事件对象

事件对象是对当前触发的这个事件的详细信息的描述，事件对象叫`Event`

获取事件对象

之前我们学习了事件的三种绑定方式，现在我们从这三种绑定方式里面来获取事件对象

第一种

```
<button type="button" onclick="console.log(event)">按钮
</button>
```

第一种方式直接在行内代码获取，然后系统会自动向这个事件方法内注入一个事件对象`event`

第二种

```
<button type="button" onclick="aaa(event)">按钮2</button>
<script>
  function aaa(event) {
    // IE兼容性写法
    event = event || window.event
    console.log(event);
  }
</script>
```

第二种写法，这里需要处理IE兼容性

第三种

```
<button type="button" class="btn1">按钮1</button>
<script>
  var btn1 = document.querySelector(".btn1")
  btn1.onclick = function (event) {
    // 这里也需要处理IE兼容性
    event = event || window.event
    console.log(event);
  }
</script>
```

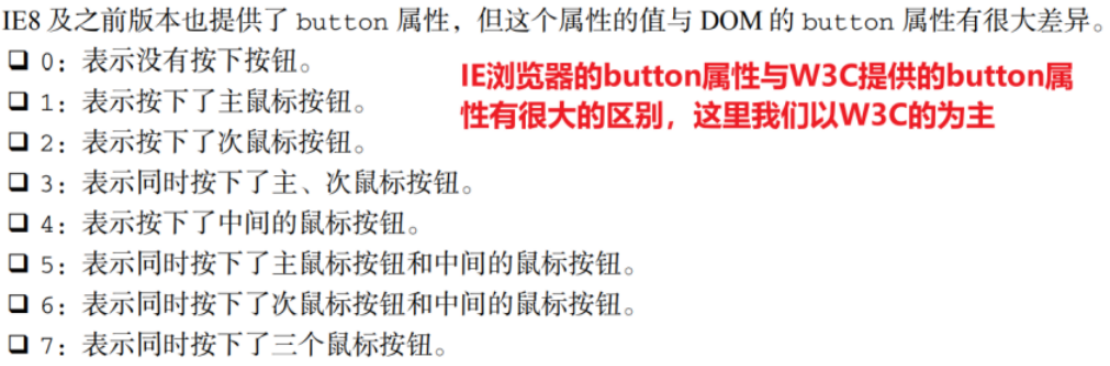
在这一种方式里面，系统会自动的向事件内部注入一个参数`event`，但是我们仍然需要处理兼容性

分析事件对象

鼠标事件对象

1. `altKey`触发当前事件的时候是否按下了`alt`键
2. `ctrlKey`触发当前事件的时候是否按下了`ctrl`键
3. `shiftKey`触发当前事件的时候是否按下了`shift`键
4. `button`触发当前事件的按键是哪一个，`0`表示鼠标主键（左键），`1`表示鼠标中键，`2`表示鼠标附键（右键）

   下图是`buttons`的按键

5. `clientX/clientY`代表鼠标事件触发的时候距离游览器左边或者上面的坐标位置
6. `screenX/screenY`代表鼠标事件触发的时候距离屏幕左边或者上面的坐标位置
7. `x/y`这个就是上面的`clientX/clientY`。出现他的原因也是因为浏览器的兼容性

```
// 如果想要获取当前的触发事件的位置距离游览器左边或者上面的距离
var x = event.x || event.clinetX;
var y = event.y || event.clinetY;
```

8. `offsetX/offsetY` 代表鼠标事件触发的位置距离**事件触发者**的左边或者上面的坐标位置
9. `pageX/pageY` 代表鼠标事件触发的位置距离页面的左边或者上面的坐标位置
10. `type` 代表当前触发的事件类型
11. `path` 代表当前事件的冒泡路径（也叫事件传递路径）
12. `bubbles` 代表当前事件是否冒泡
13. `cancelBubble` 代表当前事件是否取消了冒泡
14. `cancelable` 代表当前的事件是否可以取消冒泡
15. `target` 代表当前事件的触发者
16. `currentTarget` 代表事件的绑定者

键盘事件

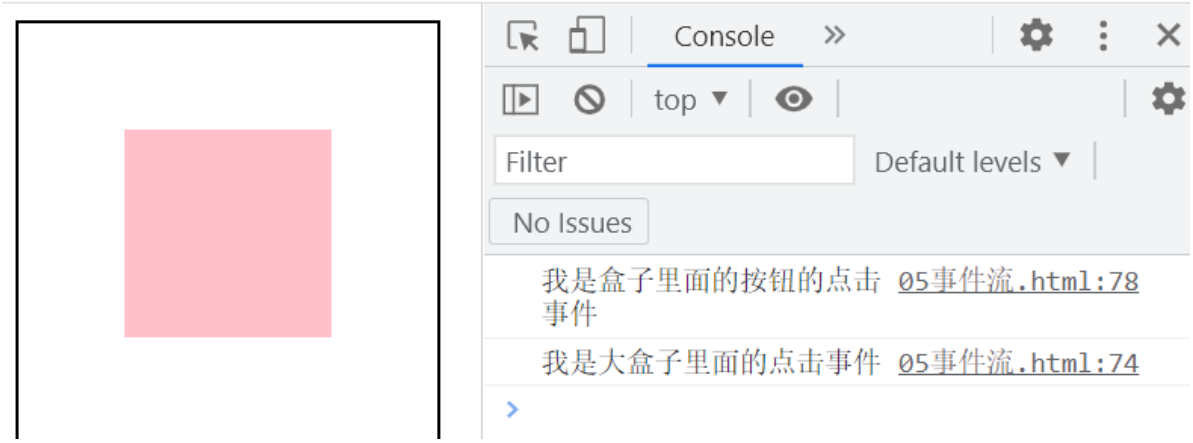
1. `keyCode` 代表当前按下的这个按键的编码
2. `key` 代表当前输入的这个字符
3. `repeat` 表示当前事件的触发是否因为之前的按键没有松开而连续触发，`true` 表示没有松开连续的触发

事件流

事件流的通俗说法叫做事件传播行为，关与事件的传播请看下面的例子

```
<body>
  <div class="big-box" onclick="aaa(event)">
    <div class="box" onclick="bbb(event)"></div>
  </div>
</body>
<script>
  function aaa(event) {
    console.log("我是大盒子里面的点击事件");
```

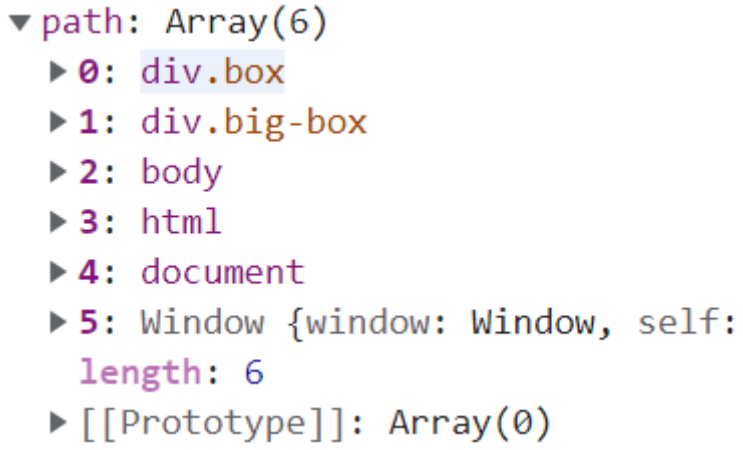
```
    }

    function bbb(event) {
      console.log("我是盒子里面的按钮的点击事件");
    }
</script>
```



当我们去点击红色的小盒子的时候，外面的盒子的事件同时也触发了，这个时候小盒子里的`onclick`事件就传播到了大盒子里面的`onclick`，这种现象叫做事件传播，也叫事件冒泡

事件冒泡

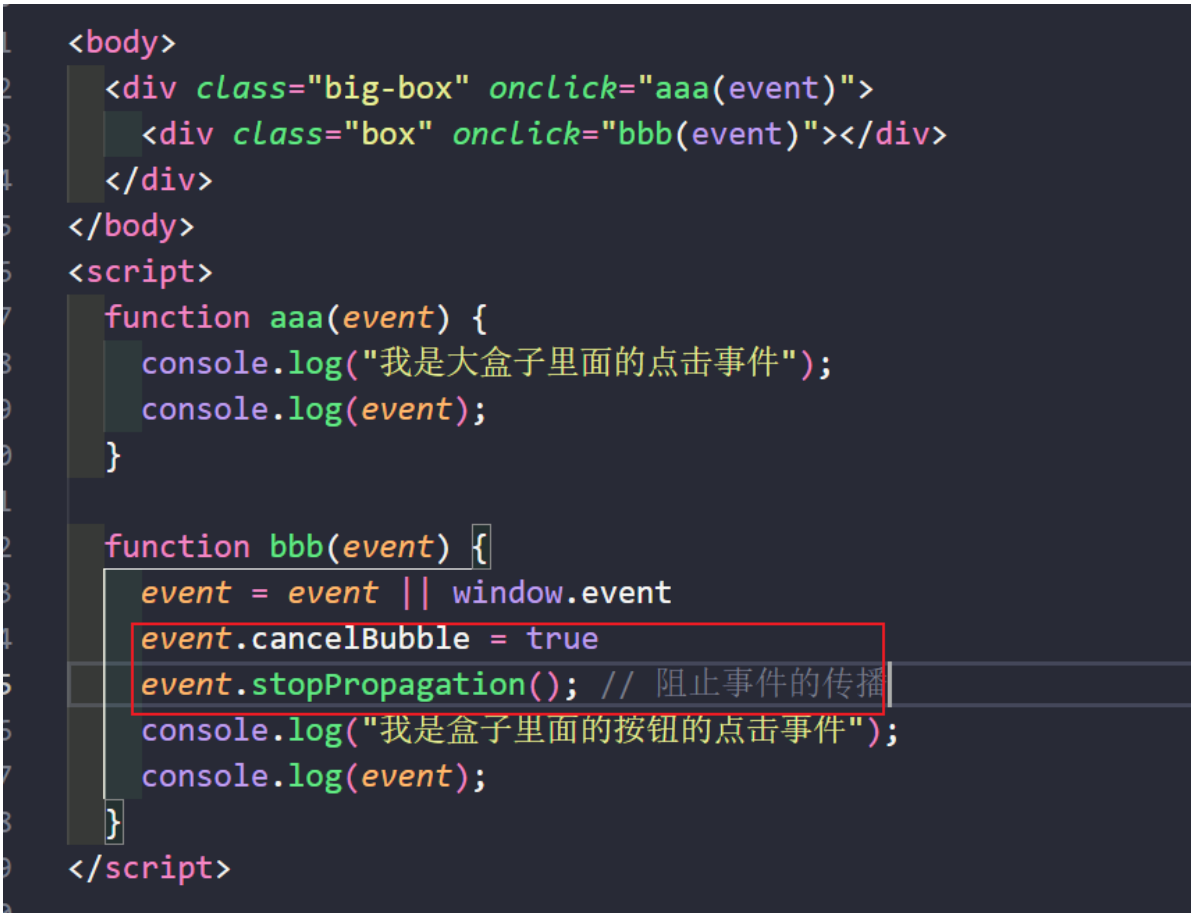
事件冒泡是指的是同一类型的事件会在元素的内部由内向外传播，这种现象叫做事件冒泡，上面的例子也就是一个事件冒泡的典型例子，同时我们也可以在事件对象里面看到事件冒泡的路径



取消事件冒泡

虽然事件默认是会存在传播行为的，但是我们在一些特定的场景下面我们需要取消事件的冒泡行为

```
<body>
  <div class="big-box" onclick="aaa(event)">
    <div class="box" onclick="bbb(event)"></div>
  </div>
</body>
<script>
  function aaa(event) {
    console.log("我是大盒子里面的点击事件");
    console.log(event);
  }

  function bbb(event) {
    event = event || window.event
    event.cancelBubble = true
    event.stopPropagation(); // 阻止事件的传播
    console.log("我是盒子里面的按钮的点击事件");
    console.log(event);
  }
</script>
```



阻止事件冒泡主要就是设置这两个点

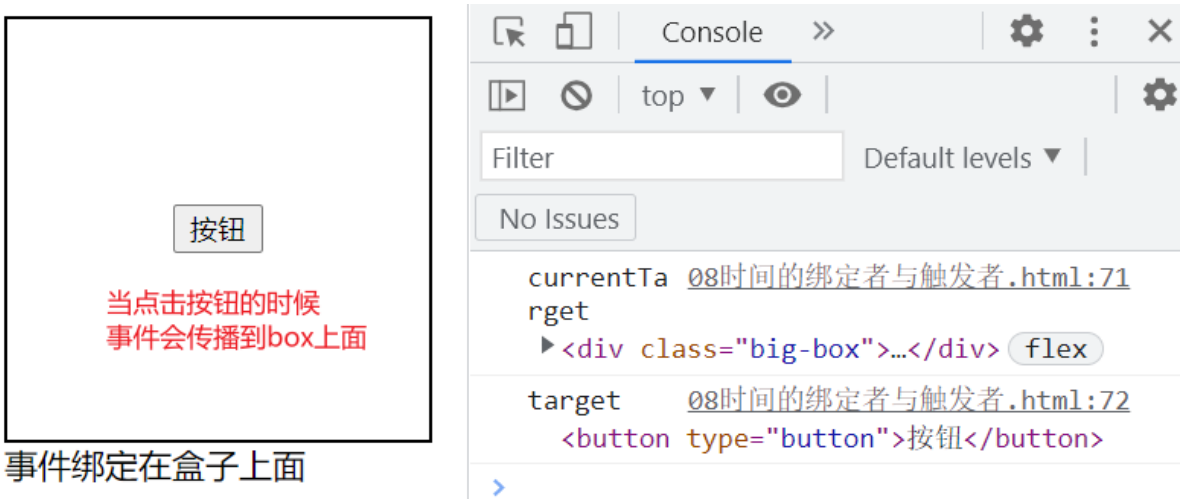
1. `event.cancelBubble = true` 取消事件事件冒泡

2. `event.stopPropagation()`阻止事件的传播行为

事件的绑定者与触发者【重点】

因为事件是有传播行为的，所以在事件的对象里面就可以看出当前这个事件是被哪一个元素触发的

```
<body>
  <div class="big-box">
    <button type="button">按钮</button>
  </div>
  事件绑定在盒子上面
</body>
<script>
  var box = document.querySelector(".big-box")
  box.onclick = function (event) {
    console.log("currentTarget", event.currentTarget); // 代表事件的绑定者
    console.log("target", event.target); // 表示事件的触发者
  }
</script>
```



当我们去点击里面的按钮的时候，事件是有按钮触发，但是事件有冒泡到了外面的盒子上面

上面的`target`表示当前事件的触发者`currentTarget`表示当前事件的绑定者

事件委托

因为事件是一个传播行为，所以我们可以认为内部的事件总会冒泡到外部，这样我们就可以事件一个特殊的场景，叫事件委托

请看下面代码

```
<div class="box">
    <button type="button" class="btn1">按钮1</button>
    <button type="button" class="btn1">按钮2</button>
    <button type="button" class="btn1">按钮3</button>
    <button type="button" class="btn1">按钮4</button>
    <h2>这是一个二号标题</h2>
    <p>这是一个段落标题</p>
</div>
<button class="add" onclick="add()">新增</button>
```

要求：将box里面的所有button都绑定一个点击事件，点击后打印当前按钮的文字

第一种方法：使用批量绑定

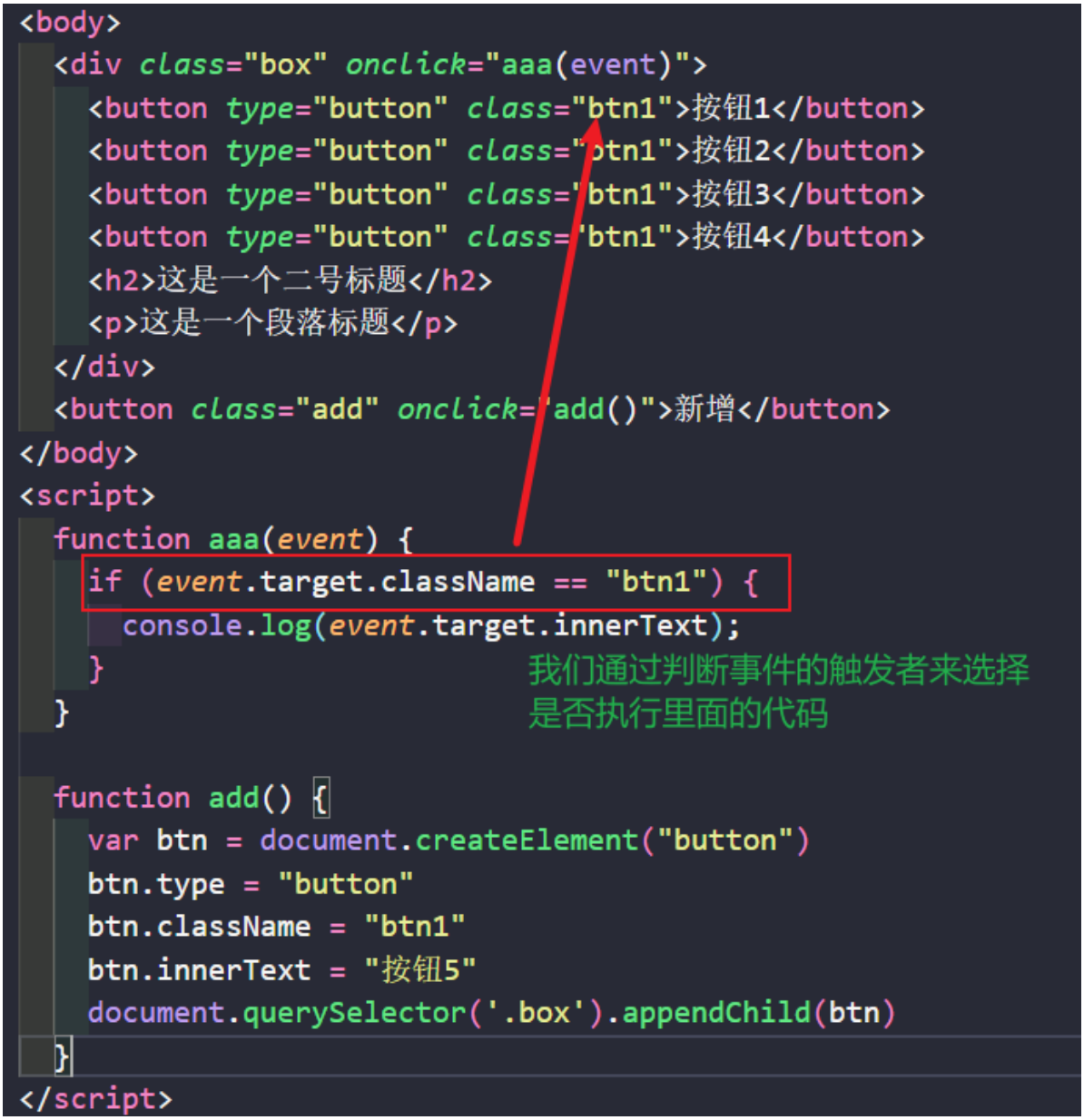
```
var btnList = document.querySelectorAll(".btn1")
btnList.forEach(function (item) {
    item.onclick = function () {
        console.log("我是" + item.innerText);
    }
})

function add() {
    var btn = document.createElement("button")
    btn.type = "button"
    btn.innerText = "按钮5"
    document.querySelector('.box').appendChild(btn)
}
```

1. 事件的绑定非常麻烦,需要去找到所有`"class='btn1'"`的元素然后遍历,然后绑定事件,这么做会非常消耗我们浏览器的性能
2. 当我们调用`add()`方法新添加一个按钮后,我们发现新添加的这个按钮没有我们前面绑定的那个事件

第二种方法：利用事件的传播行为来实现

```
<body>
  <div class="box" onclick="aaa(event)">
    <button type="button" class="btn1">按钮1</button>
    <button type="button" class="btn1">按钮2</button>
    <button type="button" class="btn1">按钮3</button>
    <button type="button" class="btn1">按钮4</button>
    <h2>这是一个二号标题</h2>
    <p>这是一个段落标题</p>
  </div>
  <button class="add" onclick="add()">新增</button>
</body>
<script>
  function aaa(event) {
    if (event.target.className == "btn1") {
      console.log(event.target.innerText);
    }
  }

  function add() {
    var btn = document.createElement("button")
    btn.type = "button"
    btn.className = "btn1"
    btn.innerText = "按钮5"
    document.querySelector('.box').appendChild(btn)
  }
</script>
```

```
<body>
  <div class="box" onclick="aaa(event)">
    <button type="button" class="btn1">按钮1</button>
    <button type="button" class="btn1">按钮2</button>
    <button type="button" class="btn1">按钮3</button>
    <button type="button" class="btn1">按钮4</button>
    <h2>这是一个二号标题</h2>
    <p>这是一个段落标题</p>
  </div>
  <button class="add" onclick="add()">新增</button>
</body>
<script>
  function aaa(event) {
    if (event.target.className == "btn1") {
      console.log(event.target.innerText);
    }
  }

  function add() {
    var btn = document.createElement("button")
    btn.type = "button"
    btn.className = "btn1"
    btn.innerText = "按钮5"
    document.querySelector('.box').appendChild(btn)
  }
</script>
```



通过上面两组方式的对比，我们发下半年在第二种方式里面，本来应该绑定在子集元素`button`上面的事件，我们把它绑定在父级的box上面，这种现象就叫**事件委托**

判断事件的触发者

在上面的事件委托里面，我们看到事件绑定在父级，然后通过`event.target`来决定是否要执行代码，这个时候就有一些问题，如下：

11.1.3 matchesSelector()方法

Selectors API Level 2 规范为 Element 类型新增了一个方法 matchesSelector()。这个方法接收一个参数，即 CSS 选择符，如果调用元素与该选择符匹配，返回 true；否则，返回 false。看例子。

```
if (document.body.matchesSelector("body.page1")){
    //true
}
```

注意在现代化标准浏览器里面
已经将这个方法统一的
调整为matches()方法

在取得某个元素引用的情况下，使用这个方法能够方便地检测它是否会被 querySelector()或 querySelectorAll()方法返回。

截至 2011 年年中，还没有浏览器支持 matchesSelector()方法；不过，也有一些实验性的实现。IE 9+通过 msMatchesSelector()支持该方法，Firefox 3.6+通过 mozMatchesSelector()支持该方法，Safari 5+和 Chrome 通过 webkitMatchesSelector()支持该方法。因此，如果你想使用这个方法，最好是编写一个包装函数。

```
<body>
  <ul class="box" onclick="aaa(event)">
    <li>第1项</li>
    <li>第2项</li>
    <li>第3项</li>
    <li>第4项</li>
    <li>第5项</li>
    <li>第6项</li>
  </ul>
  <button type="button" onclick="add()">新增</button>
</body>
<script>
  function aaa(event) {
    if (event.target.matches(".box>li:nth-child(odd)")) {
      console.log(event.target.innerText);
    }
  }

  function add() {
    var li = document.createElement("li")
    li.innerText = "第7项"
    document.querySelector(".box").appendChild(li)
  }
</script>
```

通过上面的这个`maches()`方法，我们就可以快速的判断某一个Element的元素是否符号我们想要执行事件代码的要求，这样去判断事件委托的触发者就非常方便了

事件方法里的this

在事件方法的内部`this`永远执行事件的绑定者`currentTarget`

在事件处理程序内部，对象 this 始终等于 currentTarget 的值，而 target 则只包含事件的实际目标。如果直接将事件处理程序指定给了目标元素，则 this、currentTarget 和 target 包含相同的值。来看下面的例子。

```
var btn = document.getElementById("myBtn");
btn.onclick = function(event){
    alert(event.currentTarget === this);    //true
    alert(event.target === this);           //true
};
```

事件方法中的this指向的是事件的绑定者

DOMEventObjectExample01.htm

这个例子检测了 currentTarget 和 target 与 this 的值。由于 click 事件的目标是按钮，因此这三个值是相等的。如果事件处理程序存在于按钮的父节点中（例如 document.body），那么这些值是不相同的。再看下面的例子。

事件先与响应

看下面的代码

第一个例子

```
<body>
  <a href="www.baidu.com" target="_self" id="bd">百度一下</a>
</body>
<script>
  var bd = document.querySelector("#bd")
  bd.onclick = function () {
    alert("我是a标签的点击事件")
  }
</script>
```

在上面的代码里面，我们对a标签，同时指定了`href`的链接属性吗，以及绑定了`onclick`的点击事件，这个时候如果我们点击这个链接，根据**事件先于响应**的原则，它应该会先触发`onclick`，再去执行`href`的链接跳转

第二个例子

现在我们在以表单标签为例子来看一下

```
<body>
  <form id="form1">
    <input type="text" name="userName">
    <button type="reset">重置</button>
  </form>
</body>
<script>
  var form1 = document.querySelector("#form1")
  form1.onreset = function () {
    alert("我是重置表单事件")
  }
</script>
```

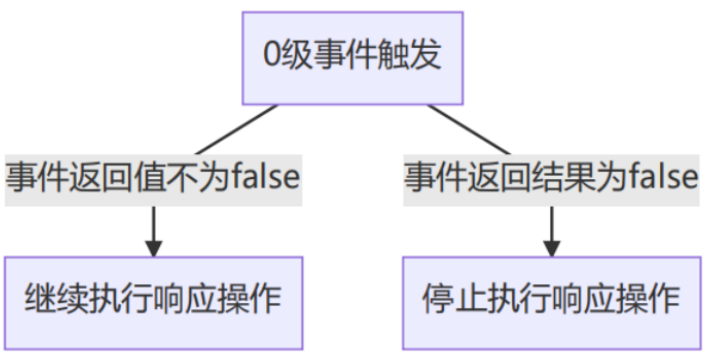
通过`reset`的例子我们也看到了，`onreset`的事件先触发，然后再去执行`reset`的重置操作

第三个例子

右键菜单事件

```
<body>
  <div class="box"></div>
</body>
<script>
  var box = document.querySelector(".box")
  box.oncontextmenu = function () {
    alert("右键菜单事件")
  }
</script>
```

事件先于响应这是原则，但是也不是所有的事件最终都会触发响应，它是有条件的



如果我们要是在刚刚的事件后面添加一个`return false`，则默认的响应行为就不会触发了，这种现象在0级事件里面叫**事件先与响应**，在2级事件里面叫**取消事件的默认行为**


```
<body>
  <a href="./10事件委托.html" id="aaa">跳转链接</a>

  <a href="./09事件委托.html" onclick="return bbb()">跳转链接2</
  a>
</body>
<script>
  var aaa = document.querySelector("#aaa")
  aaa.onclick = function () {
    alert("点击事件触发")
    // 事件触发后不一定所有的响应都会执行只有当事件的返回值不为false
    的时候才会去触发响应
    return false
  }

  function bbb() {
    alert("我是方法bbb")
    return false
  }
```