Global State Management

Terms

Client state
Context
Prop drilling
Provider
Reducer
Server state
State management libraries

Summary

- Most React applications have some state to be managed, which can be server or client state.
- React Query helps us manage and cache server state in a simple, elegant way.
- We can define local state in our components using the state or reducer hooks.
- If state management logic becomes too complex and scattered, we can consolidate it using a reducer, which is a pure function that takes the current state and an action, and returns the new state.
- To share state between components, we can lift the state up to the closest parent and pass it down as props, but this can lead to prop drilling in larger, more complex trees.
- Context allows components to share data without having to pass it down manually through each level of the component tree.
- Providers are components that wrap a group of child components and provide them with access to a specific context object.
- We can use custom hooks to access context in a more readable and reusable way.

- When an object contained by a context is updated, all components using that context will re-render.

- To minimize unnecessary re-renders, we should split up a context into smaller, more focused ones with a single responsibility. However, too finely grained contexts can result in complex, hard to maintain component trees.

- If splitting up a context doesn't make sense and we need more control over state updates and component rendering, we can use a state management library.

- There are many state management libraries for React apps. Examples are Redux, MobX, recoil, xState, Zustand, and more.

- These days, React Query and Zustand can replace the need for Redux in many applications.

CONSOLIDATING STATE UPDATES WITH A REDUCER

```tsx
interface Action {
  type: 'INCREMENT' | 'RESET';
}

const counterReducer = (state: number, action: Action): number => {
  if (action.type === 'INCREMENT') return state + 1;
  if (action.type === 'RESET') return 0;
  return state;
}


const Counter = () => {
  const [value, dispatch] = useReducer(counterReducer, 0);

  return (
    <>
      <p>{value}</p>
      <button onClick={() => dispatch({ type: 'INCREMENT' })}>
        Increment
        </button>
    </>
  );
};
```


ACTIONS WITH A PAYLOAD

```ts
interface Task {
  id: number;
  title: string;
}

interface AddTask {
  type: 'ADD';
  task: Task;
}

interface DeleteTask {
  type: 'DELETE';
  taskId: number;
}

type TaskAction = AddTask | DeleteTask;

const tasksReducer = (tasks: Task[], action: TaskAction): Task[] => {
  switch (action.type) {
    case 'ADD':
      return [action.task, ...tasks];
    case 'DELETE':
      return tasks.filter(t => t.id !== action.taskId);
  }
}
```


SHARING STATE WITH CONTEXT

```tsx
interface TasksContextType {
  tasks: Task[];
  dispatch: Dispatch<TaskAction>;
}

const TasksContext = React.createContext<TasksContextType>(
  {} as TasksContextType
);


function App() {
  const [tasks, dispatch] = useReducer(tasksReducer, []);

  return (
      <TasksContext.Provider value={{ tasks, dispatch }}>
        <NavBar />
        <HomePage />
      </TasksContext.Provider>
  );
}
```


CREATING A PROVIDER

```tsx
interface Props {
  children: ReactNode;
}

const TasksProvider = ({ children }: Props) => {
  const [tasks, dispatch] = useReducer(tasksReducer, []);

  return (
    <TasksContext.Provider value={{ tasks, dispatch }}>
      {children}
    </TasksContext.Provider>
  );
};
```

CREATING A CUSTOM HOOK TO ACCESS CONTEXT

```tsx
const useTasks = () => useContext(TasksContext);
```


MANAGING STATE WITH ZUSTAND

```tsx
import { create } from 'zustand';

interface CounterStore {
  value: number;
  increment: () => void;
}

const useCounterStore = create<CounterStore>((set) => ({
  value: 0,
  increment: () => set((store) => ({ value: store.value + 1 })),
}));



const Counter = () => {
  const { value, increment } = useCounterStore();

  return (
    <>
      <p>{value}</p>
      <button onClick={() => increment()}>Increment</button>
    </>
  );
};
```

