In the previous chapters, we have learnt what is Redux and how it works. Let us now check the integration of view part with Redux. You can add any view layer to Redux. We will also discuss react library and Redux.

Let us say if various react components need to display the same data in different ways without passing it as a prop to all the components from top-level component to the way down. It would be ideal to store it outside the react components. Because it helps in faster data retrieval as you need not pass data all the way down to different components.

Let us discuss how it is possible with Redux. Redux provides the react-redux package to bind react components with two utilities as given below −

* Provider
* Connect

Provider makes the store available to rest of the application. Connect function helps react component to connect to the store, responding to each change occurring in the store’s state.

Let us have a look at the **root index.js** file which creates store and uses a provider that enables the store to the rest of the app in a react-redux app.

import React from 'react'

import { render } from 'react-dom'

import { Provider } from 'react-redux'

import { createStore, applyMiddleware } from 'redux';

import reducer from './reducers/reducer'

import thunk from 'redux-thunk';

import App from './components/app'

import './index.css';

const store = createStore(

reducer,

window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_ && window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_(),

applyMiddleware(thunk)

)

render(

<Provider store = {store}>

<App />

</Provider>,

document.getElementById('root')

)

Whenever a change occurs in a react-redux app, mapStateToProps() is called. In this function, we exactly specify which state we need to provide to our react component.

With the help of connect() function explained below, we are connecting these app’s state to react component. Connect() is a high order function which takes component as a parameter. It performs certain operations and returns a new component with correct data which we finally exported.

With the help of mapStateToProps(), we provide these store states as prop to our react component. This code can be wrapped in a container component. The motive is to separate concerns like data fetching, rendering concern and reusability.

import { connect } from 'react-redux'

import Listing from '../components/listing/Listing' //react component

import makeApiCall from '../services/services' //component to make api call

const mapStateToProps = (state) => {

return {

items: state.items,

isLoading: state.isLoading

};

};

const mapDispatchToProps = (dispatch) => {

return {

fetchData: () => dispatch(makeApiCall())

};

};

export default connect(mapStateToProps, mapDispatchToProps)(Listing);

The definition of a component to make an api call in services.js file is as follows −

import axios from 'axios'

import { itemsLoading, itemsFetchDataSuccess } from '../actions/actions'

export default function makeApiCall() {

return (dispatch) => {

dispatch(itemsLoading(true));

axios.get('http://api.tvmaze.com/shows')

.then((response) => {

if (response.status !== 200) {

throw Error(response.statusText);

}

dispatch(itemsLoading(false));

return response;

})

.then((response) => dispatch(itemsFetchDataSuccess(response.data)))

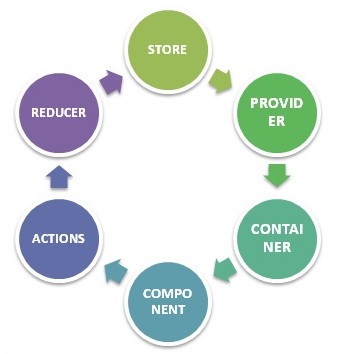
};

}

mapDispatchToProps() function receives dispatch function as a parameter and returns you callback props as plain object that you pass to your react component.

Here, you can access fetchData as a prop in your react listing component, which dispatches an action to make an API call. mapDispatchToProps() is used to dispatch an action to store. In react-redux, components cannot access the store directly. The only way is to use connect().

Let us understand how the react-redux works through the below diagram −



**STORE** − Stores all your application state as a JavaScript object

**PROVIDER** − Makes stores available

**CONTAINER** − Get apps state & provide it as a prop to components

**COMPONENT** − User interacts through view component

**ACTIONS** − Causes a change in store, it may or may not change the state of your app

**REDUCER** − Only way to change app state, accept state and action, and returns updated state.

However, Redux is an independent library and can be used with any UI layer. React-redux is the official Redux, UI binding with the react. Moreover, it encourages a good react Redux app structure. React-redux internally implements performance optimization, so that component re-render occurs only when it is needed.

To sum up, Redux is not designed to write shortest and the fastest code. It is intended to provide a predictable state management container. It helps us understand when a certain state changed, or where the data came from.