STACKTREK

Flux-architecture and Redux

Flux and Redux are both patterns used for managing state in React applications, but they have some key differences. Redux can be seen as an implementation of the Flux pattern with some modifications and additional features to improve developer experience and state management.

Main Differences between Redux and Flux

STACKTREK

Architecture vs. Library:

- Flux: Flux is a design pattern for managing data flow in React applications. It is more of a conceptual architecture rather than a specific library.
- Redux: Redux is a specific JavaScript library that implements the Flux pattern. It is a standalone library designed explicitly for managing state in applications, not just limited to React.

Centralized Store vs. Multiple Stores:

- Flux: In the Flux pattern, data flows through multiple stores. Each store is responsible for managing its own state, and components listen to changes from multiple stores.
- Redux: Redux, on the other hand, has a single centralized store that holds the entire application state. Components can access and modify this state directly, which simplifies state management and makes it easier to reason about data flow.

Main Differences between Redux and Flux

STACKTREK

Immutable State vs. Mutable State:

- Flux: In Flux, the stores often manage mutable state. To update the state, stores emit change events, and components need to listen for these events and retrieve the updated data from the stores.
- Redux: Redux enforces immutability, and state updates are made by dispatching actions. Reducers, pure functions, handle these actions and return a new immutable state. Immutability improves predictability and helps avoid data mutation-related bugs.

Dispatcher vs. Reducers:

- Flux: The Flux pattern relies on a central dispatcher to receive actions and dispatch them to registered stores. The dispatcher is responsible for coordinating data flow.
- Redux: Redux replaces the dispatcher with reducers. Reducers are pure functions that specify how the application state changes in response to dispatched actions. There is no need for a central dispatcher as actions are directly handled by reducers.

Main Differences between Redux and Flux

STACKTREK

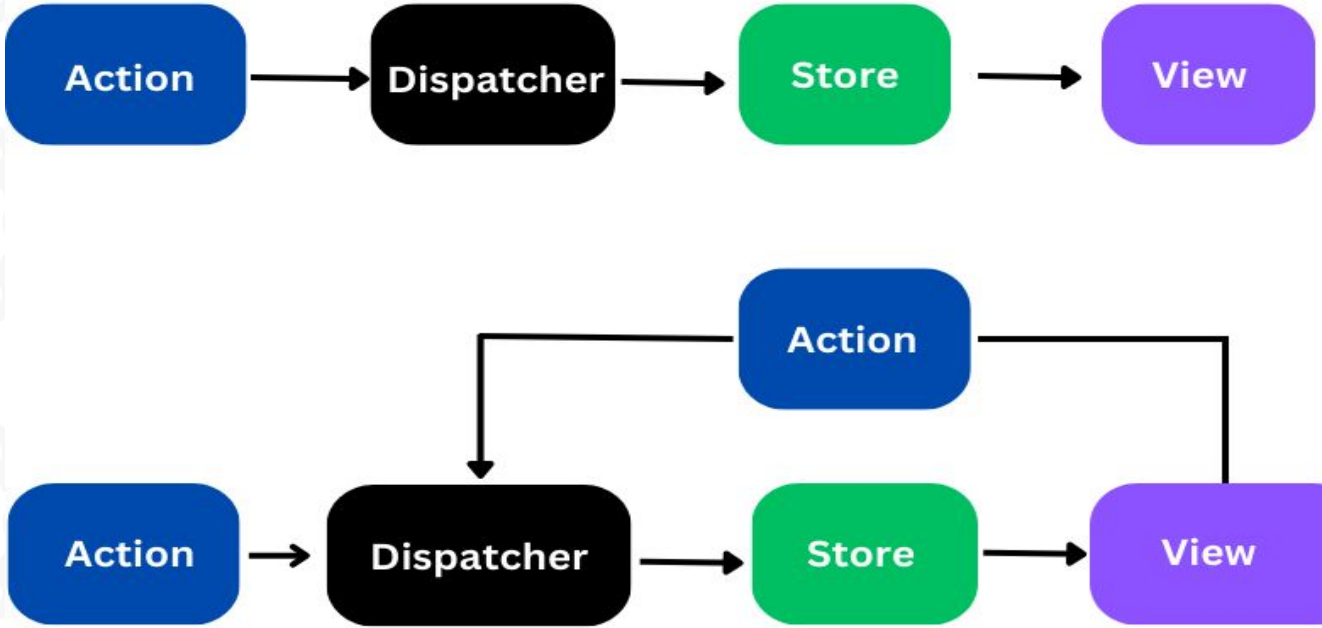
Middleware:

- Redux: Redux supports middleware, which is a powerful feature not found in the original Flux pattern. Middleware sits between dispatching actions and reaching the reducers, allowing developers to intercept, modify, or handle actions asynchronously before they reach the reducers.

STACKTREK

FLUX

- In React, "flux" refers to a design pattern used for managing the flow of data in a React application.
- It was originally introduced by Facebook to address the challenges of data management in large-scale React applications.
- Flux is not a library or framework; it is a pattern that serves as a conceptual guideline for how data should flow through an application.


Action
Dispatcher
Store
View
Action
Action
Dispatcher
Store
View

Main Components of the Flux Pattern

STACKTREK

1. Dispatcher: The dispatcher acts as the central hub that receives actions from various parts of the application and then dispatches those actions to registered callbacks (stores).

2. Actions: Actions are simple objects that represent events or user interactions (e.g., user clicks a button, data is fetched from the server, etc.). These actions are created by action creators and sent to the dispatcher.

3. Stores: Stores are where application state is kept. They register with the dispatcher and receive the actions dispatched by it. Stores manage their data and update themselves based on the action type they receive. They then notify the views about the changes in data, which triggers re-rendering of relevant components.

4. Views (React Components): React components are responsible for rendering the user interface based on the data provided by stores. When stores update their data, React components re-render with the new data, ensuring that the UI stays up to date with the application state.

STACKTREK

REDUX

- Redux is a state management library used with React (and other frameworks) to manage application state in a predictable way.
- It provides a single global state store that holds the entire application state.
- Components can access the state directly from the store instead of passing it down through props.

Key Concepts

STACKTREK

1. Store is a central object that holds the entire application state. It is the heart of the Redux architecture and serves as a single source of truth for the application's data.

2. Actions are plain JavaScript objects representing events or user interactions.

3. Reducers are pure functions that specify how the state changes in response to actions.

4. Dispatch is used to trigger actions and update the state in the store.

5. Selectors are functions to extract specific data from the state.

STACKTREK

UNIVERSE FOR DEVELOPERS