回调函数都是test,这是相同的监听函数，所以只会输出1个1

## 事件绑定/事件监听

btn.onclick = function(){}

eventTarget.addEventListener(事件名”click”, fn[, useCapture]) // useCapture默认false

eventTarget.attachEvent(事件”onclick”, callback)

## 解除事件绑定/事件监听

btn.onclick = null

eventTarget.removeEventListener(事件名”click”,fn[, useCapture])

eventTarget.detachEvent(事件名”onclick”, callback)

## 事件捕获和事件冒泡

捕获类型事件只能使用addEventListener注册，传统的注册方法只能是事件冒泡

### 什么是事件冒泡和事件捕获

**事件流**：

页面触发一个事件时，会按照一定的顺序来响应事件，事件的响应过程为事件流

事件流可分为冒泡型事件流、捕获型事件流。

（1）事件冒泡：微软公司提出的，事件由子元素传递到父元素的过程叫做冒泡（false）。

查找事件（事件响应）的顺序：

文本节点–>元素节点—>body—>html—>document（例如点击事件）<向上响应>

（2）捕获事件：网景公司提出的，事件由父元素传递到子元素的过程叫做事件捕获。（ture）

查找事件（事件响应）的顺序：

document–>html–>body–>元素节点–>文本节点 <向下响应>

### 应用

在使用"**事件监听**"的方式绑定事件时，可以设置事件的响应方式。

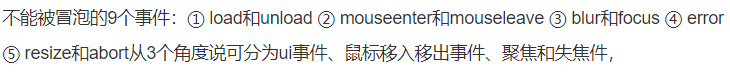
DOM对象.addEventListener(事件，事件处理程序，事件冒泡方式)

**事件冒泡方式：**

（1）默认为false，表示**冒泡阶段**完成事件处理

（2）true为**捕获阶段**完成事件处理

### 不能冒泡的事件



### 事件捕获和事件冒泡顺序问题

[事件处理函数、冒泡捕获、阻止冒泡默认事件 (qq.com)](https://ke.qq.com/webcourse/386920/104499667#taid=13292232291772264&vid=5285890787933124305) 1小时29分

[冒泡捕获流、事件与事件源对象、事件委托 (qq.com)](https://ke.qq.com/webcourse/386920/104499667#taid=13292236586739560&vid=5285890788164835845) 11分17秒

### 阻止事件冒泡的方式

#### （1）事件委托：将元素的绑定事件写起其父元素上，防止事件冒泡

#### （2）event.stopPropagation()：可以阻止事件冒泡，阻止父级元素的绑定事件

ele.addEventListener(“click”, function(e){

var e = e || window.event

// 封装阻止冒泡的兼容性写法

if(e.stopPropagation){

e.stopPropagation()

}else{  
 e.cancelBubble = true // IE上阻止冒泡的方法

}

})

## 取消默认事件

### 取消鼠标右键菜单

document.oncontextmenu = function(e) {

        // return false // 兼容性较好

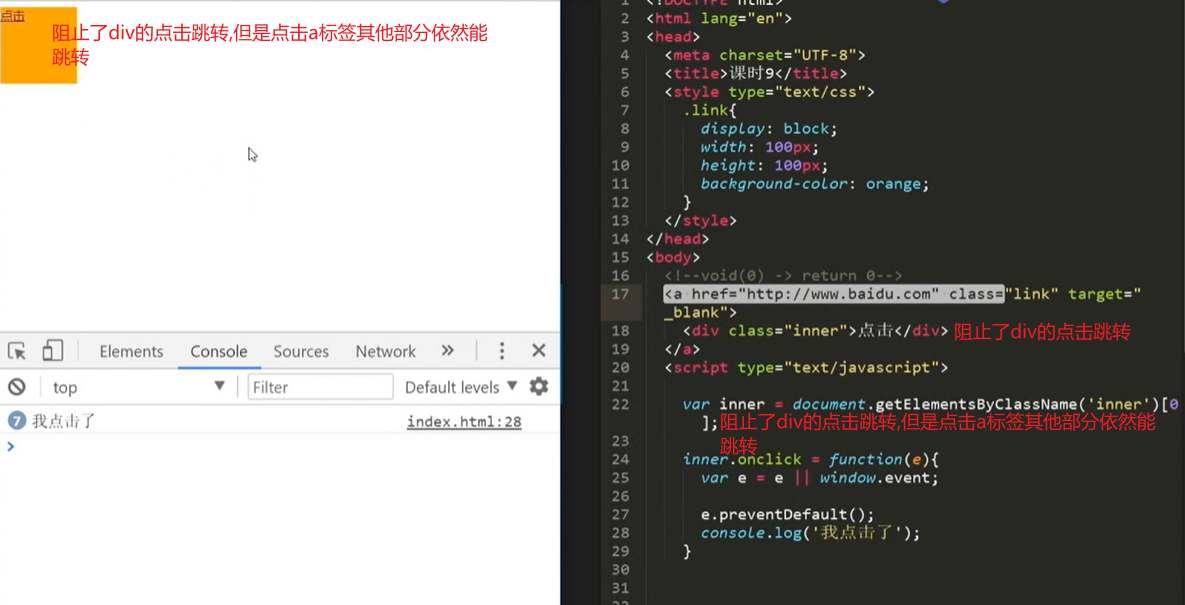
        // e.preventDefault() // W3C标准 IE9及以下不支持

        e.returnValue = false // IE9以下不兼容

    }

### 取消a标签默认事件 比较有用的小技巧

给a标签里面的其他元素取消默认事件，这样点击里面单独的元素不会跳转，点击外面的元素依然跳转



### 阻止表单提交的默认事件

submit.onclick = function(e){

var e = e || window.e

e.preventDefault()

}

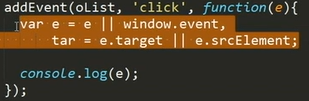
## 事件对象 事件委托

获取事件委托下触发事件的源对象在其节点列表中的索引

const index = Array.prototype.indexOf.call(源对象所处的列表集合, 源对象)

或者 const index = [].indexOf.call(源对象所处的列表集合, 源对象)

addEvent是封装的事件绑定兼容性处理的函数



IE9以下用的是srcElement ，其他浏览器(谷歌两个都有)用的是target(这两个是一个东西), e 和window.e也是一样，用来处理兼容性写法

e.target是你点中的对象

e.target.tagName是你选中对象的标签名 标签名是大写的

### 为什么使用：

对于一些列表项很多的，不能给遍历每一个项添加事件，这样性能很差.

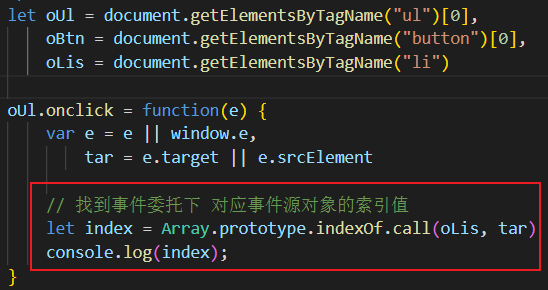
并且对于后面添加进去的子元素，是不会有对应绑定的事件。因为循环时已经绑定完了,后面的自然不会绑定,此时可以通过事件委托的方式处理。

给父级元素添加事件，通过e.target或e.srcElement找到对应的点击对象

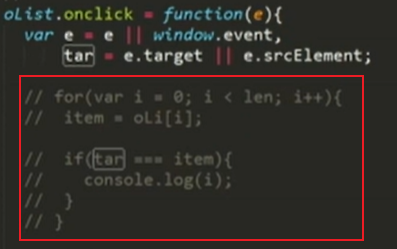
### 原理：

点击子元素，通过冒泡，触发父元素上的事件，然后通过事件对象e.target找到触发事件的源对象

### 找到事件委托下的触发事件的源对象的索引值

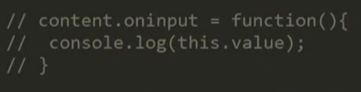


下面这种方法也可以，但是不合适,性能不好

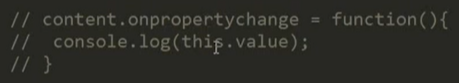


## oninput、 onchange事件

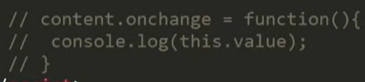
oninput是在输入的时候触发回调函数 但是在IE9以下版本无法使用



结合下面IE使用的onpropertychange事件写兼容性



onchange是在失去焦点时触发回调函数，并且是在数据发生改变时才会触发

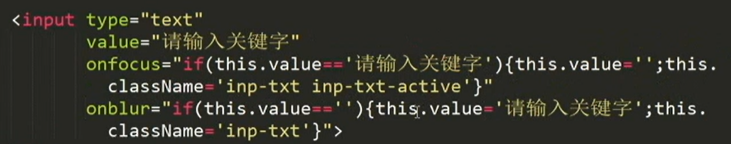


## onfocus onblur 事件

placeholder不方便修改样式，且对于浏览器而言修改的样式不太一样。有兼容性问题。

所以企业里面往往不会使用placeholder

使用onfocus和onblur书写 直接写在input输入框里面



# 鼠标行为

## 鼠标坐标

client X/Y 鼠标位置相对于当前可视区域的距离(不包括滚动条，滚动上去的距离)

screenX/Y 鼠标位置相对于屏幕的坐标

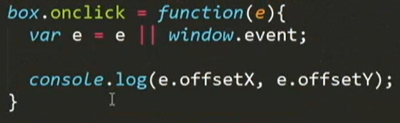
pageX/Y 鼠标位置相对于当前文档的坐标(包括滚动条，滚动上去的距离) ie9以下不支持

offsetX/Y 鼠标位置相对于块元素的坐标 包含边框 (safari不包括边框)

layerX/Y 同pageX/Y相同 IE11以下同clientX/Y

x/y 鼠标位置相对于当前可视区域的距离(不包括滚动条，滚动上去的距离)

火狐旧版的不支持，新版的不知道



**pageX/Y可以获取包括滚动出可视区域的坐标信息，但是因为不兼容IE9，所以需要自己封装一个**



## 鼠标点击行为获取

需要在IE10以上，以下的版本获取的数字都不同

元素.onmousedown = function(e){

e.button = 0/1/2 // 左中右鼠标点击事件

}

在click事件下，点击鼠标中和右键是没有效果的，因为鼠标中和右不会触发点击事件

# 键盘事件

## keydown keypress keyup

一般来说都是使用document来触发键盘事件 document.onkeydown = …..

### keydown和keypress的区别

**keydown事件**回调函数中的e里面的charCode值不管按什么按键都是0

**keypress事件**回调函数中的e里面charCode有对应的值

charCode就是ASCII码值

但是在【keydown事件】上有一个【keyCode属性】，对应的是按键的顺序码。

通过【charCode】 无法获取到像是f1 enter这些按键对应的ASCII值。

此时就可以通过keyCode来获取.

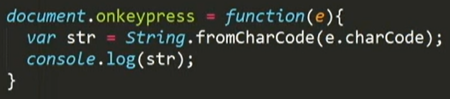
keyCode可以获取所有的按键

charCode只能获取ASCII上有的按键

需要区分大小写时可以使用charCode

String有个函数 String.fromCharCode(e.charCode) 可以将charCode码值转换为按下的按键名

该按键名是区分大小写的



# 知识小拓展

### 1、

ie8以下 getElementById(“”)里面的参数是不区分大小写的



### 2、

id不要随便使用，一般是对接后端的数据配合使用

### 3、childElementCount 等同于 children.length

### 4、

### 5、节点有无的判断方法 haschildNodes()

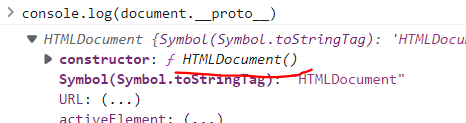
 

### 6、document的方法说明

document可以使用.getElementByTagName等等这些方法。

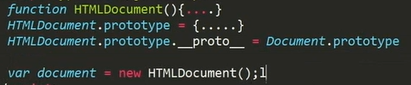
他们并不是在document上的方法，而是从Document构造函数的原型属性上继承而来的



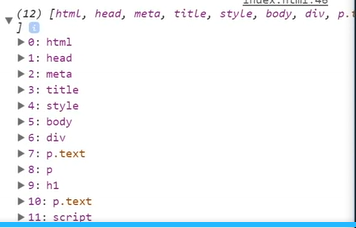
document.\_\_proto\_\_ = HTMLDocument.prototype

HTMLDocument.prototype.\_\_proto\_\_ = Document.prototype



### 7、通配符的小用处





### 8、textContent 了解

老版本火狐不支持innerText 可以使用textContent代替

### 9、o元素名

let oUl = document.getElementByTagName(“ul”)

在企业开发中一般喜欢命名前面加个o 代表object 这是一个元素节点对象，用以区分ul元素

### 10、查看怪异模式和标准模式

标准模式

怪异模式

### 11、inner HTML 和动态创建元素 两种方式效率对比

[js innerHTML和 createElement的性能比较\_头发和技术可不可以都要的博客-CSDN博客](https://blog.csdn.net/qq_57406825/article/details/124219988)

// 平均1300毫秒左右---普通字符串拼接

    for (let i = 0; i < 1000; i++) {

      document.body.innerHTML += "<div style='width:100px;height:10px;background-color:red;margin-top:5px'></div>";

    }

    // 平均5毫秒左右---往数组添加字符串

    const arr = [];

    for (let i = 0; i < 1000; i++) {

      arr.push("<div style='width:100px;height:10px;background-color:red;margin-top:5px'></div>");

    }

    document.body.innerHTML = arr.join("");

    // 平均10毫秒---动态创建元素

    for (let i = 0; i < 1000; i++) {

      const div = document.createElement("div");

      div.style.width = "100px";

      div.style.height = "10px";

      div.style.backgroundColor = "red";

      div.style.marginTop = "10px";

      document.body.appendChild(div);

    }