<https://www.youtube.com/watch?v=Od5H_CiU2vM&ab_channel=UlbiTV>

Хорошая практика - типизировать селекторы и диспатчи (писать кастомные useSelector, useDispatch) - тогда упрощается работа с обращением к состоянию

1. Redux

Создать шаблон

npx create-react-app my-app --template redux-typescript

Либо подключение

npm i --save @reduxjs/toolkit react-redux @types/react-redux

store.js

import { TypedUseSelectorHook, useDispatch, useSelector } from 'react-redux';

import { Action, configureStore, ThunkAction } from '@reduxjs/toolkit';

import { rootReducer } from '../rStore/reducers/counetrReducer';

export const store = configureStore({

  reducer: {

    showLogo: rootReducer,

  },

});

export type AppDispatch = typeof store.dispatch;

export type RootState = ReturnType<typeof store.getState>;

export type AppThunk<ReturnType = void> = ThunkAction<ReturnType, RootState, unknown, Action<string>>;

export const useAppDispatch = () => useDispatch<AppDispatch>();

export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;

root-reducer.js

const initialState: any = {

  isShowed: true,

};

// eslint-disable-next-line @typescript-eslint/default-param-last

export function rootReducer(state: any = initialState, action: any) {

  switch (action?.type) {

    case 'SHOW\_LOGO':

      return {

        ...state,

        isShowed: action.payload.isShowed,

      };

    default:

      return state;

  }

}

actionsCreators.ts

  function showLogoActionCreator(value: boolean) {

    return {

      type: 'SHOW\_LOGO',

      payload: {

        isShowed: value,

      },

    };

  }

Обернуть приложение в реакт провайдер

      <Provider store={store}>

        <AppRoutes />

        <GlobalStyles />

      </Provider>

Использование

  const dispatch = useAppDispatch();

  const state = useAppSelector(state => state);

        <p>{state.showLogo.isShowed ? 'Показывать лого' : 'Не показывать лого'}</p>

        <Button

          type="primary"

          onClick={() => {

            dispatch(showLogoActionCreator(true));

          }}

        >

          Show

        </Button>

Общий подход – создается стор, в него закидываются все reducer. Далее сам стор прокидывается в приложение через провайдер. Использование – через useAppSelector получаем стор либо его часть, через useAppDispatch изменяем его. Actions – это обычный обьект , в котором есть поле type и по желанию поле payload. Хорошая практика не писать вручную экшны – а создовать экшн-криэйторы которые будут возвращать обьект экшна. Редьюсеры всегда возвращают новый стейт! В TS пользоваться кастомными хуками для учета типизации

2. Redux Toolkit

Особенности:

- под капотом библиотека Reselect

- уже подключены/настроены devtools

- в редюсерах не нужно каждый раз возвращать новое состояние, можно изменять текущее

- для работы с санками предоставляет метод createAsyncThunk, а для обработки и записи результата уже используем extraRedusers где у нашего fetchUsers ранее созданного через createAsyncThunk уже будут 3 состояния: fullfilled pending reject

Редюсеры создаются через createSlice

Использование – подключается и используется так же как и п1, за исключением редюсера. Он создается через createSlice которые принимает имя, инишлстейт, редюсер – и из фаила возвращаются сгенерировавшиеся экшны и редюсер, который затем прокидывается в стор

import { createSlice } from '@reduxjs/toolkit';

import { fetchUsers } from '../mWare/usersMWare';

const initialState: any = {

  isShowed: true,

};

// eslint-disable-next-line @typescript-eslint/default-param-last

export const showSlice = createSlice({

  name: 'showLogo',

  initialState,

  reducers: {

    showLogo: (state, action) => {

      console.log(action);

      state.isShowed = action.payload.isShowed;

    },

  },

});

export const { showLogo } = showSlice.actions;

export default showSlice.reducer;

Как создать санку.

Сначала через createAsyncThunc создаем саму санку

export const fetchUsers = createAsyncThunk('users/fetchUsers', async (payload: any) => {

  const response = await axios('https://jsonplaceholder.typicode.com/users');

  return response.data;

});

А затем в редюсер(сразу же под reducers) добавляем блок extraReducers который будет отвечать за этапы запроса как лоадинг, ошибка, успех

import { createSlice } from '@reduxjs/toolkit';

import { fetchUsers } from '../mWare/usersMWare';

const initialState: any = {

  isShowed: true,

};

// eslint-disable-next-line @typescript-eslint/default-param-last

export const showSlice = createSlice({

  name: 'showLogo',

  initialState,

  reducers: {

    showLogo: (state, action) => {

      state.isShowed = action.payload.isShowed;

    },

  },

  extraReducers: builder => {

    builder

      .addCase(fetchUsers.pending, state => {

        state.loading = true;

        state.error = null;

      })

      .addCase(fetchUsers.fulfilled, (state, action) => {

        state.loading = false;

        state.error = null;

        state.users = action.payload;

      })

      .addCase(fetchUsers.rejected, (state, action) => {

        state.loading = false;

        state.error = action.error;

      });

  },

});

export const { showLogo } = showSlice.actions;

export default showSlice.reducer;

В самой компоненте используется санка как

            dispatch(fetchUsers({ payload: undefined }));

3. Redux Toolkit Query

Особенности:

- добавлено кэширование

- не нужно иметь то состояние которое обычно представляется в редаксе

- добавлен шортпуллинг

Библиотека предоставлет методы createApi и fetchBaseQuery. Новое апи создаем через createApi, базовый запрос указываем через fetchBaseQuery , и дальше указываем эндоипты, где гет запросы - это query, а пост - это mutation. Дальше добавляем наши апи в rootReduser i setupStore . И библиотека генерирует хуки, через которые мы можем забирать уже и данные, и функцию для вызова у пост запроса, и лоадинги ерроры и т д.

Один и тот же запрос можно вызывать по несколько раз, и если параметры будут те же, библиотека фактически не будет отправлять запрос, а возьмет данные их кэша.

Если же мы отправили пост запрос - на бэке меняются данные и мы хотим что б подтянулись эти данные в RTK состояние - необходимо при создании запроса (в createApi ) указывать invalidatedTages

Использование

Начальная части подключения в п1 выше, понятно что без редюсеров.

Создаем сервис userApi в папке services

import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';

export const userAPI = createApi({

  reducerPath: 'userAPI',

  baseQuery: fetchBaseQuery({ baseUrl: 'https://jsonplaceholder.typicode.com' }),

  endpoints: build => ({

    fetchUsers: build.query<[], void>({

      query: () => '/users',

    }),

getFormFields: builder.mutation<any, { offerCreditCardId: string; blockId: 1 | 2 }>({ // пример POST-запроса

      query: (body) => ({

        url: 'order/creditcard/limit/fields',

        method: 'POST',

        body,

      }),

    }),

  }),

});

export const { useFetchUsersQuery } = userAPI; // хуки самогенерируются

Далее нужно прописами редюсер и миддлвару в стор

const rootReducer = combineReducers({

  showLogo: showReducer,

  [userAPI.reducerPath]: userAPI.reducer, // эта строка

});

export const store = configureStore({

  reducer: rootReducer,

  middleware: getDefaultMiddleware => getDefaultMiddleware().concat(userAPI.middleware), // и эта – мидлвары добавляем к дефолтной через конкат и т.д. цепочкой

});

Теперь хук сгенерировался и его можно использовать в компонентах

  const { data, isLoading, error } = useFetchUsersQuery();

Отличие пост от гет хуков – что пост хуки вернут функцию, которую можно вызвать в какой то момент например в юзэффекте для осуществления запроса

  const [offerCreditCard, { isError }] = useOfferCreditCardMutation(); // пример