React Js Online Training

sahosoft

Online Learning Platform

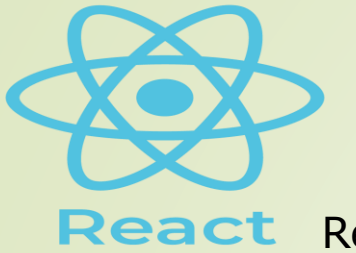
50

By: Ajeet Kumar

sahosoft

Online Learning Platform

Introduction to redux

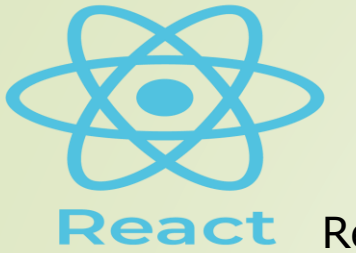


Introduction to redux

Redux is an open-source JavaScript library used to manage application state.

Redux is quite an excellent State Managment Framework usually used with React.js library. In Single Page Application, data management at client side is far more complicated than just imagine. Now, you are familiar that, ReactJS is depends on the State of the application. However,
In React.js state management is possible, but when the application gets bigger and bigger, unwanted errors and data changes are detected, and which module has changed which state and which view is updated, all these matters get complex, and we feel like, we trapped in our application.

Facebook gives the solution. Its developer has created one State management pattern called Flux.



Introduction to redux

Redux is the predictable state container for JavaScript applications. Redux entirely different from Flux.

Flux has multiple stores.

Redux has a Single Store.

Redux cannot 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.

Flux has so many stores, and each store is using different small part of the state or data of our application. In other words, each different module has its store.

Two Principles Of Redux

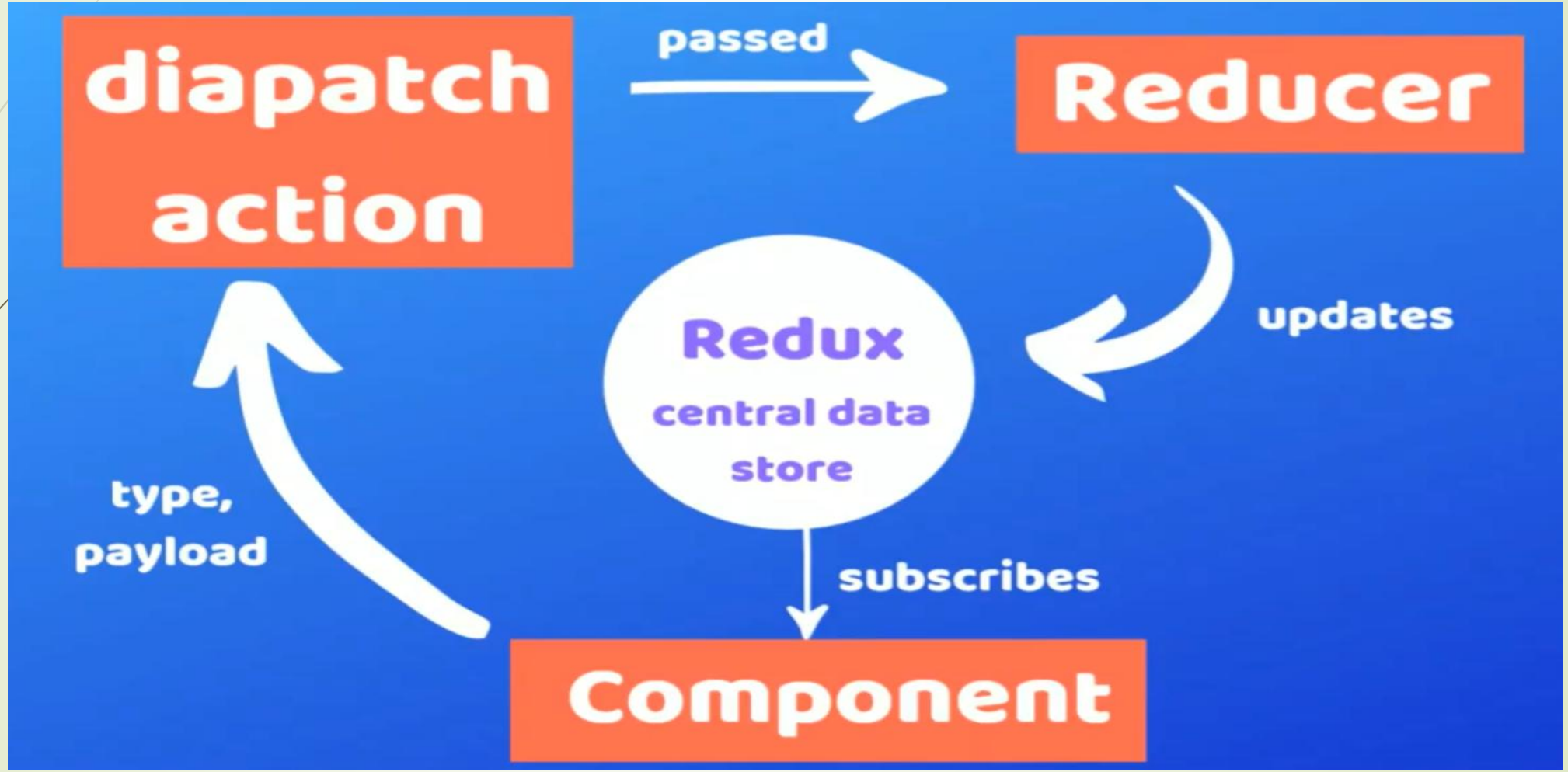
- Single source of truth.
- The state is read-only.

Introduction to redux

What is React Redux

React Redux is the official React binding for Redux. It allows React components to read data from a Redux Store, and dispatch **Actions** to the **Store** to update data. Redux helps apps to scale by providing a sensible way to manage state through a unidirectional data flow model. React Redux is conceptually simple. It subscribes to the Redux store, checks to see if the data which your component wants have changed, and re-renders your component.

Introduction to redux



Introduction to redux

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.

We will learn about all the definitions just to get exposure and an idea of them in your head first.

- **Actions**
- **Reducers**
- **Store**
- **Dispatch**
- **Connect**

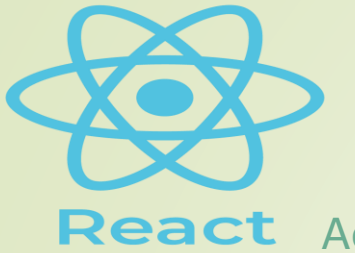
Introduction to redux

Actions

An action is a plain javascript object with two properties: **type** and (optional) **payload**. The type is generally an **uppercase** string (assigned to a constant) that describes the action. The payload is additional data that may be passed.

Actions are payloads of information that send data from your application to your store. Actions are plain JavaScript objects.

Action is sent or dispatched from the view which are payloads that can be read by Reducers.



Introduction to redux

Action

```
{
  type: ‘DELETE_TODO’,
  payload: id
}
```

Action Type

```
const DELETETODO = 'DELETE_TODO'
```

Action

```
{
  type: DELETETODO,
  payload: id,
}
```

Introduction to redux

Action creators

An action creator is a function that returns an action.

Action Creator

```
const deleteTodo = (id) => ({type: DELETE_TODO, payload: id})

deleteTodo(100);
```

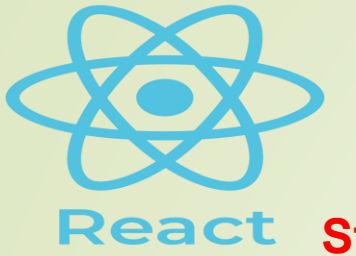
Reducers

A reducer is a function that takes two parameters: state and action. A reducer is immutable and always returns a copy of the entire state.

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

Reducer read the payloads from the actions and then updates the store via the state accordingly. It is a pure function to return a new state from the initial state..

A reducer typically consists of a switch statement that goes through all the possible action types.



Introduction to redux

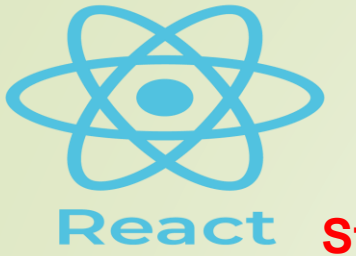
Store

The Redux application state lives in the store, which is initialized with a reducer. When used with React, a <Provider> exists to wrap the application, and anything within the Provider can have access to Redux Store.

A store is an object that brings them together. A Store is a place where the entire state of your application lists. It manages the status of the application and has a dispatch(action) function. It is like a brain responsible for all moving parts in Redux.

A store has the following responsibilities:

- Holds application state;
- Allows access to state;
- Allows state to be updated via dispatch(action);



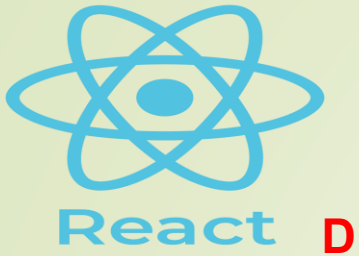
Introduction to redux

Store

Note:

it is important to note that you will only have a single store in a Redux application. If you want to split your data handling logic, you will use reducer composition instead of many stores.

```
import { createStore } from 'redux'
import todoApp from './reducers'
let store = createStore(todoApp)
```



Introduction to redux

Dispatch

dispatch is a method available on the store object that accepts an object which is used to update the Redux state. Usually, this object is the result of invoking an action creator.

Connect

The connect() function is one typical way to connect React to Redux. A connected component is sometimes referred to as a container.

Why should I use Redux?

Why should I use Redux?

- **Easily manage global state -** access or update any part of the state from any Redux-connected component
- **Easily keep track of changes with Redux DevTools -** any action or state change is tracked and easy to follow with Redux. The fact that the entire state of the application is tracked with each change means you can easily do time-travel debugging to move back and forth between changes.

The downside to Redux is that there's a lot of initial boilerplate to set up and maintain (especially if you use plain Redux without Redux Toolkit). A smaller application may not need Redux and may instead benefit from simply using the Context API for global state needs.

In my personal experience, I set up an application with Context alone, and later needed to convert everything over to Redux to make it more maintainable and organized.

Redux dependencies

Redux dependencies

Redux requires a few dependencies.

- Redux - Core library
- React Redux - React bindings for Redux
- Redux Thunk - Async middleware for Redux
- Redux DevTools Extension - Connects Redux app to Redux DevTools

Installation

Installation

To use React Redux with your React app:

npm install react-redux

You'll also need to install Redux and set up a Redux store in your app.

npm install redux

OR

npm install redux react-redux

redux thunk

Extending Redux functionality

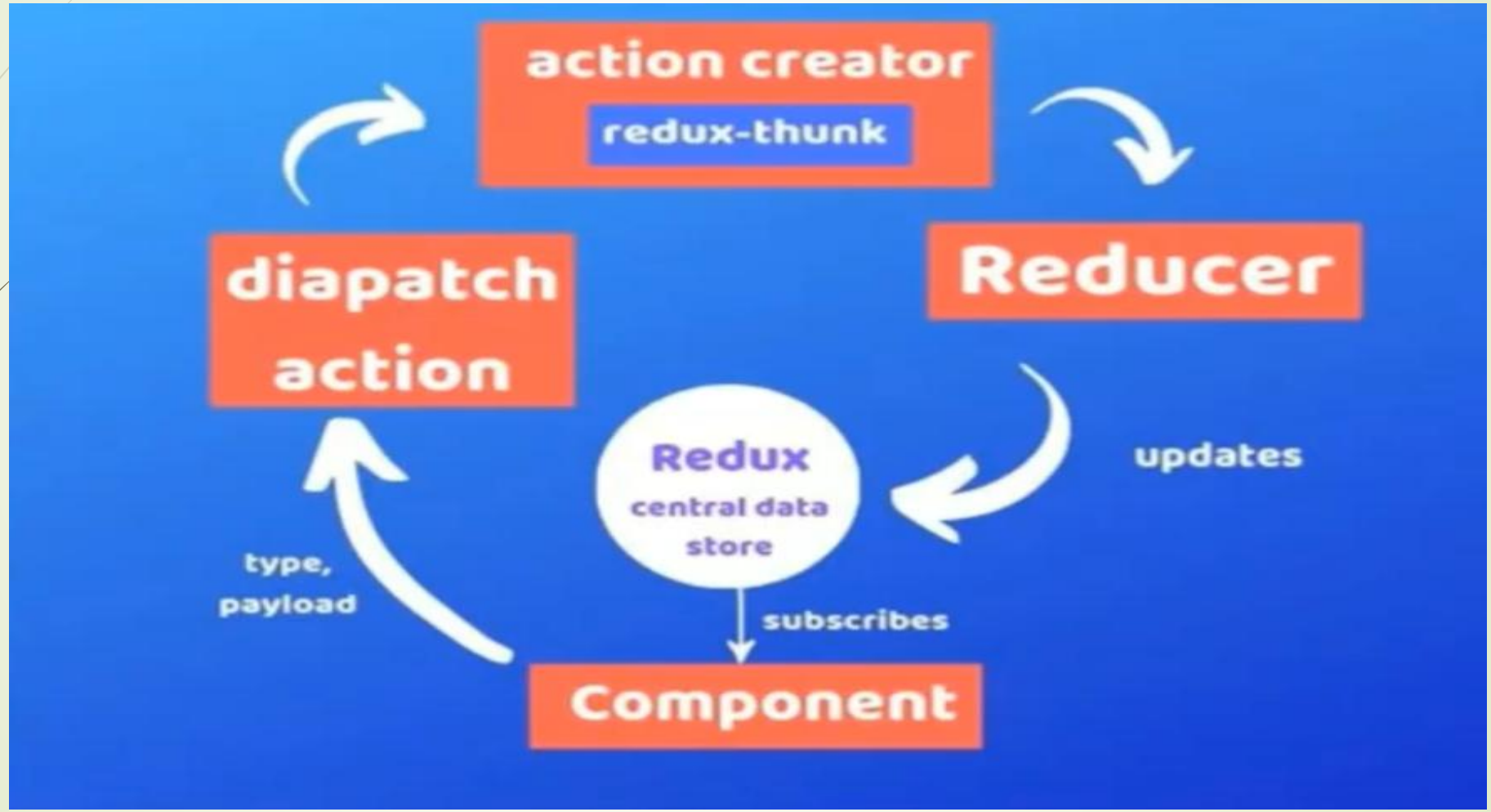
Most apps extend the functionality of their Redux store by adding middleware or store enhancers (note: middleware is common, enhancers are less common). Middleware adds extra functionality to the Redux dispatch function; enhancers add extra functionality to the Redux store.

The redux-thunk middleware, which allows simple asynchronous use of dispatch.

Install redux-thunk

npm install redux-thunk

redux thunk


action creator
redux-thunk
Reducer
updates
Redux
central data
store
subscribes
Component
type,
payload
diapatch
action