In react, component data flow from top to bottom. data cannot flow both direction.

So it is become problem for data handling. For solve this problem, we various method.

Lifting state:

In lifting solution, all state handle by top level component. Then with props we send handler and state to bottom component.

In view:



Here component A export component B, C. here component B take input but cannot send to component C. So, it is a problem to send data to other component. to solve this, we use lifting state up

A

-State

-method

C

-props.state

-props.method

B

-props.state

-props.method

Here all state and function are in top lavel component A. then with props component sent to B and C. it is called lifting state up.

Composition vs Inheritance

Problem of inheritance:

1. In inheritance, react child component extends all parent component method. If child want to use single method of parent component it not possible.
2. component is tightly coupled.
3. from child it not clear what parents does.

Pattern of share State of react

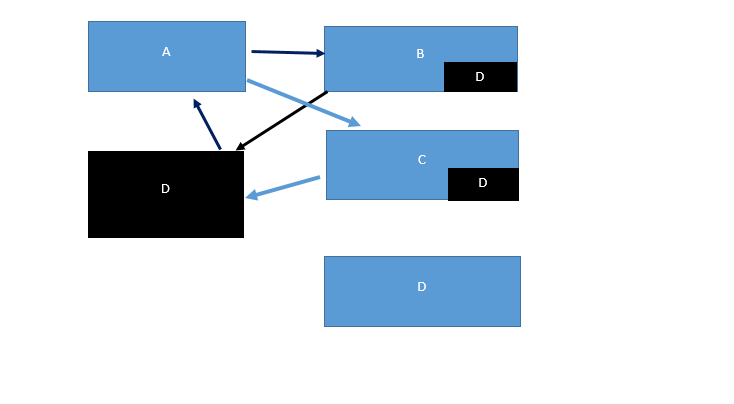
To share same functionality in all component there are two patent

1. higher order component
2. props rendering

Higher-Order Components

 higher-order component is a function that takes a component and returns a new component.

const EnhancedComponent = higherOrderComponent(WrappedComponent);

when use similar function in various component, we use higher-order component. we can solve it by lifting state up. but in react some time lifting state up is not good for component. so we use HOC in react.

We create new component with a state, that use in various component. Then we pass all the component in new state component and become other component.

Use state

1. hook use in top level of code.
2. hook use in react function only.

Syntax:

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

Example:

import React, { useState } from 'react';

function Example() {

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

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

Use Effect

React are works for –

1. render UI

2. React on user input and action

3. Render JSX

4. manage state & props

5. Evaluate State and props change

React also do various work

1. fetching data from API

2. updating Dom

3. setting any Subscription or timer

This type of work called side effect

The side effect handle in class component with the method of-

1. componentDidMount ()

2.componentDidUpdate ()

3.componentWillUpdate ()

Here is some problem to use this method –

1. repeating code

2. unorganized code

In useEffect we solve all problem in functional component.

1. Help us perform side Effect in functional components

2. Solves all the problem of lifecycle methods in class component

3. we do not repeat the code.

Use Effect is a function which run every render. We write a function inside useEffect for our purpose.

**Syntax:**

useEffect ( peram1 , peram2 );

param1: take a function.

useEffect ( () => { document.title = `You clicked ${count} times` ; } );

param2:

1. setState variable , for which useEffect called or not.

2. if param2 = [ **]** then useEffect called only one time.

useEffect(() => {

document.title = `You clicked ${count} times`;

}, **[]** );

3. useEffect function also return a function. That do as like componentWillUnmount.

useEffect(() => {

document.title = `You clicked ${count} times`;

return ()=>{

}

}, **[]** );

**Example:**

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

function Example() {

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

useEffect(() => { document.title = `You clicked ${count} times`; });

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

React Memo

1. memo use in component.

2. memo use for skip rendering a component if its props have not changed.

Example :

import { memo } from "react";

const Todos = ({todos}) => {

console.log ("child render");

return (

<>

<h2>My Todos</h2>

{ todos.map((todo, index) => {

return <p key={index}>{todo}</p>;

}) }

</>

);

};

export default memo(Todos);

ref problem in memo:

In react function are reference type data. When we called a function, the function reference is change. So that react new render function is new function. We solve by usecallback function .

useCallback

The React useCallback Hook returns a memoized callback function.

Syntex:

const addTodo = useCallback( param\_1, param\_2 );

param1: take a function.

useCallBack ( () => { document.title = `You clicked ${count} times` ; } );

param2:

1. dependency variable , for which useCalled called or not.

Example:

import { useState, useCallback } from "react";

import ReactDOM from "react-dom/client";

import Todos from "./Todos";

const App = () => {

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

const [todos, setTodos] = useState([]);

const increment = () => {

setCount((c) => c + 1);

};

const addTodo = useCallback( () => {

setTodos((t) => [...t, "New Todo"]);

}, [todos] );

return (

<>

<Todos todos={todos} addTodo={addTodo} />

<hr />

<div>

Count: {count}

<button onClick={increment}>+</button>

</div>

</>

);

};

useMemo

The React useMemo Hook returns a memoized value. The useMemo Hook only runs when one of its dependencies update.

Syntex:

const addTodo = useMemo( param\_1, param\_2 );

param1: take a function.

useCallBack ( () => { document.title = `You clicked ${count} times` ; } );

param2:

1. dependency variable , for which useCalled called or not.

Example:

import { useState, useMemo } from "react";

const App = () => {

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

const [todos, setTodos] = useState([]);

const calculation = useMemo( ()=> {

for (let i = 0; i < 1000000000; i++) {

num += 1;

}

return num;

}, [count]);

const increment = () => {

setCount((c) => c + 1);

};

const addTodo = () => {

setTodos((t) => [...t, "New Todo"]);

};

return (

<div>

<div>

<h2>My Todos</h2>

{ todos.map((todo, index) => {

return <p key={index}>{todo}</p>;

}) }

<button onClick={addTodo}>Add Todo</button>

</div>

<hr />

<div>

Count: {count}

<button onClick={increment}>+</button>

<h2>Expensive Calculation</h2>

{calculation}

</div>

</div>

);

};

useRef

useReducer

1. The useReducer Hook is similar to the useState Hook.
2. It allows for custom state logic.

Syntax

The useReducer Hook accepts two arguments.

useReducer(<reducer>, <initialState>)