МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«КАЛУЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. К. Э. ЦИОЛКОВСКОГО»

Инженерно-технологический институт
кафедра информатики и информационных технологий

О Т Ч Е Т

ДИСЦИПЛИНА: "Прикладное программирование"
Выполнил: студент гр. Б-ИСиТ-41 Варавин Е.А._____
Проверил: Винокуров А.В._____

Дата сдачи (защиты) лабораторной работы:
Результаты сдачи (защиты):
Количество рейтинговых баллов
Оценка

Калуга, 2021 г.

Введение в React - хуки

Хуки доступны в версии React 16.8. Они позволяют использовать состояние и другие функции React, освобождая от необходимости писать класс.

Хуки — обратно совместимы.

Обратная совместимость:

- Полностью на ваше усмотрение. Вы можете попробовать хуки в одних компонентах, не изменяя код в других. Хуки не обязательно использовать или изучать прямо сейчас.
- 100% обратно совместимы. Хуки не содержат изменений, которые могут поломать ваш существующий код.

Основные хуки

useState - Возвращает значение с состоянием и функцию для его обновления.

Во время первоначального рендеринга возвращаемое состояние (state) совпадает со значением, переданным в качестве первого аргумента (initialState).

Функция setState используется для обновления состояния. Она принимает новое значение состояния и ставит в очередь повторный рендер компонента.

Во время последующих повторных рендеров первое значение, возвращаемое useState, всегда будет самым последним состоянием после применения обновлений.

Функциональные обновления

Если новое состояние вычисляется с использованием предыдущего состояния, вы можете передать функцию в setState. Функция получит предыдущее значение и вернёт обновлённое значение.

Если функция обновления возвращает абсолютно такой же результат как и текущее состояние, то последующие повторные рендеры будут полностью пропущены.

Ленивая инициализация состояния

Аргумент initialState — это состояние, используемое во время начального рендеринга. В последующих рендерах это не учитывается. Если начальное состояние является результатом дорогостоящих вычислений, вы можете вместо этого предоставить функцию, которая будет выполняться только при начальном рендеринге

Досрочное прекращение обновления состояния

Если вы обновите состояние хука тем же значением, что и текущее состояние, React досрочно выйдет из хука без повторного рендера дочерних элементов и запуска эффектов.

useEffect - Принимает функцию, которая содержит императивный код, возможно, с эффектами.

Мутации, подписки, таймеры, логирование и другие побочные эффекты не допускаются внутри основного тела функционального компонента (называемого этапом рендеринга React). Это приведёт к запутанным ошибкам и несоответствиям в пользовательском интерфейсе.

Вместо этого используйте useEffect. Функция, переданная в useEffect, будет запущена после того, как рендер будет зафиксирован на экране. Думайте об эффектах как о лазейке из чисто функционального мира React в мир императивов.

По умолчанию эффекты запускаются после каждого завершённого рендеринга, но вы можете решить запускать их только при изменении определённых значений.

Очистка эффекта

Часто эффекты создают ресурсы, которые необходимо очистить (или сбросить) перед тем, как компонент покидает экран. Чтобы сделать это, функция переданная в useEffect, может вернуть функцию очистки.

Функция очистки запускается до удаления компонента из пользовательского интерфейса, чтобы предотвратить утечки памяти. Кроме того, если компонент рендерится несколько раз (как обычно происходит), предыдущий эффект очищается перед выполнением следующего эффекта.

Порядок срабатывания эффектов

В отличие от componentDidMount и componentDidUpdate, функция, переданная в useEffect, запускается во время отложенного события после разметки и отрисовки. Это делает хук подходящим для многих распространённых побочных эффектов, таких как настройка подписок и обработчиков событий, потому что большинство типов работы не должны блокировать обновление экрана браузером.

Однако не все эффекты могут быть отложены. Например, изменение DOM, которое видно пользователю, должно запускаться синхронно до следующей отрисовки, чтобы пользователь не замечал визуального несоответствия. (Различие концептуально схоже с пассивным и активным слушателями событий.) Для этих типов эффектов React предоставляет один дополнительный хук, называемый useLayoutEffect. Он имеет ту же сигнатуру, что и useEffect, и отличается только в его запуске.

Хотя useEffect откладывается до тех пор, пока браузер не выполнит отрисовку, он гарантированно срабатывает перед любыми новыми рендерами. React всегда полностью применяет эффекты предыдущего рендера перед началом нового обновления.

Условное срабатывание эффекта

По умолчанию эффекты запускаются после каждого завершённого рендера. Таким образом, эффект всегда пересоздаётся, если значение какой-то из зависимости изменилось.

Чтобы реализовать это, передайте второй аргумент в useEffect, который является массивом значений, от которых зависит эффект.

Массив зависимостей не передаётся в качестве аргументов функции эффекта. Тем не менее, в теории вот что происходит: каждое значение, на которое ссылается функция эффекта, должно также появиться в массиве зависимостей. В будущем достаточно продвинутый компилятор сможет создать этот массив автоматически.

useContext - Принимает объект контекста (значение, возвращённое из React.createContext) и возвращает текущее значение контекста для этого контекста. Текущее значение контекста определяется пропом value ближайшего <MyContext.Provider> над вызывающим компонентом в дереве.

Когда ближайший <MyContext.Provider> над компонентом обновляется, этот хук вызовет повторный рендер с последним значением контекста, переданным этому провайдеру MyContext. Даже если родительский компонент использует React.memo или реализует shouldComponentUpdate, то повторный рендер будет выполняться, начиная с компонента, использующего useContext.

Запомните, аргументом для useContext должен быть непосредственно сам объект

Компонент, вызывающий useContext, всегда будет перерендериваться при изменении значения контекста.

useReducer - Альтернатива для useState. Принимает редюсер типа (state, action) => newState и возвращает текущее состояние в паре с методом dispatch.

Хук useReducer обычно предпочтительнее useState, когда у вас сложная логика состояния, которая включает в себя несколько значений, или когда следующее состояние зависит от предыдущего. useReducer также позволяет оптимизировать производительность компонентов, которые запускают глубокие обновления, поскольку вы можете передавать dispatch вместо колбэков.

Указание начального состояния

Существует два разных способа инициализации состояния useReducer. Вы можете выбрать любой из них в зависимости от ситуации. Самый простой способ — передать начальное состояние в качестве второго аргумента

Ленивая инициализация

Вы также можете создать начальное состояние лениво. Для этого вы можете передать функцию init в качестве третьего аргумента. Начальное состояние будет установлено равным результату вызова init(initialArg).

Это позволяет извлечь логику для расчёта начального состояния за пределы редюсера. Это также удобно для сброса состояния позже в ответ на действие

Досрочное прекращение dispatch

Если вы вернёте то же значение из редюсера хука, что и текущее состояние, React выйдет без перерисовки дочерних элементов или запуска эффектов.