Redux & React-redux

================

-- Redux ---

#01

===

What is redux ?

• Redux is a predictable state container for javascript apps.

#Redux

•It is for javascript apps
•It is a state container
•it is predictable

#Redux is for javascript apps

•Redux is not tied to React
•It can be used with React,Angular,Vue or even in vanilla javascript
•Redux is a library for javascript application

#Redux is a state container

•Redux stores the state of your application
•Consider a react app -state of a component
• State of an app is the state represented by all individual components of the app

#Redux is predictable

•Predictable in what way ?
•Redux is state container
•The state of the app can change
Ex :todo list app -item (pending) ->item (completed)
•In redux, all state transitions are explicit and it is possible to keep track of them.
• The changes to your application's state become predictable.

# Redux is a predictable state container for javascript apps.

• Manage the state of your application in a predictable way,redux will help you.

#React-redux ::

• Why we need another tool to help manage the state ,
• Components in React have their own state.

Do we really have problem towards state management?

• React context -prevents props drilling
• useContext + useReducer ?
• Redux 1.0 -August 2015 (The above two not yet released in aug 2015)

React-redux ::

•React-Redux is the offical Redux UI binding library for react.

[React]----------------->[React-Redux]------------------------>[Redux]

#Important Points

•React is a library used to build user interfaces
•Redux is a library for managing state in a predictable way in javascript applications
•React-redux is a library that provides bindings to use React and Redux together in an applications

#02
===
Three core Concepts ::

Cake Shop (Real world scenario)

Entities		Activities	
Shop	-Stores cakes on a shelf	Customer	-Buy a cake
Shopkeeper	-At the front of the store	Shopkeeper	-Remove a cake from the shelf
Customer	-At the store entrance		-Receipt to keep track

Comparasion with Technical

[Cake shop scenario	Redux	purpose]
•Shop	Store	Holds the state of your application
•Intention to BUY_CAKE	action	Describes what happened
•Shopkeeper	Reducer	Ties the store and actions together

A store that holds the state of your application
An action that describes the changes in the state of the application
A reducer which actually carries out the state transition depending on action

#Three principals::

#First principle :(state is stored in single object)
'The state of your whole application is stored in an object tree within a single store'.
Maintain our application state in a single object which would be managed by the Redux store.

CakeShop :

Let's assume we are tracking the number of cakes on the shelf(store)

```
{
  numberOfCakes:10
}
```

#Second principle :(emiting an action)
'The only way to change the state is to emit an action,an object describing what happened'
To update the state of your app,you need to let Redux know about that with an action
Not allowed to directly update the state object.

CakeShop :

Let's shopkeeper(Reducer) know about our action -BUY_CAKE

```
{
  type:BUY_CAKE
}
```

#Third principle (state changes by emitted action)

'To specify how the state tree is transformed by action,you write pure reducers'
Reducer-(previousState,action)=>newState

CakeShop :

Reducer is the shopkeeper

```
  const reducer=(state,action)=>{
   switch(action.type){
       case BUY_CAKE:
             return {
                    numberOfCakes:state.numberOfCakes-1
                }
       }
   }
```

Overview of Three principles :

```
                 _(subscribed________ [Javascript App]___________
                |                    /                           |
                |                   / (X)                        |(dispatch)
            [Redux store]     /                               [Action]
           [     state      ]  /                                 |
                |                                                |
                |                                                |
                |______________________[Reducers]_ ___________ _|
```

#Action:
• The only way your application can interact with the store.
• Carry some information from your app to the redux store.
• Plain javaScript objects
• Have a 'type' property that indicates the type of action being performed
• The 'type' property is typically defined as string constants

```
const BUY_CAKE='BUY_CAKE'

//Action or action Creator   [Action creator is function which return action]
function buyCake(){
 return {
   type:BUY_CAKE
  }
}
```

#Reducers:
  • Specify how the app's state changes in response to actions sent to the store.
  • Function that accepts state and action as arguments, and returns the next state of the application
  • (previousState,action)=>newState

```
 //Reducers:  (previousState,action)=>newState
//State
 const initialState={
    numOfCakes:10
 }

 //Reducers
 const reducer =(state=initialState,action)=>{
  switch (action.type) {
    case BUY_CAKE:
         return {
              ...state,                    //copy of state object and update numOfCakes property
             numOfCakes:state.numOfCakes-1
         }
    default: return state
   }
  }
```

#Redux store
 One store for the entire application
 Responsibilities :
  • Holds application state
  • Allow access to state via getState()
  • Allow state to be updated via dispatch(action)
  • Registers listeners via subscribe(listener)
  • Handle unregistering of listeners via the function return by subscribe(listener)

```
  const redux=require('redux')
  const createStore=redux.createStore

 //Redux store
 const store =createStore(reducer)
 console.log('Initail state :',store.getState()) //1:State
 const unsubscribe= store.subscribe(()=> console.log('Updated state :',store.getState())) //2:subscribe
 store.dispatch(buyCake()) //3 :dispatching an action
 store.dispatch(buyCake())
 store.dispatch(buyCake())
 unsubscribe();  //4 :unsubscribing
```

#03

 #MiddleWare
  • It is suggested way to extend Redux with custom functionality
  • It provides a third-party extension point between dispatching an action , and the moment it reaches the reducer
  • Use middleware fro logging,crash reporting,performing asynchronous task etc..
  • For installing redux-logger we can use :npm install redux-logger

#04

  #Actions [Synchronous & Async]
   Synchronous Actions

  • As soon as an action dispatched,the state was immediately updated
  • If you dispatch the BUY_CAKE action,the numOfCakes was right away decremented by 1

   Asynchronous Actions

- Asynchronous API calls to fetch data from endpoint and use that data in your application

Our Application
.Fetches a list of users from an API end point and stores it in the redux store
.State
.Action
.Reducer

.State

```
state={
      loading:true,
      data:[],
     error:''
     }
```

loading	:Display a loading spinner in your component
data	:List of users
error	:Display error Msg to user

.Action

FETCH_USERS_REQUEST	-Fetch list of users
FETCH_USERS_SUCCESS	-Fetched successfully
FETCH_USERS_FAILURE	-Error fetching the data

.Reducer

```
  case :FETCH_USERS_REQUEST
         loading: true
 case : FETCH_USERS_SUCCESS
         loading :false,
         data :from API
case :FETCH_USERS_FAILURE
          loading :false,
         error :from API
```

-- React-Redux---

01:
==
- some examples refer

02
==
React Redux + Hooks

- React redux pattern
- Action creators,reducers,provide the store and connect the components
- Components can access state and dispatch actions
- React Hooks
- In React Redux v7.1 React Hooks introduced in React Redux
- It is used subscribe to store and dispatch actions without connect()

- useSelector hook is provided by react redux library which is equivalent to mapStateToProps function
- useDispatch hook is provided by react redux library which is equivalent to mapDispatchToProps function