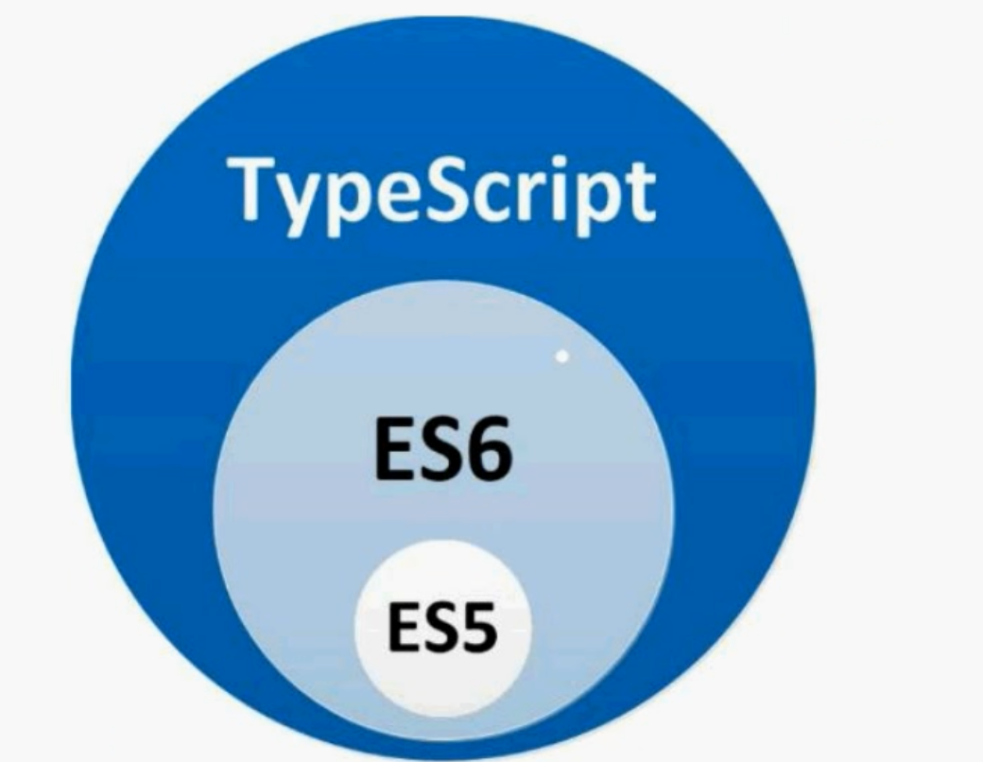
# 一 TypeScript简介与使用

## 1.1 TypeScript简介

TypeScript是由微软开发的一款开源编程语言，是JavaScript的超集，遵循ES6与ES5规范，扩展了原生JS语法，适合开发大型企业项目。



## 1.2 配置环境

本地环境：npm install -g typescript

在线环境：<http://www.typescriptlang.org/play/index.html>

## 1.3 HelloWorld

新建index.ts文件，编写如下代码：

export class Index {

run(){

console.log("hello world");

}

}

(new Index()).run();

该代码既不能在浏览器、Node中运行，需要先编译为原生JS，在命令行输入：

tsc index.ts

此时会生成另外一个文件 index.js，该文件可直接运行。

## 1.4 自动编译

每次书写完的ts文件需要编译是很痛苦的，通过配置可以实时编译，（webstorme支持自动编译），VSCode配置步骤如下：

第一步：tsc --init 生成配置文件，可以生成一个tsconfig.json文件；

第二步：点击 菜单-任务-运行任务-tsc:监视-tsconfig.json。

# 二 数据类型

## 2.1 九种类型

布尔类型： boolean

数字类型： number

字符串类型：string

数组类型： array

元组类型： tuple

枚举类型： enum

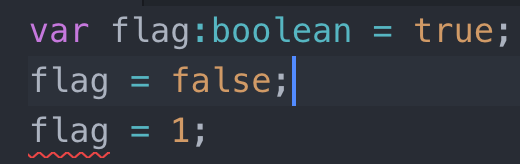
任意类型： any

void类型： void

never类型： 包括null和undefined

## 2.2 类型校验

TypeScript引入了类型校验，生命一个变量必须同时声明变量的类型：



如图所示，1是数值类型，已经报错。

## 2.3 数组 array

在typescript中可以直接使用es5的定义语法，也可以指定数组中元素类型：

var arr1 = [1,2,'str']; *//ES5原生数组*

var arr2:number[] = [1,2,3]; *//数组元素只能是数字*

var arr3:Array<number> = [1,2,3]; *//数组元素只能是数字*

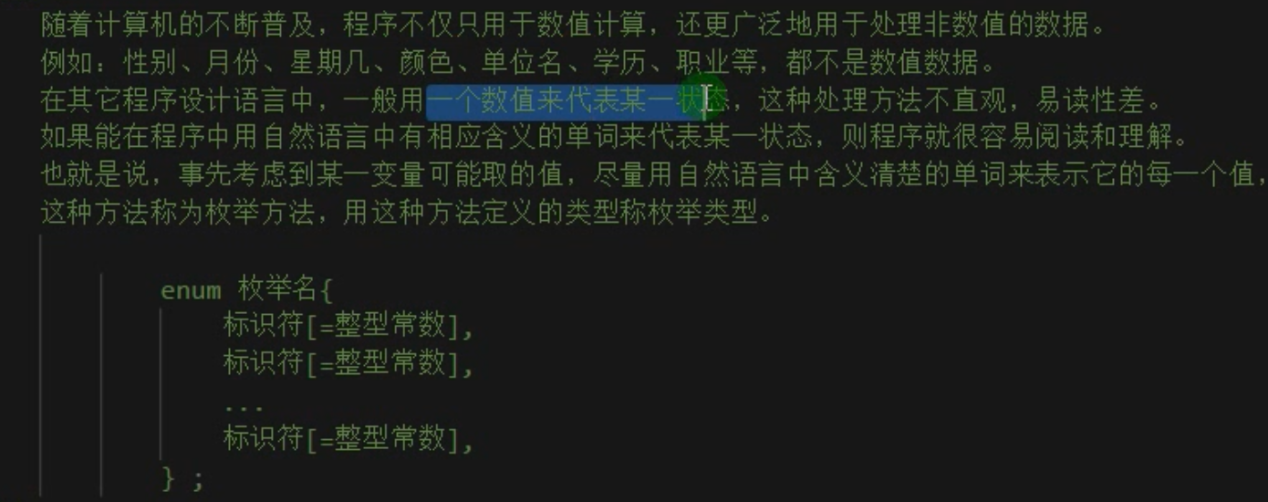
## 2.4 元组类型 tuple

元组和数组类似，严格来说和JS中的数组很像，数组内的元素可以是不同的类型:

let arr1:[number,string] = [1,'aaa'];

let arr2:[number,string] = [1,2,'aaa']; *//报错*

## 2.5 枚举 enum



enum Flag {

success = "成功",

error = "失败"

}

let f:Flag = Flag.success;

console.log(f); *//打印成功*

enum Color {blue, red, green=3,'红色'}

let a:Color = Color.blue;

let b:Color = Color.red;

let c:Color = Color.green;

let d:Color = Color.红色;

console.log(a); *//输出0 如果标识符没有赋值，它的值就是下标*

console.log(b); *//输出1*

console.log(c); *//输出3*

console.log(d); *//输出4*

## 2.6 任意类型 any

let num:any = 123;

console.log(typeof num);

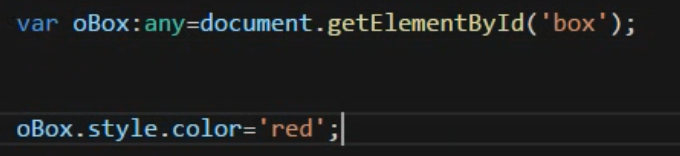
num = false;

console.log(typeof num);

num = 'str';

console.log(typeof num);

任意类型的使用场景，通常在前端开发中有如下场景：



如果没有any，在typescript中会报错提示oBox是个object类型，但是typescript没有object类型！

## 2.7 其他类型 never

never类型除了never修饰外，undefined和null也是never类型的子类型。

### 2.7.1 未定义 undefined

变量声明了但是没有赋值，即undefined，如下所示：

var num:number;

console.log(num): //在typescript中报错

如果这样写：

var num:undefined;

console.log(num): //不会报错，输出undefined

如果我们希望num在是数字，或者是undefined的时候都不报错，写法如下：

let num:number | undefined;

console.log(num); *//undefined*

num = 3;

console.log(num); *//3*

### 2.7.2 空类型 null

当定义一个变量类型为null时，变量的值只能是null

var num:null;

num = null; //正确

num = 123; //错报错

### 2.7.3 never

var a:never;

a = 123; //报错

never类型是任何类型的子类型，也可以赋值给任何类型；然而，没有类型是never的子类型或可以赋值给never类型（除了never本身之外）。 即使 any也不可以赋值给never。通常表现为抛出异常或无法执行到终止点（例如无线循环）。比如：

let x: never;

*// 运行正确，never 类型可以赋值给 never类型*

x = (()=>{ throw new Error('exception')})();

*// 返回值为 never 的函数可以是抛出异常的情况*

function error(message: string): never {

throw new Error(message);

}

*// 返回值为 never 的函数可以是无限循环这种无法被执行到的终止点的情况*

function loop(): never {

while (true) {}

}

## 2.8 函数返回值类型 void

见函数章节

## 2.9字符串特性

*//字符串模板作为参数传递--不需要方法的小括号*function test(templateStr,name,age){  
 console.log(templateStr);  
 console.log(name);  
 console.log(age);  
}  
let myname = 'lisi';  
let getAge = function () {  
 return 18;  
};  
test`name is ${myname},age is ${getAge()}`;

# 三 函数

## 3.1 函数定义

function run(name:string, age:number) :boolean {

return false;

}

当然，ts也支持匿名函数

## 3.2 函数返回值与void

void表示函数没有返回值

function run1():void {

console.log("没有return");

}

function run2():number {

console.log("必须返回数值类型");

return 1;

}

## 3.3 参数默认值

function test(a:string,b:string = 'hi'){  
 console.log(a);  
 console.log(b);  
}  
test('xx','yy');  
test('xx');

注意：默认值必须在参数列表最后面。

## 3.4 可选参数

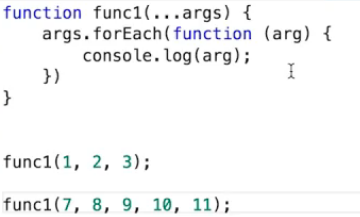
ES5中，方法的实参和形参可以不一样，但是ts中必须一样，如果要求二者不一样，ts中可以配置可选参数。

function test(a:string,b?:string,c:string = 'hi'){  
 console.log(a);  
 console.log(b);  
 console.log(c);  
}  
test('xx'); *//一个参数也可以执行，因为b可选  
//此时输出b的值为undefined*

声明了可选参数，需要在函数体内部说明没有传入可选参数，可选参数应该如何处理；

可选参数只能写在必选参数后面，不能写在默认参数后面。

## 3.5 剩余参数



当然参数如果出现剩余参数，函数声明中也可以额外书写多个参数，调用时按顺序解析。

## 3.6 函数重载

重载：同一个函数名，因为参数的不同而执行不同结果，但是typescript和java不一样，如下所示：

function getInfo(name :string) :string;

function getInfo(age :number) :string;

function getInfo(str :any) :any {

if ( typeof str === 'string') {

return "我叫：" + str;

} else {

return "我的年龄是：" + str;

}

}

console.log(getInfo("张三"));

console.log(getInfo(12));

# 四 类 class

## 4.1 类的定义

class Person {

name :string; *//属性*

age :number; *//属性*

constructor(a:string, b:number){ *//构造函数*

this.name = a;

this.age = b;

}

info() :void {

console.log(this.name + "的年龄是：" + this.age);

}

}

let p = new Person('张三',40);

p.info();

TS同时也支持ES6中的extends继承，不过要注意继承的新类的构造函数需要super()。

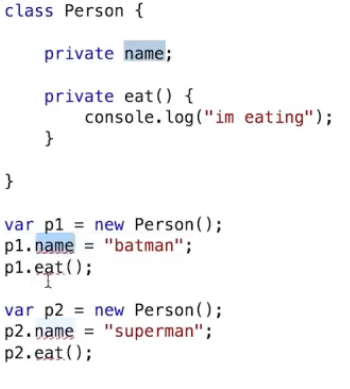
TS也同时兼容了ES5中的static关键字。

## 4.2 类属性修饰符

public： 默认的修饰符，可以不写，在类里面、子类、类外面都可以访问

private： 在类里面可以访问，子类、类外部都没法访问

protected： 在类里面、子类里面可以访问，类外部无法访问



如上所示，私有成员访问会报错。

注意：如果没有修饰符，则默认都是public，但是在构造函数中，有所区别。

如果构造函数的参数没有修饰符，那么在构造函数外部是无法使用这个参数的；

如果构造函数的参数使用了public修饰符，那么在构造函数外部可以使用该参数

## 4.3 多态

多态：父类定义一个方法不去实现，让他的子类去实现该方法，不同的子类实现的方式不同。

## 4.4 抽象类

抽象类和抽象方法用来定义标准。抽象方法不包含具体的实现。

注意：抽象类不能实例化，只有其子类才能实例化。

abstract class Animal {

abstract eat() :any; *//抽象方法只能出现在抽象类中*

}

# 五 接口interface

## 5.1 接口的定义

抽象类和抽象方法虽然提供了标准，但是只针对类中的某个函数设立了规范。接口对类本身设立了规范。

## 5.2 属性接口

属性接口更多是对json对象提供了约束。

interface Person {  
 name:string;  
 age:number;  
}  
  
class Student {  
 *//构造函数支持接口* constructor(public config:Person){  
 }  
}  
  
*//在创建对象时，必须满足接口定义的属性*let s1 = new Student({  
 name:'lisi',  
 age:18  
});

## 5.3 类实现接口

interface Person {  
 name:string;  
 age:number;  
 run();  
}  
  
class Student implements Person{  
 name:string;  
 age:number;  
 constructor(){  
 }  
 *//实现接口方法* run(){  
 console.log(this.name + ' run....');  
 }  
}  
  
let s1 = new Student();  
s1.name = 'lisi';  
s1.age = 15;  
s1.run();

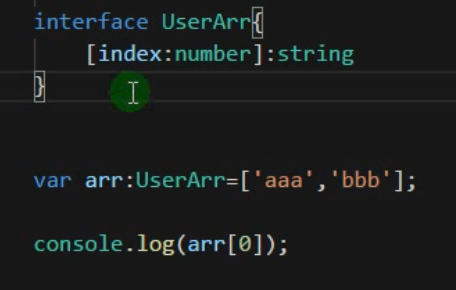
## 5.4 函数接口

针对函数的参数进行规范：



## 5.5 可索引接口

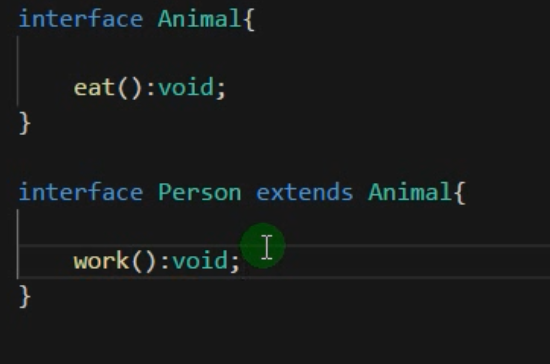
可索引接口 对数组的索引进行限制，如果数组中的索引传入了其他类型，那么报错



可索引接口 对对象也可以进行限制：json中多写字段，则报错

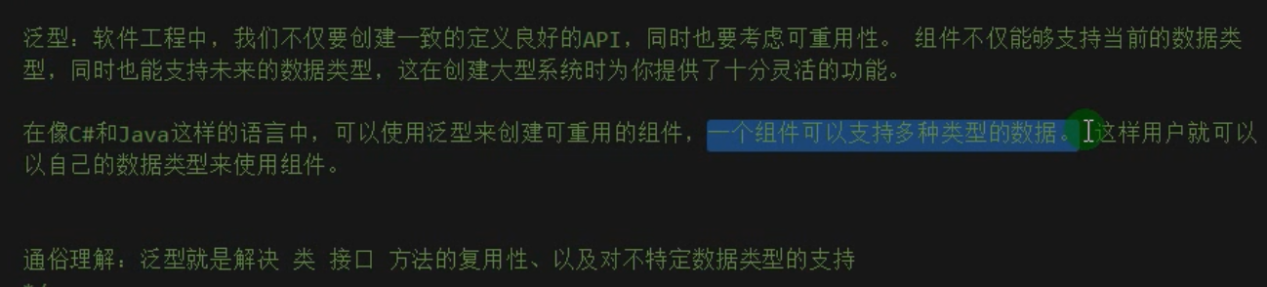


## 5.6 接口可以被继承



# 六 泛型

## 6.1 泛型的定义



有这样两个函数，都是普通的返回一个值，依据参数不同，返回不同类型：

## 6.2 泛型的应用场景

function f1(value :string) :string {

return value;

}

function f2(value :number) :number {

return value;

}

*//上述的明显函数的创建显得累赘，通过泛型解决*

function fn<T>(value :T) :T {

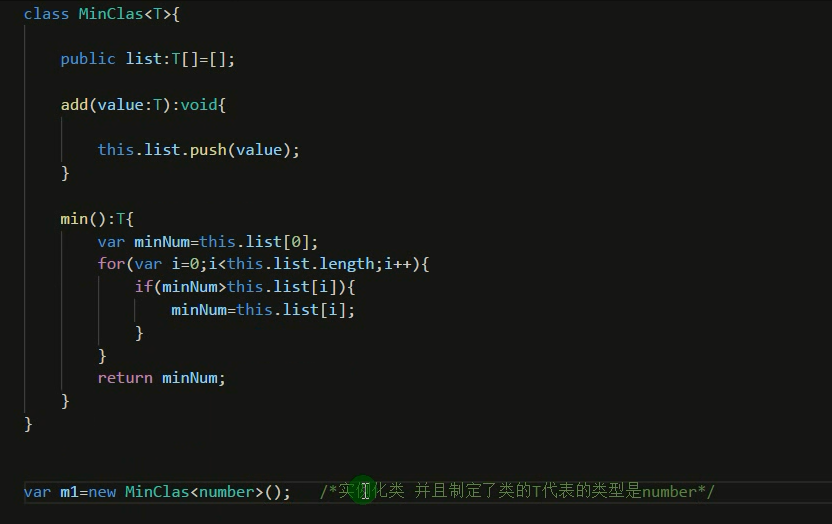
return value;

}

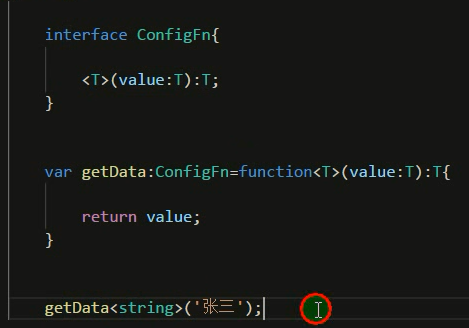
console.log(fn<number>(123));

console.log(fn<string>("test"));

## 6.3 泛型应用于类



## 6.4 泛型应用于函数接口

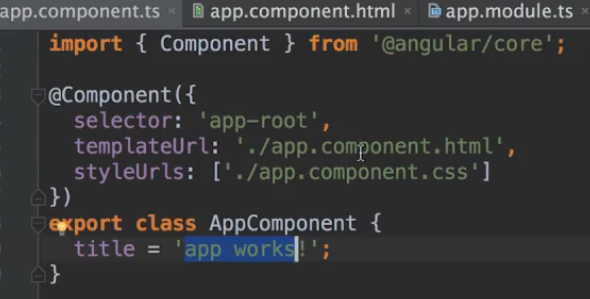


## 6.5 泛型应用于类接口



# 七 注解 annotation

注解为程序的元素（类、方法、变量）加上更直观更明了的说明，这些说明信息与程序的业务逻辑无关，而是供指定的工具或框架使用。



# 八 类型定义文件

类型定义文件后缀为 .d.ts 。

类型定义文件用来帮助开发者在TS中使用已有的JS工具包，如：jQuery。

这个文件其实是一个typescript文件模块，提供给开发者直接使用，比如引入jQuery的类型定义文件后，就可以使用$与jQuery的方法：

