1. React 소개(1)

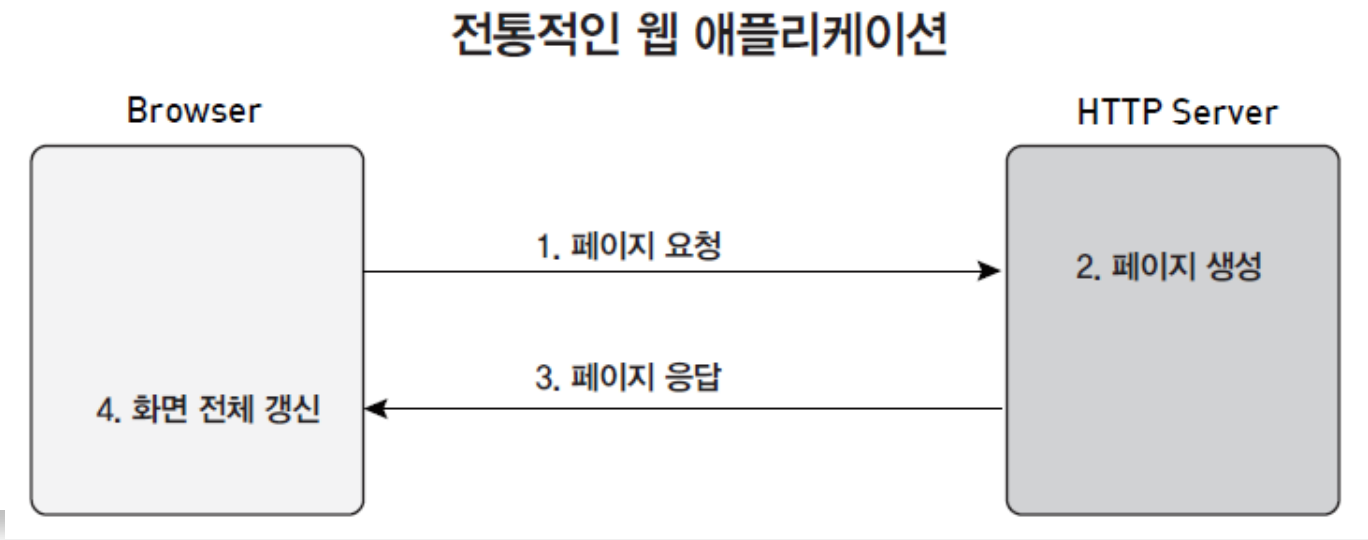
React란?

- Facebook에서 만든 UI를 작성하기 위한 자바스크립트 라이브러리
  - A JavaScript library for building user interfaces
- Javascript 코드만을 이용해 조합형 UI를 작성함.
  - JSX와 같은 형식을 사용하지만 결국 트랜스파일되어 자바스크립트 코드로 실행함.
- MVC 중 V(View)에 집중함
- 데이터가 지속적으로 변하는 대규모 애플리케이션의 구축을 위해서 만들어졌음
- 프레임워크가 아니라 라이브러리
  - 프레임워크는 전체 시스템의 구조를 변경해야 함.
  - 라이브러리는 전체 시스템의 골격을 유지한 채로 일부만 변경할 수도 있음.

1. React 소개(2)

전통적인 웹 애플리케이션

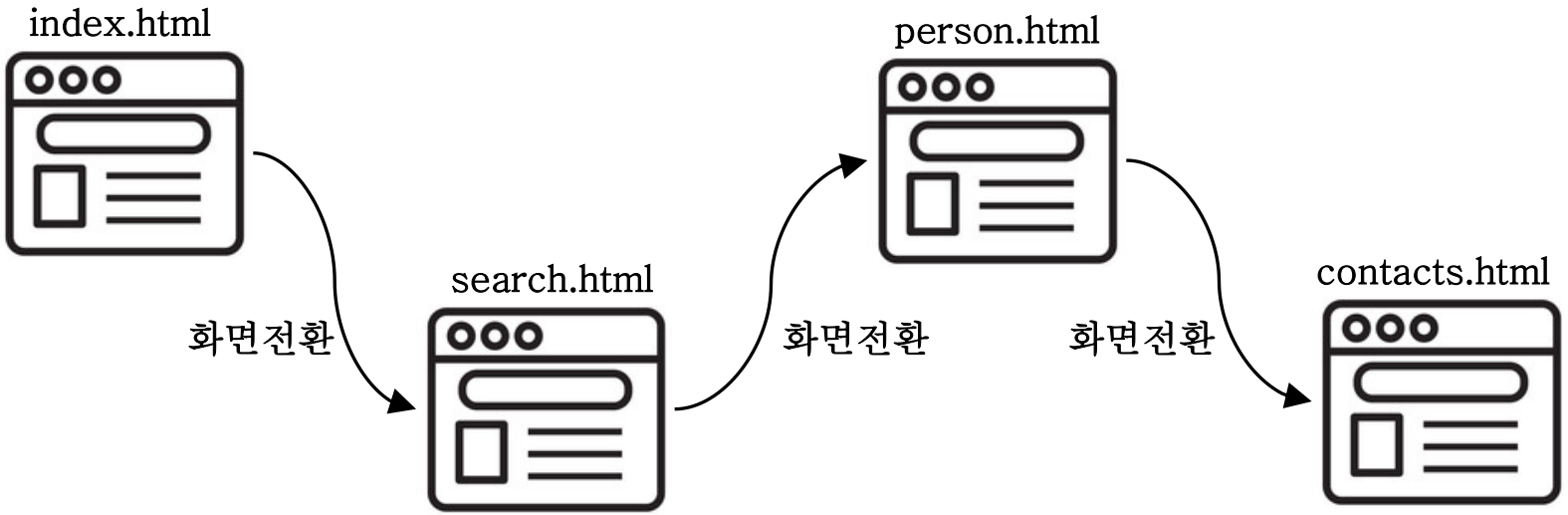
- 요청의 단위가 페이지임
- 화면의 페이지 단위로 구성됨. 따라서 화면의 변경을 위해 서버로부터 새로운 페이지를 내려받음
  - 페이지 단위의 요청과 응답의 문제점은 화면 일부분만 갱신하고 싶어도 페이지 전체를 HTTP 서버로부터 다시 받아와야 한다는 것을 의미
  - HTTP 서버는 브라우저의 요청이 있을 때마다 페이지 전체를 재생성하여 전송



1. React 소개(3)

AJAX를 이용한 멀티 페이지 애플리케이션

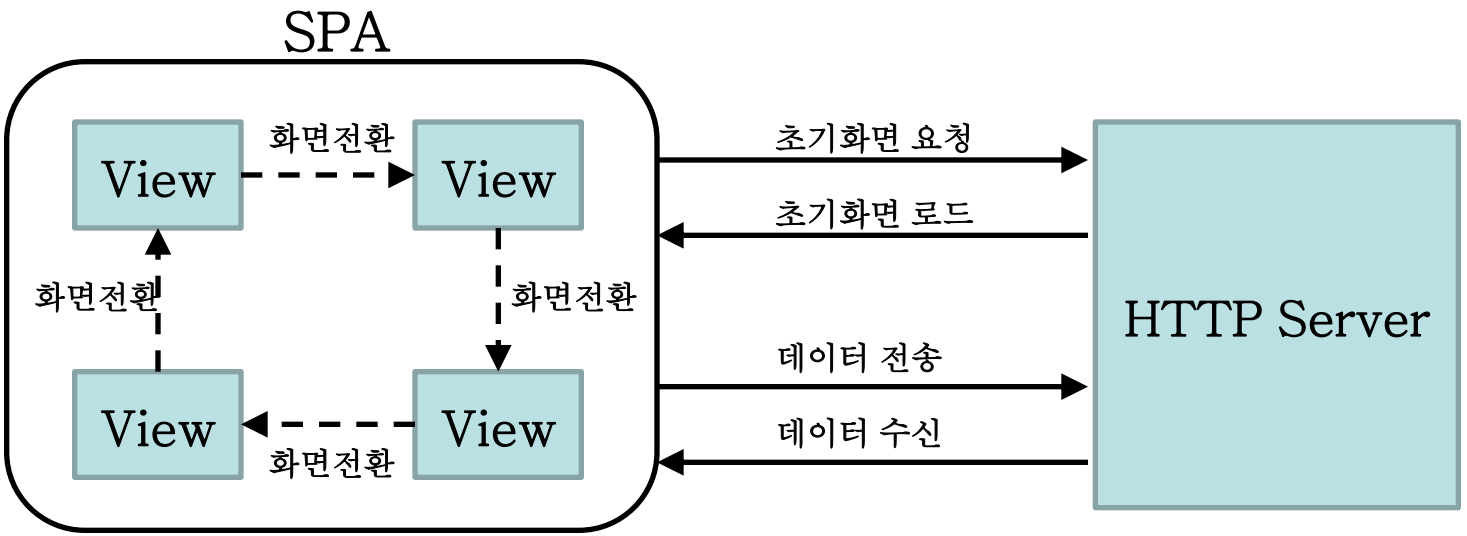
- 화면이 전환될 때는 새로운 HTML 페이지로 이동함
- 전화된 HTML 페이지가 초기 화면을 구성함
- 데이터를 이용한 UI 생성과 서버로 데이터를 전송하는 작업은 AJAX를 이용함



1. React 소개(4)

단일 페이지 애플리케이션

- SPA(Single Page Application)
- 하나의 페이지로 애플리케이션의 모든 화면을 제공함.
- AJAX, Socket 등의 방법을 이용해 서버와 데이터 교환



1. React 소개(5)

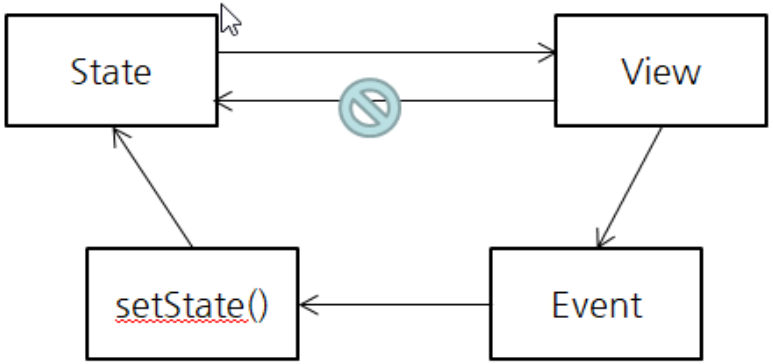
단일 페이지 애플리케이션의 단점

- 효과적인 상태관리가 요구됨
  - 애플리케이션에서 보여줘야 하는 데이터. 즉 상태(State)를 브라우저에서 관리해야 함.
  - 각각의 화면마다 상태를 관리하기 쉽지 않음. 한 상태 데이터를 여러 화면이 이용하는 경우가 있기 때문에...
- 느린 DOM
  - SPA는 DOM을 아주 빈번히 갱신하는데, 브라우저의 DOM을 직접 다루는 것은 대단히 느리다.
  - 이것으로 인해 화면 전화, 데이터 변경이 일어날 때 사용자 경험이 훼손될 수 있다.
- HTML 마크업을 생성하도록 자바스크립트 코드로 제어해야 함.
  - HTML 마크업을 문자열로 이어붙이는 작업은 개발 생산성을 떨어뜨림.
  - 일부 라이브러리가 렌더링 기능을 제공함.

1. React 소개(6)

React의 특징

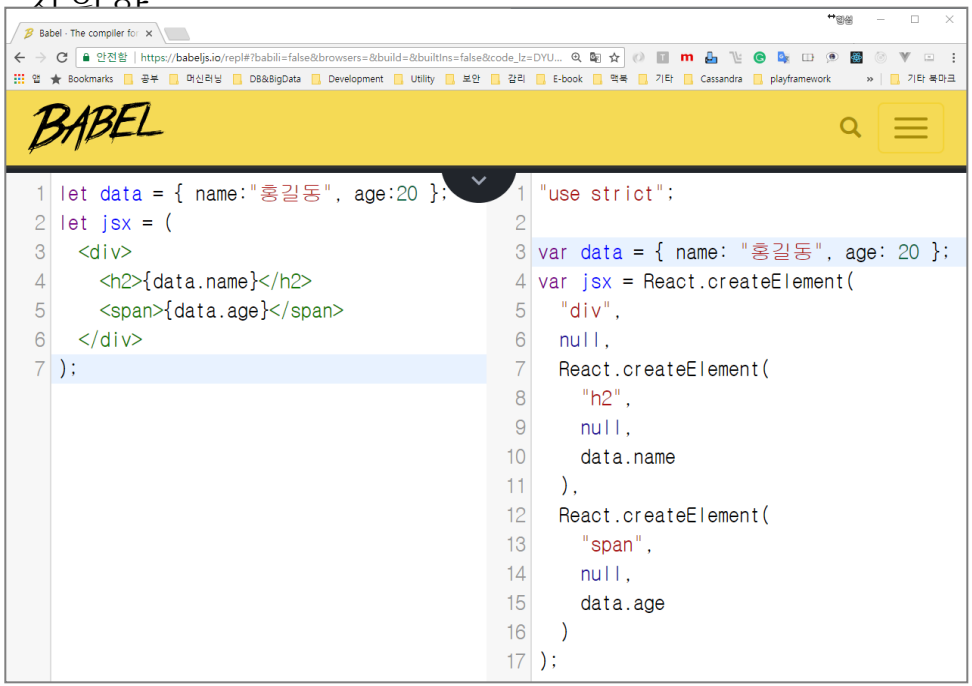
- 상태 관리
  - 컴포넌트 단위로 상태를 관리할 수 있음
  - Flux, Redux와 같은 라이브러리를 사용하면 상태관리를 앱단위로 중앙집중화할 수 있음
- 단방향 데이터 바인딩
  - 상태 데이터가 변경되면 즉시 UI가 반영됨
  - 번거로워보일 수 있지만 상태 데이터가 변경되는 과정이 명시적으로 예측 가능하고, 추적가능하도록 함.



1. React 소개(7)

React의 특징(이어서)

- JSX(Javascript XML)
  - 템플릿이 아닌 XML과 유사한 자바스크립트 확장 문법임.
  - 필수는 아니지만 자바스크립트 코드 내부에서 HTML 마크업을 사용할 수 있도록 지원함



1. React 소개(8)

React의 특징(이어서)

- 컴포넌트 기반의 개발
  - React 컴포넌트는 독립적인 컴포넌트 --> 뛰어난 재사용성
  - 템플릿, HTML 지시문등을 이용하지 않고 자바스크립트 언어로 개발함.
    - 템플릿을 이용하는 것은 표현적, 기능적 제약이 있을 수 있음
  - 마크업, 스타일, 자바스크립트 코드의 분리가 아닌 결합
    - 밀접한 연관성이 있으므로...
    - 관심사의 분리를 컴포넌트 단위를 만드는 방식으로 분리시킴.

1. React 소개(9)

React의 특징(이어서)

- 가상 DOM : Virtual DOM
  - DOM 조작은 느리므로 가상 DOM을 사용함
    - 앱의 상태가 바뀔때 전체 DOM을 업데이트하지 않고 원하는 DOM 상태와 유사한 가상 트리를 형성함.
    - 이후 전체 DOM 모드를 생성하지 않고도 실제 DOM을 가상 DOM과 같이 만드는 방법을 알아냄
  - 잠시 jQuery의 상황을 살펴보자면...
    - jQuery는 업데이트시에 Selector를 사용해 DOM 요소를 직접 찾아 업데이트함.
    - 가상 DOM이 필요없음
  - React는?
    - Always re-render on update!!
    - 개발자가 원하는 출력물만을 선언적으로 작성하기 때문에 컴포넌트 전체를 re-render 하듯이 개발할 수 밖에 없음
    - 따라서 UI 성능을 위해서 가상 DOM이 반드시 필요함
    - 가상 DOM 트리를 비교하면서 필요한 부분만 업데이트함. --> 화면 업데이트 로직은 신경쓰지 않아도 됨.
  - 가상 DOM의 개념을 익힐 수 있는 동영상
    - https://www.youtube.com/watch?v=BYbgopx44vo

1. React 소개(10)

React의 특징(이어서)

- Virtual DOM과 Diff 알고리즘
  - DOM 트리의 노드들을 비교하면서 노드가 다른 유형일 경우 기존 노드를 버리고 새로운 노드로 교체
  - 노드가 같은 유형인 경우
    - Attribute와 Style을 비교하여 변경함.
    - 커스컴 컴포넌트의 경우이면서 전달하는 속성이나 상태가 다른 경우 새로운 속성을 컴포넌트로 전달해 rerender가 실행됨

1. React 소개(10)

React의 특징(이어서)

- 컴포넌트 기반의 개발
  - React 컴포넌트는 독립적인 컴포넌트 --> 뛰어난 재사용성
  - 템플릿, HTML 지시문등을 이용하지 않고 자바스크립트 언어로 개발함.
    - 템플릿을 이용하는 것은 표현적, 기능적 제약이 있을 수 있음
  - 마크업, 스타일, 자바스크립트 코드의 분리가 아닌 결합
    - 밀접한 연관성이 있으므로...
    - 관심사의 분리를 컴포넌트 단위를 만드는 방식으로 분리시킴.

2. 첫번째 React App 작성

- React와 Babel을 CDN 방식으로 참조하여 간단하게 구성
- sample1.html

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>첫번째 React App</title>
</head>
<body>
    <div id="app"></div>
    <script src="https://unpkg.com/react/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom/umd/react-dom.development.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.38/browser.js"></script>
    <script type="text/babel">
    //함수형 React Component!!
    let Hello = function(props) {
        return (
            <h1>Hello World!!!</h1>
        );
    }
    ReactDOM.render(<Hello />, document.getElementById('app'));
    </script>
</body>
</html>
```

2. 첫번째 React App 작성

sample2.html

- 조금더 복잡한 함수형 React 컴포넌트 작성.
- 두개의 컴포넌트를 조합하여 UI 구성
- props를 이용해 컴포넌트로 렌더링에 필요한 데이터 전달
  - 구체적인 내용은 천천히 학습

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>두번째 React App</title>
</head>
<body>
    <div id="app"></div>
    <script src="https://unpkg.com/react/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom/umd/react-dom.development.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.38/browser.js"></script>
```

(다음 페이지에 이어짐)

2. 첫번째 React App 작성

Sample2.html(이어서)

```
<script type="text/babel">
  //함수형 React 컴포넌트
  var Title = function(props) {
     return(
        <div>
           <h2>{props.title}</h2>
           <hr />
        </div>
     )
  }

  var TodoList = function(props) {
     var todolist = props.todos.map((item, index) => {
        let status = ''
        if (item.done == true)  status ='완료'
        else status = '진행중'
        return (<li key={item.id}>{item.todo} - {status}</li>)
     });

     return (
        <ul>{todolist}</ul>
     );
  }
( 다음 페이지에 이어짐 )
```

2. 첫번째 React App 작성

Sample2.html(이어서)

```
    var App = function() {
        var data = {
            title : '해야할 일 목록',
            todos : [
                { id:1, todo:"리액트 공부", done:false },
                { id:2, todo:"ES6 정리", done:true },
                { id:3, todo:"HTML5 학습", done:true },
                { id:4, todo:"명상", done:false }
            ]
        }

        return (
            <div>
                <Title title={data.title} />
                <TodoList todos={data.todos} />
            </div>
        );
    }

    ReactDOM.render(<App />, document.getElementById('app') );
    </script>
</body>
</html>
```

2. 첫번째 React App 작성

React 개발 워크 플로우

- 예전에는……
  - 모든 자바스크립트를 한 파일에 작성하고 모두 한 페이지에서 <script> 태그로 참조
  - 외부 라이브러리 몇 개 추가로 참조한 뒤 페이지 작성
- 대규모 웹앱이라면 예전과 같이 할 수 없음.
- 일반적인 요구사항
  - JSX를 작성하고 자바스크립트로 트랜스파일
  - 코드를 모듈 패턴으로 작성
  - 의존성에 대한 관리와 자바스크립트 코드 번들링
  - Webpack!!