【TypeScript 4.5】003-第 3 章 常用类型

一、基元类型 string、number 和 boolean

1、基本含义

> 注意全是小写字母！

string：字符串类型，如“Hello World!”；

number：数字类型，包含整数、小数等，如100，-44，1.25；

boolean：布尔类型，只有 `true` 与 `false` 两个值！

2、举个例子

> 补充：设置编译成的 js 文件输出的文件夹位置
>
> 在配置文件中修改："outDir": "./dist"

代码演示

```
let str : string = "hello typescript!"
let num : number = 100
let bool : boolean = true
```

编译结果

```
"use strict";
let str = "hello typescript!";
let num = 100;
let bool = true;
```

二、数组类型

1、两种方式

> type 为任意类型！

方式一：使用类型加上中括号：`type[]`；

方式二：泛型方式，`Array<type>`

2、代码示例

```
// 方式一
let arr : number[] = [1, 2, 3]
arr = ['a'] // 报错：不能将类型“string”分配给类型“number”。

// 方式二：泛型写法
let arr2 : Array<number> = [1, 2, 3]
arr2 = ['a'] // 报错：不能将类型“string”分配给类型“number”。
```

三、any 类型

1、概述

当不希望某个特定值导致类型检查错误时可将其声明为 any 类型！

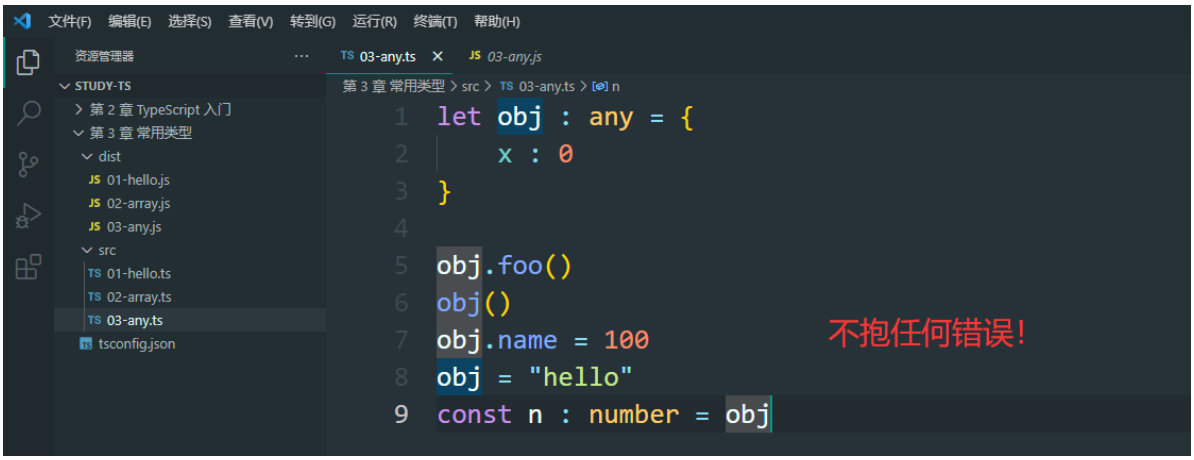
2、代码演示

代码示例

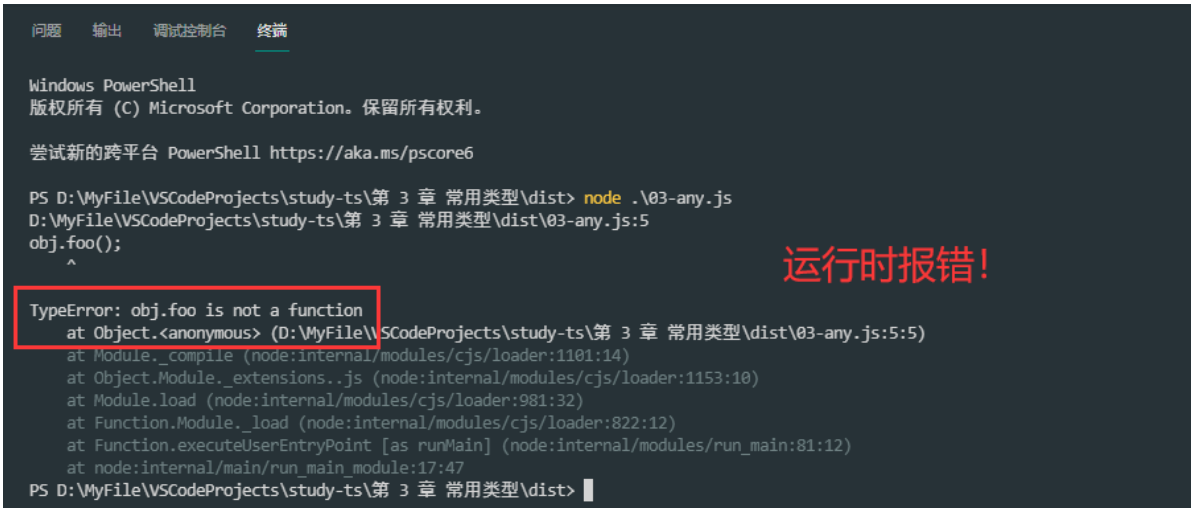
```
let obj : any = {
    x : 0
}

obj.foo()
obj()
obj.name = 100
obj = "hello"
const n : number = obj
```

不报任何错误

> 啊！打了个错别字！竟然还被自己发现了！



运行生成的 js 文件结果



四、变量上的类型注释

1、显式指定变量类型

```
let myName : string = "zibo" // 冒号 + 类型
// 下面是大众编写习惯，上面是本人编写习惯！
let myName: string = "zibo"
```

2、代码示例与解释

```
let myName : string = "zibo" // 这么写没问题

// 一般不用写类型，因为类型推断很强大
let mName = "zibo"
mName = 100 // 报错：不能将类型“number”分配给类型“string”。
```

五、函数类型

1、代码分析

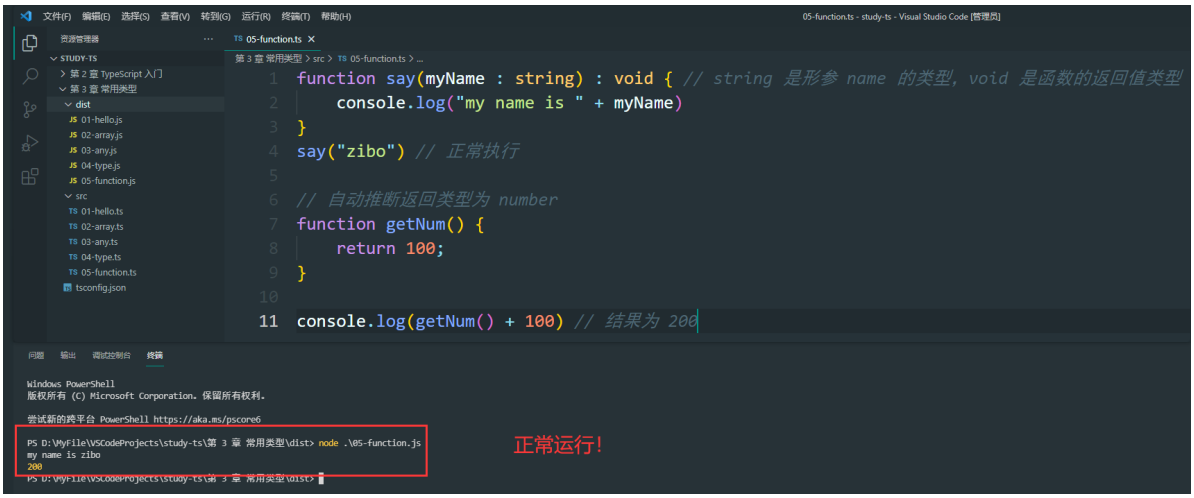
```
function say(myName : string) : void { // string 是形参 name 的类型，void 是函数的返回值类型
    console.log("my name is " + myName)
}
```

2、代码演示

代码示例与解释

```
function say(myName : string) : void { // string 是形参 name 的类型，void 是函数的返回值类型
    console.log("my name is " + myName)
}
say("zibo") // 正常执行

// 自动推断返回类型为 number
function getNum() {
    return 100;
}

console.log(getNum() + 100) // 结果为 200
```

执行结果



3、匿名函数

代码示例

```
let people = ["大哥", "二哥", "三哥"] // 此处的 people 自动推断为 string 数组类型
// 遍历
people.forEach(p => { // 此处的 p 自动推断为 string 类型
    console.log(p)
})
```

执行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 3 章 常用类型\dist> node .\05-function2.js
大哥
二哥
三哥
```

六、对象类型

1、概述

什么是对象

带有任何属性的 JavaScript 值都可以看做是对象！

代码分析

```
// printCoord 参数的类型是一个对象
// 对象的两个属性可以使用逗号（,）或分号（;）分隔
function printCoord(pt: { x: number; y: number }) {
    console.log("x 的值为" + pt.x)
    console.log("y 的值为" + pt.y)
}
// 传入对象
printCoord({x: 3, y: 7})
// 当然，也可以这么写
let pt = {x: 3, y: 7}
printCoord(pt)
```

2、可选属性

含义

如上代码，属性 x 和 y 不一定都要传入，在可选参数后面加一个问号（?）即可实现！

代码示例

```
// 在参数对象属性后面加一个问号（?）就表示是可选属性
function printCoord(pt: { x: number; y?: number }) {
    console.log("x 的值为" + pt.x)
    console.log("y 的值为" + pt.y)
}
// 传入 x 和 y
printCoord({x: 3, y: 7})
// 只传入 x
printCoord({x: 3})
```

运行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 3 章 常用类型\dist> node .\06-obj.js
x 的值为3
y 的值为7
x 的值为3
y 的值为undefined # 这是未传入 y 的情况
```

3、使用可选属性

注意点

使用可选属性之前需要**判断**其是否传入！

两个判断方法

```
// 在参数对象属性后面加一个问号（?）就表示是可选属性
function printCoord(pt: { x: number; y?: string }) {
    console.log("x 的值为" + pt.x)
    // 判断方法一
    if(pt.y !== undefined){
        console.log("判断方法一：传入了 y ！")
        console.log("1 y 的值为" + pt.y)
    }
    // 判断方法二：仅与方法一写法不同，含义一致！
    if(pt.y){
        console.log("判断方法二：传入了 y ！")
        console.log("2 y 的值为" + pt.y)
    }
    // 判断方法三
    console.log("判断方法三：必须执行！")
    // 这里没有问号会报错：对象可能为“未定义”。！
    console.log("3 y 的值为" + pt.y?.toUpperCase())

}
// 传入 x 和 y
printCoord({x: 3, y: '7'})
// 只传入 x
printCoord({x: 3})
```

执行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 3 章 常用类型\dist> node .\06-obj.js
x 的值为3
判断方法一：传入了 y ！
1 y 的值为7
判断方法二：传入了 y ！
2 y 的值为7
判断方法三：必须执行！
3 y 的值为7
x 的值为3
判断方法三：必须执行！
3 y 的值为undefined
```

七、联合类型

1、概述

含义

有时候一个属性可以是 number， 也可以是 string， 我们可以使用 `number | string` 来表示**联合类型**!

联合类型可以是**两个或多个**类型的联合!

代码分析

```
let p: number | string = 100
p = "大哥刘备! " // 不会报错!
```

2、函数中使用

代码示例

```
function printSth(x: number | string | boolean){
    if(typeof x === "number"){
        console.log("当数字用! ")
    }else if(typeof x === "string"){
        console.log("当字符串用! ")
    }else{
        console.log("当数布尔值用! ")
    }
}
printSth(100)
printSth('100')
printSth(true)
```

执行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 3 章 常用类型\dist> node .\07-uni.js
当数字用!
当字符串用!
当数布尔值用!
```

3、字符串与字符串数组的判断

代码演示

```ts
function printSth(x: string | string[]){
    if(Array.isArray(x)){ // 数组
        console.log(x.join(" and "))
    }else {
        console.log(x)
    }
}
printSth("大哥")
printSth(["大哥", "二哥", "三哥"])
```

执行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 3 章 常用类型\dist> node .\07-uni.js
大哥
大哥 and 二哥 and 三哥
```

结论

联合类型使得类型"**变宽**"，但使用的时候必须进行判断！

4、调用共有方法

代码示例

```ts
function printSth(x: number[] | string){
    // slice() 方法是数组和字符串共有的方法，因此无需判断
    console.log(x.slice(0, 3))
}
printSth("123456")
printSth([1, 2, 3, 4, 5, 6])
```

执行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 3 章 常用类型\dist> node .\07-uni.js
123
[ 1, 2, 3 ]
```

八、类型别名

1、概述

含义

类型别名就是**给类型一个名字**，以**多次使用**它！

代码示例

```
// 1、对象类型
// 分隔符号说明：可以使用英文分号（;）、逗号（,）或者什么也不写！
type Student = {
    name: string,
    age: number
}
// 2、联合类型
type ID = number | string
// 3、一般类型（其他所有类型）
type myString = string
```

2、使用

代码示例及解释

```
// 1、定义对象类型
type Student = {
    name: string,
    age: number
}
// 2、定义函数
function printStudentName(stu: Student){
    console.log(stu.name)
}
// 3、调用函数
printStudentName({
    // 注意这里必须是使用逗号（,）隔开
    name: "大哥刘备",
    age: 22
})
```

生成的 js 代码

```
"use strict";
// 2、定义函数
function printStudentName(stu) {
    console.log(stu.name);
}
// 3、调用函数
printStudentName({
    // 注意这里必须是使用逗号（,）隔开
    name: "大哥刘备",
    age: 22
});
```

执行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 3 章 常用类型\dist> node .\08-alias.js
大哥刘备
```

九、接口

1、概述

通常情况下，我们将 TypeScript 称之为**结构类型**的类型系统，其原因就是 TypeScript 只关心类型的**结构和功能**，**接口**就是一种**结构类型**，它是定义**对象类型**的另外一种方式。通过关键字 `interface` 定义！

2、使用

代码示例及解释

```
// 1、定义接口
interface Point {
    x: number,
    y: number
}
// 2、定义函数
function print(p: Point){
    console.log("x坐标是" + p.x)
    console.log("y坐标是" + p.y)
}
// 3、调用函数
print({x: 100, y: 200})
```

生成的 js 代码

```
// 2、定义函数
function printPoint(p) {
    console.log("x坐标是" + p.x);
    console.log("y坐标是" + p.y);
}
// 3、调用函数
printPoint({ x: 100, y: 200 });
```

执行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 3 章 常用类型\src> node .\09-interface.js
x坐标是100
y坐标是200
```

3、扩展接口与类型别名

> 几乎所有通过 interface 接口定义的类型都可以使用 type 类型别名进行定义！

扩展接口代码示例与解释

```
// 1、定义接口
interface Point {
    x: number,
    y: number
}
// 2、扩展接口
interface BigPoint extends Point {
    z: number
}
// 3、定义函数
function printBigPoint(p: BigPoint) {
    console.log("x坐标是" + p.x)
    console.log("y坐标是" + p.y)
    console.log("z坐标是" + p.z)
}
// 4、调用函数
printBigPoint({ x: 100, y: 200, z: 300 })
```

生成的 js 代码

```
// 3、定义函数
function printBigPoint(p) {
    console.log("x坐标是" + p.x);
    console.log("y坐标是" + p.y);
    console.log("z坐标是" + p.z);
}
// 4、调用函数
printBigPoint({ x: 100, y: 200, z: 300 });
```

扩展类型别名代码示例与解释

```
// 1、定义对象类型
type Student = {
    name: string,
    age: number
}
// 2、扩展对象类型
type BigStudent = Student & {
    tall: number
}

// 3、定义函数
function printBigStudent(stu: BigStudent){
    console.log(stu.name)
    console.log(stu.age)
    console.log(stu.tall)
}
// 4、调用函数
printBigStudent({
    // 注意这里必须是使用逗号（,）隔开
    name: "大哥刘备",
    age: 22,
    tall: 178
})
```

生成的 js 代码

```
// 3、定义函数
function printBigStudent(stu) {
    console.log(stu.name);
    console.log(stu.age);
    console.log(stu.tall);
}
// 4、调用函数
printBigStudent({
    // 注意这里必须是使用逗号（,）隔开
    name: "大哥刘备",
    age: 22,
    tall: 178
});
```

4、向现有类型添加字段

接口代码示例及解释

```
// 1、定义接口
interface Point {
    x: number,
    y: number
}
// 2、添加字段
interface Point {
    z: number
}
// 3、定义函数
function printPoint(p: Point) {
    console.log("x坐标是" + p.x)
    console.log("y坐标是" + p.y)
    console.log("z坐标是" + p.z)
}
// 4、调用函数
printPoint({ x: 100, y: 200, z: 300 })
```

类型别名说明

类型**无法实现**此功能，类型创建之后**无法更改**！

十、类型断言

1、概述

含义

有时候获得一个值

TypeScript 并不知道是什么类型

但是我们可能知道！

此时我们可以使用**类型断言**来指定类型！

示例见**代码分析**。

代码分析

```
// 返回某种类型的 HTMLElement ，使用 as 进行断言
cosnt myCanvas = document.getElementById("main_canvas")
// 断言方式一
cosnt myCanvas = document.getElementById("main_canvas") as HTMLCanvasElement
// 断言方式二（与方式一等效）
cosnt myCanvas = <HTMLCanvasElement>document.getElementById("main_canvas")
```

2、将字符串断言为数字类型

> 事实上，TypeScript 只允许将类型断言转换为更具体或不太具体的类型！

代码分析

```
const myStr = "hello" as number // 报错：类型 "string" 到类型 "number" 的转换可能是错误的，因为两种类型不能充分重叠。如果这是有意的，请先将表达式转换为 "unknown"。
// 正确的方式
const myStr = ("hello" as unknown) as number
```

十一、文字类型

1、概述

含义

JavaScript 可以使用 let 和 const 来声明类型

一种是可以改变，一种是不可改变

这反应在 TypeScript 就是如何为文字创建类型。

> 所谓文字类型，就是将一段文字作为类型！

代码分析

```
let testString = "Hello World"
testString = "hello typescript"
// testString 可以表示任何可能的字符串

const constantString = "Hello World"
// testStriconstantStringng 只能表示一个可能的字符串
```

2、文字类型的使用

基本使用

```
// 1、使用 const 声明常量
const newStr = "hello world"
newStr = "hello" // 报错：无法分配到 "newStr" ，因为它是常数。
// 2、创建文字类型、
let aStr: "hello" = "hello"
aStr = "hello"
aStr = "world" // 报错：不能将类型“"world"”分配给类型“"hello"”。
```

在函数中使用

> 文字类型的使用场景和还是很多的！所谓的文字不仅仅是字符串，也可以是数字等！

```
// 1、声明函数
function say(name: string, content: "早安！" | "午安！" | "晚安！"){
    console.log(name + "说：" + content)
}
// 2、调用函数
say("訾博", "早安！")
say("訾博", "大家好！") // 报错：类型“"大家好！"”的参数不能赋给类型“"早安！" | "午安！" | "晚安！"”的参数。
```

3、类型推断引起的我问题

问题代码分析

```
// 1、声明函数
function say(name: string, content: "早安！" | "午安！" | "晚安！"){
    console.log(name + "说：" + content)
}
// 2、定义常量
const person = {
    name: "訾博",
    content: "早安！"
}
// 3、调用函数
// 报错：类型“string”的参数不能赋给类型“"早安！" | "午安！" | "晚安！"”的参数。
say(person.name, person.content)
```

解决问题：类型断言

> 下面三种方式使用其中一种即可！

```
// 1、声明函数
function say(name: string, content: "早安！" | "午安！" | "晚安！"){
    console.log(name + "说：" + content)
}
// 2、定义常量
const person = {
    name: "訾博",
    content: "早安！" as "早安！" // 方式1
} as const // 方式3
// 3、调用函数
// 报错：类型“string”的参数不能赋给类型“"早安！" | "午安！" | "晚安！"”的参数。
say(person.name, person.content as "早安！") // 方式2
```

十二、null 和 undefined 类型

1、概述

null 表示**不存在**

undefined 表示**未初始化的值**

TypeScript 有两个与之对应的类型！

2、代码演示

代码示例及解释

```
let a = undefined
let b = null
// 上面都是自动推断为相应类型了
// let c: string = undefined // 报错（严格模式）：不能将类型“undefined”分配给类型“string”。
// 在函数中使用
function doSth(age: number | null){
    // console.log(age.toFixed) // 报错：对象可能为 "null"。
    // 解决方法1：如果不为 null ，则执行
    console.log(age?.toFixed)
    // 解决方法2：断言不为 null
    console.log(age!.toFixed)
}
```

十三、枚举

1、概述

枚举是 TypeScript 添加到 JavaScript 中的一项功能！

这个值可能是一组命名常量之一！

与大多数 TypeScript 功能不同

这个不是 JavaScript 类型级别里面添加的内容

而是添加到 TypeScript 语言和运行时的内容

只有**确定确实需要**枚举来做些事情的时候

否则没有必要使用它!

2、代码示例及解释

```
enum Fruit {
    apple = 1,
    // 下面的值自动增加 1
    pear,
    peach
}
console.log(100 + Fruit.peach) // 103
```

十四、不太常用的原语

1、概述

JavaScript 一些不太常用的原语在 TypeScript 中也实现了

我们来学习两个：bigint（非常大的整数） 和 symbol（全局唯一引用）。

2、代码示例

```
// 注意：目标低于 ES2020 时，bigInt 文本不可用。
// 需要将 tsconfig.json 下的 target 的值修改为 ES2020
const big100: bigint = BigInt(100)
const big1000: bigint = 1000n

const firstName = "訾博"
const lastName = "訾博"
console.log(firstName === lastName) // true

const firstName1 = Symbol("訾博")
const lastName1 = Symbol("訾博")
console.log(firstName1 === lastName1) // 直接报错：此条件将始终返回 "false"，因为类型 "typeof firstName1" 和 "typeof lastName1" 没有重叠。
```