2/19/25, 1:31 PM OneNote

Day-19

Agenda: Hooks, Form validation, axios (server side)

Tuesday, February 18, 2025 9:05 AM

Functional Components Vs Class Components

| Import react, (component) from 'react' [class classname extends Component] { return(); // html } } Export default language extends Component (| Functional Components | Class Components |
|---|--|--|
| Function function_name() { return(); //html } } Export default <functions, +1;="" -="" -useeffect(()="" 1.="" 2.="" advanced="" and="" apart="" are="" as="" call="" complex="" component="" components="" cons[count,="" consid.,="" data="" demount="" different="" each="" from="" functional="" functions="" have="" here="" hooks="" hooks:="" implement="" is="" it="" keyword.="" lifecycle="" logic="" maintains="" manage="" mount="" mount,="" of="" or="" own="" private="" provide="" render="" setcount]="useState(0);" setid="" some="" state="" state,="" stateful="" stateless="" the="" their="" this="" to="" types="" ui="" used="" useeffect="" usestate="" using="" we="" well=""> {}, []) Updating / render - useEffect(() => {}, []) Unmounting - useEffect(() => { return () => {};}, []) Unmounting - useEffect(si used to management without using redux Advanced Hooks: 4. useRef 5. useContext is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo</functions,> | · | · |
| return(); //html } Export default <function_name> Render()</function_name> | Function function_name() | |
| return(); //html } Export default <function, name=""> Render() { return(); //html } } Export default <classname> Simple Functions, a part from this we have some advanced functions as well and we call it as HOOKS Stateless Stateful Complex to implement UI Logic Different types of Hooks: 1. useState 1. useState 1. useState 1. useState 1. useState 2. useEffect 1. used to manage the each functional component state Consigld, setId = useState(0); Consigld, setEffect 1. used to mount, render or demount the components Mount -useEffect(() => {}, []) Updating / render - useEffect(() => {}, [dependency]) Unmounting - useEffect(() => { return () => {};}, []) 3. useContext 1. useEffect 1. useRef 1. useRef 1. useRef 2. useRef 3. useContext 3. useRef 4. useRef 5. useCallback 6. useMeno Hooks are functions that let you to use State and lifecycle features in functional</classname></function,> | { | · |
| Export default <unuers feturn();="" html="" td="" ="" }<=""><td>return(); // html</td><td>, and the second second</td></unuers> | return(); // html | , and the second |
| return(); //html } Simple Functions , apart from this we have some advanced functions as well and we call it as HOOKS Stateful It maintains their own private data - state , here we are using this keyword . It provide lifecycle hooks | } | Render() |
| Simple Functions Simple Func | Export default <function_name></function_name> | |
| Simple Functions, apart from this we have some advanced functions as well and we call it as HOOKS Stateless Mainly responsible for the UI Different types of Hooks: 1. useState Is used to manage the each functional component state constcount, setCount] = useState(2); setId = 1; setId = 1; 2. useEffect (() => {}, []) Updating / render - useEffect(() => {}, [dependency]) Unmounting - useEffect(() => { return () => {};}, []) Unmounting - useEffect (() => { return () => {};}, []) Advanced Hooks: 4. useRef | | return(); //ntmi } |
| it as HOOKS stateless Stateful Mainly responsible for the UI Different types of Hooks: 1. useState | | }Export default <classname></classname> |
| stateless Mainly responsible for the UI Complex to implement UI Logic setState method to update sused to manage the each functional component state cons[count, setCount] = useState(0); Consplid , setId] = useState(2); setId +1; useEffect Mount -useEffect(() => {}, []) Updating / render - useEffect(() => {}, , []) Unmounting - useEffect(() => { return () => {};}, []) Unmounting - useEffect(() => { return () => {};}, []) 3. useContext is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | Simple Functions , apart from this we have some advanced functions as well and we call | 1 |
| Mainly responsible for the UI Different types of Hooks: 1. useState Is used to manage the each functional component state cons[count, setCount] = useState(0); Contolid, setId] = useState(2); setEffect Is used to mount, render or demount the components Mount -useEffect(() => {}, []) Updating / render - useEffect(() => {}, [dependency]) Unmounting - useEffect () => { return () => {};}, []) 3. useContext Is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | | |
| Different types of Hooks: 1. useState Is used to manage the each functional component state cons[count, setCount] = useState(0); Cons[id, setId] = useState(2); setId +1; 2. useEffect Is used to mount, render or demount the components Mount -useEffect(() => {} , []) Updating / render - useEffect(() => {} return () => {} ;}, []) Unmounting - useEffect(() => { return () => {} ;}, []) 3. useContext Is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | stateless | |
| 1. useState Is used to manage the each functional component state cons[count, setCount] = useState(0); Cons[id , set(d] = useState(2); set(d + 1; 2. useEffect | Mainly responsible for the UI | |
| Is used to manage the each functional component state cons[count , setCount] = useState(0); Cons[id , setId] = useState(2); setId +1; 2. useEffect is used to mount, render or demount the components Mount -useEffect(() => {} , []) Updating / render - useEffect(() => {} , [dependency]) Unmounting - useEffect(() => { return () => {} ;}, []) 3. useContext is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | Different types of Hooks : | setState method to update |
| cons[count , setCount] = useState(0); Cons[id , setId] = useState(2); setId +1; 2. useEffect is used to mount, render or demount the components Mount -useEffect(() => {} , []) Updating / render - useEffect(() => {} , [dependency]) Unmounting - useEffect(() => { return () => {} ;}, []) 3. useContext is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | | |
| 2. useEffect is used to mount, render or demount the components Mount -useEffect(() => {} , []) Updating / render - useEffect(() => {} , [dependency]) Unmounting - useEffect(() => { return () => {} ;}, []) 3. useContext is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | cons[count , setCount] = useState(0); | |
| 2. useEffect is used to mount, render or demount the components Mount -useEffect(() => {} , []) Updating / render - useEffect(() => {} , [dependency]) Unmounting - useEffect(() => { return () => {} ;}, []) 3. useContext is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | Cons[id , setId] = useState(2); | |
| is used to mount, render or demount the components Mount -useEffect(() => {} , []) Updating / render - useEffect(() => {} , [dependency]) Unmounting - useEffect(() => { return () => {} ;}, []) 3. useContext is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | | |
| Mount -useEffect(() => {} , []) Updating / render - useEffect(() => {} ,[dependency]) Unmounting - useEffect(() => { return () => {} ;}, []) 3. useContext is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | | |
| Updating / render - useEffect(() => {} ,[dependency]) Unmounting - useEffect(() => { return () => {} ;}, []) 3. useContext is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | | |
| Updating / render - useEffect(() => {} ,[dependency]) Unmounting - useEffect(() => { return () => {} ;}, []) 3. useContext is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | Mount -useEffect(() => {} , []) | |
| 3. useContext is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | | |
| is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | Unmounting - useEffect(() => { return () => {} ;}, []) | |
| is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | | |
| is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | | |
| is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | | |
| is used for global state management without using redux Advanced Hooks: 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | 3 useContext | |
| 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | | |
| 4. useRef 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | Advanced Hooks | |
| 5. useCallback 6. useMemo Hooks are functions that let you to use State and lifecycle features in functional | | |
| Hooks are functions that let you to use State and lifecycle features in functional | 5. useCallback | |
| | 6. useMemo | |
| components without writing a class | Hooks are functions that let you to use State and lifecycle features in functional | |
| • | components without writing a class | |

Props & state

Props -- pass an information from one component to another components

State -- It's like a central place for storing the data

So Hook is the new introduction through which we can store the data and it gives state $% \left(1\right) =\left(1\right) \left(1\right) \left$

State is a secret place so if you modify or edit the data in a state we always use setState method for that

useEffect

| | СС | FC |
|------------|------------------------|---|
| Mount | componentDidMount() | useEffect(() => {} , []) |
| Updating | componentDidUpdate() | useEffect(() => {} ,[dependency]) |
| Unmounting | componentWillUnmount() | useEffect(() => { return () => {} ;}, []) |

```
import React ,{useEffect,useState} from "react";
const StockPriceTracker = () => {
  const[stockPrice , setStockPrice] = useState(100); // initial stock Price
```

```
//first case -- Mounting when the component is added to the DOM (first renders)
         useEffect(
              () =>
                           //mounting
                          console.log("Component Mounted called only once");
                            const interval = setInterval( () => {
  const newPrice = (Math.random() * (105 - 95) + 95).toFixed(2);
  console.log(`Stock price Update : ${newPrice}`);
                             setStockPrice(newPrice) // triggerin a re-ender
                            } , 2000)
// unmounting
                           return() => {
  clearInterval(interval);
  console.log("Component unmounted")
                            }, []
                      );
                      return (
                          <div>
                               <h2> Stock Price : {stockPrice} </h2>
                          </div>
export default StockPriceTracker;
```

Optimization

use Ref: is use to track the previous search without re-endering. We create a reference to DOM elements or stores values without triggering re-enders . It is also used in accessing DOM elements and storing mutable values <math display="block">volume volume value va

If we want to access the input field without re-endering .

useCallBack() -- Memoizing the functions

Which is used to prevent unnecessary re-creations of the functions in child components

2/19/25, 1:31 PM OneNote