

React

Part3

component Life Cycle
useEffect hook을 사용한 lifeCycle 제어
axios를 이용한 서버와의 데이터 통신

React | component Life Cycle

- LifeCycle이란 컴포넌트의 생애주기를 말한다.
- 컴포넌트의 생애주기란 컴퍼넌트의 생성 -> 업데이트(리렌더링) -> 소멸의 과정을 의미한다.
- 컴포넌트의 생애주기 자체가 중요한 것이 아니라,
개발자가 각 컴포넌트의 생애주기 조건에 따라 필요한 코드를 작성할 수 있어야 한다는 점이 중요하다.
ex> 컴포넌트가 생성될 때 ~~~ 기능을 실행하세요. 컴포넌트가 업데이트(리렌더링) 될때 ~~~ 기능을 실행해주세요 등

컴포넌트의 생애주기

- mount : 컴포넌트가 화면에 최초 렌더링되는 시점을 말한다.
- update : mount 이후 state 변경 함수 호출 등의 이유로 컴포넌트가 리렌더링되는 시점을 말한다. -
- unmount : 컴포넌트가 렌더링에서 제외되는 시점을 말한다. (페이지를 이동하면 컴포넌트에 화면에서 사라지는 경우를 말함)

- useEffect hook은 컴포넌트의 라이프 사이클 각 주기에 특정 기능을 구현할 때 사용한다.
ex> 컴포넌트가 mount될때 ~~ 기능 실행하세요, 컴포넌트가 update될때 ~~ 기능 실행해주세요
 컴포넌트가 unmount될때 ~~ 기능 실행해주세요 등등
- 이렇듯 컴포넌트의 특정 생애주기에 맞춰 기능을 수행하도록 코드를 작성하는 것을 라이프사이클 제어라 부른다.

useEffect hook 사용문법

- useEffect(실행함수, 의존성 배열);

- useEffect()의 첫번째 인자로 함수 선언문을 넣는다. (함수 선언문 : () => {})
- 함수 선언문의 {} 안에 실행할 코드를 작성한다.
- 의존성 배열이란 useEffect 안의 내용을 실행할 생애주기를 컨트롤하기 위한 내용으로 의존성 배열에 작성된 state변수의 값이 업데이트될 때 useEffect 안의 내용은 다시 실행된다.
- 만약, 의존성 배열로 빈 배열을 전달하면 컴포넌트 mount 시에만 useEffect의 내용이 실행된다.
- 만약, 의존성 배열을 작성하지 않는다면 useEffect 코드를 작성한 컴포넌트가 업데이트(리렌더링)될때마다 useEffect 안의 내용이 다시 실행된다.

React | useEffect - 2

useEffect를 사용한 코드

```
function App() {  
  useEffect(() => {  
    console.log('useEffect 실행');  
  });  
  
  return (  
    <h2>Hello React</h2>  
  )  
}
```

- 왼쪽과 같은 코드가 useEffect를 사용하는 가장 간단한 코드이다.
- 이 코드는 useEffect()의 두번째 매개변수인 의존성 배열을 작성하지 않았기 때문에 App 컴포넌트가 mount될때와 update 될때 실행된다.

그럼 useEffect를 사용한 위의 코드와 아래처럼 useEffect를 사용하지 않은 코드는 어떤 차이가 있을까?

```
function App() {  
  console.log('useEffect 실행');  
  
  return (  
    <h2>Hello React</h2>  
  )  
}
```

React | useEffect - 3

useEffect 안의 코드는 렌더링 후(html로 화면 다 그린 후) 실행된다.

그럼 이러한 차이점을 우리가 어떻게 사용할 수 있을까 생각해봐야 한다.

```
function App() {  
  for(let i = 0 ; i < 10000 ; i++){  
    console.log(i);  
  }  
  
  return (  
    <h2>Hello React</h2>  
  )  
}
```

useEffect 미사용

```
function App() {  
  useEffect(() => {  
    for(let i = 0 ; i < 10000 ; i++){  
      console.log(i);  
    }  
  });  
  
  return (  
    <h2>Hello React</h2>  
  )  
}
```

useEffect 미사용

useEffect를 사용하지 않은 왼쪽과 같은 코드는 10000번 반복문을 실행 후 렌더링 하기 때문에 사용자 입장에서 화면이 늦게 뜨는 불편한 경험을 한다.

10000번 반복이 만약 3초 걸린다면 사용자는 3초 후 'Hello React' 라는 글자를 화면에서 볼 수 있다는 말이다. (3초동안은 흰 화면만 나옴)

반면, useEffect 안에 10000번 반복 코드를 작성하면 화면을 그린 후 반복하기 때문에 사용자의 불편한 경험을 줄일 수 있다.

그렇기 때문에 useEffect 안에는 통상적으로 시간이 오래걸리는 작업(**대표적으로 서버에서 데이터 받기**)을 수행하는 코드를 작성한다.

React | useEffect - 4

useEffect의 두번째 매개변수인 의존성배열을 어떻게 작성하냐에 따라 useEffect 내용의 실행 시점에 변화를 줄 수 있다.(lifeCycle제어)

```
function App() {
  const [age, setAge] = useState(0);
  const [name, setName] = useState('kim');

  useEffect(() => {
    console.log('useEffect 실행~');
  });

  return (
    <>
      <h2>{age}</h2>
      <button type='button' onClick={(e) => {setAge(age + 1)}}>버튼1</button>
      <button type='button' onClick={(e) => {setName('Lee')}}>버튼2</button>
    </>
  )
}
```

useEffect의 두번째 인자인 의존성배열을 작성하지 않으면 useEffect 안의 코드는 App 컴포넌트가 mount, update 되는 시점에 동작한다.

```
useEffect(() => {
  console.log('useEffect 실행~');
}, [name]);
```

의존성배열로 [name]를 작성하면 App컴포넌트가 mount될때, state 변수 name이 변하여 app 컴포넌트가 리렌더링 될때만 useEffect 안의 내용이 실행된다.

```
useEffect(() => {
  console.log('useEffect 실행~');
}, []);
```

의존성배열을 빈배열로 주면 App 컴포넌트가 mount 될때만 useEffect 안의 내용이 실행된다

```
useEffect(() => {
  console.log('useEffect 실행~');
}, [age]);
```

의존성배열로 [age]를 작성하면 App컴포넌트가 mount될 때, state 변수 age가 변하여 app 컴포넌트가 리렌더링 될때만 useEffect 안의 내용이 실행된다.

```
useEffect(() => {
  console.log('useEffect 실행~');
}, [age, name]);
```

의존성배열로 [age, name]를 작성하면 App컴포넌트가 mount될때, state 변수 age가 변하거나 state 변수 name이 변하여 app 컴포넌트가 리렌더링 될때 useEffect 안의 내용이 실행된다.

React | useEffect - 정리

useEffect를 사용하면 컴포넌트가 unmount되는 시점의 코드 작성도 가능하다.

이 내용은 다음에 알아보고 이번 시간에는 넘어가도록 하겠다.

useEffect 정리

- useEffect를 사용하면 컴포넌트의 생애주기(mount, upadte, unmount)에 따른 실행코드를 제어할 수 있다.
- useEffect 안의 코드는 화면 렌더링 후 실행된다.
- 그렇기 때문에 useEffect 안의 내용은 통상적으로 실행에 오래 걸리는 코드(서버에서 데이터 받기)를 작성한다.
- useEffect()의 첫번째 인자로는 화살표 함수를 전달하여 실행코드를 작성하고, 두번째 인자로 의존성배열을 작성하여 실행시점을 제어할 수 있다.
- 의존성 배열을 작성하지 않는다면 useEffect 안의 코드는 컴포넌트가 mount되는 시점과 update 되는 시점에 실행된다.
- 의존성 배열을 빈배열([])로 작성되면 컴포넌트가 mount되는 시점에만 실행된다.
- 의존성 배열에 state변수를 추가하면, 컴포넌트가 mount되는 시점과 의존성배열에 추가한 state변수의 값이 변해 컴포넌트가 리렌더 될때 실행 된다.
- 의존성 배열에는 필요한만큼 state 변수를 여러 개 추가할 수 있다.

React | axios를 이용한 서버와의 데이터 통신 - 1

axios 라이브러리를 사용하면 서버와 데이터를 통신할 수 있다.

데이터 통신이란 데이터를 주고 받는다는 의미다.

우리는 서버로 spring을 사용하기 때문에 axios를 이용하면 서버인 spring과 화면단인 react가 서로서로 데이터를 주고 받을 수 있다.

react에서 spring 서버로 통신 명령어를 보내는 것을 요청(request), 이 요청에 대한 결과를 spring이 react로 보내주는 것을 응답(response)라 한다.

axios 라이브러리를 사용하기 위해 먼저 'npm install axios' 명령어를 터미널에 입력하여 axios 라이브러리를 설치해야 한다.

- npm 가

axios 활용 서버와의 통신 문법

데이터 통신은 Restful API 문법을 활용한다.(Restful API는 Spring에서 학습 예정)

[] 안에 작성한 내용은 생략 가능한 내용임을 의미한다.

- 서버의 데이터를 react에서 받기

```
axios.get(url, [설정내용]).then(통신 성공 후 실행 내용).catch(실행 실패 시 실행 내용);
```

```
axios.delete(url, [설정내용]).then(통신 성공 후 실행 내용).catch(실행 실패 시 실행 내용);
```

- react의 데이터를 서버에 전달하기

```
axios.post(url, [전달할 데이터], [설정내용]).then(통신 성공 후 실행 내용).catch(통신 실패 시 실행 내용);
```

```
axios.put(url, [전달할 데이터], [설정내용]).then(통신 성공 후 실행 내용).catch(통신 실패 시 실행 내용);
```


React | axios를 이용한 서버와의 데이터 통신 - 2

서버와의 통신 기능을 학습하기 전에 먼저 웹에서 cors에 대해 알아야 한다.

CORS(Cross-Origin Resource Sharing) - !

우리 말로 번역하면 '교차 출처 리소스 공유' 라 할 수 있다. - ?

이는 출처가 다른 자원들을 공유한다는 뜻이며, 한 출처에 있는 자원을 다른 출처에서 사용 가능하도록 하는 개념이다.

여기서 말하는 출처는 하나의 웹 사이트라 생각하면 되며, 자원은 데이터를 의미한다고 생각하자.

예를 들어, 구글과 네이버를 생각해보자. 구글에 있는 내 메일을 네이버 메일에서 공유해서 사용할 수 있을까?

이 때의 구글과 네이버를 각각 출처라고 하며, 메일이라는 데이터를 자원이라 생각하면 된다. 당연하게도 구글과 네이버는 다른 출처(사이트)이다.

그리고 구글에서의 정보를 네이버에서 사용 가능한가? 라고 물어본다면 '기본적으로 불가능하다.' 라고 한다. 그 이유는 출처가 다른 사이트끼리의 데이터 공유는 기본적으로 제한하고 있기 때문이다. 이를 CORS 라 한다. '기본적으로 불가능'이라 한 것은 설정을 통해 CORS를 허용할 수 있다는 말이다.

서버와의 통신을 학습하기 전에 CORS를 먼저 언급한 이유는 우리가 SPRING으로 만드는 백서버와 REACT로 만드는 프론트서버는 애초에 서로 다른 사이트이기 때문에 두 사이트간(SPRING과 REACT로 만든 코드) 데이터 통신이 불가능하다. 말했듯이 서로 다른 출처끼리는 데이터를 공유할 수 있는 CORS 개념이 기본적으로 제한되어 있기 때문이다. 그렇기 때문에 우리는 SPRING과 REACT가 서로 CORS를 허용할 수 있도록 설정을 해야 한다.

React | axios를 이용한 서버와의 데이터 통신 - 3

리액트에서 SPRING 서버와 통신할 수 있도록 CORS를 허용할 수 있는 설정 코드를 아래와 같이 작성한다.

이 코드를 현재 하나하나 다 이해할 필요는 없다.

(이해를 다 하려면 네트워크 개념을 다 공부해야 해요ㅠ. 저도 네트워크를 다 이해하고 있지 않아요. 몰라도 그냥 쓰라는대로 사용하면 되는거예요)

react 프로젝트에 있는 vite.config.js 파일을 다음과 같이 변경하세요(파일 제공하니 그냥 복사/붙여넣기해서 사용)

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: {
    proxy: {
      // 경로가 "/api" 로 시작하는 요청을 대상으로 proxy 설정
      '/api': {
        // 요청 전달 대상 서버 주소 설정
        target: 'http://localhost:8080',
        // 요청 헤더 host 필드 값을 대상 서버의 호스트 이름으로 변경
        changeOrigin: true,
        // 요청 경로에서 '/api' 제거
        rewrite: (path) => path.replace(/^\/api/, ''),
      }
    }
  }
})
```

React | axios를 이용한 서버와의 데이터 통신 - get()

axios.get() 함수를 사용하면 서버로부터 데이터를 받아 올 수 있다.

```
// 서버에서 받은 나이값을 저장할 state 변수
const [age, setAge] = useState(0);

useEffect(() => {
  axios.get().then().catch();
}, []);
```

- state 변수 age는 서버로부터 나이값을 받으면 저장할 변수이다.
- useEffect 안에 서버로부터 데이터를 받아오기 위해 axios.get() 함수를 작성한다.
- useEffect안에 작성한 이유는 데이터 통신은 인터넷환경에서 진행되기 때문에 속도가 느려질 우려가 있으므로 html코드를 먼저 렌더링하기 위함이다.
- 의존성배열에 빈 배열([])을 작성하여 useEffect 안의 내용을 컴포넌트 마운트 시에만 실행하게 하였다.

```
useEffect(() => {
  axios.get('/api/age')
    .then(() => {})
    .catch(() => {});
}, []);
```

- get('/api/age') 에서 '/api/age'는 spring 서버의 접근 url(주소)이다.
- then()와 catch() 함수 안은 화살표 함수로 실행 내용을 작성한다.
- then()에는 데이터 통신 성공 후 실행 내용을 작성한다.
- catch()에는 데이터 통신 중 오류 발생 시 실행 코드를 작성한다.

React | axios를 이용한 서버와의 데이터 통신 - get()

```
useEffect(() => {  
  axios.get('/api/age')  
    .then((res) => {  
      console.log(res);  
    })  
    .catch((error) => {  
      console.log(error);  
    });  
}, []);
```

```
▶ {data: 20, status: 200, statusText: 'OK', headers: AxiosHeaders, config: {...}, ...}
```

- then() 에 작성하는 화살표 함수는 매개변수를 하나 갖는다.
- 이 매개변수를 출력해보면 응답에 대한 다양한 정보를 객체 타입으로 갖고 있음을 확인할 수 있다.
- 이 매개변수는 함수의 매개변수명이기 때문에 아무렇게나 작성할 수 있지만 응답정보를 갖고 있기 때문에 response의 줄임말 res를 많이 사용한다.
- 현재 우리에게 중요한 것은 응답에 대한 여러 정보 중 실제 서버로부터 받은 데이터이며, 이 데이터는 응답 정보 중 키 값이 data인 곳에 담겨 온다.
- 결론적으로 서버로부터 받은 데이터는 res.data를 통해 접근 가능하다.

```
▶ AxiosError {message: 'Request failed with status code 404', name: 'AxiosError', code: 'ERR_BAD_REQUEST', config: {...}, request: XMLHttpRequest, ...}
```

- catch() 에 작성하는 화살표 함수는 매개변수를 하나 갖는다.
- 이 매개변수를 출력해보면 응답실패에 대한 다양한 정보를 객체 타입으로 갖고 있음을 확인할 수 있다.
- 이 매개변수는 함수의 매개변수명이기 때문에 아무렇게나 작성할 수 있지만 통신실패정보를 갖고 있기 때문에 error를 많이 사용한다.
- 여러 오류 데이터 중 키가 message인 정보가 오류에 대한 이유를 간단히 설명한 것이다.
- 404는 요청 경로를 찾을 수 없거나, 페이지명이 잘못되었을 때 발생하는 오류 코드이다.

React | axios를 이용한 서버와의 데이터 통신 - post()

axios.post() 함수를 사용하면 서버로 데이터를 전달 할 수 있다.

```
useEffect(() => {
  axios.post('/api/name', {name : 'kim'})
    .then((res) => {
      console.log(res);
    })
    .catch((error) => {
      console.log(error);
    });
});
```

- axios.post()의 첫번째 인자로 get()함수와 마찬가지로 접근할 서버 주소를 넣는다.
- axios.post()는 get()함수와 다르게 두번째 인자로 서버에 전달할 데이터를 작성한다.
- 서버에 전달할 데이터는 객체 타입으로 작성한다.
- 객체 타입으로 데이터를 전달하기 때문에 한 번에 여러 데이터를 서버로 전달할 수 있다.
- then()과 catch()의 모든 내용은 get()과 동일하다.
- axios.post()는 서버로 데이터를 전달하는 것이기 때문에 then()에서는 데이터를 받는 코드 보다는, 데이터 전달 결과에 대한 코드를 작성한다.