# Currying

Currying is the transformation of a function with multiple arguments into a sequence of single-argument functions. That means converting a function like this f(a, b, c, ...) into a function like this f(a)(b)(c)... . Currying is helpful when you have to frequently call a function with a fixed argument.

Normal Function :

function isGreaterThan(a, b) {

return b > a;

}

isGreaterThan(2, 5)

Currying function :

function isGreaterThan(a) {

return function(b) {

return b > a;

}

}

isGreaterThan(2)(5);

const isGreaterThan = a => b => b > a;

# object Update

1. In JavaScript we can copy object and update the object without mutable the value of principal object with object.assign() method.

Syntax: Object.assign(target, sources);

Code:

const obj = { a: 1 };

const copy = Object.assign({}, obj);

console.log(copy);

1. but now we use spread operator for update the object.

# redux with JavaScript

1. **component:** First we create a html screen where work with state.

Code:

<html lang="en">

<head>

</head>

<body>

    <div class="container">

        <p>Home</p>

        <h1 id="counter"></h1>

            <button id="inc"> increment </button>

            <button id="dec"> decrement </button>

</div>

    <script src="react.js"></script>

</body>

</html>

1. **initial the value:** For redux use we first create initial state. This state we pass to redux for use state change and UI interaction.

const initialState = {

    value: 0

}

1. **create action/reducer function:**

function counterReducer(state = initialState, action) {

  if (action.type == "increment") {

    return {

      ...state,

      value: state.value + 1,

    };

  } else if (action.type == "decrement") {

    return {

      ...state,

      value: state.value - 1,

    };

  } else {

    return state;

  }

}

1. **store create:** we use createStore() method for create store. Here we save state and action.

const store = Redux.createStore(counterReducer);

1. **dispatch method:**

inc.addEventListener("click", () => {

  store.dispatch({

    type: "increment",

  });

});

dec.addEventListener("click", () => {

store.dispatch({

    type: "decrement",

});

});

1. **Subscribe:**

store.subscribe(rendar);

1. **update UI :**

const rendar = () => {

  const state = store.getState();

  console.log(state);

  counter.innerText = state.value.toString();

};

# React-Redux

**Normal React-Redux:**

1. **Install:** we should install “redux” then “react- redux”

npm i redux

npm I react-redux

1. **Value initiate:**

const initialState = {

    value: 0

}

1. **Action create:**

function counterReducer (state = initialState, action) {

switch(action.type){

case “increment”:

    return {

      ...state,

      value: state.value + 1,

    };

case “decrement”:

    return {

      ...state,

      value: state.value - 1,

    };

  }

default:

    return state;

}

}

1. **Store create:** in react-redux we import createStore from redux. Then pass reducer functions.

Import {createStore} from “redux”

const store = createStore(counterReducer);

1. **Provider:** now we provide store in component by wrapping the whole app. Then pass the store.

Import {Provider} from “redux”;

Import store from “. /app”;

<Provider store={store}>

<Counter />

</Provider>

1. **Subscribe / Connect:** we subscribe with redux store with connect() function. In connect() we pass out component which is interaction with store.

**Syntax:** connect (argument1, argument2) (viewComponent);

**Argument 1:** we pass function which takes state value as argument.

const argument1 = (state) => {

  return {

    count: state.value,

  };

};

**Argument 2**: we pass function with action dispatch value:

const mapDispatchToProps = (dispatch) => {

return {

inc: () =>

      dispatch({

        type: "increment",

        // payload: value,

 }),

 dec: () =>

      dispatch({

        type: "decrement",

        // payload: value,

  }),

};

};

**ViewConponent:**

import {connect} from "react-redux"

function Counter() {

    return (

    <div>

        <h1> {count} </h1>

        <div>

            <button onClick={inc} > increment </button>

            <button onClick={dec} > decremnt </button>

        </div>

    </div>

  )

}

export default connect (mapStateToProps, mapDispatchToProps)(Counter);

**full code:**

import { connect } from "react-redux"

function Counter() {

    return (

    <div>

        <h1> {count} </h1>

        <div>

            <button onClick={inc} > increment </button>

            <button onClick={dec} > decremnt </button>

        </div>

    </div>

  )

}

const mapStateToProps = (state) => {

  return {

    count: state.value,

  };

};

const mapDispatchToProps = (dispatch) => {

  return {

inc: () =>

      dispatch({

        type: "increment",

        // payload: value,

 }),

 dec: () =>

      dispatch({

        type: "decrement",

        // payload: value,

  }),

  };

};

export default connect (mapStateToProps, mapDispatchToProps)(Counter);

# react-Redux hook:

1. **Install:** we should install “redux” then “react- redux”

npm i redux

npm I react-redux

1. **Value initiate:**

const initialState = {

    value: 0

}

1. **Action create:**

function counterReducer (state = initialState, action) {

switch(action.type){

case “increment”:

    return {

      ...state,

      value: state.value + 1,

    };

case “decrement”:

    return {

      ...state,

      value: state.value - 1,

    };

  }

default:

    return state;

}

}

1. **Store create:** in react-redux we import createStore from redux. Then pass reducer functions.

Import {createStore} from “redux”

const store = createStore(counterReducer);

1. **Provider:** now we provide store in component by wrapping the whole app. Then pass the store.

Import {Provider} from “redux”;

Import store from “. /app”;

<Provider store={store}>

<Counter />

</Provider>

1. **useSelector():** we use redux hook useSelector for take state value in component.

code:

import { useSelector } from "react-redux";

function Counter () {

const count = useSelector((state)=> state.value)

    return (

    <div>

        <h1> {count} </h1>

        <div>

            <button onClick={inc} > increment </button>

            <button onClick={dec} > decrement </button>

        </div>

    </div>

  )

}

1. **useDispatch ():** we pass function name for action occurred in reducer.

**code:**

function HookCounter() {

  const count = useSelector((state)=> state.value);

  const  setCount = useDispatch();

  const inc = () => {

    setCount({

        type: "increment",

    })

  }

  const dec = () => {

    setCount ({

      type: "decrement",

    })

  }

  return (

    <div>

      <h1> {count} </h1>

      <div>

        <button onClick={inc}> increment </button>

        <button onClick={dec}> decrement </button>

      </div>

    </div>

  );

}

export default HookCounter;

# redux combine reducer

Redux always use single reducer in mechanism. but we want to use multiple reducer in react, then we need to combine multiple reducer in single reducer. We do it with combineReducers() method.

1. **Reducer/action -1 :**

const initialState = {

  value: 20,

};

export const actionReducer1 = (state = initialState, action) => {

switch (action.type) {

  case "reducer1/increment":

      return {

        ...state,

        value: state.value + 1,

   };

  case "reducer1/decrement":

      return {

        ...state,

        value: state.value - 1,

   };

   default:

      return state;

}

};

1. **Reducer/action -2:**

const initialState = {

  value: 0,

};

export const actionReducer2 = (state = initialState, action) => {

switch (action.type) {

 case "reducer2/increment":

      return {

        ...state,

        value: state.value + 5,

   };

  case "reducer2/decrement":

      return {

        ...state,

        value: state.value - 5,

   };

   default:

      return state;

  }

};

1. **CombineReducer():** Combine the two reducuer by combine method:

import { combineReducers } from "redux";

import { legacy\_createStore as createStore } from "redux";

import { actionReducer1 } from "./reducer1/actionsReducer1";

import { actionReducer2 } from "./reducer2/actionsReducer2";

const rootReducer = combineReducers({

  reduer1: actionReducer1,

  reduer2: actionReducer2,

});

const combinestore = createStore(rootReducer);

export default combinestore;

# redux toolkit

1. **install redux toolkit:** we install redux toolkit by command in terminal.

npm install @reduxjs/toolkit

1. **initial value:** we create an initial value ;

**code:**

const initial = {

count: 0,

}

1. **createSlice:** import createSlice() function and reducer function. createSlice() accepts an object. In object we pass - initialState, name, reducers. this value are fixed in name.

**code:**

import { createSlice } from '@reduxjs/toolkit'  
const counterSlice = createSlice({  
 name: 'counter',  
 initialState: 0,  
 reducers: {  
 increment: (state) =>{

state.value++;

},

decrement: (state) => {

state.value--;

}  
 },  
})

**Syntax:**  createSlice ({name: string, initialState: any, reducers: Object})

* 1. **initialState:** The initial state value for this slice of state.
  2. **Name:** A string name for this slice of state. Generated action type constants will use this as a prefix.
  3. **Reducers:** An object containing Redux "case reducer" functions.

Code:

import { createSlice } from '@reduxjs/toolkit'  
const counterSlice = createSlice({  
 name: 'counter',  
 initialState: 0,  
 reducers: {  
 increment: (state,action) =>{ state.value = state.value + 1},  
 },  
})

* 1. **Return Value**[**​**](https://redux-toolkit.js.org/api/createslice#return-value)**:** createSlice will return an object that looks like:

{  
 name : string,  
 reducer : ReducerFunction,  
 actions : Record<string, ActionCreator>,  
 caseReducers: Record<string, CaseReducer>.  
 getInitialState: () => State  
}

1. **reducer:** We export reducer method of createSlice() methods.

import { createSlice } from '@reduxjs/toolkit'  
const counterSlice = createSlice({  
 name: 'counter',  
 initialState: 0,  
 reducers: {  
 increment: (state) =>{

state.value++;

},

decrement: (state) => {

state.value--;

}  
 },  
})

export default counterSlice.reducer;

1. **Redux Store:**