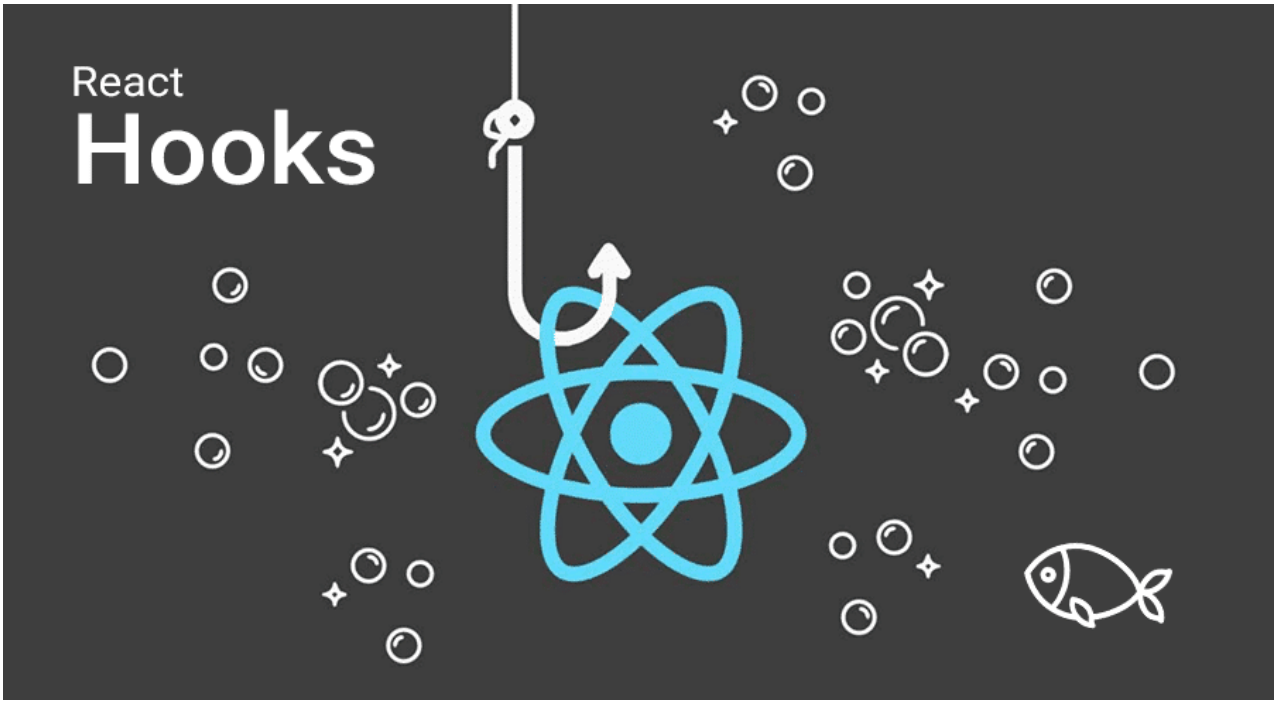
React.Js Hooks



Hooks are the new feature introduced in the React 16.8 version. It allows you to use state and other React features without writing a class. Hooks are the functions which "hook into" React state and lifecycle features from function components. It does not work inside classes.

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.

Rules of Hooks

Hooks are similar to JavaScript functions, but you need to follow these two rules when using them. Hooks rule ensures that all the stateful logic in a component is visible in its source code. These rules are:

- Only call Hooks at the top level

Do not call Hooks inside loops, conditions, or nested functions. Hooks should always be used at the top level of the React functions. This rule ensures that Hooks are called in the same order each time a component renders.

- Only call Hooks from React functions

You cannot call Hooks from regular JavaScript functions. Instead, you can call Hooks from React function components. Hooks can also be called custom Hooks.

Pre-requisites for React Hooks

- Node version 6 or above
- NPM version 5.2 or above
- Create-react-app tool for running the React App

React Hooks Installation

To use React Hooks, we need to run the following commands:

- *$ npm install react@16.8.0-alpha.1 --save*
- *$ npm install react-dom@16.8.0-alpha.1 --save*

The above command will install the latest React and React-DOM alpha versions which support React Hooks. Make sure the **package.json** file lists the React and React-DOM dependencies as given below.

- "react": "^16.8.0-alpha.1",
- "react-dom": "^16.8.0-alpha.1",

Hooks State

Hook state is the new way of declaring a state in React app. Hook uses useState() functional component for setting and retrieving state. Let us understand Hook's state with the following example.

App.js

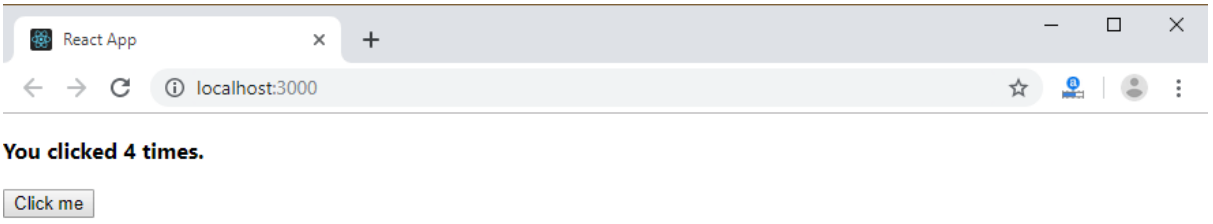
```
import React, { useState } from 'react';


function CountApp() {
  // 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>
  );
}

export default CountApp;
```

output:



In the above example, useState is the Hook which needs to call inside a function component to add some local state to it. The useState returns

a pair where the first element is the current state value/initial value, and the second one is a function that allows us to update it. After that, we will call this function from an event handler or somewhere else. The useState is similar to this.setState in class. The equivalent code without Hooks looks like as below.

App.js

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


class CountApp extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
  render() {
    return (
      <div>
        <p><b>You clicked {this.state.count} times</b></p>
```

```
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}

export default CountApp;
```

Hooks Effect

The Effect Hook allows us to perform side effects (an action) in the function components. It does not use components lifecycle methods that are available in class components. In other words, Effects Hooks are equivalent to componentDidMount(), componentDidUpdate(), and componentWillUnmount() lifecycle methods.

Side effects have common features which the most web applications need to perform, such as:

- Updating the DOM,
- Fetching and consuming data from a server API,
- Setting up a subscription, etc.

Let us understand Hook Effect with the following example.

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


function CounterExample() {
  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>
  );
}

export default CounterExample;
```

The above code is based on the previous example with a new feature in which we set the document title to a custom message, including the number of clicks.

Output: