7월 1일 기술면접 문제

1. REAL DOM VS VIRTUAL DOM 각각 설명하시오

- Real DOM: 문서 객체 모델(Document Object Model)의 약자로, API를 통해 페이지 콘텐츠와 소통한다. DOM은 개발자가 자바스크립트를 이용해 문서 콘텐츠에 접근하고 조작할 수 있게 한다. DOM API를 사용하면 문서 안의 정보를 읽거나 변경, 삭제하는 등 웹 페이지의 자체 내용을 바꿀 수 있다. DOM의 용도는 두 가지다. 첫째, DOM은 자바스크립트(또는 다른 언어)가 이해할 수 있는 형태로 마크업을 번역함으로써 구조화된 웹 페이지 ‘지도’를 제공한다. DOM이 없다면 자바스크립트는 문서에 대한 어떤 것도 알 수 없다. 자바스크립트가 DOM을 통해 접근할 수 있는 문서 전체를 ‘노드(node)’라고 하며, 문서 안의 개별 요소도 노드라고 한다. 둘째, DOM은 노드에 접근할 수 있는 메서드나 함수를 자바스크립트에 제공한다. 예를 들어 문서 안의 모든 p 태그 목록을 가져온다거나 .collapsible 클래스를 갖는 요소의 자식 요소 중에서 .toggle이라는 클래스를 갖는 모든 요소를 가져오는 등의 작업이 가능하다.

- Virtual DOM: 말그대로 가상의 DOM을 의미한다. Virtual DOM 기술은 DOM 조작에 있어 비효율적인 이유에서 나오게 되었다. 실제 DOM의 변경사항에 대해 DOM에서 수행해야 할 모든 변경 사항을 가상돔(virtual DOM)에서 수행한 다음 실제 DOM에 전달함으로써 위에서 언급한 계산 단계가 줄어든다. 여러 번의 변경사항이 있더라도 모든 변경 사항을 하나로 그룹화하여 한번만 수행한다.

출처:

<웹디자이너를 위한 자바스크립트> - 매트 마키스 저

<https://mygumi.tistory.com/190>

<https://code-masterjung.tistory.com/33>

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

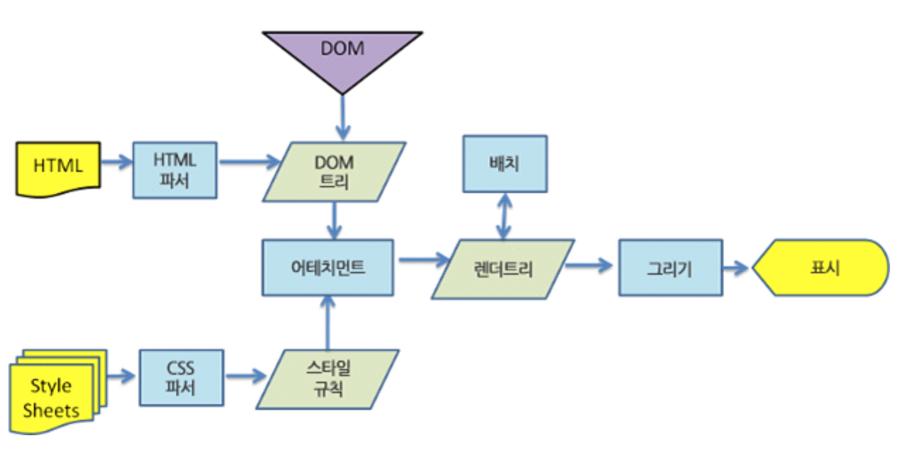
기존에는 아래 코드와 같이, Javascript 또는 Jquery를 통해 원하는 DOM 노드를 찾은 후 변경하는 행위를 했다.

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

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

하지만 수천가지의 노드가 존재할 수 있기에, 계산하기 위해 큰 비용을 투자하게 된다. 그 결과, 성능 저하로 인한 페이지 속도 지연이 발생한다.

간단히 설명하자면, 아래의 그림과 같은 동작이 DOM 조작을 할 때마다 영향을 끼치게 되는 것이다.



이러한 배경 속에서 Virtual DOM 이 탄생하게 되었다. 이러한 Virtual DOM에 대해 코드를 통해 표현할 수 있다.

<ul id='myId'>

<li>Item 1</li>

<li>Item 2</li>

<ul>

위와 같은 HTML 코드가 있는 상태에서 id가 'myId' ul 태그 안에 li 태그 하나를 추가할 것이다. 이 작업을 위해 우리는 기존 방법으로는 javascript, jquery를 통해 노드를 찾은 후 필요한 노드를 삽입했다.

let domNode = {

tag: 'ul'

attributes: { id: 'myId' }

children: [

// where the LI's would go

]

};

domNode 객체를 Virtual DOM으로 보면 된다. 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.

출처: <https://mygumi.tistory.com/190>

3. 버츄얼돔이 동작하는 DIFF알고리즘에 대해 설명하시오.

React의 Reconciliation(두 가지 이상의 생각·요구 등을 조화시키는 과정)은 어떤 변경에 대한 전/후 엘리먼트 트리를 비교(Diff)하여 갱신이 필요한 부분만을 찾아 업데이트하는 것을 의미한다. React는 렌더링에서 Reconciliation 작업을 선행하기 때문에 플랫폼 UI에 대한 제어를 최소화하는 것이다(보통 UI 제어 비용은 비싸기 때문이다). 즉 브라우저에서 DOM에 대한 제어를 최소화하는 것이다.

다시 한번 정리해보면, React 컴포넌트는

1. render()에서 새로운 엘리먼트 트리를 생성하고,

2. 이전 엘리먼트 트리와 비교하여 변경 점을 찾아 업데이트한다.

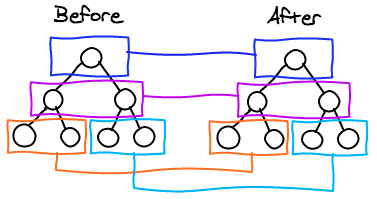
그런데 기존의 Diff 알고리즘은 O(n^3)의 시간복잡도를 가지고 있다. 그래서 React는 다음 두 가지 가정을 가지는 휴리스틱 알고리즘으로 O(n)에 근사할 수 있도록 구현하였다.

다른 타입의 두 엘리먼트는 다른 트리를 만들 것이다. 각 렌더링에서 유지되는 엘리먼트에 key 프로퍼티를 통해 같은 엘리먼트라는 것을 알린다. (같은 레벨에서만 유효하다.)

즉 DIFF알고리즘이란, 사전 업데이트 된 virtual DOM과 업데이트 된 virtual DOM을 비교하는 과정을 diffing이라고 하며, 그렇게 계산 및 비교하는 알고리즘을 DIFF 알고리즘이라고 칭한다.

Level By Level

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



Elements Of Different Types 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.

{/\* Before \*/}

<div>

<Counter /> {/\* Will unmount \*/}

</div>

{/\* After \*/}

<span>

<Counter /> {/\* Will mount, Did mount \*/}

</span>

DOM Elements Of The Same Type 같은 위치에서 엘리먼트가 DOM을 표현하고 그 타입이 같은 경우

1. 엘리먼트의 attributes를 비교한다.

2. 변경된 attributes만 업데이트한다.

3. 자식 엘리먼트들에 diff 알고리즘을 재귀적으로 적용한다.

{/\* Before \*/}

<div className="before" title="stuff" />

{/\* After \*/}

<div className="after" title="stuff" /> {/\* Update className \*/}

Component Elements Of The Same type 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.

{/\* Before \*/}

<Counter value="3" />

{/\* After \*/}

{/\* Will recevie props, Will update, Render --> diff algorithm recurses \*/}

<Counter value="4" />

출처:

<https://meetup.toast.com/posts/110>

<https://code-masterjung.tistory.com/33>