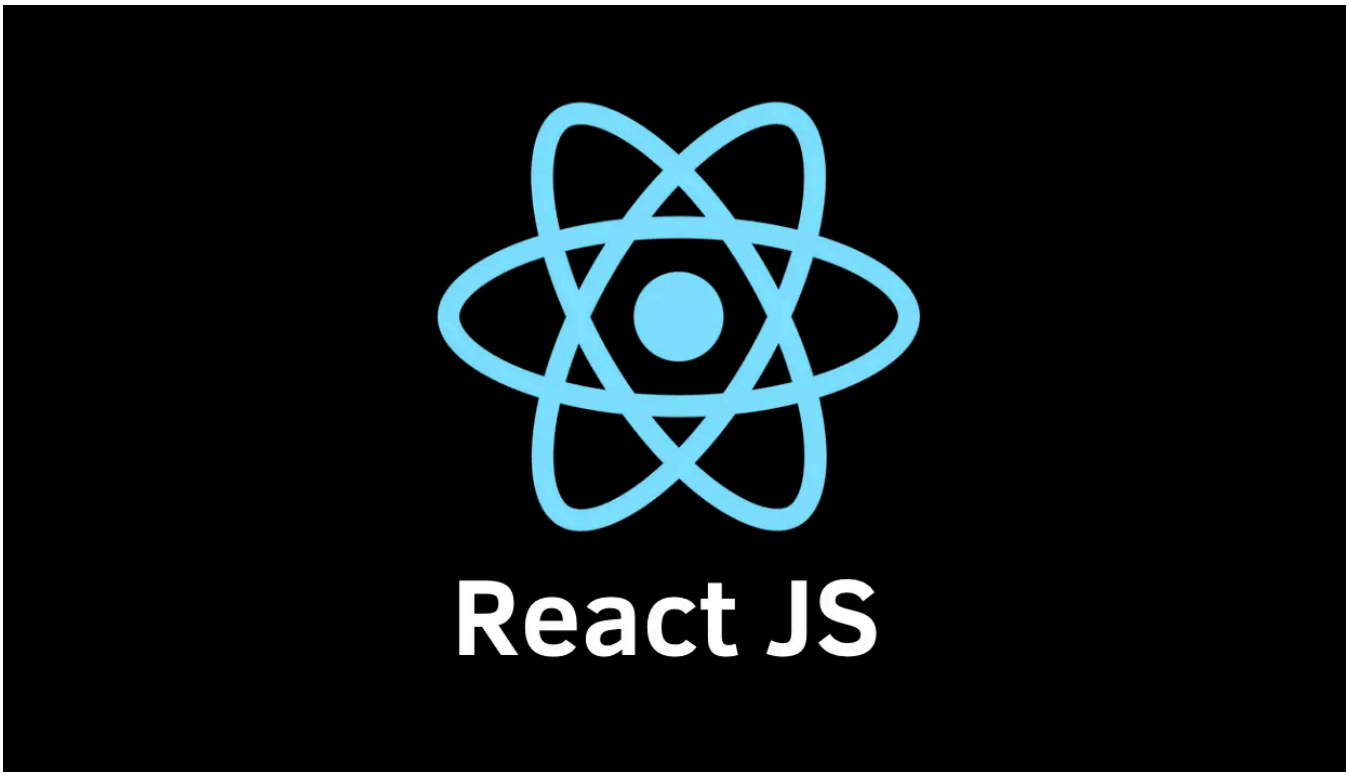
Stackademic · Follow publication

Member-only story

React State Management: What is Prop Drilling

Tech Virtuoso · Follow

Published in Stackademic
6 min read · Mar 14, 2024



> *Anyone can access my article; readers who are not members can follow this link to read the entire text.*

In the process of React development, state management is an unavoidable topic. Both beginners and experienced developers face the challenge of effectively managing component states. React provides us with various state management

solutions, such as direct state passing (commonly known as “prop drilling”), Context API, and external state management libraries like Redux. Each approach has its own use cases and pros and cons. Today, let’s start by discussing what prop drilling is.

What is State Management?

State management is a core and inevitable part of any dynamic application. In React, the state of a component reflects its dynamic attribute values, such as whether a checkbox is checked or what text is entered in a text box. React provides each component with a dynamic data store — through the `this.state` in class components or the `useState()` hook in function components, we can access and modify the internal state of a component. When the component state changes, React automatically re-renders the component to display the latest state.

What are Props?

In React’s component-based development, understanding the concept of Props (properties) is fundamental. Props act as the bridge for communication between components, allowing us to pass data from one component to another. Today, we’ll not only discuss what Props are but also delve into the world of prop drilling to see how it plays a role in React development.

In React, when we define a custom component and use JSX to pass properties and child components, React encapsulates this information into an object — this is what Props refers to. With Props, we can easily achieve data passing and reuse between components.

For example, the following code demonstrates how to use Props to display “Hello, Hulk” on the page:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
const element = <Welcome name="Hulk" />;
root.render(element);
```

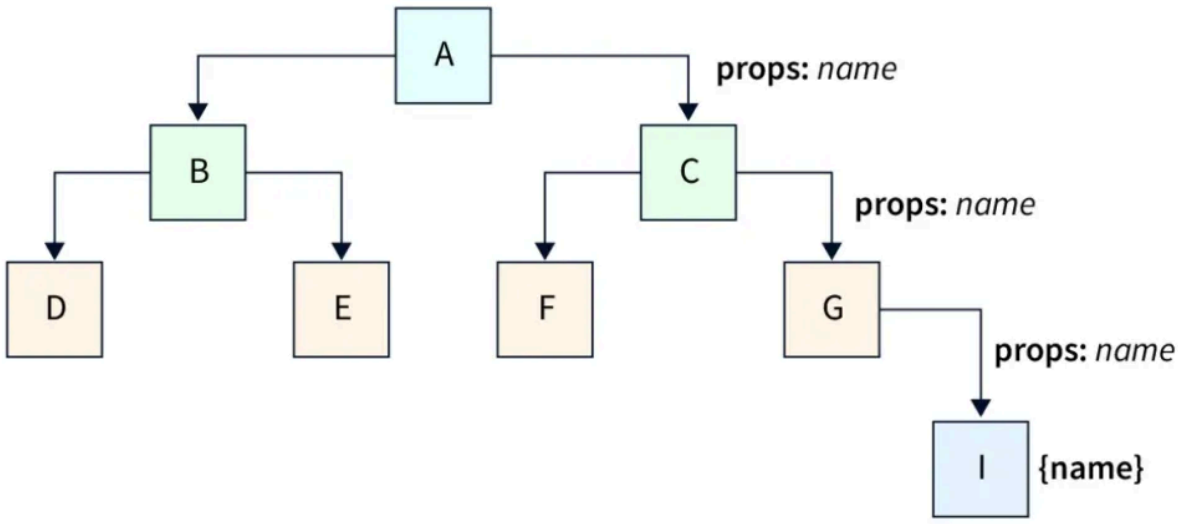
What is Prop Drilling?

In a typical React application, data often needs to be passed between components via Props. When dealing with deeply nested components, manually sharing this data can become complex and challenging. Additionally, sharing data between two child components can be even more cumbersome. In such cases, a global state management approach is needed to simplify this process.

Prop drilling refers to the process in React where data needs to be passed through multiple interconnected components to reach the component that ultimately needs it. This process is called “drilling” because it forces every intermediate component to receive unnecessary data and pass it on to the next component, and so on until the data reaches its destination. This approach can significantly impact the reusability of components and the performance of the application.

When writing clean, reusable code that adheres to the DRY (Don’t Repeat Yourself) principle, passing data through multiple components may not be a good practice.

However, prop drilling is sometimes beneficial for smaller applications, as there are fewer components and conditions to manage.



Why Avoid Prop Drilling?

In React application development, prop drilling is a common pattern that involves passing props from one component through multiple levels to another component. While this approach may be viable in some cases, it is generally recommended to avoid prop drilling for the following reasons:

Maintenance Issues

Prop drilling requires developers to manually pass state and data through all intermediate layers that do not need it to update the state of components lower in the tree. This results in lengthy and difficult-to-maintain code. Whenever you need to modify, add, or remove state, you may need to modify the way props are passed between multiple components, increasing maintenance costs.

Increased Error Possibilities

Renaming issues: It is easy to accidentally change the names of props during the passing process, leading to data interruptions or errors.

Structural refactoring: When refactoring certain data structures, it is necessary to ensure that all components receiving that prop are adjusted accordingly, which is prone to errors.

Overpassing: Sometimes, some props are not needed at certain intermediate levels but are still passed, leading to unnecessary complexity and performance loss.

Improper use of default props: Improper or insufficient use of default props may result in unexpected behavior, increasing debugging difficulty.

Complexity in Large Projects

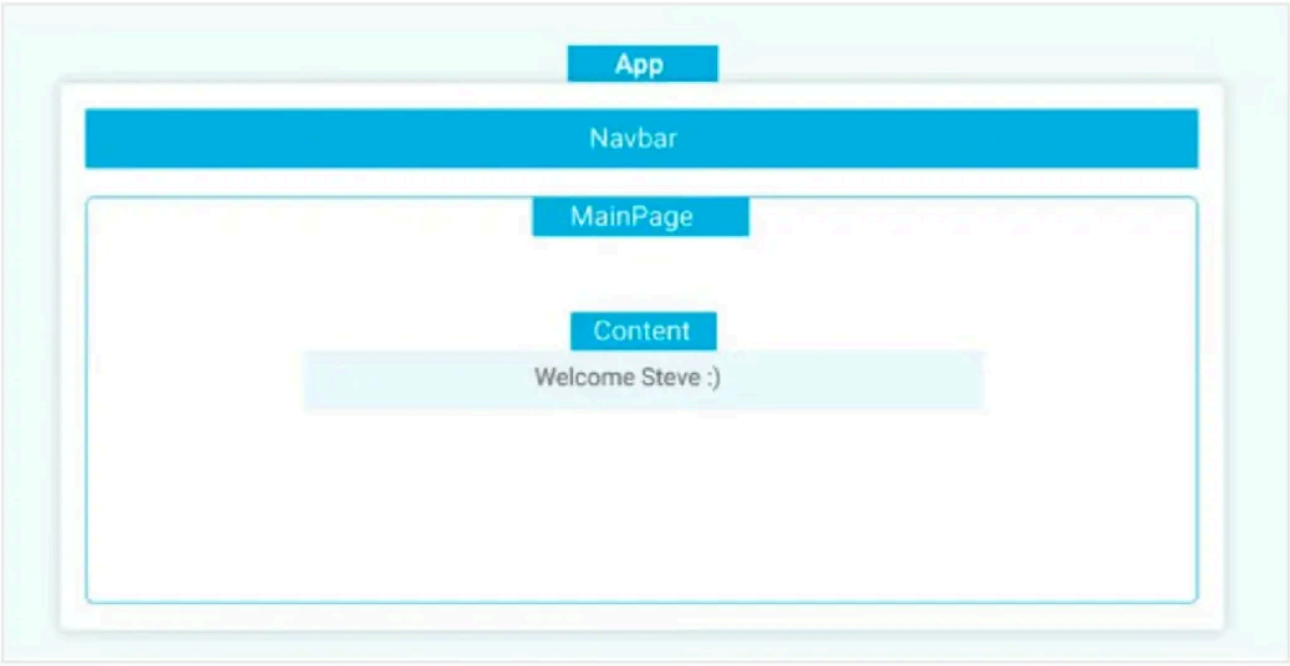
In large projects, prop drilling can be particularly frustrating. The component hierarchy may be very deep, making it very complex to track the flow of a prop during maintenance and refactoring, especially when multiple teams or modules are involved, coordinating changes can be very difficult.

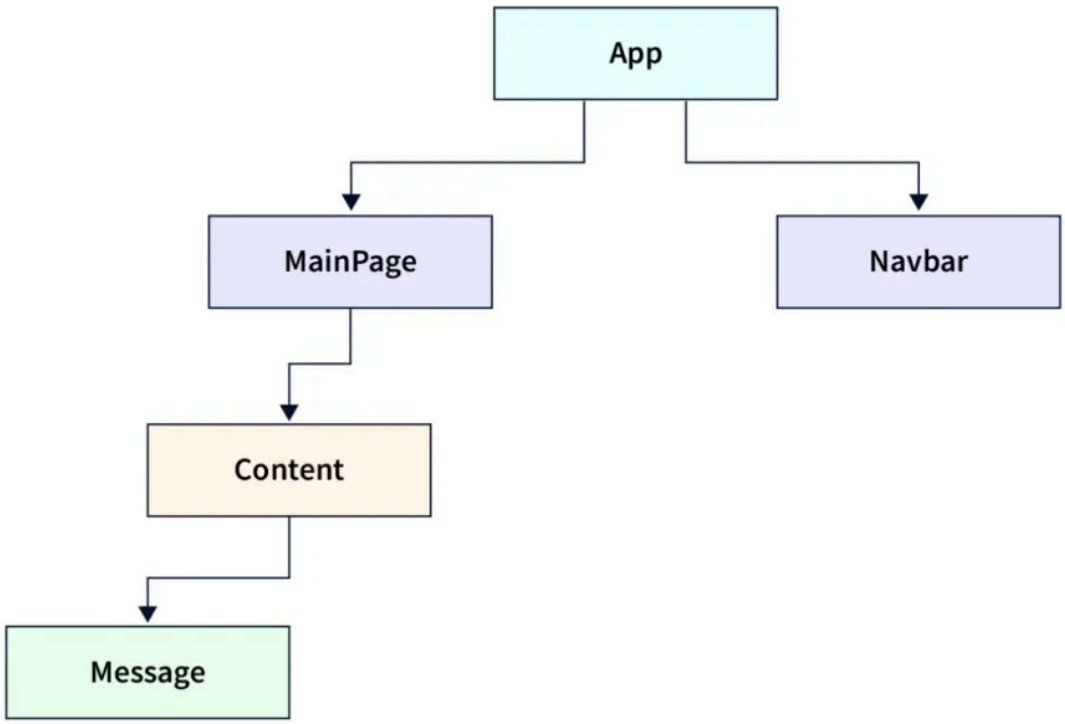
Performance Impact

Although React efficiently handles most performance issues, unnecessary prop passing can cause unnecessary component re-renders, especially in large applications, leading to performance degradation.

A Simple Example to Explore Prop Drilling

Suppose we are developing an application where, after the user logs in, a welcome message with the user's name is displayed on the page. The structure of our application is roughly as follows:





App Component: This is the root component, which has the user's state information.

Navbar Component: Displays the navigation bar of the application.

MainPage Component: The main page component, which needs to pass user information to its child components.

Content Component: The content component, which also needs to pass user information to its child components.

Message Component: The message component that actually displays the welcome message, requiring user information.

```
import { useState } from 'react';

function App() {
  const [user, setUser] = useState({ name: 'Aegon' });
  return (
    <div>
      <Navbar />
      <MainPage user={user} />
    </div>
  );
}

function Navbar() {
  return <nav style={{ background: '#10ADDE', color: '#fff' }}>Demo App</nav>;
}

function MainPage({ user }) {
  return (
    <div>
      <h3>Main Page</h3>
      <Content user={user} />
    </div>
  );
}

function Content({ user }) {
  return (
    <div>
      <Message user={user} />
    </div>
  );
}

function Message({ user }) {
  return <p>Welcome {user.name}</p>;
}

export default App;
```

In the above example, we pass the `user` object layer by layer, ultimately passing it to the `Message` component. While this approach is straightforward, as the application scales, it introduces unnecessary complexity, increases component coupling, and makes tracking and managing data flow difficult.

How to Fix Prop Drilling Issues

To avoid prop drilling issues, React provides a powerful API — the Context API. The Context API allows developers to directly pass data across component layers without manually passing props through each layer.

By using Context, we can create a context containing user information and provide the value of this context in the `App` component. This way, any component needing this information can consume these values through Context without passing them through intermediate components.

After refactoring with the Context API, the code will be more concise, and the coupling between components will be greatly reduced, making data flow management more intuitive and easier to maintain.

I will discuss more about the Context API in the next article. Don't forget to give me a clap or follow me for in-depth technical articles and the latest interpretations of front-end trends to help you stay ahead and competitive in technology. Your support is my greatest support and the motivation for me to continue sharing high-quality content.

Stackademic 🎓

Thank you for reading until the end. Before you go:

- Please consider **clapping** and **following** the writer! 👏
- Follow us **X** | **LinkedIn** | **YouTube** | **Discord**
- Visit our other platforms: **In Plain English** | **CoFeed** | **Venture** | **Cubed**
- More content at **Stackademic.com**

React | Reactjs | JavaScript | Front End Development