



Categories

Interesting Stuff

← Seminar on Web

Context Menu Using Raw JavaScript →

JavaScript Mistakes You Must Avoid

Posted on [May 8, 2011](#) by [iFadey](#)

[+](#) Share / Save [↓](#)

Search

Archives

- [October 2011](#)
- [September 2011](#)
- [August 2011](#)
- [July 2011](#)
- [May 2011](#)
- [April 2011](#)
- [February 2011](#)
- [January 2011](#)
- [December 2010](#)
- [November 2010](#)
- [October 2010](#)
- [September 2010](#)
- [August 2010](#)

If you are new to JavaScript and you write raw JavaScript or use any framework (jQuery, Mootools, Dojo, YUI) with it, you must avoid few mistakes. Actually these are my experiences when I was learning JavaScript.

Equality Operator

You may know that in js, two operators are used for comparing values. First is == (two equal signs). This operator compare the values but it doesn't compare the data type of operands. For example if first operand is 1 and the second is true, the result will be true.

```
if( 1 == true ) {  
    //this code will run  
}
```

Here are more examples.

```
1 == "1"           //true  
"true" == true    //false  
1 == true         //true  
"0" == 0          //true  
"" == 0           //true  
" " == 0          //true  
"Str" == false   //false  
"Str" == true    //false
```

- July 2010
- June 2010
- May 2010
- April 2010
- March 2010
- November 2009
- October 2009

Recent Posts

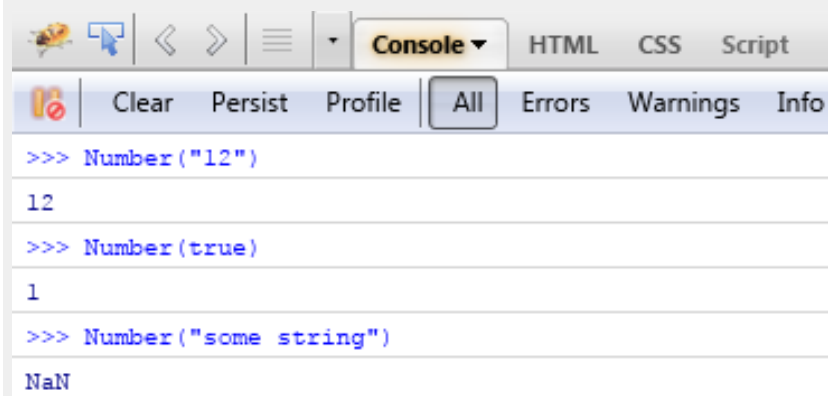
- [Interesting Stuff on the Web – 3](#)
- [Interesting Stuff on the Web – 2](#)
- [Browser Plugin Detection](#)
- [Interesting Stuff on the Web – 1](#)
- [Interview Experience With Google](#)
- [Context Menu Using Raw JavaScript](#)
- [JavaScript Mistakes You Must Avoid](#)
- [Seminar on Web](#)

Some of the results are unexpected specially for those who don't know how JavaScript evaluates == operator. Actually every operand (no matter what data type it has) is converted to Number data type before comparison.

Consider the first example (1 == "1"). The first operand is already a number so no conversion occurs. Second operand is string so it's converted or parsed to number. Now the second operand "1" (string) is converted to 1 (number).

In the second example ("true" == true) is false because if the string contain characters other than digits, conversion to number will return NaN which means Not a Number. If anything is compared with NaN, comparison will always return false.

You can check what value will be returned after conversion to number using the [Number](#) constructor. Following are some tests in Firebug.



```
>>> Number("12")
12
>>> Number(true)
1
>>> Number("some string")
NaN
```

Now you maybe wondering how === operator works. Equality operator (with 3 equal signs) in JavaScript compare not only the value of operands, but also the data type. If the data type of operands is different, it will always return false. Both the data type and value of operands must be same in order to make the condition true.

```
4 === 4           //true
"2" === 2         //false
1 === true        //false
"true" === true   //false
```

Not Assigning null to Reference Types

It's common mistake that many js developers don't assign null value to variables of reference types (object or Array) when their use is finished. Take a look at following example.

```
var arr = [1, 2, 3];

//perform some processing on arr

//assign null to arr when its use is finished
arr = null;
```

The benefit of assigning null to reference types is that the garbage collector of js engine automatically reclaims the memory used by that variable. Remember that this is specially important for variables whose scope is large like global variables. Local variables are automatically destructed when they are out of scope (specially js engines with Mark and Sweep garbage collector).

Reference Variable Initialization

Never try to initialize multiple reference variables (like object and array) in a single assignment statement. This can be better understood using an example.

```
1  var arr1 = [1, 2, 3]
2    , arr2 = ['a', 'b', 'c'];
3
4  //reset both arrays
5  arr1 = arr2 = [];
6
7  //add a single item in arr2 and arr1
8  arr2.push( 32 );
9  arr1.push( 10 );
10
11 //print both arrays and you will see same result
12 //OUTPUT: 10, 32
13 alert( arr1.join() );
14 alert( arr2.join() );
```

On line 1 and 2, two arrays are created. Then those arrays are reinitialized with blank array in a single statement on line 5. The problem with this statement is that now both arrays (arr1, arr2) are pointing to same array in the memory. So changes in one will automatically be reflected in the other.

On line 8, integer 32 is added in arr2 and on line 9, integer 10 is added in arr1. To check the output of both arrays, join method is called on each of them on lines 13 and 14. Note that both arrays contain same values.

Don't Forget Keyword var

In JavaScript you can declare variables with keyword `var`. But it also allows you to use variables without any declaration. There's a very critical difference between these two ways of using variables. Consider the following example.

```
function createVar() {  
    var myVar = 'local';  
};  
  
alert( myVar ); //output: undefined
```

As you can see from the above example when a variable is declared with keyword `var`, it's not accessible in the outer scope. Let's change this example to define a variable without `var` in function.

```
function createVar() {  
    myVar = 'local';  
};  
  
alert( myVar ); //output: local
```

Note that a variable defined without keyword `var` in function is also accessible from the global scope. In other words `var` makes the variable local. So be careful when using variables in JavaScript. Always declare your variables using keyword `var` before using them.

Not Using Event Delegation

Attaching event handler is simple in JavaScript. For example following code adds a click handler to anchor tag having id myLink.

```
document.getElementById('myLink').addEventListener( 'click', function()  
    //you code goes here...  
}, false );
```

Now suppose you want to add a click handler to all the td elements in table. Are you going to write a handler for each td in table?

```
<table id="myTable">  
  <tbody>  
    <tr>  
      <td>1, 1</td>  
      <td>1, 2</td>  
    </tr>  
  
    <tr>  
      <td>2, 1</td>  
      <td>2, 2</td>  
    </tr>  
  </tbody>  
</table>
```

Here event delegates help us. In our case we will attach a single click event handler to myTable and within that we will check to see if a td element is clicked or not. In this way we don't have to write and attach event handlers to every td in table. Such kind of handler is known as event delegate. Here's the code for it.

```
document.getElementById( 'myTable' ).addEventListener( 'click', function(e) {
    if( e.target && e.target.nodeName == 'TD' ) {
        console.log( e.target.innerHTML );

        //to access id
        //console.log( e.target.id );

        //to access className
        //console.log( e.target.className );
    }
}, false );
```

innerText VS innerHTML

New js developers are often confused between innerHTML and innerText properties. Both innerHTML and innerText are used with element objects. innerHTML gets the html code inside the element and innerText gets the text inside the element.

innerHTML

Html

```
<div id="myDiv">
```

```
This text is in DIV.
```

```
<p>A para in div element.</p>
```

```
</div>
```

Javascript

```
document.getElementById('myDiv')  
    .innerHTML;
```

Output

```
This text is in DIV.
```

```
<p>A para in div element.</p>
```

The dark gray background in Html is actually the output of innerHTML property. Note that the Html tags (in our case <p>) are also included in the output.

Let's take a look at innerText example.

innerText

Html

```
<div id="myDiv">  
  This text is in DIV.  
  <p>A para in div element.</p>  
</div>
```

Javascript

```
document.getElementById('myDiv')  
  .innerText;
```

Output

```
This text is in DIV. A para in div element.
```

As you can see from the above example that `innerText` gets the text inside the element (without html tags).

Adding Nodes in Bulk

It's common in JavaScript to append list of nodes to some element in DOM. For example you may want to append a list of names to ul received from server using Ajax call. One way of doing this is shown below.

```
window.onload = function() {
```

```
//ul element - <ul id="list"></ul>
var list = document.getElementById( 'list' );

var item = null;

//suppose this json is returned from ajax call
var ajaxResponse = [
    { 'name' : 'Haiku' },
    { 'name' : 'Linux' },
    { 'name' : 'OS X' },
    { 'name' : 'Windows' }
];

//add all names in ajaxReponse to documentFragment
for( var i in ajaxResponse ) {
    item = document.createElement( 'li' );
    item.appendChild( document.createTextNode( ajaxResponse[ i ].name ) );
    list.appendChild( item );
}
} //end onload

/*
...: OUTPUT :...
<ul id="list">
<li>Haiku</li>
<li>Linux</li>
<li>OS X</li>
<li>Windows</li>
</ul>
*/
```

The problem with this method is that each time the item is appended to list in "for in" loop, the DOM gets updated immediately. This hurts performance because DOM manipulation is costly process.

"DocumentFragment is light weight version of document and it has no visual representation on the web page."

The same output can be achieved using DocumentFragment. DocumentFragment is light weight version of document and it has no visual representation on the web page. Following is the example to append list items using DocumentFragment.

```
window.onload = function() {
  //create DocumentFragment
  var documentFragment = document.createDocumentFragment();

  var list = document.getElementById( 'list' ); //<ul id="list"></ul>
  var item = null;

  //suppose this json is returned from ajax call
  var ajaxResponse = [
    { 'name' : 'Haiku' },
    { 'name' : 'Linux' },
    { 'name' : 'OS X' },
    { 'name' : 'Windows' }
  ];

  //add all names in ajaxReponse to documentFragment
  for( var i in ajaxResponse ) {
    item = document.createElement( 'li' );
    item.appendChild( document.createTextNode( ajaxResponse[ i ].na
```

```
documentFragment.appendChild( item );  
}  
  
//append all items in documentFragment to list  
list.appendChild( documentFragment );  
}
```

John Resig wrote an excellent [post](#) explaining DocumentFragment and its impact on performance.

DOM Manipulation using innerHTML

Never use arithmetic assignment operator (+=) with innerHTML to add new markup. Whenever you change innerHTML, DOM updation occurs (Browser redraws the markup). So adding new markup using += operator (specially in a loop) decreases performance.

```
var container = document.getElementById( 'container' );  
  
for( var i = 1; i <= 10; ++i ) {  
    container.innerHTML += 'Item ' + i + '<br />';  
}
```

Always use a temporary variable to store the markup you want to assign to innerHTML as shown below.

```
var container = document.getElementById( 'container' )
```

```
, str = '';  
  
for( var i = 1; i <= 10; ++i ) {  
    str += 'Item ' + i + '<br />';  
}  
  
container.innerHTML += str;
```

This entry was posted in [Blog](#), [JavaScript](#). Bookmark the [permalink](#).

[← Seminar on Web](#)

[Context Menu Using Raw JavaScript →](#)

46 Responses to *JavaScript Mistakes You Must Avoid*



tom says:

May 8, 2011 at 11:43 pm

great! thank you for that informative post.

Tom



rob d says:

May 9, 2011 at 4:24 pm

Great Post! I learned many good points!



Andres Descalzo says:

May 9, 2011 at 4:26 pm

in the example of “innerText” you “innerHTML”. thank you for that informative post.



Rodger says:

May 9, 2011 at 4:47 pm

FYI: Your “innerText” example calls “innerHTML” again. Probably just a cut-n-paste error. No need to include this comment.



PCTip says:

May 9, 2011 at 4:56 pm

Thanks for this post. I like the way you handle DOM elements and event delegation. That are really useful tips.



Paul says:

May 9, 2011 at 6:06 pm

Two quick notes.

First, it's better to `delete arr;` rather than set it to null. Three reasons for this: it's more obvious what you are doing, the GC knows at that moment that the var can be collected, and null is a data type which slows the parser down during static analysis. But you are

absolutely right that you want that old array marked for clean up.

Second, not all browsers have innerText. There is a way to do this with feature detection, but I can't remember it off hand. (Also, there is a typo in the code for innerText. It uses innerHTML.)



iF says:

May 9, 2011 at 8:27 pm

Thanks for tip Paul.

Second: Yes you are right. Not all browsers support innerText. Like Firefox use `textContent` instead of `innerText`

Typo is fixed now.



Vinny says:

May 9, 2011 at 6:42 pm

Hey, nice article; totally using doc fragments now, much more streamlined.

Also, for `innerText` vs `innerHTML` are they both suppose to be the same code?



IainB says:

May 9, 2011 at 6:45 pm

The `innerText` image has `.innerHTML()` in the javascript section... should be `innerText()`



Christopher says:

May 9, 2011 at 9:13 pm

Good article! Special thanks for explaining the “Equality Operator” 😊



Jigar Shah says:

May 9, 2011 at 9:43 pm

Can you suggest a good (small 😊) book which can help me to clear basic js concepts ?



iFadhey says:

May 9, 2011 at 10:01 pm

Here's a very nice book on javascript which is free for online reading:

[Eloquent JavaScript](#)

Buy a copy of it as well if you like it to support the author:

[Amazon Link](#)



Garrett says:

May 10, 2011 at 4:07 am

“First, it's better to delete arr; rather the set it to null. Three reasons for this:”

Here's one reason not to do it: It doesn't work.

Check the result of that operation.

```
(function() {  
  var arr = [1,2,3]  
  alert(delete arr); // false  
  alert(arr);  
})();
```

Kangax article explains it.

<http://perfectionkills.com/understanding-delete>

And BTW, An array is not a reference type; a “Reference”, as defined in ES3, consists of a base object and the property name. In the example above, the base object of `arr` is what is called an Activation Object in ES3.



Adelar says:

May 10, 2011 at 6:03 am

Great article 😊



EJ Frias says:

May 10, 2011 at 12:02 pm

Very informative article..

Btw, just saw a typo (the 1 == true //false) on

```
4 === 4 //true
```

```
"2" === 2 //false
```

```
1 == true //false
```

```
"true" === true //false
```

Cheers ^_^



iFadey says:

May 10, 2011 at 1:27 pm

fixed!



Marcos Zanona says:

May 10, 2011 at 5:41 pm

Thanks a lot man, really instructive, I will surely keep this in my bookmarks.



Sandeep says:

May 10, 2011 at 7:56 pm

Good and exhaustive list



Ian says:

May 10, 2011 at 8:53 pm

for...in? That's no good. Tip #1 should be never to use for...in, it's the slowest possible loop.

<http://jsperf.com/custom-for-loop-vs-regular-for-loop/6>



Marc Qualie says:

May 23, 2011 at 7:52 pm

That would be true if it were for Array types, but the data type was Object. Since Object types don't have a length property, and normally won't have integers as keys, a normal for loop can't be used.

Pingback: [Javascript Mistakes You Must Avoid « Clear Illusion](#)



Nicholas Johnson says:

May 11, 2011 at 11:48 am

Great article, good techniques and some stuff I didn't know, especially the stuff about explicit garbage collection by setting reference types to null. Thanks!

Pingback: [Webprogrammierung & Design » Blog Archive » JavaScript Tutorial: JavaScript einbinden – http://www.html-seminar.de](#)

Pingback: [jQuery Weekly Roundup #4: 5/11/11 - webdesigncrowd.com](#)



Filip says:

May 12, 2011 at 6:45 am

in the “Don’t Forget Keyword var” part, I think you really need to stress that not defining variables with the var keywords and hence putting them in the global scope is a very bad thing. Firstly these variables are visible to everything, secondly, when referencing the variable in your code, it takes longer to get to it as you have to go through all the scopes in the hierarchy.

It should also be mentioned that variable scope in JavaScript is delimited by a function declaration not by block. Since function can also contain other functions, variables defined in the top level function are visible to all the contained functions as well, but then we start getting into closures... probably another post, hey?



iFadley says:

May 13, 2011 at 5:08 am

Thanks for highlighting it. You are right I must have gone in more detail about it



Alexander Trefz says:

May 13, 2011 at 6:58 pm

“So adding new markup using += operator (specially in a loop) decreases performance.” And in the “Good-Example” below you used it yourself. Its not the Operator, it is the same error as with DocumentFragment, “Avoid multiple Reflows for actions that could be done with one reflow”.



mwilcox says:

May 13, 2011 at 7:13 pm

One of the better articles on JS mistakes. Congrats!

You don't point out that Firefox does not support innerText. It uses textContent. And innerText != textContent.

<http://clubajax.org/plain-text-vs-innertext-vs-textcontent/>

Pingback: [jQuery, Railscasts, Mobile Applications, & More | Tom McFarlin](#)



Jake Verbaten says:

May 14, 2011 at 2:21 am

As is already mentioned "innerText" is not supported by all browsers.

The problem with this is that "textContent" is the standard way to manipulate text as set by the W3C standards.

"innerText" is an IE only function that's not in the standard. It's like recommending you use "attachEvent" or htc files.

For future reference please do not spread examples with IE only proprietary code.

Pingback: [Link-urile săptămânii 9-15 mai | Staicu Ionuț-Bogdan](#)



alemarko says:

May 16, 2011 at 12:04 pm

Great article, you should just correct one bit:

>> Actually every operand (no matter what data type it has) is converted to Number data type before comparison.

Definitely not true, because strings comparisons would not work.

```
“abc” == “def” // false
```

```
“abc” == “abc” // true -> you say it would be false, same as Nan == Nan
```

Also, maybe to warn readers about result of following comparison and teach/remind them what for function ‘isNaN()’ is used:

```
Nan == Nan // false
```

```
isNaN(2) // false
```

```
isNaN(Nan) // true
```

Regards,

Alex

Pingback: [Linkhub – Woche 19-2011 | PehBehBeh](#)



51 Website Design says:

May 17, 2011 at 5:42 am

Uhh you have one problem. All browsers don't support innertext, still):. Great post though.



Ryan Gasparini says:

May 19, 2011 at 1:20 am

The Event portion will not work for IE users. `addEventListener` is the W3C version, “e” (a.k.a. `MouseEvent`) is not provided to event functions (is `window.event`), and `target` is actually `srcElement`.



David says:

May 21, 2011 at 3:36 am

Great article. I liked the part about document fragment. I never heard about that. Is this implemented consistently among major browsers?



iFadey says:

May 21, 2011 at 8:47 am

Yes David it's supported in all major browsers (even in IE6)



Colm Britton says:

May 23, 2011 at 1:30 pm

Hi,

Excellent article, I found it very useful. Although I do have one point to make. I'm not sure your first tip about the == operator is 100% correct. If all strings are converted to Numbers and returned as NaN then no 2 strings would match;

NaN == NaN always returns false

however

if you compare 2 identical strings true will be returned

“test” == “test” will return true

but according to your explanation “test” would be converted to NaN so the comparison would become

NaN == NaN

and as I said earlier this returns false.

Hope that helps.



iF says:

May 23, 2011 at 2:00 pm

I think must have mentioned this point. If both operands are strings, then they are not converted to number for comparison. That's why it's returning true.

If a number is compared with string, the string is converted to Number. If any non numeric value is stored in string, then it returns NaN. Any operand compared with NaN will always return false.

This happens with == (double equal) operator



Pascal says:

May 27, 2011 at 2:24 am

Really great article!

I knew some parts of it already but especially the null assignment for garbage collection is something good to know 😊



Junior says:

May 27, 2011 at 5:50 am

I think you should learn a bit more about the language before going out giving “advice” on the internets. Assigning to null is only necessary if your variable is eating a lot of memory AND it’s being kept in a closure that will be kept. Garbage collection exists precisely so you don’t need to manage memory manually.



Senior says:

October 20, 2011 at 1:16 pm

Maybe you should learn a bit more about spelling before posting derogatory posts on the *internets*. If you had left out your first sentence your point would not have been lost as it is at present.



James Chalmers says:

May 27, 2011 at 2:07 pm

Great informative post, plenty of tips for beginners and some old gotcha’s for

professionals to watch out for. One minor nitpick: can you replace all instances of the word 'specially' with 'especially' in your post?



James



Cristian Giordano says:

May 27, 2011 at 2:29 pm

Thanks for the post, its good to read pure JS articles rather than a JS framework.



Michiel Bakker says:

May 27, 2011 at 3:21 pm

Nice post. I consider most of the examples/statements made to be common knowledge, but after reading the comments, I guess it's not. But it's nice to see some of the mistakes we all made when we first started tinkering with JS. 😊

For other stuff in this post I usually use jQuery, which really does most of the heavy lifting, in the most effective way. But it makes you lazy, which is a bad thing I guess.

Never heard (or read) about `createDocumentFragment()`. It's nice to learn some new stuff once in a while!

Thanks,

Michiel

Pingback: [HTML5, CSS, Javascript useful links](#) « 달달한 바람같이

iFadey supports [Firefox](#) and [Maxthon 3+](#)

Designed and Developed by [iFadey](#)

Copyright 2009 - 2011. All Rights Reserved