Redux Toolkit

Introduction

Redux is a state management library that helps manage application states in a predictable and consistent manner. It is widely used in modern web development with React and has been an essential tool for building scalable and maintainable applications. However, working with Redux can sometimes be challenging, especially when it comes to setting up and managing reducers, actions, and selectors. To address these challenges, the Redux team has released Redux Toolkit, a package that simplifies the process of setting up and using Redux.

Challenges with Redux

- **Boilerplate Code:** Setting up Redux involves a lot of boilerplate code. You must create separate files for actions, reducers, and store configuration. This can be time-consuming and error-prone.
- **Complex Reducers:** Writing complex reducers can be difficult, especially when dealing with nested data structures or asynchronous actions.
- **Debugging:** Debugging Redux applications can be challenging, especially when dealing with large and complex application states.

Redux Toolkit library

Redux Toolkit provides a streamlined way of working with Redux, eliminating many of the common pain points of building large-scale Redux applications. It is designed to be backward-compatible with existing Redux code, making it easy to adopt and integrate into existing projects. Some of its key features include:

- A "slice" API that simplifies the process of creating Redux reducers
- A "createAsyncThunk" API that simplifies the process of handling asynchronous actions
- A simplified and standardized file structure for Redux code

- Automatic generation of Redux actions and reducers for common use cases
- A collection of other useful utilities and middleware for Redux.

Migrating from Redux to Redux Toolkit

Install Redux Toolkit

Start by installing Redux Toolkit in your application by running the command:

```
npm install @reduxjs/toolkit
yarn add @reduxjs/toolkit
```

Create slices

Replace your existing Redux actions and reducers with "slices," which are predefined Redux logic blocks in Redux Toolkit. You can create slices using the createSlice() function provided by Redux Toolkit.

To define a slice using createSlice, you need to call the function and pass in an object that contains the name, initialState, and reducers properties.

- name: This property is used to define the name of the slice. It is used to generate the action types for the slice automatically and to create action creators with the correct names.
- initialState: This property is used to define the initial state of the slice. It is used to set the initial state of the store when it is first created.
- reducers: This property defines a set of reducer functions that can update the slice's state. It takes an object as an argument where each key-value pair represents a case reducer. The key is the name of the action, and the value is a function that updates the state. When an action is dispatched, the corresponding reducer function will be called, and the slice's state will be updated accordingly. The payload property of the action object can be accessed using action.payload inside the reducer functions.

```
const todoSlice = createSlice({
  name:'todo',
  initialState:initialState,
  reducers:{
      // this is add action
```

```
      add:(state, action)=>{
              state.todos.push({
                  text:action.payload,
                  completed: false
              })
      },
      toggle:(state, action)=>{
          state.todos.map((todo, i)=>{
              if(i==action.payload){
                  todo.completed=!todo.completed;
              }
              return todo;
          })
      }
  }
});
```

Migrating Store

Replace your existing store creation code with the configureStore() function provided by Redux Toolkit. This function simplifies the store creation process by automatically adding common middleware and other settings.

```
export const store = configureStore({
  reducer:{
      todoReducer,
      noteReducer
  }
})
```

Dispatching Actions

The actions object is exported separately from the reducer.

```
export const todoReducer=todoSlice.reducer;
export const actions = todoSlice.actions;
```

You can use this object to access your actions by importing them and passing them as arguments to the dispatch function.

```
<button className="btn btn-warning"
onClick={()=>{disptach(actions.toggle(index))}}>Toggle</button>
```

Setting Up Selectors

You can define a new selector function called todoSelector, which extracts the todoReducer slice of the state.

```
// selector
export const todoSelector = (state)=>state.todoReducer.todos;
```

Once you have defined your selectors, you can use the useSelector hook from the react-redux library to access the selected data in your components.

Create React App with Redux Template

Redux Toolkit provides a template for creating a React app with Redux preconfigured, which you can use to start quickly with building a new Redux-powered React application. Run the following command to create a new React app with the Redux Toolkit template:

```
npx create-react-app my-app --template redux
```

This will create a new React app with the Redux Toolkit template and install all the necessary dependencies.

Extra Reducers

Extra Reducer allows you to execute an action which is the action of some other reducer. It allows you to share an action, invoke an action or dispatch an action that belongs to some other reducer. Using extra reducers like this can help simplify your code and make it easier to share actions between different reducers in your Redux store.

For Example, if we want to dispatch a notification when a todo is added. Whenever an action with the type todo/add is dispatched, the function defined in the extraReducers property is executed.

```
const notificationSlice = createSlice({
  name:'notification',
  initialState,
  reducers:{
      reset: (state, action)=>{
          state.message="";
      }
  },
  extraReducers:{
      "todo/add":(state, action)=>{
          console.log("todo/add in notificationReducer");
        state.message="Todo is created";
      }
  }
});
```

The reset action allows you to clear the notification message when it is no longer needed, such as after a user has read the message. "reset" action that can be dispatched to reset the message property to an empty string.

Creating Extra Reducers using Builder and addCase

Creating extra reducers using the Builder and Case API in Redux Toolkit allows you to handle actions dispatched from other slices of your Redux store without hardcoding their names into your reducer.

```
const notificationSlice = createSlice({
  name:'notification',
  initialState,
  reducers:{
      reset: (state, action)=>{
          state.message="";
      }
  },
  extraReducers:(builder)=>{
      builder.addCase(actions.add, (state, action)=>{
          state.message="Todo is created";
      })
  }
});
```

The builder argument is an instance of the ActionReducerMapBuilder class, which provides methods for adding new case reducers to your slice. The addCase method takes two arguments: the action creator function and a callback function that handles the action. In this case, we pass the actions.add action creator function, which is defined elsewhere in our application, and a callback function that sets the message property of our state to "Todo is created". This approach allows us to handle actions from other parts of our application without tightly coupling our reducers to the actions that they handle.

Creating Extra Reducers using Maps

The extraReducers field uses a map object to define an action handler for the add action. The key of the map object is the action type, and the value is the function that will handle the action. In this case, when the add action is dispatched, the message is set to "Todo is created".

```
const notificationSlice = createSlice({
  name:'notification',
  initialState,
  reducers:{
      reset: (state, action)=>{
          state.message="";
      }
  },
  extraReducers:{
      // map objects: [key]: value
      [actions.add]: (state, action)=>{
          state.message="Todo is created";
      }
  }
});
```

Summarizing it

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

- Learned about challenges with Redux.
- Learned what Redux Toolkit is.

- Learned how to migrate from redux to the redux toolkit.
- Learned how to create react app with the redux template.
- Learned about Extra Reducers.

Some References:

- Getting started with Redux Toolkit: link
- Installation: link