0. 컴포넌트?

특정 코드 뭉치를 다른 부분에 이식하거나 재사용하기 위해 사용하는 코드 블록 단위

파일 단위로 작성한 후 필요한 위치에서 임포트해 사용할 수 있다.

1. component 사용원리?

첫 번째 줄에서 import한 {component}를 상속받아 R003 클래스(새로만든js파일)에서 사용

return된 html코드를 render()함수를 사용해 화면에 표시

App.js 에서 import해서 사용가능

2. 생명주기?

component의 생성, 변경, 소멸 과정

생성 : constructor() getDerivedStateFormProps() render() componentDidMount()

3. render()?

render()는 return되는 html형식의 코드를 화면에 그려주는 함수

화면 내용이 변경돼야 할 시점에 자동으로 호출

4. constructor(props)

생명주기 함수 중 가장 먼저 실행되며, 처음 한번만 호출

component 내부에서 사용되는 변수(state)를 선언하고

부모 객체에서 전달받은 변수(props)를 초기화할 때 사용한다.

super()함수는 가장 위에 호출해야 한다

5. getDerivedStateFromProps(props, state)

constructor() 다음으로 실행된다.

component가 새로운 props를 받게 됐을 때 state를 변경해준다.

App.js에서 전달한 prop\_value라는 변수를 props.prop\_value로 접근해 값을 가져올 수 있다.

6. componentDidMount()

작성한 함수들 중 가장 마지막으로 실행된다.

render()가 return 되는 html 형식의 코드를 화면에 그려진 후 실행된다.

화면이 그려진 후에 실행돼야하는 이벤트 처리, 초기화 등에서 가장 많이 활용되는 함수다.

7. setState()

변수의 선언과 초기화를 동시에 실행한다.

8. shouldComponentUpdate(props,state)

setState()로 인해서 state의 변경이 발생했기 떄문에 ‘변경’단계의 생명주기 함수 shouldComponentUpdate()가 실행된다.

Boolean 유형의 데이터를 반환하는데 return이 true일 경우 render()를 한 번 더 호출한다.

NOTE

함수의 반환값에 따라 render() 재실행 가능한다는 점을 이용

Props나 state변수가 변경될 때 화면을 다시 그리며 제어할 수 있다.

* 직접 해보니 componentDidMount까지 왔다가 constructor 건너뛰고 getDerived 나오고 render 나옴 새로운 getDerived 나오고부터 this.state가 변경됨

9. ES는 표준화된 스크립트 언어

ES 뒤에 붙은 숫자는 버전

파이썬에서의 f-string == js에서의 백틱+${변수}

f’hello {name}’ == `hello ${name}`

아래는 파이썬과 동일함

startswith

endswith

파이썬의object in list == includes

arr.some도 있음

Object not in list

10. ES5에서 사용하던 var는 변수를 재선언 재할당할 수 있다.

=>변수의 사용범위가 불확실해지거나 의도하지 않은 변숫값 변경 발생

ES6에서 단점을 보완하기 위해서 let const 추가

let은 재할당 허용

const 재선언 재할당 허용하지 않음

11~16

11. 전개 연산자 사용하기

배열이나 객체 변수를 직관적이고 편리하게 합치거나 추출할 수 있게 도와주는 문법

변수 앞에 …(마침표3개)

//js array

        var varArray1 = ['num1','num2'];

        var varArray2 = ['num3','num4'];

        var sumVarArr = [varArray1[0],varArray1[1],varArray2[0],varArray2[1]];

        // var sumVarArr = [].concat(varArray1,varArray2);

        console.log(`1. sumVarArr : ${sumVarArr}`);

        // ES6 Array

        let sumLetArr = [...varArray1,...varArray2];

        console.log('2. sumLetArr : '+sumLetArr);

        const [sum1,sum2,...remain] = sumLetArr;

        console.log('3. sum1 : '+sum1+', sum2 : '+sum2+', remain : '+remain);

        var varObj1 = {key1 : 'val1',key2:'val2'};

        var varObj2 = {key2:'new2',key3:'val3'};

        // javascript Object

        var sumVarObj = Object.assign({},varObj1,varObj2);

        console.log('4. sumVarObj : '+JSON.stringify(sumVarObj));

        // ES6 Object

        var sumLetObj = {...varObj1,...varObj2};

        console.log('5. sumLetObj : '+JSON.stringify(sumLetObj));

        var {key1,key3,...others} = sumLetObj;

        console.log('6. key1 : '+key1+', key3 : '+key3+', others : '+JSON.stringify(others));

12. class

ES5 prototype과 동일

JS는 함수를 객체로 사용할 수 있다

200제 51페이지 직접 보는게 좋을 듯

13. 화살표 함수

Function 대신 => 사용

This로 component의 state 변수에 접근해 사용할 수 있다.

Return 문자열 생략, this를 bind해야하는 문제도 생기지 않음

콜백함수내부에서는 this가 window객체이기 때문에 this로 state변수에 접근하려면 에러발생

.bind(this) 해주면 component 사용가능

14. forEach() 함수

배열의 처음부터 마지막순번까지 모두 작업하는 경우 forEach()문 쓰는 것이 간편

        ForEach\_Arr.forEach((result)=>{

            ForEach\_newArr.push(result);

        })

순번과 배열의 크기 정보를 사용하지 않음

0부터 배열의 크기만큼 반복하여 순서대로 배열 값 반환

반복문이 실행될 때마다 콜백 함수로 result를 받아 ForEach\_newArr에 넣는다

15. map()

forEach()와 마찬가지로 for문에서 사용하던 순번과 배열의 크기 변수를 사용하지 않음

차이점은 forEach()와 달리return을 사용해 반환값을 받을 수 있음

화살표 함수는 return 구문을 생략할 수 있다

Map\_Arr.map(x=>x) 동일Map\_Arr.map(x=>{return x})

파이썬 map이랑 유사한듯

16. jQuery

    input\_alert = (e) => {

        var input\_val = $('#inputId').val();

        alert(input\_val);

    }

        <input id = "inputId" name = "inputName"/>

        <button id = "buttonId" onClick={e => this.input\_alert(e)}>

         jQuery Button

        </button>

제이쿼리 접근할 때$붙여서 사용한다 id가 inputId인 <input>태그에 접근

input태그의 value값을 input\_val에 저장

17~

17. props 사용하기

Props 부모 컴포넌트가 자식 컴포넌트에 데이터를 전달할 때 사용

Props를 전달받은 자식 컴포넌트에서는 데이터를 수정할 수 없다.

데이터를 변경하기 위해서는 컴포넌트 내부에서만 사용하는 변수에 값을 넣어 사용

App.js

<Props props\_val = "THIS IS PROPS"/>

R017\_Props.js

        let props\_value = this.props.props\_val;

        props\_value += ' from App.js'

this.props.뒤에 App.js에서 전달받은 props 변수명을 붙이면 해당 데이터 사용가능

데이터 수정해야 할 경우, props 자체가 아닌 컴포넌트 내부 변수(props\_value)에 옮겨 가공한다.

App.js에서 넘겨받은 문자열 “THIS IS PROPS” + “from App.js”가 출력됨

즉, 자식 컴포넌트에서 수정된 것이 출력됐다.

18. props 자료형 선언하기

import datatype from 'prop-types';

R018\_PropsDatatype.propTypes = {

    String: datatype.number,

propTypes가 데이터 유효성 검증을하고 콘솔에 경고 메시지 출력

19. props Boolean으로 사용하기

하위 컴포넌트에 true/false 할당

Props 변수를 선언하고 값을 안주면 true가 기본값으로 할당됨

App.js

        <PropsBoolean BooleanTrueFalse = {false}/>

        <PropsBoolean BooleanTrueFalse/>

R019\_PropsBoolean.js

            <div style={{padding:"0px"}}>

                {BooleanTrueFalse ? '2. ' : '1. '}

                {BooleanTrueFalse.toString()}

            </div>

결과 => 1. False 2. True

20. props 객체형으로 사용하기

import datatype from 'prop-types';

    render() {

        let{

            ObjectJson

        } = this.props

        return (

            <div style={{padding:"0px"}}>

                {JSON.stringify(ObjectJson)}

            </div>

        )

    }

}

R020\_PropsObjVal.propTypes = {

    ObjectJson : datatype.shape({

        react:datatype.string,

        twohundred : datatype.number

    })

}

하위 컴포넌트들의 변수의 자료형을 선언할 때는 shape유형 사용 + prop-types

21. props를 필수 값으로 사용하기

props의 자료형 선언할 때prop-types

자료형 설정 대신 isRequired를 조건으로 추가하면 변숫값이 없는 경우 경고메시지 발생

App.js에서 props를 무조건 받아야하는 변수

R021\_PropsRequired.propTypes = {

    ReactString : datatype.isRequired,

}

22. props를 기본값으로 정의하기

Props의 기본값은 부모 컴포넌트에서 값이 넘어 오지 않았을 때 사용한다.

defaultProps라는 문법 사용

App.js

<PropsDefault ReactNumber = {200}/>

R022\_PropsDefault.js

R022\_PropsDefault.defaultProps = {

    ReactString : "리액트",

    ReactNumber : 400

}

23. props의 자식 컴포넌트에 node 전달하기

props를 하위 컴포넌트 태그 안쪽에 선언해 전달하는 것 이외에도 하위 컴포넌트 태그 사이에 작성된 node를 전달 할 수 있다. <PropsNode>여기html 어쩌고..</PropsNode>

Node란

* Html 문서를 구성하는 포괄적인 개념이다. 노드의 종류에는 문서 요소, 속성, 텍스트, 주석 등이 있다. <p id=’a’>bc</p>에서 <p>태그가 요소, id가 속성,bc가 텍스트

App.js

        <PropsNode>

          <span>node from App.js</span>

        </PropsNode>

R023\_PropsNode

            <div style={{padding:"0px"}}>

                {this.props.children}

            </div>

24. state 사용하기

props는 상위 컴포넌트에서 하위 컴포넌트로 데이터를 전달할 때 사용

state는 하나의 컴포넌트 안에서 전역 변수처럼 사용 global

    constructor(props){

        super(props);

        this.state = {

            StateString : this.props.reactString,

            StateNumber : 200,

        }

    }

    render() {

        return (

            <div style={{padding:"0px"}}>

                {this.state.StateString}{this.state.StateNumber}

            </div>

        )

    }

}

가장 먼저 실행되는 생성자 함수 constructor 안에서 state 변수의 초깃값을 정의해야한다.

this.state.변수명 문법으로 state변수에 접근

25. setState() 함수 사용하기

this.state.변수명 = value와 같이 state를 직접 변경하면 render()함수를 호출하지 않으므로

화면에 보이는 state 값은 바뀌기 전 상태로 남게 된다.

setState() 함수로 state를 변경해야 render()함수를 호출해 변경된 값을 화면에 보여줄 수 있다