43. Redux

Цель:

Познакомиться со следующими понятиями redux:

- action type, action creator
- reducer
- store

План занятия:

- Что такое redux:
    - концепция
    - data flow
- actions
- redusers
- store

Конспект:

Redux:

Redux является предсказуемым контейнером состояния для JavaScript приложений.

Мотивация redux

- Вынужденность управлять большим количеством состояний: ответы сервера, локальные данные о состоянии приложения и др.
- Управление постоянно изменяющимися состояниями в какой-то момент приводило к непониманию, что происходит в приложении
- “**Вы больше не можете контролировать когда, почему и как состояние обновилось**”

Примеры данных в redux:

- ответы сервера
- данные, созданные локально, но еще не сохраненные на сервере
- UI-состояния
- выделенный таб
- показанное модальное окно
- нумерация страниц и т.д

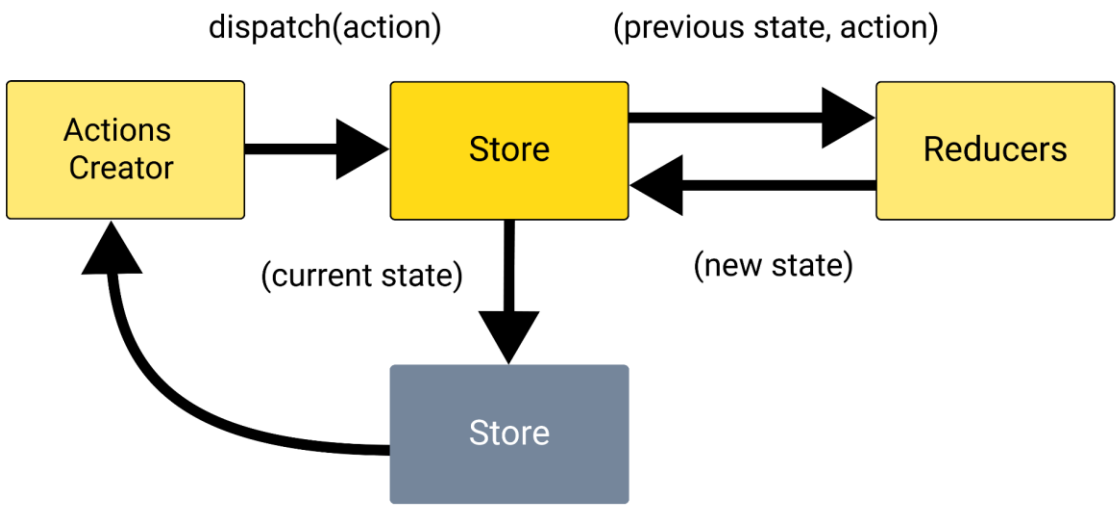
Три принципа redux

Redux может быть описан тремя фундаментальными принципами:

1) **Единственный источник правды.** Состояние всего вашего приложения сохранено в дереве объектов внутри одного стора.

2) **Состояние только для чтения.** Единственный способ изменить состояние — это применить экшен — объект, который описывает, что случится.
3) **Мутации написаны, как чистые функции.** Для определения того, как дерево состояния будет трансформировано экшенами, вы пишете чистые редюсеры.

Redux Data Flow



Actions && Action Creators

Экшены — это структуры, которые передают данные из вашего приложения в стор. Они являются *единственными* источниками информации для стора. Вы отправляете их в стор, используя метод store.dispatch() или хук useDispatch.

Экшены — это обычные JavaScript-объекты. Экшены должны иметь поле `type`, которое указывает на тип исполняемого экшена. Типы должны быть, как правило, заданы, как строковые константы. Обычно это делают для возможности вынесения констант в отдельный модуль и для удобного переиспользования в последующем.

Генераторы экшенов (Action Creators) — не что иное, как функции, которые создают экшены.

Reducers

Редюсеры определяют, как состояние приложения изменяется в ответ на экшены, отправленные в стор. Помните, что экшены только описывают, что произошло, но не описывают, как изменяется состояние приложения.

Редюсер (reducer) — это чистая функция, которая принимает предыдущее состояние и экшен (state и action) и возвращает следующее состояние (новую версию предыдущего).

(previousState, action) => newState
Список того, что **никогда** нельзя делать в редюсере:

- Непосредственно изменять то, что пришло в аргументах функции;
- Выполнять какие-либо сайд-эффекты: обращаться к API или осуществлять переход по роутам;
- Вызывать не чистые функции, например `Date.now()` или `Math.random()`.

Обращаем внимание на следующие версии при написании reducers:

- Не изменяем state (mutations). Необходимо вычислить новую версию состояния(state) и возвращать ее
- Необходимо возвращать предыдущую версию состояния в default ветку

Композиция редюсеров

```
import { combineReducers } from 'redux';

import todosReducer from './features/todos/todosSlice';
import filtersReducer from './features/filters/filtersSlice';

const rootReducer = combineReducers({
  todos: todosReducer,
  filters: filtersReducer,
});

export default rootReducer;
```

Каждый из этих дочерних редюсеров управляет только какой-то одной частью глобального состояния. Параметр `state` разный для каждого отдельного дочернего редюсера (visibilityFilter, todos) и соответствует той части глобального состояния, которой управляет этот дочерний редюсер.

Store

Стор - это не класс. Это просто объект с некоторыми методами в нем.

Стор содержит все дерево состояний вашего приложения. Единственный способ поменять состояние внутри него - это направить(dispatch) action на него.

Хук useSelector

useSelector - это аналог mapStateToProps. Хук принимает на вход селектор - метод, который принимает redux state и возвращает из него необходимые данные.

Хук useStore

useStore - получение store целиком.

const { dispatch, getState, subscribe } = useStore();