How create a React + Redux App

井民全, Jing, mqjing@gmail.com

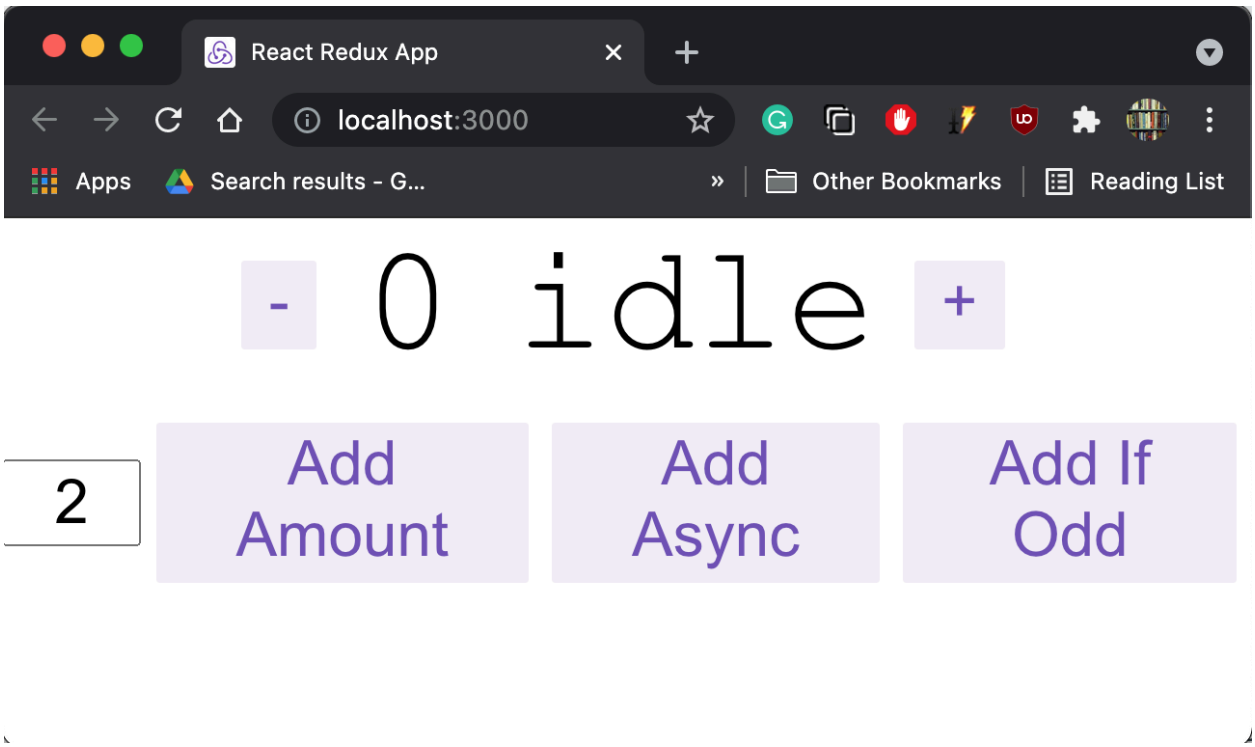


Fig. The moddify result by adding state string.

GitHub: Source

Why Redux?

Redux is a library for management the App state. React has in-build hooks such as useState, ...etc. to do the same thing. However, the Redux way can provide a central way to control the app state. Check the following statement from official doc.

> The whole global state of your app is stored in an object tree inside a single store. The only way to change the state tree is to create an action, an object describing what happened, and dispatch it to the store[1].

This document is just my note for the React + Redux template that includes

1. How to create a React + Redux template app?
2. How to create a store for the app? (in **store**.ts)
3. How to define the component state update logic (in **xxxSlices**.ts)
4. How a component extracts the state from the store (using **selector** hook)
5. How a component sends activitie to the correspond reducer (using **dispatch**)
6. Have an example mixing React in-build state function (useState) and the Redux useSelect? (in **Counter**.tsx)

However, you should read the tutorial from the Redux site. Ok, show the instructions and the code.

By Jing.

Table of contents

1. Create React + Redux app template **3**

2. Code **3**

2.1. Step 1: Define the App Store 3

2.2. Step 2: Define the component Redux logic 4

2.3. Step 3: Define the React Component 6

2.4. App Main 8

3. Install & Run **9**

4. Clean **9**

5. References **9**

1. Create React + Redux app template

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

2. Code

2.1. Step 1: Define the App Store

File: src/app/store.ts

```
import { configureStore, ThunkAction, Action } from '@reduxjs/toolkit';
import counterReducer from '../features/counter/counterSlice';

// (1) Create the store for app states and the reducers that handle the update logic
export const store = configureStore({
  // (2) The root reducer
  reducer: {
    counter: counterReducer,
    // (key) counter: we want to have a "state.counter" section of the Redux state object
    // (reducer) counterReducer: we want the counterReducer function to be in charge of deciding
if and how to update the state.counter section whenever an action is dispached

    // (3) Other features
    // xxx: xxxReducer,
  },
});

// (4) Define the AppDispatch for action routing and trigger the component state update
export type AppDispatch = typeof store.dispatch;

// (5) Define the RootState that unified state of components
//     Depends
//     Code: the state selector in components
//     File: Component slice source(xxxSlice.ts)
export type RootState = ReturnType<typeof store.getState>;

// (6) Define helper AppThunk type for creating thunk object to handle the
//     "async" activity logic and the dispatch
//     Depends
```

```
//    Code: the thunk object that Async update state logic (version: manual)
//    File: Component slice source(xxxSlice.ts)
// Ref
//    1. https://bloggie.io/@_ChristineOo/understanding-typings-of-redux-thunk-action
//    2. https://github.com/reduxjs/redux-thunk#why-do-i-need-this
export type AppThunk<ReturnType = void> = ThunkAction<
  ReturnType,
  RootState,
  unknown,
  Action<string>
>;
```

2.2. Step 2: Define the component Redux logic

File: src/features/counter/counterSlice.ts

```
import { createAsyncThunk, createSlice, PayloadAction } from '@reduxjs/toolkit';
import { RootState, AppThunk } from '../../app/store';
import { fetchCount } from './counterAPI';

// (1) Define the state type where your the component included
export interface CounterState {
  value: number;                    // (1) value: number
  status: 'idle' | 'loading' | 'failed';    // (2) status: 'idle' | 'loading' | 'failed'
}

// (2) The state initial value
const initialState: CounterState = {
  value: 0,
  status: 'idle',
};

// (3a) Async Example: Async update state logic (version: createAsyncThunk)
//    Create a thunk to perform async logic of action types
//    1. Input:
//       (a) Action type prefix: "counter/fetchCount"
//       (b) callback function: async logic
//    2. Export:
//       (a) Export a promise lifecycle action types
// Reference
// 1. https://redux-toolkit.js.org/api/createAsyncThunk
export const incrementAsync = createAsyncThunk(
  'counter/fetchCount',
```

```
  async (amount: number) => {
    const response = await fetchCount(amount); // Wait for 0.5 sec to resolve the response.
    // The value we return becomes the `fulfilled` action payload
    return response.data; // <--- action.payload
  }
);

export const counterSlice = createSlice({
  name: 'counter',
  initialState,

  reducers: {
    // (5) Examples for update state based on associated actions
    increment: (state) => {
      state.value += 1;
    },
    decrement: (state) => {
      state.value -= 1;
    },

    // (6) An example for update state using action payload
    incrementByAmount: (state, action: PayloadAction<number>) => {
      state.value += action.payload;
    },
  },

  // (7) Add extra reducers:
  //     Purpose: Add reducers for step (3a) that the actions generated by createAsyncThunk
  extraReducers: (builder) => {
    builder
      .addCase(incrementAsync.pending, (state) => {
        state.status = 'loading';
      })
      .addCase(incrementAsync.fulfilled, (state, action) => {
        state.status = 'idle';
        state.value += action.payload;
      });
  },
});

export const { increment, decrement, incrementByAmount } = counterSlice.actions;

// (8) Define the selector functions
//     Purpose: Extract data from the Redux store state
export const selectCount = (state: RootState) => state.counter.value;
export const selectStatus = (state: RootState) => state.counter.status;

// (9) Async Example: Async update state logic (version: manual)
```

```
//    Purpose: Create a thunk to perform async logic of action types
export const incrementIfOdd = (amount: number): AppThunk => (
  dispatch,
  getState
) => {
  const currentValue = selectCount(getState());
  if (currentValue % 2 === 1) {
    dispatch(incrementByAmount(amount));    // dispatch the incrementByAmount
  }
};

// (10) Export the component reducers
export default counterSlice.reducer;
```

2.3. Step 3: Define the React Component

File: /src/features/counter/Counter.tsx

```
import React, { useState } from 'react';

// (1) Import the AppSelector and AppDispatch hooks for access the store
import { useAppSelector, useAppDispatch } from '../../app/hooks';

// (2) Import the component actions and the state selector
//    [actions]:  for trigger component state updates and
//    [selector]: for extacting the componet state from store
import {
  decrement,
  increment,
  incrementByAmount,
  incrementAsync,
  incrementIfOdd,
  selectCount,
  selectStatus
} from './counterSlice';

import styles from './Counter.module.css';

// (2) The Component
export function Counter() {
  // (3) [access store] Extract the count state
  //    Render
  //    Any time an action has been dispatched and the Redux store has been updated,
```

```
//    useSelector will re-run our selector function. If the selector returns a different value
//    than last time, useSelector will make sure our component re-renders with the new value.
//
//    Ref
//    https://redux.js.org/tutorials/essentials/part-2-app-structure#reading-data-with-useselector
const count:number = useAppSelector(selectCount);

// (4) [access store] Extract the status state
const status:string = useAppSelector(selectStatus);

// (5) [access store] Get the app dispatch function
const dispatch = useAppDispatch();

// (6) The React build-in hooks for component state
const [incrementAmount, setIncrementAmount] = useState('2');
const incrementValue = Number(incrementAmount) || 0;

return (
  <div>
    <div className={styles.row}>
      <button
        className={styles.button}
        aria-label="Decrement value"
        onClick={() => dispatch(decrement())}
      >
        -
      </button>

      <span className={styles.value}>{count}</span>
      <span className={styles.value}>{status}</span>

      <button
        className={styles.button}
        aria-label="Increment value"
        onClick={() => dispatch(increment())}
      >
        +
      </button>
    </div>
    <div className={styles.row}>
      <input
        className={styles.textbox}
        aria-label="Set increment amount"
        value={incrementAmount}
        onChange={(e) => setIncrementAmount(e.target.value)}
      />
      <button
        className={styles.button}
```

```
        onClick={() => dispatch(incrementByAmount(incrementValue))}
      >
        Add Amount
      </button>
      <button
        className={styles.asyncButton}
        onClick={() => dispatch(incrementAsync(incrementValue))}
      >
        Add Async
      </button>
      <button
        className={styles.button}
        onClick={() => dispatch(incrementIfOdd(incrementValue))}
      >
        Add If Odd
      </button>
    </div>
  </div>
 );
}
```

2.4. App Main

File: src/App.tsx

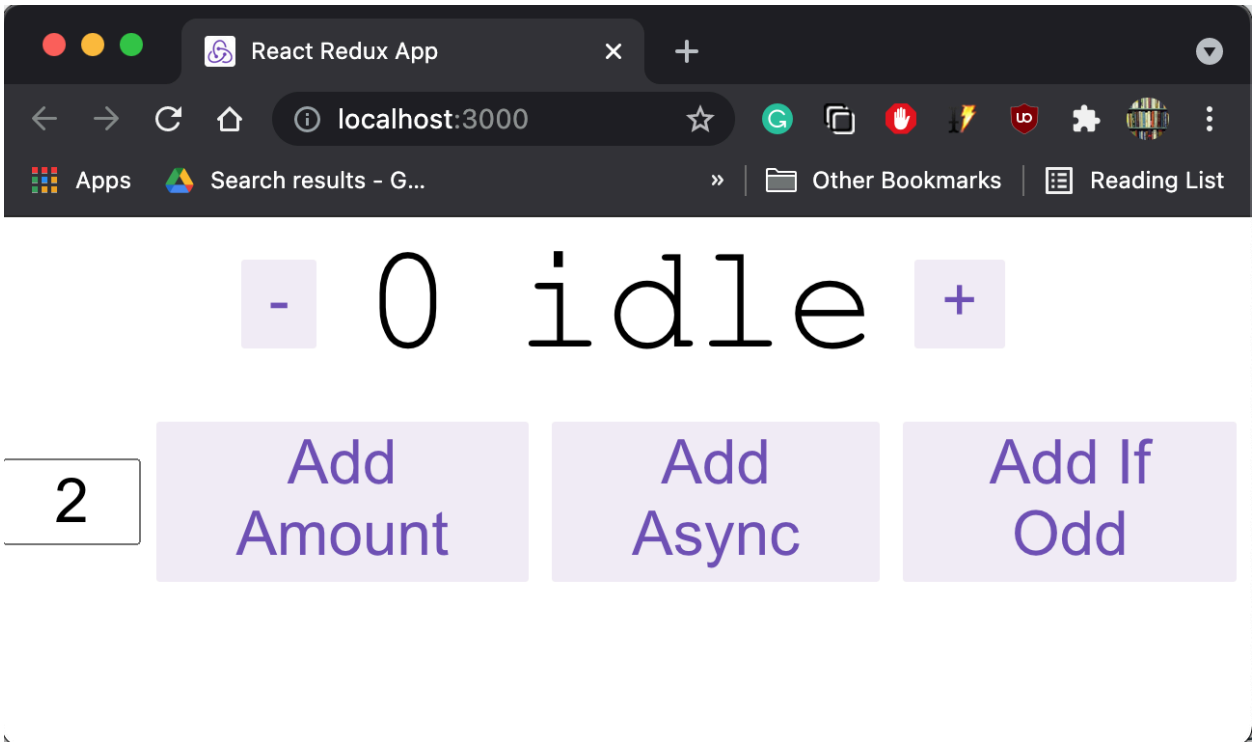
```
import React from 'react';
import logo from './logo.svg';
import { Counter } from './features/counter/Counter';
import './App.css';

function App() {
 return (
  <div>
   <Counter />
  </div>
 );
}

export default App;
```

3. Install & Run

```
yarn install
yarn start
```



4. Clean

File: ./package.json

```
...

"clean": "(rm -fr node_modules; rm -fr build; find . -name \"*.js\" -type f|xargs rm -f; find . -name \"*.map\" -type f|xargs rm -f)"

..
```

yarn clean

5. References

1. Getting Start with Redux, https://redux.js.org/introduction/getting-started
2. Redux Essentials, Part 2: Redux App Structure, https://redux.js.org/tutorials/essentials/part-2-app-structure
3. Redux Toolkit TypeScript Quick Start, https://redux-toolkit.js.org/tutorials/typescript
4. createAsyncThunk, https://redux-toolkit.js.org/api/createAsyncThunk