React.js

jsx 와 컴포넌트

first coding

first coding

React.js

1. JSX
2. 컴포넌트

first coding

React.js

JSX

JSX

- JSX
  - A syntax extension to JavaScript.
  - 자바스크립트의 확장 문법 → 자바스크립트의 문법을 확장 시킨 것
  - JavaScript와 XML/HTML을 합친 것 → JavaScript and XML의 앞 글자를 따서 JSX라고 부름.

```
const element = <h1>Hello, React.js</h1>;
```

  - 자바스크립트와 HTML 코드가 결합된 상태

JSX의 역할

- JSX 역할
  - JSX는 내부적으로 XML/HTML 코드를 자바스크립트로 변환하는 과정을 거친다.

```
class Hello extends React.Component{
    rend() {
        return <h1>Hello, React.js {this.props.msg}</h1>
    }
}

ReactDOM.render(
    <Hello msg="SON"></Hello>, document.getElementById("root")
);
```

JSX의 역할

- JSX 역할
  - JSX 코드를 자바스크립트 코드로 변환하는 역할을 하는 것이 바로 리액트의 createElement()라는 함수
  - JSX 문법을 사용하면 리액트에서는 내부적으로 모두 createElement라는 함수를 사용하도록 변환
  - 최종적으로는 이 createElement() 함수를 호출한 결과로 자바스크립트 객체 생성

```
class Hello extends React.Component {
  render() {
    return React.createElement('div', null, 'Hello {this.props.msg}')
  }
}

ReactDOM.render(
  React.createElement(Hello, {msg:'SON'}, null), document.getElementById("root")
);
```

JSX의 역할

- JSX 역할
  - 아래 두 개의 코드는 JSX를 사용한 코드와 사용하지 않은 코드이며, 모두 동일한 역할을 함.

```
const elemnet1 = (
  <h1 className='title'>Hello, React.js </h1>
)

const elemnet2 = React.createElement(
  'h1',
  { className : 'title'},
  'Hello, React.js'
)
```

- 리액트는 JSX 코드를 모두 createElement () 함수를 사용하는 코드로 변환하기 때문에 리액트에서 JSX를 사용하는 것은 필수가 아님.
- 하지만JSX를 사용했을 때 코드가 더욱 간결해지고 생산성과 가독성이 올라가기 때문에 사용하는 것을 권장

JSX의 역할

- JSX 역할
  - createElement() 함수를 호출한 결과로 자바스크립트 객체가 생성 → 리액트에서는 이 객체를 엘리먼트라고 함.

```
const elemnet3 = {
  type: 'h1',
  props: {
    className: 'title',
    children: 'Hello, React.js'
  }
}
```

JSX의 역할

- JSX 역할
  - createElement () 함수의 파라미터

```
const elemnet2 =
React.createElement(
  type,
  [props],
  [...children]
)
```

- **Type** : 엘리먼트의 유형
  - 나 같은 HTML 태그가 올 수도 있고, 다른 리액트 컴포넌트가 들어갈 수도 있다
- **props**
  - 엘리먼트의 속성
- **children**
  - 현재 엘리먼트가 포함하고 있는 자식 엘리먼트

JSX의 장점

- JSX의 장점
  - 코드가 간결해짐

JSX 사용함

```
<div>Hello, {name}</div>
```

JSX 사용 안 함

```
React.createElement('div', null, `Hello, ${name}');
```

  - 코드가 간결해지면서 가독성이 향상됨
    - 코드를 작성할 때뿐만 아니라 유지 보수 관점에서 가독성 이 높을수록 유리

JSX의 장점

- JSX의 장점
  - Injection Attack 해킹에 대비한 보안 코딩이 가능
  - Injection Attack
    - 입력창에 문자나 숫자 같은 일반적인 값이 아닌 자바스크립트 코드와 같은 소스코드를 입력하여 해당 코드가 실행되도록 만드는 해킹 방법

```
const element = <h1>{title}</h1>;
```

- JSX 코드에서는 괄호를 사용해서 title 변수를 임베딩combedding (삽입)하고 있다.
- 하지만 기본적으로 ReactDOM은 렌더링하기 전에 임베딩된 값을 모두 문자열로 변환하기 때문에 명시적으로 선언되지 않은 값(소스코드)은 괄호 사이에 들어갈 수 없다.
- 결과적으로 XSS(cross-site-scripting attacks) 를 방어

JSX 코딩해보기

- Player.jsx : 컴포넌트 작성

```
import React from "react";

function Player(props){
    return (
        <ul>
            <li>선수이름 : ${props.playerName} </li>
            <li>선수번호 : ${props.playerNumber} </li>
        </ul>
    )
}

export default Player;
```

JSX 코드 작성해 보기

- Team.jsx : 상위 컴포넌트 작성

```
import React from "react";
import Player from "./Player";

function Team(props){
    return (
        <div>
            <Player playerName='SON' playerNumber='7'></Player>
            <Player playerName='LEE' playerNumber='20'></Player>
            <Player playerName='PARK' playerNumber='9'></Player>
        </div>
    )
}

export default Team;
```

JSX 코드 작성해 보기

- index.js 수정

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import Team from './test/Team';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Team />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

first
coding

엘리먼트 렌더링

엘리먼트

- 엘리먼트의 정의
  - 리액트 앱을 구성하는 가장 작은 블록들
  - 화면에 나타나는 내용을 기술하는 자바스크립트 객체를 가르키는 용어 → 엘리먼트
  - 리액트 엘리먼트는 브라우저에 표현되는 DOM 엘리먼트의 가상 표현
    - 즉 화면에 아직 랜더링 되지 않은 것을 리액트 엘리먼트(Virtual DOM)라 한다
    - 리엑트 엘리먼트는 화면에 보이는것을 기술(표현)한 것이고 엘리먼트를 토대로 화면에 출력되는 DOM을 만드는 것

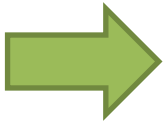
```
const element = <h1>Hello, React.js</h1>;
```

    - 위의 코드가 리액트 엘리먼트를 생성하는 코드.
    - 리액트는 이 엘리먼트로 화면에 출력할 DOM엘리먼트를 생성

엘리먼트

- 엘리먼트의 구조
  - 리액트 엘리먼트는 자바스크립트 객체 형태
  - 컴포넌트 유형 (예: Button)과 속 성(예: color) 및 내부의 모든 자식children에 대한 정보를 포함하고 있는 일반적인 자바스크립트 객체

```
{
    type: 'button',
    props: {
        className: 'bg-green',
        children: {
            type: 'b',
            props: {
                children: 'Hello, element!'
            }
        }
    }
}
```



```
<button class="bg-green">
    <b>
        Hello, React.js
    </b>
</button>
```

엘리먼트

- 엘리먼트의 구조
  - 리액트 엘리먼트는 자바스크립트 객체 형태

```
import React from "react";

function Button(props) {
    return (
        <button className={`bg-${props.color}`}>
            <b>
                {props.children}
            </b>
        </button>
    )
}

function ConfirmDialog(props) {
    return (
        <div>
            <p>내용을 확인하셨다면 확인버튼을 눌러주세요. </p>
            <Button color='Green'>확인</Button>
        </div>
    )
}

export default ConfirmDialog;
```

```
<button class="bg-green">
    <b>
        Hello, React.js
    </b>
</button>
```

엘리먼트

- **엘리먼트의 특징**
  - 불변성mmutable 을 갖고 있음
    - 한 번 생성된 엘리먼트는 변하지 않는다 → 엘리먼트 생성 후에는 children이나 attributes를 바꿀 수 없다
  - 엘리먼트를 변경을 위해서는 기존의 것을 변경하는 것이 아닌 새로운 엘리먼트를 만들어 기존의 엘리먼트와 바꾸면 된다.
  - 엘리먼 트는 불변성을 갖고 있기 때문에 화면에 새로운 내용을 보여주기 위해 새로운 엘리먼트를 만들 어서 기존 엘리먼트가 연결되어 있는 부분에 바꿔 적용하면 된다.
  - 리액트 엘리먼트의 불변성이라는 특징은 상태 관리와 더 불어 화면이 갱신되는 횟수(얼마나 화면이 자주 바뀌는지)는 실제 리액트를 이용한 개발 과정 에서 성능에 큰 영향을 미치는 요소 이므로 잘 기억하고 있어야 함.

엘리먼트 렌더링하기

- Root DOM node

```
<div id="root"></div>
```

  - root라는 id를 가진 태그
  - 모든 리액트 앱에 필수적으로 들어가는 코드
  - 이 태 그 안에 리액트 엘리먼트들이 렌더링
  - 이 태그 안에 있는 모든 것이 리액트 DOM에 의해서 관리
  - 리액트만으로 만 들어진 모든 웹사이트들은 단 하나의 Root DOM node를 가지게 됨.
  - 기존에 있던 웹사이트에 추가적으로 리액트를 연동하게 되면 여러 개의 분리된 수많은 Root DOM node를 가질 수도 있음.

엘리먼트 렌더링하기

- Root DOM node

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <ConfirmDialog/>
  </React.StrictMode>
);
```

  - 엘리먼트를 하나 생성하고 생성된 엘리먼트를 root div에 렌더링하는 코드
  - 렌더링을 위해 ReactDOM의 render()라는 함수를 사용
  - 리액트 엘리먼트 HTML 엘리먼트에 렌더링하는 역할
  - 리액트의 엘리먼트는 리액트의 Virtual DOM에 존재하는 것이고, HTML 엘리먼트는 실제 브라우저의 DOM에 존재하는 것.
  - 리액트의 엘리먼트가 렌더링되는 과정은 Virtual DOM에서 실제 DOM으로 이동하는 과정

렌더링된 엘리먼트 업데이트하기

- 렌더링된 엘리먼트 업데이트하기
  - 엘리먼트는 한 번 생성되면 바꿀 수 없기 때문에 엘리먼트를 업데이트하기 위해서는 엘리먼트를 새로 생성 해야 함.

```
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import Tick from './test/Tick';

const root = ReactDOM.createRoot(document.getElementById('root'));

function tick(){
  root.render(
    <React.StrictMode>
      <Tick/>
    </React.StrictMode>
  );

}

setInterval(tick, 1000);

reportWebVitals();
```

시계 만들기

- Clock.jsx

```jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import reportWebVitals from './reportWebVitals';
import Clock from './test/Clock';

const root = ReactDOM.createRoot(document.getElementById('root'));

function tick(){
  root.render(
    <React.StrictMode>
      <Clock/>
    </React.StrictMode>
  );

}

setInterval(tick, 1000);

reportWebVitals();
```

```jsx
import React from "react";

function Clock(props) {
    return (
        <div>
            <h1>Clock</h1>
            <h3>현재 시간 : {new Date().toLocaleTimeString()}</h3>
        </div>
    )
}

export default Clock;
```

first
coding

컴포넌트와 Props

컴포넌트

- **컴포넌트**
  - 리액트에서는 모든 페이지가 컴포넌트로 구성되어 있고, 하나의 컴포넌트는 또 다른 여러 개의 컴포넌트의 조합으로 구성될 수 있다.
  - 컴포넌트 기반이라고 부르는 것은 작은 컴포넌트들이 모여서 하나의 컴포넌트를 구성하고, 또 이러한 컴포넌트들이 모여서 전체 페이지를 구성하기 때문
  - 리액트 컴포넌트는 개념적으로 자바스크립트의 함수와 비슷함.
  - 리액트 컴포넌트는 props 값을 입력을 받아서 리액트 엘리먼트 객체를 반환.
    - 리액트 컴포넌트가 해주는 역할은 어떠한 속성들을 입력으로 받아서 그에 맞는 리액트 엘리먼트를 생성하여 리턴 해주는 것

Props

- Prop의 개념
  - 리액트 컴포넌트의 속성
    - 같은 리액트 컴포넌트에서 눈에 보이는 글자나 색깔 등의 속성을 바 꾸고 싶을 때 사용하는 컴포넌트의 속 재료

    - 컴포넌트의 모습과 속성을 결정하는 것이 바로 props
    - props는 컴포넌트에 전달 할 다양한 정보를 담고 있는 자바스크립트 객체

Props

- Prop의 특징
  - props 의 중요한 특징은 읽기 전용 (ReadOnly)
    - 값을 변경할 수 없다
    - props 의 값은 리액트 컴포넌트가 엘리먼트를 생성하기 위해서 사용하는 값.
    - 다른 props의 값으로 엘리먼트를 생성하려면 새로운 값을 컴포넌트에 전달하여 새로 엘리먼트를 생성 해야 한다.
    - 이 과정에서 엘리먼트가 다시 렌더링 된다.

```
// pure 함수
// input 을 변경하지 않으며 같은 input 에 대해서 항상 같은 output 을 리턴
function sum(a,b) {
    return a + b;
}
```

→ **Pure 하다**

같은 입력 값에 대해서는 항상 같은 출력 값을 리턴 한다

```
// impure 함수
// input 을 변경함
function withdraw(account, amount) {
    account.total -= amount;
}
```

→ **Impure 하다**

입력으로 받은 파라미터 값을 변경했다.

Props

- Prop의 특징
  - 컴포넌트의 특징
    - All React components must act like pure functions with respect to their props.

      ➔ 모든 리액트 컴포넌트는 그들의 props에 관해서는 Pure 함수 같은 역할을 해야 한다.

      모든 리액트 컴포넌트는 props를 직접 바꿀 수 없고, 같은 props에 대해서는 항상 같은 결과를 보여줄 것!
    - 함수가 Pure 하다는 것은 함수 내부에서 함수의 입력 값인 파라미터를 바꿀 수 없다는 의미 이기 때문에 리액트 컴포넌트에서는 props를 바꿀 수 없다는 의미
    - Pure 함수는 같은 입력 값에 대해서는 항상 같은 결과를 보여줘야 하기 때문에, 리액트 컴포넌트 관점에서 같은 props에 대해서 항상 같은 결과(리액트 엘리먼트)를 보여줘야 한다.

리액트 컴포넌트의 props는 컴포넌트 내부에서 바꿀 수 없고,
같은 props가 들어오면 항상 같은 엘리먼트를 반환해야 한다

Props

- Prop의 사용법
  - JSX를 사용하는 경우에는 아래 코드와 같이 키와 값으로 이루어진 키-값 쌍의 형태로 컴포넌트에 props를 넣을 수 있다.

```
function Team(props){
    return (
        <Player playerName='SON' playerNumber={7}></Player>
    )
}
```

  - 각 속성에 값을 넣을 때 중괄호를 사용한 것과 사용하지 않은 것의 차이
  - props에 값을 넣을 때에도 문자열 이외에 정수, 변수, 그리고 다른 컴포넌트 등이 들어갈 경우에는 중괄호로 감싸주어야 한다.
  - 문자열을 중괄호로 감싸도 상관없다.

Props

- Prop의 사용법
  - JSX를 사용하는 경우에는 아래 코드와 같이 키와 값으로 이루어진 키-값 쌍의 형태로 컴포넌트에 props를 넣을 수 있다.

```
function Team(props){
    return (
        <Player playerName='SON' playerNumber={7}></Player>
    )
}
```

  - 각 속성에 값을 넣을 때 중괄호를 사용한 것과 사용하지 않은 것의 차이
  - props에 값을 넣을 때에도 문자열 이외에 정수, 변수, 그리고 다른 컴포넌트 등이 들어갈 경우에는 중괄호로 감싸주어야 한다.
  - 문자열을 중괄호로 감싸도 상관없다.

컴포넌트 만들기

- **컴포넌트의 종류**
  - 클래스 컴포넌트와 함수 컴포넌트
    - 리액트의 초기 버전에서는 클래스 컴포넌트를 주로 사용
    - 함수 컴포넌트를 개선해서 주로 사용하게 되었다.
    - 함수 컴포넌트를 개선하는 과정에서 훅(hook)이 개발되어 현재 리액트 개발에서는 거의 혹을 사용

컴포넌트 만들기

- **함수 컴포넌트**
  - 모든 리액트 컴포넌트는 Pure 함수 같은 역할을 해야 한다
  - 리액트의 컴포넌트를 일종의 함수

```
function  Welcome(props){
    return <h1>안녕~!! {props.name} </h1>
}
```

  - 이 함수는 하나의 props 객체를 받아서 인사말이 담긴 리액트 엘리먼트 리턴 ➔ 리액트 컴포넌트 (함수형 컴포넌트)

컴포넌트 만들기

- **클래스 컴포넌트**
  - 클래스 컴포넌트는 자바스크립트 ES6의 클래스라는 것을 사용해서 만들어진 형태의 컴포넌트
  - 클래스 컴포넌트의 경우에는 함수 컴포넌트에 비해서 몇 가지 추가적인 기능을 갖고 있다.

```
class Welcome extends React.Component {
    render() {
      return <h1>안녕~!! {props.name} </h1>
    }
}
```

- 함수 컴포넌트 Welcome과 동일한 역할을 하는 컴포넌트를 클래스 형태로 만든 것
- 함수 컴포넌트와의 가장 큰 차이점은 리액트의 모든 클래스 컴포넌트는 React.Component를 상속mberitance받아서 만든다는 것
- React.Component라는 클래스를 상속받아서 Welcome이라는 클래스를 만들었고, 이는 React.Component를 상속받았기 때문에 결과적으로 리액트 컴포넌트가 되는 것.

컴포넌트 만들기

- **컴포넌트의 이름**
  - 리액트는 소문자로 시작하는 컴포넌트를 DOM 태그로 인식하기 때문에 컴포넌트의 이름은 항상 대문자로 시작해야 한다
  - 예를 들어 나 과 같이 사용하는 것은 내장 컴포넌트라는 것을 뜻하며, 'div'나 'span'과 같은 문자열 형태로 React.createElement() 에 전달됩니다.
  - 하지만 <Foo /> 와 같이 대문자로 시작하는 경우에는 React.createElement (Foo)의 형태로 컴파일되며 자바스크립트 파일 내에서 사용자가 정의했거나 임포트mport한 컴포넌트를 가리킵니다.
  - 그렇기 때문에 컴포넌트 이름은 항상 대문자로 시작해야 합니다.

컴포넌트 만들기

- **컴포넌트 렌더링**
  - 컴포넌트를 다 만든 이후에 실제로 렌더링
    - 렌더링을 위해서는 가장 먼저 컴포넌트(**엘리먼트를 생성하는 역할**)로부터 엘리먼트를 생성

```
// DOM 태그를 사용한 element
const element1 = <div />;

// 사용자가 정의한 컴포넌트를 사용한 element
const element2 = <Player name="손흥민" />;

function Player(props) {
  return <h1> 안녕, {props.name}</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  element2
);
```

컴포넌트 합성

- **컴포넌트 합성**
  - 여러 개의 컴포넌트를 합쳐서 하나의 컴포넌트를 만드는 것
  - 리액트에서는 컴포넌트 안에 또 다른 컴포넌트를 사용할 수 있기 때문에, 복잡한 화면을 여러 개의 컴포넌트 로 나눠서 구현할 수 있다.

```
// 컴포넌트
function Player(props) {
  return <h1> 안녕, {props.name}</h1>;
}

// 컴포넌트
function App() {
  return (
    <div>
        <Player name='손흥민'/>
        <Player name='이강인'/>
        <Player name='김민재'/>
    </div>
  );
}
```

컴포넌트 추출

- 컴포넌트 추출Extracting components
  - 컴포넌트 합성과 반대로 복잡한 컴포넌트를 쪼개서 여러 개의 컴포넌트로 나누는 것
  - 큰 컴포넌트에서 일부를 추출해서 새로운 컴포넌트를 만드는 것
    - 컴포넌트가 작아질수록 해당 컴포넌트의 기능과 목적이 명확해지고, props도 단순해지기 때문에 컴포넌트의 재사용성이 올라가고 동시에 개발 속도도 향상.

컴포넌트 추출

- 컴포넌트 추출Extracting components
  - 댓글 영역을 표현하는 컴포넌트
    - 회원 정보 출력 영역
    - 댓글 출력 영역
    - 댓글 작성 날짜 출력 영역

```
function Comment(props){
    return (
        <div className="comment">

            {/* 회원 정보 */}
            <div className="userinfo">
                <img className="userimg"
                    src="{props.user.imgUrl}"
                    alt="{props.user.userName}"
                />
                <div className="userinfoname">
                    {props.user.userName}
                </div>
            </div>

            {/* reply Content */}
            <div className="commentContent">
                {props.content}
            </div>

            {/* reply replydate */}
            <div className="replydate">
                {props.replydate}
            </div>

        </div>
    )
}
```

컴포넌트 추출

- 컴포넌트 추출Extracting components
  - 댓글 영역을 표현하는 컴포넌트

```
function UserImg(props) {
    return (
        <img className="userimg"
            src={props.user.imgUrl}
            alt={props.user.userName}
            width="200"
        />
    )
}
```

```
function UserInfo(props) {
    return (
        <div className="userinfo">
            <UserImg user={props.user} />
            <div className="userinfoname">
                {props.userName}
            </div>
        </div>
    )
}
```

```
function Comment(props) {
    return (
        <div className="comment">

            {/* 회원 정보 */}
            <UserInfo user={props.user}/>

            {/* reply Content */}
            <div className="commentContent">
                {props.content}
            </div>

            {/* reply replydate */}
            <div className="replydate">
                {props.replydate}
            </div>

        </div>

    )
}
```

컴포넌트 추출

- 컴포넌트 추출Extracting components
  - 댓글 영역을 표현하는 컴포넌트

```
import logo from './logo.svg';
import './App.css';
import Comment from './components/Comment';

const comment = {
  user : {
      userName : '손흥민',
      imgUrl :
'https://assets.goal.com/v3/assets/bltcc7a7ffd2fbf71f5/bltaf10a2ea551a3e54/6360dc8f67675010b765f257/GettyImages
-1432946487.jpg'
  },
  content : '대한민국~!!',
  replydate : '2022-12-25'
}

// 컴포넌트
function App() {
  return (
    <Comment user={comment.user} content={comment.content} replydate={comment.replydate}/>
  );
}

export default App;
```

댓글 컴포넌트 만들기

- CommentList.jsx

```
import React from "react";
import Comment from "./Comment";

const comments = [
    {         …     },
    {         …     },
    {         …     }
]

function CommentList(props) {
    return (
        <div>
            {comments.map((comment) => {
                return (
                    <Comment user={comment.user} content={comment.content} replydate={comment.replydate} />
                );
            })}

        </div>

    )
}

export default CommentList;
```

댓글 컴포넌트 만들기

- CommentList.jsx

```
const comments = [
    {
        user: {
            userName: '손흥민1',
            imgUrl: 'https://assets.goal.com/v3/assets/aa.jpg'
        },
        content: '대한민국~!!111',
        replydate: '2022-12-25'
    },
    {
        user: {
            userName: '손흥민2',
            imgUrl: 'https://assets.goal.com/v3/assets/bb.jpg'
        },
        content: '대한민국~!!222',
        replydate: '2022-12-26'
    },
    {
        user: {
            userName: '손흥민3',
            imgUrl: 'https://assets.goal.com/v3/cc.jpg'
        },
        content: '대한민국~!!333',
        replydate: '2022-12-27'
    }
]
```

댓글 컴포넌트 만들기

- CommentList.jsx

```
import logo from './logo.svg';
import './App.css';
import CommentList from './components/CommentList';


// 컴포넌트
function App() {
  return (
    <CommentList />
  );
}

export default App;
```