React-Hooks

---

1)useState
2)useEffect
3)useContext
4)useReducer
5)useCallback
6)useMemo
7)useRef

8)useImperativeHandle
9)useLayout
10)useDebugValue
11)custom Hooks

Introduction :
==========

i)What

- Hooks are a new feacture addition in React version 16.8 which allows you to use React features without having to write a class.

  ex :state of a component
- Hook's don't work inside classes.

ii)Why

#Reason 1

- Understand how this keyword works in javascript
- Remember to bind event handlers in class components

#Reason 2

- There is no particular way to reuse stateful component logic
- HOC and render props patterns to address this problem

  (You have restructure components for HOC,Render props but it makes code harder to follow)
- There is need to share strateful logic in a better way

#Reason 3

- Create components for complex scenarios such as datafetching and subscribing to events.
- Related code is not organized in one place

  Ex :DataFetching -in componentDidMount and componentDidUpdate

  Event listeners -in componentDidMount and componentWillUnmount
- Above DataFetching and Event listeners both are written in componentDidMount
- Because of stateful logic -Cannot break components into smaller ones

iii)To Use React hooks

- React version 16.8 or higher
- Can't use hooks inside of a class component
- Hook's don't replace your existing knowledge of react concepts
- Instead,Hook's provide a more direct API to the React concepts you already know.

iv)Advantages

- Hook's are a new feature addition in React version 16.8
- They allow you to use React features without having to write a class
- Avoid whole confusion with 'this' keyword
- Allow you to reuse stateful logic

• Organize the logic inside a component into resuabe isolated units

v)Five Important Rules for Hooks

Before we create our own Hook, let's review a few of the major rules we must always follow.

• Never call Hooks from inside a loop, condition or nested function
• Never call a Hook from a regular function

• Hooks should sit at the top-level of your component
• Only call Hooks from React functional components
• Hooks can call other Hooks

1)useState

=======

• Hook uses useState() functional component for setting and retrieving state.
• useState is used for state management
ex : const [state,setState]=useState(initialVal); where initialVal=0

#Imp Points :

• The useState hook lets you to add state to functional components
• In classes,the state is always an Object
• But ,with the useState hook,the state doesn't have to be an Object.(It can be number,string,array,Object..)
• The useState hook returns an array with 2 elements
• The first element is the current value of the state,and the second element is a state setter function
• If New state value depends on the previous state value..? You can pass a function to the setter function.
• When dealing with Objects or arrays,always make sure to spread your state variable and then call the setter function.

2)useEffect

========

• The Effect hook lets you to perform side effects (an action) in functional components.
• It is a close replacement for componentDidMount,componentDidUpdate,componentWillUnmount.
• It does not use class components lifecycle methods .

It uses in ..

• Updating the DOM,
• Fetching and consuming data from a server API,
• Setting up a subscription, etc.

#syntax :

```
useEffect(() => {
    effect
    return () => {
        cleanup
    }
}, [input])
```

Note :

#Run effects only once

• useEffect(() => {
effect}, [] ==componentDidMount

#Initial run+Conditionally run effects

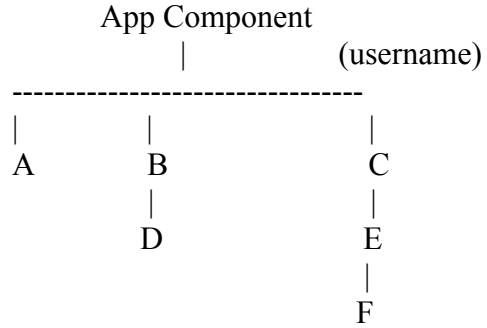
• useEffect(() => {

effect}, [dependencies]) ==componentDidUpdate

#cleanup code
- useEffect(() => {
  effect return () => {
    cleanup
  }}) ==componentWillUnmount

3)useContext
=========
•Context provides a way to pass data through the component tree without having to pass props manually at every level

#component tree



• If you want to pass data (ex :username Comp C-to -F) ,Actually we can pass props manually at every level like C-E,E-F.
•Instead of passing props manually at every level we can use context

Steps
i)Create the context
ii)Provide a context value
iii)Consume the context value

i)Create the context

```
//#Step :01
const UserContext=React.createContext()

const UserProvider=UserContext.Provider

const UserConsumer=UserContext.Consumer
```

ii)Provide a context value

```
 {/* Step:02 */}
     <UserProvider value="Viru">
       <C/>
   </UserProvider>
```

iii)Consume the context value

```
 //Step:03
 <UserConsumer>
             {
               (username)=>{
                  return <h2>Hello {username}....!</h2>
               }
```

```
            }
</UserConsumer>
```

- But ,in hooks first two steps common,third step is like...

```
const username= useContext(UserContext)
const channel= useContext(ChannelContext)
```

4)useReducer
==========

- useReducer is a hook that is used for state management
- It is an alternative to useState
- useState is built using useReducer

(In mind ...? What is difference between useState and useReducer)

#Hooks so far
- useState -state
- useEffect -side effects
- useContext -context API
- useReducer -reducers

#Reduce method in js
- The reduce() method executes a reducer function (that you provide) on each element of the array, resulting in single output value.

syntax :
---
array.reduce(reducer, 5)
where reducer function accepts two values
const reducer = (accumulator, currentValue) => accumulator + currentValue;

ex :
```
const array1 = [1, 2, 3, 4];
const reducer = (accumulator, currentValue) => accumulator + currentValue;
array.reduce(reducer, 5) //15
```

#reduce vs useReducer

[reduce in javascript	|| useReducer in react]
• array.reduce(reducer,initialVal)	•useReducer(reducer,initialState)
• singleVal=reducer(accumulator,currentVal)	•newState =reducer(currentState,action)
•reduce method return a single value	• useReducer returns a pair of values [newState,dispatch]

Imp Points :
- •useReducer is a hook that is used for state management in React
- •useReducer is realted to reducer function
- •useReducer(reducer,initialState)
- •reducer(currentState,action)
- •It return newState and Dispatch
- •Where action is instructor to reducer to which action to be perform

#useReducer with useContext
- •useReducer -Local state management
- •Share state between components -Global state management (useReducer+useContext)

#What is difference between useState and useReducer

Scenario	useState	useReducer
•Type of state	Number,String,boolean	Object or Array
•Number of state transitions	one or two	Too many
•Related state transitions	No	Yes
•Business logic	No	complex business logic
•Local vs Global	Local	Global

5)useCallback
= =========
i)what
•useCallback will return a memoized version of the callback that only changes if one of the dependencies has changed.
ii)Why
•This is useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders (e.g. shouldComponentUpdate)
•It caches function itself
syntax :

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

For more Reference :https://kentcdodds.com/blog/usememo-and-usecallback

6)useMemo
= =======
•The useMemo is a hook used in the functional component of react that returns a memoized value
•It caches function return value
```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

7)useRef
======
• useRef is used access DOM nodes
```
const refContainer = useRef(initialValue);
```
•useRef returns a mutable ref object whose .current property is initialized to the passed argument (initialValue).

ex:
```
inputEl.current.focus();
```

• The useRef Hook is a function that returns a mutable ref object whose . ... current property is initialized with the passed argument (initialValue).
The returned object will persist for the full lifetime of the component

11)custom Hooks
= ==========

- A custom Hook is basically a javascript function whose name starts with 'use'
- A custom hook can also call other hooks if required

why ?
- Share logic -Alternative HOC's and Render props