Redux

Introducción de Redux

Redux intenta hacer predecibles las mutaciones del estado imponiendo ciertas restricciones en cómo y cuándo pueden realizarse las actualizaciones. Estas restricciones se reflejan en los tres principios de Redux.

1. El estado de toda tu aplicación está almacenado en un árbol guardado en un único store.
2. La única forma de modificar el estado es emitiendo una acción, un objeto describiendo que ocurrió.
3. Para especificar cómo se transforma el árbol de estado por las acciones, se utilizan funciones puras (reducers).

Store

El store es un único árbol de objetos que almacena el estado de la aplicación. La única forma de modificar el estado es emitiendo una acción, un objeto describiendo que ocurrió. Para crear un store:

```
createStore(reducer, [preloadedState], [enhancer])
```

- Reducer: Una función que devuelve el siguiente estado
- PreloadedState: (opcional) estado inicial en el store
- Enhancer: (opcional) es una función para mejorar el store con persistencia

Store: createStore

```
import { createStore } from 'redux'

function todos(state = [], action) {
 switch (action.type) {
   case 'ADD_TODO':
     return state.concat([action.text])
   default:
     return state
 }
}
const store = createStore(todos, ['Use Redux'])
store.dispatch({
 type: 'ADD_TODO',
 text: 'Read the docs'
})
console.log(store.getState())
```

Estados

En Redux, todo el estado de la aplicación se almacena en un único objeto. En el árbol de estado se almacenan tanto datos, como el estado de la UI. Conviene tener separados este tipo de datos dentro del árbol del estado. Es recomendable tener un árbol de estado lo más plano posible, para ello, usamos IDs asociadas a cada campo del estado.

Tenemos los siguientes métodos para trabajar con estados:

- `getState()`
- `dispatch(action)`
- `subscribe(listener)`
- `replaceReducer(nextReducer)`

Estados: dispatch

Realiza una acción sobre el estado para poder cambiarlo, tiene como único argumento una función que devuelve el estado que se modifica.

```
import { createStore } from 'redux'
const store = createStore(todos, ['Use Redux'])

function addTodo(text) {
 return {
   type: 'ADD_TODO',
   text
 }
}

store.dispatch(addTodo('Read the docs'))
store.dispatch(addTodo('Read about the middleware'))
```

Estados: suscribe(listener)

Añade un listener que se llama cuando cambia el estado del store, tiene como argumento la función que se ejecuta al recibir el listener.

Hay que usar con cuidado esta función puesto que va a escuchar los cambios en el estado y llamar a la función siempre que ocurra, por tanto la función debe tener restricciones.

Conviene no usar en muchos componentes este proceso.

Estado: suscribe(listener)

```
function select(state) {
   return state.some.deep.property
 }
  let currentValue
 function handleChange() {
   let previousValue = currentValue
   currentValue = select(store.getState())
    if (previousValue !== currentValue) {
     console.log(
       'Some deep nested property changed from',
       previousValue,
       'to',
       currentValue
     )
   }
 }
  const unsubscribe = store.subscribe(handleChange)
 unsubscribe()
```

Acciones

Las acciones se usan a través de reducers, que son funciones funciones de tipo stateless que cambian el estado de la aplicación en función de la acción que entra como parámetro.

Dentro de un reducer no se debe hacer:

- Modificar los argumentos de la función.
- Realizar tareas como llamadas a un API o de rutado.
- Llamar a una función que varíe con el tiempo Date.now() o Math.random().

react-redux

React Redux es el enlace oficial de React para Redux. Permite a componentes React leer datos del store de Redux y enviar acciones al store para actualizar los datos.

Para instalar react-redux en nuestro proyecto:

```
npm install redux
npm install react-redux
npm install react-dom
```

Provider

Permite alcanzar el store de redux a cualquier componente de react

```
import React from 'react'
import ReactDOM from 'react-dom'

import { Provider } from 'react-redux'
import store from './store'

import App from './App'

const rootElement = document.getElementById('root')
ReactDOM.render(
 <Provider store={store}>
   <App />
 </Provider>, rootElement )
```

Connect

Permite conectar con el store desde cualquier componente

```
import { connect } from 'react-redux'
import { increment, decrement, reset } from './actionCreators'

// const Counter = ...
const mapStateToProps = (state /*, ownProps*/) => {
 return {
   counter: state.counter
 }
}
const mapDispatchToProps = { increment, decrement, reset }

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

Connect: obtener datos con mapStateToProps

La función mapStateToProps debe devolver un objeto que contenga los datos que necesita el componente

```
function mapStateToProps(state) {
   return {
     a: 42,
     todos: state.todos,
     filter: state.visibilityFilter
   }
 }
```

mapStateToProps

- Tienen la responsabilidad de "remodelar" los datos de la tienda según sea necesario para ese componente
- Tienen que ser rápidos procesando porque van a ser llamados en el ciclo de vida del componente
- Deben ser “stateless” y síncronos

mapStateToProps: parámetro opcional ownProps

Enlaza las propiedades del componente con el store

```
function mapStateToProps(state, ownProps) {
   const { visibilityFilter } = state
   const { id } = ownProps
   const todo = getTodoById(state, id)
    // component receives additionally:
   return { todo, visibilityFilter }
 }
  // Later, in your application, a parent component renders:
 ;<ConnectedTodo id={123} />
 // and your component receives props.id, props.todo, and props.visibilityFilter
```

Connect: acciones con mapDispatchToProps

Si llamamos a connect sin el segundo argumento, utilizamos el método dispatch por defecto de Redux, podemos redefinir la función declarándola en connect()

```
connect()(MyComponent)
// equivalente
connect(null,null)(MyComponent)
// otra forma
connect(mapStateToProps)(MyComponent)
```

```
function Counter({ count, dispatch }) {
   return (
     <div>
       <button onClick={() => dispatch({ type: 'DECREMENT' })}>-</button>
       <span>{count}</span>
     </div>
   )
 }
```