**1. 서버사이드 렌더링이란 무엇인가**

서버 사이드 렌더링이란 페이지를 서버측에서 먼저 간단하게 띄워주고, 완전하게 렌더링된 페이지를 클라이언트에게 다시 보내주는 기법을 말합니다.

전통적인 웹은 대부분 Server Side Rendering 이었습니다. 즉, 서버에서 브라우저에 나타나는 형태 그대로를 HTML로 만들어서 제공하고 브라우저는 HTML을 표시하는 방식입니다.

하지만 Ajax를 이용하는 등의 방법이 보편화되면서 SPA 개념이 생겨나고 SSR 효과를 가질 수 없게되었으며, 그것을 추가로 조치하는일이 생겨났습니다.

출처 :

<https://medium.com/@donggyu9410/%EA%B0%80%EC%9E%A5-%EC%89%AC%EC%9A%B4-%EB%B0%A9%EB%B2%95%EC%9C%BC%EB%A1%9C-%EB%A6%AC%EC%95%A1%ED%8A%B8%EC%97%90%EC%84%9C-%EC%84%9C%EB%B2%84%EC%82%AC%EC%9D%B4%EB%93%9C-%EB%A0%8C%EB%8D%94%EB%A7%81-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-966702610664>

**2. SPA란 무엇이고 어떤 장점이 있는가.**

단일 페이지로 구성된 웹 어플리케이션을 말한다. SPA는 화면이동 시에 필요한 데이터를 서버사이드에서 HTML으로 전달받지 않고(서버사이드 렌더링 X), 필요한 데이터만 서버로부터 JSON으로 전달 받아 동적으로 렌더링한다.

**기존 어플리케이션과 SPA의 차이**

기존 어플리케이션은 화면이동 시에 화면 이동에 필요한 HTML을 서버사이드에서 받아서 처음부터 다시 로딩하기 때문에 시간이 걸린다.

반면, SPA에서는 화면 구성에 필요한 모든 HTML을 클라이언트가 갖고 있고 서버사이드에는 필요한 데이터를 요청하고 JSON으로 받기 때문에 기존의 어플리케이션에 비해 화면을 구성하는 속도가 빠르다.

**장점**

하나하나 화면 전체를 렌더링할 필요가 없기 때문에 화면이동이 빠르다.

화면에 필요한 부분의 데이터만 받아서 렌더링 하기 때문에 처리과정이 효율적이다.

유저에 입장해서 사용하기 편리하다.

**단점**

처음 화면을 로딩할 때, 모든 화면이 미리 준비되어 있어야 하기 때문에 로딩에 시간이 걸린다.

어플리케이션을 구현하는데 보다 시간이 걸리며 복잡하다.

SEO 등 검색엔진 노출 문제등이 있습니다.

출처 <https://velog.io/@josworks27/SPA-%EA%B0%9C%EB%85%90>

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

디자인 패턴이란?

소프트웨어 디자인 패턴(software design pattern)은 소프트웨어 디자인에서 특정 맥락에서 공통적으로 발생하는 문제에 대해 재사용 가능한 해결책이다. - 위키피디아

**Observer 패턴**

Observer 패턴은 하나의 상태에 의존하는 여러 개의 객체들이 존재하는 케이스에서 유용하다.

React에서는, 서로 연결되진 않았지만 데이터가 동기화 되어 있으면 하는 두 컴포넌트가 존재할 때 유용하다.

Observer 패턴은 Single source of truth에 의존한다. 이는 보통 Subject 라고 불린다.

동작 방식은 먼저 Observer라고 불리는 것이 자신을 Subject에 attach 한 후 계속 기다린다. 그리고 Subject가 변경되면 자신의 최신 데이터를 attach 된 Observer 들을 한테 notify 한다.



예시로 축구 경기의 점수를 알려주는 웹 사이트를 만든다고 해보자.

우린 한 섹션은 현재 점수를 보여주고 다른 섹션은 골을 누가 넣었는지에 대한 히스토리를 보여주는 앱을 원한다. 또 골이 기록될 때 마다, 두 섹션이 동시에 업데이트가 되길 원한다.



두 섹션의 컴포넌트들이 서로 멀리 떨어져 있다고 가정해보자.

다음은 Observer 패턴을 이용하여 코드를 구성하는 방법이다.



우리의 GoalHistory를 GameSubject에 attach하고 골이 기록될 때 까지 기다린다. 골이 기록되면 골을 기록한 팀과 시간의 리스트가 업데이트 된다.

이렇게 Observer들을 정의함으로써 컴포넌트들이 서로 동기화되도록 할 수 있다.

**장점**

컴포넌트 간의 낮은 의존성: 컴포넌트들이 서로 동기화 되기 위해 같은 트리에 있을 필요가 없다.

단방향 데이터 흐름: 변화가 한 곳에서만 일어나며 추적하기 쉽다.

**단점**

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

**Provider 패턴**

Provider 패턴은 .Net Microsoft 앱 내에서 클래스의 초기화에 사용되는 다양한 조미료(React에선 props)를 제공하기 위해 처음 사용되었다. 이 패턴은 컴포넌트들이 ‘전역’ 상태를 가질 수 있게 하는 React Context API에 유용하게 사용되었다.

당신이 다수의 컴포넌트에서 사용가능한 일반적인 객체를 만들기를 원하고 그 객체가 변할때 마다 자식 컴포넌트들이 업데이트 되길 원할 때 유용하다. - React Context.Provider 공식문서

이 패턴의 주요한 이점은 트리 내의 컴포넌트마다 각각 다 props로 값을 넘겨야 하는 상황을 피할 수 있다는 것이다. 이 문제는 보통 “Prop Drilling”으로 잘 알려져 있다.

**동작 방식**은 먼저 Provider가 컴포넌트 트리의 상위 레벨에 설정된 Context 객체에 일부 값들을 설정한다. Consumer라고 불리는 자식 컴포넌트들은 props로 값들을 전달받는 것 대신, context로부터 값들을 직접 가져올 수 있다.



**장점**

prop drilling을 예방: 부모와 자식 컴포넌트간의 의존성을 덜어준다.

단방향 데이터 흐름

**단점**

보이지 않는 복잡성: 어떤 컴포넌트가 부모이고 데이터가 어디서 오는지 알기 어려움

전역적인 값을 사용하게 함 => 전반적으로 앱의 응집력이 높아짐 (**React 공식 문서에는 Context를 적게 사용하는 것을 추천**한다)

**container + presenter**

**프레젠테이션 컴포넌트**

어떻게 보여지는지와 관련있다.

프레젠테이션 컴포넌트와 컨테이너 컴포넌트가 모두 그 안에 들어가 있을것(\*\*)이고, 일부 DOM 마크업과 스타일도 가지고 있다.

종종 this.props.children 을 통해서 노출된다.

Flux 액션이나 stores 등과 같은 앱의 나머지 부분들에 의존적이지 않다.

데이터를 가져오거나 변경하는 방법에 대해서 관여할 필요가 없다.

props 를 통해 배타적으로 callback 함수와 데이터를 받는다.

상태를 거의 가지고 있지 않다(만약 상태를 가지고 있다면, 데이터에 관한 것이 아닌 UI 상태에 관한 것이다).

만약 상태, 생명주기, hooks, 또는 퍼포먼스 최적화가 필요없다면, 유틸함수로서 쓰여질것이다.

예를들면 페이지, 사이드바, 스토리, 유저정보, 리스트 등이 있다.

**컨테이너 컴포넌트**

어떻게 동작하는지와 관련있다.

프레젠테이션 컴포넌트와 마찬가지로 프레젠테이션 컴포넌트와 컨테이너 컴포넌트 모두 가지고 있지만 감싼 divs 를 제외하고는 DOM 마크업을 가지고 있지 않는다. 스타일 역시 가지고 있지 않는다.

데이터와 기능(행동)을 프레젠테이션 컴포넌트와 다른 컴포넌트에 제공한다.

Flux(or Redux) 액션을 호출하고, 프레젠테이션 컴포넌트에 콜백함수로써 제공한다.

데이터 소스 역할을 하기 때문에 상태가 자주 변경된다.

직접 만드는것 보단 대게 React Redux 의 connect() 함수, Relay 의 createContainer() 함수, Flux Utils 의 Container.create()와 같은 Higher Order Components를 이용해서 만들어진다.

예를들면 유저페이지, 팔로워 사이드바, 스토리 컨테이너, 팔로우한 유저 리스트 등이 있다.

**장점**

이 방법으로 컴포넌트를 작성하면 당신의 앱(기능)과 UI 에 대한 구분을 이해하기가 더 수월하다.

재사용성이 더 뛰어나다. 완전히 서로 다른 상태값과 함께 같은 프레젠테이션 컴포넌트를 사용할 수 있고, 재사용 될 수 있는 별도의 컨테이너 컴포넌트로 변경할 수 있다.

프레젠테이션 컴포넌트는 말하자면 앱의 팔레트와 같다. 앱의 싱글페이지 위에서 앱의 로직을 건드리지 않고 디자이너에게 모든 변화를 조정하게 할 수 있다.

이것은 사이드바, 페이징, 컨텍스트메뉴와 같은 레이아웃 컴포넌트를 추출하도록 할것이고, 이것은 동일한 마크업이나 몇몇의 컨테이너 레이아웃을 반복해서 작성하는 대신 this.props.children 을 통해서 구현될 수 있다.

컴포넌트는 DOM 을 생성하지 말아야 합니다. 컴포넌트는 단지 UI 와 관련된 것들을 조합하는 것을 제공하는 것이 필요합니다.

출처 <https://blueshw.github.io/2017/06/26/presentaional-component-container-component/>

<https://dev-yong.tistory.com/12>

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