20 DOM 事件体系（一）

更新时间：2020-05-12 13:57:53

> 先相信你自己，然后别人才会相信你。——屠格涅夫

DOM 事件体系相关的基本功，大家需要从以下几个方面来掌握：

- DOM 事件流
- 事件对象
- 事件代理
- 自定义事件

DOM 事件流

前置知识

在理解事件流之前，大家首先要对以下三个名词有确切的认知：

> 事件流：它描述的是事件在页面中传播的**顺序**

> 事件：它描述的是发生在浏览器里的动作。这个动作可以是用户触发的，也可以是浏览器触发的。像点击（click）、鼠标悬停（mouseover）、鼠标移走（mousemove）这些都是事件。

> 事件监听函数：事件发生后，浏览器如何响应——用来应答事件的函数，就是事件监听函数，也叫事件处理程序。

事件流的演进

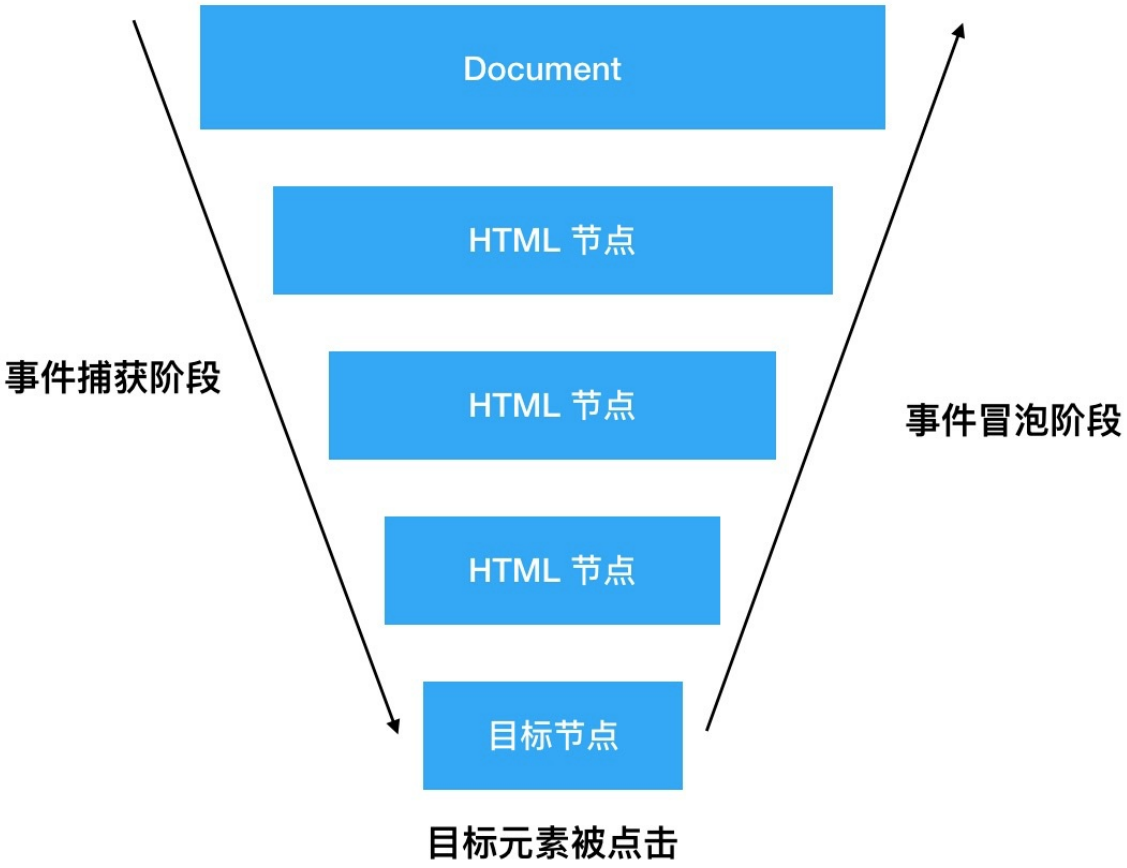
当下广为大家所接受的 JS 事件流规范，也并非一蹴而就。早年， IE 和 NetScape 两家浏览器厂商在事件机制的设计上，争得你死我活，谁也不认可谁。IE 提出了冒泡流，而 NetScape 只认捕获流。两家各干各的，搞得前端程序员那段日子过得很难，每次做网页兼容性适配都是一把鼻涕一把泪。好在后来正义的 W3C 介入了，在 W3C 的统一组织下，JS 同时支持了冒泡流和捕获流，并以此为确切的事件流标准。这个标准也叫做“DOM2事件流”。不标准的我们不聊，下面我们所有的讨论，都围绕这个板上钉钉的“DOM2事件流”展开。

一个事件的旅行

W3C 标准约定了一个事件的传播过程要经过以下三个阶段：

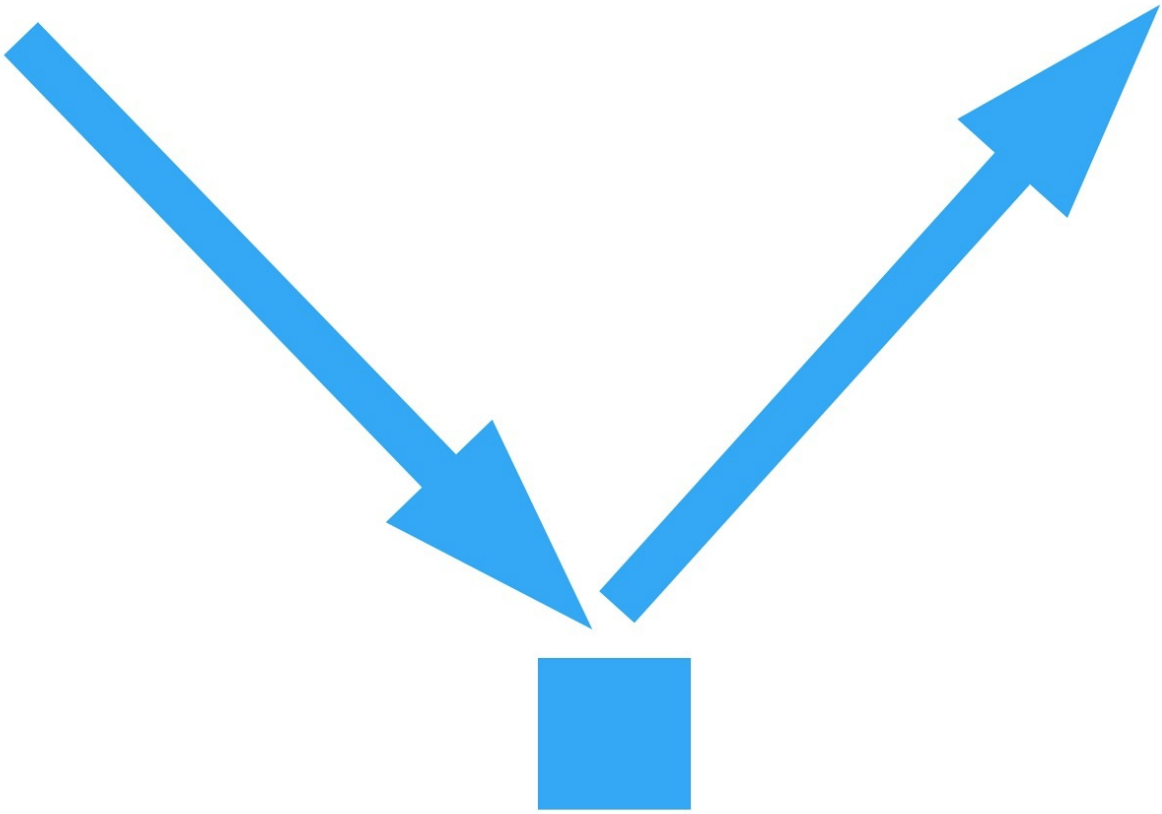
1. 事件捕获阶段
2. 目标阶段
3. 事件冒泡阶段

理解这个过程最好的方式就是读图了，下图中的箭头就代表着时间的“穿梭”路径：



当事件被触发时，首先经历的是一个捕获过程：事件会从最外层的元素开始“穿梭”，逐层“穿梭”到最内层元素。这个穿梭过程会持续到事件抵达它目标的元素（也就是真正触发这个事件的元素）为止。此时事件流就切换到了“目标阶段”——事件被目标元素所接收。然后事件会被“回弹”，进入到冒泡阶段——它会沿着来时的路“逆流而上”，一层一层再走回去。

这个过程很像是大家小时候玩蹦床：从高处下落，触达蹦床后再弹起、回到高处，整个过程呈一个对称的“V”字形：



事件对象基础

当事件在层层的 DOM 元素中穿梭时，它可没闲着——所到之处，它都会触发当前元素上安装的事件处理函数。比如说你点击了上面图示中的 button 节点，但其实 div 节点上也安装了 click 这个事件的处理函数。那么当你点击 button 节点触发的这个 click 事件经过 div 节点时，div 节点上的处理函数照样会被触发。

当 DOM 接受了一个事件、对应的事件处理函数被触发时，就会产生一个事件对象 event 作为处理函数的入参。这个对象中囊括了与事件有关的信息，比如事件具体是由哪个元素所触发、事件的类型等等。

在触发 DOM 上的某个事件时，会产生一个事件对象 event。这个对象中包含着所有与事件有关的信息。

包括导致事件的元素，事件的类型以及其他与特定事件相关的信息。

现在我们来写一个简单的 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>
</head>
<body>
  <div class="outer">
    <button id="button">点击我</button>
  </div>
</body>
</html>
```

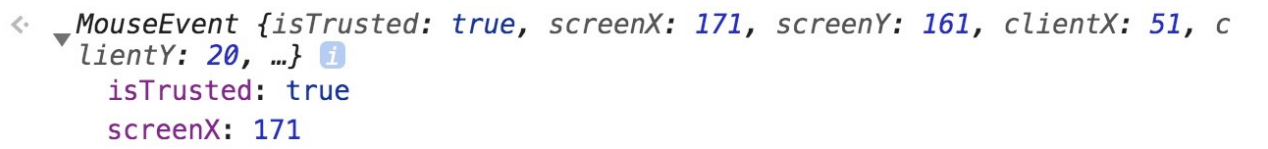
咱们想看看这个 DEMO 中 button 的点击事件对象是啥样的，我们可以这么写对应的处理函数：

```
function clickProcessor(event) {
  console.log(event)
}
```

没错，event 就是事件处理函数的第一个入参。我们把这个处理函数安装到 button 身上：

```
const button = document.getElementById('button')
button.addEventListener('click', clickProcessor)
```

现在我们触发一个鼠标的点击动作时，对应的 event 对象就长这样：



我们可以看到里面除了一事件普遍都会具备的基本的信息，还包括了一些某一类事件特有的补充信息（比如针对点击事件，这里记录了鼠标位置相关的信息）。这个事件对象很有意思，我们接下来着重对它展开剖析。

事件对象考点梳理

在事件对象中，有一些属性和方法，是我们特别常用的。这部分东西比较碎，但考察频率较高。面试官有时会单独问你，但更多的是倾向于在编码类题目中直接看你能不能用它们来写代码。我们把这部分考点总结如下：

1. currentTarget

它记录了事件当下正在被哪个元素接收，即”正在经过哪个元素“。这个元素是一直在改变的，因为事件的传播毕竟是个层层穿梭的过程。

如果事件处理程序绑定的元素，与具体的触发元素是一样的，那么函数中的 `this`、event.`currentTarget`、和 event.`target` 三个值是相同的。我们可以以此为依据，判断当前的元素是否就是目标元素。

2. target

指触发事件的具体目标，也就是最具体的那个元素，是事件的真正来源。

就算事件处理程序没有绑定在目标元素上、而是绑定在了目标元素的父元素上，只要它是由内部的目标元素冒泡到父容器上触发的，那么我们仍然可以通过 target 来感知到目标元素才是事件真实的来源。

（以上两个属性，是我们下节实现事件代理的好帮手）

3. preventDefault 阻止默认行为

这个方法用于阻止特定事件的默认行为，如 `a` 标签的跳转等。

```
e.preventDefault();
```

stopPropagation 不再派发事件

这个方法用于终止事件在传播过程的捕获、目标处理或起泡阶段进一步传播。调用该方法后，该节点上处理该事件的处理程序将被调用，事件不再被分派到其他节点。

```
e.stopPropagation();
```

有时我们不希望一个事件的触发带来“一石激起千层浪”的效果，希望把它的影响面控制在目标元素这个范围内。这种情况下，千万别忘了 stopPropagation。

事件对象，是可以手动创建的

事件对象不一定需要你通过触发某个具体的事件来让它“自然发生”，它也可以手动创建的：

我们可以借助 Event() 构造函数, 来创建一个新的事件对象 Event。

```
event = new Event(typeArg, eventInit);
```

事件对象的这个特性，是我们创建自定义事件的基础——可能一些同学对自定义事件还比较陌生，但它确实非常重要。在四五年前，自定义事件就已经是考察一个前端是否资深的重要标准。随着前端技术的蓬勃发展，对资深前端的要求不断提高，自定义事件的能力变成了基础层次的能力，但它的不可或缺性仍然不可改变。

结语

本节，我们针对事件机制中最最基础的一系列知识点进行了复习。在下节，我们会基于此展开对自定义事件、事件的绑定/委托等具备一定综合性的面试热点展开探讨。

}

19 DOM基本功

21 DOM事件体系（二）