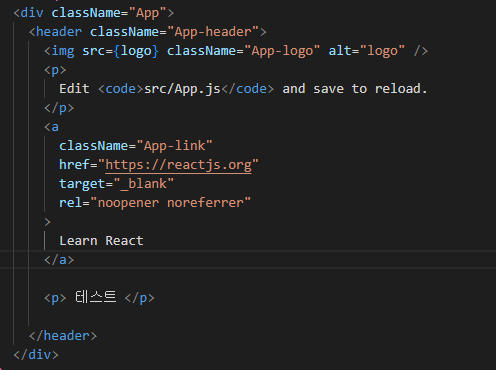
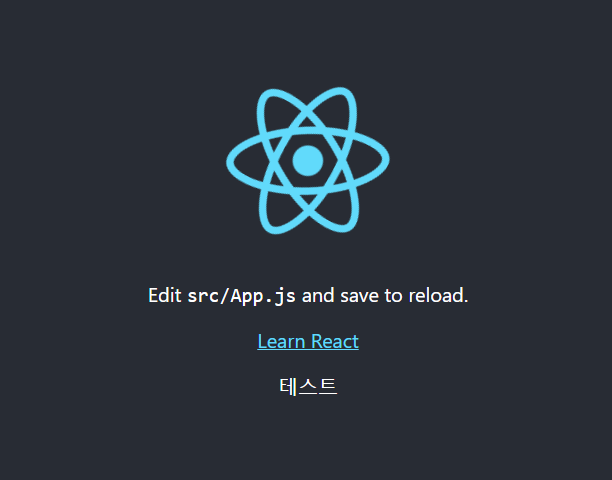
11.1

리액트

리액트에서 제공하는 라이브러리를 이용해서 만들어보자.

따로따로 component를 만들어서 하나로 모으는 형태로 진행되는 react



결국 하나하나가 컴포넌트가 됨.

데이터를 주고받아야 할 경우 props를 통해서 데이터를 주고 받음

**가상 DOM**을 이용하여 변화, 수정되는 부분만 찾아서 바뀌는 ‘DIFF‘가 주요 특징임.

(가상 DOM : 똑같이 복제를 해서 원본 dom에서 수정된 부분만 바꿔주는 역할을 함)

(페이스북 같이 사진의 용량이 많아 php형태에서는 시간이 오래 걸렸지만, react라이브러리를 통해 변화를 빠르게 적용할 수 있음)

리액트에서 쓰는 ‘**jsx 문법’**

- Babel

- 최신 문법을 하위 문법 (es6->IE) 에서 실행되도록 변환

- jsx로 작성된 코드를 순수한 js코드로 변환시켜줌

왜 jsx를 쓰는가? 가독성이 좋고, 컴파일 과정에서 오류 인지하기가 쉬움

Jsx 문법

● 최상위 요소는 하나

● 최상위 요소를 리턴하는 경우 ()로 감싼다

● 자식들을 바로 렌더링하고 싶으면 <>자식들</> 을 사용 => fragment

● 자바스크립트 표현식을 사용하려면 {표현식}을 이용

● if 문은 사용할 수 없다: 삼항연산자 혹은 && 이용

● style을 이용해 인라인 스타일 지정 가능

● 자식요소가 있으면 꼭 닫아야 하고 없으면 열어서 닫음

<p>hello</p>

<br />

// 예제만들기 react\src\App.js

- 단순히 선언만 해서 되면 안됨. 무조건 감싸는 애가 하나 있야어야함. (div등)

    // 1. 반드시 부모 요소가 있어야 함.

    // <>

    // <h1> 리액트 </h1>

    // <h2> 리액트 안녕 </h2>

    // </>

    //div 가 넣기 싫으면 그냥 <>만해서 감싸두기만 하면 됨 (fragment)

  /\* {}를 이용하여 js의 표현식을 사용할 수 있음. \*/

  return (

    <>

    <h1> {name} 리액트 </h1>

    <h2> 리액트 안녕 </h2>

    </>

);

      {name == '리액트' ? <h1>리액트 입니다</h1> : <h2> 아닙니다 </h2>}

&&,|| <🡨 얘좀 중요함



  /\* && , || 구분 \*/

  const name = 'undefined' // false

  return (

    <div>

      {name || '리액트'}

      {/\* name값이 undefined 이므로 리액트는 false, name이 나오게 됨 \*/}

      {name && '리액트'}

      {/\* 얘는 반대 \*/}

    </div>

  );

  /\* 리액트 스타일 적용 \*/

  const name = '리액트'

  /\* style안에 css 작성, 케밥형태 '-' 가 아니라 카멜 형태로 넣어줘야 한다 \*/

  const style = {

    backgroundColor : 'red',

    color : 'black',

    fontSize : '48px'

  }

  return (

    <div style = {style}> {/\* jsx에서 텍스트는 인라인 스타일로 넣을 수 있다 \*/}

      {name}

    </div>

  );

css파일 적용

  /\* 리액트 css 파일 적용 \*/

  const name = '리액튼='

  return (

    <div className = 'react'>

      {name}

    </div>

  /\* jss에서는 class가 아닌 className을 적용함으로 사용할 수 있다. \*/

  );

  /\* 닫는태그의 필요성 \*/

  const name = '리액트'

  return (

    <div className = 'react'> {name} </div>

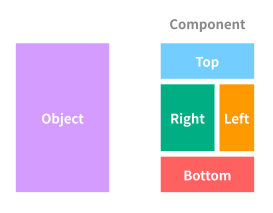
    <input></input> /\* input에 대해 닫는태그를 만들어 줘야 함 \*/

     <input /> /\* 이렇게도 사용가능 \*/

    );

  }

component



클래스컴포넌트, function컴포넌트 두개가 있다.

import React, {Component} from 'react';

import './App.css';

/\* class 컴포넌트 \*/

class App extends Component { /\* 1번줄, import를 사용하여 리액트에서 값을 가져온다 \*/

    render () {

      /\* 부모의 정의된 함수를 가져올 수 있는 override \*/

      const name = 'react';

      return <div> {name} </div>

      /\* 실질적으로 시작되는거는 index.js \*/

    }

  }

export default App;

//function MyComponent(){} es6버전

const MyComponent = () => {

    return <div> 새로운 컴포넌트 입니다. </div>

}

export default MyComponent

/\* 이걸 해줘야 다른 곳에서 import 해서 사용할 수 있다. \*/

import './App.css';

  /\* 아까 만든 myComponent를 불러와야함\*/

import MyComponent from './Mycomponent';

/\* function 컴포넌트 \*/

const App = () => {

  return <MyComponent></MyComponent>

  return <MyComponent />

}

export default App;

props – 컴포넌트끼리 데이터를 주고받게 해주는 뭐 아무튼 그런 데이터

const MyComponent = (props) => {

    return <div> 안녕 {props.name} 새로운 컴포넌트 입니다. </div>

}

export default MyComponent

import './App.css';

import MyComponent from './Mycomponent';

const App = () => {

  // return <MyComponent></MyComponent>

  return <MyComponent name = 'React' />

  /\* name이라는 값인 props로 React를 넣어줌. \*/

}

 /\* prop를 통해 상위 컴포넌트에서 하위 컴포넌트의 자료를 받을 수 있어짐. \*/

export default App;

MyComponent.defaultProps = {

    name : '기본이름'

    // App.js에 name이 지정되어 있지 않아도, defaultprops가 있기에 기본이름이 나가게 됨.

}

const MyComponent = (props) => {

    /\* children 속성 \*/

    /\* 리턴값이 두줄 이상일 때는 () 꼭 붙여줘야함!!!!!!!!!! \*/

    return (

    <div>

    <div> 안녕 {props.name} 새로운 컴포넌트 입니다. </div> <br />

        칠드런 값은 {props.children} 입니다.

    </div>

    /\* .children component사이의 값을 가져올 수 있어짐. \*/

    )

}

const App = () => {

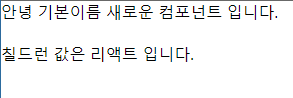
  // return <MyComponent></MyComponent>

  return <MyComponent>

    리액트

  </MyComponent>

}



**비 구조화 할당**

const MyComponent = ({name, children}) => { /\* props를 지정하지 않고 바로 할 수도 있음 \*/

const MyComponent = (props) => {

    /\* 비 구조화 할당!!!!!!! \*/

    const {name, children} = props

    /\* prop에 가져온 것을 각각 name, cildren에 넣을 수 있는 \*/

    return (

    <div>

    <div> 안녕 {name} 새로운 컴포넌트 입니다. </div> <br />

        칠드런 값은 {children} 입니다.

    </div>

    )

}

const MyComponent = ({name, children}) => {

속성값 검사

const App = () => {

  // return <MyComponent></MyComponent>

  return <MyComponent name={3}>

  {/\* name에 숫자를 줬을 경우 (참고로 숫자는 {} 안에 줌) \*/}

    리액트

  </MyComponent>

}

import PropTypes from 'prop-types' /\* 먼저 선언해줘야함 \*/

const MyComponent = ({name, children}) => {

    // const {name, children} = props

   return (

    <div>

    <div> 안녕 {name} 새로운 컴포넌트 입니다. </div> <br />

        칠드런 값은 {children} 입니다.

    </div>

    )

}

MyComponent.defaultProps = {

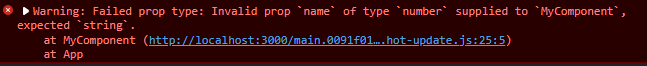
    name : '기본이름'

}

MyComponent.propTypes = {

    name : PropTypes.string /\* 타입이 스트링인지 검사 \*/

}



import PropTypes from 'prop-types'

import {Component} from 'react'

    /\* static, class의 extends 활용한 방법 \*/

class MyComponent extends Component{

    static defaultProps = {

        name : '기본이름'

    }

    static propTypes = {

        name : PropTypes.string,

        forNumber : PropTypes.number.isRequired /\* 숫자 필수값이 있는지를 확인 \*/

    }

    render () {

       const {name, forNumber, children} = this.props

       return (

        <div>

        <div> 안녕 {name} 새로운 컴포넌트 입니다. </div> <br />

            칠드런 값은 {children} 입니다.<br />

            좋아하는 숫자는 {forNumber} 입니다.

        </div>

        )

    }

}

State 컴포넌트 내부에서 변경할 수 있는 데이터

import React, { Component } from 'react';

class Counter extends Component {

    constructor(props){

        super(props) //component의 생성자 함수를 호출해줌

        //state 초기화

        this.state = {

            number : 0

        }

    }

    /\* Counter 객체 실행,

        this,state 에서 0 초기화

        render 실행되며서 나머지 쭉 실행

        애초에 0으로 초기화 되어 있기 때문에, number은 0이 되어있음

    \*/

    render() {

        const {number} = this.state /\* 넘버는 0이 될 것임 \*/

        return (

            <div>

                <h1> {number} </h1>

                <button onClick={()=>{

                    /\* state 변경 방법 \*/

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

                }}>+1</button>

            </div>

        );

    }

}

export default Counter;

import React, { Component } from 'react';

import './App.css';

import Counter from './Counter';

const App = () => {

  return <Counter />

}

export default App;

// 이전것을 의미하는 prevstate도 있다

// setState 3총사

    render() {

        const {number,number1,fixedNumber} = this.state

        return (

            <div>

                <h1> {number} </h1>

                <h2> 고정 {fixedNumber} </h2>

                <button onClick={()=>{

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

                    this.setState((prevState)=>{

                        return {number : number + 1}

                    })

                }}>+1</button>

                <button onClick={()=>{

                    this.setState({number1 : number1 -1})

                }}>-1</button>

            </div>

        );

    }

                <h1> {number} </h1>

                <h2> 고정 {fixedNumber} </h2>

                <button onClick={()=>{

                    this.setState({

                        number : number+1

                    }, ()=>{

                        console.log(`state가 호출되었습니다`);

                        console.log(this.state);

                    })

Hooks

클래스보다 function을 많이 사용하기 때문에

useState 이용한 예시

    /\* HOOK \*/

        const [message,setMessage] = useState('DEFAULT')

        /\*

            this.state = {

            message = ''

        }

            이것과 동일한 상태임

        \*/

        /\*state에 있는 setMessage를 업데이트 해줌 \*/

        const onClickEnter = () => {return setMessage('안녕하세요')}

        const onClickLeave = () => {return setMessage('안녕히가세요')}

        return (

            <div>

                <button onClick={onClickEnter} > 입장 </button>

                <button onClick={onClickLeave} > 퇴장 </button>

                <h1>{message}</h1>

                {/\* 다시! 렌더링이 된다 \*/}

            </div>

        );

        // 1. useState 사용

        /\*  const [초기값, 업데이트 함수] = useState(초기값) \*/

        // 2. useState 사용

        /\*  업데이트 함수 (업데이트 원하는 값) \*/

        const [color,setcolor] = useState('black')

        return (

            <div>

                <button onClick={onClickEnter} > 입장 </button>

                <button onClick={onClickLeave} > 퇴장 </button>

                <h1 style={{color}}>{message}</h1>

                <button style={{color:'red'}} onClick={()=>{setcolor('red')}}> 빨 </button>

                <button style={{color:'green'}} onClick={()=>{setcolor('green')}}> 초 </button>

                <button style={{color:'blue'}} onClick={()=>{setcolor('blue')}}> 파 </button>

                {/\* 버튼을 눌렀을 때 setcolor의 값을 바꿔서 색을 바꿔준다.

                이때, onclick에 function을 적용해서 그 안에 setcolor을 넣어준다 \*/}

            </div>

        );