# 介绍

TS是js的超集，是js语法的扩展；TS通过类型注解提供编译时的静态类型检查。

安装：npm install -g typescript

文件扩展名：.ts

TS转换成JS文件：tsc test.ts

# 基本类型

1. 任意类型：any; 可以赋值任意类型 any[]
2. number: 各种进制、整数等
3. 数组类型：number[] 或Array<number>
4. 元组：表示已知元素**数量**和**类型**的**数组;** 例：

let x : [string, number, boolean] = [‘string’, 0xf00d, true]

1. 枚举: enum 枚举类型用于定义数值集合; 例：

enum Family { father, mather, sister };

let a : Family = Family.mather; // 2

该集合的值默认从0开始排序，若第一个值使用方法算出的，则必须给第二个元素赋值。

1. 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 {

console.log(‘hello world’);

}

1. null 表示对象值缺失
2. undefined 用于初始化变量为一个未定义的值
3. never 其他类型的子类型，表达从不会出现的值。

注：若要支持多类型采用 ‘|’, let a : number | string | undefined

类型推断: let a = 9; // 会自动推断a为number类型

类型断言：将一个变量的类型转换成另一种类型

var a = true;

var b:number = <number> <any> a;

console.log(typeof b) // boolean

作用域：全局、类作用域、局部作用域

运算法：略

循环：for (var val in list)

此时的val需要先进行声明；若list里面啥都有，需要声明为any

循环使用for of最好

# 函数：

1. 关于函数返回值

let myFunction = function person() : string {}

返回的必须是string类型的

1. 带参数

function add(x: number, y: number): number{ … }

1. 可选参数：

上例如果传的不是两个参数会报错

function add(x: number, y?: number): number{ … }

用？标识的参数为可选参数，可不传值。

1. 默认参数

上例如果不传值，但函数又要使用时，可以将改参数变成默认参数；但可选参数和默认参数不能同时设置，因为设置了默认参数，也有可选参数的特质。

function add(x: number, y: number = 1): number{ return x + y };

add(100);

1. 剩余参数 类似于arguments

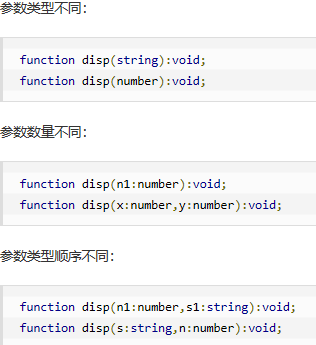
将不确定数量的参数作为数组传入

function buildName(firstName: string, **...restOfName: string[]**){ … }

-

1. 函数重载

重载是方法名字相同，而参数不同，返回类型可以相同也可以不同；



# 联合类型 ‘|’

通过管道符号让一个变量支持多种类型；例：

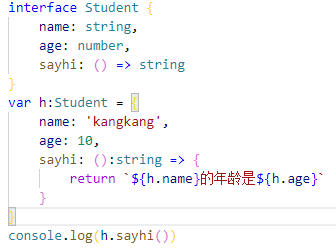
var val : string | Boolean = ‘string’;

也可以用在数组

# 接口：

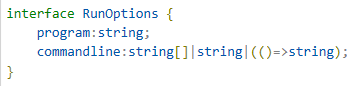
是一系列抽象方法的声明，是一些方法特征的集合，这些方法都应该是抽象的，需要由具体的类去实现，然后第三方就可以通过这组抽象方法调用，让具体的类执行具体的方法。

和对象：



注：return中用this.name会报错：类型“typeof globalThis”上不存在属性“name”。

联合类型和接口：



使用一次之后，可以继续赋值其他类型的数据

接口与数组：(这有什么用？？？)

interface nameList{ [index: string]:string }

该接口规定了该数组的索引值是字符串类型，该数组元素也是字符串类型；

这样设置当：

var a:nameList = [‘a’, ‘b’, ‘c’, ‘d’];

a[2]无法获取第三个元素值。

接口继承：单继承和多继承 和class继承一样，使用extend

interface Musician extends Person{ … }

interface Child extends IParent1, IParent2{ … }

支持同时继承多个接口，这个和class不一样

# 类



**继承：extend**

class child\_class\_name extends parent\_class\_name

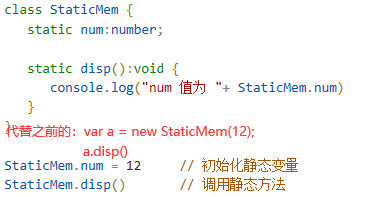
类的继承：不支持继承多个类，但支持继承多重类，例：

class Father extend Grandfather{ … }

class child extend Father{ … }

**关键词：static**

用于定义类的数据成员为静态。可以直接通过类名调用。



作用：被static修饰的变量属于**类变量/类方法**，可以通过**类名.变量名**直接引用，而不需要new出一个类。

不同的类有自己的静态资源，这可以实现静态资源分类。避免重名。

**instanceof运算符：**

判断对象是否是指定的类型，如果是返回 true，否则返回 false。

class Person{ }

var obj = new Person()

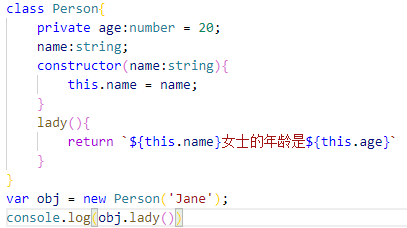
var isPerson = obj instanceof Person;

判断obj是否是Person类实例化出来的

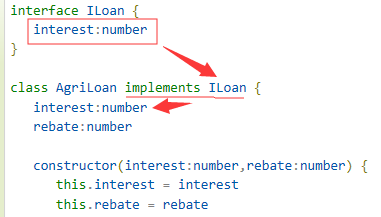
访问控制：

1. **public（默认）** : 公有，可以在任何地方被访问。
2. **protected** : 受保护，可以被其自身以及其子类和父类访问。
3. **private** : 私有，只能被其定义所在的类访问。

私有的变量及使是该对象的实例化也无法调用，可以在类中的方法调用

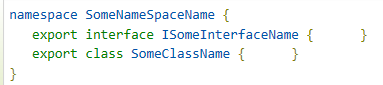


**类与接口：implements**



# 命名空间

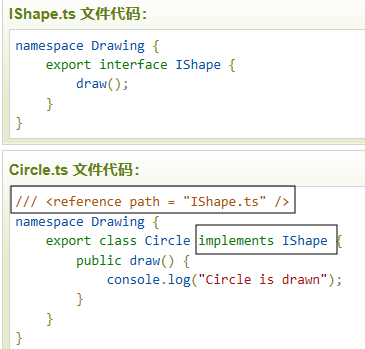
作用：解决重名问题。



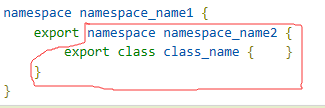
把接口和类等用命名空间包裹起来；通过export进行引出；

调用 ：SomeNameSpaceName.SomeClassName;

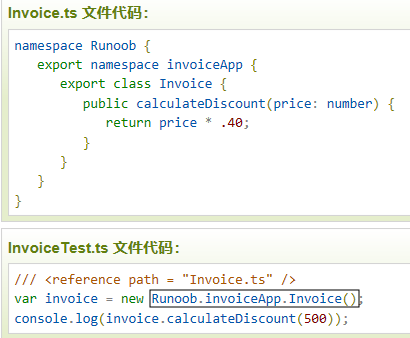
当命令空间文件单独存放在一个文件时，调用方法：



命名空间支持嵌套：



例：



# 模块

模块是在其自身的作用域里执行，并不是在全局作用域，这意味着定义在模块里面的变量、函数和类等在模块外部是不可见的，除非明确地使用 export 导出它们。类似地，我们必须通过 import 导入其他模块导出的变量、函数、类等。

