How to check the object type on runtime in TypeScript?

Asked 3 years, 9 months ago Active 3 days ago Viewed 65k times

Ask Question

49

I'm trying to find a way to pass an object to function in and check it type in a runtime. This is a pseudo code:

```
func(obj:any){
  if(typeof obj === "A"){
    // do something
  }
  else if(typeof obj === "B"{
    //do something else
  }

}
 a:A;
 b:B;
 func(a);
```

But typeof is always return "object" and I could not find a way to get the real type of "a" or "b". The instanceof did not work either and return the same. Any idea how to do it in a TypeScript?

Thank you for your help!!!

Share Improve this question Follow

asked May 19 '17 at 20:02 Eden1971 529 ●1 ●4 ●7

Add a comment

7 Answers

Active Oldest Votes

74

> **Edit**: I want to point out to people coming here from searches that this question is specifically dealing with non-class types, ie object shapes as defined by `interface` or `type` alias. For class types you can use JavaScript's `instanceof` to determine the class an instance comes from, and TypeScript will narrow the type in the type-checker automatically.

Types are stripped away at compile-time and do not exist at runtime, so you can't check the type at runtime.

What you can do is check that the shape of an object is what you expect, and TypeScript can assert the type at compile time using a user-defined type guard that returns true (annotated return type is a "type predicate" of the form `arg is T`) if the shape matches your expectation:

```
interface A {
  foo: string;
}

interface B {
  bar: number;
}

function isA(obj: any): obj is A {
  return obj.foo !== undefined
}

function isB(obj: any): obj is B {
  return obj.bar !== undefined
}

function func(obj: any) {
  if (isA(obj)) {
    // In this block 'obj' is narrowed to type 'A'
    obj.foo;
  }
  else if (isB(obj)) {
    // In this block 'obj' is narrowed to type 'B'
    obj.bar;
  }
}
```

Example in Playground

How deep you take the type-guard implementation is really up to you, it only needs to return true or false. For example, as Carl points out in his answer, the above example only checks that expected properties are defined (following the example in the docs), not that they are assigned the expected type. This can get tricky with nullable types and nested objects, it's up to you to determine how detailed to make the shape check.

Share Improve this answer Follow edited Sep 26 '19 at 15:03 answered May 19 '17 at 20:32 Aaron Beall 36.9k ●19 ●70 ●94

check this: aliolicode.com/2016/04/23/type-checking-typescript Please be shore, you see this line also: console.log(john instanceof Person); // true ...cheers! – peter70 Aug 3 '17 at 13:19

@peter70 That only works with class instances, not other types (like interfaces). OP mentioned `instanceof` didn't work so I assume he has a non-class instance object. – Aaron Beall Dec 3 '17 at 2:13

It is possible when transporting type information into runtime code. E.g. with a custom transformer in TS > 2.4. – Christian Jan 10 '19 at 11:56

Ugh, I wish there were a more elegant, less verbose way to do this, such as being able to assert the type in the if statement itself without the need for all these helper functions. – 55 Cancri Apr 12 '20 at 13:03

Why cant typescript allow `if (isA(obj)): obj is A { ... }` ? – 55 Cancri Apr 12 '20 at 13:05

Add a comment

22

Expanding on Aaron's answer, I've made a transformer that generates the type guard functions at compile time. This way you don't have to manually write them.

For example:

```
import { is } from 'typescript-is';

interface A {
  foo: string;
}

interface B {
  bar: number;
}

if (is<A>(obj)) {
  // obj is narrowed to type A
}

if (is<B>(obj)) {
  // obj is narrowed to type B
}
```

You can find the project here, with instructions to use it:

https://github.com/woutervh-/typescript-is

Share Improve this answer Follow answered Sep 19 '18 at 7:21 user7132587 367 ●3 ●5

1 `typescript-is` is not good. they force me to use `ttypescript`. – ian park Jan 9 '19 at 16:17

@ianpark actually you are not forced to use `ttypescript`, you can also programmatically compile your project with the `typescript` API and configure the transformers yourself. `ttypescript` is the *recommended* way because it does this for you. When you use a transformer you have no other choice at the moment. And what's wrong with `ttypescript` anyway? ;-) – user7132587 Jan 11 '19 at 9:14

yes. you're right. `ttypescript` is recommended way and it is good solution. My opinion is for whom do not want to add another compiler. If you use `typescript-is`, you should add another compiler or write own compile logic. It will become another complexity. – ian park Jan 15 '19 at 4:33

Add a comment

4

The OPs question was "I'm trying to find a way to pass an object to function in and check it type in a runtime".

Since a class instance is just an object the correct answer is to use a class instance and instanceof when runtime type checking is needed, use interface when not.

In my codebase, I will typically have a class which implements an interface and the interface is used during compilation for pre-compile time type safety, while classes are used to organize my code as well as do runtime type checks in typescript.

Works because routerEvent is an instance of NavigationStart class

```
if (routerEvent instanceof NavigationStart) {
  this.loading = true;
}

if (routerEvent instanceof NavigationEnd ||
  routerEvent instanceof NavigationCancel ||
  routerEvent instanceof NavigationError) {
  this.loading = false;
}
```

Will not work

```
// Must use a class not an interface
export interface IRouterEvent { ... }
// Fails
expect(IRouterEvent instanceof NavigationCancel).toBe(true);
```

Will not work

```
// Must use a class not a type
export type RouterEvent { ... }
// Fails
expect(IRouterEvent instanceof NavigationCancel).toBe(true);
```

As you can see by the code above, classes are used to compare the instance to the types NavigationStart|Cancel|Error.

Using instanceof on a Type or Interface is not possible since the ts compiler strips away these attributes during its compilation process and prior to being interpreted by JIT or AOT. Classes are a great way to create a type which can be used precompilation as well as during the JS runtime.

Share Improve this answer Follow answered Feb 22 '19 at 3:33 Alpha G33k 1,367 ●1 ●7 ●18

Add a comment

1

I've been playing around with the answer from Aaron and think it would be better to test for typeof instead of just undefined, like this:

```
interface A {
  foo: string;
}

interface B {
  bar: number;
}

function isA(obj: any): obj is A {
  return typeof obj.foo === 'string'
}

function isB(obj: any): obj is B {
  return typeof obj.bar === 'number'
}

function func(obj: any) {
  if (isA(obj)) {
    console.log("A.foo:", obj.foo);
  }
  else if (isB(obj)) {
    console.log("B.bar:", obj.bar);
  }
  else {console.log("neither A nor B")}
}

const a: A = { foo: 567 }; // notice i am giving it a number, not a string
const b: B = { bar: 123 };

func(a);  // neither A nor B
func(b);  // B.bar: 123
```

Share Improve this answer Follow answered Sep 9 '18 at 3:12 Carl Sorenson 111 ●2 ●2

Add a comment

0

No, You cannot reference a `type` in runtime, but **yes** you can convert an `object` to a `type` with `typeof`, and do validation/sanitisation/checks against this `object` in runtime.

```
const plainObject = {
  someKey: "string",
  someKey2: 1,
};
type TypeWithAllOptionalFields = Partial<typeof plainObject>; //do further utility typings

function customChecks(userInput: any) {
  // do whatever you want with the 'plainObject'
}
```

Above is equal as

```
type TypeWithAllOptionalFields = {
  someKey?: string;
  someKey2?: number;
};
const plainObject = {
  someKey: "string",
  someKey2: 1,
};
function customChecks(userInput: any) {
  // ...
}
```

but without duplication of keynames in your code

Share Improve this answer Follow edited May 28 '20 at 10:29 answered May 28 '20 at 4:21 kairun 46 ●5

Add a comment

0

You can call the constructor and get its name

`let className = this.constructor.name`

Share Improve this answer Follow answered Mar 10 at 15:32 Shady Smaoui 51 ●3

Add a comment

-1

Alternative approach without the need of checking the type

What if you want to introduce more types? Would you then extend your if-statement? How many such if-statements do you have in your codebase?

Using types in conditions makes your code difficult to maintain. There's lots of theory behind that, but I'll save you the hazzle. Here's what you could do instead:

Use polymorphism

Like this:

```
abstract class BaseClass {
    abstract theLogic();
}

class A extends BaseClass {
    theLogic() {
       // do something if class is A
    }
}


class B extends BaseClass {
    theLogic() {
       // do something if class is B
    }
}
```

Then you just have to invoke theLogic() from whichever class you want:

```
let a: A = new A();
a.theLogic();

let b: B = new B();
b.theLogic();
```

Share Improve this answer Follow answered May 30 '20 at 11:07 Stefan Woehrer 620 ●4 ●9

Add a comment

Your Answer

Sign up or log in

Sign up using Google

Sign up using Facebook

Sign up using Email and Password

Post as a guest

Name

Email

Required, but never shown

Post Your Answer

By clicking "Post Your Answer", you agree to our terms of service, privacy policy and cookie policy

Not the answer you're looking for? Browse other questions tagged typescript types runtime detect typeof or ask your own question.

Linked

0 Typescript check generic types equals
1 Check if property of an object is of a specific type
1 how to check a function's type in typescript?
4 Javascript type check am I doing it right?
4 How typescript compares declared types
4 ts-jest: TypeScript type errors are ignored
1 Typescript doesn't type check on runtime
2 How to check custom interface type in Typescript
2 How do you annotate an unnamed arrow function in typescript

See more linked questions

Related

802 How to create a new object instance from a Type
1416 What's the canonical way to check for type in Python?
1766 How to determine a Python variable's type?
1346 What are the differences between type() and isinstance()?
1887 Determine the type of an object?
833 How to check if type of a variable is string?
1608 Check if a value is an object in JavaScript
371 Interface type check with Typescript
1073 How to convert a string to number in TypeScript?
342 How do I prevent the error "Index signature of object type implicitly has an 'any' type" when compiling typescript with noImplicitAny flag enabled?

Hot Network Questions

What tool do I need in order to remove these pedals?
Is a comment aligned with the element being commented a good practice?
Am I allowed to use images from sites like Pixabay in my YouTube videos?
Is it feasible to circumnavigate the Earth in a sailplane?
Functorial kernel in derived category
How can the intelligence of a super-intelligent person be assessed?
Word for the animal providing motive power for a vehicle?
Is it ever worth it to refinance an auto loan for a higher APR?
Should we ask ambiguous questions on an exam?
Self-Replicating Numbers
Is there any reason to use F flat in notating this blues riff (jazz)?