**TypeScript 数据类型**

TypeScript 包含的数据类型如下表:

|  |  |  |
| --- | --- | --- |
| **数据类型** | **关键字** | **描述** |
| 任意类型 | any | 声明为 any 的变量可以赋予任意类型的值。 |
| 数字类型 | number | 双精度 64 位浮点值。它可以用来表示整数和分数。  let binaryLiteral: number = 0b1010; // 二进制  let octalLiteral: number = 0o744; // 八进制  let decLiteral: number = 6; // 十进制  let hexLiteral: number = 0xf00d; // 十六进制 |
| 字符串类型 | string | 一个字符系列，使用单引号（**'**）或双引号（**"**）来表示字符串类型。反引号（**`**）来定义多行文本和内嵌表达式。  let name: string = "Runoob";  let years: number = 5;  let words: string = `您好，今年是 ${ name } 发布 ${ years + 1} 周年`; |
| 布尔类型 | boolean | 表示逻辑值：true 和 false。  let flag: boolean = true; |
| 数组类型 | 无 | 声明变量为数组。  // 在元素类型后面加上[]  let arr: number[] = [1, 2];  // 或者使用数组泛型  let arr: Array<number> = [1, 2]; |
| 元组 | 无 | 元组类型用来表示已知元素数量和类型的数组，各元素的类型不必相同，对应位置的类型需要相同。  let x: [string, number];  x = ['Runoob', 1]; // 运行正常  x = [1, 'Runoob']; // 报错  console.log(x[0]); // 输出 Runoob |
| 枚举 | enum | 枚举类型用于定义数值集合。  enum Color {Red, Green, Blue};  let c: Color = Color.Blue;  console.log(c); // 输出 2 |
| void | void | 用于标识方法返回值的类型，表示该方法没有返回值。  function hello(): void {  alert("Hello Runoob");  } |
| null | null | 表示对象值缺失。 |
| undefined | undefined | 用于初始化变量为一个未定义的值 |
| never | never | never 是其它类型（包括 null 和 undefined）的子类型，代表从不会出现的值。 |

**注意：**TypeScript 和 JavaScript 没有整数类型。其中number，string，Boolean，null，undefined是基本数据类型在js中也有

### Ts中，定义变量时可以在变量名后面加冒号然后写类来约束变量的类，也可以不写因为ts有自动类型推断功能，一旦一个变量的类型确定了，就不能给他赋予另外类型的值

## 1，基本数据类型

### number，string，Boolean，null，undefined

|  |
| --- |
| let a : string = ‘Hello’ 或者 let a = ‘Hello’ |

## 需要注意

### 现在a是string类型，如果你怎么写就报错：a = 5 ，不能把number类型赋值给string类型变量a

### 在ts中一个数组的所有元素的类型必须一致，这个和Java和c++是一样的

### 在ts中元组是一种特殊的数组，他必须指明元素的个数以及每一个元素的类型

### 在typescript中声明一个字面量的对象是没有问题的，但是引用对象的属性的时候却报错，

|  |
| --- |
|  |

#### 这是为什么？

### 原来在ts中定义对象变量需要这样子写

|  |
| --- |
| let stu: {name:string,age:number} ={  name:'Jessie',  age:18 } |

### 在ts中定义一个类的写法

|  |
| --- |
| class Student{  name:string = ''  age:number = 0  gender:string =''  //注意在ts中构造函数一定是constructor而且没有重载，只有一种写法，要么有参数，要么没有参数  constructor(){} //无参构造 }    let stu = new Student() stu.name = 'Jack' stu.age = 30 stu.gender= 'male' console.log(stu) |

#### 有参数

|  |
| --- |
| class Student{  name:string = ''  age:number = 0  gender:string =''  //注意在ts中构造函数一定是constructor而且没有重载，只有一种写法，要么有参数，要么没有参数  constructor(n:string,a:number,g:string){  this.name = n  this.age = a  this.gender = g  } }  let stu = new Student('Jade',19,'female')   console.log(stu) |

## 在ts中还可以有一个可有可无的属性，这时候需要在这个属性名称后面加一个？如，下面的代码也是可以正常运行的

|  |
| --- |
| class Student{  name:string = ''  age:number = 0  gender:string =''  email?:string  // constructor() {  // }  //注意在ts中构造函数一定是constructor而且没有重载，只有一种写法，要么有参数，要么没有参数  constructor(n:string,a:number,g:string){  this.name = n  this.age = a  this.gender = g  } }  let stu = new Student('Jade',19,'female')  stu.email = 'Jade1234@gmail.com' console.log(stu) |

# Ts中的any和unknown类型

## any 是任意类型，具有所有类型的行为，可被执行，可访问属性，超脱于类型系统之外。 unknown 则是类型更安全的any，同样可以将任何类型赋给它，但不能执行任何操作，必须用类型断言来显示说明类型才能去执行操作。any类型的数据可以赋值给任意类型的数据，unknown类型的数据只能赋值给any或者unknown类型的数据

|  |
| --- |
| //any 和unknown类型 let obj:any = {  name:'Jack',  age:20,  gender:'male' }  console.log(obj.name) obj.email = 'Obj123@live.com' console.log(obj) |

### 由于any类型可以赋值给任意类型，所以下面的代码可以之前执行

|  |
| --- |
| let obj:any = {  name:'Jack',  age:20,  gender:'male' }  console.log(obj.name) obj.email = 'Obj123@live.com' console.log(obj)  let n :number = obj console.log(n) |
|  |

### 上面的例子中n的类型本来是number，但是当他接受了一个any类型的变量的值后，你用typeof n

### 检查，它显示object，输出的值也是一个对象，但是当你尝试去修改它的name属性，就会报一个错说number类型的变量没有name属性

### any类型的变量可以和在js里面一样，可以任意操作所以如果需要定义一个对象字面量用any类型比较方便，当然也不能乱用，否则就没有ts的类型严格特点了。能够知道类型的就尽量用这个类型，确实不好确定类型的采用any

## unkown类型的一些用法

|  |
| --- |
| let unstr:unknown = 'HELLO'  let sm = (unstr as string).toLowerCase() console.log(sm)  let unnum:unknown = 100 let res = (unnum as number) +1 |

### 这样子写也会执行

|  |
| --- |
| let unstr:unknown = 'HELLO' if(typeof unstr === "string"){  console.log(unstr.toLowerCase()) } |

### 注意：当你这么写代码是不会报错，但是代码不会执行

|  |
| --- |
| let unstr:unknown = 'HELLO' if(unstr instanceof String){ //这样子判断不可靠  console.log(unstr.toLowerCase()) //不会执行  } |

### unstr的数据类型现在是string但是它不是String的实例，因为当你给他赋值另外一种类型，他的数据类型又改变了，也就是说对应unknown类型的分为缩小判断应该使用typeof而不应该使用instanceof

### 注意：不要用unknown类型变量来接受一个对象字面量，因为你无法操作他是属性，如果必须使用可以先将这个unknown对象赋值给一个any类型的变量，然后再操作他的属性

## any类型还有一个好处就是可以用来做类型转换，如下面的代码是合法的

|  |
| --- |
| let a1 = '1' let b1:number = a1 as any as number   console.log(b1) |

### 注意：上面的代码中，如果你直接let b1:number = a1 as number是会报错的

## Ts中的元组其实就是一种数组，不过他可以在里面使用不同类型的元素。而且他是元素个数必须是确定的不能多，也不能少，这种数据的应用场景比较少

|  |
| --- |
| let x: [string, number];  x = ['Runoob', 1]; // 运行正常  x = [1, 'Runoob']; // 报错  console.log(x[0]); // 输出 Runoob |

## ts中的枚举类型

|  |
| --- |
| enum Gender {Female,Male}  let person = {  name:'xiaoming',  gender: Gender.Male }  console.log(person) //结果：{ name: 'xiaoming', gender: 1 } |

### 枚举的应用场景：在一些需要使用数组来表示的地方可以使用枚举，可以起到见名知义的作用，比直接使用数字方便，枚举的本质就是把一些number类型的变量放在一块，构成一个区间，你只能在在区间里面取一个值。所以使用枚举的地方可以用number来限定类型

# ts中的void

## void是用来描述函数没有返回值是情况，当一个函数没有返回值，我们把他的返回值类型设置为void

# ts中的never

## never表示一个永不存在的类型，一般我们很少使用，创建于报错