

ReactJS Developer

Módulo 5



Introducción a *Hooks*

Hooks

Hooks es una nueva característica en **React 16.8**.

Te permiten usar el estado y otras características de React **sin escribir una clase**. Básicamente los *Hooks* son una forma de **reutilizar la lógica de estado de un componente**.

En React, un *Hook* es **una función que empieza con use**. React ofrece *Hooks* o “ganchos” a diversas funcionalidades. También puedes crear los tuyos propios.

Se dice que los *Hooks* son ganchos porque nos permiten **conectar un componente React con diversas funcionalidades extras**.



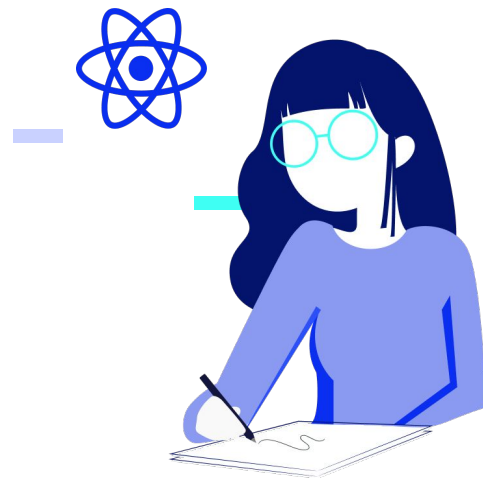
Dos reglas para el uso de hooks

1. **Llama *Hooks* solo en el nivel superior:**
no los llames dentro de ciclos, condicionales o funciones anidadas. En cambio, **usa siempre *Hooks* en el nivel superior de tu función en React, antes de cualquier retorno prematuro.** Siguiendo esta regla, te aseguras de que los *hooks* se llamen en el mismo orden cada vez que un componente se renderiza. Esto es lo que permite a React preservar correctamente el estado de los *hooks* entre múltiples llama- dos a **useState** y **useEffect** (para los más curiosos, vamos a explicar esto más en detalle en la próxima slide).
2. **Llama *Hooks* solo en funciones de React:**
no llames *Hooks* desde funciones JavaScript regulares. En vez de eso, puedes:
 - Llamar *Hooks* desde componentes funcionales de React.
 - Llamar *Hooks* desde *Hooks* personalizados (aprenderemos acerca de ellos más adelante).

Siguiendo esta regla, te aseguras de que toda la lógica del estado de un componente sea claramente visible desde tu código fuente.

UseState

El *Hook* **useState** permite **acceder al estado de React**. Se utiliza para crear **variables de estado**, es decir, variables que, al cambiar, ocasionan una actualización de los componentes relacionados.



Características de *useState*

- Esta función crea **una variable de estado**.
- Recibe **el valor por defecto de dicha variable**.
- Retorna **un array con estos elementos**:
 - **En la posición 0**, se encuentra **una variable JS para leer esa porción del estado**.
 - **En la posición 1**, se encuentra **una función JS para escribir esa porción del estado**, generando también los re-render correspondientes.

```
1  import { useState } from "react";
2
3  function Counter(props) {
4    const [count, setCount] = useState(0);
5
6    return (
7      <div onClick={() => setCount(count + 1)}>
8        Has hecho click {count} veces
9      </div>
10    );
11  }
12
13  export default Counter;
14
```

UseEffect

El *Hook* de efecto te permite llevar a cabo **efectos secundarios en componentes funcionales**.

Peticiones de datos, establecimiento de suscripciones y actualizaciones manuales del *DOM* en componentes de React son ejemplos de efectos secundarios.

Hay dos tipos de efectos secundarios en los componentes de React: aquellos que **necesitan una operación de saneamiento** y los que **no la necesitan**.

El *saneamiento* es **la operación de limpieza necesaria para este efecto**. Hay efectos que **no necesitan limpieza y efectos que sí**. En otras palabras: el saneamiento son **operaciones que ejecuta React al desmontar el componente para liberar recursos**.



- Efectos **sin saneamiento**: a veces, queremos ejecutar código adicional *después* de que React haya actuali- zado el *DOM*. Peticiones de red, mutaciones manuales del *DOM* y registros, son ejemplos de efectos que no requieren una acción de saneamiento. Decimos esto porque podemos **ejecutarlos y olvidarnos de ellos inmediatamente**.
- Efectos **con saneamiento**: por ejemplo, si queremos **establecer una suscripción a alguna fuente de datos externa**.




```
1  import { useEffect, useState } from "react";
2
3  function Counter(props) {
4    const [count, setCount] = useState(0);
5
6    useEffect(() => {
7      fetch('/counter')
8        .then(r => r.json())
9        .then(d => setCount(d.count))
10    });
11
12    return (
13      <div onClick={() => setCount(count + 1)}>
14        Has hecho click {count} veces
15      </div>
16    );
17  }
18
19  export default Counter;
20
```

```
1  import { useEffect, useState } from "react";
2
3  function Counter(props) {
4    const [count, setCount] = useState(0);
5
6    useEffect(() => {
7      subscribeCounter
8        .then(d => setCount(d.count))
9
10     return async function cleanup()
11     {
12       await unsubscribeCounter();
13     }
14   })
15
16   return (
17     <div onClick={() => setCount(count + 1)}>
18       Has hecho click {count} veces
19     </div>
20   );
21 }
22
23 export default Counter;
```

¿Qué hace el hook `useEffect`?

Con este *hook*, le estamos indicando a React que **el componente tiene que hacer algo después de renderizarse**. React **recordará la función que le hemos pasado** (nos vamos a referir a ella como nuestro “efecto”) **y la llamará más tarde después de actualizar el DOM**.

En este efecto, actualizamos el título del documento, pero también podríamos hacer peticiones de datos o invocar alguna *API* imperativa.



¿Por qué usar `useEffect` dentro del componente?

Poner **`useEffect`** dentro del componente nos permite **acceder a la variable de estado `count`** (o a cualquier prop) directamente desde el efecto. No necesitamos una *API* especial para acceder a ella, ya que se encuentra **en el ámbito de la función**.

Los *hooks* aprovechan los *closures* de JavaScript y evitan introducir *APIs* específicas de React donde JavaScript ya proporciona una solución.

¿Se ejecuta *useEffect* después de cada renderizado?

¡Sí! Por defecto se ejecuta **después del primer renderizado y después de cada actualización**. Más tarde explicaremos cómo modificar este comportamiento.

En vez de pensar en términos de “montar” y “actualizar”, puede resultarte más fácil pensar en efectos que ocurren **“después del renderizado”**. React se asegura de que el *DOM* se ha actualizado *antes* de llevar a cabo el efecto.



UseContext

El *hook* **useContext** es una forma programática de acceder al valor pasado por el **Provider** de dicho context más cercano en el árbol:

```
1  import { useContext } from "react";
2  import ThemeContext from "../context/ThemeContext";
3
4  function ThemedText(props) {
5      const theme = useContext(ThemeContext);
6
7      return (
8          <p style={{ color: theme.foreColor }}>{props.children || props.text}</p>
9      );
10 }
11
12 export default ThemedText;
```

**¡Sigamos
trabajando!**

