ThePower business school

🇪🇸 UseEffect ReactJS[3]

Introducción

Vamos a continuar viendo como se ejecuta un useEffect después de cada renderizado. En ocasiones tenemos que hacer uso de la dependencias del useEffect para escuchar los cambios que se producen y dicho Hook volver a lanzar la funcionalidad asociada.

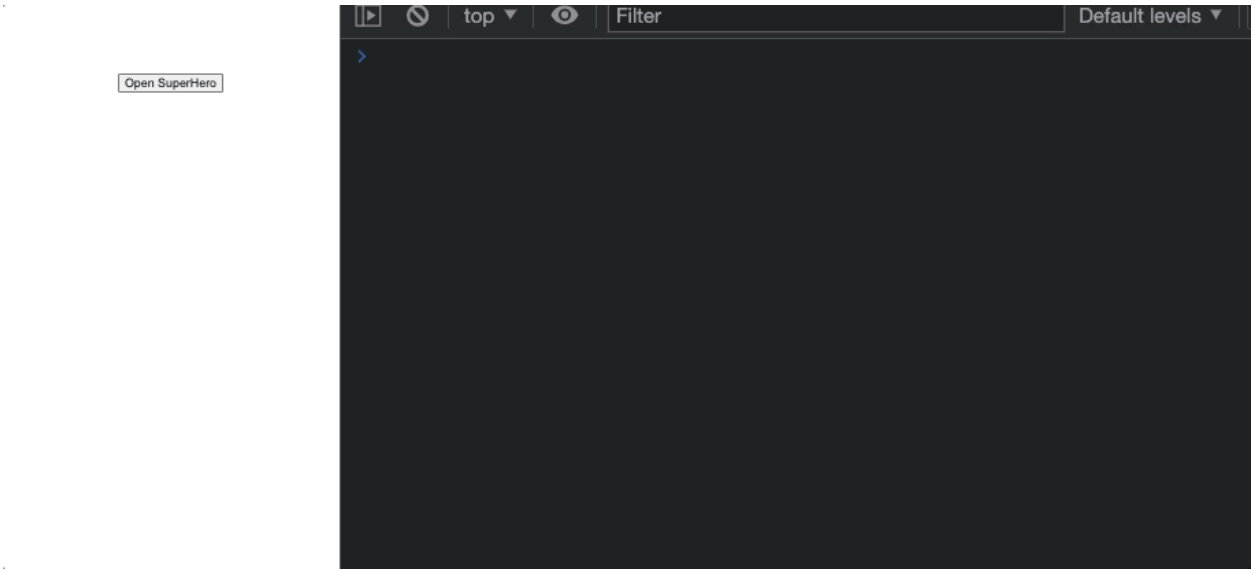
Code

Creamos un componente llamado `CodeEffectUpdate.ts` y añadimos dos inputs en los que podamos cambiar el texto.

```
import { useState, useEffect } from "react";

export const MessageComponent = () => {
  const [myInfo, setMyInfo] = useState({
    name: "Peter",
    lastName: "Parker",
  });

  useEffect(() => {
    console.log("Llamado después de cada Render");

    // ¿Ocurrirá solo al desmontar el componente? 👩‍🎨
    return () => console.log("Desmonto el componente");
  });

  return (
    <div>
      <h4>
        {myInfo.name} {myInfo.lastName}
      </h4>
      <input
        type="text"
        value={myInfo.name}
        onChange={(e) => setMyInfo({ ...myInfo, name: e.target.value })}
      />
      <input
        type="text"
        value={myInfo.lastName}
```

```
        onChange={(e) => setMyInfo({ ...myInfo, lastName: e.target.value })}
      />
    </div>
  );
};

export const CodeEffectUpdate = () => {
  const [visible, setVisible] = useState(false);

  return (
    <>
      {visible && <MessageComponent />}
      <button onClick={() => setVisible(!visible)}>Open SuperHero</button>
    </>
  );
};
```

De este modo cada vez que se produzca un cambio en nuestro componente padre solicitamos realizar un nuevo render. Lo curioso es que nos limpia la funcionalidad antes de cada renderizado. Vamos a verlo en la consola de Chrome.



Con esto podrás deducir que el return que hacemos en el useEffect, también llamada función `cleanUp` no solo actúa al desmontar el componente, sino que se ejecuta previamente a la nueva invocación de un useEffect 🤯 ¡De ahí que te diésemos el ejemplo de los sockets y los listeners antes!

Ejemplo de count

```
import { useEffect, useState } from 'react';

const Count = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log('useEffect ran. count is: ', count);
  }, [count]); // 👈 Añade las variables que quieres trackear

  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};

export default Count;
```

Usamos el hook useEffect para esperar a que el estado se actualice en React. Puede agregar las variables de estado que desea rastrear a la matriz de dependencias del enlace y la función que pasa a useEffect se ejecutará cada vez que cambien las variables de estado.