

void类型，此方法没有返回值

let arr: number[] = [1, 2];

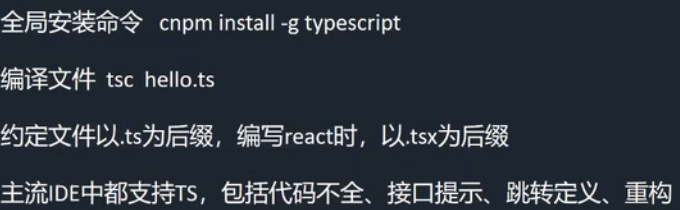
// 或者使用数组泛型

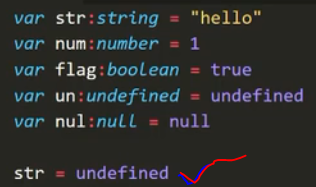
let arr: Array<number> = [1, 2];

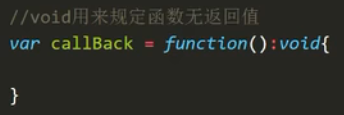
这里的泛型类型是number类型，所以元素只能是number

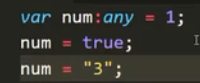


元组的数据类型需要对应

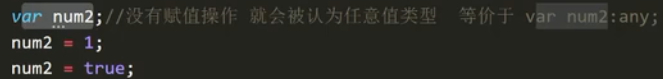


子类型null和undefined可以直接赋值给string

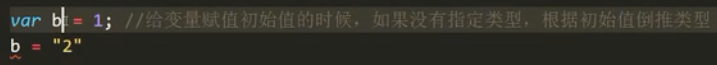




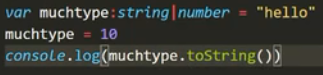
any任意类型



没有赋值操作，所以是any类型



已经赋值了，所以被反推为number类型，不能被赋值字符串

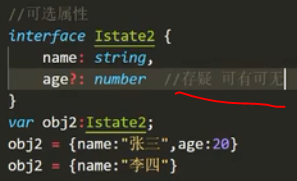
联合类型，此例可以是string或number

toString()是number和string共同支持的，所以可以是muchtype

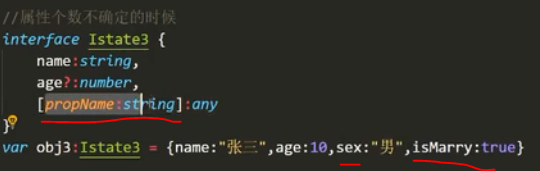
但是不能用muchtype.length，因为number没有这方法



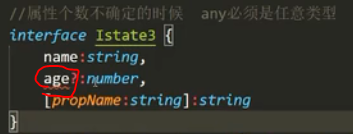
接口类型，赋值时必须实现里面的方法（属性）



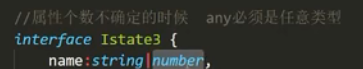
可选的实现



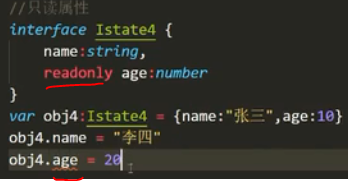
不确定属性的任意类型



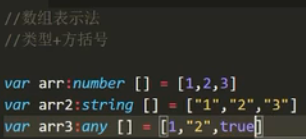
跟string冲突了，有了[propName:string]:string，那么在接口中只能有string类型



联合类型

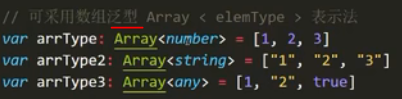


实现了这接口之后就不能再修改只读属性了

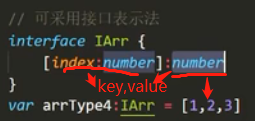


任意类型的数组，可以跟元组比较一下

元组是指定了类型和个数



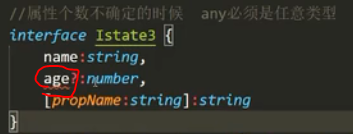
泛型数组

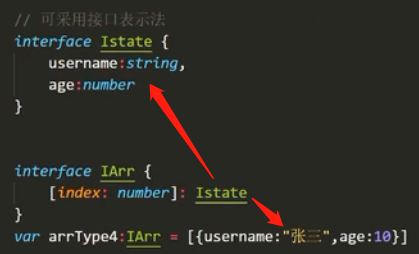


这里的key就是[1,2,3]的索引0,1,2，是number类型

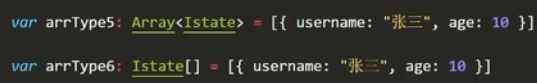
value就是[1,2,3]

属于一种个数不确定的情况，类似于下面

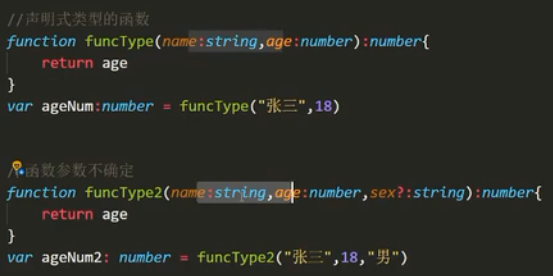




值是接口Istate类型



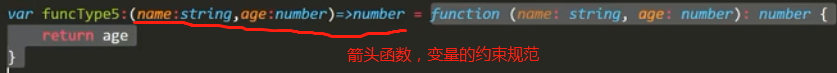
泛型，类型指定为Istate类型



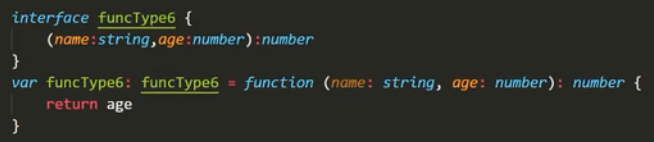
函数参数的不确定性



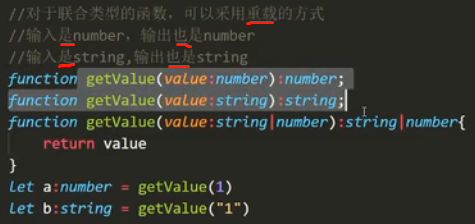
函数参数默认值



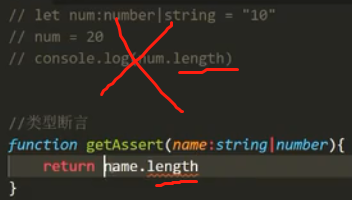
funcType5是函数类型，所以赋值了一个函数



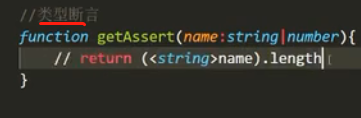
变量必须是一个函数，此处接口中的(name:string,age:number):number是函数（方法）



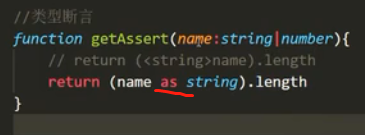
重载的联合函数



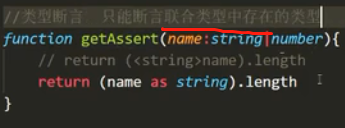
length不是string和number的共同属性，所以不能用



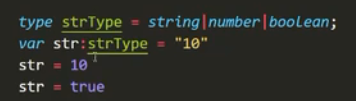
类型断言

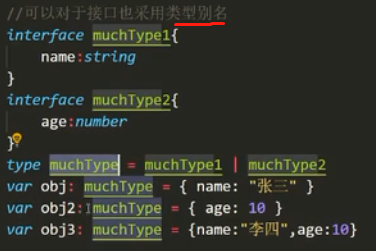


为了防止尖括号在编译时被当成标签，所以一般采用as的类型断言方式

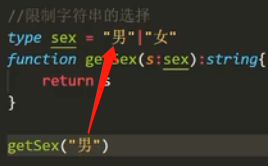


只能断言联合类型中存在的类型

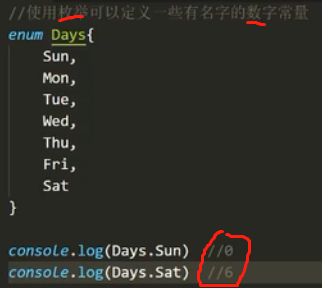




接口采用联合类型别名，可以两个接口都实现



联合类型（理解为联合值？）限定了传入的值必须是男女

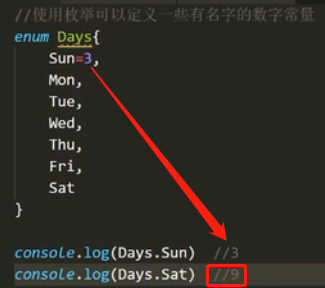


从0开始

C:\Users\ADMINI~1\AppData\Local\Temp\1575964253(1).png

双向映射

C:\Users\ADMINI~1\AppData\Local\Temp\1575964230(1).png



没有默认值就从0开始，这里有默认值从3开始

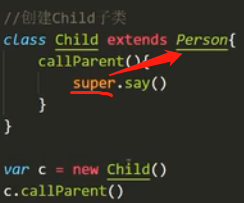


没有修饰就是public类型



C:\Users\ADMINI~1\AppData\Local\Temp\1575964661(1).png

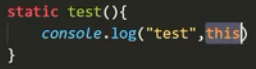
实例不能访问私有属性



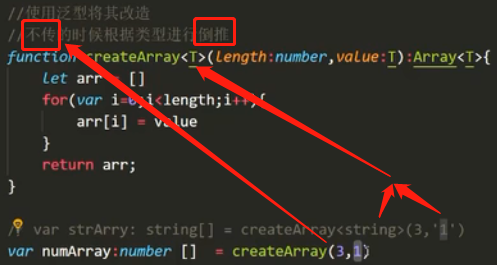
子类中的super就是父类



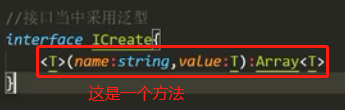
子类实例不能访问父类的受保护属性，但在子类中可以访问



静态中没有this属性，只有在实例中才有



泛型类型的倒推



接口中的键值对是一个方法



接口泛型方法的实现，也就是赋值一个函数（方法）



3反推出T是number,所以返回Array<T>是一个number数组

类型反推