

REDUX

- Redux** - is an architecture
- Predictable State Container or Manager
 - Open-Source JavaScript Library

why ... ?

- we have to use Redux for the Application State Management purpose

when ... ?

- when you are developing large scale application

- for example

either Banking,

Insurance,

Health Care or

any complex application

that time, we have to use redux architecture for the application state management

VIDEO – 3 REACT VS REDUX

React JS – Open-Source JavaScript Library

- It is used to develop UI application
- It can't provide global communication between multiple components
- It will can't maintain huge data in the application

Redux – Open-Source JavaScript Library

- Predictable State Container or Manager
- It will maintain whole application state in uni-directional
- It will provide global communication between multiple components

```

graph TD
    App[App] --> Child1[Child 1]
    App --> Child2[Child 2]
    App --> Child3[Child 3]
    Child1 --> Child4[Child 4]
    Child2 --> Child5[Child 5]
    Child3 --> Child7[Child 7]
    Child4 --> Child8[Child 8]
    Child5 --> Child9[Child 9]
    Child7 --> Child10[Child 10]
    Child8 --> Child1[Child 1]
  
```

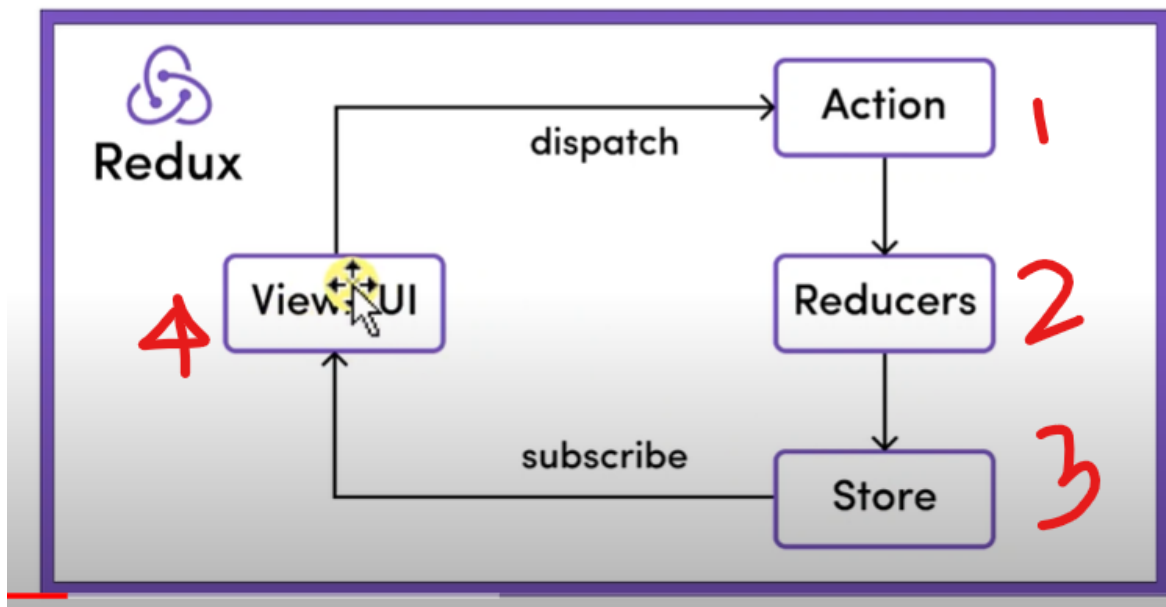


VIDEO – 4 Redux Architecture data flow

=> we have 5 main components

01. action
02. reducer
03. store
04. state
05. view components or UI
react js components

Core Principles of Redux



EXPLANATION OF DIAGRAM →

** All the components in the react will get the data from **store**

** how store know about like data → like what type, which type of data by using → **Reducers**

** **Action means -> object -> data -> state.**

Action contains →

1)payload

2)type

How data will go →

Action-→

store.dispatch()→Reducers(updation)→store(is immutable we cant change anything in store)→subscribe/connect→components

VIDEO – 5 Redux principles

Principles of Redux ... I

01. State is read only
02. Changes Should be made with pure functions
03. Single Source of truth

All 3 explained below →

01. State is read only

=> State - data | information

=> we cannot change the updated State in redux store

=> that means just we can access updated state from redux Store

02. Changes Should be made with pure functions I

=> In Redux, there is only one way to update State is,

only with Pure Functions - that means through Reducer

03. Single Source of truth

=> for entire Redux Application, we have to create single store

that single store can have only updated states

VIDEO – 6 redux life cycle components

life cycle components ... ?

=> we have 5 main components

01. **action**
02. **reducer**
03. **store**
04. **state**
05. **view components** or react js components

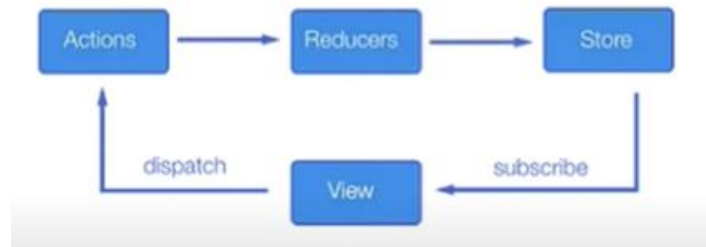
VIDEO – 7 redux life cycle methods

life cycle methods ... ?

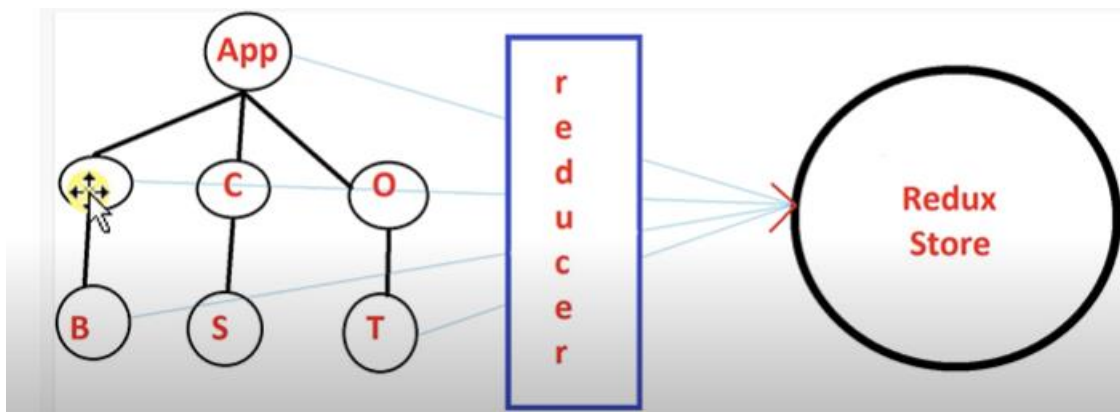
```
React.createStore( root reducer , middleware )  
Store.dispatch( action )  
Store.getState( update State )  
Store.subscribe( listener )  
Store.replaceReducer( nextReducer )  
useSelector( updated State )  
connect() method
```

VIDEO -8

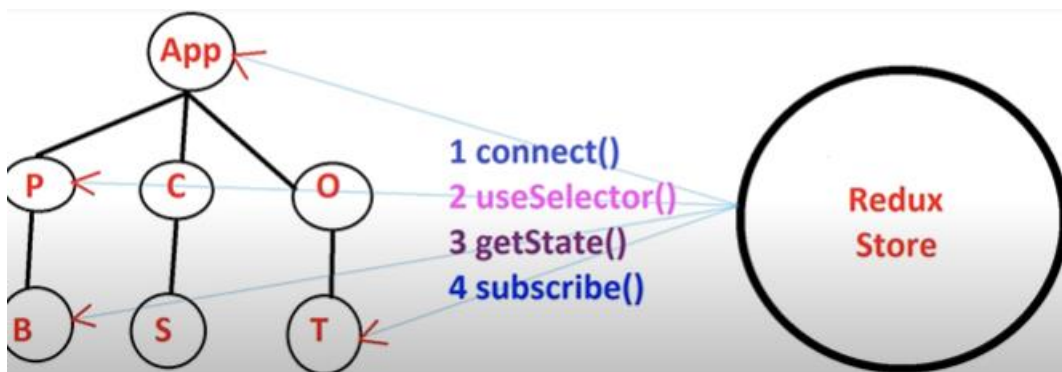
explain about Redux flow , components and methods



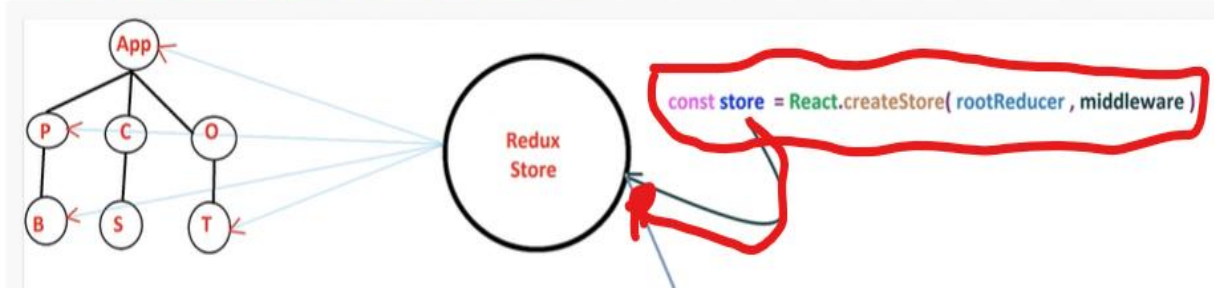
01. Passing data from component to redux Store through reducers



02. Getting updated data from redux store to React Component (view component | UI components)



03.Creating Redux Store



rootReducer -> mandatory

middleware(redux thunk,redux saga)-> optional

VIDEO – 9 ACTIONS IN REDUX

- Action** - Is a pure JavaScript Object or Plain Object
- Payload of the information or
 - Actions are JavaScript object that contains information
 - Actions are the only source of information for the store.

- Action contains – type :
- Payload :

type: add, delete, submit, edit, increase, decrease

Payload – any data

: 1222
: "rgv"
: false | true
: array [10 , 20]
: object { id:10 , name: "rgv" }
: empty | null

It basically carries a payload of information from the application to the store.

It only tells us what has happened in your component

STNTAX

```
Const Actions = {  
  type: "",  
  payload: ""  
}
```

- type – property is mandatory
- payload – is not mandatory

- To change the state, we need to call the store.dispatch()

- using store.dispatch(action) method,

we can dispatch / send action object to the reducer

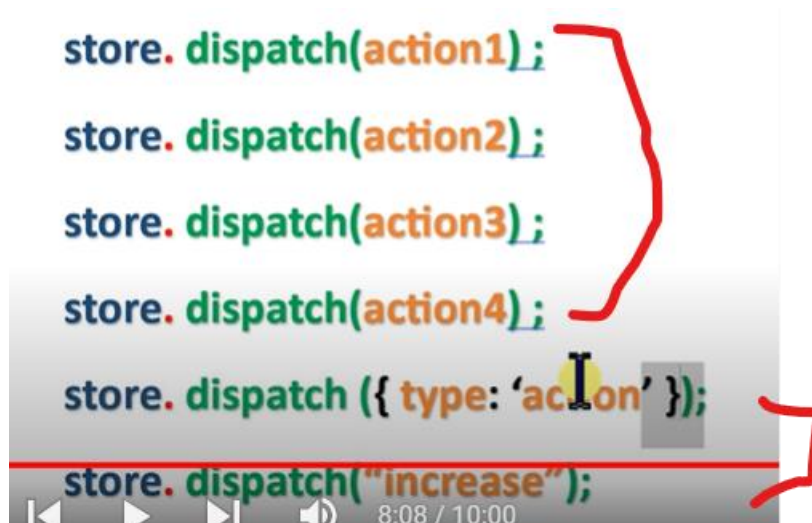
```
const action1 = {  
    type: "add",  
    payload: "rgv"  
}
```

```
const action2 = {  
    type: "add",  
    payload: [10, 20, 30]  
}
```

```
const action3 = {  
    type: "add",  
    payload: {  
        id: 10,  
        name: "ramu"  
    }  
}
```

```
Const action4 = "increase";
```

```
store.dispatch(action1);  
store.dispatch(action2);  
store.dispatch(action3);  
store.dispatch(action4);  
store.dispatch({ type: 'action' });  
store.dispatch("increase");
```



IMP

DIFFERENCE B/W Action and Action Creators

Actions are the plain javascript objects

- Action Creators are the javascript functions that create an action.

// Creating Action creator

```
function AddUser(data)
```

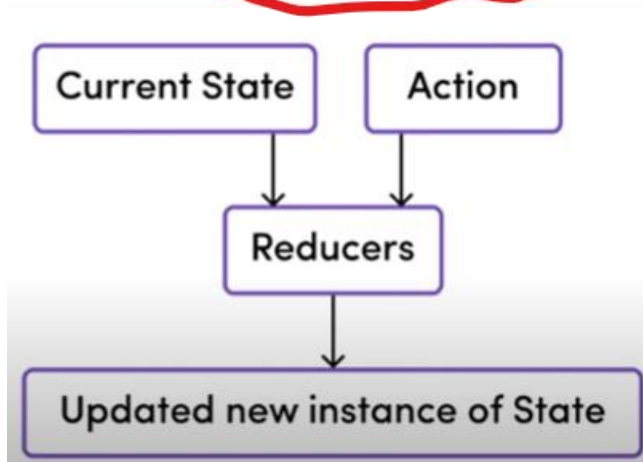
```
{
```

```
  return { type: 'ADD_USER', payload: data }
```

```
}
```

VIDEO – 10 Reducer in Redux

Reducer: The Reducer is a **Pure JavaScript function** that takes the current state and a dispatched action as two inputs, which will produce updated state / new state.



```
const initialState = {  
  Count: 0,  
}
```

```
const counterReducer = (state = initialState, action) => {  
  switch (action.type) {  
    case "INCREASE": return { ...State, State.Count + 1 };  
    case "DECREASE": return { ...State, State.Count - 1 };  
    default: return State;  
  }  
}
```

- depends on your application requirement, you can create multiple reducers

- for example, add, delete, update, edit reducers

- using combine Reducers, we will combine all the reducers.


```
const rootReducer = combineReducers ({  
  |           counter: counterReducer,  
           name: namesReducer  
           })
```

```
const store = createStore(rootReducer)
```

VIDEO – 11 REDUX STORE

Redux Store - The store is the object which holds the immutable updated state of the application.

- Store object acts as an Immutable that means – we cannot edit or change state in Redux Store

STNTAX TO CREATE STORE →

```
const store = React.createStore()
```

- we will create redux Store Object using,
React.createStore() – method

```
const store = createStore(rootReducer)
```

```
<Provider store={store}>
```

```
<App/>
```

```
</Provider>
```

OR WE PASS MIDDLE WARE ALSO

```
const store = createStore(rootReducer, applyMiddleware(thunk /  
saga ))
```

```
<Provider store={store}>
```

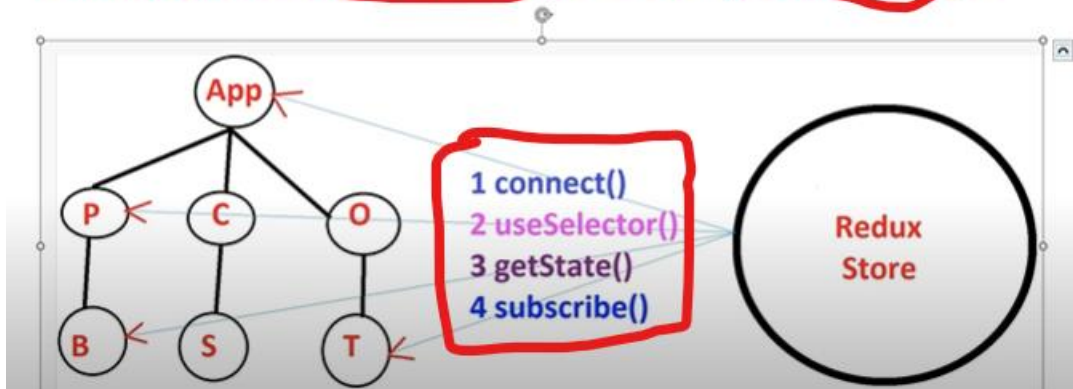
```
<App/>
```

```
</Provider>
```

**WHAT IS USE MIDDLEWARE(THUNK , SAGA) -→
BY DEFAULT REDUX WILL FOLLOW SYNCHORNOUS
SO BY USING MIDDLEWARE(THUNK , SAGA) WE
CAN PERFORM ASYNCHORS ACTIONS**

VIDEO – 12 CONNECT METHOD IN REDUX

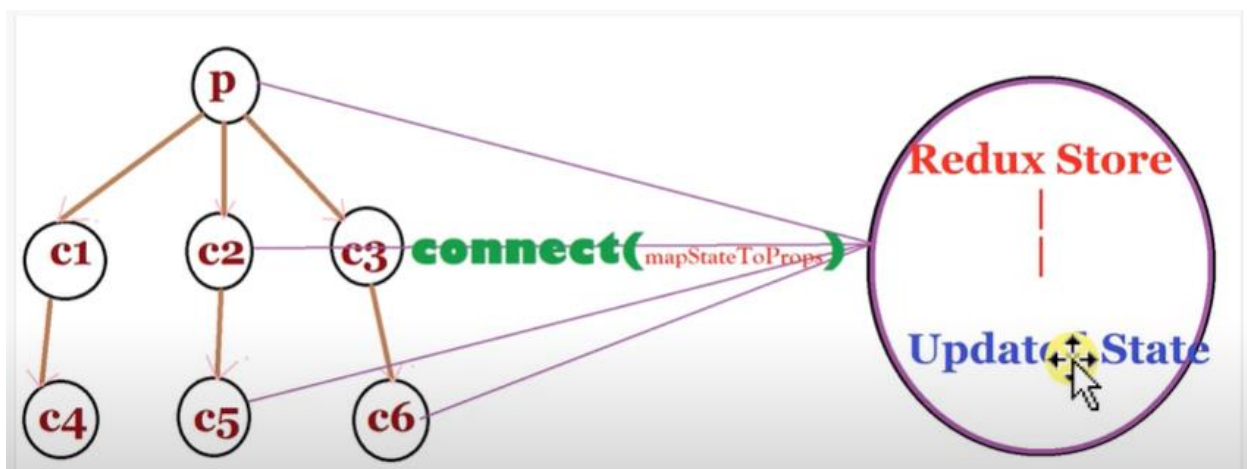
How to get updated State from redux Store to React Components



1)

Connect () in Redux

- ⇒ Connect is a Redux function
- ⇒ It connects a React Components to Redux Store



⇒ Connect () method will take 4 inputs as arguments

```
function connect(mapStateToProps, mapDispatchToProps, mergeProps, options)
```

⇒ mapStateToProps is a function

⇒ It deals with your Redux store's state

⇒ If a mapStateToProps function is specified, the new wrapper component will subscribe to Redux store updates.

Function Employee()

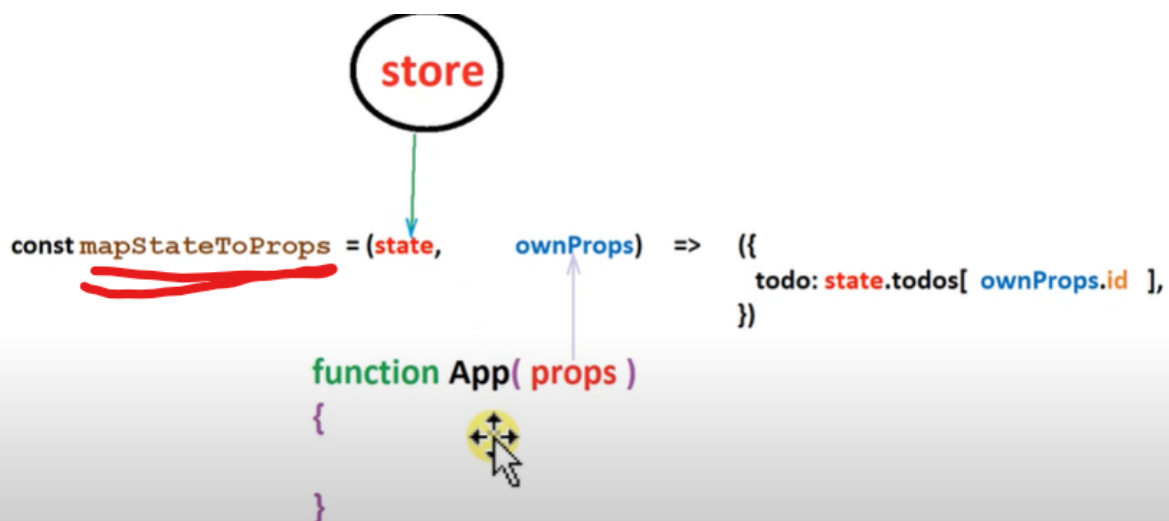
```
{  
  -----;  
}
```

```
function connect(mapStateToProps) (Employee)
```

⇒ This means that any time the store is updated, mapStateToProps will be called.

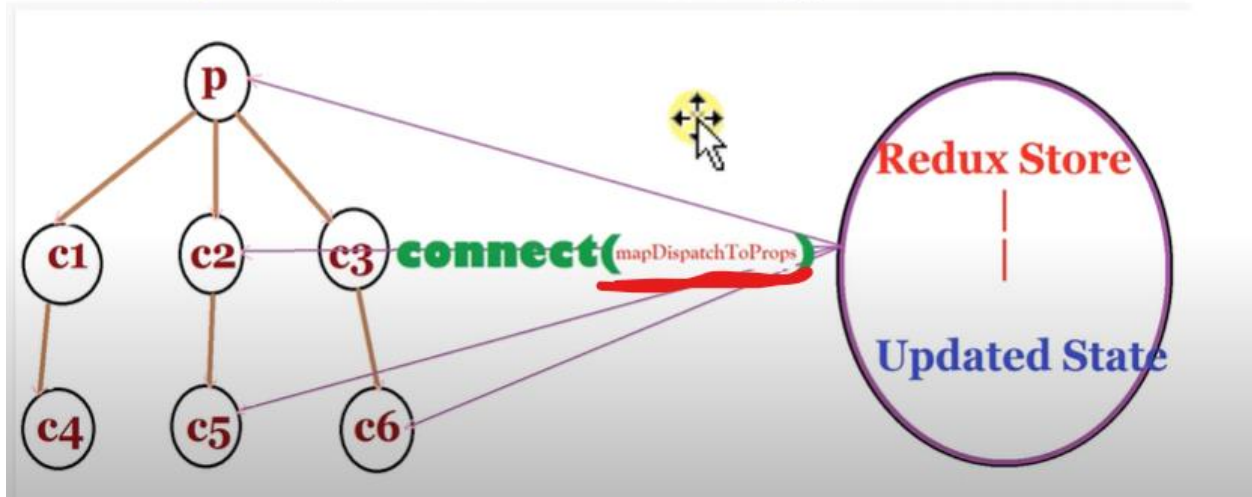
⇒ If your mapStateToProps function is declared as taking two parameters, it will be called whenever the store state changes or when the wrapper component receives new props

⇒ `const mapStateToProps = (state) => ({ todos: state.todos })`



⇒ you have to give this as second parameter to the connect () method

⇒ It may be Object or function or not supplied



VIDEO – 13

`useSelector()` and `useDispatch()`