07. React
Part2

@Pilot - Christine

Attention!

This training is **interview-driven**.

To become a qualified developer, *ChatGPT* and *YouTube* will always be your best friends.

Outline

- Functional Component
- Hooks
- Conditional rendering
- Rendering lists
- Responding to events
- Controlled vs Uncontrolled Component
- HOC
- React.Memo
- Pure function

Function Component

- A function component is a JavaScript function that returns **JSX** and uses React **Hooks** to manage state and lifecycle behavior
  - Written as a plain JavaScript function
  - Uses Hooks (useState, useEffect, etc.) to manage state and side effects
  - NO this keyword
  - Cleaner syntax and easier to read
  - Officially recommended by React

Function Component vs Class Component

Aspect	Function Component	Class Component
Syntax	Function	ES6 Class
State	`useState`	`this.state`
Side effects	`useEffect`	Lifecycle methods
`this`	❌ Not used	✅ Required
Logic reuse	Custom Hooks	HOCs / render props
Code length	Shorter	Longer
Recommendation	✅ Preferred	❌ Legacy

hooks

- **Hooks**: Special functions that let you “hook into” React features (state, lifecycle, context, etc.) from function components
  - Only usable in function components
  - Naming convention: **useHookName**
- Rules of Hooks
  - Only call hooks from react functions
    - Function Components
    - Custom Hooks
  - Only call Hooks at the top level
    - Do **NOT** call Hooks inside loops, conditions, nested functions, or try/catch/finally blocks

State Hooks - useState

- State Management: **Triggers Re-render**
  - Syntax: const [state, setState] = useState(initialValue);
    - state: the current value
    - setState: function to update the value
    - initialValue: state value on the first render
  - Functional Updates: new state depends on the previous state
    - Required when multiple updates happen quickly
    - Multiple updates or updates that depend on state
  - useState with Objects and Arrays
  - **Do not mutate state directly**, always use setter function

State Hooks - useReducer

- Manage **complex state logic** in components, It's similar to useState, but it's better when:
  - Complex state objects
  - Many state updates depend on each other
  - Want Redux-style logic without Redux
- Syntax:
  - **const [state, dispatch] = useReducer(reducer, initialState)**
    - initialState: the default state value
    - state: the current state
    - dispatch: Function to send actions to reducer
    - reducer: **Pure function** that defines how state updates, return the next state

useState vs useReducer

`useState`	`useReducer`
Simple state (e.g., numbers, strings)	Complex state (objects, arrays)
Update directly with `setState`	Use actions and a reducer function
Scattered logic in many event handlers	Centralized logic in one reducer function

Effect Hooks - useEffect

- Perform **side effects** in function components
  - Fetching data from an API
  - Setting timers (setTimeout, setInterval)
  - Subscribing to events
- Cleanup Function (optional): clean up side effects such as timers or event listeners
  - Prevents memory leaks
  - Stops background tasks
  - Required for subscriptions & timers

```
useEffect(() => {
  // side effect logic

  return () => {
    // cleanup (optional)
  };
}, [dependencies]);
```

Effect Hooks - useEffect

- **Lifecycle Comparison**

```
useEffect(() => {
  // componentDidMount + componentDidUpdate

  return () => {
    // componentWillUnmount
  };
}, [deps]);
```

Dependency Array	Behavior
NO dependency array	Runs after every render
EMPTY dependency array []	**componentDidMount** Runs once when the component mounts
dependency array with variable **[count]**	**componentDidUpdate** Runs on initial render Runs again only when count changes
with return callback implemented	**componentWillUnmount**

Performance Hooks - useMemo and useCallback

- Performance Optimization
- useMemo**(cb, dependencyArray)**: Memorize a calculated value
  - Avoid recomputing expensive values on every render unless the dependencies change
- useCallback**(cb, dependencyArray)**: Memorize a function
  - Prevent unnecessary recreation of functions between renders, especially useful when passing callbacks to child components to avoid re-renders

Hook	Memoizes	Returns	Use for
`useMemo`	✅ Value	`any` value	Expensive calculations
`useCallback`	✅ Function	`function`	Stable functions passed to children

Ref Hooks - useRef

- useRef is a way to remember something without causing a re-render
  - State: screen updates
  - Ref: memory only
- Syntax: const myRef = useRef(initialValue);
  - myRef.current: the stored value
  - Changing .current does **NOT** re-render the component
- Common use
  - Accessing DOM elements
    - Focusing an input automatically
    - Controlling <audio> or <video>

Context Hooks - useContext

- **useContext(MyContext):** read a value from a React Context inside a functional component
- To share data between components without passing props manually at every level (“prop drilling”)
- **Common use cases**
  - Theme (light/dark)
  - Authentication state
  - Language/locale settings
  - Global user settings or app config

Router Hooks

- router hooks connect your component to the URL
  - **useNavigate:** Navigate programmatically
  - **useParams:** Read URL parameters
  - **useLocation:** Read current URL info
  - useSearchParams: Read & update query params

Custom Hook

- Extract and reuse logic that uses hooks across multiple components, without duplicating code
  - Starts with the word **use**
  - Can call other React hooks
  - Extracts and **reuses** logic across components
- Common use cases:
  - useFetch(url): Fetch API data with loading/error states
  - useDebounce(value, delay): Debounce input for search/filtering

Others

- Conditional rendering
  - if vs && vs ? :
- Rendering lists
  - filter()
  - map() -> key
- Responding to events
  - Event handlers must be passed, not called
  - e.stopPropagation() stops the event handlers attached to the tags above from firing
  - e.preventDefault() prevents the default browser behavior for the few events that have it

Controlled vs Uncontrolled Components

- **Controlled component:** React controls the data
  - Input value comes from React state (**onChange**)
  - Every keystroke updates state
  - UI is always in sync with state
- **Uncontrolled component:** The DOM controls the data
  - Input value lives in the DOM (**ref**)
  - React reads it only when needed
  - No re-render on typing

Controlled vs Uncontrolled Components

Aspect	Controlled	Uncontrolled
Data source	React state	DOM
`useState`	Required	Not required
`useRef`	Optional	Required
Re-render on input	Yes	No
Validation	Easy	Hard
Recommended	✅ Yes	⚠️ Limited use

HOC

- **Higher-order component (HOC)**: A Higher-Order Component is a function that takes a component and returns a new component with additional behavior, commonly used for **logic reuse** in React
  - Reuse logic between multiple components
  - Authentication, Theming, Performance optimization

Aspect	HOC	Custom Hook
Introduced	Older React	Modern React
Used in	Class & Function	Function only
Adds UI	Yes	No
Recommended today	⚠️ Limited	✅ Yes

React.Memo

- React.memo is a **higher-order component** that prevents a functional component from re-rendering if its props have not changed
  - Performs a shallow comparison of props
  - If props are equal: skip render
  - If props changed: re-render
- Performance optimization

Tool	Memoizes
React.memo	Component
useMemo	Value
useCallback	Function

Pure function

- A pure function is a function that always returns the same output for the same input and has no side effects
  - Same input → same output
  - No side effects
- **Function components should be pure**
  - Only calculate JSX from props and state
  - NOT cause side effects
- React calls components twice (development)
  - If component is pure: no problem
  - If impure: bugs appear

Why does useEffect run twice?

- React renders components twice in **development** under **Strict Mode** to help detect side effects and ensure safe cleanup, and this behavior does not occur in production
    - React runs the effect, cleans it up, and then runs it again

Additional materials

- useLayoutEffect

Homework - Requirement

- 全程recording: 八股闭眼 & 摘耳机, 无AI辅助
- 全程recording: Coding可适当AI辅助, 但需写完(边解释边写), 模拟真实面试情景
- 录音提交地址: https://drive.google.com/drive/folders/1Mrm341uOIS8c2Ty6NwYvvSybHa6hESem?usp=drive_link
- 提交格式: 小组+日期, 例如Group1_12/07/2025
- 提交截止时间: due第二天5PM (周四作业递延到下周一)

Homework - Class Code

- Write the code as taught in class, take a screenshot of your code, and post it in the Wechat group. **It's due the same night.**

Homework - Short Answer Questions

1. How to import and export components using react and ES6?
2. What are functional components, How are they different from class components?
3. What are hooks in React? What are Rules of Hooks?
4. Explain the following hooks (what it does, what's the syntax). What are their class component equivalents?
    a. useState
    b. useEffect
    c. useRef
    d. useCallback and useMemo
5. Explain PureComponent vs Pure Function
6. Is React rendering twice a bug?

Homework - Short Answer Questions

7. What are custom hooks for? What is the naming convention for them?
8. Explain the useContext hook.
9. What is a pure function? What are the advantages to writing pure functions?
10. What is the difference between a controlled component and uncontrolled component?
11. How can you render only once using useEffect?
12. What is HOC?
13. What is the difference between useState and useDispatch hook?
14. What is React.memo? How does it differ from useMemo? Explain in terms of syntax and how they work.
15. Explain the difference between useMemo vs useCallback (what it does, what's the syntax). What are some use-cases?

Homework - Coding Questions

1. Create a parent component and child component. The parent component has a property that contains a list of two languages: [“JavaScript”, “Java”]. By default, it displays the first language, “JavaScript”. This property is sent to the child component.
The child component renders a button that can be clicked to toggle the view (updates the parent component). It should toggle from “JavaScript” to “Java”, or vice versa.

    A. Do this using props.

Homework - Coding Questions

2. Implement a Search Filter for Users

Create a React functional component that displays a list of users. Add a search input box that filters the displayed authors dynamically as the user types.

API: https://jsonplaceholder.typicode.com/users

- As the user types, filter the list of authors to show names that include the entered text
- If the search input is empty, show all users.

Homework - Coding Questions

3. Build a Simple Music Player UI Using useRef

Create a functional React component that uses useRef to control a <audio> element. The component should allow users to play and pause a music track using a button.

Requirements

- Use React functional components and the useRef hook.
- Create an <audio> element that loads a sample music file (you can use a public URL or a local file).
- Add a Play/Pause button that toggles the audio playback.
- Change the button label between "Play" and "Pause" based on the current state.

When the user clicks the button:

- If the audio is paused, it should start playing.
- If the audio is playing, it should pause.