

Home	React Native	ReactJS	TypeScript	JavaScript	Framework7	COA	HTML	CSS	Selenium
------	--------------	---------	------------	------------	------------	-----	------	-----	----------

React 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:

1. 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 components renders.

2. 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 from custom Hooks.

Pre-requisites for React Hooks

1. Node version 6 or above
2. NPM version 5.2 or above
3. 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",
```

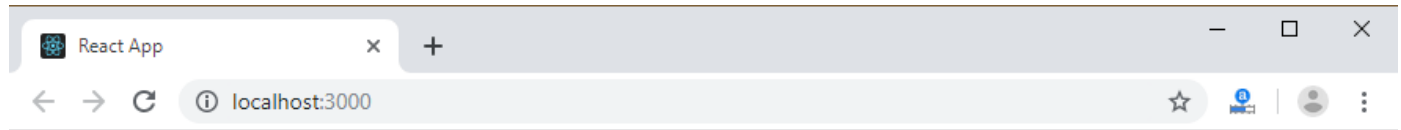
Hooks State

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 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>  
  );  
}
```

```
    </div>
  );
}
export default CountApp;
```

output:

You clicked 4 times.

Click me

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 which 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 which 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.

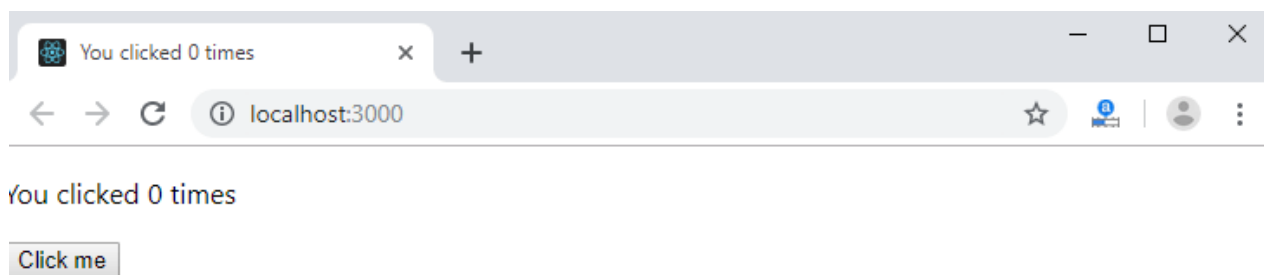
Let us understand Hook Effect with the following example.

```
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 which we set the document title to a custom message, including the number of clicks.

Output:



In React component, there are two types of side effects:

1. Effects Without Cleanup
2. Effects With Cleanup

Effects without Cleanup

It is used in `useEffect` which does not block the browser from updating the screen. It makes the app more responsive. The most common example of effects which don't require a cleanup are manual DOM mutations, Network requests, Logging, etc.

Effects with Cleanup

Some effects require cleanup after DOM updation. For example, if we want to set up a subscription to some external data source, it is important to clean up memory so that we don't introduce a memory leak. React performs the cleanup of memory when the component unmounts. However, as we know that, effects run for every render method and not just once. Therefore, React also cleans up effects from the previous render before running the effects next time.

Custom Hooks

A custom Hook is a JavaScript function. The name of custom Hook starts with "use" which can call other Hooks. A custom Hook is just like a regular function, and the word "use" in the beginning tells that this function follows the rules of Hooks. Building custom Hooks allows you to extract component logic into reusable functions.

Let us understand how custom Hooks works in the following example.

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

const useDocumentTitle = title => {
  useEffect(() => {
    document.title = title;
```

```
    }, [title])
  }

function CustomCounter() {
  const [count, setCount] = useState(0);
  const incrementCount = () => setCount(count + 1);
  useDocumentTitle(`You clicked ${count} times`);
  // useEffect(() => {
  //   document.title = `You clicked ${count} times`
  // });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={incrementCount}>Click me</button>
    </div>
  )
}

export default CustomCounter;
```

In the above snippet, useDocumentTitle is a custom Hook which takes an argument as a string of text which is a title. Inside this Hook, we call useEffect Hook and set the title as long as the title has changed. The second argument will perform that check and update the title only when its local state is different than what we are passing in.



Note: A custom Hook is a convention which naturally follows from the design of Hooks, instead of a React feature.

Built-in Hooks

Here, we describe the APIs for the built-in Hooks in React. The built-in Hooks can be divided into two parts, which are given below.

Basic Hooks

- useState
- useEffect
- useContext

Additional Hooks

- useReducer
- useCallback
- useMemo
- useRef
- useImperativeHandle
- useLayoutEffect
- useDebugValue

[← Prev](#)[Next →](#)

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share




Learn Latest Tutorials

 Splunk tutorial Splunk	 SPSS tutorial SPSS	 Swagger tutorial Swagger	 T-SQL tutorial Transact-SQL	 Tumblr tutorial Tumblr
 React tutorial ReactJS	 Regex tutorial Regex	 Reinforcement learning tutorial Reinforcement Learning	 R Programming tutorial R Programming	 RxJS tutorial RxJS
 React Native tutorial React Native	 Python Design Patterns Python Design Patterns	 Python Pillow tutorial Python Pillow	 Python Turtle tutorial Python Turtle	 Keras tutorial Keras


Preparation



Aptitude
Aptitude



Logical
Reasoning
Reasoning



Verbal Ability
Verbal Ability



Interview
Questions
Interview Questions



Company
Interview
Questions
Company Questions

Trending Technologies




Artificial
Intelligence
Artificial
Intelligence




AWS Tutorial
AWS



Selenium
tutorial
Selenium




Cloud
Computing
Cloud Computing



Hadoop tutorial
Hadoop



ReactJS
Tutorial
ReactJS



Data Science
Tutorial
Data Science




Angular 7
Tutorial
Angular 7



Blockchain
Tutorial
Blockchain



Git Tutorial
Git



Machine
Learning Tutorial
Machine Learning




DevOps
Tutorial
DevOps


B.Tech / MCA




DBMS tutorial
DBMS




Data Structures
tutorial
Data Structures



DAA tutorial
DAA



Operating
System
Operating System



Computer
Network tutorial
Computer Network

 Compiler Design tutorial Compiler Design	 Computer Organization and Architecture Computer Organization	 Discrete Mathematics Tutorial Discrete Mathematics	 Ethical Hacking Ethical Hacking	 Computer Graphics Tutorial Computer Graphics
 Software Engineering Software Engineering	 html tutorial Web Technology	 Cyber Security tutorial Cyber Security	 Automata Tutorial Automata	 C Language tutorial C Programming
 C++ tutorial C++	 Java tutorial Java	 .Net Framework tutorial .Net	 Python tutorial Python	 List of Programs Programs
 Control Systems tutorial Control System	 Data Mining Tutorial Data Mining	 Data Warehouse Tutorial Data Warehouse		