Recoil - State Management

Recoil is a state management library for React that aims to provide a more flexible and scalable way to manage state in complex applications. Unlike other state management solutions, Recoil integrates seamlessly with React's Concurrent Mode, offering a modern approach to state management that can handle asynchronous data fetching and complex state dependencies efficiently.

How Recoil works and its key features:

Recoil lets you create a data-flow graph that flows from atoms (shared state) through selectors (pure functions) and down into your React components.

1. Atoms

    - Atoms are the smallest units of state.
    - Any component that subscribes to an atom will re-render when the atom's state changes.
    - Usage:

```
import { atom } from 'recoil';

const textState = atom({
  key: 'textState', // unique ID (with respect to other
atoms/selectors)
  default: '', // default value (aka initial value)
});
```

2. Selectors

    - Selectors are pure functions that derive state from atoms or other selectors.
    - They can compute derived state and are useful for data transformation or aggregation.
    - Selectors can also handle asynchronous operations, making them suitable for fetching data from APIs.
    - Usage:

```
import { selector } from 'recoil';

const charCountState = selector({
    key: 'charCountState', // unique ID
    get: ({get}) => {
        const text = get(textState);
        return text.length;
    },
});
```

- Note: The get property of a selector is a function that gives access to the `{get}` param, a function that allows to retreive value of any recoil atom or selector.

Asynchronous Data using Atoms and Selectors

- Selectors in Recoil can perform asynchronous operations, such as fetching data from an API, and provide derived state.
- Flow:
    - Define a selector to fetch data asynchronously.
    - Use this selector as the default value for an atom.
    - Access the atom's state in React components using Recoil hooks.

```
export const countAtom = atom({
    key: "countAtom",
    default: selector({
        key: "countSelector",
        get: async () => {
            const res = await axios.get("/api-url");
            return res.data;
        }
    })
})
```

Atom Family & Selector Family

`atomFamily()` and `selectorFamily()` are functions used to create multiple instances of atoms and selectors, respectively, based on parameters. These parameters can be dynamic values that determine the specific instance of the atom or selector.

AtomFamily

`atomFamily()` is a function in Recoil that allows you to create atoms dynamically based on a parameter or set of parameters.

Usage: You pass a key and a function to atomFamily. The function takes parameters and returns the default value for the atom. Each unique set of parameters results in a separate instance of the atom.

Example:

```
export const todoAtomFamily = atomFamily({
    key: "todoAtomFamily",
    default: (id) => {
        return todos.find(todo => todo.id == id)
    }
})
```

SelectorFamily

`selectorFamily()` is a function in Recoil that allows you to create selectors dynamically based on a parameter or set of parameters.

Usage: Similar to atomFamily, you provide a key and a function to selectorFamily. This function also takes parameters and returns the selector's value based on those parameters.

Use Case: Fetching data asynchronously with Atom Family and selector family using param `id`

```
export const todoAsyncAtomFamily = atomFamily({
    key: "todoAsyncAtomFamily",
    default: selectorFamily({
        key: "todoAsyncSelectorFamily",
        get: (id) => async ({ get }) => {
            const res = await axios.get(`https://api.example.com/todo?
id=${id}`);
            return res.data.todo;
        }
    })
});
```

Recoil Hooks

Recoil provides several hooks to interact with its state management system in a React application. Here's a detailed explanation of the hooks useRecoilValue, useRecoilState, and useSetRecoilValue, along with examples to illustrate their usage.

1. `useRecoilValue()`
    - Used to read the value of an atom or selector, returns the value as it is.
    - Usage,

      ```
      const text = useRecoilValue(textAtom);
      ```

2. `useRecoilState()`
    - Analogous to useState hook, returns a variable to read the value and a setter function to update the value of an atom or selector.
    - Usage,

      ```
      const [text, setText] = useRecoilState(textAtom);
      ```

3. `useSetRecoilValue()`

- Used to get a setter function to update value of a recoil atom and selector.
- Usage,

```
const setText = useSetRecoilValue(textAtom);
```

Understanding Loadable States

In Recoil, when you're dealing with asynchronous operations or atoms that might potentially fail, it's essential to have a clear understanding of the different states that the atom value can be in. These states are represented using a Loadable object, which can be in one of the following states:

- **Loading**: The atom's value is currently being fetched or computed.
- **Has Value**: The atom's value has successfully resolved and contains valid data.
- **Has Error**: An error occurred while fetching or computing the atom's value.

To get a Loadable object representing the state of an atom's value, you can use the useRecoilStateLoadable or useRecoilValueLoadable hook provided by Recoil.

These hooks return a Loadable object, which contains information about the current state of the atom's value, including whether it is loading, has resolved with data, or has resulted in an error.

1. `useRecoilStateLoadable()`

    - Used to access the state of an atom and subscribe to updates, returning a Loadable object that represents the current state of the atom's value.
    - Usage,

```
    import { useRecoilStateLoadable } from 'recoil';
    import { myAtom } from './state'; // Replace with your atom

    function MyComponent() {
    const [atomLoadable, setAtom] =
useRecoilStateLoadable(myAtom);

    const handleUpdate = () => {
        setAtom(newValue); // Update atom value
    };

        return (
            <div>
            {atomLoadable.state === 'loading' && <p>Loading...
</p>}
            {atomLoadable.state === 'hasValue' && <p>Value:
{atomLoadable.contents}</p>}
            {atomLoadable.state === 'hasError' && <p>Error:
{atomLoadable.contents}</p>}
            <button onClick={handleUpdate}>Update Atom</button>
```

```
            </div>
        );
    }
```

2. `useRecoilValueLoadable()`

    - Used to access the state of an atom without subscribing to updates, returning a Loadable object representing the current state of the atom's value.
    - Usage,

```
    import { useRecoilValueLoadable } from 'recoil';
    import { myAtom } from './state'; // Replace with your atom

        function MyComponent() {
        const atomLoadable = useRecoilValueLoadable(myAtom);

        return (
            <div>
            {atomLoadable.state === 'loading' && <p>Loading...
</p>}
            {atomLoadable.state === 'hasValue' && <p>Value:
{atomLoadable.contents}</p>}
            {atomLoadable.state === 'hasError' && <p>Error:
{atomLoadable.contents}</p>}
            </div>
        );
    }
```