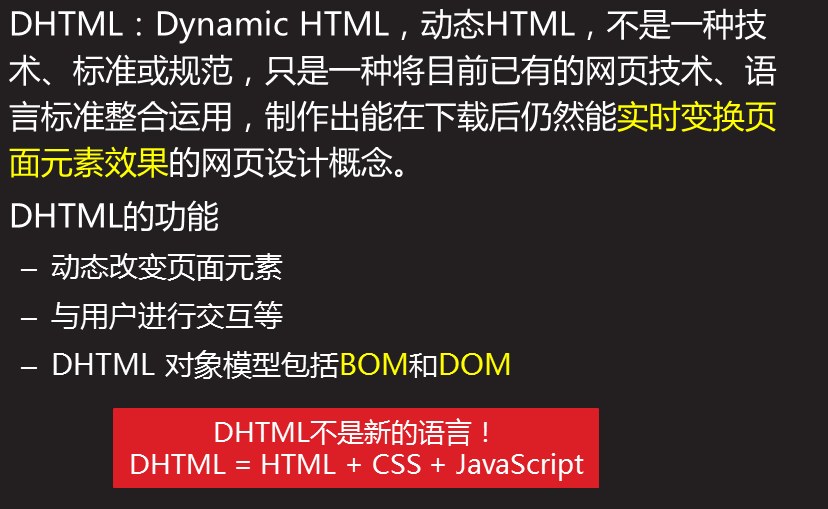
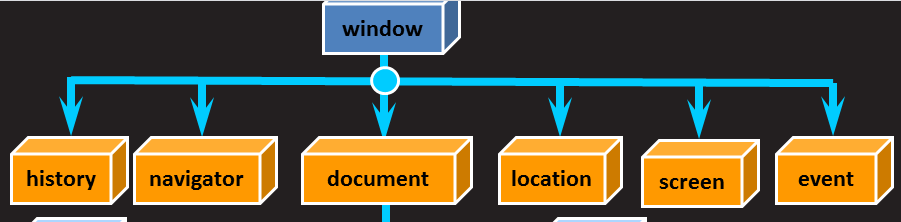
1. DHTML（动态网站）



静态网站（HTML+CSS）：没有动画效果的网站

1. DHTML的对象模型（7个对象）



history 黑丝特瑞

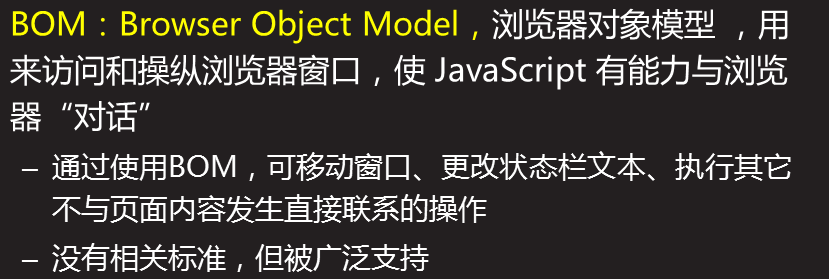
navigator 拿wer给得儿

location 楼k神

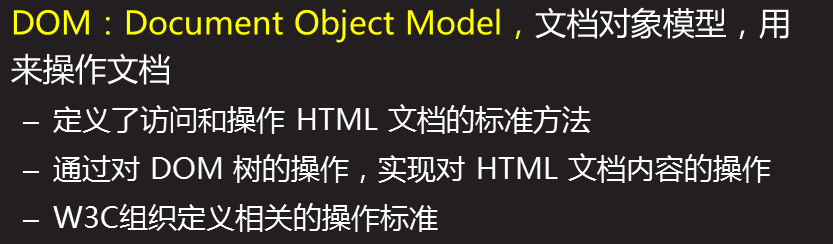
screen 死鬼莹

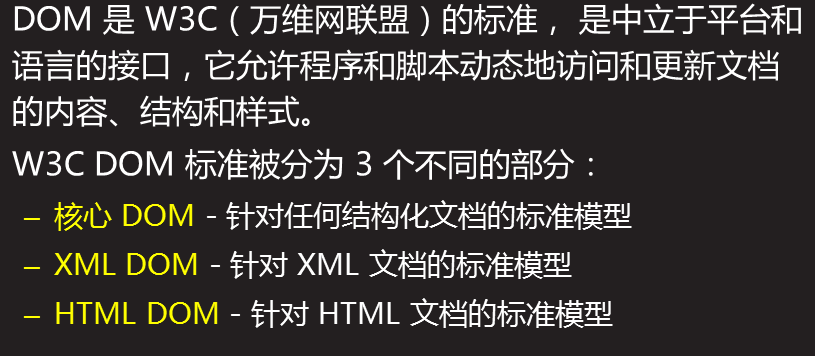
event 一雯特

1. BOM（7个对象/6个对象（除了document对象））



1. DOM:（document对象）

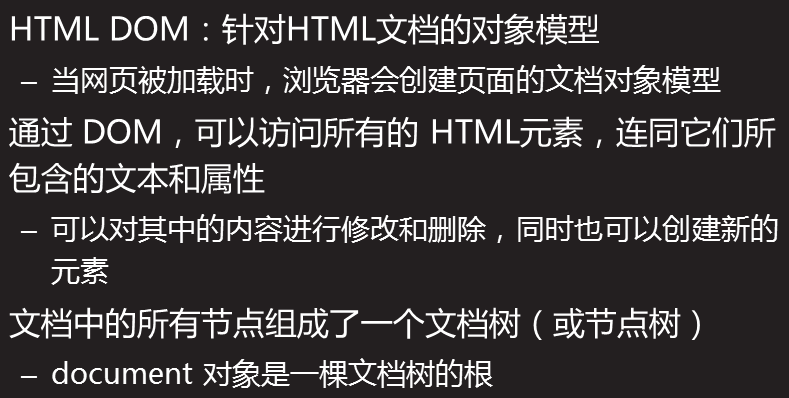




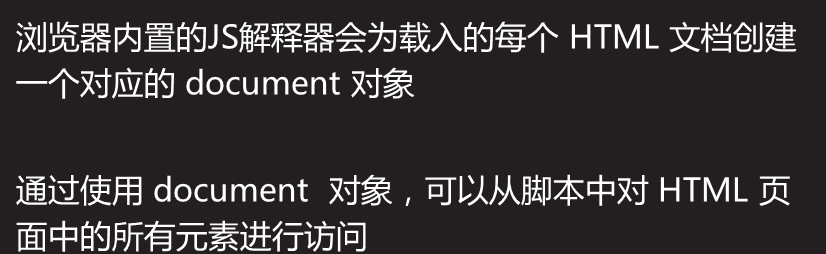
核心 DOM ：没有兼容问题，很繁琐，可以操作XML,HTML。

HTML DOM ：有兼容问题，使用简单，可以操作HTML。

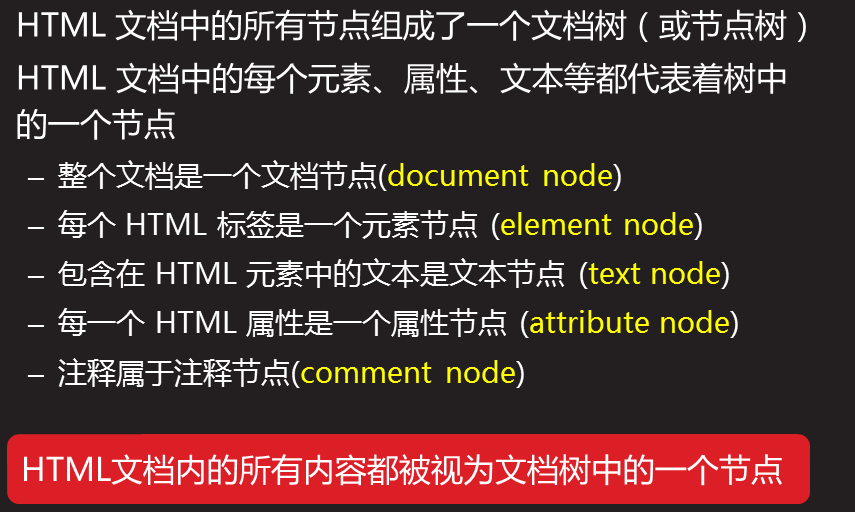
1. HTML DOM（了解）



1. document 对象



1. 节点树（文档树）---》页面上的所有节点组成的树（包含空格(body/head)当做文本节点，找标签，文本，注释...）



console.log(document);//整个文档---》树根

console.log(document.firstChild);//<!DOCTYPE html>

console.log(document.lastChild);//html

console.log(document.lastElementChild);//html

console.log(document.head);//head 简便法

console.log(document.body);//body 简便法

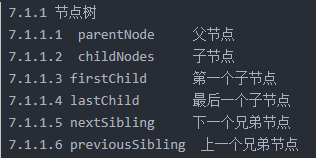
console.log(document.lastChild.firstChild);//head 节点树

console.log(document.lastChild.lastChild);//body 节点树

console.log(document.lastElementChild.firstElementChild);//head 元素树

console.log(document.lastElementChild.lastElementChild);//body 元素树





parentNode 父节点 派温特

childNodes 子节点

firstChild 第一个子节点

lastChild 最后一个子节点 拉斯特

nextSibling 下一个兄弟节点

previousSibling 上一个兄弟节点 普瑞贼二四

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>文档标题</title>

</head>

<body>

<a href="#">我的链接</a>

<h1>我的标题</h1>

<script></script>

</body>

</html>

//节点树---->firstChild

console.log(document.body.firstChild);//#text---》空格

//节点树

console.log(document.body.childNodes);// [text, a, text, h1, text, script]

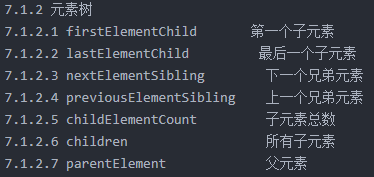
console.log(document.body.childNodes[3]);//h1

console.log(document.body.firstChild.nextSibling.nextSibling.nextSibling);//h1

console.log(document.body.lastChild.previousSibling.previousSibling);//h1

1. 元素树---》页面上的元素节点组成的树（不会包含空格当做文本节点，只找标签）





firstElementChild 第一个子元素

lastElementChild 最后一个子元素

nextElementChild 下一个兄弟元素

previousElementSibling 上一个兄弟元素

childELementCount 子元素总数

Children 所有子元素

parentElement 父元素

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>文档标题</title>

</head>

<body>

<a href="#">我的链接</a>

<h1>我的标题</h1>

<script></script>

</body>

</html>

//元素树---->firstElementChild

console.log(document.body.firstElementChild);//a

//元素树

console.log(document.body.children);// [a, h1, script]

console.log(document.body.children[1]);//h1

console.log(document.body.firstElementChild.nextElementSibling);//h1

console.log(document.body.lastElementChild.previousElementSibling);//h1

兼容：（元素树和节点树 在IE678用法颠倒, <https://blog.csdn.net/c9316/article/details/82686439>）

举例：previousSibling：调用者是节点。IE678中指前一个元素节点（标签）。在火狐谷歌IE9+以后都指的是前一个节点（包括空文档和换行节点）。

previousElementSibling：在火狐谷歌IE9都指的是前一个元素节点。

在IE678中用previousSibling，在火狐谷歌IE9+以后用previousElementSibling来查找元素节点

解决：下一个兄弟节点=节点.previousElementSibling || 节点.previousSibling

console.log(document.body.firstElementChild||document.body.firstChild);

9.获取元素（标签）----》9种

(1-3 HTML获取元素）

1. 通过ID获取元素document.getElementById("id名 盖特埃利理门此白ID")

<h1 id="one">1111</h1>

console.log(document.getElementById("one"));

1. 通过标签名来获取元素，结果是类数组（获取多个标签）node.getElementsByTagName("标签名 盖特百里门磁白太哥")

<ul>

<li>000</li>

<li>111</li>

<li>222</li>

</ul>

console.log(document.getElementsByTagName("li"));

//[li, li, li] 结果是：类数组

1. 通过name属性来获取元素，结果是类数组（获取多个标签）document.getElementsByName("name属性")

<form action="">

<input type="checkbox" name="ha"/>ha1

<input type="checkbox" name="ha"/>ha2

<input type="checkbox" name="ha"/>ha3

</form>

console.log(document.getElementsByName("ha"));

//[input, input, input]

（5-4 CSS获取元素）

1. 通过class获取元素,结果是类数组（获取多个标签）node.getElementsByClassName("class名")

问题：存在兼容性



<p class="aa">aaa</p>

<p class="bb">bbb</p>

<p class="aa">ccc</p>

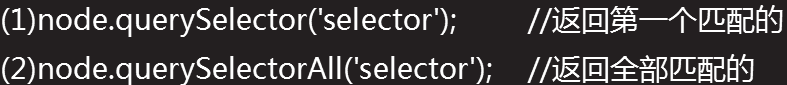
console.log(document.getElementsByClassName("aa"));// [p.aa, p.aa]

console.log(document.getElementsByClassName("aa")[1]);//<p class="aa">ccc</p>

console.log(document.getElementsByClassName("aa").length);//2

console.log(document.getElementsByClassName("bb"));//[p.bb]

1. 通过css选择器获取元素（13种选择器）



querySelector，结果：一个值

querySelectorAll，结果是：类数组 快瑞C来客特儿

问题：存在兼容性



var hf=document.getElementById("four");

console.log(hf.querySelectorAll("p"));// [p.aa, p.bb, p.aa] 查找的是id=four里面所有的p

console.log(document.querySelector("p>u"));//u

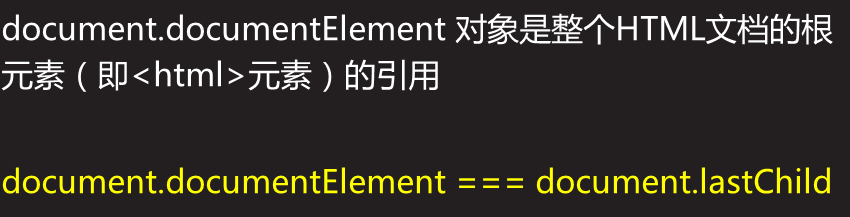
查找页面上的u ,结果：一个值

console.log(document.querySelectorAll(".aa"));//[p.aa, p.aa]//查找页面上所有的class=aa的标签，结果是：类数组

（6-9其他方法---了解）

6.



7.

8.

9.

10. 获取元素的内容(3种)

1.innerHTML(所有内容）

var h1=document.getElementById("one");

var h1con=h1.innerHTML;//获取

console.log(h1con);//获取 1111<i>2222</i>

h1.innerHTML='<a href="#">3333</a>4444';//设置 可以识别标签

console.log(h1.innerHTML);//<a href="#">333</a>444

2.textContent（文本内容）



var h1=document.getElementById("one");

var h1con=h1.textContent;//获取

console.log(h1con);//11112222

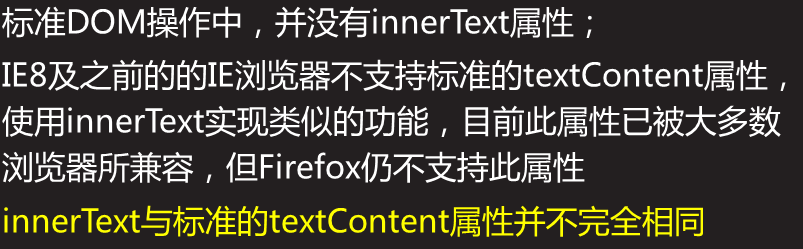
h1.textContent='<a href="#">3333</a>4444';//设置 不可以识别标签

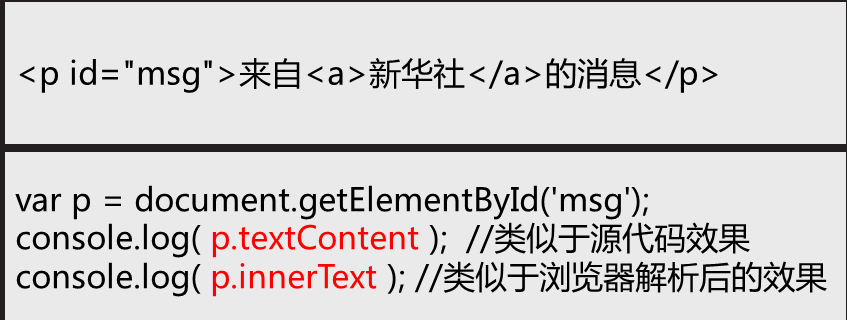
console.log(h1.textContent);//<a href="#">3333</a>4444

问题：



解决：innerText

3.innerText（文本内容,解决textContent兼容）



var h1=document.getElementById("one");

var h1con=h1.innerText;//获取

console.log(h1con);//11112222

h1.innerText='<a href="#">3333</a>4444';//设置 不可以识别标签

console.log(h1.innerText);//<a href="#">333</a>444

11属性

1. Attributes 属性集合



var a=document.getElementById("one");

console.log(a.attributes);//集合，类数组

console.log(a.attributes.length);//4

console.log(a.attributes[1]);//id="one"

1. 读取（获取）属性（4种方法）

var a=document.getElementById("one");

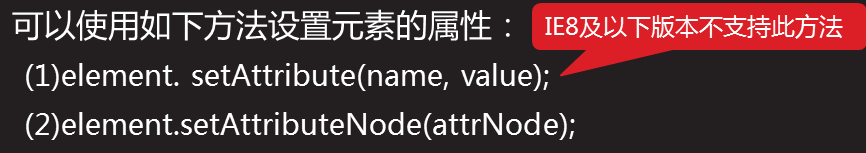
console.log(a.attributes[1].value);//one

console.log(a.attributes["id"].value);//one

console.log(a.getAttributeNode("id").value);//one

console.log(a.getAttribute("id"));//one 有兼容IE8及一下版本不支持

1. 修改（设置）属性（2种方法）



var a=document.getElementById("one");

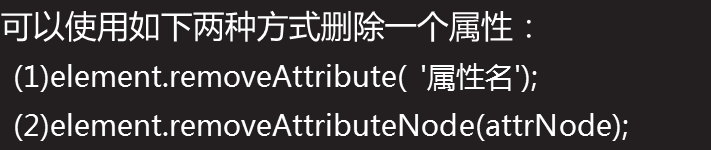
1. setAttribute("class","bb");//class 由aa--->bb 有兼容

var attra=document.createAttribute("class");

attra.nodeValue="cc";

a.setAttributeNode(attra);//先创建，再修改 class 由aa--->cc

4.移除属性（2种方法）



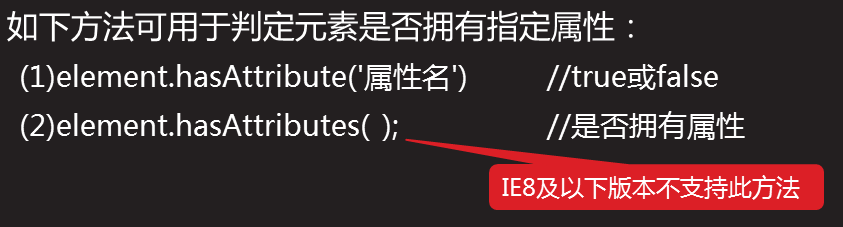
var a=document.getElementById("one");

a.removeAttribute("class");//有兼容IE8及一下版本不支持 删除class属性

var attra=a.attributes[2];

a.removeAttributeNode(attra);//先找到，再移除 删除class属性

5.判断属性



var a=document.getElementById("one"); console.log(a.hasAttribute("class"));//true console.log(a.hasAttributes("class"));//true

1. 样式操作
2. 核心DOM

只能操作行内（内嵌）样式-->style里面的所有内容(核心DOM)

//麻烦获取

console.log(div.getAttribute("style"));//只能获取html属性

div.setAttribute("style","border:10px solid blue;");//替换

//麻烦设置 div.setAttribute("style","height:200px;width:300px;border:10px solid blue;");//要想修改，需要全部重写

1. HTML DOM

只能操作行内（内嵌）样式-->style里面的单个属性(HTML DOM)

//简单行内获取

console.log(div.style.border);

//简单行内设置

div.style.border="10px solid blue";

//获取 所有方法样式（兼容）

if(div.currentStyle){

console.log(div.currentStyle.color);//IE

console.log(div.currentStyle.fontSize);//IE

}else{

console.log(getComputedStyle(div).color);//ff

console.log(getComputedStyle(div).fontSize);//ff

}

//设置 所有方法样式--->只能设置内嵌 内嵌>内部和外部

div.style.color="pink";

1. 创建节点（5种）
2. 创建元素节点document.createElement(“标签名”)

在页面上显示节点，要分三步

var body=document.body;

//1.创建元素节点

var h1=document.createElement("h1");

// console.log(h1.nodeName);//H1

// console.log(h1.nodeValue);//null

//2.添加内容

h1.innerHTML="hahaha";

//3.插入

body.appendChild(h1);

1. 创建文本节点（了解）document.createTextNode("文本内容")

//1.创建文本节点

var he=document.createTextNode("hello");

console.log(he.nodeName);//#text

console.log(he.nodeValue);//hello

//2.插入

body.appendChild(he);

1. 创建注释节点（了解）document.createComment("注释内容")

//1.创建注释节点

var zs=document.createComment("我是注释节点");

console.log(zs.nodeName);//#comment

console.log(zs.nodeValue);//我是注释节点

//2.插入

body.appendChild(zs);

1. 创建属性节点（了解）

//1.创建属性节点

var clas=document.createAttribute("class");

//2.设置属性值

clas.nodeValue="one";

//3.插入到div标签里

document.getElementById("diva").setAttributeNode(clas);

5.创建文档片段节点（了解）



<h1>1111</h1>

<b>aaa</b>

<script>

//1.创建文档片段

var fg=document.createDocumentFragment();

//2.把页面中的内容放入到文档片段里

//-->注意：开始b在页面上不显示，因为他要作为文档片段的内容fg.appendChild(document.querySelector("b"));

console.log(fg.childNodes[0]);//验证把b给了文档片段

//3.插入 ---->插入成功才会在页面上显示b

document.body.appendChild(fg);

</script>

或者

<div id="two"></div>

//3.将文档片段重新放入到页面(div)里

document.getElementById("two").appendChild(fg);

1. 插入节点（2种）
2. appendChild（作为最后一个子节点，插入）



<ul>

<li>111</li>

<li>222</li>

</ul>

<script>

var ul=document.querySelector("ul");

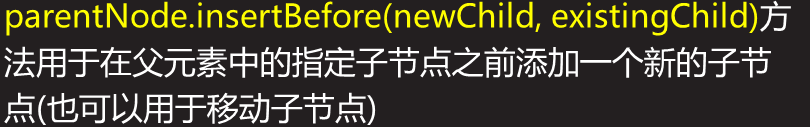
var lastli=document.createElement("li");

lastli.textContent="333";

ul.appendChild(lastli);//只能插入到父元素的最后

</script>

1. insertBefore(指定子节点前面，插入，移动）



<ul>

<li>111</li>

<li>222</li>

</ul>

<script>

var ul=document.querySelector("ul");

var newli=document.createElement("li");

newli.textContent="000";

//console.log(ul.children[0]);

ul.insertBefore(newli,ul.children[0])//插入

</script>

<ul>

<li>111</li>

<li>222</li>

<li>333</li>

<li>444</li>

</ul>

<script>

var ul=document.querySelector("ul");

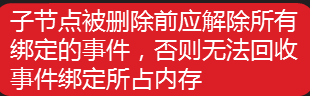
ul.insertBefore(ul.children[3],ul.children[0]);//移动

</script>

1. 删除节点



注意：



//1.找到父元素--方法1

var ul=document.getElementById("menu");

//2.删除

ul.removeChild(ul.children[1]);

//1.找到父元素---方法2

var ul=document.getElementById("menu");

//2.找要删除的元素

var lid=document.getElementsByTagName("li")[1];

//3.删除

ul.removeChild(lid);

1. 替换节点



<ul id="menu">

<li>aaa</li>

<li>bbb</li>

<li>ccc</li>

</ul>

<script>

//1.找到父节点

var ul=document.getElementById("menu");

//2.创建新节点

var newli=document.createElement("li");

newli.innerHTML="<u>uuuu</u>";

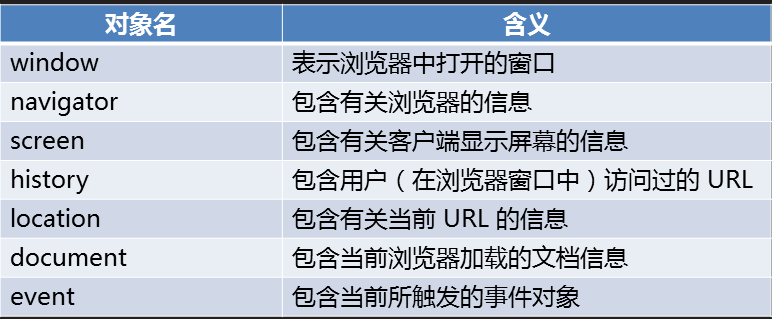
//3.替换

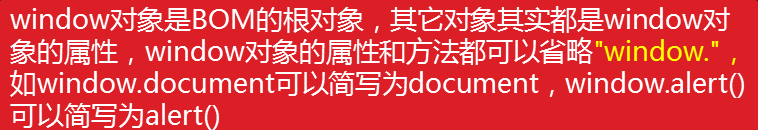
ul.replaceChild(newli,ul.children[1]);

</script>

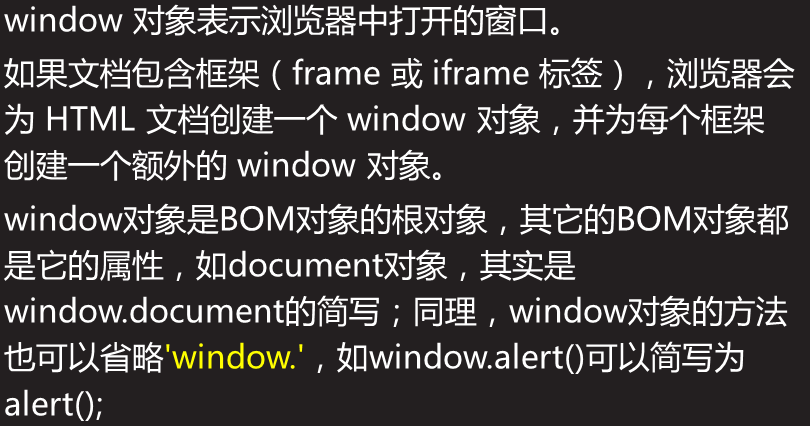
BOM

1. BOM模型

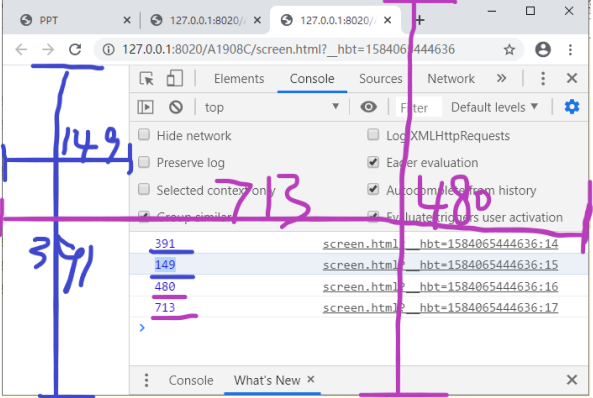




1. window对象
2. window的概述







IE8 或更早的版本不支持window对象的outerWidth/outerHeight属性 console.log(window.innerHeight);//391

//浏览器窗口可视区高度（不包括浏览器控制台、菜单栏、工具栏）

console.log(window.innerWidth);//149

//浏览器窗口可视区宽度（不包括浏览器控制台、菜单栏、工具栏）

console.log(window.outerHeight);//480

//浏览器窗口的高度(包括浏览器控制台、菜单栏、工具栏)

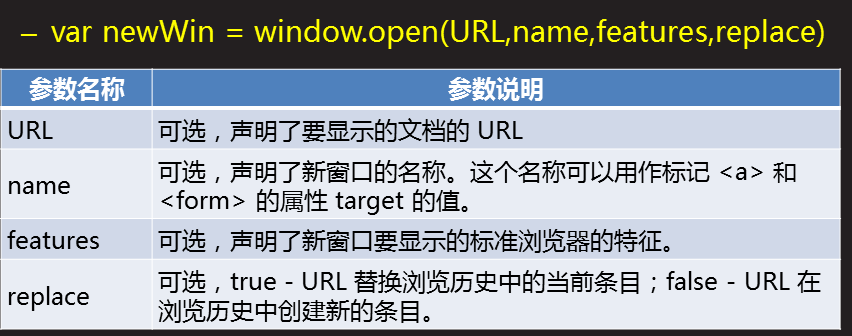
console.log(window.outerWidth);//713

//浏览器窗口的宽度(包括浏览器控制台、菜单栏、工具栏)



1. 窗口的打开和关闭

打开新窗口



**Features**



关闭窗口

1. window.close();//关闭当前窗口
2. 某个变量名.close();//关闭指定的某个窗口

<!--<a href="1.html" target="\_blank">打开窗口</a>-->

<button id="btn">打开窗口</button>

<button id="btn2">关闭当前窗口</button>

<button id="btn3">关闭小窗口</button>

<script>

//window.alert("111");

// console.log(window.innerHeight);

// console.log(window.outerHeight);

var newopen;

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

btn.onclick=function(){

//打开newopen小窗口

newopen=window.open("1.html","\_blank","height=300,width=300,left=400,top=100");

}

var btn2=document.getElementById("btn2");

btn2.onclick=function(){

window.close();//关闭了窗口，当前自己的大窗口关闭了，没有关闭小窗口

}

var btn3=document.getElementById("btn3");

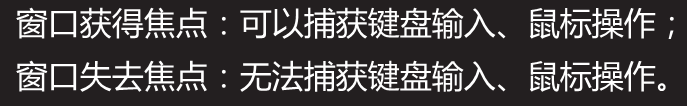
btn3.onclick=function(){

newopen.close();//关闭小窗口(newopen)

}

</script>

1. 获取焦点和失去焦点



某个input.focus（） //获取焦点

某个input.blur（） //失去焦点

autofocus="autofocus" 自动获取焦点 HTML5

<form action="">

<input type="text" id="foa" />

<input type="text" id="fob" />

<input type="text" id="foc" autofocus="autofocus"/>

</form>

<button id="btn">获取焦点</button>

<button id="btn2">失去焦点</button>

<script>

var foc=document.getElementById("foc");

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

btn.onclick=function(){

foc.focus();//获取焦点

}

var btn2=document.getElementById("btn2");

btn2.onclick=function(){

foc.blur();//失去焦点

}

</script>

1. 对话框
2. 警告框 window.alert() alert()
3. 输入提示框 window.prompt() prompt()
4. 确认框 window.confirm() confirm()
5. 定时器



1. 周期性定时器



1.第一步 声明一个全局的对象来接受定时器

var timer=null;

2.启动定时器

timer=setInterval(执行语句（一般函数），时间间隔（毫秒）)

写法：

timer=setInterval(function(){}，1000)

或者

timer=setInterval(show，1000)

function show(){}

3.停止定时器

clearInterval(timer);

timer=null;

//1.第一步 声明一个全局的对象来接受定时器

var timer=null;

//2.启动定时器

var n=0;

timer=setInterval(function(){

n++;

console.log(n);

//3.停止定时器

if(n>=5){

clearInterval(timer);

timer=null;

}

},1000)

1. 一次性定时器



1.第一步 声明一个全局的对象来接受定时器

var timer=null;

2.启动定时器

timer=setTimeout(执行语句（一般函数），时间间隔（毫秒）)

写法：

timer=setTimeout(function(){}，1000)

或者

timer=setTimeout(show，1000)

function show(){}

3.停止定时器

clearTimeout(timer);

timer=null;

<button>再睡5秒</button>

<script>

//1.创建一个全局对象，接受定时器

var timer=null;

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

btn.onclick=function(){

btn.innerHTML="起床了";

if(timer==null){//防止按钮多次点击弹多个框

//2.启动定时器

timer=setTimeout(function(){

alert("你该起床了");

//删除定时器

clearTimeout(timer);

timer=null;

btn.innerHTML="再睡5秒";

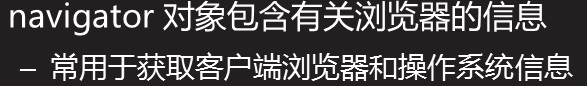
},5000)

}

}

</script>

2.navigator对象



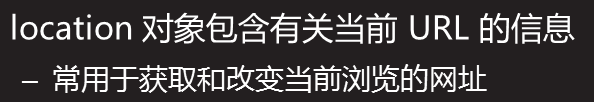


for(var attr in navigator){

console.log(attr+"--->"+navigator[attr]);

}

1. Location对象



<button id="btn">跳转到1.html</button>

console.log(location.href);//获取当前页面的网址 获取

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

btn.ondblclick=function(){//双击跳转页面 设置

location.href="1.html";

}

属性



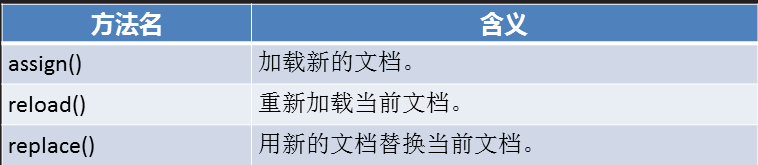
[http://127.0.0.1:8020/A1908C/navigator.html?\_\_hbt=1583982922077#111](http://127.0.0.1:8020/A1908C/navigator.html?__hbt=1583982922077" \l "111)

console.log(location.hash);//#111

console.log(location.host);//127.0.0.1:8020

console.log(location.hostname);//127.0.0.1

方法



<h1 onclick="openz()">111</h1>

function openz(){

//location.href="1.html";//替换打开新页面 有历史记录

//location.replace("1.html");//替换打开新页面 没有历史记录

//location.assign("1.html");//替换打开新页面 有历史记录

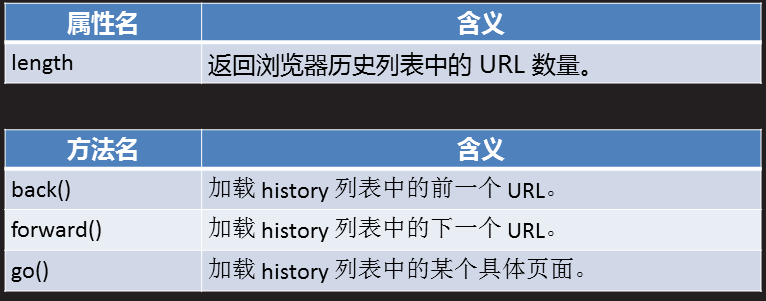
location.reload();//刷新 没有历史记录

}

1. history对象



属性和方法



console.log(history.length);//历史列表中URL的数量

history.go(1);//前往下一个页面

history.go(-1);//返回上一个页面

history.go(0);//刷新

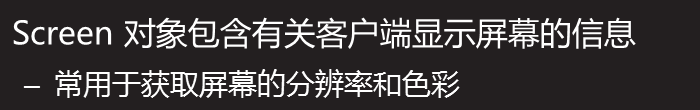
history.go(2);//前往下一个页面的下一个页面

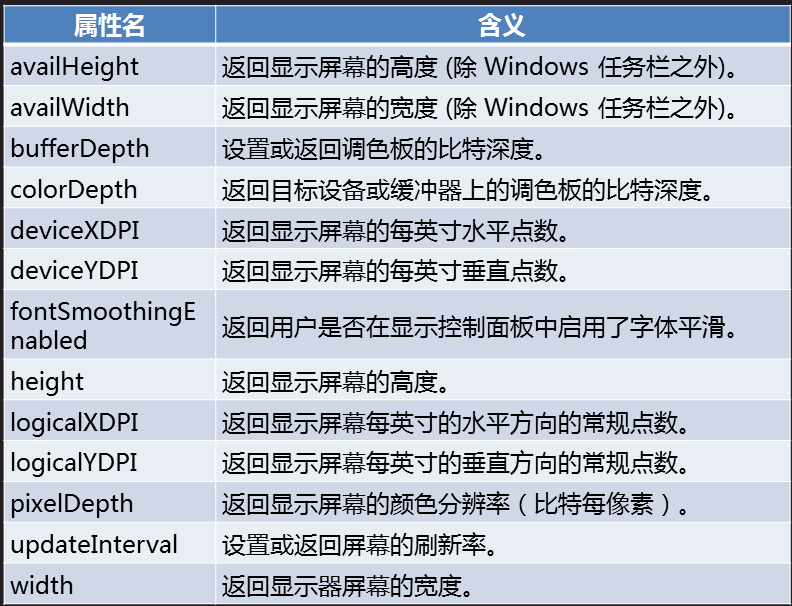
history.go(-2);//返回上一个的上一个页面

history.back();//返回上一个页面

history.forward();//前往下一个页面

1. screen对象





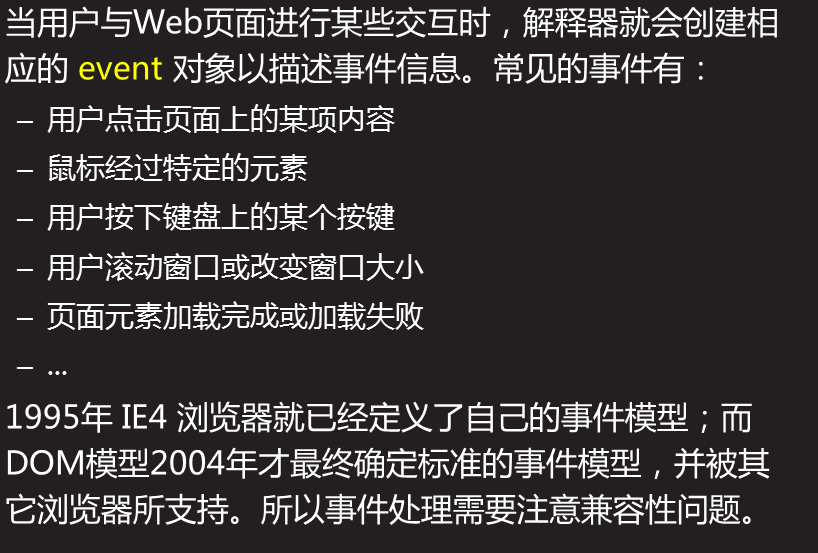
console.log(screen.availHeight);//显示浏览器的屏幕的高度(除去下面任务栏)

console.log(screen.availWidth);//除任务栏之外，显示屏的宽

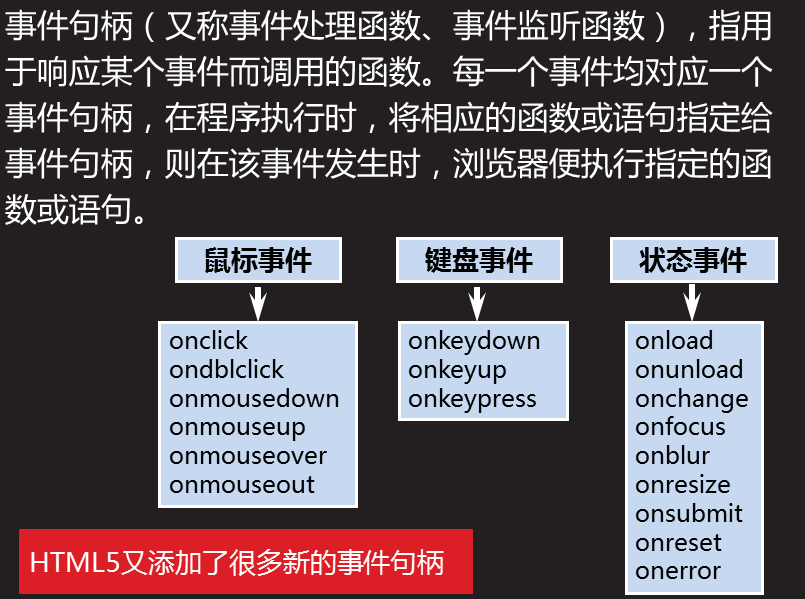
console.log(screen.height);//显示浏览器的屏幕的高度(没有除去下面任务栏)

console.log(screen.width);//显示屏的宽度(包含任务栏)

1. event对象
2. 事件的概述（注意：兼容）



1. 事件句柄（事件）



1.鼠标事件

1.onclick 点击

2.ondblclick 双击

3.onmouseover 移入 onmouseout 移出

鼠标移动到自身时候会触发事件（直接移动到子级元素上时，由里向外），同时移动到其子元素身上****也会****触发事件，（多次触发父级元素，由父级元素移动到子级元素上时，有冒泡）

4.onmouseup 移入元素鼠标抬起 onmousedown 移入元素鼠标按下

5.onmousemove 移入元素 反复触发

6.onmouseenter 移入 onmouseleave 移出

鼠标移动到自身是会触发事件（直接移动到子级元素上时，由外向里），但是移动到其子元素身上****不会****触发事件，（不会多次触发父级元素，由父级元素移动到子级元素上时，没有冒泡）

2.键盘事件

1.onkeydown 键盘按下

2.onkeyup 键盘抬起

3.onkeypress 键盘按下抬起（键盘按键被按下并释放一个键时发生）

3.状态事件

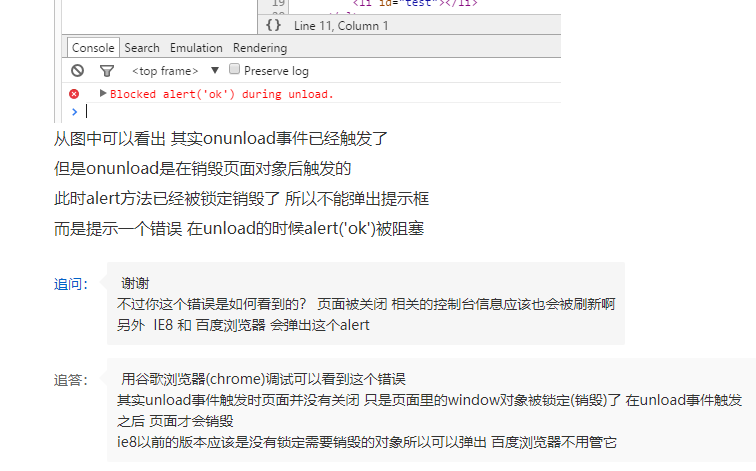
1.onload 预加载 window.onload=function(){}

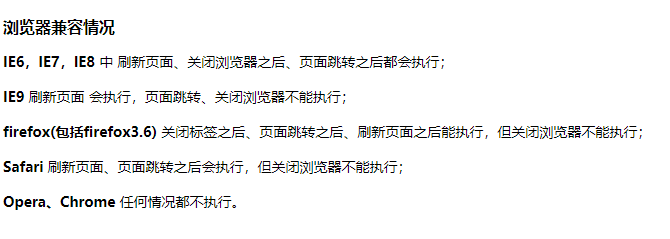
2.onunload 用户退出页面时触发（点击一个页面，提交表单，关闭浏览器窗口，刷新等 兼容）

window.onunload=function(){

alert("ok");

}

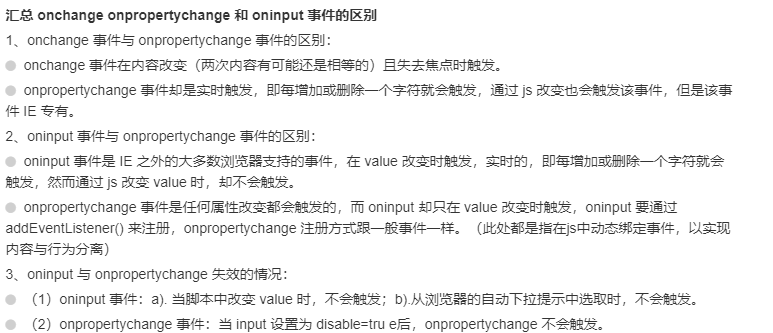




3.onchange 值改变并且失去焦点（select可以用）

oninput value值改变（select不可用） 不兼容IE

onporterchange html值发生改变时触发 IE专有



4.onfocus 获取焦点

5.onblur 失去焦点

6.onreset 重置 都要把事件添加到form上，配合input reset一块使用

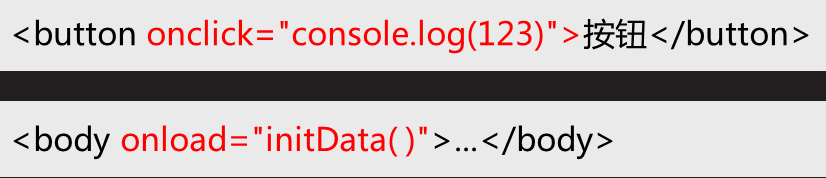
7.onsubmit 提交 都要把事件添加到form上，配合 input submit一块使用

8.onerror 加载外部文件时（文档或图片）出错的时候触发

<img src="imga/8.jpg" onerror="javascript:this.src='imga/1.jpg';">

9.onresize 浏览器被重置大小时<https://www.runoob.com/try/try.php?filename=tryjsref_onresize>

1. 事件定义（3种方法）
2. 直接在HTML中定义元素的事件相关属性（HTML事件）





注意：this

<div onclick="console.log('hahaha')">元素div</div>

<button onclick="show(this)">按钮</button>

<script>

console.log(this);//整个js中会有一个this,this--->window

function show(n){

console.log(n);//n-->this-->button(DOM)

console.log(this);//this-->window

console.log("AAA");

}

function showA(){//普通函数中

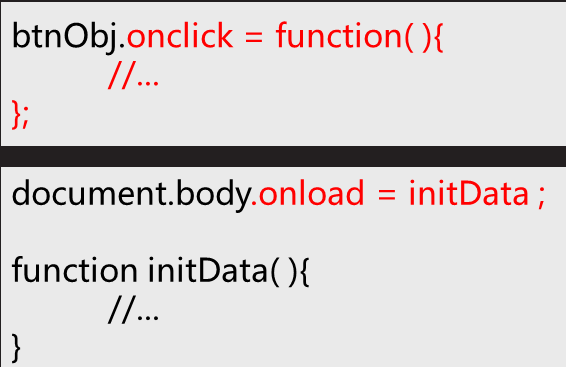
console.log(this);//this-->window

}

showA();

</script>

1. 在javascript中为元素的事件相关属性赋值（DOM0）





注意：不能是同一个事件多次，可以是不同的事件，this指的是DOM

<div id="diva">按钮</div>

<script>

var diva=document.getElementById("diva");

diva.onclick=function(){

console.log(this);//this-->div(DOM)

console.log("111");

}

diva.onclick=function(){

console.log("222");//最终输出222,后面的事件把前面的事件给覆盖了

}

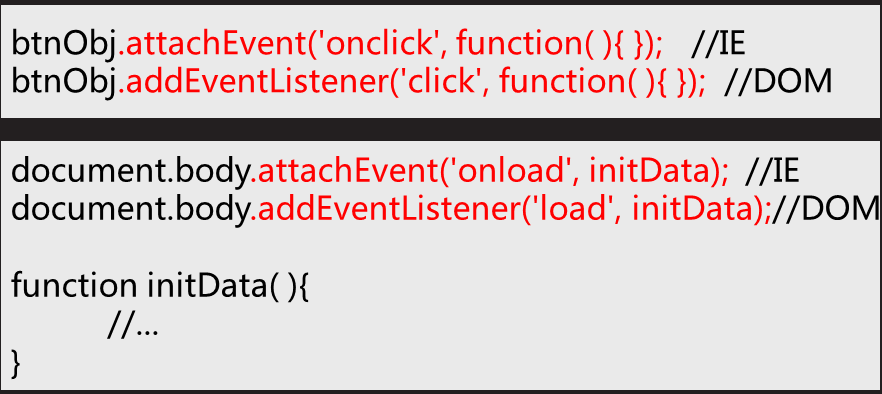
diva.onmouseover=function(){

console.log("333");

}

</script>

1. 高级事件处理方式（事件监听），一个事件可以绑定多个监听函数（DOM2）





<button id="btn">按钮</button>

<script>

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

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

console.log(this);//this-->button(DOM)

console.log("111");

})

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

**console.log("222");**

**}**)

if(btn.attachEvent){

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

console.log("333");

})//没有第三个参数，无法指定冒泡方向，就是由里向外

}else{

btn.addEventListener("click",function(){//FF/DOM

console.log("333");

},false)//默认：冒泡方向是由里向外(false) ,true:由外向里

}

*</scrip*t>

//用对象封装多个方法

<https://www.bbsmax.com/A/A7zgXyZkJ4/>

//封装

function addHandler(element,type,handler){

if(element.addEventListener){

element.addEventListener(type,handler,false);

}

else if(element.attachEvent){

element.attachEvent('on'+type,handler);

}

else{

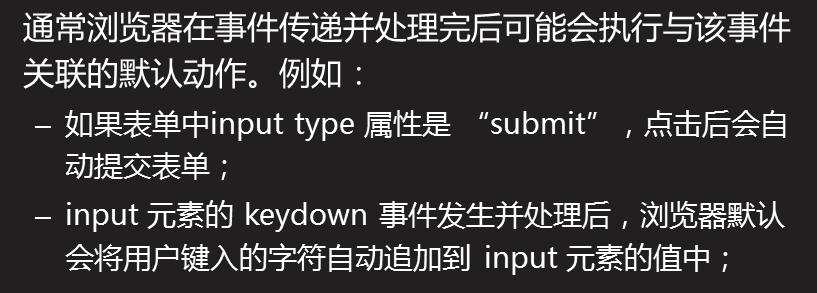
element['on'+type] = handler;

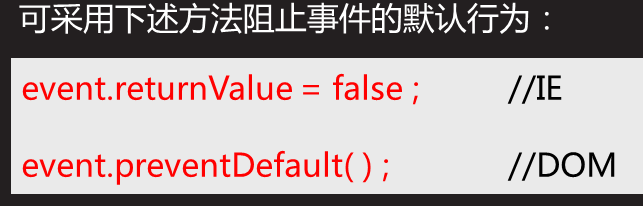
}

}

addHandler(btn,"click",function(){console.log("333");})

1. 取消事件的默认行为





if(typeof e.preventDefault!="undefined"){

e.preventDefault();

}else{

e.returnValue=false;

}

<a href="a.html" id="anb">aaa</a>

<script>

var anb=document.getElementById("anb");

function stopEvent(e){//封装

if(typeof e.preventDefault!="undefined"){

e.preventDefault();

}else{

e.returnValue=false;

}

}

anb.onclick=function(){

var e =window.event || arguments.callee.caller.arguments[0];

stopEvent(e);

alert("hello");

}

</script>

5. 获取event对象（Event 有兼容， 2种）

window.event//谷歌，IE 不支持火狐

arguments.callee.caller.arguments[0]//谷歌，火狐 不支持IE

1.不需要传参法

var e = window.event||arguments.callee.caller.arguments[0];

<a href="a.html" id="anb">aaa</a>

<script>

var anb=document.getElementById("anb");

anb.onclick=function(){

var e =window.event || arguments.callee.caller.arguments[0];

stopEvent(e);

alert("hello");

}

</script>

2.需要传参的方式

var e = event || window.event || arguments.callee.caller.arguments[0];

<a href="a.html" id="anb">aaa</a>

<script>

var anb=document.getElementById("anb");

anb.onclick=function(event){

var e = event || window.event || arguments.callee.caller.arguments[0];

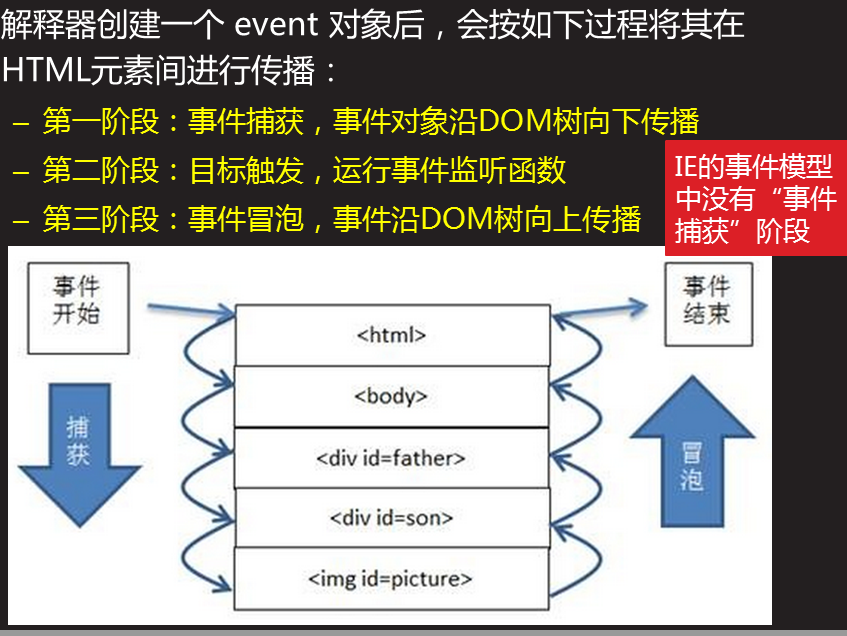
stopEvent(e);

alert("hello");

}

</script>

6.事件周期



1. 冒泡：方向--->由里向外（span--->p----->div）

<div id="diva">

<p id="pb">

<span id="spanc"></span>

</p>

</div>

<script>

var diva=document.getElementById("diva");

var pb=document.getElementById("pb");

var spanc=document.getElementById("spanc");

diva.onclick=function(){

alert("div");

}

pb.onclick=function(){

alert("p");

}

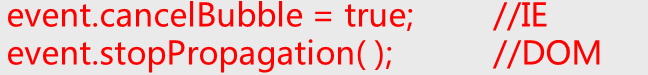
spanc.onclick=function(){

alert("span");

}

</script>

2.取消事件冒泡



function stopP(){

var e = window.event || arguments.callee.caller.arguments[0];

if(e&&e.stopPropagation){

e.stopPropagation();//FF

}else{

e.cancelBubble=true;//IE

}

}

1. 获取目标对象



<div id="diva">111</div>

<script>

diva.onclick=function(event){

var e=event||window.event||arguments.callee.caller.arguments[0];

var mn=e.srcElement||F;

console.log(mn);

console.log(this);

}

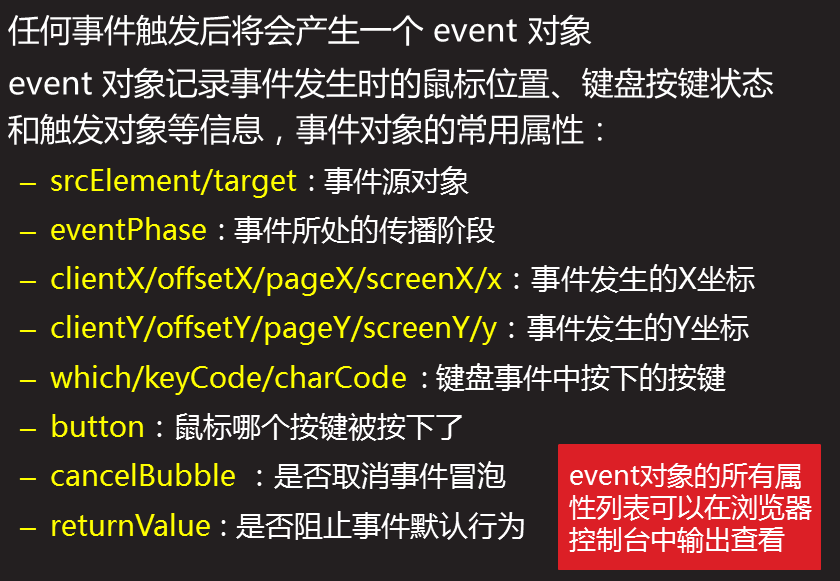
</script>

有冒泡的时候：

目标源对象：始终如一（你触发事件的元素）

this:随着元素的变化而变化（你绑定事件的元素）

1. event对象的常用属性



KeyCode:键盘的键码号

<input type="text" id="inp"/>

<script>

var inp=document.getElementById("inp");

inp.onkeydown=function(event){

var e=event||window.event||arguments.callee.caller.arguments[0];

console.log(e.keyCode);

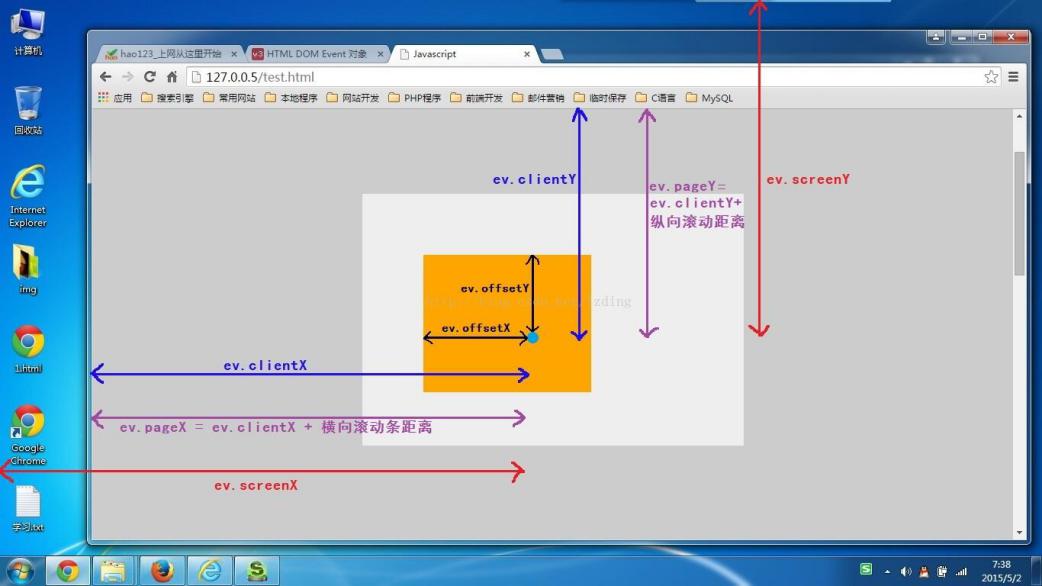
// if(e.keyCode==13){

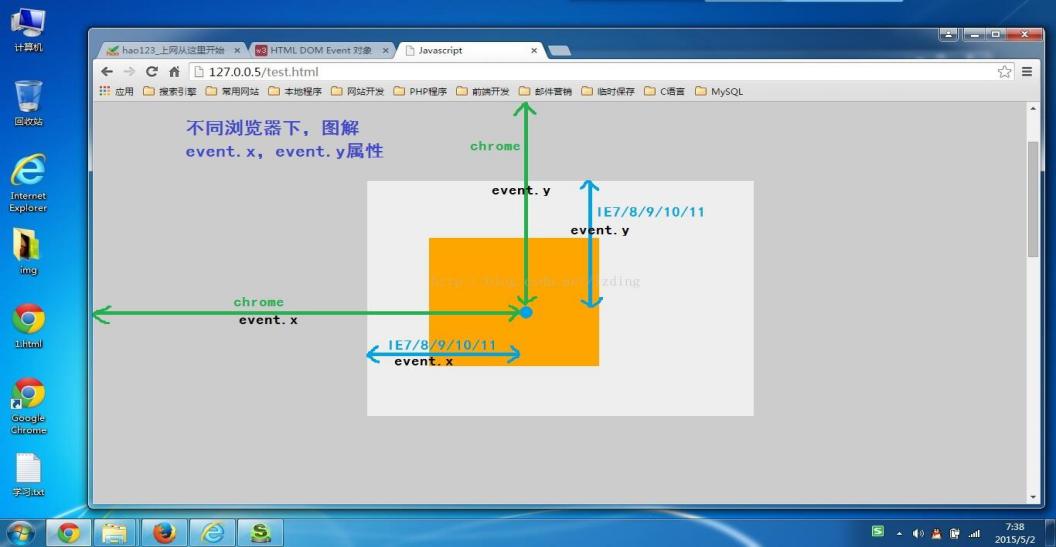
// console.log("Enter");

// }

}

</script>





<https://blog.csdn.net/learner_boy/article/details/72190462>

<https://www.cnblogs.com/jiangxiaobo/p/6593584.html>

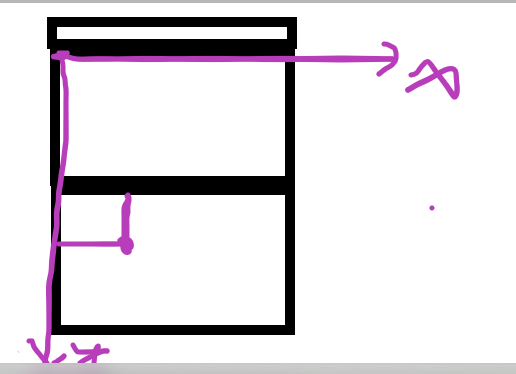
**e.x/e.y:**

event.clientX返回事件发生时,鼠标相对于客户窗口的X坐标  
   event.X也一样

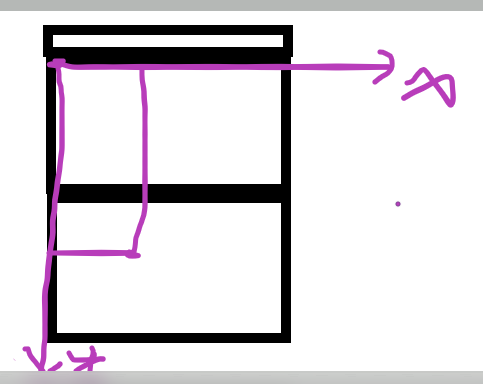
   但是如果设置事件对象的定位属性值为relative(父相子绝）  
   event.clientX不变

opera，chrome和safari的event.x返回值和event.clientX相同  
   IE中： event.X返回事件对象的相对于本体（父元素的边框）的坐标

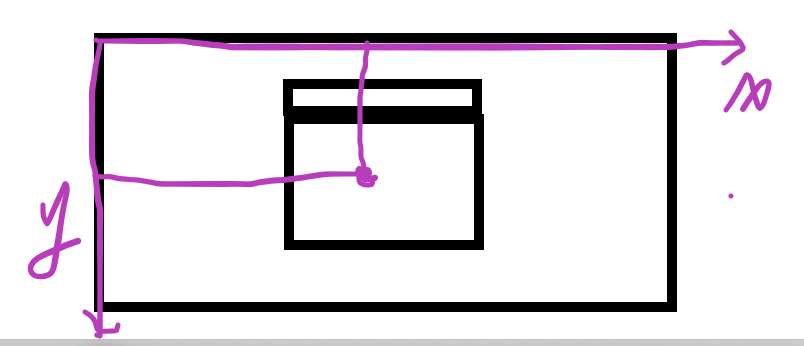
clientX/clientY:相对于浏览器显示区域



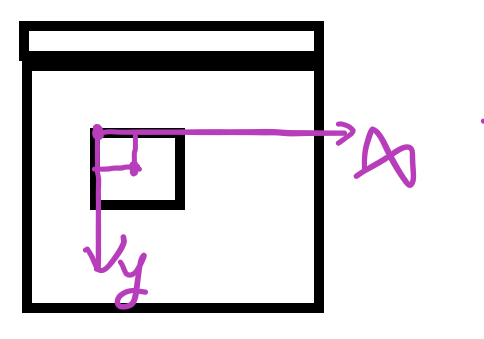
pageX/pageY:相对于页面原点坐标



screenX/screenY:相当于显示屏幕的原点坐标



offsetX/offsetY：相对于事件源对象的原点坐标



1. 移除事件（2种）

1.DOM0级,移除事件 btn.onclick=null;

2.DOM2级, 移除事件

if(diva.removeEventListener){

diva.removeEventListener("click",sha,false);//FF

}else{

diva.detachEvent("onclick",sha);//IE

}

<https://blog.csdn.net/xiasohuai/article/details/83063293>

1. DOM0级事件和DOM2级事件

<https://www.cnblogs.com/sumg/p/3787769.html>

7. 元素偏移和大小

offsetWidth 水平方向 width + 左右padding + 左右border

offsetHeight 垂直方向 height + 上下padding + 上下border

clientWidth水平方向 width + 左右padding

clientHeight 垂直方向 height + 上下padding

offsetTop 获取当前元素到 定位父节点(父相子绝) 的top方向的距离(没有父节点，相对于body)

offsetLeft 获取当前元素到 定位父节点 (父相子绝)的left方向的距离(没有父节点，相对于body)

offsetParent:查找位置父元素

scrollWidth （兼容）

内部的元素的宽度=内部元素的clientWidth(width+padding)

外部元素的宽度(内部元素宽度没有被挡住)=外部元素的width+3

外部元素的宽度(内部元素宽度被挡住)=（内部元素的宽度+内部元素的内边距+内部元素的外边距+内部元素的边框）内部元素的外盒宽度+外部元素(自己)的内边距（火狐不加自己）

scrollHeight （兼容）

内部的元素的高度=内部元素的clientHeight(height+padding)

外部元素的高度(内部元素高度没有被挡住)=外部元素的height+3

外部元素的高度(内部元素高度被挡住)=（内部元素的高度+内部元素的内边距+内部元素的外边距+内部元素的边框）内部元素的外盒高度+外部元素(自己)的内边距(火狐不加自己)

document.documentElement.scrollTop 返回文档的滚动top方向的距离（当窗口发生滚动时值改变）

document.documentElement.scrollLeft 返回文档的滚动left方向的距离（当窗口发生滚动时值改变）

兼容(用或运算，document.documentElement.scrollTop||document.body.scrollTop)

document.body.scrollTop 返回文档的滚动top方向的距离（当窗口发生滚动时值改变）

document.body.scrollLeft 返回文档的滚动left方向的距离（当窗口发生滚动时值改变）

<script>

document.onscroll=function(){

var topa=document.documentElement.scrollTop||document.body.scrollTop;

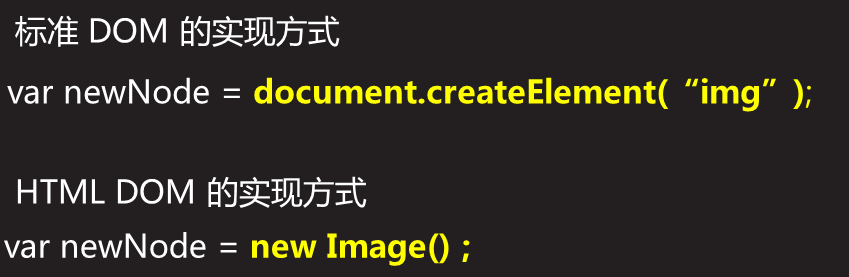
console.log(topa);

}

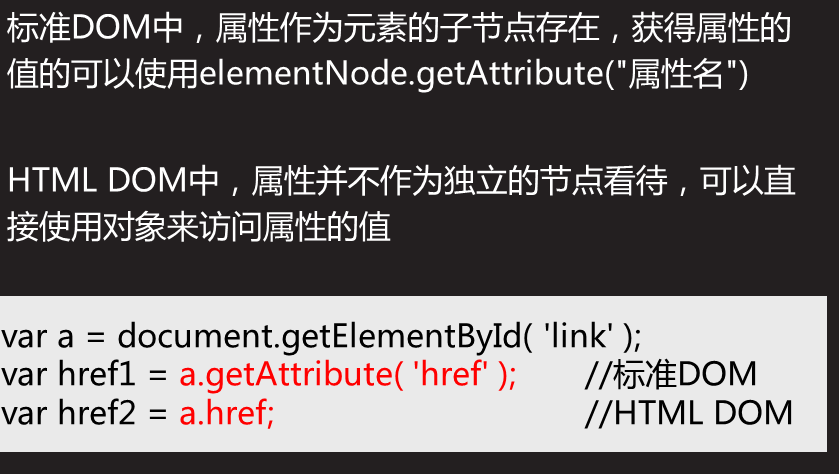
</script>

HTML DOM

1.创建图片



1. 属性操作



1. 样式操作



