Lessons on setTimeout() and clearTimeout()

Posted At : April 23, 2010 12:28 PM | Posted By : Jon Hartmann
Related Categories: Javascript

My world was rocked the other day while looking through some source code for a Javascript widget at work. Normally, I'm the one giving lessons in JS techniques, but I was absolutely floored to see how my coworker had used setTimeout() and clearTimeout() in ways I'd never seen before.

A Simple Example

Ok, so if you've done more than a little bit of JS development, you probably have made use of Javascript's setTimeout() method. You may not have made use of clearTimeout() — I didn't know it existed until I saw it in the code — but its purpose should be clear enough from the name, and I'll use it in an example later. Let's create a simple example of using setTimeout():

```
<div id="clock"></div>
<InvalidTag type="text/javascript">
    var clockDisplay = document.getElementById('clock');

    function updateClock() {
        var time = new Date();
        clockDisplay.innerHTML = time.getHours() + ':' + time.getMinutes() + ':' + time.getSeconds();

        setTimeout('updateClock()', 1000);
    }

    updateClock();
</script>
```

Ok, so that is a simple little clock that ticks along as you go... not the best constructed Javascript, but it shows what you can do with setTimeout().

It takes a Function

What I'd never realized was that setTimeout() accepted a function name instead of a string for execution, and that if you passed a function name, it respects encapsulation. That means you can get away with some better formatted code like this:

```
<div id="clock"></div>
<InvalidTag type="text/javascript">
    (function() {
        var clockDisplay = document.getElementById('clock');

        function updateClock() {
            var time = new Date();
            clockDisplay.innerHTML = time.getHours() + ':' + time.getMinutes() + ':' + time.getSeconds();

            setTimeout(updateClock, 1000);
        }

        updateClock();
    })();
</script>
```

This is much nicer! The code has been nicely wrapped up so that it executes on load and keeps things isolated in its own tiny encapsulation.

What Really Blew My Mind

What really blew me away is that setTimeout() returns a value. What?

```
clock = setTimeout(updateClock, 1000);
```

Is valid, and not only is it valid, the clock variable is useful! What for? For clearTimeout() of course! By passing the value returned from setTimeout() to clearTimeout() you can stop an existing timeout from executing. Now, let's look at a slightly expanded example that shows this better:

```
<div id="clock"></div>
<button id="start">Start</button>
<button id="stop">Stop</button>
<InvalidTag type="text/javascript">
    (function() {
        var clockDisplay = document.getElementById('clock');
        var clock;

        function updateClock() {
            var time = new Date();
            clockDisplay.innerHTML = time.getHours() + ':' + time.getMinutes() + ':' + time.getSeconds();

            clock = setTimeout(updateClock, 1000);
        }

        updateClock();

    document.getElementById('stop').onclick = function() {
        clearTimeout(clock);
        return false;
        };

        document.getElementById('start').onclick = function() {
            updateClock();
            return false;
        };
    })();
</script>
```

Here we can see the clock expanded with a button that stops the clock from executing. Along with it, I added a start button so you can test starting/stopping it easily. Our setTimeout() returns a variable that we then store, and the stop click calls clearTimeout with that value to stop it from executing.

What Does This Mean?

This means you can completely leave string based setTimeout() calls behind, and gain a much finer control on the execution of timeout triggers. This probably isn't news to some people, but given that I'd somehow managed to miss this trick in the last 3 years of JS development, I thought I'd blog about it.

If one where really implementing a "clock" type thing, it would be more proper to use setInterval() and clearInterval(). These function just the same as their timeout counterparts, but setInterval() continues to execute until cleared, rather than just once. For example, while I had to set a new timeout ever execution, you could have just set up the interval once with setInterval(). The only down side to this is that if you clear it you have to call setInterval() again.