[사용자] 가상 DOM과 Life cycle(생명주기)

구현 어려움 정도	개념
구현 완료	☑
기간	@December 14, 2024
정리 완료	☑

가상 DOM과 Life cycle의 과정을 통해 리액트가 어떻게 동작하는지, DOM이 어떻게 우리가 보는 화면을 조작하는지 알 수 있기 때문에 이는 매우 중요한 개념입니다.

웹 페이지에 눈에 보이다가 사라질 때 까지의 한 과정을 생명주기(Life cycle)라고 합니다.

Life cycle에 대해 알아보기 전, DOM과 가상 DOM에 대해 먼저 알아보겠습니다.

DOM

- Document Object Model
- DOM은 HTML 단위(ex: div) 하나 하나를 객체로 생각하는 모델로 텍스트 노트, 자식 노드 등 하위의 어떤 값을 가지고 있는 트리구조 형식입니다.
- 즉 DOM은 HTML 코드를 해석해서 요소들을 트리 형태로 구조화해 표현하는 데이터입니다.

DOM의 단점

- DOM 트리 중 하나가 업데이트 되면 DOM은 트리구조기 때문에 새로운 요청이나 변경 사항이 발생할 때마다 매번 리렌더링을 해 속도가 느려지고 성능 이슈가 발생한다는 단점이 있습니다.
- 예를 들어 토글 버튼을 클릭하면 단순히 버튼 1개를 클릭 했을 뿐이지만

▶ 토글 버튼입니다

1. 실시간으로 DOM을 뒤져서 찾고
2. 수정되거나 변화가 일어난 부분을 찾아 list들이 생겼다가 없어졌다하는 연산을 컴포넌트 3개가 모두 합니다.

이렇게되면 너무 많은 연산이 발생합니다.

▼ 토글 버튼입니다
1번 list
2번 list
3번 list

이러한 DOM의 단점 때문에 생겨나게 된 것이 가상 DOM입니다.

가상 DOM

- Virtual DOM
- 진짜 DOM이 있으면 똑같은 가상의 DOM을 만듭니다.
- 이 가상 DOM은 눈에 보이지 않으며 메모리 상에서만 존재합니다. (눈에 보이는 것이 진짜 DOM)
- state, props 등이 변경되어 값을 갱신 했을 때, 값을 가상 DOM에만 전부 올려놓습니다.
- 그리고 마지막에 진짜 DOM과 가상 DOM을 비교하여 바뀐 부분만 진짜 DOM에 반영합니다.

이것이 React에서의 가상 DOM의 개념입니다.

정리하면,

1. 진짜 DOM은 처음 페이지에 진입 했을 때 한 번 화면을 그려줌
2. 데이터가 변했을 때 가상 DOM에 변한 부분이 올라감
3. 그 후 진짜 DOM과 가상 DOM을 비교해 변경된 부분만 진짜 DOM에 반영

이렇게 가상 DOM을 사용하면 DOM을 매번 업데이트 하는 것 보다 연산이 적기 때문에, 빠르고 간결하게 DOM을 업데이트 할 수 있습니다.

그렇다면 DOM은 느릴까요?

반은 맞고 반은 틀립니다.

가상 DOM을 사용하면 실제로 DOM을 건드리는 횟수는 줄어듭니다.

가상 DOM은 계속 뭔가를 업데이트 해줄 것이고, 반영하기 위한 뭔가를 만들어야 하기 때문에 연산이 들어가게 됩니다.

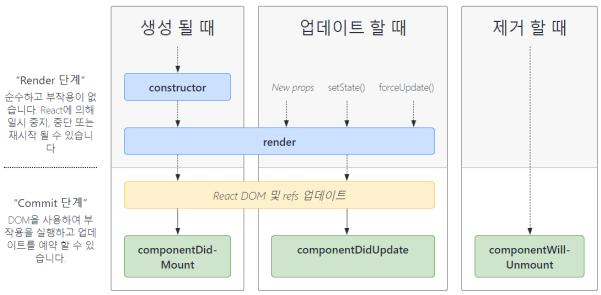
따라서 사이트 구조에 따라 다릅니다.

예를 들어, 위의 예시인 토글 같은 경우도 토글 리스트를 생기고 사라지게 할 때 리스트를 개별로 생기게하고 사라지는 것이 아니라 div 하나로 감싸서 div를 없앴다 생기게 하면 연산이 많이 들지 않습니다.

따라서 때로는 기존 DOM에서 변경 사항을 바로 반영해주는 것이 더 빠를 수 있습니다.

생명주기(Life Cycle)

화면에 컴포넌트가 나타났다가 사라지기 까지의 과정을 생명주기라 합니다.



컴포넌트의 생명주기는 크게 3단계로 구분할 수 있습니다.

1. 생성 될 때
2. 업데이트 할 때
3. 제거 할 때

1. 생성 될 때

실제 DOM에 얹혀지는 것으로 마운트(mount)라고 합니다.

- `constructor`는 생성자 함수(class에 나오는 개념)로 컴포넌트를 생성할 때 사용하는 함수로, 생성할 때 무언가를 초기화 해줄 때 컴포넌트에 필요한 값을 넣어줄 때 사용
- `render`: 가상 DOM에서 바뀐 부분이 DOM으로 올라가는 것
- 이 render가 완료되면 "mount되었다"라고 한다. 이것을 componentDid-Mount라 함

2. 업데이트 할 때

사용자의 클릭, 어떤 동작의 발생(이벤트) 등 컴포넌트의 내용이 바뀌는 것을 업데이트라 합니다.

- `New props` 부모 컴포넌트가 자식한테 주는 데이터가 변경 됐을 때,
- `setState()` 부모 컴포넌트가 가진 데이터를 바꿔줄 때
- `forceUpdate()` 강제로 업데이트가 일어날 때
- `rerendering` 부모 컴포넌트가 업데이트 되었을 때 발생
    - rerendering**했을 때 업데이트 되는 이유**
        - DOM은 트리 구조
        - 트리 구조상 부모 컴포넌트가 바뀌면 자식 컴포넌트도 영향을 받아 바뀌기 때문

어떤 경우에 의해 컴포넌트가 수정 됐을 때 업데이트가 일어나는데

- 이 때, 가상 DOM에서 변경된 부분을 DOM에 옮겨 갈아 끼웁니다.(Render)
- 그리고 나서 커포넌트가 업데이트 되는데 이것을 componentDidUpdate라 합니다.
- 업데이트하면 리렌더링 됩니다.

3. 제거 할 때

화면에서 컴포넌트가 DOM에서 아예 빠져버리는 것으로 **unMount라 합니다.**

컴포넌트가 제거될 때, 컴포넌트가 사라진다는 메서드가 한 번 실행됩니다.