https://www.pluralsight.com/guides/handle-parent-child-data-relationships-in-redux

* [**Introduction**](https://www.pluralsight.com/guides/handle-parent-child-data-relationships-in-redux#module-introduction)
* [Passing Data Between Components](https://www.pluralsight.com/guides/handle-parent-child-data-relationships-in-redux#module-passingdatabetweencomponents)
* [Conclusion](https://www.pluralsight.com/guides/handle-parent-child-data-relationships-in-redux#module-conclusion)
* [Top](https://www.pluralsight.com/guides/handle-parent-child-data-relationships-in-redux" \l "top)

**Introduction**

Redux is an excellent library for state management in client-side JavaScript-based apps. It is framework agnostic, meaning it can be used with any other UI or JavaScript framework. Redux maintains the state of the whole app in a single immutable state object, which cannot be accessed or modified directly.

When working on React-Redux apps, you will come across many use cases where you have to pass and manage data between components. In this guide, you will learn how to handle parent-child data relationships in Redux.

**Passing Data Between Components**

As mentioned earlier, you cannot change the state object or the global store directly. To change the state object, you need to use actions and reducers. An action is simply a JavaScript object that passes information to reducers, and reducers are pure functions that return an updated state based on the current action that was dispatched.

Consider an example where you have counters in the parent and child components. You need to update the counter in the parent component from the child component and vice-versa.

So the initial, or default, state will look as follows.

1const initialState = {

2 parentCounter: 0,

3 childCounter: 0,

4};

js

To update the state you need to create a reducer, as shown below.

1function reducer(state = initialState, action) {

2 switch (action.type) {

3 case "INCREMENT\_PARENT":

4 return { ...state, parentCounter: state.parentCounter + 1 };

5 case "INCREMENT\_CHILD":

6 return { ...state, childCounter: state.childCounter + 1 };

7 default:

8 return state;

9 }

10}

js

In the parent component, dispatch the INCREMENT\_CHILD action so the reducer updates the counter in the child component. To make the dispatch method available as a prop, use the connect() method. Also, pass the mapStateToProps argument to the connect method so the parent's counter value is sent as the prop to the component.

1import React, { Component } from "react";

2import { connect } from "react-redux";

3

4import Child from "./Child";

5

6class Parent extends Component {

7 incrementChildCounter = () => {

8 this.props.dispatch({ type: "INCREMENT\_CHILD" });

9 };

10 render() {

11 return (

12 <div className="parent-component">

13 <div>

14 This is the parent component - [COUNTER: {this.props.counter}]

15 </div>

16 <button onClick={this.incrementChildCounter}>

17 Increment Child Counter

18 </button>

19 <Child />

20 </div>

21 );

22 }

23}

24

25const mapStateToProps = (state) => ({

26 counter: state.parentCounter,

27});

28

29export default connect(mapStateToProps)(Parent);

jsx

Follow the same steps for the child component, except the child component needs to dispatch the INCREMENT\_PARENT action.

1import React, { Component } from "react";

2import { connect } from "react-redux";

3

4class Child extends Component {

5 incrementParentCounter = () => {

6 this.props.dispatch({ type: "INCREMENT\_PARENT" });

7 };

8 render() {

9 return (

10 <div className="child-component">

11 <div>This is the child component - [COUNTER: {this.props.counter}]</div>

12 <button onClick={this.incrementParentCounter}>

13 Increment Parent Counter

14 </button>

15 </div>

16 );

17 }

18}

19

20const mapStateToProps = (state) => ({

21 counter: state.childCounter,

22});

23

24export default connect(mapStateToProps)(Child);

jsx

In the main App component, wrap the Parent component with the Provider component from the react-redux package and pass the store as a prop.

1import { Provider } from "react-redux";

2import { createStore } from "redux";

3

4const store = createStore(reducer);

5

6function App() {

7 return (

8 <div className="App">

9 <Provider store={store}>

10 <Parent />

11 </Provider>

12 </div>

13 );

14}

jsx

**Conclusion**

With Redux, you might complain that there is a lot of boilerplate and overhead for creating actions and reducers and architecting the app. Even Redux suggests that you should start with a basic React app and use Redux only when the app state grows and it becomes difficult to predict state. You can refer to [this](https://redux.js.org/faq/general#when-should-i-use-redux) section of the Redux docs to read more on the topic.

12

[LEARN MORE](https://www.pluralsight.com/product/paths)

We use cookies to make interactions with our websites and services easy and meaningful. For more information about the cookies we use or to find out how you can disable cookies, [click here.](https://www.pluralsight.com/privacy)

Disable cookiesAccept cookies and close this message