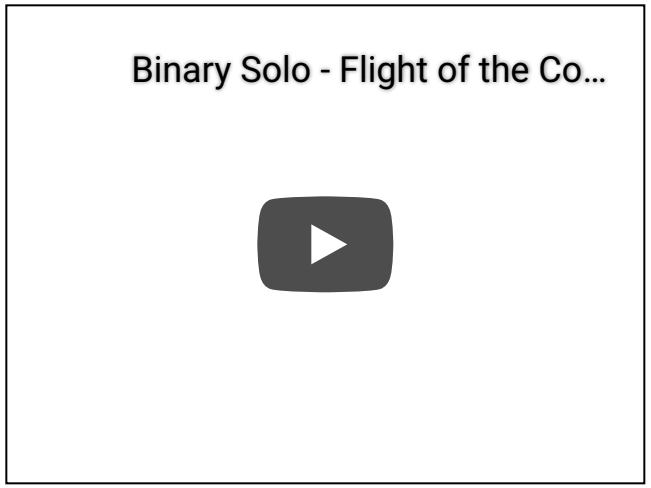
Tilde or the Floor? Practical use for JavaScript bitwise operators. (#1)

Submitted by Piotr Rochala on Sat, 11/26/2011 - 12:10



JavaScript bitwise operators are one of these features of JavaScript that we always forget about. It seems like we assume there is no good use for them in a client-side code. Why would we care about `ones` and `zeros` if we're not using binary? Why would we ever care for binary AND, OR, XOR or left/right shift?

I am not going to explain how these operators are affecting binary numbers because there are plenty of resources available covering this topic.

Instead, in a series of articles, I will focus on a practical use for few of them showing how to make your JavaScript code faster, more compact and unfortunately often less readable.

Bitwise NOT - The ~ operator

Apart from "inverting the bits of its operand" bitwise NOT in JavaScript is actually very useful not only when it comes to binary. Firstly, it has a very interesting effect on integers - it converts the integer to -(N+1) value. For example:

```
~2 === -3; //true
~1 === -2; //true
```

```
~0 === -1; //true
~-1 === 0; //true
```

However, the most practical way of utilizing the power of this operator is to **use it as a replacement for `Math.floor()` function** as double bitwise NOT performs the same operation a lot quicker. You can use it, to convert any floating point number to a integer without performance overkill that comes with `Math.floor()`. Additionally, when you care about minification of your code, you end up using 2 characters (2 tildes) instead of 12.

Example use of double tilde / double bitwise NOT operator:

```
~~2 === Math.floor(2); //true, 2
~~2.4 === Math.floor(2); //true, 2
~~3.9 === Math.floor(3); //true, 3
```

Performance difference converting a list of 100000 float numbers to integer:

#	Browser	Math.floor()	Bitwise double NOT ~~
#1	Firefox 7.0.1	42ms	29ms
#2	Firefox 7.0.1	44ms	28ms
#3	Chrome 15	63ms	64ms
#4	Chrome 15	63ms	68ms
#5	IE8	265ms	192ms
#6	IE8	324ms	190ms

This test is also available here: http://jsperf.com/jsfvsbitnot.

 Performance difference between Math.floor() and JavaScript bitwise double NOT

I must admit these results are very surprising - ~~Internet Explorer actually performs better than Chrome!~~ Math.floor() is still faster on Internet Explorer than bitwise NOT. All other browsers are indeed behaving as expected - JavaScript bitwise NOT performs a lot better.

Summary

Use double bitwise NOT when:

1. You want to convert the number from float to integer.
2. You want to perform same operation as Math.floor() but a lot quicker.
3. You want to minimalize your code.

Do not use double bitwise NOT when:

1. You run Google Chrome (apparently?).
2. You care about readability of your code.

Next..

In next article I will describe binary left shift and right shift and how to practially use them in your JavaScript code.

Tags:
JavaScript

neonguru • 4 years ago

Be careful with negative numbers: ~~-2.4 !== Math.floor(-2.4)
result: -2 !== -3

4 | • Reply • Share ›

Anonymous • 5 years ago

~~2.4 === Math.floor(2); //true, 2

shouldn't it be Math.floor(2.4) ? :-)

1 | • Reply • Share ›

Felix Gnass • 6 years ago

The ~ operator is also useful in combination with String.indexOf.
Instead of testing for == -1 or >= 0 you can do the following:

```
var s = "foo";
if (!~s.indexOf("boo")) console.log("no boo in foo");
if (~s.indexOf("oo")) console.log("woohoo");
```

1 ^ | v • Reply • Share ›

Michael Bøcker-Larsen ➚ Felix Gnass • 3 years ago

Would't it be a better example if the second was 'fo' as index would be 0 which would otherwise be treated as false?

^ | v • Reply • Share ›

B.Max • a year ago

Using `| 0` (e.g: 2.4 | 0 === 2) for flooring and `| 1` (e.g: 2.4 | 1 === 3) seems better.

^ | v • Reply • Share ›

elle • a year ago • edited

Shouldn't the single ~ be (-n + 1) instead of -(n + 1)?

^ | v • Reply • Share ›

xgqfrms • 3 years ago

That ~~ is a double NOT bitwise operator.

^ | v • Reply • Share ›

Rahul Gupta • 5 years ago

Nice Article for JavaScript operations

^ | v • Reply • Share ›

ngoldman • 6 years ago

re: `~~2 === Math.floor(2); //true, 3`
nope! should be `//true, 2`

^ | v • Reply • Share ›

Oncle Tom • 6 years ago

Thanks for this explanation. I only knew about the ~ to -(N+1) (which is great when dealing with Array.indexOf).

By the way, Chrome 29 seems to greatly fix the problem, hence it improved gradualy its performance since Chrome 21.

▲ | ▼ • Reply • Share ›

zap • 7 years ago

It appears that since Chrome 25 the V8 is actually doing a ~~number when Math.floor is called. (See the linked jsperf test above)

▲ | ▼ • Reply • Share ›

espretto@gmail.com ➚ zap • 6 years ago

that'd be too bad since `~~` rounds towards zero and converts to a 32-bit Integer. How would you correctly floor number in range [Math.pow(2, 32), Number.MAX_VALUE] without x - (x %1) which i think might also lead to wrong results in some cases because of the limits of ieee's float's bias and mantisse. not sure on that last one though.

▲ | ▼ • Reply • Share ›

Tim Oxley • 7 years ago

Perhaps the reason the results are so surprising is that jsperf numbers are measured in "operations per second (higher is better)", so "Internet Explorer actually performs better than Chrome!" is the opposite of true.

▲ | ▼ • Reply • Share ›

Piotr ➚ Tim Oxley • 7 years ago

thanks Tim for pointing this out, I can't believe I missed this important detail!

▲ | ▼ • Reply • Share ›

Chris • 7 years ago

Thanks for posting this. I don't quite understand floating point representation well enough to see how two sequential bitwise inversions effectively recasts the object as an integer. Do you have any good reference materials that explains why we can do

this neat trick? Thanks!

▲ | ▼ • Reply • Share ›

Smithf67 • 5 years ago

I appreciate you sharing this article.Thanks Again. Really Cool. ddfdgeddgcbfgdcd

▲ | ▼ 1 • Reply • Share ›

L. Nielsen • 8 years ago

When using bit operations to remove decimals, both ~~n and any of the more traditional ways like (n|0) or (n>>0), the operations isn't equivalent to Math.floor, which rounds down, but rather it rounds towards zero (commonly known as "truncating" rounding).
That's actually great since JavaScript doesn't have a Math.trunc function, and there isn't any other (simple) way to get that functionality (because (n

▲ | ▼ 1 • Reply • Share ›

a0dh ➚ L. Nielsen • 3 years ago

ES6 has Math.trunc available

▲ | ▼ • Reply • Share ›

ALSO ON **ROCHA.LA**

Fun With Flags winning awards and stuff!

1 comment • 4 years ago


Avatar

Ibrar allshore — your'e great

Excuse me, there's an (HTML5) elephant in the room!

1 comment • 4 years ago


Avatar

Phil — This is really impressive!

jQuery slimScroll

503 comments • 4 years ago


Avatar

Srinu Dola — Have same issue, any one knows the fix please let me know

HTML5 Mohawk Generator

1 comment • 4 years ago


Avatar

rochala — Check out this patriotic mohawk: http://rocha.la/mohawk?p=f,...

Subscribe Add Disqus to your siteAdd DisqusAdd