State Management in React JS

By

Narasimha Rao T

Microsoft.Net FSD Trainer

Professional Development Trainer

tnrao.trainer@gmail.com

4. Redux Introduction

Redux is a predictable state container for JavaScript apps, often used with React. It enforces a unidirectional data flow: State is read-only, changes happen via pure functions (reducers), and actions describe "what happened."

- **Core Principles**:
  - Single source of truth (one store).
  - State is read-only (dispatch actions to change).
  - Changes via pure reducers.

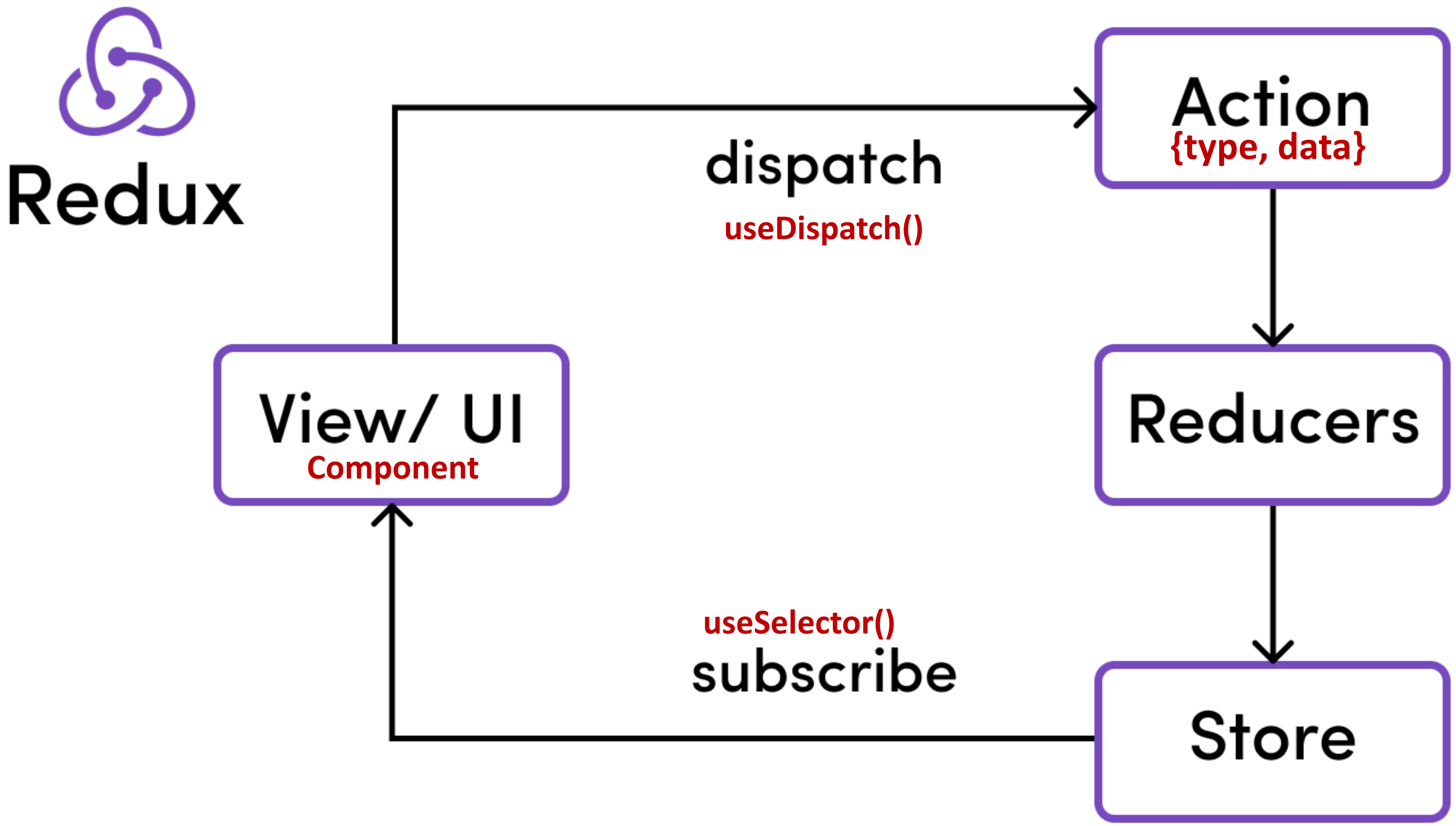
Exploring The Core Redux Concepts

1. Store
2. Actions
3. Reducers

1. **Store**: it brings the actions and reducers together, holding and changing the state for the whole app — there is only one store.

2. **Actions**: An object that have two properties, one describing the **type of action,** and one describing what should be changed in the app state.

3. **Reducers**: Functions that implement the behavior of the actions. They change the state of the app, based on the action.


Redux
Action
{type, data}
dispatch
useDispatch()
View/ UI
Component
Reducers
useSelector()
subscribe
Store

- **Use Cases**: Large apps with complex state interactions (e.g., e-commerce carts, real-time dashboards).
- **Pros**: Predictable, debuggable (time-travel debugging), middleware support (e.g., Redux Thunk for async).
- **Cons**: Boilerplate-heavy; overkill for simple apps.

Redux vs Context API

Aspect	Context API	Redux
Setup Complexity	Simple (built-in, no install)	More boilerplate (install `@reduxjs/toolkit`)
Scalability	Good for small-medium apps	Excellent for large, complex state logic
Performance	Can cause re-renders on value changes	Optimized with selectors (e.g., `useSelector`)
When to Use	Simple shared state (themes, auth)	Global state with actions/reducers

Rule of Thumb: Use Context for quick sharing; Redux for apps needing strict patterns and middleware.

5. Key Players in Redux

Store: Details

The Store is the single JavaScript object that holds the entire application state. It's created once and acts as the "single source of truth."

- **Responsibilities**:
  - Holds the state tree.
  - Allows access to state via `getState()`.
  - Dispatches actions via `dispatch(action)`.
  - Subscribes to changes via `subscribe(listener)`.
- **Creation**: Use `configureStore()` from `@reduxjs/toolkit` (modern way).

- **Key Methods**:
  - `store.getState()`: Returns current state.
  - `store.dispatch(action)`: Triggers reducers.
- **One per App**: Never create multiple stores.

Actions: Details

Actions are plain JavaScript objects that describe "what happened" in the app. They are the only way to trigger state changes.

- **Structure**: `{ type: string, payload: any }` (type is required; payload is optional data).
- **Creation**: Use action creators (functions returning actions) for reusability.
- **Types**: Synchronous (simple objects) or async (via middleware).
- **Example**:

  ```
  const addTodo = (text) => ({
    type: 'todos/add',
    payload: { id: Date.now(), text }
  });
  ```

- **Dispatching**: `dispatch(addTodo('Learn Redux'))`.

Reducers: Details

Reducers are pure functions that specify how the state changes in response to an action. They take current state and action, returning a new state (never mutate!).

- **Signature**: `(state, action) => newState`.
- **Rules**:
    - Pure: Same inputs → same output; no side effects.
    - Immutable: Use spread operators.
    - Initial State: Provide default if `state` is `undefined`.

- Example:

```
const todosReducer = (state = [], action) => {
  switch (action.type) {
    case 'todos/add':
      return [...state, action.payload];
    case 'todos/remove':
      return state.filter(todo => todo.id !== action.payload);
    default:
      return state;
  }
};
```

6. Steps to Implement Redux to Manage State in an Application

Using `@reduxjs/toolkit` (recommended for simplicity):

1. **Install Dependencies:** `npm install @reduxjs/toolkit react-redux`.
2. **Create Reducer**: Create reducers and actions
3. **Configure Store**: Use `configureStore()` to set up the store with reducers and middleware.
4. **Provide Store**: Wrap app in `<Provider store={store}>` from `react-redux`.
5. **Connect Components**:
    - Read state: `useSelector((state) => state.todos)`.
    - Dispatch actions: `useDispatch()`.

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com