顾名思义，“事件代理”即是把原本需要绑定在子元素的响应事件（click、keydown.....）委托给父元素，让父元素担当事件监听的职务。事件代理的原理是**DOM元素的事件冒泡**。

捕获阶段：从window对象传导到目标节点（上层传到底层）称为“捕获阶段”（capture phase），捕获阶段不会响应任何事件；
目标阶段：在目标节点上触发，称为“目标阶段”
冒泡阶段：从目标节点传导回window对象（从底层传回上层），称为“冒泡阶段”（bubbling phase）。事件代理即是利用事件冒泡的机制把里层所需要响应的事件绑定到外层；

事件委托的优点：

1. 可以大量节省内存占用，减少事件注册，比如在ul上代理所有li的click事件。

2. 可以实现当新增子对象时无需再对其绑定。

实现方式

jQuery事件delegate()实现事件委托

```
$("#myLinks").delegate("#goSomewhere", "click", function () {
  location.href = "http://www.baidu.com";
});
```

事件委托的注意事项：

使用“事件委托”时，并不是说把事件委托给的元素越靠近顶层就越好。事件冒泡的过程也需要耗时，越靠近顶层，事件的”事件传播链”越长，也就越耗时。如果DOM嵌套结构很深，事件冒泡通过大量祖先元素会导致性能损失

如何给动态生成的元素进行事件委托：

有 on ，live ，delegate ，bind

1. live 把事件委托交给了document（根节点），document 向下去寻找符合条件的元素（），不用等待document加载结束也可以生效。

2. delegate可指定事件委托对象，相比于live性能更优，直接锁定指定选择器；

3. on事件委托对象选填，如果不填，即给对象自身注册事件，填了作用和delegate一致。

4. bind只能给调用它的时候已经存在的元素绑定事件，不能给未来新增的元素绑定事件，存在局限性。

原生js实现jq的on方法（如下）

```
HTMLElement.prototype.on = function(events, callback){
    let evs = events.split(' ');
    for(let event of evs){
        this.addEventListener(event, callback);
    }
    // 如果你想像JQuery一样支持链式调用，可以在这里返回this
    // return this;
}
```