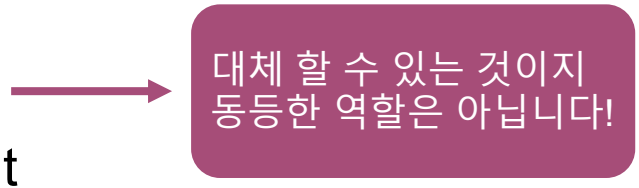
Hooks & Context

Basic hooks

- useState
  - state를 대체
- useEffect
  - 라이프 사이클 훅을 대체
    - componentDidMount
    - componentDidUpdate
    - componentWillUnmount
- useContext



과거: function 컴포넌트에서 state사용 X

현재: function 컴포넌트에서 state를 사용 O

why function component?

- Class는 컴포넌트 사이에서 상태와 관련된 로직을 재사용하기 어려움
- Class 에서 복잡한 컴포넌트들은 이해하기 어려움
- Class는 컴파일 단계에서 코드 최적화가 어려움
- Class는 컴포넌트의 생명주기 사이에 this.state를 공유한다
  - 하지만 생명주기 사이에 바뀐 state를 정확하게 표현해야 할때는 중요

custom hooks

- 공통으로 사용 가능한 나만의 훅을 만듬

Additional Hooks

- useReducer
  - 다수의 하윗값을 포함하는 복잡한 정적 로직을 만드는 경우
  - 다음 state가 이전 state에 의존적인 경우
  - Redux를 안다면 쉽게 사용 가능
- useMemo, useCallback
  - 성능 최적화
- useRef
  - 성능 최적화

class 컴포넌트는 render가 되면서 유지가 되지만 function 컴포넌트는 새로 생성되는 경향이 있기 때문에 성능 최적화가 필요.

컴포넌트 간 통신

- 하위 컴포넌트 변경
- 상위 컴포넌트 변경

props를 이용해서 변경,
다만 컴포넌트가 많아지면 복잡도가 증가하면서
문제점이 생긴다 -> 그래서 등장한게 context

Context

하위 컴포넌트 전체에 데이터를 공유하는 법

- 데이터를 Set(제공자)
  - 가장 상위 컴포넌트(프로바이더)
- 데이터를 Get
  - 모든 하위 컴포넌트에서 접근 가능
  - 컨슈머로 하는 법(class, function 2개다 사용 가능)
  - 클래스 컴포넌트의 this.context 로 하는 법
  - 펑셔널 컴포넌트의 useContext 로 하는 법(가장 많이 쓰임)