# Web API

一、Web API介绍

1、API的概念

API（Application Programming Interface,应用程序编程接口）是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力，而又无需访问源码，或理解内部工作机制的细节。

（1）任何开发语言都有自己的API

（2）API的特征输入和输出(I/O)

（3）var max = Math.max(1, 2, 3);

（4）API的使用方法(console.log('adf'))

2、Web API的概念

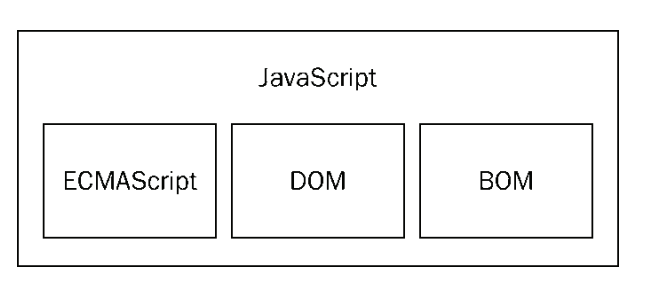
浏览器提供的一套操作浏览器功能和页面元素的API(BOM和DOM)。

此处的Web API**特指浏览器提供的API**(一组方法)，Web API在后面的课程中有其它含义。

掌握常见浏览器提供的API的调用方式

**MDN-Web API**

3、JavaScript的组成



（1）ECMAScript - JavaScript的核心

定义了JavaScript 的语法规范。

JavaScript的核心，描述了语言的基本语法和数据类型，ECMAScript是一套标准，定义了一种语言的标准与具体实现无关。

（2）BOM - 浏览器对象模型

一套操作**浏览器**功能的API。

通过BOM可以操作浏览器窗口，比如：弹出框、控制浏览器跳转、获取分辨率等。

（3）DOM - 文档对象模型

一套操作**页面元素**的API。

DOM可以把HTML看做是文档树，通过DOM提供的API可以对树上的节点进行操作。

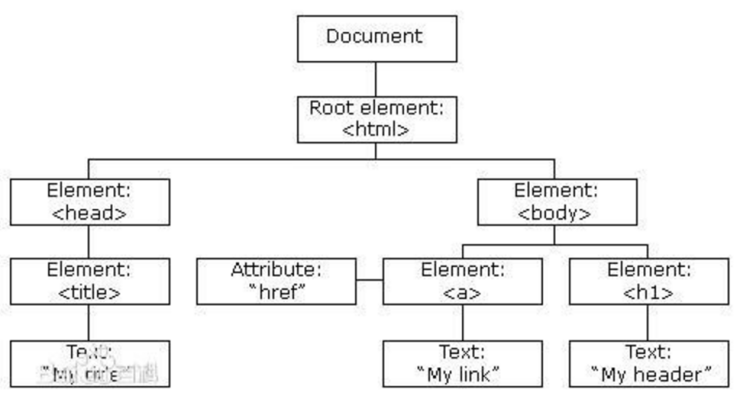
二、DOM

1、DOM的概念

**文档对象模型**（Document Object Model，**简称DOM**），是W3C组织推荐的处理可扩展标记语言的标准编程接口。它是一种与平台和语言无关的应用程序接口(API),它可以动态地访问程序和脚本，更新其内容、结构和www文档的风格(目前，HTML和XML文档是通过说明部分定义的)。文档可以进一步被处理，处理的结果可以加入到当前的页面。DOM是一种基于树的API文档，它要求在处理过程中整个文档都表示在存储器中。

把文档抽象成一个对象。

2、DOM又称为文档树模型。



文档：一个网页可以称为文档

节点：网页中的所有内容都是节点（标签、属性、文本、注释等）

元素：网页中的标签（元素节点）

属性：标签的属性（属性节点）

3、DOM经常进行的操作

（1）获取元素。

（2）对元素进行操作(设置其属性或调用其方法)。

（3）动态创建元素。

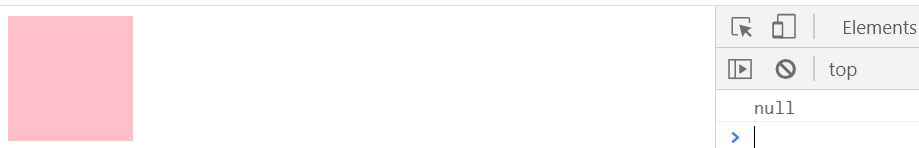
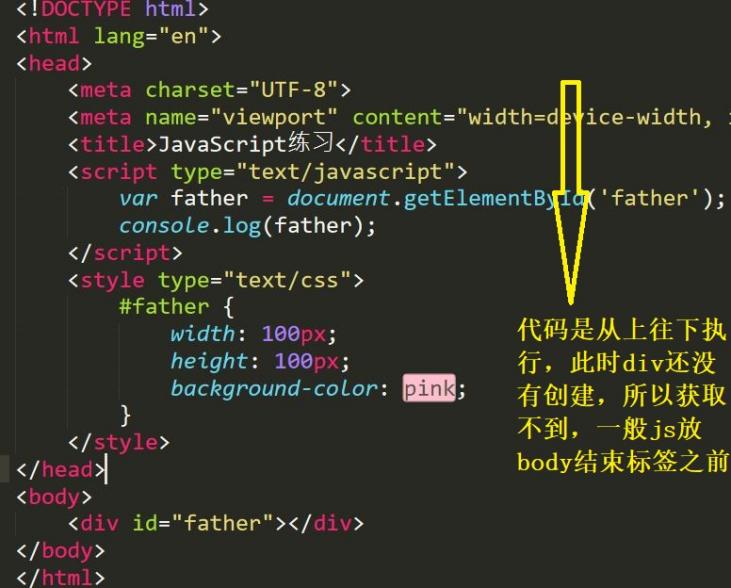
（4）事件(什么时机做相应的操作)。

三、获取页面元素

为什么要获取页面元素

例如：我们想要操作页面上的某部分(显示/隐藏，动画)，需要先获取到该部分对应的元素，才进行后续操作

document代表就是整个文档。获取页面元素的时候，要等元素加载后才能获取到，如果在加载前去获取，是为null。一般把script标签放在body的结束标签的前面。



1、根据id获取元素

|  |
| --- |
| <body>  <div id="father"></div>  <script type="text/javascript">  var father = **document.getElementById('father');**  console.log(father);  console.dir(father);// console.dir()用于专门输出对象，不会对对象做任何的处理。  </script>  </body>  // 获取到的数据类型 HTMLDivElement，对象都是有类型的。  // 返回了一个对象，它的类型是 HTMLDivElement,在\_\_proto\_\_中可以找到。 |
| 12 |
| 注意：  1、由于id名具有唯一性，部分浏览器支持直接使用id名访问元素，但不是标准方式，不推荐使用。  2、只能由document进行调用。 |

2、根据标签名获取元素，返回HTMLcollection 集合、伪数组，跟数组比较类似，但是其本质不是数组。

通过标签获取到的**集合是动态集合**。就是当 我们获取完这个集合对象之后，如果网页上动态的添加了一个div元素，那这个集合会动态发生改变。

|  |
| --- |
| <head>  <script type="text/javascript">  var divs = document.getElementsByTagName('div');  console.log(divs.length);// 输出 0，页面还没有创建div标签  </script>  </head>  <body>  <div id="father"></div>  <div>111</div>  <div>222</div>  <script type="text/javascript">  **console.log(divs.length); // 输出3**  /\*var divs = document.getElementsByTagName('div');  console.log(divs);  for (var i = 0; i < divs.length; i++) {  console.log(divs[i]);// divs 这个集合里边的元素是一个HTMLDivElement类型的对象。  }\*/  </script>  </body> |
| 1  2 |

（1）获取子元素

|  |
| --- |
| <body>  <div id="father">  <div>11</div>  <div>111</div>  </div>  <script type="text/javascript">  var father = document.getElementById('father');  var divs = father.getElementsByTagName('div');  console.log(divs);  </script>  </body> |
| 1 |

3、根据name属性获取元素，不太推荐使用

|  |
| --- |
| <body>  <div name="father"></div>  <div id = father></div>  <input type="text" name="father">  <script type="text/javascript">  var elements = document.getElementsByName('father');  console.log(elements);  </script>  </body> |
| getElementsByName 在不同的浏览器其中工作方式不同。在IE和Opera中，在谷歌中没有这个问题。  getElementsByName() 方法还会返回那些 id 为指定值的元素（就是跟name同名）。所以你要小心使用该方法，最好不要为元素的 name 和 id 赋予相同的值。 IE 和 Edge 都返回一个 HTMLCollection, 而不是NodeList 。  截取MDN |
| 1 |

4、根据类名获取元素

有浏览器兼容的问题，IE9以后才支持的，用不用要根据用户需求，如果需要兼容到IE8以前的浏览器，那么我们就不可以用这个方法。

返回一个包含了所有指定类名的子元素的类数组对象。类名通过空格分隔

|  |
| --- |
| <body>  <div class="father"></div>  <div class = father></div>  <p class="father"></p>  <script type="text/javascript">  var elements = document.getElementsByClassName('father');  console.log(elements);  </script>  </body> |
| 1 |

5、根据选择器获取元素

（1）query:查询，querySelector()与指定选择器或选择器组**匹配的第一个 HTMLElement对象**。 如果找不到匹配项，则返回null。

1. querySelectorAll()跟querySelector()的用法一样，不同点就是它会返回所有找到的选择器。返回的对象是 NodeList 。

把选择器写复杂了就可以找到具体的元素。具有浏览器兼容性问题，IE8以后才支持。在移动端浏览器上就没有问题。就可以只用他们俩，在PC上会有小问题。

|  |
| --- |
| <body>  <div class="father"></div>  <div class = "father"></div>  <p id="hi"></p>  <span class="father"></span>  <script type="text/javascript">  var element = document.**querySelector**('.father');  console.log(element);  var element = document.**querySelector**('#hi');  console.log(element);  var elements = document.**querySelectorAll**('div.father');// 写并集选择器  console.log(elements);  </script>  </body> |
| 1 |

6、变量名称不能为top

top是window自带的一个属性，此属性默认是window对象，变量名字不能为top，它是一个只读的。

四、事件

事件：触发-响应机制，当什么时候做什么事情。绝大部分元素都能设置事件。

（1）获取按钮

（2）给按钮注册事件

1、事件三要素

（1）事件源:触发(被)事件的元素，谁触发了事件

（2）事件名称: click 点击事件

（3）事件处理程序:事件触发后要执行的代码(函数形式)

2、事件的基本使用

|  |
| --- |
| <body>  <input type="button" id='btn' value="点击按钮">  <div id='div'>123</div>  <script type="text/javascript">  var btn = document.getElementById('btn');  btn.onclick = function () {  alert('按钮点击');  };  var div = document.getElementById('div');  div.onclick = function () {  alert('盒子点击');  }  </script>  </body> |
| 点击按钮切换图片  <body>  <input type="button" id='btn' value="点击切换图片">  <img src="images/h.jpg" width="220" height="220" id="img">  <script type="text/javascript">  var btn = document.getElementById('btn');  var img = document.getElementById('img');// 这句话可以放到onclick中去，点击的时候再去找，用户可能不点击。  var flag = false;// true 表示是h.jpg。false表示另外一张  btn.onclick = function () {  if (flag === true) {  img.src = 'images/h.jpg';  flag = false;  } else {  img.src = 'images/kmr.jpeg';  flag = true;  }  };  </script>  </body> |

五、属性操作

1、非表单元素的属性

一般表单元素就是可以跟用户进行交互，如文本框，按钮等等，而div、p只是把数据、文本展示出来。

DOM对应着页面上的一些标签，DOM是在内存中的，标签是直接写在页面上的。

一般我们获得DOM对象或者元素的属性，是对应着标签的属性，但是也有例外。

href、title、id、src、className。

DOM对象的对应的标签内的**class属性的名字**叫做**className**，因为class在js中是关键字，关键字不可以作为变量或者属性的名字。

|  |
| --- |
| var link = document.getElementById('link');  console.log(link.href);  console.log(link.title);  var pic = document.getElementById('pic');  console.log(pic.src); |

2、点击按钮显示隐藏div

（1）函数中 --- this 指向Window

（2）方法中 --- this 指向的是这个方法的所属对象

（3）构造函数中 --- this 就是构造函数创建的对象

（4）事件处理函数中的 this --- 指向事件源，谁调用的该事件就指向谁。

|  |
| --- |
| <style type="text/css">  div {  height: 200px;  width: 200px;  background-color: pink;  }  .showBox {  display: block;  }  .hiddenBox {  display: none;  }  </style> |
| <script type="text/javascript">  var btn = document.getElementById('btn');  var isShow = false;  btn.onclick =function () {  var box = document.getElementById('box');  // var isShow = false;// 这个变量是在局部变量中，等等事件处理函数进行的时候，它又等false，要设置成全局变量  if (isShow) {  // btn.value = '显示';  this.value = '显示';  box.className = 'showBox';  //box.style.display = 'block';  isShow = false;  } else {  // btn.value = '隐藏';  this.value = '隐藏';  box.className = 'hiddenBox';  //box.style.display = 'none';  isShow = true;  }  }  </script> |

3、美女相册案例

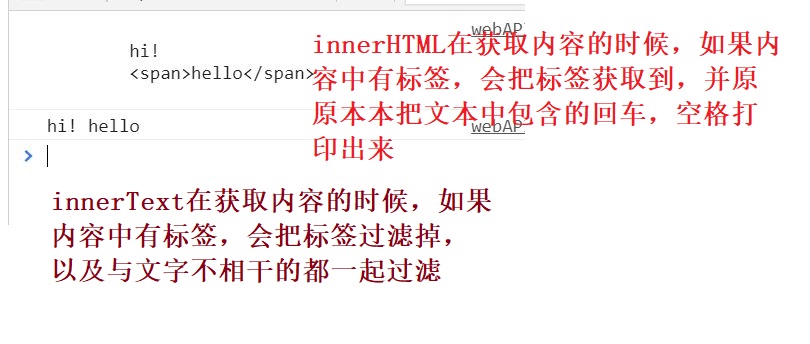
|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>WebAPI练习</title>  <style type="text/css">  #photo a {  float: left;  }  .clearfix:after {  content: ".";  visibility: hidden;  display: block;  height: 0;  clear: both;  }  .clearfix {  \*zoom: 1;  }  </style>  </head>  <body>  <h2>相册</h2>  <div class="clearfix" id="photo">  <a href="images/1.jpg" title="NO.1"><img src="images/1-small.jpg" width="100"></a>  <a href="images/2.jpg" title="NO.2"><img src="images/2-small.jpg" width="100"></a>  <a href="images/3.jpg" title="NO.3"><img src="images/3-small.jpg" width="100"></a>  <a href="images/4.jpg" title="NO.4"><img src="images/4-small.jpg" width="100"></a>  </div>  <div><img id="images" src="images/placeholder.png" width="400" height="250"></div>  <p id="choose">选择一个图片</p>  <script type="text/javascript">  var photo = document.getElementById('photo');  var alinks = photo.getElementsByTagName('a');//要获取多个a标签对象  for (var i = 0; i < alinks.length; i++) {  var link = alinks[i];  link.onclick = function () {  var images = document.getElementById('images');  //images.src = link.href;// 注册这个事件时，函数并没有执行，当点击之后，系统才帮我们自动调用这个函数，此时 link已经是最后一个a标签对象了。  images.src = this.href;// this指向了事件源  var choose = document.getElementById('choose');  choose.innerHTML = this.title;  //console.dir(choose);  return false;// 这句话让a的href链接失效  }  }  </script>  </body>  </html> |
| 12 |
| 1、a标签还能跳转到图片标签，一般图片要使用占空间比较小的图片。  2、我们注册事件的时候，这个事件并不会执行。是需要我们去点击之后系统才会自动执行，去调用，当循环结束的时候，link变量是指向了最后一个a标签变量，它是全局变量。 |

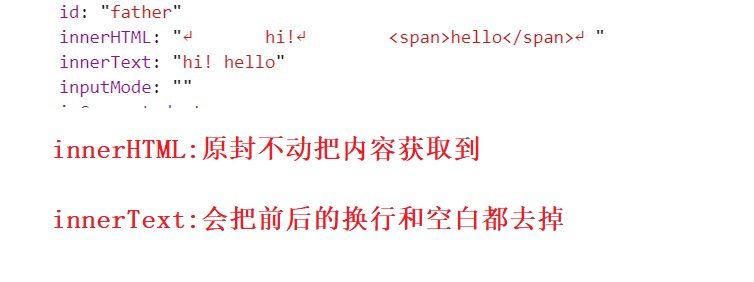
4、innerHTML和innerText

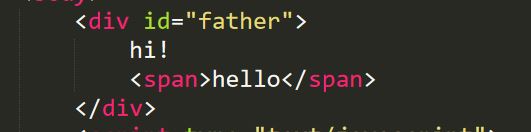
获取开始标签和结束标签之间的内容。必须要有开始和结束标签。HTML不是一个单词，他是四个单词。

|  |
| --- |
| <div id="father">  hi!  <span>hello</span>  </div>  <script type="text/javascript">  var father = document.getElementById('father');  console.log(father.innerHTML);  console.log(father.innerText);  console.dir(father);  father.innerHTML = '';//清空内容  // father.innerHTML = '<strong>黑马</strong>123';  father.innerText = '<strong>黑马</strong>123';  </script> |

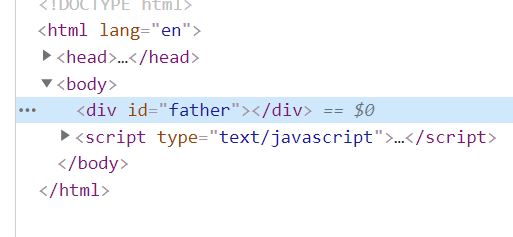
（1）innerHTML、innerText打印



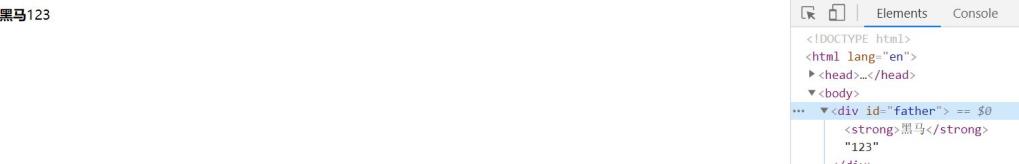




（2）清空innerHTML、innerText



（3）设置innerHTML、innerText



通过innerHTML，设置内容，如果内容中带有标签，会以**HTML的方式来解析**。



通过innerText，设置内容，如果内容中带标签，在网页上会把标签直接显示出来，其实它使用的是**转义字符**才能直接显示。

（4）innerText与textContent

两个都获取内部文本。

但是各个浏览器的兼容性不同。

谷歌：两个都支持

新版火狐：两个都支持

旧版火狐：只支持textContent

新版IE：IE9以上支持两个

旧版IE：以下只支持innerText

此时可以进行兼容性处理，在后边有很多库会帮我们处理这个问题。

|  |
| --- |
| 1、浏览器是否支持元素的某个属性  可以通过 typeof 关键字来判断，当属性存在的时候，返回的是该属性的值；当属性不存在的时候，返回的是undefined。  <body>  <p id="p"><b>nihiao</b></p>  <script type="text/javascript">  var p = document.getElementById('p');  console.log(typeof p.textContent);  console.log(typeof p.a);  </script>  </body>  1 |
| <body>  <p id="p"><b>nihiao</b></p>  <script type="text/javascript">  var p = document.getElementById('p');  console.log(getinnerText(p));  function getinnerText(element) {  //判断当前浏览器是否支持元素的innerText属性，支持的话就返回element.innerText获取内容。  // 如果不支持就使用 element.textContent获取内容。  if (typeof element.innerText === 'string') {  return element.innerText;  } else {  return element.textContent;  }  }  </script>  </body>  一般不需要这么做，库会帮我们去封装的。 |

1. HTML转义符

如果要在页面上显示一个尖括号、或者双引号，只能通过**转义符**，如果直接写符号，浏览器会对它们进行解析。

|  |
| --- |
| " &quot;  ' &apos;  & &amp;  < &lt; // less than 小于  > &gt; // greater than 大于  空格 &nbsp;  © &copy; |

6、表单元素属性

value 用于大部分表单元素的内容获取(option除外)

type 可以获取input标签的类型(输入框或复选框等)

下边的这三个用法稍微有区别，html就这三个。

当html中的标签属性，只有一个值的时候，DOM中对应的元素的属性值是布尔类型。

disabled 禁用属性，它的值就只有disabled（disabled = "disabled"）;只是禁用了普通用户去文本框输入，开发人员能通过js代码方式去修改文本框的值。

checked 复选框选中属性。

selected 下拉菜单选中属性。

|  |
| --- |
| <body>  <input type="button" value="禁用文本框输入" id="noInput">  <input type="button" value="更改文本" id="change">  <input type="text" id="text">  <script type="text/javascript">  var noInput = document.getElementById('noInput');  noInput.onclick = function () {  var text = document.getElementById('text');  console.log(text.disabled);  console.log(text.type);  }  </script>  </body>  1 |
| <body>  <input type="button" value="禁用文本框输入" id="noInput">  <input type="button" value="更改文本" id="change">  <input type="text" id="text" value="123">  <script type="text/javascript">  var noInput = document.getElementById('noInput');  noInput.onclick = function () {  var text = document.getElementById('text');  text.disabled = true;  }  var change = document.getElementById('change');  change.onclick = function () {  var text = document.getElementById('text');  text.value = 'hello world';  }  </script>  </body> |

（1）文本框赋值案例

要求：

1.当页面加载完毕，给所有的文本框赋值

2.当点击按钮的时候，获取所有的文本框的值，并使用|分割输出

|  |
| --- |
| <body>  <input type="text">  <input type="text">  <input type="text">  <input type="text">  <input type="text">  <input type="text">  <input type="text">  <input type="text">  <input type="text">  <input type="text">  <input type="button" value="获取文本" id="btn">  <p id="p">输出内容</p>  <script type="text/javascript">  var input\_elements = document.getElementsByTagName('input');  var input;    /\*for (var i = 0; i < input\_elements.length-1; i++) {  input = input\_elements[i];  input.value = i + 1;  }这段代码优化一下  \*/  for (var i = 0; i < input\_elements.length; i++) {  input = input\_elements[i];  if (input.type === 'text') {  input.value = i + 1;  }  }  // input = input\_elements[input\_elements.length-1];其实按钮可以通过 id 来进行获取，这就不用靠位置  var btn = document.getElementById('btn');  btn.onclick = function () {  var array = [];//相当于是每次进来创建一块新的内存区域，它是局部变量  for (var i = 0; i < input\_elements.length; i++) {  input = input\_elements[i];  if (input.type === 'text') {  array.push(input.value);  }  }  //console.log(array.join('|'));  var p = document.getElementById('p');  p.innerText = array .join('|');  }  </script>  </body>  这种方法使用的是用数组来拼接最后的结果，不会有另外的空间重新开辟，会在数组后添加空间。 |
| var btn = document.getElementById('btn');  btn.onclick = function () {  **var str = '';**  for (var i = 0; i < input\_elements.length; i++) {  input = input\_elements[i];  if (input.type === 'text') {  **str += input.value + '|'; //**当重新给str赋值的时候，常量'abc'不会被修改，依然在内存中，不确定浏览器什么时候才会将他回收。  }  }  **str = str.substr(0,str.length - 1);**  var p = document.getElementById('p');  **p.innerText = str;**  } |
| 1 |

如果在过程中涉及到字符串的拼接问题，要考虑到程序的性能，字符串的拼接是比较耗费时间空间的。因为字符串是不可变的。

（2）检测用户名是否是3-6位，密码是否是6-8位，如果不满足要求高亮显示文本框

|  |
| --- |
| <body>    <input type="text" placeholder="请输入账号"><br />  <input type="password" placeholder="请输入密码"><br />  <input type="button" value="登录" id="login">  <script type="text/javascript">  var login = document.getElementById('login');  login.onclick = function () {  var inputs = document.getElementsByTagName('input');  var user='',password='';  user = inputs[0].value;  password = inputs[1].value;  if (user.length < 3 || user.length > 6) {  inputs[0].className = 'yellow';  **return; //格式有问题下边就不用登录了**  } else {  /\*inputs[0].className = 'clearColor';\*/  **inputs[0].className = '';// 可以用空来表示，把样式去掉**  }  password = inputs[1].value;  if (password.length < 6 || password.length > 8) {  inputs[1].className = 'yellow';  return; //格式有问题下边就不用登录了  } else {  /\*inputs[0].className = 'clearColor';\*/  inputs[1].className = '';// 可以用空来表示，把样式去掉  }  console.log('执行登录');  }  </script>  </body>  1 |

（2）设置下拉框中的选中项

|  |
| --- |
| <body>  <input type="button" value="随机选择" id="btn">  <select id="selFruit">  <option value="1">西瓜</option>  <option value="2">苹果</option>  <option value="3">葡萄</option>  <option value="4">芒果</option>  <option value="5">杨梅</option>  </select>  <script type="text/javascript">  function getRandom(min, max) {  return Math.floor(Math.random()\*(max - min + 1) + min);  }  var btn = document.getElementById('btn');  btn.onclick = function () {  var selFruit = document.getElementById('selFruit');  // 页面中可能存在着多个其他option，我们只获取水果的option  var options = selFruit.getElementsByTagName('option');  //options[getRandom(0,4)].selected = true;  options[getRandom(0,options.length-1)].selected = true;  }  </script>  </body>  有个疑问，就是这样设置option 的selected属性之后，不是有多个被选中吗，为什么不冲突 |

（3）搜索文本框

点击文本框的时候（获得焦点），把原有的灰色文字清除，同时输入文字的话，字体颜色是为黑色，然后点击非文本框区域的时候（失去焦点），恢复到原来的样子。

当文本框获得焦点，如果文本框里的内容是“请输入文本”，清空文本框，并且让字体颜色变为黑色。

当文本框失去焦点，如果文本框里的内容为空，还原文本框里的文字，并且让字体颜色变为灰色。

|  |
| --- |
| <body>  <input type="text" value="请输入文本" class="gray" id="txtSearch">  <button type="button" id="btn">搜索</button>  <script type="text/javascript">  var txtSearch = document.getElementById('txtSearch');  txtSearch.onfocus = function () {  if (this.value === '请输入文本') {  this.value = '';  this.className = '';  }  }  txtSearch.onblur = function () {  if (this.value === '') {// 可以换成 this.value.length === 0 使用长度来判断，比较常用  this.value = '请输入文本';  this.className = 'gray';  }  }  </script>  </body>  这段代码会有问题，当用户输入“请输入文本”这几个字的时候，当再次点击的时候会被清空掉。这里是需要进行处理的。  视频的处理方式是，当用户输入这几个字后，失去焦点时，把颜色改成灰色  if (this.value === ''|| this.value === '请输入文本')  个人感觉直接用placeholder，基本能达到效果。网上有模拟实现placeholder的效果，可以进行模仿。大概是添加一个span，定位上去 |

（4）全选反选

详情以及代码看example目录下的文件。

7、自定义属性操作

一般用来命名一些在数据库对应的名字。

以下是一些方法。

（1）getAttribute() 获取标签行内属性，也能获取自定义标签属性的值。

（2）setAttribute() 设置标签行内属性，能添加自定义属性。

（3）removeAttribute() 移除标签行内属性

与element.属性的区别: 上述三个方法用于获取任意的行内属性。

|  |
| --- |
| <body>  <div id="box" class="test" age="18">hello world</div>  <script type="text/javascript">  var box = document.getElementById('box');  console.log(box.className);  console.log(box.id);  // 自定义属性，不能通过使用下面的方式获取到，box元素对应的div标签的自定义属性。  console.log(box.age);  **console.log(box.getAttribute('id'));**  console.log(box.getAttribute('age'));  box.setAttribute('class','changeBysetAttribute');  box.setAttribute('age',22);  console.log(box.className);  console.log(box.getAttribute('age'));  box.removeAttribute('id');  console.log(box.id);  </script>  </body> |
| 12  3 |

六、样式操作（没有视频）

1、使用style方式设置的样式显示在标签行内

|  |
| --- |
| var box = document.getElementById('box');  box.style.width = '100px';  box.style.height = '100px';  box.style.backgroundColor = 'red';  操作样式的时候，使用类样式 使用style？  // 当设置多个样式属性的时候使用类样式方便。  // 当设置样式属性比较少的时候使用style比较方便。 |

注意：

// 样式属性：background-color

// DOM中style的属性：backgroundColor

通过样式属性设置宽高、位置的属性类型是字符串，需要加上px。当设置宽度和高度的时候必须带单位，如果不带单位，有错误。

（1）获取属性值

style.left获取的是**标签中style属性设置的样式属性的值**，如果标签中的style没有设置该样式属性，我们获取到的是空字符串。

如果是在CSS样式中的 left 则是获取不到的。得通过其他的方法去获取。

当我们给样式属性设置style.left = ‘10px10px’，这个值是非法值，浏览器会帮我们去过滤掉非法值，不给属性赋值。

（2）attributes属性，能直接找到元素所有的样式属性。

|  |
| --- |
| <div id="father" style="color: red;"></div>  var father = document.getElementById('father');  console.dir(father.attributes);  2 |

（3）如何获取网页元素的CSS样式值、要求浏览器兼容性问题

|  |
| --- |
| if(window.getComputedStyle){  return window.getComputedStyle(ele,null);//标准浏览器  }else{  ele.currentStyle;//currentStyle是IE的属性  } |

2、类名操作

修改标签的className属性相当于直接修改标签的类名。

|  |
| --- |
| var box = document.getElementById('box');  box.**className** = 'show'; |

3、案例

（1）图片切换二维码案例

|  |
| --- |
| <body>  <div class="box" id="boxid">  <img src="images/456.png" class="hidden" id="img">  </div>  <script type="text/javascript">  var box = document.getElementById('boxid');  box.onmouseover = function () {  var img = document.getElementById('img');  img.className = 'showPic';  }  box.onmouseout = function () {  var img = document.getElementById('img');  img.className = 'hidden';  }  </script>  </body> |
| <style type="text/css">  .box {  width: 50px;  height: 50px;  position: fixed;  right: 0;  top: 50%;  transform: translate(0,-50%);  background: url(images/bgs.png) no-repeat -159px -51px;  }  .box img {  position: absolute;  left: -149px;  top: -50px;  }  .showPic {  display: block;  }  .hidden {  display: none;  }  </style> |

（2）当前输入的文本框高亮显示

|  |
| --- |
| <body>  <input type="text"><br>  <input type="text"><br>  <input type="text"><br>  <input type="text"><br>  <input type="text"><br>  <input type="text"><br>  <input type="text"><br>  <input type="text"><br>  <input type="button" value=" 提交 ">  <script type="text/javascript">  var textElements = document.getElementsByTagName('input');  for (var i = 0; i < textElements.length; i++) {  var text = textElements[i];  if (text.type !== 'text') {  continue;  }  text.onfocus = function () {  for (var i = 0; i < textElements.length; i++) {  var text = textElements[i];  if (text.type !== 'text') {  continue;  }  text.style.backgroundColor = '';  }  this.style.backgroundColor = '#ccc';  }  }  </script> |

（3）设置div盒子的大小和位置

|  |
| --- |
| #box {  width: 100px;  height: 100px;  background-color: pink;  } |
| <body>  <input type="button" value="点击移动位置" id="btn">  <div id="box"></div>  <script type="text/javascript">  var btn = document.getElementById('btn');  btn.onclick = function () {  var box = document.getElementById('box');  box.style.position = 'absolute';  box.style.left = '200px';  box.style.top = '200px';  box.style.width = "200px";  box.style.height = "200px";  console.log(box.style);  }  </script>  </body>  1 |

（4）隔行变色和高亮显示

|  |
| --- |
| <script type="text/javascript">  var fruitList = document.getElementById('fruitList');  var bg;  var lis = fruitList.getElementsByTagName('li');  for (var i = 0; i < lis.length; i++) {  var li = lis[i];  if (i % 2 === 0) {  li.style.backgroundColor = 'skyblue';  } else {  li.style.backgroundColor = 'red';  }  li.onmouseover = function () {// 只是注册事件，并没有执行，如果写出li，那么当点击的时候，这个li停在了第四个,this就是指向了当前的事件源  **//li.style.backgroundColor = 'yellow';**  bg = this.style.backgroundColor;  **this.style.backgroundColor = 'yellow';**  }  li.onmouseout = function () {  **this.style.backgroundColor = bg;**  }  }  </script> |
|  |

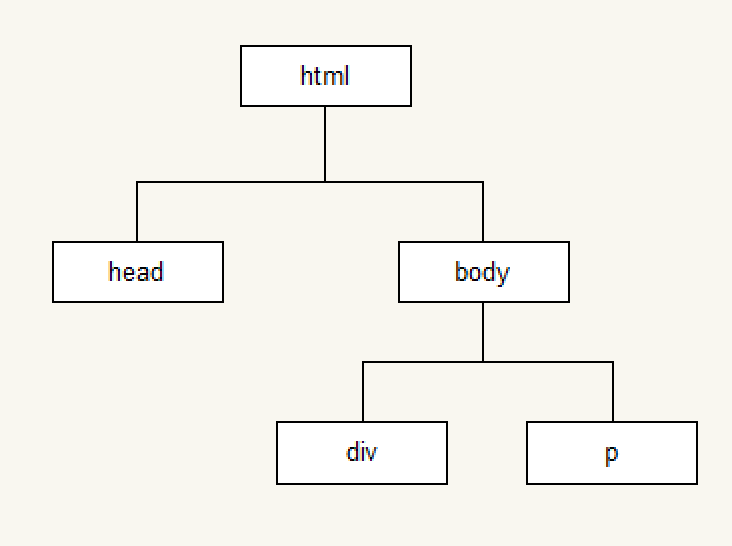
（5）tab切换

|  |
| --- |
| <body>  <div class="box">  <div class="hd" id="hdid">  <span class="current">体育</span>  <span>新闻</span>  <span>娱乐</span>  <span>综合</span>  </div>  <div class="bd" id="bdid">  <div class="current">我是体育模块</div>  <div>我是新闻模块</div>  <div>我是娱乐模块</div>  <div>我是综合模块</div>  </div>  </div>  <script type="text/javascript">  var hd = document.getElementById('hdid');  var spans = hd.getElementsByTagName('span');  var bd = document.getElementById('bdid');  var divs = bd.getElementsByTagName('div');  for (var i = 0; i < spans.length; i++) {  var span = spans[i];  span.setAttribute('index',i);  span.onmouseover = function () {  for (var i = 0; i < spans.length; i++) {  spans[i].className = '';  }  this.className = 'current';  for (var i = 0; i < divs.length; i++) {  divs[i].className = '';  }  //divs[this.getAttribute('index')].className = 'current';  // 先转换成整数  var index = parseInt(this.getAttribute('index'));  divs[index].className = 'current';  }  }  </script>  </body> |
| <style type="text/css">  .box {  width: 300px;  height: 300px;  border: 1px solid #000;  margin: 100px auto;  }  .hd span {  display: inline-block;  width: 70px;  line-height: 70px;  background-color: pink;  text-align: center;  cursor: pointer;  }  .box .hd .current {  background-color: orange;  }  .bd {  height: 230px;  background-color: skyblue;  }  .bd div {  display: none;  }  .box .bd .current {  display: block;  }  </style> |

4、总结

|  |
| --- |
| DOM 文档对象模拟    常见的DOM操作  获取元素 getElementById() getElementsByTagName()  给元素注册事件 onclick onmouseover onmouseout onfocus onblur  操作元素的属性  非表单元素 href title src alt等  表单元素 type value checked disabled selected  公共属性 id className style  样式操作 className style  自定义属性 setAttribute() getAttribute() removeAttribute() |

七、模拟文档树结构



DOM是一个树形的结构，所以叫做文档树，树里面的所有内容都是节点。

<div> 是一个**元素**节点

<div id="box"> id 就是一个**属性**节点

<div id="box">hello<box> hello是一个**文本**节点

<!-- 这是一个注释 --> 这是一个**注释**节点。

节点属性

1、nodeType 节点的类型，通过console.dir 能找到节点对象的这些值

- 1 元素节点

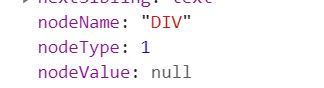
- 2 属性节点

- 3 文本节点

- nodeName 节点的名称(标签名称)

- nodeValue 节点值

- 元素节点的nodeValue始终是null



2、创建一些具有相同属性的对象。构造函数。获取对象没有的属性，属性的值是undefined。

|  |
| --- |
| <body>  <div class="box" id="boxid"></div>  <script type="text/javascript">  var box = document.getElementById('boxid');  console.dir(box);  function Node(options) { // 这里可能要传入分成多的参数，而且参数的是无序，我们无法知道参数的对应关系，所以可以通过对象的方式来解决  /\*if (options.className) {  this.className = options.className;  } else {  this.className = '';  }  this.className = options.className ? options.className : '';  这两种写法代码数量都比较多  \*/  /\*设置属性的默认值\*/  this.classNames = options.classNames || '';  this.id = options.id || '';  /\*跟节点相关的属性\*/  // 标签的名称  this.nodeName = options.nodeName || '';  // 节点类型 元素节点：1 属性节点：2 文字节点：3  this.nodeType = options.nodeType || 1;  this.nodeValue = options.nodeValue || null;  // 孩子节点  this.children = options.children || [];  }  var html = new Node({  nodeName: 'html'  });  var head = new Node({  nodeName: 'head'  })  var body = new Node({  nodeName: 'body'  })  html.children.push(head);  html.children.push(body);  var div = new Node({  nodeName: 'div'  })  var p = new Node({  nodeName: 'p'  })  body.children.push(div);  body.children.push(p);  console.dir(html);  </script>  </body> |
| 2 |
| DOM文档就是通过这种类似的机制去创建DOM对象。系统自动帮我们去完成这些事情了。  在运行的时候，浏览器内部维护了一棵DOM树。  1、深刻理解DOM。  2、了解节点相关的属性，nodeName、nodeType、nodeValue。  3、了解节点的层次结构，父节点、子节点、兄弟节点等等。 |

3、节点层级

（1）childNodes和children

**children是会实时更新。**

childNodes和children的区别，childNodes获取的是**子节点**，children获取的是**子元素**

|  |
| --- |
| 子节点，在html文件的书写中，**回车换行也是一个文本节点**。  12 |
| <body>  <div id="box">盒子  <span>hello</span>  <p>hi</p>  <!-- 这是注释 -->  </div>  <script type="text/javascript">  var box = document.getElementById('box');  console.dir(box);  console.dir(box.childNodes);  </script>  </body> |

（2）parentNode，父节点就只有一个

|  |
| --- |
| <body>  <div id="box">盒子  <span>hello</span>  <p>hi</p>  <!-- 这是注释 -->  </div>  <script type="text/javascript">  var box = document.getElementById('box');  // 从子节点中获取子元素节点  /\* 这样能够获得子元素节点，但是比较麻烦，可以在 MDN 中找到其他相关方法，也可以模仿里边好的写法，可以先判断是否有孩子节点  var nodes = box.childNodes;  for (var i = 0; i < nodes.length; i++) {  if (nodes[i].nodeType !== 1) {  continue;  }  console.log(nodes[i]);  }\*/  var elements = box.children;// 获得元素对象数组  console.log(elements);  var parentOfBox = box.parentNode;  console.log(parentOfBox);  </script>  </body>  1 |

（3）隔行变色

|  |
| --- |
| <body>  <div id="box">  <ul id="fruit">  <li>西瓜</li>  <li>橙子</li>  <li>葡萄</li>  <li>杨梅</li>  <li>芒果</li>  </ul>  </div>  <script type="text/javascript">  var fruit = document.getElementById('fruit');  //var lis = fruit.getElementsByTagName('li');  //如果是这种情况就不适合用getElementsByTagName  /\*<div>  <span></span>  <p></p>  </div>\*/  if (fruit.hasChildNodes()) {**//使用方法时要到MDN查看下是否有兼容性问题**  var lis = fruit.children;  for (var i = 0; i < lis.length; i++) {  if (i % 2 === 0) {  lis[i].style.backgroundColor = 'red';  } else {  lis[i].style.backgroundColor = 'yellow';  }  }  }  </script>  </body>  1 |

（4）firstChild和firstElementChild

它们都是只读的。firstElementChild具有兼容性的问题。可以捅MDN提供处理适用于 IE8、IE9 和 Safari的方法。就是用一些大家都支持的方法，再通过一定逻辑处理，达到想要的效果。

|  |
| --- |
| <body>  <div id="box">  <p>1111</p>  <span>2222</span>  <ul>  <li>33</li>  <li>44</li>  </ul>  </div>  <script type="text/javascript">  var box = document.getElementById('box');  // 获取第一个子节点  console.log(box.firstChild);// 输出文本节点  // 获取第一个子元素  console.log(box.firstElementChild);// 他是具有兼容性的问题，查 MDN  // 处理浏览器的兼容性  function getFirstElementChild(node) {  var node, nodes = node.childNodes, **i = 0**;  while (node = nodes[**i++**]) {  if (node.nodeType === 1) {  return node;  }  }  return null;  }  console.log(getFirstElementChild(box));  // 获取最后的子节点  console.log(box.lastChild);  // 获取最后的子元素  console.log(box.lastElementChild);  // 处理浏览器兼容问题  function getLastElementChild(node) {  var node, nodes = node.childNodes, **i = nodes.length - 1**;  while (node = nodes[**i--**]) {  if (node.nodeType === 1) {  return node;  }  }  return null;  }  console.log(getLastElementChild(box));  </script>  </body> |

（5）菜单

|  |
| --- |
| 1、<a href="javascript:void(0)">葡萄</a>  （1）可以用a标签代替按钮，经过的时候会变成小手。  （2）冒号前边是协议，如 http://www.baidu.com，前边的是javascript，告诉浏览器要执行js代码。 |
| 1. href = “javascript:void(0)”，href切换浏览器中的地址   浏览器的地址栏也是可以执行javascript代码的 void(0)就是什么都不干。 |
| 3、void 是运算符  执行void 后面的表达式，并始终返回undefined。  如果写href = “javascript:0”，会在页面输出 0 。谷歌浏览器是做过处理，所以没有出现  href = “javascript:’0’”，谷歌浏览器会输出0。  1 |

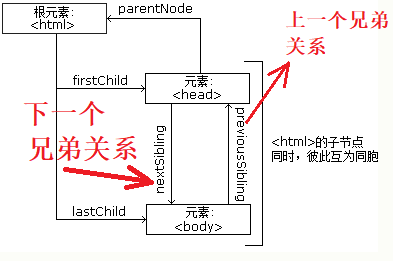
虽然这么做是可行的，但是利用JavaScript：伪协议来执行javascript代码是不推荐的，它会去计算一下，做了一些事情，推荐的做法是为链接元素绑定click事件，return false取消后续内容的执行。

|  |
| --- |
| <script type="text/javascript" src="index.js"></script>  <style type="text/css">  ul {  list-style: none;  }  a {  text-decoration: none;  }  #box ul li {  width: 100px;  height: 50px;  background-color: pink;  display: inline-block;  text-align: center;  line-height: 50px;  }  #box ul li a {  width: 100px;  height: 50px;  display: block;  }  #box ul .current {  background-color: orange;  }  </style> |
| <body>  <div id="box">  <ul><!-- 通常只是给父盒子一个id属性 -->  <li class="current"><a href="**javascript:void(0)**">西瓜</a></li>  <li><a href="javascript:void(0)">葡萄</a></li>  <li><a href="javascript:void(0)">杨梅</a></li>  <li><a href="javascript:void(0)">芒果</a></li>  <li><a href="javascript:void(0)">西红柿</a></li>  </ul>  </div>  <script type="text/javascript">  var box = document.getElementById('box');  var ul = getFirstElementChild(box);  var lis = ul.children;  for (var i = 0; i < lis.length; i++) {  var link = getFirstElementChild(lis[i]);  link.onclick = **linkClick**;// 这里不能带上括号，否则就变成了调用函数，返回undefined，点击后没有效果  }  function linkClick() {  for (var i = 0; i < lis.length; i++) {  var li = lis[i];  li.className = '';  }  // this指向的是当前点击事件源，就是点击了哪个a标签  this.parentNode.className = 'current';  // 取消链接后续内容的执行  return false;  }  </script>  </body> |
| 点击函数如果都是一样的话，可以在外边定义成一个函数，而不是循环写整个函数体。  函数带了小括号的话，是函数调用，赋值的话，函数名就可以。 |

（6）nextSibling和previousSibling获取的是节点，

获取元素对应的属性是nextElementSibling和previousElementSibling

获取的是元素nextElementSibling和previousElementSibling有兼容性问题，IE9以后才支持



|  |
| --- |
| <body>  <div>第一个盒子</div>  <div>第二个盒子</div>  <div id="box">第三个盒子</div>  <div>第四个盒子</div>  <div>第五个盒子</div>  <script type="text/javascript">  var box = document.getElementById('box');  console.log(box.nextSibling);  console.log(box.previousSibling);  console.log(box.nextElementSibling);  console.log(box.previousElementSibling);  function getNextElementSibling(element) {  var el = element;  while (el = el.nextSibling) {// 这里写成了el = element.nextSibling,造成了死循环  if (el.nodeType === 1) {  return el;  }  }  return null;  }  console.log(getNextElementSibling(box));  </script>  </body>  1 |

1. 创建元素的三种方式

1、document.write()

使用document.write()会把之前的整个页面都覆盖掉，head标签里的内容都会不见。它局限于只能在页面加载的时候去输出。不能在事件中使用，会覆盖掉。用的情况比较少。

一般是用在网页中出现那个客服小窗口那里，一般去购买的，别人直接把js挂上去就能使用。

|  |
| --- |
| <body>  <input type="button" value="按钮生成段落" id="btn">  <script type="text/javascript">  var btn = document.getElementById('btn');  btn.onclick = function () {  document.write('hello <p>hi</p>');  }  </script>  </body>  12 |

2、innerHTML

用途：创建简单的元素，并没有携带事件。如果带事件，推荐使用createElement

网页上的所有东西都是画出来的。每次设置innerHTML的时候，会触发浏览器DOM树会进行重绘，会导致效率问题，需要斤进行优化。

如果用字符串先把内容进行拼接，最后在使用一次innerHTML，这样使得innerHTML就只绘制一次，但是字符串拼接，重新开辟内存，需要耗费时间，因为字符串不可变。

数组只会内存的后面追加空间，而不会重新开辟新的空间。

|  |
| --- |
| <body>  <input type="button" value="按钮生成水果列表" id="btn">  <div id="box"></div>  <script type="text/javascript">  /\*var fruit = ['西瓜','葡萄','橙子','橘子'];  var btn = document.getElementById('btn');  btn.onclick = function () {  var box = document.getElementById('box');  box.innerHTML = '<ul>';// 重绘  for (var i = 0; i < fruit.length; i++) {  box.innerHTML += '<li>'+fruit[i]+'</li>'; // 重绘  }  //box.innerHTML = '</ul>';写成等号的话，会覆盖前边的内容，要进行叠加  box.innerHTML += '</ul>'; // 重绘  console.dir(box.innerHTML);//0 无法进行叠加？？？  }\*/  /\*var fruit = ['西瓜','葡萄','橙子','橘子'];  var btn = document.getElementById('btn');  btn.onclick = function () {  var box = document.getElementById('box');  var html = '<ul>';  for (var i = 0; i < fruit.length; i++) {  html += '<li>'+fruit[i]+'</li>';  }  html += '</ul>';  box.innerHTML = html;// 重绘一次  }\*/  var fruit = ['西瓜','葡萄','橙子','橘子'];  var btn = document.getElementById('btn');  btn.onclick = function () {  var box = document.getElementById('box');  var htmlArray = [];  htmlArray.push('<ul>');  for (var i = 0; i < fruit.length; i++) {  htmlArray.push('<li>'+fruit[i]+'</li>');  }  htmlArray.push('</ul>');  box.innerHTML = htmlArray.join('');// 默认的话，会以逗号隔开  }  </script>  </body> |

3、document.createElement()

（1）首先在内存中创建一个DOM对象

（2）此时的对象还不知到放在哪里，把创建的对象放到DOM树中去，此时会**重新绘制**

（3）然后去设置对象的属性（文字，样式...）

|  |
| --- |
| <body>  <input type="button" value="按钮生成段落" id="btn">  <div id="box">  <span>hello</span>  </div>  <script type="text/javascript">  var btn = document.getElementById('btn');  btn.onclick = function () {  var p = document.**createElement**('p');  var box = document.getElementById('box');  box.**appendChild**(p);// 这是属于追加的元素，会放在最后  p.innerText = 'ni hello';  p.style.color = 'purple';  }  </script>  </body>  1 |

（4）高亮列表选项

|  |
| --- |
| <body>  <input type="button" value="按钮生成列表" id="btn">  <div id="box"></div>  <script type="text/javascript">  var btn = document.getElementById('btn');  btn.onclick = function() {  var ul = document.**createElement**('ul');  var box = document.getElementById('box');  box.**appendChild**(ul);  var fruit = ['西瓜','橙子','葡萄','芒果'];  for (var i = 0; i < fruit.length; i++) {  var li = document.createElement('li');  ul.appendChild(li);  //li.innerText = fruit[i];// 在旧火狐中使用的是 textContent  **setinnerText**(li,fruit[i]);  li.onmouseover = **mouseOver**;// 不要把函数体放在这，循环一次，会在内存中创建一次，在外边存储一次就ok  li.onmouseout = mouseOut;  }  }  function setinnerText(element, text) {  if (typeof element.innerText === 'string') {  element.innerText = text;  } else {  element.textContent = text;  }  }  function **mouseOver**() {  this.style.backgroundColor = 'rgba(0,0,0,.4)';  }  function mouseOut() {  this.style.backgroundColor = '';  }  </script>  </body>  1 |

（5）动态创建表格案例

|  |
| --- |
| <style type="text/css">  #box table {  border-collapse: collapse;  }  #box table a {  text-decoration: none;  }  </style> |
| <body>  <div id="box"></div>  <script type="text/javascript">  var headDatas = ['姓名','科目','分数','操作'];  var bodyDates = [  {name: 'zs1', subject: '英语', score: '99'},  {name: 'zs2', subject: '英语', score: '99'},  {name: 'zs3', subject: '英语', score: '99'},  {name: 'zs4', subject: '英语', score: '99'},  {name: 'zs5', subject: '英语', score: '99'},  {name: 'zs6', subject: '英语', score: '99'},  {name: 'zs7', subject: '英语', score: '99'},  {name: 'zs8', subject: '英语', score: '99'}  ];  var table = document.createElement('table');  var box = document.getElementById('box');  box.appendChild(table);  table.border = '1px';  table.width = '400px';  // 创建表头  var thead = document.createElement('thead');  table.appendChild(thead);  var tr = document.createElement('tr');  thead.appendChild(tr);  tr.style.height = '40px';  tr.style.backgroundColor = 'lightgray';  for (var i = 0; i < headDatas.length; i++) {  var th= document.createElement('th');  tr.appendChild(th);  setInnerText(th,headDatas[i]);  }  // 创建数据  var tbody = document.createElement('tbody');  table.appendChild(tbody);  tbody.style.textAlign = 'center';  for (var i = 0; i < bodyDates.length; i++) {  var tr = document.createElement('tr');  tbody.appendChild(tr);  // 遍历对象要采用 for in  for (key in bodyDates[i]) {  td = document.createElement('td');  tr.appendChild(td);  setInnerText(td,bodyDates[i][key]);  }  /\*var td = document.createElement('td');  tr.appendChild(td);  setInnerText(td,bodyDates[i].name);  td = document.createElement('td');  tr.appendChild(td);  setInnerText(td,bodyDates[i].subject);  td = document.createElement('td');  tr.appendChild(td);  setInnerText(td,bodyDates[i].score);  td = document.createElement('td');\*/  td = document.createElement('td');  tr.appendChild(td);  var link = document.createElement('a');  td.appendChild(link);  setInnerText(link,'删除');  link.href = 'javascript:void(0)';  link.onclick = linkClick;  }  function linkClick() {  console.log('111111');  //要先找到父元素，才能删除子节点，table不是它的父元素，是祖先  tbody.removeChild(this.parentNode.parentNode);  }  </script>  </body>  11 |

table没有style设置的吗???

4、效率对比

|  |
| --- |
| <body>  <script type="text/javascript">  //1 使用innerHTML进行拼接，循环1000次  function innerHTMLRate() {  var d1,d2;  d1 = new Date();  for (var i = 0; i < 1000; i++) {  document.body.innerHTML += '<div style="width: 100px;height: 2px;border: 1px solid red;"></div>'  }  d2 = new Date();  console.log(d2 - d1);  }  //innerHTMLRate(); // 大概 2000多毫秒  //2 使用数组进行拼接  function arrayRate() {  var d1,d2;  var array = [];  d1 = new Date();  for (var i = 0; i < 1000; i++) {  array.push('<div style="width: 100px;height: 2px;border: 1px solid red;"></div>');  }  document.body.innerHTML = array.join('');  d2 = new Date();  console.log(d2 - d1);  }  //arrayRate(); 大概 10 毫秒  //3 使用createElement  function createElementRate() {  var d1,d2;    d1 = new Date();  for (var i = 0; i < 1000; i++) {  var div = document.createElement('div');  div.style.width = '100px';  div.style.height = '2px';  div.style.border = '1px solid red';  document.body.appendChild(div);  }    d2 = new Date();  console.log(d2 - d1);  }  //createElementRate(); 大概30 毫秒左右  </script> |
| 1  2  3  2跟3虽然有点差距，但是3的代码结构比较容易理解，也方便写事件，2的拼接字符串可能有些麻烦，比较难理解里边的结构。 |

5、insertBefore()

会将其从当前位置移动到新位置（在将节点附加到其他节点之前，不需要从其父节点删除该节点）。

|  |
| --- |
| var insertedNode = parentNode.insertBefore(newNode, referenceNode);  insertedNode 被插入节点(newNode)  parentNode 新插入节点的父节点  newNode 用于插入的节点  referenceNode newNode 将要插在这个节点之前  如果 referenceNode 为 null 则 newNode 将被插入到子节点的末尾。 |
| 函数返回被插入过的子节点 |

6、replaceChild()

方法用指定的节点替换当前节点的一个子节点，并返回被替换掉的节点。就是返回旧节点

|  |
| --- |
| parentNode.replaceChild(newChild, oldChild);  参数  newChild  用来替换 oldChild 的新节点。如果该节点已经存在于 DOM 树中，则它首先会被从原始位置删除。  oldChild  被替换掉的原始节点。 |

7、appendChild

**Node.appendChild()** 方法将一个节点附加到指定父节点的子节点列表的末尾处。如果将被插入的节点已经存在于当前文档的文档树中，那么 appendChild() 只会将它从原先的位置移动到新的位置（不需要事先移除要移动的节点）。原来父节点中的子节点索引会动态变化，更新。见下边水果例子。

这意味着，一个节点不可能同时出现在文档的不同位置。所以，如果某个节点已经拥有父节点，在被传递给此方法后，它首先会被移除，再被插入到新的位置。若要保留已在文档中的节点，可以先使用  [Node.cloneNode()](https://developer.mozilla.org/zh-CN/docs/Web/API/Node/cloneNode) 方法来为它创建一个副本，再将副本附加到目标父节点下。请注意，用 cloneNode 制作的副本不会自动保持同步。

|  |
| --- |
| <body>  <input type="button" value="按钮" id="btn">  <div id="box">  <ul>  <li>11</li>  <li>22</li>  <li>33</li>  </ul>  <ul id="sel"></ul>  </div>  <script type="text/javascript">  var btn = document.getElementById('btn');  btn.onclick = function () {  var box = document.getElementById('box');  var ul = box.children[0];  var div = document.createElement('div');  div.innerText = 'hello';  //ul.**insertBefore**(div,ul.children[0]);  //ul.**replaceChild**(div,ul.children[1]);  var sel = document.getElementById('sel');  sel.**appendChild**(ul.children[1]);  }  </script>  </body> |
| 12 |

8、cloneNode()

克隆函数

参数 true 表示复制节点的同时复制节点中的内容（文本、图片等等）

参数 false 表示只复制当前节点，不复制里面的内容。

需要克隆的元素.cloneNode(false);

9、选择水果案例

|  |
| --- |
| <style type="text/css">  select {  background-color: #33cccc;  width: 200px;  height: 200px;当使用multiple属性的时候，就能生效  text-align: center;  font-size: 20px;  }  </style> |
| <body>  <select id="left\_sel" multiple="multiple">  <option>西瓜</option>  <option>葡萄</option>  <option>杨梅</option>  <option>橙子</option>  <option>芒果</option>  </select>  <input id="btn1" type="button" value=">>">  <input id="btn2" type="button" value="<<">  <input id="btn3" type="button" value=">">  <input id="btn4" type="button" value="<">  <select id="right\_sel" multiple="multiple"></select>  <script type="text/javascript">  // 全部右移  var btn1 = document.getElementById('btn1');  var right\_sel = document.getElementById('right\_sel');  var left\_sel = document.getElementById('left\_sel');  btn1.onclick = function () {  var left\_options = left\_sel.children;  var left\_options\_length = left\_options.length;  //appendChild（）,options已经是拥有父节点，移动时会从原来的地方删除，会导致原来父元素的子节点数目随着动态减少。而且索引也会自行重新排序，见下边图片解析  /\***for (var i = 0; i < left\_options.length; i++) {**  **right\_sel.appendChild(left\_options[i]);//执行完这句话， left\_options.length;会自动减一，动态变化**  **}**\*/  for (var i = 0; i < left\_options\_length; i++) {  right\_sel.appendChild(left\_options[0]);  }  // 也可以使用**逆向循环**，从最后的子元素开始取，这样就不会有排序的问题。但是移动过去后，显示顺序会相反  }  // 全部左移  var btn2 = document.getElementById('btn2');  btn2.onclick = function () {  var right\_options\_length = right\_sel.children.length;  for (var i = 0; i < right\_options\_length; i++) {  left\_sel.appendChild(right\_sel.children[0]);  }  }  // 单独右移  var btn3 = document.getElementById('btn3');  btn3.onclick = function () {  var record\_obj = [];  // 先判断哪些被选中,存储对象，不是存索引，因为最后移动的时候，索引是动态变化的  var left\_opition\_selected = left\_sel.children;  for (var i = 0; i < left\_opition\_selected.length; i++) {  if (left\_opition\_selected[i].selected) {  record\_obj.push(left\_opition\_selected[i]);  // 去除选中后的阴影效果  left\_opition\_selected[i].selected = false;  }  }  // 调试console.log(record\_obj);  for (var i = 0; i < record\_obj.length; i++) {  right\_sel.appendChild(record\_obj[i]);  }  }  // 单独左移  var btn4 = document.getElementById('btn4');  btn4.onclick = function () {  var record\_obj = [];  // 先判断哪些被选中,存储对象，不是存索引，因为最后移动的时候，索引是动态变化的  var right\_options\_selected = right\_sel.children;  for (var i = 0; i < right\_options\_selected.length; i++) {  if (right\_options\_selected[i].selected) {  record\_obj.push(right\_options\_selected[i]);  // 去除选中后的阴影效果  right\_options\_selected[i].selected = false;  }  }  console.log(record\_obj);  for (var i = 0; i < record\_obj.length; i++) {  left\_sel.appendChild(record\_obj[i]);  }  }  </script>  </body>  1  2 |

当数组，字符串，DOM进行增加，删除操作的时候，要注意它自身的长度是否会跟着动态进行改变，以及里边的索引变化的情况。动态变化的话，要采用其他方式进行处理。

（1）使用innerHTML来简化左右全移

|  |
| --- |
| /\*for (var i = 0; i < left\_options\_length; i++) {  right\_sel.appendChild(left\_options[0]);//这种方式是直接把对象移动过去，事件就不会丢失  }\*/  right\_sel.innerHTML += left\_sel.innerHTML;  left\_sel.innerHTML = ''; |

（2）出现的问题

|  |
| --- |
| 1、使用innerHTML移动子元素的话，如果子元素有事件，**移动之后元素的事件丢失**。  2、当使用innerHTML清空子元素的时候，如果子元素有事件，此时会**发生内存泄漏**，此时事件处理函数还会继续在内存中存储着。这块内存不会被释放，还一直被占有着。  3、使用removeChild的话，就会把子元素的事件一并释放掉。 |

九、事件详解

监听器就是事件处理函数

1、addEventListener

可以**多次注册事件**，onclick是赋值，只能赋值一次，多次赋值会有覆盖的情况。

参数

Type：表示监听事件类型的字符串。

Listener：当所监听的事件类型触发时，会接收到一个事件通知（实现了 Event 接口的对象）对象。listener 必须是一个**实现了 EventListener 接口的对象？？？**，或者是**一个函数**。

|  |
| --- |
| <body>  <input type="button" value="按钮" id="btn">  <script type="text/javascript">  var btn = document.getElementById('btn');  btn.addEventListener('click',function () {  console.log('第一个事件');  });  btn.addEventListener('click',function () {  console.log('第二个事件');  });  </script>  </body>  1 |

2、addEventListener兼容性处理

|  |
| --- |
| <body>  <input type="button" value="按钮" id="btn">  <script type="text/javascript">  var btn = document.getElementById('btn');  // 浏览器兼容性问题，IE9以后才执行  btn.addEventListener('click',function () {  console.log('第一个事件');  });  btn.addEventListener('click',function () {  console.log('第二个事件');  });  // 注册事件的第三种方式  // 也有浏览器兼容性问题，IE的老版本特有的方法，支持IE，支持IE6-10支持  /\*btn.attachEvent('onclick',function (){  console.log('hi');  });\*/  // 处理兼容性问题  // eventName是不带 on的，如click、mousuover、mouseout  function addEventListener(element, eventName, fn) {  if (element.addEventListener) {  element.addEventListener(eventName,fn);  } else if (element.attachEvent) {  element.attachEvent('on'+eventName, fn);  } else {  **//element.onclick,对象的属性也可以写成下边的形式，访问对象属性的方式**  **element['on'+eventName] = fn;**  }  }  addEventListener(btn,'click',function () {  console.log('我是兼容性事件');  });  </script>  1 |

3、移除事件

|  |
| --- |
| <body>  <input type="button" value="点击移除事件" id="btn">  <script type="text/javascript">  var btn = document.getElementById('btn');  // 1  /\*btn.onclick = function () {  console.log('执行一次后，移除事件');  btn.onclick = null;  }\*/  // 2 如果想要移除事件，注册事件的时候不能使用匿名函数  /\*function removerEvent() {  console.log('addEventListener执行一次后，移除事件');  btn.removeEventListener('click',removerEvent);  }  btn.addEventListener('click',removerEvent);\*/  // 3 attach事件的移除。  function removerEvent() {  console.log('attachEvent执行一次后，移除事件');  btn.dettach('onclick',removerEvent);  }  btn.attach('onclick',removerEvent);  </script>  </body> |

十、事件的三个阶段

就是谁先去处理事情，是父元素，还是子元素。

addEventListener 的第三个参数为false的时候：事件冒泡。

addEventListener 的第三个参数为true的时候：事件捕获，从外往里。

**onclick和attach都只有事件冒泡**

注：点击onclick事件也可以由代码去触发：**btn.click();**

1、三个阶段

第一个阶段：捕获阶段。

第二个阶段：执行当前点击的元素。

第三个阶段：冒泡阶段。

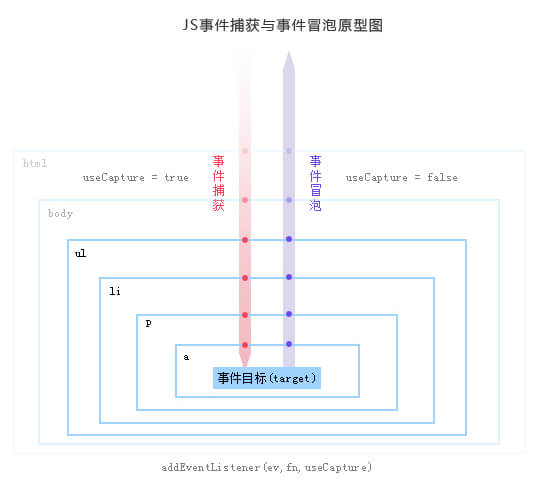
三个互相嵌套的盒子的事件从最里边的盒子开始触发，从里往外像是冒泡一样

**事件在触发的时候会经过三个阶段**。我们在代码中只能处理，第一或者第三中的一个阶段，其实这三个阶段都会发生，如果用的冒泡阶段，那么捕获阶段也会发生。只不过我们没法通过代码来进行干预。

|  |
| --- |
| <body>  <div id="box1">  <div id="box2">  <div id="box3"> </div>  </div>  </div>  <script type="text/javascript">  var box1 = document.getElementById('box1');  var box2 = document.getElementById('box2');  var box3 = document.getElementById('box3');  var array = [box1,box2,box3];  for (var i = 0; i < array.length; i++) {  array[i].addEventListener('click',boxClick,false);  }  function boxClick() {  console.log(this.id);  }  </script>  </body>  1 |

2、事件冒泡的作用。

事件处理函数里边有一个参数叫做事件参数。



如上图所示，事件模型是指分为三个阶段：

捕获阶段：在事件冒泡的模型中，捕获阶段不会响应任何事件；

目标阶段：目标阶段就是指事件响应到触发事件的最底层元素上；

冒泡阶段：冒泡阶段就是事件的触发响应会从最底层目标一层层地向外到最外层（根节点）。

事件代理即是利用事件冒泡的机制把里层所需要响应的事件绑定到外层事件。

（1）e 事件参数（事件对象）

当事件发生的时候，可以获取到一些和事件相关的数据。

|  |
| --- |
| 1、DOM标准中，是给事件处理函数一个参数。e就是事件对象，在老版本的IE中获取事件对象的方式是window.event。  2、处理事件对象的浏览器兼容性问题  e = e || window.event，如果不支持的话，后边的e值可能为空或者undefined，这样e = window.event。  3、e.eventPhase事件的阶段，1表示捕获阶段，2表示目标阶段，我们点击的这个元素在执行的一个过程，3表示冒泡阶段。  4、e.target 获取真正触发事件的对象。如在ul中点击li这个要再继续了解一下？？？这个也有兼容性的问题，在老版本的IE版本中 srcElement。（这样处理兼容性问题 var target = e.target || e.srcElement）  5、e.currentTaget 事件处理函数所属的那个对象。跟事件处理函数中的this是一样的。  <body>  <div id="box1">  <div id="box2">  <div id="box3"> </div>  </div>  </div>  <script type="text/javascript">  var box1 = document.getElementById('box1');  var box2 = document.getElementById('box2');  var box3 = document.getElementById('box3');  var array = **[box1]; // c采用了事件委托验证**  for (var i = 0; i < array.length; i++) {  array[i].addEventListener('click',boxClick,false);  }  function boxClick(e) {  e = e || window.event;  console.log(e.eventPhase);  var target = e.target || e.srcElement;  console.log(target);  console.log(e.currentTarget);  }  </script>  </body>  <style type="text/css">  #box1 {  width: 400px;  height: 400px;  background-color: red;  }  #box2 {  width: 300px;  height: 300px;  background-color: pink;  }  #box3 {  width: 200px;  height: 200px;  background-color: blue;  }  </style>  这里的结果跟事件委托有关系。有设置了事件才可能是目标阶段。蓝色它把事件委托给了最外层的红色，需要冒泡上来，才能打印。有设置的事件就执行。  1 |

e.target // 就是你真正点击触发事件的目标对象，

当系统调用的时候，会给它传一个参数，我们直接获取数据就ok

（2）事件对象(e.type)

多个事件指向同一个处理函数时，就能用e.type,可以减少内存的使用。

|  |
| --- |
| <body>  <div id="box1"></div>  <script type="text/javascript">  var box1 = document.getElementById('box1');  box1.onclick = fn;  box1.onmouseover = fn;  box1.onmouseout = fn;  function fn(e) {  e = e || window.event;  switch (e.type) {  case 'click':  console.log('click');  break;  case 'mouseover':  console.log('mouseover');  break;  case 'mouseout':  console.log('mouseout');  break;  }  }  </script>  </body>  1 |

（3）clientX、clientY

获取**鼠标**在浏览器的可视区域的坐标。如果页面有滚动条，坐标会随着可视区域的变化而变化

（4）pageX、pageY

鼠标在当前页面中的位置。

有兼容性问题，IE9以后支持

pageY = clientY + 页面滚动出去的距离（**不是滚动条滚动的距离**）

document.documentElement是文档的根元素，是html标签，不是文档声明<!DOCTYPE html>

|  |
| --- |
| <style type="text/css">  #box1 {  margin: 100px;  width: 100px;  height: 100px;  background-color: red;  }  body {  height: 1000px;  }  </style>  <body>  <div id="box1"></div>  <script type="text/javascript">  var box1 = document.getElementById('box1');  box1.onclick = fn;  function fn(e) {  //console.log(e.**clientX**,e.**clientY**);  console.log(e.**pageX**,e.**pageY**);  }  </script>  </body>  12  3 |

（5）pageX、pageY兼容性问题

有兼容性问题，IE9以后支持

pageY = clientY + 页面滚动出去的距离

|  |
| --- |
| 1、输出页面滚动的距离  document.body.scrollLeft（横向）  document.body.scrollTop（纵向）  有的浏览器是使用  document.documentElement是文档的根元素，是html标签，不是文档声明<!DOCTYPE html>  document.documentElement.scrollLeft  这个也要处理一下兼容性问题  Chrome最新版使用body的那种方式输出 0，要使用documentElement才可以 |
| <style type="text/css">  body {  height: 1000px;  }  </style>  </head>  <body>  <script type="text/javascript">  var img = document.getElementById('img');  document.onclick = fn;  function fn(e) {  e = e || window.event;  console.log(document.body.scrollLeft);  console.log(document.body.scrollTop);// 在chrome中输出0  console.log(document.documentElement.scrollLeft);  console.log(document.documentElement.scrollTop);// 在chrome中输出带有小数  var scroll = getScroll();  console.log(scroll.scrollLeft,scroll.scrollTop);  }  // 页面滚动出去的距离  function getScroll() {  // document.body.scrollTop); 在chrome中输出0  // document.documentElement.scrollTop); 在chrome中输出带有小数  var scrollLeft = document.body.scrollLeft || document.documentElement.scrollLeft;  var scrollTop = document.body.scrollTop || document.documentElement.scrollTop;  return {  scrollLeft: scrollLeft,  scrollTop: scrollTop  }  }  </script>  </body> |

|  |
| --- |
| 2、处理pageX、pageY兼容性问题  <style type="text/css">  body {  height: 1000px;  }  </style>  </head>  <body>  <img src="images/tianshi.gif" id="img">  <script type="text/javascript">  document.onclick = fn;  function fn(e) {  e = e || window.event;  console.log(getPage(e).pageX,getPage(e).pageY);  }  function getPage(e) {  var pageX = e.pageX || e.clientX + getScroll().scrollLeft;  var pageY = e.pageY || e.clientY + getScroll().scrollTop;  return {  pageX: pageX,  pageY:pageY  }  }  </script>  </body> |

（6）offsetLeft、offsetTop偏移量，获取在盒子中的坐标

它是一个只读属性，不能去设置、赋值。

|  |
| --- |
| <style type="text/css">  body {  height: 1000px;  }  #box {  width: 200px;  height: 200px;  background-color: red;  margin: 100px;  }  </style>  <body>  <div id="box"></div>  <script type="text/javascript">  var box = document.getElementById('box');  box.onclick = fn;  function fn(e) {  e = e || window.event;  var x = getPage(e).pageX - this.offsetLeft;  var y = getPage(e).pageY - this.offsetTop;  console.log(x,y);  }  </script>  </body> |
| 1 |

（7）取消默认行为的执行、阻止冒泡。

|  |
| --- |
| 1、取消默认行为的执行  <body>  <a href="http://www.baidu.com" id="link">百度</a>  <script type="text/javascript">  var link = document.getElementById('link');  link.onclick = function (e) {  //**return false;//如果不带事件参数e就可以使用这种方式**  // DOM中的标准方法  //**e.preventDefault(); // 后续如果还有其他代码的话，会继续执行**  // 老IE中使用，非标准方式  //**e.returnValue = false;**  }  </script>  </body> |
| 2、阻止冒泡  <style type="text/css">  #box1 {  width: 400px;  height: 400px;  background-color: red;  }  #box2 {  width: 300px;  height: 300px;  background-color: pink;  }  #box3 {  width: 200px;  height: 200px;  background-color: blue;  }  </style>  <body>  <div id="box1">  <div id="box2">  <div id="box3"> </div>  </div>  </div>  <script type="text/javascript">  var box1 = document.getElementById('box1');  var box2 = document.getElementById('box2');  var box3 = document.getElementById('box3');  var array = [box1,box2,box3]; // c采用了事件委托验证  for (var i = 0; i < array.length; i++) {  array[i].addEventListener('click',boxClick,false);  }  function boxClick(e) {  e = e || window.event;  console.log(e.eventPhase);// 如果是事件捕获 输出 1 box1  console.log(this.id);  // 阻止事件的传播，取消冒泡，也可以取消 事件捕获  // 标准DOM方法  // e.stopPropagation();  // 非标准的方式，老版本IE支持,它是一个属性  e.cancelBubble = true;  }  </script>  </body>  1 |

（8）键盘事件

onkeydown键盘按下的时候

onkeyup键盘弹起的时候

onkeyup与onkeydown的区别：onkeydown的时候，我们所按的键还没有落入文本框。

onkeyup的时候，我们所按的键已经落入文本框。

e.keyCode是键盘码

|  |
| --- |
| <body>  <input type="text" id="input">  <script type="text/javascript">  var input = document.getElementById('input');  input.onkeydown = function (e) {  //console.log(e.keyCode);  if ((e.keyCode < 48 || e.keyCode > 57) && e.keyCode !== 8) {  // 取消默认行为  e.preventDefault();  }  }  </script>  </body> |

（9）其中一个作用：事件委托

事件委托，通俗地来讲，就是把一个元素响应事件（click、keydown......）的函数委托到另一个元素；

一般来讲，会把一个或者一组元素的事件委托到它的父层或者更外层元素上，真正绑定事件的是外层元素，当事件响应到需要绑定的元素上时，会通过事件冒泡机制从而触发它的外层元素的绑定事件上，然后在外层元素上去执行函数。

举个例子，比如一个宿舍的同学同时快递到了，一种方法就是他们都傻傻地一个个去领取，还有一种方法就是把这件事情委托给宿舍长，让一个人出去拿好所有快递，然后再根据收件人一一分发给每个宿舍同学；

在这里，取快递就是一个事件，每个同学指的是需要响应事件的 DOM 元素，而出去统一领取快递的宿舍长就是代理的元素，所以真正绑定事件的是这个元素。按照收件人分发快递的过程就是在事件执行中，需要判断当前响应的事件应该匹配到被代理元素中的哪一个或者哪几个。这个事情系统会通过e来告诉我们，我们去拿参数ok。

利用DOM的传播机制（点击任意元素，document的click都要出发），我们给document绑定一个点击事件，在事件中我们只需要获取事件源；根据不同的事件源做不同的事件就可以的了（这样就可以不用给元素一个个的绑定事件的了）；

当事件发生在子元素中的时候，往往会引起连锁反应，就是在它的祖先元素上也会发生这个事件，比如说你点击了div一个，也相当于点击了一个body，同样相当于点击了html，同样相当于点击了document；

简化了给所有li标签设置属性的过程，我们只需要通过给其父元素注册事件，对比，[高亮列表选项](#高亮列表选项)

|  |
| --- |
| <body>  <div>  <ul id="fruit">  <li>西瓜</li>  <li>葡萄</li>  <li>芒果</li>  <li>苹果</li>  <li>杨梅</li>  </ul>  </div>  <script type="text/javascript">  var fruit = document.getElementById('fruit');  fruit.**addEventListener**('click',fruitClick,**false**);  function fruitClick(**e**) {  //console.log(e.target);  for (var i = 0; i < fruit.children.length; i++) {  fruit.children[i].style.backgroundColor = '';  }  **e.target.style.backgroundColor = 'rgba(0,0,0,.3)';**  }  </script>  </body>  1 |

3、案例

（1）跟着鼠标飞的天使

|  |
| --- |
| <style type="text/css">  body {  height: 1000px;  }  #img {  position: absolute;  }  </style>  </head>  <body>  <img src="images/tianshi.gif" id="img">  <script type="text/javascript">  var img = document.getElementById('img');  //img.onmousemove = fn;变成了图片的移动  document.onmousemove = fn;  function fn(e) {  e = e || window.event;  //img.style.marginTop = e.clientY + 'px';  // 边偏移是相对于整个页面的。clientY是相对于可视区域，有滚动条时会有偏差。  //img.style.marginLeft = e.clientX +'px';  img.style.top = e.pageY - 10 + 'px';  img.style.left = e.pageX - 10 +'px';  }  </script>  </body> |

十一、BOM

1、BOM的概念

BOM(Browser Object Model) 是指**浏览器对象模型**，浏览器对象模型提供了独立于内容的、可以与浏览器窗口进行互动的对象结构。BOM由多个对象组成，其中代表浏览器窗口的**Window对象是BOM的顶层对象**，其他对象都是该对象的子对象。

我们在浏览器中的一些操作都可以使用BOM的方式进行编程处理，比如：刷新浏览器、后退、前进、在浏览器中输入URL等。

2、BOM的顶级对象window

window是浏览器的顶级对象，当调用window下的属性和方法时，**可以省略**window。

（1）name是window的属性，是字符串类型

定义变量名字的时候，不要用它们两的名字。

（2）top是window的属性，只能获取不能赋值

|  |
| --- |
| <script type="text/javascript">  var **name** = 123;  console.log(typeof name,name);// 输出 string 123  var **top** = 123;  console.log(typeof top,top);// 输出 object  var normal = 123;  console.log(typeof normal,window.normal);  </script>  1 |

（3）对话框 alert()、prompt()、confirm()

后边基本上不使用者三个，样式不美观、用户体验不好，在不同浏览器中，他们三长的不一样。

|  |
| --- |
| <input id="btn1" type="button" value="alert">  <input id="btn2" type="button" value="confirm">  <input id="btn3" type="button" value="prompt">  <script type="text/javascript">  var btn1 = document.getElementById('btn1');  var btn2 = document.getElementById('btn2');  var btn3 = document.getElementById('btn3');  btn1.onclick = function () {  alert('hello');  }  btn2.onclick = function () {  var result = confirm('是否确认要删除');  console.log(result);  }  btn3.onclick = function () {  var result = prompt('请输入名字','张三');  console.log(result);  }  </script>  1 |

3、页面加载事件

（1）onload

|  |
| --- |
| window.onload = function () {  // 当页面加载完成之后执行。  // 页面加载完成，即是页面上所有的元素创建完毕，并且引用的外部资源下载完毕（外部的js、css、图片等等）。  }  在body结束标签前中script标签中代码，效率会更高，因为它不需要等外部的资源加载完成。  基本是上所有标签也可以有onload事件。 |

（2）onunload

|  |
| --- |
| window.onunload = function () {  // 当用户退出页面时执行，在onunload中所有的**对话框**都无法使用，此时要卸载了，window对象会被冻结，对话框又是window的方法。此时会阻止对话框。  点击**F5**刷新界面的时候，先是卸载页面，再重新加载整个界面。  直接点击关闭就什么都无法看到。  } |

|  |
| --- |
| <body>  <script type="text/javascript">  **onload** = function () {  var box = document.getElementById('box');  console.dir(box);  console.log('页面加载ok');  }  **onunload** = function () {  alert('gogo');// 在刷新时候能看到错误。  }  </script>  <div id="box"></div>  </body> |

十二、定时器

设置定时器，如果刷新后，页面会有一定时间的空白，因为定时函数还没有执行，此时可以先让要定时执行的函数先单独调用一次，在页面上有内容。

1、setTimeout()和clearTimeout()

在指定的毫秒数到达之后执行指定的函数，只执行一次。

setTimeout() 隔一段时间执行，并且只会执行一次。像定时炸弹，炸一次就没了。

第一个参数，要执行的函数。

第二个参数，间隔的时间，单位是毫秒。

返回值是一个整数，是定时器的标识。clearTimeout的参数就是这标识。是该定时器的队列号。

可以递归调用自身模拟setInterval方法。

function hello() {

console.log('hi');

setTimeout(hello,1000);

}

hello();

2、setInterval()和clearInterval()

setInterval() 隔一段时间执行，并且会重复执行。像闹钟，每天可以循环

参数、返回值跟setTimeout一样。

定时调用的函数，可以按照给定的时间(单位毫秒)周期调用函数

|  |
| --- |
| <body>  <input type="button" value="开始2秒计时" id="btn1">  <input type="button" value="停止计时" id="btn2">  <script type="text/javascript">  var btn1 = document.getElementById('btn1');  var timeFlag;  btn1.onclick = function () {  **// timeFlag = setTimeout(fn, 2000);**  timeFlag = setInterval(fn, 1000);  };  function fn() {  console.log('爆炸一次');  }  var btn2 = document.getElementById('btn2');  btn2.onclick = function () {  **// clearTimeout(timeFlag);**  clearInterval(timeFlag);  console.log('停止计时');  };  </script>  </body> |

3、删除提示小案例

|  |
| --- |
| <style type="text/css">  #tip {  width: 200px;  height: 60px;  background-color: lightgray;  color: deeppink;  font-size: 20px;  line-height: 60px;  text-align: center;  text-shadow: 2px 2px 3px rgba(0,0,0,.8);  margin: 200px auto;  display: none;  }  </style> |
| <body>  <input type="button" value="删除" id="btn1">  <div id="tip">删除成功</div>  <script type="text/javascript">  var btn1 = document.getElementById('btn1');  btn1.onclick = function () {  var tip = document.getElementById('tip');  tip.style.display = 'block';  setTimeout(function () {  tip.style.display = 'none';  },3000);  }  </script>  </body> |

4、倒计时小案例

|  |
| --- |
| <style type="text/css">  #box {  width: 300px;  height: 300px;  margin: 100px auto;  text-align: center;  }  #box span {  background-color: orange;  font-size: 25px;  color: #fff;  text-shadow: 2px 2px 3px rgba(0,0,0,.7);  }  </style>  <body>  <div id="box">  <h2>距离xxx还有</h2>  <strong><span id="day">00</span><span>天<span></strong>  <strong><span id="hour">00</span><span>时<span></strong>  <strong><span id="minute">00</span><span>分<span></strong>  <strong><span id="second">00</span><span>秒<span></strong>  </div>  <script type="text/javascript">    //var date2 = new Date('2021-4-1');  var date2 = new Date(2021,3,1);// 3代表的是4月份  var day = document.getElementById('day');  var hour = document.getElementById('hour');  var minute = document.getElementById('minute');  var second = document.getElementById('second');  function getDiffTime(date1, date2) {  var diffSecond = (date2 - date1) / 1000;  var day = Math.floor(diffSecond / 60 / 60 / 24);  var hour = Math.floor(diffSecond / 60 / 60 % 24);  var minute = Math.floor(diffSecond / 60 % 60);  var second = Math.floor(diffSecond % 60);  day = day < 10 ? '0'+ day : day;  hour = hour < 10 ? '0'+ hour : hour;  minute = minute < 10 ? '0'+ minute : minute;  second = second < 10 ? '0'+ second : second;  return {  day: day,  hour: hour,  minute: minute,  second: second  }  }  function updateTime() {  var date1 = new Date();// 这里是实时的获取，全局变量就获得一次时间后就没有了  var diff = getDiffTime(date1,date2);  setInnerText(day,diff.day);  setInnerText(hour,diff.hour);  setInnerText(minute,diff.minute);  setInnerText(second,diff.second);  }  updateTime();  setInterval(updateTime, 1000);  </script>  </body>  1 |

5、模拟动画小案例

|  |
| --- |
| 效果，让方形在指定的距离停下 |
| <style type="text/css">  #box {  width: 200px;  height: 200px;  background-color: red;  position: absolute;  }  </style>  <body>  <input type="button" value="开始移动" id="btn">  <div id="box"></div>  <script type="text/javascript">  var btn = document.getElementById('btn');  var box = document.getElementById('box');  var speed = 10;  var distance = 600;  var timeInterupt = 30;// 定时的时间  var timeFlag;  btn.onclick = function () {  timeFlag = setInterval(moveBox,timeInterupt);  }  function moveBox() {  if ((box.offsetLeft + speed) >= distance) {  box.style.left = distance + 'px';  clearInterval(timeFlag);  return;  }  **box.style.left** = box.**offsetLeft** + speed + 'px';  //console.log(box.style.left);  }  </script>  </body> |
| 多次点击按钮的时候，会加快移动的速度，因为点击一次，会创建多一个定时器来执行这个事情，具体是怎么搞还未知。 |

十三、location 地址、网址对象

1、location对象是window对象下的一个属性，使用的时候可以省略window对象。

location可以获取或者设置浏览器地址栏的URL。

（1）location有哪些成员？

使用chrome的控制台查看、查MDN、成员

|  |
| --- |
| location.href = 'http://www.baidu.com'，比较经常使用。  assign('http://www.baidu.com') 委派，它的作用跟href的作用一样，可以让页面跳转到指定的地方。他们都可以后退。  reload()，重新加载 refresh F5刷新。 参数 **true**表示强制从服务器获取页面。对应 ctrl+F5强制刷新。 **false** 如果浏览器有缓存，直接从缓存获取页面。对应F5刷新。  replace('http://www.baidu.com') 替换，替换掉地址栏中的地址，但是不记录历史，只有记录历史的话，浏览器后退功能才能用。  hash/host/hostname/search/href…… |
| <body>  <input type="button" value="跳转" id="btn">  <script type="text/javascript">  var btn = document.getElementById('btn');  btn.onclick = function () {  console.log(location.href);  //location.href = 'http://www.baidu.com';//能有历史记录，能回退  //location.assign('http://www.baidu.com');//能有历史记录，能回退  //location.replace('http://www.baidu.com'); // 没有历史记录  location.reload(true); // 重新加载，相当于F5刷新  }  </script>  </body> |

2、URL的组成

|  |
| --- |
| scheme://host:port/path?query#fragment  http://www.itheima.com:80/a/b/index.html?name=zs&age=18#bottom  scheme:通信协议  常用的http,ftp,maito等，打开本地文件的话为 file协议  host:主机  服务器(计算机)域名系统 (DNS) 主机名或 IP 地址。  port:端口号  整数，可选，省略时使用方案的默认端口，如http的默认端口为80。可能显示为空  path:路径  由零或多个'/'符号隔开的字符串，一般用来表示主机上的一个目录或文件地址。  query:查询  可选，用于给动态网页传递参数，可有多个参数，用'&'符号隔开，每个参数的名和值用'='符号隔开。例如：name=zs  fragment:信息片断，字符串，锚点.  #: 表示当前页面 |

search会包含 ?name=zs&age=18#bottom 这样的一串数据，通常可以在F12中输出 location来查看

3、history对象

back()、forward()

go()，go(1)，前进一个页面。go(-1)回退一个页面，有多个就写多个。

|  |
| --- |
| <body>  <h2>first</h2>  <a href="history2.html">history2</a>  <input type="button" id="btn" value="前进">  <script type="text/javascript">  var btn = document.getElementById('btn');  btn.onclick = function () {  //history.forward();  history.go(1);  }  </script>  </body> |
| <body>  <h2>second</h2>  <a href="history1.html">history1</a>  <input type="button" id="btn" value="后退">  <script type="text/javascript">  var btn = document.getElementById('btn');  btn.onclick = function () {  // history.back();  history.go(-1);  }  </script>  </body>  1 |

十四、navigator对象

在chrome上可以模拟手机上浏览页面，服务器端会判断你是手机还是pc端。在pc输入www.baidu.com会自动补上前边的http://

1、userAgent

他会记录着浏览器设备的一些信息（浏览器版本号，操作系统信息等等），能够判断出你是移动端还是pc端设备



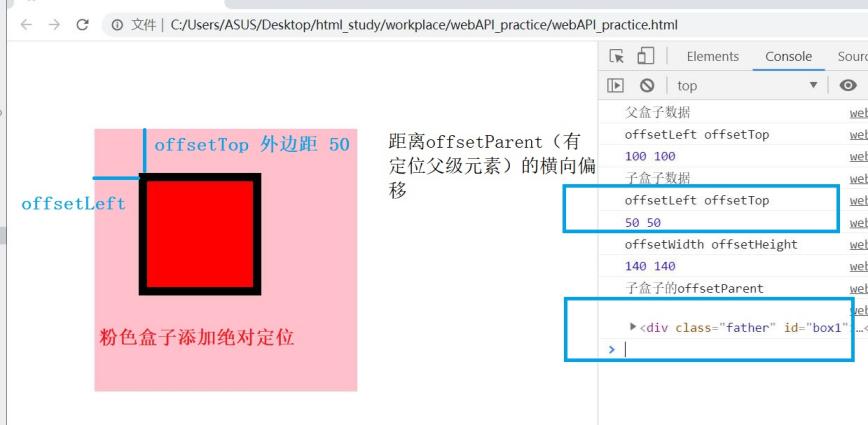
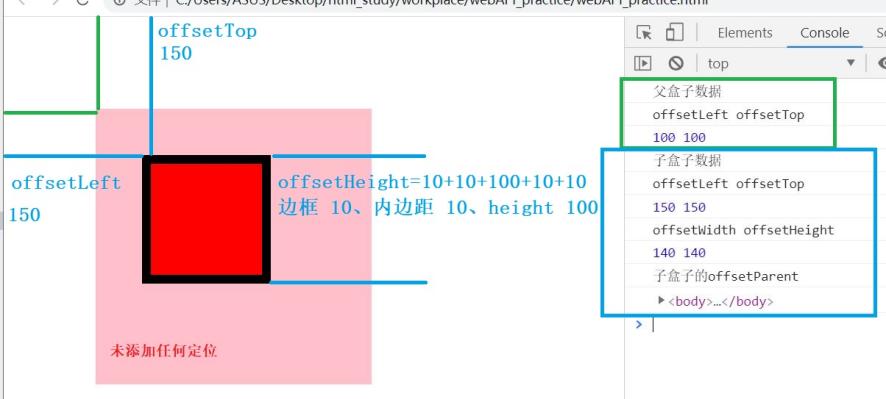


十五、偏移量

1、offsetParent：它是一个属性，用于获取距离当前元素、最近的、有定位的父级元素。如果没有定位的父元素，此时是body。

2、获取绝对位置坐标offsetLeft、offsetTop：距离offsetParent（有定位父级元素）的横向偏移。

3、offsetWidth、offsetHeight：**边框 + padding + width**。



|  |
| --- |
| <style type="text/css">  \* {  margin: 0;  padding: 0;  }  .father {  margin: 100px;  width: 300px;  height: 300px;  background-color: pink;  overflow: hidden;/\*防止嵌套垂直外边距合并\*/  position: relative;  }  .son {  width: 100px;  height: 100px;  background-color: red;  border: 10px solid #000;  margin: 50px;  padding: 10px;  }  </style>  <body>  <div class="father" id="box1">  <div class="son" id="box2"></div>  </div>  <script type="text/javascript">  var box1 = document.getElementById('box1');  var box2 = document.getElementById('box2');  console.log('父盒子数据');  console.log('offsetLeft offsetTop');  console.log(box1.offsetLeft,box1.offsetTop);  console.log('子盒子数据');  console.log('offsetLeft offsetTop');  console.log(box2.offsetLeft,box2.offsetTop);  console.log('offsetWidth offsetHeight');  console.log(box2.offsetWidth,box2.offsetHeight);  console.log('子盒子的offsetParent');  console.log(box2.offsetParent);  </script>  </body> |

4、获取上、左边框clientLeft、clientTop

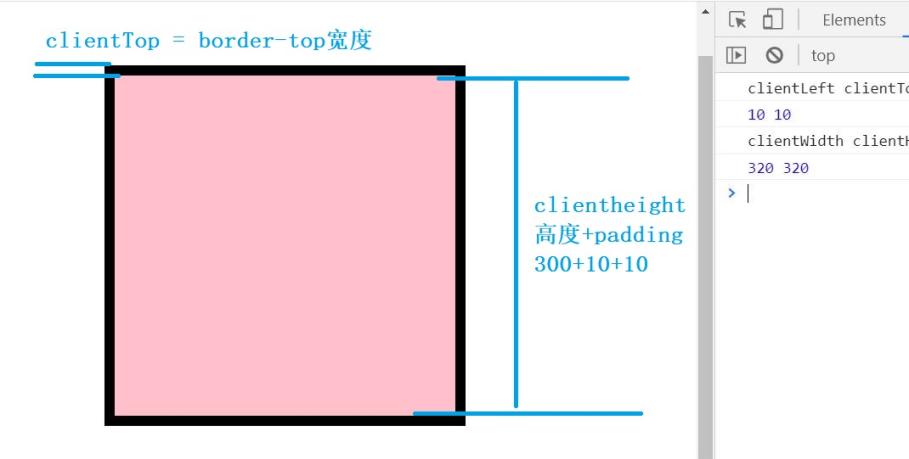
clientLeft：是border-left的宽度

clientTop：是border-top的宽度

5、clientWidth、clientHeight：**padding + width**

但是不包括边框。如果盒子内有滚动条，则还要减去滚动条的宽度、高度。

客户区的大小，用户看到有显示的地方。

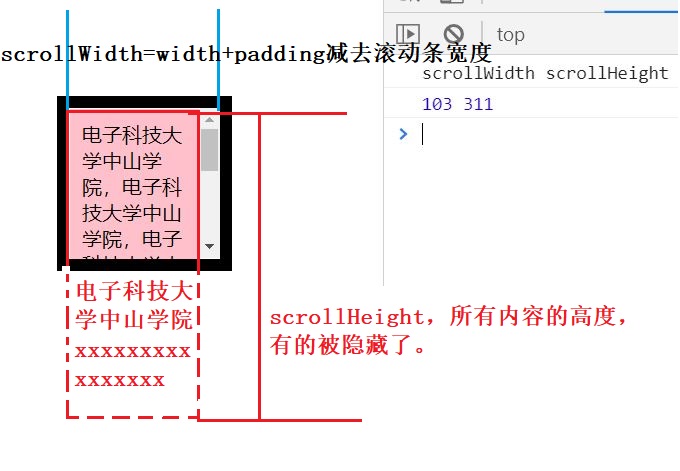


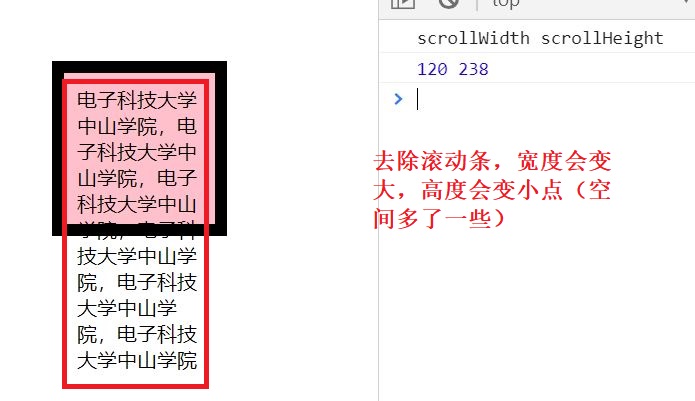
|  |
| --- |
| <style type="text/css">  \* {  margin: 0;  padding: 0;  }  .father {  margin: 100px;  width: 300px;  height: 300px;  background-color: pink;  padding: 10px;  border: 10px solid #000;  }  </style>  <body>  <div class="father" id="box1"></div>  <script type="text/javascript">  var box1 = document.getElementById('box1');  console.log('clientLeft clientTop');  console.log(box1.clientLeft,box1.clientTop);  console.log('clientWidth','clientHeight');  console.log(box1.clientWidth,box1.clientHeight);  </script>  </body> |

6、获取内容滚出距离scrollWidth、scrollHeight

scrollWidth元素的大小width + padding（不包括滚动条，滚动条是在盒子里边的，有滚动条的话，要减它的宽度。），通过overflow属性设置是否出现滚动条。

scrollHeight，里边内容的总高度，有一些被隐藏了。





|  |
| --- |
| <style type="text/css">  \* {  margin: 0;  padding: 0;  }  .father {  margin: 100px;  width: 100px;  height: 100px;  background-color: pink;  padding: 10px;  border: 10px solid #000;  overflow: auto;  }  </style>  <body>  <div class="father" id="box1">  电子科技大学中山学院，电子科技大学中山学院，电子科技大学中山学院，电子科技大学中山学院，电子科技大学中山学院，电子科技大学中山学院  </div>  <script type="text/javascript">  var box1 = document.getElementById('box1');  console.log('scrollWidth','scrollHeight');  console.log(box1.**scrollWidth**,box1.**scrollHeight**);  </script>  </body> |

7、scrollLeft、scrollTop

（1）onscroll事件



|  |
| --- |
| <style type="text/css">  \* {  margin: 0;  padding: 0;  }  .father {  margin: 100px;  width: 100px;  height: 100px;  background-color: pink;  padding: 10px;  border: 10px solid #000;  overflow: auto;  }  </style>  <body>  <div class="father" id="box1">  电子科技大学中山学院，电子科技大学中山学院，电子科技大学中山学院，电子科技大学中山学院，电子科技大学中山学院，电子科技大学中山学院  </div>  <script type="text/javascript">  var box1 = document.getElementById('box1');  box1.**onscroll** = function () {  console.log('scrollLeft','scrollTop');  console.log(box1.scrollLeft,box1.scrollTop);  }  </script>  </body> |

8、onmouseover 与 onmouseenter

onmouseover会触发事件冒泡，父盒子设置事件，子元素会冒泡事件触发

onmouseenter 不会触发事件冒泡

onmousemove与onmouseleave的区别与上边是一样的。

|  |
| --- |
| <style type="text/css">  \* {  padding: 0;  margin: 0;  }  #father {  width: 300px;  height: 300px;  background-color: red;  }  #son {  width: 100px;  height: 100px;  margin-left: 200px;  background-color: yellow;  }  </style>  <body>  <div id="father">  <div id="son"></div>  </div>  <script type="text/javascript">  var father = document.getElementById('father');  var son = document.getElementById('son');  father.onmouseenter = function (e) {  e = e || window.event;  console.log(e.target);// 输出father 直接触发  }  /\*father.onmouseover = function (e) {  e = e || window.event;  console.log(e.target);// 触发事件的son，冒泡机制  }\*/  father.onmouseleave = function (e) {  console.log(this.id+'go');  }  </script>  </body> |

十六、案例

先完成功能，再去考虑兼容性问题

1、拖拽案例---计算出边偏移

（1）先求得鼠标在按下的时候，获取鼠标在整个大盒子中的位置。

（2）鼠标的在大盒子中的位置 = 鼠标在页面的位置 - 大盒子的位置。

（3）鼠标移动的时候是在文档移动的（document）

（4）当鼠标弹起的时候，要移除鼠标移动事件。

2、弹出登录窗口

（1）例如百度的登录效果，叫做模态窗口

（2）需要有一个灰色的遮盖层，需半透明的效果，不然都遮住，一让其他的元素被挡着，除了登录相关元素。二突出显示登录窗口

3、放大镜案例

|  |
| --- |
| 要做到接近真正做到放大镜效果。  在小图片上的小方框占了小图片的比例需要  放大的显示框占大图片的比例，他们俩之间要相等，这样就显示的内容就是选中的部分。不然会显示多，或者少了。 |

webp是谷歌研发的一种图片格式，同样图片，比其他格式的体积会更小

（1）鼠标经过的时候，显示方框和大图片，当鼠标离开box的时候隐藏方框和大图

（2）当鼠标在盒子中移动的时候，让方框和鼠标一起移动

（3）当方框移动的时候，让大图片跟着移动

问题总结：（注：小手是样式的问题，不关js）

|  |
| --- |
| 1、鼠标在onmouseover的问题，鼠标已经移出box为什么还能移动方框，冒泡事件，这里涉及到了能不能给img设置鼠标经过事件，给box就ok.    onmouseover支持事件冒泡，因为方框是box的子元素，方框随着鼠标进行偏移，相当于鼠标一直在方框上面，由于冒泡机制导致box的鼠标经过事件一直存在，不会让方框消失。  如果是给img设置了onmouseover事件，鼠标经过的时候，方框会出现闪烁的效果，这是因为方框跟img是兄弟关系，方框出现导致遮挡了img，出现onmouseoout的事件，方框会消失，此时移动又触发onmouseover --- 方框遮挡触发 onmouseoout事件 --- 方框消失---- 这样子循环，会使得出现类似于闪烁效果。 |
| 2、出现边缘吸附的问题  **一开始的思路：**  一开始鼠标在方框左上角，为了鼠标在方框中心点上，把方框向左、向上移动。这导致了在边缘判断的时候以为是以为中心点为参考界限，但是此时的鼠标在盒子中的坐标被处理为边偏移的值。思路有点问题，不是以鼠标坐标为出发点，要以方框为主即边偏移。  **吸附产生的分析：**  在到达条件的时候发现被吸附了过去，移动到红色方框位置，就被吸附过去了。而且移动不出，鼠标必须移除整个方框的长度才能移动。因为鼠标坐标在计算时，X - mask.clientWidth / 2,移到中心点，鼠标在方框内计算后的值一直都是符合判断条件，就一直贴着。当时写的判断条件是以鼠标坐标作为边界判断，但是数据是被转换为边偏移了。  **解决方式：**  就是把鼠标在盒子中的坐标跟边偏移的值混了。此时要代码判断的值，是坐标转换为边偏移后的数值。  如果要以鼠标在盒子中的坐标作为边界判断，那么判断条件必须放在获取当前鼠标坐标后。不能在转为边偏移的后边。**这个方式有点绕。**  **以边偏移作为边界判断**。  1 |
| 3、盒子没有设置宽度问题    在处理完边界问题后出现的，在离开边框左边的时候，鼠标移出box右边的时候，方框没有消失，也就是说鼠标经过box事件一直还在。  用F12查看后，发现box没有设置宽度，它默认是独占一行，所以box是占了一整行，右边才会一直有box鼠标经过事件。  **解决方式**：给box添加宽度width。 |

服务器拿那边判断浏览器的类型。如果难点浏览器支持webp格式的图片，那我们看到的就是webp的格式的图，如果不支持就看到jpg格式图片，这是服务器那边处理的事情，我们暂时可以使用jpg格式的图片，大家都支持。

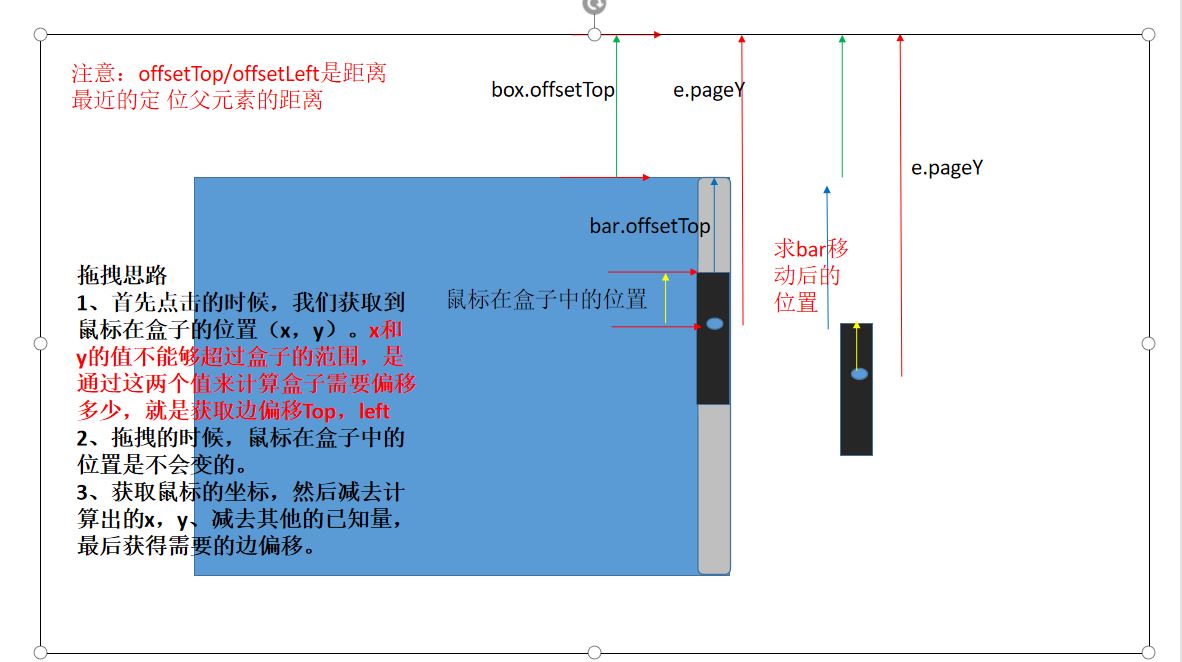
4、模拟滚动条

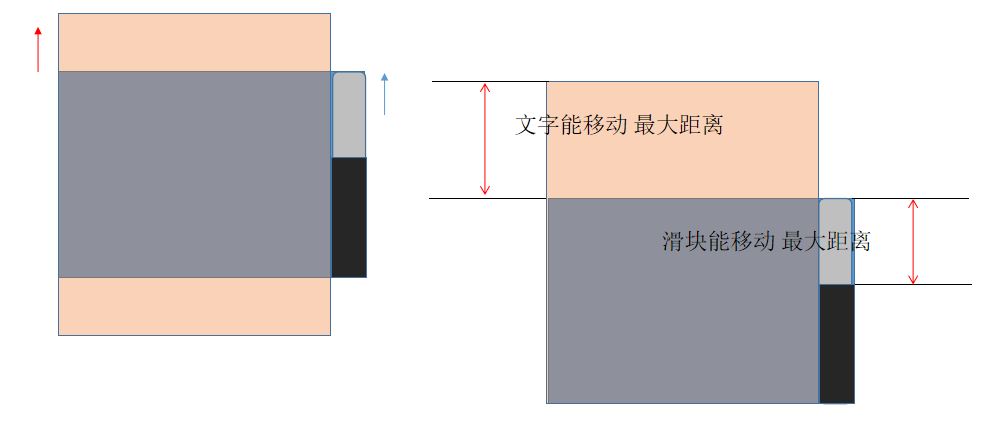
使用overflow的滚动条的样式无法设置的

（1）让滚动条能够拖拽

（2）当退拽滚动条的时候，改变内容的位置

（3）根据内容的大小，计算滚动条的高度





|  |
| --- |
| 一开始出现的滑块问题：第一次没问题，第二次有问题，向上移动了一段距离。  原因是，对拖拽的处理流程比较生疏，刚好第一次是起始的位置比较特殊，它具备的条件比较少，看不出来还有其他的已知量。  后续考虑退拽，要先考虑在中间位置的情况，能比较好知道可以利用那些已知量来进行计算  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>WebAPI练习</title>  <script type="text/javascript" src="index.js"></script>  <style type="text/css">  \* {  padding: 0;  margin: 0;  }  .box {  width: 100px;  height: 200px;  background-color: pink;  margin: 100px;  }  #scrollbox {  width: 20px;  height: 200px;  float: right;  background-color: gray;  }  #slider {  width: 20px;  height: 50px;  border-radius: 10px;  background-color: red;  position: absolute;  cursor: move;  }  </style>  </head>  <body>  <div class="box" id="cententBox">  <div id="scrollbox">  <div id="slider"></div>  </div>  </div>  <script type="text/javascript">  var slider = document.getElementById('slider');  var scrollbox = document.getElementById('scrollbox');  slider.onmousedown = function (e) {  var y = **getPage(e).pageY - scrollbox.offsetTop;**  **在不更改样式定位的情况下**  **换成这个已知量就解决这个问题var y = getPage(e).pageY - slider.offsetTop;**  scrollbox.onmousemove = function (e) {  var scrollY = **getPage(e).pageY - y;**  slider.style.top = scrollY + 'px';  }  }  scrollbox.onmouseup = function () {  scrollbox.onmousemove = null;  }  </script>  </body>  </html>  拖拽每次都是一个独立的过程，不会对下一次有影响的  1 |

5、匀速动画函数

函数封装，可以利用javascript的动态特性，随时添加属性，存储一些值。

1、先把元素、移动的距离封装成函数

|  |
| --- |
| var timeFlag = null;// 这个是全局变量需要搭配使用  function animateMoving(element, distance) {  /\*if (timeFlag) {  clearInterval(timeFlag);  timeFlag = null;  }它没执行完就清掉？它是这样的，它每次重新创建一个新的定时器，继续启动移动盒子，上一次停止的时候，函数内的值是全局变量，所有会从上一次的位置继续移动。  如果按照以下的判断的话，那么这段程序只能触发一次，想设置其他距离不行\*/  if (timeFlag === null) {  // 让多次点击不会重复创建新的定时器  timeFlag = setInterval(function () {  var speed = 10;  if ((box.offsetLeft + speed) >= distance) {  box.style.left = distance + 'px';  clearInterval(timeFlag);  return;  }  box.style.left = box.offsetLeft + speed + 'px';  console.log(box.style.left);  },30);// 30毫秒，后续看具体要求再看  }  } |

2、在对象中添加记录定时器的属性，这样函数才能给多个对象使用，否则它们之间就只能一个执行。

|  |
| --- |
| <style type="text/css">  **/\*\* {**  **padding: 0;**  **margin: 0;**  **}\*/**  #box {  width: 200px;  height: 200px;  background-color: red;  position: relative;  }  #box1 {  width: 200px;  height: 200px;  background-color: pink;  position: relative;  }  </style>  <body>  <input type="button" value="开始移动400" id="btn">  <input type="button" value="开始移动800" id="btn1">  <div id="box"></div>  <div id="box1"></div>  <script type="text/javascript">  var btn = document.getElementById('btn');  var box = document.getElementById('box');  var box1 = document.getElementById('box1');  var btn1 = document.getElementById('btn1');  btn1.onclick = function () {  animateMoving(box, 800);  animateMoving(box1,800)  }  btn.onclick = function () {  animateMoving(box, 400);  animateMoving(box1,400);  }  function animateMoving(element, distance) {  var speed = 10;  var result;  if (element.timeFlag) {  clearInterval(element.timeFlag);  element.timeFlag = null;  }  if (element.offsetLeft > distance) {  // 有可能一开始从右到左移动，这样speed本来就是负的，不加绝对值就成正的了  // speed = -speed;  speed = -Math.abs(speed);  }  // 让多次点击不会重复创建新的定时器  element.timeFlag = setInterval(function () {  /\*if (speed > 0) {  result = element.offsetLeft + speed >= distance;  } else {  result = element.offsetLeft + speed <= distance;  }  if (result) {\*/  if (Math.abs(element.offsetLeft - distance) < Math.abs(speed)) {  element.style.left = distance + 'px';  clearInterval(element.timeFlag);  return;  }  element.style.left = **element.offsetLeft + speed + 'px';**  // console.log('left ='+ element.style.left);  },30);// 30毫秒，后续看具体要求再看  }  </script>  </body>  1  先打印出数据，然后分析跟数据相关的计算，问题就出现在那几行语句中 |
| 1、存在着，如果从位置800到位置400，速度看起来会变慢的原因，本来以为速度会跟400到800一样。这里无法直接使用left，因为left的获取会比较麻烦。  也是跟盒子的外边距是有关的，把样式清除一下就ok。  \* {  padding: 0;  margin: 0;  }  1 |

6、无缝轮播图

（1）动态生成序号，页面上总共有几张图片

（2）点击序号以动画的形式切换图片

a 取消其他li的高亮显示，让当前的li高亮显示

b 点击序号，以动画的方式切换到当前点击的图片位置。

（3）鼠标放到盒子上显示箭头，离开时候隐藏箭头

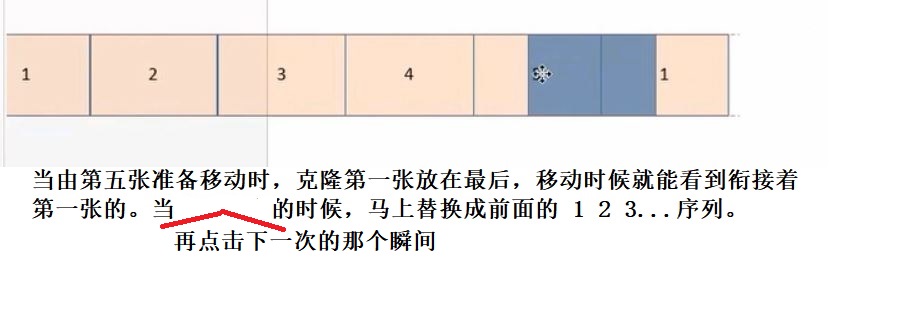
（4）实现上一张和下一张的功能

a 先实现下一张下一张的功能，上一张上一张的功能，用代码方式去触发序号的点击事件。

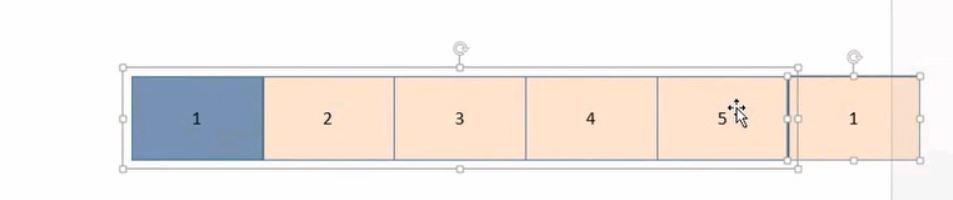
b 对第一张、最后一张无缝切换功能的实现。

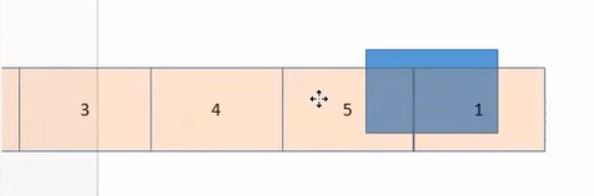
（5）自动播放功能实现

在第一张切切换到最后一张的时候，不需要通过经过其他四张图片，叫无缝轮播



第五张图切换到第一张图，表面那“第一张图”是第一张克隆出来的，应该是第六张，在继续点击下一张的时候，马上把ul 拉回原点，此时就是真正的第一张图，切换到第二张图。





当从第一张需要向左移动的，需显示的是第五张的衔接，在点击的一瞬间，马上切换到最后的克隆1那里去，此时就能无缝衔接起来了。

|  |
| --- |
| 总结：采用在图片最后面加上第一张图片的克隆   1. 在最后的一个图位置，切换的下一张的时候，是切换到克隆的第一张。再点击下一张，此时判断此图是克隆的，在移动之前，先马上把整个ul向右拉回原点left为0。然后去触发序号带的事件就ok。   **注：**克隆图没有序号去触发，所以只能按动画函数去移动，以及循环处理高亮序号。   1. 在第一个图位置，切换上一张，切换前马上把ul向左拉到克隆图片的显示，然后由克隆图片向最后一个图动画移动。 2. 自动播放，就是用定时器去定时的用代码去触发下一张的按钮事件。 |

（5）自动切换图片

**注：F12调试代码之后记得要把断点清除一下，否则执行不完整出现问题**

7、回到顶部

十七、附录

1、元素的类型

