



Lesson 5: State & Events

🕒 Created March 22, 2022 9:55 AM

☰ Tags Empty

📌 Related to Untitle... 📄 4



Một thành phần không thể thiếu nữa trong React. Giúp tạo nên các ứng dụng có nhiều logic phức tạp.

1. Lắng nghe các sự kiện với React

Xử lý các sự kiện như “click” với button, “submit” với form là một trong những phần cơ bản của bất cứ ứng dụng web nào. React có cách tiếp cận khá đơn giản với các sự kiện. Xét ví dụ sau:

```
const App = () => { const onClick = () => { console.log("Click me"); } return <button onClick={onClick}>Click me</button> }
```

Có một vài đặc điểm cần lưu ý:

1. Với JSX, các thuộc tính liên quan tới event cần phải được đổi tên theo dạng `camelCase` như đã nói ở bài trước (`onClick` thay vì `onclick` , `onSubmit` thay vì `onsubmit` , ...)
2. Các thuộc tính liên quan tới event cần nhận vào một giá trị là một function Javascript. Vì vậy, chúng cần được sử dụng cú pháp ngoặc nhọn `{}` (`onClick={<function>}` thay vì `onclick="function()"`)

Cần lưu ý là chúng ta cần truyền giá trị vào là một function cho các sự kiện. Xét ví dụ bên dưới:

```
const App = () => { const onClickMe = () => { console.log("Click me") return null; } return <button onClick={onClickMe()}>Click me</button> }
```

Ta thấy function `onClickMe()` được truyền vào với dấu ngoặc tròn. Điều này nghĩa là `onClickMe` sẽ được chạy trước, lấy kết quả trả về và truyền vào cho giá trị `onClick` của button. Kết quả trả về của `onClickMe` là `null` , do đó, thực tế thuộc tính `onClick` nhận vào null thay vì một function.



Hãy thử xử lý sự kiện `onSubmit` của form và tìm cách để có thể gọi được function `preventDefault()` của event submit đó.

Ta có thể đặt một “anonymous function” vào bên trong các thuộc tính event như ví dụ dưới đây.

```
const App = () => { return <button onClick={() => { console.log("Hello")
}}>Click me</button> }
```

2. Component Functions được thực thi như thế nào?

```
const App = () => { console.log('App') return ( <div> <A /> </div> ) } const A
= () => { console.log('A') return ( <div> <B /> </div> ) } const B = () => {
console.log('B') return ( <div> Hello React </div> ) } ReactDOM.render(<App/>,
document.getElementById('app'))
```

Các function `App`, `A` và `B` không được gọi ở bất cứ đâu trong ứng dụng trên. Thay vào đó, chúng được sử dụng với cú pháp đặc biệt của JSX, giống như những thẻ HTML. React sẽ thực hiện chạy các function component đó, thực hiện lấy kết quả trả về và render thành ứng dụng web. React sẽ tiến hành tuần tự như vậy cho tới khi không còn function nào nữa.

Như ở ví dụ trên, khi được chạy, chúng ta sẽ thấy lần lượt các giá trị “App”, “A” và “B” được hiển thị ra màn hình.

Như vậy, mặc dù các function component không được gọi trực tiếp ở bất cứ đâu trong ứng dụng, thực chất chúng được thực thi mỗi lần chúng ta sử dụng cú pháp JSX với chúng.

3. Cập nhật giao diện với các sự kiện

Xét ví dụ sau:

```
const App = () => { let count = 0; const handleClick = () => { count = count +
1; console.log("count: ", count) } return ( <div> <span>{count}</span> <button
onClick={handleClick}>Increase</button> </div> ) }
```

Chúng ta có một button và một giá trị `count`. Chúng ta mong muốn khi click vào button đó thì sẽ tăng giá trị lên 1

Với ví dụ trên, khi ta click vào button, giá trị của biến `count` sẽ được thay đổi. Khi chúng ta mở màn hình console cũng sẽ nhận được giá trị của `count` in ra màn hình. Và chúng ta cũng sẽ mong đợi rằng component `App` sẽ thực hiện việc tính toán lại để thay đổi giá trị trong cặp thẻ ``. Từ đó, giao diện sẽ được cập nhật.

Tuy nhiên thì giao diện sẽ **không** được cập nhật!

Thực tế, các biến thông thường như `count` trong ví dụ trên sẽ không làm cho React thực hiện việc tính toán lại dữ liệu và cập nhật giao diện. React sẽ hoàn toàn bỏ qua sự thay đổi của các biến đó. Khi chúng ta muốn cho React biết rằng nó cần tính toán lại giao diện, chúng ta cần sử dụng một khái niệm đặc biệt từ React: “State”

4. Sử dụng state với React hooks

State về cơ bản là một giá trị biến đặc biệt trong React. Nó là giá trị mà khi thay đổi, React sẽ tiến hành việc tính toán lại kết quả của component, và từ đó cập nhật lại giao diện. Để sử dụng được state, chúng ta cần import một function từ trong thư viện React là `useState`. `useState` và một số function khác trong thư viện được gọi là các “hooks”. Chúng ta sẽ biết tới các hooks khác của React ở những bài sau.

Cú pháp cơ bản của `useState` như sau:

```
const <tên_biến_state> = useState(<giá_trị_ban_đầu_của_state>)
```

`useState` trong React là một function với đặc điểm sau:

- Tham số đầu vào là một giá trị trong JS, đây cũng là giá trị khởi đầu cho state đó.
- Kết quả trả về là một **array**. Trong đó có 2 phần tử: phần tử thứ nhất là giá trị của state đó, phần tử thứ hai là một function khác.

Xem ví dụ dưới đây:

```
import {useState} from 'react' const App = () => { const countState =
useState(10) console.log("count: ", countState[0]); return <div>
{countState[0]}</div> }:
```

Trong ví dụ, trên, chúng ta khởi tạo một biến là `countState` với giá trị ban đầu bằng 10. `countState` có giá trị là một mảng gồm 2 phần tử. Phần tử ở vị trí 0 của `countState` chính là giá trị của state. Ở trong ví dụ trên thì giá trị của nó ban đầu là 10. Do đó, chúng ta sẽ nhìn thấy số 10 được in ra trên màn hình.

Giá trị state đặc biệt hơn các biến thông thường khác: khi thay đổi, React sẽ tiến hành việc tính toán lại kết quả của component và cập nhật lại giao diện. Để có thể cập nhật được giá trị của state. Ta cần sử dụng một function đặc biệt, đó là tham số thứ hai của `countState` :

```
import {useState} from 'react' const App = () => { const countState =
useState(10) const count = countState[0] const setCount = countState[1] const
onIncreaseClick = () => { setCount(count + 1) } return ( <div> <span>{count}<
span> <button onClick={onIncreaseClick}>Increase</button> </div> ) }
```

Function `setCount` nhận vào một tham số là giá trị tiếp theo mà state đó sẽ nhận. Như ở ví dụ trên, ban đầu, giá trị của `count` là 10. Sau khi click vào button “Increase”, giá trị của mới của `count` sẽ bằng giá trị cũ của nó cộng thêm 1 đơn vị.



Các function sử dụng để cập nhật giá trị của state thường được gọi là `setState`



Trong thực tế, người ta thường sử dụng cú pháp destructuring để khai báo biến `state` và `setState`. Cú pháp như sau:

```
const [count, setCount] = useState(10)
```

Trong các phần sau, chúng ta sẽ chủ yếu sử dụng cú pháp này.



State chính là công cụ để React có thể được viết dưới dạng declarative thay vì imperative. Hãy thảo luận thêm về vấn đề này trong phần lab.

5. Hiểu rõ hơn về state trong React

State là giá trị riêng và độc lập giữa từng thực thể component.

Xem ví dụ sau:

```
const App = () => { return ( <div> <Counter /> <Counter /> <Counter /> </div>
) } const Counter = () => { const [count, setCount] = useState(0) const
onIncrease = () => { setCount(count + 1) } return ( <div> <div>{count}</div>
<button onClick={onIncrease}>Increase</button> </div> ) }
```

Trong ví dụ trên, ta thấy có tới 3 “thực thể” của component `Counter` được sử dụng trong component `App`. Và khi click vào mỗi một button “Increase” riêng biệt, ta thấy các giá trị đếm của từng component `Counter` không giống nhau. Điều này nghĩa là các giá trị state trong React là hoàn toàn riêng biệt giữa các thực thể của một component. Khi ta cập nhật một giá trị trong mỗi thực thể, các thực thể còn lại hoàn toàn không chịu ảnh hưởng từ việc thay đổi đó.

Tại sao với state luôn dùng `const` thay vì `let`

Xem ví dụ sau:

```
const Counter = () => { let [count, setCount] = useState(0) const onIncrease =
() => { count = count + 1 } return ( <div> <span>{count}</span> <button
onClick={onIncrease}>Increase</button> </div> ) }
```

Với ví dụ trên, khi ta click vào button Increase, giá trị của count sẽ không thay đổi.

Việc cập nhật state với React không chỉ đơn giản ở việc cập nhật lại giao diện. Có những tính toán ở bên trong React để nó có thể nhận biết được phần nào của ứng dụng web cần được cập nhật. Vì vậy, việc cập nhật state thông qua hàm `setState` của nó là cần thiết để React có thể thực hiện các tính toán cần thiết.

Do đó, việc gán lại dữ liệu cho state nhờ việc khai báo với `let` là hoàn toàn không cần thiết. Việc sử dụng phép gán để cập nhật dữ liệu sẽ không giúp cho ứng dụng React được cập nhật. Sử dụng `const` để khai báo sẽ giúp lập trình viên hiểu được rằng việc gán lại dữ liệu cho state là không thể, và buộc phải dùng `setState`.

Cập nhật dữ liệu dạng reference.

Xem ví dụ sau:

```
const App = () => { const [person, setPerson] = useState({name: "MindX", age:
10}) const increaseAge = () => { person.age = person.age + 1;
setPerson(person); } return ( <div> <p>Hello! I'm {person.name}. I'm
{person.age} years old.</p> <button onClick={increaseAge}>Increase
age</button> </div> ) }
```

Trong ví dụ trên, `person` là một state trong `App`, và kiểu dữ liệu của nó là một object.

Khi người dùng click vào button “Increase Age”, chúng ta thực hiện tăng số tuổi lên một. Tuy nhiên, giao diện của ứng dụng sẽ không được cập nhật.

Lý do ở đây là khi function `setPerson` được gọi, React sẽ tiến hành so sánh giá trị của state cũ và giá trị mới sắp được cập nhật. Trong trường hợp này, kiểu dữ liệu `object` là dạng reference. Khi ta gán `person.age = person.age + 1`, cũng có nghĩa là đã cập nhật lại giá trị cũ của person. Nên khi so sánh, React sẽ thấy hai giá trị cũ và mới cùng là giá trị, và do đó không tiến hành cập nhật.

Cách làm đúng ở đây là:

```
const increaseAge = () => { setPerson({...person, age: person.age + 1}); }
```

Sử dụng spread operator để copy toàn bộ giá trị của `person` qua một object mới, sau đó sửa lại giá trị `age` tương ứng. Lúc này, giá trị cũ và giá trị mới là hai object khác nhau. React sẽ nhận biết được và cập nhật lại giao diện.

Cập nhật state dựa vào giá trị cũ của chúng

Trong nhiều trường hợp, các giá trị state mới có thể dựa vào giá trị của state cũ. Lấy ví dụ sau:

```
const App = () => {
  const [person, setPerson] = useState({name: "MindX", age: 10})
  const increaseAge = () => {
    return (
      <div>
        <p>Hello! I'm {person.name}. I'm {person.age} years old.</p>
        <button onClick={increaseAge}>Increase age</button>
      </div>
    )
  }
}
```

Khi click vào button “Increase Age”, ta muốn tăng giá trị tuổi của `person` lên 1 đơn vị

Khi này. Giá trị mới của `person.age` sẽ bằng giá trị `person.age` cũ cộng thêm 1. Ngoài cách làm như ở ví dụ trên, ta còn có thể có cách làm như sau:


```
const increaseAge = () => {
  setPerson((prev) => {
    return {
      ...prev,
      age: prev.age + 1
    }
  })
}
```

Lúc này, `setPerson` nhận vào một tham số là một `function` thay vì một `object` như ở trên. Đây là một cú pháp khác của việc thay đổi giá trị với `setState`

```
setState( function(<state-hiện-tại>) {
  return <state-mới>
} )
```

Function được truyền vào bên trong `setState` có tham số đầu vào là giá trị hiện tại của state, và nó cần phải trả ra giá trị tiếp theo của state đó. Trong ví dụ, ta thấy `prev` chính là dữ liệu của state hiện tại, và chúng ta trả ra một object mới chính là dữ liệu tiếp theo của state. Chúng ta có thể thực hiện thêm các tính toán khác ở bên trong function đó. Lúc này có thể thực hiện các phép tính với `prev`

Cách cập nhật state này có nhiều lợi ích mà chúng ta sẽ nói ở các bài sau.

 Đọc thêm về React State bằng tiếng Việt ở đây:
<https://vi.reactjs.org/docs/hooks-state.html>