link: null
title: 珠峰架构师成长计划
description: 一种类型可以通过以下形式被引入
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=21 sentences=22, words=239

13 复杂声明

13.1 类型（Types）

一种类型可以通过以下形式被引入

- 类型别名声明： ·type t = number | string;
- 接口声明： interface I { x: number[]; }
- 类声明： class C { }
- 枚举声明： enum E { A, B, C }
- import 声明指向一个类型

13.2.值(Values)

Value是我们可以在表达式中引用的运行时名称,可以通过以下的形式创建value

- let、const、var 声明
- namespace 或 module 声明包含一个 Value
- enum 声明
- class 声明
- import 声明指向一个值
- function 声明

13.3.命名空间（Namespaces）

- 类型可以存在于命名空间中。例如：有 let x: A.B.C 声明，我们说类型 C 来自命名空间 A.B

13.4.一个名字，多重含义

- A首先用作命名空间，然后用作类型名称，然后用作值

```
let m: A.A = A;
```

13.5.内建组合

- class 同时出现在 type 和 value 清单中,声明 class C {} 创建了两项内容
    - 类型 C是类 C的实例原型
    - 值 C是类 C的构造函数

```
export var Bar: { default: Bar } = { default: { count: 10 } };
export interface Bar {
    count: number;
}
```

```
import { Bar } from './bar';
console.log(Bar);
let x: Bar = Bar.default;
console.log(x.count);
```

13.6. 例子

```
namespace X {
  export interface Y { }
  export class Z { }
}
```

- 值 X（因为 namespace 声明包含了值 Z）
- 命名空间 X（因为 namespace 声明包含了类型 Y）
- 类型 Y 在命名空间 X 中
- 类型 Z 在命名空间 X 中（类的实例原型）
- 值 Z,属于 X 值的属性（类的构造器）

```
namespace X {
  export var Y: number;
  export namespace Z {
    export class C { }
  }
}
type X = string;
```

- 值 Y (number 类型，X 值的属性)
- 命名空间 Z
- 值 Z (X 值的属性)
- 类型 C (在命名空间 X.Z 中)
- 值 C (X.Z 值的属性)
- 类型 X