**Redux Essentials, Part 1: Redux Overview and Concepts**

**Redux is a pattern and library for managing and updating application state, using events called "actions".**

### Why Should I Use Redux?

Предсказуемость, возможность путешествия во времени.

### When Should I Use Redux?

Когда приложение большое и сложное, когда инфа нужна компонентам на разных уровнях. Т.е. не всегда.

### Redux Libraries and Tools

#### React-Redux

Redux не привязан к конкуретному фреймворку, но вообще создавался под react. Этот пакет нужен для упрощенной работы в реакт.

#### Redux Toolkit

Боготворят эту штуку созджатели редакса. Потому что она якобы решает проблему бойлерплейта. Многие действия, которые нужно было писать в редакс вручную, вшиты в редакс тулкит. Так же туда вшит immer(но не во всяком месте можно мутации делать) и rtk query для упрощенной работы с данными с сервака.

#### Redux DevTools Extension

Для time-travel debugging, ну и просто следить за состоянием.

## Redux Terms and Concepts

### State Management

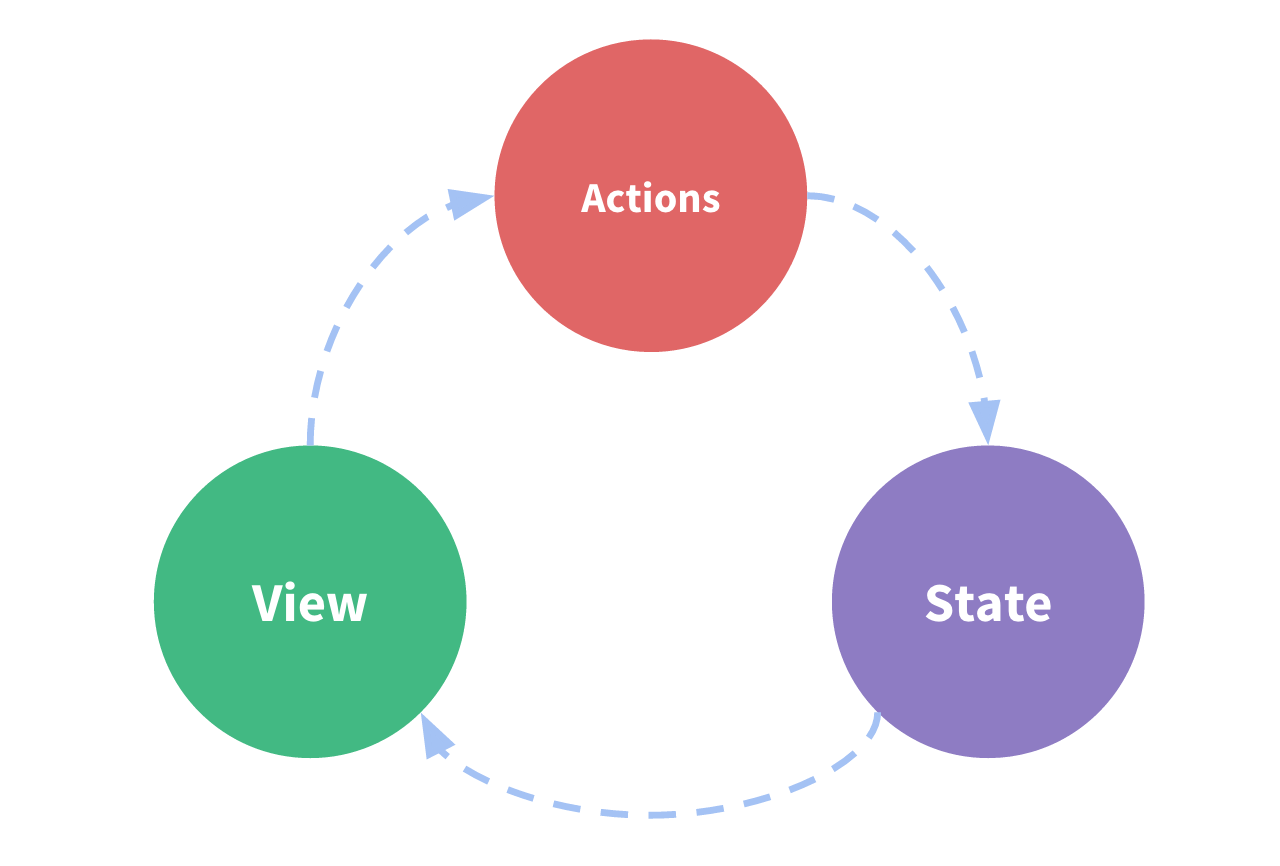
Вот зачем в терминах начинать с примера? Зачем?!

Дали пример чистого реакта – приложение счетчика.

* The **state**, the source of truth that drives our app;
* The **view**, a declarative description of the UI based on the current state
* The **actions**, the events that occur in the app based on user input, and trigger updates in the state

This is a small example of **"one-way data flow"**:

* State describes the condition of the app at a specific point in time
* The UI is rendered based on that state
* When something happens (such as a user clicking a button), the state is updated based on what occurred
* The UI re-renders based on the new state



Все ок, когда компонентам на разных уровнях не требуется одна и та же инфа. Когда же нужна, начинаются проблемы. Надо поднимать состояние и прокидывать через все дерево через пропсы. Когда приложение маленькое – ок. Когда большое – это уже пропс дриллинг. Вариант решения – вынести всю инфу в хранилище, отдельное от представления, и брать всю нужную инфу для отрисовки оттуда.

Но определение термина они так и не дали. Эта документация мне не нравится.

### Immutability

Мутабельный – значит изменяемый. В js и объекты и массивы (хаха, как будто мы не знаем, что в js – все объекты, даже примитивы и то из объекта, просто особенного, исчезающего так скажем). Иммутабельность – неизменяемость. А как быть, что делать с хранилищем? Инфа же должна меняться. Все просто – создаешь копию и меняешь копию. Есть куча методов, которые не мутируют.

### Terminology

#### Actions

Action – обычный объект с полем type. И еще может быть payload (и еще что-то вроде как по конвенции разрешали) для любой полезной инфы. Значения поля type – строка. Обычно так называется ‘где/что’.

const addTodoAction = {

type: 'todos/todoAdded',

payload: 'Buy milk'

}

Action это то, что произошло. Тут почему-то не указано, но по сути создатели просят воспринимать это как событие. (вроде)

#### Action Creators

Это функция, которая возвращает action. Зачем оно надо? Ну типо чище код. На деле в редакс так многочистильщиков, что аж грязно.

const addTodo = text => {

return {

type: 'todos/todoAdded',

payload: text

}

}

А ну и еще. Тут type – строка. По-хорошему дб переменная, чтоб, если мы где-то не так написали, на нас ругнулся вскод. Вот так чистота. Но в ртк это автоматическая штука, как и названия эти для type.

#### Reducers

Это тот, кто решает, что нужно делать. Типо обработчик события. Это функция, которая принимает текущее состояние и action (типо что произошло) и в зависимости от типа action возвращает новое состояние.

Правила:

* 1. Чистая функция, никаких сайд эффектов! Одни и те же входные данные, один и тот же результат на выходе.
  2. Не мутируют стейт, а создают копию и работают с копией.

Алгоритм действий редьюсера

1. Что произошло? (ЧИТАЕТ ACTION)
2. Меня волнует это? (проверяет, какие у него есть интересующие кейсы)
3. Волнует – делаем что-то, возвращаем новое состояние. Не волнует – возвращаем прежнее состояние.

const initialState = { value: 0 }

function counterReducer(state = initialState, action) {

// Check to see if the reducer cares about this action

if (action.type === 'counter/increment') {

// If so, make a copy of `state`

return {

...state,

// and update the copy with the new value

value: state.value + 1

}

}

// otherwise return the existing state unchanged

return state

}

Reducers can use any kind of logic inside to decide what the new state should be: if/else, switch, loops, and so on.

#### Store

Поч говорим о нем только сейчас, если это блин основная идея редакса? Крч, это хранилище, склад, холодильник. Объект, в котором лежит вся инфа, нужная в разных частях приложения. Но там лежит не все. Не все должно лежать в холодильнике. Если инфа нужна чисто одному компоненту и нигде больше, зачем ее переносить в отдельное хранилище? Она остается в локальном стейте.

А щас они зачем-то дали пример конфигурации из ртк. Коонч

import { configureStore } from '@reduxjs/toolkit'

const store = configureStore({ reducer: counterReducer })

console.log(store.getState())

// {value: 0}

#### Dispatch

Функция, которая несет action в reducer. Т.е. в параметрах у нее аргумент. Dispatch, как голубь, несущий письмо, чуваку, который должен принять какое-то решение и как-то поменять store. Dispatch – это автобус. Лимузин, самолет. Крч транспорт.

store.dispatch({ type: 'counter/increment' })

console.log(store.getState())

// {value: 1}

Придумали зашибенно вообще. А зачем мы про action creators говорили?

Крч если есть action creator, то все проще.

store.dispatch(increment())

А еще будет штука из react-redux, которая позволит упустить слово store. И будет просто dispatch(increment())

Просят воспринимать dispatch как триггер ивента. Ага, щас. Это голубь. Да и вообще, где во всей этой песне клик юзера на кнгпку?

О, показале криэйтера

const increment = () => {

return {

type: 'counter/increment'

}

}

store.dispatch(increment())

console.log(store.getState())

// {value: 2}

### Redux Application Data Flow

Своими словами:

UI показывает, например, скок осталось платьев в магазине. Пусть будет 10. Эту инфу он берет из стора и во весь экран выводит – 10 штук!!! Успейте купить!!!. В сторе написано: платьев 10 штук. Юзер захотел купить себе одно. Все, он уже нажимает оплатить. Что он сделал? – на этот вопрос отвечает action. Action Creator мутит письмо – action, где написано ЧЕЛИК КУПИЛ ПЛАТЬЕ! Если конкретнее в type будет типо «покупка». А в payload – скок платьев он купил. Мб два. Пусть будет два. Голубь, он же dispatch, на всех парах, махая крыльями, как ненормальный, мчит к reducer. В клюве у голубя вот этьо письмо. Reducer – управляющий магазина (store). Reducer такой, че принес? Читает письмо. Так, ну покупка нас волнует да. Ща, я посмотрю, что в таком случае надо делать. Угу, ну отправим ему на его пункт выдачи эти 2 платья. Ну и в сторе пометку сделаем, что платьев осталось 8. А то какие-нибудь шопоголики решат купить 10, потому что мы число не обновили, а UI дурачок позволит им это заказать. Все, reducer скопировал стор (чтоб помнить, че там было, а то потом ошибки будут, как мы поймем, что где было. Да и мутация стора UI вщ не тригеррит,в упор не видит, что что-то поменялось). Копию стора поменяли. Написали, что платьев теперь 8. UI пришло оповещение – ага, кол-во платьев изменилось(сам-то он глупенький, 10-2 сделать не может, да и не надо ему выполнять работу за всех), UI меняет свое сообщение на экране. Пишет 8 платьев осталось!! Успейте купить! Ну вот и все, круг замкнулся. Но важно заметить, что походу не одному компоненту потребовалась инфа о количестве платьев, раз это локально решить нельзя было, что в общий стор засунули.

А при первой отрисовке что?

Initial setup:

* A Redux store is created using a root reducer function
* The store calls the root reducer once, and saves the return value as its initial state
* When the UI is first rendered, UI components access the current state of the Redux store, and use that data to decide what to render. They also subscribe to any future store updates so they can know if the state has changed.

Есть начальный стейт и по нему отрисовывается все, что есть. Reducer главный такой просто: так, запоминай UI, щас стор выглядит так.

Summary

* **Redux is a library for managing global application state**
  + Redux is typically used with the React-Redux library for integrating Redux and React together
  + Redux Toolkit is the recommended way to write Redux logic
* **Redux uses a "one-way data flow" app structure**
  + State describes the condition of the app at a point in time, and UI renders based on that state
  + When something happens in the app:
    - The UI dispatches an action
    - The store runs the reducers, and the state is updated based on what occurred
    - The store notifies the UI that the state has changed
  + The UI re-renders based on the new state
* **Redux uses several types of code**
  + *Actions* are plain objects with a type field, and describe "what happened" in the app
  + *Reducers* are functions that calculate a new state value based on previous state + an action
  + A Redux *store* runs the root reducer whenever an action is *dispatched*