FREE eBook

LEARNING react-redux

Free unaffiliated eBook created from **Stack Overflow contributors.**

#react-redux

Table of Contents

About 1

Chapter 1: Getting started with react-redux 2

Remarks 2

Versions 2

Examples 3

Installation or Setup 3

Complete example 4

Hello World using React Redux 5

Credits 9

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: react-redux

It is an unofficial and free react-redux ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official react-redux.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with react-redux

Remarks

React Redux is a library which provides React bindings for Redux.

React components aware of the Redux store are called "Containers", "Smart Components" or "Higher Order Component" (HOC). Such components, to use Redux, need to:

- Subscribe to the store to get updates from Redux store
- Dispatch actions

Doing this by hand would imply using `store.subscribe` and `store.dispatch(action)` in React containers.

React Redux simplifies the binding between the Redux store and a React container component by way of the `connect` function, which maps Redux state properties and Action creators to the component's props.

`connect` is a function that creates a higher order component. Connect accepts 3 functions (`mapStateToProps`, `mapDispatchToProps`, `mergeProps`) and returns a container component, that wraps the original component to make turn it into a "connected" component:

```
import { connect } from 'react-redux';

const Customers = { ... };
const mapStateToProps = (state) => { ... }
const mapDispatchToProps = (dispatch) => { ... }

export default connect(mapStateToProps, mapDispatchToProps)(Customers);
```

See the examples section for a complete example.

Since all container componenents need to access the Redux store, the recommended way is to use a special `<Provider>` component of React Redux, which passes the store to all the children components (internally using React context).

Official documentation: http://redux.js.org/docs/basics/UsageWithReact.html

GitHub repo: https://github.com/reactjs/react-redux

Versions

Version	Release Date
5.0.3	2017-02-23
5.0.2	2017-01-11

Version	Release Date
5.0.1	2016-12-14
5.0.0	2016-12-14
4.4.6	2016-11-14
4.4.5	2016-04-14
4.4.4	2016-04-13
4.4.3	2016-04-12
4.4.0	2016-02-06
4.3.0	2016-02-05
4.2.0	2016-02-01
4.1.0	2016-01-28
4.0.0	2015-10-15
3.0.0	2015-09-24
2.0.0	2015-09-01
1.0.0	2015-08-24
0.5.0	2015-08-07
0.1.0	2015-07-12

Examples

Installation or Setup

Using `redux` directly with `react` might seem little difficult, As for every `component` you want to update when store changes, you have to subscribe that component to the `redux store`

React Redux takes care of all these and makes it really easy to write components that can request the data it needs from `redux store` and be notified Only when those data changes., This allows us to write really effective components.

To install `react-redux` all you have to do is run this `npm` command

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

And you're done.

Note: React Redux is dependent on

- React (Version 0.14 or later) and
- Redux

Complete example

Suppose we have container "CustomersContainer" which connects a "Customers" dumb component to the Redux store.

In index.js:

```
import { Component }, React from 'react';
import { render } from 'react-dom';
import { Provider } from 'react-redux';
import { createStore } from 'redux';
import rootReducer from './redux/rootReducer';
import CustomersContainer from './containers/CustomersContainer';

let store = createStore(rootReducer);

render(
  <Provider store={store}>
    <CustomersContainer />
  </Provider>,
  document.getElementById('root')
);
```

In CustomersContainer:

```
import React, { Component } from 'react';
import { connect } from 'react-redux';

// Import action creators
import { fetchCustomers } from '../redux/actions';

// Import dumb component
import Customers from '../components/Customers';

// ES6 class declaration
class CustomersContainer extends Component {
  componentWillMount() {
    // Action fetchCustomers mapped to prop fetchCustomers
    this.props.fetchCustomers();
  }

  render() {
    return <Customers customers={this.props.customers} />;
  }
}

function mapStateToProps(state) {
  return {
```

```
    customers: state.customers
  };
}

// Here we use the shorthand notation for mapDispatchToProps
// it can be used when the props and action creators have the same name
const CustomersContainer = connect(mapStateToProps, { fetchCustomers })(CustomersContainer);

export default CustomersContainer;
```

Hello World using React Redux

This guide assumes you have already installed `react`, `redux`, `react-router` and **`react-redux`** and have configured `react`, `redux` and `react-router`., If you haven't, Please do so.

Note: While `react-router` in not a dependency of `react-redux`, It's very likely that we will using it in our react application for routing and this makes it really easy for us to use react-redux.

FILENAME: `app.js`

```
'use strict';

import React from 'react';
import { render } from 'react-dom';
import { Router, Route, Link, browserHistory, IndexRoute } from 'react-router';
import { Provider } from 'react-redux';
import store from './stores';

render(
  (
    <Provider store={ store }>
      <Router history={ browserHistory }>
        {/* all the routes here */}
      </Router>
    </Provider>
  ),
  document.getElementById('app')
);
```

This file will make sense to most of you, What we're doing here is getting the **store** from `./stores` and passing it to all the routes using **Higher Order Component** **`Provider`** provided by `react-redux`.

This makes the **store** available throughout our application.

Now, let's consider this scenario. We have a component `UserComponent` which gets the data from `user` reducer and has a button which when clicked updates the data in the store.

Application Structure

Our `rootReducer` has `user` reducer

```
const rootReducer = combineReducers({
```

```
    user: userReducer,
})
export default rootReducer;
```

Our `userReducer` looks like this

```
const default_state = {
  users: [],
  current_user: {
    name: 'John Doe',
    email: 'john.doe@gmail.com',
    gender: 'Male'
  },
  etc: {}
};

function userReducer( state=default_state, action ) {

  if ( action.type === "UPDATE_CURRENT_USER_DATA" ) {
    return Object.assign( {}, state, { current_user: Object.assign( {}, state.current_user, {
[action.payload.field]: action.payload.value } ) } );
  }
  else {
    return state;
  }

}

export default userReducer;
```

And our `actions` file looks something like this

```
export function updateCurrentUserData( data ) {
  return {
    type: "UPDATE_CURRENT_USER_DATA",
    payload: data
  }
}
```

Finally, Lets work on our component

FILENAME: `UserComponent.js`

```
'use strict';

import React from 'react';
import { connect } from 'react-redux';
import * as Action from './actions';

let UserComponent = (props) => {

  let changeUserDetails = (field, value) => {
    // do nothing
  }

  return(
```

```
    <div>
      <h1>Hello { props.current_user.name }</h1>
      <p>Your email address is { props.current_user.email }</p>
      <div style={{ marginTop: 30 }}>
        <button onClick={ () => { changeUserDetails('name', 'Jame Smith') } }>Change
Name</button>
        <button onClick={ () => { changeUserDetails('email', 'jane@gmail.com') } }>Change
Email Address</button>
      </div>
    </div>
  )

}

export default UserComponent;
```

Of course this **won't work**, As we haven't connected it to the store yet.

> In case you're wondering, this is a **stateless functional component**, since we're using `redux` and we don't really need an internal state for our component, this is the right time to use it.

The `connect` method provided by `react-redux` takes in three parameters

mapStateToProps, **mapDispatchToProps** and the **Component** itself.

```
connect( mapStateToProps, mapDispatchToProps )(Component)
```

Let's add **connect** to our component **UserComponent** along with **mapStateToProps** and **mapDispatchToProps**

And let's also update our **changeUserDetails** function, so when called, It will `dispatch` an `action` to our `reducers`, and based on the type of action our reducer will kick in and make changes to the store, and once the store updated `react-redux` will re-render our component with the new data.

Sounds complicated? It really isn't.

Our **`UserComponent.js`** will look like

```
'use strict';

import React from 'react';
import { connect } from 'react-redux';
import * as Action from './actions';

const mapStateToProps = ( state, ownProps ) => {
  return {
    current_user: state.user.current_user,
  }
}

const mapDispatchToProps = ( dispatch, ownProps ) => {
  return {
    updateCurrentUserData: (payload) => dispatch( Action.updateCurrentUserData(payload) ),
  }
```

```
}


let UserComponent = (props) => {

  let changeUserDetails = (field, value) => {
    props.updateCurrentUserData({ field: field, value: value });
  }

  return(
    <div>
      <h1>Hello { props.current_user.name }</h1>
      <p>Your email address is { props.current_user.email }</p>
      <div style={{ marginTop: 30 }}>
        <button onClick={ () => { changeUserDetails('name', 'Jame Smith') } }>Change
Name</button>
        <button onClick={ () => { changeUserDetails('email', 'jane@gmail.com') } }>Change
Email Address</button>
      </div>
    </div>
  )

}

const ConnectedUserComponent = connect(
  mapStateToProps,
  mapDispatchToProps
)(UserComponent)

export default ConnectedUserComponent;
```

What we did here is added

- **mapStateToProps**: This allows us to get the data from store and when **that** data changes, our component will be re-rendered with the new data.

  Our component will only re-render if the data our **component** is requesting changes in the store and not when any other data changes in the store.

- **mapDispatchToProps**: This allows us to `dispatch actions` to all the reducers from our component.. (could be any component), And based on the `type` of action, our userReducer will kick in and return a new state with the updated data.

- **ConnectedUserComponent**: Lastly, we connected our component to the store using the `connect` method by passing all the parameters and `exported` the connected component.

- We also updated our **changeUserDetails** function to call `method` on props and also pass in the data., And `props` in turn dispatches the method we called to all reducers.

NOTE:

- If we don't return a new state from reducer, `react-redux` wont re-render our component.

Read Getting started with react-redux online: https://riptutorial.com/react-redux/topic/5797/getting-started-with-react-redux

Credits

S. No	Chapters	Contributors
1	Getting started with react-redux	Alexg2195, Community, Matteo Frana, Ori Drori, Random User, Thibaut Remy