The useRef hook in React is a built-in hook that allows you to persist mutable values across renders without causing a re-render when the value changes. It's often used for accessing DOM elements or storing values that don't need to trigger a component update.

const refContainer = useRef(initialValue);

initialValue – The initial value you want the ref to start with.

refContainer – An object with a .current property.

**Accessing DOM Elements**

import React, { useRef } from 'react';

function InputFocus() {

const inputRef = useRef(null);

const focusInput = () => {

inputRef.current.focus(); // access the DOM element

};

return (

<>

<input ref={inputRef} type="text" />

<button onClick={focusInput}>Focus Input</button>

</>

);

}

**Persisting Values Across Renders**

import React, { useRef, useEffect } from 'react';

function TimerComponent() {

const count = useRef(0);

useEffect(() => {

const interval = setInterval(() => {

count.current += 1;

console.log('Count:', count.current);

}, 1000);

return () => clearInterval(interval);

}, []);

return <div>Check the console for count updates</div>;

}

|  |  |  |
| --- | --- | --- |
| **Feature** | **useState** | **useRef** |
| Purpose | Managing component state and triggering re-renders. | Accessing DOM elements or storing mutable values without re-renders. |
| Return Value | An array with two elements: the current state value and a function to update it. | An object with a .current property that holds the mutable value. |
| Re-renders | Updating the state with the state update function triggers a re-render. | Modifying the .current property does not trigger a re-render. |
| Mutability | State should be treated as immutable; always update it using the provided function. | The .current property is mutable and can be directly modified. |
| Use Cases | Storing and updating data that affects the UI, such as form inputs, toggle states, etc. | Accessing DOM elements, storing timer IDs, or persisting values across renders without causing updates. |
| Update behaviour | State updates are asynchronous. | Ref updates are synchronous |

The useMemo hook in React is used to memoize (i.e., cache) the result of a computationally expensive function so that it's only re-evaluated when necessary, not on every render.

const memoizedValue = useMemo(() => computeFunction(), [dependencies]);

computeFunction: The function whose result you want to memoize.

[dependencies]: An array of values that, when changed, will re-run the compute function.

Avoids unnecessary recalculations.

Only recalculates when dependencies change.

Helps optimize performance in large applications.

function App({ num }) {

const expensiveCalc = () => {

console.log("Calculating...");

let result = 0;

for (let i = 0; i < 1e8; i++) result += i;

return result + num;

};

const result = expensiveCalc(); // re-runs on every render

return <div>Result: {result}</div>;

}

import { useMemo } from 'react';

function App({ num }) {

const result = useMemo(() => {

console.log("Calculating...");

let total = 0;

for (let i = 0; i < 1e8; i++) total += i;

return total + num;

}, [num]); // only recalculates if `num` changes

return <div>Result: {result}</div>;

}