## DOM

DOM1定义了一个Node接口。这个接口是作为Node类型来实现的。除了IE不能访问Node类型外，其他留言器都能访问Node类型。

### nodeType

任何节点都有一个nodeType属性。用于表述节点的类型。节点类型一共12种之多，但是我认为平时常用的dom节点类型也就4种。

一下是所有的节点类型，任何节点类型都必居其一：

（Tip：这里Node类型在IE中是无法访问的。稍后会说更普遍的方法，即使是IE也可以兼容）

*Node*.ELEMENT\_NODE // 1 \*元素节点

*Node*.ATTRIBUTE\_NODE //2 \*属性节点

*Node*.TEXT\_NODE //3 \*文本节点

*Node*.CDATA\_SECTION\_NODE //4

*Node*.ENTITY\_REFERENCE\_NODE // 5

*Node*.ENTITY\_NODE // 6

*Node*.PROCESSING\_INSTRUCTION\_NODE //7

*Node*.COMMENT\_NODE // 8 \*注释节点

*Node*.DOCUMENT\_NODE // 9

*Node*.DOCUMENT\_TYPE\_NODE //10

*Node*.DOCUMENT\_FRAGMENT\_NODE // 11

*Node*.NOTATION\_NODE // 12

以上的内容都是常量，他们的值就是数字，元素节点的就是1。这也就是说我们可以让他们作为固定的值来验证节点的类型。

例子：

<div *id* = "app">

<h1>欢迎来到软件研发中心</h1>

</div>

var somenode = document.getElementById("app");

if(somenode.nodeType == *Node*.ELEMENT\_NODE){

alert("这是个元素节点"); // 这是个元素节点

}

但是这各方式并不推荐，只适用于了解，因为在IE中，Node类型无法访问，因此会出现错误。应该如下判断。

if(*Node*.ELEMENT\_NODE == 1){

alert("这是个元素节点"); // 这是个元素节点

}

### nodeName和nodeValue

nodeName中始终保存着元素的标签名（大写的），nodeValue对于元素节点则始终是null。

### NodeList

文档中所有节点都存在着相互间的关系，比如父子，兄弟。。。

每一个节点的childNodes属性中都有一个NodeList对象，这个对象是一个类数组对象（因此不是数组），可以通过中括号语法访问NodeList中的节点成员，也可以通过item()方法来访问。当然也存在一些数组对象也存在的方法和属性，比如说length。

（Tip: item()方法不适用于数组）。

somenode.childNodes.item(0); 相当于 somenode.childNodes[0]

nodeList的一大特点是其中的内容是动态更新的，这更像是一个存在意识的“数组”，如果DOM结构发生变化，那么nodeList也会实时更新。因此nodeList并不是我们最初访问的“快照”。

nodeList中的内容是对应节点的索引。

有些时候，我们不希望他是快照，所以我们有必要把他变成一个数组。

变成数组的标准方式如下（Tip： 需要对IE进行特别处理，检验怪癖）。

function convertToArray(*nodes*){

try{

var arr = *Array*.prototype.slice.call(nodes,0);

} catch(err) {

var arr = **new** *Array*();

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

arr.push(nodes[i]);

}

}

return arr;

}

*console*.log("nodeList:" + someChildren.length); // 7

var arr = convertToArray(someChildren);

*console*.log("arr:" + arr.length); //7

*console*.log(somenodes.childNodes.item(1).parentNode.removeChild(somenodes.childNodes.item(1)));

*console*.log("nodeList:" + someChildren.length); // 6 nodeList更新了

*console*.log("arr:" + arr.length); // 7 因为成为了数组所以没变化。

### parentNode、firstChild、lastChild。hasChildNodes()

parentNode属性不多说，指的时父节点。

firstChild与lastChild分别对应childNodes[0]以及childNodes[length-1]。

previousSibling与nextSibling分别对应于前一个兄弟节点和后一个兄弟节点。当没有兄弟节点时，值为null。

hasChildNodes()是一个非常有用的方法，用来判断一个节点是否有子元素节点。这个方法在节点包含一个子节点或者多个子节点的情况下返回true，没有子节点则返回false。这比childNodes.length==0更方便。

所有节点都有一个最后的属性就是ownerDocument，该属性指向表示整个文档的文档节点，这种关系表示，任何节点都属于他所在的文档，任何节点都不能同时存在与两个或者多个文档中。通过这个属性，我们可以不必在层次节点中通过层层回溯到达顶端，而是可以直接访问文档节点。

*console*.log(somenodes.ownerDocument);



### 操作节点

因为关系指针都是只读的，所以DOM提供了一些操作节点的方法。其中最为常用的就是**appendChild()**。他用于向childNodes列表的最后插入新的子节点，参数当然是某个节点。

需要注意的是，任何DOM节点都不能同时存在于同一个文档中的多个位置，每个节点都是独一无二的（即使长得一样的也是不同的节点）。因此，如果想appendChild()的参数中带入一个已经存在于文档上的某个节点，那么这个节点就会移动到新的地方，而不是在指定位置重新生成一个新的节点。

如果需要将节点查到特殊的位置，就可以使用**insertBefore（）**方法（没有insertBefore方法，需要自己编写）。这个方法需要两个参数，通过父元素调用。第一个参数是需要插入的节点，第二个参数是参照节点。

替换节点需要使用replaceChild()方法。通过父元素调用这个方法，有两个参数，第一个参数是一个newElement,第二个参数是oldElement。注意这个方法并不是删除，也没有返回值。

如果只是想删除节点而不是替换，那么就可以使用removeChild()方法。通过父元素调用，接受一个参数，就是那个需要移除的节点。（Tip：并不是第几个子元素！）删除之后，这个方法返回被删除的那个元素。这个被删除的元素从此成为了“孤儿”。

值得注意的是，之前的增、删、改全是需要通过父元素进行调用。

如果需要克隆节点，就需要使用cloneNode()方法，被需要复制的节点所调用，返回一个被复制的节点。这个方法还接受一个参数，true的话就执行深复制，复制这个节点以及他的所有子节点树。如果是false则执行潜复制，只复制他本身。

（cloneNode并不会复制DOM节点的Js属性（事件处理就算一种属性），但会复制特性。但是IE存在bug，他会复制事件处理程序，因此在复制之前，最后移除事件处理程序）。