ANGULAR

Redux + Angular

NGRX - NGXS - Akita

Максим Иванов

Frontend Developer, Синимекс
Founder @Angular-RU на Github
@splincode

Кирилл Юсупов

Frontend Developer, Синимекс
@kyusupov33

Angular-RU - russian-speaking community

More useful information on the topics of Angular, TypeScript, NativeScript, Dart, etc.

Russia https://angular.su t.me/angular_ru

Repositories 36 People 33 Teams 12 Projects 3 Settings

Pinned repositories

Customize pinned repositories

angular-universal-starter

A simple Angular Universal repo with Angular 6+

TypeScript ★ 132 52

angular-cli-webpack

Webpack configuration modifier for @angular/cli

TypeScript ★ 39 8

change-detection-tree

Visual detecting changes in the component tree

TypeScript ★ 21 5

angular-ru-interview-questions

Вопросы на собеседовании по Angular

★ 58 23

angular-quick-starter

Учебные материалы на русском

★ 26 4

angular-awesome-list

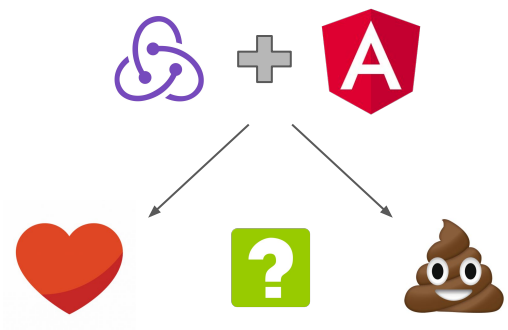
Список ресурсов по Angular на русском

★ 56 14

Для кого мы разрабатываем на Angular

Основные вопросы

Зачем нам
Redux в Angular?



Существующие
реализации

NGXS

Akita

Немного
статистики

Зачем нам Redux в Angular?

Дерево компонентов

Data-binding

One-time

Первая инициализация в представлении без последующих изменений

One-way

Изменения в представлении никак не влияют на модель, без обновления

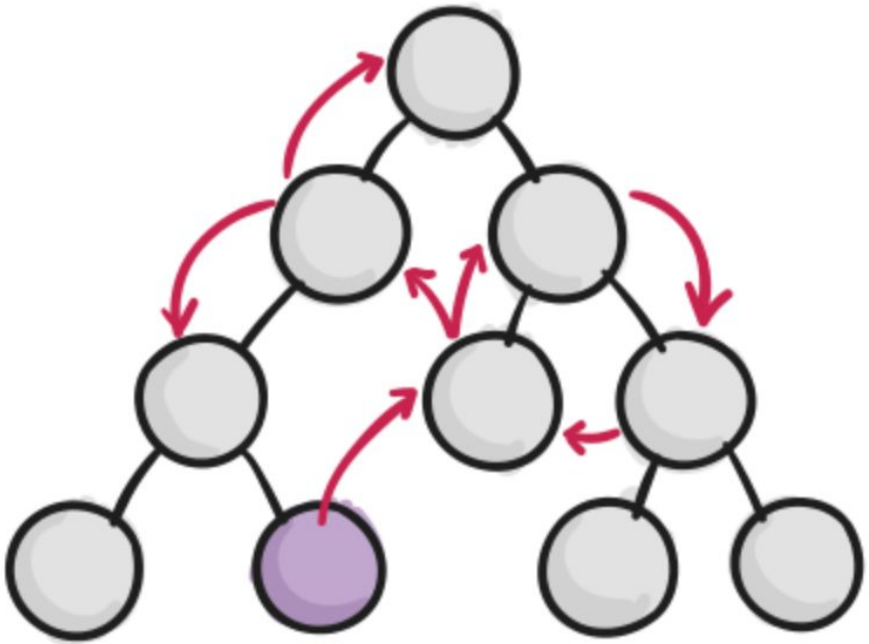
Two-way

Изменения в представлении или модели автоматически обновляют зависимость

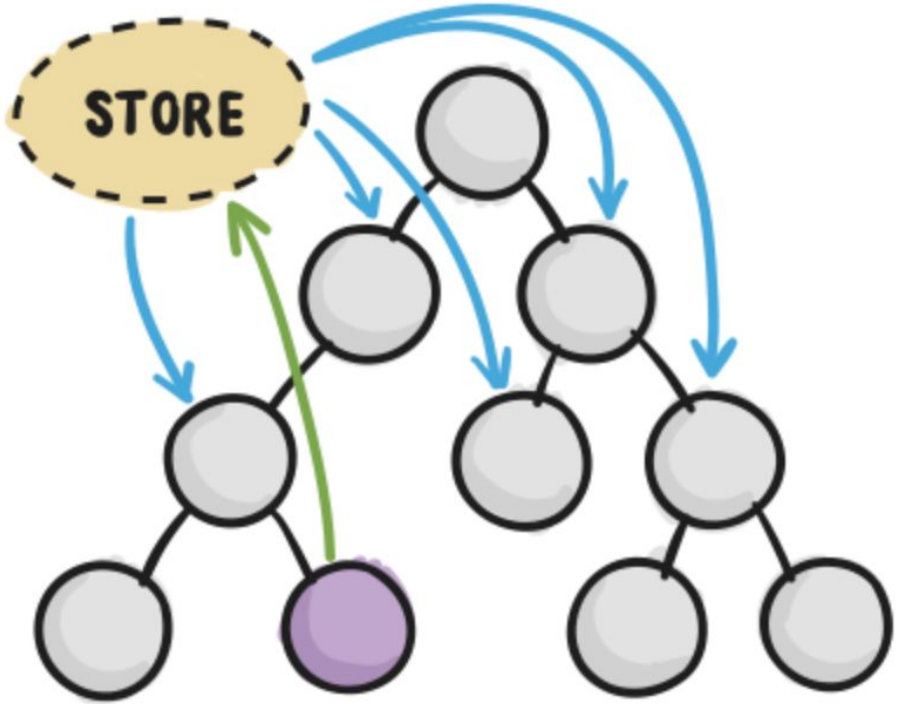
Angular as React-like

```jsx
// React (JSX)
<input value={this.state.data} onChange={this.handleChange} />

// Angular as React-like (one-way)
<input [ngModel]="data" (ngModelChange)="handleChange($event)">
```

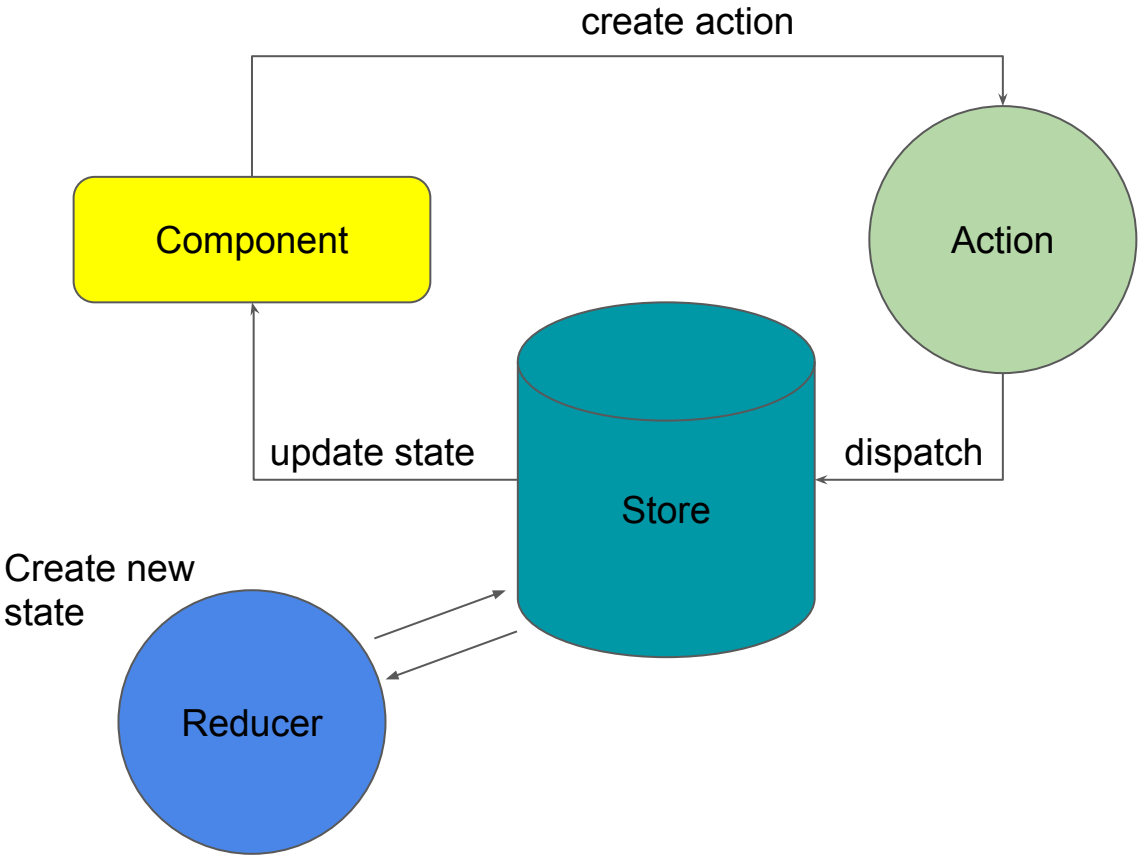
Без Redux



Redux

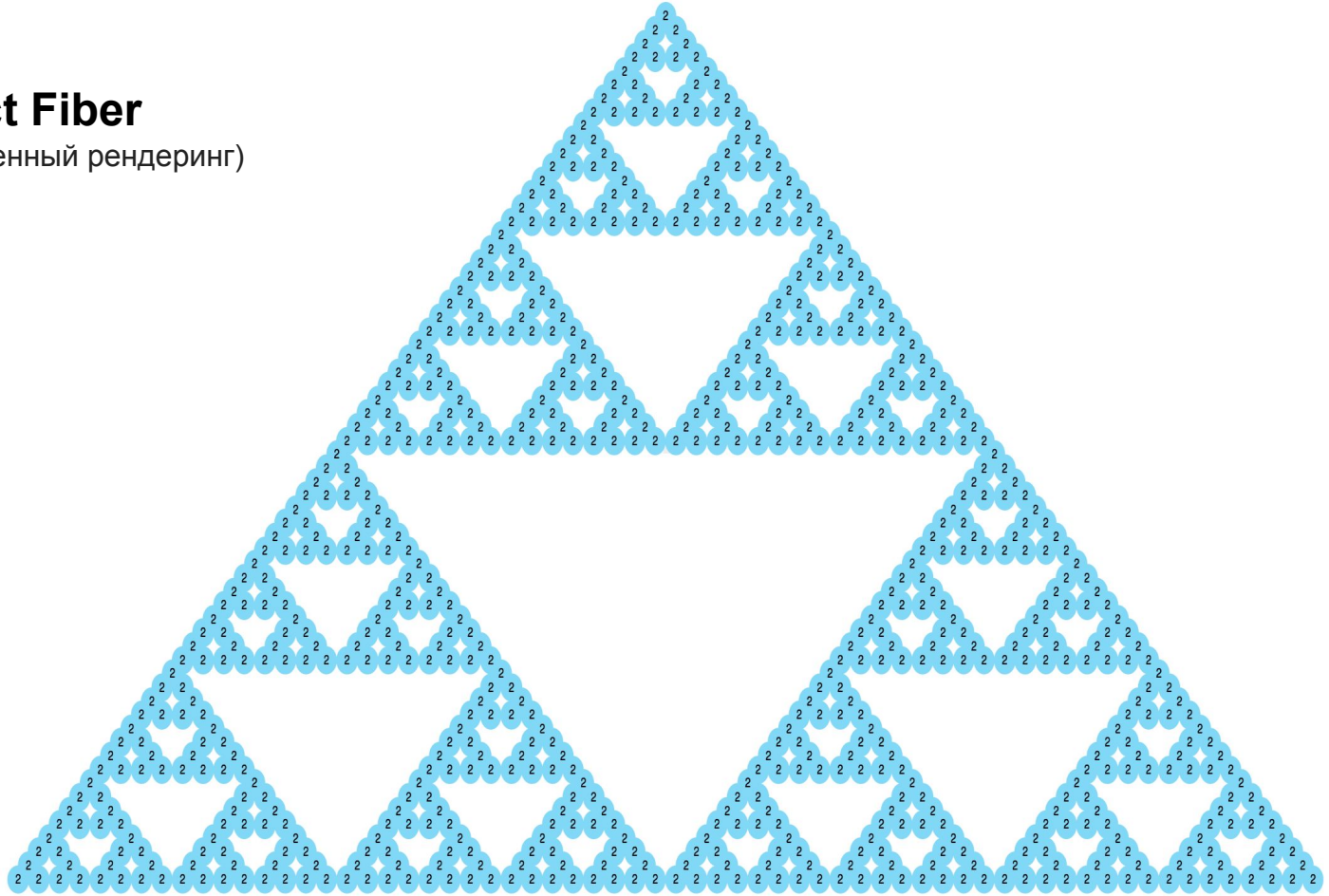


Концепция Redux



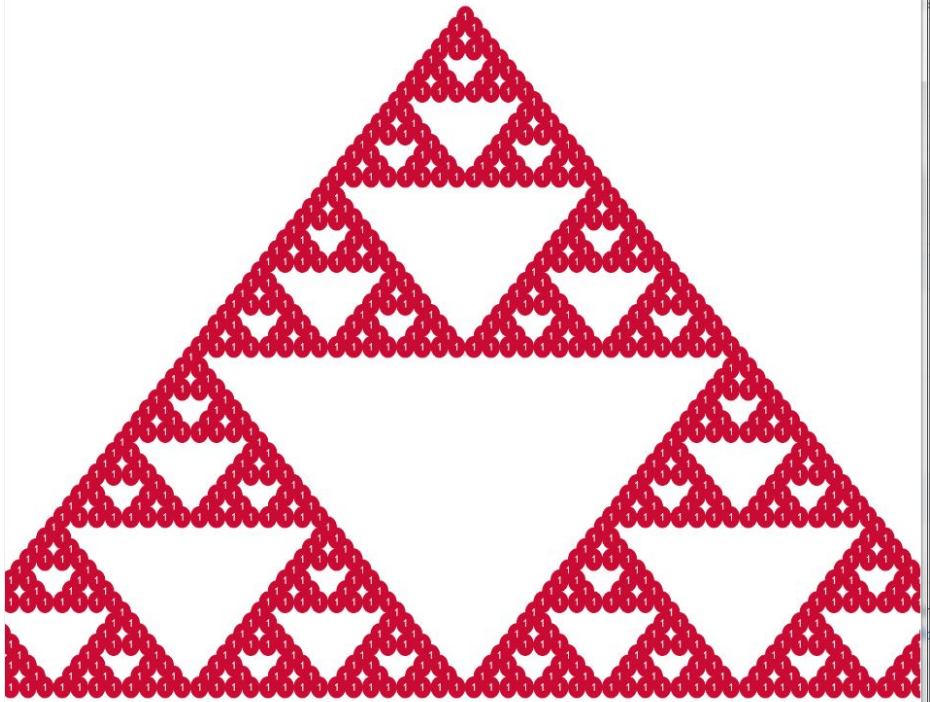
React Fiber

(отложенный рендеринг)

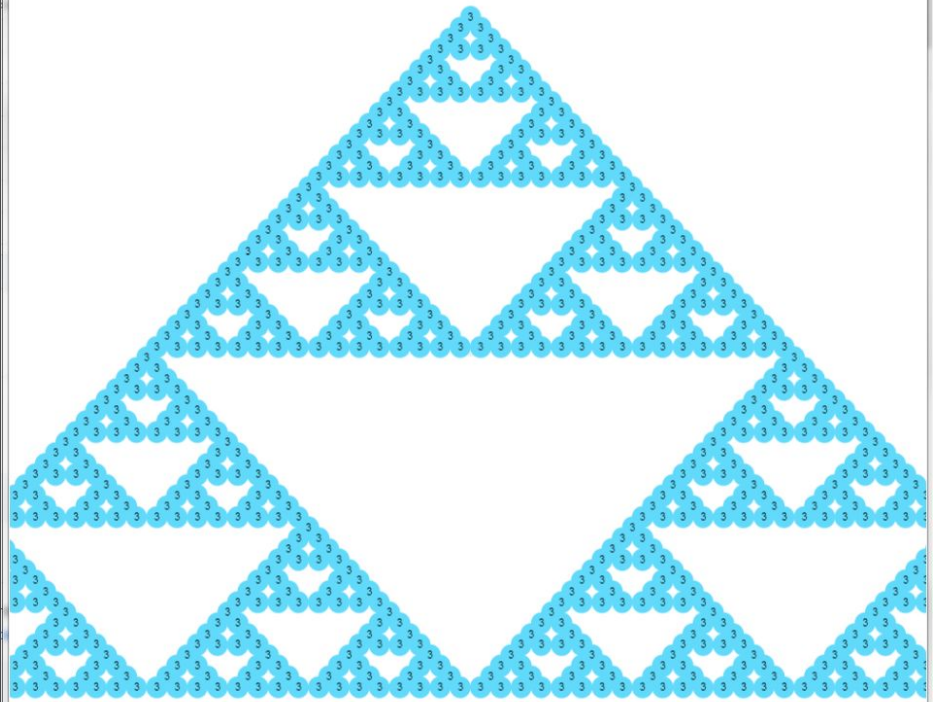


https://claudiopro.github.io/react-fiber-vs-stack-demo/fiber.html

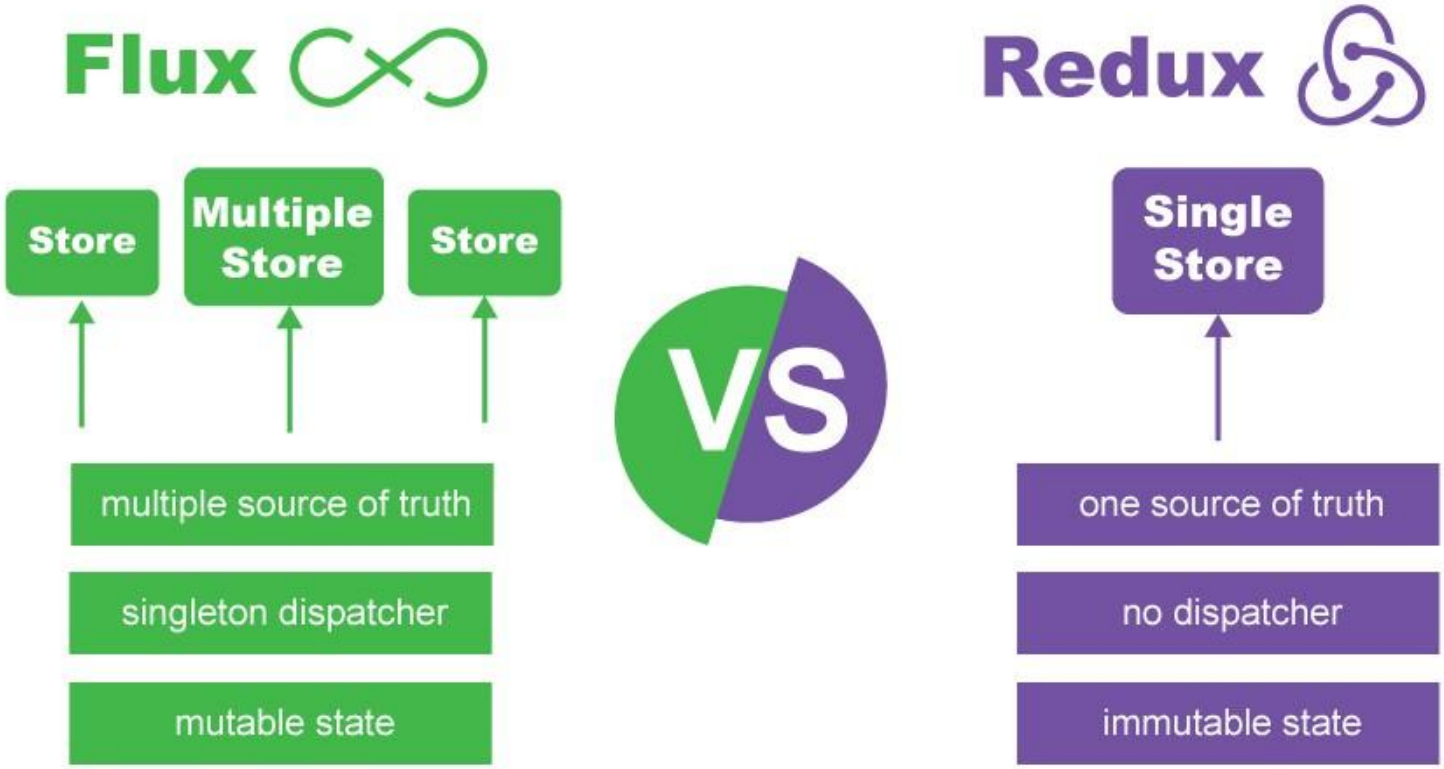
Angular 6 Example



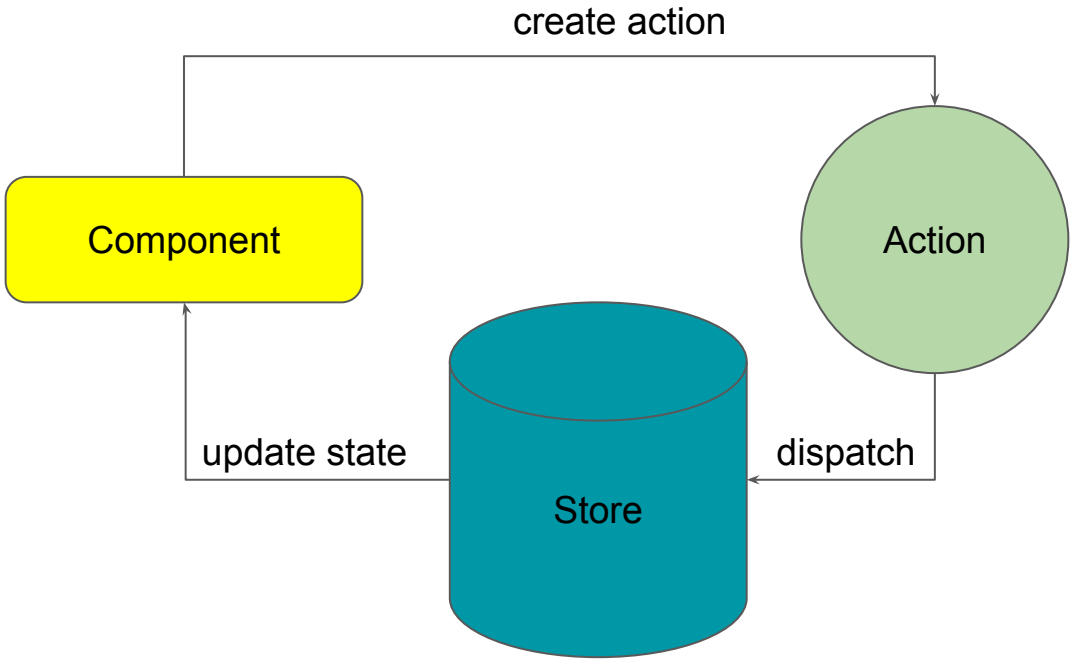
Fiber Example



DEMO


Flux
Store
Multiple Store
Store
multiple source of truth
singleton dispatcher
mutable state
VS
Redux
Single Store
one source of truth
no dispatcher
immutable state

Концепция Flux



Управление состоянием без регистрации и СМС

1. Пишем свой класс Store.ts
   который управляет потоком данных RxJS
2. Пишем state-сервис, который инициализирует поток из Store.ts
3. Инжектим state-сервис в компонентах

1. Пишем свой класс Store.ts

```ts
// store.ts
import { BehaviorSubject, Observable} from 'rxjs';


export class Store<T> {
  private _state$: BehaviorSubject<T>;

  protected constructor (initialState: T) {
    this._state$ = new BehaviorSubject(initialState);
  }

  public get state$ (): Observable<T> {
    return this._state$.asObservable();
  }

  public get state (): T {
    return this._state$.getValue();
  }

  public setState(nextState: T): void {
    this._state$.next(nextState);
  }
}
```

```
// todo-store.service.ts
export class TodoState {
  public todos: string[] = [];
}

@Injectable()
export class TodoStoreService extends Store<TodoState> {
   constructor () {
    super(new TodoState());
  }

  public addTodo(payload: string): void {
    this.setState({
      ...this.state,
      todos: [ ...this.state.todos, payload ]
    });
  }
}
```

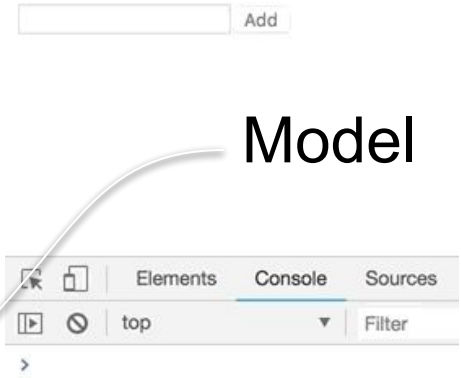
3. Инжектим state-сервис в компонентах

```
// app.component.ts
import { Component } from '@angular/core';
import { TodoStoreService } from './store/todo-store.service';

@Component({
  selector: 'my-app',
  template: `
  <div>
    <input [(ngModel)]="todo" type="text">
    <button [disabled]="!todo" (click)="addTodo(todo)">Add</button>

    <ul>
      <li *ngFor="let todo of (todoStoreService.state$ | async).todos">
        {{ todo }}
      </li>
    </ul>
  </div>
  `
})
export class AppComponent  {
  public todo: string;

  constructor(public todoStoreService: TodoStoreService) {}

  public addTodo(todo: string) {
    this.todoStoreService.addTodo(todo);
  }

}
```

getState

Model

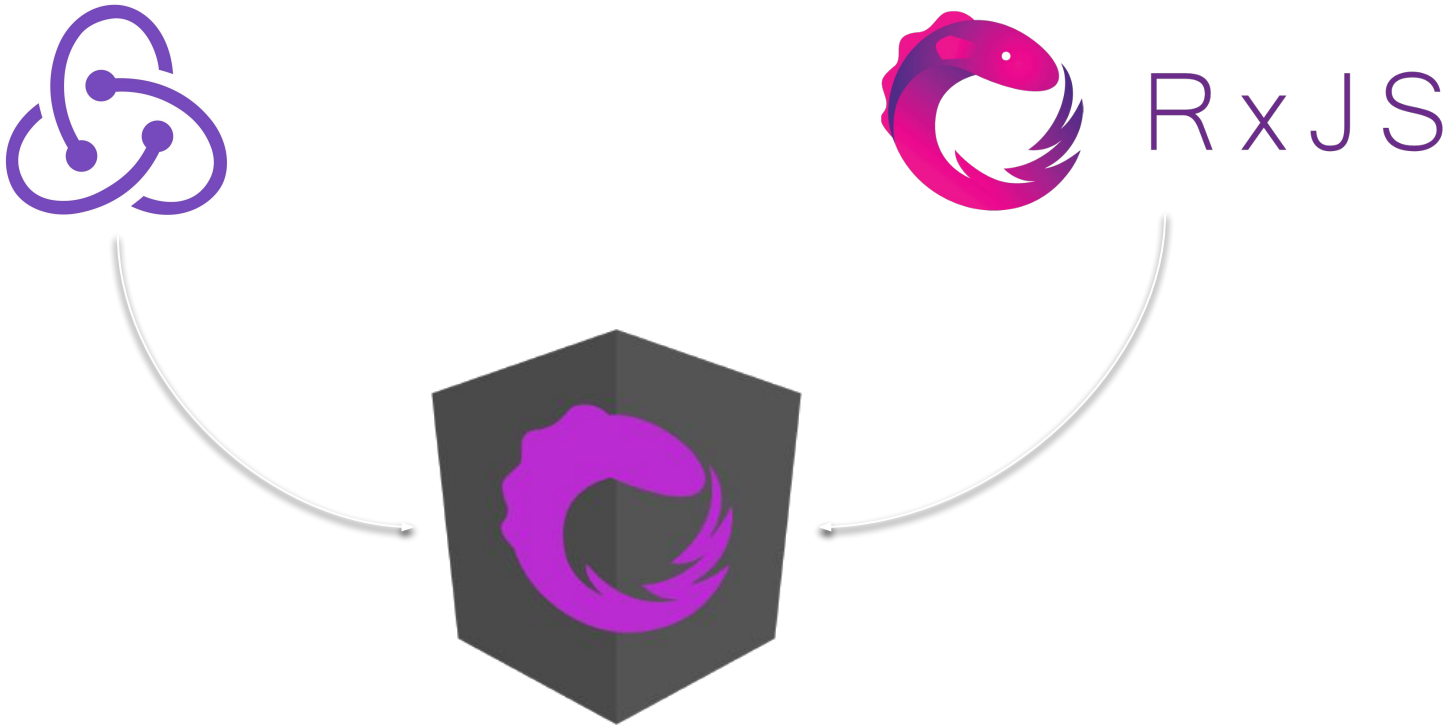
setState



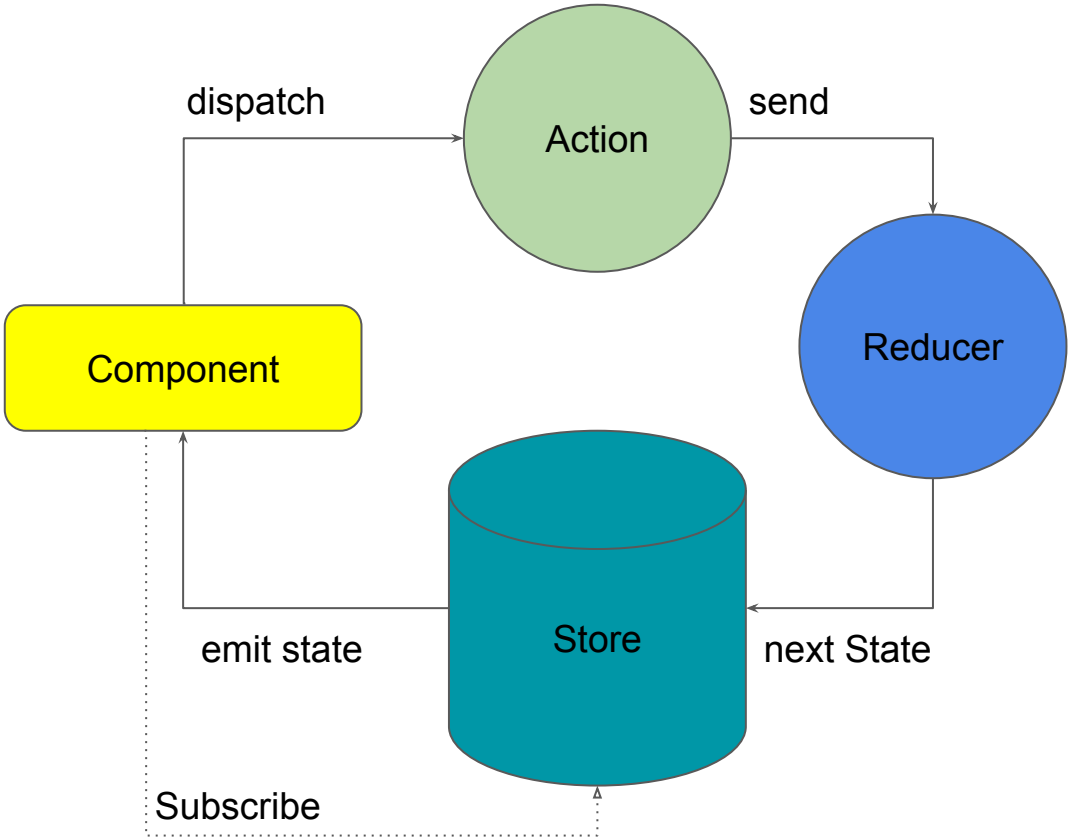
А как же:

- Dev tools?
- Logger?
- CLI?
- Immutable?

Существующие реализации паттерна Redux на Angular


RxJS

Концепция


dispatch
Action
send
Component
Reducer
Store
emit state
next State
Subscribe

Структура приложения

```
-app
  -components
  -store
    -actions
    -reducers
    -selectors
    app.reducer.ts
  app.component.html
  app.component.scss
  app.component.spec.ts
  app.component.ts
  app.module.ts
```

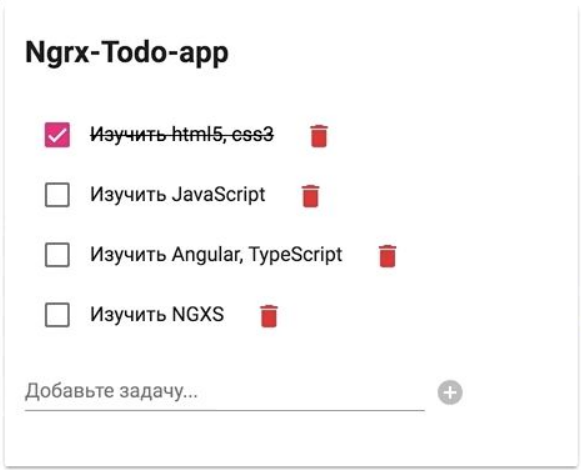
```ts
// store/app.reducer.ts
export interface AppState {
  todos: Todo[];
}


export const rootReducer: ActionReducerMap<AppState> = {
  todos: TodosReducer
};
```

```ts
// app.module.ts
import { StoreModule } from '@ngrx/store';

@NgModule({
  imports: [
    StoreModule.forRoot(rootReducer)
  ]
})
```

```
import { Action } from '@ngrx/store';
import * as TodoActions from '../actions/todo.actions';

const initialState: Todo[] = [
  ...initialTodos
];

export type Reducer<T> = (state: T, action: Action) => T;

export const TodosReducer: Reducer<Todo[]> = (state: Todo[] = initialState, action: TodoActions.TodoActionType) => {
  switch (action.type) {
    case TodoActions.ADD_TODO: {
      return [
        ...state,
        {
          id: action.id,
          text: action.text,
          completed: false
        }
      ];
    }
  }
}
```

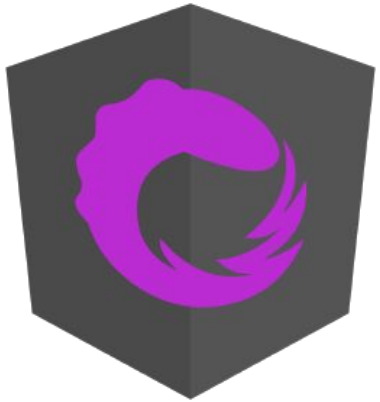
```ts
import { Store, select } from '@ngrx/store';
import * as TodoActions from '/src/app/store/actions/todo.actions';
import { AppState } from '/src/app/store/app.reducer';
import { getTodos } from '/src/app/store/selectors/todo.selectors';
import { Todo } from '/src/app/core/models/Todo';

@Component({
  selector: 'app-todo-list',
  template: `
    <app-todo [todo]="todo" *ngFor="let todo of (todos$ | async)"></app-todo>
`
})
export class TodoListComponent implements OnInit {
  public todos$: Observable<Todo[]> = this.store.pipe(select(getTodos));
  public title: string;

  constructor(private store: Store<AppState>) {}

  public ngOnInit(): void {}

  public addTodo(title: string) {
    const action = new TodoActions.AddTodoAction(title);
    this.store.dispatch(action);
    this.title = '';
  }

}
```

```ts
/**
 * @deprecated from 6.1.0. Use the pipeable `select` operator instead.
 */
select<K>(mapFn: (state: T) => K): Observable<K>;
```



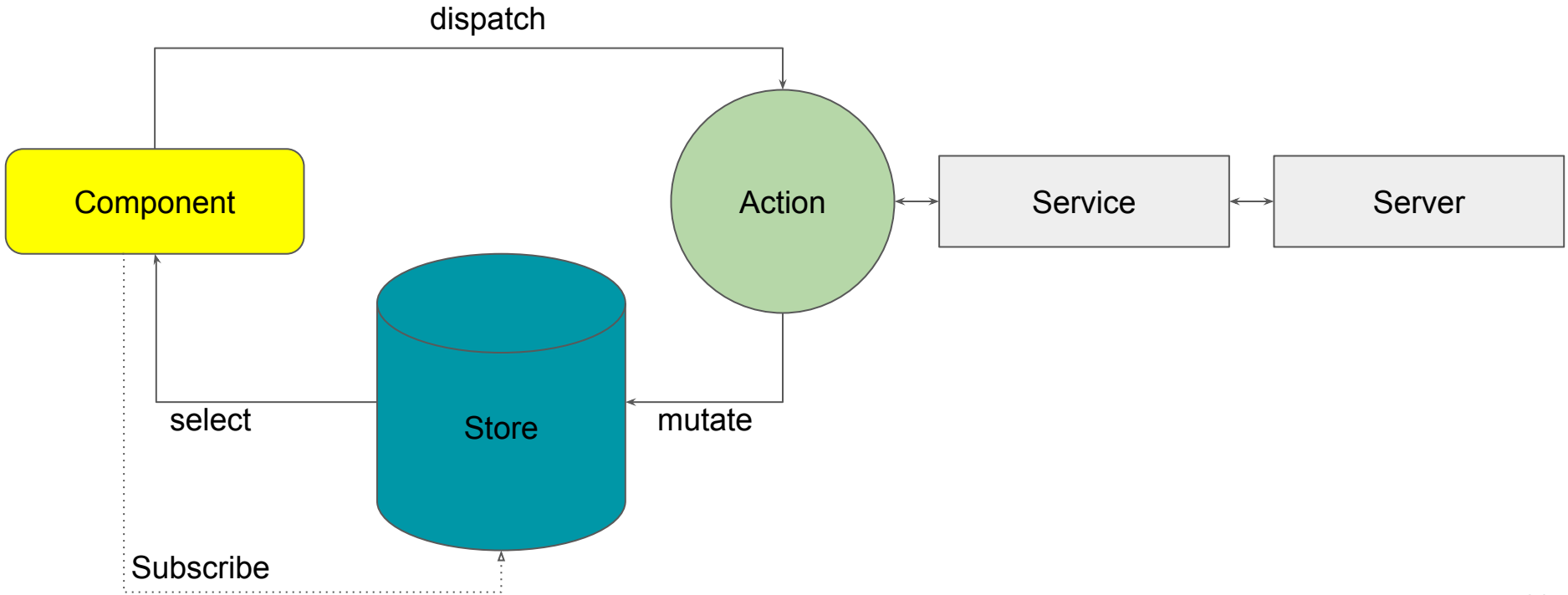
More features

1. @ngrx/effects
2. @ngrx/router-store
3. @ngrx/store-devtools
4. @ngrx/entity
5. @ngrx/schematics



— Boilerplate + class, decorators

Концепция


dispatch
Component
Action
Service
Server
Store
select
mutate
Subscribe

Структура приложения

```
-app
  -components
  -store
    -actions
        index.ts
    -states
        index.ts
  app.component.html
  app.component.scss
  app.component.spec.ts
  app.component.ts
  app.module.ts
```

1. @State
2. @Selector
3. @Actions
4. Plugins

```ts
// store/states/todo.state.ts
import { State } from '@ngxs/store';

export class TodoStateModel {

}

@State<T>(
    options: StoreOptions<T>
)

export class TodoState {
  constructor() {}
}
```

```ts
// app.module.ts
@NgModule({
  imports: [
    NgxsModule.forRoot([
      TodoState
    ])
  ]
})
```

```ts
// store/states/todo.state.ts
import { State } from '@ngxs/store';

export class TodoStateModel {

}

@State<T>(
  options: StoreOptions<T>
)

export class TodoState {
  constructor() {}
}
```

```ts
export interface StoreOptions<T> {
    /**
     * Name of the state. Required.
     */
    name: string;
    /**
     * Default values for the state.
     * If not provided, uses empty object.
     */
    defaults?: T;
    /**
     * Sub states for the given state.
     */
    children?: any[];
}
```

```
import { State } from '@ngxs/store';

export class TodoStateModel {
  public todos: Todo[];
}

@State<TodoStateModel>({
  name: 'todos',
  defaults: {
    todos: [
      ...initialTodos
    ]
  },
  children: []
})

export class TodoState {
  constructor() {}
}
```

```
export class Todo {
  public id: number;
  public completed: boolean;
  public text: string;
}
```

```
import { State } from '@ngxs/store';

export class TodoStateModel {
  public todos: Todo[];
}

@State<TodoStateModel>({
  name: 'todos',
  defaults: {
    todos: [
      ...initialTodos
    ]
  },
  children: [SubState]
})

export class TodoState {
  constructor() {}
}
```

```
@NgModule({
  imports: [
    NgxsModule.forRoot([
      TodoState,
      SubState
    ])
  ]
})
```

```
export class SubState {
  constructor() {}
}
```

Selectors

```
import { Selector } from '@ngxs/store';

export class TodoState {
  constructor() {}

  @Selector()
  public static getTodos(state: TodoStateModel)
    return state.todos;
  }

}
```

```
import { Select } from '@ngxs/store';

@Component({
  selector: 'app-todo-list',
  template: `
    <app-todo [todo]="todo" *ngFor="let todo of (todos$ | async)"></app-todo>
`
})
export class TodoListComponent implements OnInit {

  @Select(TodoState.getTodos)
  public todos$: Observable<TodoStateModel>;

  constructor() {}

  public ngOnInit(): void {
  }
}
```

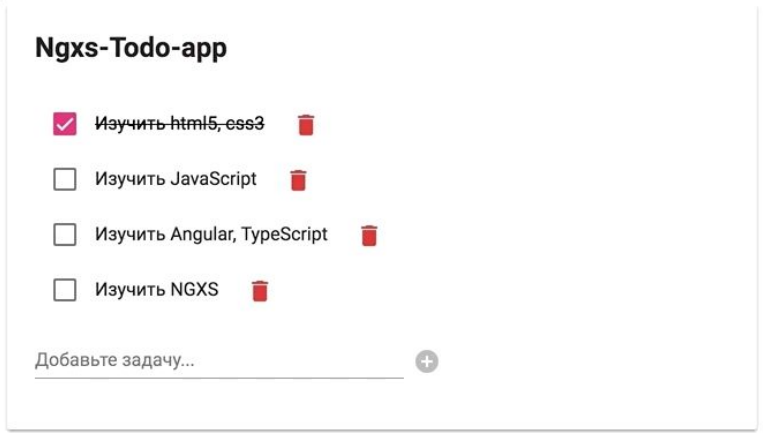
```ts
// store/actions/todo.actions.ts
export class AddTodo {
  public static readonly type = '[TODO] add';

  constructor(public payload: string) {
  }
}
```

Actions

```ts
import { State, Action, StateContext } from '@ngxs/store';
import { AddTodo } from '../actions';

export class TodoState {
  constructor() {}

  @Action(AddTodo)
  public add({ getState, patchState }: StateContext<TodoStateModel>, { payload }: AddTodo) {
    patchState({
      todos: [...getState().todos, {
        id: Math.round(Math.random() * 1000),
        text: payload,
        completed: false
      }]
    });
  }

}
```

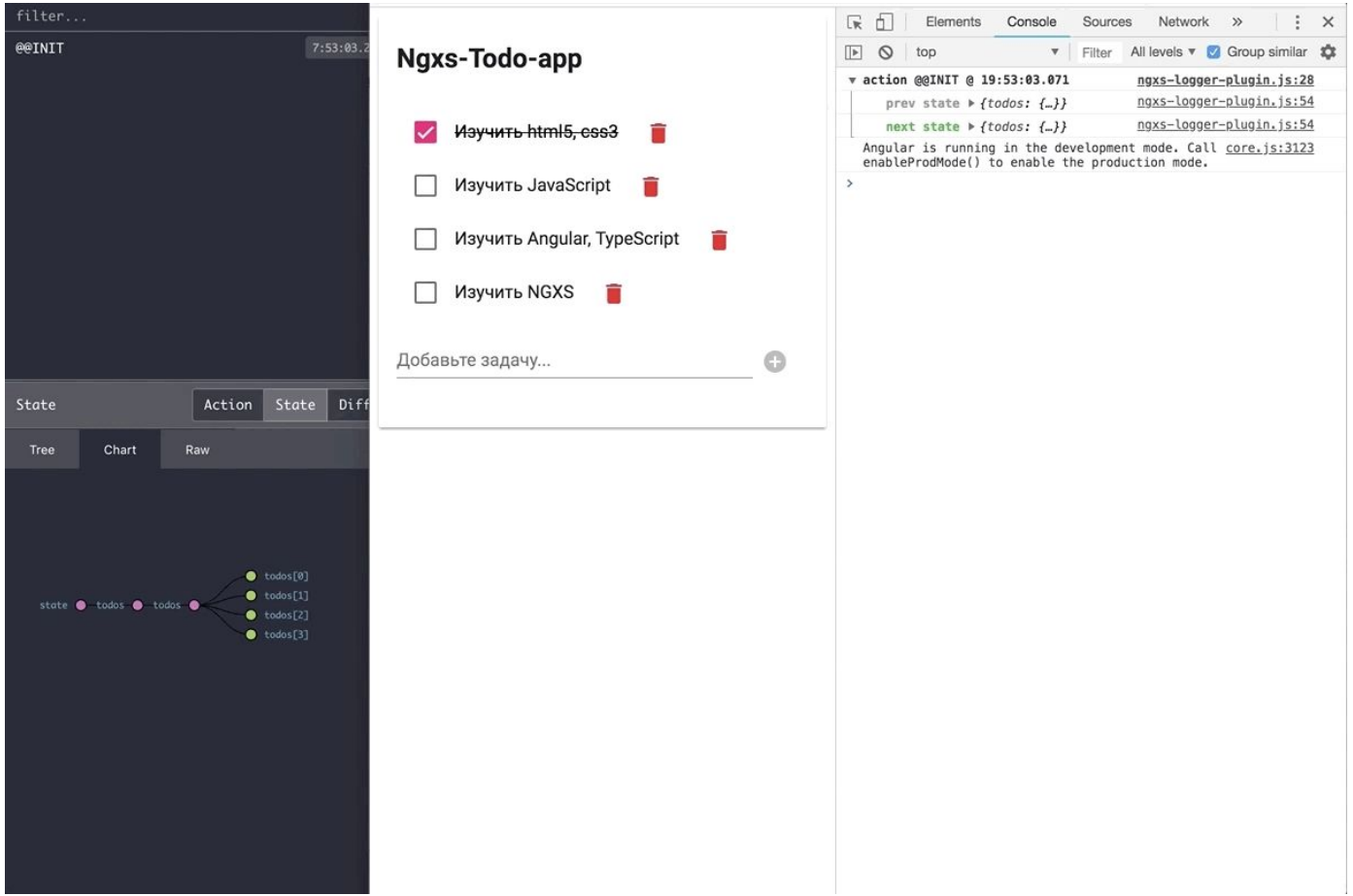
```ts
/**
 * State context provided to the actions in the state.
 */
export interface StateContext<T> {
    /**
     * Get the current state.
     */
    getState(): T;
    /**
     * Reset the state to a new value.
     */
    setState(val: T): any;
    /**
     * Patch the existing state with the provided value.
     */
    patchState(val: Partial<T>): any;
    /**
     * Dispatch a new action and return the dispatched observable.
     */
    dispatch(actions: any | any[]): Observable<void>;
}
```

```
import { Select, Store } from '@ngxs/store';
import { AddTodo } from '../../store/actions';
import { NotificationsService } from 'angular2-notifications';

@Component({
  selector: 'app-todo-list',
  templateUrl: `
    <app-todo [todo]="todo" *ngFor="let todo of (todos$ | async)"></app-todo>
`
})
export class TodoListComponent implements OnInit {
  @Select(TodoState.getTodos)
  public todos$: Observable<TodoStateModel>;
  public title: string;

  constructor(private store: Store,
              private notify: NotificationsService) {
  }

  public ngOnInit(): void {}

  public addTodo(title: string) {
    this.store.dispatch(new AddTodo(title)).subscribe(
      () => {
        this.notify.success(`Задача - ${title}, успешно добавлена!`);
      });
    this.title = '';
  }

}
```



1. Logger
2. DevTools
3. Forms
4. Storage
5. Web socket
6. Router

```
import { NgxsLoggerPluginModule } from '@ngxs/logger-plugin';
import { NgxsReduxDevtoolsPluginModule } from '@ngxs/devtools-plugin';

@NgModule({
  imports: [
    NgxsModule.forRoot([
      TodoState
    ]),
    NgxsLoggerPluginModule.forRoot(),
    NgxsReduxDevtoolsPluginModule.forRoot()
  ]
})
```

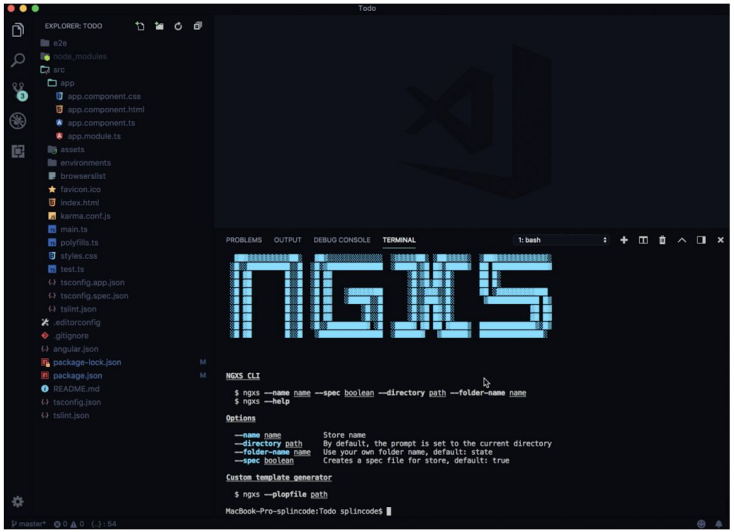

filter...
@@INIT
7:53:03.2
State
Action
State
Diff
Tree
Chart
Raw
state
todos
todos
todos[0]
todos[1]
todos[2]
todos[3]
Ngxs-Todo-app
Изучить html5, css3
Изучить JavaScript
Изучить Angular, TypeScript
Изучить NGXS
Добавьте задачу...
Elements
Console
Sources
Network
top
Filter
All levels
Group similar
action @@INIT @ 19:53:03.071
ngxs-logger-plugin.js:28
prev state {todos: {…}}
ngxs-logger-plugin.js:54
next state {todos: {…}}
ngxs-logger-plugin.js:54
Angular is running in the development mode. Call enableProdMode() to enable the production mode.
core.js:3123

@ngxs/cli

CLI

Last updated 4 days ago

Edit on GitHub



CLI Screenshot

Flux

Multiple data stores

RxJS

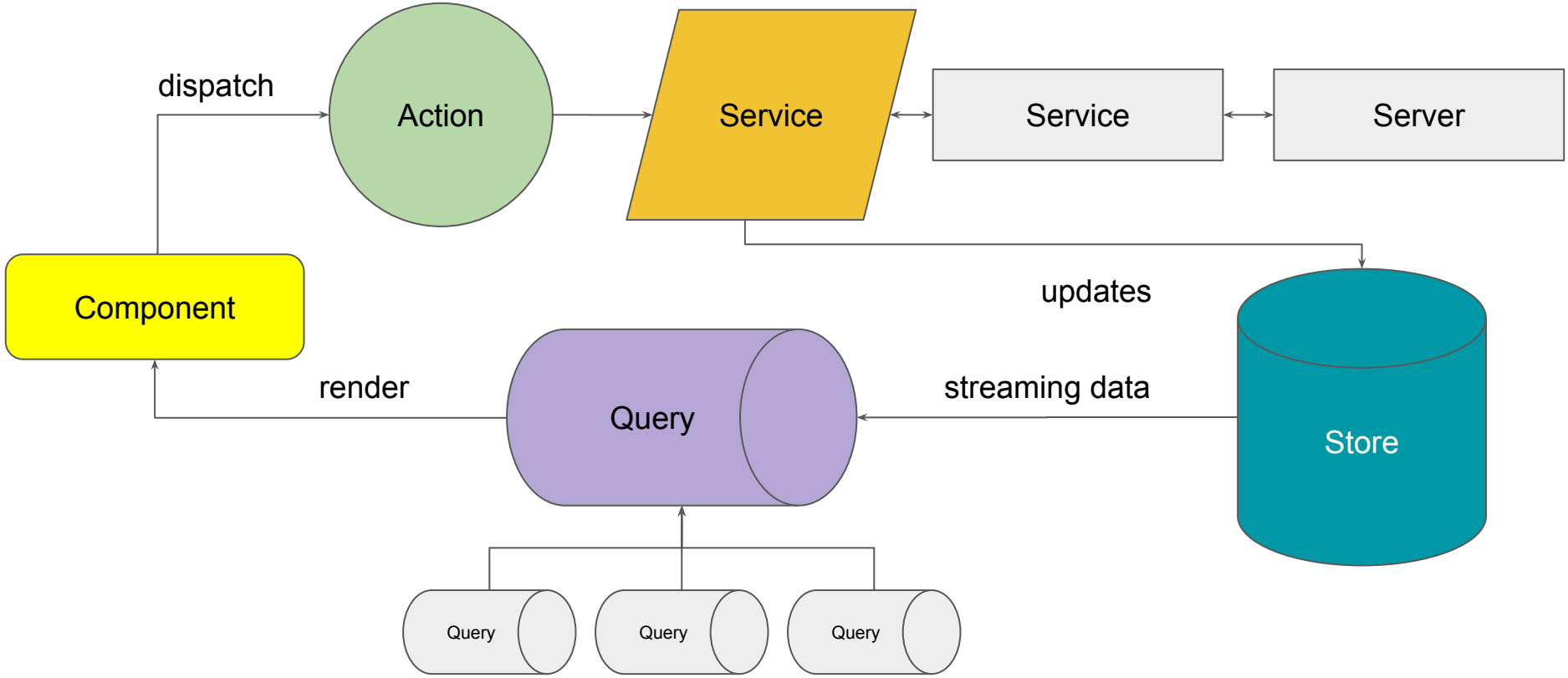
Akita

Redux

Immutable updates

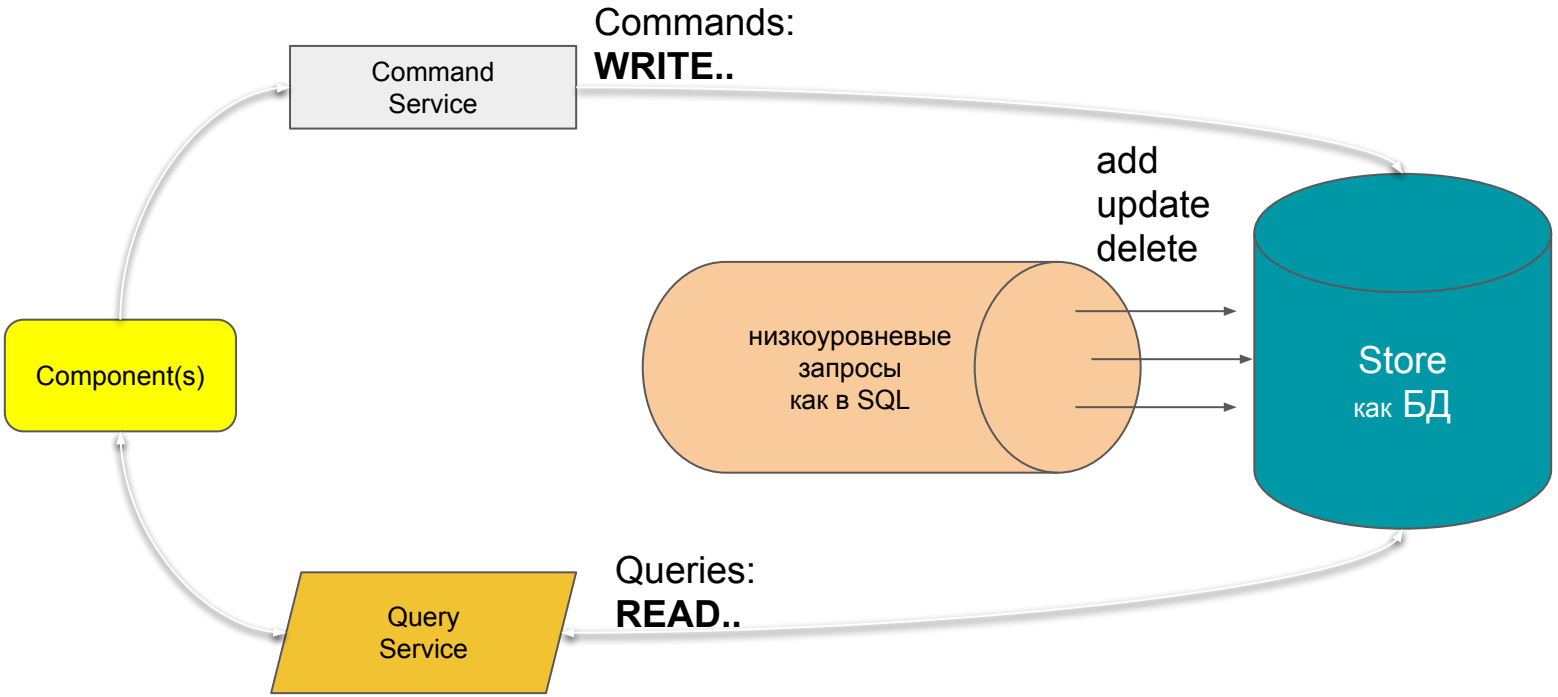
Observable Data Stores model.

Концепция

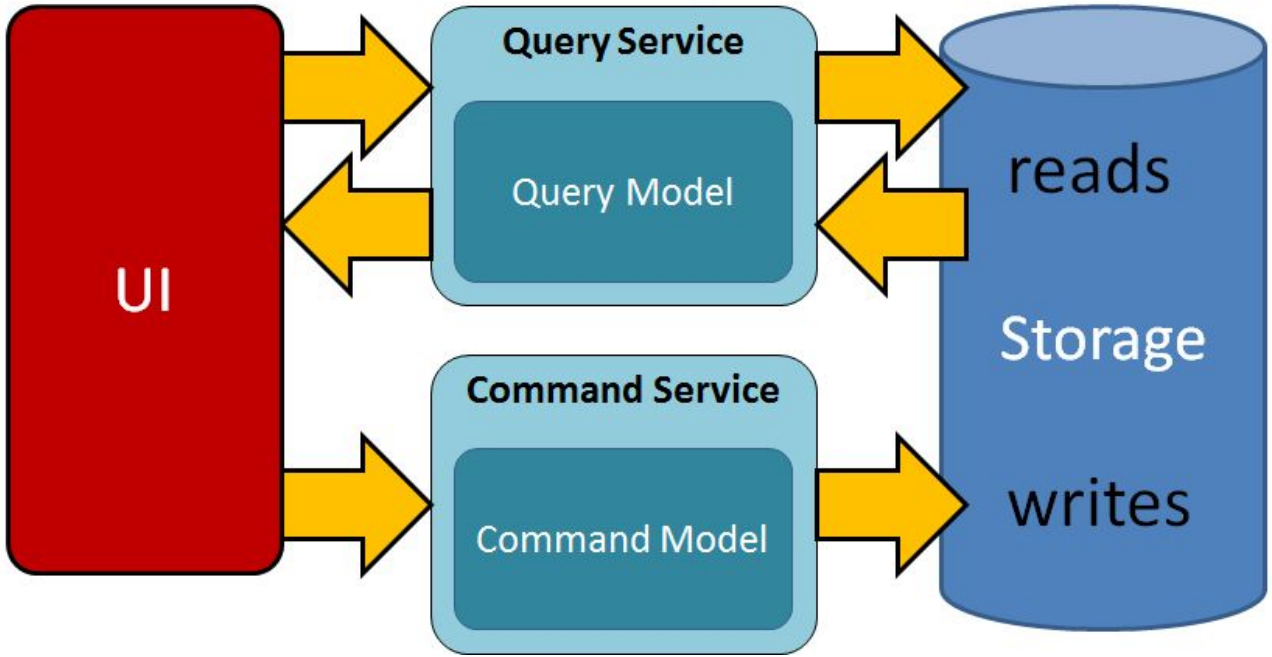


interacting with other queries

Упрощенная версия (чистый CQRS)



CQRS Pattern



Структура приложения

```
-app
  -components
  -state
     todo.model.ts
     todos.query.ts
     todos.store.ts
     todos.service.ts
  app.component.html
  app.component.scss
  app.component.spec.ts
  app.component.ts
  app.module.ts
```

DEMO

Немного статистики

Результаты

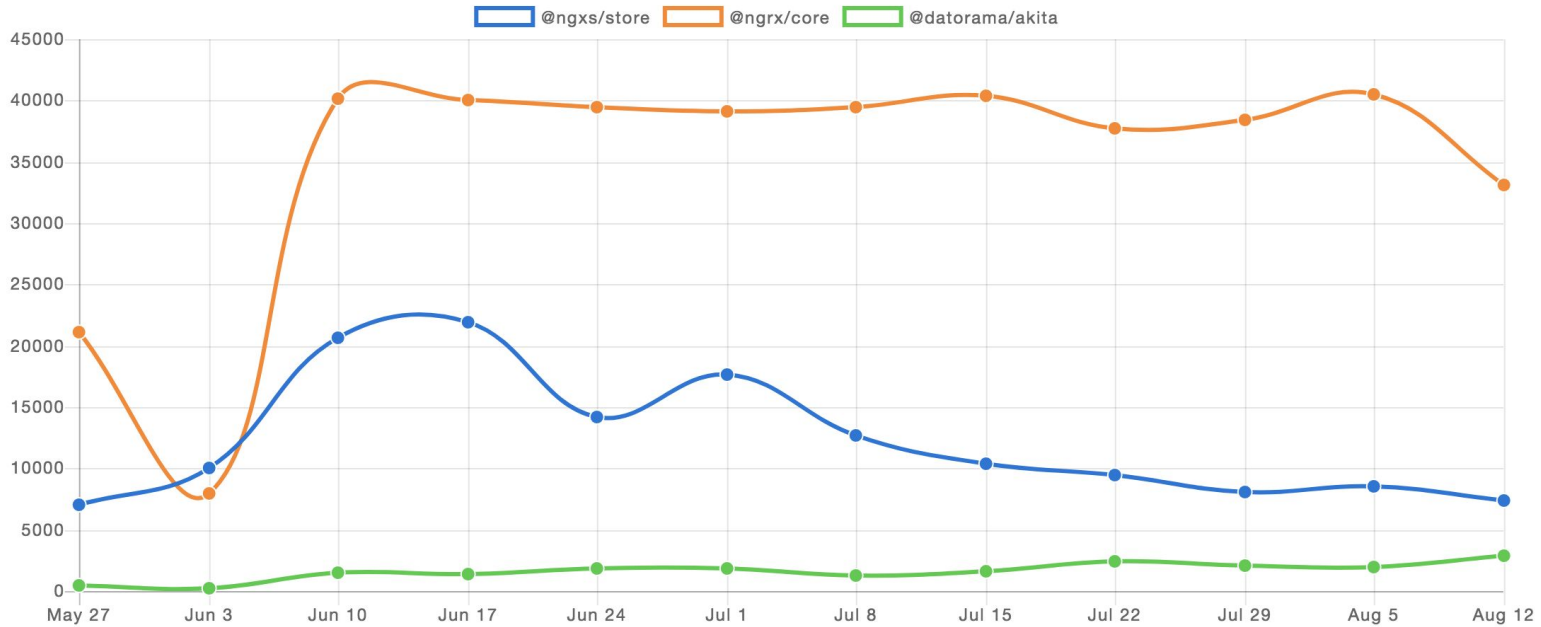
Ключевые различия		NGXS	Akita 
Github-звезды	3.4k	1.3k	371
Boilerplate	✓ ✗ ✗	✓ ✓ ✗	✓ ✗ ✗
Кривая обучения	✓ ✓ ✗	✓ ✓ ✓	✓ ✗ ✗
Простота добавления и их покрытие кода тестами	✓ ✗ ✗	✓ ✓ ✓	✓ ✓ ✗
Enterprise CQRS (aka Java)	✗ ✗ ✗	✓ ✗ ✗	✓ ✓ ✓
Документация, материалы	✗ ✗ ✗	✓ ✓ ✓	✓ ✓ ✓
CLI, Schematics, Plugins	✓ ✓ ✗	✓ ✓ ✗	✓ ✓ ✗
Подходит лучше всего:	for React bad-boys	Middle Enterprise	High Enterprise in Future

@ngxs/store vs @ngrx/core vs @datorama/akita

Enter an npm package...

@ngxs/store × @ngrx/core × @datorama/akita ×

Downloads in past 3 Months



Хороший,
Плохой,
Злой

STORE

СПАСИБО за внимание!

Полезные материалы:

https://github.com/Angular-RU/redux-demo