7월1일 천서연

1. REALDOM VS VIRTUALDOM 각각 설명하시오

DOM의 정의

**DOM**(혹은 **Real DOM**)이란 Document Object Model의 약자로서, 이를 직역하자면 문서 객체 모델이라는 의미이다. 문서 객체란 <html>이나 <body> 같은 html문서의 태그들을 JavaScript가 이용할 수 있는 객체(object)로 만든 것을 가리키며, 문서 객체 모델은 다시 말해 문서 객체를 '인식하는 방식'이라고 해석할 수 있다. 즉, DOM은 넓은 의미로 웹 브라우저가 HTML 페이지를 인식하는 방식을 의미한다. 조금 좁은 의미로 본다면 document 객체와 관련된 객체의 집합을 의미할 수도 있다[[1]](#footnote-1). 이는 HTML,JS,CSS로 생성되는 것들을 말하며, 그 예시로는 우리가 자주 보는 <div>, <input>, <a> 등이 있다.[[2]](#footnote-2).

DOM의 자료구조

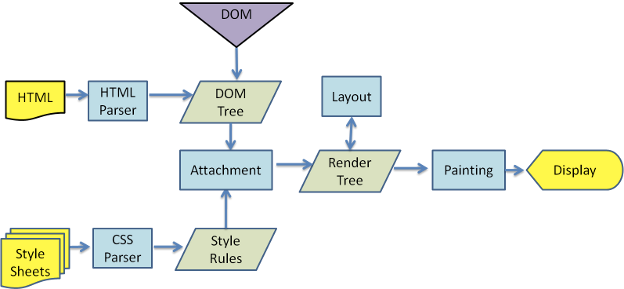
DOM은 tree 형식의 자료구조를 가지고 있는데, 하나의 root node에서 시작되어 아래로 퍼져나가는 형태로 그려진다. 여기서 **node**란, tree 구조에서 root 노드를 포함한 모든 개개의 개체를 의미하며, head, body, title, script, h1, HEADER-1 등의 태그뿐 아니라 태그 안의 텍스트나 속성 등도 모두 node에 속한다[[3]](#footnote-3).

문서 객체가 생성되는 방식

문서 객체가 생성되는 방식은 두 가지로 나누어 볼 수 있는데, 우선 웹 브라우저가 HTML 페이지에 적혀 있는 태그를 읽으면 생성하는 정적인 방식이 있다. 반대로 원래 HTML 페이지에 없던 문서객체를 JavaScript를 이용해동적으로 생성할 수도 있다[[4]](#footnote-4).

브라우저의 작동방식

일반적으로 브라우저의 작동방식은 다음과 같은데, 먼저 DOM Tree (HTML 관련)와 Render Tree (CSS와 같은 스타일 관련)가 생성된다. 그 후 Layout (Reflow) 과정을 거쳐 각 노드들은 스크린의 좌표가 주어지고, 정확히 어디에 나타나야 할 지 위치가 주어진다. 이렇게 트리의 각 노드들을 거쳐 paint() 메소드를 호출하게 되는데, 이를 렌더링 된 요소들에 색을 입히는 Painting 과정이라 부른다. 이러한 과정을 거쳐 스크린에 원하는 정보가 나타나게 되는 것이다 (아래 그림 참조)[[5]](#footnote-5).



즉, DOM에 변화가 생기면 위의 과정과 같이 렌더트리를 재생성하고(그러면 모든 요소들의 스타일이 다시 계산된다), 레이아웃을 만들고 페인팅을 하는 과정이 다시 반복된다. 따라서 DOM 조작이 많이 발생하는 복잡한 SPA(싱글 페이지 어플리케이션) 같은 경우, 그 변화를 적용하기 위해 브라우저가 많이 연산을 해야하고 이는 전체적인 프로세스를 비효율적으로 만든다[[6]](#footnote-6).

Real DOM VS Virtual DOM

Real DOM은 정적인 성격을 띄고 있으며, 이는 DOM의 변화가 빈번하게 일어나는 요즘 트렌드에 맞지 않는 특징이다[[7]](#footnote-7). 따라서 이를 보완하기 위해 DOM의 표현 방법인 Virtual DOM이 등장한다. **Virtual DOM**은 DOM에 변화가 생기면 실제 DOM에 적용되기전에 가상의 DOM 에 먼저 적용시키고, 그 최종적인 결과를 실제 DOM 으로 전달해 준다. 이로써, 브라우저 내에서 발생하는 연산의 양을 줄이면서 성능이 개선되는 것이다[[8]](#footnote-8). Virtual DOM 트리에서 무엇인가 변경되면 새로운 트리를 얻게된다. 알고리즘은 두 트리를 비교하여 변경점을 찾고, Real DOM 변경에 필요한 최소한의 변경만을 수행한다.[[9]](#footnote-9)

2. 버츄얼돔은 왜 좋은가. 어떻게 동작하는가. JQUERY의 돔 직접참조에 비해서 무엇이 개선되었는가.

DOM 조작의 실제 문제는 각 조작이 레이아웃 변화, 트리 변화와 렌더링을 일으킨다는 것이다. 그래서, 예를 들어 30개의 노드를 하나 하나 수정하면, 그 뜻은 30번의 (잠재적인) 레이아웃 재계산과 30번의 (잠재적인) 리렌더링을 초래한다.

Virtual DOM은 변화가 일어나면 이를 오프라인 DOM 트리에 적용시키는데, 이 DOM 트리는 렌더링도 되지 않기 때문에 연산 비용이 적다는 장점이 있다. 연산이 끝나고나면 모든 변화를 하나로 묶어서 그 최종적인 변화를 실제 DOM에 딱 한번만 적용시키게 된다. 비록 레이아웃 계산과 리렌더링의 규모는 커지지만, 하나로 묶어서 적용시킴으로써 연산의 횟수를 줄일 수 있다.

사실 이 과정은 Virtual DOM 없이 이뤄질 수 있는데, 변화가 있을 때 그 변화를 묶어서 DOM fragment 에 적용한 다음에 기존 DOM에 던져주면 해결이 된다. 하지만 Virtual DOM은 그 DOM fragment를 관리하는 과정을 수동으로 하나하나 작업 할 필요 없이, 자동화하고 추상화한다. 그 뿐만 아니라, Virtual DOM은 기존 값 중 어떤게 바뀌었고 어떤게 바뀌지 않았는지 자동으로 계속 파악한다.

또한, DOM 관리를 Virtual DOM 이 하도록 함으로써, 컴포넌트가 DOM 조작 요청을 할 때 다른 컴포넌트들과 상호작용을 하지 않아도 된다. 특정 DOM 을 조작할 것 이라던지, 이미 조작했다던지에 대한 정보를 공유할 필요가 없는 것이다. 즉, 각 변화들의 동기화 작업을 거치지 않으면서도 모든 작업을 하나로 묶어줄 수 있다는 장점이 있다[[10]](#footnote-10).

이와 같이 Virtual DOM 기술은 기존의 비효율적인 DOM 조작 방식을 보완해준다. 기존에는 아래와 같이, jQuery나 Javascript를 통해 원하는 dom 노드를 찾은 후 변경하는 행위를 했다.

document.getElementById('myId').appendChild(myNewNode) // javascript

$('#myId').append(myNewNode) // jquery

하지만 이와 같은 방식은 수천가지의 노드가 존재할 수 있기에, 계산하기 위해 큰 비용을 투자하게 된다.

그 결과 성능 저하로 인한 페이지 속도 지연이 발생할 수 있다. 반면, Virtual DOM을 이용하면 직접 DOM API를 사용하지 않고, domNode 객체를 활용하게 된다. 이러한 처리는 실제 DOM이 아닌 메모리에 있기 때문에 훨씬 더 빠르다.

domNode.children.push('<li>Item 3</li>');

즉, 실제 DOM이 아닌 Virtual DOM에 먼저 변경 작업을 해준다. 그리고 변경된 부분만 찾아 실제 DOM에 변경해주는 방식이다. 이로써, 기존 View 방식보다 많은 양의 연산을 줄일 수 있게 된다[[11]](#footnote-11).

3. 버츄얼돔이 동작하는 DIFF알고리즘에 대해 설명하시오.

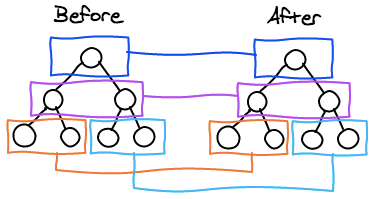
React의 Reconciliation은 변경점을 찾는 과정(**Diffing** 알고리즘)을 통하여 어떤 변경에 대한 전/후 엘리먼트 트리를 비교(Diff) 후 갱신이 필요한 부분만을 찾아 업데이트한다[[12]](#footnote-12). 두 개의 트리를 비교할 때, React는 두 엘리먼트의 루트(root) 엘리먼트부터 비교한다. 두 루트 엘리먼트의 타입이 다르면 React는 이전 트리를 버리고 완전히 새로운 트리를 구축한다. 반면, 같은 타입의 두 DOM 엘리먼트를 비교할 경우, React는 두 엘리먼트의 속성을 확인하여 동일한 내역은 유지하고 변경된 속성들만 갱신하게 된다[[13]](#footnote-13).

React는 렌더링에서 Reconciliation 작업을 선행하기 때문에 플랫폼 UI에 대한 제어를 최소화 시킬 수 있다. 즉 브라우저에서 DOM에 대한 제어를 최소화시키는 것이다. 기존의 Diff 알고리즘은 O(n^3)의 시간복잡도를 가지고 있다. 그래서 React는 다음 두 가지 가정을 가지는 휴리스틱 알고리즘으로 O(n)에 근사할 수 있도록 구현하였다.

1. 다른 타입의 두 엘리먼트는 다른 트리를 만들 것이다.

2. 각 렌더링에서 유지되는 엘리먼트에 key 프로퍼티를 통해 같은 엘리먼트라는 것을 알린다. (같은 레벨에서만 유효하다.)

Diff 방식은 트리를 비교할 때 기본적으로 서브트리들의 위치(level-by-level)를 기준으로 비교한다.



같은 위치에서 엘리먼트의 타입이 다른 경우,

1. 기존 트리를 제거 후 새로운 트리 만든다.

2. 기존 트리 제거시 트리 내부의 엘리먼트/컴포넌트들은 모두 제거한다.

3. 새로운 트리를 만들 때 내부 엘리먼트/컴포넌트들도 모두 새로 만든다.

같은 위치에서 엘리먼트가 DOM을 표현하고 그 타입이 같은 경우,

1. 엘리먼트의 attributes를 비교한다.

2. 변경된 attributes만 업데이트한다.

3. 자식 엘리먼트들에 diff 알고리즘을 재귀적으로 적용한다.

같은 위치에서 엘리먼트가 컴포넌트를 표현하고 그 타입이 같은 경우,

1. 컴포넌트 인스턴스 자체는 변하지 않는다.(때문에 컴포넌트의 state가 유지된다.)

2. 컴포넌트 인스턴스의 업데이트 전 라이프 사이클 메서드들이 호출되며 props가 업데이트된다.

3. render()를 호출하고, 컴포넌트의 이전 엘리먼트 트리와 다음 엘리먼트 트리에 대해 diff 알고리즘을 재귀적으로 적용한다[[14]](#footnote-14).

1. <https://m.blog.naver.com/magnking/220972680805> [↑](#footnote-ref-1)
2. http://blog.drakejin.me/React-VirtualDOM-And-Repaint-Reflow/ [↑](#footnote-ref-2)
3. <https://m.blog.naver.com/magnking/220972680805> [↑](#footnote-ref-3)
4. <https://m.blog.naver.com/magnking/220972680805> [↑](#footnote-ref-4)
5. <https://velopert.com/3236> [↑](#footnote-ref-5)
6. <https://velopert.com/3236> [↑](#footnote-ref-6)
7. http://blog.drakejin.me/React-VirtualDOM-And-Repaint-Reflow/ [↑](#footnote-ref-7)
8. <https://velopert.com/3236> [↑](#footnote-ref-8)
9. <https://wonism.github.io/deep-dive-into-vdom/> [↑](#footnote-ref-9)
10. <https://velopert.com/3236> [↑](#footnote-ref-10)
11. <https://mygumi.tistory.com/190> [↑](#footnote-ref-11)
12. <https://meetup.toast.com/posts/110> [↑](#footnote-ref-12)
13. <https://ko.reactjs.org/docs/reconciliation.html> [↑](#footnote-ref-13)
14. <https://meetup.toast.com/posts/110> [↑](#footnote-ref-14)