一、搭建 Typescript 环境

刘承杰

TypeScript安装过程

建议采用局部安装，即只安装到项目目录下，不建议全局安装，尤其是在用nvm管理nodejs版本的情况下更不建议了。

MacOS、Linux

所有操作都是在项目目录下进行的

```
sudo npm install typescript -D
```

```
sudo npm install ts-node -D
```

如果要直接运行.ts文件的话，输入

```
npx ts-node 文件名.ts
```

如果要编译为js文件的话，输入

```
npx tsc 文件名.ts
```

Windows

所有操作都是在项目目录下进行的

以管理员模式运行cmd，定位到项目目录

```
npm install typescript -D
```

```
npm install ts-node -D
```

如果要直接运行.ts文件的话，输入

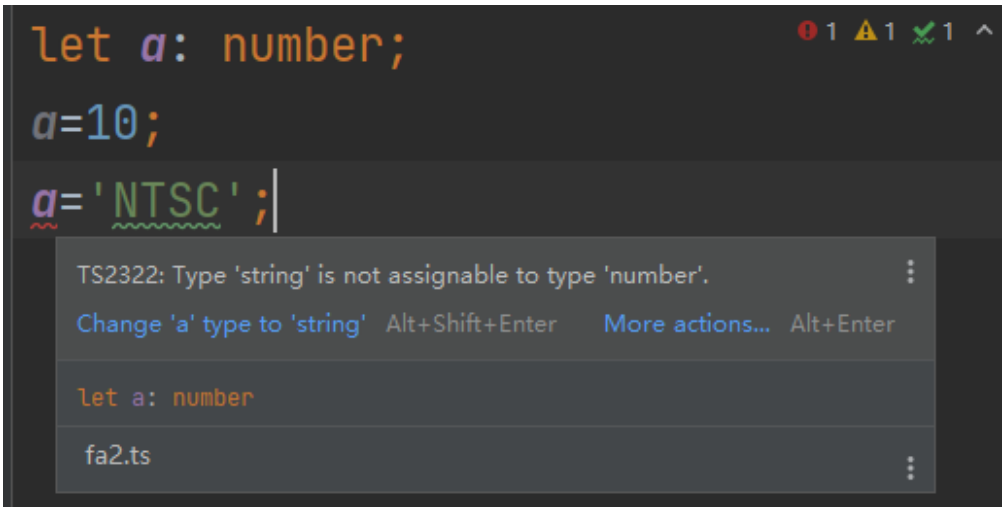
```
npx ts-node 文件名.ts
```

如果要编译为js文件的话，输入

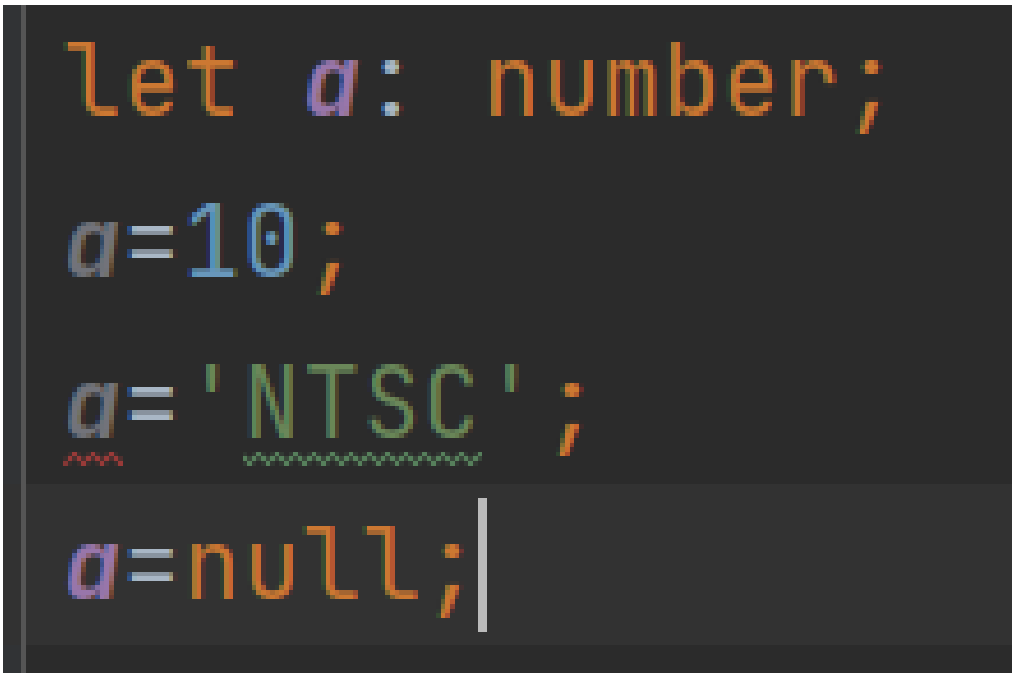
```
npx tsc 文件名.ts
```

二、基本类型

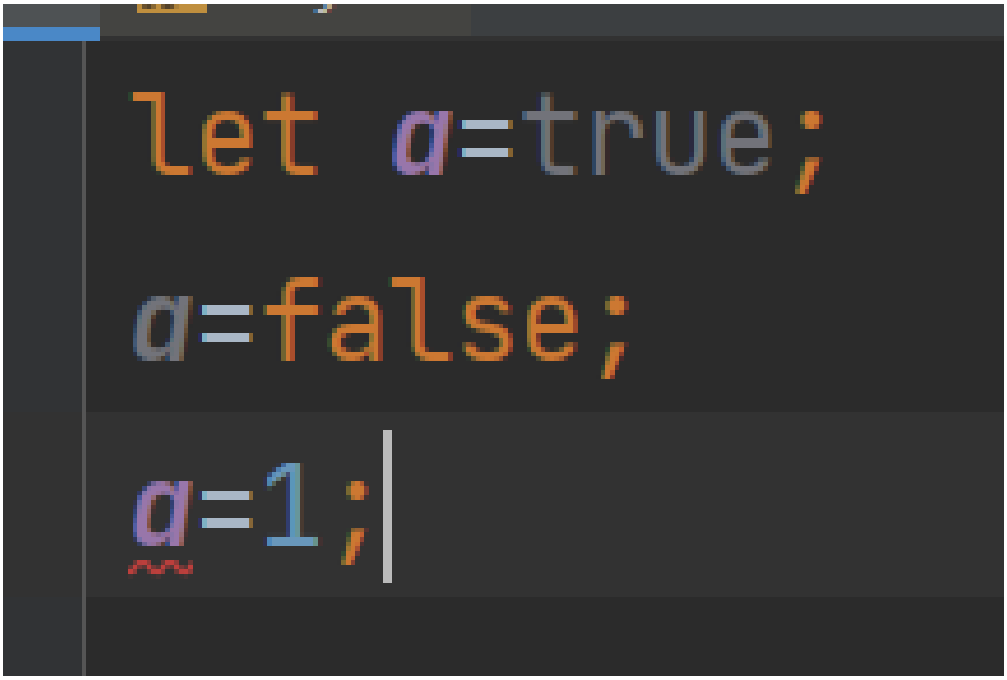
TS 相对 JS 最大的特点是引入类型的概念 而不是动态类型 即可以声明一个确定类型的变量



但是其可以为 null

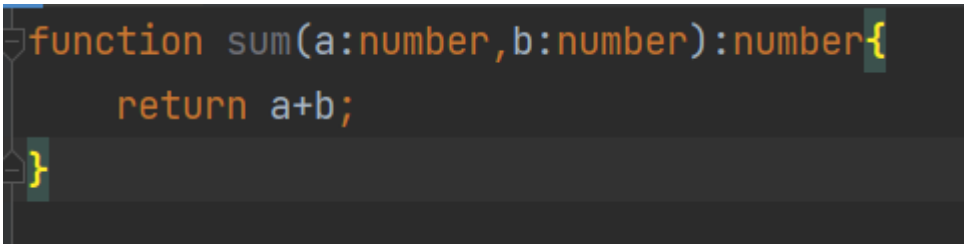


变量声明同时赋值可以同时进行类型检测



ts 可以在创建函数时制定类型 相当于作预先处理 但是不会在 JS 中体现

（ts 编译产生 js）



接下来是 typescript 的类型概述

//可以使用竖线来来凝结多个类型 成为联合类型 同理有&

Let a: boolean | string

//

1.数字类型 number

Let decimal :number=6;

Let hex :number=0xf00d;

Let binary :number=0b1010;

Let octal :number =0o744;

Let big :bigint=100n;

2.字符串类型 string

3.布尔类型 Boolean

4.任意类型 any

Let d:any;//对该变量关闭 ts 类型检测

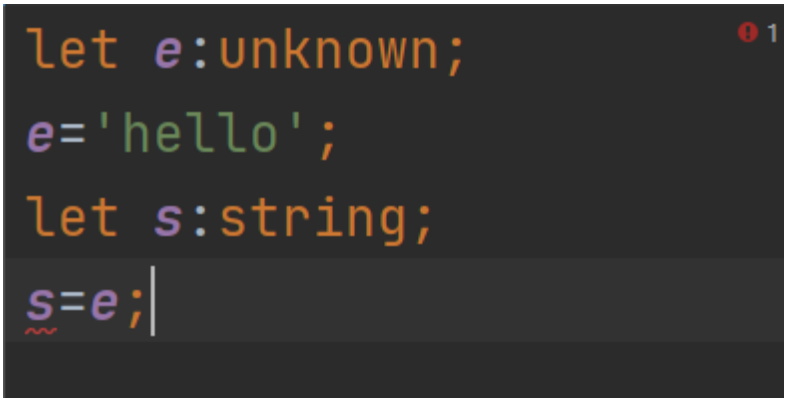
Let d;//隐式声明为 any

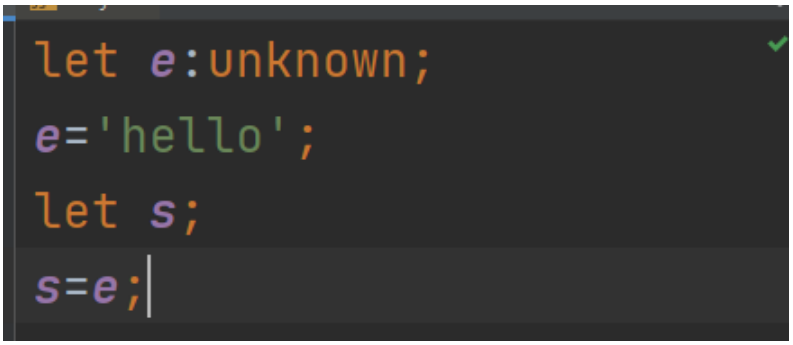
5.未知类型 unknown

any 和 unknown 的区别:

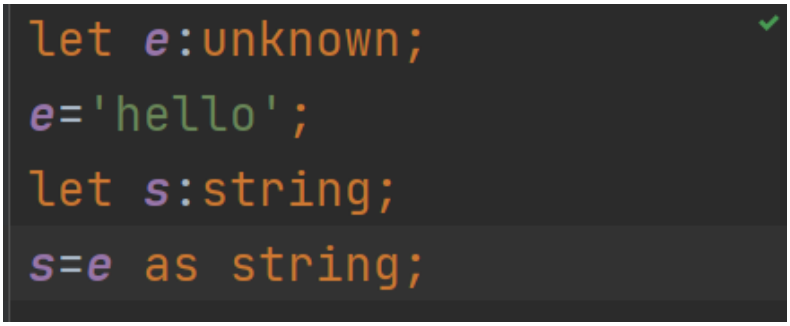
any 类型可以赋值给任意变量

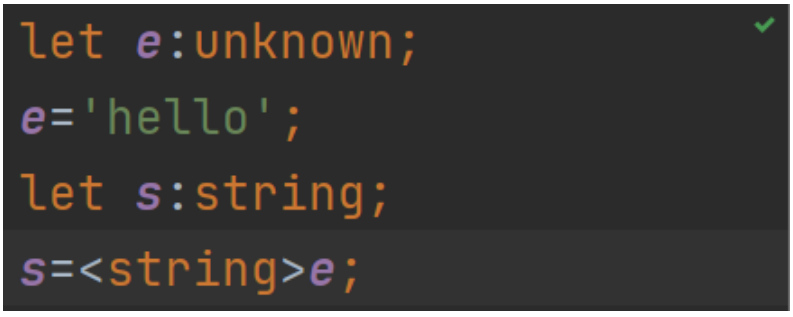
unknown 不能赋值给任意变量 即使类型本应该相等





类型断言 我说什么就是什么





6.void 返回值用 表示返回空

7.never 表示从来没有值返回 抛出异常时用

8.object 对象

```
let a:object;
a={};
a=function (){

};
a={name: string};

let b:{name:string};
b={};
b={name:"1"};
```

属性名后加？表示属性是可选属性，可有可无

```
let b:{name:string,age?:number};

b={name:"1"};
```

中括号形式表示对某一种类型的变量名不做个数要求

```
let c:{name:string,[propname:string]:any};
c={name:'lxx',a:1,b:"2",c:{c_1:"a",c_2:{}}};
```

还可以设置函数结构的类型声明 不过语法略有不同

```
//设置函数结构类型声明
let d:(a:number,b:number)=>number;
d=function (n1:number,n2:number):number{
    return n1+n2;
}
```

9.array 数组 10.元组 11.枚举

```
//数组
let e : string[];
e=['a','b','c'];
let f : number[];
let f1 : Array<number>;
let g : object[];
//元组
let h:[string,string];
h=['1','2'];
//枚举
enum gender{
    Male = 0,
    Female = 1,
}
let i:{name:string,gen:gender};
i={
    name:'lxx',
    gen:gender.Male,
}
```

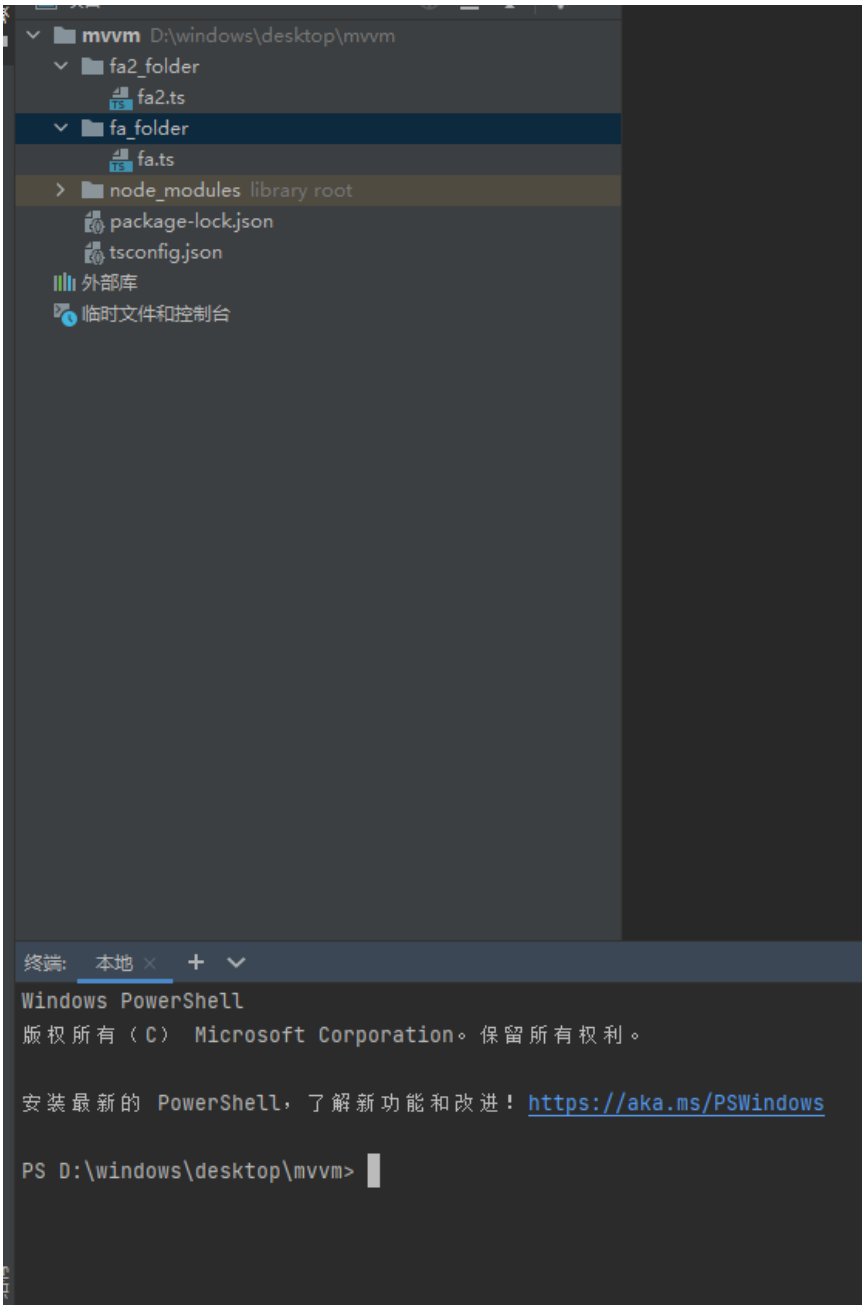
12.类型别名

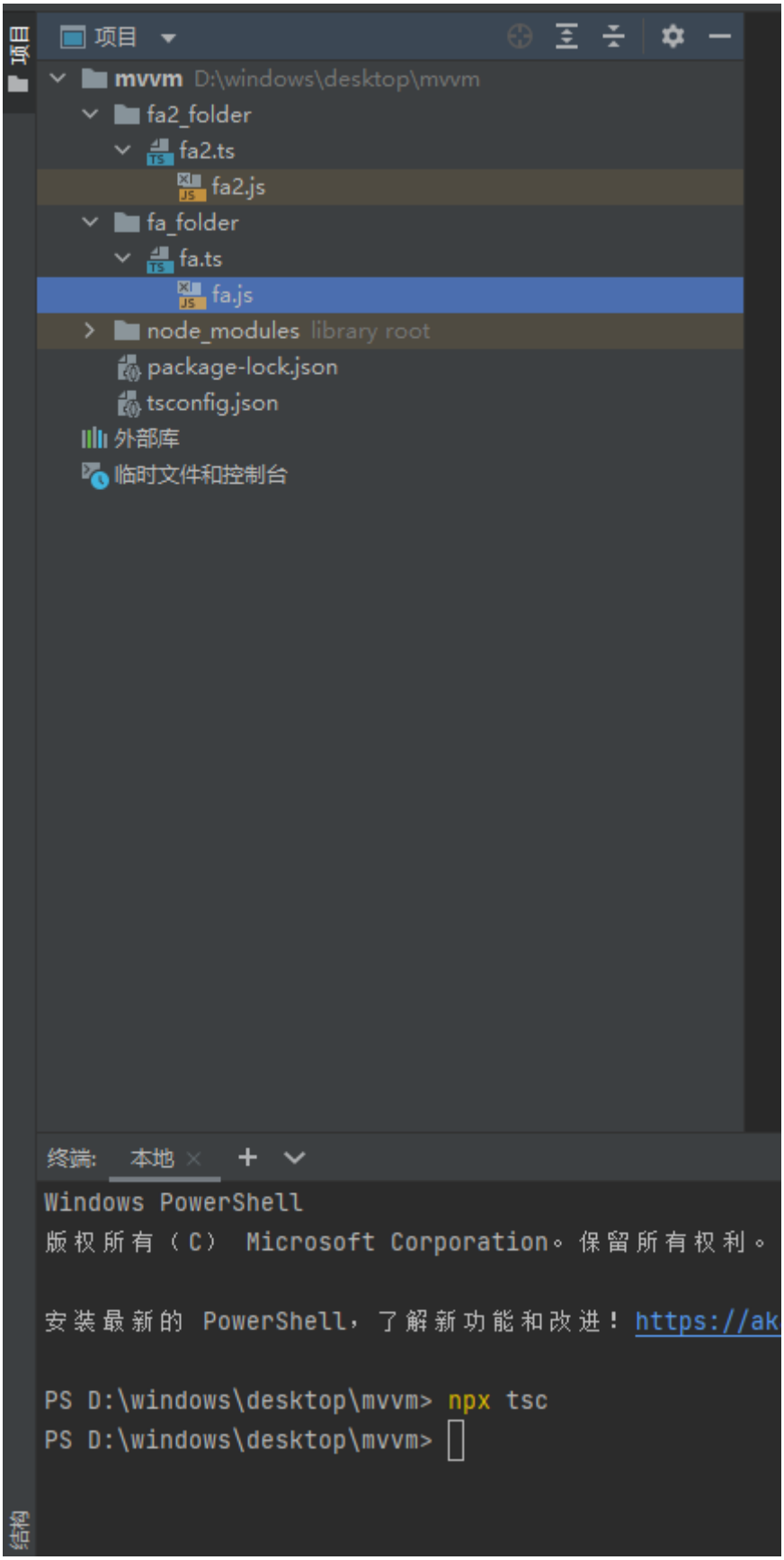
```
type mytype = 1|2|3|4|5;
let a:mytype=2;
let b:mytype;
```

三、如何自动编译

1.tsc 文件名 -w 在改变时自动监视 需要让控制台开着

2.tsc 可以直接将所有文件编译 但是不能直接执行，需要添加 tsconfig.json 文件





3.所以在有 config 的情况下可以 tsc -w 达到 global watch 效果

四、编译选项

1.include&exclude

- include
  - 定义希望被编译文件所在的目录
  - 默认值：["**/*"]
  - 示例：
    - ```
      "include":["src/**/*", "tests/**/*"]
      ```
    - 上述示例中，所有src目录和tests目录下的文件都会被编译
- exclude
  - 定义需要排除在外的目录
  - 默认值：["node_modules", "bower_components", "jspm_packages"]
  - 示例：
    - ```
      "exclude": ["./src/hello/**/*"]
      ```
    - 上述示例中，src下hello目录下的文件都不会被编译

2.extends&files

- extends
  - 定义被继承的配置文件
  - 示例：
    - ```
      "extends": "./configs/base"
      ```
    - 上述示例中，当前配置文件中会自动包含config目录下base.json中的所有配置信息
- files
  - 指定被编译文件的列表，只有需要编译的文件少时才会用到
  - 示例：
    - ```
      "files": [
          "core.ts",
          "sys.ts",
          "types.ts",
          "scanner.ts",
          "parser.ts",
          "utilities.ts",
          "binder.ts",
          "checker.ts",
          "tsc.ts"
        ]
      ```

3.compilerOptions

3.1 target 用来指定 ts 被编译成 es 的版本

"target":"ES6"

3.2 module 指定要使用的模块化规范

"module":"ES2015"

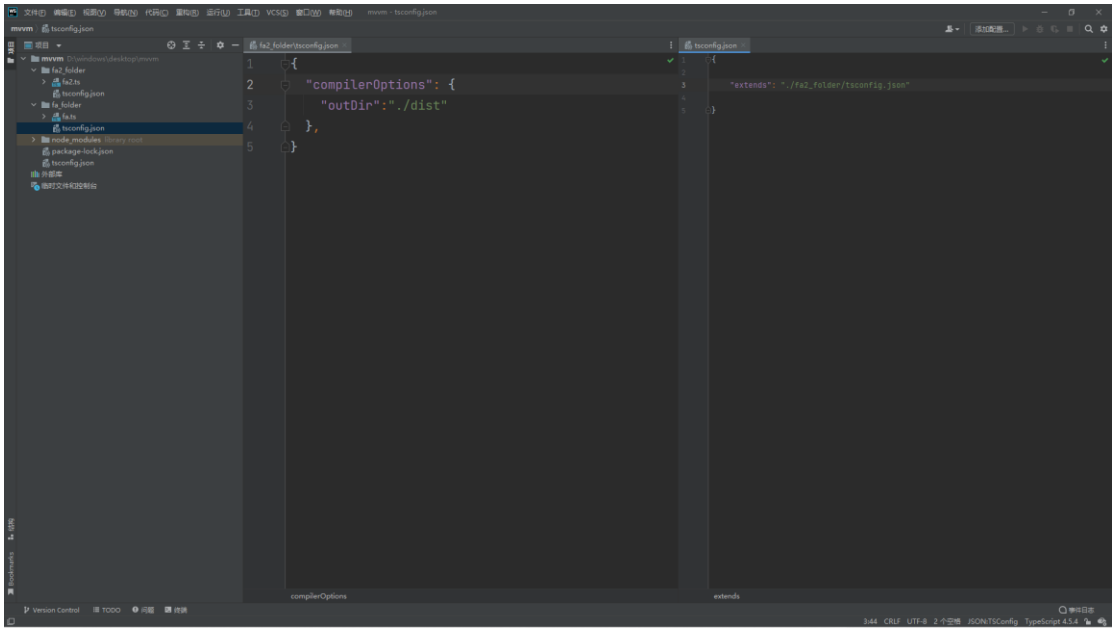
3.3 lib 指定项目中要使用的库 不建议修改

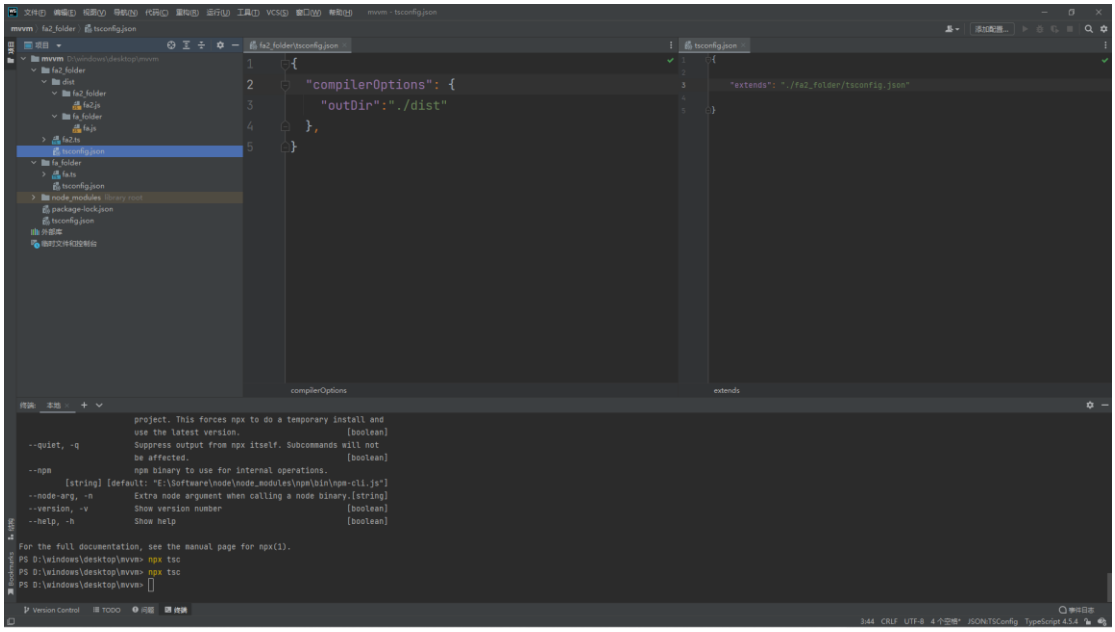
3.4 outdir 指定编译后文件和编译前文件的相对目录

//思路 假如有多组文件夹有 ts 文件，可以采用 extends 方法 继承不同文件夹的

配置文件 来让每一个文件夹中的 ts 和 js 分开存放 这是我们模块化开发的保障

仅供参考





3.5outFile 将作用域中所有的合成一个文件

3.6allowJs 是否编译 JS 文件

3.7checkJs 检查 JS 语法是否符合 ts 规范

3.8removeComments 是否移除注释

3.9noEmit 是否生成编译后文件

3.10noEmitOnError 有错误时是否生成编译后文件

3.11alwaysStrict 用来设置编译之后的文件是否使用严格模式

3.12noImplicitAny 是否允许隐式产生的 any 类型

3.13noImplicitThis 是否允许不明确类型的 this

3.14stritNullChecks 严格的检查空值

3.15strict 所有严格检查的总开关

五、大作业就不用 webpack 打包代码了 应该没必要

六、js ts 的面向对象和类

1.操作浏览器使用 window 对象 操作网页使用 document 对象 操作控制台使用 console 对象

2.ts 中类的简介 其实也和 js 差不多 类中实例属性和静态属性与其它高级语言几乎一致 readonly 开头的属性称为只读属性 不能修改

```
Class 类名 {
    属性名：类型
    Constructor(参数：类型){
        This.属性名=参数
    }
    方法名（）{
        ……
    }
```

3.构造函数 this 继承 super 关键字 抽象类 接口 略 因为真的和 js java 一模一样

4.泛型 同 C++模板 JAVA 泛型