Entendiendo los Hooks de React, ¿Cómo usar useState y useEffect en nuestros componentes?

https://desarrollofront.medium.com/entendiendo-los-hooks-de-react-c%C3%B3mo-usar-usestate-y-useeffect-en-nuestros-componentes-611b9e826dfa

Juan Pujol

En este artículo vas a aprender qué son y para qué sirven los Hooks. La nueva manera de crear componentes de React.

¿Adiós a los Componentes de Clase?

Los hooks (o ganchos en español) son una herramienta relativamente nueva, aparecieron en 2018 con la versión 16.8 de React. A partir de ese momento los componentes funcionales ganaron mucha potencia con la posibilidad de gestionar su propio estado entre otras funcionalidades.
Eso hizo que los componentes de clase, que hasta entonces tenían el monopolio de este tipo de funcionalidades, perdieran su protagonismo.
En cuanto a su uso, los hooks de React no son más que funciones que están conectadas al ciclo de vida de un componente.

El ciclo de vida de un componente de React

En React, cada componente pasa por diferentes fases en su vida, tal y como una persona puede pasar de ser un niño a una persona adulta, para después convertirse en una persona mayor. Los componentes de React pasan por tres fases: Mounting, Updating y Unmounting.

La primera fase, la de Mounting significa que el componente está en proceso de insertar su contenido en el DOM.
La segunda, Updating, se llama cuando el componente está siendo actualizado. La actualización de un componente se produce cuando hay un cambio en el estado del componente o de sus props.
La siguiente fase, la última, es Unmounting, que se llama cuando un componente tiene que ser eliminado del DOM.

Los dos hooks que vamos a ver en este artículo, useState y useEffect, se activan en la fase de Updating. Cuando actualizamos el estado de nuestro componente con useState, la aplicación tiene que volver a pintar el contenido del componente

en pantalla para tener en cuenta el cambio que hemos introducido. Después de haber actualizado el DOM, la aplicación procede a ejecutar useEffect, si está presente en el componente.

Cómo usar el Hook useState

Para que un componente funcional tenga estado propio podemos hacer uso del hook useState.

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

useState() es una función que crea internamente una variable donde podremos almacenar el estado de nuestro componente. Acepta un valor inicial para esa variable y devuelve un array con dos elementos, el valor de la variable y la función para modificarla.
Como el valor devuelto por la función es un array, podemos descomponerlo para acceder a sus elementos de manera individual.

```
const [count, setCount] = useState(0);
```

Aquí tienes un ejemplo práctico:

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

function SimpleCounter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Has hecho click {count} veces</p>
      <button onClick={() => setCount(count + 1)}>
        Hazme click!
      </button>
    </div>
  );
}

ReactDOM.render(<SimpleCounter />, document.getElementById('root'));
```

Puedes añadir tantos useState() como quieras, cada uno de ellos para una variable diferente.
La única condición es que se llame desde un nivel superior de código, no en un bloque.
Es importante saber que cuando llamamos a la función set de un useState(), se sobrescribe el contenido de la variable.

Cómo usar el Hook useEffect

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

Este hook normalmente es usado para la inicialización de variables, llamadas a APIs o para limpiar un componente antes de desmontarlo del DOM.
Es el equivalente funcional a componentDidMount, componentDidUpdate, y componentWillUnmount en los componentes de clase.

La llamada a useEffect acepta una función como argumento. Esta función se ejecuta por defecto cuando el componente se renderiza por primera vez, y después cada vez que el componente se actualice.
Ejemplo:

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

function CompoundCounter() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `Has hecho click ${count} veces`;
  });

  return (
    <div>
      <p>Has hecho click {count} veces</p>
      <button onClick={() => setCount(count + 1)}>
        Hazme click!
      </button>
    </div>
  );
}

ReactDOM.render(<CompoundCounter />, document.getElementById('root'));
```

También es posible especificar cuándo se debe ejecutar esta función con un segundo argumento opcional que le podemos pasar.

Para ello basta con añadir un segundo parámetro a la función, con la lista de los elementos de los que depende. Si el valor de uno de estos elementos que hemos indicado cambia, la función se va a ejecutar con la siguiente actualización.

```
useEffect(() => {
  document.title = `Has hecho click ${count} veces`;
}, [count]);
```

Otra posibilidad que nos permite este hook es la de especificar que se ejecute sólo una vez. Esto resulta muy útil si sólo queremos hacer una llamada AJAX para rellenar el estado de la aplicación.

```
useEffect(() => {
  async function fetchCount() {
    const response = await fetch(url);
    const remoteCount = await response.json();
    setCount(remoteCount);
  }
  fetchCount();
}, []);
```

A veces, hacemos uso de recursos que pueden quedar a medias cuando el componente se desmonta del DOM, para hacer “limpieza” de los recursos de un componente podemos especificarlo devolviendo una función en useEffect:

```
useEffect(() => {
  // Contenido de useEffect
  return () => {
    // Limpieza del componente
  };
});
```