|  |  |
| --- | --- |
| **ch38.ReactJS Redux Example** | **Date: 22-02-2022** |

**Topics**

React Props Validation,

# React Props Validation

React Redux Example

In this section, we will learn how to implements Redux in React application. Here, we provide a simple example to connect Redux and React.

**Step-1** Create a new react project using **create-react-app** command. I choose the project name: "**reactproject**." Now, install **Redux** and **React-Redux**.

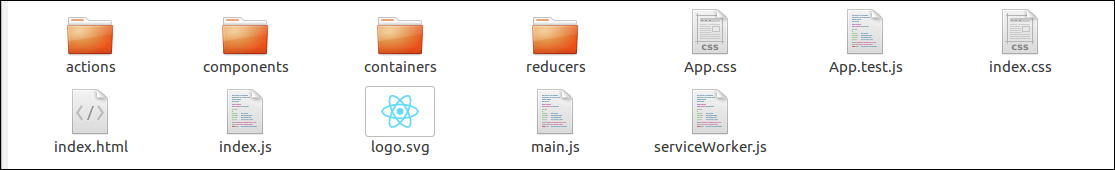
1. javatpoint@root:~/Desktop$ npx create-react-app reactproject
2. javatpoint@root:~/Desktop/reactproject$ npm install redux react-redux --save

**Step-2 Create Files and Folders**

In this step, we need to create folders and files for actions, reducers, components, and containers. After creating folders and files, our project looks like as below image.

Competitive questions on Structures in Hindi

Keep Watching



**Step-3 Actions**

It uses '**type**' property to inform about data that should be sent to the **Store**. In this folder, we will create two files: **index.js** and **index.spec.js**. Here, we have created an **action creator** that returns our action and sets an **id** for every created item.

**Index.js**

1. let nextTodoId = 0
2. export **const** addTodo = text => ({
3. type: 'ADD\_TODO',
4. id: nextTodoId++,
5. text
6. })
8. export **const** setVisibilityFilter = filter => ({
9. type: 'SET\_VISIBILITY\_FILTER',
10. filter
11. })
13. export **const** toggleTodo = id => ({
14. type: 'TOGGLE\_TODO',
15. id
16. })
18. export **const** VisibilityFilters = {
19. SHOW\_ALL: 'SHOW\_ALL',
20. SHOW\_COMPLETED: 'SHOW\_COMPLETED',
21. SHOW\_ACTIVE: 'SHOW\_ACTIVE'
22. }

**Index.spec.js**

1. **import** \* as actions from './index'
3. describe('todo actions', () => {
4. it('addTodo should create ADD\_TODO action', () => {
5. expect(actions.addTodo('Use Redux')).toEqual({
6. type: 'ADD\_TODO',
7. id: 0,
8. text: 'Use Redux'
9. })
10. })
12. it('setVisibilityFilter should create SET\_VISIBILITY\_FILTER action', () => {
13. expect(actions.setVisibilityFilter('active')).toEqual({
14. type: 'SET\_VISIBILITY\_FILTER',
15. filter: 'active'
16. })
17. })
19. it('toggleTodo should create TOGGLE\_TODO action', () => {
20. expect(actions.toggleTodo(1)).toEqual({
21. type: 'TOGGLE\_TODO',
22. id: 1
23. })
24. })
25. })

**Step-4 Reducers**

As we know, Actions only trigger changes in the app, and the Reducers specify those changes. The Reducer is a function which takes two parameters 'Action' and 'State' to calculate and return an updated State. It read the payloads from the 'Actions' and then updates the 'Store' via the State accordingly.

In the given files, each Reducer managing its own part of the global State. The State parameter is different for every Reducer and corresponds to the part of the 'State' it manages. When the app becomes larger, we can split the Reducers into separate files and keep them completely independent and managing different data domains.

Here, we are using 'combineReducers' helper function to add any new Reducers we might use in the future.

**index.js**

1. **import** { combineReducers } from 'redux'
2. **import** todos from './todos'
3. **import** visibilityFilter from './visibilityFilter'
5. export **default** combineReducers({
6. todos,
7. visibilityFilter
8. })

**Todos.js**

1. **const** todos = (state = [], action) => {
2. **switch** (action.type) {
3. **case** 'ADD\_TODO':
4. **return** [
5. ...state,
6. {
7. id: action.id,
8. text: action.text,
9. completed: **false**
10. }
11. ]
12. **case** 'TOGGLE\_TODO':
13. **return** state.map(todo =>
14. (todo.id === action.id)
15. ? {...todo, completed: !todo.completed}
16. : todo
17. )
18. **default**:
19. **return** state
20. }
21. }
22. export **default** todos

**Todos.spec.js**

1. **import** todos from './todos'
3. describe('todos reducer', () => {
4. it('should handle initial state', () => {
5. expect(
6. todos(undefined, {})
7. ).toEqual([])
8. })
10. it('should handle ADD\_TODO', () => {
11. expect(
12. todos([], {
13. type: 'ADD\_TODO',
14. text: 'Run the tests',
15. id: 0
16. })
17. ).toEqual([
18. {
19. text: 'Run the tests',
20. completed: **false**,
21. id: 0
22. }
23. ])
25. expect(
26. todos([
27. {
28. text: 'Run the tests',
29. completed: **false**,
30. id: 0
31. }
32. ], {
33. type: 'ADD\_TODO',
34. text: 'Use Redux',
35. id: 1
36. })
37. ).toEqual([
38. {
39. text: 'Run the tests',
40. completed: **false**,
41. id: 0
42. }, {
43. text: 'Use Redux',
44. completed: **false**,
45. id: 1
46. }
47. ])
49. expect(
50. todos([
51. {
52. text: 'Run the tests',
53. completed: **false**,
54. id: 0
55. }, {
56. text: 'Use Redux',
57. completed: **false**,
58. id: 1
59. }
60. ], {
61. type: 'ADD\_TODO',
62. text: 'Fix the tests',
63. id: 2
64. })
65. ).toEqual([
66. {
67. text: 'Run the tests',
68. completed: **false**,
69. id: 0
70. }, {
71. text: 'Use Redux',
72. completed: **false**,
73. id: 1
74. }, {
75. text: 'Fix the tests',
76. completed: **false**,
77. id: 2
78. }
79. ])
80. })
82. it('should handle TOGGLE\_TODO', () => {
83. expect(
84. todos([
85. {
86. text: 'Run the tests',
87. completed: **false**,
88. id: 1
89. }, {
90. text: 'Use Redux',
91. completed: **false**,
92. id: 0
93. }
94. ], {
95. type: 'TOGGLE\_TODO',
96. id: 1
97. })
98. ).toEqual([
99. {
100. text: 'Run the tests',
101. completed: **true**,
102. id: 1
103. }, {
104. text: 'Use Redux',
105. completed: **false**,
106. id: 0
107. }
108. ])
109. })
110. })

**VisibilityFilter.js**

1. **import** { VisibilityFilters } from '../actions'
3. **const** visibilityFilter = (state = VisibilityFilters.SHOW\_ALL, action) => {
4. **switch** (action.type) {
5. **case** 'SET\_VISIBILITY\_FILTER':
6. **return** action.filter
7. **default**:
8. **return** state
9. }
10. }
11. export **default** visibilityFilter

**Step-5 Components**

It is a Presentational Component, which concerned with how things look such as markup, styles. It receives data and invokes callbacks exclusively via props. It does not know where the data comes from or how to change it. It only renders what is given to them.

**App.js**

It is the root component which renders everything in the UI.

1. **import** React from 'react'
2. **import** Footer from './Footer'
3. **import** AddTodo from '../containers/AddTodo'
4. **import** VisibleTodoList from '../containers/VisibleTodoList'
6. **const** App = () => (
7. <div>
8. <AddTodo />
9. <VisibleTodoList />
10. <Footer />
11. </div>
12. )
13. export **default** App

**Footer.js**

It tells where the user changes currently visible **todos**.

1. **import** React from 'react'
2. **import** FilterLink from '../containers/FilterLink'
3. **import** { VisibilityFilters } from '../actions'
5. **const** Footer = () => (
6. <p>
7. Show: <FilterLink filter={VisibilityFilters.SHOW\_ALL}>All</FilterLink>
8. {', '}
9. <FilterLink filter={VisibilityFilters.SHOW\_ACTIVE}>Active</FilterLink>
10. {', '}
11. <FilterLink filter={VisibilityFilters.SHOW\_COMPLETED}>Completed</FilterLink>
12. </p>
13. )
14. export **default** Footer

**Link.js**

It is a link with a callback.

1. **import** React from 'react'
2. **import** PropTypes from 'prop-types'
4. **const** Link = ({ active, children, onClick }) => {
5. **if** (active) {
6. **return** <span>{children}</span>
7. }
9. **return** (
10. <a
11. href=""
12. onClick={e => {
13. e.preventDefault()
14. onClick()
15. }}
16. >
17. {children}
18. </a>
19. )
20. }
22. Link.propTypes = {
23. active: PropTypes.bool.isRequired,
24. children: PropTypes.node.isRequired,
25. onClick: PropTypes.func.isRequired
26. }
28. export **default** Link

**Todo.js**

It represents a single todo item which shows **text**.

1. **import** React from 'react'
2. **import** PropTypes from 'prop-types'
4. **const** Todo = ({ onClick, completed, text }) => (
5. <li
6. onClick={onClick}
7. style={{
8. textDecoration: completed ? 'line-through' : 'none'
9. }}
10. >
11. {text}
12. </li>
13. )
15. Todo.propTypes = {
16. onClick: PropTypes.func.isRequired,
17. completed: PropTypes.bool.isRequired,
18. text: PropTypes.string.isRequired
19. }
21. export **default** Todo

**TodoList.js**

It is a list to show visible todos{ id, text, completed }.

1. **import** React from 'react'
2. **import** PropTypes from 'prop-types'
3. **import** Todo from './Todo'
5. **const** TodoList = ({ todos, onTodoClick }) => (
6. <ul>
7. {todos.map((todo, index) => (
8. <Todo key={index} {...todo} onClick={() => onTodoClick(index)} />
9. ))}
10. </ul>
11. )
13. TodoList.propTypes = {
14. todos: PropTypes.arrayOf(
15. PropTypes.shape({
16. id: PropTypes.number.isRequired,
17. completed: PropTypes.bool.isRequired,
18. text: PropTypes.string.isRequired
19. }).isRequired
20. ).isRequired,
21. onTodoClick: PropTypes.func.isRequired
22. }
23. export **default** TodoList

**Step-6 Containers**

It is a Container Component which concerned with how things work such as data fetching, updates State. It provides data and behavior to presentational components or other container components. It uses Redux State to read data and dispatch Redux Action for updating data.

**AddTodo.js**

It contains the input field with an ADD (submit) button.

1. **import** React from 'react'
2. **import** { connect } from 'react-redux'
3. **import** { addTodo } from '../actions'
5. **const** AddTodo = ({ dispatch }) => {
6. let input
8. **return** (
9. <div>
10. <form onSubmit={e => {
11. e.preventDefault()
12. **if** (!input.value.trim()) {
13. **return**
14. }
15. dispatch(addTodo(input.value))
16. input.value = ''
17. }}>
18. <input ref={node => input = node} />
19. <button type="submit">
20. Add Todo
21. </button>
22. </form>
23. </div>
24. )
25. }
26. export **default** connect()(AddTodo)

**FilterLink.js**

It represents the current visibility filter and renders a link.

1. **import** { connect } from 'react-redux'
2. **import** { setVisibilityFilter } from '../actions'
3. **import** Link from '../components/Link'
5. **const** mapStateToProps = (state, ownProps) => ({
6. active: ownProps.filter === state.visibilityFilter
7. })
9. **const** mapDispatchToProps = (dispatch, ownProps) => ({
10. onClick: () => dispatch(setVisibilityFilter(ownProps.filter))
11. })
13. export **default** connect(
14. mapStateToProps,
15. mapDispatchToProps
16. )(Link)

**VisibleTodoList.js**

It filters the todos and renders a TodoList.

1. **import** { connect } from 'react-redux'
2. **import** { toggleTodo } from '../actions'
3. **import** TodoList from '../components/TodoList'
4. **import** { VisibilityFilters } from '../actions'
6. **const** getVisibleTodos = (todos, filter) => {
7. **switch** (filter) {
8. **case** VisibilityFilters.SHOW\_ALL:
9. **return** todos
10. **case** VisibilityFilters.SHOW\_COMPLETED:
11. **return** todos.filter(t => t.completed)
12. **case** VisibilityFilters.SHOW\_ACTIVE:
13. **return** todos.filter(t => !t.completed)
14. **default**:
15. **throw** **new** Error('Unknown filter: ' + filter)
16. }
17. }
19. **const** mapStateToProps = state => ({
20. todos: getVisibleTodos(state.todos, state.visibilityFilter)
21. })
23. **const** mapDispatchToProps = dispatch => ({
24. toggleTodo: id => dispatch(toggleTodo(id))
25. })
27. export **default** connect(
28. mapStateToProps,
29. mapDispatchToProps
30. )(TodoList)

**Step-7 Store**

All container components need access to the Redux Store to subscribe to it. For this, we need to pass it(store) as a prop to every container component. However, it gets tedious. So we recommend using special React Redux component called which make the store available to all container components without passing it explicitly. It used once when you render the root component.

**index.js**

1. **import** React from 'react'
2. **import** { render } from 'react-dom'
3. **import** { createStore } from 'redux'
4. **import** { Provider } from 'react-redux'
5. **import** App from './components/App'
6. **import** rootReducer from './reducers'
8. **const** store = createStore(rootReducer)
10. render(
11. <Provider store={store}>
12. <App />
13. </Provider>,
14. document.getElementById('root')
15. )

**Output**

When we execute the application, it gives the output as below screen.



Now, we will be able to add items in the list.



The detailed explanation of React-Redux example can be shown here: <https://redux.js.org/basics/usage-with-react>.

Next Topic[React Portals](https://www.javatpoint.com/react-portals)