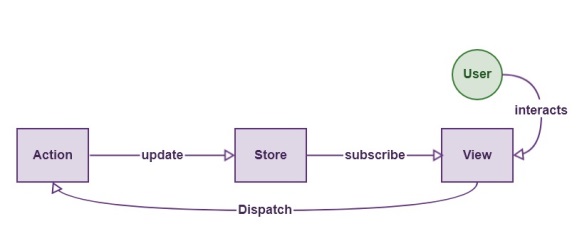
**Redux**

**Redux** is the predictable state container for JavaScript applications. Redux is also following the **Unidirectional flow,**but it is entirely different from **Flux. Flux has multiple stores.**

**Redux has a Single Store.**



**Redux can not have multiple stores.** The store is divided into various state objects. So all we need is to maintain the single store, or we can say the only source of truth.

## ****Three Principles Of Redux****

#### Single source of truth

#### The state is read-only

#### Changes are made with pure functions.

It is the state of our whole application is stored in an object within a single store.  There is an only way to change the state is to emit an action, an object describing what happened.To specify how actions transform the state, you write pure reducers.

## ****Actions****

**Actions** are payloads of information that send data from your application to your store. You send them to the store using.store.dispatch()

Actions are plain JavaScript objects. Actions must have a type property that indicates the type of action being performed. Types should typically be defined as string constants.

import { ADD\_TODO, REMOVE\_TODO } from '../actionTypes'

**Action Creators**

**Action creators** are exactly the functions that create actions.

function addTodo(text) {

return {

type: ADD\_TODO,

text

}

}

**Reducers**

Actions describe the fact that something happened but don’t specify how the application’s state changes in response. That is the job of reducers.

**Handling Actions**

(previousState, action) => newState

This is called a reducer because it is the type of function you would pass to Array.prototype.reduce(reducer, ?initialValue). It is essential that the reducer stay pure.

Following are the things you should **never** do inside a reducer:

* Mutate reducer’s arguments;
* Perform side effects like database calls, API calls, and routing transitions;
* Call non-pure functions, e.g., Date.now() or Math.random()

**Store**

A store is an object that brings them together. A store has the following responsibilities:

* Holds application state;
* Allows access to state via getState();
* Allows state to be updated via dispatch(action);
* Registers listeners via subscribe(listener);
* It handles unregistering of listeners via the function returned by subscribe(listener)

# Middleware

An **example** of a **redux middleware** is **redux**-thunk which allows you to write action creators that return a function instead of an action. ... Therefore we can say our**middleware** receives a store, then returns a function that receives a next function and returns another function that receives an action.

Initial State:

reducers/rotateReducer.js

const initialValue={

value: 10,

loading:1

};

Reducers:

const reducer = (state=initialValue, action) => {

const newState = {... state};

switch (action.type) {

case "UP":

console.log('up ',action.value);

newState.value +=action.value;

newState.loading=false;

break;

case "DOWN":

console.log('down ',action.value);

newState.value -=action.value;

break;

case "LOADING":

newState.loading=true;

}

initialValue.value=newState.value;

return newState;

};

export default reducer;

index.js

///

import reducer from './reducers/rotateReducer';

import { Provider } from "react-redux";

//

import { createStore,applyMiddleware } from "redux";

import thunk from 'redux-thunk';

const logAction = store=>{

return next=>{

return action =>{

const result = next(action);

console.log(`caught in the middleware ${JSON.stringify(result)}`);

}

}

};

const store = createStore(reducer,applyMiddleware(logAction));

//const store = createStore(reducer); //,applyMiddleware(thunk)

ReactDOM.render(

<Provider store={store}>

<App />

</Provider>, document.getElementById('root'));

Component : App.js

import { connect } from "react-redux";

import { aAction } from "./actions/aAction";

import { bAction } from "./actions/bAction";

<h2> Redux Example </h2>

<p>

</p>

Out is : {this.props.data}

<p>

<button onClick={this.props.up} > Up </button>

<button onClick={this.props.down} > Down </button>

{this.props.loading && <img src={logo} className="App-logo"/>}

</p>

const mapStateToProps = state => {

return{

data: state.value,

loading: state.loading

};

};

const mapDispatchToProps = dispatch => {

return{

up: () => dispatch({type:"UP",value:3}),

down: () => dispatch({type:"DOWN",value:3})

//up: () => dispatch(aAction(3)),

//down: () => dispatch(bAction(3))

};

};

//export default connect()(App);

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