ReactJS (Day5)

Monday, December 2, 2024 8:58 AM

What is PureComponent ?

React's PureComponent class is an optimization designed to reduce unnecessary re-renders by automatically performing a shallow comparison of the current and previous props and state.

When to Use PureComponent

- Your component's output depends purely on its props and state.
- You want to optimize performance and reduce unnecessary renders.

In state,

```
import React, { PureComponent } from "react";
class Purecomponent extends PureComponent {
  constructor() {
    super();
    this.state = {
      cnt: 0,
    };
  }
  render() {
    console.log("REnder");
    return (
      <>
        <p>PureComponent {this.state.cnt}</p>
        <button onClick={() => this.setState({ cnt: this.state.cnt + 1 })}>
          Update Count
        </button>
      </>
    );
  }
}
export default Purecomponent;
```

In Props,

```
import React, {PureComponent } from 'react'
export default class Purecomponent1 extends  PureComponent {
    constructor(props)
    {
        super(props);
        this.state = {
            cnt : this.props.cnt
        }
    }
  render() {
    console.log("Render1");

    return (
      <>
        <h1>{this.state.cnt}</h1>
        <button onClick={()=>this.setState({cnt:this.state.cnt})}>Update Count</button>
      </>
    )
  }
}
```

In Functional Component we use useMemo instead of purecomponent :

Use **useMemo** when:

1. You have heavy calculations that don't need to run every time the component re-renders.
2. You want to avoid re-rendering child components by memoizing computed values they depend on.

Syntax :-

```
const memoizedValue = useMemo(() => {
  // Expensive computation
  return result;
}, [dependencies]);
```

Example :

```jsx
import React, { useState, useMemo } from 'react';
function UseMemo() {
  const [cnt, setCnt] = useState(0);
  const [item, setItem] = useState(1);
  const mt = useMemo(() => {
    console.log('Recalculating mt...');
    return cnt + 2;
  }, [item]);
  return (
    <>
      <h1>Count: {cnt}</h1>
      <h1>Item: {item}</h1>
      <h1>Memoized Value (mt): {mt}</h1>
      <button onClick={() => setCnt(cnt + 1)}>Update Count</button>
      <button onClick={() => setItem(item + 1)}>Update Item</button>
    </>
  );
}
export default UseMemo;
```

Feature	useEffect	useMemo
Purpose	Used for side effects (e.g., data fetching, subscriptions, timers)	Used for performance optimization (memoizing values or calculations)
Trigger	Runs after every render or when dependencies change	Runs during render only when dependencies change
Return Value	No return value (side effects only)	Returns a memoized value
Usage	For interacting with the outside world (e.g., API calls, subscriptions)	For optimizing expensive calculations
Dependencies	Triggers based on dependency array (optional)	Recalculates when dependencies change
Example Use Case	Fetching data after a render or cleaning up subscriptions	Memoizing a computed value to avoid unnecessary recalculations
Execution Timing	After rendering, and optionally cleaned up on unmount	During rendering, before component re-renders
Effect on Rendering	May cause re-renders depending on what's done inside it	Does not directly trigger re-renders; only recalculates when dependencies change

What is Ref ?
In **React**, a **ref** (short for "reference") is an object that provides a way to directly access and interact with a DOM element or a React component instance.

In Class Component,
Example :

```jsx
import React, { createRef} from 'react'
import { Component } from 'react';
export default class ReF extends Component{
    constructor()
    {
        super();
        this.inputRef = createRef();
    }
    getVal()
    {
        this.inputRef.current.style.color="red";
        alert(this.inputRef.current.value);
    }
  render() {
    return (
        <>
            <h1>Ref in react</h1>
            <input ref={this.inputRef} type="text"/>
            <button onClick={()=>this.getVal()}>Get Value</button>
        </>
    )
  }
}
```

In functional component,
Example:

```jsx
import React,{useRef} from 'react'
function ReF1() {
    const ip = useRef(null);
    function getval()
    {
        ip.current.style.color='red'
        alert(ip.current.value);
```

```
    }
  return (
    <>
        <input type="text" ref={ip}/>
        <button onClick={getval}>Get Value</button>
    </>
  )
}
export default ReF1
```

What is forwardRef ?
forwardRef is a higher-order component in React that allows you to pass a **ref** through a component to one of its children.

When you create a custom component, you might want to expose a DOM element (e.g., an input, div, etc.) to the parent so that the parent can directly interact with it (for example, focusing an input). forwardRef allows you to achieve this.

Feature	**useRef**	**forwardRef**
Type	Hook (used inside functional components)	Higher-order component (HOC)
Purpose	Creates a reference to a DOM element or value within a component	Forwards a ref from a parent to a child component
Use Case	Accessing and manipulating a DOM element, or storing a value across renders	Passing a ref down to a child component that needs it
Returns	A mutable object with a current property	A new component that can accept and forward a ref
When to Use	When you need to access a DOM element or persist values without triggering re-renders	When you want a child component to accept a ref and pass it down to a DOM element inside it
Example	const inputRef = useRef(null)	const ForwardedInput = forwardRef(Input)

Example :

```
import { useRef } from "react"
import ForwardRef from "./component/ForwardRef"
function App() {
  const ip = useRef(null);
  function updateInput()
  {
    ip.current.value = "1000"
    ip.current.focus();
  }
  return (
    <>
      <ForwardRef ref={ip}></ForwardRef>
      <button onClick={updateInput}>Update Input Box</button>
    </>
  )
}
export default App;
```

```
import React, { forwardRef } from 'react'
function ForwardRef(props,ref) {
  return (
    <>
    <input type="text" ref={ref}/>
    </>
  )
}
export default forwardRef(ForwardRef);
```

What is controlled component?
In React, a **controlled component** is an input element (like <input>, <textarea>, or <select>) whose value is controlled by React state.

Example :

```
import React, { useState } from 'react'
function Controlled() {
    const [name,setName] = useState("");
    const handlechange = (e) =>
    {
        setName(e.target.value)
    }
  return (
    <>
        <input type="text" value={name} onChange={handlechange}/>
        <h1>{name}</h1>
```

```
    </>
  )
}
export default Controlled;
```

What is Uncontrolled Component?

An **uncontrolled component** in React is a form element (like <input>, <textarea>, or <select>) whose value is managed by the **DOM**, not by React's state.

When to Use Uncontrolled Components:

- When you need to handle simple forms without React managing every form element.
- When integrating with non-React libraries or code that manages form elements.
- For performance reasons, where the overhead of managing state might be unnecessary.

```
import React, { createRef } from 'react'
function Uncontrolled() {
    const ip = createRef(null);
    function getval()
    {
        ip.current.style.color='red'
    }
  return (
    <>
        <input type="text" ref={ip}/>
        <button onClick={()=>getval()}>Get val</button>
    </>
  )
}
export default Uncontrolled;
```

Feature	Controlled Component	Uncontrolled Component
Value Management	React manages the input value through state	DOM manages the input value, React does not track it
State Binding	The value is bound to a state variable	The value is not bound to any React state
Usage	Useful for complex forms where validation, formatting, or conditional behavior is needed	Simpler forms or when using non-React libraries
Accessing Value	The value is accessed via state	The value is accessed via ref
Re-rendering	The component re-renders every time the state changes	No re-rendering happens on input change
Control	React has full control over the input element	Input value is controlled by the DOM
Example	<input value={value} onChange={handleChange} />	<input ref={inputRef} />

What is Higher Order Component?

A **Higher-Order Component (HOC)** is a function in React that takes a **component** as an argument and **returns a new component** with enhanced functionality.

```
import { useState } from "react";
import HOC from "./component/HOC/HOC"
function App() {
  return (
    <>
      <HOC cmp={<Counter/>}></HOC>
    </>
  )
}
function Counter() {
  const [cnt,setCnt] = useState(0);
return (
  <>
  <h1>{cnt}</h1>
  <button onClick={()=>setCnt(cnt+1)}>Update Count</button>
  </>
)
}
export default App
```

```
import React from 'react';
function HOC(props) {
  return (
```

```
    <>
      <h2>Enhanced Component:</h2>
      {props.cmp}
    </>
  );
}
export default HOC;
```