Typescript: Interfaces vs Types

Asked 3 years, 4 months ago Active 1 month ago Viewed 82k times

479

98

What is the difference between these statements (interface vs type)?

```
interface X {
    a: number
    b: string
}

type X = {
    a: number
    b: string
};
```

typescript

edited Jan 7 '18 at 22:05
Thomas Eding
24.6k 9 54 97

asked May 15 '16 at 1:53
user6101582

2 Found this article explaining the differences - medium.com/@martin_hotell/… – Sandeep G B Oct 2 '18 at 20:11

6 Answers

451

As per the TypeScript Language Specification:

> Unlike an interface declaration, which always introduces a named object type, **a type alias declaration** can introduce a name for any kind of type, including primitive, union, and intersection types.

+200

Interface types have many similarities to type aliases for object type literals, but since interface types offer more capabilities they are generally preferred to type aliases. For example, the interface type

```
interface Point {
    x: number;
    y: number;
}
```

could be written as the type alias

```
type Point = {
    x: number;
    y: number;
};
```

However, doing so means the following capabilities are lost:

- ~~An interface can be named in an extends or implements clause, but a type alias for an object type literal cannot~~ No longer true since TS 2.7.
- An interface can have multiple merged declarations, but a type alias for an object type literal cannot.

edited Jun 7 at 9:58

Ilario Pierbattista
2,497 2 19 30

answered May 15 '16 at 2:01

Binary Birch Tree
6,476 1 10 11

79 What does "multiple merged declarations" mean in the second difference? – jrahhali Mar 19 '17 at 23:21

47 @jrahhali if you define interface twice, typescript merges them into one. – Andrey Fedorov Jul 5 '17 at 5:12

25 @jrahhali if you define type twice, typescript gives you error – Andrey Fedorov Jul 5 '17 at 5:13

12 @jrahhali `interface Point { x: number; } interface Point { y: number; }` – Nahuel Greco Jul 6 '17 at 15:53

16 I believe first point `extends or implements` is no longer the case. Type can be extended and implemented by a `class` . Here's an example typescriptlang.org/play/… – dark_ruby Sep 1 '17 at 13:39

386

The current answers and the official documentation are outdated. And for those new to TypeScript, the terminology used isn't clear without examples. Below is a list of up-to-date differences.

1. Objects / Functions

Both can be used to describe the shape of an object or a function signature. But the syntax differs.

Interface

```
interface Point {
  x: number;
  y: number;
}

interface SetPoint {
  (x: number, y: number): void;
}
```

Type alias

```
type Point = {
  x: number;
  y: number;
};

type SetPoint = (x: number, y: number) => void;
```

2. Other Types

Unlike an interface, the type alias can also be used for other types such as primitives, unions, and tuples.

```
// primitive
type Name = string;

// object
type PartialPointX = { x: number; };
type PartialPointY = { y: number; };
```

```
// tuple
type Data = [number, string];
```

3. Extend

Both can be extended, but again, the syntax differs. Additionally, note that an interface and type alias are not mutually exclusive. An interface can extend a type alias, and vice versa.

Interface extends interface

```
interface PartialPointX { x: number; }
interface Point extends PartialPointX { y: number; }
```

Type alias extends type alias

```
type PartialPointX = { x: number; };
type Point = PartialPointX & { y: number; };
```

Interface extends type alias

```
type PartialPointX = { x: number; };
interface Point extends PartialPointX { y: number; }
```

Type alias extends interface

```
interface PartialPointX { x: number; }
type Point = PartialPointX & { y: number; };
```

4. Implements

A class can implement an interface or type alias, both in the same exact way. Note however that a class and interface are considered static blueprints. Therefore, they can not implement / extend a type alias that names a union type.

```
interface Point {
  x: number;
```

```
class SomePoint implements Point {
  x: 1;
  y: 2;
}

type Point2 = {
  x: number;
  y: number;
};

class SomePoint2 implements Point2 {
  x: 1;
  y: 2;
}

type PartialPoint = { x: number; } | { y: number; };

// FIXME: can not implement a union type
class SomePartialPoint implements PartialPoint {
  x: 1;
  y: 2;
}
```

5. Declaration merging

Unlike a type alias, an interface can be defined multiple times, and will be treated as a single interface (with members of all declarations being merged).

```
// These two declarations become:
// interface Point { x: number; y: number; }
interface Point { x: number; }
interface Point { y: number; }

const point: Point = { x: 1, y: 2 };
```

edited Jul 11 at 13:43

answered Oct 6 '18 at 18:38

jabacchetta

8,046 3 25 42

26 This provides better detailed info on how to use type vs interface. – thibmaek Oct 26 '18 at 12:46

of interfaces. Beyond that, they are equivalent (and I'd argue that type aliases offer more concise syntax). – maxedison Feb 9 at 7:36

4 I always using interfaces for object type literal, otherwise using types make more sense, also I think that declaration merging shouldn't be used in anyways, actually I'll never expect that an interface is being declared in another file of the project with some extra properties, type checking is made originally to make your life easier not to make it harder with this ninja-like interfaces :D – Ahmed M.Kamal Mar 4 at 21:35

4 Can this not be updated and makred as the best answer – Faktor 10 May 3 at 8:42

58

As of TypeScript 3.2 (Nov 2018), the following is true:

Aspect	Type	Interface
Can describe functions	✅	✅
Can describe constructors	✅	✅
Can describe tuples	✅	✅
Interfaces can extend it	⚠️	✅
Classes can extend it	🚫	✅
Classes can implement it (`implements`)	⚠️	✅
Can intersect another one of its kind	✅	⚠️

Can be used to create mapped types	✅	🚫
Can be mapped over with mapped types	✅	✅
Expands in error messages and logs	✅	🚫
Can be augmented	🚫	✅
Can be recursive	⚠️	✅

⚠️ In some cases

edited Apr 30 at 9:04

Brian Burns
8,520 5 51 49

answered Jan 9 at 0:09

Karol Majewski
5,975 1 10 22

6 Could you please provide more information about how the table/image you provided was generated? e.g. source code or links to documentation – iX3 Jan 23 at 16:20

1 Sure! It's a Markdown table rendered by GitHub. You can find the syntax here: help.github.com/articles/organizing-information-with-tables. If you're interested in the content itself, it's from my personal notes. I should probably elaborate and provide examples, but it's a very long read. – Karol Majewski Jan 23 at 22:31

20 yes, I meant the source of the content, not its presentation. – iX3 Jan 24 at 18:52

2 I don't believe a *class* can *extend* either a type or an interface, and I can't really see why you would want to?? – Dan King May 5 at 13:59

1 Avoid posting images of text, instead include the actual text directly into your post. Images of text are not easily parsable or searchable, and are not

25

https://www.typescriptlang.org/docs/handbook/advanced-types.html

> One difference is that interfaces create a new name that is used everywhere. Type aliases don't create a new name — for instance, error messages won't use the alias name.

answered Feb 16 '17 at 12:13

nickf
391k 176 597 694

16 This is now outdated and no longer true since TypeScript 2.1. See. medium.com/@martin_hotell/… – demisx Sep 22 '18 at 13:44

2

Examples with Types:

// create a tree structure for an object. You can't do the same with interface because of lack of intersection (&)

```
type Tree<T> = T & { parent: Tree<T> };
```

// type to restrict a variable to assign only a few values. Interfaces don't have union (|)

```
type Choise = "A" | "B" | "C";
```

// thanks to types, you can declare NonNullable type thanks to a conditional mechanism.

```
type NonNullable<T> = T extends null | undefined ? never : T;
```

Examples with Interface:

// you can use interface for OOP and use 'implements' to define object/class skeleton

```
interface IUser {
```

```
}

class User implements IUser {
    user = "user1"
    password = "password1"

    login(user: string, password: string) {
        return (user == user && password == password)
    }
}
```

// you can extend interfaces with other interfaces

```
interface IMyObject {
    label: string,
}

interface IMyObjectWithSize extends IMyObject{
    size?: number
}
```

answered May 29 at 12:11

Przemek Struciński

743 6 8

0

the documentation has explained

> - One difference is that interfaces create a new name that is used everywhere. Type aliases don't create a new name — for instance, error messages won't use the alias name.in older versions of TypeScript, type aliases couldn't be extended or implemented from (nor could they extend/implement other types). As of version 2.7, type aliases can be extended by creating a new intersection type
> - On the other hand, if you can't express some shape with an interface and you need to use a union or tuple type, type aliases are usually the way to go.

Interfaces vs. Type Aliases

1 3