hillel IT school

Lesson 37

План заняття

- Redux toolkit
- axios

Redux toolkit

У Redux є певі недоліки:

- Дуже багато коду для простих дій: створення об’єкту action, створення функції actionCreator, підключення redux store до компонентів (класвоих).
- Імутабельний state у reducer - слабочитаємий код.

Переваги redux-toolkit:

- Actions, actionCreator - створюються автоматично.
- Mutable state – immerjs (https://immerjs.github.io/immer/).
- Зручна обробка асинхронних операцій.
- Готовий функціонал для операцій з даними.

Redux toolkit

Redux Toolkit (RTK) — це офіційний рекомендований підхід для написання логіки Redux. Пакет @reduxjs/toolkit охоплює основний пакет redux і містить методи API та загальні залежності, які, необхідні для створення програми Redux. Redux Toolkit використовує запропоновані передові практики, спрощує більшість завдань Redux, запобігає поширеним помилкам і полегшує написання програм Redux.

Якщо ви сьогодні пишете будь-яку логіку Redux, вам слід використовувати Redux Toolkit для написання коду!

RTK містить утиліти, які допомагають спростити багато поширених випадків використання, включаючи налаштування store, створення reducers і написання незмінної логіки оновлення, і навіть створення slice стану одночасно.

Документація https://redux-toolkit.js.org/introduction/getting-started

Redux toolkit

Для завантаження RTK використовуємо

npm install @reduxjs/toolkit react-redux

Нові методи:

- configureStore() - обгортає createStore, щоб забезпечити спрощені параметри конфігурації. Він може автоматично поєднувати ваші reducers, додаває будь-яке проміжне програмне забезпечення Redux, яке ви надаєте, включає redux-thunk за замовчуванням і дозволяє використовувати розширення Redux DevTools.
- createReducer() - дозволяє надавати таблицю пошуку типів action вам більше не порібно писати switch. Крім того, він автоматично використовує бібліотеку immer (https://github.com/immerjs/immer), що дозволити вам писати простіші функції.

Redux toolkit

- createAction() - генерує функцію створення action.
- createSlice() - приймає об’єкт функцій reducer, назву slice та значення початкового state та автоматично генерує reducer фрагмента з відповідними actionCreator і action types.
- combineSlices() - об’єднує кілька фрагментів в один reducer і дозволяє lazy load фрагментів після ініціалізації.
- createAsyncThunk - приймає рядок action type та функцію, яка повертає promise, і генерує канал, який відправляє типи action, що pandig/resolve/reject на основі promise.
- createEntityAdapter - генерує набір багаторазових reducer і selector для керування нормалізованими даними в сховищі.
- Утиліта createSelector із бібліотеки Reselect.

configureStore()

configureStore спрощує процес створення store. Його параметри:

- reducer - це одна функція редуктора, яка використовуватиметься як кореневий reducer.
- middleware - необов’язковий параметр який буде прийми усі middleware таа конфігорувати їх.
- devTools - необов’язковий параметр який підключає devTools.
- preloadedState - необов’язковий параметр який може задати стан ще до підключення reducers.
- enhancers - додатковий масив підсилювачів redux store або функція callback для налаштування масиву розширювачів.

configureSlice()

Функція, яка приймає початковий стан, об’єкт функцій reducer та name, а також автоматично генерує actionCreator і action type, які відповідають reducer і state.

Цей API є стандартним підходом для написання логіки Redux.

Внутрішньо він використовує createAction і createReducer, тому ви також можете використовувати Immer для написання «мутуючих» незмінних оновлень.

Параметри:

- initialState - Початкове значення state для цього фрагмента store. Це також може бути функція "лінивого ініціалізатора", в першу чергу корисно для таких випадків, як читання початкового стану з localStorage.
- name - Назва рядка для цього фрагмента state. Згенеровані константи типу action використовуватимуть це як префікс.

configureSlice

- reducers - Об’єкт, що містить функції Redux (функції, призначені для обробки певного типу action, еквівалентні одному оператору case в reducer). Ключі в об’єкті використовуватимуться для генерації констант action type. Крім того, якщо будь-яка інша частина програми надсилає action з тим самим рядком типу, буде запущено відповідний reducer. Тому ви повинні давати функціям описові назви. Цей об’єкт буде передано createReducer, тому reducers можуть безпечно "мутувати" наданий їм стан.
- extraReducers - дозволяє createSlice відповідати та оновлювати власний стан у відповідь на інші типи дій, крім типів, які він згенерував. Часто використовується з createAsyncThunk.
- selectors - Набір селекторів, які отримують стан фрагмента як свій перший параметр та будь-які інші параметри. Кожен селектор матиме відповідний ключ в отриманому об’єкті селекторів.
- reducerPath - Вказує на те, де має розташовуватися фрагмент. За замовчуванням ім'я. Це використовується combineSlices і створеними за замовчуванням slice.selectors.

configureSlice

createSlice поверне об'єкт з властивостями:

- name - строка name.
- reducer - функція reducer.
- actions - об’єкти action, які генеруються за допомогою actionCreator.
- getInitialState - функція яка повертає initialState,
- reducerPath - вказує на шлях до reducer,
- selectSlice - повертає selector для slice;
- selectors - об’єкт selectors,

createAsyncThunk

Функція, яка приймає рядок типу дії Redux, і функцію зворотного виклику, яка має повертати promise. Він генерує action type життєвого циклу promise на основі префікса типу action, який ви передаєте, і повертає actionCreator thunk, який запускатиме зворотний виклик promise і відправлятиме action життєвого циклу на основі повернутого promise.

Це абстрагує стандартний рекомендований підхід для обробки життєвих циклів асинхронних запитів.

Він не генерує жодних функцій reducer, оскільки не знає, які дані ви отримуєте, як ви хочете відстежувати стан завантаження або як потрібно обробляти action, які ви повертаєте. Ви повинні написати власну логіку reducer, яка обробляє ці action, з будь-яким станом завантаження та логікою обробки, яка підходить для вашої програми.

createAsyncThunk

Парамери:

- type - Рядок, який використовуватиметься для створення додаткових констант типу action Redux, що представляє життєвий цикл асинхронного запиту.

Наприклад для типу 'users/upload' Redux згенерує

- pending: 'users/upload/pending'
- fulfilled: 'users/upload/fulfilled'
- rejected: 'users/upload/rejected'

- payloadCreator - Функція зворотного виклику, яка має повернути promise, що містить результат деякої асинхронної логіки. Він також може повертати значення синхронно. Якщо є помилка, вона має або повернути reject, що містить екземпляр помилки, або звичайне значення, наприклад описове повідомлення про помилку.

createAsyncThunk

Функція payloadCreator може містити будь-яку логіку, необхідну для обчислення відповідного результату. Це може включати стандартний запит на вибірку даних AJAX, кілька викликів AJAX із результатами, об’єднаними в остаточне значення, тощо.

Функція payloadCreator буде викликана з двома аргументами:

- arg: єдине значення, що містить перший параметр, який було передано actionnCreator thunk під час його відправки. Це корисно для передачі таких значень, як елементи ідентифікатора, які можуть знадобитися як частина запиту. Якщо вам потрібно передати кілька значень, передайте їх разом в об’єкти, наприклад dispatch(fetchUsers({id: 1, sortBy: 'name'})).
- thunkAPI: об’єкт, що містить усі параметри, які зазвичай передаються до функції Redux, а також додаткові параметри.

createAsyncThunk

ThunkAPI params:

- dispatch - метод відправлення Redux store
- getState - метод оримання state
- extra - «додатковий аргумент», наданий проміжному програмному забезпеченню Thunk під час налаштування, якщо доступний
- requestId - унікальне значення ідентифікатора рядка, яке було автоматично згенеровано для ідентифікації цієї послідовності запитів
- signal: об’єкт AbortController.signal, який можна використовувати, щоб побачити, чи інша частина логіки програми позначила цей запит як такий, що потребує скасування.

createAsyncThunk

ThunkAPI params:

- rejectWithValue(value, [meta]) - це допоміжна функція, яку ви можете повернути (або додати) у свій actionCreator, щоб повернути відхилену відповідь із визначеним payload і meta. Він передасть будь-яке значення, яке ви йому надасте, і поверне його в payload reject action. Якщо ви також передасте meta, його буде об’єднано з існуючим rejectedAction.meta.
- fulfillWithValue(value, meta) - це допоміжна функція, яку ви можете повернути у своєму творці дій для виконання зі значенням, маючи можливість додавати до completedAction.meta.

createAsyncThunk

createAsyncThunk повертає стандартний actionCreator преобразователя Redux. Функція actionCreatorThunk матиме actionCreator для незавершених, виконаних і відхилених випадків, доданих як вкладені поля.

Наприклад fetchUsersThunk поверне:

- fetchUsersThunk .pending, actionCreator, який надсилає action 'users/fetchUsersThunk /pending'
- fetchUsersThunk .fulfilled, actionCreator, який надсилає action 'users/fetchUsersThunk /fulfilled'
- fetchUsersThunk .rejected, actionCreator, який надсилає action 'users/fetchUsersThunk /rejected'

createAsyncThunk

Після відправлення thunk:

- dispatch panding action,
- викличте зворотній виклик payloadCreator і зачекає, поки повернутий promise завершиться
- якщо promise виконається successfully => dispatch виконану action зі значенням promise як action.payload
- якщо promise вирішена зі значенням, що повертає rejectWithValue(value) => dispatch reject зі значенням, переданим у action.payload, і rejected як action.error.message
- якщо promise faled та не був оброблений за допомогою rejectWithValue => dispatch reject з серіалізованою версією значення помилки як action.error
- Поверне виконаний promise, що містить остаточну надісланий action (або виконаний, або відхилений об’єкт дії)

axios

Axios (https://axios-http.com/ru/docs/intro) - це HTTP-клієнт, заснований на Promise для node.js і браузера. Він ізоморфний (він може працювати в браузері та node.js з тим же базовим кодом). На стороні сервера він використовує нативний http-модуль node.js, тоді як на стороні клієнта (браузера) він використовує XMLHttpRequests.

Особливості:

- Робить XMLHttpRequests запити з браузера
- Робить http запити з node.js
- Підтримує Promise API
- Перехоплює запити і відповіді
- Преобразовує дані запиту та відповіді
- Отменяет запросы
- Автоматичне перетворення для JSON-данних
- Підтримка на стороні клієнта для захисту від XSRF

axios

Інсталяція

npm install axios

Axios instance

Ви можете створити новий екземпляр axios з конфігурацією користувача. Для цього вам порыбно викликати метод create()

Наприклад:

```
axios.create({
  baseURL: "https://reqres.in/api/",
  signal: signal,
  headers: {
    "Content-Type": "application/json",
  },
});
```

create()

Параметри:

- baseURL (строка) - Базовий URL для всіх запитів.
- headers (об'єкт) - Об'єкт заголовків HTTP.
- timeout (число) - Максимальний час очікування для запиту в мілісекундах.
- signal (об'єкт AbortController) - може використовуватися для відміни запиту.

Axios methods

- axios#request(config)
- axios#get(url[, config])
- axios#delete(url[, config])
- axios#head(url[, config])
- axios#options(url[, config])
- axios#post(url[, data[, config]])
- axios#put(url[, data[, config]])
- axios#patch(url[, data[, config]])
- axios#getUri([config])

Request config

Параметри:

- url - URL-адреса сервера, яка буде використовуватися для запиту
- method - це метод запиту, який слід використовувати при подачі запиту
- headers - заголовкі для запиту
- params - URL-параметри, які надсилаються разом із запитом. Повинні бути звичайним об'єктом або об'єктом URLSearchParams
- Data - дані, які посилаються як тіло запиту. Використовуються тільки з методами запиту 'PUT', 'POST', 'DELETE , і 'PATCH'
- Докладніше тут https://axios-http.com/ru/docs/req_config

Response sheme

Відповідь на запит містить наступну інформацію.

- data - це відповідь, надана сервером
- status - це код стану HTTP-запиту
- statusText - це повідомлення про стан HTTP-запиту
- headers - заголовки HTTP-запиту, на які відповів сервер
- config - це конфігурація, яка була надана `axios` для запиту
- request - це запит, який згенерував цю відповідь. // Це останній екземпляр ClientRequest у node.js (у перенаправленнях) та екземпляр XMLHttpRequest у браузері.

Дякую за увагу