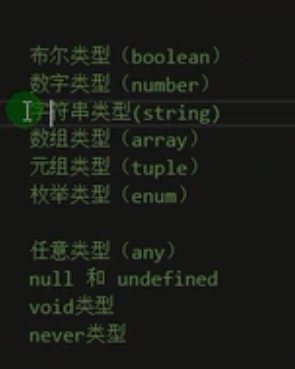
# Typescript

# 1.安装

1. npm i -g typescript
2. tsc hellowrod.ts 编译
3. tsc init 下载tsconfig.json文件 配置输出文件位置 “outDir:’’.js”
4. vscode 终端 运行 监视

# 2.类型



// boolean布尔类型

var issure: boolean = false

console.log(issure)

// number类型

var number: number = 5

console.log(number)

// 字符串类型

var str: string = "你好"

console.log(str)

// arr 数组类型（两种）

// 1.

var arr1: number[] = [1, 2, 3, 4]

console.log(arr1)

// 2.

var arr2: Array<number> = [1, 2, 3, 22]

console.log(arr2)

//tuple 元组类型，属于数组的一种

var tuple: [number, string] = [1, "22"]

console.log(tuple)

// enum 枚举类型

enum Status {

success = 1, error = 2

}

var s: Status = Status.success

console.log(s)

// 没有赋值是打印下标

enum Color { red, blue, green }

var c: Color = Color.green

console.log(c)

// 任意类型 any

var any: any = 122

any = [1, '52']

console.log(any)

var dom: any = document.getElementsByClassName("box")[0];

dom.style.color = "red"

// null undefined

var unde: undefined;

console.log(unde)

var unde1: undefined | number;

unde1 = 1122

console.log(unde1)

// void 定义方法表示没有返回值

function fun1(): void {

console.log('run')

}

fun1()

// never 其他类型 包括null undefined代表从不会出现的值

// var a: never；

// a=3(错误类型)

var a: never

a = (() => {

throw new Error("弄错了")

})()

# 3.函数

## 1.函数定义

// 1.

function fun0(): number {

console.log(1111)

return 1

}

fun0()

var fun2 = function (): number {

console.log('fn2')

return 1

}

fun2()

## 2.传参

### 可选参数

// 传递参数,可选参数，必须配置最后面

function fun3(i: number, v?: string): string {

console.log(i, v)

return 'www'+v

}

var funhs = function (i: string): string {

return `${i}`

}

### 默认参数

function fun4(i: number = 6): string {

console.log(i)

return 'w'

}

fun4(9)

### 剩余参数

// 剩余参数

function fun5(a: number, ...result: number[]): number {

var i = a;

for (var j = 0; j < result.length; j++) {

i += result[j]

}

console.log(i)

return i

}

fun5(1, 2, 3, 4)

### 函数重载

// 函数重载

function fun6(i: number): string;

function fun6(i: string): number;

function fun6(i: any): any {

if (typeof i === 'number') {

return i

} else {

return i

}

}

fun6('www')

# 4.es5类的继承

## 1. 实例方法

function web(){

this.name=’张三’

this.fun1=function(){//实例方法

return ’aaaa’

}

}

//原型链方法

web.prototype.fun2=function(){

return this.name+”13岁”

}

Var webNew=new web()

alert(webNew.name)

alert(webNew. fun2())

## 2.对象冒充继承

function web(){

this.name=’张三’

this.fun1=function(){//实例方法

return ’aaaa’

}

}

//原型链方法

web.prototype.fun2=function(){

return this.name+”13岁”

}

function person(){

web.call(this)

}

var personNew=new person()

alert(personNew.name)

alert(personNew. fun2())//undefined

存在问题： 对象冒充继承只能继承构造函数里面的属性和方法，不能继承原型链的属性和方法

## 3.原型链继承

function web(){

this.name=’张三’

this.fun1=function(){//实例方法

return ’aaaa’

}

}

//原型链方法

web.prototype.fun2=function(){

return this.name+”13岁”

}

function person(){}

person.prototype=new web()

var personNew=new person()

alert(person.name)

alert(person.fun2())

属性方法虽然继承了 但是还有一种情况就是传参的时候获取不到

function web(name,age){

this.name=name

this.fun1=function(){//实例方法

return this.name

}

}

//原型链方法

web.prototype.fun2=function(){

return this.name+age

}

function person(){}

person.prototype=new web()

var personNew=new person(‘王五’,13)

alert(person.name)//undefined

alert(person.fun2())//undefined

## 4.对象冒充原型链混合继承

function web(name,age){

this.name=name

this.fun1=function(){//实例方法

return this.name

}

}

//原型链方法

web.prototype.fun2=function(){

return this.name+age

}

function person(name,age){

web.call(this, name,age)

}

person.prototype=new web()

var personNew=new person(‘王五’,13)

alert(person.name)

alert(person.fun2())

# ts定义类

## 1.定义

class person {

name:string

contructor(name:strign){

this.name=name

}

Run():string{

Return this.name

}

}

Var new=new person(‘zhangsan ’)

Alert(new.run())

## 2.继承extends Super

class person {

name:string

contructor(name:strign){

this.name=name

}

Run():string{

Return this.name

}

}

Class web extends person{  
 contructor(name:strign){

Super(name)

}

}

Var new=new web(‘网站’);

Alert(new.run())

## 3.类的修饰符

// public 公有 当前类 子类 类外部可以访问 默认

// protected 保护类型 当前类 子类可以访问

// private 私有 当前类可以访问

## 4. 静态属性，静态类

Class person{

Static age:number=1//静态属性

Name:string

Contructor(name:sting){

This.name=name

}

Run():void{

Alert(this.name)

}

Static work():void{ 、//静态方法

Alert(this.name)

}

}

Var new=new person()

New.run()

Person.work()

## 5.多态

/ 类的多态

class personOne {

name: string

constructor(name: string) {

this.name = name

}

run():any{

alert('我是父类')

}

}

class cat extends personOne{

constructor(name: string) {

super(name)

}

run():void{

alert('我是猫'+this.name)

}

}

class dog extends personOne{

constructor(name: string) {

super(name)

}

run():void{

alert('我是狗'+this.name)

}

}

var animal=new cat('吃老鼠')

animal.run()

var animal1=new dog('吃肉')

animal1.run()

// 定义方法自己不去实现，他的子类去实现,每个子类有自己的表现

## 抽象方法

abstract class personOne {

name: string

constructor(name: string) {

this.name = name

}

abstract run():any

}

class child extends personOne{

constructor(name: string) {

super(name)

}

run():any{

alert(this.name)

}

}

var child1=new child('张三')

child1.run()

// 定义抽象方法里面不能被实例化，必须子类来进行实现

# 6.接口

## 1.定义传入的参数

function person(name:string):void{

}

person('www')

function person1(label:{name:string}):void{

}

person1({name:'账单'})

## 2.对象的约束 interface

// 对象的约束

interface each{

name:string;

age:number

}

function person(restrict:each){

alert(restrict.name+'--'+restrict.age)

}

person({name:'sss',age:5})

## 3.可选属性

//可选属性

interface each{

name?:string;

age:number

}

function person(restrict:each){

alert(restrict.name+'--'+restrict.age)

}

person({age:5})

## 4.原生请求封装

// 原生请求封装

interface config{

type:string;

url: string;

data?:string;

dataType:string

}

function ajax(config:config){

var xhr=new XMLHttpRequest();

xhr.open(config.type,config.url,true);

xhr.send(config.data)

xhr.onreadystatechange=function(){

if(xhr.readyState==4&&xhr.status==200){

}

}

}

ajax({

type:'get',

url: 'http://a.itying.com/api/productlist',

data:'name=zhagnsan',

dataType:'json'

})

## 5.函数类型接口 对方法传入的参数以及返回参数进行约束

interface person{

(name:string,age:number):string

}

var new1:person=function(name:string,age:number):string{

return name+age

}

console.log(new1('zhanfsan ',44))

## 对类的约束

1. interface anima{
2. name:string;
3. eat(str:string):void
4. }
5. class dog implements anima{
6. name:string
7. constructor(name:string){
8. this.name=name
9. }
10. eat(){
11. alert(this.name)
12. }
13. }
14. var new9=new dog('够');
15. new9.eat()

## 8.接口扩展

// 接口扩展

interface animal{

eat():void

}

interface peroson extends animal{

watch():void

}

class web implements peroson{

name:string

constructor(name:string){

this.name=name

}

eat():void{

console.log(this.name+'我是animal')

}

watch():void{

console.log(this.name+'我是person')

}

}

var per=new web('菜菜')

per.eat()

per.watch()

# 7.泛型

/function web(val:any):any{

}

// 要求传入类型与返回数据类型一样 泛型

function web<T>(val:T):T{

// return val

// }

web<string>('ww')

// 要求传入类型与返回数据类型不一样 泛型

function web<T>(val:T):any{

// return val

// }

web('ww')