Compare useEffect, useLayoutEffect and useEffectEvent in React

In this article, let's explore React's data fetching hooks - useEffect, useLayoutEffect, and useEffectEvent - comparing their functionalities for efficient application deployment.

In this article, let's explore React's data fetching hooks - useEffect, useLayoutEffect, and useEffectEvent - comparing their functionalities for efficient application deployment.

React hooks provide an efficient way to manage side effects in React components. 3 of the most popular hooks are useEffect, useLayoutEffect, and useEffectEvent. Each hook has its own usage, let's compare them together to make the right choice when you program React!

useEffect

useEffect is a basic hook in React that allows you to implement side effects like DOM editing, asynchronous operations, and data fetching in functional components. This hook is a function with two arguments: effect function and dependency array.

The effect function contains the code that implements the secondary effect, and the dependency array determines when the effect runs. If the dependency array is empty, the effect function runs only once on the initial rendering of the component. Otherwise, the effect function runs whenever the value in the dependent array changes.

Here's an example of using the useEffect hook to fetch data:

```
import React from "react"; function App() { const [data, setData] = React.useSta
```

```
") .then((response) => response.json()) .then((data) => setData(data)); ]
```

This code demonstrates an App component that fetches data from an external API using the **useEffect** hook . **UseEffect** 's **Effect function** fetches sample data from the JSONPlaceholder API. It then parses the JSON response and sets the retrieved data to data.

With data state, the **App** component displays the **title** attribute of each item in the state.

Characteristics of the useEffect hook

1. 'Friendly' to asynchronous operation, making data fetching more convenient.
2. Run after rendering the component, making sure the hook doesn't block the UI.
3. Implement cleanup by returning a function.

useLayoutEffect

useLayoutEffect is similar to the useEffect hook but runs synchronously after all DOM mutations. This means it runs before the browser can cover the screen, making it suitable for tasks that require precise control over DOM layout and style, such as measuring an element's size, resizing it, or animate its position.

Here's an example of using the **useLayoutEffect** hook to change the width of a button element:

```
import React from "react"; function App() { const button = React.useRef(); React
```

Click Me

```
); } export default App;
```

The above code block increases the width of the button element to 12 pixels using the useLayoutEffect hook. This ensures that the button width increases before the button appears on the screen.

Outstanding features of the useLayoutEffect hook

1. Synchronous implementation, capable of blocking the UI if operations within it are heavy.
2. Best choice for reading and writing data directly to the DOM.

useEffectEvent

useEffectEvent is a React hook that solves the dependencies of the useEffect hook. If you're familiar with useEffect, you should know that its array of dependencies can be complex. Sometimes you have to put more values ??in the dependency array than is absolutely necessary.

For example:

```
import React from "react"; function App() { const connect = (url) => { // logic
```

This code shows the **App** component that manages connections to external services. The connect function connects to a specific URL, while the **logConnection** function records the connection details. Finally, the **onConnected** function calls **logConnection** to log a successful connection message when the device connects.

The useEffect hook calls the connect function, then sets up an **onConnected** callback function to run when the device fires the **onConnected** event . This callback logs a connection message. It returns a cleanup function that fires when the component is unmounted. The cleanup function is responsible for disconnecting that device.

The dependency array has a url variable and an **onConnected** function . The App component will create an onConnected function on each render. This will cause the **useEffect** function to run on a loop that continuously renders the **App** component .

There are many ways to handle the problem of repeating useEffect. However, the most efficient way to do this without adding unnecessary values ??to the dependency array is via the useEffectEvent hook.

```
import React from "react"; function App() { const connect = (url) => { // logic
```

By wrapping the **onConnected** function with the useEffectEvent hook, the useEffectEvent hook can always read the latest value of the message **and** loginOptions **parameters** before passing it to the useEffect hook. This means **useEffect** does not need to rely on the **onConnected** function or the value passed to it.

useEffectEvent is useful when you want **useEffect** to depend on a specific value, although this effect fires an event that requires other values ??that you don't want as dependencies in useEffect.

Features of the useEffectEvent hook

1. Best suited for event-driven application effects.
2. The useEffectEvent hook does not work with event handlers like onClick, onChange…

In summary, each of the above fetch hooks is suitable for different situations:

1. **Fetch data** : useEffect is a great choice.
2. **Direct DOM manipulation** : If you need to make synchronous changes to the DOM before repainting, choose useLayoutEffect.
3. **Lightweight operations** : For operations that don't risk blocking the user interface, you can feel free to use useEffect.
4. **Event-driven side effects** : Use the useEffectEvent hook to encapsulate events and the useEffect hook to run side effects.

ReactJS hooks open up a world of possibilities, and understanding the differences between useEffect, useLayoutEffect, and useEffectEvent hooks can significantly impact how you handle side effects and DOM manipulation.