|  |  |
| --- | --- |
| **ch31.ReactJS Higher-Order Components** | **Date: 22-02-2022** |

**Topics**

React Props Validation,

# React Props Validation

It is also known as HOC. In React, HOC is an advanced technique for reusing component logic. It is a function that takes a component and returns a new component. According to the official website, it is not the feature(part) in React API, but a pattern that emerges from React compositional nature. They are similar to JavaScript functions used for adding additional functionalities to the existing component.

A higher order component function accepts another function as an argument. The **map** function is the best example to understand this. The main goal of this is to decompose the component logic into simpler and smaller functions that can be reused as you need.

### **Syntax**

1. **const** NewComponent = higherOrderComponent(WrappedComponent);

We know that component transforms props into UI, and a higher-order component converts a component another component and allows to add additional data or functionality into this. **Hocs** are common in **third-party** libraries. The examples of HOCs are **Redux's connect** and **Relay's createFragmentContainer**.

Now, we can understand the **working of HOCs** from the below example.

1. //Function Creation
2. function add (a, b) {
3. **return** a + b
4. }
5. function higherOrder(a, addReference) {
6. **return** addReference(a, 20)
7. }
8. //Function call
9. higherOrder(30, add) // 50

In the above example, we have created two functions **add()** and **higherOrder()**. Now, we provide the add() function as an **argument** to the higherOrder() function. For invoking, rename it **addReference** in the higherOrder() function, and then **invoke it**.

Here, the function you are passing is called a callback function, and the function where you are passing the callback function is called a **higher-order(HOCs)** function.

### **Example**

Create a new file with the name HOC.js. In this file, we have made one function HOC. It accepts one **argument** as a component. Here, that component is **App**.

**HOC.js**

1. **import** React, {Component} from 'react';
3. export **default** function Hoc(HocComponent){
4. **return** **class** **extends** Component{
5. render(){
6. **return** (
7. <div>
8. <HocComponent></HocComponent>
9. </div>
11. );
12. }
13. }
14. }

Now, include **HOC.js** file into the **App.js** file. In this file, we need to **call** the HOC function.

1. App = Hoc(App);

The App component wrapped inside another React component so that we can modify it. Thus, it becomes the primary application of the Higher-Order Components.

**App.js**

1. **import** React, { Component } from 'react';
2. **import** Hoc from './HOC';
4. **class** App **extends** Component {
5. render() {
6. **return** (
7. <div>
8. <h2>HOC Example</h2>
9. JavaTpoint provides best CS tutorials.
10. </div>
11. )
12. }
13. }
14. App = Hoc(App);
15. export **default** App;

**Output**

When we execute the above file, it will give the output as below screen.



## **Higher-Order Component Conventions**

* Do not use HOCs inside the render method of a component.
* The static methods must be copied over to have access to them. You can do this using hoist-non-react-statics package to automatically copy all non-React static methods.
* HOCs does not work for refs as 'Refs' does not pass through as a parameter or argument. If you add a ref to an element in the HOC component, the ref refers to an instance of the outermost container component, not the wrapped component.

Next Topic[React Code Splitting](https://www.javatpoint.com/react-code-splitting)