Урок 6.2. Хуки useMemo, useCallback, useReducer

Изучаем дополнительные хуки реакта, которые помогут оптимизировать приложение, ускорить выполнение некоторых вычислений, уменьшить количество ненужного перерендера.

`useCallback`

Данный хук используется для того, чтобы мемоизировать функции и пересоздавать их только в том случае, если была изменена одна из зависимостей, указанных в хуке.

Возьмём простой компонент:

```
import { useState } from 'react';

export default function MyComponent() {
    const [ state, setState ] = useState(0);

    const myFunc = () => {
        console.log('Моя функция');
    }

    return (
        <div></div>
    )
}
```

В данном случае мы видим использование состояния `state`. При его изменении через функцию `setState` произойдёт перерендер всего компонента, и соответственно, функция `myFunc` будет пересоздаваться каждый раз при изменении состояния. Это не оптимально, поскольку состояние вообще никак не влияет на нашу функцию. Поэтому для того, чтобы функция не пересоздавалась, можно использовать хук `useCallback`.

```
import { useState, useCallback } from 'react';
```

```
export default function MyComponent() {
    const [ state, setState ] = useState(0);

    // Используем useCallback, принимающий на вход
    // первым параметром необходимую функцию,
    // вторым параметром – массив зависимостей, при изменении которых
    // будет выполнено пересоздание данной функции.
    const myFunc = useCallback(() => {
        console.log('Моя функция');
    }, []);

    return (
        <div></div>
    )
}
```

useMemo

В отличие от useCallback, useMemo мемоизирует (запоминает) не сам объект функции, а результат вычисления функции. Если параметры пришли те же самые, то нам не нужно заново выполнять сложные вычисления, ведь результат будет тот же.

Создадим компонент:

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

// функция, которая вычисляет квадрат числа
const pow = (n) => {
    console.log('pow', n);
    return n * n;
}

function MemoComponent() {
    // Необходимое нам состояние: при его изменении пересчёт результата должен быть выполн
ен
    const [num, setNum] = useState(1);

    // Состояние, которое никак не влияет на результат вычисления функции pow
    const [isGreen, setIsGreen] = useState(false);

    // Вычисляем результат (используя useMemo)
    // Функция pow будет вызвана только при изменении num, при
    // изменении isGreen ничего не произойдёт.
    // Без использования useMemo функция sum будет выполняться при
    // любом изменении состояния.
    const result = useMemo(() => sum(num), [num]);
```

```
    return (
        <div>
            <button onClick={() => setNum(num + 1)}>{num}</button>
            <button
                onClick={() => setIsGreen(!isGreen)}
                style={{backgroundColor: isGreen ? 'green' : 'red'}}
            >
                Кнопка {result}
            </button>
        </div>
    )
}

export default MemoComponent;
```

useReducer

Данный хук по своему названию похож на что-то, что мы использовали при изучении Redux, однако он относится к React. Его цель в том, чтобы упростить работу с состоянием в компоненте за счёт создания функции-редьюсера.

```
import {useReducer, useState} from "react";

// Создадим функцию-редьюсер.
// На вход она принимает два параметра: state (текущее состояние)
// и action – объект, который содержит два поля: type (строка, тип действия) и
// payload – опциональное поле, которое содержит необходимые для изменения
// состояния параметры
function reducer(state, action) {
    // Воспользуемся конструкцией switch, чтобы в зависимости от типа
    // действия делать нужные изменения в состоянии.
    switch (action.type) {
        case 'plus':
            return {
                ...state,
                count: state.count + 1
            };
        case 'minus':
            return {
                ...state,
                count: state.count - 1
            }
        default:
            return state;
    }
}

function ReducerComponent() {
    // Используем хук useReducer, который принимает на вход функцию-редьюсер,
```

```
    // созданную ранее, и начальное состояние – в данном случае это объект
    // с полем count

    // useDispatch возвращает массив, первый элемент которого –
    // состояние (так же, как и в useState), второй – dispatch –
    // функция, которую мы будем использовать для изменения состояния,
    // передавая на вход необходимый объект с action.
    const [state, dispatch] = useReducer(reducer, { count: 0 });

    // Деструктурируем объект состояния, получая поле count
    const {count} = state;

    return (
        <div>
            {/* На клик кнопки вызываем dispatch, передавая объект action
                с полем type со значением 'minus' – это значит, что в
                функции-редьюсере в switch мы попадём в case 'minus' */}
            <button onClick={() => dispatch({ type: 'minus' })}>-</button>
            {count}
            {/* На клик кнопки вызываем dispatch, передавая объект action
                с полем type со значением 'plus' – это значит, что в
                функции-редьюсере в switch мы попадём в case 'plus' */}
            <button onClick={() => dispatch({ type: 'plus' })}>+</button>
        </div>
    )
}

export default ReducerComponent;
```