기존에 요소에 접근하여 값을 바꾸는 것이 아니라,

그 요소의 값이 리뉴얼 되면 그냥 document를 통쨰로 다시 쓰는 방식으로

사용자 경험을 좋게하면서 성능을 최대한 아낀 페이스북에서 개발한 React임

render함수를 호출해서 재구성도 함

이전과의 차이를 비교해 필요한 부부만 다시 write

virtual DOM

"지속저그로 데이터가 변화하는 대규모 애플리케이션 구축하기"

리액트는 뷰 만을 신경씀(MVC와 달리)

따라서 리액트는 프레임워크가 아니라 라이브러리임

yarn create react-app (프로젝트명)

cd (프로젝트명)

yarn start를 하면 로컬서버 구동됨

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

JSX란?

자바스크립트의 확장 문법이며 = XML과 매우 비슷함

JSX에서

*function* App(){

  return(

    <div>

      Hello <b>react</b>

    </div>

  );

}

는

*function* App(){

  return React.createElement("div", null, "hello", React.createElement("b", null, "react"));

}

를 반환하며,

이를 편리하게 사용하기 위해 React.createElement를 재외하고 JSX문법으로 return에 html의 형태로 코드를 작성하면 된다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

App,js : App에서 JSX를 반환(main 화면)

index.js: 만든 컴포넌트를 태그를 쓰듯이 render안에 작성함

root.render(

  <*React.StrictMode*>

    <*App* />

  </*React.StrictMode*>

);

결국 renderin 되면 public/index.html에 html로 완성됨

JSX문법!!

1. 리액트 컴포넌트는 하나의 부모로 시작하는 DOM을 만족해야됨
2. *function* App() {  //App이라는 컴포넌트를 생성하고, return로 App을 하면 함수에서 반환하고 있는 내용을 보여준다
3. return (        //이건 html도 문자열도 아닌 JSX 문법임
4. <*Fragment*>
5. <h1>React</h1>
6. <h2>넌 뒤졌다</h2>
7. </*Fragment*>
8. );
9. }

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

2. JSX 안에는 JS를 쓸 수 있다.

JSX 내부에서 {}로 감싸고 사용하면 된다.

*function* App() {  //App이라는 컴포넌트를 생성하고, return로 App을 하면 함수에서 반환하고 있는 내용을 보여준다

*const* jsVariable = "진짜로";

  return (        //이건 html도 문자열도 아닌 JSX 문법임

    <*Fragment*>

      <h1>React</h1>

      <h2>넌 뒤졌다 {jsVariable}</h2>

    </*Fragment*>

  );

}

오 JS에서 저장하면 바로바로 웹상에 반영됨

*function* App() {  //App이라는 컴포넌트를 생성하고, return로 App을 하면 함수에서 반환하고 있는 내용을 보여준다

*const* jsVariable = "진짜로";

  return (        //이건 html도 문자열도 아닌 JSX 문법임

    <*Fragment*>

      <h1>React</h1>

      <h2>넌 뒤졌다 {jsVariable == "진짜로" ? (<h1>진짜로</h1>): (<h3>쩝</h3>)}</h2>

    </*Fragment*>

  );

&&를 이용하면 보여주고 안보여주고를 더 간단하게 가능

*const* jsVariable = "1";

  return (        //이건 html도 문자열도 아닌 JSX 문법임

    <*Fragment*>

      <h1>React</h1>

      <h2>넌 뒤졌다 {jsVariable === 1 && <h1>진짜로</h1>}</h2>

    </*Fragment*>

  );

결과: 안나옴 (===)

한줄 return은 여전히 ()를 안하고 작성해도 무관함

undefined는 단독으로는 render자체가 안됨

따라서 || 을이용해서 처리함

6. 인라인 스타일링

문자열 형태가 아니라 객체 형태로 넣어야됨

*const* jsVariable = 1;

*const* name = undefined;

*const* style = {

    backgroundColor : "black",

    color: "white"

  };

  return (        //이건 html도 문자열도 아닌 JSX 문법임

    <*Fragment*>

      <h1>React</h1>

      <h2>넌 뒤졌다 {jsVariable === 1 && <h1 style={style}>진짜로</h1>}</h2>

      <h3>{name || "undefined입니다"}</h3>

    </*Fragment*>

  );

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

7. className

JSX에서는 class = 대신 className = 을 사용한다.

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

css

.reactColor{

*background-color*: brown;

*color*: aqua

}

App.js

<*Fragment*>

      <h1 className='reactColor'>React</h1>

      <h2>넌 뒤졌다 {jsVariable === 1 && <h1 style={style}>진짜로</h1>}</h2>

      <h3>{name || "undefined입니다"}</h3>

    </*Fragment*>

<Fragment>는 <>와 동일 (<div>가 꺼려진다면 ES6에서 나온 fragment를 이용)

8. 닫아야 하는 태그

<*Fragment*>

      <h1 className='reactColor'>React</h1>

      <h2>넌 뒤졌다 {jsVariable === 1 && <h1 style={style}>진짜로</h1>}</h2>

      <h3>{name || "undefined입니다"}</h3>

      <input></input>

      <input/>

    </*Fragment*>

html에서는 input은 열기만 해도 되지만 JSX에서는 닫기를 해야됨

또한 <input/>를 사용하면 여는 동시에 닫을 수 있는 self-closing 태그를 작성할 수 있다.

9. 주석

<input/> {/\* 이게 주석임 \*/}

2.5 ESLint와 Prettier 사용하기

스크린샷, 텍스트, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명를 선택해서 ESLint가 알려주는 오류를 빌드 전에 알 수 있다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명를 통해서 저장을 통해서만도 prettier를 사용할 수 있다.

3장. 컴포넌트

3.1 클래스형 컴포넌트

2장까지는 함수 컴포넌트에 대해서 다뤘음

함수형 컴포넌트와 class형 컴포넌트는 둘 다 동일함

*class* App extends Component {

  render() {

*const* name = "react";

    return <div calssName="react">{name}</div>;

  }

}

다만 나중에 배우는 state기능 및 라이프사이클 기능을 사용할 수 있는 것과 임의 메서드를 정의할 수 있다는 점이 다르다./

클래스형 컴포넌트 = render(JSX)의 구성으로 되어 있어야한다.

3-2 컴포넌트 생성

*const* MyComponent = () *=>* {

    //react component는 대문자로 시작해야됨

    return <div>마이 뉴 띵</div>;

};

export default MyComponent;

import logo from './logo.svg';

import './App.css';

import MyComponent from './MyComponent';

*const* App = () *=>* {

    return <*MyComponent* />;

};

export default App;

3-3. props

MyComponent.js

*const* MyComponent = (*props*) *=>* {

    //react component는 대문자로 시작해야됨

    return (

        <*Fragment*>

            <h1>{*props*.name} = Me</h1>

            <h2>{*props*.age}살 입니다</h2>

            <div>마이 뉴 띵</div>

        </*Fragment*>

    );

};

App.js

*const* App = () *=>* {

    return <*MyComponent* name="Kim" age="20" />;

};

component에서 props를 설정하면, app.js에서 넘겨주는 properties를 사용해서 JSX를 구성할 수 있다.

props의 갑의 default도 설정이 가능하다.

MyComponent.defualtProps = {

    name: 'Park',

    age: '24'

};

만약 컴포넌트 tag안에 text를 추가했다면,

그것은 props 요소의 children으로 들어감

<*MyComponent* name="Kim">children</*MyComponent*>

<div>{*props*.children} is here</div>

ES6의 비구조 할당 문법을 활용하면 props의 값을 추출할 수 있다.

propTypes를 이용하여 props에 이용하는 특성의 타입을 지정할 수 있다.

isRequired를 통해서 props가 지정되지 않았을 console창에 경고를 띄울 수 있다.

import PropTypes from 'prop-types';

MyComponent.propTypes = {

    name: PropTypes.string,

    age: PropTypes.number.isRequired

};

클래스 형에서는

this.props를 조회하면 props객체를 불러올 수 있다.

3-4. state

React에서 state은 컴포넌트 내부에서 바뀔 수 있는 값을 의미한다.

*constructor*(*props*) {

        super(*props*);

        //state의 초깃값 설정

        this.state = {

            number: 0,

        };

    }

state를 초기화해주는 생성자

render() {

*const* { number } = this.state;

        return (

            <*Fragment*>

                <h1>{number}</h1>

                <button

                    onClick={() *=>* {

                        this.setState({ number: number + 1 });

                        //setState를 이용해서 새로운 값을 넣을 수 있다.

                    }}

                >

                    +1

                </button>

            </*Fragment*>

        );

    }

버튼을 누를때마다 setState 메서드를 이용하여서 state를 업데이트 한다.

setState는 비동기로 작동

전체코드

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

*class* Counter extends Component {

*constructor*(*props*) {

        super(*props*);

        //state의 초깃값 설정

        this.state = {

            number: 0,

        };

    }

    render() {

*const* { number } = this.state;

        return (

            <*Fragment*>

                <h1>{number}</h1>

                <button

                    onClick={() *=>* {

                        this.setState({ number: number + 1 });

                        //setState를 이용해서 새로운 값을 넣을 수 있다.

                    }}

                >

                    +1

                </button>

            </*Fragment*>

        );

    }

}

export default Counter;

constructor에서 지정하지 않고 그냥 state={}으로 초기화 가능

state={

        number: 0

    }

this.setState({ number: number + 1 });

this.setState({ number: this.state.number + 1 });

이렇게 해도 동시에 2가 증가하지는 않는다.

왜냐하면 this.state.number가 첫번째 setState를 실행하면서 동시에 바뀌지는 않기 때문이다.

따라서 setState에 인자를 넘긴으로서 다양한 함수 구성을 구성할 수 있다.

onClick={() *=>* {

                        this.setState((*prevState*, *props*)*=>*({

                            number:  *prevState*.number + 1

                        }));

                        //()=>({객체}) 화살표 함수로 setState에 들어갈 객체를 그대로 반환한다.

                        //setState를 이용해서 새로운 값을 넣을 수 있다.

                    }}

또한 비동기식은 setState에서 동기식으로 작업을 처리하고 싶다면 두번째 인자로 callback을 걸면된다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

onClick={() *=>* {

                        this.setState((*prevState*, *props*) *=>* ({

                            number: *prevState*.number + 1,

                        }),

                        ()*=>*{

                            console.log("setState 종료됨");

                        });

                        //()=>({객체}) 화살표 함수로 setState에 들어갈 객체를 그대로 반환한다.

                        //setState를 이용해서 새로운 값을 넣을 수 있다.

                    }}

텍스트, 스크린샷, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명