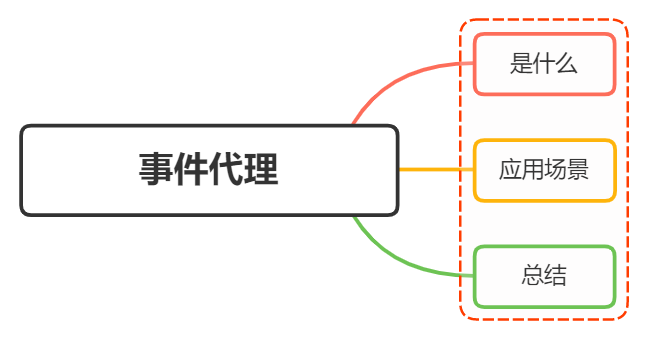
# 面试官：解释下什么是事件代理？应用场景？



## 一、是什么

事件代理，俗地来讲，就是把一个元素响应事件（click、keydown......）的函数委托到另一个元素

前面讲到，事件流的都会经过三个阶段： 捕获阶段 -> 目标阶段 -> 冒泡阶段，而事件委托就是在冒泡阶段完成

事件委托，会把一个或者一组元素的事件委托到它的父层或者更外层元素上，真正绑定事件的是外层元素，而不是目标元素

当事件响应到目标元素上时，会通过事件冒泡机制从而触发它的外层元素的绑定事件上，然后在外层元素上去执行函数

下面举个例子：

比如一个宿舍的同学同时快递到了，一种笨方法就是他们一个个去领取

较优方法就是把这件事情委托给宿舍长，让一个人出去拿好所有快递，然后再根据收件人一一分发给每个同学

在这里，取快递就是一个事件，每个同学指的是需要响应事件的 DOM元素，而出去统一领取快递的宿舍长就是代理的元素

所以真正绑定事件的是这个元素，按照收件人分发快递的过程就是在事件执行中，需要判断当前响应的事件应该匹配到被代理元素中的哪一个或者哪几个

## 二、应用场景

如果我们有一个列表，列表之中有大量的列表项，我们需要在点击列表项的时候响应一个事件

<ul id="list">  
 <li>item 1</li>  
 <li>item 2</li>  
 <li>item 3</li>  
 ......  
 <li>item n</li>  
</ul>

如果给每个列表项一一都绑定一个函数，那对于内存消耗是非常大的

// 获取目标元素  
const lis = document.getElementsByTagName("li")  
// 循环遍历绑定事件  
for (let i = 0; i < lis.length; i++) {  
 lis[i].onclick = function(e){  
 console.log(e.target.innerHTML)  
 }  
}

这时候就可以事件委托，把点击事件绑定在父级元素ul上面，然后执行事件的时候再去匹配目标元素

// 给父层元素绑定事件  
document.getElementById('list').addEventListener('click', function (e) {  
 // 兼容性处理  
 var event = e || window.event;  
 var 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);  
 }  
});

还有一种场景是上述列表项并不多，我们给每个列表项都绑定了事件

但是如果用户能够随时动态的增加或者去除列表项元素，那么在每一次改变的时候都需要重新给新增的元素绑定事件，给即将删去的元素解绑事件

如果用了事件委托就没有这种麻烦了，因为事件是绑定在父层的，和目标元素的增减是没有关系的，执行到目标元素是在真正响应执行事件函数的过程中去匹配的

举个例子：

下面html结构中，点击input可以动态添加元素

<input type="button" name="" id="btn" value="添加" />  
<ul id="ul1">  
 <li>item 1</li>  
 <li>item 2</li>  
 <li>item 3</li>  
 <li>item 4</li>  
</ul>

使用事件委托

const oBtn = document.getElementById("btn");  
const oUl = document.getElementById("ul1");  
const num = 4;  
  
//事件委托，添加的子元素也有事件  
oUl.onclick = function (ev) {  
 ev = ev || window.event;  
 const target = ev.target || ev.srcElement;  
 if (target.nodeName.toLowerCase() == 'li') {  
 console.log('the content is: ', target.innerHTML);  
 }  
  
};  
  
//添加新节点  
oBtn.onclick = function () {  
 num++;  
 const oLi = document.createElement('li');  
 oLi.innerHTML = `item ${num}`;  
 oUl.appendChild(oLi);  
};

可以看到，使用事件委托，在动态绑定事件的情况下是可以减少很多重复工作的

## 三、总结

适合事件委托的事件有：click，mousedown，mouseup，keydown，keyup，keypress

从上面应用场景中，我们就可以看到使用事件委托存在两大优点：

* 减少整个页面所需的内存，提升整体性能
* 动态绑定，减少重复工作

但是使用事件委托也是存在局限性：

* focus、blur这些事件没有事件冒泡机制，所以无法进行委托绑定事件
* mousemove、mouseout这样的事件，虽然有事件冒泡，但是只能不断通过位置去计算定位，对性能消耗高，因此也是不适合于事件委托的

如果把所有事件都用事件代理，可能会出现事件误判，即本不该被触发的事件被绑定上了事件