



# 17-1 React 中的合成事件背景与其必要性

## 资源

### 1. 响应事件

使用 React 可以在 JSX 中添加 **事件处理函数**。其中事件处理函数为自定义函数，它将在响应交互（如点击、悬停、表单输入框获得焦点等）时触发。

React 基于浏览器的事件机制自身实现了一套事件机制，包括事件注册、事件冒泡、事件派发等，这套事件机制我们通常称之为**合成事件**。

相比于原生事件机制，React 合成事件机制不仅考虑了浏览器兼容性，不需要用户再去做这些兼容性开发，只需要关注 React 本身。React 合成事件还考虑了内存与性能，如实现了事件委托。

## 内存与性能

因为事件处理程序在现代 Web 应用中可以实现交互，所以很多开发者会错误地在页面中大量使用它们。在创建 GUI 的语言如 C# 中，通常会给 GUI 上的每个按钮设置一个 onclick 事件处理程序。这样做并不会有什么性能损耗。在 JavaScript 中，页面中

事件处理程序的数量与页面整体性能直接相关。原因有很多。首先，每个函数都是对象，都占用内存空间，对象越多，性能越差。其次，为指定事件处理程序所需访问 DOM 的次数会先期造成整个页面交互的延迟。只要在使用事件处理程序时多注意一些方法，就可以改善页面性能。

## 事件委托

”过多事件处理程序“的解决方案是使用**事件委托**。**事件委托利用事件冒泡，可以只使用一个事件处理程序来管理一种类型的事件**。例如，click 事件冒泡到 document。这意味着可以为整个页面指定一个 onclick 事件处理程序，而不用为每个可点击元素分别指定事件处理程序。

因此，只要可行，就应该考虑只给 document 添加一个事件处理程序，通过它处理页面中所有某种类型的事件。事件委托具有以下优点：

1. document 对象随时可用，任何时候都可以给它添加事件处理程序，（不用等待 DOMContentLoaded 或 load 事件）。这意味着只要页面渲染出可点击的元素，就可以无延迟地起作用。
2. 节省花在设置页面事件处理程序上的时间。只指定一个事件处理程序既可以节省 DOM 引用，也可以节省时间。
3. 减少整个页面所需的内存，提升整体性能。

以上内容来自《JavaScript 高级程序设计》第 4 版。

## React 中的事件委托

React 中的合成事件是使用了事件委托。并且 React 结合自己的场景，从 React17 之后，把事件委托在了可控 Container 层。

在 React 16 或更早版本中，React 会由于事件委托对大多数事件执行

`document.addEventListener()`。但是一旦你想要局部使用 React，那么 React 中的事件会影响全局，如下面这个例子，当把 React 和 jQuery 一起使用，那么当点击 input 的时候，document 上和 React 不相关的事件也会被触发，这符合 React 的预期，但是并不符合用户的预期。

## e.stopPropagation() seems to not be working as expect. #4335

 Closed

luokuning opened this issue on 10 Jul 2015 · 9 comments



luokuning commented on 10 Jul 2015



I found this issue when I attempted to integrate ReactJs and a jQuery widget.

I bind an click event on input element and document like follow:

```
var Search = React.createClass({
  handleClick: function(e) {
    e.stopPropagation();
  },
  render: function() {
    return <input onClick={this.handleClick} />
  }
});
document.addEventListener('click', function() {
  console.log('propagation')
}, false);
```

And everytime I click the input element, chrome devtool logs 'propagation' message. Should not document element can not receive click event? I'm confused, Did I miss something?



syranide commented on 10 Jul 2015

Contributor



React (currently) uses a listener on the document for (almost) all events, so by the time your component receives the event it has already bubbled all the way up to the document and stopping it only stops it from synthetically bubbling up through the React hierarchy.

Related [#2050](#)

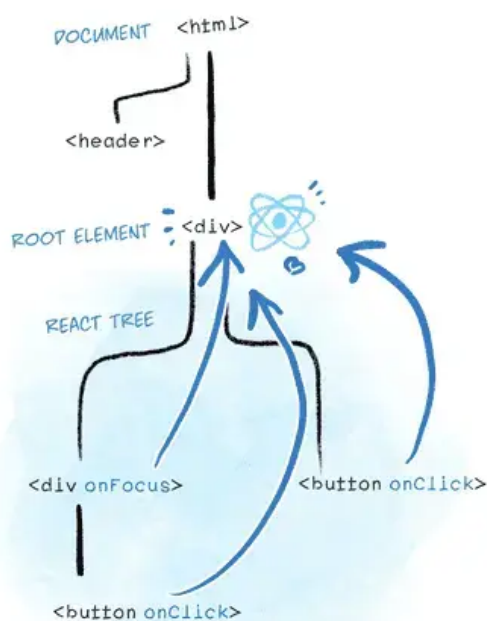
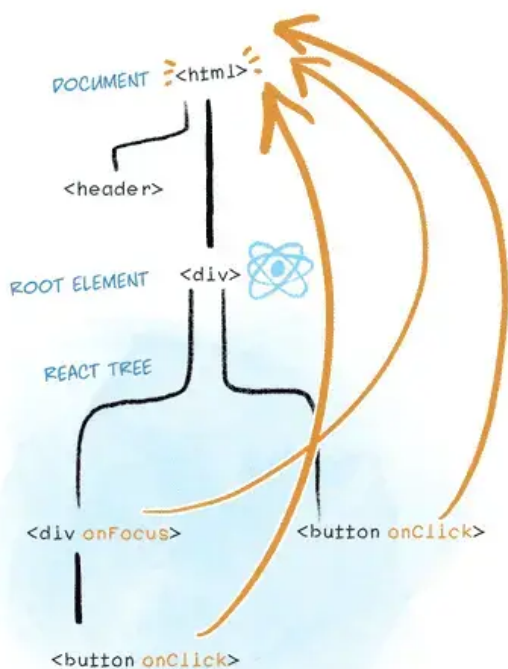
 2

@稀土掘金技术社区

令人开心的是，这次的 React 17 就解决了这个问题 ~，React17 之后 不再将事件添加在 `document` 上，而是添加到渲染 React 树的根 DOM 容器中：

这种改变不仅方便了局部使用 React 的项目，还可以用于项目的逐步升级，如一部分使用 React 18，另一部分使用 React 19，事件是分开的，这样也就不会相互影响。当然这并不是鼓励大家在一个项目中使用多个 React 版本，而只是作为一种临时处理的过渡 ~

下图形象描述了这次的变更：



@稀土掘金技术社区

## 不适合委托的事件

但是并不是所有事件都适合委托，比如 cancel、scroll 等，这些事件在 DOM 中的冒泡行为并不一致，因此 React 中专门列举出了不把事件委托给 container 的事件：

react/packages/react-dom-bindings/src/events/DOMPluginEventSystem.js

JavaScript

// 需要分别附加到媒体元素的事件列表。

```
export const mediaEventTypes: Array<DOMEventName> = [
  'abort',
  'canplay',
  'canplaythrough',
  'durationchange',
  'emptied',
  'encrypted',
  'ended',
  'error',
```

```

    'loadeddata',
    'loadedmetadata',
    'loadstart',
    'pause',
    'play',
    'playing',
    'progress',
    'ratechange',
    'resize',
    'seeked',
    'seeking',
    'stalled',
    'suspend',
    'timeupdate',
    'volumechange',
    'waiting',
  ];

```

// 我们不应该将这些事件委托给容器，而是应该直接在实际的目标元素上设置它们。这主要是!

```

export const nonDelegatedEvents: Set<DOMEventName> = new Set([
  'cancel',
  'close',
  'invalid',
  'load',
  'scroll',
  'scrollend',
  'toggle',

```

// 为了减少字节数，我们将上述媒体事件数组插入到这个 Set 中。

// 注意："error" 事件并不是一个独占的媒体事件，也可能发生在其他元素上。我们不会!

```

    ...mediaEventTypes,
  ]);

```