# JavaScript基础入门

第9天课堂笔记（本课程共10天）

# 目录

[JavaScript基础入门 1](#_Toc2927)

[目录 2](#_Toc25006)

[一、快捷位置和尺寸 3](#_Toc20592)

[1.1 offsetLeft属性和offsetTop属性 3](#_Toc4186)

[1.2 offsetWidth和offsetHeight 5](#_Toc18942)

[1.3 clientWidth和clientHeight 6](#_Toc5910)

[二、运动 7](#_Toc20272)

[2.1 定时器 7](#_Toc24685)

[2.2 简单运动模型 8](#_Toc30985)

[2.3 定时器的停止 8](#_Toc8041)

[2.4 简单运动需要注意的事儿 9](#_Toc7193)

[三、无缝连续滚动 10](#_Toc21834)

[四、JSON 12](#_Toc24552)

[4.1 最简单的JSON示例 12](#_Toc4527)

[4.2 JSON的嵌套 13](#_Toc27364)

[4.3 JSON项的添加和删除 13](#_Toc10584)

[4.4 JSON的遍历 13](#_Toc3764)

[五、运动框架 14](#_Toc16299)

# 一、快捷位置和尺寸

[offsetLeft](https://developer.mozilla.org/zh-CN/docs/Web/API/HTMLElement/offsetLeft) [offsetTop](https://developer.mozilla.org/zh-CN/docs/Web/API/HTMLElement/offsetTop) [offsetParent](https://developer.mozilla.org/zh-CN/docs/Web/API/HTMLElement/offsetParent) [offsetWidth](https://developer.mozilla.org/zh-CN/docs/Web/API/HTMLElement/offsetWidth) [offsetHeight](https://developer.mozilla.org/zh-CN/docs/Web/API/HTMLElement/offsetHeight) [clientWidth](https://developer.mozilla.org/zh-CN/docs/Web/API/Element/clientWidth) [clientHeight](https://developer.mozilla.org/zh-CN/docs/Web/API/Element/clientHeight)

DOM已经提供给我们计算后的样式，但是还觉得不方便，所以DOM又提供给我们一些API：

ele.offsetLeft

ele.offsetTop

ele.offsetWidth

ele.offsetHeight

ele.clientWidth

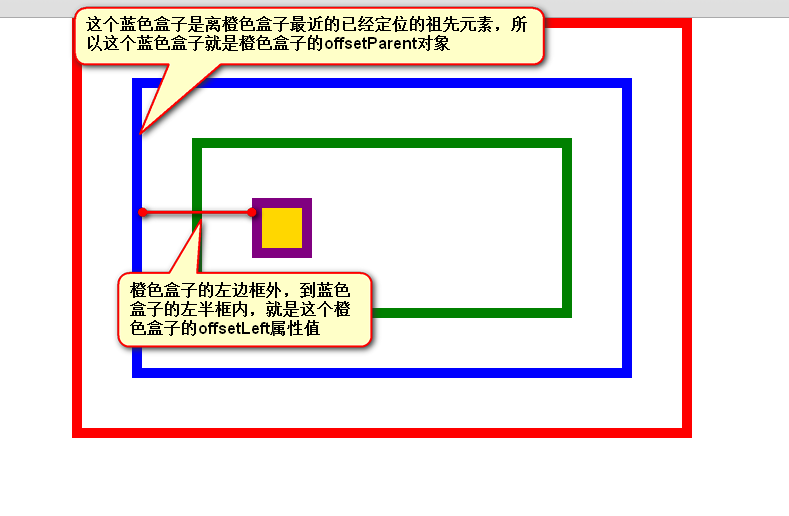
ele.clientHeight

## 1.1 offsetLeft属性和offsetTop属性

这两个属性的兼容性**非常差**，不要着急，我们慢慢来看。

IE9、IE9+、Chrome等高级浏览器：

一个元素的offsetLeft值，就是这个元素左边框外，到自己的offsetParent对象的左边框内的距离。



每一个元素，天生都有一个属性，叫做offsetParent，表示自己的“偏移参考盒子”，我们不称呼中文，就叫offsetParent。这个offsetParent是谁呢？

**就是自己祖先元素中，离自己最近的已经定位的元素，如果自己的祖先元素中，没有任何盒子进行了定位，那么offsetParent对象就是body。**

|  |
| --- |
| 1. op.offsetParent |

无论这个盒子自己是否定位，自己的offsetParent就是按照上述方法寻找。

IE6、IE7：

IE6、7的offsetParent对象是谁，和高级浏览器有非常大的不同。

情形1：自己如果没有定位属性，那么自己的offsetParent对象就是自己的祖先元素中离自己最近的有width或者有height的元素：

|  |
| --- |
| 1. <div class="box1"> 2. <div class="box2"> → 你好，我有宽度 , offsetParent 3. <div class="box3"> → 你好，我没有宽高 4. <p></p> → 你好，我没有定位 5. </div> 6. </div> 7. </div> |

|  |
| --- |
| 1. <div class="box1"> 2. <div class="box2"> 3. <div class="box3"> → 你好，我没有宽高，有定位 , offsetParent 4. <p></p> → 你好，我没有定位 5. </div> 6. </div> 7. </div> |

情形2：自己如果有定位属性

那么自己的offsetParent就是自己祖先元素中离自己最近的有定位的元素。

数值就是自己的左外边框到offsetParent对象的左内边框的值。

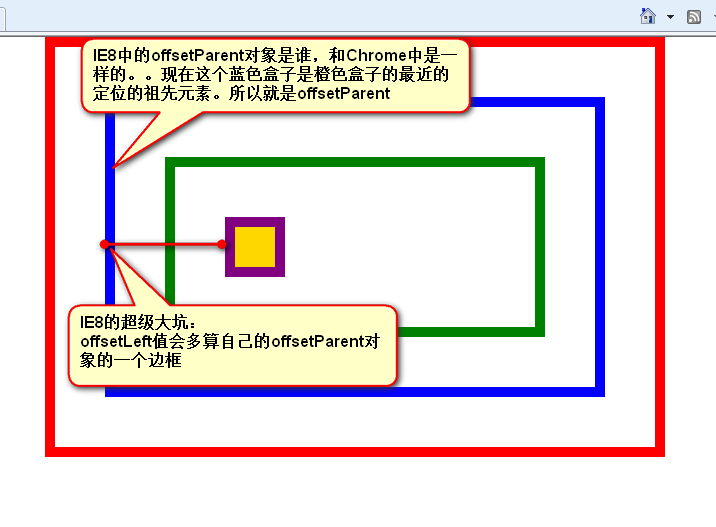
IE8：

IE8的offsetParent是谁呢？和高级浏览器一致：

无论自己是否定位，自己的offsetParent就是自己祖先元素中，离自己最近的已经定位的元素。

这一点，没有任何兼容问题！

**但是，多算了一条边框**



总结：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | IE6、7 | IE8 | | IE9、IE9+、高级浏览器 |
| offsetParent | 如果自己没有定位，那么就是自己父亲中有width或者有height或者有定位的元素。  如果自己有定位，那么就是和高级浏览器一致。 | 和高级浏览器一致 | 自己祖先元素中，离自己最近的已经定位的元素 | |
| offsetLeft | 和高级浏览器一致 | 多算一条border | | 自己的border外到offsetParet对象的border内 |

兼容性解决办法，不是能力检测，也不是版本检测，而是善用这个属性，要确保属性的使用条件：

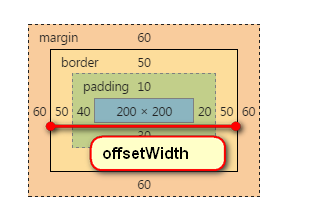
**自定位，父无边 (父亲也要定位，但是为了顺口，就不多说了)**

这样的话，所有浏览器的值都是一样的，offsetLeft、offsetTop值是number类型的，可以直接参与运算，不需要parseInt()

## 1.2 offsetWidth和offsetHeight

全线兼容，是**自己**的属性，和别的盒子无关。

一个盒子的offsetWidth值就是自己的 width+左右padding+左右border的宽度



如果盒子没有宽度，那么所有浏览器都将把px值当做offsetWidth，而不是100%；

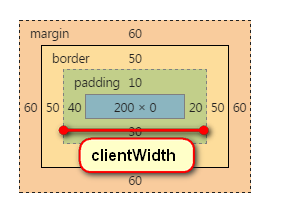
如果盒子没有高度，用文字撑的，所有浏览器都将把px值当做offsetHeight

总结一下，全线兼容。

特别注意，IE6、7、8下，盒子没有高度，文字撑的，用currentStyle.height是auto。体现出了offsetHeight的好用。

## 1.3 clientWidth和clientHeight

全线兼容，就一丢丢IE6的问题



client表示“客户端”这里就是一个名字而已，不用在意这个名字。

clientWidth就是自己的width+padding的值。 也就是说，比offsetWidth少了border。

如果盒子没有宽度，那么那么所有浏览器都将把px值当做clientWidth，而不是100%

如果盒子没有高度，用文字撑的，**IE6 clientHeight是0，其他浏览器都是数值**。

总结一下，我们的6个属性要铭记于心，就offsetLeft、offsetTop比较闹腾，但是合理使用，也没兼容问题了。

# 二、运动

## 2.1 定时器

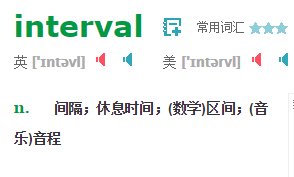
[setInterval](https://developer.mozilla.org/zh-CN/docs/Web/API/Window/setInterval)  [clearInterval](https://developer.mozilla.org/zh-CN/docs/Web/API/Window/clearInterval)

window对象有一个方法，叫做

|  |
| --- |
| 1. window.setInterval(函数，间隔时间); |

能够使每间隔时间，调用函数一次。我们习惯叫做定时器，按理说叫做“间隔器”。

单词：



演示：

|  |
| --- |
| 1. window.setInterval(function(){ 2. console.log(“你好”); 3. },1000); |

间隔时间是以毫秒为单位，1000毫秒就是1秒。

“毫”就是千分之一，

“厘”就是百分之一，

“分”就是十分之一

第一个参数，是一个函数，所以可以把一个匿名函数往里放，更可以用一个有名函数的引用放里面。

|  |
| --- |
| 1. function **fun**(){ 2. console.log("你好"); 3. } 4. window.setInterval(**fun**,1000); |

哲学上讲，setInterval()能够让函数每间隔时间执行。

我们说过，window对象，可以不写，所以：

|  |
| --- |
| 1. setInterval(function(){ 2. console.log(Math.random()); 3. },1000); |

定时器没有所谓的start、begin方法，只要setInterval了，定时器就开始运行了。

## 2.2 简单运动模型

视觉暂留：把连续相关的画面，连续播放，就是运动了。

|  |
| --- |
| 1. var nowleft = 111; 2. //定时器 3. setInterval(function(){ 4. //这个函数，就是每20毫秒调用一次。 5. **nowleft += 10**; 6. oDiv.style.left = nowleft + "px"; 7. },20); |

间隔时间是20毫秒，那么1秒中执行函数50次。也就是说，这个动画是每秒50帧，50fps。

那么现在我们来研究一个事儿，如果让这个盒子跑得更快？

上面的案例中，数值20间隔时间，这个数字越小运动越快；

数值10叫做步长，每一步的变化量，这个数字越大运动越快。

我们就有一个感觉，JavaScript描述动画，描述的是每一步的改变，并不是直接描述终点。这给我们的工作会带来不便，我们下午解决这个事儿。

举个例子，去森林公园玩儿的路线：

JS的模式，是告诉你第1毫秒你直走80厘米，第2毫秒你继续直走80厘米……

## 2.3 定时器的停止

setInterval的时候，要给这个定时器一个变量引用，停止的时候只需要clearInterval(timer)。

|  |
| --- |
| 1. timer = setInterval(function(){ 3. },20) 4. **clearInterval(timer)**; |

## 2.4 简单运动需要注意的事儿

我们的开始按钮是：

|  |
| --- |
| 1. startBtn.onclick = function(){ 2. //设置定时器 3. timer = setInterval(function(){ 4. nowleft += 2; 5. oDiv.style.left = nowleft + "px"; 6. }, 20); 7. } |

这个按钮持续点击，盒子运动越来越快。这是因为每次点击，盒子身上就有更多的定时器在作用。

解决办法，就是四个字的口诀“**设表先关**”。

|  |
| --- |
| 1. startBtn.onclick = function(){ 2. **//设表先关** 3. **clearInterval(timer);** 4. //设置定时器 5. timer = setInterval(function(){ 6. nowleft += 2; 7. oDiv.style.left = nowleft + "px"; 8. }, 20); 9. } |

还要注意一个事情：当盒子到终点，自己停止。比如起点是100，终点我们想要600自动停止：

下面的方法是错误的：

|  |
| --- |
| 1. var timer = setInterval(function(){ 2. **if(nowleft < 600){** 3. nowleft += **13**; 4. oDiv.style.left = nowleft + "px"; 5. **}else{** 6. clearInterval(timer); 7. **}** 8. },200); |

初始值是100，所以盒子的运动轨迹就是

100、113、126……594、607停表

所以盒子停下来的位置，不是我们想要的600，而是607

所以解决办法，就是验收、拉回终点、停表：“**拉终停表**”

|  |
| --- |
| 1. var timer = setInterval(function(){ 2. nowleft += 7; 3. if(nowleft > 600){ 4. nowleft = 600; 5. clearInterval(timer); 6. } 7. oDiv.style.left = nowleft + "px"; 8. console.log(nowleft); 9. },20); |

# 三、无缝连续滚动

原理：

页面上是6个图片，编号0、1、2、3、4、5。

复制一倍在后面，长长的火车在移动：



当你赋值的后半段火车的0号头贴到了盒子的左边框的时候，那么就

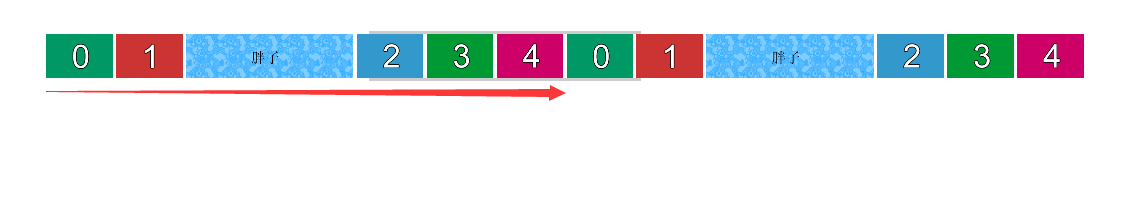


瞬间移动到原点，重新执行动画：



编程不难，但是降低偶合性是一个大问题，专业的前端是要思考这个事情的。

下面的红箭头的长度，就是折返点的数值：



解决方法有两个：

**方法1**：遍历前半部分（复制一倍之前）所有的li，把所有的li的宽度累加，累加之后就是折返点。

我们上午学习的offsetWidth，这个东西不带margin。所以累加的时候，有需要得到计算后的margin十分麻烦。所以我们不考虑方法1。

**方法2**：我们发现，折返点就是假火车的第1张图的offsetLeft值。所以，如果原来的li的个数是lilength，那么假火车的第1张图就是lis[length]

chrome、火狐、IE10开始，JS的执行，不等到图片加载完毕。

[Onload?](https://developer.mozilla.org/zh-CN/docs/Web/API/GlobalEventHandlers/onload)

所以我们的轮播图，所有li都没有宽度，li都是浮动的，浮动的都是收缩的，图有多宽li就有多宽。

所以，你的chrome运行的时候，图片都没有加载呢，js就着急读取offsetLeft值。如何解决？

|  |
| --- |
| 1. window.onload = function(){ 2. alert(oBox.offsetWidth); 3. } |

我们不喜欢写window.onload ， 因为一个页面只能有一个window.onload 。页面也乱。

所以我们要学习新的事件

|  |
| --- |
| 1. image.onload |

当图片加载完毕的时候

# 四、JSON

## 4.1 最简单的JSON示例

[JSON规格？](http://www.ietf.org/rfc/rfc4627.txt) [JSON？](http://www.json.org/)

JSON叫做JavaScript Object Notation， JavaScript对象表示法。由JS大牛Douglas发明。

我们之前学习过数组：

|  |
| --- |
| 1. var arr = [“东风”,”西风”,”南风”,”北风”] |

数组很好用，arr[2] 就是南风。

但是我们发现，数组的下标，只能是阿拉伯数字，不能是我们任意取的。

JSON的示例：

|  |
| --- |
| 1. var obj = { 2. "name":"考拉", 3. "age" : 18, 4. "sex" : "不祥" 5. }; 6. console.log(obj.age); //18 |

语法：

|  |
| --- |
| 1. { 2. "k" : v, 3. "k" : v, 4. "k" : v, 5. "k" : v 6. } |

然后就能用点语法，访问某一个属性。.就是“的”

|  |
| --- |
| 1. obj.age; //obj这个对象的age属性 |

如果不用点语法，也可以使用[]来表示属性，需要注意的是，[]里面是变量：

|  |
| --- |
| 1. var a = "age"; 2. console.log(obj[a]); //18 |

如果不想用变量，必须加引号：

|  |
| --- |
| 1. obj["age"] //18 |

## 4.2 JSON的嵌套

JSON里面的v，可以又是一个JSON

|  |
| --- |
| 1. var obj = { 2. "name":"考拉", 3. "age" : 18, 4. "sex" : "不祥", 5. "shengao" : 193, 6. "peiou" : { 7. "name" : "Angelababy", 8. "age" : 16, 9. "shengao" : 168 10. } 11. }; |

所以，想得到168这个数字：

|  |
| --- |
| 1. obj.peiou.shengao |

Ajax课程大量用到JSON。

## 4.3 JSON项的添加和删除

如果想增加obj里面的项，那么就用点语法赋值：

|  |
| --- |
| 1. var obj = { 2. "name":"考拉", 3. "age" : 18 4. }; 5. obj.sex = "刚变完性"; |

如果想删除某一个属性，使用delete关键字：

|  |
| --- |
| 1. delete obj.age; |

## 4.4 JSON的遍历

for…in语句是专门用来遍历JSON的语法：

|  |
| --- |
| 1. for(var k in obj){ 2. console.log(k + "的值是" + obj[k]); 3. } |

k会依次等于我们的obj里面的属性名，然后在循环语句里面，用obj[k]来读取这个值。

## 4.4 JSON的序列化和反序列化

[序列化JSON.stringify](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify) [反序列化JSON.parse](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse)

* JSON.stringify:将json转换为string
* JSON.parse将string转换为json五、运动框架

为什么要封装一个运动框架呢？因为你不知道运动的复杂。

我们现在想一个情况，一个盒子初始位置是：

left:100px;

top:100px;

现在，我想用3000毫秒时间，让这个盒子运动到

left:700px;

top:250px;

也就是说变化量：

△left = 600px;

△top = 150px;

我们想一下，如果我们现在的动画的间隔是20毫秒，也就是说3000毫秒能执行150次函数。

也就是说：

left的变化步长是600px / 150 = 4px;

而top的变化步长是 150px / 150 = 1px;

牵一发而动全身。动画不直观，所以我们迫切的需要一个牛逼的函数，直接告诉谁运动、终点是什么、总时间：

|  |
| --- |
| 1. animate(oDiv,{"width":700,"height":250},3000); |

性能问题，Chrome浏览器能够支持最小5的interval，每秒200帧；

IE6、7、8、9只能支持最小50的interval，每秒20帧