目标

- 事件监听
  - addEventListener()事件监听（IE9以及IE9以后支持） 掌握
  - attachEvent()事件监听（IE678才支持） 了解
  - 事件监听兼容性解决方案 了解
- 删除事件（解绑事件）
- 删除事件兼容性解决方案 了解
- DOM事件流
  - 事件冒泡
  - 事件捕获
- 事件对象
  - 什么是事件对象
  - 事件对象的使用
  - 事件对象的兼容性处理
  - 事件对象的属性和方法
  - e.target 和 this 的区别
- 阻止默认行为
- 阻止事件冒泡
- 今日总结
- 今日作业

作业讲评

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style>
```

```
        table {
          width: 600px;
          cursor: pointer;
        }


        table td {
          text-align: center;
        }


        table th {
          background-color: blueviolet;
        }


        table tr {
          background-color: pink;
        }
    </style>
</head>
<body>
    <div id="dv">
        请输入姓名:
        <input type="text" value="" id="uname"/>
        <br/> 请输入邮箱:
        <input type="text" value="" id="email"/>
    </div>

    <br/>
    <input type="button" value="添加" id="btn" />
    <br/>
    <br/>


    <table border="1" cellpadding="0" cellspacing="0" id="tb">
        <thead>
            <tr>
                <th>姓名</th>
                <th>邮箱</th>
            </tr>
        </thead>
        <tbody id="tbd">
            <tr>
                <td>小黑</td>
                <td>xiaohei@163.com</td>
            </tr>
        </tbody>
    </table>



    <script>
        // 获取所需对象
```

```
        var btn = document.getElementById("btn");
        var uname = document.getElementById("uname");
        var email = document.getElementById("email");
        var tbd = document.getElementById("tbd");


        // 给按钮绑定单击事件
        btn.onclick=function(){
            // 获取姓名跟邮箱输入框的内容
            var unameValue = uname.value;
            var emailValue = email.value;
            if(unameValue=="" || emailValue==""){
                alert("请将数据填入完全");
                return ;
            }


            // 创建td标签以及设置td中标签的内容
            var td1 = document.createElement("td");
            td1.innerHTML = unameValue;


            var td2 = document.createElement("td");
            td2.innerHTML = emailValue;


            // 创建tr标签,把上面两个td添加tr中
            var tr = document.createElement("tr");
            tr.appendChild(td1);
            tr.appendChild(td2);


            // 把tr添加到tbody中
            tbd.appendChild( tr );


            uname.value = "";
            email.value = "";
        }
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <!-- 一组复选框 要求name属性值要一样的  -->
   爱好:篮球 <input type="checkbox" name="hobby">
         足球 <input type="checkbox" name="hobby">
         台球 <input type="checkbox" name="hobby">
         羽毛球 <input type="checkbox" name="hobby">
         保龄球 <input type="checkbox" name="hobby">
         <br/>
    <input type="button" value="全选" onclick="quanxuan()">
    <input type="button" value="取消" onclick="quxiao()">
    <input type="button" value="反选" onclick="fanxuan()">


    <script>
        // 第一种 通过name属性值获取标签对象 document.getElementsByName("name属性值") 得到也是一个集合,也就是伪数组
        // var checkboxs = document.getElementsByName("hobby");
        // console.log( checkboxs );


        // 第二种 使用css3中的属性选择器(想了解的话,可以百度)
        // var checkboxs = document.querySelectorAll("input[name='hobby']");
        // console.log( checkboxs );


        // 获取所有name为hobby的input标签
        var checkboxs = document.getElementsByName("hobby");


        function quanxuan(){
            for(var i=0;i<checkboxs.length;i++){
                checkboxs[i].checked = true;
            }
        }
        function quxiao(){
            for(var i=0;i<checkboxs.length;i++){
                checkboxs[i].checked = false;
            }
        }
        function fanxuan(){
            for(var i=0;i<checkboxs.length;i++){
                // 先获取原来的选中状态,接着取反,赋值赋回给 复选框的选中状态
                checkboxs[i].checked =  !checkboxs[i].checked ;
            }
        }

    </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;
        }
```

```
        li {
            list-style-type: none;
        }

        a {
            text-decoration: none;
            font-size: 14px;
        }

        .nav {
            margin: 100px;
        }

        .nav>li {
            position: relative;
            float: left;
            width: 125px;
            height: 41px;
            text-indent: 15px;
        }

        .nav li a {
            display: block;
            width: 100%;
            height: 100%;
            line-height: 41px;
            color: #333;
        }

        .nav>li>a:hover {
            background-color: #eee;
        }

        .nav ul {
            display: none;
            position: absolute;
            top: 41px;
            left: 0;
            width: 100%;
            border-left: 1px solid #FECC5B;
            border-right: 1px solid #FECC5B;
        }

        .nav ul li {
            border-bottom: 1px solid #FECC5B;
            text-indent: 15px;
        }

        .nav ul li a:hover {
            background-color: #FFF5DA;
        }
    </style>
</head>
```

```
<body>
    <ul class="nav">
        <li>
            <a href="#">微博</a>
            <ul>
                <li>
                    <a href="#">私信</a>
                </li>
                <li>
                    <a href="#">评论</a>
                </li>
                <li>
                    <a href="#">@我</a>
                </li>
            </ul>
        </li>
        <li>
            <a href="#">博客</a>
            <ul>
                <li>
                    <a href="#">博客评论</a>
                </li>
                <li>
                    <a href="#">未读提醒</a>
                </li>
            </ul>
        </li>
        <li>
            <a href="#">邮箱</a>
            <ul>
                <li>
                    <a href="#">免费邮箱</a>
                </li>
                <li>
                    <a href="#">VIP邮箱</a>
                </li>
                <li>
                    <a href="#">企业邮箱</a>
                </li>
                <li>
                    <a href="#">新浪邮箱客户端</a>
                </li>
            </ul>
        </li>
    </ul>


    <script>
        // 获取.nav子元素li
        // var lis = document.querySelectorAll(".nav>li");
        // console.log( lis );
```

```
        // 获取.nav子元素li
        // var nav = document.querySelector(".nav");
        // console.log( nav.children );


        // 获取.nav子元素li
        var lis = document.querySelectorAll(".nav>li");
        for(var i=0;i<lis.length;i++){
            // 给每个li绑定鼠标移上事件
            lis[i].onmouseover=function(){
                // console.log( this );
                // console.log( this.children );
                // console.log( this.children[1] );


                // 显示li下面的ul
                this.children[1].style.display = "block";
            }


            lis[i].onmouseout=function(){
                this.children[1].style.display = "none";
            }
        }
    </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>
        .register {
            width: 600px;
            margin: 100px auto;
        }

        .register .message {
            display: inline-block;
            font-size: 12px;
            color: #999;
            background: url(images/mess.png) no-repeat left center;
```

```
            padding-left: 20px;
        }

        .register .wrong {
            color: red;
            background-image: url(images/wrong.png);
        }

        .register .right {
            color: green;
            background-image: url(images/right.png);
        }
    </style>
</head>


<body>
    <div class="register">
        <input type="password" class="ipt">
        <p class="message">请输入6~16位密码</p>
    </div>


    <script>
        // 获取对象
        var input = document.querySelector("input");
        var p = document.querySelector(".message");


        input.onblur = function(){
            // console.log( input.value );
            // console.log( input.value.length );


            // 输入的内容长度不在6~16位之间 就提示用户以及改变叉叉图片
            var len = input.value.length;
            if(len<6 || len>16){
                p.innerHTML = "您输入的位数不对,要求6~16位";
                p.className = "message wrong";
            }else{
                p.innerHTML = "您输入的正确";
                p.className = "message right";
            }
        }
    </script>
</body>


</html>
```

学习目标

- 能够使用removeChild()方法删除节点
- 能够完成动态生成表格案例
- 能够使用传统方式和监听方式给元素注册事件
- 能够说出事件流执行的三个阶段
- 能够在事件处理函数中获取事件对象
- 能够使用事件对象取消默认行为
- 能够使用事件对象阻止事件冒泡
- 能够使用事件对象获取鼠标的位置
- 能够完成跟随鼠标的天使案例

节点操作

删除节点

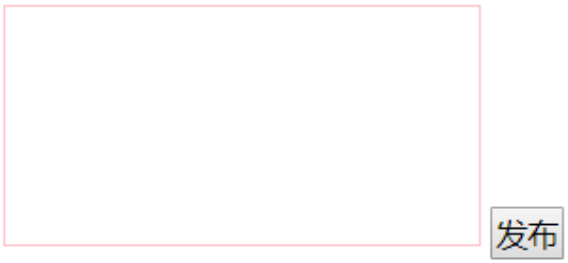
```
父节点.removeChild(节点)      从父节点中删除一个子节点,返回值是被删除的节点
```

举例:点击按钮每次删除父元素中的第一个子元素

```
<button>删除</button>
<ul>
    <li>熊大</li>
    <li>熊二</li>
    <li>光头强</li>
</ul>
```

```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>
    <button>删除</button>
    <ul>
        <li>熊大</li>
        <li>熊二</li>
        <li>光头强</li>
    </ul>


    <script>
        // 获取按钮对象
        var btn = document.querySelector("button");

        // 获取ul对象
        var ul = document.querySelector("ul");


        btn.onclick=function(){
            // 点击按钮每次删除父元素中的第一个子元素


            // 获取ul的子元素
            // console.log( ul.children );
            // 获取ul的第一个子元素  children会动态更新
            // console.log( ul.children[0] );


            if(ul.children.length > 0){// 判断ul的子元素列表长度是否大于0
                // 父对象.removeChild(子对象)  从父节点中删除一个子节点,返回值是被删除的节点
                ul.removeChild( ul.children[0] );
            }else{
                alert("ul里面没有子元素了,删除失败");
            }
        }
    </script>
</body>
</html>
```

案例:删除留言



- 大大很帅 删除

分析:

1. 当我们把文本域里面的值赋给li的时候,多添加一个删除的空链接
2. 需要把所有的链接获取出来,当我们点击当前的链接的时候,删除当前链接所在的li
3. 阻止链接跳转需要设置href="javascript:void(0)"

代码:

```
<!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
            // javascript:void(0)也是空链接的一种,点击以后页面不会回到顶部,而且点击a链接以后,字体的颜色不会变化
            li.innerHTML = content+"<a href='javascript:void(0)' style='float:right'>删除</a>";


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


            // 清空文本域的值
            textarea.value = "";


            // 获取li里面的a标签,因为a标签是点击按钮以后,跟随li生成的,这个a标签不是一打开网页就有的
            var as = document.querySelectorAll("ul li a");
            // console.log( as );
            for(var i=0;i<as.length;i++){
                as[i].onclick=function(){
                    // console.log( this );
                    // console.log( this.parentNode );


                    // 用户确认框 window.confirm("内容") 点击确定返回true 点击取消返回false


                    if(  window.confirm("你真的要删除人家嘛?")  ){
                        // 点击a标签以后,删除对应的留言,其实就是删除a标签所在的li标签
                        // 父元素.removeChild(子元素);


                        ul.removeChild( this.parentNode );
                    }else{
                        alert("讨厌!就知道你不舍得人家");
                    }

                }
            }
        }


    </script>
</body>


</html>
```

复制（克隆）节点

节点.cloneNode() 克隆节点/拷贝节点 返回值是节点的一个副本

注意:

1. 如果括号参数为**空或者false**,则是**浅拷贝**,即只克隆复制节点本身,不克隆里面的子节点
2. 如果括号参数为**true**,则是**深度拷贝**,会复制节点本身以及里面所有的子节点

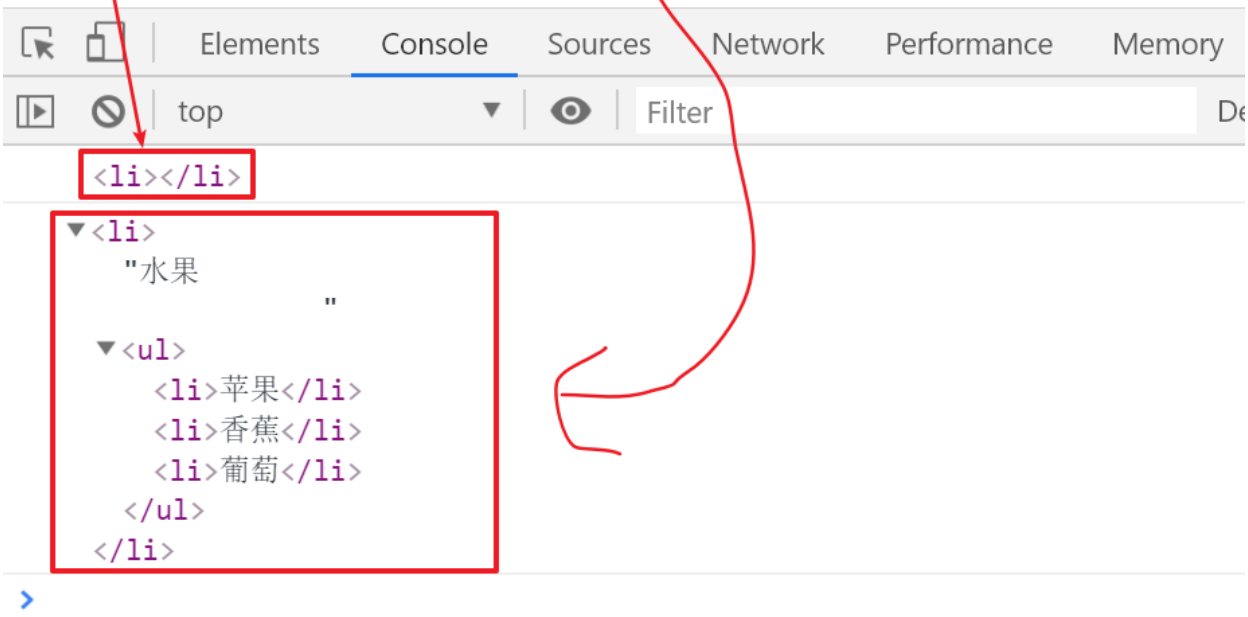
举例:

```
<ul>
    <li>水果
        <ul>
            <li>苹果</li>
            <li>香蕉</li>
            <li>葡萄</li>
        </ul>
    </li>
    <li>手机</li>
     <li>电脑</li>
 </ul>
```

```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>水果
            <ul>
                <li>苹果</li>
                <li>香蕉</li>
                <li>葡萄</li>
            </ul>
        </li>
        <li>手机</li>
        <li>电脑</li>
    </ul>


    <button>浅拷贝</button>
    <button>深拷贝</button>


    <script>
        // 复制（克隆）节点
        // 节点.cloneNode(参数)   克隆节点/拷贝节点   返回值是节点的一个副本
        // 注意:
        // 如果括号参数为空或者false,则是"浅拷贝",即只克隆复制节点本身,"不克隆里面的子节点"
        // 如果括号参数为true,则是"深度拷贝",会复制节点本身以及里面"所有的子节点"

        // 获取两个按钮对象
        var btns = document.querySelectorAll("button");
        // 获取ul中的第一个li标签,也就是水果li
        var ul = document.querySelector("ul");
        // console.log( ul );
        // console.log( ul.children );
        var fruits =  ul.children[0];


        btns[0].onclick=function(){
            // 如果括号参数为空或者false,则是"浅拷贝",即只克隆复制节点本身,"不克隆里面的子节点"
            // console.log( fruits.cloneNode() );
            console.log( fruits.cloneNode(false) );
        }
        btns[1].onclick=function(){
             // 如果括号参数为true,则是"深度拷贝",会复制节点本身以及里面"所有的子节点"
            console.log( fruits.cloneNode(true) );
        }
</script>
</body>
</html>
```

- 水果
  - 苹果
  - 香蕉
  - 葡萄
- 手机
- 电脑

浅拷贝 深拷贝



创建元素的三种方式

1. **document.write()**
2. **element.innerHTML**
3. **document.createElement()**

区别:

1. document.write是直接将内容写入页面的内容流,但是文档流执行完毕,则它会导致页面全部重绘
2. innerHTML是将内容写入某个DOM节点,不会导致页面全部重绘

3. innerHTML创建多个元素效率更好(可以拼接字符串,也可以采取数组形式拼接),结构稍微复杂

4. createElement()创建多个元素效率稍低一点点,但是结构更清晰

总结:不同浏览器下,innerHTML效率要比createElement高

举例:

```
<button>点击</button>
<p>abc</p>
<div class="inner"></div>
<div class="create"></div>

1. 给按钮绑定单击事件,点击以后,在网页中输出一个div标签
2. 字符串拼接的方式拼接100个a链接内容到.inner层中的内容
3. 定义一个空数组,给数组添加100个元素,每个元素的内容都是<a href="#">百度</a>,最后
把数组通过arr.join拼接成字符串赋值给.inner层中的内容
 4. 使用for循环配合document.createElement以及父节点.appendChild添加一百个a链接
 到.create层中
```

```
<!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>
    <button>点击</button>
    <p>abc</p>
    <div class="inner"></div>
    <div class="create"></div>


    <script>
        /* 创建元素的三种方式
        1. document.write()
        2. element.innerHTML
        3. document.createElement() */


        // 区别:
        // document.write是直接将内容写入页面的内容流,但是文档流执行完毕,则它会导致页面全部重绘
        // innerHTML是将内容写入某个DOM节点,不会导致页面全部重绘
        // innerHTML创建多个元素效率更好(可以拼接字符串,也可以采取数组形式拼接),结构稍微复杂
        // createElement()创建多个元素效率稍低一点点,但是结构更清晰


        // 1. 给按钮绑定单击事件,点击以后,在网页中输出一个div标签
        var btn = document.querySelector("button");
        btn.onclick=function(){
            document.write("<div>我是div标签内容</div>");
        }


        // 2. 字符串拼接的方式拼接100个a链接内容到.inner层中的内容
        /* var inner = document.querySelector(".inner");

        var str = "";
        for(var i=1;i<=100;i++){
            str += "<a href='https://www.baidu.com'>百度</a>";
        }
        inner.innerHTML = str; */



        // 3. 定义一个空数组,给数组添加100个元素,每个元素的内容都是<a href="#">百度</a>,最后,把数组通过arr.join拼接成字符串赋值给.inner层中的内容
        var inner = document.querySelector(".inner");


        var arr = [];
        for(var i=1;i<=100;i++){
            arr.push("<a href='https://www.jd.com'>京东</a>");
        }
        // console.log( arr );


        // 数组转成字符串
        var str = arr.join("");
        // console.log( str );
        inner.innerHTML = str;


        // 4. 使用for循环配合document.createElement以及父节点.appendChild添加一百个a链到.create层中
        var create = document.querySelector(".create");

        for(var j=1;j<=100;j++){
            var a = document.createElement("a");
            a.href = "https://www.qq.com";
            a.innerHTML = "腾讯";
            create.appendChild( a );
        }
    </script>
</body>
</html>
```

innerTHML和createElement效率对比

innerHTML字符串拼接方式（效率低）

```
<script>
 var d1 = +new Date();

    var str = '';
```

```
    for (var i = 0; i < 1000; i++) {
        document.body.innerHTML += '<div style="width:100px; height:2px; border:1px solid blue;"></div>';
    }

    var d2 = +new Date();
     console.log(d2 - d1);
</script>
```

innerHTML数组方式（效率高）

```
    <script>
        var d1 = +new Date();
        var array = [];
        for (var i = 0; i < 1000; i++) {
            array.push('<div style="width:100px; height:2px; border:1px solid blue;"></div>');
        }
        document.body.innerHTML = array.join('');
        var d2 = +new Date();
        console.log(d2 - d1);
     </script>
```

createElement方式（效率一般）

```
    <script>
        var d1 = +new Date();

        for (var i = 0; i < 1000; i++) {
            var div = document.createElement('div');
            div.style.width = '100px';
            div.style.height = '2px';
            div.style.border = '1px solid blue';
            document.body.appendChild(div);
        }
         var d2 = +new Date();
         console.log(d2 - d1);
      </script>
```

练习:动态生成表格

姓名	科目	成绩	操作
小龙女	JavaScript	100	删除
杨过	HTML	98	删除
陆无双	CSS	99	删除
郭靖	Java	88	删除
黄蓉	PHP	20	删除

分析:

1. 因为里面的学生数据都是动态的,我们需要js动态生成. 这里我们模拟数据,自己定义好数据,数据我们采取数组对象形式存储
2. 所有的数据都是放在tbody里面的行里面
3. 因为行很多,我们需要循环创建多个行(对应多少人)
4. 每个行里面又有很多单元格(对应里面的数据),我们还需要继续使用循环创建多个单元格,并且把数据存入里面(双重for循环)
5. 最后一列单元格是删除,需要单独创建单元格
6. 最后添加删除操作,点击删除,可以删除当前行

代码:

```
<table cellspacing="0">
 <thead>
 <tr>
 <th>姓名</th>
 <th>科目</th>
 <th>成绩</th>
 <th>操作</th>
 </tr>
 </thead>
  <tbody>

 </tbody>
</table>
 <script>
 // 1.先去准备好学生的数据
 var datas = [{
 name: '小龙女',
```

```
    subject: 'JavaScript',
    score: 100
    }, {
    name: '杨过',
    subject: 'HTML',
    score: 98
    }, {
    name: '陆无双',
    subject: 'CSS',
    score: 99
    }, {
    name: '郭靖',
    subject: 'Java',
    score: 88
    }, {
    name: '黄蓉',
    subject: 'PHP',
    score: 20
    }];
    </script>
```

```
<!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>
        thead{
            background: #cccccc;
        }
    </style>
</head>
<body>
    <table cellspacing="0" border="1" width="500">
        <thead>
            <tr height="50">
                <th>姓名</th>
                <th>科目</th>
                <th>成绩</th>
                <th>操作</th>
            </tr>
        </thead>


        <tbody>

        </tbody>
    </table>


    <script>
        // 1.先去准备好学生的数据,这样的数组可以叫数组对象,也可以叫json数组
        var datas = [{
```

```
        name: '小龙女',
        subject: 'JavaScript',
        score: 100
    }, {
        name: '杨过',
        subject: 'HTML',
        score: 98
    }, {
        name: '陆无双',
        subject: 'CSS',
        score: 99
    }, {
        name: '郭靖',
        subject: 'Java',
        score: 88
    }, {
        name: '黄蓉',
        subject: 'PHP',
        score: 20
    }];
    /* console.log( datas );
    console.log( datas[0] );
    console.log( datas[0]["name"] );
    console.log( datas[0]["score"] );
    console.log( datas[0]["subject"] ); */


    /* // 获取tbody标签
    var tbody = document.querySelector("tbody");


    // 创建行
    var tr = document.createElement("tr");
    tr.align = "center";


    // 创建列
    var td1 = document.createElement("td");
    td1.innerHTML = datas[0]["name"];


    var td2 = document.createElement("td");
    td2.innerHTML = datas[0]["subject"];


    var td3 = document.createElement("td");
    td3.innerHTML = datas[0]["score"];


    var td4 = document.createElement("td");
    td4.innerHTML = "<a href='javascript:void(0)'>删除</a>";


    // 再把列添加到行中
    tr.appendChild(td1);
    tr.appendChild(td2);
    tr.appendChild(td3);
    tr.appendChild(td4);

    // 最后把行添加到tbody中
    tbody.appendChild(tr); */



    /* // 获取tbody标签
    var tbody = document.querySelector("tbody");


    for(var i=0;i<datas.length;i++){
        // 创建行
```

```
        var tr = document.createElement("tr");
        tr.align = "center";


        for(var j in datas[i]){
            // console.log( datas[i][j] );
            // 创建列
            var td1 = document.createElement("td");
            td1.innerHTML = datas[i]["name"];


            var td2 = document.createElement("td");
            td2.innerHTML = datas[i]["subject"];


            var td3 = document.createElement("td");
            td3.innerHTML = datas[i]["score"];


            var td4 = document.createElement("td");
            td4.innerHTML = "<a href='javascript:void(0)'>删除</a>";
        }


        // 再把列添加到行中
        tr.appendChild(td1);
        tr.appendChild(td2);
        tr.appendChild(td3);
        tr.appendChild(td4);

        // 最后把行添加到tbody中
        tbody.appendChild(tr);
    } */


    // 获取tbody标签
    var tbody = document.querySelector("tbody");


    for(var i=0;i<datas.length;i++){
        var tr = document.createElement("tr");
        tr.align = "center";
        for(var j in datas[i]){
            var td = document.createElement("td");
            td.innerHTML = datas[i][j];
            tr.appendChild( td );
        }


        // 创建删除a链接
        var del = document.createElement("td");
        // 在行内绑定时候,可以给行内绑定参数传一个this关键字,那么这个this在函数那边需要有个形参接收
        del.innerHTML = "<a href='javascript:void(0)' onclick='deleteRow(this)'>删除<a>";
        // 把a链接添加到tr行中
        tr.appendChild( del );

        // 最后把tr添加到tbody中
        tbody.appendChild( tr );
    }


    function deleteRow(obj){
        // 默认这里的this代表 window对象
        // console.log( this );


        // obj接收行内绑定传过来的this,这个obj就是当前对象
        // console.log( obj );
```

```
            // console.log( obj.parentNode.parentNode );


            // console.log( tbody );


            tbody.removeChild( obj.parentNode.parentNode );
        }
    </script>
</body>
</html>
```

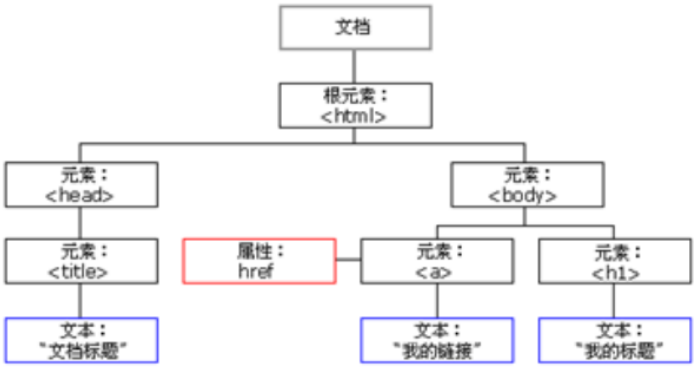
DOM的核心总结

文档对象模型(Document Object Model,简称DOM),是W3C组织推荐的处理可扩展标记语言(HTML或者XML)的标准编程接口

W3C已经定义了一系列的DOM接口,通过这些DOM接口可以改变网页的内容、结构和样式

1.对于Javascript,为了能够是JavaScript操作HTML,Javascript就有了一套自己的dom编程接口

2.对于HTML,dom使得html形成了一棵dom树,包含文档、元素、节点



我们获取过来的DOM元素是一个对象（object），所以称为 文档对象模型

关于dom操作，我们主要针对于元素的操作。主要有创建节点、增加节点、删除节点、改、查、属性操作、事件操作。

创建节点

1. document.write
2. innerHTML
3. createElement

增加节点

1. appendChild

2. insertBefore

删除节点

父节点.removeChild(子节点)

改

主要修改dom的元素属性,dom元素的内容、属性、表单的值等

1. 修改元素的属性: src、href、title等
2. 修改双标签元素内容: innerHTML、innerText
3. 修改表单元素: value、type、disabled、checked等
4. 修改元素样式: style、className

查

主要用于获取dom元素

1. DOM提供的方法: getElementById、getElementsByTagName
2. H5提供的新方法: getElementsByClassName、querySelector、querySelectorAll
3. 利用节点操作获取相应元素: 父节点(parentNode)、子节点(children)、兄弟节点(previousElementSibling,nextElementSibling)

属性操作

主要针对于自定义属性

1. setAttribute: 设置dom元素的属性值
2. getAttribute: 得到dom的属性值
3. removeAttribute移除属性

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
```

事件操作（重点）

事件三要素

- 事件源（谁）:触发事件的元素
- 事件类型（做什么）: 例如 onclick 点击事件
- 事件处理程序（怎么做）:事件触发后要执行的代码(函数形式)，事件处理函数

事件的绑定方式

事件的绑定方式有二种:**行内绑定与动态绑定**

行内绑定事件

说明:所谓的行内绑定事件 是将事件写在HTML标签里面

格式:

```
<标签名 事件名="函数名()" />
```

动态绑定事件 重点

说明:是将事件名写在JS代码里面

动态事件绑定的步骤如下:

```
第一步:获取事件源(获取对象)
第二步:注册事件(绑定事件)
第三步:添加事件处理程序(一般是一个匿名函数)
```

格式:

```
通过JS获取标签对象.事件名 = 事件的处理程序    事件的处理程序一般是一个"匿名函数"
```

事件高级

注册事件（2种方式）

给元素添加事件,称为注册事件或者绑定事件

注册事件有两种方式:传统方式和监听注册方式

传统注册方式	监听注册方式
利用 on 开头的事件 onclick	w3c 标准 推荐方式
<button onclick= "alert('hi~')" ></button>	addEventListener() 它是一个方法
btn.onclick = function() {}	IE9 之前的 IE 不支持此方法，可使用 attachEvent() 代替
特点： 注册事件的唯一性	特点：同一个元素同一个事件可以注册多个监听器
同一个元素同一个事件只能设置一个处理函数，最后注册的处理函数将会覆盖前面注册的处理函数	按注册顺序依次执行

事件监听

addEventListener()事件监听（IE9以及IE9以后支持） 掌握

```
eventTarget.addEventListener(type,listener[,useCapture])
```

eventTarget.addEventListener()方法将指定的监听器注册到 eventTarget（目标对象）上，当该对象触发指定的事件时，就会执行事件处理函数。

该方法接收三个参数:

- **type**:事件类型字符串,比如click、mouseover,注意这里不要带on
- **listener**:事件处理函数,事件发生时,会调用该监听函数
- **useCapture**:可选参数,是一个布尔值,默认是false.学会DOM事件流后,我们再进一步学习

attachEvent()事件监听（IE678才支持） 了解

```
eventTarget.attachEvent(eventNameWithOn,callback)
```

eventTarget.attachEvent()方法将指定的监听器注册到 eventTarget（目标对象）上，当该对象触发指定的事件时，指定的回调函数就会被执行。

该方法接收两个参数:

- **eventNameWithOn**:事件类型字符串,比如onclick、onmouseover,这里要带on
- **callback**:事件处理函数,当目标触发事件时回调函数被调用
- 注意:IE8以及早期版本才支持attachEvent

举例:

```
<button>传统注册事件</button>
<button>方法监听注册事件</button>
<button>ie9 attachEvent</button>

 1. 传统方式注册事件
 (1) 注册事件的唯一性
 (2) 同一个元素 同一个事件只能设置一个处理函数,最后注册的处理函数将会覆盖前面注册的处理函数

 2. 事件侦听注册事件 addEventListener
(1) 里面的事件类型是字符串 必定加引号 而且不带on
(2) 同一个元素 同一个事件可以添加多个侦听器(事件处理程序),按注册顺序依次执行

3. attachEvent ie9以前的版本支持
(1) 里面的事件类型是字符串 必定加引号 要带on
(2) 同一个元素 同一个事件可以添加多个侦听器(事件处理程序),按注册顺序倒序执行
```

```
<!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>
    <!-- 行内绑定事件,在标签中写事件属性 -->
    <button onclick="fn()">行内绑定-传统注册事件</button>
    <button>动态绑定-传统注册事件</button>
    <button>方法监听注册事件</button>
    <button>ie9 attachEvent</button>


    <script>
        function fn(){
            alert( "我是行内绑定的" );
        }
        // 给一个元素绑定事件,我们也可以称之为注册事件
        // 注册事件有两种方式,一种是传统方式,另一种是事件监听的方式
        var btns = document.getElementsByTagName("button");

        /* 1. 传统方式注册事件
        (1) 注册事件的唯一性
        (2) 同一个元素 同一个事件只能设置一个处理函数,最后注册的处理函数将会覆盖前面注册的处理函数 */
        btns[1].onclick = function(){
            alert("我是行内绑定的111111");
```

```
        }
        btns[1].onclick = function(){
            alert("我是行内绑定的2222222");
        }
        btns[1].onclick = function(){
            alert("我是行内绑定的333333333");
        }

        /* 2. 事件侦听注册事件 addEventListener  IE9以及IE9才可以用addEventListener方法绑定事件
        (1) 里面的事件类型是字符串 必定加引号 而且不带on
        (2) 同一个元素 同一个事件可以添加多个侦听器（事件处理程序）,按注册顺序依次执行 */


        // 语法: 对象.addEventListener("事件类型,不带on",匿名函数或者函数名 )
        btns[2].addEventListener("click",function(){
            alert("我是事件监听绑定的按钮11111");
        })
        btns[2].addEventListener("click",function(){
            alert("我是事件监听绑定的按钮22222");
        })
        btns[2].addEventListener("click",function(){
            alert("我是事件监听绑定的按钮333333");
        })


        // 如果是写函数名的话,不需要使用引号包裹,也不需要写括号
        btns[2].addEventListener("click", fn2 );


        function fn2(){
            alert("我是有名函数fn2");
        }

       /*  3. attachEvent ie9以前的版本支持    ie9以下如果要需要使用监听的方式,需要使用
        (1) 里面的事件类型是字符串 必定加引号 要带on
        (2) 同一个元素 同一个事件可以添加多个侦听器（事件处理程序）,按注册顺序倒序执行 */
        // 语法 对象.attachEvent("事件类型,带on",匿名函数或者函数名 )
        /* btns[3].attachEvent("onclick",function(){
            alert("我是IE9以下浏览器专用的绑定方式11111");
        });


        btns[3].attachEvent("onclick",function(){
            alert("我是IE9以下浏览器专用的绑定方式22222");
        });


        btns[3].attachEvent("onclick", fn3 );


        function fn3(){
            alert("我是有名函数fn3,我用于测试attachEvent监听方式");
        } */
    </script>
</body>
</html>
```

事件监听兼容性解决方案 了解

封装一个函数，函数中判断浏览器的类型:

```
function addEventListener(element,eventName,fn){
 // 判断当前浏览器是否支持addEventListener方法
 if(element.addEventListener){
 element.addEventListener(eventName,fn);// 第三个参数默认是false
 }else if(element.attachEvent){
 element.attachEvent("on"+eventName,fn);
 }else{
 // 相当于element.onclick = fn;
 element["on" + eventName] = fn;
  }
 }
```

测试代码:

```
<!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>
    <button>按钮</button>


    <script>
        /* 参数说明
        element代表要绑定事件的元素
        eventName事件类型,不带on
        fn匿名函数或者有名函数名 */
        function addEventListener(element,eventName,fn){
            // 判断当前浏览器是否支持addEventListener方法
            if(element.addEventListener){// 判断是否支持addEventListener方法
                element.addEventListener(eventName,fn);// 第三个参数默认是false
            }else if(element.attachEvent){// 判断是否支持attachEvent方法
                element.attachEvent("on"+eventName,fn);
            }else{
                // 相当于element.onclick = fn;
                element["on" + eventName] = fn;
            }
        }


        var btns = document.getElementsByTagName("button");
        addEventListener( btns[0] , "click", function(){
            alert(11111111);
        } );
    </script>
</body>
</html>
```

删除事件（解绑事件）

1. 传统注册方式

```
eventTarget.on事件名称 = null;

比如:eventTarget.onclick = null;
```

2. 方法监听注册方式

```
① eventTarget.removeEventListener(type,listener[,useCapture]);
② eventTarget.detachEvent(eventNameWithOn,callback);
```

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>
    <style>
        div {
            width: 100px;
            height: 100px;
            background-color: pink;
            margin-bottom: 10px;
        }
    </style>
</head>
<body>
    <div>1</div>
    <div>2</div>
    <div>3</div>


    <script>
        // 目标,实现每个div只有被点击一次


        // 删除事件（解绑事件）,解除元素上所有绑定的指定的事件

        // 获取所有div标签
        var divs = document.getElementsByTagName("div");


        /* 传统注册方式
        eventTarget.on事件名称 = null;
        比如:eventTarget.onclick = null; */


        divs[0].onclick = function(){
            alert("我是第一个div");
            // 用disabled禁用div是没有效果的
            // divs[0].disabled = true;


            // 传统方式解绑  语法:    对象.on事件类名 = null
            divs[0].onclick = null;
        }

        /* 方法监听注册方式
        ① eventTarget.removeEventListener(type,listener[,useCapture]);
        ② eventTarget.detachEvent(eventNameWithOn,callback); */


        // 给第二个div使用监听的方式绑定事件,需要后续需要解绑,请使用有名函数
        divs[1].addEventListener("click", fn1 );

        function fn1(){
            alert("我是第二个div");


            //语法 eventTarget.removeEventListener(type,listener[,useCapture])
            // 如果要解绑监听方法绑定的事件,需要是有名函数
            divs[1].removeEventListener("click" , fn1 );
        }


        // 给第三个div使用监听的方式绑定事件,需要后续需要解绑,请使用有名函数(这个我们使用attachEvent方法绑定,测试IE9以下的浏览器)
        /* divs[2].attachEvent("onclick", fn2 );


        function fn2(){
            alert("我是第三个div,我用于测试IE9以下的浏览器");
            // 解绑语法 eventTarget.detachEvent(eventNameWithOn,callback);
            divs[2].detachEvent( "onclick", fn2);
        } */



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

删除事件兼容性解决方案　了解

```
// 参数说明
// element是元素,
// eventName是事件类名,不带on
// fn可以传匿名函数,也可以传有名函数名

function removeEventListener(element,eventName,fn){
 // 判断当前浏览器是否支持 removeEventListener 方法
 if(element.removeEventListener){
 element.removeEventListener(eventName,fn);// 第三个参数 默认是false
  }else if(element.detachEvent){
  element.detachEvent("on"+eventName,fn);
  }else{
  element["on" + eventName] = null;
  }
}
```

DOM事件流

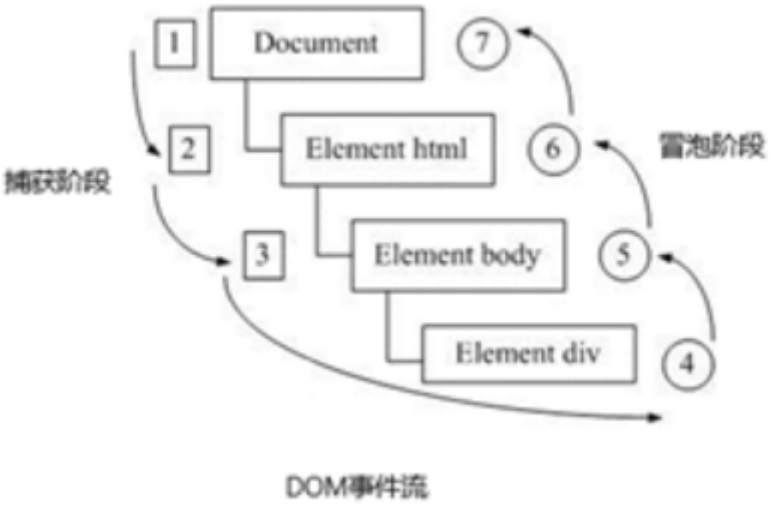
html中的标签都是相互嵌套的，我们可以将元素想象成一个盒子装一个盒子，document是最外面的大盒子。
当你单击一个div时，同时你也单击了div的父元素，甚至整个页面。

那么是先执行父元素的单击事件，还是先执行div的单击事件？

事件流描述的是从页面中接收事件的顺序.

事件发生时会在元素节点之间按照特定的顺序传播,这个传播过程,我们就称之为DOM事件流.

> 比如:我们给页面中的一个div注册了单击事件，当你单击了div时，也就单击了body，单击了html，单击了document。



DOM事件流

事件冒泡:IE最早提出,事件开始时由最具体的元素接收,然后逐级向上传播到DOM最顶层节点的过程.

事件捕获:网景最早提出,由DOM最顶层节点开始,然后逐级向下传播到最具体的元素接收的过程.

当时的2大浏览器霸主谁也不服谁!
IE 提出从目标元素开始，然后一层一层向外接收事件并响应，也就是冒泡型事件流。
Netscape（网景公司）提出从最外层开始，然后一层一层向内接收事件并响应，也就是捕获型事件流。

江湖纷争，武林盟主也脑壳疼！！！

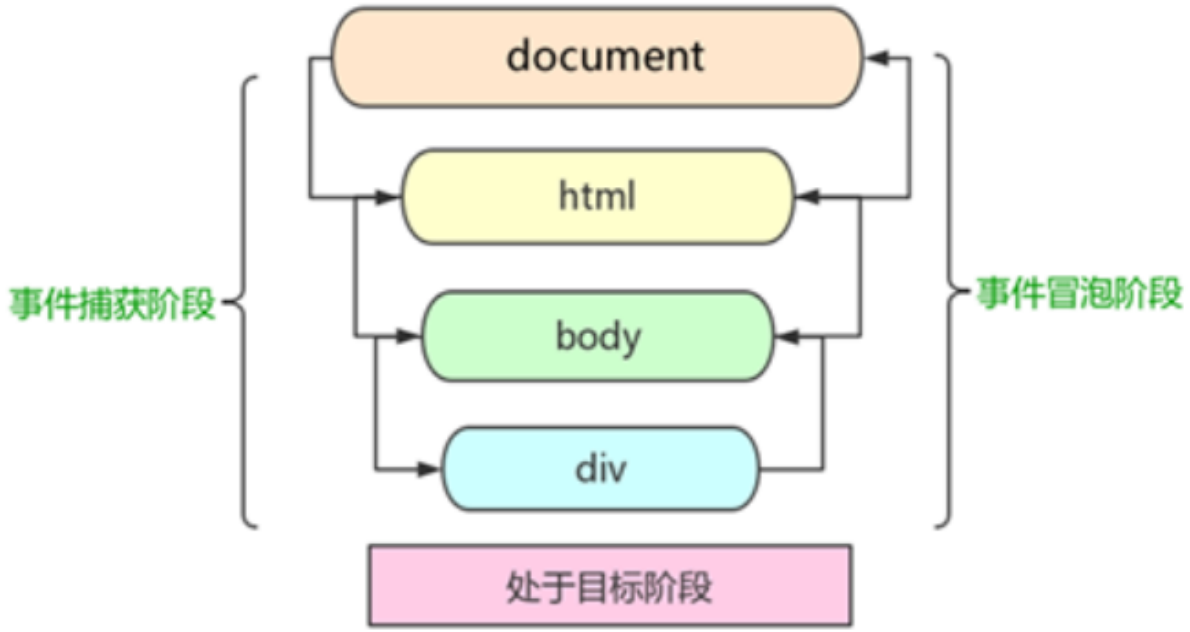
最终，w3c 采用折中的方式，平息了战火，制定了统一的标准 ——— 先捕获再冒泡。
现代浏览器都遵循了此标准，所以当事件发生时，会经历3个阶段。

DOM 事件流会经历3个阶段:

1. **捕获阶段**
2. **当前目标阶段**
3. **冒泡阶段**

为了帮助大家理解,我们向水里面扔一块石头，首先它会有一个下降的过程，这个过程就可以理解为从最顶层向事件发生的最具体元素（目标点）的捕获过程;

之后会产生泡泡，会在最低点（最具体元素）之后漂浮到水面上，这个过程相当于事件冒泡。



事件发生时会在元素节点之间按照特定的顺序传播,这个传播过程,我们就称之为DOM事件流.

注意:

1. JS代码中只能执行捕获或者冒泡两个阶段中的**其中一个阶段**
2. **onclick传统绑定事件跟attachEvent监听绑定事件的方式只能得到冒泡阶段**
3. addEventListener(type,listener,[,useCapture])这种监听方式,才可以掩饰捕获阶段; 第三个参数如果是true,表示在事件捕获阶段调用事件处理程序; 如果是false(默认不写就是false),表示事件冒泡阶段调用事件处理程序.
4. 实际开发中我们**很少使用事件捕获,我们更关注事件冒泡**.
5. **有些事件是没有冒泡的**,比如 **onblur,onfocus,onmouseenter,onmouseleave**
6. 事件冒泡有时候会带来麻烦,有时候又会帮助我们很巧妙的做某些事件,我们后面讲解

事件冒泡

```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>
        *{
            margin: 0;
            padding: 0;
        }
        .father{
            width: 480px;
            height: 480px;
            background: skyblue;
            padding:10px;
            margin:50px auto;
        }
        .father .son{
            width: 100px;
            height: 100px;
            background: pink;
            margin:0 auto;
            margin-top: 200px;
        }
    </style>
</head>
<body>
    <div class="father">
        <div class="son">son盒子</div>
     </div>


     <script>
        // 注意:
        // JS代码中只能执行捕获或者冒泡两个阶段中的其中一个阶段
        // onclick传统绑定事件跟attachEvent监听绑定事件的方式只能得到冒泡阶段
        // addEventListener(type,listener,[,useCapture])这种监听方式,才可以掩饰捕获阶段; 第三个参数如果是true,表示在事件捕获阶段调用事件处理程序;  如果是false(默认不写就是false),表示事件冒泡阶段调用事件处理程序.
        // 实际开发中我们很少使用事件捕获,我们更关注事件冒泡.
        // 有些事件是没有冒泡的,比如onblur,onfocus,onmouseenter,onmouseleave
        // 事件冒泡有时候会带来麻烦,有时候又会帮助我们很巧妙的做某些事件,我们后面讲解


        //  为了演示事件冒泡,我们分别给document,html,body,.father, .son都绑定鼠标单击事件
        // console.log( document );

        // 事件的冒泡阶段是从 目标阶段 开始往上,从下到上的过程,最顶层的是document


        // 给document绑定单击事件
        document.onclick=function(){
            alert("document点击事件");
        }
        // 给html绑定单击事件
        // console.log( document.documentElement );
        document.documentElement.onclick=function(){
            alert("html点击事件");
        }
        // 给body绑定单击事件
        document.body.onclick=function(){
            alert("body点击事件");
        }
        // 给.father绑定单击事件
        var father = document.querySelector(".father");
        father.onclick=function(){
            alert("father点击事件");
        }
        // 给.son绑定单击事件
        var son = document.querySelector(".son");
        son.onclick=function(){
            alert("son点击事件");
        }
     </script>
</body>
</html>
```

事件捕获

```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>
        *{
            margin: 0;
            padding: 0;
        }
        .father{
            width: 480px;
            height: 480px;
            background: skyblue;
            padding:10px;
            margin:50px auto;
        }
        .father .son{
            width: 100px;
            height: 100px;
            background: pink;
            margin:0 auto;
            margin-top: 200px;
        }
    </style>
</head>
<body>
    <div class="father">
        <div class="son">son盒子</div>
    </div>


    <script>
        // 注意:
        // JS代码中只能执行捕获或者冒泡两个阶段中的其中一个阶段
        // onclick传统绑定事件跟attachEvent监听绑定事件的方式只能得到冒泡阶段
        // addEventListener(type,listener,[,useCapture])这种监听方式,才可以掩饰捕获阶段; 第三个参数如果是true,表示在事件捕获阶段调用事件处理程序;  如果是false(默认不写就是false),表示事件冒泡阶段调用事件处理程序.
        // 实际开发中我们很少使用事件捕获,我们更关注事件冒泡.
        // 有些事件是没有冒泡的,比如onblur,onfocus,onmouseenter,onmouseleave
        // 事件冒泡有时候会带来麻烦,有时候又会帮助我们很巧妙的做某些事件,我们后面讲解


        // 事件的捕获阶段是从顶级document,由上到下到达目标阶段的过程
        // addEventListener监听绑定方式第三个参数没有或者设置false,表示冒泡阶段
        // addEventListener监听绑定方式第三个参数设置为 true ,表示捕获阶段
        // 给document绑定单击事件
        document.addEventListener("click",function(){
            alert("document点击事件");
        },true)


        // 给html绑定单击事件
        // console.log( document.documentElement );
        document.documentElement.addEventListener("click",function(){
            alert("html点击事件");
        },true)

        // 给body绑定单击事件
        document.body.addEventListener("click",function(){
            alert("body点击事件");
        },true)
        // 给.father绑定单击事件
        var father = document.querySelector(".father");
        father.addEventListener("click",function(){
            alert("father点击事件");
        },true)


        // 给.son绑定单击事件
        var son = document.querySelector(".son");
        son.addEventListener("click",function(){
            alert("son点击事件");
        },true)

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

事件对象

什么是事件对象

就是在事件发生后，跟事件相关的一系列信息数据的集合都放到这个对象里面，这个对象就是事件对象。

比如:

1. 谁绑定了这个事件。
2. 鼠标触发事件的话，会得到鼠标的相关信息，如鼠标位置。
3. 键盘触发事件的话，会得到键盘的相关信息，如按了哪个键。

事件对象的使用

事件触发发生时就会**产生事件对象**，并且**系统会以实参的形式**传给事件处理函数。

所以，**在事件处理函数中声明1个形参用来接收事件对象**。

```
eventTarget.onclick = function(event){
  // 这个event 就是事件对象,我们还喜欢写成e或者evt
};
```

```
eventTarget.addEventListener("click",function(event){
  // 这个event 就是事件对象,我们还喜欢写成e或者evt
});
```

```
eventTarget.attachEvent("onclick",function(event){
  // 这个event 就是事件对象,我们还喜欢写成e或者evt
});
```

事件对象的兼容性处理

事件对象本身的获取存在兼容问题:

1. 标准浏览器中是浏览器给方法传递的参数，只需要定义形参 e 就可以获取到。

2. 在 IE6~8 中，浏览器不会给方法传递参数，如果需要的话，需要到 window.event 中获取查找。

解决方法:

```
e = e || window.event;
```

特别说明:

- 只要"||"前面为false, 不管"||"后面是true 还是 false，都返回 "||" 后面的值。
- 只要"||"前面为true，不管"||"后面是true 还是 false，都返回 "||" 前面的值。

```
console.log(false || 222);  // 222
console.log(true || 222); // true

console.log(undefined || 222);  // 222
console.log("abc" || 222);// "abc"
```

代码:

```
<!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>
    <button>按钮1</button>
    <button>按钮2</button>
    <button>按钮3</button>


    <script>
        // 事件对象的使用
        // 事件触发发生时就会产生事件对象，并且系统会以实参的形式传给事件处理函数。
        // 所以，在事件处理函数中声明1个形参用来接收事件对象。


        // 事件对象的兼容性处理
        // 事件对象本身的获取存在兼容问题：
        // 1. 标准浏览器中是浏览器给方法传递的参数，只需要定义形参 e 就可以获取到。
        // 2. 在 IE6~8 中，浏览器不会给方法传递参数，如果需要的话，需要到 window.event 中获取查找。
        // 解决方法：
        // e = e || window.event;
        // 特别说明：
        // 只要"||"前面为false, 不管"||"后面是true 还是 false，都返回 "||" 后面的值。
        // 只要"||"前面为true,  不管"||"后面是true 还是 false，都返回 "||" 前面的值。

        var btns = document.getElementsByTagName("button");
        /* eventTarget.onclick = function(event){
        // 这个event 就是事件对象,我们还喜欢写成e或者evt
        }; */


        btns[0].onclick=function(e){
            // 这个event 就是事件对象,我们还喜欢写成e或者evt或者event
            // console.log( event );
            // window.event是关键字,一个字母都不能改
            // console.log( window.event );


            // 如果e不为undefined,我们就得到e;如果e为undefined,我们就得到window.event
            e = e || window.event;
```

```
            console.log( e );
        }



        /* eventTarget.addEventListener("click",function(event){
        // 这个event 就是事件对象,我们还喜欢写成e或者evt
        }); */


        /* btns[1].addEventListener("click",function(e){
            console.log( e );
        }) */



        /* eventTarget.attachEvent("onclick",function(event){
        // 这个event 就是事件对象,我们还喜欢写成e或者evt
        }); */


        /* btns[2].attachEvent("onclick",function( evt ){
            // 这个event 就是事件对象,我们还喜欢写成e或者evt
            console.log( evt );
        }); */
    </script>
</body>
</html>
```

事件对象的属性和方法

事件对象属性方法	说明
e.target	返回触发事件的对象 标准
e.srcElement	返回触发事件的对象 非标准 ie6~8使用
e.type	返回事件的类型 比如 click mouseover 不带on
e.stopPropagation()	该方法阻止冒泡 标准
e.cancelBubble = true	该属性阻止冒泡 非标准 ie6~8使用
e.preventDefault()	该方法阻止默认事件(默认行为) 标准 比如不让链接跳转
e.returnValue = false	该属性阻止默认事件(默认行为) 非标准 ie6~8使用 比如不让链接跳转
return false	利用return false 也能阻止默认行为 没有兼容问题 但是return后面的代码不执行

e.target 和 this 的区别

- this 是事件绑定的元素（绑定这个事件处理函数的元素）。
- e.target 是事件触发的元素。

通情况下target 和 this是一致的，

但有一种情况不同，那就是在**事件冒泡**时（父子元素有相同事件，单击子元素，父元素的事件处理函数也会被触发执行），

这时候this指向的是父元素，因为它是绑定事件的元素对象，

而target指向的是子元素，因为他是触发事件的那个具体元素对象。

举例: 通情况下e.target 和 this是一致的

```
<div>123</div>

// e.target 和 this指向的都是div
```

```
<!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>
        div{
            width: 200px;
            height: 200px;
            background: pink;
        }
    </style>
</head>
<body>
    <div>123</div>


    <script>
        // 获取div对象
        var div = document.getElementsByTagName("div")[0];
        // 通情况下e.target 和 this是一致的
        div.onclick=function( e ){
            // 兼容写法获取事件对象
            e = e || window.event;
            console.log( e );
            console.log("");

            // e.target 返回触发事件的对象
            console.log( e.target );
            console.log( e.srcElement );
            console.log( this );
        }
    </script>
</body>
</html>
```

举例:事件冒泡下的e.target和this 就有所不同了

```
<ul>
    <li>abc</li>
    <li>abc</li>
    <li>abc</li>
</ul>

// 我们给ul 绑定了事件 那么this 就指向ul
// e.target 触发了事件的对象 我们点击的是li e.target 指向的就是li
```

```
<!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>abc1111</li>
        <li>abc2222</li>
        <li>abc3333</li>
     </ul>

     <script>
        // 事件冒泡下的e.target和this  就有所不同了

        //  获取ul对象
        var ul = document.querySelector("ul");

        // 我们给ul 绑定了事件 那么this 就指向ul
        ul.onclick=function(e){
            e = e || window.event;
            // e.target 触发了事件的对象 我们点击的是li  e.target 指向的就是li
            console.log( e.target );
            console.log( this );
            console.log( "" );
        }
    </script>
</body>
</html>
```

阻止默认行为

html中一些标签有默认行为，例如a标签被单击后，默认会进行页面跳转。
form表单中的提交按钮,默认点击以后,也会提交表单数据

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>
    <style>
        div{
            width: 300px;
            height: 300px;
            background: skyblue;
        }
    </style>
</head>
<body>
    <div>123</div>


    <a href="http://www.baidu.com">百度</a>


    <form action="http://www.baidu.com">
        <input type="submit" value="提交" name="sub">
    </form>


    <script>
        var div = document.querySelector("div");
        div.onclick=function(e){
            e = e || window.event;
            // e.type返回事件类型,不带on
            console.log( e.type );
            console.log("");
        }
        div.onmouseover=function(e){
            e = e || window.event;
            // e.type返回事件类型,不带on
            console.log( e.type );
            console.log("");
        }
        div.onmouseout=function(e){
            e = e || window.event;
            // e.type返回事件类型,不带on
            console.log( e.type );
            console.log("");
        }


        // 2. 阻止默认行为（事件） 让链接不跳转 或者让 提交按钮不提交
        var a = document.querySelector("a");
        a.onclick=function(e){
            e =  e || window.event;
            // e.preventDefault() 该方法阻止默认事件(默认行为) 标准 比如不让链接跳转
            e.preventDefault();


            // e.returnValue = false 该属性阻止默认事件(默认行为) 非标准 ie6~8使用 比如不让链接跳转
            // e.returnValue = false;


            // return false 利用return false 也能阻止默认行为 没有兼容问题 但是return后面的代码不执行
```

```
            // return false;
        }


        // 3.阻止表单提交
        var input = document.querySelector("input");
        input.onclick=function(e){
            e = e || window.event;
            // 阻止默认行为
            e.preventDefault();


            // e.returnValue = false;


            // return false;
        }
    </script>
</body>
</html>
```

阻止事件冒泡

事件冒泡本身的特性，会带来的坏处，也会带来的好处。

- 标准写法:利用事件对象里面的stopPropagation()方法
- 非标准写法: IE 6~8 利用事件对象的cancelBubble = true属性

举例:

```
// 常见事件对象的属性和方法
e.stopPropagation(); // stop 停止 Propagation 传播

// 阻止冒泡 dom 推荐的标准 stopPropagation()
 e.cancelBubble = true; // 非标准 cancel 取消 bubble 泡泡
```

```
<!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>
        *{
            margin: 0;
            padding: 0;
        }
        .father{
            width: 480px;
            height: 480px;
            background: skyblue;
            padding:10px;
            margin:50px auto;
        }
```

```
        .father .son{
            width: 100px;
            height: 100px;
            background: pink;
            margin:0 auto;
            margin-top: 200px;
        }
    </style>
</head>
<body>
    <div class="father">
        <div class="son">son盒子</div>
     </div>


     <script>
        // 给document绑定单击事件
        document.onclick=function(){
            alert("document点击事件");
        }


        // 给html绑定单击事件
        document.documentElement.onclick=function(){
            alert("html点击事件");
        }
        // 给body绑定单击事件
        document.body.onclick=function(){
            alert("body点击事件");
        }
        // 给.father绑定单击事件
        var father = document.querySelector(".father");
        father.onclick=function(e){
            alert("father点击事件");


            // 非标准写法: IE 6~8 利用事件对象的cancelBubble = true 属性


            e = e || window.event;
            e.cancelBubble = true;
        }
        // 给.son绑定单击事件
        var son = document.querySelector(".son");
        son.onclick=function(e){
            alert("son点击事件");


            // 阻止事件冒泡
            // 事件冒泡本身的特性，会带来的坏处，也会带来的好处。
            // 标准写法:利用事件对象里面的e.stopPropagation()方法
            // 非标准写法: IE 6~8 利用事件对象的cancelBubble = true属性
            e = e || window.event;
```

```
            e.stopPropagation();
        }
    </script>
</body>
</html>
```

阻止事件冒泡的兼容性处理 要记 面试有可能会考

```
// 先判断有没有e事件对象,如果有,再判断e中stopPropagation存在吗
if(e && e.stopPropagation){
  e.stopPropagation();
}else{
   window.event.cancelBubble = true;
}
```

今日总结

xmind总结知识点

今日作业

- 把上课讲到事件高级-注册事件以及绑定事件的代码至少写一遍
- 把上课讲的事件冒泡和事件捕获的代码至少写一遍
- 把上课讲的获取事件对象的代码至少写一遍
- 把上课讲的事件对象的属性和方法敲一遍
- 冒泡兼容写法代码敲一遍, 并且验证是否可行