

# React - 3, 4장

🕒 생성일	@2022년 5월 6일 오전 11:23
▼ 분류	React-리액트를 다루는 기술 책
☰ 주제	React
📅 날짜	

## 컴포넌트

- 데이터가 주어졌을때 이에 맞춰 UI를 만들어주고  
함수형에서는 Hooks, 클래스에서는 라이프사이클 API를 이용해 컴포넌트가 화면에서 나타나고 사라지거나 변화가 일어날때 주어진 작업들을 처리할수 있으며  
임의 메서드를 만들어 특별한 기능을 붙여줄수 있다.

## 함수형 컴포넌트

- 클래스형 컴포넌트보다 선언하기 편하고 메모리 자원도 덜 사용함
- 프로젝트를 완성해 빌드한 후 배포할때 결과물의 파일 크기가 클래스형 컴포넌트보다 작음
- state와 라이프사이클API를 사용하지 못하는 단점이 있었으나 리액트 v16.8 업데이트 이후 Hooks기능이 도입되면서 해결되었음
- 리액트 공식 메뉴얼에서는 컴포넌트를 새로 작성할때 함수형 컴포넌트와 Hooks 사용을 권장하고 있음
- 선언할때 function키워드 사용하는것과 화살표함수를 사용하는것에는 큰 차이가 없음

## prototype과 class

- prototype

```
function Dog(name){  
  this.name= name;  
}
```

```
Dog.prototype.say= function(){
  console.log(this.name+ ' : 멍멍!');
}
var dog= new Dog('검둥이');
dog.say();
//=> 검둥이 : 멍멍!
```

## • class

```
class Dog{
  constructor(name){
    this.name= name;
  }
  say(){
    console.log(this.name+ ' : 왈왈!');
  }
}
const dog= new Dog('흰둥이');
dog.say();
//=> 흰둥이 : 왈왈!
```

# 모듈

## • 내보내기- export

- export default 컴포넌트이름;
- 다른파일에서 이 파일을 import할때, 위에 선언한 컴포넌트를 불러오도록 설정함

## • 불러오기- import

- import구문을 사용하는 두번째 줄은 만든 컴포넌트를 불러옴
- 함수형과 클래스형 모두 사용 방법 동일

```
import 컴포넌트이름 from '컴포넌트파일경로';
const App=() =>{
  return(<컴포넌트이름/>);
};
export default App;
```

## Poros

- 컴포넌트 속성을 설정할때 사용하는 요소
  - props값은 해당 컴포넌트를 불러와 사용하는 부모 컴포넌트에서 설정할수 있음
    - ex) App.js가 부모 컴포넌트
- 

## JSX내부에서 props 렌더링

```
import './App.css';
import MyComponent from './MyComponent';
function App() {
  return (
    <div class="react">
      <MyComponent name='React' />
    </div>
  );
}
export default App;
```

```
src > JS MyComponent.js > [⌕] default
1  import React from 'react';
2
3  const MyComponent = props => {
4    return (
5      <div>
6        Hi, Everyone!!!:) {props.name}
7      </div>
8    );
9  };
10
11 export default MyComponent;
```

출력 결과



Hi, Everyone!!!:) React

---

## props 기본값 설정- defaultProps

```
import './App.css';
import MyComponent from './MyComponent';
function App() {
  return (
    <div class="react">
      <MyComponent/>
    </div>
  );
}
export default App;
```

```
rc > JS MyComponent.js > ...
1 import React from 'react';
2
3 const MyComponent = props => {
4   return (
5     <div>
6       Hi, Everyone!!!:) {props.name}
7     </div>
8   );
9 };
10
11 MyComponent.defaultProps = {
12   name: 'none'
13 };
14
15 export default MyComponent;
```

출력 결과

Hi, Everyone!!!:) none

## 태그사이의 내용- children

- 컴포넌트 태그 사이의 내용을 보여줌

```
1 import './App.css';
2 import MyComponent from './MyComponent';
3 function App() {
4   return (
5     <div class="react">
6       <MyComponent>REACT</MyComponent>
7     </div>
8   );
9 }
10 export default App;
```

```
myComponent.js > <!-- default
import React from 'react';

const MyComponent = props => {
  return (
    <div>
      Hi, Everyone!!!:) {props.name}
      <br />
      children:: {props.children}
    </div>
  );
};

MyComponent.defaultProps = {
  name: 'none'
};

export default MyComponent;
```

출력 결과

Hi, Everyone!!!:) none  
children:: REACT

## 비구조 할당으로 props 내부값 추출

- 배열, 객체 안에 들어있는 값을 쉽게 추출
- 아래 두 코드들은 같은 결과

```
const array= [1, 2];  
const one= array[0];  
const two= array[1];
```

```
const array= [1, 2];  
const [one, two]= array;
```

출력 결과는 같다

Hi, Everyone!!!:) none  
children:: REACT

```
import './App.css';  
import MyComponent from './MyComponent';  
function App() {  
  return (  
    <div className="react">  
      <MyComponent name={1}>REACT</MyComponent>  
    </div>  
  );  
}  
  
export default App;
```

```

1 import React from 'react';
2
3 const MyComponent = props => {
4   const {name, children}= props;
5   return (
6     <div>
7       Hi, Everyone!!!:) {name}
8       <br />
9       children:: {children}
10    </div>
11  );
12 };
13
14 MyComponent.defaultProps = {
15   name: 'none'
16 };
17
18 export default MyComponent;

```

- 함수의 파라미터가 객체라면 그값을 바로 비구조화해 사용

```

import React from 'react';

const MyComponent = ({name, children}) => {
  return (
    <div>
      Hi, Everyone!!!:) {name}
      <br />
      children:: {children}
    </div>
  );
};

MyComponent.defaultProps = {
  name: 'none'
};

export default MyComponent;

```

## propTypes

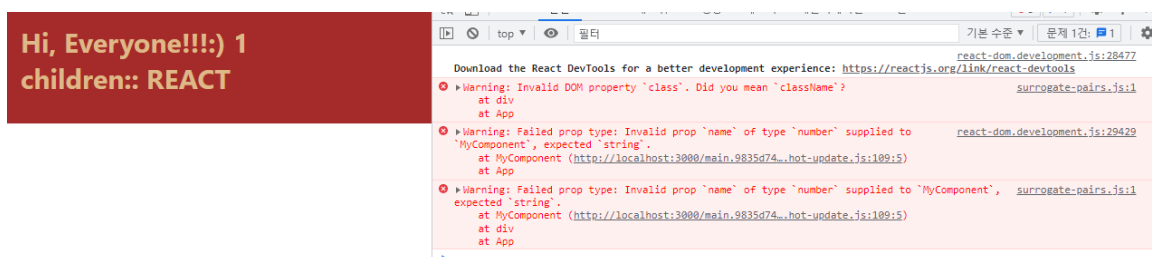
- 컴포넌트의 필수 props를 지정하거나 props의 타입을 지정할때 사용
- import PropTypes from 'prop-types';

```

c > JS MyComponent.js > ...
1  import React from 'react';
2  import PropTypes from 'prop-types';
3
4  const MyComponent = ({name, children}) => {
5    return (
6      <div>
7        Hi, Everyone!!!:) {name}
8        <br />
9        children:: {children}
10     </div>
11   );
12 };
13
14 MyComponent.defaultProps = {
15   name: 'none'
16 };
17 MyComponent.propTypes = {
18   name: PropTypes.string
19 };
20
21 export default MyComponent;

```

- propType을 잘못 지정했을경우 - 화면에 출력은 됨



## 필수 propTypes설정- isRequired

- propTypes을 지정하지 않았을때 경고 메시지를 줌

- propTypes을 지정할때 뒤에 isRequired를 붙임

```
1 import './App.css';
2 import MyComponent from './MyComponent';
3 function App() {
4
5   return (
6     <div className="react">
7       <MyComponent name={1}>REACT</MyComponent>
8     </div>
9   );
10 }
11
12 export default App;
13
```

```
1 import React from 'react';
2 import PropTypes from 'prop-types';
3
4 const MyComponent = ({name, children,num}) => {
5   return (
6     <div>
7       Hi, Everyone!!!:) {name}
8       <br />
9       children:: {children}
10      <br />
11      숫자:: {num}
12    </div>
13  );
14
15  MyComponent.defaultProps = {
16    name: 'none'
17  };
18  MyComponent.propTypes = {
19    name: PropTypes.string,
20    num: PropTypes.number.isRequired
21  };
22
23 export default MyComponent;
24
```

- 필수값을 지정하지 않았을 경우 뜨는 경고 메세지

Hi, Everyone!!!:) 1  
children:: REACT  
숫자::

Warning: Invalid DOM property "class". Did you mean "className"?

Warning: Failed prop type: Invalid prop 'name' of type 'number' supplied to 'MyComponent', expected 'string'.

Warning: Failed prop type: Invalid prop 'name' of type 'number' supplied to 'MyComponent', expected 'string'.

Warning: Failed prop type: The prop 'num' is marked as required in 'MyComponent', but its value is 'undefined'.

- propTypes종류
  - array: 배열
  - bool: true 혹은 false값
  - func: 함수
  - number: 숫자
  - object: 객체
  - string: 문자열
  - instanceOf(클래스): 특정 클래스의 인스턴스



- ex) instanceof(MyClass))
- oneOf(['dog', 'cat']): 주어진 배열 요소중 값 하나
- oneOfType([React.PropTypes.string, PropTypes.number])  
: 주어진 배열안의 종류중 하나
- any: 아무 종류

## useState

- useState함수의 인자에는 상태의 초기값- 값의 형태는 자유
- 함수를 호출시 배열이 반환
  - 첫번째 원소- 현재 상태
  - 두번째 원소- 상태를 바꾸어주는 함수 : setter함수
- 배열 비구조화 할당을 통해 이름을 자유롭게 지정할수 있음

```
import './App.css';
import MyComponent from './MyComponent';
import Say from './Say';
function App() {

  return (
    <div className="react">
      <MyComponent name='he' num={111}>REACT</MyComponent>
      <br />
      <hr />
      <br />
      <Say/>
    </div>
  );
}
export default App;
```

- msg= 변수
- setMsg= 변수에 대한 setter
- useState('변수의 초기값')

```
JS Say.js > [e] default
import React from 'react';

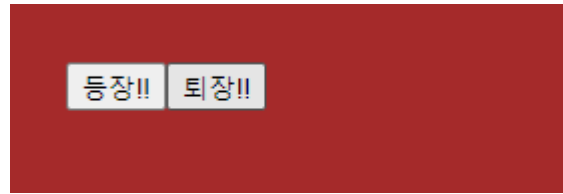
const Say = () => {
  const [msg, setMsg]= React.useState('');
  const onClickEnter= ()=> setMsg('어서오세요!');
  const onClickLeave= ()=> setMsg('안녕히가세요!!');

  return (
    <div>
      <button onClick={onClickEnter}>등장!!</button>
      <button onClick={onClickLeave}>퇴장!!</button>
      <h1>{msg}</h1>
    </div>
  );
};

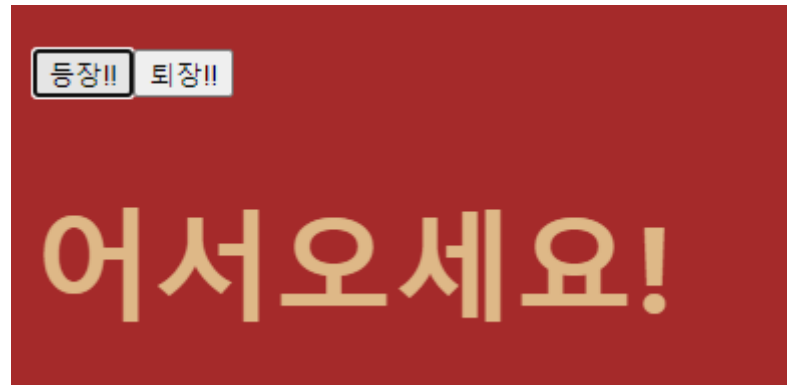
export default Say;
```

## 출력 결과

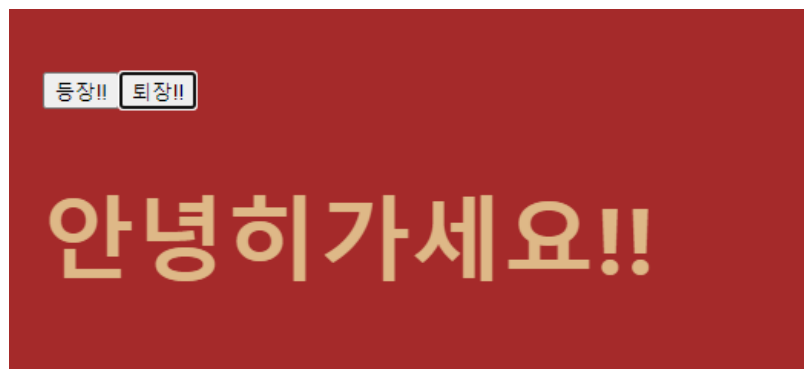
- 초기 화면



- 첫번째 버튼 클릭시



- 두번째 버튼 클릭시



- 
- useState 여러번 사용

```
import React from 'react';

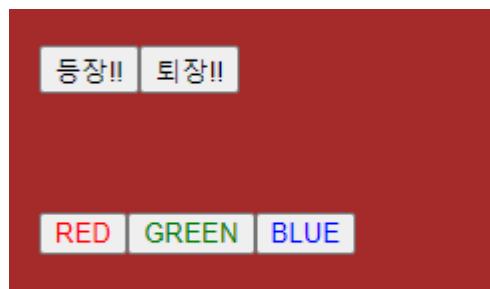
const Say = () => {
  const [msg, setMsg]= React.useState('');
  const onClickEnter= ()=> setMsg('어서오세요!');
  const onClickLeave= ()=> setMsg('안녕히가세요!!');

  const [color, setColor]= React.useState('black');
  return (
    <div>
      <button onClick={onClickEnter}>등장!!</button>
      <button onClick={onClickLeave}>퇴장!!</button>
      <h1 style={{color}}>{msg}</h1>
      <button style={{color:'red'}} onClick={()=> setColor('red')}> RED </button>
      <button style={{color:'green'}} onClick={()=> setColor('green')}> GREEN </button>
      <button style={{color:'blue'}} onClick={()=> setColor('blue')}> BLUE </button>
    </div>
  );
};

export default Say;
```

## 출력 결과

- 초기 화면



- 버튼 클릭 시



- RED 버튼 클릭시

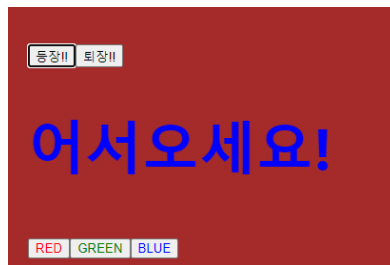




- GREEN 버튼 클릭시



- BLUE버튼 클릭시



## 사용시 주의 사항

- state값을 바꿀때- useState를 통해 전달받은(return) setter함수를 사용
- 배열이나 객체를 업데이트 할때
  - 배열이나 객체 사본 만들기→ 사본에 값을 업데이트→ 사본의 상태를 setState 혹은 setter함수를 통해 업데이트

```
const object= {a: 1, b: 2, c:3};    //상태값을 가정한 객체
const nextObject= {...object, b: 4}    //b의 값을 덮어씀
//...object: object의 모든 내용 복제
```

## 정리

- props: 부모 컴포넌트가 설정
  - state: 컴포넌트 자체적으로 지닌 값
  - 부모 컴포넌트의 state를 자식 컴포넌트의 props로 전달하고, 자식 컴포넌트에서 특정 이벤트가 발생할때 부모 컴포넌트의 메서드를 호출하면 props도 유동적으로 사용할수 있음
  - 클래스형 권장 x, 함수형 권장 o
- 

## 이벤트 핸들링

- 사용자가 웹 브라우저에서 DOM요소들과 상호작용하는 것
- onclick='' ⇒ 옛날 방식 addEventListener로 전환
- 리액트의 이벤트 시스템
  - 웹브라우저의 Html이벤트와 인터페이스가 동일 ⇒사용 방법 동일

```
<button onClick={onClickEnter}>등록!!</button>  
<button onClick={onClickLeave}>퇴장!!</button>
```

⇒

```
const onClickEnter= ()=> setMsg('어서오세요!');  
const onClickLeave= ()=> setMsg('안녕히가세요!!');
```

- 이벤트 사용시 주의 사항
  - 카멜 표기법으로 작성
  - 함수형태의 값을 전달: 리액트에서는 함수 형태의 객체를 전달
    - 바로 만들어 전달: callback형식
    - 렌더링 부분 외부에 미리 만들어 전달: 별도의 함수를 연결
- DOM요소에만 이벤트를 설정할수 있음
  - div, button, input등 DOM요소에는 이벤트를 설정할수 있지만, 직접 만든 컴포넌트에는 이벤트를 자체적으로 설정할수 없음

- <컴포넌트이름 onClick={~}/> ⇒ X
- <div onClick={this,props.onClick}> {~} </div> ⇒ 0

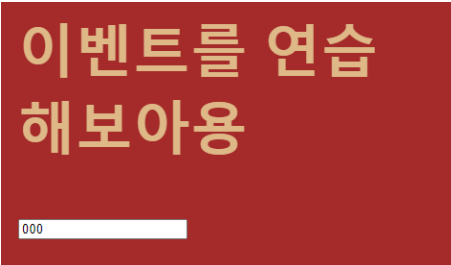
## onChange 이벤트 핸들링

```
import './App.css';
import Event from './Event';
function App() {
  return (
    <div className="react">
      <br />
      <Event />
    </div>
  );
}
export default App;
```

```
rc > JS Event.js > (0) default
1  import React from 'react';
2
3  const Event = () => {
4    return (
5      <div>
6        <h1>이벤트를 연습해보아용</h1>
7        <input type="text" name="msg" placeholder="Message" onChange={(e) => {console.log(e);}} />
8      </div>
9    );
10 };
11
12 export default Event;
```

- onChange={(e) => {...}} ⇒ 익명 함수

## 출력 결과



```

SyntheticBaseEvent {
  _reactName: 'onChange',
  _targetInst: null,
  type: 'change',
  nativeEvent: InputEvent,
  target:
    input, ...
  bubbles: true
  cancelable: false
  currentTarget: null
  defaultPrevented: false
  eventPhase: 3
  isDefaultPrevented: f functionThatReturnsFalse()
  isPropagationStopped: f functionThatReturnsFalse()
  isTrusted: true
  nativeEvent: InputEvent {
    isTrusted: true,
    data: '0',
    isComposing: false,
    inputType: 'insertText',
    dataTransfer:
      target: input
    timeStamp: 199286.5
    type: 'change'
    _reactName: "onChange"
    _targetInst: null
  }
  [[Prototype]]: Object

```

- 콘솔에서 기록되는 e객체는 syntheticEvent로 웹 브라우저의 네이티브 이벤트를 감싸는 객체. 네이티브 이벤트와 인터페이스가 같으므로 순수 자바스크립트에서 Html이벤트를 다룰때와 똑같이 사용하면 된다.

## state에 input값 담기- 버튼 누를시 msg값을 공백으로 설정

```
import React from 'react';

const Event = () => {
  const [msg, setMsg] = React.useState('');
  return (
    <div>
      <h1>이벤트를 연습해보아용</h1>
      <input type='text' name='msg' placeholder='Message'
        value={msg} onChange={(e) => {setMsg(e.target.value);}} />
      <button onClick={() => {alert(msg); setMsg('');}}> 확인! </button>
    </div>
  );
};

export default Event;
```

## 출력 결과



## 임의 메서드 만들기

- 생성자 함수는 함수 컴포넌트에서 불필요

```
import React from 'react';

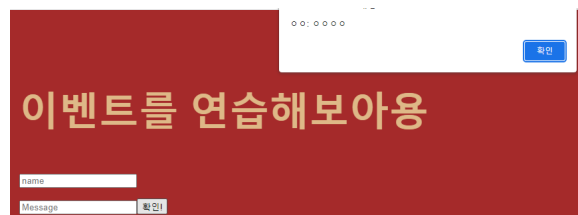
const Event = () => {
  const [msg, setMsg] = React.useState('');
  const handlerChange = (e) => {
    setMsg(e.target.value);
  }
  const handlerClick = (e) => {
    alert(msg);
    setMsg('');
  }
  return (
    <div>
      <h1>이벤트를 연습해보아용</h1>
      <input type='text' name='msg' placeholder='Message'
        value={msg} onChange={handlerChange}/>
      <button onClick={handlerClick}> 확인! </button>
    </div>
  );
};

export default Event;
```

- 출력 결과는 위와 동일

## OnKeyPress 이벤트 핸들링

### 출력 결과



- 상태값을 JSON형태로 구성
  - 하나의 form이라는 값에 여러 하위값을 포함시킴
- input태그의 내용이 바뀔시  
e.target.name⇒ name or



```

1  import React from "react";
2
3  const Event = () => {
4    const [form, setForm] = React.useState({
5      name: "",
6      msg: ""
7    });
8    const {name, msg} = form;
9    const handlerChange = (e) => {
10     const nextForm = {
11       ...form,
12       [e.target.name]: e.target.value
13     };
14     setForm(nextForm);
15   };
16   const handlerClick = (e) => {
17     alert(name + ': ' + msg);
18     setForm({
19       name: '',
20       msg: ''
21     });
22   };
23   const handleKeyPress = (e) => {
24     if (e.key === 'Enter') {
25       handlerClick();
26     }
27   };
28   return (
29     <div>
30       <h1>이벤트를 연습해보아요</h1>
31       <input
32         type="text"
33         name="name"
34         placeholder="name"
35         value={name}
36         onChange={handlerChange}
37       />
38       <br />
39       <input
40         type="text"
41         name="msg"
42         placeholder="Message"
43         value={msg}
44         onChange={handlerChange}
45         onKeyPress={handleKeyPress}
46       />
47       <button onClick={handlerClick}> 확인! </button>
48     </div>
49   );
50 };
51
52 export default Event;
53

```

msg

→ 상태값에 입력값을 덮어씌움