

浏览器事件：为什么会有捕获过程和冒泡过程？

winter 2019-04-16



你好，我是 winter。这一节课，我们进入了浏览器的部分，一起来学习一下事件。

事件概述

在开始接触具体的 API 之前，我们要先了解一下事件。一般来说，事件来自输入设备，我们平时的个人设备上，输入设备有三种：

键盘；

鼠标；

触摸屏。

这其中，触摸屏和鼠标又有一定的共性，它们被称作 **pointer 设备**，所谓 **pointer 设备**，是指它的输入最终会被抽象成屏幕上面的一个点。但是触摸屏和鼠标又有一定区别，它们的精度、反应时间和支持的点的数量都不一样。

我们现代的 UI 系统，都源自 WIMP 系统。WIMP 即 Window Icon Menu Pointer 四个要素，它最初由施乐公司研发，后来被微软和苹果两家公司应用在了自己的操作系统上（关于这个还有一段有趣的故事，我附在文末了）。

WIMP 是如此成功，以至于今天很多的前端工程师会有一个观点，认为我们能够“点击一个按钮”，实际上并非如此，我们只能够点击鼠标上的按钮或者触摸屏，是操作系统和浏览器把这个信息对应到了一个逻辑上的按钮，再使得它的视图对点击事件有反应。这就引出了我们第一个要讲解的机制：捕获与冒泡。

捕获与冒泡

很多文章会讲到捕获过程是从外向内，冒泡过程是从内向外，但是这里我希望讲清楚，为什么会有捕获过程和冒泡过程。

我们刚提到，实际上点击事件来自触摸屏或者鼠标，鼠标点击并没有位置信息，但是一般操作系统会根据位移的累积计算出来，跟触摸屏一样，提供一个坐标给浏览器。

那么，把这个坐标转换为具体的元素上事件的过程，就是捕获过程了。而冒泡过程，则是符合人类理解逻辑的：当你按电视机开关时，你也按到了电视机。

所以我们可以认为，捕获是计算机处理事件的逻辑，而冒泡是人类处理事件的逻辑。

以下代码展示了事件传播顺序：

```
1 <body>
2   <input id="i"/>
3 </body>
```

 复制代码

```
1 document.body.addEventListener("mousedown", () => {
2   console.log("key1")
3 }, true)
4
5 document.getElementById("i").addEventListener("mousedown", () => {
6   console.log("key2")
7 }, true)
8
9 document.body.addEventListener("mousedown", () => {
```

 复制代码

```
10     console.log("key11")
11 }, false)
12
13 document.getElementById("i").addEventListener("mousedown", () => {
14     console.log("key22")
15 }, false)
```

我们监听了 body 和一个 body 的子元素上的鼠标按下事件，捕获和冒泡分别监听，可以看到，最终产生的顺序是：

“key1”

“key2”

“key22”

“key11”

这是捕获和冒泡发生的完整顺序。

在一个事件发生时，捕获过程跟冒泡过程总是先后发生，跟你是否监听毫无关联。

在我们实际监听事件时，我建议这样使用冒泡和捕获机制：默认使用冒泡模式，当开发组件时，遇到需要父元素控制子元素的行为，可以使用捕获机制。

理解了冒泡和捕获的过程，我们再看监听事件的 API，就非常容易理解了。

addEventListener 有三个参数：


事件名称；

事件处理函数；

捕获还是冒泡。

事件处理函数不一定是函数，也可以是个 JavaScript 具有 handleEvent 方法的对象，看下例子：

```
1 var o = {
2   handleEvent: event => console.log(event)
```

 复制代码

```
3 }  
4 document.body.addEventListener("keydown", o, false);
```

第三个参数不一定是 bool 值，也可以是个对象，它提供了更多选项。

once：只执行一次。

passive：承诺此事件监听不会调用 preventDefault，这有助于性能。

useCapture：是否捕获（否则冒泡）。

实际使用，在现代浏览器中，还可以不传第三个参数，我建议默认不传第三个参数，因为我认为冒泡是符合正常的人类心智模型的，大部分业务开发者不需要关心捕获过程。除非你是组件或者库的使用者，那就总是需要关心冒泡和捕获了。

焦点

我们讲完了 pointer 事件是由坐标控制，而我们还没有讲到键盘事件。

键盘事件是由焦点系统控制的，一般来说，操作系统也会提供一套焦点系统，但是现代浏览器一般都选择在自己的系统内覆盖原本的焦点系统。

焦点系统也是视障用户访问的重要入口，所以设计合理的焦点系统是非常重要的产品需求，尤其是不少国家对可访问性有明确的法律要求。


在旧时代，有一个经典的问题是如何去掉输入框上的虚线框，这个虚线框就是 Windows 焦点系统附带的 UI 表现。

现在 Windows 的焦点已经不是用虚线框表示了，但是焦点系统的设计几十年间没有太大变化。

焦点系统认为整个 UI 系统中，有且仅有一个“聚焦”的元素，所有的键盘事件的目标元素都是这个聚焦元素。

Tab 键被用来切换到下一个可聚焦的元素，焦点系统占用了 Tab 键，但是可以用 JavaScript 来阻止这个行为。

浏览器 API 还提供了 API 来操作焦点，如：

 复制代码


```
1 document.body.focus();  
2  
3 document.body.blur();
```

其实原本键盘事件不需要捕获过程，但是为了跟 pointer 设备保持一致，也规定了从外向内传播的捕获过程。

自定义事件

除了来自输入设备的事件，还可以自定义事件，实际上事件也是一种非常好的代码架构，但是 DOM API 中的事件并不能用于普通对象，所以很遗憾，我们只能在 DOM 元素上使用自定义事件。

自定义事件的代码示例如下（来自 MDN）：

 复制代码

```
1 var evt = new Event("look", {"bubbles":true, "cancelable":false});  
2 document.dispatchEvent(evt);
```

这里使用 Event 构造器来创建了一个新的事件，然后调用 dispatchEvent 来在特定元素上触发。

我们可以给这个 Event 添加自定义属性、方法。

注意，这里旧的自定义事件方法（使用 document.createEvent 和 initEvent）已经被废弃。

总结

今天这一节课，我们讲了浏览器中的事件。

我们分别介绍了事件的捕获与冒泡机制、焦点机制和自定义事件。

捕获与冒泡机制来自 pointer 设备输入的处理，捕获是计算机处理输入的逻辑，冒泡是人类理解事件的思维，捕获总是在冒泡之前发生。

焦点机制则来自操作系统的思路，用于处理键盘事件。除了我们讲到的这些，随着输入设备的不断丰富，还有很多新的事件加入，如 Geolocation 和陀螺仪等。

最后给你留个小问题。请你找出你所知道的所有事件类型，和它们的目标元素类型。

WIMP 的小故事

WIMP 是由 Alan Kay 主导设计的，这位巨匠，同时也是面向对象之父和 Smalltalk 语言之父。

乔布斯曾经受邀参观施乐，他见到当时的 WIMP 界面，认为非常惊艳，不久后就领导苹果研究了新一代麦金塔系统。

后来，在某次当面对话中，乔布斯指责比尔盖茨抄袭了 WIMP 的设计，盖茨淡定地回答：“史蒂夫，我觉得应该用另一种方式看待这个问题。这就像我们有个叫施乐的有钱邻居，当我闯进去想偷走电视时，却发现你已经这么干了。”

但是不论如何，苹果和微软的数十代操作系统，极大地发展了这个体系，才有了我们今天的 UI 界面。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言(16)



阿成

事件的种类对应了浏览器的能力，我们并不需要记住所有的事件及其细节，我们只需要在用到某种能力的时候去查找相应的事件类型下的某一事件即可。

补充一下楼下仁兄的答案：移动端设备上的touch系列，以及新标准(好像也挺多年了，不过兼容性让人望而却步...)的pointer系列

2019-04-16



9



injser

评论越来越少...果然贵在坚持

2019-05-30



1



5



bestRktnZnnn

请问自定义事件主要应用场景是什么呢

2019-05-23



2



4



Even

<https://developer.mozilla.org/zh-CN/docs/Web/Events> 直接看MDN

2019-04-26



3



fighting

01: UI事件, load, unload判断页面是都加载完成;

02: 焦点事件, focus, blur

03: 鼠标事件, click, dblclick, mousedown, mouseup, scroll

04: 键盘事件 keydown, keyup, keypress, textInput,

2019-08-16



2



令狐洋葱

老师对于这类历史还是有一些了解的哈, 不知道这类知识哪里获取的, 是想了解就去搜索的么?

作者回复: 前面有讲学习方法

2019-08-15



1



sugar

补充一下：最后一关wimp小故事 我在乔布斯传上也看到过～ 想看完整版的可以去了解^^

2020-04-27



withMango

其实我也要看书去补充具体的详情

2020-01-08



junjun

感觉讲的最好和实际应用联系起来，不然完全看不出有什么意义，毕竟最后还是为了应用。

2019-10-09



马成

事件触发的顺序总是 先从外向内捕获，然后再从内向外冒泡

2019-09-05



ethan

解惑了，赞

2019-06-12



张驰Terry

讲得真好

2019-06-04



桃翁

我其实没有明白为什么写组件就要用到捕获事件呢？

2019-05-23



1





dwqyun

那子元素的事件委托给父元素时，添加事件监听的第三个参数直接设置为true，在捕获过程就判断父元素上的事件目标会比冒泡好些嘛？

2019-04-24



花狗是我

这钱花的值

2019-04-18



レイン小雨

有意思

2019-04-16

