Những điều cần biết về useEffect() hook

Presented by: Hau Nguyen | Easy Frontend

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 changes
function useEffect(callback, dependencies) {}
```

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

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

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

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

```
MOUNTING
- rendering
- run useEffect()

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

UNMOUNTING
- run `useEffect() cleanup`.
```

3. Dùng `useEffect()` kèm điều kiện

```
function App() {
  const [filters, setFilters] = useState();

  useEffect(() => {
    // EVERY
    // No dependencies defined
    // Always execute after every render

    return () => {
      // Execute before the next effect or unmount.
    };
  });


  useEffect(() => {
    // ONCE
    // Empty dependencies
    // Only execute once after the FIRST RENDER

    return () => {
      // Execute once when unmount
    };
  }, []);

  useEffect(() => {
    // On demand
    // Has dependencies
    // Only execute after the first RENDER or filters state changes

    return () => {
      // Execute before the next effect or unmount.
    };
  }, [filters]);
}
```

4. Dùng `useEffect()` không có cleanup

```
function App() {
  const [postList, setPostList] = useState([]);

  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();
  }, []);

  return <div>Post list length: {postList.length}</div>;
}
```

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);
    };
  }, []);

  return (
    <div style={{ fontSize: '48px' }}>{timeString}</div>
  );
}
```

6. Chuyển từ life cycles sang useEffect() hook

```
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.

8. Next step

- Các bài code ví dụ sử dụng useEffect() ❤️

🌐 Link tham khảo

- Introduction to react hooks: https://reactjs.org/docs/hooks-intro.html
- React hooks API reference: https://reactjs.org/docs/hooks-reference.html
- React hooks FAQ: https://reactjs.org/docs/hooks-faq.html
- Using useEffect() hook: https://reactjs.org/docs/hooks-effect.html