**Запросы данных (Fetch Data)**

Если вам нужно выполнять запрос к серверу (например, запрос через fetch или сторонние библиотеки), используйте useEffect:

import { useState, useEffect } from "react";

function App() {

  const [data, setData] = useState([]);

  useEffect(() => {

    fetch("<https://api.example.com/items>")

      .then((response) => response.json())

      .then((data) => setData(data));

  }, []); // Пустые зависимости: эффект сработает только один раз при монтировании

  return (

      {data.map((item) => (

        {item.name}

      ))}

  );

}

**Подписка на события**

Используйте useEffect, чтобы добавить обработчики событий и удалить их при размонтировании.

import { useEffect } from "react";

function App() {

  useEffect(() => {

    const handleResize = () => console.log("Размер окна изменился");

    window.addEventListener("resize", handleResize);

    // Очистка при размонтировании

    return () => {

      window.removeEventListener("resize", handleResize);

    };

  }, []); // Пустые зависимости: слушатель добавляется только один раз, при монтировании

  return;

}

**Сложные обновления с зависимостями**

Если эффект зависит от какого-либо состояния или пропсов, вы можете указать их в массиве зависимостей:

import { useState, useEffect } from "react";

function App() {

  const [count, setCount] = useState(0);

  useEffect(() => {

    console.log(`Текущий счетчик: $

{count}`);

    document.title = `Вы нажали

${count} раз`;

    return () => {

      console.log("Очистка эффекта (если необходима)");

    };

  }, [count]); // Эффект сработает при изменении count

  return (

      Вы нажали {count} раз(а)

       setCount(count + 1)}>Нажми меня

  );

}

**Очистка таймеров**

Управление таймерами можно организовать с помощью возврата функции очистки:

import { useState, useEffect } from "react";

function App() {

  const [count, setCount] = useState(0);

  useEffect(() => {

    const timer = setInterval(() => {

      setCount((prevCount) => prevCount + 1);

    }, 1000);

    // Очистка эффекта — удаляем таймер при размонтировании

    return () => clearInterval(timer);

  }, []); // Таймер устанавливается только один раз при монтировании

  return Счётчик: {count};

}

**Синхронизация пропсов или локального состояния**

Иногда нужно синхронизировать состояние или реквизиты компонента с внешними системами:

import { useState, useEffect } from "react";

function App({ userId }) {

  const [user, setUser] = useState(null);

  useEffect(() => {

    // обновляем данные пользователя при смене userId

    fetch([https://api.example.com/user/${userId}](https://api.example.com/user/$%7buserId%7d))

      .then((response) => response.json())

      .then((data) => setUser(data));

  }, [userId]); // Срабатывает при изменении userId

  return {user && {user.name}};

}

**Ключевые моменты о useEffect**

Компонент рендерит сначала, а потом запускается эффект.

   Это означает, что useEffect выполняется асинхронно и после завершения текущего рендеринга.

**Очистка эффектов.**

   Если вернуть функцию из эффекта, React выполнит её для очистки предыдущего эффекта перед следующим вызовом (или перед размонтированием).

Не мутируйте состояния или DOM внутри эффекта.

   Эффект нужно использовать для управления сторонними функциями и взаимодействием с внешними источниками.

Список зависимостей обязателен для управления обновлениями.

   Либо указывайте зависимые переменные, либо оставляйте массив пустым ([]), чтобы ограничивать перезапуск эффекта.

Бесконечные циклы.

   Будьте осторожны с состояниями, используемыми внутри эффектов, чтобы не вызывать бесконечный цикл перезапуска.

Итог

Хук useEffect универсален и заменяет практически все методы управления эффектами в классовых компонентах. Используйте его для выполнения побочных действий (API-запросы, подписки на события, манипуляции с DOM и т. д.), а также не забывайте учитывать правильный массив зависимостей и очищать ресурсы, чтобы избежать утечек памяти.