21 DOM事件体系（二）

更新时间：2020-04-30 08:59:53

> 耐心是一切聪明才智的基础。——柏拉图

本节我们针对 DOM 事件的两个重要的命题点展开剖析。

自定义事件

大家到现在所了解到的事件，基本都离不开浏览器的行为。比如点击鼠标、按下键盘等等，这些都可以被浏览器感知到，进而帮助我们转换成一个“信号”触发对应处理函数。但是还有一些行为，是浏览器感知不到的。比如说大家看这样一段 html：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
  <style>
    #divA,
    #divB,
    #divC {
      width: 100px;
      height:100px;
    }

    #divA {
      background-color: #333;
    }

    #divB {
      background-color: #dd5990;
    }

    #divC {
      background-color: #ccc990;
    }
  </style>
</head>
<body>
  <div id="divA">我是A</div>
  <div id="divB">我是B</div>
  <div id="divC">我是C</div>
</body>
</html>
```

如果我们仅仅想监听 divA 这个元素上的点击行为，我们可以用 addEventListener 来安装监听函数：

```
<script>
  var divA = document.getElementById('divA')
  document.addEventListener('click',function(){
    console.log('我是小A')

  })
</script>
```

这是大家非常熟悉的操作。但是，如果我现在想实现这样一种效果：

在点击A之后，B 和 C 都能感知到 A 被点击了，并且做出相应的行为——就像这个点击事件是点在 B 和 C 上一样。

是不是觉得有点意思了？我们知道，借助时间捕获和冒泡的特性，我们是可以实现父子元素之间的行为联动的。但是此处，A、B、C三者位于同一层级，他们怎么相互感知对方身上发生了什么事情呢？

这道题的解法其实有很多，尤其是在流行框架频出的今时今日，相信大家能联想到的办法就更多了。但是在早年，在工程师们的手段还没有这么多的时候，这道题最经典的解法就是咱们标题里写的——自定义事件。

“A被点击了”这件事情，可以作为一个事件来派发出去，由 B 和 C 来监听这个事件，并执行各自身上安装的对应的处理函数。在这个思路里，“A被点击了”这个动作挺特别，特别就特别在浏览器不认识它。因为浏览器不认识它，所以浏览器不肯”帮忙“，不会帮咱去感知和派发这个动作。不过没关系，感知和派发，咱都可以自己来实现：

首先大家需要了解的是，自定义事件的创建。比如说咱们要创建一个本来不存在的"clickA"事件，来表示 A 被点击了，咱们可以这么写：

```
var clickAEvent = new Event('clickA');
```

OK，现在事件有了，我们来完成事件的监听和派发：

```
// 获取 divB 元素
var divB = document.getElementById('divB')
// divB 监听 clickA 事件
divB.addEventListener('clickA',function(e){
  console.log('我是小B，我感觉到了小A')
  console.log(e.target)
})

// 获取 divC 元素
var divC = document.getElementById('divC')
// divC 监听 clickA 事件
divC.addEventListener('clickA',function(e){
  console.log('我是小C，我感觉到了小A')
  console.log(e.target)
})

// A 元素的监听函数也得改造下
divA.addEventListener('click',function(){
  console.log('我是小A')
  // 注意这里 dispatch 这个动作，就是我们自己派发事件了
  divB.dispatchEvent(clickAEvent)
  divC.dispatchEvent(clickAEvent)
})
```

我们可以看到如下输出：

说明我们的自定义事件生效了！

不过大家可能也注意到了，这里我们安装和派发事件必须得拿到 divC、divA 这样确切的元素才行。有的时候，为了进一步解耦，我们也会考虑把所有的监听函数都塞到 document 这个元素上去，由它来根据不同类型的自定义事件采取不同的动作。这种思路，就和事件代理非常相似了。

事件代理

前面咱们讲自定义事件，主要是怕大家掉坑里了。啥意思呢？我个人其实是很少考察候选人自定义事件的，在我的观念里，我默认你 js 基础只要够好，这题一定没问题（js 基础可以通过其它形式考察）。但是，对一些原生 js 情怀非常强烈的面试官，尤其是从前端史前时期一路写过来、做了 Team Leader 后代码越写越少不太能跟上潮流的面试官来说，自定义事件能说明很多问题（毕竟他早年也写了不少）。咱们防患于未然，不管他要不要考，只管先学过来。

自定义事件的考察频率，因面试官而异。而事件代理，则是妥妥的高频考点。你落到我手里，我得问你；你落到我同事手里，我同事也得问你。很多同学嫌太基础，觉得是问应届生的问题。这个想法对了一半——确实是基础，不过基础的考察可是不分人群的。所以说，事件代理可得打起精神来学了！（敲黑板）

事件代理，又叫事件委托。如果你之前没有了解过这玩意儿是啥，现在也不用急着去搜概念。这玩意儿不适合一上来就怼概念，我们直接来做题：

真题：在如下的 HTML 里，我希望做到点击每一个 li 元素，都能输出它内在的文本内容。你会怎么做？

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <ul id="poem">
    <li>鹅鹅鹅</li>
    <li>曲项向天歌</li>
    <li>白毛浮绿水</li>
    <li>红掌拨清波</li>
    <li>锄禾日当午</li>
    <li>汗滴禾下土</li>
    <li>谁知盘中餐</li>
    <li>粒粒皆辛苦</li>
    <li>背不动了</li>
    <li>我背不动了</li>
  </ul>
</body>
</html>
```

一个比较直观的思路是让每一个 li 元素都去监听一个点击动作：

```
<script>
  // 获取 li 列表
  var liList = document.getElementsByTagName('li')
  // 逐个安装监听函数
  for (var i = 0; i < liList.length; i++) {
    liList[i].addEventListener('click', function (e) {
      console.log(e.target.innerHTML)
    })
  }
</script>
```

你要真这么干，这题一分都没有。谨记，一旦在 DOM 系列面试题中遇到符合下列三个特征的问题：

1. 要你安装监听某一个事件的监听函数（事件相同）
2. 监听函数是被安装在一系列具有相同特征的元素上（元素特征相同，一般来说就是具备同样的父元素）
3. 这一系列相同特征元素上的监听函数还干的都是一样的事儿（监听逻辑相同/雷同）

这时候你脑子里一定要冒出这四个大字——事件代理！

怎么搞？楼上我们给10个 li 安装了 10 次监听函数，累不累？开销大不大？仔细想想，10个监听函数干的还都是一样的事情，咱们能不能把这个逻辑收敛到一个元素上去？答案是能，为啥能？因为有事件冒泡啊！你想，点击任何一个 li，是不是这个点击事件都会被冒到它们共同的爹地—— ul 元素上去？那我能不能让 ul 来做这个事儿？答案是能，因为 ul 不仅能感知到这个冒上来的事件，它还可以通过 e.target 拿到实际触发事件的那个元素，做到无缝代理：

```
var ul = document.getElementById('poem')
ul.addEventListener('click', function(e){
  console.log(e.target.innerHTML)
})
```

大家谨记，e.target 就是指触发事件的具体目标，它记录着事件的源头。所以说，不管咱们的监听函数在哪一层执行，只要我拿到这个 e.target，就相当于拿到了真正触发事件的那个元素。拿到这个元素后，我们完全可以模拟出它的行为，实现无差别的监听效果。

像这样利用事件的冒泡特性，把多个子元素的同一类型的监听逻辑，合并到父元素上通过一个监听函数来管理的行为，就是事件代理。通过事件代理，我们可以减少内存开销、简化注册步骤，大大提高开发效率。

}

20 DOM 事件体系（一）

22 真题手把手：事件的防抖与节流