目录

作业讲评

```
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <style>
        .input_text {
            border: 1px solid #d9d9d9;
            color: #aaa;
            outline: none;
        }

        .input_text_focus {
            border: 1px solid #ffd6db;
            color: #888;
            outline: none;
        }
    </style>
</head>


<body>
    <div class="new_header">
        <div class="new_wrap">
            <form id="hder_login_form_new" method="post">
                <input type="text" class="input_text" id="uname" value="邮箱/ID/手机号">
            </form>
        </div>
    </div>


    <script>
        // 获取对象
        var input = document.querySelector("#uname");
        // console.log( input );
        // 绑定获取焦点事件
        input.onfocus=function(){
            // 如果input.value内容等于"邮箱/ID/手机号"
            if(input.value == "邮箱/ID/手机号"){
```

```
            if(input.value == "邮箱/ID/手机号"){
                input.value = "";
            }
            input.style.color = "#888888";
            input.style.border = "1px solid pink";
        }
        input.onblur = function(){
            // 如果input.value内容等于""
            if(input.value == ""){
                input.value = "邮箱/ID/手机号";
                input.style.color = "#afafaf";
                input.style.border = "1px solid #d9d9d9";
            }else{
                input.style.border = "1px solid pink";
            }
        }
    </script>
</body>


</html>
```

```
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        .iconfont {
            font-family: iconfont, sans-serif;
            font-style: normal;
            -webkit-text-stroke-width: .2px;
            -moz-osx-font-smoothing: grayscale;
        }
        #J_event{
           position: relative;
           background-color:#cd1c1c;
        }
        #J_event_lk {
            display: block;
            width: 100%;
            height: 80px;
            background: url(luzhou.jpg) no-repeat 50% 0;
        }


        #luzhou{
            position: relative;
```

```
        }


        #J_event_close {
            cursor: pointer;
            position: absolute;
            right: 5px;
            top: 5px;
            background: url(close.jpg);
            width: 20px;
            height: 20px;
        }
    </style>
</head>


<body>
    <div id="J_event">
        <div id="luzhou">
            <a id="J_event_lk" href="javascript:void(0)" target="_blank"></a>
            <i id="J_event_close"></i>
        </div>
    </div>


    <script>
        // 获取关闭按钮对象
        var close_btn = document.getElementById("J_event_close");


        // 获取整个大盒子对象
        var box = document.getElementById("J_event");


        // console.log( close_btn );
        // console.log( box );


        // 给关闭按钮绑定单击事件
        close_btn.onclick=function(){
            // 隐藏大盒子
            box.style.display = "none";
        }
    </script>
</body>


</html>
```

```html
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>


<body>
    <button id="btn">开关灯</button>
    <script>
        // 获取按钮对象
        var btn = document.getElementById("btn");


        // 获取body对象
        var objBody = document.body;


        // 定义一个变量,保存现在开关灯的状态  如果body背景颜色为白色,就代表开灯;如果body背景颜色为黑色,就代表关灯  注意:status需要是全局变量
        var status = "open";


        // 给按钮绑定单击事件
        btn.onclick = function(){
            // 因为if用到objBody.style.backgroundColor,所以出问题以后,可以输出objBody.style.backgroundColor的值
            // console.log( objBody.style );
            // console.log(objBody.style.backgroundColor );


            // 第一种实现方法
            /* if(objBody.style.backgroundColor == "" || objBody.style.backgroundColor == "white"){
                objBody.style.backgroundColor = "black";
            }else if(objBody.style.backgroundColor == "black"){
                objBody.style.backgroundColor = "white";
            } */



            // 输出status的值
            // console.log( status );


            if(status == "open"){
                objBody.style.backgroundColor = "black";
                // 改变开关灯的状态
                status = "close";
            }else if(status == "close"){
                objBody.style.backgroundColor = "white";
                // 改变开关灯的状态
                status = "open";
            }
        }
    </script>
</body>


</html>
```

```html
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <style>
        * {
            margin: 0;
            padding: 0;
            list-style: none;
        }

        .tn-person-r .tn-title {
            float: left;
            font-size: 14px;
            position: relative;
            margin-left: 20px;
            width: 80px;
        }

        .tn-person-r .tn-title .tn-tab {
            border: 1px solid #FCFCFC;
            border-width: 0 1px;
            color: #4C4C4C;
            display: inline-block;
            cursor: pointer;
            position: relative;
            z-index: 9999;
            padding: 0 2px 0 0;
```

```
            text-decoration: none;
            width: 76px;
            height: 40px;
            line-height: 40px;
            text-align: center;
        }

        .tn-person-r .tn-arrow {
            display: inline-block;
            width: 8px;
            height: 5px;
            margin: 0 0 0 5px;
            overflow: hidden;
            vertical-align: middle;
            font-size: 12px;
            line-height: 13px;
            -webkit-transform: none;
            -moz-transform: none;
            -o-transform: none;
            background: url(icon.png) 0 -977px no-repeat;
        }

        .tn-person-r .tn-new {
            margin-top: -10px;
            position: absolute;
            background: url(icon.png) 0 -1017px;
            width: 14px;
            height: 11px;
        }

        .tn-person-r .tn-topmenulist-a-weibo,
        .tn-topmenulist-a-other {
            width: 77px;
        }

        .tn-person-r .tn-topmenulist-a {
            border: 1px solid #EBBE7A;
            border-top: 0;
            overflow: hidden;
            -moz-box-shadow: 3px 3px 3px rgba(0, 0, 0, .1);
            -webkit-box-shadow: 3px 3px 3px rgba(0, 0, 0, .1);
            box-shadow: 3px 3px 3px rgba(0, 0, 0, .1);
            top: 41px;
            left: 0;
        }

        .tn-person-r .tn-topmenulist .tn-text-list {
            border-bottom: 1px solid #FECC5B;
            margin: 0 0 -2px;
            width: 80px;
        }

        .tn-person-r .tn-topmenulist .tn-text-list li {
            width: 50px;
            line-height: 31px;
            border-bottom: 1px solid #FECC5B;
            color: #333;
            padding: 0 15px;
        }
    </style>
</head>


<body>
    <div class="top-nav">
        <div class="tn-bg">
            <div class="tn-header">
                <div class="tn-person-r">
                    <div class="tn-title " id="SI_Top_Weibo">
                        <a target="_blank" href="javascript:;" class="tn-tab" id="weibo">
                            <i>微博<em class="tn-new" style="display:none;"></em>
                                <span class="tn-arrow"> </span>
                            </i>
                        </a>
                        <div style="display: none;" class="tn-topmenulist tn-topmenulist-a tn-topmenulist-a-weibo" id="weibo_list">
                            <ul class="tn-text-list">
                                <li>私信</li>
                                <li>评论</li>
                                <li>@我</li>
                            </ul>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <script>
        // 获取大盒子对象
        var SI_Top_Weibo = document.getElementById("SI_Top_Weibo");
        // 获取微博对象
        var weibo = document.getElementById("weibo");
        // 获取微博列表
        var weibo_list = document.getElementById("weibo_list");
        // 鼠标移上大盒子以后
        SI_Top_Weibo.onmouseover=function(){
            // 改变微博背景颜色
            weibo.style.backgroundColor = "#cccccc";


            // 显示微博列表
            weibo_list.style.display = "block";
        }
        // 鼠标移出大盒子以后
        SI_Top_Weibo.onmouseout=function(){
            // 改变微博背景颜色
            weibo.style.backgroundColor = "white";
```

```
            // 隐藏微博列表
            weibo_list.style.display = "none";
        }


        // 鼠标移上微博以后
        weibo.onmouseover=function(){
            // 改变微博的样式
            weibo.style.color = "#f3942e";
        }
        // 鼠标移出微博以后
        weibo.onmouseout=function(){
            weibo.style.color = "#4c4c4c";
        }


        // 获取.tn-text-list里面所有li标签
        var lis = document.querySelectorAll(".tn-text-list li");
        // console.log( lis.length );
        for(var i=0;i<lis.length;i++){
            // 输出i看看对不对,看看是不是咱们想要的0 1 2
            // console.log( i );// 这里i没问题,但是马上一下就输出0 1 2



            // 特别注意:绑定事件跟触发事件会同时发生的,是边绑定边触发吗?  还是先绑定了事件才机会触发事件   是先绑定后触发事件    只能绑定完毕以后,才有机会触发事件


            // 给每个li绑定鼠标移上事件
            lis[i].onmouseover=function(){
                // console.log("来也匆匆~");


                // console.log( i );// 匿名函数里面代码,只有事件被触发以后才会执行


                // 使用this关键字  大家暂时这样记,  "this谁触发事件,this代表谁"
                // console.log( this );


                // lis[i].style.backgroundColor = "#fff5da";
                // lis[i].style.color = "#ff8400";


                this.style.backgroundColor = "#fff5da";
                this.style.color = "#ff8400";
            }

            // 给每个li绑定鼠标移出事件
            lis[i].onmouseout=function(){
                // console.log("去也冲冲~");


                // console.log( i );// 匿名函数里面代码,只有事件被触发以后才会执行


                // lis[i].style.backgroundColor = "#ffffff";
                // lis[i].style.color = "#4c4c4c";


                this.style.backgroundColor = "#ffffff";
                this.style.color = "#4c4c4c";
            }
        }


        /* lis[0].onmouseover=function(){
            lis[0].style.backgroundColor = "#fff5da";
            lis[0].style.color = "#ff8400";
        }
        lis[0].onmouseout=function(){
            lis[0].style.backgroundColor = "#ffffff";
            lis[0].style.color = "#4c4c4c";
        }
        lis[1].onmouseover=function(){
            lis[1].style.backgroundColor = "#fff5da";
            lis[1].style.color = "#ff8400";
        }
        lis[1].onmouseout=function(){
            lis[1].style.backgroundColor = "#ffffff";
            lis[1].style.color = "#4c4c4c";
        }
        lis[2].onmouseover=function(){
            lis[2].style.backgroundColor = "#fff5da";
            lis[2].style.color = "#ff8400";
        }
        lis[2].onmouseout=function(){
            lis[2].style.backgroundColor = "#ffffff";
            lis[2].style.color = "#4c4c4c";
        } */
    </script>
</body>


</html>
```

学习目标

- 能够使用js控制css样式
- 能够说出排他操作的一般实现步骤
- 能够使用html5中的dataset方式操作自定义属性
- 能够根据提示完成百度换肤的案例
- 能够根据提示完成全选案例
- 能够根据提示完成tab栏切换案例
- 能够区分元素节点、文本节点、属性节点
- 能够获取指定元素的父元素
- 能够获取指定元素的所有子元素
- 能够说出childNodes和children的区别
- 能够使用createElement创建页面元素

样式属性操作

我们可以通过 JS 修改元素的大小、颜色、位置等样式。

常用方式

样式属性	功能
元素.style	行内样式操作
元素.className	类名样式操作

方式1:通过操作style属性

```
元素对象的style属性也是一个对象!
元素对象.style.样式属性 = 值;
```

方式2:通过操作className属性

```
元素对象.className = 值;
因为class是关键字，所有使用className。
```

注意:

1. 使用 element.style 获得修改元素样式 如果样式比较少 或者 功能简单的情况下使用
2. 如果样式修改比较多,可以采取操作类名方法更改元素样式
3. class因为是个保留字,因此使用className来操作元素类名属性
4. className 会直接更新元素的类名,会覆盖原先的类名

5. 如果想要保留原先的类名，我们可以这么做 多类名选择器

举例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style>
        .first{
            border:5px solid blue;
        }


        .change{
            background: purple;
            color:#fff;
            font-size:25px;
            margin-top: 100px;
        }
    </style>
</head>
<body>
    <div class="first">文本</div>


    <script>
        // 我们可以通过 对象.className 来 设置对象的class类名属性或者  获取对象的class类名属性
        // 因为class是js中一个关键字,所以使用className



        // 获取.first对象
        var first = document.querySelector(".first");
        // console.log( first );
        // 通过  对象.style 设置元素样式
        // first.style.backgroundColor = "purple";
        // first.style.color = "#fff";
        // first.style.fontSize = "25px";
        // first.style.marginTop = "100px";



        // 注意:
        // 使用 element.style 获得修改元素样式 如果样式比较少 或者 功能简单的情况下使用
        // 如果样式修改比较多,可以采取操作类名方法更改元素样式
        // class因为是个保留字,因此使用className来操作元素类名属性
        // className 会直接更新元素的类名,会覆盖原先的类名
        // 如果想要保留原先的类名，我们可以这么做 多类名选择器


        // 通过 对象.className 获取对象的class属性值
        // console.log( first.class );// undefined
        // console.log( first.className );// first


        // 通过 对象.className = "类名"  可以给对象设置class属性值
        // 特别注意:  对象.className = "类名"会覆盖元素原有的class属性值
        // first.className = "change";
```

```
        // 如果想保留原类名
        // 方法一:使用多类名选择器
        // first.className = "first change";


        // 方法二: 使用+=
        // first.className =  first.className+" change";
        first.className += " change";
    </script>
</body>
</html>
```

排他操作

什么是排他操作?

也可以理解为排他思想,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）**
3. **注意顺序不能颠倒，首先干掉其他人，再设置自己**

按钮1 按钮2 按钮3 按钮4 按钮5

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        button{
            border:none;
            width: 80px;
            height: 50px;
            line-height: 50px;
            text-align: center;
            outline:none;
        }
    </style>
</head>
<body>
    <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>
        // 什么是排他操作?
        // 也可以理解为排他思想,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:
        // 所有元素全部清除样式（干掉其他人）
        // 给当前元素设置样式 （留下我自己）
        // 注意顺序不能颠倒，"首先干掉其他人，再设置自己"


        // 我们要实现的功能是:点击哪个按钮,哪个按钮背景颜色就变色粉色
        var btns = document.querySelectorAll("button");
        for(var i=0;i<btns.length;i++){
            btns[i].onclick=function(){
                // 谁触发事件,this就代表谁
                // console.log( this );


                // 干掉其他按钮的背景颜色
                for(var j=0;j<btns.length;j++){
                    // 遍历所有的按钮对象
                    // console.log( btns[j] );
                    btns[j].style.backgroundColor = "";
                }


                // 再单独设置自己的背景颜色
                this.style.backgroundColor = "pink";
            }
        }
    </script>
</body>
</html>
```

案例:百度换肤

分析:

1. 给4个小图片利用循环绑定点击事件
2. 当我们点击某个图片的时候,我们就让页面的背景改为当前的图片
3. 核心思路:点击以后把当前图片的src路径取出来,设置为body标签的背景即可

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style>
        * {
            margin: 0;
            padding: 0;
        }

        body {
            background: url(images/1.jpg) no-repeat center top;
        }

        li {
            list-style: none;
        }

        .baidu {
            overflow: hidden;
            margin: 100px auto;
            background-color: #fff;
            width: 410px;
            padding-top: 3px;
        }

        .baidu li {
            float: left;
            margin: 0 1px;
            cursor: pointer;
        }

        .baidu img {
```

```
            width: 100px;
        }
    </style>
</head>
<body>
    <ul class="baidu">
        <li><img src="images/1.jpg"></li>
        <li><img src="images/2.jpg"></li>
        <li><img src="images/3.jpg"></li>
        <li><img src="images/4.jpg"></li>
    </ul>


    <script>
        var lis = document.querySelectorAll(".baidu li");
        // console.log( lis );
        for(var i=0;i<lis.length;i++){
            lis[i].onclick=function(){
                // 谁触发this就代表谁
                // console.log( this );


                // 目标是拿到当前被点击的li里面img标签的src属性,赋值body标签的背景图片
                var src = this.getElementsByTagName("img")[0].src ;
                // console.log( src );
                document.body.style.backgroundImage = "url("+src+")";
            }
        }
    </script>
</body>
</html>
```

练习:表格隔行变色

代码	名称	最新公布净值	累计净值	前单位净值	净值增长率
003526	农银金穗3个月定期开放债券	1.075	1.079	1.074	+0.047%
003526	农银金穗3个月定期开放债券	1.075	1.079	1.074	+0.047%
003526	农银金穗3个月定期开放债券	1.075	1.079	1.074	+0.047%
003526	农银金穗3个月定期开放债券	1.075	1.079	1.074	+0.047%
003526	农银金穗3个月定期开放债券	1.075	1.079	1.074	+0.047%
003526	农银金穗3个月定期开放债券	1.075	1.079	1.074	+0.047%

分析:

1. 鼠标经过事件 onmouseover 鼠标离开 onmouseout
2. 鼠标经过tr行,当前行改变背景颜色,鼠标离开去掉当前的背景颜色
3. 第一行(thead里面的行)不需要变换颜色,因此我们获取的是tbody里面的行

练习:全选

	商品	价钱
	iPhone8	8000
	iPad Pro	5000
	iPad Air	2000
	Apple Watch	2000

要求:

```
1. 点击最上面的复选框,下面所有的复选框都选中(全选功能)
2. 再次点击最上面的复选框,下面所有的复选框都选中(取消全选功能)
3. 如果下面复选框全部选中,最上面的复选框就自动选中
4. 如果下面复选框有一个没有选中,最上面的复选框就不选中
5. 所有复选框一开始默认都不选中
```

分析:

```
1. 全选和取消全选        让下面所有复选框的checked属性跟随最上面复选框即可
2. 下面复选框需要全部选中,上面全选才能选中
给下面所有复选框绑定点击事件,每个点击,都要循环查看下面所有复选框是否都被选中,
如果有一个没被选,上面全选就不选中了
3. 可以设置一个变量,来控制全选是否选中
```

自定义属性操作

获取属性值

语法:

```
element.属性名
或者
element.getAttribute("属性名")
```

区别:

```
element.属性名    一般用于获取元素本身自带的属性    比如 img自带src属性
```

```
element.getAttribute("属性名")
主要用于获取自定义的属性,自定义属性就是我们程序员自己定义的属性
```

设置属性值

语法:

```
element.属性名 = "属性值"
element.setAttribute("属性名","属性值");
```

区别:

```
element.属性名 = "属性值" 主要设置内置属性值
element.setAttribute("属性名","属性值"); 主要设置自定义的属性
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!-- 比如以下标签,id跟class属性就是内置属性,而index是自定义属性 -->
    <div id="demo" index="1" class="nav">div的内容</div>


    <script>
        // 在HTML标签中,我们是可以自定义的添加属性的,而标签原本自带的属性,我们可以叫,而我们程序员给标签添加的属性,我们叫自定义属性


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


        // 获取内置属性 对象.属性名
        console.log( div.id );// demo
        console.log( div.class );// undefined
        // 如果使用  对象.属性名 这种方式获取类名 需要使用className 不能使用class
        console.log( div.className );// nav


        // 我们想获取自定义属性 能不能也使用 对象.属性名 这种方式呢 ?  不行
        console.log( div.index );// undefined
        console.log("");


        // 如果想要获取自定义属性 需要使用  对象.getAttribute("属性名")
        console.log( div.getAttribute("index") );// 1
        console.log("");


        // 设置内置属性  对象.属性名 = 属性值
        // div.style.color = "red";
        // div.style.border= "1px solid blue";
        div.style = "color:red;border:1px solid blue";


        // 设置自定义属性 用 对象.属性名 = 属性值 设置不了自定义属性
        div.abc = "大佬带带我,原来我是个萌新";


        // 设置自定义属性 需要使用  对象.setAttribute("属性名","属性值");
        div.setAttribute("hello","你好啊,自定义属性");
        console.log("");


        // 那么我们可以使用 getAttribute获取内置属性吗?
        console.log(  div.getAttribute("id") );

        // 那么我们可以使用 setAttribute设置内置属性吗?
        div.setAttribute("title","我是一个div,我是一个大盒子");


        // 总结:  要记
        // getAttribute方式可以获取内置属性也可以获取自定义属性,而 对象.属性名 的方式只能获取内置属性
        // setAttribute方式可以设置内置属性也可以设置自定义属性,而 对象.属性名="属性值" 的方式只能设置内置属性
    </script>
</body>
</html>
```

特别注意: 使用getAttribute和setAttribute操作类名需要使用class

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="demo" class="nav">div的内容</div>


    <script>
        var div = document.getElementById("demo");
        // 特别注意:  使用getAttribute和setAttribute操作类名需要使用"class",而不是className
        console.log( div.getAttribute("class") );// nav
        console.log( div.getAttribute("className") );// null


        div.setAttribute("class","nav abc");// 成功设置
        div.setAttribute("className","nav edg");// 无效
    </script>
</body>
</html>
```

移除属性

```
element.removeAttribute("属性名");

可以移除自定属性也可以移除自有属性
```

举例:

```html
<div id="demo" index="1" class="nav">div的内容</div>
<script>
    // 移除属性使用的语法是  对象.removeAttribute("属性名")
    // 对象.removeAttribute("属性名")可以移除内置属性也可以移除自定义属性
    var div = document.getElementById("demo");

    // 移除内置class属性
    div.removeAttribute("class");

    // 移除自定义属性index
    div.removeAttribute("index");
</script>
```

H5自定义属性　扩展知识

自定义属性目的: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。

自定义属性获取是通过getAttribute("属性名") 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。**

所以,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",1)
```

获取H5自定义属性

```
比如 <div data-index="1"></div>

1. 兼容性获取    element.getAttribute("data-index");
2. H5新增获取方法 element.dataset.index 或者 element.dataset["index"]
但是要注意ie 11以后才支持;如果自定义属性里面有多个-链接的单词，我们获取的时候采取 驼峰命名法
```

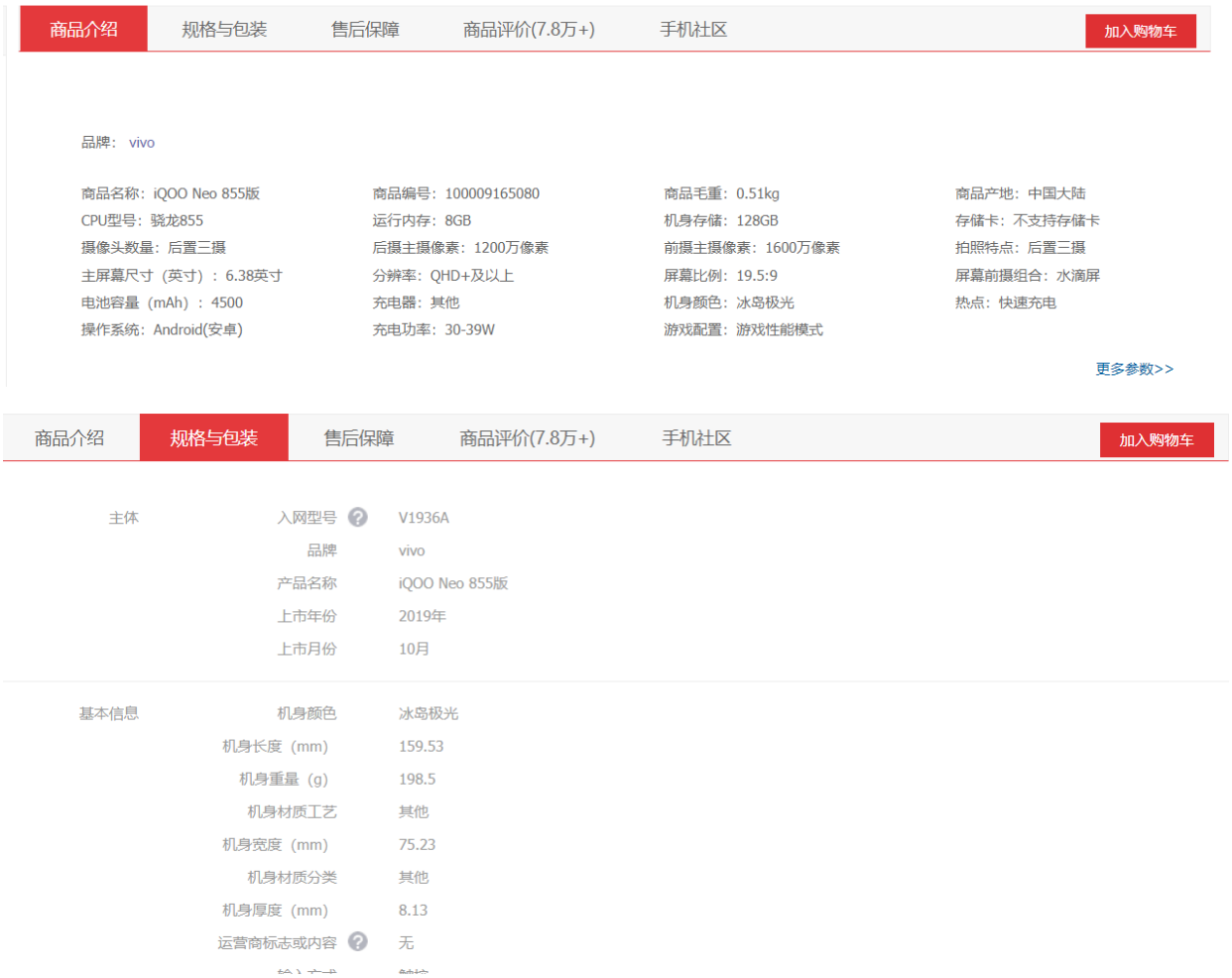
举例:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div data-index="1">div的内容</div>
    <p data-spm-max-idx="淘宝复制的自定义属性">p段落内容</p>


    <script>
        // H5自定义属性    扩展知识
        // 自定义属性目的: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。
        // 自定义属性获取是通过getAttribute("属性名") 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。


        // 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",1)


        var p = document.querySelector("p");
        // 设置自定义属性,一定要使用setAttribute("属性名","属性值")这种方式
        p.setAttribute("data-abc","我是abc");


        // 获取H5自定义属性
        // 比如 <div data-index="1"></div>
        // 1. 兼容性获取    element.getAttribute("data-index"); 推荐大家使用
        console.log( p.getAttribute("data-abc") );
        console.log("");


        // 2. H5新增获取方法 element.dataset.index 或者 element.dataset["index"]
        console.log( p.dataset.abc );
        console.log( p.dataset["abc"] );
        console.log("");
        // 但是要注意ie 11以后才支持;如果自定义属性里面有多个-链接的单词，我们获取的时候采取 驼峰命名法
        console.log( p.getAttribute("data-spm-max-idx") );
        console.log( p.dataset.spmMaxIdx );
        console.log( p.dataset["spmMaxIdx"]);
    </script>
</body>
</html>
```

练习:tab栏

要求:

当鼠标点击上面相应的选项卡(tab),下面内容跟随变化



分析:

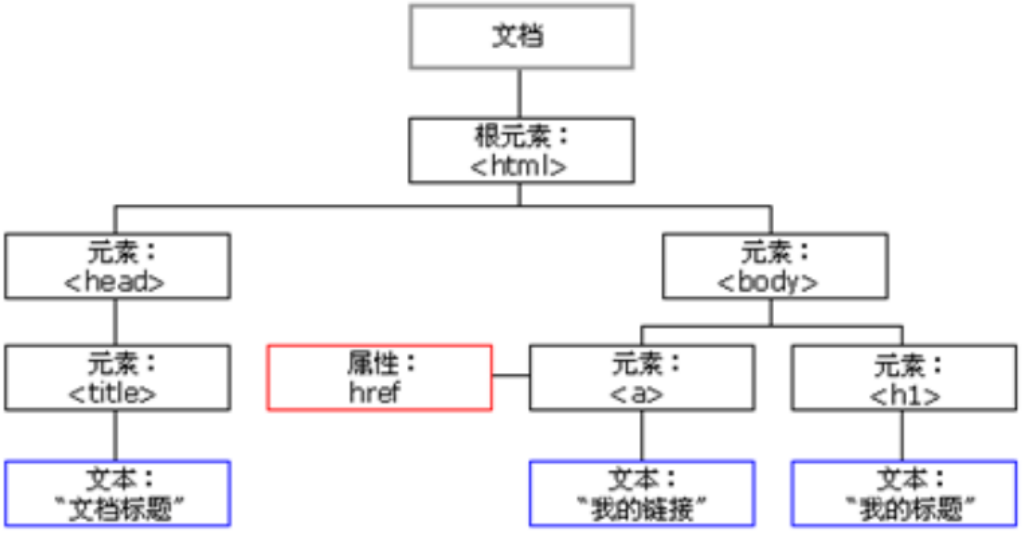
```
1. Tab栏切换分为2个模块
2. 上模块是选项卡,点击某一个,当前被点击那个底色会变色红色,其余不变(排他思想),可以修改类名
3. 下模块为内容,会跟随上面的选项卡变化
4. 规律:下模块显示内容和上面的选项卡一一对应
5. 给上面的tab_list里面的所有小li添加自定义属性,属性值从0开始编号
6. 当我们点击tab_list里面的某个小li,让tab_con里面对应序号的内容显示,其余隐藏(排他思想)
```

节点操作

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，也可以创建或删除。



一般地，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。

常见三种节点 元素节点(找到的HTML标签) 属性节点(找到HTML标签上的属性) 文本节点(找到HTML标签中的内容,或者换行,空格都会算文本节点)

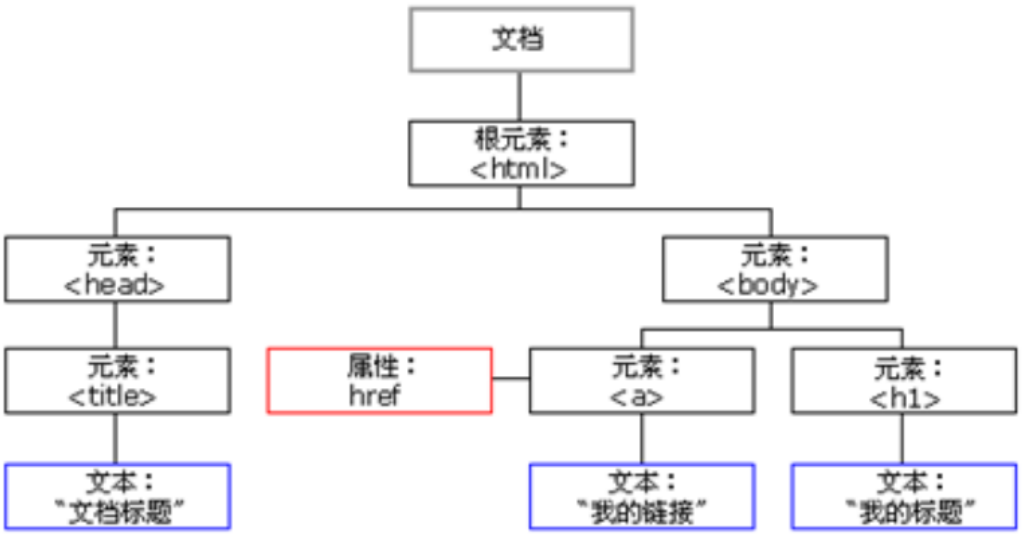
- 元素节点 nodeType 为 1
- 属性节点 nodeType 为 2
- 文本节点 nodeType 为 3 （文本节点包含文字、空格、换行等）

我们在实际开发中，节点操作主要操作的是**元素节点**

详细可以参考 https://www.cnblogs.com/xiaoleidiv/p/3347483.html

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**父子兄弟层级关系**。



父级节点 要记

node.parentNode

- parentNode属性可返回某节点的父节点,注意是最近的一个父节点
- 如果指定的节点没有父节点,则返回null

举例:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div class="demo">
        <div class="box">
            <span class="erweima">二维码</span>
        </div>
     </div>


     <script>
        // 获取.erweima对象
        var erweima = document.querySelector(".erweima");


        // 获取父级节点(获取父元素)  大家可以简单理解为节点就是元素
        // node.parentNode
        // parentNode属性可返回某节点的父节点,注意是最近的一个父节点
        // 如果指定的节点没有父节点,则返回null
        console.log( erweima.parentNode );// .box
        console.log( erweima.parentNode.parentNode );// .demo
        console.log( erweima.parentNode.parentNode.parentNode );// body
        console.log("");


        // 输出HTML元素
        console.log( document.documentElement );// HTML
        console.log( document.documentElement.parentNode );// document
        // 如果指定的节点没有父节点,则返回null
        console.log( document.documentElement.parentNode.parentNode );// null
     </script>
</body>
</html>
```

子节点

获取所有子节点

父节点.childNodes　　返回包含指定节点的子节点的集合

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

获取子元素节点 要记

父节点.children 是一个只读属性,返回所有的子元素节点.它只返回子元素节点,其他节点不返回(这个需要我们重点掌握) 特别需要注意的是, 父节点.children获取的是动态节点

```
<ul>
    <li>我是li1</li>
    <li>我是li2</li>
    <li>我是li3</li>
    <li>我是li4</li>
</ul>
```

获取第1个子节点

```
父节点.firstChild 返回第一个子节点,找不到则返回null
包含所有的节点,不管是文本节点还是元素节点
```

获取最后1个子节点

```
父节点.lastChild 返回最后一个子节点,找不到则返回null
包含所有的节点,不管是文本节点还是元素节点
```

获取第1个子元素节点

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

获取最后1个子元素节点

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

注意: firstElementChild 跟 lastElementChild 这两个有兼容性问题,IE9以上才支持

实际开发中，firstChild 和 lastChild 包含其他节点，操作不方便，而 firstElementChild 和 lastElementChild 又有兼容性问题，那么我们如何获取第一个子元素节点或最后一个子元素节点呢?

解决方法:

1. 如果想要第一个子元素节点,可以使用　　父节点.children[0]
2. 如果想要最后一个子元素节点,可以使用　　父节点.children[父节点.children.length-1]

举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <ul>
        <li>我是li1</li>
        <li>我是li2</li>
        <li>我是li3</li>
        <li>我是li4</li>
    </ul>



    <script>
        // 先获取ul对象
        // var ul = document.querySelector("ul");
        var ul = document.getElementsByTagName("ul")[0];


        // console.log( ul );


        /* 获取所有子节点
        父节点.childNodes        返回包含指定节点的子节点的集合
        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 */
        console.log( ul.childNodes );
        console.log("");


        /* 获取所有子元素节点  要求记住
        父节点.children      是一个只读属性,返回所有的子元素节点.它只返回子元素节点,其他节点不返回(这个需要我们重点掌握)     特别需要注意的是, 父节点.children获取的是动态节点 */
        console.log( ul.children );
        console.log( ul.children[0].innerHTML );
        console.log( ul.children[3].innerHTML );
        console.log( ul.children[ ul.children.length-1 ].innerHTML );
        console.log("");


        /* 获取第1个子节点
        父节点.firstChild 返回第一个子节点,找不到则返回null
        包含所有的节点,不管是文本节点还是元素节点 */
        console.log( ul.firstChild );


        /* 获取最后1个子节点
        父节点.lastChild 返回最后一个子节点,找不到则返回null
        包含所有的节点,不管是文本节点还是元素节点 */
        console.log( ul.lastChild );
        console.log("");


        /* 获取第1个子元素节点
        父节点.firstElementChild 返回第一个子元素节点,找不到则返回null */
        console.log( ul.firstElementChild );

        /* 获取最后1个子元素节点
        父节点.lastElementChild 返回最后一个子元素节点,找不到则返回null */
        console.log( ul.lastElementChild );
        console.log("");


        // 注意:firstElementChild 跟 lastElementChild 这两个有兼容性问题,IE9以上才支持


        // 实际开发中，firstChild 和 lastChild 包含其他节点，操作不方便，而 firstElementChild 和 lastElementChild 又有兼容性问题，那么我们如何获取第一个子元素节点或最后一个子元素节点呢？


        // 解决方法:
        // 如果想要第一个子元素节点,可以使用        父节点.children[0]
        // 如果想要最后一个子元素节点,可以使用    父节点.children[父节点.children.length-1]
        console.log( ul.children );
        console.log( ul.children[0] );
        console.log( ul.children[ ul.children.length-1 ] );
    </script>
</body>
</html>
```

兄弟节点　了解,比较少用到

下一个兄弟节点

当前元素.nextSibling　返回当前元素的下一个兄弟节点,找不到返回null 包含元素节点或者 文本节点等等

上一个兄弟节点

当前元素.previousSibling　返回当前元素的上一个兄弟节点,找不到返回null　包含元素节点或者 文本节点等等

下一个兄弟元素节点（有兼容性问题）　IE9以上才支持

当前元素.nextElementSibling　返回当前元素的下一个兄弟元素节点,找不到返回null

上一个兄弟元素节点（有兼容性问题）　IE9以上才支持

当前元素.previousElementSibling　返回当前元素上一个兄弟元素节点,找不到返回null

举例:

```
<p>我是p</p>
<div>我是div</div>
<span>我是span</span>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <p>我是p</p>
    <div>我是div</div>
    <span>我是span</span>


    <script>
        // 兄弟节点    了解,比较少用到
        var div = document.getElementsByTagName("div")[0];
        // console.log( div );


        /* 下一个兄弟节点
        当前元素.nextSibling    返回当前元素的下一个兄弟节点,找不到返回null 包含元素节点或者 文本节点等等 */
        console.log( div.nextSibling );


        /* 上一个兄弟节点
        当前元素.previousSibling    返回当前元素的上一个兄弟节点,找不到返回null    包含元素节点或者 文本节点等等 */
        console.log( div.previousSibling );
        console.log("");


        /* 下一个兄弟元素节点（有兼容性问题）    IE9以上才支持
        当前元素.nextElementSibling    返回当前元素的下一个兄弟元素节点,找不到返回null */
        console.log( div.nextElementSibling );

        /* 上一个兄弟元素节点（有兼容性问题）    IE9以上才支持
        当前元素.previousElementSibling    返回当前元素上一个兄弟元素节点,找不到返回null */
        console.log( div.previousElementSibling );
    </script>
</body>
</html>
```

注意: nextElementSibling和previousElementsSibling有兼容性问题,IE9移上才支持.问:那么我们改如何解决兼容性问题呢?

答:自己封装一个兼容性的函数

代码:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Document</title>
</head>
<body>
 <p>我是p</p>
 <div>我是div</div>
  <span>我是span</span>
  <script type="text/javascript">
  // 封装函数获取下一个元素节点
  function getNextElementSibling(element) {
  var el = element;
  while (el = el.nextSibling) {
  if (el.nodeType === 1) { // nodeType 1为元素节点 2为属性节点 3为文本节点
  return el;
  }
  }
  return null;
  }

  // 封装函数获取上一个元素节点
  function getPreviousElementSibling(element) {
  var el = element;
  while (el = el.previousSibling) {
  if (el.nodeType === 1) { // nodeType 1为元素节点 2为属性节点 3为文本节点
  return el;
  }
  }
  return null;
  }
  var objDiv = document.querySelector("div");
  console.log( getNextElementSibling(objDiv) );
```

```
35  console.log( getPreviousElementSibling(objDiv) );
36  </script>
37 </body>
38 </html>
```

创建节点 要记

```
1 document.createElement("标签名")
```

该方法创建指定标签名的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,所以我们也称之为**动态创建元素节点**

添加节点 要记

```
1 父节点.appendChild(节点)
```

父节点.appendChild(节点) 方法将一个节点添加到指定父节点的子节点列表**末尾** 类似与数组中的push方法

```
1 父节点.insertBefore(节点,指定元素)
```

父节点.insertBefore(节点,指定元素) 方法将一个节点添加到父节点的**指定子元素前面**

举例:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <ul>
        <li>苹果</li>
        <li>香蕉</li>
    </ul>


     <script>
        // 我们可以通过js创建节点,其实简单理解创建标签
        // 语法   document.createElement("标签名")

        // 获取ul标签
        var ul = document.querySelector("ul");
        // console.log( ul );


        // 创建一个新的li标签
        var li = document.createElement("li");
        // 设置新的li标签中的内容
        li.innerHTML = "葡萄";

        // 添加节点
        // 父节点.appendChild(节点)   方法将一个节点添加到指定父节点的子节点列表末尾   类似与数组中的push方法
        ul.appendChild(li);


        // 父节点.insertBefore(节点,指定元素)    方法将一个节点添加到父节点的指定子元素前面
        var li2 = document.createElement("li");
        li2.innerHTML = "桃子";
        ul.insertBefore(li2,  ul.children[0] );


        var li3 = document.createElement("li");
        li3.innerHTML = "大冬瓜";
        console.log( ul.children );
        ul.insertBefore(li3,  ul.children[2] );


    </script>
</body>
</html>
```

案例:简单版发布留言

发布

- 沙发~
- 楼主牛逼

分析

1. 点击按钮之后,动态创建一个li
2. 创建li的同时,把文本域里面的值通过innerHTML赋值给li,在把li添加到ul里面
3. 如果想要新的留言后面显示就用appendChild

```
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <style>
        * {
            margin: 0;
            padding: 0;
        }

        body {
            padding: 100px;
        }

        textarea {
            width: 200px;
            height: 100px;
            border: 1px solid pink;
            outline: none;
            resize: none;
        }

        ul {
            margin-top: 50px;
        }

        li {
            width: 300px;
            padding: 5px;
            background-color: rgb(245, 209, 243);
            color: red;
            font-size: 14px;
            margin: 15px 0;
        }
    </style>
</head>


<body>
    <textarea></textarea>
    <button>发布</button>
    <ul>
```

```
    </ul>

    <script>
        // 点击发布按钮以后,需要把文本域中的内容提取出来,生成一个li标签,li标签的内容,就是文本域,输入的内容,再把li标签放到ul里面
        // 获取所需对象
        var btn      = document.querySelector("button");
        var textarea = document.querySelector("textarea");
        var ul       = document.querySelector("ul");

        btn.onclick=function(){
            // 获取文本域中的内容
            var content =  textarea.value ;


            // 判断文本域内容是否为空
            if(content == ""){
                alert("您没有输入内容");
                // return可以终止函数的执行
                return ;
            }


            // console.log( textarea.innerHTML );// 获取文本域的值
            // console.log( textarea.innerText );// 获取文本域的值


            // 创建一个新的li标签
            var li = document.createElement("li");
            // 设置li的标签内容为文本域中的内容
            li.innerHTML = content;


            // 把li标签添加到ul中
            ul.appendChild( li );


            // 清空文本域的值
            textarea.value = "";
        }
    </script>
</body>


</html>
```

案例:新浪下拉菜单

分析

```
1. 导航栏里面的li都要有鼠标经过效果,所以需要循环给他们都绑定鼠标事件
2. 当鼠标经过li时,里面的第二个孩子ul显示;当鼠标离开时候,ul隐藏
```

今日总结

xmind要做

今日作业

具体看作业文件夹