Full Stack Software Development

Course : Advanced Frontend Development Using React

Lecture On : React Hooks - Part I

upGrad

In the last class, you learnt

- Class-based vs functional components
- Why use Hooks when class-based components serve the same purpose?
- Using state in functional components
- Routing using Hooks

Poll 1

Which React Hook is used to maintain state variables within a component?

A. useRedux

B. useEffect

C. useState

D. useStateVar

Poll 1 (Answer)

Which React Hook is used to maintain state variables within a component?

A. useRedux

B. useEffect

C. useState

D. useStateVar

Poll 2

Which of the following React Router Hooks is used to retrieve key-value pairs of URL parameters?

A. useParams

B. useLocation

C. useURLParams

D. useQueryParams

Poll 2 (Answer)

Which of the following React Router Hooks is used to retrieve key-value pairs of URL parameters?

A.   useParams

B.   useLocation

C.   useURLParams

D.   useQueryParams

Today's Agenda

1. Apply useState() Hook
   - Multiple State Variables

2. Apply useEffect() Hook
   - Side effects
   - Cleanup in side effects
   - React lifecycle Hooks

3. Apply useContext() Hook

Today's Agenda

1. Apply useState() Hook
   - Multiple State Variables

2. Apply useEffect() Hook
   - Side effects
   - Cleanup in side effects
   - React lifecycle Hooks

3. Apply useContext() Hook

useState Hook

You have already seen how to use useState in the last session.

What does calling useState do?

It declares a 'state variable'. The value of this variable is preserved across function calls, that is, re-rendering the component does not reset the value of the variable

Example:

```
const [ myStateVar, setMyStateVar ] = React.useState(null);
```

myStateVar is initialised with the value of null, however, if you now call *setMyStateVar* to update it and the component re-renders, the new value is preserved and it is not reset as null again

Which arguments does useState take?

The useState Hook only takes a single optional argument that initialises the state variable. The value of this variable can belong to any data type, that is, primitive, object or array

Example:

```
const [ userName, setUserName ] = React.useState('Anonymous');
```

It creates a state variable called 'userName' that is initialised with the string value 'Anonymous'

What does useState return?

The useState Hook function call returns an array with two values: the current state and an updater function

Example:

```javascript
const [ name, setName ] = React.useState('Anonymous');
// can also be written as:
const nameState = React.useState('Anonymous');
const name = nameState[0];
const setName = nameState[1];
```

Correct and incorrect ways of updating the values of state variables are as follows:

```
// correct way to update the name
setName('John Doe');  // correct

// incorrect way to update the name
name = 'John Doe';
```

Multiple State Variables

You have seen earlier how you can create multiple state variables by calling the useState Hook multiple times. This keeps the different state variables logically separate

You could obtain an outcome similar to that from the following code by maintaining a single state object, but this is not necessary and might lead to unnecessary grouping of logically separate information:

```javascript
const [state, setState] = React.useState({
  name: 'Anonymous',
  status: 'Hello world!',
  isOnline: false
});

// update only the name
setState({
  ...state,
  name: 'John Doe'
});
```

Multiple State Variables

This pattern makes the state management difficult as the complexity of the component grows and you would need to keep adding more state variables

Hence, it is better to do the following:

```
const [name, setName] = React.useState('Anonymous');
const [status, setStatus] = React.useState('Hello world');
const [isOnline, setOnline] = React.useState(false);
```

Now, to update any single state variable, you need not worry about the other state variables

This makes it easier to add/remove state variables as necessary

Phone Directory

The first step for converting the *PhoneDirectory* component to a functional component would be to apply the useState Hook instead of *this.state*.

```
this.state = {
  subscribersList: [
  // initial state...
  ]
}
```

becomes:

```
const [subscribersList,setSubscribersList] = useState([
  // initial state...
]);
```

[Code Reference](#)

Poll 3

Given the following initial state, what will be the value of the state variable after the update?

```
const [state, setState] = useState({ foo: 'bar' });
setState({ bar: 'baz' });
```

A.   { foo: 'bar', bar: 'baz' }

B.   { bar: 'baz' }

C.   { foo: 'bar' }

D.   null

Poll 3 (Answer)

Given the following initial state, what will be the value of the state variable after the update?

```
const [state, setState] = useState({ foo: 'bar' });
setState({ bar: 'baz' });
```

A.   { foo: 'bar', bar: 'baz' }

B.   { bar: 'baz' }

C.   { foo: 'bar' }

D.   null

Poll 4

Which of the following is true regarding the useState() Hook?

A.   It will return two things, the current state and the method used to update the state.

B.   It will return nothing.

C.   The useState Hook is not a replacement of this.setState().

Poll 4 (Answer)

Which of the following is true regarding the useState() Hook?

A. **It will return two things, the current state and the method used to update the state.**

B. It will return nothing.

C. The useState Hook is not a replacement of this.setState().

useEffect Hook

Side-Effects

With the help of **useEffect** Hook, one can perform 'side-effects' in functional React components.

Examples of side-effects include (but are not limited to):

- Fetching data from an API

- Subscribing to events

- Direct DOM manipulation

Cleanup in Side-Effects

- Moreover, side-effects can be of two types: those that require a cleanup and those that don't require a cleanup

- Effects like network calls, manual DOM manipulations and logging typically do not require cleanup

- On the other hand, a component that subscribes to a particular event as a side-effect will need to remove the event listener when it unmounts as a cleanup

React Lifecycle Hooks

- Earlier, you learnt how to add state management to functional components in React using the useState Hook

- Now, let's take a look at how to implement another feature of class components in pure functional components, that is, lifecycle methods, such as *componentDidMount, componentWillReceiveProps* and *componentWillUnmount*

A Timer Component

Here, let's implement a simple timer component with useEffect:

```javascript
function Timer() {
  const [tick, setTick] = React.useState(0);
  useEffect(() => {
    setInterval(() => setTick(tick + 1), 1000);
  });

  return <div>{tick} seconds have elapsed</div>;
}
```

The result is a component that starts ticking every second on mount

useEffect Hook API

What does useEffect do?

Using the useEffect Hook, you can tell React that your component needs to do something (the 'effect') after render. In the timer example, the following code:

```
useEffect(() => {
  setInterval(() => setTick(tick + 1), 1000);
});
```

tells React that it should start incrementing the tick every second as soon as the component mounts

27

useEffect Hook API

What inputs does useEffect take?

The useEffect Hook takes two inputs:
- A callback function to execute as the effect
- An optional array of dependencies

You could rewrite the timer example as shown below:

```
useEffect(() => {
  setTimeout(() => setTick(tick + 1), 1000);
}, [tick]);
```

This code tells React that it should increment the tick once every second after the tick value changes, resulting in the timer ticking on endlessly

28

When Are Effects Invoked?

- Earlier, you learnt how to add state management to functional components in React using the useState Hook

- Now, let's look at how to implement another feature of class components in pure functional components, that is, lifecycle methods, such as *componentDidMount, componentWillReceiveProps* and *componentWillUnmount*

- Generally, effects run on each update if you do not specify any dependencies. In case you do provide dependencies to the useEffect Hook, then the effect runs each time one or more of the dependencies change

- React uses a shallow equality check to determine if a dependency has changed

Hook for componentDidMount

- If you pass an empty or non-empty array of the dependencies to a useEffect call, it behaves in a similar manner as *componentDidMount*. For example:

```
useEffect(() => {
    // this gets executed on only the first render since the dependency list
is empty
});
```

- You looked an example of a non-empty dependency list in your timer example earlier. Let's recap a similar example:

```
useEffect(() => {
    // this gets executed whenever one of the variables a or b change
}, [a, b]);
```

- Passing a non-empty list of dependencies is an optimisation technique used to prevent unnecessary re-renders

Hook for componentDidUpdate

- If you omit the dependencies to a useEffect call, it behaves in similar manner as *componentDidUpdate*. For example:

```
function ProductList() {
    useEffect(() => {
  // fetch data from API
  fetchProducts();
    });

}
```

- The above component will call the API to load the product list data on every render

Hook for componentWillUnmount

- If you return a function from what you pass to *useEffect*, it is invoked as a cleanup. For example:

```javascript
useEffect(() => {
    window.addEventListener('message', onMessage);
    return () => {
        // cleanup function
        window.removeEventListener('message', onMessage);
    }
}, []);
```

- However, a major difference between this and *componentWillUnmount* is that, in former case, the effect cleanup will be invoked on every update (re-render) and not necessarily just once right before the component unmounts (as in the case of *componentWillUnmount*)

Effect Cleanup

Why does the effect cleanup run on every update?

Consider the following example from a messenger app:

```javascript
function Chat(props) {
    useEffect(() => {
        subscribeToMessages(props.chatId);
        return () => {
            unsubscribeFromMessages(props.chatId);
        }
    }, [props.chatId]);
}
```

The chat window can show only one active chat at a time, which pertains to the *chatId* (of the active conversation)

Effect Cleanup

- Whenever the active chat changes, the *subscribe* and *unsubscribe* functions take a *chatId* that is passed down as a prop to the Chat component

- When the prop changes, you need to unsubscribe from the old *chatId* and subscribe to the new one, so that you can start receiving messages on the currently active conversation by subscribing to the new chatId

- If the cleanup function ran only once before the component unmounted, the prop change would not be accounted for and you would continue to receive messages from the old (inactive) conversation that was first loaded on component mount

Use Multiple Effects to Separate Concerns

- As you learnt earlier in case of the useState Hook, as the complexity of your component grows, using multiple state variables helps separate concerns. Similarly, using multiple effects within the same component helps separate logically unrelated operations. For example:

```
useEffect(() => {
    fetchData();
});

useEffect(() => {
    subscribeToEvents();
    return () => unsubscribeFromEvents();
});
```

- Putting the *fetchData* and *subscribeToEvents* in separate Hooks helps keep them logically separate, unlike *componentDidMount* where these unrelated effects would have been grouped together

Phone Directory

In the *ShowSubscribers* component, you can update the document title with the total number of subscribers whenever the list of subscribers is passed to the component as a prop update, as shown below:

```
useEffect(() => {
    if(subscribersList && subscribersList.length)
        document.title = "Phone Directory - Number of Subscribers " +
subscribersList.length
},[subscribersList]);
```

[Code reference](#)

Poll 5

When will the following effect execute during the component lifecycle?

```
useEffect(() => {
    // do something...
});
```

A.   Once right after the component renders for the first time

B.   Each time the component is re-rendered, except for the first time

C.   Each time the component renders

D.   Never

Poll 5 (Answer)

When will the following effect execute during the component lifecycle?

```
useEffect(() => {
    // do something...
});
```

A. Once right after the component renders for the first time

B. Each time the component is re-rendered, except for the first time

C. Each time the component renders

D. Never

Poll 6

When will the following effect execute during the component lifecycle?

```
useEffect(() => {
    // do something...
}, []);
```

A.   Once right after the component renders for the first time

B.   Each time the component is re-rendered, except for the first time

C.   Each time the component renders

D.   Never

Poll 6 (Answer)

When will the following effect execute during the component lifecycle?

```
useEffect(() => {
    // do something...
}, []);
```

A.   Once right after the component renders for the first time

B.   Each time the component is re-rendered, except for the first time

C.   Each time the component renders

D.   Never

useContext Hook

React Context

Using **useContext** Hook, you can consume the value of a React context

Let's take a look at the following example to understand this concept better:

```javascript
const theme = {
    christmas: { color: 'red' },
    normal: { color: 'black' }
}

const ThemeContext = React.createContext(theme.normal);

function App() {
    return (
        <ThemeContext.Provider value={theme.christmas}>
            <MyComponent />
        </ThemeContext.Provider>
    );
}
```

React Context

Now, you can consume the value of the theme provided within *MyComponent* as shown below:

```
function MyComponent() {
    const theme = React.useContext(ThemeContext);    // in this case,
christmas

    return <div style={{ backgroundColor: theme.color }}>...</div>;
}
```

Note that *useContext* takes the value from the nearest provider. If there were multiple nested providers, the value from the innermost provider (which is an ancestor of the consuming component in the component tree) would be used.

Also note that, in case of no provider, the value would be default value of the context, which is specified when *React.createContext* is invoked.

Phone Directory

In the Phone Directory application, you can use the React context to share information regarding the total number of subscribers with individual components, such as the footer, as shown below:

```
const SubscriberCountContext = React.createContext();

<SubscriberCountContext.Provider value={subscribersList.length}>
  <Footer ></Footer>
</SubscriberCountContext.Provider>
```

[Code reference](#)

Poll 7

Which of the following is the correct syntax for consuming a context value using the React Hook?

A. React.useContext(MyContext.Provider)

B. React.useContext(MyContext.Consumer)

C. React.useContext(MyContext)

D. None of the above

Poll 7 (Answer)

Which of the following is the correct syntax for consuming a context value using the React Hook?

A. React.useContext(MyContext.Provider)

B. React.useContext(MyContext.Consumer)

C. React.useContext(MyContext)

D. None of the above

Poll 8

What will be the value of context variable that is received inside MyComponent?

```
const MyContext = React.createContext(16);
...
<MyContext.Provider={42}>
  <MyContext.Provider={23}>
    <MyComponent />
  </MyContext.Provider>
</MyContext.Provider>
```

A. 42

B. 23

C. 16

D. null

Poll 8 (Answer)

What will be the value of context variable that is received inside MyComponent?

```
const MyContext = React.createContext(16);
...
<MyContext.Provider={42}>
  <MyContext.Provider={23}>
    <MyComponent />
  </MyContext.Provider>
</MyContext.Provider>
```

A.   42

B.   23

C.   16

D.   null

Poll 9

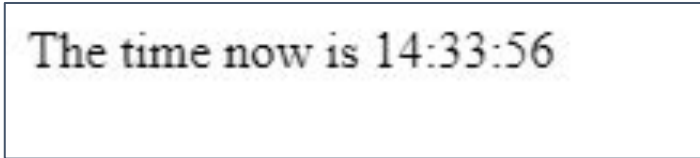Recall the implementation of the useContext Hook in the Phone Directory application and take a look at the code snippet given below.

```
<SubscriberCountContext.Provider value={subscribersList.length}>
    <Footer ></Footer>
</SubscriberCountContext.Provider>
```

Select the correct option among those given below.

A. Placing the Footer inside the SubscriberCountContext allows the Footer component to consume the data generated by SubscriberCountContext.

B. Placing the Footer inside the SubscriberCountContext does not allow the Footer component to consume the data generated by SubscriberCountContext.

Poll 9 (Answer)

Recall the implementation of the useContext Hook in the Phone Directory application and take a look at the code snippet given below.

```
<SubscriberCountContext.Provider value={subscribersList.length}>
    <Footer ></Footer>
</SubscriberCountContext.Provider>
```

Select the correct option among those given below.

A. **Placing the Footer inside the SubscriberCountContext allows the Footer component to consume the data generated by SubscriberCountContext.**

B. Placing the Footer inside the SubscriberCountContext does not allow the Footer component to consume the data generated by SubscriberCountContext.

Implement a clock component that displays the current time in hh:mm:ss 24-hour format and updates every second, as shown in the following screenshot:

The time now is 14:33:56

The stub code is provided here.
The solution code is provided here.

All the code used in today's session can be found in the

link provided below:

https://github.com/upgrad-edu/react-hooks/tree/Session8

Doubt Clearance (5 minutes)

Important Questions

Important Questions

1. Do Hooks replace render props and higher order components?

2. Why do we use array destructuring (square brackets notation) in useState?

3. What is the purpose of useEffect hook?

4. Give an example on how to use context.

5. How do you use contextType?

6. Explain about Context API in React. When should you use it?

Key Takeaways

- The useState Hook is used for state management within a functional React component

- The useState Hook returns a tuple of the live state variable and a function to update it

- The useEffect Hook is used to implement lifecycle methods in functional React components

- Multiple useEffect Hooks can be used within the same component to achieve separation of concerns

- The useContext Hook is used to access React context within functional components

These tasks are to be completed after today's session:

MCQs
Coding Questions
Course Project (Part B) - Checkpoint 1

In the next class, we will discuss

1. More Hooks
   - useMemo
   - useCallback
   - useReducer
2. Implementing Custom Hooks

upGrad
#RahoAmbitious

Thank You!