Вводная часть

Redux является предсказуемым контейнером состояния для JavaScript приложений.

Все состояние вашего приложения сохранено в объекте внутри одного хранилища (store). Единственный способ изменить дерево состояния - это вызвать действие (action), объект описывающий то, что случилось. Чтобы указать, каким образом действия преобразовывают дерево состояния, вы пишете чистые функции.

*Функция является чистой, если всегда возвращает одинаковое значение для одинакового набора аргументов.*

Redux основан на 3 главных концепциях:

* Существует единственный источник данных для всего состояния приложения.
* Это состояние только для чтения. Чтобы изменить состояние, необходимо сделать копию всего хранилища.
* Все изменения в состоянии приложения делаются чистыми функциями (редюсеры).

При использовании в сочетании с лучшими практиками, разработанными сообществом, эти принципы позволяют создавать поддерживаемые и простые в тестировании приложения.

# Единственный источник данных

Например, состояние простого приложения со списком задач может быть простым массивом с объектами задач.

const state = [

{

id: 1,

task: 'Do laundry',

completed: true

},

{

id: 2,

task: 'Paint fence',

completed: false

}

];

Состоянием для сайта социальных медиа может быть ассоциативный массив, содержащий информацию о постах, уведомлениях, профиле пользователя и иные социальные данные.

const defaultState = {

posts: [

// post objects to appear in user's feed

],

notifications: [

// unread notifications for the user

],

messages: [

// new messages

],

friends: [

// other online users

],

profile: null

}

Независимо от размера приложения, все данные состояния хранятся в одном объекте.

# Состояние только для чтения

Например, обработчик добавления задачи к списку не сможет непосредственно добавить новое задание в массив. Вместо этого обработчик отправит действие, говорящее: “Привет, приложение. Пора добавить задачу “Купить молока” в массив с заданиями.”

Действие в Redux это простой объект JavaScript, выражающий намерение изменить состояние объекта. Он содержит минимальную информацию, требуемую для описания того, что должно измениться в результате действий пользователя. Единственный обязательный атрибут действия это его тип, все остальные данные, включенные в действие, будут специфичны для конкретного приложения и типа произведенного действия. Когда пользователь добавляет задачу “Купить молока”, действие будет выглядеть так:

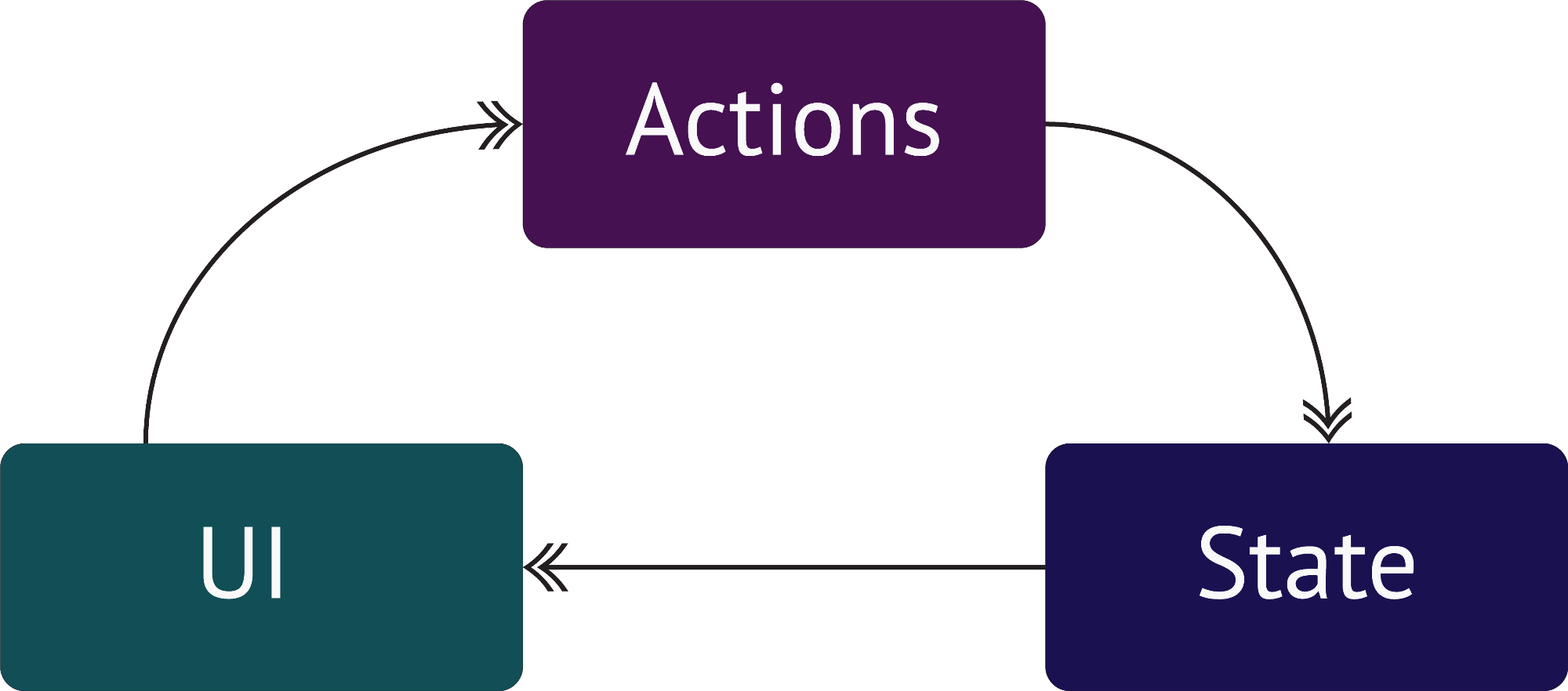
{

type: 'ADD\_TODO',

task: 'Buy milk',

id: 3

}



# Изменения производятся чистыми функциями

Для прослушивания действий есть отдельная функция. Это по сути один большой переключатель **switch**, реагирующий на поле **type** действия. Каждое действие, которое может возникнуть в вашем приложении, нуждается в соответствующем case с функцией рассчитывающей новое состояние приложения на основе текущего состояния и данных действия.

Например, функция для действия 'ADD\_TODO' не добавляет новое значение в массив заданий. В таком случае она не будет чистой, так как модифицирует массив. Вместо этого функция для 'ADD\_TODO' создаст копию массива задач, добавит новое задание в конец этого массива и затем вернет новый массив как новое состояние приложения.

(currentState, action) => {

switch(action.type){

case 'ADD\_TODO':

const nextState = [

...currentState,

{

id: action.id,

task: action.task,

completed: false

}

];

return nextState;

break;

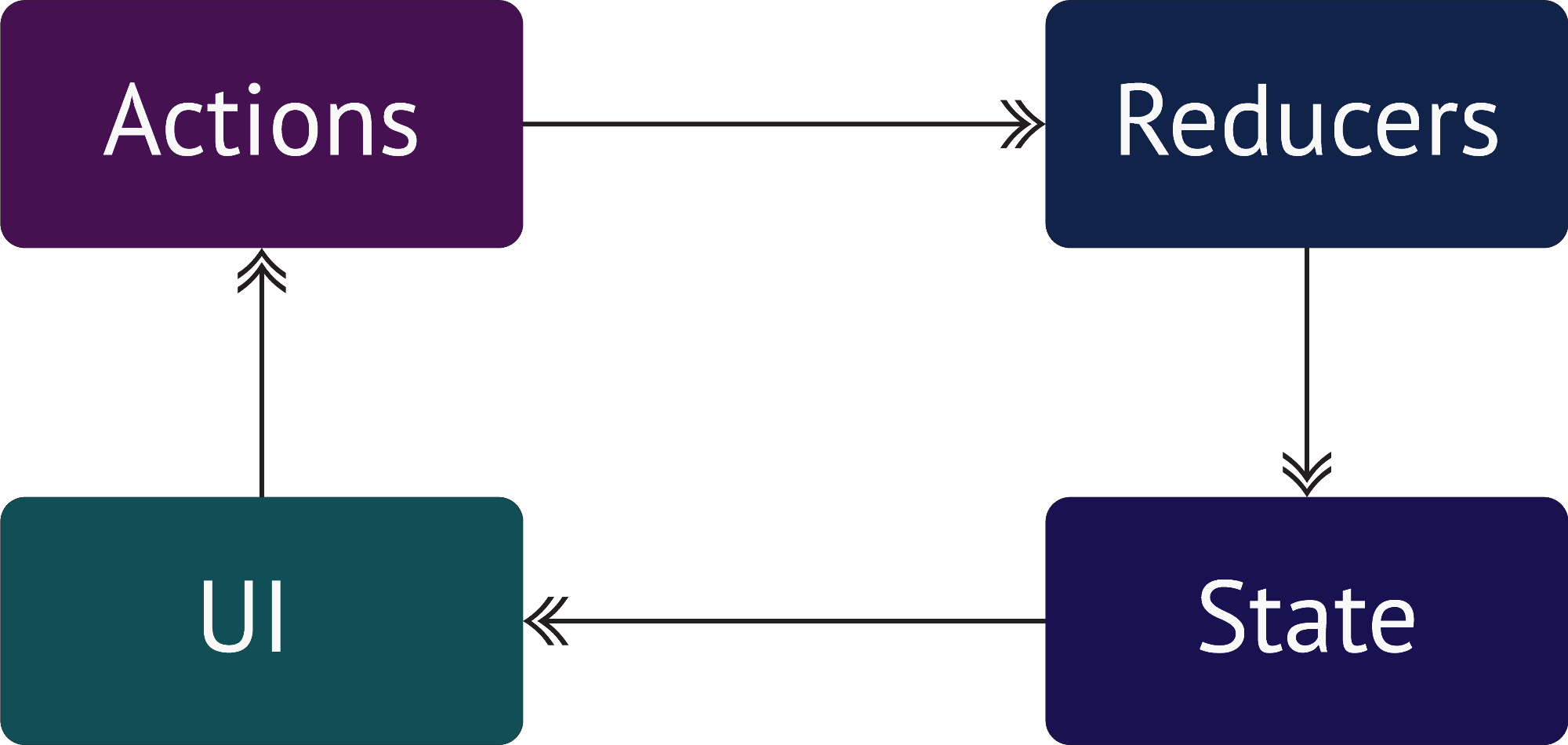
default:

return currentState;

}

};

Эта чистая функция знающая, как трансформировать текущее состояние приложения со всеми действиями в обновленное состояние приложения, называется корневой редьюсер. Тот факт, что корневой редьюсер рассчитывает следующее состояние вместо того, чтобы модифицировать текущее, очень важен в Redux.



Лабораторная работа

Создайте следующий HTML документ.

<!DOCTYPE html>

<html>

<head>

<title>Redux счетчик</title>

<script *src*="https://unpkg.com/redux@latest/dist/redux.min.js"></script>

</head>

<body>

<div>

<p>

Счетчик:

<span *id*="value">0</span>

<button *id*="increment" *onclick*="incrementClick()">+</button>

<button *id*="decrement" *onclick*="decrementClick()">-</button>

<button *id*="decrement" *onclick*="showState()">Показать состояние</button>

</p>

</div>

<script *src*="index.js"></script>

</body>

</html>

Добавьте скрипт index.js с содержимым

function reducer(state, action) {

*if* (typeof state === 'undefined') {

*return* 0

}

*switch* (action.type) {

*case* 'INCREMENT':

*return* state + 1

*case* 'DECREMENT':

*return* state - 1

*default*:

*return* state

}

}

function incrementClick() {

store.dispatch({ type: 'INCREMENT' });

}

function decrementClick() {

store.dispatch({ type: 'DECREMENT' });

}

var store = Redux.createStore(reducer);

var valueEl = document.getElementById('value');

function showState() {

valueEl.innerHTML = store.getState().toString();

}

Задание

Добавить в лабораторную работу «список задач» библиотеку redux и реализовать работу через эту библиотеку.

Минимально в хранилище должен быть массив с задачами. В каждой задаче минимум 2 поля

* Id – идентификатор задачи
* Text – текст задачи, введенный в поле ввода

В качестве названий действий использовать:

* 'ADD\_TODO' – добавление задачи в хранилище
* 'REMOVE\_TODO' – удаление задачи из хранилища

Полезные ссылки и литература

1. <https://github.com/rajdee/redux-in-russian>
2. http://prgssr.ru/development/rabota-s-redux-vvedenie.html