接口的概念： 1. 用来描述数据的状态 2.接口中没有具体的实现（抽象）

接口可以描述函数，对象，以及类的概念。。。

例：

符号可以不写 可以分号 可以逗号

接口有自己的修饰符

Interface IV {

Color: string

Taste: string

Size: number

}

Let veg: IV = {

readonly Color: ‘red’

Taste: ‘sweet’

Size?: 3

}

?: 可选参数（和函数的可选参数一样）

readonly 只读属性

ts中语法语法 as const

例如：const x = { name: ‘zf’, age: 13 } as const;可以来表示属性是仅读的

接口中的属性如果没有+?

Interface IFruit {

Name: string

Color: string

Taste: ‘sweet’ | ‘sour’

}

Interface MyFruit extends IFruit {

Age: number

}

可以继承多个，继承多个要兼容多个

继承类

Let a !:MyType

Let r = a as string

Interface IArr { 可索引接口 可以通过索引来取值

[key:number]:any

}

Let myArr:IArr = { 0:1,1:2,3:2 }

以上描述对象

接口可以用来描述类和类的实例 采用抽象类来进行一个描述

抽象类添加abstract关键字 增加后此类不能被new 抽象类可以定义抽象方法，定义后的抽象方法或属性必须要子类实现

抽象类中可以存在非抽象的方法或者属性

abstract Class Animal {

abstract name: string

abstract speak():void

speak() { //已经实现好的

console.log(‘speak’)

}

}

class Cat extends Animal {

name=’猫’

eat(): void {}

}

class Tom extends Cat {}

接口同样可以限制类型 接口中都是抽象的 不能有具体的实现

interface IEat {

eat: () => void

}

泛型

在定义类型的时候，不确定是什么类型，需要等待使用的时候才能确定类型 用T来表示

class Cat {

constructor(public name:string, public age: number) {}

}

class Dog {

constructor(public type:string) {}

}

function getInstance<T,K>(…args:any[]) => T, name: K, age ?: number) :K {

return name

}

let r = getInstance(Cat, 1)

泛型就是根据传入的类型来 决定对应的类型

interface ICallback(T) {

(item: T): void

}

type IForEach = <T>(arr:T[ ], callback: ICallback<T>) => void;

let forEach = (arr, cb) {

for (let i = 0; I < arr.length; i++) {

cb(arr[i])

}

}

forEach([1,2,3,5], item => {

console.log(item)

})

交换参数

function swap<T, K>(tuple: [T, K]): [K, T] {}

无法保证两个泛型相加T+T=>T 借用泛型约束extends

function sum<T extends number>(a: T, b: T) {

return a + b

}

type IA = { a: 1 };

type IB = { b: 1 };

条件分发

function getVal<T extends IA | IB>(val: T) {}

交叉类型 IA & IB

let o1 = { name: ‘zf’, age: 12 };

keyof 取某个 泛型中所有key

function getValueFromKey<T extends object, K extends keyof T>(obj: T, key: K) {

return obj[key]

}

let val = getValueFromKey(o1, ‘age’);