## Reducer, combineReducers(), Pure and Impure Functions

### Reducer

#### Reducers определяют, как должно измениться состояние

Можно создавать и отправлять actions и обрабатывать ошибки, но они никак не влияют на состояние. Чтобы обрабатывать входящие actions, необходимо настроить redusers.

Actions — это просто способы сообщить, что произошло некое событие, и привести какую-то информацию о том, что произошло, не более того. Задача redusers — указать, как изменится состояние хранилища в ответ на эти действия.

Reducers это простые функции, которые имеют одну цель. Reducers — это чистые функции, которые принимают предыдущие состояние и action в качестве аргументов и возвращают следующее состояние.

Согласно документации Redux reducers называются, потому что сигнатура их метода выглядит как данные, передаваемые в Array.prototype.reduce

Reducers должны быть чистыми функциями, а это означает, что с учетом ввода они будут каждый раз выдавать один и тот же соответствующий вывод. Это контрастирует с actions или middleware, где получаются побочные эффекты и часто возникают вызовы API. Выполнение чего- либо асинхронного или нечистого (например, вызов Date.now или Math.random()) в redusers — это антишаблон, который может ухудшить производительность или надежность приложения.

Документация Redux содержат такой пункт: «Получив те же аргументы, он должен вычислить следующее состояние и вернуть его. Без сюрпри- зов. Никаких побочных эффектов. Без вызовов API. Никаких изменений. Просто расчет». [redux.js.org/basics/reducers]redux.js.org/basics/reducers.

#### Начальное состояние

Redusers работают над изменением единственного в Redux хранилища. Проектирование начального состояния любого приложения будет влиять на то, как работает пользовательский интерфейс приложения (и в то же время оно подвержено влиянию этой работы), но, как правило, рекомендуется хранить сырые данные по возможности отделенными от данных пользовательского интерфейса. Один из способов сделать это — хранить значения, подобные идентификаторам в initialState, отдельно от их данных и использовать идентификаторы для поиска данных.

const initialState = {  
 authenticated: false,  
 profilePicture: null,  
 id: null,  
 name: null,  
 token: null,  
}

#### Настройка reducers для реагирования на входящие действия

При настройке начального состояния нужно создать несколько redusers для обработки входящих actions, чтобы обновлять хранилище. Reducers обычно используют инструкцию switch, чтобы обновить состояние в соответствии с типом входящих действий. Они возвращают новую копию состояния (не ту же самую версию с изменениями), которая затем будет применяться для обновления хранилища. Reducers также действуют по принципу «поймать все», чтобы гарантировать, что неизвестные действия просто вернут существующее состояние. Reducers выполняют вычисления и должны возвращать один и тот же результат каждый раз на основе заданного ввода — никаких побочных эффектов или неясных процессов быть не должно.

Reducers производят расчет того, как должно измениться хранилище. В большинстве приложений много reducers, каждый из которых отвечает за часть хранилища. Это помогает сохранять файлы лаконично и сфокусированно.

import initialState from '../constants/initialState'  
import \* as types from '../constants/types'  
  
//Функция, принимающая два параметра - состояние и действие   
export function loading(state = initialState.loading, action) {  
 // Обычно используется инструкция switch для явного управления всеми типами действия и возврата состояния по умолчанию  
 switch (action.type) {  
 // Если действие имеет тип loading, возвращает true для нового значения состояния  
 case types.app.LOADING:  
 return true  
 // Обработка случая loaded и возвращение соответствующего варианта false  
 case types.app.LOADED:  
 return false  
 // Возвращение существующего состояния по умолчанию  
 default:  
 return state  
 }  
}

Когда будет отправлен action, связаный с загрузкой, хранилище Redux сможет что-то с ним сделать. Когда action передает и проходит через любое существующее промежуточное ПО, Redux привлекает reducers, чтобы определить, какое новое состояние должно быть создано на основе action.

### Объединение редукторов в нашем хранилище combineReducers()

К хранилищу подключен всегда только один reducer. Для подключение нескольких reducers используется combineReducers.

// src/reducers/index.js  
  
import { combineReducers } from 'redux'  
  
//Импортирование reducers, чтобы их можно было добавить к корневому reducer  
import { error } from './error'  
import { loading } from './loading'  
import { pagination } from './pagination'  
import { user } from './user'  
import { comments, commentIds } from './comments'  
  
const rootReducer = combineReducers({  
 commentIds,  
 comments,  
 error,  
 loading,  
 pagination,  
 user  
})  
  
export default rootReducer

### Pure(Чистая) and Impure(Нечистая) Functions

Функция должна удовлетворять двум условиям, чтобы считаться «чистой»:

* Каждый раз функция возвращает одинаковый результат, когда она вызывается с тем же набором аргументов
* Нет побочных эффектов

**Одинаковый вход => Одинаковый выход**

const add = (x, y) => x + y;  
add(2, 4); // 6

**Нечистые функции = непостоянные результаты**

Нечистая функция полагается на общее состояние для выполнения своей работы путем увеличения переменной за пределами своей области.

let x = 2;  
const add = (y) => {  
 x += y;  
};  
add(4); // x === 6 (the first time)

Эта модель кошмар для разработчиков.

**Нет побочных эффектов** таких как:

* меняющийся вход
* HTTP-вызовы
* запись на диск
* вывод на экран