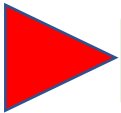


useState

Rsfpsf

1. Giới thiệu hook `useState()`

- Là một hook cơ bản.
- Giúp mình có thể dùng state trong functional component.
- Input: initialState (giá trị hoặc function)
- Output: một mảng có 2 phần tử tương ứng cho `state` và `setState`
- Cách dùng: `const [name, setName] = useState('Hau');`



useState

```
function TodoList() {  
  const [todoList, setTodoList] = useState(['love', 'easy', 'frontend']);  
  
  function removeTodo(index) {  
    // Remember to clone into a new array  
    const newTodoList = [...todoList];  
  
    newTodoList.splice(index, 1);  
    setTodoList(newTodoList);  
  }  
  
  return (  
    <ul className="todo-list">  
      {todoList.map((todo, index) => (  
        <li  
          key={todo.id}  
          onClick={() => removeTodo(index)}  
        >  
          {todo.title}  
        </li>  
      )}}  
    </ul>  
  )
```



5. Những điều lưu ý khi dùng `useState()`

- `useState()` use REPLACING instead of MERGING.

```
// setState() in class component: MERGING
this.state = { name: 'Hau', color: 'red' };
this.setState({ color: 'green' });
// --> { name: 'Hau', color: 'green' }
```

```
// useState() in functional component: REPLACING
const [person, setPerson] = useState({ name: 'Hau', color: 'red' });
setPerson({ color: 'green' });
// --> { color: 'green' }
```

```
// useState() in functional component: REPLACING
const [person, setPerson] = useState({ name: 'Hau', color: 'red' });
setPerson({ ...person, color: 'green' });
// --> { name: 'Hau', color: 'green' }
```



useState chỉ chạy 1 lần thôi

- Initial state chỉ dùng cho lần đầu, những lần sau nó bị bỏ rơi.

```
function ColorBox() {  
  // redundant code for sub-subsequent re-render  
  const initColor = getComplicatedColor();  
  const [color, setColor] = useState(initColor);  
  
  function handleBoxClick() {  
    setColor('green');  
  }  
  
  return (  
    <div  
      className="color-box"  
      style={{ backgroundColor: color }}  
      onClick={handleBoxClick}  
    ></div>  
  );  
}
```



Muốn tính toán gì đó mà chỉ chạy 1 lần thì dùng hàm như trên

```
function ColorBox() {  
  const [color, setColor] = useState(() => {  
    // You're safe here  
    // This function will be called once  
    const initColor = getComplicatedColor();  
    return initColor;  
  }));  
  
  function handleBoxClick() {  
    setColor('green');  
  }  
}
```



Các bạn nhớ nè

- useState() giúp functional component có thể dùng state.
- useState() trả về một mảng 2 phần tử [name, setName].
- useState() áp dụng replacing thay vì merging như bên class component.
- Initial state callback chỉ thực thi 1 lần đầu.



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

function App(props) {
  const [count, setCount] = useState(0);

  function handleCout()
  {
    console.log("Tăng");
    setCount(count + 1);
  }

  function handleCout1(data)
  {
    console.log(data);
    setCount(count - 1);
  }

  return (
    <div>
      {count}
      <button onClick={handleCout}>Tăng</button>
      <button onClick={()=>handleCout1("Giảm")}>Giảm</button>
    </div>
  );
}

export default App;
```



1. **Side effects** là gì? Có bao nhiêu loại?

Side effects là gì?

- Gọi API lấy dữ liệu.
- Tương tác với DOM.
- Subscriptions.
- setTimeout, setInterval.

Theo tài liệu chính thức thì chia làm 2 loại side effects:

1. Effects **không cần clean up**: gọi API, tương tác DOM
2. Effects **cần clean up**: subscriptions, setTimeout, setInterval.

Ref: <https://reactjs.org/docs/hooks-effect.html>



2. Giới thiệu hook `useEffect()`

- Là một hook cơ bản trong React hooks.
- Sử dụng cho side effects.
- Mỗi hook gồm 2 phần: `side effect` và `clean up` (optional)
- Được thực thi sau mỗi lần render.
- Được thực thi ít nhất một lần sau lần render đầu tiên.
- Những lần render sau, chỉ được thực thi nếu có dependencies thay đổi.
- Effect cleanup sẽ được thực thi trước run effect lần tiếp theo hoặc unmount.



```
// callback: Your side effect function
// dependencies: Only execute callback if one of your dependencies
function useEffect(callback, dependencies) {}
```

Chạy 3 rồi 1, rồi 3 rồi 2 rồi 1, rồi cứ 3 2 1

```
function App() {
  // executed before each render
  const [color, setColor] = useState('deeppink');

  // executed after each render
  useEffect(() => {
    // do your side effect here ... 1

    return () => {
      // Clean up here ...
      // Executes 2 the next render or unmount
    };
  }, []);

  // rendering
  return <h1>Easy Frontend</h1>; 3
}
```

Lần đầu tiên thì
trong return sẽ
không chạy



MOUNTING

- rendering
- run `useEffect()`

UPDATING

- rendering
- run `useEffect() cleanup` nếu dependencies thay đổi.
- run `useEffect()` nếu dependencies thay đổi.

UNMOUNTING

- run `useEffect() cleanup`.



Luôn luôn đc chạy sau mỗi lần render

```
useEffect(() => {  
  // EVERY  
  // No dependencies defined  
  // Always execute after every render  
  
  return () => {  
    // Execute before the next effect or unmount.  
  };  
});
```



Chạy đúng 1 lần sau lần render đầu tiên

```
useEffect(() => {  
  // ONCE  
  // Empty dependencies  
  // Only execute once after the FIRST RENDER  
  
  return () => {  
    // Execute once when unmount  
  };  
}, []);
```



State filters thay đổi thì mới thực hiện

```
useEffect(() => {  
  // On demand  
  // Has dependencies  
  // Only execute after the first RENDER or filters state changes  
  
  return () => {  
    // Execute before the next effect or unmount.  
  };  
}, [filters]);  
}
```




```
useEffect(() => {  
  async function fetchData() {  
    try {  
      // TODO: Should split into a separated api file instead of using  
      fetch directly  
      const queryParamsString = queryString.stringify();  
      const requestUrl = `http://js-post-api.herokuapp.com/api/posts?  
_limit=10`;  
      const response = await fetch(requestUrl);  
      const responseJSON = await response.json();  
      const { data, pagination } = responseJSON;  
  
      console.log({ data, pagination });  
      setPostList(data);  
    } catch (error) {  
      console.log('Failed to fetch posts: ', error.message);  
    }  
  }  
  
  fetchData();  
}, []);
```



5. Dùng `useEffect()` có cleanup

```
function Clock() {  
  const [timeString, setTimeString] = useState(null);  
  const intervalRef = useRef(null);  
  
  useEffect(() => {  
    intervalRef.current = setInterval(() => {  
      const now = new Date();  
      const hours = `0${now.getHours()}`.slice(-2);  
      const minutes = `0${now.getMinutes()}`.slice(-2);  
      const seconds = `0${now.getSeconds()}`.slice(-2);  
      const currentTimeString = `${hours}:${minutes}:${seconds}`;  
  
      setTimeString(currentTimeString);  
    }, 1000);  
  
    return () => {  
      clearInterval(intervalRef.current);  
    };  
  }, []);  
}
```



```
class App extends PureComponent {  
  componentDidMount() {  
    console.log('Component Did Mount');  
  }  
  
  componentWillUnmount() {  
    console.log('Component Will Unmount');  
  }  
}
```

viết lại tương đương với hooks

```
function App() {  
  useEffect(() => {  
    console.log('Component Did Mount');  
  
    return () => {  
      console.log('Component Will Unmount');  
    };  
  }, []);  
}
```



```
class App extends PureComponent {  
  componentDidMount() {  
    console.log('Component Did Mount or Did Update');  
  }  
  
  componentDidUpdate() {  
    console.log('Component Did Mount or Did Update');  
  }  
}
```

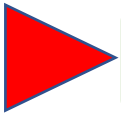
viết lại tương đương với hooks

```
function App() {  
  useEffect(() => {  
    console.log('Component Did Mount or Did Update');  
  });  
}
```



7. Những lưu ý cần nhớ

- Side effect là gì? Có bao nhiêu loại?
- Có thể kèm điều kiện để thực thi `useEffect()`
- Có thể dùng nhiều `useEffect()`
- Tư duy về side effects khi dùng `useEffect()` hook, thay vì life cycle.



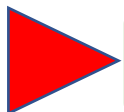
Link

Lấy data

+ có nhiều effect thì nó chạy từ effect trên xuống dưới

```
useEffect(()=>{
  async function fetchAPI(){
    try {
      const resquesURL = "http://js-post-api.herokuapp.com/api/posts?_limit=1&_page=1";
      const response= await fetch(resquesURL);
      const responseJSON = await response.json();
      console.log({responseJSON})

      const {data} = responseJSON;
      // setState nữa là ok
    } catch (error) {
      console.log("Lỗi rồi");
    }
  }
  fetchAPI();
},[]);
```

useRef

+ giá trị của nó sẽ không thay đổi, như biến toàn cục vậy, nhưng nó không gây ra re-render
+ Nếu thời gian gõ giữa các chữ dưới 300 ms thì không thực hiện submit
+ Nếu chạy đc 100ms mà gõ tiếp thì phải reset lại timeout đi

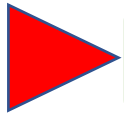
```
function PostFiltersForm(props) {
  const { onSubmit } = props;
  const [searchTerm, setSearchTerm] = useState('');
  const typingTimeoutRef = useRef(null);

  function handleSearchTermChange(e) {
    const value = e.target.value;
    setSearchTerm(value);

    if (!onSubmit) return;

    // SET -- 100 -- CLEAR, SET -- 300 --> SUBMIT
    // SET -- 300 --> SUBMIT
    if (typingTimeoutRef.current) {
      clearTimeout(typingTimeoutRef.current);
    };

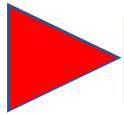
    typingTimeoutRef.current = setTimeout(() => {
      const formValues = {
        searchTerm: value,
      };
      onSubmit(formValues);
    }, 300);
  }
}
```



useRef

+ giúp truy cập đến
DOM

```
function App() {  
  const ref = useRef(null);  
  
  console.log({ ref });  
  useEffect(() => {  
    ref.current.focus();  
  }, []);  
  
  return (  
    <>  
    <input type='text' ref={ref} />  
    <button>CLICK BUTTON</button>  
    </>  
  );  
}  
  
export default App;
```



setInterval

Chỉ cần khởi tạo 1 lần thì cứ bao nhiêu giây nó sẽ lặp lại cái hàm đó

```
useEffect(() => {  
  setInterval(() => {  
    countRef.current = countRef.current + 1;  
    console.log({ countRef });  
  }, 1000);  
}, []);
```



1. useCallback() là gì?

Là một react hooks giúp mình tạo ra một **memoized callback** và chỉ tạo ra callback mới khi **dependencies thay đổi**.

- Nhận vào 2 tham số: 1 là **function**, 2 là **dependencies**.
- Return **memoized callback**
- Chỉ tạo ra function mới khi dependencies thay đổi.
- Nếu dùng empty dependencies thì không bao giờ tạo ra function mới.

```
// Mỗi lần App re-render  
// --> tạo ra một function mới  
// --> Chart bị re-render  
function App() {  
  const handleChartTypeChange = (type) => {}  
  return <Chart onChange={handleChartTypeChange} />;  
}
```



```
// Mỗi lần App re-render  
// --> tạo ra một function mới  
// --> Chart bị re-render  
function App() {  
  const handleChartTypeChange = (type) => {}  
  return <Chart onTypeChange={handleChartTypeChange} />;  
}
```

```
// Mỗi lần App re-render  
// --> nhờ có useCallback() nó chỉ tạo function một lần đầu  
// --> Nên Chart ko bị re-render.  
function App() {  
  const handleChartTypeChange = useCallback((type) => {}, [])  
  return <Chart onTypeChange={handleChartTypeChange} />;  
}
```




```
// Mỗi lần App re-render  
// --> tạo ra một mảng mới  
// --> Chart bị re-render do props thay đổi  
function App() {  
  const data = [{}, {}, {}];  
  return <Chart data={data} />;  
}
```

```
// Mỗi lần App re-render  
// --> nhờ có useMemo() nó chỉ tạo ra cái mảng 1 lần đầu  
// --> Nên Chart ko bị re-render.  
function App() {  
  const data = useMemo(() => [{}, {}, {}], [])  
  return <Chart data={data} />;  
}
```




3. So sánh useCallback() vs useMemo()

GIỐNG NHAU

- Đều áp dụng kĩ thuật memoization.
- Đều nhận vào 2 tham số: function và dependencies.
- Đều là react hooks, dùng cho functional component.
- Dùng để hạn chế những lần re-render dư thừa (micro improvements).

KHÁC NHAU

#	useCallback()	useMemo()
return	memoized callback	memoized value
code	useCallback((type) => {}, [])	useMemo(() => [{}, {}, {}], [])



```
# Install axios package  
npm install --save axios
```

```
try {  
  const url = 'https://js-post-api.herokuapp.com/api/products?  
_limit=10&_page=1';  
  const response = await axios.get(url);  
  console.log(response);  
} catch (error) {  
  console.log('Failed to fetch products: ', error);  
}
```



```
Con1.jsx U JS App.js U JS Store.js U X
bai17redux > src > Store > JS Store.js > VanTho
1  var redux = require('redux');
2  const VanThoInitialState = {
3    state1:true,
4    state2:"Nguyen Van Tho"
5  }
6  const VanTho = (state = VanThoInitialState, action) => {
7    switch (action.type) {
8      case "HanhDong1":
9        const VanThoState1 = {...state, state1: !state.state1}
10       console.log(action.truyenvStore);
11       return VanThoState1
12     case "HanhDong2":
13       const VanThoState2 = {...state, state2: "Đã thay đổi"}
14       return VanThoState2
15     case "HanhDong3":
16       return state
17     default:
18       return state
19   }
20 }
21
22 var myStore = redux.createStore(VanTho);
23 myStore.subscribe(function(){
24   console.log(JSON.stringify(myStore.getState()));
25 });
```



```
Con1.jsx U X JS App.js U JS Store.js U
bai17redux > src > Components > Con1.jsx > Con1 > handleCallFunctionSentData
1 import React from 'react';
2 import { useDispatch, useSelector } from "react-redux"
3
4 function Con1(props) {
5   const state2 = useSelector(state => state.state2)
6   const dispatch = useDispatch();
7
8   const handleCallFunction= ()=>{
9     dispatch({type:"HanhDong2"});
10  }
11  const handleCallFunctionSentData = (data) =>
12  {
13    const truyenveStore = data;
14    dispatch({type:"HanhDong1", truyenveStore})
15  }
16  return (
17    <div>
18      {state2}
19      <button onClick={handleCallFunction}>Thực hiện gọi hàm</button>
20      <button onClick={()=>handleCallFunctionSentData("Tham số truyền tới")}>Thực hiện gọi hàm truyền t
21    </div>
22  );
23 }
24 export default (Con1);
```



- + useSelector : truyền state từ store về component**
- + useDispatch : truyền hàm từ store về component**

Lấy ảnh : <https://picsum.photos/>