**Làm thế nào để kết hợp React Hooks với Class Component**

4.754 lượt xem - Thời gian đọc: 2 phút đọc

Mặc dù mục tiêu phát triển của **React Hooks** từ **functional component** sinh ra không phải phục vụ cho mục đích sử dụng nó bên trong một **class components.** Tuy nhiên, đây là một câu hỏi cũng đáng để chúng ta tử. Và đôi khi, trong trường hợp nào đó chúng ta sẽ cần điều này.

**Giải pháp**

Để bắt đầu giải quyết vấn đề, chúng ta sẽ tạo một custom hook **useScreenWidth** đơn giản làm ví dụ. Mục đích của nó là lấy chiều rộng màn hình thực tế, ở mọi khung hình

import { useEffect, useState } from 'react';

export function useScreenWidth() {

const [width, setWidth] = useState(window.innerWidth);

useEffect(() => {

const handler = (event) => {

setWidth(event.target.innerWidth);

};

window.addEventListener('resize', handler);

return () => {

window.removeEventListener('resize', handler);

};

}, []);

return width;

}

Trong đây sử dụng **useState** và **useEffect** được tích hợp sẵn trong React. Để tìm hiểu về 2 react hooks này, bạn có thể đọc hai bài viết bên dưới

* [Tìm hiểu về useState()](https://codestus.com/posts/react-hooks-series-tim-va-hieu-ve-usestate)
* [Tìm hiểu về useEffect()](https://codestus.com/posts/tim-hieu-ve-react-hook-useeffect)

Giờ thì bắt đầu thôi

**Sử dụng HOC (Higher Order Component)**

Đây là một giải pháp hết sức, phổ biến trong nhiều trường hợp nó tương tự như [HOF trong JS](https://eloquentjavascript.net/05_higher_order.html), **Redux** cũng dùng cách này để viết để giải quyết các vấn đề của họ.

**Higher Order Component (HOC)** là một kỹ thuật nâng cao trong React, giúp xử lý các logic cần tính toán trước khi **render** một **component**. Sau đó trả về một **Component** mà chúng ta muốn kết xuất.

Các giá trị sau khi tính toán có thể truyền xuống **<Component />** dưới dạng **props** thông thường. Đây cũng là cách để chúng ta tính toán sử dụng **React Hooks** kết hợp với các **Class Components.**

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

import useScreenWidth from 'src/custom-hooks/useScreenWidth'

export default function withWidthScreenHook(Component) {

return function(props) {

const widthScreen = useScreenWidth();

return <Component width={widthScreen} {...props} />

}

}

Bước cuối cùng là chỉ cần bọc thành phần lớp hiện có của chúng ta bằng HOC đó. Và sau đó, chúng ta chỉ đơn giản sử dụng **this.props.width** như bất kỳ **props** nào khác được truyền cho thành phần component.

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

import withWidthScreenHook from 'src/hoc/withWidthScreenHook'

class App extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<p>Wdith: {this.props.width}</p>

);

}

}

export default withWidthScreenHook(App)

Bản demo: [Tại đây](https://codesandbox.io/s/stupefied-bardeen-i14kt?file=/src/App.js)

**Sử dụng Function Component**

Ngoài sử dụng Higher Order Component (HOC), bạn vẫn có thể sử dụng **Function Component** để xây dựng các hooks bên trong nó và sử dụng các giá trị đó bên trong **Class Component**.

import React from 'react'

import useScreenWidth from 'src/custom-hooks/useScreenWidth'

funtion ScreenWidth() {

const width = useScreenWidth();

return React.children(width);

}

Tương tự như thế, chúng ta gọi **Function Component** trong **Class Component** và lấy các giá trị, cũng như phương thức mà chúng ta định nghĩ dưới dạng một **callback**:

import React from 'react'

import ScreenWidth from 'src/components/ScreenWidth'

class App extends React.Component {

render() {

return (

<ScreenWidth>

{(width) => (

<p>Width: {width}px</p>

)}

</ScreenWidth>

);

}

}

**Kết luận**

Tóm tắt lại, chúng ta có 2 cách để sử dụng các **React Hooks kể từ phiên bản v16.8** bằng 2 cách bạn có thể tham khảo:

* **Sử dụng HOC (Higher Order Component)**
* **Sử dụng Function Component**