

COMP6080

🌟 ReactJS - useMemo, Memo, useCallback

Author(s): Hayden Smith



[\(Download as PDF\)](#)

Re-Renders

- Everytime a useState hook state is updated, where that hook exists and the components under it will be re-evaluated and re-rendered.
- Even if part of your code doesnt use the updated hook, it still would be re-rendered by ReactJS.
- Re-render components can get very slow.



UseMemo + Memo + useCallback

- **useMemo** is a ReactJS hook that allows you to cache a result for subsequent re-renders if you know it isn't changing constantly.
- **memo** is a ReactJS higher order component that allows you to memoise a component based on certain conditions.
- **useCallback** is a ReactJS hook that allows you to cache a function definition for subsequent re-renders if you know it isn't changing constantly.



UseMemo

If you store a value in a `useMemo` instead of a `useState`, When ReactJS goes to re-render the component, and it sees a `useMemo`, ReactJS will skip evaluating it and just return the cached value.

```
1 const App = () => {
2   const [toolTipShow, setToolTipShow] = useState(false);
3
4   const filteredList = useMemo(() => {
5     filteredListOfItmes(list, filter);
6   }, []);
7
8   return (
9     <>
10      <ToolTip set={setToolTipShow} show={toolTipShow} />
11      <List items={filteredList}>
12    </>
13  );
14 };
```

`useMemo1.js`



UseMemo - When Use?

- When there is a considerable delay in re-renders.
- Use the ReactJS debugger to find the expensive component or function causing the rendering delay.
- Example: You have a large list of items, and one main tooltip that activates when you hover over a list item. Using memo on the list of items will prevent them from being unnecessarily re-rendered when you hover.

UseMemo - Invalidation

- The useMemo hook works similar to useEffect, it has parameters you pass into it that should be a react use state result. This means that you can trigger the cache to update when and if only the states you pass into useMemo update.

```
1 const filteredList = useMemo(() => {  
2     filteredListOfItmes(list, filter);  
3 }, [filter, list]);
```

[useMemo2.js](#)

Side Effects

Using memoisation without understanding the side effects can cause your ReactJS app to not update when you expect it to update during a state update.



Memo

- The `memo()` higher order component that you wrap your react components with, has a function which has access to the old props passed into the component and the new props.
- This allows you to only update the component when a certain prop updates and not any prop updates. Returning `false` from this function means you want react to update the cache and re-render the component



Memo

```
1 import { memo, useState } from 'react';
2
3 const Greeting = memo(function Greeting({ name }) {
4   console.log("Greeting was rendered at", new Date().toLocaleTimeString());
5   return <h3>Hello{name && ', '}{name}!</h3>;
6 });
7
8 export default function MyApp() {
9   const [name, setName] = useState('');
10  const [address, setAddress] = useState('');
11  return (
12    <>
13      <label>
14        Name{'': ''}
15        <input value={name} onChange={e => setName(e.target.value)} />
16      </label>
17      <label>
18        Address{'': ''}
19        <input value={address} onChange={e => setAddress(e.target.value)} />
20      </label>
21      <Greeting name={name} />
22    </>
23  );
24 }
```

memo1.js



Memo

- (Optionally) This allows you to only update the component when a certain prop updates and not any prop updates. Returning false from this function means you want react to update the cache and re-render the component

```
1 memo(..., (prevProps, nextProps) => true);
```

[memo2.js](#)



UseCallback

- `useCallback` is similar to `useMemo` exception it is used for functions instead of values

```
1 const MyComponent = ({ onClick }) => {
2   return <button onClick={onClick}>Click Me</button>;
3 };
4
5 const ParentComponent = () => {
6   const [count, setCount] = useState(0);
7
8   // Without useCallback, a new function is created on every render
9   const handleClick = () => {
10     setCount(count + 1);
11   };
12
13   // With useCallback, handleClick will only be recreated if 'count' changes
14   const memoizedHandleClick = useCallback(() => {
15     setCount(count + 1);
16   }, [count]);
17
18   return <MyComponent onClick={memoizedHandleClick} />;
19 };
```

`useCallback1.js`

Feedback



Or go to the [form here](#).

