RotateFadeImageApp Using useReducer() Documentation

RotateFadeImageApp Documentation

This Flutter application demonstrates how to manipulate the properties of an image (rotation and opacity) using the `useReducer` hook from the `flutter_hooks` package. It includes several features such as rotating the image, fading it in/out, resetting it to its initial state, and performing undo operations.

Key Concepts

1. **`useReducer`**:
    - The `useReducer` hook is used to manage state changes based on dispatched actions. In this case, it's used to manage the image's rotation, opacity, and history.
    - It provides a state object (`ImageState`) and a `dispatch` function to apply actions (e.g., rotate, fade, reset, etc.) that modify the state.
2. **ImageState**:
    - The `ImageState` class is designed to hold the rotation angle, opacity, and a history of previous states for undo functionality.
3. **Actions**:
    - Actions represent different operations that can be performed on the image (e.g., rotating, fading, etc.).
    - `ImageAction` is an enum that defines the actions available for the image: `rotateLeft`, `rotateRight`, `fadeOut`, `fadeIn`, `reset`, and `undo`.
4. **Image Manipulation**:
    - The app allows users to rotate the image left or right, fade it in or out, undo previous changes, and reset the image to its original state.

Application Structure

1. `ImageState` Class

```
dart
CopyEdit
class ImageState {
  final double rotation; // Rotation angle in degrees
  final double opacity; // Opacity (1 = full, 0 = invisible)
  final List<ImageState> history; // State history for undo

  ImageState({
    required this.rotation,
    required this.opacity,
    required this.history,
  });

  // Copy method to create a new ImageState with updated values
  ImageState copyWith({
    double? rotation,
    double? opacity,
    List<ImageState>? history,
  }) {
    return ImageState(
      rotation: rotation ?? this.rotation,
      opacity: opacity ?? this.opacity,
      history: history ?? this.history,
    );
  }
}
```

- **Purpose**: `ImageState` keeps track of the image's rotation, opacity, and its history.
- `copyWith` **method**: Creates a new instance of `ImageState` with the given updates.

2. `imageReducer` Function

```
dart
CopyEdit
ImageState imageReducer(ImageState state, ImageAction action) {
  switch (action) {
    case ImageAction.rotateLeft:
      return state.copyWith(
        rotation: state.rotation - 15, // Rotate counterclockwise
        history: [...state.history, state],
      );
    case ImageAction.rotateRight:
      return state.copyWith(
        rotation: state.rotation + 15, // Rotate clockwise
        history: [...state.history, state],
      );
    case ImageAction.fadeOut:
      return state.copyWith(
        opacity: (state.opacity - 0.2).clamp(0.0, 1.0), // Reduce opacity
        history: [...state.history, state],
      );
    case ImageAction.fadeIn:
      return state.copyWith(
        opacity: (state.opacity + 0.2).clamp(0.0, 1.0), // Increase opacity
        history: [...state.history, state],
      );
    case ImageAction.reset:
      return ImageState(rotation: 0, opacity: 1, history: []); // Reset state
    case ImageAction.undo:
      if (state.history.isEmpty) return state; // No history to undo
      return state.history.last.copyWith(
        history: state.history.sublist(0, state.history.length - 1),
      );
  }
}
```

- **Purpose**: The `imageReducer` function updates the `ImageState` based on the dispatched action. It modifies the rotation or opacity and maintains a history of states for undo functionality.

3. `RotateFadeImageApp` Widget

```
dart
CopyEdit
class RotateFadeImageApp extends HookWidget {
  const RotateFadeImageApp({super.key});

  @override
  Widget build(BuildContext context) {
    final future = useMemoized(() {
      return NetworkAssetBundle(Uri.parse(imageUrl))
          .load(imageUrl)
          .then((data) => data.buffer.asUint8List());
    }, []);

    final snapshot = useFuture(future); // Fetch image from network
    final imageState = useReducer<ImageState, ImageAction>(
      imageReducer,
      initialState: ImageState(rotation: 0, opacity: 1, history: []),
      initialAction: ImageAction.reset,
    );

    return Scaffold(
      appBar: AppBar(title: Text("Rotate & Fade Image with useReducer")),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            AnimatedOpacity(
              opacity: imageState.state.opacity,
              duration: Duration(milliseconds: 500),
              child: Transform.rotate(
                angle: imageState.state.rotation *
                    (3.1415926535 / 180), // Convert degrees to radians
                child: Image.network(
                  imageUrl, // Example image
                  width: 200,
                  height: 200,
                ),
              ),
            ),
            SizedBox(height: 20),
            Column(
              children: [
                Row(
                  mainAxisAlignment: MainAxisAlignment.spaceAround,
                  children: [
                    ElevatedButton(
                      onPressed: () =>
                          imageState.dispatch(ImageAction.rotateLeft),
                      child: Text("Rotate Left"),
                    ),
                    SizedBox(width: 10),
                    ElevatedButton(
                      onPressed: () =>
                          imageState.dispatch(ImageAction.rotateRight),
                      child: Text("Rotate Right"),
                    ),
                  ],
                ),
                SizedBox(height: 10),
                Row(
                    mainAxisAlignment: MainAxisAlignment.spaceAround,
                    children: [
                      ElevatedButton(
                        onPressed: () =>
                            imageState.dispatch(ImageAction.fadeOut),
                        child: Text("Fade Out"),
                      ),
                      ElevatedButton(
                        onPressed: () =>
                            imageState.dispatch(ImageAction.fadeIn),
                        child: Text("Fade In"),
                      ),
                    ]),
              ],
            ),
            SizedBox(height: 10),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                ElevatedButton(
                  onPressed: () => imageState.dispatch(ImageAction.reset),
                  child: Text("Reset"),
                ),
                SizedBox(width: 10),
                ElevatedButton(
                  onPressed: imageState.state.history.isNotEmpty
                      ? () => imageState.dispatch(ImageAction.undo)
                      : null,
                  child: Text("Undo"),
                ),
              ],
            ),
          ],
        ),
      ),
    );
  }
}
```

- **Purpose**: The `RotateFadeImageApp` widget manages the UI, interacts with the `useReducer` hook to dispatch actions, and controls the image's rotation and opacity.
- `useMemoized` **&** `useFuture`: Used to load the image data from the network asynchronously and render it once fetched.

Features

1. **Rotate Left / Rotate Right**:
   - The image rotates by 15 degrees left or right with each button press.
2. **Fade In / Fade Out**:
   - The image fades in or out by adjusting its opacity in increments of 0.2 (clamped between 0 and 1).
3. **Reset**:
   - Resets the image's rotation to 0 and opacity to 1.
4. **Undo**:
   - Undoes the last image state change (rotation or opacity) by accessing the history stack.

Usage

1. The app starts with a fully visible image in its default orientation.
2. Press **Rotate Left** or **Rotate Right** to rotate the image by 15 degrees.
3. Press **Fade In** or **Fade Out** to adjust the opacity.
4. Press **Reset** to return the image to its default state.
5. Press **Undo** to revert the most recent change.

Conclusion

This application demonstrates effective use of `flutter_hooks` and `useReducer` to manage and manipulate UI state in a declarative manner. It provides a clean and modular way to control image transformations (rotation and opacity) while maintaining an undo history for flexibility.