

**REACT-JS Cơ bản**

|  |
| --- |
| - Lifting State Up |
| - Composition vs Inheritance |

# **Lifting State Up**

Trong React thường có một số components cần cập nhật dữ liệu đã được thay đổi giống nhau, đó là cách mà “lifting state up” được tổ chức bằng cách chia sẻ state với nhau thông qua component cha gần nhất của chúng. Hãy xem đoạn code sau về một vấn đề mà chúng ta sẽ gặp phải tương tự sau này:

Đầu tiên chúng ta có 1 component để thông báo lúc nào thì nước đạt nhiệt độ sôi:

function BoilingVerdict(props) {

if (props.celsius >= 100) {

return <p>The water would boil.</p>;

}

return <p>The water would not boil.</p>;

}

const scaleNames = {

c: 'Celsius',

f: 'Fahrenheit'

};

class TemperatureInput extends React.Component {

constructor(props) {

super(props);

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

this.state = {temperature: ''};

}

handleChange(e) {

this.setState({temperature: e.target.value});

}

render() {

const temperature = this.state.temperature;

const scale = this.props.scale;

return (

<fieldset>

<legend>Enter temperature in {scaleNames[scale]}:</legend>

<input value={temperature}

onChange={this.handleChange} />

<BoilingVerdict

celsius={parseFloat(temperature)} />

</fieldset>

);

}

}

class Calculator extends React.Component {

render() {

return (

<div>

<TemperatureInput scale="c" />

<TemperatureInput scale="f" />

</div>

);

}

}

Chúng ta thấy vấn đề của code ở trên là khi chúng ta nhập nhiệt độ vào một trong hai trường input thì input nào chỉ biết sự thay đổi của input đó, mà yêu cầu của chúng ta là mong muốn nhiệt độ được cập nhật và chuyển đổi tương ứng ở cả hai input với nhau.

Ta sẽ đi viết lại code trên theo “lifting state up” như sau:

const scaleNames = {

c: 'Celsius',

f: 'Fahrenheit'

};

function BoilingVerdict(props) {

if (props.celsius >= 100) {

return <p>The water would boil.</p>;

}

return <p>The water would not boil.</p>;

}

function toCelsius(fahrenheit) {

return (fahrenheit - 32) \* 5 / 9;

}

function toFahrenheit(celsius) {

return (celsius \* 9 / 5) + 32;

}

function tryConvert(temperature, converter) {

const input = parseFloat(temperature);

if (Number.isNaN(input)) {

return '';

}

const output = converter(input);

const rounded = Math.round(output \* 1000) / 1000;

return rounded.toString();

}

class TemperatureInput extends React.Component {

constructor(props) {

super(props);

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

}

handleChange(e) {

this.props.onTemperatureChange(e.target.value);

}

render() {

const temperature = this.props.temperature;

const scale = this.props.scale;

return (

<fieldset>

<legend>Enter temperature in {scaleNames[scale]}:</legend>

<input value={temperature}

onChange={this.handleChange} />

</fieldset>

);

}

}

class Calculator extends React.Component {

constructor(props) {

super(props);

this.handleCelsiusChange = this.handleCelsiusChange.bind(this);

this.handleFahrenheitChange = this.handleFahrenheitChange.bind(this);

this.state = {temperature: '', scale: 'c'};

}

handleCelsiusChange(temperature) {

this.setState({scale: 'c', temperature});

}

handleFahrenheitChange(temperature) {

this.setState({scale: 'f', temperature});

}

render() {

const scale = this.state.scale;

const temperature = this.state.temperature;

const celsius = scale === 'f' ? tryConvert(temperature, toCelsius) : temperature;

const fahrenheit = scale === 'c' ? tryConvert(temperature, toFahrenheit) : temperature;

return (

<div>

<TemperatureInput

scale="c"

temperature={celsius}

onTemperatureChange={this.handleCelsiusChange} />

<TemperatureInput

scale="f"

temperature={fahrenheit}

onTemperatureChange={this.handleFahrenheitChange} />

<BoilingVerdict

celsius={parseFloat(celsius)} />

</div>

);

}

}

Như trên ta có thể thấy là các component input sẽ sử dụng state của component Calculator để chia sẻ thông tin, dữ liệu qua nhau.

# **Composition vs Inheritance**

React có một mô hình thành phần cấu tạo mạnh mẽ, và nên sử dụng mô hình này thay vì sử dụng cách kế thừa để dụng lại code của các components.

## **Containment**

function FancyBorder(props) {

return (

<div className={'FancyBorder FancyBorder-' + props.color}>

{props.children}

</div>

);

}

function WelcomeDialog() {

return (

<FancyBorder color="blue">

<h1 className="Dialog-title">

Welcome

</h1>

<p className="Dialog-message">

Thank you for visiting our spacecraft!

</p>

</FancyBorder>

);

}

Đôi lúc chúng ta cũng cần tạo ra nhiều vị trí nội dung con bên trong component bằng cách truyền component vào trong props và gọi chúng ra ở các ví trí khác nhau:

function SplitPane(props) {

return (

<div className="SplitPane">

<div className="SplitPane-left">

{props.left}

</div>

<div className="SplitPane-right">

{props.right}

</div>

</div>

);

}

function App() {

return (

<SplitPane

left={

<Contacts />

}

right={

<Chat />

} />

);

}

## **Specialization**

Sự chuyên môn hoá component

function Dialog(props) {

return (

<FancyBorder color="blue">

<h1 className="Dialog-title">

{props.title}

</h1>

<p className="Dialog-message">

{props.message}

</p>

{props.children}

</FancyBorder>

);

}

function WelcomeDialog() {

return (

<Dialog

title="Welcome"

message="Thank you for visiting our spacecraft!" />

);

}

class SignUpDialog extends React.Component {

constructor(props) {

super(props);

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

this.handleSignUp = this.handleSignUp.bind(this);

this.state = {login: ''};

}

render() {

return (

<Dialog title="Mars Exploration Program"

message="How should we refer to you?">

<input value={this.state.login}

onChange={this.handleChange} />

<button onClick={this.handleSignUp}>

Sign Me Up!

</button>

</Dialog>

);

}

handleChange(e) {

this.setState({login: e.target.value});

}

handleSignUp() {

alert(`Welcome aboard, ${this.state.login}!`);

}

}

## **Inheritance**

Trường hợp này dùng để thừa kế các components mà cần viết lại logic phức tạp bên trong component được thừa kế.

# **Introducing Hooks**

Hooks trong React được thêm vào từ phiên bản React 16.8. Chúng giúp chúng ta có thể sử dụng được state và các tính năng khác mà không cần phải viết lại class.

Hooks là một hàm trong javascript, nhưng chúng yêu cầu bắt buộc tuân theo 2 rule sau:

* Hooks chỉ được gọi ở mức 1. Không được gọi Hooks bên trong vòng lặp, trong lệnh điều kiện, hoặc lồng bên trong một hàm.
* Chỉ được gọi Hooks bên trong function components. Không được gọi Hook bên trong hàm của javascript thông thường.

import React, { useState } from 'react';

function Example() {

// Declare a new state variable, which we'll call "count"

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

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

## **State Hook**

Sử dụng useState:

import React, { useState } from 'react';

function Example() {

// Declare a new state variable, which we'll call "count"

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

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

## **Effect Hook**

Effect hook tương đương với componentDidMount, componentDidUpdate, và componentWillUnmount trong react class.

import React, { useState, useEffect } from 'react';

function Example() {

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

// Similar to componentDidMount and componentDidUpdate:

useEffect(() => {

// Update the document title using the browser API

document.title = `You clicked ${count} times`;

});

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

sử dụng Effect với “clean up”

import React, { useState, useEffect } from 'react';

function FriendStatus(props) {

const [isOnline, setIsOnline] = useState(null);

function handleStatusChange(status) {

setIsOnline(status.isOnline);

}

useEffect(() => {

ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);

return () => {

ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);

};

});

if (isOnline === null) {

return 'Loading...';

}

return isOnline ? 'Online' : 'Offline';

}

Chúng ta có thể sử dụng nhiều State, và Effect Hooks trong cùng một component:

function Form() {

// 1. Use the name state variable

const [name, setName] = useState('Mary');

// 2. Use an effect for persisting the form

useEffect(function persistForm() {

localStorage.setItem('formData', name);

});

// 3. Use the surname state variable

const [surname, setSurname] = useState('Poppins');

// 4. Use an effect for updating the title

useEffect(function updateTitle() {

document.title = name + ' ' + surname;

});

// ...

}

**Bài tập:**

**Yêu cầu viết lại form demo có sử dụng React Hooks (viết các class component thành function component)**