**useRef và forwardRef trong React**

**Nội dung bài viết**

* [ref là gì?](https://xdevclass.com/useref-va-forwardref-trong-react/#ref_la_gi)
* [Truy cập DOM nodes hoặc React elements](https://xdevclass.com/useref-va-forwardref-trong-react/#Truy_cap_DOM_nodes_hoac_React_elements)
* [Lưu giữ một biến có thể mutate](https://xdevclass.com/useref-va-forwardref-trong-react/#Luu_giu_mot_bien_co_the_mutate)
  + [Trong class component](https://xdevclass.com/useref-va-forwardref-trong-react/#Trong_class_component)
  + [Trong functional component](https://xdevclass.com/useref-va-forwardref-trong-react/#Trong_functional_component)
* [forwardReef là gì](https://xdevclass.com/useref-va-forwardref-trong-react/#forwardReef_la_gi)
* [Tóm lại](https://xdevclass.com/useref-va-forwardref-trong-react/#Tom_lai)
* [Tham khảo](https://xdevclass.com/useref-va-forwardref-trong-react/#Tham_khao)

Trong bài viết này, cùng mình khám phá một hook khá là hay ho của React, đó là useRef. Chúng ta dùng nó cho việc gì và một số lời khuyên dành cho các bạn.

useRef hook là một function trả về một object với thuộc tính current được khởi tạo thông qua tham số truyền vào. Object được trả về này có thể mutate và sẽ tồn tại xuyên suốt vòng đời của component.

const refContainer = useRef**(**initialValue**)**;

Có 2 lý do chính mà chúng ta sẽ sử dụng useRef: **Truy cập DOM nodes hoặc React elements** và **lưu giữ một biến có thể mutate.**

Đọc thêm:

* [Thay đổi mutate state React sao cho chuẩn](https://xdevclass.com/loi-pho-bien-trong-react-mutate-state/)
* [Bạn đã thực sự hiểu về tham trị và tham chiếu Javascript?](https://xdevclass.com/ban-da-thuc-su-hieu-ve-tham-tri-va-tham-chieu-javascript/)

Nhưng trước tiên tìm hiể useRef thì chúng ta nên tìm hiểu ref là cái gì.

**ref là gì?**

Trong React, ref là một thuộc tính của một tag hay một element đại diện cho chính nó. ref cho phép chúng ta truy cập đến DOM node hoặc React element đã được mount. Trong vanila Javascript, chúng ta làm việc với DOM elements bằng cách gọi document.getElementById(). Với ref trong React chúng ta không cần phải làm vậy. Thuộc tính ref sẽ tham chiếu đến chính xác element cần dùng.

**Ví dụ:**

**<input** type="text" ref={textInput} **/>**

ref nhận vào một biến hoặc một function. Nếu là function thì function này sẽ được chạy khi element được mount.

**Ví dụ:**

**<button** ref={(element) =**>** console.log(element)}>Send**</button>**

Đây là kết quả element  
log

**Truy cập DOM nodes hoặc React elements**

Nếu bạn làm việc với React được một khoảng thời gian rồi thì bạn có thể đã từng sử dụng ref cho việc này. Dưới đây là ví dụ về việc sử dụng ref trong class component:

**import** React, **{** Component, createRef **}** from "react";

**class** CustomTextInput **extends** Component **{**

textInput = createRef**()**;

focusTextInput = **()** => this.textInput.current.focus**()**;

render**()** **{**

**return** **(**

**<>**

**<**input type="text" ref=**{**this.textInput**}** />

<button onClick={this.focusTextInput}>Focus the text input</button**>**

**<**/**>**

**)**;

**}**

**}**

Và nó tương đương với functional component

**import** React, **{** useRef **}** from "react";

const CustomTextInput = **()** => **{**

const textInput = useRef**()**;

focusTextInput = **()** => textInput.current.focus**()**;

**return** **(**

**<>**

**<**input type="text" ref=**{**textInput**}** />

<button onClick={focusTextInput}>Focus the text input</button**>**

**<**/**>**

**)**;

**}**

Lưu ý là trong functional component thì chúng ta sử dụng useRef thay vì sử dụng createRef. Nếu chúng ta tạo một ref bằng cách sử dụng createRef trong một functional component, React sẽ tạo mới một instance ref mỗi lần re-render thay vì giữ nguyên instance xuyên suốt các quá trình render.

**Lưu giữ một biến có thể mutate**

Cả trong class component và functional component sử dụng hook, chúng ta có 2 cách để giữ data không bị tạo mới lại giữa các lần re-render:

**Trong class component**

* **Trong state của component:** Mỗi lần state thay đổi, component sẽ bị re-render.
* **Trong một biến instance:** Biến sẽ tồn tại suốt vòng đời của component. Thay đổi instance của biến sẽ không gây nên re-render.

**Trong functional component**

* **Trong state của component:** useState hoặc useReducer. Cập nhật biến state sẽ gây nên re-render component.
* **Trong một ref:** Tương đương với các biến instance trong class component. Việc mutate thuộc tính .current sẽ không gây nên re-render.

Chúng ta cần để ý là trong một ứng dụng React, không nhất thiết phải re-render lại component khi chúng ta cập nhật một giá trị. Hãy xem ví dụ bên dưới

**import** React, **{**useRef, useState**}** from 'react'

const MessageInputComponent = **()** => **{**

const **[**message, setMessage**]** = useState**(**""**)**

const sentMessage = useRef**(**0**)**;

const sendMessage = **()** => **{**

**if(**sentMessage.current === 3**){**

**return** alert**(**"Message Limit Reached"**)**

**}**

sentMessage.current += 1

//code to handle sending message

**}**

**return(**

**<**div**>**

**<**input onChange = **{(**e**)** => setMessage**(**e.target.value**)}** value=**{**message**}**/>

<button onClick={sendMessage}>Send</button**>**

**<**/div**>**

**)**

**}**

**export** **default** MessageInputComponent

Ở đây, chúng ta sử dụng cả useState và useRef. message state đại diện cho value của input, khi input thay đổi thì components sẽ re-render và update value cho input. sentMessage ref có nhiệm vụ đếm số lần message được gửi đi, và nó không yêu cầu component re-render khi sentMessage thay đổi.

Để ý cách mà giá trị useRef thay đổi. Nó không cần phải dùng phương thức set như bên useState. Để thay đổi giá trị, chúng ta chỉ cần thay đổi trực tiếp giá trị được lưu trong thuộc tính current của useRef.

**forwardReef là gì**

Như chúng ta đã tìm hiểu bên trên thì ref giúp chúng ta truy cập đến một element, vậy nó có thể truy cập đến một component React hay không? test thử nhé

**import** React from "react";

const Input = **()** => **<**input type="text" style=**{**style**}** /**>**;

**export** **default** Input;

**import** React, **{** useRef, useEffect **}** from "react";

**import** Input from "./Input";

**function** App**()** **{**

const inputRef = useRef**(null)**;

useEffect**(()** => **{**

**if** **(**inputRef.current**)** **{**

inputRef.current.focus**()**;

**}**

console.log**({** inputRef **})**;

// eslint-disable-next-line react-hooks/exhaustive-deps

**}**, **[])**;

**return** **(**

**<**div**>**

**<**Input ref=**{**inputRef**}** />

</div**>**

**)**;

**}**

**export** **default** App;

Chúng ta sẽ nhận một thông báo tham chiếu đến **Input Component** là null

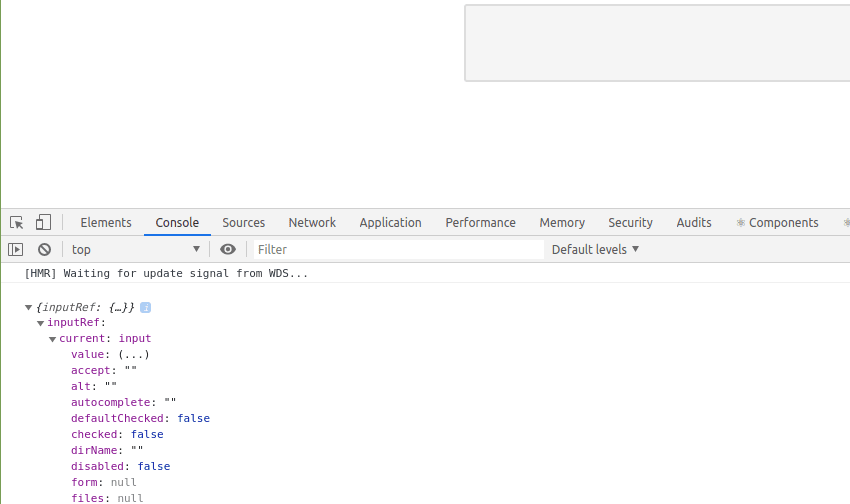


Để fix vấn đề này ta dùng forwardRef như một HOC cho **Input Component**

**import** React, **{** forwardRef **}** from "react";

const Input = **(**props, ref**)** => **<**input ref=**{**ref**}** type="text" style=**{**style**}** /**>**;

**export** **default** forwardRef**(**Input**)**;



**Tóm lại**

* Một ref được tạo khi component đã mount. ref được gán cho một element, muốn truyền ref thông qua component thì dùng forwardRef.
* Ref có thể được sử dụng để truy cập đến DOM node hoặc React element. Ngoài ra còn dùng để lưu trữ các biến có thể mutate mà không làm re-render component.

Hy vọng bài viết giúp mọi người hiểu được ref, useRef và forwardRef và cách sử dụng của chúng. Hẹn mọi người ở các bài viết khác 😛

**Tham khảo**

* <https://medium.com/javascript-in-plain-english/implementing-useref-in-react-732908aa1998>
* <https://medium.com/trabe/react-useref-hook-b6c9d39e2022>
* <https://dev.to/renatobentorocha/how-to-use-react-forwardref-1h2l>