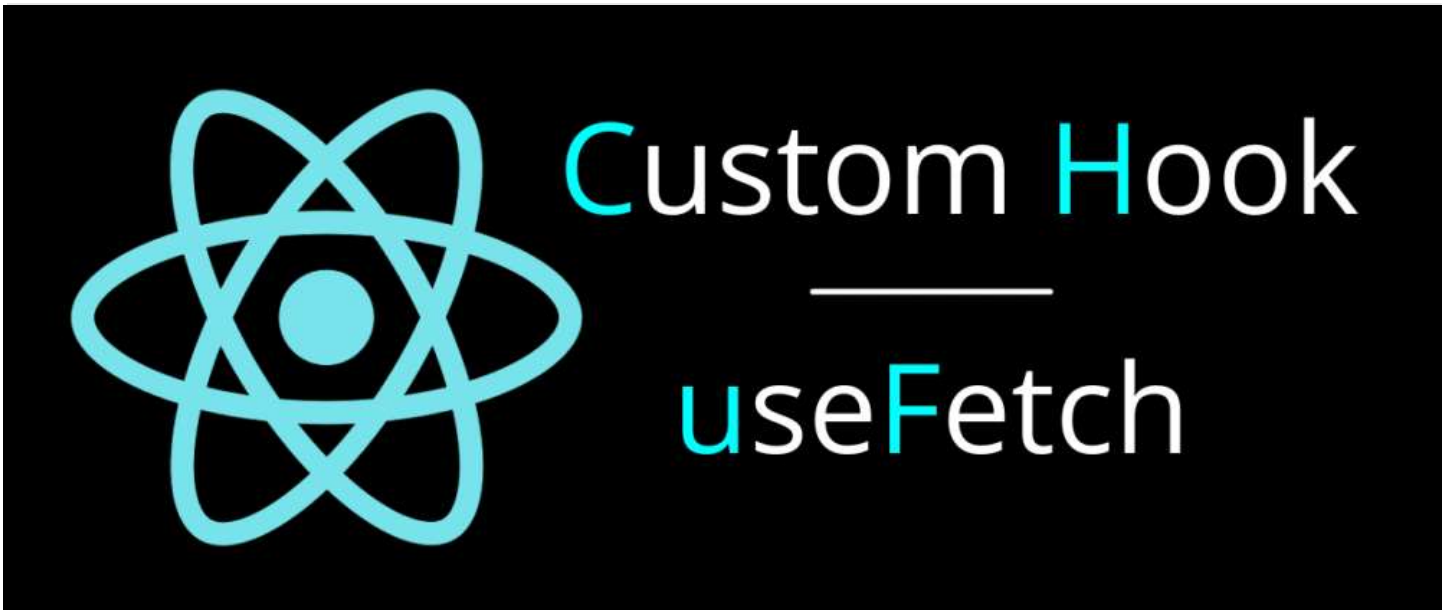


Andrew
Posted on Apr 24

React Custom Hook - useFetch

#javascript #webdev #react #beginners

Why useFetch?

It's very common to fetch data when the user goes to a certain page. We also use common logic when fetching that data.

There's also a fair amount of boilerplate/logic that crowds our components and it's not very DRY (Don't Repeat Yourself).

These are all good reasons to make a custom hook. We can outsource that boilerplate/logic into one separate file. That file will hold the function (hook) which will return for us what we need to use in our components.

The Old Way

In this example, I'll use the `useState` hook to keep track of the loading state, any error, and the data. I'll use the `useEffect` hook to run all of that code. Lastly, I'm using axios to fetch the data, and a cancel token to cancel any unfinished requests that we don't need anymore.

```
//App.js
import { useState, useEffect } from 'react';
import axios from 'axios';
```

```
import axios from 'axios';
import './App.css';

function App() {
  const [quote, setQuote] = useState(null);
  const [loading, setLoading] = useState(null);
  const [error, setError] = useState(null);

  useEffect(() => {
      setLoading('loading...')
      setQuote(null);
      setError(null);
      const source = axios.CancelToken.source();
      axios.get('https://api.quotable.io/random', { cancelToken: source.token })
      .then(res => {
          setLoading(false);
          setQuote(res.data.content);
      })
      .catch(err => {
          setLoading(false)
          setError('An error occurred. Awkward..')
      })
      return () => {
          source.cancel();
      }
  }, [])

  return (
    <div className="App">
      <button onClick={fetchQuote}>Fetch Quote</button>
      { loading && <p>{loading}</p> }
      { quote && <p>"{quote}"</p> }
      { error && <p>{error}</p> }
    </div>
  )
}

export default App;
```

That's a lot of code. Let's move most of it.

The New Way

We'll create another file called `useFetch.js`. You want to start the name of a custom hook with "use" so that React knows to treat it like a hook.

Let's copy over the import statements, all 3 useStates, and the useEffect function.

```
//useFetch.js
import { useState, useEffect } from 'react';
import axios from 'axios';

function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(null);
  const [error, setError] = useState(null);

  useEffect(() => {
      setLoading('loading...')
      setData(null);
      setError(null);
      const source = axios.CancelToken.source();
      axios.get(url, { cancelToken: source.token })
      .then(res => {
          setLoading(false);
          //checking for multiple responses for more flexibility
          //with the url we send in.
          res.data.content && setData(res.data.content);
          res.content && setData(res.content);
      })
      .catch(err => {
          setLoading(false)
          setError('An error occurred. Awkward..')
      })
      return () => {
          source.cancel();
      }
  }, [url])

  return { data, loading, error }

export default useFetch;
```

You may have noticed some changes. First of all, the function (which is our hook), is named useFetch. It receives a parameter which is the url we want to get data from.

We also changed `setQuote` to `setData`, making it more versatile. Notice that we also check for multiple responses to make it more flexible as well.

Lastly, our useFetch function (hook) returns our data, loading, and any error.

> I put those in an object so we can use *object* destructuring when accessing those in our component. That way, the order doesn't matter when we destructure them, and we can rename them if we want. I'll show you that next.

Using useFetch in Our Component

So, back in our App component, we'll import our useFetch hook from `useFetch.js`, and pass in the url we want to fetch data from. We'll use object destructuring to access what we need. Lastly, we'll rename data to quote.

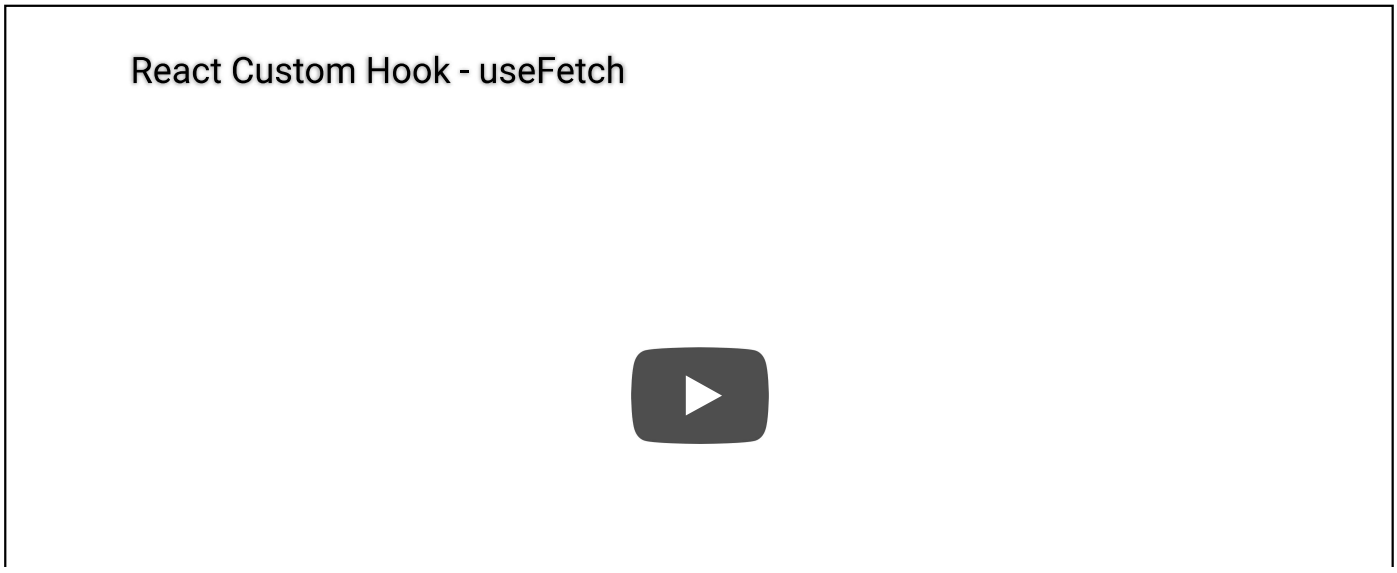
```
import useFetch from './useFetch';
import './App.css';

function App() {
  const { data: quote, loading, error } = useFetch('https://api.quotable.io/random')

  return (
    <div className="App">
      { loading && <p>{loading}</p> }
      { quote && <p>"{quote}"</p> }
      { error && <p>{error}</p> }
    </div>
  );
}

export default App;
```

Muuuuuch cleaner 😎.

Conclusion

Custom hooks are very useful for cleaning up your code. You can use React hooks inside of your custom hooks (they're all functions after all! 👍). You can encapsulate a lot of repetitive logic, then return what you need from the custom hook.

I have a YouTube video if you want to see it in action.



If you like learning about similar topics, feel free to check out my YouTube or Instagram.

Hope this helped somebody and thanks for reading!

-Andrew

Discussion (9)

dastasoft • Apr 26 • Edited on Apr 26

Very good article, it's good not to repeat and also isolate the logic of the component.

One tip, for me mixing the load state between a string and a boolean is a bit confusing, I know JS allows you to do this and a non-empty string is truthy but personally I think it's better to keep the same data type throughout the application, especially when you are using it only as a flag not to display content.

The different states you have there are closely related so it might be a good idea to use a useReducer, check this article by Kent C. Dodds I bet it will be useful to improve the hook.

Andrew • Apr 26

Agreed. I used useReducer on my previous article (dev.to/techcheck/react-hooks-usere...). Good call on Kent's article 👌.

Fidel Castro • Jun 3

this is great thank you!

Andrew • Jun 4

Thanks!

Chris Frewin • Apr 26

Ah, this is exactly what I've been looking for - I'd love to extend it with configurable options like get vs post, and other things like headers for auth for example. Great!

Andrew • Apr 26

For sure! And thanks! 🙏

Dave • Nov 21

Good write up on the hook. I was looking to create a useFetch hook and came to this, though my intention was completely different.

I was looking to create an useFetch hook that returns a fetch function that handles all the error (displaying them in snackbar), so you will end up handling all the error in one place.

For the same functionality in your write up, with additional feature to memorize and caching the results, may I also recommend useSWR. It's created by the same people behind NextJS and vercel and it's really good.

Karunamay Murmu • Jul 14

How can we call the custom hook on button click event?

alextrastero • Aug 31

You would need a new hook in that case, one that returns the actual function instead of calling it inside `useEffect`, something like:

```
function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(null);
  const [error, setError] = useState(null);

  const myFetch = useCallback(() => {
    setLoading('loading...')
    setData(null);
    setError(null);
    axios.get(url).then(res => {
      setLoading(false);
      // checking for multiple responses for more flexibility
      // with the url we send in.
      res.data.content && setData(res.data.content);
```

```
        res.content && setData(res.content);
      })

      .catch(err => {
        setLoading(false)
        setError('An error occurred. Awkward..')
      })
  }, [url]);

  return { myFetch, data, loading, error }
}
```

Code of Conduct • Report abuse

Andrew

Aspiring web developer. Lover of Javascript and React 👍

LOCATION
Cincinnati, OH

JOINED
Mar 9, 2021

More from Andrew

React Hooks - useReducer
#javascript #webdev #react #beginners

React Hooks - useContext
#javascript #webdev #beginners #react

React Hooks - useEffect
#javascript #react #webdev #beginners