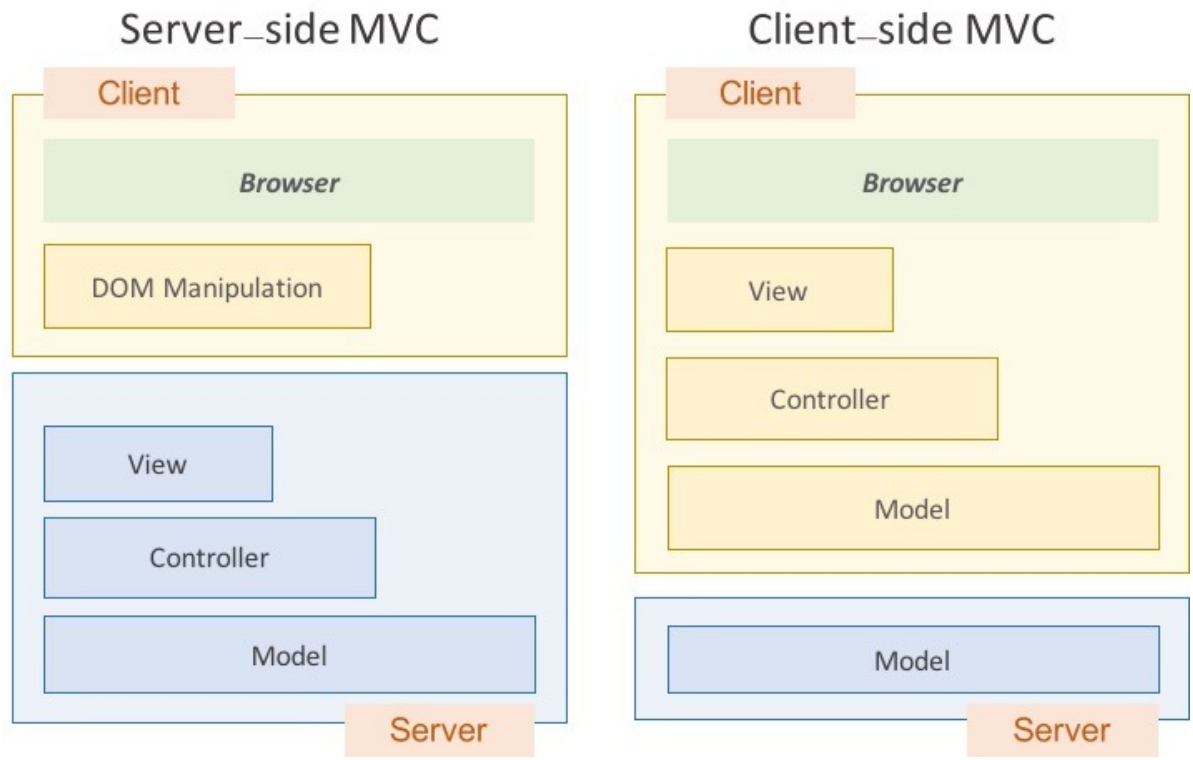
서버 사이드 렌더링 그리고 클라이언트 사이드 렌더링



클라이언트 사이드 렌더링 & 서버 사이드 렌더링

"

렌더링의 현주소

모바일의 시대가 도래하면서, 모바일 환경에 맞춰진 웹 페이지 즉 모바일 웹에 대한 니즈가 폭발적으로 증가했고 그에 따른 성능 이슈도 함께 거론되었다. 데스크탑에 비해 성능이 낮은 모바일, 스마트폰을 통해 웹 페이지를 출력하기 위해서는 기존에 있었던 방식과는 다른 접근이 필요했고 그에 따라서 Singla Page web Application 기법 (SPA)이 등장하게 되었다.

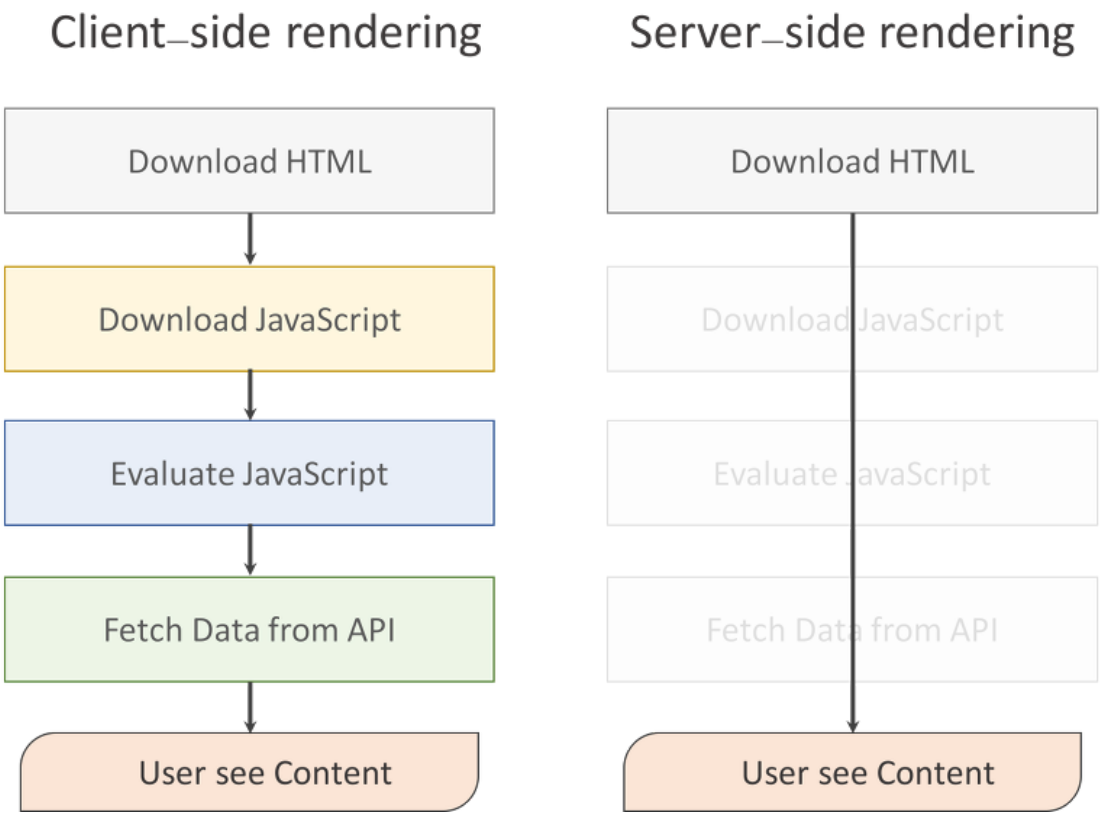
SPA는 브라우저에 로드되고 난 뒤에 페이지 전체를 서버에 요청하는 것이 아니라 최초 한번 페이지 전체를 로딩한 이후 부터는 데이터만 변경하여 사용할 수 있는 웹 애플리케이션을 의미한다. 전통적인 웹 방식(서버 사이드 렌더링)은 이 SPA 방식에 비해 성능 문제를 보였다. 요청 시 마다 새로고침이 일어나며 페이지를 로딩할 때마다 서버로부터 리소스를 전달받아 해석하고 화면에 렌더링하는 방식이였기 때문이다.

SPA는 트래픽을 감소시키고 사용자에게 더 나은 경험을 제공했다. 서버는 단지 JSON 파일만 보내주는 역할을 했고, html을 그리는 역할은 클라이언트 측에서 자바스크립트가 수행하게 된 것이다. 바로 이것이 **클라이언트 사이드 렌더링(Client-side rendering)**이다.

Angular JS와 Backbone JS같은 Single Page를 생성하기 쉬운 JS 프레임워크들이 등장했고 특히 AngularJS는 러닝커브를 제외한 많은 장점들 때문에 개발자들에게 큰 호응을 얻었다. (하지만 AngularJS는 느리다)

클라이언트 쪽이 점점 무거워지자 이에 반대로 View만 관리하자는 철학으로 React가 등장하게 되었다.

이렇게 클라이언트 사이드 렌더링과 서버 사이드 렌더링의 자리 싸움이 시작된 것이다.



각 렌더링의 장단점 비교

클라이언트 사이드 렌더링의 경우, 사용자의 행동에 따라 필요한 부분만 다시 읽어들이기 때문에 서버 측에서 렌더링하여 전체 페이지를 다시 읽어들이는 것보다 빠른 인터랙션을 기대할 수 있다. 서버 사이드 렌더링을 한다 하더라도 Ajax 기능을 위해 클라이언트 렌더링 요소가 포함될 수 밖에 없다. 이러한 점으로 미루어보아 클라이언트 측에서 렌더링을 하게 되면 서버 사이드 렌더링이 따로 필요하지 않기 때문에 일관성있는 코드를 작성할 수 있다.

하지만 문제도 존재한다.

클라이언트 사이드 렌더링은 페이지를 읽어들이는 시간, 자바스크립트를 읽어들이는 시간, 그리고 자바스크립트가 화면을 그리는 시간까지 모두 마쳐야 콘텐츠가 사용자에게 보여진다. 여기에 웹 서버에서 콘텐츠 데이터라도 가져와야 한다면 그 시간은 더욱 길어진다.

즉 초기 구동 속도가 느리다는 단점이 존재하는 것이다.

물론 초기 구동 속도를 제외하면 그 다음부터는 빠른 인터랙션의 성능을 보인다.

그리고 어떻게 보면 치명적이라고 할 수 있는 **검색 엔진 최적화의 문제가 존재한다.**

대부분의 웹 크롤러, 봇들이 자바스크립트 파일을 실행시키지 못한다는 것이다. 때문에 HTML 에서만 콘텐츠를 수집하게 되고 클라이언트 사이드 렌더링되는 페이지를 빈 페이지로 인식하게 된다. 추가적으로 **보안문제가 발생**한다. 기존의 서버 사이드 렌더링에서는 사용자에 대한 정보를 서버 측에서 세션으로 관리를 했다. 그러나 클라이언트 측에는 쿠키말고는 사용자에 대한 정보를 저장할 공간이 마땅치 않다.

서버사이드 렌더링의 장점은 반대의 경우로 생각하면 쉬울 것이다.

유저가 처음으로 컨텐츠를 접하게 되는 시점을 당길 수 있고, 서버따로, 클라이언트 따로 작성하던 코드가 하나로 합쳐진다.

물론 SEO 적용도 문제없다. 문제점은 사용자와 인터랙션 하는 부분일 것이다. 매번 서버에 request 요청을 통해서 해결해야하기 때문이다. DOM 조작에 있어서도 요청하는 과정과 엄청난 탐색비용으로 애를 먹고 있다.

React가 이 부분에 있어서 많은 해결책을 제시했지만 이는 두고 봐야 그 결과를 알 수 있을 것이다.

Angular 2(Angular)가 등장하고, Web Component에 대한 표준을 위해 각 브라우저 벤더들이 3년에 걸쳐 회의를 진행하고 있다. 구글은 이미 Web Component 시대를 대비하여 Polymer를 내놓았고, 각 벤더들은 주도권을 빼앗기지 않기 위해 표준화를 늦추고 있는 것 같다. 잘게 쪼갤 것인가, 하나로 뭉칠 것인가, 그 무엇하나 완벽하지 않고 각각의 장단점이 존재하는 것 같다. 맹목적으로 무엇이 좋다고 그 기술 스택을 사용하는 것이 아니라, 제공할 서비스가 어느 부분에 초점을 맞춰서 진행되느냐에 따라서 장점이 빛을 발하고 단점이 중요하지 않은 요소로 적용될 것이다.