React Native Redux Tutorial 1.2

ReactNavigation 4.x (12/09/19)

INSTALACIÓN

1.- Creamos un nuevo proyecto React Native

```
react-native init reduxNativeDam
```

2.- Instalamos los paquetes Redux

```
npm install --save redux react-redux
```

3.- Instalamos React Navigation

```
npm install --save react-navigation
```

4.- Instalamos React Navigation Addons

```
npm install react-native-reanimated react-native-gesture-handler react-native-screens --save
```

5.- Instalamos React Navigation Addons, cambios en la versión 4

```
npm install react-navigation-stack react-navigation-tabs
```

6.- En **android/app/build.gradle** añadimos

```
implementation 'androidx.appcompat:appcompat:1.1.0-rc01'
implementation 'androidx.swiperefreshlayout:swiperefreshlayout:1.1.0-alpha02'
```

7.- En **MainActivity.java:**

```
package com.reactnavigation.example;
import com.facebook.react.ReactActivity;
+ import com.facebook.react.ReactActivityDelegate;
+ import com.facebook.react.ReactRootView;
+ import
com.swmansion.gesturehandler.react.RNGestureHandlerEnabledRootView;

public class MainActivity extends ReactActivity {

  @Override
  protected String getMainComponentName() {
    return "Example";
  }

+  @Override
+  protected ReactActivityDelegate createReactActivityDelegate() {
+    return new ReactActivityDelegate(this, getMainComponentName()) {
+      @Override
+      protected ReactRootView createRootView() {
+       return new RNGestureHandlerEnabledRootView(MainActivity.this);
+      }
+    };
+  }
}
```

8.- Testeamos la app

```
react-native run-android
```

ESTRUCTURA

1.- Creamos una directorio **src** que contiene:

- carpeta **components** con los componentes de la app
- carpeta **redux** que a su vez contiene 2 carpetas
    - **actions**
    - **reducers**

2.- Dentro de components creamos dos ficheros embrionales **Home.js** y **Screen.js**

Home.js

```
import React, { Component } from 'react'
import { Text, View } from 'react-native'

export default class Home extends Component {
 render() {
   return (
     <View>
       <Text> Home component </Text>
     </View>
   )
 }
}
```

Screen.js

```
import React, { Component } from 'react'
import { Text, View } from 'react-native'

export default class Screen extends Component {
 render() {
   return (
     <View>
       <Text> Screen component </Text>
     </View>
   )
 }
}
```

NAVIGATION

1.- Configuramos nuestra navegación en el fichero **App.js**

```
import React, { Component } from 'react';
import { createAppContainer } from "react-navigation";
import { createMaterialTopTabNavigator } from "react-navigation-tabs";
import { createStackNavigator } from "react-navigation-stack";
import Home from './src/components/Home';
import Screen from './src/components/Screen';

const AppNavigator = createMaterialTopTabNavigator(
 {
   Home: Home,
   Screen: Screen
 },
 {
   tabBarPosition: 'bottom',
   initialRouteName: "Home",
   tabBarOptions: {
     tabStyle: {
       backgroundColor: '#FFF',
     },
     style: {
       backgroundColor: 'FFF',
     },
     showLabel: true,
     activeTintColor: '#000',
     inactiveTintColor: '#cccccc'
   },
 }
);

const RootStack = createStackNavigator(
 {
   Main: {
     screen: AppNavigator
   }
 }
);
const AppContainer = createAppContainer(RootStack);
export default class App extends Component {
 render() {
   return <AppContainer />;
 }
}
```

ACTIONS

Dentro de la carpeta redux/actions creamos **types.js** y **indexActions.js**

Dentro del fichero **types** almacenamos unas constantes de tipo string que sólo describen el tipo de action.

types.js

```
export const INCREMENT_VALUE = 'INCREMENT_VALUE';
export const DECREMENT_VALUE = 'DECREMENT_VALUE';
```

En el fichero indexActions creamos las funciones que devuelven un action.

indexActions.js

```
import { INCREMENT_VALUE, DECREMENT_VALUE } from './types';

export const incrementValue = () => {
   return {
       type: INCREMENT_VALUE,
       payload: null
   }
}

export const decrementValue = () => {
   return {
       type: DECREMENT_VALUE,
       payload: null
   }
}
```

REDUCERS

Las funciones reductoras (reducers) toman 2 parámetros (estado y action) y devuelven un nuevo estado. La función reductora decide cómo afecta al estado cada action.

Podemos definir cuantos reducers queramos para modularizar nuestro estado global.

Dentro de la carpeta redux/reducers creamos el fichero **valueReducer.js**

```
import { INCREMENT_VALUE, DECREMENT_VALUE } from '../actions/types';

const initialState = {
   counter: 0
}
const valueReducer = (state = initialState, action) => {
   switch(action.type) {
       case INCREMENT_VALUE:
           return {
               counter: state.counter + 1
           };
       case DECREMENT_VALUE:
           return {
               counter: state.counter - 1
           }
       default:
           return state;
   }
}
export default valueReducer;
```

STORE

La creación del store es el resultado de la llamada al método **createStore** de *redux*. El método **createStore** toma un reducer como primer argumento, pero en nuestro caso hemos hecho uso de **combineReducers** para poder inyectar varios reducers.

Los reducers son los que generan el estado de nuestra aplicación.

```
import { createStore, combineReducers, applyMiddleware } from 'redux';
import valueReducer from './src/redux/reducers/valueReducer';

const rootReducer = combineReducers({
   counter: valueReducer,
});

const configureStore = () => {
   return createStore(rootReducer, applyMiddleware(logger));
}

const logger = store => next => action => {
   console.log('dispatching', action)
   console.log('Current state', store.getState())
   let result = next(action)
   console.log('Next state', store.getState())
   return result
}

export default configureStore;
```

PROVIDER

Redux mantiene el estado global para toda la aplicación. Al envolver la aplicación con el Provider todos los componentes descendientes pueden usar connect para acceder al store de Redux. Modificamos **index.js**

```
/** @format */
import * as React from 'react';
import {AppRegistry} from 'react-native';
import App from './App';
import {name as appName} from './app.json';
import { Provider } from 'react-redux';
import configureStore from './store';

const store = configureStore();

export default function Main() {
   return (
       <Provider store = { store }>
           <App />
       </Provider>
   );
}

AppRegistry.registerComponent(appName, () => Main);
```

REFACTOR STYLES

Home.js

```
import React, { Component } from 'react'
import { Text, View, Button, StyleSheet, TouchableOpacity } from 'react-native'

export default class Home extends Component {

 _decrement = () => {

 }

 _increment = () => {

 }

 render() {
   return (
     <View style={styles.mainContainer}>
       <Text> Home component </Text>
       <View style={styles.buttonContainer}>
         <TouchableOpacity
           onPress={this._decrement}
           style={styles.button}
         >
           <Text style={styles.text}>-</Text>
         </TouchableOpacity>
           <Text style={styles.text}>0</Text>
         <TouchableOpacity
           onPress={this._increment}
           style={styles.button}
         >
           <Text style={styles.text}>+</Text>
         </TouchableOpacity>
       </View>

     </View>
   )
 }
}

const styles = StyleSheet.create({
 mainContainer: {
   flex: 1,
   alignItems: "center",
```

```
    justifyContent: "center"
  },
  buttonContainer: {
    width: "100%",
    justifyContent: "space-around",
    flexDirection: "row",
    alignItems: "center"
  },
  button: {
    borderWidth:1,
    borderColor:'rgba(0,0,0,0.2)',
    alignItems:'center',
    justifyContent:'center',
    width:100,
    height:100,
    backgroundColor:'#FFF',
    borderRadius:100
  },
  text: {
    fontSize: 20,
    fontWeight: 'bold'
  }
});
```

Screen.js

```
import React, { Component } from 'react'
import { Text, View, Button, StyleSheet, TouchableOpacity } from 'react-native'

export default class Screen extends Component {

  _decrement = () => {

  }

  _increment = () => {

  }

  render() {
    return (
      <View style={styles.mainContainer}>
        <Text> Screen component </Text>
        <View style={styles.buttonContainer}>
          <TouchableOpacity
```

```
            onPress={this._decrement}
            style={styles.button}
          >
            <Text style={styles.text}>-</Text>
          </TouchableOpacity>
            <Text style={styles.text}>0</Text>
          <TouchableOpacity
            onPress={this._increment}
            style={styles.button}
          >
            <Text style={styles.text}>+</Text>
          </TouchableOpacity>
        </View>

      </View>
    )
  }
}

const styles = StyleSheet.create({
  mainContainer: {
    flex: 1,
    alignItems: "center",
    justifyContent: "center"
  },
  buttonContainer: {
    width: "100%",
    justifyContent: "space-around",
    flexDirection: "row",
    alignItems: "center"
  },
  button: {
    borderWidth:1,
    borderColor:'rgba(0,0,0,0.2)',
    alignItems:'center',
    justifyContent:'center',
    width:100,
    height:100,
    backgroundColor:'#FFF',
    borderRadius:100
  },
  text: {
    fontSize: 20,
    fontWeight: 'bold'
  }
});
```

CONNECT

El primer argumento que se le pasa a **connect** es **mapStateToProps** que se usa para seleccionar la parte de datos del **store** que el componente conectado necesita.

1.- Se le llama cada vez que el **store** cambia
2.- Recibe el **store** completo y debe devolver un objeto con los datos que necesita el componente

El segundo argumento, **mapDispatchToProps** se utiliza para enviar acciones al **store**. **Dispatch** es una función del **store** de Redux y de esta manera podemos disparar un cambio en el estado global (**store**)
La única forma de acceder al store es mediante **connect.**

Modificamos el fichero **Screen.js** y también el fichero **Home.js** que sería igual que este.

Screen.js

```
import React, { Component } from 'react'
import { Text, View, Button, StyleSheet, TouchableOpacity } from 'react-native'
import { connect } from 'react-redux';
import { incrementValue, decrementValue } from '../redux/actions/indexActions';

class Screen extends Component {
 constructor(props) {
   super(props);
 }

 _decrement = () => {
   this.props.removeValueFromCounter();
   console.log(this.props);
 }

 _increment = () => {
   this.props.addValueToCounter();
 }

 render() {
   return (
     <View style={styles.mainContainer}>
       <Text> Screen component </Text>
       <View style={styles.buttonContainer}>
         <TouchableOpacity
           onPress={this._decrement}
           style={styles.button}
```

```
          >
            <Text style={styles.text}>-</Text>
          </TouchableOpacity>
            <Text style={styles.text}>{this.props.counter.counter}</Text>
          <TouchableOpacity
            onPress={this._increment}
            style={styles.button}
          >
            <Text style={styles.text}>+</Text>
          </TouchableOpacity>
        </View>

      </View>
    )
  }
}

const styles = StyleSheet.create({
  mainContainer: {
    flex: 1,
    alignItems: "center",
    justifyContent: "center"
  },
  buttonContainer: {
    width: "100%",
    justifyContent: "space-around",
    flexDirection: "row",
    alignItems: "center"
  },
  button: {
    borderWidth:1,
    borderColor:'rgba(0,0,0,0.2)',
    alignItems:'center',
    justifyContent:'center',
    width:100,
    height:100,
    backgroundColor:'#FFF',
    borderRadius:100
  },
  text: {
    fontSize: 20,
    fontWeight: 'bold'
  }
});

const mapStateToProps = state => {
  return {
```

```
    counter: state.counter
  }
}
const mapDispatchToProps = dispatch => {
 return {
    addValueToCounter: () => {
      dispatch(incrementValue())
    },
    removeValueFromCounter: () => {
      dispatch(decrementValue())
    }
  }
}

export default connect(mapStateToProps, mapDispatchToProps)(Screen)
```