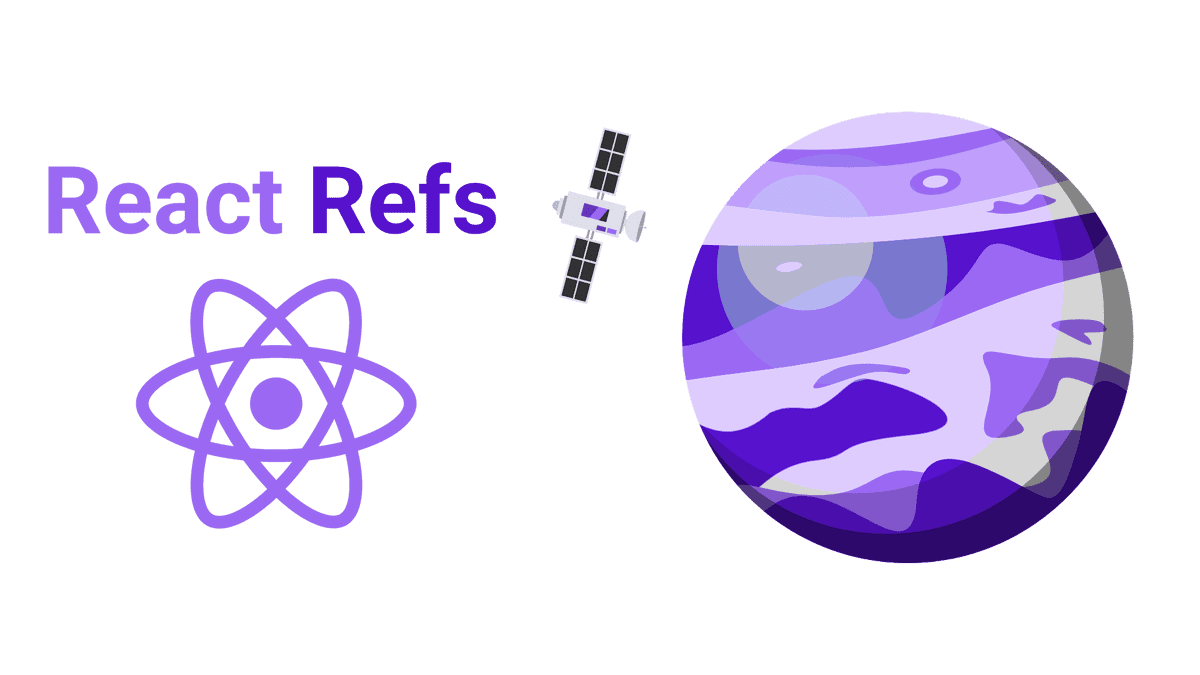
**Tất cả những gì bạn cần biết về Ref trong React**

1.850 lượt xem - Thời gian đọc: 5 phút đọc



**React ref là gì?**

React Ref (React reference) hiểu đơn giản là một đối tượng tham chiếu đến một biến, một component giữ cho giá trị của nó không thay đổi giữa các lần **render** và truy xuất các giá trị đó thông qua key **current**.

**Làm thế nào để sử dụng nó?**

**Trước phiên bản 16.8.6**

Với các phiên bản trước **16.8.6** bạn chỉ có thể sử dụng cách sau ở **class component**

import React from 'react'

// Standard

class MyComponent extends React.Component {

constructor() {

this.myRef = React.createRef();

}

render() {

return <div>Demo</div>

}

}

// Quick define

class MyComponent extends React.Component {

myRef = React.createRef();

render() {

return <div>Demo</div>

}

}

*Lưu ý rằng 2 cách ở đay đều như nhau, nhưng với việc khai báo bằng constructor bạn có thể gán****defaultValue****cho****ref***

**Từ phiên bản 16.8.6 về sau**

Bạn có thể sử dụng hooks **useRef** với **FC** vô cùng ngắn gọn như sau:

import { useRef } from 'react'

function MyComponent() {

const myRef = useRef(null);

return <div>demo</div>

}

**Truy xuất giá trị và cập nhật giá trị cho ref**

Sau khi khởi tạo, bạn chỉ việc truy xuất key **current** để lấy giá trị hoặc gán cho ref một giá trị mà cần giữ cho nó không thay đổi giữa các lần render

import { useRef } from 'react'

function MyComponent() {

const myRef = useRef(null);

useEffect(() => {

myRef.current = "Hello world";

}, []);

console.log(myRef.current);

return (<div>Demo</div>)

}

// Result 1: null

// Result 2: Hello world

*Nhưng có một lưu ý mà đã được nhắc từ đầu về****ref****nó sẽ không trigger render lại component. Vì thế, nếu không sử dụng****state****hoặc một số kỹ thuật khác để trigger render lại component. Bạn sẽ không thấy được giá trị của****ref****thay đổi trên giao diện.*

**Những điều không nên làm với ref**

Bạn không bao giờ nên cập nhật hoặc đọc trực tiếp tham chiếu trong trong khi render, ngoại lệ duy nhất là với loại **Lazy initialization**

**Mà Lazy initialization là gì?**

Là khi bạn kiểm tra ref không có giá trị thì mới gán cho nó một giá trị mới. Nó chỉ hữu ích khi bạn sử dụng với **React.createPortal**

// should not

function MyComponent() {

const container = useRef();

container.current = document.querySelector("#container");

return ....

};

// √

function MyComp() {

const container = useRef();

if(!container)

container.current = document.querySelector("#container");

return React.createPortal(<Demo />, container.current);

}

**Lý do tại sao không nên làm vậy?**

Đó là do cơ chế **concurrent rendering**. Với **concurrent mode** , quá trình này sẽ diễn ra bất đồng bộ, có thể việc hiển thị của một số thành phần sẽ bị “tạm dừng” để giữ nhiều nhất 60 frames/giây để giữ cho hiệu xuất tương tác vẫn ở mức tốt.

Vì vậy, sẽ có thể tạo ra sự không thống nhất nếu một tham chiếu được sử dụng trong quá trình **rerender**

**Khi nào nên sử dụng?**

Okay, bây giờ chúng ta đã biết compoennt sẽ không re-render sao khi thay đổi tham chiếu **ref**. Vậy nó hữu dụng khi nào?.

Có rất nhiều trường hợp ở đây, xem qua thử nhé.

**Tham chiếu đến real DOM Elements**

Trong trường hợp này, bạn có thể đặt **ref** cho các thể **jsx** như sau

function MyComp() {

const inputRef = useRef(null);

return <input type="text" ref={inputRef} />

}

Sau đó, bạn có thể thử truy cập DOM element bằng **inputRef.current**

Cho một ví dụ như sau

function MyComp() {

const inputRef = useRef(null);

const onSubmitForm = (e) => {

e.preventDefault();

console.log(inputRef.current.value);

}

return <form onSubmit={onSubmitForm}>

<input type="text" ref={inputRef} />

<button htmlType="submit">Submit</button>

</form>

}

Việc này sẽ giảm thiểu được lượng lớn lần **re-render lại component** nếu chúng ta sử dụng các thông thường bằng cách đặt và thay đổi **state** của input.

*Lưu ý: Nếu phần tử được hiển thị có điều kiện, bạn có thể thích sử dụng callback kết hợp với ref*

function MyComp() {

const [isShowingForm, setShowingForm] = useState(false);

const inputRef = useRef(null);

const refCallback = useCallback((node) => {

console.log(node);

}, [])

const onSubmitForm = (e) => {

e.preventDefault();

console.log(inputRef.current.value);

}

return <>

<button htmlType="button" onClick={() => setShowingForm(!isShowingForm)}>

{isShowingForm ? "To Off" : "To On"}

</button>

{

isShowingForm && <form onSubmit={onSubmitForm}>

<input type="text" ref={refCallback} />

<button htmlType="submit">Submit</button>

</form>

}

</>

}

*Lưu ý rằng: Bạn có thể thấy sử dụng phải sử dụng****useCallback****bên trong****ref****. Nếu không****useCallback****sẽ được gọi lại ở mỗi lần hiển thị, điều này sẽ ảnh hưởng không tốt đến tính nhất quán.*

**Sử dụng forwardRef cho component**

Trong trường hợp bạn muốn **ref** một **component** chứ không phải là các giá trị nhỏ bên trong một component nữa. Chúng ta sẽ cần quan tấm đến [forwardRef](https://reactjs.org/docs/forwarding-refs.html).

// forwardRef

const FComp = forwardRef((props, ref) => {

return <div {...props} ref={ref}></div>

})

// the another ways

function FCompo({ customRef, ...props }) {

return <div {...props} ref={customRef}></div>

}

**Áp dụng**

Một số trường hợp khác là lưu trữ giá trị không cần kích hoạt render lại, chẳng hạn như khi bạn chỉ sử dụng nó trong trình nghe sự kiện.

**Lắng nghe sự kiện**

**Đếm số lần kết xuất**

Việc này sẽ hữu ích nếu bạn cần đo đếm số lần re-render của component để cân nhắc việc tối ưu hiệu xuất.

Đến đây có vẻ hơi dài, vì vậy mình chỉ nêu một số ứng dụng mà bạn có thể thử làm với ref trong một số cases.

**Làm thế nào React assign DOM node cho ref?**

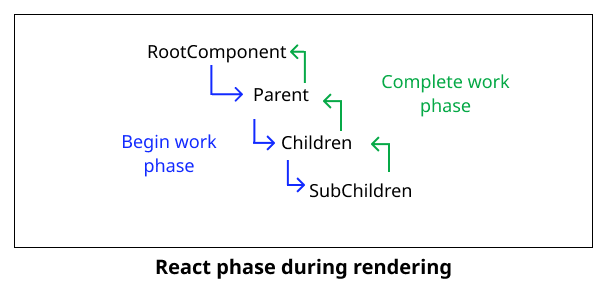
Trước đó, chúng ta đã thấy trường hợp sử dụng chính là tham chiếu đến một DOM node. React làm thế như nào nhỉ?

Một điều bạn sẽ cần hiểu giữa **useEffect** và **useLayoutEffect** hooks: **useLayoutEffect** sẽ thực thi đồng bộ sau giai đoạn **render** trái ngược với **useEffect** thực thi bất đồng bộ.

Trong lần đầu tiên, React sẽ biến đổi các React elements thành các **Fiber node**

[**Fiber node**](https://www.notion.so/Fiber-node-2aea31bb509e415ba5f85a781f5b7e53)

Về cơ bản, trong quá trình **render**, React sẽ xử lý từ node gốc cho đến thành phần sâu nhất. Sau đó, nó sẽ đi lên trong cây component.



*Lưu ý: Ở đây chúng ta có một cây đơn giản không có thành phần anh chị em khác. Khi có anh chị em, nó xử lý một nhánh, hoàn thành công việc và chuyển sang nhánh khác.*

**Bắt đầu giai đoạn làm việc với ref**

Khi xử lý một node, từ trên xuống dưới, React có thể phát hiện khi nào một node là HostComponent (tức là các thẻ div, p, ...) và được gán một ref cho nó. Nếu đúng như vậy, React sẽ gắn cờ node này và đặt trên node đó một **“fiber node”** ref chứa tham chiếu đến phần tử (về cơ bản là một đối tượng có khóa hiện tại như chúng ta đã thấy trước đó).

**Kết thúc giai đoạn**

Sau đó, khi React đi đến node cuối cùng, nó sẽ xuất hiện trên cây, tại thời điểm này, các cờ **ref** đã thiết lập trước đó có hiệu lực. Nó sẽ thông báo cho các **“fiber node” cha.**

Và cuộc trao đổi này diễn ra với từng **“fiber node”** cho đến khi chúng ta quay lại **“fiber node parent”.**

**Kết luận**

React ref có nhiều trường hợp sử dụng mà chúng ta đã thấy trước đây, đừng ngần ngại cho biết khi nào bạn đang sử dụng chúng.