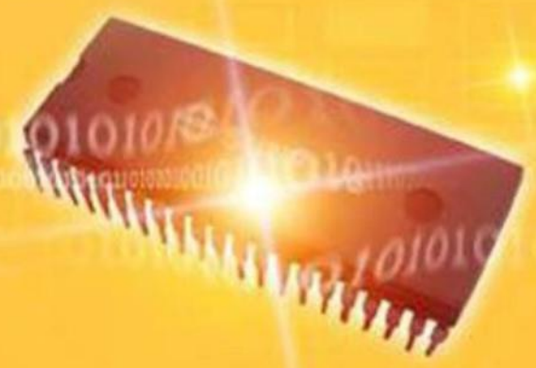
# JS-DOM



# 教学目标

- 掌握ByClassName类名获取文档元素时兼容性的处理
- 掌握获取文档元素方法的封装
- 掌握如何获取元素的样式
- 掌握DOM的概念
- 掌握什么是DOM树?
- 掌握节点、节点类型及节点关系
- 掌握节点常见操作方法

# 复习：如何获取文档元素

- 通过ID获取

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

- 通过标签名获取

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

- 通过class类名获取

  var div=document.getElementsByClassName(“ds”);

  老版本浏览器是否兼容，如IE6,7,8.

  思考：是否有其他方法来获取?

```
var div1=getId("div1");
var div2=getId(".div2");
```

## ✦ 实现原理

封装一个方法；

通过document.all()方法获取所有标签（FF不支持）；

或者通过document.getElementsByTagName("*");

利用for循环遍历所有元素，若元素.className == 预设类名；

将其推入数组；

for循环结束后return该数组

- 成品代码

```
function getClass(cls) {
    //找到所有的标签
    var elem=document.all?document.all:document.getElementsByTagName("*");
    //新建一个数组
    var arr=[];
    //遍历所有找到的标签元素elem的个数
    for(var i=0;i<elem.length;i++){
        //判断所有元素elem[i]的类名与我们传进去的参数cls若为真则执行
        if(elem[i].className==cls){
            //把相等的添加到数组中
            arr.push(elem[i]);
        }
    }
    //for循环结束后返回该数组
    return arr;
}
//通过类名找到文档元素,就可以使用getClass方法来获取
var ds=getClass("ds");
console.log(ds);
ds[0].style.backgroundColor="red";
ds[1].style.backgroundColor="blue";
ds[2].style.backgroundColor="green";
```

# 通过class类名查找文档元素的兼容性处理

- 思考

通过document.all或者document.getElementsByTagName(“*”)在文档里找所有标签相同的类名，效率不高，可不可以封装一个能缩小范围的方法呢，以前学过的缩小范围方法是通过什么来做的呢?

# 通过class类名查找文档元素的兼容性处理

- 成品代码

```
function getClass(preId,cls) {
    //找到ID
    var pId=document.getElementById(preId);
    //通过ID找到所有的元素
    var elem=pId.all?pId.all:pId.getElementsByTagName("*");
    //新建一个数组
    var arr=[];
    //遍历所有找到的标签元素elem的个数
    for(var i=0;i<elem.length;i++){
        //判断所有元素elem[i]的类名与我们传进去的参数cls若为真则执行
        if(elem[i].className==cls){
            //把相等的添加到数组中
            arr.push(elem[i]);
        }
    }
    //for循环结束后返回该数组
    return arr;
}
//通过ID名缩小范围
var ds=getClass("dd","ds");
console.log(ds);
ds[0].style.backgroundColor="red";
ds[1].style.backgroundColor="blue";
```

- 成品代码

```
function getId(id) {
    return document.getElementById(id);
}
var div1=getId("div1");
var div2=getId("div2");
var div3=getId("div3");
console.log(div1);
console.log(div2);
console.log(div3);
div1.style.backgroundColor="red";
div2.style.backgroundColor="blue";
div3.style.backgroundColor="green";
```

# 通过标签名获取文档元素封装方法

- 成品代码

```
function getEle(ele) {
    return document.getElementsByTagName(ele);
}
var div=getEle("div");
console.log(div);
div[0].style.backgroundColor="red";
div[1].style.backgroundColor="blue";
div[2].style.backgroundColor="green";
```

例13-4

# 获取元素的样式

- 下面的代码会弹出什么？能获取到元素的样式吗？

```
#div1{width:200px; height:200px; }
<div id="div1"></div>
 console.log(div1.style.width);
```

  - 下面的代码会弹出什么？能获取到元素的样式吗？

```
<div id="div2" style="width:200px; height:200px;"></div>
console.log (div2.style.width);
```

行间样式可以通过 div1.style.width 获取样式

# 获取元素的非行间样式

- currentStyle 支持IE6-11

  ie所支持的获取非行间样式的方法

  用法：对象.currentStyle.样式名

  例：div1.currentStyle.width

- getComputedStyle IE9+谷歌火狐等

  非ie所支持的获取非行间样式的方法

  用法：window.getComputedStyle(对象，参数).样式名

  Gecko 2.0 (Firefox 4 / Thunderbird 3.3 / SeaMonkey 2.1) 之前，第二个参数“伪类”是必需的（如果不是伪类，设置为null），现在的浏览器版本不是必需参数了。

  例：window.getComputedStyle(div1,null).width

  window.getComputedStyle(a,“:hover”).width

例13-6

# 获取元素的样式兼容写法

- currentStyle 支持IE6-11
- getComputedStyle IE9+谷歌火狐等

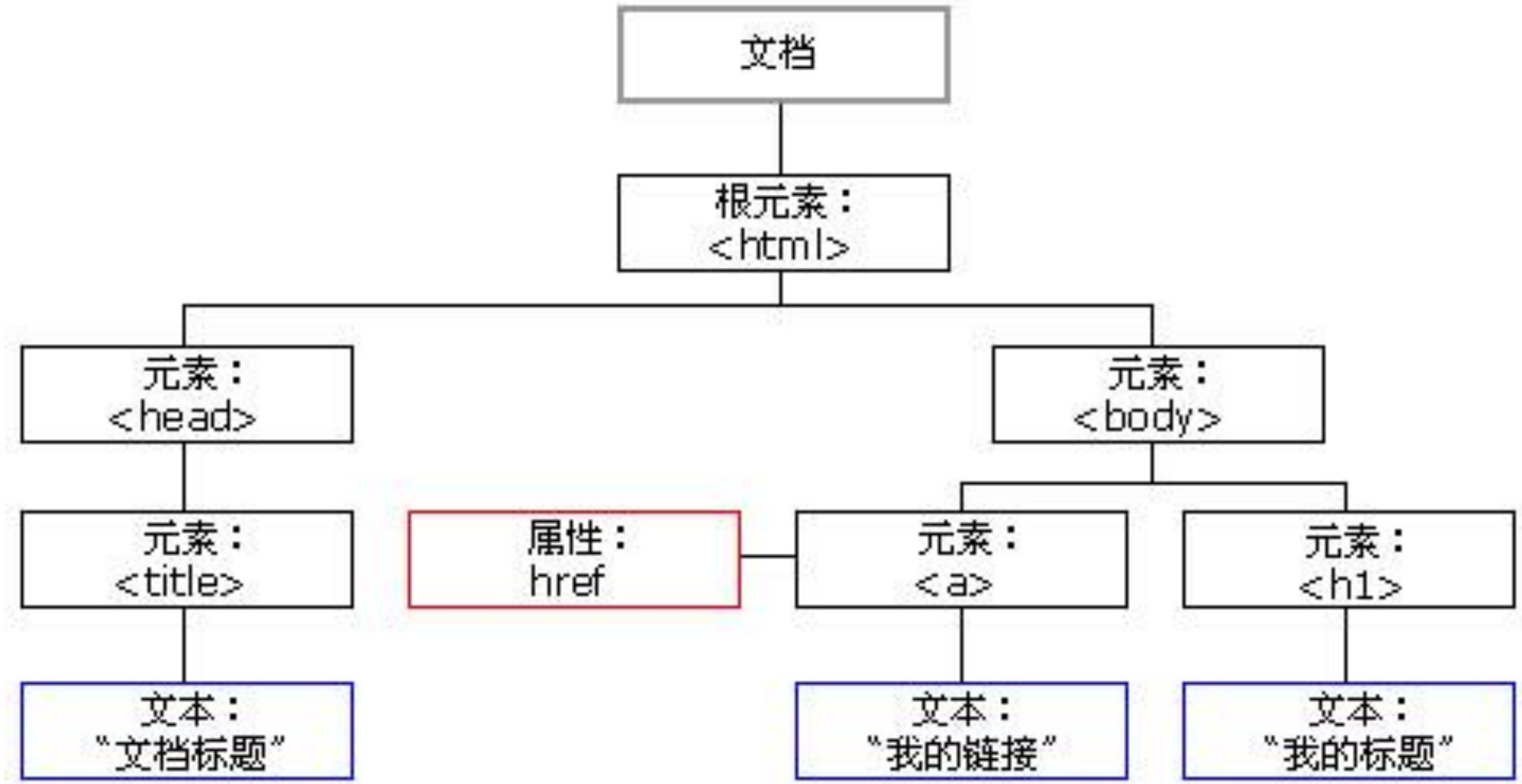
```
function getStyle(ele,attr){
    if(ele.currentStyle){
        return ele.currentStyle[attr];
    }else{
        return window.getComputedStyle(ele,null)[attr];
    }
}
console.log(getStyle(div1,"width"));
console.log(getStyle(div2,"width"));
```

例13-7

# 什么是DOM

- Document Object Model（文档对象模型）
  - 缩写为DOM；
  - DOM是针对HTML和XML文档的一个API（应用程序编程接口）；
  - DOM描绘了一个层次化的节点树，允许开发人员添加、移除和修改页面的某一部分。

# DOM树




DOM树中的所有节点均可通过JS进行访问。所有HTML元素（节点）均可被修改，也可以创建或删除节点。

HTML 文档中的所有内容都是节点。

## ✦ 节点类型

- HTML 文档中的所有内容都是节点

  整个文档是一个文档节点

  每个 HTML 元素是元素节点

  HTML 元素内的文本是文本节点(回车也是文本节点)

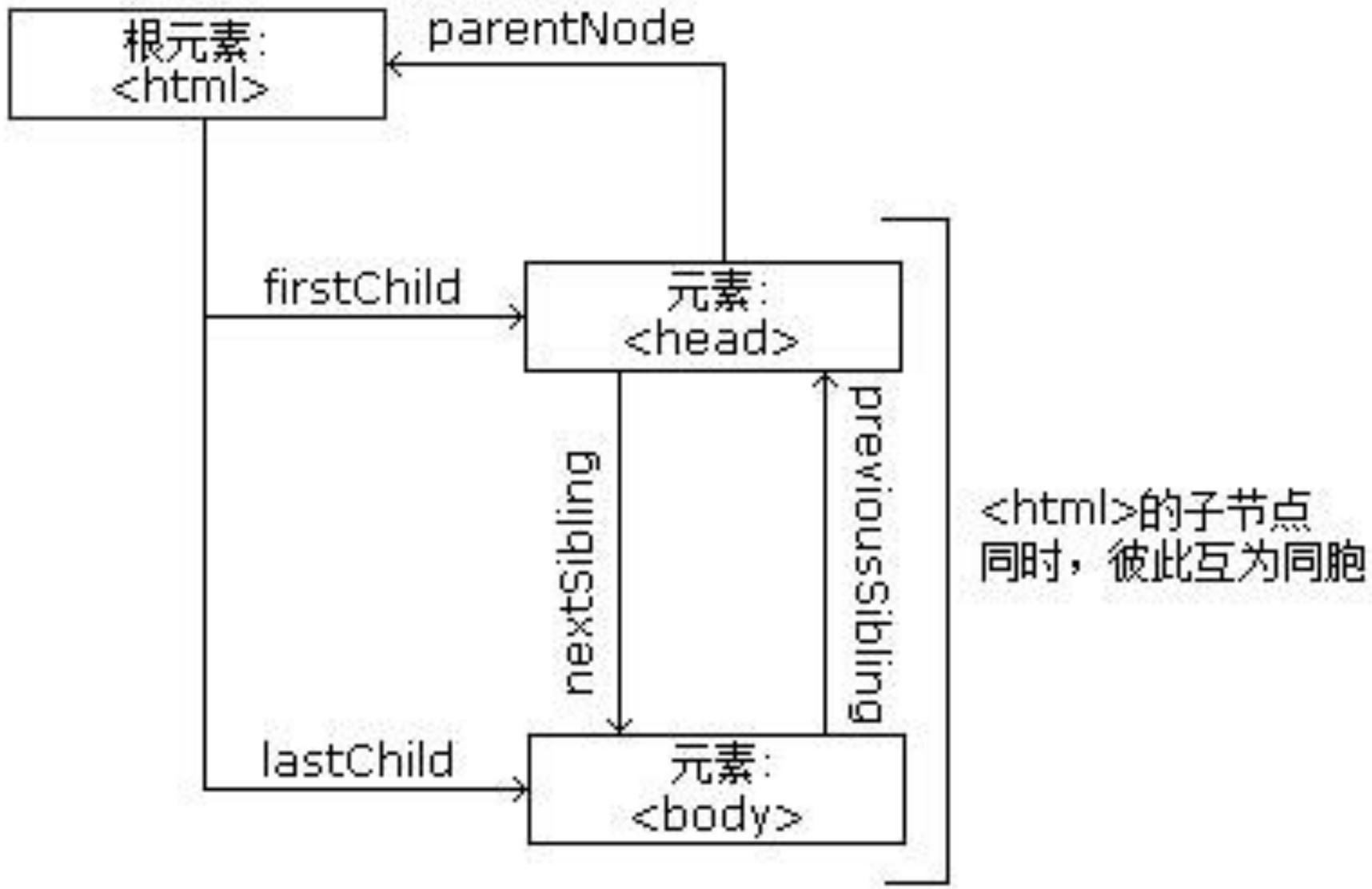
  每个 HTML 属性是属性节点

  注释是注释节点

# DOM文档对象模型

## ✦ 节点关系

下面的图片展示了节点树的一部分，以及节点之间的关系：



## ✦ 节点关系

- 子节点

  childNodes：所有子节点 （浏览器都支持）

  children：所有是标签类型的子节点 （浏览都支持）

```
<ul id="ul">
    <li class="li1">1234</li>
    <li>红色</li>
    <li>5678<span>span标签</span></li>
    <li>蓝色</li>
</ul>
<script>
    var ul=document.getElementById("ul");
    //获取ul下面所有的子节点
    console.log(ul.childNodes);
    //获取ul下面所有的标签类型的子节点
    console.log(ul.children);
</script>
```

例13-8

## ✦ 节点关系

- 父节点

  parentNode：父节点 （浏览器都支持）

```
var ul=document.getElementById("ul");
var li=ul.getElementsByTagName("li");
//获取ul的父节点
console.log(ul.parentNode);
//获取li的父节点
console.log(li[0].parentNode);
```

## ✦ 节点关系

- 下一个兄弟节点

nextSibling：

下一个兄弟节点 （浏览器都支持，在IE6、IE7、IE8中能获取所有标签类型的下一个兄弟节点，而IE9以及谷歌等现在浏览器中能获取所有类型的下一个兄弟节点）

nextElementSibling：

所有标签类型的下一个兄弟节点 （IE9以及谷歌等现在浏览器支持）

```
var ul=document.getElementById("ul");
var li=ul.getElementsByTagName("li");
//在IE9及谷歌等现代浏览器中获取的是第一个li的下一个所有类型的兄弟节点
//在IE6、IE7、IE8浏览器中获取的是第一个li的下一个标签类型的兄弟节点
console.log(li[0].nextSibling);
//只有IE9及谷歌等现代浏览器支持，获取第一个li的下一个标签类型的兄弟节点
console.log(li[0].nextElementSibling);
```

例13-10

- 节点关系
  - 下一个兄弟节点兼容写法

    nextSibling 和 nextElementSibling

```
function next(obj){
    if(obj.nextElementSibling){
        //IE9及谷歌等现代浏览器支持
        return obj.nextElementSibling;
    }else{
        //IE6、IE7、IE8支持
        return obj.nextSibling;
    }
}
console.log(next(li[0]));
```

## ✦ 节点关系

- 上一个兄弟节点

previousSibling：

上一个兄弟节点 （浏览器都支持，在IE6、IE7、IE8中能获取所有标签类型的上一个兄弟节点，而IE9以及谷歌等现在浏览器中能获取所有类型的上一个兄弟节点）

previousElementSibling：

所有标签类型的上一个兄弟节点 （IE9以及谷歌等现在浏览器支持）

```
var ul=document.getElementById("ul");
var li=ul.getElementsByTagName("li");
//在IE9及谷歌等现代浏览器中获取的是第二个li的上一个所有类型的兄弟节点
//在IE6、IE7、IE8浏览器中获取的是第二个li的上一个标签类型的兄弟节点
console.log(li[1].previousSibling);
//只有IE9及谷歌等现代浏览器支持，获取第二个li的上一个标签类型的兄弟节点
console.log(li[1].previousElementSibling);
```

例13-11

## ✦ 节点关系

- 上一个兄弟节点兼容写法

previousSibling和 previousElementSibling

```
function previous(obj){
    if(obj.previousElementSibling){
        //IE9及谷歌等现代浏览器支持
        return obj.previousElementSibling;
    }else{
        //IE6、IE7、IE8支持
        return obj.previousSibling;
    }
}
console.log(previous(li[1]));
```

## ✦ 节点关系

- 第一个子节点

firstChild：

第一个子节点 （浏览器都支持，在IE6、IE7、IE8中能获取ul的第一个所有类型的子节点，而IE9以及谷歌等现在浏览器中能获取ul的第一个标签类型的子节点）

firstElementChild：

所有标签类型的第一个子节点 （IE9以及谷歌等现在浏览器支持）

```
var ul=document.getElementById("ul");
var li=ul.getElementsByTagName("li");
//在IE9及谷歌等现代浏览器中获取的是ul的第一个所有类型的子节点
//在IE6、IE7、IE8浏览器中获取的是ul的第一个标签类型的子节点
console.log(ul.firstChild);
//只有IE9及谷歌等现代浏览器支持,获取ul的第一个标签类型的子节点
console.log(ul.firstElementChild);
```

例13-12

## ✦ 节点关系

- 第一个子节点兼容写法

  firstChild和 firstElementChild

```
function firstC(obj){
    if(obj.firstElementChild){
        //IE9及谷歌等现代浏览器支持
        return obj.firstElementChild;
    }else{
        //IE6、IE7、IE8支持
        return obj.firstChild;
    }
}
console.log(firstC(ul));
```

## ✦ 节点关系

- 最后一个子节点

lastChild：

最后一个子节点 （浏览器都支持，在IE6、IE7、IE8中能获取ul的最后一个所有类型的子节点，而IE9以及谷歌等现在浏览器中能获取ul的最后一个标签类型的子节点）

lastElementChild：

所有标签类型的最后一个子节点 （IE9以及谷歌等现在浏览器支持）

```
var ul=document.getElementById("ul");
var li=ul.getElementsByTagName("li");
//在IE9及谷歌等现代浏览器中获取的是ul的第一个所有类型的子节点
//在IE6、IE7、IE8浏览器中获取的是ul的第一个标签类型的子节点
console.log(ul.firstChild);
//只有IE9及谷歌等现代浏览器支持,获取ul的第一个标签类型的子节点
console.log(ul.firstElementChild);
```

例13-13

## ✦ 节点关系

- 最后一个子节点兼容写法

  lastChild和 lastElementChild

```
function lastC(obj){
    if(obj.lastElementChild){
        //IE9及谷歌等现代浏览器支持
        return obj.lastElementChild;
    }else{
        //IE6、IE7、IE8支持
        return obj.lastChild;
    }
}
console.log(lastC(ul));
```

## ✦ 节点关系

- 获取属性节点

  getAtrributeNode：从当前元素中获取属性节点 （浏览器都支持）

```
<ul id="ul">
    <li class="li1" id="firstLi" a="你好" b="hello">1234</li>
    <li>红色</li>
    <li>5678<span>span标签</span></li>
    <li>蓝色</li>
</ul>
<script>
    var ul=document.getElementById("ul");
    var li=ul.getElementsByTagName("li");
    //获取当前HTML元素的属性节点
    console.log(li[0].getAttributeNode("class"));
    console.log(li[0].getAttributeNode("id"));
    console.log(li[0].getAttributeNode("a"));
    console.log(li[0].getAttributeNode("b"));
</script>
```

```
class="li1"
id="firstLi"
a="你好"
b="hello"
```

例13-14

## ✦ 回顾节点类型

- HTML 文档中的所有内容都是节点

  整个文档是一个文档节点

  每个 HTML 元素是元素节点

  HTML 元素内的文本是文本节点(回车也是文本节点)

  每个 HTML 属性是属性节点

  注释是注释节点

# 节点的相关属性

- nodeName 属性返回节点的名称

```
<ul id="ul">
    <li class="li1" id="firstLi" a="你好" b="hello">1234<!--我是注释--></li>
    <li>红色</li>
    <li>5678<span>span标签</span></li>
    <li>蓝色</li>
</ul>
```

```
var li=ul.getElementsByTagName("li");
```

元素节点的 nodeName 是大写标签名称

```
var li=ul.getElementsByTagName("li");
//元素节点:nodeName是大写标签名称
console.log(li[0].nodeName);//LI
```

属性节点的 nodeName 是属性名称

```
//属性节点:nodeName是属性名称
var attr=li[0].getAttributeNode("class");
console.log(attr);//class="li1"
console.log(attr.nodeName);//class
```

例13-15

# 节点的相关属性

- nodeName 属性返回节点的名称

```
<ul id="ul">
    <li class="li1" id="firstLi" a="你好" b="hello">1234<!--我是注释--></li>
    <li>红色</li>
    <li>5678<span>span标签</span></li>
    <li>蓝色</li>
</ul>
```

```
var li=ul.getElementsByTagName("li");
```

文本节点的 nodeName 永远是 #text

```
//文本节点:nodeName永远是 #text
var text=li[0].firstChild;
console.log(text);//"1234"
console.log(text.nodeName);//#text
```

文档节点的 nodeName 永远是 #document

```
//文档节点:nodeName永远是 #document
var doc=document.nodeName;
console.log(doc);//#document
```

# 节点的相关属性

- nodeName 属性返回节点的名称

```
<ul id="ul">
    <li class="li1" id="firstLi" a="你好" b="hello">1234<!--我是注释--></li>
    <li>红色</li>
    <li>5678<span>span标签</span></li>
    <li>蓝色</li>
</ul>
```

```
var li=ul.getElementsByTagName("li");
```

注释节点的 nodeName 永远是 #comment

```
//注释节点:nodeName永远是 #comment
var comment=li[0].lastChild;
console.log(comment.nodeName);//#comment
```

# 节点的相关属性

- nodeValue 属性返回节点的值

```
<ul id="ul">
    <li class="li1" id="firstLi" a="你好" b="hello">1234<!--我是注释--></li>
    <li>红色</li>
    <li>5678<span>span标签</span></li>
    <li>蓝色</li>
</ul>
```

```
var li=ul.getElementsByTagName("li");
```

对于元素节点，nodeValue 返回值是undefined 或 null

```
//元素节点:nodeValue返回值是undefined 或 null
console.log(li[0].nodeValue);//null
```

对于元素节点，用innerHTML设置值 / 获取值

```
//元素节点如果想设置值/获取值:innerHTML
console.log(li[0].innerHTML);//1234
```

# 节点的相关属性

- nodeValue 属性返回节点的值

```
<ul id="ul">
    <li class="li1" id="firstLi" a="你好" b="hello">1234<!--我是注释--></li>
    <li>红色</li>
    <li>5678<span>span标签</span></li>
    <li>蓝色</li>
</ul>
```

```
var li=ul.getElementsByTagName("li");
```

对于文本节点，nodeValue 返回文本内容

```
//文本节点:nodeValue返回文本内容
var text=li[0].firstChild;
console.log(text.nodeValue);//1234
```

对于属性节点，nodeValue 返回属性值

```
//属性节点:nodeValue返回属性值
var attr=li[0].getAttributeNode("class");
console.log(attr.nodeValue);//li1
```

例13-16

# 节点的相关属性

- nodeValue 属性返回节点的值

```
<ul id="ul">
    <li class="li1" id="firstLi" a="你好" b="hello">1234<!--我是注释--></li>
    <li>红色</li>
    <li>5678<span>span标签</span></li>
    <li>蓝色</li>
</ul>
```

```
var li=ul.getElementsByTagName("li");
```

对于注释节点，nodeValue 返回文本内容

```
//注释节点:nodeValue返回注释内容
var comment=li[0].lastChild;
console.log(comment.nodeValue);//我是注释
```

对于文档节点，nodeValue 返回值是undefined 或 null

```
//文档节点:nodeValue返回undefined 或 null
var doc=document.nodeValue;
console.log(doc);//null
```

例13-16

# 节点的相关属性

- nodeType 属性返回节点的类型

```
<ul id="ul">
    <li class="li1" id="firstLi" a="你好" b="hello">1234<!--我是注释--></li>
    <li>红色</li>
    <li>5678<span>span标签</span></li>
    <li>蓝色</li>
</ul>
```

```
var li=ul.getElementsByTagName("li");
```

对于元素节点，返回 1

```
//元素节点:返回1
console.log(li[0].nodeType);//1
```

对于属性节点 返回 2

```
//属性节点:nodeType返回2
var attr=li[0].getAttributeNode("class");
console.log(attr.nodeType);//2
```

例13-17

# 节点的相关属性

- nodeType 属性返回节点的类型

```
<ul id="ul">
    <li class="li1" id="firstLi" a="你好" b="hello">1234<!--我是注释--></li>
    <li>红色</li>
    <li>5678<span>span标签</span></li>
    <li>蓝色</li>
</ul>
```

```
var li=ul.getElementsByTagName("li");
```

对于文本节点，返回 3（回车也是文本节点）

```
//文本节点:nodeType返回3
var text=li[0].firstChild;
console.log(text.nodeType);//3
```

对于注释节点 返回 8

```
//注释节点:nodeType返回8
var comment=li[0].lastChild;
console.log(comment.nodeType);//8
```

例13-17

## ✦ 节点的相关属性

- nodeType 属性返回节点的类型

```
<ul id="ul">
    <li class="li1" id="firstLi" a="你好" b="hello">1234<!--我是注释--></li>
    <li>红色</li>
    <li>5678<span>span标签</span></li>
    <li>蓝色</li>
</ul>
```

```
var li=ul.getElementsByTagName("li");
```

对于文档节点 返回 9

```
//文档节点:nodeType返回9
var docs=document.nodeType;
console.log(docs);//9
```

## ✦ 常用的节点操作方法

- document.createElement(“标签名”)：创建新标签元素
- document.createTextNode(“标签名”)：创建文本节点
- 父元素.appendChild(node)：向子元素末尾插入一个节点
- 父元素.insertBefore(node,targetNode)：向目标子节点targetNode之前插入节点
- 父元素.replaceChild(newNode,oldNode)：newNode替换节点oldNode
- 父元素.removeChild(node)：移除父节点的某个子节点

后四个方法操作的都是某个节点的子节点，也就是说要使用这几个方法必须先取得父节点（parentNode）,另外，并不是所有类型的节点都有子节点，如果在不支持子节点的节点上调用这些方法，将会导致错误发生。

- 常用的节点操作方法

```
var body=document.getElementsByTagName("body")[0];
var div1=document.createElement("div");
var text1=document.createTextNode("今天天气不错");
var div2=document.createElement("div");
var div3=document.createElement("div");
var text3=document.createTextNode("下雨");
var div4=document.createElement("div");
var text4=document.createTextNode("今天天气晴朗");
//在body的子节点最后插入新创建的div1元素节点
body.appendChild(div1);

//在div1的子节点后面插入新创建的text1文本节点
div1.appendChild(text1);

//把新创建的div2元素节点插入到div1元素节点的前面
body.insertBefore(div2,div1);

//把新创建的div3元素节点插入到div1元素节点的前面
div3.appendChild(text3);
body.insertBefore(div3,div1);

//用新创建的div4元素节点去替换div1元素节点
div4.appendChild(text4);
body.replaceChild(div4,div1)

//移除父节点的某个子节点
body.removeChild(div3);
```

例13-18

## ✦ 常用的节点操作方法

- cloneNode(boolean) : 复制一个节点

  true：深复制，复制节点及其整个子节点树

  false : 浅复制，只复制节点本身。

注意：cloneNode（）方法不会复制添加到DOM节点中的JavaScript，例如事件处理程序等。IE在此存在一个bug，即它会复制事件处理程序，所以我们建议在复制之前最好先移除事件处理程序。

- 常用的节点操作方法

```
var body=document.getElementsByTagName("body")[0];
var div1=document.createElement("div");
var text1=document.createTextNode("今天天气不错");
body.appendChild(div1);
div1.appendChild(text1);
div1.onclick=function () {
    alert(1);
};
//复制一个节点,如果参数为true,是深复制,复制节点及其整个子节点树
var clones1=div1.cloneNode(true);
//把复制的节点插入到body里
body.appendChild(clones1);

//复制一个节点,如果参数为false,只复制节点本身
var clones2=div1.cloneNode(false);
//把复制的节点插入到body里
body.appendChild(clones2);
```

## ✦ 常用的属性操作方法

- 创建新元素，并设置和获取className和id

```
var body=document.getElementsByTagName("body")[0];
var div1=document.createElement("div");
var text1=document.createTextNode("今天天气不错");
div1.className="myNewDiv";
div1.id="newBox";
body.appendChild(div1);
console.log(div1.className);
console.log(div1.id);
```

```
<div class="myNewDiv" id="newBox"></div>
```

思考：自定义的属性可以像上面那样设置和获取吗？试一试

```
div1.age="10";
div1.city="BJ";
```

## ✦ 常用的属性操作方法

- 如果要设置、获取或删除属性该怎么办?

元素.setAttribute(name,value)：设置节点上属性的值

元素.getAttribute(name)：获取节点上属性的值

元素.removeAttribute(name)：删除节点上的属性

```
//设置
div1.setAttribute("age","10");
div1.setAttribute("city","BJ");
//获取
console.log(div1.getAttribute("age"));
console.log(div1.getAttribute("city"));
//删除
console.log(div1.removeAttribute("age"));
console.log(div1.removeAttribute("class"));
```

例13-20

# 课程总结

- 理解包含不同层次节点的DOM
- 常见节点操作方法

  添加、删除、替换、遍历（增删改查）

- 属性常见操作方法

  获取节点上name属性：getAttribute(name)

  设置节点上name属性为value：setAttribute(name,value)

  删除节点上的name属性：

  removeAttribute(name)

新浪微博发布效果，老师展示效果样例

凌阳教育
www.sunplusedu.com

值得信赖的教育品牌

Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com