【前端面试】谈谈你对JS点击事件的理解和...

前言

这是一道比较经典的面试题，主要考察的是面试者原生 JS 的基础怎么样，以及对 JS 的一些理解。很多小伙伴可能大概会想到事件捕获和冒泡等，确实没有题。但是很多小伙伴都是模模糊糊，理解得不够透彻，今天我们就来梳理一下。

1.JS 有哪些事件?

虽然我们这里主要讲的是 JS 点击事件，但是我们还是很有必要了解一下 JS 里面有哪些事件。以便于让我们更好的理解事件在 JS 中扮演的角色以及过程。

JS 事件是用户与网页进行交互时发生的事情，比如用户点击网页，这就是一个点击事件，在网页上按下键盘，这就是键盘事件，还有很多其它的事件...

事件分类：

1.1 鼠标事件

事件名	解释
onclick	点击鼠标时触发此事件
ondblclick	双击鼠标时触发此事件
onmousedown	按下鼠标时触发此事件
onmouseup	鼠标按下后又松开时触发此事件
onmouseover	当鼠标移动到某个元素上方时触发此事件
onmousemove	移动鼠标时触发此事件
onmouseout	当鼠标离开某个元素范围时触发此事件

1.2 键盘事件

事件名	解释
onkeypress	当按下并松开键盘上的某个键时触发此事件
onkeydown	当按下键盘上的某个按键时触发此事件
onkeyup	当放开键盘上的某个按键时触发此事件

1.3 窗口事件

事件名	解释
onabort	图片在下载过程中被用户中断时触发此事件
onbeforeunload	当前页面的内容将要被改变时触发此事件
onerror	出现错误时触发此事件
onload	页面内容加载完成时触发此事件
onmove	当移动浏览器的窗口时触发此事件
onresize	当改变浏览器的窗口大小时触发此事件
onscroll	当滚动浏览器的滚动条时触发此事件
onstop	当按下浏览器的停止按钮或者正在下载的文件被中断时触发此事件
oncontextmenu	当弹出右键上下文菜单时触发此事件
onunload	改变当前页面时触发此事件

1.4 表单事件

事件名	解释
onblur	当前元素失去焦点时触发此事件
onchange	当前元素失去焦点并且元素的内容发生改变时触发此事件
onfocus	当某个元素获得焦点时触发此事件
onreset	当点击表单中的重置按钮时触发此事件
onsubmit	当提交表单时触发此事件

2.点击事件执行过程

用户点击按钮或者点击其它元素时，触发点击事件，这个过程并不是立即执行的，它是需要走一遍既定的流程的。

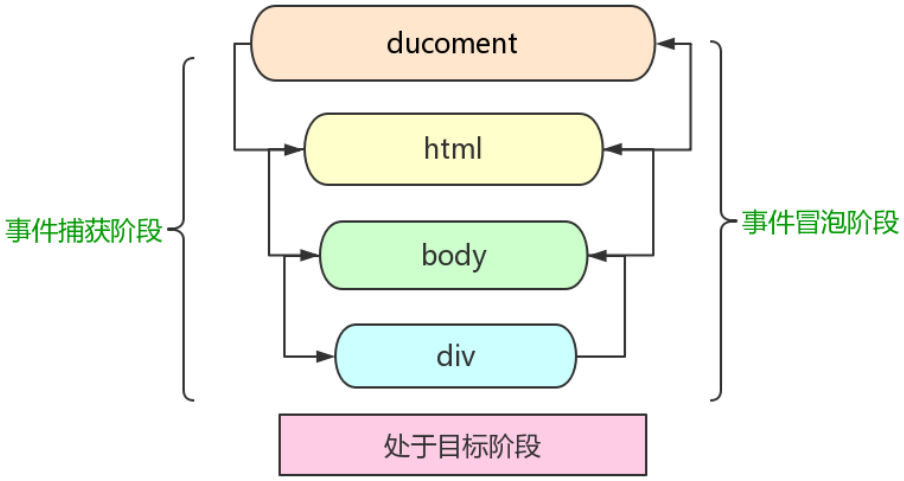
一个点击事件被触发，主要会走以下流程：

1. 用户点击某个按钮，即触发点击事件。
2. 浏览器从顶层 document 元素发出一个**事件流。**
3. 事件流顺着 DOM 逐层向下查找触发事件的目标元素，这就是常说的**事件捕获。**
4. 如果在查找过程中遇到了相同的事件，比如其它元素也绑定点击事件，那么默认不执行，继续往下找。
5. 查找到目标元素后，就会执行目标元素所绑定的事件函数，这也就是常说的**事件目标**阶段。

6. 到这儿整个点击事件还没有完，浏览器会逆向执行该操作，也就是我们所说的**事件冒泡**。
7. 事件冒泡阶段，默认会触发相同的事件，也就是我们刚刚在事件捕获阶段，遇到相同的事件未执行，因为默认在这个冒泡阶段执行。

上面的流程大致就是一个点击事件的执行过程，这中过程也被称作 DOM 的**事件模型**。其实总结下来主要就三步：**事件捕获阶段->事件目标阶段->事件冒泡阶段**。

看图理解：



看图就很好理解了，整个就是一个环形，分别对应了三个阶段。

3.代码演示

3.1 正常事件流

上一节我们说目标元素的点击事件函数是在浏览器找到它的时候执行的，在冒泡阶段会执行相同的事件函数，我们来验证一下。

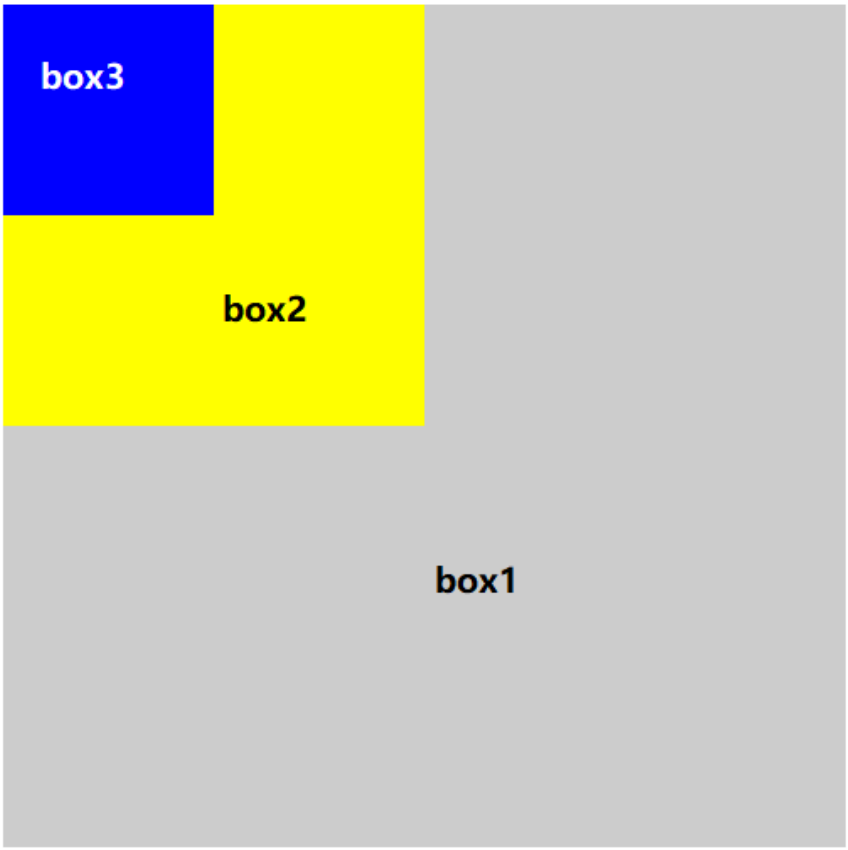
示例代码：

```
<head>
  <style>
    #box1 {
      width: 200px;
      height: 200px;
      background-color: #ccc;
    }

    #box2 {
      width: 100px;
      height: 100px;
      background-color: yellow;
    }

    #box3 {
      width: 50px;
```

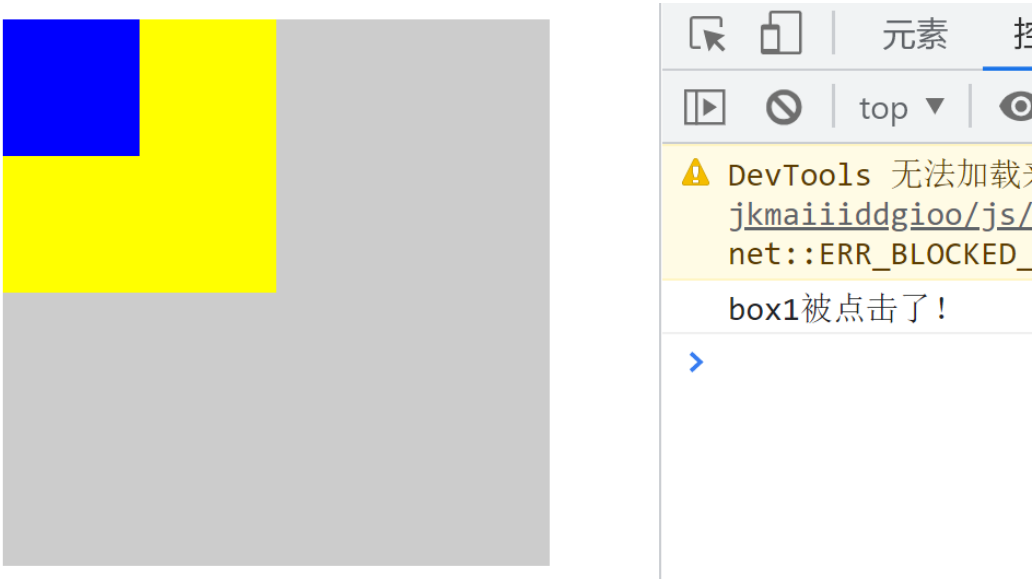
```
      height: 50px;
      background-color: blue;
    }
  </style>
</head>

<body>
  <div id="box1" onclick="box1Click">
    <div id="box2" onclick="box2Click">
      <div id="box3" onclick="box3Click"></div>
    </div>
  </div>
</body>
<script>
  // 获取 3 个元素节点
  let box1 = document.getElementById("box1");
  let box2 = document.getElementById("box2");
  let box3 = document.getElementById("box3");
  // 绑定点击事件
  box1.addEventListener("click", box1Click);
  box2.addEventListener("click", box2Click);
  box3.addEventListener("click", box3Click);
  // 事件处理函数
  function box1Click(){
    console.info("box1 被点击了！")
  }
  function box2Click(){
    console.info("box2 被点击了！")
  }
  function box3Click(){
    console.info("box3 被点击了！")
  }
</script>
```

上段代码中我们简单写了 3 个 div，它们是层级关系。然后分别给每个 div 都加上了一个点击事件，且定义了相应的处理事件函数。

页面展示：

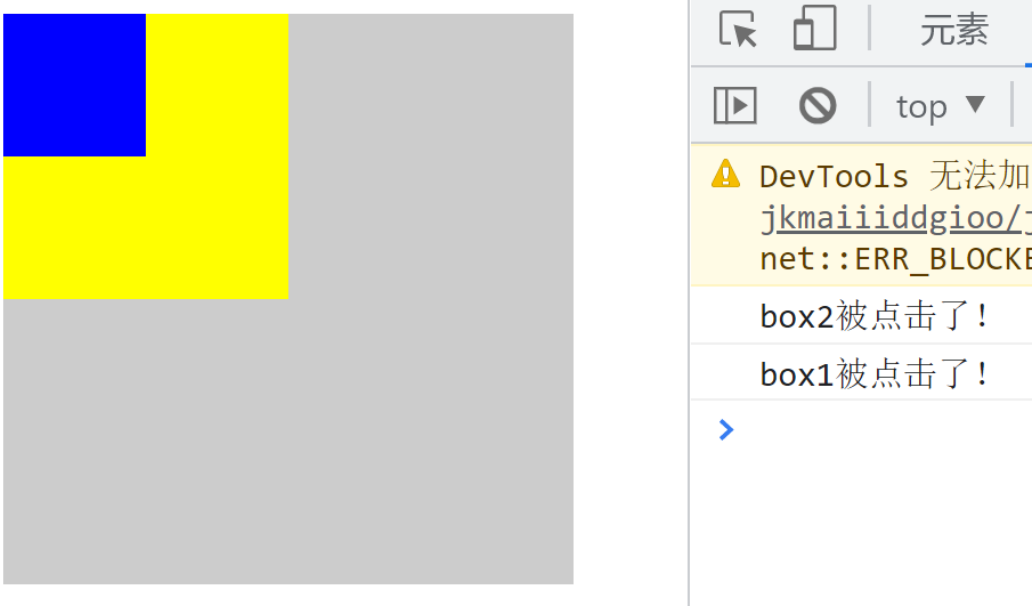


点击 box1，输出结果：



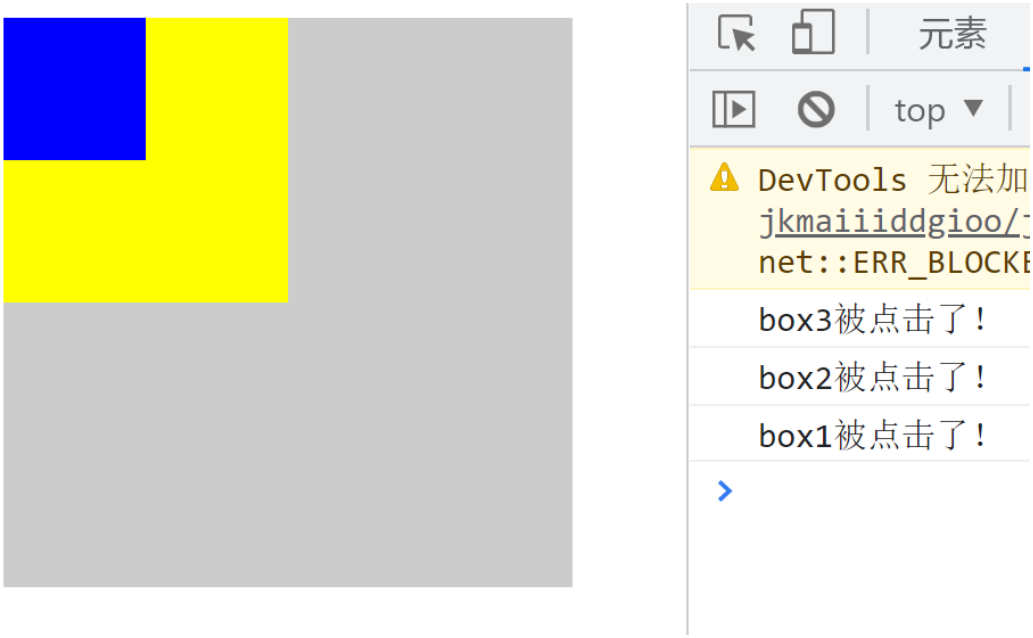
这里只执行了 box1 的点击事件函数，因为事件流操作向下查找到目标元素后，便会向上开始冒泡，box2 和 box3 都不在整个过程中。

点击 box2，输出结果：



点击 box2 时，box1 和 box2 都在事件流这整个过成功，超找到目标元素后，执行了 box2 的事件处理函数，然后向上冒泡，执行了 box1 的事件函数。

点击 box3，输出结果：



点击 box1，这个时候三个元素都处于事件流当中，都会在冒泡阶段执行对应的事件处理函数。

3.2 阻止冒泡

上一节中的正常事件流中，点击 box3，box1 和 box2 的点击事件函数也被执行了，在很多时候这可能不是我们想要的。我们只想点击 box1 的时候只执行 box1 的事件函数，这个时候我们就可以通过阻止冒泡来解决这个问题。

因为我们最开始说：整个事件流过程中，在冒泡阶段会执行在捕获阶段发现的相同的事件函数。所以我们阻止冒泡就可以解决问题。

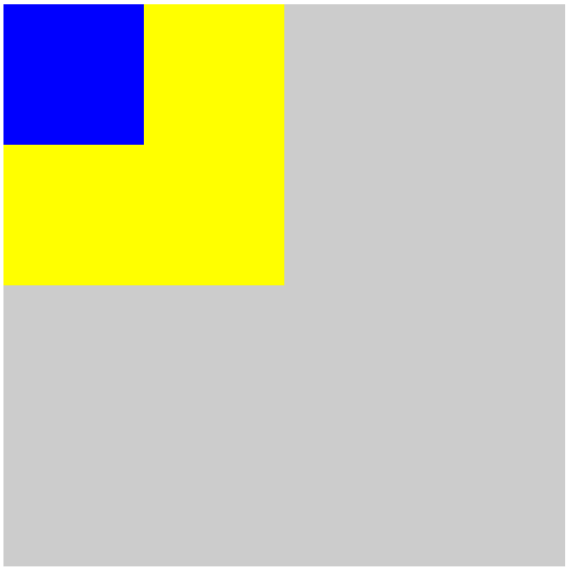
示例代码：

```
function box3Click(e){
```

```
    console.info("e",e);
    e.stopPropagation();
    console.info("box3 被点击了！")
}
```

我们更改 box3de 事件处理函数，函数会默认接收一个事件参数，可以调用 stopPropagation() 阻止冒泡。

点击 box3，输出结果：



这时候点击 box3，只有 box3 的点击事件函数被执行了。

3.3 捕获阶段执行

在未阻止冒泡的前提下，我们点击 box3，执行顺序依次是：**box3->box2->box1。**这是因为所有事件都是在冒泡阶段执行的，那么如果我们想要执行的顺序改为：**box1->box2->box3。**这种情况我们该怎么做呢?

想要实现这种情况，我们得先了解 addEventListener 这个函数。

使用方法：

```
element.addEventListener(event, function, useCapture)
```

可以看到 addEventListener 函数还可以接收第三个参数，前两个参数大家都懂，这里就不用提了。

第三个参数 useCapture 接收一个 Boolean 值，它得作用主要如下：

- true：事件在捕获阶段执行
- false：默认值，事件在冒泡阶段执行

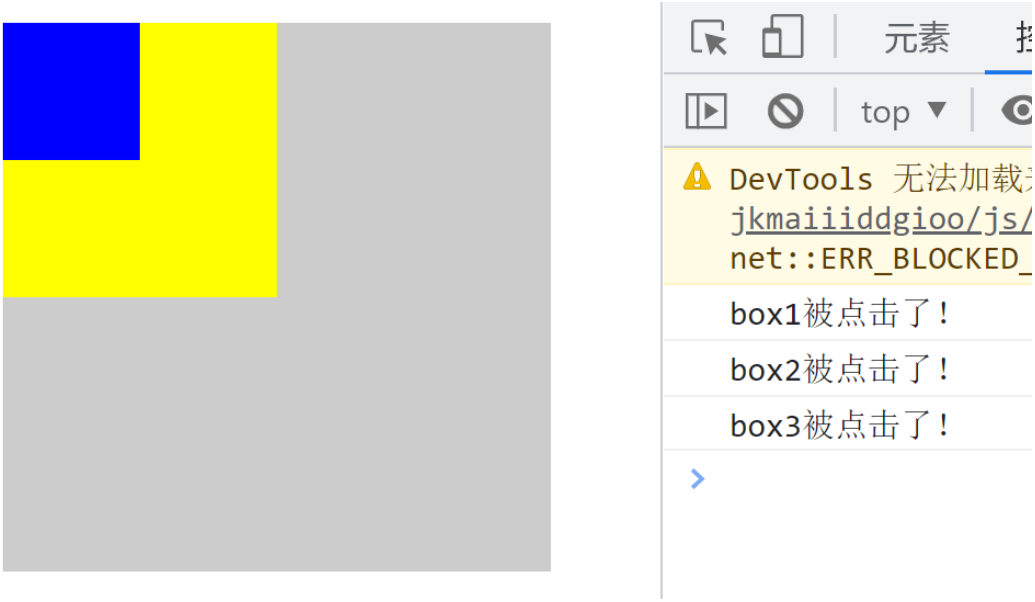
知道了第三个参数的含义以后，要实现我们的需求就简单了，我们只需要事件函数在捕获阶段执行即可。

示例代码：

```
box1.addEventListener("click", box1Click,true);
```

```
box2.addEventListener("click", box2Click,true);
box3.addEventListener("click", box3Click,true);
```

点击 box3，输出结果：



这个时候执行顺序就是我们想要的了。

4.事件委托

事件委托是事件模型中的一个经典应用。

假如有这样一个需求：

有 100 个 li 标签，我们要给每个 li 标签都添加一个点击事件。

初学者常见的一个错误做法就是循环列表，然后分别添加点击事件。这种做法非常消耗内存的，而且也非常繁琐，假如有 100000 个列表呢?

这个时候我们思考一下事件模型，我们是不是可以事件模型中的冒泡来实现这个需求呢？我们将点击事件绑定在 li 标签的父级，当我们点击 li 标签是，在冒泡阶段就会触发 li 标签上一层的点击事件，这也算是变相给 li 标签添加了点击事件。

示例代码：

```
<body>
  <ul id="ul" onclick="ulClick">
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
  </ul>
</body>
<script>
  // 事件委托
  let ul = document.getElementById("ul");
  ul.addEventListener("click", ulClick);
```

```
  function ulClick(e) {
    // 兼容性处理
    let event = e || window.event;
    let target = event.target || event.srcElement;
    // 判断是否匹配目标元素
    if (target.nodeName.toLocaleLowerCase === 'li') {
      console.log('the content is: ', target.innerHTML);
    }
    console.log("li 标签被点击了：", e.target.innerHTML);
  }
</script>
```

我们指在 ul 上添加了点击事件，点击 li 标签时，在冒泡阶段便会执行 ul 上的事件函数。

输出结果：

5.一些兼容性问题

event 兼容性：

- IE 下：event 对象有 srcElement 属性,但是没有 target 属性。
- 火狐或谷歌下：event 对象有 target 属性,但是没有 srcElement 属性。

event || window.event：

- 也是为了兼容 IE

阻止冒泡：

- e.stopPropagation()
- window.event.cancelBubble = true（兼容 IE）

万恶之源 IE。

总结

JS 的事件模型其实不复杂，只要理解三个阶段，一切便可迎刃而解，至于 EventLoop 事件循环，这不是我们这节要讨论的。

三个阶段：事件捕获、事件目标、事件冒泡。