JavaScript

Simple JavaScript Debugging

Page 2 — The Agony of Debug

Let's talk about some debugging techniques that will work in every browser.

We're going to rely on the most basic line of communication: the alert() function. You may want to take a look at this logging mechanism, though it has a couple of drawbacks: you have to add a to every page you want to debug, and it may not work on older browsers.

There is one particular problem with using alert() for debugging purposes, and that is that you get one alert box per message. If you use it in a loop, you'll have to hit the Return key repeatedly to dismiss the messages, and it can get to the point where you have to force quit your browser because it's taking too long to get through all the alert boxes.

I've created a simple log() function that avoids this problem by lumping all your messages together and displaying them together every second. That may sound pretty fast, but most JavaScript loops take less than a second to execute. Take a look at the source code, (Right-click and choose "Save As..." to download the script).

So let's start with a worst-case scenario. You click a button that's supposed to do something, but there's no reaction at all from the browser and your JavaScript console isn't showing any errors. What's going on?

The best way to find out is to narrate your code with log() calls. Here's an example of some code that's supposed to count to a certain number:

```
countTo(7);

function countTo (number)
{
  var target = document.getElementById('counter');

  log('starting countTo');

  if (number > 1)
  {
    log('number is too small to count to');
    return;
  }

  for (var i = 1; i <= number; i++)
  {
    log('count is ' + i);
    target.innerHtml += i + '... ';
  }
}
```

I always put in a log() call right at the top of the function, just to make sure it's getting invoked in the first place. When this version of the code runs, we get this trace:

```
starting countTo
number is too small to count to
```

Well, that's obviously wrong. You probably already spotted it — we need to test (number < 1), not (number > 1).

After changing that, it looks like the function executes properly, but we still don't see any output. We already keep track of the count variable, but how do we know our target element is really getting updated?

Enter the inspect() function. It's a tiny function like log() that tells you about the properties that the objects you're working with possess. I've bundled it with the source code for log(). There's also another function in there called inspectValues() that tells you what each property is set to, but its output tends to be pretty large — definitely not something you want to put inside a loop.

Let's add an inspect call to see what's really going on:

```
for (var i = 1; i <= number; i++)
{
  log('count is ' + i);
  target.innerHtml += i + '... ';
  inspect(target);
}
```

We get this output:

```
Object possesses these properties:
innerHtml, nodeName, nodeType, parentNode, childNodes,
previousSibling, nextSibling, attributes, ownerDocument, localName,
tagName, id, title, lang, className, align, offsetTop, offsetLeft,
offsetWidth, offsetHeight, offsetParent, innerHTML...
```

Wait, we've got both an innerHtml property and an innerHTML property. That can't be good. That's where our problem is — because JavaScript is case-sensitive, we're writing to the wrong property. If we change that, everything works.

Now, this was a pretty elementary example. The same principles apply in real life, though. The key is getting as much information about what's going on as possible. These log(), inspect(), and inspectValues() functions are good ways to find this out.

Next time, we'll look at using at using Venkman, a heavy-duty JavaScript debugger. Until then, happy coding.

Did you love this article? Did you hate it? Think you can do better? Send us your Feedback. Feedback submitted here will be considered for publication on Webmonkey or Wired News, so if you don't want us to print your comments, please say so in your email.

[an error occurred while processing this directive] [an error occurred while processing this directive]