'any' vs 'Object'

174

30

I am looking at TypeScript code and noticed that they use:

```
interface Blablabla {
   field: Object;
}
```

What is the benefit of using `Object` vs `any`, as in:

```
interface Blablabla {
  field: any;
}
```

typescript

edited Sep 6 '18 at 22:41

Alexander Abakumov
5,382 5 50 75

asked Sep 23 '13 at 13:58

Olivier Refalo
34.1k 18 79 111

8 Answers

`Object` is more restrictive than `any`. For example:

The `Object` class does not have a `nomethod()` function, therefore the transpiler will generate an error telling you exactly that. If you use `any` instead you are basically telling the transpiler that anything goes, you are providing no information about what is stored in `a` - it can be anything! And therefore the transpiler will allow you to do whatever you want with something defined as `any`.

So in short

- `any` can be anything (you can call any method etc on it without compilation errors)
- `Object` exposes the functions and properties defined in the `Object` class.

edited Mar 13 at 13:02

answered Sep 23 '13 at 14:28

Nypan
3,884 3 14 26

231

Bit old, but doesn't hurt to add some notes.

When you write something like this

```
var a: any;
var b: Object;
var c: {};
```

- **a** has no interface, it can be anything, the compiler knows nothing about its members so minimal type checking is done when accessing/assigning both to itself and its members. Basically, you're telling the compiler to "*back off, I know what I'm doing, so just trust me*";
- **b** has the Object interface, so ONLY the members defined in that interface are available for **b**. It's still JavaScript, so everything extends Object;
- **c** extends Object, like anything else in TypeScript, but add no members. Since type compatibility in TypeScript is based on structural subtyping, not nominal subtyping, **c** ends up being the same as **b** because they have the same interface: the Object interface.

And that's why

and why

```
a.toString(); // Ok: whatever, dude, have it your way
b.toString(); // Ok: toString is defined in Object
c.toString(); // Ok: c inherits toString from Object
```

So both `Object` and `{}` are equivalent for TypeScript. I don't see anybody really using it. Too restrictive.

If you declare functions like these

```
function fa(param: any): void {}
function fb(param: Object): void {}
```

with the intention of accepting anything for param (maybe you're going to check types at run-time to decide what to do with it), remember that

- inside **fa**, the compiler will let you do whatever you want with **param**;
- inside **fb**, the compiler will only let you access Object's members, and you'll end up having to do a lot of typecasting there...

So, basically, when you don't know the type, go with **any** and do run-time type checking.

Obviously, OO inheritance rules still apply, so if you want to accept instances of derived classes and treat them based on their base type, as in

```
interface IPerson {
    gender: string;
}

class Person implements IPerson {
    gender: string;
}

class Teacher extends Person {}

function func(person: IPerson): void {
    console.log(person.gender);
```

the base type is the way to do it, not **any**. But that's OO, out of scope, I just wanted to clarify that **any** should only be used when you don't know whats coming, and for anything else you should annotate the correct type.

UPDATE:

Typescript 2.2 added an `object` type, which specifies that a value is a non-primitive: (i.e. not a `number`, `string`, `boolean`, `symbol`, `undefined`, or `null`).

Consider functions defined as:

```
function b(x: Object) {}
function c(x: {}) {}
function d(x: object) {}
```

`x` will have the same available properties within all of these functions, but it's a type error to call `d` with anything non-primitive:

```
b("foo"); //Okay
c("foo"); //Okay
d("foo"); //Error: "foo" is a primitive
```

edited Oct 31 '17 at 20:46
Retsam
7,122 4 30 55

answered Mar 1 '15 at 16:03
diegovilar
2,619 1 10 5

2 Does anybody know *why* they decided to add `{}` then if they already had `Object`? (or vice versa, whichever came first) There's got to be some slight difference, right? – CletusW May 25 '17 at 16:01

3 `{}` is the normal way to define (inline) interfaces, only that in this case you are defining an interface with no members. The slight difference is well explained in the response: "`{}` extends `Object`, like anything else in TypeScript". – DanielM Jun 29 '17 at 12:32

7 I want to down vote you for the line *So, basically, when you don't know the type, go with `any` and do run-time type checking.* Don't use `any`, instead use a union of the types you are checking against: `TypeA|InterfaceB|string`. If you also have a default case for an unknown type, add either `{}` or `Object` to the union. – ILMTitan Feb 2 '18 at 18:09

Typescript docs are sometimes confusing, for example `But variables of type Object only allow you to assign any value to them - you can't`

24

`any` is something specific to TypeScript is explained quite well by alex's answer.

`Object` refers to the JavaScript `object` type. Commonly used as `{}` or sometimes `new Object`. Most things in *javascript* are compatible with the object data type as they inherit from it. But `any` is *TypeScript* specific and compatible with everything in both directions (not inheritance based). e.g. :

```
var foo:Object;
var bar:any;
var num:number;

foo = num; // Not an error
num = foo; // ERROR

// Any is compatible both ways
bar = num;
num = bar;
```

answered Sep 23 '13 at 23:18

basarat

149k 28 278 383

1 Your answer is quite vague and mixes `Object` and `object` which are different types in TypeScript. – m93a Apr 30 '18 at 15:36

@m93a: Can you expand on what is the difference between `Object` and `object` in TS? – Alexander Abakumov Jun 18 '18 at 19:22

1 This is probably the best source to learn the difference. The main point is that `object` is a type for everything that is not primitive, while `Object` is an interface that contains common things like `toString` and such. The number `42` would be an `Object` but not an `object`. – m93a Jun 20 '18 at 14:22

16

Contrary to .NET where all types derive from an "object", in TypeScript, all types derive from "any". I just wanted to add this comparison as I think it will be a common one made as more .NET developers give TypeScript a try.

answered Sep 24 '13 at 15:00

yellowbrickcode

14

> All types in TypeScript are subtypes of a single top type called the Any type. The any keyword references this type. The Any type is the one type that can represent any JavaScript value with no constraints. All other types are categorized as primitive types, object types, or type parameters. These types introduce various static constraints on their values.

Also:

> The Any type is used to represent any JavaScript value. A value of the Any type supports the same operations as a value in JavaScript and minimal static type checking is performed for operations on Any values. Specifically, properties of any name can be accessed through an Any value and Any values can be called as functions or constructors with any argument list.

Objects do not allow the same flexibility.

For example:

```
var myAny : any;

myAny.Something(); // no problemo

var myObject : Object;

myObject.Something(); // Error: The property 'Something' does not exist on value of type
'Object'.
```

edited Sep 24 '13 at 10:35

answered Sep 23 '13 at 14:28

Alex Dresko
3,704 1 28 50

1 Good answer, could be better to show the lack of flexibility – basarat Sep 23 '13 at 21:32

2 Edited. Maybe not the best amendment, but is at least truthful. :) – Alex Dresko Sep 24 '13 at 10:35

Adding to Alex's answer and simplifying it:

answered Aug 18 '17 at 4:34

kg11
961 9 13

0

Keep in mind that you should never use types **Number**, **String**, **Boolean**, or **Object** as accordingly to the Typescript documentation

> These types refer to non-primitive boxed objects that are almost never used appropriately in JavaScript code.

answered Apr 30 at 10:21

Kamil
45 1 8

-2

try this :

```
private a: Object<any> = {};

constructor() {
    a.name = "foo";
}
```

answered Sep 19 '17 at 8:06

Chandru
6,058 2 21 34

1 Code-only answers are discouraged because they do not explain how they resolve the issue. Please update your answer to explain how this improves on the other accepted and upvoted answers this question already has. Also, this question is 4 years old, your efforts would be more appreciated by users who have recent unanswered questions. Please review How do I write a good answer. – FluffyKitten Sep 19 '17 at 9:03
