**DOM一些属性方法介绍以及兼容性问题**

DOM中Element类型：（以下所有的属性仅仅是只读属性）

一、node.nodeType 以数字值返回指定节点的节点类型。

存在 12 种不同的节点类型，其中可能会有不同节点类型的子节点(前三个为重要)：

if(ul.nodeType == Node.ELEMENT\_NODE){   
alert(‘Node is an element’);   
}   
上述代码，在标准浏览器下可以正常的运行，在非标准的浏览器（ie8之下）下不能正常的运行。   
解决办法：   
if(ul.nodeType ==1){ alert(‘Node is an element’); } 使用nodeType属性与数字比较。

二、nodeName值为元素类型的标签名比如ul、 p等。   
nodeValue值为null。   
parentNode可能是document或者Element，子节点可能。Element,Text,Comment,ProcessingInstruction, CDATASection, EntityReference。

操作特性

一、HTML元素属性的方式获得特性

所有HTML元素通过HTMLElement类型或者它的子类型（HTMLDivElement 、HTMLImageElement）表示，HTMLElement类型是继承自Element类型，只不过多加了一些特有的属性，切子类型也有自己特有的属性和方法，比如img有src，title等特有属性。特别注意的是由于class是js中的关键字，要使用className获得class 的值。

对于获取和设置方式以及注意地地方在二三中通过对比的方式表述。

二、element.getAttribute(attributename)返回指定属性名的属性值。

这个方法传入一个属性名的字符串，不区分大小写。   
比如ul.getAttribute(‘class’);由于参数为字符串，所以可以使用class不必使用className。通过这个方法可以获得自定义属性，在一些含有非关键字中使用的字符的属性名，用这个方法获得属性的值特别方便，比如在HTML5规范中，自定义属性前应该加上data-前缀以方便验证，其中包含了非法字符‘-’,可以使用ul.getAttribute(‘data-index’);获得。

特别的，当attributename为style时这个方法在IE7之前的版本返回的是一个对象，其他版本的浏览器返回的是CSS文本。当attributename为onclick这样的事件处理程序时这个方法在IE7之前的版本返回的是一个方法或者null，其他版本的浏览器返回的是函数代码字符串。

通过HTML元素属性的方式获得特性与getAttribute（）差别比较大。   
1、自定义属性访问上的差别：   
解析html代码的时候，标准的浏览器不会将元素的自定义属性加入到DOM对象中，在js中通过属性的方式访问自定义属性是不存在的结果为undefined，非标准的浏览器会将这个自定义属性加入到DOM对象中，在js中会访问到结果。

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
</**head**>  
<**body**>  
<**ul id="ul1" index="hehe"**>  
 <**li**>11111111</**li**>  
 <**li**>22222222</**li**>  
 <**li**>33333333</**li**>  
 <**li**>44444444</**li**>  
</**ul**>  
  
<**script type="text/javascript"**>  
 **var *ul*** = **document**.getElementById(**'ul1'**);  
 alert(***ul***.**id**); */\*标准与非标准都是div1\*/* alert(***ul***.**index**);*/\*标准为undefined 非标准为hehe\*/*</**script**>  
</**body**>  
</**html**>

2.当属性为style时，返回一个对象，当属性为处理事件时，返回被复制的函数。

综上所述，在操纵DOM对象属性的时候，一般情况下使用HTML元素属性的方式获得特性，只有在获得自定义属性的值和一些含有非法字符的属性名的值的时候使getAttribute方法。

三、element.setAttribute(attributename,attributevalue)添加指定的属性，并为其赋指定的值。

与HTML元素属性的方式设置特性的方式相比，这个方法可以添加一些没有的自定义属性并赋值，但通过HTML元素属性的方式设置特性不会被设置。同时这个方法在IE7之前的版本也存在异常，所以除了设置自定义属性外，其他方式通过HTML元素属性的方式设置特性。

四、RemoveAttribute()移除特性，但是不经常使用，IE6之前的版本不支持这个方法。

操作内容

一、childNodes：表示某个元素子节点的集合，返回NodeList 对象。

在标准浏览器下：返回的子节点中包含文本类型、元素类型、注释类型等，特别的包含文本类型中空文本即所谓的换行。   
在非标准的浏览器下：返回的子节点中不包含文本类型中空文本，同时与节点位置、解析方式等也有关。

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
</**head**>  
<**body**>  
<**ul id="ul1"**>  
 <**li**>22222</**li**>  
 111111  
 *<!--33333-->*</**ul**>  
<**script type="text/javascript"**>  
 **var *oul*** = **document**.getElementById(**'ul1'**);  
 alert(***oul***.**childNodes**.**length**);  
</**script**>  
  
在标准浏览器下：得到的结果为5（li元素前的换行空文本，li元素，注释到</li>之间这一段文本，注释，注释到</ul>的空白） 在非标准的浏览器下得到的结果为1 但是将<li>22222</li> 111111互换位置之后得到的结果为2.  
</**body**>  
</**html**>

所以当运行下面代码的时候为出问题：  
<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**meta charset="UTF-8"**>  
</**head**>  
<**body**>  
<**ul id="ul1"**>  
 <**li**>11111111</**li**>  
 <**li**>22222222</**li**>  
 <**li**>33333333</**li**>  
 <**li**>44444444</**li**>  
</**ul**>  
  
<**script type="text/javascript"**>  
 **var *oul*** = **document**.getElementById(**'ul1'**);  
 **for**(**var *i*** = 0; ***i***<***oul***.**childNodes**.**length**; ***i***++){  
 ***oul***.**childNodes**[***i***].**style**.**backgroundColor** = **'red'**;  
 }  
 */\*标准： Cannot set property 'backgroundColor' of undefined\*/  
 /\*非标准：（ie8之前）正常\*/*</**script**>  
</**body**>  
</**html**>  
原因很简单，就是因为在标准浏览器下childNodes返回的子节点中包含非元素类型的节点。

解决上述问题的方式：   
通过nodeType属性判断子节点是否为元素类型：

**for**(**var *i*** = 0; ***i***<oul.**childNodes**.**length**; ***i***++){  
 **if**(oul.**childNodes**[***i***].**nodeType** === 1)  
 oul.**childNodes**[***i***].**style**.**backgroundColor** = **'red'**;  
}

在非法嵌套的html文档中由于解析方式不同，在标准和非标准浏览器下呈现的结果不同，这完全于浏览器内核的解析方式有关。比如：

<!doctype **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"** />  
</**head**>  
<**body**>  
<**ul id="ul1"**>  
 <**li**>11111111</**li**>  
 <**li**>22222222</**li**>  
 <**li**>33333333</**li**>  
 <**li**>44444444</**li**>  
 <**p**>5555555</**p**>  
</**ul**>  
  
<**script type="text/javascript"**>  
 **var *oul*** = **document**.getElementById(**'ul1'**);  
 **for**(**var *i*** = 0; ***i***<***oul***.**childNodes**.**length**; ***i***++){  
 **if**(***oul***.**childNodes**[***i***].**nodeType** === 1)  
 ***oul***.**childNodes**[***i***].**style**.**backgroundColor** = **'red'**;  
 }  
</**script**>  
</**body**>  
</**html**>

对于一些分标准的浏览器比如ie7 解析的时候他会把这种那个不符合语义的p元素放到最后一个li中，但对于其他的浏览器并不会这样。换句话说，在标准浏览器下p是ul的子节点，但是在非标准的浏览器下p是最后一个li的子节点，这完全和浏览器内核的系欸小方式有关。

由此可以看出，对于在书写html文档的时候，结构化语义是多么的重要，至少能为添加js方便不少。

二、element.childern仅仅返回元素类型的子节点集合，返回NodeList 对象。

Children比childNodes要好得多，因为他仅仅获取那些为元素类型的子节点。但是还是不能免于非法嵌套带来的问题，这本身至于浏览器的近稀饭时有关，与用那种属性没有关系。

三、element.firstChild（firstElementChild）：获取第一个子点   
element.lastChild（lastElementChild）：获取最后一个子节点   
element.nextSibling（nextElementSibling）：获取相邻的下一个兄弟子节点   
element.previousSibling（previousElementSibling）：获取相邻的上一个兄弟节点

以firstChild为例说明一下性质，其他的大同小异。   
在标准情况下：返回的结果可能是文本类型（空文本）。   
在非标准的情况下：如果有元素子节点，那么一定返回第一个元素类型的子节点。

<!doctype **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"** />  
</**head**>  
<**body**>  
<**ul id="ul1"**>  
 <**li**>11111111</**li**>  
 <**li**>22222222</**li**>  
 <**li**>33333333</**li**>  
 <**li**>44444444</**li**>  
</**ul**>  
<**script type="text/javascript"**>  
 **var *oul*** = **document**.getElementById(**'ul1'**);  
 alert(***oul***.**firstChild**.**nodeName**);  
 */\*标准下text非标准为li\*/*</**script**>  
</**body**>  
</html>

解决这种情况   
1是firstElementChild。   
firstElementChild仅仅是在标准浏览器下有效，在非标准的浏览器下是没有定义的。在标准浏览器下仅仅返回第一个元素类型的子节点，如果没有返回null。   
再js中加入：

**function** *firstChild*(obj){  
 **if**(obj.**firstElementChild** === **undefined**){*//检测是否为标准浏览器* **return** obj.**firstChild**;*//不是标准浏览器，用firstChild返回第一个元素子节点，可能为null* }**else**{  
 **return** obj.**firstElementChild**;*//是标准浏览器，用firstElementChild返回第一个元素子节点，可能为null* }  
}  
  
**var *oFirst*** = *firstChild*(oul);  
**if**(***oFirst***){*//判断有没有第一个元素子节点排除空节点的情况* ***oFirst***.**style**.**backgroundColor** = **'orange'**;  
}**else**{  
 alert(**'没有第一个元素'**);  
}

2直接用children：推荐

**if**(oul.children[0]){  
 oul.children[0].**style**.**backgroundColor** = **'orange'**;  
}**else**{  
 alert(**'没有第一个元素'**);  
}

四、element.parentNode： 当前节点的父级节点，无兼容性问题。

Eg：点击隐藏将这个列表隐藏  
<!DOCTYPE **HTML**>  
<**html**>  
<**head**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"**>  
 <**title**>无标题文档</**title**>  
 <**script**>  
 **window**.onload = **function**() {  
 **var** aA = **document**.getElementsByTagName(**'a'**);  
 **for** (**var** i=0; i<aA.**length**; i++) {  
 aA[i].onclick = **function**() {  
 **this**.**parentNode**.**style**.**display** = **'none'**;  
 }  
 }  
 }  
 </**script**>  
</**head**>  
<**body**>  
<**ul id="ul1"**>  
 <**li**>11111 <**a href="javascript:;"**>隐藏</**a**></**li**>  
 <**li**>22222 <**a href="javascript:;"**>隐藏</**a**></**li**>  
 <**li**>33333 <**a href="javascript:;"**>隐藏</**a**></**li**>  
 <**li**>44444 <**a href="javascript:;"**>隐藏</**a**></**li**>  
</**ul**>  
</**body**>  
</**html**>

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

非ie7以下的浏览器：   
如果没有定位父级 offsetParent 为body   
如果有定位父级 offsetParent 为定位父级   
Ie7以下的浏览器：   
如果没有定位父级 自身没有定位 offsetParent 为body，自身有定位的话 为html   
如果当前元素的某个父级触发了layout，那么offsetParent就会被指向到这个触发了layout特性的父节点上。

如果有定位父级 offsetParent 为定位父级

<!DOCTYPE **HTML**>  
<**html**>  
<**head**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"**>  
 <**title**>无标题文档</**title**>  
 <**style**>  
 **div** {**padding**: 40**px** 50**px**;}  
 **#div1** {**background**: **red**;}  
 **#div2** {**background**: **green**; **zoom**: 1;**position**: **relative**;}  
 **#div3** {**background**: **orange**;}  
 </**style**>  
 <**script**>  
 **window**.onload = **function**() {  
  
 **var** oDiv3 = **document**.getElementById(**'div3'**);  
 alert( oDiv3.**offsetParent**.**id** );  
 }  
 </**script**>  
</**head**>  
  
<**body id="body1"**>  
<**div id="div1"**>  
 <**div id="div2"**>  
 <**div id="div3"**></**div**>  
 </**div**>  
</**div**>  
</**body**>  
</**html**>

Div2 zoom：1；属性触发layout，并且div3没有定位，所以在ie7一下的版本父元素div2，同时，div2相对定位，在其他浏览器中父元素为div2。解决了兼容性问题。

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

非ie7以下的浏览器：   
如果没有定位父级 offsetLeft [top]是到html的距离。   
如果有定位父级 是到定位父级的距离。   
ie7以下：   
如果自己没有定位，无论是否有没有父级定位offsetLeft[Top]是到body的距离   
如果自己有定位，那么就是到定位父级的距离（没有父级定位情况下是到html的距离）。

<!DOCTYPE **HTML**>  
<**html**>  
<**head**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"**>  
 <**title**>无标题文档</**title**>  
 <**style**>  
 **div** {**padding**: 40**px** 50**px**;}  
 **#div1** {**background**: **red**;}  
 **#div2** {**background**: **green**; **position**: **relative**;}  
 **#div3** {**background**: **orange**; **position**: **relative**;}  
 </**style**>  
 <**script**>  
 **window**.onload = **function**() {  
 **var** oDiv3 = **document**.getElementById(**'div3'**);  
 alert( oDiv3.**offsetTop** );  
 }  
 </**script**>  
</**head**>  
<**body id="body1"**>  
<**div id="div1"**>  
 <**div id="div2"**>  
 <**div id="div3"**></**div**>  
 </**div**>  
</**div**>  
</**body**>  
</**html**>

例子：获得任意一个元素的相对  
于页面的位置  
<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**style type="text/css"**>  
 **body**{**padding**:0; **margin**:0;}  
 */\*由于offsetLeft和offsetParsent在没有定位父级的时候父级不同\*/* **div**{**padding**: 40**px** 50**px**;}  
 **#div1**{**background-color**: **#008000**;}  
 **#div2**{**background-color**: **#FF0000**;**position**: **relative**;}  
 **#div3**{**background-color**: **#FFA500**;**position**:**relative**;}  
 */\*设置position offsetLeft在字节没有定位的时候是相对body的\*/* </**style**>  
</**head**>  
<**body**>  
<**div id="div1"**>  
 <**div id="div2"**>  
 <**div id="div3"**>  
 </**div**>  
 </**div**>  
</**div**>  
  
<**script type="text/javascript"**>  
  
 **var *oDiv*** = **document**.getElementById(**'div3'**);  
  
 **function** *getPosition*(obj){  
  
 **var** pos = {**left**:0, **top**:0};  
 **while**(obj){  
 pos.**left** += obj.**offsetLeft**;  
 pos.**top** +=obj.**offsetTop**;  
 obj = obj.**offsetParent**;  
 }  
 **return** pos;  
 }  
 alert(*getPosition*(***oDiv***).**left**);  
</**script**>  
</**body**>  
</**html**>

七、长高：   
element.style.width : 样式宽 = width   
element.clientWidth : 可视区宽 = width + padding   
element.offsetWidth: 占位宽 = width + padding + border

八、   
node.appendChild(node)   
node.insertBefore(newnode,existingnode)   
node.removeChild(node)   
node.replaceChild(newnode,oldnode) node为oldnode的父节点   
这几个函数既能操作已有的节点，也能操作动态创建的节点（createElement（））   
node.insertBefore(newnode,existingnode)当existingnode当null时在ie下会报错解决的方式就是：用if判断如果为null执行什么操作，否则执行什么操作。

简单的留言板：  
<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**meta charset="UTF-8"**>  
</**head**>  
<**body**>  
  
<**input type="text" name="text" id="text"** />  
<**input type="button" name="btn" id="btn" value="添加"** />  
  
<**ul id="ul1"**>  
  
</**ul**>  
  
  
<**script type="text/javascript"**>  
 **var *text*** = **document**.getElementById(**'text'**);  
 **var *btn*** = **document**.getElementById(**'btn'**);  
 **var *oul*** = **document**.getElementById(**'ul1'**);  
  
 ***btn***.onclick =**function**(){  
  
 **var** li = **document**.createElement(**'li'**);  
 li.**innerHTML** = ***text***.**value**;  
 **var** oa = **document**.createElement(**'a'**);  
 oa.**innerHTML** = **'删除'**;  
 oa.**href** = **'javascript:;'**;  
 oa.onclick = **function**(){  
 ***oul***.removeChild(**this**.**parentNode**);  
 }  
 li.appendChild(oa);  
  
 **if**(!***oul***.**children**[0]){  
 ***oul***.appendChild(li);  
 }**else**{  
 ***oul***.insertBefore(li,***oul***.**children**[0]);  
 }  
  
  
 **if**(***oul***.**children**.**length**>5){  
 ***oul***.removeChild(***oul***.**children**[***oul***.**children**.**length**-1])  
 }  
 }  
  
</**script**>  
</**body**>  
</**html**>

**javascript之解决dom中存在的空白节点问题**

下面有一段html文档

<**body**>  
<**h1**>Introduction to the DOM</**h1**>  
<**p class="test"**>There are a number of reasons why the DOM is awesome, here are some:</**p**>  
<**ul**>  
 <**li id="everywhere"**>It can be found everywhere.</**li**>  
 <**li class="test"**>It's easy to use.</**li**>  
 <**li class="test"**>It can help you to find what you want, really quickly.</**li**>  
</**ul**>  
</**body**>

当我们得到ul标签的下一个节点的时候，我们会习惯的引用下面的js

**var *every*** = **document**.getElementById( **"everywhere"** );  
*// and remove it from the document***console**.info(***every***.**parentNode**.**childNodes**[0].**nodeType**);

但是我们发现，控制台打印的nodeType:3.(chorme浏览器测试)

出现这样的问题是因为什么呢？

这是因为在UL标签和LI标签之间出现了空格，被浏览器误认为是文本节点，所以打印出nodeType=3,而实际我们需要得到的是元素节点LI

如何解决这个问题呢？

原理很简单，即去除节点之间的空格即可；

下面就是一段清除空格的函数

**function** *cleanWhitespace*(oEelement){  
 **for**(**var** i=0;i<oEelement.**childNodes**.**length**;i++){  
 **var** node=oEelement.**childNodes**[i];  
 **if**(node.**nodeType**==3 && !/\S/.test(node.**nodeValue**)){  
 node.**parentNode**.removeChild(node)  
 }  
 }  
}

通过使用此函数，控制台打印出来的nodeType:1--》这才是正解。

具体详细例子见如下index.html

<**html**>  
<**head**>  
 <**title**>Introduction to the DOM</**title**>  
 <**script**>  
 *// We can't manipulate the DOM until the document  
 // is fully loaded* **window**.onload = **function**(){  
 *// Locate the element with an ID of 'everywhere'* **var** every = **document**.getElementById( **"everywhere"** );  
 *// and remove it from the document* **var** a=every.**parentNode**;  
 **console**.info(a.**childNodes**[0].**nodeType**);  
  
 *cleanWhitespace*(a)  
 **console**.info(a.**childNodes**[0].**nodeType**);  
 *//清除空白函数* **function** *cleanWhitespace*(oEelement){  
 **for**(**var** i=0;i<oEelement.**childNodes**.**length**;i++){  
 **var** node=oEelement.**childNodes**[i];  
 **if**(node.**nodeType**==3 && !/\S/.test(node.**nodeValue**)){  
 node.**parentNode**.removeChild(node)  
 }  
 }  
 }  
  
 };  
  
  
 </**script**>  
</**head**>  
<**body**>  
<**h1**>Introduction to the DOM</**h1**>  
<**p class="test"**>There are a number of reasons why the DOM is awesome, here are some:</**p**>  
<**ul**>  
 <**li id="everywhere"**>It can be found everywhere.</**li**>  
 <**li class="test"**>It's easy to use.</**li**>  
 <**li class="test"**>It can help you to find what you want, really quickly.</**li**>  
</**ul**>  
</**body**>  
</**html**>