Hook - useCallback

By Umesh Bichukale - CODEMIND Technology

Contact us 7758 0942 41

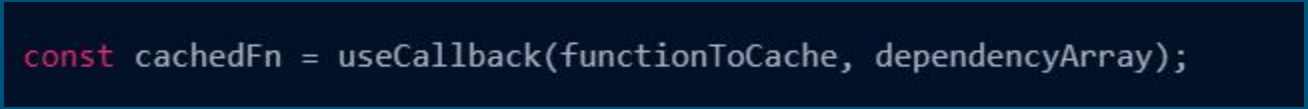
useCallback

- Introduction to useCallback
- useCallback Syntax
- useCallback Hook Usage
- Example : Without using the useCallback
- Example : Using React useCallback
- Common ways to execute JS
- Console Objects
- Variables
- Data types

Introduction to useCallback

- The useCallback is a react hook that lets us memoize a function block between the subsequent rendering of a component to improve the performance of the application. It simply means we can cache a function's definition, and we can avoid general re-computing of it again and again on every render; instead, we can instruct the react to only reconstruct it when we necessarily need, it by passing a set of dependency values in the form of an array.

useCallback Hook Syntax



```
const cachedFn = useCallback(functionToCache, dependencyArray);
```

The useCallback hook in React accepts two parameters.

functionToCache - It is the function definition we want to cache, so that its automatic re-rendering can be avoided.

dependencyArray - It is an array with a list of dependencies, which means values. It is an array of those values, which, if changed, then we want the function to render itself again and re-compute the result.

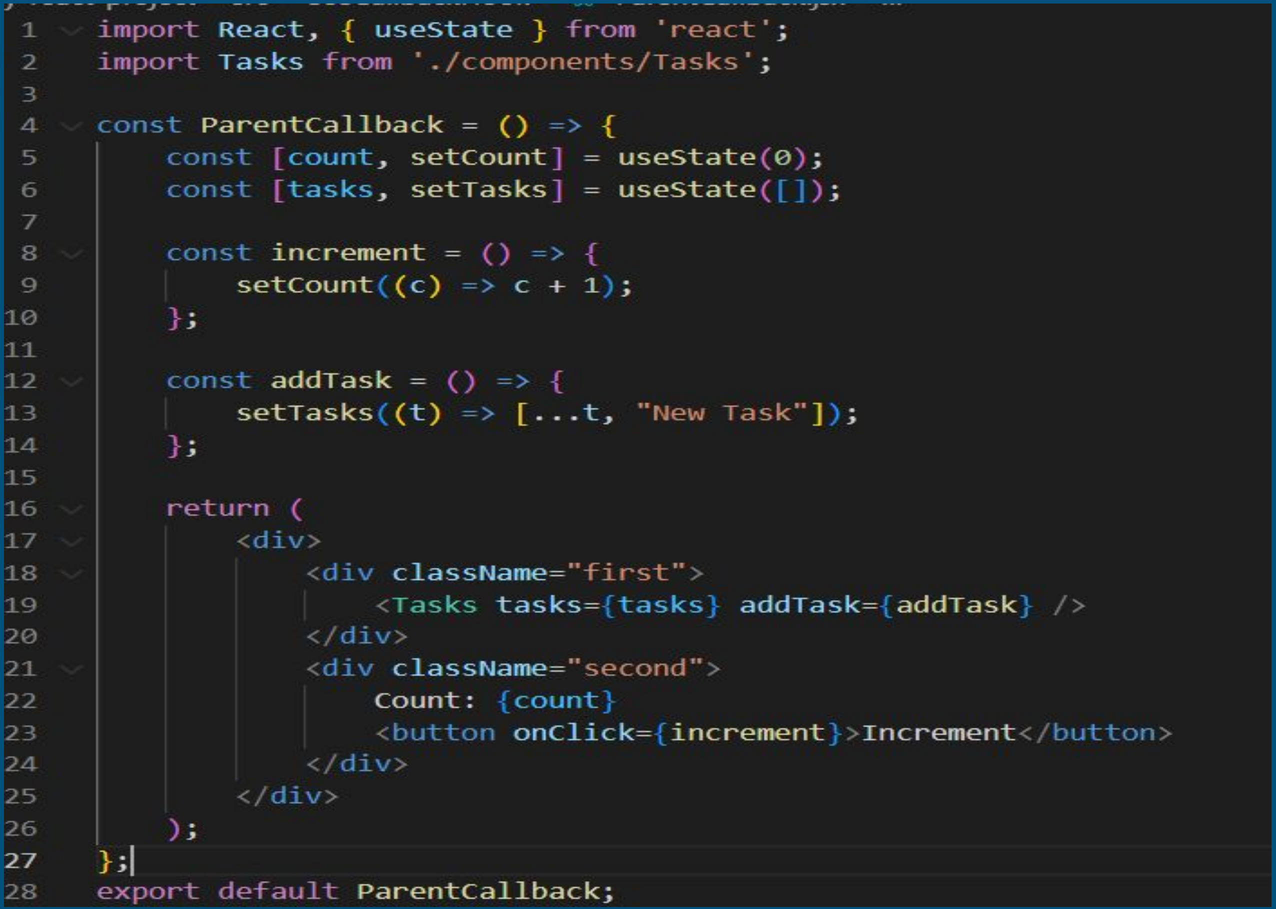
useCallback Hook Usage

1] It improves the performance by reducing the unnecessary computations and providing already stored callback.

2] It is similar to React useMemo Hook, the difference is it returns a callback and useMemo returns a value.

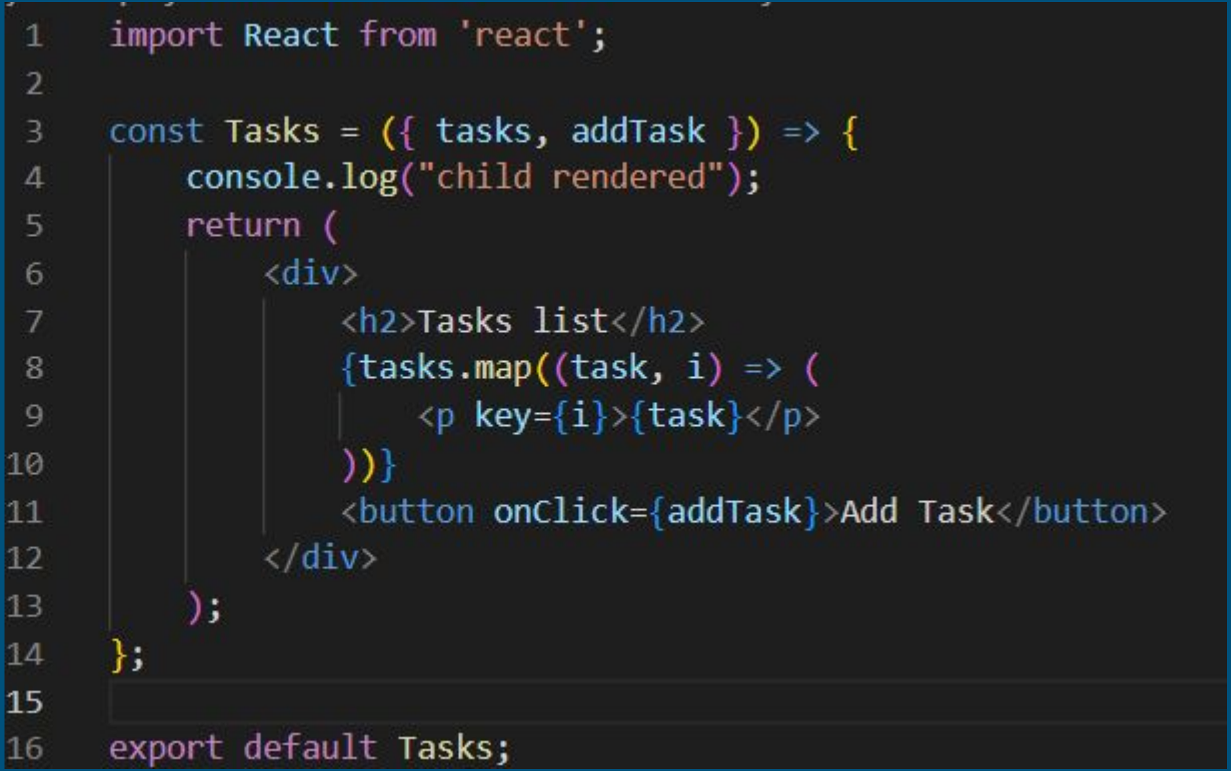
3] This is useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders.

Example : Without using the useCallback



```
import React, { useState } from 'react';
import Tasks from './components/Tasks';

const ParentCallback = () => {
    const [count, setCount] = useState(0);
    const [tasks, setTasks] = useState([]);

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

    const addTask = () => {
        setTasks((t) => [...t, "New Task"]);
    };

    return (
        <div>
            <div className="first">
                <Tasks tasks={tasks} addTask={addTask} />
            </div>
            <div className="second">
                Count: {count}
                <button onClick={increment}>Increment</button>
            </div>
        </div>
    );
};
export default ParentCallback;
```

Example : Without using the useCallback



```
import React from 'react';

const Tasks = ({ tasks, addTask }) => {
    console.log("child rendered");
    return (
        <div>
            <h2>Tasks list</h2>
            {tasks.map((task, i) => (
                <p key={i}>{task}</p>
            ))}
            <button onClick={addTask}>Add Task</button>
        </div>
    );
};

export default Tasks;
```

Explanation (Without useCallback)

ParentCallback Component:

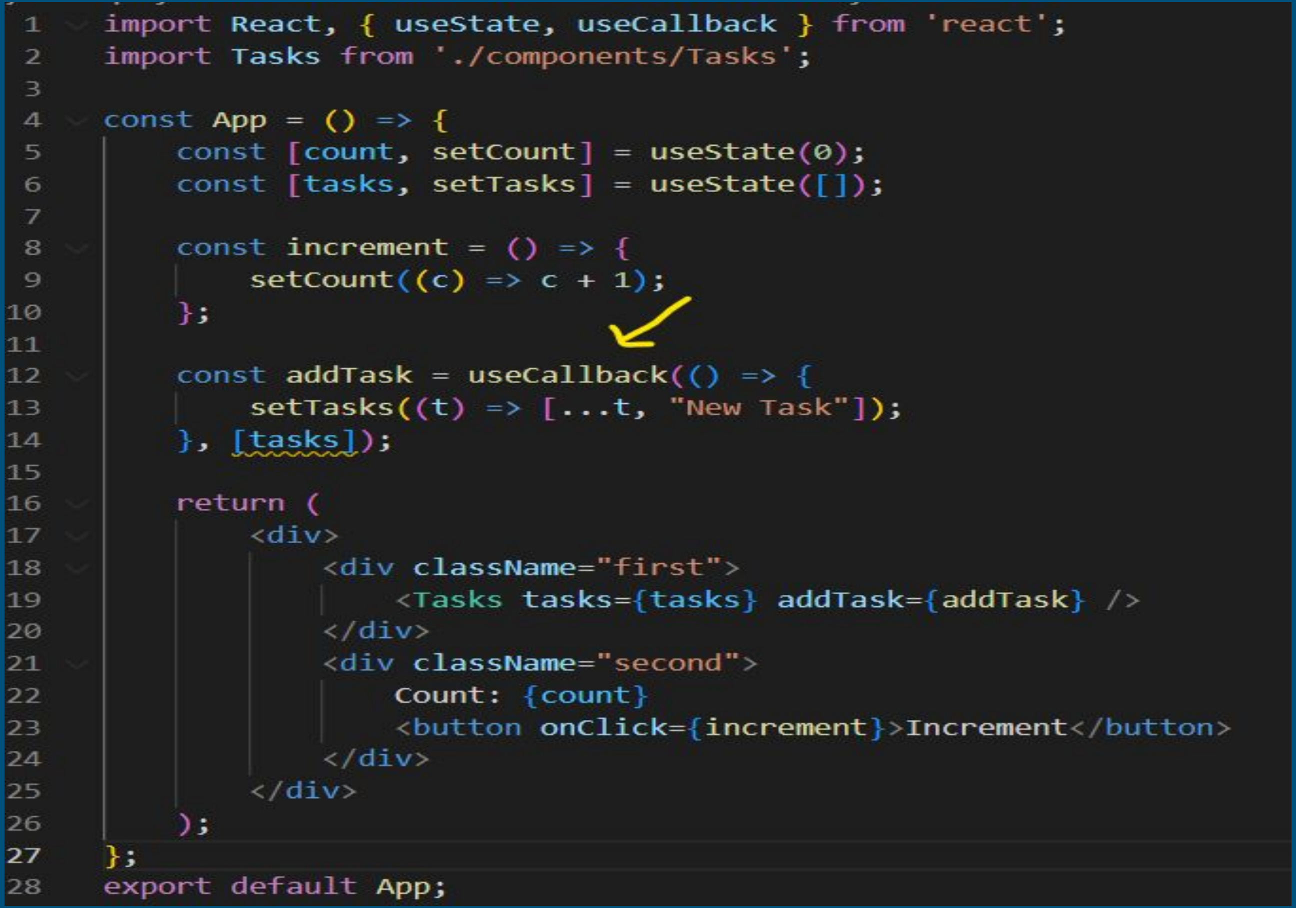
- count and tasks are state variables.
- increment is a function to increase the count by 1.
- addTask is a function to add a new task to the list.
- Tasks component is rendered, passing tasks and addTask as props.

Tasks Component:

- Receives tasks and addTask as props.
- Displays the list of tasks.
- Logs "child rendered" each time it re-renders.
- Has a button to add a new task using addTask.

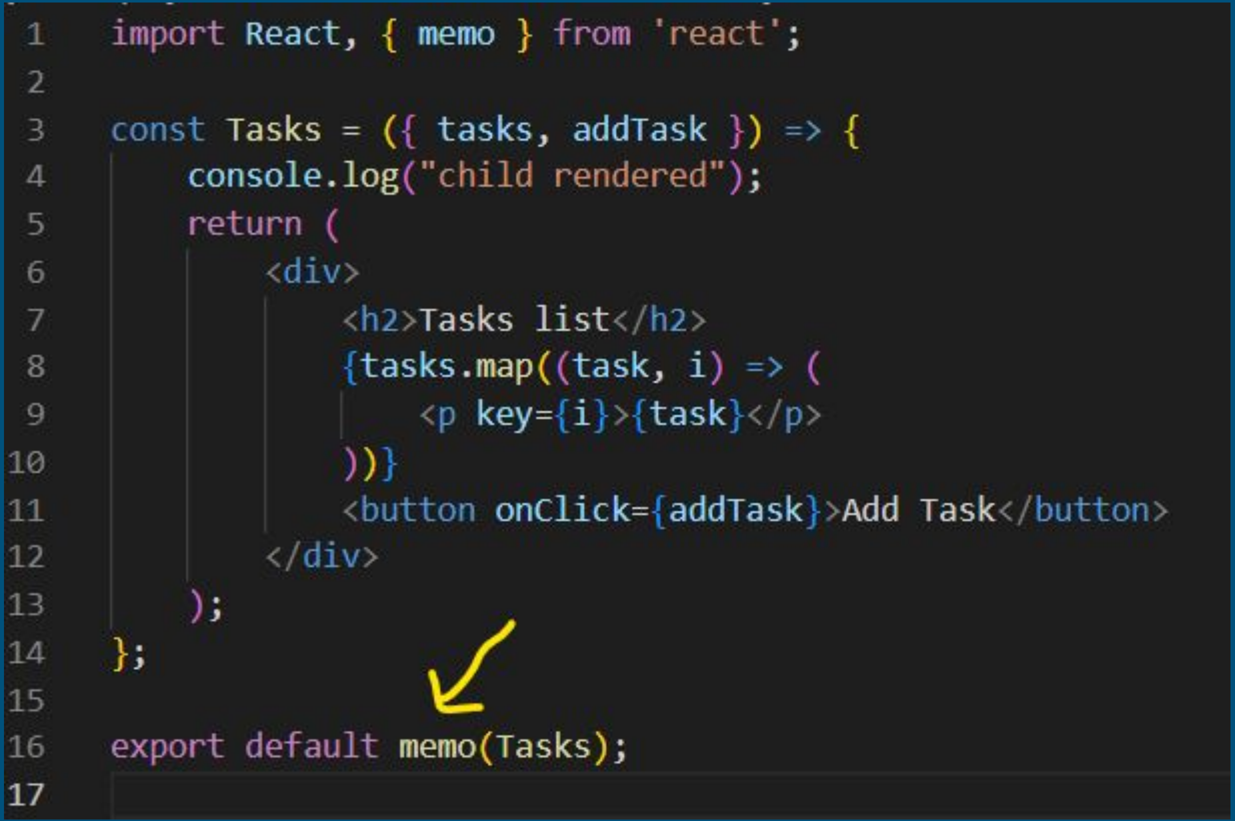
Problem: Every time ParentCallback re-renders (e.g., when count changes), the addTask function is recreated, causing Tasks to re-render unnecessarily.

Example : Using React useCallback



```jsx
import React, { useState, useCallback } from 'react';
import Tasks from './components/Tasks';

const App = () => {
    const [count, setCount] = useState(0);
    const [tasks, setTasks] = useState([]);

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

    const addTask = useCallback(() => {
        setTasks((t) => [...t, "New Task"]);
    }, [tasks]);

    return (
        <div>
            <div className="first">
                <Tasks tasks={tasks} addTask={addTask} />
            </div>
            <div className="second">
                Count: {count}
                <button onClick={increment}>Increment</button>
            </div>
        </div>
    );
};
export default App;
```

Example : Using React useCallback



```
import React, { memo } from 'react';

const Tasks = ({ tasks, addTask }) => {
    console.log("child rendered");
    return (
        <div>
            <h2>Tasks list</h2>
            {tasks.map((task, i) => (
                <p key={i}>{task}</p>
            ))}
            <button onClick={addTask}>Add Task</button>
        </div>
    );
};

export default memo(Tasks);
```

Explanation (With useCallback)

ParentCallback Component:

- useCallback is used to memoize the addTask function. This means addTask will only change if its dependencies (tasks) change.
- increment function and state variables are the same as before.
- Tasks component is rendered the same way, but now addTask is memoized.

Tasks Component:

  - memo is used to memoize the entire component. This means Tasks will only re-render if its props (tasks or addTask) change.
  - Everything else is the same as before.

With useCallback and memo:

- The addTask function is memoized, so it's not recreated on every render.
- Tasks component is memoized, so it only re-renders if tasks or addTask change.
- This reduces unnecessary re-renders, improving performance.

CODEMIND
Your Success Partner

Thank you

CODEMIND
Your Success Partner