

36 | 浏览器事件：为什么会有捕获过程和冒泡过程？ | 极客时间

winter 2019-04-16



讲述：winter 大小：7.46M

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出，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：

```
<body>
  <input id="i"/>
</body>
```

□复制代码

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：

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

```
document.body.focus();
document.body.blur();
```

[📄 复制代码](#)

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）：

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

[📄复制代码](#)

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。



重学前端

每天 10 分钟，重构你的前端知识体系

winter 程劭非
前手机淘宝前端负责人



新版升级：点击「👤请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 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。



bd2star

精选留言(3)

•



Carson

尝试按照老师提到的完备性原则，列出所有类型。经过查找，发现自己之前了解的事件类型只占全部类型不到三分之一。

由于留言 2000 字的限制，在末尾省略了一些具体事件名称。

资源事件 (Resource events)

目标元素：有 src 属性的元素。

事件：

- * cached
- * error
- * abort
- * load
- * beforeunload
- * unload

网络事件 (Network events)

目标元素：浏览器 (window) 各家浏览器在这个事件上没有达成共识。

- * online
- * offline

焦点事件 (Focus events)

目标元素：包括四类元素，链接类元素，表格类元素，嵌入类元素，所有带有 tabindex 属性的元素

事件：

- * focus
- * blur

WebSocket 事件 (WebSocket events)

目标元素：WebSocket

事件：

- * open

- * message
- * error
- * close

历史事件 (Session History events)

目标元素: 浏览器

事件:

- * pagehide
- * pageshow
- * popstate

CSS 动画事件 (CSS Animation events)

目标元素: 带 CSS 动画的元素

事件:

- * animationstart
- * animationend
- * animationiteration

CSS 过渡事件 (CSS Transition events)

目标元素: 带 CSS 过渡的元素

事件:

- * transitionstart
- * transitioncancel
- * transitionend
- * transitionrun

表格事件 (Form events)

目标元素：form 元素

事件：

- * reset
- * submit
- * change

打印事件 (Printing events)

目标元素：浏览器 (window)

事件：

- * beforeprint
- * afterprint

文字输入事件 (Text Composition events)

目标元素：可以输入文字的元素。使用拼音输入法也会触发该事件。

事件：

- * compositionstart
- * compositionupdate
- * compositionend

视窗事件 (View events)

目标元素：浏览器 (window) 或 document

事件：

- * fullscreenchange
- * fullscreenerror
- * resize
- * scroll

粘贴板事件 (Clipboard events)

目标元素: 任何指定元素或者粘贴板对象

事件:

- * cut
- * copy
- * paste

键盘事件 (Keyboard events)

目标元素: 任何指定元素

事件:

- * keydown
- * keypress
- * keyup

鼠标事件 (Mouse events)

目标元素: 任何指定元素

事件:

- * auxclick
- * click
- * contextmenu
- * dblclick
- * mousedown
- * mouseenter
- * mouseleave
- * mousemove
- * mouseover
- * mouseout
- * mouseup
- * pointerlockchange

- * pointerlockerror
- * select
- * wheel

拖拽事件 (Drag & Drop events)

目标元素：当前被拖拽的元素

事件：

- * drag
- * dragend
- * dragenter
- * dragstart
- * dragleave
- * dragover
- * drop

多媒体事件 (Media events)

目标元素：多媒体元素，包括 video 和 audio 元素。

事件：省略。

幕后事件 (Progress events)

目标元素：

- * 在进行 XMLHttpRequest 的元素
- * 读取外部资源的元素，包括 img、audio、video、style 或者 link

事件：省略。

存储事件 (Storage events)

事件：

* change (目标元素 input、select 和 textarea)

* storage (localStorage 和 sessionStorage)

□

23