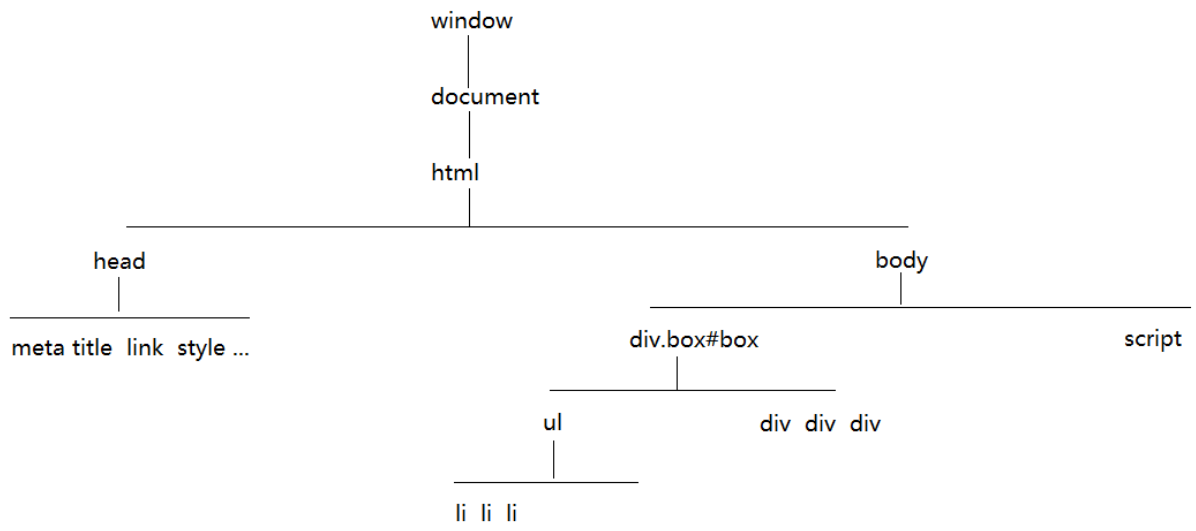
DOM树

> dom tree
> 当浏览器加载HTML页面的时候，首先就是DOM结构的计算，计算出来的DOM结构就是DOM树（把页面中的HTML标签像树桩结构一样，分析出之间的层级关系）

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>珠峰培训</title>
</head>
<body>
<div class="box" id="box">
    <ul>
        <li>新闻</li>
        <li>电影</li>
        <li>音乐</li>
    </ul>
    <div>最新新闻</div>
    <div>最新电影</div>
    <div>最新音乐</div>
</div>
```

```
18.  </body>
19.  </html>
```



DOM树描述了标签和标签之间的关系（节点间的关系），我们只要知道任何一个标签，都可以依据DOM中提供的属性和方法，获取到页面中任意一个标签或者节点

在JS中获取DOM元素的方法

`getElementById`

> 通过元素的ID获取指定的元素对象，使用的时候都是 `document.getElementById('')` 此处的document是限定了获取元素的范围，我们把它称之为“上下文(context)”

1. getElementById的上下文只能是document

> 因为严格意义上，一个页面中的ID是不能重复的，浏览器规定在整个文档中既可以获取这个唯一的ID

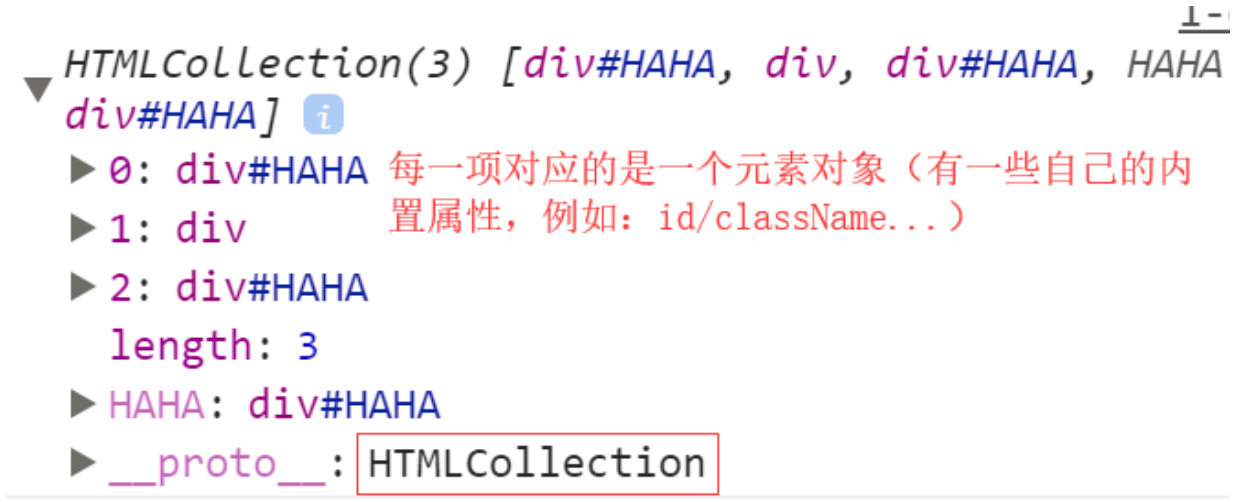
2.如果页面中的ID重复了，我们基于这个方法只能获取到第一个元素，后面相同ID元素无法获取

3.在IE6~7浏览器中，会把表单元素(input...)的name属性值当做ID来使用（建议：以后使用表单元素的时候，不要让name和id的值有冲突）

`getElementsByTagName`

> `[context].getElementsByTagName`在指定的上下文中，根据标签名获取到一组元素集合（HTMLCollection）

1. 获取的元素集合是一个类数组（不能直接的使数组中的方法）



2.它会把当前上下文中，子子孙孙（后代）层级内的标签都获取到（获取的不仅仅是儿子级的）

3.基于这个方法获取到的结果永远都是一个集合（不管里面是否有内容，也不管有几项，它是一个容器或者集合），如果想操作集合中具体的某一项，需要基于索引获取到才可以

`getElementsByClassName`

> `[context].getElementsByClassName()`在指定的上下文中，基于元素的样式类名（class='xxx'）获取到一组元素集合

1.真实项目中，我们经常是基于样式类来给元素设置样式，所以在JS中，我们也会经常基于样式类来获取元素，但是此方法在IE6~8下不兼容

兼容处理方案参考：

```
Node.prototype.queryElementsByClassName = function queryElementsByClassName() {
    if (arguments.length === 0) return [];
    var strClass = arguments[0],
        nodeList = utils.toArray(this.getElementsByTagName('*'));
    strClass = strClass.replace(/^ +| +$/g, '').split(/ +/);
    for (var i = 0; i < strClass.length; i++) {
        var reg = new RegExp('(^| +)' + strClass[i] + '( +|$)');
        for (var k = 0; k < nodeList.length; k++) {
            if (!reg.test(nodeList[k].className)) {
                nodeList.splice(k, 1);
                k--;
            }
        }
    }
```

```
15.         return nodeList;
16.     };
```

`getElementsByName`

> `document.getElementsByName()`它的上下文也只能是document，在整个文档中，基于元素的name属性值获取一组节点集合（也是一个类数组）

1.在IE浏览器中（IE9及以下版本），只对表单元素的name属性起作用（正常来说：我们项目中只会给表单元素设置name，给非表单元素设置name，其实是一个不太符合规范的操作）

`querySelector`

> `[context].querySelector()` 在指定的上下文中基于选择器（类似于CSS选择器）获取到指定的元素对象（获取的是一个元素，哪怕选择器匹配了多个，我们只获取第一个）

`querySelectorAll`

> 在querySelector的基础上，我们获取到选择器匹配到的所有元素，结果是一个节点集合（NodeList）

1. querySelector/querySelectorAll 都是不兼容IE6~8浏览器的（不考虑兼容的情况下，我们能用ById或者其它方式获取的，也尽量不要用这两个方法，这两个方法性能消耗较大）

`document.head`

> 获取HEAD元素对象

`document.body`

> 获取BODY元素对象

`document.documentElement`

> 获取HTML元素对象

```
1. //=>需求：获取浏览器一屏幕的宽度和高度（兼容所有的浏览器）
```

```
document.documentElement.clientWidth || document.body.clientWidth

document.documentElement.clientHeight || document.body.clientHeight
```

面试题：获取当前页面中所有ID为HAHA的和元素（兼容所有的浏览器）

```
//=>不能使用querySelectorAll

/*
 * 1.首先获取当前文档中所有的HTML标签
 * 2.依次遍历这些元素标签对象，谁的ID等于HAHA，我们就把谁存储起来即可
 */
function queryAllById(id) {
    //->基于通配符*获取到整个文档中所有的HTML标签
    var nodeList = document.getElementsByTagName('*');

    //->遍历集合中的每一项，把元素ID和传递ID相同的这一项存储起来
    var ary = [];
```

```
13.     for (var i = 0; i < nodeLis
    t.length; i++) {
14.         var item = nodeList[i];
15.         item.id === id ? ary.pus
    h(item) : null;
16.     }
17.     return ary;
18. }
19. console.log(queryAllById('HAH
    A'));
```

节点（node）

> 在一个HTML文档中出现的所有东西都是节点
>
> - 元素节点（HTML标签）
> - 文本节点（文字内容）
> - 注释节点（注释内容）
> - 文档节点（document）
> - ...

每一种类型的节点都会有一些属性区分自己的特性和特征

- nodeType：节点类型

- nodeName：节点名称
- nodeValue：节点值

元素节点

nodeType：1

nodeName：大写标签名

nodeValue：null

文本节点

nodeType：3

nodeName：’#text’

nodeValue：文本内容

在标准浏览器中会把空格/换行等都当做文本节点处理

注释节点

nodeType：8

nodeName：’#comment’

nodeValue：注释内容

文档节点

nodeType：9

nodeName：’#document’

nodeValue：null

描述节点之间关系的属性

`parentNode`

> 获取当前节点唯一的父亲节点

`childNodes`

> 获取当前元素的所有子节点
>
> - 子节点：只获取儿子级别的
> - 所有：包含元素节点、文本节点等

`children`

> 获取当前元素所有的元素子节点
>
> 在IE6~8中会把注释节点也当做元素节点获取到，所以兼容性不好

`previousSibling`

> 获取当前节点的上一个哥哥节点（获取的哥哥可能是元素也可能是文本等）

> previousElementSibling：获取上一个哥哥元素节点（不兼容IE6~8）

`nextSibling`

> 获取当前节点的下一个弟弟节点
>
> nextElementSibling：下一个弟弟元素节点（不兼容）

`firstChild`

> 获取当前元素的第一个子节点（可能是元素或者文本等）
>
> firstElementChild

`lastChild`

> 获取当前元素的最后一个子节点
>
> lastElementChild

需求一：获取当前元素的所有元素子节点

> 基于children不兼容IE低版本浏览器（会把注释当做元素节点）

```
/*
 * children: get all the element nodes of the current element
 * @parameter
 *    curEle: [object] current element
 * @return
 *    [Array] all the element nodes
 * by team on 2018/04/07 12:36
 */
function children(curEle) {
    //=>首先获取当前元素下所有的子节点,然后遍历这些节点,筛选出元素的(NODE-TYPE===1),把筛选出来的结果单独存储起来即可
    var nodeList = curEle.childNodes,
        result = [];
    for (var i = 0; i < nodeList.length; i++) {
```

```
            var item = nodeList[i];
            if (item.nodeType === 1)
    {
                result.push(item);
            }
        }
        return result;
}
console.log(children(course));
```

需求二：获取当前元素的上一个哥哥元素节点

> previousSibling：上一个哥哥节点
> previousElementSibling：上一个哥哥元素节点，但是不兼容

```
/*
 * prev: get the last elder brothe
r element node of the current ele
ment
 * @parameter
 *    curEle: [object] current ele
ment
 * @return
 *    [object] last elder brother
```

```
    element
 7.  * by team on 2018/04/07 12:44
 8.  */
 9. function prev(curEle) {
10.     //=>先找当前元素的哥哥节点,看是否为元素节点,不是的话,基于哥哥,找哥哥的上一个哥哥节点...一直到找到元素节点或者已经没有哥哥了(说明我就是老大)则结束查找
11.     var pre = curEle.previousSibling;
12.     while (pre && pre.nodeType !== 1) {
13.         /*
14.          * pre && pre.nodeType !== 1
15.          *   pre是验证还有没有，这样写代表有，没有pre是null
16.          *   pre.nodeType是验证是否为元素
17.          */
18.         pre = pre.previousSibling;
19.     }
20.     return pre;
21. }
```

回去后扩展：next下一个弟弟元素节点，prevAll获取

所有哥哥元素节点，nextAll获取所有弟弟元素节点，siblings获取所有兄弟元素节点，index获取当前元素的索引...

关于DOM的增删改

`createElement`

> 创建一个元素标签(元素对象)
> `document.createElement([标签名])`

`appendChild`

> 把一个元素对象插入到指定容器的末尾
> `[container].appendChild([newEle])`

`insertBefore`

> 把一个元素对象插入到指定容器中某一个元素标签之前
> `[container].insertBefore([newEle], [oldEle])`

`cloneNode`

把某一个节点进行克隆

`[curEle].cloneNode()`：浅克隆，只克隆当前的标签
`[curEle].cloneNode(true)`：深克隆，当前标签及其里面的内容都一起克隆了

`removeChild`

在指定容器中删除某一个元素

`[container].removeChild([curEle])`

`set/get/removeAttribute`

设置/获取/删除 当前元素的某一个自定义属性

```
var oBox=document.getElementById('box');

//=>把当前元素作为一个对象，在对象对应的堆内存中新增一个自定义的属性
oBox.myIndex = 10;//=>设置
console.log(oBox['myInde
```

```
x']);//=>获取
delete oBox.myIndex; //=>删除

//=>基于Attribute等DOM方法完成自定义属性的设置
oBox.setAttribute('myColor','red'); //=>设置
oBox.getAttribute('myColor'); //=>获取
oBox.removeAttribute('myColor'); //=>删除

上下两种机制属于独立的运作体制，不能互相混淆使用
- 第一种是基于对象键值对操作方式，修改当前元素对象的堆内存空间来完成
- 第二种是直接修改页面中HTML标签的结构来完成（此种办法设置的自定义属性可以在结构上呈现出来）

基于setAttribute设置的自定义属性值都是字符串
```

```
> var oBox=document.getElementById('box');
< undefined
> oBox.setAttribute('n',100)
< undefined
> oBox.n
< undefined
> oBox.getAttribute('n')
< "100"
> oBox.m=200;
< 200
> oBox.getAttribute('m')
< null
> oBox.m
< 200
```

需求：解析一个URL字符串问号传参和HASH值部分