对声明合并的爱与恨

更新时间：2019-07-12 09:24:27

> 理想的书籍是智慧的钥匙。
>
> ——列夫·托尔斯泰

声明合并是指 TypeScript 编译器会将名字相同的多个声明合并为一个声明，合并后的声明同时拥有多个声明的特性。我们知道在 JavaScrip 中，使用var关键字定义变量时，定义相同名字的变量，后面的会覆盖前面的值。使用let定义变量和使用 const 定义常量时，不允许名字重复。在 TypeScript 中，接口、命名空间是可以多次声明的，最后 TypeScript 会将多个同名声明合并为一个。我们下来看个简单的例子：

```
interface Info {
  name: string
}
interface Info {
  age: number
}
let info: Info
info = { // error 类型"{ name: string; }"中缺少属性"age"
  name: 'lison'
}
info = { // right
  name: 'lison',
  age: 18
}
```

可以看到，我们定义了两个同名接口Info，每个接口里都定义了一个必备属性，最后定义info类型为Info时，info的定义要求同时包含name和age属性。这就是声明合并的简单示例，接下来我们详细学习。

4.3.1. 补充知识

TypeScript的所有声明概括起来，会创建这三种实体之一：**命名空间、类型和值**：

- 命名空间的创建实际是创建一个对象，对象的属性是在命名空间里export导出的内容；

- 类型的声明是创建一个类型并赋给一个名字；
- 值的声明就是创建一个在JavaScript中可以使用的值。

下面这个表格会清晰的告诉你，每一种声明类型会创建这三种实体中的哪种，先来说明一下，第一列是指声明的内容，每一行包含4列，表明这一行中，第一列的声明类型创建了后面三列哪种实体，打钩即表示创建了该实体：

声明类型	创建了命名空间	创建了类型	创建了值
Namespace	√		√
Class		√	√
Enum		√	√
Interface		√	
Type Alias类型别名		√	
Function			√
Variable			√

可以看到，只要命名空间创建了命名空间这种实体。Class、Enum两个，Class即是实际的值也作为类使用，Enum编译为JavaScript后也是实际值，而且我们讲过，一定条件下，它的成员可以作为类型使用；Interface和类型别名是纯粹的类型；而Funciton和Variable只是创建了JavaScript中可用的值，不能作为类型使用，注意这里Variable是变量，不是常量，常量是可以作为类型使用的。

4.3.2. 合并接口

我们在本节课一开始的例子中，简单示范了一下接口声明的合并，下面我们来补充一些内容。

多个同名接口，定义的非函数的成员命名应该是不重复的，如果重复了，类型应该是相同的，否则将会报错。

```
interface Info {
    name: string
}
interface Info {
    age: number
}
interface Info {
    age: boolean // error 后续属性声明必须属于同一类型。属性“age”的类型必须为“number”，但此处却为类型“boolean”
}
```

对于函数成员，每个同名函数成员都会被当成这个函数的重载，且合并时后面的接口具有更高的优先级。来看下多个同名函数成员的例子：

```
interface Res {
    getRes(input: string): number
}
interface Res {
    getRes(input: number): string
}
const res: Res = {
    getRes: (input: any): any => {
        if (typeof input === 'string') return input.length
        else return String(input)
    }
}
res.getRes('123').length // error 类型“number”上不存在属性“length”
```

4.3.3. 合并命名空间

同名命名空间最后会将多个命名空间导出的内容进行合并，如下面两个命名空间：

```
namespace Validation {
    export const checkNumber = () => {}
}
namespace Validation {
    export const checkString = () => {}
}
```

上面定义两个同名命名空间，效果相当于：

```
namespace Validation {
    export const checkNumber = () => {}
    export const checkString = () => {}
}
```

在命名空间里，有时我们并不是把所有内容都对外部可见，对于没有导出的内容，在其它同名命名空间内是无法访问的：

```
namespace Validation {
    const numberReg = /^[0-9]+$/
    export const stringReg = /^[A-Za-z]+$/
    export const checkString = () => {}
}
namespace Validation {
    export const checkNumber = (value: any) => {
        return numberReg.test(value) // error 找不到名称“numberReg”
    }
}
```

上面定义的两个命名空间，numberReg没有使用export导出，所以在第二个同名命名空间内是无法使用的，如果给 const numberReg 前面加上 export，就可以在第二个命名空间使用了。

4.3.4. 不同类型合并

命名空间分别和类、函数、枚举都可以合并，下面我们来一一说明：

(1) 命名空间和类

这里要求同名的类和命名空间在定义的时候，类的定义必须在命名空间前面，最后合并之后的效果，一个包含一些以命名空间导出内容为静态属性的类，来看例子：

```
class Validation {
    checkType() { }
}
namespace Validation {
    export const numberReg = /^[0-9]+$/
    export const stringReg = /^[A-Za-z]+$/
    export const checkString = () => { }
}
namespace Validation {
    export const checkNumber = (value: any) => {
        return numberReg.test(value)
    }
}
console.log(Validation.prototype) // { checkType: fun () {} }
console.log(Validation.prototype.constructor)
/**
{
    checkNumber: ...
    checkString: ...
    numberReg: ...
    stringReg: ...
}
*/
```

(2) 命名空间和函数

在JavaScript中，函数也是对象，所以可以给一个函数设置属性，在TypeScript中，就可以通过声明合并实现。但同样要求，函数的定义要在同名命名空间前面，我们再拿之前讲过的计数器的实现来看下，如何利用声明合并实现计数器的定义：

```
function countUp () {
    countUp.count++
}
namespace countUp {
    export let count = 0
}
countUp()
countUp()
console.log(countUp.count) // 2
```

(3) 命名空间和枚举

可以通过命名空间和枚举的合并，为枚举拓展内容，枚举和同名命名空间的先后顺序是没有要求的，来看例子：

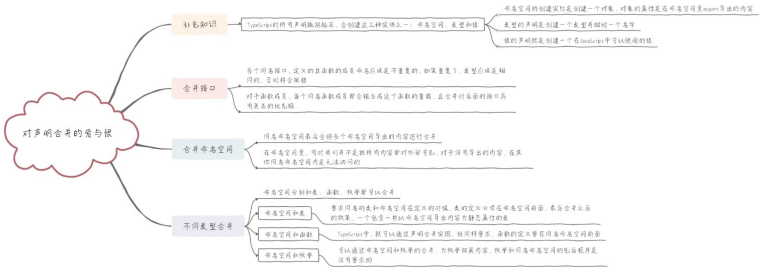
```
enum Colors {
    red,
    green,
    blue
}
namespace Colors {
    export const yellow = 3
}
console.log(Colors)
/*
{
    0: "red",
    1: "green",
    2: "blue",
    red: 0,
    green: 1,
    blue: 2,
    yellow: 3
}
*/
```

通过打印结果你可以发现，虽然我们使用命名空间增加了枚举的成员，但是最后输出的值只有key到index的映射，没有index到key的映射。

小结

本小节我们学习了编译器对于相同命名的声明的合并策略，这个策略能够帮我们实现一些类型定义的复用，比如多个函数定义可合并为一个函数的重载，还可以利用声明合并实现一些复杂的类型定义。但是有时我们会无意地定义了一个之前定义过的名字，造成声明合并了，再使用这个新定义的时候，发现应用了一些这里未定义的类型校验，所以我们在定义名字的时候要注意这一点。本小节我们讲了接口、命名空间、不同类型是如何合并的，也学习了如何利用声明合并来定义复杂的数据结构。

下个小节我们来学习混入，在使用一些框架或者插件的时候，你可能听过这个概念，我们可以将公共逻辑抽取出来，然后通过混入实现复用。在TypeScript中，混入还需要考虑类型，所以我们下个小节来看下如何在TypeScript中实现混入。



使用命名空间封装代码

混入，兼顾值和类型的合并操作

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示

目前暂无任何讨论