ARYATECHNO

React Redux

Topics : React JS
Written on January 03, 2024

React Redux is a state management library for React applications. It is based on the principles of Flux and is designed to help manage the state of a React application in a predictable way. Redux is commonly used in conjunction with React, providing a centralized and predictable state container.

Here's an overview of key concepts and how to use React Redux:

1. Installation:

To use React Redux, you need to install both the `react-redux` and `redux` packages:

```
npm install react-redux redux
# or
yarn add react-redux redux
```

2. Setup Redux Store:

Create a Redux store to manage the application state. This is where your entire application's state is stored. Define reducers to handle different parts of the state.

```
// store.js
import { createStore } from 'redux';
import rootReducer from './reducers'; // Assume you have reducers in a separate file

const store = createStore(rootReducer);

export default store;
```

3. Create Reducers:

Reducers are functions that specify how the state changes in response to actions. Each reducer typically handles a specific part of the state.

```
// reducers.js
const initialState = {
counter: 0,
};

const rootReducer = (state = initialState, action) => {
switch (action.type) {
case 'INCREMENT':
```

```
return { ...state, counter: state.counter + 1 };
case 'DECREMENT':
return { ...state, counter: state.counter - 1 };
default:
return state;
}
};

export default rootReducer;
```

4. Connect React Components:

Use the `connect` function from `react-redux` to connect React components to the Redux store. This function creates container components that are aware of the Redux state.

```
// MyComponent.js
import React from 'react';
import { connect } from 'react-redux';

const MyComponent = ({ counter, increment, decrement }) => {
return (
<div>
<p>Counter: {counter}</p>
<button onClick={increment}>Increment</button>
<button onClick={decrement}>Decrement</button>
</div>
);
};

const mapStateToProps = (state) => ({
counter: state.counter,
});

const mapDispatchToProps = (dispatch) => ({
increment: () => dispatch({ type: 'INCREMENT' }),
decrement: () => dispatch({ type: 'DECREMENT' }),
});

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

5. Provider Component:

Wrap your entire application with the `Provider` component from `react-redux`. This makes the Redux store available to all components in your application.

```
// App.js
import React from 'react';
import { Provider } from 'react-redux';
import store from './store';
import MyComponent from './MyComponent';

const App = () => {
```

```
return (
<Provider store={store}>
<MyComponent />
</Provider>
);
};

export default App;
```

Now, your React components can read data from the Redux store and dispatch actions to update the state. React Redux handles the subscription to the store and efficiently re-renders components when the state changes.