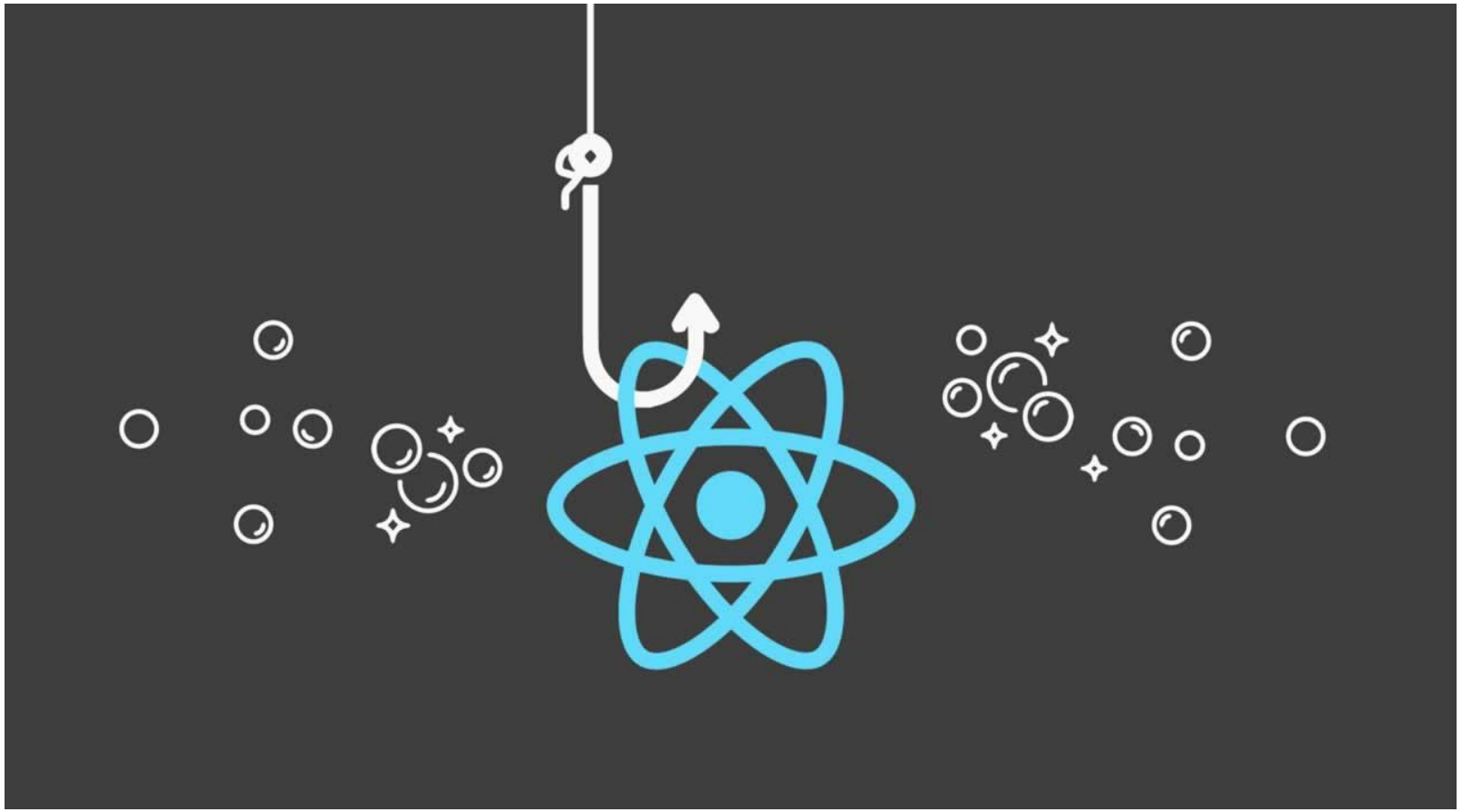


Hooks

let you use state and other React features without writing a class.

Menu

- Why?
- useState
- useEffect
- useContext
- useReducer

Why?

Why Hooks?

We know that components and top-down data flow help us organize a large UI into small, independent, reusable pieces. **However, we often can't break complex components down any further because the logic is stateful and can't be extracted to a function or another component.** Sometimes that's what people mean when they say React doesn't let them "separate concerns."

These cases are very common and include animations, form handling, connecting to external data sources, and many other things we want to do from our components. When we try to solve these use cases with components alone, we usually end up with:

- **Huge components** that are hard to refactor and test.
- **Duplicated logic** between different components and lifecycle methods.
- **Complex patterns** like render props and higher-order components.

We think Hooks are our best shot at solving all of these problems. **Hooks let us organize the logic *inside* a component into reusable isolated units:**

https://medium.com/@dan_abramov/making-sense-of-react-hooks-fdbde8803889

In Closing...

Embracing Hooks as benefited us in the following ways:

- ✔ How we managing state has become easier to reason about
- ✔ Our code is significantly simplified, and more readable
- ✔ It's easier to extract and share stateful logic in our apps.

https://blog.bitsrc.io/why-we-switched-to-react-hooks-48798c42c7f

useState

- useState – get the default value and return two things – the state and the function that changes the state.
- The setXxx() function replace the entire state and not only one of the keys!
- You can use as much useState()'s as you want.
- Pay attention – every setXxx will call the function to render it again. BUT sometimes react will batch them toghther if in the same function

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

function Calculator() {
  const [values, setValues] = useState({ num1: '', num2: '' });
  const [txtRes, setTxtRes] = useState('');

  const btnAdd = () => {
    let res = parseInt(values.num1) + parseInt(values.num2);
    setTxtRes(res);
  }

  const chgNum1 = (num1) => {
    setValues({ num1, num2: values.num2 });
    console.log(111);
    setTxtRes('...');//this will be batched with the setValues call two lines above
  }

return (
    <div>
      num1: <input type="text" value={values.num1} onChange={(e) =>
chgNum1(e.target.value)} /><br />
      num2: <input type="text" value={values.num2} onChange={(e) =>
chgNum2(e.target.value)} /> <br />
      <button onClick={btnAdd}>+</button> <br />
      result={txtRes}
    </div>
  )
}
```

useEffect

- Use for side effects like Data fetching, setting up a subscription, and manually changing the DOM
- By default, it runs both after the first render and after every update. With no dependency array.
- With dependency array…next page

```
//runs when users changed - also the first assignment to []
 //dont need to write this twice - in add user and in delete user!
 useEffect(() => {
   console.log('useEffect users changed! update the db...', users);
 });
```

Like componentDidMount and componentDidUpdate combained

useEffect

- You can think of useEffect Hook as componentDidMount, componentDidUpdate, and componentWillUnmount combined.
- BUT there is a way to distinct between them. Using the optional dependency array and return function

```
//runs when users changed - also the first assignment to []
 //dont need to write this twice - in add user and in delete user!
 useEffect(() => {
   console.log('useEffect users changed! update the db...', users);
 }, [users]);

 //componentDidMount - runs once after the first render
 useEffect(() => {
            …code

   return function cleanUp() { //componentWillUnmount
     console.log('cleaning up...');
   };
 }, []);
```

Like componentDidUpdate

componentWillUnmount

componentDidMount

useContext

- The context and contextprovider
- Here the context definition
- In the next slide using the context

```
import React, { useState, createContext } from 'react'
import uuid from 'uuid';

export const HobbyContext = createContext();

export default function HobbyContextProvider(props) {
 const [hobbies, setHobbies] = useState([
  { id: '1', name: 'Flight', times: 2, icon: 'Flight' },
…
 ]);

 const AddHobby = (name, times) => {
  let newItem = {
   id: uuid(), name, times,
   icon: ['Flight', 'Music', 'Another', 'Sport'][Math.floor(Math.random() * 4)]
  };
  setHobbies([...hobbies, newItem]);
 }

 const RemoveHobby = (id) => {
  setHobbies(hobbies.filter(hob => hob.id !== id));
 }

 return (
  <HobbyContext.Provider value={{ hobbies, AddHobby, RemoveHobby }}>
   {props.children}
  </HobbyContext.Provider>
 )
}
```

useContext

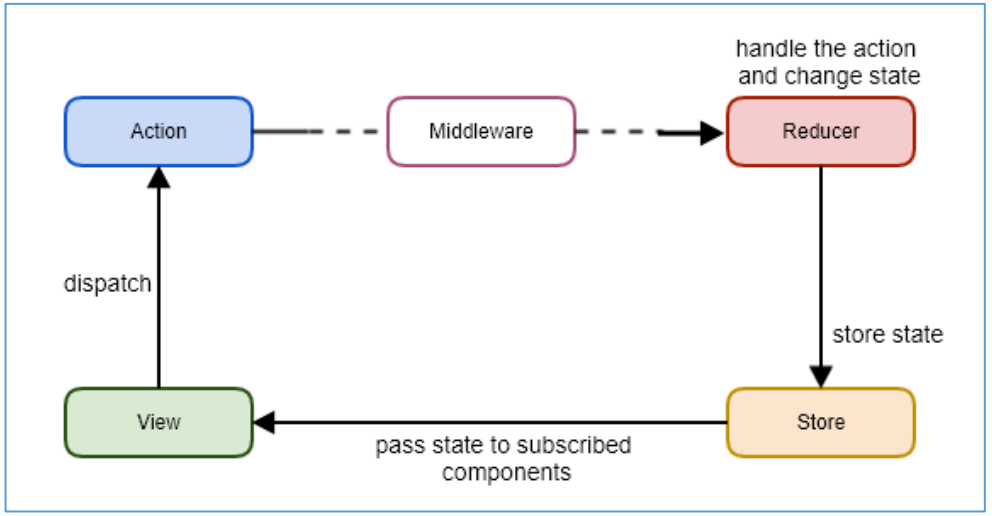
- Using the context

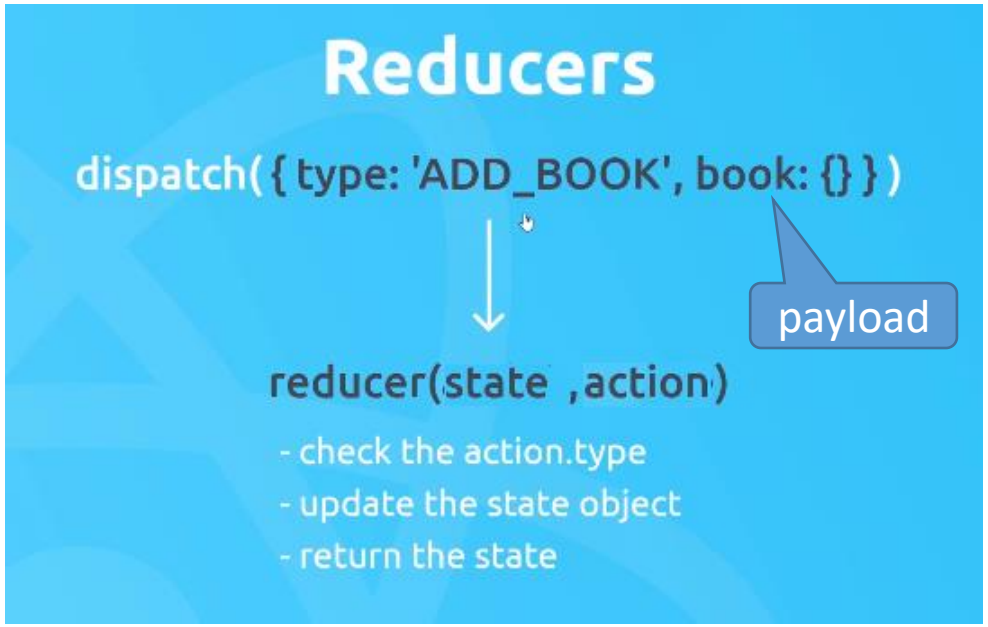
```
import { HobbyContext } from './Contexts/HobbyContext';

export default function HobbiesList() {
  const { hobbies, RemoveHobby } = useContext(HobbyContext);
…
 <IconButton color="secondary" aria-label="delete" onClick={() => { RemoveHobby(hob.id) }}>
…
```

useReducer

- Design pattern
- The user do something in the view, this will call a reducer function through a dispatch of an action.
- Then the state will be changed so the view will be modified





useReducer

- HobbyReducer.js

```
export const ADD_HOBBY = 'ADD_HOBBY';
export const REMOVE_HOOBY = 'REMOVE_HOOBY';

export const HobbyReducer = (state, action) => {
   switch (action.type) {
      case ADD_HOBBY:
         let newHobby = {
            id: uuid(),
            name: action.hobby.name,
            times: action.hobby.times,
            icon: ['Flight', 'Music', 'Another', 'Sport'][Math.floor(Math.random() * 4)]
         };
         return [...state, newHobby];
      case REMOVE_HOOBY:
         return state.filter(hobby => hobby.id !== action.id);
      default:
         return state;
   }
}
```

useReducer

- Using the reducer by dispatch with action.type and action.payload

```
import { ADD_HOBBY } from './Reducers/HobbyReducer’;
…
export default function AddHobby() {
 …
 const { dispatch } = useContext(HobbyContext);

 const btnAddHobby = () => {
  dispatch({ type: ADD_HOBBY, hobby: { name: HobbyName, times: Times } });
 }
```