Typescript

——JS plus

Time：2024-07-09

目 录

[一、基础 1](#_Toc171443513)

[1.类型 1](#_Toc171443514)

[1.1 基础类型 1](#_Toc171443515)

[1.2 对象类型：接口 3](#_Toc171443516)

[1.3 数组类型 3](#_Toc171443517)

[1.4 函数类型 4](#_Toc171443518)

[1.5 类型断言 6](#_Toc171443519)

[2.声明 8](#_Toc171443520)

[2.1 声明语句 8](#_Toc171443521)

[2.2 声明文件 8](#_Toc171443522)

[2.3 全局变量 8](#_Toc171443523)

[二、编程 9](#_Toc171443524)

[1.基础 9](#_Toc171443525)

[1.1 保留关键字 9](#_Toc171443526)

[1.2 基础类型 9](#_Toc171443527)

[1.3 变量声明 10](#_Toc171443528)

[1.4 运算符 11](#_Toc171443529)

[1.5 条件语句 15](#_Toc171443530)

[1.6 循环语句 16](#_Toc171443531)

[1.7 函数 19](#_Toc171443532)

[1.8 内置对象 22](#_Toc171443533)

[三、进阶 25](#_Toc171443534)

[1.基础 25](#_Toc171443535)

[1.1 类型别名 25](#_Toc171443536)

[1.2 字符串字面量类型 25](#_Toc171443537)

[1.3 元组 25](#_Toc171443538)

[1.4 枚举 26](#_Toc171443539)

[1.5 类 26](#_Toc171443540)

[1.6 类与接口 28](#_Toc171443541)

[1.7 泛型 29](#_Toc171443542)

[2.工程 32](#_Toc171443543)

[四、伴生问题 33](#_Toc171443544)

[1.NPM 33](#_Toc171443545)

[X.参考 34](#_Toc171443546)

# 一、基础

工具体验typescript在线

Typescript特点：

弱类型、静态类型（编译时类型检查防止运行时再出现错误，对冲JS动态类型错误）、面向对象

JavaScript 和 TypeScript 中不管加号两侧是什么类型，都可以通过隐式类型转换计算出一个结果——而不是报错

总结：



## 1.类型

### 1.1 基础类型

#### （1）原始数据类型

Number：数字

String：字符串

Boolean：bool类

Array：数组let myArray : number[] = [1, 2, 3, 4];

Void：可以同void表示没有任何返回值得函数，function xxx():void {}

Undefined：特殊值，未赋值，声明未初始化

Null：特殊值，空值，手动给的

(Undefined和Null是所有类型得子类型，可以赋给其它类型，但void不行)

#### （2）任意值

Any：任意

普通类型在赋值过程中改变类型时不被允许的

枚举类型 自己定义一个属于自己的类型

Enum color {red, blue, green}

类型验证

Document.write(typeof varName)

类型别名

Type newNumber = number;

打印数字：num+””，接一个空串转为字符串

模板字符串：’string’

可换行，可使用${var}

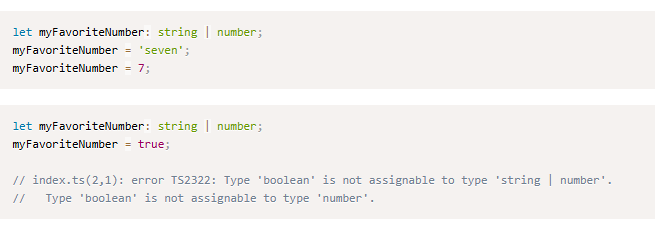
#### （3）类型推论

TypeScript 会在没有明确的指定类型的时候推测出一个类型，这就是类型推论。

如果定义的时候没有赋值，不管之后有没有赋值，都会被推断成 any 类型而完全不被类型检查：

#### （4）联合类型

联合类型 let num:string|number



当 TypeScript 不确定一个联合类型的变量到底是哪个类型的时候，我们只能访问此联合类型的所有类型里共有的属性或方法。

#### （5）变量声明

自动推断

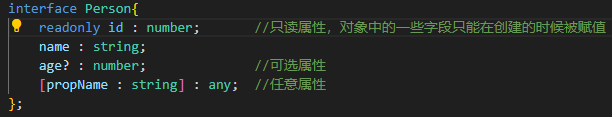
声明变量：let/var

声明常量：const

使用“varName:type”指定变量类型，let myVar:string = “Lucida”;

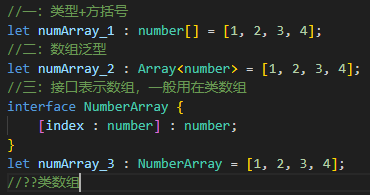
### 1.2 对象类型：接口

interface



### 1.3 数组类型

“类型+方括号”表示法



类数组？？

Any大法好：



### 1.4 函数类型

定义方式：函数声明、函数表达式



函数表达式=匿名函数

参数和返回值必须匹配

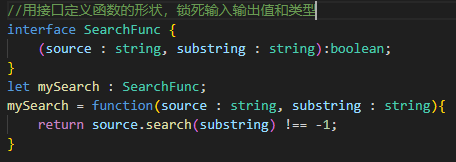
#### （1）函数表达式

①类型自动推论；

②手动添加类型，使用“=>”运算符，左边是输入类型，右边输出类型；

#### （2）接口定义函数形状

采用函数表达式|接口定义函数的方式时，对等号左侧进行类型限制，可以**保证以后对函数名赋值时保证参数个数、参数类型、返回值类型不变**。



#### （3）可选参数

使用？

可选参数必须接在必需参数后面

#### （4）参数默认值

Name : type = value

#### （5）剩余参数



#### （6）重载

？？？

### 1.5 类型断言

魔法？？？

法一：Value as type

法二：<type>value

类型断言（Type Assertion）可以用来手动指定一个值的类型。

核心：要使得 A 能够被断言为 B，只需要 A 兼容 B 或 B 兼容 A 即可

注意：

类型断言只能够「欺骗」TypeScript 编译器，无法避免运行时的错误，反而滥用类型断言可能会导致运行时错误

类型断言只会影响 TypeScript 编译时的类型，类型断言语句在编译结果中会被删除；

#### （1）使用场景

①将一个联合类型断言为其中一个类型



②将一个父类断言为更加具体的子类

③将任何一个类型断言为Any

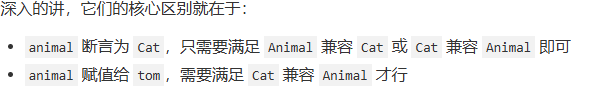
将一个变量断言为 any 可以说是解决 TypeScript 中类型问题的最后一个手段。

④将 any 断言为一个具体的类型

遇到烂代码为了避免Any类型继续传播，可以选择继续改进，通过累心断言把Any断言为精确的类型

#### （2）tips

类型断言 VS 类型声明



## 2.声明

[声明文件 · TypeScript 入门教程 (xcatliu.com)](https://ts.xcatliu.com/basics/declaration-files.html#%E6%96%B0%E8%AF%AD%E6%B3%95%E7%B4%A2%E5%BC%95)

需要注意的是，声明语句中只能定义类型，切勿在声明语句中定义具体的实现5：

类似C++里的typename？？

declare 定义的类型只会用于编译时的检查，编译结果中会被删除。

### 2.1 声明语句

？？？

Declare var

### 2.2 声明文件

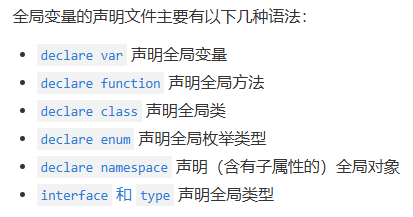
#include???

把声明语句放到一个单独的文件（jQuery.d.ts）中，这就是声明文件,

声明文件必需以 .d.ts 为后缀

？？声明文件是啥？怎么写？

### 2.3 全局变量



需要注意的是，声明语句中只能定义类型，切勿在声明语句中定义具体的实现5。

### 2.4 const声明

const 声明是声明变量的另一种方式。

与let声明相似，但是就像它的名字所表达的，它们被赋值后不能再改变。 换句话说，它们拥有与let相同的作用域规则，但是不能对它们重新赋值。

它们引用的值是不可变的，但是除非使用特殊的方法去避免，实际上const变量的内部状态是可修改的，可以设置对象的成员设置成只读的。

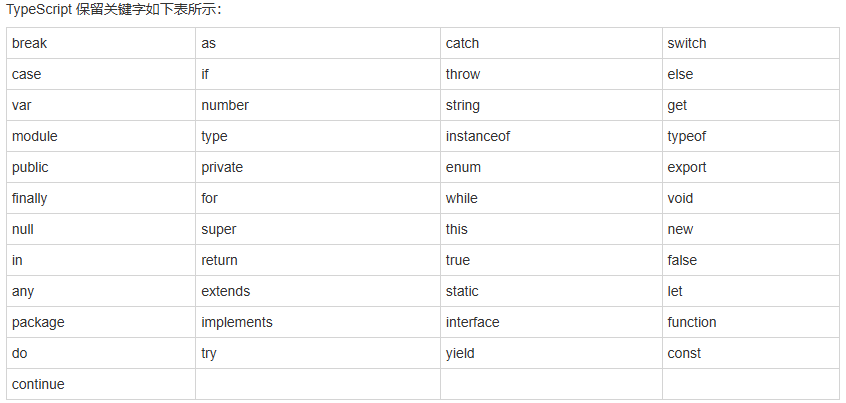
# 二、编程

TS程序组成

模块、函数、变量、语句、表达式和注释

## 1.基础

### 1.1 保留关键字



区分大小写；

每行分号可选，建议加上；

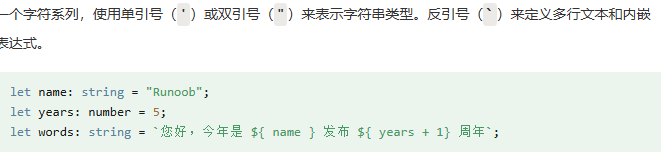
注释同C风格

### 1.2 基础类型

Any：任意类型

Number：双精度64为浮点数

String：单/双引号表示字符串类型



Boolean：bool类型，逻辑值

Type []：数组类型

[type\_1, type\_2]：元组类型

Enum：枚举

Void：用于标识方法返回值的类型，表示该方法没有返回值

Null：表示对象值缺失

Undefined：用于初始化变量为一个未定义的值

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。

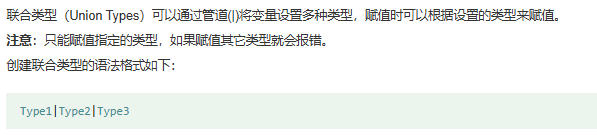
#### （1）Any类型

①变量的值会动态改变时，比如来自用户的输入，任意值类型可以让这些变量跳过编译阶段的类型检查；

②改写现有代码时，任意值允许在编译时可选择地包含或移除类型检查；

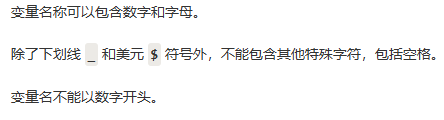
③定义存储各种类型数据的数组时。

#### （2）联合类型



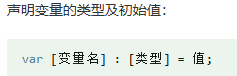
### 1.3 变量声明

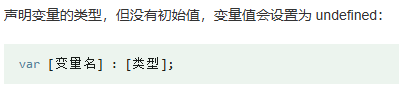
#### （1）命名规则



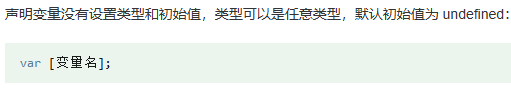
#### （2）声明方式

四种

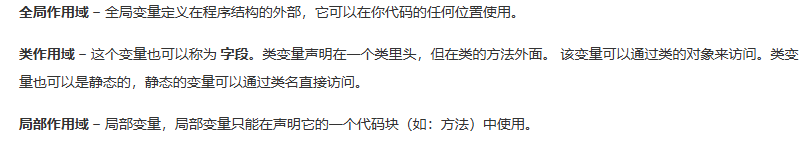






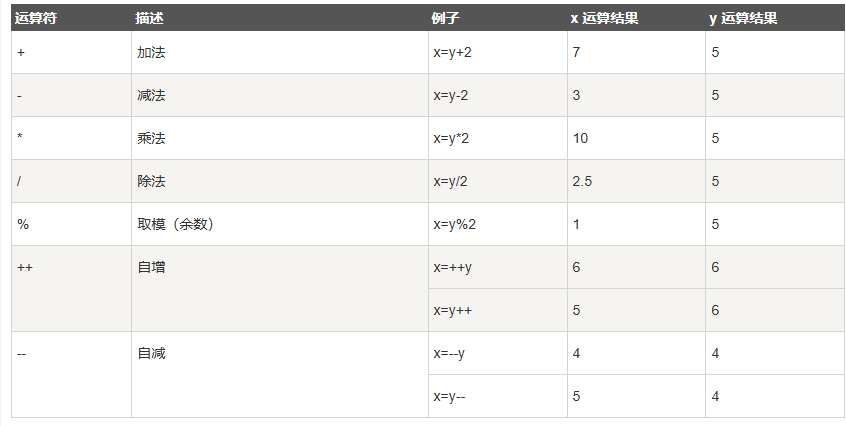


#### （3）变量作用域

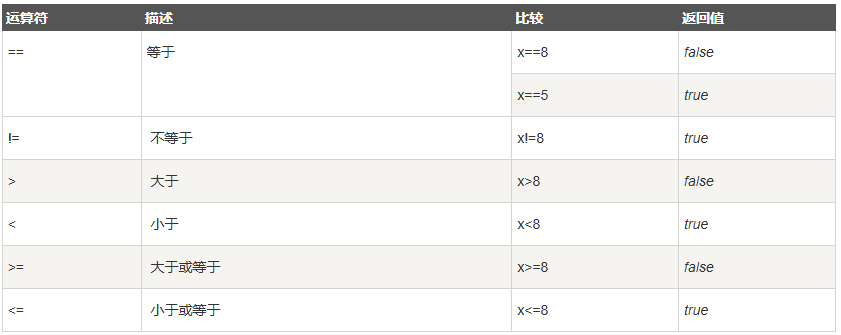


### 1.4 运算符

#### （1）算术运算符



#### （2）关系运算符



注意

“===”：等同符。当等号两边的值为相同类型的时候，直接比较等号两边的值，值相同则返回true，若等号两边的值类型不同时直接返回false。

“==”：等值符。当等号两边的值为相同类型时比较值是否相同，类型不同时会发生类型的自动转换，转换为相同的类型后再作比较。

#### （3）逻辑运算符

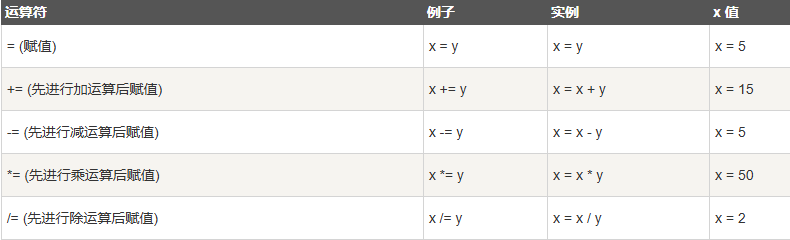


注意一下&&、||仍遵守短路规则

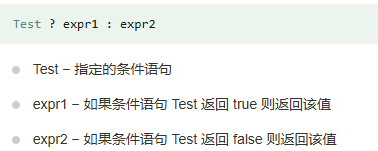
#### （4）位运算符



#### （5）赋值运算符



#### （6）三元运算符



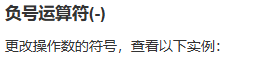
#### （7）类型运算符

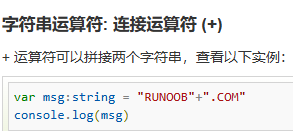
Typeof：一元运算符，返回操作数的数据类型

Instanceof：object instanceof class，判断对象是否为指定的类型（检查一个对象是否是某个类的实例）

类型检查：[typescript instanceof关键字详解|极客教程 (geek-docs.com)](https://geek-docs.com/typescript/typescript-questions/54_tk_1705012689.html)

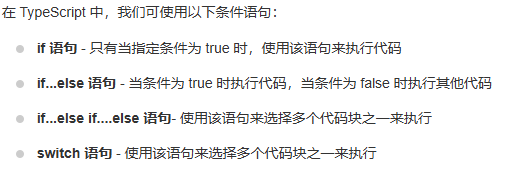
#### （8）其它运算符



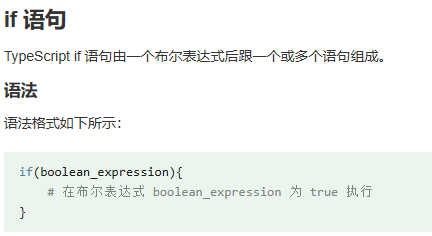


In：操作符可以安全的检查一个对象上是否存在一个属性，它通常也被作为类型保护使用

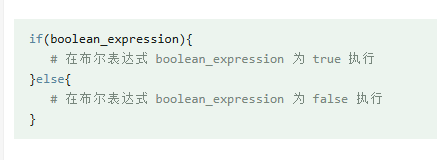
### 1.5 条件语句



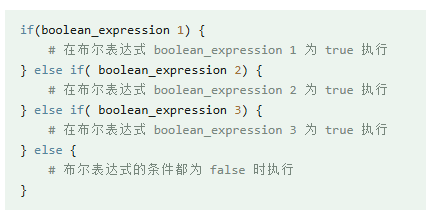
#### （1）if exp



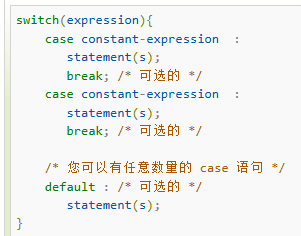
#### （2）if…else exp



#### （3）if…else if…else exp

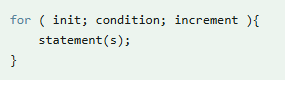


#### （4）switch…case exp



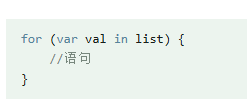
### 1.6 循环语句

#### （1）for



#### （2）for…in

for...in 语句用于一组值的集合或列表进行迭代输出，val类型需为string或any



#### （3）for…of 、forEach、every 和 some 循环

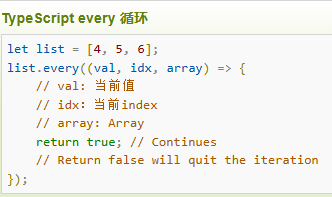


for...of 语句创建一个循环来迭代可迭代的对象，允许遍历 Arrays（数组）, Strings（字符串）, Maps（映射）, Sets（集合）等可迭代的数据结构等

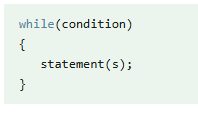
\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Foreach及every应该为可迭代容器得方法

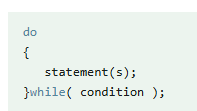




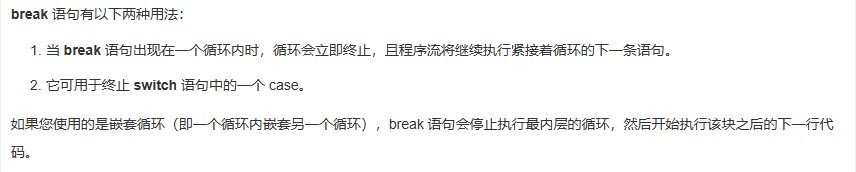
#### （4）while



#### （5）do…while



#### （6）break



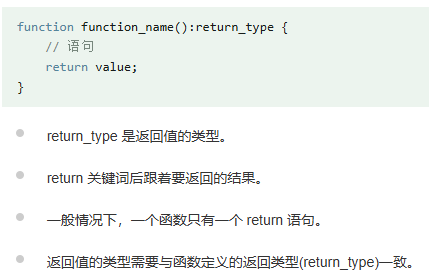
#### （7）continue



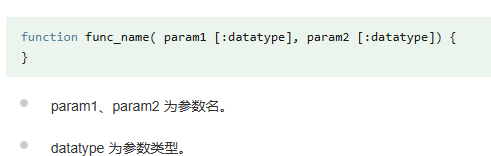
### 1.7 函数

定义、声明、调用略

#### （1）函数返回值



#### （2）带参数函数



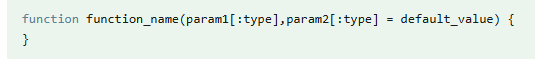
#### （3）可选参数

使用问号标识

可选参数必须放在必须参数后



#### （4）默认参数

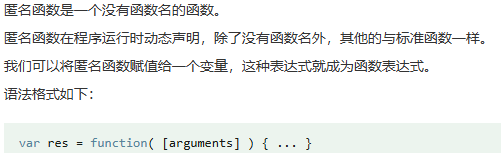


#### （5）剩余参数

？？解包？？



#### （6）匿名函数



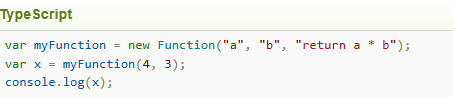
自调用：匿名函数自调用在函数后使用 () 即可

(function(args){statements;})(args)

#### （7）构造函数

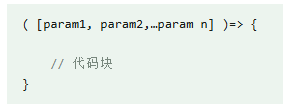
支持JS内置构造函数Function()来定义函数





#### （8）lambda函数

（没想到还有这个）



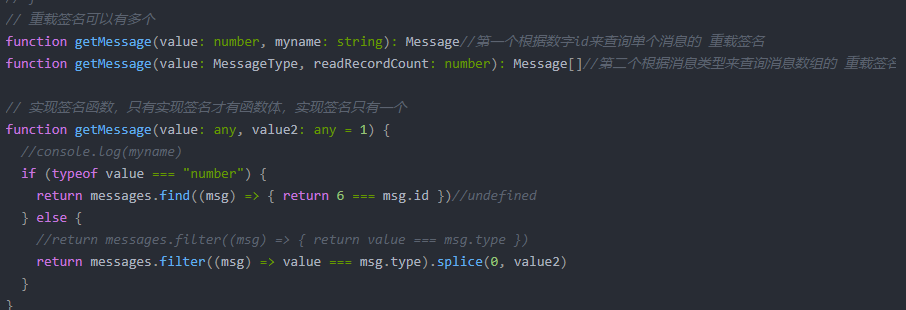
单个参数 **()** 是可选的；

无参数时可以设置空括号；

#### （9）函数重载★

参考：[深度掌握TypeScript中的重载【函数重载、方法重载】\_ts 方法重载-CSDN博客](https://blog.csdn.net/w1099690237/article/details/134822663)

重载签名可以有多个，但是实现签名只能有一个



### 1.8 内置对象

#### （1）Number

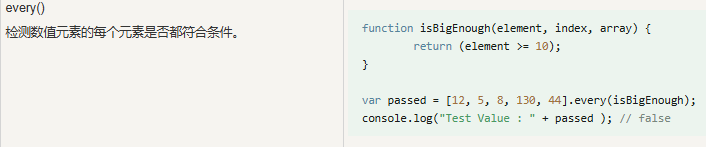




#### （2）String

#### （3）Array

Every方法



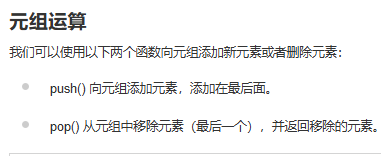
（类似于传入一个函数对象，改对象得返回值和参数类型已经制定）

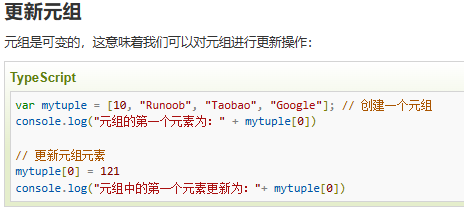
#### （4）map

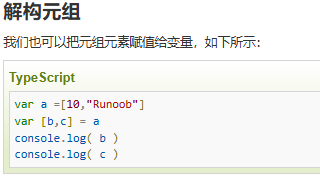
### 1.9 元组

创建：







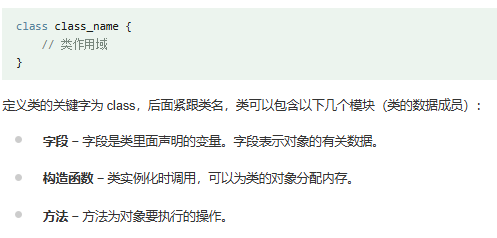


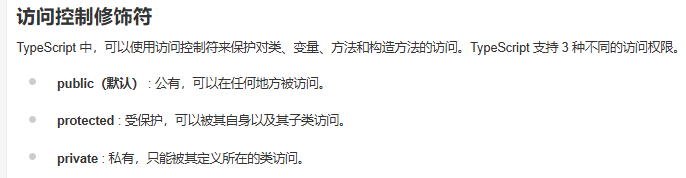
### 1.10 接口

Interface

Extends

### 1.11 类





实例化：new

继承：extends

调用父类：super

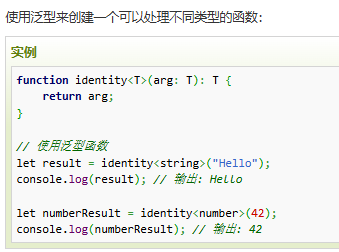
### 1.11 对象



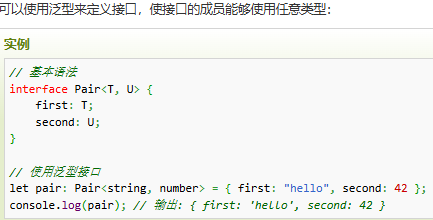
### 1.12 泛型



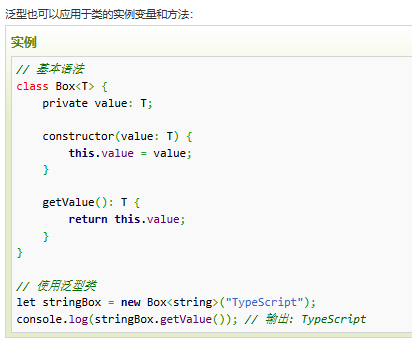
#### （1）泛型函数



#### （2）泛型接口

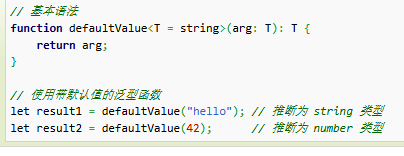


#### （3）泛型类



泛型约束：使用接口和继承

泛型与默认值



### 1.13 命名空间



### 1.14 模块

TypeScript 模块的设计理念是可以更换的组织代码。

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。

两个模块之间的关系是通过在文件级别上使用 import 和 export 建立的。



### 1.15 声明文件

[Typescript 书写声明文件（可能是最全的） - 掘金 (juejin.cn)](https://juejin.cn/post/6844904034621456398)

# 三、进阶

## 1.基础

### 1.1 类型别名



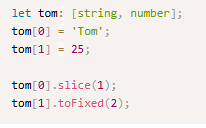
### 1.2 字符串字面量类型

字符串字面量类型用来约束取值只能是某几个字符串中的一个。

使用type关键字

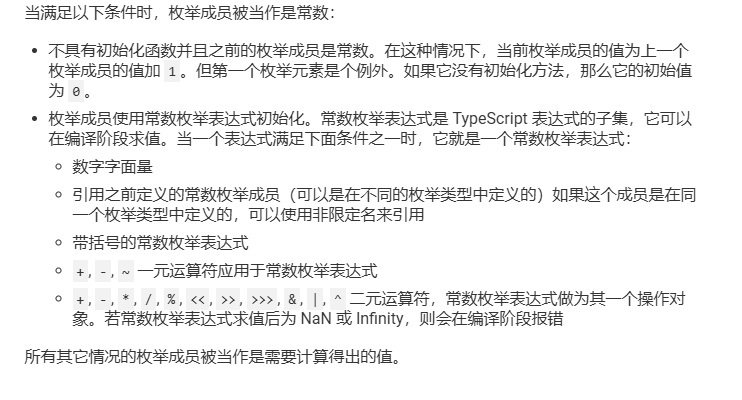
### 1.3 元组

数组合并了相同类型的对象，而元组（Tuple）合并了不同类型的对象。



### 1.4 枚举

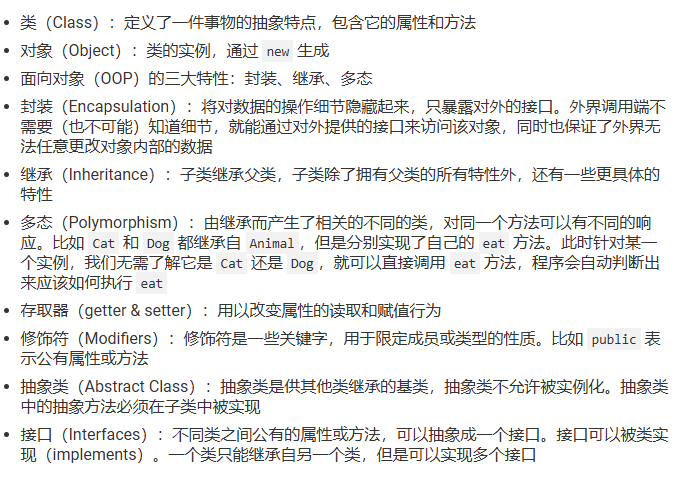




外部枚举：

使用declare enum

### 1.5 类



#### （1）基本语法

Class：定义类

Constructor：定义构造函数

New：生成实例

Extends：实现继承

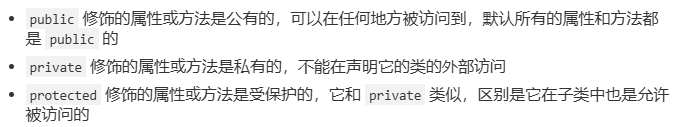
Super：调用父类

Static：修饰符，静态方法，直接通过类来调用

Abstract：定义抽象类

#### （2）类的用法

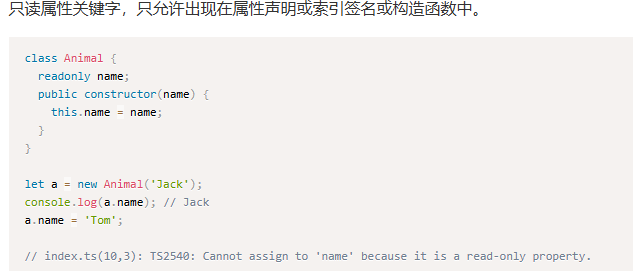
访问修饰符



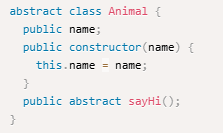
#### （3）参数属性

修饰符和readonly还可以使用在构造函数参数中，等同于类中定义该属性同时给该属性赋值，使代码更简洁。





#### （4）抽象类



抽象类不允许被实例化；

抽象类中的抽象方法必须被子类实现

### 1.6 类与接口

#### （1）类实现接口

实现（implements）是面向对象中的一个重要概念。一般来讲，一个类只能继承自另一个类，有时候不同类之间可以有一些共有的特性，这时候就可以把特性提取成接口（interfaces），用 implements 关键字来实现。

不同类之间提取共有的特性为接口interface，然后去实现，使用implement关键字

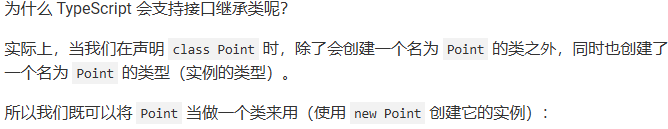


#### （2）接口继承接口

接口与接口之间可以是继承关系：

#### （3）接口继承类

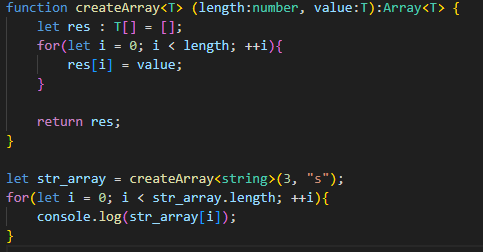
特殊：Typescript中接口可以继承类



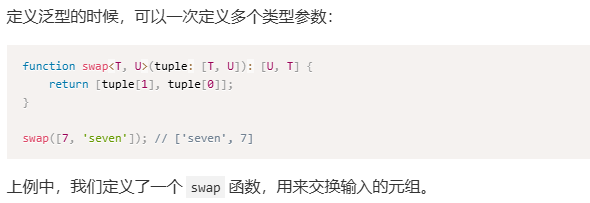
注意这里只继承了实例属性和实例方法，即不包含构造函数、静态属性方法

### 1.7 泛型

泛型（Generics）是指在定义函数、接口或类的时候，不预先指定具体的类型，而在使用的时候再指定类型的一种特性。



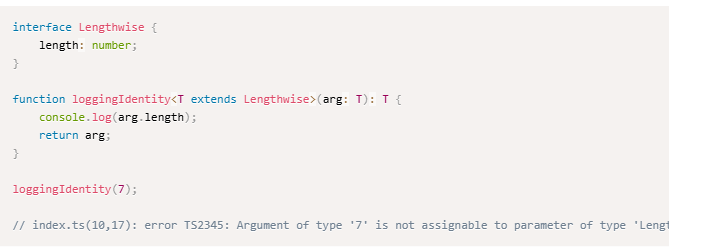
#### （1）多个类型参数



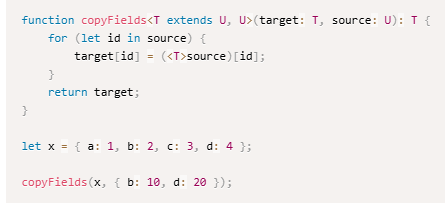
#### （2）泛型约束

传入的类型不一定有要用到的属性或方法，使用约束来限定传入类型的种类（or 符合某种条件的类型）

使用接口，使用extends



多个类型之间也可以相互约束



#### （3）泛型接口 ？

？？？？

#### （4）泛型类

## 2.工程

代码检查：[代码检查 · TypeScript 入门教程 (xcatliu.com)](https://ts.xcatliu.com/engineering/lint.html)

正确配置VS+TS+ESLint：[正确配置 Visual Studio Code 开发和调试 TypeScript - 优雅的前端 - SegmentFault 思否](https://segmentfault.com/a/1190000018777683)

# 四、伴生问题

## 1.NPM

[NPM 使用介绍 | 菜鸟教程 (runoob.com)](https://www.runoob.com/nodejs/nodejs-npm.html)

# X.参考

教程：[TypeScript 入门教程 (xcatliu.com)](https://ts.xcatliu.com/)

扩展阅读：[扩展阅读 · TypeScript 入门教程 (xcatliu.com)](https://ts.xcatliu.com/advanced/further-reading.html)

NPM：[TypeScript 阮一峰 | 阮一峰 TypeScript 教程 (p6p.net)](https://typescript.p6p.net/)

Git使用：[最全的git命令（详细）和对常见git操作流程讲解 - 个人文章 - SegmentFault 思否](https://segmentfault.com/a/1190000042347483)