TypeScript

基础内容

初识TS

- 安装`npm i typescript -g`
- 查看版本`tsc -v`
- 编译ts文件`tsc ts文件.ts`生成js文件并运行
- 浏览器不支持ts，始于js，终于js
- 使用工具
  - `npm install @types/node -D`
  - `npm i ts-node -g`
  - 命令行输入`ts-node ts文件.ts`直接使用node运行ts文件

基础类型

```
// 字符串
let str:string = "TS"
let muban:string = `web ${str}`
console.log(muban)

// 数值(NaN/Infinity无穷大/0xf00d十六进制/0b1010二进制/0o744八进制)
let num1:number = NaN
let num2:number = 0xf00d

// 布尔值
let b:boolean = false
console.log(b)   // false
let a:boolean = Boolean(1)
console.log(a)   // true  隐式转换
let a:boolean = Boolean(1)
console.log(a)
let a:Boolean = new Boolean(1) // new返回的是对象，用大写Boolean接收
console.log(a)

// 空值
//  - js中没有空值(Void)的概念，ts中用void表示没有任何返回值的函数
function fnVoid():void {
    return //此函数将不能有返回值，可以不写或为空
}
//  - 也可以定义 undefined 和 null 类型
let u:void = undefined
let n:void = null
console.log(u,n)  // undefined null

// undefined
let u2:undefined = undefined
// null
let n2:null = null
```

注意点

- void定义的空值不能赋值给其他变量

```
let v:void = undefined
let srt:string = '123'
// str = v   此时编译会报错！！！
```

任意类型

```
// any类型（可以任意赋值，类似js变量）
let anys:any = "我很大，你小心一点"
anys = 18
anys = {}
anys = Symbol("123")

// unknown
// - unknown比any类型更安全，不能取调用属性和方法，不能是引用类型
// - unknown类型只能作为父类型，不能充当子类型，也就是不能赋值给别人
// - unknown类型只能赋值给any类型或unknown类型
let un:unknown = "我很大，你小心一点"
un = 18
// un.a  直接报错!
```

接口和对象类型

Object、object、`{}`

```
// Object 与原型链有关,原型链的顶端，
// ts中 Object包含了所有类型，类似于与any或function
let a:Object = 1
let a1:Object = {}
let a2:Object = []
let a:Object = "123"

// object 常用于泛指约束，代表非原始类型
let b:object = '123 '    //错误，不支持原始类型（数字、字符串、布尔）
let b1: object = []      //正确，支持所有引用类型（函数、数组、对象）

// {} 可以理解为 new Object(),支持所有的类型
let c:{} = {name:1}
c.age=18   // 错误，对象类型无法对变量进行赋值操作
```

对象类型

- interface （接口）声明对象的关键词，类似于类的定义
  - 声明出现重名时会进行自动合并
- readonly 设置属性为`只读`，定义后不可修改
  - 指针只读，引用类型中的内容仍可以修改
- ? 可选值属性，在定义对象时可为空
- `[propName:String]:any` 任意属性，可以在对象中有其他任意类型
  - `[propName:String]:String | number` 任意属性、联合类型，对象中可以使用多种数据类型
- 函数类型 `()`
- 合并继承 `interface B extends A { ... }`

```
// 声明对象  interface约束接口
interface 命名A{
    name:String
}

interface 命名A{     // 重名时会自动合并
    readonly age:number,     // readonly 只读，该属性不能做修改
    sex?:String     // ? 表示可选值属性，在定义对象时可为空
    [propName:String]:any   // [propName:String]  任意属性，可以在对象中有其他任意类型
}

// 定义对象
let obj1:命名A = {name:"夏之一周"} // 报错，未指定必填的age属性
let obj2:命名A = {name:"夏之一周",age:15} // 正确，sex为可选值
let obj3:命名A = {name:"夏之一周",age:15，sex:"男"} // 正确
obj2.age=18 // 报错，无法分配到name，因为它是只读属性

interface Person{
    name:String,
    age:number,                 // 报错，因为任意类型定义为String，不能使用 number
    [propName:String]:String    // 任意属性，且指定为字符类型，所以不能使用其他类型
}

[propName:String]:String | number  // 任意属性，联合类型可以使用String、number类型


// 声明对象中的函数类型
interface Dog{
    name:String,
    cd():number,         // void表示函数无返回值，也可以为number/string等
}
// 定义带有函数类型的对象
let Dog1={  name:"哈巴狗",
            cb:():number=>{ return 123 }
         }


// 合并继承
interface A{
```

```
    name:String
}
interface B extends A{
    age:number
}
let b={name:"法外狂徒张三",age:18}
```

数组

```
// 定义数组普通类型 直接限定类型
let arr:number[] = [1,2,3]  // 定义number类型的数组
let arr:boolean[] = [true,false]  // 定义boolean类型的数组
或
let arr:Array<boolean> = [true,false]

// 定义对象数组,使用interface
interface X {
    name:string,
    age?:number,      // ? 表示该项可有可无
}
let arr:X[] = [{name:"夏之一周"},{name:"红烧肉",age:16}]

// 定义多维数组,几个[]表示几维
let arr:number[][] = [[1],[20],[3]]
或
let arr:Array<Array<boolean>> = [[true],[false]]

// 大杂烩，任意类型
let arr:any[] = [1,"夏天", true]
// 大杂烩，指定元组，一一对象
let arr:[number,string,boolean] = [1,"夏天", true]

// 函数参数的接受时定义类型、arguments这种`类数组`的定义- IArguments
function a(...args:any[]){
    console.log(args)
    let abc:IArguments = arguments
}
a("1",2)

// IArguments的原理:
interface A {
    callee:Function
    length:number
    [index:number]:any
}
```

函数

- 函数重载：方法名相同、参数不同；返回值可同可不同
    - 如果参数类型不同，则操作函数参数类型应设置为`any`
    - 如果参数数量不同，可将不同的参数设置为可选值

```
// 函数传参类型限制
// 参数个数、参数类型、返回值类型
// 如果指定传参的默认值，可在调用时不传，否则为必传值
// 加 ? 表示可选值，默认为undefined
const fn = function(name:string,age:number=30,sex?:string):string {
    return name+age
}

// 数组/对象 形式
interface User {
    name:string,
    age:number
}
const fn = function(user:User):User{
    return user
}
let a=fn({"张三",55})




// 重载函数（规则） 定义,可以是多套
function fn1 = function(params:number):void
function fn1 = function(parmas:string,params2:any):void
// 执行函数逻辑的定义（应满足上述所有规则要求）,执行时会从上述规则中选择执行，具体根据使用时的传参来定
function fn1 = function(parmas:any,parmas2?:any):void{
    console.log(parmas+parmas2)
}

fn1(1) // 执行第一个规则
fn("夏之一周") //执行第二个规则
```

联合类型|类型断言|交叉类型

- 联合类型：为某个值指定的多种类型
- 交叉类型：合并多个属性
- 类型断言：推断取值的类型并执行某些操作，
    - 使用中需额外谨慎，避免出现程序运行的错误

```
// 联合类型，可定义为指定的多种类型
// 手机号可能是number/string 类型时：
```

```ts
let phone:number | string = 123456
// 函数传参可能是多种类型
let fn = function(type:number|boolean):boolean{
    return !!type
}


// 交叉类型 合并多个属性
interface People{
    name:String,
    age:number
}
interface Man{
    sex:number
}
const xiaoming = (man:People & MaN):void=>{
   console.log(man)
}
xiaoming({
    name:"交叉属性",
    age:15,
    sex:1
})


// 类型断言
let fn = function (num:number | strimg):void {

}
```