React

HOOKS

- useState
- useEffect
- useContext
- useRef
- useReducer
- useCallback
- useMemo
- Custom Hooks

By Junaid Arshad

React Hooks

Hooks allow us to "hook" into React features such as state and lifecycle methods.

Rules:

- Hooks can only be called inside React function components.
- Hooks can only be called at the top level of a component.
- Hooks cannot be conditional

1. useState

To use the useState Hook, we first need to import it into our component.

Initialize useState

We initialize our state by calling useState in our function component.

useState accepts an initial state and returns two values:

- **The current state.**
- **A function that updates the state.**

Example:

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

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

2. useEffect

- The useEffect Hook allows you to perform side effects in your components.
- Some examples of side effects are: fetching data, directly updating the DOM, and timers.
- useEffect accepts two arguments. The second argument is optional.
- useEffect(<function>, <dependency>)

Example:

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

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

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

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

3. useContext

- React Context is a way to manage state globally.
- It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

Example:

```
const themes = {
  light: {
    foreground: "#000000",
    background: "#eeeeee"
  },
  dark: {
    foreground: "#ffffff",
    background: "#222222"
  }
};

const ThemeContext = React.createContext(themes.light);

function App() {
  return (
    <ThemeContext.Provider value={themes.dark}>
      <Toolbar />
    </ThemeContext.Provider>
  );
}

function Toolbar(props) {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

function ThemedButton() {
  const theme = useContext(ThemeContext);
  return (
    <button style={{ background: theme.background, color: theme.foreground }}>
      I am styled by theme context!
    </button>
  );
}
```

4. useRef

- The useRef Hook allows you to persist values between renders.
- It can be used to store a mutable value that does not cause a re-render when updated.
- It can be used to access a DOM element directly.

Example:

```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` points to the mounted text input element
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </>
  );
}
```

5. useReducer

- The useReducer Hook is similar to the useState Hook.
- It allows for custom state logic.
- The reducer function contains your custom state logic and the initialStatecan be a simple value but generally will contain an object.
- The useReducer Hook returns the current stateand a dispatchmethod.

Example:

```
const initialState = {count: 0};

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
    </>
  );
}
```

6. useCallback

- **The React useCallback Hook returns a memoized callback function.**
- This allows us to isolate resource intensive functions so that they will not automatically run on every render.
- The useCallback Hook only runs when one of its dependencies update.
- useCallback is a React Hook that lets you cache a function definition between re-renders.

Example:

```
import { useCallback } from 'react';

export default function ProductPage({ productId, referrer, theme }) {
  const handleSubmit = useCallback((orderDetails) => {
    post('/product/' + productId + '/buy', {
      referrer,
      orderDetails,
    });
  }, [productId, referrer]);
```

7. useMemo

- **The React useMemo Hook returns a memoized value.**
- **The useMemo Hook only runs when one of its dependencies update.**
- ***The useMemo and useCallback Hooks are similar. The main difference is that useMemo returns a memoized value and useCallback returns a memoized function.***

Example:

```
import { useMemo } from 'react';

function TodoList({ todos, tab }) {
  const visibleTodos = useMemo(
    () => filterTodos(todos, tab),
    [todos, tab]
  );
  // ...
}
```

8. Custom Hook

- Hooks are reusable functions.
- When you have component logic that needs to be used by multiple components, we can extract that logic to a custom Hook.
- Custom Hooks start with "use".

  Example: useFetch.

Example:

```
import {useState , useEffect} from "react";

// Remember to start the name of your custom hook with "use"
function useCustomHook(initializer , componentName){
    const [counter , setCounter] = useState(initializer);

    // Increases the value of counter by 1
       function resetCounter(){
        setCounter(counter + 1);
    }


    useEffect(() => {
        // Some logic that will be used in multiple components
        console.log("The button of the "
        + componentName + " is clicked "
        + counter + " times.");
    } , [counter , componentName]);

    // Calls the useEffect hook if the counter updates
    return resetCounter;
}

export default useCustomHook;
```

Thanks

Don’t forget Like and Follow

By Junaid Arshad