**==========================================================**

[**https://shorturl.at/sMS17**](https://shorturl.at/sMS17)

**Hooks**

**useState**

**useEffect**

**useReducer**

**useCallback**

**useMemo**

**==========================================================**

Hooks

- Hook is an advanced part in ReactJS.

- It allows us to use advanced react features without writing a class.

- It is having simplicity as good as functional

components and features as good as class components.

- Hooks are introduced in February 2019 v16.8

- To work with hooks we must use functional components.

- Hooks wont work in class components.

- Hooks make code more readable.

Rule :-

- Hooks are declared before their use, implies declare at the top.

- Hooks are declared inside component.

Create react application

>create-react-app hookapp

switch to application

>cd hookapp

download boostrap

>yarn add boostrap --save

execute application

>yarn start

==================================================

useState hook

==================================================

- state is not supported by functional components.

- to work with state useState hook is used.

Eg01

- create a class component

- declare state variable count = 0

- on button click increment and decrement values

\*\*\*Eg01.js\*\*\*

import React from 'react'

export default class Eg01 extends React.Component {

constructor() {

super()

this.state = {

count: 0

}

}

render() {

return (

<div>

<button className='btn btn-outline-primary p-3' onClick={this.dec}> - </button>

<button className='btn btn-success mx-2 btn-lg'>{this.state.count} </button>

<button className='btn btn-outline-primary p-3' onClick={this.inc}> + </button>

</div>

)

}

inc = () => {

this.setState({

count: this.state.count + 1

})

}

dec = () => {

this.setState({

count: this.state.count - 1

})

}

}

- implement the same using functional components

import { useState } from "react"

export default function Eg01() {

const [count, setCount] = useState(0)

return (

<div>

<h1 className='text-primary'>{count}</h1>

<button className="btn btn-outline-info px-5 m-3"

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

<button className="btn btn-outline-info px-5 m-3"

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

</div>

)

}

Eg02

useState with objects

- Spot the issue

\*\*\*Eg02.js\*\*\*

import { useState } from "react";

export default function Eg02() {

const [name, setName] = useState({

fname: 'Fname',

lname: 'Lname'

})

return (

<div className="container mt-5">

<input type="text" placeholder="Enter First Name"

onChange={e => setName({ fname: e.target.value })} />

<input type="text" placeholder="Enter Last Name"

onChange={e => setName({ lname: e.target.value })} />

<h3>First Name:- {name.fname}</h3>

<h3>Last Name:- {name.lname}</h3>

</div>

)

}

- issue is the 'useState' hook also can not preserve previous state.

- so do it manually using spred operator(...), as

onChange={e => setName({...name, fname: e.target.value })}

onChange={e => setName({...name, lname: e.target.value })}

\*\*\*Eg03.js\*\*\*

import { useState } from "react";

export default function Eg03() {

const [times, setTimes] = useState([])

let lap = () => {

let today = new Date()

setTimes([

...times,

{

id: times.length,

value: today.getMinutes() + ":" + today.getSeconds() + ":" + today.getMilliseconds()

}

])

}

return (

<div className="container mt-5">

<button onClick={lap} className="btn btn-outline-warning"><b>LAP </b> </button>

<ol>

{

times.map(item => (

<li className="m-2 p-2" key={item.key}>{item.value} </li>

))

}

</ol>

</div>

)

}

Conclusion:-

- This hook lets us add state in functional components.

- In class component, the state must be an object.

- With useState hook, state must not be an object.

- The useState hook returns an array of two elements.

- First element is the current value of state.

- Second element is the state setter method.

Note:- when dealing with objects or arrays, always use

'spread' operator with state.

==================================================

useEffect Hook

==================================================

- Consider the requirement

- provide custom title to a page instead of 'React App'

- title should be button clicked 'clicked x times'

- this is done by two lifecycle methods, componentDidMount() and

componentDidUpdate().

- Consider this class component code

import React from 'react'

export default class Eg01 extends React.Component {

constructor() {

super()

this.state = {

count: 0

}

}

componentDidMount() {

document.title = `Clicked ${this.state.count} times`

}

componentDidUpdate() {

document.title = `Clicked ${this.state.count} times`

}

render() {

return (

<div>

<button onClick={

() => this.setState({ count: this.state.count + 1 })

}>

Click Me

</button>

<br />

Clicked {this.state.count} times

</div>

)

}

}

- now we will do this with functional component

import { useEffect, useState } from "react";

export default function Eg01() {

const [count, setCount] = useState(0)

useEffect(() => {

document.title = `Clicked ${count} times`

})

return (

<div>

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

</div>

)

}

Conditional Rendering

import React from 'react'

export default class Eg02 extends React.Component {

constructor() {

super()

this.state = {

count: 0,

name: ''

}

}

componentDidMount() {

document.title = `Clicked ${this.state.count} times`

}

componentDidUpdate(prevProps, prevState) {

if (prevState.count != this.state.count)

console.log('Component updated')

document.title = `Clicked ${this.state.count} times`

}

render() {

return (

<div>

<button onClick={

() => this.setState({ count: this.state.count + 1 })

}>

Click Me

</button>

<br />

Clicked {this.state.count} times

<br />

<input type='text' onChange={e => { this.setState({ name: e.target.value }) }}></input>

</div>

)

}

}

import { useEffect, useState } from "react";

export default function Eg01() {

const [count, setCount] = useState(0)

const [name, setName] = useState('')

useEffect(() => {

console.log('useEffect Called')

document.title = `Clicked ${count} times`

}, [count])

return (

<div>

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

<br />

<input type="text" value={name} onChange={e => setName(e.target.value)}></input>

</div>

)

}

- Execute only once, no further update

- Using class component - coponentDidMount()

import React from "react";

export default class Eg03 extends React.Component {

constructor() {

super()

this.state = {

x: 0,

y: 0

}

}

logMouse = e => {

this.setState({

x: e.clientX,

y: e.clientY

})

}

componentDidMount() {

window.addEventListener('mousemove', this.logMouse)

console.log(this.state)

}

render() {

return (

<div>

X - {this.state.x} ... Y - {this.state.y}

</div>

)

}

}

import { useEffect, useState } from "react"

export default function Eg03() {

const [x, setX] = useState(0)

const [y, setY] = useState(0)

const logMouse = e => {

setX(e.clientX)

setY(e.clientY)

}

useEffect(() => {

console.log('useEffect called')

window.addEventListener('mousemove', logMouse)

},[])

return (

<div>

X - {x} ... Y - {y}

</div>

)

}

API Calls - axios

>yarn add axios --save

use url :- https://jsonplaceholder.typicode.com/

//fetch all data

import { useEffect, useState } from "react";

import axios from 'axios'

export default function Eg04() {

const [posts, setPosts] = useState([])

useEffect(() => {

axios.get("https://jsonplaceholder.typicode.com/posts")

.then((posRes) => {

setPosts(posRes.data)

}, (errRes) => {

console.log(errRes)

})

}, [])

return (

<div>

<table className="table table-bordered table-warning w-50 mx-auto table-striped table-hover text-primary">

<thead>

<tr >

<th>Sr no</th>

<th>UserID</th>

<th>Title</th>

</tr>

</thead>

<tbody>

{

posts.map((e, i) => (

<tr>

<td className="text-primary">{i + 1} </td>

<td className="text-danger"> {e.userId} </td>

<td className="text-info">{e.title} </td>

</tr>

))

}

</tbody>

</table>

</div>

)

}

//fetch Single

import axios from "axios"

import { useEffect, useState } from "react"

export default function Eg05() {

const [post, setPost] = useState([])

const [id, setId] = useState(1)

const [nid, setNid] = useState(1)

const fetchData = () => {

setPost([])

setNid(id)

}

useEffect(() => {

axios.get(`https://jsonplaceholder.typicode.com/posts/${id}`)

.then((posRes) => {

setPost(posRes.data)

}, (errRes) => {

console.log(errRes)

})

}, [nid])

return (

<div>

<input type='number'

placeholder="Enter id"

onChange={e => setId(e.target.value)}></input>

<button onClick={fetchData}>Fetch</button>

<h4>{JSON.stringify(post)} </h4>

</div>

)

}

==================================================

useReducer Hook

==================================================

//Data fetching using useState and useEffect

import { useState, useEffect } from 'react'

import axios from 'axios'

export default function Eg01() {

const [data, setData] = useState([])

const [loading, setLoading] = useState(true)

const [error, setError] = useState('')

useEffect(() => {

axios.get('https://restcountries.com/v2/all')

.then((posRes) => {

setLoading(false)

setData(posRes.data)

setError('')

}, (errRes) => {

setLoading(false)

setData([])

setError('Error Occured:- ' + errRes)

})

}, [])

return (

<div>

{loading ? 'Loading' : JSON.stringify(data)}

<br />

{error ? error : null}

</div>

)

}

//Data fetching using useReducer

import { useEffect, useReducer } from "react"

import axios from 'axios'

const initialState = {

data: [],

loading: true,

error: ''

}

const reducer = (state, actions) => {

switch (actions.type) {

case 'SUCCESS':

return {

loading: false,

data: actions.payload,

error: ''

}

case 'ERROR':

return {

loading: false,

data: [],

error: 'Error occured' + actions.payload

}

default:

return state

}

}

export default function Eg01() {

const [state, dispatch] = useReducer(reducer, initialState)

useEffect(() => {

axios.get('https://restcountries.com/v2/all')

.then((posRes) => {

console.log(posRes)

dispatch({ type: 'SUCCESS', payload: posRes })

}, (errRes) => {

console.log(errRes)

dispatch({ type: 'ERROR', payload: errRes })

})

}, [])

return (

<div>

{state.loading ? 'Loading' : JSON.stringify(state.data.data)}

<br />

{state.error ? state.error : null}

</div>

)

}

Spot differences in above two examples

- useState hook can handle simple data

- useReducer can handle complex data

==========================================

useCallback hook (Performance Optimization Eg01)

==========================================

- See the performance issue in the code

- here we passed functions as props.

Directory structure.

useCallback

- ParentCompo.js

- Title.js

- Count.js

- Button.js

\*\*\*Title.js\*\*\*

import React from "react"

function Title(){

console.log('Rendering Title')

return(

<h2>

useCallBack Hook

</h2>

)

}

//export default Title

//////////01

export default React.memo(Title)

\*\*\*Count.js\*\*\*

import React from "react"

function Count({ text, count }) {

console.log(`Redering ${text}`)

return <div>{text} - {count} </div>

}

//export default Count

//////////01

export default React.memo(Count)

\*\*\*Button.js\*\*\*

import React from "react"

function Button({ handleClick, children }) {

console.log('Rendering Button - ', children)

return (

<button onClick={handleClick}>

{children}

</button>

)

}

//export default Button

//////////01

export default React.memo(Button)

\*\*\*ParentCompo.js\*\*\*

import { useCallback, useState } from "react"

import Title from "./Title"

import Count from "./Count"

import Button from "./Button"

function ParentComponent() {

const [age, setAge] = useState(25)

const [salary, setSalary] = useState(50000)

/\*//////////02

const incrementAge = () =>{

setAge(age + 1)

}

const incrementSalary = () =>{

setSalary(salary + 1000)

}

\*/

//////////02

const incrementAge = useCallback(() => {

setAge(age + 1)

}, [age])

const incrementSalary = useCallback(() => {

setSalary(salary + 1)

}, [salary])

return (

<div>

<Title />

<Count text='Age' count={age} />

<Button handleClick={incrementAge}>incrementAge</Button>

<Count text="Salary" count={salary} />

<Button handleClick={incrementSalary}>Increment Salary</Button>

</div>

)

}

export default ParentComponent

Note:-

1. Here if we observe the console, at each button click all components are rendering.

2. To limit this rendering use React.memo

3. Now spot the issue, still there is rerendering of Button component

4. Again to optimise this we will use the 'useCallback' hook.

5. it calls function only if the dependent value i.e. second argument to useCallback changes.

6. Now the issue is solved.

==========================================

useMemo hook (Performance Optimization Eg02)

==========================================

import { useMemo, useState } from "react";

export default function Eg01() {

const [counterOne, setCounterOne] = useState(1)

const [counterTwo, setCounterTwo] = useState(1)

const incrementOne = () => {

setCounterOne(counterOne + 1)

}

const incrementTwo = () => {

setCounterTwo(counterTwo + 1)

}

/\*

const isEven = () => {

console.log('Even called')

let i = 0

while (i < 1000000000)

i++

return counterOne % 2 === 0

}

\*/

const isEven = useMemo(() => {

console.log('Even called')

let i = 0

while (i < 1000000000)

i++

return counterOne % 2 === 0

}, [counterOne])

return (

<div>

<button onClick={incrementOne}>Count 1 - {counterOne} </button>

<br /><br />

{/\*<span>{isEven() ? 'Even' : 'Odd'} </span>\*/}

<span>{isEven ? 'Even' : 'Odd'} </span>

<br /><br />

<button onClick={incrementTwo}>Count 2 - {counterTwo} </button>

</div>

)

}

- looking at the code above, useCallback and useMemo are working the same.

- when function to be returned go with useCallback

- and when value to be returned go with useMemo.

==============================================

Higher Order Components

==============================================

- Create Cloths component with functionality

to calculate total + 18% tax

- Create the same component Food with 10% tax

Directory Structure

<>

src

HOCeg

- Cloths.js

- Food.js

- myComponent.js

\*\*\*Cloths.js\*\*\*

import React from 'react'

export default class Cloths extends React.Component {

constructor() {

super()

this.state = {

final: 0

}

}

render() {

return (

<div>

<h1 style={{ color: 'blue' }}>Welcome to Cloths Department</h1>

<form className='w-25 mx-auto' onSubmit={this.calculate}>

<div className='form-group'>

<label>Qunatity</label>

<input className='form-control'

type='number'

placeholder='Qunatity'

name='qty'></input>

</div>

<div className='form-group'>

<label>Rate</label>

<input className='form-control'

type='number'

placeholder='Rate'

name='rate'></input>

</div>

<input type='submit' value='Calculate' className='btn btn-primary my-2'></input>

<h3 style={{ color: 'navy' }} >Total amount:- {this.state.final}</h3>

</form>

</div>

)

}

calculate = (e) => {

e.preventDefault()

let tax = 18

let total = e.target.qty.value \* e.target.rate.value

let final = total + total \* tax / 100

this.setState({ final })

}

}

Similarly create Food.js

\*\*\*myComponent.js\*\*\*

import React from 'react'

import Cloths from "./Cloths";

import Food from './Food';

export default class MyComponent extends React.Component{

render(){

return(

<div>

<Cloths/>

<Food/>

</div>

)

}

}

--------------------

Here problem is that, the code is duplicated.

Def. Higher Order Components is an advanced technique in

ReactJS for reusing component functionality.

In this technique we pass component as argument and

it returns a component.

Eg

const EnhancedComponent = HigherOrderComponent(WrappedComponent)

create TaxCalc -> HOC

\*\*\*TaxCalc.js\*\*\*

import React from 'react'

const TaxCalc = (WrappedComponent, mytax, dept) => {

class TaxCalc extends React.Component {

constructor() {

super()

this.state = {

final: 0

}

}

render() {

return (

<div>

<h1 style={{ color: 'blue' }}>Welcome to {dept} Department</h1>

<form className='w-25 mx-auto' onSubmit={this.calculate}>

<div className='form-group'>

<label>Qunatity</label>

<input className='form-control'

type='number'

placeholder='Qunatity'

name='qty'></input>

</div>

<div className='form-group'>

<label>Rate</label>

<input className='form-control'

type='number'

placeholder='Rate'

name='rate'></input>

</div>

<input type='submit' value='Calculate' className='btn btn-primary my-2'></input>

<WrappedComponent final={this.state.final} />

</form>

</div>

)

}

calculate = (e) => {

e.preventDefault()

let tax = mytax

let total = e.target.qty.value \* e.target.rate.value

let final = total + total \* tax / 100

this.setState({ final })

}

}

return TaxCalc

}

export default TaxCalc

update Cloths and Food components as

\*\*\*Cloths.js\*\*\*

import React from 'react'

import TaxCalc from './TaxCalc'

class Cloths extends React.Component {

render() {

return (

<div>

<h3 style={{ color: 'navy' }} >Total amount with tax:- {this.props.final}</h3>

</div>

)

}

}

export default TaxCalc(Cloths, 18, 'Cloths')