educación IT
CODE YOUR FUTURE

ReactJS Developer

Módulo 5

Context API

Context API

En una aplicación típica de React, los datos se pasan **de arriba hacia abajo** (de padre a hijo) a través de `props`, pero esta forma puede resultar incómoda para ciertos tipos de `props` (como por ejemplo, localización, el tema de la interfaz) que son necesarias para muchos componentes dentro de una aplicación.

`Context` nos proporciona una forma de compartir valores como estos entre componentes **sin tener que pasar explícitamente una *prop* a través de cada nivel del árbol.**

Context está diseñado para compartir datos que pueden considerarse “globales” para un árbol de componentes en React, como el usuario autenticado actual, el tema o el idioma preferido.

Context se usa principalmente cuando **algunos datos tienen que ser accesibles por muchos componentes en diferentes niveles de anidamiento.** Aplícalo con moderación porque hace que la reutilización de componentes sea más difícil. Si solo deseas evitar pasar algunos props a través de muchos niveles, l**a composición de componentes suele ser una solución más simple que Context.**

Crear el contexto

Crea un objeto `Context`: cuando React renderiza un componente que se suscribe a este objeto `Context`, este leerá el valor de contexto actual del **`Provider` más cercano en el árbol.**

El argumento **`defaultValue`** es usado únicamente cuando un componente **no tiene un `Provider` superior a él en el árbol.** Este valor por defecto puede ser útil para probar componentes de forma aislada sin contenerlos.

Nota: pasar `undefined` como valor al `Provider` no hace que los componentes que lo consumen utilicen **`defaultValue`.**

```
1  import React from "react";
2
3  const ThemeContext = React.createContext(/*defaultValue*/);
4
5  export default ThemeContext;
6
```

Provider

Cada objeto `Context` viene con un componente **`Provider`** de React que **permite que los componentes que lo consumen se suscriban a los cambios del contexto.**

El componente `Provider` **acepta una prop `value` que se pasará a los componentes consumidores que son descendientes de este `Provider`.**

Un `Provider` puede estar conectado **a muchos consumidores.** Los `Providers` pueden estar anidados para sobreescribir los valores más profundos dentro del árbol.

Todos los consumidores que son descendientes de un `Provider` se vuelven a renderizar **cada vez que cambia la prop `value` del `Provider`.**

La propagación del `Provider` a sus consumidores descendientes (incluyendo `.contextType` y `useContext`) **no está sujeta al método `shouldComponentUpdate`,** por lo que el consumidor se actualiza **incluso cuando un componente padre evita la actualización.**

Los cambios se determinan comparando los valores nuevos y antiguos utilizando **el mismo algoritmo que `Object.is`.**

Todos los consumidores que son descendientes de un `Provider` se vuelven a renderizar **cada vez que cambia la prop value del `Provider`.**

La propagación del `Provider` a sus consumidores descendientes (incluyendo `.contextType` y `useContext`) **no está sujeta al método**

`shouldComponentUpdate`, por lo que el consumidor se actualiza **incluso cuando un componente padre evita la actualización.**

Los cambios se determinan comparando los valores nuevos y antiguos utilizando **el mismo algoritmo que `Object.is`.**

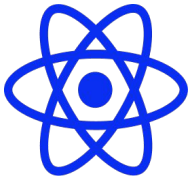
```
1  import ThemeContext from "./ThemeContext";
2
3  function ThemeProvider(props) {
4      return <ThemeContext.Provider value={props.theme} {...props} />;
5  }
6
7  export default ThemeProvider;
```

```
1   import ThemeProvider from "./context/ThemeProvider";
2   import Layout from "./Layout";
3
4   function App(props)
5   {
6       return (
7           <ThemeProvider theme={{ foreColor: 'black' }}>
8               <Layout>
9                   <div>Contenido principal</div>
10              </Layout>
11          </ThemeProvider>
12      )
13  }
14
15  export default App;
```

Consumer

Es un componente de React que **se suscribe a cambios de contexto.** Al usar este componente puedes **suscribirte a un contexto dentro de un componente de función.**

`Consumer` requiere una función como hijo: la función **recibe el valor de contexto actual y devuelve un nodo de React.** El argumento `value` pasado a la función **será igual a la `prop value` del `Provider` más cercano para este contexto en el árbol.** Si no hay un proveedor superior para este contexto, el argumento `value` será igual al **`defaultValue`** que se pasó a **`createContext()`.**



```
1  import ThemeContext from "./context/ThemeContext";
2
3  function ThemedText(props) {
4      return (
5          <ThemeContext.Consumer>
6              {(value) => {
7                  <p style={{ color: value.foreColor }}>{props.children || props.text}</p>;
8              }}
9          </ThemeContext.Consumer>
10     );
11 }
12
13 export default ThemedText;
14
```

```jsx
import ThemeProvider from "./context/ThemeProvider";
import Layout from "./Layout";
import ThemedText from "./ThemedText";

function App(props)
{
    return (
        <ThemeProvider theme={{ foreColor: 'red' }}>
            <Layout>
                <div>
                    <ThemedText>Texto en color rojo</ThemedText>
                </div>
            </Layout>
        </ThemeProvider>
    )
}

export default App;
```

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!