Reactive

Arnav Gupta

April 19, 2024

Contents

1 Virtual DOM Reconciliation

Reactivity is the automatic update of the UI due to a change in the application's state.

As a developer, focus on the state of the application and let the framework reflect that state in the UI.

The VDOM is a lightweight representation of the UI in memory. It is synchronized with the real DOM as follows:

1. save current VDOM
2. components and/or application state updates the VDOM
3. a re-render is triggered by the framework
4. compare VDOM before update with VDOM after update
5. reconcile the difference, identifying a set of DOM patches
6. perform patch operations on real DOM
7. back to step 1

Reconciliation operations:

- if node type changes, the whole subtree is rebuilt

- if node type is the same, attributes are compared and updated
- if a node is inserted into the list of same node type siblings, all children would be updated (if no more info provided)
  - key prop should be used when updating children of the same node type, where each key must be stable and unique, and the key should be assigned when data is created, not rendered

2 State with Hooks, Context, Signals

Approaches to managing state:

1. `useState` hook for local component state, and pass state to children
2. `useContext` hook to access state, without passing as props
3. Signals

Hook functions are stored in slots associated with each component, and have a slot number associated. A function is used to get/create a hook in a slot.

Hooks in Preact are functional methods to compose state and side effects.

- `useState` used to get and set state.
- `useContext` used to access state context without prop drilling
- `useRef` used to get a reference to a DOM node inside a functional component
- `useEffect` used to trigger side-effects on state change
- `useReducer` used for complex state logic similar to Redux

Can also define custom hooks with funtions that return values and callbacks.

With Context, pass state and other values to children without prop drilling:

1. define context object type
2. create shared context object
3. make context Provider node ancestor of components using context
4. use context in child components

Signals are a state management module introduced by Preact:

- state acts as a model, and signals and mutations can be exported as well

For state with signals:

- keep state minimal and well-organized, so each signal is focused only on a certain part of the state
- use signals only where needed, otherwise performance issues
  - use signals to share state between parts of the application
  - for local app state, use `useState`
- keep components small and focused, so break UI down into small focused components that each manage a specific piece of state