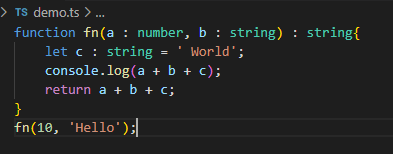
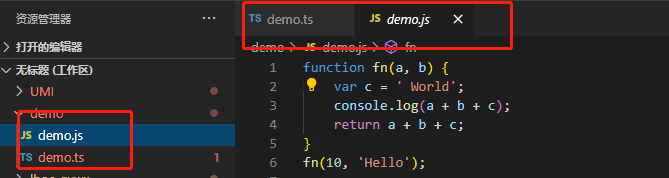
**TypeScript学习总结**

1. **typescript开发环境搭建**
2. 安装node环境
3. 全局安装typescript（npm install typescript -g / yarn global add typescript） 控制台输 入tsc -v查看ts版本
4. 建立项目：本地建立ts项目，创建.ts文件



1. 执行.ts文件 直接node demo.ts 输出错误，node不能直接执行.ts文件，首先通过t tsc demo.ts 将ts文件编译成js文件，然后通过node demo.js执行



1. ts-node安装与使用（a）直接执行.ts文件
2. **Typescript静态类型**

定义：简单理解，就是你变量一旦定了类型，就不可以再改变类型了（let m : number = 10即原则上m永远是number类型，不能改变成其它类型）

1. 基础类型：number、string、boolean、undefined、null、symbol、void(理解为：undefined和null的整合，即void可代表任一值)
2. 对象类型：
3. Object对象定义：

① let obj : object = {...}

② let obj : {a: number, b: string, ...} = {a: 10, b: ‘20’, ...}

2. Array数组定义：

① let arr : string[] = [‘a’, ‘b’, ‘c’, ...]

② let arr : (string | number|boolean)[] = [1, 2, ‘a’, ‘b’, 3, ‘c’, ...]

③ let arr : Array<string | number> = [‘a’, 10, ...];

3. Tuple元祖定义：(简单理解：数据初始化时，数组每一项的类型和数组长度固定)

① let arr : [number, string] = [10, ‘a’]

② let arr : Array<[number,string]> = [[10, ‘a’]]

4. 枚举类型定义：

① enum color {red, yellow, blue, black, green} => 调用方式：color[2] === ‘blue’ color.blue === 2（适用场景：职务类型、身份类型、状态等）

5. 类对象定义

① let obj : Date = new Date();

6. 函数定义：

① let fn : Function = function () {...}

② let fn : () => void = () => {...}

1. 其它类型：
2. any类型：定义任意类型变量，不作任何约束，编译时会跳过对其类型检查(慎用any类型，会造成问题比较多，比如：类型污染等)
3. let obj : any = 10 | ‘aa’| true...

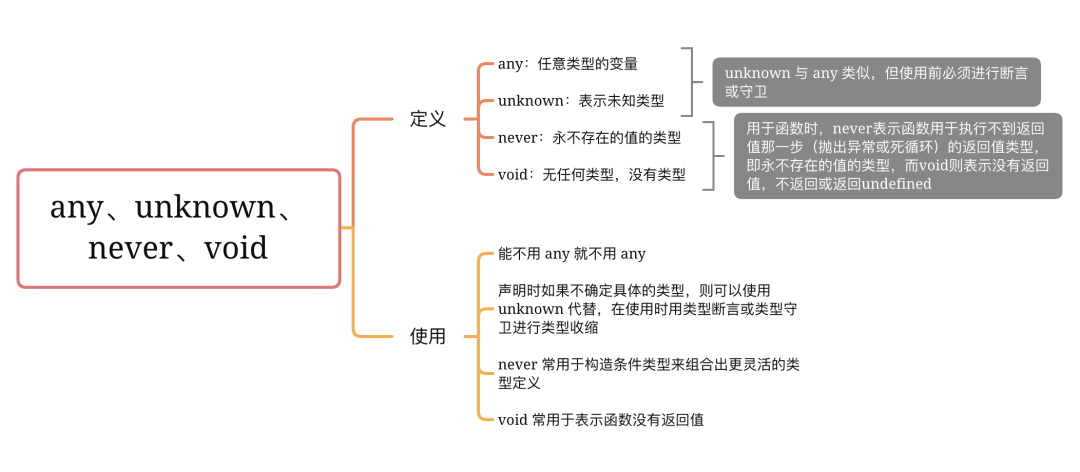
2. unknown类型：定义未知类型，即写代码的时候还不知道具体会是怎样的数据类型

1. let obj : unknown = 10 | ‘aa’| true...

3. never类型：用不存在值的类型



**重点问题？：any、unknown、never、void区别**



4. 联合类型：

① let obj : string | number | boolean = ‘aa’| 10 | true...

5. 自定义类型：(接口interface、类型别名type、class类)

① 接口interface

interface list {name: string, age: number, man: boolean, ...}

let obj : list = {name: ‘a’, age: 20, man: true}

1. 类型别名type

type list = {a: number, b: string, c: boolean, ...} | (string | number | boolean) | ...

let arr : list[] = [{a: 10, b: ‘aa’, c: true, ...}]

let value : list = ‘aaa’ | 100 | true | ...

③ class类

class List{

name: string;

age: number;

btn: boolean;

...

}

let arr : List[] = [{name: ‘a’, age: 30, btn: true, ...}];

思考：setTimeout等定时器定义为什么类型？

1. **Typescript中的类型注解和类型推断（重点）**

类型注解定义：简单理解，就是初始化数据的时候，就要定义好数据的类型

let str : string = ‘abc’;

类型推断定义： 简单理解，就是数据初始化完成以后，使用数据时可以推断出数据的类型 let m : number | string = 100; m调用时，为number类型

1. **Typescript函数参数和返回值类型定义**
2. 函数有参数和返回值的定义

function fn(a: number, b: number): number{return a+b;}

1. 函数无返回值的定义

function fn(): void{console.log(100);}

1. never返回值类型（场景：函数永远执行不完）

function fn(): never{throw new Error();console.log(‘error’);}

1. 函数参数为object的解构

function fn({a, b}: {a: number, b: number, c: string, ...}) : number{return a+b;}

1. **Typescript数组类型**
2. 基础数组类型

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

1. 联合数组类型

Let arr : (number | string)[] = [1, ‘a’, ‘b’, 2, ...]

1. 数组中对象类型

Let arr : {a: number, b: string}[] = [{a: 10, b: ‘aa’}, ...];

1. 类型别名定义

type obj = {a: number, b: string, c: boolean};

Let arr : obj[] = [{a: 10, b: ‘aaa’, c: true}, ...]

1. class类定义

class List{

name: string;

age: number;

btn: boolean;

...

}

Let arr : List[] = [{name: ‘a’, age: 30, btn: true, ...}];

1. **Typescript元组类型**

定义：简单理解就是一个数组，数组初始化时，数组的长度和数组每项的类型就固定了

let arr : [number, string, boolean, string, ...] = [10, ‘aa’, false, ‘bb’, ...]

let arr: [number, string, boolean][] = [

[10, ‘a’, true],

[20, ‘b’, false],

...

];

let arr : Array<[number, string, Boolean]> = [

[10, ‘a’, true],

[20, ‘b’, false],

...

];

1. **Typescript中的interface接口**
2. interface接口类型定义

interface List {a: number, b: string, ...}

let obj : List = {a: 10, b: ‘aaa’, ...}

1. interface和type的区别

type类型别名，可以直接赋值类型，而interface不能直接赋值类型

type val = string;

interface Cont { val : string }

1. 接口非必选值定义（?）

interface List {

name: string,

age: number,

btn: boolean,

love?: string

}

let obj : List = {name: ‘wang’, age: 20, btn: false, love: ‘sports’}

let obj2: List = {name: ‘ping’, age: 18, btn: true}

1. 接口允许加入任意值

interface List {

name: string,

age: number,

btn: boolean,

love?: string

[propname: string]: (string | number | boolean) | any

}

let obj2: List = {name: ‘ping’, age: 18, btn: true, love: ‘run’, sex: ‘man’, val: {a: 11, b: ‘22’, ...}, list: [1, 2, true, ‘111’, ...]}

1. 接口里的方法(fn: Function | fn: ()=> string | fn(): string)

interface List {

name: string,

age: number,

fn(): string,

love?: string

[propname: string]: (string | number | boolean) | any

}

let obj : List = {name: ‘a’,age: 10, fn: function(){return ‘aa’}}

1. 接口对类的约束（implements）

interface List {

name: string,

age: number,

status: boolean,

...

}

class Person implements List{

name = ‘aaa’;

age = 10;

status = true;

sex = ‘man’;

love = ‘sports’;

constructor() {}

fn():string{return ‘aaa’;}

}

1. 接口间的继承

interface List {

name: string,

age: number,

status: boolean,

...

}

interface Son extends List{

fn(): string

}

1. **Typescript中的类**
2. super关键字

class Person {

content = "Hi，帅哥";

sayHello() {

return this.content;

}

}

class Son extends Person {

sayLove() {

return "I love you";

}

sayHello() {

return super.sayHello() + ‘,你好!’;

}

}

const p = new Son();

console.log(p.sayHello(), p.sayLove());

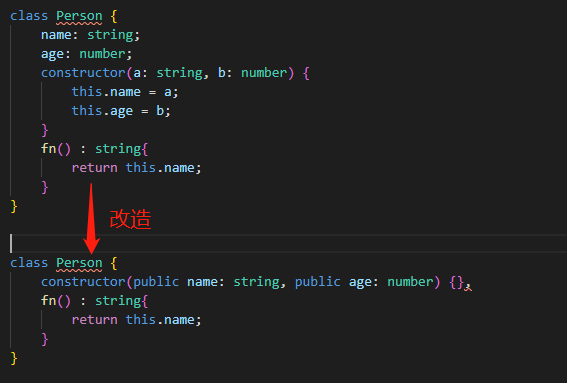
1. 类的访问类型

① public：默认类型（类的内部和外部都可调用）

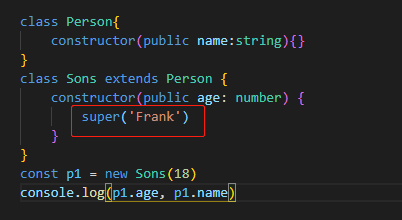
② protected: 受保护类型（类内部和继承的子类内部可调用）

③ private: 私有属性（只有类内部可调用）

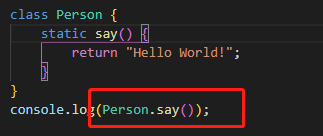
3. 类的构造函数constructor



4. 类继承的构造器



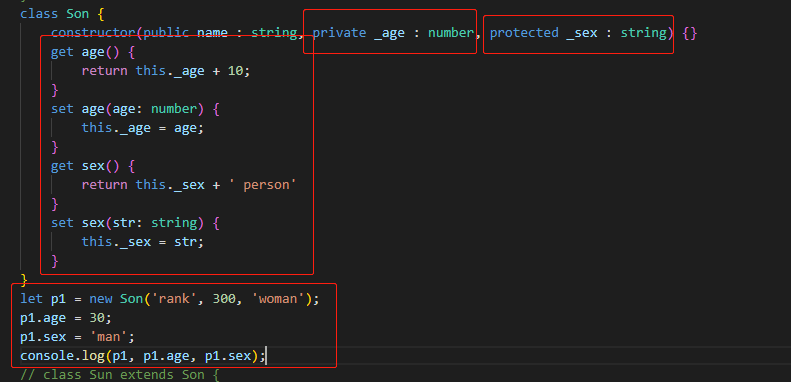
5. static定义静态属性



1. readOnly只读属性

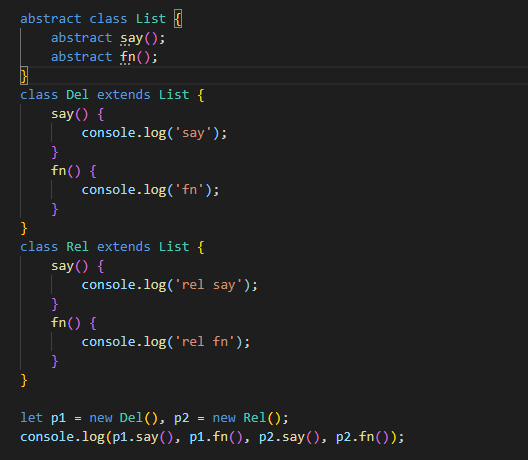


7. 类的Getter 和 Setter（类中private、protected属性，外部调用和修改）



8. 抽象类abstract

定义：简单理解，抽象类 - 类似于父类，需要继承，定义抽象类会伴随抽象方法，而继承抽象类的子类必须实现这些抽象方法



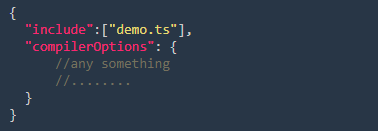
1. **配置文件 - tsconfig.json（只执行tsc）**
2. 初始化生成tsconfig.json文件

① 全局安装typescript：yarn global add typescript

② 初始化配置文件：tsc -init

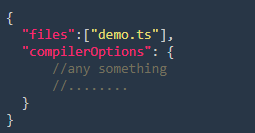
2. 编译范围配置项

① include 要编译的文件



② exclude 不编译的文件

③ files 编译的文件集（类似include）

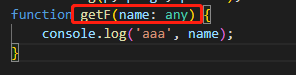


3. compilerOptions配置项

① removeComments 编译过滤掉注释（编译后不带注释）

② strict 严格模式

③ noImplicitAny any类型是否必须添加类型注解any



④ strictNullChecks 是否强制检查null类型（true是，false否）

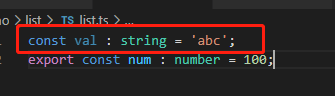
⑤ rootDir入口文件夹

⑥ outDir出口文件夹

⑦ allowJs可编译.js文件

⑧ sourceMap生成信息对照

⑨ noUnusedLocals 是否有定义了未使用的变量等



⑩ noUnusedParameters 是否有定义了未使用的参数等

1. **联合类型和类型保护**

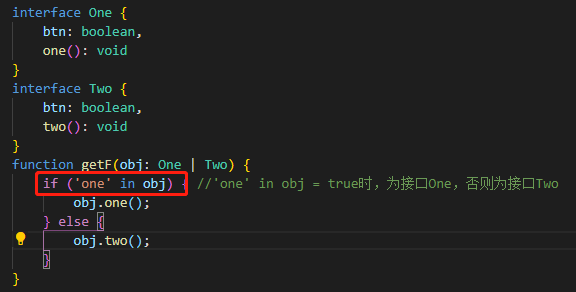
联合类型定义：简单理解，一个变量可能有两种或两种以上的类型

let value : (number | string | boolean) = 100 | ‘aaa’| true

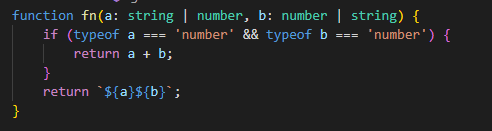
类型保护：简单理解，联合类型才需要类型保护，类型保护的方案很多，下面是几种常用的方式：

1. 类型保护 - 类型断言

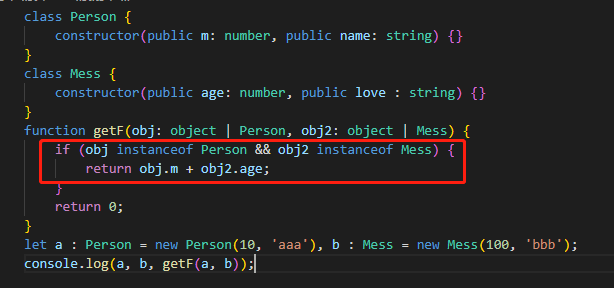
 2. 类型保护 - in语法



3. 类型保护 - typeof语法

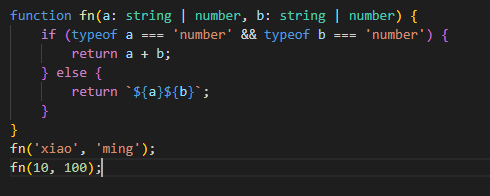


4. 类型保护 - instanceof语法

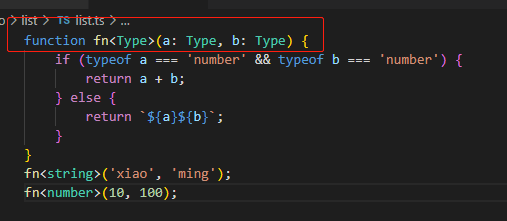


1. **TypeScript函数泛型（难点）**

泛型定义：简单理解，就是泛指的类型。



思考题：如何限定fn函数调用时，两个参数都为字符串或都为数字，即两个参数类型相同？



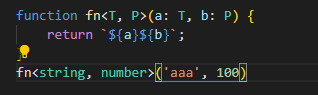
泛型的定义使用<>包裹，包裹里面的值即泛型的名字；

1. 泛型中的数组

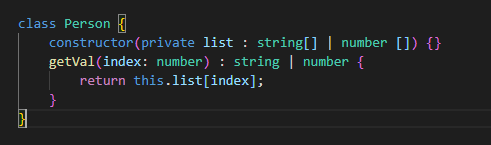
function fn<item>(arr: item[]){}

fn<string>([‘a’, ‘b’, ...])、fn<object>([{‘a’: 1, ‘b’: 2}, {...}])

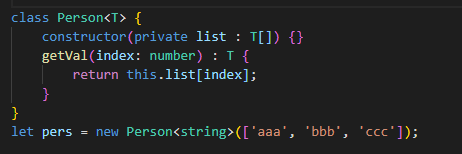
1. 多个泛型定义



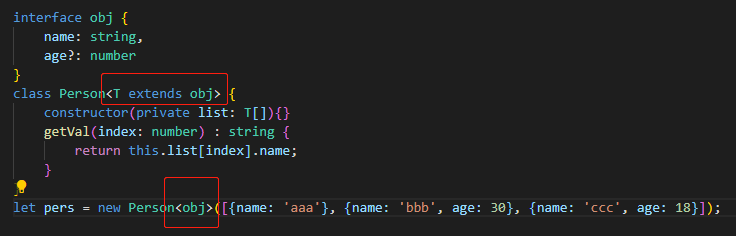
1. **TypeScript类中泛型（难点）**
2. 类中联合类型



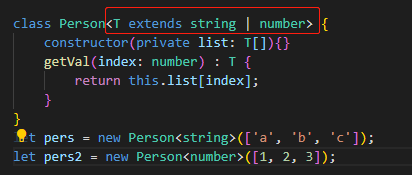
1. 类中泛型



1. 类中泛型的继承

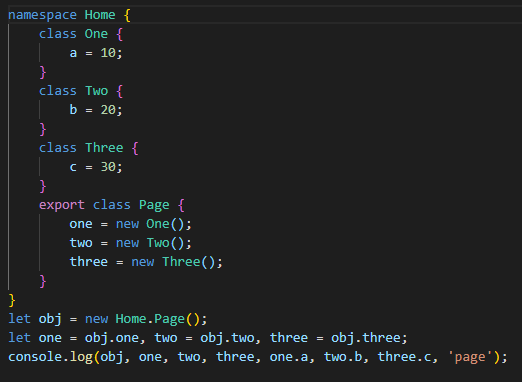


1. 泛型约束

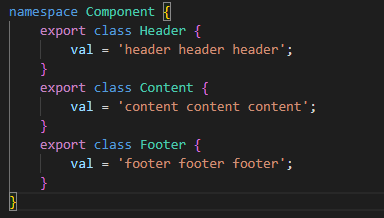


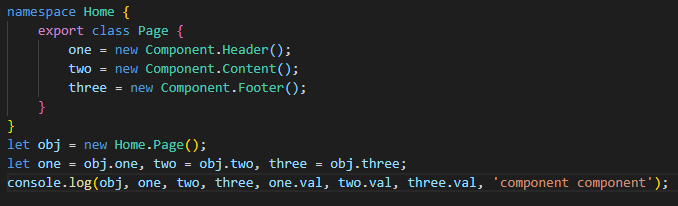
1. **TypeScript的命名空间（Namespace）**

命名空间定义：声明的关键字是namespace，要暴露的类关键词export，只有暴露的类才是全局的，避免其它的类生成全局污染了。

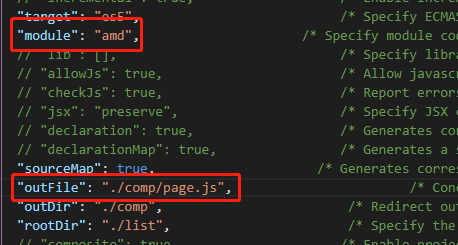


1. 命名空间实现组件化





1. ts多文件编译成一个文件



tsconfig.json配置如下：

module模式为：amd，outFile：编译后文件的信息

1. 子命名空间定义：



1. **Parcel打包TypeScript代码**

parcel一个开包即用的打包工具，类似于webpack，感兴趣自己学习下。

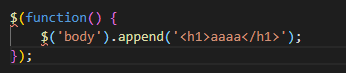
安装：yarn add parcel@next -D

1. **Typescript中引用第三方工具（例如：JQuery）**

以CDN的方式引入jquery文件

index.html引入https://cdn.bootcdn.net/ajax/libs/jquery/3.5.1/jquery.js

.ts文件中使用jquery



如图所示，提示：找不到名称’$’，解决方案如下：

1. 安装types/jquery

npm i @types/jquery

1. declare var $: any;
2. 自己写一个.d.ts声明文件的类库（自行学习配置吧）