

BROWSER OPT.

→ Website optimization is a process of improving & website's loading speed in the browser.

What is Memoization?

- **Memoization** is a speed optimization technique in programming, where given a function, you return a cached version of the output if the same inputs are used.
- A memoized function **remembers** the results of an output for a given set of inputs.
- In **React**, we can memoize components (where the inputs are props), functions, or just a regular computed value.
- When memoizing components, it does a **shallow** comparison (by default) of the props, and if it sees that it has changed, then it will re-render.
- Memoization is not free. You are trading space for time.

→ There are basically three reasons after react comp. is re-render

→ { state is changed
→ { props is changed
→ } if any change occurs in react parent's comp that trigger re-render for child comp }

React.memo()

→ React memo is higher-order comp. that wraps around a component to memoize the rendered output and avoid unnecessary rendering. This improves performance because it memoizes the result and skips rendering to recall the last render result.

There are two ways you can wrap your component with React.memo.

```

const myComponent = React.memo((props) => {
    /* render using props */
});

export default myComponent;

```

Another option is to create a new variable to store the memoized component and then export the new variable:

```

const myComponent = (props) => {
    /* render using props */
};

export const MemoizedComponent = React.memo(myCom

```

This is the example of React.memo()

React.memo() Example

```

12:30 PM Wed 25 Oct
 1 Import React, {memo} from 'react';
 2 Import './Child.css';
 3
 4 const Child = props => {
 5     const [counter] = props;
 6
 7     console.log("Child is Rendering!!!");
 8
 9     return (
10         <div className="Child">
11             <h1>Child - Counter: </h1>
12         </div>
13     );
14 }
15
16 export default memo(Child);
17

```

Local: http://localhost:3000/
On Your Network: http://192.168.0.10:3000/
Note that the development build is not optimized.
To create a production build, use `yarn build`.

Child

10

```

12:30 PM Wed 25 Oct
 1 import React, {useState} from 'react';
 2 Import './App.css';
 3 Import Child from './Child';
 4
 5 function App() {
 6     const [counter, setCounter] = useState(0);
 7
 8     return (
 9         <div className="App">
10             <h1>Hello from Parent: <counter></h1>
11             <button onClick={() => setCounter(counter + 1)}>+1</button>
12             <Child counter={counter}></Child>
13         </div>
14     );
15 }
16
17 export default App;
18

```

Local: http://localhost:3000/
On Your Network: http://192.168.0.10:3000/
Note that the development build is not optimized.
To create a production build, use `yarn build`.

Hello from Parent: 20

Child - 20

Parents



2J

```

1 import React, {useState} from 'react';
2 import './App.css';
3 import Child from "../Child";
4
5 function App() {
6   const [counter, setCounter] = useState(0);
7   const [input, setInput] = useState('');
8
9   return (
10     <div className="App">
11       <h1>Hello from Parent: {counter}</h1>
12       <button onClick={() => setCounter(counter + 1)}>+1</button>
13       <input type="text" onChange={e => setInput(e.target.value)} value={input} />
14       <Child counter={counter} updateCounter={() => setCounter(counter + 1)} />
15     </div>
16   );
17 }
18
19 export default App;
20

```

Run: npm start

Local: http://localhost:3000/
On Your Network: http://192.168.0.10:3000/

Note that the development build is not optimized.
To create a production build, use `yarn build`.

Hello from Parent: 6

+1

Child - 6

Elements Console Sources

[HMR] Waiting for update signal from WDS... log.js:24
Child is Rendering!!! index.js:8
Child is Rendering!!! index.js:8

→ When we giving any value inside input field at that time Child is rendering because it is find the new reference of function....
So, here you have to memorize the function in React.

For memorizing the function we have to use `useCallback()`.

This hook is used to optimize a React application by returning a memoized function which helps to prevent unnecessary re-rendering of a function. This hook stores the cached value of the function and only updates the function if the passed dependencies changes.

Syntax

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

```
import React, {useCallback, useState} from 'react';
import './App.css';
import Child from './Child';

function App() {
  const [counter, setCounter] = useState(0);
  const [input, setInput] = useState('');

  const updateCounterFromChild = useCallback(() => setCounter(counter + 1), [counter]);
  const updateCounter = useCallback(() => setCounter(counter + 1), [counter, setCounter]);

  return (
    <div className="App">
      <h1>Hello from Parent: {counter}</h1>
      <button onClick={() => updateCounterFromChild()}>+1</button>
      <input type="text" onChange={e => setInput(e.target.value)} value={input} />
    </div>
  );
}

export default App;
```

Local: http://localhost:3000/
On Your Network: http://192.168.0.10:3000/
Note that the development build is not optimized.
To create a production build, use `yarn build`.

Karen B

```
import React, {memo} from 'react';
import './Child.css';

const Child = props => {
  const [counter, updateCounter] = props;

  console.log("Child is Rendering!!!");

  return (
    <div className="Child">
      <h1>Child - {counter}</h1>
      <button onClick={updateCounter}>Click</button>
    </div>
  );
}

export default memo(Child);
```

Local: http://localhost:3000/
On Your Network: http://192.168.0.10:3000/
Note that the development build is not optimized.
To create a production build, use `yarn build`.

Child

useMemo()

useMemo() Hook

`useMemo` will only recompute the memoized value when one of the dependencies has changed. This optimization helps to avoid expensive calculations on every render.

→ useMemo is same like useCallback

The screenshot shows a development setup with two main windows. On the left is WebStorm IDE displaying the file `index.js` which contains code for a child component that uses `useMemo` to memoize a complex calculation. On the right is a browser window showing the result of the code execution. The browser title is "Hello from Parent: 14". Inside a box, it says "Child - 500000000 14" with a "Click" button below it. The browser's developer tools console tab shows logs: "[HMR] Waiting for update signal from WDS...", "Child is Rendering!!!", and "14 Child is Rendering!!!". The browser address bar shows "localhost:3000".

```
12:49 PM Wed 25 Oct
WebStorm File Edit View Navigate Code Refactor Run Tools VCS Window Help
memorization | src > Bi Child > index.js
memorization (~WebStormProjects/memoization) .../src/Child/index.js
AppIndex.js ChildIndex.js
1 import React, {memo, useMemo} from 'react';
2 import './Child.css';
3
4 const Child = props => {
5   const {counter, updateCounter} = props;
6
7   const childNumber = useMemo(()=>{
8     let output = 0;
9     for (let i=0; i < 500_000_000; i++) {
10       output++;
11     }
12     return output;
13   }, [1]);
14
15   console.log("Child is Rendering!!!");
16
17   return (
18     <div className="Child">
19       <h1>Child - {childNumber} {counter}</h1>
20       <button onClick={updateCounter}>Click</button>
21     </div>
22   );
23 };
24
25
CMD+shift+U
Run: npm start x
Local: http://localhost:3000/
On Your Network: http://192.168.0.10:3000/
Note that the development build is not optimized.
To create a production build, use yarn build.
Run: npm start x
File: index.js
12:49 PM Wed 25 Oct
React App
Hello from Parent: 14
Child - 500000000 14
Click
Elements Console Sources
[HMR] Waiting for update signal from WDS... log.js:24
Child is Rendering!!! index.js:16
14 Child is Rendering!!! index.js:16
>
```

More details about second argument in `React.memo()`

The screenshot shows a browser window with multiple tabs open. The active tab is titled "redux" and contains the code for a child component named `Child`. The component logs a message when it renders and returns a `div` containing an `h1` element and a button with the `onClick` handler set to `incHandler`. The browser's address bar shows "reduxpratice".

```
JS App.js M JS index.js ...store U JS index.js ...src M JS counterReducerjs U Counter.jsx U Parents.jsx U Child.jsx U redux U
reduxpratice > src > Child.jsx > ...
1 import React from 'react'
2
3 function Child({count,incHandler}) {
4   console.log('hello i am child components')
5   return (
6     <div>
7       <h1>hello child count:{count.inc}</h1>
8       <br/>
9       <button onClick={incHandler}>incfromchild</button>
10    </div>
11  )
12}
13
14 //if you are return true means its not re-render (it means previous and current props is same only) so if you able to return false then
15 //it will re-render (because its understand both props is different)
16 //this function is executed every time if it return false then jsx is render otherwise if this function result true on that time jsx is
17 //not re-render
18 function comHandler(prevProps,nextProps)
19 {
20   console.log('preprops',prevProps)
21   console.log('nextprops',nextProps)
22   console.log('this for value',prevProps.count.inc==nextProps.count.inc)
23   console.log('this is for the handler',prevProps.incHandler==nextProps.incHandler)
24   console.log('result is',prevProps.count.inc==nextProps.count.inc && prevProps.incHandler==nextProps.incHandler)
25   return prevProps.count.inc==nextProps.count.inc && prevProps.incHandler==nextProps.incHandler
26 }
```

→ If function return true
that time re-render is not
happening, if it false
then re-render is happening