Основы работы с Redux

Алексей Елисеев, aleksey_elyseev@epam.com

План

1. Redux API: Actions, Reducers, Store
2. Redux Data Flow
3. Middleware
4. Архитектура React приложений
5. React-Redux

Redux v1.0.0

14 августа, 2015

«Love what you're doing with Redux!»
Jing Chen, Flux

«I asked for comments on Redux in FB's internal JS discussion group, and it was universally praised. Really awesome work.»
Bill Fisher, Flux

Что такое Redux

Библиотека представляет инструментарий для управления данными приложения

Redux применяется со многими известными JS-фреймворками, например Angular 1 и 2, Vue

Размер Redux — около 2Кб, включая зависимости

Три принципа

1. Один источник данных о текущем состоянии приложения
2. Единственный способ изменить состояние приложение это применить действие — суть, объект, который описывает что произойдет
3. Функции, ответственные за изменения состояния (*редьюсеры в терминологии Redux*) написаны как чистые функции

Redux API

```
import {
    createStore,
    combineReducers,
    bindActionCreators,
    applyMiddleware,
    compose
} from 'redux'

const Store = createStore(/*...*/)

const {getState, dispatch, subscribe, replaceReducer} = Store
```

Пример создания Redux Store

```
import {createStore} from 'redux'

const reducer = (state = 1, {type, payload}) => {
    switch (type) {
        case 'INCREMENT':
            return state + 1;

        case 'DECREMENT':
            return state - 1;

        case 'CHANGE_COUNTER':
            return state + payload;

        default:
            return state;
    }
}
const Store = createStore(reducer); // Store.getState() => 1
Store.dispatch({type: 'INCREMENT'}); // Store.getState() => 2
Store.dispatch({type: 'INCREMENT'}); // Store.getState() => 3
Store.dispatch({type: 'DECREMENT'}); // Store.getState() => 2
Store.dispatch({type: 'CHANGE_COUNTER', payload: 20}); // Store.getState() => 22
```

Actions

1. **Actions** это носители информации от приложения к хранилищу (Store)
2. **Actions** — простые Javascript объекты
3. объект **Action** обязательно содержит свойство **type**, название действия

```
const isPlainObject = require('lodash/isPlainObject');
isPlainObject(action) // => true
action.hasOwnProperty('type') // => type
```

Flux Standart Action (FSA)

```
//typescript
type Action = {
    type: string|symbol,
    payload?: any,
    meta?: any,
    error?: boolean
}
```

Actions creators

```
const change = val => ({
    type: 'CHANGE_COUNTER',
    payload: val
})

const increment = change(1)
const decrement = change(-1)
```

Bounded action creators

```
const Store = createStore(/*...*/)
const {dispatch} = Store;
const bindChangeToDispatch = dispatch => val => dispatch(change(val))
const bindedChange = bindChangeToDispatch(dispatch)
const bindedIncrement = bindChangeToDispatch(dispatch)(1)

// somewhere in code, as on click handler, for example
Store.dispatch(increment())
bindedIncrement()
```

Reducers

1. Reducers — это обработчики действий передаваемых в Actions
2. Reducers — только чистые функции, нет сайд-эффектов, нет асинхронных вызовов

```
const reducer = (prevState, action) => nextState

//looks like
reduce((acc, val) => acc)
```

Пример проверки редьюсера на отсутствие побочных эффектов

```
'use strict'
// https://www.npmjs.com/package/deepfreeze
const deepfreeze = require('deepfreeze')

const initialState = {/*...*/}
const actionCreator = (...args) => {
    // return some action object
}

const reducer = (state, action) => {
    // do something
}

deepfreeze(initialState)

expect(reducer(initialState, actionCreator(...params))).to.not.throw
```

Комбинация редьюсеров

```
import {createStore, combineReducers} from 'redux'

const stateSchema = {
    user: {
        name: 'string', lastname: 'string'
    },
    mail: {
        letter: 'any[]'
    }
}

const userReducer = (state, {type, payload}) => {
    /*
    state = { name: 'string', lastname: 'string' }
    */
    return state
}

const mailReducer = (state, {type, payload}) => {
    /*
    state = { letter: 'any[]' }
    */
    return state
}

const Store = createStore(combineReducers({user: userReducer, mail: mailReducer}))
```

Store

1. Управляет данными приложения
2. Возращает текущий статус приложения **Store.getState()**
3. Обновляет данные приложение с помощью Actions — **Store.dispatch(action)**
4. Добавляет и удаляет обработчики

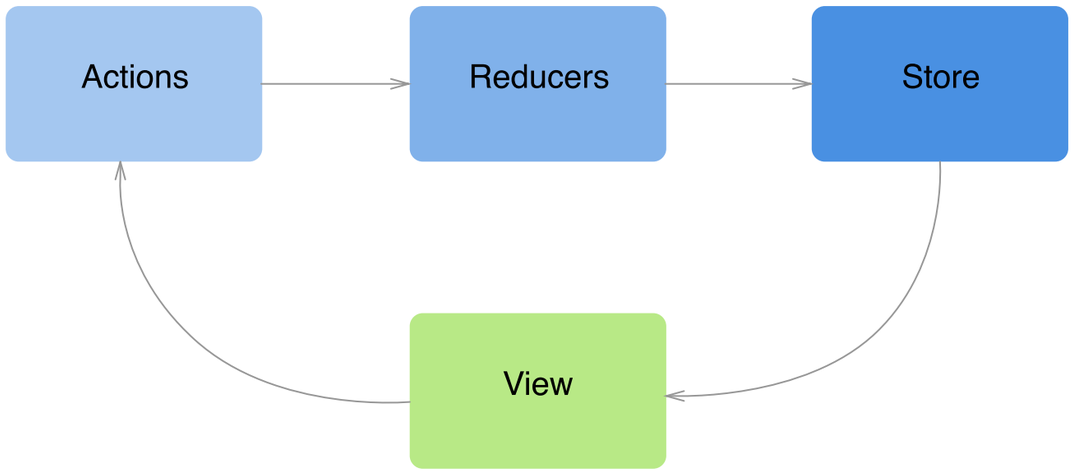
```
const Store = createStore(/*...*/)
const handler = function() {/*...*/}

const unsubscribe = Store.subscribe(handler)

// somewhere later

unsubscribe()
```

Redux data flow



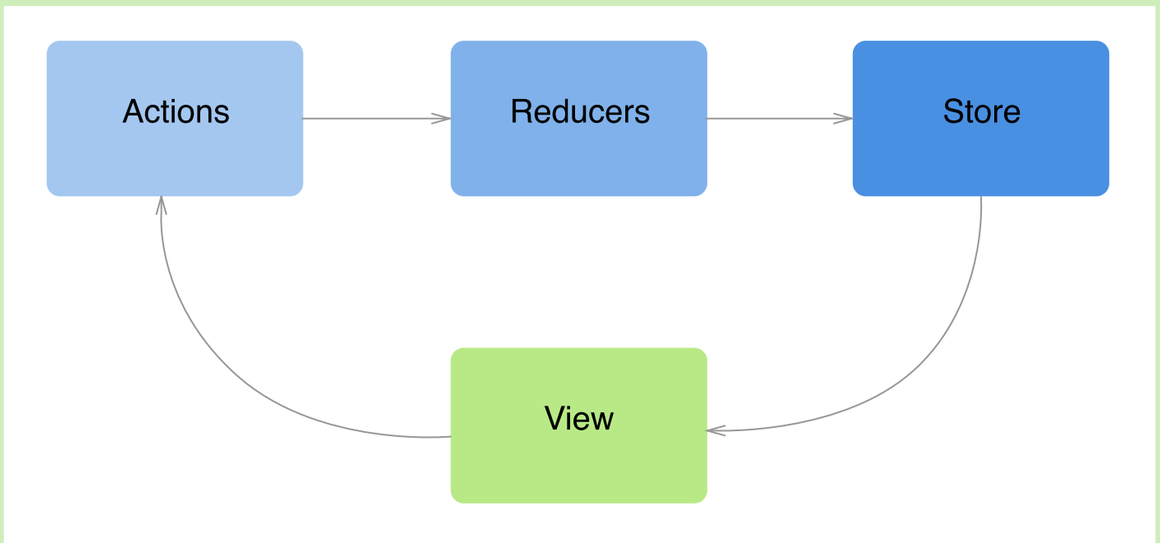
Пример простого приложения

5 + 4 + 3 = 12

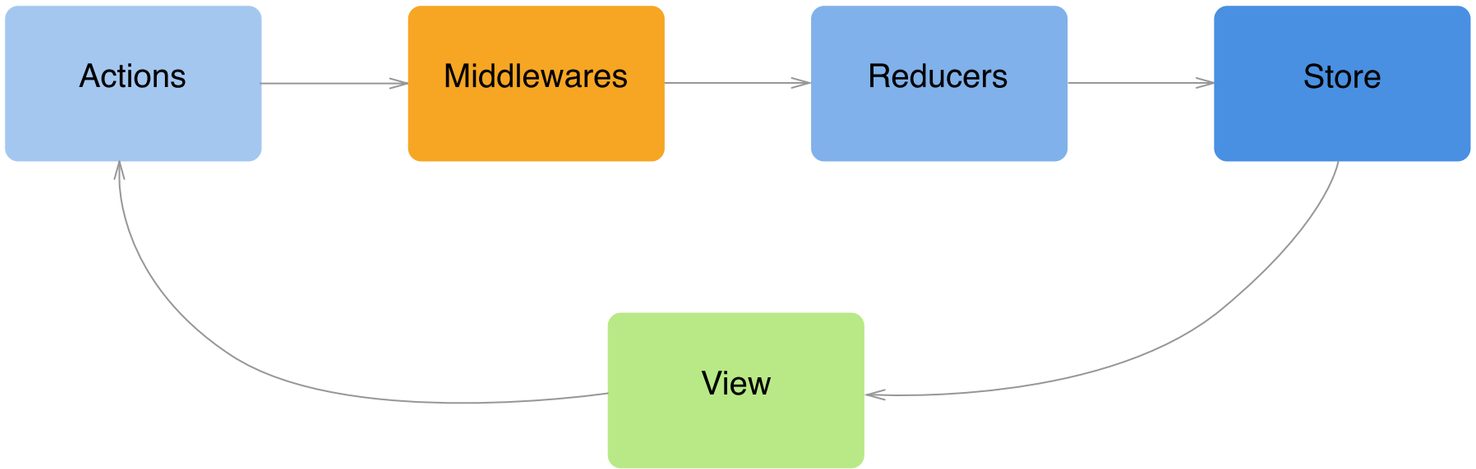
− + − + − +

Сайд эффекты

Actions простые объекты, а **Reducers** чистые функции, которые возвращают новое состояние — сайд-эффекты, например ajax запросы, невозможны



Middlewares



Middleware — это функция, которая принимает текущий Store, следующий в цепочке middleware и текущее действие.

Middleware не обязательно чистая функция — и поэтому может делать запросы на сервер, отложенное выполнение действий, и т.д.

Примеры реализации Middleware

```
// logging
const logMiddleware = store => next => action => {
    const {type, ...data} = action;
    console.group(type);
    console.log(JSON.stringify(data));
    console.groupEnd();
    next(action);
};

// async middleware @see redux-thunk
const thunkMiddlware = store => next => action => {
    if (typeof action === 'function') {
        action(store.dispatch);
    } else {
        next(action);
    }
}

// action sample for this middleware
const requestAction = url => dispatch => {
    dispatch(startLoadingDataAction())
    fetch(`api/${url}`)
        .then(response => dispatch(setResponseAction(response)))
        .catch(err => dispatch(displayErrorAction(err)))
}
```

Подключение Middleware

```
// logging
const logMiddleware = store => next => action => {
    const {type, ...data} = action;
    console.group(type);
    console.log(JSON.stringify(data));
    console.groupEnd();
    next(action);
};

// async middleware @see redux-thunk
const thunkMiddlware = store => next => action => {
    if (typeof action === 'function') {
        action(store.dispatch);
    } else {
        next(action);
    }
}

const Store = createStore(reducer, applyMiddleware(logMiddleware, thunkMiddlware))
```

Примеры готовых решений

1. Redux-logger — логирование
2. Redux-thunk — для организации простых сайд-эффектов
3. Redux-saga — готовое решение для организации сложных последовательностей сайд-эффектов
4. Redux-debounce — debounce для actions
5. Redux-promise — поддержка actions
   `{type: string, payload: Promise}`

...

Redux API recap

```
import {
    createStore,
    combineReducers,
    bindActionCreators,
    applyMiddleware,
    compose
} from 'redux'

const Store = createStore(/*...*/)

const {getState, dispatch, subscribe, replaceReducer} = Store
```

Redux API recap #2

1. **createStore(reducer, initialState, enhancer)**, initialStore определяется либо на этом этапе, но чаще как значение по умолчанию в reducers, `function mail(state = [], action)`
2. **combineReducers(reducers)**, `reducers: {A: reducerA, B: reducerB}`
3. **bindActionCreators(actionCreators, dispatch)**, actionCreators функция, или объект `{key => actionCreator}`
4. **applyMiddleware(...middlewares)**, функции удовлетворяющие Redux middleware API `({getState, dispatch}) => next => action`
5. **compose(...function)** — композиция функций, (*для реализации цепочки middleware*)

Архитектура приложения «Container Components, Presentational Components»

Presentational Components

1. Содержат как Presentation так и Container Components — часто содержат только верстку
2. Никогда не содержат зависимостей от модели приложения — *actions, state...*
3. Как правило это stateless компоненты
4. State, если он есть определяет состояние UI, а не данных

Container Components

1. Содержат как Presentation так и Container Components, но не содержат верстки, не содержат стилей — максимум могут быть врапперы.
2. Содержат зависимости от модели приложения
3. Часто statefull

Пример

```
class UserInput extends React.Component {
    constructor(props) {
        super(props)
        this.state = {user: props.user}
    }
    componentDidMount() {
        fetch(`/api/getUser?id=${this.state.user.id}`)
            .then(response => this.setState({user: response.user}))
    }
    render() {
        return (
            <div className="input">
                <label className="input-label">{this.state.user.label}</label>
                <input className="input-field" type="text" value={this.state.user.name}/>
            </div>
        )
    }
}

//usage
<UserInput user={{id: 8877}}/>
```

Рефакторинг

```
// Presentational Component
function Input({label, value}) {
   return (
           <div className="input">
               <label className="input-label">{label}</label>
               <input className="input-field" type="text" value={value}/>
           </div>
       )
}

import CancelablePromise from 'cancelable-promise'

// Container Component
class UserInput extends React.Component {
    componentWillUnmount() {
        this.request.cancel()
    }
    componentDidMount() {
        this.request = new CancelablePromise(resolve => {
             fetch(`/api/getUser?id=${this.state.user.id}`)
                .then(response => resolve(response))
        })
        this.request.then(response => this.setState({user: response.user}))
    }
    render() {
        return <Input label={this.state.user.label} value={this.state.user.value}/>
    }
}
```

Рефакторинг #2

```
// thunk-action
const requestUser = id => dispatch => {
    fetch(`/api/getUser?id=${id}`)
        .then(response => dispatch(setUser(response)))
}

// Presentational Component
function Input({label, value}) {
   return (
           <div className="input">
               <label className="input-label">{label}</label>
               <input className="input-field" type="text" value={value}/>
           </div>
       )
}

// Container Component
class UserInput extends React.Component {
    // or componentDidMount, or componentWillUpdate, or componentWillReceiveProps ...
    constructor(props) {
       super(props)
       requestUser(props.user.id)
    }
    render() {
        const {user: {value, label}} = this.props
        return value ? <SomeLoadingIndicator/> : <Input label={label} value={value}/>
    }
}
```

Пример добавляем Provider

React-Redux

React-Redux API

```
import {Provider, connect} from 'react-redux';
import AppRootComponent from './app';
import {createStore} from 'redux';

const Store = createStore(/*...*/)

<Provider store={Store}><AppRootComponent/><Provider>

// somewhere
@connect(state => ({
    users: state.users
}),
(dispatch, props) => ({
    onClick() {
        dispatch(someActionCreator(props))
    }
}))
class Users extends React.Component {}

const UserContainer = connect(
    state => {users: state.users},
    (dispatch, props) => ({onClick() {dispatch(someActionCreator(props))}})
)(User)
```

React-Redux API #2

1. **`<Provider store>`** передает store в контесте дочерних компонентов
2. **`connect([mapStateToProps], [mapDispatchToProps], [mergeProps], [options])`** вычисляет свойства из Store
   - `mapStateToProps(state, [ownProps])` — функция
   - `mapDispatchToProps(dispatch, [ownProps])` — объект или функция
   - `mergeProps(stateProps, dispatchProps, ownProps)` — по умолчанию Object.assign, разрешение конфликтов между собственными свойствами и вычисляемыми

Использование селекторов

```
import {createSelector, defaultMemoize} from 'reselect'

const getUsers = state => state.users
const getLetters = state => state.letters

const getMailBoxes = createSelector(getUsers, getLetters, (users, letter) => {
    // filter, find, reduce, e.t.c
})

// selector are memoized functions
const _complexFunction = {/*...*/}
const complexFunction = defaultMemoize(_complexFunction /* [equalityCheck], === by default */)

// using with mapStateToProps
const mapStateToProps = state => ({
    users: getUsers(state),
    letters: getLetters(state),
    mail: getMailBoxes(state)
})
```

Пример с использованием react-redux

Ссылки, инструменты

1. Redux, код, документация
2. React-redux, код, документация
3. Redux-ui, код, документация
4. Normalizr, код и документация, примеры
5. Reselect, код и документация

Ссылки, статьи

1. Ссылки из официальной документации
    - видео,
    - примеры,
    - статьи
2. Getting Started with Redux
3. Building React Applications with Idiomatic Redux
4. Обзор Redux without profanity, раздел о Redux-UI
5. Контейнер- и простые компоненты, статья по архитектуре React-приложений
6. Примеры из лекции решение «в лоб», использование провайдера, интеграция с react-redux

Спасибо!