24.05.03

▼ useReducer

- useState를 대체할 수 있음
- 컴포넌트의 상태관리를 위해 기본적으로는 state를 많이 사용하나 좀 더 복잡한 상태관리가 필요한 경우 Reducer사용
- 좀 더 복잡한 프로세스 처리 가능
- state를 변경하는 부분(Count 감소, 증가 시키는 부분)이 useState는 내부에 위치하며, useReducer는 외부에 위치한다

```
import React, { useReducer } from "react";
const [state, dispatch] = useReducer(reducer, initialState
```

state

- 컴포넌트에서 사용할 상태

dispatch (함수)

- 첫번째 인자인 reducer 함수 실행
- `컴포넌트 내`에서 state의 업데이트를 일으키기 위해 사용하는 함수
- action 객체를 인자로 받으며 action은 어떤 행동인지를 나타내는 type 속성과 해당 행동과 관련된 데이터(payload)를 담고 있다
  - action type만 정의하여 사용

```
<button onClick={() => dispatch({ type: "INCREMENT" })}
```

  - action type과 데이터를 정의하여 사용

```
<button onClick={() => dispatch({ type: "INCREMENT", pa
```

reducer (함수)

- `컴포넌트 외`에서 state를 업데이트하는 함수
- 현재 state, action 객체를 인자로 받아 기존의 state를 대체하여 새로운 state반환

- 상기 dispatch 함수에 의해 실행되며 컴포넌트 외부에서 state를 업데이트 하는 로직을 담당

intitalState

- 초기 state

init

- 초기함수 (초기 state를 조금 지연해서 생성하기 위해 사용)

▼ useContext

- 리액트의 일반적인 데이터 흐름은 부모 컴포넌트에서 자식 컴포넌트로 props를 통해 단방향으로 흐름 (엄청난 큰 컴포넌트 트리가 있다고 가정할 때 공통적으로 필요한 전역적인 데이터가 있을 수 있다)
- 앱 안에서 전역적으로 사용되는 데이터를 여러 컴포넌트끼리 쉽게 공유할 수 있는 방법 제공
- props로 데이터를 일ㅇ리이 전달해 주지 않아도 해당 데이터를 가지고 있는 상 위 컴포넌트에 그 데이터가 필요한 하위 컴포넌트가 접근할 수 있다
  ⇒ 사용자 정보, 테마, 언어 등 전역적인 데이터를 전달하기에 편리
- 상위 컴포넌트의 데이터가 필요한 하위 컴포넌트들은 이를 사용해서 해당 데이터를 받아오면됨

▼ useRef

- 초기화된 변경 가능한 ref 객체 반환 → 반환된 객체는 컴포넌트의 전 생애주기를 통해 유지
- 저장공간 또는 DOM요소에 접근하기 위해 사용되는 React Hook (Ref = 참조)
- useRef로 관리하는 값은 값이 변해도 화면이 렌더링되지 않음

▼ forwardRef

- 함수형 컴포넌트는 인스턴스가 없기 때문에 ref 속성을 사용할 수 없다
- forwardRef로 감싸주게되면 ref 사용 가능
- 부모 컴포넌트에서 자식 컴포넌트 안의 DOM element에 접근하고 싶을 때 사용

1. 부모 컴포넌트에서 useRef()를 선언하고 자식 컴포넌트에 보냄

2. 자식 컴포넌트를 forwardRef()로 감싸고 부모에서 사용할 함수를 useImperativeHandle()로 감싼다
3. 부모 컴포넌트에서 current 프로퍼티를 통해 함수를 사용

▼ useImperativeHandle

- 자식 컴포넌트에서 부모 컴포넌트로 함수나 메서드를 노출하거나 커스터마이징할 때 사용
- 부모 컴포넌트는 자식 컴포넌트 내부의 특정 함수나 메서드에 직접 접근할 수 있으며 이를 활용하여 자식 컴포넌트의 인터페이스를 더 쉽게 조작하거나 노출 할 수 있음

▼ useEffect

- 컴포넌트가 렌더링 될 때마다 특정 작업을 실행할 수 있도록 함
- 클래스형 컴포넌트에서 사용할 수 있엇던 생명주기 메소드를 함수형 컴포넌트에서도 사용할 수 있게됨
- 기본형태

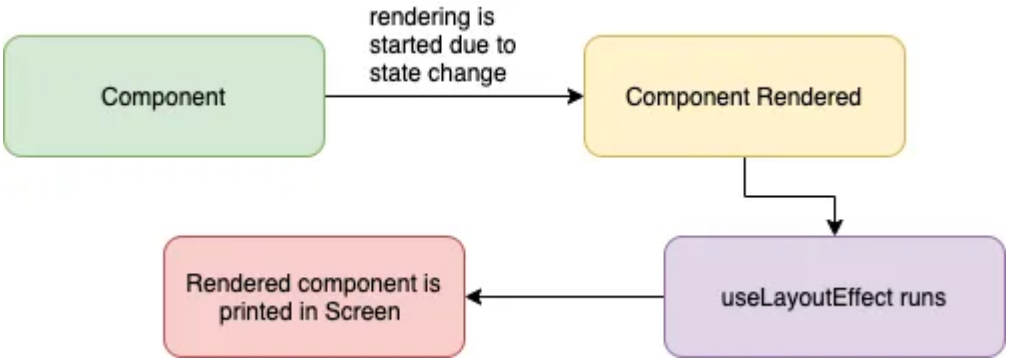
```
useEffect( function, deps )
```

function : 수행하고자 하는 작업

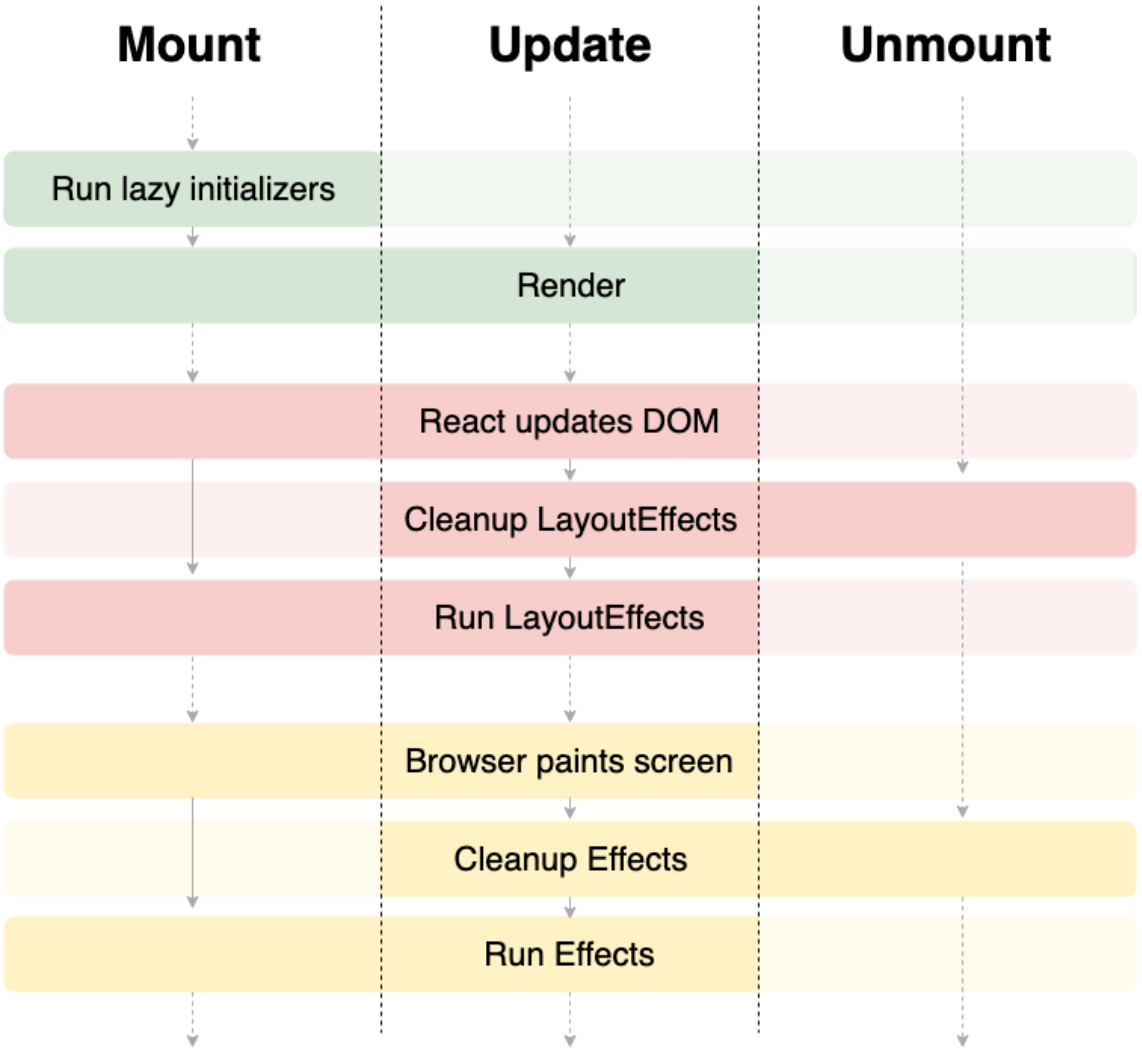
deps : 배열 형태, 검사하고자 하는 특정 값 or 빈 배열

▼ useLayoutEffect

- 컴포넌트들이 render된 후 실행되며 그 이후에 paint가 됨
- 동기적으로 실행
- dom을 조작하는 코드가 존재하더라고 사용자는 깜빡임을 경험하지 않는다
- 로직이 복잡할 경우 사용자가 레이아웃을 보는데까지 시간이 오래걸린다는 단점



▼ useInsertionEffect



- useEffect와 형태와 사용법은 동일하나 호출 순서에 차이
- DOM 업데이트 → LayoutEffect실행 → DOM이 화면에 그려짐 → Effect실행
- 화면이 그려지기 전에 state가 설정되어야 할 때 사용 ⇒ DOM변경 전 동기적으로 스타일을 주입하는데 사용
- 서버사이드에서는 실행되지않고 클라이언트사이드에서만 실행

- CSS-in-JS 라이브러리를 사용하거나 렌더링 전 다크모드 여부를 결정할 때 사용한다
- useInsertionEffect는 컴포넌트에서 다른 Effect가 실행될 때 이미 실행되어 있음이 보장되므로 style이 주입된 상태에서 Effect 실행 가능

▼ useMemo

- useMemo에서 Memo라는 말은 Memoization을 뜻한다
- 동일한 값ㅇ르 리턴하는 함수를 반복적으로 호출할 경우 맨 처음 값을 메모리에 저장해서 필요할 때마다 또 다시 계산하지 않고 메모리에서 꺼내서 재사용을 하는 기법
  ⇒ 자주 필요한 값을 맨 처음 계산할 때 캐싱 해놓아서 그 값이 필요할 때마다 다시 계산을 하는 것이 아니라 캐시에서 꺼내서 사용하는 것

```
const value = useMemo(() => {
    return calculate();
}. [item])
```

- 콜백 함수, 배열 두개의 인자를 받는다
  - 콜백 함수 : 메모이제이션을 해줄 값을 계산해서 리턴해주는 함수 → 콜백 함수가 리턴하는 값이 useMemo가 리턴하는 값
  - 배열 (의존성 배열) : useMemo는 배열 안의 요소의 값이 업데이트될 때만 콜백 함수를 다시 호출해서 메모이제이션 된 값을 업데이트해서 다시 메모이제이션을한다
    빈 배열을 넘겨주면 컴포넌트가 처음 마운트 되었을 때만 값을 계산하고 이후에는 항상 메모이제이션된 값을 꺼내와서 사용한다
- 값을 재활용하기 위해서 따로 메모리를 소비해서 저장하는 것이기 때문에 불필요한 값까지 전부 모아 메모이제이션을 하면 성능에 좋지않다

▼ useCallback

- 첫번째 인자로 전달된 함수를 메모이제이션(저장)하고 컴포넌트가 재랜더링될 때마다 새로 생성되지않고 의존성 배열에 명시된 값이 변경될 때만 함수가 새로 생성됨
  → 불필요한 컴포넌트의 재랜더링 방지
  ⇒
  메모이제이션 기법으로 컴포넌트 성능을 최적화 시켜주는 도구

- 인자로 전달한 콜백 함수 그 자체를 메모이제이션

▼ useTransition

- UI를 차단하지 않고 상태를 업데이트 할 수 있는 리액트 훅
- useTransition은 컴포넌트 최상위 수준에서 호출되어 startTransition을 통해 우선순위가 낮은 상태 업데이트(setState)들을 transition이라고 표시
  - startTransition는 동기함수여야 한다
  - transition으로 표시도니 setState는 다른 setState 업데이트시 중단된다 → 다른 상태 업데이트가 있을 경우 그것을 먼저 처리
  - 텍스트 입력을 제어하는 데 사용할 수 없다
- 리액트는 UI 렌더링시 우선순위에 따라 업데이트 할 수 있게 된다

▼ use

- Promise를 정의하는 데 필요한 상용구 코드의 양을 줄여 데이터 가져오기 프로세스를 단순화하도록 설계된 솔루션
- 조건, 블록, 루프 내부에서 호출할 수 있는 특별한 특성을 가지고있다
- 더 많은 구성 요소를 추가하지 않고 논리 및 조건부 흐름을 추가하려는 경우 유연한 Hook이 된다

▼ useDebugValue

- React DevTools에서 사용자 지정 hook에 label을 추가할 수 있는 React Hook
- 아무것도 return하지 않는다

```
useDebugValue(value, format?)
```

value

- React DevTools에 보여주고 싶은 값으로, 어떤 타입이든 가능

optional format

- formatting 함수

- 컴포넌트를 검사할 때, React DevTools는 value를 매개변수로 받아 이 formatting 함수를 호출
- formatting된 value를 return
- 함수를 작성하지 않은 경우에는 value 그 자체가 보여짐

▼ UseId

- useRef를 사용하여 이전 ID를 저장하고, 다음 ID를 계산하여 반환
  → 생성된 ID는 컴포넌트의 내부에서만 사용되며 외부에 노출되지 X
- 주의할 점은 `useId Hook의 반환값을 key를 위해서는 사용하지 않아야함`