（接notes25）

7. 操作元素

7.5. 排他思想

如果有同一组元素，我们想要某一个元素实现某种样式，需要用到循环的排他思想算法：
(1)所有元素全部清除样式(干掉其他人)
(2)给当前元素设置样式(留下我自己)

```
<button>按钮1</button>
  <button>按钮2</button>
  <button>按钮3</button>
  <button>按钮4</button>
  <button>按钮5</button>
  <script>
      // (1)获取所有按钮元素
      var btns = document.getElementsByTagName('button');
      for (var i = 0; i < btns.length; i++) {
          // btns得到的是伪数组 里面的每一个元素btn[i]
          btns[i].onclick = function () {
              // console.log(11);
              // 先把所有的按钮背景颜色去掉
              for (var i = 0; i < btns.length; i++) {
                  btns[i].style.backgroundColor = '';
              }
              // 然后才让当前的元素背景颜色为pink
              this.style.backgroundColor = 'pink';
          }
      }
  </script>
```

7.6. 自定义属性的操作

(1)获取属性值

element.属性 获取属性值
element.getAttribute('属性');

区别：
element.属性 获取内置属性值（元素本身自带的属性）
element.getAttribute('属性'); 主要获得自定义的属性（标准）我们程序员自定义的属性

(2)设置属性值

element.属性='值' 设置内置属性值
element.setAttribute('属性','值');

```
<div id="demo" index="1" class="nav"></div>
    <script>
        var div = document.querySelector('div');
        // 获取元素的属性值
        // (1)element.属性
        // (2)element.getAttribute('属性')
        console.log(div.getAttribute('id'));
        console.log(div.getAttribute('index'));

        // element.setAttribute('属性','值');     主要针对于自定义属性
        div.setAttribute('index', 2);
        div.setAttribute('class', 'footer'); //class 特殊  这里面写的就是class  不是className

        // 移除属性 removeAttribute(属性)
        div.removeAttribute('index');
    </script>
```

7.7. H5自定义属性

自定义属性的目的：是为了保存并使用数据。有些数据可以保存到页面中而不用保存到数据库中

自定义属性可以通过getAttribute('属性')获取

但是有些自定义属性很容易引起歧义，不容易判断是元素的内置属性还是自定义属性

H5给我们新增了自定义属性：

设置H5自定义属性
H5规定自定义属性data-开头作为属性名并且赋值
比如：

```
<div data-index="1"></div>
```

或者使用js设置

```
element.setAttribute('data-index',2);
```

获取H5自定义属性
兼容性获取 element.getAttribute('data-index');
H5新增element.dataset.index 或者element.dataset['index'] 有兼容性问题 IE11以上才支持

8. 节点操作

8.1. 为什么要学节点操作

获取元素通常有两种方式：

(1)利用DOM提供的方法获取元素

例如：document.getElementsByTagName() document.getElementById() document.querySelector()等

(2)利用节点层级关系获取元素

利用父子兄节点关系获取元素
逻辑性强，兼容性稍差
节点操作更简单

8.2. 节点概述

网页中的所有内容都是节点（标签、属性、文本、注释等），在DOM中，节点使用node来表示

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

一般的，节点至少拥有**nodeType（节点类型）、nodeName（节点名称）和nodeValue（节点值）**这三个基本属性

- 元素节点 nodeType为1
- 属性节点 nodeType为2
- 文本节点 nodeType为3（文本节点包含文字、空格、换行等）

在我们实际开发中，节点操作主要操作的是元素节点

8.3. 节点层级

利用DOM树可以把节点划分为不同的层级关系，常见的是父子兄层级关系

(1)父级节点

node.parentNode

```
<div>我是div</div>
    <span>我是span</span>
    <ul>
        <li>我是li</li>
        <li>我是li</li>
        <li>我是li</li>
        <li>我是li</li>
        <li>我是li</li>
        <li>我是li</li>
        <div class="box">
            <span class="erweima">x</span>
        </div>
        <script>
            // 父节点 parentNode
            var erweima=document.querySelector('.erweima');
            // 得到的是离元素最近的父级节点（亲爸爸） 如果找不到父节点就返回为空
            console.log(erweima.parentNode);
            // var box=document.querySelector('.box');
            // console.dir(box);
        </script>
    </ul>
```

(2)子级节点

parentNode.childNodes (标准)

parentsNode.childNodes 返回包含指定节点的子节点的集合，该集合为即时更新的集合

注意：

- 返回值里面包含了所有的子节点，包含元素节点，文本节点等
- 如果只想要获得里面的元素节点，则需要专门处理。所以我们一般不提倡使用childNodes

```
var ul=document.querySelector('ul');
for (var i=0;i<ul.childNodes.length;i++){
    if (ul.childNodes[i].nodeType==1){
        //ul.childNodes[i]是元素节点
        console.log(ul.childNodes[i]);
    }
}
```

parentNode.children (非标准)

parentNode.children 是一个只读属性，返回所有的子元素节点。它只返回了子元素节点，其余节点不返回（这个是我们重点掌握的）

虽然children是一个非标准，但是得到了各个浏览器的支持，因此我们可以放心使用

parentNode.firstChild

firstChild返回第一个子节点，找不到则返回null，也是包含所有的节点

parentNode.lastChild

parentNode.last返回最后一个子节点，找不到则返回null，也是包含所有的节点

parentNode.firstElementChild

parentNode.firstElementChild 返回第一个子元素节点，找不到则返回null

parentNode.lastElementChild

parentNode.lastElementChild返回最后一个子元素节点，找不到则返回null

注意：这两个方法有兼容性问题，IE9以上才支持

(3)兄弟节点

node.nextSibling

nextSibling返回当前元素的下一个兄弟节点，找不到则返回null。同样，也是包含所有的节点

node.previousSibling

previousSibling 返回当前元素的上一个兄弟节点，找不到则返回null。同样也是包含所有的节点

node.nextElementSibling

node.nextElementSibling 返回当前元素下一个兄弟元素节点，找不到则返回null

node.previousElementSibling

node.previousElementSibling 返回当前元素的上一个兄弟节点，找不到则返回null

注意：这两个方法有兼容性问题，IE9以上才支持

如何解决兼容性问题?

自己封装一个兼容性的函数

```
function getNextElementSibling(element){
    var el=element;
    while (el=el.nextsibling){
        if (el.nodeType===1){
            return el;
        }
    }
    return null;
}
```

8.4. 创建节点

```
document.createElement('tagName')
```

document.createElement() 方法创建由tagName 指定的HTML元素。因为这些元素原先不存在，是根据我们的需求动态生成的，所以我们也称为动态创建元素节点。

8.5. 添加节点

```
node.appendChild(child)
```

node.appendChild(child) 方法将一个节点添加到父节点的子节点列表末尾。类似于css里面的after伪元素

```
node.insertBefore(child,指定元素)
```

node.insertBefore(child,指定元素) 方法将一个节点添加到父节点的指定子节点前面。类似于css里面的before伪元素

```
    <ul>
        <li>123</li>
    </ul>
    <script>
        // (1)创建元素节点
        var li = document.createElement('li');
        // (2)添加节点 node.appendChild(child) node父级  child子级  后面追加元素，类似于数组中的p
        var ul=document.querySelector('ul');
        ul.appendChild(li);
        // (3)添加节点 node.insertBefore(child,指定元素);
        var lili=document.createElement('li');
        ul.insertBefore(lili,ul.children[0]);

        // (4)我们想要在页面添加一个新的元素：1.创建元素 2.添加元素
    </script>
```