

# useEffectEvent(钩子)

Eddy 艾迪

✉ [hello@eddychang.me](mailto:hello@eddychang.me)



# 什麼是 useEffectEvent (What)

`useEffectEvent`：一個React勾子，可以讓你從Effects中提取「非響應式邏輯(non-reactive logic)」，到一個被稱為 `Effect Event` 的可重覆使用的函式中

註: 最早(2018)在官方Github中 [issue#14099](#) 有針對 `useCallback` 在某些實作場景上的問題討論，社群中也有類似作用的自訂勾子(例如 `useEventCallback`)，後來RFC訂名為`useEvent`，在v18時的Canary頻道就已加入實驗性質實作(2022)

# 類型定義

```
useEffectEvent?: <Args, F: (...Array<Args>) => mixed>(callback: F) => F
```

泛型參數 `Args` 代表函數參數泛型類型，`F` 是 `(...Array<Args>) => mixed` 函式類型

函式簽名 接受一個類型 `F` 的 `callback`，回傳一個相同類型 `F` 的函式

`mixed` 類型 Flow 類型系統中所有類型的超類型(supertype)，相當於 TypeScript 的

`unknown` 類型。允許函式返回任何類型的值，提供靈活性(而不是 `void`)

# 語法

```
useEffectEvent(callback)
```

**參數 callback** 一個包含Effect Event邏輯的函式。當你用 `useEffectEvent` 定義一個Effect Event時，callback在被呼叫時，必定能從props與state存取到最新的值。這能協助避免「過期/陳舊閉包(stale closures)」的問題。

**回傳值** 回傳一個Effect Event函式。可以(也只能) `useEffect` , `useLayoutEffect` 或 `useInsertionEffect` 裡呼叫

## Reactive value(響應式值)

props、state 等可能因重新渲染而改變的值(註: ref與ref.current並不是)

## Non-reactive logic(非響應式邏輯)

事件處理函式中的邏輯，需"手動"觸發(如點擊按鈕)才會執行。可讀取響應式值，但不會自動響應其變動。

## Reactive logic(響應式邏輯)

Effects 中的邏輯，會"自動"執行或重新執行以進行同步化。需將響應式值加入相依變數，變動時 React 會用新的值重新執行 Effect。

# 什麼是 **Effect Event** (作用事件)

**Effect Events** 類似 **事件處理函式**，主要區別在於：

**事件處理函式**：響應 **使用者互動** 而執行（如點擊按鈕）

**Effect Event**：由你從 **Effects** (作用) 中觸發

**Effect Event** 讓你能夠在 Effect 的響應性(reactivity)中，與不應該具有響應性的程式碼之間「打破鏈結」，將非響應式邏輯從響應式邏輯中分離出，以下為幾個重點：

1. 它不是響應式的，必須從依賴項中省略
2. 它總是能存取到最新的**props**和**state**，可避免過期/陳舊閉包問題(穩定的函式引用)
3. 它只能在 Effects 裡呼叫，切勿不要傳遞到其它元件或勾子

## 為什麼要使用 useEffectEvent (Why)

1. 針對 `useCallback` 在特定使用場景下的應用問題(過於經常更新，導致穩定性問題)  
[issue#14099](#)
2. 有目的地搭配最新的 `React Compiler` 最佳化方式
3. 新版本中的 `eslint-plugin-react-hooks` 裡的 `set-state-in-effect` 檢查規則的  
解決方案之一 [issues/34743](#)

# 什麼時候和場景使用 useEffectEvent (When/Where)

「某些情況應該像事件一樣執行，且能存取到最新狀態與屬性」

1. 在Effects中混合了「響應式」與「非響應式邏輯」時，需要分離或提取出來的情況
2. 訂閱外部事件或計時器回呼 (Callbacks)、分析日誌記錄 (Analytics Logging)，回呼函式可能需要最新的 state 或 prop 值時
3. 避免 ESLint 錯誤與隱藏 Bug

來源: [useEvent RFC](#)

```
// (!) Approximate behavior
function useEvent(handler) {
  const handlerRef = useRef(null);

  // In a real implementation, this would run before layout effects
  useEffect(() => {
    handlerRef.current = handler;
  });

  return useCallback((...args) => {
    // In a real implementation, this would throw if called during render
    const fn = handlerRef.current;
    return fn(...args);
  }, []);
}
```

## 如何使用 useEffectEvent (How) - 原始範例

```
function ChatRoom({ roomId, theme }) {  
  useEffect(() => {  
    const connection = createConnection(serverUrl, roomId);  
    connection.on('connected', () => {  
      showNotification('Connected!', theme);  
    });  
    connection.connect();  
    return () => connection.disconnect();  
  }, [roomId, theme]);  
  
  return <h1>Welcome to the {roomId} room!</h1>  
}
```

# 如何使用 useEffectEvent (How) - 改進範例

```
function ChatRoom({ roomId, theme }) {  
  const onConnected = useEffectEvent(() => {  
    showNotification('Connected!', theme);  
  });  
  useEffect(() => {  
    const connection = createConnection(serverUrl, roomId);  
    connection.on('connected', () => {  
      onConnected();  
    });  
    connection.connect();  
    return () => connection.disconnect();  
  }, [roomId]);  
  
  return <h1>Welcome to the {roomId} room!</h1>  
}
```