REACT HOOKS

Trong ReactJs chúng ta biết rằng, có 2 kiểu viết component: Class Component và Functional Component.

Trong đó, functional Component có những hạn chế nhất định như: không có state, không có life-cycle method...

Nhưng functional component có ưu điểm là dễ unit test hơn, phù hợp với trường phái viết code theo kiểu hướng function thay vì hướng đối tượng (OOP).

React Hooks ra đời để khắc phục những nhược điểm của functional component .

Cho tới thời điểm hiện tại dựa trên trang chính thức của React, có các Hooks sau:

- Basic Hooks
    - useState
    - useEffect
    - useContext
- Additional Hooks
    - useReducer
    - useCallback
    - useMemo
    - useRef
    - useImperativeHandle
    - useLayoutEffect
    - useDebugValue

1/ UseState

Dùng khi nào?

Khi muốn dữ liệu thay đổi thì giao diện tự động được cập nhật (render lại theo dữ liệu).

Cú pháp sử dụng

useState Hook cho phép chúng ta khai báo 1 local state trong functional component với cú pháp:

`const [state, setState] = useState(initialStateValue)`

Trong đó:

- **state:** định nghĩa tên biến để lưu giá trị của state, giá trị của state có thể là đơn giá trị, một object, hay một mảng, .v.v.
- **setState:** là hàm dùng để thay đổi giá trị của state, việc cập nhật giá trị của state bắt buộc phải thông qua hàm này.
- **initialStateValue**: là giá trị khởi tạo của state, giá trị này chỉ có ý nghĩa duy nhất 1 lần khi state được "sinh ra" thui nha!

Tham khảo mã nguồn ví dụ của useState tại đây :

https://github.com/longden214/react-hooks-tutorial/tree/useState

(*) Lưu ý

- Component được re-render sau khi **`setState`**

- InitialStateValue chỉ được dùng cho lần đầu

- Set State với callback

- InitialStateValue với callback

- Set State là thay thế state bằng giá trị mới

2/ UseEffect

UseEffect giúp dễ dàng quản lý lifecycle của bất kì functional component nào.

Dùng khi nào?

Khi các bạn muốn thực hiện Side Effect(khi dữ liệu của chương trình thay đổi)

Cú pháp sử dụng

useEffect Hook cho phép chúng ta khải báo sử dụng 1 effect với cú pháp như sau:

`useEffect(callback, paramsArray?)`

Trong đó:

- **callback:** là hàm thực thi một công việc nào đó khi effect được khởi chạy
- **paramsArray (optional):** là một mảng các tham số để theo dõi, khi giá trị các tham số này thay đổi, effect mới được khởi chạy.

Thông thường, khi không cung cấp paramsArray, effect sẽ được chạy trong mọi lần component re-render. Điều này có thể dẫn tới một số vấn đề không mong muốn về hiệu năng do handlerFunction sẽ được gọi rất nhiều lần một cách mất kiểm soát (thậm chí là vô hạn).

Các tình huống của useEffect:

1. useEffect(callback)
   VD: https://github.com/longden214/react-hooks-tutorial/tree/useEffectNotParamsArray
2. useEffect(callback,[])
   VD: https://github.com/longden214/react-hooks-tutorial/tree/useEffectWithDependenciesIsEmpty
3. useEffect(callback,[deps])
   VD: https://github.com/longden214/react-hooks-tutorial/tree/useEffectWithDependenciesIsNotEmpty

(*) Lưu ý:

1. useEffect(callback)
   - Gọi callback mỗi khi component re-render
   - Gọi callback sau khi component thêm element vào DOM
2. useEffect(callback,[])
   - Chỉ gọi callback 1 lần sau khi component mounted và không muốn gọi lại sau khi component re-render
3. useEffect(callback,[deps])
   - Callback sẽ được gọi lại mỗi khi deps thay đổi

------ Lưu ý bao gồm cả 3 trường hợp ----------

- Callback luôn được gọi sau khi component mounted
- Cleanup function luôn được gọi trước khi component unmounted
- Cleanup function luôn được gọi trước khi callback được gọi (trừ lần mounted)
  VD: https://github.com/longden214/react-hooks-tutorial/tree/useEffectWithCleanupFunc

3/ UseLayoutEffect

So sánh quá trình useEffect và useLayoutEffect

useEffect	**useLayoutEffect**
1.Cập nhật lại state	1.Cập nhật lại state
2.Cập nhật lại DOM (mutated)	2.Cập nhật lại DOM (mutated)
3.Render lại UI	3.Gọi Cleanup nếu deps thay đổi
4. Gọi Cleanup nếu deps thay đổi	4.Gọi useLayoutEffect callback
5.Gọi useEffect callback	5.Render lại UI

Dùng khi nào?

useLayoutEffect: Chỉ sử dụng trong các trường hợp bạn cần thay đổi DOM hoặc thực hiện các phép tính đo lường, các trường hợp xảy ra lỗi từ **useEffect. Hãy nghĩ đến useLayoutEffect như một giải pháp.**

Cú pháp sử dụng

```
useLayoutEffect(callback, paramsArray?)
```

Trong đó:

- **callback:** là hàm thực thi một công việc nào đó khi effect được khởi chạy
- **paramsArray (optional):** là một mảng các tham số để theo dõi, khi giá trị các tham số này thay đổi, effect mới được khởi chạy.

VD : https://github.com/longden214/react-hooks-tutorial/tree/useLayoutEffect

4/ useRef

Dùng khi nào?

Lưu các giá trị qua một tham chiếu bên ngoài function component

Cú pháp sử dụng

```
const refContainer = useRef(initialValue)
```

VD: https://github.com/longden214/react-hooks-tutorial/tree/useRef

5/ useCallback

- **useCallback** giúp tránh tạo ra những hàm mới không cần thiết trong function component
- cho phép bạn lưu lại **sự tồn tại của một hàm** vào bộ nhớ giữa các lần re-render của component và chỉ thay đổi khi các sự phụ thuộc được thay đổi.
- Vì thế, bạn có thể sử dụng useCallback để ngăn việc khởi tạo lại một hàm nào đó mỗi lần component được render lại. Điều này có ý nghĩa khi function đó được truyền vào như callback props của nhiều components con.

Cú pháp sử dụng

```
useCallback(callback, paramsArray?)
```

Trong đó:

- **callback:** là hàm thực thi một công việc nào đó khi effect được khởi chạy
- **paramsArray (optional):** là một mảng các tham số để theo dõi, khi giá trị các tham số này thay đổi, mới được khởi chạy.

VD : https://github.com/longden214/react-hooks-tutorial/tree/useCallback

6/ useMemo

Phân biệt ***memo*** và ***useMemo***:

- memo:

+ Là higher-order component

+ Bao quanh component

+ Tránh component được re-render trong những trường hợp không cần thiết

- useMemo:

+ Là một hook

+ Viết trong phần thân của 1 function component

+ Tránh thực hiện lại một logic nào đó không cần thiết

Cú pháp sử dụng

```
useMemo(callback, paramsArray?)
```

Trong đó:

- **callback:** là hàm thực thi một công việc nào đó khi effect được khởi chạy
- **paramsArray (optional):** là một mảng các tham số để theo dõi, khi giá trị các tham số này thay đổi, mới được khởi chạy.

VD: https://github.com/longden214/react-hooks-tutorial/tree/useMemo

7/ useReducer

useReducer là phiên bản nâng cấp của **useState** và cách thức hoạt động của nó giống với React-Redux reducer

Dùng khi nào?

- useState: dùng cho những state đơn giản(kiểu dữ liệu nguyên thủy: number, string, boolean) hoặc array, object. Nhưng các object chỉ có 1 cấp, không bị lồng nhiều cấp con, hoặc số lượng state trong component ít
- useReducer: phù hợp trong các tình huống state trở lên phức tạp hơn, khi có nhiều state trong 1 component, hoặc các state phụ thuộc nhau.

Cú pháp sử dụng

```
const [state, dispatch] = useReducer(reducer, initialState, init);
```

Trong đó:

- **state:** định nghĩa tên biến để lưu giá trị của state, giá trị của state có thể là đơn giá trị, một object, hay một mảng, .v.v.
- **dispatch:** một hành động giúp action được kích hoạt, từ đó dẫn tới việc state được thay đổi
- **reducer:** hàm xử lý action được kích hoạt từ dispatch, trả về giá trị cùng kiểu dữ liệu với state
- **initialState:** là giá trị khởi tạo của state, giá trị này chỉ có ý nghĩa duy nhất 1 lần khi state được "sinh ra" thui nha!
- **init:** chức năng được thực hiện chính xác một lần duy nhất khi useReducerlần đầu tiên được gọi

*Các bước tạo ra useReducer

1. InitState

2. Actions

3. Reducer

4. Dispatch

VD: https://github.com/longden214/react-hooks-tutorial/tree/useReducer

8/ useContext

Giúp đơn giản hóa việc truyền dữ liệu từ component cha xuống các component con mà không cần phải sử dụng tới props

VD: CompA => CompB => CompC

UseContext giúp truyền dữ liệu từ CompA xuống thẳng CompC mà không qua trung gian CompB nữa

Cú pháp sử dụng

useContext Hook cho phép chúng ta truy cập tới 1 context như sau:

```
const value = useContext(Context)
```

Trong đó:

- **value:** sẽ là biến chứa giá trị context hiện tại được trả về.
- **Context:** là React Context mà bạn đã tạo trước đó.

*Các bước tạo Context:

B1. Create context

- Tạo ra 1 phạm vi giúp truyền dữ liệu trong phạm vi đó(Gỉa sử tạo Context ở phạm vi CompA ,nó giúp truyền dữ liệu xuống bất cứ component con nào của compA)

B2. Provider

- Giúp truyền đi dữ liệu

B3. Consumer

- Giúp compC nhận dữ liệu của compA truyền xuống

VD: https://github.com/longden214/react-hooks-tutorial/tree/useContext

9/ useImperativeHandle

- useImperativeHandle: Component child có thể tùy chỉnh các trả về ref cho Component cha. useImperativeHandle phải đi kèm với forwardRef
- Thể hiện tính đóng gói (private), giúp chỉ lấy ra những method hay properties cần thiết

Cú pháp sử dụng

```
useImperativeHandle(ref,callback)
```

Trong đó:

- **ref:** tham chiếu ref được forvard từ trên xuống
- **callback:** return 1 Object chứa những method hoặc properties, được gán lại cho ref được forvard từ trên xuống

VD: https://github.com/longden214/react-hooks-tutorial/tree/useImperativeHandle

10/ useDebugValue

- Được sử dụng với DevTools, để hiển thị các nhãn với các custom hooks
- Cách dùng rất đơn giản:

```
useDebugValue(label)
```

(*)Lưu ý: Không được khuyến khích dùng, chỉ dùng 1 phần cho các shared library