What is Redux?

> By definition, Redux is a predictable state container for JavaScript apps.

n other words, Redux is a stand-alone library. It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test.

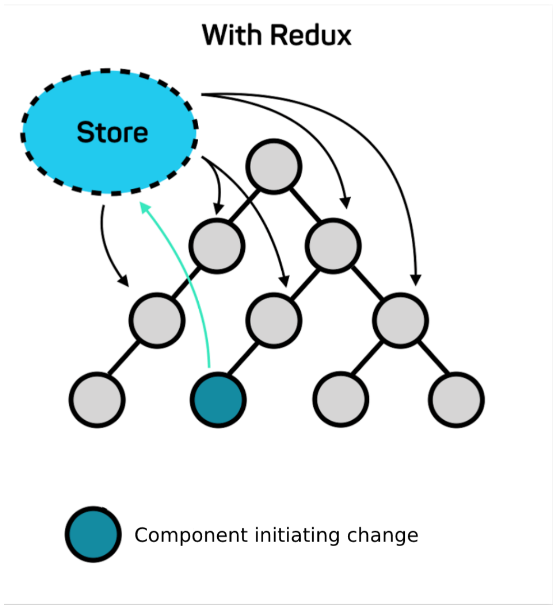
On top of that, it provides a great developer experience, such as live code editing combined with a time-traveling debugger.

Principles of Redux

The predictability of Redux is determined by the three most important principles as given below:

- **Single Source of Truth:** The state of your whole application is stored in an object tree within a single store. As the whole application state is stored in a single tree, it makes debugging easy, and development faster.
- **The state is Read-only:** The only way to change the state is to emit an action, an object describing what happened. This means nobody can directly change the state of your application.
- **Changes are made with pure functions:** To specify how the state tree is transformed by actions, you write pure reducers. A reducer is a central place where state modification takes

place. Reducer is a function that takes state and action as arguments and returns a newly updated state.

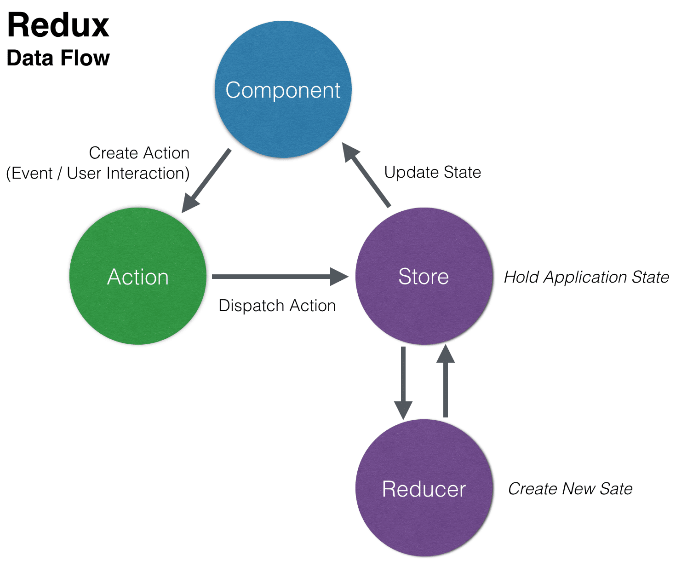


How Redux works?

To fully understand the Redux concept we first need to take a look at the man building block. Redux has three main parts: Actions, Reducers, and Store. Let’s explore what each one does:

- **Actions:** are used to send information from the application to the store. Sending data to the store is needed to change the application state after user interaction, internal events, or API calls.
- **Reducers:** are the most important building block and it’s important to understand the concept.
- **Store:** The store is the central objects that holds the state of the application.

To make it easy to understand, let's imagine this scenario, the component triggers an action to change the value of the global state, The action gets dispatched to the store. The store calls the reducer to create a new state based on the dispatched action. When the new state is created the component updates its view.



‹ Previous

next ›