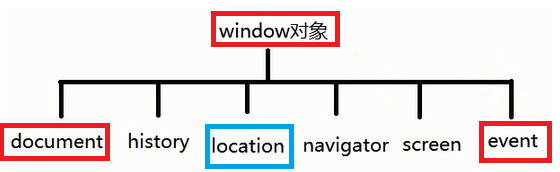
BOMAndDOM

1.BOM

Broswer Object Model  浏览器对象模型



window对象常用的属性和方法：

三个弹出框: 特点阻塞代码执行

alert("内容")   弹出框

confirm("提示信息")  确认框   确定返回true 取消返回false

prompt("提示信息","默认值")    输入框 返回数值为字符串

两个定时器：

    setInterval（执行任务，间隔时间）: 连续执行定时器  clearInterval()

  setTimeout(执行任务,间隔时间):  用法同上 只执行一次  延时性 clearTimeout() 停止定时器(一般用的不太多)

案例：延迟关闭

<html>

    <head>

        <meta charset="utf-8" />

        <title></title>

        <style>

            #box{

                width: 200px;

                height: 200px;

                background-color: aqua;

            }

        </style>

    </head>

    <body>

        <input type="button" value="开始" id="begin">&nbsp;&nbsp;

        <input type="button" value="停止" id="end"></br>

        <div id="box"></div>

    </body>

</html>

<script>

    var d = document.getElementById("box");

    var b = document.getElementById("begin");

    var e = document.getElementById("end");

    var t = null;//注意全局变量定时器的作用域

    b.onclick = function(){

        t = setTimeout(function(){

            d.style.display = "none";

        },3000);

    }

    e.onclick = function(){

        clearTimeout(t);

    }

</script>

一个弹出窗口方法: open("路径","窗口的名称没有什么意义"，"弹出窗口的外观") : 返回值是弹出的子窗口

外观主要考虑:width height left top

关闭窗口 close()

//在主窗口中打开子窗口

open("sub.html","heihei","width=200,height=200,left=200,top=200");

//关闭当前窗口

close();

一个事件：

onload事件:onload 事件会在页面或图像加载完成后立即发生。

window.onload = function(){

        console.log("页面加载完成");

    }

    console.log("页面正在加载");

下面代码有什么问题?

<!DOCTYPE html>

<html>

    <head>

        <meta charset="utf-8">

        <title></title>

        <script>

            var d = document.getElementById("t");

            d.innerHTML = "666";

        </script>

    </head>

    <body>

        <p id="t"></p>

    </body>

</html>

两种解决方案:

1.

window.onload = function(){

                var d = document.getElementById("t");

                d.innerHTML = "666";

            }

2.

<script type="text/javascript">

     var d = document.getElementById("t");

    d.innerHTML = "666";

</script>

window对象的属性和方法都可以省略window, 直接使用

所有全句变量都是window对象的属性

所有全局函数都是window对象的方法

    var num = 666;

    document.write(window.num);

    function fun(){

        document.write("heihei");

    }

    window.fun();

2.location 地址对象

页面跳转：

href属性：代表当前网页地址

location.href  可读写

    //当前页面地址 读

    document.write(location.href);

    //页面在三秒后跳到外网

    setTimeout(function(){

        location.href = "http://www.baidu.com";

        },3000);

location.replace("url")

//location.replace("index.html");

location.replace("http://www.baidu.com");   覆盖原先网页 没有浏览痕迹

页面刷新：

location.reload();

<input type="text">

<button onclick="fun()">刷新</button>

    function fun(){

        location.reload();

    }

3.history

history对象包含用户(在浏览器窗口中)访问过的 URL。

方法 ：

back() 加载 history 列表中的前一个 URL。

forward() 加载 history 列表中的下一个 URL。

go() 加载 history 列表中的某个具体页面，或者要求浏览器移动到指定的页面数量（负数为后退，正数为前进）

案例：不同页面间的跳转

        1.</br>

        <a href="2.html">向后</a>

        <button onclick="history.forward()">下一页</button></br>

        <button onclick="history.go(1)">跳转</button></br>

        2.</br>

        <a href="3.html">向后</a></br>

        <a href="1.html">向前</a></br>

        <button onclick="history.forward()">下一页</button></br>

        <button onclick="history.back()">上一页</button></br>

        3.</br>

        <a href="2.html">向前</a></br>

        <button onclick="history.back()">上一页</button></br>

4.document 文档对象  作用 找对象

每个载入浏览器的 HTML 文档都会成为 Document 对象。

Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问

Document 对象是 Window 对象的一部分，可通过 window.document 属性对其进行访问

document.write("<font color='red'>"+ 123 +"</font>"); write方法带解析功能

document.write() 向文档写 HTML 表达式 或 JavaScript 代码  注意：该方法在有事件的情况下会覆盖掉全页面

<button onclick="fun()">按钮</button>

function fun(){

        document.write("下课啦");

}

getElementById() 返回对拥有指定 id 的第一个对象的引用。

getElementsByName() 返回带有指定名称的对象集合。

getElementsByTagName() 返回带有指定标签名的对象集合。

getElementsByClassName（） 通过类名返回

querySelector()  通过选择器

<!DOCTYPE html>

<html>

    <head>

        <meta charset="utf-8">

        <title></title>

    </head>

    <body>

        <input type="text" name="txt" value="123"/></br>

        <input type="text" name="txt" value="456"/></br>

        <input type="text" name="txt" value="777"/></br>

        <input type="text" name="txt" value="888"/></br>

    </body>

</html>

<script>

    var a = document.getElementsByName("txt");

    var str = " ";

    for(var i=0; i<a.length; i++){

        str += a[i].value;

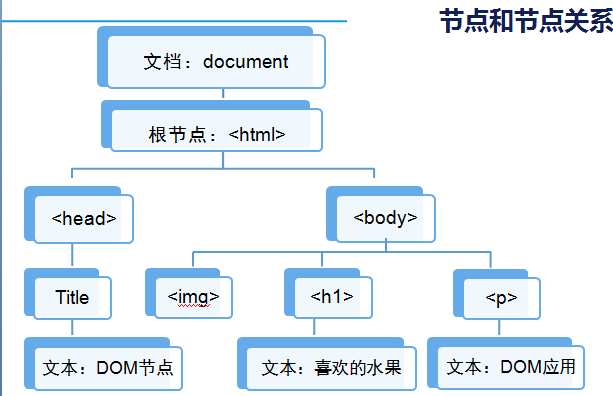
    }

    document.write(str);

</script>

 5.DOM

DOM（Document Object Model），文档对象模型。

****

根据层次关系访问节点： (包括文本和元素)

parentNode 返回节点的父节点

childNodes 返回子节点集合，数组名[i]访问子节点

firstChild 返回节点的第一个子节点，最普遍的用法是访问该元素的文本节点

lastChild  返回节点的最后一个子节点

nextSibling 下一个节点

previousSibling 上一个节点

<body id="bd">

        <p>你好!</p>

        <ul>

            <li>1</li>

            <li>2</li>

            <li>3</li>

            <li>4</li>

        </ul>

    </body>

</html>

<script>

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

    var str = a.firstChild.nextSibling.innerHTML;

    document.write(str);

</script>

通过层级关系访问元素节点

firstElementChild 返回节点的第一个子节点，最普遍的用法是访问该元素的文本节点

lastElementChild 返回节点的最后一个子节点

nextElementSibling 下一个节点

previousElementSibling 上一个节点

var str = a.firstElementChild.innerHTML;

节点类型：

通过nodeType属性来判断节点类型

1代表元素节点

2代表属性节点

3代表文本节点

<p id = "\_p">123</p>

var p = document.getElementById("\_p");

    //元素类型     1

    document.write(p.nodeType);

    //文本类型

    document.write(p.firstChild.nodeType);//3

    //属性节点

    document.write(p.getAttributeNode("id").nodeType);//2

6.节点的操作

document.createElement(HTML标签名)   //创建一个元素节点

document.createTextNode(文字)    //创建一个文本节点

node.appendChild(newChild)   //newChild 被添加到孩子列表中的末端。

node.insertBefore(newChild, referenceNode) // 将 newChild 节点插入到 referenceNode 之前。

node.removeChild(oldChild)   //删除 oldChild子节点。

node.replaceChild(newChild, oldChild)  //用newChild节点替换oldChild节点

1.先创建

2.在连接

创建节点:

//创建节点

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

e.innerHTML = "春晓";

//连接节点

document.body.appendChild(e);

var e1 = document.createElement("p");

e1.innerHTML = "春眠不觉晓";

document.body.appendChild(e1);

删除节点：

    <body>

        <ul id="list">

            <li>1</li>

            <li>2</li>

            <li>3</li>

            <li>4</li>

            <li>5</li>

            <li>6</li>

            <li>7</li>

        </ul>

    </body>

    <input type="button" onclick="del()" value="删除"/>

</html>

<script>

    function del(){

        var e = document.getElementById("list");

        //判断该节点是否有子元素  遍历

        if(e.hasChildNodes()){

            //删除最后一个元素

            e.removeChild(e.lastElementChild);

        }

    }

</script>

节点.remove(无参) 直接删除 以后这个用的更多一些

list.lastElementChild.remove();

作业

0.课堂案例，整理知识点

1.广告倒计时（自动关闭） //必须

2.动态向列表中插入节点  //必须

3.按钮的动态交互   ele[0].checked    img.setAttribute("src","../img/dog.jpg");  //尽可能

4动态创建表格      //尽可能

5.动态删除一行表格    //不要求

7.表格的动态创建

    var t = document.createElement("table");

    t.border = "1px";

    for(var i=0; i<3; i++){

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

        for(var j=0; j<3; j++){

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

            td.innerHTML = parseInt(Math.random()\*100)\*9+1;

            tr.appendChild(td);

        }

        t.appendChild(tr);

        //添加一行

        var b = document.createElement("td");

        b.innerHTML = "删除";

        tr.appendChild(b);

        b.onclick = function(){

            //this代表当前点击的对象

            this.parentNode.remove();//remove方法无惨，谁调用它删除谁

        }

    }

    document.body.appendChild(t);

this关键字：调用当前函数的对象本身。

8.childNodes/过滤空白节点

childNodes返回的是所有的孩子节点，但是实际孩子节点的长度并不是我们想要的那样

        <ul id = "list">

            <li>1</li>

            <li>2</li>

            <li>3</li>

        </ul>

    var e = document.getElementById("list");

    var a = e.childNodes;

通过观察a的长度，我们发现length为7，原因是方法childNodes返回了文本Text

过滤文本节点:

    for(var i=0; i<a.length; i++){删除

                //如果节点为文本节点，则根据nodeType

        if(a[i].nodeType == 3){

            a[i].remove();

        }

    }

    for(var i=0; i<a.length; i++){

        document.write(a[i].innerHTML + " ");

    }

9. outerHTML/innerText/innerHTML

简单的说innerHTML和outerHTML、innerText的不同之处在于：

innerHTML

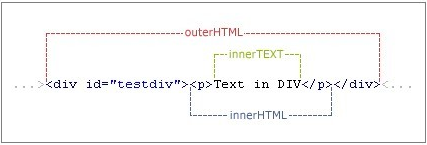
将元素中所有的内容都获取到 包括HTML标签 但是不包括自身标签

innerText

将元素的内容获取出来不包括HTML标签

outerHTML

将自身以及子元素所有的内容都获取出来 包括HTML标签 包括自身标签



   <body >

        <div id="d">

            老王<span>住你隔壁</span>

        </div>

    </body>

</html>

<script>

    var x  = document.getElementById("d");

    console.log(x.innerHTML);

    console.log(x.innerText);

    console.log(x.outerHTML);

</script>

10.自定义属性及getAttribute方法

<div id="d"></div>

//setAttribute("属性名称","属性值名称")

//getAttribute("属性名称") : 返回属性名称对应的属性值

//我们可以通过setAttribute设置自定义属性

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

    d.setAttribute("heihei",123);

    document.write(d.getAttribute("heihei"));

案例：按钮交互改变元素属性

<style>

            \* {

                font-size: 12px;

                font-family: "微软雅黑",arial;

                line-height: 25px;

            }

            div {

                padding: 5px;

                text-align: center;

            }

            div span{

                display: block;

            }

</style>

<body>

        <p>

            选择你喜欢的书:

            <input type="radio" name="myImg" onclick="book()" /> 五年高考

            <input type="radio" name="myImg" onclick="book()"/>  三年模拟

        </p>

        <div id="">

            <img id = "mimg"/>

            <span></span>

        </div>

</body>

    var r = document.getElementsByName("myImg");

    var oImg = document.getElementById("mimg");

    function book(){

        if(r[0].checked){

            //两种访问方式

            //oImg.setAttribute("src","img/1.jpg");

            oImg.src = "img/1.jpg";

            oImg.nextElementSibling.innerHTML = "你是个傻子";

        }else if(r[1].checked){

            oImg.setAttribute("src","img/2.jpg");

            oImg.nextElementSibling.innerHTML = "我是个傻子";

        }

    }

11. 获取非行内样式（兼容问题）

两个概念：行间样式与非行间样式

行间样式：行间样式就是写在标签体内的样式：如，<div style="color:red"></div>，在这里color样式就是行间样式。

非行间样式：非行间样式就是指不是写在标签体内的style属性的样式。如<style>....</style>内的定义的样式或引用的外部css文件都是非行间样式。

<div id="box" style="color: red;"> </div>

var d = document.getElementById("box");

        //获取行内样式

document.write(d.style.color);

#box{

     color: blue;

}

var d = document.getElementById("box");

//获取非行内样式

document.write(d.style.color); 返回空

----------------------------------------

#box{

     color: red;

     font-size: 22px;

}

//照抄  兼容写法

var d = document.getElementById("box");

function getStyle(obj,attr){    //获取非行间样式，obj是对象，attr是值

   if(obj.currentStyle){       //针对ie获取非行间样式

       return obj.currentStyle[attr];

   }else{

       return getComputedStyle(obj,false)[attr];   //针对非ie

   };

};

document.write(getStyle(d,"color") + "</br>");

document.write(getStyle(d,"font-size"));

12.insertBefore：添加到一个节点前面

父节点.insertBefore(要添加的子节点,参照节点);

参照节点如果为null 那就和appendChild()一样

<html>

    <head>

        <meta charset="utf-8">

        <title></title>

    </head>

    <body>

        <ul id="list">

            <li>槑槑</li>

            <li>晶晶</li>

            <li>叕叕</li>

        </ul>

    </body>

</html>

<script>

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

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

    o.innerHTML = "淼淼";

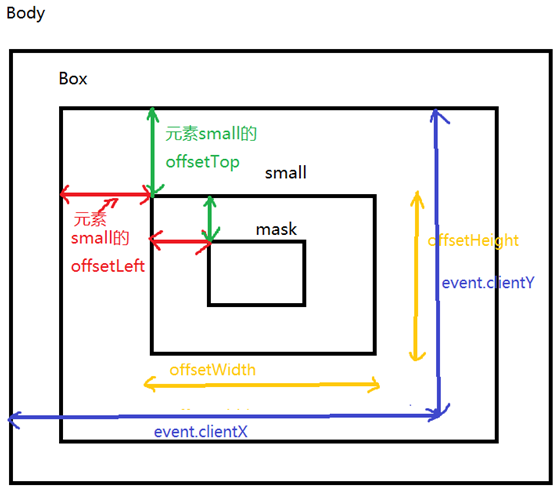
    //如果参数2位null，则和appendChild一样

    //list.insertBefore(o,null);

    list.insertBefore(o,list.children[2]);

</script>

13.offsetWidth/offsetHeight/offsetLeft/offsetTop



        <style>

            \*{

                margin: 0;

                padding: 0;

            }

            #big{

                border: 1px solid red;

                position: relative;

                height: 400px;

                width: 400px;

            }

            #small{

                border: 1px solid red;

                position: absolute;

                height: 100px;

                width: 100px;

                left: 80px;

                top: 70px;

            }

        </style>

    </head>

    <body>

        <div id="big">

            <div id="small">

            </div>

        </div>

    </body>

</html>

<script>

    var d = document.getElementById("small");

    //只读

    document.write(d.offsetLeft + " " + d.offsetTop + "</br>");

    //这2是哪来的?

    document.write(d.offsetWidth + " " + d.offsetHeight);

    //d.style.left = 200 + "px";

style.left返回字符串 如20px, offsetLeft返回数值 如20

style.left读写,offsetLeft只读

style.left必须要事先定义，否则值为空

//style.height style.width style.top 也一样

14.window.onscroll事件

回到顶部案例:

    <body>

        <div id="t">

            <p>111111111111111</p>

                        ....

            <p>111111111111111</p>

        </div>

        <button onclick="fun()">按钮</button>

    </body>

</html>

<script>

    var \_top = null;

    window.onscroll = function(){

        //兼容问题  获取滚动条移动的垂直距离

        \_top = document.body.scrollTop || document.documentElement.scrollTop;

        console.log(\_top);

    }

    function fun(){

        document.body.scrollTop = document.documentElement.scrollTop = 0;

    }

</script>