

리액트의 메모이제이션, useMemo vs useCallback

리액트의 메모이제이션, useMemo vs useCallback

리액트는 컴포넌트 기반으로 작동하기 때문에, 컴포넌트가 매번 렌더링될 때마다 불필요한 계산을 반복적으로 수행할 경우 성능상 문제가 될 수 있습니다. 이를 해결하기 위해 리액트에서 제공하는 메모이제이션 기능을 사용할 수 있습니다. 메모이제이션은 계산 결과를 캐시하여, 같은 인자로 함수가 호출될 때 이전에 계산된 결과를 반환하여 성능을 개선하는 기법입니다. (리렌더링 최적화)

기본적으로 useMemo와 useCallback은 리렌더링을 최적화하는데 도움이 되도록 만들어진 Hook이다. 이 Hook들은 주어진 렌더에서 수행해야 하는 작업의 양을 줄이고, 컴포넌트가 다시 렌더링해야 하는 횟수를 줄이면서 리렌더링을 최적화하게 된다.



React는 다음과 같은 조건에서 리렌더링을 진행한다.

- 자신의 state가 변경될 때
- 부모 컴포넌트로부터 전달받은 props가 변경될 때
- 부모 컴포넌트가 리렌더링 될 때
- forceUpdate 함수가 실행될 때

[리렌더링이란?]

이전에 생성한 컴포넌트 정보와 다시 렌더링한 정보를 비교해 최소한의 연산으로 DOM 트리를 업데이트 하는 것을 말한다.

즉, 이전의 Virtual DOM과 현재의 Virtual DOM을 비교하여 변경된 값에 대해 DOM 트리를 업데이트 해주는 것이다.

리액트에서는 메모이제이션을 위해 `useMemo` 와 `useCallback` Hook을 제공합니다. 이 둘은 보통 비슷한 상황에서 사용되지만, 차이점이 존재합니다.

useMemo

`useMemo` Hook을 사용하면 **계산 결과(값;value)**를 캐시할 수 있습니다. 이 Hook은 두 개의 인자를 받습니다. 첫 번째 인자는 **캐시할 계산 함수(function)**이며, 두 번째 인자는 **의존성 배열(dependency array)**입니다. 의존성 배열은 계산 함수에서 참조하는 값들의 배열입니다. 의존성 배열의 값이 변경되었을 때에만 계산 함수가 실행됩니다.

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

위의 코드에서 `computeExpensiveValue` 함수의 실행 결과는 `a`와 `b`로 결정됩니다. `a`나 `b`의 값이 변경되지 않는 한 이 함수는 매번 같은 결과를 반환합니다. 따라서 `useMemo`를 사용해 이 함수의 결과를 캐시하면, **성능상 이점**이 있습니다.

```
import React, { useState, useCallback, useMemo } from "react";

export default function App() {
  const [buttonX, setButtonX] = useState(0);
  const [buttonY, setButtonY] = useState(0);

  const handleButtonX = () => {
    setButtonX((prev) => prev + 1)
  };

  const handleButtonY = () => {
    setButtonY((prev) => prev + 1)
  };

  useMemo(() => {console.log(buttonX)}, [buttonX]);

  return (
    <>
      <button onClick={handleButtonX}>X</button>
      <button onClick={handleButtonY}>Y</button>
    </>
  );
}

* useMemo의 deps가 buttonX이기 때문에 X 버튼을 누를때만 console.log(buttonX)가 실행
```

useCallback

`useCallback` Hook은 **메모이제이션된 콜백 함수(function)**를 반환합니다. 이 Hook도 `useMemo`와 비슷한 역할을 하지만, 반환값이 함수입니다. `useCallback`은 두 개의 인자를 받습니다. 첫 번째 인자는 **캐시할 콜백 함수(callback function)**이며, 두 번째 인자는 **의존성 배열**입니다. `useMemo`와 마찬가지로, 의존성 배열의 값이 변경되었을 때에만 함수가 실행됩니다.

```
const memoizedCallback = useCallback(() => {
  doSomething(a, b);
}, [a, b]);
```

위의 코드에서 `memoizedCallback` 은 `a` 와 `b` 로 결정되는 `doSomething` 함수입니다. `a` 나 `b` 의 값이 변경되지 않는 한, 이 함수는 매번 같은 결과를 반환합니다. `useCallback` 을 사용해 이 함수를 캐시하면, 이전에 생성된 함수를 재사용할 수 있습니다. 따라서 `useCallback` 을 사용하면 불필요한 함수 생성을 줄일 수 있어 성능상 이점이 있습니다.

```
import React, { useState, useCallback, useMemo } from "react";

function App() {
  const [buttonX, setButtonX] = useState(0);
  const [buttonY, setButtonY] = useState(0);

  const handleButtonX = () => {
    setButtonX((prev) => prev + 1)
  };

  const handleButtonY = () => {
    setButtonY((prev) => prev + 1)
  };

  const returnUseCallback = useCallback(() => {console.log(buttonY)}, [buttonX]);

  returnUseCallback();

  return (
    <>
      <button onClick={handleButtonX}>X</button>
      <button onClick={handleButtonY}>Y</button>
    </>
  );
}

export default App;

* useCallback 구문은 () => {console.log(buttonY)} 라는 함수를 반환하지만, deps인 buttonX의 변화가 생길때만 반환되게 된다.
```


`useCallback`의 `deps`가 변환될 때 반환되는 함수는, 이전의 함수와 형태가 같지만 새로운 함수이다. 이는 새로운 무기명 함수를 반환한 것이며, 이전의 함수와 값이 같을 뿐 다른 메모리 주소를 가지고 있다.

결론

`useMemo`와 `useCallback`은 메모이제이션을 위한 Hook입니다. 이 둘은 비슷한 상황에서 사용되지만, 반환값의 차이점이 존재합니다. `useMemo`는 **계산 결과를 반환하는 함수를 캐시**하고, `useCallback`은 **콜백 함수를 캐시**합니다. 이 두 Hook을 적절히 사용하면, 성능상 이점을 얻을 수 있습니다.

useMemo와 useCallback의 차이

React Hooks에 대해 공부하면서 `useMemo`와 `useCallback`의 차이에 대해 궁금증이 생기게 되었다.

 https://velog.io/@hang_kem_0531/useMemo와-useCallback의-차이



React.memo

단순히 `useCallback`의 사용만으로는 하위 컴포넌트의 리렌더링을 방지할 수 없다. 하위 컴포넌트가 `PureComponent` 이어야만 비로소 불필요한 리렌더링을 막을 수 있다. 컴포넌트를 `React.memo()`로 래핑하면 해당 컴포넌트는 `PureComponent`가 된다.

React.PureComponent

`React.PureComponent`는 `React.Component`에 `shouldComponentUpdate()`가 적용된 버전이다.

컴포넌트가 `React.memo()`로 래핑 될 때, React는 컴포넌트를 렌더링하고 결과를 Memoizing 한다. 그리고 다음 렌더링이 일어날 때 **props가 같다면**, React는 Memoizing 된 내용을 재사용한다.

```
export function Movie({ title, releaseDate }) {  
  return (  

```

```

    <div>
      <div>Movie title: {title}</div>
      <div>Release date: {releaseDate}</div>
    </div>
  );
}

export const MemoizedMovie = React.memo(Movie);

```

* 위 컴포넌트에서는 title이나 releaseDate 같은 props가 변경되지 않으면 다음 렌더링 때 메모이징 된 내용을 그대로 사용하게 된다.

useMemo와 useCallback을 사용하지 말아야 할 경우


1. host 컴포넌트에 (div span a img...) 전달하는 모든 항목에 대해 쓰지 말아야 한다. 리엑트는 여기에 함수 참조가 변경되었는지 신경쓰지 않는다. (ref, mergeRefs는 여기에서 제외된다.)
2. useCallback useMemo의 의존성 배열에 완전히 새로운 객체와 배열을 전달해서는 안된다. 이는 항상 의존성이 같지 않다는 결과를 의미하며, 메모이제이션을 하는데 소용이 없다. useEffect useCallback useMemo의 모든 종속성은 참조 동일성을 확인한다.
3. 전달하려는 항목이 새로운 참조여도 상관없다면, 사용하지 말아야 한다. 매번 새로운 참조여도 상관없는데, 새로운 참조라면 메모이제이션하는 것이 의미가 없다.

useMemo와 useCallback을 사용해야 할 경우

1. 계산 비용이 많이 들고, 사용자의 입력 값이 map과 filter을 사용했을 때와 같이 이후 렌더링 이후로도 참조적으로 동일할 가능성이 높은 경우, useMemo를 사용하는 것이 좋다.
2. ref 함수를 부수작용과 함께 전달하거나, mergeRef-style 과 같이 wrapper 함수 ref를 만들 때 useMemo를 쓰자. ref 함수가 변경이 있을 때마다, 리엑트는 과거 값을 null로 호출하고 새로운 함수를 호출한다. 이 경우 ref 함수의 이벤트 리스너를 붙이거나 제거하는 등의 불필요한 작업이 일어날 수 있다. 예를 들어, useIntersectionObserver가 반환하는 ref의 경우 ref 콜백 내부에서 observer의 연결이 끊기거나 연결되는 등의 동작이 일어날 수 있다.
3. 자식 컴포넌트에서 useEffect가 반복적으로 트리거되는 것을 막고 싶을 때 사용하자.
4. 매우 큰 리엑트 트리 구조 내에서, 부모가 리렌더링 되었을 때 이에 다른 렌더링 전파를 막고 싶을 때 사용하자. 자식 컴포넌트가 React.memo React.PureComponent일 경우, 메모이제이션된 props를 사용하게 되면 딱 필요한 부분만 리렌더링 될 것이다.

useMemo와 useCallback의 차이

React Hooks에 대해 공부하면서 useMemo와 useCallback의 차이에 대해 궁금증이 생기게 되었다.

 https://velog.io/@hang_kem_0531/useMemo와-useCallback의-차이

