Лекция 3. Состояние и побочные эффекты

Изучите, как взаимодействие пользователя и внешние данные изменяют пользовательский интерфейс с течением времени. Сосредоточьтесь на управлении состоянием и обработке побочных эффектов с помощью хуков React.

1. Управление состоянием с помощью useState

Хук useState — это фундаментальный хук React, позволяющий функциональным компонентам иметь состояние. Он предоставляет способ объявлять и обновлять переменные состояния внутри компонента.

Синтаксис: Чтобы использовать useStateхук, мы вызываем его и передаем в качестве аргумента начальное состояние (initialState). Он возвращает массив с двумя элементами: текущим значением состояния (state) и функцией (setState) для обновления состояния.

```
const [state, setState] = useState(initialState);
```

- В качестве имени state переменной можно выбрать любое слово. Рекомендуется использовать имя переменной, точно описывающее, какое состояние сохраняется или обновляется, например inputValue, page, number, , name, и так далее;
- Аналогично, при использовании setStateфункции у нас есть возможность гибко выбирать имя функции. Однако, согласно общепринятой практике, следует определенному шаблону: setStateимя функции должно начинаться с «set», за которым следует имя соответствующей переменной состояния. Например, если переменная state— mail, то соответствующая setStateфункция обычно называется setMail.

Примечание

При вызове метода `getState() setState` React перерисовывает компонент и обновляет состояние новым значением. Важно отметить, что при использовании useState`getState()` переменная состояния не обязательно должна быть объектом. Она может быть примитивным значением (например, числом, строкой или логическим значением) или комплексным значением (например, массивом или объектом).

Пример 1:

Давайте создадим Counter компонент, который будет поддерживать собственное состояние. При нажатии кнопки «Увеличить» мы обновим состояние и запустим перерисовку компонента.

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

const Counter = () => {
  const [count, setCount] = useState(0);
```

```
  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
};
```

В этом примере useStatexyk используется для объявления переменной состояния countc начальным значением 0. setCountФункция используется для обновления countпеременной при каждом нажатии кнопки.

Построчное объяснение:

- Строка 1: Мы импортируем useStatexyk из библиотеки React, чтобы использовать его функциональность;
- Строка 3: Мы определяем Counterкомпонент, используя стандартный синтаксис функций;
- Строка 4: Мы инициализируем состояние с помощью useStateобработчика;
  1. countобозначает значение счетчика. Его начальное значение равно 0, указанное в скобках useState(0);
  2. setCountЭто функция, которая обновляет состояние по мере необходимости.
- Строки 6-8: Эта стрелочная функция JavaScript обрабатывает логику обновления состояния. Она выполняется при нажатии кнопки "increment". Внутри функции мы используем её setCountдля обновления состояния;
- Строки 10-15: отображают разметку компонента. В строке 12 countотображается текущее значение состояния ( ). Кнопка в строке 13 использует onClickатрибут для определения функции, которая будет выполняться при нажатии. В данном случае мы передаем функцию increment.

Пример 2:

Давайте создадим Formкомпонент с его независимым состоянием. Мы предлагаем пользователю ввести свое имя в поле ввода, и в зависимости от его ввода мы будем изменять отображаемое содержимое.

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

const Form = () => {
```

```
    const [userName, setUserName] = useState("");

    const handleChange = (event) => {
      const inputValue = event.target.value;
      setUserName(inputValue);
    };

    return (
      <div>
        <input
          type="text"
          value={userName}
          onChange={handleChange}
          placeholder="Your name"
        />
        <p>Hello, {userName}!</p>
      </div>
    );
  };
```

В этом примере useStateхук используется для объявления переменной состояния userNameс начальным значением в виде пустой строки. Функция setUserNameобновляет userNameпеременную всякий раз, когда это необходимо.

Построчное объяснение:

- Строка 1: Мы импортируем useStateхук из библиотеки React, чтобы использовать его функциональность;
- Строка 3: Мы определяем Formкомпонент, используя стандартный синтаксис функций;
- Строка 4: Мы устанавливаем начальное состояние с помощью useStateобработчика событий;

1. userNameпредставляет значение поля ввода, изначально установленное как пустая строка (""), указанная в скобках useState("");
2. setUserNameЭто функция, которая позволяет нам обновлять состояние по мере необходимости.

- Строки 6-9: Эта стрелочная функция JavaScript обрабатывает логику обновления состояния. Она срабатывает, когда пользователь что-то вводит в поле. Мы получаем значение поля ввода внутри функции, используя метод `.` event.target.value. Затем мы используем эту setUserNameфункцию для обновления состояния значением из поля ввода;
- Строки 11-21: Отображается разметка компонента.

1. В строке 15 мы присваиваем значение userNameв качестве начального значения для входных данных, используя valueатрибут;

2. В строке 16 мы используем onChangeатрибут для указания функции, которая будет вызвана при изменении входных данных.

2. Обработка побочных эффектов с помощью useEffect

Этот useEffectхук позволяет синхронизировать компонент с внешними факторами/сервисами, включая получение данных, подписки, ручные изменения и т. д.

Синтаксис:

Первый аргумент (setup) — это **стрелочная функция** , которая будет выполняться после каждого рендеринга. Второй аргумент (dependencies) — это **необязательный массив** зависимостей. Если они dependenciesуказаны, эффект будет повторно выполнен только в том случае, если одна из зависимостей изменилась с момента последнего рендеринга. Если массив зависимостей опущен, эффект будет выполняться после каждого рендеринга.

```
useEffect(setup, dependencies)
```

Поскольку нам известно, что это **setup** должна быть стрелочная функция и **dependencies** массив, мы получаем следующую запись о **useEffect** хуке.

```
useEffect(() => {
  // Something that we need to do
  // after the component is rendered or updated
}, [<optional_dependencies>])
```

Примечание

Компоненты React часто используют комбинацию хуков `useEffec t<react -компоненты>` и useState`<react-компоненты>`. Эти хуки работают в связке для управления состоянием и побочными эффектами внутри компонентов.

Пример 1:

Давайте создадим Articles компонент, который будет использовать useEffect хук для присвоения данных состоянию articles. Это отличная возможность изучить мощное сочетание хуков React.

```
import { useEffect, useState } from "react";
import fetchData from "../service/api";

const Articles = () => {
  const [articles, setArticles] = useState([]);

  useEffect(() => {
    fetchData()
```

```
      .then((resp) => {
        setArticles(resp.jokes);
      })
      .catch((error) => {
        console.error(error);
      });
  }, []);

  return (
    // Render some markup based on the articles data
  );
};
```

В этом компоненте мы используем хук useEffectдля обеспечения articles заполнения состояния данными. Как упоминалось ранее, функция useEffect выполняется после каждого рендеринга, гарантируя, что данные будут отображены пользователю, если они получены. Это обеспечивает бесперебойную работу с пользователем и актуальный контент.

Построчное объяснение:

- Строка 1: Мы импортируем useEffect хуки useStateиз библиотеки React, чтобы использовать её функциональность;
- Строка 2: Мы импортируем fetchData функцию из "../service/api". Эта функция отвечает за выполнение запроса к API и получение данных;
- Строка 4: Articles Компонент определен с использованием стандартного синтаксиса функций;
- Строка 5: Мы инициализируем состояние с помощью useStatехука, представляющего начальное значение переменной articles. В этом примере это пустой массив;
- Строки 7-15: Реальный пример использования крючка useEffect;
  - Строки 7 и 15: это синтаксис. Именно так мы будем его использовать;
  - Строка 8: fetchDataвызывается функция. Ожидается, что эта функция выполнит запрос к API и вернет промис;
  - Строки 9-11: Когда промис разрешается (thenблок), он получает respобъект. Данные извлекаются с respпомощью resp.jokes, который устанавливается в articlesсостояние с помощью setArticles;
  - Строки 12-14: Если во время запроса к API (в блоке) возникает ошибка catch, она выводится в консоль с помощью console.error.
- Строки 17-19: Отображается разметка компонента.

Пример 2:

Давайте создадим Counterкомпонент для отслеживания countersзначения. Задача состоит в том, чтобы записывать

значение переменной counterкаждый раз, когда оно изменяется. Для этого мы будем использовать useEffectхук с массивом зависимостей, состоящим из этой counterпеременной.

```
import React, { useState, useEffect } from "react";

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

  useEffect(() => {
    console.log("Count has changed:", count);
  }, [count]);

  const handleIncrement = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={handleIncrement}>Increment</button>
    </div>
  );
};
```

Построчное объяснение:

- Строка 1: Мы импортируем useEffectхуки useStateиз библиотеки React, чтобы использовать её функциональность;
- Строка 3: Стандартный синтаксис функции определяет компонент "Счетчик";
- Строка 4: Мы инициализируем состояние с помощью useStatехука, представляющего начальное значение переменной count. В этом примере это 0;
- Строки 6-8: Реальный сценарий использования крючка useEffect;
  - Строка 7: эта логика будет выполняться всякий раз, когда изменяется значение в массиве зависимостей. В данном случае это countпеременная;
  - Строка 8: массив зависимостей. Мы сообщаем React, что при изменении значения переменной countкод должен выполняться внутри useEffectфункции-хука.
- Строки 14-19: Отображается разметка компонента.

Пожалуйста, уделите немного времени, чтобы изучить консоль и понаблюдать за логикой выполнения стрелочной функции внутри useEffectхука. Стрелочная функция сначала выполняется при первоначальной отрисовке, а затем вызывается снова всякий раз,

когда counterизменяется значение переменной. Вы можете убедиться в этом, посмотрев соответствующие записи в консоли.

Вопросы для самопроверки:

1. Какое из следующих утверждений о хуке **useState** является верным?
2. Каково назначение функции **setState**, возвращаемой хуком **useState**?
3. Как задать начальное значение состояния при использовании хука **useState**?
4. Каково назначение хука **useEffect** в React?
5. Какие два основных аргумента принимает хук **useEffect**?