

Is there a “null coalescing” operator in JavaScript?

Asked 10 years, 6 months ago Active 6 months ago Viewed 283k times

▲ Is there a null coalescing operator in Javascript?

1202 ▼ For example, in C#, I can do this:

```
String someString = null;
var whatIWant = someString ?? "Cookies!";
```



180

The best approximation I can figure out for Javascript is using the conditional operator:

```
var someString = null;
var whatIWant = someString ? someString : 'Cookies!';
```

Which is sorta icky IMHO. Can I do better?

javascript

operators

null-coalescing-operator

null-coalescing

edited Feb 22 '14 at 9:24



user2864740

45.4k 8 84 154

asked Jan 24 '09 at 18:18



Daniel Schaffer

33.3k 27 103 153

15 note from 2018: `x ?? y` syntax is now in stage 1 proposal status - [nullish coalescing](#) – Aprillion Jun 3 '18 at 9:00

1 There is now a [Babel plugin](#) which incorporates this exact syntax. – Jonathan Sudiaman Jul 29 at 14:56

Note from 2019: now is stage 3 status! – Daniel Schaffer Jul 29 at 16:19

8 Answers



The JavaScript equivalent of the C# null coalescing operator (`??`) is using a logical OR (`||`):

1891 `var whatIWant = someString || "Cookies!";`



There are cases (clarified below) that the behaviour won't match that of C#, but this is the general, terse way of assigning default/alternative values in JavaScript.

Clarification

Regardless of the type of the first operand, if casting it to a Boolean results in `false`, the assignment will use the second operand. Beware of all the cases below:

```
alert(Boolean(null)); // false
alert(Boolean(undefined)); // false
alert(Boolean(0)); // false
alert(Boolean("")); // false
alert(Boolean("false")); // true -- gotcha! :)
```

This means:

```
var whatIWant = null || new ShinyObject(); // is a new shiny object
var whatIWant = undefined || "well defined"; // is "well defined"
var whatIWant = 0 || 42; // is 42
var whatIWant = "" || "a million bucks"; // is "a million bucks"
var whatIWant = "false" || "no way"; // is "false"
```

edited Jan 24 '09 at 19:10

answered Jan 24 '09 at 18:25



Ates Goral


109k 22 119 179

40 Strings like "false", "undefined", "null", "0", "empty", "deleted" are all true since they are non-empty strings. – [some](#) Jan 25 '09 at 5:25

2 @Ates Goral: +1 and `Array.prototype.forEach = Array.prototype.forEach || function(... is WHAT?` – [Marco Demaio](#) Jul 9 '10 at 18:22 ✎

91 Of note is that `||` returns the first "truey" value or the last "falsey" one (if none can evaluate to true) and that `&&` works in the opposite way: returning the last truey value or the first falsey one. – [Justin Johnson](#) Oct 25 '10 at 2:09 ✎

17 FYI to anybody that still cares, the 0 and empty string being evaluated the same as nulls if you use the type's constructor to declare it. `var whatIWant = new Number(0) || 42; // is Number {[[PrimitiveValue]]: 0}` `var whatIWant = new String("") || "a million bucks"; // is String {length: 0, [[PrimitiveValue]]: ""}` – [Kevin Heidt](#) Nov 5 '14 at 18:43 ✎

- 5 @LuisAntonioPestana var value = myObj && myObj.property || '' will fall back to '' if either myObj or myObj.property is falsy. – Ates Goral Jan 3 '17 at 15:56 

66

```
function coalesce() {
  var len = arguments.length;
  for (var i=0; i<len; i++) {
    if (arguments[i] !== null && arguments[i] !== undefined) {
      return arguments[i];
    }
  }
  return null;
}

var xyz = {};
xyz.val = coalesce(null, undefined, xyz.val, 5);

// xyz.val now contains 5
```

this solution works like the SQL coalesce function, it accepts any number of arguments, and returns null if none of them have a value. It behaves like the C# ?? operator in the sense that "", false, and 0 are considered NOT NULL and therefore count as actual values. If you come from a .net background, this will be the most natural feeling solution.

edited Nov 19 '15 at 18:42



Сухой27


46.6k 6 52 99

answered Mar 8 '14 at 5:24



Brent Larsen

822 6 9

- 7 The for(var i in arguments) doesn't guarantee iteration order, according to [MDN](#). See also [Why is JavaScript's For...In loop not recommended for arrays?](#) and [Why is using "for...in" with array iteration such a bad idea?](#). – MvG Aug 17 '15 at 9:41 
- 12 Apologies for such a late addition, but I just wanted to note for completeness that this solution does have the caveat that it has no short-circuit evaluation; if your arguments are function calls then they will **all** be evaluated regardless of whether their value is returned, which differs from the behaviour of the logical OR operator, so is worth noting. – Haravikk Nov 15 '17 at 11:45

43

If || as a replacement of C#'s ?? isn't good enough in your case, because it swallows empty strings and zeros, you can always write your own function:

```
function $N(value, ifnull) {
  if (value === null || value === undefined)
    return ifnull;
  return value;
}

var whatIWant = $N(someString, 'Cookies!');
```

edited Nov 9 '17 at 16:42

answered Jan 24 '09 at 18:45



sth

173k 44 251 338

-
- 1 alert(null || "") still alerts an empty string, and I think I actually like that alert("" || 'blah') alerts blah rather than an empty string - good to know though! (+1)
– [Daniel Schaffer](#) Jan 24 '09 at 18:56
-
- 1 I think I might actually prefer defining a function that returns `false` if (strictly) null/undefined and `true` otherwise - using that with a logical or; it could be more readable than many nested functions calls. e.g. `$N(a) || $N(b) || $N(c) || d` is more readable than `$N($N($N(a, b), c), d)`. – [Bob](#) Nov 28 '13 at 6:13
-

Brent Larsen's solution is more generic – [Assimilator](#) Jan 15 '16 at 1:21

`if .. return .. else ... return` is the perfect case for a ternary. `return (value === null || value === void 0) ? ifnull : value;` – [Alex McMillan](#) May 15 '18 at 2:25

Yes, it is coming soon. See [proposal here](#) and [implementation status here](#).

29

It looks like this:

`x ?? y`

Example

```
const response = {
  settings: {
    nullValue: null,
    height: 400,
    animationDuration: 0,
    headerText: '',
    showSplashScreen: false
  }
}
```

```

    }
  };

  const undefinedValue = response.settings?.undefinedValue ?? 'some other default'; //
  result: 'some other default'
  const nullValue = response.settings?.nullValue ?? 'some other default'; // result: 'some
  other default'
  const headerText = response.settings?.headerText ?? 'Hello, world!'; // result: ''
  const animationDuration = response.settings?.animationDuration ?? 300; // result: 0
  const showSplashScreen = response.settings?.showSplashScreen ?? true; // result: false

```

answered May 7 '18 at 9:32



vaughan

3,524 4 37 50

Nobody has mentioned in here the potential for NaN, which--to me--is also a null-ish value. So, I thought I'd add my two-cents.

12

For the given code:

```

var a,
    b = null,
    c = parseInt('Not a number'),
    d = 0,
    e = '',
    f = 1
;

```

If you were to use the || operator, you get the first non-false value:

```
var result = a || b || c || d || e || f; // result === 1
```

If you use the typical coalesce method, [as posted here](#), you will get c, which has the value: NaN

```
var result = coalesce(a,b,c,d,e,f); // result.toString() === 'NaN'
```

Neither of these seem right to me. In my own little world of coalesce logic, which may differ from your world, I consider undefined, null, and NaN as all being "null-ish". So, I would expect to get back d (zero) from the coalesce method.

If anyone's brain works like mine, and you want to exclude NaN, then this method will accomplish that:

```
function coalesce() {
  var i, undefined, arg;

  for( i=0; i < arguments.length; i++ ) {
    arg = arguments[i];
    if( arg !== null && arg !== undefined
        && (typeof arg !== 'number' || arg.toString() !== 'NaN') ) {
      return arg;
    }
  }
  return null;
}
```

For those who want the code as short as possible, and don't mind a little lack of clarity, you can also use this as suggested by @impinball. This takes advantage of the fact that NaN is never equal to NaN. You can read up more on that here: [Why is NaN not equal to NaN?](#)

```
function coalesce() {
  var i, arg;

  for( i=0; i < arguments.length; i++ ) {
    arg = arguments[i];
    if( arg !== null && arg === arg ) { //arg === arg is false for NaN
      return arg;
    }
  }
  return null;
}
```

edited May 23 '17 at 12:10



Community ♦

1 1

answered May 26 '15 at 21:07



Kevin Nelson

6,417 3 23 36

Best practices - treat arguments as array-like, take advantage of NaN !== NaN (typeof + num.toString() === 'NaN' is redundant), store current argument in variable instead of arguments[i] . – [Isiah Meadows](#) Aug 17 '15 at 0:54

@impinball, your suggested edit doesn't work, it returns NaN instead of 0 (zero) from my test case. I could technically remove the !== 'number' check since I've already evaluated that it's not null or undefined, but this has the advantage of being very clear to anyone reading this code and the condition will work regardless of order. Your other suggestions do shorten the code slightly, so I will use those. – [Kevin Nelson](#) Aug 20 '15 at 16:40

- 2 @impinball, I found your bug in your suggested edit, you left it as arg !== arg, but you need it to be arg === arg ...then it works. However, that has the disadvantage of being very unclear as to what you are doing...requires comment in code to prevent being removed by the next person that goes through the code and thinks arg === arg is redundant...but I'll put it up anyway. – [Kevin Nelson](#) Aug 20 '15 at 16:48 ✎

Good catch. And by the way, that is a fast way of checking NaNs taking advantage of the fact `NaN !== NaN`. If you would like, you can explain that. – [Isiah Meadows](#) Aug 22 '15 at 7:47

for what it's worth, I agree that NaN is really just another null value. I'd update my answer, but you've got it covered. – [Brent Larsen](#) Feb 1 '18 at 16:33

Currently no support, but the JS-standardization process is working on it: <https://github.com/tc39/proposal-optional-chaining>

7

answered Jul 16 '17 at 21:17



[Elmar Jansen](#)

881 1 5 8

3 That's something slightly different: `a?.b` (access `a.b` if `a` exists), rather than `a ?? b` (if `a` exists, then `a`, else `b`). – [Steve Bennett](#) Aug 23 '18 at 3:38

beware of the JavaScript specific definition of null. there are two definitions for "no value" in javascript. 1. Null: when a variable is null, it means it contains no data in it, but the variable is already defined in the code. like this:

4

```
var myEmptyValue = 1;
myEmptyValue = null;
if ( myEmptyValue === null ) { window.alert('it is null'); }
// alerts
```

in such case, the type of your variable is actually Object. test it.

```
window.alert(typeof myEmptyValue); // prints Object
```

2. Undefined: when a variable has not been defined before in the code, and as expected, it does not contain any value. like this:

```
if ( myUndefinedValue === undefined ) { window.alert('it is undefined'); }
// alerts
```

if such case, the type of your variable is 'undefined'.

notice that if you use the type-converting comparison operator (`==`), JavaScript will act equally for both of these empty-values. to distinguish between them, always use the type-strict comparison operator (`===`).

edited Apr 8 '14 at 14:53



sshow

6,428

3

37

63

answered Jan 24 '09 at 18:35



farzad

7,349

5

27

38

- 1 Actually, null is a value. It's a special value of type Object. A variable being set to null means it contains data, the data being a reference to the null object. A variable can be defined with value undefined in your code. This is not the same as the variable not being declared. – [Ates Goral](#) Jan 24 '09 at 18:54

The actual difference between a variable being declared or not: `alert(window.test)/*undefined*/; alert("test" in window)/*false*/; window.test = undefined; alert(window.test)/*undefined*/; alert("test" in window)/*true*/; for (var p in window) { /*p can be "test"*/ }` – [Ates Goral](#) Jan 24 '09 at 18:59

- 1 however (a bit paradoxal) you can **define** a variable with the **undefined** value `var u = undefined;` – [Serge](#) Sep 11 '15 at 13:08



After reading your clarification, [@Ates Goral](#)'s answer provides how to perform the same operation you're doing in C# in JavaScript.

3

[@Gumbo](#)'s answer provides the best way to check for null; however, it's important to note the difference in `==` versus `===` in JavaScript *especially* when it comes to issues of checking for `undefined` and/or `null`.



There's a really good article about the difference in two terms [here](#). Basically, understand that if you use `==` instead of `===`, JavaScript will try to coalesce the values you're comparing and return what the result of the comparison *after* this coalescence.

edited Jan 24 '09 at 18:29

answered Jan 24 '09 at 18:23



Tom

9,126

4

44

60

One things that bugged me about that article (and Jash) is, an undefined `window.hello` property being evaluated to null for some reason. It should be undefined instead. Try it Firefox error console and see for yourself. – [Ates Goral](#) Jan 24 '09 at 19:20

