

Цели занятия

- узнать что такое Redux и когда его стоит использовать;
- познакомиться с основными принципами Redux;
- создать своё первое хранилище с помощью Redux;

Полезные материалы

- [Официальная документация Redux Toolkit](#)
- [Базовое руководство по Redux Toolkit](#)

Что такое Redux?

Redux — это библиотека для предсказуемой работы с глобальным состоянием в приложениях, которые обладают большим количеством сущностей, т.е. имеют довольно обширный функционал.

Название читается как «Редакс» и составлено из двух слов: reduce и flux. Reduce — это функция, которая приводит большую структуру данных к одному значению. А [Flux](#) это архитектурный принцип, которым вдохновлялись разработчики редакса.

Глобальное состояние приложения — это часть приложения, которая используется, учитывается и изменяется в разных модулях и компонентах этого приложения.

Глобальное состояние должно быть доступно из любой части приложения, а при его изменении компоненты приложения должны произвести необходимые вычисления. То есть каждый компонент нашего приложения может как получить состояние, так и вызвать действие, которое изменит это состояние.

Когда стоит использовать Redux?

Ошибка многих новичков заключается в том, что когда они впервые узнают о Redux, то пытаются выносить в Redux абсолютно все состояния компонентов, чем сильно усложняют своё же приложение.

Redux это **глобальное** состояние приложения и должно хранить в себе только те состояния, которые влияют сразу на несколько компонентов. Либо если это состояние может изменяться из различных частей приложения.

Например, у нас есть форма добавления отзыва, которая имеет внутри себя несколько состояний, чтобы отображать ошибки и т.д. В данном случае лучше использовать **локальное** состояние компонента (то есть `useState`), так как мы управляем им только внутри одного компонента.

Но также у нас есть ещё и корзина пользователя — количество товаров в корзине мы отображаем в шапке, а вот добавлять в корзину и удалять из корзины мы будем с помощью компонентов, которые находятся внутри страницы. То есть в данном случае, мы работаем с состоянием сразу **из нескольких разных частей** и поэтому нам лучше использовать Redux.

Также Redux позволяет получить полную историю того, как изменялось значение вашего хранилища с течением времени и поэтому может быть полезен, когда вам необходимо где-то использовать подобную историю изменений.

Кроме того, Redux в чистом виде никак не связан с React напрямую и вы можете использовать Redux в любых JavaScript приложениях, если вам необходимо работать с состоянием.

Анатомия Redux

Хранилище (store) — это обычный JavaScript объект, который хранит внутри себя все необходимые **глобальные данные** нашего приложения. Оно позволяет нам в любой момент времени получать их из любого компонента, а также подписываться на изменения. Любое изменение хранилища влияет на те компоненты, которые его используют, т.е. компоненты будут автоматически обновляться.

Действие (action) — как правило, если у нас есть хранилище, то должны быть и действия, которые каким-либо образом на него влияют. Например, если мы говорим про корзину, то как минимум у нас должны быть действия для удаления и добавления товара в корзину. Действия вызываются внутри событий, которые происходят в компонентах с помощью функции `dispatch`

Редуктор (reducer) — при запуске действия обязательно что-то происходит и состояние приложения изменяется. Это работа редукторов. Редуктор — это **чистая** функция, которая вычисляет следующее состояние дерева на основании его предыдущего состояния и применяемого действия.

Чего не должен делать редуктор?

- редуктор это всегда чистая функция, поэтому он не должен мутировать аргументы, которые он получает, вместо этого он возвращает новое состояние (принцип неизменяемости)
- не должен иметь побочных эффектов, для этого в Redux есть отдельный функционал, который мы рассмотрим позднее (либо вы можете использовать `useEffect`)
- не должен вызывать функции, которые не относятся к состоянию, а также использовать нечистые функции, такие как `Math.random()`, так как сам редуктор тоже станет нечистым

От теории к практике

За несколько лет своего развития Redux получил много возможностей и на текущий момент имеет сразу несколько библиотек, каждая из которых отвечает за что-то свое. Мы с вами не будем использовать Redux в чистом виде, вместо этого мы будем использовать библиотеку [Redux Toolkit](#), которая значительно упрощает работу с Redux. Это рекомендуемый способ от самих разработчиков Redux и в современном мире так делают чаще всего.

Устанавливаем библиотеки

- [@reduxjs/toolkit](#) — упрощает работу с Redux и имеет некоторые дополнительные возможности
- [react-redux](#) — имеет набор функций и хуков, которые сделаны специально для React-приложений

Создаем хранилище

Хранилище это основа всего, оно одно на всё приложение, но может состоять из нескольких частей. Для начала мы сделаем корзину пользователя

```
// Внутри файла store.js
import { configureStore } from "@reduxjs/toolkit";

export const store = configureStore({
  reducer: {},
});
```

Подключаем хранилище в приложение

```
// Файл index.js
import { createRoot } from "react-dom/client";
import { Provider } from "react-redux";
import { store } from "./store";
import App from "./App";
```

```

const rootElement = document.getElementById("root");
const root = createRoot(rootElement);

root.render(
  // Мы обернули наше приложение в провайдер
  // чтобы оно имело доступ к хранилищу
  <Provider store={store}>
    <App />
  </Provider>
);

```

Создаём корзину

Всё хранилище приложение состоит из нескольких частей, первая из наших частей это корзина товаров. Она может иметь несколько редукторов и действий, которые мы создаём в отдельном файле.

```

// Файл cart-reducer.js
import { createSlice } from "@reduxjs/toolkit";

export const cartSlice = createSlice({
  name: "cart",

  // Начальное состояние хранилища, товаров нет
  initialState: {
    products: [],
  },

  // Все доступные метода
  reducers: {
    // Добавить товар, первый параметр это текущее состояние
    // А второй параметр имеет данные для действия
    addProduct: (prevState, action) => {
      return {
        ...prevState,
        // Внутри action.payload информация о добавленном товаре
        // Возвращаем новый массив товаров вместе с добавленным
        products: [...prevState.products, action.payload],
      };
    },
  },
});

// Экспортируем наружу все действия
export const { addProduct } = cartSlice.actions;

```

```
// И сам редуктор тоже
export default cartSlice.reducer;
```

Добавляем корзину в хранилище

Снова открываем файл `store.js`, который мы создали на первом шаге и добавляем в него редуктор корзины:

```
import { configureStore } from "@reduxjs/toolkit";
import cartReducer from './cart-reducer';

export const store = configureStore({
  reducer: {
    cart: cartReducer,
  },
});
```

Вызываем действие внутри компонента

Осталось лишь научиться получать состояние внутри компонента и вызывать действие по клику на кнопку:

```
import { useSelector, useDispatch } from "react-redux";
import { addProduct } from "./cart-reducer";

function Cart() {
  // Получаем частичку глобального состояния
  const count = useSelector((state) => state.cart.products.length);
  const dispatch = useDispatch();

  function handleClickAdd() {
    // Создаём действие, вся информация, которую передаём
    // будет доступна в свойстве action.payload
    const action = addProduct({ id: 4884 });

    // Отправляем действие
    dispatch(action);
  }

  return (
    <div>
```

```
    <div>{`В корзине ${count} товаров`}</div>
    <button onClick={handleClickAdd}>Добавить товар</button>
  </div>
);
}
```

Поначалу вас может напугать количество кода, которое пришлось написать, но это действительно упрощает работу, когда ваше приложение становится сложнее и сложнее.

Обязательно заглядывайте на онлайн-занятие, чтобы лучше разобраться в принципе работы Redux и увидеть больше примеров!