React Hooks

27 December 2022 03:33 PM

Hooks are the new feature introduced in the React 16.8 version.

It allows you to use state and other React features(lifecycle method) without writing a class.

Hooks are backward-compatible, which means it does not contain any breaking changes. Also, it does not replace your knowledge of React concepts.

When to use a Hooks

If you write a function component, and then you want to add some state to it, previously you do this by converting it to a class. But, now you can do it by using a Hook inside the existing function component.

Hook Rules

There are 3 rules for hooks:

- 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

React useState Hook

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

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.

React useEffect Hooks

The Effect Hook allows us to perform side effects (an action) in the function components.

Effects Hooks are equivalent to componentDidMount(), componentDidUpdate(), and componentWillUnmount() lifecycle methods.

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>)

1. No dependency passed:

```
useEffect(()=>{

  //Runs on every render

});
```

2. An empty array:

```
useEffect(()=>{

  //Runs only on the first render
},[]);
```

3. Props or state values:

```
useEffect(()=>{

   //Runs on the first render

   //And any time any dependency value changes
},[prop,state]);
```

React useContext Hook

```
constUserContext =createContext()

<UserContext.Providervalue={user}>


Const user =useContext(UserContext);
```