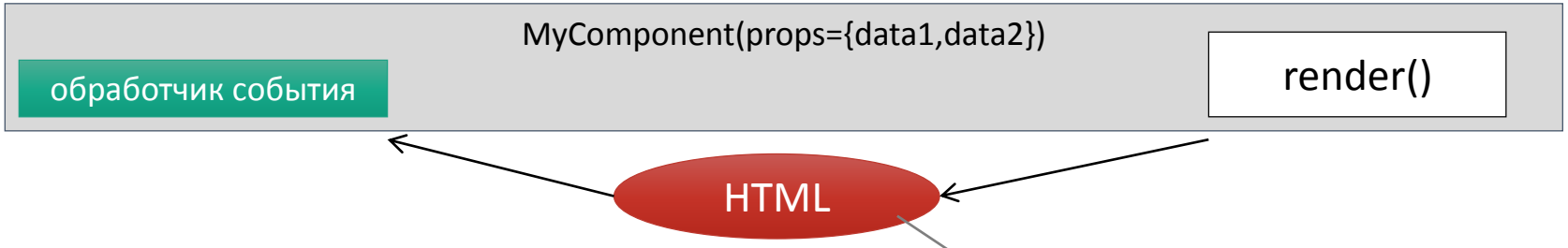
ПРОДОЛЖАЕМ ИЗУЧАТЬ REDUX

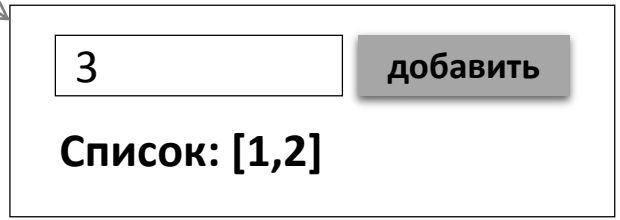
ОБЩАЯ СХЕМА РАБОТЫ С REDUX

ШАГ 1: СОЗДАЕМ ЧИСТЫЙ КОМПОНЕНТ

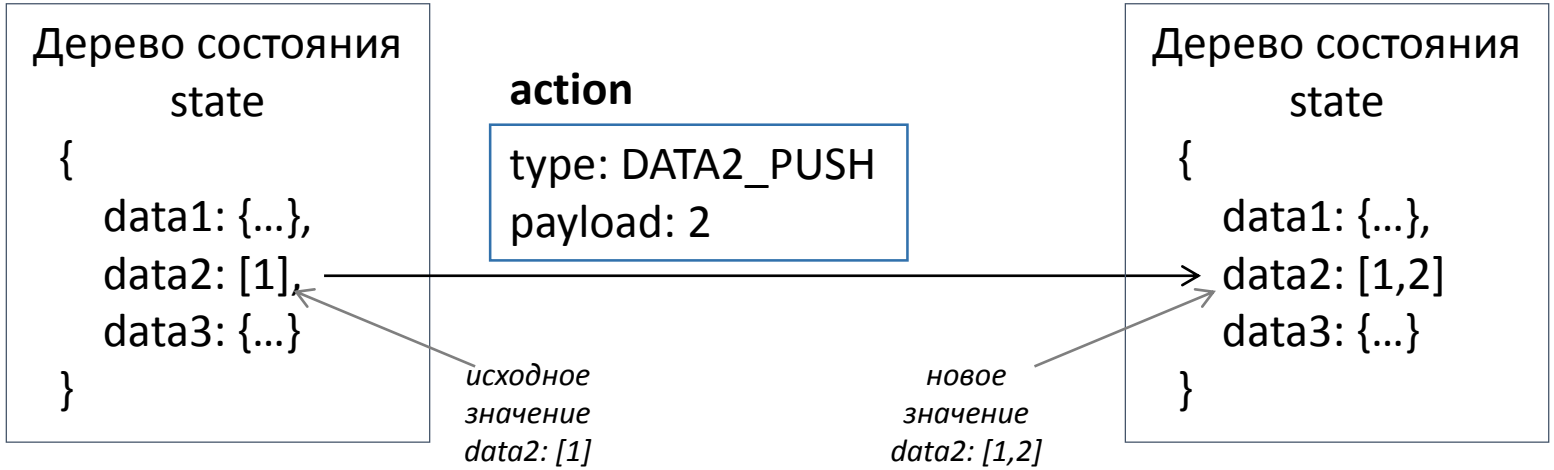
Создаем чистый компонент, который зависит только от props:



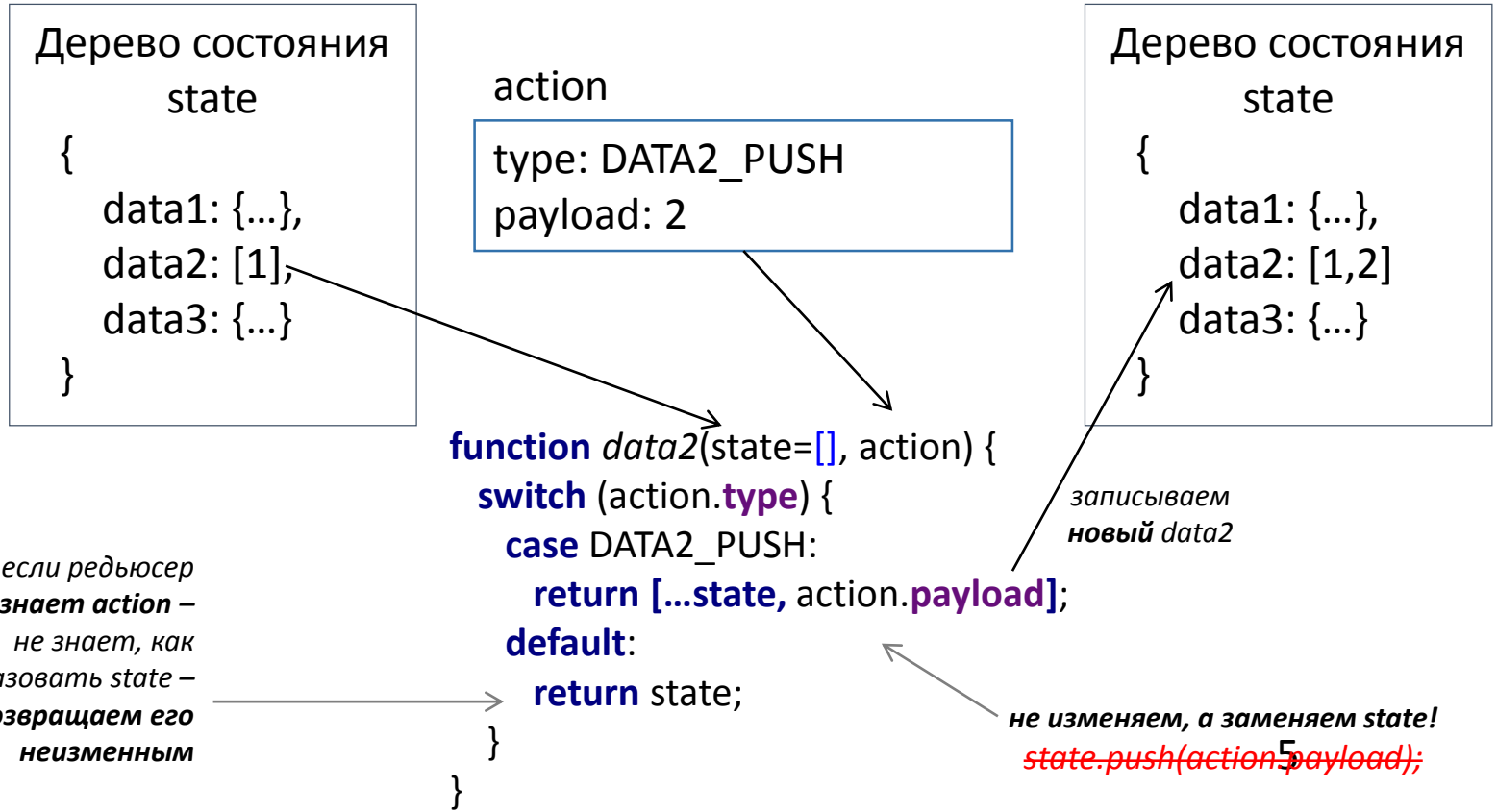

```
class MyComponent extends PureComponent {
   data2Click() { console.log(this.inp.value); }
   render() {
     const {data1, data2} = this.props;
     return
       <div>
         <div><input ref={(inp)=>this.inp=inp}></div>
         <button onClick={this.data2Click.bind(this)}>добавить</div>
         <div>Список: {data2.toString()}</div>
       </div>;
  }
}
```



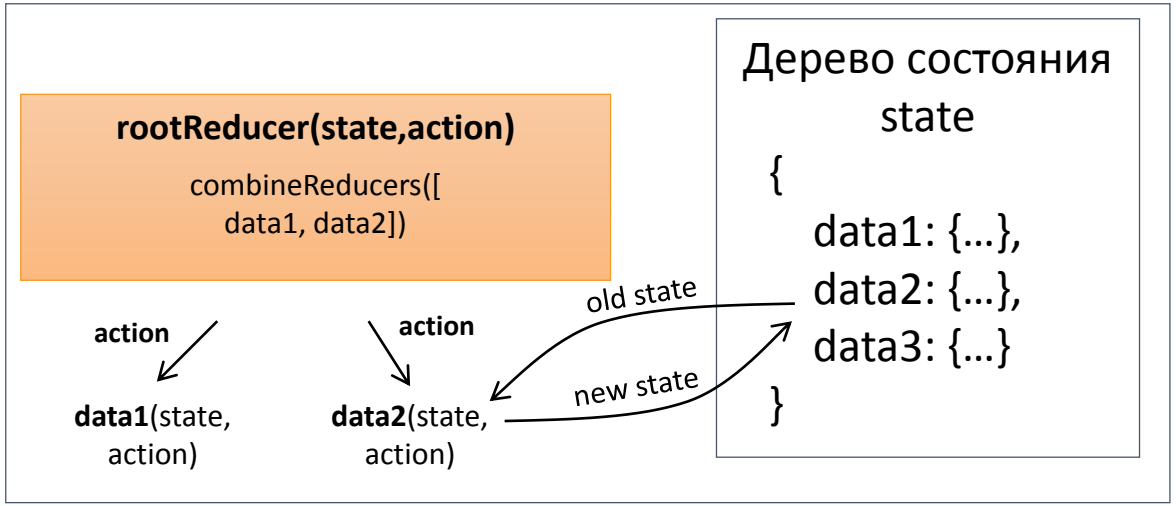
ШАГ 2: РЕШАЕМ, ЧТО ХРАНИТСЯ В STATE, КАКИЕ МОГУТ БЫТЬ ACTION И КАК БУДЕТ МЕНЯТЬСЯ STATE ПРИ ИХ ПРИМЕНЕНИИ



ШАГ 3: ПИШЕМ ЧИСТУЮ ФУНКЦИЮ-РЕДЬЮСЕР ДЛЯ ИЗМЕНЕНИЯ ЧАСТИ STATE



ШАГ 4: СОЗДАЕМ КОРНЕВОЙ РЕДЬЮСЕР, КОМБИНИРУЕМ РЕДЬЮСЕРЫ И СОЗДАЁМ STORE




```
function rootReducer(state = {},action) {
   combineReducers([
      data1, data2]);
}
```

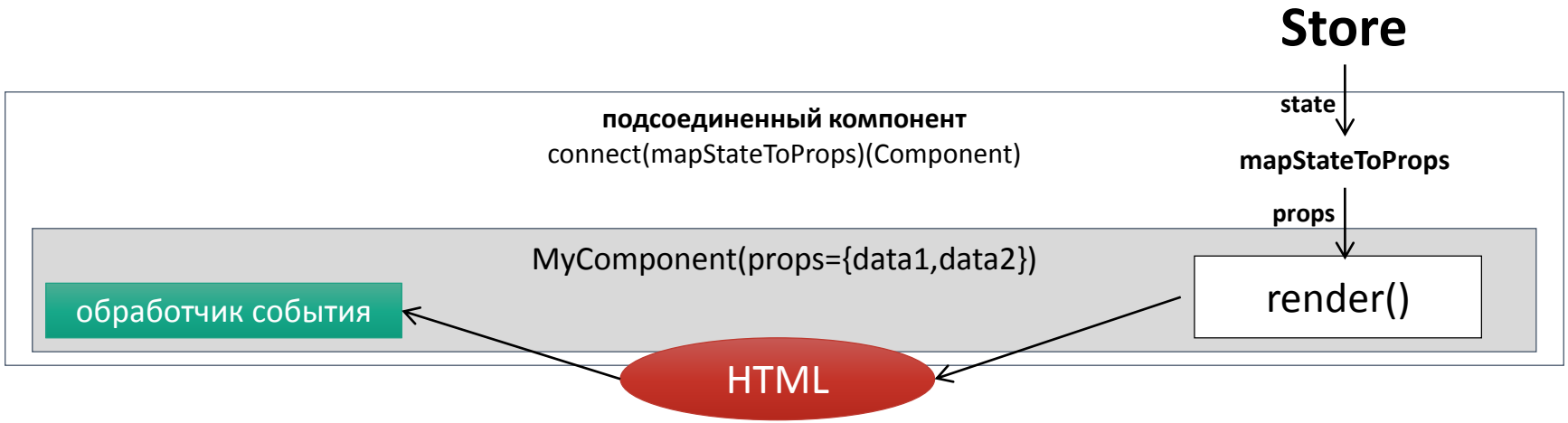
другой способ задать начальное значение state

store = createStore(rootReducer, initialState)

корневой редьюсер как параметр

начальное значение state (опционально)

ШАГ 5: СОЕДИНЯЕМ STORE И КОМПОНЕНТ



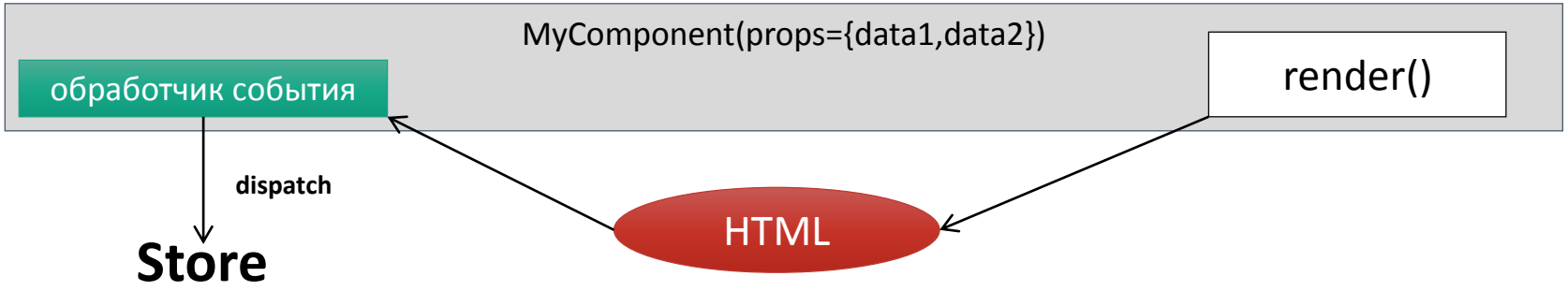
7

```
function mapStateToProps(state) {
  return {
    data1: state.data1,
    data2: state.data2
  }
}
export default connect(mapStateToProps)(MyComponent)
```

7

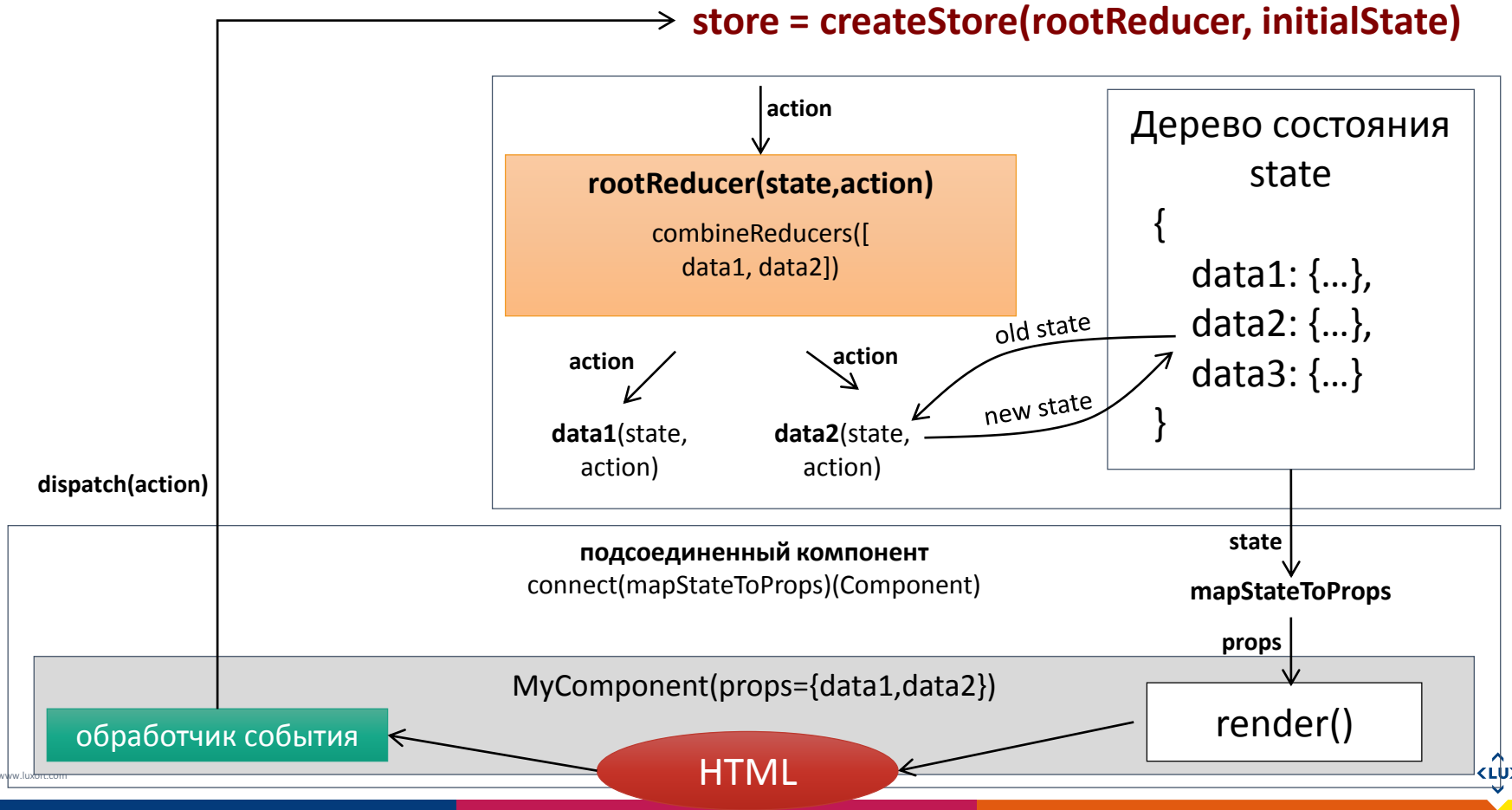
ШАГ 6: СОЕДИНЯЕМ ОБРАБОТЧИК СОБЫТИЯ И STORE

Создаем чистый компонент, который зависит только от props:

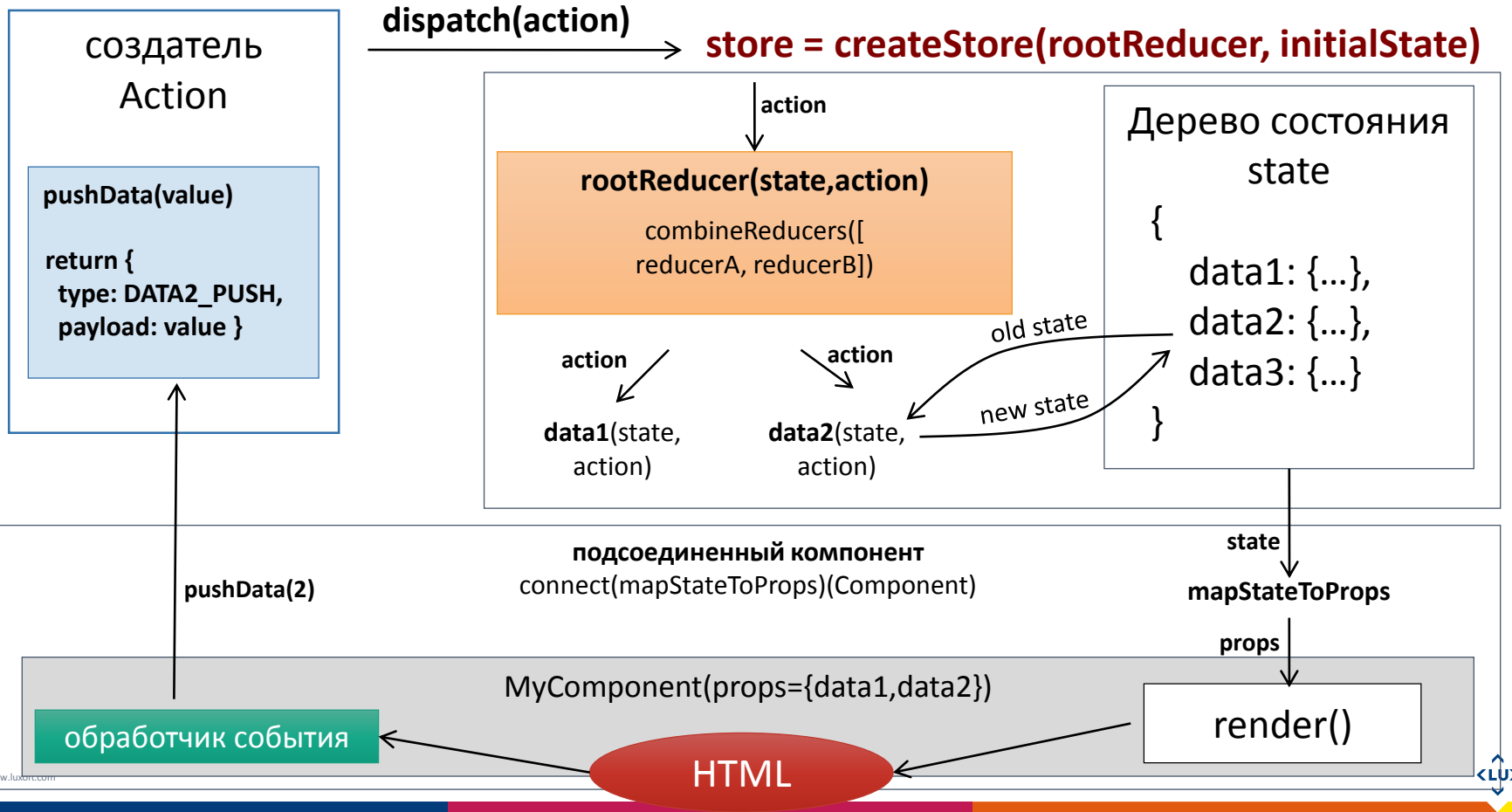



```
class MyComponent extends PureComponent {
  data2Click() {
    let {dispatch} = this.props;
    dispatch({
      type: DATA2_PUSH,
      payload: this.inp.value
    })
  }
  render() {
    const {data1, data2} = this.props;
    return
      <div>
        <div><input ref={(inp)=>this.inp=inp}></div>
        <div onClick={this.data2Click.bind(this)}>{data2}</div>
      </div>;
  }
}
```

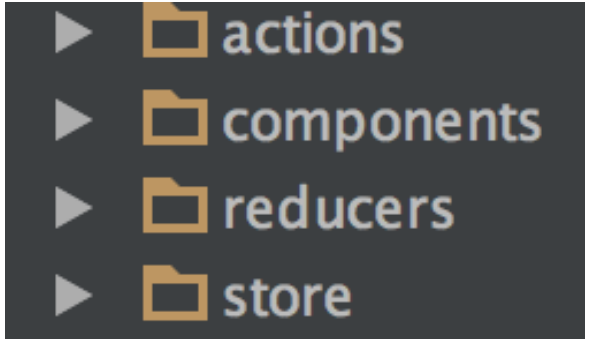
REDUX: ОБЩАЯ СХЕМА РАБОТЫ



REDUX: ОБЩАЯ СХЕМА РАБОТЫ + ACTION CREATOR



СТРУКТУРА ПАПОК REDUX

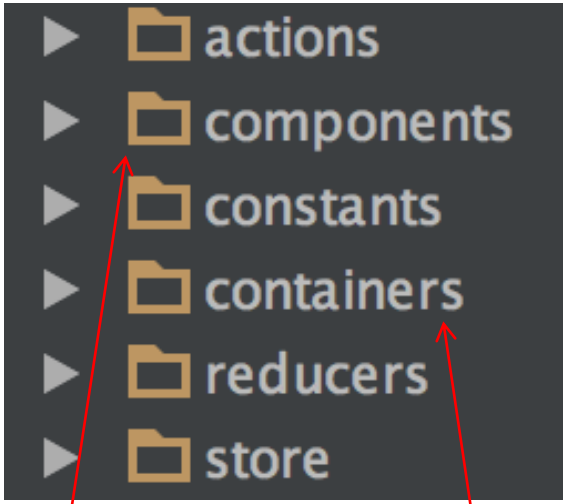


- Как правило, используется следующая структура папок: Actions, Components, Reducers и Store
- В папке **Actions** обычно хранятся файлы со всеми действиями / **генераторами действий**, которые предполагается использовать в приложении.
- В папке **Components** очевидно размещаются все **компоненты**.
- В папке **Reducers** содержится иерархия **редьюсеров**.
- В папке **Store** обычно хранится один файл с использованием redux **createStore** и некоторые используемые промежуточные слои.

СТРУКТУРА ПАПОК REDUX

- Такая структура вполне подходит, но ее можно улучшить и дополнить разделением функций. В чем разница:
- В папке **Constants** можно хранить список **констант** (обычно для типов действий), чтобы гарантировать, что преобразователи и компоненты используют одну и ту же переменную.

- В данной структуре папка **Components** содержит **компоненты React**, которые определяются исключительно свойствами и никак не связаны с Redux. Они должны оставаться неизменными независимо от используемых маршрутизатора, библиотеки данных и т. д.
- В папке **Containers** хранятся компоненты React, которые **взаимодействуют с Redux, Router** и т. д. Они в большей степени связаны с приложением.

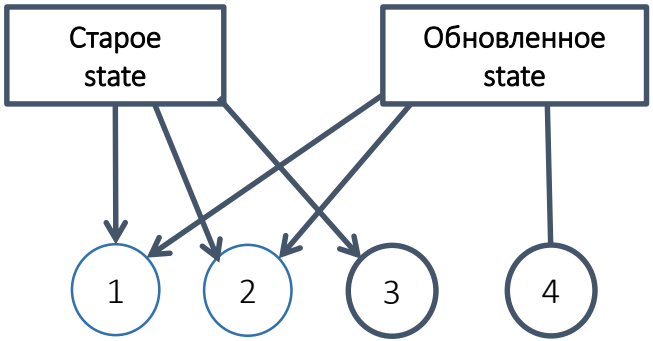


в **компонентах** –
чистые компоненты,
не связанные с Redux

в **контейнерах** –
подсоединенные к
Redux компоненты

ПОТРЕБЛЕНИЕ ПАМЯТИ В REDUX

- Наличие неизменяемого состояния не означает, что оно каждый раз полностью создается заново.
- В приведенном примере видно, что части состояния 1 и 2 не изменяются, и для них не требуется выделение дополнительной памяти.
- Только часть 3 изменилось в 4, поэтому для нее потребуется дополнительная память.
- Однако после того как все компоненты будут обновлены, часть 3 отправится в корзину, и память будеть освобождена.



REDUX: РАБОТА С СЕРВЕРОМ (ПОКА БЕЗ THUNK)

```
componentDidMount(){
   this.loadData();
}
loadData(){
   let {dispatch} = this.props;
   dispatch(startLoading());
   fetch('http://localhost:4730')
      .then(function(response) {
         return response.json();
      }).then(function(json) {
      dispatch(addData(json.gridRecords))
   }).then(function(){
      dispatch(stopLoading());
   })
}
```

как обычно, работу с сервером выполняем в componentDidMount – это подходящее место для отправки сообщения на сервер

отправляем в STORE action о начале загрузки, чтобы отобразить сообщение, что загрузка началась

теперь отправляем запрос на сервер

пришел ответ от сервера с данными – отправляем в STORE данные и тип – добавь данные в STATE

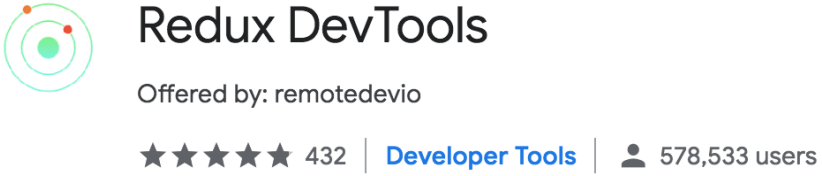
загрузка завершена – диспэтчим action об остановке загрузки

REDUX DEV TOOLS

REDUX DEVTOOLS: УСТАНОВКА

Redux DevTools – плагин для Chrome.

Необходимо скачать его в Chrome Web Store: https://chrome.google.com/webstore



Также измените Store/index.js:

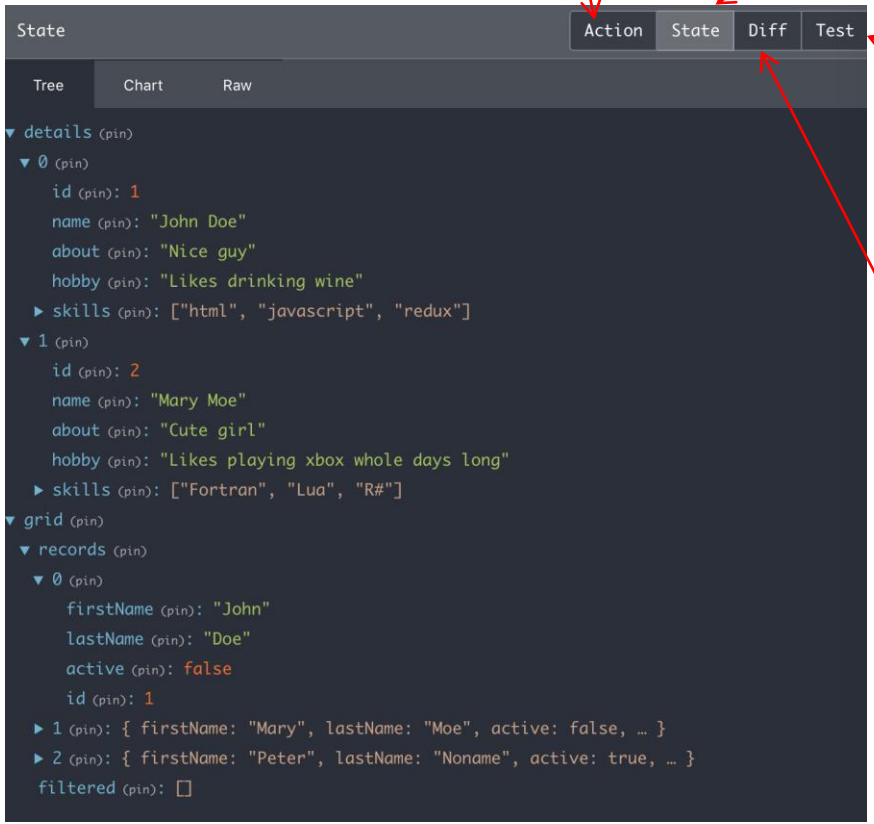
```
export default function configureStore(initialState) {
  return createStore(rootReducer, {},
    window.devToolsExtension ? window.devToolsExtension() : f => f);
}
```

так мы сообщаем DevTools об изменениях Store

REDUX DEVTOOLS: ДЕРЕВО ОБЪЕКТОВ

просмотр действий

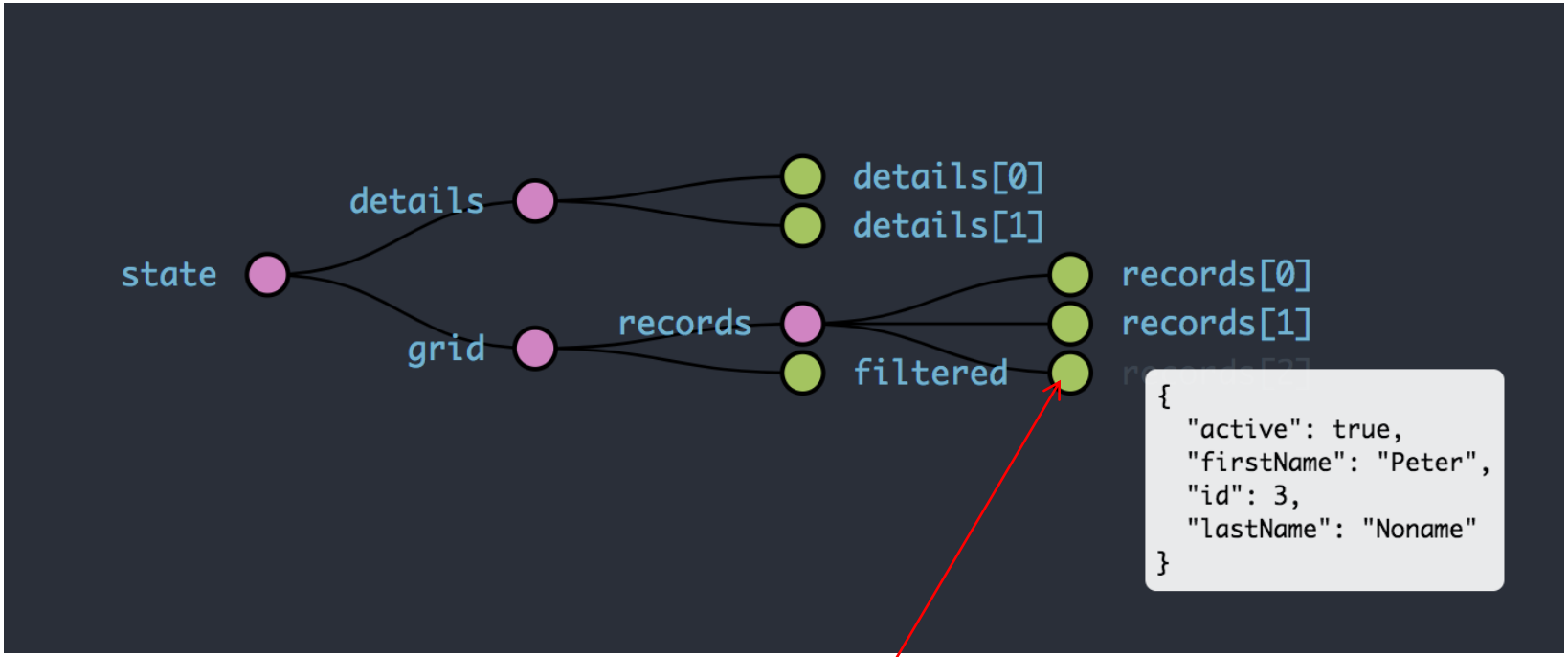
просмотр текущего состояния



автогенерация тестов по истории изменения состояний

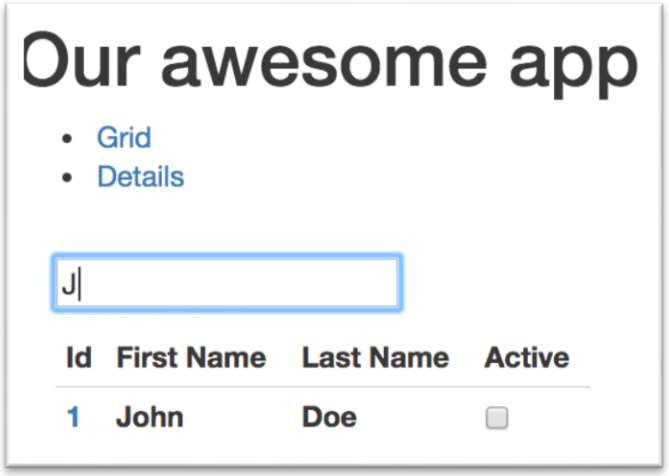
просмотр изменений состояний

REDUX DEVTOOLS: REDUX DEVTOOLS: ДИАГРАММА STATE



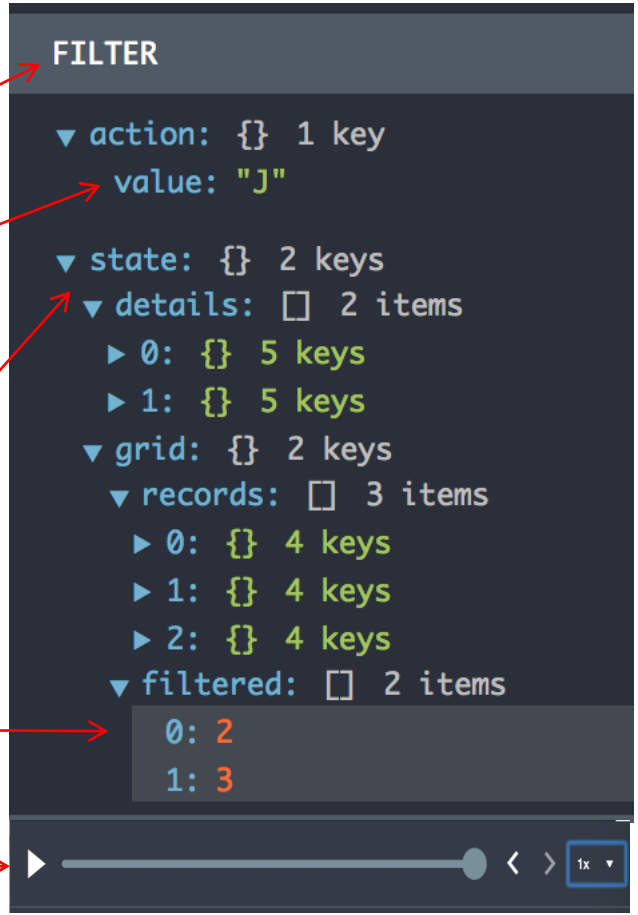
подводя мышку, можно получить детальную информацию

REDUX DEVTOOLS: ОТСЛЕЖИВАНИЕ ИЗМЕНЕНИЙ СОСТОЯНИЯ



action.type

action.value



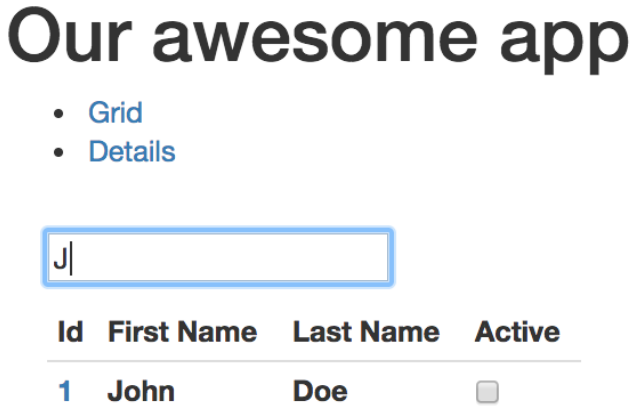
результат изменения state

отфильтрованные (спрятанные) id

можно воспроизвести всю историю применения action – и изменений state

REDUX DEVTOOLS: СГЕНЕРИРОВАННЫЙ ТЕСТ

```
import reducers from '../../reducers';
test('reducers', () => {
  let state;
  state = reducers({
    grid: {
      records: [
        {firstName: 'John', lastName: 'Doe', active: false, id: 1},
        {firstName: 'Mary', lastName: 'Moe', active: false, id: 2 },
        {firstName: 'Peter', lastName: 'Noname', active: true, id: 3}],
      filtered: []
    }
  }, {type: 'FILTER', value: 'J'});
  expect(state).toEqual({
    grid: {
      records: [
        {firstName: 'John', lastName: 'Doe', active: false, id: 1},
        {firstName: 'Mary', lastName: 'Moe', active: false, id: 2 },
        {firstName: 'Peter', lastName: 'Noname', active: true, id: 3}],
      filtered: [2, 3]
    }
  });
});
```



прежнее значение filtered – записи не отфильтрованы (показывать все)

action: фильтруем по значению 'J'

отфильтрованы записи с id 2 и 3

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ О REDUX

СТОИТ ЛИ ПОМЕЩАТЬ ЛЮБОЕ СОСТОЯНИЕ В STORE?

- Не существует “правильного” ответа на этот вопрос. Некоторые пользователи предпочитают хранить все данные в Redux, чтобы поддерживать полностью сериализуемую и контролируемую версию своего приложения на протяжении всего времени. Другие предпочитают выносить некритичные данные (UI состояние), как, например “is this dropdown currently open”, в состояние компонента.
- Ниже описаны некоторые негласные правила для определения тех частей данных, которые должны хранится в Redux-хранилище:

- Остальные части приложения используют эти данные?
- Потребуется ли в дальнейшем возможность создавать данные, основанные на этих данных?
- Эти данные используются для управления несколькими компонентами?
- Важна ли Вам возможность восстанавливать это состояние в какой-то момент времени, т.е. отладка по времени (time travel debugging)?
- Требуется ли кэшировать данные, т.е. использовать то, что уже хранится в состоянии вместо повторного запроса?

КАК МНЕ ХРАНИТЬ ВЛОЖЕННЫЕ ИЛИ ДУБЛИРУЮЩИЕСЯ ДАННЫЕ В МОЕМ СОСТОЯНИИ?

- Данные с идентификаторами, вложенностью или отношениями, как правило, следует хранить в “нормализированном” стиле: каждый объект должен быть сохранен однажды, идентифицирован, и другие ссылающиеся на него объекты должны хранить только идентификатор, а не копировать весь объект.

Поскольку мы рассматриваем наш Redux Store как «базу данных», здесь также применяются многие принципы проектирования баз данных. Например, если у нас есть отношение «многие ко многим», мы можем моделировать это, используя промежуточную таблицу, в которой хранятся идентификаторы соответствующих элементов.
Для согласованности мы, вероятно, также хотели бы использовать один и тот же подход byId и allIds, который мы использовали для фактических таблиц элементов, например:

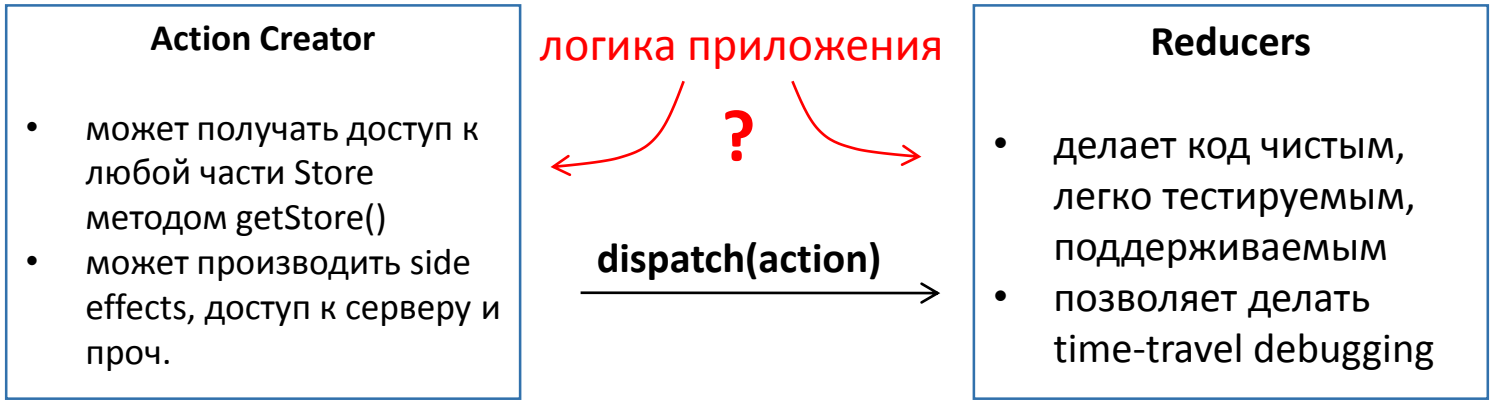
```
{
  entities: {
    authors : { byId : {}, allIds : [] },
    books : { byId : {}, allIds : [] },
    authorBook : {
      byId : {
        1 : {
          id : 1,
            authorId : 5,
            bookId : 22
        },
        2 : {
          id : 2,
            authorId : 5,
            bookId : 15,
        }
        3 : {
          id : 3,
            authorId : 42,
            bookId : 12
        }
      },
      allIds : [1, 2, 3]

    }
  }
}
```

КАК ДОЛЖНА ВЫГЛЯДИТЬ МОЯ ФАЙЛОВАЯ СТРУКТУРА? КАК Я ДОЛЖЕН ГРУППИРОВАТЬ ГЕНЕРАТОРЫ ДЕЙСТВИЙ И РЕДЮСЕРЫ В ПРОЕКТЕ?

- Поскольку Redux - это просто библиотека хранения данных, у нее нет точного мнения о том, как ваш проект должен быть структурирован. Тем не менее, есть ряд общих паттернов, которые большинство разработчиков Redux склонны использовать:
- **Rails-style**: отдельные директории для Actions, Constants, Reducers, Containers и Components
- **Domain-style**: отдельные директории для фичи или домена, возможно, с поддиректориями для каждого типа файлов
- **Ducks**: похож на domain-style, но явно связывающий действия и редьюсеры, часто определяя их в том же файле

КАК Я ДОЛЖЕН РАЗДЕЛЯТЬ ЛОГИКУ МЕЖДУ РЕДЬЮСЕРАМИ И ГЕНЕРАТОРАМИ ДЕЙСТВИЙ (ACTION CREATORS)?



Пример получения данных из другой части Store (пока без Thunk):

```
import store from '../store';
export const SOME_ACTION = 'SOME_ACTION';
export function someAction() {
   return {
      type: SOME_ACTION,
      items: store.getState().otherReducer.items,
   }
}
```

ОБСУЖДЕНИЕ: ВОЗМОЖНОСТЬ ПОЛУЧЕНИЯ В РЕДЬЮСЕРЕ ПОЛНОГО СОСТОЯНИЯ

Допустим, в редьюсере foo надо получить часть информации из bar:

```
function bar (barState = 4, action) {
  if (action.type === 'ODD') return 3
  else return 4
}
function foo (fooState = 0, action, fullState) {
  return fooState + fullState.bar
}
combineReducers({
  foo,
  bar,
}, {
  extraArg: state => state,
})
```

так сделать не получится

Ответ Дэна Абрамова (автора Redux)

gaearon commented on 25 May 2016

Это было предложено много раз, поэтому мы, вероятно, должны рассмотреть это предложение.

Я опасаюсь делать это по следующим причинам:

- Будет передаваться не актуальное, а **предыдущее** состояние – ведь до того, как оно попадет в ваш редьюсер, какой-то другой редьюсер мог его поменять. Что делать, если вы хотите иметь обновленную версию глобального состояния?
- Это работает только на одном уровне. Что делать, если вы добавляете дополнительный слой combineReducers()?

Если вы сразу же подключаете такие вещи вручную [то есть пишите свой собственный combineReducers вместо использования стандартного], ни одна из этих проблем не является серьезной проблемой, потому что вы **полностью контролируете ситуацию**.

ПРАКТИКА

Блок 3.

Задание 2.
Продолжаем изучать Redux

Задание 3.
Работаем с сервером