

Управление состоянием

State management

State management: что это и зачем?

Показательный пример

ewf

по названию
по описанию
везде

Заметка №1

Создать

Описание заметки

Места

все посетить посещенные

2018 ©spt3O

Character

Swatches

3

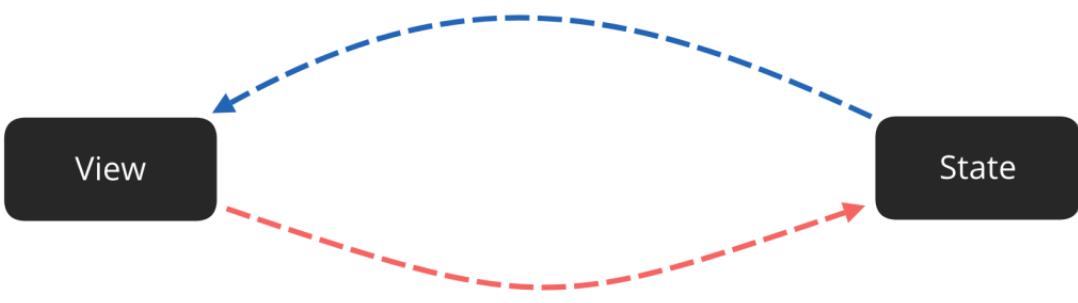
С чем были проблемы?

- Приходилось постоянно держать кучу обработчиков на разные события, вовремя их добавлять и удалять
- Необходимо было всегда (или почти всегда) понимать, почему сработал тот или иной обработчик
- Обработчики делали что-то в нескольких несвязанных местах, терялся контроль над происходящим в приложении

Определения

- *View* – часть приложения, отвечающая за отображение данных на странице
- *Состояние приложения (Application State)* – структурированные данные о конкретном экземпляре приложения для одного конкретного пользователя

Двусторонняя связь



- Некоторые изменения View вызывают изменения State.
Например, ввод текста или выбор радиокнопки

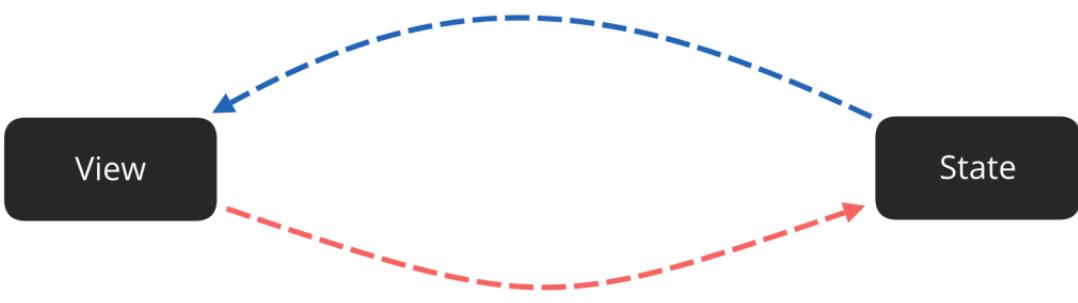
React

- Есть функция `setState`

```
this.setState((state, props) => {
    return { counter: state.counter + props.step };
})
```

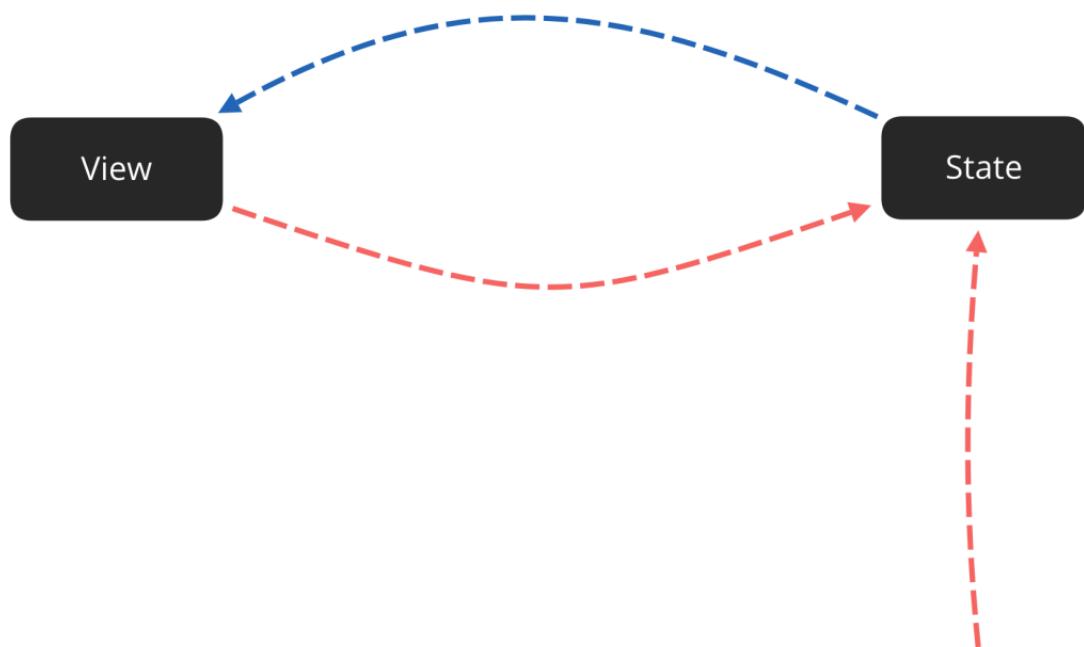
- Любые изменения `state` принудительно вызывают `render()`

Двусторонняя связь

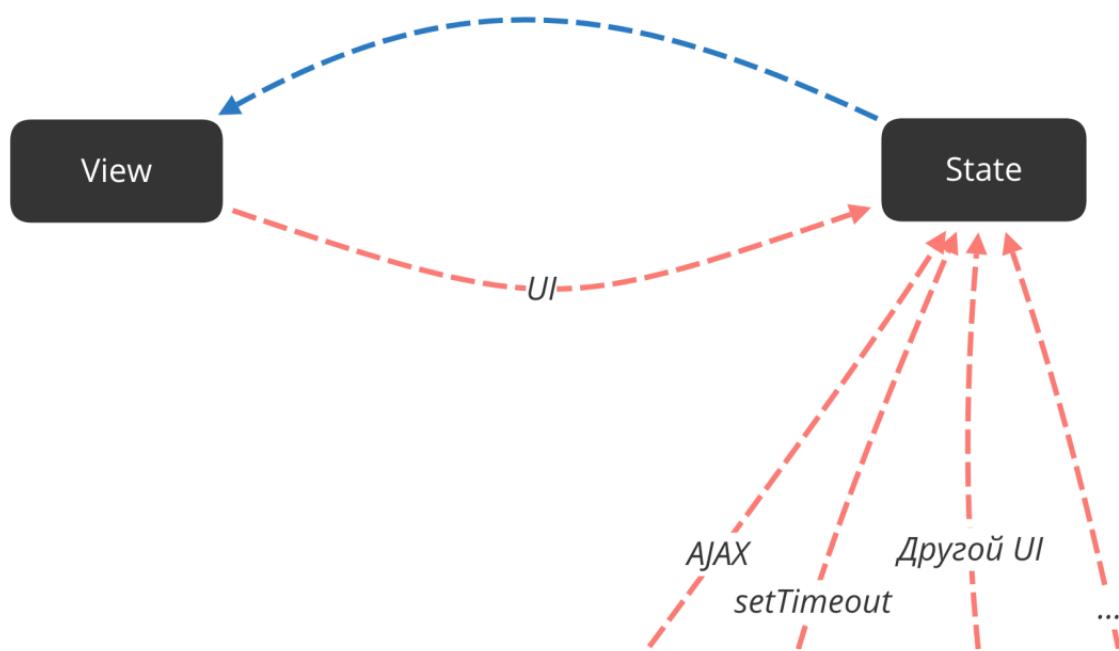


- Некоторые изменения View вызывают изменения State.
Например, ввод текста или выбор радиокнопки

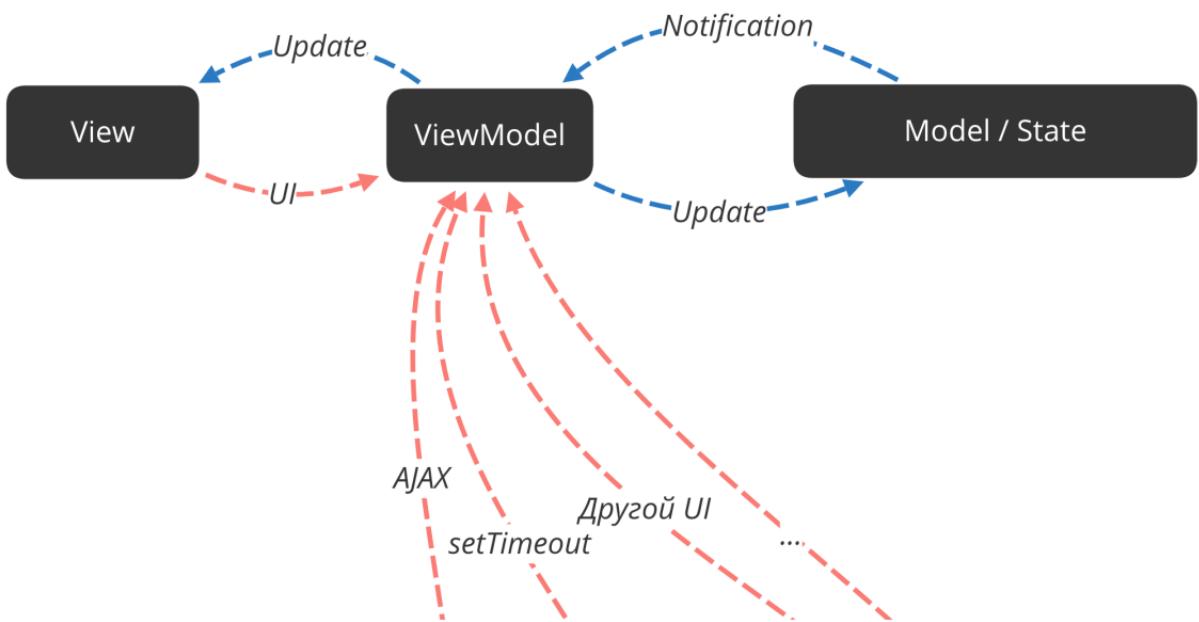
Двусторонняя связь



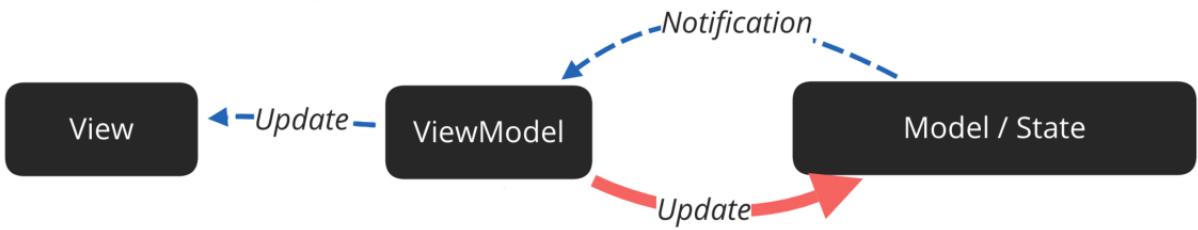
Двусторонняя связь



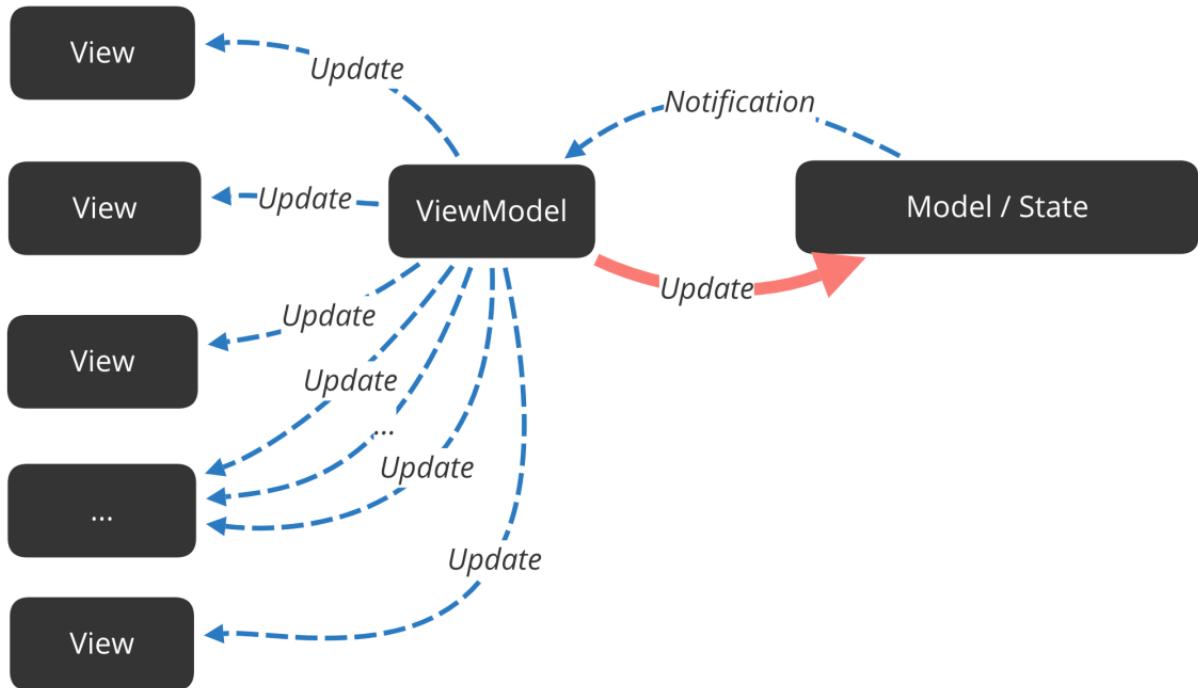
MVVM (Model-View-ViewModel)



В чем здесь проблема?



Вопросы к MVVM:

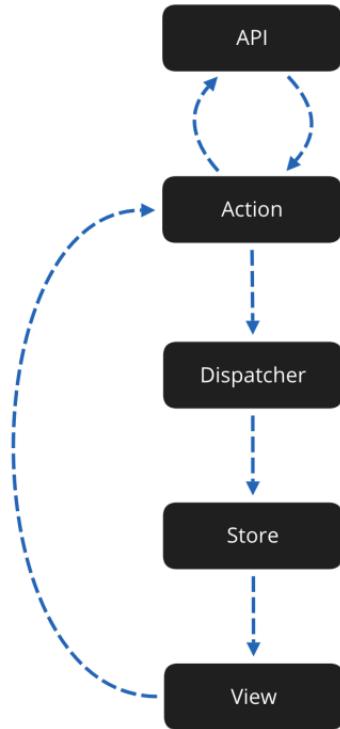


Вопросы к MVVM:

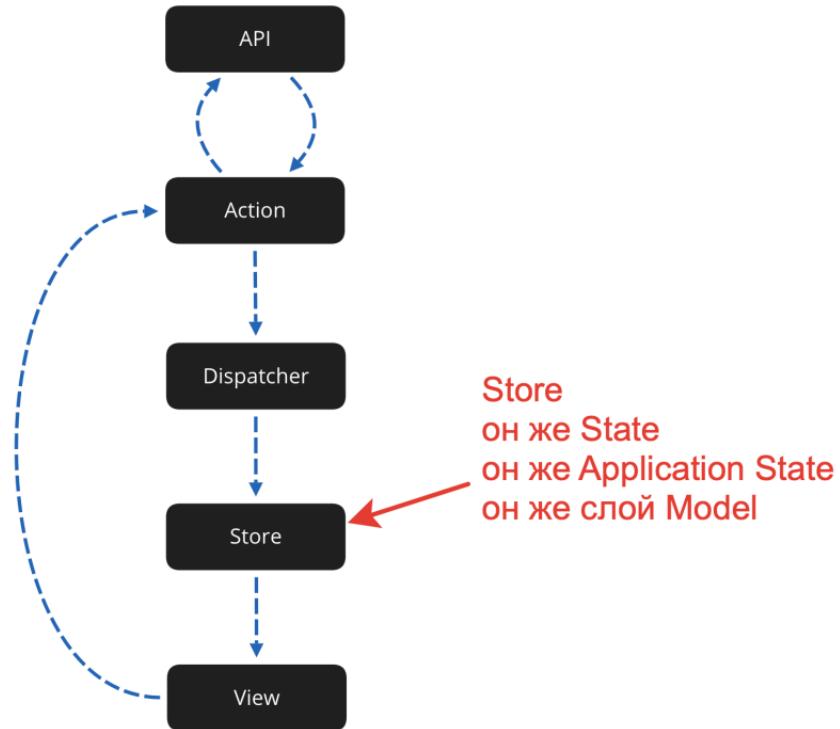
- Как контролировать внесение изменений в State? Откуда пришли данные в State? из View/из AJAX-запроса/из записи в storage
- Как вообще отлаживать изменение View после изменения State?
- Как убедиться в том, что после изменений State остался корректным?

Flux-архитектура

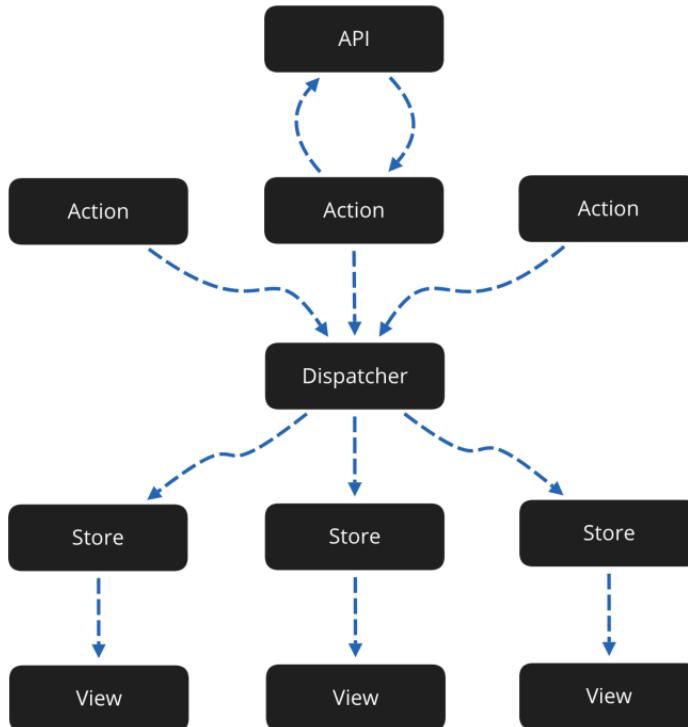
Вот такая:



Вот такая:

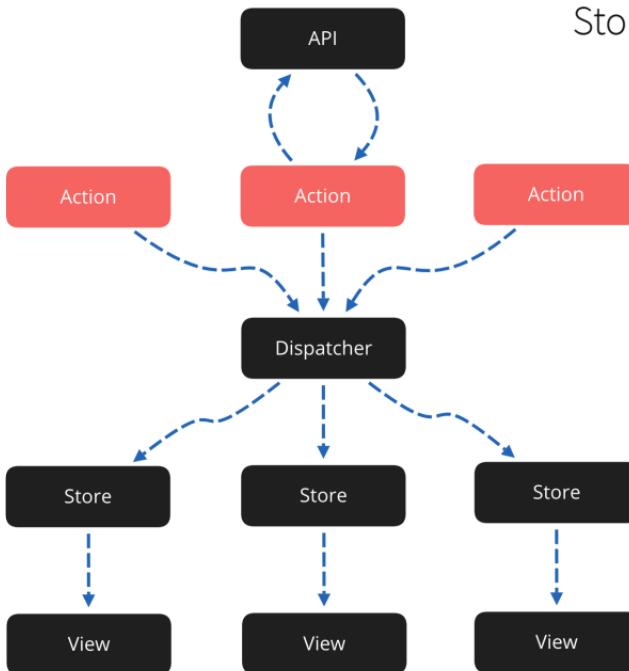


На самом деле вот такая:



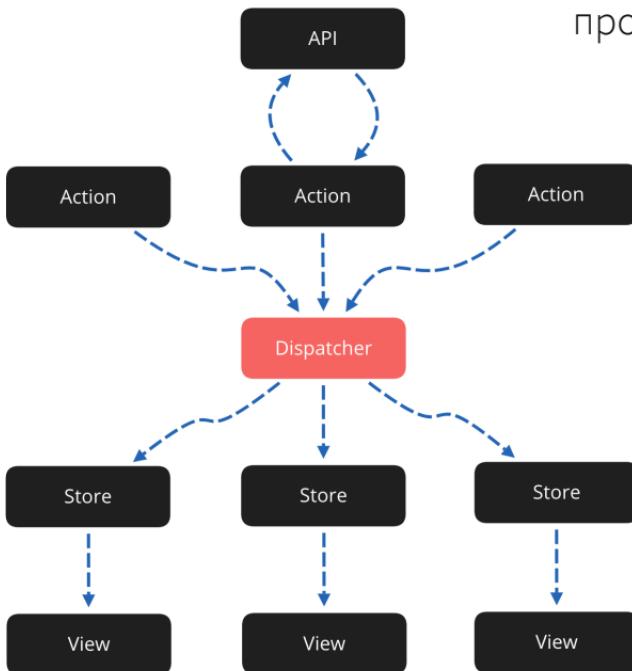
Actions

Действия, которые поступают в диспетчер с целью отработать в Store



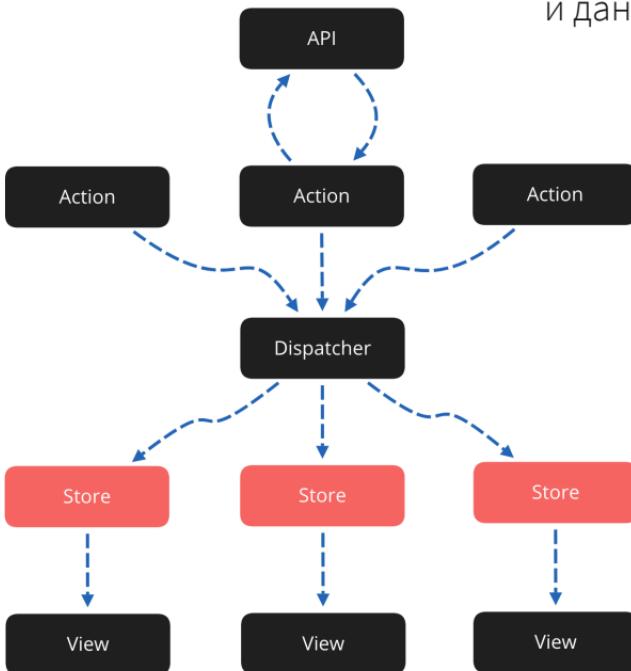
Dispatcher

Сущность, которая упорядочивает и синхронизирует проброс Actions в Store



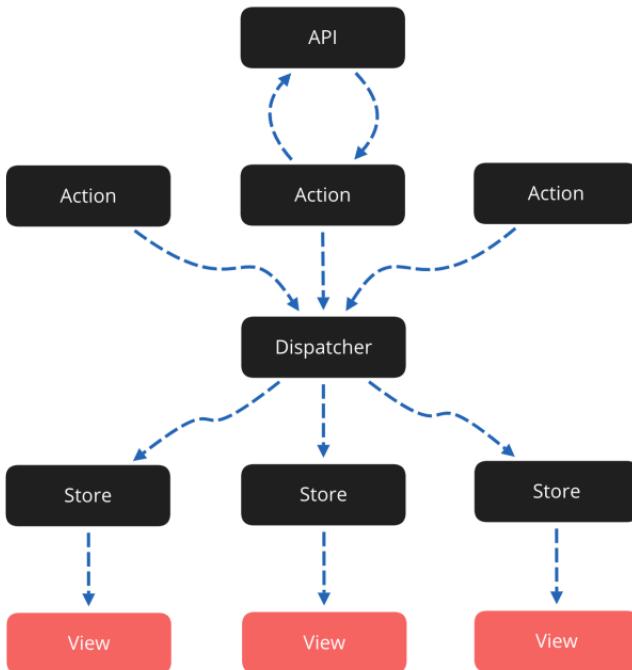
Store

Состояние приложения.
Изменяется *строго на*
основе старого состояния
и данных из Action

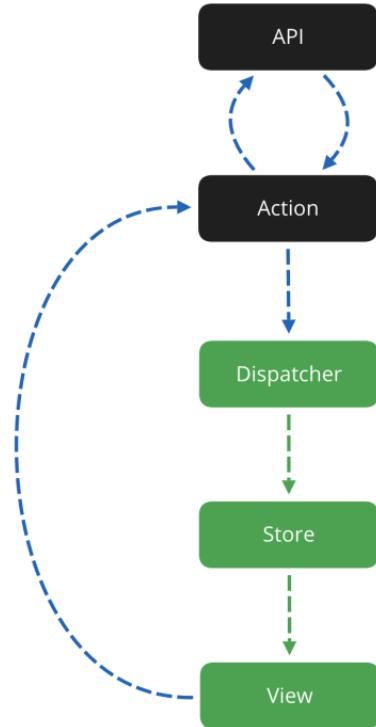


View

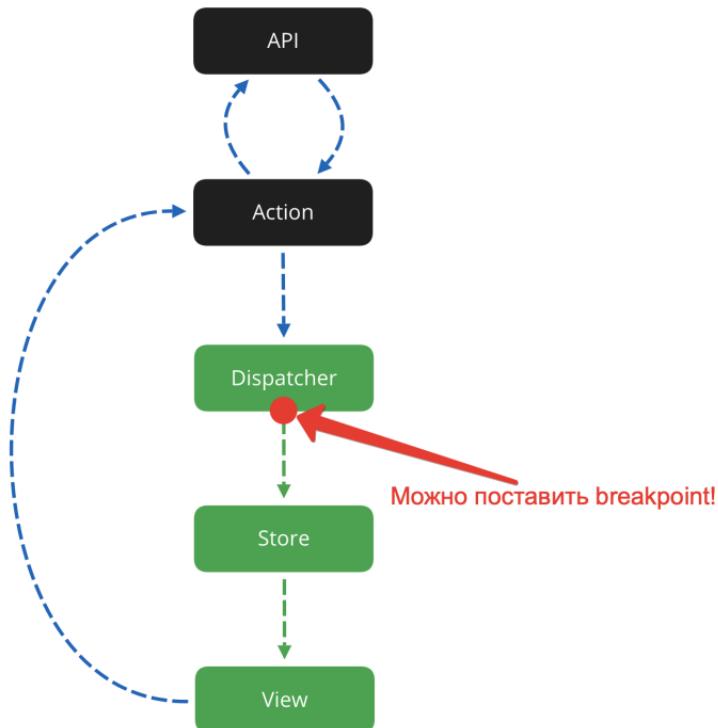
Конечная точка
использования
вашего Store



А в чем особенность?



Модель взаимодействия синхронная!



Вопросы к MVVM:

- Как контролировать внесение изменений в Store (откуда пришли данные в Store? из View/из AJAX-запроса/из записи в storage (нужно подчеркнуть))

actions!

Вопросы к MVVM:

- Как контролировать внесение изменений в Store (откуда пришли данные в Store? из View/из AJAX запроса/из записи в storage (нужно подчеркнуть))

~~actions!~~

- Как вообще отлаживать изменение View после изменения Store?

~~dispatcher!~~

Вопросы к MVVM:

- Как контролировать внесение изменений в Store (откуда пришли данные в Store? из View/из AJAX запроса/из записи в storage (нужно подчеркнуть))

~~actions!~~

- Как вообще отлаживать изменение View после изменения Store?

~~dispatcher!~~

- Как убедиться в том, что после изменений Store остался корректным?

Вопросы к Flux:

- Как контролировать внесение изменений в Store (откуда пришли данные в Store? из View/из AJAX запроса/из записи в storage (нужно подчеркнуть))

~~actions!~~

- Как вообще отлаживать изменение View после изменения Store?

~~dispatcher!~~

- Как убедиться в том, что после изменений Store остался корректным?

Существующие решения

Существующие решения

ngrx (Angular)

vuex (Vue)

reflux (Redux)

storeon

effector

mobx

redux

Существующие решения

ngrx (Angular)

vuex (Vue)

reflux (Redux)

storeon

effector

mobx

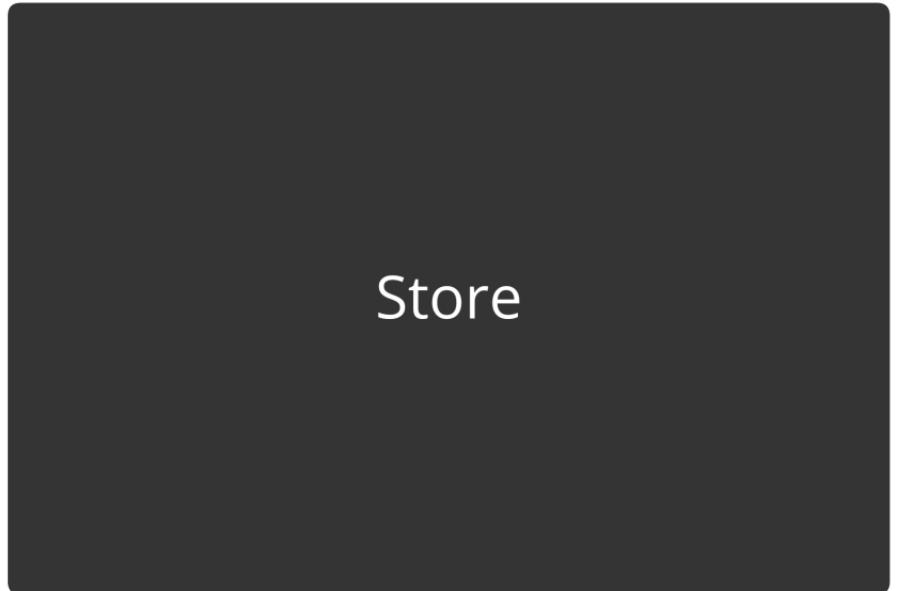
redux



Redux

Фичи Redux

- Один большой Store



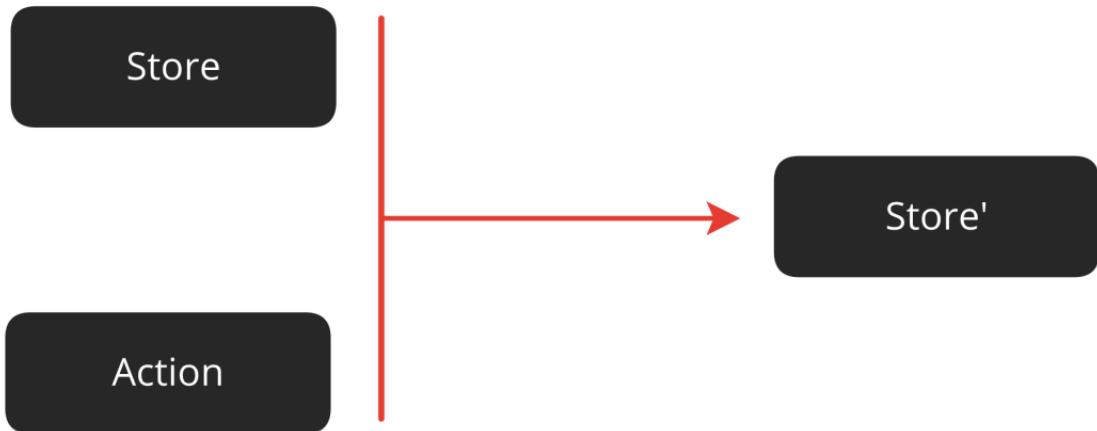
Store

Ликбез: чистая функция

- Чистая функция **не обладает побочными эффектами**: не изменяет свои параметры или глобальные переменные
- Чистая функция **является детерминированной**: всегда возвращает одинаковые значения на одинаковых данных

Фичи Redux

- Есть reducers



- *reducer* является чистой функцией, которая возвращает новый store на основе старого и данных из action

reducers

```
export const notesReducer = (
    state: ApplicationState,
    action: IAction
) => {
    switch (action.type) {
        case 'INCREMENT_NOTE_COUNTER': {
            const { noteCount } = state;
            const { newNoteCount } = action;

            return {
                ...state,
                noteCount: noteCount + newNoteCount
            };
        }
    }
}
```

reducers

Обычно выглядит, как длинный switch-case:

```
export const notesReducer = (  
    state: ApplicationState,  
    action: IAction  
) => {  
    switch (action.type) {  
        case 'INCREMENT_NOTE_COUNTER': {  
            const { noteCount } = state;  
            const { newNoteCount } = action;  
  
            return {  
                ...state,  
                noteCount: noteCount + newNoteCount  
            };  
        }  
    }  
};
```

reducers

Является чистой функцией:

```
export const notesReducer = (
    state: ApplicationState,
    action: IAction
) => {
    switch (action.type) {
        case 'INCREMENT_NOTE_COUNTER': {
            const { noteCount } = state;
            const { newNoteCount } = action;

            return {
                ...state,
                noteCount: noteCount + newNoteCount
            };
        }
    }
}
```

Вопросы к Flux:

- Как контролировать внесение изменений в Store (откуда пришли данные в Store? из View/из AJAX запроса/из записи в storage (нужно подчеркнуть))

~~actions!~~

- Как вообще отлаживать изменение View после изменения Store?

~~dispatcher!~~

- Как убедиться в том, что после изменений Store остался корректным?

Вопросы к Flux:

- Как контролировать внесение изменений в Store (откуда пришли данные в Store? из View/из AJAX запроса/из записи в storage (нужно подчеркнуть))

~~actions!~~

- Как вообще отлаживать изменение View после изменения Store?

~~dispatcher!~~

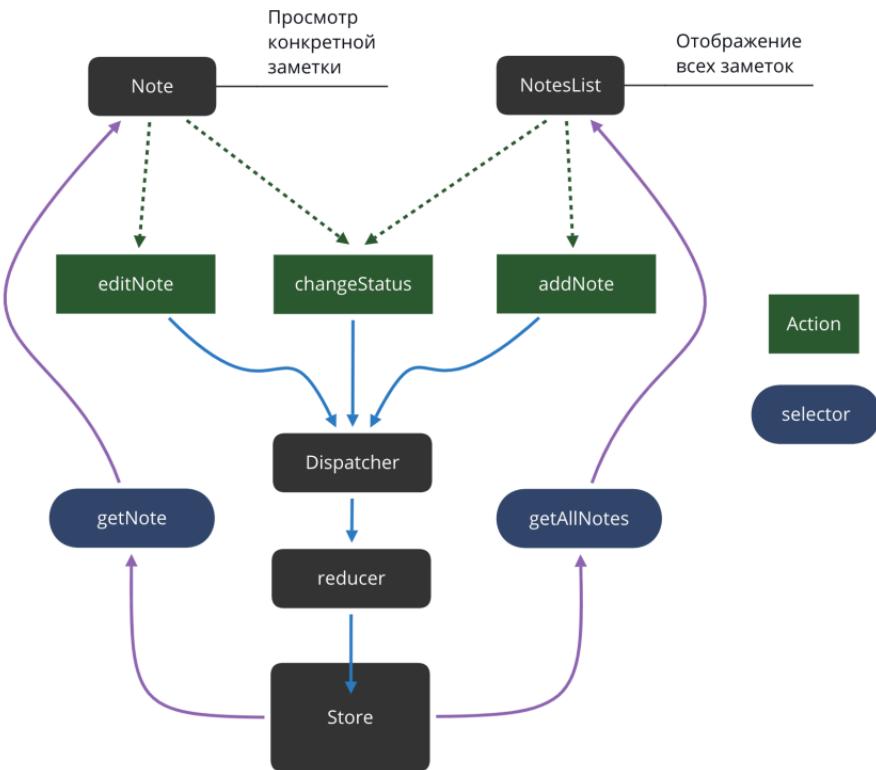
- Как убедиться в том, что после изменений Store остался корректным?

Некоторые моменты, которые полезно знать перед тем как начать разрабатывать с Redux:

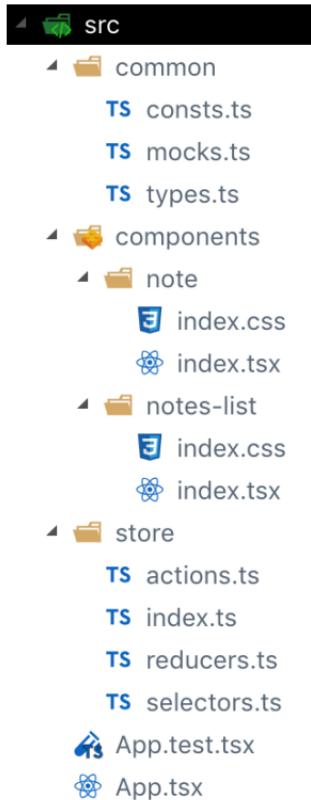
- store и вся работа с ним кладется в отдельную папку (опционально)
- Библиотека `redux` для работы со Store
- Библиотека `react-redux` для использования в компонентах React
- Есть Redux DevTools, который отлично помогает при разработке

Связка React ↔ Redux

Пример приложения



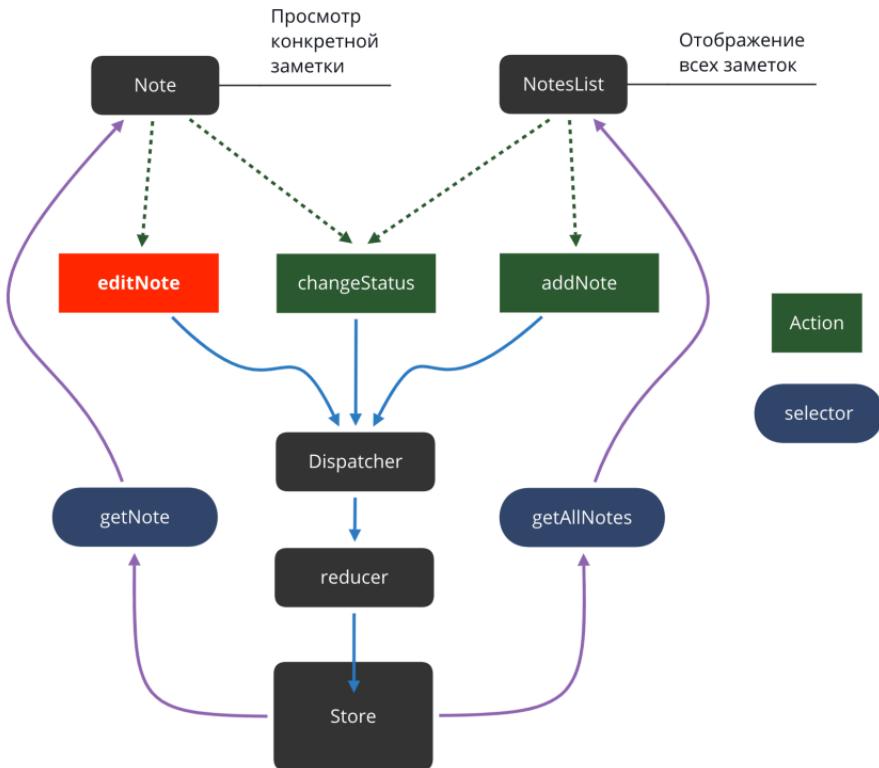
Пример приложения



actions

```
src
  common
    consts.ts
    mocks.ts
    types.ts
  components
    note
      index.css
      index.tsx
    notes-list
      index.css
      index.tsx
  store
    actions.ts
    index.ts
    reducers.ts
    selectors.ts
    App.test.tsx
    App.tsx
```

actions



actions

/src/store/actions.ts

```
import {  
    IAddNotePayload,  
    IEditNotePayload,  
    IChangeStatusPayload,  
    ACTIONS,  
    IAction  
} from '../common/types';  
  
export const editNote = ({ id, newNote }: IEditNotePayload): IAction =>  
{  
    type: ACTIONS.EDIT_NOTE,  
    payload: {  
        id,  
        newNote  
    }  
};
```

Каждый action – есть функция

/src/store/actions.ts

```
import {  
    IAddNotePayload,  
    IEditNotePayload,  
    IChangeStatusPayload,  
    ACTIONS,  
    IAction  
} from '../common/types';  
  
export const editNote = ({ id, newNote }: IEditNotePayload): IAction =>  
{  
    type: ACTIONS.EDIT_NOTE,  
    payload: {  
        id,  
        newNote  
    }  
};
```

Приимает payload

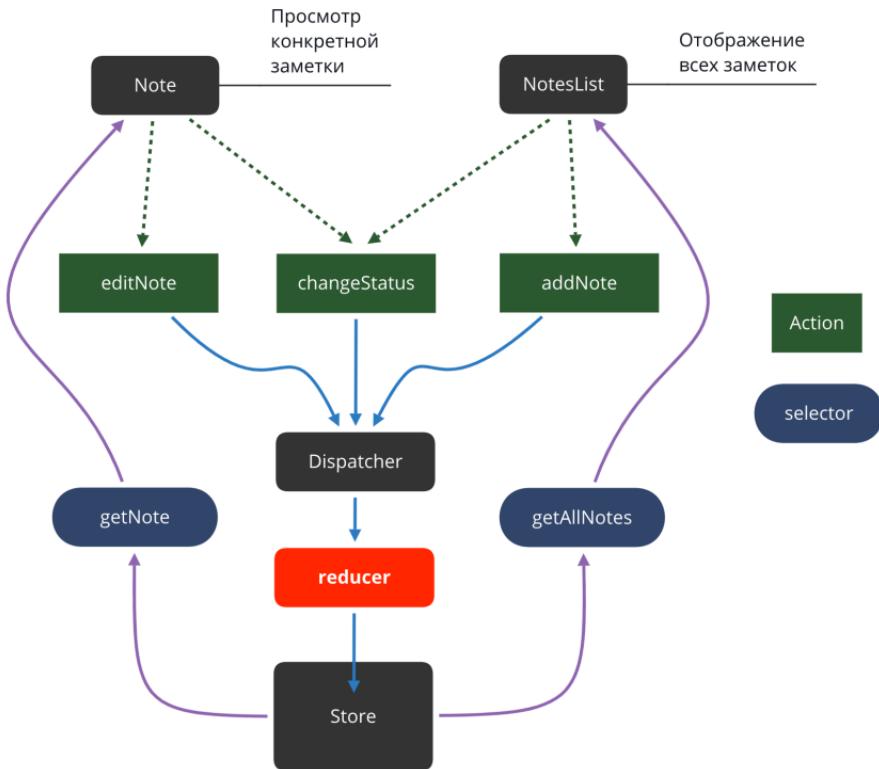
/src/store/actions.ts

```
import {  
    IAddNotePayload,  
    IEditNotePayload,  
    IChangeStatusPayload,  
    ACTIONS,  
    IAction  
} from '../common/types';  
  
export const editNote = ({ id, newNote }: IEditNotePayload): IAction =>  
{  
    type: ACTIONS.EDIT_NOTE,  
    payload: {  
        id,  
        newNote  
    }  
};
```

reducers

```
src
  common
    consts.ts
    mocks.ts
    types.ts
  components
    note
      index.css
      index.tsx
    notes-list
      index.css
      index.tsx
  store
    actions.ts
    index.ts
    reducers.ts reducers.ts
    selectors.ts
    App.test.tsx
    App.tsx
```

reducers



reducers

/src/store/reducers.ts

```
import { ..., IEditNotePayload, IAction } from '../common/types';
...
// состояние Store при инициализации приложения
export const initialState = { notes: initialNotes };

const notesReducer = (
    state: ApplicationState = initialState,
    action: IAction
) => {
    ...
    switch (action.type) {
        case ACTIONS.EDIT_NOTE:
            ...
    }
}
```

В аргументах приходит тот самый action

/src/store/reducers.ts

```
import { ..., IEditNotePayload, IAction } from '../common/types';
...
// состояние Store при инициализации приложения
export const initialState = { notes: initialNotes };

const notesReducer = (
    state: ApplicationState = initialState,
    action: IAction
) => {
    ...
    switch (action.type) {
        case ACTIONS.EDIT_NOTE:
            ...
    }
}
```

Действие определяется по типу action'a

/src/store/reducers.ts

```
import { ..., ACTIONS, IAction } from '../common/types';
...
// состояние Store при инициализации приложения
export const initialState = { notes: initialNotes };

const notesReducer = (
    state: ApplicationState = initialState,
    action: IAction
) => {
    ...
    switch (action.type) {
        case ACTIONS.EDIT_NOTE:
            ...
    }
}
```

Нужно задавать начальное состояние

/src/store/reducers.ts

```
import { ..., IEditNotePayload, IAction } from '../common/types';
...
// состояние Store при инициализации приложения
export const initialState = { notes: initialNotes };

const notesReducer = (
    state: ApplicationState = initialState,
    action: IAction
) => {
    ...
    switch (action.type) {
        case ACTIONS.EDIT_NOTE:
            ...
    }
}
```

Один из case'ов подробнее:

/src/store/reducers.ts

```
import { ApplicationState } from './index';
...
case ACTIONS.EDIT_NOTE: {
    const { id, newNote } = action.payload;
    const noteWithId = { ...newNote, id };

    const noteIndex = state.notes.findIndex(
        (note: INote) => note.id === id
    );

    return {
        ...state,
        notes: [...state.notes].splice(noteIndex, 1, noteWithId);
    }
}
```

reducer возвращает новый Store:

/src/store/reducers.ts

```
import { ApplicationState } from './index';
...
case ACTIONS.EDIT_NOTE: {
    const { id, newNote } = action.payload;
    const noteWithId = { ...newNote, id };

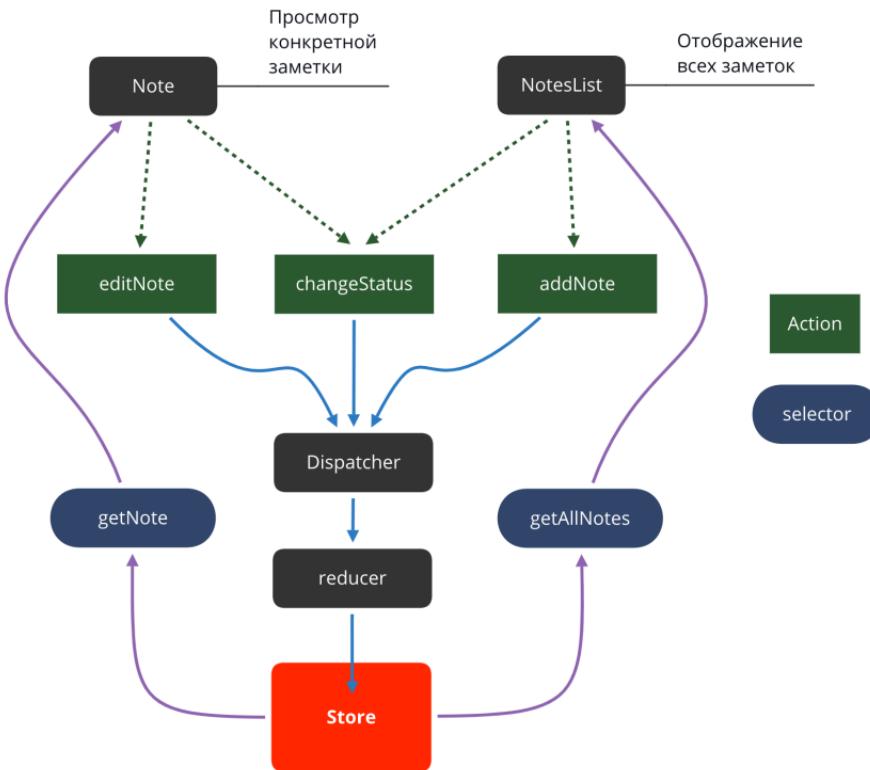
    const noteIndex = state.notes.findIndex(
        (note: INote) => note.id === id
    );

    return {
        ...state,
        notes: [...state.notes].splice(noteIndex, 1, noteWithId);
    }
}
```

Инициализация Store

```
src
  common
    consts.ts
    mocks.ts
    types.ts
  components
    note
      index.css
      index.tsx
    notes-list
      index.css
      index.tsx
  store
    actions.ts
    index.ts index.ts
    reducers.ts
    selectors.ts
    App.test.tsx
    App.tsx
```

Инициализация Store



Инициализация Store

/src/store/index.ts

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

import { INote } from '../common/types';
import reducer, { initialState } from './reducers';

export interface ApplicationState {
    notes: INote[];
}

export default createStore(
    reducer,
    initialState as any
);
```

Передаем reducers

/src/store/index.ts

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

import { INote } from '../common/types';
import reducer, { initialState } from './reducers';

export interface ApplicationState {
    notes: INote[];
}

export default createStore(
    reducer,
    initialState as any
);
```

Значение Store по умолчанию

/src/store/index.ts

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

import { INote } from '../common/types';
import reducer, { initialState } from './reducers';

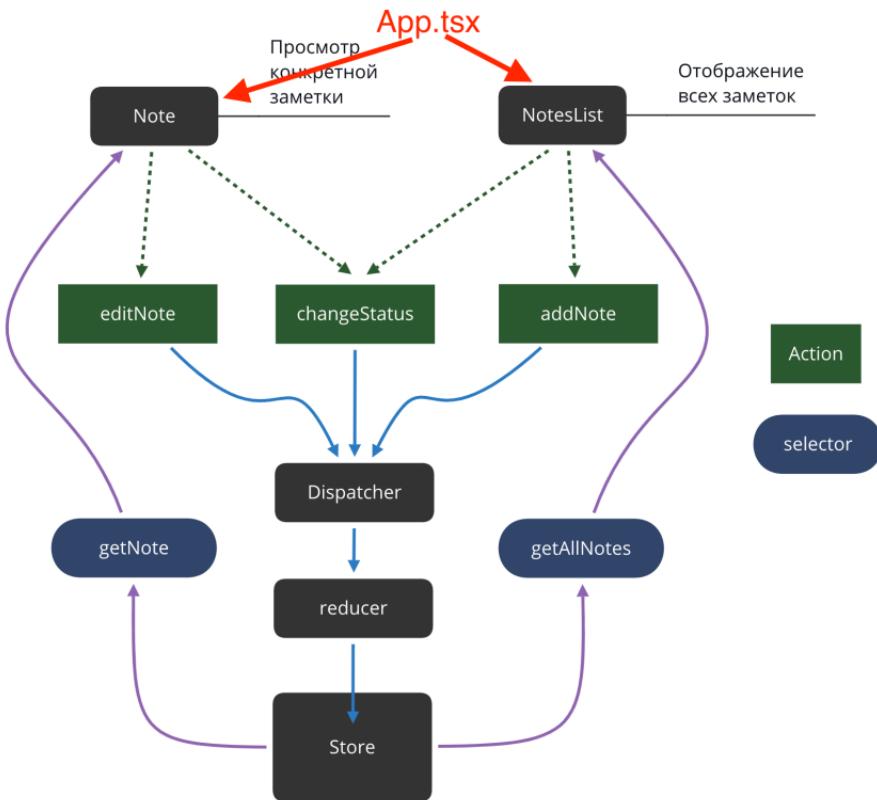
export interface ApplicationState {
    notes: INote[];
}

export default createStore(
    reducer,
    initialState as any
);
```

Как подключить к React?

```
src
  common
    consts.ts
    mocks.ts
    types.ts
  components
    note
      index.css
      index.tsx
    notes-list
      index.css
      index.tsx
  store
    actions.ts
    index.ts
    reducers.ts
    selectors.ts
  App.test.tsx
  App.tsx
```

Как подключить к React?



Как подключить к React?

/src/App.tsx

```
import React, { Component } from 'react';
import { Provider } from 'react-redux';
import { Route, Switch } from 'react-router-dom';
import store from './store';

export default class App extends Component {
  public render() {
    return (
      <Provider store={store}>
        <div className="App">
          <Switch>
            <Route path="/note/:id" component={Note} />
            <Route path="/" component={NotesList} />
          </Switch>
        </div>
      </Provider>
    );
  }
}
```

Ликбез: react-router

/src/App.tsx

```
import React, { Component } from 'react';
import { Provider } from 'react-redux';
import { Route, Switch } from 'react-router-dom';
import store from './store';

export default class App extends Component {
  public render() {
    return (
      <Provider store={store}>
        <div className="App">
          <Switch>
            <Route path="/note/:id" component={Note} />
            <Route path="/" component={NotesList} />
          </Switch>
        </div>
      </Provider>
    );
  }
}
```

На самом деле обычные
роуты, на которые будет
реагировать React

Вернемся к redux

/src/App.tsx

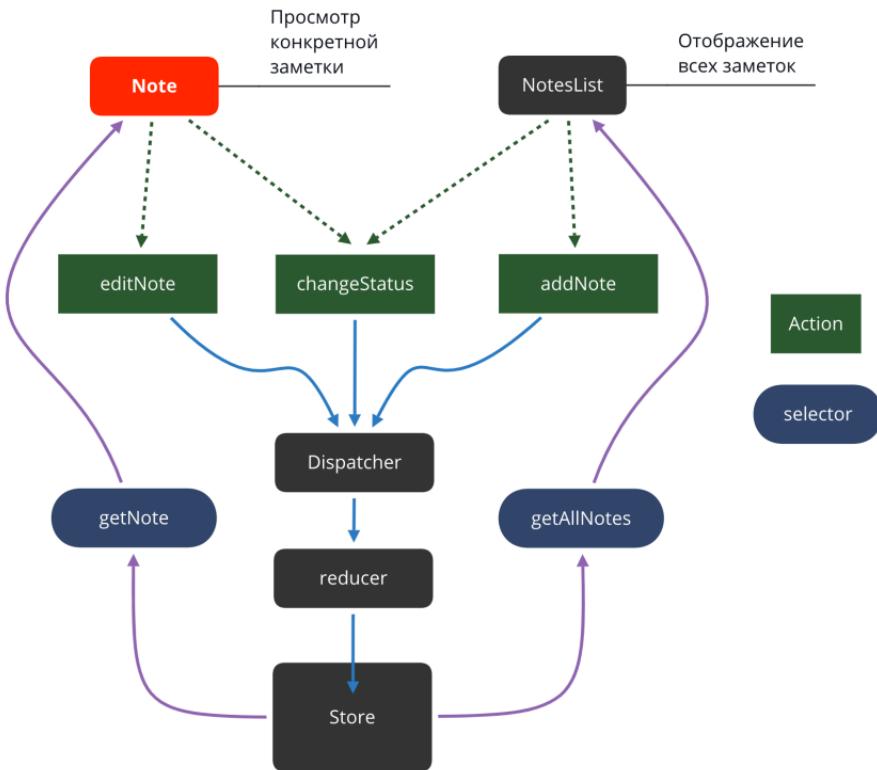
```
import React, { Component } from 'react';
import { Provider } from 'react-redux';
import { Route, Switch } from 'react-router-dom';
import store from './store';

export default class App extends Component {
    public render() {
        return (
            <Provider store={store}>
                <div className="App">
                    <Switch>
                        <Route path="/note/:id" component={Note} />
                        <Route path="/" component={NotesList} />
                    </Switch>
                </div>
            </Provider>
        );
    }
}
```

Работаем с redux в компонентах

```
src
  common
    consts.ts
    mocks.ts
    types.ts
  components
    note
      index.css
      index.tsx
    notes-list
      index.css
      index.tsx
  store
    actions.ts
    index.ts
    reducers.ts
    selectors.ts
  App.test.tsx
  App.tsx
```

Работаем с redux в компонентах



Работаем с redux в компонентах

/src/components/note/note.tsx

```
import { connect } from 'react-redux';
import { ApplicationState } from '../../store';
import { changeStatus, editNote } from '../../store/actions';
...

type StateProps = ReturnType<typeof mapStateToProps>;
type DispatchProps = { dispatch: (action: IAction) => void };
type Props = StateProps & DispatchProps & ...;

class Note extends Component<Props, IOwnState> {
    ...
}

const mapStateToProps = (state: ApplicationState, props: OwnProps) => ({
    currentNote: state.notes.find(note => note.id === props.id);
});

export default connect(mapStateToProps)(Note);
```

connect()()

/src/components/note/note.tsx

```
import { connect } from 'react-redux';
import { ApplicationState } from '../.../store';
import { changeStatus, editNote } from '../.../store/actions';
...

type StateProps = ReturnType<typeof mapStateToProps>;
type DispatchProps = { dispatch: (action: IAction) => void };
type Props = StateProps & DispatchProps & ...;

class Note extends Component<Props, IOwnState> {
    ...
}

const mapStateToProps = (state: ApplicationState, props: OwnProps) => ({
    currentNote: state.notes.find(note => note.id === props.id);
});

export default connect(mapStateToProps)(Note);
```

connect(...)(Note) возвращает новый компонент, который имеет доступ в Store

mapStateToProps

/src/components/note/note.tsx

```
import { connect } from 'react-redux';
import { ApplicationState } from '../../store';
import { changeStatus, editNote } from '../../store/actions';
...
type StateProps = ReturnType<typeof mapStateToProps>;
type DispatchProps = { dispatch: (action: IAction) =>
type Props = StateProps & DispatchProps & ...;

class Note extends Component<Props, IOwnState> {
    ...
}

const mapStateToProps = (state: ApplicationState, props: OwnProps) => ({
    currentNote: state.notes.find(note => note.id === props.id);
});

export default connect(mapStateToProps)(Note);
```

mapStateToProps передается первым аргументом в connect(...) и подкладывается в this.props поля из Store

Как получить доступ к Store?

/src/components/note/note.tsx

```
...
type StateProps = ReturnType<typeof mapStateToProps>;
type Props = StateProps & DispatchProps;
class Note extends Component<Props, IOwnState> {
    ...
    public render() {
        return (
            ...
            <div className="note-page__field">
                Дата создания
                <div className="note-page__created-date">
                    {this.props.currentNote.created}
                </div>
            </div>
            ...
        )
    }
}
```

Данные будут подложены в this.props

/src/components/note/note.tsx

```
...
type StateProps = ReturnType<typeof mapStateToProps>;
type Props = StateProps & DispatchProps;
class Note extends Component<Props, IOwnState> {
    ...
    public render() {
        return (
            ...
            <div className="note-page__field">
                Дата создания
                <div className="note-page__created-date">
                    {this.props.currentNote.created}
                </div>
            </div>
            ...
        )
    }
}
```

Как вызвать событие?

/src/components/note/note.tsx

```
...
type DispatchProps = { dispatch: (action: IAction) => void };
type Props = StateProps & DispatchProps;
class Note extends Component<Props, IOwnState> {
  ...
  public onEditButtonClick = (e: ReactMouseEvent) => {
    ...
    // Кладем в currentNote данные об отредактированной заметке
    const currentNote = ...;

    this.props.dispatch({
      type: ACTIONS.EDIT_NOTE,
      payload: { id, newNote: currentNote }
    });
  };
  ...
}
```

Генерируем action прямо в dispatch

/src/components/note/note.tsx

```
...
type DispatchProps = { dispatch: (action: IAction) => void };
type Props = StateProps & DispatchProps;
class Note extends Component<Props, IOwnState> {
    ...
    public onEditButtonClick = (e: ReactMouseEvent) => {
        ...
        // Кладем в currentNote данные об отредактированной заметке
        const currentNote = ...;

        this.props.dispatch({
            type: ACTIONS.EDIT_NOTE,
            payload: { id, newNote: currentNote }
        });
    };
    ...
}
```

Используем actionCreators

/src/components/note/note.tsx

```
import { editNote } from '../../store/actions';
...
type DispatchProps = { dispatch: (action: IAction) => void };
type Props = StateProps & DispatchProps;
class Note extends Component<Props, IOwnState> {
    ...
    public onEditButtonClick = (e: ReactMouseEvent) => {
        ...
        // Кладем в currentNote данные об отредактированной заметке
        const currentNote = ...;

        this.props.dispatch(editNote({ id, newNote: currentNote }));
    };
    ...
}
```

Используем actionCreators

/src/components/note/note.tsx

```
import { editNote } from '../../store/actions';
...
type DispatchProps = { dispatch: (action: IAction) => void };
type Props = StateProps & DispatchProps;
class Note extends Component<Props, IOwnState> {
    ...
    public onEditButtonClick = (e: ReactMouseEvent) => {
        ...
        // Кладем в currentNote данные об отредактированной заметке
        const currentNote = ...;

        this.props.dispatch(editNote({ id, newNote: currentNote }));
    };
    ...
}
```

Dispatch'им событие из компонента

/src/components/note/note.tsx

```
import { editNote } from '../../store/actions';
...
type DispatchProps = { dispatch: (action: IAction) => void };
type Props = StateProps & DispatchProps;
class Note extends Component<Props, IOwnState> {
    ...
    public onEditButtonClick = (e: ReactMouseEvent) => {
        ...
        // Кладем в currentNote данные об отредактированной заметке
        const currentNote = ...;

        this.props.dispatch(editNote({ id: [56], newNote: [ЗАМЕТКА] }));
    };
    ...
}
```

actionCreator возвращает данные с типом action'a

/src/store/actions.ts

```
import {
    IAddNotePayload,
    IEditNotePayload,
    IChangeStatusPayload,
    ACTIONS,
    IAction
} from '../common/types';

const editNote = ({ id: [56], newNote: [ЗАМЕТКА] }: IEditNotePayload) =>
{
    type: ACTIONS.EDIT_NOTE // 'EDIT_NOTE',
    payload: {
        id: [56],
        newNote: [ЗАМЕТКА]
    }
};

...
```

Принимаем данные в reducer'e:

/src/store/reducers.ts

```
import { ApplicationState } from './index';
...
case ACTIONS.EDIT_NOTE: { // 'editNote'
  const { id: [56], newNote: [ЗАМЕТКА] } = action.payload;

  // noteWithId - это [ЗАМЕТКА №56]
  const noteWithId = { ...newNote [ЗАМЕТКА], id [56] };

  // находим идентификатор редактируемой заметки
  const index = state.notes.findIndex((note: INote) => note.id === id);

  // копируем массив заметок и вставляем на место старой новую
  return {
    ...state,
    notes: [...state.notes].splice(index, 1, noteWithId: [ЗАМЕТКА №56]);
  };
}
...
```

Кладем Store в компонент

/src/components/note/note.tsx

...

```
type StateProps = ReturnType<typeof mapStateToProps>;
type DispatchProps = { dispatch: (action: IAction) => void };
type Props = StateProps & DispatchProps & ...;

class Note extends Component<Props, IOwnState> {
  ...
}

const mapStateToProps = (state: ApplicationState, props: OwnProps) => ({
  currentNote: state.notes.find(note => note.id === id: [56]);
  // Теперь this.props.currentNote - это [ЗАМЕТКА №56]
});

export default connect(mapStateToProps)(Note);
```

Получаем Store из this.props

/src/components/note/note.tsx

```
...
type StateProps = ReturnType<typeof mapStateToProps>;
type Props = StateProps & DispatchProps; // подкладываем StateProps в this.prop
class Note extends Component<Props, IOwnState> {
    ...
    public render() {
        return (
            ...
            <div className="note-page__field">
                Дата создания
                <div className="note-page__created-date">
                    {this.props.currentNote.created} // это [ЗАМЕТКА №56]
                </div>
            </div>
            ...
        )
    }
}
```

Получаем Store из this.props

/src/components/note/note.tsx

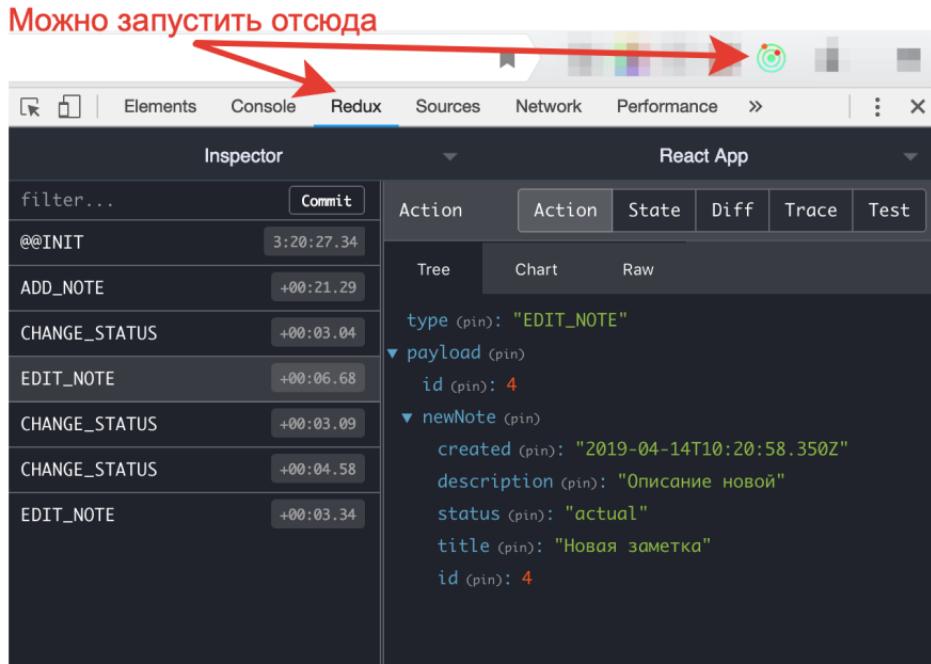
```
...
type StateProps = ReturnType<typeof mapStateToProps>;
type Props = StateProps & DispatchProps; // подкладываем StateProps в this.prop
class Note extends Component<Props, IOwnState> {
    ...
    public render() {
        return (
            ...
            <div className="note-page__field">
                Дата создания
                <div className="note-page__created-date">
                    {this.props.currentNote.created} // это [ЗАМЕТКА №56]
                </div>
            </div>
            ...
        )
    }
}
```

Redux DevTools

- Ссылка на документацию и мануал по установке
- Позволяет следить за Store в режиме дебага

Redux DevTools

- Запуск



Redux DevTools

- Отображается вся история action'ов для экземпляра приложения

The screenshot shows the Redux DevTools extension integrated into the Chrome developer tools. The interface has a top navigation bar with tabs for Elements, Console, Redux, Sources, Network, Performance, and more. The 'Redux' tab is active.

The left sidebar, titled 'Inspector', contains a 'filter...' input field and a list of actions:

- @@INIT 3:20:27.34
- ADD_NOTE +00:21.29
- CHANGE_STATUS +00:03.04
- EDIT_NOTE +00:06.68
- CHANGE_STATUS +00:03.09
- CHANGE_STATUS +00:04.58
- EDIT_NOTE +00:03.34

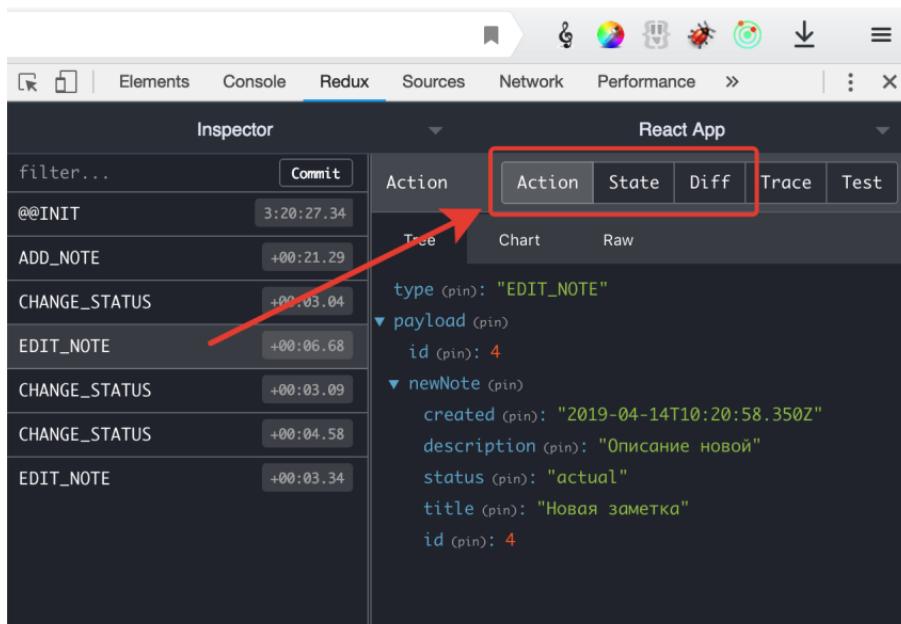
A red arrow points from the bottom of this sidebar to the text 'История всех действий со Store' (History of all actions from Store) at the bottom of the sidebar area.

The main panel is titled 'React App' and shows the state tree for the latest action. The action object is expanded, showing its type, payload, and newNote properties. The payload contains an id of 4. The newNote object includes created, description, status, title, and id properties.

Action	Action	State	Diff	Trace	Test
type (pin): "EDIT_NOTE"					
payload (pin)					
id (pin): 4					
newNote (pin)					
created (pin): "2019-04-14T10:20:58.350Z"					
description (pin): "Описание новой"					
status (pin): "actual"					
title (pin): "Новая заметка"					
id (pin): 4					

Redux DevTools

- Для каждого action'a (или даже нескольких) можно посмотреть параметры action'a, состояние store после выполнения и diff стора



Redux DevTools

The screenshot shows the Redux DevTools extension integrated into the Chrome developer tools interface. The top navigation bar includes tabs for Elements, Console, Redux (which is selected), Sources, Network, Performance, and more. Below the tabs is a toolbar with various icons. The main area is divided into two sections: 'Inspector' on the left and 'React App' on the right.

Inspector Panel:

- Contains a 'filter...' input field and a 'Commit' button.
- A list of actions with their types and execution times:
 - @@INIT (3:20:27.34)
 - ADD_NOTE (+00:21.29)
 - CHANGE_STATUS (+00:03.04)
 - EDIT_NOTE (+00:06.68)
 - CHANGE_STATUS (+00:03.09)
 - CHANGE_STATUS (+00:04.58)
 - EDIT_NOTE (+00:03.34)

React App Panel:

- A table with columns: Action, Action, State, Diff, Trace, Test.
- Below the table are three tabs: Tree (selected), Chart, and Raw.
- A detailed view of the most recent action (EDIT_NOTE):
 - type** (pin): "EDIT_NOTE"
 - payload** (pin):
 - id** (pin): 4
 - newNote** (pin):
 - created** (pin): "2019-04-14T10:20:58.350Z"
 - description** (pin): "Описание новой"
 - status** (pin): "actual"
 - title** (pin): "Новая заметка"
 - id** (pin): 4

Page Number: 89

Redux DevTools

Elements Console Redux Sources Network Performance >⋮×

Inspector React App

filter... Commit

@@INIT 3:20:27.34

ADD_NOTE +00:21.29

CHANGE_STATUS +00:03.04

EDIT_NOTE +00:06.68

CHANGE_STATUS +00:03.09

CHANGE_STATUS +00:04.58

EDIT_NOTE +00:03.34

State Action State Diff Trace Test

Tree Chart Raw

notes (pin)

0 (pin): { created: "2019-04-09...", description: ""},
1 (pin): { created: "2019-04-03T16:59:59.834Z",
description: "Не забыть показать Redux DevTools",
id: 2, status: "actual", title: "Прочитать лекцию про Redux"},
2 (pin): { created: "2019-01-22...", description: "//"},
3 (pin): { created: "2019-04-14...", description: "On"}

Redux DevTools

The screenshot shows the Redux DevTools extension integrated into a browser's developer tools. The top navigation bar includes tabs for Elements, Console, Redux (which is selected), Sources, Network, Performance, and more. Below the tabs is a toolbar with icons for bookmarking, sharing, and other developer features.

The main interface is divided into two sections: "Inspector" on the left and "React App" on the right. The Inspector section contains a list of actions with their names and execution times. The React App section shows the state tree for the "notes" key.

Inspector:

- filter...
- Commit
- @@INIT 3:20:27.34
- ADD_NOTE +00:21.29
- CHANGE_STATUS +00:03.04
- EDIT_NOTE +00:06.68
- CHANGE_STATUS +00:03.09
- CHANGE_STATUS +00:04.58
- EDIT_NOTE +00:03.34

React App:

Diff Action State Diff Trace Test

Tree Raw

notes (pin)

3 (pin)

- created (pin): '2019-04-14T10:20:48.630Z' => '2019-04-14T10:20:58.350Z'
- description (pin): 'Описание новой заметки' => 'Описание новой'

Идеальная библиотека?

Да, но:

1. В Store теперь хочется складывать вообще все
2. В redux'e store не знает о том, какие его части нужны для каждой конкретной view и пересчитывает все в функциях, которые прорасыпают store (mapStateToProps)
3. Нет асинхронности:(
4. Слишком много писать

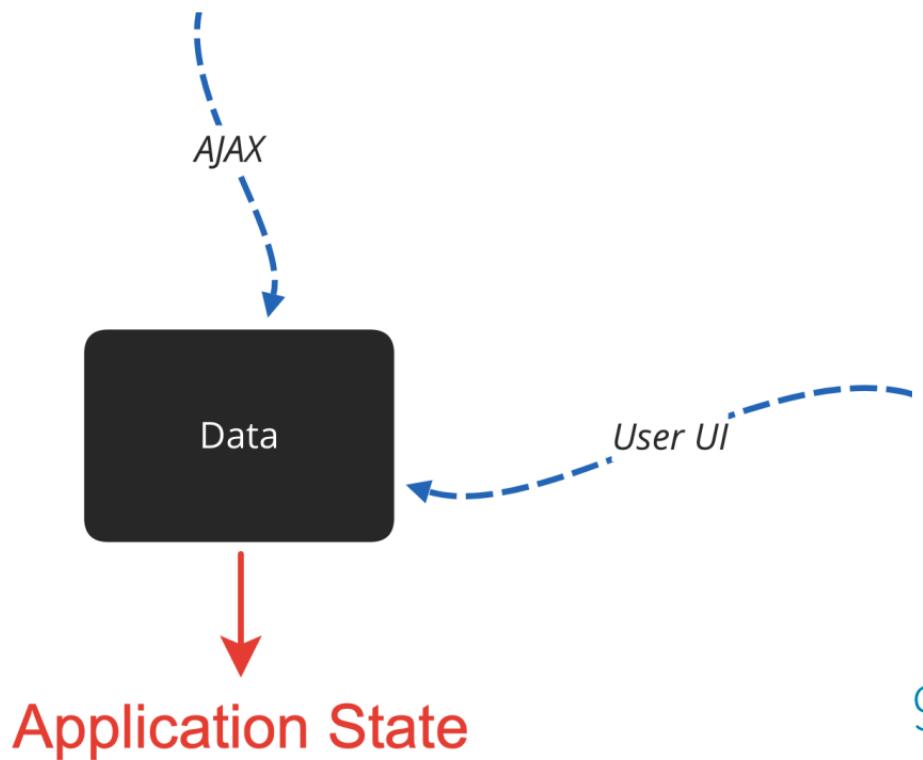
Проблема 1. Что делать, если Store стал
одной большой помойкой

Куда класть данные и когда использовать Application State?

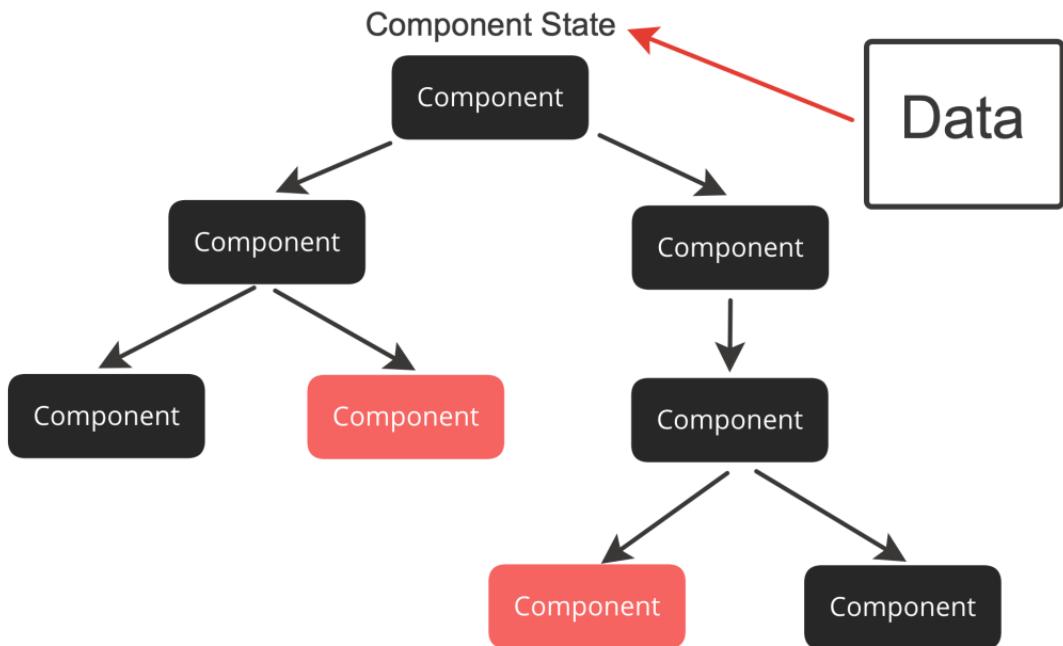
Данные могут относиться к **3 категориям**:

- Данные, которые могут поменяться из нескольких мест
- Данные, которые нужны в нескольких несвязанных частях приложения (компонентах)
- Начальные данные для приложения

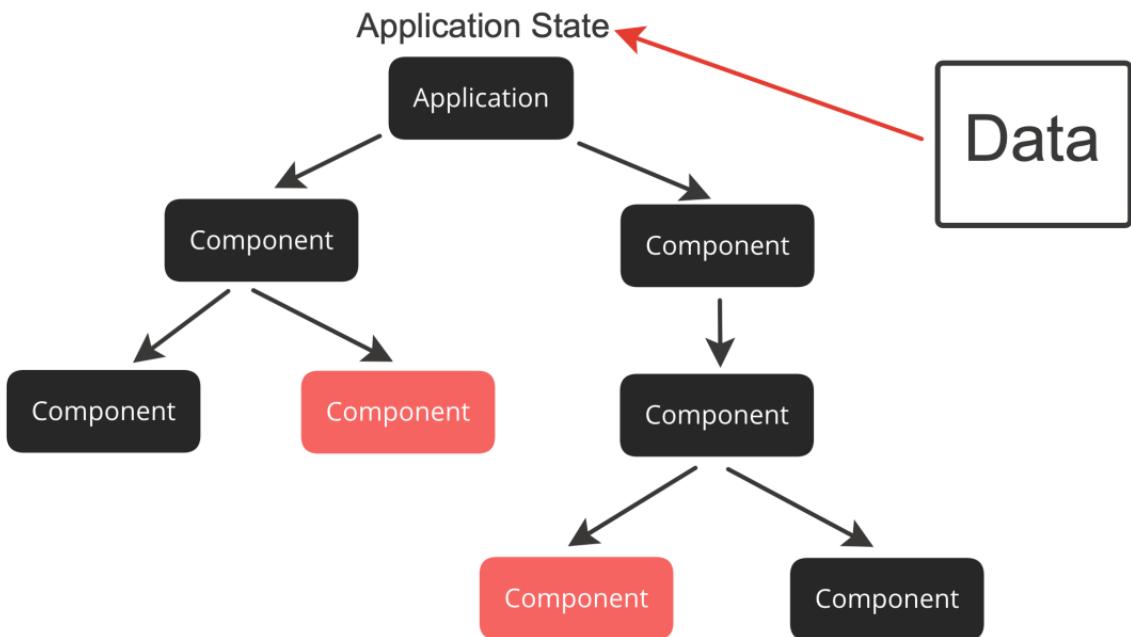
Данные, которые могут поменяться из
нескольких мест



Данные, которые нужны в нескольких несвязанных частях приложения
(компонентах)



Данные, которые нужны в нескольких несвязанных частях приложения
(компонентах)



Начальные данные для приложения

Речь о данных, которые пришли с сервера: переводы, какие-то настройки, и т.д.

- Хорошо бы, чтобы эти данные хранились в одном месте...
- И чтобы достать их можно было из любого компонента...

Проблема 2. Как избежать rerender'a во всех местах?

Хорошо бы, чтобы пересчитывались
данные только для тех View, данные для
которых реально поменялись...

Библиотека Reselect!

Чтобы было сложнее, добавим новую

фичу

```
▽ src
  ▽ common
    TS consts.ts
    TS mocks.ts
    TS types.ts
  ▽ components
    ▽ last-status
      |   ⚙ index.tsx
    ▽ note
      # index.css
      ⚙ index.tsx
    ▽ notes-list
      # index.css
      ⚙ index.tsx
  ▽ store
    TS actions.ts
    TS index.ts
    TS reducers.ts
    TS selectors.ts
  ⚙ App.tsx
```

Чтобы было сложнее, добавим новую фичу

/src/components/last-status/index.tsx

```
import { getLastStatus } from '../../store/selectors';
...
type StateProps = ReturnType<typeof mapStateToProps>

const LastStatus = ({ lastStatus }: StateProps) => {
    return (
        <h2>Статус последней заметки: {lastStatus.toString()}</h2>
    )
}

function mapStateToProps(state: ApplicationState) {
    return {
        lastStatus: getLastStatus(state)
    };
}
```

Чтобы было сложнее, добавим новую
фиичу

/src/components/last-status/index.tsx

```
import { getLastStatus } from '../../store/selectors';
...
type StateProps = ReturnType<typeof mapStateToProps>

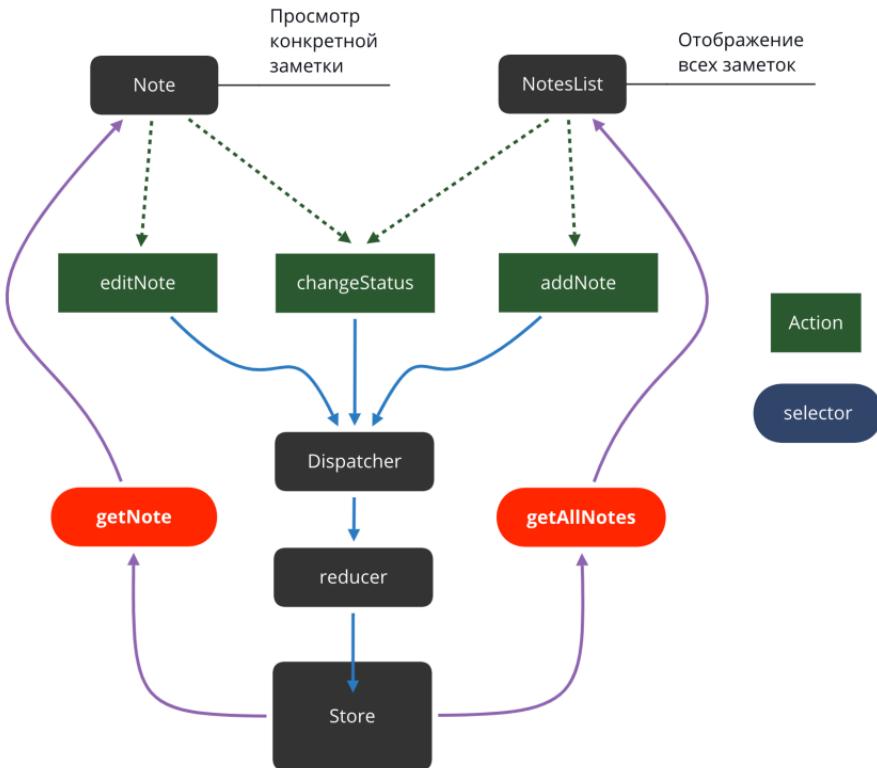
const LastStatus = ({ lastStatus }: StateProps) => {
    return (
        <h2>Статус последней заметки: {lastStatus.toString()}</h2>
    )
}

function mapStateToProps(state: ApplicationState) {
    return {
        lastStatus: getLastStatus(state)
    };
}
```

Про селекторы

```
src
  common
    consts.ts
    mocks.ts
    types.ts
  components
    note
      index.css
      index.tsx
    notes-list
      index.css
      index.tsx
  store
    actions.ts
    index.ts
    reducers.ts
    selectors.ts
  App.test.tsx
  App.tsx
```

Про селекторы



Как выглядят селекторы?

/src/store/selectors.ts

```
import { ApplicationState } from './index';

export function getNoteById(state: ApplicationState, id: string) {
  const { notes } = state;

  return notes.find(note => note.id === parseInt(id, 10)) ||
    notes[notes.length - 1];
}

export function getAllNotes(state: ApplicationState) {
  return state.notes;
}

export function getLastStatus({ notes }: ApplicationState) {
  return notes[notes.length - 1].status;
}
```

Селектор может быть простой

/src/store/selectors.ts

```
import { ApplicationState } from './index';

export function getNoteById(state: ApplicationState, id: string) {
  const { notes } = state;

  return notes.find(note => note.id === parseInt(id, 10)) ||
    notes[notes.length - 1];
}

export function getAllNotes(state: ApplicationState) {
  return state.notes;
}

export function getLastStatus({ notes }: ApplicationState) {
  return notes[notes.length - 1].status;
}
```

Может содержать логику

/src/store/selectors.ts

```
import { ApplicationState } from './index';

export function getNoteById(state: ApplicationState, id: string) {
  const { notes } = state;

  return notes.find(note => note.id === parseInt(id, 10)) ||
    notes[notes.length - 1];
}

export function getAllNotes(state: ApplicationState) {
  return state.notes;
}

export function getLastStatus({ notes }: ApplicationState) {
  return notes[notes.length - 1].status;
}
```

Селекторы можно комбинировать

/src/store/selectors.ts

```
import { ApplicationState } from './index';

export function getNoteById(state: ApplicationState, id: string) {
  const notes = getAllNotes(state);

  return notes.find(note => note.id === parseInt(id, 10)) ||
    notes[notes.length - 1];
}

export function getAllNotes(state: ApplicationState) {
  return state.notes;
}

export function getLastStatus({ notes }: ApplicationState) {
  return notes[notes.length - 1].status;
}
```

Оптимизируем селекторы

/src/store/selectors.ts

```
import { createSelector } from 'reselect'
import { ApplicationState } from './index';
...
export function getLastNote(state: ApplicationState) {
  const { notes } = state;

  return notes[notes.length - 1]
}

export const getLastStatus = createSelector(
  [getLastNote],
  (lastNote: INote) => {
    return lastNote.status;
}
)
```

Оптимизируем селекторы

/src/store/selectors.ts

```
import { createSelector } from 'reselect'
import { ApplicationState } from './index';
...

export function getLastNote(state: ApplicationState) {
    const { notes } = state;

    return notes[notes.length - 1]
}

export const getLastStatus = createSelector(
    [getLastNote],
    (lastNote: INote) => {
        return lastNote.status;
    }
)
```

Оптимизируем селекторы

/src/store/selectors.ts

```
import { createSelector } from 'reselect'
import { ApplicationState } from './index';
...

export function getLastNote(state: ApplicationState) {
    const { notes } = state;

    return notes[notes.length - 1]
}

export const getLastStatus = createSelector(
    [getLastNote],
    (lastNote: INote) => {
        return lastNote.status;
    }
)
```

Оптимизируем селекторы

/src/store/selectors.ts

```
import { createSelector } from 'reselect'
import { ApplicationState } from './index';
...
export function getLastNote(state: ApplicationState) {
    const { notes } = state;

    return notes[notes.length - 1]
}

export const getLastStatus = createSelector(
    [getLastNote],
    (lastNote: INote) => {
        return lastNote.status;
    }
)
```

Оптимизируем селекторы

/src/store/selectors.ts

```
import { createSelector } from 'reselect'
import { ApplicationState } from './index';
...
export function getLastNote(state: ApplicationState) {
  const { notes } = state;

  return notes[notes.length - 1]
}

export const getLastStatus = createSelector(
  [getLastNote],
  (lastNote: INote) => {
    return lastNote.status;
}
)
```

Нужный селектор
пересчитывается только тогда,
когда изменяются данные,
которые приходят из
зависимых селекторов

Reselect: итоги

Зачем:

- "If the state tree is large, or the calculation expensive, repeating the calculation on every update may cause performance problems. Reselect can help to avoid these unnecessary recalculations."

Что можно еще:

- Можно передавать несколько зависимых селекторов и пользоваться всеми возвращаемыми данными
- Можно отдельно передавать функцию сравнения для определения необходимости пересчета
- Еще много функционала, можно почитать в документации

Проблема 3. Асинхронность?

[Async Flow](#): документация redux

Библиотеки для работы с асинхронностью в Redux:

- [redux-thunk](#) (рекомендуется документацией redux)
- [redux-promise](#)
- [redux-saga](#)
- [redux-observable](#)

Проблема 3. Асинхронность?

Async Flow: документация redux

Библиотеки для работы с асинхронностью в Redux:

- redux-thunk (рекомендуется документацией redux)
- redux-promise
- redux-saga
- redux-observable

redux-thunk

- Простая библиотека для работы с side-effect в redux.
- Разрешает вам писать ваши action creators так, чтобы они возвращали функцию, а не action
- Можно использовать как для асинхронных действий, так и для более специфичных случаев (например, условного срабатывания события)

Инициализация redux-thunk

```
src
  common
    consts.ts
    mocks.ts
    types.ts
  components
    note
      index.css
      index.tsx
    notes-list
      index.css
      index.tsx
  store
    actions.ts
    index.ts index.ts
    reducers.ts
    selectors.ts
    App.test.tsx
    App.tsx
```

Было

/src/store/index.ts

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

import { INote } from '../common/types';
import reducer, { initialState } from './reducers';

export interface ApplicationState {
    notes: INote[];
}

export default createStore(
    reducer,
    initialState as any
);
```

Стало

/src/store/index.ts

```
import { createStore } from 'redux';
import thunk from 'redux-thunk';

import { INote } from '../common/types';
import reducer, { initialState } from './reducers';

export interface ApplicationState {
  notes: INote[];
}

export default createStore(
  reducer,
  initialState as any,
```

Добавилась мидлвара thunk

/src/store/index.ts

```
import { createStore } from 'redux';
import thunk from 'redux-thunk';

import { INote } from '../common/types';
import reducer, { initialState } from './reducers';

export interface ApplicationState {
  notes: INote[];
}

export default createStore(
  reducer,
  initialState as any,
```

Выводим данные в редьюсере

```
src
  common
    consts.ts
    mocks.ts
    types.ts
  components
    note
      index.css
      index.tsx
    notes-list
      index.css
      index.tsx
  store
    actions.ts
    index.ts
    reducers.ts
    selectors.ts
    App.test.tsx
    App.tsx
```

Было

/src/common/reducers.ts

```
case ACTIONS.ADD_NOTE: {
  const {
    note: newPureNote
  } = payload as IAddNotePayload;

  const id = Math.max(...notes.map((note: INote) => note.id)) + 1;

  const newNote: INote = {
    ...newPureNote,
    id
  };

  return { ...state, notes: [...notes, newNote] };
}
```

Стало

/src/common/reducers.ts

```
case ACTIONS.ADD_NOTE: {
  const {
    note: newPureNote,
    randomAsyncNumber
  } = payload as ExtendRandomNumber<IAddNotePayload>;
  const id = Math.max(...notes.map((note: INote) => note.id)) + 1;
  const newNote: INote = {
    ...newPureNote,
    id,
    randomAsyncNumber
  };
  return { ...state, notes: [...notes, newNote] };
}
```

Стало

/src/common/reducers.ts

```
case ACTIONS.ADD_NOTE: {
  const {
    note: newPureNote,
    randomAsyncNumber
  } = payload as ExtendRandomNumber<IAddNotePayload>;
  const id = Math.max(...notes.map((note: INote) => note.id)) + 1;
  const newNote: INote = {
    ...newPureNote,
    id,
    randomAsyncNumber
  };
  return { ...state, notes: [...notes, newNote] };
}
```

Создаем асинхронные actions

```
src
  common
    consts.ts
    mocks.ts
    types.ts
  components
    note
      index.css
      index.tsx
    notes-list
      index.css
      index.tsx
  store
    actions.ts
    index.ts
    reducers.ts
    selectors.ts
    App.test.tsx
    App.tsx
```

Было

/src/common/actions.ts

```
export const addNote = ({ note }: IAddNotePayload) => ({
  type: ACTIONS.ADD_NOTE,
  payload: {
    note
  }
});
```

Стало

/src/common/actions.ts

```
export const addNoteRequest = ({ note }: IAddNotePayload) => {
  return (dispatch: Dispatch) => {

    setTimeout(() => {
      const randomAsyncNumber = Math.ceil(Math.random() * 100);

      dispatch(addNoteSuccess({ note, randomAsyncNumber }))
    }, 3000)
  }
}

export const addNoteSuccess = ({
  note,
  randomAsyncNumber
}: ExtendRandomNumber<IAddNotePayload>) => ({
  type: ACTIONS.ADD_NOTE,
  payload: { note, randomAsyncNumber }
});
```

Action creator возвращает функцию

/src/common/actions.ts

```
export const addNoteRequest = ({ note }: IAddNotePayload) => {
  return (dispatch: Dispatch) => {
    setTimeout(() => {
      const randomAsyncNumber = Math.ceil(Math.random() * 100);

      dispatch(addNoteSuccess({ note, randomAsyncNumber }))
    }, 3000)
  }
}

export const addNoteSuccess = ({
  note,
  randomAsyncNumber
}: ExtendRandomNumber<IAddNotePayload>) => ({
  type: ACTIONS.ADD_NOTE,
  payload: { note, randomAsyncNumber }
});
```

Функция принимает dispatch и вызывает его

/src/common/actions.ts

```
export const addNoteRequest = ({ note }: IAddNotePayload) => {
  return (dispatch: Dispatch) => {
    setTimeout(() => {
      const randomAsyncNumber = Math.ceil(Math.random() * 100);

      dispatch(addNoteSuccess({ note, randomAsyncNumber }))
    }, 3000)
  }
}

export const addNoteSuccess = ({
  note,
  randomAsyncNumber
}: ExtendRandomNumber<IAddNotePayload>) => ({
  type: ACTIONS.ADD_NOTE,
  payload: { note, randomAsyncNumber }
});
```

В dispatch() передается другой action

/src/common/actions.ts

```
export const addNoteRequest = ({ note }: IAddNotePayload) => {
  return (dispatch: Dispatch) => {

    setTimeout(() => {
      const randomAsyncNumber = Math.ceil(Math.random() * 100);

      dispatch(addNoteSuccess({ note, randomAsyncNumber }))
    }, 3000)
  }
}

export const addNoteSuccess = ({
  note,
  randomAsyncNumber
}: ExtendRandomNumber<IAddNotePayload>) => ({
  type: ACTIONS.ADD_NOTE,
  payload: { note, randomAsyncNumber }
});
```

dispatch находится в замыкании и может быть
вызван когда угодно

/src/common/actions.ts

```
export const addNoteRequest = ({ note }: IAddNotePayload) => {
  return (dispatch: Dispatch) => {
    setTimeout(() => {
      const randomAsyncNumber = Math.ceil(Math.random() * 100);

      dispatch(addNoteSuccess({ note, randomAsyncNumber }))
    }, 3000)
  }
}

export const addNoteSuccess = ({
  note,
  randomAsyncNumber
}: ExtendRandomNumber<IAddNotePayload>) => ({
  type: ACTIONS.ADD_NOTE,
```

Проблема 4. Слишком много писать

"Но даже в мой дремучий лес кривой тропой пришел прогресс..."

- redux-toolkit

Redux Toolkit

Библиотека для простой и понятной организации вашего store, включает в себя:

- configureStore()
- createReducer()
- createAction()
- createSlice()
- createSelector()

configureStore()

- Позволяет сконфигурировать store, передав только reducers
- Можно передавать middlewares, preloadedState, enhancers, но не обязательно
- Расширенное использование: см. [документацию](#)

configureStore()

/src/store/index.ts

Применяется очень просто:

```
const store = configureStore({  
  reducer,  
  middleware,  
  devTools: process.env.NODE_ENV !== 'production',  
  preloadedState,  
  enhancers: [reduxBatch]  
});
```

Или так:

```
const store = configureStore({ reducer });
```

createReducer()

- Позволяет писать чуть меньше кода при создании reducers
- Включает в себя immer (библиотека для разрешения мутаций store)
- Рассматривать не будем (потому что существует createSlice())

createAction()

- Позволяет писать чуть меньше кода при создании actions
- Позволяет писать actions одной строкой (не вынося отдельно название)
- Рассматривать не будем (потому что существует createSlice())

createSlice()

- Позволяет создать actions и reducers для них с согласованными типами
- На вход можно передать только reducers и initialState, а так же имя slice
- На выходе получим объект с 4 полями, в числе которых actions и reducers
- Теперь нельзя добавить action без reducer случайно и тратить время на отладку описок

createSlice()

Сигнатура

```
function createSlice({  
  reducers: Object<string>, ReducerFunction | ReducerAndPrepar  
  initialState: any,  
  name: string,  
  extraReducers?:  
    | Object  
    | ((builder: ActionReducerMapBuilder) => void)  
)
```

createSlice()

/src/store/reducers.ts

Было

```
const notesReducer = (
    state: ApplicationState = initialState,
    action: IAction
) => {
    const { type, payload } = action;
    const { notes } = state;

    switch (type) {
        case ACTIONS.ADD_NOTE
        ...
    }
}
```

createSlice()

/src/store/actions.ts

Было

```
import { ..., ACTION } from '../common/types';

export const addNote = ({ note }: IAddNotePayload) => ({
  ...
});

export const editNote = ({ id, newNote }: IEditNotePayload) => ({
  ...
});

export const changeStatus = ({ id, newStatus }: IChangeStatusPayload)
```

createSlice()

/src/store/slice.ts

Стало

```
const notesStore = createSlice({
    name: 'notes',
    initialState,
    reducers: {
        addNote(state, { payload }: PayloadAction<IAddNotePayload>) {
            ...
            state.notes.push(newNote); // Прямо мутируем стейт
        },
        ...
    }
}
```

createSlice()

/src/store/slice.ts

Стало

```
const notesStore = createSlice({
    name: 'notes',
    initialState,
    reducers: {
        addNote(state, { payload }: PayloadAction<IAddNotePayload>) {
            ...
            state.notes.push(newNote); // Прямо
        },
        ...
    }
})
```

В одном месте сразу генерируются actions и reducers

createSelector()

- Заданная функция из reselect для того, чтобы не устанавливать отдельную библиотеку

Итоги

Хозяйке на заметку: готовим Redux

- Установка
- Выбираем async flow
- Какие бывают reducers?
- Создаем события и обработчики
- Создаем store
- Выбираем способ взаимодействия со store
- Подключаем store к реакту
- Используем данные из store в реакте
- Используем действия для изменения store в реакте

Хозяйке на заметку: готовим Redux

Установка

```
npm i redux react-redux @reduxjs/toolkit
```

Хозяйке на заметку: готовим Redux

Выбираем async flow

- redux-thunk (рекомендуется документацией redux)
- redux-promise
- redux-saga
- redux-observable

Хозяйке на заметку: готовим Redux

Какие бывают reducers?

- full-state reducer (используют весь store, изменяя его часть или целиком)
- slice reducers (используют часть store)
- Как правило используются **slice reducers**
- Чтобы собрать slice-reducers в один reducer, используется обертка `combineReducers()`
- При использовании `configureStore()` не нужно ничем оберачивать, достаточно просто передать объект

Хозяйке на заметку: готовим Redux

Создаем события и обработчики

- `createSlice()` из библиотеки `redux-toolkit` для создания `actions` и `reducers`

Хозяйке на заметку: готовим Redux

Создаем store

- configureStore() из библиотеки redux-toolkit для создания store
- Включить флаг devTools при локальной разработке
- Передать initialState, если он отличается от того, который определен в reducers
- Добавить миддлвару для вашего async flow (thunkMiddleware, sagaMiddleware, примеры есть в документации библиотек)

Хозяйке на заметку: готовим Redux

Выбираем способ взаимодействия со store

- Каждый нижеперечисленный момент одинаково важен и выбрать нужно что-то одно для проекта
- Можно писать или не писать отдельные селекторы
- Можно писать селекторы как функции, можно через `createSelector()`
- Можно комбинировать или не комбинировать селекторы
- Можно использовать функции сравнения или не использовать
- Можно использовать функции сравнения в `createSelector()`

Хозяйке на заметку: готовим Redux

Подключаем store к реакту

- <Provider store={store}>...</Provider>
- Оборачивает bundle-компонент приложения и пробрасывает в него store

Хозяйке на заметку: готовим Redux

Используем данные из store в реакте

- mapStateToProps в компонентах-классах

Хозяйке на заметку: готовим Redux

Используем действия для изменения store в реакте

- mapDispatchToProps в компонентах-классах

Ссылки

- Документация по Redux
- Статья про использование Redux на сервере
- Один из принципов redux – EventSourcing
- Один из принципов redux – CQRS
- Повышение производительности React и Redux с Reselect
- Документация по React-Redux
- Репозиторий Redux
- Redux DevTools Extension

Вопросы?