《鸿蒙北向应用开发基础》之

TypeScript基础介绍

软通教育教学教研部

CONTENTS

【课程目标】

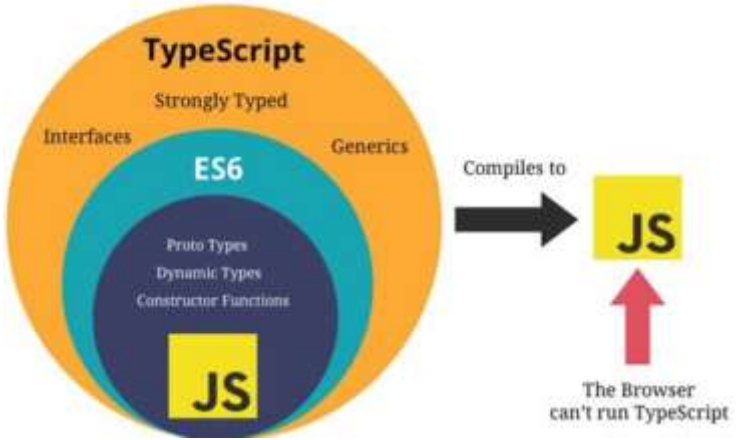
- 掌握TS几种数据类型的声明方式；
- 掌握使用let和const声明变量的区别；
- 掌握Number、String、Array、Map、Set等对象及其方法的应用；
- 了解什么是类，掌握类的定义方式；
- 掌握TypeScrip支持的3种访问修饰符。

01 TypeScript概述

TypeScript简介

TypeScript是由微软开发的自由、开源的编程语言。

1. 是JavaScript的超集
2. 向JS添加了静态类型和基于类的面向对象编程
3. typescript是一种面向对象的编程语言
4. 可以编译为纯JavaScript



TypeScript的优势

1. 对于大型项目，增强了代码可维护性
2. 在编译阶段检查类型，发现大部分错误
3. 支持JavaScript新特性
4. 生态繁荣

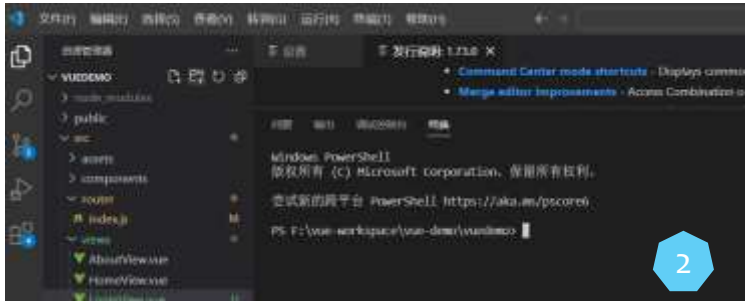
官网地址：https://www.typescriptlang.org/zh/

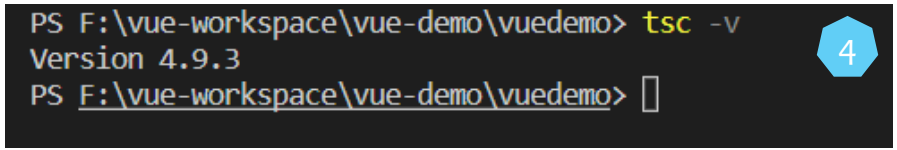
学习资源：https://www.typescriptlang.org/zh/docs/

演练场：https://www.typescriptlang.org/zh/play

环境安装

1. •VS Code下载链接：https://code.visualstudio.com/
2. •VS Code中开启终端，菜单：终端 > 新建终端
3. •在终端命令行中输入：npm install -g typescript
4. •安装完成后使用如下命令查看版本：tsc -v





TS版的HelloWorld

创建helloworld文件夹

使用VS Code打开文件夹

编写代码

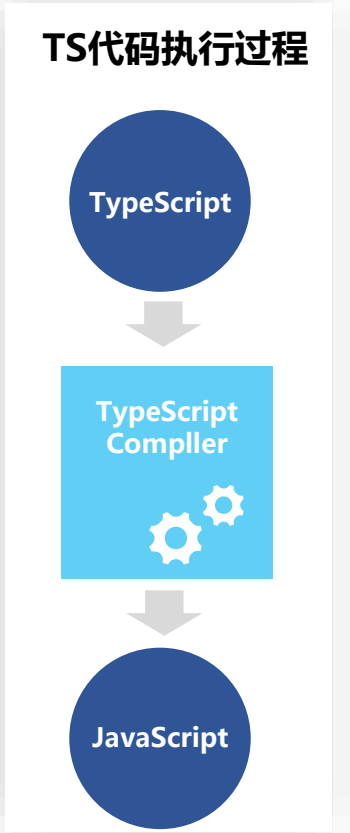
```
const s: string = "你好，世界！";
console.log(s);
```

编译ts代码

```
tsc helloworld.ts
```

运行ts代码

```
PS F:\ts-workspace\helloworld> node helloworld.js
你好，世界！
```



typescript是区分

分号是可选的：行末可以使用

单行注释 //

多行注释 /* */

02 TypeScript数据类型

JS原有类型：

boolean

布尔类型（true、false）

number

数值，支持十进制、十六进制、二进制、八进制

string

字符串，可以使用单引号或双引号表示，模板字符串使用反引号

数组

tuple（元组）

null、undefined

TS新增类型：

enum（枚举）

any（任意值）

void

声明：

```
let isOK: boolean = false;
let price: number = 9.19;
```

TypeScript和JavaScript都没有整数类型，整数和浮点数都是使用number表示的

声明字符串

- 可以使用单引号或者双引号表示

```
let myname:string = "jerry";
console.log(myname)
myname = 'petter';
console.log(myname)
```

模板字符串

- 定义多行文本和内嵌表达式，使用反引号包围，嵌入使用${ expr }

```
let word: string = 'good good study';
let age: number = 18;

let myword: string = `I am ${age} years old!
    so , I must
    ${word}!`;

console.log(myword);
```

模板字符串比使用+拼接字符串更方便。

1 使用[ ]定义数组

```
let names :string[] = ["张飞","刘备","关羽"];
```

2 使用数组泛型，Array<元素类型|元素类型>

单一类型：

```
let lucknumbers: Array<number> = [3,5,20,44,56];
```

混合类型：

```
let luckthings: Array<string | number> = ['a',1,'b',2,3];
```

```
let names :string[] = ["张飞","刘备","关羽"];
let lucknumbers: Array<number> = [3,5,20,44,56];
let luckthings: Array<string|number> = ['a',1,'b',2,3];
console.log(luckthings[4]);
console.log(names[2]);
console.log(lucknumbers[0]);
console.log(luckthings[4]);
```

元组类型用来表示已知元素数量和类型的数组

类似数组，长度固定，各元素类型可以不同，但对应位置的数据类型必须相同

```
let student: [string, number, string];
student = ["petter",18,"1班"];
console.log(student[0]);
console.log(student[1]);
console.log(student[2]);
```

枚举

枚举类型用于定义数值集合

```
// 枚举
enum Color {Red, Green, Blue};
let c: Color = Color.Green;
console.log(c);//输出为1
```

默认从0开始，可以手动指定开始值

```
// 枚举
enum Color {Red = 1, Green, Blue};
let c: Color = Color.Green;
console.log(c);//输出为2
```

或者全部手动赋值

```
// 枚举
enum Color {Red = 1, Green = 3, Blue = 4};
let c: Color = Color.Green;
console.log(c);//输出为3
```

可以通过枚举值找到映射的名字

```
let colorName: string = Color[1]
console.log(colorName);// 输出：Red
```

any&void&null&undefined

any

- 任意值是TypeScript针对编程时类型不明确的变量使用的一种数据类型
- 在编译阶段，当不希望类型检查器对这些值进行检查时，可以将这些变量标记为any类型

Void

- 表示函数没有值返回

Null

- 表示对象为空

Undefined

- 表示变量未设置值

```
// any
let x:any = 1;
x = "I am string";
x = false;
```

```
function noRetun(): void {

}
```

联合类型

通过管道符 (|)为变量指定多种类型，除指定类型外，赋值其他类型会报错

语法：type1 |type2 |type3

联合类型数组：

```
let myarr:number[]|string[];
myarr = [1,2,3,4];
console.log(myarr);
myarr = ["a","b","c","d"];
console.log(myarr);
```

联合类型做参数：

```
function unionFun(courseType:string|number){
    console.log("course type:",courseType);
}
unionFun("java");
unionFun(1);
```

```
let myval:string|number
myval = 12;
console.log("myval:",myval);
myval = "hello";
console.log("myval:",myval);
// myval = true; //报错
```

03 变量&运算符&条件语句&循环

变量声明

- 使用let

```
let age: number = 18;
let i = 10;
```

- 使用const

表示常量，一旦赋值，不可再次进行赋值

```
const week = 7;
```

运算符

- 算术运算符：+ - * / % ++ --
- 逻辑运算符: && || !
- 关系运算符: == != > < >= <=
- 位运算符：& | ~ ^ << >> >>>
- 赋值运算符：= += -= *= /=
- 三元运算符： Test ? expr1 : expr2
- 类型运算符：
  - typeof: 返回操作数的类型
  - instanceof：判断对象是否为指定类型

```
let num = 10;
    console.log(typeof num); // 输出number
```

条件语句

1. if
2. if ..else
3. if..else if..else
4. switch

1. for循环
2. while循环
3. do-while循环

- 对一组值的集合或者列表进行迭代输出
- 取出的是key或索引

```
//for in 循环
let students: string[] = ["jerry","petter","jhon"];
let val:any;
for (val in students) {
    console.log(val)
}
```

for ...of循环

- ES6引入 for..of 用来替代 for..in 和 forEach，可以遍历字符串、数组、集合等可迭代的数据结构

- 取出的是元素（内容）

```
// for of循环
let students: string[] = ["jerry","petter","jhon"];
let val:any;
for (val of students) {
    console.log(val)
}
```

- 对数组中的每个元素执行一次回调，不可停止或者中断循环，仅用于对数组的迭代

```
//forEach循环
let students: string[] = ["jerry","petter","jhon"];

students.forEach((val, idx, array) => {
    console.log(val); //当前值
    console.log(idx); //索引
    console.log(array);  //原数组
});
```

04 常用对象及方法

Number

- Number对象是原始数值的包装对象

  let num = new Number(value);
- 如果一个参数值不能转换为一个数字将返回NaN（非数字值）

```
console.log("最大值为: " +
Number.MAX_VALUE);
console.log("最小值为: " +
Number.MIN_VALUE);
console.log("负无穷大: " +
Number.NEGATIVE_INFINITY);
console.log("正无穷大:" +
Number.POSITIVE_INFINITY);
```

➢ Number对象常见属性：

属性	描述
MAX_VALUE	可表示的最大值，约为1.79E+308，大于此值代表infinity
MIN_VALUE	可表示的最小值，接近0，约为5e-324，小于此值会转为0，最大的负数是-MIN_VALUE
NaN	Not a Numer非数字值
NEGATIE_INFINITY	负无穷大，溢出时返回此值，该值小于MIN_VALUE
POSITIVE_INFINITY	正无穷大，溢出时返回此值，该值大于MAX_VALUE

Number对象方法

方法	描述
toFixed()	将数字转字符串并保留到指定小数位数
toLocalString()	将数字转换为用本地数字格式展示的字符串
toPrecision()	转换为指定长度
toString()	转为为字符串
valueOf()	返回Numer对象的原始数字值

```
let num1 = 123.123456;
console.log("toFixed():"+num1.toFixed());
console.log("toFixed(2):"+num1.toFixed(2));
console.log("toFixed(7):"+num1.toFixed(7));
console.log("toLocaleString():"+num1.toLocaleString());
console.log("toPrecision():"+num1.toPrecision());
console.log("toPrecision(1):"+num1.toPrecision(1));
console.log("toPrecision(2):"+num1.toPrecision(2));
console.log("toPrecision(3):"+num1.toPrecision(3));
console.log("toPrecision(4):"+num1.toPrecision(4));
console.log("toString():"+num1.toString());
console.log("valueOf():"+num1.valueOf());
```

- 字符串对象

var str = new String("HelloWorld");

var str = "HelloWorld";

- String对象常见属性

length：返回字符串的长度

- String常用方法

方法	描述
charAt()	返回指定位置的字符
concat()	连接字符串并返回新字符串
indexOf()	查找字符串并返回首次出现的位置
lastIndexOf()	从后向前查找字符串，位置计算仍从开头(0)处计算。
match()	正则匹配
replace()	正则替换
search()	正则搜索，返回找到位置，找不到返回-1
split()	将字符串分割成字符串数组
slice()	提取字符串片段并返回被提取部分
substring()	提取指定索引间的字符串
toLowerCase()	字符串转小写
toUpperCase()	字符串转大写

常用方法示例


```
let myword = "good good study,day day up";

console.log("length:"+myword.length);
console.log("charAt(3):"+myword.charAt(3));// 第一个字符索引是0
console.log("concat():"+myword.concat("!!!"));
console.log("indexOf():"+myword.indexOf("oo"));//返回 1
console.log("lastIndexOf():"+myword.lastIndexOf("oo"));//返回 6
console.log("match():"+myword.match(/oo/g));// 全局查找oo，返回：oo,oo
console.log("replace():"+myword.replace(/oo/g,"00"));// 返回g00d g00d study,day day up
console.log("search():"+myword.search(/oo/));// 返回1
console.log("split():"+myword.split(" "));// 返回：good,good,study,day,day,up
console.log("slice():"+myword.slice(1,5));// 返回：ood
console.log("slice():"+myword.slice(-1));// 返回：p
console.log("slice():"+myword.slice(-5,25));//返回：ay u
console.log("slice():"+myword.slice(-5,-4));//返回：a
console.log("substring():"+myword.substring(1,5));// 返回：ood
console.log("toLowerCase():"+myword.toLowerCase());
console.log("toUpperCase():"+myword.toUpperCase());
```

观察slice和substring的区别：slice索引可以使用负索引，表示从末尾计算

substring的负索引会直接取值为0，在从末尾计算时，最好使用slice方法。

构造方法

可以通过Array对象创建数组，Array对象的构造方法

1. 接收数组大小
2. 初始化数组列表，元素间使用逗号分隔

```
let prices:number[] = new Array(4);
let studNames:string[] = new Array("jerry","petter","jhon");
```

常见方法

方法	描述
concat()	连接两个数组（元素类型要相同）
forEach()	对数组每个元素都执行一次回调函数。
indexOf()	搜索数组，返回所在位置，如果找不到返回-1
lastIndexOf()	搜索元素最后出现的位置（从后向前找，第一次找到后返回，返回下标位置）
join()	使用指定字符串连接数组，默认使用“，”
pop()	删除数组的最后一个元素，并返回删除的元素
push()	向数组末尾添加一个或多个元素，并返回新的长度
shift()	删除并返回数组中的第一个元素
unshift()	向数组的开头添加一个或多个元素，并返回新的长度

slice()	选取数组的一部分，并返回一个新数组
reverse()	反转数组的元素顺序
sort()	对数组中的元素进行排序
map()	指定一个函数处理数组中的每个元素，并返回处理后的数组
reduce()	将数组计算为一个值，从左到右
reduceRight()	将数组计算为一个值，从右到左
every()	检测数组中的每一个元素是否都符合条件
some()	检测数组中是否有元素符合条件
filter()	从数组中过滤出符合条件的元素
toString()	将数组转换为字符串并返回
splice()	向数组添加项目或者从数组删除项目，并返回删除的项目，详情见后。

使用示例

```
let arr1 = ["a","b","c"];
let arr2 = ["d","e","f"];
console.log("concat():"+arr1.concat(arr2));

arr1.forEach(function(value){
    console.log(value);
});
console.log("indexOf():"+arr1.indexOf("b")); // 返回：1
console.log("join():"+arr1.join());// 返回：a,b,c
console.log("join():"+arr1.join("|"));// 返回：a|b|c
console.log("pop():"+arr1.pop());
console.log("arr1:"+arr1);
console.log("push():"+arr1.push("1"));
console.log("arr1:"+arr1);
console.log("shift():"+arr1.shift());
console.log("arr1:"+arr1);
console.log("unshift:"+arr1.unshift("aa"));
console.log("arr1:"+arr1);
console.log("slice():"+arr1.slice(1,2));
arr1.reverse();
console.log("arr1 reverse():"+arr1);
console.log("sort():"+arr1.sort());
console.log("arr1:"+arr1);
```

```
let arr3 = [1,4,9,16,25];
console.log("map():"+arr3.map(Math.sqrt));
console.log("reduce():"+arr3.reduce((a,b) =>{return a+b;}));
console.log("reduceRight():"+arr3.reduceRight((a,b) =>{return a-b;}));
console.log("every():"+arr3.every((value)=>{return value >0}));
console.log("every():"+arr3.every((value)=>{return value >10}));
console.log("some():"+arr3.some((value)=>{return value >10}));
console.log("filter():"+arr3.filter((value)=>{return value >10}));
console.log("toString():"+arr3.toString());
```

splice()函数详解

语法：

array.splice(index,howmany,item1,...,itemX)

参数	描述
index	必须，整数，指定添加或者删除的位置，如果为负数则表示从数组末尾开始计算位置
howmany	可选，要删除的项目数，如果为0则不删除任何项目
item1,...,itemX	可选，要添加到数组中的新项目

```
let arr4 = ["a","b","c","d","e"];
console.log("removed:"+arr4.splice(2,0,"1"));
console.log("arr4:"+arr4);
console.log("removed:"+arr4.splice(2,2,"1","2"));
console.log("arr4:"+arr4);
console.log("removed:"+arr4.splice(2,2));
console.log("arr4:"+arr4);
```

元组

数组中的元素类型通常相同（any[]类型的数组可以不同），如果要存储不同类型，则需要使用元组。访问方式和数组类似，通过索引来访问，但取出时需要注意元素的类型。

声明元组并使用下标访问：

```
let tuple1 = [1,"a",2,"b"];
console.log(tuple1[1]);
```

常用方法：

push()：向元组最后面添加元素，返回长度

pop()：从元组的最后面移除元素，并返回移除的元素

数组中的其他方法也可以在元组中找到对应版本

```
let tuple1 = [1,"a",2,"b"];
console.log(tuple1[1]);
console.log("push():"+tuple1.push(3));
console.log("tuple1:"+tuple1);
console.log("pop():"+tuple1.pop());
console.log("tuple1:"+tuple1);
console.log("reverse():"+tuple1.reverse());
```

- 可以把元组元素赋给变量

```
let [a,b,c] = tuple1;
console.log("a:"+a);
console.log("b:"+b);
console.log("c:"+c);
```

Map对象记录键值对，并且保留键的原始插入顺序。Map是ES6引入的新的数据结构

声明及初始化：

```
let studMap = new Map();

let studMap1 = new Map([[1001,"jerry"],
    [1002,"petter"],
    [1003,"jhon"],
    [1004,"mary"]
]);//以数组的形式传入键值对
```

常用属性：

size：返回Map对象键值对数量

常用方法

方法	描述
set()	设置键值对，返回Map对象
get()	返回键对应的值，如果不存在则返回undefined
has()	判断Map中是否包含键对应的值
keys()	返回包含Map key的Iterator对象
values()	返回包含Map value的Iterator对象
entries()	返回一个Iterator对象，按插入顺序包含Map对象中每个元素的[key,value]数组
delete()	删除Map中的元素，成功返回true，失败返回false
clear()	移除所有键值对

编译:
tsc --target es6 example2.ts
Map为es6新增，默认可能不支持，需要在编译时使用--target参数指定目标库

```
let studMap = new Map();

studMap.set(1,"zhangshan");
studMap.set(2,"lisi");
studMap.set(3,"masan");
studMap.set(4,"zhaoliu");

console.log("size:"+studMap.size);
console.log("get():"+studMap.get(2));
console.log("has():"+studMap.has(4));
console.log("has():"+studMap.has(5));
console.log("delete()"+studMap.delete(4));
console.log("size:"+studMap.size)
studMap.clear();
console.log("size:"+studMap.size)
```

Map迭代

Map对象保留着对象插入顺序，每次迭代返回[key,value]数组

通过for..of迭代

```
let studMap1 = new Map([[1001,"jerry"],
    [1002,"petter"],
    [1003,"jhon"],
    [1004,"mary"]
]);

// 使用for ... of 迭代key
for(let key of studMap1.keys()){
    console.log(key);
}
console.log("------");
// 使用for ... of 迭代value
for(let value of studMap1.values()){
    console.log(value);
}
```

```
console.log("------");
// 利用entries方法
for(let entry of studMap1.entries()){
    console.log(entry[0],entry[1]);
}
console.log("------");
//利用对象解析
for(let [key,value] of studMap1){
    console.log(key,value);
}
console.log("------");
for(let [key,value] of studMap1.entries()){
    console.log(key,value);
}
```

```
// 使用forEach
studMap1.forEach((value,key) =>{console.log(key,value)})//第一个参数是value，第二个参数是key
```

Set

Set对象可以存储任何类型的数据，但值必须唯一

- **声明和初始化:**

```
let studSet = new Set();
let studSet1 = new Set(["jerry","petter","jhon","marry"]);//将数组转set
```

- **常用属性:**

  size：返回元素个数

- **常用方法:**

方法	描述
add()	在尾部添加元素，并返回当前set对象
delete()	删除元素，并返回删除是否成功
has()	返回set中是否存在某元素
clear()	清除所有元素，返回值为void
values()	返回包含元素的Iterator对象
forEach()	以插入顺序迭代，每次迭代执行一次回调

```
let studSet = new Set();
let studSet1 = new Set(["jerry","petter","jhon","marry"]);// 将数组转Set

console.log("site:"+studSet1.size);
console.log("add():",studSet1.add("zhangsan"));
console.log("delete():",studSet1.delete("zhangsan"));
console.log("studSet1:",studSet1);
console.log("has():",studSet1.has("zhangsan"));


studSet1.clear();
console.log("studSet1:",studSet1);
```

```
let studSet1 = new Set(["jerry","petter","jhon","marry"]);// 将数组转Set

for(let value of studSet1.values()){
    console.log(value);
}
console.log("------");
for(let value of studSet1){
    console.log(value);
}

console.log("------");

studSet1.forEach((value)=>{console.log(value);});
```

Set应用

1 数组去重

```
let setArr = new Set([1,2,2,3,3,3,4,4,4,4,5,5,5,5,5]);
console.log(setArr);
```

2 并集&交集&差集

```
let a = new Set([1,2,3]);
let b = new Set([2,3,4]);
// 并集
let union = new Set([...a,...b]);// ...展开运算符，把数组或对象内容展开，es6新增
console.log("union:",union);

// 交集
let intersect = new Set([...a].filter(x => b.has(x)));
console.log("insertsect:",intersect);

// 差集
let diff = new Set([...a].filter(x=> !b.has(x)));
console.log("diff:",diff);
```

05 类&接口&对象

类

- TypeScript是面向对象的，支持类，接口等
- 类描述对象的共同属性和方法

类的定义：

```
class class_name{
        //字段
        //构造方法
        //方法
}
```

```
class Cat{
    // 字段
    nickname:string;
    age:number;
    // 构造方法
    constructor(nickname:string,age:number){
        this.nickname = nickname;
        this.age = age;
    }
    // 方法
    sayHello():void{
        console.log("Hello,I am ",this.nickname);
    }
}
let katty = new Cat("Katty",3);
katty.sayHello();
```

访问修饰符

访问修饰符可以控制对类、变量、方法和构造方法的访问。

TypeScrip支持3种访问修饰符：

1. public：公共，可以在任何地方访问，默认。
2. protected：保护，自身及子类访问
3. private：私有，本类内部访问

```
class Cat{
    // 字段
    nickname:string;
    age:number;
    private friend:string;
    brother:string;// 默认为public
    protected father:string
    // 构造方法
    constructor(nickname:string,age:number){
        this.nickname = nickname;
        this.age = age;
    }
    // 方法
    sayHello():void{
        console.log("Hello,I am ",this.nickname);
    }
}
 let katty = new Cat("Katty",3);
 katty.brother = "big Bosi Cat";
 katty.friend = "HelloKatte";//无法访问，报错
```

类的继承

TypeScript支持类的继承，使用extends关键字，子类不能继承父类的私有成员（方法和属性）和构造方法。

语法： class sub_class extends super_class

```
class BosiCat extends Cat{

    showAge():void{
        console.log("Bosi cat age:",this.age);
    }

}

let bosiCat = new BosiCat("bosi",2);
bosiCat.showAge();
```

方法重写

- 子类重新定义父类中的方法
- super关键字用于引用父类的属性和方法

```
class BosiCat extends Cat{
    showAge():void{
        console.log("Bosi cat
age:",this.age);
    }
    sayHello(): void {
        super.sayHello();
        console.log("this is child bosi cat!");
    }

}
let bosiCat = new BosiCat("bosi",2);
bosiCat.sayHello();
```

通过static关键字定义的成员可以直接通过类名访问。

```
class BosiCat extends Cat{
    static nation:string;
    showAge():void{
        console.log("Bosi cat age:",this.age);
    }

    sayHello(): void {
        super.sayHello();
        console.log("this is child bosi cat!");
    }

}
BosiCat.nation = "波斯";
```

接口

接口定义属性和方法声明，方法都是抽象，需要具体的类进行实现

TypeScript接口定义：

interface interface_name{ }

接口继承和实现：

- 接口可以使用extends关键字继承一个或者多个接口，接口允许多重继承。
- 类可以通过implements关键字实现接口。

```
interface IAnimal{
    animalType:string;
    sex ?:string;// 带？表示可选属性
    walk(): void;// 也可以写成：  walk:() => void;
}
let fish: IAnimal = {
    animalType: "fish",

    walk(){
        console.log("I am swimming!");
    }
};
console.log(fish.animalType);
console.log("---------");
fish.walk();
console.log("---------");
```

对象

对象是保护一组键值对的实例，可以保护标量、函数、数组、对象等

```
let obj_name = {
    key1: "value1",
    key2: "value2",
    key3: ["v1","v2","v3"],
    key4: function(){}
}
```

```
let students = {
    stud1: "张飞",
    stud2: "刘备",
    stud3: "关羽"
}

console.log(students.stud2);
```

【课程小结】

- TypeScript支持的数据类型有：boolean、number、string、array、tuple、null、undefined、enum（枚举）、any（任意值）、void
- TypeScript是面向对象的，支持类，接口等；类描述对象的共同属性和方法
- TypeScrip支持3种访问修饰符：
  public：公共，可以在任何地方访问，默认。
  protected：保护，自身及子类访问
  private：私有，本类内部访问

软通教育
ISOFTSTONE EDUCATION

SOFTSTONE
软通动力
数字转型新动力

匠心育人，学以致用

软通教育教学教研部