JS盒子模型：通过JS可以获取CSS盒子模型中关于盒子的相关的样式

**1、通过style获取盒子的样式：**

元素.style.样式属性，例如：oDiv.style.width

但是只能获取写在元素的行内样式上的样式属性的值，如果行内上没有写这个样式，获取的就是空

只有这样写在行内上才可以获取到width的值 <div id="div1" style="width: 200px;">

var oDiv = document.getElementById("div1");

console.log(oDiv.style.width); ==> "200px"

**2、通过getComputedStyle或者currentStyle来获取盒子的样式：**

所有在页面中出现的元素都是经过浏览器计算渲染过的(有的是我们自己写的，有的是浏览器默认的)

获取元素经过计算的样式(所有的样式，不管写在哪，不管是否写，只要渲染出来了，都能获取到)

window.getComputedStyle([元素],[伪类])获取的是一个CSSStyleDeclaration样式集合(对象数据类型的)

如果想获取具体的某一个属性的，我们可以通过属性名获取属性值就可以了

getComputedStyle在IE6~8下不兼容，用currentStyle来出来

元素.currentStyle 获取的是一个对象

var oDiv = document.getElementById("div1");

console.dir(window.getComputedStyle(oDiv, null).margin);

console.dir(oDiv.currentStyle.margin);

每一个浏览器对默认值存在差异，为了保证JS获取的时候的准确性我们通常在CSS样式开始的时候把常用的样式都初始化一下

**3、通过浏览器提供好的属性和方法获取样式信息**

**.clientWidth** width+左右padding

**.clientHeight** height+上下padding

**.offsetWidth**  clientWidth+左右边框的宽度

**.offsetHeight**  clientHeight+上下边框的高度

**.scrollWidth和 .scrollHeight**分两种情况：

1)在没有内容溢出的情况下，scrollWidth/scrollHeight和clientWidth/clientHeight值相等

2)在有内容溢出的情况下:

不管是否隐藏了溢出的内容(overflow: hidden)，我们的计算是这样处理的:

scrollWidth 约等于 实际内容的真实宽度(被内容撑开后的宽度)+左padding

scrollHeight 约等于 实际内容的真实高(被内容撑开后的高度)+上padding

在项目中通常用scrollHeight获取当前HTML页面的总高度(不管有几屏的内容都算)

**console.log(document.documentElement.scrollHeight||document.body.scrollHeight);**

只获取当前浏览器一屏的高度

**document.documentElement.clientHeight||document.body.clientHeight**

先写document.documentElement在写document.body，这样就可以完全的兼容了

**.clientTop**  上边框的高度

**.clientLeft**  左边框的宽度

**.offsetTop** 当前元素距离父级参照物的上偏移量(从最外面的边框开始算，也就是得到的值不包含边框的高度)

**.offsetLeft**  当前元素距离父级参照物的左偏移量(从最外面的边框开始算，也就是得到的值不包含边框的宽度)

**.scrollTop** 竖向滚动条卷去的高度

**.scrollLeft**  横向滚动条卷去的宽度

scrollTop和scrollLeft 都有边界值的，当到达边界值的时候，不管你在写多大或者多小，值都不在继续改变了，例如：最小是0，你写-100，它的值还是0不改变的

**偏移量：**当前元素距离父级参照物左上角的 x轴和y轴 的位移

默认情况下的偏移量，都是相对于浏览器的

**offsetParent：父级参照物(偏移参照物)**

**parentNode：父亲节点**

父级参照物和父亲节点没有必然的关系，在默认情况下，页面中的所有的元素都在同一个水平面上，**默认的父级参照物都是body**

如果想改变默认的参照物关系，需要让元素脱离body这个平面，需要使用**position**设置(relative、absolute、fixed...)

盒子模型中除了**scrollTop和scrollLeft是可读写的**，其余的都是**只读的**(只能获取值不能修改值)

**四种处理兼容的方式：**

1、try catch因为每一次都要先执行一个不兼容，才会报错，才可以执行catch中的，所以性能不是很好

2、typeof 检测数据类型，每一次都需要typeof一下才会出结果，性能也不是最好的

3、window.getComputedStyle 首先获取这个属性的值，然后把这个值转换为布尔类型，再判断条件是否成立

4、"getComputedStyle" in window 直接查找，返回true或者false，我个人喜欢用最后一种方式来节约性能

获取当前页页面的URL地址：**window.location.href;**

**window.location.href：**在当前页面跳转:当前页面关闭，打开新的页面

window.location.href = "https://www.baidu.com/";

**window.open：**打开一个新的浏览器窗口实现页面跳转，原有的页面不关闭。该方法没有返回值，不会返回当前页面的URL地址

window.open("https://www.baidu.com/");

**图片延时加载原理：**

第一屏幕的图片延迟加载：开始给一张小的默认图，当页面加载完(或者是过一段时间再加载真实的图片)

第二屏幕及以后的图片延迟加载：先给一张小的默认图片，当我滚动到这个区域的时候，再开始加载我们的图片

**var** oImg = document.getElementsByTagName("img")[0];  
//500毫秒后才开始向服务器请求真实图片资源，并且做判断，只有正常加载出来后，才替换我们之前的默认图片  
**var** timer = window.setTimeout(**function** () {  
 //第一步：创建空的图片对象  
 **var** m = **new** Image;// <==> var m = document.createElement("img");  
 //第二步：把真实的图片地址给这个对象的src属性，让真实的图片开始加载  
 m.src = oImg.getAttribute("trueImg");  
 //第三步：判断是否可以正常的加载，可以的话我们让oImg变成真实的图片地址  
 m.onload = **function** () {  
 //onload:图片加载完成事件，可以进入这个事件说明图片可以正常的加载出来  
 oImg.src = **this**.src;  
 m = **null**;//创建的那个就没用了，我们销毁他  
 }  
}, 500);

不在第一屏幕的：当前图片所在盒子的底边框距离body的y轴偏移量<=浏览器窗口底边框距离body的y轴偏移量的时候，说明整张图片都进入我们的视野了，动态创建一个img标签加载我们真实地址，加载成功，把这张图片添加到盒子中

**瀑布流实现的原理：**

1、有五个并排的div，宽度设置为固定值，高度自适应

2、获取了多组图片数据，第一次拿出前五条数据，分别依次插入到五个div中，并且记录当前五个div的offsetHeight值(把这个五个值存储到数组中，然后数组按照由小到大排序)

3、在拿出五个数据，本次按照offsetHeight存储的值的由小到大的顺序依次的插入(循环数组，依次插入即可)，插入完成后，我们在重新的计算当前的高度，替换原有数组的数据

4、以后重复操作即可

**window.navigator.userAgent** 获取浏览器的版本信息