2024\_1학기 프론트엔드프레임워크 4/17 3~6

**3. 리액트 기초**

개발자창의 console 탭을 열어보면, 버튼이 클릭될 때 마다 count 변수값이 변경되고 있음을 알 수 있다.

count 변수 값은 변경되었지만, 화면이 다시 그려지지 않은 것이다. (렌더링 되지 않는 것)

변수값이 변경되다고 해서, 리액트 앱 화면이 다시 렌더링 되지 않는다.

# useState 훅(hook)

# useState와 상태 변수

평범한 자바스크립트 변수의 값이 변경될 때, 화면은 다시 렌더링 되지 않는다.

어떤 값이 변경될 때 마다 화면이 다시 렌더링 되어야 한다면,

그 값은 평범한 자바크립트 변수 대신 리액트 상태 변수에 저장되어야 한다.

즉 리액트 상태 변수의 값이 변경될 때는 화면이 다시 렌더링 된다.

### useState 훅

리액트에는 이름이 use로 시작하는 특별한 함수들이 제공된다.

이 함수들을 함수라고 부르지 않고 훅(hook)이라고 부른다.

useState 훅을 이용해서 리액트 상태 변수를 만들어야 한다.

import React, { useState } from 'react';

useState 훅 import

const [count, setCount] = useState(0);

리액트 상태 변수 만드는 코드.

이렇게 만들어진 상태 변수는 리액트 내부에 만들어진다. 0 부분은 그 상태 변수의 최초 값이다.

그 상태 변수의 현재 값이 리턴되어 count 상수에 대입된다.

그 상태 변수 값을 변경하려면, setCount 함수를 호출해야 한다.

화면이 다시 렌더링 될 때마다, App 함수가 다시 호출되고

아래 코드도 다시 다시 실행된다.

  const [count, setCount] = useState(0);

setCount 함수는 리액트 내부의 상태 변수 값을 변경하는 것이지,

count 상수 값을 변경하는 것이 아니다.

setCount 함수 호출로 변경된 상태 변수 값은

다음 렌더링 때 아래 코드가 다시 호출될 때 그 값이 리턴되어 count 상수에 대입된다.

const [count, setCount] = useState(0);

렌더링 시 > message 상태 값 > input태그 > setmessage 호출 > 입력 된 값 message값 채워짐

사용자가 input 태그에 문자를 하나 입력할 때 마다,

onChange 이벤트가 발생하고

setMessage 함수가 호출되어 입력된 값이 message 상태 값에 채워진다.

상태 값이 변경되었으므로, 화면이 다시 렌더링함.

### useState 훅

평범한 자바스크립트 변수의 값이 변경될 때, 화면은 다시 렌더링 되지 않는다.

useState 훅을 이용해서 리액트 상태 변수를 만들어야 한다.

const [count, setCount] = useState(0);

# 커스텀 훅

const initValue = {name: "", age: NaN, gender: "", departmentId: 0, enabled: false};

  const [student, setStudent] = useState(initValue);

  const handler = (e: any) => e.target.type === "checkbox" ?

                                setStudent({...student, [e.target.name]: e.target.checked}) :

                                setStudent({...student, [e.target.name]: e.target.value})

입력폼을 구현할 때 마다, 위와 같은 코드를 반복해서 구현해야 한다.

예를 들어 상품 정보 입력폼을 구현하려면, 다음과 같이 구현해야 한다.

  const initValue = {title: "", price: NaN, category: "" };

  const [product, setProduct] = useState(initValue);

  const handler = (e: any) => e.target.type === "checkbox" ?

                                setProduct({...product, [e.target.name]: e.target.checked}) :

                                setProduct({...product, [e.target.name]: e.target.value})

저 두 코드들은 매우 유사하다.

student, product 이름을 제외하면 사실상 동일한 코드이다.

그런데 useState 함수를 훅(hook)이라고 부르는 것처럼,

훅을 사용하는 코드를 추출하여 만든 함수를 커스텀 훅(custom hook)이라고 부른다.

유사한 코드들을 반복 구현하는 것이 귀찮아서 그 코드를 함수로 추출하여 공유하려고 할 때,

만약 그 반복되는 코드에 useState 같은 훅 함수 호출이 포함되어 있다면,

유사한 코드를 공유하려고 새로 만든 함수를 커스텀 훅(custom hook)이라고 부른다.

5. 렌더링과 todo앱

리액트 상태가 변경되어 렌더링 될 때,

상태가 변경된 컴포넌트만 렌더링 된다

부모 컴포넌트가 렌더링될 때, 자식 컴포넌트들도 전부 렌더링된다.

처음 렌더링 할 때는 당연히 모든 컴포넌트가 렌더링 되지만,

그 다음 렌더링부터는 상태가 변경된 컴포넌트와 그 자식 컴포넌트들만 렌더링 된다.

컴포넌트 상태 값이 배열일 때, 배열의 원소 값만 바뀌어서는 렌더링되지 않는다.

상태 값이 새 배열로 바뀌어야, 즉 setter에 새 배열을 전달해야 렌더링 된다.

[…arr], […person]

컴포넌트 상태 값이 객체일 때, 객체의 속성 값만 바뀌어서는 렌더링되지 않는다.

상태 값이 새 객체로 바뀌어야 렌더링 된다.

## 렌더링 안되는 이유

상태 값이 객체나 배열일 때,

렌더링 여부를 판단하기 위해 리액트는 객체나 배열의 참조만 비교하고,

그 내부 내용은 비교하지 않는다.

배열이나 객체의 내부 값들까지 전부 비교하는 것은 너무 느려서 비효율적이기 때문이다.

차라리 내부 값이 바뀔 때 마다, 배열이나 객체를 복제해서 set 하는 것이 더 빠르다.

그래서 리액트 상태 값이 객체나 배열일 때, 그 내부 값이 바뀔 때 마다, 복제해서 set 해야 한다.

# todo 앱

## 화면 분할

컴포넌트 단위로 렌더링 되고, 상태 변경이 있는 컴포넌트만 렌더링 된다.

따라서 효율적인 렌더링을 위해서, 화면의 각 부분을 컴포넌트로 분할하는 것이 좋다.

input 태그가 포함된 컴포넌트의 경우,

키가 하나 눌러질 때마다 렌더링 되어야 한다.

따라서 입력 폼 부분을 컴포넌트로 분할해야 한다.

## 컴포넌트 분할

오로지 효율만 중시하는 군대는, 전황 상태를 감시하고 그 변화에 따라 작전을 변경하고 지시하는 장교와,

오로지 주어진 임무만 기계적으로 실행하는 사병의 역할을 분리한다.

### presentation component

화면을 담당한다.

화면 제어를 위한 상태만 일부 가지고 있고, 중요한 데이터나 상태를 가지고 있지 않다. (지휘자가 아님)

props로 전달된 데이터를 화면에 출력하는 부분만 담당하는, 가볍고 단순한 컴포넌트.

이벤트가 발생하면 지휘자에게 보고해야 한다.

보고하기 위한 function이 props로 주어져야 함.

### container component

중요한 데이타와 상태를 관리한다. (지휘자)

화면 출력은 하지 않는다. (화면 출력은 부하들의 역할)

이벤트 발생을 보고 받기 위한 function을 부하들에게 주어야 함. (props 전달)

화면에 출력할 데이터를 부하들에게 주어야 함. (props 전달)

부하들의 이벤트 보고를 받아서, 데이터를 수정하고 상태를 변경해야 함.

06 useEffect 훅

## 컴포넌트 생명 주기와 useEffect 훅

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

### 마운트(mount)

컴포넌트가 Virtual DOM 트리 구조에 처음 추가될 때.

즉 컴포넌트가 화면에 처음 보이게 되었을 때.

### 업데이트(update)

컴포넌트 렌더링 될 때

### 언마운트(unmount)

컴포넌트가 Virtual DOM 트리에서 제거될 때.

즉 컴포넌트가 화면에서 보이지 않게 되었을 때.

## useEffect 훅

### 호출 형태 #A: 어떤 값이 변경될 때 자동 호출

useEffect(콜백함수, [값\_목록])

useEffect 훅 함수의 파라미터는

1) 콜백 함수와

2) 상태 변수 값이나 props 변수 값 목록 배열이다.

위와 같이 useEffect 훅 함수가 호출되면, 다음과 같은 일을 한다.

- useEffect 함수가 호출될 때 마다 전달된 값 목록 배열을 다음과 같이 비교한다.

- useEffect 함수가 호출될 때 마다 전달된 값 목록 배열의 값 중 하나가

직전에 호출되었을 때 전달된 값과 다르면,

즉 값 목록 배열의 값 중 하나가 이전과 달라졌으면,

콜백 함수를 호출한다.

- 값 목록 배열의 값이 직전과 같으면, 아무일도 하지 않고 그냥 리턴한다.

이 기능을 활용해야 하는 경우가 종종 있다.

상태 변수 값이나 props 변수 값이 변경될 때 마다 어떤 일을 해야 하는 경우

- 디버깅을 위해서 그 값을 로그에 출력하기

- 서버에 그 변경된 값을 전송해서 DB에 저장하기

### 호출 형태 #B: mount

useEffect(콜백함수, [])

useEffect 훅 함수의 파라미터는

1) 콜백 함수와

2) 빈 배열이다.

위와 같이 useEffect 훅 함수가 호출되면, 다음과 같은 일을 한다.

- 이 훅을 호출한 컴포넌트가 이제 막 마운트 되었다면 콜백 함수를 호출한다.

- 그 다음 부터는 콜백 함수를 다시 호출하지 않는다.

이 기능을 활용해야 하는 경우가 자주 있다.

컴포넌트가 화면에 처음 보이게 되었을 때, 어떤 초기화 작업을 해야 하는 경우

- 화면 출력에 필요한 데이터를 서버에서 받아와서 리액트 상태 변수에 저장하기

- 어떤 시간 간격으로 해야하는 작업이 있다면, 타이머 시작하기

### 호출 형태 #C: unmount

useEffect(콜백함수가 리턴하는 함수, [])

만약 mount 시점의 초기화 작업을 구현했다면,

아마도 unmount 시점의 마무리 작업 구현도 필요할 것이다.

예를 들어 mount 시점에 타이머를 시작했다면, unmount 시점에 그 타이머를 종료시켜야 한다.

unmount 시점에 해야할 작업을 콜백함수의 inner 함수로 구현해서 리턴해야 한다.

콜백함수가 리턴한 함수는, 컴포넌트가 언마운트(unmount) 될 때 자동으로 호출된다.

### 호출 형태 #D: update

useEffect(콜백함수)

위와 같이 useEffect 훅을 호출하면,

컴포넌트가 업데이트(렌더링) 될 때 마다 반복해서 콜백함수가 호출된다.

이 기능을 활용해야 하는 경우가 별로 없다.