

Цели занятия

- узнать больше про хуки в React;
- повторить useState, изучить useEffect;
- разобрать что такое роутинг и использовать react-router-dom;

Полезные материалы

- [Введение в хуки React](#)
- [Как использовать useEffect](#)
- [Документация React Router \(eng\)](#)

Что такое хуки в React?

Изначально в React все основные компоненты создавались в виде классов и так было на протяжении пяти лет. Но за эти годы накопилось довольно много проблем с таким подходом, поэтому разработчики искали пути решения этих проблем, что в итоге привело к функциональным компонентам и хукам.

Хуки это определенные функции в React, которые позволяют использовать состояние, вызывать побочные эффекты, оптимизировать сложные расчёты и так далее.

С помощью функциональных компонентов и хуков можно использовать все возможности React и в итоге это будет гораздо проще, чем реализовать то же самое с помощью классов. Причём только внутри функциональных компонентов можно использовать хуки. Подробнее о хуках и причинах их использовать [читайте в данной статье](#).

На данном курсе не разбираются классовые компоненты, потому что в современной разработке их используют всё реже и реже. Мы будем изучать только то, с чем вы будете работать ежедневно — хуки и функциональные компоненты.

Правила хуков

Хуки — обычные JavaScript-функции, но существует два правила, которым нужно следовать. Причём в проектах, созданных с помощью CreateReactApp есть инструмент, который автоматически проверяет что вы соблюдаете данные правила.

1. Используйте хуки только на верхнем уровне

Не вызывайте хуки внутри циклов, условных операторов или вложенных функций. Вместо этого всегда используйте хуки только внутри React-функций, до возврата какого-либо значения из них.

Исполнение этого правила гарантирует, что хуки вызываются в одинаковой последовательности при каждом рендере компонента. Это позволит React правильно сохранять состояние хуков между множественными вызовами `useState` и `useEffect`.

2. Вызывайте хуки только из React-функций

Не вызывайте хуки из обычных функций JavaScript.

Вместо этого можно:

- Вызывать хуки из функционального компонента React.
- Вызывать хуки из пользовательского хука.

Следуя этому правилу, можно гарантировать, что вся логика состояния компонента чётко видна из исходного кода.

useState

Данный хук позволяет использовать состояние внутри любого функционального компонента, подробно мы разбирали его на предыдущем занятии, поэтому сейчас останавливаться на нём не будем.

useEffect

Второй базовый хук, который очень часто используется при создании компонентов. Позволяет использовать любые побочные эффекты внутри компонента, а также выполнять какой-то код строго после того, как компонент будет отрендерен в браузере или удален со страницы.

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

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

  useEffect(() => {
    // Обновляем заголовок документа с помощью API браузера
    document.title = `Вы нажали ${count} раз`;
  });

  return (
    <div>
      <p>Вы нажали {count} раз</p>
      <button onClick={() => setCount(count + 1)}>Нажми на меня</button>
    </div>
  );
}
```

Побочными эффектами в React-компонентах могут быть: загрузка данных, оформление подписки и изменение DOM вручную.

Неважно, называете ли вы эти операции «побочными эффектами» (или просто «эффектами») или нет, скорее всего вам доводилось ранее использовать их в своих компонентах.

Существует два распространённых вида побочных эффектов в компонентах React: компоненты, которые требуют и не требуют сброса. Давайте рассмотрим оба примера более детально.

Эффекты без сброса

Иногда мы хотим **выполнить дополнительный код после того, как React обновил DOM**. Сетевые запросы, изменения DOM вручную, логирование — всё это примеры эффектов, которые не требуют сброса. После того как мы запустили их, можно сразу забыть о них, ведь больше никаких дополнительных действий не требуется.

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

// Это эффект без сброса
useEffect(() => {
  document.title = `Вы нажали ${count} раз`;
});
```

Что же делает `useEffect`?

Используя этот хук, вы говорите React сделать что-то после рендера. React запомнит функцию (то есть «эффект»), которую вы передали и вызовет её после того, как внесёт все изменения в DOM. В этом эффекте мы устанавливаем заголовок документа, но мы также можем выполнить запрос данных.

Почему же мы вызываем `useEffect` непосредственно внутри компонента?

Это даёт нам доступ к переменной состояния `count` (или любым другим пропсам) напрямую из эффекта. Нам не нужен специальный API для доступа к этой переменной — она уже находится у нас в области видимости функции.

Выполняется ли `useEffect` после каждого рендера?

Разумеется! По умолчанию он будет выполняться после каждого рендера и обновления. Мы рассмотрим, как настраивать это немного позже. React гарантирует, что он запустит эффект только после того, как DOM уже обновился.

Эффекты со сбросом

Ранее мы рассмотрели побочные эффекты, которые не требуют сброса. Однако есть случаи, когда сброс всё же необходим. Например, нам может потребоваться установить подписку на какой-нибудь внешний источник данных. В этом случае очень важно выполнять сброс, чтобы не случилось утечек памяти! Давайте сравним, как мы можем это реализовать с помощью классов и хуков.

```
// Это эффект со сбросом
useEffect(() => {
  const intervalId = setInterval(() => {
    document.title = `Прошло уже ${count} секунд!`;
    setCount((prevCount) => prevCount + 1);
  }, 1000);

  // Это и есть сброс — выполнится после удаления компонента
  // Если не напишем сброс, то интервал будет работать постоянно
  return () => {
    clearInterval(intervalId);
  };
});
```

Зачем мы вернули функцию из нашего эффекта?

Это необязательный механизм сброса эффектов. Каждый эффект может вернуть функцию, которая сбросит его. Это даёт нам возможность объединить вместе логику оформления и отмены подписки. Они, всё-таки, часть одного и того же эффекта!

Когда именно React будет сбрасывать эффект?

React будет сбрасывать эффект перед тем, как компонент размонтируется. Однако, как мы уже знаем, эффекты выполняются не один раз, а при каждом рендере. Вот почему React также сбрасывает эффект из предыдущего рендера, перед тем, как запустить следующий.

Зависимости в useEffect

Чуть выше мы рассмотрели эффекты, которые имели только один параметр — функцию с побочным эффектом. Но также любой эффект может принимать и второй параметр, который является массивом и называется его зависимостями. Если мы передаём зависимости, то эффект будет работать только при первом рендере и после изменения этих зависимостей.

На примерах:

```
// Зависимостей нет, работает при каждом рендере
useEffect(() => {
  document.title = `Вы нажали ${count} раз`;
});
```

```
// Зависимости указаны, но они пустые
// Сработает только один раз при первом рендере
useEffect(() => {
  document.title = `Вы нажали ${count} раз`;
}, []);
```

```
// В качестве зависимости указано count
// Сработает при первом рендере и при любом изменении count
useEffect(() => {
  document.title = `Вы нажали ${count} раз`;
}, [count]);
```

Рекомендуется всегда указывать максимально точный список зависимостей, чтобы эффект срабатывал только тогда, когда необходимо, а не на каждый рендер компонента.

Обратите внимание, что зависимости это всегда массив и поэтому их можно указывать сколько угодно. Но если у вас по какой-то причине становится слишком много зависимостей (больше 5, например), то это часто говорит, что код следует упростить, разбить на несколько эффектов и так далее.

Роутинг в приложениях

Роутинг это возможность отображать тот или иной компонент на экране в зависимости от URL-адреса, на котором находится пользователь. Чаще всего это используется для того, чтобы показывать компонент-страницу только по определенному адресу, который мы указали. Также в задачи роутинга входит переход от одной страницы к другой с помощью ссылок, но при использовании React обычно это не требует полной перезагрузки страницы, поэтому страницы отображаются почти мгновенно.

Роутинг не входит в библиотеку React и поэтому для роутинга можно выбирать разные сторонние решения от других разработчиков. В рамках данного курса мы будем использовать самую популярную библиотеку для этого — [ReactRouter](#).

Не стоит переживать по поводу данной библиотеки: её используют крупнейшие компании, она активно разрабатывается и обновляется, у неё отличная документация, в интернете есть огромное количество статей с её использованием, а также ответ на любую проблему, которая может появиться.

Она доступна в NPM и перед тем как начать знакомство необходимо установить её в ваш учебный проект: **npm install react-router-dom**

Используем ReactRouter

В рамках данного занятия мы познакомимся лишь с несколькими самыми важными компонентами этой библиотеки, а более сложные примеры вы всегда сможете посмотреть в официальной документации.

Итак, давайте наконец-то сделаем наш сайт мультистраничным:

```
import React from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";

import PageIndex from "../components/PageIndex/PageIndex";
import PageProduct from "../components/PageProduct/PageProduct";
import PageNotFound from "../components/PageNotFound/PageNotFound";

function App() {
  return (
    // Эти два компонента обязательно нужно добавить,
    // чтобы роутинг в приложении работал корректно
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<PageIndex />} />

        /* Параметр path указываем по какому адресу будут доступна эта страница */
        <Route path="/product" element={<PageProduct />} />

        /* Если человек ввел другой адрес, то показываем страницу 404 */
        <Route path="*" element={<PageNotFound />} />
      </Routes>
    </BrowserRouter>
  );
}
```

Основным компонентом в роутинге является `Route`, который задаёт правила по какому адресу следует открывать ту или иную страницу. Всё остальное библиотека сделает самостоятельно.

А для того, чтобы сделать переходы между страницами, библиотека предоставляет компонент `Link`. По сути это обычная ссылка как и в HTML, но с той лишь разницей, что она по умолчанию использует современные возможности браузера и позволяет не перезагружать страницу полностью.

Больше примеров с роутингом и ответы на вопросы будут на онлайн-занятии, обязательно заходите!