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 evaulating 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 their 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 an 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

```jsx
import { memo, useState } from 'react';

const Greeting = memo(function Greeting({ name }) {
  console.log("Greeting was rendered at", new Date().toLocaleTimeString());
  return <h3>Hello{name && ', '}{name}!</h3>;
});

export default function MyApp() {
  const [name, setName] = useState('');
  const [address, setAddress] = useState('');
  return (
    <>
      <label>
        Name{': '}
        <input value={name} onChange={e => setName(e.target.value)} />
      </label>
      <label>
        Address{': '}
        <input value={address} onChange={e => setAddress(e.target.value)} />
      </label>
      <Greeting name={name} />
    </>
  );
}
```

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

```jsx
const MyComponent = ({ onClick }) => {
  return <button onClick={onClick}>Click Me</button>;
};

const ParentComponent = () => {
  const [count, setCount] = useState(0);

  // Without useCallback, a new function is created on every render
  const handleClick = () => {
    setCount(count + 1);
  };

  // With useCallback, handleClick will only be recreated if 'count' changes
  const memoizedHandleClick = useCallback(() => {
    setCount(count + 1);
  }, [count]);

  return <MyComponent onClick={memoizedHandleClick} />;
};
```

useCallback1.js

🦻 Feedback

Or go to the form here.