

# Hooks versus Redux

**Estimated time needed:** 3 minutes

React hooks and Redux are powerful tools for managing state in React applications, each offering unique strengths and suitable use cases. Let's learn about both the tools and their specific utilization in React applications.

## Objectives

After completing this reading, you will be able to:

- Describe the key features, benefits, and limitations of React Hooks
- Describe the key features, benefits, and limitations of Redux
- Compare Hooks and Redux use cases

## React Hooks

### Overview

React hooks are a feature introduced in React 16.8 and are used in functional components to utilize state and other React features without writing a class. Hooks provide a way to reuse stateful logic across components.

### Key Features

- **useState:** This lets functional components set up and care for their own state variables, allowing them to store data that can change over time.
- **useEffect:** This lets functional components do side effects like getting data or changing the DOM after every view, which is how lifecycles work.
- **useContext:** This property lets functional components use context values passed down from parent components. This property makes it easier for data to be shared across the component tree.
- **Custom hooks:** Developers can put stateful logic that can be used repeatedly into custom hooks. This hook makes code reuse easier and separates complex behavior into components.

### Benefits

- **Easy to use:** Hooks are easy to use in React components by putting stateful code inside functional components.
- **Code reusability:** Hooks enable developers to create reusable logic that can be shared across components, improving code organization and maintainability.
- **Reduced boilerplate:** Hooks eliminate the need for class components and reduce boilerplate code associated with state management.

### Limitations

- **Learning curve:** Developers familiar with class components may need time to adapt to the functional paradigm and the nuances of hooks.
- **Complex state management:** While `useState` and `useContext` are suitable for simple state management, more complex applications may require additional patterns or libraries for state management.

## Redux

### Overview

Redux is a container for states that must be kept in JavaScript apps. It's mainly used with React, but you can use it with any other view library or framework. It provides a centralized store for managing application state and enables predictable state updates through actions and reducers.

### Key Concepts

- **Store:** This holds the whole state tree of an app and is the only source of truth.
- **Actions:** These are just plain JavaScript objects that let you change the state.
- **Reducers:** They are used to instruct the app on how to change its state when you perform an action.

### Benefits

- **Predictable state management:** Redux needs data to move only in one direction that cannot be changed. This movement makes it easier to understand and plan for state changes.
- **Centralized state:** Redux keeps all apps' state in one place, which makes it easier for all its parts to see and change data.
- **Debugging and time travel:** Redux DevTools have powerful debugging tools, such as time-travel debugging that lets you see how states change over time.
- **Community and ecosystem:** Redux has a large, active community, and its ecosystem comprises many middleware, services, and development tools.

## Limitations

- Boilerplate: Redux often requires writing more boilerplate code than local component state management or other state management solutions.
- Learning curve: The concepts of Redux, especially actions, reducers, and middleware, can be challenging for beginners or developers unfamiliar with functional programming paradigms.
- Overkill for small applications: For smaller applications with simple state management needs, Redux may introduce unnecessary complexity.

## Comparison

### Use Cases

- There are hooks that can be used to manage the state of a single component, share code between components, and control the state of a single component.
- When more than one part of an app needs to share complex app state, or when the app has a lot of events and complex data flow, Redux is a great tool to use.

### Learning Curve

- Hooks: Relatively easier to learn, especially for developers already familiar with functional components and React's lifecycle methods.
- Redux: Has a steeper learning curve due to its concepts like actions, reducers, and middleware, particularly for developers new to state management patterns.

### Scalability

- Hooks: Well-suited for small to medium-sized applications and simpler state management needs. It may become cumbersome for large-scale applications with extensive state management requirements.
- Redux: Scales well for large applications with complex state management needs, providing a structured approach to manage application state and maintainability.

### Developer Experience

- Hooks: Offer a more streamlined and intuitive development experience for managing state within components, with less boilerplate and a more functional programming style.
- Redux: Offers a strong and reliable state management solution, which is especially helpful for big teams or applications that need to handle complex data flow.

## Conclusion

React hooks and Redux are powerful tools for managing state in React applications, each with strengths and suitable use cases. Hooks offer simplicity, code reusability, and a more functional approach to state management within components. On the other hand, Redux provides a centralized and predictable state management solution, ideal for larger applications with complex data flow requirements. The choice between Hooks and Redux depends on the application's specific needs, complexity, and scale, as well as the preferences and familiarity of the development team.

## Author(s)

Richa Arora

© IBM Corporation. All rights reserved.