horizontal line

**Aspire Software Solutions**

202 Parishram Complex Navrangpura

Ahmedabad, 380009

Using Redux in Angular - The RxJs Way

**1st December 2018**

# **OVERVIEW**

This tutorial will give you a basic understanding and power of Redux through overview of it’s core functionalities. The core vision of this post to understand how to use power of redux in Angular

# **Understanding Redux**

People coming from **React** background will be familiar with concept of redux. So before jumping into it first we will learn why the combination of **React** and **Redux** is so popular.

# **How/Why redux is used with React**

Let’s first understand the principle behind React working.

**What it should do :**

* + Render HTML content with the data provided.
  + Have multiple UI states depending on the data, so that you understand why it is shown.
  + Dispatch actions on user interaction or life cycle events (conditional)
  + Animation using ReactCSSTransitionGroup or any other libraries.

**What it should not do :**

* + React is a representation library. It should give a view of the data provided to it and nothing more than that.

The Component should just display the UI, for the data that is available. If the data is not present, show a default state or an error state depending upon the scenario.The component shouldn’t have a check if the data is not available. It should dispatch an action to fetch the data.The best solution will be to fetch data from actions. The component will take care of rendering HTML content based on data provided.

**So where to store data?**

**Storing data in its local state.** You can store your data in local state needed by current component but the problem arises when it is needed in some other components too, then we have to pass deep and deep or we may broadcast it as a event. But the problem arises when our application becomes big and we have a large number of data set, then the state management becomes very difficult. And what to do when we need to pass data from child component to parent component as **React** is unidirectional. So what’s the solution for this, well here the **Store** comes into picture. What is **Store**? In simple words it is a common data tree which stores your all states. Let’s dive more deeper and bring **Redux** into picture.

# **Redux**

We will here be talking in general about Redux and believe me we will not be doing any block diagram or any sort of flow chart here, the only thing we will be focusing on will be understanding of Redux concepts in as simple manner as possible. So to start building any Redux application or converting your existing application to Redux based application, you need to understand key concepts.

* **Store**
* **Reducers**
* **State**
* **Subscribe**
* **Dispatch**

**So what are these fancy words?** No problem we will go step by step, first we will understand their concept in very simple manner using daily life example then we will dive into their programming concepts.

**Store :** Store is like the defence minister of the country, which depends on two things the **Reducer** and **State** and responds accordingly.

**Reducer :** Reducer is like the army of the country, which depends on two things the **State** and **Action** and responds accordingly.

**State :** State can be defined as current state of country.

**Subscribe :** Subscribe is like Action Base of the army.

**Dispatch :** Dispatch can be defined as, when the action of the army is needed.

Let's suppose a scenario, where there is peace in the country means it’s current **State** is Peace. Now it attacked on it’s north frontier by some invaders, so now what will happen is that, the northern frontier will contact to Defence Minister of the country which is here **Store,** the store in turn will **Dispatch** an **Action** to attack, and depending on **Action** type the current **State** of the country will be changed to Attacked.

**Now let’s do it programmatically**

We will use ***npx create-react-app react-redux*** for generating our application

Navigate to root folder of your app and install Redux using ***npm i redux --save***

Create a file inside ***src*** folder and name it as ***reduxDemo.js.*** Copy the code given below and paste it there, we will discuss the code later.

***import React, { Component } from 'react';  
import { createStore } from 'redux';  
class ReduxDemo extends Component {  
 render() {  
 var curState = 'peace';  
 // create reducer: it requires two things state and action  
 const reducer = function (state, action) {  
 if (action.type === 'attack') {  
 return action.payload;  
 }  
 return state;  
 }  
 // create store: it requires two things reducer and state  
 const store = createStore(reducer, curState);  
 // create subscriber  
 store.subscribe(() => {  
 console.log('store is now : ', store.getState());  
 });  
 // dispatch  
 store.dispatch({  
 type: 'attack',  
 payload: 'Attacked'  
 });  
 return (  
 <div>  
 Hello redux  
 </div>  
 );  
 }  
}  
export default ReduxDemo;***

Now in **App.js** just import this file and include it in the template like below.

***import React, { Component } from 'react';***

***import logo from './logo.svg';***

***import './App.css';***

***import ReduxDemo from './reduxDemo';***

***class App extends Component {***

***render() {***

***return (***

***<div className="App">***

***<header className="App-header">***

***<img src={logo} className="App-logo" alt="logo" />***

***<ReduxDemo />***

***</header>***

***</div>***

***);***

***}***

***}***

***export default App;***

So if we look to our **reduxDemo.js** file, there we have imported **createStore** from **redux** library which we have installed earlier. Now **createStore** is very important for creating our store. The **createStore** function will give **store** ability to :

* **Holds application state;**
* **Allows access to state via getState();**
* **Allows state to be updated via dispatch(action);**
* **Registers listeners via subscribe(listener);**
* **Handles unregistering of listeners via the function returned by subscribe(listener);**

Now our store is ready and we can **dispatch** our **action**, but before that we have to **subscribe** in order to get result. For subscribing we have to just call **store’s subscribe function** and it will give us the **current state** by calling **store’s getCurrentState function** according to the **action dispatched.** Now we are all set to go, we just need to **dispatch action** and we will get the **state** accordingly, above we have **dispatched action of type attack and payload Attacked** by calling **store’s dispatch function** in result it will **reset current state** with **dispatched value** and when the **getCurrentState function of store** will be called it will return the **updated state**.

So we will run the above application, in the browser console we will get result as: **store is now : Attacked**

# 

# 

# 

# 

# 

# 

# 

# **Implementing redux with Angular (Using NgRx Store)**

So far we have learnt basic concept and implementation(in react) of Redux Store. Now we will learn how to implement redux functionality in Angular, for this we will build a small project.

**Starting the project :**

* ***ng new ngrx-tutorial***
* ***cd ngrx-tutorial***
* ***npm install @ngrx/store --save***
* ***ng serve --port 6060 --open***

**Creating the Model :**

Create the following folder and file: ***/src/app/todos/models/todos.ts*** and place inside of it the following code:

***export class ToDos {***

***id: string;***

***todo: string;***

***constructor(id?: string, todo?: string) {***

***this.id = id;***

***this.todo = todo;***

***}***

***}***

**Creating the Reducer :**

Now that we have a model, we need to create a reducer. A reducer is what takes the incoming action and decides what to do with it. It takes the previous state and returns a new state based on the given action.

Create the following folder and file: ***/src/app/reducers/todo.reducer.ts*** with the following contents:

***import { ToDos } from '../todos/models/todos';***

***const initialState: ToDos = {***

***id: '1',***

***todo: 'initial to do'***

***};***

***export function reducer(state: ToDos[] = [initialState], action: any) {***

***switch (action.type) {***

***case 'Todos\_Create':***

***return [...state, action.payload];***

***case 'Todos\_Delete':***

***return state.filter(({ id }) => id !== action.id);***

***default:***

***return state;***

***}***

***}***

**Creating our Dispatcher :**

Create the following folders and files:

* ***/src/app/todos/create-todos***
  + ***create-todos.component.css***
  + ***create-todos.component.html***
  + ***create-todos.component.ts***
* ***/src/app/todos/todos-list***
  + ***todos-list.component.css***
  + ***todos-list.component.html***
  + ***todos-list.component.ts***

Create the following code inside ***/src/app/todos/create-todos/create-todos.component.html***

***<div style="max-width:300px;">***

***<h3>Create To Do's</h3>***

***<div class="form-group">***

***<input class="form-control" type="text" placeholder="To Do" #todo>***

***</div>***

***<button class="btn btn-success" (click)="saveTodo(todo.value)">Save</button>***

***</div>***

Create the following code inside ***/src/app/todos/create-todos/create-todos.component.ts***

***import { Component, OnInit } from '@angular/core';***

***import { Store } from '@ngrx/store';***

***import { ToDos } from '../models/todos';***

***@Component({***

***selector: 'app-create-todos',***

***templateUrl: './create-todos.component.html',***

***styleUrls: ['./create-todos.component.css']***

***})***

***export class CreateTodosComponent implements OnInit {***

***constructor(private store: Store<ToDos[]>) { }***

***ngOnInit() {***

***}***

***saveTodo(todo) {***

***this.store.dispatch({***

***type: 'Todos\_Create',***

***payload: {***

***id: 'todo\_' + this.makeid(),***

***todo: todo***

***}***

***});***

***}***

***makeid = () => {***

***var text = "";***

***var possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";***

***for (var i = 0; i < 5; i++)***

***text += possible.charAt(Math.floor(Math.random() \* possible.length));***

***return text;***

***}***

***}***

Create the following code inside ***/src/app/todos/create-todos/todos-list.component.html***

***<div \*ngIf="todos">***

***<h2>To Do's</h2>***

***<table class="table table-bordered">***

***<thead>***

***<tr>***

***<th>ID</th>***

***<th>To Do</th>***

***<th>Action</th>***

***</tr>***

***</thead>***

***<tbody>***

***<tr \*ngFor="let todo of todos | async">***

***<td \*ngIf="todo">{{todo.id}}</td>***

***<td \*ngIf="todo">{{todo.todo}}</td>***

***<td \*ngIf="todo" (click)='removeTodo(todo.id)'>Delete</td>***

***</tr>***

***</tbody>***

***</table>***

***</div>***

Create the following code inside ***/src/app/todos/create-todos/todos-list.component.ts***

***import { Component, OnInit } from '@angular/core';***

***import { Observable } from 'rxjs';***

***import { Store } from '@ngrx/store';***

***import { ToDos } from '../models/todos';***

***@Component({***

***selector: 'app-todos-list',***

***templateUrl: './todos-list.component.html',***

***styleUrls: ['./todos-list.component.css']***

***})***

***export class TodosListComponent implements OnInit {***

***todos: Observable<ToDos[]>;***

***constructor(private store: Store<ToDos[]>) {***

***this.todos = store.select('todos');***

***}***

***ngOnInit() {***

***}***

***removeTodo(id) {***

***this.store.dispatch({***

***type: 'Todos\_Delete',***

***id: id***

***});***

***}***

***}***

**Update our App.module and App.component :**

Update the following code inside ***/src/app/app.component.html***

***<div class="container">***

***<div class="row">***

***<div class="col-sm-4">***

***<app-create-todos></app-create-todos>***

***</div>***

***</div>***

***<div class="row">***

***<div class="col-sm-12">***

***<app-todos-list></app-todos-list>***

***</div>***

***</div>***

***</div>***

Update the following code inside ***/src/app/app.module.ts***

***import { BrowserModule } from '@angular/platform-browser';***

***import { NgModule } from '@angular/core';***

***import { StoreModule } from '@ngrx/store';***

***import { reducer } from './reducers/todo.reducer';***

***import { AppComponent } from './app.component';***

***import { TodosListComponent } from './todos/todos-list/todos-list.component';***

***import { CreateTodosComponent } from './todos/create-todos/create-todos.component';***

***@NgModule({***

***declarations: [***

***AppComponent,***

***CreateTodosComponent,***

***TodosListComponent***

***],***

***imports: [***

***BrowserModule,***

***StoreModule.forRoot({***

***todos: reducer***

***})***

***],***

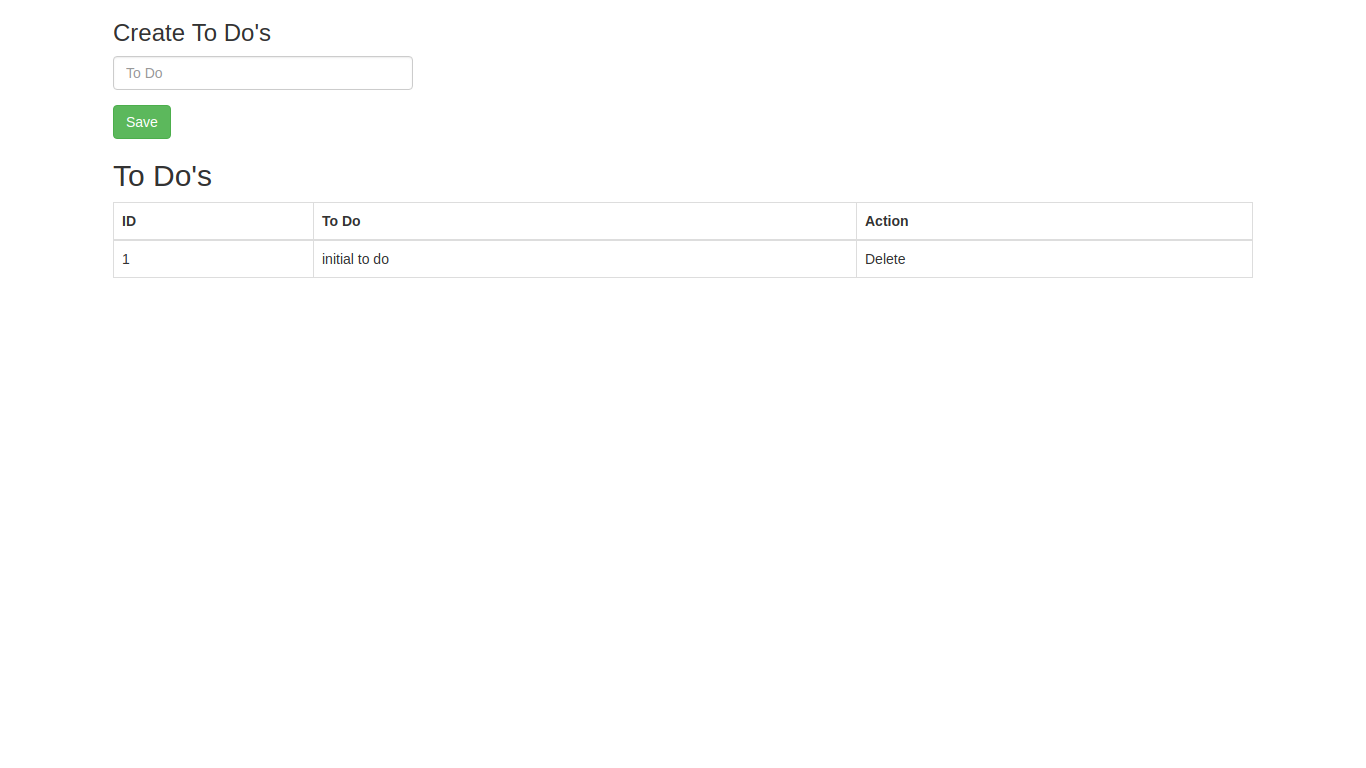
***providers: [],***

***bootstrap: [AppComponent]***

***})***

***export class AppModule { }***

Now our project is ready, if there is no error you will find it running on port 6060. Navigate to <http://localhost:6060/> , following page will be displayed:



We are ready to create out first to do, enter your data in given input field and hit save, in result and action will be dispatched with **type Todos\_Create** and **payload To Do data**, on receiving this action the **reducer** will update the current to do list and our inserted to do will be displayed inside the given data table.

**Conclusion :**

That’s **Using Redux in Angular - The RxJs Way,** Of course, there's a lot more to it, but the sole purpose of this blog was to give you all basic understanding of Redux working and its implementation in React and Angular.

For more exciting topic on Redux and NgRx Store you can visit:

* [Redux](https://redux.js.org/)
* [RxJs](https://rxjs-dev.firebaseapp.com/guide/subject)
* [ngrx Store](https://github.com/ngrx)

You can find the above example project on:

* [React Redux Example](https://github.com/ashutosh-aspire/react-redux-example)
* [NgRx Example](https://github.com/ashutosh-aspire/ngrx-store-example)