Simplifying React Hooks

Meet Zaveri | Follow
Dec 4 · 6 min read

focuses on folks who are coming from class components

Typical overview of what we are going to do in this post

In this article, I tried my best to make it easier for you to understand what are react hooks. I have written this for guys coming from traditional class component pattern who want to explore react hooks.

We'll cover how to use `useState` , `useEffect()` and more techniques to simulate the experience you were having in class component era

✨ Hooks and Class Components

Hooks don't work inside classes and that gives us benefit of using them in functional component.

But what about **lifecycles and state** which we were leveraging in class components? Well, there is a way for replicating them in form of hooks.

> Hooks are functions that let you "hook into" React state and lifecycle features from function components. Hooks don't work inside classes, they let you use React without classes — React.js Official Documentation

💡 **Let's talk about managing state: useState**

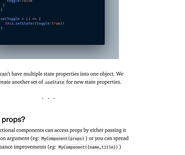
`useState` is a hook which helps us manage state.

```
const [toggle,setToggle] = useState(false)
```

- A state property which holds the current value and can keep track of
- A function which sets state property. In this case it updates "toggle" property
- Initial value for state property

You can get a glimpse from this above image as what `useState` hook offers. So when you call `setToggle(true)` then `toggle` value changes from `false` to `true`

If we were to replicate this into **class component** then, it would look like this



Unlike class, we can't have multiple state properties into one object. We have to always create another set of `useState` for new state properties.

. . .

How to use props?

As we know, functional components can access props by either passing it directly in function argument (eg: `MyComponent(props)` or you can spread them for performance improvements (eg: `MyComponent({name,title})`)

So there's no explicit API or section on how should we use props in react hooks.

🔗 APIs, where we should call them ?!

Old Class-Ical way

Usually we would write network requests(APIs) in '`componentDidMount`' lifecycle which is the preferable one as per react documentation suggests.

```
componentDidMount(){
this.props.getData()
}
```

But for hooks we have a similar way to do network requests in particular fashion.

useEffect())

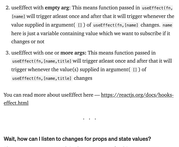
As documentation says, it's a *hook that lets you perform side effects.*

> Remember that any useEffect( ) hook will be called atleast once at time of render phase regardless of its arguments

We can call our APIs in `useEffect()` as following:

```
import React, { useEffect } from 'react';
import ProfilePage from "./ProfilePage";

const ProfileContainer = () => {

  useEffect(() => {
    fetch('/api/user') // perform any side effects or network requests
  }, []); // => empty array means 'run once'

  return <ProfilePage />;
};

export default ProfileContainer;
```

example_useEffect_empty_arr.js hosted with ❤ by GitHub | view raw



So as you can see if we pass empty array `[]` in second argument of the function and it will be executed only once when that component **completes its first render**. This is because there're no values in array(i.e. second arg) to keep track of whether those values change or not.

Take a look at this scenario

1. useEffect with **no arge**: This means function passed in `useEffect()`will always trigger when there's re-render of the component
2. useEffect with **empty arg**: This means function passed in `useEffect(fn, [])`will trigger atlease once and after that it will trigger whenever the value supplied in argument(`[]`) of `useEffect(fn,[])`changes. same here is just a variable containing value which we want to subscribe if it changes or not
3. useEffect with one or **more args**: This means function passed in `useEffect(fn,[name,title])` will trigger atleast once and after that it will trigger whenever the value(s) supplied in argument(`[]`) of `useEffect(fn,[name,title]` changes

You can read more about useEffect here — https://reactjs.org/docs/hooks-effect.html

. . .

Wait, how can I listen to changes for props and state values?

The most common approach in class component for this use case was achieved using `componentDidUpdate`

```
componentDidUpdate(prevProps,prevState){
  if(prevProps.data !== this.props.data){
  // do some action
}
}
```

Discuss how `useEffect()` hook can be used for it

This can be achieved in hooks by writing your logic in function of `useEffect` and by passing your value on which you want to check if that changed in dependency array `[]`

Passing our logic into function and list of dependency(s) into dependency array will conclude that whenever any of the dependency(s)'s value changes, then the function will be re-triggered hence our logic will be applied as per old method in class component

```
import React, { useEffect } from 'react';

const Container = () => {
  const searchVal = prop.searchVal;
  const makeAPICall = () => {
    // do your ajaxy activities, side effects or network effects
  }
  useEffect(makeAPICall,[searchVal])
  return Input(onKeyDown:handle);
};
export default Container;
```

comparision_example.js hosted with ❤ by GitHub | view raw

> **Again, *take note that this useEffect() will always be executed atleast once as I said earlier. This means that if your `prop.searchVal` hasn't changed then also your hook will be triggered.***

To overcome this, you could add extra condition in function which differentiates initial state(value) of `prop.searchVal`

. . .

When can I know that my component is going to unmount?

This can also be achieved via passing `[]` empty arge in useEffect with main function returning another function where we can have unmount logic in it

You will get clear understanding when you'll try it with

```
import React, { useEffect } from 'react';

const ProfileContainer = () => {

  // generally I prefer this as useEffect with empty arg[]
  useEffect(() => {
    fetch('/api/user') // perform any side effects or network requests

    // if you can have return() for this function which will be executed when component will
    return () => {
      console.log("Unmounting ...");
      // cancelRequestPromise();
    }
  }, []); // => empty array means 'run once'

  return <div>Hello World !</div>;
};

export default ProfileContainer;
```

example_unmount_hook_js hosted with ❤ by GitHub | view raw

Note there are two things here. 1) the empty braces 2) return function to contain unmounting logic

This means this useEffect snippet will act like:

1. initial logic in function gets executed (as in component just mounted)
2. the logic inside return function gets executed when component is unmounted

How can I wrap them ?

Same way as we did with class components. Yeah literally, we wrap as `activeWrapper(Dashboard)` .

📖 Conclusion

So folks, these were essential barebones for getting up and running with hooks.

Please report or suggest me if I have included any misinformation, incorrect logic or lack of proof.

I will do another post on covering other hooks considering the amount of response I get here.

Thanks, keep grinding!

React | Hooks | React Hooks | Web Development

31 claps | Twitter | Facebook | LinkedIn | More

WRITTEN BY
Meet Zaveri | Follow
Promoted Engineer with a keen interest in Computer Science.
Passionate about Open Source, JavaScript and its ecosystem.

Medium | **The Startup** | Follow
Medium's largest active publication, followed by +755K people.
Follow to join our community.

More From Medium

More from The Startup
Is M1 Mac Worthy or Good for Developers? [Developer Review]
Stefan Samcovic in The... · Nov 23 · 6 min read · 1.3K

More from The Startup
My Software Engineer Roadmap
Joana Carrasco in The Startup · Nov 21 · 7 min read · 2.4K

More from The Startup
The Email That Got Me Promoted to CTO
James Williams in The Startup · Nov 21 · 8 min read · 3.6K

Learn more.
Medium is an open platform where 170 million readers come to find insightful and dynamic thinking. Here, ideas and undiscovered voices alike dive into the heart of any topic and bring new ideas to the surface. Learn more

Make Medium yours.
Follow writers, publications, and topics that matter to you, and we'll deliver them to your homepage and in your inbox. Explore

Share your thinking.
If you have a story to tell, knowledge to share, or a perspective to offer — welcome to Medium. It's easy and free to post your thinking on any topic. Write on Medium

Medium | About | Help | Legal