upGrad

#RahoAmbitious

Full Stack Software Development

upGrad

Course: Advanced Frontend Development Using React

Lecture on: Introduction to React Hooks

In the last class, you learnt:

- Single-Page and Multi-Page Applications and their advantages over each other
- Types of components: Smart and Dumb
- Difference between smart and dumb components
- Routing in React
- Implementing routing in the application using a node package called ‘react-router-dom’
- Developing a functionality for deleting a subscriber

Poll 1

Which of the following components provide data and logic to the UI-based components?

A. Dumb components
B. Class components
C. Functional components
D. Smart components

Poll 1 (Answer)

Which of the following components provide data and logic to the UI-based components?

A. Dumb components
B. Class components
C. Functional components
D. Smart components

Poll 2

Which of the following is NOT a primary component of the React Router?

A. <Link>
B. <Route>
C. <BrowserRouter>
D. <LinkRouter>

Poll 2 (Answer)

Which of the following is NOT a primary component of the React Router?

A. <Link>

B. <Route>

C. <BrowserRouter>

D. <LinkRouter>

Today's Agenda

1. Introduction to React Hooks
    - Class-based vs functional components
    - Why Hooks?
2. Properties in Hooks
    - Using state in functional components
3. Routing in Hooks
    - Hooks in React Router library

Introduction to React Hooks

Class-Based Components vs Functional Components

There are two different ways to define components in React:

- Class-based components
  - Stateful
  - Access to lifecycle hooks
- Functional components
  - Stateless
  - Pure functions; hence, no lifecycle hooks

upGrad

Class-Based Component - Example

Example:

```
class MyCounter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: props.count
    };
  }
  onCounterClick = () => {
    this.setState({
        count: this.state.count + 1
    });
  }
  render() {
    return <div onClick={this.onCounterClick}>
              {this.state.count}
           </div>;
  }
}
```

Usage:

```
<MyCounter count={0} />
```

Converting to Functional Component

The previous example of class-based component can be converted to a functional component as follows:

```
function MyCounter(props) {
  return <div>
    {props.count}
  </div>;
}
```

Usage:

```
<MyCounter count={0} />
```

But then, how do you use the lifecycle hooks or even state in functional components?

Converting to Functional Component With Hooks

The answer is '**hooks**'.

Hooks are functions that let you 'hook into' the React state and lifecycle features of class components (not to be confused with React lifecycle hooks!)

With hooks, our example would be complete.

```
function MyCounter(props) {
  const [count, setCount] = React.useState(0);
  const onCounterClick = () => setCount(count + 1);

  return <div onClick={onCounterClick}>
    {props.count}
  </div>;
}
```

Poll 3

State whether the following is True or False.

Functional components in React essentially comprise just the render method.

A. True

B. False

Poll 3 (Answer)

State whether the following is True or False.

Functional components in React essentially comprise just the render method.

A. True

B. False

React Hooks

- Hooks are a new addition in React version 16.8
- Hooks allow using lifecycle methods and state in pure functional components without using class, leading to cleaner and more manageable code with good separation of business logic concerns
- Hooks are themselves functions provided either by the React library (built-in) or created by you
- Their use is optional. There are no plans to remove support for class-based components from React
- 100% backward-compatible and do not introduce any breaking changes

Why Hooks?

The release of hooks in React 16.8 solved the following pain points:

1. **Managing State:** Reusing logic between multiple components can lead to wrapper hell or deeply nested components
2. **Side Effects:** Unrelated mixed in logic in lifecycle methods can get repetitive, and cause unnecessary side effects
3. **Optimization:** Hooks might reduce your bundle size

Side Effects

- A side effect is generally anything that affects something outside the scope of the function being executed or, in the context of React, anything that modifies state outside of its local environment
- Common side effects include data fetching, setting up subscriptions, and manually changing the DOM in React components

Properties in Hooks

Properties in Hooks

upGrad

useState

- The useState is a React Hook that can be called inside a function component to add some local state to it
- React will preserve this state between re-renders
- useState returns a pair: The current state value and a function that lets you update it. You can call the updater from an event handler or anywhere else
- The only argument to useState is the initial state. The initial state argument is only used during the first render

Note: You will learn more about the useState hook and its implementation in the next session.

Properties in Hooks

upGrad

Using State in Functional Components

Let's rewrite the previous example with support for state management:

```
function MyCounter(props) {
  // Let's use the useState hook, return a state variable
  // and an updater function
  const [count, setCount] = React.useState(props.count);
  return <div onClick={ () => setCount(count + 1) }>
    {count}
  </div>;
}
```

Here, two things are defined: a state variable and a setter function to update the state variable. You can consume the state variable in JSX as you would with any other regular variable. And when you invoke the setter to change its value, React will automatically re-render the component

Poll 4

Which of the following is not true for state management with functional React components?

A. The useState hook is used to keep state within a component.

B. All state variables can be maintained separately instead of merging them into a single state object.

C. A component can have only a single state variable.

D. The state should only be updated using the updater function returned from the useState Hook, never directly.

Poll 4 (Answer)

Which of the following is not true for state management with functional React components?

A. The useState hook is used to keep state within a component.

B. All state variables can be maintained separately instead of merging them into a single state object.

C. A component can have only a single state variable.

D. The state should only be updated using the updater function returned from the useState Hook, never directly.

Implementing Properties Using Hooks

Adding multiple states

First, a component is used to display user name and status:

```
function User(props) {
  return <div>
    <div>Username: {props.name}</div>
    <div>Status: {props.status}</div>
  </div>;
}
```

upGrad

Implementing Properties Using Hooks

Then, another component is used to edit user name and status:

```
function UserManager(props) {
  const [name, setName] = useState('Anonymous');
  const [status, setStatus] = useState('Hello there!');

  return <div>
    <div>Username: {name}</div>
    <div>Status: {status}</div>

    <User name={name} status={status} />
  </div>;
}
```

This way, you can add more and more properties to a component, as required

Important Rules for Using Hooks

Things to remember while using Hooks

- Only call Hooks at the top level; never call Hooks inside loops, conditions or nested functions

  This is because the order in which all hooks are called must be the same on every render

- Only call Hooks from React functional components (or custom Hooks); never call Hooks directly from regular JavaScript functions

Properties in Hooks

upGrad

Phone Directory Application: Replacing this.state With useState

With reference to our Phone Directory application, you can simply update the class component to a functional component:

```
function PhoneDirectory() { ... }
```

Update references to *this.state* and *this.setState* to the useState Hook:

```
// this.state.subscribersList = [ ... ]
const [subscribersList, setSubscriberList] = useState([...]);
// this.setState({ subscribersList: [...] })
setSubscribersList([...]);
```

Code reference

Routing in Hooks

Routing in Hooks

upGrad

React Router

- React Router is a library used to handle routing within a React application
- Its primary components include:
  - Routers: <BrowserRouter>, <HashRouter>
  - Route Matchers: <Route>, <Switch>
  - Navigation: <Link>, <NavLink>, <Redirect>

React Router: Basic Example

Let’s consider a multi-page application, for example, an app that has a Home page and an About page. The page to be rendered is defined by the URL route:

- **myapp.com/about**: Loads the About page
- **myapp.com/home**: Should load the home page
- And simply, **myapp.com** loads the home page as well by default

Routing in Hooks

upGrad

React Router: Basic Example

Using BrowserRouter:

```
import { BrowserRouter as Router } from 'react-router-dom';

function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>
      <Switch>
        <Route exact path="/">
          <Home />
        </Route>
        <Route path="/about">
          <About />
        </Route>
    </Router>
  )
}
```

Routing in Hooks

upGrad

React Router Hooks

The following Hooks are provided by React Router:

- useHistory
- useLocation
- useParams
- useRouteMatch

Routing in Hooks

upGrad

useHistory

- The **useHistory** Hook returns the reference to the history object that you can use for navigation in the browser
- Example (*GoToHomeButton* component):

```
import { useHistory } from "react-router-dom";

function GoToHomeButton() {
  let history = useHistory();

  function handleClick() {
    history.push("/home");  // supports the HTML5 history API
  }

  return (
    <button onClick={handleClick}>Go to home page</button>
  );
}
```

Routing in Hooks

upGrad

useLocation

- The **useLocation** Hook returns a reference to the location object, which contains details of the current URL of the app
- Example (Click tracking):

```
import { useLocation } from "react-router-dom";

function App() {
  const location = useLocation();
  clickTracker('page_view', location.pathname); // track page view event on
every render

  return (
    <Switch>
      /* ... */
    </Switch>
  );
}
```

Routing in Hooks

upGrad

useParams

- The **useParams** Hook returns an object of key/value pairs of URL parameters
- Example (Displaying a particular product on a page):

```
import { useParams } from "react-router-dom";

function ProductPage() {
  const { productId } = useParams();
  // render productId details...
}

function App() {
  return <Router>
    <Switch>
      <Route exact path="/"><Home /></Route>
      <Route path="/product/:productId"><ProductPage /></Route>
    </Switch>
  </Router>;
}
```

Routing in Hooks

upGrad

useRouteMatch

- The **useRouteMatch** Hook attempts to match the current URL in the same way as a <Route>. It's mostly useful for getting access to the match data without actually rendering a <Route>
- Example (Render something based on a route match object):

```
function ProductPage() {
  return (
    <Route
      path="/product/:productId"
      render={({ match }) => {
        // Render something based on the match...
      }}
    />
  );
}
```

Routing in Hooks

upGrad

useRouteMatch

Alternately, you can use the useRouteMatch hook to achieve the same outcome in the previous example:

```
function ProductPage() {
  const match = useRouteMatch("/product/:productId");

  // Render something based on the match...
}
```

Poll 5

Which of the following is not a built-in React Router Hook?

A. useHistory
B. useLocation
C. useRoute
D. useParams

Poll 5 (Answer)

Which of the following is not a built-in React Router Hook?

A. useHistory

B. useLocation

C. useRoute

D. useParams

Poll 6

Which of the following is not a valid router in React Router library?

A. BrowserRouter
B. HashRouter
C. MemoryRouter
D. DynamicRouter

Poll 6 (Answer)

Which of the following is not a valid router in React Router library?

A. BrowserRouter

B. HashRouter

C. MemoryRouter

D. DynamicRouter

Poll 7

Which of the following is not a valid Hook in React Router library?

A. useHistory
B. useLocation
C. usePath
D. useParams

Poll 7 (Answer)

Which of the following is not a valid Hook in React Router library?

A. useHistory

B. useLocation

C. usePath

D. useParams

Hands-on Exercise

upGrad

Build a simple React app consisting of four pages: a default home page, a product listing page, a product details page and a cart page. The routes corresponding to each of these pages should look like the following:
Home page: /home
Product listing page: /products
Product details page: /products/<productId>
About page: /about

Add a navigation bar to the top of the app with the links to the various pages. You can use a hard-coded productId for the navbar. You can put dummy content on all of these pages

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/Session7

Doubt Clearance (5 minutes)

Important Questions

Important Questions

upGrad

1. Why were Hooks introduced in React?
2. What are the rules needs to follow for hooks? Do I need to rewrite all my class components with hooks? Do they cover all use cases for classes?
3. What is the stable release for hooks support?
4. What are the sources used for introducing hooks?
5. What is the purpose of eslint plugin for hooks?

Key Takeaways

- React Hooks help use state and other lifecycle methods in functional components
- Hooks offer a cleaner separation of concerns as compared to class-based components; however, there is no plan by the React team to deprecate class-based components in the future
- The order in which Hooks are called during each render should be the same, that is, Hooks should not be invoked inside a condition, a loop or a nested function, or from a regular JS function that is not a React component
- The React Router library offers a few Hooks to handle routing, namely, useHistory, useLocation and useParams

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

MCQs
Coding Questions
Course Project (Part B) - Starter Code and Configuration Guidelines

In the next class, we will discuss:

- Applying useState() Hook
  - Multiple State Variables
- Applying useEffect() Hook
  - Side effects
  - Cleanup in side effects
  - React lifecycle Hooks
- Applying useContext() Hook

upGrad

#RahoAmbitious

Thank You!