

Writing Efficient JavaScript

By

Gopalarathnam Venkatesan
Yahoo!, Inc.

“We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.”

- Donald Knuth, paraphrasing C.A.R Hoare

Agenda

Section 1. Core JavaScript Language

Section 2. JavaScript and DOM

Section 1: Core JavaScript Language

- Not an introduction to JavaScript
- Common programming practices to be avoided when writing JavaScript programs
- Guidelines for writing efficient JavaScript programs covering the core language

Primitive Data Types and Implicit Conversion

- Corresponding to numbers, strings, boolean, JavaScript has some primitive data types – Number, String, Boolean
- Every time a method is accessed on a primitive type (say a string), it is implicitly converted to its primitive object (String in this case), and the method of the object is run for assigning the result back to the variable of that type.

Primitive Data Types and Implicit Conversion (contd.)

- Consider:

```
var s = "hello, world";  
for (var i = 0; i < s.length; i++) {  
    alert(s.charAt(i));  
}
```

- Here the string “s” is converted to a String object twice every run of the loop, for accessing the “length” and “charAt” properties.

Primitive Data Types and Implicit Conversion (contd.)

- Better way to write:

```
var s = new String("hello, world"), i, n;  
for (i = 0, n = s.length; i < n; i++) {  
    alert(s.charAt(i));  
}
```

- Here, the object is created, and hence for accessing the “length”, and “charAt” properties, there was no implicit conversion needed. The object is anyway freed once it goes out of scope.

Accessing properties and methods (contd.)

- Consider:

```
var p = {"greet":"hello"}, s = "greet", t;  
eval("t = p." + s)
```

- Better way to write:

```
var p = {"greet":"hello"}, s = "greet", t;  
t = p[s];
```

- A code that does not use eval, performs 85-95% faster than the one that uses it. So, avoid it wherever possible.

Cache function pointers

- Consider:

```
function doMyDuty(collection)
{
    var i, n = collection.length;
    for (i = 0; i < n; i++)
        doStuff(collection[i]);
}
```

- Here doStuff is a function outside the scope of doMyDuty()

Cache function pointers (contd.)

- The global lookup for every iteration can be avoided by rewriting the doMyDuty as:

```
function doMyDuty(collection)  
{  
    var i, n = collection.length,  
        fcn = doStuff;  
    for (i = 0; i < n; i++)  
        fcn(collection[i]);  
}
```

Functions and callbacks

- Consider the scenario of passing a callback function to a function:

```
var callback = new Function("// ... ");  
foo(somedata, callback);
```
- Using the “Function” constructor is equivalent to using “eval”, since the string inside the constructor needs to be evaluated to be function code.

Functions and callbacks (contd.)

- The better and faster alternative:

```
var callback = function () { // ... };  
foo(somedata, callback);
```

- If you don't need to re-use the callback function (name), write the function definition directly. In JavaScript, a function can be used wherever expressions can be used:

```
foo(somedata, function () { /// ... });
```

Avoid the use of “with”

- Consider:

```
with (elem.style) {  
    color = "#fff";  
    // ...  
}
```

- Usage of “with” creates an extra scope around it, and the contents of the scope are not known ahead for optimisation.
- Understanding the code is painful.

Avoid using global variables

- By implementation, global variables are indexed by names.
- Code inside a function tries to resolve a global variable by stepping through every scope until it reaches the global object.
- Since variables in global scope exist through the life of the script, memory deallocation can be done only at the end of execution.

Avoid using try/catch inside loops

- The try/catch block creates a local scope and creates the exception object at runtime that lives in that scope.
- Consider:

```
for ( // ... ) {  
    try {  
        // ...  
    } catch (e) { // ... }  
}
```

Avoid using try/catch inside loops (contd.)

- Can be better re-written as:

```
try {  
    for ( // ... ) {  
        // ...  
    }  
} catch (e) {  
    // ...  
}
```

Random Stuff

- Do not use `setTimeout('myFunc()', ...)`, instead use `setTimeout(myFunc, ...)`, the former does an `eval()`.
- Either `Array.join()` or `String.concat()` is better than using `“+”` when concatenating a lot of strings.

```
var s = “hello” + “ world”  
// better alternatives  
var s = [“hello”, “ world”].join(“”)  
var s = String.concat(“hello”, “ world”)
```

Section 1: Summary

- Be aware about implicit conversions
- Do not use `eval()` when not necessary
- Do not use the “Function” constructor
- Do not use the “with” statement
- Do not wrap try/catch within loops
- Avoid the use of global variables
- Do not pass function as a string to `setTimeout()`

Section 2: JavaScript and DOM

- Some performance considerations when working with the browser.
- Taking advantage of browser specific optimisations.
- Not applicable for standalone programs or those that run outside the context of a web browser (server-side JavaScript, embed etc.).

Property Access

- The following snippet of code tries to dynamically modify the styles of an element:

```
var d = document.getElementById("photos-container");  
d.style.display = ""; // make it visible  
d.style.background = "#fff";  
d.style.color = "#000";  
// some more changes to d.style
```

Property Access (contd.)

- Since we're accessing the “style” property of the element every time (in the example), we can cache the property accesses to gain a small performance increase:

```
var d = document.getElementById("photos-container"),
    e = d.style;
e.display = ""; // make it visible
e.background = "#fff";
e.color = "#000";
// some more changes to e (d.style)
```

Property Access (contd.)

- Every time a “style” gets changed for an visible element, there is an internal reflow happening in the browser. This can slow up a little bit.
- Make sure the browser has as minimum reflows as possible:

```
var d = document.getElementById("photos-container"),
    e = d.style;
e.background = "#fff";
e.color = "#000";
// some more changes to e (d.style)
e.display = ""; // and finally... make it visible
```

Property Access (contd.)

- Or, another “not so recommended” alternative: update the style with a single access:

```
var d = document.getElementById("photos-container"),
    style;
style = "background: #fff; color: #000; display:
block;";
if (typeof d.style.cssText != "undefined") {
    d.style.cssText = style;
} else {
    d.setAttribute("style", style);
}
```

DOM methods vs innerHTML

- Though purists might not be agreeing, dumping HTML into innerHTML is more than 75% faster than using DOM methods especially when adding a lot of elements.
- <http://gopalarathnam.com/examples/javascript/dommethode.html>

Having minimal reflows

- Some properties like `offsetWidth`, `offsetHeight` has internal reflows when accessed, that might unnecessarily hit your performance.
- Cache the property and re-use for your calculations.

```
var d = document.getElementById("some-elem"), ow;
if (d) {
    ow = d.offsetWidth;
    // use "ow" for your computations ...
}
```

Having minimal reflows (contd.)

- If you are appending a lot of elements to a DOM node, detach the node, append the children, and re-attach the node.

```
var foo = document.getElementById("foo"),
    container =
    document.body.removeChild(foo);
for (// ...) {
    container.appendChild(some_elem);
}
document.body.appendChild(container);
```

Event Handling

- When attaching the same event handler for a lot of elements in the page, it is best to attach the event handler to the container.
- Consider a rating control, instead of attaching the mouse events to every star (typically on local or message boards, it would be 10 x 5 at least), attach it to the parent element that has the messages.
- <http://gopalarathnam.com/examples/javascript/container-event.html>

Event Handling (contd.)

- We can save memory by using a single event handler for the container as opposed to be having for every sub-element.
- http://gopalarathnam.com/gopal_v/examples/javascript/container-event.html

Section 2: Summary

- Cache element property accesses
- When modifying a lot of styles, hide it or remove it from the DOM, and re-attach it
- Use innerHTML when inserting a lot of elements into a node
- Cache offsetWidth/offsetHeight properties before using them since they cause an internal reflow
- When binding the same event handler to an element's container, try to bind to its container to conserve memory.

That's all folks!

- Slides:
 - <http://gopalarathnam.com/talks/>
- Examples:
 - <http://gopalarathnam.com/examples/javascript>