A Drip of JavaScript

Measuring JavaScript Performance with console.time

Originally published in the A Drip of JavaScript newsletter.

At one point or another almost every JavaScript developer has to deal with optimizing performance-critical code. But how exactly do you go about it? This drip will show you how to use `console.time` as a low-cost performance testing tool.

First, let's suppose that we have a scientific application that makes frequent use of the factorial function.

If you're not familiar with the factorial in mathematics, it basically works like this: Any input to the factorial function must be a non-negative integer. The factorial of X is the product of every positive integer less than or equal to X. In other words, the factorial of 3 is 1 * 2 * 3, the factorial of 4 is 1 * 2 * 3 * 4, and so on.

Here we have a simple implementation of `factorial`.

```
function factorial (num) {
    if (num < 0) {
        throw new Error("Number cannot be negative.");
    }

    if (num % 1 !== 0) {
        throw new Error("Number must be an integer.");
    }

    // The base case
```

```
    if (num === 0 || num === 1) {
        return 1;
    }

    // The general case
    return num * factorial(num - 1);
}

console.log(factorial(4));
// => 24
```

In order to optimize `factorial`, we need to measure its current performance so that we have something to compare against.

```
console.time("factorial test");

for (var i = 1; i < 100000; i++) {
    factorial(20);
}

console.timeEnd("factorial test");
// => 512.46ms
```

The `console.time` method starts a timer with the name that you give it. In this case we called our timer `"factorial test"`. Correspondingly, when the `console.timeEnd` method is called with the same name, it will halt the timer and log how many milliseconds have elapsed.

You'll note that in order to get roughly accurate performance measurements, I reran the factorial calculation 100,000 times and it still only took about half a second. Modern JavaScript engines running on modern hardware are **fast**.

Now let's try optimizing `factorial`.

```
function factorial (num) {
    if (num < 0) {
```

```
        throw new Error("Number cannot be negative.");
    }

    if (num % 1 !== 0) {
        throw new Error("Number must be an integer.");
    }

    function factorialEquation (num) {
        // The base case
        if (num === 0 || num === 1) {
            return 1;
        }

        // The general case
        return num * factorialEquation(num - 1);
    }

    return factorialEquation(num);
}

console.time("factorial test");

for (var i = 1; i < 100000; i++) {
    factorial(20);
}

console.timeEnd("factorial test");
// => 81.39ms
```

The optimization in this case is quite simple. Rather than checking `factorial`'s inputs every time the equation is recursively called, we have separated out the equation logic from the input checking logic and ensured that the input checks are only run once. The performance improvements are pretty substantial. Our test case goes from half a second to less than one tenth of a second.

Keep in mind that the performance characteristics of a given piece of code vary by JavaScript engine, so for real performance tests you should check in each browser

you support.

It is also important to note that the vast majority of JavaScript code doesn't need to be heavily optimized. In the general case, readability and maintainability should trump optimization. Generally you will only need to optimize JavaScript that is part of your program's critical path, and then only if it has been found to be too slow for your use case.

The `console.time` and `console.timeEnd` methods are available in all modern browsers, including IE 11.

Now you have one more tool in your belt to help you put together excellent JavaScript applications.

Thanks for reading!

Josh Clanton
