# 1. 서버사이드 렌더링이란 무엇인가

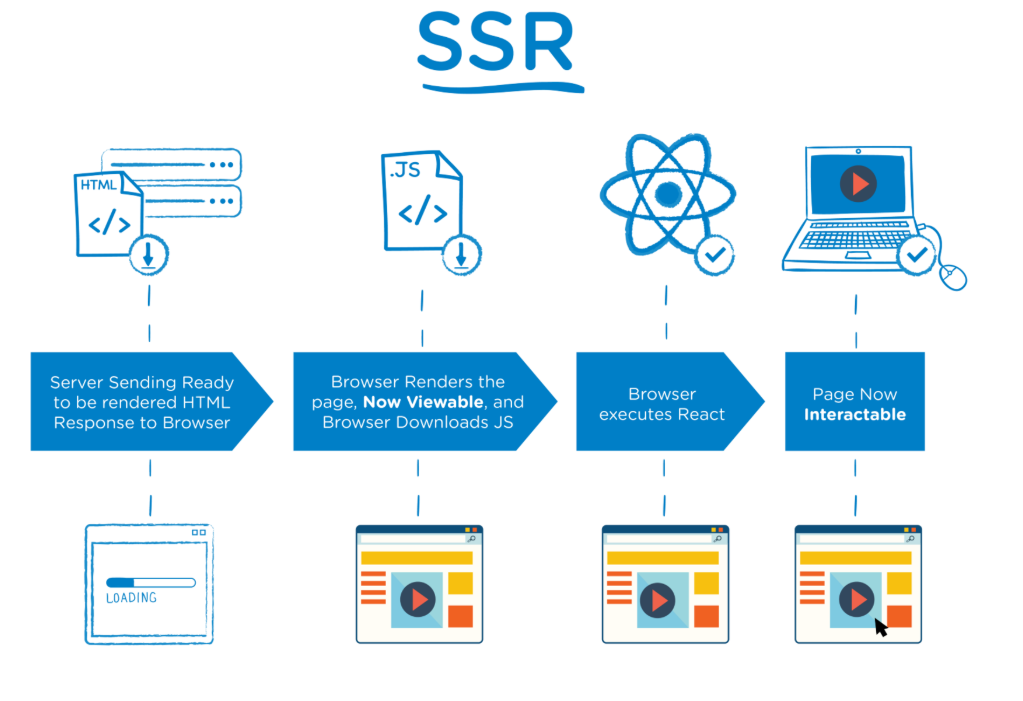
* 렌더링이란 웹 페이지에 접속했을 때 그 페이지를 화면에 그려주는 것을 의미한다. 서버사이드 렌더링은 서버에서 HTML과 view를 생성하여 화면을 제공하면 브라우저에서는 이를 그대로 표시하는 방식을 의미한다.

1. 장점

* 검색엔진최적화(SEO) 가능: SSR 방식의 가장 큰 장점
* 성능개선: 첫 렌더링된 html을 클라이언트에게 전달해주기때문에 초기로딩속도를 많이 줄여줄 수 있다. 자바스크립트 파일을 불러오고 렌더링 작업이 완료되기 전에 사용자가 사이트 컨텐츠를 이용할 수 있게 된다.

1. 단점

* 프로젝트의 복잡도: React에서 서버사이드렌더링을 구현할 경우 Router와 Redux와 함께 사용하다보면 복잡해질 수 있다.
* 성능의 악화 가능성: 서버사이드 렌더링을 하게 될 때는, ReactDOMServer.renderToString 함수를 사용한다. 이 함수는 동기적으로 작동한다. 그래서, 렌더링하는동안은 이벤트루프가 막히게 된다.



# 2. SPA란 무엇이고 어떤 장점이 있는가.

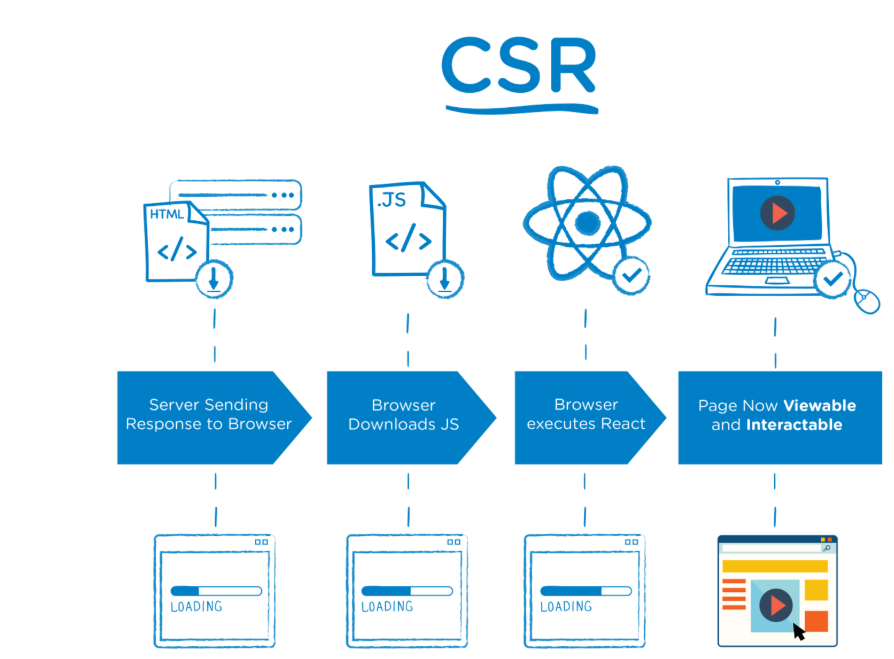
* SPA(Single Page Application)는 처음에 하나의 빈 페이지만 서버 측에서 제공하고 이후 view는 Client(브라우저)에서 자바스크립트를 통해 렌더링 하는 방식으로 작동하며 이런 렌더링 방식을 클라이언트사이드 렌더링이라고 한다.

1. 장점

* 트래픽감소: 필요한 데이터만 받아서 화면 표시
* 사용자경험: 새로고침이 발생하지 않아 사용자가 네이티브 앱과 비슷한 경험을 할 수 있다.

1. 단점

* 검색엔진 최적화 어려움: 자바스크립트 위주로 돌아가는 프로젝트는 자바스크립트 엔진이 돌아가지 않으면 원하는 정보를 표시해주지 못함. 크롬에서 react로 만든 웹앱의 소스를 확인하면, 내용이 비어있다. 그렇기때문에 검색엔진 크롤러가 데이터들을 제대로 수집하지 못한다.



→ SSR과 CSR 방식을 섞어서 첫 페이지 로드 시에는 SSR 방식을 이용하고 이후 모든 페이지에서는 CSR을 활용하여 두 방법의 장, 단점을 보완하는 방식도 활용되고 있음

출처1) <https://jaroinside.tistory.com/24>

출처2) <https://velog.io/@zansol/%ED%99%95%EC%9D%B8%ED%95%98%EA%B8%B0-%EC%84%9C%EB%B2%84%EC%82%AC%EC%9D%B4%EB%93%9C%EB%A0%8C%EB%8D%94%EB%A7%81SSR-%ED%81%B4%EB%9D%BC%EC%9D%B4%EC%96%B8%ED%8A%B8%EC%82%AC%EC%9D%B4%EB%93%9C%EB%A0%8C%EB%8D%94%EB%A7%81CSR>

# 3. 리액트에 사용되는 디자인패턴은 어떤 것들이 있는가. 각각 설명하시오.

* 디자인 패턴이란 다양한 상황에서 코드에 적용할 수 있는 잘 정의된 솔루션이다. 디자인 패턴은 모든 언어에 적용되며 어떤 클래스나 객체를 사용할지, 서로 어떤 영향을 주는 지에 관한 일종의 개요이다.

1. Observer 패턴: 서로 연결되지는 않았지만 데이터가 동기화되어 있길 원하는 두 컴포넌트가 있을 때 유용하다. observer 패턴은 Single source of truth에 의존하며 보통 이를 Subject라고 부른다.

* 작동 방식

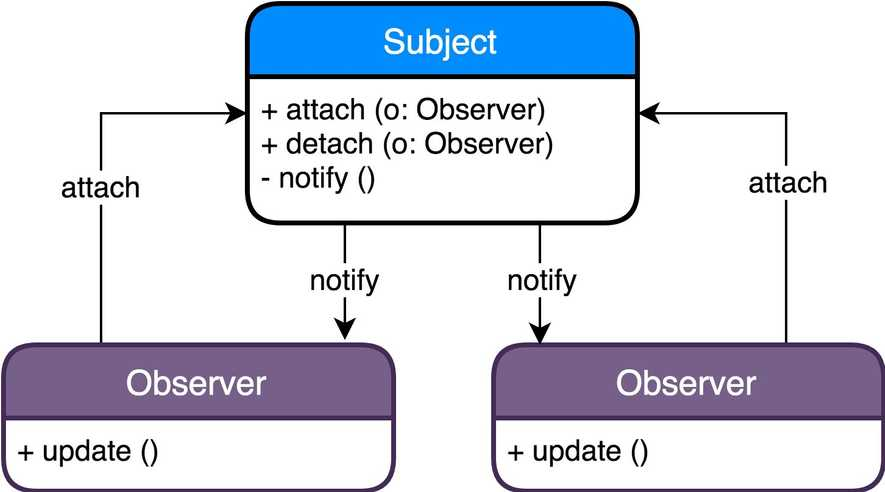
1. Observer가 자신을 Subject에 attatch 한 후 대기.
2. Subject가 업데이트
3. Subject 가지고 있는 최신 데이터를 Observer에 알림

* 장점

1. 컴포넌트 간의 낮은 의존성: 컴포넌트들이 서로 동기화 되기 위해 같은 트리에 있을 필요가 없다.
2. 단방향 데이터 흐름: 변화가 한 곳에서만 일어나며 추적하기 쉽다.

* 단점

1. 메모리 누수: 사용하지 않는 Observer들을 피하기 위해 사용되지 않는 컴포넌트를 Subject로 부터 detach할 필요가 있다.



1. Provider 패턴: 다수의 컴포넌트에서 사용 가능한 일반적인 객체를 만들기 원하고 그 객체가 변할 때마다 자식 컴포넌트가 업데이트 되길 원할 때 유용하다.

* 작동 방식

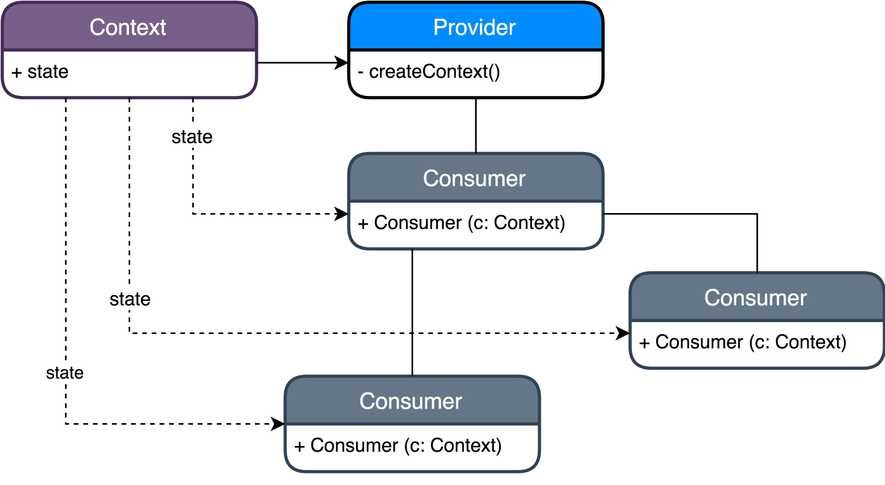
1. Provider가 컴포넌트 트리의 상위 레벨에 설정된 Context 객체에 일부 값 설정
2. Consumer라고 불리는 자식 컴포넌트는 props로 값을 전달받는 대신 context에서 직접 값을 가져옴

* 장점

1. prop drilling을 예방: 부모와 자식 컴포넌트간의 의존성을 덜어준다.
2. 단방향 데이터 흐름

* 단점

1. 보이지 않는 복잡성: 어떤 컴포넌트가 부모이고 데이터가 어디서 오는지 알기 어려움
2. 전역적인 값을 사용: 전반적으로 앱의 응집력이 높아짐 (React 공식 문서에는 Context를 적게 사용하는 것을 추천)



출처3) <https://delivan.dev/react/programming-patterns-with-react-hooks-kr/>