

**REACT-JS Cơ bản**

|  |
| --- |
| - Conditional Rendering |
| - Lists and Keys |
| - Forms |

# 

# **Conditional Rendering**

Trong React các bạn có thể tạo ra nhiều Component khác nhau và gói gọn chúng theo từng hành vi khác nhau cụ thể mà các bạn muốn. Sau đó các bạn có thể render chúng tuỳ thuộc vào điều kiện nào đó hoặc trạng thái nào đó (state) của chương trình đang chạy.

Bạn sử dụng các câu lệnh if và cú pháp điều kiện condition ? exprIfTrue : exprIfFalse giống như trong javascript.

ví dụ:

if ( a === "1") {

return <MyComponent />

}

hoặc

return (

<MyComponent>

{

a === 1 ? <MyComTrue /> : <MyComFalse />

}

</MyComponent>

)

Ví dụ trong React dưới đây chúng ta có 2 components và tuỳ thuộc vào state hay điều kiện giá trị nào đó được truyền trong props để render component nào:

function UserGreeting(props) {

return <h1>Welcome back!</h1>;

}

function GuestGreeting(props) {

return <h1>Please sign up.</h1>;

}

Chúng ta sẽ tạo một component Greeting để thực hiện điều đó như sau:

function Greeting(props) {

const isLoggedIn = props.isLoggedIn;

if (isLoggedIn) {

return <UserGreeting />;

}

return <GuestGreeting />;

}

ReactDOM.render(

// Try changing to isLoggedIn={true}:

<Greeting isLoggedIn={false} />,

document.getElementById('root')

);

ở trên chúng ta thấy rằng tuỳ thuộc vào giá trị được truyền vào trong props.isLoggedIn là true hay false thì ta sẽ render component nào của 1 trong 2 component trên.

### **Element Variables**

Chúng ta cũng có thể sử dụng biến để lưu element và truyền vào trong hàm render để in ra màn hình tuỳ thuộc vào điều kiện ở trong state của component.

Bên cạnh component Greeting ở trên chúng ta tạo thêm 2 component nút Logout và Login để set lại giá trị state, sau đó dựa vào state này để render ra component Greeting và nút Logout hay Login.

Dòng code sau sẽ cho thấy điều đó:

class LoginControl extends React.Component {

constructor(props) {

super(props);

this.handleLoginClick = this.handleLoginClick.bind(this);

this.handleLogoutClick = this.handleLogoutClick.bind(this);

this.state = {isLoggedIn: false};

}

handleLoginClick() {

this.setState({isLoggedIn: true});

}

handleLogoutClick() {

this.setState({isLoggedIn: false});

}

render() {

const isLoggedIn = this.state.isLoggedIn;

let button;

if (isLoggedIn) {

button = <LogoutButton onClick={this.handleLogoutClick} />;

} else {

button = <LoginButton onClick={this.handleLoginClick} />;

}

return (

<div>

<Greeting isLoggedIn={isLoggedIn} />

{button}

</div>

);

}

}

ReactDOM.render(

<LoginControl />,

document.getElementById('root')

);

### **Inline If with Logical && Operator**

Viết biểu thức điều kiện ngay trên dòng JSX bằng cách lồng chúng trong cặp dấu ngoặc {}, và toán tử logic && của javascript. Ví dụ:

function Mailbox(props) {

const unreadMessages = props.unreadMessages;

return (

<div>

<h1>Hello!</h1>

{unreadMessages.length > 0 &&

<h2>

You have {unreadMessages.length} unread messages.

</h2>

}

</div>

);

}

const messages = ['React', 'Re: React', 'Re:Re: React'];

ReactDOM.render(

<Mailbox unreadMessages={messages} />,

document.getElementById('root')

);

bởi vì true && expression thì phần tử bên phải sau && sẽ được hiển thị ra output, còn nếu biểu thức bên trái trước && sai thì sẽ được bỏ qua luôn.

### **Inline If-Else with Conditional Operator**

Một cách khác để viết điều kiện trong JSX là sử dụng câu lệnh điều khiển if else trong javascript [condition ? true : false](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Conditional_Operator) , ví dụ dưới đây:

render() {

const isLoggedIn = this.state.isLoggedIn;

return (

<div>

The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.

</div>

);

}

và cũng có thể viết được cả trong trường hợp nếu là output element component:

render() {

const isLoggedIn = this.state.isLoggedIn;

return (

<div>

{isLoggedIn ? (

<LogoutButton onClick={this.handleLogoutClick} />

) : (

<LoginButton onClick={this.handleLoginClick} />

)}

</div>

);

}

lưu ý rằng nếu logic ở trong phần này nó lớn và phức tạp thì nên tách ra thành các component có logic riêng.

### **Preventing Component from Rendering**

Khi chúng ta không muốn tiếp tục xuất ra màn hình một component nào đó trong đoạn render nếu chúng không thoả mãn điều kiện thì có thể bằng cách là return null trong ví dụ sau:

function WarningBanner(props) {

if (!props.warn) {

return null;

}

return (

<div className="warning">

Warning!

</div>

);

}

class Page extends React.Component {

constructor(props) {

super(props);

this.state = {showWarning: true};

this.handleToggleClick = this.handleToggleClick.bind(this);

}

handleToggleClick() {

this.setState(state => ({

showWarning: !state.showWarning

}));

}

render() {

return (

<div>

<WarningBanner warn={this.state.showWarning} />

<button onClick={this.handleToggleClick}>

{this.state.showWarning ? 'Hide' : 'Show'}

</button>

</div>

);

}

}

ReactDOM.render(

<Page />,

document.getElementById('root')

);

# **Lists and Keys**

Trong phần này ta biết được cách chuyển đổi list trong javascript bằng cách sử dụng hàm map() để lặp qua các phần tử trong một mảng như đoạn code sau:

const numbers = [1, 2, 3, 4, 5];

const doubled = numbers.map((number) => number \* 2);

console.log(doubled);

kết quả trả về của hàm map() sẽ cho ra một mảng mới, và với hàm log trên sẽ cho ra output [2, 4, 6, 8, 10]

Trong React thì đây là cách khi chúng ta muốn chuyển đổi một mảng thành một list các phần tử element và đưa vào trong render.

### **Rendering Multiple Components**

Chúng ta có thể tạo ra một tập hợp các components và đưa chúng vào trong JSX bằng cách viết vào trong cặp dấu {} .

Ví dụ dưới đây tạo ra một list các element <li>

const numbers = [1, 2, 3, 4, 5];

const listItems = numbers.map((number) =>

<li>{number}</li>

);

sau đó đưa vào trong JSX để render vào DOM

ReactDOM.render(

<ul>{listItems}</ul>,

document.getElementById('root')

);

bởi vì khi code chúng ta thường phải sử dụng các dạng list này nhiều lần nên chúng ta có thể tạo ra một component dạng list để sau này còn có thể tái sử dụng

function NumberList(props) {

const numbers = props.numbers;

const listItems = numbers.map((number) =>

<li>{number}</li>

);

return (

<ul>{listItems}</ul>

);

}

const numbers = [1, 2, 3, 4, 5];

ReactDOM.render(

<NumberList numbers={numbers} />,

document.getElementById('root')

);

nhưng với cách viết trên sẽ bị báo thiếu key cho các element <li> vì vậy chúng ta sẽ thêm thuộc tính key vào cho từng <li> như sau:

function NumberList(props) {

const numbers = props.numbers;

const listItems = numbers.map((number) =>

<li key={number.toString()}>

{number}

</li>

);

return (

<ul>{listItems}</ul>

);

}

const numbers = [1, 2, 3, 4, 5];

ReactDOM.render(

<NumberList numbers={numbers} />,

document.getElementById('root')

);

## **Keys**

Key giúp React xác định từng element khác nhau để nắm được item nào bị thay đổi, thêm vào hoặc bị xoá đi. Với điều kiện key phải là giá trị không bị trùng với item khác trong một list, ví dụ sử dụng id làm key:

const todoItems = todos.map((todo) =>

<li key={todo.id}>

{todo.text}

</li>

);

nhưng khi không có id thì chúng ta có thể sử dụng index của mảng để làm key, index được truyền vào trong tham số thứ 2 của hàm map():

const todoItems = todos.map((todo, index) =>

// Only do this if items have no stable IDs

<li key={index}>

{todo.text}

</li>

);

### **Extracting Components with Keys**

Lưu ý với việc gán key sao cho đúng cách. Ví dụ cách sau sai:

function ListItem(props) {

const value = props.value;

return (

// Wrong! There is no need to specify the key here:

<li key={value.toString()}>

{value}

</li>

);

}

function NumberList(props) {

const numbers = props.numbers;

const listItems = numbers.map((number) =>

// Wrong! The key should have been specified here:

<ListItem value={number} />

);

return (

<ul>

{listItems}

</ul>

);

}

const numbers = [1, 2, 3, 4, 5];

ReactDOM.render(

<NumberList numbers={numbers} />,

document.getElementById('root')

);

bởi vì key chỉ gán cho các items của mảng khi chúng ta tạo ra một list các items bằng vòng lặp thay vì gán cho từng item riêng lẻ bên trong một component, ví dụ sau đúng:

function ListItem(props) {

// Correct! There is no need to specify the key here:

return <li>{props.value}</li>;

}

function NumberList(props) {

const numbers = props.numbers;

const listItems = numbers.map((number) =>

// Correct! Key should be specified inside the array.

<ListItem key={number.toString()}

value={number} />

);

return (

<ul>

{listItems}

</ul>

);

}

const numbers = [1, 2, 3, 4, 5];

ReactDOM.render(

<NumberList numbers={numbers} />,

document.getElementById('root')

);

### **Keys Must Only Be Unique Among Siblings**

Các key phải không được trùng nhau giữa các item ngang hàng cạnh nhau.

### **Embedding map() in JSX**

Chúng ta còn một cách viết khác khi nhúng map() vào trong JSX:

function NumberList(props) {

const numbers = props.numbers;

return (

<ul>

{numbers.map((number) =>

<ListItem key={number.toString()}

value={number} />

)}

</ul>

);

}

tức là viết trực tiếp bên trong return của hàm render.

# **Forms**

Viết form ở trong React cũng giống với HTML và có toàn bộ các hành vi mặc định trong trình duyệt.

class NameForm extends React.Component {

constructor(props) {

super(props);

this.state = {value: ''};

this.handleChange = this.handleChange.bind(this);

this.handleSubmit = this.handleSubmit.bind(this);

}

handleChange(event) {

this.setState({value: event.target.value});

}

handleSubmit(event) {

alert('A name was submitted: ' + this.state.value);

event.preventDefault();

}

render() {

return (

<form onSubmit={this.handleSubmit}>

<label>

Name:

<input type="text" value={this.state.value} onChange={this.handleChange} />

</label>

<input type="submit" value="Submit" />

</form>

);

}

}

\* Bài tập: Viết một chương trình đăng ký form submit với các thông tin như request demo theo mẫu sau:

* Name (required)
* Email (required)
* Please choose your preferred contact channel: \* (Skype, Phone) dạng radio
* Phone (nếu chọn Phone)
* Skype (nếu chọn Skype)
* Website URL (required)
* Comment (required)
* I'm interested in: \* Native App/Progressive Web App (dạng select box)

sau khi form được submit thì ghi log ra console các thông tin đã nhập.

