

React 重點整理

簡報人：Neo.Wu

React 是什麼？

- 建構使用者介面的 JavaScript 函式庫
- 核心理念：**一切皆元件**
- 採用 Virtual DOM 技術，提升效能
- 適合構建 SPA、互動性高的應用程式

元件化的力量

- 元件 = UI 的最小單位，可組合、可重用
- 使用 JSX 編寫，看起來像 HTML，實際是 JS 語法
- 每個元件是函式，回傳 React 元素
- 元件名稱需以大寫開頭（PascalCase）

傳統做法的限制 (jQuery / JS)

- 沒有資料驅動 UI 的能力 → 要手動操作 DOM
- 缺乏結構化與組件化 → 維護困難
- 事件與資料分離 → 邏輯與畫面難以同步
- React 解決這些問題，**用資料決定 UI**

⚙ Virtual DOM 與效能

- Virtual DOM 是一棵 JS 物件樹，描述 UI
- 每次 state 或 props 改變，React 都會建立新 Virtual DOM
- 利用 diff 演算法比對新舊差異，只更新有變化的部分
- 減少 DOM 操作，提升效能

! 未加 key 的風險（示意）

```
{users.map(user => (  
  <li>{user}</li> // 沒有加 key!  
))}
```

- React 會用「順序」來判斷元素 → 容易出錯
- 如果重新排序或刪除元素，input 的值可能會跳位置
- 必須加上穩定且唯一的 `key`

JSX 與渲染原理

- JSX 寫法簡潔，底層會被轉譯為 `React.createElement(...)`
- 元件會渲染到 HTML 中的 `#root` 容器中
- React 18 使用 `createRoot`：

```
import { createRoot } from 'react-dom/client';  
const root = createRoot(document.getElementById('root'));  
root.render(<App />);
```

Props：父 → 子

- 元件之間透過 props 傳遞資料
- props 是唯讀的（單向資料流）
- 可重用性高

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```


State：元件自己的狀態

- 使用 `useState` 宣告變動資料
- 當 state 改變時，React 重新 render
- 不可直接修改，要透過 `setState` 函式

```
const [count, setCount] = useState(0);
```

Hooks 與生命週期對應

- Mount : `useEffect(() => {}, [])`
- Update : `useEffect(() => {}, [dep])`
- Unmount : `useEffect(() => { return () => {} }, [])`



React Hooks 快速瀏覽

- `useState` : 狀態管理
- `useEffect` : 處理副作用、生命週期
- `useRef` : 存取 DOM
- `forwardRef` : 傳遞 ref
- `useContext` : 資料共享
- `useMemo` / `useCallback` : 效能最佳化 `useMemo`
`useCallback`

學習建議

1. 熟 HTML / CSS / JS 基礎
2. 從簡單互動元件開始
3. 練習拆分元件、資料流思考
4. 熟悉 DevTools 與除錯工具
5. 建構專案從靜態頁 → 互動應用

推薦資源

- <https://react.dev/>
- <https://reactjs.org/blog/2022/03/29/react-v18.html>
- <https://react.dev/learn/start-a-new-react-project>
-

 感謝聆聽！

歡迎提問交流 