Redux in React

React Redux Library

React Redux is a popular library that provides a predictable state container for JavaScript applications using the React library. The react-redux library provides a centralized store for state management in React applications. It offers several hooks that help in optimizing code and improving application performance. Hooks are an important concept in React Redux because they allow developers to extract logic from components and reuse it across the application.

Installation

To use React Redux with your React app, install it as a dependency:

```
// If you use npm:
npm install react-redux

// Or if you use Yarn:
yarn add react-redux
```

Provider Component

The Provider component is a React component that allows you to provide the Redux store to all components in your application. It is the top-level component that wraps your entire application and makes the store available to all child components.
The Provider component takes a store prop, which is the Redux store that you want to provide to your application.

Here's an example of how to use the Provider component in a ToDo application:

```
import { useState } from "react";
import { Provider } from "react-redux";
import TodoForm from "./components/ToDoForm/ToDoForm";
import TodoList from "./components/ToDoList/ToDoList";
import store from "./redux/store";

import './App.css';

function App() {
  const [todos, setTodos] = useState([]);

  const createTodo = (text) => {
    setTodos([...todos, { id: todos.length + 1, text, completed: false}]);
  };

  const toggleTodo = (index)=>{
    const list = [...todos];
    list[index].completed = !list[index].completed;
    setTodos(list);
  }

  return (
    <div>
      <h1>To Do App</h1>
      <Provider store={store}>
        <TodoForm onCreateTodo={createTodo} />
        <TodoList todos={todos} onToggle={toggleTodo} />
      </Provider>
    </div>
  );
}

export default App;
```

By default, when the Provider element is used, all child components will have access to the entire Redux store. However, it is possible to scope store access to specific

components using the store prop of the Provider element. To do this, you can create a separate Redux store for each component that requires scoped access to the store. Then, when rendering the component, pass in the appropriate store as a prop to the Provider element.

Hooks

React Redux provides a pair of custom React hooks that allow your React components to interact with the Redux store.

useSelector

The **useSelector** hook is used to extract data from the Redux store. It takes a selector function as input and returns the selected data from the store. So, if store gets updated it will not directly impact the components. This also helps in abstraction and encapsulation of store by hiding the important object. For example, Here the useSelector hook is used to retrieve the todos state from the store. The todos state is then mapped over and rendered to the screen as a list of todo items.

```
import { useSelector } from "react-redux";
import "./ToDoList.css";

function ToDoList() {

  const todos=useSelector((state)=> state.todos)

  return (
    <div className="container">
    <ul>
      {todos.map((todo,index) => (
        <li key={todo.id}>
          <span className="content">{todo.text}</span>
          <span className={todo.completed ?
'completed':'pending'}>{todo.completed ? 'Completed': 'Pending'}</span>
          <button className="btn btn-warning">Toggle</button>
          </li>
```

```
      ))}
    </ul>
    </div>
  );
}

export default ToDoList;
```

The useSelector hook is useful for optimizing performance by avoiding unnecessary re-renders. It allows you to select only the data you need from the store, which can help reduce the amount of data that needs to be processed by the component.

useDispatch

The **useDispatch** hook is used to dispatch actions to modify the state. It returns a reference to the dispatch function provided by the store. This hook can be used to dispatch actions from any component in the application, without the need for prop drilling. The useDispatch hook can be used to dispatch actions from any component in the application.

For example, Each todo item includes a button that, when clicked, dispatches a toggleTodo action to the store. The toggleTodo action is imported from the todoActions file, which contains action creators for various todo-related actions.

The dispatch function is used to dispatch the toggleTodo action, passing in the index of the current todo item as a parameter. This will update the completed property of the selected todo item in the store, which will trigger a re-render of the TodoList component.

```
import { useSelector, useDispatch } from "react-redux";
import { toggleTodo } from "../../redux/actions/todoActions";

import "./ToDoList.css";

function ToDoList() {

  const todos=useSelector((state)=> state.todos);
```

```
  const disptach = useDispatch();

  return (
    <div className="container">
    <ul>
      {todos.map((todo,index) => (
        <li key={todo.id}>
          <span className="content">{todo.text}</span>
          <span className={todo.completed ?
'completed':'pending'}>{todo.completed ? 'Completed': 'Pending'}</span>
          <button className="btn btn-warning"
          onClick={()=>{dispatch(toggleTodo(index))}}
          >Toggle</button>
          </li>
      ))}
    </ul>
    </div>
  );
}

export default ToDoList;
```

The useDispatch hook is useful for optimizing code efficiency by providing a simplified way of dispatching actions. It allows you to avoid the need to pass dispatch down as a prop to child components.

Multiple Reducers

In a typical React Redux application, the state is managed by reducers, which are functions responsible for handling different parts of the state. The decision to use multiple reducers or a single reducer depends on the complexity of your application's state. If your application's state is simple and straightforward, a single reducer may suffice. However, as your application grows in complexity, it can become difficult to manage a large state with a single reducer. Using multiple reducers in your React Redux application can provide better organization, improved scalability, and better performance. For example, let's say you have an e-commerce application that

manages user accounts, products, and orders. You can create three separate reducers for each of these parts of the state.

Combining Reducers

Combining reducers is a technique used in React Redux to manage a complex state in a more organized and manageable way. It involves creating multiple reducers that handle different parts of the state and then combining them into a single root reducer using the **combineReducers** function. The combineReducers function takes an object as its argument, where the keys represent the keys of the root state object, and the values represent the individual reducers.

For example in this case, the root state object has two keys, todos and notes, each of which maps to the corresponding reducer.

```
import * as redux from "redux";
import { combineReducers } from "redux";
import { noteReducer } from "./reducers/noteReducer";
import {todoReducer} from "./reducers/todoReducer";

const result = combineReducers({
    todos:todoReducer,
    notes:noteReducer
})

export const store = redux.createStore(result);
```

With this setup, you can now dispatch actions to update the state managed by each reducer separately. For example, to add a todo item, you can dispatch the following action:

```
{
  type: 'ADD_TODO',
  id: 1,
  text: 'Buy milk'
}
```

This action will be handled by the todoReducer, which will update the todos state accordingly. Similarly, to add a note, you can dispatch the following action:

```
{
  type: 'ADD_NOTE',
  id: 1,
  text: 'Call John'
}
```

This action will be handled by the noteReducer, which will update the notes state accordingly.

Summarizing it

Let's summarize what we have learned in this Lecture:

- Learned about React Redux library.
- Learned about Provider Component.
- Learned about useSelector Hook.
- Learned about useDispatch Hook.
- Learned about Multiple Reducers.
- Learned about Combining the Reducers

Some References:

- Redux API Reference: link
- React Redux Quick Start: link