

REACTJS

Hà Nội, ngày 29 tháng 12 năm 2023



I. Giới thiệu về DOM và Virtual DOM

I.1. Giới thiệu về DOM

DOM (Document Object Model) là một biểu diễn phân cấp của cấu trúc của một trang web hoặc tài liệu XML. Nó cung cấp một cách để truy cập và tương tác với cấu trúc này thông qua mã lập trình, giúp thay đổi nội dung, kiểu dáng và cấu trúc của trang web một cách động.

- HTML elements như là objects
- properties của tất cả HTML elements
- methods để truy cập đến tất cả HTML elements
- events cho tất cả HTML elements

I. Giới thiệu về DOM và Virtual DOM

I.2. DOM Tree

- document node
- element node
- text node

```
<html> document node
- <head></head> element node
- <body>
  - <div>
    - <p>
      Xin chào text node
    </p>
  </div>
</body>
- </html>
```

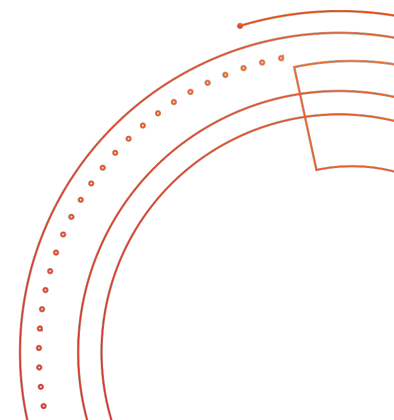


I. Giới thiệu về DOM và Virtual DOM

I.3. Cách truy xuất

Trực tiếp

- `document.getElementById('id_cần_tìm')`
- `document.getElementsByTagName('div')`
- `document.getElementsByName('tên_cần_tìm')`
- `document.querySelector('#id p.class')`
- `document.querySelectorAll('p.class')`



I. Giới thiệu về DOM và Virtual DOM

I.3. Cách truy xuất


Gián tiếp

- `Node.parentNode`
- `Node.childNodes`
- `Node.firstChild`
- `Node.lastChild`
- `Node.nextSibling`
- `Node.previousSibling`



I. Giới thiệu về DOM và Virtual DOM

I.4. Giới thiệu về Virtual DOM

- Virtual DOM là một bản sao ảo của DOM thực tế, được lưu trữ trong bộ nhớ của máy tính.
 - Virtual DOM giúp giảm độ phức tạp của thao tác DOM bằng cách thực hiện các thay đổi trên bản sao ảo trước, sau đó so sánh với DOM thực tế và chỉ cập nhật những phần cần thiết.
- 

I. Giới thiệu về DOM và Virtual DOM

I.4. Giới thiệu về Virtual DOM

DOM- DOM tree

```
<html>  document node
- <head></head> element node
- <body>
  - <div>
    - <p>
      Xin chào text node
    </p>
  </div>
</body>
- </html>
```

Virtual DOM tree

```
- Element("html")
  - Element("head")
  - Element("body")
    - Element("div")
      - Element("p")
      - Element("span")
```



I. Giới thiệu về DOM và Virtual DOM

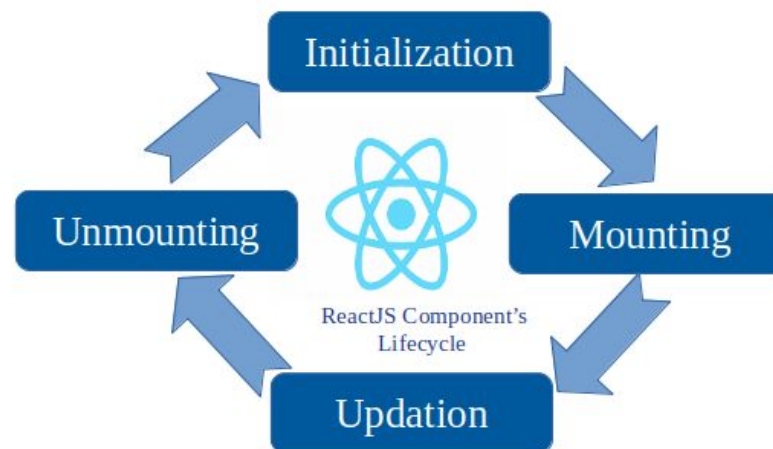
I.5. So sánh

Đặc Điểm	DOM	Virtual DOM (React)
Đặc Điểm Chung		
Cấu Trúc	Cây phân cấp thực tế của DOM	Cây phân cấp ảo được lưu trữ trong bộ nhớ
Truy Cập và Tương Tác	JavaScript, HTML, CSS	JavaScript và React API
Đặc Điểm Khác Nhau		
Tốc Độ Thao Tác	Chậm do truy cập trực tiếp	Nhanh do tương tác ảo và cập nhật linh hoạt
Cập Nhật	Gây tác động lớn đến hiệu suất khi cập nhật trực tiếp	Minimize tác động đến DOM, tối ưu hóa cập nhật
Đặc Tính	Thực tế, được hiển thị trên trình duyệt	Ảo, chỉ tồn tại trong bộ nhớ
Thao Tác Tương Tác	Đòi hỏi cập nhật và vẽ lại mỗi khi có thay đổi	So sánh và chỉ cập nhật phần thay đổi, tối ưu hóa
Thách Thức	Overhead khi thao tác trực tiếp lên DOM	Duy trì cấu trúc ảo có thể tốn kém

II. Lifecycle

II.1. Giới thiệu

Trong ReactJS, "Lifecycle" (vòng đời) là chuỗi các phương thức mà một component đi qua trong quá trình rendering, updating, và unmounting. Lifecycle giúp bạn thực hiện các hành động cụ thể tại các điểm khác nhau trong quá trình sống của một component.





II. Lifecycle

II.2. Lifecycle Trong Class Component

- **Mounting Phase (Quá Trình Mounting):**
 - **constructor():** Khởi tạo state và bind các phương thức.
 - **static getDerivedStateFromProps(props, state):** Được gọi trước khi một component được render. Trả về state mới nếu cần.
 - **render():** Trả về phần tử React để hiển thị.
 - **componentDidMount():** Được gọi sau khi component đã được hiển thị trên DOM.

II. Lifecycle

- **Updating Phase (Quá Trình Updating):**
 - **static `getDerivedStateFromProps(props, state)`:** Gọi lại khi props hoặc state thay đổi. Trả về state mới nếu cần.
 - **`shouldComponentUpdate(nextProps, nextState)`:** Xác định xem component có cần re-render hay không.
 - **`render()`:** Re-render component.
 - **`getSnapshotBeforeUpdate(prevProps, prevState)`:** Trả về một giá trị để chuyển cho `componentDidUpdate()`.
 - **`componentDidUpdate(prevProps, prevState, snapshot)`:** Được gọi sau khi component đã được re-render.
- **Unmounting Phase (Quá Trình Unmounting):**
 - **`componentWillUnmount()`:** Được gọi trước khi component bị unmounted và bị xóa khỏi DOM.



II. Lifecycle

```
componentDidMount() {  
    // Thực hiện các tác vụ sau khi component được tạo ra  
    // chẳng hạn như kết nối với API hoặc tải dữ liệu  
}  
  
componentWillReceiveProps(nextProps) {  
    // Xử lý các thay đổi props  
}  
  
shouldComponentUpdate(nextProps, nextState) {  
    // Quyết định xem component có cần được render lại hay không  
}  
  
componentWillUpdate(nextProps, nextState) {  
    // Thực hiện các tác vụ trước khi component được render lại  
}  
  
componentDidUpdate(prevProps, prevState) {  
    // Thực hiện các tác vụ sau khi component được render lại  
}  
  
componentWillUnmount() {  
    // Thực hiện các tác vụ trước khi component bị hủy bỏ  
    // chẳng hạn như hủy kết nối với API  
}
```

II. Lifecycle

II.2. Lifecycle Trong function Component

- **Mounting và Updating Phases (Quá Trình Mounting và Updating):**
 - **useEffect(() => {}, []):** Tương đương với cả mounting và updating phases. Thay thế cả componentDidMount và componentDidUpdate. Sử dụng dependencies array để kiểm soát kết quả của useEffect.
 - **useLayoutEffect(() => {}, []):** Tương tự như useEffect, nhưng chạy trước khi browser paint.
- **Unmounting Phase (Quá Trình Unmounting):**
 - **useEffect(() => { return () => {} }, []):** Tương đương với componentWillUnmount.

II. Lifecycle

```
const [data, setData] = useState("");

useEffect(() => {
  // componentDidMount và componentDidUpdate
  // ...

  return () => {
    // componentWillUnmount
    // ...
  };
}, [/* dependencies */]);

useLayoutEffect(() => {
  // useEffect trước khi browser paint
  // ...
}, [/* dependencies */]);
```

THANK YOU

FOR
WATCHING