Lesson 4: JSX & Props

Created	March 22, 2022 9:55 AM
Tags	Empty
Related to Untitle...	3

💡 Hiểu về cách thức truyền dữ liệu và render dữ liệu lên trang web

1. JSX là gì?

Cú pháp trông giống như HTML mà chúng ta sử dụng trong các component thực sự không phải HTML thông thường. Đó là một cú pháp mới có tên là JSX - Javascript XML.

JSX là cú pháp do đội ngũ React phát triển, sử dụng chủ yếu trong React. Tuy nhiên vẫn có thể sử dụng JSX cho một vài công cụ khác như VueJS.

Mục đích của JSX được tạo ra để có thể tạo ra các element một cách tường minh và đơn giản. JSX cho phép developer có thể tạo ra các đoạn HTML nhanh chóng, kèm với khả năng có thể chèn các giá trị JS vào bên trong để tạo ra trang web có nội dung động.

2. Bạn có thể không cần dùng JSX với React

ReactJS có thể hoạt động mà không cần tới cú pháp JSX. React có sẵn một function giúp chúng ta có thể tạo ra phần view. Xét 2 ví dụ sau:

```
function App() { return ( <div className="app"> <h1>Hello, World!</h1> </div>
) }
```

Sử dụng JSX

```
import React from 'react' function App() { return React.createElement( "div",
{className: "app"}, React.createElement("h1", {}, "Hello, World!") ) }
```

Không sử dụng JSX

Hai ví dụ này sẽ cho kết quả giống nhau. Tuy nhiên, dễ thấy cách viết thứ hai khó hơn khá nhiều. Và với những component có nhiều giao diện, cách viết thứ 2 sẽ trở nên rất khó bảo trì sau này. Vì vậy, mặc dù JSX là không hoàn toàn bắt buộc với React, tuy nhiên gần như toàn bộ tất cả các dự án React đều sử dụng JSX.

> 💡 Thực tế thì JSX sau cùng sẽ được biến đổi thành `React.createElement` nhờ một công cụ tên là Babel. Đó là lý do tại sao ở ví dụ với CodePen ở bài trước, chúng ta cần chọn Babel làm JS Pre-processor.

---

3. JSX không phải là HTML

JSX mặc dù có cú pháp trông rất giống HTML, tuy nhiên nó có một vài đặc điểm khác biệt so với HTML thông thường

1. Single parent root: Các component React cần phải return một thẻ bao ngoài duy nhất hoặc một array, không thể trả về nhiều hơn hai thẻ.
2. `className` thay vì `class`. Đây là một lý do kĩ thuật. `class` là một từ khoá trong JS. Vì vậy, đội ngũ React đã sử dụng `className` thay vì `class` để tránh các lỗi.
3. `style` nhận giá trị là một object, thay vì là cú pháp CSS thông thường
4. Các thuộc tính HTML sẽ được đổi tên theo kiểu `camelCase`
5. Đối với các đoạn JSX nằm trên nhiều dòng, JSX cần phải được bọc bên trong cặp ngoặc tròn `()`
6. Component do chúng ta viết buộc phải được sử dụng ở dưới dạng tên viết hoa.

Chúng ta sẽ đi sâu vào các thành phần trên ở các phần dưới.

---

4. Render giá trị JS với JSX

JSX cho phép chúng ta có thể output các giá trị Javascript trực tiếp vào bên trong. Cú pháp để làm điều này là sử dụng dấu ngoặc nhọn `{}`. Xét ví dụ sau:

```
const App = () => { const randomAge = Math.floor(Math.random() * 10) const imgSr
"https://upload.wikimedia.org/wikipedia/commons/thumb/b/b6/Image_created_with_a_
Image_created_with_a_mobile_phone.png" return ( <div> Hello, my name is MindX. I
old. <img src={imgSrc} /> </div> ) }
```

Bằng cách này, chúng ta có thể dễ dàng tạo ra các trang web có nội dung động, có thể hiển thị ra màn hình các giá trị JS cần thiết.

Một điểm lưu ý là các giá trị JS có thể được chèn vào các thuộc tính của các thẻ HTML. Như ở ví dụ trên, ta thấy thẻ `img` với thuộc tính `src` đã nhận vào giá trị là `imgSrc` có kiểu dữ liệu là `string`.

> 📌 Không phải tất cả các kiểu dữ liệu đều có thể được **render** trong JSX. Kiểu dữ liệu object dạng key-value không thể được render vì nó không phải là một "React child" hợp lệ. Có thể biến object thành một string và sau đó render bằng function `toString`

> 👨‍💻 Các bạn có thể thử với array, function và các kiểu dữ liệu nguyên thuỷ khác và theo dõi output.

Có thể thực thi một số câu lệnh bên trong JSX, miễn sao chúng có kết quả là một **giá trị JS.** Xem ví dụ dưới đây:

```
const weekday = [ "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday" ]; const App = () => { return <div>Today is {weekday[new
Date().getDay()]}</div> }
```

Ở trong ví dụ trên, `weekday[new Date().getDay()]` sẽ trả ra kết quả là một `string` nên có thể được hiển thị trên màn hình. Tuy nhiên, chúng ta không thể sử dụng những câu lệnh `if`, `for`, `while` bên trong JSX, vì chúng không phải là các giá trị trong JS.

---

5. JSX với styling:

Có nhiều cách để có thể style với JSX trong React, tương tự với HTML. Cơ bản sẽ có hai kiểu như sau:

1. Style với file CSS bên ngoài, sử dụng HTML `class` và `id`
2. Inline style

Với cách thứ nhất, cách viết CSS không có gì khác biệt. Điểm khác biệt ở đây là chúng ta cần dùng `className` thay cho `class` thông thường. Điểm khác biệt ở đây là chúng ta có thể import file css vào bên trong component với cú pháp `import "<file>.css"`. Xét ví dụ sau:

```
.App { text-align: center; font-weight: bold; }
```

App.css

```
import "./App.css" const App = () => { return ( <div className="App"> Hello,
world! </div> ) }
```

App.js

Đối với inline-style, đây là một điểm khác biệt tương đối lớn so với HTML thông thường:

- `style` trong JSX nhận giá trị là một **object (key-value)**
- Các key CSS phải được viết dưới dạng `camelCase`
- Các value CSS cần phải được viết dưới dạng `string` hoặc `number`

> 💡 `camelCase` là một quy ước đặt tên phổ biến. Đặc điểm của `camelCase` là từ đầu tiên sẽ bắt đầu với chữ thường, các từ tiếp theo sẽ được viết hoa chữ cái đầu tiên. Một vài ví dụ với `camelCase` là `totalValue`, `aboutUs`, `tenBien`... Với JS, quy ước đặt tên tiêu chuẩn là `camelCase`

Xét ví dụ inline-style sau:

```
const App = () => { return ( <div style={{backgroundColor: 'yellow', fontSize:
18}}>Hello, World!</div> ) }
```

Một điều lưu ý ở đây là với các giá trị CSS nhận vào là số, đơn vị được sử dụng sẽ là `px`. Nếu muốn đơn vị khác, chúng ta cần chuyển giá trị thành string và thêm đơn vị đo vào sau giá trị đó.

> 📌 Cần lưu ý phân biệt hai dấu ngoặc nhọn ở trong ví dụ trên: dấu ngoặc nhọn thứ nhất là ký hiệu đánh dấu việc chúng ta chèn biến Javascript vào trong, dấu ngoặc nhọn thứ hai là ký hiệu bắt đầu của một object javascript.

6. Props là gì?

Props là tham số đầu vào của các component trong React. Props là một trong những khái niệm cực kỳ quan trọng của React.

Props có giá trị là một object. Nó chính là tham số thứ hai trong function `React.createElement` (tham số thứ hai trong `React.createElement` chính là một object).

Liên tưởng đơn giản, props trong React chính là các thuộc tính trong HTML. Điểm khác biệt ở đây là chúng ta có thể tự định nghĩa những thuộc tính đó, thay vì với HTML, các thuộc tính được định nghĩa sẵn.

```
const App = () => { const x = 1; const y = 2; return ( <div> <Sum a={x} b={y}
/> </div> ) } const Sum = (props) => { console.log(props) // {a: 1, b: 2}
return <div>The value is: {props.a + props.b}</div> }
```

Props có thể nhận giá trị thuộc tất cả các kiểu dữ liệu trong JS. Props chính là phương tiện để lưu chuyển dữ liệu bên trong React.

Props hoàn toàn do chúng ta tự định nghĩa. Các components do chúng ta định nghĩa không hiểu được các giá trị thuộc tính HTML như `src`, `id`, `className`. Chúng đơn thuần là các key trong một object props. Chúng ta sẽ cần gán lại cho các thẻ HTML tương ứng bên trong component.

```
const App = () => { return ( <div> <NameCard className="name-card" id="alice"
/> </div> ) } const NameCard = (props) => { return ( <div className=
{props.className}> <div id={props.id}>Name: Alice</div> </div> ) }
```

Props là read-only, nghĩa là chúng ta **không thể thay đổi** được giá trị props bên trong một component.

📌 **Tầm quan trọng của props trong React**: props cho phép tạo ra các Component có khả năng tái sử dụng cao. Thay vì dữ liệu được hard-coded bên trong component, props cho phép component có thể nhận được dữ liệu đầu vào mỗi lần sử dụng nó. Đây chính là công cụ để kết nối các component lại với nhau. Vì vậy, props cho phép ứng dụng được chia nhỏ thành nhiều phần.

🤔 Điều gì xảy ra nếu như Props nhận được một giá trị là một `function` thay vì các kiểu dữ liệu khác?

---

7. Children Props

Các thẻ HTML có thể chứa bên trong nó các thẻ HTML khác, ví dụ như `div`, `p`, ... Tương tự như vậy, các thẻ "HTML" do chúng ta tự tạo cũng có thể làm được điều tương tự thông qua một giá trị props đặc biệt có tên là `children`. Xét ví dụ sau:

```
import "./Card.css"; const Card = (props) => { return <div className="card">
{props.children}</div> }
```

Card.js

```
.card { padding: 10px; border: 1px solid black; border-radius: 8px; box-
shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19); }
```

Card.css

```
import Card from "./Card.js"; const App = () => { return ( <Card> <div>Inside
a card</div> </Card> ) }
```

App.js

Cũng tương tự như các props thông thường khác, `children` có thể nhận giá trị là bất cứ kiểu dữ liệu nào. Với ví dụ ở trên, children nhận vào giá trị là một React Element.

`children` props giúp chúng ta có khả năng "compose" các component lại với nhau. Thay vì cố định giá trị bên trong `Card`, lúc này `Card` có thể cho bất cứ component nào nằm trong nó có thêm các thuộc tính CSS ở trên.

🤔 Điều gì xảy ra nếu như `children` props lại là một function, và function đó trả lại kết quả là JSX?

---

8. Smart & dump components

Xét 2 ví dụ sau:

```
const Sum = () => { const x = 1
const y = 2 return <div>{x + y}
</div> } <Sum /> <Sum /> <Sum />
```

Smart Component

```
const Sum = (props) => { const
{x, y} = props return <div>{x +
y}</div> } <Sum x={1} y={2} />
<Sum x={2} y={3} /> <Sum x={7} y=
{5} />
```

Dump Component

Trong ví dụ trên thì bên trái, component `Sum` có xử lý logic bên trong, còn bên phải thì không.

Phần bên trái là một Smart Component, và phần bên phải là một Dump Component. Trong thực tế thì chúng ta viết càng nhiều dump component nghĩa là chúng ta càng smart 🙂

Ở ví dụ bên trái, smart component không thể tái sử dụng, vì mỗi lần chúng ta dùng nó, nó luôn cho kết quả là 3.

Ngược lại, với ví dụ bên phải, dump component có thể được sử dụng để in ra tổng 2 số bất kỳ khi chúng ta truyền giá trị vào bên trong.

Tuy nhiên, không phải lúc nào dump component cũng tốt. Vì nếu cho phép truyền quá nhiều props sẽ dẫn đến code khó đọc và khó bảo trì hơn do nhiều thành phần tham gia vào logic của component hơn. Việc quyết định "dump" / "smart" bao nhiêu là đủ còn tuỳ thuộc khá nhiều vào kinh nghiệm và từng tình huống xử lý khác nhau.