

Управление состоянием

State management


Тошкилин Сергей

План лекции

1. State management: что это и зачем?
2. Flux-архитектура
3. Существующие решения
4. Redux
5. Использование Redux в React
6. Идеальная библиотека?

State management: что
это и зачем?

Показательный пример







☐ по названию

☐ по описанию

☒ **езде**

Заметка №1 **Создать**

Описание заметки

Места    

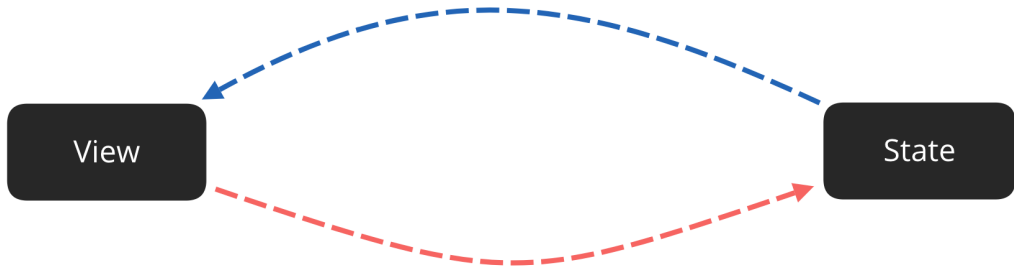
все **посетить** **посещенные**

2018 ©spt30

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
- Изменения State вызывают перерисовки во View.

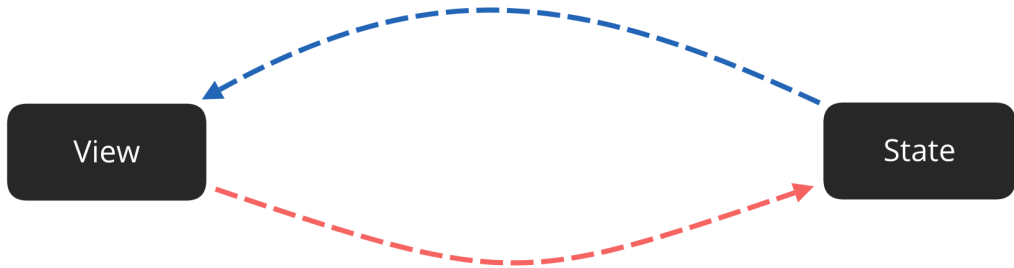
React

- Есть функция `setState`

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

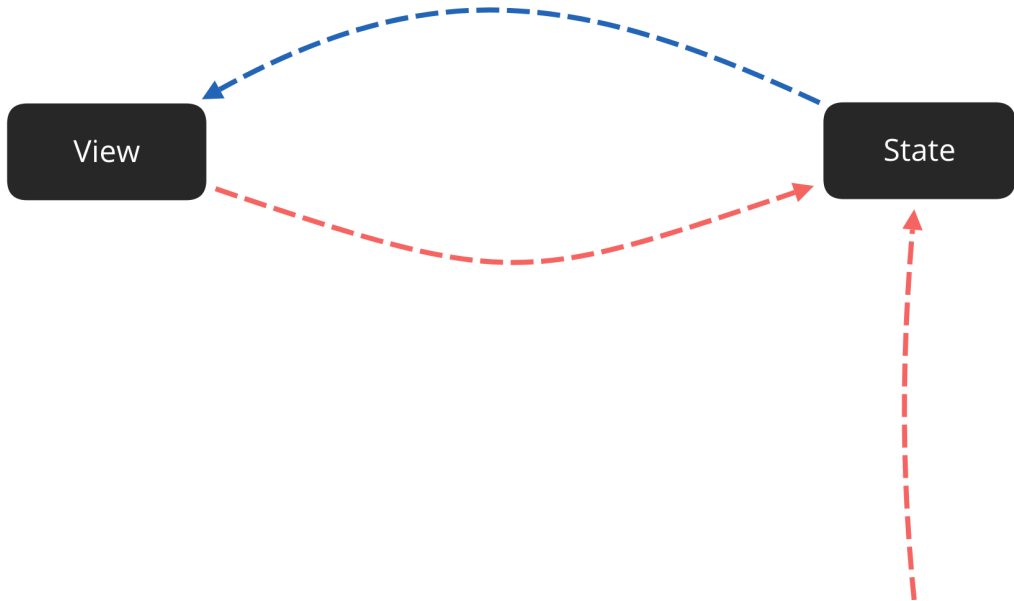
- Любые изменения `state` принудительно вызывают `render()`

Двусторонняя связь

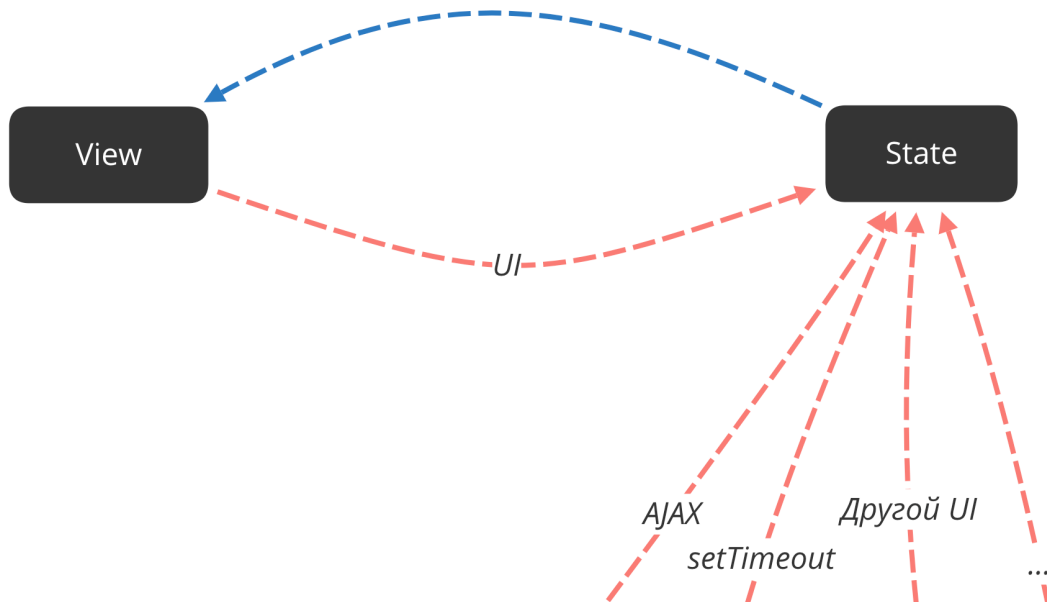


- Некоторые изменения View вызывают изменения State. Например, ввод текста или выбор радиокнопки
- Изменения State вызывают перерисовки во View.

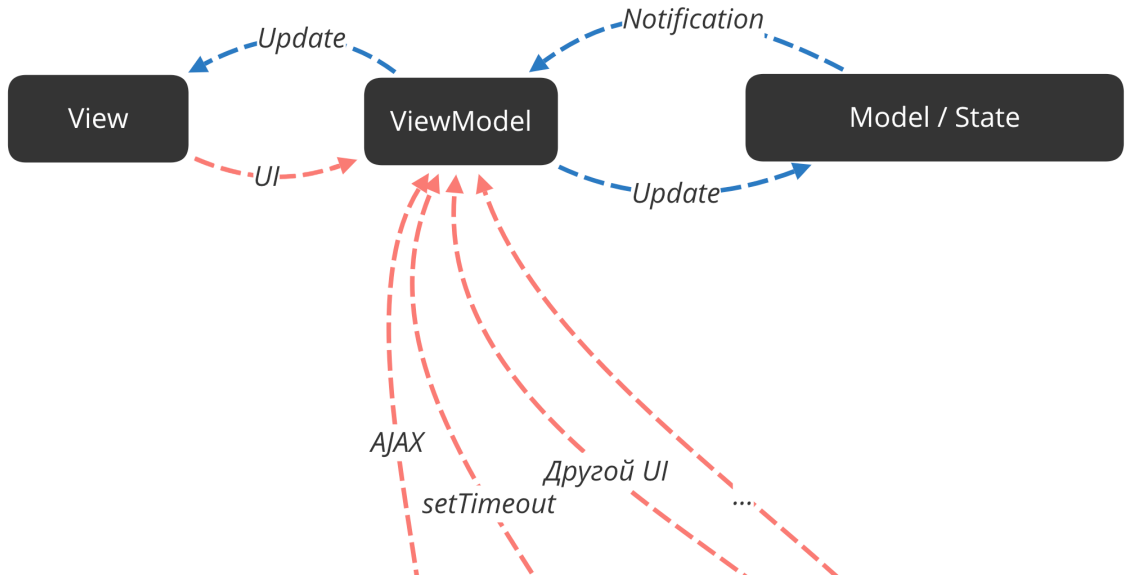
Двусторонняя связь



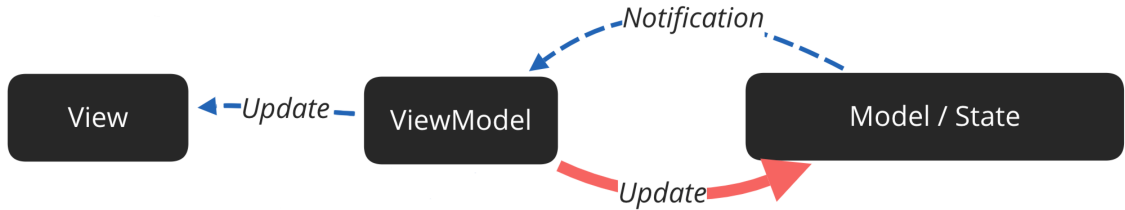
Двусторонняя связь



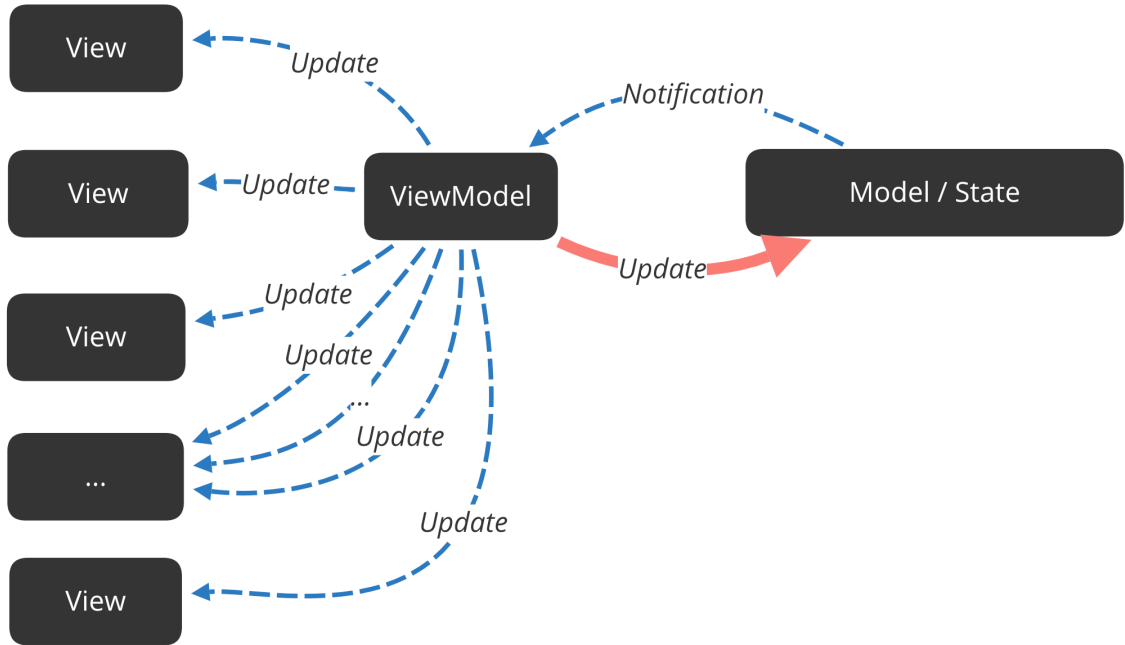
MVVM (Model-View-ViewModel)



В чем здесь проблема?



Вопросы к MVVM:

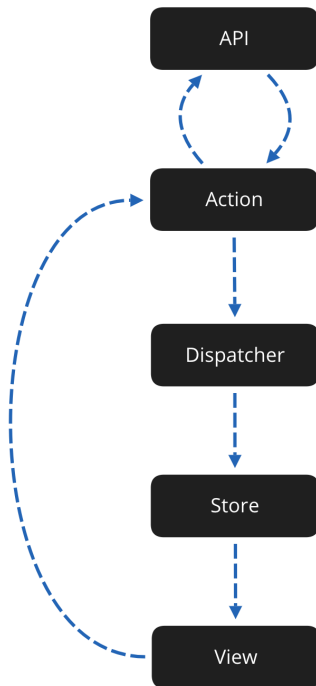


Вопросы к MVVM:

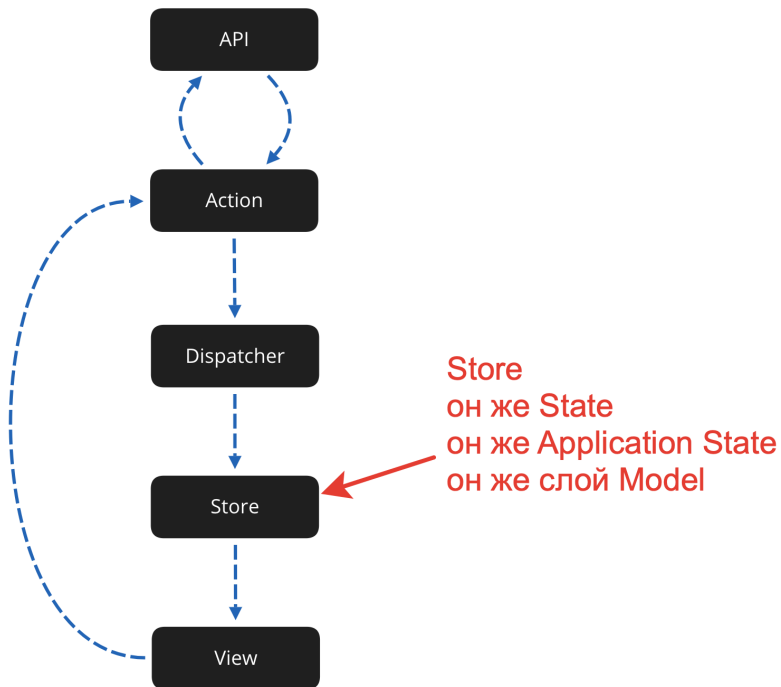
- Как контролировать внесение изменений в State?
Откуда пришли данные в State? из View/из AJAX-запроса/из записи в storage (нужное подчеркнуть)
- Как вообще отлаживать изменение View после изменения State?
- Как убедиться в том, что после изменений State остался корректным?

Flux-архитектура

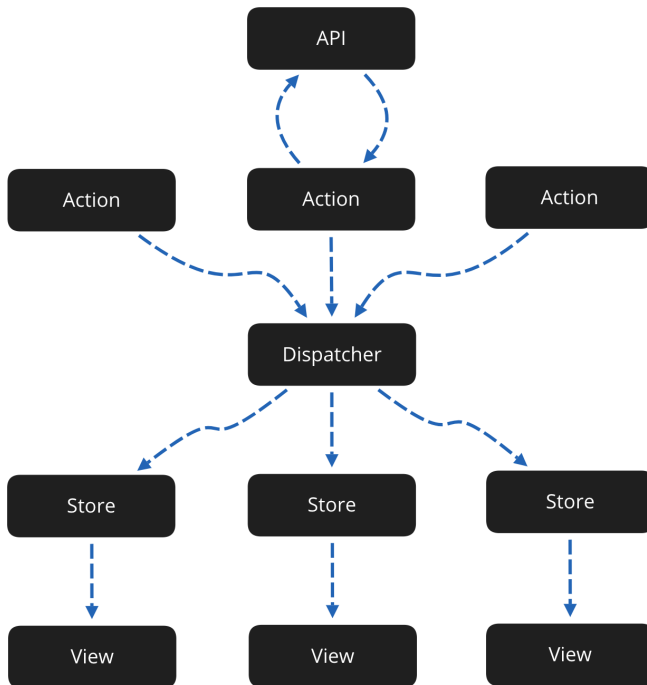
Вот такая:



Вот такая:

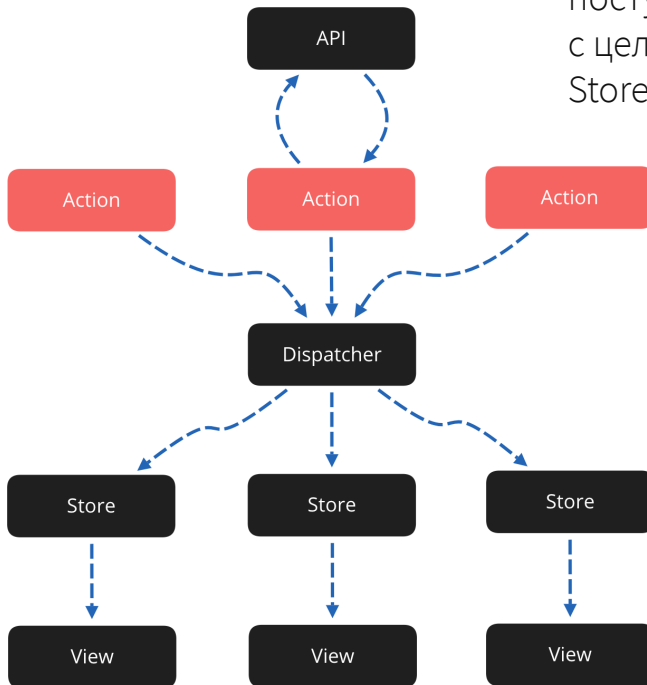


На самом деле вот такая:



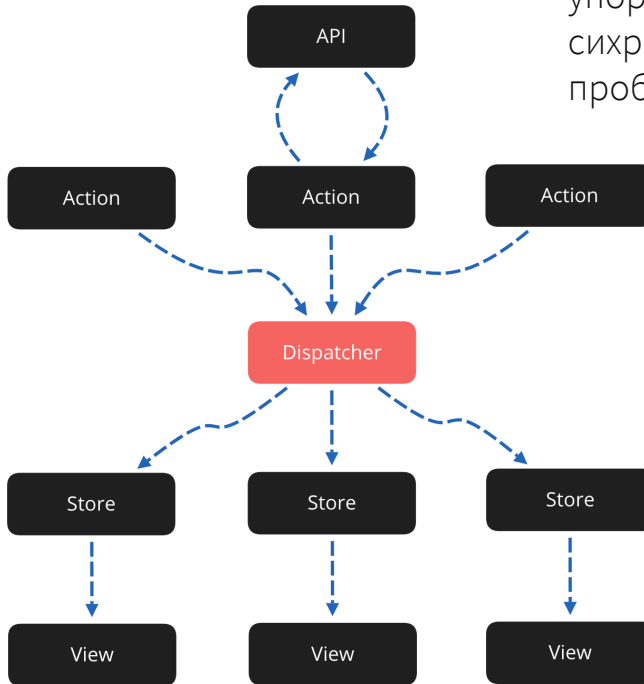
Actions

Действия, которые
поступают в диспетчер
с целью обработать в
Store



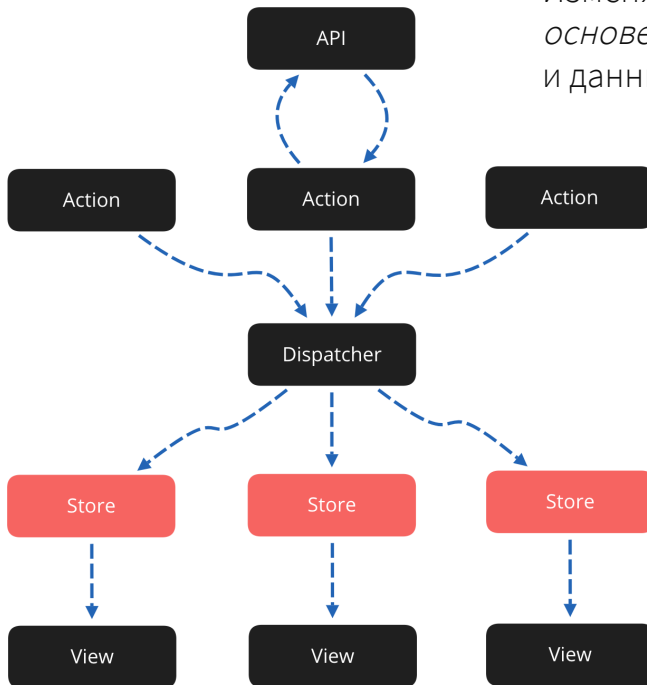
Dispatcher

Сущность, которая
упорядочивает и
синхронизирует
проброс Actions в Store



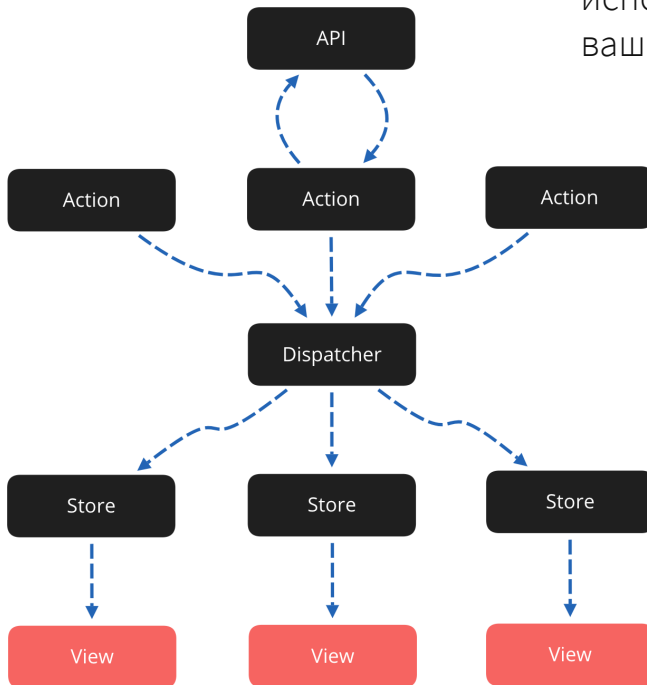
Store

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

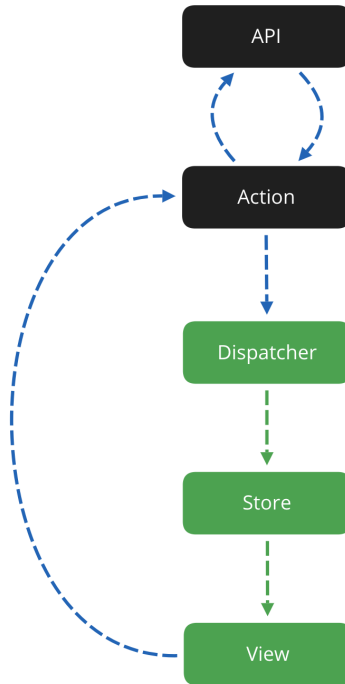


View

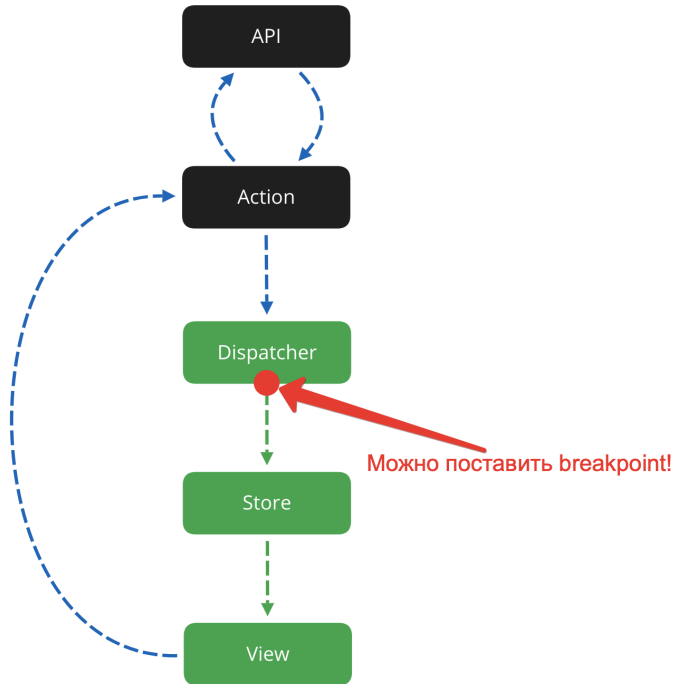
Конечная точка
использования
вашего Store



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 (нужное 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 (нужное 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 (нужное 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 (нужное 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

Angular

Vue

React

ngrx

vuex

reflux

mobx

redux

Существующие решения

Angular

Vue

React

ngrx

vuex

reflux

mobx

redux

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

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  
      };  
    }  
    ...  
  }  
}
```

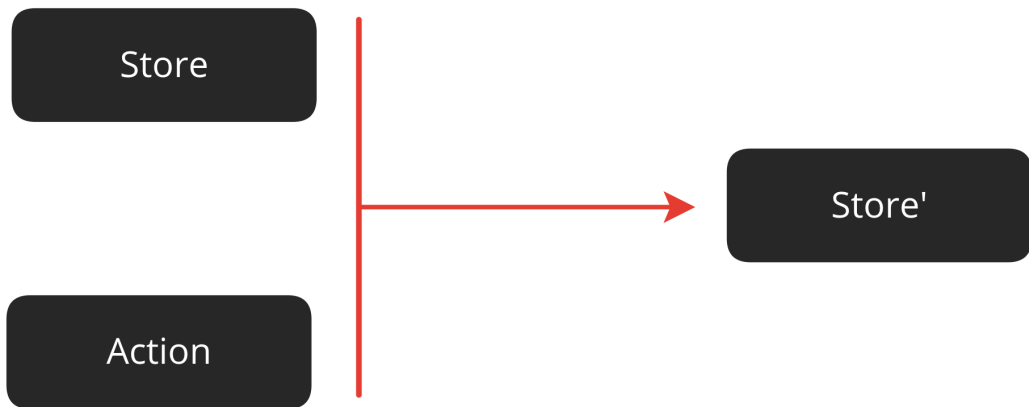
reducers

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  
      };  
    }  
  }  
  ...
```

Фичи Redux

- Есть reducers



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

Вопросы к Flux:

- ~~Как контролировать внесение изменений в Store (откуда пришли данные в Store? из View/из AJAX-запроса/из записи в storage (нужное 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 (нужное 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 разрабатываться с Redux:

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

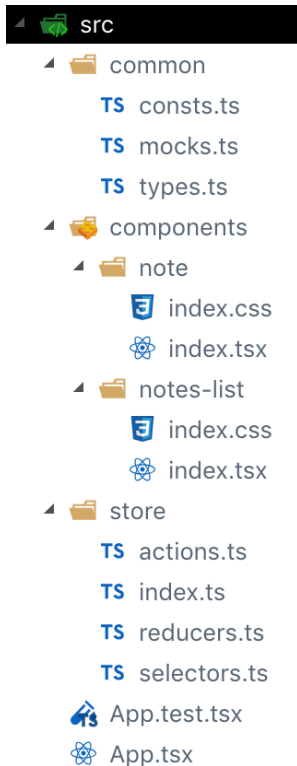
Перерыв

Можно задавать вопросы:)

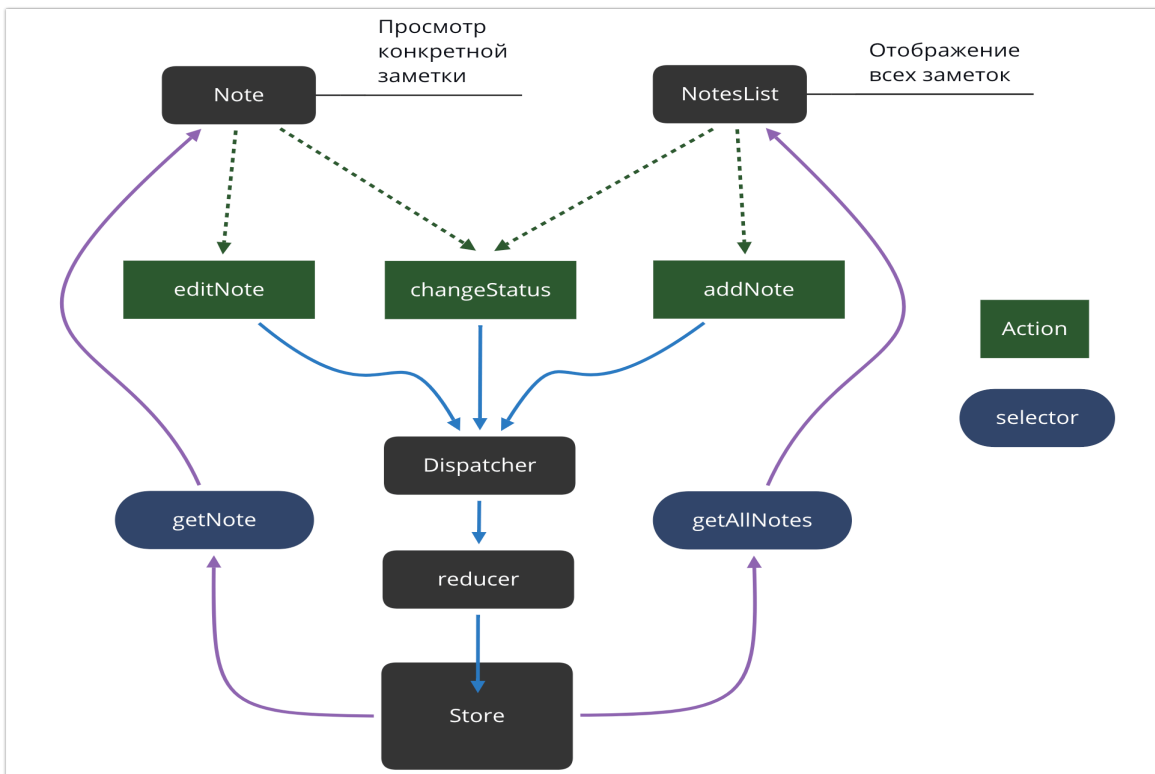
Связка React ↔ Redux

Демо

Пример приложения



Пример приложения



types

- src
 - common
 - TS consts.ts
 - TS mocks.ts
 - TS types.ts**
 - components
 - 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.test.tsx
 - App.tsx

types

```
/src/common/types.ts
```

```
export enum NOTE_STATUS {  
  actual = 'actual',  
  old = 'old',  
  deleted = 'deleted'  
}
```

```
interface IPureNote { ... }
```

```
export interface INote extends IPureNote { ... }
```

```
export enum ACTIONS { ... }
```

types

```
/src/common/types.ts
```

```
export enum NOTE_STATUS { ... }
```

```
export interface IPureNote { title: string;  
  description: string;  
  status: NOTE_STATUS;  
  created: Date;  
}
```

```
export interface INote extends IPureNote {  
  id: number;  
}
```

```
export enum ACTIONS { ... }
```

types

```
/src/common/types.ts
```

```
export enum NOTE_STATUS { ... }
```

```
export interface IPureNote { ... }
```

```
export interface INote extends IPureNote { ... }
```

```
// Все типы Action'ов
```

```
export enum ACTIONS {  
  ADD_NOTE = 'ADD_NOTE',  
  EDIT_NOTE = 'EDIT_NOTE',  
  CHANGE_STATUS = 'CHANGE_STATUS'  
}
```

types

```
/src/common/types.ts
```

```
// Payloads – ПОЛЕЗНАЯ(!) информация, которая передается в action
export interface IEditNotePayload {
    id: number;
    newNote: IPureNote;
}

export interface IAddNotePayload { ... }

export interface IChangeStatusPayload { ... }

// Интерфейс, который описывает Action целиком
export interface IAction { ... }
```

types

```
/src/common/types.ts
```

```
// Payloads – ПОЛЕЗНАЯ(!) информация, которая передается в action
export interface IAddNotePayload { ... }

export interface IEditNotePayload { ... }

export interface IChangeStatusPayload { ... }

// Интерфейс, который описывает Action целиком
export interface IAction {
  type: ACTIONS;
  payload:
    | IAddNotePayload
    | IEditNotePayload
    | IChangeStatusPayload;
}
```

actions

- src
 - common
 - TS consts.ts
 - TS mocks.ts
 - TS types.ts
 - components
 - 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.test.tsx
 - App.tsx

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
  }
});
```


Принимает 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
  }
});
```

Возвращает IAction

/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
 - TS consts.ts
 - TS mocks.ts
 - TS types.ts
 - components
 - 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.test.tsx
 - App.tsx

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:
      ...
  }
};
```

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


Инициализация Store

- src
 - common
 - TS consts.ts
 - TS mocks.ts
 - TS types.ts
 - components
 - 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.test.tsx
 - App.tsx

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
);
```

ApplicationState пишем здесь

```
/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
 - TS consts.ts
 - TS mocks.ts
 - TS types.ts
 - components
 - 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.test.tsx
 - App.tsx

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
 - TS consts.ts
 - TS mocks.ts
 - TS types.ts
 - components
 - 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.test.tsx
 - App.tsx

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: Props) => ({
  currentNote: state.notes.find(note => note.id === 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: Props) => ({  
  currentNote: state.notes.find(note => note.id === 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: Props) => ({
  currentNote: state.notes.find(note => note.id === id);
});

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

mapStateToProps передается
первым аргументом в
connect(...) и подкладывает 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 подložены 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 }));
  };
  ...
}
```

Еще раз весь цикл на
примере

Dispatch'им событие из компоненты

/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 // 'editNote',
  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 индекс редактируемой заметки
  const index = state.notes.findIndex(
    (note: INote) => note.id === id
  );
  // копируем массив заметок и вставляем на место старой новую
  return {
    ...state,
    notes: [...notes].splice(index, 1, noteWithId: [ЗАМЕТКА №56]);
  };
}
```

Кладём 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: Props) => ({  
  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
```

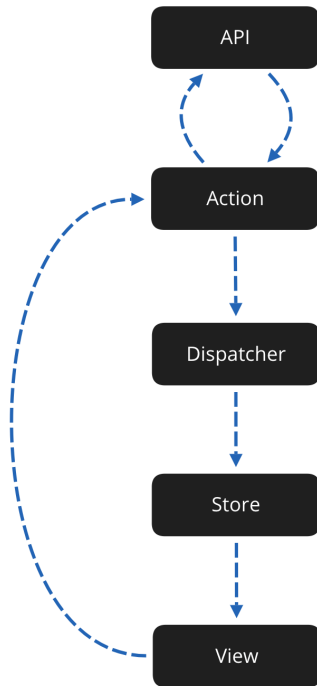
```
...  
type StateProps = ReturnType<typeof mapStateToProps>;  
  
type Props = StateProps & DispatchProps; // подкладываем StateProps в this.props  
class Note extends Component<Props, IOwnState> {  
  ...  
  public render() {  
    return (  
      ...  
      <div className="note-page__field">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

Еще раз картинка flux:



Перерыв

Можно задавать вопросы:)

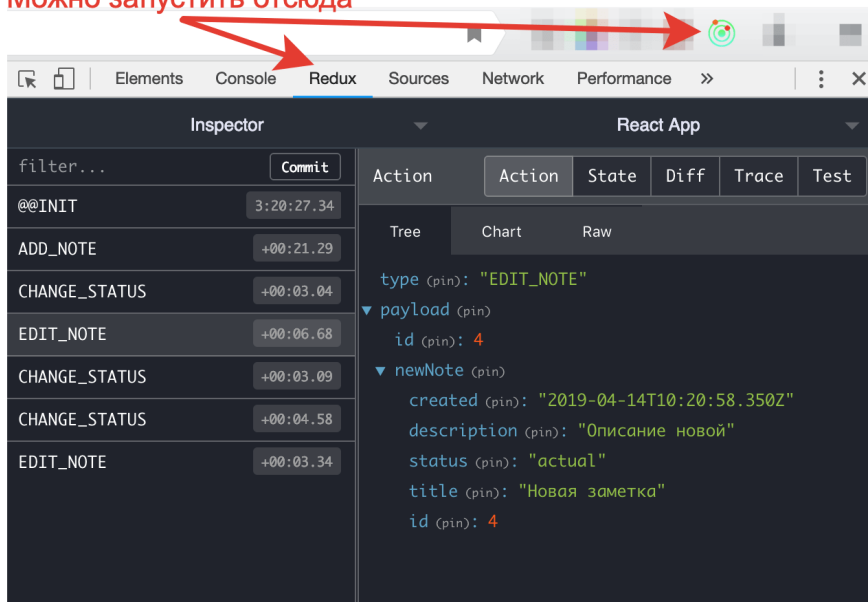
Redux DevTools

- [Ссылка на документацию и мануал по установке](#)
- Позволяет следить за Store в режиме дебага

Redux DevTools

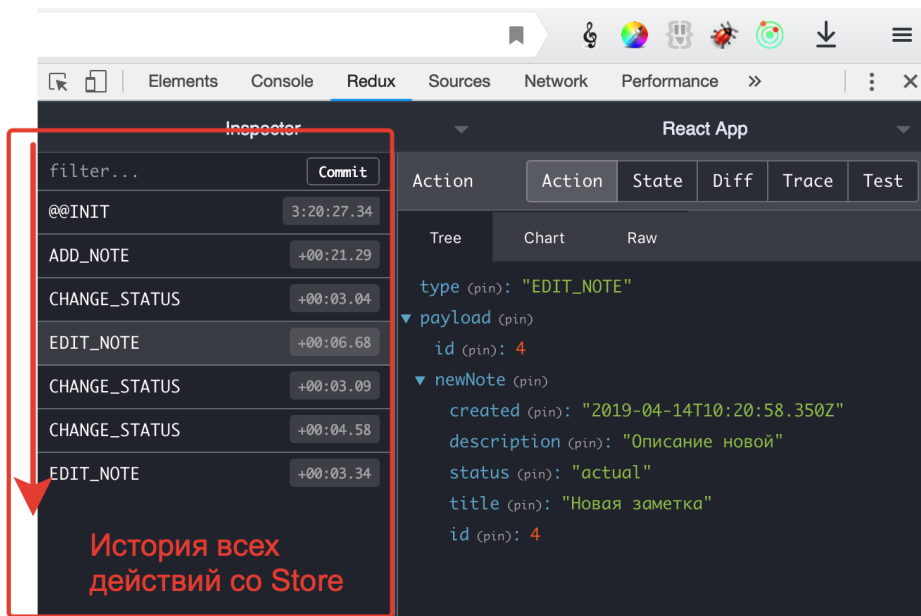
- Запуск

Можно запустить отсюда



Redux DevTools

- Отображается вся история action'ов для экземпляра приложения



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 displays the Redux DevTools interface. The top toolbar includes icons for bookmark, undo, redo, reset, debug, save, and a menu. Below the toolbar, the 'Elements' tab is selected, showing a list of components: filter..., @@INIT, ADD_NOTE, CHANGE_STATUS, EDIT_NOTE, CHANGE_STATUS, CHANGE_STATUS, and EDIT_NOTE. The 'Inspector' panel on the left shows the state of the application after the last action, with a 'Commit' button. The 'React App' panel on the right shows the current state of the application, with tabs for Action, State, Diff, Trace, and Test. The 'Tree' tab is selected, showing the state structure:

```
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

The screenshot displays the Redux DevTools interface. The top bar includes navigation icons and tabs for Elements, Console, Redux (selected), Sources, Network, and Performance. The main area is divided into two panels: the Inspector on the left and the React App on the right.

Inspector Panel: Shows a list of components and their state. The components are filter..., @@INIT, ADD_NOTE, CHANGE_STATUS, EDIT_NOTE, CHANGE_STATUS, CHANGE_STATUS, and EDIT_NOTE. Each component has a corresponding state object and a timestamp.

React App Panel: Shows the Redux state and the React App components. The state is an array of notes. The components are notes, 0, 1, 2, and 3.

State:

```
{
  "notes": [
    {
      "created": "2019-04-09...",
      "description": "",
      "id": 0,
      "status": "actual",
      "title": ""
    },
    {
      "created": "2019-04-03T16:59:59.834Z",
      "description": "Не забыть показать Redux DevTools",
      "id": 2,
      "status": "actual",
      "title": "Прочитать лекцию про Redux"
    },
    {
      "created": "2019-01-22...",
      "description": "//",
      "id": 3,
      "status": "actual",
      "title": ""
    }
  ]
}
```

Redux DevTools

The screenshot displays the Redux DevTools interface. The top bar includes navigation icons and tabs for Elements, Console, Redux (selected), Sources, Network, and Performance. The main area is divided into two panels: the Inspector on the left and the React App on the right.

Inspector Panel:

- 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 Panel:

The React App panel shows the state of the application. The state is an object with a `notes` property, which is an array of 3 objects. The state is displayed in a tree view, with the `notes` array expanded to show its contents.

The state structure is as follows:

```
{
  notes: [
    {
      created: '2019-04-14T10:20:48.630Z',
      description: 'Описание новой заметки',
      id: 2
    }
  ]
}
```

The `notes` array contains 3 objects. The first object is highlighted in green, showing its `created` and `description` properties. The `description` property is highlighted in green and contains the text "Описание новой заметки".

То же демо, только с
Redux DevTools

Идеальная библиотека?

Да, но:

1. В Store теперь хочется складывать вообще все
2. В redux'e store не знает о том, какие его части важны для каждой конкретной view
3. Нет асинхронности:(

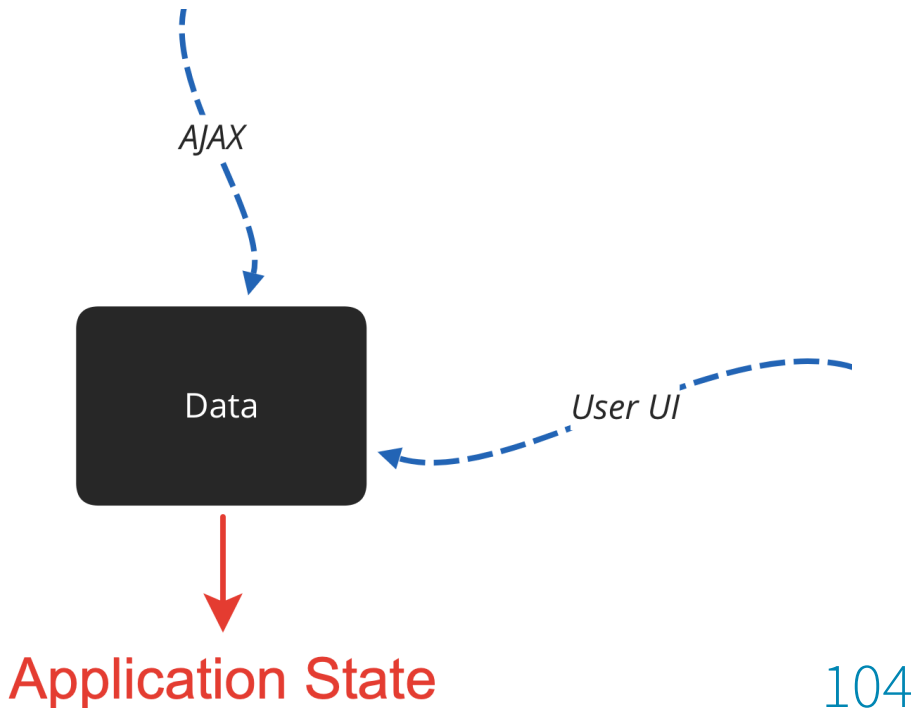
Проблема 1. Что делать, если Store стал одной большой помойкой

Куда класть данные и когда использовать Application State?

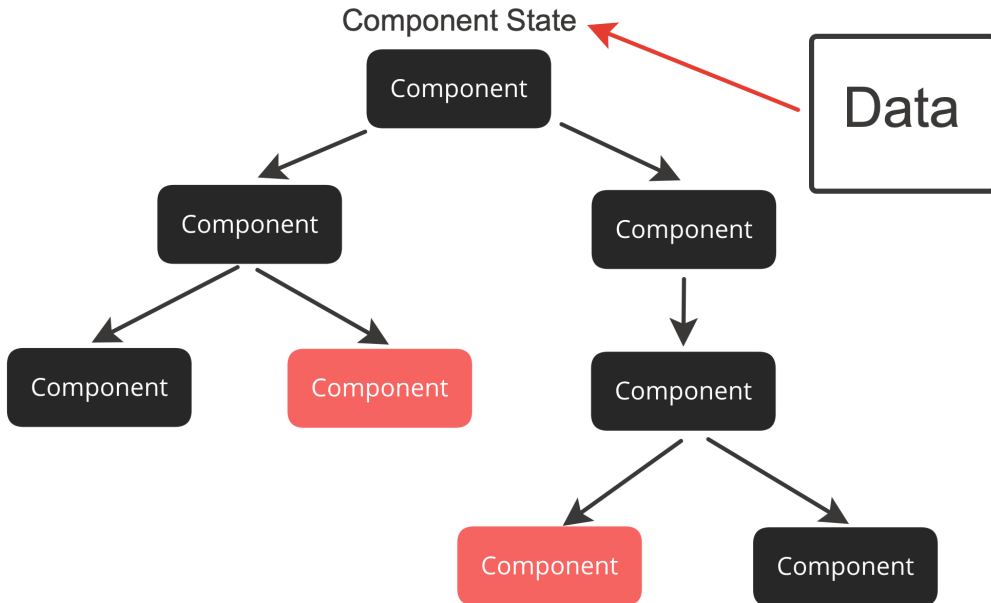
Все данные делятся на 3 типа:

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

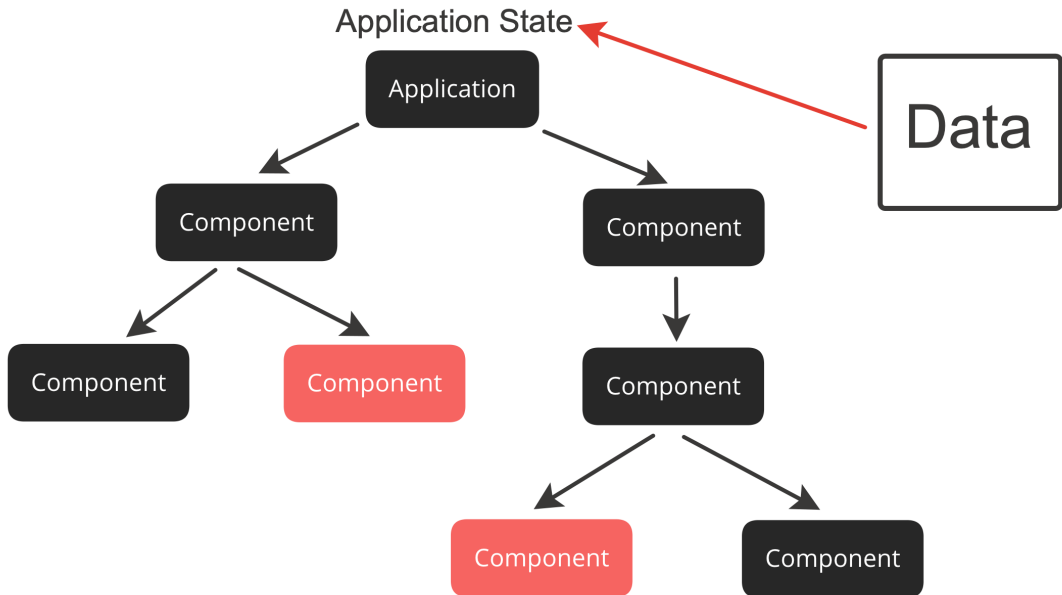
Данные, которые могут поменяться из
нескольких мест



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...
- [React Context API!](#)
- «Context provides a way to pass data through the component tree without having to pass props down manually at every level»

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

Хорошо бы, чтобы перерисовывались только те View, данные для которых реально поменялись...

Библиотека Reselect!

Проблема 3. Асинхронность?

[Async Flow](#): документация redux

Библиотеки для работы с асинхронностью в Redux:

- [redux-thunk](#) (рекомендуется, чтобы не заморачиваться)
- [redux-promise](#)
- [redux-saga](#) (true-way)
- [redux-observable](#)

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?