**React: You are Using useEffect() Wrong, Do This Instead**

[[Marcelo Botelho](https://medium.com/@botelhomarcelo7?source=post_page-----740660e33420--------------------------------)](https://medium.com/@botelhomarcelo7?source=post_page-----740660e33420--------------------------------)

[[Stackademic](https://blog.stackademic.com/?source=post_page-----740660e33420--------------------------------)](https://blog.stackademic.com/?source=post_page-----740660e33420--------------------------------)

[Marcelo Botelho](https://medium.com/@botelhomarcelo7?source=post_page-----740660e33420--------------------------------)

·

Follow

Published in

[Stackademic](https://blog.stackademic.com/?source=post_page-----740660e33420--------------------------------)

·

5 min read

·

6 days ago

53

5



Once one decides to move forward with learning **React**, hooks are among the first things to learn (and to be frustrated with). Hooks are essential parts of **React**, as they were created to solve several problems that appeared in the first couple of versions of **React**, when every rendering was done inside the component’s lifecycle functions, such as **componentDidMount()**, **componentWillMout(),** **componentDidUpdate()**.

That said, the first hooks everyone starts touching are **useState()** and **useEffect()**. The first is used for state management and control when the component should be rendered again, while the second behaves somewhat similarly to the lifecycle functions stated above.

The **useEffect()**hookcan receive two outputs: the first is a callback function, while the second is optional and defines when this hook should be called.

useEffect((prevProps) => { //prevProps are optional and has some specific uses. Compare with what happens with the lifecycle functions.  
 //Custom function content….  
 custom function content…  
  
 return () => {  
 // Code to run when the component is unmounted or when dependencies change  
 // It helps in avoiding memory leaks and unexpected behavior  
 };  
 }, [dependencies in array form]);

One caveat that gets a lot of beginners is how the second parameter works. Here is a resume:

*Case A: If nothing is added, then****useEffect****will run at every change of state inside the current component.*

*Case B: If an empty array is added ([]), then the****useEffect****will run only once when the component is mounted.*

*Case C: If some array is provided ([state]), then****useEffect****will run every time the state changes*

*Case C\*: If some array is provided ([state1, state2, ….],****useEffect****will run every time****ANY****of these states changes.*

Now that we recalled how **useEffect** works, it’s time to state some of the drawbacks of using it as most people do.

The main idea of the **useEffect** hook is to synchronize data transfer with external **APIs** or another system, like when you are accessing a database, or waiting for an **HTTP** request to complete. The trouble is that we tend to use this hook in every situation possible inside our code, especially **Case A** and C\* listed above, and the code can become incredibly unreadable with just a couple of lines of code, including triggering a loop if you change one of the states in the dependency array during the process.

This can make your code inefficient too, as useEffect works as if you were stepping aside to run some code and then coming back to the main thread. This is inefficient.

Great, now you know why to avoid it. But how?

Let’s talk about each one of the Cases in detail:

*Case A — No dependency array: This one should be abolished from your code, as it will certainly trigger unnecessary calculations every time a state changes. In this case, you should specify which states really should trigger this function using a dependency array.*

*Case B — Empty dependency array: This is one of the good ones, the only recommendation that I can provide is to keep just one of these for each component and wrap its content into a function.*

*Case C — One dependency state only. It’s ok to use if you are processing external data. Otherwise, you should change it to the solution I will provide below.*

*Case C\* — Multiple dependency states in the same****useEffect****. This is the one I consider the most troublesome. I recommend you try to untangle the states into different useEffect hooks before anything, as it makes your code very unreadable.*

Now for the solution that I promised. Let’s consider these two Components (Parent and Child):

// ParentComponent.js  
import React, { useState, useEffect } from 'react';  
import ChildComponent from './ChildComponent';  
  
function ParentComponent() {  
 const [count, setCount] = useState(0);  
 const [message, setMessage] = useState('Hello from Parent!');  
  
useEffect(() => {  
 setMessage(`Button clicked ${count} times!`);  
},[count]}  
  
 return (  
 <ChildComponent count={count} message={message} setCount={setCount} />  
 );  
}  
  
export default ParentComponent;  
  
  
  
// ChildComponent.js  
import React from 'react';  
  
function ChildComponent({ count, message, setCount }) {  
 return (  
 <div>  
 <h3>Child Component</h3>  
 <p>Received Count from Parent: {count}</p>  
 <p>Received Message from Parent: {message}</p>  
 <button onClick={() => {setCount(count+1)}>Click Me</button>  
 </div>  
 );  
}  
  
export default ChildComponent;

Now let’s explain what is happening here:

*1 — Once the user clicks the button on the****ChildComponent****, we change the state “count” by incrementing 1. It will take one render loop to happen and change the state.*

*2 — Once the state “count” changes, the child component will be rendered again and it will also trigger the****useEffect****hook on both components, which will trigger the change in the “message” state. Again, it will only happen in the next render.*

*3 — When the “message” state changes, then another render happens in the components changing the message.*

*In this case, we ended up having two renders. It may now seem much, but it can grow at a large scale once you have more states at play.*

*Now look what happens when we make the following changes in the components:*

// ParentComponent.js  
import React, { useState } from 'react';  
import ChildComponent from './ChildComponent';  
  
function ParentComponent() {  
 const [count, setCount] = useState(0);  
 const [message, setMessage] = useState('Hello from Parent!');  
const incrementCount = () => {  
 setCount(count + 1);  
setMessage(`Button clicked ${count + 1} times!`);  
}  
  
 return (  
 <ChildComponent count={count} message={message}   
 callbackFunction={incrementCount} />  
 );  
}  
  
export default ParentComponent;  
  
  
// ChildComponent.js  
import React from 'react';  
  
function ChildComponent({ count, message, callbackFunction }) {  
 return (  
 <div>  
 <h3>Child Component</h3>  
 <p>Received Count from Parent: {count}</p>  
 <p>Received Message from Parent: {message}</p>  
 <button onClick={callbackFunction}>Click Me</button>  
 </div>  
 );  
}  
  
export default ChildComponent;

We changed the code to pass a **Callback Function** to the Child Component. You may notice that:

*· We don’t have a****useEffect****anymore defined on the Parent Component. This makes the code easier to read, as we can understand our code as, let’s say, more linear and single-threaded than the original.*

*· We don’t have to wait for two render cycles to display our final message, or worse, render both components two times.*

*· We can separate concerns between components, making them more reusable and easier to read or adapt, as we can put whatever we want in the callback function.*

*· Both the state changes at the same time, avoiding the chaining of****useEffect****statements.*

And that’s it for today. Thank you for reading!

**Stackademic**

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

* *Please consider****clapping****and****following****the writer! 👏*
* *Follow us on*[***Twitter(X)***](https://twitter.com/stackademichq)*,*[***LinkedIn***](https://www.linkedin.com/company/stackademic)*, and*[***YouTube***](https://www.youtube.com/c/stackademic)***.***
* *Visit*[***Stackademic.com***](http://stackademic.com/)*to find out more about how we are democratizing free programming education around the world.*

[React](https://medium.com/tag/react?source=post_page-----740660e33420---------------react-----------------)

[React Hook](https://medium.com/tag/react-hook?source=post_page-----740660e33420---------------react_hook-----------------)

[Programming](https://medium.com/tag/programming?source=post_page-----740660e33420---------------programming-----------------)

[JavaScript](https://medium.com/tag/javascript?source=post_page-----740660e33420---------------javascript-----------------)

[Nextjs](https://medium.com/tag/nextjs?source=post_page-----740660e33420---------------nextjs-----------------)