**DOM**

1. **Node关系**

* NodeList是有生命的、会呼吸的对象，而不是第一次访问就拍摄下来的快照。

Var first = someNode.childNodes[0];

Var first = someNode.childNodes.item(0);

Var count = someNode.length;

firstChild、lastChild、previousSibling、nextSibling

* nodeList不是数组

function convertToArray(nodes) {

var array = null;

console.log(Object.prototype.toString.call(nodes));// [object NodeList]

try{

array = Array.prototype.slice.call(nodes , 0);//非ie

} catch(ex){

array = new Array();

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

array.push(nodes[i]);

}

}

return array;

}

var divs = document.body.childNodes;

console.log(Object.prototype.toString.call(convertToArray(divs)));

// [object Array]

* 节点操作

appendChild、insertBefore(第二个参数传null和appendChild一样)、removeChild、replaceChild、cloneNode

cloneNode传true和false区别：false单独克隆这个节点，true克隆这个节点以及其子代。

1. **Document类型**

* nodeType值为9
* nodeName的值为”#document”
* nadeValue的值为null
* 子节点有doctype、html
* Html元素后面的注释不会列入nodelist
* documentElement是html标签，body是body标签，title是title标签，URL是完整URL，domain是域名，referrer是取得来源网页的URL
* document.getElementsByTagName('\*')得到所有节点
* document.open

<script type="text/javascript">

function createNewDoc()

{

var new\_doc = document.open("text/html","replace");

var txt = "<html><body>这是新的文档</body></html>";

new\_doc.write(txt);

new\_doc.close();

}

</script>

<body>

<button onclick="createNewDoc()">点击写入新文档</button>

</body>

1. **Element**

* nodeType的值为1
* nodeName的值为元素的标签名
* nodeValue为null
* 访问元素的标签名可以通过Element.tagName或者Element.nodeName，等价，得到大写的标签名
* 通过Element .getAttribute（“”）可以访问自定义属性，而Element.属性名，不行，一般只在取自定义属性时用getA方法
* 直接给元素设置自定义属性是不会成为元素的特性的，要用setAttribute。removeAttribute彻底删除属性
* Attributes属性中包含了NamedNodeMap，是一个动态集合，包含了一系列节点，nodeName就是属性名，nodeValue就是属性的值。有方法getNamedItem，removeNamedItem（效果和removeAttribute一样），setNamedItem（传一个参数），item（返回节点的位置）。

div1.attributes.**getNamedItem('id')**.nodeValue === div1.attributes**['id']**.nodeValue

* 一般来说只用Attributes属性来遍历元素的特性。

function printAttr(element) {

var pairs = new Array();

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

if(element.attributes[i].specified){//兼容ie

var attrName = element.attributes[i].nodeName,

attrValue = element.attributes[i].nodeValue;

pairs.push(attrName+"=\""+attrValue+"\"")

}

}

console.log(pairs.join(" "));

}

1. **Text类型**

* nodeType值为3
* nodeName是“#text
* nodeValue是包含的文本
* 没有子节点”
* div1.childNodes[0].nodeValue 和div1.firstChild.nodeValue都可以取到文本节点并修改。
* Document.createTextnode(“文本内容”)创建文本节点，相连文本节点可以用normalize()连接

1. **Comment类型**

* nodeType为8
* nodeName为“#comment”
* nodeValue是注释的内容
* 没有子节点
* 与text类似可以通过nodeValue或data属性访问

1. **DocumentType**

* nodeType值为10
* nodeName为doctype的名词
* nodeValue没有
* 没有子节点
* 保存在Document.doctype中，通过Document.doctype.name能访问到<!DOCTYPE html>中的html文本

1. **DocumentFargment**

* nodeType为11
* nodeName为“#document-fragment”
* nodeValue为null
* parentNode为null

1. **Attr类型**

* nodeType为2
* nodeName是特性的名词
* nodeValue是特性的值
* parentNode没有
* 没有子节点
* 虽然是节点但是不被认为是DOM文档树的一部分。通常使用getAttribute、setAttribute、removeAttribute操作节点

1. **DOM扩展**

* 呈现模式：区别标准模式还是混杂模式

如果是标准模式Document.compatMode值为“CSS1Compat”

如果是混杂模式Document.compatMode值为“BackCompat”

* 滚动

scrollIntoView()：滚动窗口以便在视口中看到。如果传入true或者忽略，会尽可能滚到元素顶部。所有浏览器兼容

只有Safari和chrome实现：

scrollIntoViewIfNeeded()：只在视口中不可见的情况下让元素可见！如果传入true则尽量在视口中部显示

scrollByLInes()：将元素的内容！滚动到指定行数的高度。

scrollByPages()：将元素的内容！滚动到指定页面的高度。

* Children属性

IE中的childNodes不包含回车，其他浏览器都包含。所以又引入children属性。在子节点都是元素节点的时候，children属性和childNodes是相同的，除了Firefox外，其他浏览器都支持children集合，Ie中的children集合会包含注释节点

* Contains方法

用于检测传入的节点是不是调用函数的节点的后代。但是Firefox不支持这个方法，firefox在dom3中提供了compareDocumentPosition（）方法，返回1：无关，2：居前，4：居后，8：父类，16：子孙类

兼容写法

function contains(refNode, otherNode) {

if(typeof refNode.contains == "function" && (!client.engine.webkit || client.engine.webkit >= 522)){

return refNode.contains(otherNode);

} else if(typeof refNode.compareDocumentPosition == "function"){

return !!(refNode.compareDocumentPosition(otherNode) & 16);

} else {

var node = otherNode.parentNode;

do{

if(refNode === node){

return true;

} else {

node = node.parentNode;

}

}while (node !== null);

return false;

}

}

* **操作内容**

innerText属性可以得到节点内的所有文本包括子孙，也可以设置节点的文本。Firefox不支持innerText但是有textContent方法，所以兼容写法：

function getInnerText() {

return (typeof element.textContent == "string") ?element.textContent : element.innerText;

}

innerHTML属性和innerText类似，得到所有子html，包括元素、注释、文本。但是不能动态添加script代码和style，添加方式仅IE，firefox支持且麻烦，用document.createElement(“script”);添加，

outerText，outerHTML调用的时候包括调用它的标签。读的结果和inner一样，写的时候会代替掉调用它的标签。

在插入大量标签时，innerHTML比多次DOM操作好，再设置innerHTML时会创建一个HTML解析器，运行速度比js快得多，但是多次创建和销毁解析器有损性能，所以也要控制innerHTML的使用次数。

**DOM操作技术**

1. **动态脚本**

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

script.type = "text/javascript";

script.innerText = "alert(1);";//或者使用appendChild(document.createTextNode("alert(1);"))IE低版本不兼容，使用script.text, Safari3.0以下不兼容

document.getElementsByTagName("body")[0].appendChild(script);

eval("alert(1);");

1. **动态样式**

和脚本类似，不过没有script.text类似的属性，可以用style.styleSheet.cssText=””;

1. **nodelist**

nodelist是实时的所以

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

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

divs.appendChild(div);

}//是死循环应该用for (var i = 0，len= divs.length; i < len; i++)

HTMLCollection、NodeList、NamedNodeMap

var divs = document.getElementsByTagName("div");

alert(divs instanceof HTMLCollection); //true

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

var children = div.childNodes; //获取div元素子节点集合

alert(children instanceof NodeList); //true

var attrs = div.attributes; //获取div元素的特性

alert(attrs instanceof NamedNodeMap); //true