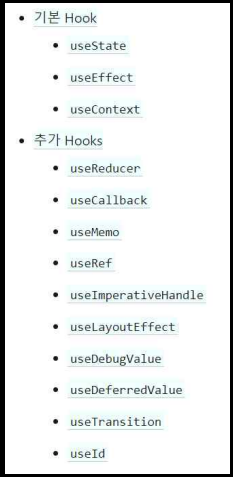
React Hook 소개

React.js

목차

1. 기본 Hook
-useState
-useEffect
-useContext
2. 추가 Hook
-useRef
-useMemo
-useReducer

https://ko.legacy.reactjs.org/docs/hooks-reference.html

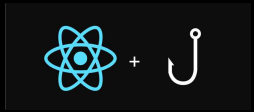


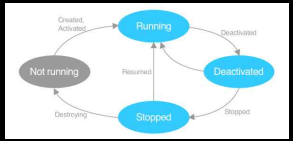
Hook을 사용하는 이유

React Hooks는 함수형 컴포넌트에서 상태(state)와 생명주기(lifecycle) 기능을 사용할 수 있도록 도와주는 기능입니다.

클래스형 컴포넌트에서는 state와 라이프사이클 메서드를 사용했지만, 함수형 컴포넌트에서는 Hooks를 사용하여 같은 기능을 구현할 수 있습니다.

이를 통해 코드의 재사용성을 높이고 상태 관리, 부작용 처리 등을 더 간편하게 할 수 있습니다.





useState

-react hooks

useState는 함수 컴포넌트에서 **상태**를 관리하기 위해 사용됨.

useState를 사용하면 함수 컴포넌트 내에서 상태 변수를 생성하고 해당 상태를 갱신할 수 있음.

이 함수는 배열을 반환하며, 첫 번째 요소는 현재 상태의 값이고, 두 번째 요소는 상태를 갱신하는 함수.

useState

-react hooks

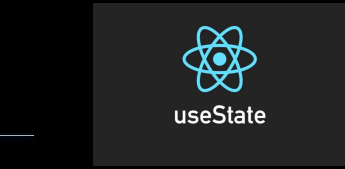
// const [변수, 함수] = useState(초기값)

useState를 선언하는 방법

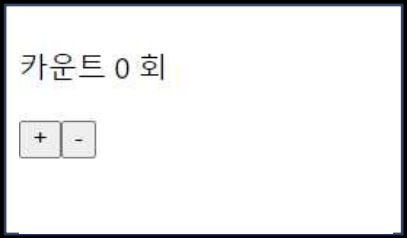
```
const [count, setCount] = useState(0);
```

useState

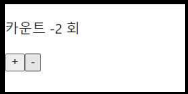
-react hooks




```jsx
import { useState } from 'react';

function App() {
  const [count, setCount] = useState(0);
  console.log({ count });

  return (
    <div className="App">
      <p>카운트 {count} 회</p>
      <button onClick={() => setCount(count + 1)}>+</button>
      <button onClick={() => setCount(count - 1)}>-</button>
    </div>
  );
}

export default App;
```


카운트 0 회
+
-


카운트 2 회
+
-


카운트 -2 회
+
-


{count: 0}
{count: 0}
{count: 1}
{count: 1}
{count: 2}
{count: 2}


{count: 0}
{count: 0}
{count: -1}
{count: -1}
{count: -2}
{count: -2}

useEffect

-react hooks

useEffect는 리액트 함수 컴포넌트에서 side effects를 수행하기 위해 사용되는 훅.

side effects란 주로 데이터 가져오기, DOM 조작 등과 같은 작업.

useEffect는 컴포넌트가 렌더링되고 난 후에 실행되는 코드를 작성할 때 사용됩니다.

useEffect

-react hooks

// useEffect = (() => {

....

},[])

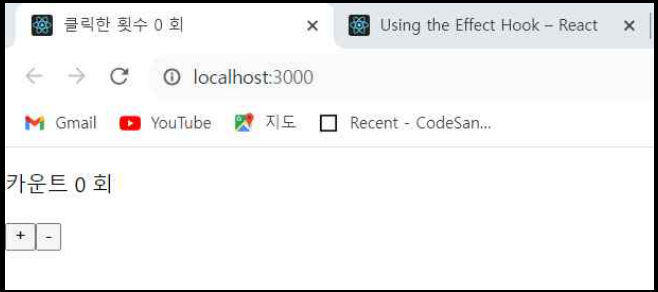
useEffect를 선언하는 방법

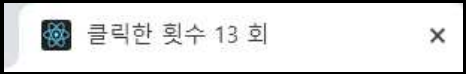
```
useEffect(() => {
  // 브라우저 API를 이용하여 문서 타이틀을 업데이트합니다.
  document.title = `You clicked ${count} times`;
});
```

useEffect

-react hooks

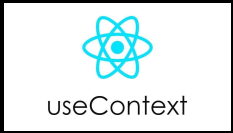

```jsx
import { useState, useEffect } from 'react';

function App() {
  const [count, setCount] = useState(0);
  console.log({ count });

  useEffect(() => {
    // 브라우저 API를 이용하여 문서 타이틀을 업데이트합니다.
    document.title = `클릭한 횟수 ${count} 회`;
  });

  return (
    <div className="App">
      <p>카운트 {count} 회</p>
      <button onClick={() => setCount(count + 1)}>+</button>
      <button onClick={() => setCount(count - 1)}>-</button>
    </div>
  );
}

export default App;
```


클릭한 횟수 0 회
Using the Effect Hook – React
localhost:3000
Gmail
YouTube
지도
Recent - CodeSan...
카운트 0 회
+
-


클릭한 횟수 13 회

useContext

-react hooks



useContext는 컴포넌트 트리 안에서 전역으로 상태를 공유하기 위해 사용됨.

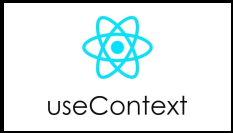
주로 상태 관리를 위한 컨텍스트(Context) API와 함께 사용됩니다.

컨텍스트는 컴포넌트 트리 안에서 전역적으로 데이터를 공유할 수 있는 방법을 제공함.

useContext를 사용하면 컨텍스트로 제공된 값을 현재 컴포넌트에서 손쉽게 사용할 수 있음.

useContext

-react hooks



// const 변수 = useContext(컨텍스트)

useState를 선언하는 방법

```
const MyContext = createContext();

const contextValue = useContext(MyContext);
```

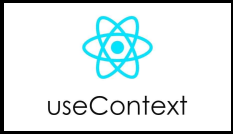
useContext를 사용하기 위해서는 먼저 React.createContext를 사용하여 컨텍스트를 생성해야 함.

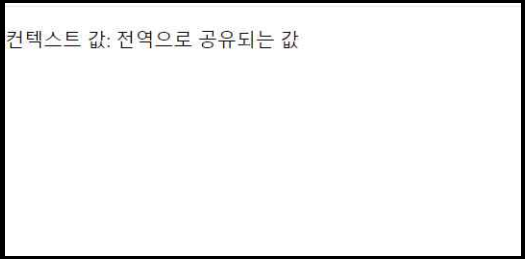
그 후에 컨텍스트를 제공하는 컴포넌트(Context.Provider)를 사용하여 값을 설정하고, useContext를 호출하여 해당 값을 가져와서 사용할 수 있음.

useContext

-react hooks

```jsx
import React, { createContext, useContext } from 'react';

// 컨텍스트 생성
const MyContext = createContext();

// 컨텍스트를 제공하는 컴포넌트
function MyProvider({ children }) {
  const sharedValue = '전역으로 공유되는 값';

  return (
    <MyContext.Provider value={sharedValue}>{children}</MyContext.Provider>
  );
}

// 값을 사용하는 컴포넌트
function MyComponent() {
  // useContext를 사용하여 컨텍스트의 값을 가져옴
  const contextValue = useContext(MyContext);

  return <p>컨텍스트 값: {contextValue}</p>;
}

// MyProvider를 사용하여 컨텍스트를 제공하는 상위 컴포넌트
function App() {
  return (
    <MyProvider>
      <MyComponent />
    </MyProvider>
  );
}

export default App;
```




컨텍스트 값: 전역으로 공유되는 값

useRef

-react hooks

useRef는 함수 컴포넌트에서 DOM 요소에 접근하거나 값을 기억하기 위해 사용됨.

useRef를 사용하면 함수 컴포넌트 내에서 DOM 요소에 직접 접근하거나, 컴포넌트 간에 데이터를 공유하며 상태를 유지할 수 있음

useRef

-react hooks

useRef()

// const 변수 = useRef(초기값)

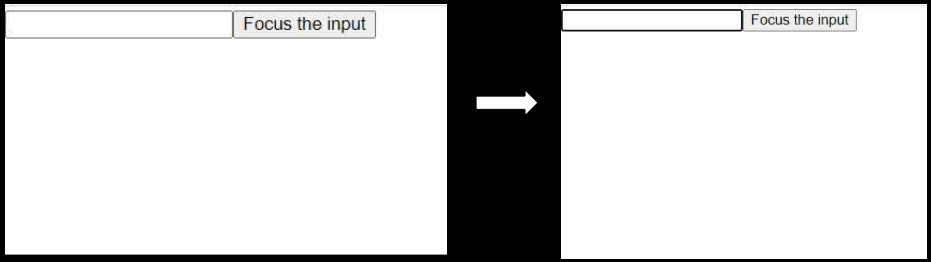
useRef를 선언하는 방법

```
import React, { useRef } from 'react';

const inputEl = useRef(null);
```

useRef

-react hooks

```jsx
import React, { useRef } from 'react';

function App() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` points to the mounted text input element
    inputEl.current.focus();
  };

  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </>
  );
}

export default App;
```


Focus the input
Focus the input

useReducer

-react hooks

useReducer는 컴포넌트의 상태를 관리하고 업데이트하는 데 사용.

useReducer는 주로 복잡한 상태 로직을 다룰 때 유용하며, 상태를 업데이트하는 로직을 외부로 분리하여 관리할 수 있음.

useReducer는 상태와 액션을 입력으로 받아 새로운 상태를 반환하는 함수(reducer)를 사용함.
이렇게 함으로써 상태 업데이트 로직이 컴포넌트 내부에 직접 존재하지 않고, 외부 함수로 분리되어 더 모듈화되고 관리하기 쉬워집니다.

useReducer

-react hooks

// const [state, dispatch] = useReducer(reducer함수, 초기값);

useReducer를 선언하는 방법

```
import React, { useReducer } from 'react';

const [state, dispatch] = useReducer(counterReducer, { count: 0 });
```

useReducer

-react hooks


```jsx
import React, { useReducer } from 'react';

// reducer 함수
const counterReducer = (state, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    default:
      return state;
  }
};
function App() {
  // useReducer를 사용하여 상태와 dispatch 함수 획득
  const [state, dispatch] = useReducer(counterReducer, { count: 0 });

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'INCREMENT' })}>증가</button>
      <button onClick={() => dispatch({ type: 'DECREMENT' })}>감소</button>
    </div>
  );
}

export default App;
```

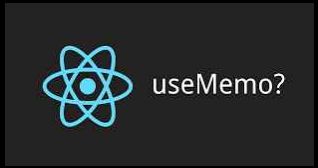

Count: 0
증가
감소


Count: -1
증가
감소


Count: 1
증가
감소

useMemo

-react hooks



useMemo는 성능 최적화를 위해 사용됨.
이 훅은 계산 비용이 많이 드는 함수의 결과를 메모이제이션하고, 의존성이 변경될 때만 함수를 다시 계산함.
이를 통해 렌더링 성능을 향상시킬 수 있음.

일반적으로 useMemo는 렌더링 중에 계산이 반복되는 경우에 사용됨.
예를 들어, 어떤 계산 결과를 컴포넌트의 렌더링이 변경되지 않는 한 동일하게 유지하고 싶을 때 useMemo를 사용할 수 있음.

useMemo

-react hooks

// const [state, dispatch] = useReducer(reducer함수, 초기값);

useReducer를 선언하는 방법

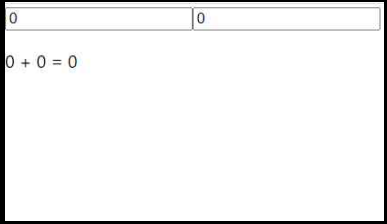
```
import React, { useReducer } from 'react';

const [state, dispatch] = useReducer(counterReducer, { count: 0 });
```

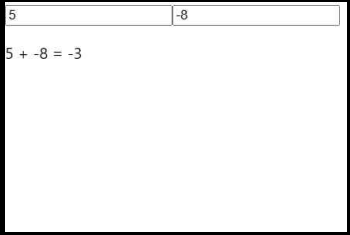
useMemo

-react hooks

```jsx
import React, { useReducer } from 'react';

// reducer 함수
const counterReducer = (state, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    default:
      return state;
  }
};
function App() {
  // useReducer를 사용하여 상태와 dispatch 함수 획득
  const [state, dispatch] = useReducer(counterReducer, { count: 0 });

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'INCREMENT' })}>증가</button>
      <button onClick={() => dispatch({ type: 'DECREMENT' })}>감소</button>
    </div>
  );
}

export default App;
```

```jsx
  return (
    <div>
      {/* 입력값과 상태를 연결하여 숫자를 입력할 수 있는 input 요소를 생성합니다. */}
      <input
        type="number"
        value={numberA}
        onChange={(e) => setNumberA(Number(e.target.value))}
      />
      <input
        type="number"
        value={numberB}
        onChange={(e) => setNumberB(Number(e.target.value))}
      />

      {/* AddNumbers 컴포넌트를 렌더링하고, 현재의 numberA와 numberB를 prop으로 전달합니다. */}
      <AddNumbers a={numberA} b={numberB} />
    </div>
  );
}

export default App;
```


0
0
0 + 0 = 0


5
-8
5 + -8 = -3

감사합니다