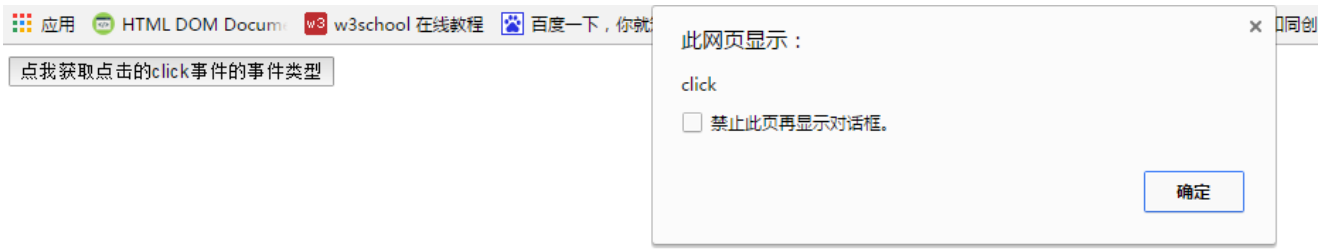
Day 12---事件对象

一、事件对象event(109)

在触发 DOM 上的某个事件时，会产生一个事件对象 event ，这个对象中包含着所有与事件有关的信息。包括导致事件的元素、事件的类型以及其他与特定事件相关的信息。例如，鼠标操作导致的事件对象中，会包含鼠标位置的信息，而键盘操作导致的事件对象中，会包含与按下的键有关的信息。所有浏览器都支持 event 对象，但支持方式不同。

1. 无论哪种事件处理程序，都会有一个event的参数，包含着事件的基本信息。

```
<body>
    <button id="btn">点我获取点击的click事件的事件类型</button>
    <script type="text/javascript">
        var btn = document.getElementById("btn");
        //当点击事件发生的时候，浏览器会把发生的这次点击封装成一个事件对象，通过时间处理程序的参数传进来。
        //在处理程序内部就可以使用这个事件对象了。
        btn.onclick = function (event) {
            alert(event.type);  // click
        }
    </script>
</body
```



2. 事件类型不一样，event可用的属性与方法也不一样。不过不管哪种类型的事件，都具有下面的属性和方法。

属性 / 方法	类型	读/写	说明
bubbles	Boolean	只读	表明事件是否冒泡
cancelable	Boolean	只读	表明是否可以取消事件的默认行为
currentTarget	Element	只读	其事件处理程序当前正在处理事件的那个元素
defaultPrevented	Boolean	只读	为 true 表示已经调用了 preventDefault(DOM3级事件中新增)
detail	Integer	只读	与事件相关的细节信息
eventPhase	Integer	只读	调用事件处理程序的阶段：1表示捕获阶段，2表示“处于目标”，3表示冒泡阶段
preventDefault()	Function	只读	取消事件的默认行为。如果 cancelable 是true，则可以使用这个方法
stopImmediatePropagation()	Function	只读	取消事件的进一步捕获或冒泡，同时阻止任何事件处理程序被调用（DOM3级事件中新增
target	Element	只读	事件的目标
trusted	Boolean	只读	为 true 表示事件是浏览器生成的。为 false 表示事件是由开发人员通过 JavaScript 创建的（DOM3级事件中新增）
type	String	只读	被触发的事件的类型
view	AbstractView	只读	与事件关联的抽象视图。等同于发生事件的 window 对象

二、DOM2 级事件处理(110)

“DOM2级事件”定义了两个方法，用于处理指定和删除事件处理程序的操作：addEventListener()和removeEventListener() 。

2.1 addEventListener()和removeEventListener()

> 所有 DOM 节点中都包含这两个方法，并且它们都接受 3 个参数：
>
> 1. 要处理的事件名(例如：click、focus等。 如意这里写事件名字的时候不能带on)

> 2. 作为事件处理程序的函数
> 3. 一个布尔值。最后这个布尔值参数如果是 true ， 表示在捕获阶段调用事件处理程序；如果是 false ， 表示在冒泡阶段调用事件处理程序。

```
<body>
    <button id="btn">点我或双击我</button>
    <br/></br/>
    <button id="cacelBtn1">点我取消单击事件</button>
     <br/></br/>
    <button id="cacelBtn2">点我取消双击事件</button>
    <script type="text/javascript">
        var btn = document.getElementById("btn");
        //定义事件处理程序。
        function handler (event) {
            if(event.type == "click"){ // 根据事件的类型不同，进行不同的处理
                console.log("你单击了我");
            }else if(event.type = "dbclick"){
                console.log("你双击我了我");
            }
        }
        //给click事件注册监听器
        btn.addEventListener("click", handler, false);
        //给dblclick事件注册监听器
        btn.addEventListener("dblclick", handler, false);

        var cancelBtn1 = document.getElementById("cacelBtn1");
        var cancelBtn2 = document.getElementById("cacelBtn2");

        //点击这个按钮把第一个按钮的单击事件移除
        cancelBtn1.onclick = function () {
            btn.removeEventListener("click", handler, false);
        }
        //点击这个按钮把第一个按钮的双击事件移除
        cancelBtn2.onclick = function () {
            btn.removeEventListener("dblclick", handler, false);
        }
    </script>
</body>
```

2.2 一些注意点

> 可以给一个元素多次添加同一个事件的多个处理程序，那么浏览器会按照添加的顺序顺序执行。

```
<body>
    <button id="btn">点我或双击我</button>
    <script type="text/javascript">
        var btn = document.getElementById("btn");
        //使用匿名函数给click事件添加第一个处理程序
        btn.addEventListener("click", function () { //先注册先执行
            alert("第一次hell");
        }, false);
        //使用匿名函数给click事件添加第二个处理程序
        btn.addEventListener("click", function () { //后注册后执行
            alert("第二次hello");
        }, false);
        //这两个事件处理程序都会执行，顺序为添加(注册)顺序:
    </script>
</body>
```

> 移除监听器(处理程序)的时候，必须和注册时使用的是同一个函数。否则移除失败
>
> 比如：如果注册和移除都是使用的匿名函数，那么一定会移除不起作用

```
<body>
    <button id="btn">点我或双击我</button>
    <script type="text/javascript">
        var btn = document.getElementById("btn");
        //使用匿名函数添加事件处理程序
        btn.addEventListener("click", function () {
            alert("欢迎来到育知同创");
        }, false);
        // 移除使用的处理程序虽然和注册的处理程序代码一样，但是因为两次都是用的匿名函数，所以
        // 两次用的肯定不是同一个函数。所以移除不起作用。
        btn.removeEventListener("click", function () {
            alert("欢迎来到育知同创");
        }, false);
    </script>

</body>
```

注意：

1. 大多数情况下，都是将事件处理程序添加到事件流的冒泡阶段，这样可以最大限度地兼容各种浏览器。最好只在需要在事件到达目标之前截获它的时候将事件处理程序添加到捕获阶段。如果不是特别需要，不建议在事件捕获阶段注册事件处理程序。
2. IE9、Firefox、Safari、Chrome和 Opera 支持 DOM2 级事件处理程序。(ie9以前不支持)

三、event对象的高级属性

3.1 offsetX和offsetY(114)

> 光标相对于触发元素边界的X、Y坐标
>
> 其实是光标相对于触发元素的左上角的坐标。(把左上角的位置看做0，0)

```
<body>
    <div></div>
    <script type="text/javascript">
        var div = document.getElementsByTagName("div")[0];
        div.addEventListener("mousemove", function (event) {
            console.log('offsetX:' + event.offsetX + ", offsetY:" + event.offsetY);
        }, false);
    </script>
</body>
```

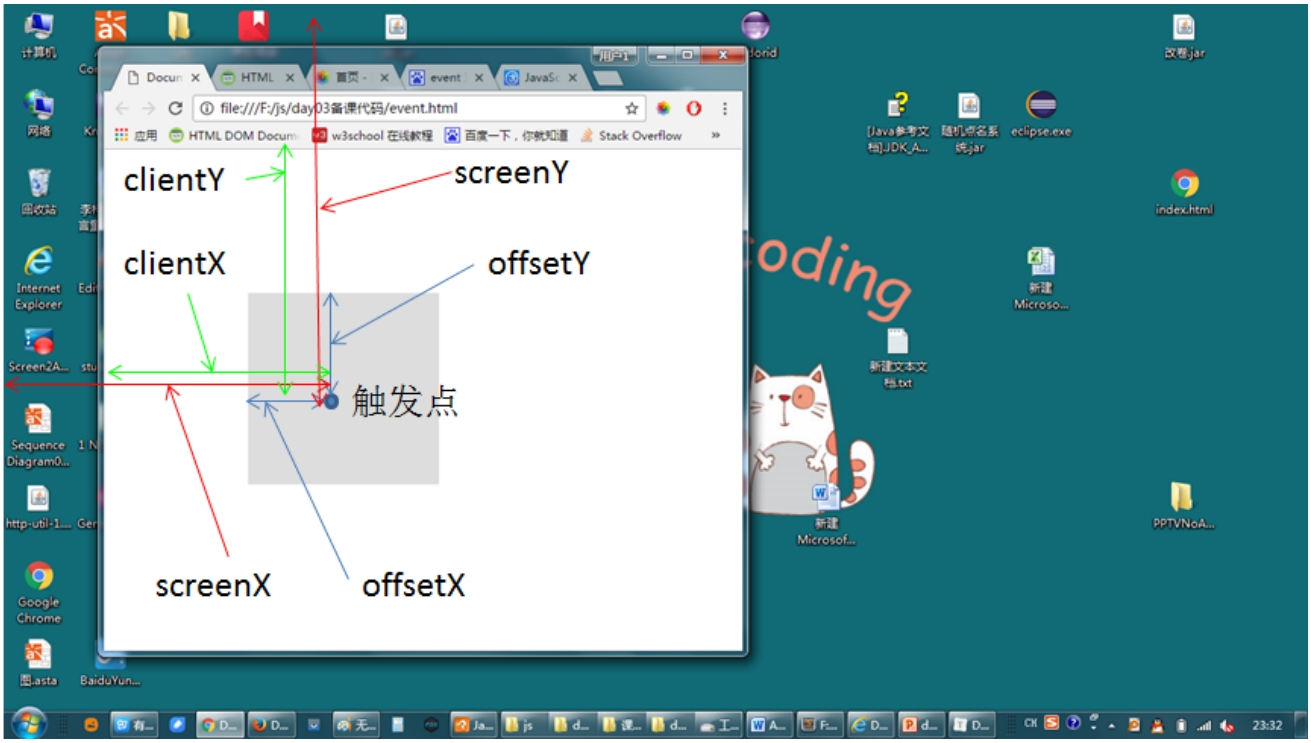
3.2 screenX和screenY(116)

> 当前光标相对于屏幕边缘的x、y坐标
>
> 其实就是相对于屏幕左上角的坐标。(可以看出是绝对坐标)

```
<body>
    <div></div>
    <script type="text/javascript">
        var div = document.getElementsByTagName("div")[0];
        document.documentElement.addEventListener("mousemove", function (event) {
            // console.log('offsetX:' + event.offsetX + ", offsetY:" + event.offsetY);
            console.log("scrrenX:" + event.screenX + ", scrrenY:" + event.screenY);
        }, false);
    </script>
</body>
```

3.3 clientX和clientY(115)

> 当前关闭相对于浏览器浏览器窗口客户区域左上角的坐标(客户区域不包括状态栏、菜单栏等。)
>
> 三种坐标的图示如下：



3.4 stopPropagation()(118)

> 如果bubbles是true(允许冒泡)，则stopPropagation可以阻止事件进一步捕获或冒泡

- 正常情况当把鼠标放在外部div和内部div重合的区域的时候，连个div都可以收到事件。当在内部div的处理程序添加stopPropagation()方法后,则鼠标放在重合区域的上方是外部div就收不到冒泡事件了

```
<body>
    <div id="outer">
         <div id="inner"></div>
    </div>
    <script type="text/javascript">
        var outer = document.getElementById("outer");
        var inner = document.getElementById("inner");
        outer.addEventListener("mouseover", function (event) {
            console.log("外部的div....")
        }, false);
        inner.addEventListener("mouseover", function (event) {
            console.log("内部的div");
           event.stopPropagation();
        }, false);
    </script>
</body>
```

3.4 preventDefault()(119)

> 取消事件的默认行为：

```
<body>
    <form  id="form" action="">
        <input type="text" name="user"><br/><br/>
        <input type="submit" value="提交">
        <p>本演示可以阻止提交按钮默认提交表单的动作</p>
    </form>
    <script type="text/javascript">
        var form = document.getElementById("form");
        form.addEventListener("submit", function (event) {
            //阻止表单的默认提交事件
            event.preventDefault();
        }, false);
    </script>
</body>
```

四、事件代理(事件委托)(113)

当一个页面中需要给比较多的元素添加时间处理程序的时候，以前的做法是通过找到每个元素，然后逐一的去添加。这样做有一些缺点：

- 多次去dom树中查找元素，比较耗费时间。
- 太多事件处理程序，比较占资源。特别个别浏览器(如ie)对垃圾回收这块又做的不够好。
- 程序中过多的事件处理，代码后期不好管理。

这时我们可以用一种叫做事件委托的方式去解决。

> 事件委托的原理：

假设现在要处理多个具有并列关系元素的click事件，当我点击这些元素中的任何一个元素，则事件一定会通过冒泡的方式，冒泡到他的上层的父节点元素然后一直冒到window，所以这个时候我们就可以在他的上层元素中添加事件处理程序，来统一处理这些事件，在处理的过程中可以通过获取target的id来知道是点击的哪个具体的元素。这种方式就称之为事件委托。

```
<body>
    <ul id="myGirls">
        <li id="zhiling">志玲</li>
        <li id="baizhi">柏芝</li>
        <li id="fengjie">凤姐</li>
        <li id="yifei">亦非</li>
    </ul>
    <script type="text/javascript">
        var myGirls = document.getElementById("myGirls");
        myGirls.addEventListener("click", function(event) {
            var tagetId = event.target.id;
            // alert(tagetId);
            switch (tagetId) {
                case "zhiling":
                    alert("我是志玲，哥哥你要干吗");
                    break;
                case "baizhi":
                    alert("我是柏芝，我以后再也不找冠希了");
                    break;
                case "fengjie":
                    alert("我是凤姐，我已经整容了，没有以前那么丑了");
                    break;
                case "yifei":
                    alert("我是亦非，我干爹是富豪");
                    break;
                default:
                    alert("你一个都不点是啥意思");
                    break;
            }
        }, false);
    </script>
</body>
```

> 说明：

1. 完全可以考虑给document添加一个事件处理程序，用来处理页面上发生的某种特定类型的事件。
2. 比较适合事件委托的事件：click 、 mousedown 、 mouseup 、 keydown 、 keyup 和 keypress。