The useRef Hook

(https://github.com/learn-co-curriculum/react-hooks-use-ref) (https://github.com/learn-co-curriculum/react-hooks-use-ref/issues/new)

Learning Goals

- Understand common use cases for the `useRef` hook
- Use the `useRef` hook to access DOM elements
- Use the `useRef` hook to persist data across multiple component renders

Introduction

In this lesson, we'll explore how to use the `useRef` hook and some common use cases for it. You can find starter code with the examples we'll discuss in the `src/components` directory. Run `npm install && npm start` to run the example code and code along!

useRef and Ref Variables

The `useRef` hook gives us a way to capture a **reference** to values that are accessible across multiple renders of our component. In some ways, it's similar to the `useState` hook: the `useState` hook also lets us keep track of values across multiple renders of our component, like this:

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

function CounterState() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount((count) => count + 1);
  }

  return (
    <div>
```

```
      <h1>CounterState</h1>
      <button onClick={handleClick}>{count}</button>
    </div>
  );
}
```

In this example, we use the `useState` hook to create some new internal state within React that we can access using the `count` variable every time our component re-renders. Also, **calling `setCount` will trigger a re-render**.

Using a ref instead, our component would look like this:

```
import React, { useRef } from "react";

function CounterRef() {
  const count = useRef(0);

  function handleClick() {
    count.current = count.current + 1;
    console.log(count.current);
  }

  return (
    <div>
      <h1>CounterRef</h1>
      <button onClick={handleClick}>{count.current}</button>
    </div>
  );
}
```

To break down the code:

- We must first import the `useRef` hook, just like with the other hooks we've seen
- We call `useRef` and pass in an initial value

- Calling `useRef` creates a new internal value in React and gives us access to that value in a **ref variable**, which is an **object** with just one key: `current`. It looks like this: `{ current: 0 }`
- To update the value of the ref in React's internals, we update its `current` property: `count.current = count.current + 1`

The key difference between these approaches is that in the `useRef` example, updating the ref variable **does not cause our component to re-render**. Try out both buttons in the browser to see the difference.

`useRef` still allows us to have a variable that persists between renders of our component, but since updating its value doesn't trigger a re-render, we use it in different situations than when we'd use `useState`. You can think of this ref variable almost like an **instance variable** for your function components.

Let's see some good use cases for the `useRef` hook.

Persisting Values Across Renders

Let's build out a price tracking component. The features of this component are:

- Every 1 second, generate a new random price
- If the old price is less than the new price, use a green font color to indicate a rise in price
- If the old price is greater than the new price, use a red font color to indicate a drop in price

Here's some starter code that implements the first feature of generating a random price each second:

```
import React, { useEffect, useState } from "react";
import { makeRandomNumber } from "../utils";

function Ticker() {
  const [price, setPrice] = useState(0);
  const [color, setColor] = useState("black");

  useEffect(() => {
    // every 1 second, generate a new random price
    const id = setInterval(() => setPrice(makeRandomNumber), 1000);
```

```
      return function () {
        clearInterval(id);
      };
    }, []);

    return (
      <div>
        <h1>TickerMaster</h1>
        <h2 style={{ color: color }}>Price: ${price}</h2>
      </div>
    );
  }
```

What we want is a way to set the color based on the change in price between the previous render and the current render. Since we want to change the color based on the price, we can start off by writing out a side effect with the price as the dependency:

```
useEffect(() => {
  // we need some way to get the prevPrice...
  if (price > prevPrice) {
    setColor("green");
  } else if (price < prevPrice) {
    setColor("red");
  } else {
    setColor("black");
  }
}, [price]);
```

To make this work, we need to persist the previous price. This is where we can use the `useRef` hook! Since our goal is to:

- Access the same data across renders
- Not re-render the component when saving this data

`useRef` is a good tool for the job of storing the previous price. Here's how we'd use it:

```
import React, { useEffect, useRef, useState } from "react";
import { makeRandomNumber } from "../utils";

function Ticker() {
  const [price, setPrice] = useState(0);
  const [color, setColor] = useState("black");
  // create the ref and set its initial value
  const prevPriceRef = useRef(price);

  useEffect(() => {
    // use the current value of the ref
    const prevPrice = prevPriceRef.current;
    if (price > prevPrice) {
      setColor("green");
    } else if (price < prevPrice) {
      setColor("red");
    } else {
      setColor("black");
    }
    // set the new value of the ref (note: this doesn't trigger a re-render)
    prevPriceRef.current = price;
  }, [price]);

  useEffect(() => {
    const id = setInterval(() => setPrice(makeRandomNumber), 1000);
    return function cleanup() {
      clearInterval(id);
    };
  }, []);

  return (
    <div>
```

```
      <h1>TickerMaster</h1>
      <h2 style={{ color: color }}>Price: ${price}</h2>
    </div>
  );
}
```

Try using this component in the browser. Explore the component's render cycle by adding some console messages and see how the values in state and in the ref change over time.

Accessing DOM Elements

Another common use case for the `useRef` hook is to gain access to the actual DOM elements being created by our React components. In general, we want to give React control over the DOM based on the JSX that is returned by our components. However, sometimes it is also useful to gain access to the actual DOM elements for a few uses outside of the React rendering cycle, such as:

- using a third-party library that needs access to a DOM element
- accessing input values in a non-controlled form
- setting focus on an element
- measuring the size of a DOM element
- working with a `<canvas>` or `<video>` element

To use a ref on a DOM element, we first create the ref using the `useRef` hook, just like before:

```
function Box() {
  const elementRef = useRef();

  return (
    <div>
      <h1>Box</h1>
      <button>Measure</button>
    </div>
  );
}
```

Then, we can attach the ref to a DOM element by adding a special `ref` attribute to our JSX element:

```
function Box() {
  const elementRef = useRef();

  return (
    <div ref={elementRef}>
      <h1>Box</h1>
      <button>Measure</button>
    </div>
  );
}
```

Now, we can access information about that DOM element in our component:

```
function Box() {
  const elementRef = useRef();

  function handleMeasureClick() {
    const div = elementRef.current;
    console.log("Measurements: ", div.getBoundingClientRect());
  }

  return (
    <div ref={elementRef}>
      <h1>Box</h1>
      <button onClick={handleMeasureClick}>Measure</button>
    </div>
    );
  }
```

Another example of using the `useRef` hook to access a DOM element is in the `TickerChart` component included in the `src/components` directory. In that example, we're using a `<canvas>` element to draw out a graph of the price changes over time.

Conclusion

Like `useState` , the `useRef` hook gives us some internal React values that will persist across renders of our component. Unlike **state**, when we update a **ref**, React will not automatically re-render our component. This makes refs useful for keeping track of persistent data in our components, similar to an instance variable.

A **ref variable** can also be used to gain access to DOM elements using the `ref` prop on a JSX element.

Resources

- [useRef (https://reactjs.org/docs/hooks-reference.html#useref)](https://reactjs.org/docs/hooks-reference.html#useref)