45. Redux-middleware

Цель:

Познакомиться с понятием redux-middleware:

- архитектура приложения с асинхронными операциями
- что такое middleware

План занятия:

- Архитектура приложения с асинхронными операциями
- Что такое middleware
    - Redux-thunk
    - Redux-saga

Конспект:

Архитектура приложения с асинхронными операциями

По умолчанию, экшены в Redux являются синхронными, что, является проблемой для современных приложений, котором постоянно необходимо взаимодействовать с API, или выполнять другие асинхронные действия. Для решении этой проблемы у Redux есть middleware, которая стоит между диспатчом экшена и редюсером. Существует две самые популярные middleware библиотеки для асинхронных экшенов в Redux, это — Redux Thunk.

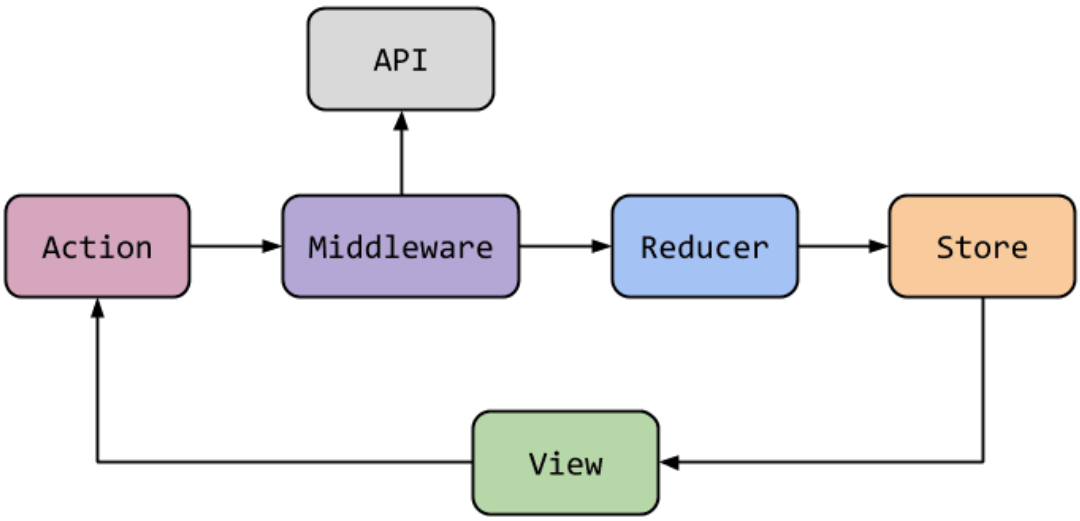
Middleware

Middleware помогают нам получить место для расширения кода между отправкой экшена и редьюсером. Т.е. мидлвары — функции, которые последовательно вызываются в процессе обновления данных в хранилище.

Redux-мидлвары используют для:

- логирования
- обработки ошибок
- общения с асинхронным AP и т.д.

Особенностью мидлвара является то, что они компонуемы. Можно объединить несколько мидлваров вместе, где каждый мидлвар будет независимым. Каждый мидлвар не будет знать и влиять на то, что происходит до или после него в цепочке.



applyMiddleware(...middlewares) — ..middlewares (*arguments*): Функции, которые соответствуют Redux *middleware API*. Каждый мидлвар получает dispatch и getState функции в качестве именованных аргументов и возвращает функцию. Эта функция будет передана next dispatch-методу мидлвара и, как ожидается, вернет функцию экшена, вызывающую next(action) с возможно другим аргументом или позже или, возможно, не вызывая его вообще. Последний мидлвар в цепочке получит реальный dispatch метод стора в качестве next параметра, таким образом завершая цепочку. Следовательно, сигнатурой мидлвара является ({ getState, dispatch }) => next => action.

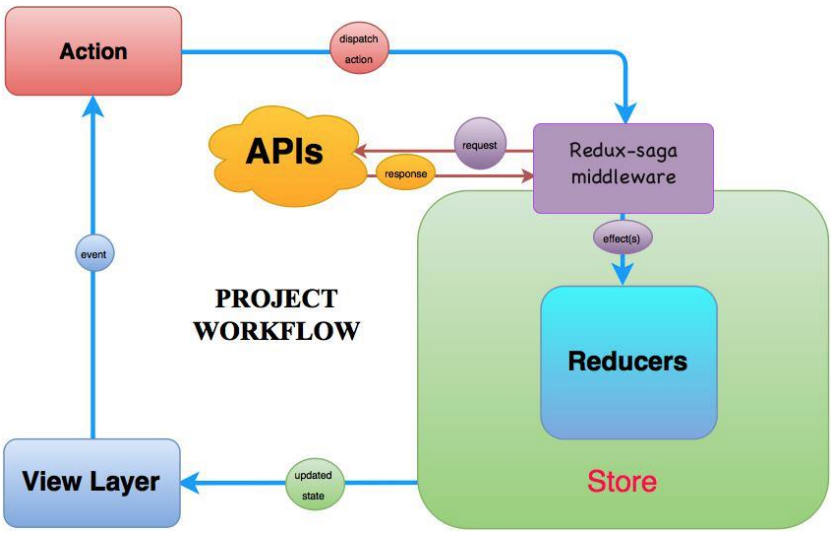
Преимуществом использования redux-thunk/redux-saga является то, что компонент не знает, что выполняется асинхронное действие.

Т.к. промежуточный слой автоматически передает функцию dispatch в функцию, которую возвращает генератор действий, то снаружи, для компонента, нет никакой разницы в вызове синхронных и асинхронных действий (и компонентам больше не нужно об этом беспокоиться). Тем самы еще проще разделять логику компонентов и бизнес логику.

Redux-thunk

Redux Thunk это middleware библиотека, которая позволяет вам вызвать action creator, возвращая при этом функцию вместо объекта. Функция принимает метод dispatch как аргумент, чтобы после того, как асинхронная операция завершится, использовать его для диспатчинга обычного синхронного экшена, внутри тела функции.

Redux-saga



Redux-saga — это библиотека, которая призвана упростить и улучшить побочные эффектов (т.е. таких действий как асинхронные операции, например, загрузки данных), облегчить тестировании и лучше справляться с ошибками.

Saga — это как отдельный поток в приложении, который отвечает за побочные эффекты. Redux-saga — это мидлвар redux, что означает, что этот поток может запускаться, останавливаться и отменяться из основного приложения с помощью обычных действий redux, оно имеет доступ к полному состоянию redux приложения и также может диспатчить действия redux.

Библиотека использует концепцию ES6, под названием генераторы, для того, чтобы сделать эти асинхронные потоки легкими для чтения, написания и тестирования.

Гереraторы

Генераторы (Generators) это функции которые могут быть остановлены и продолжены, вместо выполнения всех выражений в один проход.

Когда мы вызываем функцию-генератор, она возвращает объект-итератор. И с каждым вызовом метода итератора next() тело функции-генератора будет выполняться до следующего yield выражения и затем останавливаться.

Saga Helpers

takeEvery — позволяет одновременно запускать несколько тасков одновременно.

takeLatest — позволяет получать ответ только от последнего запроса (например, чтобы всегда показывать последнюю версию данных).

Если у нас есть несколько саг, то можно их объединить в общую сагу-наблюдатель **(watcher saga)**. Это позволяет добавить больше гибкости для реализации сложной логики (но это может быть лишним для простых приложений).

https://redux-saga.js.org/docs/api#take every pattern-saga-args

Call - выполняет функцию. Если он возвращает promise, то приостанавливает сагу до тех пор, пока promise не вызовет resolve.

Put — диспатчит экшн.

Select - запуск функции выбора для получения данных из state.