01. 리액트는 어쩌다 만들어졌을까?

리액트 학습을 본격적으로 하기 전에, 리액트라는 라이브러리가 어쩌다가 만들어졌는지 알면 리액트를 이해하는데 도움이 될 것입니다.

JavaScript 를 사용하여 HTML 로 구성한 UI 를 제어해보셨다면, DOM 을 변형시키기 위하여 우리가 어떤 작업을 해야하는지 익숙 할 것입니다. 브라우저의 DOM Selector API 를 사용해서 특정 DOM 을 선택한뒤, 특정 이벤트가 발생하면 변화를 주도록 설정해야되지요.

한번 HTML/JS 로 만들어진 카운터 예시를 확인해봅시다.


```
<h2 id="number">0</h2>
<div>
  <button id="increase">+1</button>
  <button id="decrease">-1</button>
</div>
```

위와 같이 HTML 이 구성되어 있고, `id` 를 사용하여 각 DOM 을 선택한뒤, 원하는 이벤트가 발생하면 DOM 의 특정 속성을 바꾸어주어야 하죠.

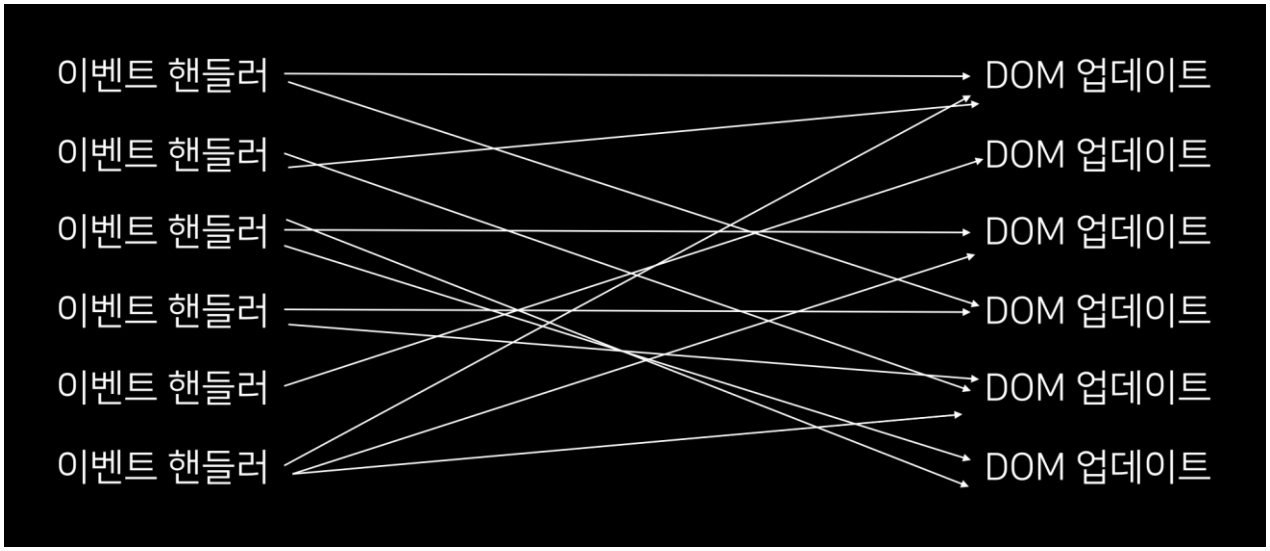
```
const number = document.getElementById('number');
const increase = document.getElementById('increase');
const decrease = document.getElementById('decrease');

increase.onclick = () => {
  const current = parseInt(number.innerText, 10);
  number.innerText = current + 1;
};

decrease.onclick = () => {
  const current = parseInt(number.innerText, 10);
  number.innerText = current - 1;
};
```

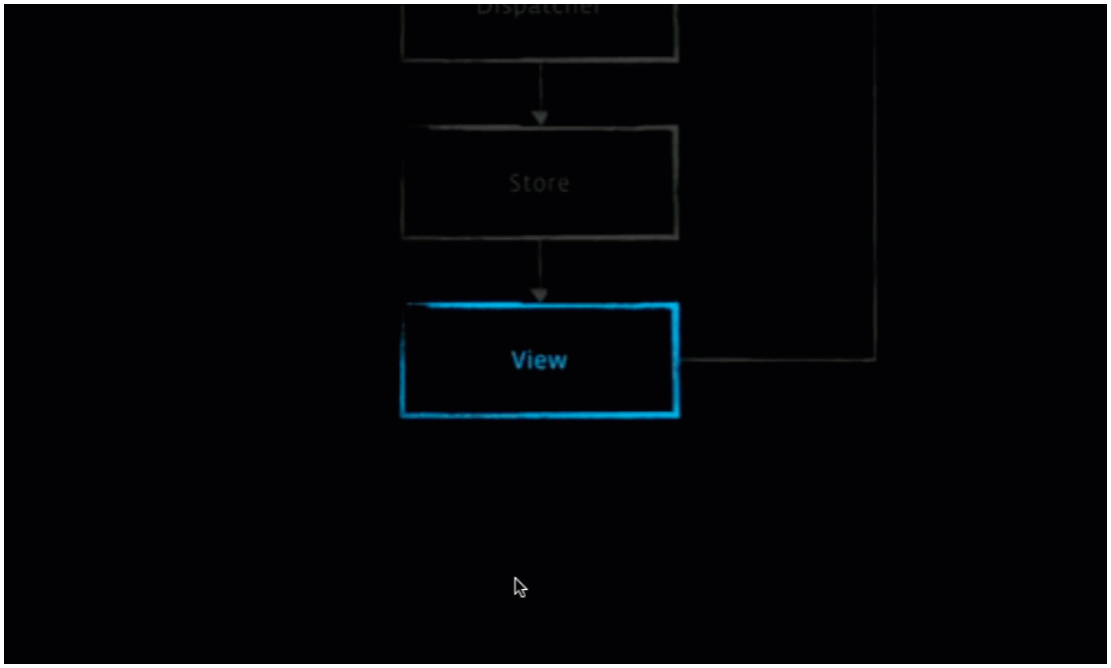
현재 위 코드를 보면 "+1 버튼이 눌리면, `id` 가 `number` 인 DOM 을 선택해서 `innerText` 속성을 1 씩 더해줘라" 라는 규칙이 있지요. 사용자와의 인터랙션이 별로 없는 웹페이지라면 상관없겠지만, 만약에 인터랙션이 자주 발생하고, 이에 따라 동적으로 UI 를 표현해야된다면, 이러한 규칙이 정말 다양해질것이고, 그러면 관리하기도 힘들어질것입니다. 숙련된 JavaScript 개발자라면, 코드를 최대한 깔끔하게 정리하여 쉽게 유지보수를 할 수도 있겠지만, 대부분의 경우 웹 애플리케이션의 규모가 커지면, DOM 을 직접 건드리면서 작업을 하면 코드가 난잡해지기 쉽습니다.

처리해야 할 이벤트도 다양해지고, 관리해야 할 상태값도 다양해지고, DOM 도 다양해지게 된다면, 이에 따라 업데이트를 하는 규칙도 많이 복잡해지기 때문에, 조금 과장을 많이 하자면 코드가 다음과 같은 형태가 됩니다.



그래서, Ember, Backbone, AngularJS 등의 프레임워크가 만들어졌었는데, 이 프레임워크들은 작동방식이 각각 다르지만, 쉽게 설명하자면 자바스크립트의 특정 값이 바뀌면 특정 DOM 의 속성이 바뀌도록 연결을 해주어서, 업데이트 하는 작업을 간소화해주는 방식으로 웹개발의 어려움을 해결해주었습니다.

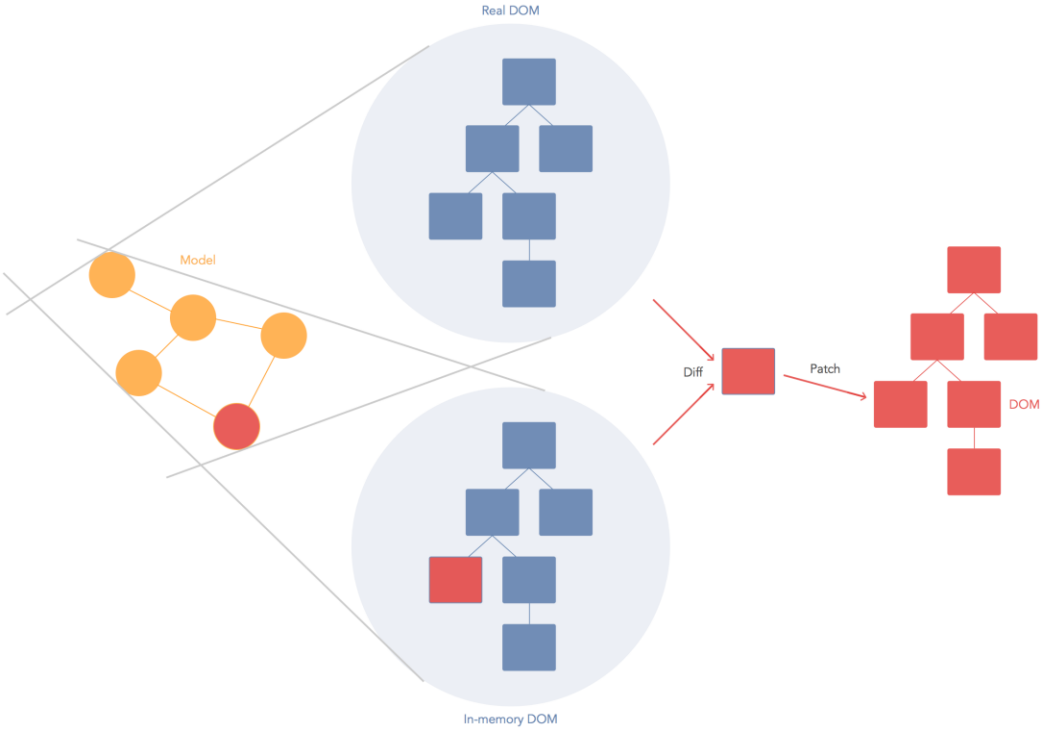
하지만 리액트의 경우에는 조금 다른 발상에서 만들어졌습니다. 리액트는 어떠한 상태가 바뀌었을때, 그 상태에 따라 DOM 을 어떻게 업데이트 할 지 규칙을 정하는 것이 아니라, 아예 다 날려버리고 처음부터 모든걸 새로 만들어서 보여준다면 어떨까? 라는 아이디어에서 개발이 시작되었습니다.



출처: [Introduction to React.js]

그러면 "업데이트를 어떻게 해야 할 지" 에 대한 고민을 전혀 안해도 되기 때문에 개발이 정말 쉬워질 것입니다. 하지만, 정말로 동적인 UI 를 보여주기 위해서 모든걸 다 날려버리고 모든걸 새로 만들게 된다면, 속도가 굉장히 느릴 것입니다. 작은 웹애플리케이션이라면 상관없겠지만 규모가 큰 웹애플리케이션이라면 상상도 할 수 없는 일이죠.

하지만, 리액트에서는 Virtual DOM 이라는 것을 사용해서 이를 가능케 했습니다.



Virtual DOM 은 가상의 DOM 인데요, 브라우저에 실제로 보여지는 DOM 이 아니라 그냥 메모리에 가상으로 존재하는 DOM 으로서 그냥 JavaScript 객체이기 때문에 작동 성능이 실제로 브라우저에서 DOM 을 보여주는 것 보다 속도가 훨씬 빠릅니다. 리액트는 상태가 업데이트 되면, 업데이트가 필요한 곳의 UI 를 Virtual DOM 을 통해서 렌더링합니다. 그리고 나서 리액트 개발팀이 만든 매우 효율적인 비교 알고리즘을 통하여 실제 브라우저에 보여지고 있는 DOM 과 비교를 한 후, 차이가 있는 곳을 감지하여 이를 실제 DOM 에 패치시켜줍니다. 이를 통하여, "업데이트를 어떻게 할 지" 에 대한 고민을 하지 않으면서, 빠른 성능도 지켜낼 수 있게 되었습니다.