JavaScript由三部分组成：

* ECMAScript (European Computer Manufacturers Association)
* ECMAScript 与Web浏览器没有依赖关系，Web只是ECMAScript 实现可能的宿主环境之一。利用ECMAScript 的核心类型和语法提供更多具体的功能，以便实现针对环境的操作。

ECMAScript 包括：语法、类型、语句、关键字、保留字、操作符、对象

* DOM (Document Object Model) 文档对象模型是针对XML但经过扩展用于HTML的API(Application Programming Interface)。DOM把整个页面映射为一个多层节点结构。借助DOM提供的API，开发人员可以轻松地删除、添加、替换或修改任何节点。
* BOM (Browser Object Model) 只处理浏览器窗口和框架：弹出新浏览器窗口、移动、缩放和关闭浏览器窗口；提供浏览器详细信息的navigator对象；提供浏览器所加载页面的详细信息的location对象；提供用户显示器分辨率详细信息的screen对象；对cookies的支持....

### 什么是DOM

* 作用: 可以把HTML文档模型化，当作对象来处理。
* HTML DOM 定义了用于 HTML 的一系列标准的对象，以及访问和处理 HTML 文档的标准方法。HTML DOM 独立于平台和编程语言。它可被任何编程语言诸如 Java、JavaScript 和 VBScript 使用。

### DOM内容概念

* 文档(document): HTML或XML文件
* 节点(node): HTML文档中的所有内容都可以称之为节点，常见的节点有**元素节点、属性节点、文本节点、注释节点**
* 元素(element): HTML文档中的标签及其包裹的内容可以称为元素

console.dir(变量名); 以对象的形式查看元素。

console.log(变量名); 把标签打印出来。

获取元素

|  |  |
| --- | --- |
|  | 获取到的是一个元素 |
| element.getElementsByClassName | 获取到的是一个**live** elements list  如果再往element上appendChild，这个list也会跟着更新 |
| element.getElementsByTagName |
| element.children |

注意：document.querySelector() 这个选择器得到的就**不会**时时更新，只有最初的数据。

getElements获得的是伪数组，IE8中伪数组没有push方法，要封闭兼容函数。伪数组有默认属性length，但是这个length是固定不变的，即使给它添加其他元素length也不会自动改变，所以要手动地length++.

var push = [].push;

push = {

apply: function (arr1, arr2){

for(var i = 0;i<arr2.length;i++){

arr1[arr1.length**++**] = arr2[i];

}

}

};

head / body / title不用获取可以用点语法直接使用：document.body

阻止a标签跳转：

1. <a onclick = “return false; ”>
2. <a href = javascript: ; ”>
3. <a href = javascript: ”> 在IE6中会造成gif有短暂的停顿
4. <a href = javascript: void(0); ”>
5. <a href = javascript: void(0) ”> 在IE6中会造成gif有短暂的停顿
6. 对象 . onclick = function( ){ ... ; return false; };
7. 事件源 . 事件 = function( ){ 事件处理程序 };

button.onclick = function (alert(“abc”) ){ };

1. 标签 事件 = “事件处理程序”<button onclick = “alert(‘abc’)”>在标签中书写的onclick称为标签的属性；当标签通过document.getElementById() 转换成对象后，onclick 就变成了对象的属性。对象的属性赋值是函数的时候，称其为方法。

input.onclick是在给按钮绑定事件，后面的函数是绑定的事件，函数在调用时都会执行。

区别：

|  |  |
| --- | --- |
| 凡是在for循环的内部中有函数，函数中就不要使用for循环中的变量，要用this. | 在函数内部使用for循环，变量照样可以使用，不受限制 。 |

想要结束函数的执行，用return.

innerText获取 / 设置标签内的普通文本，就像copier，写什么就呈现什么；

box.innerText = "&lt;div>I am div</div>"; 1

innerHTML是获取 / 设置标签内的标签。

|  |  |
| --- | --- |
| js语句 | 解析为 |
| box.innerHTML = "<div>Good.</div>"; | Good. |
| box.innerHTML= "&lt;div>Good.</div>"; | <div>&lt;div&gt;Good.</div> |
| box.innerHTML = "<div>3<5</div>"; | 3<5 |

innerHTML也可以获取到内容，前提是标签里面的内容就是普通文本而不是符合语义的标签。

## innerHTML属性是”传统的“技术，比childNodes[0].nodeValue便捷。

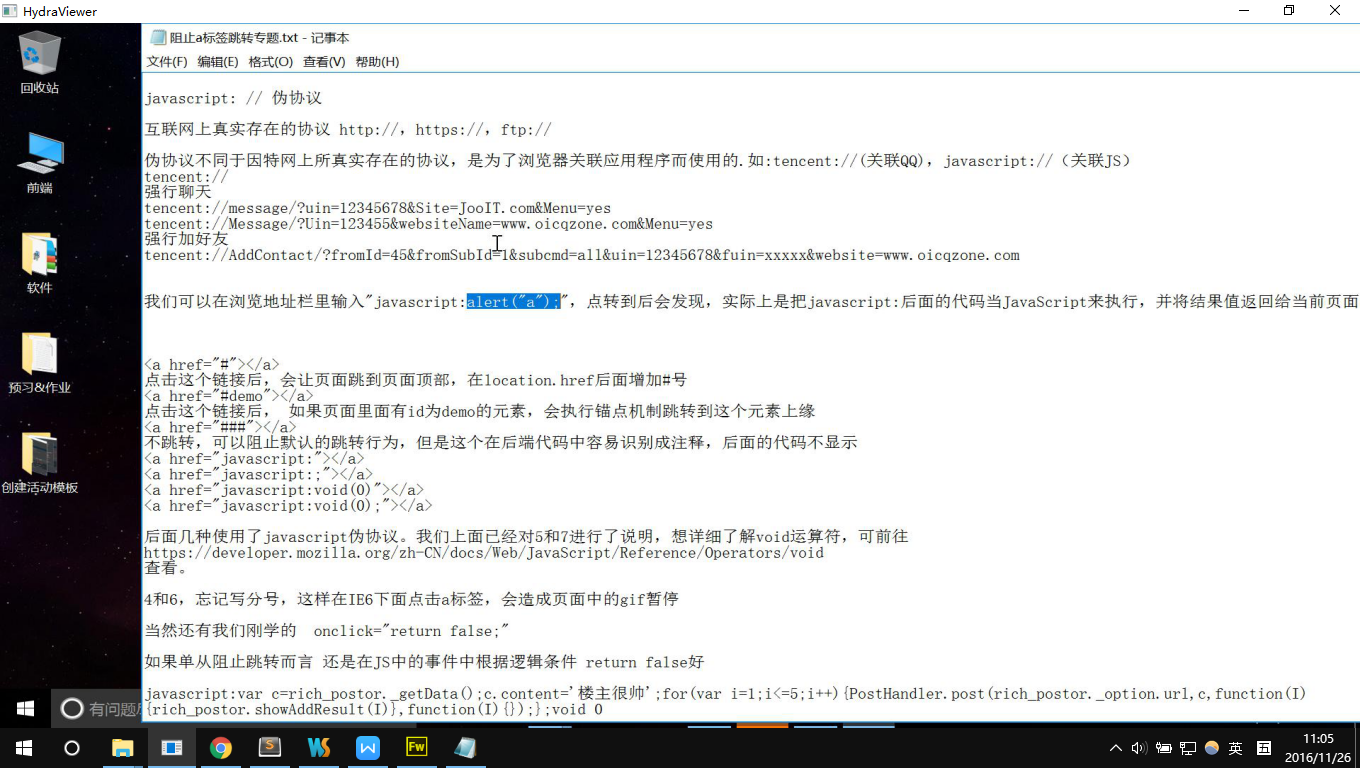
November27 文本框

1. 禁用文本框disabled
2. 用type属性批量禁用文本框
3. 批量设置和获取input里面的value属性，然后用数组把值拼成字符串(push)。join可以把数组中的字符串拼接，还可以传入分隔的参数。
4. placeholder
5. onfocus onblur 如果用户输入的与默认的一样怎么办？
6. select-option 皇上翻牌 selected= “selected”每次换不同的美女怎么办？
7. 表单验证：注册向后台发送数据

checked

slected

1. 全选、不选、反选checked，有一个不选就不选。
2. 排他思想：element.className = “ ”;
3. 点击标签，出现相应的列表:news, entertainment, sports



|  |  |  |  |
| --- | --- | --- | --- |
| **事件** | | | |
| onclick单击 | 双击 | onmouseover | onmouseout |
| onfocus | onblur |  |  |
|  |  |  |  |

自定义属性：预先没有规定的标签属性，无法映射到对象身上。IE678却可以。

setAttribute方法是DOM Level1的组成部分，它可以设置任意元素节点的任意属性。在DOM Level1出现之前，可以通过另一种办法设置大部分元素的属性，这个办法到现在仍然有效。

我个人更喜欢用setAttribute，起码不必费心去记忆哪些元素的哪些属性可以用DOM之前的哪些方法设置。DOM的另一个优势是可移植性好。老方法只适用于Web文档，DOM则适用于任何一种标记语言。

|  |  |
| --- | --- |
| 非DOM解决方案 | 等价的DOM |
| img.src = “” | img.setAttribute(“src”, “”) |
| element.value = “btn” | element.setAttribute(“value”, “btn”) |
| element.className = “” | element.setAttribute(“class”, “”) |

|  |  |  |
| --- | --- | --- |
| div.getAttribute(“id”); | setAttribute(“class”,“abc”); | removeAttribute(“id”); |
| 获取任意标签属性 | 设置任意标签属性 | 移除任意标签属性 |
| 预先规定的标签属性和自定义的标签属性都可以使用； | | |

标签和对象的自定义属性是不会相互影响的。

通过setAttribute对文档做出修改后，在通过浏览器的view source选项去查看源代码时，看到的仍然是**改变前**的属性值，也就是说，setAttribute做出的修改不会反映在文档本身的源代码里。这种”表里不一“的现象源自DOM的工作模式：先加载文档的静态内容，再动态刷新，动态刷新不影响文档的静态内容。这正是DOM的真正威力：对页面**内容**进行刷新却不需要在浏览器里刷新页面。

Nov 29

1. 隔行变色
2. 节点操作：ul.childNodes 找到所有的节点，是DOM标准中规定的方法，获取节点的方式所有浏览器都支持；

|  |  |  |  |
| --- | --- | --- | --- |
|  | nodetype | nodeValue | nodeName |
| 元素节点 | 1 元旦1月1日 | null | 大写的标签名 |
| 属性节点 | 2 二叔 | 属性值 | 属性名 |
| 文本节点 | 3 三文鱼 | 文本 | #text |
| 注释节点 | 8 猪八戒 | 注释中的内容 | #comment |
| 属性节点不是这个标签的子节点，类似“经纪人”。  var attrNode = ul**.**getAttributeNode (“id”) ; | | | |

<li>resources/赛马.mp3</li>

|  |  |  |
| --- | --- | --- |
|  | 获取到是的： | 类型是： |
| .firstChild | “resources/赛马.mp3” | object |
| .firstChild.nodeValue | resources/赛马.mp3 | string |

1. 只获取子元素：ul.children 不是DOM标准中规定的方法，因为很常用，所以浏览器都支持。
2. 随机翻牌
3. 表单验证：

onkeyup键盘按下，onkeydown键盘弹起，next sibling，nextElementSibling 获取下一个兄弟元素节点（IE678不兼容），element . previousSibling，element . previousElementSibling

1. 开灯：document.body document.head

console **.** log(document **.**title); 打印出来的是title里面的内容。

console.log(document.body.style); style是body的属性，这个属性也是对象。style对象的属性对应的就是样式属性，但是命名是JS的驼峰命名法。

通过style设置的是**行内**样式，通过style获取的也是行内样式。如果想要获取行间样式，要用window.getComputedStyle(selector)方法，例如：

window.getComputedStyle(DOM对象).height;

div.currentStyle.left或 div.current["left"]; //IE

数据类型是字符串，而且如果属性有单位，获取的也有单位，设置也要带单位。

设置样式属性的值，只要属性值，千万不要带分号。

因为设置的是行内样式，所以路径名要根据标签所在的文件去设置。

设置属性有两种方法：1是类名，2是style属性，有些需要计算的情况必须用后者。

### Opacity

概述：opacity指定了一个元素后面的背景被覆盖的程度。当opacity属性的值应用于某个元素上时，是把这个元素（包括它的内容）当成一个整体看待，因此该元素和它包含的子元素具有相同的透明度，哪怕它们的初始透明度并不相同。

/\*0-1 0全透明 1不透明 \*/  
/\*background-color: ;\*/  
  
/\*0-1 0全透明 1不透明 IE678不支持 子元素会继承透明度\*/  
/\*opacity: 0.5;\*/

1. rgba(255, 0, 0, 0.3)属性：0代表全透明，1代表完全不透明，子元素不能继承该属性，IE678不支持
2. opacity属性：0代表全透明，1代表完全不透明，子元素可以继承该属性，IE678不支持

Nov 30,2016

1. 克隆节点element **.** cloneNode(boolean);
2. 深层克隆true：把当前节点和内部所有的节点都克隆；浅层克隆false只克隆当前节点，克隆出来的节点就和之前的元素没关系了。
3. 追加节点：让父元素调用appendChild方法，father **.** appendChild(新节点)会追加到父元素的最后。如果是页面上原有的标签father **.** appendChild (原标签)会把它从原来的位置取出来放到新的位置。

为了加深对DOM的理解，请看下面的例子：

|  |
| --- |
| <body>  <div class="demo"></div>  <button>clone</button>  <div class="box"></div>  <script>  var demo = document.getElementsByClassName("demo")[0];  var box = document.getElementsByClassName("box")[0];  var btn = document.getElementsByTagName("button")[0];  btn.onclick = function() {  var clone = demo.cloneNode(true);  box.appendChild(clone); //box中的元素在不断地增加  }  </script>  </body> |
| var btn = document.getElementsByTagName("button")[0];  var clone = demo.cloneNode(true);  btn.onclick = function() {  box.appendChild(clone); //box中的子元素一直只有一个  } |

在例1中，每点击一次按钮，就会新克隆出来一个DOM元素，然后把这个新的克隆元素不断地放进盒子里；例2中，在点击之前，已经克隆好了一个，点击时就不会再克隆出新的元素了，所以每一次append都是一个先把已经放进去的拿出来再重新放回去的过程。每一次点击都在重复这个动作，box中的元素一直只有一个。这也就解释了如果没有克隆直接append的话，就会把该元素从原来的位置拿出来，然后放在新的位置，因为“星星还是那颗星星，月亮还是那个月亮”。

1. 在父元素father的某个子元素son之前插入新的子元素： father **.** insertBefore (新节点，son)
2. father **.** removeChild (son)
3. 许愿墙：随机生成15个纸箱，点击在最上显示，单击头部关闭纸条（一个是不显示，一个从结构上移除）
4. 动态创建结构：
5. innerHTML可以限定范围，用途广泛。如果想要追加，分两步，一是获取原本的innerHTML，二是拼接上新的符合标签规则的字符串。

box.innerHTML += “<ul><li>abc</li></ul>”;

1. 用createElement或createTextNode或cloneNode方法创建出来的节点只是JavaScript世界里的孤儿，只是保存在内存中。利用appendChild或insertBefore方法为它寻找父亲，才能呈现在浏览器窗口里。
2. reateTextNode(data) 可创建文本节点，data是字符串，可规定此节点的文本。
3. document **.** write (“ ”) ; write只能被document调用，可以将符合标签规则的文本变成标签，但是很难限定范围。

<input type="button" value = "val">

<div>aaa</div>

<script>

1 document.write("<div>bbb</div>");

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

input.onclick = function(){

2 document.write("<div style = 'background-color: pink'>ccc</div>");

}

</script>

第1行的div可以正常渲染，其位置与<script>出现的位置有关1；

第2行的write会把其他所有标签（包括style）都清掉，只留下自己，相当于重新打开一个页面重新渲染2。

1. 性能问题：1.大量拼接字符串；2.字符串转换为文档对象；3.把对象挂接到DOM树上； 4.渲染引擎根据新的DOM树重新渲染（最耗时间）。

box.innerHtml = arr.join(""); 这样只渲染一次

1. 先document.createElement，然后box.appendChild。createElement只能被document调用。
2. str **.** indexOf(); 空字符串返回0，找不到返回-1。
3. 百度搜索bug的解决：
4. li重叠问题
5. 如果没有匹配项就不需要追加li
6. 文本框中如果是空字符串，也不需要追加li

点击按钮隐藏盒子的方法：

1. opacity: 0;
2. rgba (0, 0, 0, 0);
3. display: none;
4. visibility: hidden;
5. margin: 为负
6. 定位到页面外
7. width: 0; height: 0;
8. z-index 为负？
9. 直接把元素去掉
10. overflow:hidden

element **.** parentNode : 只读属性 当前节点的父级节点，只能是元素节点。

如果当前的文本框获取焦点，其他的自动失去焦点，一个浏览器只能有一个焦点。

元素.childNodes.length 元素的子节点的个数

元素.attributes.length 元素的属性的个数

元素.nodeType 元素的节点类型

元素.children.length 元素只读属性子节点列表的个数

console**.**table(data) 按上、下、清空，

console.clear

|  |  |  |
| --- | --- | --- |
| ul.childNodes | ul.children | appendChild |
| lastChild | lastElementChild | removeChild |
| firstChild | firstElementChild | insertBefore |
| previousSibling | previousElementSibling | getAttribute |
| nextSibling | nextElementSibling | removeAttribute |
| parentNode只能是父元素 | | |

document.createElement (“ ”) 和document.write (“ ”) 只能是document

元素.firstChild 只读属性，元素的第一个子节点

元素.firstElementChild 第一个元素类型的子节点

元素.lastChild

元素.lastElementChild

元素.nextSibling 下一个兄弟

元素.nextElementSibling 下一个兄弟元素节点

元素.previousSibling

元素.previousElementSibling 上一个兄弟节点

元素.parentNode : 只读 属性 当前节点的父级节点

元素.offsetParent : 只读 属性 离当前元素最近的一个有定位属性的父节点

元素.offsetLeft[Top] : 只读 属性 当前元素到定位父级的距离（偏移值）到当前元素的offsetParent的距离

style.width : 样式宽

clientWidth : 可视区宽 = 样式宽 + padding

offsetWidth : 占位宽 = 样式宽 + padding + border

元素.getAttribute(属性名称); 方法 获取指定元素的指定属性的值

元素.getAttributeNode

元素.removeAttribute(属性名称); 方法 移除指定的元素的指定的属性

document.createElement(标签名称); 创建元素

父级.removeChild(要删除的元素); 删除元素

父级.appendChild(要添加的元素) 方法 追加子元素

父级.insertBefore(新的元素，被插入的元素) 方法 在指定元素前面插入一个新元素

父级.replaceChild(新节点，被替换节点) 替换子节点

nodeType

nodeValue

nodeName

元素.childNodes.length 元素的子节点的个数

元素.attributes.length 元素的属性的个数

元素.nodeType 元素的节点类型

元素.children.length 元素只读属性子节点列表的个数

元素.firstChild 只读属性，元素的第一个子节点

元素.firstElementChild 第一个元素类型的子节点

元素.lastChild

元素.lastElementChild

元素.nextSibling 下一个兄弟

元素.nextElementSibling 下一个兄弟元素节点

元素.previousSibling

元素.previousElementSibling 上一个兄弟节点

元素.parentNode : 只读 属性 当前节点的父级节点

元素.offsetParent : 只读 属性 离当前元素最近的一个有定位属性的父节点

元素.offsetLeft[Top] : 只读 属性 当前元素到定位父级的距离（偏移值）到当前元素的offsetParent的距离

style.width : 样式宽

clientWidth : 可视区宽 = 样式宽 + padding

offsetWidth : 占位宽 = 样式宽 + padding + border

元素.getAttribute(属性名称); 方法 获取指定元素的指定属性的值

元素.getAttributeNode(获取元素节点)

元素.removeAttribute(属性名称); 方法 移除指定的元素的指定的属性

document.createElement(标签名称); 创建元素

父级.removeChild(要删除的元素); 删除元素

父级.appendChild(要添加的元素) 方法 追加子元素

父级.insertBefore(新的元素，被插入的元素) 方法 在指定元素前面插入一个新元素

父级.replaceChild(新节点，被替换节点) 替换子节点

nodeType

nodeValue

nodeName

(MDN 网站查询)