Hema Coding School

YouTube Link: https://www.youtube.com/@HemaCodingSchool

useCallback

Purpose

- useCallback is a React hook that returns a memoized version of the callback function that only changes if one of the dependencies has changed.
- It is useful to prevent the creation of new function instances on every render, which can improve performance when passing callbacks to child components.

```
const memoizedCallback = useCallback(() => {
  doSomething(a, b);
}, [a, b]);
```

Use Case

- When you pass a callback to a child component and want to prevent unnecessary re-renders of the child component.

Example

Without useCallback:

```
import React, { useState } from "react";

function Example() {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(count + 1);
  };

  return <ChildComponent increment={increment} />;
}

function ChildComponent({ increment }) {
  console.log("Child rendered");
  return <button onClick={increment}>Increment</button>;
}
```

With useCallback:

```
import React, { useState, useCallback } from "react";

function Example() {
  const [count, setCount] = useState(0);

  const increment = useCallback(() => {
    setCount(count + 1);
  }, [count]);

  return <ChildComponent increment={increment} />;
}

function ChildComponent({ increment }) {
  console.log("Child rendered");
  return <button onClick={increment}>Increment</button>;
}
```

Hema Coding School

YouTube Link: https://www.youtube.com/@HemaCodingSchool

useMemo

Purpose

- useMemo is a React hook that returns a memoized value. It only recomputes the memoized value when one of the dependencies has changed.
- It is used to avoid expensive calculations on every render.

Syntax

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

Use Case

- When you have a computationally expensive function whose result you want to cache and reuse until certain dependencies change.

Example

Without useMemo:

```
import React, { useState } from "react";

function Example() {
  const [number, setNumber] = useState(0);

  const isEven = (() => {
    for (let i = 0; i < 1000000000; i++) {} // Expensive computation
    return number % 2 === 0;
  })();

  return <div>{isEven ? "Even" : "Odd"}</div>;
}
```

With useMemo:

```
import React, { useState, useMemo } from "react";

function Example() {
  const [number, setNumber] = useState(0);
  const isEven = useMemo(() => {
    for (let i = 0; i < 1000000000; i++) {} // Expensive computation
    return number % 2 === 0;
  }, [number]);
  return <div>{isEven ? "Even" : "Odd"}</div>;
}
```

Hema Coding School

YouTube Link: https://www.youtube.com/@HemaCodingSchool

Key Differences

- **Purpose**: useCallback is used for memoizing functions, while useMemo is used for memoizing values.
- **Return Value**: useCallback returns a memoized function, whereas useMemo returns a memoized value.
- **Use Case**: Use useCallback when you want to optimize function references, especially useful when passing callbacks to child components. Use useMemo when you want to optimize expensive calculations that return values.

Interview Questions:

Callback:

1. **What is useCallback in React?**
   - **Answer**: useCallback is a hook in React that returns a memoized version of a callback function. It only changes if one of the dependencies has changed. It is used to prevent the creation of new function instances on every render, which can improve performance when passing callbacks to child components.
2. **When would you use useCallback?**
   - **Answer**: You would use useCallback when you pass a function as a prop to a child component and want to prevent unnecessary re-renders of that child component. It is particularly useful in optimizing performance when dealing with expensive operations or deep component trees.
3. **How does useCallback differ from useMemo?**
   - **Answer**: useCallback is used to memoize functions, whereas useMemo is used to memoize values. useCallback returns a memoized function, while useMemo returns a memoized value resulting from a computation.
4. **Can you give an example where useCallback prevents unnecessary re-renders?**
   - **Answer**: Yes, consider a parent component passing a function to a child component. Without useCallback, the function would be recreated on every render, causing the child to re-render. With useCallback, the function is memoized and only changes when its dependencies change, preventing unnecessary re-renders.
5. **What are the dependencies in useCallback, and how should they be handled?**
   - **Answer**: Dependencies in useCallback are the values or variables that the callback function relies on. The callback function is recreated only if these dependencies change. It's important to correctly specify these dependencies to ensure the function has access to the latest values and to avoid unnecessary recreations.

useMemo

1. **What is useMemo in React?**
   - **Answer**: useMemo is a hook in React that returns a memoized value. It only recalculates the memoized value when one of its dependencies changes. It is used to optimize performance by avoiding expensive calculations on every render.

2. **When would you use useMemo?**
    - **Answer**: You would use useMemo when you have a computationally expensive function and you want to cache its result. This prevents the function from being recalculated on every render, improving performance by recalculating only when necessary dependencies change.
3. **Can you provide an example of useMemo optimizing an expensive computation?**
    - **Answer**: Yes, consider a component that performs a costly calculation, like filtering a large dataset. Without useMemo, this calculation would run on every render. With useMemo, you can memoize the result, so the calculation only reruns when its dependencies change.
4. **How does useMemo help in optimizing performance?**
    - **Answer**: useMemo helps optimize performance by memoizing the result of an expensive computation. This means that the computation is only redone when necessary, reducing the amount of work React needs to do on each render, thus improving performance.
5. **What should you be careful about when using useMemo?**
    - **Answer**: When using useMemo, it's important to correctly specify dependencies to ensure the memoized value is recalculated when needed. Overuse of useMemo can add unnecessary complexity to your code. It should be used only when there is a clear performance benefit.

Scenario-based Questions

1. **Scenario: You have a component that fetches data from an API and filters it based on some criteria. Would you use useCallback or useMemo to optimize this, and why?**
    - **Answer**: In this scenario, useMemo would be more appropriate to memoize the filtered data, so the filtering operation only happens when the criteria or the data changes. If there is a callback function that is passed to child components to trigger the filtering, useCallback could be used to memoize that function.

2. **Scenario: A parent component passes a handler function to multiple child components, and you notice that these child components re-render unnecessarily. How would you address this issue?**
    - **Answer**: I would use useCallback to memoize the handler function. This ensures that the same function instance is passed to the child components on re-renders, preventing unnecessary re-renders of the child components due to the function reference changing.