## 类型声明

类型声明是TS非常重要的一个特点。

通过类型声明可以**指定TS中变量（参数、形参）的类型。**

指定类型后，当为变量赋值时，TS编译器**会自动检查值是否符合类型声明**，符合则赋值，否则报错。

简而言之，类型声明给变量设置了类型，使得变量只能存储某种类型的值。

语法：

let 变量: 类型; // 声明变量及类型。

let 变量: 类型 = 值; //声明变量后直接进行赋值。

// 函数中参数类型和返回值类型

function fn(参数: 类型, 参数: 类型): 类型{

...

}

示例1：

// 声明一个变量a，同时指定它的类型为number

let a: number;

// a 的类型设置为了number，在以后的使用过程中a的值只能是数字

a = 10;

a = 33;

a = 'hello'; //此行代码会报错，因为变量a的类型是number，不能赋值字符串

// 如果变量的声明和赋值是同时进行的，TS可以自动对变量进行类型检测

let c = false; // TS自动检测为Boolean类型

c = true;

c = 123;//此时会报错

示例2：

// JS中的函数是不考虑参数的类型和个数的

//错误示例

function sum(a, b){

return a + b;

}

// 这种情况下调用函数时怎么传参数，都不会报错

console.log(sum(123, 456)); //输出结果为579

console.log(sum(123, "456")); //"123456"

// 给传入的参数、返回值 加上类型声明

function sum(a: number, b: number): number{

return a + b;

}

sum(123, "456");// 这时会报错，不允许传入字符串

sum(123); // 传少了参数，也会报错

**注：**TS在有报错时执行编译仍然会生成编译文件，**有无报错不影响编译**。但是后续可以通过**配置文件**增加字段来选择让其有报错时不编译。

**自动类型判断**

TS拥有自动的类型判断机制

当对变量的声明和赋值是同时进行的，TS编译器会自动判断变量的类型

所以如果你的变量的声明和赋值是同时进行的，可以省略掉类型声明

## 类型

|  |  |  |
| --- | --- | --- |
| 类型 | 例子 | 描述 |
| number | 1，-22,2.8 | 任意数字 |
| string | 'hi', "hello" | 任意字符串 |
| boolean | true、false | 布尔值true或false |
| 字面量 | 其本身 | 限制变量的值就是该字面量的值 |
| any | \* | 任意类型 |
| unknown | \* | 类型安全的any |
| void | 空值（undefined） | 没有值（或undefined） |
| never | 没有值 | 不能是任何值 |
| object | {name: '孙悟空'} | 任意的JS对象 |
| array | [1,2,3] | 任意JS数组 |
| tuple | [4,5] | 元素，TS新增类型，固定长度数组 |
| enum | enum{A, B} | 枚举，TS中新增类型 |

### 字面量

// 直接使用字面量进行类型声明

let a: 10;

a = 10; // √

a = 11; // ×，报错

// 可以使用 | 来连接多个类型（联合类型）

let b: "male" | "female";

b = "male";

b = "female"

let c: boolean | string;

c = true

c = "hello";

### any（不建议使用）

// any 表示的是任意类型，一个变量设置类型为any后相当于对该变量关闭了TS的类型检测

// 声明变量如果不指定类型，则TS解析器会自动判断变量的类型为any（隐式的any）

let d: any; //显式

let d; //隐式

d = 10;

d = 'hello';

d = true;

//使用TS时，不建议使用any类型

### unknown

// unknown 表示未知类型的值

let e: unknown;

e = 10;

e = "hello"

**注：any类型的变量，可以赋值给任意变量**。 例如 d 为any类型，将它赋值给s，s = d;

此时不仅any自身的类型检查是关闭的，被赋值的变量的类型检查也会关闭。

**unknown类型是类型安全的any，是不能直接赋值给其他变量的**，否则会报错。

但可以**①在值类型与变量类型相同时通过判断语句将unknown类型的变量赋值给其他变量**，或**②使用类型断言**。因此更推荐使用unknown。

let s: string;

// 报错！unknown类型不能直接赋值给其他变量。

s = e;

// 想用unknown类型赋值给其他变量，可以采取以下方式

if(typeof e === "string"){

s = e;

}

// 在unknown的值类型为string的时候赋值给s，否则不执行。

// 类型断言，可以用来告诉解析器变量的实际类型

/\*\*

\* 语法：（两种写法）

\* 变量 as 类型

\* <类型>变量

\*/

s = e as string;

s = <string>e;

### void 和 never

// void 用来表示空，以函数为例，就表示没有返回值的函数

function fn(): void{

}

// never 表示永远不会返回结果

//报错函数，只要一调用就报错了，程序一报错代码立即结束，就没有返回值了，这种函数返回值就可以设置为 never，用得比较少。

function fn2(): never{

throw new Error('报错了!');

}

### Object 对象

// object 表示一个js对象

let a: object;

a = {};

a = function(){

};

// {} 用来指定对象中可以包含哪些属性

// 语法：{属性名：属性值, 属性名：属性值...}

//令b等于一个对象，并且对象里要有一个name属性。

let b: {name:string};

b = {}; //报错，里面没有name属性。

b = {name: '孙悟空'}; // √

b = {name: '孙悟空', age: 18}; // 报错，多了一个age属性

// 在属性名后面加上？表示该属性是可选的，可有可无。

let c: {name:string, age?: number};

c = {name: '张三', age: 18}; // √

c = {name: '李四'} // √

如果想要实现一个固定的属性，其他不限数量的属性均为可选，该如何做呢？

// [propName: string]任意字符串类型的属性名; [propName: string]：any 表示任意类型的属性。

let d: {name: string, [属性名：string]：any}

d = {name: '王五', age: 18, gender: '男'};

如果想要限制 function函数 对象 的参数和返回值：

/\*\*

\* 设置函数结构的类型声明：

\* 语法：（形参：类型，形参：类型 ...） => 返回值

\*/

let d: (a: number, b: number)=>number;

d = function (n1, n2): number{

return 10;

}

### array 数组

/\*

\* 数组的类型声明：（两种方式）

\* 类型[]

\* Array<类型>

\*/

// string[] 表示字符串数组

let e: string[];

e = ['a', 'b', 'c'];

// number[] 表示数值数组

let f: number[];

// 等价于

let g: Array<number>

g = [1, 2, 3]

### tuple 元组

/\*

\* 元组，就是固定长度的数组

\* 数组长度可变，元组效率比起数组好一点，因此，在数组长度不变的情况下改用元组会好一点。

\* 语法：[类型，类型，...，类型]

\*/

let h：[string, string, number];

h = ['hello', "你好", 123];

### enum 枚举

// 定义一个枚举，里面有两个值male和female

//注：编译时编译器会将枚举值从上到下按0开始分配对应值。此时male对应0，female对应1.

enum Gender {

Male, // =0

Female // =1

// 我要是想指定默认值，直接在各个枚举项后面 =1， =2， 给补上就行，但是一般不需要。

}

let i: {name: string, gender: Gender};

i = {

name: '孙悟空'，

gender：Gender.Male

};

console.log(i.gender === Gender.Male);