이벤트, 상태

React 상호작용

이벤트 핸들링

이벤트 핸들링

- function eventHandler() {}
- onClick={eventHandler}
- onClick={(e) => this.deleteRow(id, e)}
- onClick={this.deleteRow.bind(this, id)}

이벤트 핸들링

- 이벤트 사용 시 주의사항
  - 이벤트 이름은 카멜 표기법으로 작성
  - 이벤트에 실행할 자바스크립트 코드를 전달하는 것이 아니라 **함수 형태의 값**을 전달
  - DOM 요소에만 이벤트를 설정 할 수 있다.
- 이벤트 타입
  - onClick
  - onChange
  - onKeyPress

Hook

hook이란

- 원래 존재하는 어떤 기능에 마치 갈고리를 거는 것처럼 끼어 들어가 같이 수행되는 것
- 리액트의 **state**와 생명주기 기능에 갈고리를 걸어 원하는 시점에 정해진 함수를 실행하도록 만든 것으로 이때 실행되는 함수를 훅이라고 함.
- 훅의 이름은 모두 **use**로 시작
- 훅은 무조건 함수 컴포넌트의 최상위 레벨에서만 호출
  - 반복문이나 조건문 안에서 호출하면 안됨
  - 훅은 컴포넌트가 렌더링 될 때마다 매번 같은 순서로 호출되어야 함.
  - 리액트 함수 컴포넌트에서만 훅을 호출할 수 있음

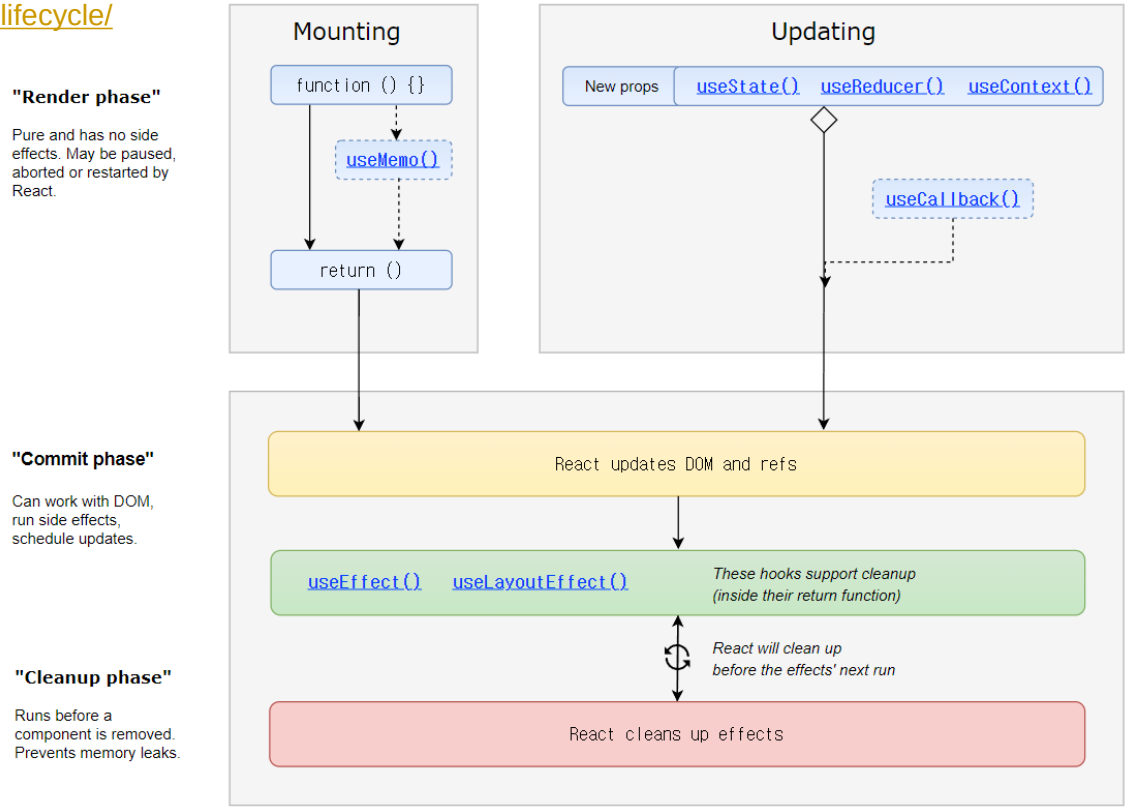
hooks

- useState
- useEffect
- useMemo
- useCallback
- useRef
- useReduce

https://ko.react.dev/reference/react/hooks

function React hook Lifecycle

https://wavez.github.io/react-hooks-lifecycle/

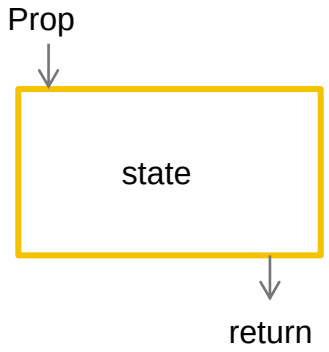


리액트 컴포넌트의 변경가능한 데이터

state

state

- 리액트 컴포넌트의 변경가능한 데이터
  - 입력값 props를 이용해서 새로운 UI를 만들어서 리턴하며 props가 변경되면 리렌더링이 발생
  - state를 변경해도 리 렌터링이 발생함




```
import {useState} from 'react';

function MyComponent(props) {
   //const _node = useState('welcome');
   const [node, SetNode] = useState('welcome');
   return ( <div></div>  )
}
```

state

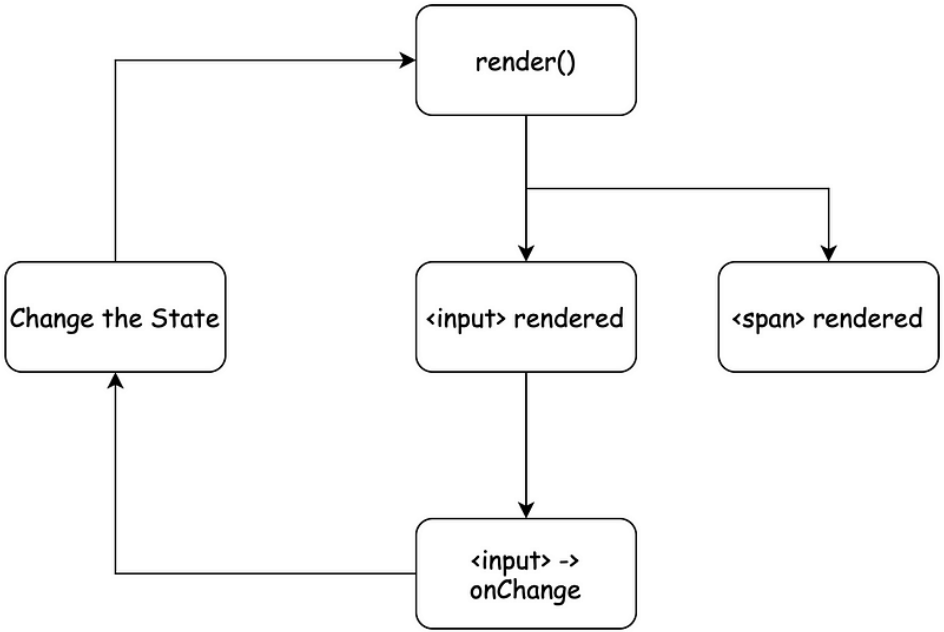
- 리액트 컴포넌트의 변경 가능한 데이터
  - 렌더링이나 데이트 흐름에 사용되는 값만 state에 포함해야 함
  - state가 변경되면 컴포넌트 리렌더링이 발생
  - 자바스크립트 객체이며 직접적인 변경이 불가능하며 state 함수를 통해서 변경
  - 한 컴포넌트에서 여러 번 사용해도 됨
- **useState() 훅을 이용하여 개발자가 직접 정의**

배열이 반환되며 배열의 비구조화 할당을 통해 이름 지정

```
const  [ msg,  SetMsg ]  =  useState('hello');
```

현재 상태변수 상태를 바꾸는 함수(setter) 현재 상태의 초깃값

렌더링


render()
Change the State
<input> rendered
 rendered
<input> ->
onChange

Ref로 값 참조하기

ref

- **DOM을 직접 선택**
  - 특정 앨리먼트의 크기 확인, 스크롤바 위치 설정, 포커스 설정이나 video 관련 라이브러리, D3, chart.js와 같은 외부 라이브러리 사용 시 이용
  - ref는 전역으로 작동하지 않고 컴포넌트 내부에서만 작동
- Refs는 렌더링에 사용되지 않는 값을 고정하기 위한 escape hatch
  - state와 달리 ref의 current 값을 설정하면 리렌더가 트리거되지 않음.
  - state와 마찬가지로 ref는 컴포넌트의 렌더링 간에 정보를 유지할 수 있음.
  - 렌더링 중에 ref.current를 읽거나 쓰지 말 것. 컴포넌트를 예측하기 어렵게 만듬.
- **useRef Hook을 호출해 ref를 달라고 React에 요청.**
  - state와 달리 ref는 읽거나 설정할 수 있는 current라는 프로퍼티를 호출할 수 있는 자바스크립트 순수객체.
  - ref는 숫자, 문자열, 객체, 심지어 함수 등 모든 것을 가리킬 수 있음.
    https://ko.react.dev/learn/referencing-values-with-refs

컴포넌트에 ref를 추가하기

```
import React, { useState, useRef } from 'react';

function InputSample() {
  const [inputs, setInputs] = useState({ name:'',  nickname:'' });
  const nameInput = useRef();

  const { name, nickname } = inputs;

  const onChange = e => {
    const { value, name } = e.target;
    setInputs({ ...inputs,  [name]: value  });
  };

  const onReset = () => {
    setInputs({   name: '',   nickname: ''    });
    nameInput.current.focus();
  };
```

```
return (
    <div>
      <input
        name="name"
        placeholder="이름"
        onChange={onChange}
        value={name}
        ref={nameInput}
      />
      <button onClick={onReset}>초기화</button>
</div>
  );
}

export default InputSample;
```

컴포넌트에 ref를 추가하기

- useRef Hook을 가져와 컴포넌트에 ref를 추가

```
import { useRef } from 'react';
```

- 컴포넌트 내에서 **useRef** Hook을 호출
  - 참조할 초깃값을 유일한 인자로 전달하면 useRef 는 다음과 같은 객체를 반환

```
const  nameInputRef = useRef(0);
```

```
{  current: 0   }          // useRef에 전달한 값
```

- 선택하고 싶은 DOM 앨리먼트에 ref 속성 설정

```
<input ref={nameInputRef } />
```

- **ref.current** 프로퍼티를 통해 변경하거나 읽을 수 있음

```
nameInputRef .current.focus()
```

ref와 state 비표

refs	state
useRef(initalValue) 는 {current:initalValue}을 반환	UseState(initialValue)은 state 변수의 현재 값과 setter 함수[value, setValue] 를 반환
ref 를 바꿔도 리렌더 되지 않음	State를 바꾸면 리덴더 됨
Mutable-렌더링 프로세스 외부에서 current 값을 수정 및 업데이트 할 수 있음	Immutable state를 수정하기 위해서는 state 설정함수를 반드시 사용하여 리렌터 대기열에 넣어야 함
렌더링 중에는 current값을 읽거나 쓰면 안됨	언제든지 state를 읽을 수 있음. 그러나 렌더마다 변경되지 않는 자체적인snapshot이 있음

ref를 사용할 시기

- 컴포넌트가 React를 “외부”와 외부 API—컴포넌트의 형태에 영향을 미치지 않는 브라우저 API 와 통신해야 할 때 ref를 사용

- ref로 작업할 때 mutation 방지에 대해 걱정할 필요가 없습니다. 변형하는 객체가 렌더링에 사용되지 않는 한, React는 ref 혹은 해당 콘텐츠를 어떻게 처리하든 신경 쓰지 않습니다.

컴포넌트가 렌더링될 때마다 특정 작업을 수행하도록 설정하는 **hook**

Effects

https://ko.react.dev/learn/synchronizing-with-effects

Effect 란

- **React 코드를 벗어난 특정 외부 시스템과 동기화하기 위해 사용**
  - 브라우저 API, 써드파티 위젯, 네트워크 등을 포함
  - 예를 들어 React의 state을 기준으로 React와 상관없는 구성 요소를 제어하거나, 서버 연결을 설정하거나, 구성 요소가 화면에 나타날 때 분석 목적의 로그를 전송할 수도 있습니다. *Effect*를 사용하면 렌더링 후 특정 코드를 실행하여 React 외부의 시스템과 컴포넌트를 동기화할 수 있습니다.
- **Effect는 화면 업데이트가 이루어지고 나서 실행**
  - 이 시점이 React 컴포넌트를 외부 시스템(네트워크 또는 써드파티 라이브러리와 같은)과 동기화하기 좋은 타이밍입니다.
- **렌더링 자체에 의해 발생하는 부수 효과를 특정하는 것으로, 특정 이벤트가 아닌 렌더링에 의해 직접 발생**
  - 채팅에서 메시지를 보내는 것은 *이벤트*입니다. 왜냐하면 이것은 사용자가 특정 버튼을 클릭함에 따라 직접적으로 발생합니다. 그러나 서버 연결 설정은 *Effect*입니다

Effect를 작성하는 법

- **1단계 : Effect 선언.**
  - 기본적으로 Effect는 모든 렌더링 후에 실행됩니다.
- **2단계: Effect 의존성 지정.**
  - 대부분의 Effect는 모든 렌더링 후가 아닌 *필요할 때*만 다시 실행되어야 합니다. 예를 들어, 페이드 인 애니메이션은 컴포넌트가 나타날 때에만 트리거 되어야 합니다. 채팅 방에 연결, 연결 해제하는 것은 컴포넌트가 나타나거나 사라질 때 또는 채팅 방이 변경될 때만 발생해야 합니다. *의존성*을 지정하여 이를 제어하는 방법을 배우게 될 것입니다.
- **3단계: 필요한 경우 클린업 함수 추가.**
  - 일부 Effect는 수행 중이던 작업을 중지, 취소 또는 정리하는 방법을 지정해야 할 수 있습니다. 예를 들어, “연결”은 “연결 해제”가 필요하며, “구독”은 “구독 취소”가 필요하고, “불러오기(fetch)“는 ”취소” 또는 ”무시”가 필요합니다. 이런 경우에 Effect에서 *클린업 함수(cleanup function)*를 반환하여 어떻게 수행하는지 배우게 될 것입니다.

Effect를 작성하는 법

- **1단계: Effect 선언하기**
  - 컴포넌트가 렌더링 될 때마다 React는 화면을 업데이트한 다음 useEffect 내부의 코드를 실행합니다.
  - 다시 말해, useEffect는 화면에 렌더링이 반영될 때까지 코드 실행을 "지연"시킵니다.

```
import { useEffect } from 'react';
```

```
function MyComponent() {

useEffect(() => {

// 이곳의 코드는 *모든* 렌더링 후에 실행됩니다

});

return <div />;

}
```

Effect를 작성하는 법

- **2단계: Effect의 의존성 지정하기**
  - React에게 Effect를 **불필요하게 다시 실행하지 않도록 지시**
  - useEffect 호출의 두 번째 인자로 ***의존성(dependencies)*** **배열**을 지정
  - 의존성 배열로 [isPlaying]을 지정하면 React에게 이전 렌더링 중에 isPlaying이 이전과 동일하다면 Effect를 다시 실행하지 않도록 해야 한다고 알려줍니다

마운트 될때만 실행하고 싶을 때

```
useEffect(() => {

// ...

}, [ ] );
```

특정 값이 업데이트될 때만 실행하고 싶을 때

```
useEffect(() => {

// ...

}, [isPlaying]);
```

3단계: 필요하다면 클린업을 추가하세요

- 컴포넌트가 언마운트되기 전이나 업데이트되기 직전(**Effect**가 다시 실행되지 전)에 어떠한 작업을 수행하고 하는 경우 클린업 추가
  - name이 업데이트되어서 렌더링될 때마다 클린업 함수 호출되어 업데이트 되기 직전의 상태 확인

```
useEffect(() => {
  console.log("마운트")
   return () => {    console.log("언마운트", name);   };
}, [name]);
```

```
useEffect(() => {
  const connection = createConnection();
  connection.connect();
  return () => {
    connection.disconnect();
  };
}, []);
```

  - 언마운트될 때만 호출하고 싶다면 빈배열

```
useEffect(() => {
  console.log("마운트")
   return () => {    console.log("언마운트");   };
}, [ ]);
```

input 태그

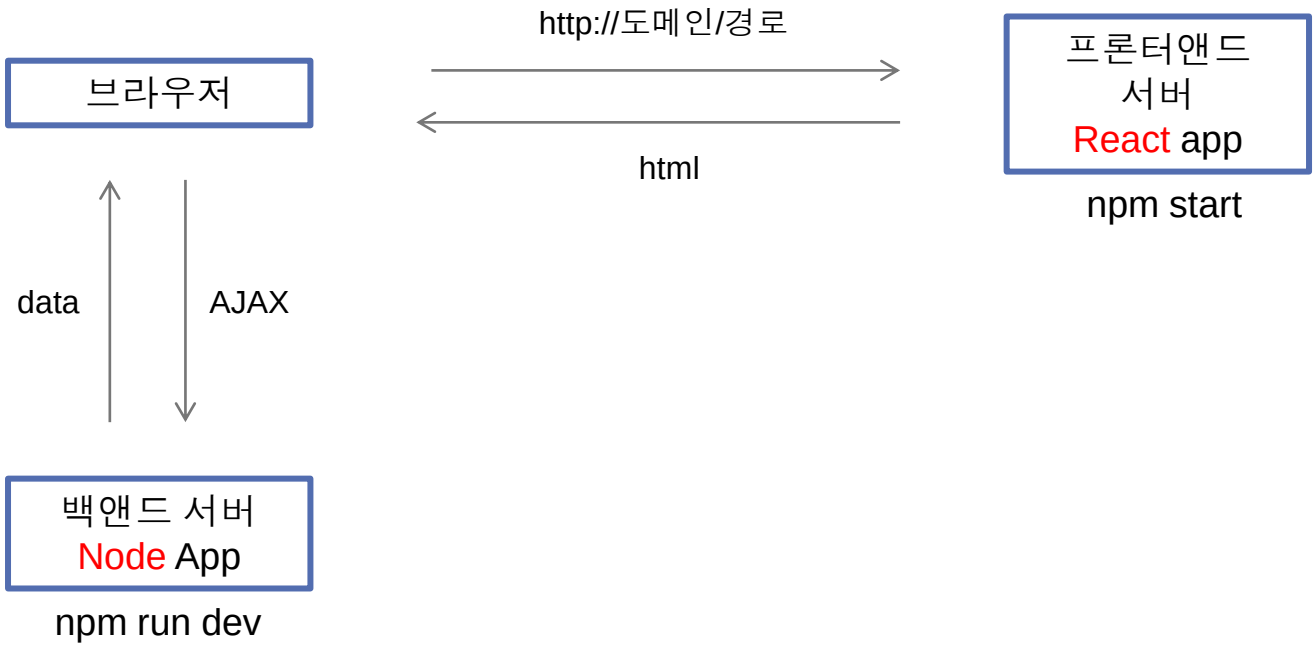
- **state 지정**
- **onChange 이벤트에서 setXXX 함수에 value 전달**
  - input태그에 onChange가 없으면 Warning 발생
  - 객체 복사해서 변경된 값만 update

```
const [form, setForm] = useState({ username: "", message: "" });
const { username, message } = form;
const ref_name = useRef(null);
ref_name.current.focus();

 <input type="text" placeholder="이름"  name="username"
      ref={ref_name}
      value={username}
      onChange={(e) => setForm(
            { ...form, [e.target.name]: e.target.value  }
       )}
    />
```

AJAX

Ajax 호출하기


http://도메인/경로
브라우저
프론터앤드
서버
React app
npm start
html
data
AJAX
백앤드 서버
Node App
npm run dev

https://axios-http.com/kr/docs

axios

- axios 라이브러리 이용
- `npm i axios`
- get(), post(), put(), delete()

```
axios.get('/user', { params: { ID: 12345 }
  })
  .then(function (response) {
    console.log(response);
    console.log(response.data);
  })
  .catch(function (error) {
    console.log(error);
  })
  .finally(function () {
    // 항상 실행되는 영역
  });
```

```
axios.get('/user/12345')
  .then(function (response) {
     console.log(response.data);
  })
```

axios

- **query String**

```
axios.get('/user?ID=12345' )
```

```
axios.get('/user', { ID: 12345 } )
```

```
axios.get('/user', {
     data: { ID: 12345 }
  })
```

- **URL params**

```
axios.get('/user/12345')
```

```
axios.get('/user', {
     params: { ID: 12345 }
  })
```

- **JSON String**

```
axios.post('/user', {userId:12345, username:"hong"} )
```

```
axios.put('/user/12345', {userId:12345} )
```

```
import axios from "axios";
import React from "react";

const baseURL =
"https://jsonplaceholder.typicode.com/posts/1";

export default function App() {
  const [post, setPost] = React.useState(null);

  React.useEffect(() => {
    axios.get(baseURL).then((response) => {
      setPost(response.data);
    });
  }, []);
  if (!post) return null;
  return (
    <div>
      <h1>{post.title}</h1>
      <p>{post.body}</p>
    </div>
  );
}
```