**1. REALDOM VS VIRTUALDOM 각각 설명하시오**

**Real DOM = DOM** (Document Object Model) : HTML문서를 프로그래밍적으로 접근 가능하게 해주는 인터페이스이다.(설계도 정보) HTML 파일이 브라우저에 표현되는 과정(랜더링)을 살펴보면 다음과 같은데,  
1) 웹 문서(HTML파일)를 로드 한다.  
2) 웹 문서를 브라우저가 이해할 수 있는 구조로 구성하여 메모리에 올린다.  
3) 그리고 브라우저는 브라우저가 이해할 수 있는 구조를 그려낸다.  
이때 브라우저의 랜더링 엔진이 모든 요소(Element)와 속성(Attribute)등을 각각의 객체로 만들고 이것을 트리 구조로 구성하는데, 이를 **DOM**이라고 한다.

**DOM의 동작과정**을 살펴보면 다음과 같다.  
1) DOM Tree 생성 - 브라우저가 전달 받은 HTML DOM으로 변환  
2) Render Tree 생성 - css 등 스타일 요소 attach DOM Tree와 같게 Render Tree 생성  
3) Layout - 각 노드들이 어디에 나타야할지 스크린 좌표 계산  
4) Paint - 랜더링 된 요소에 색을 입히는 과정

Element를 추가, 수정, 생성 하면서 DOM이 수정 되면 Render Tree생성, Layout 생성, Paint가 다시 이루어지는데, 이는 리소스가 변경 될 때마다 전체 페이지를 갱신 시킨다고 생각하면 된다. 복잡한 SPA(싱글 페이지 어플리케이션) 에서는 DOM 조작이 많이 발생하는데, 그 뜻은 그 변화를 적용하기 위해 브라우저가 많이 연산을 해야한단 소리고, 전체적인 프로세스를 비효율적으로 만든다는 것이다. 이를 해결하기 위해 생겨난 개념이 Virtual DOM이다.

**Virtual DOM** : 메모리 상에서만 존재하는 가상의 DOM이다. 변화가 일어나면, 실제로 브라우저의 DOM 에 새로운걸 넣는것이 아니라, 자바스크립트로 이뤄진 가상 DOM 에 한번 렌더링을 하고, **기존의 DOM 과 비교**를 한 다음에 **정말 변화가 필요한 곳에만 업데이트를 해주는 것.** 즉, 데이터가 바뀌었을 때 어떻게 업데이트를 할 지를 고려하는 것이 아닌, 일단 바뀐 데이터로 그려놓고 비교를 한 다음, 바뀐 부분만 바꾸는 것이다

출처:

<https://ryublock.tistory.com/41>

<https://velog.io/@denny6389/React-%EA%B8%B0%EC%88%A0-%EB%AC%B8%EC%A0%9C>

<https://hashnode.com/post/the-one-thing-that-no-one-properly-explains-about-react-why-virtual-dom-cisczhfj41bmssp53mvfwmgrq>

**2. 버츄얼돔은 왜 좋은가. 어떻게 동작하는가. JQUERY의 돔직접참조에 비해서 무엇이 개선되었는가.**

Virtual DOM 을 사용하면, 실제 DOM 에 접근하여 조작하는 대신에, 이를 추상화 시킨 스크립트 객체(DOM의 사본)를 구성하여 사용한다. React 에서 데이터가 변하여 브라우저상의 실제 DOM 을 업데이트 할 때에는 다음과 같이 3가지 절차를 밟는다.

1) 데이터가 업데이트되면, 전체 UI 를 Virtual DOM 에 리렌더링 한다.  
2) 이전 Virtual DOM 에 있던 내용과 현재의 내용을 비교한다.  
3) 바뀐 부분만 실제 DOM 에 적용이 된다.

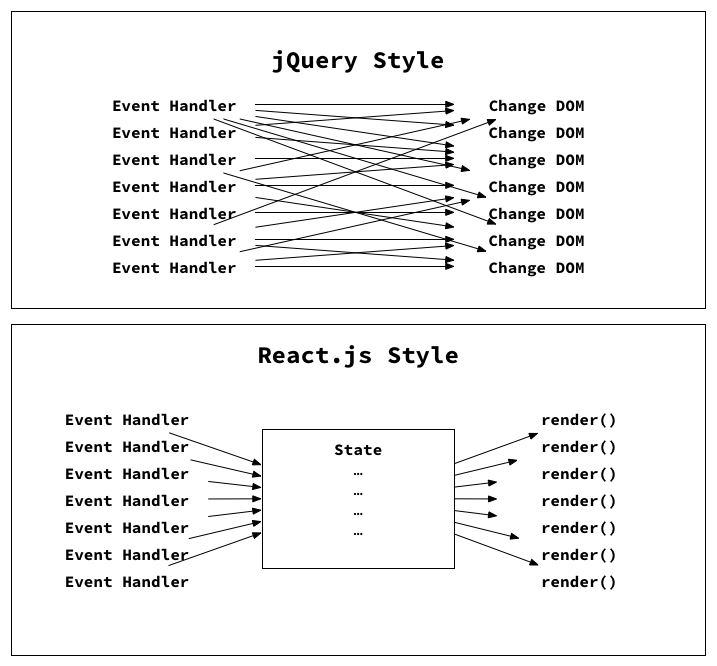
jQuery의 [DOM 조작 메서드](http://api.jquery.com/category/manipulation/)는 신이 내린 선물이었고, 요상하기 그지 없는 DOM을 다루는 고통을 완화하는 데에 도움을 주었지만, 그 밑에 깔린 퍼포먼스 문제를 전혀 해결하지 않았다. 모바일이 대세인 웹 브라우징 환경에 우리가 대응하게 되면서, 느린 UI 상호작용은 빈번해졌고 훨씬 두드러지게 되었다.

우리는 대부분의 경우 단일 페이지 웹 앱(SPA)과 제3자 외부 컨텐츠가 넘쳐나는 페이지들을 만든다. 이 경우, 첫 렌더링 이후에 아주 느린 성능을 보인다. 페이지마다 천개가 넘는 DOM 요소를 조작하고, 페이지에 온갖 HTML 요소를 비동기적으로 불러온 뒤 그것들에 스타일을 적용해야만 한다. 이제 더 이상은 간단한 HTML 문서가 아니게 되어버렸고, 아주 느린 UX가 발생하게 되는 것이다. 모든 자원이 다운로드되고 나면, 단지 DOM 요소를 재배치하거나, 컨텐츠를 업데이트하는 정도의 작업만 수행할 뿐인데, 왜 이렇게 느려지는 것일까? 이 문제의 가장 큰 원인은 바로 [유명 라이브러리들](http://jsperf.com/knockout-js-vs-direct-dom-manipulation/25)의 DOM 조작 메서드가 아주 느리다는 점이다.

가상 DOM 라이브러리들은 거대한 UI 전체를 항상 다시 렌더링하는 것을 원칙으로 둔다. 훨씬 빠르고 유지보수가 용이한 웹 UI를 만든다는 목표를 충족시키기 위하여, 가장 빠른 DOM 조작들만 허용하고, 이러한 DOM 조작마저도 반드시 라이브러리 자체에 의한 명령을 통하여서만 이루어지도록 하는 것이다. 가상 DOM에 대한 이러한 떠오르는 접근 방식은 단순히 DOM 조작을 일괄 처리하는 것을 능가한다. 또한 이는 HTML Templating에 견주어 보아도 대부분의 경우 이를 능가한다. 이 라이브러리들은 대개 선언적인 **객체 추상화**(Declarative Object Abstractions)를 통하여 UI 코드 덩어리들을 다룬다. 이러한 추상물은 보통 컴포넌트라고 불린다.

**객체 추상화** : 객체들의 공통적인 프로퍼티(이름, 값)와 메서드(클래스나 객체에 숙해 있는 함수)를 뽑아내는 작업. 코드상에서 구현(로직) 부분을 제외한 오직 선언 부분만을 설계하는 것을 말한다.

프레임워크에 의하여 별도로 정의된 컴포넌트 객체 덩어리의 전/후 변화를 [비교(diff)](http://en.wikipedia.org/wiki/Diff), 그리고 반드시 필요할 때에만 변화 내용을 DOM에 기록하기. 이런 접근법을 통하여 가상 DOM 접근법은 [성능뿐만](http://jsperf.com/virtual-dom-vs-real-dom) [아니라](http://elm-lang.org/blog/Blazing-Fast-Html.elm) [유지보수성까지](http://jonykrau.se/posts/the-value-of-react) [향상시킨다고](http://jlongster.com/Removing-User-Interface-Complexity,-or-Why-React-is-Awesome) [증명되었다](http://blog.arkency.com/2014/07/why-we-use-react-js-in-our-rails-projects/).



출처 :

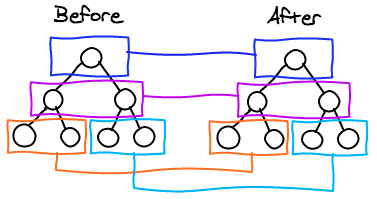
<https://brunch.co.kr/@cadenzah/3>

<http://52.78.22.201/tutorials/react/react-intro-and-give-it-a-try/>

**3. 버츄얼돔이 동작하는 DIFF알고리즘에 대해 설명하시오.**

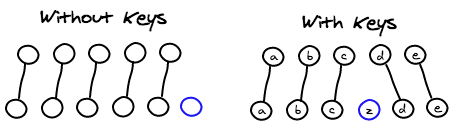
**Level by Level**

두 개의 임의의 트리 사이에 최소 수정 횟수를 찾아내는 건 O(n3) O(n3) 문제이다. 당신이 생각하는대로 이것은 우리의 use case상에서 다루기 쉬운 문제가 아니다. React는 단순하지만 강력한 heuristics를 사용하여 O(n)O(n)에 적절한 근사값을 찾아낸다. React는 트리를 오직 단계별로 받아들인다. 이건 복잡성을 과감하게 낮추며, web application에서는 컴포넌트가 트리의 다른 레벨로 옮겨지는 것이 매우 드문 일이기 때문에 이건 그다지 큰 손해가 아니다. 그것들은 보통은 자신과 같은 레벨 사이에서(자식들 사이에서) 자리를 옮긴다.



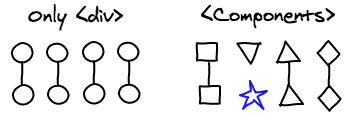
**List**

하나의 iteration에서 5개의 컴포넌트를 render하는 컴포넌트가 있는데, 다음에 새로운 컴포넌트를 이 5개 사이에 끼워 넣는다고 생각해보자. 이 정보만으로는 어떻게 이전의 리스트와 이후의 리스트에 속해있는 컴포넌트를 매핑해야 하는지 알기 힘들다.  
기본적으로 React는 리스트 내부에 있는 컴포넌트를 순서대로 매핑한다. (이전 리스트의 첫번째 컴포넌트는 이후 리스트의 첫번째 컴포넌트, 두번째는 두번째... 이렇게) 우리는 Key attribute를 이용해 React가 어떤 컴포넌트들이 서로 매핑되는지 알게 만들 수 있다. 실제로 이런 방식은 자식들 사이에 있는 unique key를 찾아내기 쉽다.



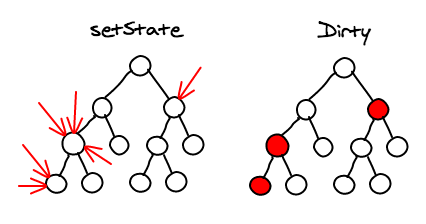
**Components**

React 앱은 보통 여러개의 사용자 정의 컴포넌트로 구성된다. 그리고 이것들은 결과적으로 대부분 div들로 이루어진 트리 구조로 변형된다. React diff algorithm은 이 추가적인 정보를 이용해서 같은 class인 컴포넌트들만 체크 할 것이다.  
예를 들어 <Header>가 <ExampleBlock>로 교체되면, React는 header를 지워버리고, example block을 만든다. 우리는 우리의 귀중한 시간을 전혀 닮은 구석이 없을 법한 두 컴포넌트를 매칭하는데 쓸 필요가 없다.



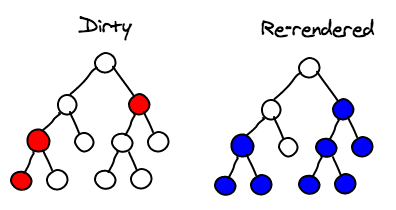
**Batching**

우리가 Component에서 setState를 호출할 때마다, React는 그것에 dirty체크를 한다. 그리고 event loop가 끝났을 때, React는 모든 dirty 체크된 component들을 확인하고 다시 렌더링한다.  
이 일괄 작업은 event loop 동안에는 DOM이 오직 한 번만 update된다는 것을 의미한다. 이 속성으로 앱의 성능을 좋게 만들 수 있으며, 이건 일반적으로 작성한 Javascript를 통해서는 무척 힘든 일이다. React application에서는 그냥 기본으로 사용할 수 있다.



**Sub-tree Rendering**

setState가 호출될 때, component는 자신의 자식을 위해 virtual DOM을 다시 빌드한다. 만약 우리가 setState를 root element에서 호출한다면, 전체 React앱은 다시 rendering될 것이다. 모든 component들은 (심지어 변경이 되지 않았더라도), 자신의 render 메소드를 호출할 것이다. 사실 이건 조금 무섭고 비효율적으로 들리지만 실제로 이건 꽤 잘 작동한다. 왜냐하면 우린 실제 DOM 노드를 조작하는 것이 아니므로. 무엇보다, UI를 어떻게 표시할지 이야기하자. 왜냐하면 화면 공간은 한정되어 있고, 우리는 보통 한 순간에 대략 수백에서 수천의 항목을 화면에 표시하기 때문이다. Javascript는 전체 interface를 관리할 수 있는 충분히 빠른 business logic를 가지고 있다.  
React코드를 작성할 때 중요한 다른 포인트는 뭔가가 바뀌었을 때마다 무조건 root node에서 setState를 호출하지 않는다는 것이다. change event를 받았거나 그 하위의 몇 컴포넌트에서만 setState를 호출한다. 아마 최상위까지 가는 일은 매우 드물 것이다. 이것 사용자가 조작하는 부분에서만 국소적인 변화가 있다는 것을 의미한다.

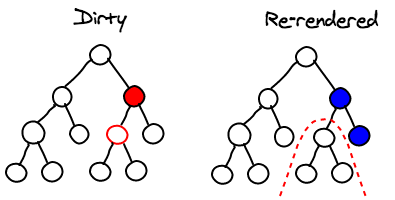


**Selective Sub-tree Rendering**

마지막으로, 몇몇 Sub-tree가 다시 렌더링되지 않았으면 할 때가 있다. 아래 메소드를 component에서 구현한다면 component의 이전과 다음 props/state를 이용해 React에게 해당 component가 변하지 않았는지 혹은 다시 렌더링하지 않아도 되는지 말해줄 수 있다. 만약 이 메소드가 적절하게 구현되었다면 아주 큰 성능 향상을 기대할 수 있다.

boolean **shouldComponentUpdate**(**object** nextProps, **object** nextState)

이걸 사용하기 위해, 우리는 Javascript 객체들을 비교할 수 있어야 한다. 이 비교가 shallow여야 하는가 deep이여야 하는가 같은 종류의 수많은 이슈가 있다. 만약 그게 deep비교여야 한다면 우리는 불변 데이터 구조(immutable data structures)를 사용하거나 deep copy를 해야 한다.  
그리고 우리는 이 function이 항상 호출된다는 것을 기억해야 한다. 그래서 이 function에서 걸리는 시간이 component를 render하는데 걸리는 시간보다 더 적은 시간이 걸린다는 것을 확실하게 해야 한다. re-rendering이 필요하지 않더라도.



출처 : <https://nillk.tistory.com/27>