# 리액트 스터디 3주차

### 20 황수연

## 보일러플레이트 프로젝트의 목적

내부원리를 잘 몰라도 리액트 코드를 쓸 수 있게 해준다.

만들 프로젝트에는 js파일들, 라이브러리가 여러 개 있는 프로젝트 파일이 있다. 이 파일들은 ES6나 ES2016로 쓰여져있는데, 자바스크립트 문법과는 조금 다르다. 현재 ES6의 모든 기능을 완전히 지원하는 브라우저가 없으므로 ES6 코드를 쓸 수 있어도 브라우저에서 실제로 실행하기 어려울 수 있다. 따라서 툴링(tooling) 혹은 트랜스파일(transpile)의 과정을 거친다. (*트랜스파일: 코드를 같은 수준의 다른 언어로 바꿔주는 과정) 트랜스파일을 위해 웹팩(webpack)이라는 도구를 이용하고,  바벨(babel) 혹은 바벨JS(babel.js)라고 불리는 라이브러리로 구성할 것이다. 웹팩과 바벨의 목적은 ES6 코드가 브라우저에서 바로 실행되지 못하는 문제를 해결하기 위해 이를 트랜스파일하여 브라우저에서 돌아가는 코드로 변형하는 것이다. 웹팩과 바벨을 거친 후에,  application.js, main.js, app.js(파일명 중요X)같은 하나의 파일로 나오게 된다.

## npm 설치

필요한 파일들 설치 후 cmd 창에

```
>> npm start
```

입력하면 보일러 플레이트 패키지를 시작하고 로컬 서버를 실행시킨다.

*이 패키지 안에서 js파일을 작성하고, 바벨과 웹팩이 모든 것들을 합치고 이를 ES5로 변환하여 브라우저에서 돌아가게끔 만든다는 것 기억하기. 그리고 이 모든 파일들을 볼 수 있는 로컬 서버를 이용할 수 있게 한다.

이 패키지의 디폴트 호스트는 `localhost:8080`이다.

## 리액트의 이해

리액트는 자바스크립트 라이브러리로, 웹 브라우저에 보여지는 HTML을 만드는 라이브러리이다. 따라서 리액트 코드를 작성하는 것은 컴포넌트(component) 혹은 뷰(view)를 만드는 것과 같다.

컴포넌트는 소스 코드의 일부이며, 이 코드가 HTML을 생성한다. 그러므로 컴포넌트나 뷰를 떠올릴 때 HTML을 생성하는 것이라고 생각하면 된다. 컴포넌트는 HTML을 만드는 자바스크립트 함수의 모음집이다. 각 컴포넌트들은 각자 다른 함수나 목적을 갖는다. 그러므로 자

바스크립트로 컴포넌트를 만드는 것은 궁극적으로 HTML을 만드는 것이다. 컴포넌트 생성 후, 이는 DOM과 같은 형태로 만들어져야 한다는 것을 확인해야 한다. 만든 컴포넌트는 자동으로 HTML 문서에 삽입되는 것이 아니다.

코드 작성 시 각 파일 당 무조건 하나의 컴포넌트만 만든다.

리액트 코드를 작성하는 것은 여러 개의 다른 컴포넌트를 작성하는 것이다. 그리고 이 컴포넌트들을 다른 방식으로 배치하면서 복잡한 애플리케이션을 간단하게 만들 수 있다.

## 클래스 기반 컴포넌트

클래스 컴포넌트는 내부적인 정보를 저장하려 할 때 사용한다. 예를 들어 입력창에 사용자가 입력한 내용을 다른 컴포넌트에 알려주려고 하는 경우 사용한다. 이러한 점이 함수형 컴포넌트와 다르다. 클래스 기반으로 생성한 리액트 컴포넌트는 반드시 render 메소드가 정의되어 있어야 한다. 그 자체로 렌더링할 수 있는 능력이 필요하기 때문이다. (즉, JSX 변환 능력)

## JSX

JSX는 부분적인 템플릿 혹은 변형된 자바스크립트로, 자바스크립트 안에 HTML처럼 보이는 소스코드를 사용할 수 있게 도와준다. 그러나 ES6의 `const` 와 같은 구문은 JSX가 해석할 수 없다. 따라서 웹팩과 바벨같은 보일러플레이트 패키지가 필요하다. 이들의 목적은 JSX를 실제 바닐라 자바스크립트로 변환하여 브라우저가 이해할 수 있게 만드는 것이기 때문이다.

JSX를 사용하는 이유는, 해당 컴포넌트를 렌더링(컴포넌트를 HTML 페이지에 띄우는 것)할 때 실제 HTML을 생성하고 DOM에 삽입하기 위함이다.  결국 궁극적인 목적은 자바스크립트 코드를 HTML로 만들기 위한 것이다. 또한, 복잡한 컴포넌트를 만드는 경우 Javascript 코드가 길어지고 복잡해지기 때문에 이러한 점을 방지하고 더욱 깔끔하게 보기 위해 사용한다.

⇒ 과정: JSX → HTML → 페이지의 DOM에 삽입(실제 사용자가 보는 것)

또한 JSX는 그 자체로 중첩을 통해 사용하면서 HTML을 흉내낼 수 있다.

```
const App = function(){
  return <ol>
      <li>1</li>
      <li>2</li>
      <li>3</li>
      </ol>;
}
```

## 자바스크립트 모듈

자바스크립트 모듈은 서로 분리된 코드 파일들을 다른 파일들 혹은 다른 곳에 설치한 라이브러리와 분리할 수 있다. 따라서 다른 파일에 선언된 코드들은 그 파일에 접근할 것이라고 따로 명령하지 않는 한 현재 우리가 보는 코드와는 아무런 소통을 할 수 없다. 다음과 같은 문장을 쓰면 다른 파일에 작성된 코드/라이브러리 를 불러와 사용할 수 있다.

```
import React from 'react';
```

> react라는 라이브러리를 불러와 React라는 변수에 할당한다.

리액트는 두 개의 라이브러리로 나뉘어진다.  위에서 불러온 코어 리액트 라이브러리는 리액트 컴포넌트와 어떻게 작동하는지(렌더링 방법, 이것들을 모으는 방법 등)를 알고 있다.  그러나 실제로 DOM에 렌더링하는 기능을 가지고 있는, 즉 이 컴포넌트를 가져와 DOM에 삽입하는 라이브러리는  리액트 돔(React DOM)이라고 부른다.

→ 리액트는 컴포넌트를 생성, 관리 / 리액트 돔은 실제 DOM과 상호작용

# Export 구문과 컴포넌트 불러오기

```
import SearchBar from './components/search_bar';  // 주소
```

직접 만든 새 파일이 있다면, 그 파일에 대한 참조나 주소를 제공해야하고, npm으로 설치한 라이브러리라면 패키지 이름만 작성하면 된다.

# 이벤트 핸들링

1. 이벤트 핸들러 선언

   이벤트 핸들러는 이벤트 발생 시마다 실행되는 함수이다.

2. 살펴보려는 요소에 이벤트 핸들러 전달

보통 핸들러 이름은 `handle/on + 이벤트가 일어나는 요소의 이름 + 이벤트 이름` 으로 작성한다. (ex. onInputChange) 이후, 살펴보려는 요소에 이벤트 핸들러를 전달할 때는, 태그 안에 `on이벤트 이름 = { 메소드 혹은 이벤트핸들러 참조명 }` 으로 한다. (ex. <input `onChange = {this.onInputChange}` /> ) 이 예시는 input이라는 요소를 만들고, onChange라는 프로퍼티에 {}안의 value를 저장하는 것이다.

또한 이벤트 핸들러를 요소에 추가하면 이벤트가 일어날 때마다 event 객체를 전달받는다.

```
onInputChange(event){
  ...
}
```

이때 매개변수 이름이 꼭 event일 필요는 없다. 이 event객체는 발생한 이벤트에 대한 정보, 컨텍스트를 갖고 있다.

## State(상태) 이해하기

State란 자바스크립트 객체로서, 사용자 이벤트를 저장하고 그것에 반응할 때 이용된다. 클래스 기반의 컴포넌트들은 그 자체의 State 객체를 갖는다. 컴포넌트의 state가 바뀔 때마다 컴포넌트는 리렌더링하고, 자식 요소들도 강제로 리렌더링하도록 한다. 오직 클래스 기반 컴포넌트만 state를 가진다.

컴포넌트 안의 state를 사용하기 전에 state 객를 거쳐야 한다. 초기화하기 위해서는 state 프로퍼티를 클래스의 constructor 메소드 안에 넣는다.

```
class SearchBar extends Component {
  constructor(props){
    super(props);

    this.state = { term: '' };
}
```

모든 자바스크립트 클래스는 특별한 메소드인 constructor를 가진다. 이 함수는 해당 클래스의 새로운 인스턴스가 생성될 때마다 자동적으로 제일 먼저 호출된다. 클래스 안에서 뭔가를 설정할 때, 예를 들면 변수나 상태값을 초기화할 때 활용된다.

위에서 상속받은 Component는 그 자체만의 constructor 함수를 갖는다. 부모 클래스에 메소드를 미리 정의했을 때, super를 통해 부모 클래스의 메소드를 호출할 수 있다.

state를 쓸 때마다 새로운 객체를 생성하면서 초기화를 하고 this.state에 할당한다. 전달한 객체에는 state를 기록하고자 하는 프로퍼티를 포함한다. 위와 같은 경우에는 term이라는 프로퍼티를 가져오려고 하는 것이다. 따라서 검색 input 값을 사용자가 업데이트 할 때마다 프로퍼티 term이 업데이트되거나 그것의 변경사항을 받아온다.

컴포넌트 안에서, constructor함수 이외의 곳에서 state값을 변경하기 위해서는 `this.setState`를 사용한다. 따라서 `this.state.term = event.target.value;` 같은 식으로는 변경할 수 없다. setState는 지속적으로 유지되게 하며 리액트 백단에서 state를 위한 여러 가지 일을 한다. 그냥 값을 바꾸면 리액트는 값이 바뀐 것을 알지 못하므로 setState() 라는 메소드를 통해 리액트에게 알리게 되는 것이다.

state가 업데이트되거나 this.setState가 호출될 때 컴포넌트가 자동적으로 리렌더링되도록 하고, 렌더링 메소드의 업데이트된 모든 정보를 DOM에 push한다. render 메소드는 this.state.term을 참조하고, 컴포넌트가 리렌더링될 때마다 DOM 안의 this.state.term이 업데이트된다.

# 제어 필드(control field)

제어 필드는 input처럼 state에 의해 그 값이 세팅되는 form element를 말한다. input에게 this.state.term을 통해 value를 제공받았다면 이 input은 이제 제어 컴포넌트로 바뀌게 된다. 제어 컴포넌트는 state에 의해 값이 설정된다. 따라서 state가 변하면 값이 변한다. 이전에는 input이 state에게 업데이트되어야 한다고 명령을 했다면, 제어 컴포넌트로 된 지금은 반대로 작동한다. input 변화는 여전히 state에게 변화한 값이 있다고 말해준다. this.setState는 컴포넌트가 리렌더링되게 하고, 리렌더링될 때 input 값은 this.state.term에 새로 설정된다.

⇒전체적인 과정: 앱이 시작되면 searchBar 인스턴스를 렌더링한다. 새로운 인스턴스의 생성으로 constructor 메소드가 호출되고, this.state.term 은 빈 문자열로 초기화된다.  컴포넌트는 this.state.term의 값을 가져와서 input 값을 렌더링한다. 따라서 초기값은 빈 문자열이다. 사용자가 문자를 입력하면 state가 업데이트된다. this.state.term의 값이 input값과 같게 되는 것이다. 이때 input이 바뀌지 않았다는 점이 중요하다. onChange핸들러가 실행될 때 input값은 실제로 바뀌지 않는다는 것이다. 이벤트 핸들러가 실행되면 this.state.term에 새로운 값이 대입된다. setState가 호출될 때 컴포넌트는 즉시 리렌더링되므로 렌더링 함수가 다시 호출되면 input 값은 this.state.term의 새 값을 받아 업데이트된다. 결국 컴포넌트는 렌더링을 끝내고, input의 새로운 값은 화면에 나타나게 된다.

중요한 점은 사용자가 문자를 입력하는 것이 실제로 입력 값을 바꾼 것이 아니라, 이벤트를 발생시키도록 한 것이다. 이 이벤트를 통해 state를 업데이트했기 때문에 input 값이 바뀌도록 하는것이다.

input값은 state와 같다. 이것이 유용한 이유는 input안에 기본값을 정하고 싶을 때 사용할 수 있기 때문이다.(placeholder 아님) 또한 input값을 jquery를 사용하는 것과 같이 쉽게 읽어올 수 있다.

# 실습

1. 함수를 작성하면 DOM에 렌더링되는 컴포넌트의 인스턴스를 만든다. 따라서 ReactDOM.render(App); 과 같은 식으로 쓰면 class를 전달해준거나 마찬가지이므로, DOM에 렌더링하기 전에 컴포넌트를 인스턴스화해야한다.
2. 유튜브 API 사용하기

   1) 유튜브 API key를 위해 회원가입 하기

2) 단순한 검색을 제공하는 패키지 설치

3.

```
class SearchBar extends React.Component{
  ...
}
```

위의 코드는 아래같이 쓰는 것이 가능하다.

```
import React, { Component } from 'react';

class SearchBar extends Component{
  ...
}
```

import문에서 {}의 의미는 react 라이브러리를 불러와 component 프로퍼티를 Component 라는 변수로 가져오라는 뜻이다.