Is there a dedicated function to check null and undefined in TypeScript?

Asked 4 years, 4 months ago Active 16 days ago Viewed 318k times

278

37

Since TypeScript is strongly-typed, simply using `if () {}` to check null and undefined doesn't sound right.

Does TypeScript has dedicated function or syntax sugar for this?

typescript

asked Mar 10 '15 at 23:23

David Liu
4,455 7 27 34

5 `Since TypeScript is strongly-typed` I couldn't find this in it's docs and I have doubts about it... – pawciobiel Aug 31 '15 at 14:01

2 Recommend to read up on the latest non-nullable types , this is Typescript 2 , but already in beta as of today. [Non-nullable types #7140] (github.com/Microsoft/TypeScript/pull/7140) – RyBolt Aug 5 '16 at 13:10

1 TypeScript has no dedicated functions to do anything. It's a typing system and a transpiler, not a library. – user663031 Aug 9 '17 at 8:18

18 Answers

293

Using a juggling-check, you can test both `null` and `undefined` in one hit:

```
if (x == null) {
```

If you use a strict-check, it will only be true for values set to `null` and won't evaluate as true for undefined variables:

```
if (x === null) {
```

```
var a: number;
var b: number = null;

function check(x, name) {
    if (x == null) {
        console.log(name + ' == null');
    }

    if (x === null) {
        console.log(name + ' === null');
    }

    if (typeof x === 'undefined') {
        console.log(name + ' is undefined');
    }
}

check(a, 'a');
check(b, 'b');
```

Output

> "a == null"
>
> "a is undefined"
>
> "b == null"
>
> "b === null"

edited Jun 13 '18 at 7:59

answered Mar 11 '15 at 10:38

Fenton
165k 46 302 329

32 What is "juggling-check"? – kolobok Aug 1 '16 at 12:55

11 @akapelko it is where the type is juggled (i.e. "can we make this type a boolean"). So an empty string is treated as a boolean false, for example. A common bug when juggling is: `"false" == false` a non-empty string like "false" evaluates to `true`. – Fenton Aug 1 '16 at 13:34

6 This is due to JS's 'type coercion'. – Astravagrant Jan 16 '17 at 13:19

2 @JonGunter that would be true of truthy/falsey `if(x)` style checks, but not `if(x == null)` , which only catches `null` and `undefined` . Check it using `var c: number = 0; check(c, 'b');` it is not "nully", `null` , or `undefined` . – Fenton Aug 9 '17 at 18:47

190

```
if( value ) {
}
```

will evaluate to `true` if `value` is not:

- `null`
- `undefined`
- `NaN`
- empty string `''`
- `0`
- `false`

typescript includes javascript rules.

edited Apr 16 '18 at 15:09
kingdaro
5,788 1 17 29

answered May 17 '17 at 6:50
Ramazan Sağır
2,118 1 9 10

4 What if value is of boolean type? – ianstigator Oct 6 '17 at 22:56

yes, return type is boolean – Ramazan Sağır Oct 11 '17 at 7:57

can you combine two variables eg. if(value1 && value2) to check if both of them are undefined ? – Akshayraj Kore Nov 17 '17 at 16:52

4 @RamazanSağır yeah thanks I know that, but the fact is 0 value is something valid that I can have, the only check I want to do is that the variable is neither null or undefined. I have read that I can do it by using val != null (the != instead of !== also checks undefined value) – Alex Feb 27 '18 at 10:31

3 This solution will not work if the tslint rule - "strict-boolean-expressions" is enabled. – ip_x Sep 24 '18 at 11:43

35

```
let a;
let b = null;
let c = "";
var output = "";

if (a == null) output += "a is null or undefined\n";
if (b == null) output += "b is null or undefined\n";
if (c == null) output += "c is null or undefined\n";
if (a != null) output += "a is defined\n";
if (b != null) output += "b is defined\n";
if (c != null) output += "c is defined\n";
if (a) output += "a is defined (2nd method)\n";
if (b) output += "b is defined (2nd method)\n";
if (c) output += "c is defined (2nd method)\n";

console.log(output);
```

gives:

```
a is null or undefined
b is null or undefined
c is defined
```

so:

- checking if (a == null) is right to know if a is null or undefined
- checking if (a != null) is right to know if a is defined
- checking if (a) is wrong to know if a is defined

edited Jan 11 '17 at 0:57

answered Oct 21 '16 at 11:25

Juangui Jordán

1,543 19 21

Why would you use the TypeScript playground for this? Nothing here has anything to do with TypeScript. – user663031 Aug 10 '17 at 4:37

10 Because the question was related to Typescript, I was trying to test different proposed solutions against the Typescript transpiler. – Juangui Jordán Aug 11 '17 at 7:51

4 The TS transpiler would not transform any of this code at all. – user663031 Aug 11 '17 at 8:32

> Does TypeScript has dedicated function or syntax sugar for this

31

No. I just do `something == null` same as JavaScript.

edited Oct 11 '16 at 4:40

answered Mar 10 '15 at 23:42

basarat
150k 28 282 386

1 I like doing two equals `myVar == null`. Just another option. – David Sherret Mar 11 '15 at 2:09

25 `== null` is the correct way to test for null & undefined. `!!something` is a useless coercion in a conditional in JS (just use `something`). `!!something` will also coerce 0 and '' to false, which is not what you want to do if you are looking for null/undefined. – C Snover Mar 11 '15 at 3:04

26

I think this answer needs an update, check the edit history for the old answer.

Basically, you have three deferent cases null, undefined, and undeclared, see the snippet below.

```
// bad-file.ts
console.log(message)
```

You'll get an error says that variable `message` is undefined (aka undeclared), of course, the Typescript compiler shouldn't let you do that but REALLY nothing can prevent you.

```
// evil-file.ts
// @ts-gnore
console.log(message)
```

The compiler will be happy to just compile the code above. So, if you're sure that all variables are declared you can simply do that

```
if ( message != null ) {
    // do something with the message
}
```

```
if ( typeof(message) !== 'undefined' && message !== null ) {
    // message variable is more than safe to be used.
}
```

Note: the order here `typeof(message) !== 'undefined' && message !== null` is very important you have to check for the `undefined` state first atherwise it will be just the same as `message != null` , thanks @Jaider.

edited Jul 10 at 18:10

answered Jul 10 '18 at 7:39
Ahmed M.Kamal
699 1 7 19

4 M. Kamal if something = 0, your verification with !something will give you problems. – arturios Nov 21 '18 at 10:35

1 @arturios can you please give me an example!! – Ahmed M.Kamal Nov 21 '18 at 13:24

2 @arturios But 0 is already a falsy value in JavaScript !! so what is the point here? – Ahmed M.Kamal Nov 21 '18 at 17:01

1 @Al-un nope, see it in action here – Ahmed M.Kamal Jan 17 at 23:45

1 the updated version is wrong. The first thing to check should be undefined... like: `if(typeof something !== 'undefined' && something !== null) {...}` – Jaider Jul 10 at 17:17

14

```
if(data){}
```

it's mean !data

- null
- undefined
- false
- ....

answered Jan 20 '17 at 12:18
artemitSoft
216 5 14

can you combine two variables eg. if(value1 && value2) to check if both of them are undefined ? – Akshayraj Kore Nov 17 '17 at 16:54

13

You may want to try

```
if(!!someValue)
```

with `!!` .

Explanation

The first `!` will turn your expression into a `boolean` value.

Then `!someValue` is `true` if `someValue` is *falsy* and `false` if `someValue` is *truthy*. This might be confusing.

By adding another `!` , the expression is now `true` if `someValue` is *truthy* and `false` if `someValue` is *falsy*, which is much easier to manage.

Discussion

Now, why do I bother myself with `if (!!someValue)` when something like `if (someValue)` would have give me the same result?

Because `!!someValue` is precisely a boolean expression, whereas `someValue` could be absolutely anything. This kind of expression will now alow to write functions (and God we need those) like:

```
isSomeValueDefined(): boolean {
  return !!someValue
}
```

instead of:

```
isSomeValueDefined(): boolean {
  if(someValue) {
    return true
  }
  return false
}
```

answered Apr 12 '18 at 8:52
avi.elkharrat
1,994 1 20 30

so, if someValue is 'false'(with string type), then !!someValue is false(boolean type)? – paul cheung Feb 12 at 3:47

I guess you may say so.This technic is precisely used to avoid having this kind of confusion. I hope you like it! – avi.elkharrat Feb 12 at 8:41

but what confused me is !!'false' equals true. Just because of this case, i can not use this technic. – paul cheung Feb 12 at 9:07

`!!'false'` is in deed `true` because `'false'` is a valid string – avi.elkharrat Feb 12 at 9:43

so this technic can not cover this case, or is there a workaround solution? – paul cheung Feb 12 at 9:56

11

For `Typescript 2.x.x` you should do it in a following way:

tl;dr

```
function isDefined<T>(value: T | undefined | null): value is T {
  return <T>value !== undefined && <T>value !== null;
}
```

Why?

In this way `isDefined()` will respect variable's type and the following code would know take this check in account.

Example 1 - basic check:

```
function getFoo(foo: string): void {
  //
}

function getBar(bar: string| undefined) {
  getFoo(bar); //ERROR: "bar" can be undefined
  if (isDefined(bar)) {
    getFoo(bar); // Ok now, typescript knows that "bar' is defined
```

Example 2 - types respect:

```
function getFoo(foo: string): void {
  //
}

function getBar(bar: number | undefined) {
  getFoo(bar); // ERROR: "number | undefined" is not assignable to "string"
  if (isDefined(bar)) {
    getFoo(bar); // ERROR: "number" is not assignable to "string", but it's ok - we know
it's number
  }
}
```

edited Feb 19 at 12:24

answered Aug 30 '18 at 12:57
Sergei Panfilov
6,347 9 45 76

5

If you are using TypeScript, it is a better approach to let the compiler check for nulls and undefineds (or the possibility thereof), rather than checking for them at run-time. (If you do want to check at run-time, then as many answers indicate, just use `value == null`).

Use the compile option `strictNullChecks` to tell the compiler to choke on possible null or undefined values. If you set this option, and then there is a situation where you **do** want to allow null and undefined, you can define the type as `Type | null | undefined` .

answered May 23 '17 at 7:56
user663031

5

If you want to pass `tslint` without setting `strict-boolean-expressions` to `allow-null-union` or `allow-undefined-union` , you need to use `isNullOrUndefined` from `node` 's `util` module or roll your own:

```
// tslint:disable:no-null-keyword
export const isNullOrUndefined =
  <T>(obj: T | null | undefined): obj is null | undefined => {
    return typeof obj === "undefined" || obj === null;
```

Not exactly syntactic sugar but useful when your tslint rules are strict.

answered Nov 14 '17 at 21:01
Graeme Wicksted
1,216 1 12 19

0

All,

The answer with the most votes, does not really work if you are working with an object. In that case, if a property is not present, the check will not work. And that was the issue in our case: see this sample:

```
var x =
{ name: "Homer", LastName: "Simpson" };

var y =
{ name: "Marge"} ;

var z =
{ name: "Bart" , LastName: undefined} ;

var a =
{ name: "Lisa" , LastName: ""} ;

var hasLastNameX = x.LastName != null;
var hasLastNameY = y.LastName != null;
var hasLastNameZ = z.LastName != null;
var hasLastNameA = a.LastName != null;



alert (hasLastNameX + ' ' + hasLastNameY + ' ' + hasLastNameZ + ' ' + hasLastNameA);

var hasLastNameXX = x.LastName !== null;
var hasLastNameYY = y.LastName !== null;
var hasLastNameZZ = z.LastName !== null;
var hasLastNameAA = a.LastName !== null;

alert (hasLastNameXX + ' ' + hasLastNameYY + ' ' + hasLastNameZZ + ' ' + hasLastNameAA);
```

Outcome:

```
true , false, false , true (in case of !=)
true , true, true, true (in case of !==) => so in this sample not the correct answer
```

plunkr link: https://plnkr.co/edit/BJpVHD95FhKlpHp1skUE

answered Oct 10 '17 at 11:11
Ben Croughs
1,431 14 25

This is not good test. None of those values are *strictly* `null` . Try this: plnkr.co/edit/NfiVnQNes1p8PvXd1fCG?p=preview – simonhamp Nov 9 '17 at 12:46

0

A faster and shorter notation for `null` checks can be:

```
value == null ? "UNDEFINED" : value
```

This line is equivalent to:

```
if(value == null) {
       console.log("UNDEFINED")
} else {
    console.log(value)
}
```

Especially when you have a lot of `null` check it is a nice short notation.

answered Oct 26 '18 at 7:46
Harry Stylesheet
1

I had this issue and some of the answer work just fine for `JS` but not for `TS` here is the reason.

```
if(couldBeNullOrUndefined == null) {
  console.log('null OR undefined', couldBeNullOrUndefined);
} else {
  console.log('Has some value', couldBeNullOrUndefined);
}
```

That is all good as JS has no Types

```
//TS
let couldBeNullOrUndefined?: string | null; // THIS NEEDS TO BE TYPED AS undefined ||
null || Type(string)

if(couldBeNullOrUndefined === null) { // TS should always use strict-check
  console.log('null OR undefined', couldBeNullOrUndefined);
} else {
  console.log('Has some value', couldBeNullOrUndefined);
}
```

In TS if the variable wasn't defined with `null` when you try to check for that `null` the `tslint` | compiler will complain.

```
//tslint.json
...
"triple-equals":[true],
...
```

```
 let couldBeNullOrUndefined?: string; // to fix it add | null

 Types of property 'couldBeNullOrUndefined' are incompatible.
      Type 'string | null' is not assignable to type 'string | undefined'.
        Type 'null' is not assignable to type 'string | undefined'.
```

answered Feb 21 at 1:29

T04435
3,238 28 38

Late to join this thread but I find this JavaScript hack very handy in checking whether a value is undefined

```
}
```

answered Apr 9 at 16:03
Shahid Manzoor Bhat
364 1 6 21

0

Usually I do the juggling-check as Fenton already discussed. To make it more readable, you can use isNil from ramda.

```
import * as isNil from 'ramda/src/isNil';

totalAmount = isNil(totalAmount ) ? 0 : totalAmount ;
```

answered Jun 21 at 0:23
Neo
73 1 1

-1

you can use

```
if(x === undefined)
```

edited May 23 '17 at 15:53
Julian
15.3k 7 59 94

answered May 23 '17 at 7:23
akshay reddy
31 1

-1

Since TypeScript is a typed superset of ES6 JavaScript. And lodash are a library of javascript.

Using lodash to checks if value is null or undefined can be done using `_.isNil()` .

```
_.isNil(value)
```

value (*): The value to check.

Returns

(boolean): Returns true if value is nullish, else false.

Example

```
_.isNil(null);
// => true

_.isNil(void 0);
// => true

_.isNil(NaN);
// => false
```

Link

Lodash Docs

answered Aug 2 '18 at 11:42
7e2e63de
17 1

1 Why this method are -2 ? Lodash is not good with type script ? – Thomas Poignant Jun 13 at 14:28

-2

I always write it like this:

```
var foo:string;

if(!foo){
   foo="something";
}
```

AlexB
472 2 5 13

16 Wouldn't work for numbers because `0` also passes the `!foo` test. – hasen May 18 '16 at 17:58

9 Does not work for booleans either, where `undefined` is different than `false` . This is very common with optional boolean function parameters, where you should use the common JavaScript approach: `function fn(flag?: boolean) { if (typeof flag === "undefined") flag = true; /* set default value */ }` – Gingi May 27 '16 at 18:00

Seems to work ok for booleans: `var isTrue; if(isTrue)//skips, if(!isTrue)// enters if(isTrue === undefined)//enters` . Also tried it in typescript with `var isTrue:boolean` which was undefined, and the same if checks. @Gingi, is there something different about what you tried and what I tried? – Drenai Aug 7 '16 at 19:10