import classes from './DemoOutput.module.css'

import React from 'react';

const DemoOutput = (props) => {

    console.log("DemoOutput.js")

    return (<>{props.show && <h1>Hello World </h1>}</>)

    ;

  };

  export default React.memo(DemoOutput);

To avoid unnecessary reevaluation use React.memo(DemoOutput);

import React from 'react';

import classes from './Button.module.css';

const Button = (props) => {

  return (

    <button

      type={props.type || 'button'}

      className={`${classes.button} ${props.className}`}

      onClick={props.onClick}

      disabled={props.disabled}

    >

      {props.children}

    </button>

  );

};

export default React.memo(Button);

but in this case it will not work because it was onclick function which act as object , in javascript object === object is false. So it will re-render.

import React, { useState,useCallback } from 'react';

import './App.css';

import DemoOutput from './components/Demo/DemoOutput';

import Button from './components/UI/Button/Button';

function App() {

  const[showOutput,SetShowOutput]=useState(false)

  const changeToogler =useCallback(()=>{

    SetShowOutput((prev)=>{

      return !prev

    })

  },[])

  console.log("App.js")

  return (

    <div className="app">

      <center><Button onClick={changeToogler}>Toogler</Button></center>

      <h1>Hi there!</h1>

      <DemoOutput show={false}></DemoOutput>

    </div>

  );

}

export default App;

we will use useCallBack() for avoid this issue. This is as same as below example how it handle this object.

let obj1 = {}

let obj2 = {}

obj1 ===obj2 // return false

now

obj1 = obj2

obj1 === obj2 //return true

useCallBack(()=>{

},[**dependencies**])

Closure: [Closures - JavaScript | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures)

useCallback with dependencies:

import React, { useState,useCallback } from 'react';

import './App.css';

import DemoOutput from './components/Demo/DemoOutput';

import Button from './components/UI/Button/Button';

function App() {

  const[showOutput,SetShowOutput]=useState(false)

  const[allowTogler,SetAllowTogler]=useState(false)

  const changeToogler =useCallback(()=>{

    if(allowTogler){

      SetShowOutput((prev)=>{

        return !prev

      })

   }

  },[allowTogler])

  const changeAllowToogler =useCallback(()=>{

    SetAllowTogler((prev)=>{

      return !prev

    })

  },[])

  console.log("App.js")

  return (

    <div className="app">

       <center><Button onClick={changeAllowToogler}>Allow Toogler</Button></center>

      <center><Button onClick={changeToogler}>Toogler</Button></center>

      <h1>Hi there!</h1>

      <DemoOutput show={false}></DemoOutput>

    </div>

  );

}

export default App;

state schedulers:

setstate will update the state when all are reevaluated.

Set state scheduled in state scheduler ,if multiple states for same are updating then it takes latest state. But in real time we can’t replicate this.

useMemo():

useMemo() with dependencies hook is used to avoid re-rendering some line of code which re-execute because of other state/props change in same function.

If its array and object, we can use useMemo() in both parent and child.

In summary, the **useEffect** hook is used to perform side effects in a React component, while the **useMemo** hook is used to optimize the performance of a component by memoizing the results of a calculation or function.