【TypeScript 4.5】004-第 4 章 类型缩小

一、typeof 类型守卫

1、什么是类型缩小

含义

TypeScript 类型缩小就是**从宽类型转化为窄类型**的过程

类型缩小常用于处理**联合类型变量**的场景

代码分析

```
function padLeft(padding: number | string, input: string): string {
    return new Array(padding + 1).join(" ") + input // 报错：运算符“+”不能应用于类型“string | number”和“number”。
}
```

2、使用 typeof 进行代码改造

改造后的代码

```
function padLeft(padding: number | string, input: string): string {
    if(typeof padding === "number") {
        return new Array(padding + 1).join(" ") + input
    }
    return padding + input
}
console.log(100, "哈哈哈")
console.log("大哥", "刘备")
```

执行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 4 章 类型缩小\dist> node .\01-typeof.js
100 哈哈哈
大哥 刘备
```

3、typeof 类型守卫

概述

返回当前类型的字符串表示。

使用示例

```
typeof a === "object"
// 除了 object ，还有 string、number、bigint、boolean、symbol、undefined、function
```

问题代码示例

```
function printAll(strs: string | string[] | null): string {
    // 需要说明的是当 strs 的值为 string[] 类型的时候返回的是 "object"
    // 而当 strs 的值为 null 的时候返回的也是 "object"
}
```

二、真值缩小

1、概述

说明

真值检查是在 JavaScript 中经常要做的事情

我们可以使用条件、&&、||、布尔否定（!）来进行真值检查

代码分析

```
function getUserOnlineMessage (numUsersOnline: number) {
    if (numUsersOnline) { // 如果此处 numUsersOnline 的值为0、NaN、""（空字符串）、0n（bigint零的版本）、null、undefined，则为 false
        return `现在共有 ${numUsersOnline} 人在线！`
    }
    return "现在无人在线！"
}
// 下面两种结果都返回 true ，值与 if() 括号里面的值判断标准一致！
Boolean("hello")
!!"world" // 一个!将其转换为文字类型，里昂一个!将其转换为布尔类型！
```

2、解决 typeof 类型守卫中的问题

```
function printAll(strs: string | string[] | null) {
    // 要判断 strs 是 string[] 且不是 null， 可以这么写！
    if(strs && typeof strs === "object"){
        // ...
    }
}
```

三、等值缩小

1、说明

TypeScript 也可以使用分支语句做全等（===）、全不等（!==）、等于（==）、不等于（!=）来做等值检查，实现类型缩小。

2、代码示例

> 真是过于简单了！都不想写代码示例了！

```
// 全等
function doSth(str1: string | null, str2: string | number) {
    if (str1 === str2) {
        str1.toUpperCase()
        str2.toLowerCase()
    }
}

// 不等（这一点需要注意）
function doSth1(value: number | null | undefined, x: number) {
    if(value != null){ // 这里的 null 替换成 undefined 结果也是一样的！
        value *= x
        console.log(value)
    }
}
doSth1(100, 5) // 500
doSth1(null, 5) // 什么也不打印
doSth1(undefined, 5) // 注意：这个也是什么也不打印！已自动将其过滤！
```

四、in 操作符缩小

1、概述

说明

JavaScript 有个运算符，**用来确定对象是否具有某个名称的属性**，这个运算符就是 in 运算符！

代码分析

```
// 格式
value in X
// value 为字符串（表示属性名）
// 结果若为 true ，要求 X 具有可选或必需属性的类型的值
// 结果若为 false，要求 X 具有可选或缺失属性的类型的值
```

2、代码演示

代码示例及解释

> 结果若为 true ，要求 X 具有可选或必需属性的类型的值！

```
type Fish = {
    swim: () => void
}
type Bird = {
    fly: () => void
}
function move(animal: Fish | Bird){
    if("swim" in animal){
        return animal.swim
    }
    return animal.fly
}
```

> 再加入一个 People 类型，使其具有两者的（可选）属性！

```
type Fish = {
    swim: () => void
}
type Bird = {
    fly: () => void
}
type People = {
    swim?: () => void,
    fly?: () => void
}
function move(animal: Fish | Bird | People){
    if("swim" in animal){
```

```
        // animal: Fish | People
        // 我们可以将其断言为 Fish
        return (animal as Fish).swim
    }
    // animal: Bird | People
    // 我们可以将其断言为 Bird
    return (animal as Bird).fly
}
```

> 也可以不用断言，在其有该方法的时候才执行！

```
type Fish = {
    swim: () => void
}
type Bird = {
    fly: () => void
}
type People = {
    swim?: () => void,
    fly?: () => void
}
function move(animal: Fish | Bird | People) {
    if ("swim" in animal) {
        // animal: Fish | People
        return animal?.swim
    }
    // animal: Bird | People
    return animal?.fly
}
```

五、instanceof 操作符缩小

1、概述

说明

JavaScript 使用 instanceof 操作符来检查一个值**是否是另一个值的实例**

instanceof 也是一个**类型保护**

TypeScript 在由 instanceof 保护的分支中来实现**类型缩小**

代码分析

```
X instanceof Foo // 用来检查 X 的原型链是否含有 Foo.prototype
```

2、代码演示

代码示例

```
function logValue(x: Date | string) {
    if (x instanceof Date) {
        console.log(x.toUTCString())
    } else {
        console.log(x.toUpperCase())
    }
}
logValue(new Date())
logValue("hello world")
```

执行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 4 章 类型缩小\dist> node .\05-instanceof.js
Mon, 07 Feb 2022 01:08:07 GMT
HELLO WORLD
```

六、分配缩小

1、概述

说明

当我们为任何变量赋值的时候

TypeScript 会查看赋值的右侧

并适当缩小左侧

代码示例

```
// let x: string | number
let x = Math.random() < 0.5 ? 10 : "hello world"
```

2、代码演示

代码示例及解释

```
// let x: string | number （自动推断为联合类型）
let x = Math.random() < 0.5 ? 10 : "hello world"
// let x: number
x = 1
console.log(typeof x)
console.log(x)
// let x: string
```

```
x = "good morning"
console.log(typeof x)
console.log(x)
// 手动分配测试
let y: number | string
y = 100
console.log(typeof y)
console.log(y)
y = "good morning"
console.log(typeof y)
console.log(y)
// 结论：与分配自动推断结果一致！
```

执行结果

> 此处变成了具体的类型而不是联合类型，可参考下面的控制流分析！

```
PS D:\MyFile\VSCodeProjects\study-ts\第 4 章 类型缩小\dist> node .\06-assignment.js
number
1
string
good morning
number
100
string
good morning
```

七、控制流分析

1、概述

说明

基于可达性的代码分析即控制流分析！见代码示例！

代码示例

```
function padLeft(padding: number | string, input: string) {
    if (typeof padding === "number") {
        return new Array(padding + 1).join(" ") + input
    }
    return padding + input
}
```

2、代码演示

代码示例及解释

```
function test() {
    let x: number | string | boolean
    x = Math.random() < 100
    // let x: boolean
    console.log(typeof x)
    console.log(x)
    if(Math.random() < 100) {
        x = "hello world"
        // let x: string
        console.log(typeof x)
        console.log(x)
    } else {
        x = 100
        // let x: number
        console.log(typeof x)
        console.log(x)
    }
    return x
}
let q = test()
q = 100
q = "hi"
// q = true // 报错：不能将类型“boolean”分配给类型“string | number”。
// 说明此处将函数中 x 的值推断为 string | number ，下面的判断将上面的 boolean 类型覆盖了！
console.log(typeof q)
console.log(q)
```

执行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 4 章 类型缩小\dist> node .\07-ctrl.js
boolean
true
string
hello world
string
hi
```

八、使用类型谓词

1、概述、

说明

有时候我们想**直接控制**整个代码的类型变化

为了定义一个用户定义的类型保护

我们只需要定义一个函数

并使其返回值是一个类型谓词即可

代码示例

> pet is Fish 就是所谓的类型谓词，意思是：如果 pet 里面有 swim 这个属性，pet 就是 Fish 类型！**格式：参数名 is 类型**

```
function isFish (pet: Fish | Bird): pet is Fish {
    return (pet as Fish).swim !== undefined
}
```

2、代码演示

代码示例及解释

```
type LiuBei = {
    name: string,
    fight: boolean
}
type ZiBo = {
    name: string,
    code: boolean
}
function isMe(people: LiuBei | ZiBo): people is ZiBo {
    return (people as ZiBo).name === "訾博" && (people as ZiBo).code === true
}
function getPeople(): ZiBo | LiuBei {
    if(Math.random() > 0.5){
        return { name: "訾博", code: true }
    } else {
        return { name: "刘备", fight: true }
    }
}
let people = getPeople()
// 此时如果 isMe(people) 返回为 true ，TypeScript 就知道 me 的类型是 ZiBO ，反之为 LiuBei
if(isMe(people)){
    console.log(people.name)
    console.log(people.code)
} else {
    console.log(people.name)
    console.log(people.fight)
}
// 下面是深度一点的使用
const somePeople: (ZiBo | LiuBei)[] = [getPeople(), getPeople(), getPeople(), getPeople(), getPeople()]
// 过滤出 ZiBo ，下面两行代码是等价的!
const zibo1: ZiBo[] = somePeople.filter(isMe)
const ZIBO2: ZiBo[] = somePeople.filter(isMe) as ZiBo[]
// 写得更复杂点!
const zibo3: ZiBo[] = somePeople.filter((people): people is ZiBo => { // people 的小括号是一定要带的
    if (people.name === "刘备"){
        return false
```

```
    }
    return isMe(people)
})
```

执行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 4 章 类型缩小\dist> node .\09-type.js
刘备
true
[ { name: '訾博', code: true }, { name: '訾博', code: true } ]
[ { name: '訾博', code: true }, { name: '訾博', code: true } ]
[ { name: '訾博', code: true }, { name: '訾博', code: true } ]
```

九、受歧视的 unions

1、概述

union 即联合类型

我们一直在用简单的类型来缩小单个变量

但 JavaScript 中大多数处理的是稍微复杂的结构

2、代码演示

发现问题

```
// 圆形与方形
interface Shape {
    kind: "circle" | "square",
    radius?: number,
    sideLength?: number
}
// 计算圆的面积
function getArea (shape: Shape) {
    if(shape.kind === "circle"){
        // 【问题】这么写看上去很理想，但是无法保证 kind 为 circle 的时候 radius 就一定是已定义的!
        return Math.PI * shape.radius! ** 2
    }
}
```

解决问题

```
// 圆形与方形
interface Circle {
    kind: "circle",
    radius: number
}
interface Square {
    kind: "square",
    sideLength: number
}
type Shape = Circle | Square
// 计算圆的面积
function getArea (shape: Shape) {
    if(shape.kind === "circle"){
        return Math.PI * shape.radius ** 2
    }
}
console.log(getArea({kind: "circle", radius: 3}))
```

执行结果

```
PS D:\MyFile\VSCodeProjects\study-ts\第 4 章 类型缩小\dist> node .\010-unions.js
28.274333882308138
```

十、never 类型与穷尽性检查

1、概述

在缩小范围的时候

我们可以将联合体的选项减少

直到删除了所有可能性

这个时候我们使用 never 类型表示

never 类型表示不应该存在的状态

never 可以分配给任何类型

但没有任何类型可以分配给 never

除了 never 本身

2、代码示例及解释

```
// 圆形、方形、三角形
interface Circle {
    kind: "circle",
    radius: number
```

```ts
}
interface Square {
    kind: "square",
    sideLength: number
}
interface Triangle {
    kind: "triangle",
    sideLength: number
}
type Shape = Circle | Square | Triangle
// 计算圆的面积
function getArea (shape: Shape) {
    switch(shape.kind){
        case "circle":
            return Math.PI * shape.radius ** 2
        case "square":
            return shape.sideLength ** 2
        default:
            // 报错：不能将类型“Triangle”分配给类型“never”。
            // 报错意味着还有可能，也就做了穷尽性检查！
            const _exhaustiveCheck: never = shape
            return _exhaustiveCheck
    }
}
```