

04 八个JS中你见过的类型

更新时间：2019-07-01 14:16:50



生活永远不像我们想像的那样好，但也不会像我们想像的那样糟。

——莫泊桑

这小节你学习起来会很轻松，这是你正式接触 **TypeScript** 语法的第一节课，是最基础的语法单元。这节课我们将学习在 **JavaScript** 中现有的八个数据类型，当然这并不是 **JavaScript** 中的所有数据类型，而是现在版本的 **TypeScript** 支持的基本类型，在学习基础类型之前，我们先来看下如何为一个变量指定类型：

为一个变量指定类型的语法是使用"变量: 类型"的形式，如下：

```
let num: number = 123
```

如果你没有为这个变量指定类型，编译器会自动根据你赋给这个变量的值来推断这个变量的类型：

```
let num = 123
num = 'abc' // error 不能将类型""123""分配给类型"number"
```

当我们给 **num** 赋值为123但没有指定类型时，编译器推断出了 **num** 的类型为 **number** 数值类型，所以当给 **num** 再赋值为字符串"abc"时，就会报错。

这里还有一点要注意，就是 **number** 和 **Number** 的区别：TS中指定类型的时候要用 **number**，这个是TypeScript的类型关键字。而 **Number** 为JavaScript的原生构造函数，用它来创建数值类型的值，它俩是不一样的。包括你后面见到的 **string**、**boolean** 等都是TypeScript的类型关键字，不是JavaScript语法，这点要区分开。接下来我们来看本节课的重点：八个JS中你见过的类型。

2.1.1 布尔类型

类型为布尔类型的变量的值只能是 **true** 或 **false**，如下：

```
let bool: boolean = false;
bool = true;
bool = 123; // error 不能将类型"123"分配给类型"boolean"
```

当然了，赋给 `bool` 的值也可以是一个计算之后结果是布尔值的表达式，比如：

```
let bool: boolean = !!0
console.log(bool) // false
```

2.1.2 数值类型

TypeScript 和 JavaScript 一样，所有数字都是浮点数，所以只有一个 `number` 类型，而没有 `int` 或者 `float` 类型。而且 TypeScript 还支持 ES6 中新增的二进制和八进制数字字面量，所以 TypeScript 中共支持二、八、十和十六四种进制的数值。

```
let num: number;
num = 123;
num = "123"; // error 不能将类型"123"分配给类型"number"
num = 0b1111011; // 二进制的123
num = 0o173; // 八进制的123
num = 0x7b; // 十六进制的123
```

2.1.3 字符串

字符串类型中你可以使用单引号和双引号包裹内容，但是可能你使用的 `tslint` 规则会对引号进行检测，使用单引号还是双引号可以在 `tslint` 规则里配置。你还可以使用 ES6 语法——模板字符串，拼接变量和字符串更为方便。

```
let str: string = "Lison";
str = "Li";
const first = "Lison";
const last = "Li";
str = `${first} ${last}`;
console.log(str) // 打印结果为:Lison Li
```

另外还有个和字符串相关的类型：**字符串字面量类型**。即把一个字符串字面量作为一种类型，比如上面的字符串 `"Lison"`，当你把一个变量指定为这个字符串类型的时候，就不能再赋值为其他字符串值了，如：

```
let str: 'Lison'
str = 'haha' // error 不能将类型""haha""分配给类型""Lison""
```

2.1.4 数组

在 TypeScript 中有两种定义数组的方式：

```
let list1: number[] = [1, 2, 3];
let list2: Array<number> = [1, 2, 3];
```

第一种形式通过 `number[]` 的形式来指定这个类型元素均为 `number` 类型的数组类型，这种写法是推荐的写法，当然你也可以使用第二种写法。注意，这两种写法中的 `number` 指定的是数组元素的类型，你也可以在这里将数组的元素指定为任意类型。如果你要指定一个数组里的元素既可以是数值也可以是字符串，那么你可以使用这种方式：`number|string[]`，这种方式我们在后面学习联合类型的时候会讲到。

当你使用第二种形式定义时，`tslint` 可能会警告让你使用第一种形式定义，如果你就是想用第二种形式，可以通过在 `tslint.json` 的 `rules` 中加入 `"array-type": [false]` 关闭 `tslint` 对这条的检测。

后面我们讲接口的时候，还会讲到数组的一个特殊类型：`ReadonlyArray`，即只读数组。

2.1.5 null 和 undefined

null 和 undefined 有一些共同特点，所以我们放在一起讲。说它们有共同特点，是因为在 JavaScript 中，undefined 和 null 是两个基本数据类型。在 TypeScript 中，这两者都有各自的类型即 undefined 和 null，也就是说它们既是实际的值，也是类型，来看实际例子：

```
let u: undefined = undefined; // 这里可能会报一个tslint的错误：Unnecessary initialization to 'undefined'，就是不能给一个值赋undefined，但我们知道这是可以的，所以如果你的代码规范想让这种代码合理化，可以配置tslint，将"no-unnecessary-initializer"设为false即可
let n: null = null;
```

默认情况下 undefined 和 null 可以赋值给任意类型的值，也就是说你可以把 undefined 赋值给 void 类型，也可以赋值给 number 类型。当你在 tsconfig.json 的"compilerOptions"里设置了 "strictNullChecks": true 时，那必须严格对待。undefined 和 null 将只能赋值给它们自身和 void 类型，void类型我们后面会学习。

2.1.6 object

object 在 JS 中是引用类型，它和 JS 中的其他基本类型不一样，像 number、string、boolean、undefined、null 这些都是基本类型，这些类型的变量存的是他们的值，而 object 类型的变量存的是引用，看个简单的例子：

```
let strInit = "abc";
let strClone = strInit;
strClone = "efg";
console.log(strInit); // 'abc'

let objInit = { a: "aa" };
let objClone = objInit;
console.log(objClone) // {a:"aa"}
objInit.a = "bb";
console.log(objClone); // { a: 'bb' }
```

通过例子可以看出，我们修改 objInit 时，objClone 也被修改了，是因为 objClone 保存的是 objInit 的引用，实际上 objInit 和 objClone 是同一个对象。

当我们希望一个变量或者函数的参数的类型是一个对象的时候，使用这个类型，比如：

```
let obj: object
obj = { name: 'Lison' }
obj = 123 // error 不能将类型"123"分配给类型"object"
```

这里有一点要注意了，你可能会想到给 obj 指定类型为 object 对象类型，然后给它赋值一个对象，后面通过属性访问操作符访问这个对象的某个属性，实际操作一下你就会发现会报错：

```
let obj: object
obj = { name: 'Lison' }
console.log(obj.name) // error 类型"object"上不存在属性"name"
```

这里报错说类型 object 上没有 name 这个属性。如果你想要达到这种需求你应该使用我们后面章节要讲到的接口，那 object 类型适合什么时候使用呢？我们前面说了，当你希望一个值必须是对象而不是数值等类型时，比如我们定义一个函数，参数必须是对象，这个时候就用到object类型了：

```
function getKeys (obj: object) {
    return Object.keys(obj) // 会以列表的形式返回obj中的值
}
getKeys({ a: 'a' }) // ['a']
getKeys(123) // error 类型"123"的参数不能赋给类型"object"的参数
```

这里涉及到的函数的相关知识，我们会在后面章节介绍的，你只要在这里明白object类型的使用就可以了。

2.1.6 symbol

Symbol 是 ES6 加入的新的基础数据类型，因为它的知识比较多，所以我们单独在后面的一节进行讲解。

本节小结

本小节我们学习了八个在JavaScript中我们就见过的数据类型，它们是：布尔类型、数值类型、字符串、数组、null、undefined、object以及ES6中新增的symbol。在TypeScript中它们都有对应的类型关键字，对应关系为：

- 布尔类型：boolean
- 数值类型：number
- 字符串类型：string
- 数组：Array<type>或type[]
- 对象类型：object
- Symbol类型：symbol
- null和undefined：null 和 undefined，这个比较特殊，它们自身即是类型

这些类型是基础，我们后面的高级类型很多都是它们的组合或者变形，所以一定要把这些基础先学会。下个小节我们将学习 TypeScript 中新增的几种类型，了解更多基本类型。

