# JavaScript Memory Leak Testing

Contents

[JavaScript Memory Leak Testing 1](#_Toc320109433)

[What a memory leak is 1](#_Toc320109434)

[What a JavaScript memory leak is 1](#_Toc320109435)

[JavaScript memory leak’s consequences 1](#_Toc320109436)

[How memory leaks occur in JavaScript 1](#_Toc320109437)

[How to avoid memory leaks 1](#_Toc320109438)

[Detection of leaks 2](#_Toc320109439)

[Detecting Tools 2](#_Toc320109440)

[Examples: 2](#_Toc320109441)

[Detect circular reference with sIEve/Drip 2](#_Toc320109442)

[Detect too much recursion issue 4](#_Toc320109443)

[Appendix 5](#_Toc320109444)

[Leak Patterns 5](#_Toc320109445)

## What a memory leak is

In its simplest form, a memory leak occurs when a program does not release memory it allocated.

## What a JavaScript memory leak is

Memory leak that associates with the particular program – JavaScript.

As JavaScript is implemented differently across the different browsers, the same JavaScript can causes different memory leak problems across the different browsers. It can happen that a same code causes memory leaks in one browser but works fine in another browser.

**So a lot of memory leaks in JavaScript are browser specific.** So if you hear Browser Memory Leak, you don’t meet new concept. They are the same thing. Generally, JavaScript memory leaks are not an issue for web applications. Users navigate between pages, and each page switch causes the browser to refresh. Even if there’s a memory leak on one page, the leak is released after a page switch. The size of the leak is minor, and consequently often ignored.

Memory leaks became more of a problem when the Ajax technology was introduced. On an ajax heavy page, users don’t refresh the page very often. Ajax technology is used to update the page content asynchronously. In an extreme scenario, the whole web application is constructed on one page. (For example: <http://causes.msn.com>). In this cases, leaks accumulate and can’t be ignored.

## JavaScript memory leak’s consequences

* Leaks will slow down the browser as the process consumes more memory
* Leaks will slow down the operating system and other processes as the bigger process is given more room

As you have already known that the leaks are browser specific, some browsers handle leaks better than others (e.g. IE 8 users whole separate processes per tab)

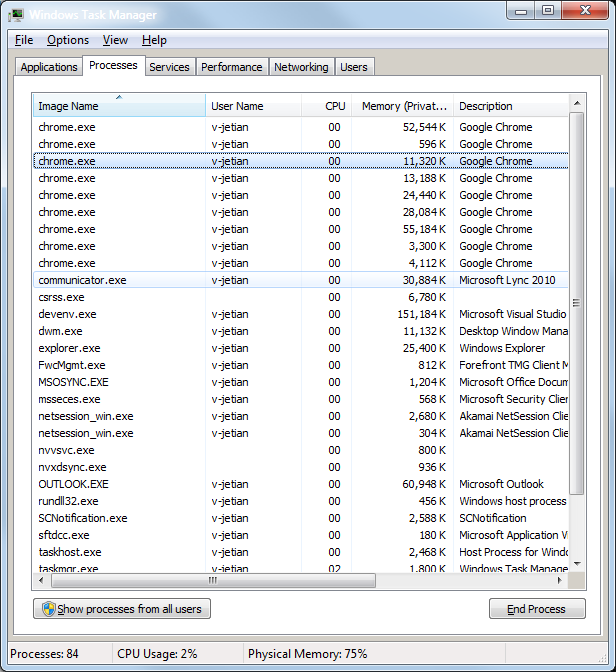
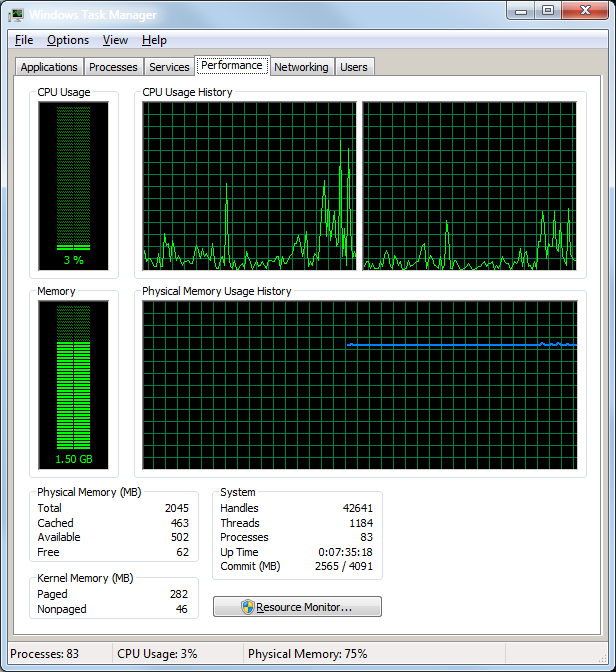
## How memory leaks occur in JavaScript

* Circular references
* Closures with DOM elements / event handlers
* Event handlers (mostly with IE)
* Too much recursion
* See Appendix ([Leak Patterns](#_Leak_Patterns)) for detail

## How to avoid memory leaks

* Use the global scope only when appropriate
* Always declare variables in functions when you use them
* Use a collector to keep references to elements with event handlers
* Avoid closure with event handlers and functions that attach event handlers or use a global event handler that delegates to listeners
* Avoid dealing with DOM elements in closures or remember to null them afterwards
* Detach circular references when you are done with them

## Detection of leaks

* Code review
* Open browser console, interact with page and notice the error messages (if any) appear in console
* Use windows Task Manager
  + 
  + 
* Use tools to detect, see the Tools section

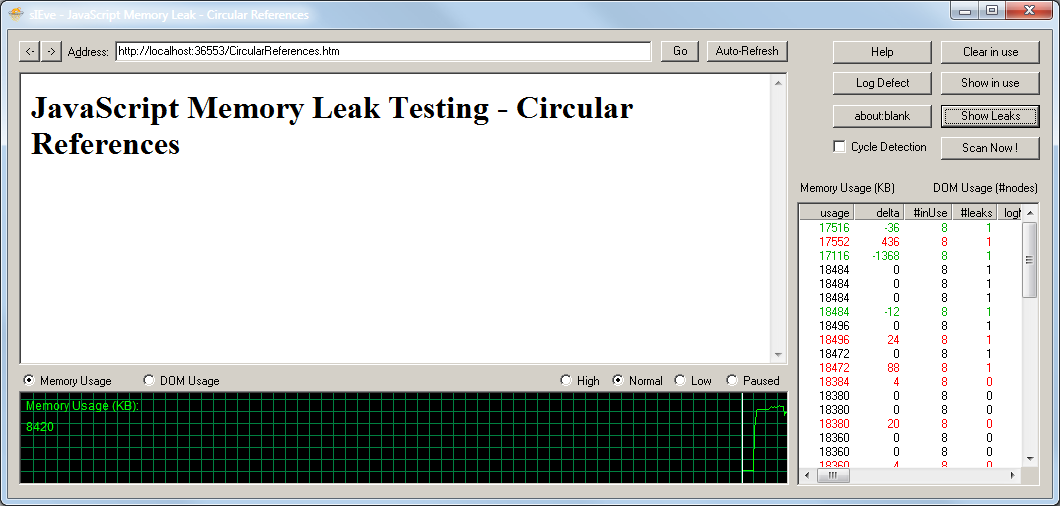
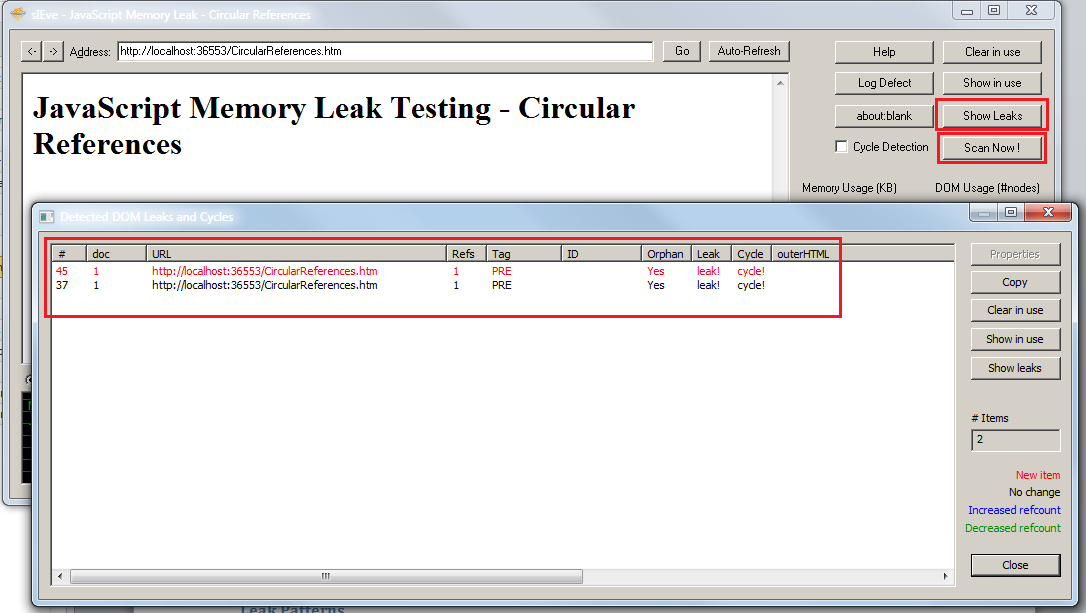
## Detecting Tools

|  |  |  |
| --- | --- | --- |
| **Name** | **For (Browser)** | **Remark** |
| [Microsoft JavaScript Memory Leak Detector (v2)](http://blogs.msdn.com/b/gpde/archive/2009/08/03/javascript-memory-leak-detector-v2.aspx) | IE6, IE7 | It can highlight the exact point in the JavaScript code where the memory leak originated. |
| [Drip](http://outofhanwell.com/ieleak/index.php?title=Main_Page) | IE | This tool only shows the result, not the reason for leaks. |
| [Leak Monitor 0.4.2](https://addons.mozilla.org/en-US/firefox/addon/2490) | Firefox | It is an open extension for Firefox. (Not supported by lasted firefox any more) |
| [SoftwareVerification JavaScript Memory validator](http://www.softwareverify.com/javascript/memory/index.html) | Firefox | You can find and track the leaks with its “hot spots” feature. But it is a commercial application, though you can use its trial version. |
| [sIEve](http://home.orange.nl/jsrosman/) | IE | Much like Drip, but is more powerful.  sIEve can’t find all leaks for your application, but it will find leaks caused by orphan nodes. The additional information, such as ID and outerHTML, can help you identify the leaking nodes. |

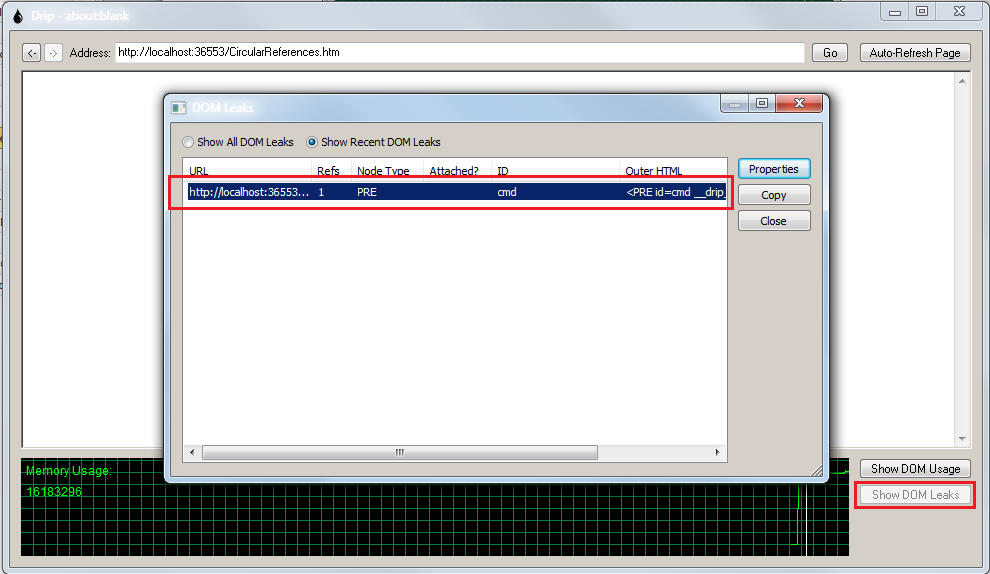
## Examples:

### Detect circular reference with sIEve/Drip

Steps with sIEve:

1. Open sIEve, and paste the target url of the testing page 
2. After the page loads complete, interact with the page or just keep the page open for a long time
3. Click the “Scan Now!” button
4. Then click “Show Leaks” button, you will see the leaked elements if any

Steps with Drip:

1. Open Drip and paste the target url
2. Interact with the page or just leave the page open for a long time
3. Click the “Show DOM Leaks” button you will see the leaked elements if any

The source code of the sample testing page (CircularReferences.htm):

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>JavaScript Memory Leak - Circular References</title>

<script type="text/javascript">

<!--

window.onload = function () {

var obj = document.getElementById("cmd");

document.getElementById("cmd").expandoProperty = obj;

alert(obj);

};

//-->

</script>

</head>

<body>

<div>

<h1>JavaScript Memory Leak Testing - Circular References</h1>

<pre id="cmd"></pre>

</div>

</body>

</html>

### Detect too much recursion issue

Please refer to the Appendix [too much recursion](#_Too_much_recursion) section.

Steps:

1. Open browser and navigate to target url
2. Press F12 (with IE / Chrome) or Ctrl+Shift+K (with Firefox) to open Console
3. Interact with the page and notice the error messages appear in the console if any
4. Investigate if it is caused by recursion.

## Appendix

### Leak Patterns

As web developers know, Internet Explorer (IE) is different from Firefox and other browsers. In this section, the memory leak patterns and issues discussed are mainly targeted at, but not limited to, IE. Good practices should be applicable to all browsers.

#### Circular references

Circular references are the root cause of nearly every leak (As discussed in the above section). Generally, IE can handle the circular references and dispose of them correctly in the JavaScript world. The exception occurs when DOM objects are introduced. Circular references occur and cause the DOM node to leak when the JavaScript object holds the reference to the DOM element, and the DOM element’s property holds the JavaScript object. [Listing 1](#_Listing_1._Leak) shows a code sample that’s often used to demonstrate this problem in articles about memory leaks.

##### Listing 1. Leak by circular references

var obj = document.getElementById(“someLeakingDIV”);

document.getElementById(“someLeakingDIV”).expandoProperty = obj;

To resolve the issue, explicitly set the expandoProperty to null when you’re ready to remove the node from the document.

#### Clossures

Closures cause memory leaks because they create circular references without being aware of it. The parent function’s variable will be held as long as the closure is alive. Its life cycle goes beyond the function scope, which will cause a leak if not handled carefully. [Listing 2](#_Listing_2._Closure) shows a leak caused by closure, which is a common coding style in JavaScript.

##### Listing 2. Closure that leaks

<html>

<head>

<script type=”text/javascript”>

window.onload = function () {

var obj = document.getElementById(“element”);

// this creates a closure over “element”

// and will leak if not handled properly.

obj.onclick = function(evt) {

alert(“leak the element DIV”);  
};

};

</script>

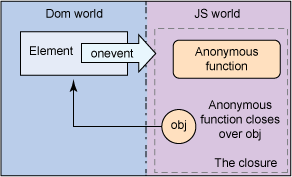
</head>

<body>

<div id=”element”>Leaking DIV</div>  
</body>  
</html>

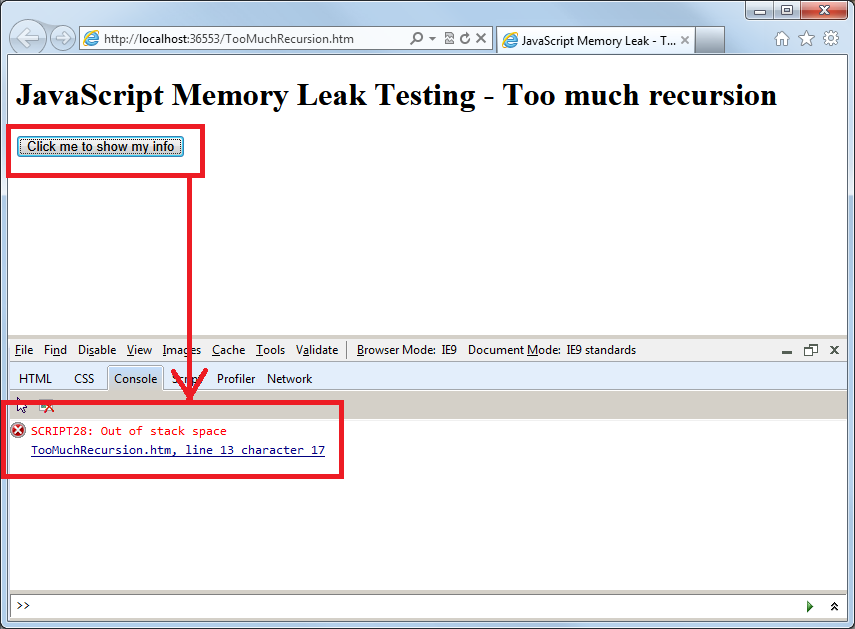
If you use sIEve – a tool that detects orphan nodes and memory leaks – you’ll notice that the element DIV is being referenced twice. One of the reference is held by the closure (the anonymous function assigned to the *onclick* event) and can’t be deleted even if you remove the node. If your application removes the element node later, the JavaScript reference will still hold an orphan node. That orphan node will cause the memory leak.

To understand why closure creates circular references you can refer to the below diagram:

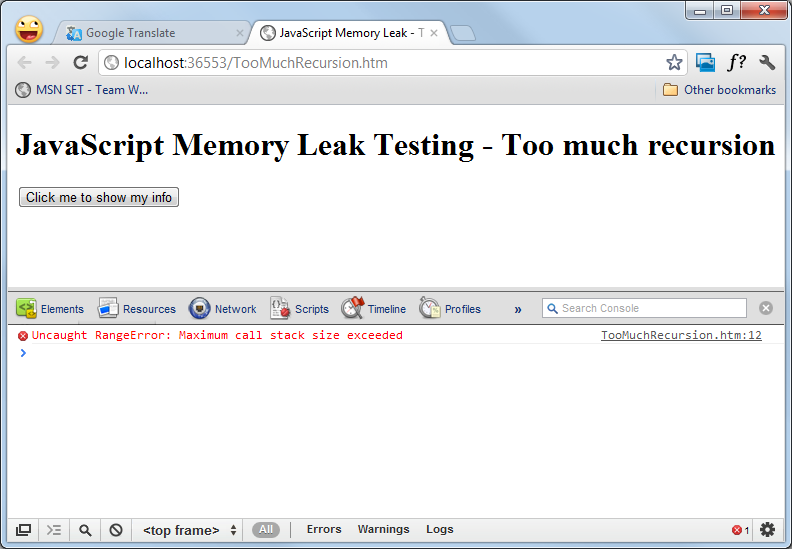


#### Too much recursion

This is the most serious issue among the leaks mentioned in this article. It is caused by unintentional redefine some javascript methods with the same names. The consequence is that it will run out of memory so the browser (All kind of browsers) will stop all the javascript execution to avoid crash and so that the page’s functionality is broken. Below screenshot is an example:



Screen shot with IE 9



Screen shot with Chrome

The source code of this example is listed below.

From the source code we can easily find out the root cause is the Object.prototype.toString() method is redefined 2 times, but in the practical case, this can be very hard to find out because the duplicated redefining may be placed in 2 or more separate js files.

This example can be repro’d in IE 9, Chrome, Firefox, but not in IE 7, 8.

The source code of TooMuchRecursion.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>JavaScript Memory Leak - Too much recursion</title>

<script type="text/javascript">

<!--

(function () {

// back up the original toString() method

Object.prototype.toString2 = Object.prototype.toString;

// rewrite the toString() method

Object.prototype.toString = function () {

var originalString = this.toString2();

if (arguments.length > 0) {

return "{ " + originalString + " }\r\n";

} else {

return "{ " + originalString + " }";

}

};

})();

(function (window) {

// back up the original toString() method

Object.prototype.toString2 = Object.prototype.toString;

// rewrite the toString() method

Object.prototype.toString = function () {

var originalString = this.toString2();

if (arguments.length > 0) {

return "{ " + originalString + " }\r\n";

} else {

return "{ " + originalString + " }";

}

};

function showInfo(o, infoDivId) {

var infoDiv = document.getElementById(infoDivId);

infoDiv.innerHTML += o.toString(true);

}

window.showInfo = showInfo;

})(window);

//-->

</script>

</head>

<body>

<div>

<h1>JavaScript Memory Leak Testing - Too much recursion</h1>

<p><input type="button" value="Click me to show my info" onclick="showInfo(this, 'cmd');" /></p>

<pre id="cmd"></pre>

</div>

</body>

</html>