

这次还搞不清type和interface，买块豆腐自己撞吧

说到 `type` 和 `interface` ,很多时候很多人都是用的很随意，因为有时候可以用 `type` ，也可以用 `interface` 。那到底时候用 `type` 什么时候用 `interface` 呢？

要搞明白这个问题，就先得搞明白这两者各自的特点

interface

`interface` 我们叫他 接口，`ts` 设计出来主要用于定义对象类型，可以对对象类型 进行描述
比方说，我们现在要对一个对象的类型进行约束，你可以这样来

```
1 css
2 复制代码
3 let obj: { age: number; name: string } = { age: 28, name: "constRen" };
```

这样写的话，属性多的时候，可能就不是很美妙了，还有就是，每写一个对象，都要去写一下类型，很麻烦。作为一个程序员，这能忍？

所以，接口就应运而生

```
1 typescript
2 复制代码
3 interface User { name: string; age: number; sex: string; hobby?:
  string;};let obj1: User = { name: "constRen", age: 28, sex: "man", hobby:
  "basketball",};let obj2: User = { name: "张三", age: 28, sex:
  "man",};interface addTp { (num1: number, num2: number): number;};const add:
  addTp = (num1, num2) => { return num1 + num2;};
```

这就是接口的简单用法

type

`type` 就是一个类型别名，说白了就是一个小名，绰号。。。我不相信有人每次称呼乔丹都是迈克尔乔丹，我们一般都是称之为乔丹，这就是别名，把科比布莱恩特称之为科比，把勒布朗詹姆斯称之为詹姆斯，把扬尼斯·西纳·乌戈·阿德托昆博称之为字母哥。。。这就是别名。它只是给类型起

一个新名字(随便自定义一个类型, 使用 `type` 给他一个名字就行, 加上 `type` 的写法就是一个表达式, 所以注定它用途广泛啊)

```
1 ini
2 复制代码
3 type Person = {    name: string    age: number};const my: Person = {    name:
  'constRen',    age: 28};type addTp = (num1: number, num2: number) =>
  number;const add: addTp = (num1, num2) => {    return num1 + num2;};type Name =
  string;let myName: Name = "constRen";
```

这就是 `type` 的简单用法

两者相同点

- 都能定义函数和对象(见上面的代码)
- 都能继承(拓展)
 - `interface` 的继承

```
// interface 继承 interface
interface Person {
  | name: string;
}
interface Student extends Person {
  | age: number;
}

let my: Student = {
  | name: "constRen",
  | age: 28,
```

- `type` 的继承

```
// type 继承 type
type Person = {
  | name: string;
};

type Student = Person & {
  | age: number;
};

let my: Student = {
  | name: "constRen",
  | age: 28,
};
```

```
// type 继承 interface
interface Person {
  | name: string;
}

type Student = Person & { age: number };

let my: Student = {
  | name: "constRen",
  | age: 28,
};
```

两者的不相同点

`type` 可以，`interface` 不行的

`type` 会给一个类型起个新名字，有时和 `interface` 很像，但是 `type` 可以作用于原始值，联合类型，元组以及其它任何你需要手写的类型。

- 声明基本类型、元组

```
1 ini
2 复制代码
3 type Name = string; // 基本类型
type Arrtp = number | string; // 联合类型
const arr: Arrtp[] = [1, "2", 3, "4"]; // 元组
type Person = { name: Name; };
type Son = Person & { age: number }; // 交叉类型
type Father = Person & { sex: string };
type SfArr = [Son, Father]; // 元组类型
const list: SfArr = [ { name: "ren_son", age: 10 }, { name: "ren_father", sex: "man" },];
```

- `keyof`、`in`、`Pick`、`Omit`、`Exclude`、`Extract`..... 这些只有 `type` 可以

```

$ testTs.ts > ...
1
2   type Keys = "name" | "sex";
3
4   type KTp = {
5     [key in Keys]: string;
6   };
7
8   type KTp = {
9     name: string;
10    sex: string;
11  };
12  const test: KTp = {
13    name: "Paweł",
14    sex: "Grzybek",
15  };
16

```

- 就不一一举例了，那些 `ts` 的中高级以及体操类型，基本都是使用 `type`

`interface` 可以，`type` 不行的

- 合并重复声明: `interface` 重复声明就会合并，`type` 不支持，会给你报个错

```

$ testTs.ts > ...
1   interface Person {
2     name: string;
3   };
4
5   interface Person {
6     age: number;
7   };
8
9
10
11
12
13
14  const person: Person = {
15    age: 18,

```

类型 "{ age: number; }" 中缺少属性 "name"，但类型 "Person" 中需要该属性。 ts(2741)

testTs.ts(2, 3): 在此处声明了 "name"。

const person: Person

查看问题 (Alt+F8) 没有可用的快速修复

- 从这个例子可以看到，本身 `Person` 里面只有 `name` 属性，没有 `age` 属性，但是重复声明之后就会合并，就有了 `name` 和 `age` 两个属性，在没有加**可选**的情况下，少了这个属性就会报错。我们再来看 `type` 的重复声明

```
testTs.ts > ...
type Person {
  name: string;
};

type Person = {
  age: number;
};

type Person {
  age: number;
};
```

标识符“Person”重复。 ts(2300)

```
type Person = {
  age: number;
}
```

[查看问题 \(Alt+F8\)](#) 没有可用的快速修复

- 可以看到，直接报错了，已经重复

很多文章都说 `type` 可以使用 `typeof` ， `interface` 不行，其实不然

```
testTs.ts > ...
1  let str = "11";
2  type A = typeof str;
3
4  interface B {
5    name: typeof str;
6  }
7
8  const a: A = '22';
9
10 const b: B = {
11   name: '2',
12 };
13
14
15
16
17
18
19
20 const c: B = {
21   name: 222
22 }
```

再把报错信息截出来

```

TS testTs.ts > ...
1  let str = "11";
2  type A = typeof str;
3
4  interface B {
5    | name: typeof str;
6  }
7
8  const a: A = '22';
9
10 const b: B = {
11   | name: '2',
12 };
13
14
15 不能将类型“number”分配给类型“string”。 ts(2322)
16  testTs.ts(5, 3): 所需类型来自属性 "name", 在此处的 "B" 类型上声明
17  | 该属性
18
19  (property) B.name: string
20  co 查看问题 (Alt+F8) 没有可用的快速修复
21  |  name:222

```

都是可以使用的

还有说 `interface` 不能搞 联合类型 的

```
interface Info {
  name: string;
  sex: string;
  age: string | number;
  id: string | number;
}

const o1: Info = {
  name: "constRen",
  sex: "man",
  age: 22,
  id: 5135651215121,
};

const o2: Info = {
  name: "constRen",
  sex: "man",
  age: "22",
  id: "5135651215120",
};
```

但是这种是对象里面具体属性搞了一下联合类型，像这种 `type result = boolean | number | string | null` 这种直接是基本类型的那真的是无能为力了。

总结

类型别名和接口非常相似，在大多数情况下你可以在它们之间自由选择。几乎所有的 interface 功能都可以使用 type 实现，关键区别在于 `interface` 主要用于描述一个对象。

`type` 是类型别名，用于给各种类型定义别名，联合类型或者 元组类型，以及 `ts` 体操类型，基本上都是使用 `type`

拓展：

联合类型

联合类型一般适用于基本类型的合并，它使用 `|` 符号进行连接

```
1 ini
2 复制代码
3 type result = 'name' | 1 | true | null
```

交叉类型

交叉类型则适用于对象或者函数的合并，它使用 `&` 符号进行连接, *具有所有类型的特性*, 取他们类型的 **合集**

```
1 ini
2 复制代码
3 type result = T & U
```

`T & U` 表示一个新的类型，其中这个类型**包含T和U中所有的键**，这和 `JavaScript` 中的 `Object.assign()` 函数的作用非常类似。