|  |  |
| --- | --- |
| **ch23.ReactJS Refs** | **Date: 22-02-2022** |

**Topics**

React Props Validation,

# React Props Validation

# React Refs

Refs is the shorthand used for **references** in React. It is similar to **keys** in React. It is an attribute which makes it possible to store a reference to particular DOM nodes or React elements. It provides a way to access React DOM nodes or React elements and how to interact with it. It is used when we want to change the value of a child component, without making the use of props.

## **When to Use Refs**

Refs can be used in the following cases:

* When we need DOM measurements such as managing focus, text selection, or media playback.
* It is used in triggering imperative animations.
* When integrating with third-party DOM libraries.
* It can also use as in callbacks.

## **When to not use Refs**

* Its use should be avoided for anything that can be done **declaratively**. For example, instead of using **open()** and **close()** methods on a Dialog component, you need to pass an **isOpen** prop to it.
* You should have to avoid overuse of the Refs.

## **How to create Refs**

In React, Refs can be created by using **React.createRef()**. It can be assigned to React elements via the **ref** attribute. It is commonly assigned to an instance property when a component is created, and then can be referenced throughout the component.

1. **class** MyComponent **extends** React.Component {
2. constructor(props) {
3. **super**(props);
4. **this**.callRef = React.createRef();
5. }
6. render() {
7. **return** <div ref={**this**.callRef} />;
8. }
9. }

## **How to access Refs**

In React, when a ref is passed to an element inside render method, a reference to the node can be accessed via the current attribute of the ref.

Competitive questions on Structures in Hindi

Keep Watching

1. **const** node = **this**.callRef.current;

## **Refs current Properties**

The ref value differs depending on the type of the node:

* When the ref attribute is used in HTML element, the ref created with **React.createRef()** receives the underlying DOM element as its **current** property.
* If the ref attribute is used on a custom class component, then ref object receives the **mounted** instance of the component as its current property.
* The ref attribute cannot be used on **function components** because they don't have instances.

## **Add Ref to DOM elements**

In the below example, we are adding a ref to store the reference to a DOM node or element.

1. **import** React, { Component } from 'react';
2. **import** { render } from 'react-dom';
4. **class** App **extends** React.Component {
5. constructor(props) {
6. **super**(props);
7. **this**.callRef = React.createRef();
8. **this**.addingRefInput = **this**.addingRefInput.bind(**this**);
9. }
11. addingRefInput() {
12. **this**.callRef.current.focus();
13. }
15. render() {
16. **return** (
17. <div>
18. <h2>Adding Ref to DOM element</h2>
19. <input
20. type="text"
21. ref={**this**.callRef} />
22. <input
23. type="button"
24. value="Add text input"
25. onClick={**this**.addingRefInput}
26. />
27. </div>
28. );
29. }
30. }
31. export **default** App;

**Output**



## **Add Ref to Class components**

In the below example, we are adding a ref to store the reference to a class component.

### **Example**

1. **import** React, { Component } from 'react';
2. **import** { render } from 'react-dom';
4. function CustomInput(props) {
5. let callRefInput = React.createRef();
7. function handleClick() {
8. callRefInput.current.focus();
9. }
11. **return** (
12. <div>
13. <h2>Adding Ref to Class Component</h2>
14. <input
15. type="text"
16. ref={callRefInput} />
17. <input
18. type="button"
19. value="Focus input"
20. onClick={handleClick}
21. />
22. </div>
23. );
24. }
25. **class** App **extends** React.Component {
26. constructor(props) {
27. **super**(props);
28. **this**.callRefInput = React.createRef();
29. }
31. focusRefInput() {
32. **this**.callRefInput.current.focus();
33. }
35. render() {
36. **return** (
37. <CustomInput ref={**this**.callRefInput} />
38. );
39. }
40. }
41. export **default** App;

**Output**



## **Callback refs**

In react, there is another way to use refs that is called "**callback refs**" and it gives more control when the refs are **set** and **unset**. Instead of creating refs by createRef() method, React allows a way to create refs by passing a callback function to the ref attribute of a component. It looks like the below code.

1. <input type="text" ref={element => **this**.callRefInput = element} />

The callback function is used to store a reference to the DOM node in an instance property and can be accessed elsewhere. It can be accessed as below:

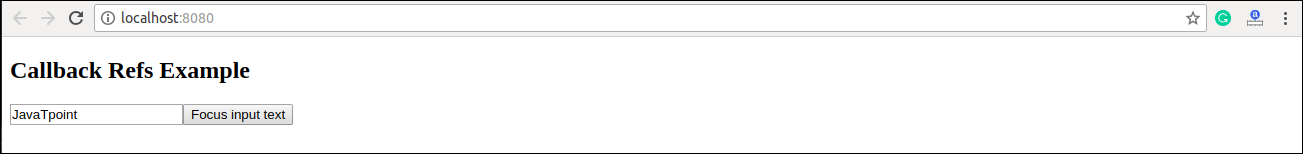
1. **this**.callRefInput.value

The example below helps to understand the working of callback refs.

1. **import** React, { Component } from 'react';
2. **import** { render } from 'react-dom';
4. **class** App **extends** React.Component {
5. constructor(props) {
6. **super**(props);
8. **this**.callRefInput = **null**;
10. **this**.setInputRef = element => {
11. **this**.callRefInput = element;
12. };
14. **this**.focusRefInput = () => {
15. //Focus the input using the raw DOM API
16. **if** (**this**.callRefInput) **this**.callRefInput.focus();
17. };
18. }
20. componentDidMount() {
21. //autofocus of the input on mount
22. **this**.focusRefInput();
23. }
25. render() {
26. **return** (
27. <div>
28. <h2>Callback Refs Example</h2>
29. <input
30. type="text"
31. ref={**this**.setInputRef}
32. />
33. <input
34. type="button"
35. value="Focus input text"
36. onClick={**this**.focusRefInput}
37. />
38. </div>
39. );
40. }
41. }
42. export **default** App;

In the above example, React will call the "ref" callback to store the reference to the input DOM element when the component **mounts**, and when the component **unmounts**, call it with **null**. Refs are always **up-to-date** before the **componentDidMount** or **componentDidUpdate** fires. The callback refs pass between components is the same as you can work with object refs, which is created with React.createRef().

**Output**



## **Forwarding Ref from one component to another component**

Ref forwarding is a technique that is used for passing a **ref** through a component to one of its child components. It can be performed by making use of the **React.forwardRef()** method. This technique is particularly useful with **higher-order components** and specially used in reusable component libraries. The most common example is given below.

### **Example**

1. **import** React, { Component } from 'react';
2. **import** { render } from 'react-dom';
4. **const** TextInput = React.forwardRef((props, ref) => (
5. <input type="text" placeholder="Hello World" ref={ref} />
6. ));
8. **const** inputRef = React.createRef();
10. **class** CustomTextInput **extends** React.Component {
11. handleSubmit = e => {
12. e.preventDefault();
13. console.log(inputRef.current.value);
14. };
15. render() {
16. **return** (
17. <div>
18. <form onSubmit={e => **this**.handleSubmit(e)}>
19. <TextInput ref={inputRef} />
20. <button>Submit</button>
21. </form>
22. </div>
23. );
24. }
25. }
26. export **default** App;

In the above example, there is a component **TextInput** that has a child as an input field. Now, to pass or forward the **ref** down to the input, first, create a ref and then pass your ref down to **<TextInput ref={inputRef}>**. After that, React forwards the ref to the **forwardRef** function as a second argument. Next, we forward this ref argument down to **<input ref={ref}>**. Now, the value of the DOM node can be accessed at **inputRef.current**.

## **React with useRef()**

It is introduced in **React 16.7** and above version. It helps to get access the DOM node or element, and then we can interact with that DOM node or element such as focussing the input element or accessing the input element value. It returns the ref object whose **.current** property initialized to the passed argument. The returned object persist for the lifetime of the component.

### **Syntax**

1. **const** refContainer = useRef(initialValue);

### **Example**

In the below code, **useRef** is a function that gets assigned to a variable, **inputRef**, and then attached to an attribute called ref inside the HTML element in which you want to reference.

1. function useRefExample() {
2. **const** inputRef= useRef(**null**);
3. **const** onButtonClick = () => {
4. inputRef.current.focus();
5. };
6. **return** (
7. <>
8. <input ref={inputRef} type="text" />
9. <button onClick={onButtonClick}>Submit</button>
10. </>
11. );
12. }

Next Topic[React Fragments](https://www.javatpoint.com/react-fragments)