JavaScript - State - Redux

# General

Redux is a predictable state container for JavaScript single page applications. Commonly used with React to handle state without the use of unnessecary chains of props.

Redux state management comes with the advantage of being able to scale into large applications, and the ability to trace every mutation back to the action which caused it. For example, a users journey could be reproduced by a list of their redux actions.

Common reasons to use redux are:

* You have reasonable amounts of data changing over time
* You need a single source of truth for your state
* You find that keeping all your state in a top-level component is no longer sufficient
* Persist state to local storage and boot it up from there
* Prefill state in SSR and send it over html

## Concept

The core redux concept is, It saves a current value, runs a single function to update that value when needed, and notifies any subscribers that something has changed.

The whole global state of the application is stored in an object tree, inside a single store.

The state is never mutated directly, instead to change the state tree, an **action** needs to be created, which contains an object describing what has happened. This then gets dispatched to the store. How the state is updated in response to the action is defined in a reducer function, which calculates the new state based on the old state and action.

An example action might be:

{ type: 'ADD\_TODO', text: 'new todo' }

The reducer function takes state and action, then returns new state. For example:

function counterReducer(state = { value: 0 }, action) {

switch (action.type) {

case 'counter/incremented':

return { value: state.value + 1 }

case 'counter/decremented':

return { value: state.value - 1 }

default:

return state

}

}

This can then be created into a redux store, using the redux api:

let store = createStore(counterReducer)

The store can then be used in two ways:

The store can be subscribed to, allowing for updates in the UI upon state change. Commonly a view binding library is used for this, such as react/redux. For example:

store.subscribe(() => console.log(store.getState()))

The store can have its state mutated, using the dispatch method:

store.dispatch({type: 'counter/incremented'})

## ReduxToolKit

Redux toolkit simplifies the creation of state trees and reducers. For example:

import { createSlice, configureStore } from '@reduxjs/toolkit'

const counterSlice = createSlice({

name: 'counter',

initialState: {

value: 0

},

reducers: {

incremented: state => {

state.value += 1

},

decremented: state => {

state.value -= 1

}

}

})

export const { incremented, decremented } = counterSlice.actions

const store = configureStore({

reducer: counterSlice.reducer

})

store.subscribe(() => console.log(store.getState()))

store.dispatch(incremented())

<https://redux.js.org/understanding/thinking-in-redux/three-principles>

<https://redux.js.org/tutorials/index>

<https://css-tricks.com/learning-react-redux/>

<https://blog.isquaredsoftware.com/2017/05/idiomatic-redux-tao-of-redux-part-1/>

<https://blog.isquaredsoftware.com/2017/05/idiomatic-redux-tao-of-redux-part-2/>

<https://redux.js.org/introduction/learning-resources>

<https://redux.js.org/style-guide/style-guide>