React Hooks for Greater Good

Written by Seth Corker on Benevolent Bytes

A look at the React Hooks feature proposal It's an exciting time to be a React developer. React is coming out with some great features that will hopefully allow for a better developer experience and ultimately a better experience for end users. Features like Suspense are making easier to craft React apps with features users have come to expect. Currently in the proposal stage for React 16.7.0 is Hooks. On the surface, it looks like a more stylistic change but in the same way that, you *can* write React without using jsx, but it's more enjoyable and easier to understand once you get the hang of it. So what are Hooks and why should you take note?

What's a Hook? A Hook is just a way to...

> Use state and other React features without writing a class

That's it. Although it may not seem like a big deal, it will mean that it's easier than ever to break components into more reusable parts. This allows for behaviours to be compartmentalized to avoid massive components and to adhere more closely to SRP. The easiest way to understand what hooks are and how we might use them is to jump into some code, so let's take a look at how we might use a simple hook provided with React.

How can I use a hook? Let's start with the `useState` hook, it does what it says on the tin and allows state to be used within a functional component. Here is a simple component which is designed to have a simple toggle between on and off.

View the code on GitHub

If you're used to creating a class to encapsulate some trivial state, you'll notice how concise using the `useState` hook is.

We start by setting our single value state:

```
const [isOn, toggleOn] = useState(false);
```

The first item in the array is the state itself and the second is the function we use to set the state. Hopefully, this will make it easier to make more reusable state. Although, this example is just to illustrate how the `useState` hook can be used in a component.

A post about hooks on the official React site goes into the motivation behind the proposal. One of the symptoms to some problems in React are as follows:

> If you look at a typical React application in React DevTools, you will likely find a "wrapper hell" of components surrounded by layers of providers, consumers, higher-order components, render props, and other abstractions.

It can become a nightmare when you want to promote reusability within a React project, even most popular libraries like React Router have powerful and easy to use HoC's but often result in mandatory unwrapping components when debugging. It can also add a layer of complexity or require some major refactoring in order to share stateful logic between components. This leads to the key benefit:

> Hooks allow you to reuse stateful logic without changing your component hierarchy.

It's clear that this is a great step in a brighter future for React. I think this promotes greater opportunity for code re-usability within the community and will make components easier to understand. You can now get the job done more concisely.

An Example I made a small sample site using some components which only use the `useState` hook. If you like the look of the example, take a look at Marx by Matthew Blode, which is used for the base stylesheet.

Take a look at the basic form element from the example project. It uses an object to store the form data which we can then submit to receive the output.

```
import React, { useState } from "react"

function byInitialVal(a, c) {
  a[c.props.id] = c.props.defaultValue || ""
  return a
}
```

```
export default function Form({ name, children, onSubmit }) {
  const childArray = React.Children.toArray(children)
  const initialState = childArray.reduce(byInitialVal, {})
  const [values, setValue] = useState(initialState)

  function createFormField(child) {
    return (
      <div>
        <label htmlFor={child.props.id}>
          <span style={{ textTransform: "capitalize" }}>{child.props.id}</span>
          <child.type
            type={child.props.type}
            id={child.props.id}
            value={values[child.props.id]}
            onChange={({ target }) =>
              setValue({
                ...values,
                [child.props.id]: target.value,
              })
            }
          />
        </label>
      </div>
    )
  }

  return (
    <form
      onSubmit={e => {
        e.preventDefault()
        onSubmit(values)
      }}
      onReset={() => setValue(initialState)}
    >
      {React.Children.map(children, createFormField)}
      <hr />
      <input type="reset" value="Reset" />
      <input type="submit" value="Submit" />
    </form>
  )
}
```

Using the form is quite easy and flexible, it's not ideal but it could suffice for a simple form which is easy to understand.

A snippet of how to use the form (View the code on GitHub)

Take a peek at the source code for my react-hooks-example to get a better understanding of how you might use hooks.

When can I use hooks? This feature is under proposal so the API might still have a few subtle iterations to go before it's in a release but it can't hurt to start playing around with it. Try it in **React v16.7.0-alpha** and see what you think. I'm excited to see how some popular React libraries adapt and evolve to use hooks in the future.

I hope you gained something from this quick run-through of the React hook feature proposal. Let me know if I missed anything or if there are any thoughts you have on hooks and the future of React.

Source: Github Repo

Demo: https://react-hooks-example.sethcorker.com/