08 | 类与接口的相互操作

周爱民（Aimingoo）

目录

```
class MyClass {

}
```

```
interface IClass {

}
```

```
type T = {

}
```

```
let x = {

};
type T = typeof x;
```

ECMAScript / JavaScript			ECMAScript class	TypeScript class	interface	{ ... }	{ ... }
可见性	private			Y			
	protected			Y			
	public			Y	N(*2)		
类型语义	override			Y			
	abstract			Y	N(*1)		
	(Parament property)			Y			
	(Merging)			Y	Y		
	(Overload)			Y	Y	Y	
	(Signature)			Y	Y	Y	
成员修饰	*(Optional)*		N(*2)	Y	Y	Y	N(*5)
	readonly		N(*2)	Y	Y	Y	N(*2)
继承	A implements T	A		Y			
		T (*3)		Y	Y	Y	Y
ES/JS	A extends B	A	Y	Y	Y		
		B (*4)	Y	Y	Y	Y	Y
	async		Y	Y	N(*2)	N(*2)	Y
	*		Y	Y	N(*2)	N(*2)	Y
	static		Y	Y			
	accessor		Y	Y			
	#		Y	Y			
	(Member)	*(property)*	Y	Y	Y	Y	Y
		(method)	Y	Y	Y	Y	Y
		(Field)	Y	Y			
		get/set	Y	Y	Y	Y	Y

A：总是可以把类名（类类型）当成接口来用，但接口声明的语法特性少于类声明；

B：也总是可以把对象类型（以及对象字面实例的类型）当成接口来用，但是，1、少了 extends 继承，2、在实例声明中不能支持可选属性等5种语法；3、不能合并名字。并且，相较于接口声明，对象类型在签名语法上略受限制。

C：基于A、B，我们只需要讨论类与接口的相互操作（字面类型可以当作接口来处理）。

*1 可以用交叉使接口得到类似性质。
*2 不能使用相应的关键字或语法，但可以使用替代方法得到类似性质（与*1类似）。
*3 T必须是一个名字（类型标识符名或限定名）而不能是匿名类型，所谓限定名是指`Namespace.TypeName`这样的语法。
*4 在class声明中使用的X必须是类名，而在interface中使用的X可是以类、接口和字面对象类型的名字（并且类名在这里是引用隐式声明的接口名）。
*5 不能从**字面对象**中取得一个“带有可选成员声明的类型”，因为在概念上这个成员就是既存的；一个成员“可能/可选”存在，在TS类型系统中要么使用接口/类型声明中的可选属性，要么使用索引签名。

可选属性 vs. 可选参数 vs. 缺省参数

1. 在接口、对象或类的声明中，属性使用`?:`来声明的是可选参数，可选参数意味着：
   - 该属性可以不存在于对象成员列表中，即 `x` in obj === false。
   - 该属性是可以读取并返回 undefined 值的，而它的类型中也包括 undefined

2. 可选参数总是在方法、函数的界面上，它表明该参数可以不传值
   - 它等义于一个值包含了 undefined 类型的缺省参数
   - 它必须位于参数列表的末尾

3. 如果一个参数使用了其它类型的缺省参数，那么该参数不能同时是一个可选参数（即，不能既是“可选参数”，又是“缺省为非 undeinfed 的缺省参数”。例如：
   - `function f(x?: number = 10) { ... }` *// 非法的（TS1015）*

4. 从语法上，参数初始器不能写入到类型声明中。例如：
   - `type T = (x: number = 10) => void;` *// 非法的（TS2371）*

接口的用法：继承树设计

极客时间

```
type BirdAndHorse = ...
type BirdOrHorse = ...

class MyClass implements Bird, Horse {

}
```

类的接口

```
class MyClass implements MyClass {
    …
}

type T = Omit<MyClass, never>;
let x: MyClass = new MyClass();
```

接口的用法2：维护原型成员

```
interface MyClass {
    c: string;
}

class MyClass {
    a: string;
    b: number;
}

MyClass.prototype.c = 'abc';
```

接口的用法3：方法重载

```
interface IFoo {
    foo(s: string): void;
    foo(n: number): void;
}

class MyClass implements IFoo {
    foo() {

    }
}

let x: IFoo = new MyClass;
```

总结

1. 类隐式地实现了一个与它同名的接口，并且
   - 该接口被声明为 `MyClass.prototype` 的类型；
   - 该接口可以被重复声明，且用户声明的成员不强制要求在类声明块中实现。

2. 在 TypeScript 中可以使用基于接口的多重继承
   - 可以使用交叉和联合来辅助设计类继承树
   - 可以在构造方法中返回定制的对象（但必须是MyClass的实例）

3. 在不能使用基于 `extends` 或 `implements` 的继承兼容时，接口、类、对象等结构类型之间将使用结构兼容检查。

名词/概念

名词、术语

重载：overload
合并、同名合并：merging，merging names
声明合并：declaration merging

概念

- 声明合并（Declaration merging）意味着编译器将使用相同名称声明的两个单独的声明合并到一个定义中（*into a single definition*）。
  @see: https://www.typescriptlang.org/docs/handbook/declaration-merging.html
- 声明（Declaration）至少在以下三组之一中创建实体：命名空间、类型或值（*creates entities in at least one of ...*）。
  @see: https://www.typescriptlang.org/docs/handbook/declaration-merging.html
- 显式声明函数可以用哪些不同方式调用的方法称为重载（Overload），而在TypeScript中使用重载签名（overload signatures）来指明函数能用不同的方法调用（*can be called in different ways by writing overload signatures*）。并且这也可以用于方法重载和构造方法重载上。
  @see: https://www.typescriptlang.org/docs/handbook/2/functions.html#function-overloads

Q&A

Q: 课程中讲到的typeof是什么?

A: 这里的 `typeof` 不是 JavaScript 中的运算符，而是 TypeScript 在类型系统中的运算符，它返回一个变量（名字）的类型。这里取到的类型可以是推断的类型，也可以是标注的类型。有关这个运算符，我们会在“14 | 在JS与TS间的互通访问技术 ”中再提及，并且还会在本课程第二篇中详细讲述。

Q: “联合多个接口等同于找到它们的父类”有什么意义?

A: 在 OOP 中，提供层次清晰的继承结构是一个良好的设计实践。但是在实际开发时，“设计与实现一个具体的类”才往往是更自然的。因为更多的抽象层级，也就意味着更难触及到具体的业务逻辑。这也是为什么大多数的开发人员难以“将类继承结构设计到两层以上”的原因。然而一旦我们理解了“联合多个接口等同于找到它们的父类”，那么我们就可以先开发具体代码、设计具体类，然后通过“联合那些有近似概念的类”来找到它们公共的父类，并快速得到父类的接口设计。这对于从既有代码中抽象、剥离一个框架或库是有极大帮助的。

Q: 为什么我在重现这些类声明的示例时，总是出现ts(2564)错误?

A: 一般情况下，使用`tsc --init`初始化的 tsconfig.json 中，“strict” 参数会被自动配置为 true 值，这会严格检查那些未初始化的类声明成员，因此会有 TS2564 错。而在示例项目中，我添加了一个定制的 tsconfig.json，其中的 “strict” 缺省为false。


极客时间
THANKS