nexign

Преимущества RTK Query и поучительный случай на enterprise проекте

Марина Агафонова

Middle front-end (ReactJS) разработчик компании Nexign

Обо мне

Марина Агафонова

Работала:

- back-end (.NET)
- full-stack (.NET, Django, ReactJS, Angular)
- front-end (ReactJS)

Выпускница магистратуры ИТМО по направлению «Веб-технологии»

nexign

Что такое RTK Query

- какие проблемы решает
- особенности API
- примеры базового использования

Какие проблемы решает

- Отслеживание состояния загрузки (если визуально нужно отображать спиннер)

  `isLoading, isFetching`

- Избежание повторных запросов одних и тех же данных

- Управление временем хранения кэша при взаимодействии пользователя с интерфейсом

Особенности API RTK Query

- логика работы с данными основана на createSlice и createAsyncThunk в Redux Toolkit
- эндпоинты API определяются заранее
- способность генерации хуков React
- предоставление опции “cache entry lifecycle”
- полностью написан на TypeScript

Использование RTK Query

Создание API slice

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react'
import type { Pokemon } from './types'

// Define a service using a base URL and expected endpoints
export const pokemonApi = createApi({
  reducerPath: 'pokemonApi',
  baseQuery: fetchBaseQuery({ baseUrl: 'https://pokeapi.co/api/v2/' }),
  endpoints: (builder) => ({
    getPokemonByName: builder.query<Pokemon, string>({
      query: (name) => `pokemon/${name}`,
    }),
  }),
})

// Export hooks for usage in functional components, which are
// auto-generated based on the defined endpoints
export const { useGetPokemonByNameQuery } = pokemonApi
```


nexign

Использование RTK Query

Настройка хранилища (store)

```
import { configureStore } from '@reduxjs/toolkit'
// Or from '@reduxjs/toolkit/query/react'
import { setupListeners } from '@reduxjs/toolkit/query'
import { pokemonApi } from './services/pokemon'

export const store = configureStore({
  reducer: {
    // Add the generated reducer as a specific top-level slice
    [pokemonApi.reducerPath]: pokemonApi.reducer,
  },
  // Adding the api middleware enables caching, invalidation, polling,
  // and other useful features of `rtk-query`.
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(pokemonApi.middleware),
})

// optional, but required for refetchOnFocus/refetchOnReconnect behaviors
// see `setupListeners` docs - takes an optional callback as the 2nd arg for
customization
setupListeners(store.dispatch)
```

Использование RTK Query

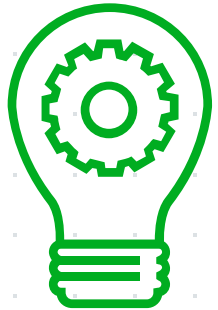
Вызов useQuery хука в ReactJS компоненте

```
import * as React from 'react'
import { useGetPokemonByNameQuery } from './services/pokemon'

export default function App() {
  // Using a query hook automatically fetches data and returns query values
  const { data, error, isLoading } = useGetPokemonByNameQuery('bulbasaur')
  // Individual hooks are also accessible under the generated endpoints:
  // const { data, error, isLoading } =
pokemonApi.endpoints.getPokemonByName.useQuery('bulbasaur')

  // render UI based on data and loading state
}
```


nexign

Хуки RTK Query

- useQuery
- useMutation
- useQueryState
- useQuerySubscription
- useLazyQuery
- useLazyQuerySubscription
- usePrefetch



nexign

Automated Re-fetching

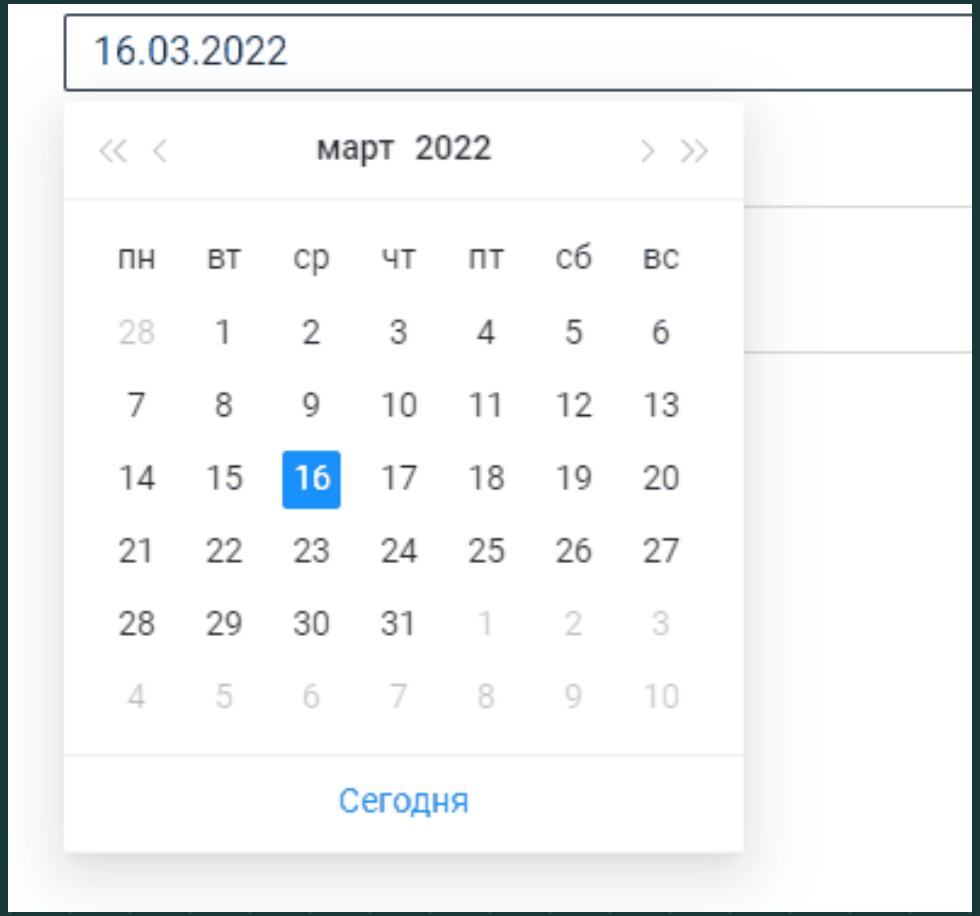
- tagTypes
- invalidatesTags
- providesTags

```
1  import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react'
2  import type { Post, User } from './types'
3
4  export const api = createApi({
5    baseQuery: fetchBaseQuery({ baseUrl: '/' }),
6    tagTypes: ['Posts'],
7    endpoints: (build) => ({
8      getPosts: build.query<Post[], void>({
9        query: () => 'posts',
10       providesTags: (result) =>
11         result ? result.map(({ id }) => ({ type: 'Posts', id })) : ['Posts'],
12     }),
13     addPost: build.mutation<Post, Partial<Post>>({
14       query: (body) => ({
15         url: `post`,
16         method: 'POST',
17         body,
18       }),
19       invalidatesTags: ['Posts'],
20     }),
21     getPost: build.query<Post, number>({
22       query: (id) => `post/${id}`,
23       providesTags: (result, error, id) => [{ type: 'Posts', id }],
24     }),
25   }),
26 })
27
28 export const { useGetPostsQuery, useGetPostQuery, useAddPostMutation } = api
```


nexign

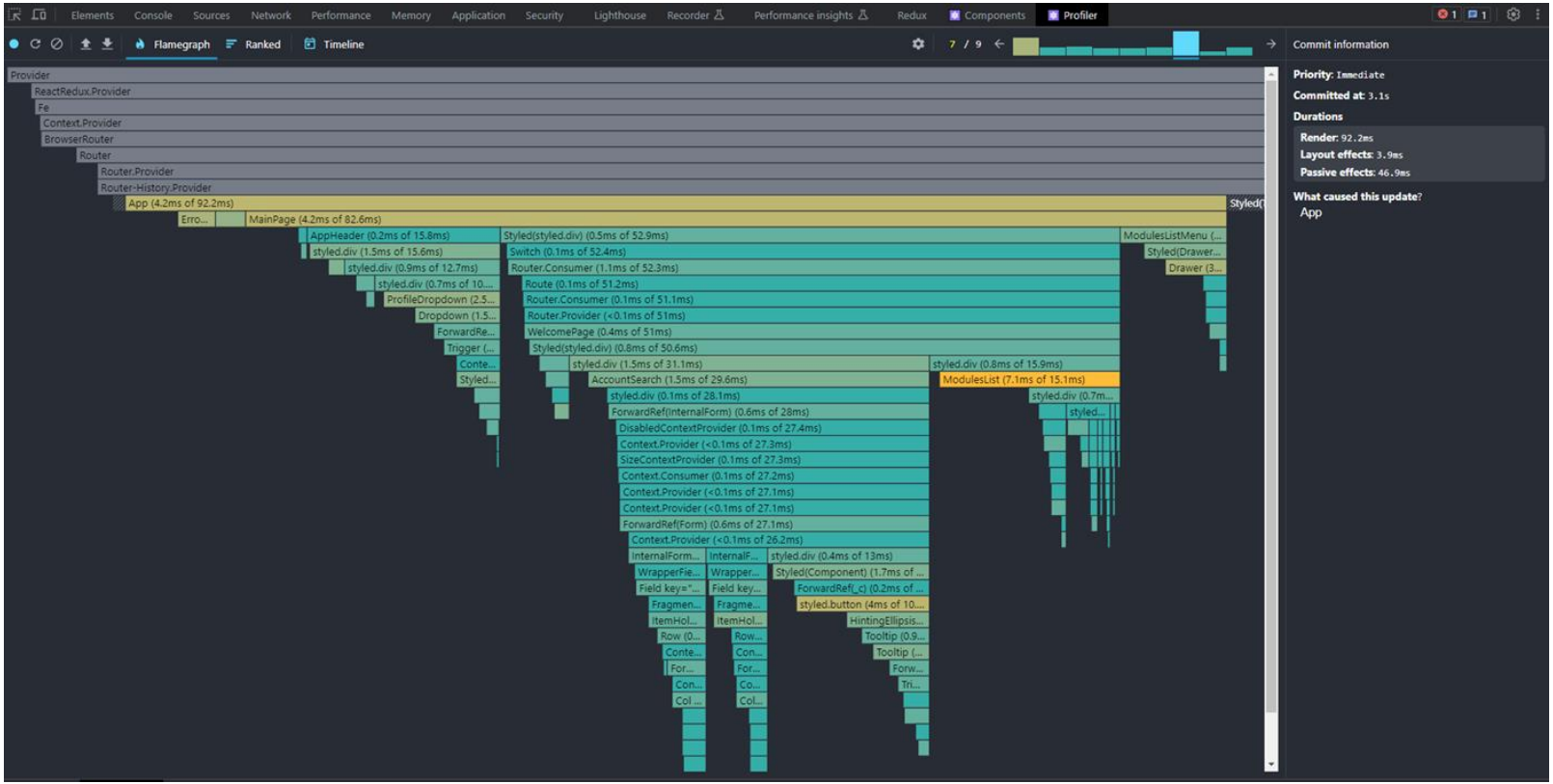
Фантастические баги и где они обитают

Случай на enterprise проекте: как искали причину бага и как фиксили


16.03.2022
март 2022
пн вт ср чт пт сб вс
28 1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31 1 2 3
4 5 6 7 8 9 10
Сегодня

nexign

React Developer Tools: Profiler



nexign

Причина бага

```
  someEndpoint: builder.query<string[], string>({
    query: (someParam: string) => `someUrl/${someParam}`,
  }),
```

```
const { result } = api.useSomeEndpointQuery('someParam', {
  selectFromResult: state => ({
    result: state.data || [],
  }),
});
```

Checklist

- Обращать внимание на то, как написан selectFromResult в options хуков
- Не задавать явных значений в selectFromResult (пустые массивы, объекты). Лучше сделать дефолтное значение через константу/обернуть логику в useMemo
- Проверять рендеринг через React DevTools или другой удобный инструмент

Q&A