React Hook 강의(1)

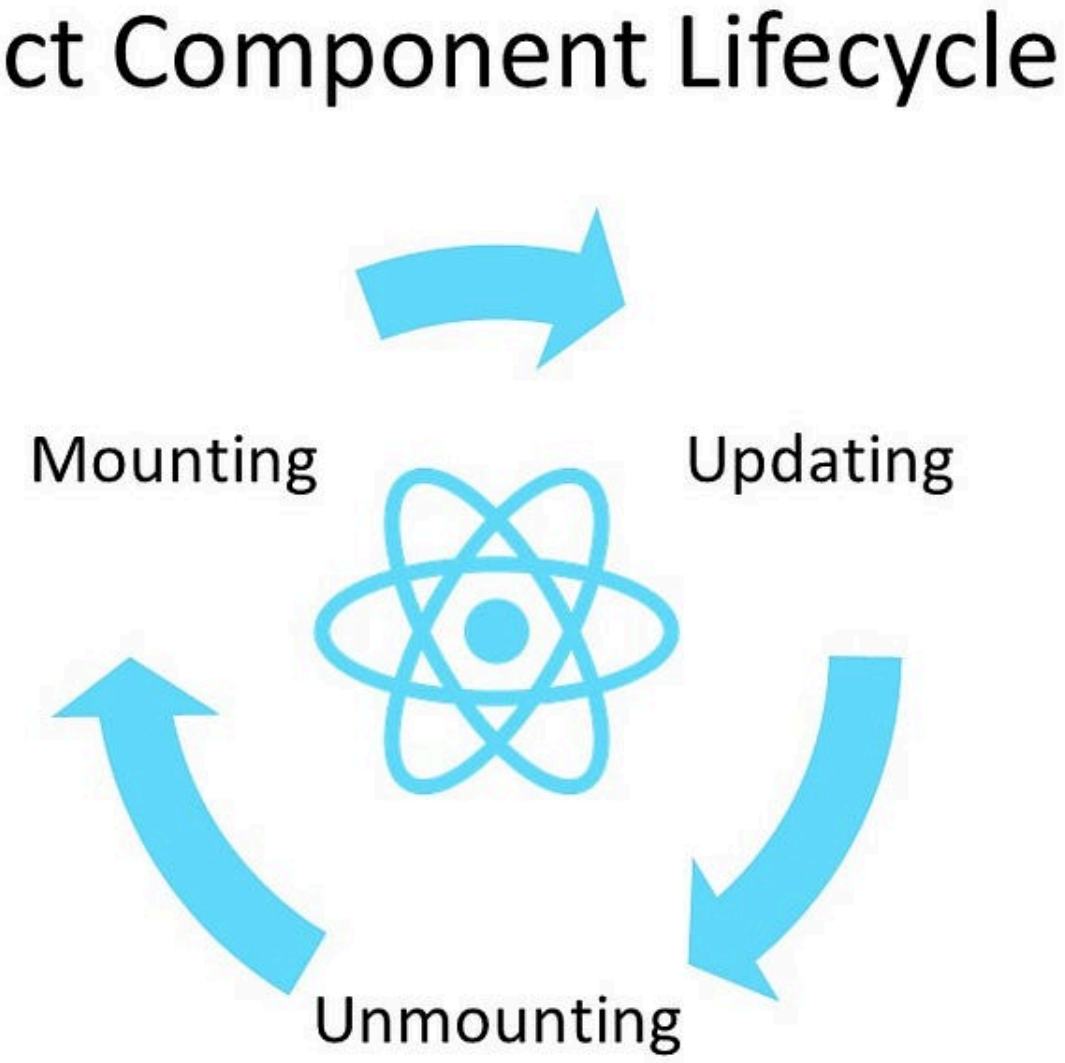
useState, useRef, useEffect, useMemo

~ by tunalee

React Hook 이란?

- 함수형 컴포넌트에서 라이프사이클 기능을 쉽게 사용할 수 있도록 해주는 함수
- 함수형 컴포넌트에서 상태를 관리할 수 있도록 해주는 함수

Life Cycle



useState

- UI에 영향을 미치는 데이터
- State가 변경되면 컴포넌트가 다시 렌더링됨

useState (count 값이 증가할 때마다 Count 컴포넌트 호출)

```
const Count = () => {
    const [stateCount, setStateCount] = useState(0);
    const handleStateCount = () => {
        setStateCount(stateCount + 1);
        console.log('state: ', stateCount);
    }
    return (
        <div>
            <button onClick={handleStateCount}>state count: {stateCount}</button>
        </div>
    )
}
export default Count
```

useState vs 변수

```
const Count = () => {
    const [stateCount, setStateCount] = useState(0);
    let varCount = 0;
    const handleStateCount = () => {
        setStateCount(stateCount + 1);
        console.log('state: ', stateCount);
    }
    const handleVarCount = () => {
        varCount++;
        console.log('var: ', varCount);
    }
    return (
        <div>
            <button onClick={handleStateCount}>state count: {stateCount}</button>
            <button onClick={handleVarCount}>var count:{varCount}</button>
        </div>
    )
}
export default Count
```

useRef

- State가 변경되어도 컴포넌트가 렌더링 되지 않음
- 렌더링이 되어도 값은 유지 됨

```
{current: ...}
```

useRef (count 값이 변경될 때마다 렌더링은 되지 않고, 값은 바뀜)

```
const Count = () => {
    const refCount = useRef(0);

    const handleRefCount = () => {
        refCount.current = refCount.current + 1;
        console.log('ref: ', refCount.current);
    }
    return (
        <div>
            <button onClick={handleRefCount}>ref count:{refCount.current}</button>
        </div>
    )
}
export default Count
```

useEffect

```
useEffect(() => {
  console.log('컴포넌트가 화면에 나타남');

  return () => {
    console.log('컴포넌트가 화면에서 사라짐');
  };
}, [특정 상태나 props]);
```

useEffect

```
useEffect(() => {
      console.log('useEffect가 실행 되었습니다.')
});
```

```
useEffect(() => {
      console.log('useEffect가 실행 되었습니다.')
},[]);
```

useEffect (렌더링 될 때마다 실행)

```
import './App.css'
import React, {useEffect, useState} from "react";

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

  useEffect(() => {
        console.log('useEffect가 실행 되었습니다.')
  });
  const handleUpdate = () => {
    setCount(count + 1)
  }
  return (
    <div>
      <button onClick={handleUpdate}>{count}</button>
    </div>
  )
}

export default App
```

useEffect (마운트 때만 실행)

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

  useEffect(() => {
        console.log('useEffect가 실행 되었습니다.')
  },[]);
  const handleUpdate = () => {
    setCount(count + 1)
  }
  return (
    <div>
      <button onClick={handleUpdate}>{count}</button>
    </div>
  )
}
export default App
```

useEffect (dependency array 값이 변경 될 때)

```
function App() {
  const [text, setText] = useState("");

  useEffect(() => {
        console.log('useEffect가 실행 되었습니다.')
  },[text]);
  return (
    <div>
      <input
          type="text"
          onChange={(e) => setText(e.target.value)}
          value={text}
      />
    </div>
  )
}
export default App
```

useEffect (clean up 을 안 해줄 때)

```
function Timer() {
    const [seconds, setSeconds] = useState(0);

    useEffect(() => {
        const time = setInterval(() => {
            setSeconds(prevSeconds => prevSeconds + 1);
            console.log(seconds);
        }, 1000);

    }, []);
    return (
        <div>
            <h1>{seconds}초 경과</h1>
        </div>
    );
}
export default Timer;
```

useEffect (clean up 을 안 해줄 때)

```
function App() {
  const [showTimer, setShowTimer] = useState(false);
  return (
    <div>
      {showTimer && (<Timer />)}
      <button onClick={()=>setShowTimer(!showTimer)}>toggle timer</button>
    </div>
  )
}
```

useEffect (clean up 을 해줄 때)

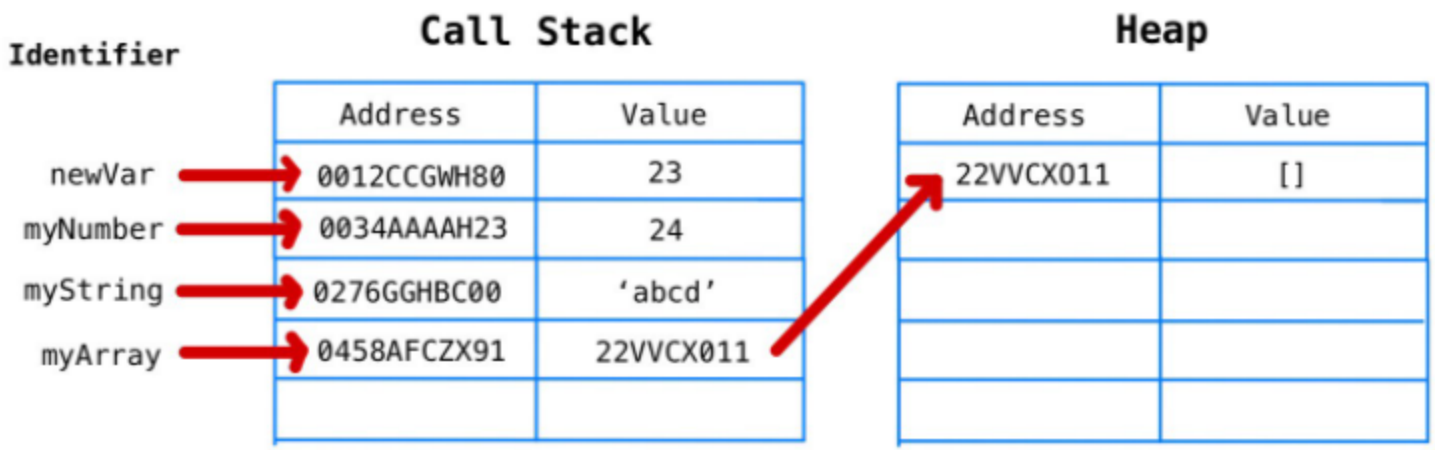
```
function Timer() {
    const [seconds, setSeconds] = useState(0);

    useEffect(() => {
        const time = setInterval(() => {
            setSeconds(prevSeconds => prevSeconds + 1);
            console.log(seconds);
        }, 1000);
        return () => clearInterval(time);
    }, []);
    return (
        <div>
            <h1>{seconds}초 경과</h1>
        </div>
    );
}
export default Timer;
```

useEffect (의존성 배열에 객체 따위의 참조값이 있을 때)

```
function App() {
    const [isOn, setIsOn] = useState(false);
    const obj = {
        name: isOn ? 'on': 'off'
    }
    const [count, setCount] = useState(0)

    useEffect(() => {
        console.log('useEffect가 실행 되었습니다.')
    }, [obj])
    return (
        <div>
            <button onClick={()=>(setCount(count+1))}>{count}</button>
            <button onClick={()=>(setIsOn(!isOn))}>{obj.name}</button>
        </div>
    )
}
export default App
```


Identifier
Call Stack
Address
Value
newVar
0012CCGWH80
23
myNumber
0034AAAAH23
24
myString
0276GGHBC00
'abcd'
myArray
0458AFCZX91
22VVCX011
Heap
Address
Value
22VVCX011
[]

useMemo

- 캐싱을 통헤 렌더링 할 때마다 계속 실행되지 않도록 할 수 있음
- 컴포넌트 내 동일 계산을 여러번 안 하도록 방지

useMemo

```
useMemo(()=>{
    ...
}, [])
```

useMemo

```
function App() {
    const [isOn, setIsOn] = useState(false);
    const obj = useMemo(() => {
        return { name: isOn ? 'on' : 'off' };
    }, [isOn]);
    const [count, setCount] = useState(0)

    useEffect(() => {
        console.log('useEffect가 실행 되었습니다.')
    }, [obj])
    return (
        <div>
            <button onClick={()=>(setCount(count+1))}>{count}</button>
            <button onClick={()=>(setIsOn(!isOn))}>{obj.name}</button>
        </div>
    )
}

export default App
```

useMemo (메모이제이션 적용 전)

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

    const expensiveCalculation =() => {
        console.log('비용이 많이 드는 계산 중...');
        let result = 0;
        for (let i = 0; i < 1000000000; i++) {
            result += i;
        }
        return result;
    }

    return (
        <div>
            <h1>값: {expensiveCalculation()}</h1>
            <button onClick={() => setCount(count + 1)}>카운트: {count}</button>
        </div>
    );
}

export default Expensive;
```

useMemo (메모이제이션 적용 후)

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

    const expensiveCalculation =useMemo(() => {
        console.log('비용이 많이 드는 계산 중...');
        let result = 0;
        for (let i = 0; i < 1000000000; i++) {
            result += i;
        }
        return result;
    }, [])

    return (
        <div>
            <h1>값: {expensiveCalculation}</h1>
            <button onClick={() => setCount(count + 1)}>카운트: {count}</button>
        </div>
    );
}
export default Expensive;
```

실습

```
function App() {
    useEffect(() => {
        console.log('날 깨우지 말고 10번 count 하도록 ㅡㅡ^')
    });
    ...
}
```