Resize Observer API with React

Written by Seth Corker on Benevolent Bytes

There are many APIs that make the web a powerful platform, so why not take advantage of some of them and learn how you can integrate them with React.

The promise of the web is to make information accessible to everyone and part of this goal means supporting different devices in an elegant way. You may be familiar with media queries, a great way to adapt the layout based on the screen size or device properties. This a great tool but it doesn't give us any flexibility when working with React components. The Resize Observer API fits this purpose much better.

What is it?

The Resize Observer API allows you to detect when a particular element changes size and how large it is. The advantage of this API over media queries is the ability to watch to particular elements rather than the entire screen.

Want to see it in action? Take a look at the demo.

Why is it useful?

Making responsive components is challenging because you can only do so much with media queries. Instead, we could define the breakpoints we want our component to use and adjust the layout accordingly. The same component in different places on the page can respond to size changes based on the parent and not the whole viewport.

How do I use it?

Let's take a look at the scenario I described above. To achieve this, a custom hook will provide an easy to use API we can reuse.

The Hook

I created the `useResponsiveBreakpoints` hook which accepts the ref of the element we want to observe size changes on and a list of named breakpoints. In this example, I only care about a single element but it's perfectly reasonable to watch multiple elements if you need to.

```
export default function useResponsiveBreakpoints(elRef, breakpoints) {
  const firstQuery = Object.keys(breakpoints[0])[0]
  const [breakSize, setBreakSize] = React.useState(firstQuery)

  const observer = React.useRef(
    new ResizeObserver(entries => {
      // Only care about the first element, we expect one element ot be watched
      const { width } = entries[0].contentRect

      setBreakSize(findBreakPoint(breakpoints, width))
    })
  )

  React.useEffect(() => {
    if (elRef.current) {
      observer.current.observe(elRef.current)
    }

    return () => {
      observer.current.unobserve()
    }
  }, [elRef, observer])

  return breakSize
}
```

Using the hook

The hook can be used like so, the first argument is our element. The second is a list of breakpoints. Each breakpoint contains a single key so we can use the names however we want, and a value. When the element has a width of 350, the breakpoint will be `medium` when it's 590 it will be `large` and so on.

```
const size = useResponsiveBreakpoints(targetRef, [
  { small: 200 },
  { medium: 400 },
  { large: 600 },
])
```

This is just an example hook for a problem I've faced before when trying to create more reusable and responsive components. If you've faced this issue in the past then give the Resize Observer a go, it has reasonable support across browsers except for IE, Edge and Safari. For those browsers, you might need to look for a more rudimentary solution.

Resources

- Resize Observer API on MDN
- Want the source for the demo? Take a look at the repository on GitHub.