

React: Controlled and uncontrolled components useEffect

НАШИ ПРАВИЛА



Включенная камера



Вопросы по поднятой руке



Не перебиваем друг друга



Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору



Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

Повторим;)

■ Какие способы стилизации компонентов вы знаете?

■ Как изменить состояние компонента?


■ Для чего используется styled из emotion?

■ Как создать шаблон стилей с помощью emotion?

ЦЕЛЬ

Понять разницу между контролируемым и неконтролируемым компонентом, изучить хук `useEffect`

ПЛАН ЗАНЯТИЯ

- 
1. Controlled components
 2. Uncontrolled components
 3. `useEffect`

Controlled and uncontrolled components



В React компоненты могут быть разделены на две основные категории:

- **контролируемые (controlled)**
- **неконтролируемые (uncontrolled).**

Эти термины относятся к тому, как компонент управляет своим состоянием и данными.



Контролируемые компоненты

Контролируемый компонент - это компонент, который управляет своим состоянием с помощью React.

Любые изменения ввода пользователя или другие события приводят к обновлению состояния компонента через `setState`.

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

const ControlledComponent = () => {
  const [inputValue, setInputValue] = useState('');

  const handleChange = (e) => {
    setInputValue(e.target.value);
  };

  return (
    <input
      type="text"
      value={inputValue}
      onChange={handleChange}
    />
  );
};
```


Неконтролируемые компоненты

Неконтролируемый компонент - это компонент, в котором состояние не контролируется React.

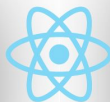
Вместо этого, данные хранятся в DOM

Когда выбирать между контролируруемыми и неконтролируемыми компонентами:

- **Контролируемые компоненты:** Полезны, когда React должен полностью контролировать состояние компонента, особенно при работе с формами. Позволяют React легко управлять вводом и обновлять UI в ответ на изменения.
- **Неконтролируемые компоненты:** Могут быть удобными, когда вам нужно интегрироваться с кодом или библиотеками, которые управляют DOM напрямую. Они также могут уменьшить необходимость в использовании состояния и `setState`.

useEffect

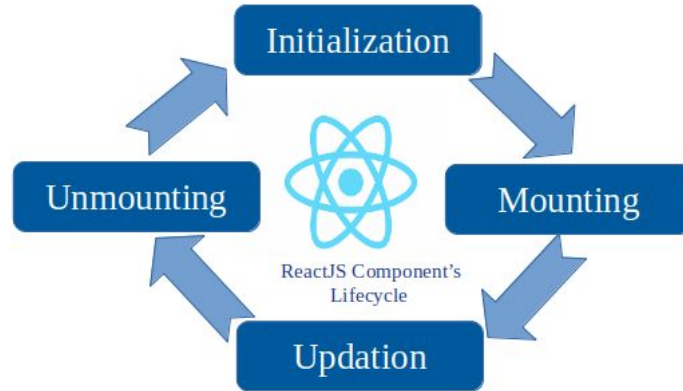
```
useEffect(() => {  
  ...  
}, [])
```



```
componentDidMount() {}  
componentDidUpdate() {}  
componentWillUnmount() {}
```

3 этапа жизни компонента:

- визуализация компонента - **Mounting**(монтаж);
- обновление компонента - **Updating**(обновление);
- удаление компонента из DOM - **Unmounting** (размонтаж).



Что такое useEffect?

useEffect — это хук, который можно использовать для выполнения действия в определенные моменты жизненного цикла компонента.

useEffect принимает два параметра.

- Первый аргумент — это функция обратного вызова
- Второй аргумент – массив зависимостей. Второй аргумент является необязательным.

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

Использование useEffect сразу после создания элемента (Mounting)

Если мы передаем второй аргумент в виде пустого массива, побочный эффект в функции обратного вызова сработает только один раз при первой визуализации компонента.

```
function MyComponent() {  
  useEffect(() => {  
    // This side effect will only run once, after the first render  
  }, [])  
}
```

Вариантом использования для этого может быть получение данных из API

Использование useEffect для при обновлении (Updating)

- При каждом рендере компонента

```
function MyComponent() {  
  useEffect(() => {  
    // The side effect will run after every render  
  })  
}
```

- при изменении значений зависимостей, переданных в массиве

```
import { useEffect, useState } from 'react'  
function MyComponent({ prop }) {  
  const [state, setState] = useState('')  
  useEffect(() => {  
    // the side effect will only run when the props or state changed  
  }, [prop, state])  
}
```

Примером использования для этого может быть функция поиска.

Использование useEffect для при размонтировании (Unmounting)

Это функция очистки, которая позволяет нам остановить побочные эффекты непосредственно перед размонтированием компонента.

```
function MyComponent() {  
  useEffect(() => {  
    // this side effect will run after every render  
    return () => {  
      // this side effect will run before the component is unmounted  
    }  
  })  
}
```




**re-rendering of
components**

Повторный рендеринг компонентов в React — это процесс обновления визуального представления компонента в результате изменения его состояния или пропсов. React воссоздает виртуальное дерево компонентов, сравнивает его с предыдущим состоянием и определяет минимальное количество изменений, необходимых для обновления пользовательского интерфейса.



Случаи повторного рендеринга

1. Изменение состояния (useState)

Когда вызывается функция изменения состояния, компонент перерендеривается с новым состоянием

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

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

  const increment = () => {
    setCount(count + 1); // Изменение состояния
  };

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

Случаи повторного рендеринга

2.Изменение пропсов

Когда компонент получает новые пропсы, он перерендеривается с новыми значениями пропсов

```
import React from 'react';

function Greeting({ name }) {
  return <p>Hello, {name}!</p>;
}

// Повторный рендеринг при изменении пропсов
<Greeting name="Alice" />
<Greeting name="Bob" />
```



Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH