SOFTVIRA

THE ULTIMATE CHEAT SHEET FOR REACT HOOKS

softvira.com

useState :

- This hook is used to manage state in functional components.
- It takes an initial state value and returns an array with two elements: the current state value and a function to update the state.

EXAMPLE:

```
Import React , { useState } from 'react'  ;

function Counter () {
    const [ count , setCount ] = useState (0)
    return (
      <div>
        <p>You clicked {count} times</p>
        <button onClick= {() => setCount (count + 1 ) }>
            Click me
        </button>
      </div>
    );
}
```

useEffect:

- This hook is used to perform side effects in functional components.
- It takes a function as an argument and is called after the component has been rendered.

EXAMPLE:

```
Import React , { useState , useEffect } from 'react'  ;
function Clock  () {
    const [ time , setTime ] = useState (new Date());
     useEffect (() =>{
        const  timer = setInterval (()=> {
            setTimer(new Date () );
        } , 1000 ) ;
        return ()=>clearInterval (timer);
    } , [] ) ;
    return (
        <div>
            <h1> The time is {time.toLocaleTimeString()}</h1>
        </div>
    );
}
```

useContext:

- This hook is used to access a context object created by React.createContext() function.
- It returns the current context value and re-renders the component when the context value changes.

EXAMPLE:

```
import { createContext , useContext } from 'react' ;

const MyContect = createContext ('default value')

function Child ()  {
    const value = useContext(MyContext);
        return <p>Value:  {value}</p>;
}
   function Parent ()  {
      return (
        <Mycontext.Provider value = "hello">
           <Child  />
        <Mycontext.Provider>
       );
    }
```

useReducer:

- This hook is used to manage more complex states in functional components.
- It takes a reducer function and an initial state value and returns an array with two elements: the current state value and a dispatch function to update the state.

EXAMPLE:

```
import React, { useReducer } from ‘react' ;
const initialState = { count: 0 };
function reducer(state, action) {
  switch (action.type) {
    case 'increment' :
      return { count: state.count + 1 };
    case 'decrement' :
      return { count: state.count - 1 };
      default:
        throw new  Error();
    }
  }

  function Counter() {
    const [state, dispatch] = useReducer (reducer, initialState);

  return (
  <div>
  <p>Count: {state.count}</p>
  <button onClick={() => dispatoh({ type: ‘increment’ })}>+</button>
  <button onClick={() => dispatoh({ type: decrement’ })}>-</button>
  </div>
  );
}
```

useCallback:

- This hook is used to memoize a function and avoid unnecessary re-renders of child components.
- It takes a function and an array of dependencies and returns a memoized version of the function.

EXAMPLE:

```
import React, { useCallback, useState } from ‘react’;

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

const increment = useCallback(() => {
      setCount(count + 1);
    }, [count]);

    return (
      <div>
          <p>Count: {count)</p>
          <button onClick  {increnent}>Increment</button>

      </div>

    );
}
```

useMemo:

- This hook is used to memoize a value and avoid unnecessary re-computation of expensive operations.
- It takes a function and an array of dependencies and returns a memoized value.

EXAMPLE:

```
import React, { useMemo, useState } from 'react' ;

function ExpensiveCaloulation({ value }) {
   // This is a placeholder function representing a computationally expensive
   const expensiveCalculation = (n) => {

   let result = 0;
   for (let  = 0; i < n;  i++)  {
       result  += I;

   }
   return result;

  };

      const result = useMemo(() => expensiveCaloulation(value), [valuel);

      return <divsThe Tesult of the expensive calculation is {Tesult}</div>;
      }
      {

      function App() {
           const [value, setValue] = useState(10000);

       return (
      <div>
         <input type="number' value=(value) onChange={(e) => setValue(e.target
         <ExpensiveCaloulation value={value} />
      </div>

   );

}
```

useId:

- The useId hook is used for generating a unique ID in functional components.
- It takes an optional prefix as an argument and returns a unique ID.

EXAMPLE:

```
import { useId } from 'react-id-generator' ;

function MyComponent() {
  const [username, setUsername] = useState ( ' ' ) ;
  const [password, setPassword]  =  = useState ( ' ' ) ;
  const inputId = useId();

  return (
        <form>
           <label htmlFor={inputID}>Usernane:</label>
           <input id={inputID} values ={username} onChange={e =>
   setUsername(e. targ
           <label htmlFor={inputid}>Password:</label>
           <input  Id={inputID} value=(password) onChange= {e => setPassword(e.
    targ

    </form>

   );
}
```

useDeferredValue:

- The useDeferredValue hook is used for deferring the update of a state value until the next render cycle.
- It takes a state value as an argument and returns a deferred state value.

EXAMPLE:

```
import { useState, useDeferredValue } from ‘react' ;

function MyComponent() {
    const [value, setValue] = useState(0);
    const deferredValue = useDererreovalue(value, { timeouts: 500 });
    return (
        <div>
          <p>Value: {value}</p>
          <button onClick ={() => setValue(value + 1)}>Increments</button>
          <p>Deferred Value: {deferredValue}</p>
        </div>

    );
}
```

useTransition:

- The useTransition hook is used for managing the loading state of a component.
- It takes a loading state as an argument and returns a tuple with the current state and a function to set the state.

EXAMPLE:

```
import { useState, useTransition, animated } from 'react-spring’

function MyComponent() {
const [items, setItems] = useState([ ]);
const transitions = useTransition(items,  {
    from: { opacity: 0  } ,
    enter: {opacity: 1 } ,
    leave: { opacity: 0} ,
  });

  return (
    <div>
     <button onClick={() => setTtems([...items, itens.length + 1]})3>Add Item)3>Add
  Item

     {transitions((style, item) => (
        <animated diy style=(style}>{item}</animated.div/>
      ))}
  </div>

   );
}
```

useSyncExternalStore:

- The useSyncExternalStore hook is recommended for reading and subscribing from external data sources (stores)
- It accepts three arguments: a function to register a callback, a function that returns the current value of the store, and an optional function that returns the snapshot used during server rendering.

EXAMPLE:

```
import { useSyncExternalStore } from ‘react' ;

function App() {
  const width = useSyncExternalStore(
    (listener) => {
      window. addEventListener(' resize’, listener);
        return () => {

      window. removeEventListener(' resize’, listener);
        };
    } ,
      () => window.innerWidth
       // () => -1 ,
    ) ;

return <p>Size: {width}</p>;

}

export default App;
```

SOFTVIRA

Aftab Umer

Software Engineer

I HOPE YOU FIND THESE REACT HOOKS CHEAT SHEETS HELPFUL. HAPPY CODING!