Chapter 7. 훅(Hooks)

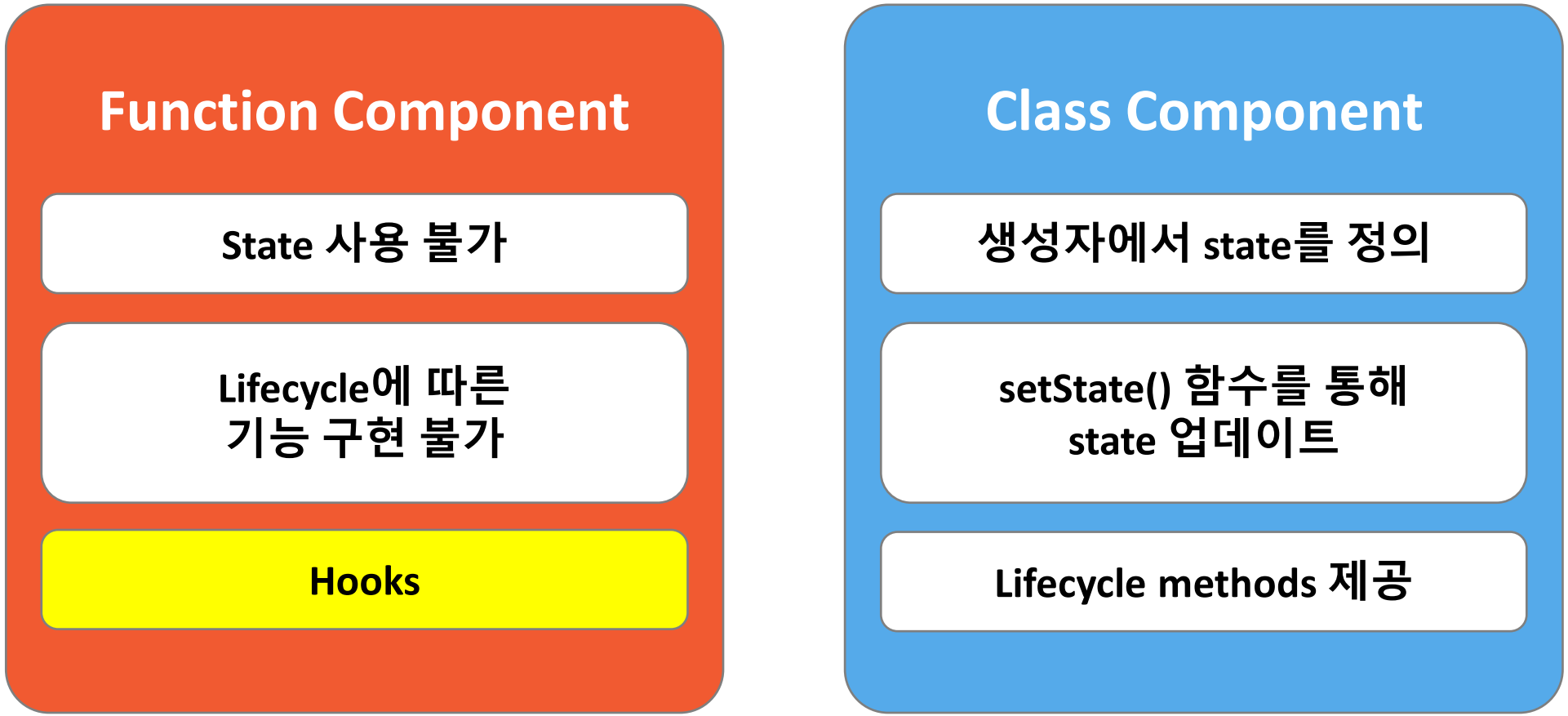
교재: 처음만난 리액트 (저자: 이인제, 한빛출판사)

Contents

- CHAPTER 7: 훅

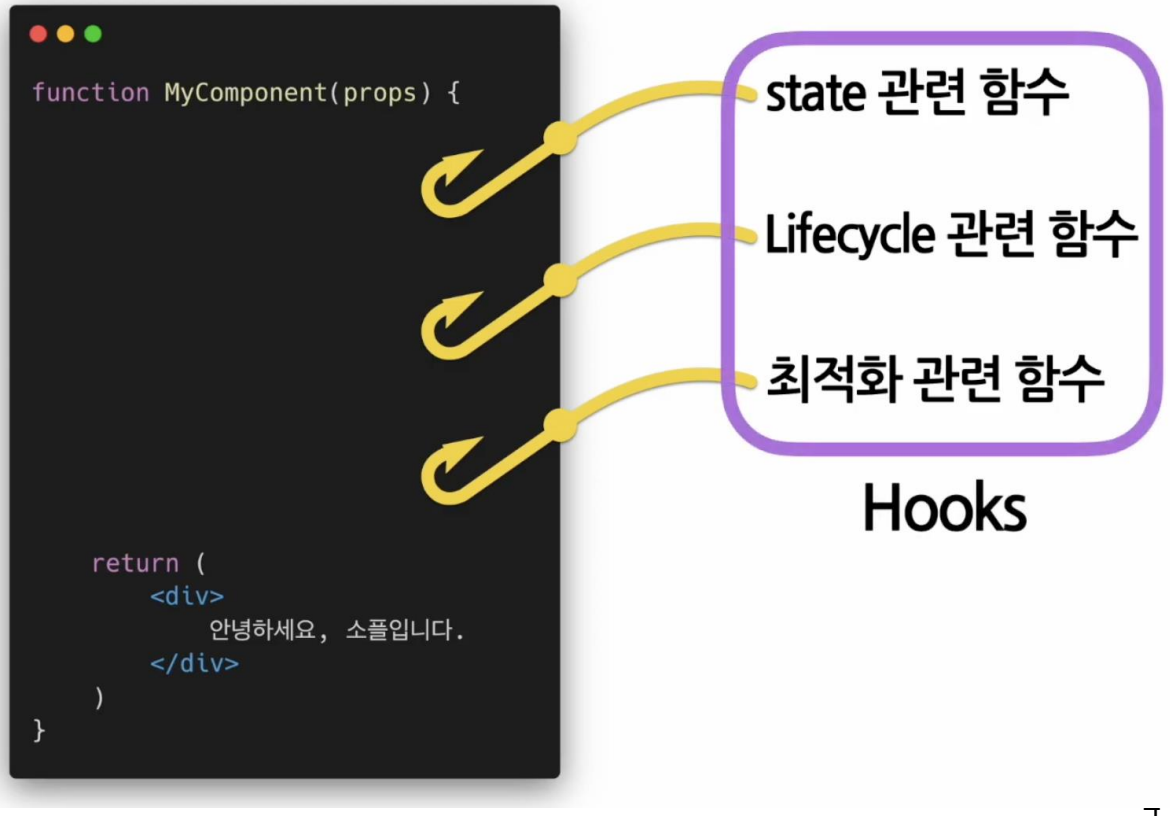
SECTION 7.1 훅이란 무엇인가?

- Hooks



SECTION 7.1 훅이란 무엇인가?

- **Hooks**
  - Hook - 갈고리
  - 원래 존재하는 어떤 기능에 마치 갈고리를 거는 것처럼 끼어 들어가 같이 수행되는 기능
  - 리액트의 state와 생명주기 기능에 갈고리를 걸어 원하는 시점에 정해진 함수를 실행되도록 만든 것 - 이 때 실행되는 함수가 훅
  - 훅의 이름은 모두 use로 시작



SECTION 7.2 useState()

- **useState() -** state를 사용하기 위한 훅
  - 함수 컴포넌트에서는 기본적으로 state를 제공하지 않기 때문에 state를 사용하려면 useState() 훅을 사용해야 함

```jsx
import React, { useState } from "react";

function Counter(props) {
    var count = 0;

    return (
        <div>
            <p>총 {count}번 클릭했습니다.</p>
            <button onClick={() => count++}>
                클릭
            </button>
        </div>
    );
}
```

[state가 없는 함수 컴포넌트]

SECTION 7.2 useState()

◦ useState() 사용법

```
const [변수명, set함수명] = useState(초기값);
```

- useState()를 호출할 때는 state의 초기값을 파라미터로 전달
- 리턴 값 - 배열
  1) state로 선언된 변수
  2) state의 set 함수

SECTION 7.2 useState()

◦ useState() 사용 예제

- Class 컴포넌트와 달리 변수 각각에 대해 set함수가 따로 존재

```
1  import React, { useState } from "react";
2
3  function Counter(props) {
4    const [count, setCount] = useState(0);
5
6    return (
7      <div>
8        <p>총 {count}번 클릭했습니다.</p>
9        <button onClick={() => setCount(count + 1)}>클릭</button>
10     </div>
11   )
12 }
13
14 export default Counter;
```

SECTION 7.3 useEffect()

useEffect() - Side effect를 수행하기 위한 훅

- Side effect 사전적 의미 - 부작용
- 리액트에서의 Side effect - 효과, 영향, effect

  ex) 서버에서 데이터 수신, 수동으로 DOM 변경

  - 다른 컴포넌트에 영향을 미칠 수 있으며 렌더링 중에는 작업이 완료될 수 없기 때문
  - 즉, 렌더링이 끝난 이후에 실행되어야 하는 작업

- 클래스 컴포넌트에서 제공하는 생명주기 함수 componentDidMount(), componentDidUpdate(), ComponentWillMount()와 동일한 기능을 하나로 통합해서 제공
- useEffect() 훅만으로 위 생명주기 함수와 동일한 기능을 수행할 수 있음

SECTION 7.3 useEffect()

◦ useEffect() 사용법

```
useEffect(이펙트 함수, 의존성 배열);
```

- 의존성 배열 – 이 이펙트가 의존하고 있는 배열, 배열 안에 있는 변수 중 하나라도 값이 변경되면 이펙트 함수가 실행됨
- 이펙트 함수는 처음 컴포넌트가 렌더링 된 이후, 업데이트로 인한 재랜더링 이후에 실행됨

- 의존성 배열이 빈 값 - Effect function이 mount, unmount 시에 단 한 번씩만 실행 됨

```
useEffect(이펙트 함수, []);
```

- 의존성 배열 생략 - 컴포넌트가 업데이트될 때마다 호출

```
useEffect(이펙트 함수);
```

SECTION 7.3 useEffect()

- **useEffect() 사용 예제 - 의존성 배열 생략(mount, update 시 호출)**

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

function Counter(props) {
  const [count, setCount] = useState(0)

  useEffect(() => {
    document.title = `총 ${count}번 클릭했습니다.`;
  })

  return (
    <div>
      <p>총 {count}번 클릭했습니다.</p>
      <button onClick={() => setCount(count + 1)}>클릭</button>
    </div>
  )
}

export default Counter;
```

DOM 변경 후 effect 함수 실행

componentDidMount(),
componentDidUpdate(),
생명주기 함수와 같은 역할

SECTION 7.3 useEffect()

- **useEffect() 사용 예제 - 의존성 배열 빈 배열(mount, unmount 시 호출)**
  - 리턴 함수는 컴포넌트가 마운트 해제될 때 호출됨

```
2
3  function Counter(props) {
4    const [count, setCount] = useState(0);
5
6    useEffect(() => {
7      document.title = `총 ${count}번 클릭했습니다..`;
8    }, [count]);
9
10   useEffect(() => {
11     return () => {
12       console.log(`useEffect unmount 호출됨. 최종 count: ${count}`);
13     }
14   }, []);
15
16   return (
17     <div>
18       <p>총 {count}번 클릭했습니다.</p>
19       <button onClick={() => setCount(count + 1)}>클릭</button>
20     </div>
21   );
22 }
```

return() 함수는 생명주기 함수 componentDidWillUnmout() 같은 역할

여러 개의 useEffect() 사용 가능

SECTION 7.3 useEffect()

◦ useEffect() 사용법 - 의존성 배열

- 배열 - 변수들 중 하나라도 값이 변경되었을 때 실행

```
1   useEffect(() => {
2     document.title = `총 ${count}번 클릭했습니다.`;
3   }, [count]);
```

- 빈 배열 - mount와 unmount 될 때 한 번씩만 실행

```
1   useEffect(() => {
2     document.title = `총 ${count}번 클릭했습니다.`;
3   }, []);
```

- 생략 – 컴포넌트가 업데이트 될 때마다 실행

```
1   useEffect(() => {
2     document.title = `총 ${count}번 클릭했습니다.`;
3   });
```

SECTION 7.3 useEffect()

- **useEffect() 구조**

```
useEffect(() => {
    // 컴포넌트가 마운트 된 이후,
    // 의존성 배열에 있는 변수들 중 하나라도 값이 변경되었을 때 실행됨
    // 의존성 배열에 빈 배열([])을 넣으면 마운트와 언마운트시에 단 한 번씩만 실행됨
    // 의존성 배열 생략 시 컴포넌트 업데이트 시마다 실행됨
    ...

    return () => {
        // 컴포넌트가 마운트 해제되기 전에 실행됨
        ...
    }
}, [의존성 변수1, 의존성 변수2, ...]);
```

SECTION 7.4 useMemo()

◦ **useMemo() - Memoized value를 리턴하는 훅**

- Memoization (메모이제이션)

- 컴퓨터 프로그램이 동일한 계산을 반복해야 할 때, 이전에 계산한 값을 메모리에 저장함으로써 동일한 계산의 반복 수행을 제거하여 프로그램 실행 속도를 빠르게 하는 기술 -> 성능 최적화

- 메모이제이션 된 결과값이 Memoized value

SECTION 7.4 useMemo()

◦ 렌더링마다 호출되는 컴포넌트 함수

- 컴포넌트의 렌더링은 수시로 계속 일어날 수 있음
- 렌더링될 때마다 컴포넌트 함수가 호출되고 내부 로직들이 수행됨

```
function MyComponent({ x, y }) {
  const value = compute(x, y);
  return <div>{value}</div>;
}

function compute(x, y) {
  // 오래 걸리는 작업
}
```

- compute() 함수가 복잡한 연산을 수행한다면 컴포넌트가 재랜더링될 때마다 UI 지연 발생
- compute() 함수의 인자 x, y 값이 매번 바뀌지 않는다면 다시 계산할 필요가 없음
  - Memoization 기법 적용

SECTION 7.4 useMemo()

◦ useMemo() 사용법

```
const memoizedValue = useMemo(
    () => {
        // 연산량이 높은 작업을 수행하여 결과를 반환
        return computeExpensiveValue(의존성 변수1, 의존성 변수2);
    },
    [의존성 변수1, 의존성 변수2]
);
```

- 의존성 배열의 요소 값이 업데이트될 때만 콜백 함수를 다시 호출
  - momoization된 값을 업데이트 후 다시 memoization 해줌

- useMemo()로 전달된 함수는 렌더링이 일어나는 동안 실행됨
- 렌더링이 일어나는 동안 실행돼서는 안되는 작업은 useMemo()에 사용하면 안됨
  - useEffect()에서 실행돼야 할 이펙트 작업

SECTION 7.4 useMemo()

- **useMemo() - 의존성 배열**
  - 의존성 배열을 넣지 않을 경우, 매 렌더링마다 함수가 실행되므로 효과 없음

```
const memoizedValue = useMemo(
    () => computeExpensiveValue(a, b)
);
```

  - 의존성 배열이 빈 배열일 경우, 컴포넌트 마운트 시에만 값을 계산

```
const memoizedValue = useMemo(
    () => {
        return computeExpensiveValue(a, b);
    },
    []
);
```

SECTION 7.4 useMemo()

◦ 함수 컴포넌트에 useMemo() 훅 적용

```
function MyComponent({ x, y }) {
  const value = useMemo(() => compute(x, y), [x, y]);
  return <div>{value}</div>;
}

function compute(x, y) {
  // 오래 걸리는 작업
}
```

- useMemo() 훅을 이용한 memorization 적용
- x, y 값이 이전 렌더링했을 때와 동일할 경우, 저장했던 value 값을 재활용
- x, y 값이 이전 렌더링했을 때와 달라졌을 경우, compute() 함수를 호출하여 결과값을 계산한 후 value에 할당

SECTION 7.5 useCallback()

- **useCallback() – 함수를 메모이제이션 하기 위한 훅**
  - useMemo() Hook과 유사하지만 값이 아닌 함수를 반환
  - 첫 번째 인자로 넘어온 함수를, 의존성 배열 내의 값이 변경될 때까지 저장하여 재사용

```
const memoizedCallback = useCallback(
    () => {
        doSomething(의존성 변수1, 의존성 변수2);
    },
    [의존성 변수1, 의존성 변수2]
);
```

  - useCallback() 훅을 사용하지 않고 컴포넌트 내에 함수를 정의할 경우, 렌더링이 일어날 때마다 함수가 새로 정의됨
  - useCallback() 훅을 사용하면 특정 변수의 값이 변한 경우에만 함수를 다시 정의

SECTION 7.6 useRef()

- **useRef() – Reference를 사용하기 위한 훅**
  - 리액트 레퍼런스 – 특정 컴포넌트에 접근할 수 있는 객체
  - useRef()는 레퍼런스 객체를 반환
  - 레퍼런스 객체의 .current 속성 – 현재 참조하고 있는 엘리먼트

- **useRef() 사용법**

```
const refContainer = useRef(초깃값);
```

  - 초기값으로 초기화된 레퍼런스 반환
  - 초기값이 null이면 .current 값이 null인 객체 반환 - { current: null }

SECTION 7.6 useRef()

- **useRef() 예제**

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

function TextInputWithFocusButton(props) {
  const inputElem = useRef(null);

  const onButtonClick = () => {
    inputElem.current.focus();
  }

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

export default TextInputWithFocusButton;
```

SECTION 7.7 훅의 규칙

◦ 규칙1 - 훅은 무조건 최상위 레벨에서만 호출해야 한다.

- 리액트 함수 컴포넌트의 최상위 레벨에서 호출
  - 반복문, 조건문, 중첩된 함수 안에서 호출하면 안됨
- 컴포넌트가 렌더링 될 때마다 매번 같은 순서로 호출되어야 함

```
1 function MyComponent(props) {
2   const [name, setName] = useState('induk');
3
4   if (name !== '') {
5     useEffect(() => {
6       ...
7     }
8   }
9
10  ...
11 }
```

잘못된 Hook 사용법

SECTION 7.7 훅의 규칙

- **규칙2 - 리액트 함수 컴포넌트에서만 훅을 호출해야 한다.**
  - 일반적인 자바스크립트 함수에서 훅을 호출하면 안 됨
  - 훅은 리액트 함수 컴포넌트에서 호출하거나 직접 만든 커스텀 훅에서만 호출

SECTION 7.8 나만의 훅 만들기

◦ 커스텀 훅(Custom Hook)

- 리액트에서 기본 제공되는 훅 이외에 추가로 필요한 기능을 직접 만들어서 사용

  - 여러 컴포넌트에서 반복적으로 사용되는 로직을 훅으로 만들어 재사용하기 위함

  - 이름이 use로 시작하고 내부에서 다른 Hook을 호출하는 하나의 자바스크립트 함수

SECTION 7.8 나만의 훅 만들기

◦ 커스텀 훅을 만들어야 하는 상황

- 사용자의 온라인 상태를 알려주는 컴포넌트

```jsx
import React, { useState, useEffect } from "react";

function UserStatus(props) {
    const [isOnline, setIsOnline] = useState(null);

    useEffect(() => {
        function handleStatusChange(status) {
            setIsOnline(status.isOnline);
        }

        ServerAPI.subscribeUserStatus(props.user.id, handleStatusChange);
        return () => {
            ServerAPI.unsubscribeUserStatus(props.user.id, handleStatusChange);
        };
    });

    if (isOnline === null) {
        return '대기중...';
    }
    return isOnline ? '온라인' : '오프라인';
}
```

동일한 코드

- 사용자의 온라인 상태를 컬러로 표시해주는 컴포넌트

```jsx
import React, { useState, useEffect } from "react";

function UserListItem(props) {
    const [isOnline, setIsOnline] = useState(null);

    useEffect(() => {
        function handleStatusChange(status) {
            setIsOnline(status.isOnline);
        }

        ServerAPI.subscribeUserStatus(props.user.id, handleStatusChange);
        return () => {
            ServerAPI.unsubscribeUserStatus(props.user.id, handleStatusChange);
        };
    });

    return (
        <li style={{ color: isOnline ? 'green' : 'black' }}>
            {props.user.name}
        </li>
    );
}
```

SECTION 7.8 나만의 훅 만들기

◦ 커스텀 훅 추출하기

- 두 개의 자바스크립트 함수에서 하나의 로직을 공유하고 싶을 때 새로운 함수를 만듦
- 리액트 컴포넌트와 훅은 모두 함수이기 때문에 동일한 방법을 사용

- 두 개의 컴포넌트에서 중복되는 로직을 추출하여 커스텀 훅으로 정의

```
1 import { useState, useEffect } from "react";
2
3 function useUserStatus(userId) {
4     const [isOnline, setIsOnline] = useState(null);
5
6     useEffect(() => {
7         function handleStatusChange(status) {
8             setIsOnline(status.isOnline);
9         }
10
11        ServerAPI.subscribeUserStatus(userId, handleStatusChange);
12        return () => {
13            ServerAPI.unsubscribeUserStatus(userId, handleStatusChange);
14        };
15    });
16
17    return isOnline;
18 }
```

SECTION 7.8 나만의 훅 만들기

커스텀 훅 규칙

- 커스텀 훅은 특별한 규칙이 없음
  - 파라미터, 리턴 값을 개발자가 직접 정의해서 사용

- 커스텀 훅은 단순한 함수와 같지만 다음과 같은 훅의 규칙은 적용됨
  - 훅의 이름은 use로 시작
  - 훅은 최상위 레벨에서만 호출해야 함
  - 리액트 함수 컴포넌트에서만 훅을 호출해야 함

- 커스텀 훅의 이름을 use로 시작하지 않는다면?
  - 특정 함수 내부에서 훅을 호출하는지 알 수 없기 때문에,
    훅의 규칙 위반 여부를 자동으로 확인할 수 없음

SECTION 7.8 나만의 훅 만들기

◦ 커스텀 훅 사용

- 중복된 로직을 커스텀 훅으로 추출하여 기존 컴포넌트에 사용

```
1 function UserStatus(props) {
2     const isOnline = useUserStatus(props.user.id);
3
4     if (isOnline === null) {
5         return '대기중...';
6     }
7     return isOnline ? '온라인' : '오프라인';
8 }
9
10 function UserListItem(props) {
11     const isOnline = useUserStatus(props.user.id);
12
13     return (
14         <li style={{ color: isOnline ? 'green' : 'black' }}>
15             {props.user.name}
16         </li>
17     );
18 }
```

SECTION 7.8 나만의 훅 만들기

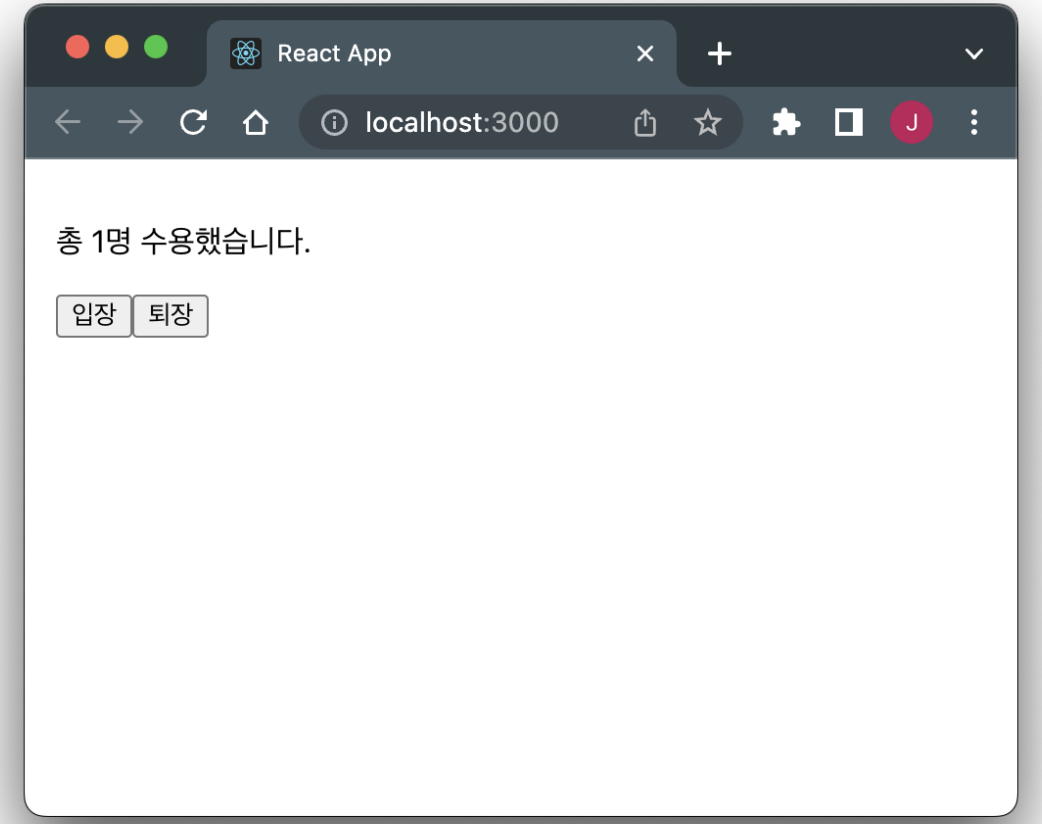
◦ 커스텀 훅 사용

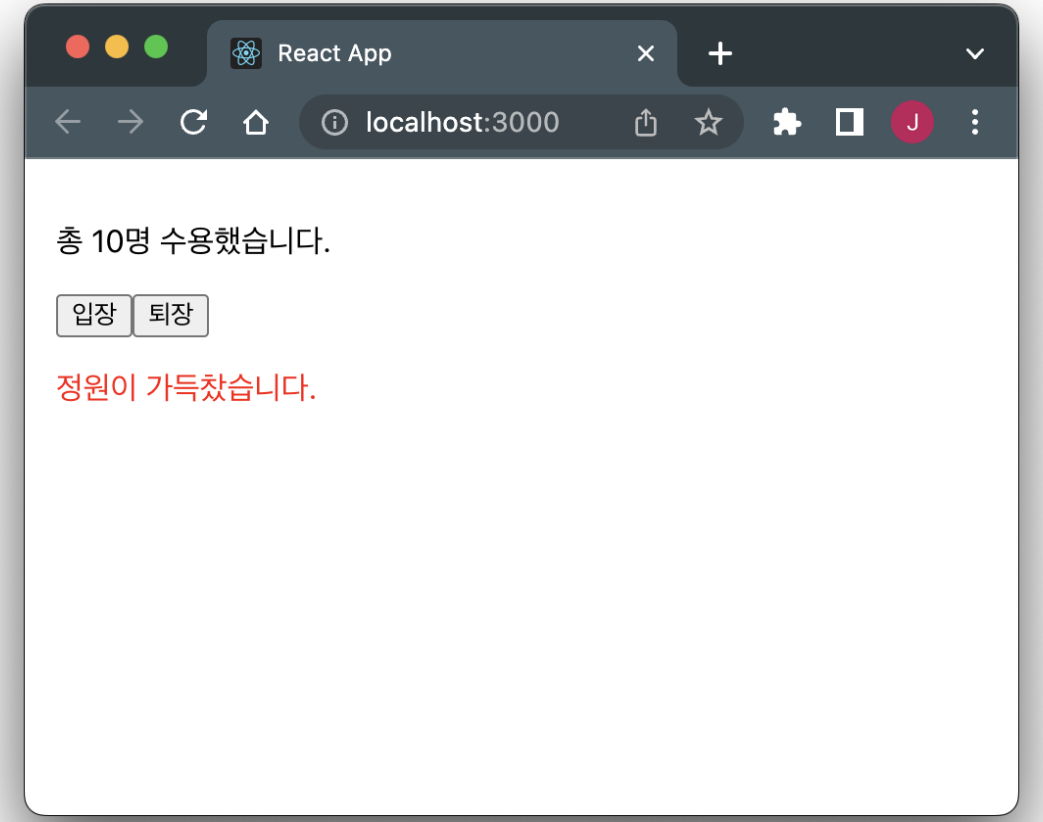
- 여러 개의 컴포넌트에서 하나의 커스텀 훅을 사용할 때 state는 공유될까?
  - 컴포넌트 내부에 있는 모든 state와 effects는 전부 분리됨
  - 각각의 커스텀 훅 호출에 대해 분리된 state를 얻게 됨

- 커스텀 훅의 호출은 독립적으로 동작

PRACTICE 7.9 훅을 사용한 컴포넌트 개발

- useState(), useEffect(), Custom Hook 사용하는 컴포넌트 만들기





PRACTICE 7.9 훅을 사용한 컴포넌트 개발

- useCounter() 커스텀 훅 - useCounter.jsx
  - 초기 카운트 값을 받아서 count라는 이름의 state 관리
  - 카운트 증가(increaseCount()), 감소(decreaseCount()) 함수 제공

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

function useCounter(initialValue) {
  const [count, setCount] = useState(initialValue);

  const increaseCount = () => setCount((count) => count + 1);
  const decreaseCount = () => setCount((count) => Math.max(count - 1, 0));

  return [count, increaseCount, decreaseCount];
}

export default useCounter;
```

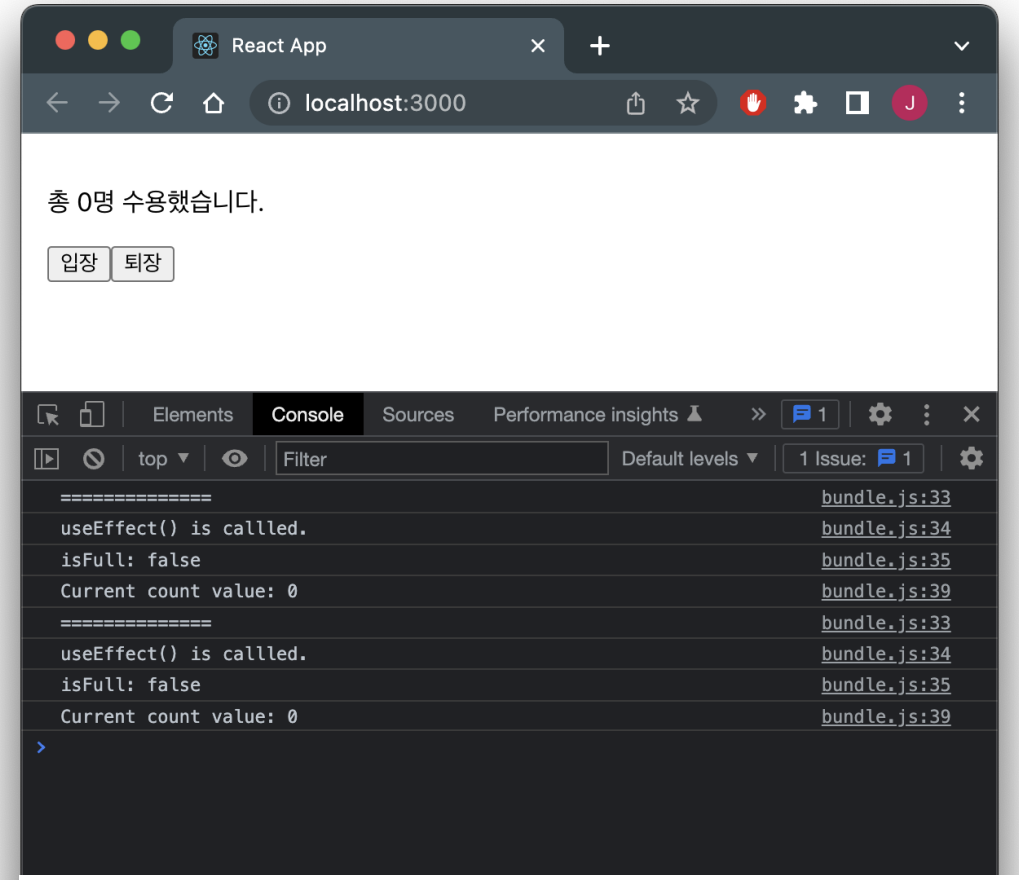
PRACTICE 7.9 훅을 사용한 컴포넌트 개발

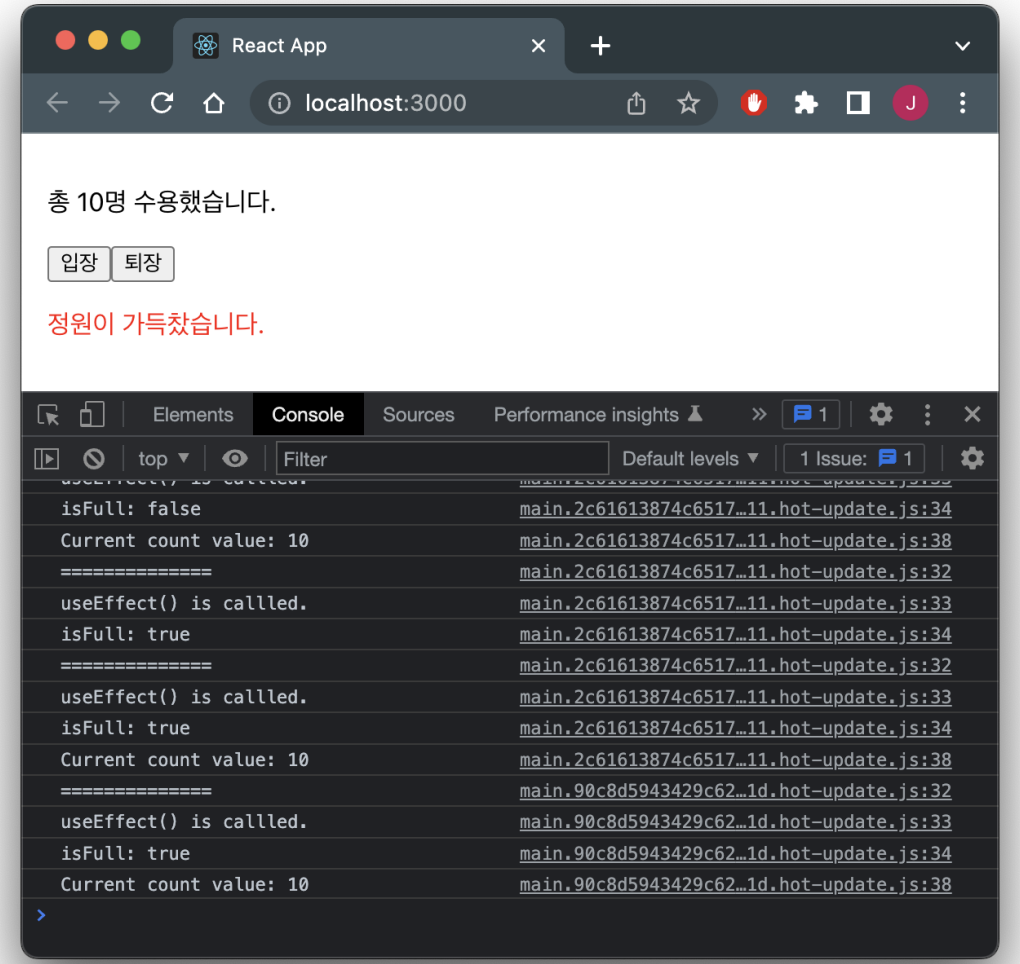
- useCouter() 훅을 사용하는 함수 컴포넌트 - Accomodate.jsx
  - useCounter() 훅을 사용하여 카운트 관리
  - useEffect() - 의존성 배열이 없는 훅, 의존성 배열이 있는 훅

```jsx
import React, { useState, useEffect } from "react";
import useCounter from "./useCounter";

const MAX_CAPACITY = 10;

function Accomodate(props) {
  const [isFull, setIsFull] = useState(false);
  const [count, increaseCount, decreaseCount] = useCounter(0);

  useEffect(() => {
    console.log('==============');
    console.log('useEffect() is callled.');
    console.log(`isFull: ${isFull}`);
  });

  useEffect(() => {
    setIsFull(count >= MAX_CAPACITY);
    console.log((`Current count value: ${count}`));
  }, [count]);

  return (
    <div style={{ padding: 16 }}>
      <p>{`총 ${count}명 수용했습니다.`}</p>

      <button onClick={increaseCount}>입장</button>
      <button onClick={decreaseCount}>퇴장</button>

      {isFull && <p style={{ color: 'red' }}>정원이 가득찼습니다.</p>}
    </div>
  );
}

export default Accomodate;
```

PRACTICE 7.9 훅을 사용한 컴포넌트 개발

- 실행 결과





React.Strict mode is on

> StrictMode renders components twice (on dev but not production) in order to detect any problems with your code and warn you about them (which can be quite useful).

[요약]

- Hook
  - 훅이란?
    - 리액트의 state와 생명주기 기능에 갈고리를 걸어 원하는 시점에 정해진 함수를 실행되도록 만든 것
  - useState()
    - state를 사용하기 위한 훅
    - 함수 컴포넌트에서는 기본적으로 state라는 것을 제공하지 않음
    - 클래스 컴포넌트처럼 state를 사용하고 싶으면 useState() 훅을 사용
    - 사용법
      - const [변수, set함수] = useState(변수의 초기값);
      - 변수 각각에 대해 set 함수가 따로 존재

[요약]

- Hook
  - useEffect()
    - 사이드 이펙트를 수행하기 위한 훅
    - 사이드 이펙트 - 서버에서 데이터를 받아오거나 수동으로 DOM을 변경하는 등의 작업
    - useEffect() 훅만으로 클래스 컴포넌트의 생명주기 함수들과 동일한 기능 수행 가능
    - 사용법
      - useEffect(이펙트 함수, 의존성 배열);
      - 의존성 배열 안에 있는 변수 중에 하나라도 값이 변경되었을 때 이펙트 함수가 실행됨
      - 의존성 배열에 빈 배열([])을 넣으면 마운트와 언마운트 시 한 번씩만 실행됨
      - 의존성 배열 생략 시 컴포넌트가 업데이트될 때마다 호출됨
      - 선언된 컴포넌트의 props와 state에 접근할 수 있음
      - useEffect()에서 리턴하는 함수는 컴포넌트가 해제될 때 호출됨

[요약]

- Hook
  - useMemo()
    - Memoized value를 리턴하는 훅
    - 연산량이 높은 작업이 매번 렌더링될 때마다 반복되는 것을 피하기 위해 사용
    - 렌더링이 일어나는 동안 실행되므로 렌더링이 일어나는 동안 실행돼서는 안되는 작업을 useMomo()에 넣으면 안 됨
    - 사용법
      - const memoizedValue = useMemo(값 생성 함수, 의존성 배열);
      - 의존성 배열에 들어있는 변수가 변했을 경우에마 값 생성 함수를 호출하여 결과값 반환
      - 그렇지 않은 경우에는 기존 함수의 결과값을 그대로 반환
      - 의존성 배열을 생략하면 렌더링이 일어날 때마다 매번 함수가 실행되므로 의미 없음

[요약]

- Hook
  - useCallback()
    - useMemo() 훅과 유사하지만 값이 아닌 함수를 반환
    - 컴포넌트 내에 함수를 정의하면 매번 렌더링이 일어날 때마다 함수가 새로 정의되므로 useCallback() 훅을 사용하면 불필요한 함수 재정의 작업을 없앨 수 있음
    - 사용법
      - const memoizedCallback = useCallback(콜백 함수, 의존성 배열);
      - 의존성 배열에 들어있는 변수가 변했을 경우에마 콜백 함수를 다시 정의해서 반환
  - useRef()
    - 레퍼런스를 사용하기 위한 훅 (레퍼런스: 특정 컴포넌트에 접근할 수 있는 객체)
    - 매번 렌더링될 때마다 항상 같은 레퍼런스 객체를 반환
    - 사용법
      - const refContainer = useRef(초기값);
      - .current라는 속성을 통해서 접근

[요약]

- Hook의 규칙
  - 무조건 최상위 레벨에서만 호출해야 함
    - 반복문이나 조건문 또는 중첩된 함수들 안에서 훅을 호출하면 안 됨
    - 컴포넌트가 렌더링될 때마다 매번 같은 순서로 호출되어야 함
  - 리액트 함수 컴포넌트에서만 훅을 호출해야 함
    - 훅은 리액트 함수 컴포넌트에서 호출하거나 직접 만든 커스텀 훅에서만 호출할 수 있음

- 커스텀 훅
  - 이름이 use로 시작하고 내부에서 다른 훅을 호출하는 단순한 자바스크립트 함수
  - 파라미터로 무엇을 받을지, 어떤 값을 리턴할지를 개발자가 직접 정할 수 있음
  - 중복되는 로직을 커스텀 훅으로 추출하여 재사용성을 높일 수 있음
  - 이름이 use로 시작하지 않으면 특정 함수의 내부에서 훅을 호출하는지 알 수 없기 때문에 훅의 규칙 위반 여부를 자동으로 확인할 수 없음