Detecting an undefined object property

Asked 11 years, 1 month ago Active 13 days ago Viewed 1.1m times

2710

What's the best way of checking if an object property in JavaScript is undefined?

javascript object undefined

508

edited Nov 9 '15 at 1:06
Ry- ♦
178k 42 366 382

asked Aug 26 '08 at 7:25
Matt Sheppard
70.4k 45 100 127

42 Answers

1 2 next

2580

Use:

```
if (typeof something === "undefined") {
    alert("something is undefined");
}
```

If an object variable which have some properties you can use same thing like this:

```
if (typeof my_obj.someproperties === "undefined"){
    console.log('the property is not available...'); // print into console
}
```

edited Mar 9 at 12:30
Ry- ♦

answered Jan 6 '09 at 12:27
Erwin

9 if something is null the it is defined (as null), but you can conjugate the too checks. The annoying detail of the above code is that you can't define a function to check it, well you can define the function... but try to use it. – neu-rah Jun 25 '12 at 19:20

5 @neu-rah why can't you write a function? why wouldn't something like this work? It seems to work for me. Is there a case I'm not considering? jsfiddle.net/djH9N/6 – Zack Sep 24 '12 at 19:01

7 @Zack Your tests for isNullorUndefined did not consider the case where you call isNullOrUndefined(f) and f is undeclared (i.e. where there is no "var f" declaration). – pnkfelix Feb 15 '13 at 15:08

98 Blah, thousands of votes now. This is the worst possible way to do it. I hope passers-by see this comment and decide to check… ahem… *other* answers. – Ry- ♦ May 14 '14 at 3:05

16 You can just use `obj !== undefined` now. `undefined` used to be mutable, like `undefined = 1234` what would cause interesting results. But after Ecmascript 5, it's not writable anymore, so we can use the simpler version. codereadability.com/how-to-check-for-undefined-in-javascript – Bruno Buccolo Mar 15 '16 at 20:50

873

I believe there are a number of incorrect answers to this topic. Contrary to common belief, "undefined" is **not** a keyword in JavaScript and can in fact have a value assigned to it.

Correct Code

The most robust way to perform this test is:

```
if (typeof myVar === "undefined")
```

This will always return the correct result, and even handles the situation where `myVar` is not declared.

Degenerate code. DO NOT USE.

```
var undefined = false;  // Shockingly, this is completely legal!
if (myVar === undefined) {
    alert("You have been misled. Run away!");
}
```

Additionally, `myVar === undefined` will raise an error in the situation where myVar is undeclared.


22.1k 30 136 264


21k 6 35 47

125 +1 for noting that myVar === undefined will raise an error if myVar was not declared – Enrique Dec 19 '11 at 18:27

67 I find the first justification given here for not using `=== undefined` bewildering. Yes, you can assign to `undefined`, but there is no legitimate reason to do so, and it's predictable that doing so may break your code. In C you can `#define true false`, and in Python you can assign to `True` and `False`, but people don't feel the need to design their code in those languages in such a way as to protect against the possibility of themselves deliberately sabotaging their own environment elsewhere in the code. Why is the possibility of assigning to `undefined` even worth considering here? – Mark Amery Jan 13 '13 at 19:05

18 in addition to Marks comments, I don't get this: "myVar === undefined will raise an error in the situation where myVar is undeclared." - why is this bad? Why would I *not* want to have an error if I'm referencing undeclared variables? – eis Aug 20 '13 at 15:12

5 Also keep in mind you can always do `void 0` to get the value that `undefined` points to. So you can do `if (myVar === void 0)`. the `0` isn't special, you can literally put any expression there. – Claudiu Oct 3 '13 at 17:45

27 In modern browsers (FF4+, IE9+, Chrome unknown), it's no longer possible to modify `undefined`. MDN: undefined – user247702 Feb 7 '14 at 14:09

200

Despite being vehemently recommended by many other answers here, `typeof` *is a bad choice*. It should never be used for checking whether variables have the value `undefined`, because it acts as a combined check for the value `undefined` and for whether a variable exists. In the vast majority of cases, you know when a variable exists, and `typeof` will just introduce the potential for a silent failure if you make a typo in the variable name or in the string literal `'undefined'`.

```
var snapshot = …;

if (typeof snaposhot === 'undefined') {
    //          ^
    // misspelled¹ – this will never run, but it won't throw an error!
}


var foo = …;

if (typeof foo === 'undefned') {
    //                   ^
    // misspelled – this will never run, but it won't throw an error!
}
```

So unless you're doing feature detection², where there's uncertainty whether a given name will be in scope (like checking `typeof module`

```
var foo = …;

if (foo === undefined) {
    ⋮
}
```

Some common misconceptions about this include:

- that reading an “uninitialized” variable (`var foo`) or parameter (`function bar(foo) { … }` , called as `bar()`) will fail. This is simply not true – variables without explicit initialization and parameters that weren’t given values always become `undefined` , and are always in scope.
- that `undefined` can be overwritten. There’s a lot more to this. `undefined` is not a keyword in JavaScript. Instead, it’s a property on the global object with the Undefined value. However, since ES5, this property has been *read-only* and *non-configurable*. No modern browser will allow the `undefined` property to be changed, and as of 2017 this has been the case for a long time. Lack of strict mode doesn’t affect `undefined` ’s behaviour either – it just makes statements like `undefined = 5` do nothing instead of throwing. Since it isn’t a keyword, though, you can *declare* variables with the name `undefined` , and those variables could be changed, making this once-common pattern:

  ```
  (function (undefined) {
      // …
  })()
  ```

  more dangerous than using the global `undefined` . If you have to be ES3-compatible, replace `undefined` with `void 0` – don’t resort to `typeof` . (`void` has always been a unary operator that evaluates to the Undefined value for any operand.)

With how variables work out of the way, it’s time to address the actual question: object properties. There is no reason to ever use `typeof` for object properties. The earlier exception regarding feature detection doesn’t apply here – `typeof` only has special behaviour on variables, and expressions that reference object properties are not variables.

This:

```
if (typeof foo.bar === 'undefined') {
    ⋮
}
```

is *always exactly equivalent* to this[3]:

```
}
```

and taking into account the advice above, to avoid confusing readers as to why you're using `typeof`, because it makes the most sense to use `===` to check for equality, because it could be refactored to checking a variable's value later, and because it just plain looks better, **you should always use `=== undefined` [3] here as well**.

Something else to consider when it comes to object properties is whether you really want to check for `undefined` at all. A given property name can be absent on an object (producing the value `undefined` when read), present on the object itself with the value `undefined`, present on the object's prototype with the value `undefined`, or present on either of those with a non-`undefined` value. `'key' in obj` will tell you whether a key is anywhere on an object's prototype chain, and `Object.prototype.hasOwnProperty.call(obj, 'key')` will tell you whether it's directly on the object. I won't go into detail in this answer about prototypes and using objects as string-keyed maps, though, because it's mostly intended to counter all the bad advice in other answers irrespective of the possible interpretations of the original question. Read up on object prototypes on MDN for more!

[1] unusual choice of example variable name? this is real dead code from the NoScript extension for Firefox.

[2] don't assume that not knowing what's in scope is okay in general, though. bonus vulnerability caused by abuse of dynamic scope: Project Zero 1225

[3] once again assuming an ES5+ environment and that `undefined` refers to the `undefined` property of the global object. substitute `void 0` otherwise.

edited Dec 14 '17 at 20:58

answered Feb 26 '14 at 21:17

Ry- ♦

178k 42 366 382

@BenjaminGruenbaum True but completely misleading. Any non-default context can define its own `undefined`, hiding the default one. Which for most practical purposes has the same effect as overwriting it. – blgt Mar 25 '14 at 14:32

21 @blgt That's paranoid and irrelevant for anything practical. Every context can override console.log, redefine Array prototype methods, and even override Function.prototype.call` hooking, and altering every time you call a function in JavaScript. Protecting against this is very paranoid and rather silly. Like I (and minitech) said, you can use `void 0` to compare against undefined but again - that's silly and overkill. – Benjamin Gruenbaum Mar 25 '14 at 14:41

17 I wish I had more than one upvote to give. This is the most correct answer. I really wanna stop seeing `typeof something === "undefined")` in code. – Simon Baumgardt-Wellander Feb 20 '18 at 19:37

@BenjaminGruenbaum For lazy programmers, `void 0` is (for once) both shorter *and* safer! That's a win in my book. – wizzwizz4 Apr 11 '18 at 18:31

3 This really should be the accepted answer. It is the most thorough and up-to-date. – Patrick Michaelsen Apr 10 at 20:40

151

- **undefined** means that the variable value has not been defined; it is not known what the value is.
- **null** means that the variable value is defined and set to null (has no value).

Marijn Haverbeke states, in his free, online book "Eloquent JavaScript" (emphasis mine):

> There is also a similar value, null, whose meaning is 'this value is defined, but it does not have a value'. The difference in meaning between undefined and null is mostly academic, and usually not very interesting. **In practical programs, it is often necessary to check whether something 'has a value'. In these cases, the expression something == undefined may be used, because, even though they are not exactly the same value, null == undefined will produce true.**

So, I guess the best way to check if something was undefined would be:

```
if (something == undefined)
```

Hope this helps!

Edit: In response to your edit, object properties should work the same way.

```
var person = {
    name: "John",
    age: 28,
    sex: "male"
};

alert(person.name); // "John"
alert(person.fakeVariable); // undefined
```

edited Apr 1 '10 at 12:41

answered Aug 26 '08 at 7:36

Pandincus

8,031 6 37 60

40 if (something == undefined) is better written as if (something === undefined) – Sebastian Rittau Nov 30 '09 at 9:47

57 It should be pointed out that this is not entirely safe. `undefined` is just a variable that can be re-assigned by the user: writing `undefined = 'a';` will cause your code to no longer do what you think it does. Using `typeof` is better and also works for variables (not just properties) that haven't been declared. – Gabe Moothart Apr 14 '10 at 15:18

5 This interpretation of the "Eloquent Javascript" comment is *backward*. If you really do just want to check for undefined, the suggested code will not work (it will also detect the condition defined but no value has been assined yet [i.e.null]).a null value. The suggested code "if (something == undefined) ..." checks for *both* undefined and null (no value set), i.e. it's interpreted as "if ((something is undefined) OR (something is null)) ..." What the author is saying is that often what you *really* want is to check for *both* undefined and null. – Chuck Kollars May 17 '12 at 22:35

120

What does this mean: **"undefined object property"**?

Actually it can mean two quite different things! First, it can mean *the property that has never been defined* in the object and, second, it can mean the *property that has an undefined value*. Let's look at this code:

```
var o = { a: undefined }
```

Is `o.a` undefined? Yes! Its value is undefined. Is `o.b` undefined? Sure! There is no property 'b' at all! OK, see now how different approaches behave in both situations:

```
typeof o.a == 'undefined' // true
typeof o.b == 'undefined' // true
o.a === undefined // true
o.b === undefined // true
'a' in o // true
'b' in o // false
```

We can clearly see that `typeof obj.prop == 'undefined'` and `obj.prop === undefined` are equivalent, and they do not distinguish those different situations. And `'prop' in obj` can detect the situation when a property hasn't been defined at all and doesn't pay attention to the property value which may be undefined.

So what to do?

1) You want to know if a property is undefined by either the first or second meaning (the most typical situation).

```
obj.prop === undefined // IMHO, see "final fight" below
```

2) You want to just know if object has some property and don't care about its value.

Notes:

- You can't check an object and its property at the same time. For example, this `x.a === undefined` or this `typeof x.a == 'undefined'` raises `ReferenceError: x is not defined` if x is not defined.
- Variable `undefined` is a global variable (so actually it is `window.undefined` in browsers). It has been supported since ECMAScript 1st Edition and since ECMAScript 5 it is **read only**. So in modern browsers it can't be *redefined to true* as many authors love to frighten us with, but this is still a true for older browsers.

Final fight: `obj.prop === undefined` vs `typeof obj.prop == 'undefined'`

Pluses of `obj.prop === undefined`:

- It's a bit shorter and looks a bit prettier
- The JavaScript engine will give you an error if you have misspelled `undefined`

Minuses of `obj.prop === undefined`:

- `undefined` can be overridden in old browsers

Pluses of `typeof obj.prop == 'undefined'`:

- It is really universal! It works in new and old browsers.

Minuses of `typeof obj.prop == 'undefined'`:

- `'undefned'` (*misspelled*) here is just a string constant, so the JavaScript engine can't help you if you have misspelled it like I just did.

Update (for server-side JavaScript):

Node.js supports the global variable `undefined` as `global.undefined` (it can also be used without the 'global' prefix). I don't know about other implementations of server-side JavaScript.

edited Sep 26 '15 at 19:01

answered Aug 8 '13 at 20:28

Konstantin Smolyanin

12.5k 6 41 38

says nothing about `undefined` as a member of `global`. Also neither `console.log(global);` nor `for (var key in global) { ... }` doesn't show *undefined* as a member of *global*. But test like `'undefined' in global` show the opposite. – Konstantin Smolyanin Sep 11 '13 at 10:53

3 It didn't need extra documentation since it's in the EcmaScript spec, which also says that `[[Enumerable]]` is false :-) – Bergi Sep 11 '13 at 11:00

4 Regarding `Minuses of typeof obj.prop == 'undefined'`, this can be avoided by writing as `typeof obj.prop == typeof undefined`. This also gives a very nice symmetry. – hlovdal Oct 24 '14 at 11:01

2 @hlovdal: That's totally pointless vs. `obj.prop === undefined`. – Ry- ♦ Apr 11 '18 at 21:38

When we are true to the question headline **„*Detecting* an undefined property“**, not true to the (different and much easier) question in the first sentence („check if undefined...“), you answer `if ('foo' in o)`… your answer is truly the first correct answer here. Pretty much everybody else just answers that sentence. – Frank Nocke Jun 11 '18 at 8:46

66

The issue boils down to three cases:

1. The object has the property and its value is not `undefined`.
2. The object has the property and its value is `undefined`.
3. The object does not have the property.

This tells us something I consider important:

There is a difference between an undefined member and a defined member with an undefined value.

But unhappily `typeof obj.foo` does not tell us which of the three cases we have. However we can combine this with `"foo" in obj` to distinguish the cases.

```
                            |  typeof obj.x === 'undefined' | !("x" in obj)
1.                  { x:1 } |  false                       | false
2.   { x : (function(){})() } |  true                        | false
3.                       {} |  true                        | true
```

Its worth noting that these tests are the same for `null` entries too

```
                            |  typeof obj.x === 'undefined' | !("x" in obj)
                 { x:null } |  false                       | false
```

value.

For example: I've just been refactoring a bunch of code that had a bunch of checks whether an object had a given property.

```
if( typeof blob.x != 'undefined' ) {  fn(blob.x); }
```

Which was clearer when written without a check for undefined.

```
if( "x" in blob ) { fn(blob.x); }
```

But as has been mentioned these are not exactly the same (but are more than good enough for my needs).

edited Jan 30 '14 at 2:49

answered Jun 8 '11 at 4:04

Michael Anderson
49k 7 100 151

9 Hi Michael. Great suggestion, and I think it does make things cleaner. One gotcha that I found, however, is when using the ! operator with "in". You have to say `if (!("x" in blob)) {}` with brackets around the in, because the ! operator has precedence over 'in'. Hope that helps someone. – Simon East Jun 15 '11 at 0:28

Sorry Michael, but this is incorrect, or at least misleading, in light of the original question. 'in' is not a sufficient way to test whether an object property has typeof undefined. For proof, please see this fiddle: jsfiddle.net/CsLKJ/4 – Tex Feb 25 '12 at 12:04

2 Those two code parts do a different thing! Consider and object given by `a = {b: undefined}` ; then `typeof a.b === typeof a.c === 'undefined'` but `'b' in a` and `!('c' in a)` . – mgol Sep 27 '12 at 14:07

3 +1. The OP doesn't make it clear whether the property exists and has the value *undefined*, or whether the property itself is undefined (i.e. doesn't exist). – RobG Apr 1 '14 at 1:12

I would suggest changing point (2.) in your first table to `{ x : undefined }` or at least add it as another alternative to (2.) in the table - I had to think for a moment to realize that point (2.) evaluates to `undefined` (although you mention that later on). – mucaho May 14 '15 at 16:32

43

```
if ( typeof( something ) == "undefined")
```

This worked for me while the others didn't.


Peter Ajtai
49.1k 10 110 134


Kevin
439 4 2

47 parens are unnecessary since typeof is an operator – aehlke Aug 10 '10 at 11:22

12 But they make it clearer what is being checked. Otherwise it might be read as `typeof (something == "undefined")` . – Abhi Beckert Sep 6 '12 at 0:28

If you need the parentheses, then you should learn operator precedence in JS: developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/… – Ian Mar 7 '14 at 17:17

11 Parenthesis are useful precisely because you do NOT need to learn operator precedence in JS, nor do you need to speculate whether future maintenance programmers will need to learn operator precedence in JS. – DaveWalley Apr 11 '14 at 14:48

27 Parenthesis are useful to clarify things. But in this case they just make the operator look like a function. No doubt this clarifies the intent of the programmer. But if you're unsure about operator precedence you should rather write it as `(typeof something) === "undefined"` . – Robert May 14 '14 at 18:31

39

I'm not sure where the origin of using `===` with `typeof` came from, and as a convention I see it used in many libraries, but the typeof operator returns a string literal, and we know that up front, so why would you also want to type check it too?

```
typeof x;                    // some string literal "string", "object", "undefined"
if (typeof x === "string") {   // === is redundant because we already know typeof
returns a string literal
if (typeof x == "string") {    // sufficient
```

edited Jun 29 '11 at 7:21


Simon East
37.2k 11 110 106

answered Sep 22 '10 at 14:20


Eric
423 4 2

Great point Eric. Is there a performance hit from checking type also? – Simon East Jun 29 '11 at 7:16

5 @Simon: quite the contrary - one could expect slight performance hit from avoiding coercion in '===' case. Quick and dirty test has shown '===' is 5% faster than '==' under FF5.0.1 – Antony Hatchkins Dec 18 '11 at 8:24

5 More thorough test has shown that under FF,IE and Chrome '==' is more or less faster than '===' (5-10%) and Opera doesn't make any difference at all: jsperf.com/triple-equals-vs-twice-equals/6 – Antony Hatchkins Dec 18 '11 at 9:55

7 `==` is one less character than `===` :) – svidgen Jun 28 '13 at 14:54

24

Crossposting my answer from related question How to check for "undefined" in JavaScript?

Specific to this question, see test cases with `someObject.<whatever>` .

Some scenarios illustrating the results of the various answers: http://jsfiddle.net/drzaus/UVjM4/

(Note that the use of `var` *for* `in` *tests make a difference when in a scoped wrapper)*

Code for reference:

```
(function(undefined) {
    var definedButNotInitialized;
    definedAndInitialized = 3;
    someObject = {
        firstProp: "1"
        , secondProp: false
        // , undefinedProp not defined
    }
    // var notDefined;

    var tests = [
        'definedButNotInitialized in window',
        'definedAndInitialized in window',
        'someObject.firstProp in window',
        'someObject.secondProp in window',
        'someObject.undefinedProp in window',
        'notDefined in window',

        '"definedButNotInitialized" in window',
        '"definedAndInitialized" in window',
        '"someObject.firstProp" in window',
        '"someObject.secondProp" in window',
        '"someObject.undefinedProp" in window',
        '"notDefined" in window',

        'typeof definedButNotInitialized == "undefined"',
        'typeof definedButNotInitialized === typeof undefined',
```

```
            'typeof definedAndInitialized == "undefined"',
            'typeof definedAndInitialized === typeof undefined',
            'definedAndInitialized === undefined',
            '! definedAndInitialized',
            '!! definedAndInitialized',

            'typeof someObject.firstProp == "undefined"',
            'typeof someObject.firstProp === typeof undefined',
            'someObject.firstProp === undefined',
            '! someObject.firstProp',
            '!! someObject.firstProp',

            'typeof someObject.secondProp == "undefined"',
            'typeof someObject.secondProp === typeof undefined',
            'someObject.secondProp === undefined',
            '! someObject.secondProp',
            '!! someObject.secondProp',

            'typeof someObject.undefinedProp == "undefined"',
            'typeof someObject.undefinedProp === typeof undefined',
            'someObject.undefinedProp === undefined',
            '! someObject.undefinedProp',
            '!! someObject.undefinedProp',

            'typeof notDefined == "undefined"',
            'typeof notDefined === typeof undefined',
            'notDefined === undefined',
            '! notDefined',
            '!! notDefined'
        ];

        var output = document.getElementById('results');
        var result = '';
        for(var t in tests) {
            if( !tests.hasOwnProperty(t) ) continue; // bleh

            try {
                result = eval(tests[t]);
            } catch(ex) {
                result = 'Exception--' + ex;
            }
            console.log(tests[t], result);
            output.innerHTML += "\n" + tests[t] + ": " + result;
        }
    })();
```

```
definedButNotInitialized in window: true
definedAndInitialized in window: false
someObject.firstProp in window: false
someObject.secondProp in window: false
someObject.undefinedProp in window: true
notDefined in window: Exception--ReferenceError: notDefined is not defined
"definedButNotInitialized" in window: false
"definedAndInitialized" in window: true
"someObject.firstProp" in window: false
"someObject.secondProp" in window: false
"someObject.undefinedProp" in window: false
"notDefined" in window: false
typeof definedButNotInitialized == "undefined": true
typeof definedButNotInitialized === typeof undefined: true
definedButNotInitialized === undefined: true
! definedButNotInitialized: true
!! definedButNotInitialized: false
typeof definedAndInitialized == "undefined": false
typeof definedAndInitialized === typeof undefined: false
definedAndInitialized === undefined: false
! definedAndInitialized: false
!! definedAndInitialized: true
typeof someObject.firstProp == "undefined": false
typeof someObject.firstProp === typeof undefined: false
someObject.firstProp === undefined: false
! someObject.firstProp: false
!! someObject.firstProp: true
typeof someObject.secondProp == "undefined": false
typeof someObject.secondProp === typeof undefined: false
someObject.secondProp === undefined: false
! someObject.secondProp: true
!! someObject.secondProp: false
typeof someObject.undefinedProp == "undefined": true
typeof someObject.undefinedProp === typeof undefined: true
someObject.undefinedProp === undefined: true
! someObject.undefinedProp: true
!! someObject.undefinedProp: false
typeof notDefined == "undefined": true
typeof notDefined === typeof undefined: true
notDefined === undefined: Exception--ReferenceError: notDefined is not defined
! notDefined: Exception--ReferenceError: notDefined is not defined
!! notDefined: Exception--ReferenceError: notDefined is not defined
```

edited May 23 '17 at 12:34 answered Jan 13 '13 at 17:43

18

I didn't see (hope I didn't miss it) anyone checking the object before the property. So, this is the shortest and most effective (though not necessarily the most clear):

```
if (obj && obj.prop) {
  // Do something;
}
```

If the obj or obj.prop is undefined, null, or "falsy", the if statement will not execute the code block. This is *usually* the desired behavior in most code block statements (in JavaScript).

edited Feb 5 '13 at 18:39

answered Sep 25 '12 at 18:41

Joe Johnson
1,631 12 19

2 If you want to know why this works: Javascript: Logical Operators and truthy / falsy – mb21 Feb 4 '13 at 16:57

if you want to assign the property to a variable if it's defined, not null and not falsey, else use some default value, you can use: `var x = obj && obj.prop || 'default';` – Stijn de Witt Nov 1 '15 at 2:27

I believe the question is for checking against undefined explicitly. Your condition check against all false values of JS. – NikoKyriakid Jul 20 '18 at 9:49

18

If you do

```
if (myvar == undefined )
{
    alert('var does not exists or is not initialized');
}
```

it will fail when the variable `myvar` does not exists, because myvar is not defined, so the script is broken and the test has no effect.

Because the window object has a global scope (default object) outside a function, a declaration will be 'attached' to the window object.

For example:

The global variable *myvar* is the same as *window.myvar* or *window['myvar']*

To avoid errors to test when a global variable exists, you better use:

```
if(window.myvar == undefined )
{
    alert('var does not exists or is not initialized');
}
```

The question if a variable really exists doesn't matter, its value is incorrect. Otherwise, it is silly to initialize variables with undefined, and it is better use the value false to initialize. When you know that all variables that you declare are initialized with false, you can simply check its type or rely on `!window.myvar` to check if it has a proper/valid value. So even when the variable is not defined then `!window.myvar` is the same for `myvar = undefined` or `myvar = false` or `myvar = 0` .

When you expect a specific type, test the type of the variable. To speed up testing a condition you better do:

```
if( !window.myvar || typeof window.myvar != 'string' )
{
    alert('var does not exists or is not type of string');
}
```

When the first and simple condition is true, the interpreter skips the next tests.

It is always better to use the instance/object of the variable to check if it got a valid value. It is more stable and is a better way of programming.

(y)

edited May 20 '16 at 18:11

answered Aug 12 '11 at 14:40

Codebeat

4,547 5 41 77

14

In the article *Exploring the Abyss of Null and Undefined in JavaScript* I read that frameworks like Underscore.js use this function:

```
function isUndefined(obj){
    return obj === void 0;
```

edited Nov 9 '14 at 12:22

Peter Mortensen
14.5k 19 89 118

answered Dec 19 '13 at 10:44

Marthijn
2,637 2 25 47

3 `void 0` is just a short way of writing `undefined` (since that's what *void* followed by any expression returns), it saves 3 charcters. It could also do `var a; return obj === a;` , but that's one more character. :-) – RobG Apr 1 '14 at 1:16

2 `void` is a reserved word, whereas `undefined` is not i.e. while `undefined` is equal to `void 0` by default, you can assign a value to `undefined` e.g. `undefined = 1234` . – Brian M. Hunt Sep 14 '15 at 13:08

`isUndefined(obj)` : 16 chars. `obj === void 0` : 14 chars. 'nough said. – Stijn de Witt Nov 1 '15 at 2:41

13

Simply anything is not defined in JavaScript, is **undefined**, doesn't matter if it's a property inside an **Object/Array** or as just a simple variable...

JavaScript has `typeof` which make it very easy to detect an undefined variable.

Simply check if `typeof whatever === 'undefined'` and it will return a boolean.

That's how the famous function `isUndefined()` in AngularJs v.1x is written:

```
function isUndefined(value) {return typeof value === 'undefined';}
```

So as you see the function receive a value, if that value is defined, it will return `false` , otherwise for undefined values, return `true` .

So let's have a look what gonna be the results when we passing values, including object properties like below, this is the list of variables we have:

```
var stackoverflow = {};
stackoverflow.javascipt = 'javascript';
var today;
var self = this;
var num = 8;
var list = [1, 2, 3, 4, 5];
var y = null;
```

```
isUndefined(stackoverflow); //false
isUndefined(stackoverflow.javascipt); //false
isUndefined(today); //true
isUndefined(self); //false
isUndefined(num); //false
isUndefined(list); //false
isUndefined(y); //false
isUndefined(stackoverflow.java); //true
isUndefined(stackoverflow.php); //true
isUndefined(stackoverflow && stackoverflow.css); //true
```

As you see we can check anything with using something like this in our code, as mentioned you can simply use `typeof` in your code, but if you are using it over and over, create a function like the angular sample which I share and keep reusing as following DRY code pattern.

Also one more thing, for checking property on an object in a real application which you not sure even the object exists or not, check if the object exists first.

If you check a property on an object and the object doesn't exist, will throw an error and stop the whole application running.

```
isUndefined(x.css);
VM808:2 Uncaught ReferenceError: x is not defined(…)
```

So simple you can wrap inside an if statement like below:

```
if(typeof x !== 'undefined') {
  //do something
}
```

Which also equal to isDefined in Angular 1.x...

```
function isDefined(value) {return typeof value !== 'undefined';}
```

Also other javascript frameworks like underscore has similar defining check, but I recommend you use `typeof` if you already not using any frameworks.

I also add this section from MDN which has got useful information about typeof, undefined and void(0).

Strict equality and undefined
You can use undefined and the strict equality and inequality operators to determine whether a variable has a value. In the following code, the variable x is not defined, and the if statement evaluates to true.

```
var x;
if (x === undefined) {
   // these statements execute
}
else {
   // these statements do not execute
}
```

Note: The strict equality operator rather than the standard equality operator must be used here, because x == undefined also checks whether x is null, while strict equality doesn't. null is not equivalent to undefined. See comparison operators for details.

Typeof operator and undefined
Alternatively, typeof can be used:

```
var x;
if (typeof x === 'undefined') {
   // these statements execute
}
```

One reason to use typeof is that it does not throw an error if the variable has not been declared.

```
// x has not been declared before
if (typeof x === 'undefined') { // evaluates to true without errors
   // these statements execute
}

if (x === undefined) { // throws a ReferenceError

}
```

> bound to the global object, so checking the existence of a variable in the global context can be done by checking the existence of a property on the global object (using the in operator, for instance).

> **Void operator and undefined**
>
> The void operator is a third alternative.

```
var x;
if (x === void 0) {
   // these statements execute
}

// y has not been declared before
if (y === void 0) {
   // throws a ReferenceError (in contrast to `typeof`)
}
```

more > here

edited Oct 2 at 2:51

answered May 24 '17 at 14:15

Alireza
62.7k 15 200 136

12

'*if (window.x) { }*' is error safe

Most likely you want `if (window.x)` . This check is safe even if x hasn't been declared (`var x;`) - browser doesn't throw an error.

Example: I want to know if my browser supports History API

```
if (window.history) {
    history.call_some_function();
}
```

window is an object which holds all global variables as its members, and it is legal to try to access a non-existing member. If **x** hasn't been declared or hasn't been set then `window.x` returns **undefined**. **undefined** leads to **false** when **if()** evaluates it.

edited Nov 23 '14 at 13:19

Peter Mortensen
14.5k 19 89 118

answered Feb 10 '14 at 16:26

DenisS
1,327 15 13

But what if you run in Node? `typeof history != 'undefined'` actually works in both systems. – Stijn de Witt Nov 1 '15 at 2:28

11

```
"propertyName" in obj //-> true | false
```

answered May 5 '14 at 0:13

sam
26.8k 2 35 35

11

Reading through this, I'm amazed I didn't see this. I have found multiple algorithms that would work for this.

Never Defined

If the value of an object was never defined, this will prevent from returning `true` if it is defined as `null` or `undefined` . This is helpful if you want true to be returned for values set as `undefined`

```
if(obj.prop === void 0) console.log("The value has never been defined");
```

Defined as undefined Or never Defined

If you want it to result as `true` for values defined with the value of `undefined` , or never defined, you can simply use `=== undefined`

```
if(obj.prop === undefined) console.log("The value is defined as undefined, or never
defined");
```

Defined as a falsy value, undefined,null, or never defined.

Commonly, people have asked me for an algorithm to figure out if a value is either falsy, `undefined` , or `null` . The following works.

```
if(obj.prop == false || obj.prop === null || obj.prop === undefined) {
    console.log("The value is falsy, null, or undefined");
}
```

answered Feb 15 '15 at 3:10

Travis
705 12 28

4 I think you can replace the last example with `if (!obj.prop)` – Stijn de Witt Nov 1 '15 at 2:24

@StijndeWitt, you can, I was pretty inexperienced when I wrote this, and my English seems to have been equally bad, nevertheless, there isn't anything *incorrect* in the answer – Travis Apr 7 '17 at 14:31

2 `var obj = {foo: undefined}; obj.foo === void 0 -> true` . How is that "never defined as `undefined` "? This is wrong. – Patrick Roberts Jun 15 '17 at 18:40

10

Compare with `void 0` , for terseness.

```
if (foo !== void 0)
```

It's not as verbose as `if (typeof foo !== 'undefined')`

answered Jan 2 '14 at 12:59

bevacqua
29.2k 43 145 271

3 But it will throw a ReferenceError if `foo` is undeclared. – daniel1426 Mar 7 '14 at 22:46

1 `'use strict'` ! – bevacqua Feb 15 '15 at 4:30

'undefined'){ /* define Promise */ } – gaperton Oct 4 '18 at 3:05

10

You can get an array all undefined with path using the following code.

```
function getAllUndefined(object) {

        function convertPath(arr, key) {
            var path = "";
            for (var i = 1; i < arr.length; i++) {

                path += arr[i] + "->";
            }
            path += key;
            return path;
        }


        var stack = [];
        var saveUndefined= [];
        function getUndefiend(obj, key) {

            var t = typeof obj;
            switch (t) {
                case "object":
                    if (t === null) {
                        return false;
                    }
                    break;
                case "string":
                case "number":
                case "boolean":
                case "null":
                    return false;
                default:
                    return true;
            }
            stack.push(key);
            for (k in obj) {
                if (obj.hasOwnProperty(k)) {
                    v = getUndefiend(obj[k], k);
                    if (v) {
                        saveUndefined.push(convertPath(stack, k));
```

```
            stack.pop();

        }

        getUndefiend({
            "": object
        }, "");
        return saveUndefined;
    }
```

jsFiddle link

edited Nov 9 '14 at 12:16
Peter Mortensen
14.5k 19 89 118

answered Oct 17 '11 at 11:22
Anoop
19.7k 9 50 70

While it won't affect the validity of your code, you've got a typo: `getUndefiend` should be `getUndefined` . – icktoofay May 14 '13 at 3:02

9

The solution is incorrect. In JavaScript,

```
null == undefined
```

will return true, because they both are "casted" to a boolean and are false. The correct way would be to check

```
if (something === undefined)
```

which is the identity operator...

edited Nov 9 '14 at 12:07
Peter Mortensen
14.5k 19 89 118

answered Aug 26 '08 at 12:38
Ricky
4,210 2 27 29

3 To be clear, `===` is type equality + (primitive equality | object identity), where primitives include strings. I think most people consider `'abab'.slice(0,2) === 'abab'.slice(2)` unintuitive if one considers `===` as the identity operator. – clacke Jul 30 '10 at 8:49

8

Here is my situation:

I am using the result of a REST call. The result should be parsed from JSON to a JavaScript object.

There is one error I need to defend. If the args to the rest call were incorrect as far as the user specifying the args wrong, the rest call comes back basically empty.

While using this post to help me defend against this, I tried this.

```
if( typeof restResult.data[0] === "undefined" ) { throw  "Some error"; }
```

For my situation, if restResult.data[0] === "object", then I can safely start inspecting the rest of the members. If undefined then throw the error as above.

What I am saying is that for my situation, all the suggestions above in this post did not work. I'm not saying I'm right and everyone is wrong. I am not a JavaScript master at all, but hopefully this will help someone.

edited Aug 22 '13 at 11:52

Fred
7,079 1 38 45

answered Aug 15 '13 at 13:56

wayneseymour
172 2 5

Your `typeof` guard doesn't actually guard against anything that a direct comparison couldn't handle. If `restResult` is undefined or undeclared, it'll still throw. – user8897421 Dec 18 '17 at 13:50

In your case you could more simply check if the array is empty: `if(!restResult.data.length) { throw "Some error"; }` – Headbank Feb 28 at 15:36

8

There is a nice & elegant way to assign a defined property to a new variable if it is defined or assign a default value to it as a fallback if it ´s undefined.

```
var a = obj.prop || defaultValue;
```

It's suitable if you have a function, which receives an additional config property:

```
var yourFunction = function(config){

   this.config = config || {};
   this.yourConfigValue = config.yourConfigValue || 1;
   console.log(this.yourConfigValue);

}
```

Now executing

```
yourFunction({yourConfigValue:2});
//=> 2

yourFunction();
//=> 1

yourFunction({otherProperty:5});
//=> 1
```

edited Feb 13 '17 at 13:17

answered Mar 3 '16 at 10:05

Marian Klühspies

6,717 10 58 88

7

All the answers are incomplete. This is the right way of knowing that there is a property 'defined as undefined' :

```
var hasUndefinedProperty = function hasUndefinedProperty(obj, prop){
  return ((prop in obj) && (typeof obj[prop] == 'undefined')) ;
} ;
```

Example:

```
var a = { b : 1, e : null } ;
a.c = a.d ;

hasUndefinedProperty(a, 'b') ; // false : b is defined as 1
hasUndefinedProperty(a, 'c') ; // true : c is defined as undefined
hasUndefinedProperty(a, 'd') ; // false : d is undefined
```

```
delete a.c ;
hasUndefinedProperty(a, 'c') ; // false : c is undefined
```

Too bad that this been the right answer is buried in wrong answers >_<

So, for anyone who pass by, I will give you undefineds for free!!

```
var undefined ; undefined ; // undefined
({}).a ;                    // undefined
[].a ;                      // undefined
''.a ;                      // undefined
(function(){}()) ;          // undefined
void(0) ;                   // undefined
eval() ;                    // undefined
1..a ;                      // undefined
/a/.a ;                     // undefined
(true).a ;                  // undefined
```

edited Jun 18 '14 at 6:14

answered Jun 18 '14 at 5:21

Juan Garcia
501 5 18

7

Going through the comments, for those who want to check both is it undefined or its value is null:

```
//Just in JavaScript
var s; // Undefined
if (typeof s == "undefined" || s === null){
    alert('either it is undefined or value is null')
}
```

If you are using jQuery Library then `jQuery.isEmptyObject()` will suffice for both cases,

```
var s; // Undefined
jQuery.isEmptyObject(s); // Will return true;

s = null; // Defined as null
jQuery.isEmptyObject(s); // Will return true;
```

```
    alert('Either variable:s is undefined or its value is null');
} else {
     alert('variable:s has value ' + s);
}

s = 'something'; // Defined with some value
jQuery.isEmptyObject(s); // Will return false;
```

edited Nov 9 '14 at 12:26

Peter Mortensen
14.5k 19 89 118

answered Jul 8 '14 at 8:19

Angelin Nadar
5,860 7 33 50

jQuery will also take care of any cross-browser compatibility issues with the different JavaScript APIs. – Henry Heleine Dec 9 '14 at 22:12

7

If you are using Angular:

```
angular.isUndefined(obj)
angular.isUndefined(obj.prop)
```

Underscore.js:

```
_.isUndefined(obj)
_.isUndefined(obj.prop)
```

answered Dec 14 '14 at 22:35

Vitalii Fedorenko
82.5k 21 136 110

2 How do I add `1` to variable `x` ? Do I need Underscore or jQuery? (amazing that people will use libraries for even the most elementary operations such as a `typeof` check) – Stijn de Witt Nov 1 '15 at 2:43

I use `if (this.variable)` to test if it is defined. Simple `if (variable)` , recommended above, fails for me. It turns out that it works only

```
if (this.abc) alert("defined"); else alert("undefined");

abc = "abc";
if (this.abc) alert("defined"); else alert("undefined");
```

Run code snippet Expand snippet

It first detects that variable `abc` is undefined and it is defined after initialization.

edited May 23 '17 at 11:47 Community ♦ 1 1

answered Mar 6 '15 at 15:58 Val 1

5

I provide three ways here for those who expect weird answers:

```
function isUndefined1(val) {
    try {
        val.a;
    } catch (e) {
        return /undefined/.test(e.message);
    }
    return false;
}
function isUndefined2(val) {
    return !val && val+'' === 'undefined';
}
function isUndefined3(val) {
    const defaultVal={};
    return ((input=defaultVal)=>input===defaultVal)(val);
}
function test(func){
    console.group(`test start :`+func.name);
    console.log(func(undefined));
    console.log(func(null));
    console.log(func(1));
    console.log(func("1"));
```

```
    console.groupEnd();
}
test(isUndefined1);
test(isUndefined2);
test(isUndefined3);
```

Run code snippet Expand snippet

isUndefined1:

Try to get a property of the input value, check the error message if it exists. If the input value is undefined, the error message would be *Uncaught TypeError: Cannot read property 'b' of undefined*

isUndefined2:

Convert input value to string to compare with `"undefined"` and ensure it's negative value.

isUndefined3:

In js, optional parameter works when the input value is exactly `undefined` .

edited Sep 25 '18 at 14:57

answered Oct 28 '17 at 9:24

blackmiaool

4,637 2 12 33

4

```
function isUnset(inp) {
  return (typeof inp === 'undefined')
}
```

Returns false if variable is set, and true if is undefined.

Then use:

```
if (isUnset(var)) {
```

edited Jun 15 '11 at 1:43

Simon East
37.2k 11 110 106

answered Jul 12 '10 at 20:54

Rixius
1,145 3 17 33

5 No. Don't do this. It only takes a very simple test to prove that you cannot meaningfully wrap a `typeof` test in a function. Amazing that 4 people upvoted this. -1. – Stijn de Witt Nov 1 '15 at 2:39

I would like to show you something I'm using in order to protect the `undefined` variable:

4

```
Object.defineProperty(window, 'undefined', {});
```

This forbids anyone to change the `window.undefined` value therefore destroying the code based on that variable. If using `"use strict"`, anything trying to change its value will end in error, otherwise it would be silently ignored.

edited Nov 9 '14 at 12:27

Peter Mortensen
14.5k 19 89 118

answered Oct 9 '14 at 8:09

Seti
1,017 9 20

you can also use Proxy, it will work with nested calls, but will require one extra check:

4

```
function resolveUnknownProps(obj, resolveKey) {
  const handler = {
    get(target, key) {
      if (
        target[key] !== null &&
        typeof target[key] === 'object'
      ) {
        return resolveUnknownProps(target[key], resolveKey);
      } else if (!target[key]) {
        return resolveUnknownProps({ [resolveKey]: true }, resolveKey);
      }

      return target[key];
    },
`
```

```
}

const user = {}

console.log(resolveUnknownProps(user, 'isUndefined').personalInfo.name.something.else);
// { isUndefined: true }
```

so you will use it like:

```
const { isUndefined } = resolveUnknownProps(user,
'isUndefined').personalInfo.name.something.else;
if (!isUndefined) {
  // do someting
}
```

edited Mar 21 '18 at 17:35

answered Mar 21 '18 at 16:50

Sarkis Arutiunian
706 9 28

4

From lodash.js.

```
var undefined;
function isUndefined(value) {
  return value === undefined;
}
```

It creates a LOCAL variable named `undefined` which is initialized with the default value -- the real `undefined`, then compares `value` with the variable `undefined`.

Update 9/9/2019

I found lodash updated its implementation. See my issue and the code.

To be bullet-proof, simply use:

edited Sep 9 at 22:01

answered Jan 14 '16 at 5:43

lzl124631x

1,437 14 34

1 2 next

protected by Starx Apr 25 '12 at 8:45

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?