Hook의 규칙

Hook은 자바스크립트 함수로 정의되지만 호출 위치에 제약이 있는 특별한 유형의 재사용 가능한 UI 로직입니다.

- Hook을 최상위 레벨에서만 호출하세요
- Hook을 React 함수에서만 호출하세요

Hook을 최상위 레벨에서만 호출하세요

React에서는 `use` 로 시작하는 함수를 *Hook*이라고 부릅니다.

Hook을 반복문, 조건문, 중첩 함수, 또는 `try` / `catch` / `finally` 블록 내부에서 호출하지 마세요. 대신 Hook을 항상 React 함수의 최상위 레벨에서 호출하고, early return 이전에 사용해야 합니다. Hook은 React가 함수 컴포넌트를 렌더링하는 동안에만 호출할 수 있습니다.

- ✅ 함수 컴포넌트의 본문 최상위 레벨에서 호출하세요.
- ✅ 커스텀 Hook의 본문 최상위 레벨에서 호출하세요.

```
function Counter() {
  // ✅ Good: 함수 컴포넌트의 최상위 레벨에서 사용합니다.
  const [count, setCount] = useState(0);
  // ...
}

function useWindowWidth() {
  // ✅ Good: 커스텀 Hook의 최상위 레벨에서 사용합니다.
  const [width, setWidth] = useState(window.innerWidth);
  // ...
}
```

다음과 같이 Hook(`use` 로 시작하는 함수)을 호출하는 것은 지원되지 **않습니다**.

- 🔴 조건문이나 반복문 내부에서 Hook을 호출하지 마세요.
- 🔴 조건부 `return` 문 이후에 Hook을 호출하지 마세요.
- 🔴 이벤트 핸들러에서 Hook을 호출하지 마세요.
- 🔴 클래스 컴포넌트에서 Hook을 호출하지 마세요.
- 🔴 `useMemo`, `useReducer`, `useEffect` 에 전달된 함수 내부에서 Hook을 호출하지 마세요.
- 🔴 `try` / `catch` / `finally` 블록 내부에서 Hook을 호출하지 마세요.

이 규칙을 어기면 오류가 발생할 수 있습니다.

```
function Bad({ cond }) {
  if (cond) {
    // 🔴 Bad: 조건부 내부 (수정하려면 외부로 이동하세요!)
    const theme = useContext(ThemeContext);
  }
  // ...
}

function Bad() {
  for (let i = 0; i < 10; i++) {
    // 🔴 Bad: 반복문 내부 (수정하려면 외부로 이동하세요!)
    const theme = useContext(ThemeContext);
  }
  // ...
}

function Bad({ cond }) {
  if (cond) {
    return;
  }
  // 🔴 Bad: 조건부 `return`문 이후 (수정하려면 `return`문 이전으로 이동하세요!)
  const theme = useContext(ThemeContext);
  // ...
}

function Bad() {
  function handleClick() {
    // 🔴 Bad: 이벤트 핸들러 내부 (수정하려면 외부로 이동하세요!)
    const theme = useContext(ThemeContext);
  }
  // ...
}
```

```
function Bad() {
  const style = useMemo(() => {
    // 🔴 Bad: `useMemo` 내부 (수정하려면 외부로 이동하세요!)
    const theme = useContext(ThemeContext);
    return createStyle(theme);
  });
  // ...
}

class Bad extends React.Component {
  render() {
    // 🔴 Bad: 클래스 컴포넌트 내부 (수정하려면 클래스 컴포넌트 대신 함수 컴포넌트를 사용하세요
    useEffect(() => {})
    // ...
  }
}

function Bad() {
  try {
    // 🔴 Bad: `try`/`catch`/`finally` 블록 내부 (수정하려면 외부로 이동하세요!)
    const [x, setX] = useState(0);
  } catch {
    const [x, setX] = useState(1);
  }
}
```

이러한 실수를 잡기 위해 `eslint-plugin-react-hooks` 플러그인을 사용할 수 있습니다.

중요합니다!

커스텀 Hook은 다른 Hook을 *호출할 수 있습니다*. (이것이 바로 커스텀 Hook의 주된 목적입니다.) 커스텀 Hook도 함수 컴포넌트가 렌더링되는 동안에만 호출될 수 있기 때문입니다.

Hook을 React 함수에서만 호출하세요

일반 자바스크립트 함수에서 Hook을 호출하지 마세요. 대신 다음과 같이 사용할 수 있습니다.

✅ Hook을 React 함수 컴포넌트에서 호출하세요.
✅ Hook을 커스텀 Hook에서 호출하세요.

이 규칙을 따르면 컴포넌트의 모든 상태 관리 로직이 소스 코드에서 명확히 보입니다.

```
function FriendList() {
  const [onlineStatus, setOnlineStatus] = useOnlineStatus(); // ✅
}

function setOnlineStatus() { // ❌ 컴포넌트나 커스텀 Hook이 아닙니다!
  const [onlineStatus, setOnlineStatus] = useOnlineStatus();
}
```


이전

React가 컴포넌트와 Hook을 호출하는 방식

커뮤니티

행동 강령

팀 소개

문서 기여자

감사의 말

더 보기

블로그

React Native

개인 정보 보호

약관