2级DOM事件

之前已经学习了0级DOM事件，0级DOM事件是以属性的形式存在的

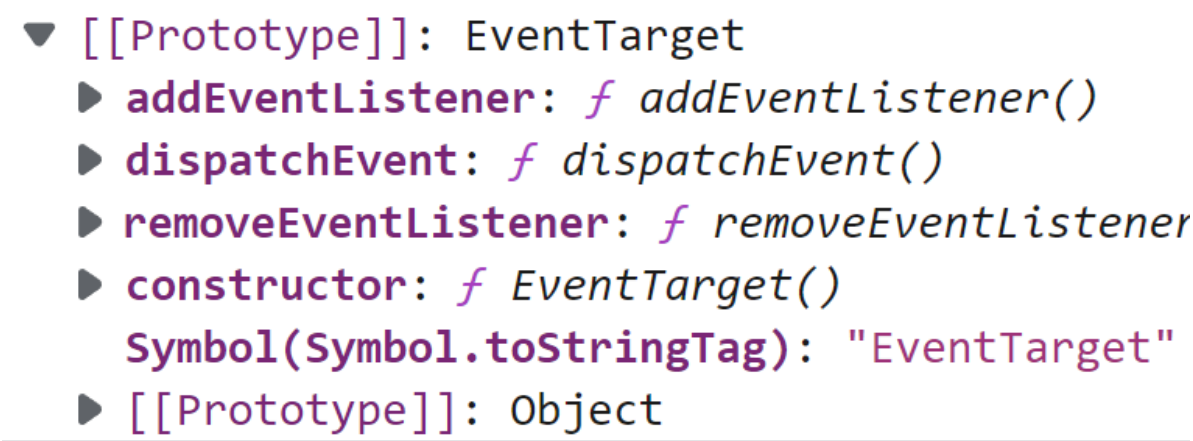
```
<!DOCTYPE html>
<html lang="zh">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>0级事件</title>
</head>

<body>
    <button onclick="sayHello(this,event)" type="button" id="btn1">按钮</button>
    <button type="button" id="btn2">按钮</button>
</body>
<script>
    // 0级事件绑定
    function sayHello(obj,event) {
        console.log(obj);
        console.log(event);
    }

    var btn2 = document.querySelector("#btn2");
    btn2.onclick = function(event){
        console.log(this);
        console.log(event);
    };
</script>

</html>
```

上面就是最基础的0级DOM事件，但是0级事件会有一些不足的地方

1. 0级事件总是由内向外传播，不能改变方向
2. 0级事件是以属性的形式存在，如果重复赋值，就会后面的覆盖前面的
3. 0级事件是以属性的形式存在，有这个属性就有这个事件，没有这个属性就没有这个事件，所以如果想实现自定义事件是不可能的

在之前讲DOM的时候，我们提到过一个DOM的结构层次图

元素素材	第一层	第二层	第三层	第四层	第五层
div标签	HTMLDivElement	HTMLElement	Element	Node	EventTarget
p标签	HTMLParagraphElement	HTMLElement	Element	Node	EventTarget
input标签	HTMLInputElement	HTMLElement	Element	Node	EventTarget
document	HTMLDocument	Document		Node	EventTarget



通过 DOM的结构层次分析，我们可以的看到在DOM的最高层里有一个`EventTarget`，这指的就是2级DOM事件

- `addEventListener()`添加事件监听
- `removeEventListener()`移除事件监听
- `dispatchEvent()`派发事件，后期自定义事件会使用到【设计模型里面的观察者模式也会用到】

事件监听

0级事件是依赖于`Element`上面的属性的形式存在，而2级事件则是依赖于`EventTarget`上面的存在，0级事件的绑定是通过属性赋值来完成，而2级事件则是通过**事件监听的方式来完成的**

```
dom.addEventListener("事件名",function(event){
    //代码体
},true/false);
```

> 在上面的语法规则里面，最后一个的参数代表事件的传播方向，**默认是false**代表事件由内向外传播，如果设置为true代表事件由外向内传播

我们可以通过上面的语法规则来完成一个简单的2级事件绑定

```
<body>
    <button type="button" id="btn1">按钮</button>
</body>
<script>
    var btn1 = document.querySelector("#btn1");
    /*
    btn1.onclick=function(event){
```

```
        console.log("我是0级事件");
    }
    */
    btn1.addEventListener("click", function (event) {
        console.log("我是2级DOM事件");
    });
</script>
```

> 原来的0级事件里面事件有名是有 `on` 的，在2级DOM事件里面是没有的

事件取消

在之前在0级DOM事件里面，因为事件是以属性的形式存在，所以我们如果想取消某个事件，直接将这个事件的属性名赋值为 `null` 就可以了，如下所示

```
<body>
    <div class="box"></div>
    <button type="button" class="btn1">按钮</button>
</body>
<script>
    var box = document.querySelector(".box");
    var btn1 = document.querySelector(".btn1");
    box.onclick = function (event) {
        console.log("我是一个盒子")
    }

    btn1.onclick = function(event){
        console.log("取消了box的事件");
        //取消事件
        box.onclick = null;
    }
</script>
```

但是这种方法在2级DOM事件里面是不行的，因为2级DOM事件是以事件监听的方式存在

```
<body>
    <div class="box"></div>
    <button type="button" class="btn1">按钮</button>
</body>
<script>
    var box = document.querySelector(".box");
    var btn1 = document.querySelector(".btn1");

    function aaa() {
        console.log("我是box里面的事件")
```

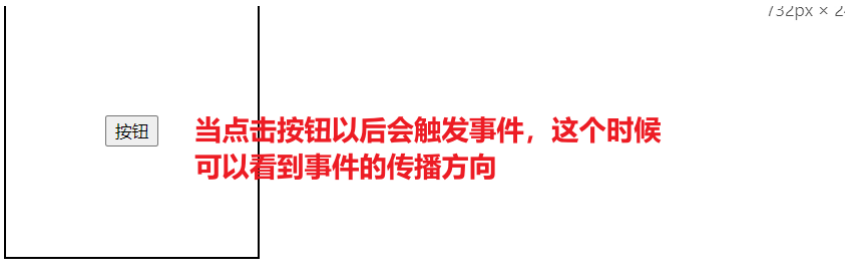
```
    }
    box.addEventListener("click", aaa);

    btn1.addEventListener("click", function () {
        //去移除box的事件
        console.log("移除box的事件");
        box.removeEventListener("click",aaa);
    })
</script>
```

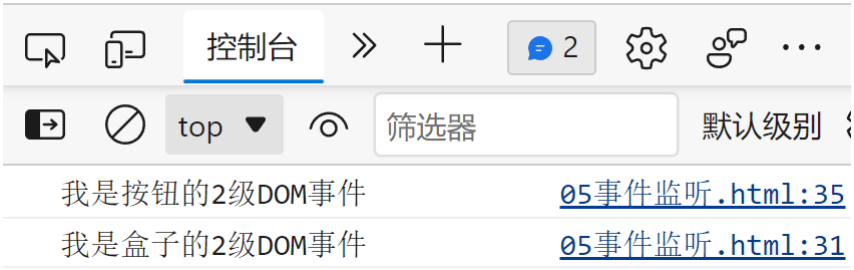
如果要移除一个2级DOM的事件，我们就要使用`removeEventListener()`来完成

事件的传播方向

在默认情况下，事件总是由内向外进行传播的，同时在2级DOM事件里面，最后一个参数默认是`false`代表的是事件冒泡（事件冒泡：事件由内向外传播），最后一个参数如果是`true`代表事件捕获（事件捕获：事件由外向内传）

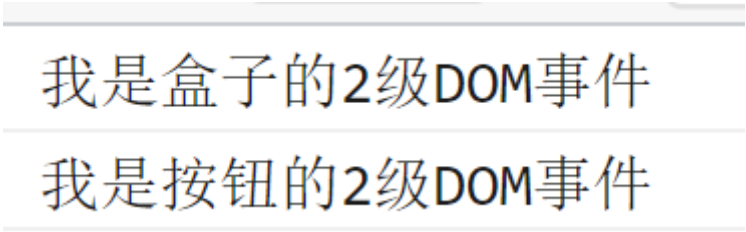
```
<body>
    <div class="box">
        <button type="button" id="btn1">按钮</button>
    </div>
</body>
<script>
    var box = document.querySelector(".box");
    var btn1 = document.querySelector("#btn1");

    box.addEventListener("click",function(event){
        console.log("我是盒子的2级DOM事件");
    },false);           //事件由内向外

    btn1.addEventListener("click", function (event) {
        console.log("我是按钮的2级DOM事件");
    },false);//事件由内向外
</script>
```





现在我们将最后一个参数换成`true`

```
box.addEventListener("click",function(event){
    console.log("我是盒子的2级DOM事件");
},true);

btn1.addEventListener("click", function (event) {
    console.log("我是按钮的2级DOM事件");
},true);
```



2级事件的事件链

在0级DOM事件里面，因为事件是以属性的形式存在，所以不可能多次赋值，也就不可以多次绑定

```html
<body>
    <button type="button" id="btn1" onclick="aaa()">按钮</button>
</body>
<script>
    function aaa() {
        console.log("我是aaa");
    }

    var btn1 = document.querySelector("#btn1");
    btn1.onclick = function (event) {
        console.log("我是bbb");
    }
</script>
```

如果在0级事件里面多次赋值，必然是后面覆盖前面，上面的代码在触发事件以后打印的结果是`"我是bbb"`

2级事件是以监听的形式存在的，所以它不会在这个问题

```html
<body>
    <button type="button" id="btn1">按钮</button>
</body>
<script>
    var btn1 = document.querySelector("#btn1");
    btn1.addEventListener("click",function(event){
        console.log("我是aaa");
    })

    btn1.addEventListener("click",function(event){
        console.log("我是bbb");
    })
</script>
```

在使用2级事件的时候，它可以同是绑定多个，同时会根据你绑定的顺序来进行事件队列的管理【谁先绑定就谁先入队】。事件像队列一样的去执行，我们就把它看成是事件链

```
<body>
    <button type="button" id="btn1">按钮</button>
</body>
<script>
    var btn1 = document.querySelector("#btn1");

    btn1.addEventListener("click",function(event){
        console.log("我是bbb");
    })
    btn1.onclick=function(){
        console.log("我是0级事件");
    }
    btn1.addEventListener("click",function(event){
        console.log("我是aaa");
    })
</script>
```

在上面的代码里面，我们可以得到2个点

1. 事件的执行顺序与绑定顺序有关，谁先绑定，谁就在事件队列的前面先执行
2. 0级事件与2级事件可以同时存在

断开事件链

在上个章节我们讲到了，2级事件是可以多次监听的，触发这个事件以后，这个事件绑定的方法就会依次向链条一样执行（事件队列）。但是我们可以在某一个地方把这个事件链断开

```
<body>
    <button type="button" id="btn1">按钮</button>
</body>
<script>
    var btn1 = document.querySelector("#btn1");

    btn1.addEventListener("click",function(event){
        console.log(event);
        console.log("我是bbb");
    })
    btn1.onclick=function(event){
        console.log("我是0级事件");
        event.stopImmediatePropagation();    //断开事件链 ， 并阻止事件传播
    }
```

```
15      btn1.addEventListener("click",function(event){
16          console.log("我是aaa");
17      })
18  </script>
```

说明：

在上面的代码里面，我们对btn1进行了3次事件绑定，但是在第2次绑定事件的时候我们调用了`event.stopImmediatePropagation()`来断开事件链，这个事件在传递到这里的时候就断开了（后面的事件绑定就不再执行了）。同时这里还要注意，它会阻止事件的冒泡与捕获