

JSX: Hiểu cách sử dụng JSX để viết giao diện.



JSX (JavaScript XML) là một cú pháp mở rộng của JavaScript, cho phép bạn viết giao diện giống như XML/HTML trong React. JSX giúp viết các thành phần giao diện (components) dễ đọc và dễ hiểu hơn. Dưới đây là cách hiểu và sử dụng JSX:

1. Cú pháp cơ bản của JSX

JSX là một sự kết hợp giữa JavaScript và HTML. Dưới đây là ví dụ cơ bản:

```
tsx Copy code
const Greeting = () => {
  return <h1>Hello, world!</h1>;
};
```

- Thẻ `<h1>` là một phần của JSX.
- JSX phải được đóng thẻ, ví dụ: `` hoặc `<div></div>`.

2. Sử dụng JavaScript trong JSX

Bạn có thể nhúng JavaScript vào JSX bằng cách sử dụng `{}`.

```
tsx Copy code
const name = "John";
const Greeting = () => {
  return <h1>Hello, {name}!</h1>;
};
```

- `name` là một biến JavaScript được sử dụng trong JSX.

3. Sử dụng điều kiện trong JSX

Có thể sử dụng các biểu thức điều kiện như `if`, `ternary operator` trong JSX.

```
tsx Copy code
const isLoggedIn = true;
const Greeting = () => {
  return <h1>{isLoggedIn ? "Welcome back!" : "Please log in."}</h1>;
};
```

4. Lặp danh sách với JSX

Bạn có thể sử dụng phương thức `.map()` để hiển thị danh sách các phần tử.

```
tsx Copy code
const items = ["Apple", "Banana", "Cherry"];
const ItemList = () => {
  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
};
```

- Thuộc tính `key` giúp React tối ưu hóa việc render danh sách.

5. Sử dụng CSS trong JSX

Có thể thêm style trực tiếp hoặc dùng className để áp dụng CSS.

```
tsx Copy code
const Greeting = () => {
  return <h1 style={{ color: "blue", fontSize: "24px" }}>Hello, world!</h1>;
};

// Hoặc dùng className
const Greeting = () => {
  return <h1 className="greeting">Hello, world!</h1>;
};
```

- `style` nhận một object với các thuộc tính CSS được viết theo camelCase.

6. JSX và các thành phần con (children)

Bạn có thể truyền các thành phần con qua JSX.

```
tsx Copy code  
const Card = ({ children }: { children: React.ReactNode }) => {  
  return <div className="card">{children}</div>;  
};  
  
const App = () => {  
  return (  
    <Card>  
      <h1>Title</h1>  
      <p>Description</p>  
    </Card>  
  );  
};  
  
↓
```

7. Quy tắc cần nhớ

- JSX phải được bao trong một phần tử gốc (parent element). Dùng `<React.Fragment>` hoặc `<>` nếu không muốn tạo thêm thẻ.

```
tsx Copy code  
const App = () => {  
  return (  
    <>  
      <h1>Title</h1>  
      <p>Description</p>  
    </>  
  );  
};  
  
↓
```

- JSX không hỗ trợ từ khóa như `class`, thay vào đó dùng `className`.

JSX là một phần quan trọng trong React, giúp bạn dễ dàng tạo và quản lý giao diện phức tạp. Hãy thử áp dụng trong dự án của bạn!

Component:
Functional Components (phổ biến hơn).
Class Components (ít dùng, nhưng cần biết để đọc mã cũ).



Trong React, **components** là các khái niệm chính để tạo giao diện. Có hai loại chính:

- Functional Components (phổ biến hơn, hiện đại).
- Class Components (ít phổ biến, nhưng quan trọng để hiểu mã cũ).

1. Functional Components (Hàm)

Định nghĩa

- Functional Components là các hàm JavaScript trả về JSX.
- Với React Hooks, bạn có thể quản lý state và lifecycle trong Functional Components, giúp chúng thay thế hầu hết các Class Components.

Cú pháp cơ bản

```
tsx Copy code  
import React, { useState } from "react";  
  
const Counter = () => {  
  const [count, setCount] = useState(0); // Sử dụng React Hook (useState)  
  
  return (  
    <div>  
      <p>Current Count: {count}</p>  
      <button onClick={() => setCount(count + 1)}>Increment</button>  
    </div>  
  );  
};  
  
export default Counter;  
  
↓
```

Ưu điểm

- Ngắn gọn, dễ viết, dễ hiểu.
- Sử dụng React Hooks để quản lý state và lifecycle (VD: `useState`, `useEffect`).
- Phù hợp với React 16.8+.

2. Class Components (Lớp)

Định nghĩa

- Class Components sử dụng cú pháp `class` của ES6.
- Trước khi React Hooks ra đời, đây là cách duy nhất để quản lý state và lifecycle trong React.

Cú pháp cơ bản

```
tsx Copy code
import React, { Component } from "react";

class Counter extends Component {
  state = { count: 0 }; // Quản lý state trong Class Component

  increment = () => {
    this.setState({ count: this.state.count + 1 }); // Cập nhật state
  };

  render() {
    return (
      <div>
        <p>Current Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}

export default Counter;
```

Lifecycle Methods

Class Components sử dụng các phương thức xử lý lifecycle:

- `componentDidMount`: Gọi sau khi component được gắn vào DOM.
- `componentDidUpdate`: Gọi khi component cập nhật.
- `componentWillUnmount`: Gọi trước khi component bị gỡ khỏi DOM.

Ví dụ:

```
tsx Copy code
class Example extends Component {
  componentDidMount() {
    console.log("Component mounted");
  }

  componentDidUpdate() {
    console.log("Component updated");
  }

  componentWillUnmount() {
    console.log("Component will unmount");
  }

  render() {
    return <div>Lifecycle Methods</div>;
  }
}
```

Nhược điểm

- Dài dòng, khó bảo trì hơn Functional Components.
- Không còn cần thiết khi đã có React Hooks.

So sánh

Tính năng	Functional Components	Class Components
State	Dùng <code>useState</code>	Quản lý bằng <code>this.state</code>
Lifecycle	Dùng Hooks như <code>useEffect</code>	Dùng lifecycle methods (<code>componentDidMount</code>)
Cú pháp	Ngắn gọn, dễ đọc	Dài dòng, phải dùng <code>class</code> và <code>this</code>
Hiệu suất	Tốt hơn (không cần xử lý <code>this</code>)	Hiệu suất kém hơn một chút
Khả năng mở rộng	Phù hợp với mọi tình huống	Dùng trong các dự án cũ hoặc đặc biệt

Khi nào dùng?

- **Functional Components:** Luôn ưu tiên dùng trong các dự án mới vì chúng đơn giản và hiện đại.
- **Class Components:** Chỉ cần biết để đọc hoặc bảo trì các dự án React cũ.

Bạn đang làm việc với dự án mới, nên sử dụng **Functional Components** kết hợp với React Hooks!



Props: Truyền dữ liệu từ component cha sang component con.
State: Quản lý trạng thái trong một component.
Event Handling: Xử lý sự kiện như `onClick`, `onChange`.



1. Props (Properties)

Props là cách để truyền dữ liệu từ component cha sang component con. Props là immutable, tức là không thể thay đổi trong component con.

Cách sử dụng props

Ví dụ: Truyền tên người dùng từ cha sang con.

```
tsx Copy code

```

```
// Parent Component
const Parent = () => {
  const userName = "Alice";

  return <Child name={userName} />;
};

// Child Component
const Child = ({ name }: { name: string }) => {
  return <p>Hello, {name}!</p>;
};

export default Parent;
```

- **Props trong Parent:** Truyền giá trị qua thuộc tính `name`.
- **Props trong Child:** Truy cập thông qua `{}` hoặc `props.name`.

Props với kiểu dữ liệu phức tạp

```
tsx
Copy code

type User = {
  id: number;
  name: string;
};

const Child = ({ user }: { user: User }) => {
  return <p>User ID: {user.id}, Name: {user.name}</p>;
};

const Parent = () => {
  const user = { id: 1, name: "Alice" };

  return <Child user={user} />;
};
```

2. State

State là dữ liệu cục bộ, dùng để lưu trữ trạng thái của một component. Khác với props, state có thể thay đổi (mutable) và thay đổi state sẽ làm component render lại.

Quản lý state với React Hooks

Sử dụng `useState` để quản lý state trong Functional Components.

```
tsx
Copy code

import React, { useState } from "react";

const Counter = () => {
  const [count, setCount] = useState(0); // Khởi tạo state

  return (
    <div>
      <p>Current Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};

export default Counter;
```

- `useState(0)` khởi tạo giá trị state ban đầu là `0`.
- Hàm `setCount` để cập nhật giá trị `count`.

State phức tạp với object

```
tsx
Copy code

const Form = () => {
  const [user, setUser] = useState({ name: "", age: 0 });

  const updateName = (e: React.ChangeEvent<HTMLInputElement>) => {
    setUser({ ...user, name: e.target.value });
  };

  return (
    <div>
      <input type="text" value={user.name} onChange={updateName} />
      <p>Name: {user.name}</p>
    </div>
  );
};


```

- Sử dụng `{ ...user }` để giữ các giá trị cũ trong object khi cập nhật state.

3. Event Handling (Xử lý sự kiện)

React cung cấp các thuộc tính xử lý sự kiện, tương tự như JavaScript DOM. Một số sự kiện phổ biến là `onClick`, `onChange`, `onSubmit`.

Xử lý sự kiện với `onClick`

```
tsx
Copy code

const Button = () => {
  const handleClick = () => {
    alert("Button clicked!");
  };

  return <button onClick={handleClick}>Click me</button>;
};
```

```
    return <button onClick={handleClick}>Click Me</button>;
};
```

Xử lý sự kiện với onChange

Ví dụ: Lấy dữ liệu từ trường nhập liệu (input).

```
tsx Copy code
```

```
const InputField = () => {
  const [value, setValue] = useState("");
  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setValue(e.target.value);
  };

  return (
    <div>
      <input type="text" value={value} onChange={handleChange} />
      <p>You typed: {value}</p>
    </div>
  );
};
```

Event với tham số

Truyền tham số vào hàm xử lý sự kiện.

```
tsx Copy code
```

```
const ItemList = () => {
  const handleClick = (item: string) => {
    alert(`You clicked on ${item}`);
  };

  return (
    <ul>
      {"Apple", "Banana", "Cherry"].map((item) => (
        <li key={item} onClick={() => handleClick(item)}>
          {item}
        </li>
      ))
    </ul>
  );
};
```

Ngăn hành vi mặc định

Dùng `preventDefault()` để ngăn hành vi mặc định, ví dụ khi xử lý form.

```
tsx Copy code
```

```
const Form = () => {
  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault(); // Ngăn tránh load
    alert("Form submitted!");
  };

  return (
    <form onSubmit={handleSubmit}>
      <button type="submit">Submit</button>
    </form>
  );
};
```

Tóm tắt

- Props:** Truyền dữ liệu từ cha sang con, không thay đổi được.
- State:** Quản lý dữ liệu cục bộ trong một component, có thể thay đổi.
- Event Handling:** Xử lý sự kiện như click, nhập liệu, submit trong React.

c. React Hooks (Cực kỳ quan trọng)

useState: Quản lý trạng thái.

useEffect: Xử lý các side effects (API calls, subscriptions).

useContext: Truyền dữ liệu qua nhiều cấp component mà không cần props.

Tìm hiểu thêm các hooks như: useReducer, useRef, useMemo, useCallback.

React Hooks (Cực kỳ quan trọng)

React Hooks là cách hàm cho phép bạn sử dụng state và lifecycle methods trong Functional Components. Đây là nền tảng của React hiện đại.

1. useState: Quản lý trạng thái

`useState` là Hook dùng để khai báo và quản lý trạng thái trong Functional Components.

Cách sử dụng

```
tsx Copy code
```

```
import React, { useState } from "react";

const Counter = () => {
```

```

const [count, setCount] = useState(0); // Khởi tạo state
const increment = () => setCount(count + 1);

return (
  <div>
    <p>Current Count: {count}</p>
    <button onClick={increment}>Increment</button>
  </div>
);
};

export default Counter;

```

- `useState(0)`: Khởi tạo state với giá trị ban đầu là `0`.
- Hàm `setCount` để cập nhật state.

2. useEffect: Xử lý side effects

`useEffect` được sử dụng để thực hiện các tác vụ bên ngoài như:

- Gọi API.
- Lắng nghe sự kiện.
- Động bộ hóa với DOM.

Cách sử dụng cơ bản

```

tsx
import React, { useState, useEffect } from "react";

const Timer = () => {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds((prev) => prev + 1);
    }, 1000);

    return () => clearInterval(interval); // Đón dép
  }, []); // Mảng rỗng: Chỉ chạy một lần khi component được mount.

  return <p>Seconds: {seconds}</p>;
};

export default Timer;

```

useEffect với dependency

- Không có dependency (`[]`): Chạy một lần khi component được mount.
- Có dependency: Chạy lại mỗi khi dependency thay đổi.
- Không truyền dependency: Chạy sau mỗi lần render (cần thận với hiệu suất).

```

tsx
useEffect(() => {
  console.log("State changed!", count);
}, [count]); // Chỉ chạy khi 'count' thay đổi.

```

3. useContext: Truyền dữ liệu qua nhiều cấp

`useContext` giúp truyền dữ liệu qua nhiều cấp component mà không cần dùng props drilling.

Cách sử dụng useContext

```

tsx
import React, { createContext, useContext } from "react";

const ThemeContext = createContext("light");

const Child = () => {
  const theme = useContext(ThemeContext); // Lấy giá trị từ context
  return <p>Current theme: {theme}</p>;
};

const Parent = () => {
  return (
    <ThemeContext.Provider value="dark">
      <Child />
    </ThemeContext.Provider>
  );
};

export default Parent;

```

- `createContext`: Tạo một context với giá trị mặc định.
- `useContext`: Sử dụng context trong component.

4. Các Hooks khác

a. useReducer

- Giống Redux nhưng đơn giản hơn, giúp quản lý state phức tạp.
- Thay thế `useState` khi có nhiều logic cập nhật state.

```
tsx Copy code

import React, { useReducer } from "react";

const reducer = (state: number, action: { type: string }) => {
  switch (action.type) {
    case "increment":
      return state + 1;
    case "decrement":
      return state - 1;
    default:
      return state;
  }
};

const Counter = () => {
  const [count, dispatch] = useReducer(reducer, 0); // Khởi tạo state với reducer

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => dispatch({ type: "increment" })}>+</button>
      <button onClick={() => dispatch({ type: "decrement" })}>-</button>
    </div>
  );
};

export default Counter;
```

b. useRef

- Dùng để truy cập trực tiếp vào DOM hoặc lưu trữ giá trị không làm component render lại.

```
tsx Copy code

import React, { useRef } from "react";

const InputFocus = () => {
  const inputRef = useRef<HTMLInputElement>(null);

  const focusInput = () => {
    inputRef.current?.focus(); // Truy cập DOM trực tiếp
  };

  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={focusInput}>Focus Input</button>
    </div>
  );
};

export default InputFocus;
```

c. useMemo

- Tối ưu hiệu suất, ghi nhớ giá trị tính toán để tránh tính lại không cần thiết.

```
tsx Copy code

import React, { useState, useMemo } from "react";

const ExpensiveCalculation = (num: number) => {
  console.log("Calculating...");
  return num * 2;
};

const App = () => {
  const [count, setCount] = useState(0);
  const [input, setInput] = useState("");

  const doubled = useMemo(() => ExpensiveCalculation(count), [count]);

  return (
    <div>
      <p>Result: {doubled}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <input value={input} onChange={(e) => setInput(e.target.value)} />
    </div>
  );
};

export default App;
```

d. useCallback

- Ghi nhớ hàm để tránh tạo lại không cần thiết khi component render.

```

tsx
import React, { useState, useCallback } from "react";

const Button = React.memo(({ handleClick }: { handleClick: () => void }) => {
  console.log("Button rendered");
  return <button onClick={handleClick}>Click Me</button>;
});

const App = () => {
  const [count, setCount] = useState(0);

  const handleClick = useCallback(() => {
    console.log("Button clicked");
  }, []);
  // Ghi nhớ hàm, không tạo mới khi render lại.

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button handleClick={handleClick} />
    </div>
  );
};

export default App;

```

Tóm tắt các React Hooks quan trọng

Hook	Công dụng
useState	Quản lý state trong component.
useEffect	Xử lý side effects (gọi API, subscriptions, v.v.).
useContext	Truyền dữ liệu qua nhiều cấp mà không cần props.
useReducer	Quản lý state phức tạp với logic cập nhật rõ ràng.
useRef	Lưu giá trị hoặc truy cập DOM trực tiếp mà không làm render lại.
useMemo	Ghi nhớ giá trị tính toán, tránh tính toán lại không cần thiết.
useCallback	Ghi nhớ hàm, tránh tạo lại không cần thiết khi render lại.

Hãy sử dụng các Hook này để làm ứng dụng của bạn hiệu quả hơn! 🚀

Ctrl ⌘ ⌂ ⌃ ⌄ ⌅ ⌏ ⌒ ⌓ ⌕

Message ChatGPT



?

ChatGPT can make mistakes. Check important info.