렌더링

언제 렌더링 되는가?
컴포넌트가 언제 렌더링 되는지 알아보기 위한 예제를 구현하자.

src/CounterA.jsx 생성

```
import React, { useState } from 'react'

function CounterA() {
  console.log("CounterA 호출됨");
  const [value, setValue] = useState(0);
  return (
    <div className="box">
      <h1>CounterA</h1>
      <button onClick={() => setValue(value + 1)}>
        { value }
      </button>
    </div>
  )
}

export default CounterA;
```

console.log("CounterA 호출됨");

CountA 함수형 컴포넌트가 렌더링 될 때 마다 위 코드가 실행될 것이다.

const [value, setValue] = useState<number>(0);

useState 함수로 상태를 생성할 때, 위와 같이 값의 타입을 적어주는 것이 좋다.

src/CounterB.jsx 생성

```
import React, { useState } from 'react'

function CounterB() {
  console.log("CounterB 호출됨");
```

```
  const [value, setValue] = useState(0);
  return (
    <div className="box">
      <h1>CounterB</h1>
      <button onClick={() => setValue(value + 1)}>
        { value }
      </button>
    </div>
  )
}

export default CounterB;
```

src/GroupComponent.jsx 생성

```
import React, { useState } from 'react'
import CounterA from './CounterA'
import CounterB from './CounterB'

function GroupComponent() {
  console.log('GroupComponent 호출됨');
  const [value, setValue] = useState(0);
  return (
    <div className='box'>
      <h1>GroupComponent</h1>
      <button onClick={() => setValue(value + 1)}>
        { value }
      </button>
      <CounterA />
      <CounterB />
    </div>
  )
}

export default GroupComponent;
```

src/App.jsx 수정

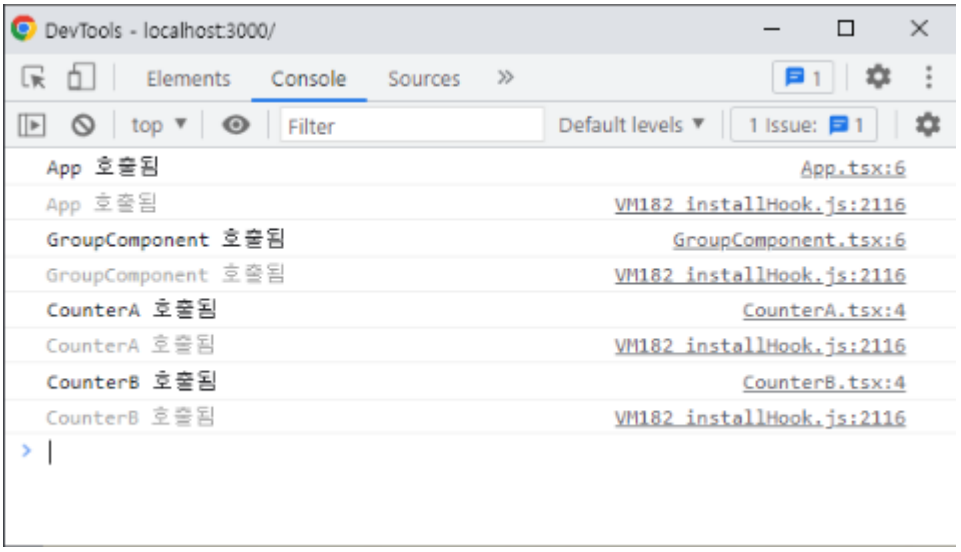
```
import React, { useState } from 'react'
import './App.css'
import GroupComponent from './GroupComponent';

function App() {
  console.log('App 호출됨');
  const [value, setValue] = useState(0);
  return (
    <div className="box">
      <h1>App</h1>
      <button onClick={() => setValue(value + 1)}>
        { value }
      </button>
      <GroupComponent />
    </div>
  );
}

export default App;
```

src/App.css 수정

```
h1 { font-size: 11pt; margin: 0; }
div.box { display: inline-block; margin: 10px; padding: 10p
x; border: 1px solid gray; }
button { display: block; padding: 0.2em 2em; margin: 0.5em;
}
```

실행

웹브라우저 개발자창의 콘솔 탭을 열고, 새로고침 하자.
새로고침 하면 리액트앱이 재시작하게 된다.

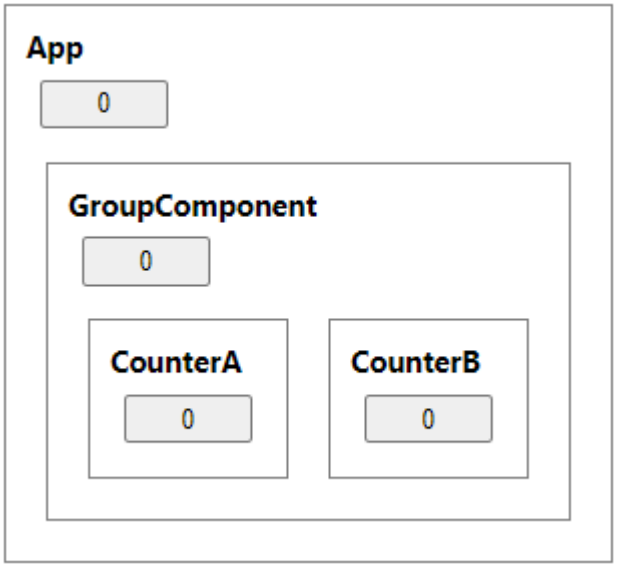


위 창에서 회색으로 출력된 메시지 부분은 무시하고, 검은색 메시지 출력만 보자.

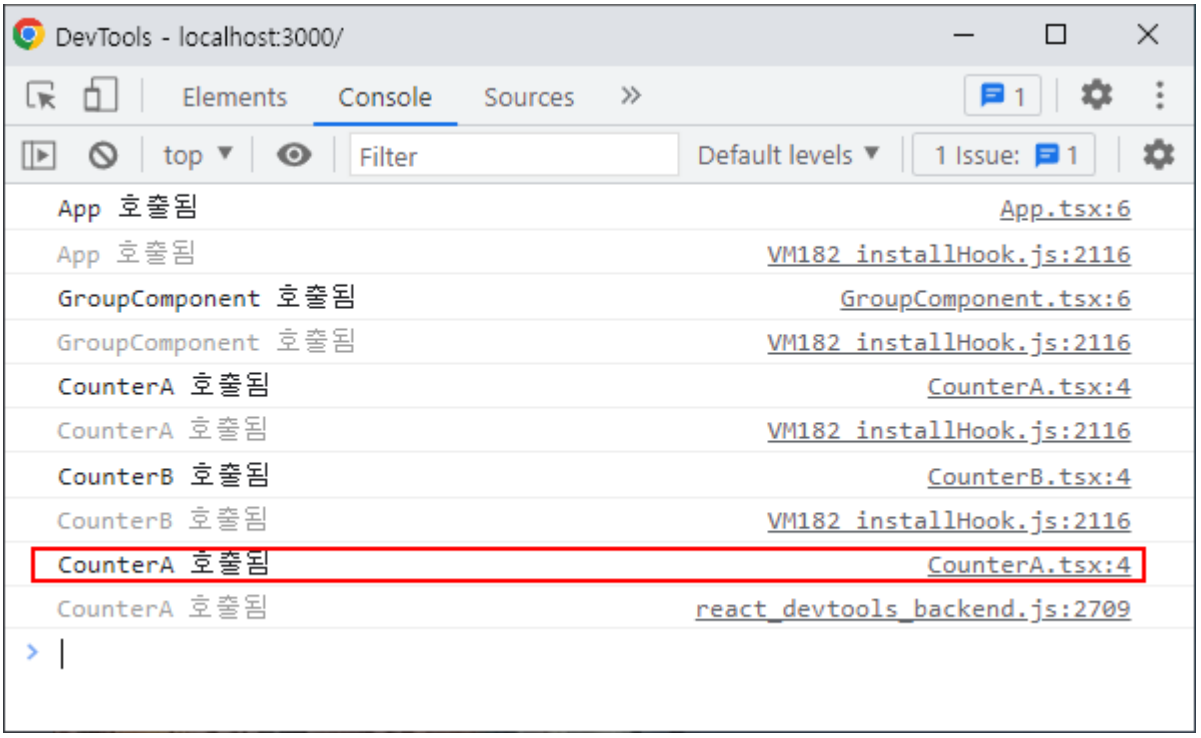
참고 : index.tsx 파일의 <React.StrictMode> 컴포넌트는 오류를 찾고 경고를 해주는 도구이다.

이 컴포넌트에 의해 회색 부분이 출력됨.
프로젝트를 빌드할 때는 이 컴포넌트가 무시되어 배포 파일에 포함되지 않는다.

처음 렌더링 될 때, 모든 컴포넌트들이 호출되었다. (모든 컴포넌트들이 렌더링 되었다)



CounterA 컴포넌트의 버튼을 클릭하자.



CounterA 컴포넌트 함수만 호출되었다. 즉 CounterA 컴포넌트만 렌더링 되었다.

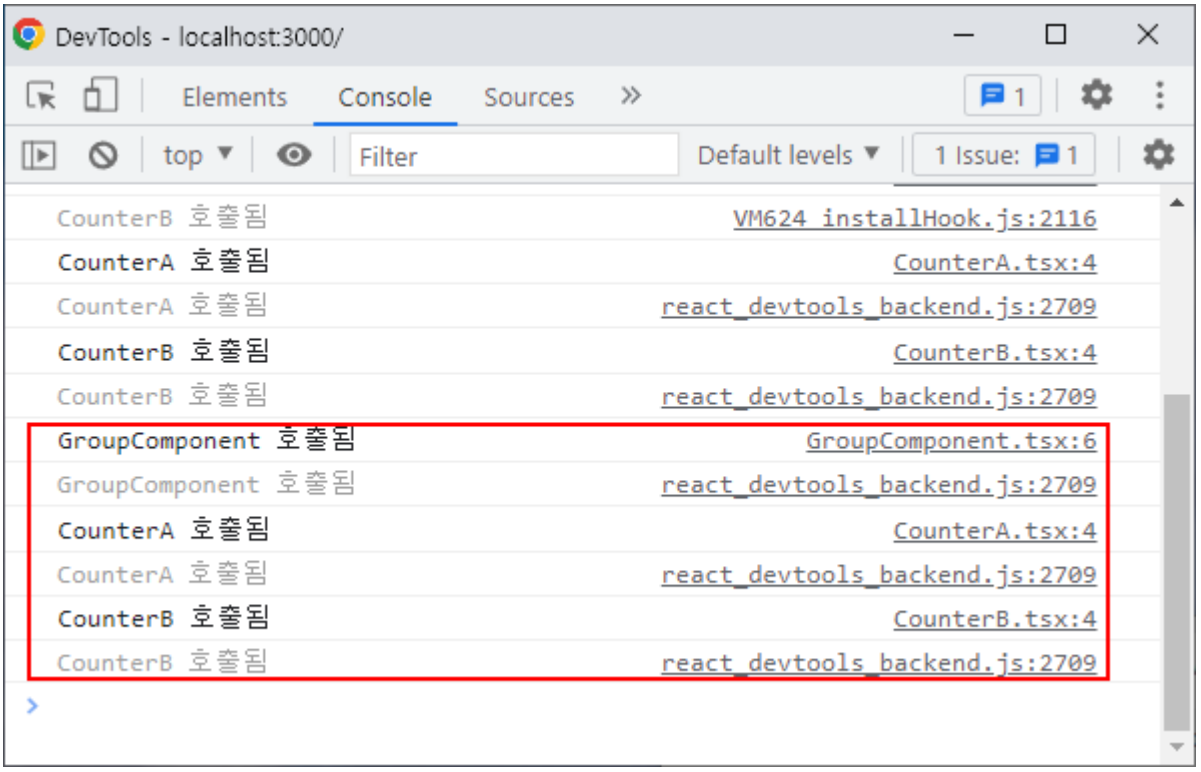
리액트 상태가 변경되어 렌더링 될 때,
상태가 변경된 컴포넌트만 렌더링 된다는 것을 알 수 있다.

실습

CounterB 컴포넌트의 버튼을 클릭하면 CounterB 컴포넌트만 렌더링 될 것이다.
확인해 보자.

실습

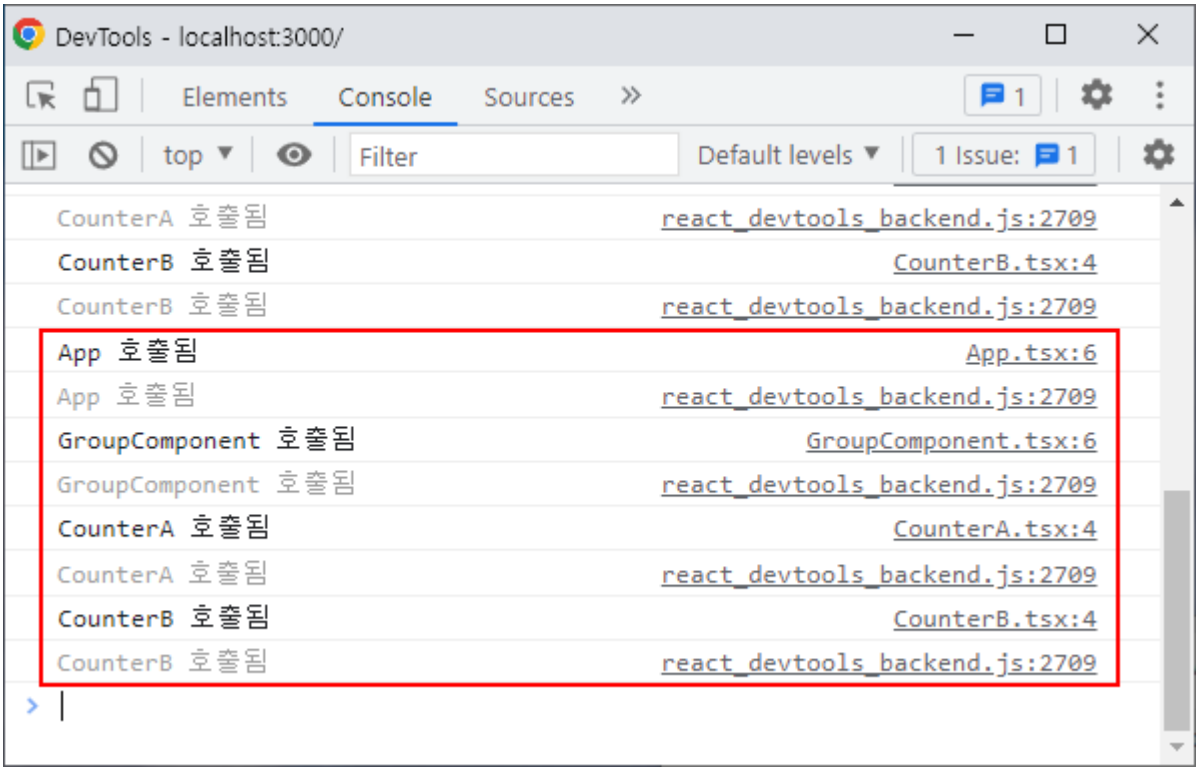
GroupComponent 컴포넌트 버튼을 클릭하자



GroupComponent 컴포넌트와 그 자식 컴포넌트들이 렌더링 되었다.

실습

App 컴포넌트 버튼을 클릭하자



요약

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.

소스코드 파일 하나에 여러 컴포넌트 구현 가능

소스코드 파일 하나에 여러 함수형 컴포넌트를 구현해도 된다.

src/GoupComponent.jsx 수정

```
import React, { useState } from 'react'

function CounterA() {
  console.log("CounterA 호출됨");
  const [value, setValue] = useState(0);
  return (
    <div className="box">
```

```
      <h1>CounterA</h1>
      <button onClick={() => setValue(value + 1)}>
        { value }
      </button>
    </div>
  )
}

function CounterB() {
  console.log("CounterB 호출됨");
  const [value, setValue] = useState(0);
  return (
    <div className="box">
      <h1>CounterB</h1>
      <button onClick={() => setValue(value + 1)}>
        { value }
      </button>
    </div>
  )
}

function GroupComponent() {
  console.log('GroupComponent 호출됨');
  const [value, setValue] = useState(0);
  return (
    <div className='box'>
      <h1>GroupComponent</h1>
      <button onClick={() => setValue(value + 1)}>
        { value }
      </button>
      <CounterA />
      <CounterB />
    </div>
  )
}

export default GroupComponent;
```

위와 같이 한 파일에 구현해도, 상태가 변경된 컴포넌트만 렌더링 되는 것은 전과 같다.

언제 렌더링 안되는가? #1

App.jsx 수정

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

function App() {
  const [arr, setArr] = useState([0, 0]);
  const increaseAt = (index) => {
    arr[index] = arr[index] + 1;
    setArr(arr);
  };
  return (
    <div className="box">
      <h1>App</h1>
      <p>{ arr.join(", ") }</p>
      <button onClick={() => increaseAt(0)} >a[0]++</button
>
      <button onClick={() => increaseAt(1)} >a[1]++</button
>
    </div>
  );
}

export default App;
```

App.css 수정

```
button { padding: 0.3em 1.3em; margin-right: 1em; }
```

버튼을 클릭하면 increaseAt 함수가 호출되고

arr[index] = arr[index] + 1;

배열의 원소의 값이 변경되고

setArr(arr);

상태 변수 setter가 호출되지만

렌더링 되지 않는다.

arr 배열의 내부 값만 변경되었을 뿐, arr 배열은 이전과 같은 배열이다.
이 경우에는 렌더링 되지 않는다.

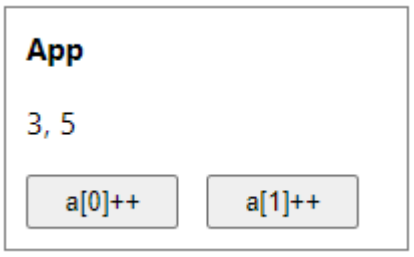
새 배열이 setArr 되어야 다시 렌더링 된다.

App.jsx 수정

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

function App() {
  const [arr, setArr] = useState([0, 0]);
  const increaseAt = (index: number) => {
    arr[index] = arr[index] + 1;
    setArr([...arr]);
  };
  return (
    <div className="box">
      <h1>App</h1>
      <p>{ arr.join(", ") }</p>
      <button onClick={() => increaseAt(0)} >a[0]++</button
>
      <button onClick={() => increaseAt(1)} >a[1]++</button
>
    </div>
  );
}

export default App;
```

[...arr]

새 배열이 생성되고, 그 새 배열에 arr 배열의 원소들이 채워진다.
...는 spread 연산자이다.



버튼을 클릭할 때 마다 렌더링 된다.

요약

컴포넌트 상태 값이 배열일 때, 배열의 원소 값만 바뀌어서는 렌더링되지 않는다.
상태 값이 새 배열로 바뀌어야, 즉 setter에 새 배열을 전달해야 렌더링 된다.

언제 렌더링 안되는가? #2

App.jsx 수정

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

function App() {
  const [person, setPerson] = useState({ name: '홍길동', ag
e: 16, height: 180});
  const increase = (key) => {
  if (key==="height") person.height++;
    else if (key==="age") person.age++;
    setPerson(person);
  };
  return (
    <div className="box">
      <h1>App</h1>
      <p>{ person.age }, { person.height }</p>
      <button onClick={() => increase("age")} >age++</butto
```

```
n>
      <button onClick={() => increase("height")} >height++
</button>
    </div>
  );
}

export default App;
```

버튼을 클릭하면 increase 함수가 호출되고

if (key==="height") person.height++;
else if (key==="age") person.age++;

person 객체의 속성이 변경되고

setPerson(person);

상태 변수 setter가 호출되지만

렌더링 되지 않는다.

person 객체의 내부 값만 변경되었을 뿐, person 객체는 이전과 같은 객체이다.
이 경우에는 렌더링 되지 않는다.

새 객체가 setPerson 되어야 다시 렌더링 된다.

App.tsx 수정

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

type Person = {
  name: string,
  age: number,
  height: number
}

function App() {
```

```
  const [person, setPerson] = useState({ name: '홍길동', ag
e: 16, height: 180});
  const increase = (key) => {
  if (key==="height") person.height++;
    else if (key==="age") person.age++;
    setPerson({...person});
  };
  return (
    <div className="box">
      <h1>App</h1>
      <p>{ person.age }, { person.height }</p>
      <button onClick={() => increase("age")} >age++</butto
n>
      <button onClick={() => increase("height")} >height++
</button>
    </div>
  );
}

export default App;
```

{...person}

person 객체가 복제된어 새 객체가 만들어진다.
...는 spread 연산자이다.

버튼을 클릭할 때 마다 렌더링 된다.

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.

요약

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