useMemo

`useMemo` — это хук в React, который используется для оптимизации производительности. Он позволяет избежать повторных вычислений тяжелых функций и сохраняет результат предыдущего вычисления, если зависимости не изменились. Вот основные моменты, которые стоит знать о `useMemo`:

Как это работает?

`useMemo` принимает два аргумента:

1. Функция, результат которой нужно "запомнить".
2. Массив зависимостей, при изменении которых функция будет перевычислена.

Пример:

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

Здесь `computeExpensiveValue` — это функция, вычисление которой занимает много времени. Хук `useMemo` запоминает её результат, и пересчитывает его только тогда, когда меняются значения `a` или `b`.

Зачем это нужно?

1. **Оптимизация производительности**: Некоторые функции или операции могут быть ресурсоемкими. Использование `useMemo` помогает уменьшить количество вычислений.
2. **Постоянство ссылок**: `useMemo` также полезен, когда нужно гарантировать, что ссылка на объект или массив не изменится между рендерами, если не изменились их зависимости.

Когда использовать?

1. Когда у вас есть вычисления, занимающие много времени или ресурсов.
2. Когда вы передаете объекты или массивы как пропы в дочерние компоненты, и хотите избежать ненужных ререндеров в этих компонентах.

Пример в контексте функционального компонента

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

function ExpensiveComponent({ value1, value2 }) {
    const expensiveValue = useMemo(() => {
        // допустим, это очень "дорогая" операция
        return value1 + value2;
    }, [value1, value2]);

    return <div>{expensiveValue}</div>;
}
```

В этом примере значение `expensiveValue` будет пересчитываться только тогда, когда изменятся `value1` или `value2`.

Важные замечания

- Не стоит злоупотреблять `useMemo`. Он полезен, но добавляет сложность и может привести к ошибкам, если использовать бездумно.

- `useMemo` не гарантирует "вечное" сохранение значения. React может сбросить кэшированные значения при нехватке памяти.

Схожесть useMemo и useEffect

`useMemo` и `useEffect` действительно имеют схожие аспекты, особенно когда речь идет о зависимостях, передаваемых в массив. Однако они служат разным целям:

`useEffect`

- Вызывается после каждого рендера компонента, если не указаны зависимости или если зависимости изменились.
- Используется для выполнения побочных эффектов, таких как работа с API, подписка на события и т.д.
- Не предоставляет кэшированных данных для повторного использования.

Пример:

```
useEffect(() => {
    // Здесь можно выполнить API-запрос, подписаться на событие и так далее
    console.log("useEffect called");
}, [dependency1, dependency2]);
```

`useMemo`

- Вызывается во время рендера компонента.
- Используется для оптимизации производительности, кэшируя результаты вычислений.
- Не подходит для побочных эффектов, таких как асинхронные операции.

Пример:

```
const memoizedResult = useMemo(() => {
    // Здесь можно выполнить ресурсоемкие вычисления
    console.log("useMemo called");
    return someHeavyComputation(dependency1, dependency2);
}, [dependency1, dependency2]);
```

Сходства

- Оба хука принимают массив зависимостей, который определяет, когда хук должен быть повторно вызван.
- Оба предназначены для оптимизации поведения компонентов.

Различия

- `useEffect` предназначен для побочных эффектов и вызывается после рендера.
- `useMemo` предназначен для мемоизации результатов и вызывается во время рендера.

Пример

Рассмотрим простой пример, в котором `useMemo` может быть полезен. Предположим, у нас есть компонент, который принимает массив чисел и возвращает сумму всех четных чисел в этом массиве. Вычисление этой суммы является "дорогой" операцией.

Сначала создадим функцию, которая вычисляет сумму четных чисел:

```
function calculateEvenSum(numbers) {
    console.log("Calculating even sum...");
    let sum = 0;
    for (let i = 0; i < numbers.length; i++) {
        if (numbers[i] % 2 === 0) {
            sum += numbers[i];
        }
    }
    return sum;
}
```

Теперь создадим React-компонент, который использует эту функцию:

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

function EvenSumComponent({ numbers, someOtherProp }) {
    const evenSum = useMemo(() => {
        return calculateEvenSum(numbers);
```

```
    }, [numbers]);

    return (
        <div>
            <p>Sum of even numbers: {evenSum}</p>
            <p>Some other prop: {someOtherProp}</p>
        </div>
    );
}
```

В этом примере `useMemo` используется для кэширования результата функции `calculateEvenSum`. Это значит, что вычисление суммы будет производиться только тогда, когда изменится массив `numbers`. Если какой-то другой проп (например, `someOtherProp`) изменится, компонент перерендерится, но дорогая операция вычисления суммы не будет повторяться.