Usando el Hook de estado

Prueba la nueva documentación de React.
Estas nuevas páginas de la documentación enseñan React moderno e incluyen ejemplos interactivos:

- Estado: la memoria de un componente
- useState

La nueva documentación reemplazará próximamente este sitio, que será archivado. Deja tu opinión aquí

Los Hooks son una nueva incorporación en React 16.8. Te permiten usar estado y otras características de React sin escribir una clase.

La página de introducción utilizó este ejemplo para familiarizarte con los Hooks:

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

function Example() {
  // Declaración de una variable de estado que llamaremos "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Empecemos aprendiendo sobre los Hooks comparando este código con uno equivalente en una clase.

Ejemplo equivalente en forma de clase

Si has usado clases en React previamente, este código te resultará familiar:

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

El estado empieza como `{ count: 0 }` y se incrementa `state.count` cuando el usuario hace click a un botón llamando a `this.setState()`. Usaremos fragmentos de esta clase en toda la página.

Nota

Puedes estar preguntándote por qué estamos usando un contador en lugar de un ejemplo más realista. Esto es porque ayuda a centrarse en la API mientras seguimos dando nuestros primeros pasos con los Hooks.

Hooks y componentes de función

Como recordatorio, un componente de función en React se ve así:

```
const Example = (props) => {
  // Puedes usar Hooks aquí!
  return <div />;
}
```

o así:

```
function Example(props) {
  // Puedes usar Hooks aquí!
  return <div />;
}
```

Puedes haber conocido previamente estos componentes como "componentes sin estado". Actualmente estamos presentando la habilidad de usar el estado de React desde ellos por lo que preferimos el nombre "componentes de función".

Los Hooks **no** funcionan en clases, pero los puedes usar en lugar de escribir clases.

¿Qué es un Hook?

Nuestro nuevo ejemplo empieza importando el Hook `useState` desde React:

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

function Example() {
  // ...
}
```

¿Qué es un Hook? Un Hook es una función especial que permite "conectarse" a características de React. Por ejemplo, `useState` es una clase Hook que te permite añadir el estado de React a un componente de función. Más adelante hablaremos sobre otros Hooks.

¿Cuándo debería usar un Hook? Si crees un componente de función y descubres que necesitas añadirle estado, antes había que crear una clase. Ahora puedes usar un Hook dentro de un componente de función existente. ¡Vamos a hacerlo ahora mismo!

Nota:

Hay algunas reglas especiales sobre dónde puedes y no puedes usar Hooks dentro de un componente. Las aprenderemos en Reglas de los Hooks.

Declarando una variable de estado

En una clase, inicializamos el estado `count` a `0` estableciendo `this.state.a` a `{ count: 0 }` en el constructor:

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
```

En un componente de función no existe `this` por lo que no podemos asignar o leer `this.state`. En su lugar, usamos el Hook `useState` directamente dentro de nuestro componente:

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

function Example() {
  // Declaración de una variable de estado que llamaremos "count"
  const [count, setCount] = useState(0);
```

¿Qué hace la llamada a `useState`? Declara una "variable de estado". Nuestra variable se llama `count`, pero podemos llamarla como queramos, por ejemplo `banana`. Esta es una forma de "preservar" algunos valores entre las llamadas de la función - `useState` es una nueva forma de usar exactamente las mismas funciones que `this.state` nos da en una clase. Normalmente, las variables "desaparecen" cuando se sale de la función, pero las variables de estado son conservadas por React.

¿Qué pasamos a `useState` como argumento? El único argumento para el Hook `useState()` es el estado inicial. Al contrario que en las clases, el estado no tiene por qué ser un objeto. Podemos usar números o strings si es todo lo que necesitamos. En nuestro ejemplo, solamente queremos un número para contar el número de clicks del usuario, por eso pasamos `0` como estado inicial a nuestra variable. (Si queremos guardar dos valores distintos en el estado, llamaríamos a `useState()` dos veces).

¿Qué devuelve `useState`? Devuelve una pareja de valores: el estado actual y una función que lo actualiza. Por eso escribimos `const [count, setCount] = useState()`. Esto es similar a `this.state.count` y `this.setState` en una clase, excepto que se obtienen juntos. Si no conoces la sintaxis que hemos usado, volveremos a ella al final de esta página.

Ahora que sabemos qué hace el Hook `useState`, nuestro ejemplo debería tener más sentido:

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

function Example() {
  // Declaración de una variable de estado que llamaremos "count"
  const [count, setCount] = useState(0);
```

Declaramos una variable de estado llamada `count` y le asignamos a `0`. React recordará su valor actual entre re-renderizados, y devolverá el valor más reciente a nuestra función. Si se quiere actualizar el valor de `count` actual, podemos llamar a `setCount`

Nota:

Puedes estar preguntándote ¿Por qué `useState` no se llama `createState`?

"Crear" no sería del todo correcto porque el estado solamente se crea la primera vez que nuestro componente se renderiza. Durante los siguientes renderizados, `useState` nos da el estado actual. Esta es también la razón por la que los nombres de los Hooks siempre empiezan con `use`. Aprenderemos sobre ello más adelante Reglas de Hooks.

Leyendo el estado

Cuando queremos mostrar el valor actual de `count` en una clase lo obtenemos de `this.state.count`:

```
<p>You clicked {this.state.count} times</p>
```

En una función podemos usar `count` directamente:

```
<p>You clicked {count} times</p>
```

Actualizando el estado

En una clase, necesitamos llamar a `this.setState()` para actualizar el estado `count`:

```
<button onClick={() => this.setState({ count: this.state.count + 1 })}>
  Click me
</button>
```

En una función ya tenemos `setCount` y `count` como variables, así que no necesitamos `this`:

```
<button onClick={() => setCount(count + 1)}>
  Click me
</button>
```

Resumen

Ahora **recapitularemos lo que hemos aprendido línea por línea** y comprobaremos si lo hemos entendido.

```
(1)  import React, { useState } from 'react';
(2)
(3)  function Example() {
(4)    const [count, setCount] = useState(0);
(5)
(6)    return (
(7)      <div>
(8)        <p>You clicked {count} times</p>
(9)        <button onClick={() => setCount(count + 1)}>
(10)         Click me
(11)       </button>
(12)     </div>
(13)   );
(14) }
```

- **Línea 1:** Importamos el Hook `useState` desde React que nos permite mantener un estado local en un componente de función.
- **Línea 4:** Dentro del componente `Example` declaramos una nueva variable de estado llamando al Hook `useState`. Este nos devuelve un par de valores, a los que damos un nombre. Llamamos `count` a nuestra variable porque guarda el número de clicks en el botón. La inicializamos a cero pasando `0` como único argumento a `useState`. El segundo elemento retornado es una función que nos permite actualizar `count`, por lo que le llamamos `setCount`.
- **Línea 9:** Cuando el usuario hace click, llamamos a `setCount` con un nuevo valor. React actualizará entonces el componente `Example` pasándole el nuevo valor de `count`.

Esto puede parecer mucho para empezar. *No tengas prisa!* Si te pierdes con esta explicación repasa el código de arriba y trata de leerlo de arriba hacia abajo. Prometemos que una vez trates de "olvidar" como funciona el estado en las clases y mires a este código con la mente despejada cobrará sentido.

Tip: ¿Qué significan los corchetes?

Habrás observado los corchetes cuando declaramos una variable de estado:

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

Los nombres de la izquierda no son parte de la API de React. Puedes nombrar tus variables de estado como quieras:

```
const [fruit, setFruit] = useState('banana');
```

Esta sintaxis de Javascript se llama "desestructuración de arrays". Significa que estamos creando dos variables `fruit` y `setFruit`, donde `fruit` se obtiene del primer valor devuelto por `useState` y `setFruit` es el segundo. Es equivalente a este código:

```
var fruitStateVariable = useState('banana'); // Returns a pair
var fruit = fruitStateVariable[0]; // First item in a pair
var setFruit = fruitStateVariable[1]; // Second item in a pair
```

Cuando declaramos una variable de estado con `useState`, devuelve un array con dos elementos. El primero es el valor actual y el segundo es una función que nos permite actualizarlo. Usar `[0]` y `[1]` para acceder a ellos es un poco confuso porque tienen un significado específico. Por ello usamos la desestructuración de arrays en su lugar.

Nota

Puedes tener curiosidad por cómo React sabe a qué componente corresponde `useState` ya que no estamos pasando algo como `this` a React. Responderemos esta pregunta y muchas otras en la sección FAQ.

Tip: Usando múltiples variables de estado

Declarando variables de estado como un par `[something, setSomething]` también es útil porque nos permite dar diferentes nombres a diferentes variables de estados si queremos usar más de una.

```
function ExampleWithManyStates() {
  // Declare multiple state variables!
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
```

En el componente de arriba tenemos `age`, `fruit`, y `todos` como variables locales y los podemos actualizar de forma individual:

```
  function handleOrangesClick() {
    // Similar a this.setState({ fruit: 'orange' })
    setFruit('orange');
  }
```

No tienes que usar obligatoriamente tantas variables de estado: las variables de estado pueden contener objetos y arrays para que puedas agrupar la información relacionada. Sin embargo, al contrario que en una clase, actualizar una variable de estado siempre lo reemplaza en lugar de combinarla.