useEffect 훅

**학습목표**

컴포넌트 생명주기

useEffect 훅

**목차**

[1. 개요 2](#_Toc133416611)

[1) 컴포넌트 생명 주기 2](#_Toc133416612)

[2) useEffect 훅 2](#_Toc133416613)

[2. 예제 3](#_Toc133416614)

[1) 호출 형태 #1: update 3](#_Toc133416615)

[2) 호출 형태 #2: 상태 변수 5](#_Toc133416616)

[3) 호출 형태 #3: mount 6](#_Toc133416617)

[4) 호출 형태 #3: mount, unmount 7](#_Toc133416618)

[3. Clock 예제 9](#_Toc133416619)

[1) 개요 9](#_Toc133416620)

[2) 구현 9](#_Toc133416621)

# 개요

## 컴포넌트 생명 주기

컴포넌트 생명 주기(life cycle) 중 특정 시점에 실행되어야 하는 기능을 useEffect 훅으로 구현한다.

### 마운트(mount)

컴포넌트가 Virtual DOM 트리 구조에 처음 추가될 때.

### 업데이트(update)

컴포넌트가 다시 렌더링 될 때

### 언마운트(unmount)

컴포넌트가 Virtual DOM 트리에서 제거될 때

## useEffect 훅

### 호출 형태 #1: update

useEffect(콜백함수)

위와 같이 useEffect 훅을 호출하면,

컴포넌트가 업데이트(렌더링) 될 때 콜백함수가 호출된다.

### 호출 형태 #2: 상태변수

useEffect(콜백함수, [상태변수\_목록])

위와 같이 useEffect 훅을 호출하면,

상태변수 목록 중 하나의 값이 바뀌어서 화면이 렌더링 될 때,

콜백함수가 호출된다.

### 호출 형태 #3: mount

useEffect(콜백함수, [])

위와 같이 useEffect 훅을 호출하면,

컴포넌트가 마운트될 때 콜백함수가 호출된다.

만약 이 콜백함수가 어떤 함수를 리턴한다면,

그 리턴된 함수는, 컴포넌트가 언마운트(unmount) 될 때 호출된다.

# 예제

## 호출 형태 #1: update

### Counter.tsx

|  |
| --- |
| import React, { useEffect, useState } from "react";  import "./Counter.css";  function Counter() {    const [count1, setCount1] = useState<number>(0);    const [count2, setCount2] = useState<number>(0);    useEffect(() => console.log('useEffect #1', count1));    useEffect(() => console.log('useEffect #2', count2));    return (      <div className="Counter">        <h1>Counter</h1>        <div>          <span>{ count1 }</span>          <button onClick={e=>setCount1(count1+1)}>++</button>        </div>        <div>          <span>{ count2 }</span>          <button onClick={e=>setCount2(count2+1)}>++</button>        </div>      </div>    );  }  export default Counter; |

### Counter.css

|  |
| --- |
| .Counter span { font-size: 15pt; margin-right: 20px; }  .Counter div { margin: 20px; } |

### App.tsx

|  |
| --- |
| import React from "react";  import Counter from "./Counter";  function App() {    return <Counter />  }  export default App; |

  useEffect(() => console.log('useEffect #1', count1));

  useEffect(() => console.log('useEffect #2', count2));

Counter 컴포넌트가 렌더링 될 때 마다, 노란색 함수들이 호출된다.

호출형태 #1

Graphical user interface, text, application

Description automatically generated

useEffect로 등록한 두 콜백 함수들이,

Counter 컴포넌트가 렌더링 될 때 마다 호출되고 있다.

## 호출 형태 #2: 상태 변수

### Counter.tsx 수정

|  |
| --- |
| import React, { useEffect, useState } from "react";  import "./Counter.css";  function Counter() {    const [count1, setCount1] = useState<number>(0);    const [count2, setCount2] = useState<number>(0);    useEffect(() => console.log('useEffect #1', count1), [count1]);    useEffect(() => console.log('useEffect #2', count2), [count2]);    return (      <div className="Counter">        <h1>Counter</h1>        <div>          <span>{ count1 }</span>          <button onClick={e=>setCount1(count1+1)}>++</button>        </div>        <div>          <span>{ count2 }</span>          <button onClick={e=>setCount2(count2+1)}>++</button>        </div>      </div>    );  }  export default Counter; |

  useEffect(() => console.log('useEffect #1', count1), [count1])

count1 상태 값이 변경될 때에만, 노란색 함수가 호출된다.

  useEffect(() => console.log('useEffect #2', count2), [count2])

count2 상태 값이 변경될 때에만, 노란색 함수가 호출된다.

Graphical user interface, text, application, email

Description automatically generated

처음에는 두 콜백함수가 모두 호출되었지만,

그 다음부터 count1 상태 값만 변경되니, 첫 번째 콜백함수만 호출되고 있다.

## 호출 형태 #3: mount

### Counter.tsx 수정

|  |
| --- |
| import React, { useEffect, useState } from "react";  import "./Counter.css";  function Counter() {    const [count1, setCount1] = useState<number>(0);    const [count2, setCount2] = useState<number>(0);    useEffect(() => console.log('useEffect #1', count1), [count1])    useEffect(() => console.log('useEffect #2', count2), [count2])    useEffect(() => console.log('mount'), [])    return (      <div className="Counter">        <h1>Counter</h1>        <div>          <span>{ count1 }</span>          <button onClick={e=>setCount1(count1+1)}>++</button>        </div>        <div>          <span>{ count2 }</span>          <button onClick={e=>setCount2(count2+1)}>++</button>        </div>      </div>    );  }  export default Counter; |

useEffect(() => console.log('mount'), [])

컴포넌트가 마운트될 때, 노란색 함수가 호출된다.

Graphical user interface, text, application

Description automatically generated

컴포넌트가 마운트될 때, 한 번 콜백함수가 호출되고 있다.

## 호출 형태 #3: mount, unmount

### Counter.tsx 수정

|  |
| --- |
| import React, { useEffect, useState } from "react";  import "./Counter.css";  function Counter() {    const [count1, setCount1] = useState<number>(0);    const [count2, setCount2] = useState<number>(0);    useEffect(() => console.log('useEffect #1', count1), [count1])    useEffect(() => console.log('useEffect #2', count2), [count2])    useEffect(() => {      console.log('mount')      return () => console.log('unmount')    }, [])    return (      <div className="Counter">        <h1>Counter</h1>        <div>          <span>{ count1 }</span>          <button onClick={e=>setCount1(count1+1)}>++</button>        </div>        <div>          <span>{ count2 }</span>          <button onClick={e=>setCount2(count2+1)}>++</button>        </div>      </div>    );  }  export default Counter; |

useEffect(() => {

    console.log('mount')

    return () => console.log('unmount')

}, [])

컴포넌트가 마운트될 때, 노란색 함수가 호출된다.

이 함수가 리턴하는 것은 () => console.log('unmount') 함수이다.

리턴된 함수는, 컴포넌트가 언마운트될 때 호출된다.

### App.tsx 수정

|  |
| --- |
| import React, { useState } from "react";  import Counter from "./Counter";  function App() {    const [visible, setVisible] = useState<boolean>(true);    return (      <div>        <h1>App</h1>        <input type='checkbox' checked={visible} style={{height: 20, width: 20}}               onChange={e=>setVisible(e.target.checked)} />        { visible ? <Counter /> : null }      </div>    );  }  export default App; |

컴포넌트가 처음 만들어진 때 => mount

컴포넌트가 화면에서 사라질 때 => unmount

App 컴포넌트의 visible 상태 값이 true일 때, Counter 컴포넌트가 마운트되고

false일 때, Counter 컴포넌트가 언마운트된다

Graphical user interface, text, application, email

Description automatically generated

# Clock 예제

## 개요

const id = setInterval(콜백함수, 1000)

위 코드는에 의해 콜백함수는 1000 밀리초 마다 한 번씩 반복 호출된다.

그런데 이 반복 호출은 웹브라우저 창이 닫힐 때까지 계속된다.

반복 호출을 중단하려면, 다음과 같이 clearInterval 함수를 호출해야 한다.

clearInterval(id)

1초 마다 컴포넌트 화면이 렌더링 되어야 한다면,

mount 될 때, setInterval 함수를 호출해서 콜백 함수를 등록해야하고,

unmount 될 때, clearInterval 함수를 호출해서 콜백 함수 등록을 취소해야 한다.

콜백 함수 등록을 취소하지 않으면, 메모리 누수가 발생한다.

## 구현

### Clock.tsx

|  |
| --- |
| import React, { useEffect, useState } from 'react'  import './Clock.css'  function Clock() {    const [time, setTime] = useState<Date>(new Date())    useEffect(() => {      const id = setInterval(() => setTime(new Date()), 1000)      return () => clearInterval(id)    }, [])    return (      <div className='Clock'>        <h1>Clock</h1>        <p>{ time.toLocaleTimeString() }</p>      </div>    )  }  export default Clock |

const id = setInterval(() => setTime(new Date()), 1000)

Clock 컴포넌트가 마운트될 때, 위 코드가 실행된다.

1초에 한 번씩 time 상태 값이 현재 시각으로 바뀐다

() => clearInterval(id)

Clock 컴포넌트가 언마운트될 때, 위 코드가 실행된다.

1초에 한 반씩 콜백함수 호출하던 것을 중단한다.

### Clock.css

|  |
| --- |
| .Clock { text-align: center; width: 200px; border: 1px solid gray;           margin: 20px auto; }  .Clock h1 { margin: 10px; }  .Clock p { font-size: 16pt; } |

### App.tsx

|  |
| --- |
| import React from "react";  import Clock from "./Clock";  function App() {    return <Clock />  }  export default App; |

Graphical user interface, application

Description automatically generated