

【TypeScript 4.5】002-第 2 章 TypeScript 入门

【TypeScript 4.5】002-第 2 章 TypeScript 入门

一、发现问题

- 1、字符串
- 2、函数
- 3、JavaScript 局限

二、解决问题：静态类型检查

- 1、在代码运行之前发现错误
- 2、静态类型检查
 - 代码示例
 - 类型检查提示

三、非异常故障

- 1、JavaScript 示例
 - 代码示例
 - 运行结果
- 2、TypeScript 示例
 - 代码示例
 - 检查结果
- 3、TypeScript 错别字提示
 - 代码示例
 - 检查结果
- 4、TypeScript 未调用函数提示
 - 代码示例
 - 检查结果
- 5、TypeScript 逻辑错误提示
 - 代码示例
 - 检查结果

四、实用工具

- 1、环境安装
 - 安装 VSCode
 - 安装 Node.js
 - 安装 TypeScript 编译器
- 2、编译 ts 文件
 - 示例代码 hello.ts
 - 编译 ts 文件
 - 编译结果
 - 运行该 js 文件
- 3、写点 ts 代码看看
 - 代码示例 hello.ts
 - 再次编译结果 hello.js
 - 发现的问题

五、优化编译，解决问题

- 1、解决 TS 与 JS 冲突问题
 - 命令
 - 演示
 - 关于参数报错问题
 - 最终结果
- 2、自动编译
 - 命令
 - 演示
- 3、发出错误

- 命令
- 制造一个错误代码
- 报错
- 编译结果
- 运行 js 代码
- 执行命令
- 比较

六、显式类型

- 1、开启严格模式
 - 修改配置文件
 - 报错了
- 2、显式类型
 - 定义显式类型
- 3、类型推断
 - 代码示例
 - 报错

七、降级编译

- 1、发现问题
 - ES6语法
 - 浏览器兼容问题
- 2、降级编译
 - 修改配置文件
 - 编译结果

八、严格模式

- 1、相关配置内容
- 2、严格模式
 - 严格模式下报错演示
 - 关闭严格模式
 - 问题
- 3、noImplicitAny
 - 含义
 - 是否开启
- 4、strictNullChecks
 - 含义
 - 没开启的情况演示
 - 开启 strictNullChecks 之后
 - 当前配置

一、发现问题

1、字符串

```
const message = "Hello world"
message.toLowerCase(); // 得到"hello world"
message(); // 报错: TypeError: message is not a function
```

2、函数

```
function fn(x) {  
    return x.flip();  
}
```

// 可知，要成功执行上面函数内容代码是要满足一定条件的，那就是 **x** 必须能够调用 **flip()** 方法；

3、JavaScript 局限

JavaScript 只能在**运行时**发现错误！

二、解决问题：静态类型检查

1、在代码运行之前发现错误

像 TypeScript 等类型检查工具，可以做到**在代码运行之前发现错误**！

2、静态类型检查

代码示例

```
const message = "hello world!"  
message()
```

类型检查提示



```
TS index.ts 1 ●  
src > TS index.ts > ...  
1  const message = "hello world!"  
2  message()  
   const message: "hello world!"  
   此表达式不可调用。  
   类型 "String" 没有调用签名。 ts(2349)  
   查看问题 没有可用的快速修复
```

三、非异常故障

1、JavaScript 示例

代码示例

```
const user = {  
  name: "大哥",  
  age: 25  
}  
console.log(user.location)
```

运行结果

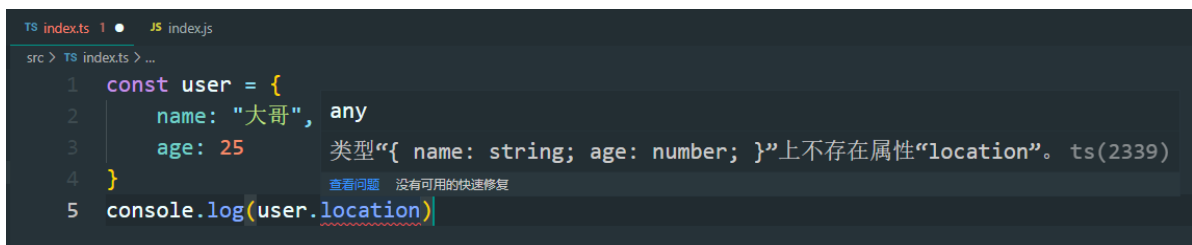


2、TypeScript 示例

代码示例

```
const user = {  
  name: "大哥",  
  age: 25  
}  
console.log(user.location)
```

检查结果

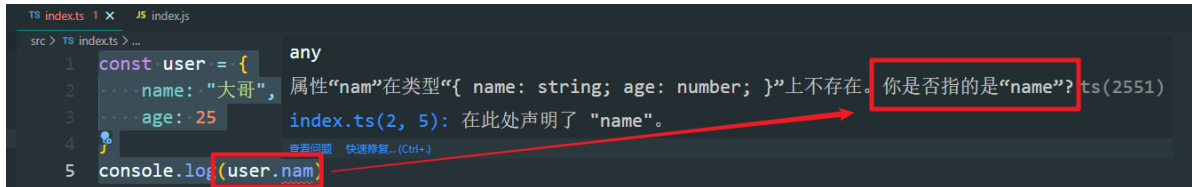


3、TypeScript 错别字提示

代码示例

```
const user = {  
  name: "大哥",  
  age: 25  
}  
console.log(user.nam)
```

检查结果

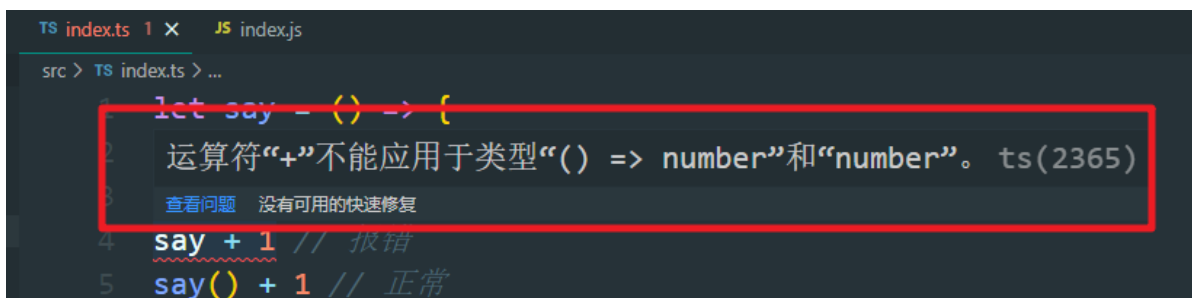


4、TypeScript 未调用函数提示

代码示例

```
let say = () => {  
  return 1  
}  
say + 1 // 报错  
say() + 1 // 正常
```

检查结果



5、TypeScript 逻辑错误提示

代码示例

```
const value = Math.random() < 100 ? 'a' : 'b'  
if (value !== 'a') {  
  
} else if (value === 'b') { // 报错!  
  
}
```

检查结果

```
TS index.ts 1 x JS index.js
src > TS index.ts > ...
1  const value const value: "a"
2  if (value ! 此条件将始终返回 "false", 因为类型 ""a"" 和 ""b"" 没有重叠。ts(2367)
3                                     查看问题 没有可用的快速修复
4  } else if (value === 'b') {
5
6  }
```

四、实用工具

1、环境安装

这里省略了，网上有很多相关教程！

安装 VSCode

<https://www.visualstudio.com/Download>

安装 Node.js

<https://nodejs.org/en/download>

安装 TypeScript 编译器

```
npm install -g typescript
```

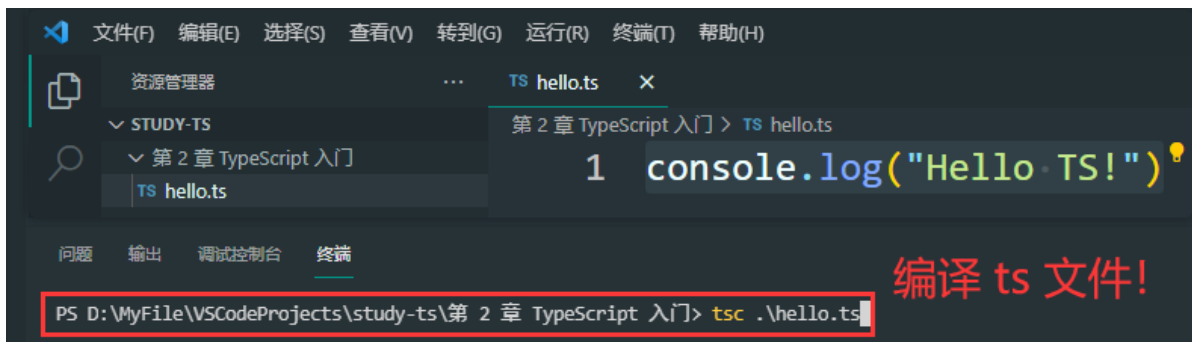
2、编译 ts 文件

示例代码 hello.ts

```
console.log("Hello TS!")
```

编译 ts 文件

```
tsc .\hello.ts
```

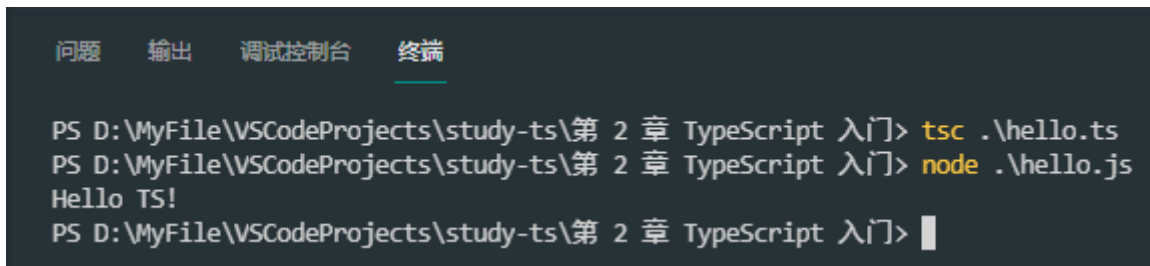


编译结果



运行该 js 文件

得到了预想的结果!



3、写点 ts 代码看看

代码示例 hello.ts

```
function say(name, age) {  
    console.log(`my name is ${name}, I am ${age} years old!`)  
}  
say("誉博", 25)
```

再次编译结果 hello.js

```
function say(name, age) {  
    console.log("my name is ".concat(name, ", I am ").concat(age, " years  
old!"));  
}  
say("皆博", 25);
```

发现的问题

- 修改 ts 代码之后每次都需要使用 tsc 手动编译;
- 提示错误：函数实现重复（TS 与 JS 冲突）；



五、优化编译，解决问题

1、解决 TS 与 JS 冲突问题

命令

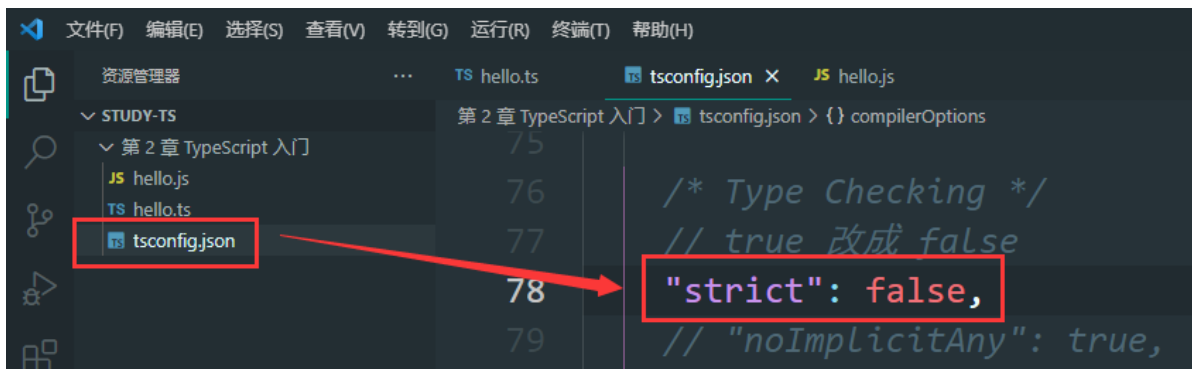
```
tsc --init # 生成配置文件
```

演示



关于参数报错问题

参数报错问题暂且放置，我们暂时先关闭严格模式！



最终结果



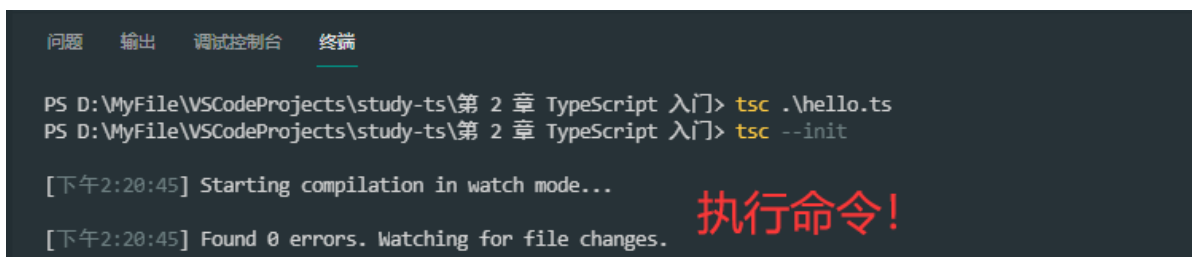
2、自动编译

命令

```
tsc --watch
```

演示

此时即实现了，当 ts 文件报错的时候，**自动编译**成对应的 js 文件，即 js 文件随着 ts 文件的保存而与 ts 文件**保持同步**！



3、发出错误

ts 报错的时候，可以正常编译成 js 文件，并可以运行！我们想当 ts 报错的时候不生成 js 文件，加上 `-noEmitOnError` 参数！

命令

```
tsc -noEmitOnError hello.ts
```

制造一个错误代码

```
function say(name, age) {  
    console.log(`my name is ${name}, I am ${age} years old!`)  
}  
say("誉博")
```

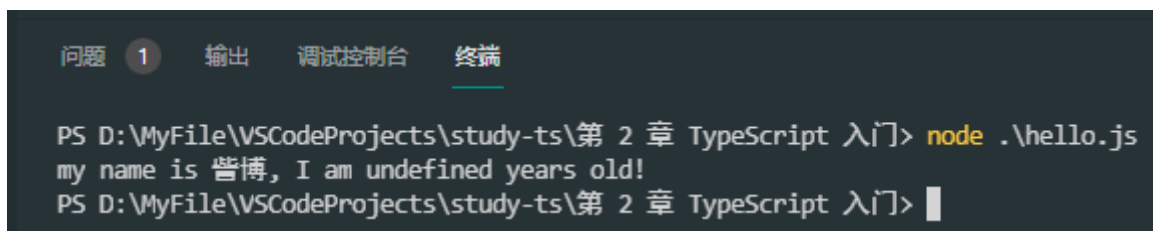
报错



编译结果

```
function say(name, age) {  
    console.log(`my name is ${name}, I am ${age} years old!`);  
}  
say("誉博");
```

运行 js 代码

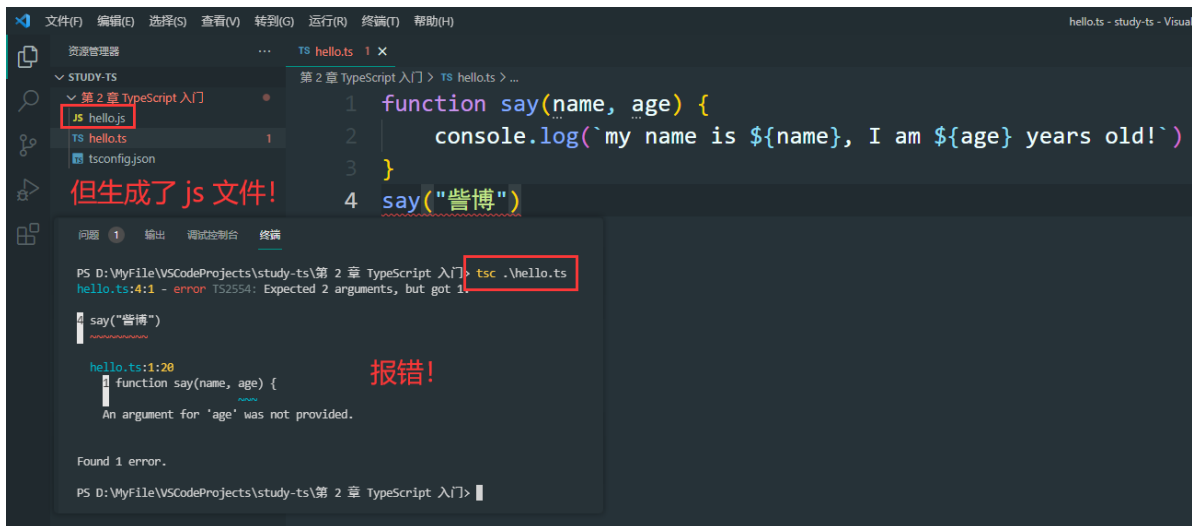


执行命令

使得 ts 报错时不生成 js 文件！（提前把之前编译的 js 文件删除！）



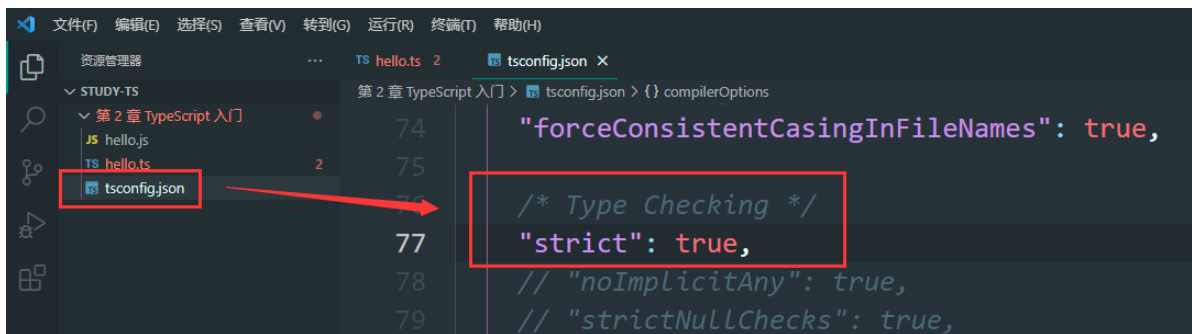
比较



六、显式类型

1、开启严格模式

修改配置文件

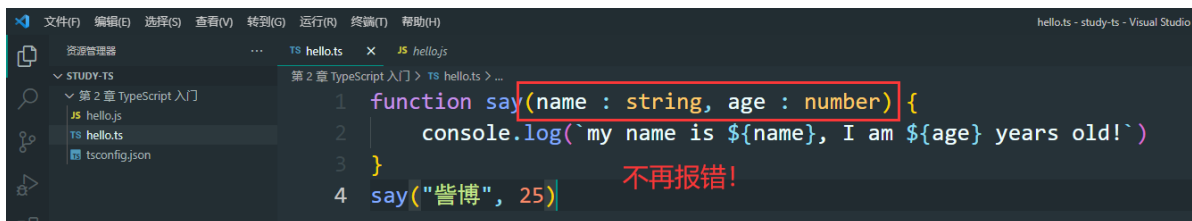


报错了



2、显式类型

定义显式类型



3、类型推断

并非一定要显式定义类型!

代码示例

```
// 类型推断
let message = "hello ts!"
message = "hello world!"
message = 100 // 报错: 不能将类型“number”分配给类型“string”。
```

报错



七、降级编译

1、发现问题

ES6语法



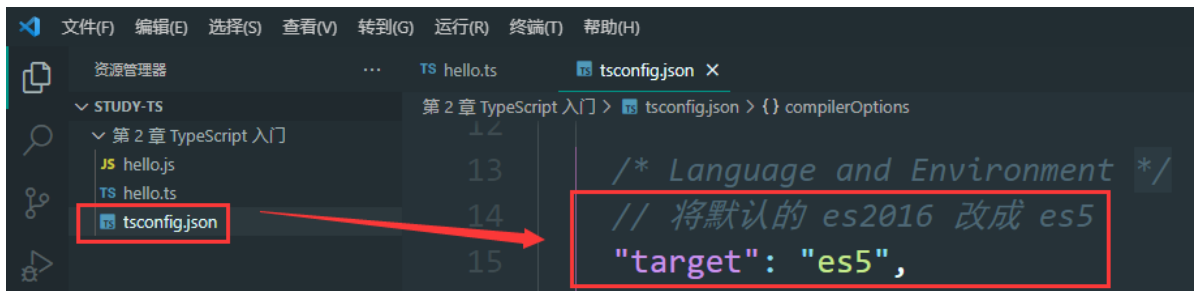
浏览器兼容问题

并非所有的浏览器都兼容 ES6 语法! 我们需要“降级编译”!

不过, 大多数浏览器都已经支持 ES6 了! ES2015 就是 ES6!

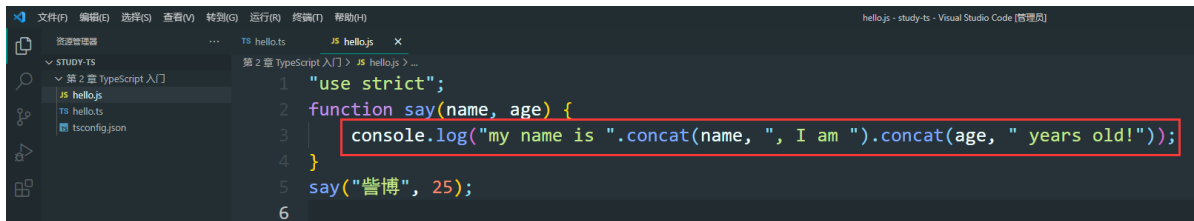
2、降级编译

修改配置文件



编译结果

测试完之后, 我把 target 恢复到了默认的 es2016, 即 ES7!



八、严格模式

1、相关配置内容

tsconfig.json 文件: 新开发的项目应该全部打开这些严格性检查!

```
/* Type Checking */
"strict": true, // 严格模式
"noImplicitAny": true, // 禁止隐式类型为 any
"strictNullChecks": true // 开启 null 和 undefined 的检查
```

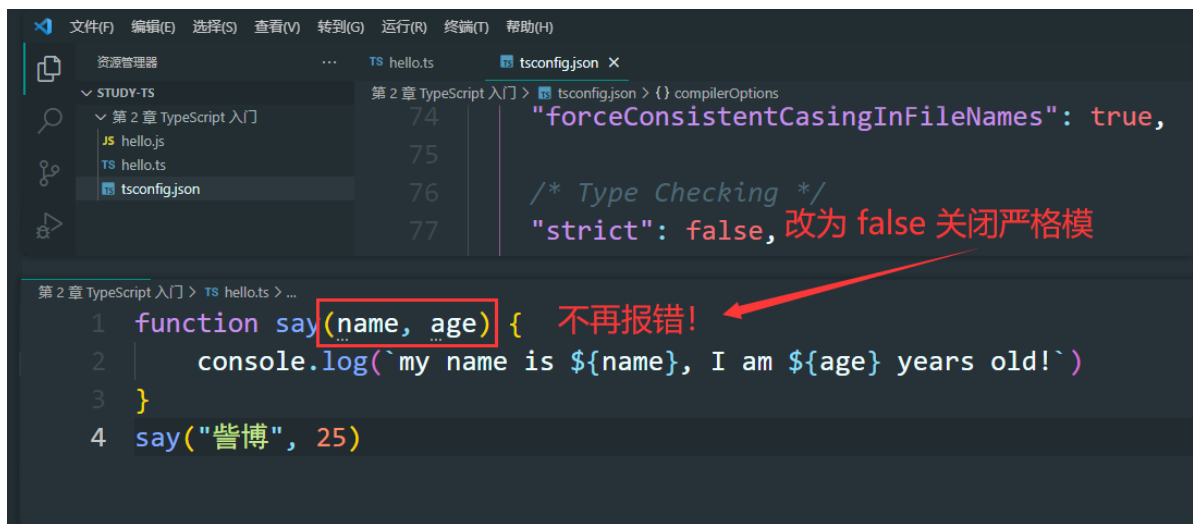
2、严格模式

严格模式下报错演示



关闭严格模式

做项目迁移时这么做! 一般不这么做!



问题

关闭严格模式, 类型隐式推断为 `any` 类型, 也就回到了普通的 `js` 代码的效果了, 与是否使用 `ts` 无区别! 我们可以 `strict` 严格模式, 我们呢也可以将 `noImplicitAny` 设置为 `true`。

3、noImplicitAny

参考文章: <https://segmentfault.com/a/1190000019768261>

含义

禁止隐式 `any` 类型!

是否开启

建议开启！开启之后如严格模式一样，没有声明类型的变量会报“.....隐式具有 any 类型”的错误！

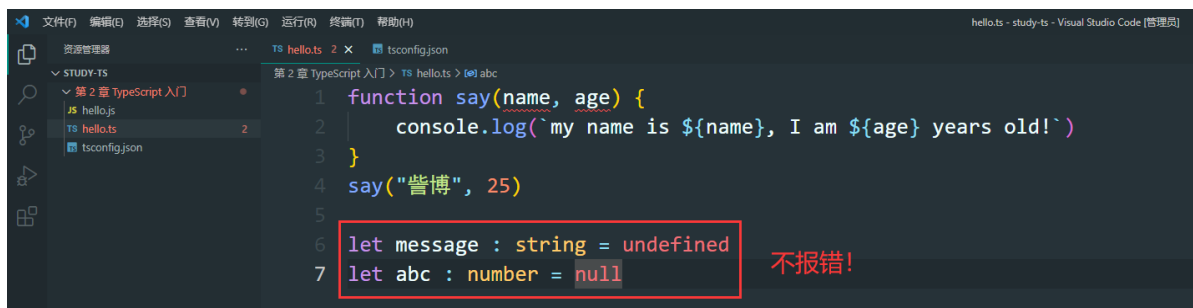
4、strictNullChecks

含义

是否开启 null 和 undefined 的检查！

没开启的情况演示

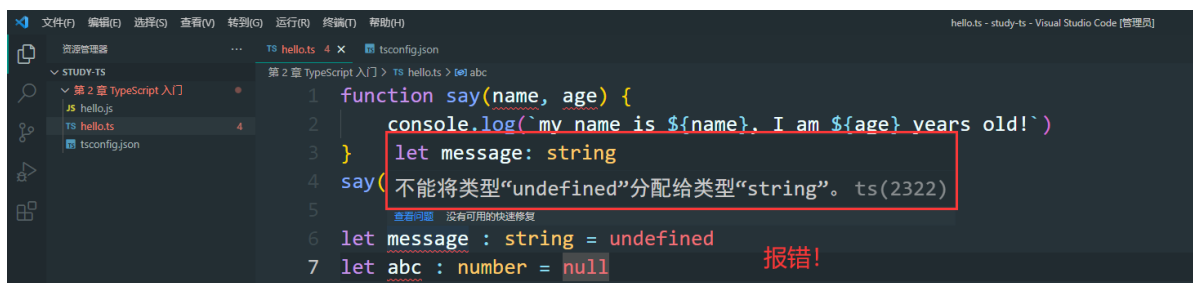
此时未开启严格模式！开启了 `noImplicitAny`！



```
1 function say(name, age) {
2   console.log(`my name is ${name}, I am ${age} years old!`)
3 }
4 say("誉博", 25)
5
6 let message : string = undefined
7 let abc : number = null
```

不报错！

开启 strictNullChecks 之后

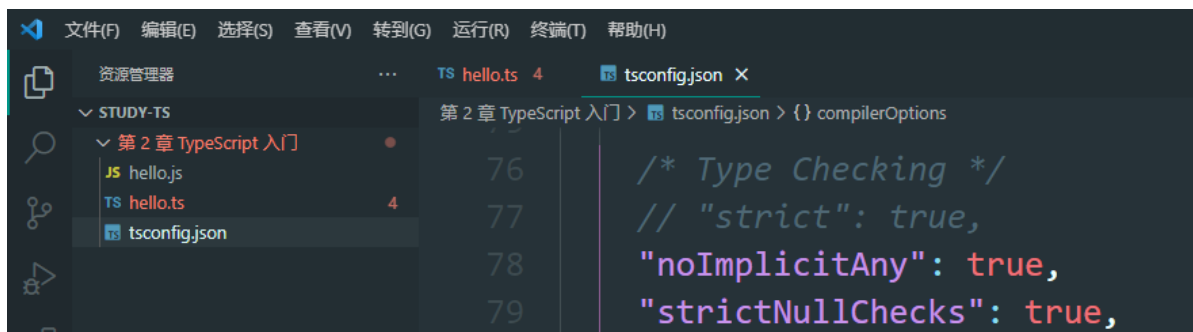


```
1 function say(name, age) {
2   console.log(`my name is ${name}, I am ${age} years old!`)
3 }
4 say("誉博", 25)
5
6 let message : string = undefined
7 let abc : number = null
```

报错！

当前配置

关闭下面两个，开启严格模式也能达到同样的效果！



```
76 /* Type Checking */
77 // "strict": true,
78 "noImplicitAny": true,
79 "strictNullChecks": true,
```

