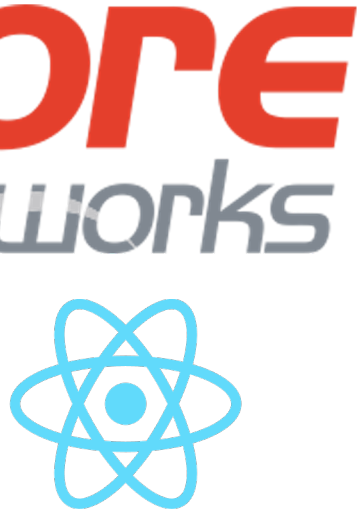
Desarrollo y programación de Aplicaciones con React

5. Componentes de tipo función



Componentes de función

Sintaxis

Los componentes funcionales son una función JavaScript , de hecho el componente raiz App se genera por defecto como componente funcional. Tiene las siguientes condiciones:

- Usa double camel case en su identificador.
- Devuelve un elemento escrito en JSX.
- Se suelen escribir en un archivo y se exportan como default o named.
- Respecto a los componentes de clase usa una sintaxis diferente para props y estado.

Componentes de función

Sintaxis

```
import React from 'react'

export default function NombreCommponente(props) {

    const [propiedad, setPropiedad] = useState("valor-inicialización-propiedad");

    return (
        <>
            {/* JSX */}
        </>
    )
}
```

Componentes de función

Props

Las props en los componentes funcionales son similares a los de clase, la diferencia radica en que tienen que ser pasadas como parámetros en la declaración del componente.

```
export default function NombreComponente(props) {
...
}
```

Componentes de función

Estado

El funcionamiento del estado en los componentes funcionales es también similar pero cambia bastante la sintaxis de declaración y actualización.

En este caso, se gestiona con un hook predefinido en React llamado useState()

*Todos los hooks comienzan por use (tanto propios de React como personalizados).

Componentes de función

useState() Declaración

Lo que hace useState es devolver un array con la propiedad del objeto de estado y su método de seteo a partir de un valor de inicialización (que puede ser de cualquier tipo de dato), por eso utilizamos desestructuring en su declaración.

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

```
const [propiedad, setPropiedad] = useState(estadoInicial);
```

Con esa declaración tendremos la propiedad del estado disponible para lectura y una función para “setear” su valor.

Componentes de función

useState() Declaración

Si el estado inicial, que solo se ejecuta una vez, establece su valor como consecuencia de un proceso lógico, se puede pasar una callback para definirlo.

```
const [propiedad, setPropiedad] = useState((parámetros) => {
        // Lógica de inicialización
        return estadoInicial;
});
```

Componentes de función

useState() Declaración

Como es muy probable que el objeto de estado tenga varias propiedades, podemos usar useState() tantas veces como lo necesitemos.

```
const [propiedad1, setPropiedad1] = useState(<valor-inicialización-propiedad1>);
const [propiedad2, setPropiedad2] = useState(<valor-inicialización-propiedad2>);
```

Componentes de función

useState() Invocación

Como ocurre en los componentes de clase, las propiedades del estado no pueden ser modificadas directamente si no que tenemos que usar su setPropiedad() que tiene la siguiente sintaxis:

setPropiedad(nuevoEstadoPropiedad); // Mismo tipo de la propiedad

Componentes de función

useState() Invocación

En el caso que se tenga que usar el estado anterior, se puede pasar una función callback de una manera similar a los componentes de clase: con una expresión que devuelva el valor modificado.

```
setPropiedad(prevPropiedad => prevPropiedad + 1);
```

Componentes de función

useState() Invocación

Si la propiedad a actualizar es un objeto con múltiples subniveles, estas funciones set no combinan las propiedades, hay que introducirlas de nuevo.

```
setObjeto(prevObjeto => {
        return {prevObjeto.propiedad + 1, ...rest}
});
```

Componentes de función

Ciclo de vida

El ciclo de vida de los componentes en React permitía en nuestros componentes con class poder ejecutar código en diferentes fases de montaje, actualización y desmontaje. De esta forma, podíamos añadir cierta funcionalidad en las distintas etapas de nuestro componente.

Componentes de función

Ciclo de vida

Con los hooks también podremos acceder a esa ciclo de vida en nuestros componentes funcionales aunque de una forma más clara y sencilla. Para ello usaremos useEffect, un hook que recibe como parámetro una función que se ejecutará cada vez que nuestro componente se renderice, ya sea por un cambio de estado, por recibir props nuevas o, y esto es importante, porque es la primera vez que se monta.

Componentes de función

Ciclo de vida

Por tanto, useEffect es una combinación de los métodos de actualización para clases combinado y que es llamado justo después de renderizarse el componente por primera vez e inicializar el estado y, posteriormente, cada vez que se actualice el estado que será lo mismo que tras ejecutarse useState().

Componentes de función

useEffect() Invocación

Este hook recibe una callback como primer argumento que será ejecutada con cualquier actualización del estado o props del componente y un segundo argumento opcional con un array de referencias.

useEffect(callback, [referencias])

Componentes de función

useEffect() Invocación

Este hook recibe una callback como primer argumento que será ejecutada con cualquier actualización del estado o props del componente y un segundo argumento opcional con un array de referencias.

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

useEffect(callback, [referencias]?);
```

Componentes de función

useEffect() Invocación

La callback del primer argumento permite definir el código con la lógica de actualización, pudiendo y siendo habitual, ser utilizada para setear un nuevo estado o leer una prop. También de manera opcional puede finalizar con una instrucción con return y otra callback que permita realizar labores de limpieza (por ejemplo subscripciones o timers) para evitar *memory leaks*.

```
useEffect(() => {
        // Lógica con estados y propiedades
        return () => // Lógica de limpieza de memoria (opcional)
});
```

Componentes de función

useEffect() Invocación

El segundo argumento opcional permite limitar la ejecución de la callback a cuando se produzcan actualizaciones en las variables que se introduzcan en el array.

```
useEffect(() => {
    // Ejecución de código solo cuando se modifique alguna de las variables
}, [variable1, variable2, ...]);
```

Si este argumento no es pasado, la callback se actualizará con todos los cambios que se produzcan en el componente.

Componentes de función

useEffect() Invocación

Si el segundo argumento es pasado como un array vacío, la callback solo se ejecutará en el inicio y fin del ciclo de vida del componente.

```
useEffect(() => {
      // Ejecución de código solo en al montar y desmontar del componente;
}, []);
```

Al poder ser referencia useEffect() puede usarse tantas veces necesitemos.

Componentes de función

Referencias a elementos del DOM

Para tener referencias a elementos del DOM de cara a su manipulación programática, por ejemplo para añadir o eliminar clases CSS, podemos usar el hook useRef().

Componentes de función

useRef() Declaración

Este hook puede usarse para varios usos, pero en general implementa una referencia al objeto del elemento HTML marcado con el atributo ref con la siguiente sintaxis.

```
import React, { useRef } from 'react’;

const objetoReferencia = useRef();

<elemento ref={objetoReferencia} />
```

Componentes de función

useRef() Uso del objeto referenciado.

El objeto referenciado puede usarse con su propiedad current para acceder al elemento y poder manipularlo con los métodos y propiedades nativos de JavaScript.

```
const handleElemento = () => {
    objetoReferencia.current.metodosDOMJS...
}
```

*Una forma de recordar el uso es que hasta current el objeto es similar al clásico document.getElementById().