함수 컴포넌트에서 state 사용

useState

<배열 비구조화 할당>

*const* array = [1,2];

*const* one = array[0];

*const* two = array[1];

*const* [one, two] = array;

원래 위와 같이 하는게 일반적이지만

비구조화 할당에서는 아래와 같이 변수를 요소 안에 구성하여 바로 배열의 값을 연결할 수 있다.

함수에서 usetState는 여러번 선언해서 사용할 수 있다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

*const* Say = () *=>* {

*const* [message, setMessage] = useState('');

*const* onClickEnter = () *=>* {setMessage('Hello')};

*const* onClickLeave = () *=>* {setMessage('Bye')};

*const* [color, setColor] = useState('black');

    return (

        <div>

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

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

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

            <br></br>

            <button style={{color: 'red'}} onClick={()*=>*{setColor('red')}}>Red</button>

            <button style={{color: 'green'}} onClick={()*=>*{setColor('green')}}>Red</button>

            <button style={{color: 'blue'}} onClick={()*=>*{setColor('blue')}}>Red</button>

        </div>

    );

};

이러한 State 에서 배열이나 객체를 업데이트하기 위해서는 사본을 만들고 다시 useState로 복사하는 과정을 거쳐야한다.

4. 이벤트 핸들링

4-1. 리액트의 이벤트 시스템

원래 html에서 onClick을 alert를 했던 과정을 react에선 어떻게 하는지를 배우는 타임

<이벤트 사용시 주의사항>

1. 이벤트 이름은 카멜 표기법으로 작성합니다
2. 이벤트에 실행할 자바스크립트 코드를 전달하는 것이 아니라, 함수 형태의 값을 전달합니다.
3. DOM요소에만 이벤트를 설정할 수 있습니다.  
   (직접 만든 컴포넌트에 onClick을 넘기는 것은 props을 넘긴 것이지 이벤트가 발생하지 않음, 태그 요소에만 적용할 수 있음)

4.1.2 이벤트 종류

clipboard

composition

keyboard

focus

form

mouse

selection

touch

ui

wheel

media

image

animation

transition

4-2. 예제로 이벤트 핸들링 익히기

input 태그에 대해서 onChange시 e(이벤트 객체)의 구성은

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

*class* EventPractice extends Component {

    render() {

        return (

            <div>

                <h1>Event Practice</h1>

                <input

                    type="text"

                    name="message"

                    placeholder="Say anything"

                    onChange={(*e*) *=>* {

                        console.log(*e*);

                    }}

                />

            </div>

        );

    }

}

e.target.value를 log하면

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명 값이 바뀔 때 마다 그 때의 value가 기록됨

onChage할때 state의 값을 update하고 button을 클릭할 때 state.message를 ‘’로 초기화 하면서 alert로 현재 message를 알려줌

*class* EventPractice extends Component {

    state = {

        message: ''

    }

    render() {

        return (

            <div>

                <h1>Event Practice</h1>

                <input

                    type="text"

                    name="message"

                    placeholder="Say anything"

                    value={this.state.message}

                    onChange={(*e*) *=>* {

                        this.setState({

                            message: *e*.target.value

                        })

                    }}

                />

                <button onClick={()*=>*{

                    alert(this.state.message);

                    this.setState({

                        message: ''

                    })

                }}>

                    confirmed

                </button>

            </div>

        );

    }

}

이때 임의 메서드를 생성해서 onChange나 onClick 에 해당하는 함수를 뽑아서 작성 가능하다.

*constructor*(*props*) {

        super(*props*);

        this.handleChange = this.handleChange.bind(this);

        this.handleClick = this.handleClick.bind(this);

    }

    handleChange(*e*) {

        this.setState({

            message: *e*.target.value,

        });

    }

    handleClick() {

        alert(this.state.message);

        this.setState({

            message: '',

        });

    }

함수 설정하고 event callback에 걸면됨

이때, constructor에서 this.함수를 this에 binding해줘야 해당 태그에서 this에 정확히 접근가능

property initializer Syntax를 사용한 메서드 작성

    handleChange = (*e*) *=>*{

        this.setState({

            message: *e*.target.value,

        });

    }

    //호출시

    this.handleChange

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

state에 userName을 추가하고

input을 하나 더 추가하면

<input

                    type='text'

                    name='userName'

                    placeholder='User Name'

                    value={this.state.userName}

                    onChange={this.handleChange}

                />

handleChange를 아래와 같이 작성하여 한번에 해결 할 수 있다.

handleChange(*e*) {

        this.setState({

            [*e*.target.name]: *e*.target.value

            //input name에 설정한 state명을 불러와서 그 값의 변화일때 그 key에 value를 저장

        });

    }

enter입력시 event 핸들하는 방법은 input에

onKeyPress={this.handleKeyPressed}

달고,

keypresshandler를 선언하면 됨

handleKeyPressed = (*e*) *=>* {

        if(*e*.key === 'Enter'){

            this.handleClick();

        }

    }

4-3. 함수를 통해서도 useState로 구성가능

4-4. 정리

JS 또는 jQuerry를 사용한 웹 app 이벤트와 비슷하다.

5. ref:DOM에 이름 달기

html에선 DOM요소에 이름을 달 때는 id를 활용한다.

<div id=”desc”>desc</div>

id는 html에서 유일해야하지만, ref는 컴포넌트 내부에서 작도하기 때문에 중복 문제에서 조금은 자유롭다.

ref는 react사용하는 naming 기법이다.

5-1. ref는 어떤 상황에서 사용해야 할까?

정답: DOM을 꼭 직접적으로 건드려야 할 때,

일단 예제로 password를 state를 활용해서 validation하는 과정까지 구성

import React, { Component } from 'react';

import './ValidationSample.css';

*class* ValidationSample extends Component {

    state = {

        password: '',

        clicked: false,

        validated: false,

    };

    handleChange = (*e*) *=>* {

        this.setState({

            password: *e*.target.value,

        });

    };

    handleclick = () *=>* {

        this.setState({

            clicked: true,

            validated: this.state.password === '0000'

        })

    };

    render() {

        return (

            <div>

                <input

                    type="password"

                    value={this.state.password}

                    onChange={this.handleChange}

                    className={

                        this.state.clicked ? (this.state.validated ? 'success' : 'failure') : ''

                    }

                />

                <button onClick={this.handleclick}>verify</button>

            </div>

        );

    }

}

export default ValidationSample;



5-2. ref 사용

ref를 만드는 방법은 콜백 함수를 사용하는 것이다.

<input ref={(*ref*)*=>*{this.input = *ref*}} />

이렇게 하면 this,input은 input 요소의 DOM을 가리킨다.

creatRef를 통한 ref 설정

input = React.createRef();

<input ref={this.input} />

validation코드에서 button 클릭시 다시 input으로 focus되도록 설정

import React, { Component } from 'react';

import './ValidationSample.css';

*class* ValidationSample extends Component {

    state = {

        password: '',

        clicked: false,

        validated: false,

    };

    handleChange = (*e*) *=>* {

        this.setState({

            password: *e*.target.value,

        });

    };

    handleclick = () *=>* {

        this.setState({

            clicked: true,

            validated: this.state.password === '0000',

        });

        this.pwd.focus();

    };

    render() {

        return (

            <div>

                <input

                    ref={(*ref*) *=>* {

                        this.pwd = *ref*;

                    }}

                    type="password"

                    value={this.state.password}

                    onChange={this.handleChange}

                    className={

                        this.state.clicked ? (this.state.validated ? 'success' : 'failure') : ''

                    }

                />

                <button onClick={this.handleclick}>verify</button>

            </div>

        );

    }

}

export default ValidationSample;

방금 같이 요소가 component에도 ref를 달 수도 있다.

component에 설정한 ref는 그 component에 있는 메서드와 properties를 둘 다 사용할 수 있다.

App.js

import './App.css';

import ScrollBox from './ScrollBox';

import React, { Component } from 'react';

*class* App extends Component {

  render() {

    return (

      <div>

            <*ScrollBox*

              ref={(*ref*)*=>* this.scrollBox=*ref*}

            />

            <button onClick={()*=>* this.scrollBox.scrollToBottom()}>

              to the Bottom

            </button>

        </div>

    );

  }

}

export default App;

ScrollBox.js

import React, { Component } from 'react';

*class* ScrollBox extends Component {

    scrollToBottom = ()*=>*{

*const* {scrollHeight, clientHeight} = this.box;

        this.box.scrollTop = scrollHeight - clientHeight;

    }

    render() {

*const* style = {

            border : '1px solid black',

            height: '300px',

            width: '300px',

            overflow: 'auto',

            position: 'relative'

        }

*const* innerStyle = {

            width: '100%',

            height: '650px',

            background: 'linear-gradient(#e66465, #9198e5)'

        }

        return (

            <div style={style}

            ref={(*ref*)*=>*{this.box=*ref*}}>

                <div style={innerStyle}/>

            </div>

        );

    }

}

export default ScrollBox;

6. 컴포넌트 반복

*class* IterationSample extends Component {

    render() {

        return (

            <ul>

                <li>Snowman</li>

                <li>Ice</li>

                <li>snow</li>

                <li>Wind</li>

            </ul>

        );

    }

}

이것과 같이 list에서 반복을 불필요하기 작성하지 않고 빠르게 할수 있는 방법이 있다.

6-1. 자바스크립트 배열의 map()함수

JS에서 map은 다음과 같은 구성을 가짐

arr.map( callback, [thisArg])

* callback: 새로운 배열 요소를 생성하는 함수  
  (파라미터: currentValue / index / array)
* thisArg(선택항목) : callback 함수 내부에서 사용할 this 레버런스
* *let* numbers = [1,2,3,4,];
* *let* processed = numbers.map((*num*)*=>num*\**num*)
* console.log(processed);

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

6-2. 데이터 배열을 컴포넌트 배열로 변환하기

<li></li>를 붙여주는 map함수를 생성하고

return으로 <ul>{namesMapped}</ul>을 하여 반복적인 li 작업을 배열과 map 메소드를 이용하여 편리하게 처리한다.

import React, { Component } from 'react';

*class* IterationSample extends Component {

    render() {

*const* names = ['Snowmane', 'Ice', 'Snow', 'Wind'];

*const* namesMapped = names.map((*names*)*=>* <li>{*names*}</li>)

        return (

            <div>

                <ul>

                    {namesMapped}

                </ul>

            </div>

        );

    }

}

export default IterationSample;

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명

하지만 이때 key값이 없는 warning이 뜬다.

이는 요소가 변경되었을 때 어떤값이 변경되었는지 알기 쉽게하기 위해서 key가 필요한 것인데

텍스트, 폰트, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

따라서 배열에서는 index를 키값으로 전달하면 된다.

*const* namesMapped = names.map((*names*, *index*)*=>* <li key={*index*}>{*names*}</li>)

6-4. 응용

list를 map으로 표시하고

li를 추가하고, li를 제거할 수 있는 기능 구현

*const* [name, setName] = useState([

        { id: '1', text: 'Snowman' },

        { id: '2', text: 'Ice' },

        { id: '3', text: 'Snow' },

        { id: '4', text: 'Wind' },

    ]);

*const* [inputText, setInputText] = useState('');

*const* [nextId, setNextId] = useState(5); //new array adding index

사용할 다양한 state들

*const* onChange = (*e*) *=>* setInputText(*e*.target.value);

*const* onClick = ()*=>*{

*const* nextNames = name.concat({

            id: nextId,

            text: inputText

        });

        //nextNames에 새로운 id와 text를 가지는 배열을 만듬

        setNextId(nextId+1);

        setName(nextNames);

        //배열 갈아낌

        setInputText('');

    }

이벤트 handler

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

*const* remove = *id* *=>* {

*const* newNames = name.filter(*name* *=>* *name*.id !== *id*);

        setNextId(nextId-1);

        setName(newNames);

        setRemvoingText('');

    }

*const* nameList = name.map((*name*) *=>* <li key={*name*.id} onDoubleClick={()*=>*{remove(*name*.id)}}>{*name*.text}</li>);

li자체를 doubleClick 했을 때, remove함수로 그 li의 id를 보내서

그 id에 해당하는 배열만 filter해서 다시 배열을 만들고 그 배열로 setName을 해줘서 delete 기능을 구성